High Performance Computing

Term 4 2018/2019

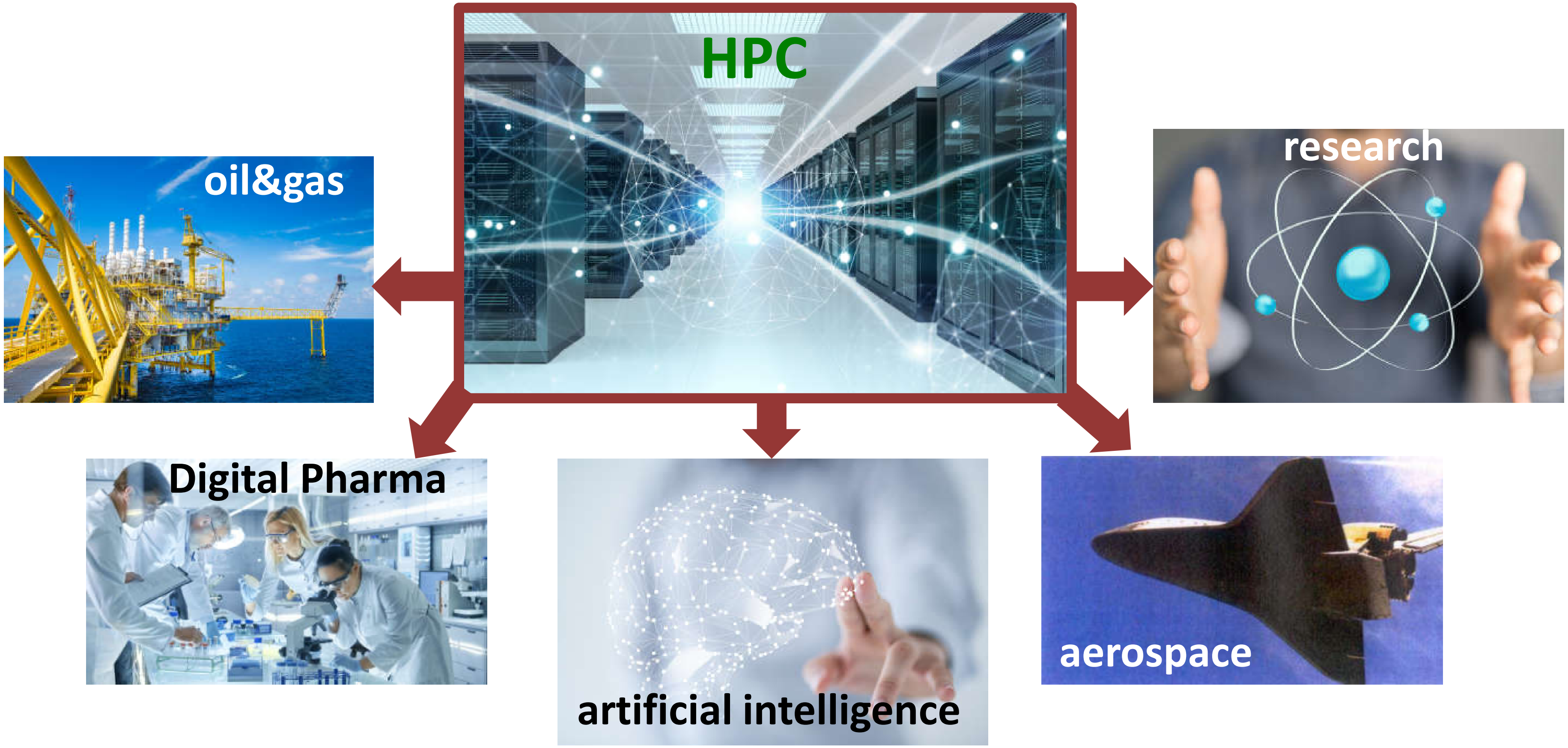
Lecture 2

Outline

- Applications of HPC and supercomputers
- Memory hierarchy
  - Matrix multiplication
- Building intuition: parallel processes
- Amdahl‘s law
- Flynn‘s taxonomy
- Shared memory and distributed memory systems
- Processes and threads
- Start with OpenMP

High Performance Computing is a strategic necessity

HPC provides competitive advantage for industrial and business companies and research groups.



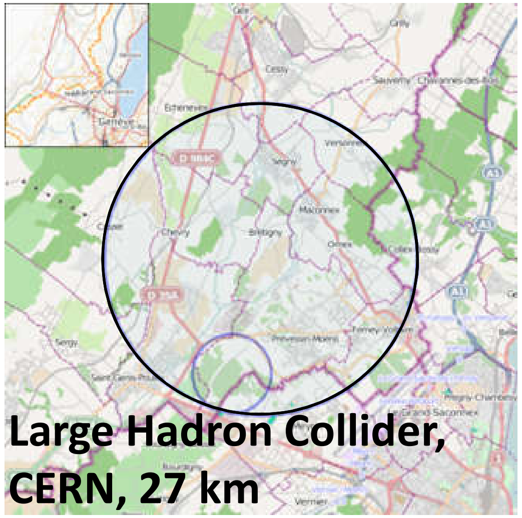
High Performance Computing is a strategic necessity

Developing just one medical drug takes ~10 years and 2.5 billion dollars[1]

Digital Pharma, for example. structure based drug discovery saves companies hundreds of millions of dollars!

[1] https://www.scientificamerican.com/article/cost-to-develop-new-pharmaceutical-drug-now-exceeds-2-

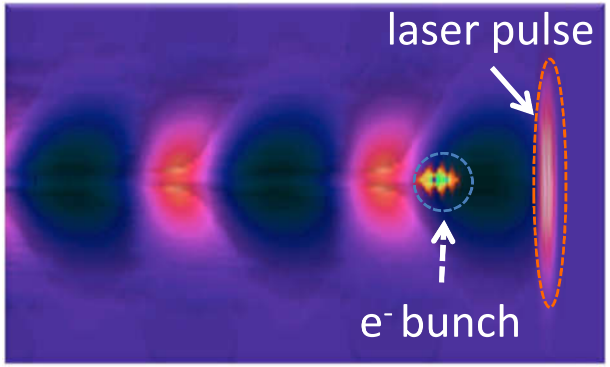
Laser plasma technology can revolutionize accelerators

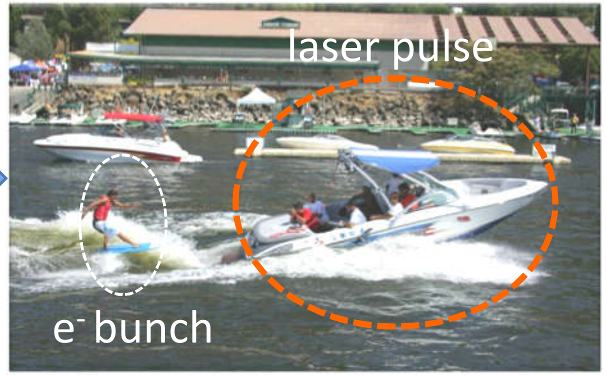


- **Conventional technology:**
  smaller details = bigger machines
- **Large Hadron Collider:** 27 km, candidate for Higgs boson. Limit of accelerating gradients. More energy = increase in size. Cost? Size?
- **Relativistic Laser Plasma:** **1 GeV in just 10 cm** using powerful lasers interacting with plasma

Laser plasma technology can revolutionize accelerators

Laser Plasma Acceleration = Surfing on a plasma wave





simulations by S. Kuschel

Most powerful industrial supercomputer is used for oil&gas exploration(Eni S.p.A.)

