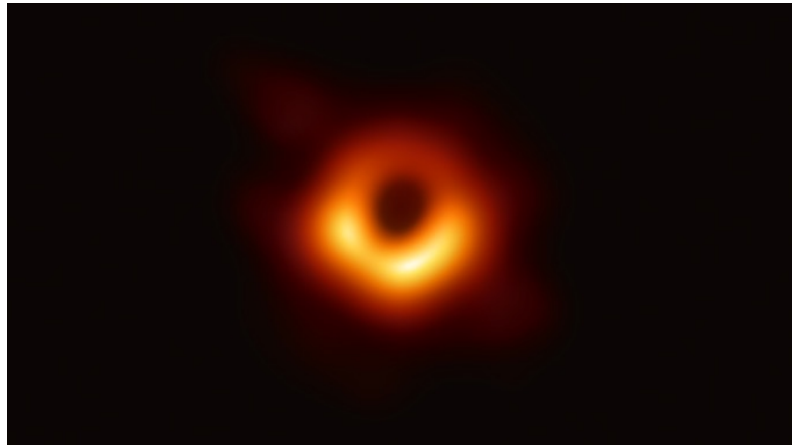
Introduction to Distributed Systems

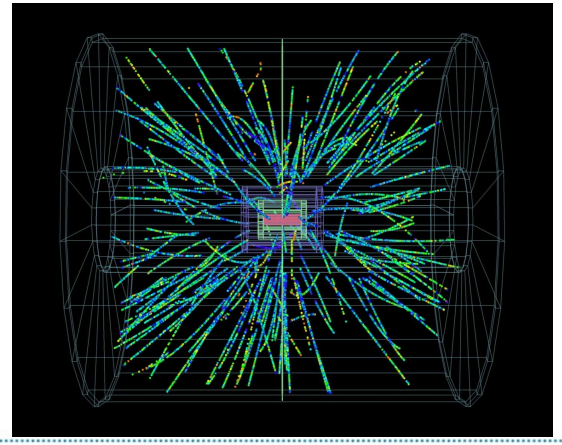
Introduction

The need for computing power

Science needs computing power

- Astronomy
  - Webb telescope
    - 57GB of data transmitted each day
  - Photo of Sagittarius A*
    - 100 million CPU hours
    - 5 PB of data to analyze (500Kgs of hard drives…)
- High energy physics, fundamental particles
  - CERN
    - 1 PB/s during experiments, 60PB/year stored
    - Large experiments manage more than 200PB
- Nuclear fission/fusion
- Earth observation, biomedical sciences
- Physics, chemistry…





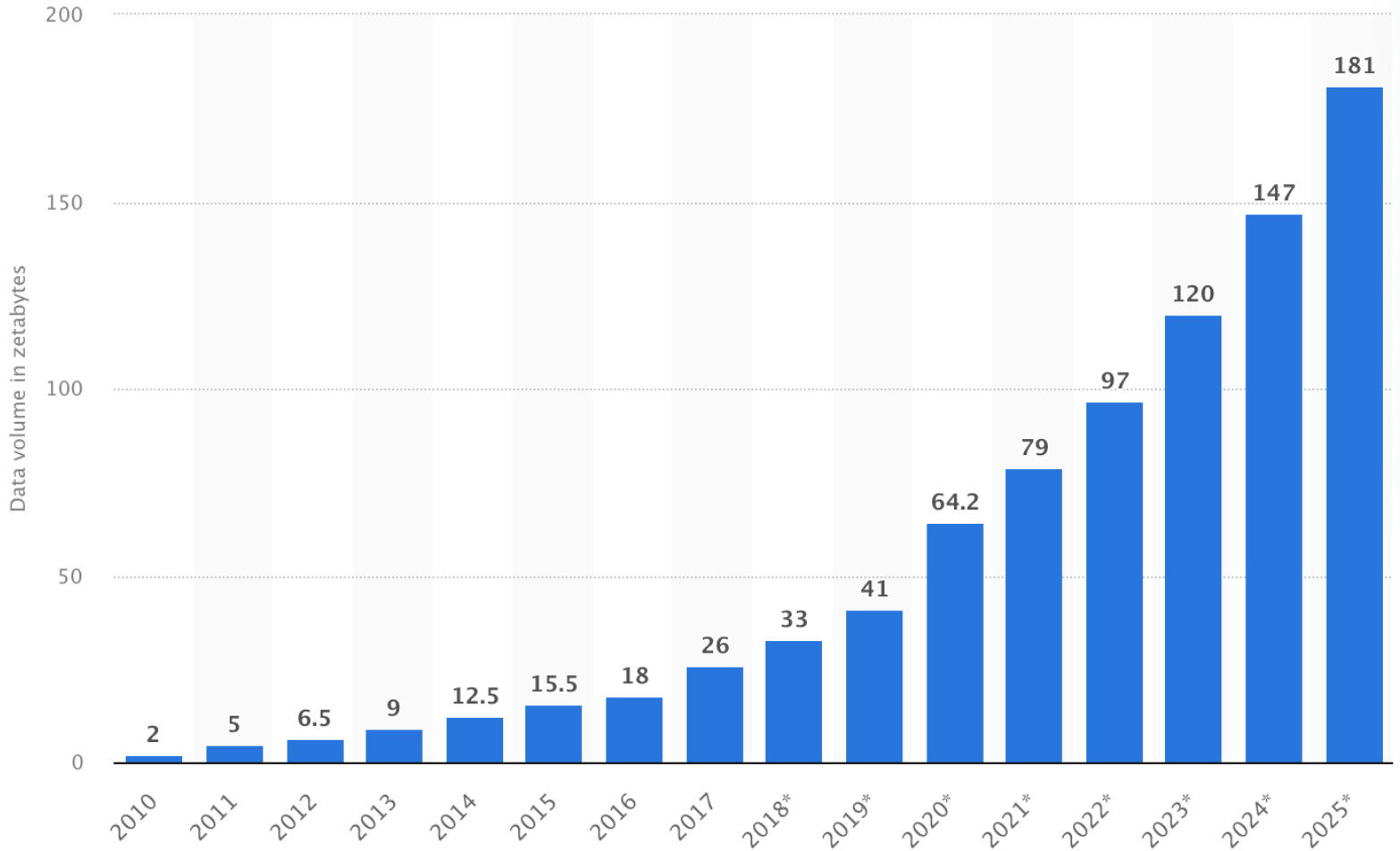
Source: https://cds.cern.ch/record/2672158/files/ATL-SOFT-SLIDE-2019-148.pdf

Some numbers

- In 2016, each person creates an estimated 1.7 MB of data/second
- In 2020
  - 404 444 users stream on Netflix every second
  - 20 tracks added per minute on Spotify
  - 500 millions of Tweets daily
  - 3.5 billions searches on Google
- GPT parameters
  - GPT-1 : 0.12 billions
  - GPT-2 : 1.50 billions
  - GPT-3 : 175 billions
  - GPT-4 : 1.74 trillion (10^12) estimated

Source : https://financesonline.com/how-much-data-is-created-every-day/

Data created, captured, copied, and consumed every year



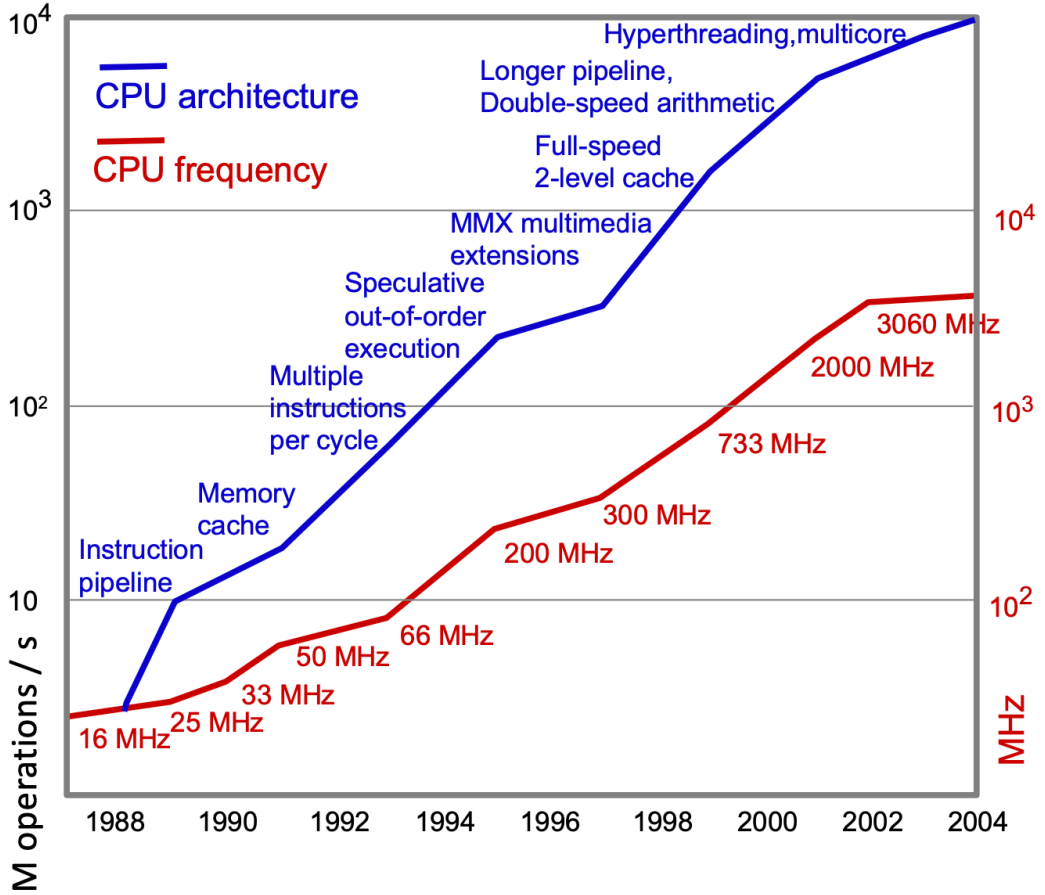
1 zettabyte =1000 exabyte
1 exabyte = 1000 petabyte
1 petabyte = 1000 terabyte

UNIVERSITE CÔTE D'AZUR

https://statista.com/statistics/871513/worldwide-data-created/

How to improve computer performance

- Features vs Frequency
  - Focus on adding features to the CPU
    - New cache technologies/algorithms
    - OOO execution
  - Or keep it simple but focus on frequency
- Aka “brainiacs” vs “demon speed”



