CS3210
Introduction

Lecture 1

Outline

- Parallel Computing
  - Motivation
  - Serial vs Parallel Computing
  - Serial vs Parallel Processing Units
- Basic of Parallel Computing
  - Decompose, Schedule and Map
  - Performance
  - Challenges
- Landscape of Parallel Computing
  - Applications
  - Systems

Cheetah vs Hyena

- Weigh 60 kg
- Fastest land animal
  - Top speed of 120 km/h
  - 0-100 km/h in 3 seconds
- Hunt alone
  - Usually over in 1 minute
- 50% of losing its catch to other predators

- Weigh 50 kg
- Top speed of 65 km/h
- Hunt in group of 4-40
- Can take down large animals like buffalo, rhino, lion

Observations

- Optimized on speed only

- Single Processor:
  - Failed to meet Moore's Law since early 2000

- Optimized to **cooperate** and **work in parallel**

- Parallel Computer:
  - A collection of processing elements that cooperate to solve problems quickly
  - Huge amount of computation power

Parallel Computing - Motivation

- In the first issue of CACM (1958), **Saul Gorn** notes that:

> "We know that the so-called parallel computers are somewhat faster than the serial ones, and that no matter how fast we make **access and arithmetic** serially, we can do correspondingly better in parallel. However access and arithmetic speeds seem to be approaching a definite limit ... Does this mean that digital speeds are reaching a limit, that digital computation of multi-variate partial differential systems must accept it, and that **predicting tomorrow's weather must require more than one day**? Not if we have truly-parallel machines, with say, a thousand instruction registers."

Hitting the Wall

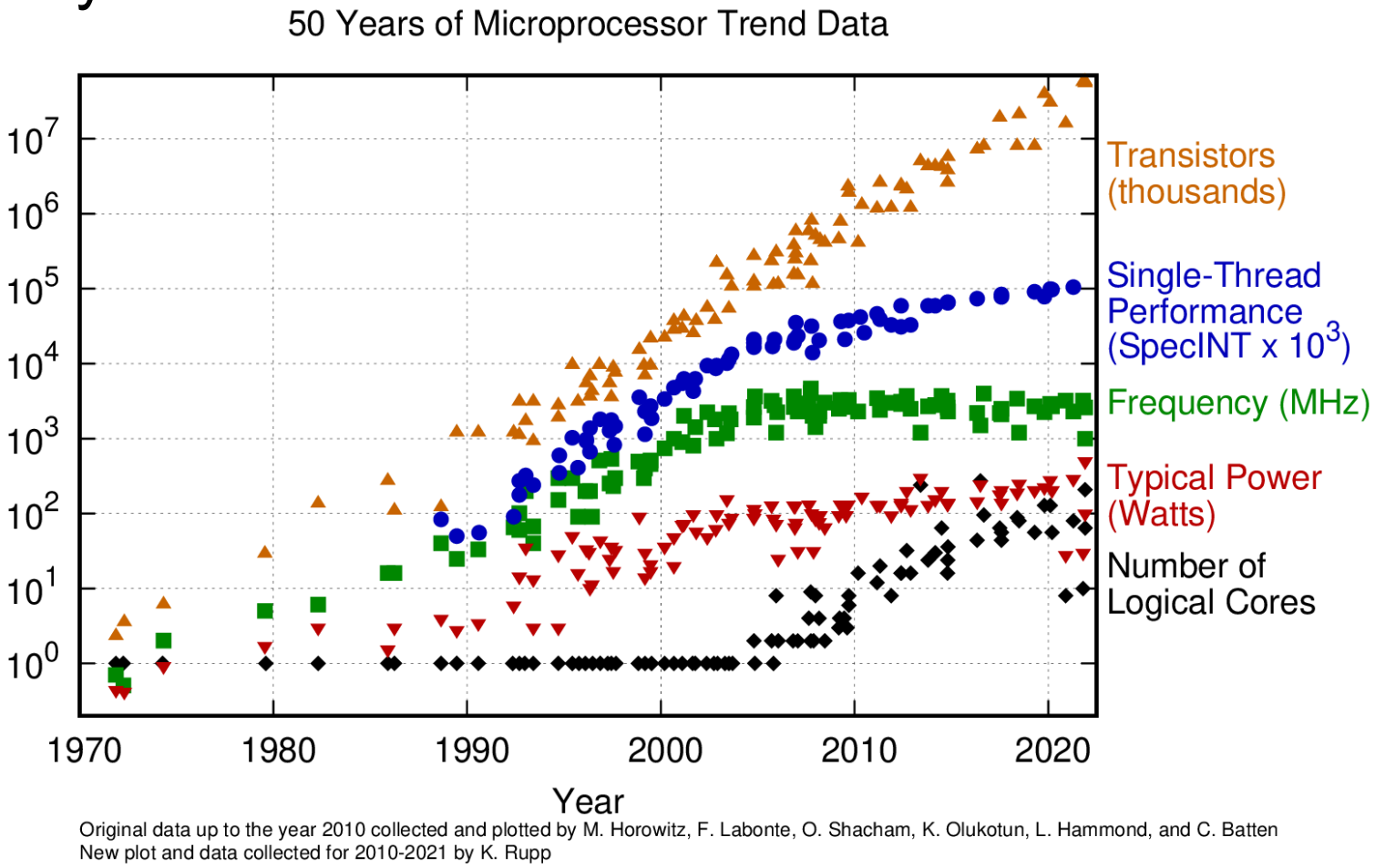
- '70s: "Supercomputers" for Scientific Computing
- Early '90s: Databases
- Inflection point in 2004: Intel hits the Power Density Wall

> "obtaining more computing power by stamping multiple processors on a single chip rather than straining to increase the speed of a single processor."
>
> John Markoff, New York Times, May 17, 2004

*Cooking aware computing

How do you make your program run faster?

- Answer *before 2004*:
  - Just wait 6 months and buy a new machine!
- Answer *after 2004*:
  - Write parallel software



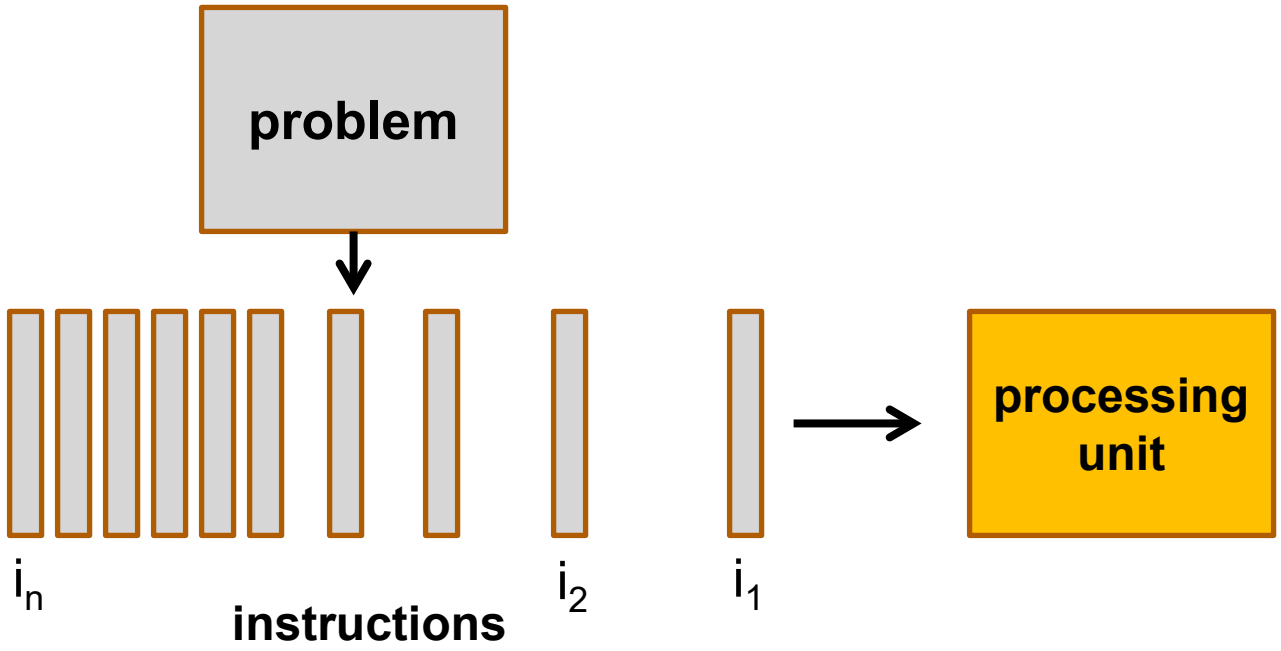
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

Parallel Computing - Challenges

- Gorn also recognized the challenges that would be posed to the system designer by the new parallel computers:

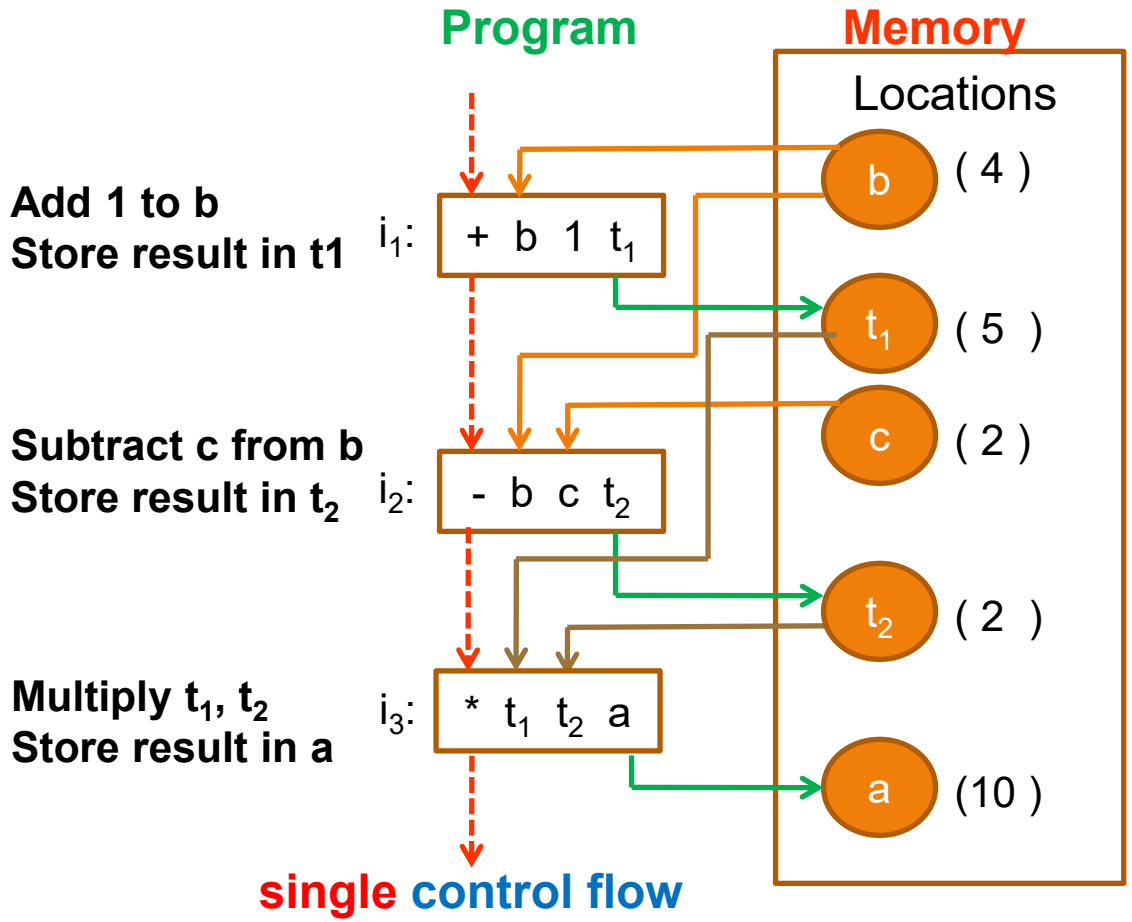
"But visualize what it would be like to program for such a machine! If a thousand subroutines were in progress simultaneously, one must program the recognition that they have all finished, if one is to use their results together. The programmer must not only synchronize his subroutines, but schedule all his machine units, unless he is willing to have most of them **sitting idle most of the time**. Not only would **programming techniques be vastly different from the serial ones** in the serial languages we now use, but they would be humanly impossible without machine intervention."

Serial Computing



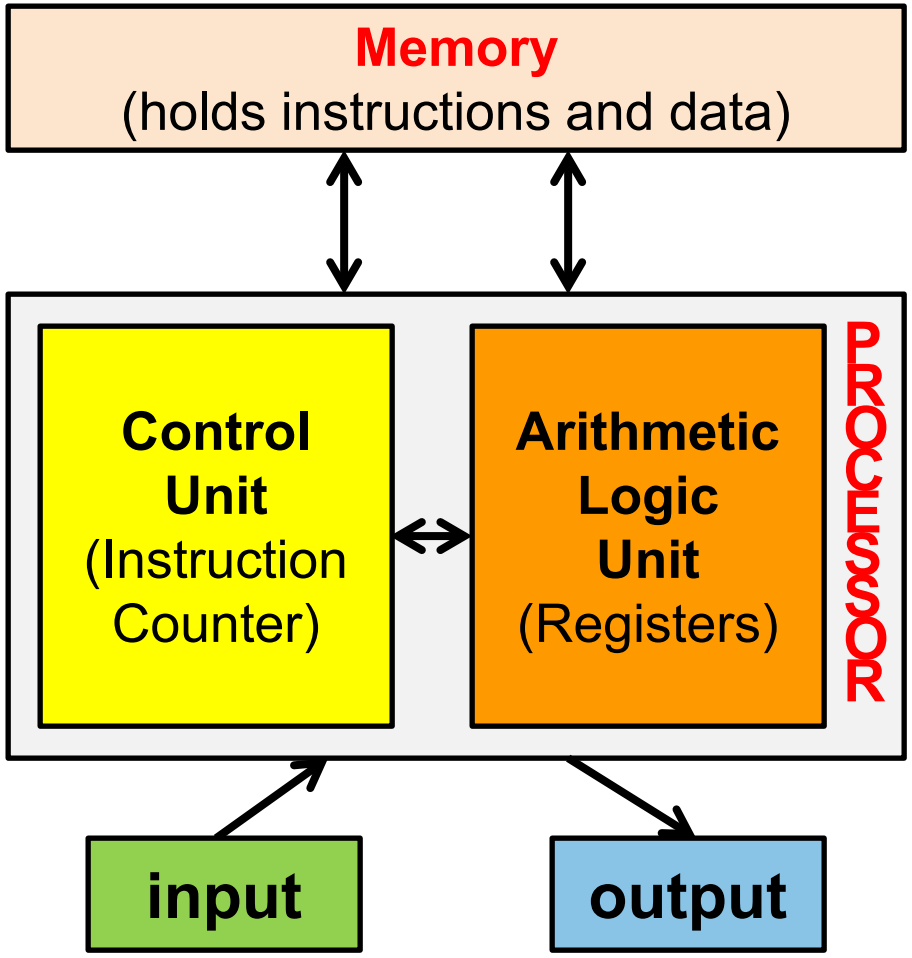
- Traditionally, a problem is divided into a discrete series of instructions
  - Instructions are executed one after another
  - Only one instruction executed at any moment in time

Example: Evaluate `a = (b+1) * (b-c)`



Sequential Computation Model

Processing Unit: von Neumann Model



von Neumann Computation Model

- Processor:
  - Performs instructions
- Memory:
  - Stores both the instructions and data in cells with unique addresses
- Control Scheme:
  - Fetches instruction one after another from memory
  - Shuttles data between memory and processor
- Memory wall
  - ➔ disparity between processor (< 1 nanosecond or 10^{-9} sec) and memory speed (100-1,000 nanoseconds)

Ways to improve performance

1. **Work hard**
   - ➢ higher clock frequency
2. **Work smart**
   - ➢ pipelining, superscalar processor
3. **Get help**
   - ➢ replication – multicore, cluster, ..

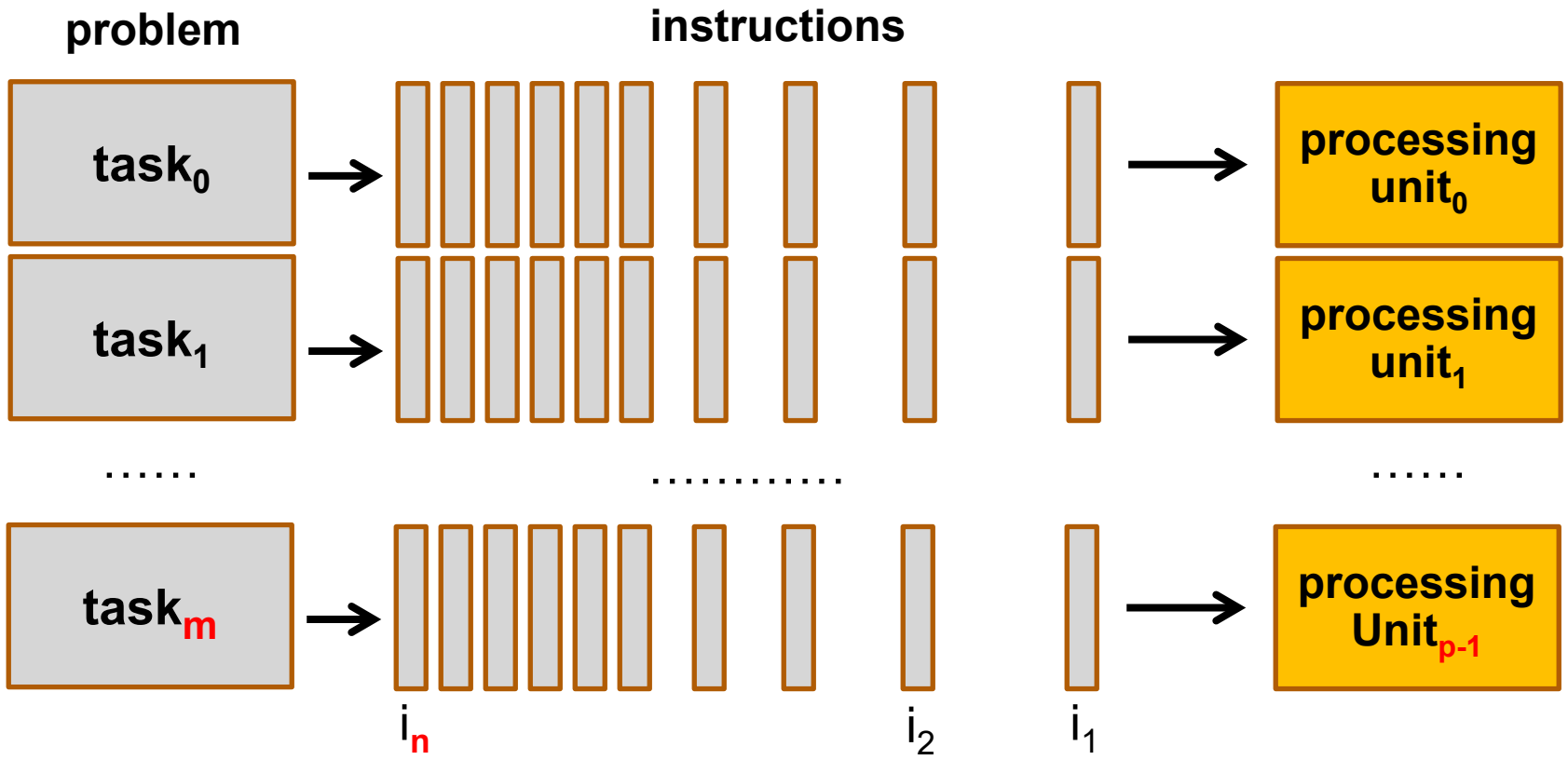
Parallel Computing

- **Simultaneous use** of **multiple processing units** to solve a problem fast / solve a larger problem

