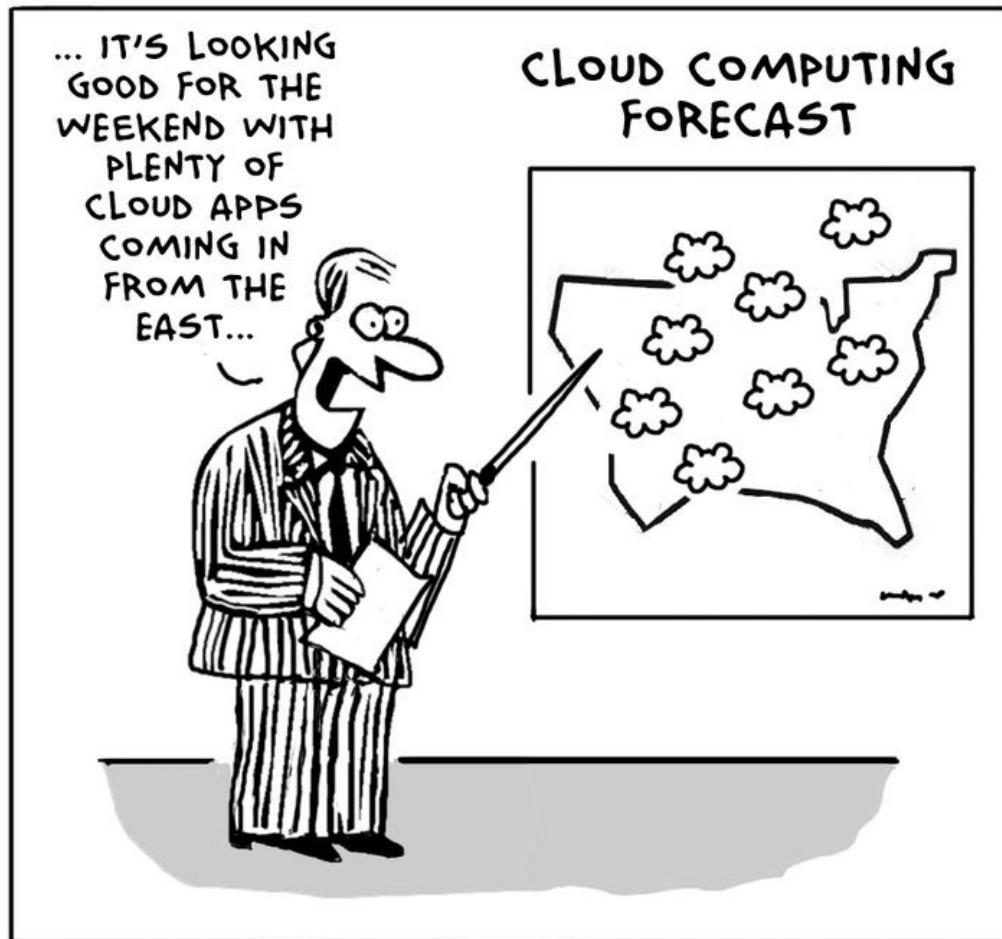


Computational Astrophysics

Computer Architectures

Alexander Knebe, Universidad Autonoma de Madrid
Violeta González Pérez

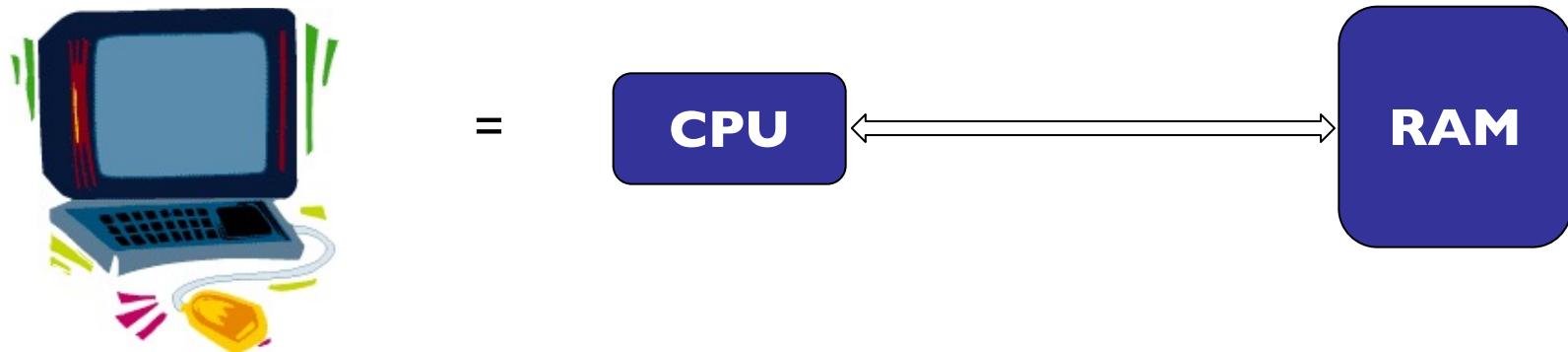


- architectures
- real machines
- computing concepts
- parallel programming

- **architectures**
- real machines
- computing concepts
- parallel programming

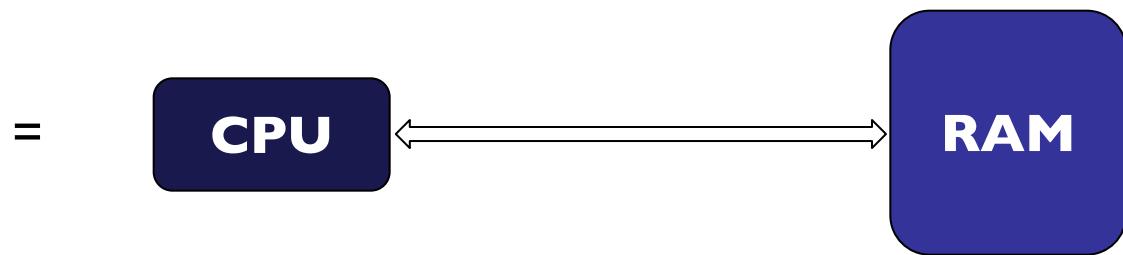
Computer Architectures

- serial machine



Computer Architectures

- serial machine

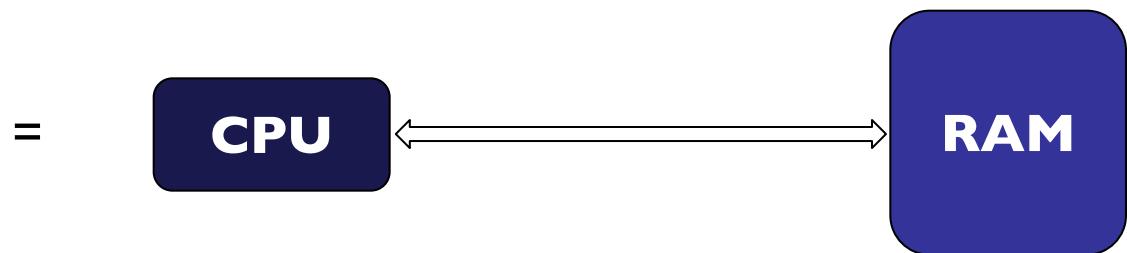


CPU:

- very primitive commands,
obtained from compilers or interpreters of higher-level languages

Computer Architectures

- serial machine

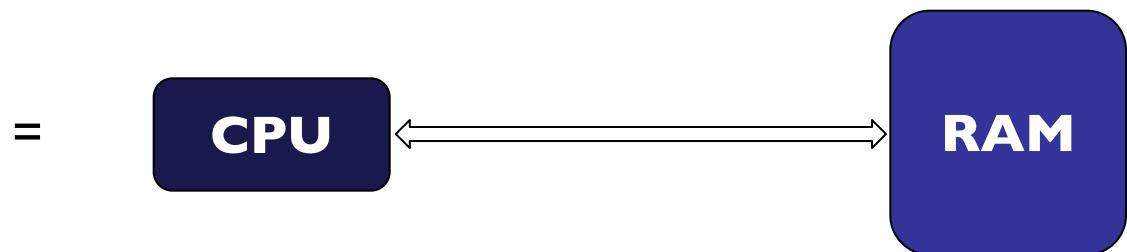


CPU:

- very primitive commands,
obtained from compilers or interpreters of higher-level languages
- cycle chain:
 - **fetch** – get instruction and/or data from memory
 - **decode** – store instruction and/or data in register
 - **execute** – perform instruction

Computer Architectures

- serial machine



CPU:

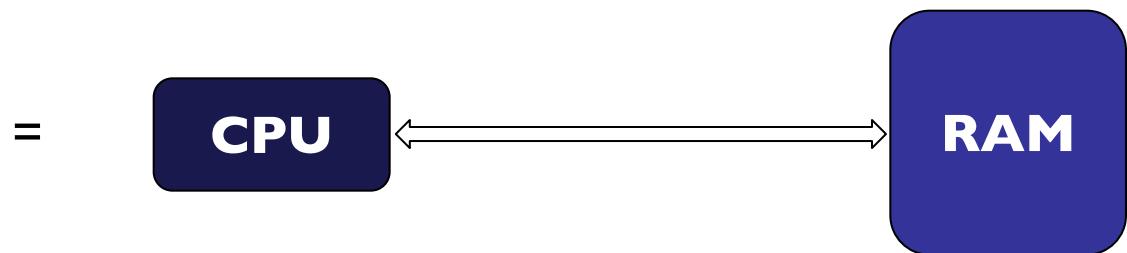
- very primitive commands,
obtained from compilers or interpreters of higher-level languages
- cycle chain:
 - **fetch** – get **instruction** and/or data from memory
 - **decode** – store **instruction** and/or data in register
 - **execute** – perform **instruction**

arithmetical/logical instructions: +, -, *, /, bitshift, if

What is a bitshift (>>, <<) good for?
Make a short C program to test this.

Computer Architectures

- serial machine

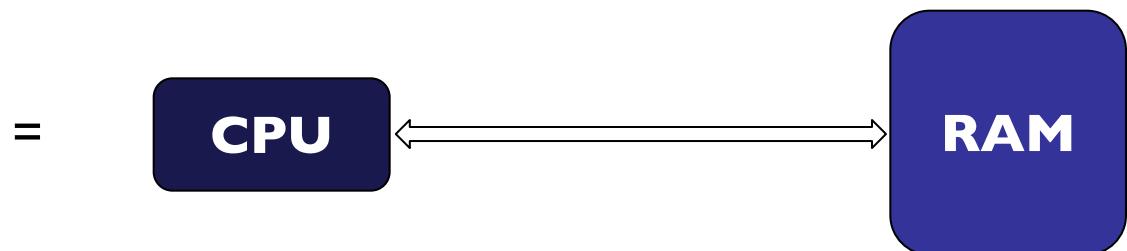


CPU:

- very primitive commands,
obtained from compilers or interpreters of higher-level languages
- cycle chain:
 - **fetch** – get instruction and/or data from memory
 - **decode** – store instruction and/or data in register
 - **execute** – perform instruction
- some CPU allow multi-threading,
i.e. already fetch next instruction while still executing

Computer Architectures

- serial machine



CPU:

- execution time: $t = n_i \times CPI \times t_c$

number of instructions

cycles per instruction
(e.g., '+' requires less cycles than '*')

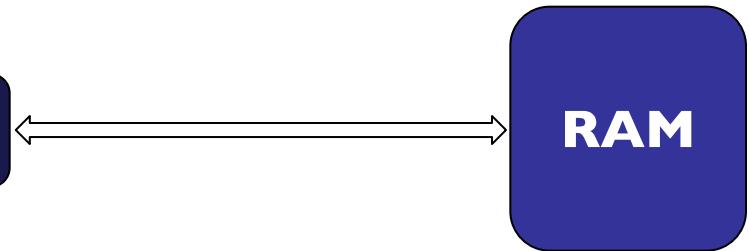
time per cycle

Computer Architectures

■ serial machine



=



CPU:

- execution time: $t = n_i \times CPI \times t_c$

number of instructions

cycles per instruction

time per cycle



speed-ups:

improve your algorithm to require less instructions

example:

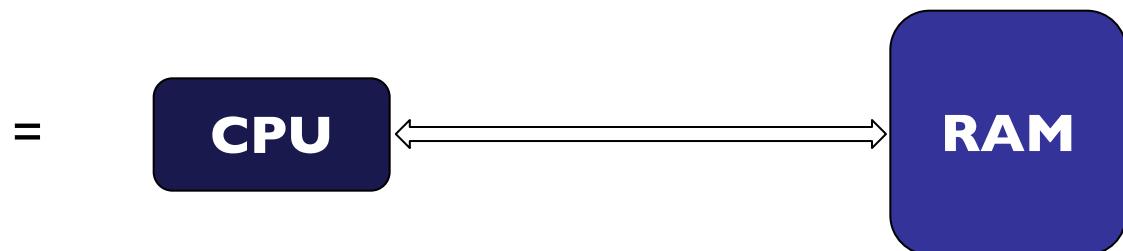
a factor like “3/(8piG)” inside a for-loop should be avoided;

define FAC=3/(8piG) outside the loop and use FAC inside the loop instead...

why?

Computer Architectures

- serial machine



CPU:

- execution time: $t = n_i \times CPI \times t_c$

number of instructions

cycles per instruction
(how many cycles does
your instruction require)



time per cycle

Complete the code to test this:

```
#include <stdio.h>
#include <math.h>
#include <time.h>
```

```
void main(){
    double x = 2.; speed-ups:
```

```
clock_t begin = clock();
double a = x*x*x;
clock_t end = clock();
double time = 1000.0*(end -
begin)/CLOCKS_PER_SEC; //ms
printf("Time x*x*x=%E ms \n",time);
```

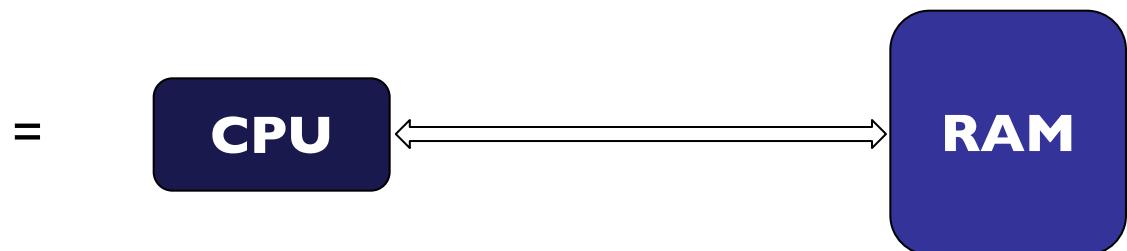
improve your algorithm to use more adequate instructions

example:

avoid at all costs pow(), log(), etc.,
e.g. pow(x, 2) should be replaced with x*x

Computer Architectures

- serial machine



CPU:

- execution time: $t = n_i \times CPI \times t_c$

number of instructions

cycles per instruction

time per cycle

speed-ups:

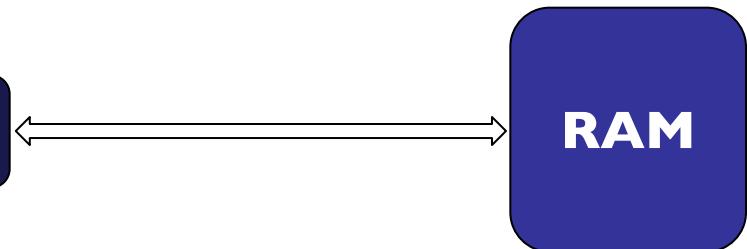
buy a machine with higher clock-frequency

Computer Architectures

■ serial machine



=



CPU:

- execution time: $t = n_i \times CPI \times t_c$

number of instructions

cycles per instruction

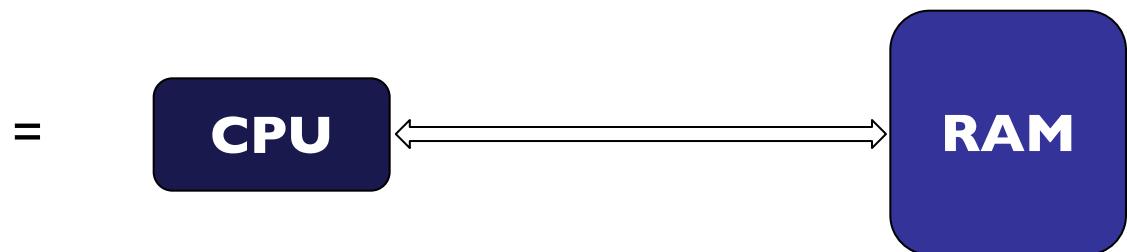
time per cycle

speed-ups:

improve your algorithm!

Computer Architectures

■ serial machine



CPU:

- execution time: $t = n_i \times CPI \times t_c$

number of instructions

cycles per instruction

time per cycle

Check the speed of your laptop!

\$ sudo lscpu

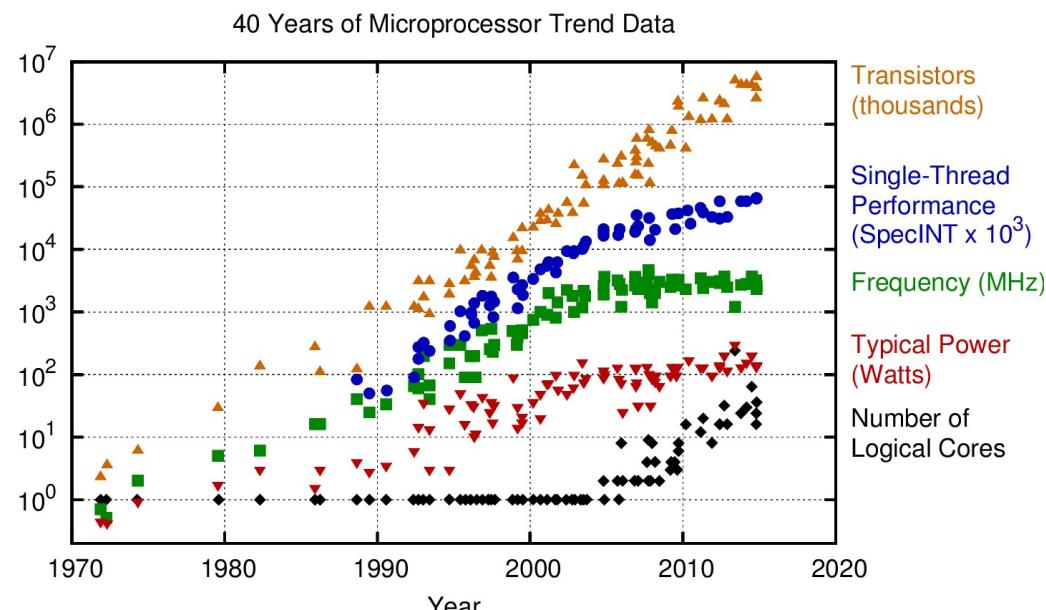
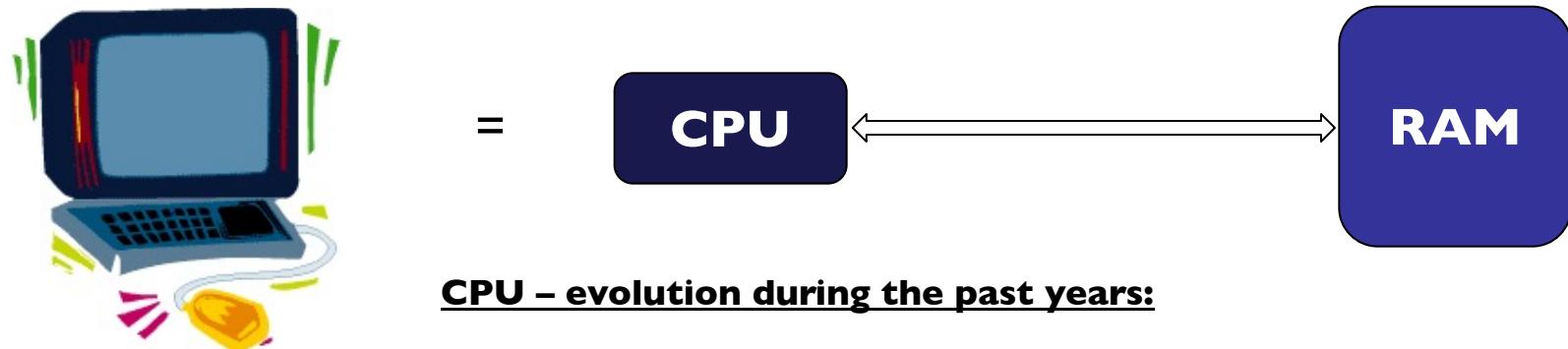
(search CPU MHz)

speed-ups:

...or wait for technology to advance ;-)

Computer Architectures

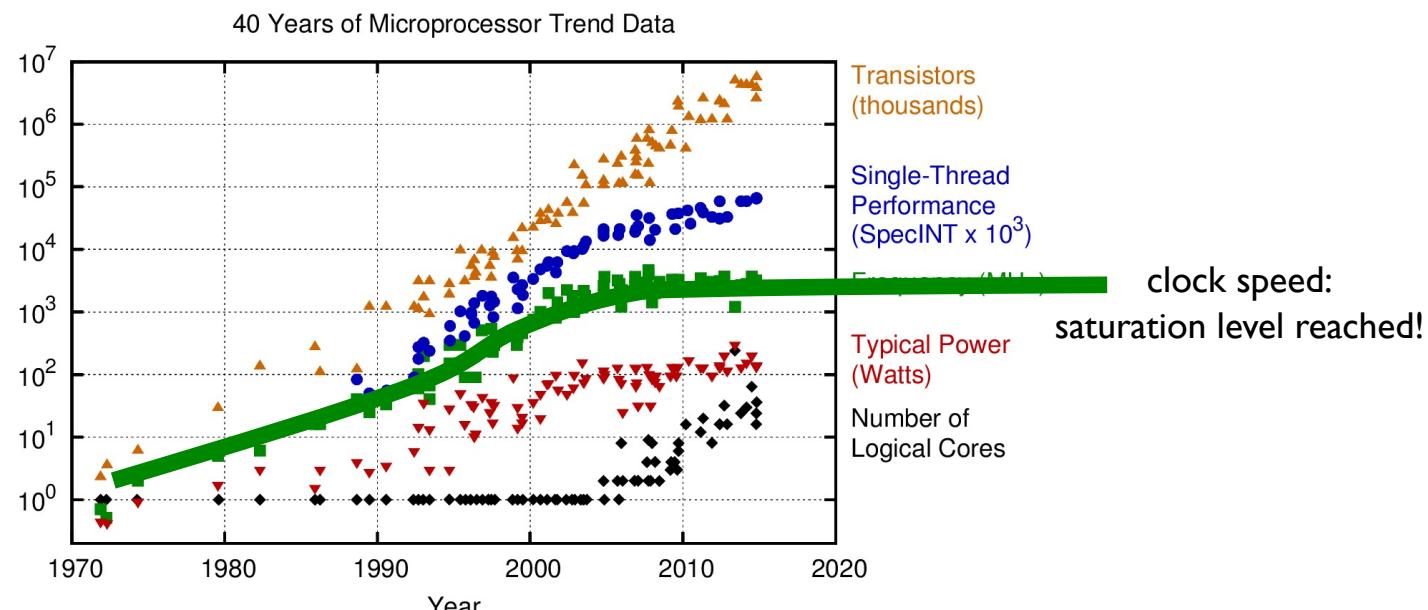
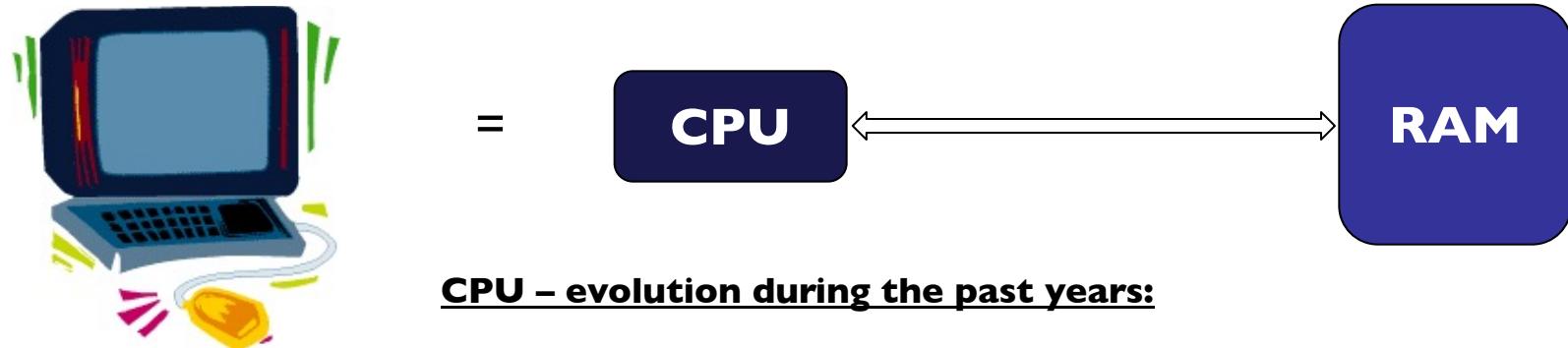
- serial machine



