# Frugal AI: Rethinking model design and objective for a sustainable future

**David Cornu**
Aristide Doussot
Gregory Sainton

**PSL-week 2025 / hosted at ESPCI**

# Provisional Planing of the course / week

- **Introduction and context** on the impact of ICTs on the environment (Day 1)

- **Part I:** Numerical optimization for energy saving
  → practical case of the matrix multiplication
  - Python naive implementation (Day 1)
  - Compiled C, from naive to low level optimization (Day 2-3)
  - CPU parallelization in C, and introduction to GPU matrix multiplication (Day 3)

- **Part II:** Artificial Neural Networks architecture tweaking for efficiency
  - Introduction on architecture design and computation efficiency (Day 4)
  - Case study of optimizing an image classification task on defined energy budget (Day 4-5)

- **Validation of the week:** full week attendance + practical work report

# Lessons materials

Slides, exercises, codes, corrections and datasets are **available on GitHub** and will be updated regularly:

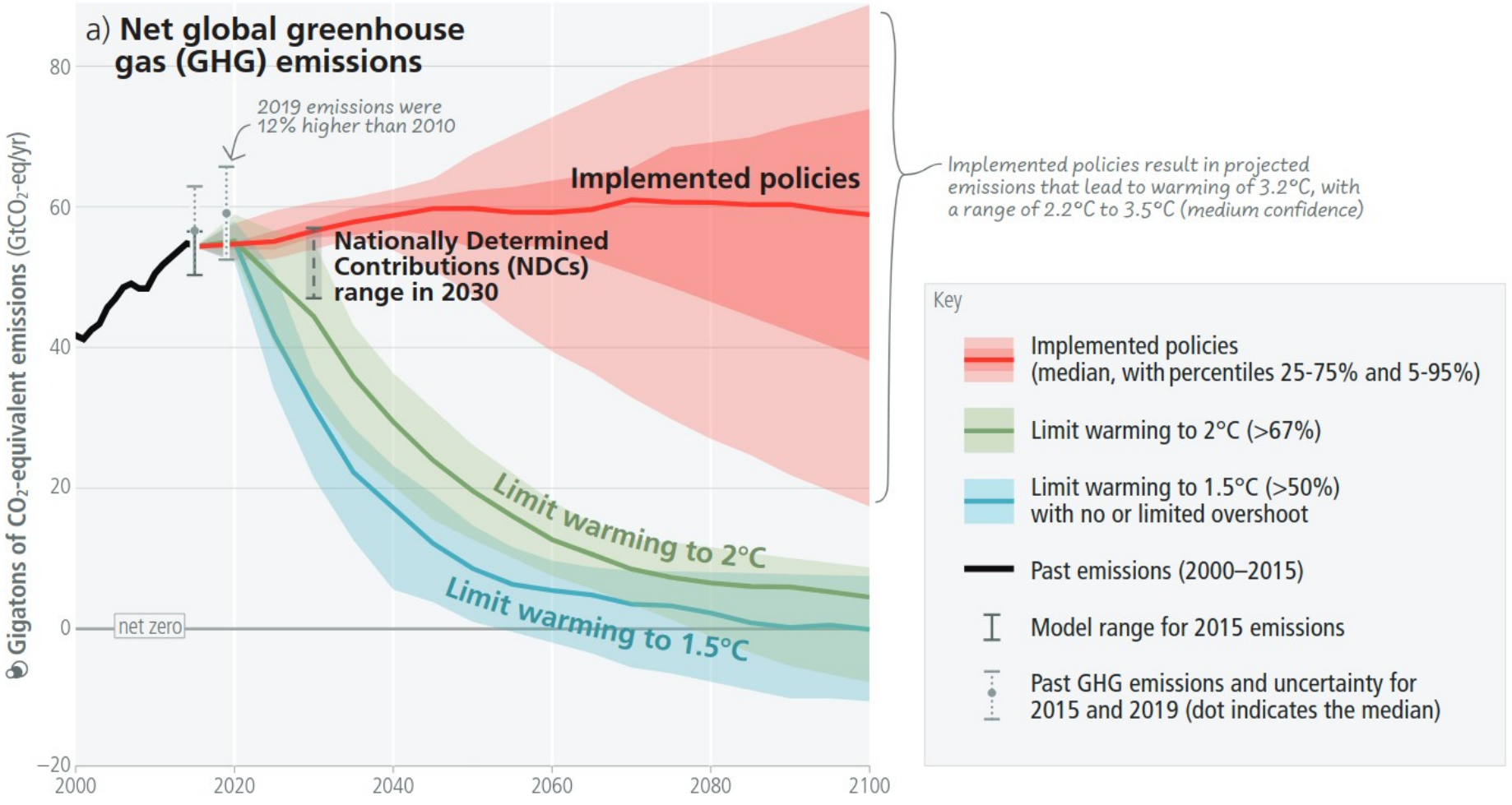
https://github.com/Deyht/frugal_ai

*git clone https://github.com/Deyht/frugal_ai*
*git pull*

Or download the repository as a zip file.

Do not copy and paste content from git-hub pages (leads to format errors).

# Global context




*From GIEC SPM 2023*

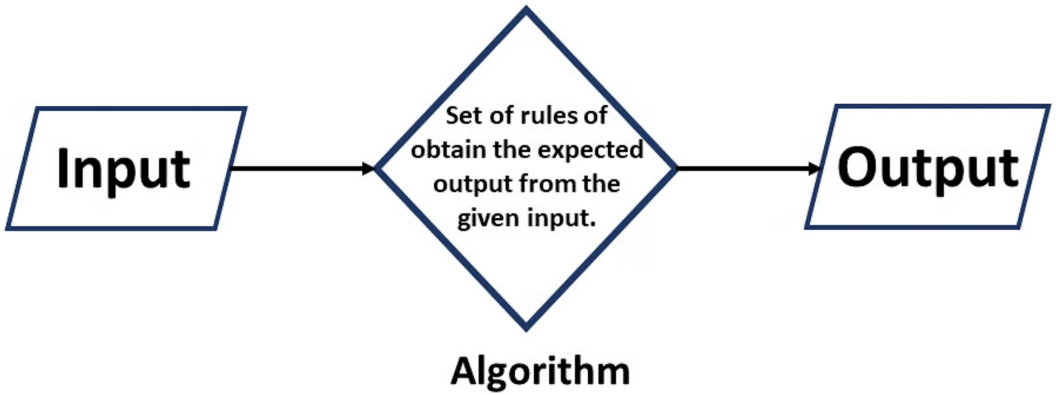
# What is AI / ML and how does it impact the environment ?

AI is a computer program that provides a solution to a problem given some data.

**Mostly like any other computer program!**

They even use the same type of numerical infrastructure.

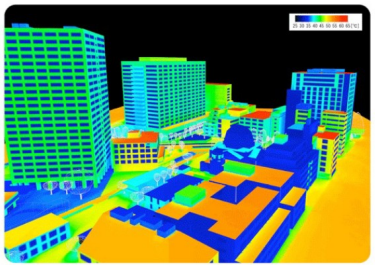
*The specificity of AI models is that they learn the inner set of rules automatically through statistical learning.*



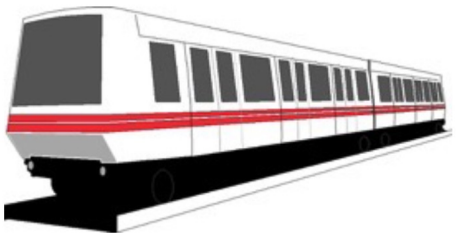

## First question is:

**How do numerical infrastructures and technologies impact the environment?**

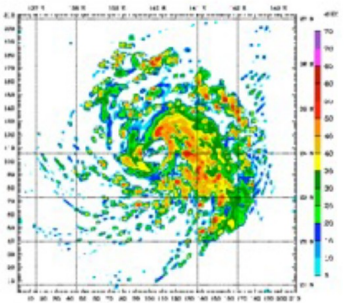
# The many forms of the numerical world









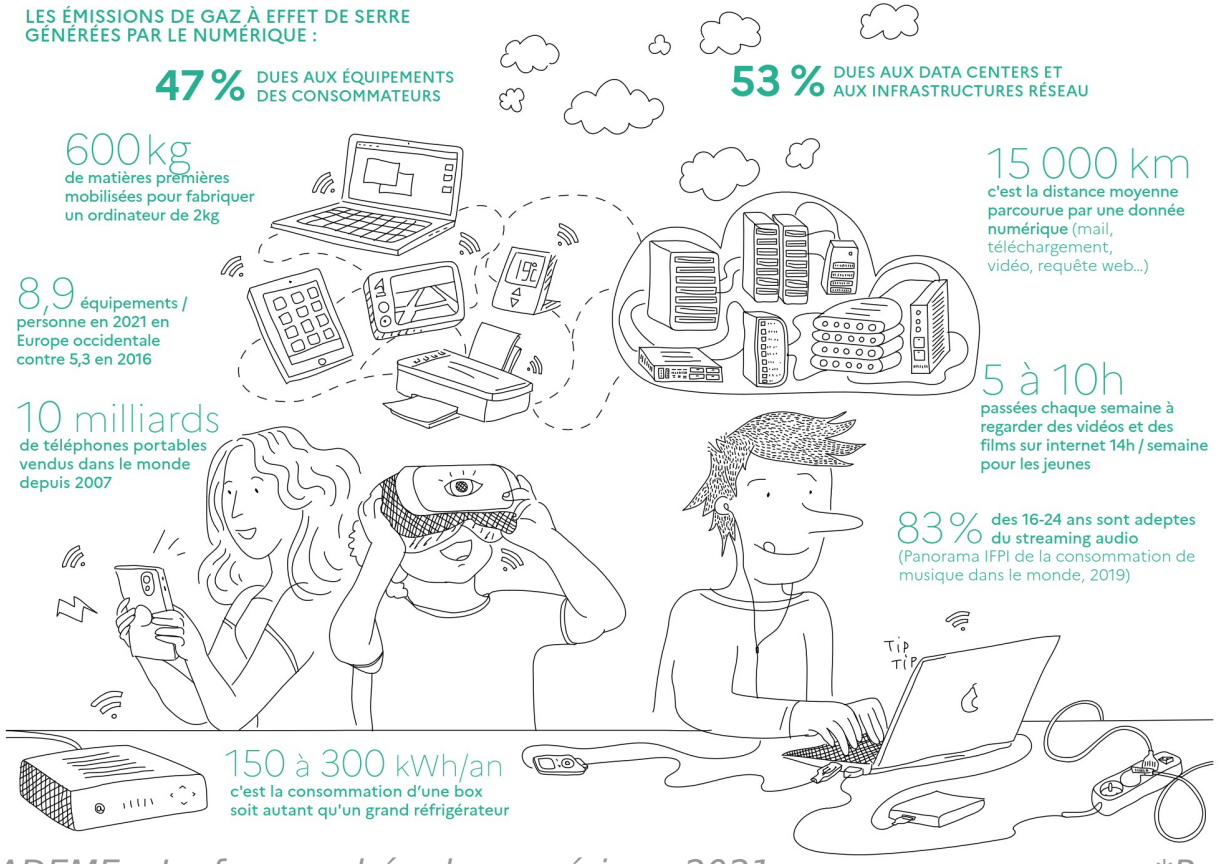

Often referred to as **ICTs = Information and communications technologies.**
**ICTs are now a part of all activity fields, so is their environmental impact**

*From Anne-Cecile Orgerie talk, « L'informatique verte moins branchée »*

# A few interesting numbers

## ICTs are responsible for ~4% of global CO2-e emissions

**=>** Might seem low, but this sector is **growing fast**, especially with the development of **AI and IoT devices. In France the share of ICT went from 2.5% in 2020 to 4.4% in 2024*.**




INTERNET AU NIVEAU MONDIAL

- 67 millions de serveurs
- 1,1 milliard d'équipements réseaux (routeurs, box ADSL…)
- 19 milliards d'objets connectés en 2019
- 48 milliards en 2025 selon les estimations

En 1 heure

- 8 à 10 milliards de mails échangés (hors spam)
- 180 millions de recherches Google

*Sourece ADEME – La face cachée du numérique 2021*

**Rapport Ademe du 09/01/2025*

# Distribution of ICT energy consumption

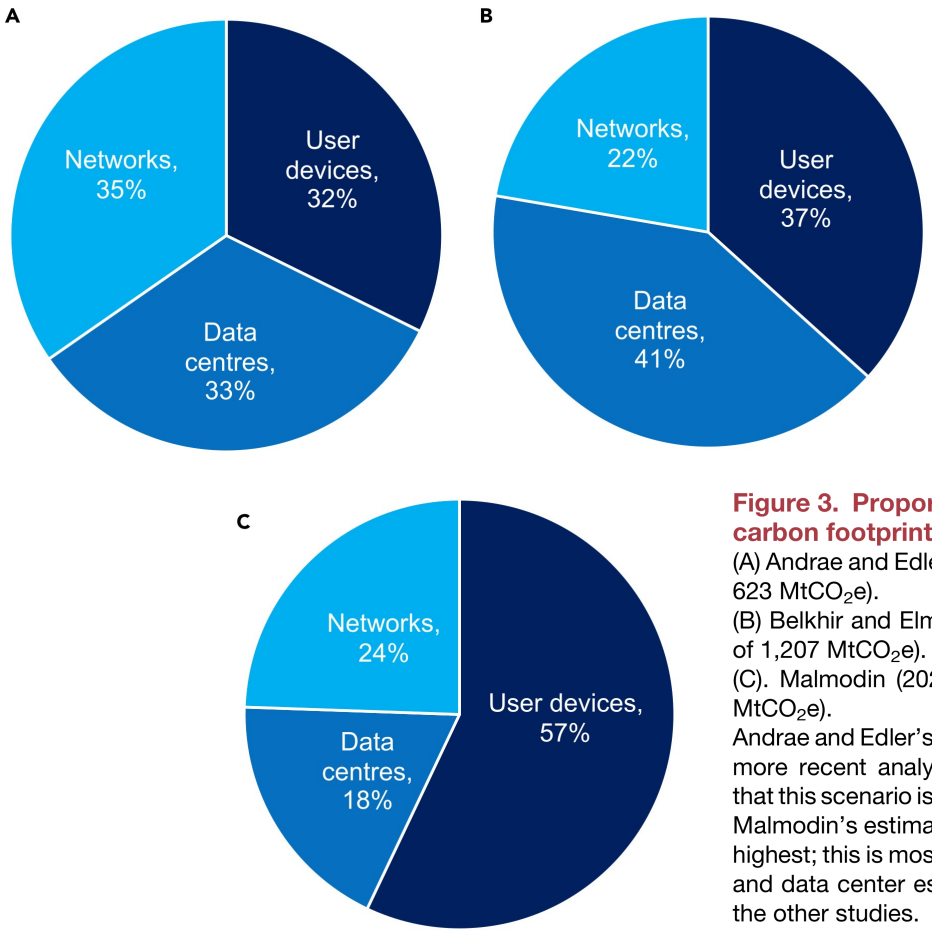



**Figure 3. Proportional breakdown of ICT's carbon footprint, excluding TV**
(A) Andrae and Edler (2015): 2020 best case (total of 623 $MtCO_2e$).
(B) Belkhir and Elmeligi (2018): 2020 average (total of 1,207 $MtCO_2e$).
(C). Malmodin (2020): 2020 estimate (total of 690 $MtCO_2e$).
Andrae and Edler's[3] best case is displayed because more recent analysis by the lead author suggest that this scenario is most realistic for 2020. Note that Malmodin's estimate of the share of user devices is highest; this is mostly because Malmodin's network and data center estimates are lower than those of the other studies.