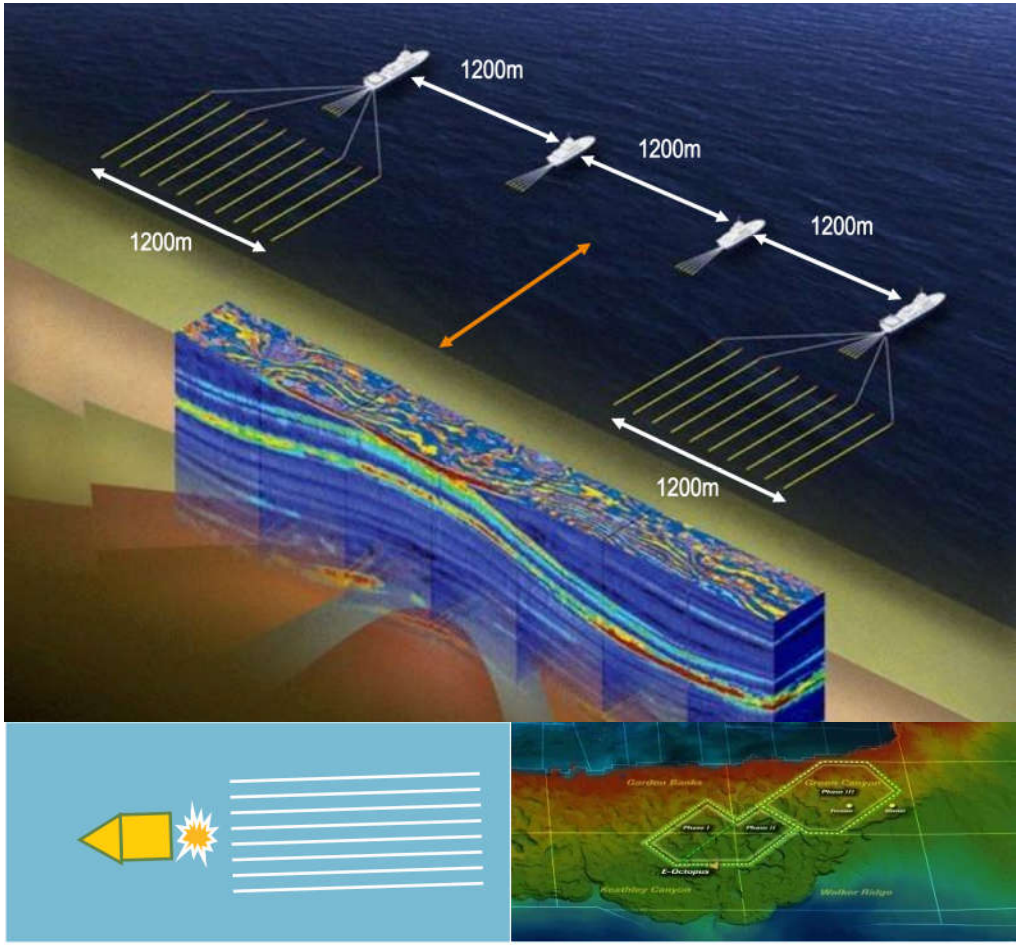
Supercomputer HPC4 installed at Eni is #15 in top500 (**12 ПФлопс**). For comparison, most powerful russian supercomputer: **Lomonosov-2, 2.5 PFlops**.

15	Eni S.p.A. Italy	**HPC4** - Proliant DL380 Gen10, Xeon Platinum 8160 24C 2.1GHz, Mellanox InfiniBand EDR, NVIDIA Tesla P100 HPE	253,600	12,210.0	18,621.1	1,320

Much shorter oil&gas exploration by using supercomputers

Seismic –Acoustic measurement
Electromagnetic
Gravity

Much shorter oil&gas exploration by using supercomputers



20 – 25,000 sensors
500 MB – 2 GB

50 – 200,000 shots
50 – 200 TB data

Data from year 2011

Creation of 3D seismic model:

~30000000 CPU core hours
(~1000 years on a normal laptop)

Without the supercomputer:

- *it is impossible to create a precise seismic model*
- *it is not clear where to drill for optimal oil output and the mistake can cost millions of dollars*

New oil&gas deposits are more and more hard to reach.

Climate modelling

$$\frac{du}{dt} - \left(f + \frac{u}{a}\operatorname{tg}\varphi\right)v + \frac{1}{a\cos\varphi}\left(\frac{\partial\Phi}{\partial\lambda} + \frac{RT}{p_s}\frac{\partial p_s}{\partial\lambda}\right) = F_u,$$

$$\frac{dv}{dt} + \left(f + \frac{u}{a}\operatorname{tg}\varphi\right)u + \frac{1}{a}\left(\frac{\partial\Phi}{\partial\varphi} + \frac{RT}{p_s}\frac{\partial p_s}{\partial\varphi}\right) = F_v,$$

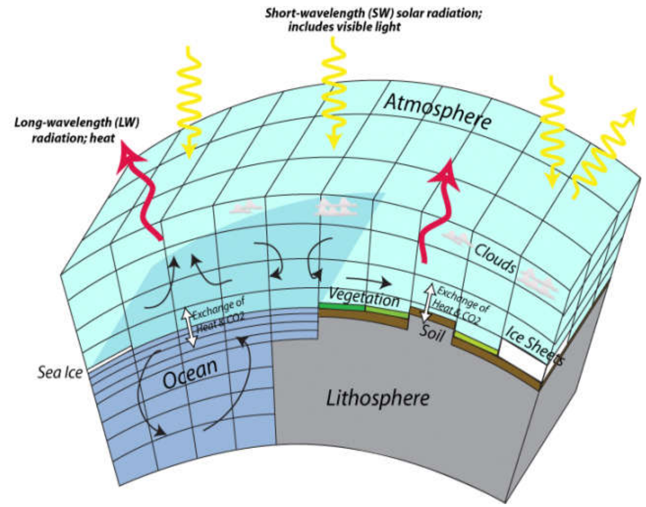
$$\frac{\partial\Phi}{\partial\sigma} = -\frac{RT}{\sigma},$$

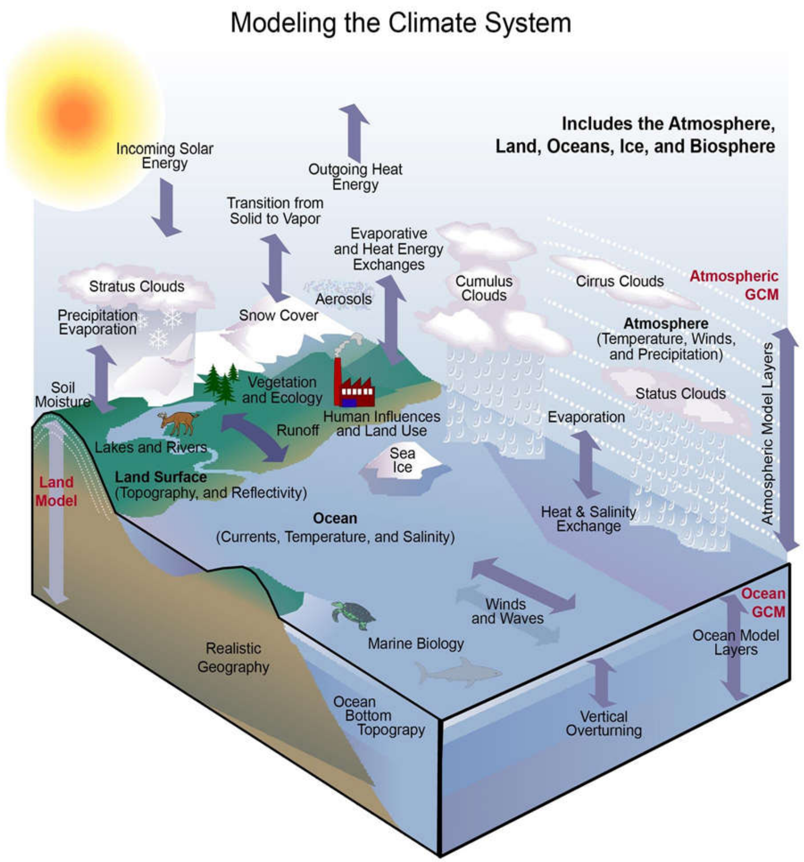
$$\frac{\partial p_s}{\partial t} + \frac{1}{a\cos\varphi}\left(\frac{\partial p_s u}{\partial\lambda} + \frac{p_s v\cos\varphi}{\partial\varphi}\right) + \frac{\partial p_s\dot\sigma}{\partial\sigma} = 0,$$

$$\frac{dT}{dt} - \frac{RT}{c_p\sigma p_s}\left[p_s\dot\sigma + \sigma\left(\frac{\partial p_s}{\partial t} + \frac{u}{a\cos\varphi}\frac{\partial p_s}{\partial\lambda} + \frac{v}{a}\frac{\partial p_s}{\partial\varphi}\right)\right] = F_T + \epsilon,$$

$$\frac{dq}{dt} = F_q - (C - E),$$

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \frac{u}{a\cos\varphi}\frac{\partial}{\partial\lambda} + \frac{v}{a}\frac{\partial}{\partial\varphi} + \dot\sigma\frac{\partial}{\partial\sigma},$$





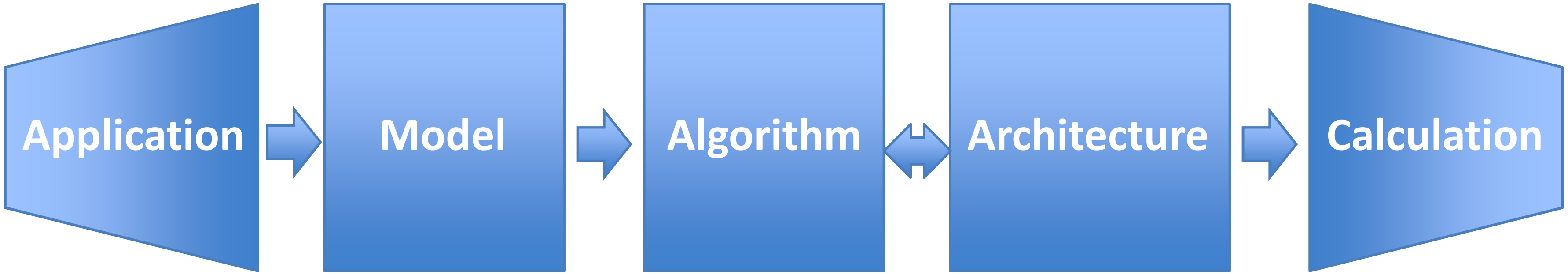
Weather modelling only:

- 100 years, 10 minutes interval ~ 5.3 * 10^6 checkpoints
- 1° latitude and longitude grid of 40 layers ~ 2.6 * 10^6 cells
- Each cell ~ 10 components to model
- Computational cost for one measurement ~ 1000 operations

Total: 10^{17} arithmetic operations, i. e. 3-30 hours on 1 TFlops

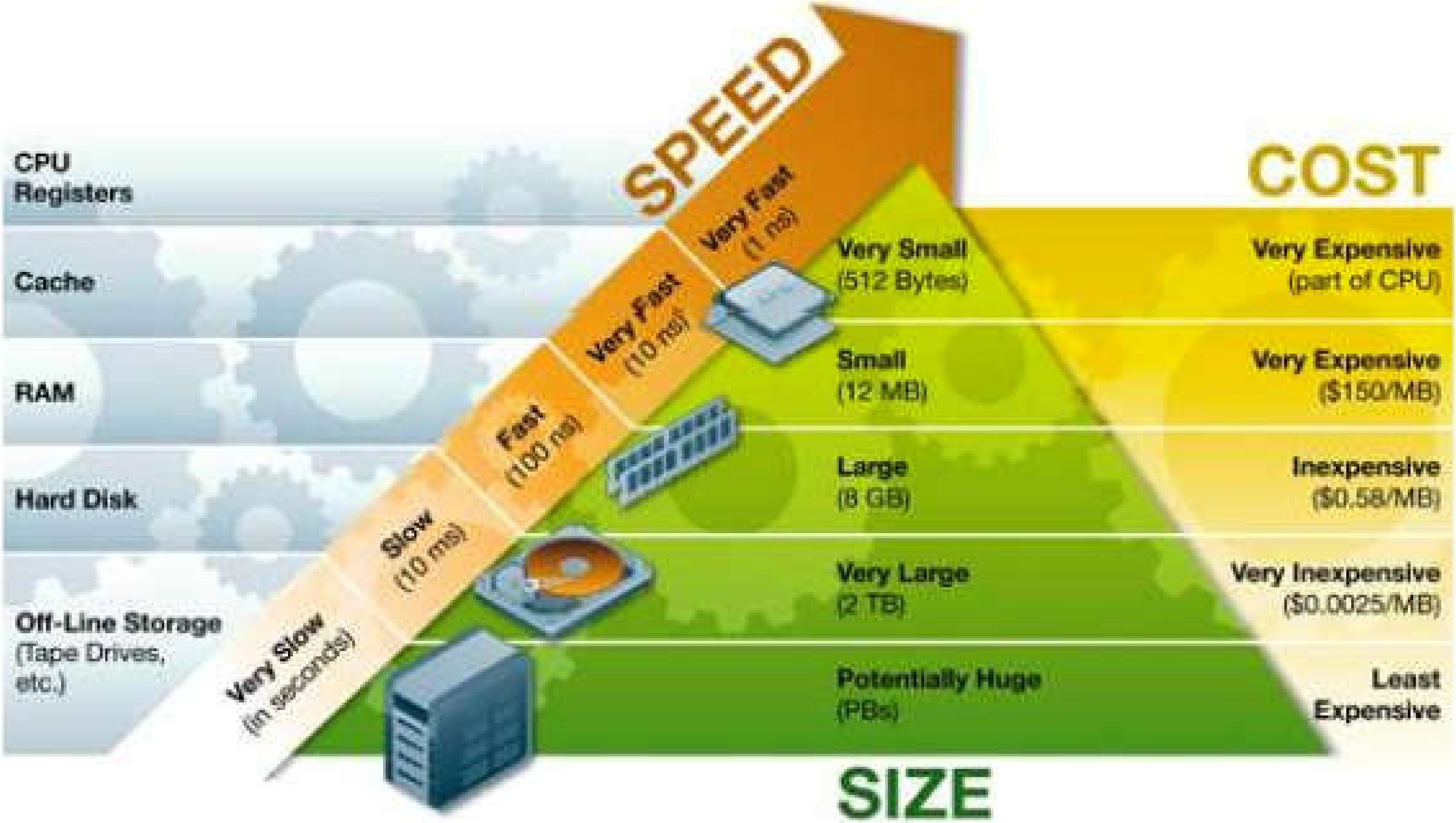
High Performance Computing is not only “computing”: sometimes thinking outside the box and knowledge of other areas helps

Multi-disciplinary approach to HPC is necessary:



- Analytical solution to the problem can save computing time
- Clever model (reduced model) can save computing time
- Clever numerical methods fit to the certain architecture can save computing time

Extended Memory Hierarchy



Source: http://www.ts.avnet.com/uk/products_and_solutions/storage/hierarchy.html

Naive matrix multiplication

$$\begin{matrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{matrix} = \begin{matrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{matrix} \times \begin{matrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{matrix}$$

NxN matrices: A, B, C

N >> 1

C = A * B

$$c_{ij}=\sum_{n=0}^{n=N-1} a_{in} * b_{nj}$$

```
for i from 0 to N-1:
    for j from 0 to N-1:
        for n from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

goo.gl/vQeUD8

1. **Write matmul in C**
2. **Exchange the order of for loops (i,j,n)->(n,i,j)**
3. **Measure execution time and if it changes try to explain why.**
4. *Try different compiler optimization options*
5. *Calculate the flops of a single CPU core of your machine*
6. *Imagine you have K workers – how would you make this code parallel?*
7. *What if you use double * instead of double **?*
8. *Implement Hilbert curve for 2D matrix allocation.*

Naive matrix multiplication

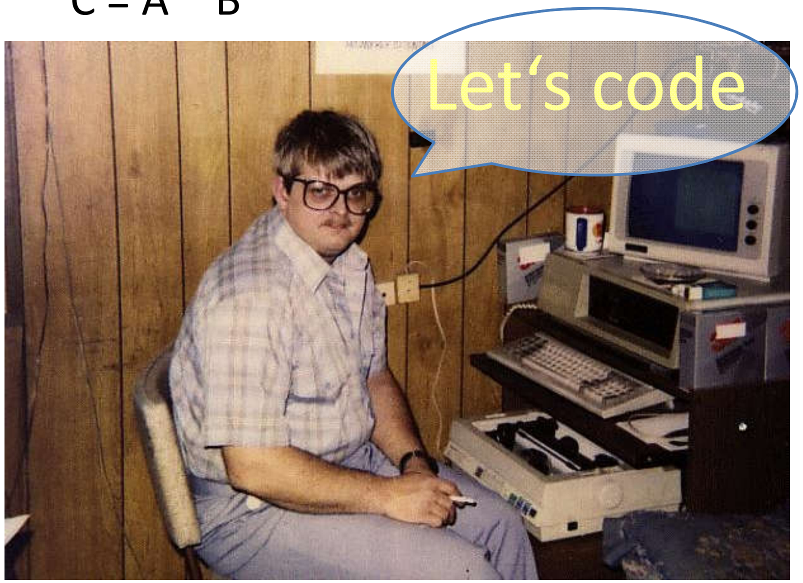
$$\begin{matrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{matrix} = \begin{matrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{matrix} \times \begin{matrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{matrix}$$

NxN matrices: A, B, C

N >> 1

$$c_{ij} = \sum_{n=0}^{n=N-1} a_{in} * b_{nj}$$

C = A * B