Introduction

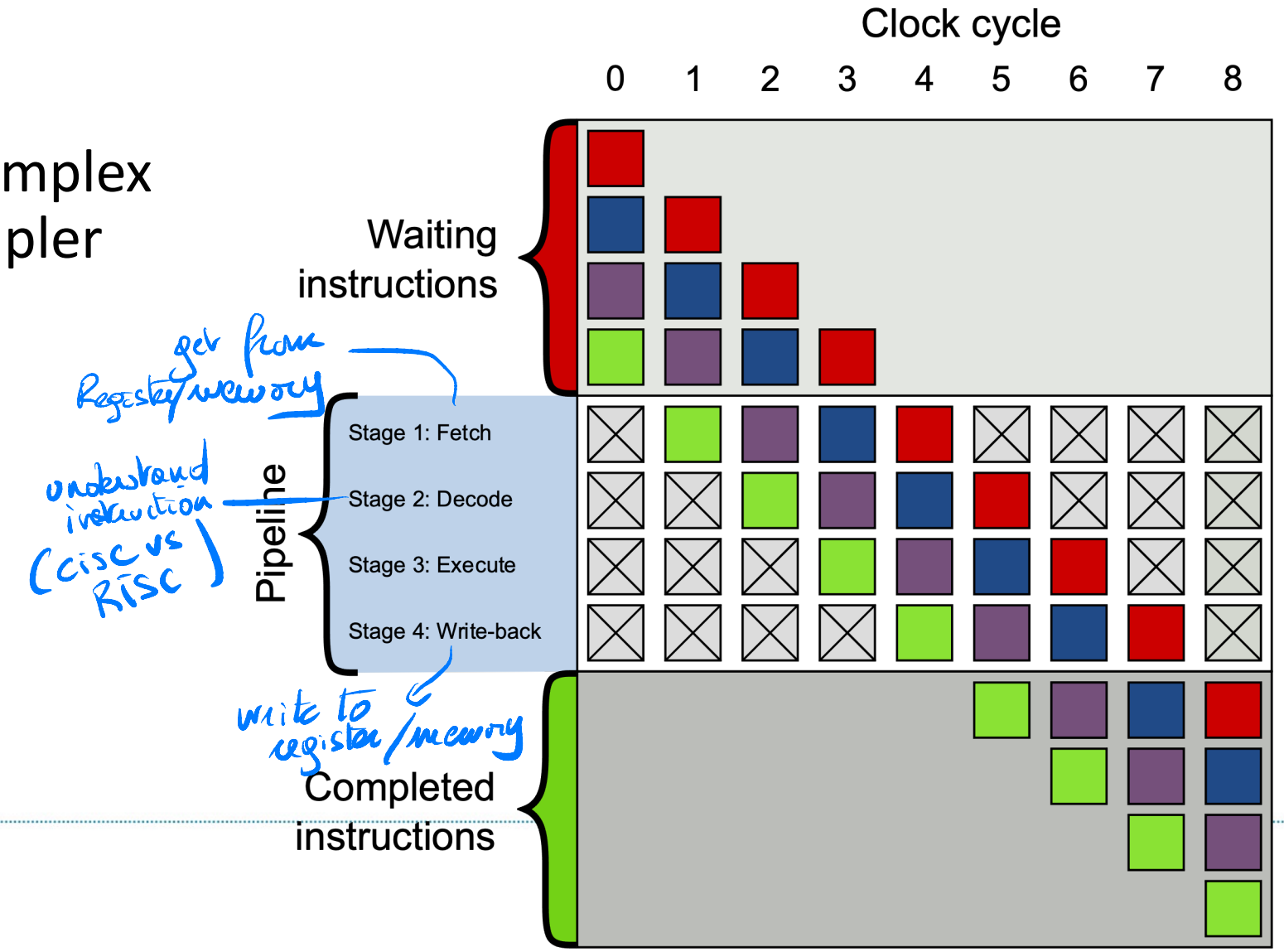
The race for performance, micro-architecture

Performance features in modern processors

- Objective
  - Improve IPC (Instructions Per Cycle) or reduce CPI (Cycles Per Instruction)
- IPC
  - How many instructions can you process in 1 time unit (usually cycle)
- CPI
  - How many cycles (or time unit) do you need to process one instruction
- IPC = 1/CPI if same time unit

Pipeline

- Split execution of complex instructions into simpler independent steps
  - Improves IPC



https://en.wikipedia.org/wiki/Branch_predictor

Example

	Microarchitecture	Pipeline stages
1993	P5 (Pentium)	5
1995	P6 (Pentium Pro)	14
1999	P6 (Pentium 3)	10
2000	NetBurst (P4, Willamette)	20
2002	NetBurst (P4, Northwood)	20
2004	NetBurst (P4, Prescott)	31
2006	NetBurst (Cedar Mill)	31
2006	Core	12-14
2008	Bonnell	16-20
2011	Sandy Bridge	14-16
2013	Silvermont	14-17
2013	Haswell 14	14
2015	Skylake 14	14
2020	Tremont	20
2022	Raptor Cove (P core)	12

Limits to pipeline

- Assumptions
  - An instruction can be divided into arbitrary small steps
    - Not true
  - No overhead between stages
    - Not true
- Structural hazard :
  - Different instructions in pipeline require same resource
  - Ex: 2 or more instructions need ALU
  - Solution : multiply resources

Limits to pipeline

- Data hazard :
  - Different instructions in pipeline work on same data
  - Example
    - i1 : R2 <- R1+R3
    - I2 : R4 <-R2*2 #what happens if R1 and R2 in pipeline ?
  - Solutions :
    - Bubbling : add NOP in pipeline
    - Out of Order Execution : reorganize instructions to avoid data hazard

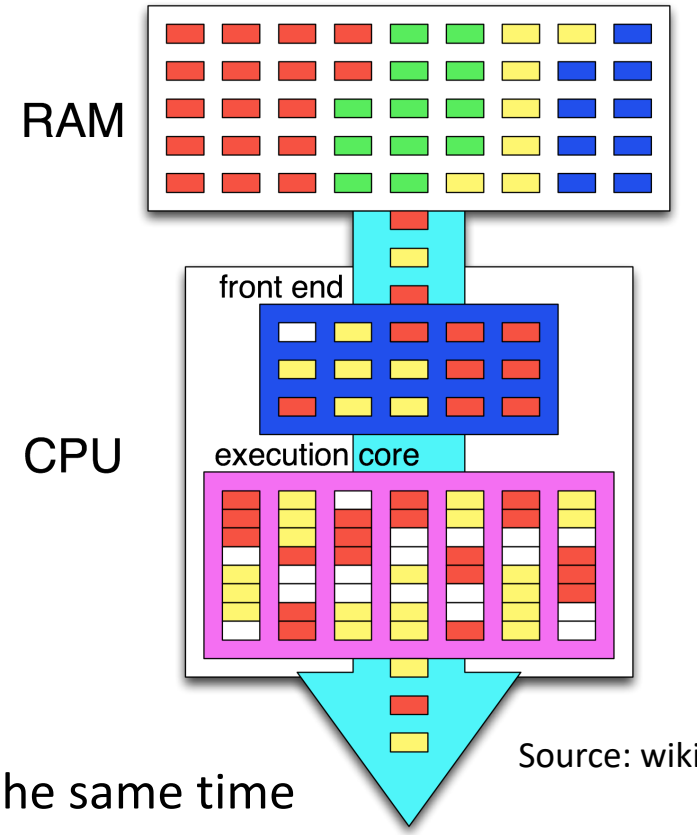
Limits to pipeline

- Control hazard
  - Next instruction in pipeline depends on result of an instruction in pipeline
  - Example
    - i1 : R2 <- R1+R3
    - i2 : bneq R2, 0, i1 # if R2 !=0 go to i1
    - i3 : …
  - After i2, do we put i1 again or i3 in pipeline ?
  - Solutions
    - Bubbling
    - OOO
    - Branch prediction
      - Make a guess, load most likely instruction
        - If right, it's all good, if wrong, there's a penalty
      - > 95% success in modern processors

Speculative execution

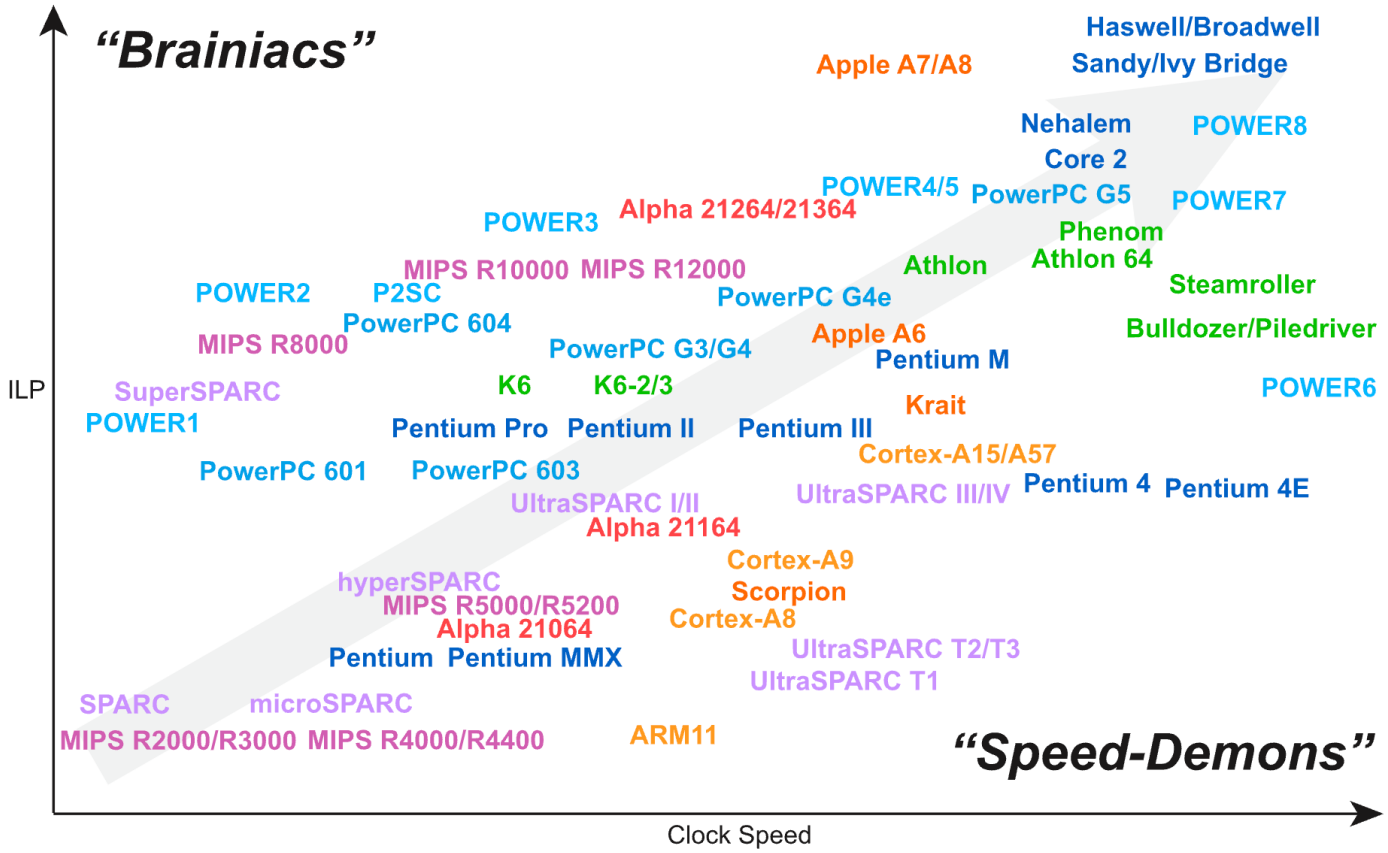
Other features

- Better caching
  - Cache == small, high speed memory
  - Caching hierarchy (L1-L2-L3)
  - More on that later
- Superscalar architecture
  - Add parallelism inside processor
    - Allow for multiple instructions to be executed at the same time
  - Can be implemented with multiple pipelines and hyperthreading
- These are parts of the “brainiacs” CPU