## ICTs are usually split into 3 categories:

- **User devices** (your smartphone or laptop)
- **Network infrastructures,** which allow the exchange of information and data
- **Data centers** that centralize the relevant data and services

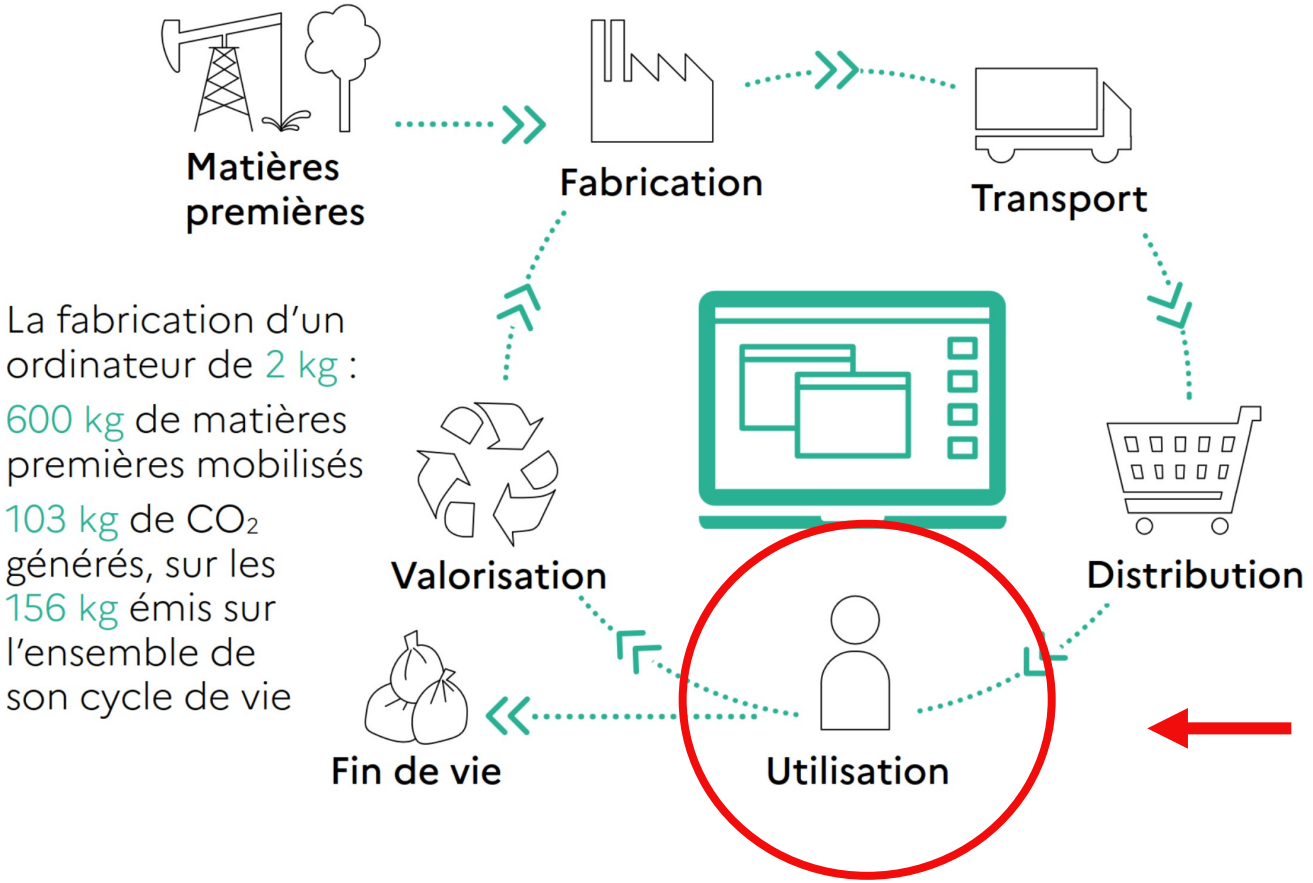
**All three categories are growing fast !**

*Estimating the respective contributions of these parts in terms of CO2-e is difficult and depends on models with many assumptions regarding:*

*- The lifespan of the hardware*
*- The local electrical mix*
*- How and where the hardware was built and the origin of the materials*
*- The exact field to which is associated a numerical activity*
*(eg. should autonomous vehicle computers count in ICT or in mobility?)*

# Life cycle of an ICT

## There is an environmental impact at every step of the life cycle




*Increasing the lifespan of an ICT is usually an efficient way of reducing CO2-e emissions, even when the new version is more efficient in use.*

*We note that planned obsolescence is responsible for many unnecessary renewals of devices.*

*End-of-life can also consume a lot of energy or have a strong impacts on the environment other than CO2-e!*

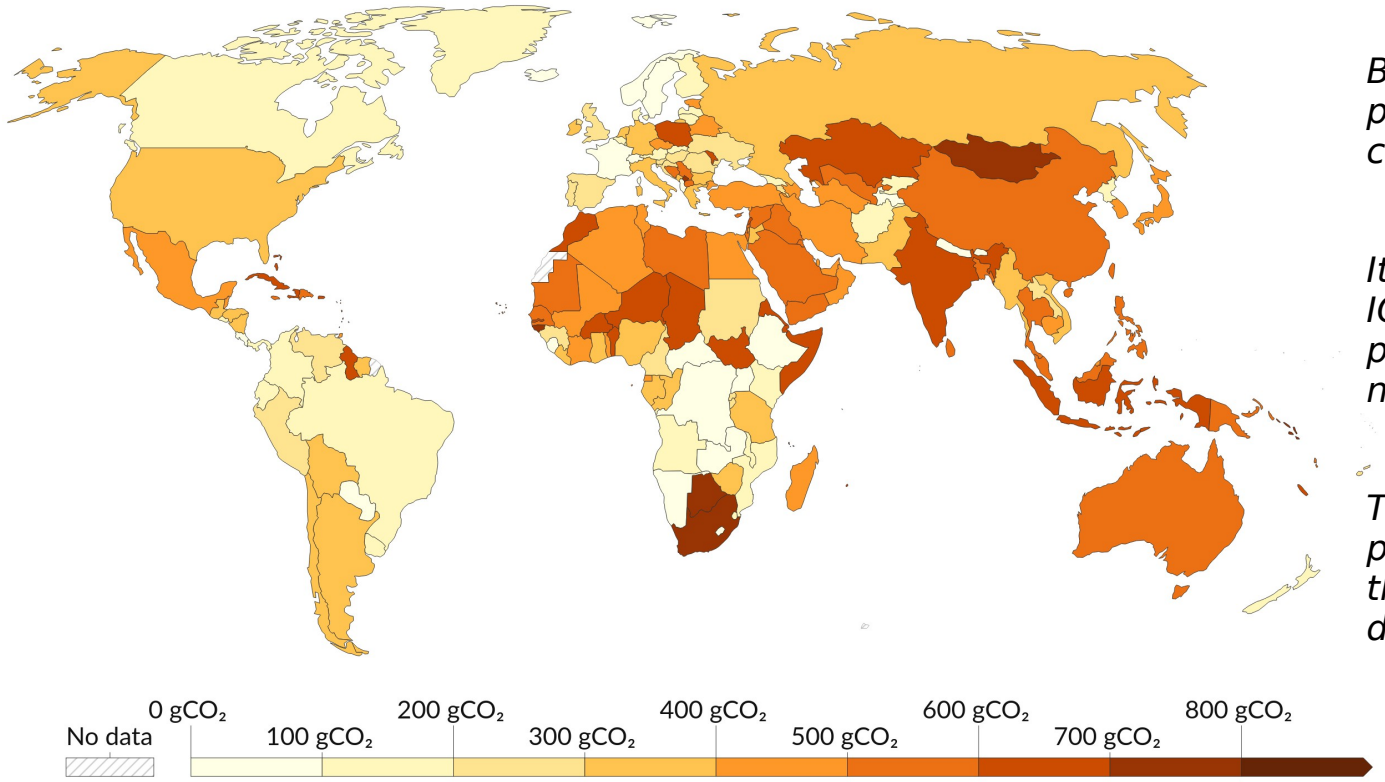
**Usually a small part of the total impact, but get most of the attention !**

*From ADEME – La face cachée du numérique*

# Carbon intensity of electricity generation, 2022

Carbon intensity is measured in grams of carbon dioxide-equivalents emitted per kilowatt-hour of electricity generated.

Our World in Data




*Building an ICT is now a worldwide process. All aspects of the life cycle can occur in different countries.*

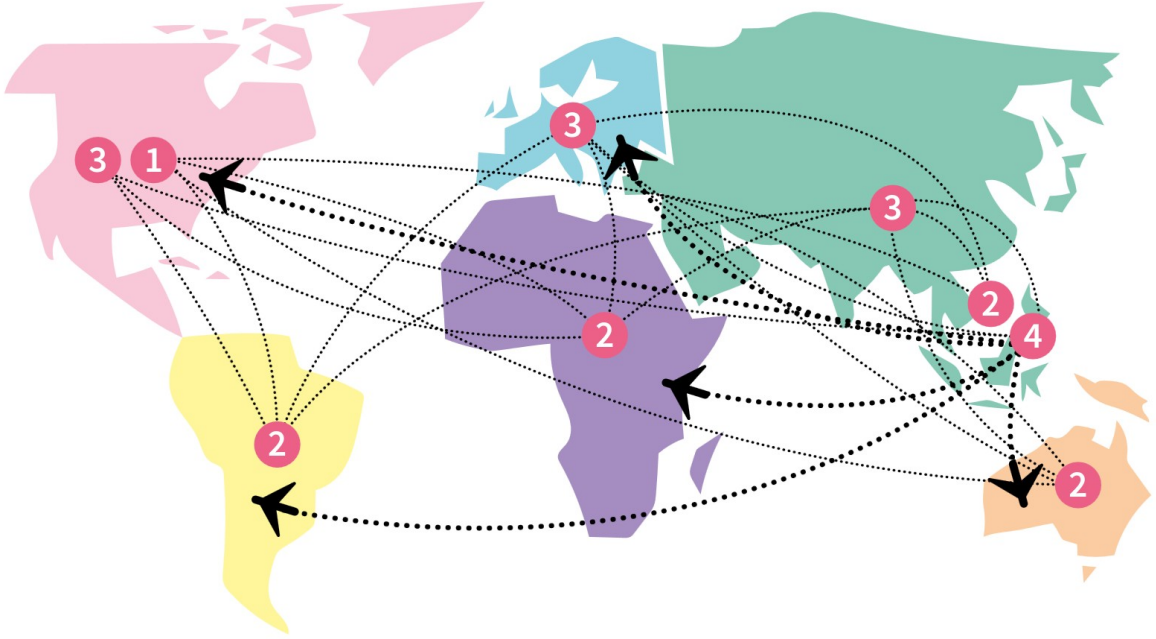
*It also impacts the network aspect of ICT as the server you use can be physically far away without you noticing it.*

*The lifespan of an ICT for which the production cost is still higher than the usage cost also varies strongly depending on where it is used.*

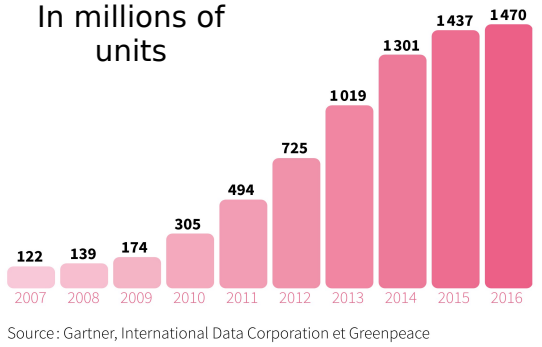
**Data source:** Ember - Yearly Electricity Data (2023); Ember - European Electricity Review (2022); Energy Institute - Statistical Review of World Energy (2023)

OurWorldInData.org/energy | CC BY

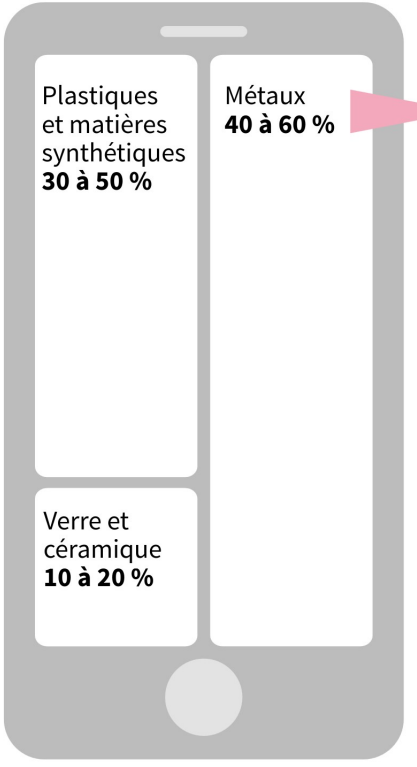
# Example case of a smartphone assembly




1 – Conception
2 – Raw materials extraction
3 – Main component manufacturing
4 – Final assembly
✈ – Worldwide distribution




Source : Gartner, International Data Corporation et Greenpeace




**PROPORTION DES MÉTAUX**

**80 à 85 %** de métaux ferreux et non ferreux : cuivre, aluminium, zinc, étain, chrome, nickel…

**0,5 %** de métaux précieux : or, argent, platine, palladium…

**0,1 %** de terres rares et métaux spéciaux : europium, yttrium, terbium, gallium, tungstène, indium, tantale…

**15 à 20 %** d'autres substances : magnésium, carbone, cobalt, lithium…

*From ADEME – Les impacts du smartphone*

# The chemical elements of a smartphone

COMPOUND INTEREST

**Elements colour key:** Alkali metal · Alkaline earth metal · Transition metal · Group 13 · Group 14 · Group 15 · Group 16 · Halogen · Lanthanide

## SCREEN

In (Indium), Sn (Tin), O (Oxygen)

**Touch: Indium tin oxide**
Used in a transparent film over the phone's screen that conducts electricity. This allows the screen to function as a touch screen. This is the major use of indium.

Al (Aluminium), Si (Silicon), O (Oxygen), K (Potassium)

**Glass: Alumina and silica**
On most phones the glass is aluminosilicate glass, a mix of aluminium oxide and silicon dioxide. It also contains potassium ions which help strengthen it.

Y (Yttrium), La (Lanthanum), Tb (Terbium), Pr (Praseodymium), Eu (Europium), Dy (Dysprosium), Gd (Gadolinium)

**Colours: Rare earth metals**
A variety of rare earth metal-containing compounds are used to help to produce the colours in a smartphone's screen. Some of these compounds are also used to help reduce light penetration into the phone. Many of the rare earths occur commonly in the Earth's crust, but often at levels too low to be economically extracted.

## ELECTRONICS

**Wiring and microelectronics**
Copper is used for wiring, and for micro-electrical components along with gold and silver. Tantalum is the major component in micro-capacitors.