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

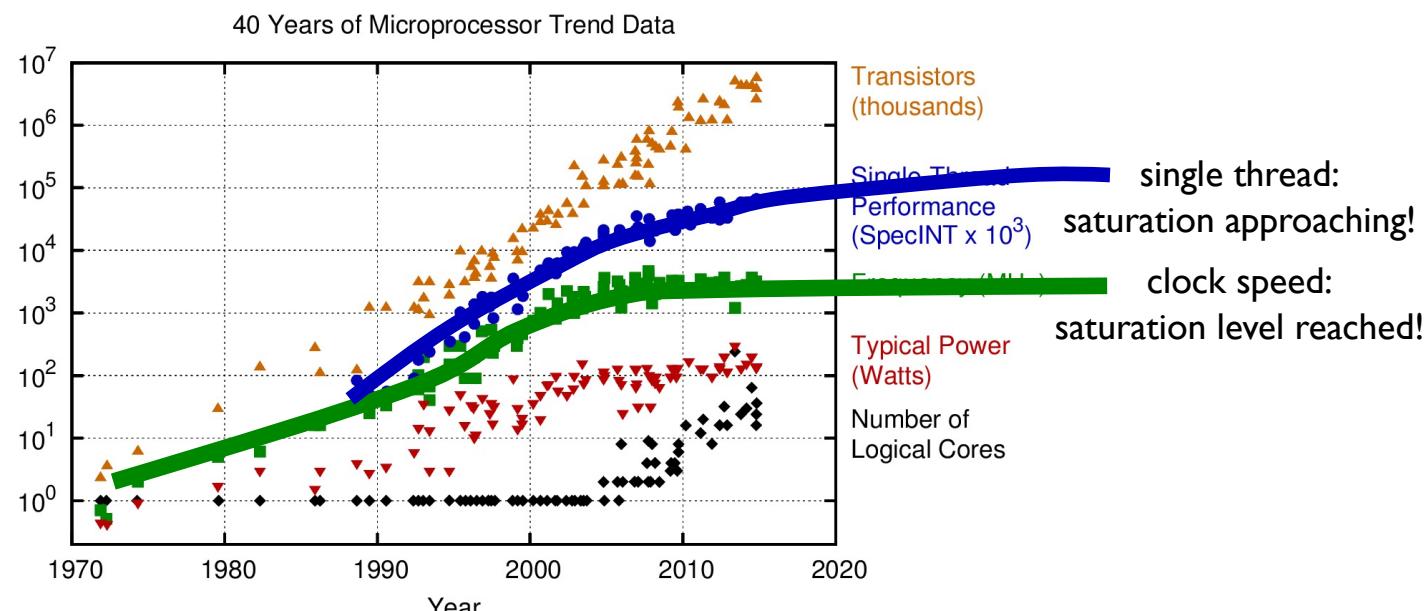
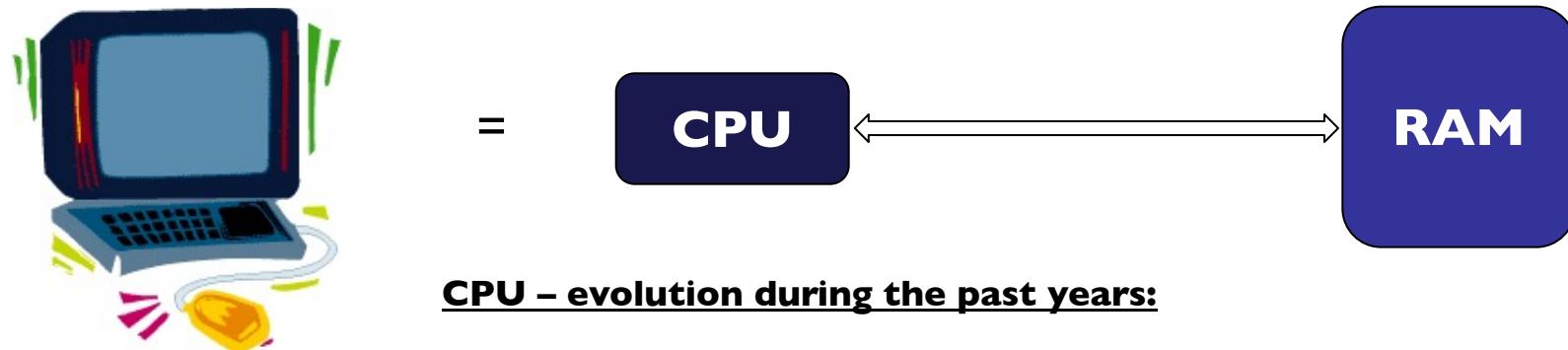
Computer Architectures

- serial machine



Computer Architectures

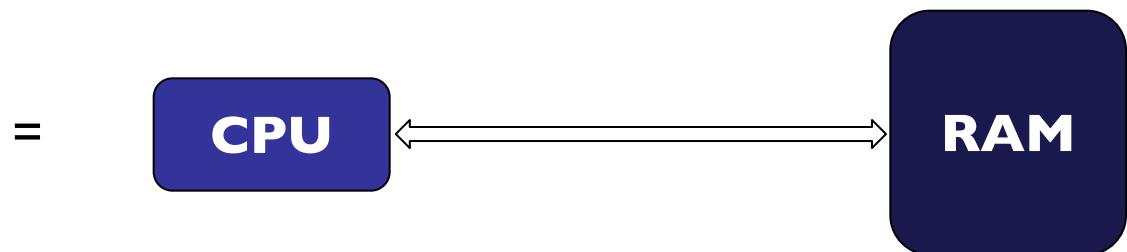
- serial machine



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Computer Architectures

- serial machine



RAM:

- Random Access Memory, i.e. read and write
- storage in binary system:
 - 1 bit = 0 or 1
 - 8 bits = 1 byte
 - 4 bytes = 1 float (=32 bits, standard for 32-bit architectures)
 - 8 bytes = 1 double (=64 bits, standard for 64-bit architectures)

Check the architecture of your laptop and the number of CPUs and threads you have:
\$ sudo lscpu

Computer Architectures

- serial machine



RAM:

- Random Access Memory, i.e. read and write
- storage in binary system:
 - 1 bit = 0 or 1
 - 8 bits = 1 byte
 - 4 bytes = 1 float (=32 bits, standard for 32-bit architectures)
 - 8 bytes = 1 double (=64 bits, standard for 64-bit architectures)
- latency = time for memory access (bus width also relevant)

Computer Architectures

- serial machine



RAM:

- Random Access Memory, i.e. read and write
- storage in binary system:
 - 1 bit = 0 or 1
 - 8 bits = 1 byte
 - 4 bytes = 1 float (=32 bits, standard for 32-bit architectures)
 - 8 bytes = 1 double (=64 bits, standard for 64-bit architectures)
- latency = time for memory access (bus width also relevant)
- speed-ups:
 - multi-threading CPU's
 - larger bus width

Computer Architectures

- serial machine



RAM:

- Random Access Memory, i.e. read and write
- storage in binary system:
 - 1 bit = 0 or 1
 - 8 bits = 1 byte
 - 4 bytes = 1 float (=32 bits, standard for 32-bit architectures)
 - 8 bytes = 1 double (=64 bits, standard for 64-bit architectures)
- latency = time for memory access (bus width also relevant)
- speed-ups:
 - multi-threading CPU's
 - **larger bus width:**
 - '80s 8-bit wide
 - '90s 16-bit wide
 - '00s 32-bit wide
 - today 64-bit wide

(internal 'highway' for data transfer)

Computer Architectures

- serial machine

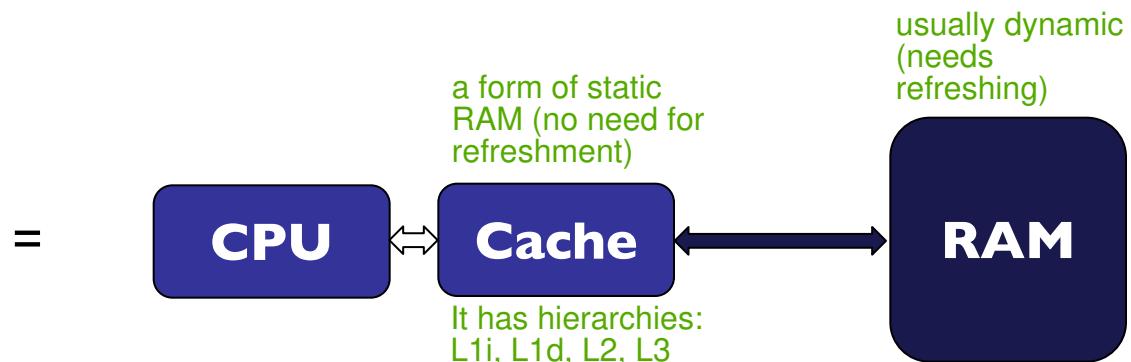


RAM:

- Random Access Memory, i.e. read and write
- storage in binary system:
 - 1 bit = 0 or 1
 - 8 bits = 1 byte
 - 4 bytes = 1 float (=32 bits, standard for 32-bit architectures)
 - 8 bytes = 1 double (=64 bits, standard for 64-bit architectures)
- latency = time for memory access (bus width also relevant)
- speed-ups:
 - multi-threading CPU's
 - larger bus width
 - clever usage of Cache

Computer Architectures

■ serial machine



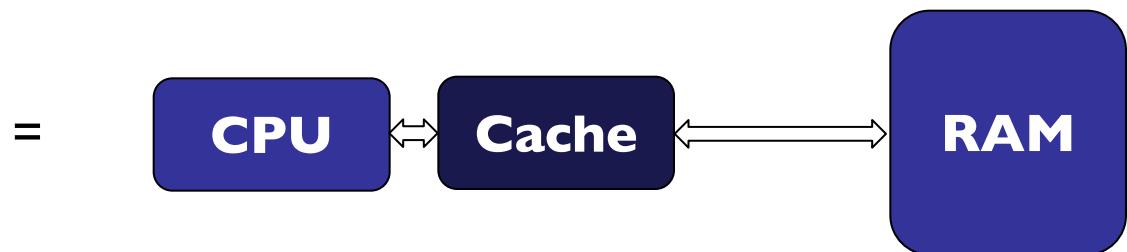
RAM:

- Random Access Memory, i.e. read and write
- storage in binary system:
 - 1 bit = 0 or 1
 - 8 bits = 1 byte
 - 4 bytes = 1 float (=32 bits, standard for 32-bit architectures)
 - 8 bytes = 1 double (=64 bits, standard for 64-bit architectures)
- latency = time for memory access (bus width also relevant)
- speed-ups:
 - multi-threading CPU's
 - larger bus width
 - **clever usage of Cache**

Explore the Cache of your laptop with
`$ sudo lscpu`
and compare it with your RAM:
`$ grep MemTotal /proc/meminfo`

Computer Architectures

- serial machine



Cache:

- Random Access Memory, i.e. read and write
- built into motherboard next to CPU
- when ‘fetch a[i]’ also ‘fetch a[i+1]’ into Cache (in fact, full lines or pages are “cached”)
- nowadays multiple Cache levels
- bad programming will lead to “Cache misses”:

$$\bullet \quad t = f \times t_{\text{Cache}} + (1-f) \times t_{\text{RAM}}$$

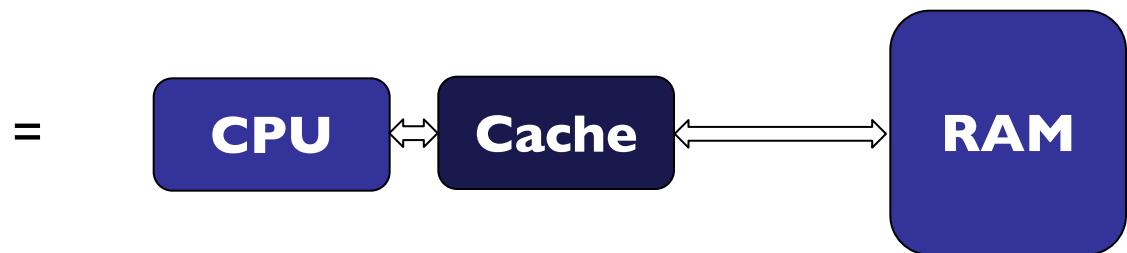
Cache hit rate

Cache access time

RAM access time

Computer Architectures

- serial machine



Cache:

- Random **A**ccess **M**emory, i.e. read and write
- built into motherboard next to CPU
- when ‘fetch a[i]’ also ‘fetch a[i+1]’ into Cache (in fact, full lines or pages are “cached”)
- nowadays multiple Cache levels
- bad programming will lead to “Cache misses”:

$$\bullet \quad t = f \times t_{\text{Cache}} + (1-f) \times t_{\text{RAM}}$$

\swarrow \searrow \searrow
Cache hit rate **Cache access time** **RAM access time**

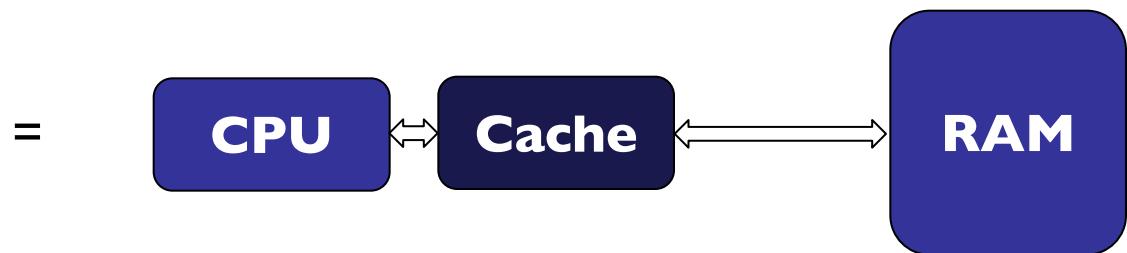
example:

$$t_{\text{Cache}} = 10 \text{ ns}, \quad t_{\text{RAM}} = 100 \text{ ns}$$

$$t_{\text{Cache}} = 10 \text{ ns}, \quad t_{\text{RAM}} = 100 \text{ ns}$$

Computer Architectures

- serial machine



Cache:

- Random Access Memory, i.e. read and write
- built into motherboard next to CPU
- when ‘fetch a[i]’ also ‘fetch a[i+1]’ into Cache (in fact, full lines or pages are “cached”)
- nowadays multiple Cache levels
- bad programming will lead to “Cache misses”:

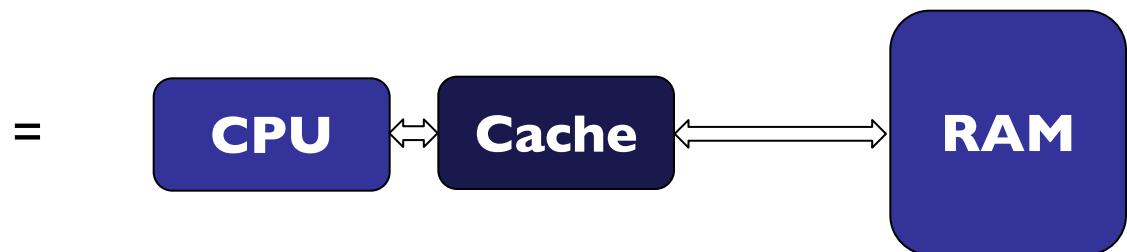
$$\bullet \quad t = f \times t_{\text{Cache}} + (1-f) \times t_{\text{RAM}}$$

\swarrow \searrow \searrow
Cache hit rate **Cache access time** **RAM access time**

example: $f = 0.1, \quad t_{\text{Cache}} = 10\text{ns}, \quad t_{\text{RAM}} = 100\text{ns}$
 $f = 0.9, \quad t_{\text{Cache}} = 10\text{ns}, \quad t_{\text{RAM}} = 100\text{ns}$

Computer Architectures

- serial machine



Cache:

- Random Access Memory, i.e. read and write
- built into motherboard next to CPU
- when ‘fetch a[i]’ also ‘fetch a[i+1]’ into Cache (in fact, full lines or pages are “cached”)
- nowadays multiple Cache levels
- bad programming will lead to “Cache misses”:

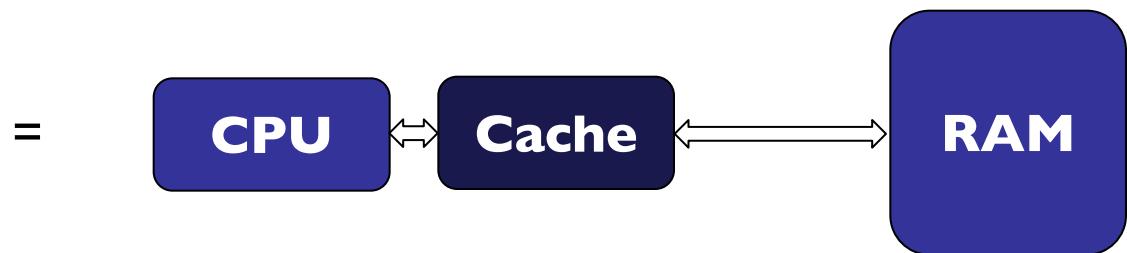
$$\bullet \quad t = f \times t_{\text{Cache}} + (1-f) \times t_{\text{RAM}}$$

\swarrow \downarrow \searrow
Cache hit rate **Cache access time** **RAM access time**

example: $f = 0.1, \quad t_{\text{Cache}} = 10\text{ns}, \quad t_{\text{RAM}} = 100\text{ns} \Rightarrow 91\text{ns}$
 $f = 0.9, \quad t_{\text{Cache}} = 10\text{ns}, \quad t_{\text{RAM}} = 100\text{ns} \Rightarrow 19\text{ns}$

Computer Architectures

- serial machine



Cache:

- Random Access Memory, i.e. read and write
- built into motherboard next to CPU
- when ‘fetch a[i]’ also ‘fetch a[i+1]’ into Cache (in fact, full lines or pages are “cached”)
- nowadays multiple Cache levels
- bad programming will lead to “Cache misses”:

$$\bullet \quad t = f \times t_{\text{Cache}} + (1-f) \times t_{\text{RAM}}$$

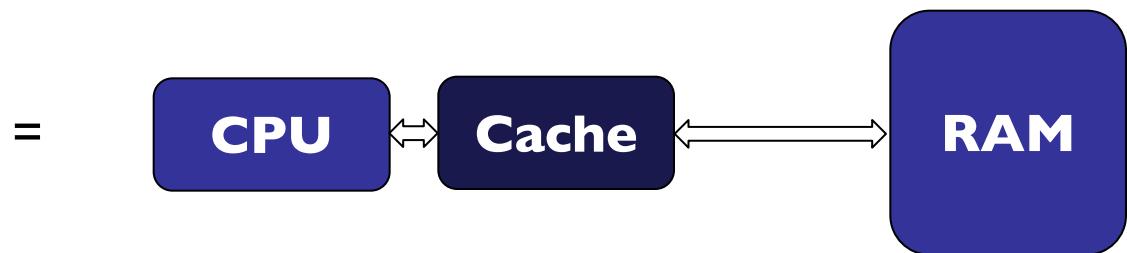
\swarrow \searrow \searrow
Cache hit rate **Cache access time** **RAM access time**

example: $f = 0.1, \quad t_{\text{Cache}} = 10\text{ns}, \quad t_{\text{RAM}} = 100\text{ns} \Rightarrow 91\text{ns}$ ↗ **factor of 4.5!**

$f = 0.9, \quad t_{\text{Cache}} = 10\text{ns}, \quad t_{\text{RAM}} = 100\text{ns} \Rightarrow 19\text{ns}$ ↗ **factor of 4.5!**

Computer Architectures

- serial machine



Cache:

- Random Access Memory, i.e. read and write
- built into motherboard next to CPU
- when ‘fetch a[i]’ also ‘fetch a[i+1]’ into Cache (in fact, full lines or pages are “cached”)
- nowadays multiple Cache levels
- bad programming will lead to “Cache misses”:

$$\bullet \quad t = f \times t_{\text{Cache}} + (1-f) \times t_{\text{RAM}}$$

Cache hit rate

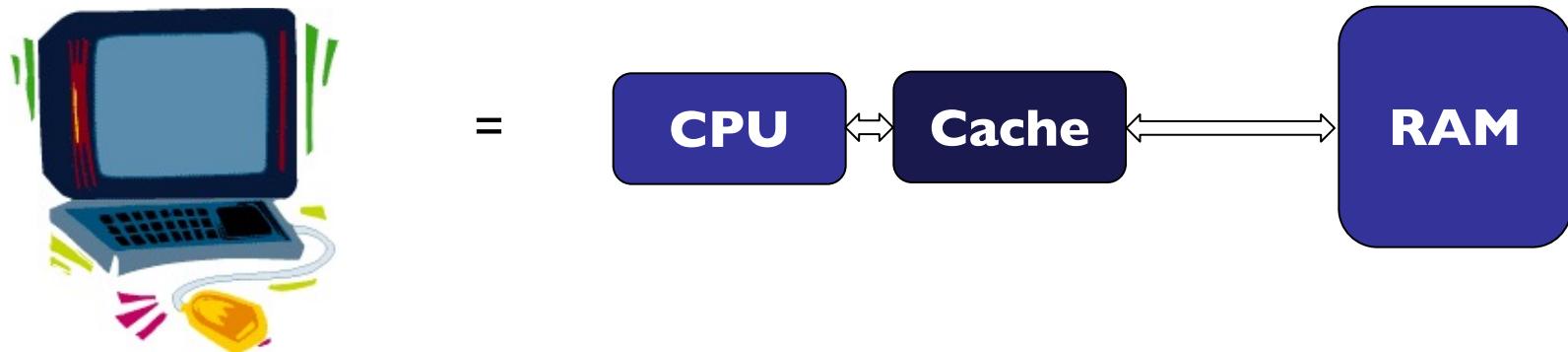
Cache access time

RAM access time

design your code in order to access contiguous memory blocks!