Source: wikipedia

Brainiacs vs Speed-Demons



UNIVERSITÉ CÔTE D'AZUR

ILP = Instruction level parallelism

Source : https://www.lighterra.com/papers/modernmicroprocessors/

Introduction

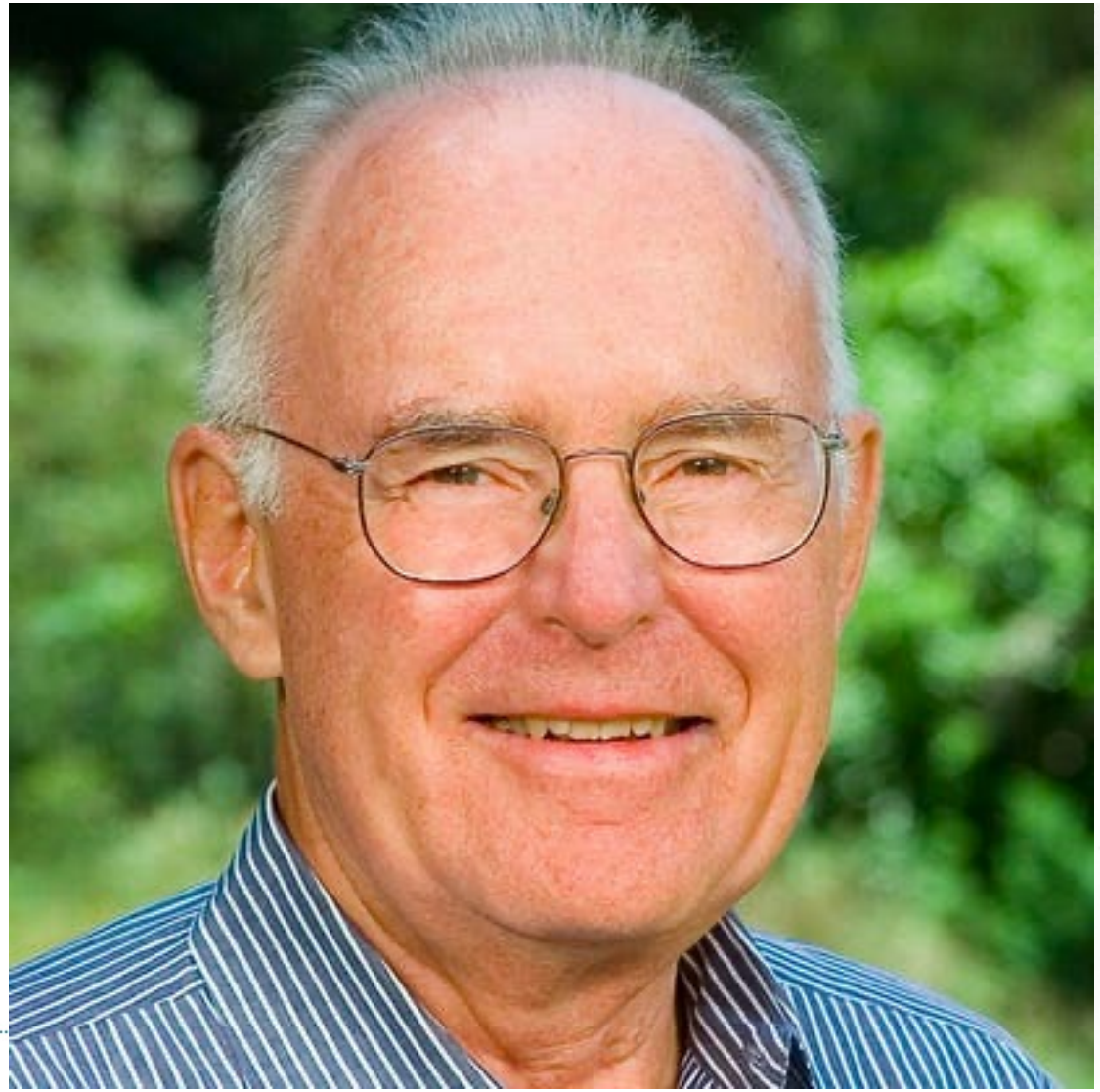
The race for performance, the big picture

Moore’s law

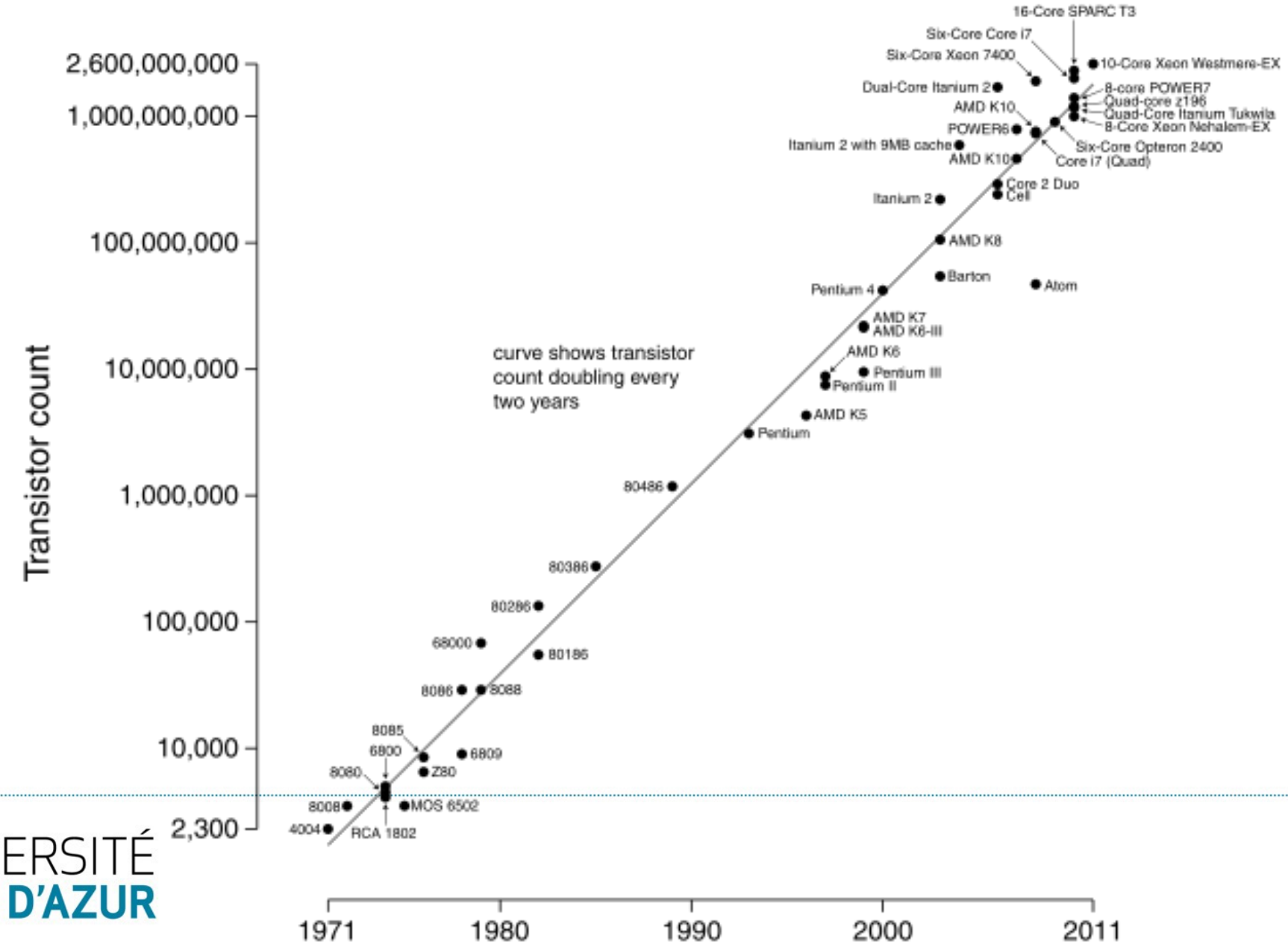
- Gordon Moore (1975):
  - Every 24 months, the number of transistors in a CPU can be doubled
- Number of transistors approximates performance
  - Not really correct but good enough approximation
- If you want your program to run faster, just wait for next generation
  - No work, it’s free!

Moore’s law

- Gordon Moore (1975):
  - Every 24 months, the number of transistors in a CPU can be doubled
- Number of transistors approximates performance
  - Not really correct but good enough approximation
- If you want your program to run faster, just wait for next generation
  - No work, it’s free!