- **Processing units** could be:
  i. A single processor with multiple cores
  ii. A single computer with multiple processors
  iii. A number of computers connected by a network
  iv. Combinations of the above

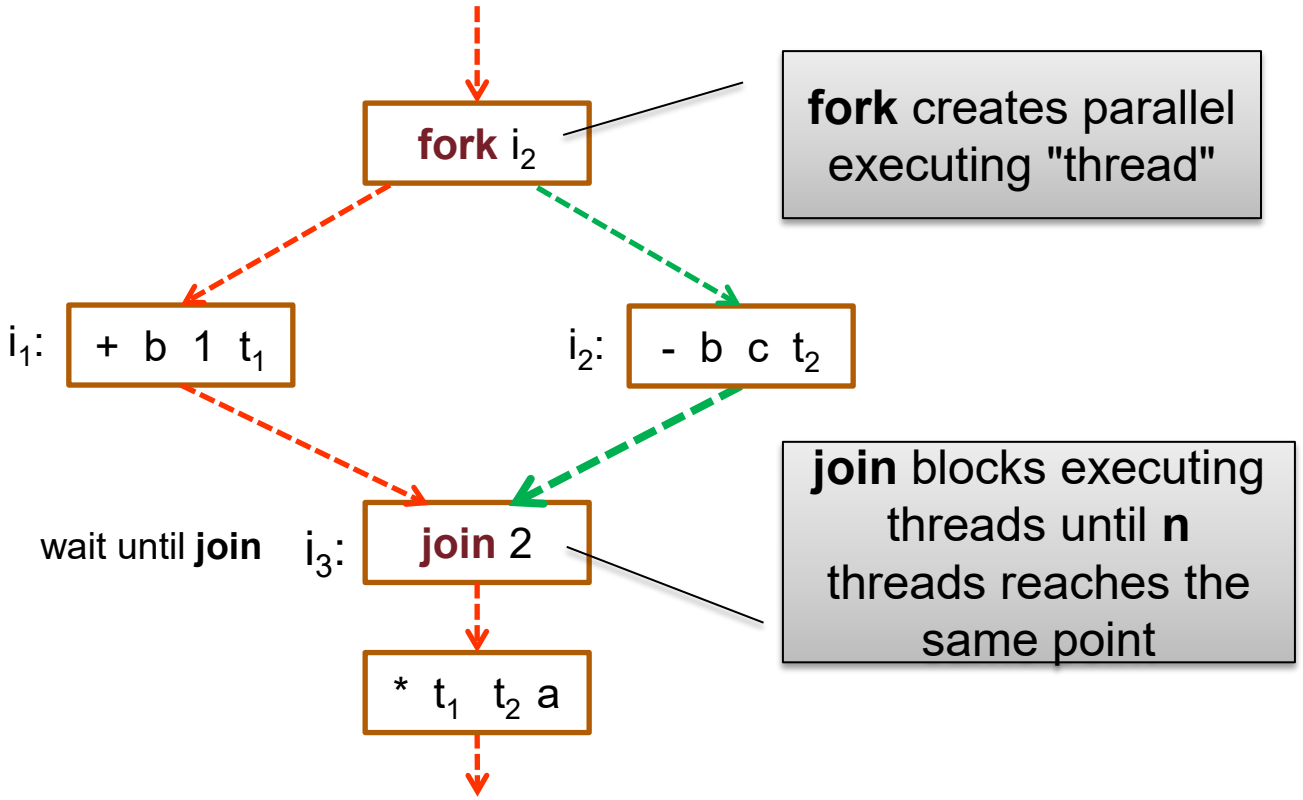
- Ideally, a problem (application) is partitioned into sufficient number of independent parts for execution on parallel processing elements

Parallel Computing



1. A problem is divided into **m** discrete parts (tasks) that can be solved concurrently
2. Each part is further broken down to a series of instructions (**i**)
3. **Instructions from each part execute in parallel on different processing units (p)**

Example: Evaluate `a = (b+1) * (b-c)`



Parallel Computing (Shared-Memory)

Why Parallel Computing

- Primary reasons:
  1. Overcome limits of serial computing
  2. Save (wall-clock) time
  3. Solve larger problems

- Other reasons:
  1. Take advantage of non-local resources
  2. Cost saving – use multiple “cheaper” (commoditized) computing resources
  3. Overcome memory constraints

- Performance - exploits a large collection of processing units that can communicate and cooperate to solve large problems fast

BASIC OF PARALLEL COMPUTING

Computational Model Attributes

[Operation Mechanism]

- The *primitive units* of computation or basic actions of the computer (data types and operations defined by the instruction set)

[Data Mechanism]

- The definition of *address spaces* available to the computation (how data are accessed and stored)

[Control Mechanism]

- The *schedulable units* of computation (rules for partitioning and scheduling the problem for computation using the primitive units)

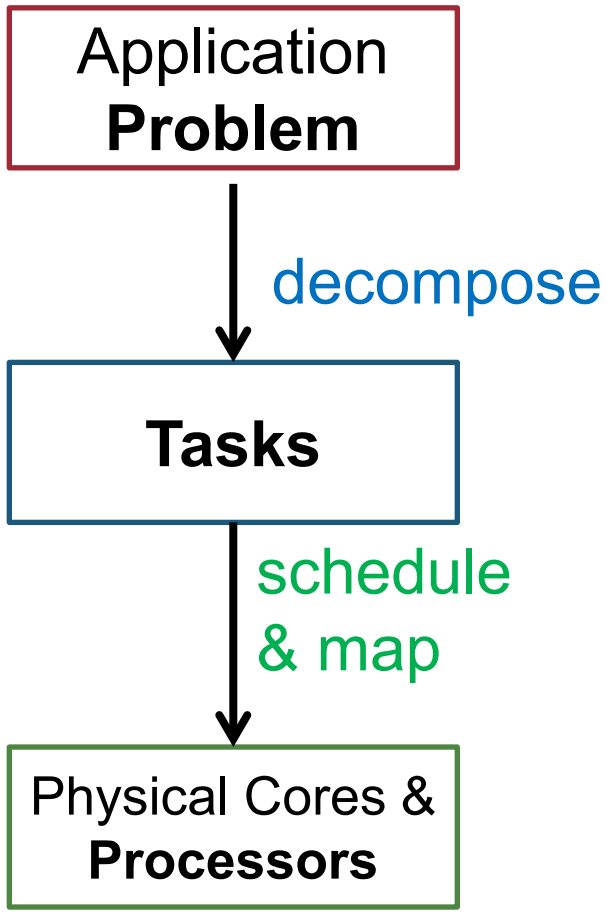
[Communication Mechanism]

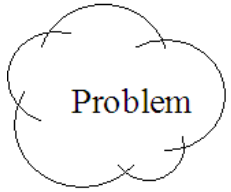
- The modes and patterns of **communication** among processing elements working in parallel, so that they can exchange needed information

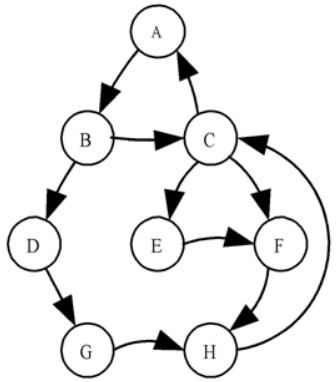
[Synchronization Mechanism]

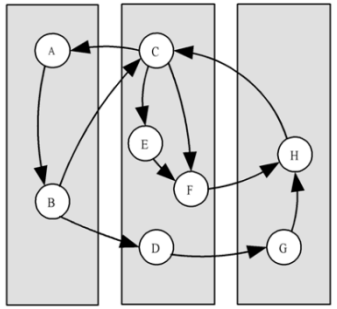
- Mechanism to ensure needed information arrives at the right time

Basics of Parallel Computing









Decomposition

- One problem → many possible *decompositions*

- Complicated and laborious process
  - Potential parallelism of an application dictates how it should be split into **tasks**
  - Size of tasks is called *granularity* – can choose different task sizes
  - Defining the tasks is challenging and difficult to automate (**why?**)

Tasks

- Tasks are coded in a parallel programming language

- **Scheduling**: Assignment of tasks to processes or threads
  - Dictates the task execution order
  - Manually defined? Static? Dynamic?

