Computing Techniques

Dr. Xavier Buffat

Beams Department – Accelerator and Beam Physics
Hadron Synchrotron Collective effects

CERN, Switzerland, Geneva

November 2018

Slides and sources at **http://cern.ch/xbuffat/2018-CAS-ComputingTechniques/**

Scope

- After this lecture you may :
  - Know the basic principles of common computer architectures
    → Determine the type of computer architecture that best fits your problem
  - Start writing code that uses efficiently your computer
    → Know the jargon to make your way through the doc on the web...

Content

- The Turing machine
- The Central Processing Unit (CPU)
  - Intrinsics, vectorisation
- Multiple CPUs
  - Multithreading
  - NUMA zones
  - Hyperthreading
- Amdhal's law and Moore's Law
- The Graphics Processing Unit (GPU)
- Computer clusters
  - The Message Passing Interface
- Volunteer and grid computing
- Summary

Content

I : Everyday computing

- The Turing machine
- The Central Processing Unit (CPU)
  - Intrinsics, vectorisation
- Multiple CPUs
  - Multithreading
  - NUMA zones
  - Hyperthreading
- Amdhal's law and Moore's Law
- The Graphics Processing Unit (GPU)
- Computer clusters
  - The Message Passing Interface
- Volunteer and grid computing
- Summary

Content

I : Everyday computing

- The Turing machine
- The Central Processing Unit (CPU)
  - Intrinsics, vectorisation
- Multiple CPUs
  - Multithreading
  - NUMA zones
  - Hyperthreading
- Amdhal's law and Moore's Law

II : High performance computing

- The Graphics Processing Unit (GPU)
- Computer clusters
  - The Message Passing Interface
- Volunteer and grid computing
- Summary

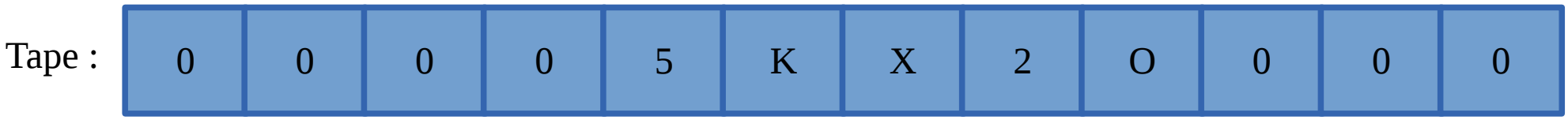
What is a computer ?

The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**

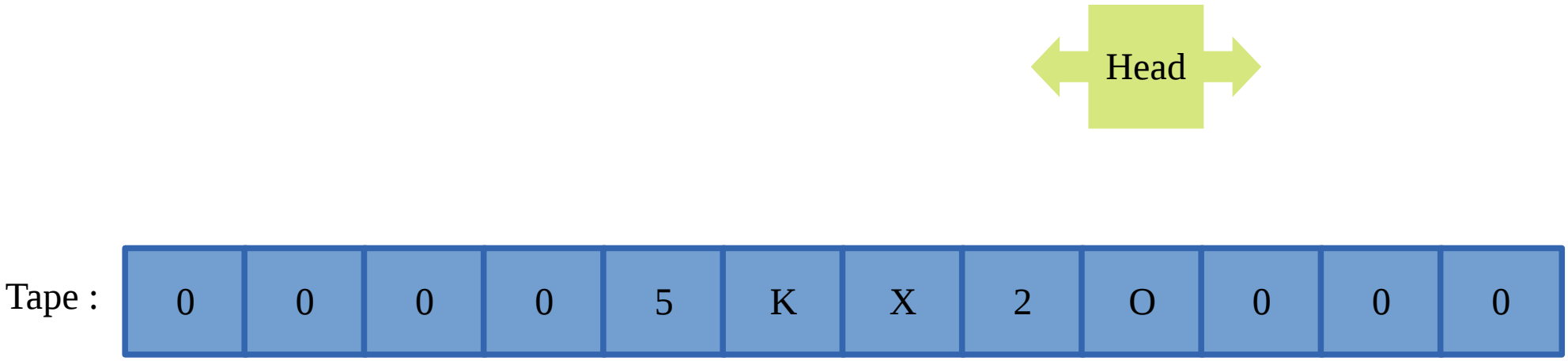
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



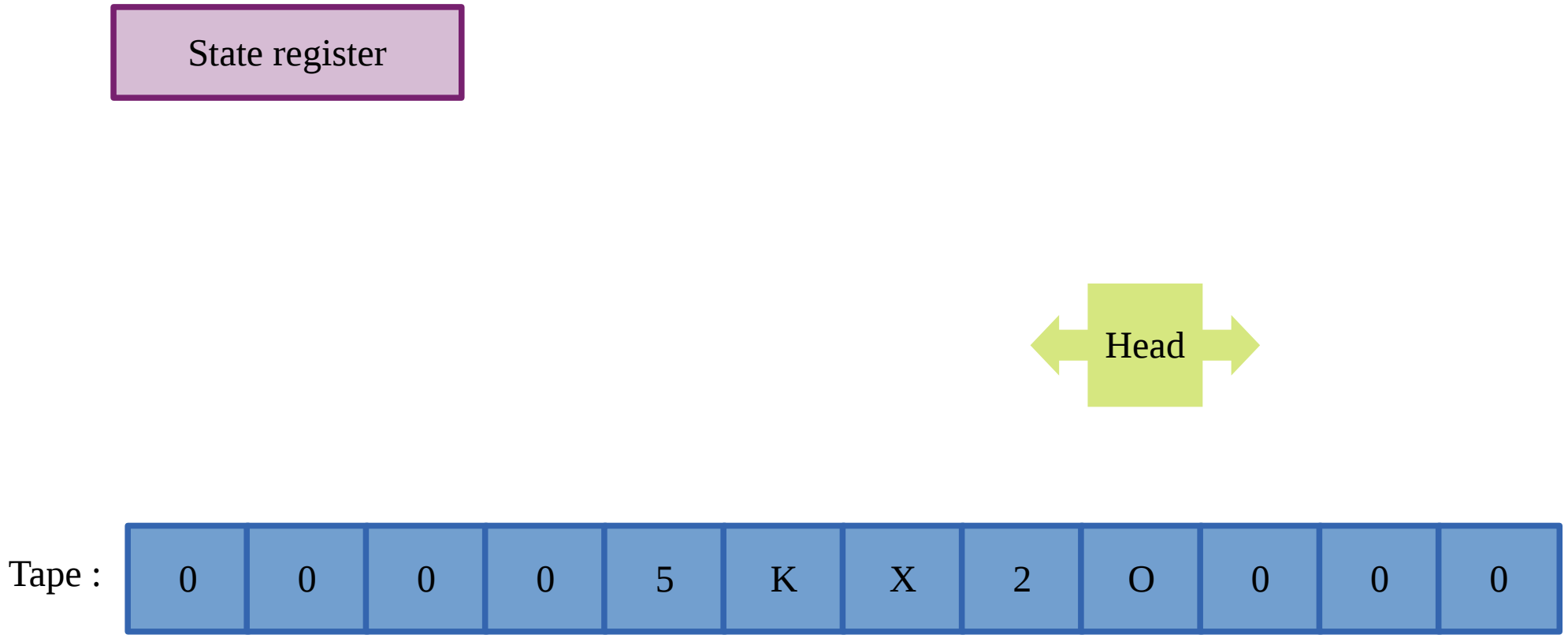
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



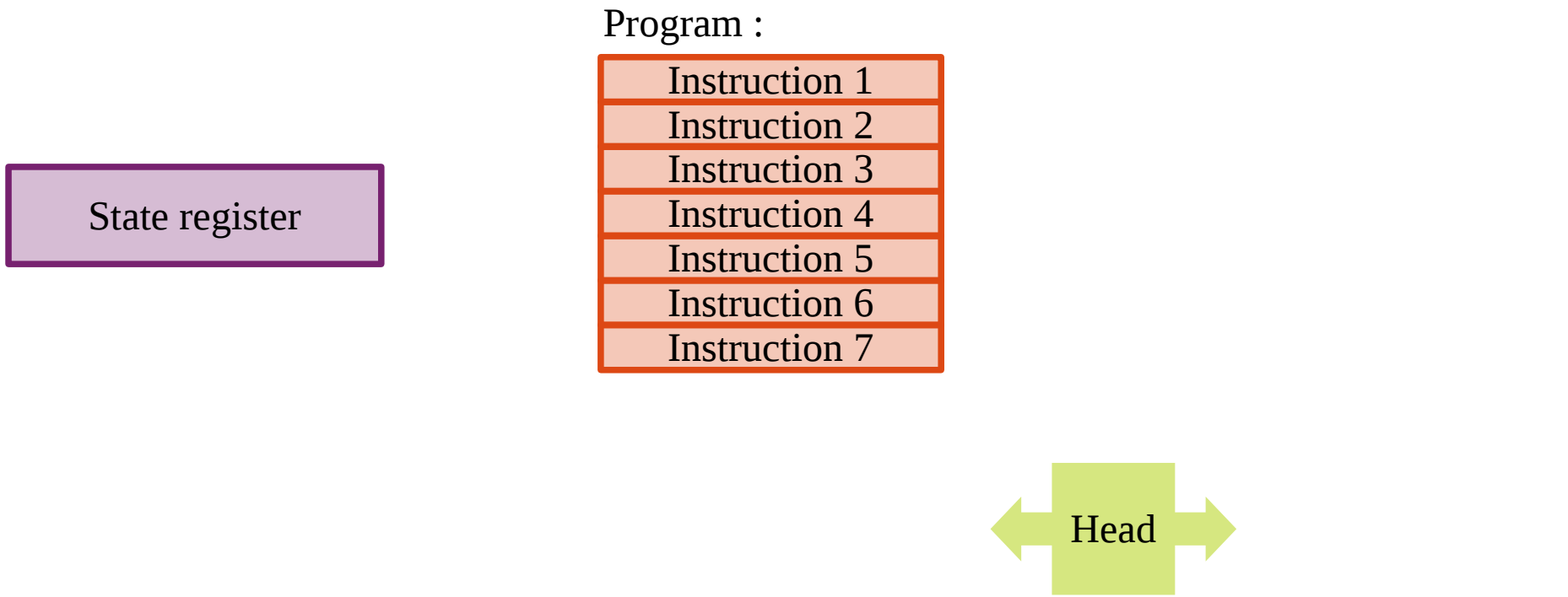
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



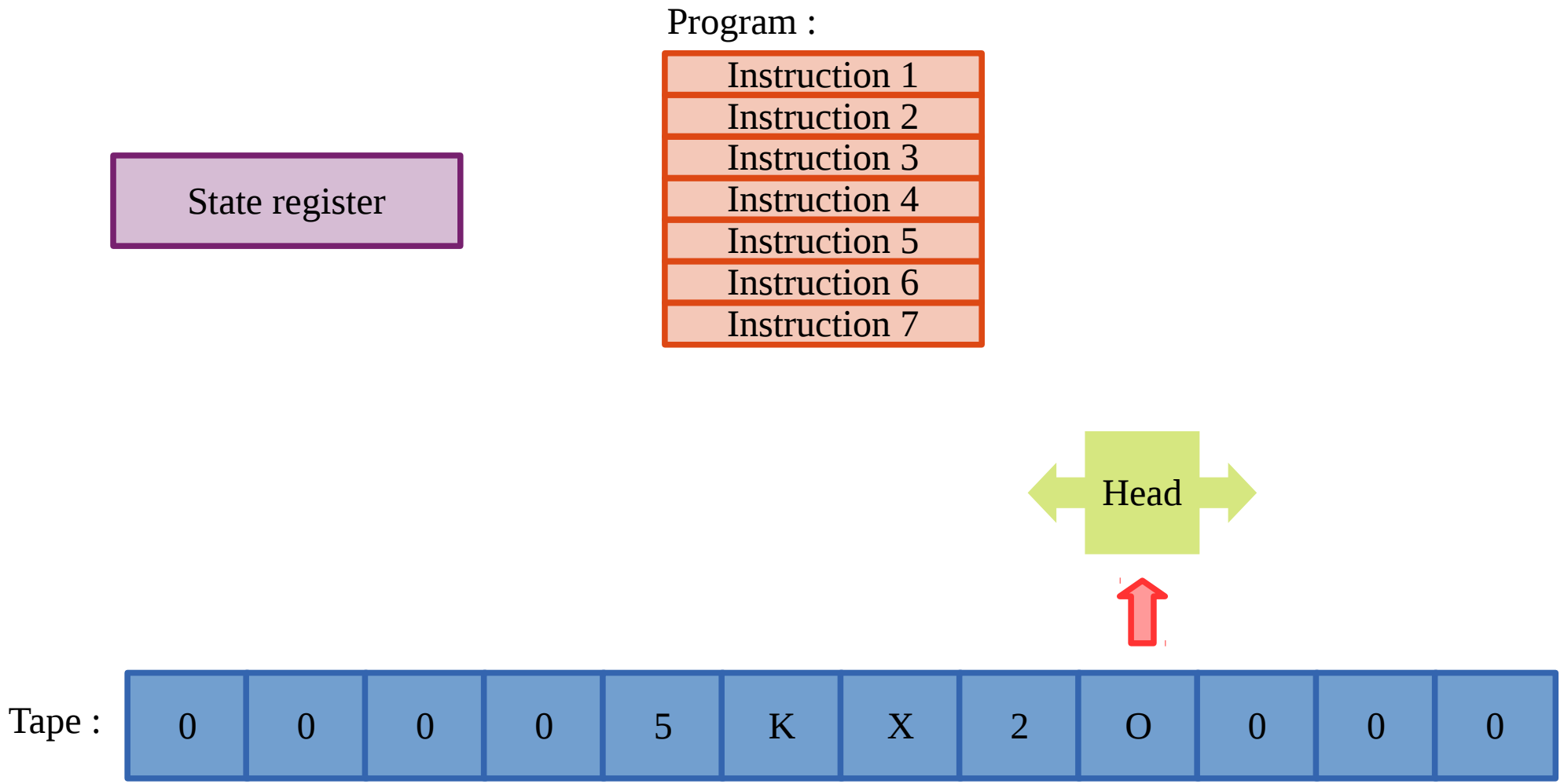
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



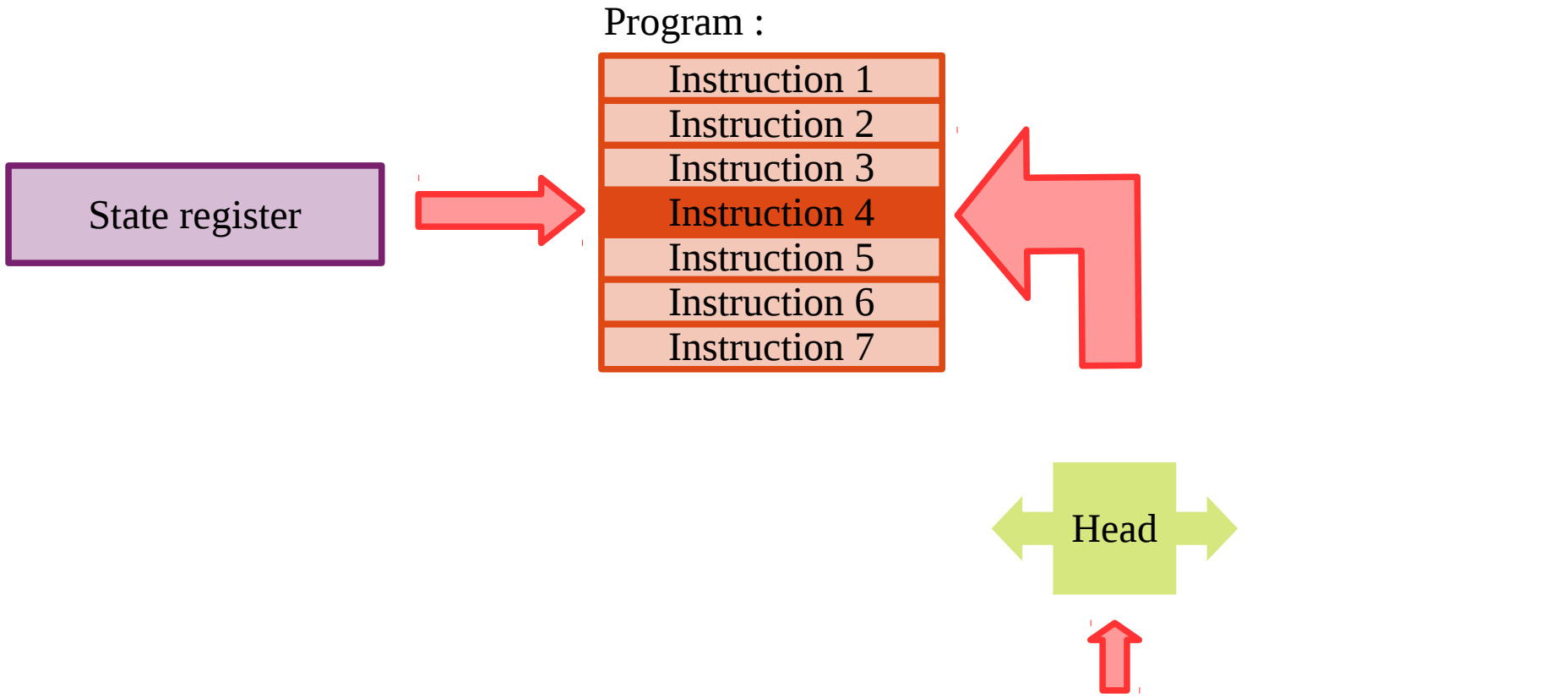
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



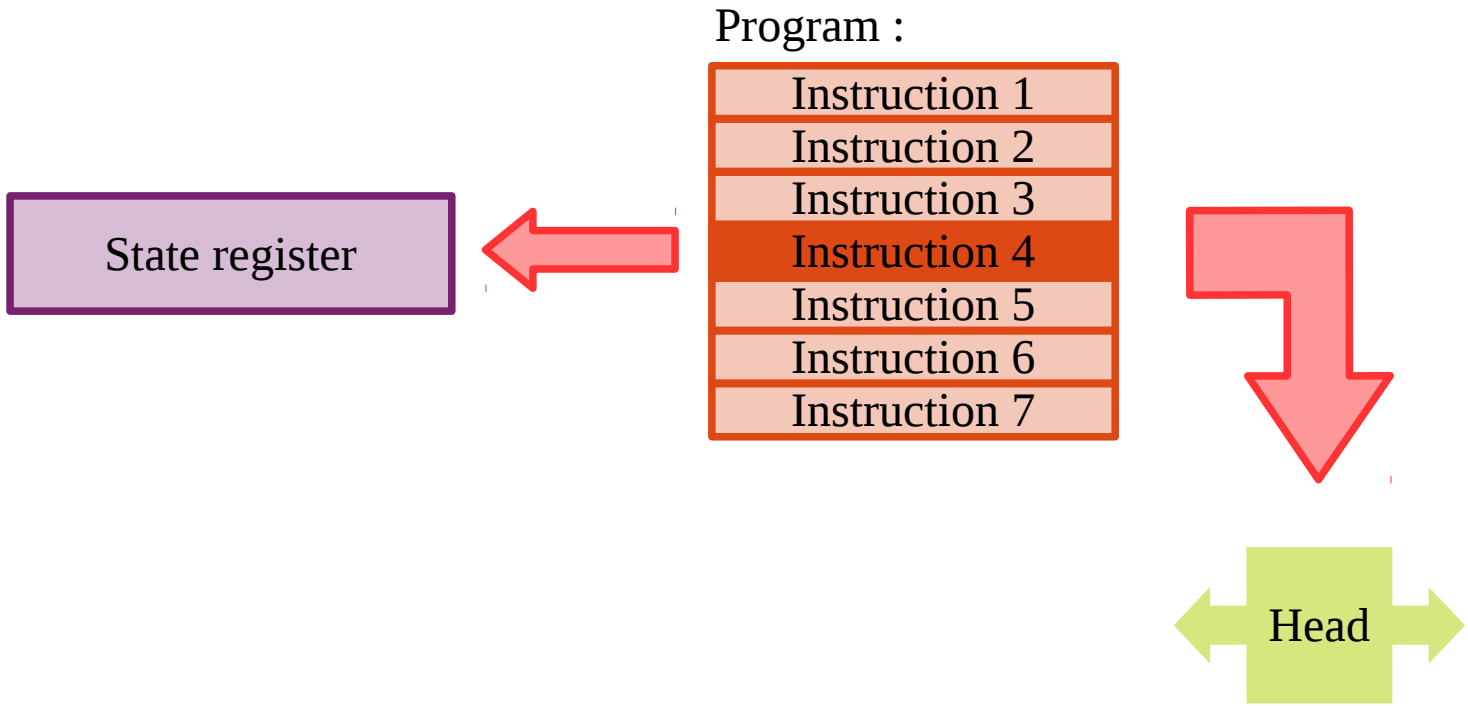
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**

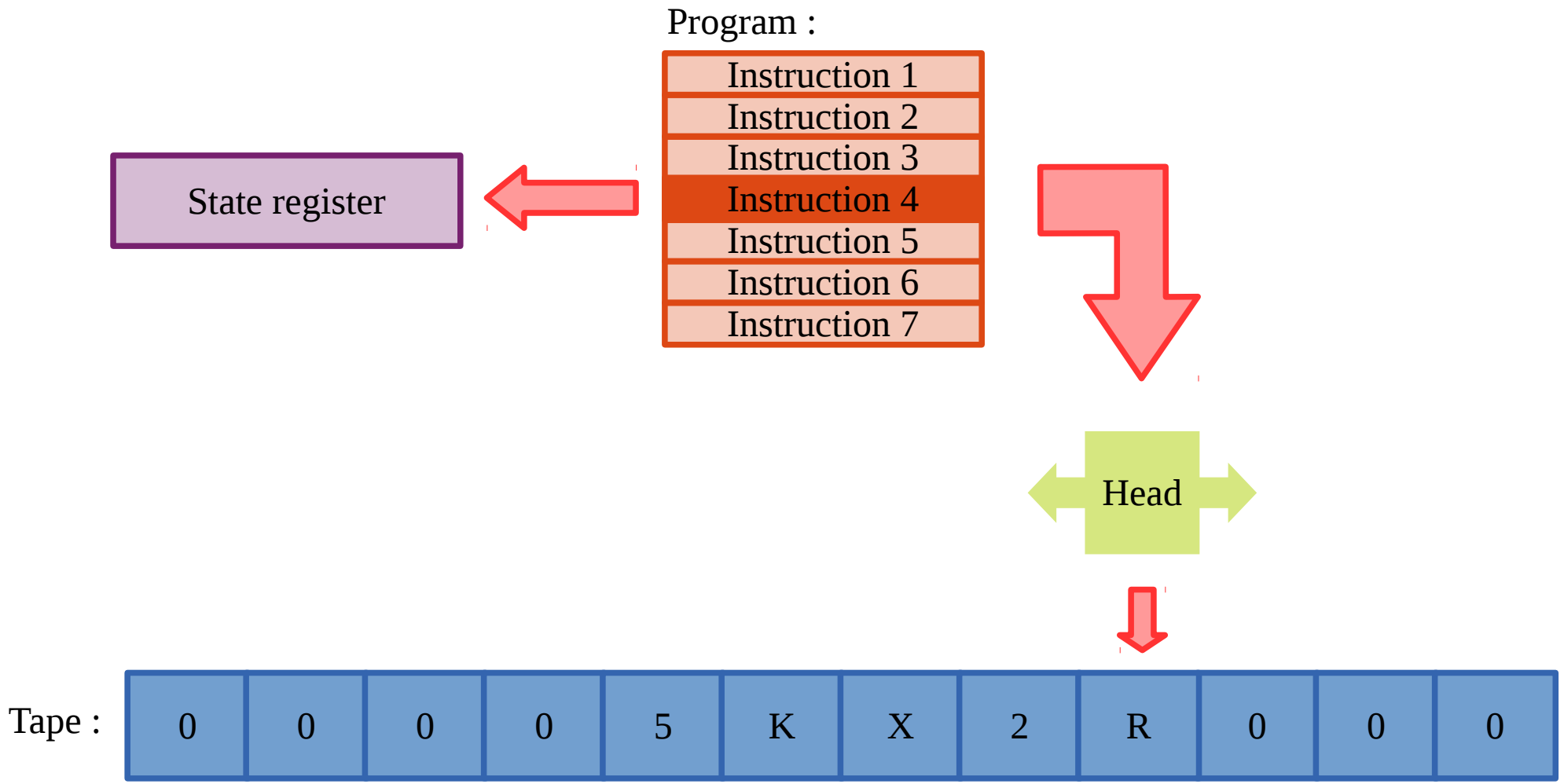


Tape :

0	0	0	0	5	K	X	2	O	0	0	0

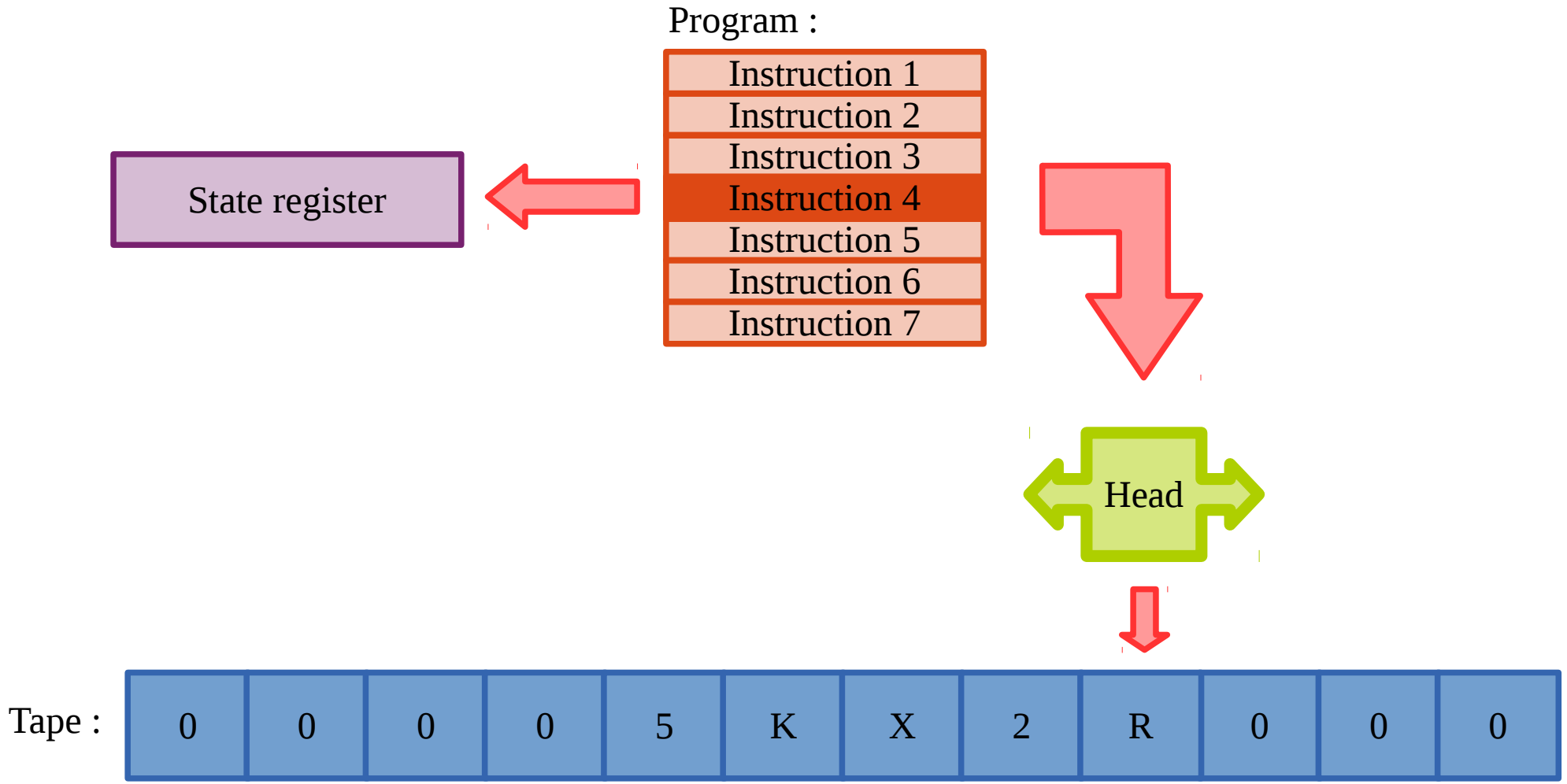
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



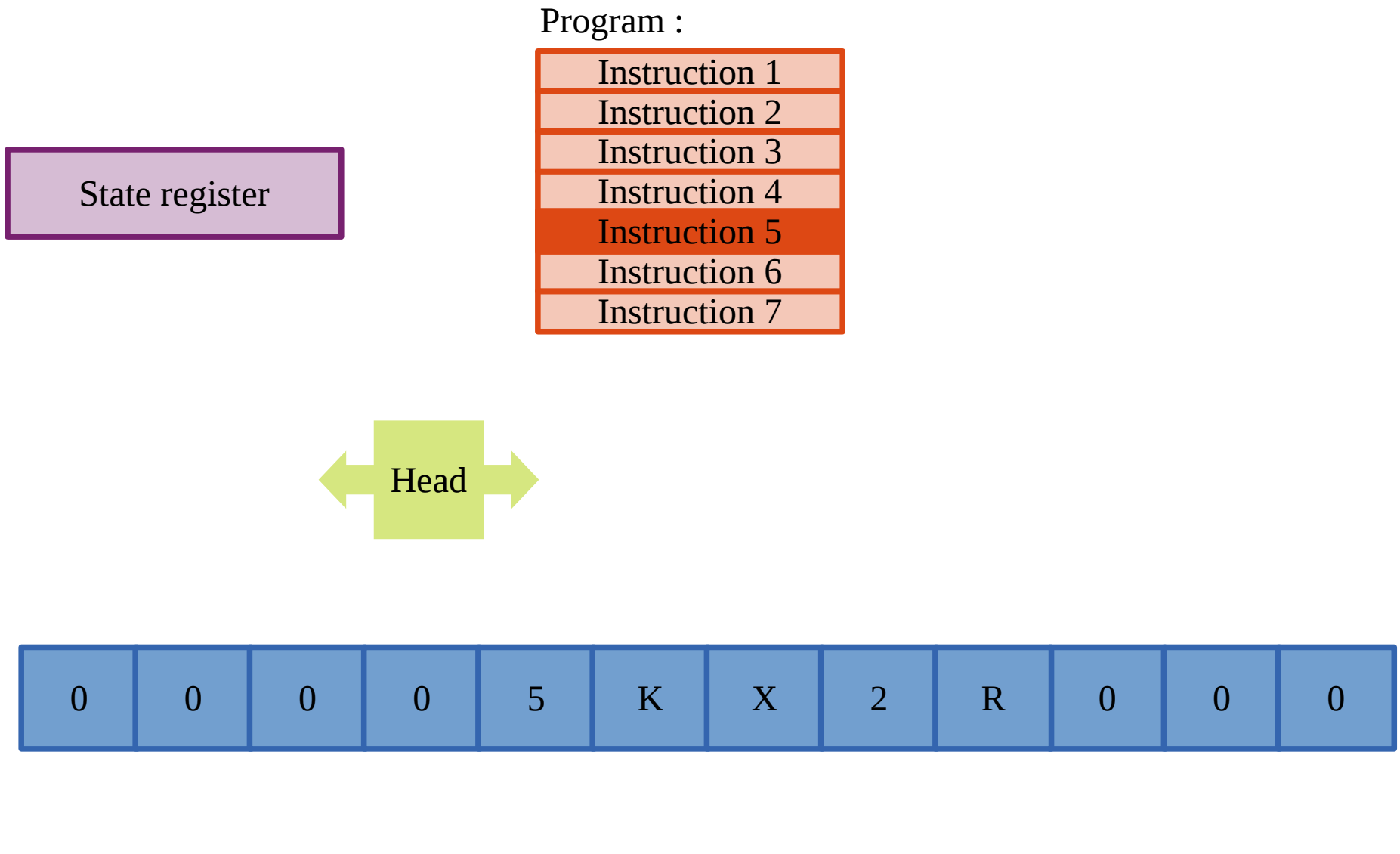
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



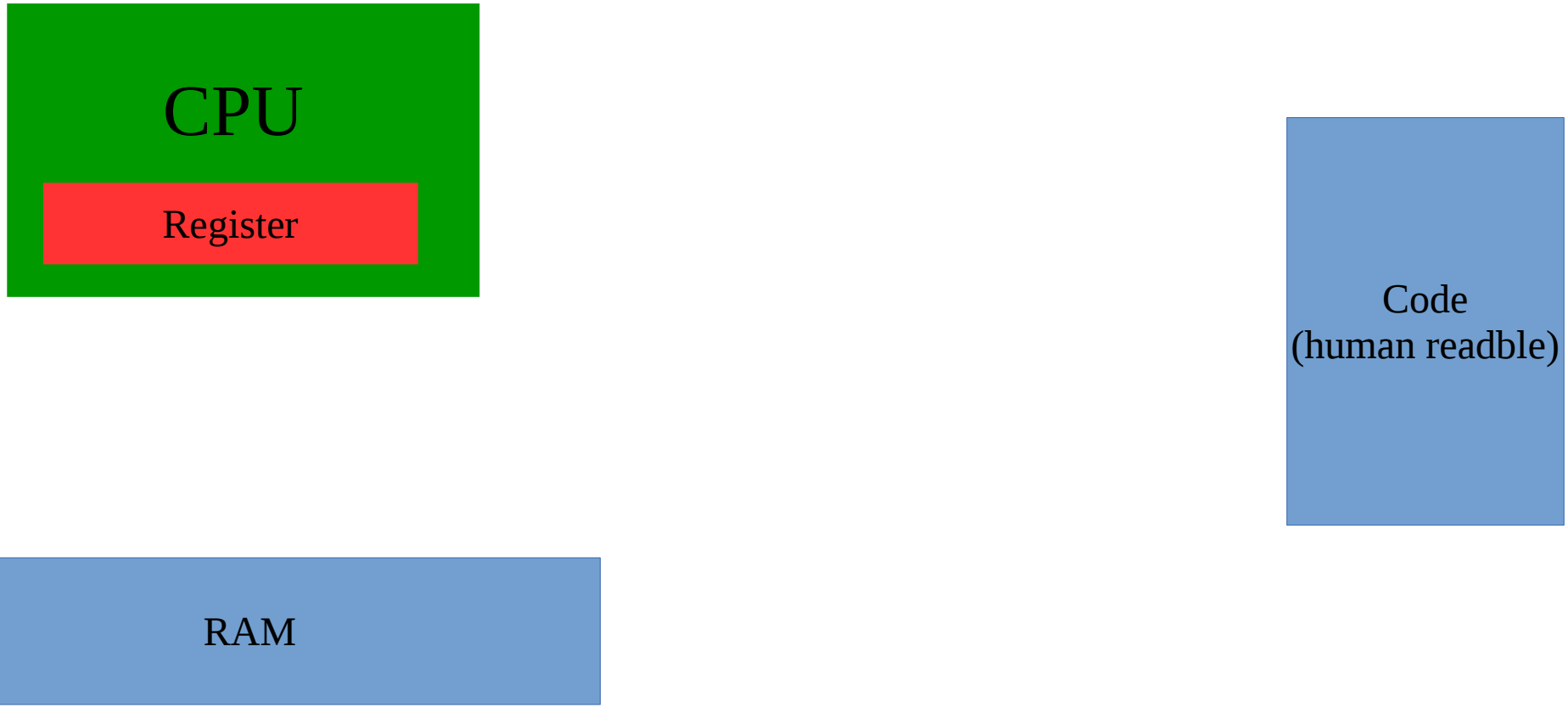
The Turing machine

- Alan Turing came up with a mathematical model of a machine that could perform a given set of instructions, on a given data set → **a computer**



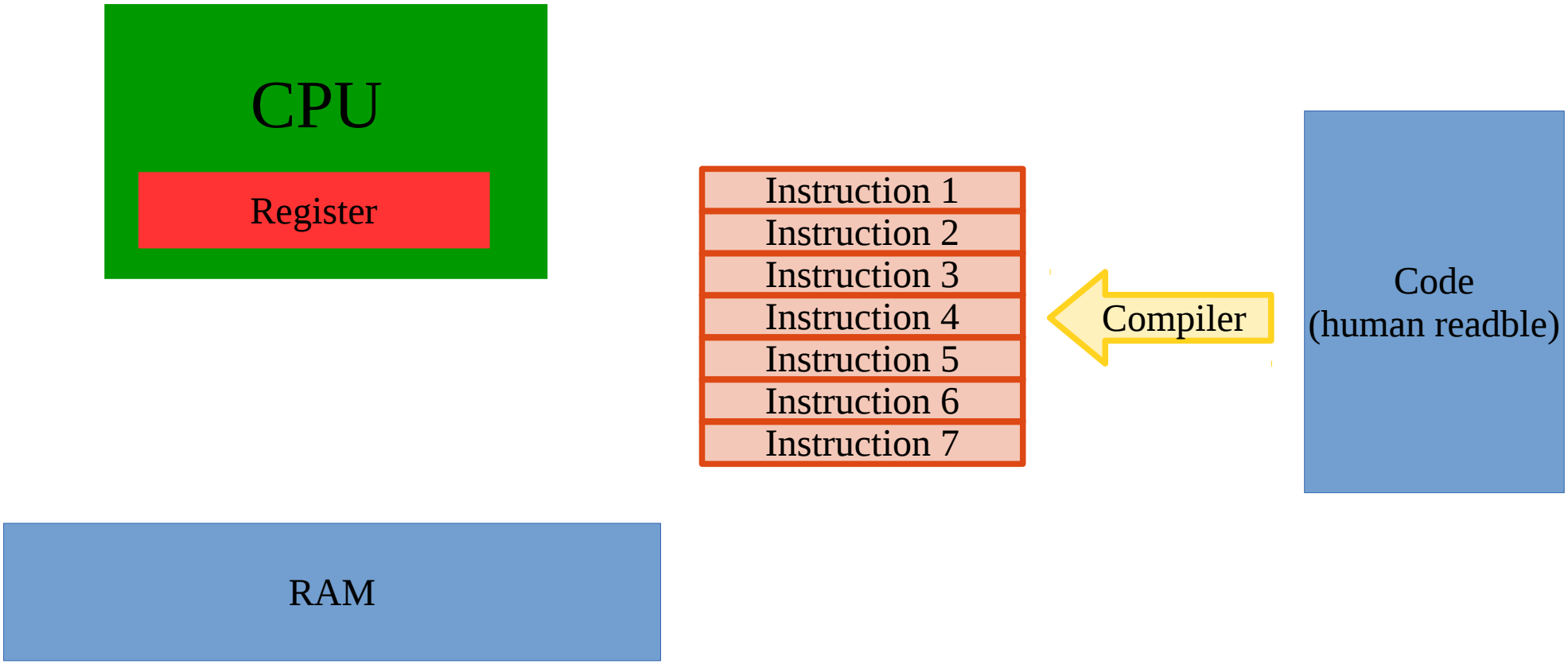
Modern (or common) computing architecture

- One can built a mechanical implementation of the Turing machine (or even out of LEGOs...)
  - The number of operations per unit time by mechanical constraints...
- The breakthrough comes with semiconductors, with nanosecond response time (→ GHz)



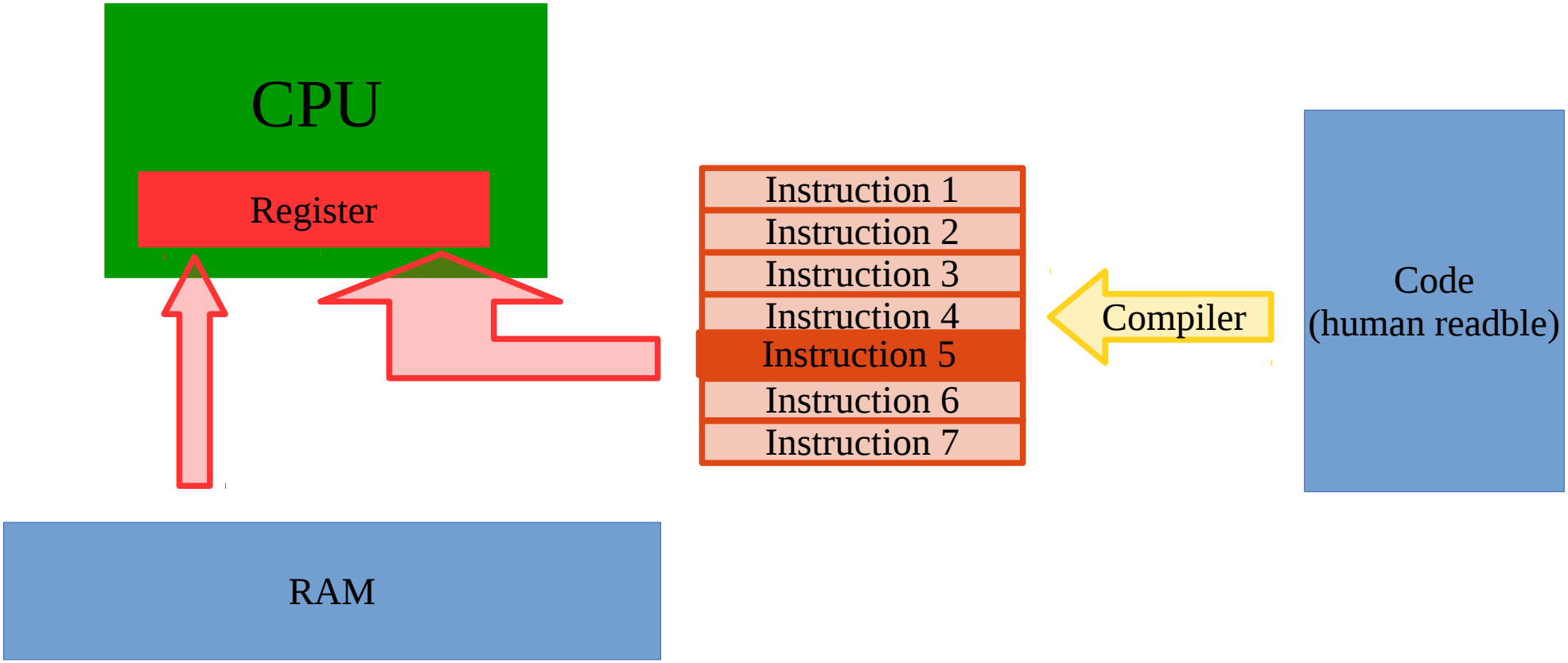
Modern (or common) computing architecture

- One can built a mechanical implementation of the Turing machine (or even out of LEGOs...)
  - The number of operations per unit time by mechanical constraints...
- The breakthrough comes with semiconductors, with nanosecond response time (→ GHz)



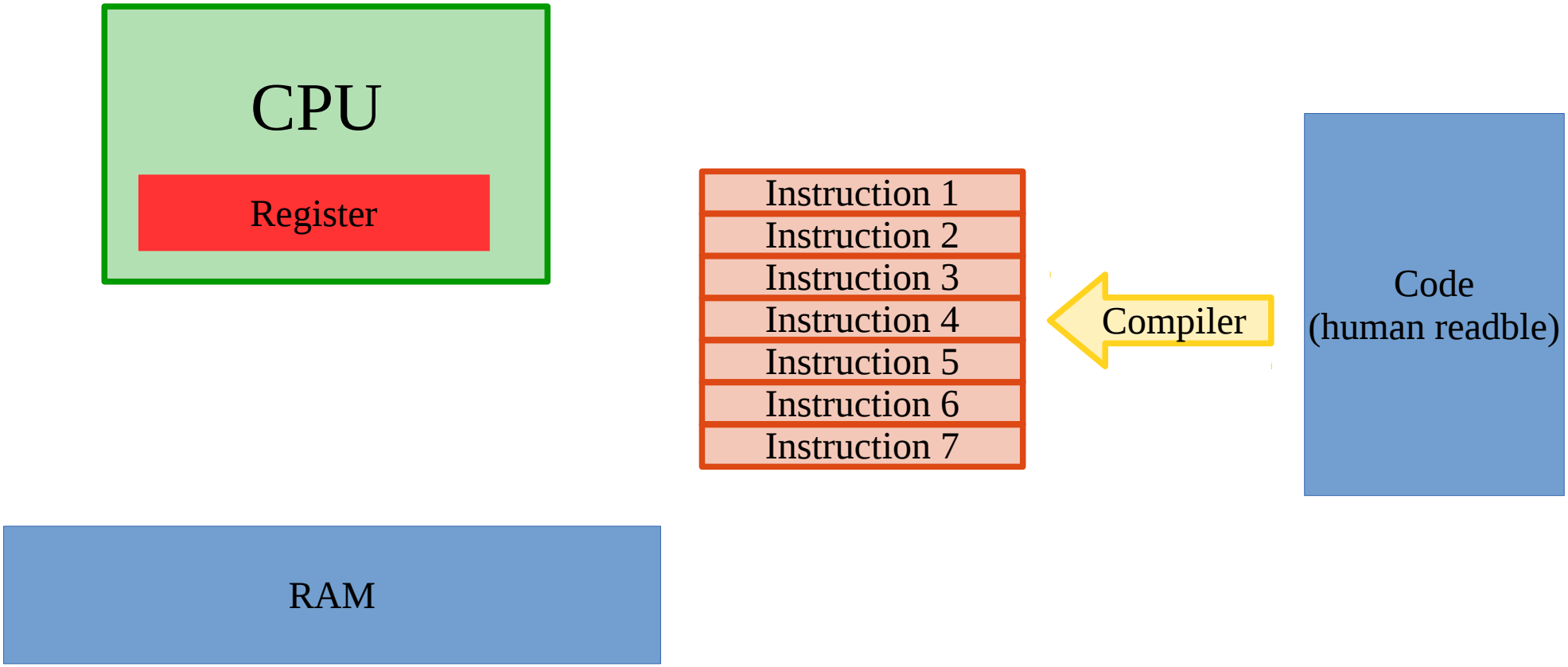
Modern (or common) computing architecture

- One can built a mechanical implementation of the Turing machine (or even out of LEGOs...)
  - The number of operations per unit time by mechanical constraints...
- The breakthrough comes with semiconductors, with nanosecond response time (→ GHz)



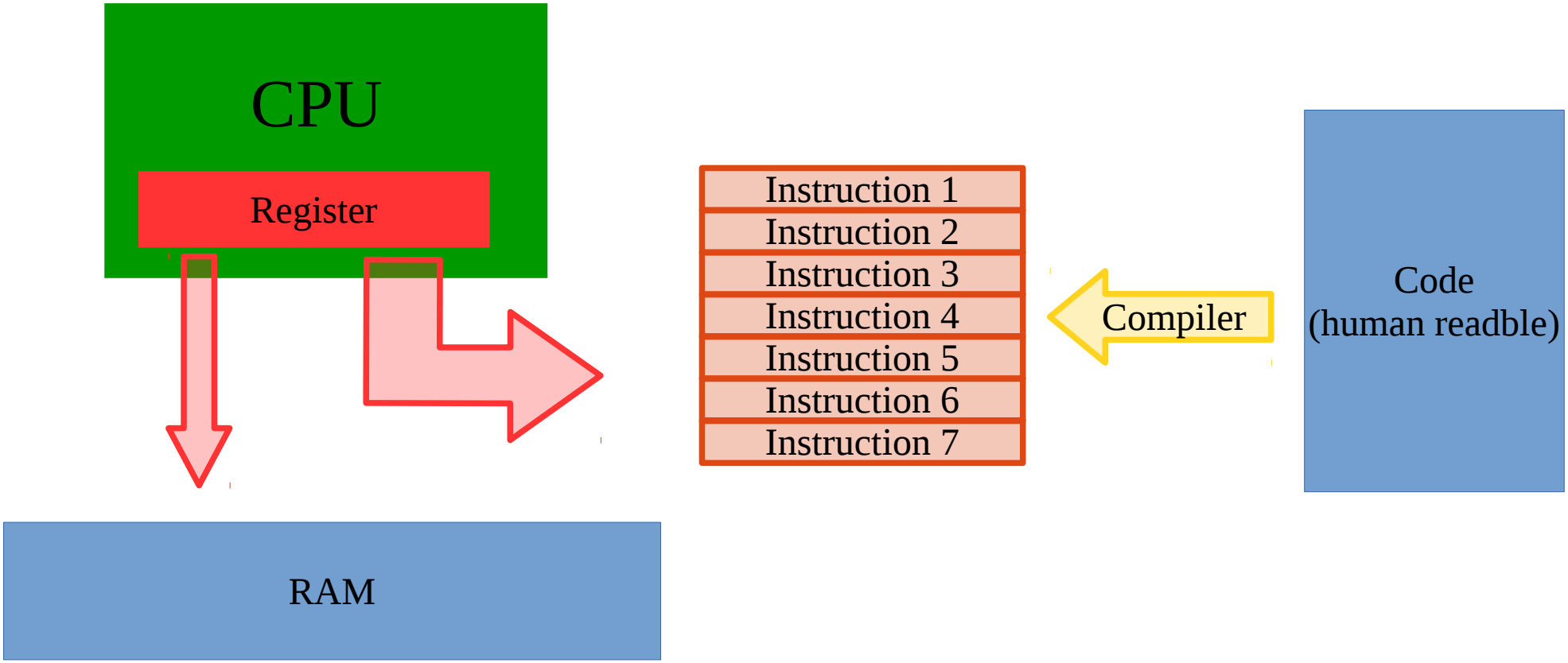
Modern (or common) computing architecture

- One can built a mechanical implementation of the Turing machine (or even out of LEGOs...)
  - The number of operations per unit time by mechanical constraints...
- The breakthrough comes with semiconductors, with nanosecond response time (→ GHz)



Modern (or common) computing architecture

- One can built a mechanical implementation of the Turing machine (or even out of LEGOs...)
  - The number of operations per unit time by mechanical constraints...
- The breakthrough comes with semiconductors, with nanosecond response time (→ GHz)



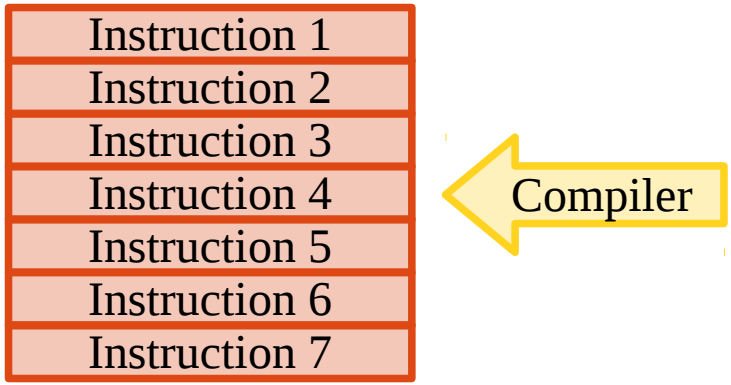
The central processing unit

CPU

- List of built in instructions to performed on a given data type (int, float, double, …)
  - Data handling
  - Control flow
  - Basic arithmetic operations, but not only!