Computer Architectures

- serial machine

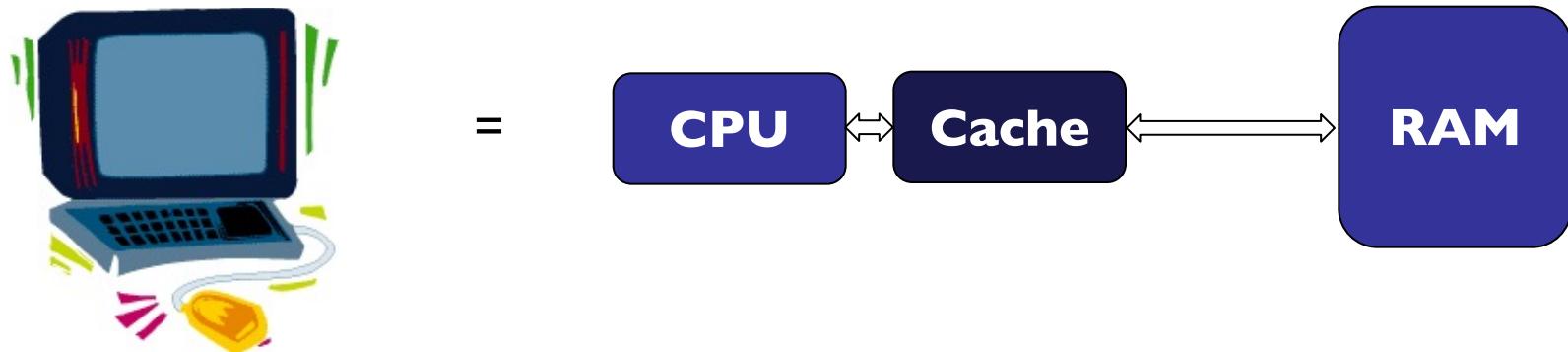


- Cache hits and misses:

- 2D array in C : `density[2][3]`

Computer Architectures

- serial machine



- Cache hits and misses:

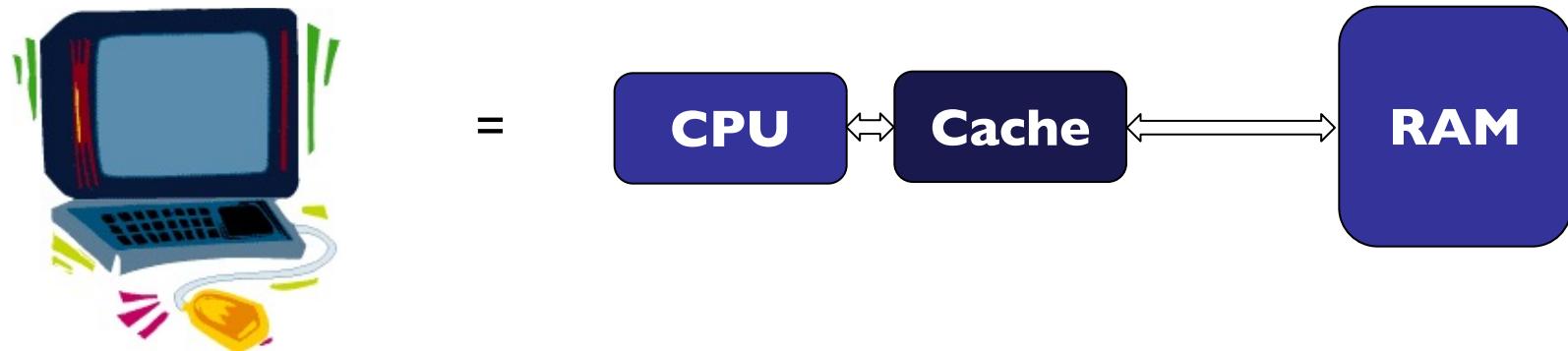
- 2D array in C : `density[2][3]`

- memory alignment:

--	--	--	--	--	--

Computer Architectures

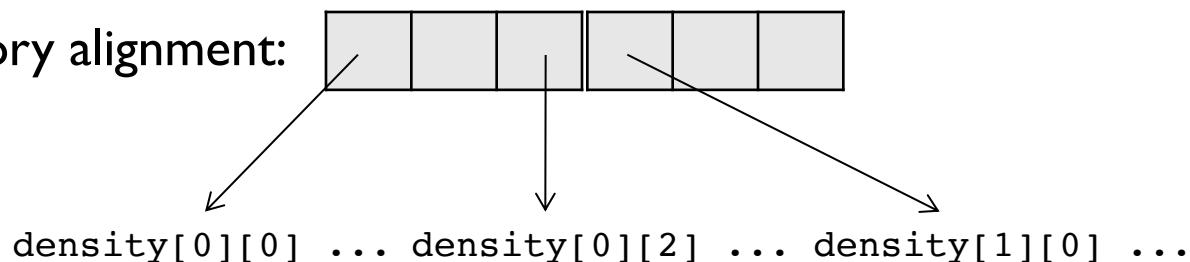
- serial machine



- Cache hits and misses:

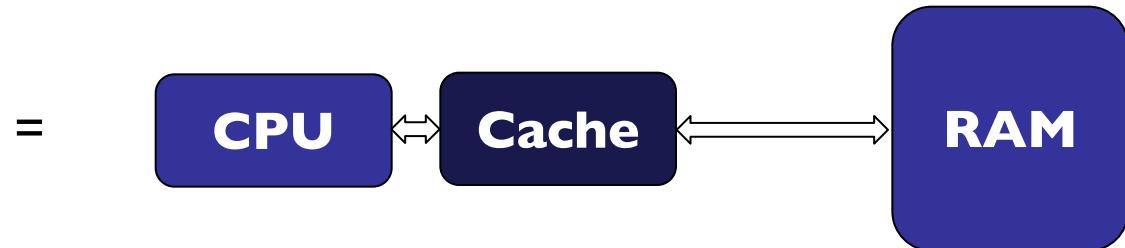
- 2D array in C : `density[2][3]`

- memory alignment:



Computer Architectures

- serial machine



- Cache hits and misses:

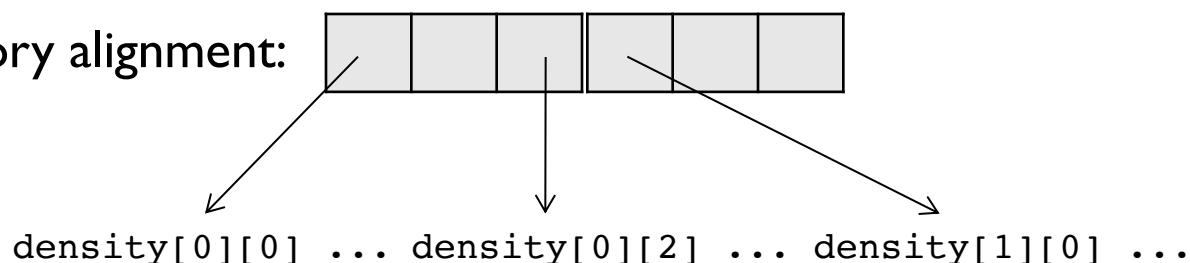
- 2D array in C:

density[2][3]

```
for(i=0; i<3; i++)
    for(j=0; j<2; j++)
        whatever(density[j][i]);
```

BAD!

- memory alignment:



Computer Architectures

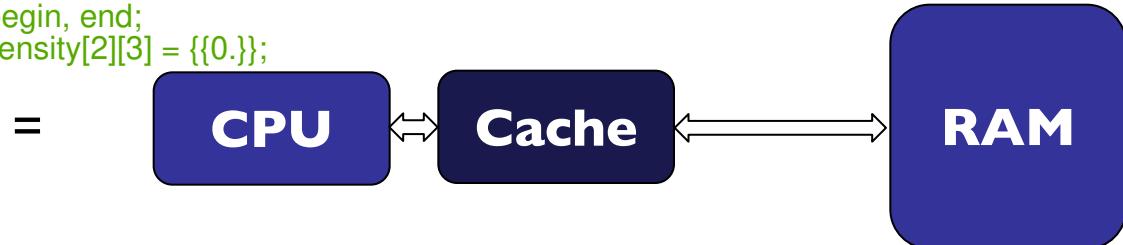
- serial machine



Test by completing the code:

```
#include <stdio.h>
#include <stdlib.h> Populate the array with random numbers from 0 to 99 with rand()%100
#include <time.h>
```

```
void main(){
    int i=0, j=0;
    double time=0.;
    clock_t begin, end;
    double density[2][3] = {{0.}};
```



- Cache hits and misses:

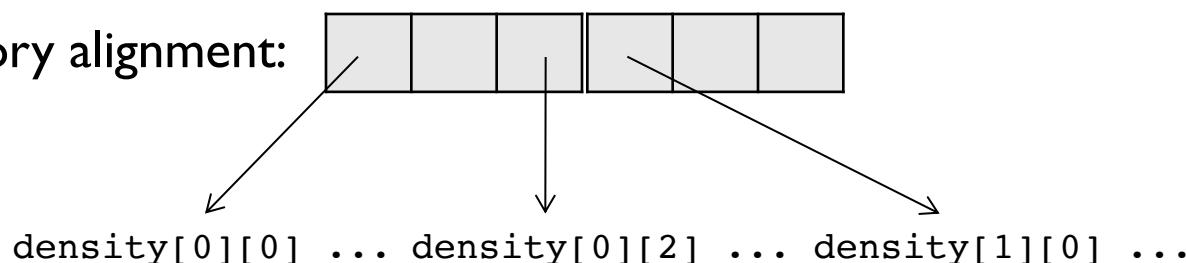
- 2D array in C:

density[2][3]

```
for(j=0; j<2; j++)
    for(i=0; i<3; i++)
        whatever(density[j][i]);
```

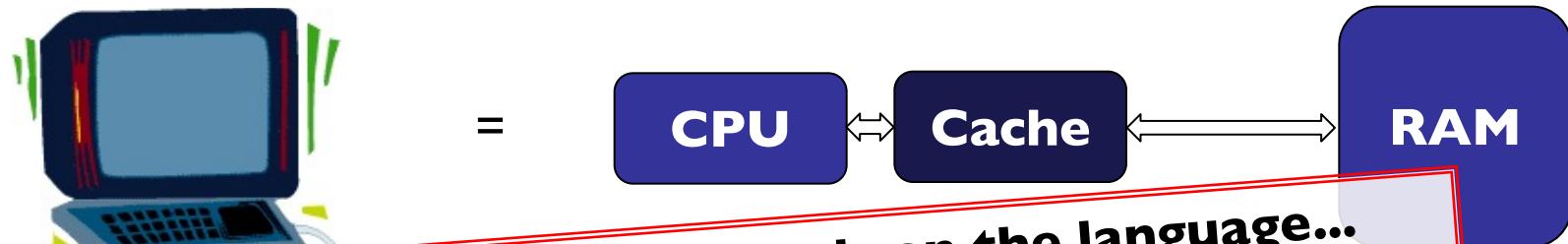
GOOD!

- memory alignment:



Computer Architectures

- serial machine

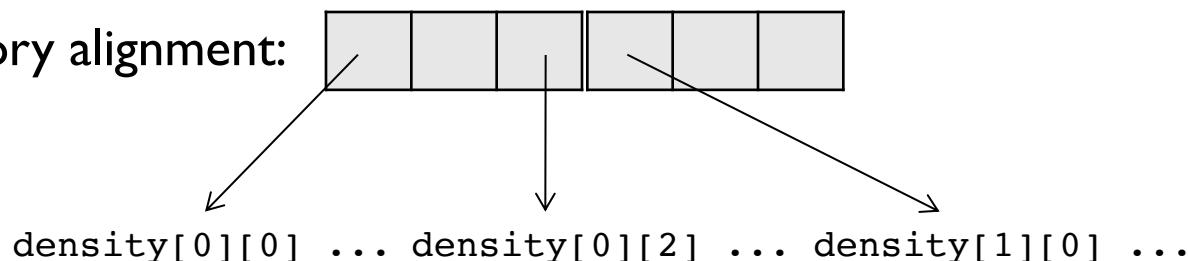


**memory alignment depends on the language...
check before programming!**

- Cache hits and misses
(e.g. C uses row-major ordering while Fortran & MATLAB use column-major ordering)
+ Python
+ Julia
- ```
for(j=0; j<2; j++)
 for(i=0; i<3; i++)
 whatever(density[j][i]);
```

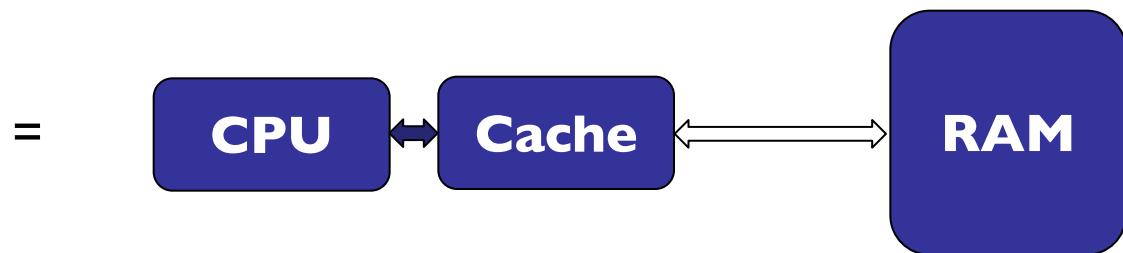
**GOOD!**

- memory alignment:



## Computer Architectures

- serial machine

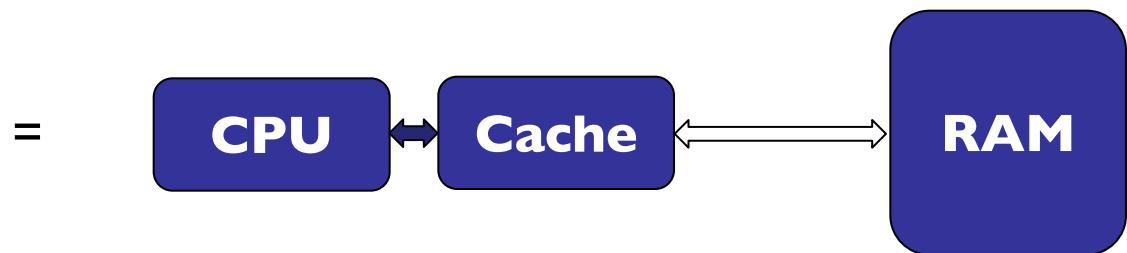


### CPU clock frequency vs. bus frequency:

- CPU frequency determines execution speed of commands
- bus frequency determines how quickly to get new commands/data

## Computer Architectures

- serial machine

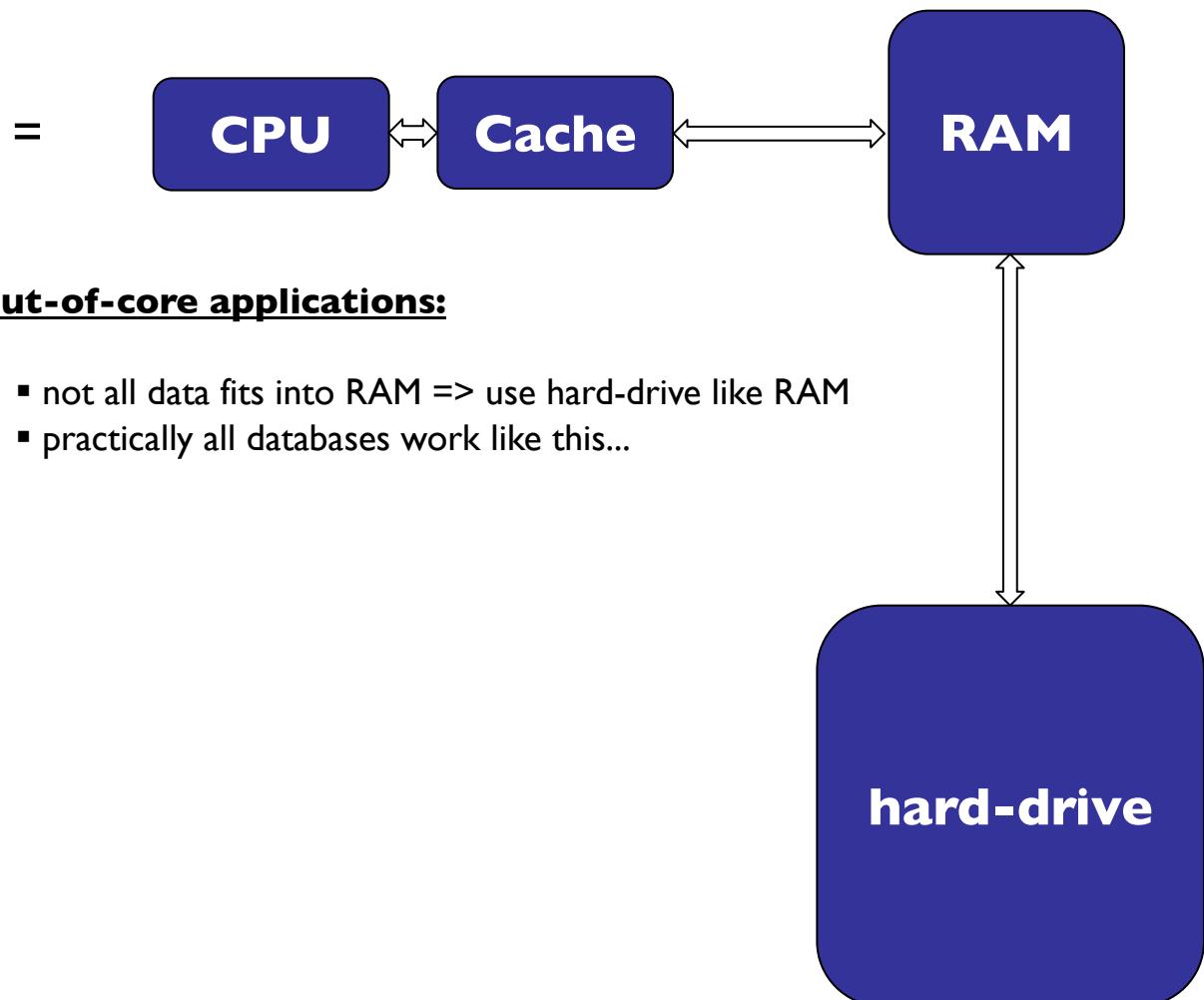


### CPU clock frequency vs. bus frequency:

- CPU frequency determines execution speed of commands
  - bus frequency determines how quickly to get new commands/data
- => bus frequency (and width) is more relevant for actual speed!

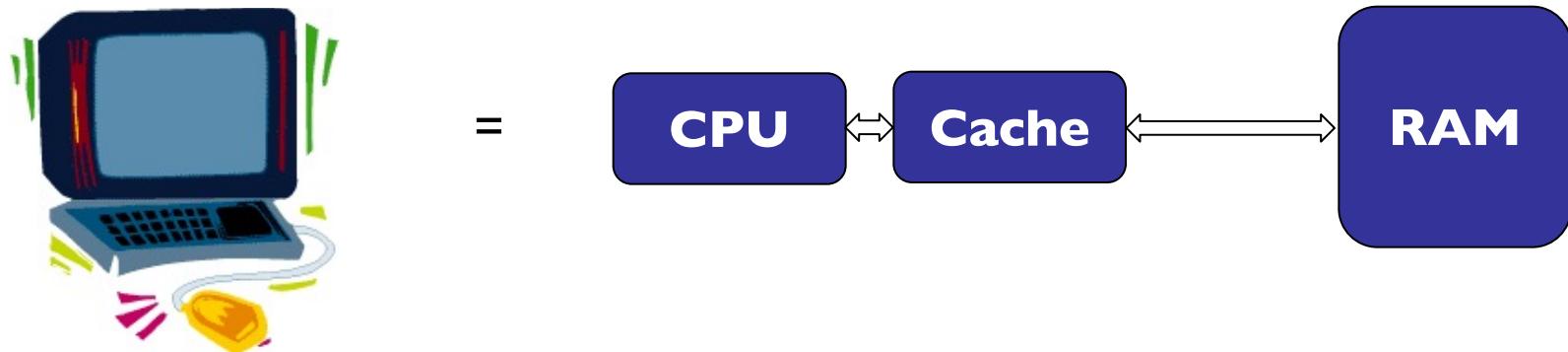
## Computer Architectures

- serial machine



## Computer Architectures

- serial machine

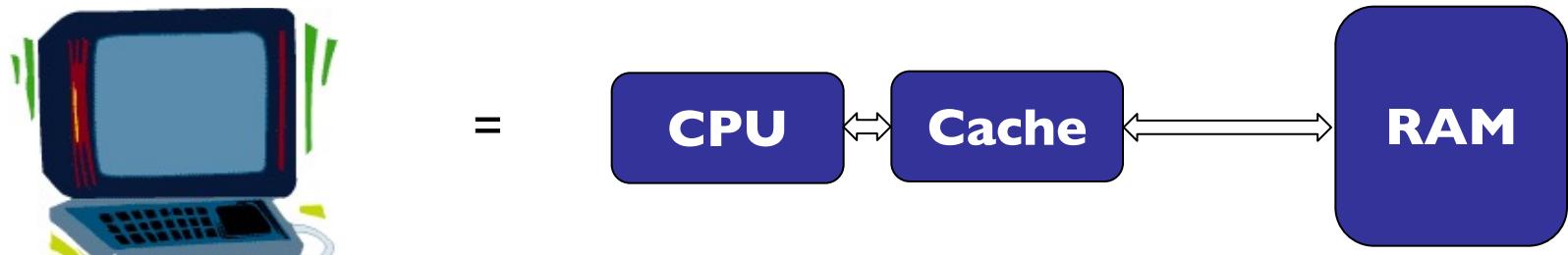


possible speed-ups by the programmer:

- improve your algorithm to require less instructions, e.g.  $f=4*\pi/\text{Grav}$
- improve your algorithm to use more adequate instructions, e.g.  $x*x$  instead of  $\text{pow}(x,2)$
- proper usage of cache, e.g. check memory alignment

## Computer Architectures

- serial machine



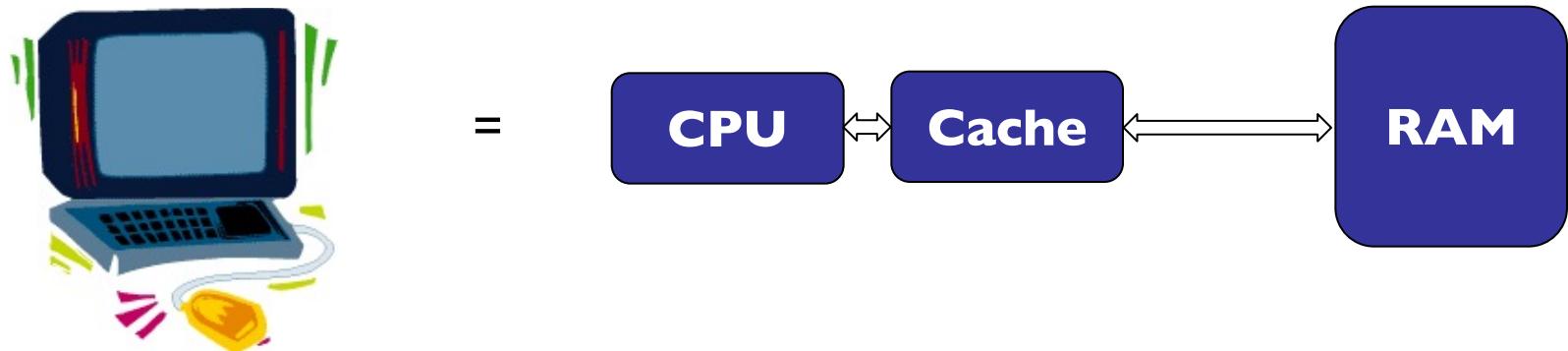
**any other possibility to speed things up?**

possible speed-ups by the programmer:

- improve your algorithm to require less instructions, e.g.  $f=4*\pi/\text{Grav}$
- improve your algorithm to use more adequate instructions, e.g.  $x*x$  instead of  $\text{pow}(x,2)$
- proper usage of cache, e.g. check memory alignment

## Computer Architectures

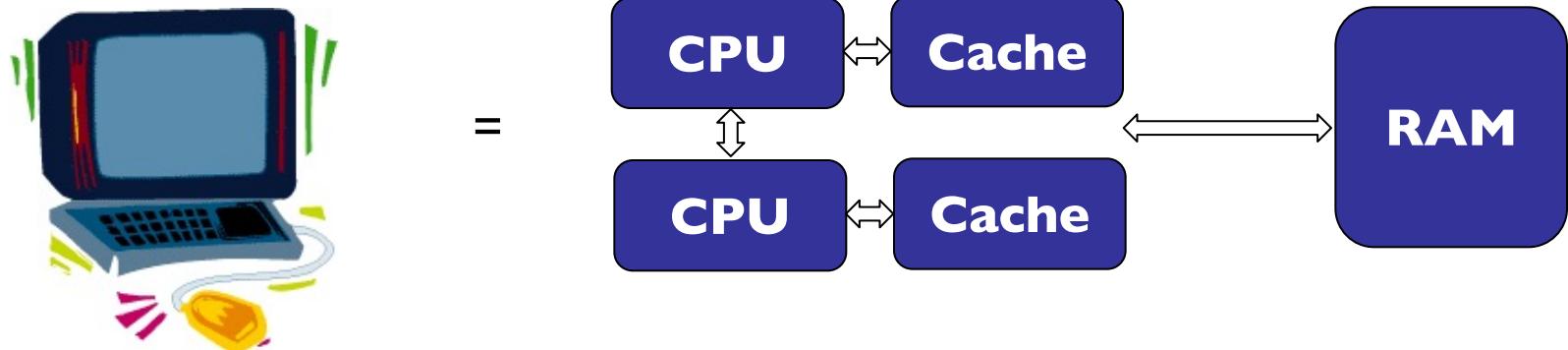
- serial machine



Accelerate your code by using GPUs, which are highly parallelised. Not explored here.