- **Mapping:** Assignment of processes/threads to physical cores/processors for execution

- Tasks may depend on each other resulting in **data** or **control dependencies**

  ➔ **impose execution order** of parallel tasks

Dependences and Coordination

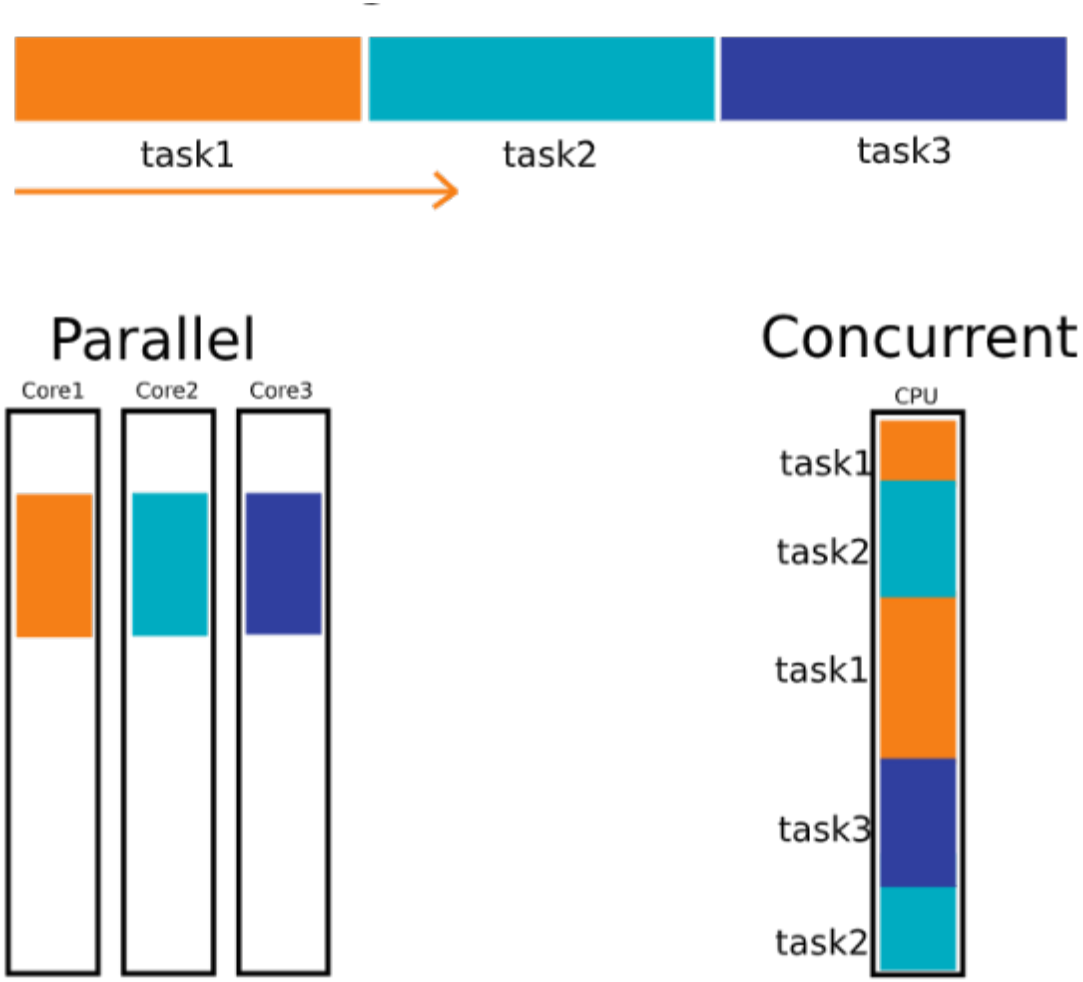
- Dependences among tasks impose constraints on scheduling

- To execute correctly, processes and threads need *synchronization* and *coordination*

  ➔ depends on information exchanges between processes and threads and in turns, depends on the hardware memory organization

- Memory organizations: **shared-memory** (threads) and **distributed-memory** (processes)

An illusion?


task1
task2
task3
Parallel
Core1
Core2
Core3
Concurrent
CPU
task1
task2
task1
task3
task2

Concurrency vs. Parallelism

Concurrency

- Two or more tasks can start, run, and complete in overlapping time periods
- They might not be running (executing on CPU) at the same instant
- Two or more execution flows make progress at the same time by interleaving their executions or by executing instructions (on CPU) at exactly the same time

Parallelism

- Two or more tasks can run (execute) simultaneously, at the exact same time
- Tasks do not only make progress, but they also actually execute simultaneously

Example – Serial Solution

- Find the sum for **n** numbers:

```
sum = 0;
for (i = 0; i < n; i++) {
    x = Compute_next_value(. . .);
    sum += x;
}
```

Serial Code

Example – Parallel Solution (1/3)

- Suppose we have **p cores** (p < n)
  - Each core can perform a partial sum of **n/p values**

```
my_sum = 0;
my_start = ……;
my_end = ……;
for (my_i = my_start; my_i < my_end; my_i++){
    my_x = Compute_next_value(. . .);
    my_sum += my_x;
}
```

  - Each core uses its own private variables and executes this block of code independently

Example – Parallel Solution (2/3)

- Upon completion, each core contains:
  - the partial sum in my_sum

- E.g. If n (=24), and p = 8
- Values generated:

1	4	3	9	2	8	5	1	1	5	3	7	2	5	0	4	1	8	6	5	1	2	3	9

- Partial sum calculated in each core:

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

Example – Parallel Solution (3/3)

- Next, a core designated as the master core adds up all values of my_sum to form the global sum (8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95)

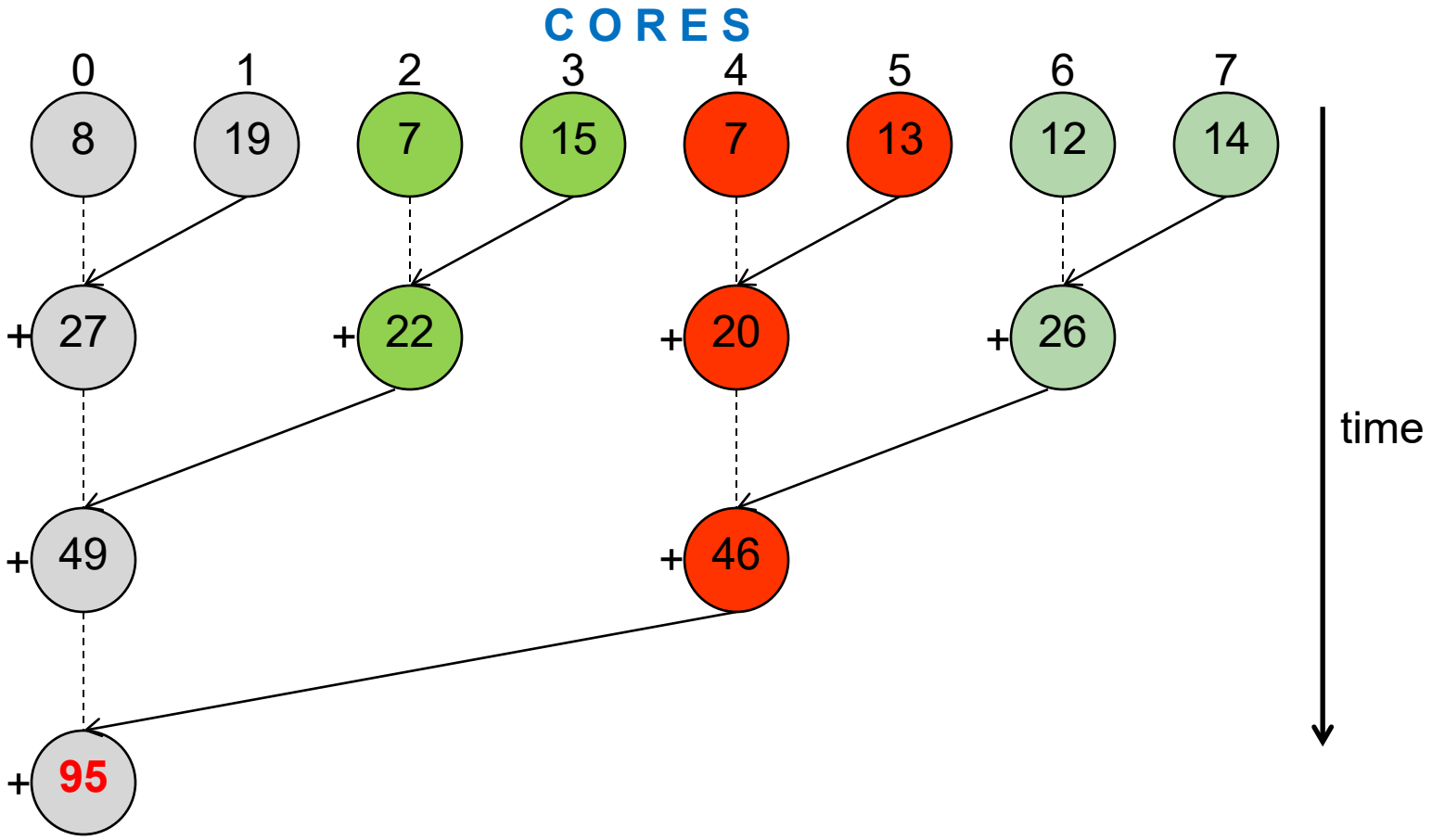
```
if (I'm the master core){
    sum = my_x;
    for each core other than myself {
        receive value from other core
        sum += value;
    }
} else {                    //not master core
     send my_x to the master;
}
```

Parallel Code

Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14

Better Parallel Algorithm

- Share the work of the global summing too:



Comparison: Two Parallel Solutions

- 8 cores
  - 1st version = 7 steps
  - 2nd version = 3 steps
  - factor of 2 improvement
- 1000 cores
  - 1st version = 999 steps
  - 2nd version = 10 steps
  - factor of 100 improvement

Parallel Performance

- Two perspectives:
  - Execution time versus Throughput
- Parallel execution time = computation time + parallelization overheads
- Parallelization overheads:
  - distribution of work (tasks) to processors
  - information exchange or synchronization
  - idle time
  - etc

Parallel Computing is HARD

"…when we start talking about parallelism and ease of use of truly parallel computers, we're talking about a problem that's as hard as any that computer science has faced. …

I would be panicked if I were in industry."