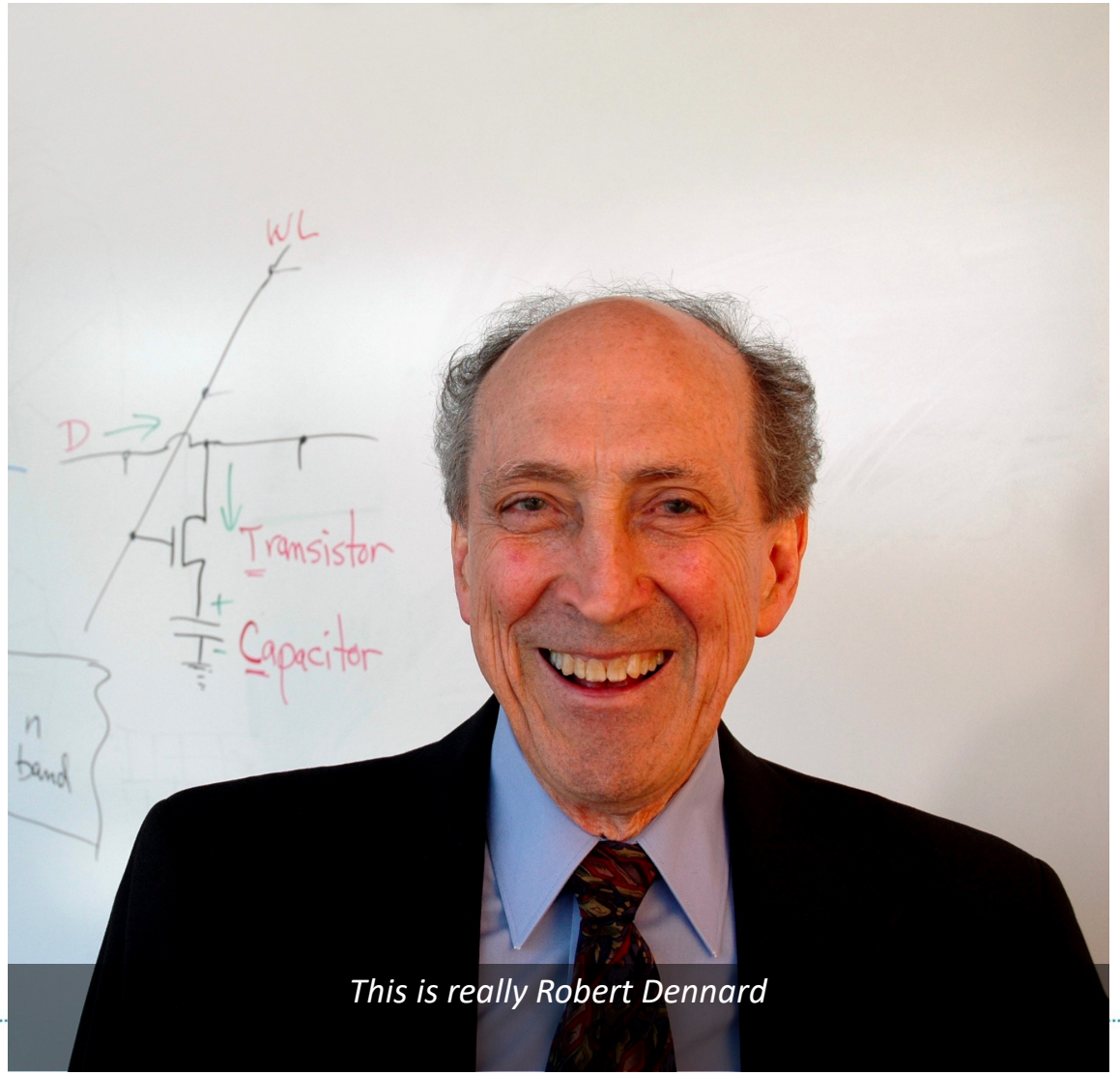
Microprocessor Transistor Counts 1971-2011 & Moore's Law



UNIVERSITÉ CÔTE D'AZUR

Dennard scaling

- If you reduce the size of a transistor, voltage and current go down
- So each time you double the number of transistors
  - Frequency can go up
  - Energy can go down



This is really Robert Dennard

UNIVERSITÉ CÔTE D'AZUR

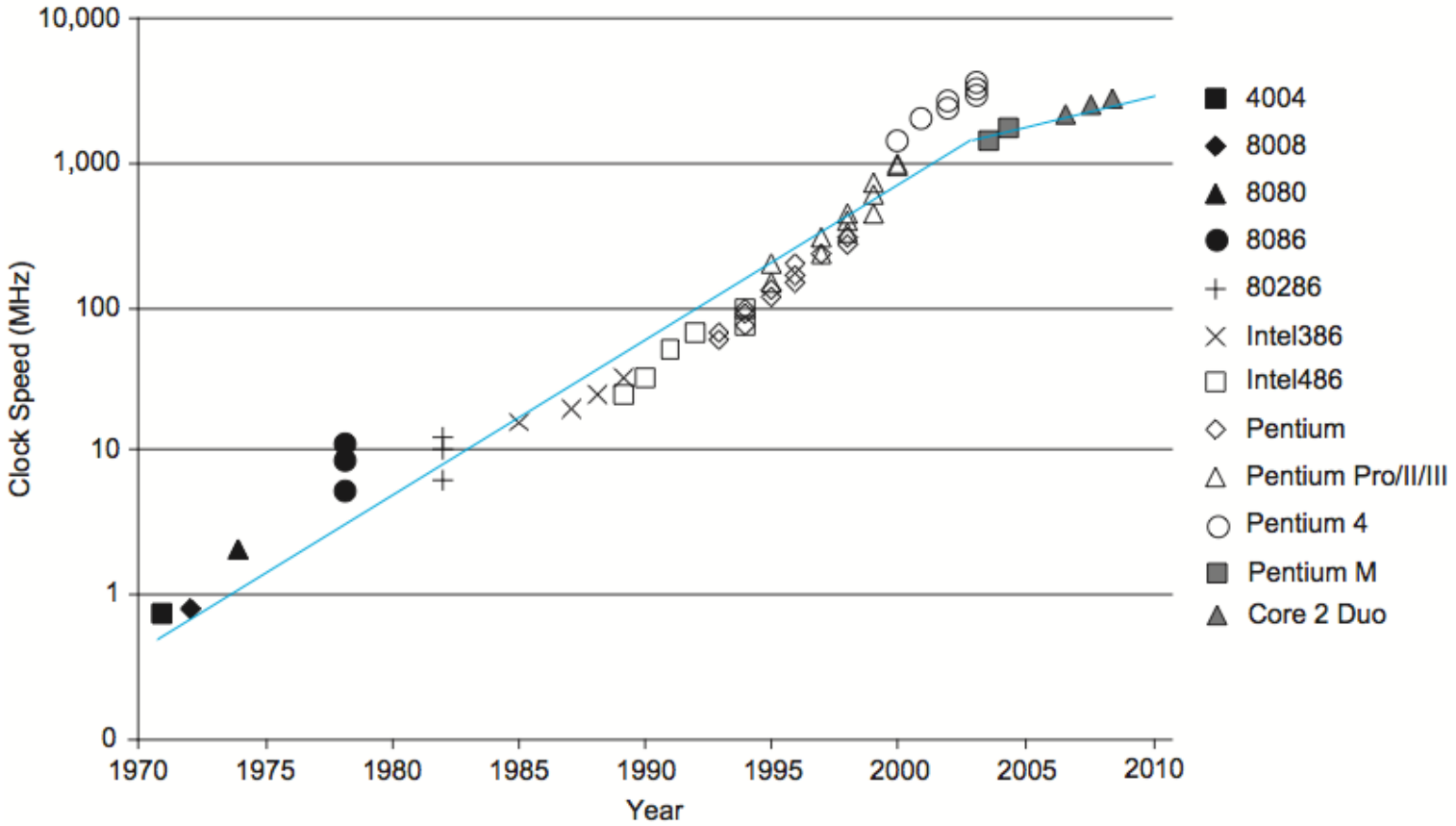
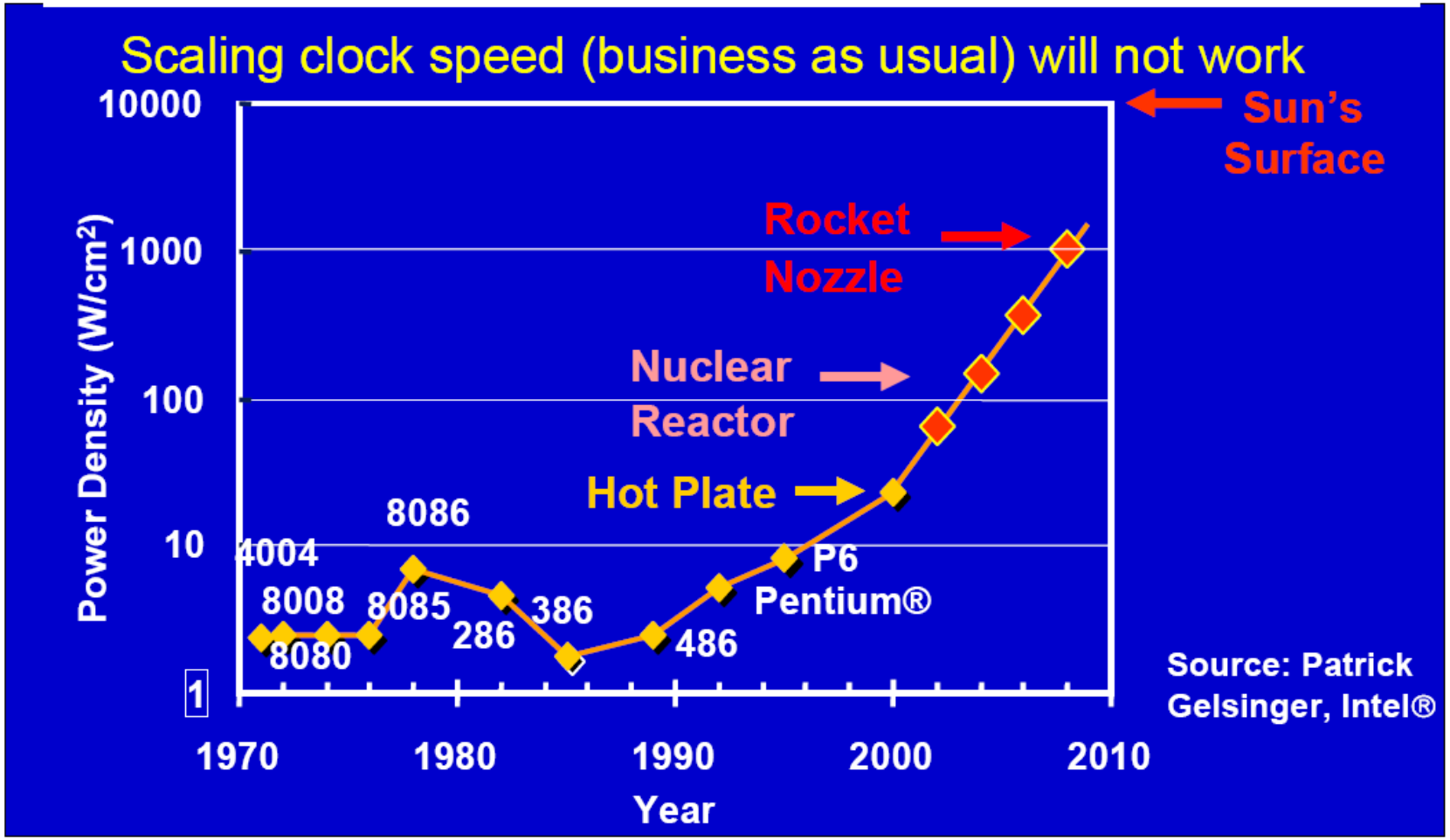


FIGURE 1.5 Clock frequencies of Intel microprocessors

From : **CMOS VLSI Design,** Weste and Harris.