Cu (Copper), Ag (Silver), Au (Gold), Ta (Tantalum)

**Microphones and vibrations**
Nickel is used in the microphone and for electrical connections. Rare earth element alloys are used in magnets in the speaker and microphone, and the vibration unit.

Ni (Nickel), Dy (Dysprosium), Pr (Praseodymium), Tb (Terbium), Nd (Neodymium), Gd (Gadolinium)

**The silicon chip**
Pure silicon is used to manufacture the chip, which is then oxidised to produce non-conducting regions. Other elements are added to allow the chip to conduct electricity.

Si (Silicon), O (Oxygen), Sb (Antimony), As (Arsenic), P (Phosphorus), Ga (Gallium)

**Connecting electronics**
Tin and lead were used in older solders; newer, lead-free solders use a mix of tin, copper and silver.

Sn (Tin), Pb (Lead)

## BATTERY

Li (Lithium), Co (Cobalt), O (Oxygen), C (Carbon), Al (Aluminium)

Most phones use lithium-ion batteries, which are composed of lithium cobalt oxide as a positive electrode and graphite (carbon) as the negative electrode. Sometimes other metals, such as manganese, are used in place of cobalt. The battery casing is often made of aluminium.

## CASING

Magnesium alloy is used to make some phone cases. Many others are made of plastics, which are carbon-based. Plastics will also include flame-retardant compounds, some of which contain bromine, whilst nickel can be included to reduce electromagnetic interference.

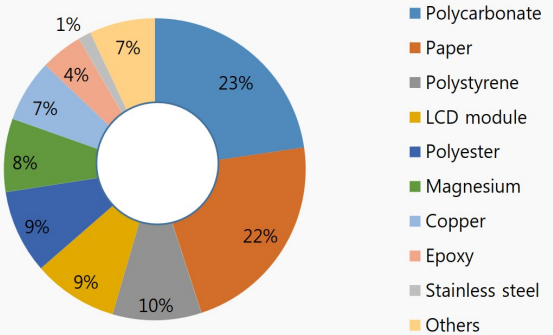
C (Carbon), Mg (Magnesium), Br (Bromine), Ni (Nickel)

# The limits of the CO2-e measurement
# Other types of impacts on the environment

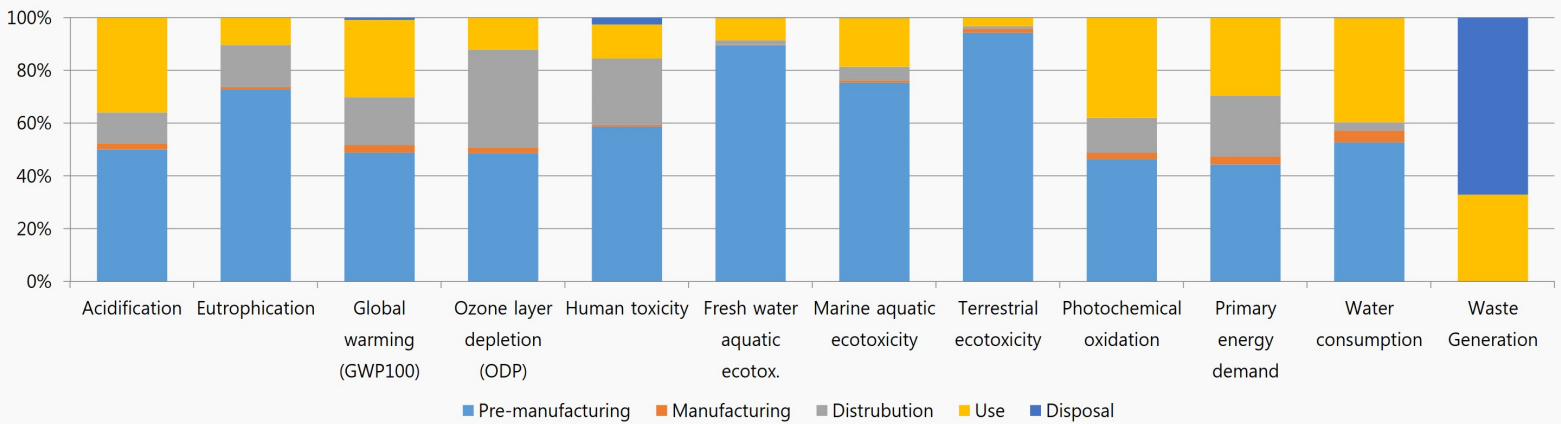
## Product Features

| | |
|---|---|
| Model name | SM-W727V (Galaxy Book) |
| Processor | Intel, Core i5, 3.1GHz Dual-Core 64bit |
| Dimension | 199.8 * 291.3 * 7.4(H*W*D) |
| Display | AMOLED, OCTA, SDC, 2160 x 1440 (FHD+) 12.0", 303.7mm 16M |
| Battery | Li-Ion 5070 mAh |
| Camera | 13 MP / 5MP |
| Wt.(g) | 1881.9g |

## Material Use

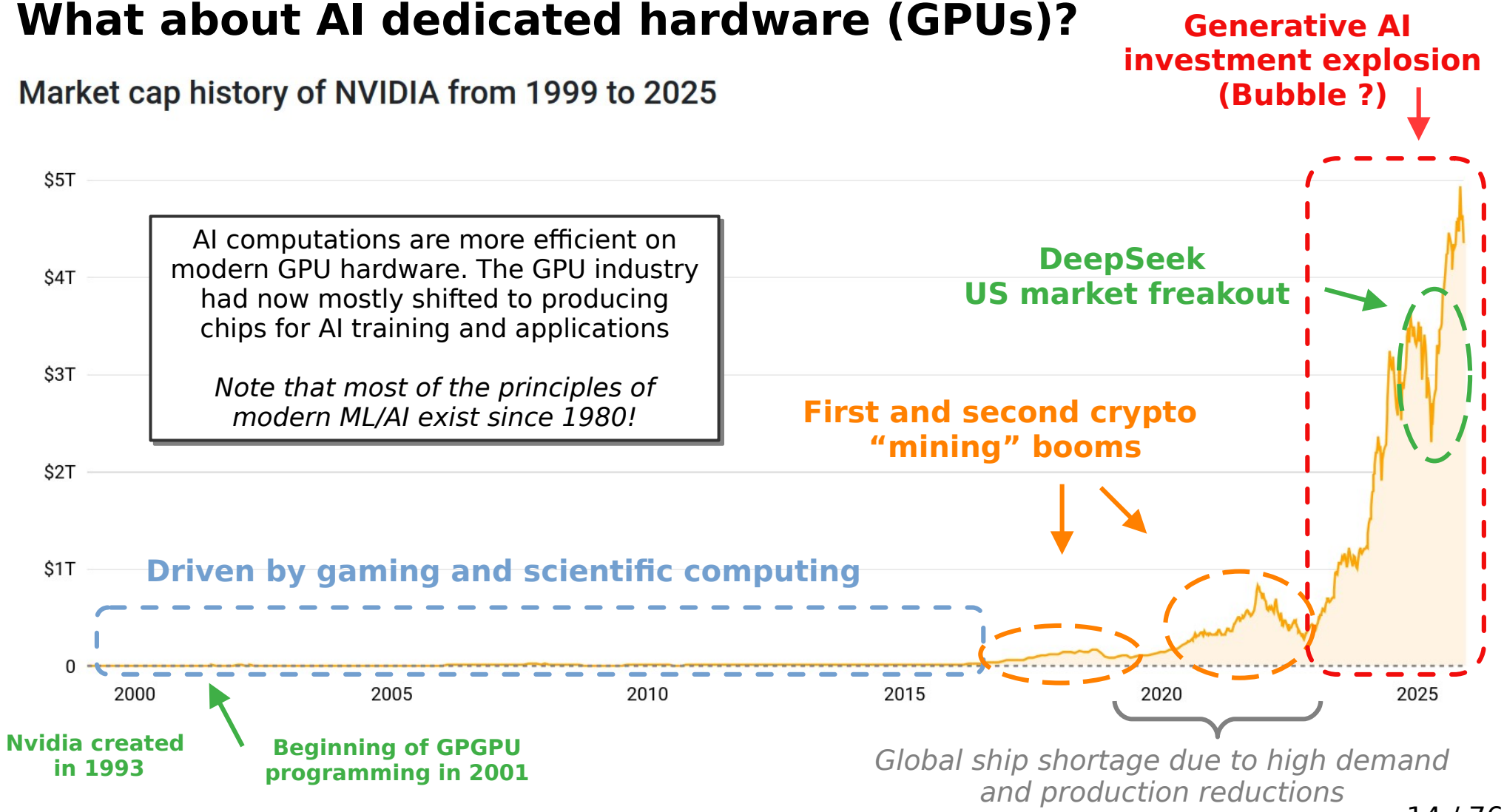



## Characterized Environment Impact

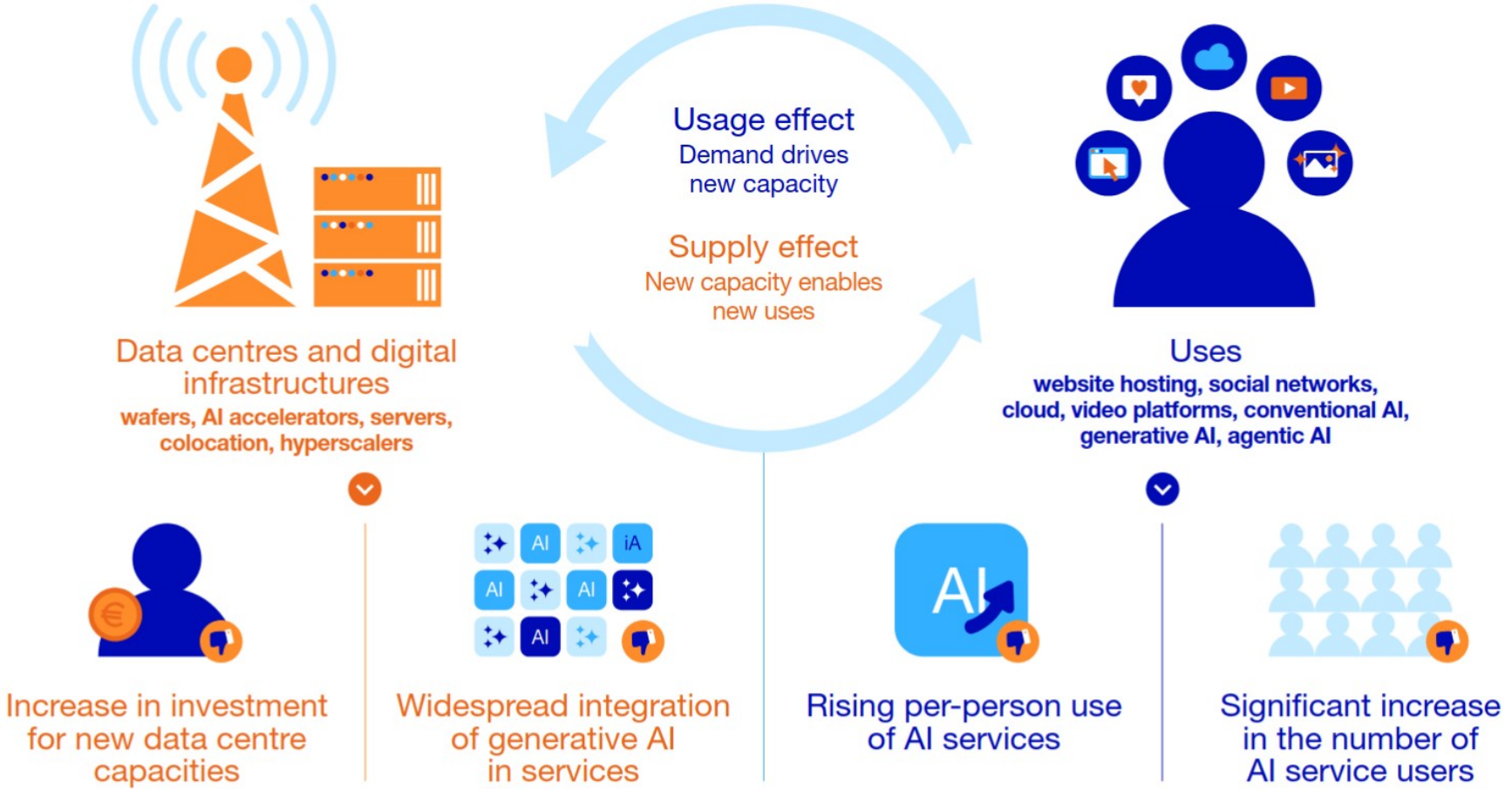



*From Samsung Life Cycle Assessement for Mobile Products, 2018*

# What about AI dedicated hardware (GPUs)?



# Supply and usage of AI services are interconnected




*From the summary report of « The Shift project » on AI*

# Supply and usage of AI services are interconnected

By 2030, the trajectory projected for the data centre sector **is unsustainable:**

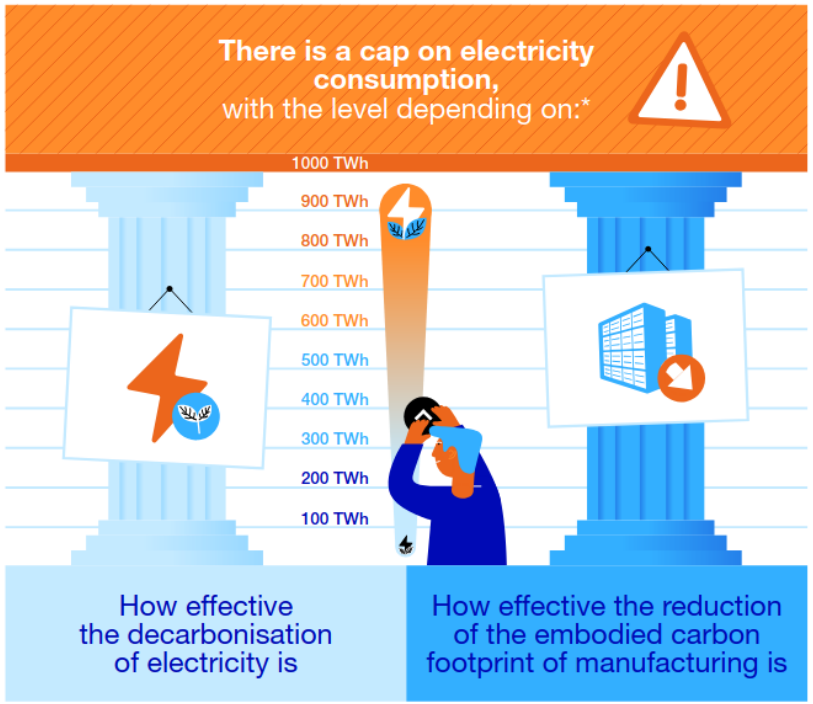
**+9% GHG per year**
despite decarbonisation of the electricity mix

VS

**-5% GHG per year**
required to reach the net-zero target

920 $MtCO_2e$ per year,
**up to twice France's annual emissions.**

## Successfully decarbonising the global data centre sector means:




*By way of illustration, achieving 90% decarbonisation of the sector would require reaching:
- 200 TWh for 111 $gCO_2e/kWh$ with a 90/10 split between use/manufacturing;
- 1000 TWh for 25 $gCO_2e/kWh$ with a 95/5 split between use/manufacturing (illustrative calculations).