John Hennessy, President, Stanford University, 2007

Compiler Challenges

- Heterogeneous processors
  - Increase in the design space for code optimization
- Auto-tuners:
  - Optimizing code at runtime
- Software-controlled memory management

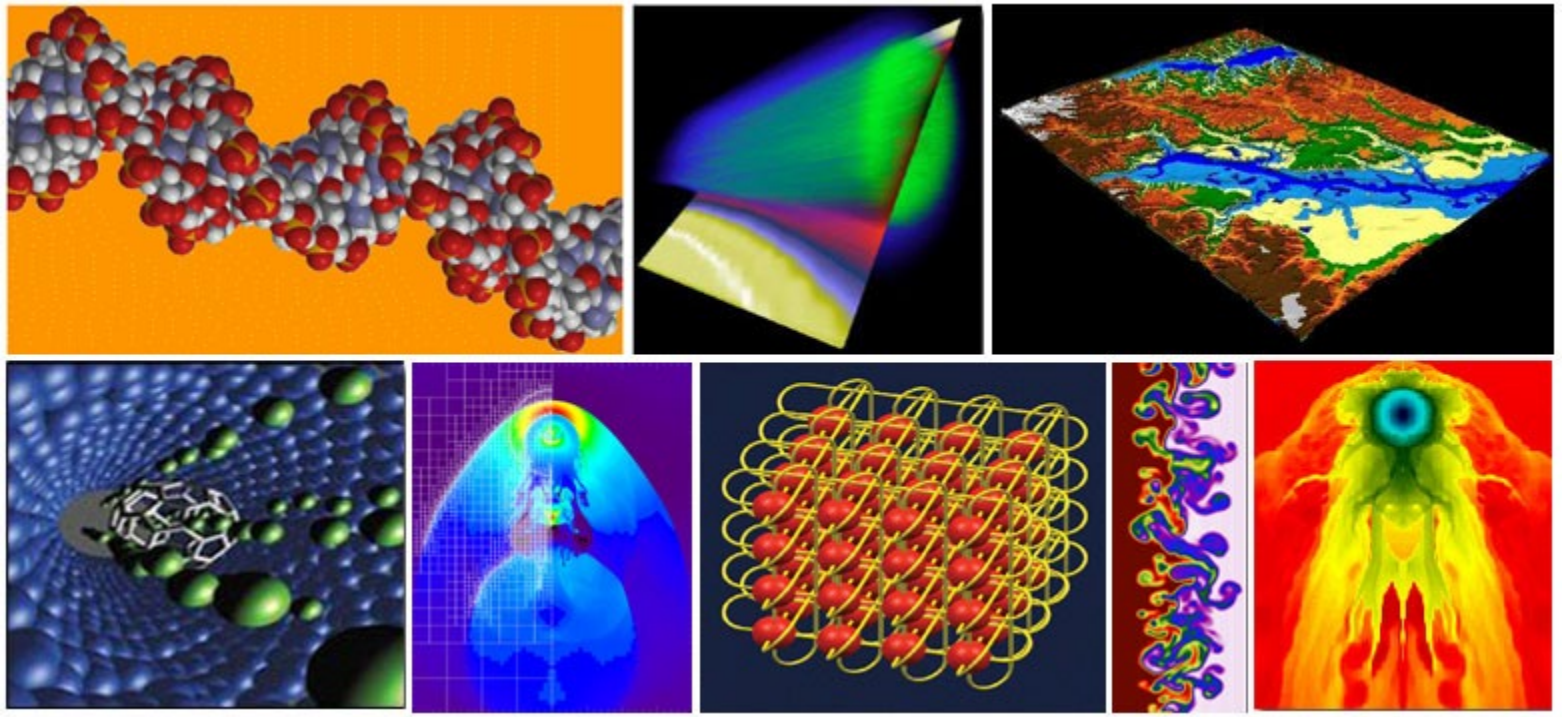
Parallel Programming Challenges

- Finding enough parallelism (Amdahl's Law!)
- Granularity
- Locality
- Load balance
- Coordination and synchronization
- Debugging
- Performance modeling / Monitoring

LANDSCAPE OF PARALLEL COMPUTING

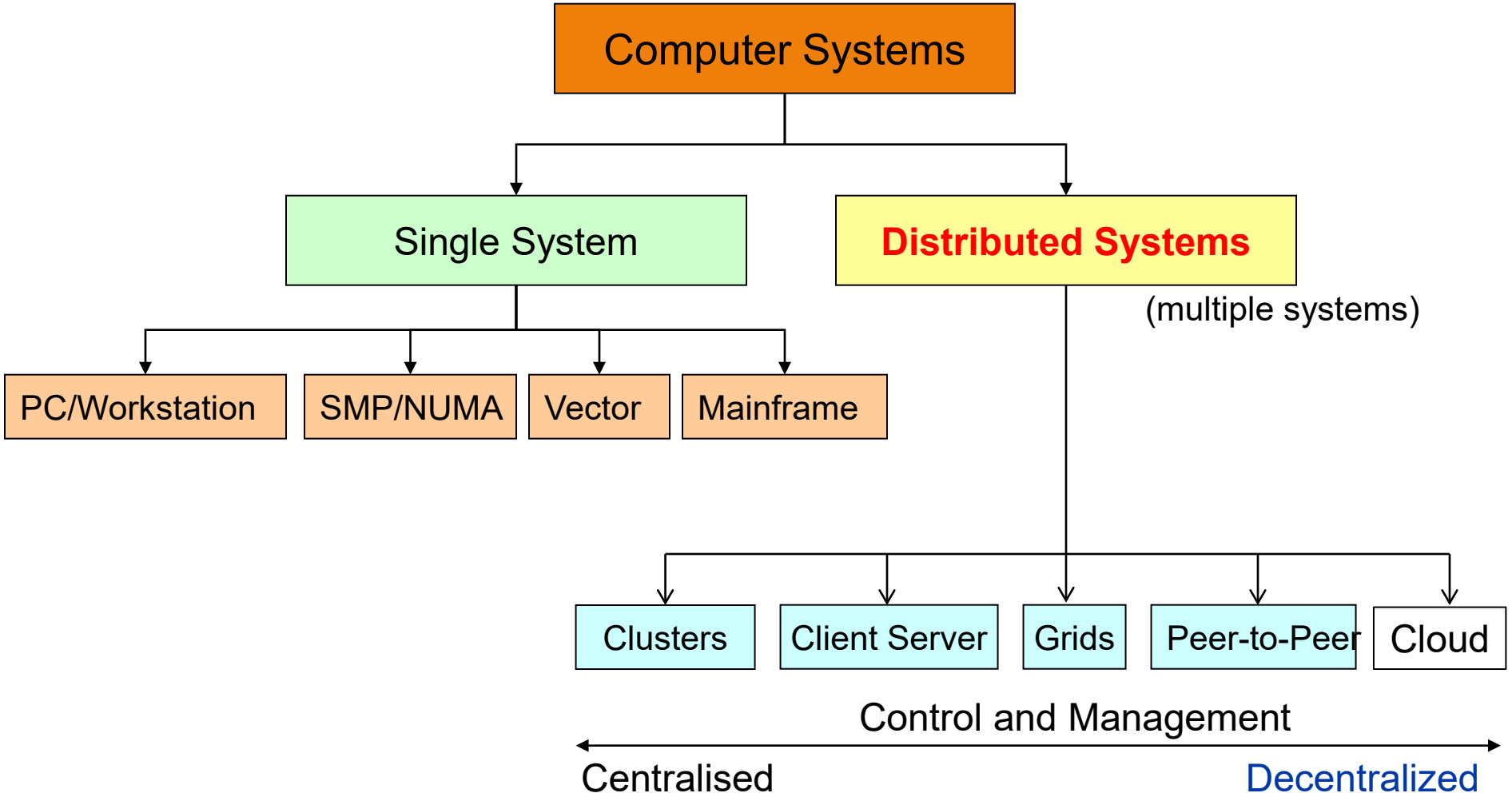
Uses of Parallel Computing

- Data mining, web search engines, web-based business services, pharmaceutical design, financial and economic modelling, etc.