Limits to Dennard's Law



Source, cours CS194 U.Berkeley

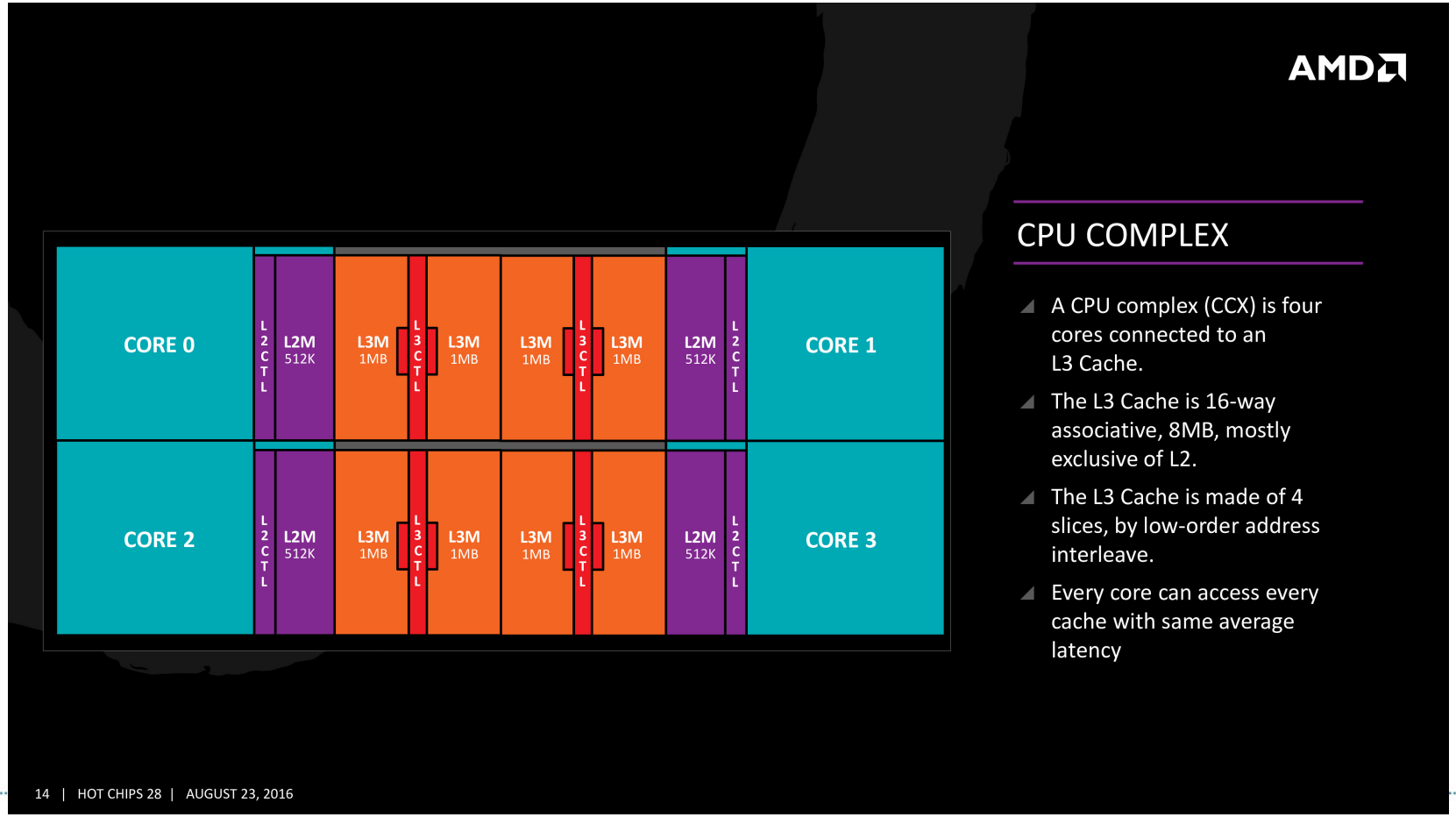
Limits to Moore's Law

- Moore's Law is still valid
  - But no more free lunch
- What went wrong?
  - Physics did
- Limits on single-core
  - Smaller transistors
  - Or larger die
- Need to pack more CPUs together
  - Multi-core

Multi-core CPUs

- Many core on the same die
  - Share some caches
  - Have a faster (direct) access to some memory region
    - Still a shared memory system
    - NUMA : Non Uniform Memory Access
- Cores can be simple or complex
  - If complex cores: each core can execute arbitrary code
  - If simple cores : all cores must execute the same code

AMD Ryzen 1 & 2



- Cores are grouped by 4 in a CCX
- Two CCX are grouped a CCD (Core Chiplet Die)
- Very modular architecture, increase CCD to have more cores
- Overall a MIMD

UNIVERSITÉ CÔTE D'AZUR

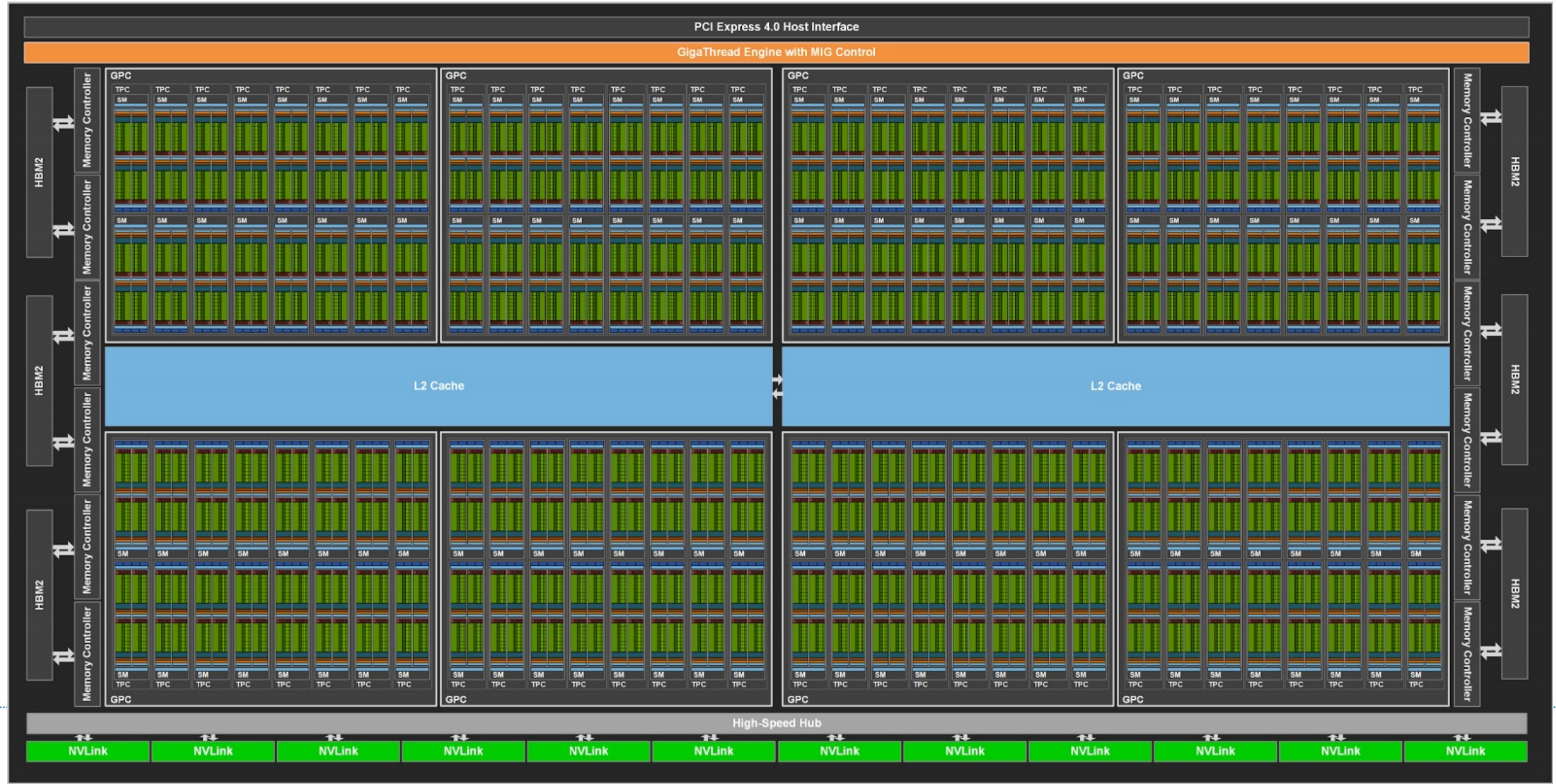
GPU

- Graphics Processing Unit (GPGPU)
  - Based on graphical processors
  - Very good at floating point computation (and Integer now)
  - Massively parallel architecture
- Hybrid execution model
  - MIMD & SIMD
  - A GPU can execute multiple programs (aka kernels).
  - Each kernel is a SIMD code executed by many different cores
- Hierarchical architecture (NVIDIA)
  - GPU Processing Clusters
    - Streaming Multiprocessors
      - CUDA Cores

- NVIDIA Ampere (GA100)
  - 128 Streaming Multiprocessors
  - 8192 Cuda Cores
  - 512 Tensor Cores

UNIVERSITÉ CÔTE D'AZUR

https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf



COTE D'AZUR

https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf

Is it enough ?