    → Square root, trigonometric functions,... can be implemented as a single instruction !

Intrinsics



- The choice of the right instruction in the 'catalogue' offered by a given CPU is the role of the compiler
  - In some (few) cases, you know better than the compiler!
  - You can impose the usage of a given instruction → Intrinsics
- An intrinsic applies to a given data type
  - 8-bits, ... , 64-bits, ...
  - For many applications its is interesting to perform the same operation on two or more numbers at once → **Parallelism**

Vectorisation

$$x \rightarrow \sqrt{x}$$

x
Float 64-bits
(= double)

Vectorisation

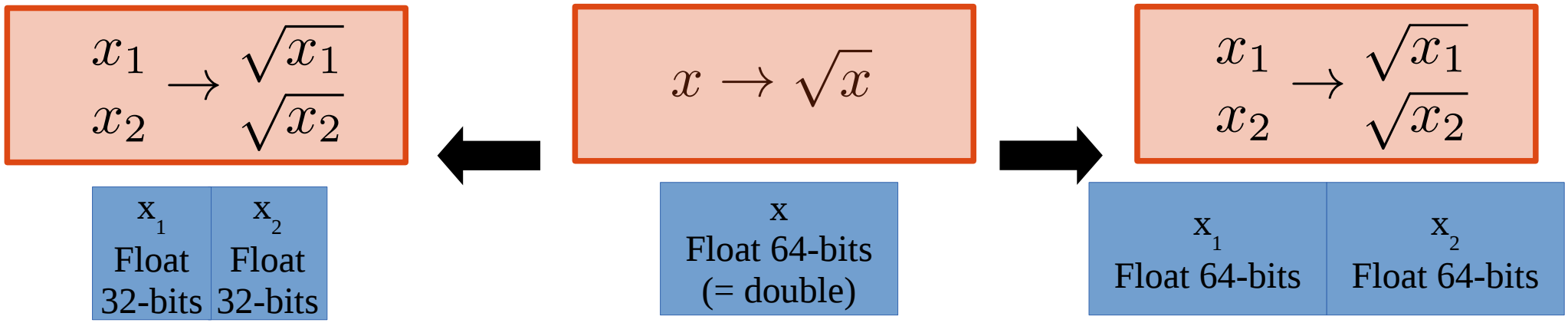
$$\begin{matrix} x_1 \\ x_2 \end{matrix} \rightarrow \begin{matrix} \sqrt{x_1} \\ \sqrt{x_2} \end{matrix}$$

$$x \rightarrow \sqrt{x}$$

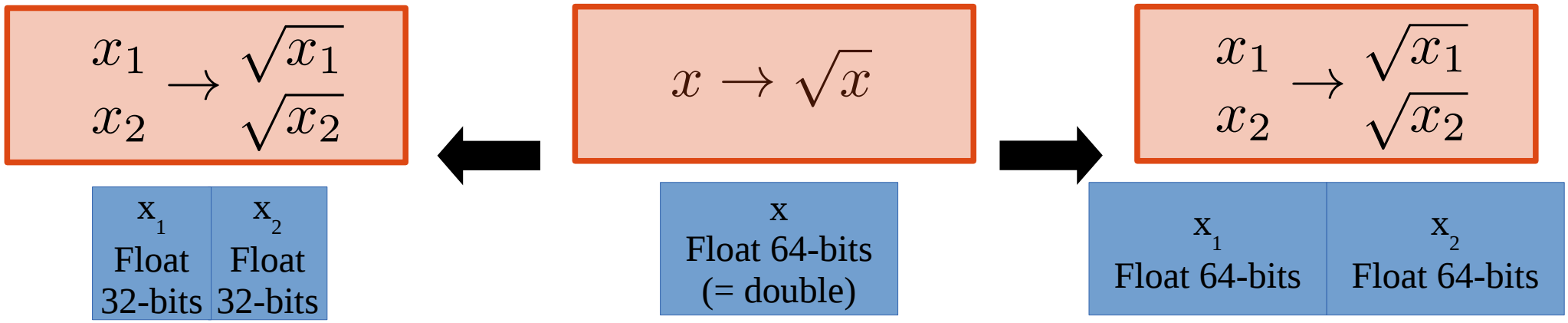
x_1	x_2
Float 32-bits	Float 32-bits

x
Float 64-bits
(= double)

Vectorisation



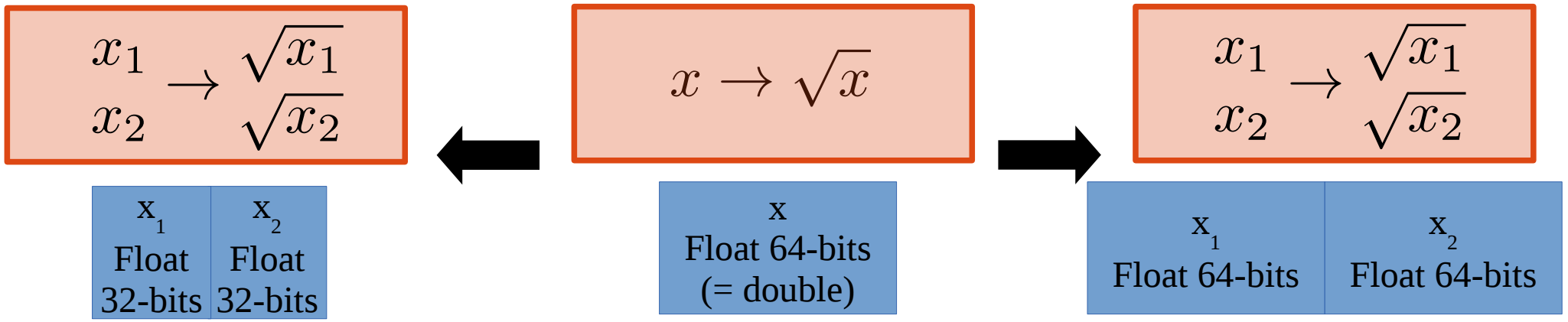
Vectorisation



- Intrinsics in C (SSE2) :

```
extern __m128d _mm_sqrt_pd(__m128d v1);
extern __m256d _mm256_sqrt_pd(__m256d v1);
```

Vectorisation

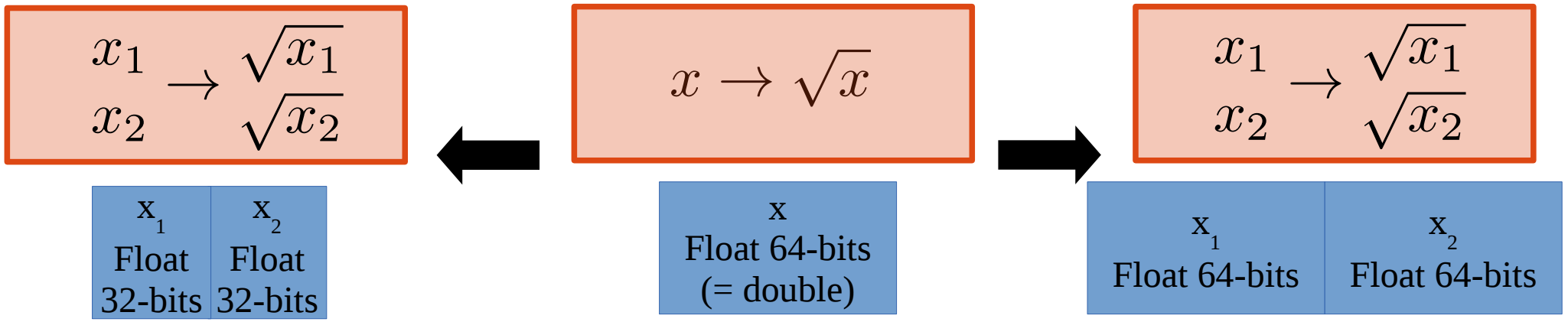


- Intrinsics in C (SSE2) :

- 4xfloats 32-bits or 2x64-bits

```
extern __m128d _mm_sqrt_pd(__m128d v1);
extern __m256d _mm256_sqrt_pd(__m256d v1);
```

Vectorisation

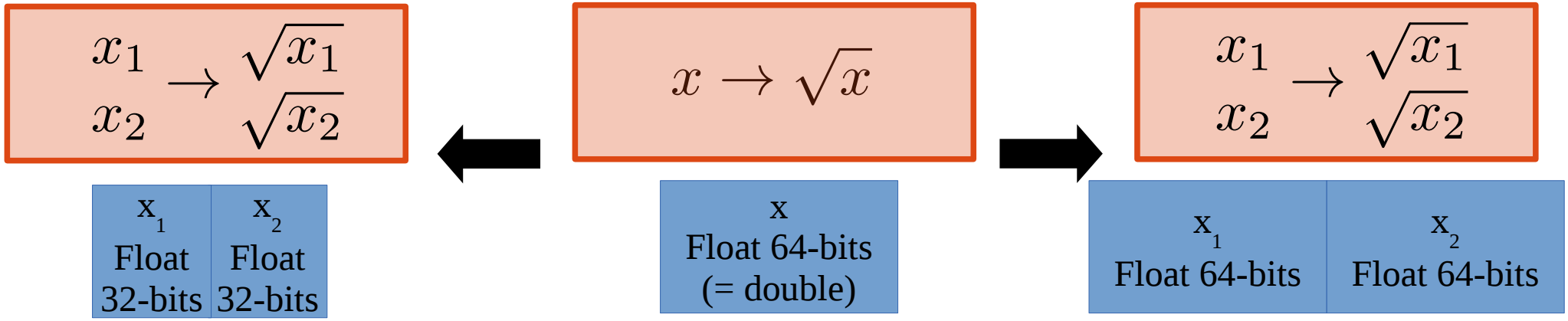


- Intrinsics in C (SSE2) :

```
extern __m128d _mm_sqrt_pd(__m128d v1);
extern __m256d _mm256_sqrt_pd(__m256d v1);
```

- 4xfloats 32-bits or 2x64-bits
- 8xfloats 32-bits or 4x64-bits

Vectorisation



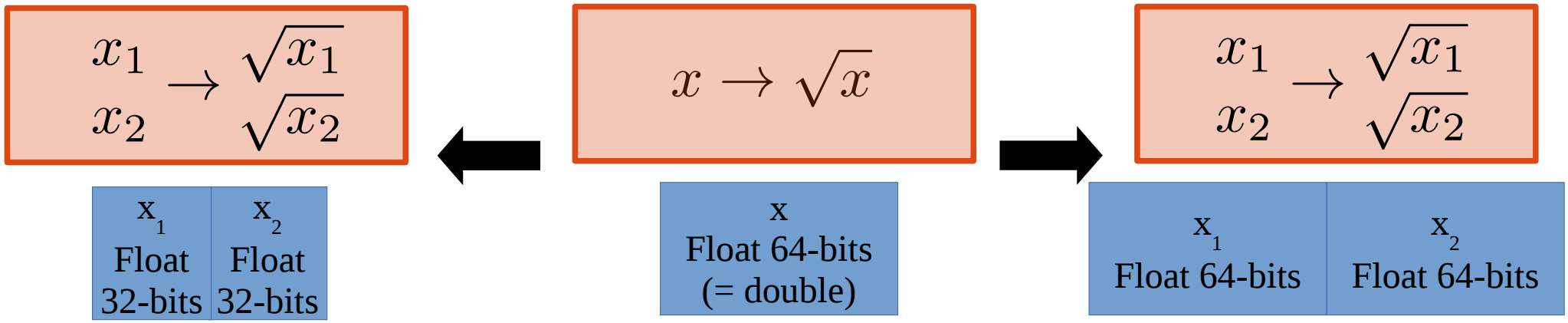
- Intrinsics in C (SSE2) :

```
extern __m128d _mm_sqrt_pd(__m128d v1);
extern __m256d _mm256_sqrt_pd(__m256d v1);
```

- 4xfloats 32-bits or 2x64-bits
- 8xfloats 32-bits or 4x64-bits

- The vectorisation capabilities are particularly trendy nowadays :
  - Intel Sandy Bridge (128-bits operations, 2011)
  - Intel Haswell (256-bits operations, 2013)
  - Intel Xeon Phi (512-bits operations, 2016)

Vectorisation



- Intrinsics in C (SSE2) :

```
extern __m128d _mm_sqrt_pd(__m128d v1);
extern __m256d _mm256_sqrt_pd(__m256d v1);
```

- 4xfloats 32-bits or 2x64-bits
- 8xfloats 32-bits or 4x64-bits

- The vectorisation capabilities are particularly trendy nowadays :
  - Intel Sandy Bridge (128-bits operations, 2011)
  - Intel Haswell (256-bits operations, 2013)
  - Intel Xeon Phi (512-bits operations, 2016) → **8x64-bits operations at once!**

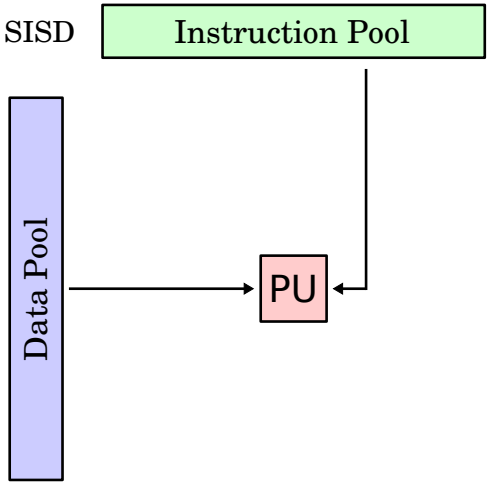
Compiler options

- Intel and Microsofts's website offers documentation about the intrinsics
- But **don't kid yourself**, there are people spending their lives writing compilers, they are usually better than you!

  → However you may help the compiler exploit the full capabilities of your hardware!
- In gcc the option -ftree-vectorize (also in -O2) enable vectorisation
- The compiler tries to recognize vectorizable patterns in your loops (some compilers are better than others...improving every day)

Flynn's taxonomy

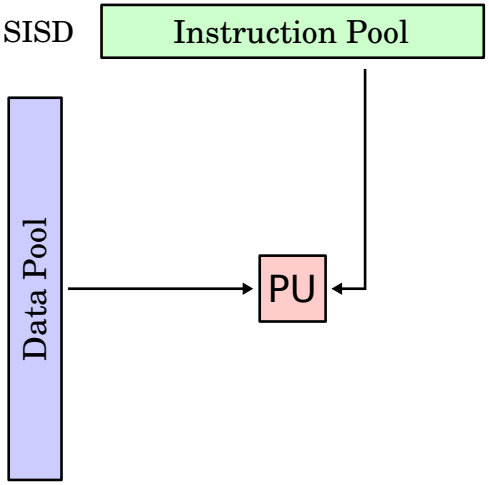
- Single instruction, single data
  - Most common / simple



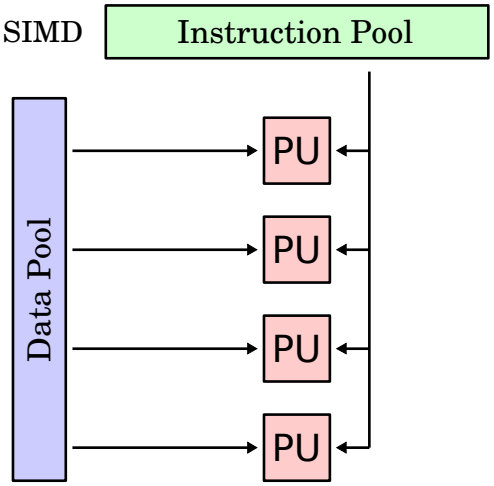
Illustrations from https://en.wikipedia.org/wiki/Flynn's_taxonomy

Flynn's taxonomy

- Single instruction, single data
  - Most common / simple



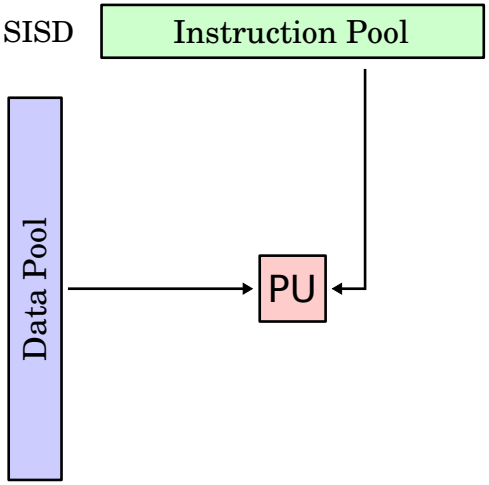
- Single instruction, **multiple** data
  - Vectorisation
  - Graphics processing units (GPU)

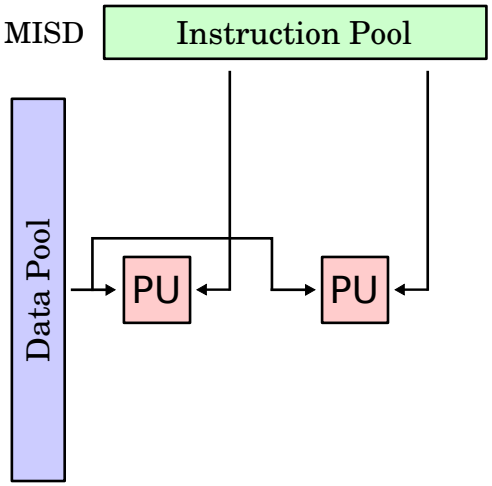


Illustrations from https://en.wikipedia.org/wiki/Flynn's_taxonomy

Flynn's taxonomy

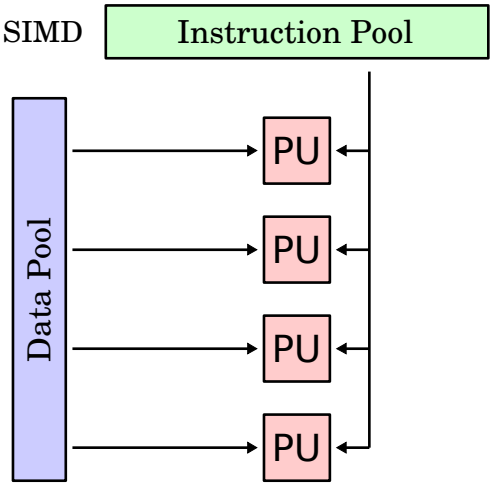
- Single instruction, single data
  - Most common / simple





- **Multiple** instructions, single data
  - ?

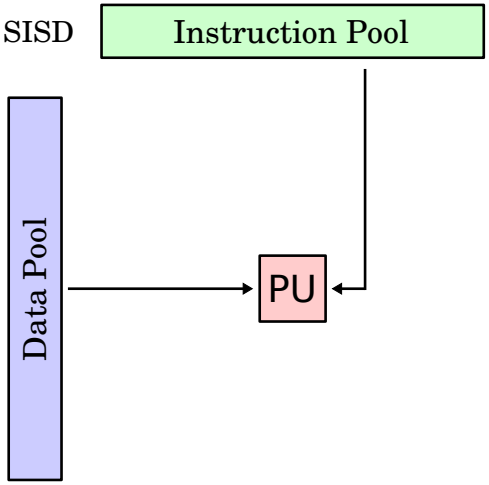
- Single instruction, **multiple** data
  - Vectorisation
  - Graphics processing units (GPU)

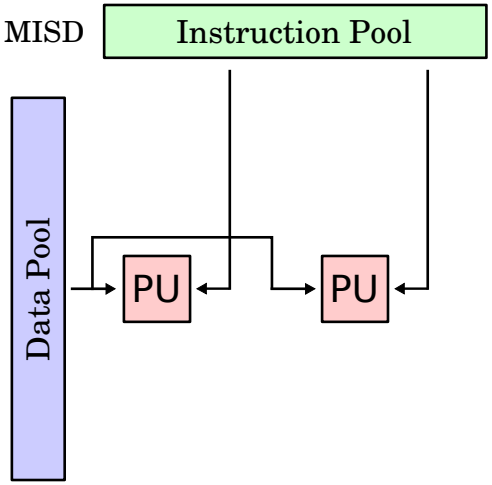


Illustrations from https://en.wikipedia.org/wiki/Flynn's_taxonomy

Flynn's taxonomy

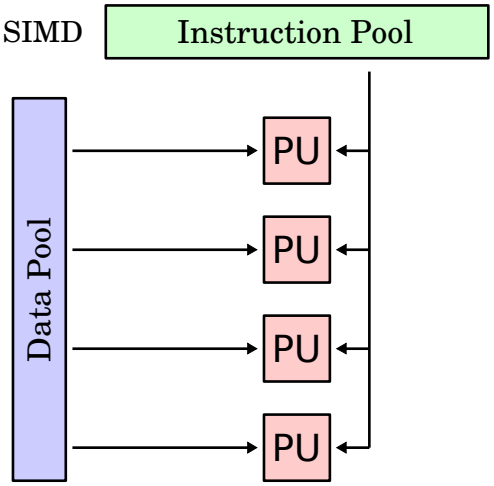
- Single instruction, single data
  - Most common / simple

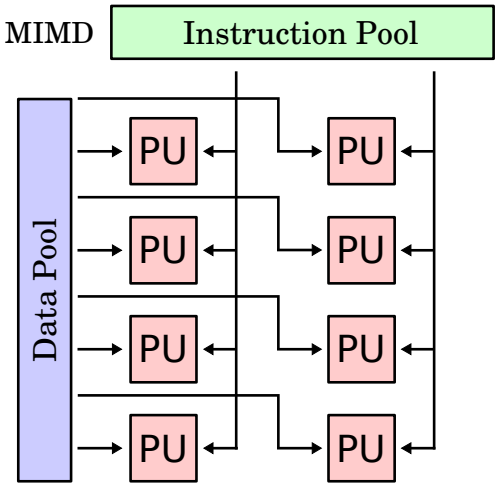




- **Multiple** instructions, single data
  - ?

- Single instruction, **multiple** data
  - Vectorisation
  - Graphics processing units (GPU)

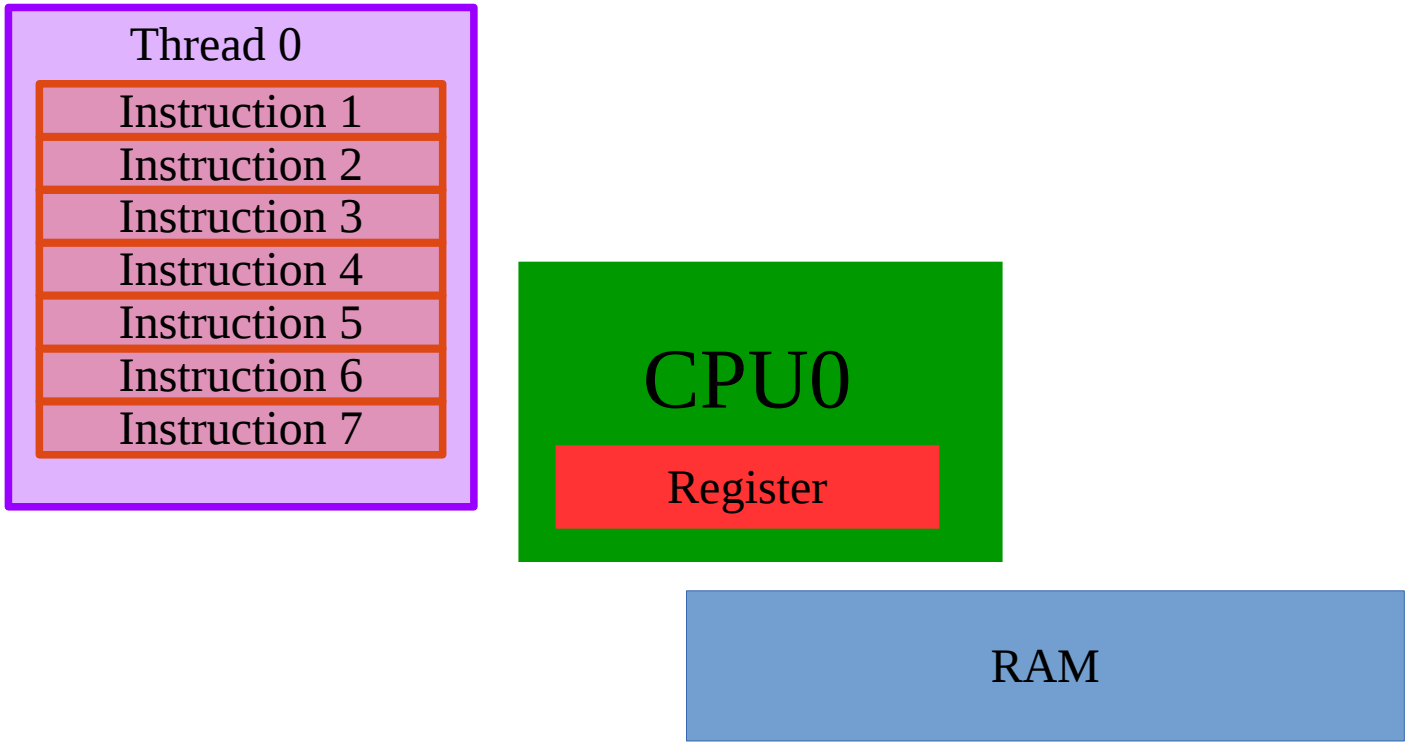




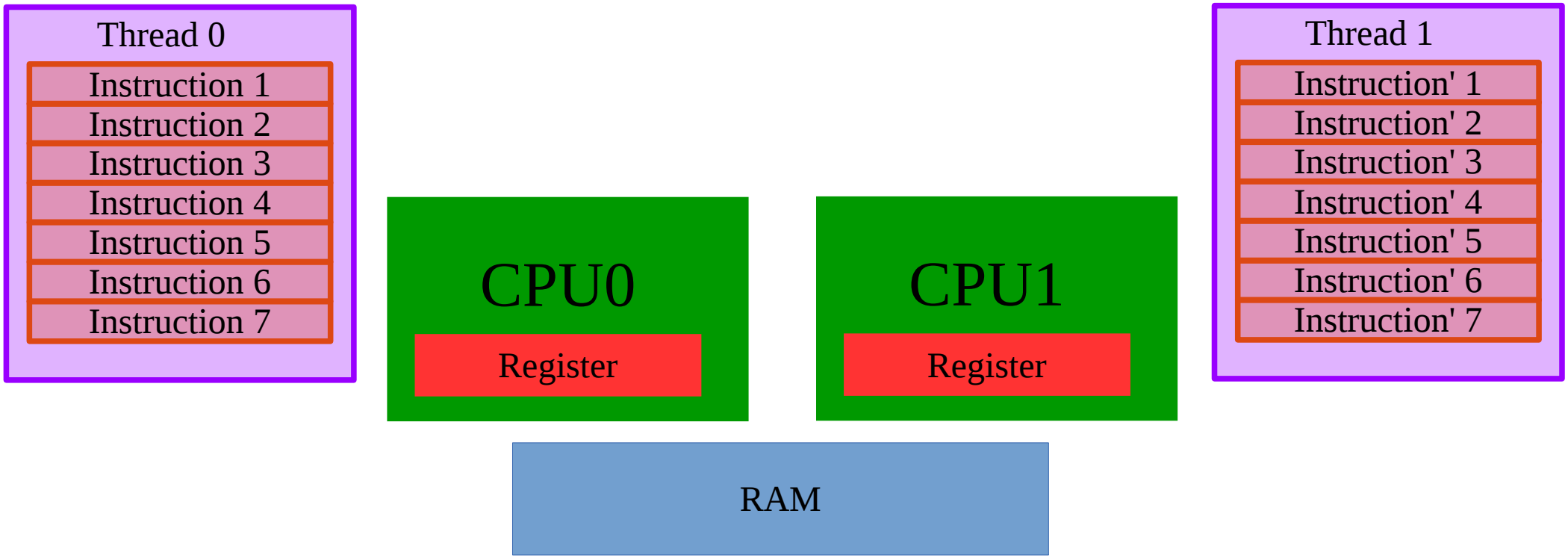
- **Multiple** instructions, **multiple** data
  - Multithreading (shared memory parallelisation)
  - Cluster (distributed memory parallelisation)

Illustrations from https://en.wikipedia.org/wiki/Flynn's_taxonomy

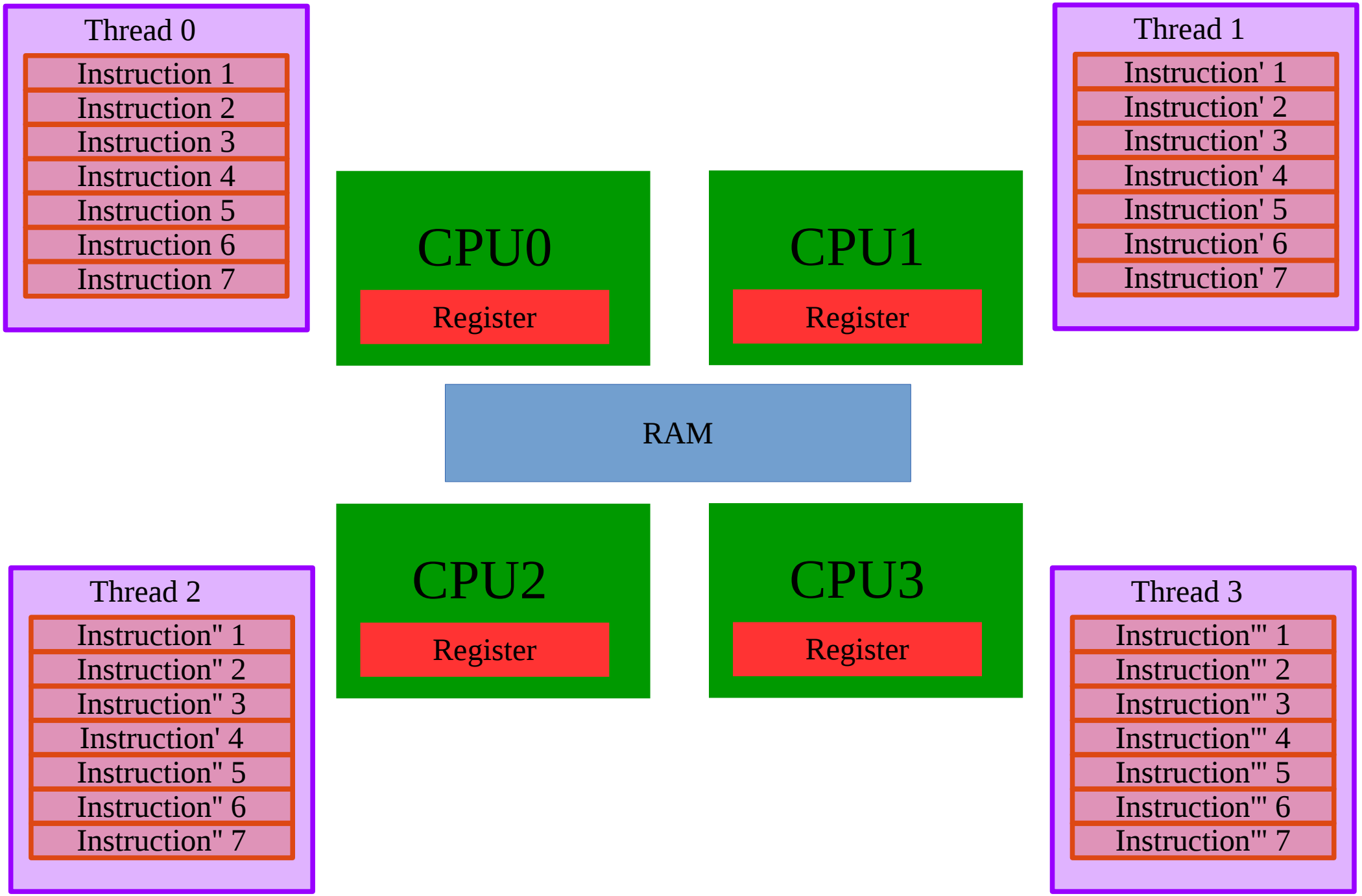
Multithreading



Multithreading



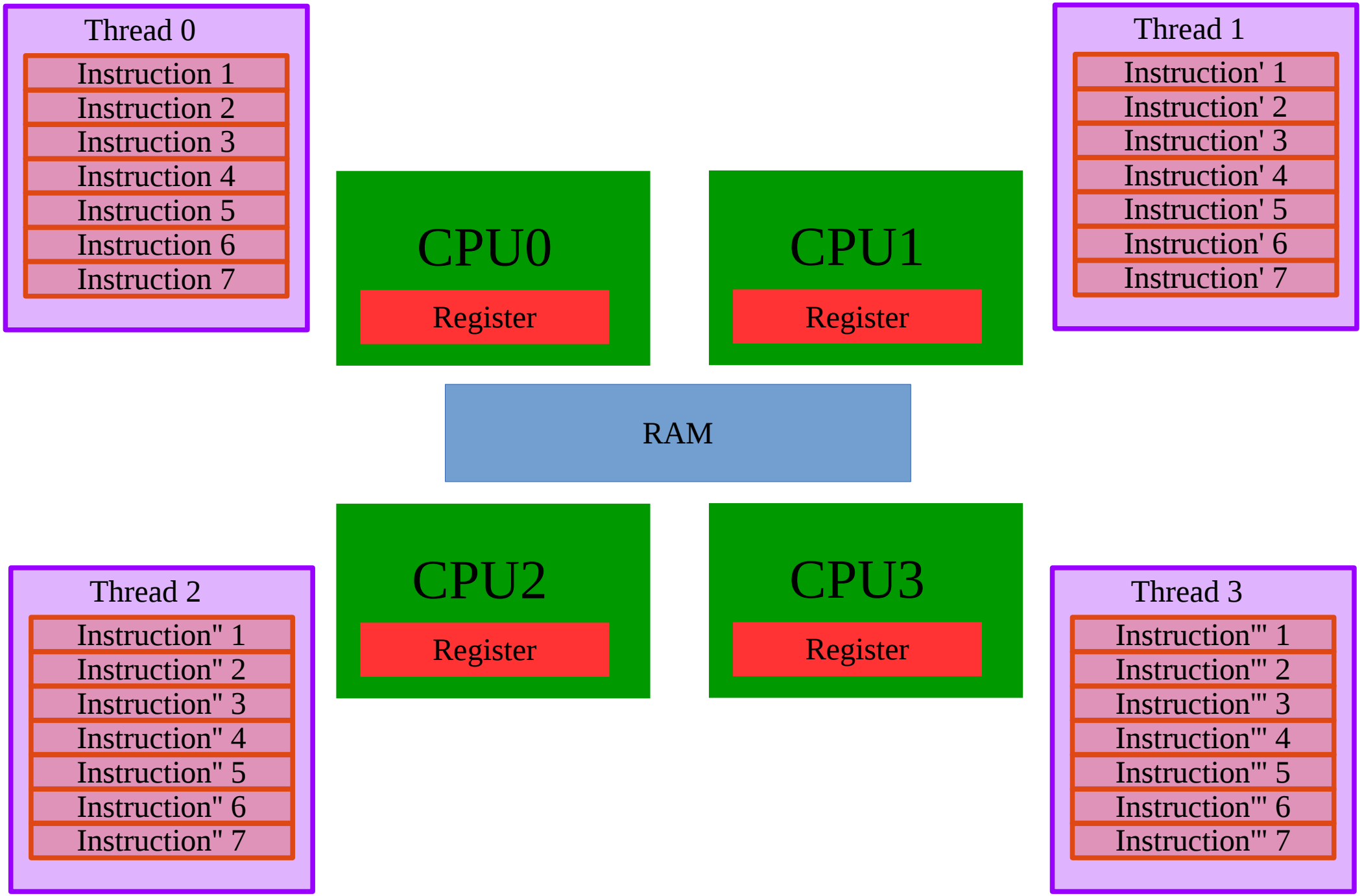
Multithreading



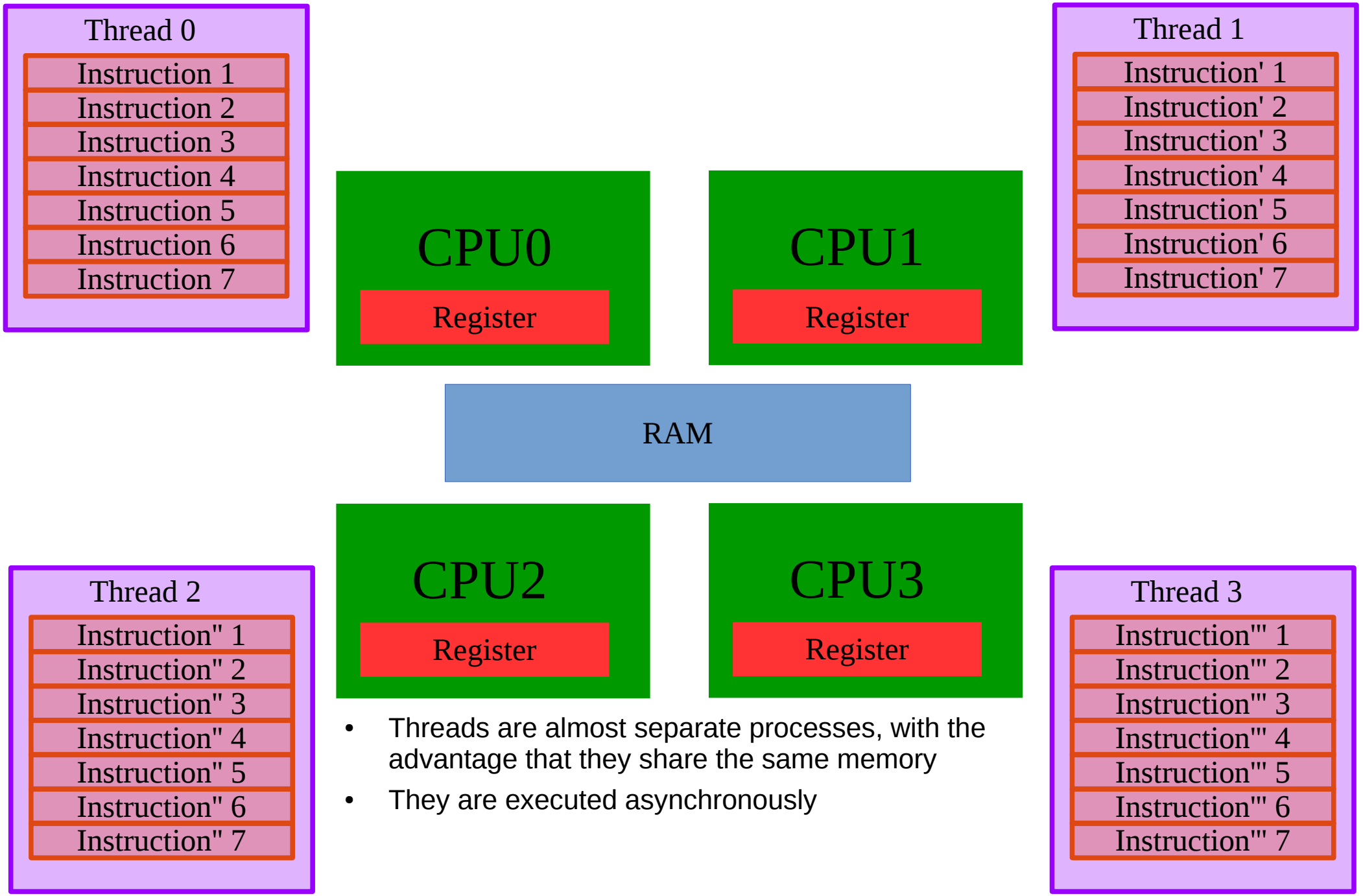
Multithreading : expectation vs reality

Multithreading : expectation vs reality

Multithreading : The limits

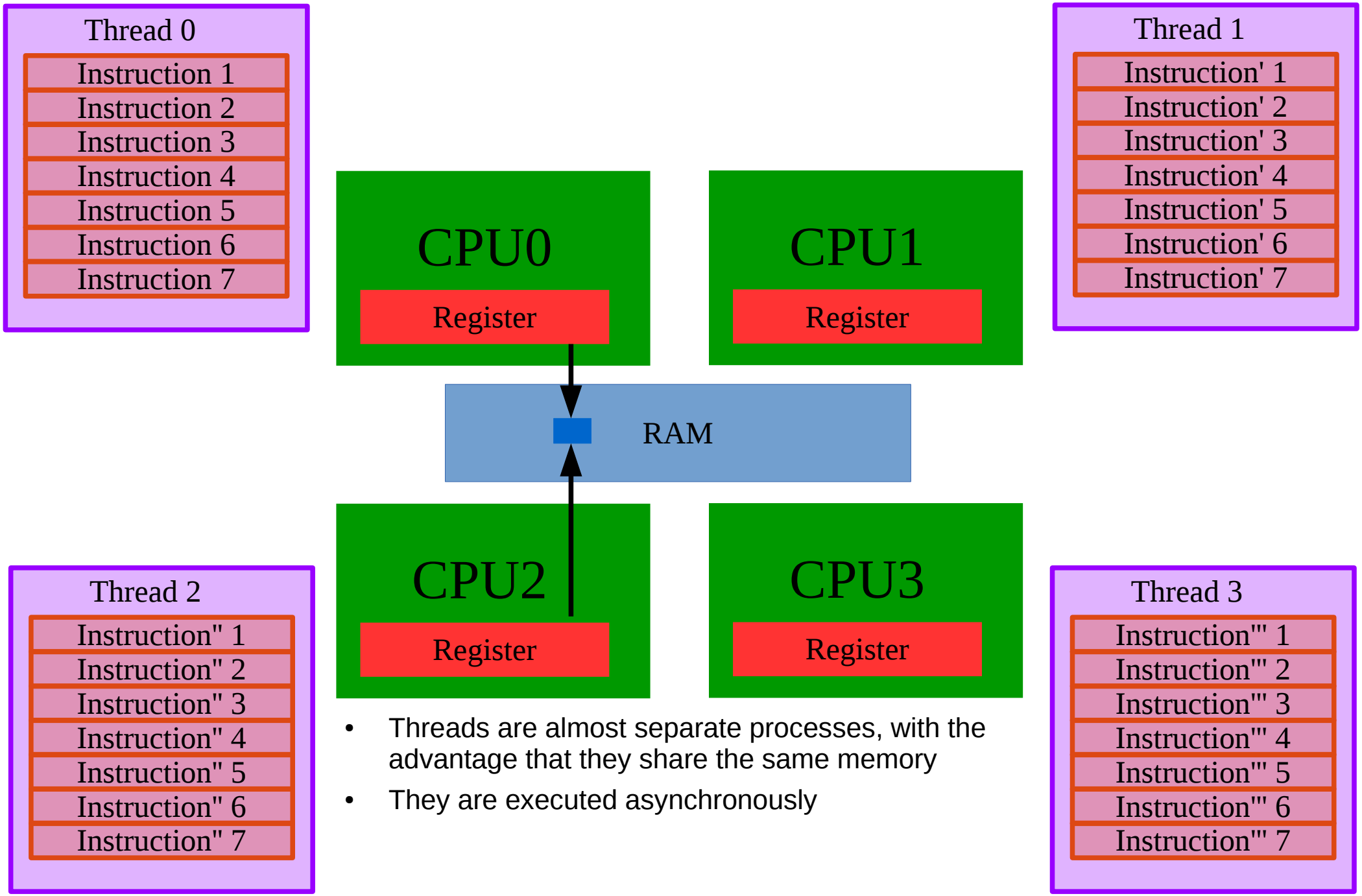


Multithreading : The limits



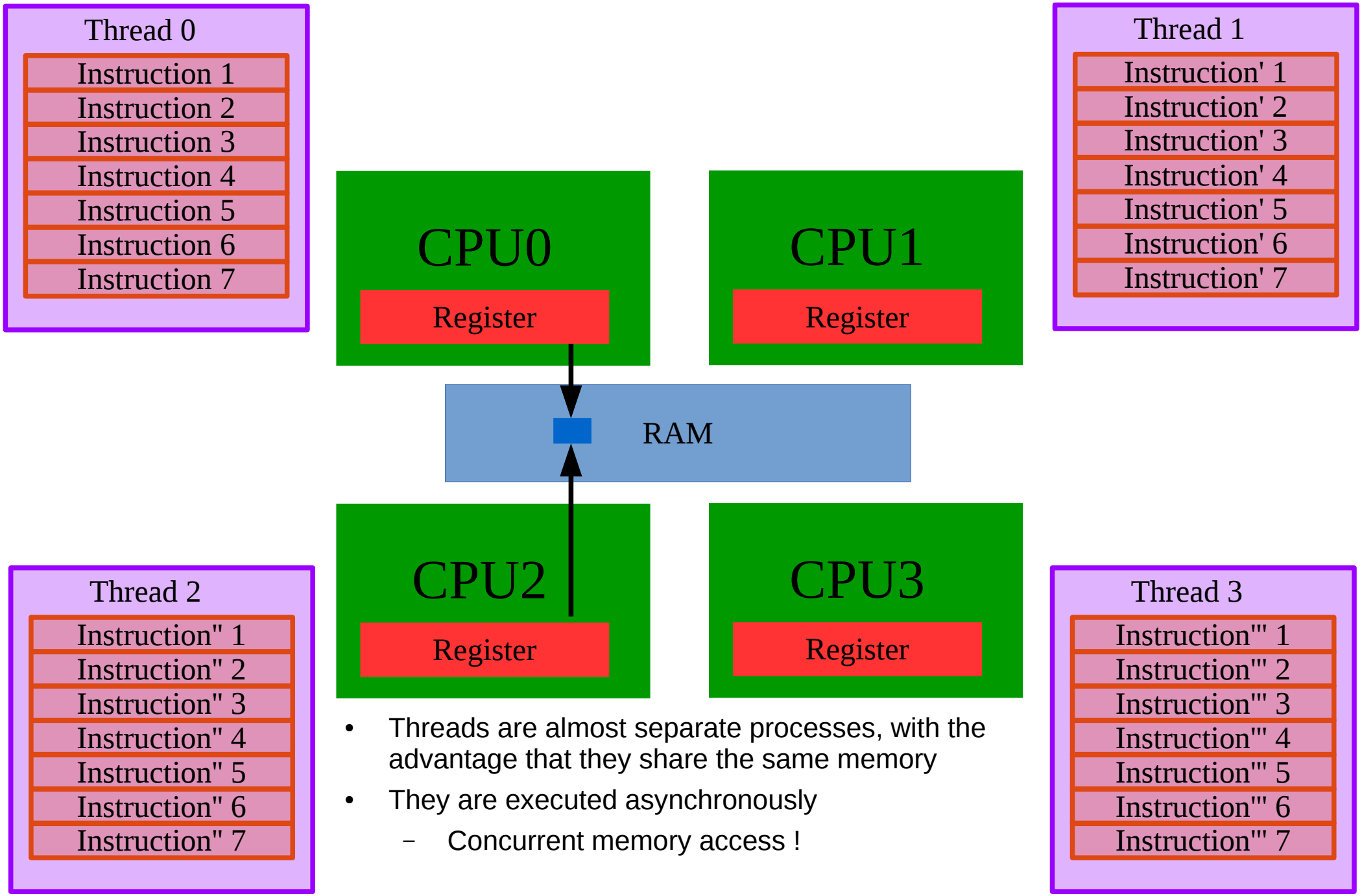
- Threads are almost separate processes, with the advantage that they share the same memory
- They are executed asynchronously

Multithreading : The limits



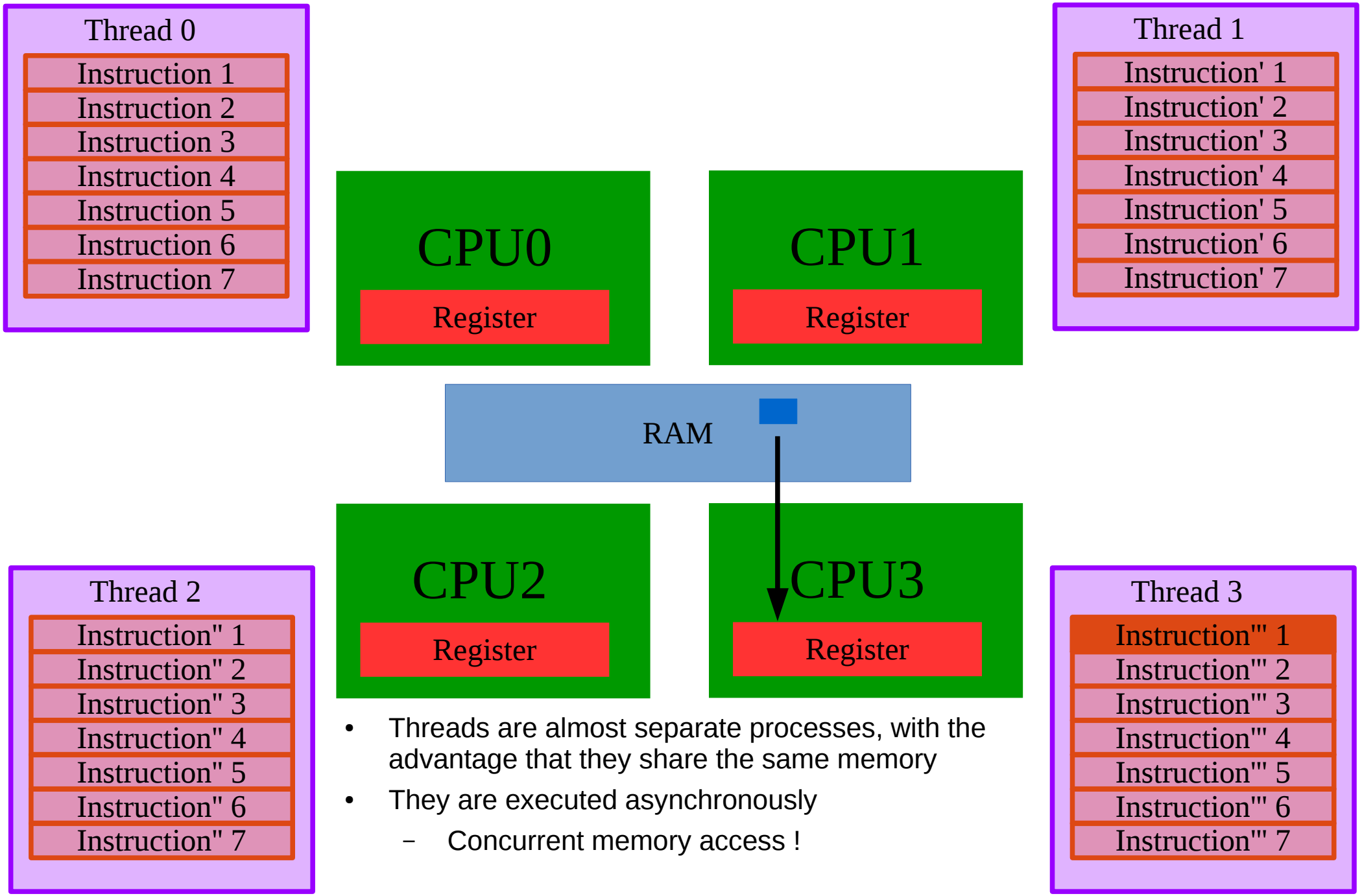
- Threads are almost separate processes, with the advantage that they share the same memory
- They are executed asynchronously

Multithreading : The limits



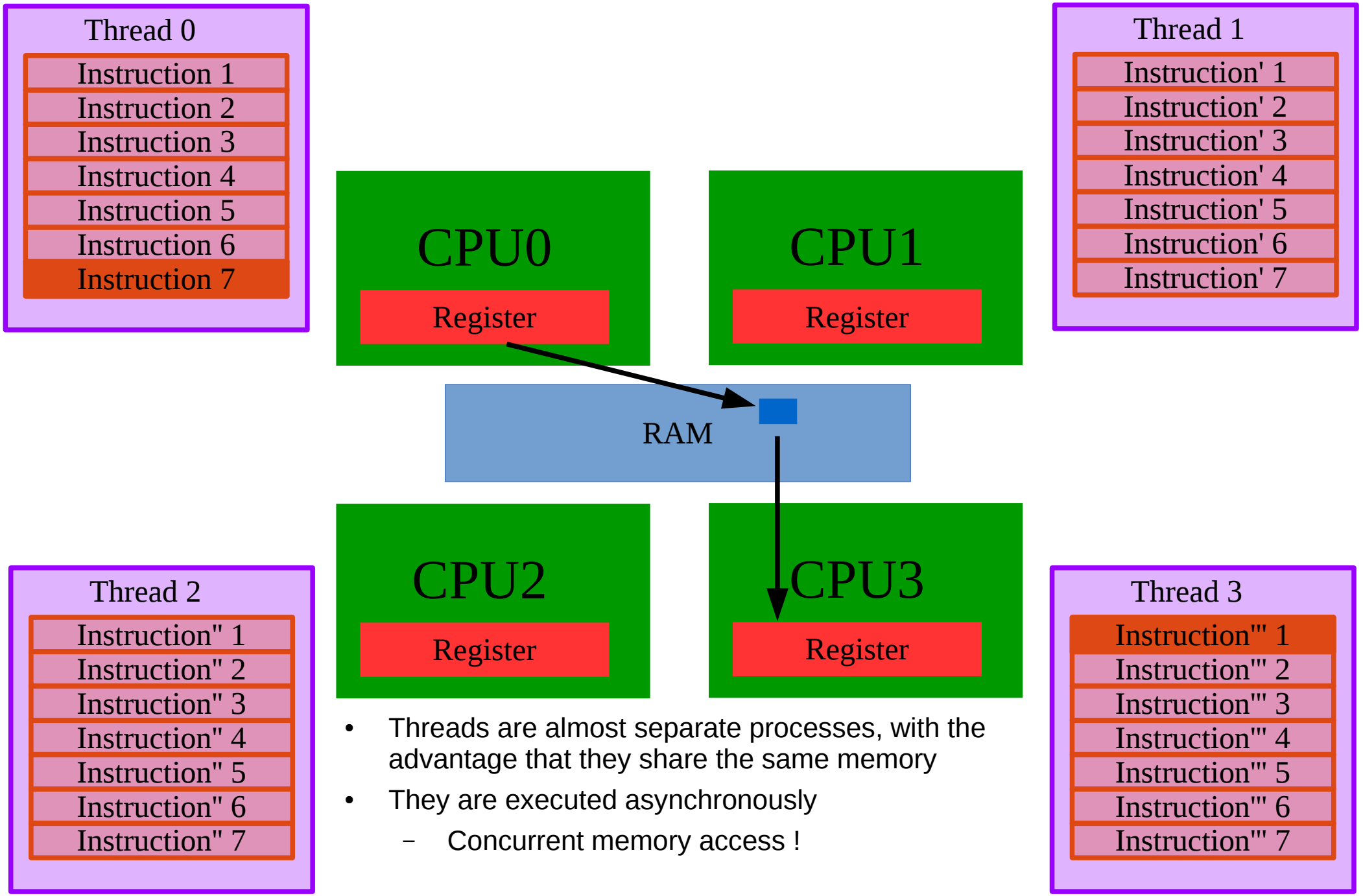
- Threads are almost separate processes, with the advantage that they share the same memory
- They are executed asynchronously
  - Concurrent memory access !