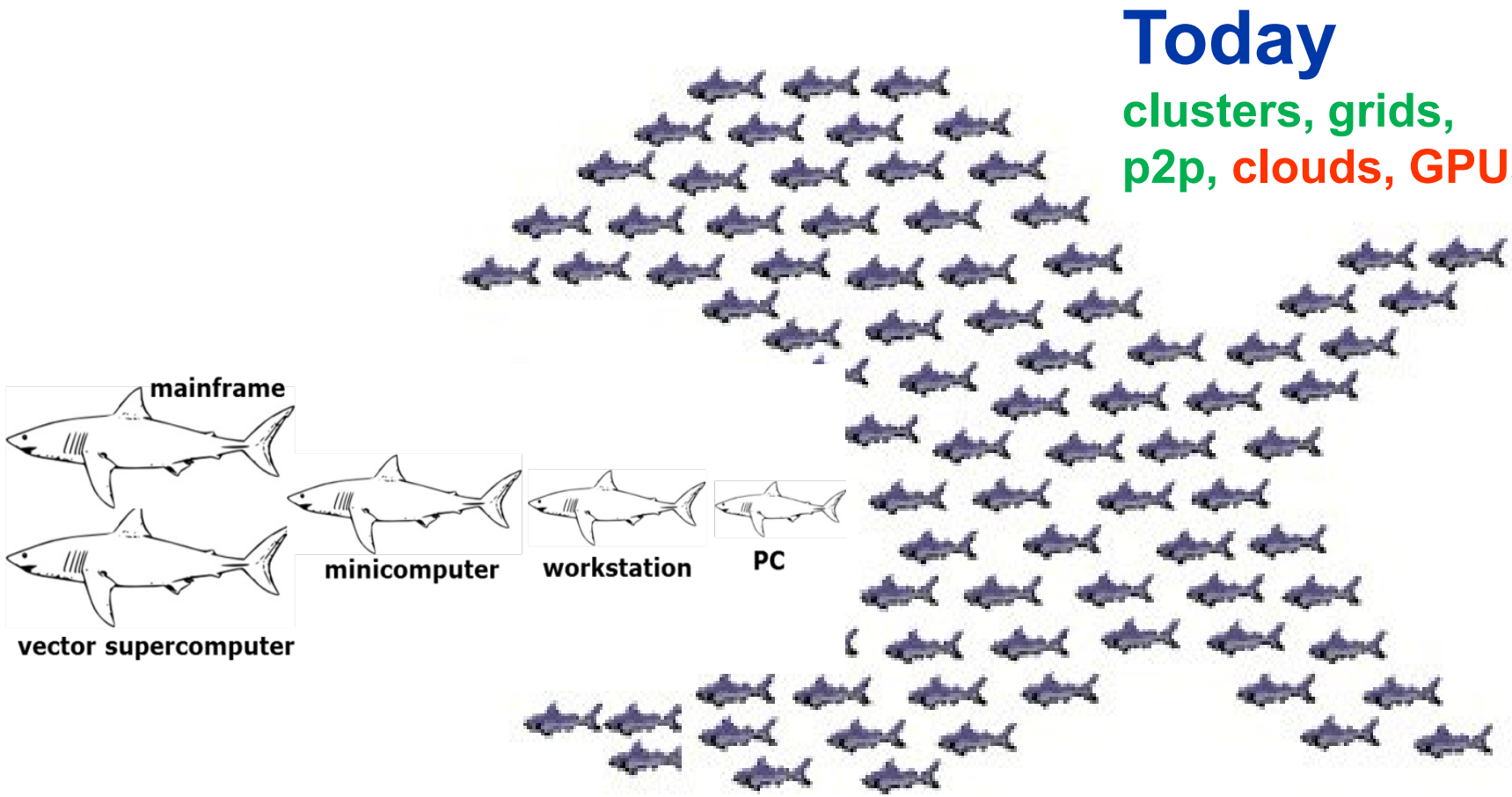


http://computing.llnl.gov/tutorials/parallel_comp

Types of Computer Systems

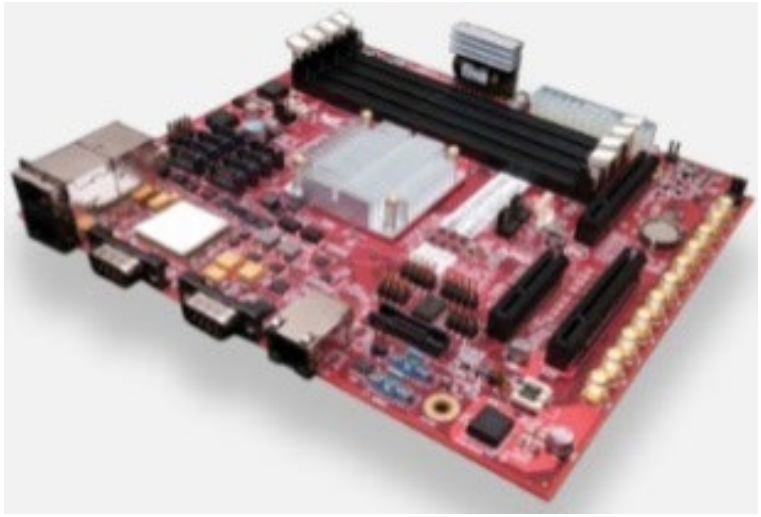


Parallel Computing "Food Chain"

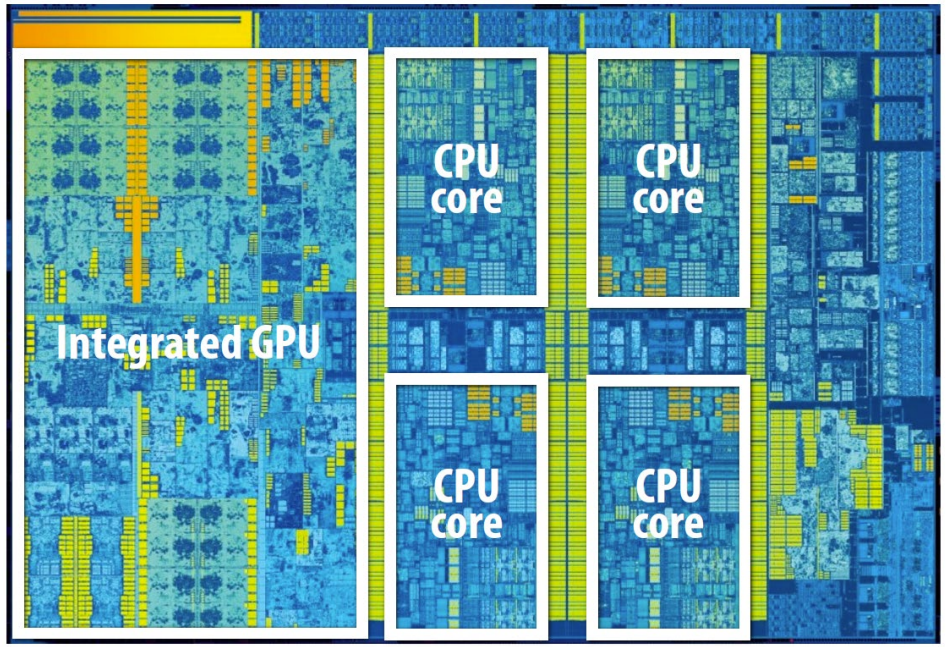


Parallel Computing and Parallel machines

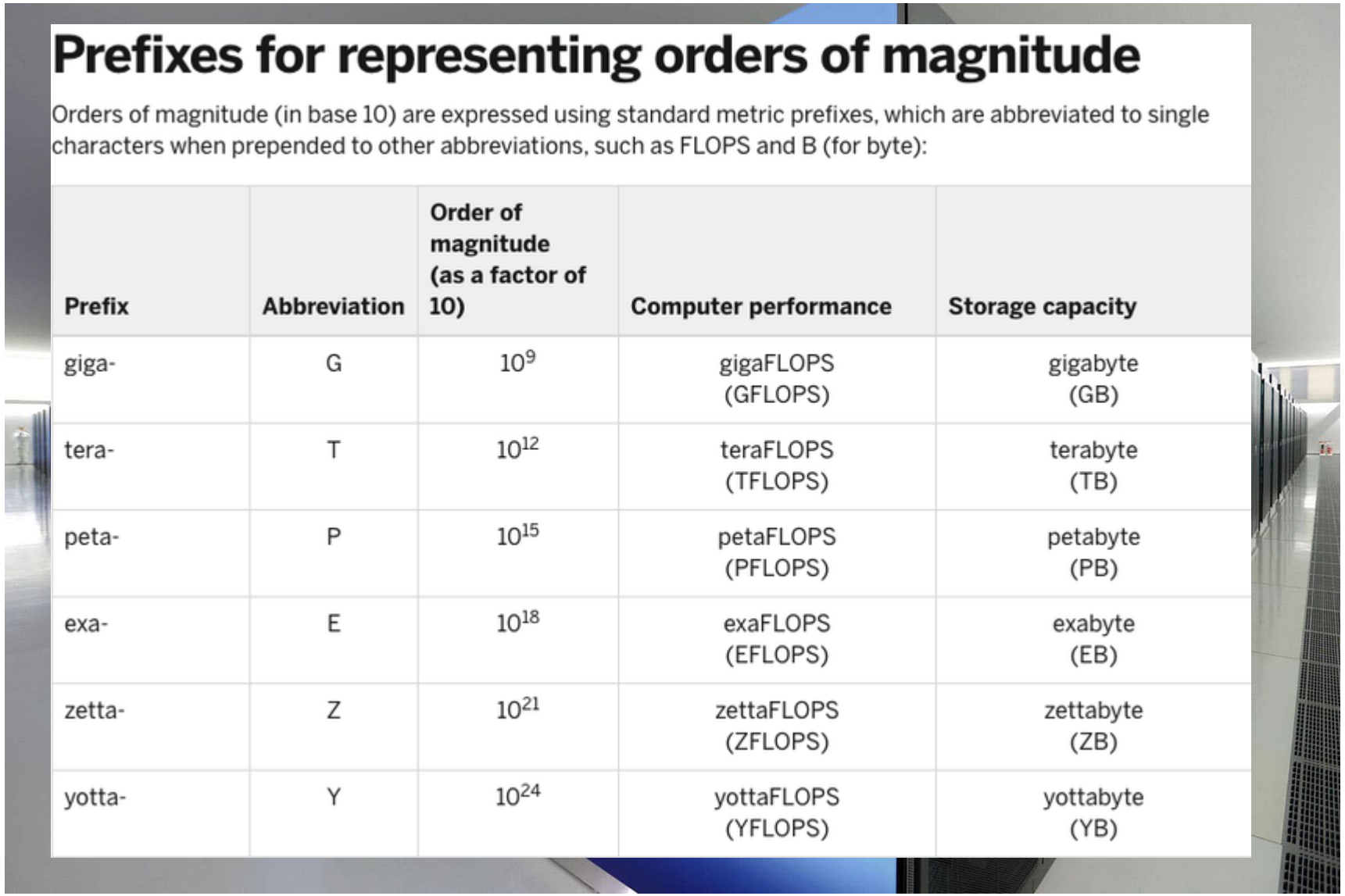
In your pocket:
ARM big.LITTLE 64-bit
(Cortex-A53 + Cortex-A73)
4-8 cores, >2GHz