- We only addressed CPUs
  - Multicore is limited
    - Max 64 x86 core (AMD Epyc)
    - 18k GPU Core (NVIDIA Hopper)
- What about the rest?
  - RAM : Up to 12TB on SuperMicro server board (X11QPL) for roughly 72k€ (2023)
  - Disks : Unlimited with expansion bays
- But still limited and expensive

Distributed Systems

Introduction

UNIVERSITÉ CÔTE D'AZUR

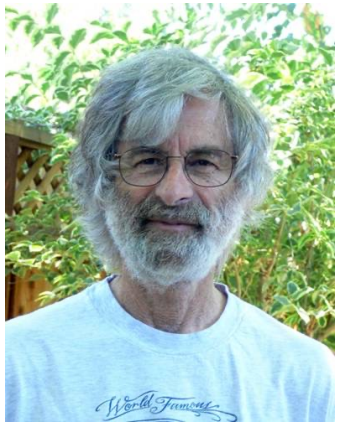
Rational

- If one machine is not enough
  - Use many!
- If one machine is critical
  - Add some for redundancy, fault tolerance
- But introduce new constraints
  - No shared memory (each machine has its own memory)
  - Need for a communication layer

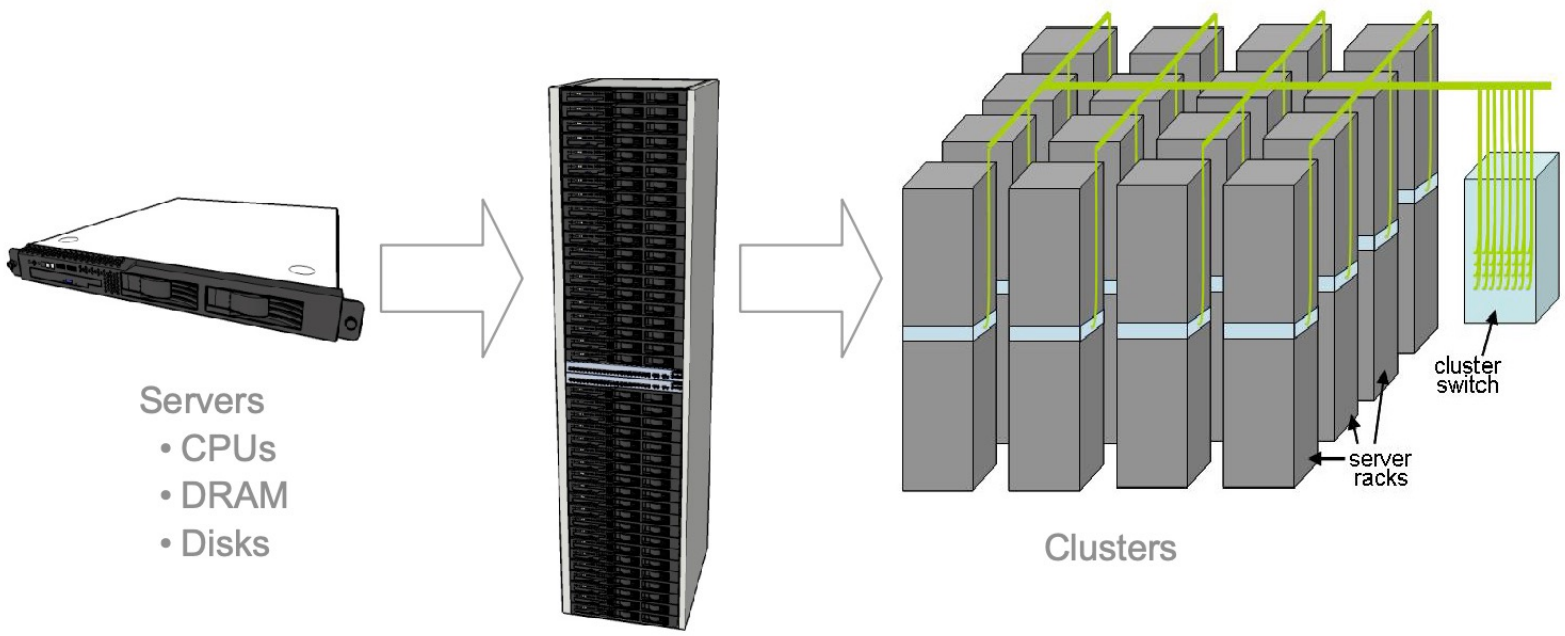
Definitions

- *A distributed system is a collection of independent computers that appears to its users as a single coherent system (Tanenbaum)*

- *A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable (Lamport)*



Architecture



UNIVERSITÉ CÔTE D'AZUR

Source : Jeff Dean, Designs, Lessons and Advice from Building Large Distributed Systems

MareNostrum 4 (Spain) :

- 384.75 TB of RAM
- 3 456 nodes
- 680 960 cores

Frontier (USA) :

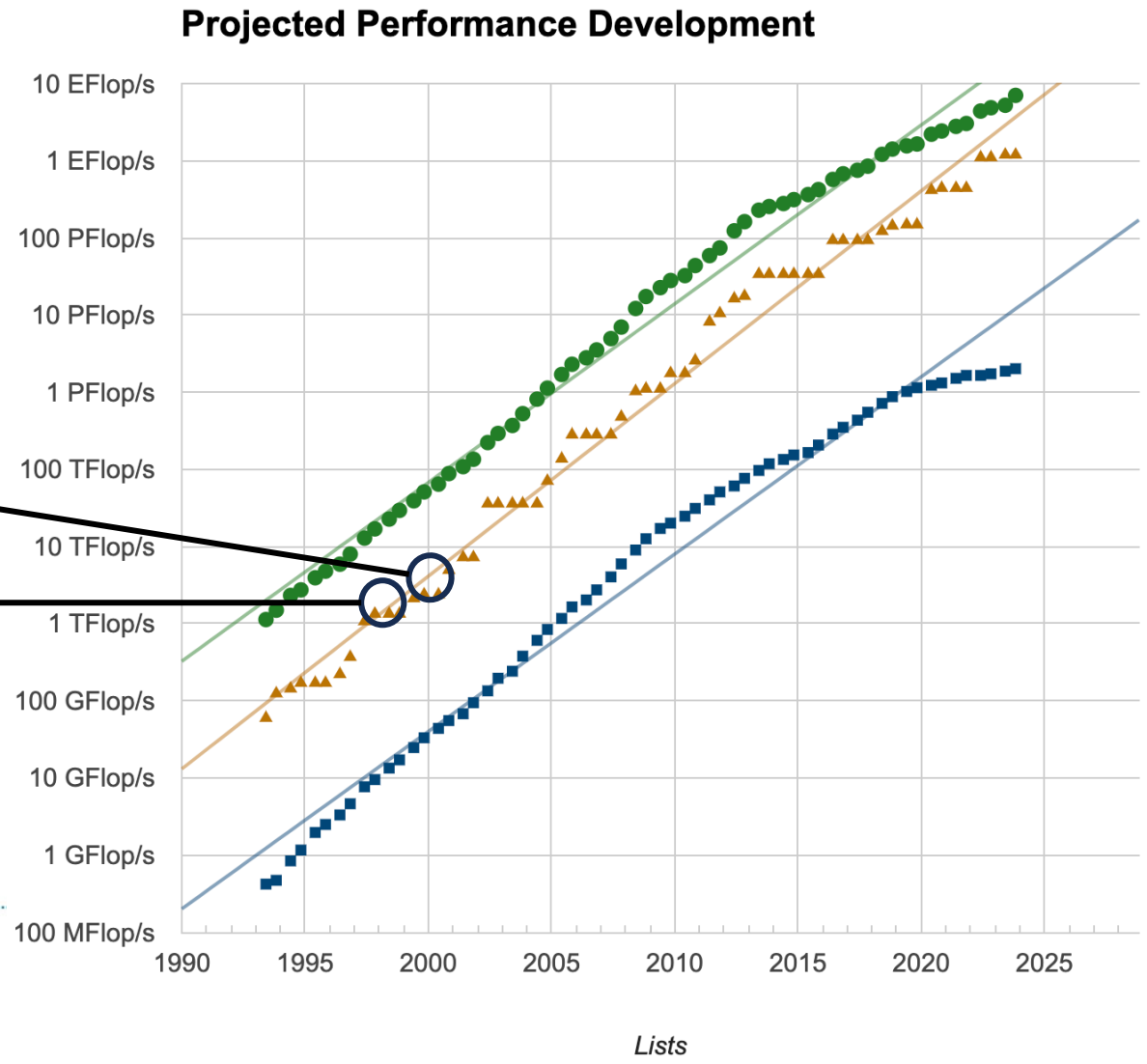
- 4.6 PB of RAM
- 9 472 nodes
- 606 208 CPU cores
- 8 335 360 GPU cores

Source : top500.org

Performance of top 500 machines

Qualcomm Adreno 740 (Samsung S23 ultra)
3681.3 GFLOPS

Apple A16 Bionic (iPhone 15)
1789.4 GFLOPS



UNIVERSITÉ CÔTE D'AZUR

Concepts of Distributed Systems

Fundamental concepts

UNIVERSITÉ
CÔTE D'AZUR

Transparency

- Hide from the user/app developer the separation of components
  - Make the system looks like One instead of many
- 8 forms of transparency (ISO Reference Model of Open Distributed Processing)
  - Access transparency : local and remote resources are accessed using identical operations
  - Location transparency : access to resources does not require knowledge of their physical/network location
  - Concurrency transparency : several processes operate concurrently using shared resources without interferences

Transparency - 2

- Replication transparency : multiples instances of resources can be used without knowledge of the replicas
- Failure transparency : hides faults, allowing users to complete their tasks despite failures
- Mobility transparency : allows the movement of resources and clients without affecting operation
- Performance transparency : reconfigure the system to improve performance as loads vary
- Scaling transparency : allows the system and app. to scale without change to the system structure or application algorithms.