## The European context: different situations but a common trend in the data centre sector

Electricity use by data centres in Europe could **double** between 2023 and 2030 and **quadruple** between 2023 and 2035*.

*Rising from 97 TWh in 2023 to 200 TWh in 2030 and 369 TWh in 2035

In Ireland, data centres already account for **over 20 %** **of available electricity,** surpassing urban residential demand.

This increase in electricity use is, **to our knowledge, not factored into energy planning scenarios.**
It could therefore jeopardise Europe's ability to meet its climate targets.

*From the summary report of « The Shift project » on AI*

*« Steering the decarbonisation of the data centre sector requires decisions that are not only technical but also societal and political. »*

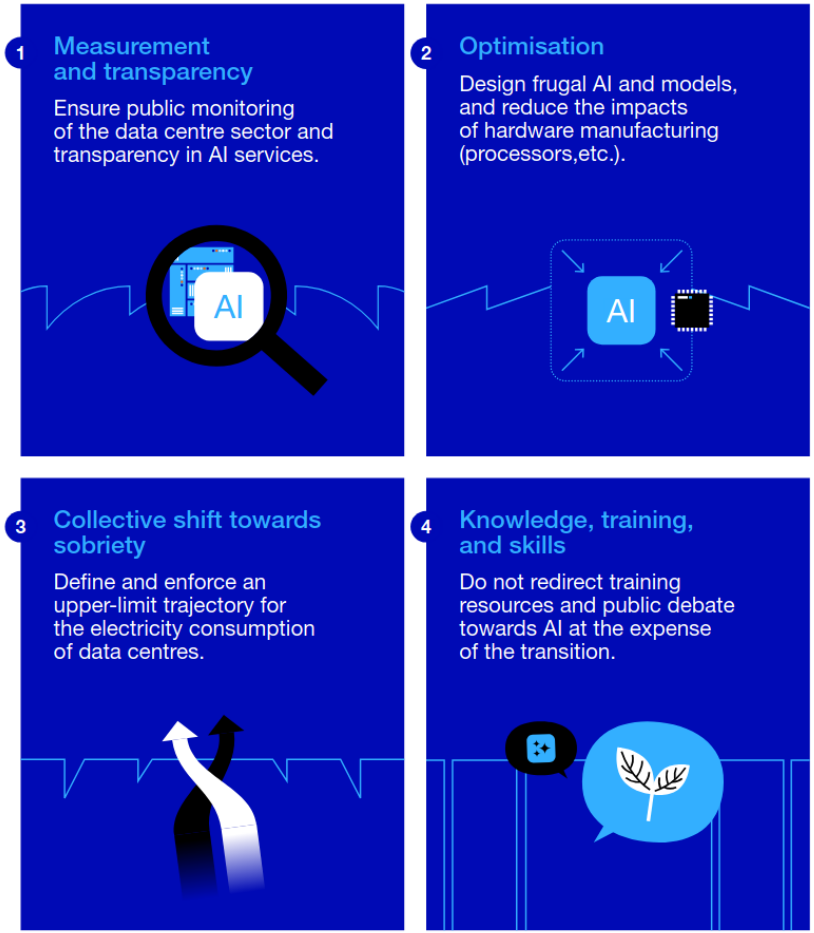
**Always establish the « need » and energy budget first, then**

## Steering our technological choices

**Design lever of action:** optimise (lightweight models, hardware impacts, etc.) and act on functionalities (transform, alter or drop certain features).

**Deployment lever of action**: adjust how widely solutions are rolled out (targeted or generalised) to match the need and the conditions for compatibility with the carbon budget.

**If these tools do not make the AI solution compatible with the reference carbon budget, it must be abandoned or replaced with a non-AI solution.**




*From the summary report of « The Shift project » on AI*

# Context elements we haven’t talked about

- **Can a numerical transition accompany an environmental transition?**
  Use of numerical algorithms or devices to optimize other transitions. Is more ICTs a solution?
- **The evolution of the data rate for various application is exploding and current intercontinental connections are facing strong limits.**
- **Geopolitics → No country is autonomous in producing ICTs. Network traffic is worldwide and requires continuity of physical infrastructures.**
- **We have almost not talked about ICTs end of life → recycling? E-waste?**
- **Usage regulations? For which applications?**

EcoInfo
Groupement de service

POUR UNE INFORMATIQUE ÉCO-RESPONSABLE

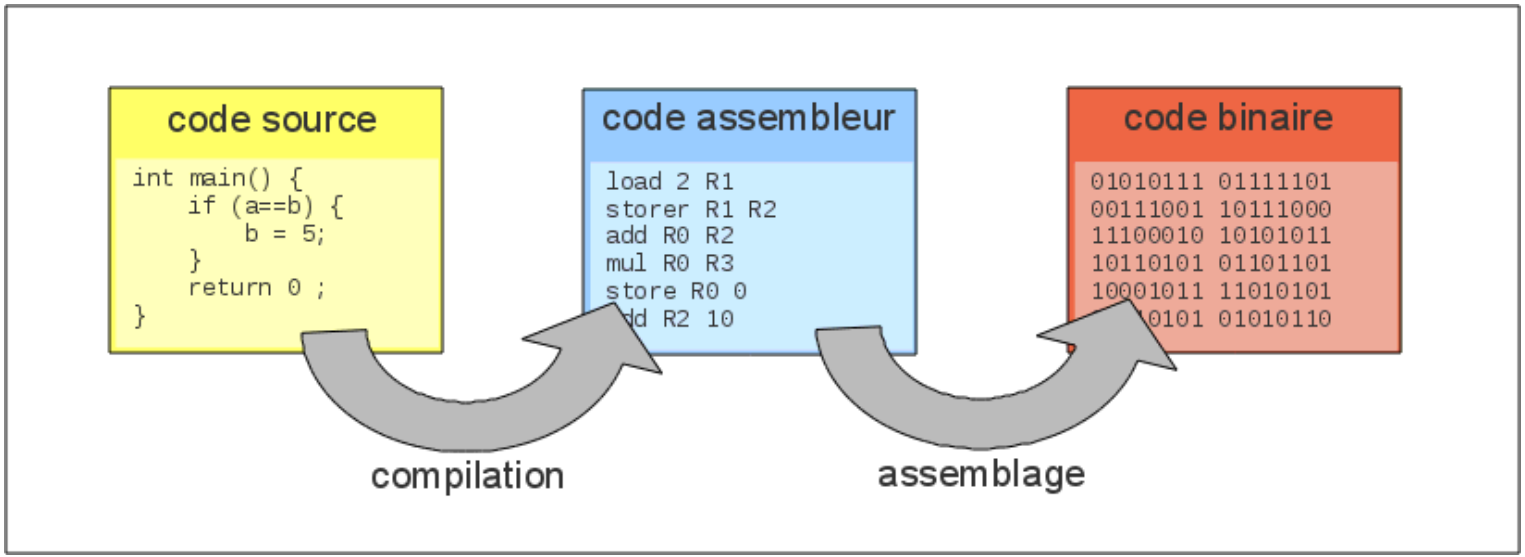
*Many resources are accessible through the EcoInfo CNRS GDR that is a community of researchers that try to better estimate and explore solutions to reduce the environmental impact of ICTs.*

# Tailored objectives of this PSL week

- Provide insights on the design and manufacturing process of ICTs.
- Identify what impacts the energy consumption of ICT systems.
- Present algorithmic optimization strategies based on a deeper understanding of ICT systems design.
- **Practical work 1:** optimizing the matrix multiplication algorithm that drives all modern AI/ML models.
- Present classical ANN architectural elements and their computational cost.
- Understand how architectural setup affects model expressivity and computational cost to find possible actions to improve efficiency.
- **Practical work 2:** optimizing an ANN model in a given energy budget.

# Re-centering on the objective of the week: Computation efficiency and energy saving

## First what is a computer program ?




**1) High level code, close to natural language**

**3) Binary machine code that can run on a CPU**

**2) Low level code, series of basic instructions**

# The many shapes of computers

*Destop / workstation*

*Laptop*

*Micro computer*

*Smartphone*


*Server*

# What are the mains components of all computers ?


***CPU***
*Do the computation*

***Storage***
*Keep the static data*

***RAM***
*Store the volatile data*

***Possible other parts and peripherals include:***

*- Screen*
*- GPU*
*- External storage readers*
*- Internal power supply*
*- Cooling solution*
*- etc .*

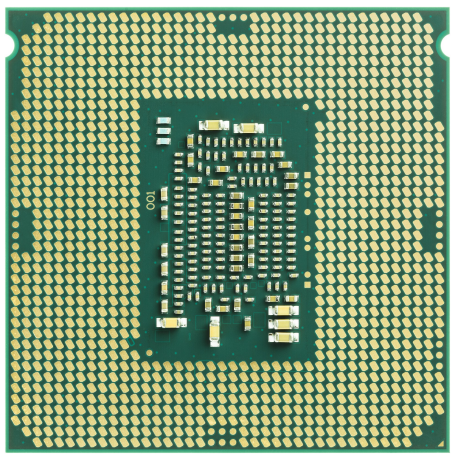
***Motherboard***
*Link all the other elements and peripherals*

# The heart of the computer: the Central Processing Unit (CPU)


**A CPU is an electronic circuit capable of executing instructions for a program, such as arithmetic, logic, or I/O operations.**

Modern CPUs are implemented on integrated circuits and combined with cache memory and peripheral interfaces.

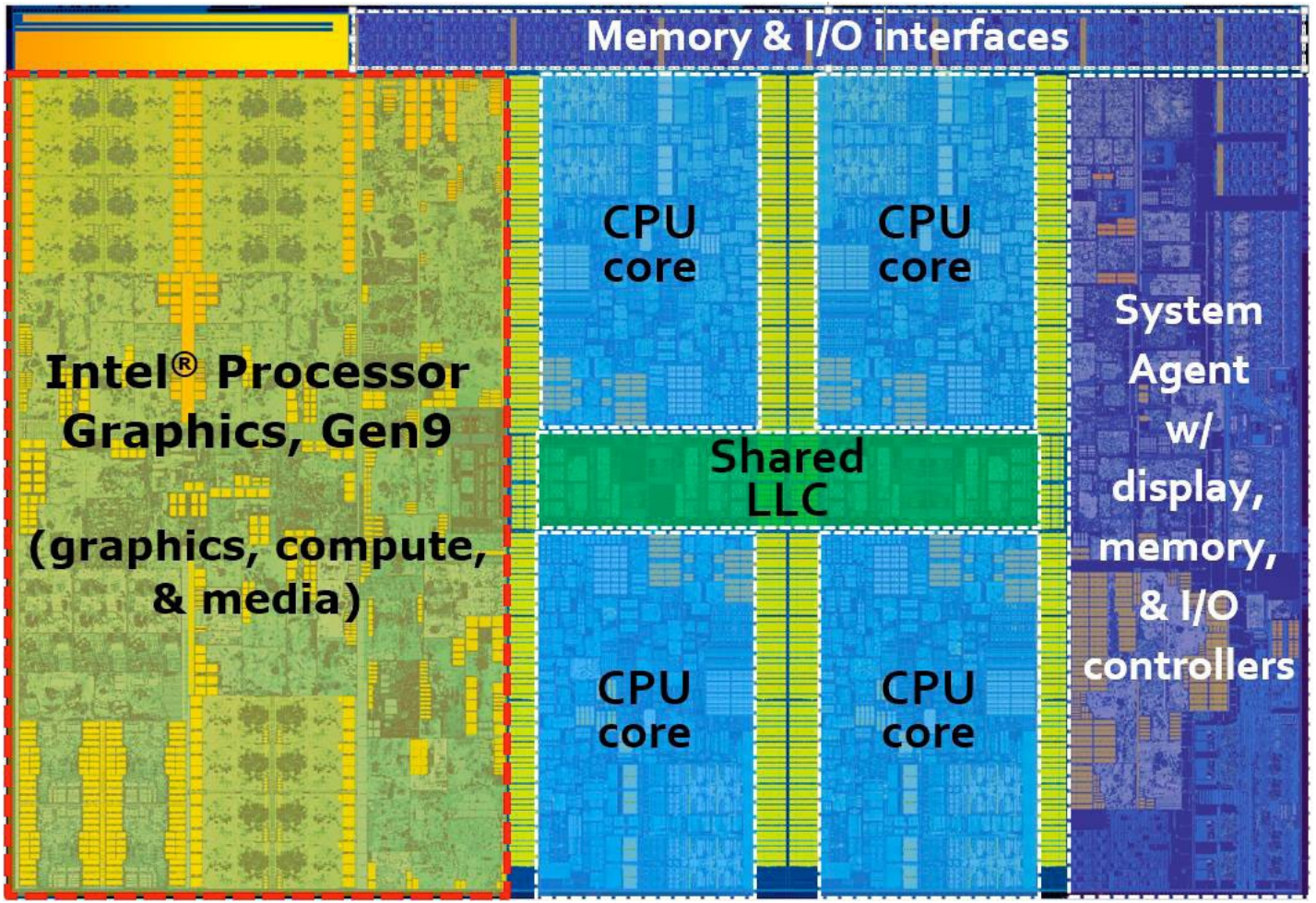
A CPU is a set of billions of transistors combined to form analogical operations that can be regrouped to form numerical instruction sets.



The “speed” of the processor is characterized by its **frequency**, which represents how many **“cycles”** are made on the processor per second.

Modern CPU are equipped with **multiple computation cores** so they can execute several independent instruction streams in parallel. This construction allows the mutualization of all the elements of the CPU that are not dedicated to computing.

# The heart of the computer: the Compute Processing Unit (CPU)



# What determines a CPU theoretical performances?

A CPU is characterized by its **IPC (Instruction per cycle)** capability. Increasing the IPC can be done by improving the memory hierarchy or the instruction pipeline.