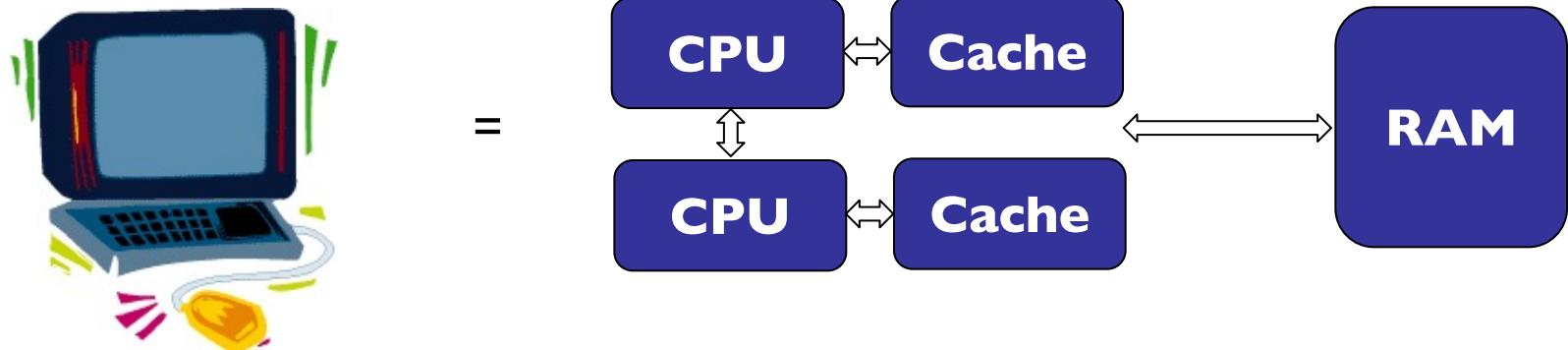
## Computer Architectures

- serial machine – multi-cored



## Computer Architectures

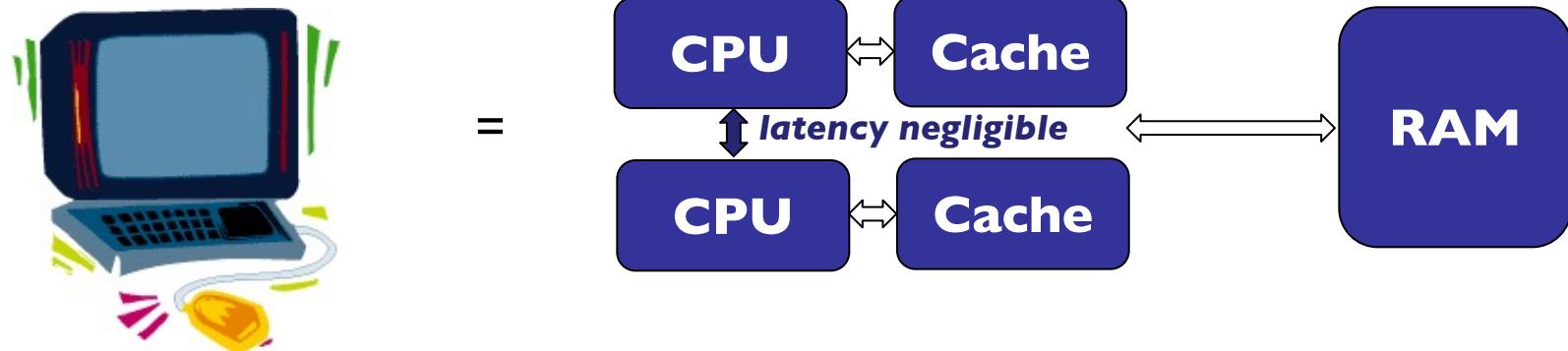
- serial machine – multi-cored



- shared memory architecture
  - easy to adapt existing serial code
  - limited by RAM to be placed into a single machine

## Computer Architectures

- serial machine – multi-cored

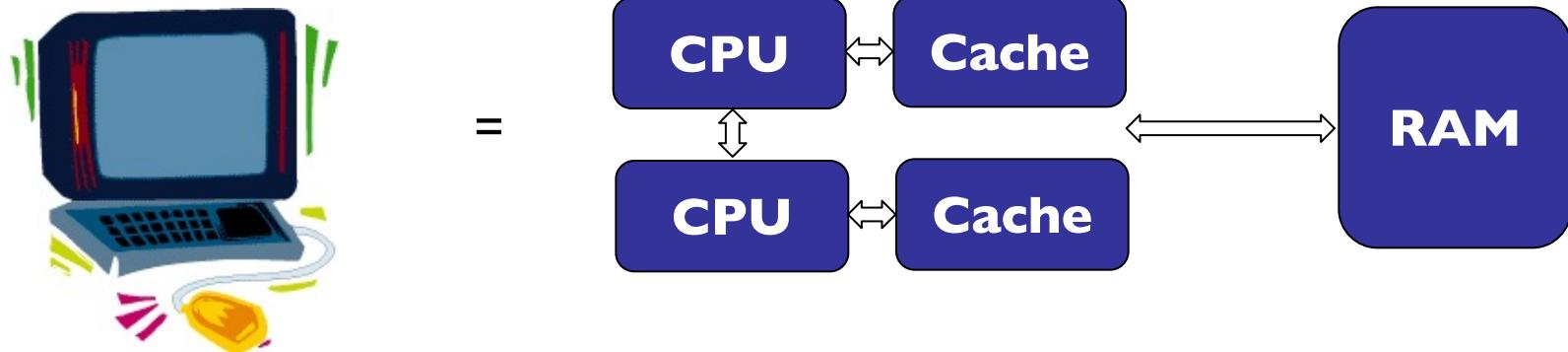


- shared memory architecture
  - easy to adapt existing serial code
  - limited by RAM to be placed into a single machine

How do we make use of the multicore capabilities?  
Note that if you compile normally, you'll just use 1 CPU

## Computer Architectures

- serial machine – multi-cored



- shared memory architecture
  - easy to adapt existing serial code
  - limited by RAM to be placed into a single machine
- OpenMP – [www.openmp.org](http://www.openmp.org)
  - most commonly used standard to parallelize code on shared memory architectures
  - primarily distribute for-loop components onto different CPU's
  - natively supported by gcc since v4.2

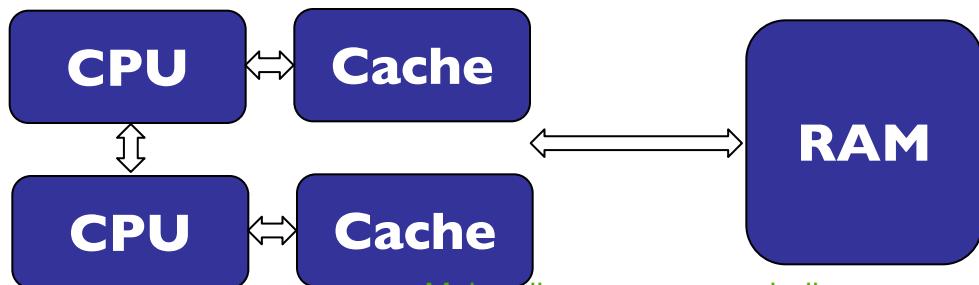
Check the version of your compiler:  
\$ gcc --version

## Computer Architectures

- serial machine – multi-cored



=



Make all your cores say hello:  
\$ gcc -o p -fopenmp test\_openmp.c

```
#include <stdio.h>
#include <omp.h>
```

```
void main(){
 #pragma omp parallel
 {
 int ID = omp_get_thread_num();
 printf("hello(%d)", ID);
 printf("world(%d)\n", ID);
 }
}
```

- shared memory architecture
  - easy to adapt existing serial code
  - limited by RAM to be placed into a single machine

- OpenMP – [www.openmp.org](http://www.openmp.org)

~~• most commonly used standard to parallelize~~

~~• you(!) have to add extra commands to the code~~

• supported by gcc since v4.2

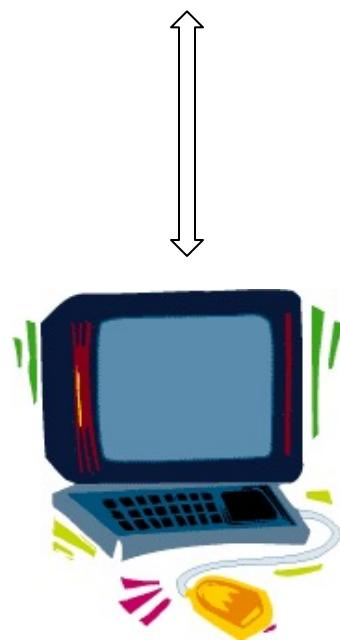
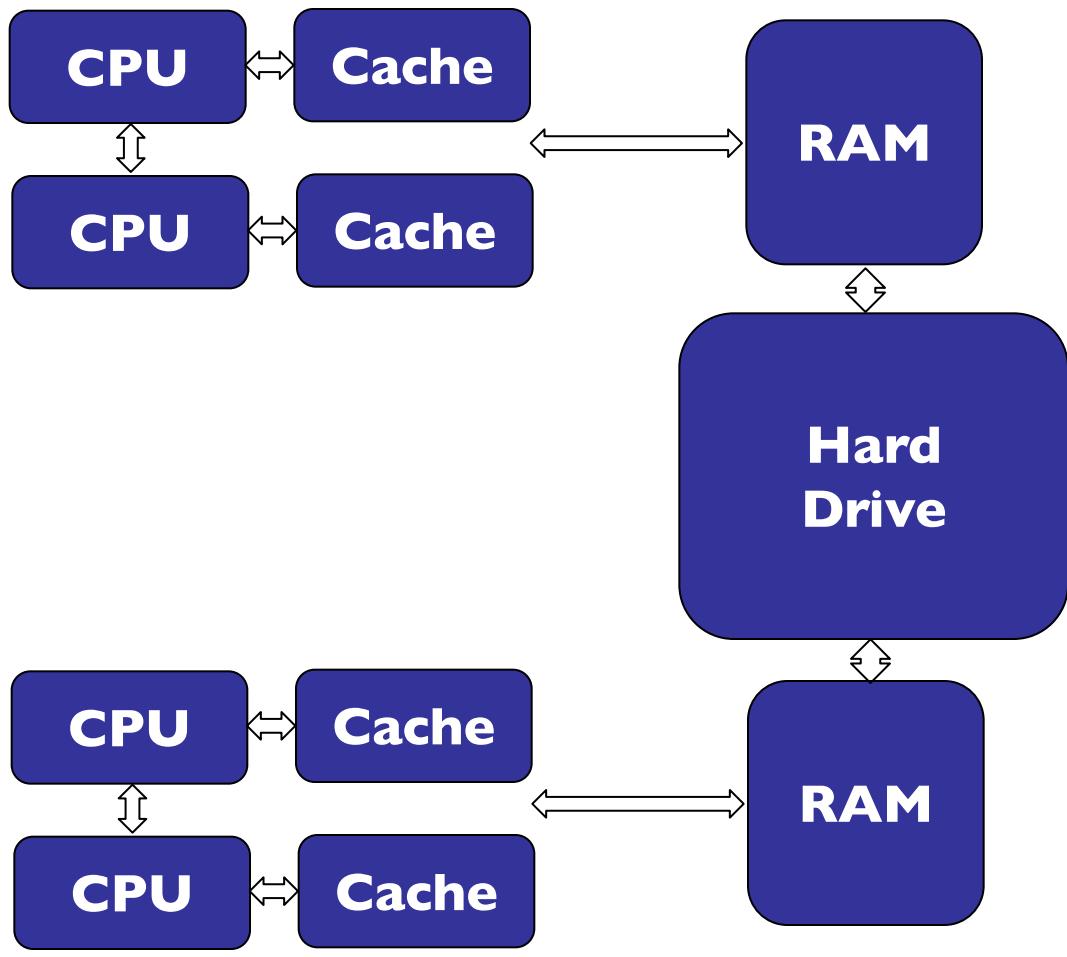
## Computer Architectures

- parallel machine      Distributed memory architecture

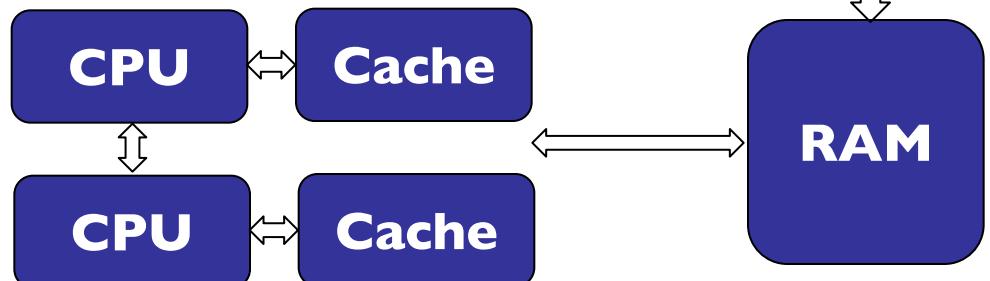
Check the number of sockets (shared memory) in your laptop with  
\$ sudo lscpu



=

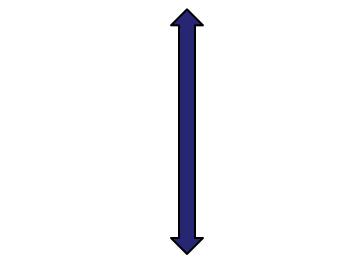
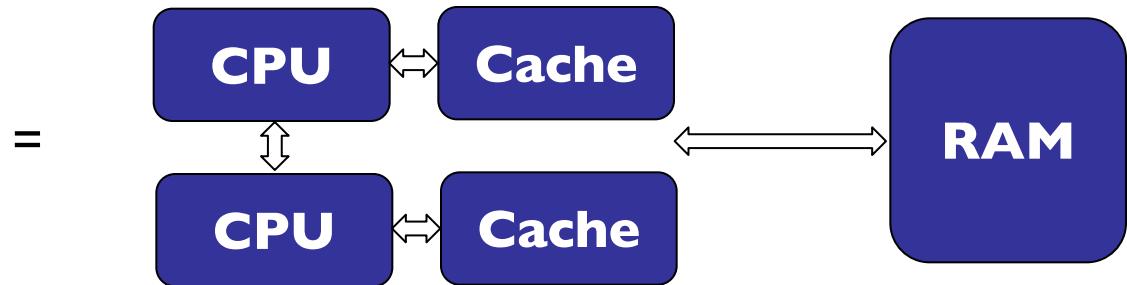


=



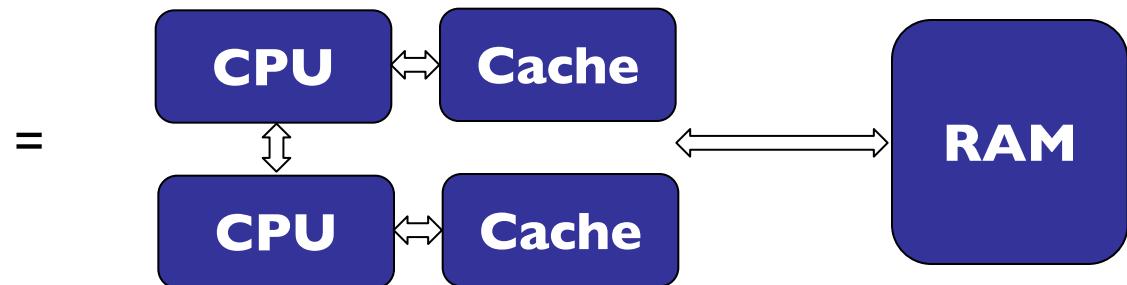
## Computer Architectures

- parallel machine



- performance highly sensitive to interconnect, e.g.

- FireWire 50 MB/s
- Gigabit 125 MB/s
- Myrinet 250 MB/s
- Infiniband 1000 MB/s
- ...

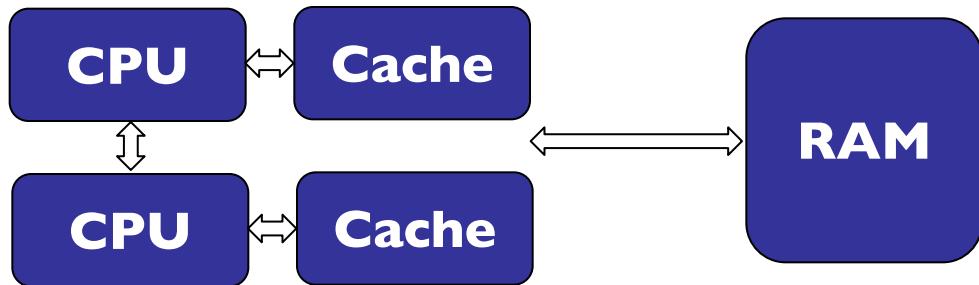


## Computer Architectures

- parallel machine



=



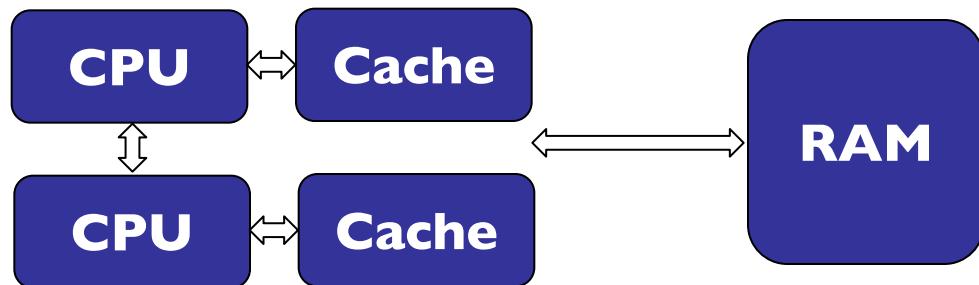
- distributed memory architecture:

- existing code difficult to adapt
- easy to built (cluster of PC's)
- speed-up limited by inter-computer communication

↓



=

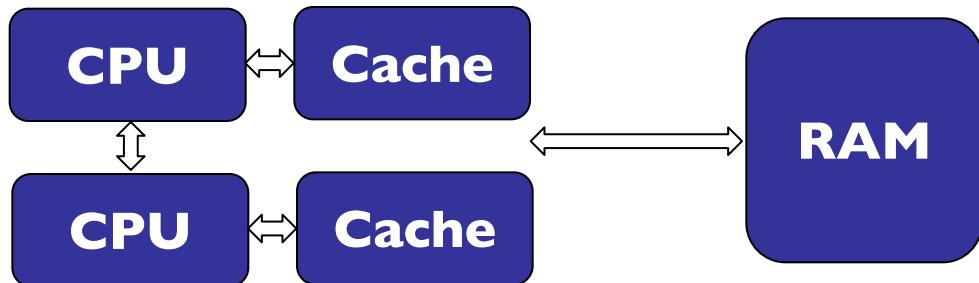


## Computer Architectures

- parallel machine



=

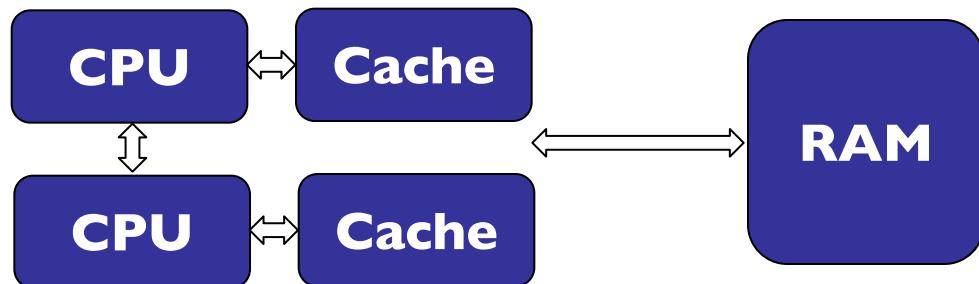


- distributed memory architecture:

- existing code difficult to adapt
- easy to built (cluster of PC's)
- speed-up limited by inter-computer communication



=



- MPI – Message Passing Interface

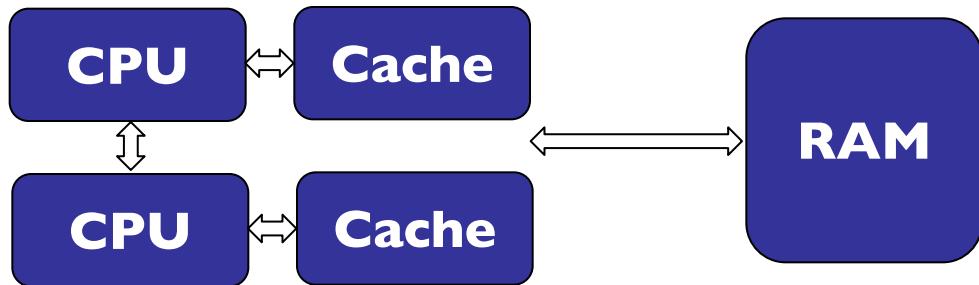
- “standard” library for work dispersal on distributed memory architectures
- e.g., [www.open-mpi.org](http://www.open-mpi.org)

## Computer Architectures

- parallel machine



=

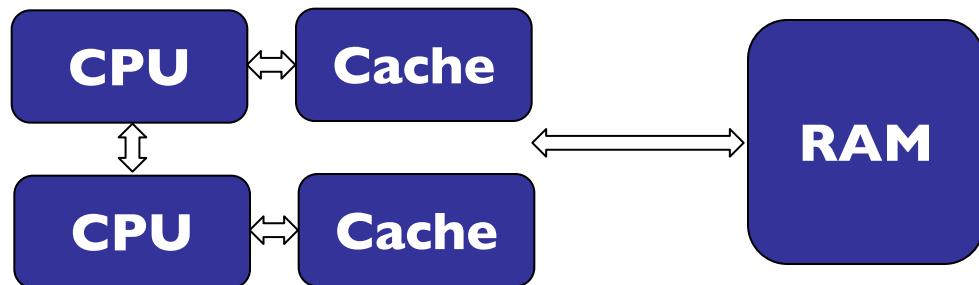


- distributed memory architecture:

- existing code difficult to port
- easier to移植 (移植)
- speed-up limited by inter-computer communication



=



- MPI – Message Passing Interface

**you(!) have to substantially restructure your code**

on distributed memory architectures  
• e.g., [www.open-mpi.org](http://www.open-mpi.org)

- architectures
- **real machines**
- computing concepts
- parallel programming

## Computer Architectures

- parallel machine in reality = multi-level hybrid machines

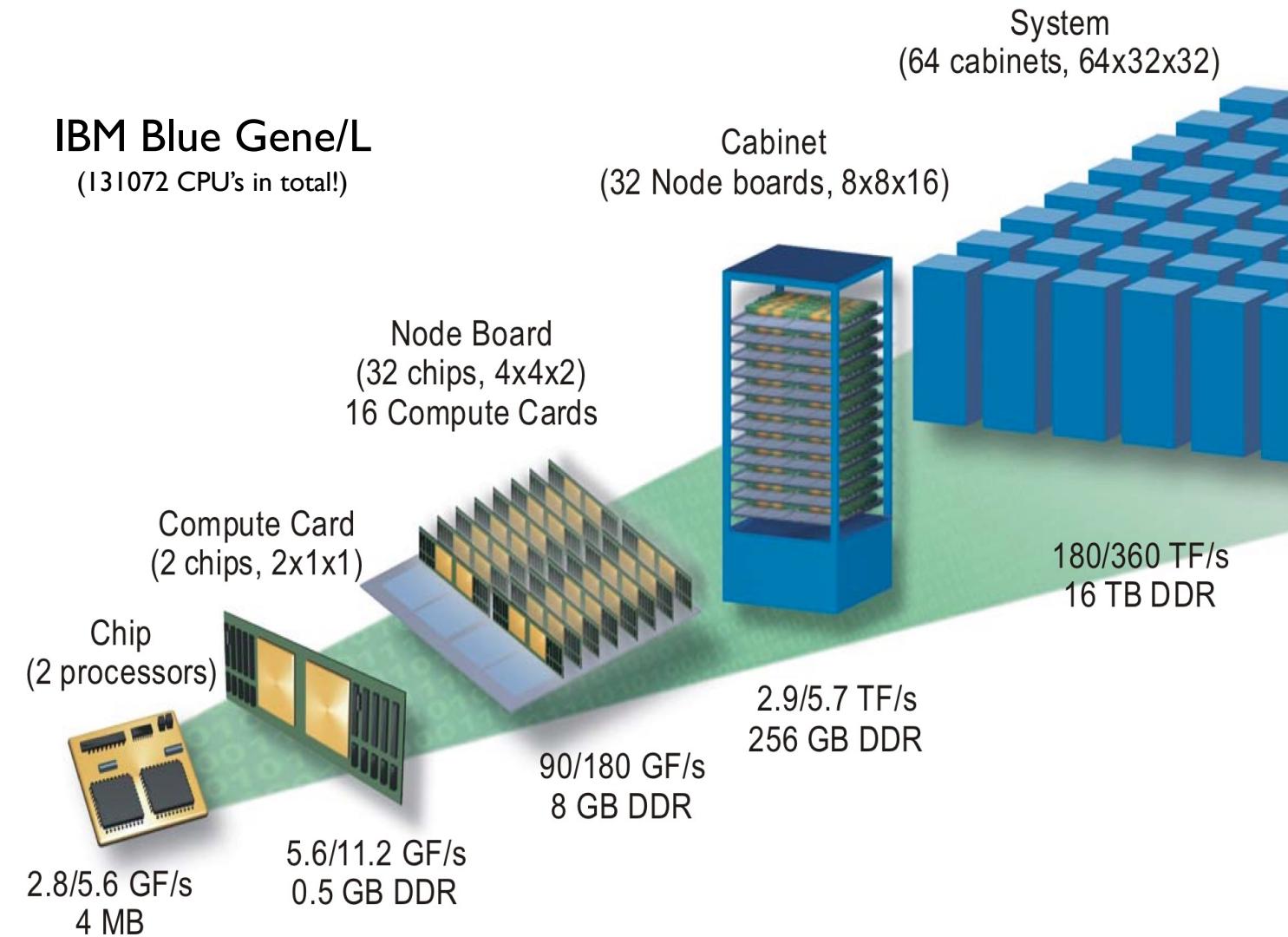
### IBM Blue Gene/L

(131072 CPU's in total!)



## Computer Architectures

- parallel machine in reality = multi-level hybrid machines



## Computer Architectures

- [www.top500.org](http://www.top500.org)

Home | TOP500 Supercomputing Sites

http://www.top500.org/

Dict-EN Dict-ES Astro UAM MAD Banking Lifestyle Mac Mail Misc Movies Newspaper Music Shopping AK TV

Home | TOP500 Supercomputing...