Transparency - 3

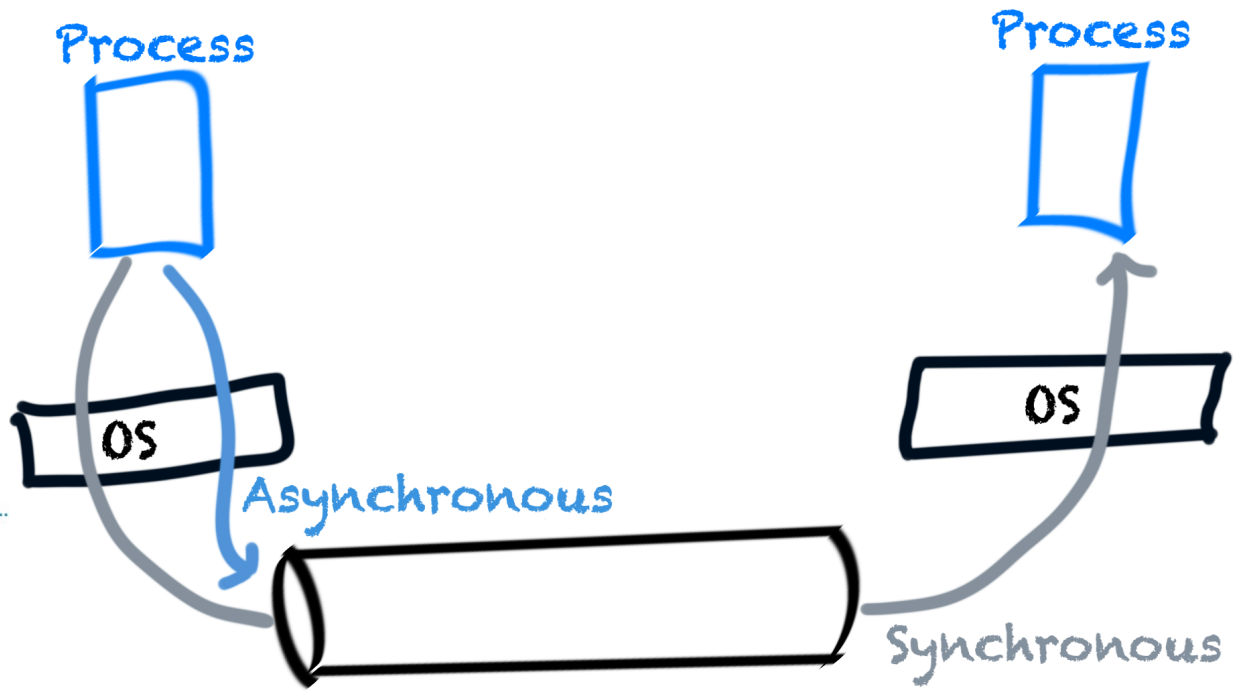
- Not all properties must exist in a distributed system
  - Design choices constrain them
- Access + location transparency => network transparency
- Example of network transparency : email
  - Email address : name@something.com
- Example of mobile transparency : mobile phones
  - Moving while on call

Communication

- Low level : sockets
- End points for sending/receiving data
  - FIFO, handle bytes
- Needs IP address and port of remote machine
  - Lack of transparency
- Sending complex data structures requires transformation
  - Serialization/Deserialization
- High level : messages, Remote Procedure Call (RPC)

Synchronous vs Asynchronous communications

- Synchronous
  - Sender is blocked until message is processed by receiver
- Asynchronous
  - Sender is blocked during sending
  - Usually relies on a message queue



Latency

- Any communication takes time
  - Base latency + bandwidth from network
  - Multiple software layers
- Avoid them if possible
  - But latency does not come only from network
  - Even memory access adds latency