Multithreading : The limits



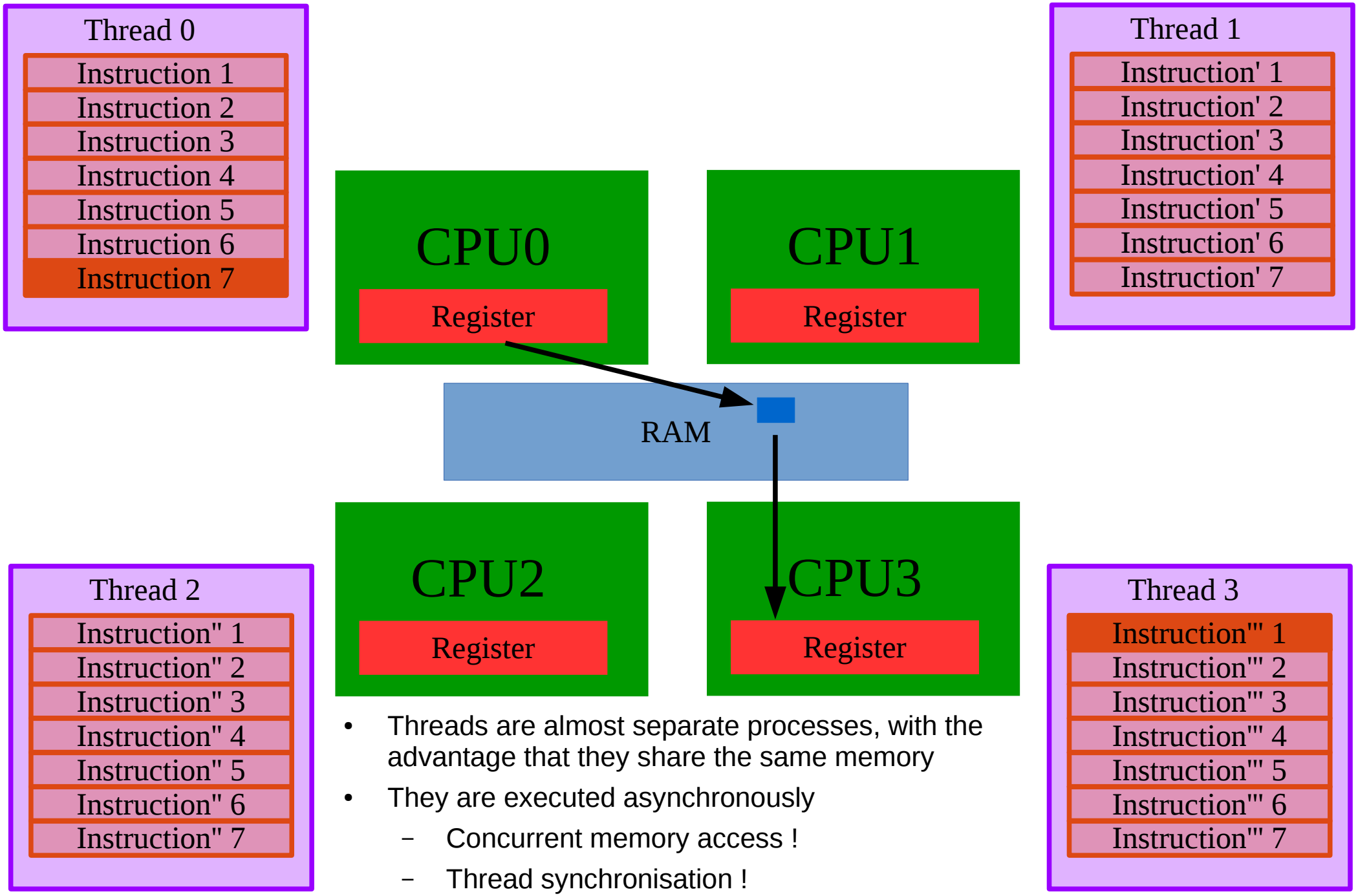
- Threads are almost separate processes, with the advantage that they share the same memory
- They are executed asynchronously
  - Concurrent memory access !

Multithreading : The limits



- Threads are almost separate processes, with the advantage that they share the same memory
- They are executed asynchronously
  - Concurrent memory access !

Multithreading : The limits



- Threads are almost separate processes, with the advantage that they share the same memory
- They are executed asynchronously
  - Concurrent memory access !
  - Thread synchronisation !

Talk is cheap , show me the code

- Linus Torvalds

Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

- Initialisation of a set of particles with random initial positions and transverse momentum

Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters

Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)

Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)

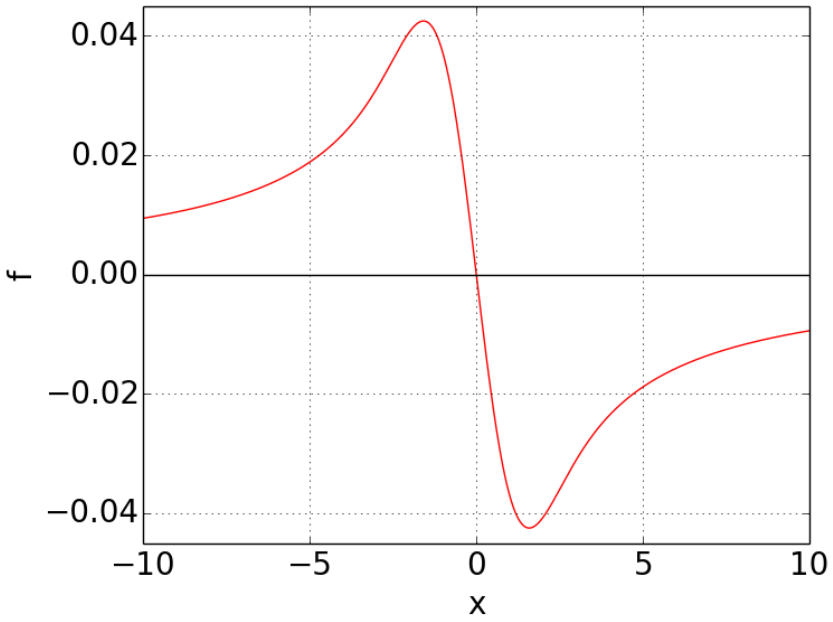
$$\begin{pmatrix} x_{t+1} \\ x'_{t+1} \end{pmatrix} = \begin{pmatrix} \cos(\Phi) & \sin(\Phi) \\ -\sin(\Phi) & \cos(\Phi) \end{pmatrix} \cdot \begin{pmatrix} x_t \\ x'_t \end{pmatrix}$$

Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)

$$\begin{pmatrix} x_{t+1} \\ x'_{t+1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ f & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\Phi) & \sin(\Phi) \\ -\sin(\Phi) & \cos(\Phi) \end{pmatrix} \cdot \begin{pmatrix} x_t \\ x'_t \end{pmatrix}$$



Example : non-linear dynamics in a collider

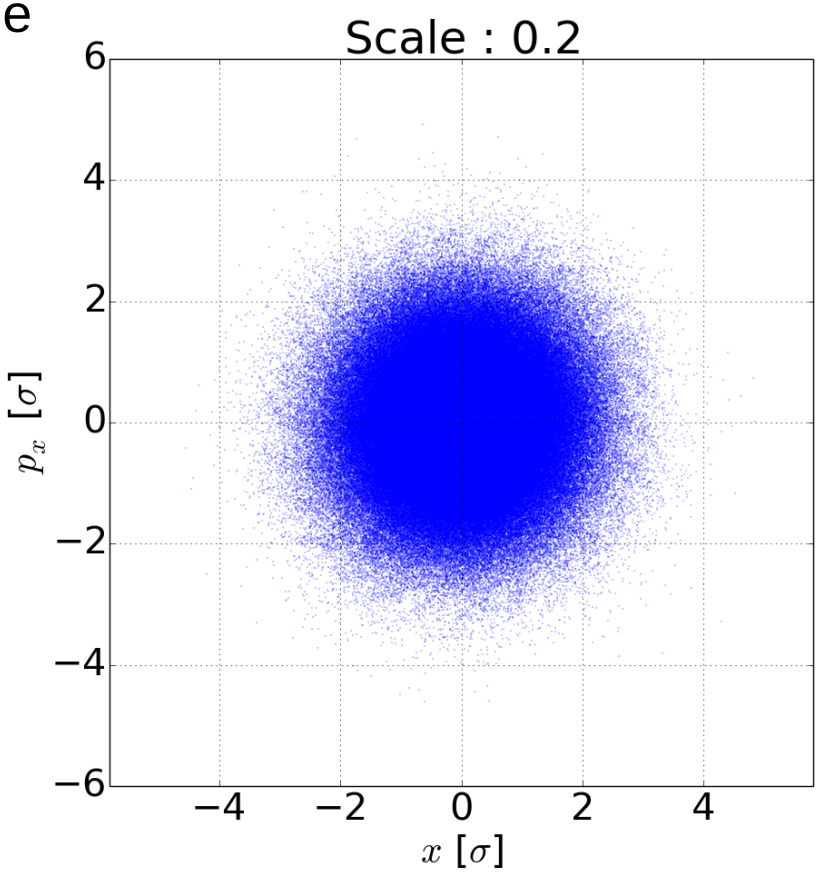
```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)
- After tracking, writing the coordinates of all particles into a file

Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

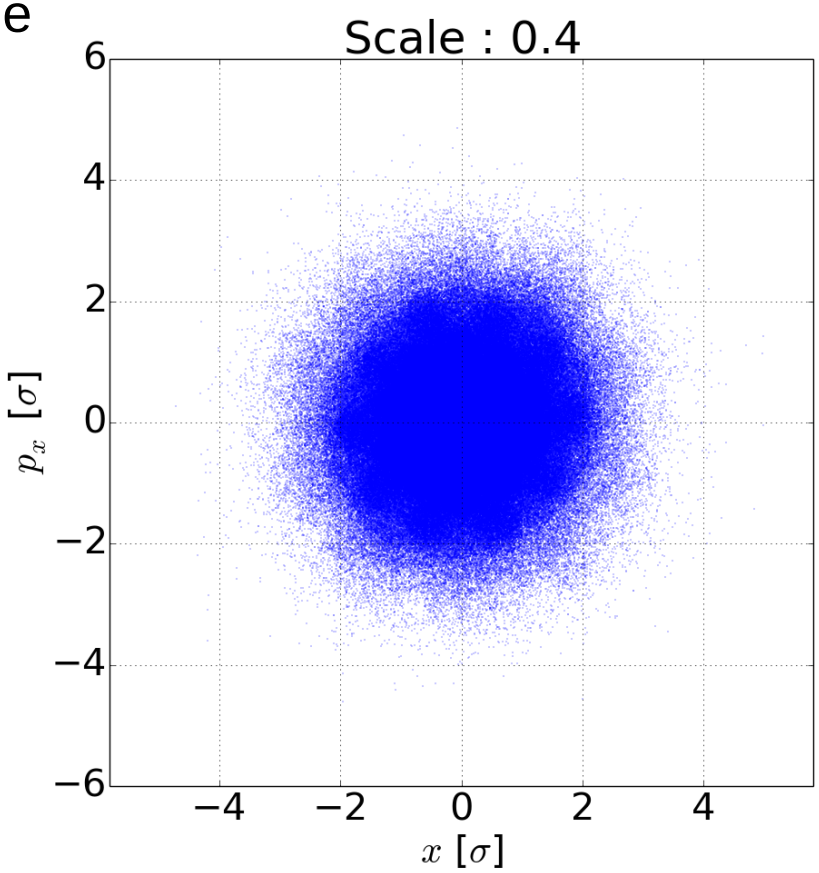
- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)
- After tracking, writing the coordinates of all particles into a file



Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

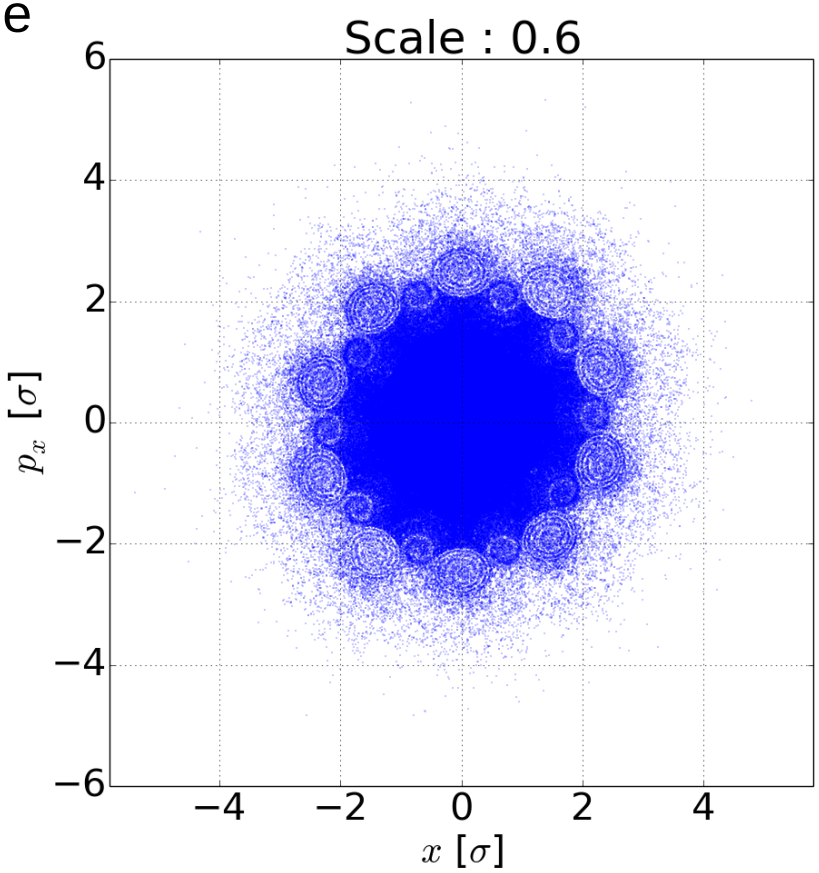
- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)
- After tracking, writing the coordinates of all particles into a file



Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

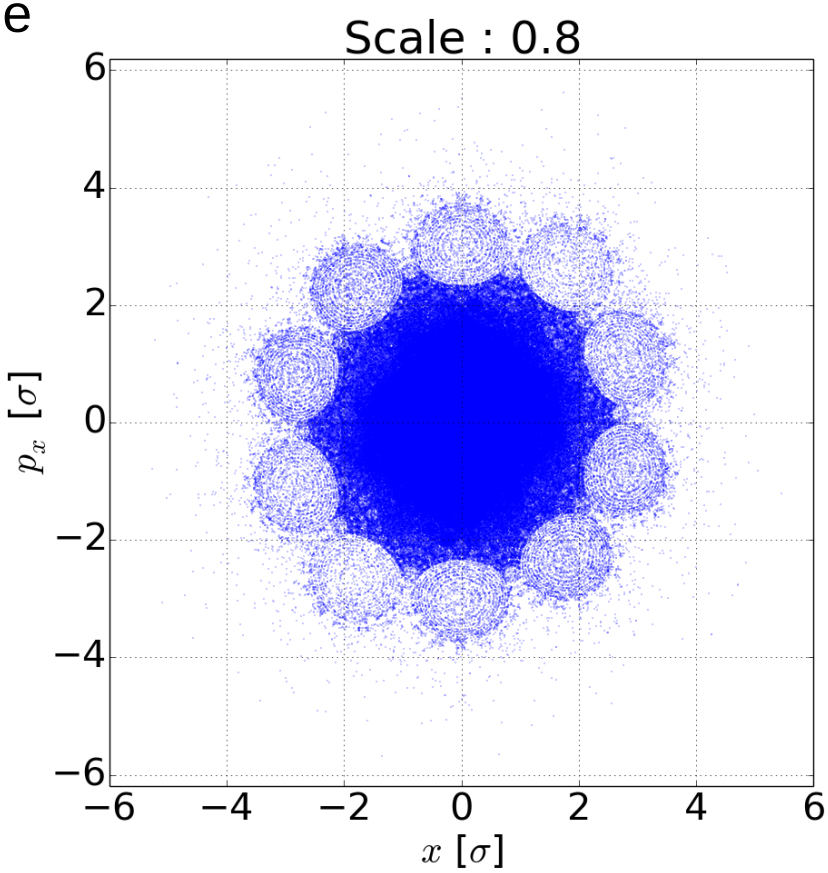
- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)
- After tracking, writing the coordinates of all particles into a file



Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

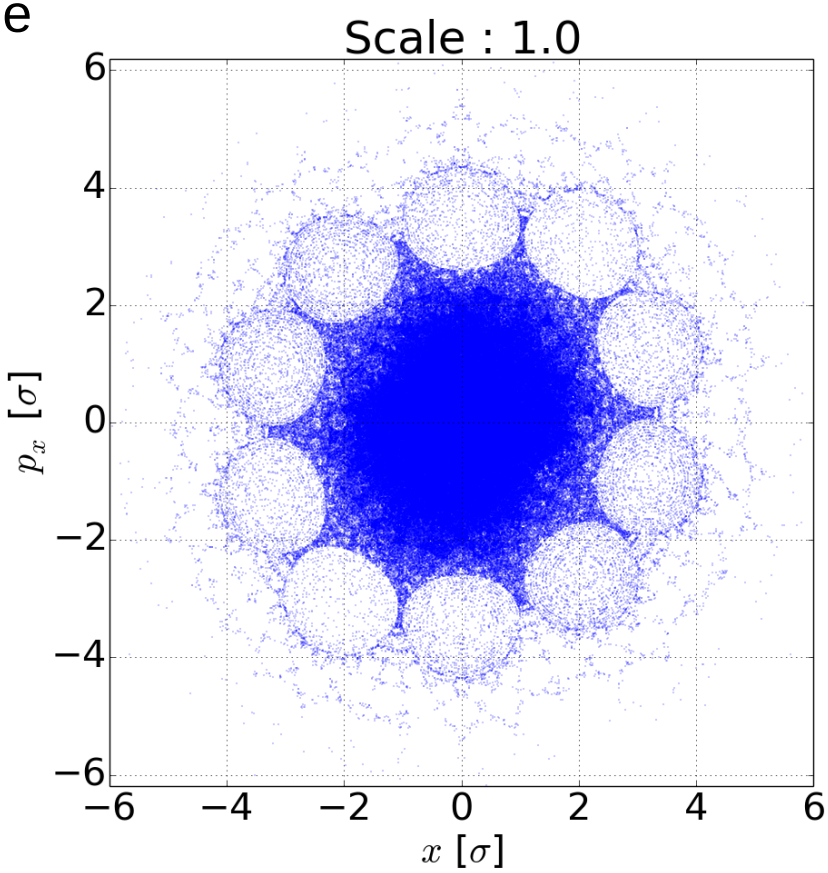
- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)
- After tracking, writing the coordinates of all particles into a file



Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

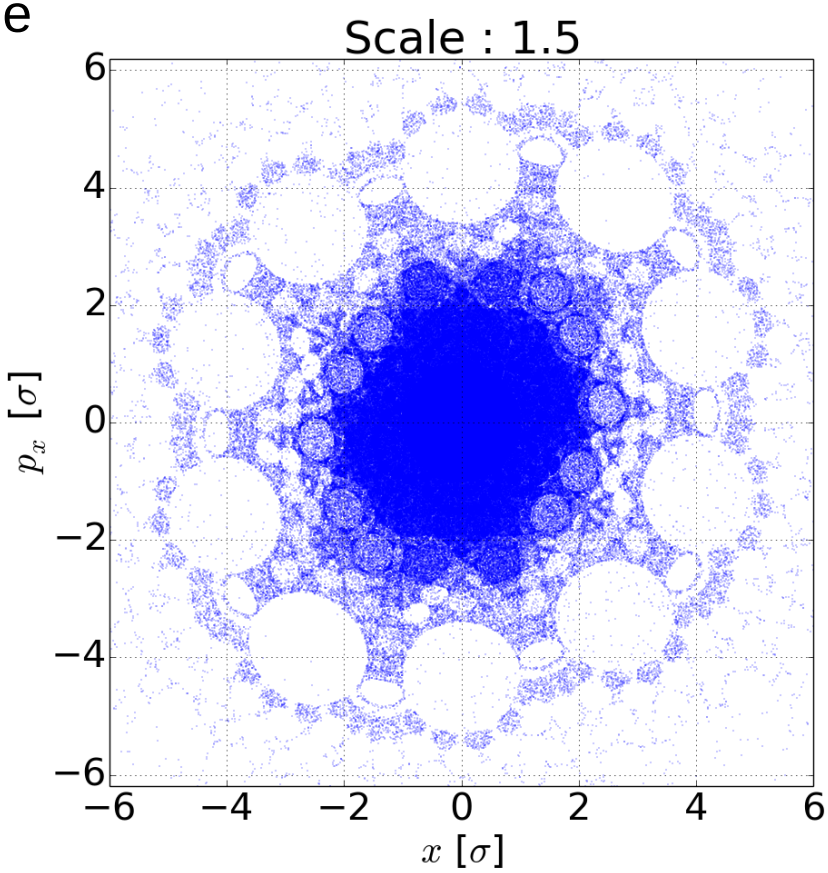
- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)
- After tracking, writing the coordinates of all particles into a file



Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

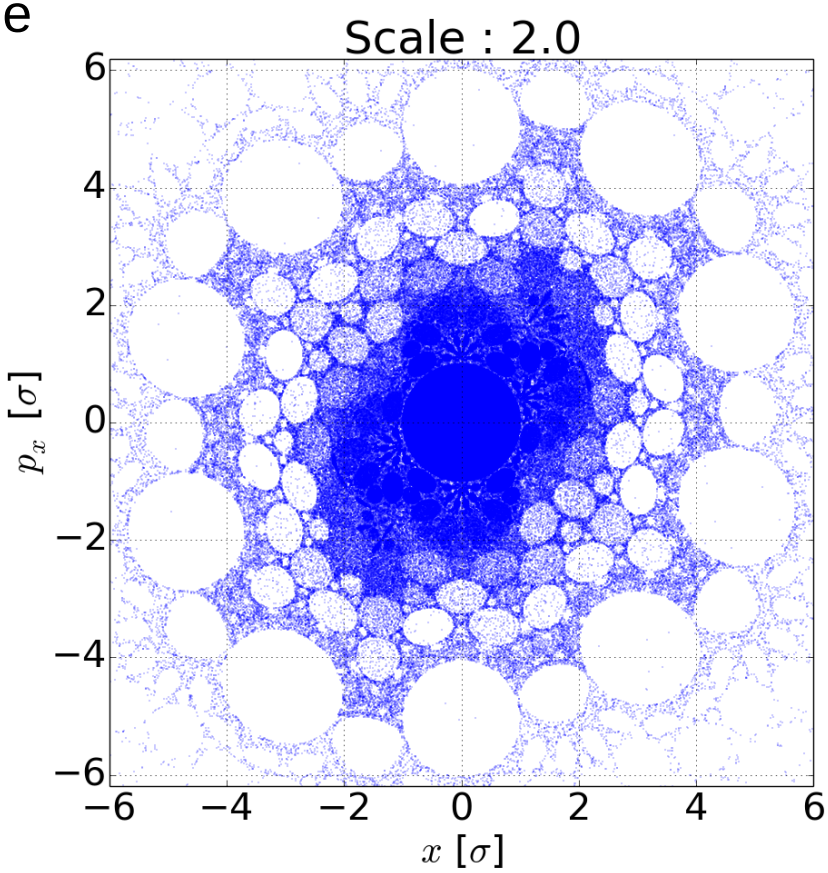
- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)
- After tracking, writing the coordinates of all particles into a file



Example : non-linear dynamics in a collider

```
#include<math.h>
#include<cstdio>
#include <sys/time.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
    const int nPart = 500000;
    double x[nPart];
    double px[nPart];
    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_PI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }

    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<nPart;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }

    FILE* file = fopen("phaseSpace.csv","w");
    for(int i=0;i<nPart;++i){
        fprintf(file,"%E,%E\n",x[i],px[i]);
    }
    fclose(file);
}
```

- Initialisation of a set of particles with random initial positions and transverse momentum
- Initialisation of tracking parameters
- Tracking through a one turn matrix and a non-linear element (here : beam-beam interaction)
- After tracking, writing the coordinates of all particles into a file



Multithreading : The hard way

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>
#include <pthread.h>

const int nPart = 500000;
double x[nPart];
double px[nPart];

int nTurn = 100;
double sinPhix = sin(2.0*M_PI*0.31);
double cosPhix = cos(2.0*M_PI*0.31);
double scale = 1.0;
double thres = 1E-10;

int main(int argc, char *argv[]){

    double radius;
    double angle;
    for(int i = 0;i<nPart;++i){
        radius = (double)rand() / RAND_MAX;
        while(radius==0){
            radius = rand();
        }
        radius = sqrt(-2.0*log(radius));
        angle = (double)2.0*M_MPI*rand()/RAND_MAX;
        x[i] = radius*sin(angle);
        px[i] = radius*cos(angle);
    }
...
```

- In order to profit from the shared memory, the variables are now declared as global
- The rest of the initialisation remains identical

Multithreading : The hard way

```
…

    pthread_t thread1;
    pthread_t thread2;
    pthread_t thread3;
    pthread_t thread4;
    int indices0[2] = {0,100000};
    int indices1[2] = {400000,500000};
    int indices2[2] = {100000,200000};
    int indices3[2] = {200000,300000};
    int indices4[2] = {300000,400000};

    pthread_create(&thread1, NULL, track, indices1);
    pthread_create(&thread2, NULL, track, indices2);
    pthread_create(&thread3, NULL, track, indices3);
    pthread_create(&thread4, NULL, track, indices4);
    track(indices0);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    pthread_join(thread4, NULL);

...
```

Multithreading : The hard way

```
…

    pthread_t thread1;
    pthread_t thread2;
    pthread_t thread3;
    pthread_t thread4;
    int indices0[2] = {0,100000};
    int indices1[2] = {400000,500000};
    int indices2[2] = {100000,200000};
    int indices3[2] = {200000,300000};
    int indices4[2] = {300000,400000};

    pthread_create(&thread1, NULL, track, indices1);
    pthread_create(&thread2, NULL, track, indices2);
    pthread_create(&thread3, NULL, track, indices3);
    pthread_create(&thread4, NULL, track, indices4);
    track(indices0);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    pthread_join(thread4, NULL);

...
```

- Preparing 4 POSIX threads, that will share the work load of the loop over the particles
  - The process running this code is 'thread 0'

Multithreading : The hard way

```
…

    pthread_t thread1;
    pthread_t thread2;
    pthread_t thread3;
    pthread_t thread4;
    int indices0[2] = {0,100000};
    int indices1[2] = {400000,500000};
    int indices2[2] = {100000,200000};
    int indices3[2] = {200000,300000};
    int indices4[2] = {300000,400000};

    pthread_create(&thread1, NULL, track, indices1);
    pthread_create(&thread2, NULL, track, indices2);
    pthread_create(&thread3, NULL, track, indices3);
    pthread_create(&thread4, NULL, track, indices4);
    track(indices0);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    pthread_join(thread4, NULL);

...
```

- Preparing 4 POSIX threads, that will share the work load of the loop over the particles
  - The process running this code is 'thread 0'
- Creating the 4 threads, that will execute simultaneously (but independantly) on the different CPUs, executing the function 'track' with different arguments
  - The thread 0 also, does its share

Multithreading : The hard way

```
…

    pthread_t thread1;
    pthread_t thread2;
    pthread_t thread3;
    pthread_t thread4;
    int indices0[2] = {0,100000};
    int indices1[2] = {400000,500000};
    int indices2[2] = {100000,200000};
    int indices3[2] = {200000,300000};
    int indices4[2] = {300000,400000};

    pthread_create(&thread1, NULL, track, indices1);
    pthread_create(&thread2, NULL, track, indices2);
    pthread_create(&thread3, NULL, track, indices3);
    pthread_create(&thread4, NULL, track, indices4);
    track(indices0);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    pthread_join(thread3, NULL);
    pthread_join(thread4, NULL);

...
```

- Preparing 4 POSIX threads, that will share the work load of the loop over the particles
  - The process running this code is 'thread 0'
- Creating the 4 threads, that will execute simultaneously (but independantly) on the different CPUs, executing the function 'track' with different arguments
  - The thread 0 also, does its share
- The join statement indicates the stops the execution of the thread 0, until the others are finished

→ Synchronisation

Multithreading : The hard way

```
void* track(void* indices_ptr){
    int* indices = (int*) indices_ptr;
    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = indices[0];i<indices[1];++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;

            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }


…
  int indices1[2] = {400000,500000};
  Pthread_create(&thread1, NULL, track, indices1);
…
```

Multithreading : The hard way

```
void* track(void* indices_ptr){
   int* indices = (int*) indices_ptr;
   double oldX = 0.0;
   double oldPx = 0.0;
   for(int turn = 0;turn<nTurn;++turn){
      for(int i = indices[0];i<indices[1];++i){
         oldX = x[i];
         oldPx = px[i];
         x[i] = cosPhix*oldX + sinPhix*oldPx;
         px[i] = -sinPhix*oldX + cosPhix*oldPx;

         if(abs(x[i]) > thres){
            px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
         }
      }
   }


…
 int indices1[2] = {400000,500000};
 Pthread_create(&thread1, NULL, track, indices1);
…
```

- The function code hasn't changes, except that the execution is performed only between the indices given in argument

Global variables, pointer to functions, type casting, complicated syntax, are you sure about this ?

- Fuego, a rabbit that is unsure about this

Multithreading : The easy way

- OpenMP allows, with a single line of code (so-called a pragma) to tell the compiler to share the load of a loops other multiple threads
  - The compiler needs to know which part of the memory is shared, which part should be allocated for each thread

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>

int main(int argc, char *argv[]){

Initialisation
Computing
Output

}
```

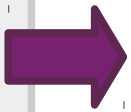
```
double oldX = 0.0;
double oldPx = 0.0;
for(int turn = 0;turn<nTurn;++turn){
    #pragma omp parallel for default(firstprivate) shared(x,px) \
    schedule(guided,1000)
   for(int i = 0;i<nPart;++i){
      oldX = x[i];
      oldPx = px[i];
      x[i] = cosPhix*oldX + sinPhix*oldPx;
      px[i] = -sinPhix*oldX + cosPhix*oldPx;
      if(abs(x[i]) > thres){
         px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
      }
   }
}
```

→ Compiler option -fopenmp

Multithreading : The easy way

- OpenMP allows, with a single line of code (so-called a pragma) to tell the compiler to share the load of a loops other multiple threads
  - The compiler needs to know which part of the memory is shared, which part should be allocated for each thread

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>

int main(int argc, char *argv[]){

    Initialisation
    Computing
    Output

}
```



```
double oldX = 0.0;
double oldPx = 0.0;
for(int turn = 0;turn<nTurn;++turn){
    #pragma omp parallel for default(firstprivate) shared(x,px) \
    schedule(guided,1000)
   for(int i = 0;i<nPart;++i){
      oldX = x[i];
      oldPx = px[i];
      x[i] = cosPhix*oldX + sinPhix*oldPx;
      px[i] = -sinPhix*oldX + cosPhix*oldPx;
      if(abs(x[i]) > thres){
         px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
      }
   }
}
```

→ Compiler option -fopenmp

Multithreading : The easy way

- OpenMP allows, with a single line of code (so-called a pragma) to tell the compiler to share the load of a loops other multiple threads
  - The compiler needs to know which part of the memory is shared, which part should be allocated for each thread

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>

int main(int argc, char *argv[]){

    Initialisation
    Computing
    Output

}
```

By default each threads
gets a copy of the variable

```
double oldX = 0.0;
double oldPx = 0.0;
for(int turn = 0;turn<nTurn;++turn){
    #pragma omp parallel for default(firstprivate) shared(x,px) \
    schedule(guided,1000)
   for(int i = 0;i<nPart;++i){
      oldX = x[i];
      oldPx = px[i];
      x[i] = cosPhix*oldX + sinPhix*oldPx;
      px[i] = -sinPhix*oldX + cosPhix*oldPx;
      if(abs(x[i]) > thres){
         px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
      }
   }
}
```

→ Compiler option -fopenmp

Multithreading : The easy way

- OpenMP allows, with a single line of code (so-called a pragma) to tell the compiler to share the load of a loops other multiple threads
  - The compiler needs to know which part of the memory is shared, which part should be allocated for each thread

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>

int main(int argc, char *argv[]){

    Initialisation
    Computing
    Output

}
```

By default each threads gets a copy of the variable

Only x and px are not copied, they remain shared

```
double oldX = 0.0;
double oldPx = 0.0;
for(int turn = 0;turn<nTurn;++turn){
    #pragma omp parallel for default(firstprivate) shared(x,px) \
    schedule(guided,1000)
   for(int i = 0;i<nPart;++i){
      oldX = x[i];
      oldPx = px[i];
      x[i] = cosPhix*oldX + sinPhix*oldPx;
      px[i] = -sinPhix*oldX + cosPhix*oldPx;
      if(abs(x[i]) > thres){
         px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
      }
   }
}
```

→ Compiler option -fopenmp

Multithreading : The easy way

- OpenMP allows, with a single line of code (so-called a pragma) to tell the compiler to share the load of a loops other multiple threads
  - The compiler needs to know which part of the memory is shared, which part should be allocated for each thread

→ A large variety of simple tools exists for various applications. You may gain factors in execution time !

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>

int main(int argc, char *argv[]){

    Initialisation
    Computing
    Output

}
```

By default each threads gets a copy of the variable

Only x and px are not copied, they remain shared