**TOP 500®**  
SUPERCOMPUTER SITES

**isc events**  
cloud computing

Mannheim, Germany  
September 26-27, 2011

PROJECT LISTS STATISTICS RESOURCES NEWS CONTACT SUBMISSIONS LINKS HOME

**TOP 10 Systems - 06/2011**

|   |                                                                                            |
|---|--------------------------------------------------------------------------------------------|
| 1 | K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect                                       |
| 2 | Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C                          |
| 3 | Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz                                                |
| 4 | Nebulae - Dawning TC3600 Blade, Intel X5650, Nvidia Tesla C2050 GPU                        |
| 5 | TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows               |
| 6 | Cielo - Cray XE6 8-core 2.4 GHz                                                            |
| 7 | Pleiades - SGI Altix ICE 8200EX/8400EX, Xeon HT QC 3.0/Xeon 5570/5670 2.93 Ghz, Infiniband |

Thu, 2011-06-16 19:24

► Japan Reclaims Top Ranking on Latest TOP500 List of World's Supercomputers



HAMBURG, Germany—A Japanese supercomputer capable of performing more than 8 quadrillion calculations per second (petaflop/s) is the new number one system in the world, putting Japan back in the top spot for the first time since the Earth Simulator was dethroned in November 2004, according to the latest edition of the TOP500 List of the world's top supercomputers. The system, called the K Computer, is at the RIKEN Advanced Institute for Computational Science (AICS) in Kobe.

» Read more

**SUPERMICRO®**  
We Keep IT Green™

APPRO  
Delivering Improved, Reliability, Availability, Serviceability (RAS)

Deep Computing?

hp

## Computer Architectures

- [www.top500.org](http://www.top500.org)

### **#136 in 06/2011**

Name: Magerit  
Vendor: IBM  
#CPU's: 3920  
performance: 100 Tflops/sec

### **#170 in 06/2011**

Name: MareNostrum  
Vendor: IBM  
#CPU's: 10240  
performance: 60 Tflops/sec

## Computer Architectures

- [www.top500.org](http://www.top500.org)

**#136 in 06/2011**

Name: Magerit  
Vendor: IBM  
#CPU's: 3920  
performance: 100 Tflops/sec

**#170 in 06/2011**

Name: MareNostrum  
Vendor: IBM  
#CPU's: 10240  
performance: 60 Tflops/sec



## Computer Architectures

- [www.top500.org](http://www.top500.org)

### **#136 in 06/2011**

Name: Magerit  
Vendor: IBM  
#CPU's: 3920  
performance: 100 Tflops/sec

*interconnect:* Infiniband, up to 1500 Gbits/sec

### **#170 in 06/2011**

Name: MareNostrum  
Vendor: IBM  
#CPU's: 10240  
performance: 60 Tflops/sec

*interconnect:* Myrinet, ca. 2Gbit/sec

- architectures
- real machines
- **computing concepts**
- parallel programming

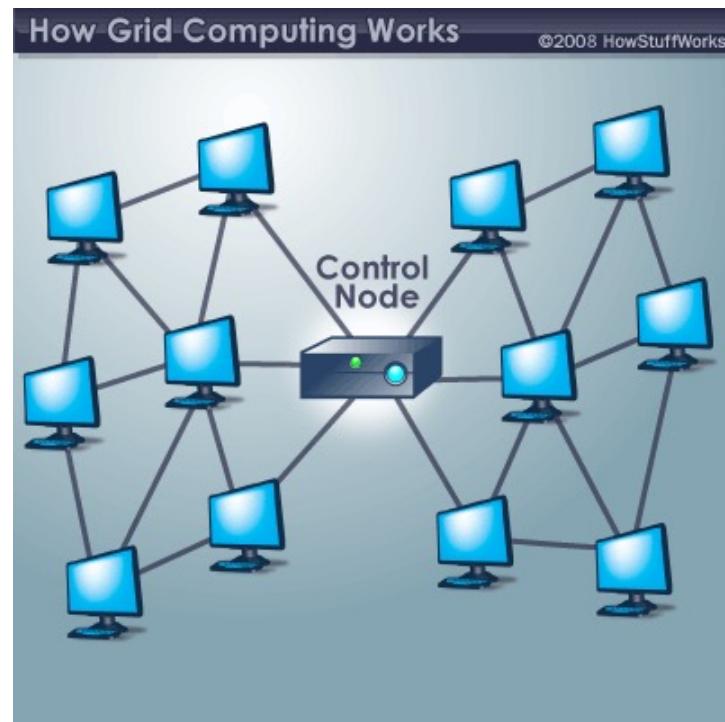
## Computer Architectures

- GRID computing / Cloud computing?

## Computer Architectures

### ■ GRID computing:

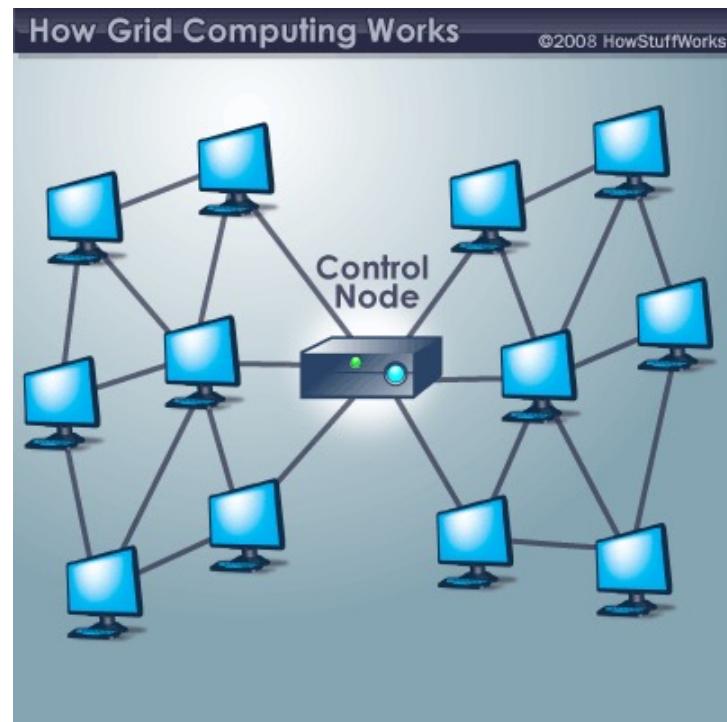
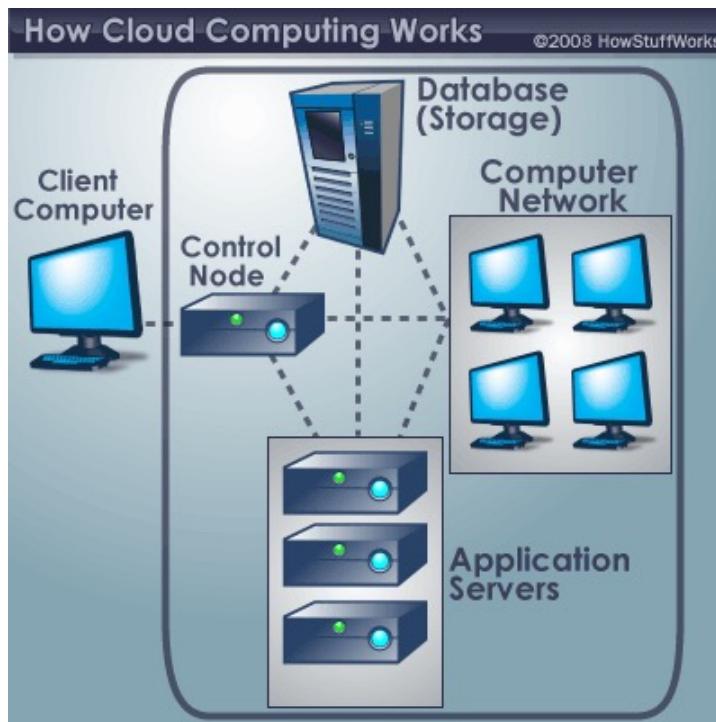
- distributed computing where resources are linked together to solve a single problem



## Computer Architectures

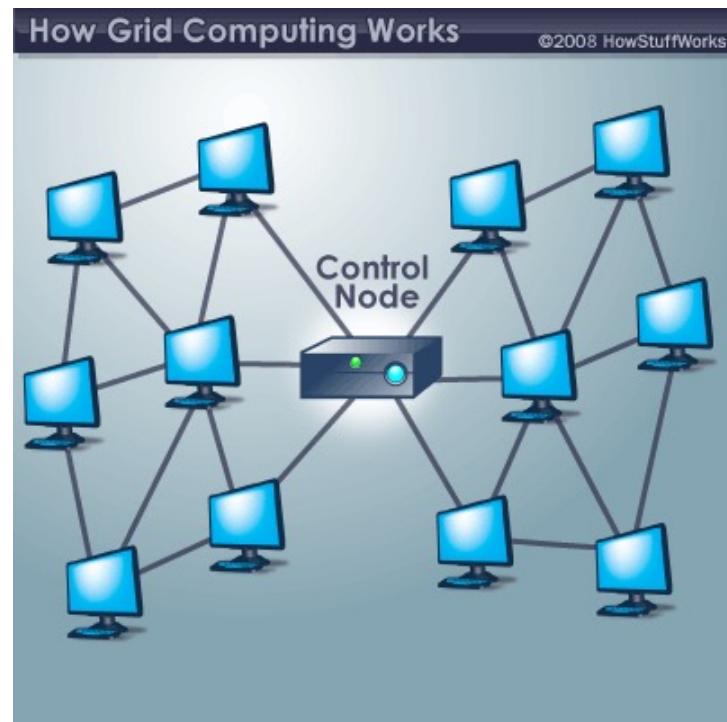
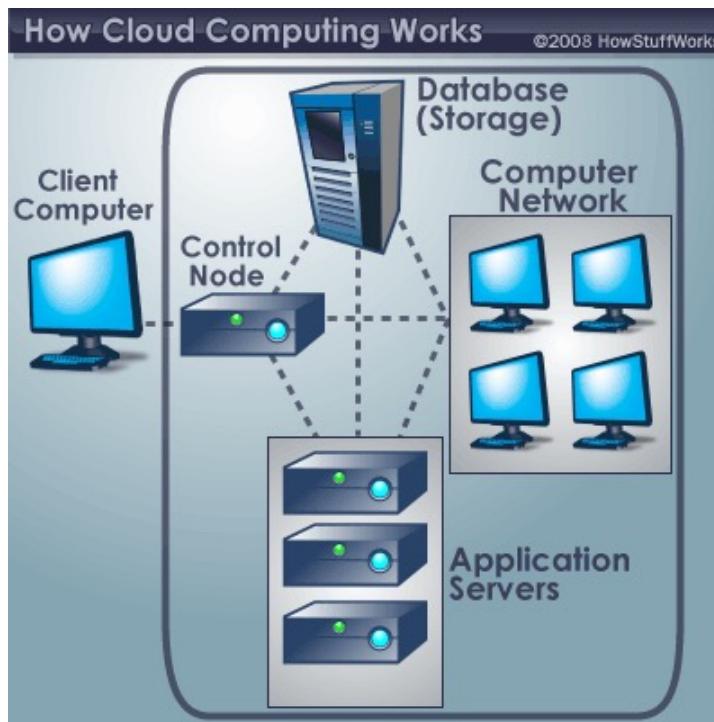
- **GRID computing:**
  - distributed computing where resources are linked together to solve a single problem

- **Cloud computing:**
  - use remote resources for your (personal) needs (music storage, email correspondence, ...)



## Computer Architectures

- GRID computing: **⇒ actually running simulations**
  - distributed computing where resources are linked together to solve a single problem
- Cloud computing: **⇒ access to results via databases**
  - use remote resources for your (personal) needs (music storage, email correspondence, ...)



## Computer Architectures

- GRID computing:       $\Rightarrow$  **actually running simulations**
  - distributed computing where resources are linked together to solve a single problem
- Cloud computing:       $\Rightarrow$  **access to results via databases**
  - use remote resources for your (personal) needs (music storage, email correspondence, ...)

*first GRID computing?*

## Computer Architectures

- GRID computing:  $\Rightarrow$  **actually running simulations**
  - distributed computing where resources are linked together to solve a single problem
- Cloud computing:  $\Rightarrow$  **access to results via databases**
  - use remote resources for your (personal) needs (music storage, email correspondence, ...)

The screenshot shows a web browser window displaying the official website for SETI@home. The URL in the address bar is <http://setiathome.berkeley.edu/>. The page features a dark blue header with the SETI@HOME logo and navigation links for various categories like Astro, UAM, MAD, Banking, Lifestyle, Mac, Mail, Misc, Movies, Newspaper, Music, Shopping, AK, and TV. Below the header, there's a main content area with sections for "What is SETI@home?", "PARTICIPATE", "ABOUT", "COMMUNITY", "YOUR ACCOUNT", and "STATISTICS". A large image of a satellite dish is on the left. At the bottom, there are "Get started" steps and news items.

**What is SETI@home?**  
SETI@home is a scientific experiment that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You can participate by running a free program that downloads and analyzes radio telescope data.

**PARTICIPATE**

- Download
- Get help
- Tell a friend
- Donate
- Porting & optimization
- ... more

**ABOUT**

- About SETI@home
- About Astropulse
- Science newsletters
- Technical news
- Server status
- Science status
- Sponsors
- ... more

**COMMUNITY**

- Message boards
- Questions & answers
- Profiles
- User search
- Teams
- Web sites & IRC
- Pictures & music

**YOUR ACCOUNT**

- Your account
- Preferences
- Certificate

**STATISTICS**

- Top participants
- Top computers
- Top teams
- Top GPU models

**Site search:**

**Languages:**

**Get started**

- 1 [Read our rules and policies](#)
- 2 [Download, install and run the BOINC software used by SETI@home. When prompted, enter the URL: <http://setiathome.berkeley.edu>](#)

Have questions or need help? Contact a volunteer using [BOINC online help](#).

Special instructions:

- [For SETI@home Classic participants](#)
- [For users of command-line and pre-5.0 clients](#)

**News**

**Another way to support SETI@home**  
In addition to crunching, you can provide some support to SETI@home by using [GoodSearch](#) and [GoodShop](#). These search engines redirect a half their advertising to revenues to charity. Just be sure to choose "University of California - SETI@home" as your charity of choice. 12 Sep 2011 | 20:38:21 UTC · [Comment](#)

**more data on the way**  
In an attempt to push some older unanalyzed files through the pipeline we encountered data that could not be successfully preprocessed or split. This has resulted in work distribution going to near zero. We are currently transferring newer files from off site storage and soon will be receiving a disk shipment from Arecibo. Once these data are on hand work distribution will pick up. 8 Sep 2011 | 16:56:49 UTC · [Comment](#)

**storage service is back up**  
We have migrated storage service to thumper. Now it's a matter of transferring raw data to thumper, preprocessing it, and splitting it. This will all take some time. The Overland server is up and its raid is operational (thanks Overland!). Diagnosis of this unit is proceeding. 28 May 2011 | 14:30:01 UTC · [Comment](#)

**POWERED BY**  Keep your computer busy when SETI@home has no work - [participate in other BOINC-based](#)

### Computer Architectures

- GRID computing:  $\Rightarrow$  **actually running simulations**
  - distributed computing where resources are linked together to solve a single problem
- Cloud computing:  $\Rightarrow$  **access to results via databases**
  - use remote resources for your (personal) needs (music storage, email correspondence, ...)

The screenshot shows the SETI@home website homepage. The top navigation bar includes links for Astro, UAM, MAD, Banking, Lifestyle, Mac, Mail, Misc, Movies, Newspaper, Music, Shopping, AK, and TV. A sidebar on the left provides links for 'Get started' (with steps 1 and 2), 'Special instructions' (for Classic participants and command-line users), and a note about other BOINC-based projects. The main content area features a large banner with the text: '• over 3.5 mio. computers participating' and '• (virtual) machine reaches ~0.8 Pflops !!!'. Below this, another section highlights 'no signs of ET yet ...' and concludes with '...but distributed computing works well!'. The right side of the page contains a 'STATISTICS' panel showing participant counts and a 'NEWS' section with recent posts.

SETI@home

http://setiathome.berkeley.edu/

SETI@home

What is SETI@home?  
SETI@home is a scientific experiment that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You can participate by running a free program that downloads and analyzes radio telescope data.

Click Here for More Information

**PARTICIPATE**

- Get help
- Porting & optimization
- ... more

**ABOUT**

- About Astropulse
- Server status
- Science status
- Sponsors
- ... more

**COMMUNITY**

- Questions
- Computers
- Help
- Teams
- Web sites & IRC
- Pictures & music

**YOUR ACCOUNT**

- Preferences
- Certificate

**STATISTICS**

- Participants
- Top computers
- Top teams
- Top GPU models

Site search:  Languages:

**Get started**

- 1 Read our [FAQ](#).
- 2 Download [SETI@home](#), install and run the BOINC software used by SETI@home. When prompted, enter the URL: <http://setiathome.berkeley.edu>

Have questions or need help? [Ask online](#).

**Special instructions:**

- For [SETI@home Classic participants](#)
- For [users of command-line and pre-5.0 clients](#).

Once these data are on hand work distribution will pick up. 8 Sep 2011 | 16:56:49 UTC · [Comment](#)

**storage service is back up**  
We have migrated storage service to thumper. Now it's a matter of transferring raw data to thumper, preprocessing it, and splitting it. This will all take some time. The Overland server is up and its raid is operational (thanks Overland!). Diagnosis of this unit is proceeding. 28 May 2011 | 14:30:01 UTC · [Comment](#)

POWERED BY Keep your computer busy when SETI@home has no work - [participate in other BOINC-based](#)

## Computer Architectures

- GRID computing: **⇒ actually running simulations**
  - distributed computing where resources are linked together to solve a single problem

- Cl

The screenshot shows the official website for SETI@home. At the top, there's a navigation bar with links for Dict-EN, Dict-ES, Astro, UAM, MAD, Mac, Mail, Banking, Misc, Movies, Newspaper, Music, Shopping, Anja, AK, and 300sheet. Below the navigation is a search bar with 'setiathome.berkeley.edu' and a user account section for 'SETI@home'. The main content area features a large banner with the text 'SETI@home' overlaid on a background image of a colorful nebula.

**What is SETI@home?**

SETI@home is a scientific experiment, based at UC Berkeley, that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You can participate by running a free program that downloads and analyzes radio telescope data.

**Join SETI@home**

Already joined? [Log in](#).

**User of the Day**

Xcure 20%

We are now on self quarantine and a pandemic is

POWERED BY

Keep your computer busy when SETI@home has no work - [participate in other BOINC-based](#)

**News**

**SETI@home and COVID-19**

SETI@home will stop distributing tasks soon, but we encourage you to continue donate computing power to science research - in particular, research on the COVID-19 virus. The best way to do this is to join [Science United](#) and check the "Biology and Medicine" box.

23 Mar 2020, 21:33:54 UTC · Discuss

**New SETI Perspectives: "How did life begin on Earth and elsewhere?"**

Richard Lawn has posted a new SETI Perspective entitled [How did life begin on Earth and elsewhere?](#).

19 Mar 2020, 22:49:24 UTC · Discuss

28 May 2011 | 14:30:01 UTC · [Comment](#)

## Computer Architectures

- GRID computing:       $\Rightarrow$  **actually running simulations**
  - distributed computing where resources are linked together to solve a single problem
- Cloud computing:       $\Rightarrow$  **access to results via databases**
  - use remote resources for your (personal) needs (music storage, email correspondence, ...)

there actually exists a GRID network with  $21 \times 10^6$  Pflops\* !!!!!!

## Computer Architectures

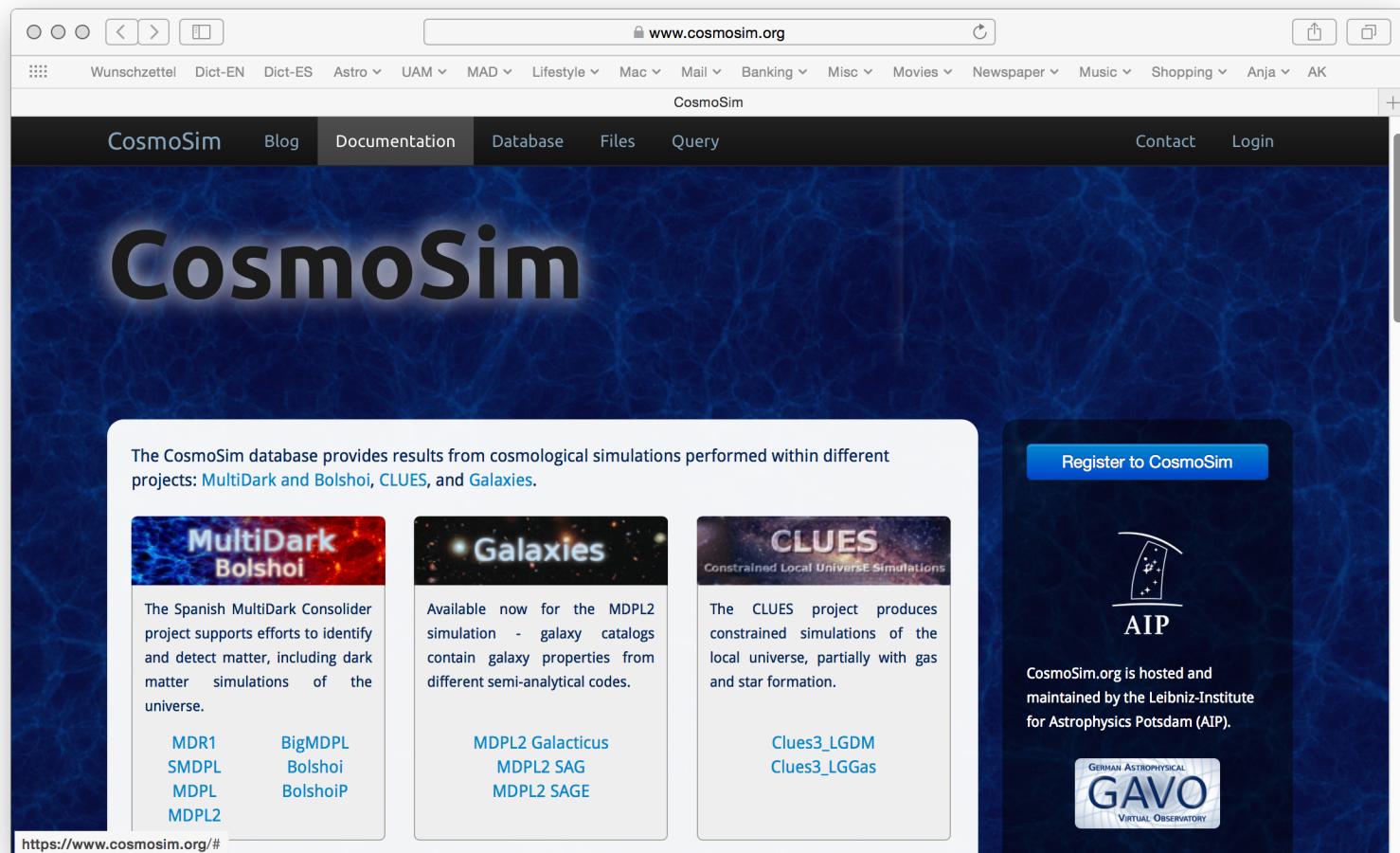
- GRID computing:       $\Rightarrow$  **actually running simulations**
  - distributed computing where resources are linked together to solve a single problem
- Cloud computing:       $\Rightarrow$  **access to results via databases**
  - use remote resources for your (personal) needs (music storage, email correspondence, ...)

there actually exists a GRID network with  $21 \times 10^6$  Pflops:

Bitcoin Network

## Computer Architectures

- GRID computing: **⇒ actually running simulations**
  - distributed computing where resources are linked together to solve a single problem
- Cloud computing: **⇒ access to results via databases**
  - use remote resources for your (personal) needs (music storage, email correspondence, ...)



- architectures
- real machines
- computing concepts
- **parallel programming**

## Computer Architectures

- how to program such machines?

## Computer Architectures

- how to program such machines?

**your algorithm must be parallel,**

then it's only a matter of using parallel libraries to distribute the work...

- how to program such machines?

**your algorithm must be parallel,**

then it's only a matter of using parallel libraries to distribute the **work...**

### **data parallelisation:**

all CPUs execute the same code, but have different parts of the data

### **task parallelisation**

all CPUs have the same data, but execute different calculations

## Computer Architectures

- how to program such machines?