```
for i from 0 to N-1:
    for j from 0 to N-1:
        for n from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

goo.gl/vQeUD8

1. Write matmul in C

2. Exchange the order of for loops (i,j,n)->(n,i,j)

3. Measure execution time and if it changes try to explain why.

4. Try different compiler optimization options

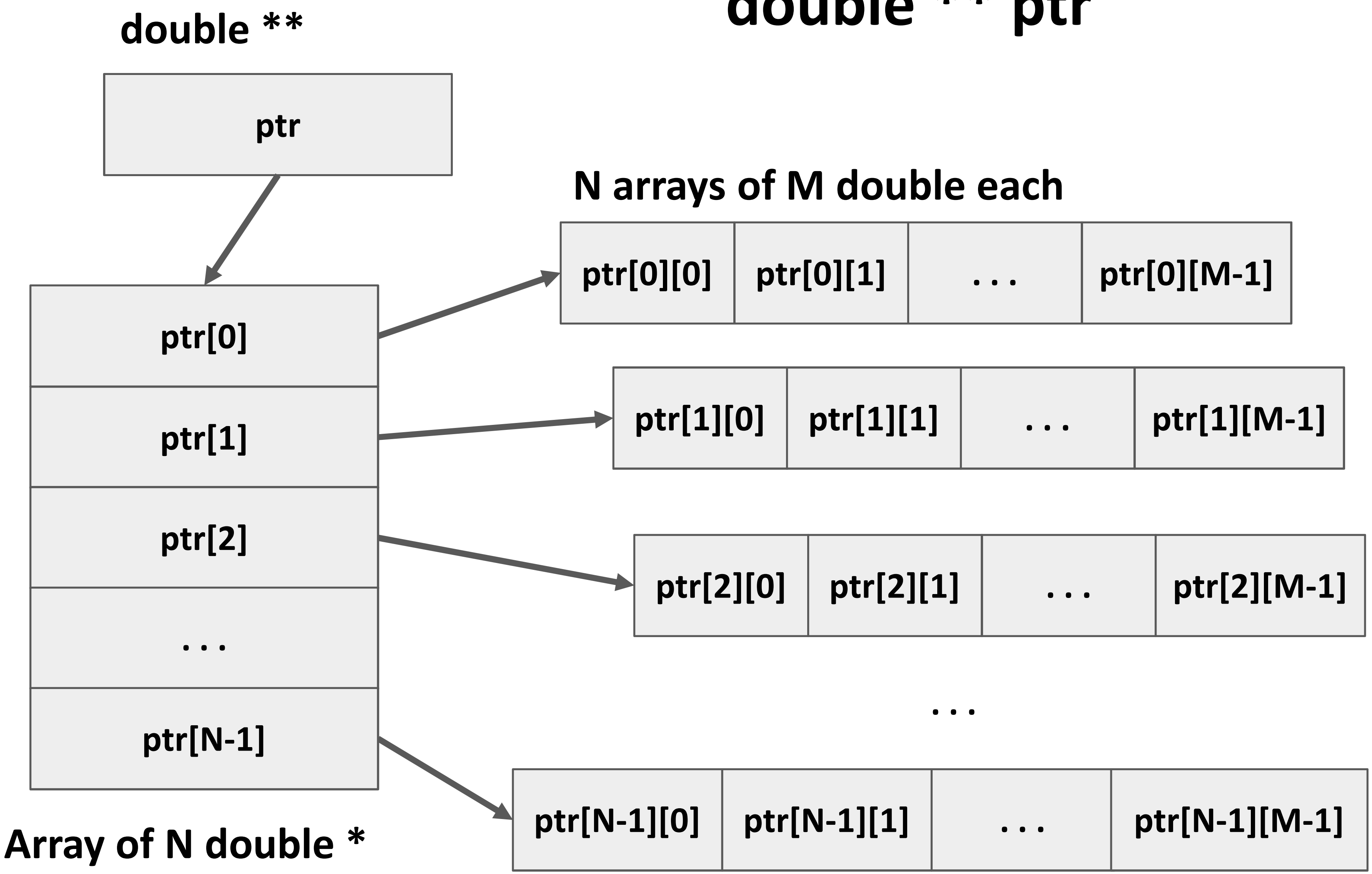
5. Calculate the flops of a single CPU core of your machine

6. Imagine you have K workers – how would you make this code parallel?

*7. What if you use double * instead of double **?*

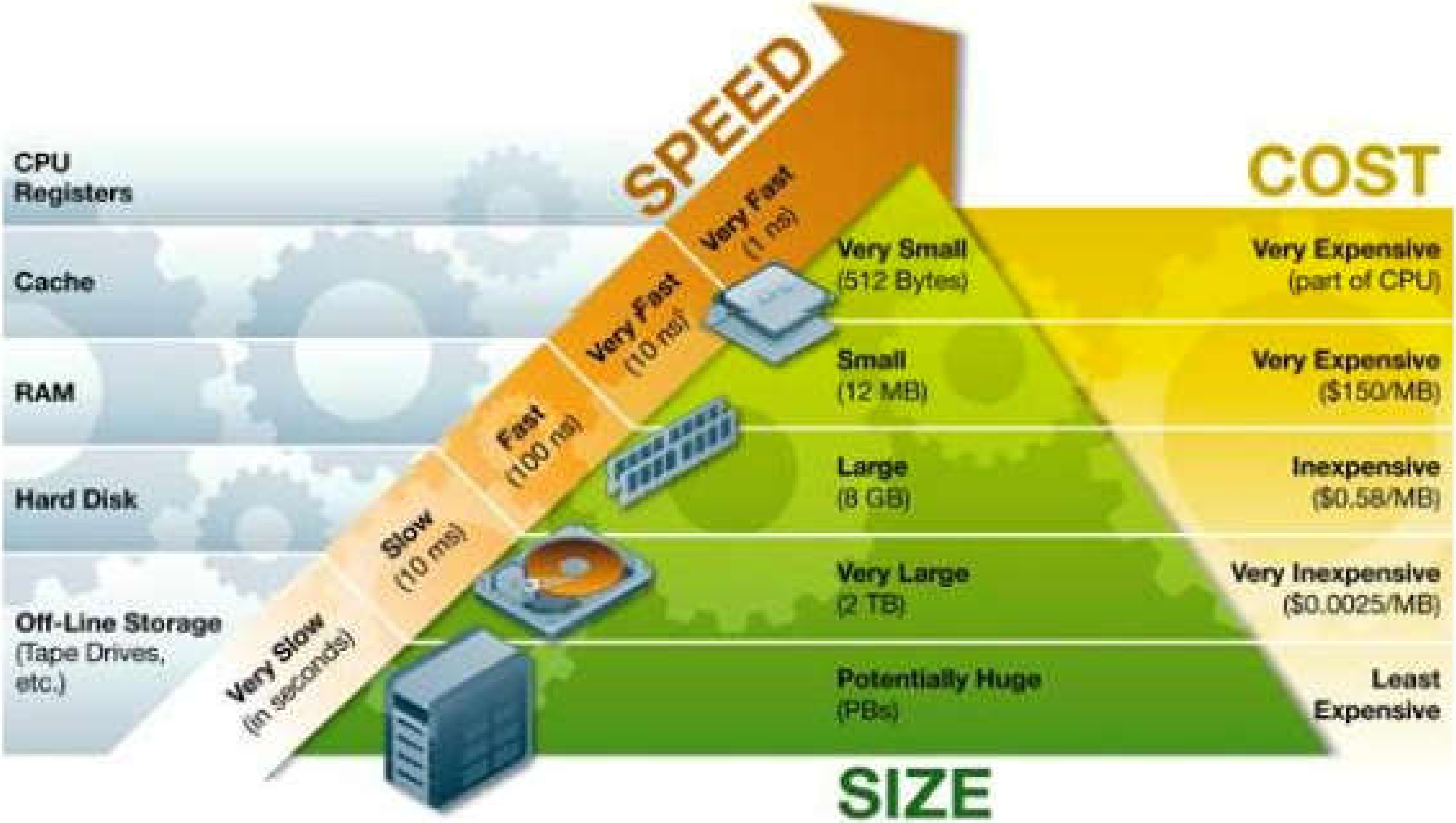
8. Implement Hilbert curve for 2D matrix allocation.

double ** ptr



double **
ptr
N arrays of M double each
ptr[0][0]
ptr[0][1]
. . .
ptr[0][M-1]
ptr[0]
ptr[1][0]
ptr[1][1]
. . .
ptr[1][M-1]
ptr[1]
ptr[2]
ptr[2][0]
ptr[2][1]
. . .
ptr[2][M-1]
. . .
. . .
ptr[N-1]
ptr[N-1][0]
ptr[N-1][1]
. . .
ptr[N-1][M-1]
Array of N double *