Intel Skylake architecture:
Quad-core CPU + multi-core
GPU integrated on one chip



2020-21 Fastest Supercomputer



Prefixes for representing orders of magnitude

Orders of magnitude (in base 10) are expressed using standard metric prefixes, which are abbreviated to single characters when prepended to other abbreviations, such as FLOPS and B (for byte):

Prefix	Abbreviation	Order of magnitude (as a factor of 10)	Computer performance	Storage capacity
giga-	G	10^9	gigaFLOPS (GFLOPS)	gigabyte (GB)
tera-	T	10^{12}	teraFLOPS (TFLOPS)	terabyte (TB)
peta-	P	10^{15}	petaFLOPS (PFLOPS)	petabyte (PB)
exa-	E	10^{18}	exaFLOPS (EFLOPS)	exabyte (EB)
zetta-	Z	10^{21}	zettaFLOPS (ZFLOPS)	zettabyte (ZB)
yotta-	Y	10^{24}	yottaFLOPS (YFLOPS)	yottabyte (YB)

2022-23 Fastest Supercomputer

Supercomputer Frontier

1,194 petaflops
8,699,904 cores
AMD Optimized 3rd Gen EPYC 64C 2GHz
22 Mwatts
No. 2 in GREEN500

Video

2020-21 Fastest Supercomputer

Supercomputer Fugaku

415 petaflops
~7300000 cores ARM A64FX
29 Mwatts
No. 33 in GREEN500

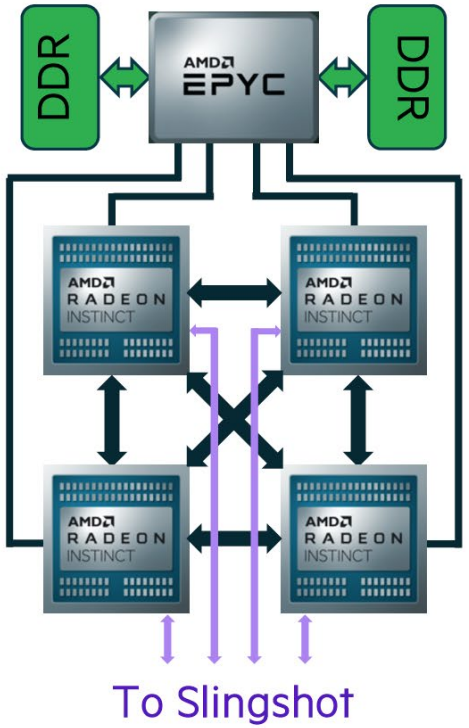
Video
Virtual tour

Usage of Frontier

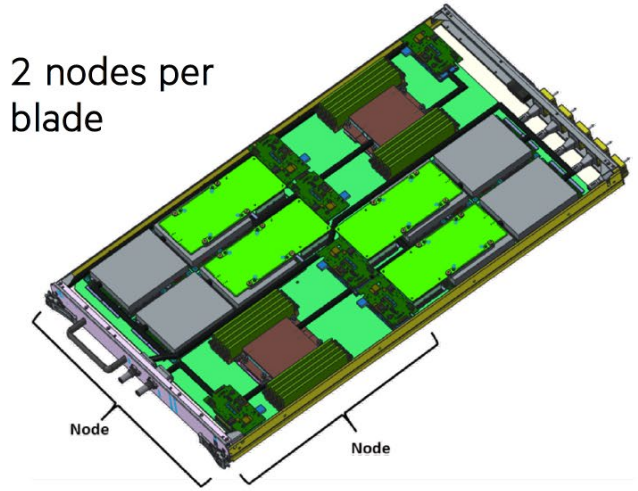
- Energy
  - Fusion energy studies
  - Combustion energy improvements
  - Innovative clean energy systems
- Materials study
- Social and scientific issues
  - Innovative drug discovery
  - Personalized and preventive medicine
- Disaster prevention and global climate problems
  - Meteorological and global environmental predictions
- Evolution of the universe

Deconstructing Frontier

- Node:
  - 1 HPC and AI Optimized AMD EPYC CPU
  - 4 Purpose Built AMD Radeon Instinct GPU
- 74 cabinets:
  - Over 9000 nodes
- Interconnect:
  - Multiple Slingshot NICs
  - Slingshot dragonfly network



AMD GPU (ORNL)