**your algorithm must be parallel,**

then it's only a matter of using parallel libraries to distribute the **work...**

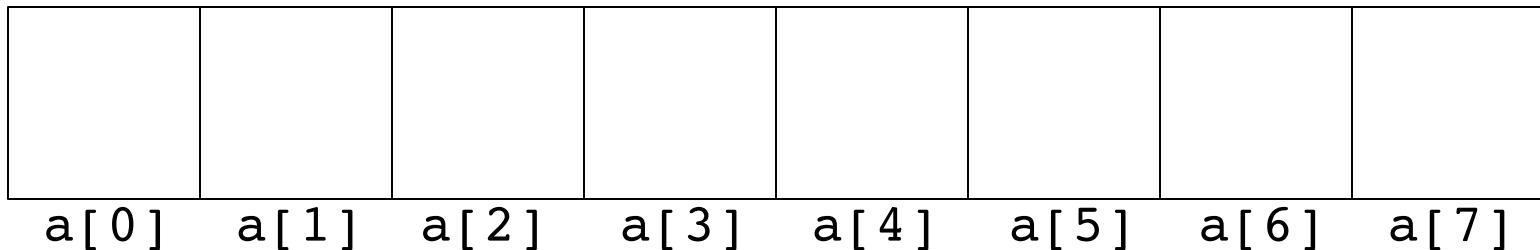
### **data parallelisation:**

all CPUs execute the same code, but have different parts of the data

## Computer Architectures

- how to program such machines?

1D array



## Computer Architectures

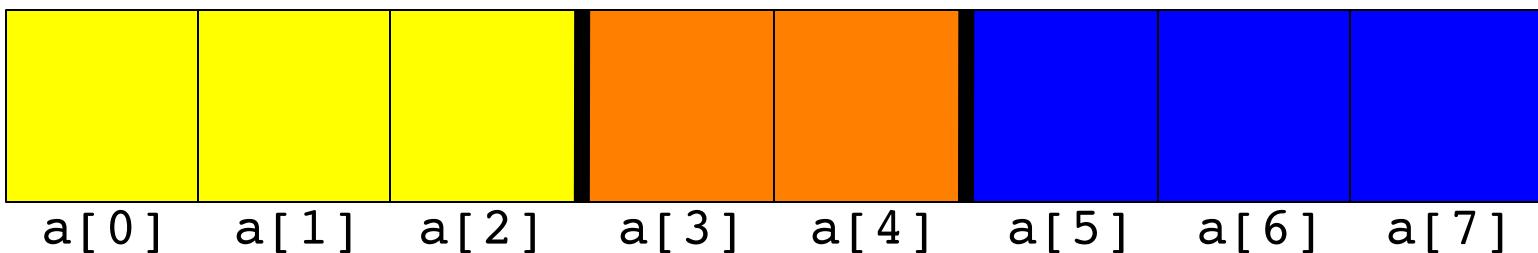
- how to program such machines?

1D array

CPU #1

CPU #2

CPU #3



## Computer Architectures

- how to program such machines?

1D array

### serial algorithm

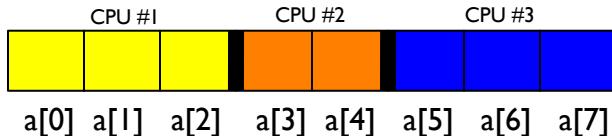
```
a[0] = STARTVALUE;

for(i=1; i<N; i++) {

 a[i] = function(a[i-1]);

}
```

Make a (serial) program that doubles the numbers.



## Computer Architectures

- how to program such machines?

1D array

### serial algorithm

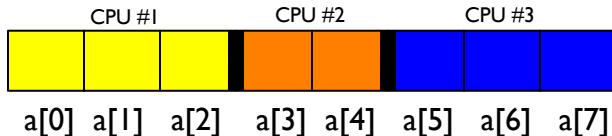
```
a[0] = STARTVALUE;

for(i=1; i<N; i++) {

 a[i] = function(a[i-1]);

}
```

$\Rightarrow$  not parallelizable as  $a[ i ]$  depends on all previous  $a[ ]$ 's!



## Computer Architectures

- how to program such machines?

1D array

### serial algorithm

```
a[0] = STARTVALUE;

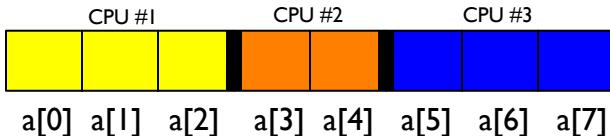
for(i=1; i<N; i++) {

 a[i] = function(a[i-1]);

}
```

$\Rightarrow$  not parallelizable as  $a[ i ]$  depends on all previous  $a[ ]$ 's!

general remark:  
recursion is elegant yet not parallelizable...



How can we make the tasks independent?

## Computer Architectures

- how to program such machines?

1D array

### serial algorithm

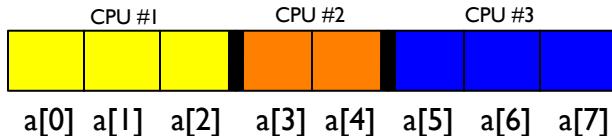
```
a[0] = STARTVALUE;

for(i=1; i<N; i++) {

 a[i] = function(a[i-1]);

}
```

### parallel algorithm



## Computer Architectures

- how to program such machines?

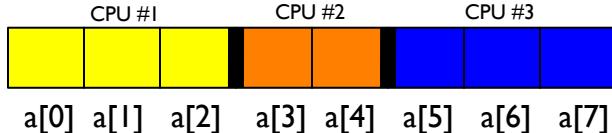
1D array

**serial algorithm**

```
a[0] = STARTVALUE;
for(i=1; i<N; i++) {
 a[i] = function(a[i-1]);
}
```

**parallel algorithm**

each CPU runs the same code,  
but on a different part of the problem...



## Computer Architectures

- how to program such machines?

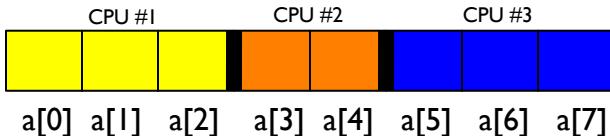
1D array

### serial algorithm

```
a[0] = STARTVALUE;
for(i=1; i<N; i++) {
 a[i] = function(a[i-1]);
}
```

### parallel algorithm

*example:*  
shared memory architecture  
(i.e. all CPU's can access the same memory)



## Computer Architectures

- how to program such machines?

1D array

**serial algorithm**

```
a[0] = STARTVALUE;

for(i=1; i<N; i++) {

 a[i] = function(a[i-1]);
}
```

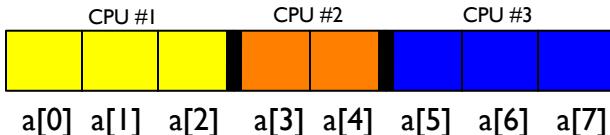
**parallel algorithm**

```
a[0] = STARTVALUE;

for(i=1; i<N; i++) {

 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
}
```

Code this up doubling values (still in series).



- how to program such machines?

1D array

### serial algorithm

```
a[0] = STARTVALUE;

for(i=1; i<N; i++) {

 a[i] = function(a[i-1]);
}
```

### parallel algorithm

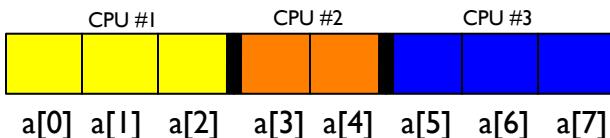
```
a[0] = STARTVALUE;

for(i=1; i<N; i++) {

 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
}
```

*the i-loop can now be parallelized as  
all a[ i ] are calculated independently*

Filling the array are independent tasks, even if this  
program is slower.



## Computer Architectures

- how to program such machines?

1D array

**serial algorithm**

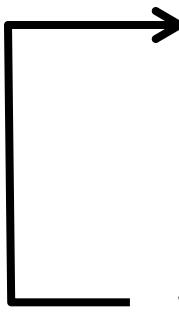
```
a[0] = STARTVALUE;
for(i=1; i<N; i++) {
 a[i] = function(a[i-1]);
}
```

**parallel algorithm**

Make your last program parallel:

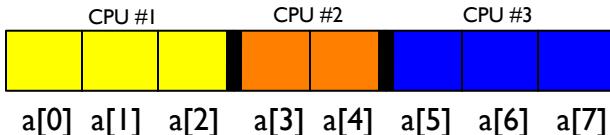
```
#pragma omp parallel
{
 int ID = omp_get_thread_num();
 # pragma omp for
 for(i=1; i<N; i++) {

 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
 printf("Thread %d: a[%d]=%.1f \n",ID,i,a[i]);
 }
}
```



*we eliminated the recursion/dependence  
by expanding it explicitly.*

*(by introducing yet another recursion, but c'est la vie...)*



Compilation:  
gcc -o p -fopenmp par\_b.c

What time do you get using clock()?

## Computer Architectures

- how to program such machines?

1D array

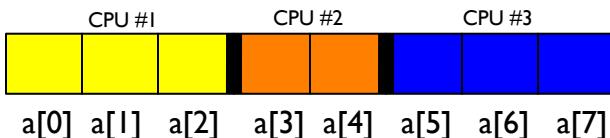
**serial algorithm**

```
a[0] = STARTVALUE;
for(i=1; i<N; i++) {
 a[i] = function(a[i-1]);
}
```

**parallel algorithm**

```
double start = omp_get_wtime();
a[0] = STARTVALUE;
#pragma omp parallel for private(i,j,b) shared(a)
for(i=1; i<N; i++) {
 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
}

double time = 1000.*omp_get_wtime() - start); //ms
printf("\n Time=%E ms \n",time);
```



## Computer Architectures

- how to program such machines?

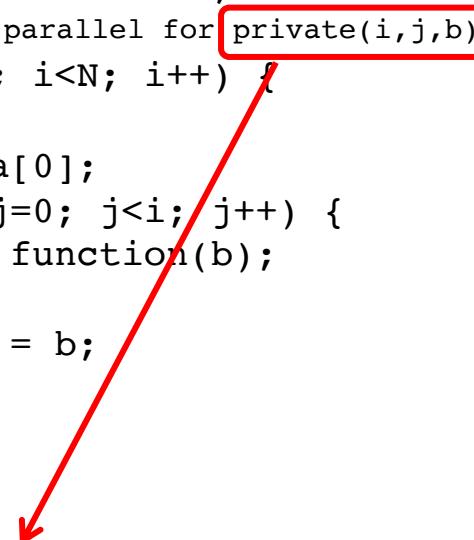
1D array

serial algorithm

```
a[0] = STARTVALUE;
for(i=1; i<N; i++) {
 a[i] = function(a[i-1]);
}
```

parallel algorithm

```
a[0] = STARTVALUE;
#pragma omp parallel for private(i,j,b) shared(a)
for(i=1; i<N; i++) {
 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
}
```

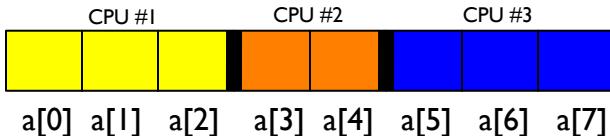


**each CPU gets its own private copy of these variables**

Observe what the program has assumed by default and what happens if you modify here variables from private to shared.

What do you expect for N?

What happens if you set b as shared?



## Computer Architectures

- how to program such machines?

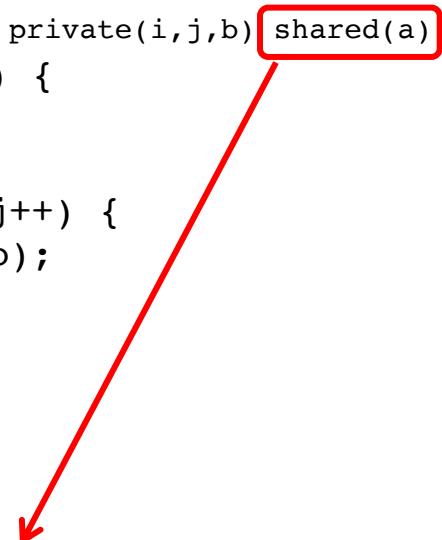
1D array

serial algorithm

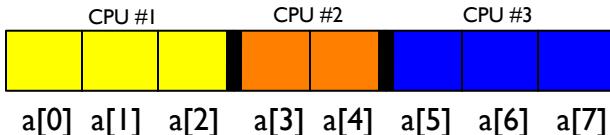
```
a[0] = STARTVALUE;
for(i=1; i<N; i++) {
 a[i] = function(a[i-1]);
}
```

parallel algorithm

```
a[0] = STARTVALUE;
#pragma omp parallel for private(i,j,b) shared(a)
for(i=1; i<N; i++) {
 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
}
```



**these variables remain where they are in RAM and can be accessed by each CPU**



## Computer Architectures

- how to program such machines?

1D array

**serial algorithm**

```
a[0] = STARTVALUE;

for(i=1; i<N; i++) {

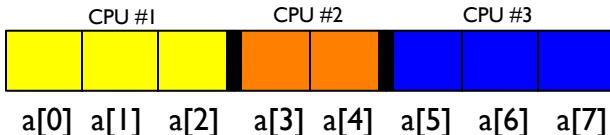
 a[i] = function(a[i-1]);
}
```

**parallel algorithm**

```
a[0] = STARTVALUE;
#pragma omp parallel for private(i,j,b) shared(a)
for(i=1; i<N; i++) {

 b = a[0];
 for(j=0; j<1; j++) {
 b = function(b);
 }
 a[i] = b;
}
```

N: shared or private?



## Computer Architectures

- how to program such machines?

1D array

**serial algorithm**

```
a[0] = STARTVALUE;

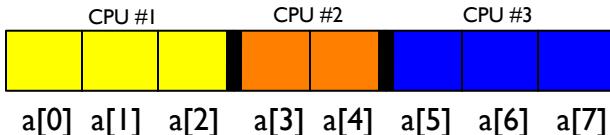
for(i=1; i<N; i++) {

 a[i] = function(a[i-1]);
}
```

**parallel algorithm**

```
a[0] = STARTVALUE;
#pragma omp parallel for private(i,j,b) shared(a,N)
 for(i=1; i<N; i++) {

 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
 }
```

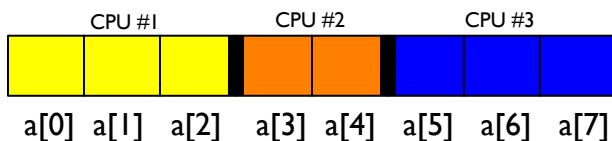


## Computer Architectures

- how to program such machines?

1D array

```
STARTVALUE = 1.35;
N = 8;
a = (float *) calloc(N, sizeof(float));
a[0] = STARTVALUE;
```



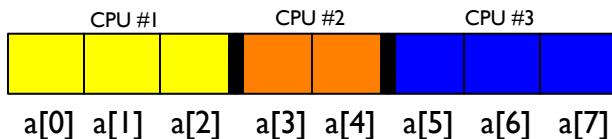
## Computer Architectures

- how to program such machines?

1D array

```
STARTVALUE = 1.35;
N = 8;
a = (float *) calloc(N, sizeof(float));
a[0] = STARTVALUE;
```

N=8, a[0]=1.35, a[1:7]=0



## Computer Architectures

- how to program such machines?

1D array

```

STARTVALUE = 1.35;
N = 8;
a = (float *) calloc(N, sizeof(float));
a[0] = STARTVALUE;

```

N=8, a[0]=1.35, a[1:7]=0

#pragma omp parallel for private(i,j,b) shared(a,N)

```

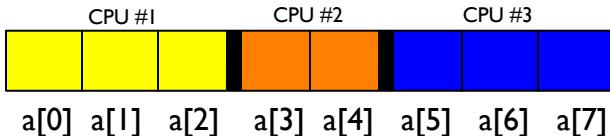
double calc_b(int i, double sv)
{
 double b;
 int j;

 b = sv;

 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}

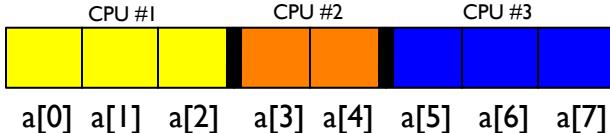
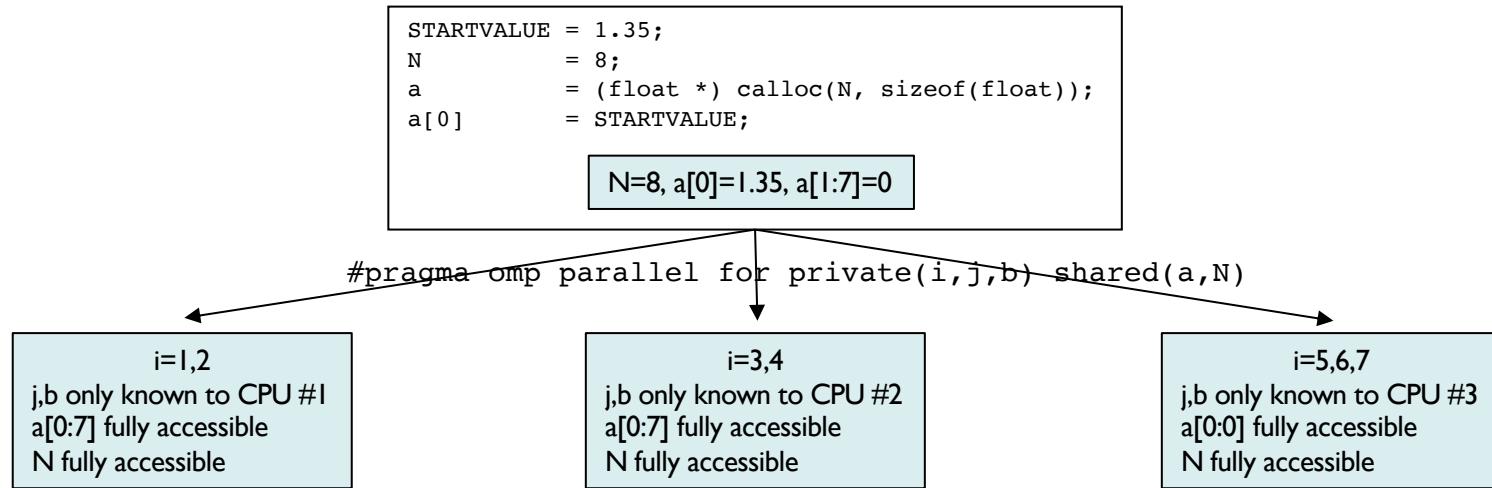
```



## Computer Architectures

- how to program such machines?

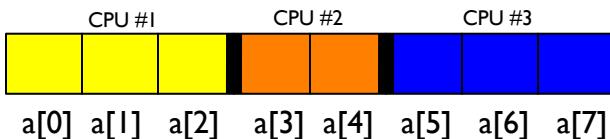
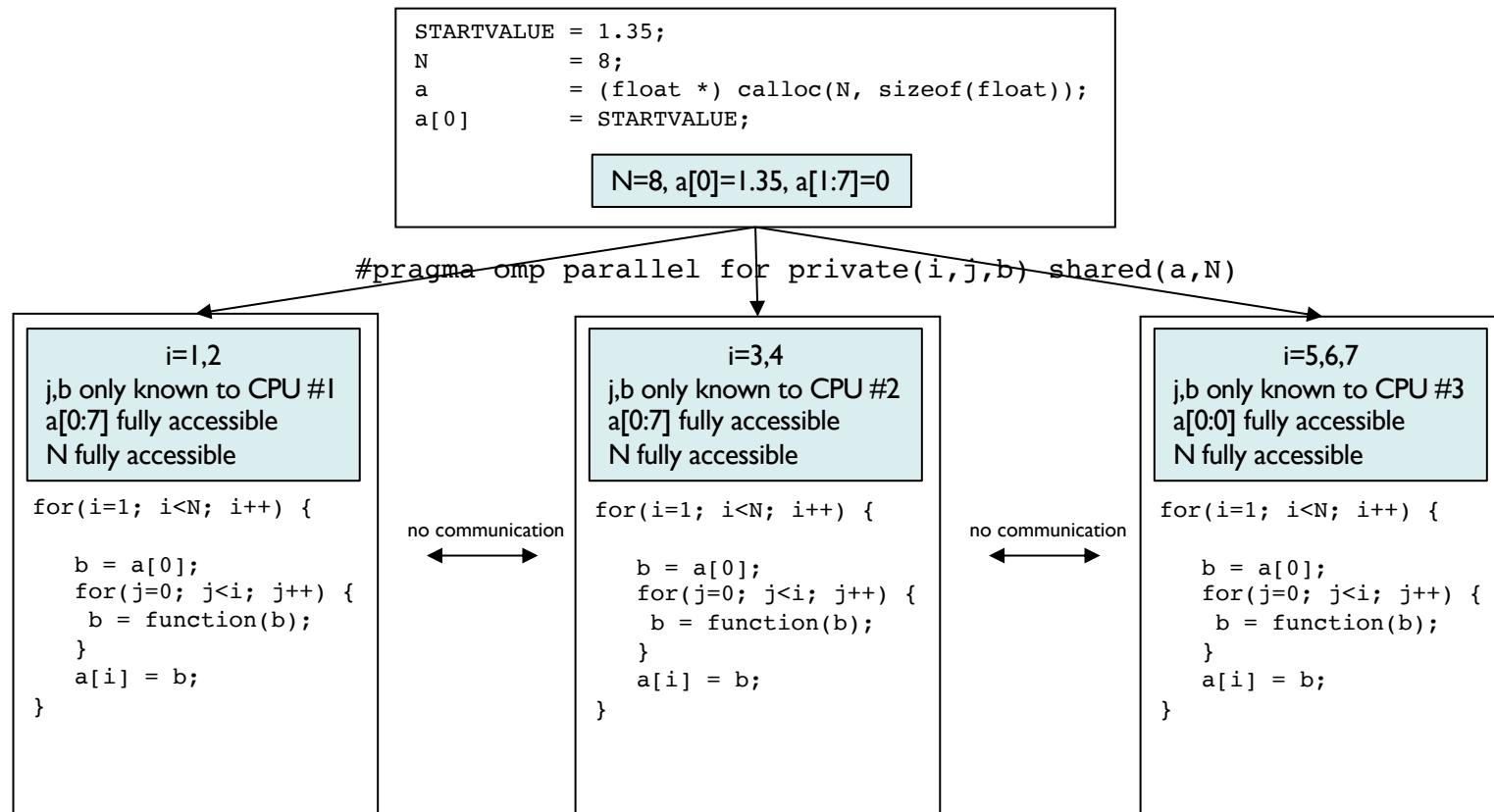
1D array



## Computer Architectures

- how to program such machines?

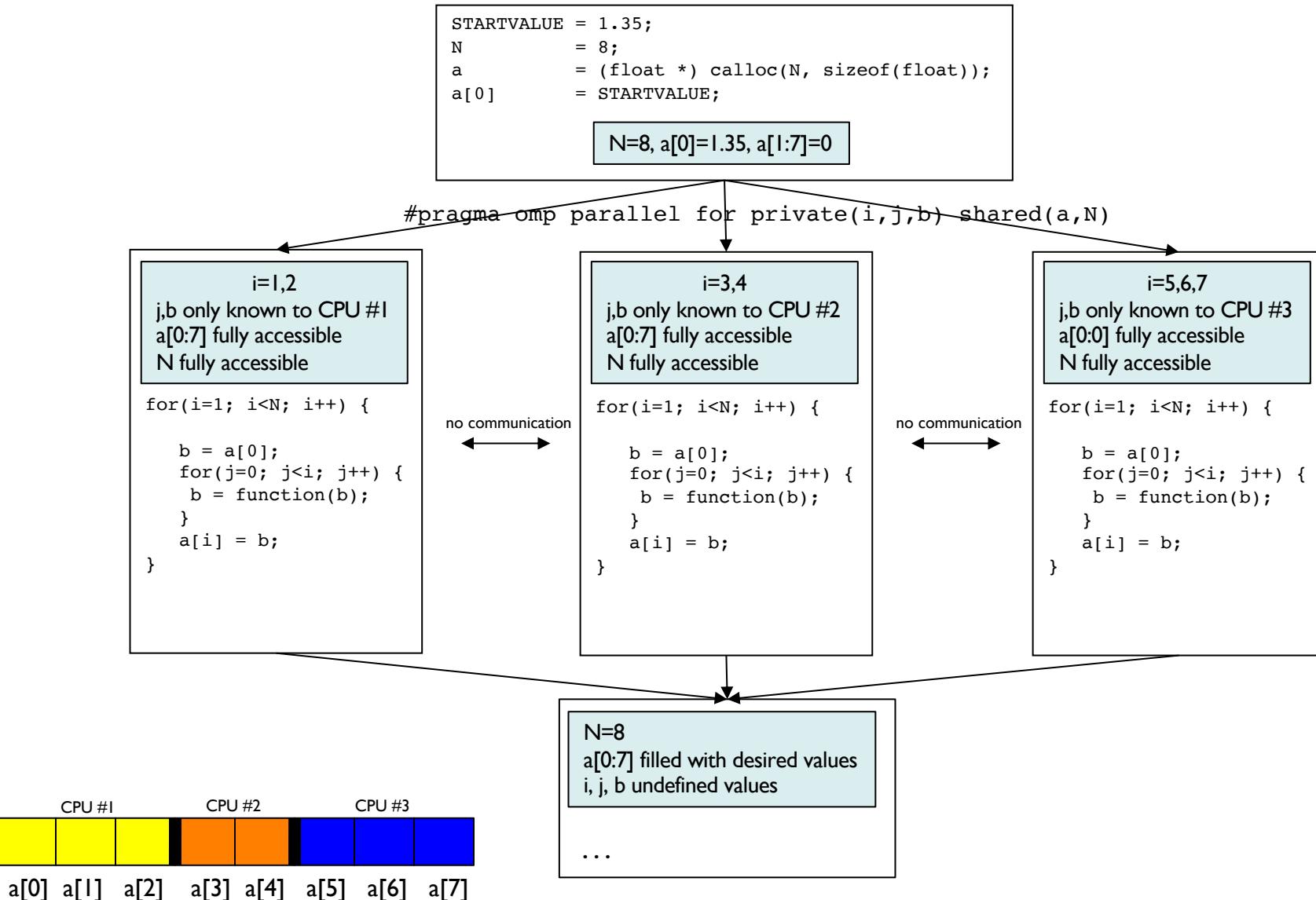
1D array



## Computer Architectures

- how to program such machines?