The **frequency** defines how many cycles operate per second. It is limited by the physical capability of the processor to support higher power draw and dissipate the generated heat. **Modern processors have dynamical frequency scaling that adapts to the current load on the processor (TurboBoost)!**

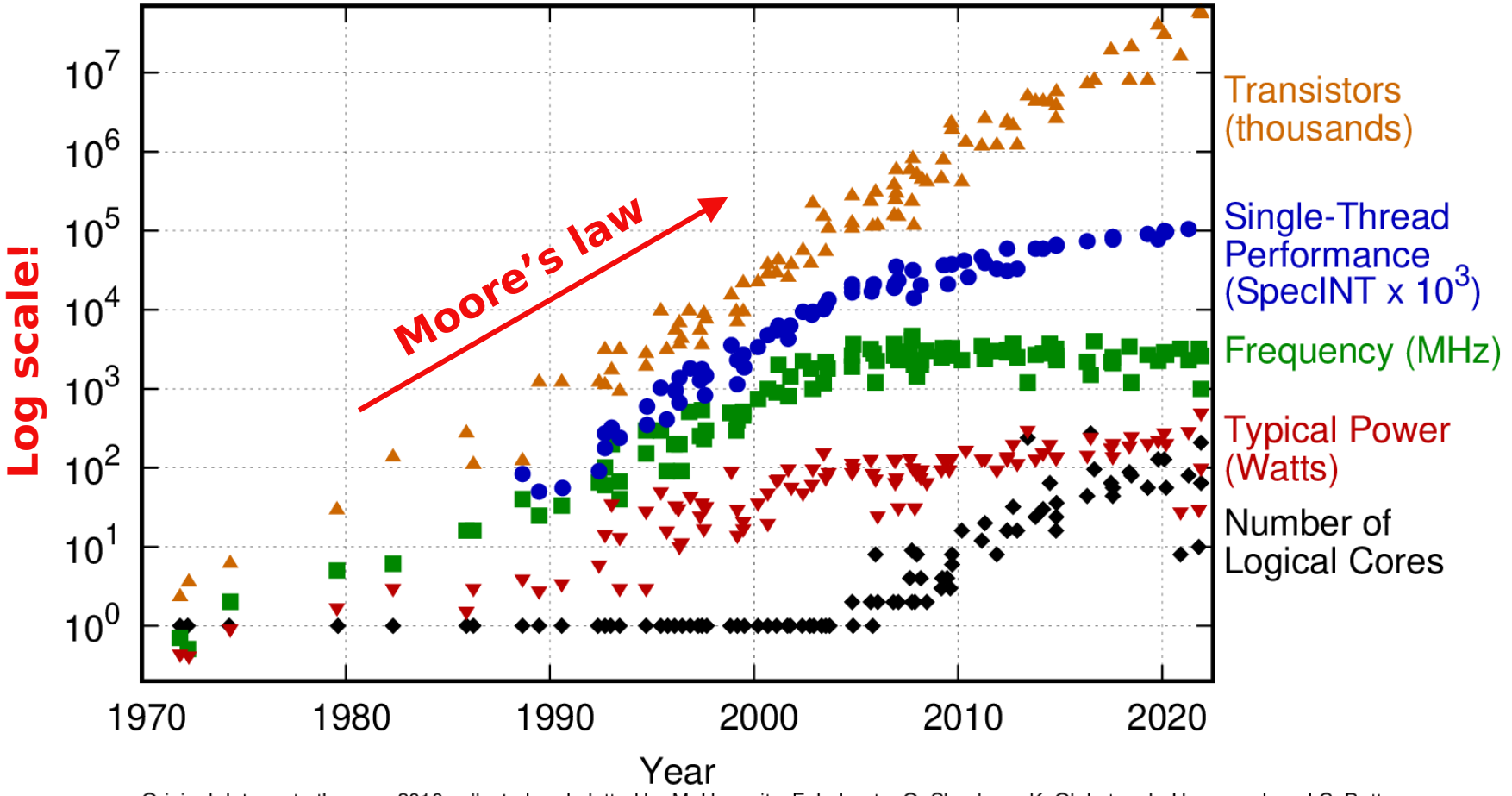
*For a short load of a few seconds, the frequency can go very high as it will not have time to saturate the integrated heat spreader thermal mass absorption. For moderate load time, the frequency can remain high depending on the cooling solution, for example, heating all the water in a water cooling loop. Then, when the cooling system is saturated, the CPU will stabilize to a lower frequency that matches the heat dissipation capability.*

The **number of cores** in a CPU can provide almost linear performance scaling depending on the application while sharing some parts of the CPU like the caches. Some CPUs have a technology called **Hyper-Threading,** allowing each physical core to handle two execution steams with independent registries and low-level cache.

*This ensures that the actual computation units of the physical core are always saturated. Most of the time the scaling is way less good than when using more physical cores. Using multiple cores will saturate the heat dissipation system much faster, so the CPU usually runs at lower speeds when all the cores are computing simultaneously.*

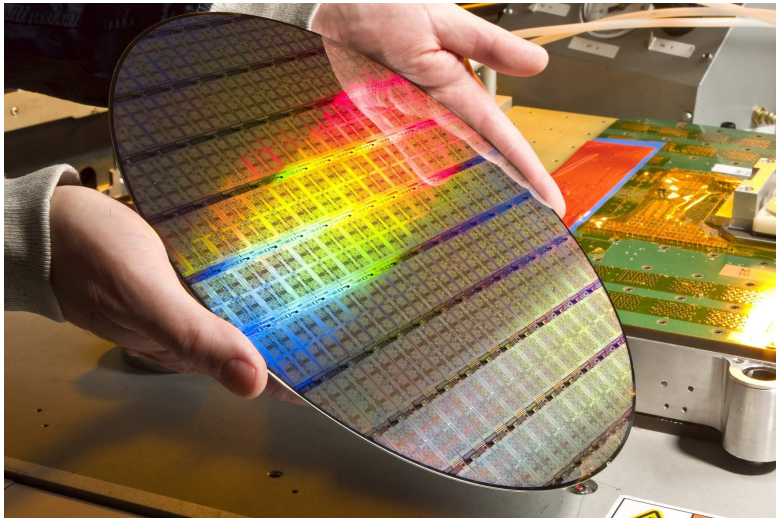
# The historical mighty quest for performance

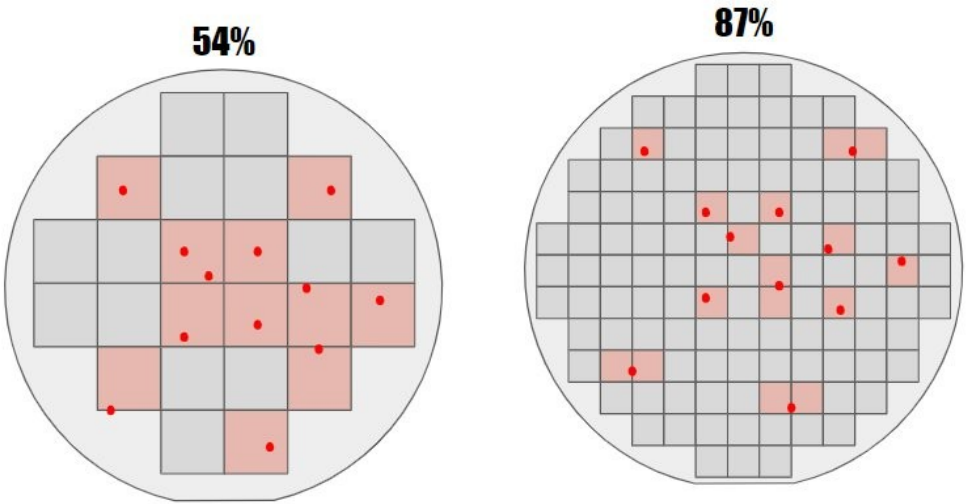
## For decades, computer engineers have tried to improve CPU performance




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

# How to build a CPU ?






**CPUs are “engraved” in batches on silicon wafers** using **photolithography**.

CPU dies are getting **larger and include more cores** to increase the computing power, while frequency stagnates. This increased size, combined with an increasing engraving **difficulty at low finesse**, implies that the **faulty die rate increases**. This leads to higher production costs for high-end processors.

The engraving quality can vary with position on the wafer, leading to a **dispersion in efficiency for a given CPU chip model of a few %** → This principle is commonly referred to as **« silicon lottery ».**

At the scale of a cluster, the cumulated variability for each component induces a variability on the full system efficiency of **up to 10%.**

# Efficiency of a computing system or facility

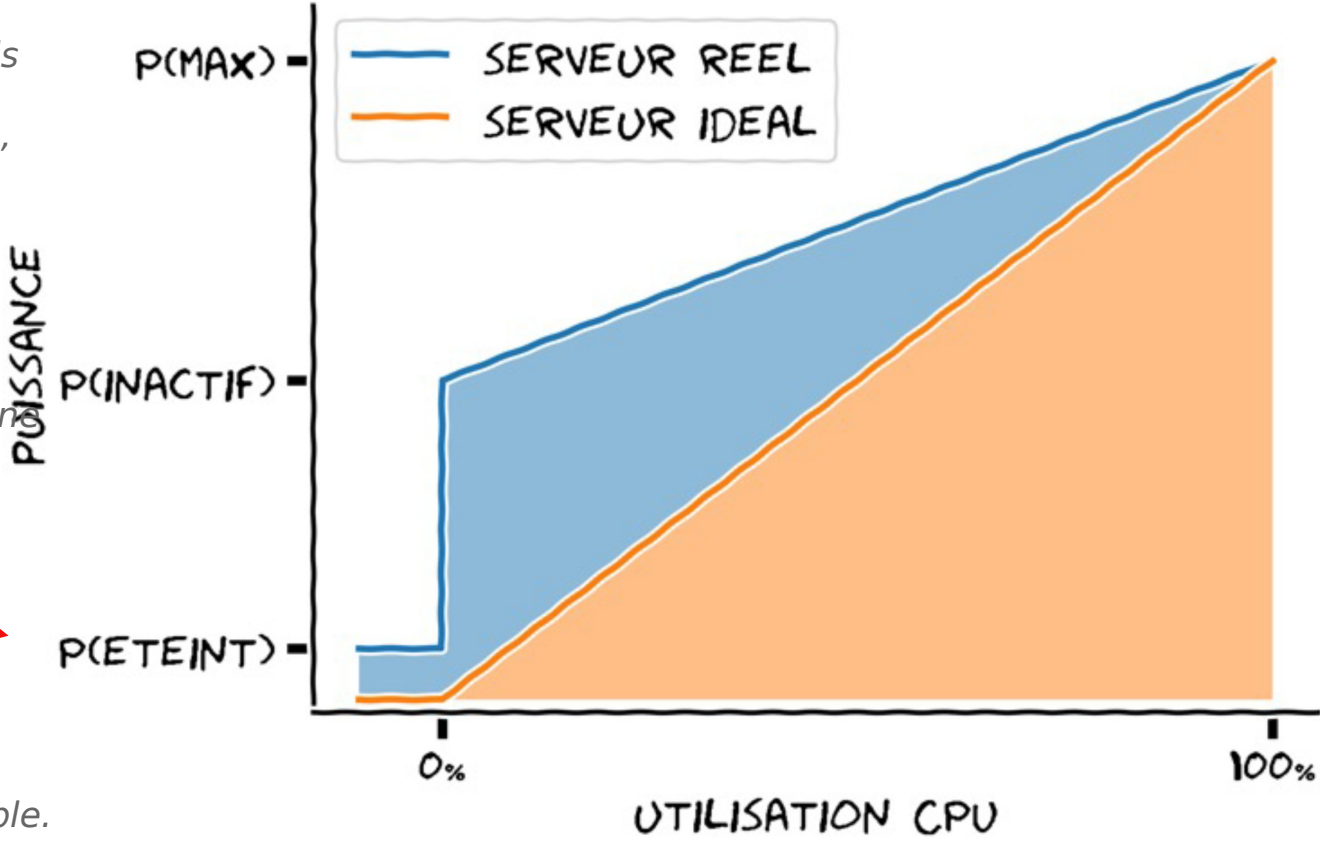
*All idle systems have a baseline power draw. For individual modern systems, it is often driven by the peripherals (e.g., screen, network cards, etc.), by the RAM, and by the residual CPU activity coming from it being ready to handle possible incoming loads.*

**Not zero!**

*For supercomputing facilities, this baseline power is usually quite high.*

**Still not zero!!!**

*While it is more striking for servers, this observation remains true for almost any device, but the residual consumption of turned off modern ICTs becomes negligible.*



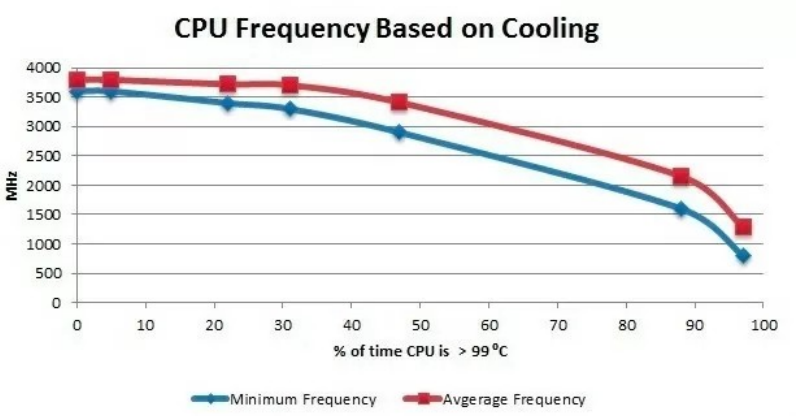

*From David Guyon*

When comparing **two applications**, the baseline can be subtracted.
When evaluating the **energy consumption** the total idle power must be redistributed to all users of the facility.

# Measuring the energy consumed by a computation

Before measuring the system power, we need to reduce **computing efficiency and power draw variability.**

During computation, the system will heat up, which can increase the power draw of the cooling solution and even lower the frequency automatically in order to reach thermal equilibrium.




**Before taking measurements, choose between the cold or hot state as your baseline.**

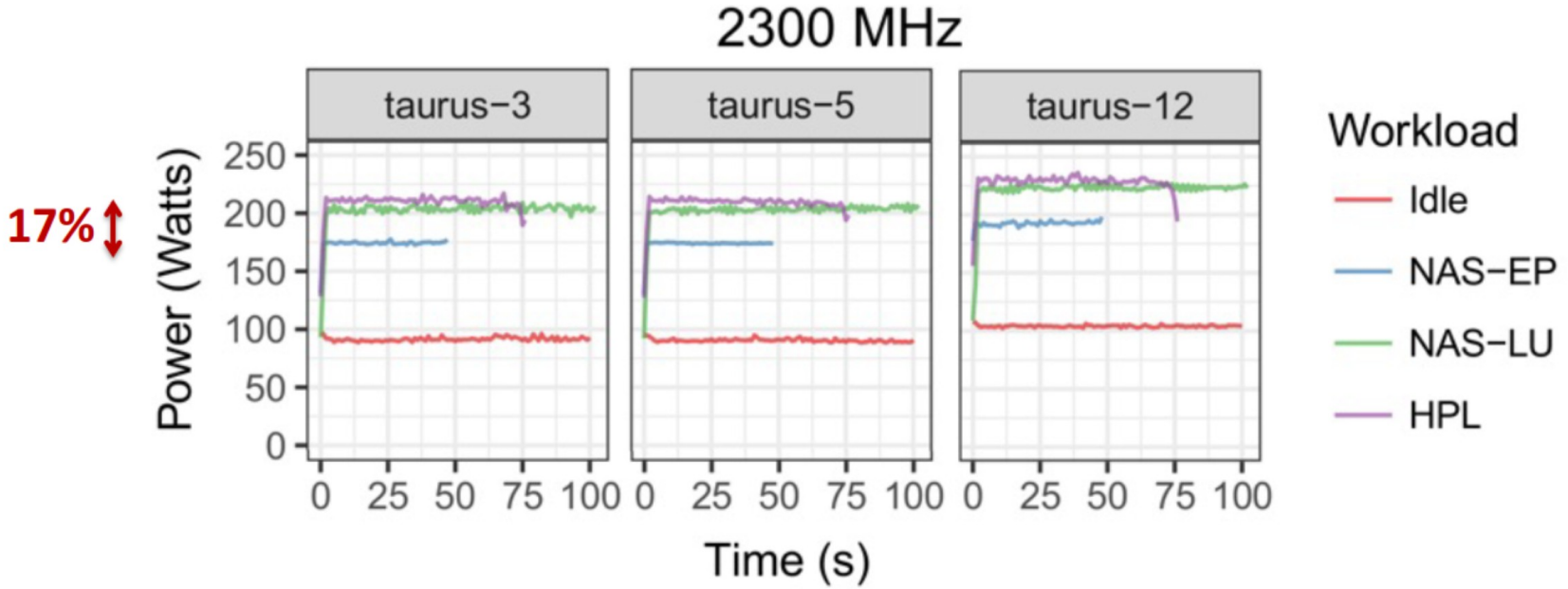
- **The cold state** is useful for comparing applications with short computation times, but it will likely underestimate the real energy consumption of the deployed software.
- **The hot state** (default) is a more accurate measurement but it increases the time and cost of the measurement as we need to « heat up » the system beforehand.