Extended Memory Hierarchy

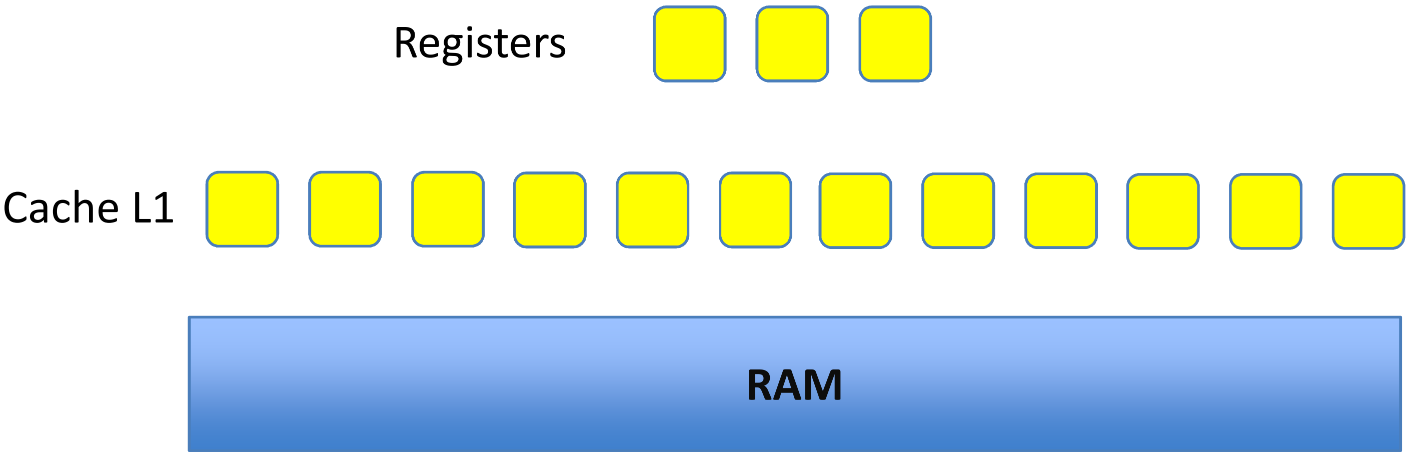


Source: http://www.ts.avnet.com/uk/products_and_solutions/storage/hierarchy.html

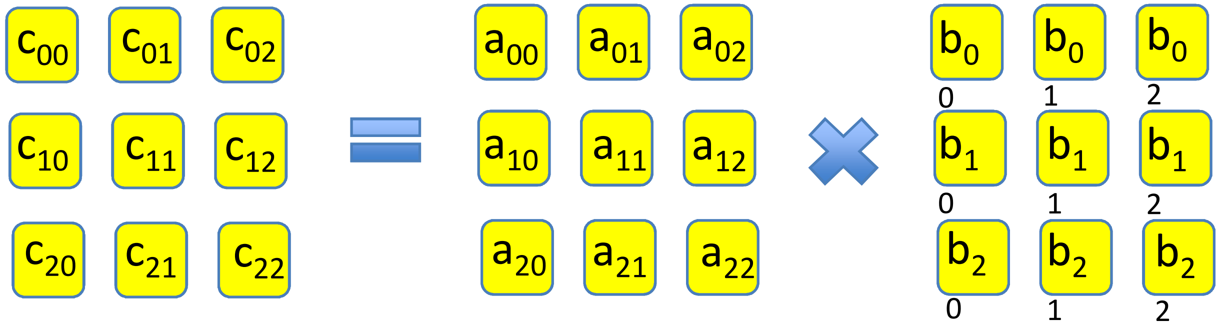
Naive matrix multiplication

$$\begin{matrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{matrix} = \begin{matrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{matrix} \times \begin{matrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{matrix}$$

```
for i from 0 to N-1:
    for j from 0 to N-1:
        for n from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

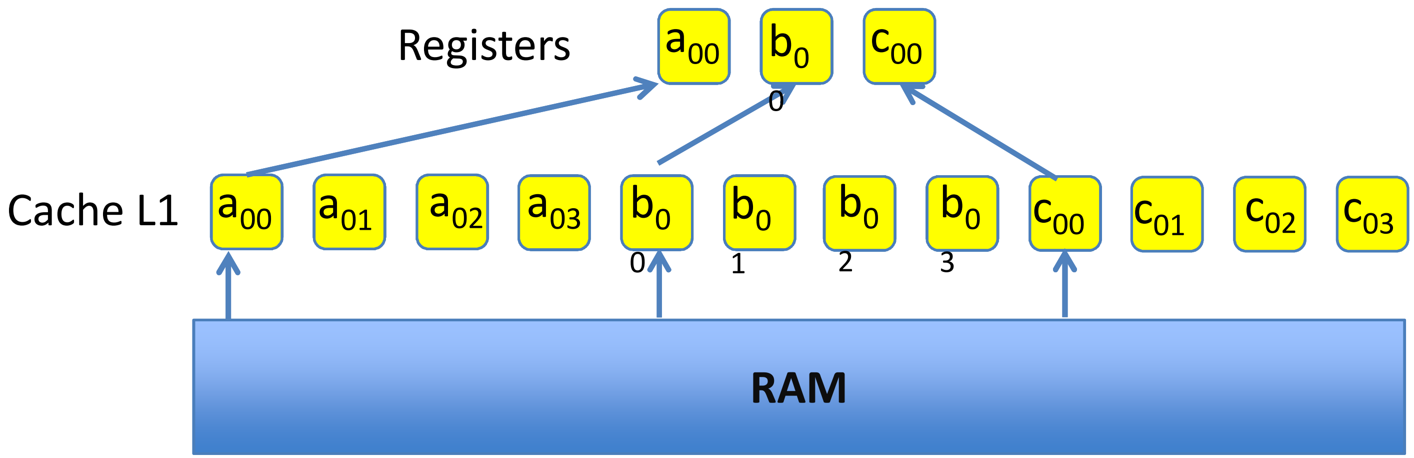


Naive matrix multiplication

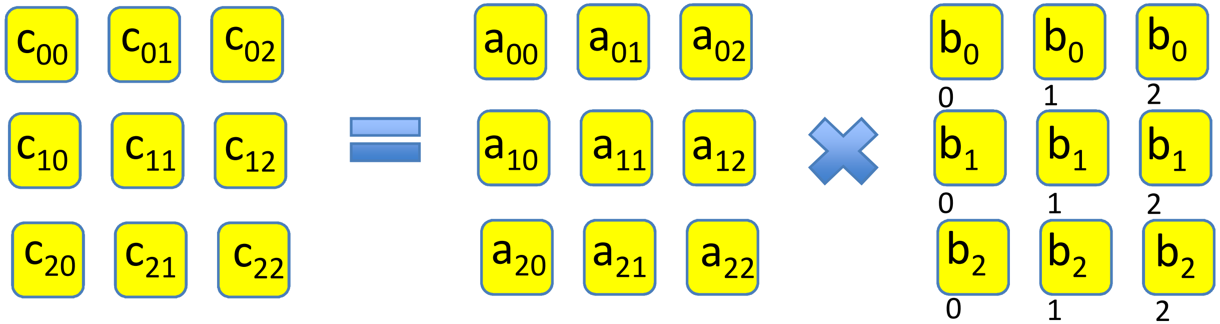



```
for i from 0 to N-1:
    for j from 0 to N-1:
        for n from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

```
i=0
j=0
n=0
C[0][0]+=A[0][0]*B[0][0]
```

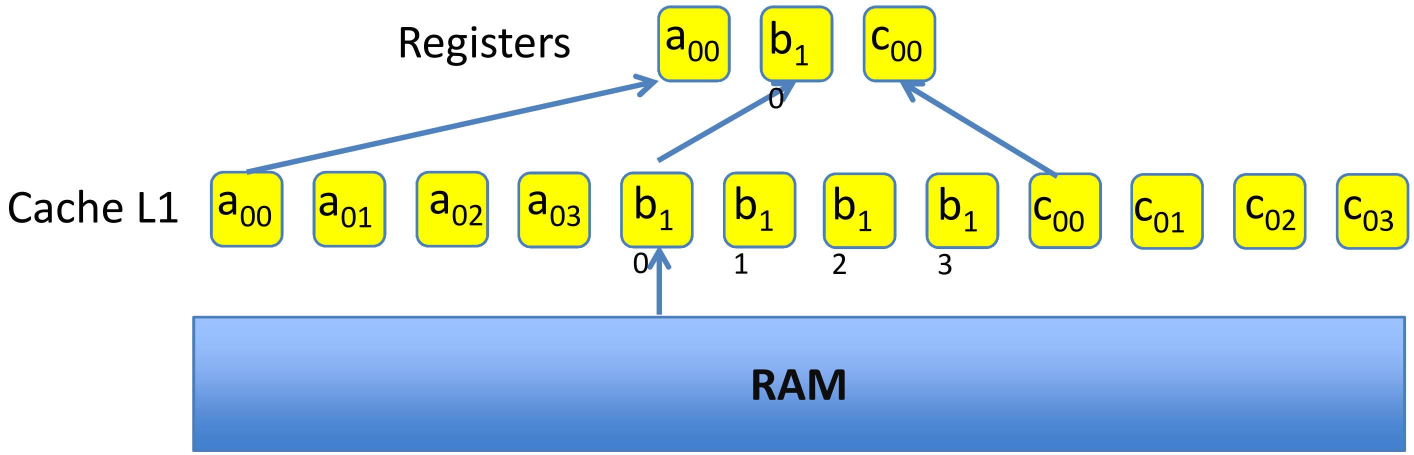


Naive matrix multiplication

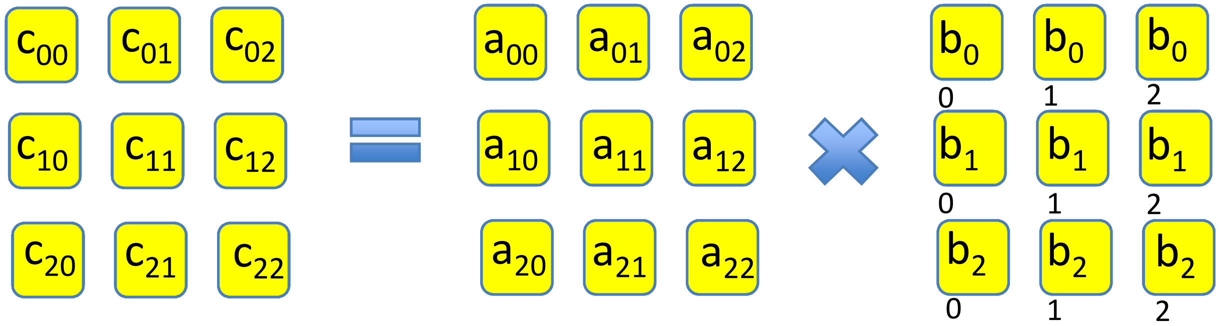



```
for i from 0 to N-1:
    for j from 0 to N-1:
        for n from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