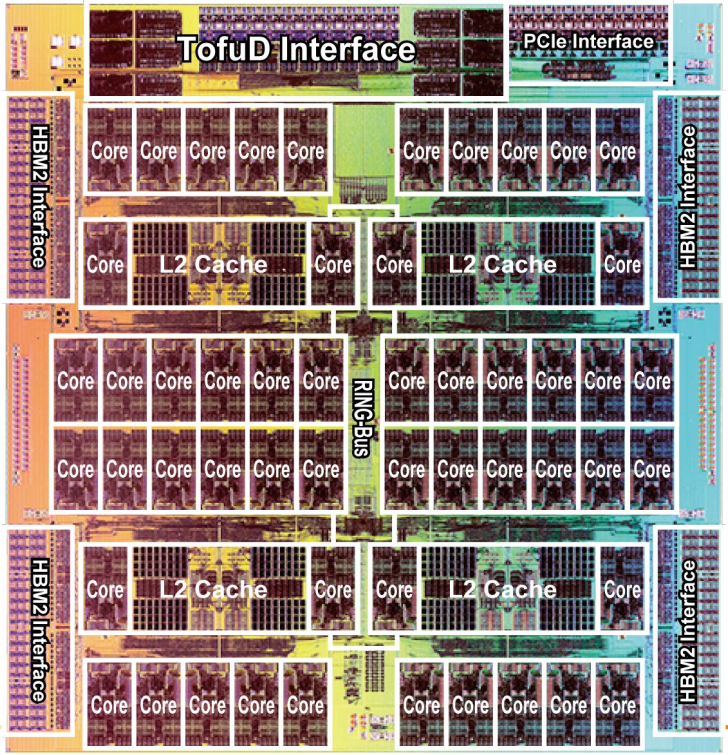


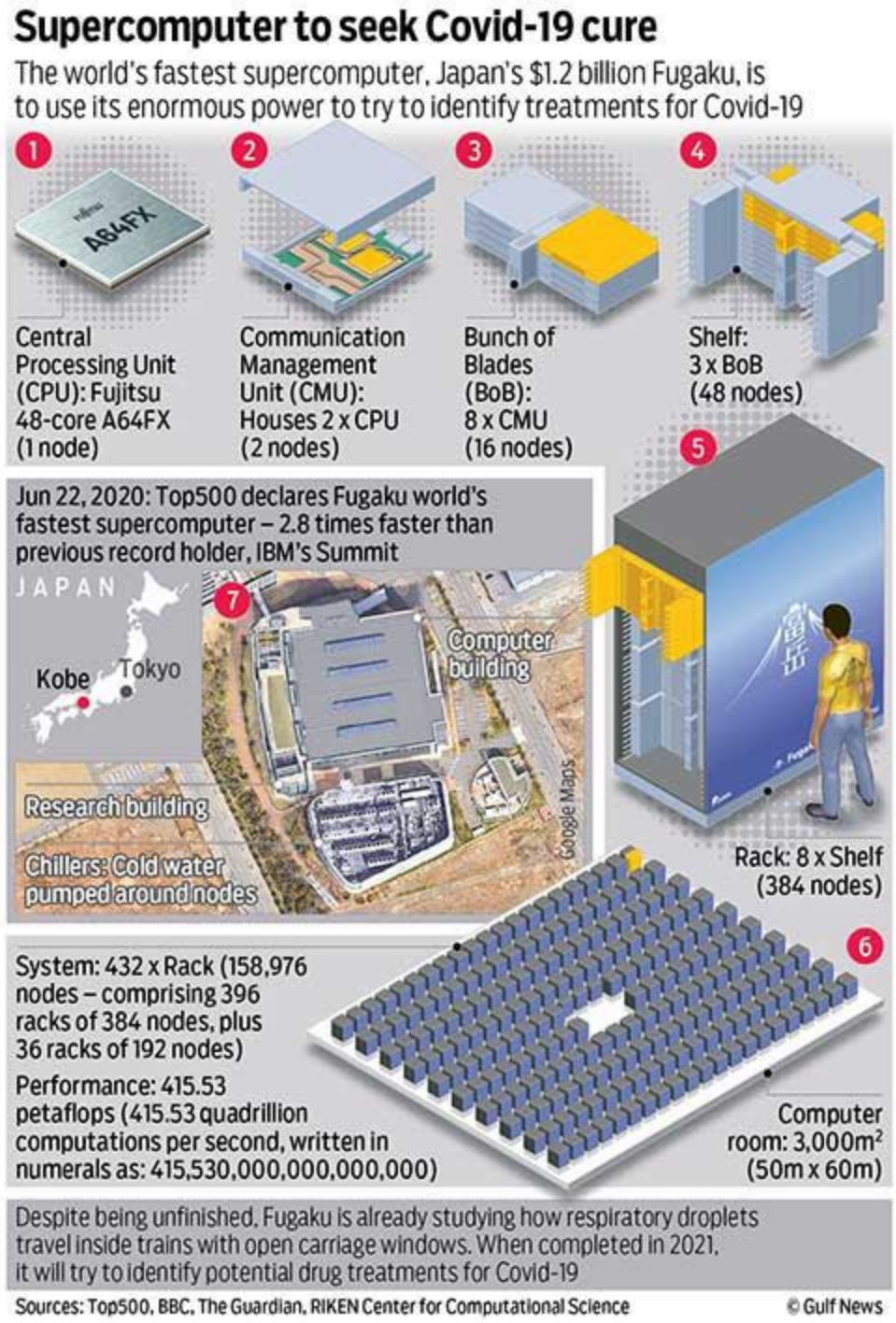
Deconstructing Fugaku

Processor (CPU):

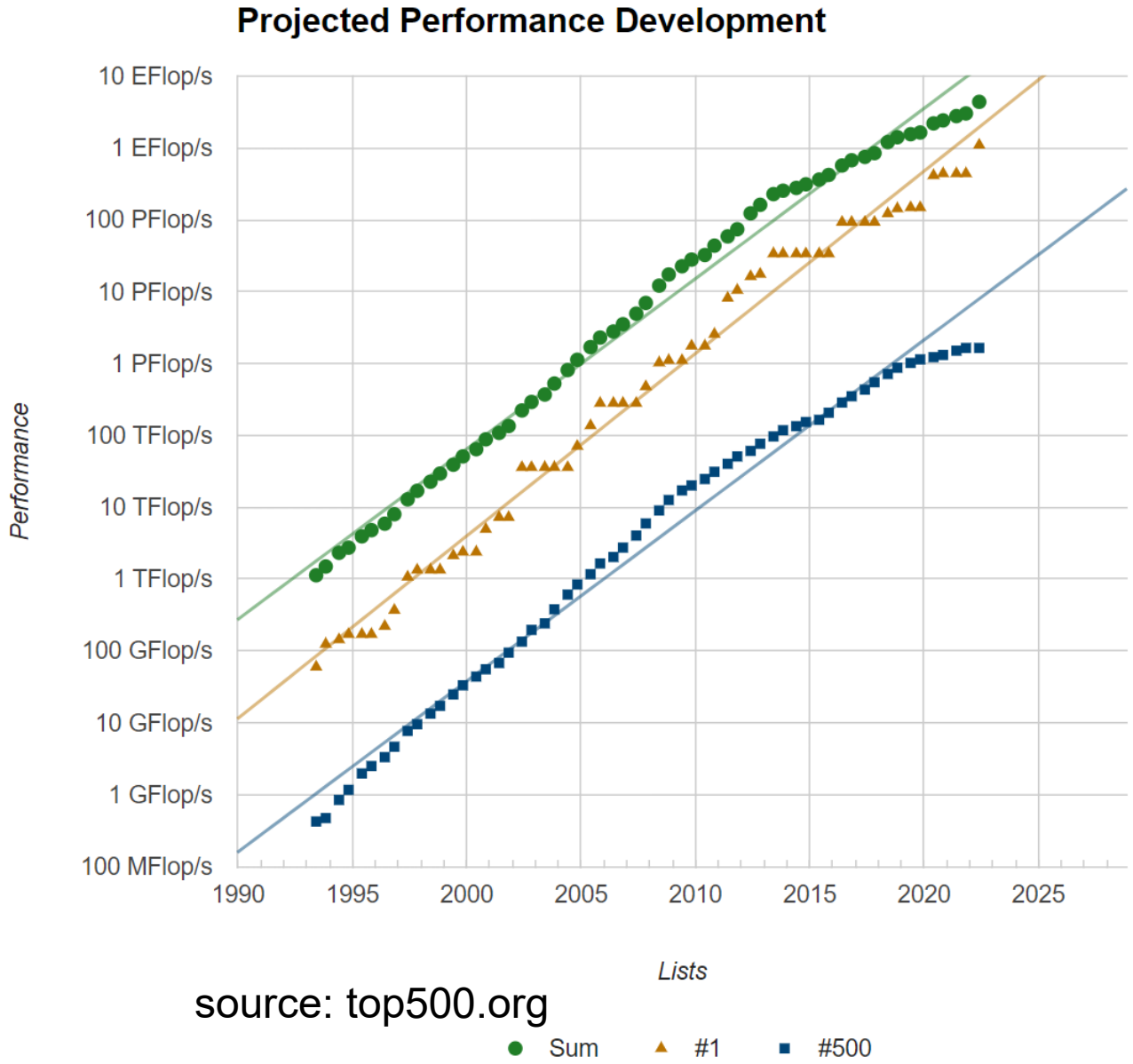
- Fujitsu's 48-core A64FX SoC (optional assistant cores)

- 158,976 nodes (CPUs):
  396 racks of 384 nodes
  36 racks of 192 nodes
- Fujitsu Tofu (Torus Fusion) interconnect





TOP500 Performance over Time



source: top500.org

name	prefix	multiplier
exa	E	10^{18}
peta	P	10^{15}
tera	T	10^{12}
giga	G	10^{9}
mega	M	10^{6}
kilo	K	10^{3}

Summary

- Serial versus parallel computing
- Parallel computing - what, why and how
- Performance and challenges
- Landscape of parallel computing

Evolution of Prominent Computers

year	location	name	first
1834	Cambridge	Difference engine	Programmable computer
1943	Bletchley	Colossus	Electronic computer
1948	Manchester	SSEM (Baby)	Stored-program computer
1951	MIT	Whirlwind 1	Real-time I/O computer
1953	Manchester	Transistor computer	Transistorized computer
1971	California	Intel 4004	Mass-market CPU & IC
1979	Cambridge	Sinclair ZX-79	Mass-market home computer
1981	New York	IBM PC	Personal computer
1987	Cambridge	Acorn A400 series	High-street RISC PC sales
1990	New York	IBM RS6000	Superscalar RISC processor
1998	California	Sun picoJAVA	Computer Based on a language

Progression of Computation Speed

year	Floating point operations per second (FLOPS)
1941	1
1945	100
1949	**1,000 (1 KiloFLOPS, kFLOPS)**
1951	10,000
1961	100,000
1964	**1,000,000 (1 MegaFLOPS, MFLOPS)**
1968	10,000,000
1975	100,000,000
1987	**1,000,000,000 (1 GigaFLOPS, GFLOPS)**
1992	10,000,000,000
1993	100,000,000,000
1997	**1,000,000,000,000 (1 TeraFLOPS, TFLOPS)**
2000	10,000,000,000,000
2007	478,000,000,000,000 (478 TFLOPS)
2009	**1,100,000,000,000,000 (1.1 PetaFLOPS)**

Computing Units

International System of Units (SI)

name	prefix	multiplier
exa	E	10^{18}
peta	P	10^{15}
tera	T	10^{12}
giga	G	10^{9}
mega	M	10^{6}
kilo	K	10^{3}
milli	m	10^{-3}
micro	μ	10^{-6}
nano	n	10^{-9}
pico	p	10^{-12}

International Electrotechnical Commission (IEC)

Name	prefix	multiplier
Exbi	Ei	2^{60}
pebi	Pi	2^{50}
tebi	Ti	2^{40}
gibi	Gi	2^{30}
mebi	Mi	2^{20}
kibi	Ki	2^{10}

Example of disk capacity:
20 Mibytes
20 MiB
20 Mebibytes
= 20 x 2^{20} = 20,971,520 bytes

Reference & Readings

- **Fugaku technical details:**
  - https://www.r-ccs.riken.jp/en/fugaku/

- Reading
  - Robert Robey, Yuliana Zamora, ”Parallel and High-Performance Computing”, Manning Publications – Chapter 1.
  - A View of the Parallel Computing Landscape, CACM Vol. 52 No. 10, pp. 56-67, October 2009. [Technical Report: The Landscape of Parallel Computing Research: A View from Berkeley, Dec 2006]