```
double oldX = 0.0;
double oldPx = 0.0;
for(int turn = 0;turn<nTurn;++turn){
    #pragma omp parallel for default(firstprivate) shared(x,px) \
    schedule(guided,1000)
   for(int i = 0;i<nPart;++i){
      oldX = x[i];
      oldPx = px[i];
      x[i] = cosPhix*oldX + sinPhix*oldPx;
      px[i] = -sinPhix*oldX + cosPhix*oldPx;
      if(abs(x[i]) > thres){
         px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
      }
   }
}
```

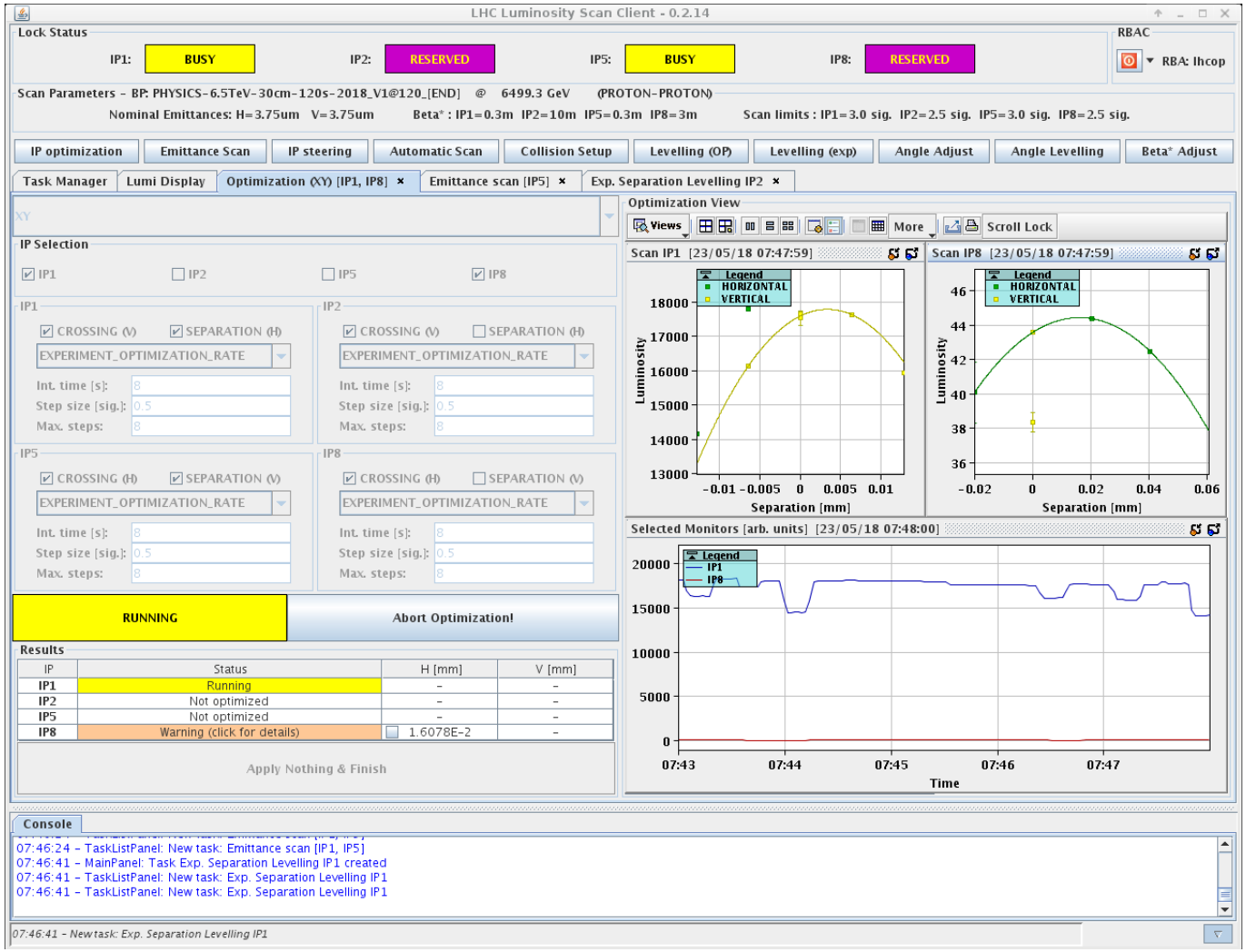
→ Compiler option -fopenmp

Heterogenous multithreading

- Pthreads can be extremely powerfull for heterogenous tasks, but is clearly a bad choice for our example application
- Multithreading comes in particularly handy for example when programming software for control systems

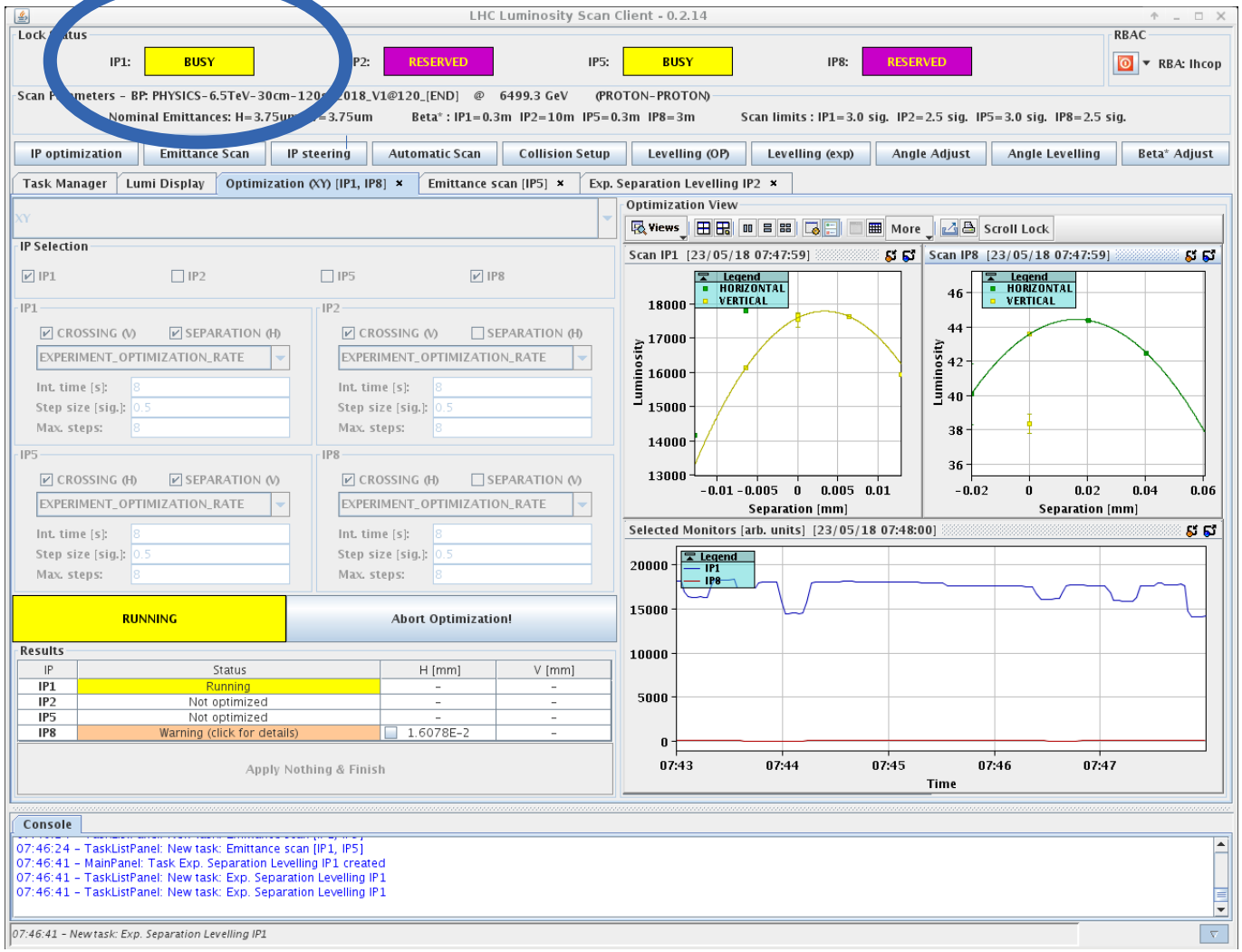
Heterogenous multithreading

- Pthreads can be extremely powerfull for heterogenous tasks, but is clearly a bad choice for our example application
- Multithreading comes in particularly handy for example when programming software for control systems



Heterogenous multithreading

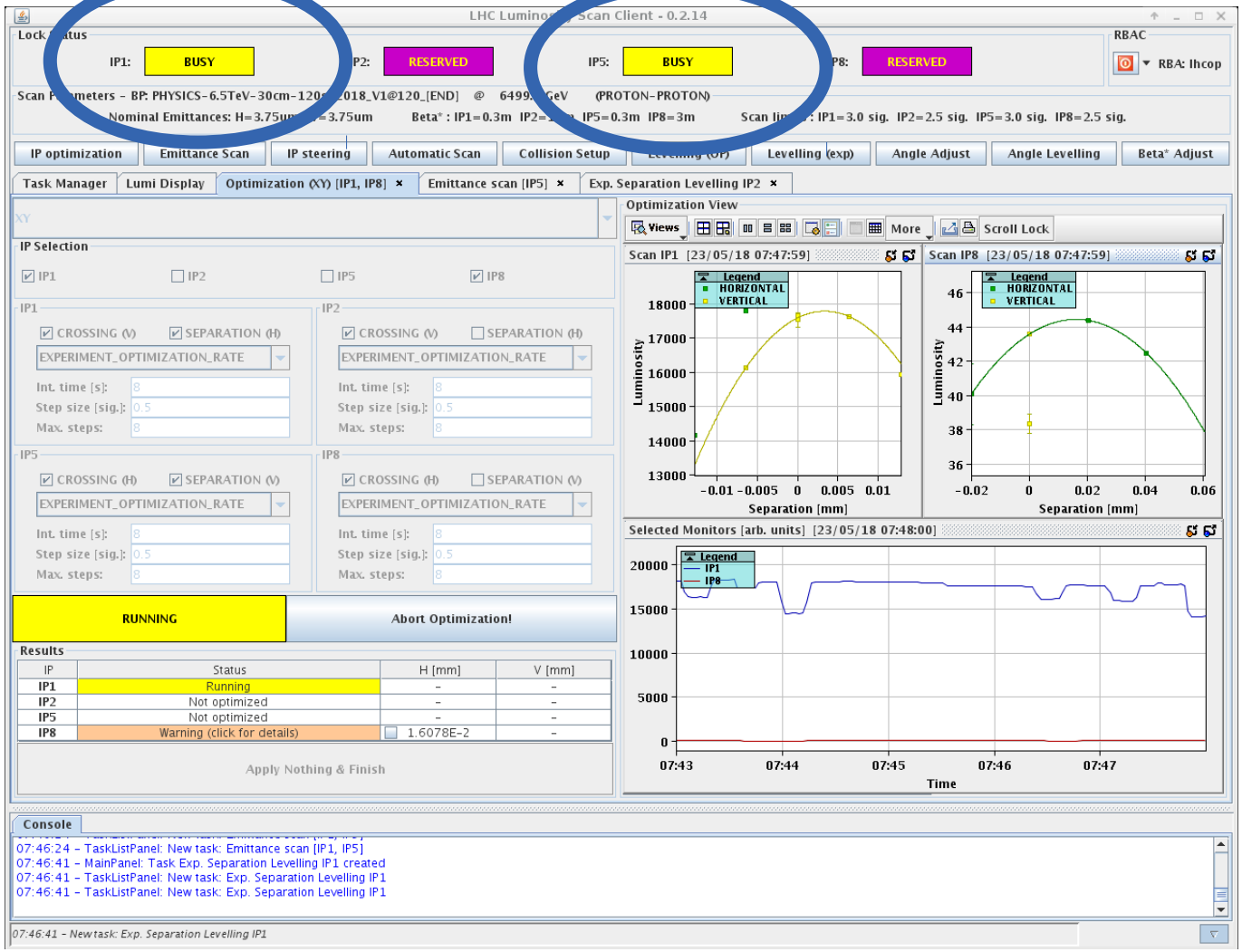
- Pthreads can be extremely powerfull for heterogenous tasks, but is clearly a bad choice for our example application
- Multithreading comes in particularly handy for example when programming software for control systems



- Sends instructions to move the orbit in IP1

Heterogenous multithreading

- Pthreads can be extremely powerfull for heterogenous tasks, but is clearly a bad choice for our example application
- Multithreading comes in particularly handy for example when programming software for control systems



- Sends instructions to move the orbit in IP1
- Sends instructions to move the orbit in IP5

Heterogenous multithreading

- Pthreads can be extremely powerfull for heterogenous tasks, but is clearly a bad choice for our example application
- Multithreading comes in particularly handy for example when programming software for control systems

- Sends instructions to move the orbit in IP1
- Sends instructions to move the orbit in IP5
- Receives online data from the two experiments

Heterogenous multithreading

- Pthreads can be extremely powerfull for heterogenous tasks, but is clearly a bad choice for our example application
- Multithreading comes in particularly handy for example when programming software for control systems

- Sends instructions to move the orbit in IP1
- Sends instructions to move the orbit in IP5
- Receives online data from the two experiments
- Responsive GUI, to look at the data and possibly generate new tasks

Heterogenous multithreading

- Pthreads can be extremely powerfull for heterogenous tasks, but is clearly a bad choice for our example application
- Multithreading comes in particularly handy for example when programming software for control systems

- Sends instructions to move the orbit in IP1
- Sends instructions to move the orbit in IP5
- Receives online data from the two experiments
- Responsive GUI, to look at the data and possibly generate new tasks

→ At least 5 threads running simultaneously, with completely different tasks

Heterogenous multithreading

- Pthreads can be extremely powerfull for heterogenous tasks, but is clearly a bad choice for our example application
- Multithreading comes in particularly handy for example when programming software for control systems

- Sends instructions to move the orbit in IP1
- Sends instructions to move the orbit in IP5
- Receives online data from the two experiments

  Responsive GUI, to look at the data and possibly generate new tasks

  → At least 5 threads running simultaneously, with completely different tasks
- In principle there can be more threads than CPUs, the OS has to priorities

Synchronisation, scheduling

- Multithreading is the most versatile tool, since any type of tasks can be performed in parallel

Synchronisation, scheduling

- Multithreading is the most versatile tool, since any type of tasks can be performed in parallel
  - Nevertheless, you can't cook the sauce before having cut the onions

Synchronisation, scheduling

- Multithreading is the most versatile tool, since any type of tasks can be performed in parallel
  - Nevertheless, you can't cook the sauce before having cut the onions

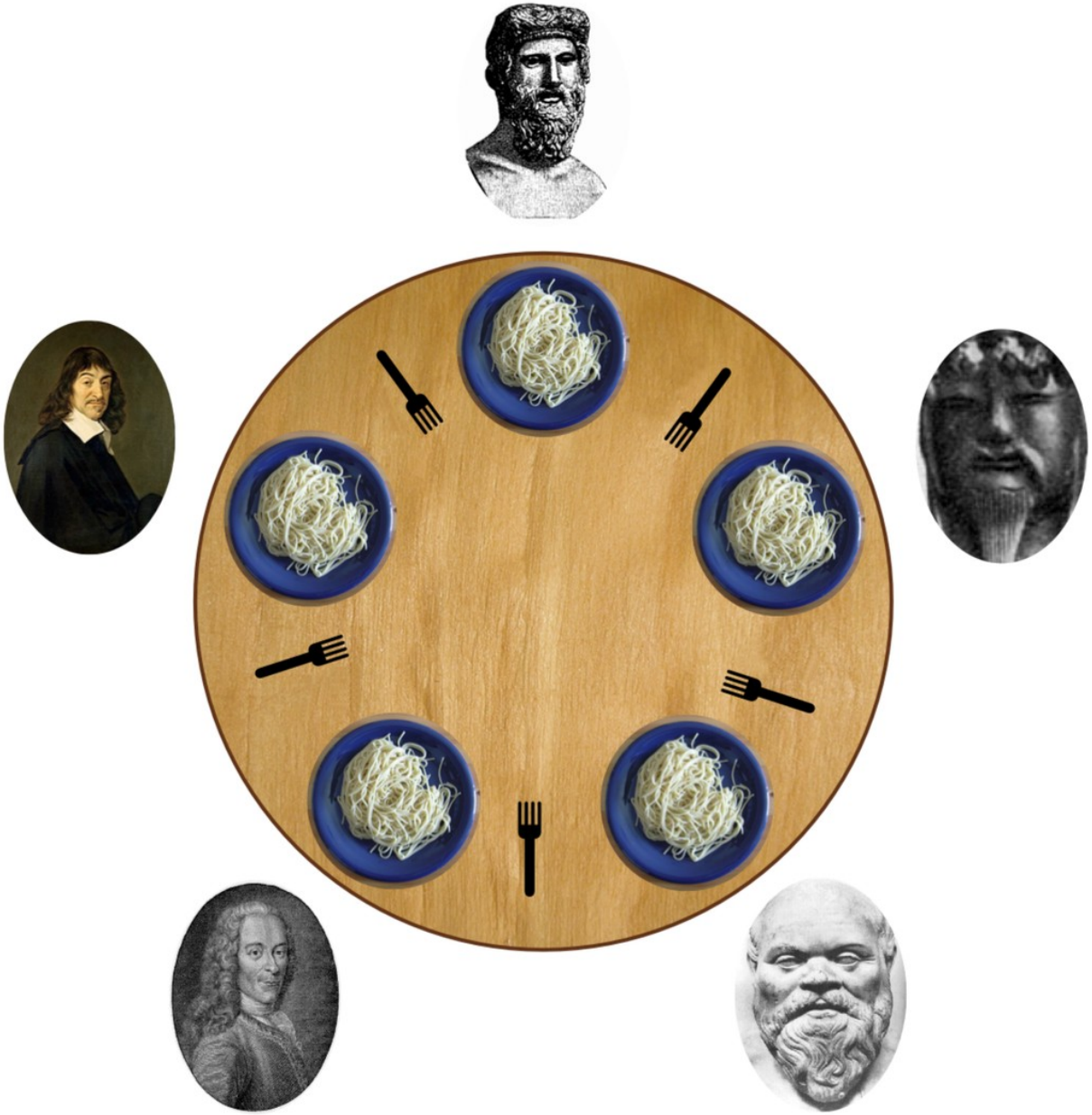
→ Like a good chef, prepare your recipe well and share the tasks accordingly!

Synchronisation, scheduling

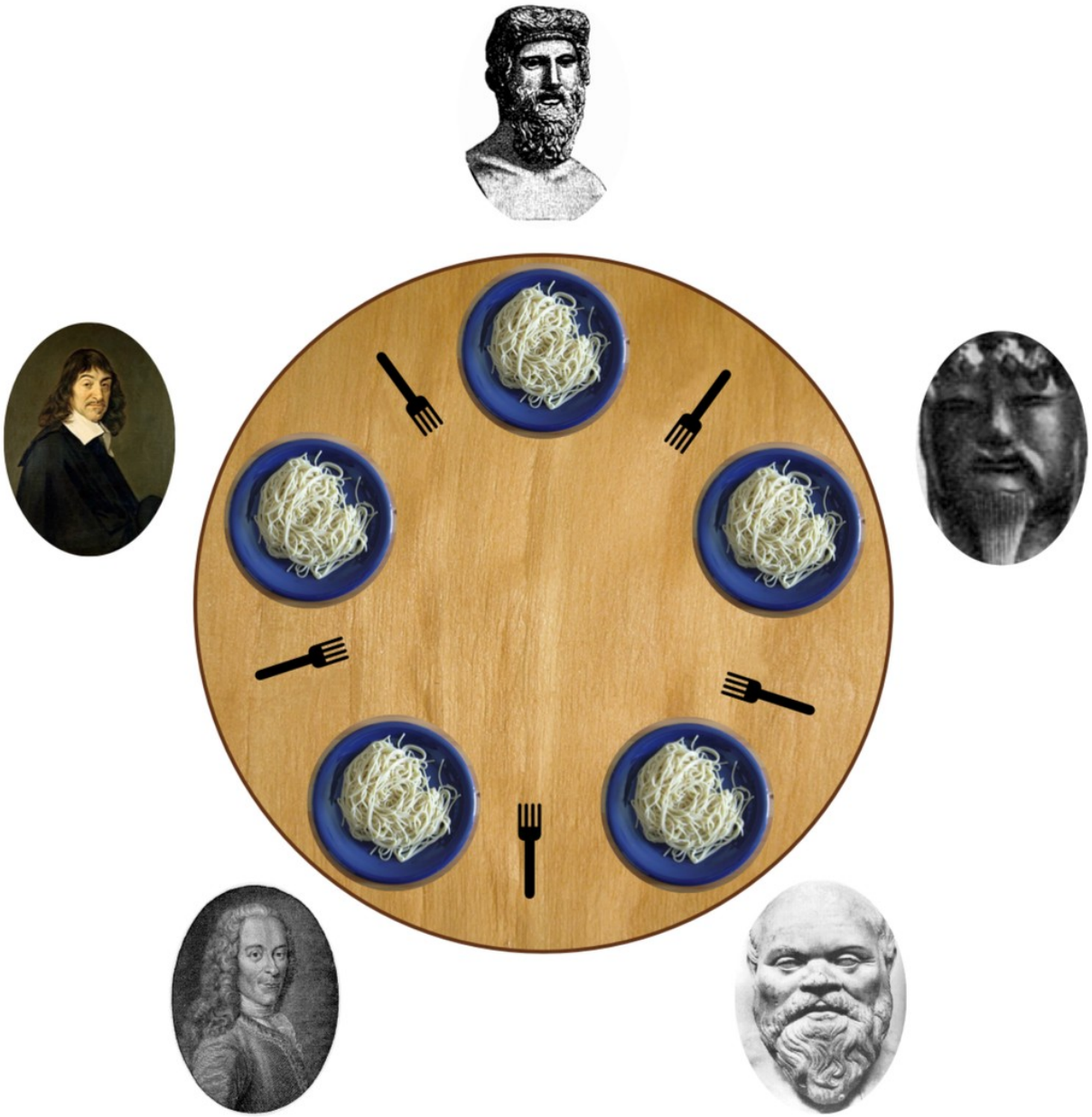
- Multithreading is the most versatile tool, since any type of tasks can be performed in parallel
  - Nevertheless, you can't cook the sauce before having cut the onions

  → Like a good chef, prepare your recipe well and share the tasks accordingly!
- What could go wrong?

More on synchronization and concurrent memory access

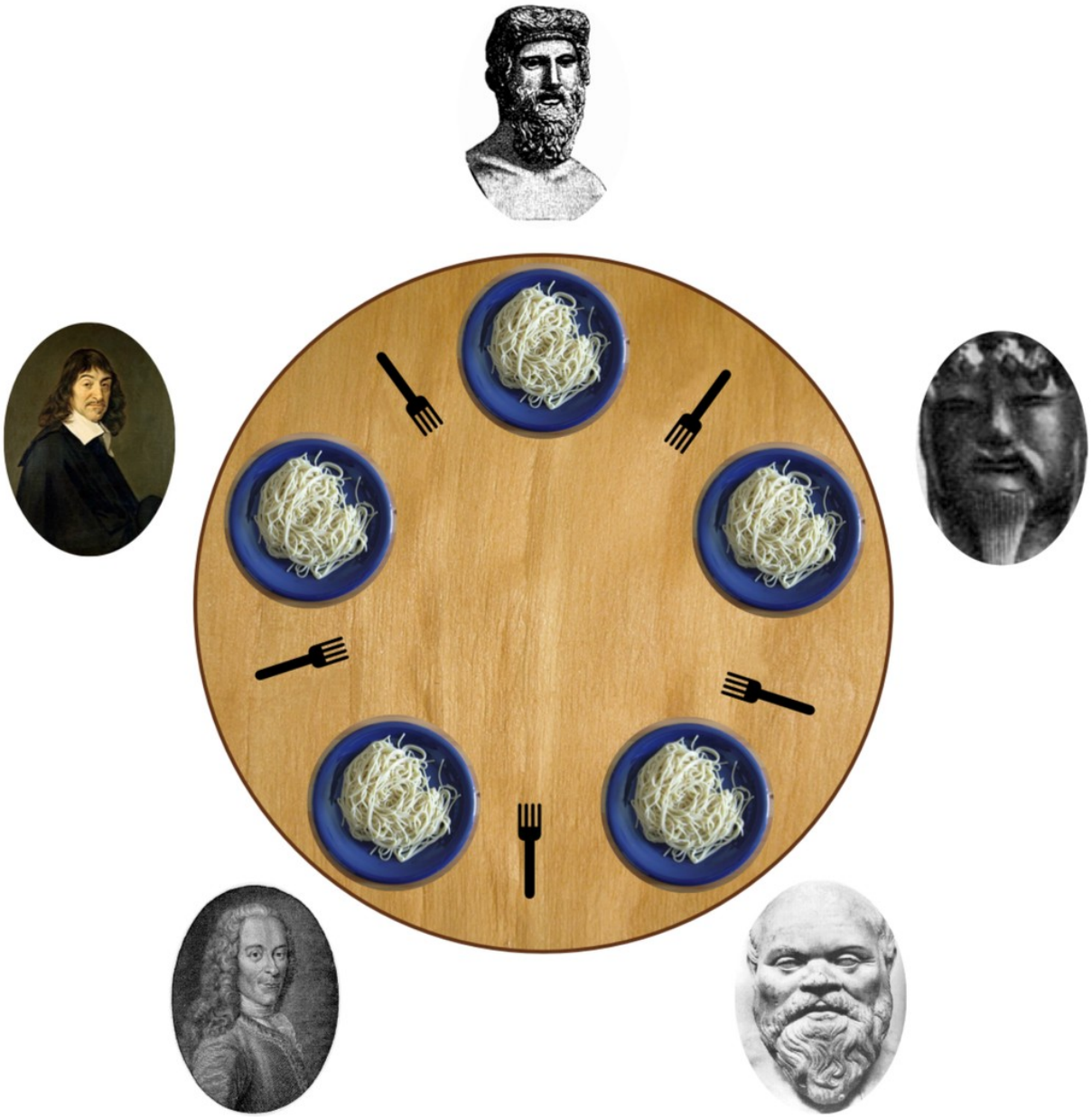


More on synchronization and concurrent memory access



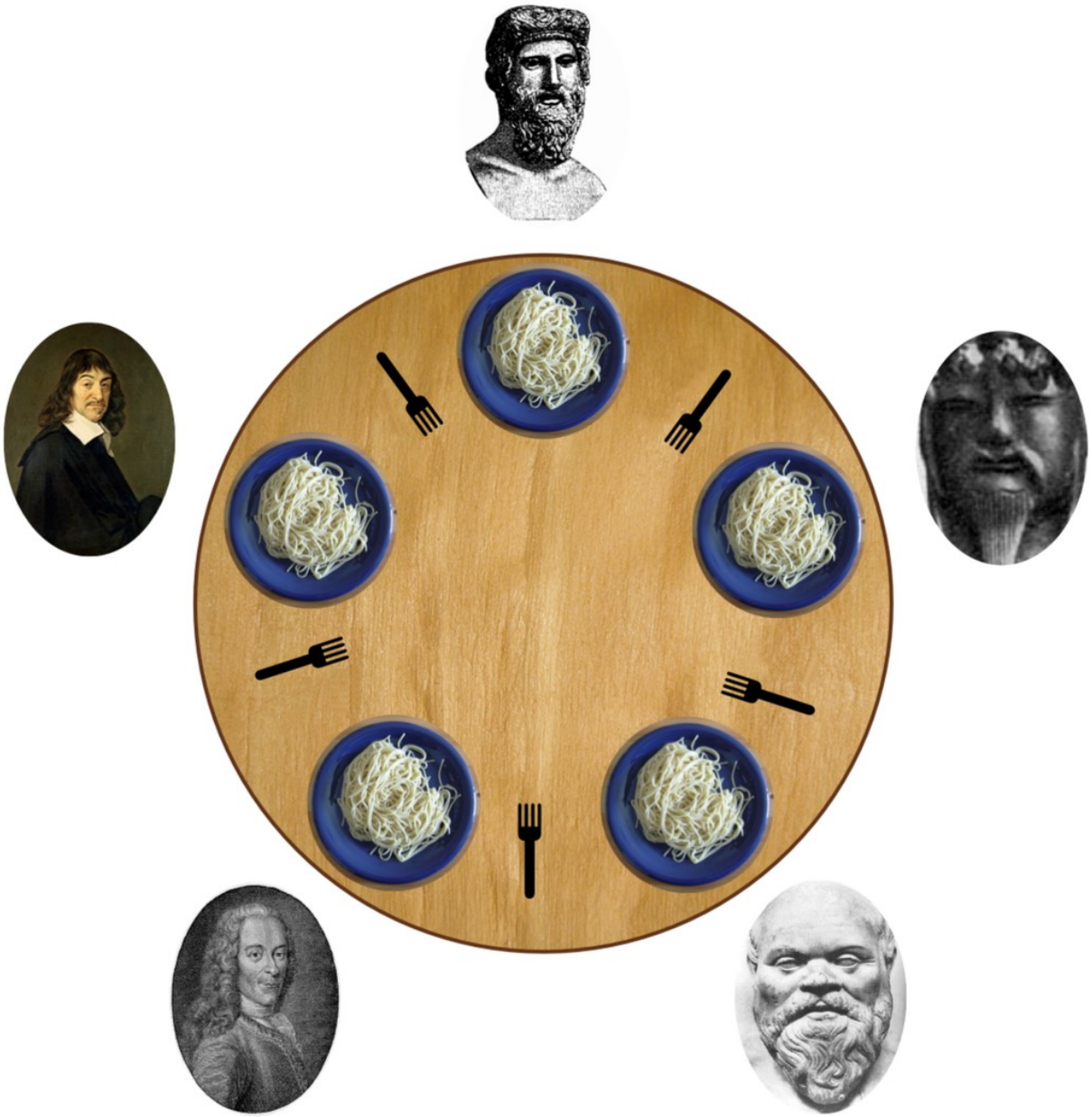
- Critical sections

More on synchronization and concurrent memory access



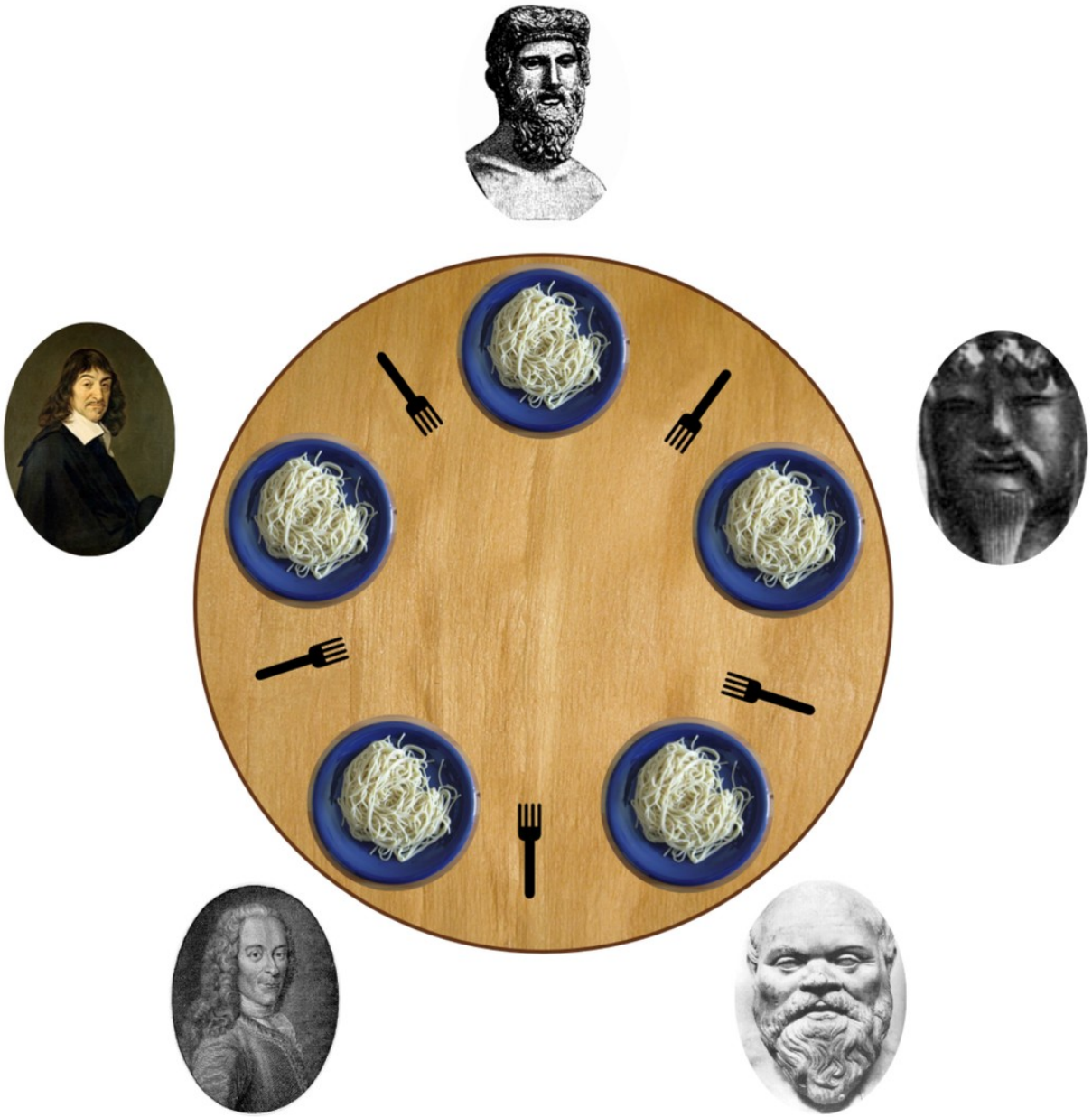
- Critical sections
- Reduction

More on synchronization and concurrent memory access



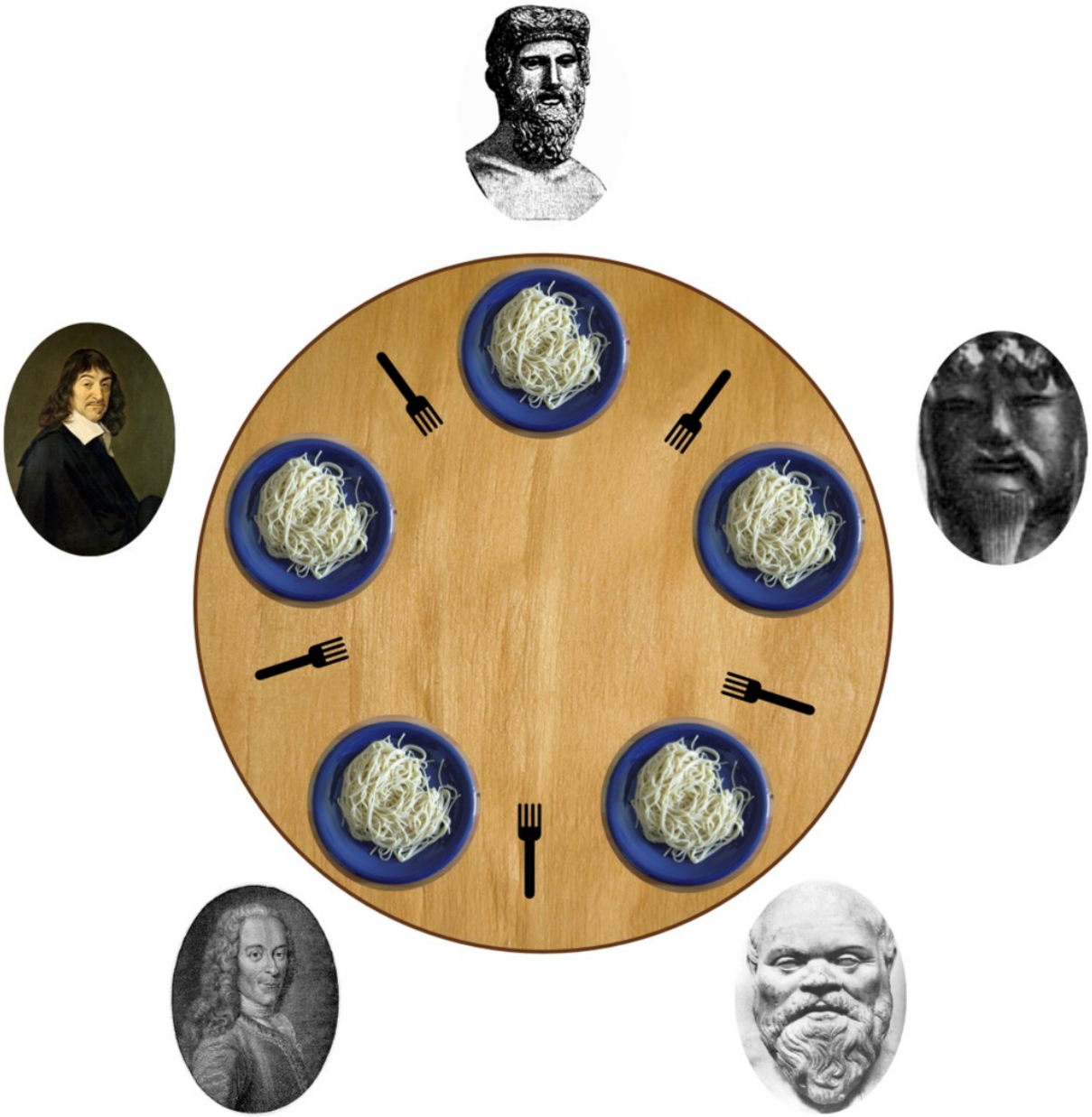
- Critical sections
- Reduction
- Atomic operations

More on synchronization and concurrent memory access



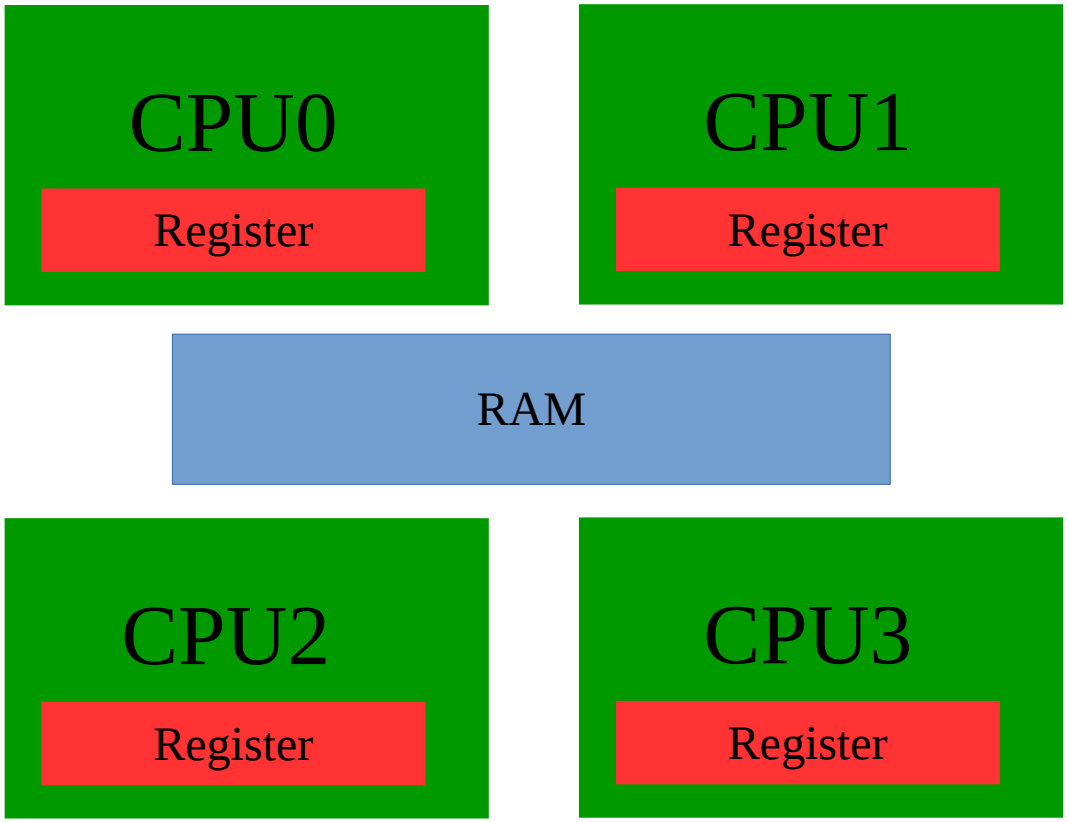
- Critical sections
- Reduction
- Atomic operations
- Locks

More on synchronization and concurrent memory access



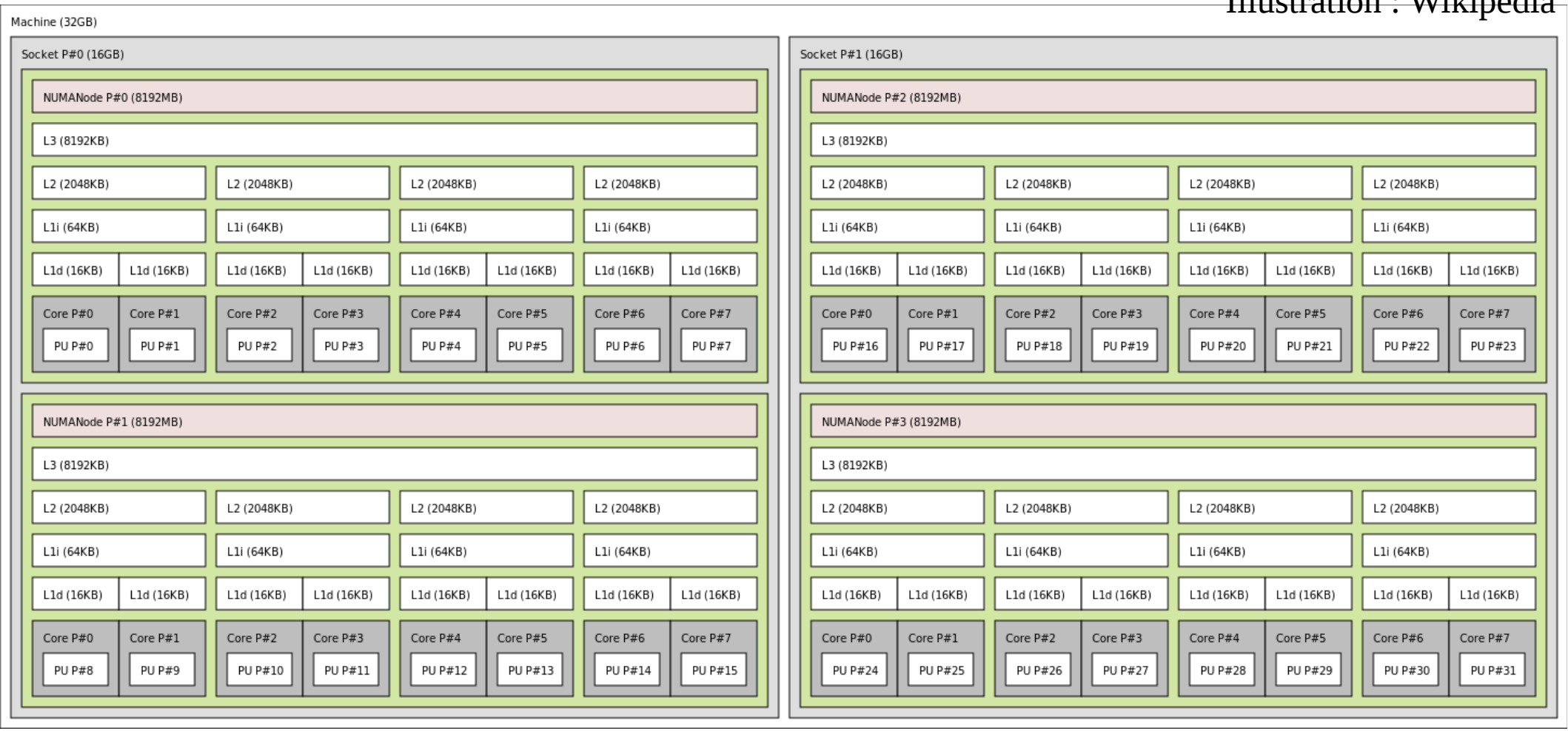
- Critical sections
- Reduction
- Atomic operations
- Locks
- ...

Non-uniform memory access (NUMA)



Non-uniform memory access (NUMA)

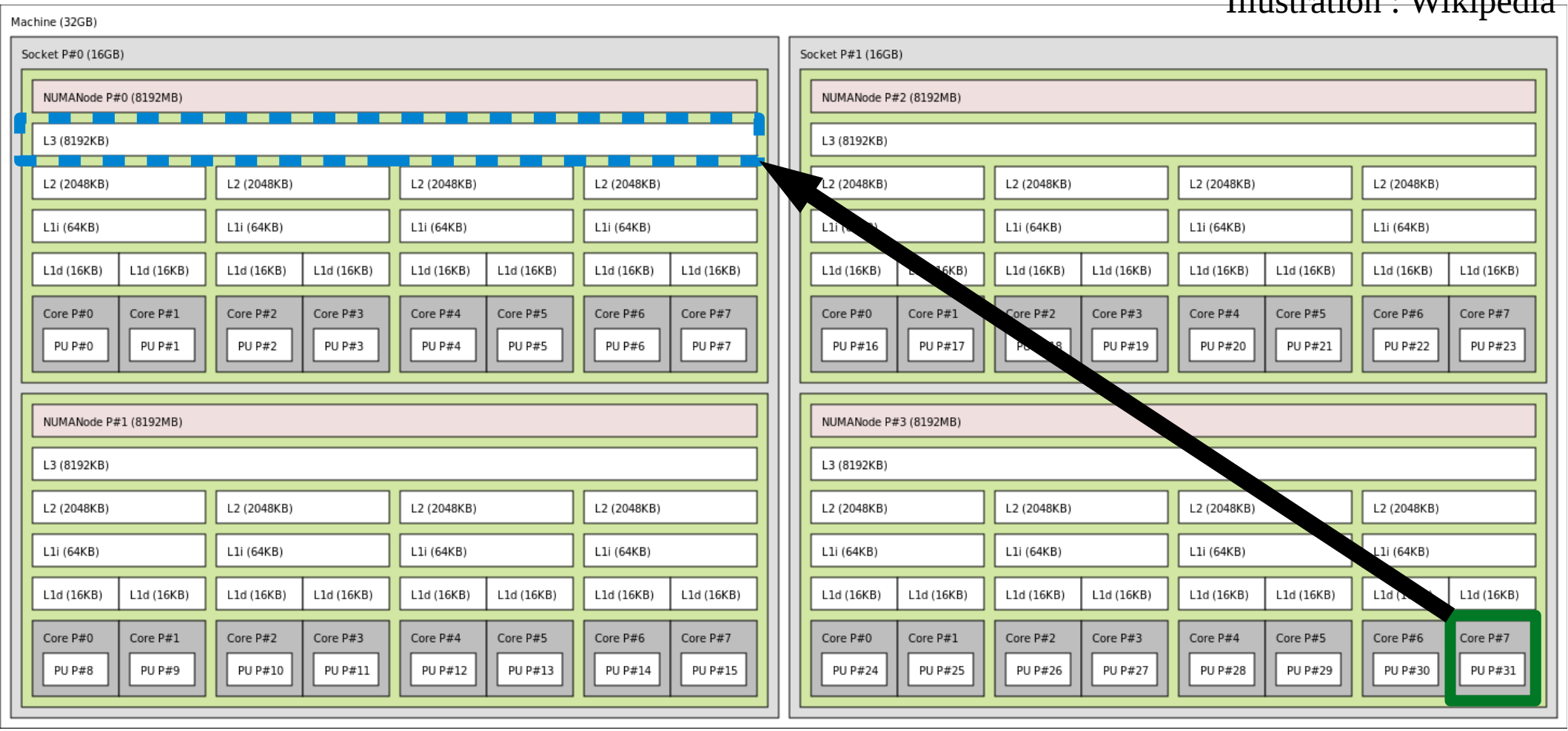
Illustration : Wikipedia



- In reality, there exists several levels of cache memory, shared between cores, sockets or nodes
  - The communication is slowest for data far from the CPU → Data locality!

Non-uniform memory access (NUMA)

Illustration : Wikipedia



- In reality, there exists several levels of cache memory, shared between cores, sockets or nodes
  - The communication is slowest for data far from the CPU → Data locality!
- Imagine x, px are allocated by core0 of socket0, in its L3 cache, the sharing of the load with core7 of socket3 will be inefficient w.r.t. the sharing with core1 of the same socket

  → Equal sharing between CPUs is usually not the most efficient → Scheduling

- Better not assign the cooking of the sauce to the waiter, since he is away from the hotplate...

Thread scheduling

```
...
   double oldX = 0.0;
   double oldPx = 0.0;
   for(int turn = 0;turn<nTurn;++turn){
      #pragma omp parallel for default(firstprivate) shared(x,px) \
      schedule(guided,1000)
      for(int i = 0;i<nPart;++i){
         oldX = x[i];
         oldPx = px[i];
         x[i] = cosPhix*oldX + sinPhix*oldPx;
         px[i] = -sinPhix*oldX + cosPhix*oldPx;
         if(x[i] > thres){
            px[i] += scale*(1.0-exp(-0.5*x[i]))/x[i];
         }
      }
   }
...
```

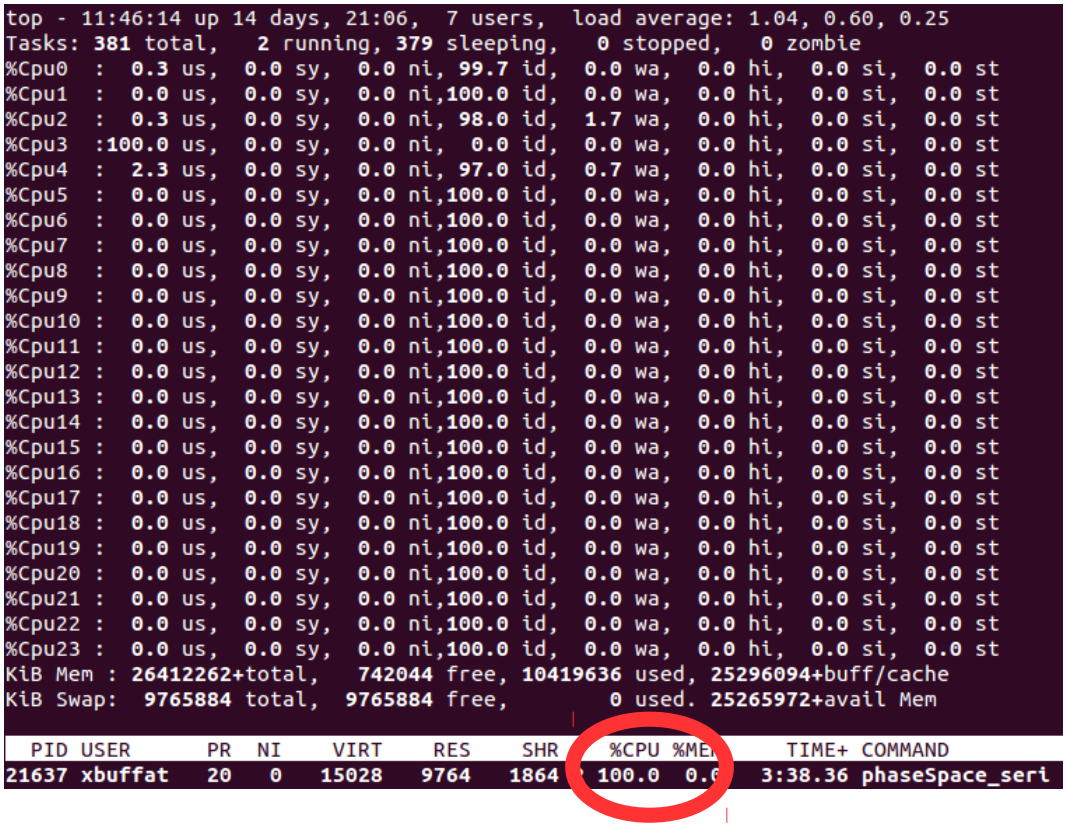
- In OpenMP, you can indicate to the compiler to chunk the load in smaller pieces and assign the chunks as the threads perform the their tasks

  → The CPU performing faster will be assigned more chunks

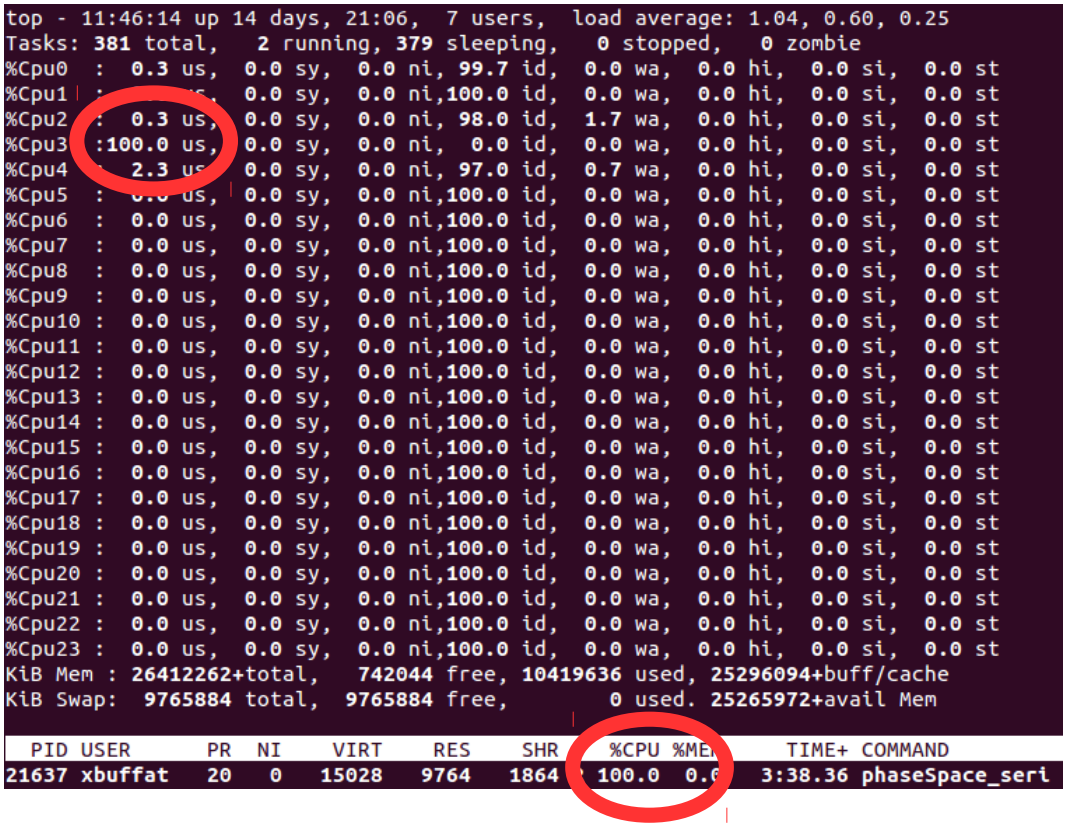
Performance

```
top - 11:46:14 up 14 days, 21:06,  7 users,  load average: 1.04, 0.60, 0.25
Tasks: 381 total,   2 running, 379 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.3 us,  0.0 sy,  0.0 ni, 98.0 id,  1.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  2.3 us,  0.0 sy,  0.0 ni, 97.0 id,  0.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 26412262+total,   742044 free, 10419636 used, 25296094+buff/cache
KiB Swap:  9765884 total,  9765884 free,        0 used. 25265972+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
21637 xbuffat   20   0   15028   9764   1864 R 100.0  0.0   3:38.36 phaseSpace_seri
```

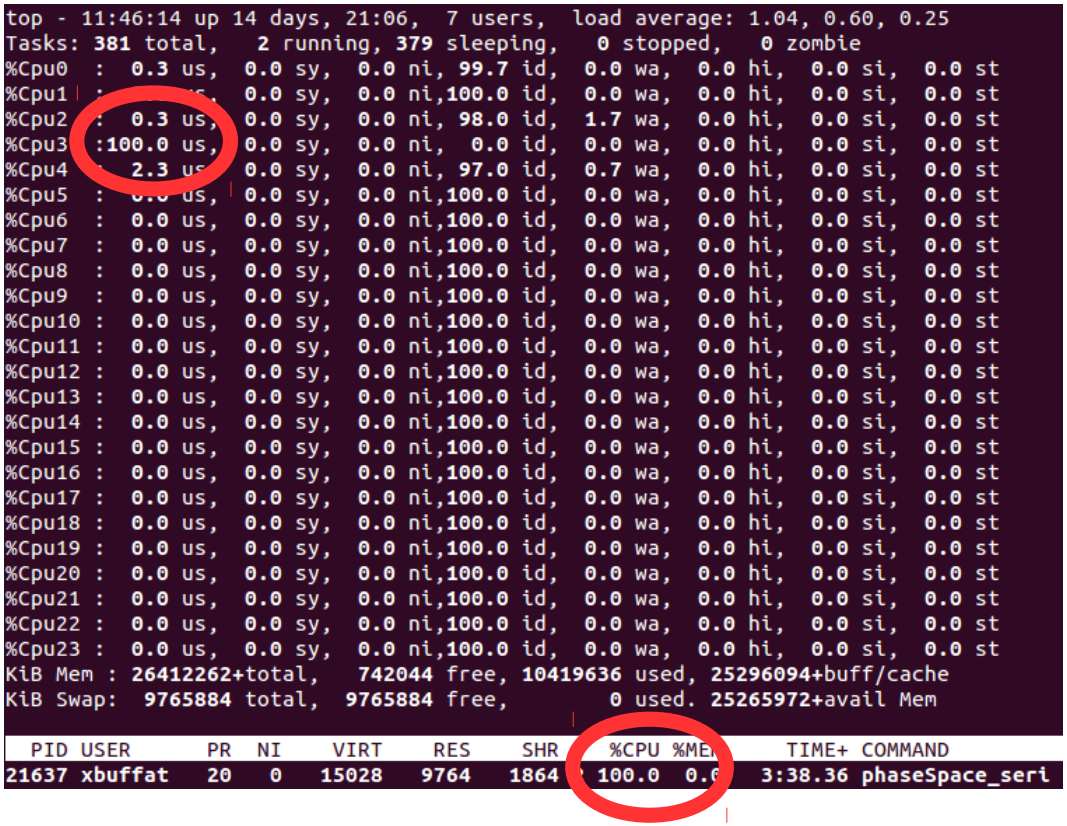
Performance



Performance



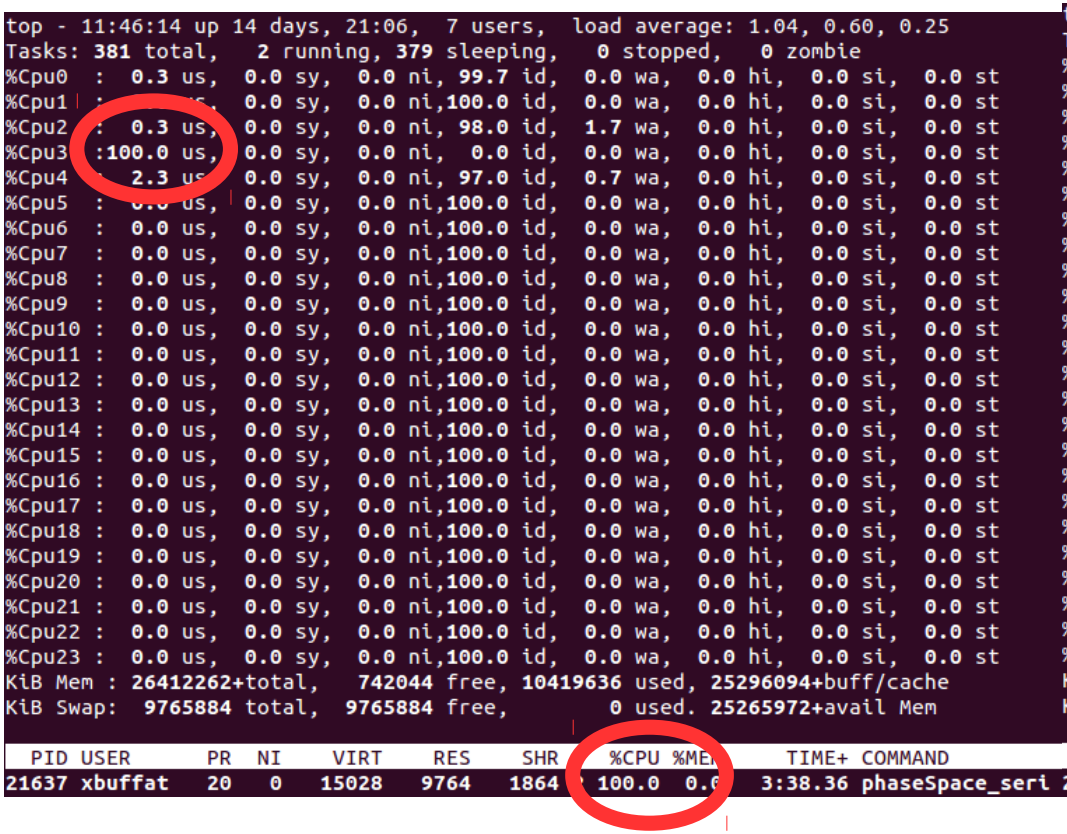
Performance

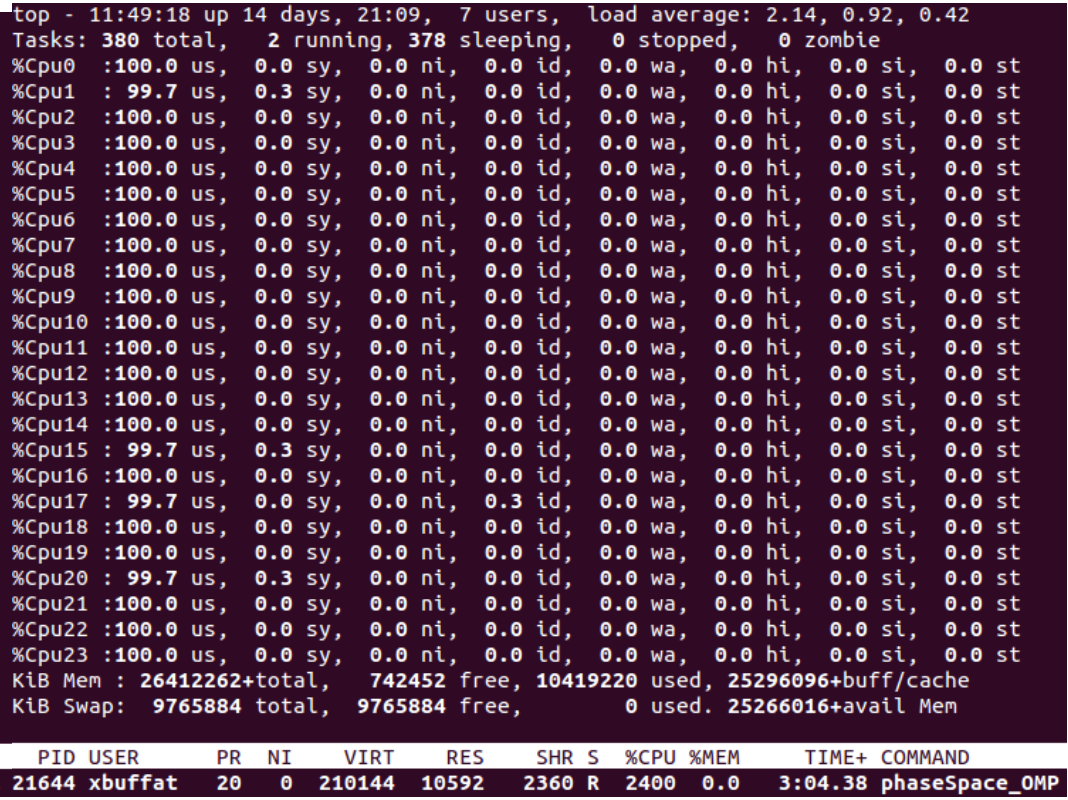


- 1 process at 100% → one CPU at 100%

  → Serial code execution time : 277s

Performance





- 1 process at 100% → one CPU at 100%

  → Serial code execution time : 277s

Performance

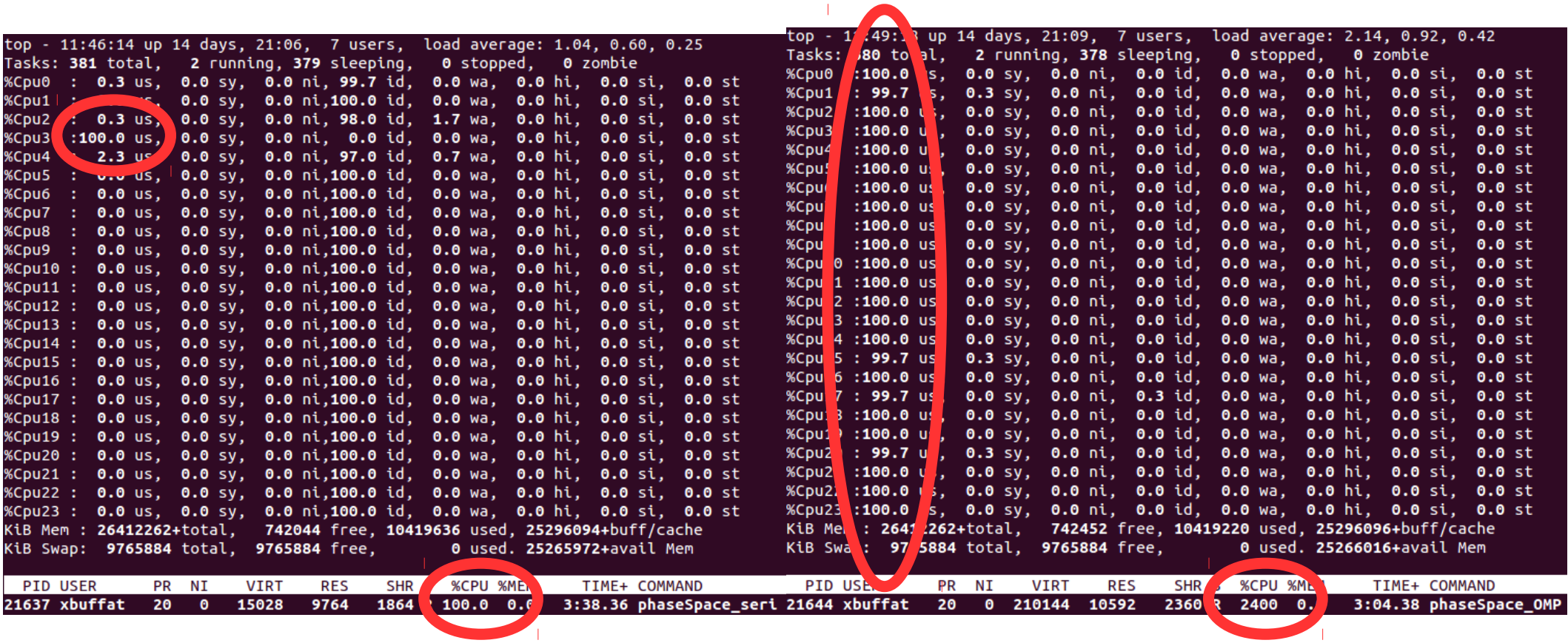
```
top - 11:46:14 up 14 days, 21:06,  7 users,  load average: 1.04, 0.60, 0.25
Tasks: 381 total,   2 running, 379 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.3 us,  0.0 sy,  0.0 ni, 98.0 id,  1.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  2.3 us,  0.0 sy,  0.0 ni, 97.0 id,  0.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 26412262+total,   742044 free, 10419636 used, 25296094+buff/cache
KiB Swap:  9765884 total,  9765884 free,        0 used. 25265972+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
21637 xbuffat   20   0   15028   9764   1864 R 100.0  0.0   3:38.36 phaseSpace_seri
```