```
i=0
j=0
n=1
C[0][0]+=A[0][1]*B[1][0]
```

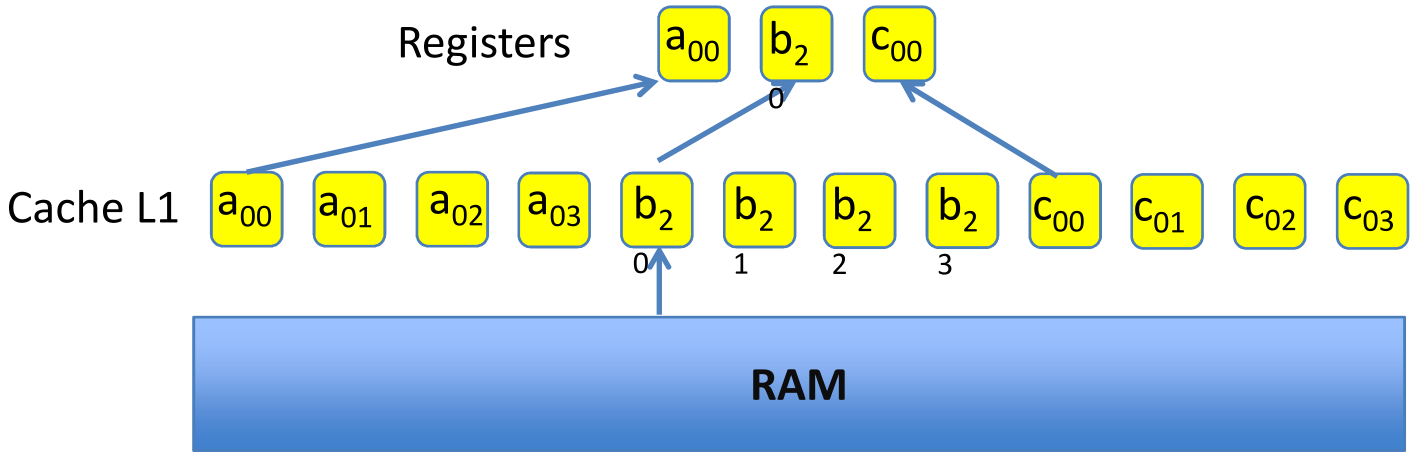


Naive matrix multiplication

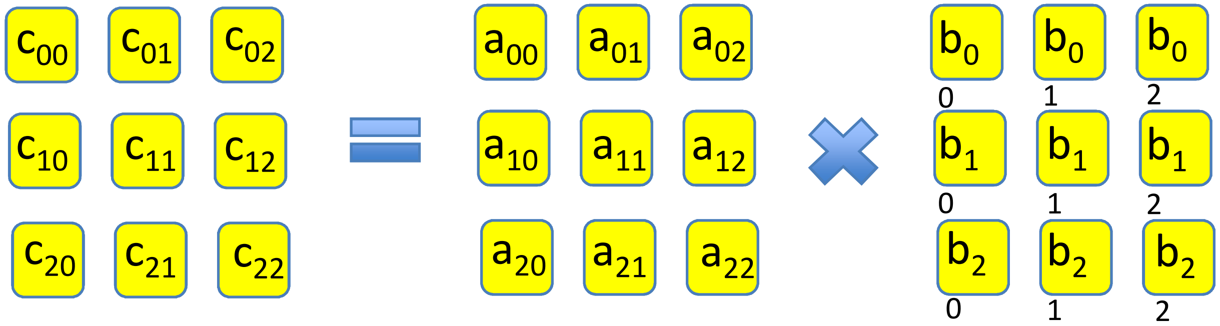



```
for i from 0 to N-1:
    for j from 0 to N-1:
        for n from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

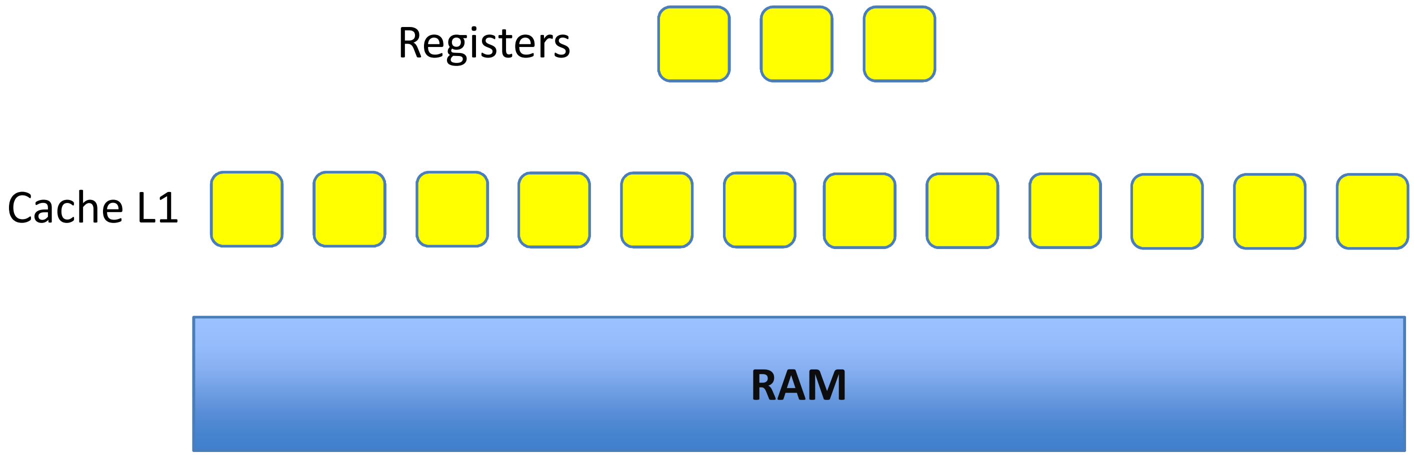
i=0
j=0
n=2
C[0][0]+=A[0][2]*B[2][0]