1D array



## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
 {
 start parallel environment
 (can be started everywhere in code...)
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
```

start parallel environment  
(can be started everywhere in code...)

parallel environment  
only for next for-loop

## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
```

start parallel environment  
(can be started everywhere in code...)

parallel environment  
only for next for-loop

each thread stores its own  
local copy of these variables

## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
```

start parallel environment  
(can be started everywhere in code...)

parallel environment  
only for next for-loop

each thread stores its own  
local copy of these variables

variables accessible  
by each thread

## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
```

start parallel environment  
(can be started everywhere in code...)

parallel environment  
only for next for-loop

each thread stores its own  
local copy of these variables

variables accessible  
by each thread

**Note:**

- if you only read the value of a variable, it can be ‘shared’
- if you write into a variable, think carefully about its status!

## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
```

start parallel environment  
(can be started everywhere in code...)

parallel environment  
only for next for-loop

each thread stores its own  
local copy of these variables

variables accessible  
by each thread

```
#pragma omp parallel for private(i,j,b) shared(a,N)
for(i=1; i<N; i++) {
```

```
 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
}
```

**Note:**

- the loop-counter has to be private
- if you only **read** the value of a variable, it can be ‘shared’
- if you **write** into a variable, think carefully about its status

## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
```

start parallel environment  
(can be started everywhere in code...)

parallel environment  
only for next for-loop

each thread stores its own  
local copy of these variables

variables accessible  
by each thread

```
#pragma omp parallel for private(i,j,b) shared(a,N)
for(i=1; i<N; i++) {
```

```
 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
 a[i] = b;
}
```

j?

**Note:**

- the loop-counter has to be private
- if you only **read** the value of a variable, it can be ‘shared’
- if you **write** into a variable, think carefully about its status

## Computer Architectures

- how to program such machines (OpenMP standard):

```
#pragma omp parallel for private() shared()
```

start parallel environment  
(can be started everywhere in code...)

parallel environment  
only for next for-loop

each thread stores its own  
local copy of these variables

variables accessible  
by each thread

```
#pragma omp parallel for private(i,j,b) shared(a,N)
for(i=1; i<N; i++) {
```

```
 b = a[0];
 for(j=0; j<i; j++) {
 b = f(b);
 }
 a[i] = b;
}
```

Note:

**there is a far more elegant way to write this code!**

- the variable `a` has to be `private`
- if you only **read** the value of a variable, it can be 'shared'
- if you **write** into a variable, think carefully about its status

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i,j,b) shared(a,N)
for(i=1; i<N; i++) {

 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }

 a[i] = b;
}
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i,j,b) shared(a,N)
for(i=1; i<N; i++) {

 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }

 a[i] = b;
}
```

*put this into a function!*

Try it out!

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i,j,b) shared(a,N)
for(i=1; i<N; i++) {
```

```
 b = a[0];
 for(j=0; j<i; j++) {
 b = function(b);
 }
```

```
 a[i] = b;
}
```

*put this into a function:*

```
double calc_b(int i, double sv)
{
 double b;
 int j;

 b = sv;

 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i,j,b) shared(a,N)
for(i=1; i<N; i++) {

 a[i] = calc_b(i, a[0]);
}

double calc_b(int i, double sv)
{
 double b;
 int j;
 b = sv;

 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i,j,b) shared(a,N)
for(i=1; i<N; i++) {

 a[i] = calc_b(i, a[0]);
}

double calc_b(int i, double sv)
{
 double b;
 int j;
 b = sv;

 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i) shared(a,N)
for(i=1; i<N; i++) {
 a[i] = calc_b(i, a[0]);
}

double calc_b(int i, double sv)
{
 double b;
 int j;

 b = sv;

 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i) shared(a,N)
for(i=1; i<N; i++) {
 a[i] = calc_b(i, a[0]);
}

double calc_b(int i, double sv)
{
 double b;
 int j;

 b = sv;
 ? #pragma omp parallel for...
 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i) shared(a,N)
for(i=1; i<N; i++) {

 a[i] = calc_b(i, a[0]);
}
```

2 reasons for not parallelizing this for-loop...

```
double calc_b(int i, double sv)
{
 double b;
 int j;

 b = sv;
 ?#pragma omp parallel for...
 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i) shared(a,N)
for(i=1; i<N; i++) {

 a[i] = calc_b(i, a[0]);
}
```

2 reasons for not parallelizing this for-loop:

- it is a recursion
- we already parallelized outside of calc\_b()

```
double calc_b(int i, double sv)
{
 double b;
 int j;

 b = sv;
 ? #pragma omp parallel for...
 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i) shared(a,N)
for(i=1; i<N; i++) {

 a[i] = calc_b(i, a[0]);
}
```

```
double calc_b(int i, double sv)
{
 double b;
 int j;

 b = sv;

 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

### **general advise:**

- make your code modular, i.e. use functions
- modular code is easier to parallelize

## Computer Architectures

- how to program such machines (OpenMP standard):

```
a[0] = STARTVALUE;

#pragma omp parallel for private(i) shared(a,N)
for(i=1; i<N; i++) {

 a[i] = calc_b(i, a[0]);
}
```

```
double calc_b(int i, double sv)
{
 double b;
 int j;

 b = sv;

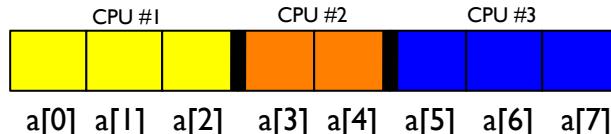
 for(j=0; j<i; j++) {
 b = function(b);
 }

 return(b);
}
```

### **general advise:**

- make your code modular, i.e. use functions
- modular code is easier to parallelize

...but which CPU gets what  $i$  values?



## Computer Architectures

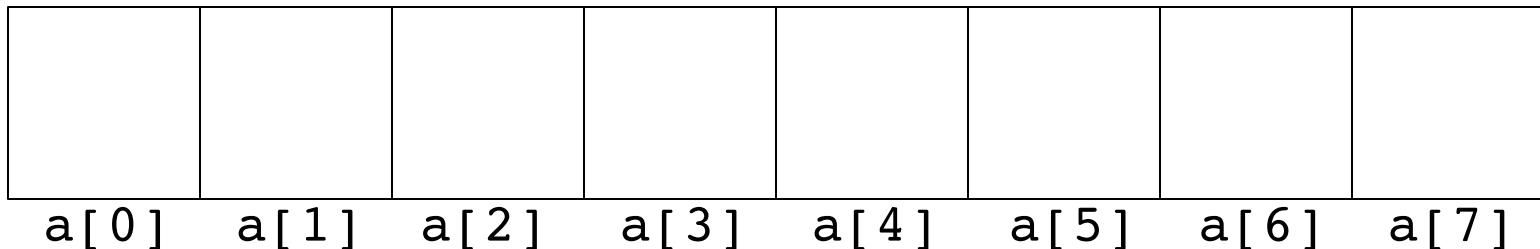
- how to program such machines?

1D array

CPU #1

CPU #2

CPU #3

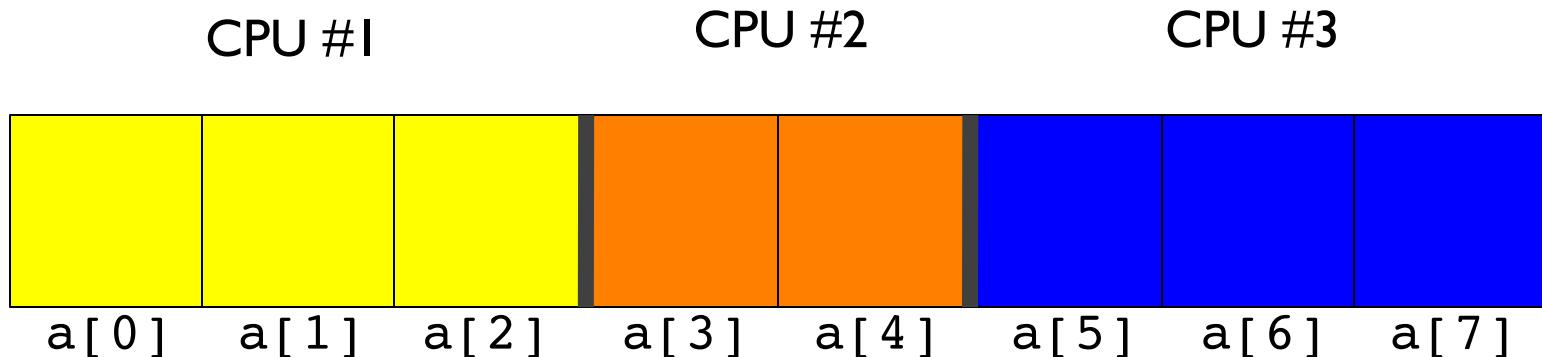


but how to divide the domain?

## Computer Architectures

- how to program such machines?

1D array

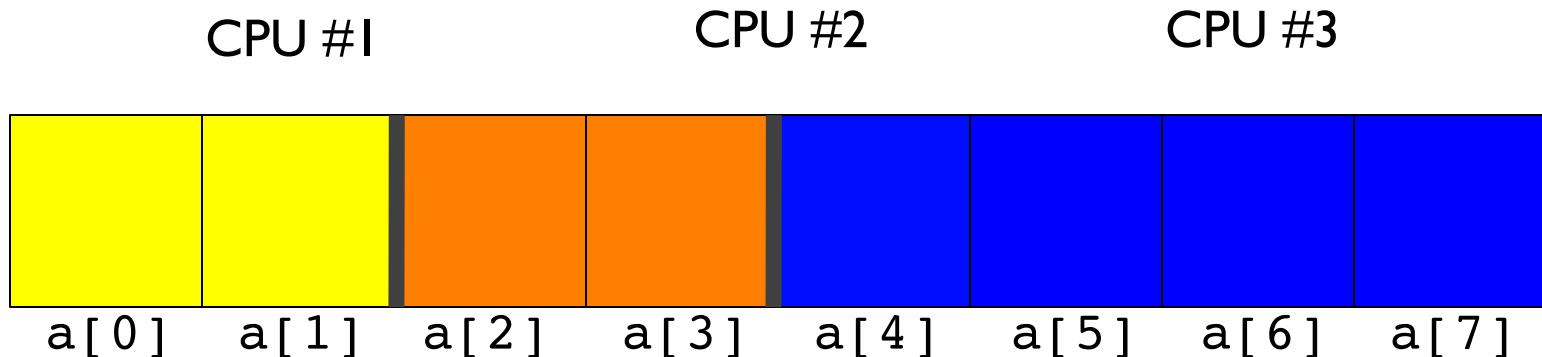


but how to divide the domain?

## Computer Architectures

- how to program such machines?

1D array

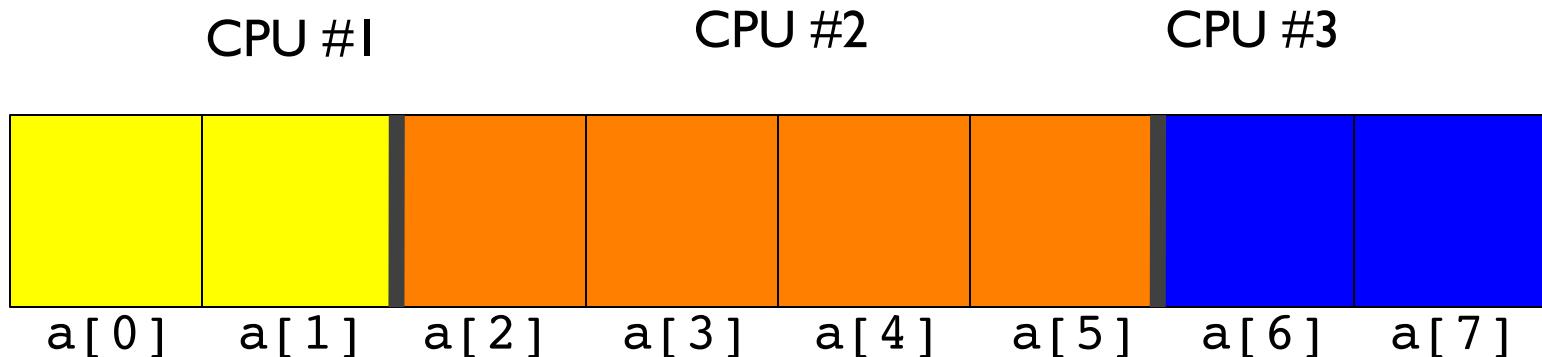


but how to divide the domain?

## Computer Architectures

- how to program such machines?

1D array

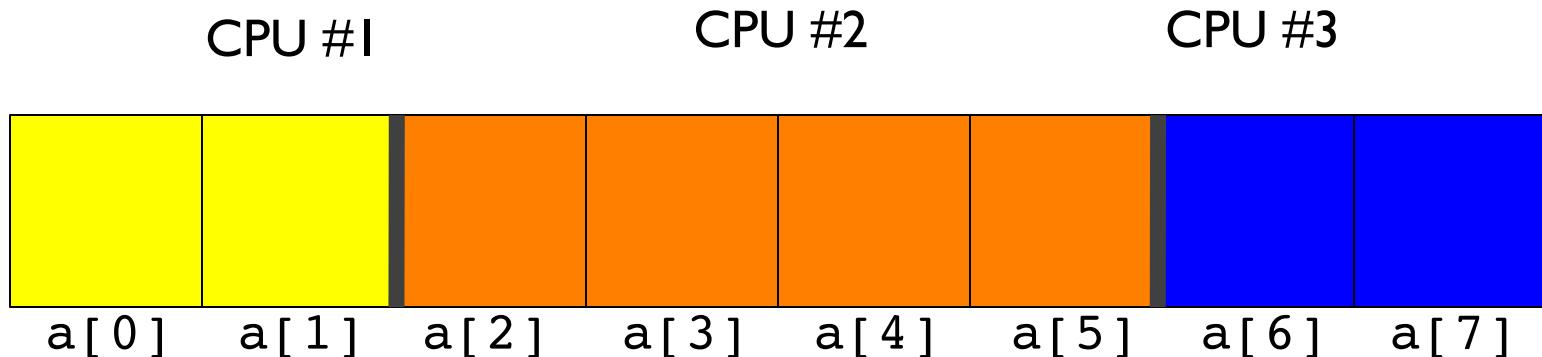


but how to divide the domain?

## Computer Architectures

- how to program such machines?

1D array



but how to divide the domain:  
distribute the **work** evenly!

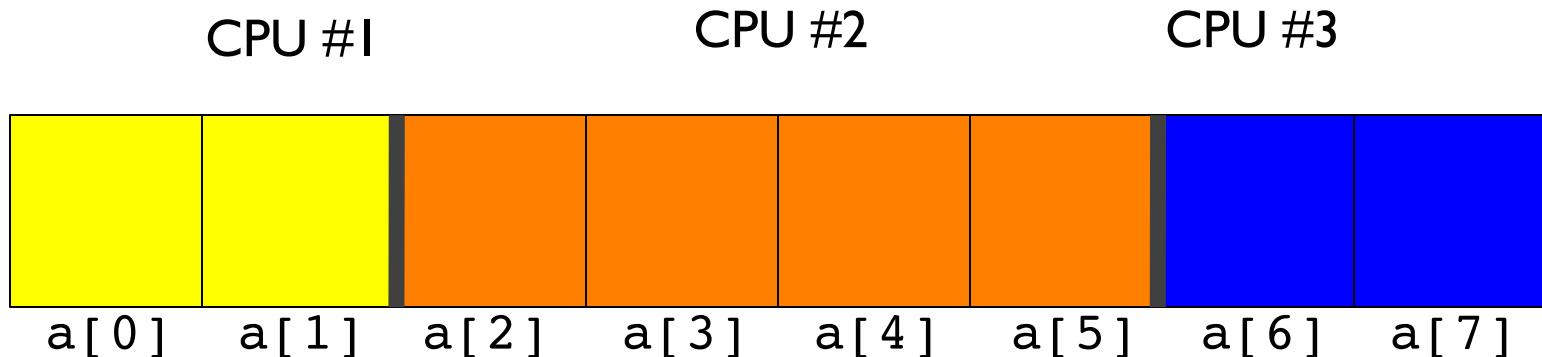
Set the number of threads to 3 in your last program, but using:  
`omp_set_num_threads(3);`

How are the iterations distributed by default in your laptop?

## Computer Architectures

- how to program such machines?

1D array



but how to divide the domain:  
distribute the **work** evenly!

```
b = a[0];
for(j=0; j<i; j++) {
 b = function(b);
}
=> CPU's dealing with higher i 's have more work to do!
```

## Computer Architectures

- how to program such machines?

1D array

### **OpenMP work distribution:**

The schedule clause affects how loop iterations are mapped onto threads

**schedule(dynamic):** loop index = 0-Nthreads-1 ↗ Nthreads ↗ Nthreads+1 ↗ etc.

Each thread grabs a "chunk" from the iteration until it's finished. Most work at runtime.

**schedule (static):** evenly divide loop index amongst Nthreads

Pre-determined and predictable by the programmer.

# pragma omp for schedule(static,chunk)

### **usage:**