```
top - 11:49:18 up 14 days, 21:09,  7 users,  load average: 2.14, 0.92, 0.42
Tasks: 380 total,   2 running, 378 sleeping,   0 stopped,   0 zombie
%Cpu0  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 : 99.7 us,  0.0 sy,  0.0 ni,  0.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 26412262+total,   742452 free, 10419220 used, 25296096+buff/cache
KiB Swap:  9765884 total,  9765884 free,        0 used. 25266016+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
21644 xbuffat   20   0  210144  10592   2360 R  2400  0.0   3:04.38 phaseSpace_OMP
```

- 1 process at 100% → one CPU at 100%

  → Serial code execution time : 277s

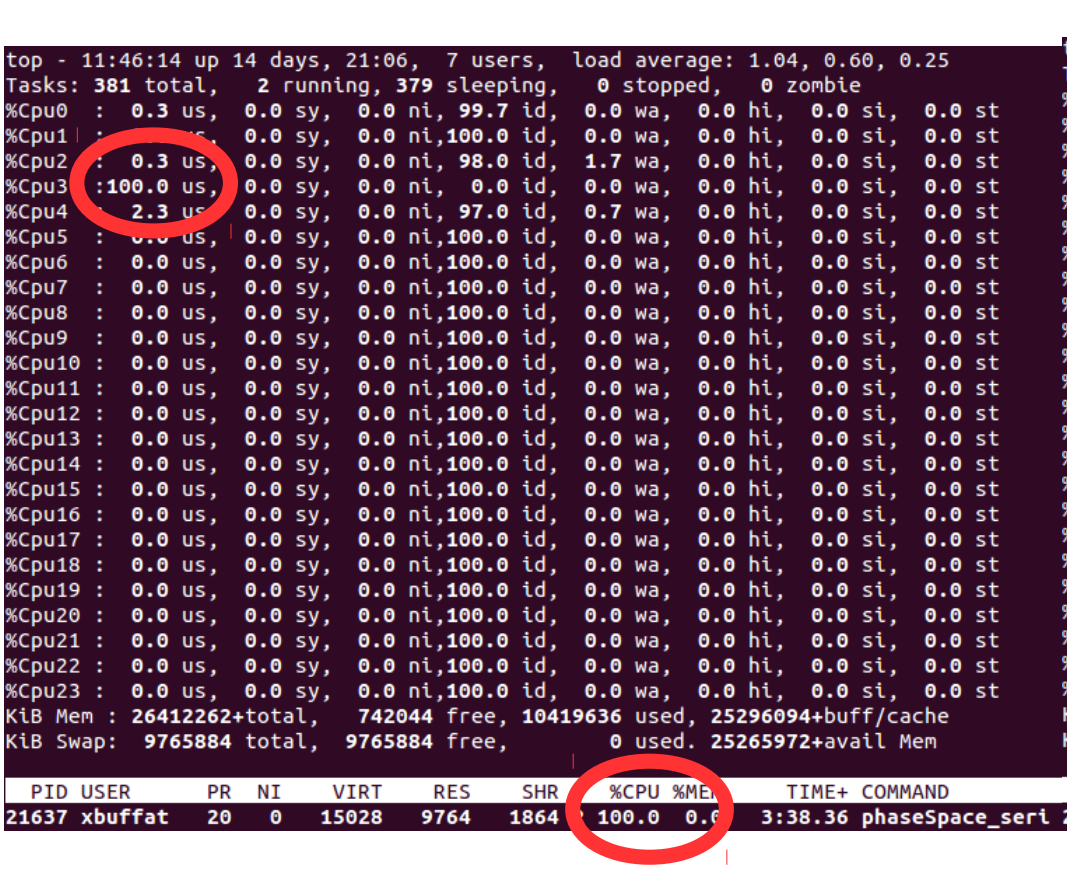
Performance

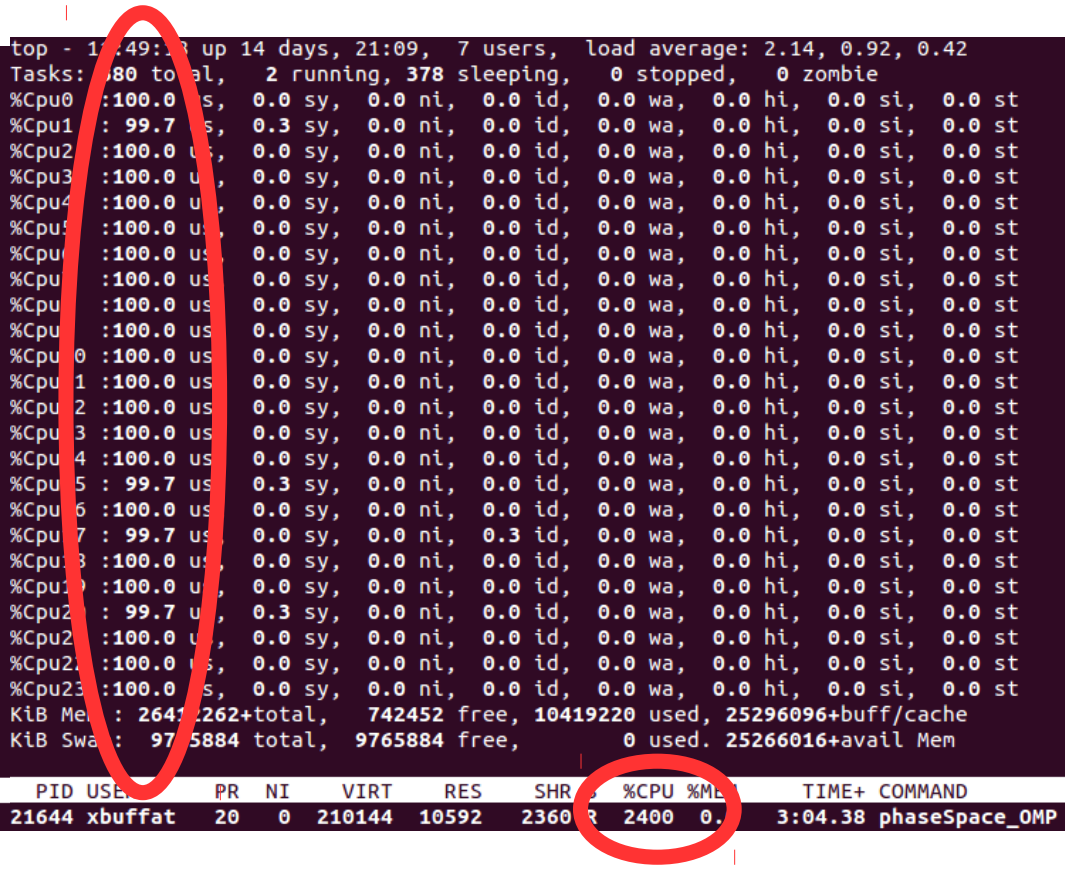


- 1 process at 100% → one CPU at 100%

  → Serial code execution time : 277s

Performance





- 1 process at 100% → one CPU at 100%

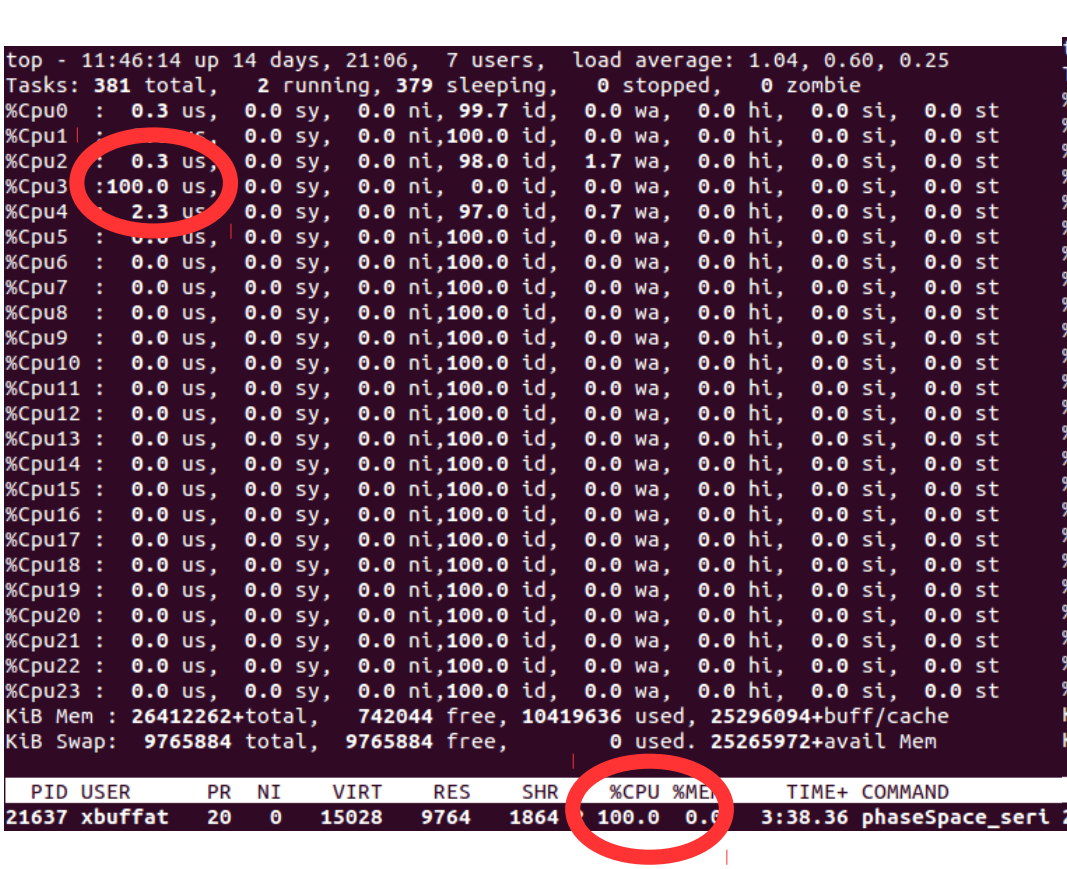
  → Serial code execution time : 277s

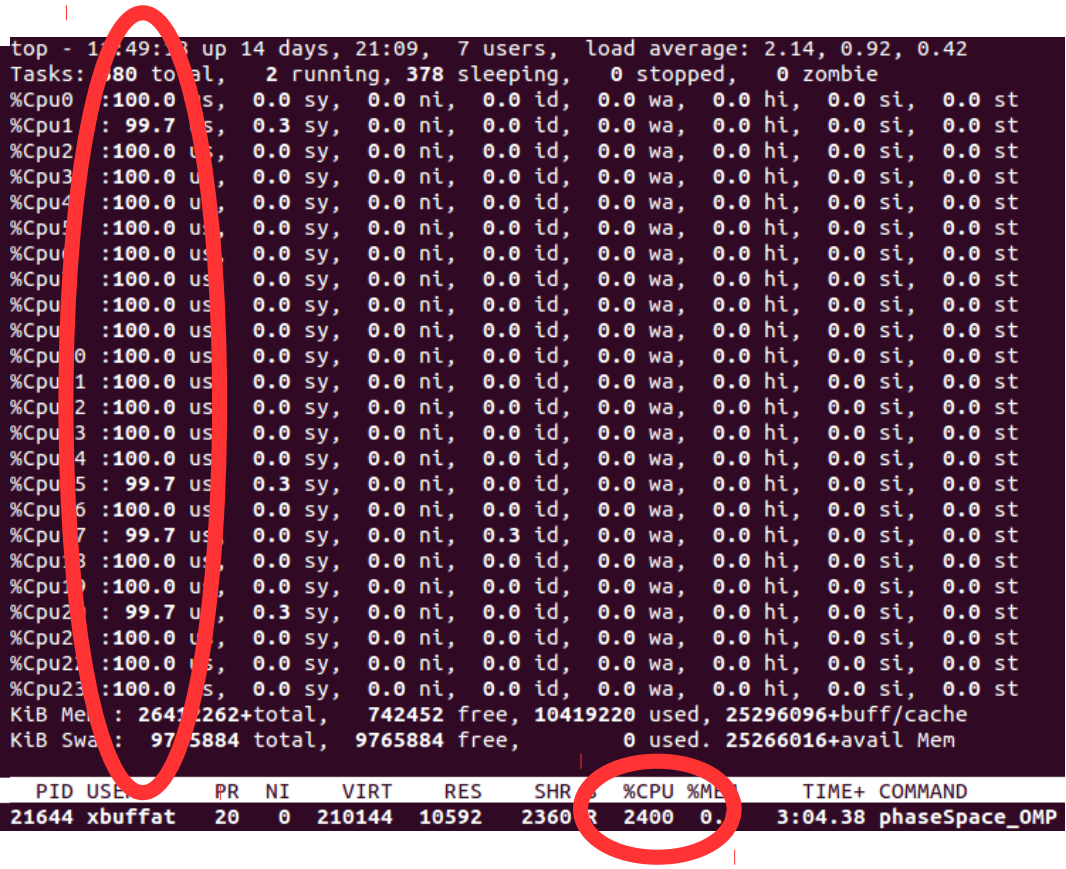
- 1 Process at 2400% (i.e. 24 threads at 100%)

  → 24 CPU at 100%

- Multithreaded code execution time : 15s

Performance





- 1 process at 100% → one CPU at 100%

  → Serial code execution time : 277s

$$\frac{277}{15} \approx 18$$

- 1 Process at 2400% (i.e. 24 threads at 100%)

  → 24 CPU at 100%

- Multithreaded code execution time : 15s

18 faster with a 12 cores machine ?

- Fuego, a rabbit that questions my sanity

Hyperthreading

```
xbuffat@LIU:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 45
Model name:            Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz
Stepping:              7
CPU MHz:               1199.953
CPU max MHz:           2800.0000
CPU min MHz:           1200.0000
BogoMIPS:              4601.40
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              15360K
NUMA node0 CPU(s):     0-5,12-17
NUMA node1 CPU(s):     6-11,18-23
```

- From the operating system point of view, hyperthreaded cores appear as two separate CPUs
- The performance is not necessarily the one of two CPUs...

Hyperthreading

```
xbuffat@LIU:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 45
Model name:            Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz
Stepping:              7
CPU MHz:               1199.953
CPU max MHz:           2800.0000
CPU min MHz:           1200.0000
BogoMIPS:              4601.40
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              15360K
NUMA node0 CPU(s):     0-5,12-17
NUMA node1 CPU(s):     6-11,18-23
```

- From the operating system point of view, hyperthreaded cores appear as two separate CPUs
- The performance is not necessarily the one of two CPUs...

Hyperthreading

```
xbuffat@LIU:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 45
Model name:            Intel(R) Xeon(R) CPU E5-2630 0 @ 2.30GHz
Stepping:              7
CPU MHz:               1199.953
CPU max MHz:           2800.0000
CPU min MHz:           1200.0000
BogoMIPS:              4601.40
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              15360K
NUMA node0 CPU(s):     0-5,12-17
NUMA node1 CPU(s):     6-11,18-23
```

- From the operating system point of view, hyperthreaded cores appear as two separate CPUs
- The performance is not necessarily the one of two CPUs...

Hyperthreading and context switching

- Cores with hyperthreading are equipped with additional hardware, allowing to swtich from one thread to another efficiently, thus avoiding idling of CPUs
- When allocating more threads than CPUs, the operating system is also capable of switching threads (→ context switching), however this operation is slower

Hyperthreading and context switching

- Cores with hyperthreading are equipped with additional hardware, allowing to swtich from one thread to another efficiently, thus avoiding idling of CPUs
- When allocating more threads than CPUs, the operating system is also capable of switching threads (→ context switching), however this operation is slower
- Hyperthreading is particularly useful for your daily life, when you are checking emails, listening to music, reading newspapers, watching a Tutorial on Youtube, preparing slides for tomorrow's meeting and writing a Nobel price winning theory in LaTeX at the same time

Hyperthreading and context switching

- Cores with hyperthreading are equipped with additional hardware, allowing to swtich from one thread to another efficiently, thus avoiding idling of CPUs
- When allocating more threads than CPUs, the operating system is also capable of switching threads (→ context switching), however this operation is slower
- Hyperthreading is particularly useful for your daily life, when you are checking emails, listening to music, reading newspapers, watching a Tutorial on Youtube, preparing slides for tomorrow's meeting and writing a Nobel price winning theory in LaTeX at the same time
- For scientific computing, it may be beneficial for codes where threads not synchronous, are non-uniformly loaded or requires heavy memory access

Hyperthreading and context switching

- Cores with hyperthreading are equipped with additional hardware, allowing to swtich from one thread to another efficiently, thus avoiding idling of CPUs
- When allocating more threads than CPUs, the operating system is also capable of switching threads (→ context switching), however this operation is slower
- Hyperthreading is particularly useful for your daily life, when you are checking emails, listening to music, reading newspapers, watching a Tutorial on Youtube, preparing slides for tomorrow's meeting and writing a Nobel price winning theory in LaTeX at the same time
- For scientific computing, it may be beneficial for codes where threads not synchronous, are non-uniformly loaded or requires heavy memory access