Once the system is in the right state, you can start measuring the **increase in power draw induced by the computation**. Then, the energy consumed by your computation can be approximated as:

$$E(J) = \Delta P(W) \times T(s)$$

*In practice, ΔP can vary during complex computations.*
*A more complete measurement would integrate the energy consumed over small timesteps.*

# Power draw application variability at 100% CPU usage

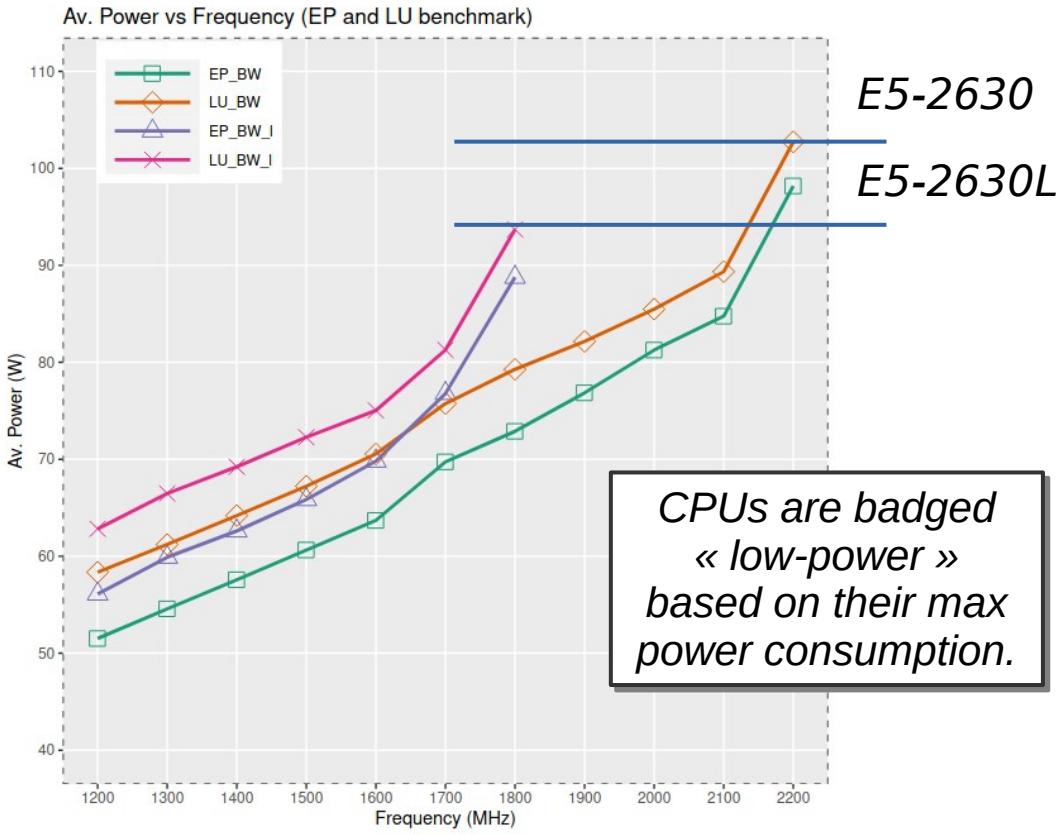


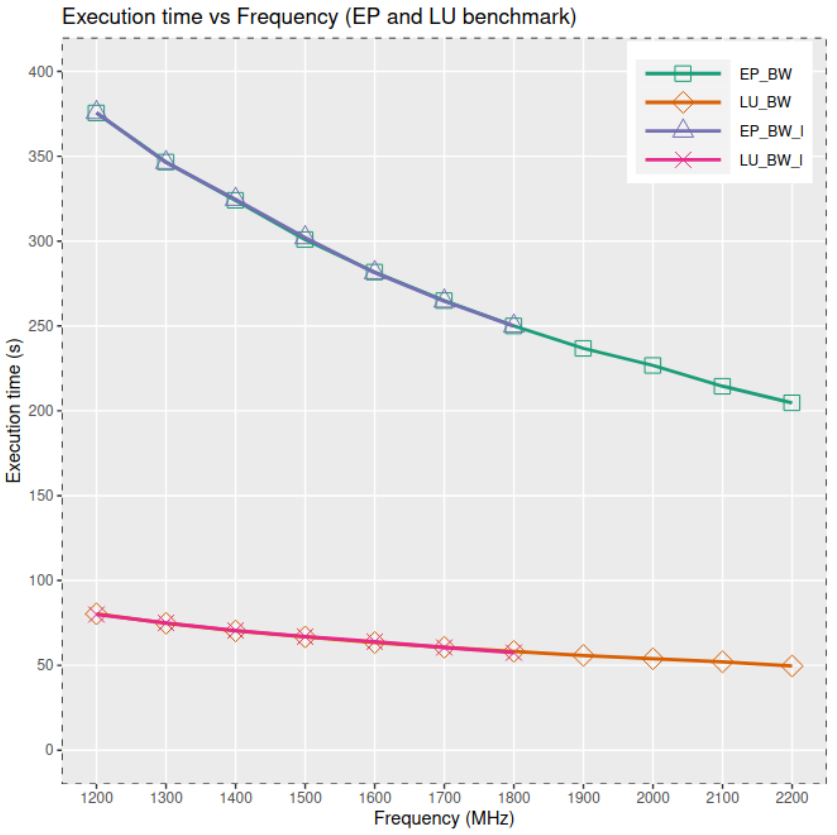

[Cluster 2017]

**17%** difference in consumption for applications fully loading the server.

*From Anne-Cecile Orgerie*

# Efficiency as a function of CPU frequency



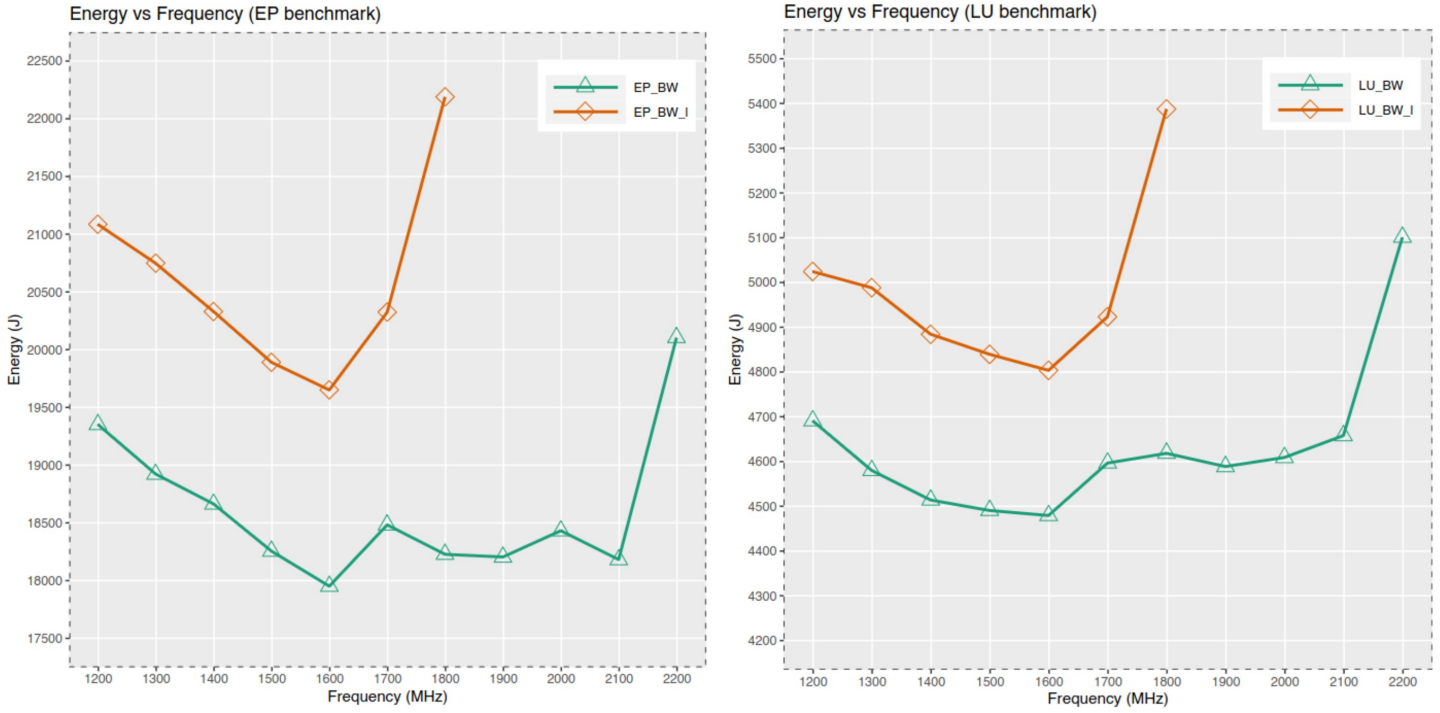




Increasing frequency increases the power draw BUT reduces execution time.
Where is the optimal point ?

*From Anne-Cecile Orgerie*

# Efficiency as a function of CPU frequency

*Total computation energy consumption*



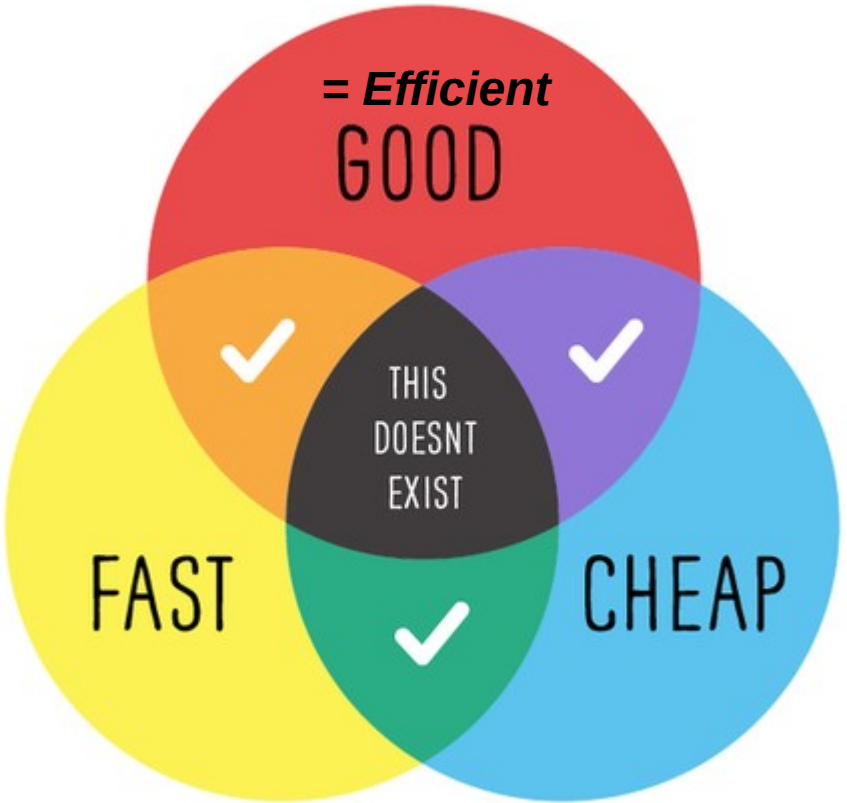

BW_l: Xeon E5-2630L v4 (Broadwell) -> *low power processor* (orange)
BW: Xeon E5-2630 v4 (Broadwell) (green)

[ISCC 2021]

*It is often possible to improve the energy efficiency of high end computing hardware by limiting the power draw or the maximum frequency directly. It might even allow to decrease the voltage while remaining stable, reducing even more the power draw.*

*From Anne-Cecile Orgerie*

# “You can’t have it all” says the founders

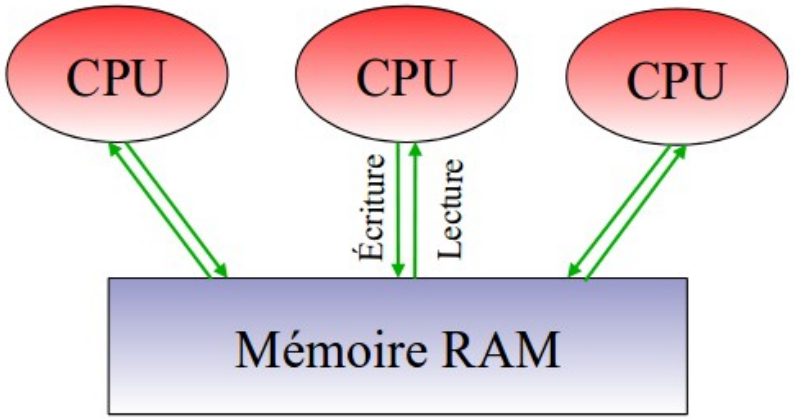


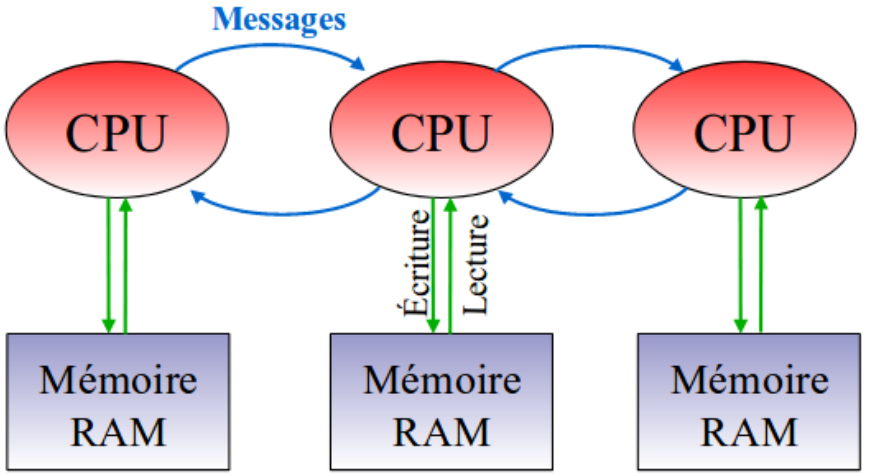

- **Flagship** chips are incredibly fast but costly and inefficient due to their high frequency and powerdraw.
- **Entry level** chips can be have low performances and be inefficient
- **Laptop** chips are usually efficient to improve the battery life but they are expensive and have lower performances due to cooling capabilities limit.
- **IoU** device chips are very efficient and relatively cheap but have very low performances
- **Server grade** chips usually consider the « good » grade to represent long term reliability. Then the performance per dollar is the second metric, and only at the end the efficiency.