```
#pragma omp parallel for private(...) shared(...) schedule(...)
```

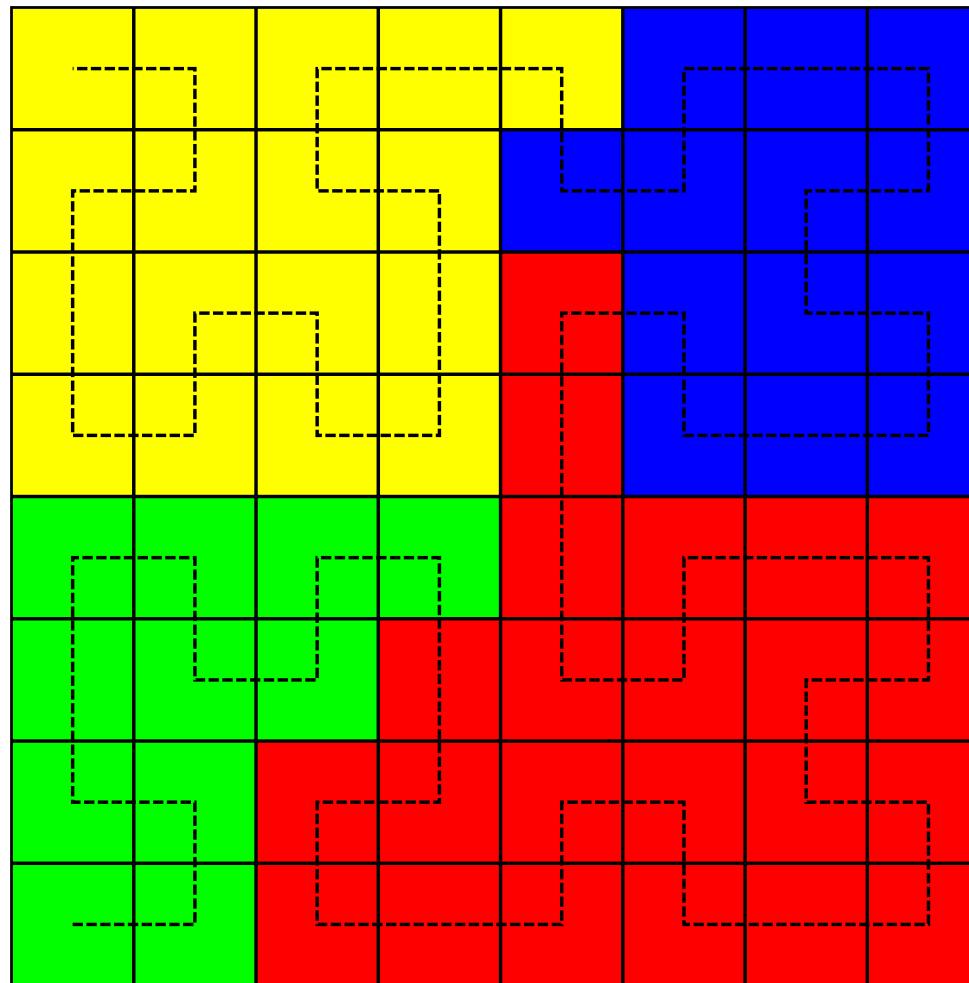
```
b = a[0];
for(j=0; j<i; j++) {
 b = function(b); => CPU's dealing with higher i's have more work to do!
}
```

Explore how your program is performed if you set schedule to either dynamic or static with different chunks.

## Computer Architectures

- how to program such machines?

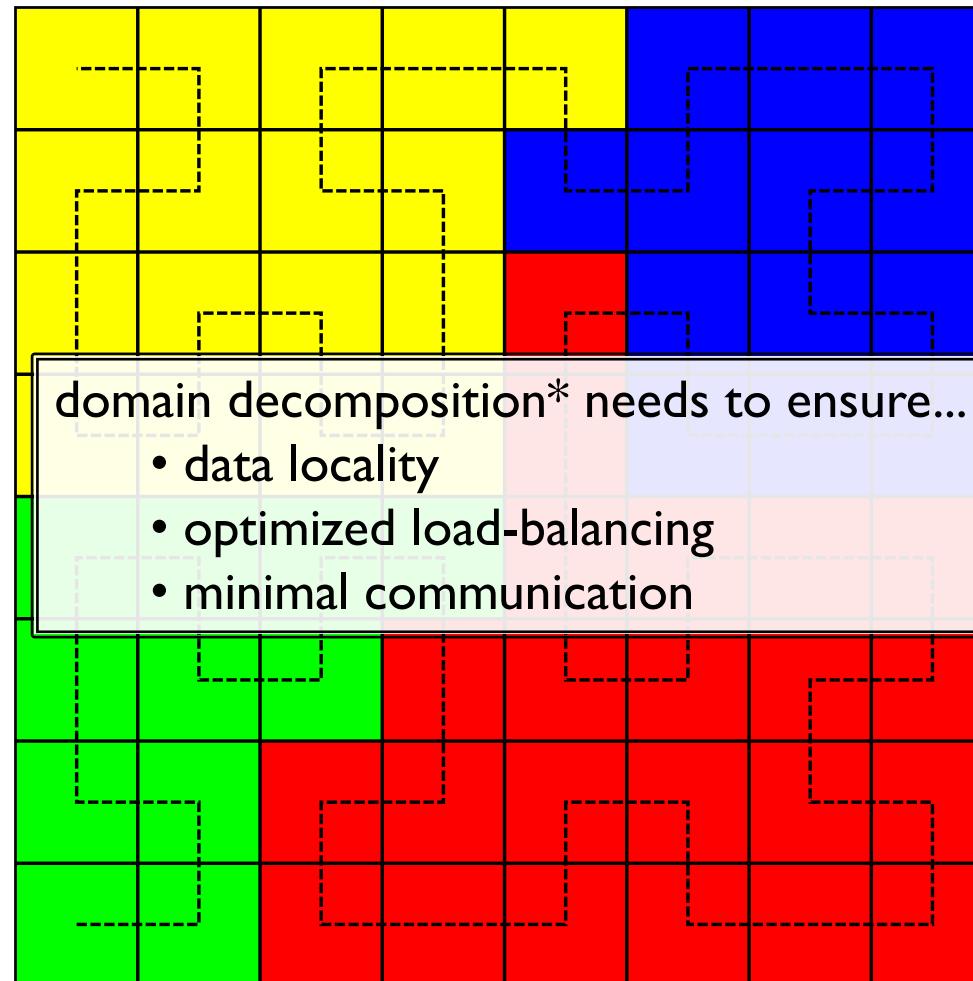
2D array



## Computer Architectures

- how to program such machines?

2D array

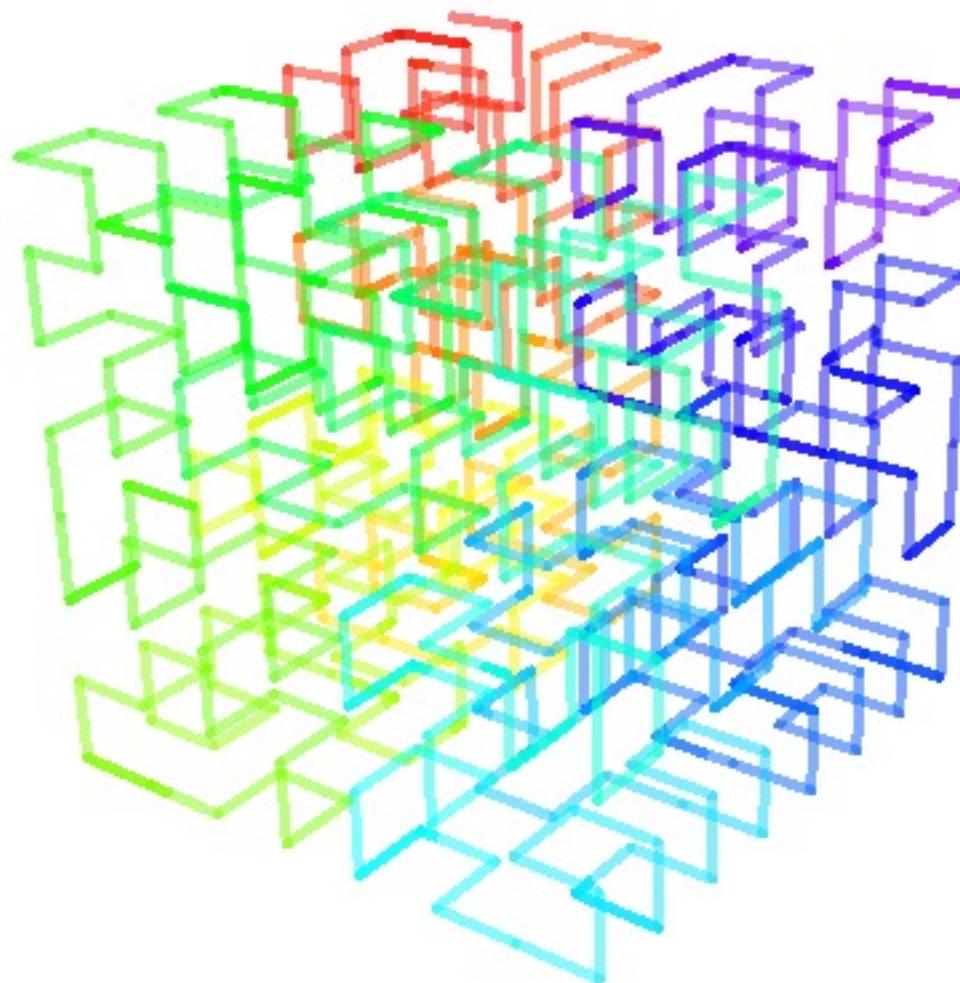


\*this is not to be confused with domain discretisation!

## Computer Architectures

- how to program such machines?

3D array



## Computer Architectures

- how to check the speed-up of your program?

## Computer Architectures

- how to check the speed-up of your program?

**strong scaling**

**weak scaling**

## Computer Architectures

- how to check the speed-up of your program?

### **strong scaling**

you keep the problem size fixed,  
but increase the number of CPU's

### **weak scaling**

you keep the number of CPU's fixed,  
but increase the problem size

## Computer Architectures

- how to check the speed-up of your program?

### **strong scaling**

you keep the problem size fixed,  
but increase the number of CPU's

*you aim at running a given problem  
as fast as possible...*

### **weak scaling**

you keep the number of CPU's fixed,  
but increase the problem size

*you aim at running the largest possible  
problem in a given amount of time...*

## Computer Architectures

- how to check the speed-up of your program?

### **strong scaling**

you keep the problem size fixed,  
but increase the number of CPU's

*you aim at running a given problem  
as fast as possible...*

### **weak scaling**

you keep the number of CPU's fixed,  
but increase the problem size

*you aim at running the largest possible  
problem in a given amount of time...*

**actually more important nowadays**

## Computer Architectures

- how to actually write a program?

## Computer Architectures

- how to actually write a program?
  - define the problem
  - decide on organisation
    - choose essential elements (variables, structures, etc.)
    - shape relevant tasks
    - design your algorithm to be parallelizable
    - draw a flowchart
  - code in your preferred language
  - test code using simple/known test cases

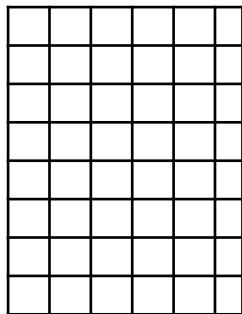
## Computer Architectures

- how to actually write a program?
  - define the problem
  - decide on organisation
    - choose essential elements (variables, structures, etc.)
    - **shape relevant tasks**
    - **design your algorithm to be parallelizable**
    - draw a flowchart
  - code in your preferred language
  - test code using simple/known test cases

**try to break problem down into pieces/modules  
that can be coded separately from each other...**

## Computer Architectures

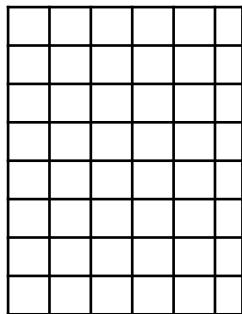
- shaping relevant tasks?



## Computer Architectures

- **shaping relevant tasks – data**

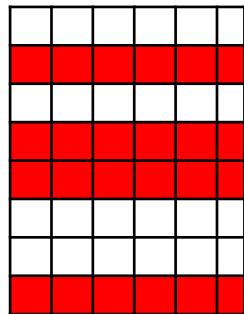
divide data into sub-sets,  
and perform different calculation with each sub-set...



## Computer Architectures

- shaping relevant tasks – data

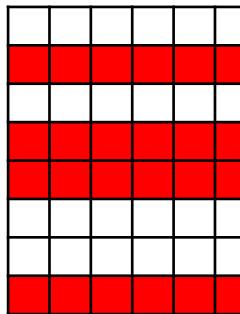
divide data into sub-sets,  
and perform different calculation with each sub-set...



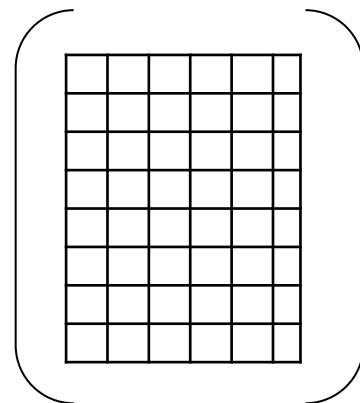
## Computer Architectures

### ■ shaping relevant tasks – data

divide data into sub-sets,  
and perform different calculation with each sub-set...



Analyse(Data)

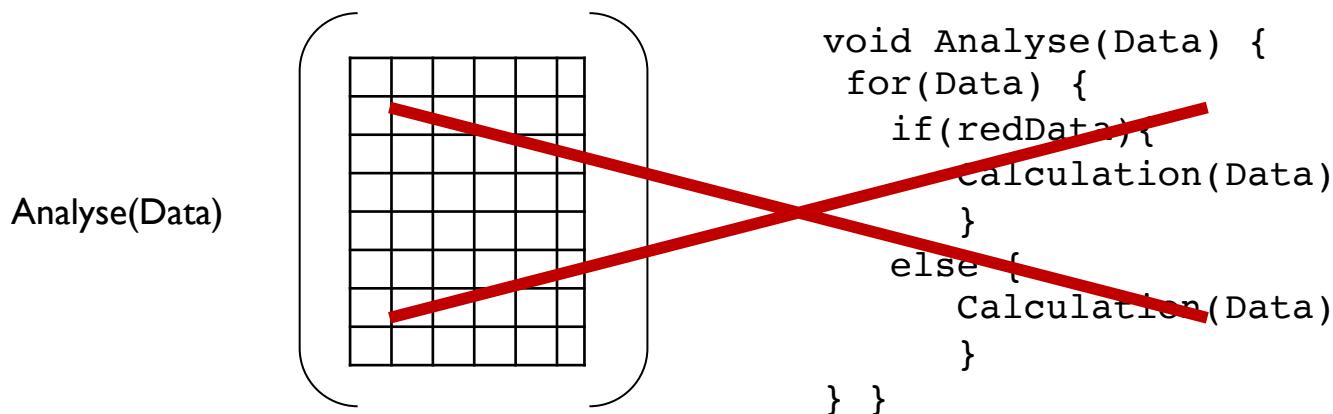
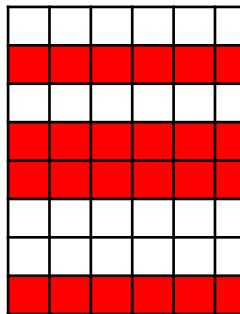


```
void Analyse(Data) {
 for(Data) {
 if(redData){
 Calculation(Data)
 }
 else {
 Calculation(Data)
 }
 } }
```

## Computer Architectures

### ■ shaping relevant tasks – data

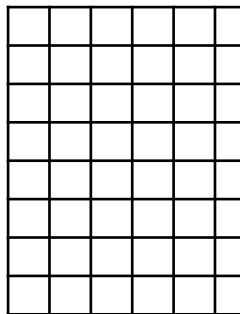
divide data into sub-sets,  
and perform different calculation with each sub-set...



## Computer Architectures

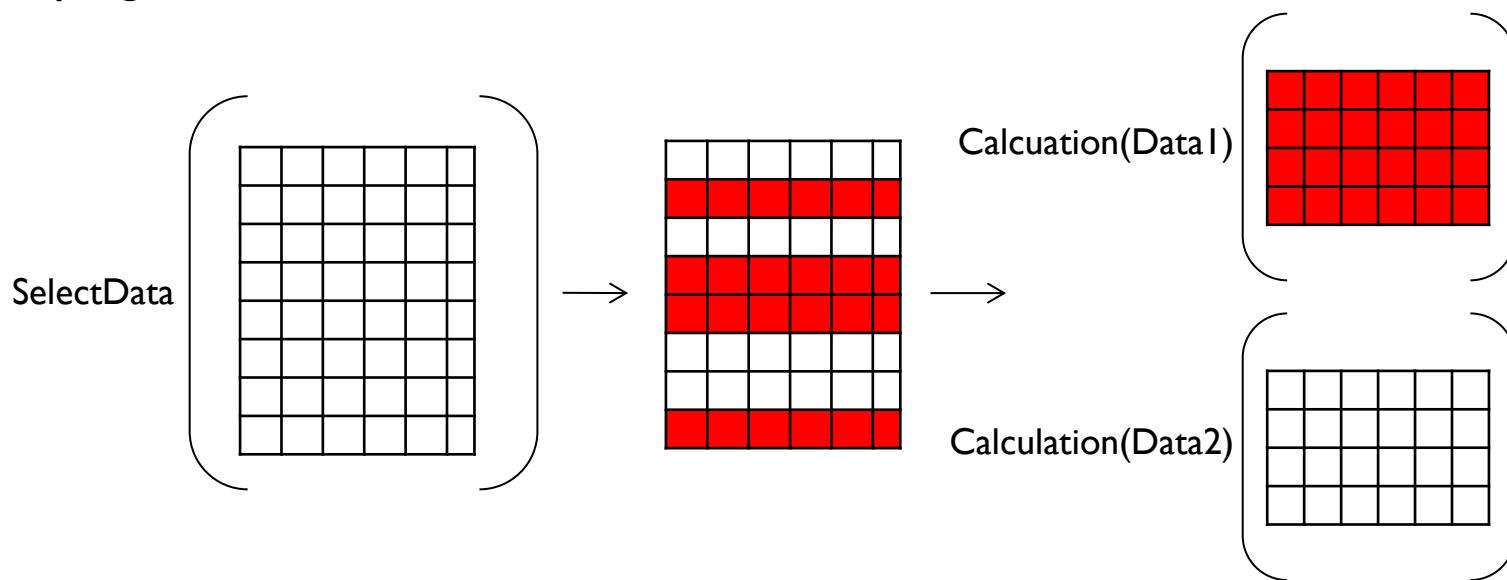
- **shaping relevant tasks – data**

divide data into sub-sets,  
and perform different calculation with each sub-set...



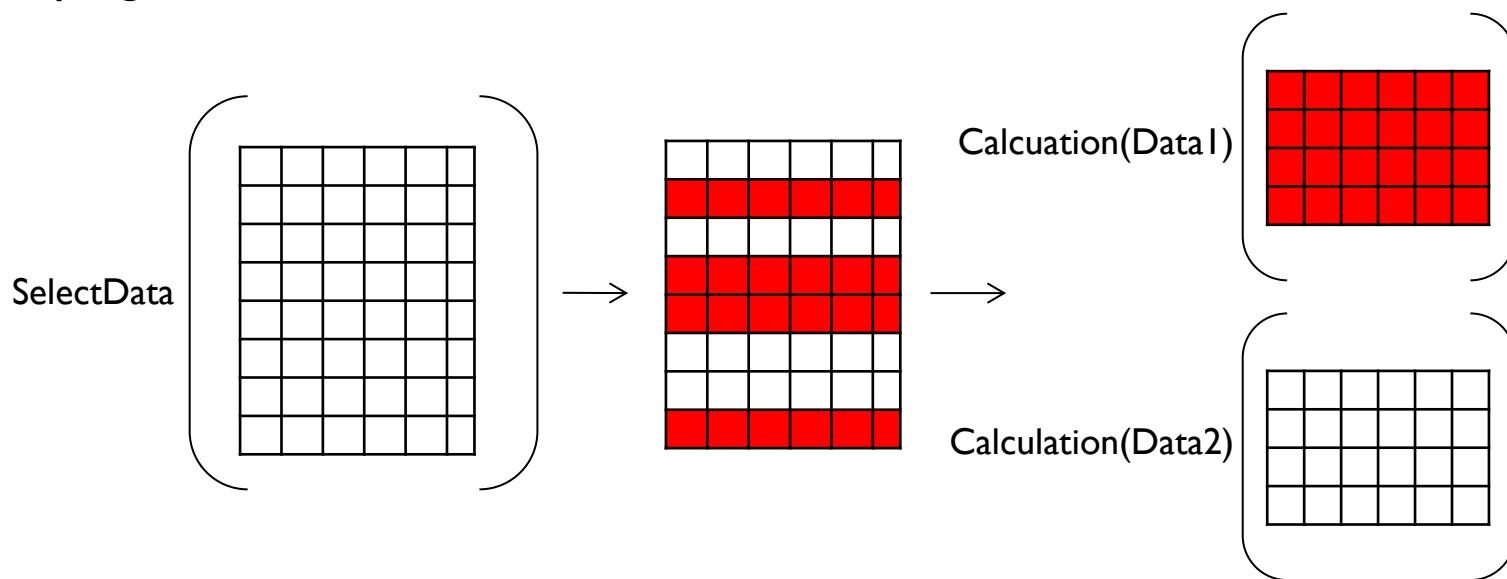
## Computer Architectures

- shaping relevant tasks – data



## Computer Architectures

- shaping relevant tasks – data



**modular and hence more flexible!**

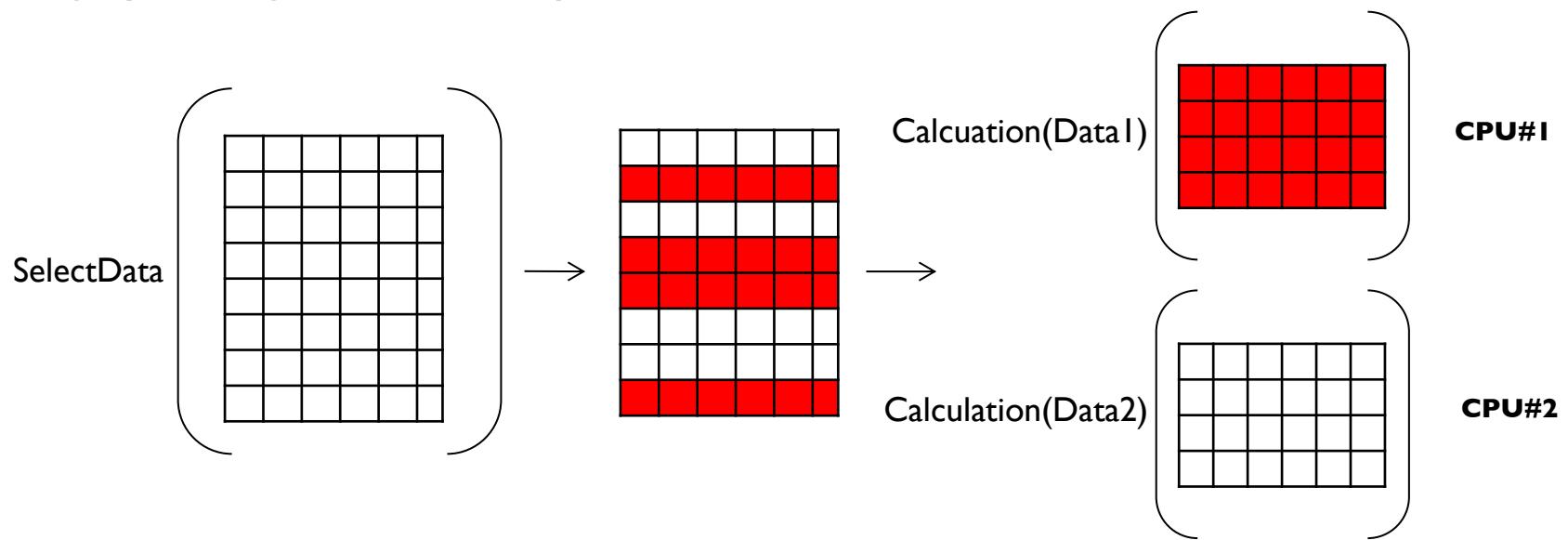
## Computer Architectures

- how to actually write a program?
  - define the problem
  - decide on organisation
    - choose essential elements (variables, structures, etc.)
    - shape relevant tasks
    - **design your algorithm to be parallelizable**
    - draw a flowchart
  - code in your preferred language
  - test code using simple/known test cases

**each CPU runs the same code,  
but on a different part of the problem...**

## Computer Architectures

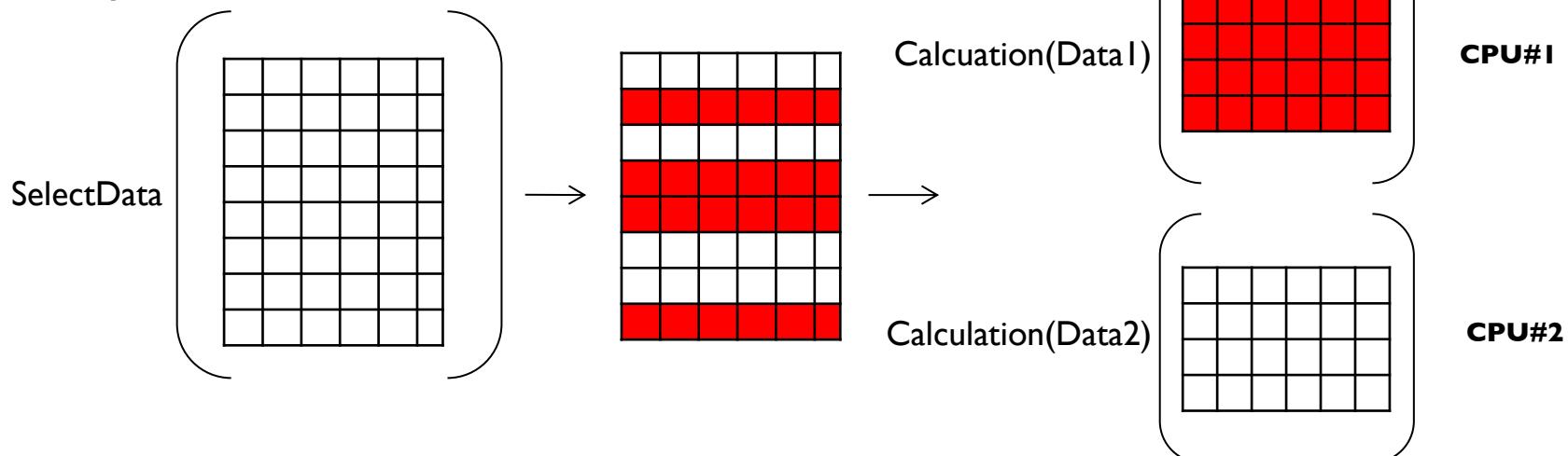
- design your algorithm to be parallelizable!



## Computer Architectures

- design your algorithm to be parallelizable!

### data parallelisation

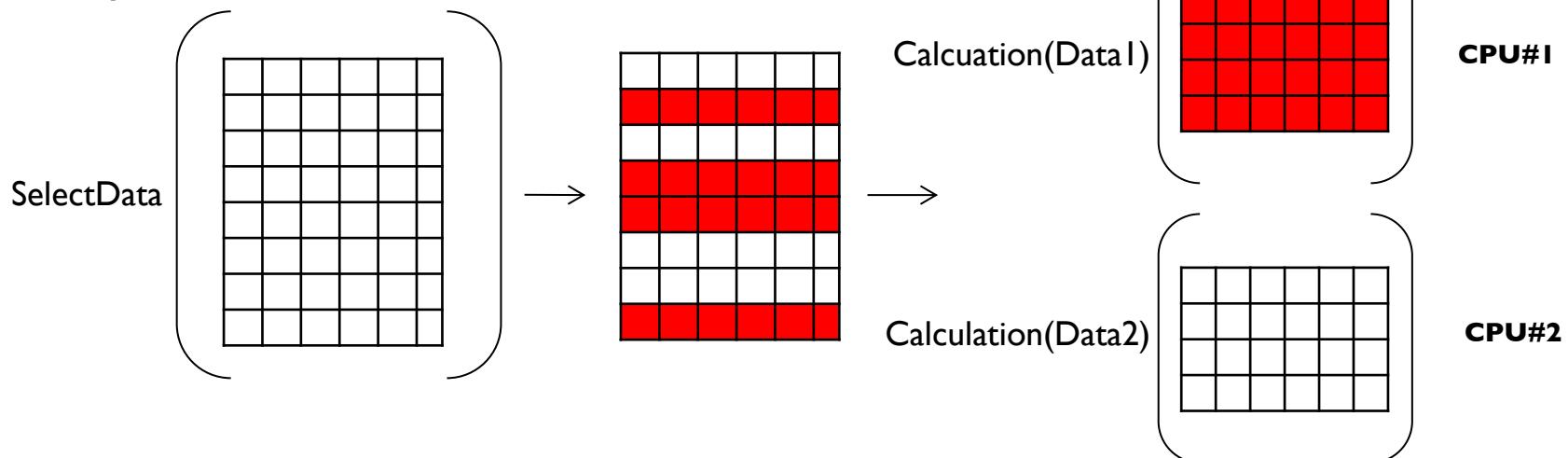


### task parallelisation

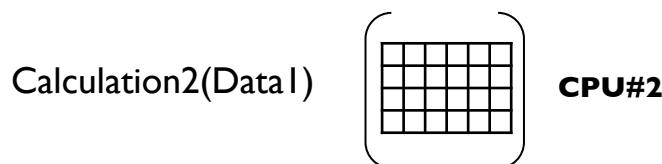
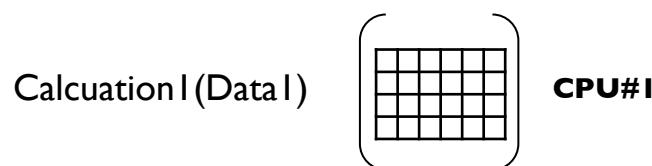
## Computer Architectures

- design your algorithm to be parallelizable!

### data parallelisation



### task parallelisation



## Computer Architectures

### ■ some coding recommendations:

- make proper use of the Cache (see above)
  - avoid complicated indices
  - know how arrays are aligned in memory
- avoid conditions, I/O, and (sub-)routine calls inside loops
- avoid unnecessary operations inside loops in general
- use multiplications rather than divisions or powers
- keep it simple!



let's put our knowledge into action:

write a code that calculates

$$\pi \approx \sqrt{12} \sum_{k=0}^{\infty} \left(-\frac{1}{3}\right)^k \frac{1}{2k+1}$$

You can first write a serial program to ensure your function is working as expected and play with the values of k.

Then think how are you going to parallelise your code.

For the total sum use: reduction(+:sum)

#pragma omp parallel for reduction(+:sum)

(This is by default initialised to 0.)

Compare the performance of your serial and parallel programs, using

# Computational Astrophysics

---

## Computer Architectures

k=0,1,2,3,4

```
pi += pow(-1.0/3.0, (double)k) / (2.*(double)k+1.);
```

k=5,6,7,8,9

```
pi += pow(-1.0/3.0, (double)k) / (2.*(double)k+1.);
```

k=10,11,12,13

```
pi += pow(-1.0/3.0, (double)k) / (2.*(double)k+1.);
```