```
double oldX = 0.0;
double oldPx = 0.0;
for(int turn = 0;turn<nTurn;++turn){
      #pragma omp parallel for default(firstprivate) shared(x,px) \
      schedule(guided,1000)
   for(int i = 0;i<nPart;++i){
      oldX = x[i];
      oldPx = px[i];
      x[i] = cosPhix*oldX + sinPhix*oldPx;
      px[i] = -sinPhix*oldX + cosPhix*oldPx;
      if(abs(x[i]) > thres){
         px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
      }
   }
}
```

- In my tracking example, the function **exp** is prone to acceleration with hyperthreading

Multithreading

Multithreading

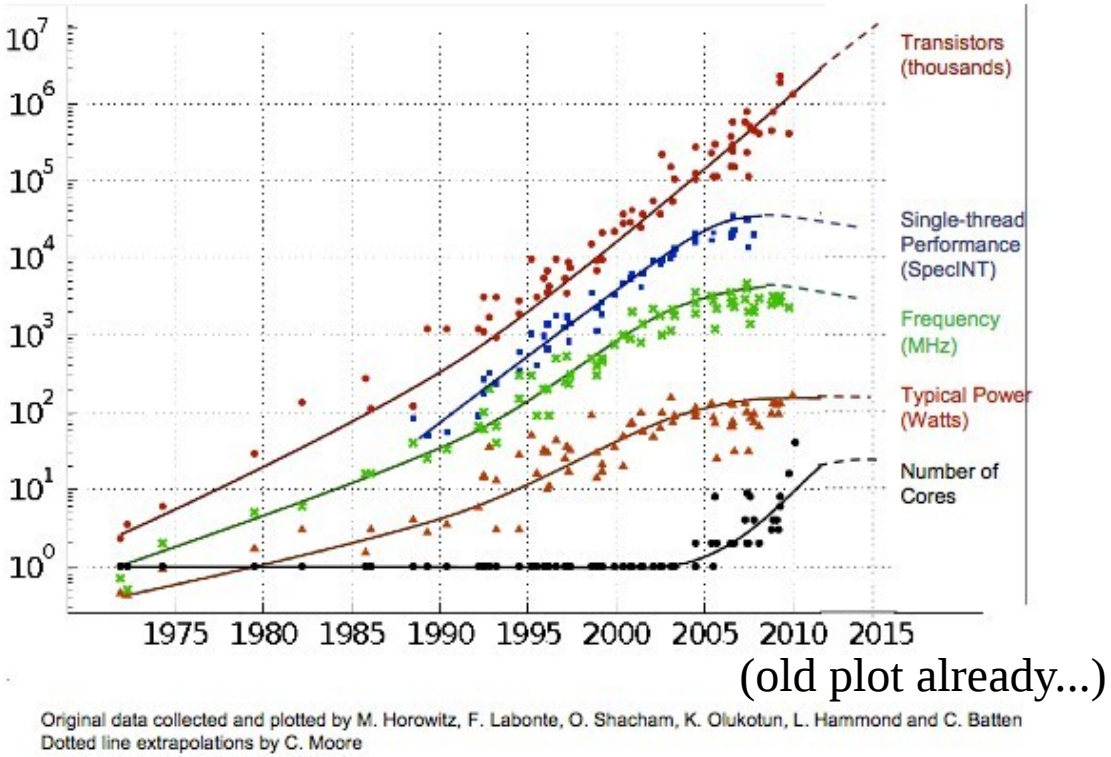
Hyperthreading

Serial, multi- and hyper-threaded codes

- Modern computers are based on a set of instructions to be performed by a processing unit (CPU) on a given data
- Most computers today allow for multithreading, i.e. multiple CPUs working in parallel on a shared memory
  - Due to technological limitations, the access to the shared memory is not uniformly efficient → The guide line is **maximise data locality**
  - Multithreading is easy (even in Fortran!), especially if you let the compiler do the job
    - It usually becomes extremely easy in high level languages such as Python or Matlab, it can be even automatic when using built-in functions or libraries (one of the reason to avoid for loops in such languages)
  - Multithreading can provide speedup linear with the number of real CPU in your machine, possibly more thanks to hyperthreading
  - The speed up can be limited by causality and memory access

    → Memory bound problem usually don't profit of such a scheme
- Hyperthreading is a technology allowing to maximise the efficiency of a single processing unit by switching fast between different threads busy with other tasks such as memory transfer
  - Even though they are not physically separate CPUs, they appear as such to the operating system (and to you)

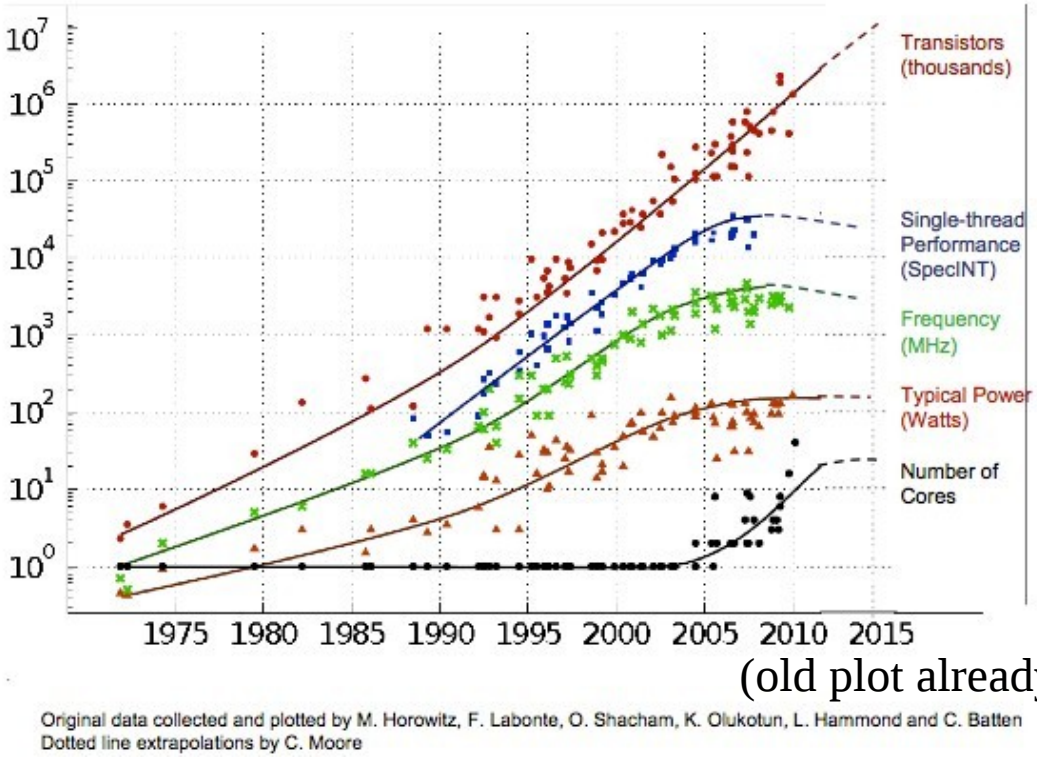
About Moore's law



(old plot already...)

- From the point of view of number of transistors, Moore's law is still ~alive (for how long?)

About Moore's law



(old plot already...)

- From the point of view of number of transistors, Moore's law is still ~alive (for how long?)
- The performance increase nowadays come with additional cores (rather than clock speed for example)

Amdahl's law

Non parallelisable code

parallelisable code

$$S = \frac{\alpha + 1 - \alpha}{\alpha + \frac{1-\alpha}{N_p}}$$

$$= \frac{1}{\alpha + \frac{1-\alpha}{N_p}}$$

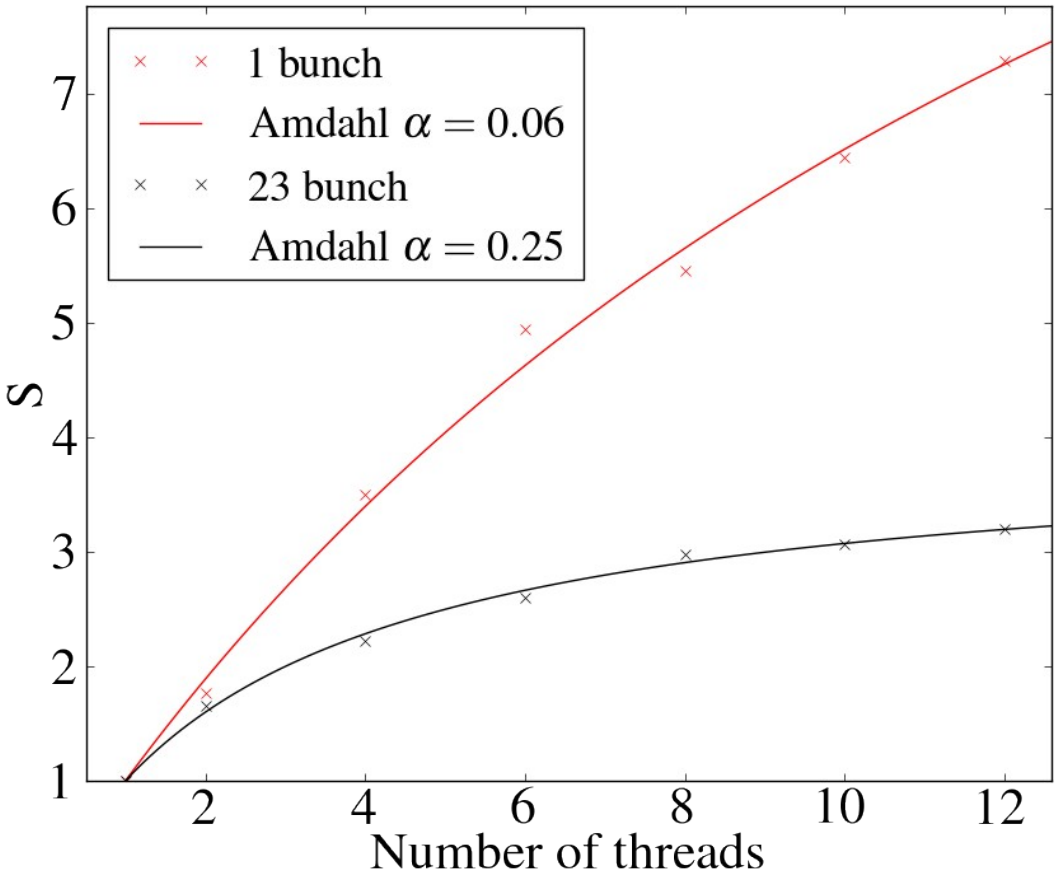
Amdahl's law

Non parallelisable code

parallelisable code

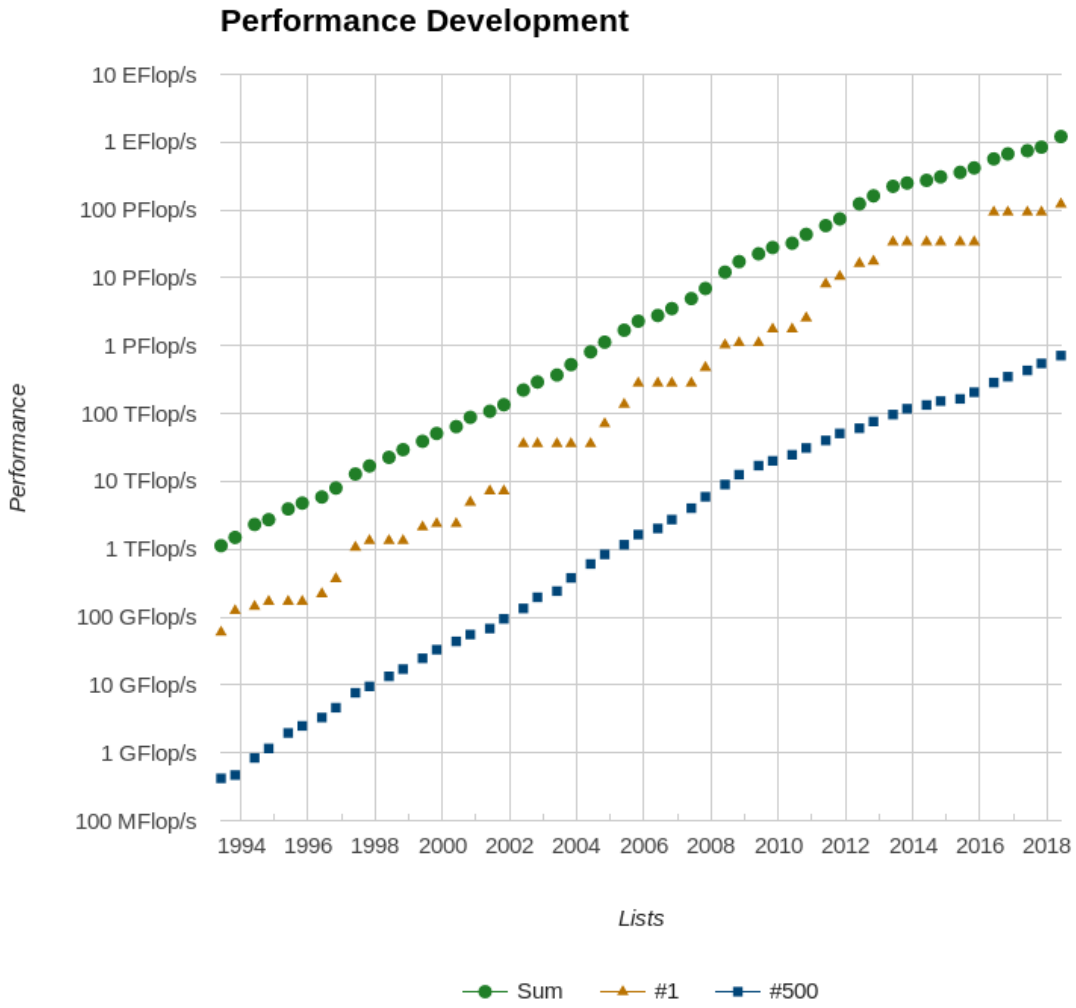
$$S = \frac{\alpha + 1 - \alpha}{\alpha + \frac{1-\alpha}{N_p}}$$

$$= \frac{1}{\alpha + \frac{1-\alpha}{N_p}}$$



- For a fixed problem size, the speed up saturates for large number of cores
  - Usually high performance codes are needed to increase the problem size rather than speed up what is achievable with a single CPU
- Comparing the speed up you obtained to Amdahl's law (fitting alpha) is a good way to asses the quality of your scheme

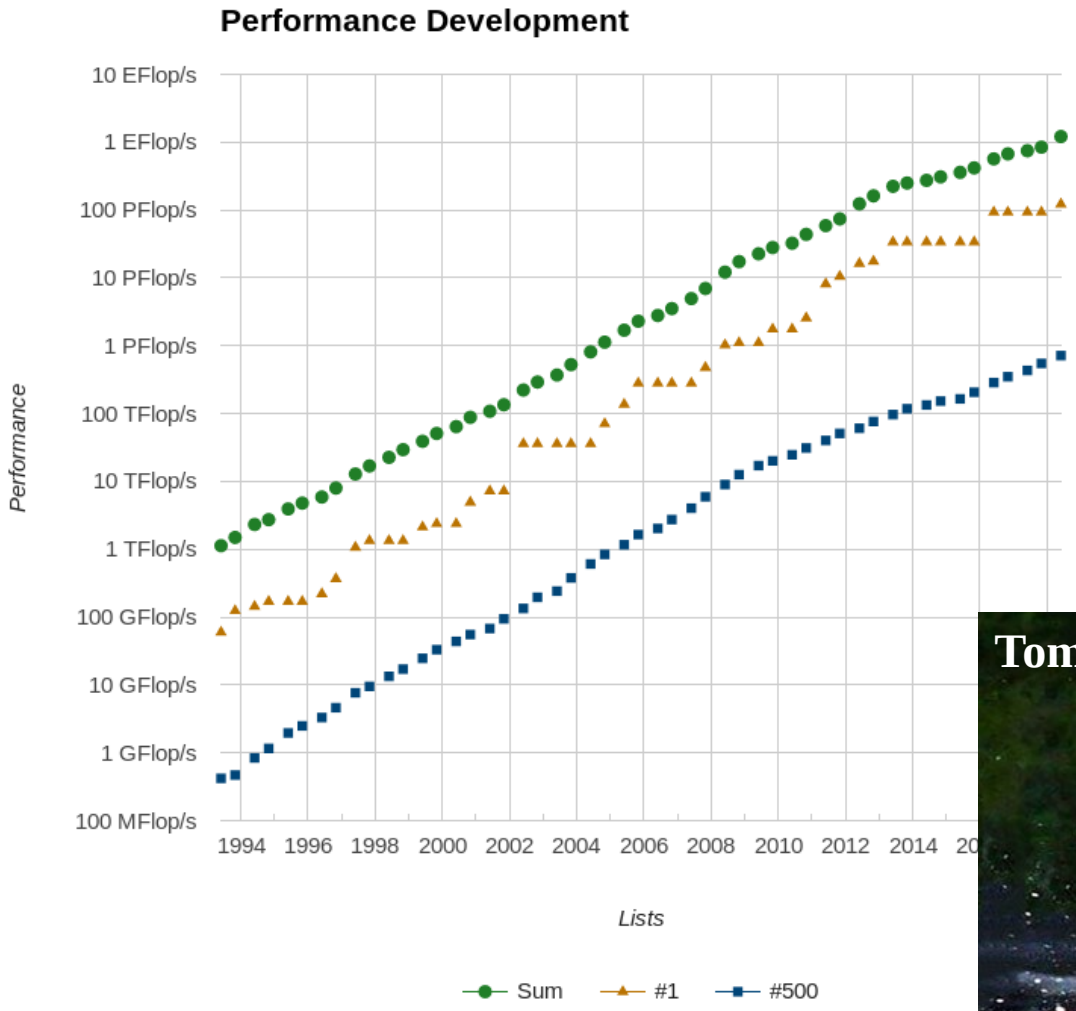
About Moore's Law



- The computing power of the most powerful machine in the world does keep increasing !

Source : http://www.top500.org/

About Moore's Law



- The computing power of the most powerful machine in the world does keep increasing !

Source : http://www.top500.org/

Content

I : Everyday computing

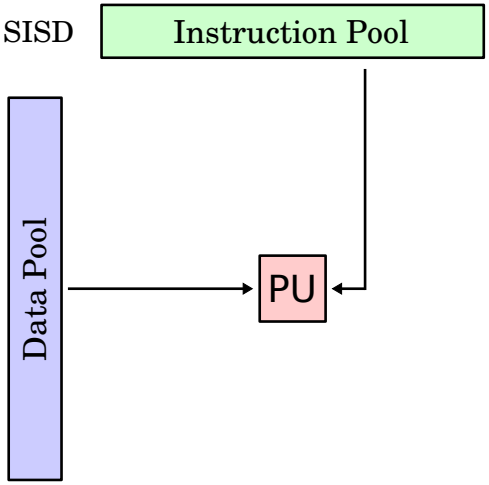
- The Turing machine
- The Central Processing Unit (CPU)
  - Intrinsics, vectorisation
- Multiple CPUs
  - Multithreading
  - NUMA zones
  - Hyperthreading
- Amdhal's law and Moore's Law

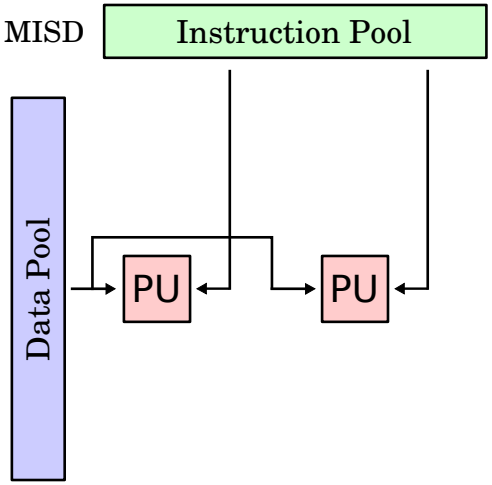
II : High performance computing

- The Graphics Processing Unit (GPU)
- Computer clusters
  - The Message Passing Interface
- Volunteer and grid computing
- Summary

Flynn's taxonomy

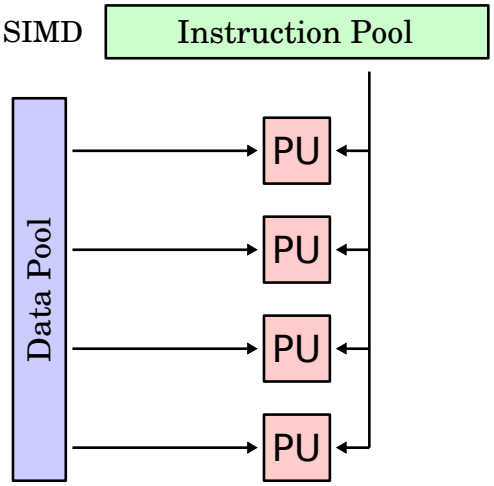
- Single instruction, single data
  - Most common / simple

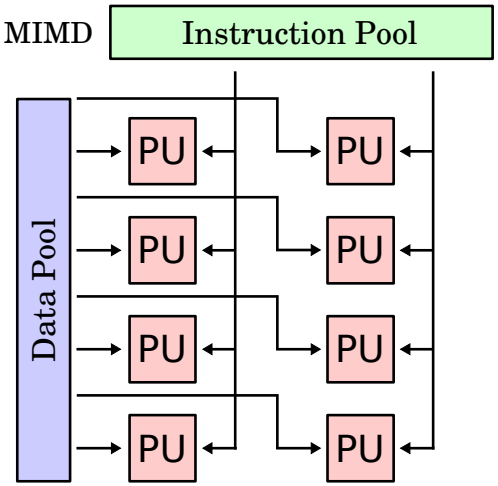




- **Multiple** instructions, single data
  - ?

- Single instruction, **multiple** data
  - Vectorisation
  - Graphics processing units (GPU)





- **Multiple** instructions, **multiple** data
  - Multithreading (shared memory parallelisation)
  - Cluster (distributed memory parallelisation)

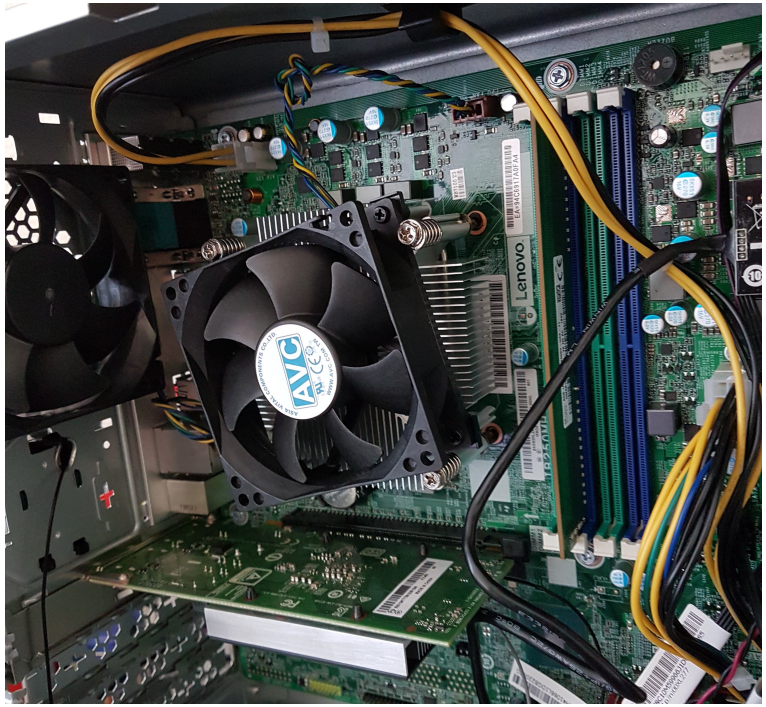
Illustrations from https://en.wikipedia.org/wiki/Flynn's_taxonomy

The Graphical Processing Unit (GPU)

- Image processing usually require a single (simple) operation to be perform on large data set (e.g. 3D projection on 2D screen)
  - → Computer games, special effects (VFX, CGI...)
  - → Large market, therefore lots of money for development !

The Graphical Processing Unit (GPU)

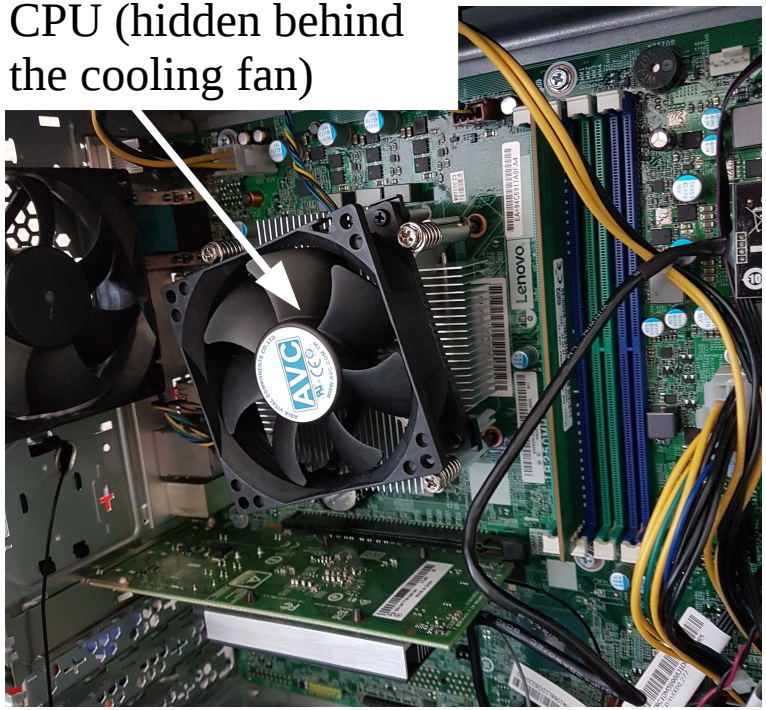
- Image processing usually require a single (simple) operation to be perform on large data set (e.g. 3D projection on 2D screen)
  - → Computer games, special effects (VFX, CGI...)
  - → Large market, therefore lots of money for development !



The Graphical Processing Unit (GPU)

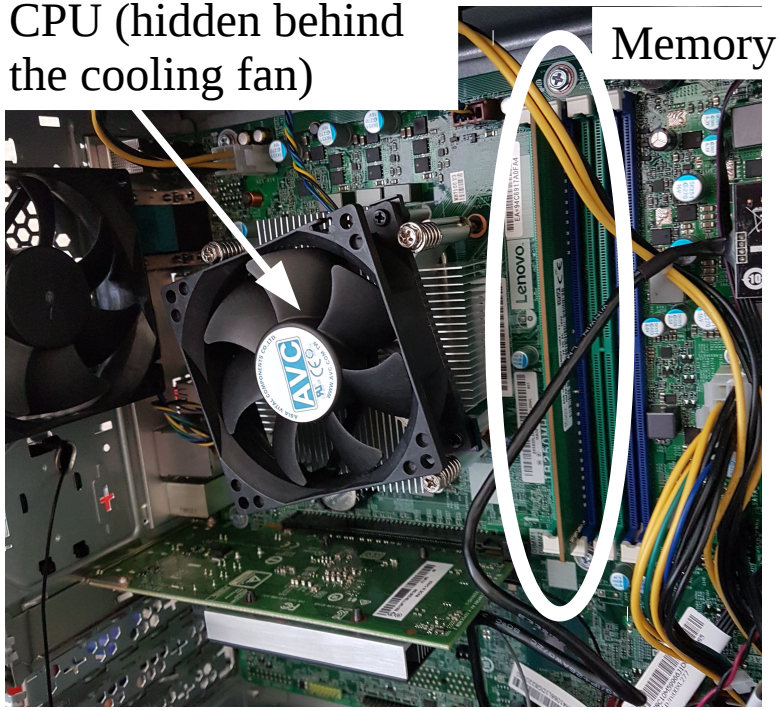
- Image processing usually require a single (simple) operation to be perform on large data set (e.g. 3D projection on 2D screen)
  - → Computer games, special effects (VFX, CGI...)
  - → Large market, therefore lots of money for development !

CPU (hidden behind the cooling fan)



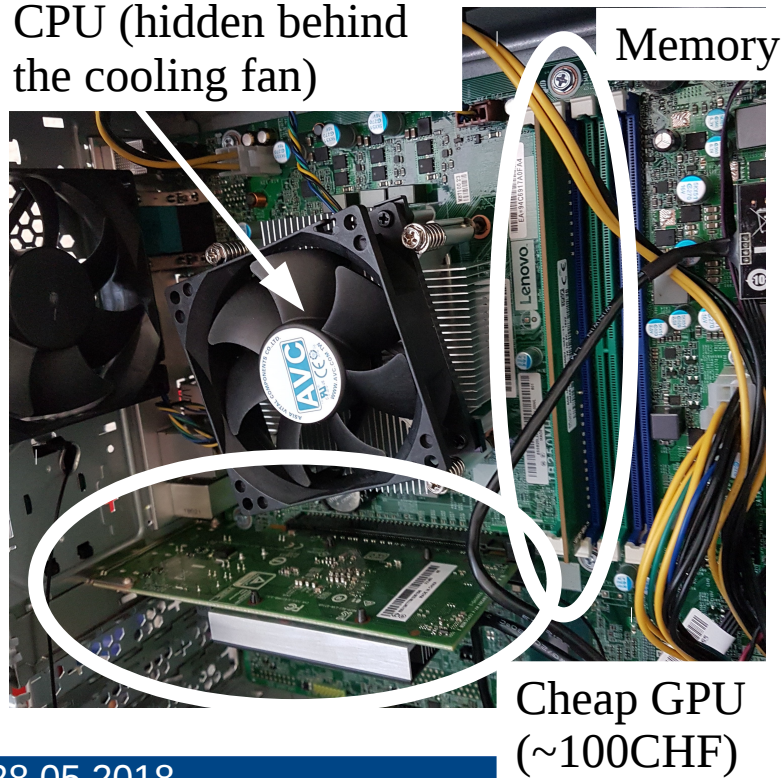
The Graphical Processing Unit (GPU)

- Image processing usually require a single (simple) operation to be perform on large data set (e.g. 3D projection on 2D screen)
  - → Computer games, special effects (VFX, CGI...)
  - → Large market, therefore lots of money for development !



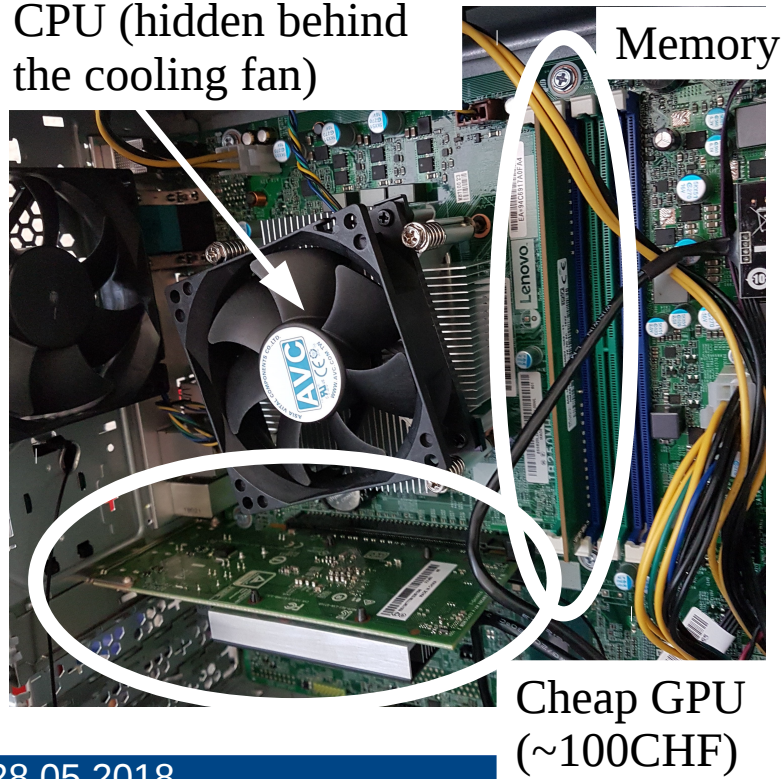
The Graphical Processing Unit (GPU)

- Image processing usually require a single (simple) operation to be perform on large data set (e.g. 3D projection on 2D screen)
  - → Computer games, special effects (VFX, CGI...)
  - → Large market, therefore lots of money for development !



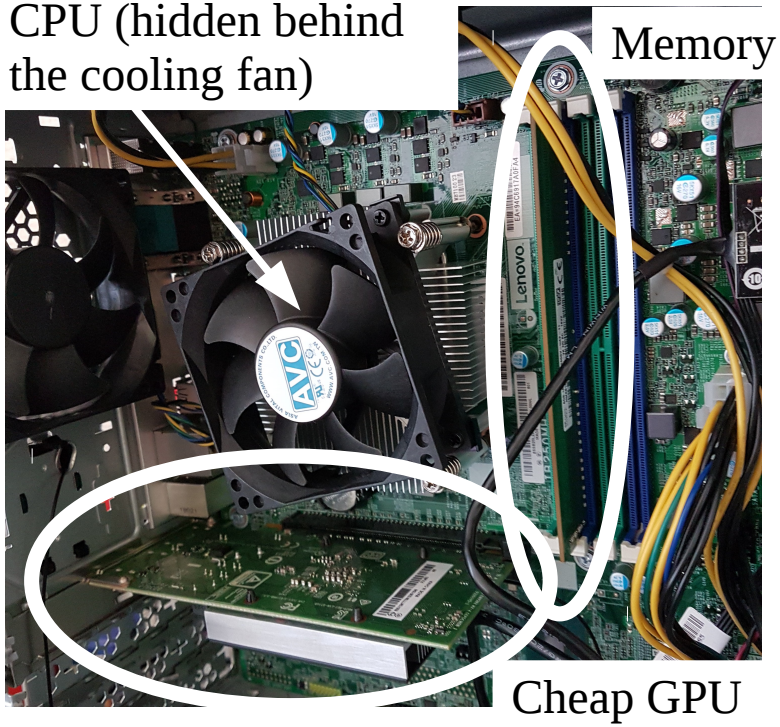
The Graphical Processing Unit (GPU)

- Image processing usually require a single (simple) operation to be perform on large data set (e.g. 3D projection on 2D screen)
  → Computer games, special effects (VFX, CGI...)
  → Large market, therefore lots of money for development !
- These developments can be very useful for scientific applications as well
  - Watch out, most scientific application require double precision, whereas image processing is usually done in single precision
  - Compute capability >1.3 indicate double precision capacity, but not necessarily good performance with double precision!
    → Dedicated cards exists → some people talk about General Purpose GPU (GPGPU)



The Graphical Processing Unit (GPU)

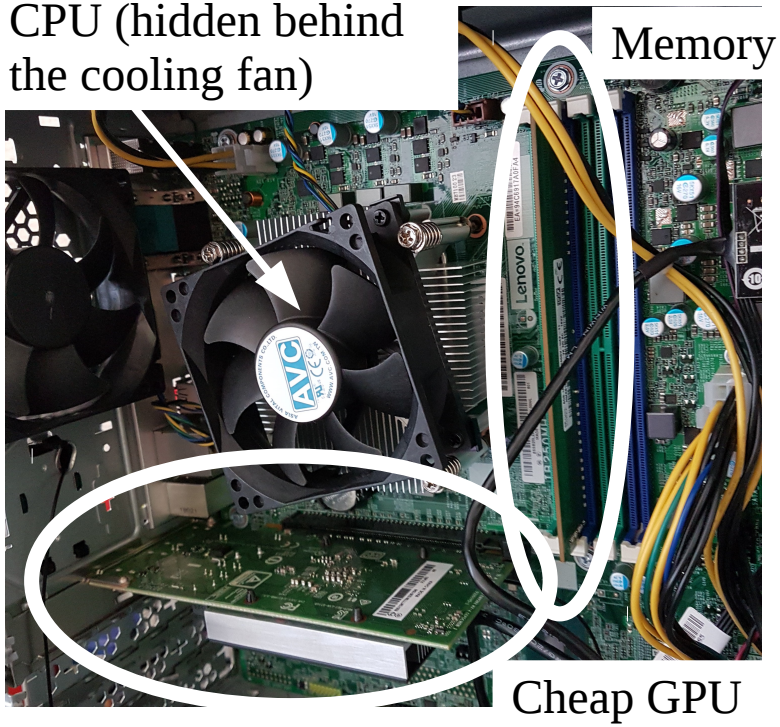
- Image processing usually require a single (simple) operation to be perform on large data set (e.g. 3D projection on 2D screen)
  - → Computer games, special effects (VFX, CGI...)
  - → Large market, therefore lots of money for development !
- These developments can be very useful for scientific applications as well
  - Watch out, most scientific application require double precision, whereas image processing is usually done in single precision
  - Compute capability >1.3 indicate double precision capacity, but not necessarily good performance with double precision!
    - → Dedicated cards exists → some people talk about General Purpose GPU (GPGPU)



Not so cheap GPUs...

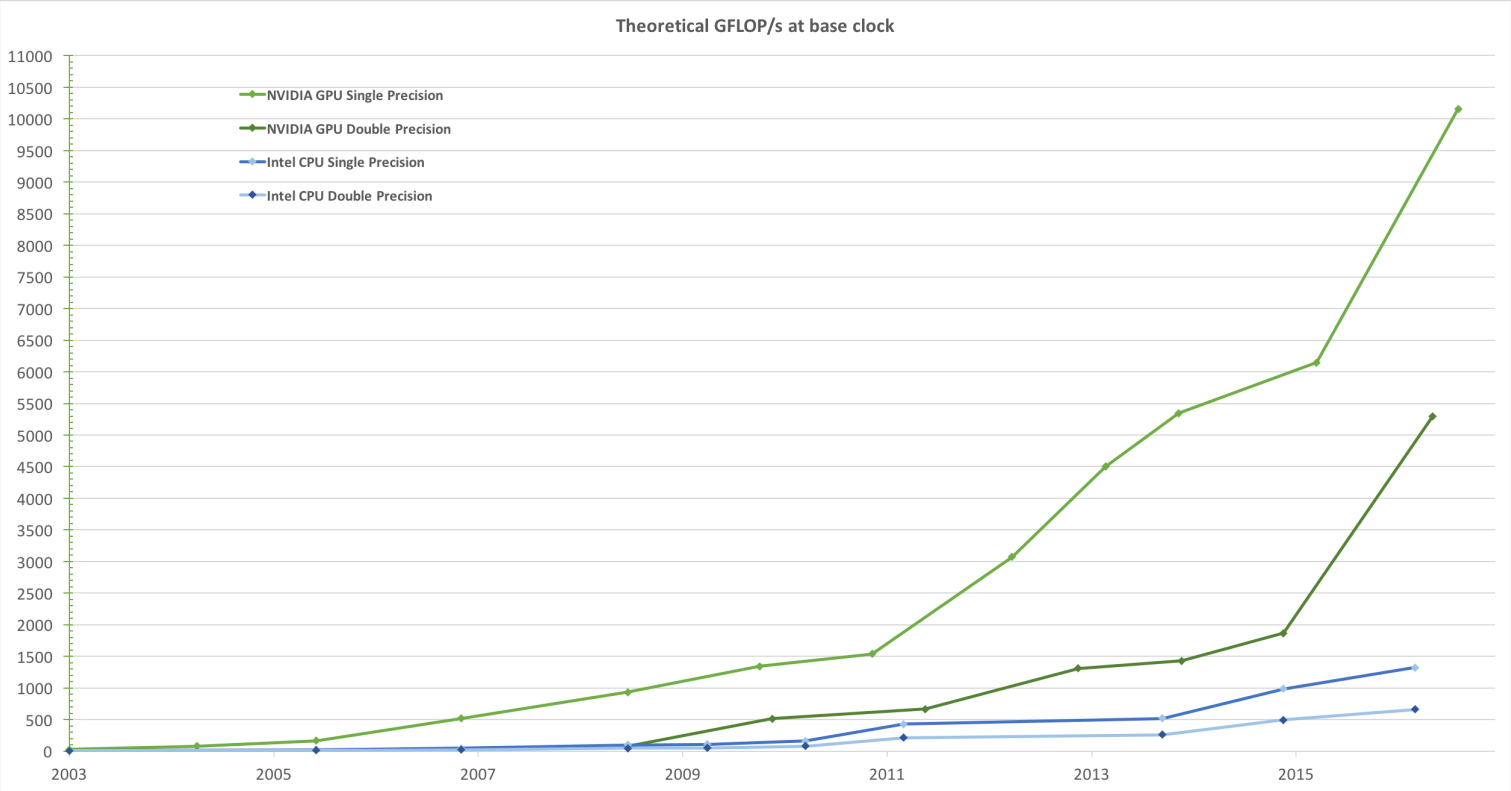
The Graphical Processing Unit (GPU)

- Image processing usually require a single (simple) operation to be perform on large data set (e.g. 3D projection on 2D screen)
  - → Computer games, special effects (VFX, CGI...)
  - → Large market, therefore lots of money for development !
- These developments can be very useful for scientific applications as well
  - Watch out, most scientific application require double precision, whereas image processing is usually done in single precision
  - Compute capability >1.3 indicate double precision capacity, but not necessarily good performance with double precision!
    - → Dedicated cards exists → some people talk about General Purpose GPU (GPGPU)



Not so cheap GPUs...

GPU / CPU



- GPUs clearly overtake CPUs in terms of number of floating point operation per second, but...

Taking vectorisation to the extreme

$$\begin{matrix} x_1 \\ x_2 \end{matrix} \rightarrow \begin{matrix} \sqrt{x_1} \\ \sqrt{x_2} \end{matrix}$$

x_1 Float 64-bits	x_2 Float 64-bits

$$x_i \rightarrow \sqrt{x_i}$$

x_1 Float 64-bits	...	x_i Float 64-bits	...	x_2 Float 64-bits

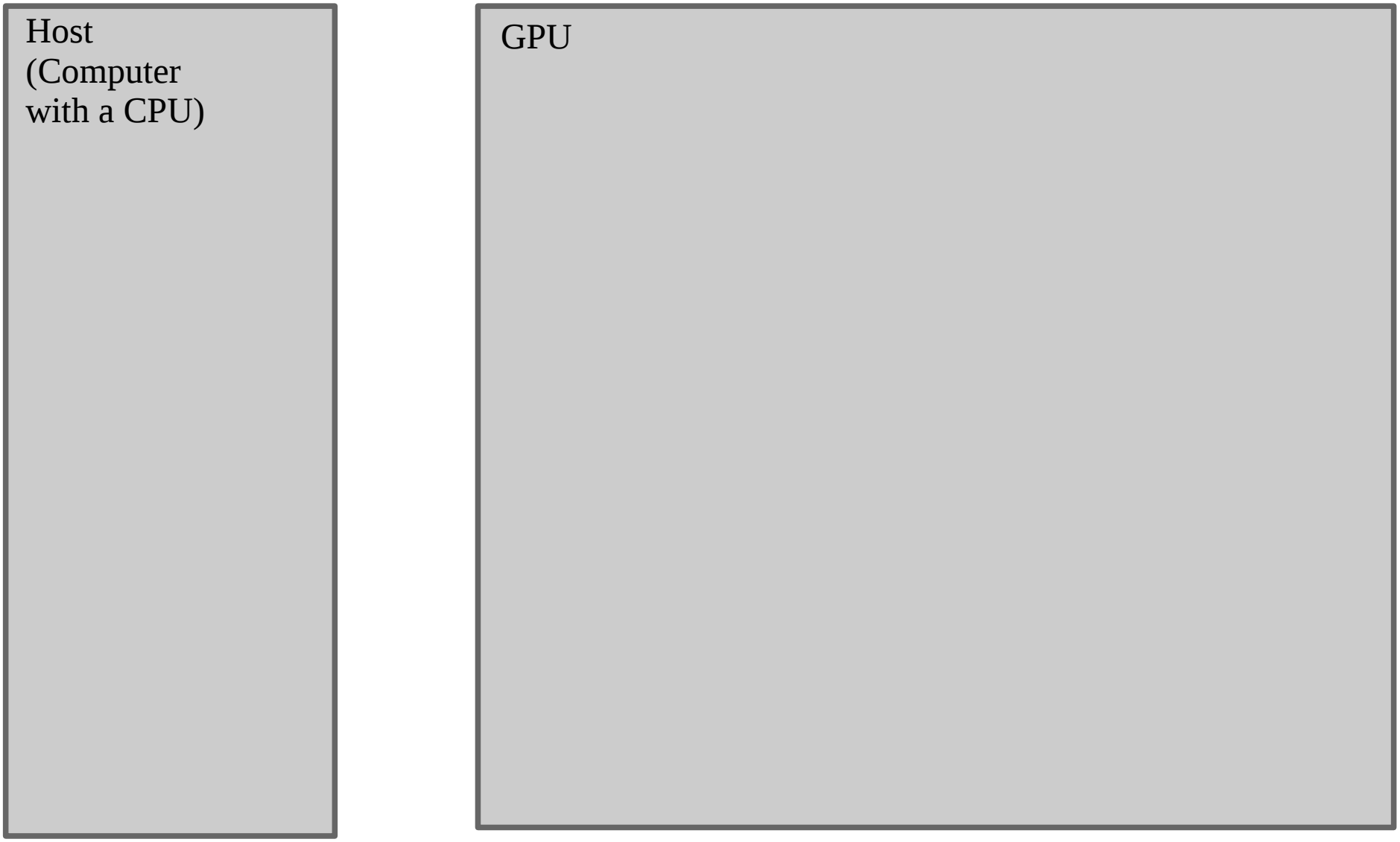
- The GPU is an extension of vectorisation to the extreme, allowing amazing speed up for some applications, useless for others...

  → The handling of large arrays takes time, the performance of a GPU has to rely on a an efficient scheduling → multithreading and fast context switching !

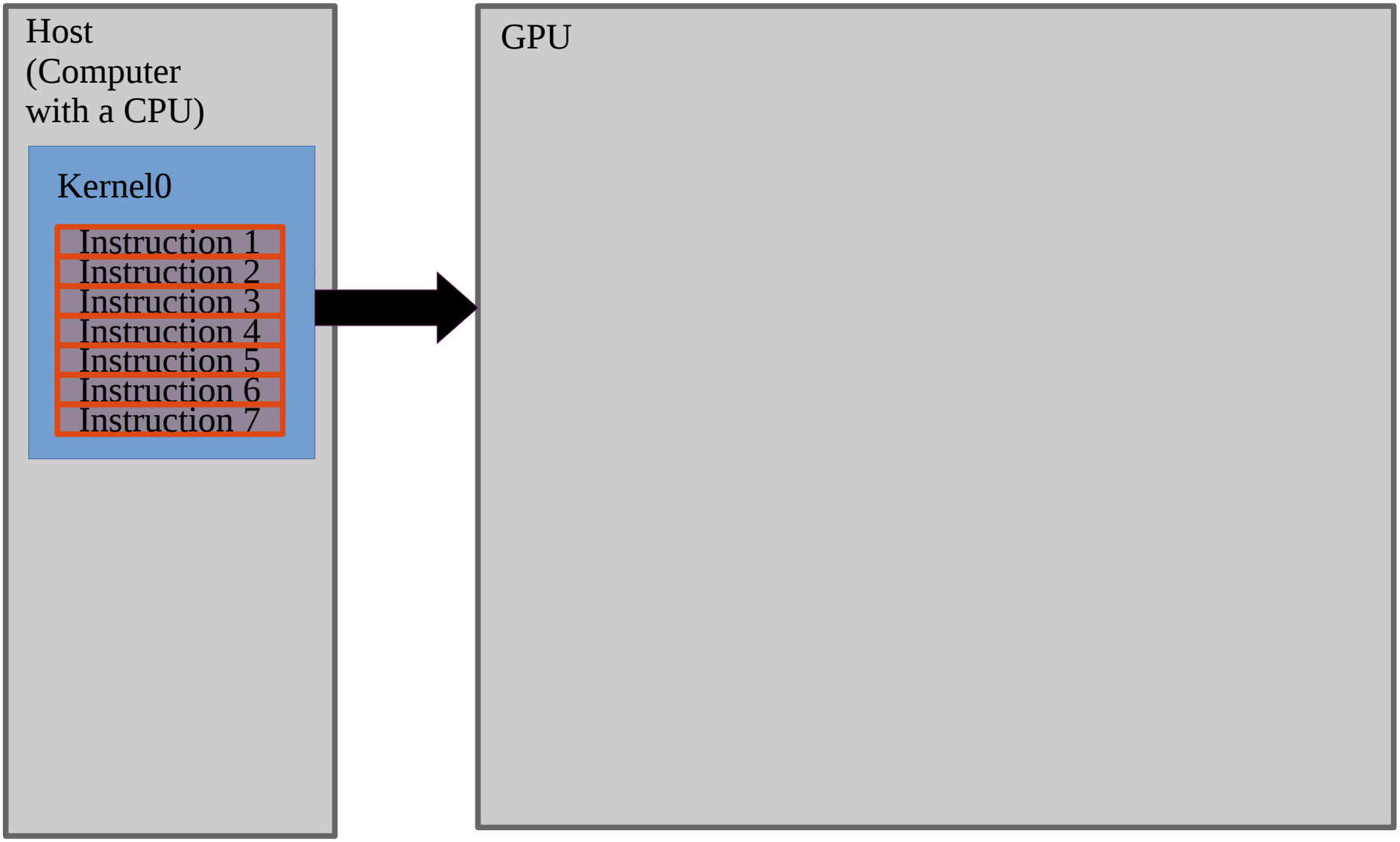
- By essence, the GPU requires a different memory architecture w.r.t. the CPU

  → The two are independent and should be handled properly !

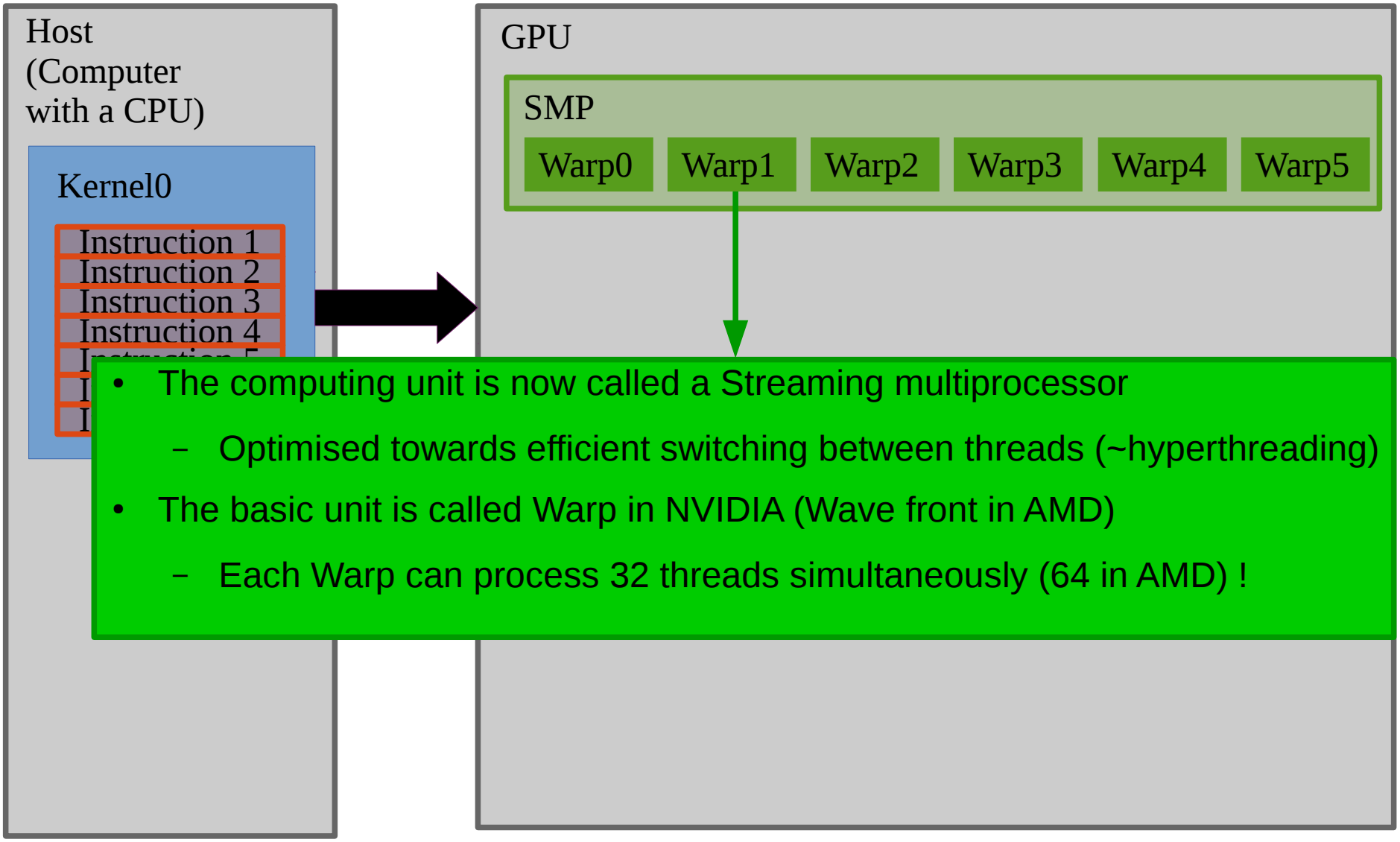
The GPU



The GPU


Host
(Computer
with a CPU)
Kernel0
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
GPU

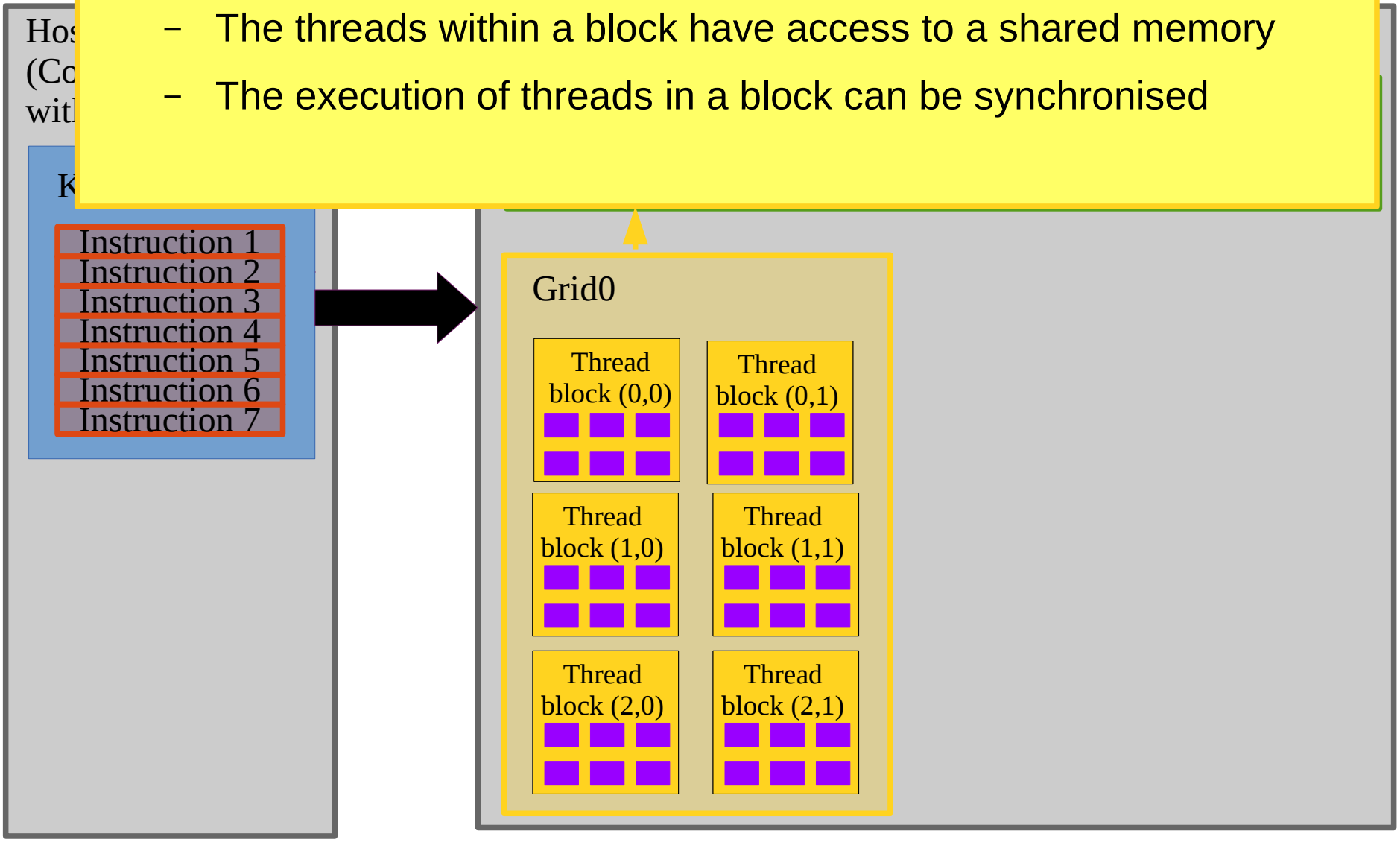
The GPU



- The computing unit is now called a Streaming multiprocessor
  - Optimised towards efficient switching between threads (~hyperthreading)
- The basic unit is called Warp in NVIDIA (Wave front in AMD)
  - Each Warp can process 32 threads simultaneously (64 in AMD) !

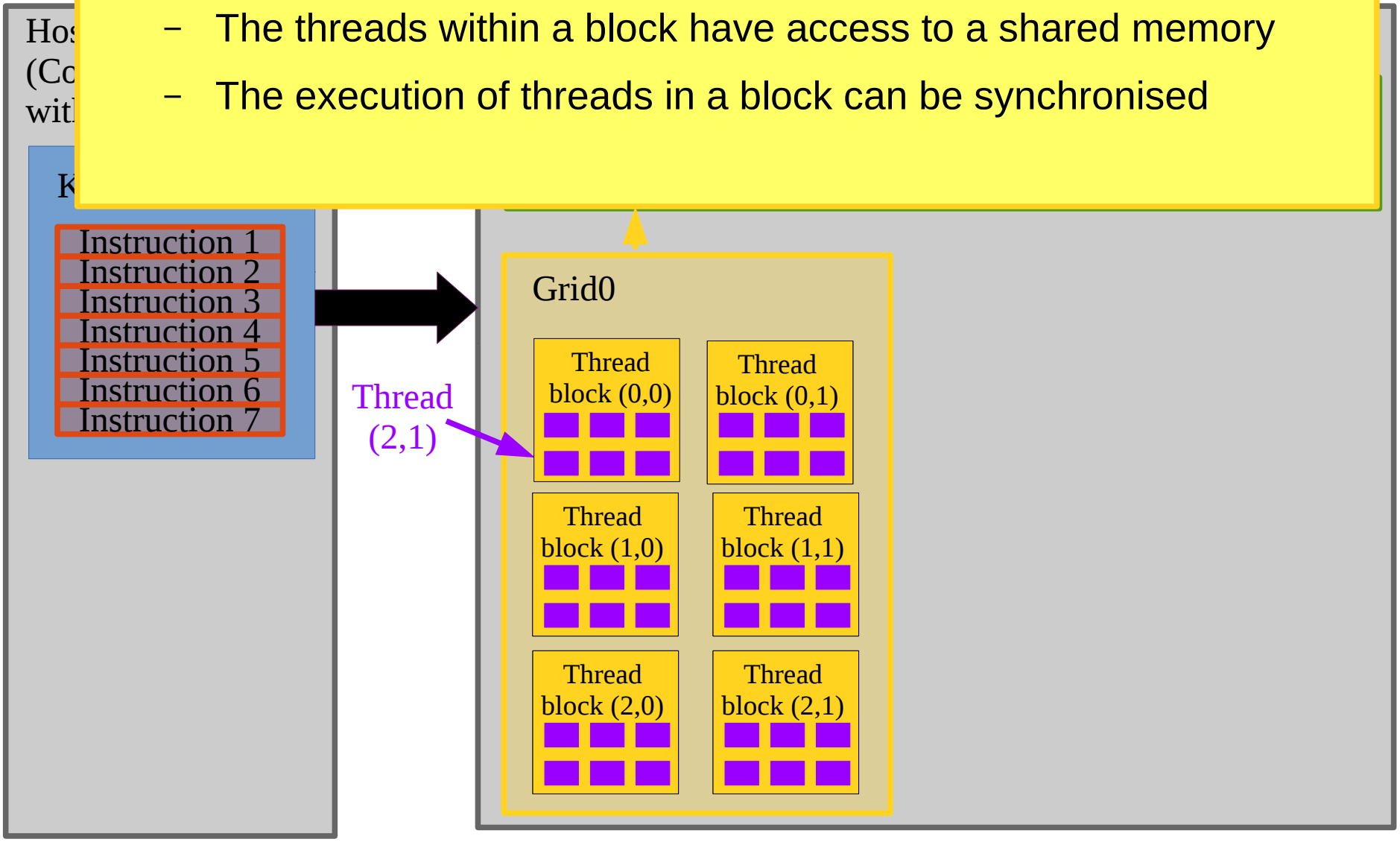
The GPU

- The threads are organised in blocks on a grid
  - The threads within a block have access to a shared memory
  - The execution of threads in a block can be synchronised

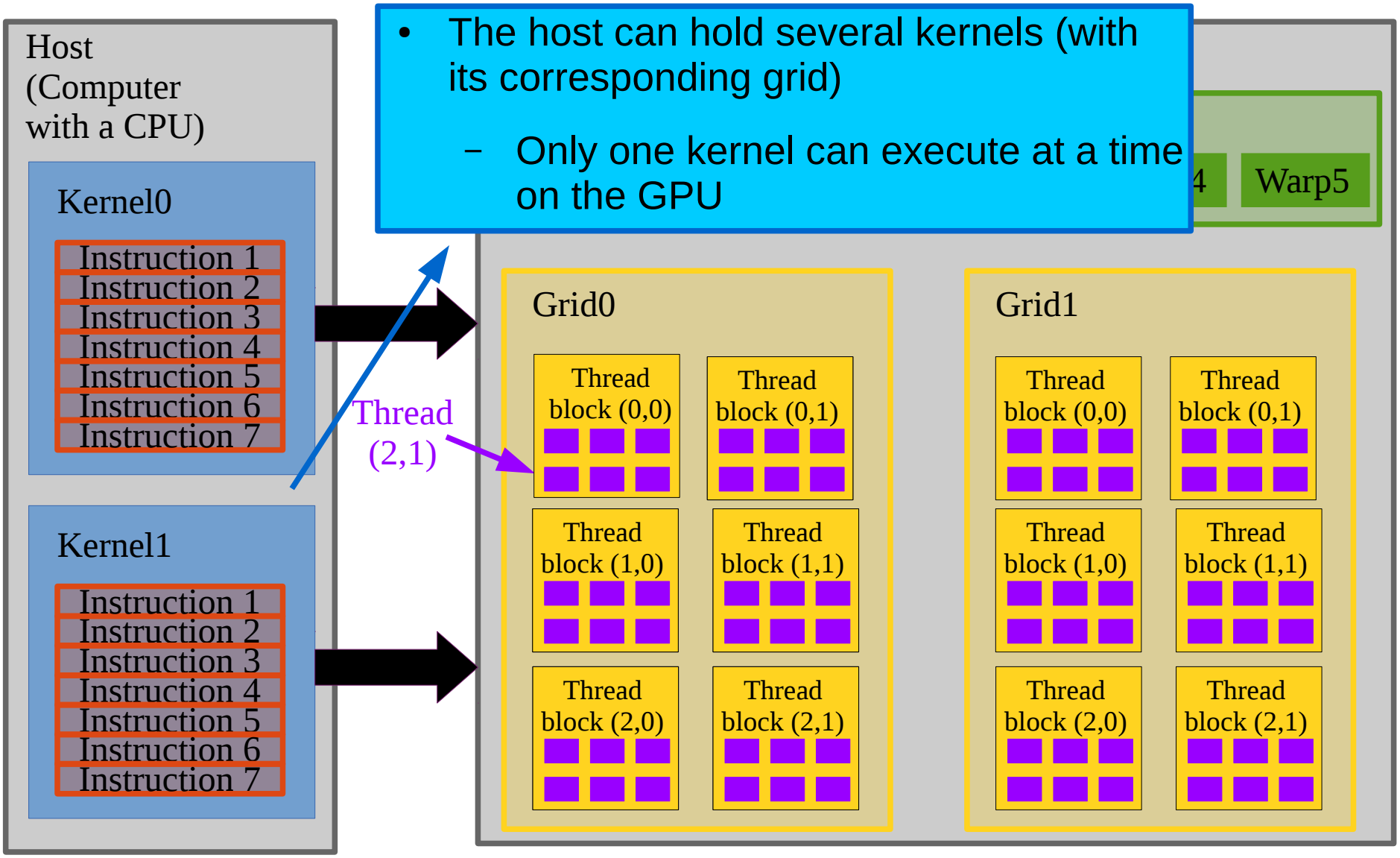


The GPU

- The threads are organised in blocks on a grid
  - The threads within a block have access to a shared memory
  - The execution of threads in a block can be synchronised

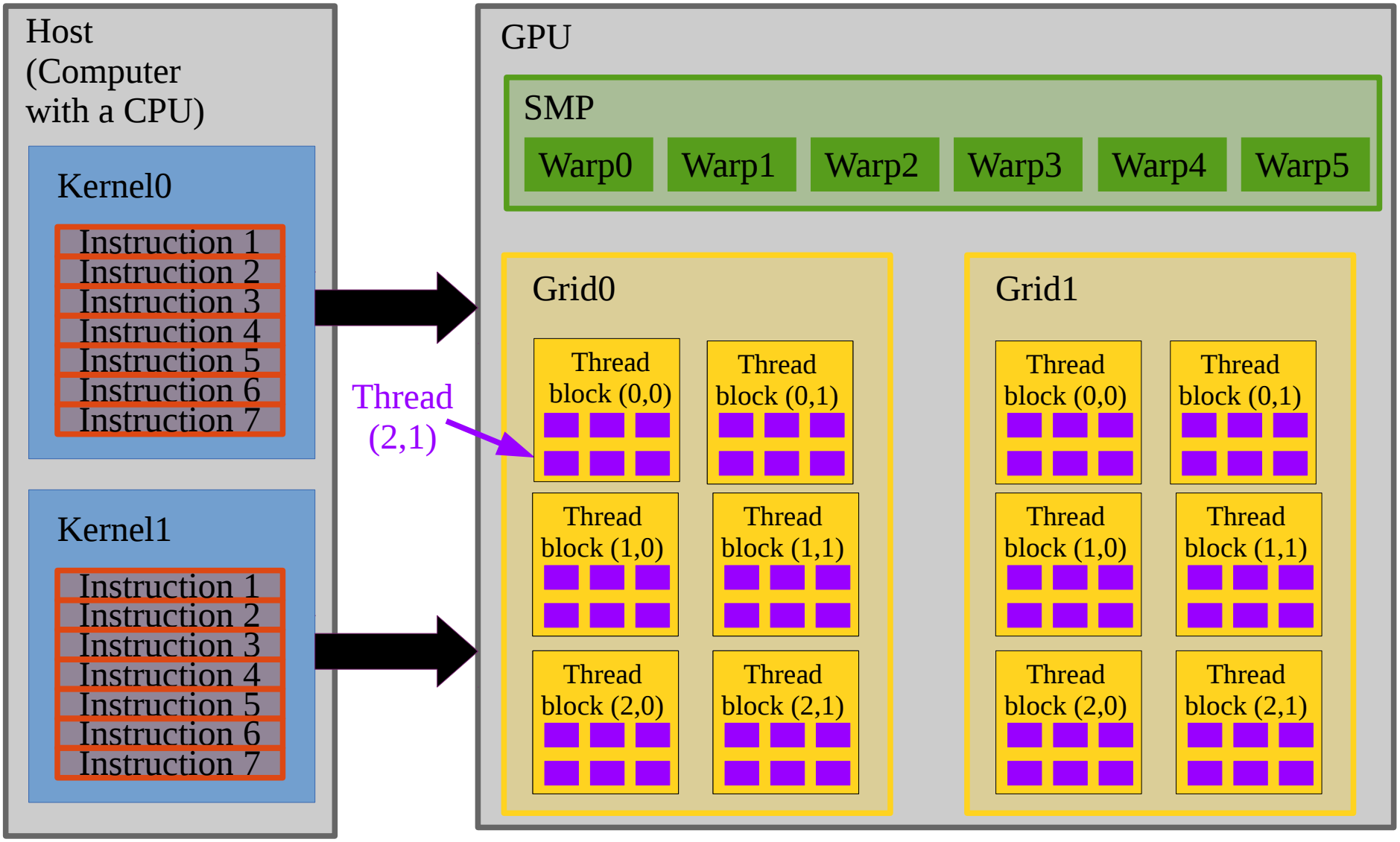


The GPU

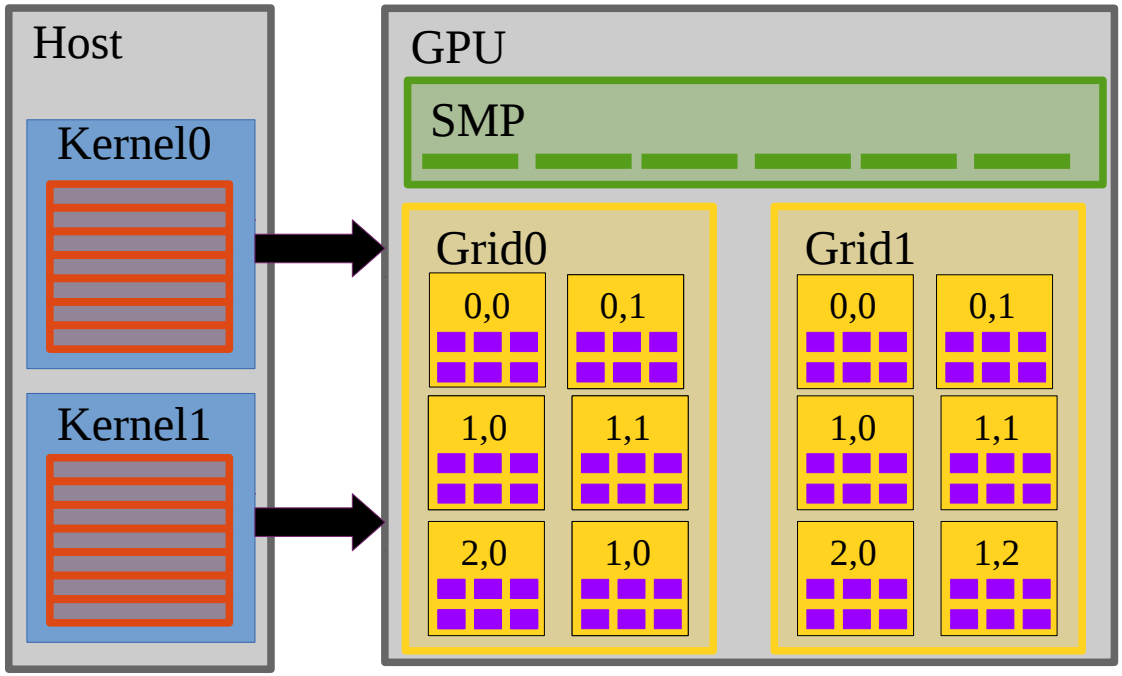


- The host can hold several kernels (with its corresponding grid)
  - Only one kernel can execute at a time on the GPU

The GPU


Host
(Computer
with a CPU)
Kernel0
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
Kernel1
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
Thread
(2,1)
GPU
SMP
Warp0
Warp1
Warp2
Warp3
Warp4
Warp5
Grid0
Thread block (0,0)
Thread block (0,1)
Thread block (1,0)
Thread block (1,1)
Thread block (2,0)
Thread block (2,1)
Grid1
Thread block (0,0)
Thread block (0,1)
Thread block (1,0)
Thread block (1,1)
Thread block (2,0)
Thread block (2,1)

Changing your mindset



- Think data parallelism, not execution parallelism
- Warps are most efficient if they are full → Each block has to be large enough (>32 / 64 , but limited to 512 - 1024)
- As for CPUs, the more local the memory, the more efficient
- Since the threads execute synchronously, avoid divergence in the execution (at least within the same block)

Example : The hard way

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>

int main(int argc, char *argv[]){
    Initialisation
    Computing
    Output

}
```

Nothing special to include, however the compiler has to be compatible (gcc → nvcc)

Example : The hard way

Initialisation

```
const int nPart = 500000;
double x[nPart];
double px[nPart];
double radius;
double angle;
for(int i = 0;i<nPart;++i){
    radius = (double)rand() / RAND_MAX;
    while(radius==0){
        radius = rand();
    }
    radius = sqrt(-2.0*log(radius));
    angle = (double)2.0*M_PI*rand() / RAND_MAX;
    x[i] = radius*sin(angle);
    px[i] = radius*cos(angle);
}
```

Copy the data from the host memory to the GPU memory

```
double* x_cuda;
double* px_cuda;
cudaMalloc(&x_cuda, nPart*sizeof(double));
cudaMalloc(&px_cuda, nPart*sizeof(double));
cudaMemcpy(x_cuda, x, nPart*sizeof(double), cudaMemcpyHostToDevice);
cudaMemcpy(px_cuda, px, nPart*sizeof(double), cudaMemcpyHostToDevice);
```

...