→ **Efficiency is almost never a priority and is mostly seen as a way to reduce the deployed usage cost.**

# What to do when one CPU is not powerful enough?

**→ Have several CPU working on the same problem! Two possible solutions, having multiple CPU in the same system, and/or using many systems → server cluster.**



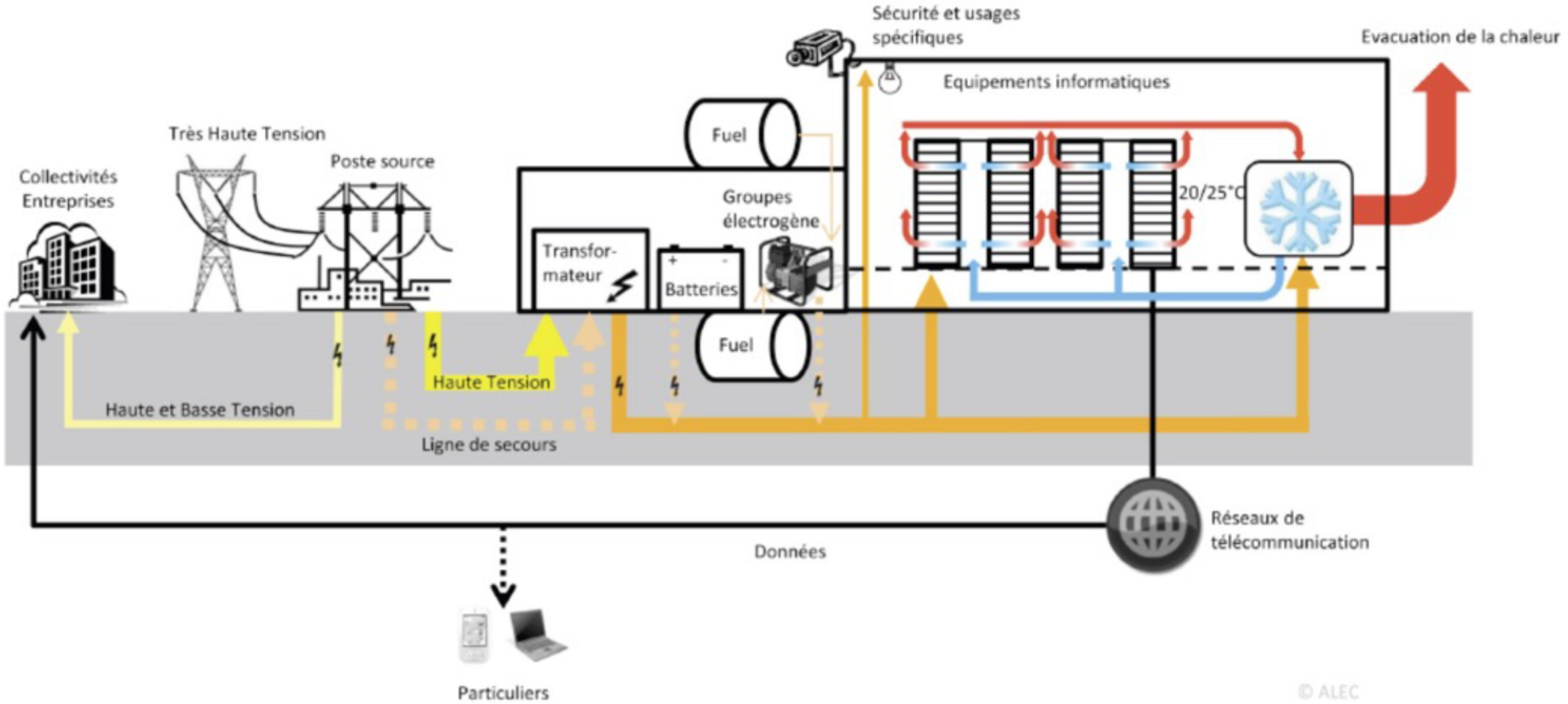

## Shared memory systems

- Parallelization induces small additional costs
- Possible memory access conflicts
- Costly hardware for many cores on the same system
- Reduced number of cores (usually < 256)

## Distributed memory systems

- Added cost due to communications
- Individual node is cheaper, but added cost of infrastructure
- Strongly impacted by the efficiency of the communication system
- Unlimited number of cores (theoretically)

# Cluster infrastructure additional impact




[Source : http://www.alec-plaineco.org/wp-content/uploads/2013/10/ALEC-Plaine-Commune-2013-Les-data-centers-sur-Plaine-Commune.pdf]

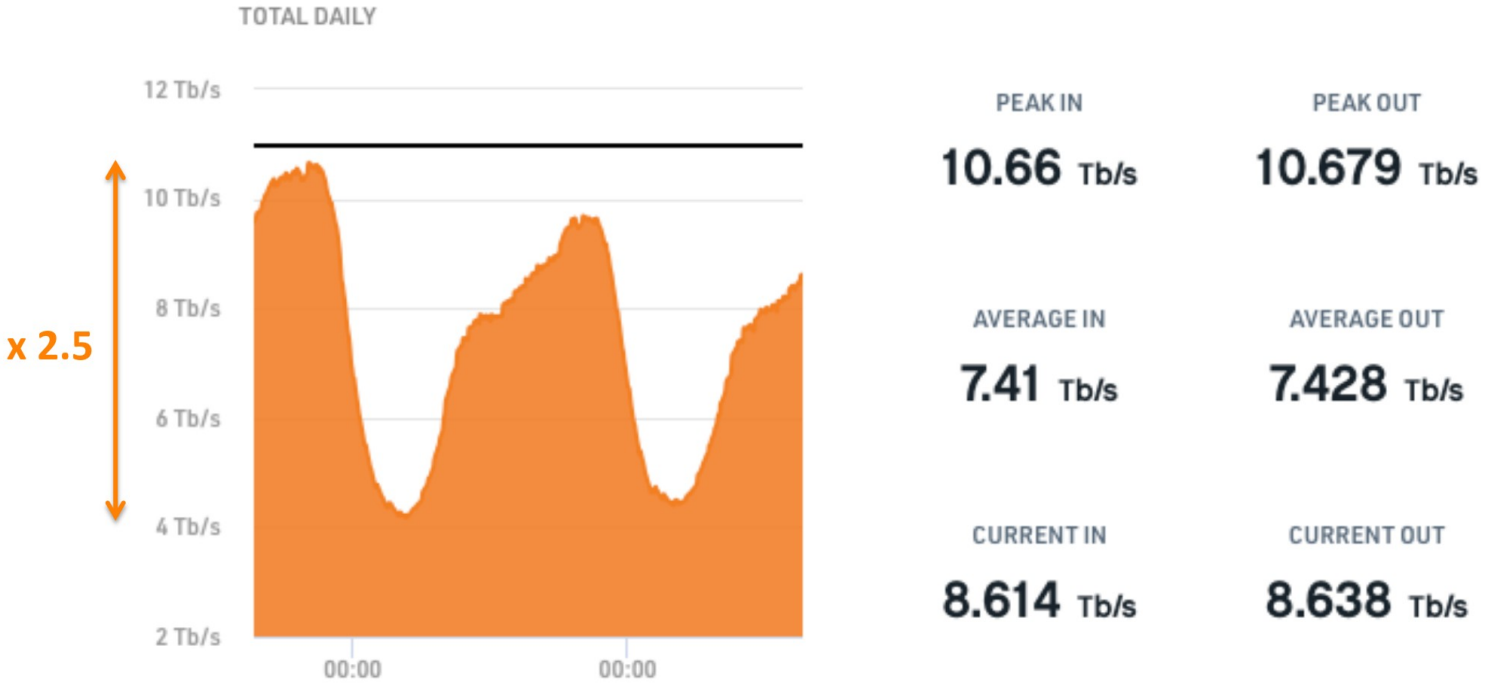
*From Anne-Cecile Orgerie*

# The problem of clusters / cloud infrastructure design

**→ For scientific clusters, the load is supposed to be almost always maximized.**

**→ For cloud servers and network infrastructures, the size of the facility is scaled on the maximum predicted load!**

The idle power consumption of servers being high, a lot of energy is wasted!




Daily aggregated traffic on AMS-IX(Amsterdam Internet eXchange Point), February 2022.

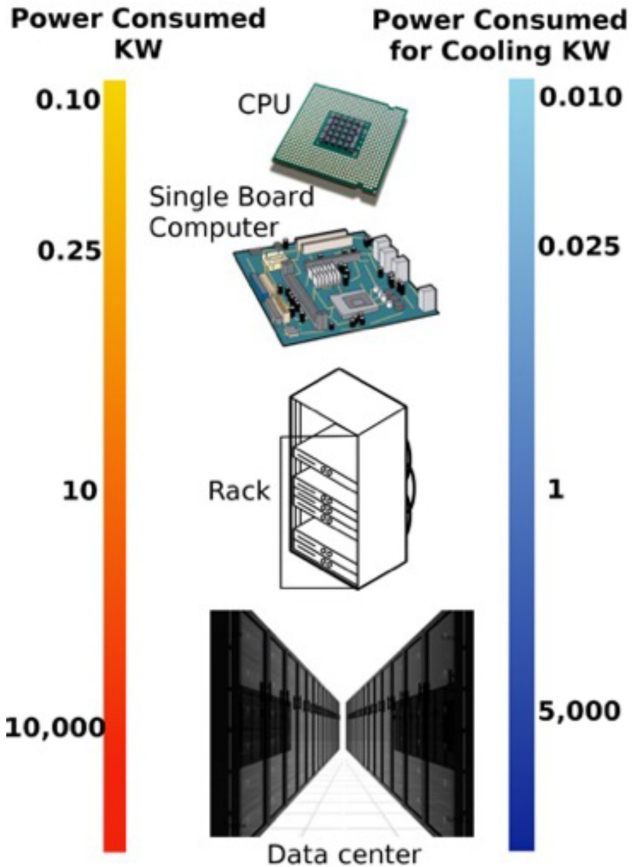
[Source : https://www.ams-ix.net/ams]

# Top 500 and Green 500 (Nov. 2023)

| Rank | System | Cores | Rmax (PFlop/s) | Rpeak (PFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 8,699,904 | 1,194.00 | 1,679.82 | 22,703 |
| 2 | **Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel<br>DOE/SC/Argonne National Laboratory<br>United States | 4,742,808 | 585.34 | 1,059.33 | 24,687 |
| 3 | **Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft<br>Microsoft Azure<br>United States | 1,123,200 | 561.20 | 846.84 | |
| 4 | **Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu<br>RIKEN Center for Computational Science<br>Japan | 7,630,848 | 442.01 | 537.21 | 29,899 |
| 5 | **LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>EuroHPC/CSC<br>Finland | 2,752,704 | 379.70 | 531.51 | 7,107 |
| 6 | **Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN<br>EuroHPC/CINECA<br>Italy | 1,824,768 | 238.70 | 304.47 | 7,404 |
| 7 | **Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 2,414,592 | 148.60 | 200.79 | 10,096 |

| Rank | TOP500 Rank | System | Cores | Rmax (PFlop/s) | Power (kW) | Energy Efficiency (GFlops/watts) |
|---|---|---|---|---|---|---|
| 1 | 293 | **Henri** - ThinkSystem SR670 V2, Intel Xeon Platinum 8362 32C 2.8GHz, NVIDIA H100 80GB PCIe, Infiniband HDR, Lenovo<br>Flatiron Institute<br>United States | 8,288 | 2.88 | 44 | 65.396 |
| 2 | 44 | **Frontier TDS** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 120,832 | 19.20 | 309 | 62.684 |
| 3 | 17 | **Adastra** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>Grand Equipement National de Calcul Intensif - Centre Informatique National de l'Enseignement Suprieur (GENCI-CINES)<br>France | 319,072 | 46.10 | 921 | 58.021 |
| 4 | 25 | **Setonix – GPU** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>Pawsey Supercomputing Centre, Kensington, Western Australia<br>Australia | 181,248 | 27.16 | 477 | 56.983 |
| 5 | 92 | **Dardel GPU** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>KTH - Royal Institute of Technology<br>Sweden | 52,864 | 8.26 | 146 | 56.491 |
| 6 | 8 | **MareNostrum 5 ACC** - BullSequana XH3000, Xeon Platinum 8460Y+ 40C 2.3GHz, NVIDIA H100 64GB, Infiniband NDR200, EVIDEN<br>EuroHPC/BSC<br>Spain | 680,960 | 138.20 | 2,560 | 53.984 |
| 7 | 5 | **LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>EuroHPC/CSC<br>Finland | 2,752,704 | 379.70 | 7,107 | 53.428 |

# How to improve efficiency and at which scale?




**→ More low power processors**
**→ Improvement of the IPC**

**→ More multi-core systems and improved efficiency**
**→ More dedicated accelerators (FPGA, GPU, NPU)**

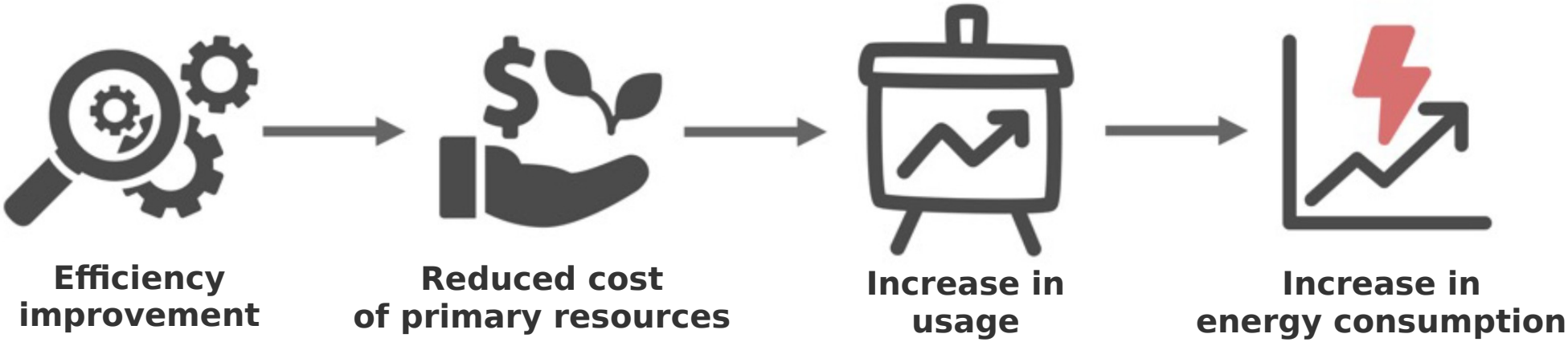
**→ Optimization of data placement**
**→ Systematize parallel computing**
**→ Improvement on the communication network**

**→ Improving the handling of bottlenecks**
**→ Strategies to smooth the load**
**→ Automatic scaling** (turning on and off parts of the cluster)

*From Anne-Cecile Orgerie*

**Improving data centers efficiency only is unlikely to be enough,**
**as illustrated by the ICT distributed contribution to the global CO2-e emissions.**

# Watch out for the rebound effect (Jevons' Paradox)!




Some optimizations will result in more accessible applications, increasing the number of users, filling any achieved saving and increasing the global demand.