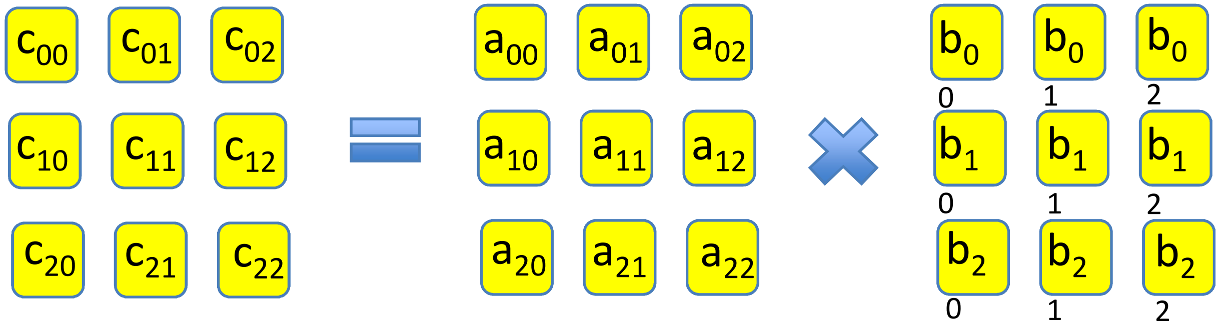
Naive matrix multiplication




```
for n from 0 to N-1:
    for i from 0 to N-1:
        for j from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

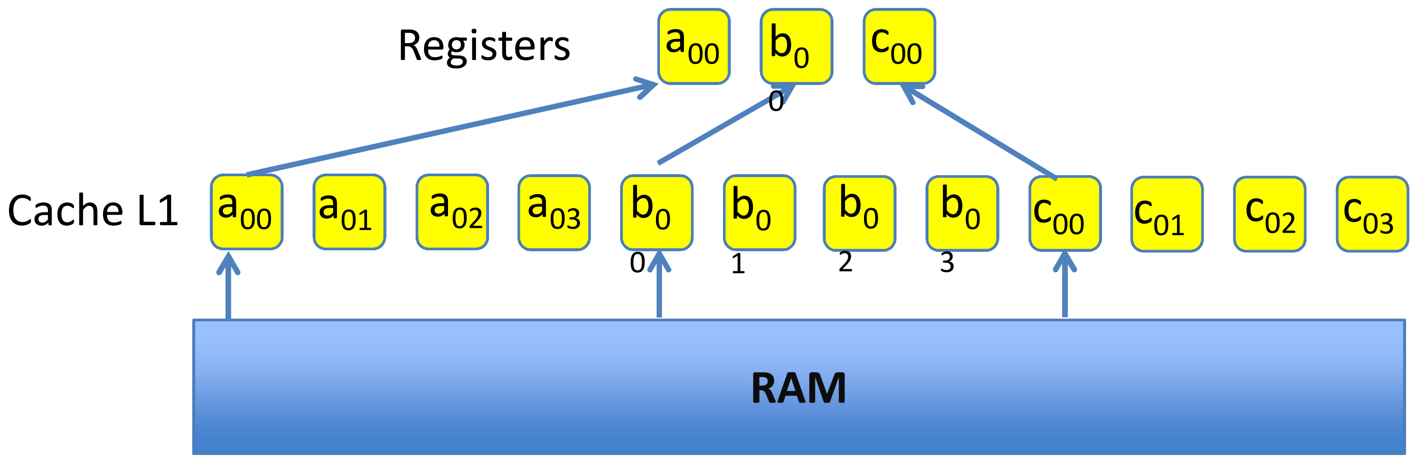


Naive matrix multiplication

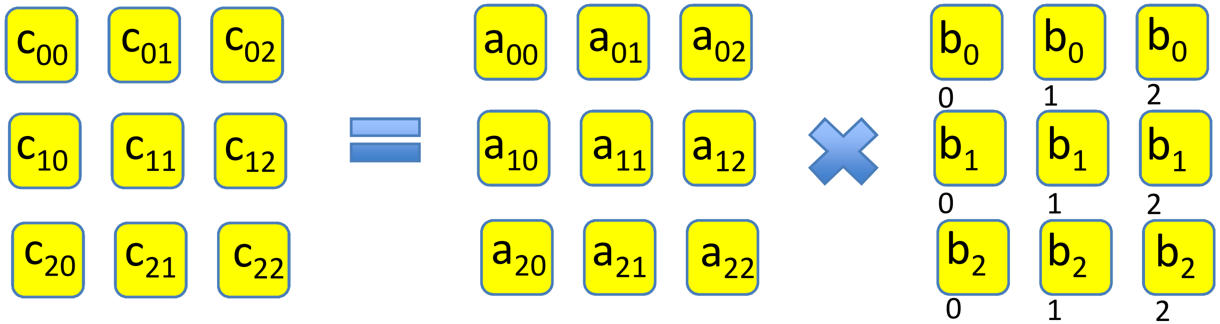



```
for n from 0 to N-1:
    for i from 0 to N-1:
        for j from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

n=0
i=0
j=0
C[0][0]+=A[0][0]*B[0][0]

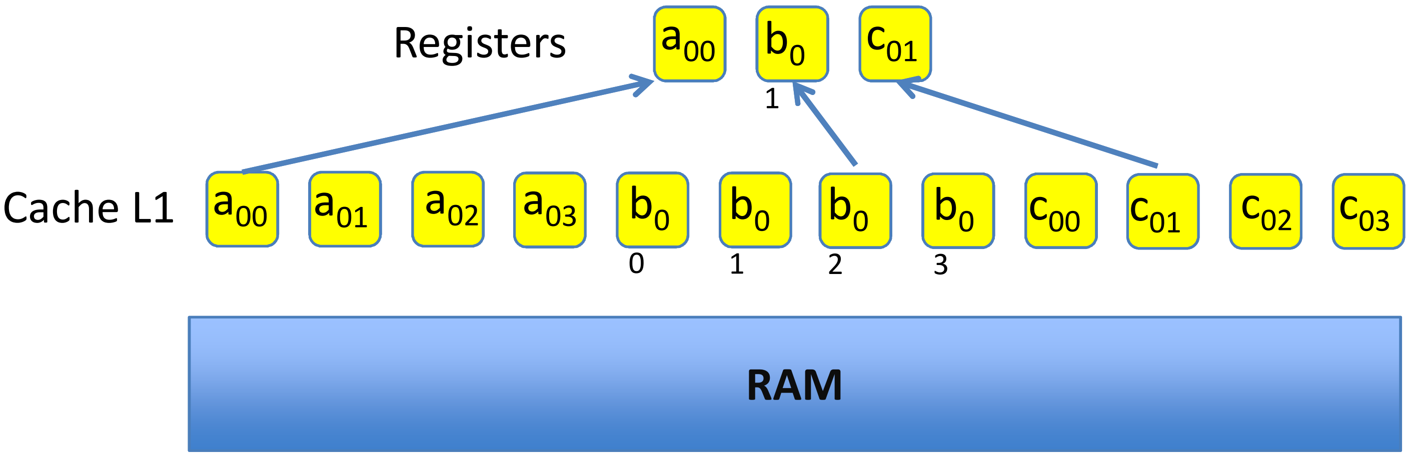


Naive matrix multiplication




```
for n from 0 to N-1:
    for i from 0 to N-1:
        for j from 0 to N-1:
            C[i][j]+=A[i][n]*B[n][j]
```

```
n=0
i=0
j=1
C[0][1]+=A[0][0]*B[0][1]
```



Parallel intuition

- Examples of parallelism in real world?

Parallel intuition
How to really understand concurrency.

Building a house

Say, we have 4 workers building a house

Construction will be faster than if it would be done by 1 worker

After each floor, workers have to synchronize

Connect their parts - communication

Can‘t build 7th floor before 3rd

Amdahl‘s law

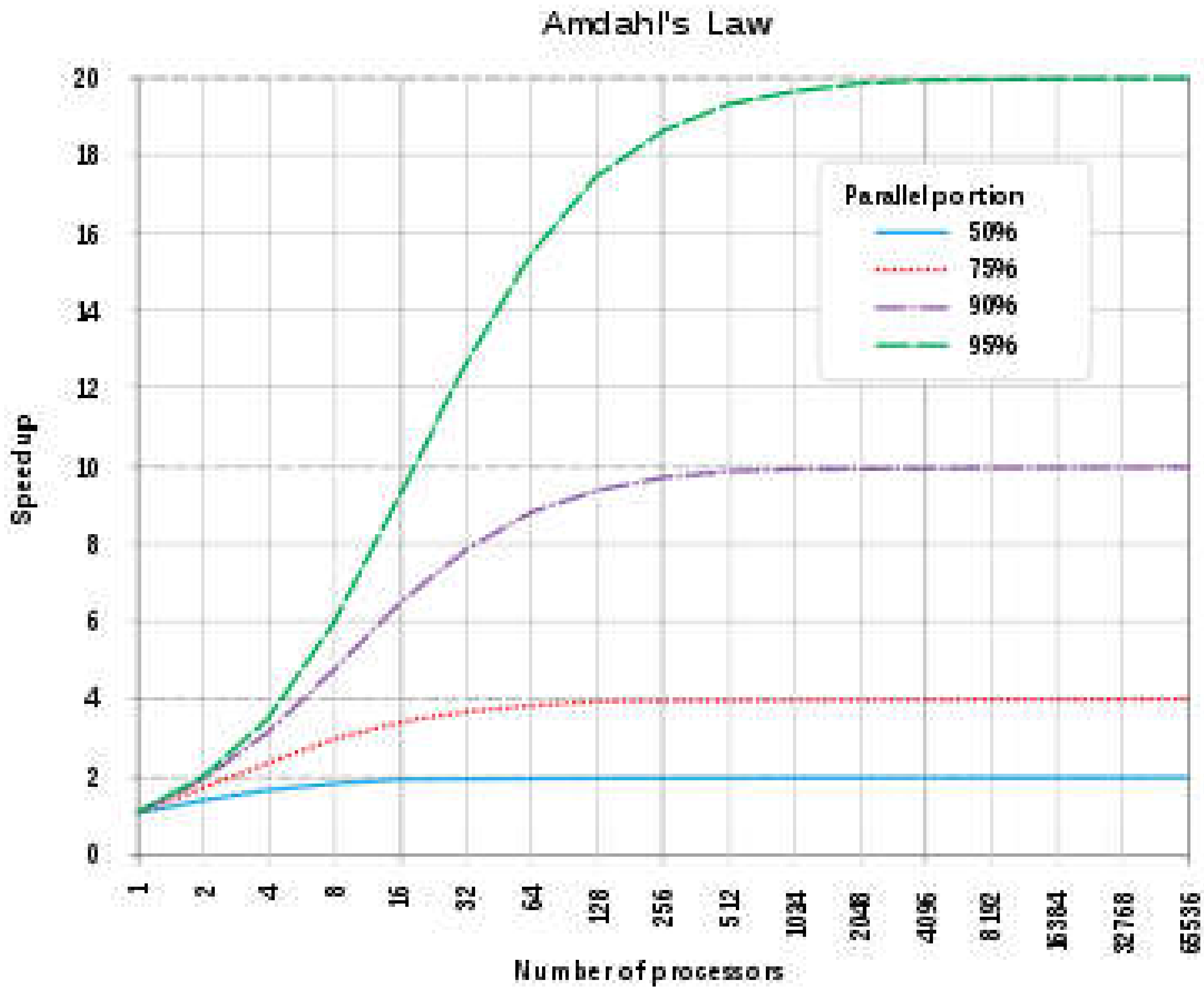
Speed-up of a parallel program/work:

$$S = \frac{1}{(1-f) + \frac{f}{n}}$$

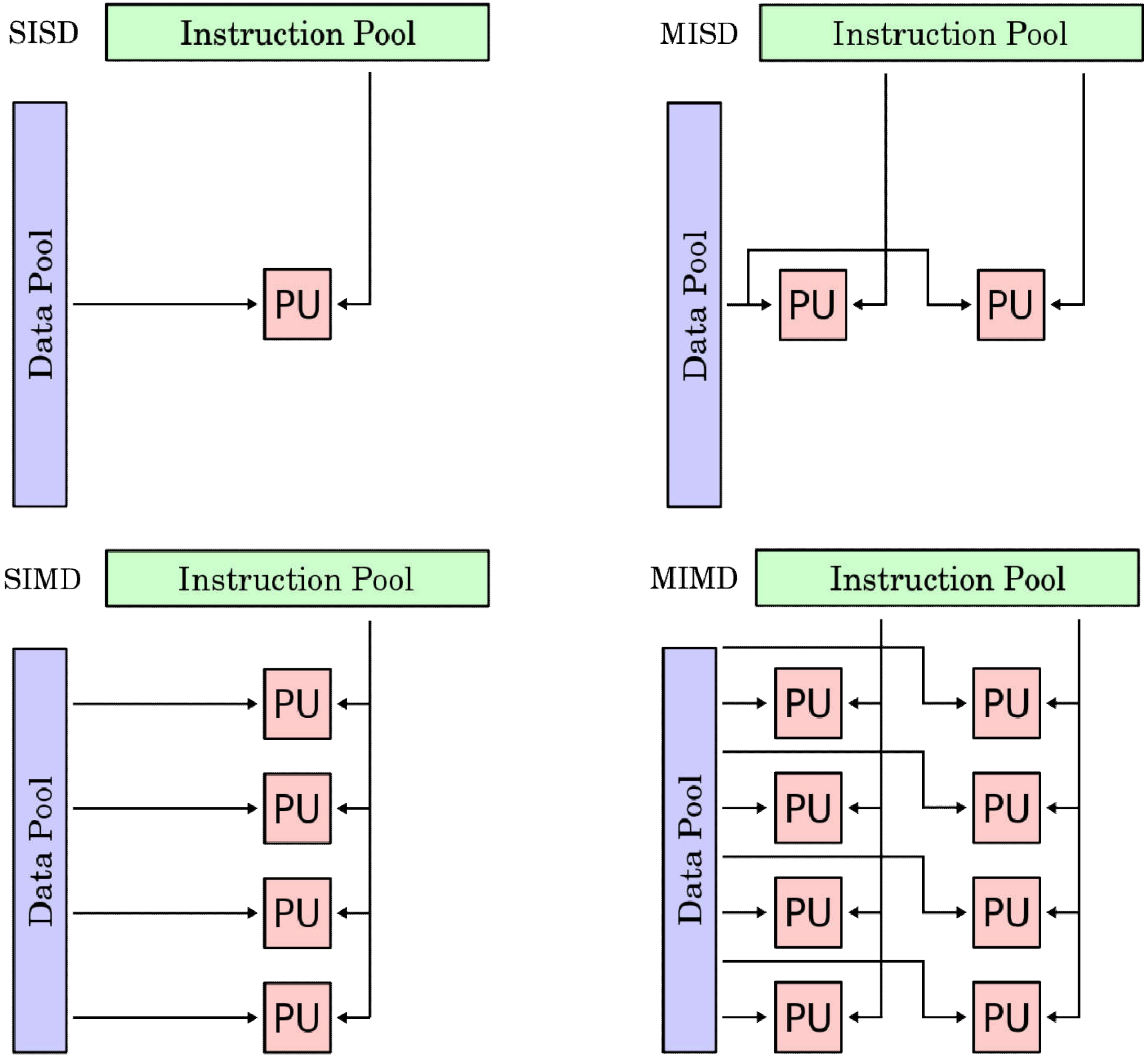
f – is a fraction of the work that can be done in parallel

n – number of parallel workers

Amdahl‘s law implication

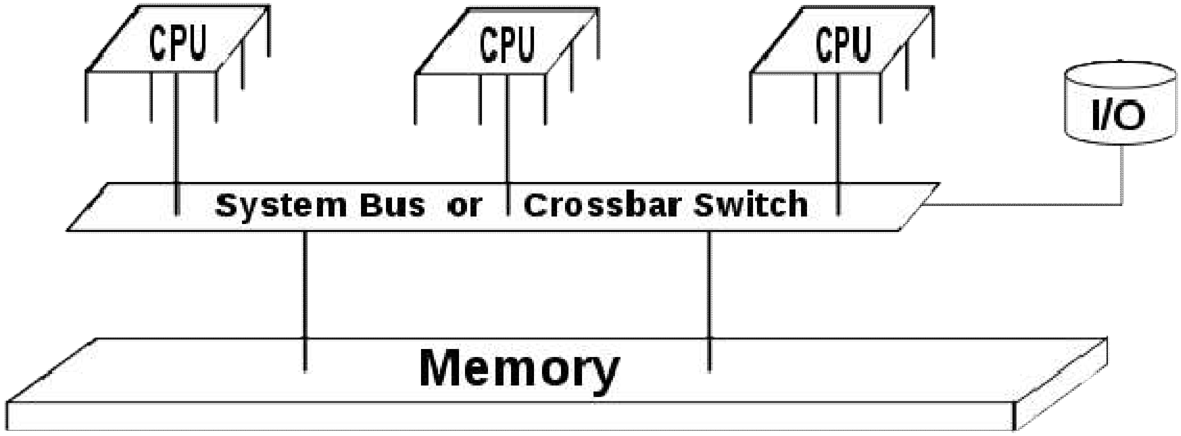


Flynn's taxonomy

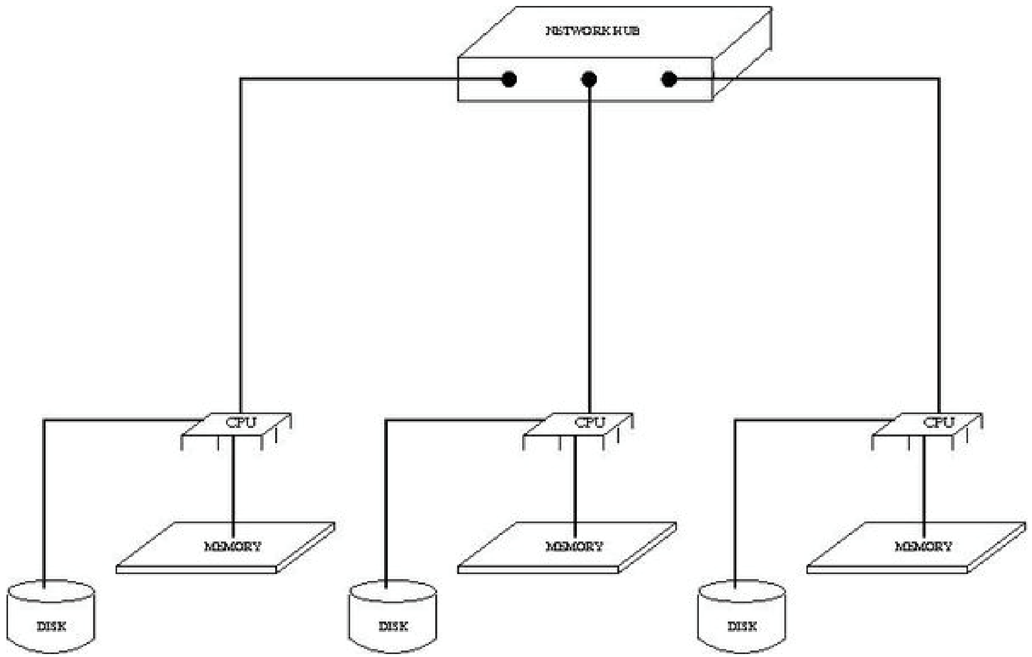


Shared memory and distributed memory

shared memory



distributed memory



Ways to work parallel

1. Scientific software (Matlab, Mathematica, ANSYS, OpenFoam, Tensorflow, PyTorch, etc)

2. Libraries (Math Kernel Library, CuDNN, CuBlas, Thrust, Boost)

3. Parallel instruments (OpenMP, MPI, Cuda, OpenCL)

Shared memory systems: processes and threads

Shared memory systems: processes and threads