Example : The hard way

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>

__global__
void track(double* x, double* px,int nPart, double sinPhix,double cosPhix,double scale,double thres)
{
   int i = blockIdx.x*blockDim.x + threadIdx.x;
   double oldX = 0.0;
   double oldPx = 0.0;

   oldX = x[i];
   oldPx = px[i];
   x[i] = cosPhix*oldX + sinPhix*oldPx;
   px[i] = -sinPhix*oldX + cosPhix*oldPx;
   if(abs(x[i]) > thres){
      px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
   }
}

int main(int argc, char *argv[]){
```

Initialisation

```
   int threadsperblock = 512;
   int blockspergrid = ceil(nPart / threadsperblock);
   for(int turn = 0;turn<nTurn;++turn){
      track<<<blockspergrid,threadsperblock>>>(x_cuda,px_cuda,nPart,sinPhix,cosPhix,scale,thres);
   }
```

Output

```
}
```

Example : The hard way

Output

Need to copy back the memory to the host

```
cudaMemcpy(x, x_cuda, nPart*sizeof(double), cudaMemcpyDeviceToHost);
cudaMemcpy(px, px_cuda, nPart*sizeof(double), cudaMemcpyDeviceToHost);
cudaFree(x_cuda);
cudaFree(px_cuda);

FILE* file = fopen("phaseSpace.csv","w");
for(int i=0;i<nPart;++i){
    fprintf(file,"%E,%E\n",x[i],px[i]);
}
fclose(file);
```

Example : The hard way

```
Fri Nov  2 12:02:23 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.130                Driver Version: 384.130                   |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla C2075         Off  | 00000000:02:00.0  On |                    0 |
| 30%   65C    P0   149W /  N/A |    617MiB /  5296MiB |    100%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla C2075         Off  | 00000000:03:00.0 Off |                    0 |
| 30%   55C   P12    31W /  N/A |    513MiB /  5301MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla C2075         Off  | 00000000:83:00.0 Off |                    0 |
| 30%   54C   P12    32W /  N/A |    513MiB /  5301MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla C2075         Off  | 00000000:84:00.0 Off |                    0 |
| 30%   53C   P12    29W /  N/A |    513MiB /  5301MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     22338      C   ./phaseSpace_CUDA                            55MiB |
+-----------------------------------------------------------------------------+
```

- 277s on a single CPU
- 2.1s on the (9 years old) GPU

  → 132x faster!

Example : the easy ways

```
double oldX = 0.0;
double oldPx = 0.0;
#pragma acc parallel
for(int turn = 0;turn<nTurn;++turn){
   #pragma acc loop
    for(int i = 0;i<nPart;++i){
        oldX = x[i];
        oldPx = px[i];
        x[i] = cosPhix*oldX + sinPhix*oldPx;
        px[i] = -sinPhix*oldX + cosPhix*oldPx;
        if(abs(x[i]) > thres){
            px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
        }
    }
}
```

→ Compiler option -fopenacc

- With OpenACC, some simple instructions lets the compiler do the jobs

  → Similar performance can be achieved for several applications, without effort!

- Several libraries exists to speed up your code without effort (also in high level languages)

Applications

Applications

- Do's (key word: **large weakly interdependent data sets**)
  - Processing of large data sets (optics correction algorithms (for large machines), fast online data processing, machine learning, ...)
  - Solution of differential equation (time domain electromagnetic / mechanical / thermal simulation, beam evolution, ...)
  - Linear algebra (matrix manipulations, frequency domain electromagnetic / mechanical simulations, beam stability, ...)
  - Monte Carlo (particle – matter interactions, detectors, vacuum, ...)
  - Beam physics (multiparticle tracking, ...)
  - …

Applications

- Do's (key word: **large weakly interdependent data sets**)
  - Processing of large data sets (optics correction algorithms (for large machines), fast online data processing, machine learning, ...)
  - Solution of differential equation (time domain electromagnetic / mechanical / thermal simulation, beam evolution, ...)
  - Linear algebra (matrix manipulations, frequency domain electromagnetic / mechanical simulations, beam stability, ...)
  - Monte Carlo (particle – matter interactions, detectors, vacuum, ...)
  - Beam physics (multiparticle tracking, ...)
  - …
- Maybe's (key word: **large strongly interdependent data sets**)
  - Particle-in-cell
  - collective beam instabilities

Applications

- Do's (key word: **large weakly interdependent data sets**)
  - Processing of large data sets (optics correction algorithms (for large machines), fast online data processing, machine learning, ...)
  - Solution of differential equation (time domain electromagnetic / mechanical / thermal simulation, beam evolution, ...)
  - Linear algebra (matrix manipulations, frequency domain electromagnetic / mechanical simulations, beam stability, ...)
  - Monte Carlo (particle – matter interactions, detectors, vacuum, ...)
  - Beam physics (multiparticle tracking, ...)
  - …
- Maybe's (key word: **large strongly interdependent data sets**)
  - Particle-in-cell
  - collective beam instabilities
- Don'ts (key word: **small data sets**)
  - Processing of small data sets
  - Single particle tracking, optics calculations
  - Heterogeneous tasks

Applications

- Do's (key word: **large weakly interdependent data sets**)
  - Processing of large data sets (optics correction algorithms (for large machines), fast online data processing, machine learning, ...)
  - Solution of differential equation (time domain electromagnetic / mechanical / thermal simulation, beam evolution, ...)
  - Linear algebra (matrix manipulations, frequency domain electromagnetic / mechanical simulations, beam stability, ...)
  - Monte Carlo (particle – matter interactions, detectors, vacuum, ...)
  - Beam physics (multiparticle tracking, ...)
  - …
- Maybe's (key word: **large strongly interdependent data sets**)
  - Particle-in-cell
  - collective beam instabilities
- Don'ts (key word: **small data sets**)
  - Processing of small data sets
  - Single particle tracking, optics calculations
  - Heterogeneous tasks

- Since 2012, about ~10 IPAC papers/year are dedicated to GPU acceleration (www.jacow.org)

What should I buy ?

- Keep in mind when comparing the performance of different hardware :
  - GPU's currently offer much more core than CPUs, but are not as flexible (Single Instruction, multiple data)
  - For the same price, you usually get less CPU's, but are fully flexible (multiple instructions, multiple data)

What should I buy ?

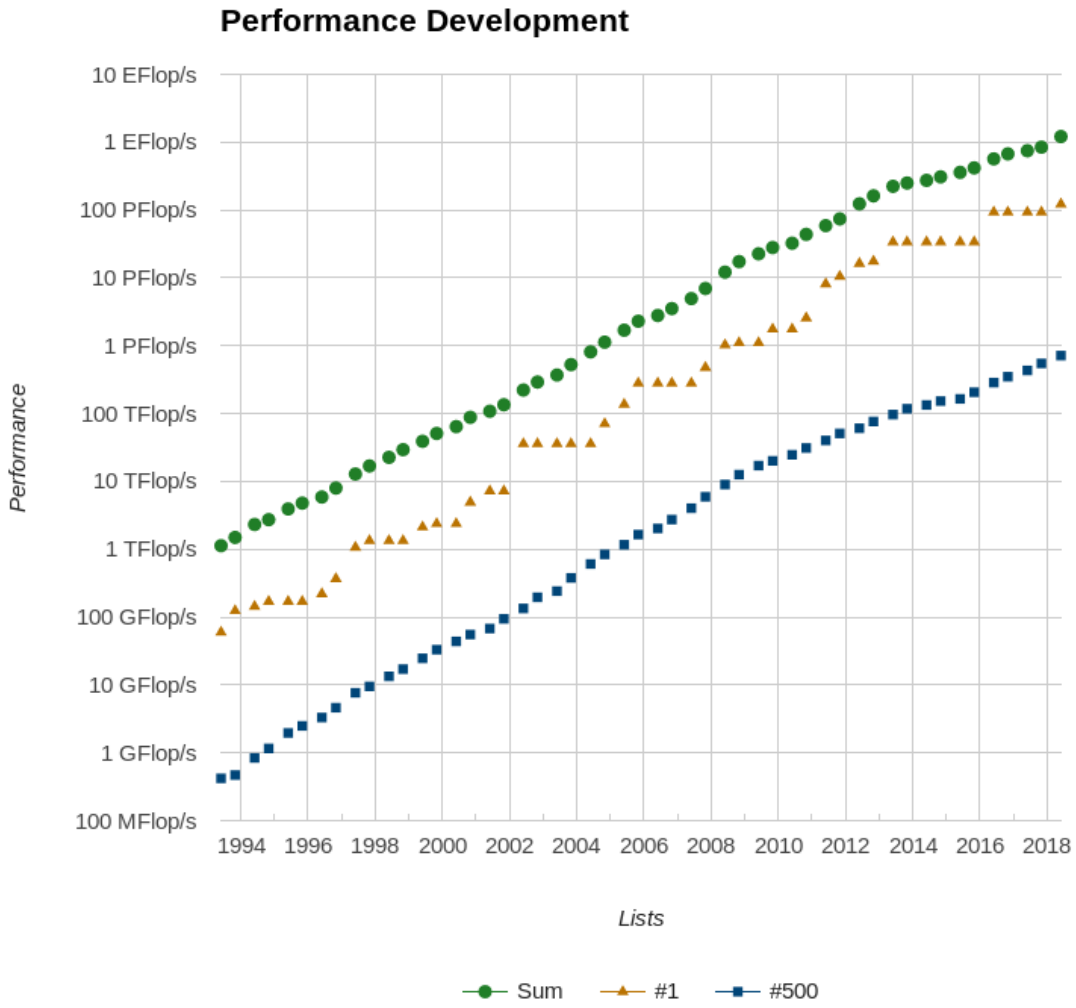
- Keep in mind when comparing the performance of different hardware :
  - GPU's currently offer much more core than CPUs, but are not as flexible (Single Instruction, multiple data)
  - For the same price, you usually get less CPU's, but are fully flexible (multiple instructions, multiple data)
  - Memory bound problems can't be accelerated with multiple CPUs, thanks to the large memory bandwidth, GPU are usually more suited

What should I buy ?

- Keep in mind when comparing the performance of different hardware :
  - GPU's currently offer much more core than CPUs, but are not as flexible (Single Instruction, multiple data)
  - For the same price, you usually get less CPU's, but are fully flexible (multiple instructions, multiple data)
  - Memory bound problems can't be accelerated with multiple CPUs, thanks to the large memory bandwidth, GPU are usually more suited

→ Always keep an eye on new technologies... (See e.g. Intel Xeon Phi, a single CPU with up to 72 cores, with vectorization of 8 doubles)

About Moore's Law



- The computing power of the most powerful machine in the world does keep increasing !

→ Computer clusters

Source : http://www.top500.org/

The computer cluster

- A set of computers (now called nodes)

Illustration : MichiganTech

The computer cluster

- A set of computers (now called nodes)
- The nodes are connected together with a fast link (→ ~Gb/s)

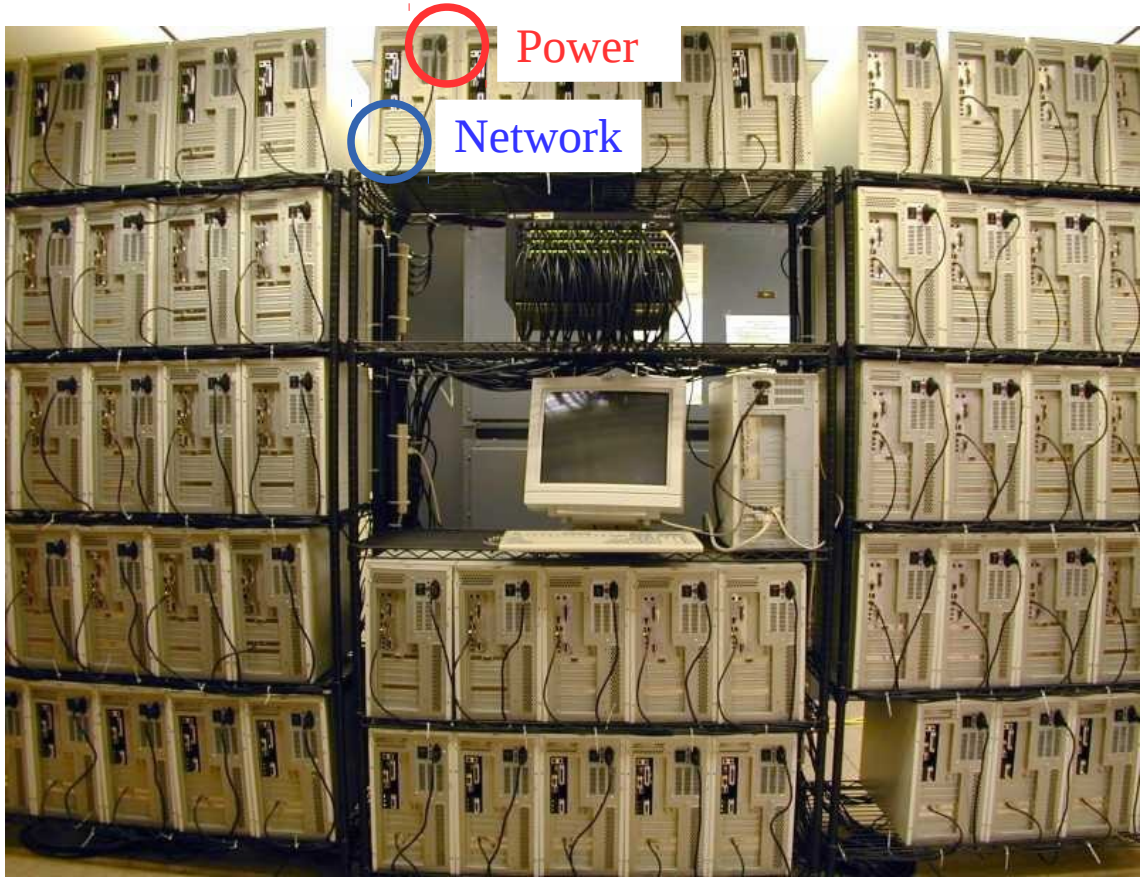


Illustration : MichiganTech

The computer cluster

- A set of computers (now called nodes)
- The nodes are connected together with a fast link (→ ~Gb/s)
- Only one master node is equiped with an interface → the set of computers acts as a single entity : a Cluster

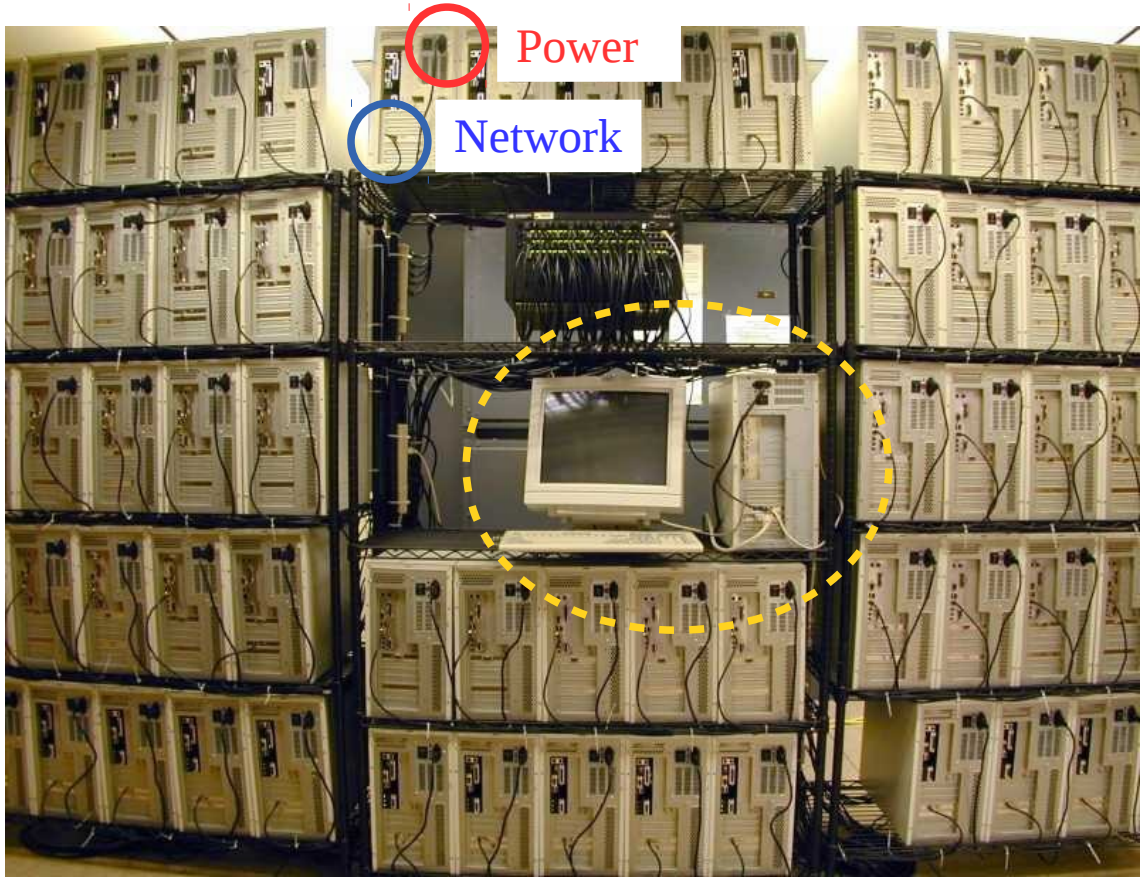


Illustration : MichiganTech

The computer cluster

- A set of computers (now called nodes)
- The nodes are connected together with a fast link (→ ~Gb/s)
- Only one master node is equiped with an interface → the set of computers acts as a single entity : a Cluster
- Various processes of a program can be run across computers

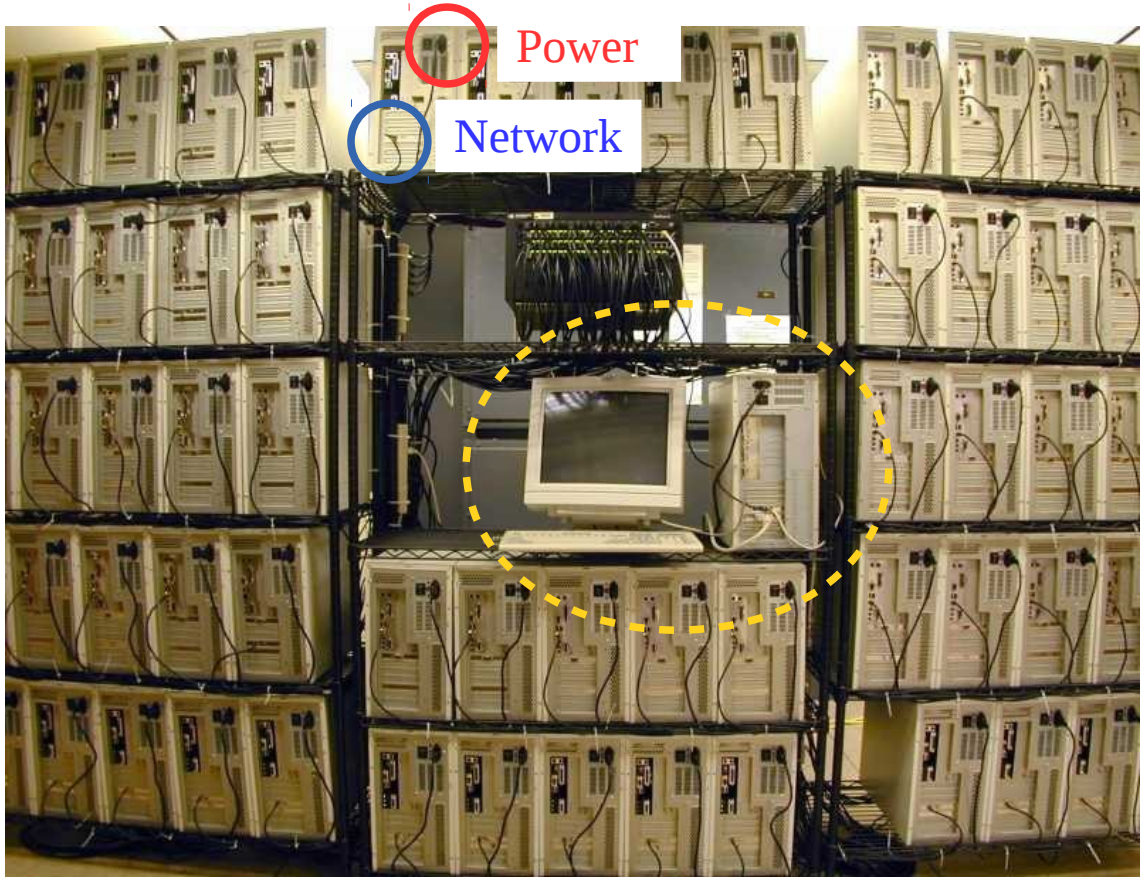


Illustration : MichiganTech

The computer cluster

- A set of computers (now called nodes)
- The nodes are connected together with a fast link (→ ~Gb/s)
- Only one master node is equiped with an interface → the set of computers acts as a single entity : a Cluster
- Various processes of a program can be run across computers
- The memory (RAM) can no longer be shared

  → Distributed memory

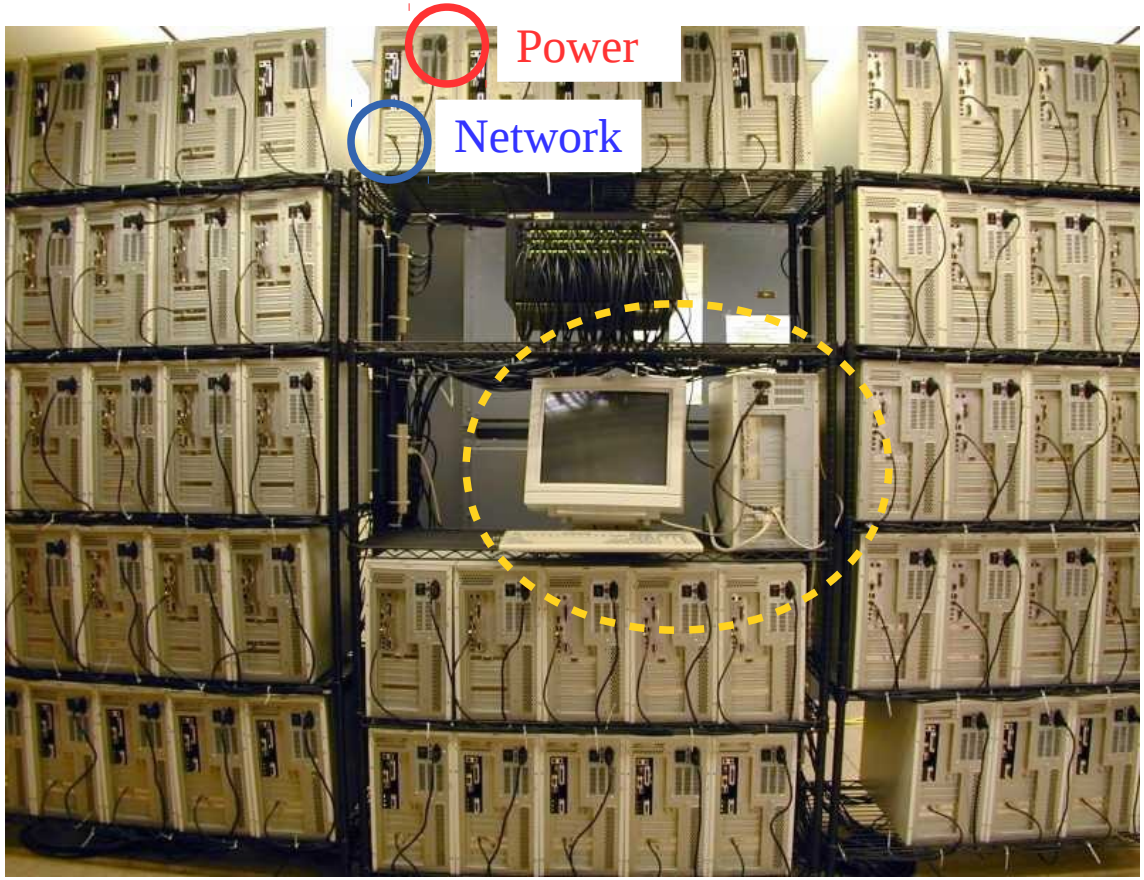


Illustration : MichiganTech

The computer cluster

- A set of computers (now called nodes)
- The nodes are connected together with a fast link (→ ~Gb/s)
- Only one master node is equiped with an interface → the set of computers acts as a single entity : a Cluster
- Various processes of a program can be run across computers
- The memory (RAM) can no longer be shared

  → Distributed memory

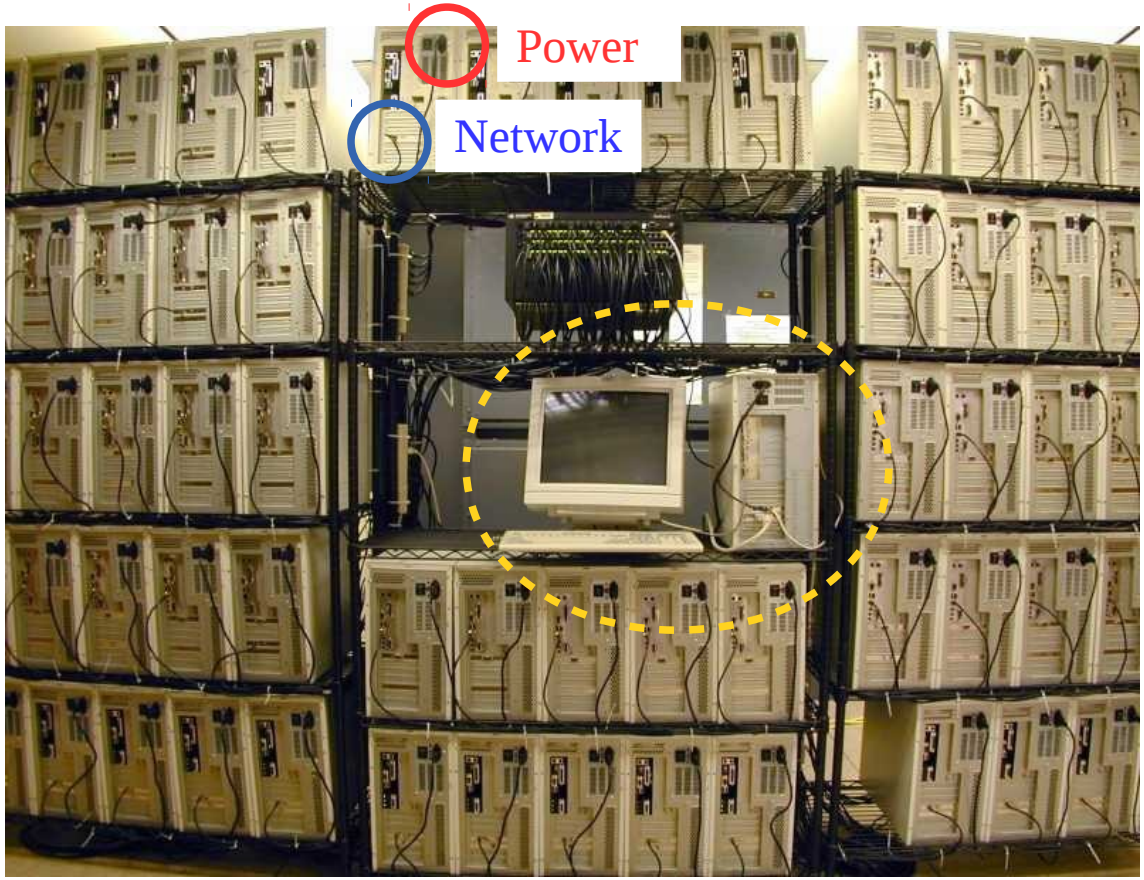


Illustration : MichiganTech

- How can we (programmers) handle this ?

Largest clusters

- Piz Daint (Swiss National Supercomputing Centre)
- 361,760 cores
- ~20 Pflop/s
- ~2 MW
- Largest supercomputer in Europe, 6th in the world

Largest clusters

- Piz Daint (Swiss National Supercomputing Centre)
- 361,760 cores
- ~20 Pflop/s
- ~2 MW
- Largest supercomputer in Europe, 6th in the world

- Summit (Oak ridge National Lab. USA)
- 2,282,544 cores
- ~120 Pflop/s
- ~8 MW
- Largest supercomputer in the world

Largest clusters

- Piz Daint (Swiss National Supercomputing Centre)
- 361,760 cores
- ~20 Pflop/s
- ~2 MW
- Largest supercomputer in Europe, 6th in the world

- Summit (Oak ridge National Lab. USA)
- 2,282,544 cores
- ~120 Pflop/s
- ~8 MW
- Largest supercomputer in the world

Source : http://www.top500.org/ (June 2018)

Most playful cluster

Most playful cluster

- Boss, I need 200 Playstation 3 for my research

Most playful cluster

- Boss, I need 200 Playstation 3 for my research
- Sure, no problem !

Most playful cluster

- Boss, I need 200 Playstation 3 for my research

- Sure, no problem !

… said no boss ever ?

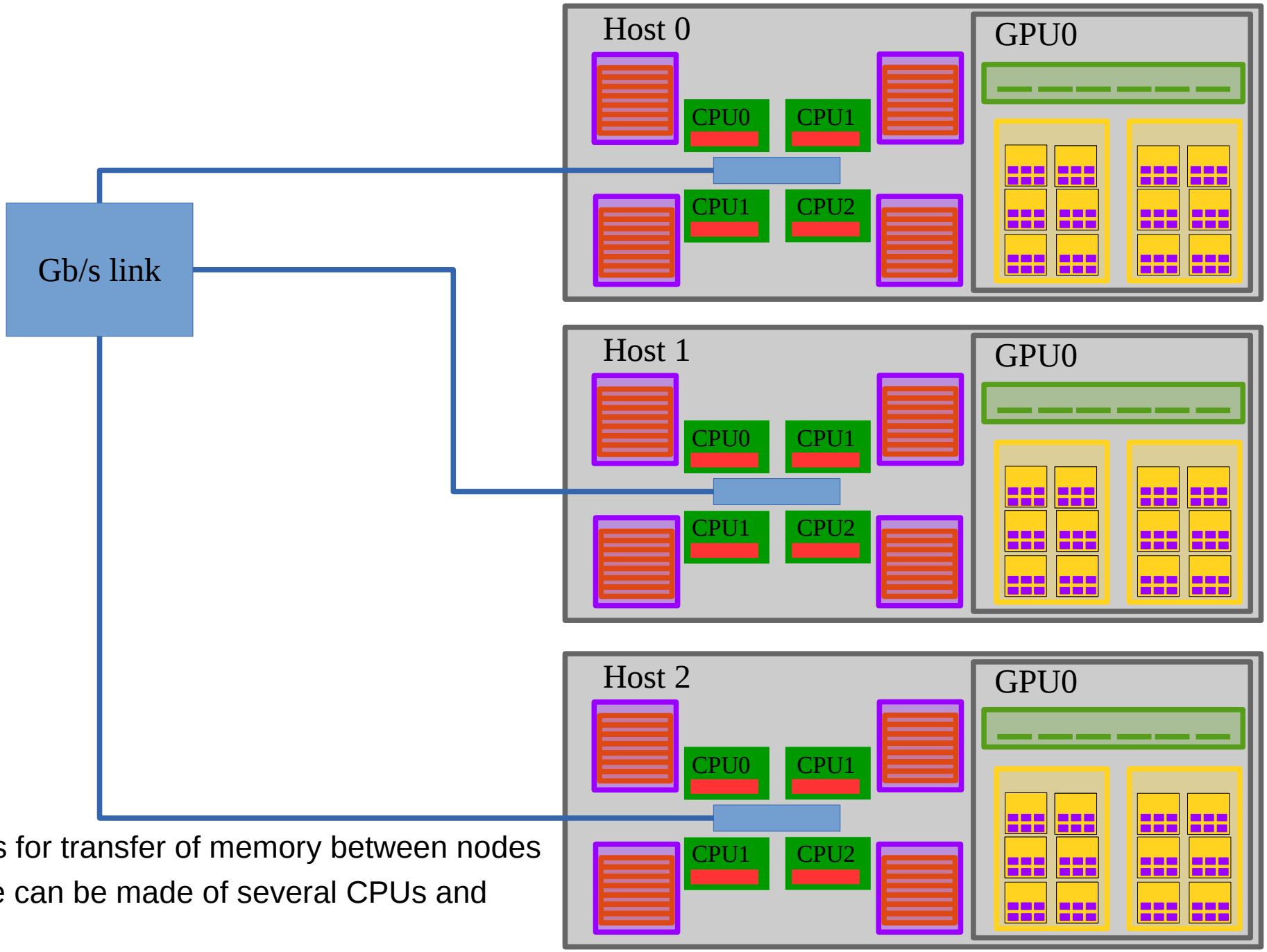
Most playful cluster

- Boss, I need 200 Playstation 3 for my research

- Sure, no problem !

… said no boss ever ?

The message passing interface



- MPI allows for transfer of memory between nodes
- Each node can be made of several CPUs and GPUs

The message passing interface

- A single code is compiled with a dedicated compiler (mpicc)
- The code is executed by a given number of processes specified by the running command
  - mpirun -np **800** *executable_name*

The message passing interface

- A single code is compiled with a dedicated compiler (mpicc)
- The code is executed by a given number of processes specified by the running command
  - mpirun -np **800** *executable_name*
- Example on CERN's HPC-BATCH

```
15:46:50 xbuffat slurmgate04 /hpcscratch/user/xbuffat/CAS2018_MPIexample
→ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             33359 batch-lon StartMPP  tpolzin  R       3:54      8 hpc[061-062,064-069]
             33339 batch-lon     bash dzambell  R    1:41:30      2 hpc[026-027]
             33341 batch-sho       gs delkhech  R    1:36:10      1 hpc006
             33340 batch-sho       gs delkhech  R    1:38:58      1 hpc005
             33345 batch-sho       gs delkhech  R    1:08:28      1 hpc009
             33344 batch-sho       gs delkhech  R    1:13:33      1 hpc008
             33343 batch-sho       gs delkhech  R    1:14:17      1 hpc007
             33348 batch-lon  dima.sh delkhech  R      56:09      1 hpc028
             26544 batch-lon   triMag  thibaut  R 41-07:04:16     10 hpc[075,078-086]
             33360   be-long phaseSpa  xbuffat  R       0:19     20 hpc-be[004,018,027,029,046,
058,061-062,065,071,073,081-082,089-090,093,097-098,132,137]
             32530   be-long HCMOctop  thibaut  R 7-02:14:14      5 hpc-be[003,024,028,033,036]
             32529   be-long HCMOctop  thibaut  R 7-02:15:02      5 hpc-be[057,083,091-092,105]
             32379   be-long HCMOctop  thibaut  R 8-04:48:40      5 hpc-be[005,009,015,034,044]
             32947 batch-lon     Simu  thibaut  R 4-04:37:31      2 hpc[185-186]
             32982 batch-lon   1_3Ghz  thibaut  R 4-01:11:19     10 hpc[051-060]
             32981 batch-lon   1_3Ghz  thibaut  R 4-01:13:31     10 hpc[016-025]
             32969 batch-lon     Simu  thibaut  R 4-02:34:41      2 hpc[014-015]
```

The message passing interface

- A single code is compiled with a dedicated compiler (mpicc)
- The code is executed by a given number of processes specified by the running command
  - mpirun -np **800** *executable_name*
- Example on CERN's HPC-BATCH

```
15:46:50 xbuffat slurmgate04 /hpcscratch/user/xbuffat/CAS2018_MPIexample
→ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             33359 batch-lon StartMPP  tpolzin  R       3:54      8 hpc[061-062,064-069]
             33339 batch-lon     bash dzambell  R    1:41:30      2 hpc[026-027]
             33341 batch-sho       gs delkhech  R    1:36:10      1 hpc006
             33340 batch-sho       gs delkhech  R    1:38:58      1 hpc005
             33345 batch-sho       gs delkhech  R    1:08:28      1 hpc009
             33344 batch-sho       gs delkhech  R    1:13:33      1 hpc008
             33343 batch-sho       gs delkhech  R    1:14:17      1 hpc007
             33348 batch-lon  dima.sh delkhech  R      56:09      1 hpc028
             26544 batch-lon   triMag  thibaut  R 41-07:04:16     10 hpc[075,078-086]
             33360   be-long phaseSpa  xbuffat  R       0:19     20 hpc-be[004,018,027,029,046,
058,061-062,065,071,073,081-082,089-090,093,097-098,132,137]
             32529   be-long HCMOctop  thibaut  R 7-02:15:02      5 hpc-be[057,083,091-092,105]
             32379   be-long HCMOctop  thibaut  R 8-04:48:40      5 hpc-be[005,009,015,034,044]
             32947 batch-lon     Simu  thibaut  R 4-04:37:31      2 hpc[185-186]
             32982 batch-lon   1_3Ghz  thibaut  R 4-01:11:19     10 hpc[051-060]
             32981 batch-lon   1_3Ghz  thibaut  R 4-01:13:31     10 hpc[016-025]
             32969 batch-lon     Simu  thibaut  R 4-02:34:41      2 hpc[014-015]
```

The message passing interface

- A single code is compiled with a dedicated compiler (mpicc)
- The code is executed by a given number of processes specified by the running command
  - mpirun -np **800** *executable_name*
- Example on CERN's HPC-BATCH

```
15:46:50 xbuffat slurmgate04 /hpcscratch/user/xbuffat/CAS2018_MPIexample
→ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             33359 batch-lon StartMPP  tpolzin  R       3:54      8 hpc[061-062,064-069]
             33339 batch-lon     bash dzambell  R    1:41:30      2 hpc[026-027]
             33341 batch-sho       gs delkhech  R    1:36:10      1 hpc006
             33340 batch-sho       gs delkhech  R    1:38:58      1 hpc005
             33345 batch-sho       gs delkhech  R    1:08:28      1 hpc009
             33344 batch-sho       gs delkhech  R    1:13:33      1 hpc008
             33343 batch-sho       gs delkhech  R    1:14:17      1 hpc007
             33348 batch-lon  dima.sh delkhech  R      56:09      1 hpc028
             26544 batch-lon   triMag  thibaut  R 41-07:04:16     10 hpc[075,078-086]
             33360   be-long phaseSpa  xbuffat  R       0:19     20 hpc-be[004,018,027,029,046,
058,061-062,065,071,073,081-082,089-090,093,097-098,132,137]
             32529   be-long HCMOctop  thibaut  R 7-02:15:02      5 hpc-be[057,083,091-092,105]
             32379   be-long HCMOctop  thibaut  R 8-04:48:40      5 hpc-be[005,009,015,034,044]
             32947 batch-lon     Simu  thibaut  R 4-04:37:31      2 hpc[185-186]
             32982 batch-lon   1_3Ghz  thibaut  R 4-01:11:19     10 hpc[051-060]
             32981 batch-lon   1_3Ghz  thibaut  R 4-01:13:31     10 hpc[016-025]
             32969 batch-lon     Simu  thibaut  R 4-02:34:41      2 hpc[014-015]
```

```
top - 15:47:21 up 26 days, 23:33,  1 user,  load average: 18.09, 4.54, 1.56
Tasks: 525 total,  42 running, 483 sleeping,   0 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13174201+total, 10142152+free, 16556744 used, 13763752 buff/cache
KiB Swap: 67108860 total, 67108860 free,        0 used. 11216652+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
32757 xbuffat   20   0  590008  23500   4256 R 100.0  0.0   0:34.96 phaseSpace_MPI
32758 xbuffat   20   0  590064  23264   3976 R 100.0  0.0   0:34.69 phaseSpace_MPI
32759 xbuffat   20   0  590060  23260   3972 R 100.0  0.0   0:34.75 phaseSpace_MPI
32760 xbuffat   20   0  590060  23276   3988 R 100.0  0.0   0:34.75 phaseSpace_MPI
32762 xbuffat   20   0  590060  23284   3992 R 100.0  0.0   0:34.70 phaseSpace_MPI
32766 xbuffat   20   0  590060  23252   3964 R 100.0  0.0   0:34.72 phaseSpace_MPI
32767 xbuffat   20   0  590060  23276   3988 R 100.0  0.0   0:34.67 phaseSpace_MPI
32768 xbuffat   20   0  590064  23264   3976 R 100.0  0.0   0:34.75 phaseSpace_MPI
32769 xbuffat   20   0  590064  23252   3964 R 100.0  0.0   0:34.70 phaseSpace_MPI
32770 xbuffat   20   0  590064  23260   3972 R 100.0  0.0   0:34.76 phaseSpace_MPI
32771 xbuffat   20   0  590060  23280   3992 R 100.0  0.0   0:34.69 phaseSpace_MPI
32772 xbuffat   20   0  590064  23264   3976 R 100.0  0.0   0:34.75 phaseSpace_MPI
32773 xbuffat   20   0  590060  23252   3964 R 100.0  0.0   0:34.76 phaseSpace_MPI
32774 xbuffat   20   0  590064  23280   3992 R 100.0  0.0   0:34.75 phaseSpace_MPI
32775 xbuffat   20   0  590064  23300   3964 R 100.0  0.0   0:34.75 phaseSpace_MPI
32776 xbuffat   20   0  590060  23320   3984 R 100.0  0.0   0:34.74 phaseSpace_MPI
32777 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.71 phaseSpace_MPI
32778 xbuffat   20   0  590060  23296   3960 R 100.0  0.0   0:34.76 phaseSpace_MPI
32782 xbuffat   20   0  590064  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32785 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32786 xbuffat   20   0  590064  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32787 xbuffat   20   0  590064  23296   3960 R 100.0  0.0   0:34.71 phaseSpace_MPI
32788 xbuffat   20   0  590064  23304   3968 R 100.0  0.0   0:34.76 phaseSpace_MPI
32789 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.75 phaseSpace_MPI
32792 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32793 xbuffat   20   0  590064  23304   3964 R 100.0  0.0   0:34.76 phaseSpace_MPI
32794 xbuffat   20   0  590060  23304   3968 R 100.0  0.0   0:34.74 phaseSpace_MPI
32795 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32761 xbuffat   20   0  590060  23276   3988 R  99.7  0.0   0:34.70 phaseSpace_MPI
32763 xbuffat   20   0  590064  23252   3964 R  99.7  0.0   0:34.64 phaseSpace_MPI
32765 xbuffat   20   0  590064  23232   3944 R  99.7  0.0   0:34.71 phaseSpace_MPI
32779 xbuffat   20   0  590064  23304   3968 R  99.7  0.0   0:34.71 phaseSpace_MPI
32780 xbuffat   20   0  590064  23296   3960 R  99.7  0.0   0:34.70 phaseSpace_MPI
32781 xbuffat   20   0  590060  23312   3976 R  99.7  0.0   0:34.71 phaseSpace_MPI
32783 xbuffat   20   0  590064  23312   3976 R  99.7  0.0   0:34.76 phaseSpace_MPI
32784 xbuffat   20   0  590060  23304   3968 R  99.7  0.0   0:34.75 phaseSpace_MPI
32791 xbuffat   20   0  590064  23296   3960 R  99.7  0.0   0:34.77 phaseSpace_MPI
32796 xbuffat   20   0  590060  23300   3964 R  99.7  0.0   0:34.72 phaseSpace_MPI
32764 xbuffat   20   0  590064  23264   3972 R  99.3  0.0   0:34.64 phaseSpace_MPI
32790 xbuffat   20   0  590060  23292   3956 R  99.3  0.0   0:34.70 phaseSpace_MPI
228257 root     20   0   23916   4248   1696 S   1.0  0.0  41:04.14 lemon-sensor-li
  2772 ceph     20   0 5010528   4.0g  20080 S   0.7  3.2 288:37.01 ceph-osd
  2991 ceph     20   0 5321264   4.2g  20008 S   0.7  3.4 314:29.28 ceph-osd
```

The message passing interface

- A single code is compiled with a dedicated compiler (mpicc)
- The code is executed by a given number of processes specified by the running command
  - mpirun -np **800** *executable_name*
- Example on CERN's HPC-BATCH

```
15:46:50 xbuffat slurmgate04 /hpcscratch/user/xbuffat/CAS2018_MPIexample
→ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             33359 batch-lon StartMPP  tpolzin  R       3:54      8 hpc[061-062,064-069]
             33339 batch-lon     bash dzambell  R    1:41:30      2 hpc[026-027]
             33341 batch-sho       gs delkhech  R    1:36:10      1 hpc006
             33340 batch-sho       gs delkhech  R    1:38:58      1 hpc005
             33345 batch-sho       gs delkhech  R    1:08:28      1 hpc009
             33344 batch-sho       gs delkhech  R    1:13:33      1 hpc008
             33343 batch-sho       gs delkhech  R    1:14:17      1 hpc007
             33348 batch-lon  dima.sh delkhech  R      56:09      1 hpc028
             26544 batch-lon   triMag  thibaut  R 41-07:04:16     10 hpc[075,078-086]
             33360   be-long phaseSpa  xbuffat  R       0:19     20 hpc-be[004,018,027,029,046,
058,061-062,065,071,073,081-082,089-090,093,097-098,132,137]
             32529   be-long HCMOctop  thibaut  R 7-02:15:02      5 hpc-be[057,083,091-092,105]
             32379   be-long HCMOctop  thibaut  R 8-04:48:40      5 hpc-be[005,009,015,034,044]
             32947 batch-lon     Simu  thibaut  R 4-04:37:31      2 hpc[185-186]
             32982 batch-lon   1_3Ghz  thibaut  R 4-01:11:19     10 hpc[051-060]
             32981 batch-lon   1_3Ghz  thibaut  R 4-01:13:31     10 hpc[016-025]
             32969 batch-lon     Simu  thibaut  R 4-02:34:41      2 hpc[014-015]
```

```
top - 15:47:21 up 26 days, 23:33,  1 user,  load average: 18.09, 4.54, 1.56
Tasks: 525 total,  42 running, 483 sleeping,   0 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13174201+total, 10142152+free, 16556744 used, 13763752 buff/cache
KiB Swap: 67108860 total, 67108860 free,        0 used. 11216652+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
32757 xbuffat   20   0  590008  23500   4256 R 100.0  0.0   0:34.96 phaseSpace_MPI
32758 xbuffat   20   0  590064  23264   3976 R 100.0  0.0   0:34.69 phaseSpace_MPI
32759 xbuffat   20   0  590060  23260   3972 R 100.0  0.0   0:34.75 phaseSpace_MPI
32760 xbuffat   20   0  590060  23276   3988 R 100.0  0.0   0:34.75 phaseSpace_MPI
32762 xbuffat   20   0  590060  23284   3992 R 100.0  0.0   0:34.70 phaseSpace_MPI
32766 xbuffat   20   0  590060  23252   3964 R 100.0  0.0   0:34.72 phaseSpace_MPI
32767 xbuffat   20   0  590060  23276   3988 R 100.0  0.0   0:34.67 phaseSpace_MPI
32768 xbuffat   20   0  590064  23264   3976 R 100.0  0.0   0:34.75 phaseSpace_MPI
32769 xbuffat   20   0  590064  23252   3964 R 100.0  0.0   0:34.70 phaseSpace_MPI
32770 xbuffat   20   0  590064  23260   3972 R 100.0  0.0   0:34.76 phaseSpace_MPI
32771 xbuffat   20   0  590060  23280   3992 R 100.0  0.0   0:34.69 phaseSpace_MPI
32772 xbuffat   20   0  590064  23264   3976 R 100.0  0.0   0:34.75 phaseSpace_MPI
32773 xbuffat   20   0  590060  23252   3964 R 100.0  0.0   0:34.76 phaseSpace_MPI
32774 xbuffat   20   0  590064  23280   3992 R 100.0  0.0   0:34.75 phaseSpace_MPI
32775 xbuffat   20   0  590064  23300   3964 R 100.0  0.0   0:34.75 phaseSpace_MPI
32776 xbuffat   20   0  590060  23320   3984 R 100.0  0.0   0:34.74 phaseSpace_MPI
32777 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.71 phaseSpace_MPI
32778 xbuffat   20   0  590060  23296   3960 R 100.0  0.0   0:34.76 phaseSpace_MPI
32782 xbuffat   20   0  590064  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32785 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32786 xbuffat   20   0  590064  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32787 xbuffat   20   0  590064  23296   3960 R 100.0  0.0   0:34.71 phaseSpace_MPI
32788 xbuffat   20   0  590064  23304   3968 R 100.0  0.0   0:34.76 phaseSpace_MPI
32789 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.75 phaseSpace_MPI
32792 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32793 xbuffat   20   0  590064  23304   3964 R 100.0  0.0   0:34.76 phaseSpace_MPI
32794 xbuffat   20   0  590060  23304   3968 R 100.0  0.0   0:34.74 phaseSpace_MPI
32795 xbuffat   20   0  590060  23300   3964 R 100.0  0.0   0:34.74 phaseSpace_MPI
32761 xbuffat   20   0  590060  23276   3988 R  99.7  0.0   0:34.70 phaseSpace_MPI
32763 xbuffat   20   0  590064  23252   3964 R  99.7  0.0   0:34.64 phaseSpace_MPI
32765 xbuffat   20   0  590064  23232   3944 R  99.7  0.0   0:34.71 phaseSpace_MPI
32779 xbuffat   20   0  590064  23304   3968 R  99.7  0.0   0:34.71 phaseSpace_MPI
32780 xbuffat   20   0  590064  23296   3960 R  99.7  0.0   0:34.70 phaseSpace_MPI
32781 xbuffat   20   0  590060  23312   3976 R  99.7  0.0   0:34.71 phaseSpace_MPI
32783 xbuffat   20   0  590064  23312   3976 R  99.7  0.0   0:34.76 phaseSpace_MPI
32784 xbuffat   20   0  590060  23304   3968 R  99.7  0.0   0:34.75 phaseSpace_MPI
32791 xbuffat   20   0  590064  23296   3960 R  99.7  0.0   0:34.77 phaseSpace_MPI
32796 xbuffat   20   0  590060  23300   3964 R  99.7  0.0   0:34.72 phaseSpace_MPI
32764 xbuffat   20   0  590064  23264   3972 R  99.3  0.0   0:34.64 phaseSpace_MPI
32790 xbuffat   20   0  590060  23292   3956 R  99.3  0.0   0:34.70 phaseSpace_MPI
228257 root     20   0   23916   4248   1696 S   1.0  0.0  41:04.14 lemon-sensor-li
  2772 ceph     20   0 5010528   4.0g  20080 S   0.7  3.2 288:37.01 ceph-osd
  2991 ceph     20   0 5321264   4.2g  20008 S   0.7  3.4 314:29.28 ceph-osd
```

- MPI defines how these processes can exchange data through the fast link
- Several implementations are available : OpenMPI, IntelMPI, MPICH, MVAPICH2,...

Example: The hard way

```
#include<math.h>
#include<cstdio>
#include<stdlib.h>
#include “mpi.h”

int main(int argc, char *argv[]){
    Initialisation
    Computing
    Output
}
```

Example : The hard way

Initialisation

```
int main(int argc, char *argv[]){
   int TAGREADY = 0;
   int TAGX = 1;
   int TAGPX = 2;
   const int nPart = 500000;
   double x[nPart];
   double px[nPart];
   MPI_Init(&argc, &argv);
   int myRank;
   int activeProcs;
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
   MPI_Comm_size(MPI_COMM_WORLD, &activeProcs);
   int chunkSize = ceil(nPart/activeProcs);
   if(myRank == 0){
      double radius;
      double angle;
      for(int i = 0;i<nPart;++i){
         radius = (double)rand() / RAND_MAX;
         while(radius==0){
            radius = rand();
         }
         radius = sqrt(-2.0*log(radius));
         angle = (double)2.0*M_PI*rand() / RAND_MAX;
         x[i] = radius*sin(angle);
         px[i] = radius*cos(angle);
      }

      for(int iProc=1;iProc<activeProcs;++iProc){
         MPI_Send(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGX, MPI_COMM_WORLD);
         MPI_Send(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGPX, MPI_COMM_WORLD);
      }
   } else{
      MPI_Status status;
      MPI_Recv(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD, &status);
      MPI_Recv(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD, &status);
   }
..
```

Example : The hard way

Initialisation

```
int main(int argc, char *argv[]){
   int TAGREADY = 0;
   int TAGX = 1;
   int TAGPX = 2;
   const int nPart = 500000;
   double x[nPart];
   double px[nPart];
   MPI_Init(&argc, &argv);
   int myRank;
   int activeProcs;
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
   MPI_Comm_size(MPI_COMM_WORLD, &activeProcs);
   int chunkSize = ceil(nPart/activeProcs);
   if(myRank == 0){
      double radius;
      double angle;
      for(int i = 0;i<nPart;++i){
         radius = (double)rand() / RAND_MAX;
         while(radius==0){
            radius = rand();
         }
         radius = sqrt(-2.0*log(radius));
         angle = (double)2.0*M_PI*rand() / RAND_MAX;
         x[i] = radius*sin(angle);
         px[i] = radius*cos(angle);
      }

      for(int iProc=1;iProc<activeProcs;++iProc){
         MPI_Send(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGX, MPI_COMM_WORLD);
         MPI_Send(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGPX, MPI_COMM_WORLD);
      }
   } else{
      MPI_Status status;
      MPI_Recv(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD, &status);
      MPI_Recv(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD, &status);
   }
..
```

Example : The hard way

Initialisation

```
int main(int argc, char *argv[]){
   int TAGREADY = 0;
   int TAGX = 1;
   int TAGPX = 2;
   const int nPart = 500000;
   double x[nPart];
   double px[nPart];
   MPI_Init(&argc, &argv);
   int myRank;
   int activeProcs;
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
   MPI_Comm_size(MPI_COMM_WORLD, &activeProcs);
   int chunkSize = ceil(nPart/activeProcs);
   if(myRank == 0){
      double radius;
      double angle;
      for(int i = 0;i<nPart;++i){
         radius = (double)rand() / RAND_MAX;
         while(radius==0){
            radius = rand();
         }
         radius = sqrt(-2.0*log(radius));
         angle = (double)2.0*M_PI*rand() / RAND_MAX;
         x[i] = radius*sin(angle);
         px[i] = radius*cos(angle);
      }

      for(int iProc=1;iProc<activeProcs;++iProc){
         MPI_Send(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGX, MPI_COMM_WORLD);
         MPI_Send(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGPX, MPI_COMM_WORLD);
      }
   } else{
      MPI_Status status;
      MPI_Recv(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD, &status);
      MPI_Recv(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD, &status);
   }
..
```

Each process is started with a different *rank*

Example : The hard way

```
int main(int argc, char *argv[]){
   int TAGREADY = 0;
   int TAGX = 1;
   int TAGPX = 2;
   const int nPart = 500000;
   double x[nPart];
   double px[nPart];
   MPI_Init(&argc, &argv);
   int myRank;
   int activeProcs;
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
   MPI_Comm_size(MPI_COMM_WORLD, &activeProcs);
   int chunkSize = ceil(nPart/activeProcs);
   if(myRank == 0){
      double radius;
      double angle;
      for(int i = 0;i<nPart;++i){
         radius = (double)rand() / RAND_MAX;
         while(radius==0){
            radius = rand();
         }
         radius = sqrt(-2.0*log(radius));
         angle = (double)2.0*M_PI*rand() / RAND_MAX;
         x[i] = radius*sin(angle);
         px[i] = radius*cos(angle);
      }

      for(int iProc=1;iProc<activeProcs;++iProc){
         MPI_Send(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGX, MPI_COMM_WORLD);
         MPI_Send(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGPX, MPI_COMM_WORLD);
      }
   } else{
      MPI_Status status;
      MPI_Recv(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD, &status);
      MPI_Recv(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD, &status);
   }
..
```

Initialisation

Each process is started with a different *rank*

We'll share the load between processes

Example : The hard way

Initialisation