Latency Comparison Numbers

```
L1 cache reference ......................... 0.5 ns
Branch mispredict ............................ 5 ns
L2 cache reference ........................... 7 ns
Mutex lock/unlock ........................... 25 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

1 ns = 10^{-9} s

https://gist.github.com/hellerbarde/2843375

Latency

- Real value * 1 000 000

https://gist.github.com/hellerbarde/2843375

UNIVERSITÉ CÔTE D'AZUR

Minute:

L1 cache reference	0.5 s	One heart beat (0.5 s)
Branch mispredict	5 s	Yawn
L2 cache reference	7 s	Long yawn
Mutex lock/unlock	25 s	Making a coffee

Hour:

Main memory reference	100 s	Brushing your teeth
Compress 1K bytes with Zippy	50 min	One episode of a TV show (including ad breaks)

Day:

Send 2K bytes over 1 Gbps network	5.5 hr	From lunch to end of work day

Week

SSD random read	1.7 days	A normal weekend
Read 1 MB sequentially from memory	2.9 days	A long weekend
Round trip within same datacenter	5.8 days	A medium vacation
Read 1 MB sequentially from SSD	11.6 days	Waiting for almost 2 weeks for a delivery

Year

Disk seek	16.5 weeks	A semester in university
Read 1 MB sequentially from disk	7.8 months	Almost producing a new human being
The above 2 together	1 year	

Decade

Send packet CA->Netherlands->CA	4.8 years	Average time it takes to complete a bachelor's degree

Physical clocks

- Each node (computer) has its own clock
  - Useful for local time-sensitive operations
    - E.g : sleep 200ms before doing something
- No computer clock is perfect
  - Drift from the perfect time
- Clock drift rate :
  - Rate at which a computer clock deviates from a perfect reference clock
- So even if clocks are set at a given time, they will deviate
- In a distributed system, clocks will drift differently on each node

Physical clocks - 2

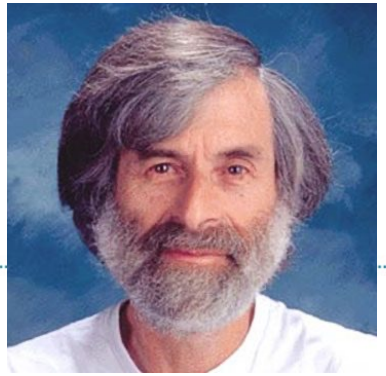
- But what is time anyway?
  - Ask a philosopher and maybe get an answer or…
  - Be pragmatic an ask a physicist
- International Atomic Time (TAI)
  - Average time of 450 atomic clocks in 80 national laboratories around the world
  - 9,192,631,770 transitions of caesium 133 atom
- Coordinated Universal Time (UTC)
  - Add (or delete) leap seconds to TAI
  - UTC signals are broadcasted from
    - land-based for milliseconds precisions
    - satellites for sub-ms

Physical time in distributed systems

- How to maintain global physical time
  - Each computer clock will drift
  - Drift will vary from computer to another
- Need to synchronize clocks regularly
  - Depends on your need
- External synchronization
  - Need to correlate events in your system with external events (accounting…)
  - Ensure clocks don’t deviate more than *D* compared to an external source
- Internal synchronization
  - Need to synchronize processes inside our systems only
  - Ensure any 2 computer clocks don’t deviate more than *D*
- External sync of *D* => internal sync of *2D*

Logical clocks

- Physical time in Distributed Systems is hard
- Do we really need it?
  - Often we don't
  - What is important is some form of internal consistency
  - E.g. : I don't care what time you woke up, I care you arrived *before* the start of the lecture
- We need *Logical clocks*
- Relation "happen-before"
  - Lamport (*Turing Award* 2013)



Synchronous vs Asynchronous Distributed Systems

- Two models to take into account time and latency
  - Strong vs No assumptions about time
- Synchronous Distributed Systems
  - Time to execute each step of a process has known lower and upper bound
  - Any message transmitted over a channel is received within a known bound
  - Each process has a local clock with knows drift rate from real time
- Asynchronous Distributed Systems
  - No bounds on process execution speed, message transmission delays, clock drift rates.
- Most real world systems are asynchronous
  - Bounds are estimated, but not guaranteed
  - But assuming system is synchronous is useful sometimes (timeout…)

Concepts of Distributed Systems

Failures

UNIVERSITÉ CÔTE D'AZUR

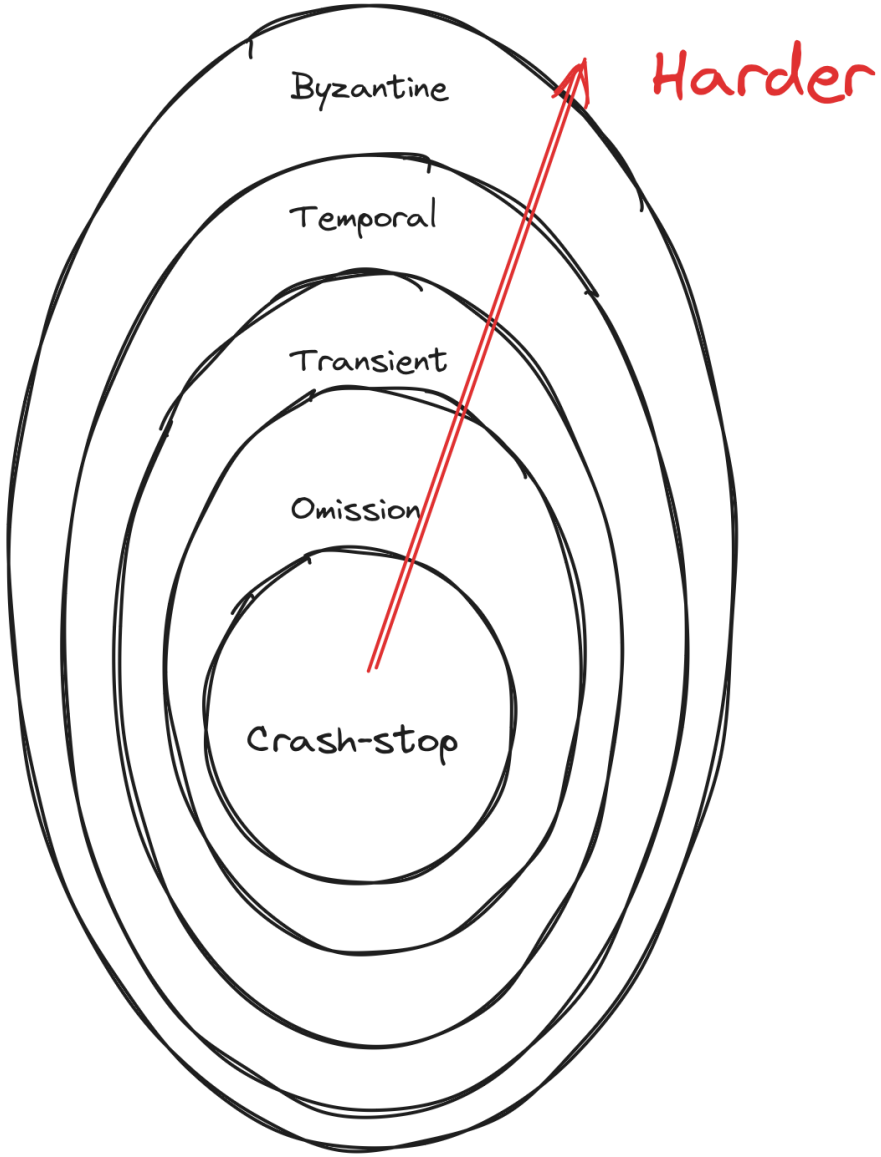
What are failures?

- Obviously it’s when something stop working
  - But how and why is important ?
- Many types of failure in distributed systems, some are
  - Crash-stop
  - Omission
  - Transient failure
  - Temporal failure
  - Byzantine failure

Failures

- Crash-stop
  - Processes/nodes crash and **never** comes back
- Omission
  - Message is lost in transit
- Transient
  - Arbitrary perturbation of the hardware (power surge, cosmic ray…) or the software (aka Heisenbugs)
- Temporal
  - Inability to meet deadline
- Byzantine
  - Processes/nodes/network exhibit arbitrary behavior, including nefarious behavior

Dealing with failures



Typical first year for a new google cluster

- ~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
- ~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
- ~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
- ~1 network rewiring (rolling ~5% of machines down over 2-day span)
- ~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
- ~5 racks go wonky (40-80 machines see 50% packetloss)
- ~8 network maintenances (4 might cause ~30-minute random connectivity losses)
- ~12 router reloads (takes out DNS and external vips for a couple minutes)
- ~3 router failures (have to immediately pull traffic for an hour)
- ~dozens of minor 30-second blips for dns
- ~1000 individual machine failures
- ~thousands of hard drive failures
- slow disks, bad memory, misconfigured machines, flaky machines, etc.
- Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44922.pdf

High Availability

- System can execute without interruption for long period of time
- Typically measured as %
  - 100%, 99%, 99.9%, 99.99%
- Very hard to maintain high availability

Availability	99%	99.9%	99.99%
Daily	14min24s	1m 26.4s	8.6s
Weekly	1hr 40m 48s	10m 4.8s	1m 0.5s
Monthly	7hr 18m 17.5s	43m 49.7s	4m 23s
Yearly	3d 15hr 39m 29.5s	8h 45m 57s	52m 35.7s

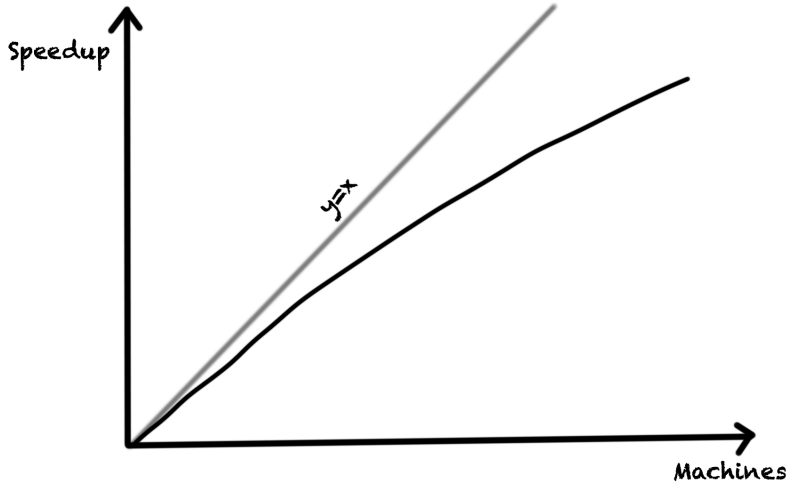
Concepts of Distributed Systems

Scalability

UNIVERSITÉ CÔTE D'AZUR

Speedup

- Measure of the relative performance
  - Sequential vs parallel (multiple machines)
- Example
  - Ts = 200 seconds
  - Tp = 100 seconds on 2 machines
  - Speedup = 2 (i.e the parallel version is twice as fast as the sequential one)
- Speedup varies
  - With the problem
  - With the number of resources



Scalability

- Capacity of a system to manage varying load through adaptation
  - Add (resp. remove) resources when load increases (resp. decreases)
- Manual scaling
  - Decided by the user or administrator
  - Could be based on expected load (e.g. time of day)
  - Slow, error prone
- Auto scaling
  - The application (framework) decides based on measured metrics
  - Fast but can have unexpected results

Vertical elasticity

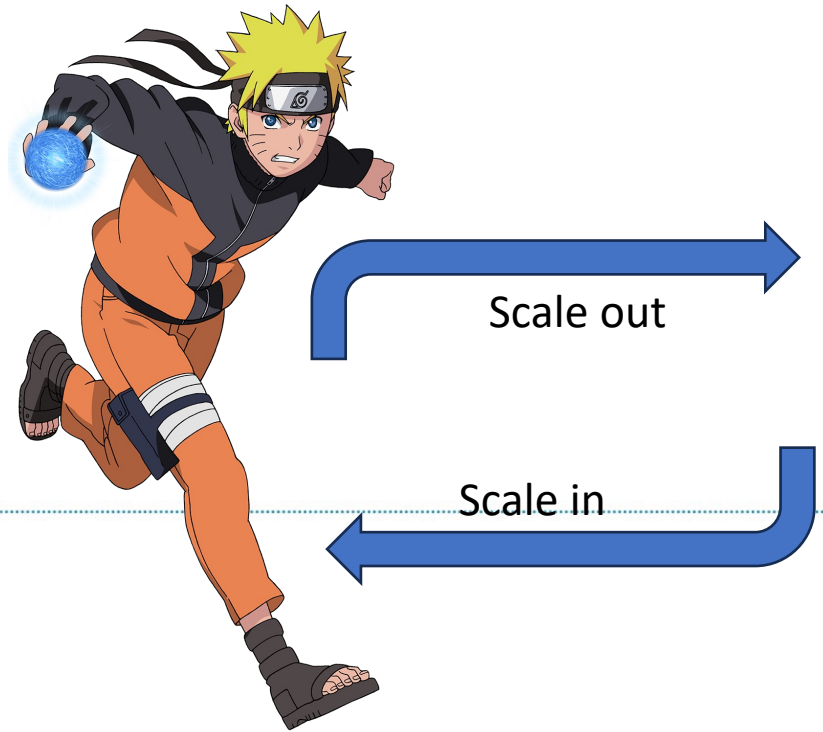
- Add/remove resources in a single instance
- Example
  - Add memory, upgrade CPU…
- Hard to do quickly and limited
- But fully transparent for the application

Scale up

Scale down

Horizontal elasticity

- Add/remove instances as needed
- Fast and potentially unlimited scaling
- But need application support



Scale out

Scale in

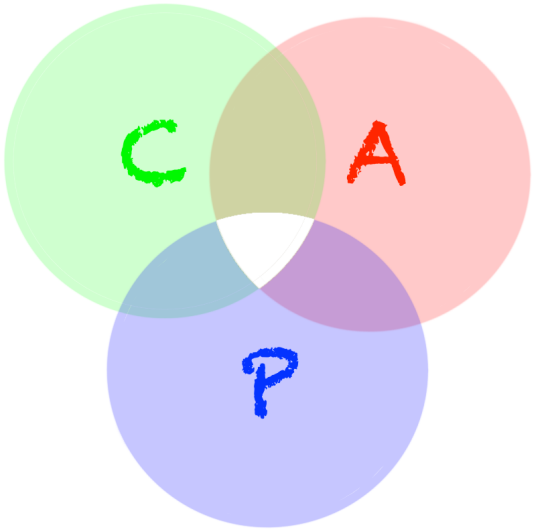
Limitations of distributed systems

The CAP theorem

UNIVERSITÉ CÔTE D'AZUR

CAP theorem

- Theorem from Eric Brewer (2000)
- A distributed system with read/write cannot provide more than 2 of the following guarantees
  - Consistency
  - Availability
  - Partition Tolerance



CAP theorem

- Consistency : a read returns the most recent write
- Availability : every request to any node returns a result (not necessarily correct)
- Partition Tolerance : the system still works in presence of network failures.
- In practice, the network is NOT reliable
  - Partitions WILL occur

CAP theorem

- So CAP theorem can be rephrased
  - In presence of network failure, you have to choose between Availability and Consistency
- A-P : When there is a partition, the system continues to work but might answer outdated/inconsistent results.
- C-P : When there is partition, some nodes might stop answering.

CAP in the wild

- A distributed DB on a single cluster
  - C-A
  - Refuses writes if network issue or node down
- Domain Name Server
  - A – P
  - Might answer incorrect address
- A multi-site distributed DB or Quorum protocols (e.g. Paxos)
  - C - P

Limitations of distributed systems

The FLP theorem

UNIVERSITÉ CÔTE D'AZUR

The agreement problem

- Basically, ensure that all machines agree on some value
  - Use of a specific protocol
- General problem with variations
  - Byzantine agreement : a single node has the initial value
  - Consensus problem : all nodes have an initial value
- 3 important properties
  - Termination (liveness) : All nodes (that have not failed) eventually decide.
  - Agreement (safety) : All non faulty nodes agree on the same value
  - Integrity (validity) : if all working nodes propose the same value, they will agree on this value.

The FLP Theorem

- Fischer, Lynch and Paterson, 1985
- In an asynchronous distributed system, there does not exist a determinist algorithm for the consensus problem
  - You cannot have all 3 properties
- But only in asynchronous model
  - In synchronous, it’s possible
- So in practice
  - Don’t forget the async model is very strict.
  - Relax some constraints with timeout, re-sending of messages…
- Real world algorithms
  - Paxos (Lamport and Leslie, 1998)
  - Raft (Ongaro and Ousterhout, 2014)

Limitations of distributed systems

The 8 distributed system fallacies

UNIVERSITÉ CÔTE D'AZUR

The ~~7~~ 8 fallacies

- 7 by Peter Deutsch, 1994
  - Assumptions engineers are likely to make when designing distributed systems
  - Wrong in the long run
- 1 more by James Gosling, 1997
- They impact infrastructure and software design

The 8 fallacies

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

The 8 fallacies

1. The network is reliable.
   - Power, wire, wireless issues…
   - Infrastructure : redundancy is necessary
   - Software : messages will be lost, so add a reliable communication or design around it
2. Latency is zero.
   - Latency varies greatly from LAN to WAN. Cannot be reduced to 0 due to speed of light
   - Software : reduce number of calls, transfer big chunk of data at conce
3. Bandwidth is infinite.
   - It’s not infinite, bit it grows steadily (40Gb/s network cards…)
   - Careful balance data size with bandwidth (and latency)
4. The network is secure.
   - Not only the wires, the environment in general
   - Malware, DDOS, spying…
   - Software : need to take security into account from the start

The 8 fallacies

5. Topology doesn't change.
   - Network route change, machines change, clients (wireless) come and go
   - Software : do not depend on specific endpoints or routes, location transparency
6. There is one administrator.
   - Multiple companies/entities, multiple OS
   - Software :
     - administrators will set constrains (ram, disk…)
     - need to provide tools for remote diagnostic, deployment, adminitration
7. Transport cost is zero.
   - Additional costs (performance or money) when using the network
   - Software : use cost-effective solutions
8. The network is homogeneous.
   - Various protocols and technologies
   - Use open standard, do not rely on proprietary protocols