```
int main(int argc, char *argv[]){
   int TAGREADY = 0;
   int TAGX = 1;
   int TAGPX = 2;
   const int nPart = 500000;
   double x[nPart];
   double px[nPart];
   MPI_Init(&argc, &argv);
   int myRank;
   int activeProcs;
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
   MPI_Comm_size(MPI_COMM_WORLD, &activeProcs);
   int chunkSize = ceil(nPart/activeProcs);
   if(myRank == 0){
      double radius;
      double angle;
      for(int i = 0;i<nPart;++i){
         radius = (double)rand() / RAND_MAX;
         while(radius==0){
            radius = rand();
         }
         radius = sqrt(-2.0*log(radius));
         angle = (double)2.0*M_PI*rand() / RAND_MAX;
         x[i] = radius*sin(angle);
         px[i] = radius*cos(angle);
      }

      for(int iProc=1;iProc<activeProcs;++iProc){
         MPI_Send(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGX, MPI_COMM_WORLD);
         MPI_Send(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGPX, MPI_COMM_WORLD);
      }
   } else{
      MPI_Status status;
      MPI_Recv(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD, &status);
      MPI_Recv(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD, &status);
   }
..
```

Each process is started with a different *rank*

We'll share the load between processes

Example : The hard way

```
int main(int argc, char *argv[]){
   int TAGREADY = 0;
   int TAGX = 1;
   int TAGPX = 2;
   const int nPart = 500000;
   double x[nPart];
   double px[nPart];
   MPI_Init(&argc, &argv);
   int myRank;
   int activeProcs;
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
   MPI_Comm_size(MPI_COMM_WORLD, &activeProcs);
   int chunkSize = ceil(nPart/activeProcs);
   if(myRank == 0){
      double radius;
      double angle;
      for(int i = 0;i<nPart;++i){
         radius = (double)rand() / RAND_MAX;
         while(radius==0){
            radius = rand();
         }
         radius = sqrt(-2.0*log(radius));
         angle = (double)2.0*M_PI*rand() / RAND_MAX;
         x[i] = radius*sin(angle);
         px[i] = radius*cos(angle);
      }

      for(int iProc=1;iProc<activeProcs;++iProc){
         MPI_Send(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGX, MPI_COMM_WORLD);
         MPI_Send(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGPX, MPI_COMM_WORLD);
      }
   } else{
      MPI_Status status;
      MPI_Recv(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD, &status);
      MPI_Recv(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD, &status);
   }
..
```

Initialisation

Each process is started with a different *rank*

We'll share the load between processes

Here only the process with rank 0 does the initialization

Example : The hard way

```
int main(int argc, char *argv[]){
   int TAGREADY = 0;
   int TAGX = 1;
   int TAGPX = 2;
   const int nPart = 500000;
   double x[nPart];
   double px[nPart];
   MPI_Init(&argc, &argv);
   int myRank;
   int activeProcs;
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
   MPI_Comm_size(MPI_COMM_WORLD, &activeProcs);
   int chunkSize = ceil(nPart/activeProcs);
   if(myRank == 0){
      double radius;
      double angle;
      for(int i = 0;i<nPart;++i){
         radius = (double)rand() / RAND_MAX;
         while(radius==0){
            radius = rand();
         }
         radius = sqrt(-2.0*log(radius));
         angle = (double)2.0*M_PI*rand() / RAND_MAX;
         x[i] = radius*sin(angle);
         px[i] = radius*cos(angle);
      }

      for(int iProc=1;iProc<activeProcs;++iProc){
         MPI_Send(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGX, MPI_COMM_WORLD);
         MPI_Send(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGPX, MPI_COMM_WORLD);
      }
   } else{
      MPI_Status status;
      MPI_Recv(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD, &status);
      MPI_Recv(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD, &status);
   }
..
```

Initialisation

Each process is started with a different *rank*

We'll share the load between processes

Here only the process with rank 0 does the initialization

Then it sends the corresponding data (chunk of the full set of particles coordinates) to other processes

Example : The hard way

```
int main(int argc, char *argv[]){
   int TAGREADY = 0;
   int TAGX = 1;
   int TAGPX = 2;
   const int nPart = 500000;
   double x[nPart];
   double px[nPart];
   MPI_Init(&argc, &argv);
   int myRank;
   int activeProcs;
   MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
   MPI_Comm_size(MPI_COMM_WORLD, &activeProcs);
   int chunkSize = ceil(nPart/activeProcs);
   if(myRank == 0){
      double radius;
      double angle;
      for(int i = 0;i<nPart;++i){
         radius = (double)rand() / RAND_MAX;
         while(radius==0){
            radius = rand();
         }
         radius = sqrt(-2.0*log(radius));
         angle = (double)2.0*M_PI*rand() / RAND_MAX;
         x[i] = radius*sin(angle);
         px[i] = radius*cos(angle);
      }

      for(int iProc=1;iProc<activeProcs;++iProc){
         MPI_Send(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGX, MPI_COMM_WORLD);
         MPI_Send(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE, iProc,TAGPX, MPI_COMM_WORLD);
      }
   } else{
      MPI_Status status;
      MPI_Recv(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD, &status);
      MPI_Recv(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD, &status);
   }
..
```

Initialisation

Each process is started with a different *rank*

We'll share the load between processes

Here only the process with rank 0 does the initialization

Then it sends the corresponding data (chunk of the full set of particles coordinates) to other processes

In the mean time, the other processes wait to receive the data

Example : The hard way

Computing

```
...
    int nTurn = 10000;
    double sinPhix = sin(2.0*M_PI*0.31);
    double cosPhix = cos(2.0*M_PI*0.31);
    double scale = 1.0;
    double thres = 1E-10;

    double oldX = 0.0;
    double oldPx = 0.0;
    for(int turn = 0;turn<nTurn;++turn){
        for(int i = 0;i<chunkSize;++i){
            oldX = x[i];
            oldPx = px[i];
            x[i] = cosPhix*oldX + sinPhix*oldPx;
            px[i] = -sinPhix*oldX + cosPhix*oldPx;
            if(abs(x[i]) > thres){
                px[i] += scale*(1.0-exp(-0.5*x[i]*x[i]))/x[i];
            }
        }
    }
...
```

The computing part hasn't changed much: only the loop is done only on a subset of particles

Example: The hard way

Output

```
...
 if(myRank==0){
    int nRecv = 1;
    while(nRecv<activeProcs){
       MPI_Status status;
       MPI_Recv(NULL, 0, MPI_BYTE, MPI_ANY_SOURCE,TAGREADY,
                     MPI_COMM_WORLD,&status);
       int iProc = status.MPI_SOURCE;
       MPI_Recv(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGX, MPI_COMM_WORLD,&status);
       MPI_Recv(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGPX, MPI_COMM_WORLD,&status);
       ++nRecv;
    }
 } else{
    MPI_Send(NULL, 0, MPI_BYTE, 0,TAGREADY, MPI_COMM_WORLD);
    MPI_Send(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD);
    MPI_Send(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD);
 }
 MPI_Finalize();
...
```

Example: The hard way

Output

```
...
 if(myRank==0){
    int nRecv = 1;
    while(nRecv<activeProcs){
       MPI_Status status;
       MPI_Recv(NULL, 0, MPI_BYTE, MPI_ANY_SOURCE,TAGREADY,
                     MPI_COMM_WORLD,&status);
       int iProc = status.MPI_SOURCE;
       MPI_Recv(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGX, MPI_COMM_WORLD,&status);
       MPI_Recv(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGPX, MPI_COMM_WORLD,&status);
       ++nRecv;
    }
 } else{
    MPI_Send(NULL, 0, MPI_BYTE, 0,TAGREADY, MPI_COMM_WORLD);
    MPI_Send(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD);
    MPI_Send(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD);
 }
 MPI_Finalize();
...
```

The process with rank 0 collects back the data from other processes

Example: The hard way

Output

```
...
 if(myRank==0){
    int nRecv = 1;
    while(nRecv<activeProcs){
       MPI_Status status;
       MPI_Recv(NULL, 0, MPI_BYTE, MPI_ANY_SOURCE,TAGREADY,
                     MPI_COMM_WORLD,&status);
       int iProc = status.MPI_SOURCE;
       MPI_Recv(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGX, MPI_COMM_WORLD,&status);
       MPI_Recv(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGPX, MPI_COMM_WORLD,&status);
       ++nRecv;
    }
 } else{
    MPI_Send(NULL, 0, MPI_BYTE, 0,TAGREADY, MPI_COMM_WORLD);
    MPI_Send(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD);
    MPI_Send(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD);
 }
 MPI_Finalize();
...
```

The process with rank 0 collects back the data from other processes

When finished, the other processes send their data to the process with rank 0

Example: The hard way

Output

```
...
 if(myRank==0){
    int nRecv = 1;
    while(nRecv<activeProcs){
       MPI_Status status;
       MPI_Recv(NULL, 0, MPI_BYTE, MPI_ANY_SOURCE,TAGREADY,
                     MPI_COMM_WORLD,&status);
       int iProc = status.MPI_SOURCE;
       MPI_Recv(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGX, MPI_COMM_WORLD,&status);
       MPI_Recv(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGPX, MPI_COMM_WORLD,&status);
       ++nRecv;
    }
 } else{
    MPI_Send(NULL, 0, MPI_BYTE, 0,TAGREADY, MPI_COMM_WORLD);
    MPI_Send(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD);
    MPI_Send(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD);
 }
 MPI_Finalize();
...
```

The process with rank 0 collects back the data from other processes

When finished, the other processes send their data to the process with rank 0

Example: The hard way

Output

```
...
 if(myRank==0){
    int nRecv = 1;
    while(nRecv<activeProcs){
       MPI_Status status;
       MPI_Recv(NULL, 0, MPI_BYTE, MPI_ANY_SOURCE,TAGREADY,
                     MPI_COMM_WORLD,&status);
       int iProc = status.MPI_SOURCE;
       MPI_Recv(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGX, MPI_COMM_WORLD,&status);
       MPI_Recv(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGPX, MPI_COMM_WORLD,&status);
       ++nRecv;
    }
 } else{
    MPI_Send(NULL, 0, MPI_BYTE, 0,TAGREADY, MPI_COMM_WORLD);
    MPI_Send(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD);
    MPI_Send(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD);
 }
 MPI_Finalize();
...
```

The process with rank 0 collects back the data from other processes

When finished, the other processes send their data to the process with rank 0

Example: The hard way

Output

```
...
 if(myRank==0){
    int nRecv = 1;
    while(nRecv<activeProcs){
       MPI_Status status;
       MPI_Recv(NULL, 0, MPI_BYTE, MPI_ANY_SOURCE,TAGREADY,
                     MPI_COMM_WORLD,&status);
       int iProc = status.MPI_SOURCE;
       MPI_Recv(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGX, MPI_COMM_WORLD,&status);
       MPI_Recv(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGPX, MPI_COMM_WORLD,&status);
       ++nRecv;
    }
 } else{
    MPI_Send(NULL, 0, MPI_BYTE, 0,TAGREADY,MPI_COMM_WORLD);
    MPI_Send(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD);
    MPI_Send(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD);
 }
 MPI_Finalize();
...
```

The process with rank 0 collects back the data from other processes

When finished, the other processes send their data to the process with rank 0

Example: The hard way

Output

```
...
 if(myRank==0){
    int nRecv = 1;
    while(nRecv<activeProcs){
       MPI_Status status;
       MPI_Recv(NULL, 0, MPI_BYTE, MPI_ANY_SOURCE,TAGREADY,
                     MPI_COMM_WORLD,&status);
       int iProc = status.MPI_SOURCE;
       MPI_Recv(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGX, MPI_COMM_WORLD,&status);
       MPI_Recv(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGPX, MPI_COMM_WORLD,&status);
       ++nRecv;
    }
 } else{
    MPI_Send(NULL, 0, MPI_BYTE, 0,TAGREADY,MPI_COMM_WORLD);
    MPI_Send(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD);
    MPI_Send(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD);
 }
 MPI_Finalize();
...
```

The process with rank 0 collects back the data from other processes

When finished, the other processes send their data to the process with rank 0

- It is your responsibility to make sure that the data send finds a receiver

  → Avoid deadlocks !

Example: The hard way

Output

```
...
 if(myRank==0){
    int nRecv = 1;
    while(nRecv<activeProcs){
       MPI_Status status;
       MPI_Recv(NULL, 0, MPI_BYTE, MPI_ANY_SOURCE,TAGREADY,
                     MPI_COMM_WORLD,&status);
       int iProc = status.MPI_SOURCE;
       MPI_Recv(&x[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGX, MPI_COMM_WORLD,&status);
       MPI_Recv(&px[iProc*chunkSize], chunkSize, MPI_DOUBLE,
                     status.MPI_SOURCE,TAGPX, MPI_COMM_WORLD,&status);
       ++nRecv;
    }
 } else{
    MPI_Send(NULL, 0, MPI_BYTE, 0,TAGREADY, MPI_COMM_WORLD);
    MPI_Send(x,chunkSize, MPI_DOUBLE, 0,TAGX, MPI_COMM_WORLD);
    MPI_Send(px,chunkSize, MPI_DOUBLE, 0,TAGPX, MPI_COMM_WORLD);
 }
 MPI_Finalize();
...
```

- It is your responsibility to make sure that the data send finds a receiver

  → Avoid deadlocks !

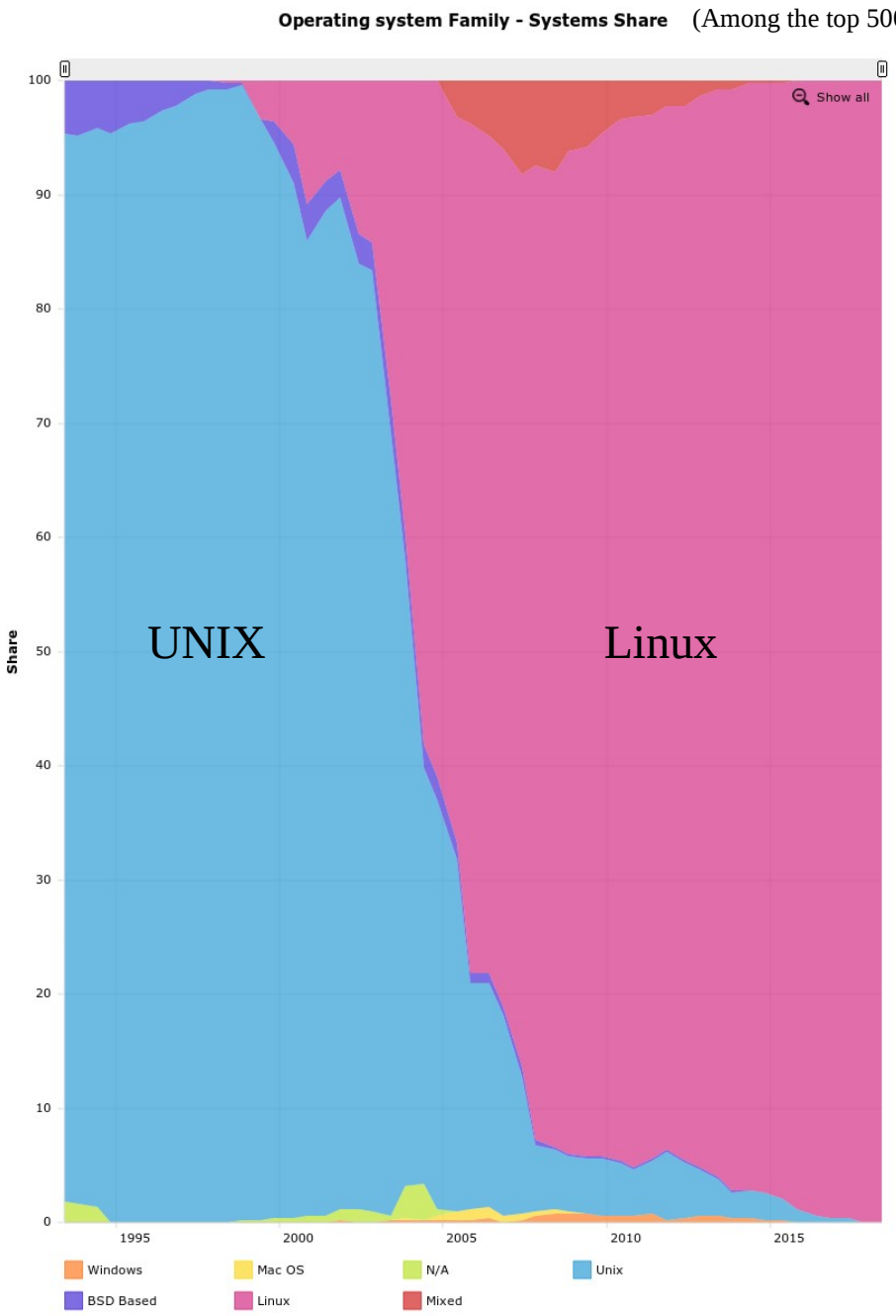
- The data type as well as the tag (user defined int) avoids miss-deliveries...

Example: The easy way

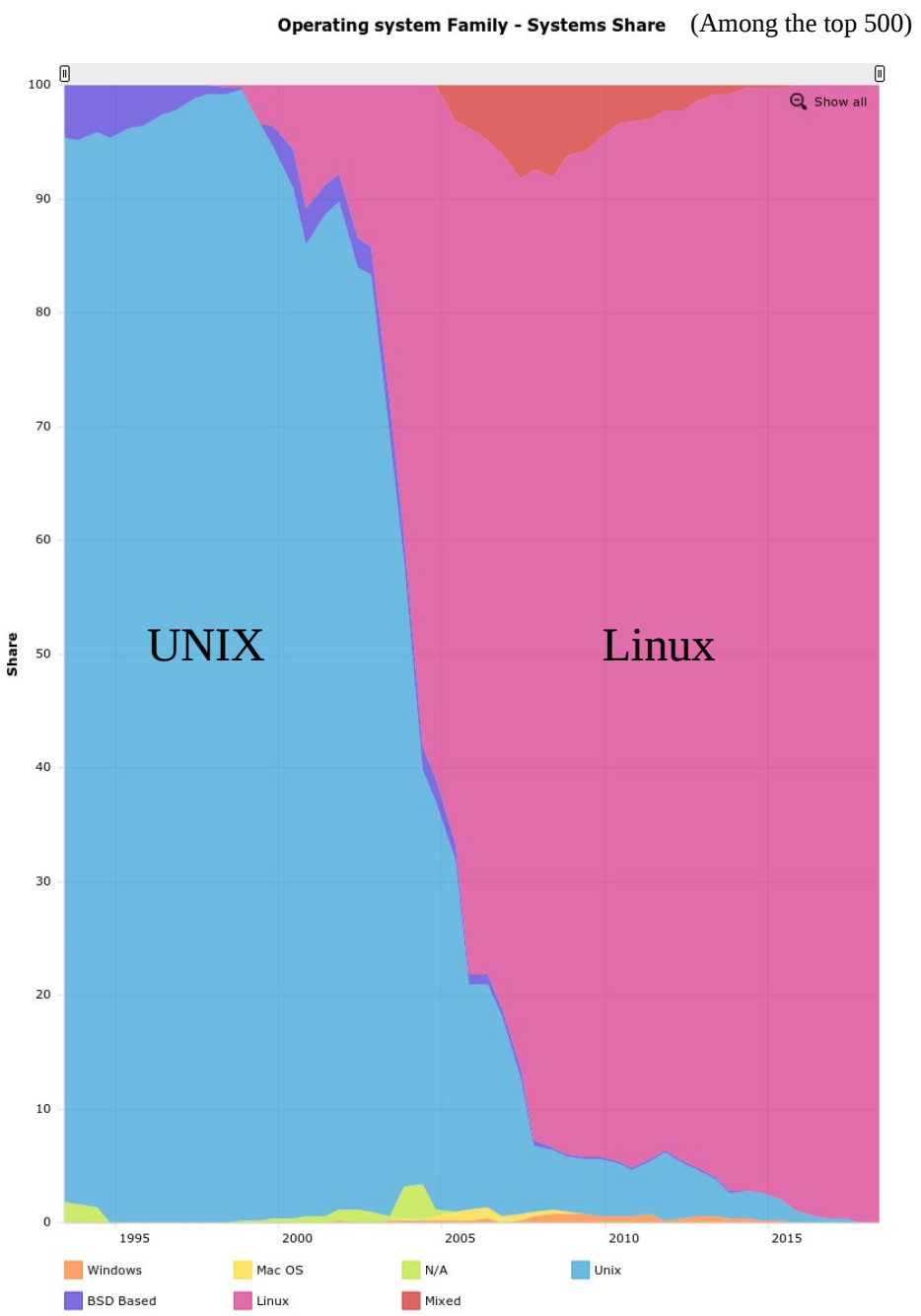
Not that the road is difficult, is that what is difficult is the way.

- Søren Kierkegaard

Operating system



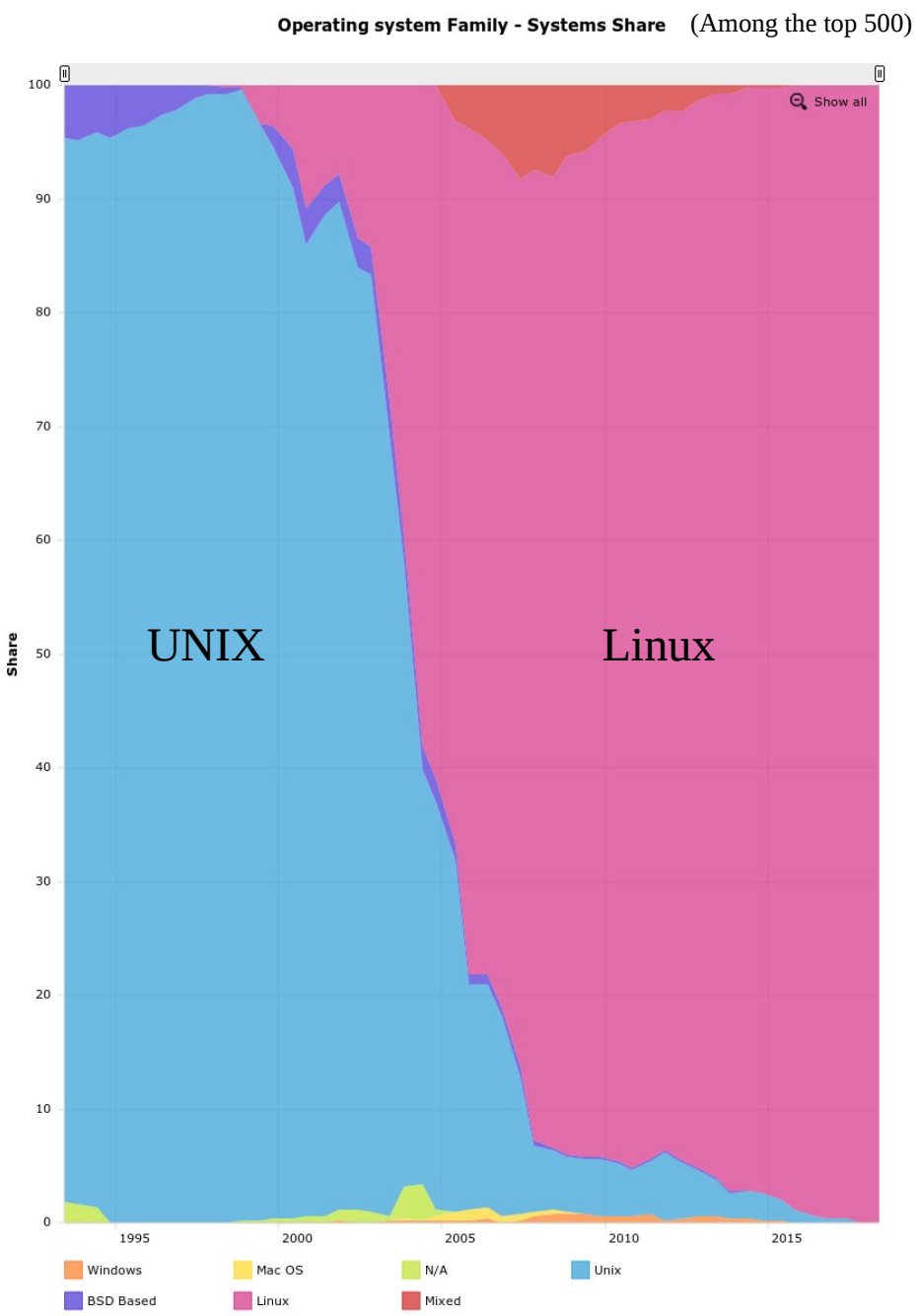
Operating system



In a world without walls and fences, who needs windows and gates ?

- Dino Esposito

Operating system



In a world without walls and fences, who needs windows and gates ?
- Dino Esposito

If you want to harness this power, you'll probably need to go out of your comfort zone
- Fuego, Chief advisory rabbit

Batch processing

- Computer clusters are usually shared resources
- A scheduler assign jobs to resources, base on job descriptions given by the user
  - In order to efficiently allocate time for the various jobs, the scheduler usually asks for quite some details about your jobs → Not for Voodoo programmers!

```
15:46:50 xbuffat slurmgate04 /hpcscratch/user/xbuffat/CAS2018_MPIexample
→ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             33359 batch-lon StartMPP  tpolzin  R       3:54      8 hpc[061-062,064-069]
             33339 batch-lon     bash dzambell  R    1:41:30      2 hpc[026-027]
             33341 batch-sho       gs delkhech  R    1:36:10      1 hpc006
             33340 batch-sho       gs delkhech  R    1:38:58      1 hpc005
             33345 batch-sho       gs delkhech  R    1:08:28      1 hpc009
             33344 batch-sho       gs delkhech  R    1:13:33      1 hpc008
             33343 batch-sho       gs delkhech  R    1:14:17      1 hpc007
             33348 batch-lon  dima.sh delkhech  R      56:09      1 hpc028
             26544 batch-lon   triMag  thibaut  R 41-07:04:16     10 hpc[075,078-086]
             33360   be-long phaseSpa  xbuffat  R       0:19     20 hpc-be[004,018,027,029,046,
058,061-062,065,071,073,081-082,089-090,093,097-098,132,137]
             32530   be-long HCMOctop  thibaut  R 7-02:14:14      5 hpc-be[003,024,028,033,036]
             32529   be-long HCMOctop  thibaut  R 7-02:15:02      5 hpc-be[057,083,091-092,105]
             32379   be-long HCMOctop  thibaut  R 8-04:48:40      5 hpc-be[005,009,015,034,044]
             32947 batch-lon     Simu  thibaut  R 4-04:37:31      2 hpc[185-186]
             32982 batch-lon   1_3Ghz  thibaut  R 4-01:11:19     10 hpc[051-060]
             32981 batch-lon   1_3Ghz  thibaut  R 4-01:13:31     10 hpc[016-025]
             32969 batch-lon     Simu  thibaut  R 4-02:34:41      2 hpc[014-015]
```

Batch processing

- Computer clusters are usually shared resources
- A scheduler assign jobs to resources, base on job descriptions given by the user
  - In order to efficiently allocate time for the various jobs, the scheduler usually asks for quite some details about your jobs → Not for Voodoo programmers!

```
15:46:50 xbuffat slurmgate04 /hpcscratch/user/xbuffat/CAS2018_MPIexample
→ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             33359 batch-lon StartMPP  tpolzin  R       3:54      8 hpc[061-062,064-069]
             33339 batch-lon     bash dzambell  R    1:41:30      2 hpc[026-027]
             33341 batch-sho       gs delkhech  R    1:36:10      1 hpc006
             33340 batch-sho       gs delkhech  R    1:38:58      1 hpc005
             33345 batch-sho       gs delkhech  R    1:08:28      1 hpc009
             33344 batch-sho       gs delkhech  R    1:13:33      1 hpc008
             33343 batch-sho       gs delkhech  R    1:14:17      1 hpc007
             33348 batch-lon  dima.sh delkhech  R      56:09      1 hpc028
             26544 batch-lon   triMag  thibaut  R 41-07:04:16     10 hpc[075,078-086]
             33360   be-long phaseSpa  xbuffat  R       0:19     20 hpc-be[004,018,027,029,046,
058,061-062,065,071,073,081-082,089-090,093,097-098,132,137]
             32530   be-long HCMOctop  thibaut  R 7-02:14:14      5 hpc-be[003,024,028,033,036]
             32529   be-long HCMOctop  thibaut  R 7-02:15:02      5 hpc-be[057,083,091-092,105]
             32379   be-long HCMOctop  thibaut  R 8-04:48:40      5 hpc-be[005,009,015,034,044]
             32947 batch-lon     Simu  thibaut  R 4-04:37:31      2 hpc[185-186]
             32982 batch-lon   1_3Ghz  thibaut  R 4-01:11:19     10 hpc[051-060]
             32981 batch-lon   1_3Ghz  thibaut  R 4-01:13:31     10 hpc[016-025]
             32969 batch-lon     Simu  thibaut  R 4-02:34:41      2 hpc[014-015]
```

Example with Slurm :

```
#!/bin/bash
#SBATCH --nodes 20
#SBATCH --tasks-per-node 40
#SBATCH --cpus-per-task 1
#SBATCH --time 00:10:00
#SBATCH --mail-user xbuffat@cern.ch

module load mpi/mvapich2/2.3rc2

srun executable
```

Batch processing

- Computer clusters are usually shared resources
- A scheduler assign jobs to resources, base on job descriptions given by the user
  - In order to efficiently allocate time for the various jobs, the scheduler usually asks for quite some details about your jobs → Not for Voodoo programmers!

```
15:46:50 xbuffat slurmgate04 /hpcscratch/user/xbuffat/CAS2018_MPIexample
→ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             33359 batch-lon StartMPP  tpolzin  R       3:54      8 hpc[061-062,064-069]
             33339 batch-lon     bash dzambell  R    1:41:30      2 hpc[026-027]
             33341 batch-sho       gs delkhech  R    1:36:10      1 hpc006
             33340 batch-sho       gs delkhech  R    1:38:58      1 hpc005
             33345 batch-sho       gs delkhech  R    1:08:28      1 hpc009
             33344 batch-sho       gs delkhech  R    1:13:33      1 hpc008
             33343 batch-sho       gs delkhech  R    1:14:17      1 hpc007
             33348 batch-lon  dima.sh delkhech  R      56:09      1 hpc028
             26544 batch-lon   triMag  thibaut  R 41-07:04:16     10 hpc[075,078-086]
             33360   be-long phaseSpa  xbuffat  R       0:19     20 hpc-be[004,018,027,029,046,
058,061-062,065,071,073,081-082,089-090,093,097-098,132,137]
             32530   be-long HCMOctop  thibaut  R 7-02:14:14      5 hpc-be[003,024,028,033,036]
             32529   be-long HCMOctop  thibaut  R 7-02:15:02      5 hpc-be[057,083,091-092,105]
             32379   be-long HCMOctop  thibaut  R 8-04:48:40      5 hpc-be[005,009,015,034,044]
             32947 batch-lon     Simu  thibaut  R 4-04:37:31      2 hpc[185-186]
             32982 batch-lon   1_3Ghz  thibaut  R 4-01:11:19     10 hpc[051-060]
             32981 batch-lon   1_3Ghz  thibaut  R 4-01:13:31     10 hpc[016-025]
             32969 batch-lon     Simu  thibaut  R 4-02:34:41      2 hpc[014-015]
```

Example with Slurm :

```
#!/bin/bash
#SBATCH --nodes 20
#SBATCH --tasks-per-node 40
#SBATCH --cpus-per-task 1
#SBATCH --time 00:10:00
#SBATCH --mail-user xbuffat@cern.ch

module load mpi/mvapich2/2.3rc2

srun executable
```

→ Total of 800 processes without multithreading

Batch processing

- Computer clusters are usually shared resources
- A scheduler assign jobs to resources, base on job descriptions given by the user
  - In order to efficiently allocate time for the various jobs, the scheduler usually asks for quite some details about your jobs → Not for Voodoo programmers!

```
15:46:50 xbuffat slurmgate04 /hpcscratch/user/xbuffat/CAS2018_MPIexample
→ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             33359 batch-lon StartMPP  tpolzin  R       3:54      8 hpc[061-062,064-069]
             33339 batch-lon     bash dzambell  R    1:41:30      2 hpc[026-027]
             33341 batch-sho       gs delkhech  R    1:36:10      1 hpc006
             33340 batch-sho       gs delkhech  R    1:38:58      1 hpc005
             33345 batch-sho       gs delkhech  R    1:08:28      1 hpc009
             33344 batch-sho       gs delkhech  R    1:13:33      1 hpc008
             33343 batch-sho       gs delkhech  R    1:14:17      1 hpc007
             33348 batch-lon  dima.sh delkhech  R      56:09      1 hpc028
             26544 batch-lon   triMag  thibaut  R 41-07:04:16     10 hpc[075,078-086]
             33360   be-long phaseSpa  xbuffat  R       0:19     20 hpc-be[004,018,027,029,046,
058,061-062,065,071,073,081-082,089-090,093,097-098,132,137]
             32530   be-long HCMOctop  thibaut  R 7-02:14:14      5 hpc-be[003,024,028,033,036]
             32529   be-long HCMOctop  thibaut  R 7-02:15:02      5 hpc-be[057,083,091-092,105]
             32379   be-long HCMOctop  thibaut  R 8-04:48:40      5 hpc-be[005,009,015,034,044]
             32947 batch-lon     Simu  thibaut  R 4-04:37:31      2 hpc[185-186]
             32982 batch-lon   1_3Ghz  thibaut  R 4-01:11:19     10 hpc[051-060]
             32981 batch-lon   1_3Ghz  thibaut  R 4-01:13:31     10 hpc[016-025]
             32969 batch-lon     Simu  thibaut  R 4-02:34:41      2 hpc[014-015]
```

Example with Slurm :

```
#!/bin/bash
#SBATCH --nodes 20
#SBATCH --tasks-per-node 40
#SBATCH --cpus-per-task 1
#SBATCH --time 00:10:00
#SBATCH --mail-user xbuffat@cern.ch

module load mpi/mvapich2/2.3rc2

srun executable
```

→ Total of 800 processes without multithreading

→ Maximum execution time

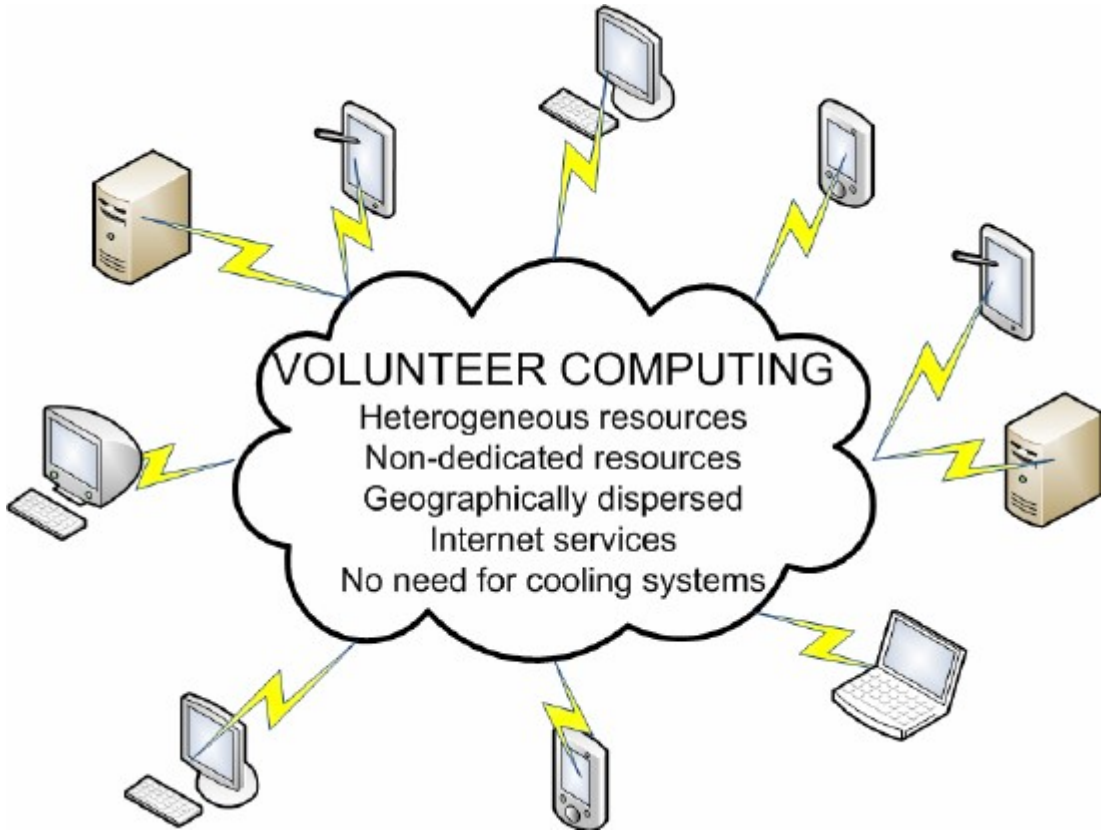
Applications

- MPI is used for every application that requires more than CPUs or GPUs than a single node can provide
  - Affordable shared memory system (multithreading) are limited to tens to hundreds of CPUs
  - Motherboards typically come with few GPU slots (fashionable for bitcoin mining)
- MPI can be using in combination with shared memory parallelism (e.g. OpenMP) and multiple GPUs
- Large memory transfer between node can become a limitation

  → Data locality is again a key for efficient parallelisation
- In accelerators :
  - Plasma simulations (particle sources)
  - Collective beam instabilities (space charge, electron clouds, beam-beam interactions, wake fields)
  - Safety (fire propagation simulation)
  - …

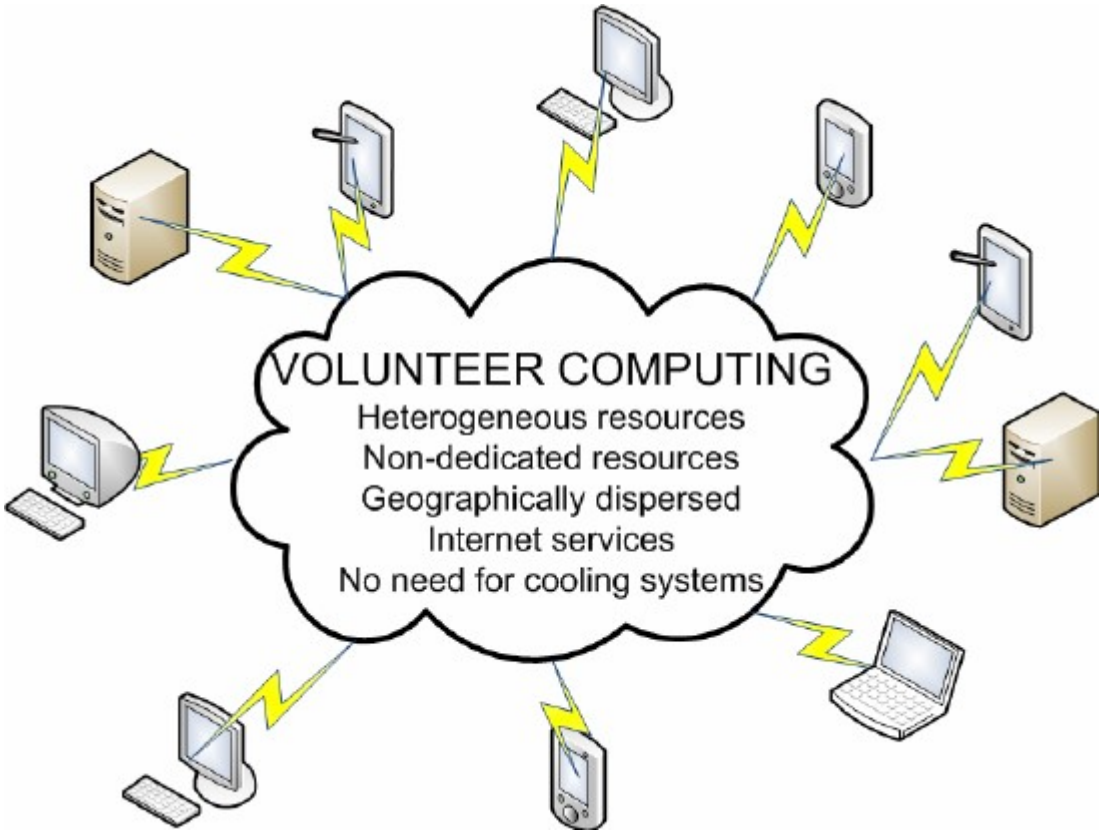
Volunteer computing

Illustration : G. Cabrera, et al., Journal of Applied Operations Research, Jan. 2013



Volunteer computing

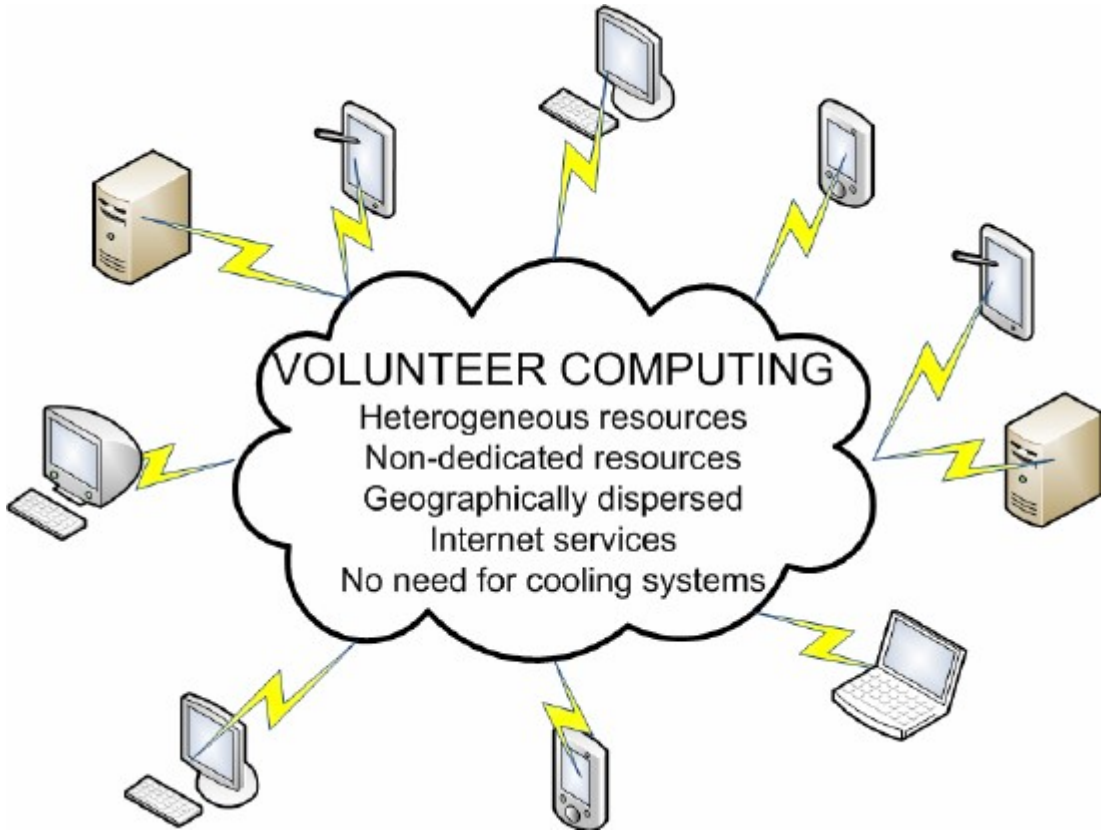
Illustration : G. Cabrera, et al., Journal of Applied Operations Research, Jan. 2013



- Significant overhead w.r.t. dedicated machines (portability, compatibility, tails, ...)

Volunteer computing

Illustration : G. Cabrera, et al., Journal of Applied Operations Research, Jan. 2013

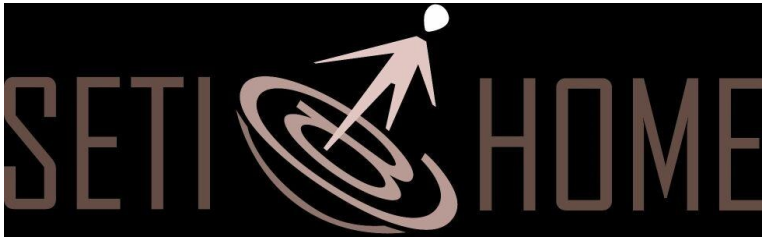


- Significant overhead w.r.t. dedicated machines (portability, compatibility, tails, ...)
- The connection through the Internet is too slow for MPI

  → A system (commonly BOINC) distributes fully independent jobs to idling computers and collects output files

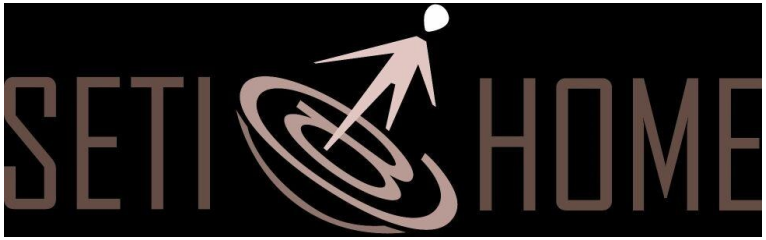
  → Appropriate only for fully independent tasks

Volunteer computing



- The largest volunteer computing project, SETI@home reached hundreds of Tflop/s
- This performance used to exceed those of computer clusters, now it is ~3 orders of magnitude less than the fastest cluster

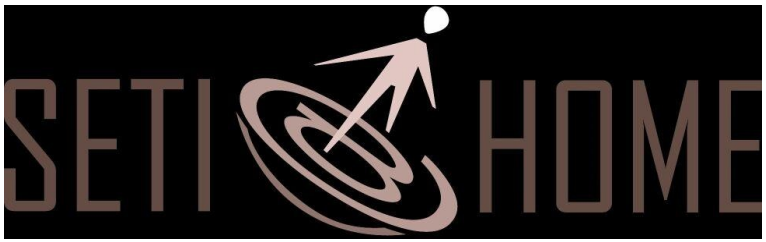
Volunteer computing



- The largest volunteer computing project, SETI@home reached hundreds of Tflop/s
- This performance used to exceed those of computer clusters, now it is ~3 orders of magnitude less than the fastest cluster

- LHC@home contributed significantly to the understanding of non-linear dynamics in the LHC and HL-LHC
- This project is still running, volunteer! http://lhcathome.web.cern.ch/

Volunteer computing

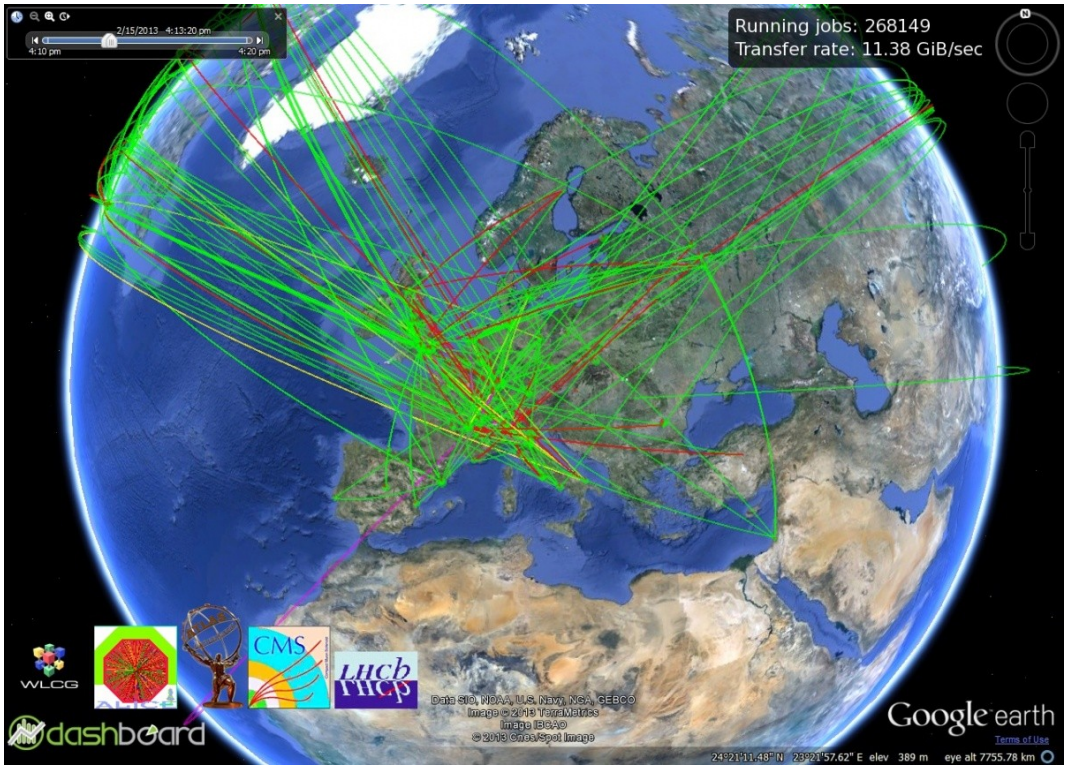


- The largest volunteer computing project, SETI@home reached hundreds of Tflop/s
- This performance used to exceed those of computer clusters, now it is ~3 orders of magnitude less than the fastest cluster

- LHC@home contributed significantly to the understanding of non-linear dynamics in the LHC and HL-LHC
- This project is still running, volunteer! http://lhcathome.web.cern.ch/

→ Nowadays the development, operational and maintenance costs of a volunteer computing project often exceeds the price of a computer clusters

Grid computing



- When a single cluster is not enough (computing, storage or even budget wise) !

  →The load is shared among distributed (but dedicated) systems using slow connections through the Internet

- The jobs sent through to the different systems are independent
  - Within each cluster, the job can be accelerated using multiple CPUs and GPUs

Summary

- Computing techniques evolve as the technology advances
  - Nowadays the goes towards several computing units working in parallel

  → Vectorisation, multithreading, hyperthreading, GPU acceleration
- Programing languages also adapt to these technologies, with various levels of complexity and efficiency

  → In C: Compiler auto-vectorisation, OpenMP, OpenACC, CUDA

  → Several task can be accelerated by a novice, just use the tools made available to you (compiler options, pragmas, accelerated libraries)
- Most of our problems can be solved with a single computer, but some applications rely on larger infrastructures
  - Computer clusters
  - Computing grids
- The choice of technology needed depends
  - On the computing requirements
  - On the memory requirements
  - Communications requirement (data locality!)

… just avoid that :