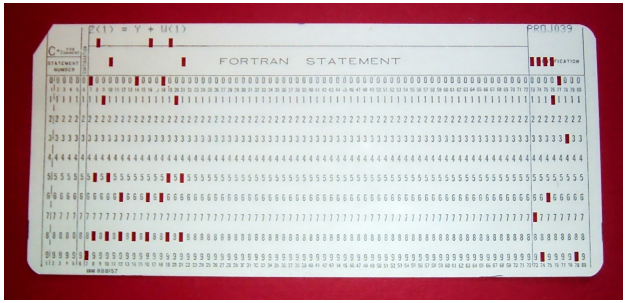
Optimizations on applications for which the demand is not changing much are the most beneficial → **but beware of the economic rebound!**

Any savings in one activity might be reinvested into other polluting applications!

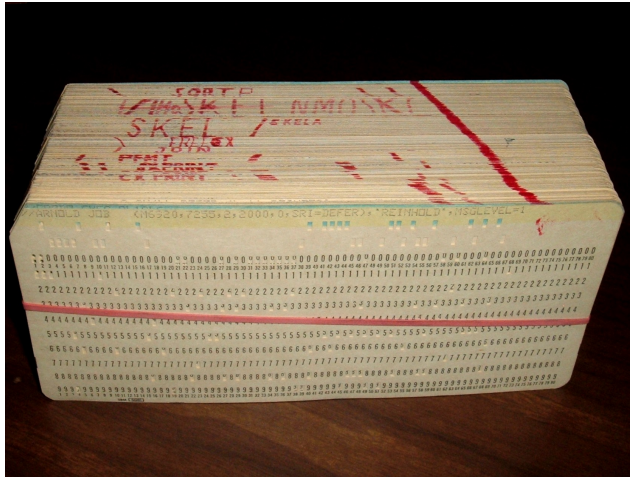
→ E.g., during the COVID pandemic, the investment in electronic devices exploded, not only due to homework but also to savings from the absence of other forms of usual expenses!

# Programming optimization in practice

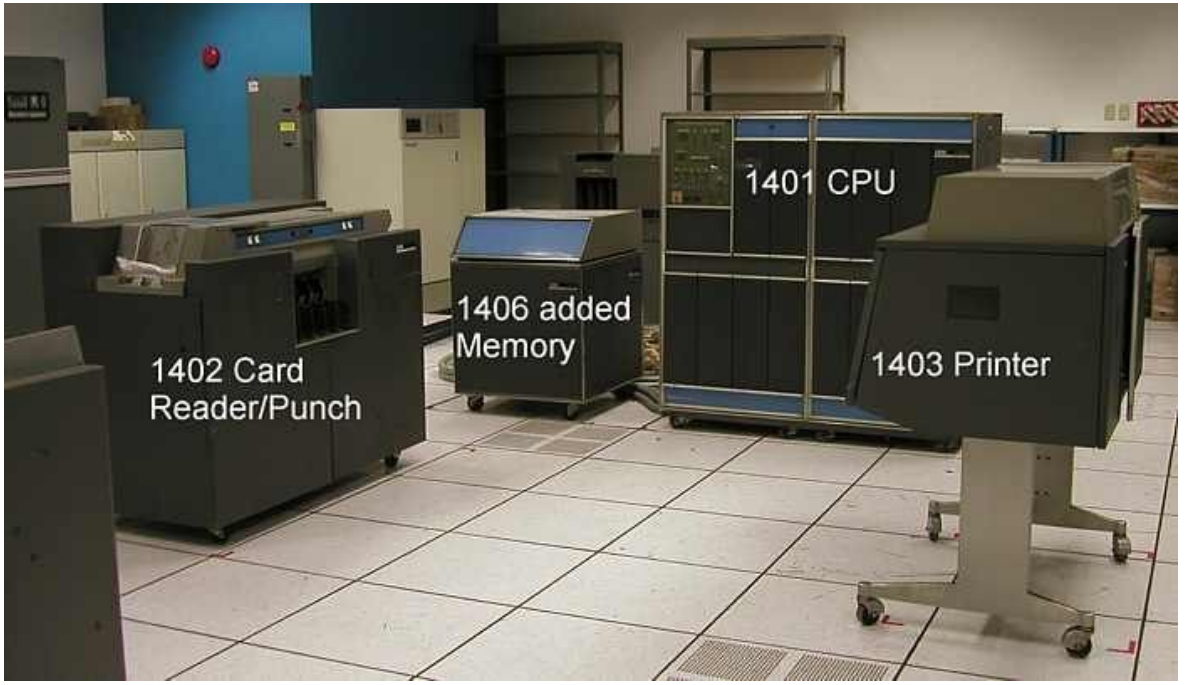
**Historically, optimization and good coding practices were a necessity!**




*A single Fortran instruction on a punched card*



*The deck of card for a full Fortran program*
*Each card represent a simple line of code*

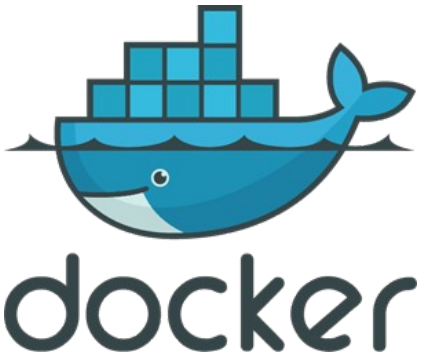



**An IBM 1401 spec sheet:**

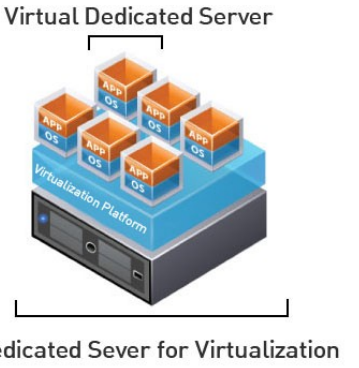
- 6-bit diode-transistor logic, with with a frequency of 90 Hz!
- 16000 Bytes of magnetic-core memory with the 1406 extension
- No storage by default, but can add a 1311 extension to support 2MB of non-volatile memory

# Current programming trends are not efficiency driven

**Interpreted or scripting programming languages**
**→ Execution overheads**




**Everything in containers**
**→ De-multiply core software**



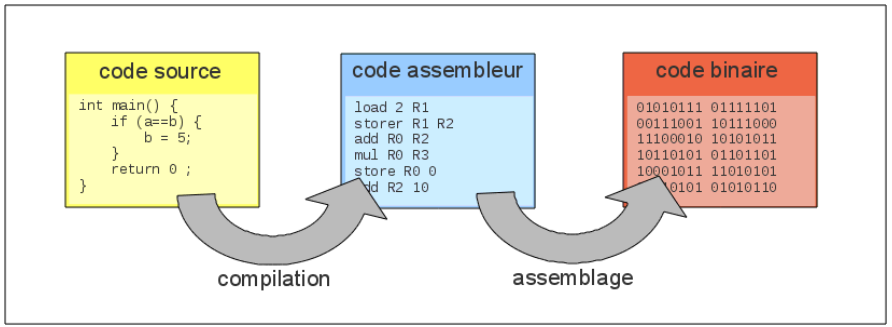

**Virtual machines**
**→un-optimized loads**

**These days, the focuses are instead:**

- Reducing development time (developers are costly)
- Ease of use and deployment (reduce friction)
- Improving security
- Constant accessibility (service continuity)
- Reduction of infrastructure costs (mutualize resources)

# Compiled VS interpreted programming languages

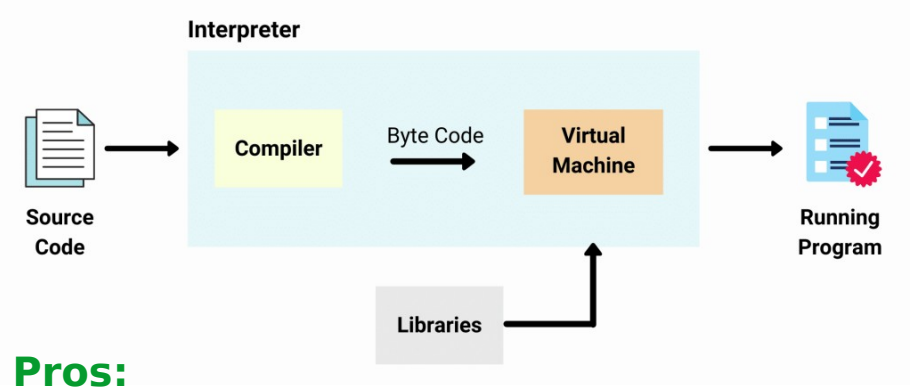
## Compiled languages
### (C, Fortran, Pascal, Rust, ...)




**Pros:**
- Very fast! Thanks to the large scope of the compiler allowing strong optimization.
- Low level. Closer to the machine language, exposes more easily the system structure.

**Cons:**
- Strict syntax, and can't run if any error is detected by the compiler
- More error prone as more control also induces more responsibility (e.g., memory management)

## Interpreted languages
### (Python, PHP, Ruby, Javascript, ...)



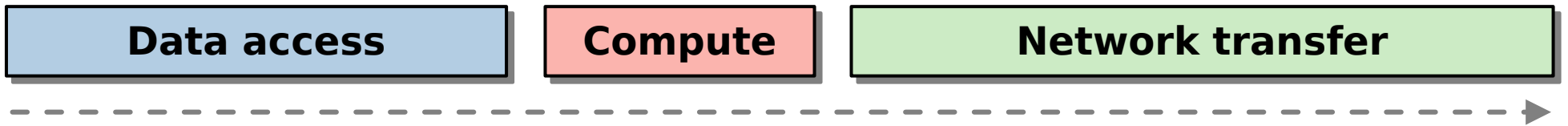

**Pros:**
- Compliant syntax
- Usually "high level", do complex things in a few lines without noticing

**Cons:**
- Slower due to on-the-fly compilation overhead and to the small scope of the interpreter
- Hide the inner workings of the system, reducing possible optimizations and global understanding of the program requirements in terms of resources and environment.

*Energy consumption comparison between programming languages → Pereira et al. 2021*

# Performance bottleneck

**The speed of a program or a system is most of the time limited by its slower element!**

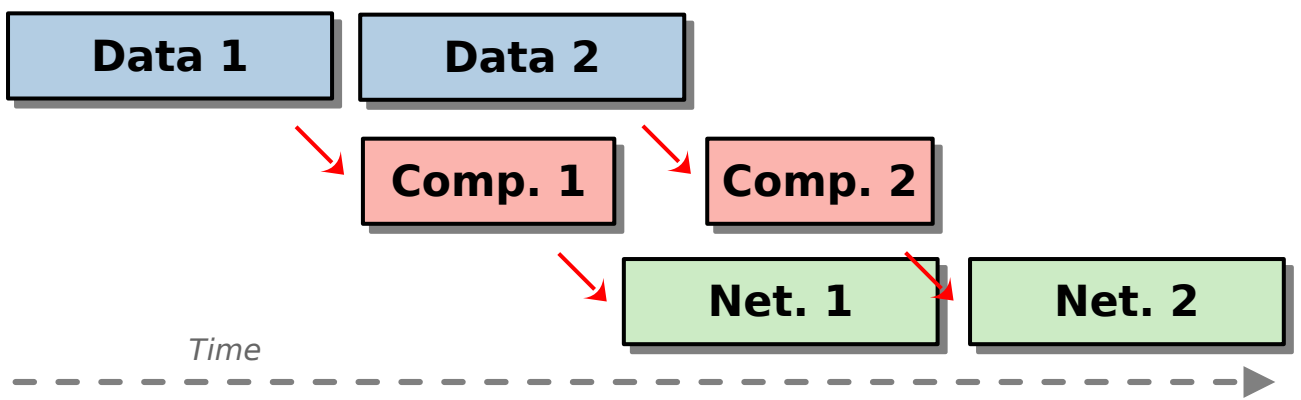



With this distribution, optimizing the “compute” part is not very useful.

**Still, it is possible to transfer stress from one sub-system to an other to some extent.**

*E.g., data to be transferred through the network can be compressed first to reduce their size. The compression time now adds to the compute part, and can potentially be optimized.*

**Also, some parts can be executed concurrently!**




**Splitting problems into concurrent independent tasks that can overlap is one of the keystones of computing resources optimization!**

**Waiting times of each sub-system must be reduced to the minimum.**

# Random Access Memory

**RAM = constant access time regardless of the data placement**

*In contrast to disks, magnetic tapes, or other storage media that use a moving read head. The access time is then dependent on the distance between the last read and the next one.*

**The RAM stores data that will need fast or frequent access and must therefore be fast.**

Fast “flash” memory is expensive to produce and requires significant computing power to manage. Therefore, the amount of RAM in a system is usually not higher than a few hundred GB, which is generally lower than its storage capacity.

*Flash memory is also used in non-volatile storage like SSDs or USB keys, but their access speed is lower, as is the price per GB. A good programming paradigm will only load relevant data in RAM.*

# Memory access principle

From the system standpoint, memory is **linear, continuous, and decomposed into cells,** each one referred to by **an address** stored directly into the Memory Management Unit (MMU)**.**

Programs reserve **chunks of memory** to work. They can be **discontinuous**.


*The memory granularity can go up to 1 bit, but better performances are obtain when handling blocks that are at least the system architecture data size (32 or 64 bits).*

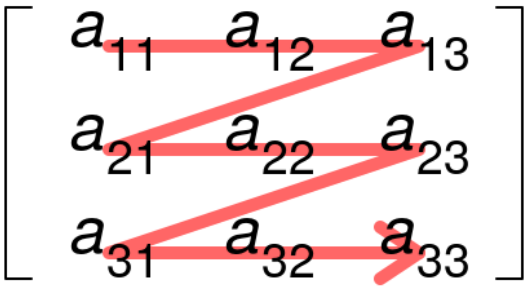
*Typical integers or float variables are 32 or 64 bits (4 or 8 bytes) and can be stored anywhere in memory. Single variable access usually don't expose the address to the programmer.*

**For data arrays, memory continuity is of the outermost importance.** Therefore they are reserved as continuous chunks and referred to by the address of their first element.


Storing a 2D matrix of 4 by 5 elements requires a continuous chunk of 20 elements. After declaring the array, we only get access to its starting position address and data type. Recovering an element is then a matter of address arithmetic.

# Multidimensional data in linear storage

Row-major order

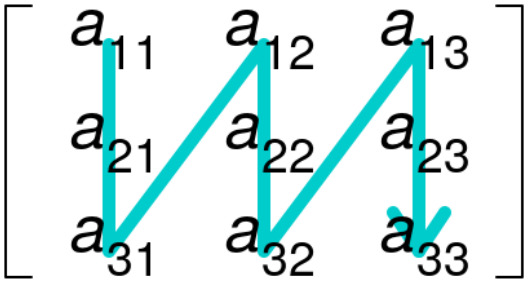



**Default order for C, C++, Python, etc…**

In a[i][j] the “fast” index on continuous data is the one on the **right (j)**

**When flattened a[i*3+j]**

Column-major order




**Default order for Fortran**

In a[i][j] the “fast” index on continuous data is the one on the **left (I)**

**When flattened a[i+j*3]**

In practice, the programmer is free to use the encoding he wants by directly accessing the flattened array for storing and loading data.

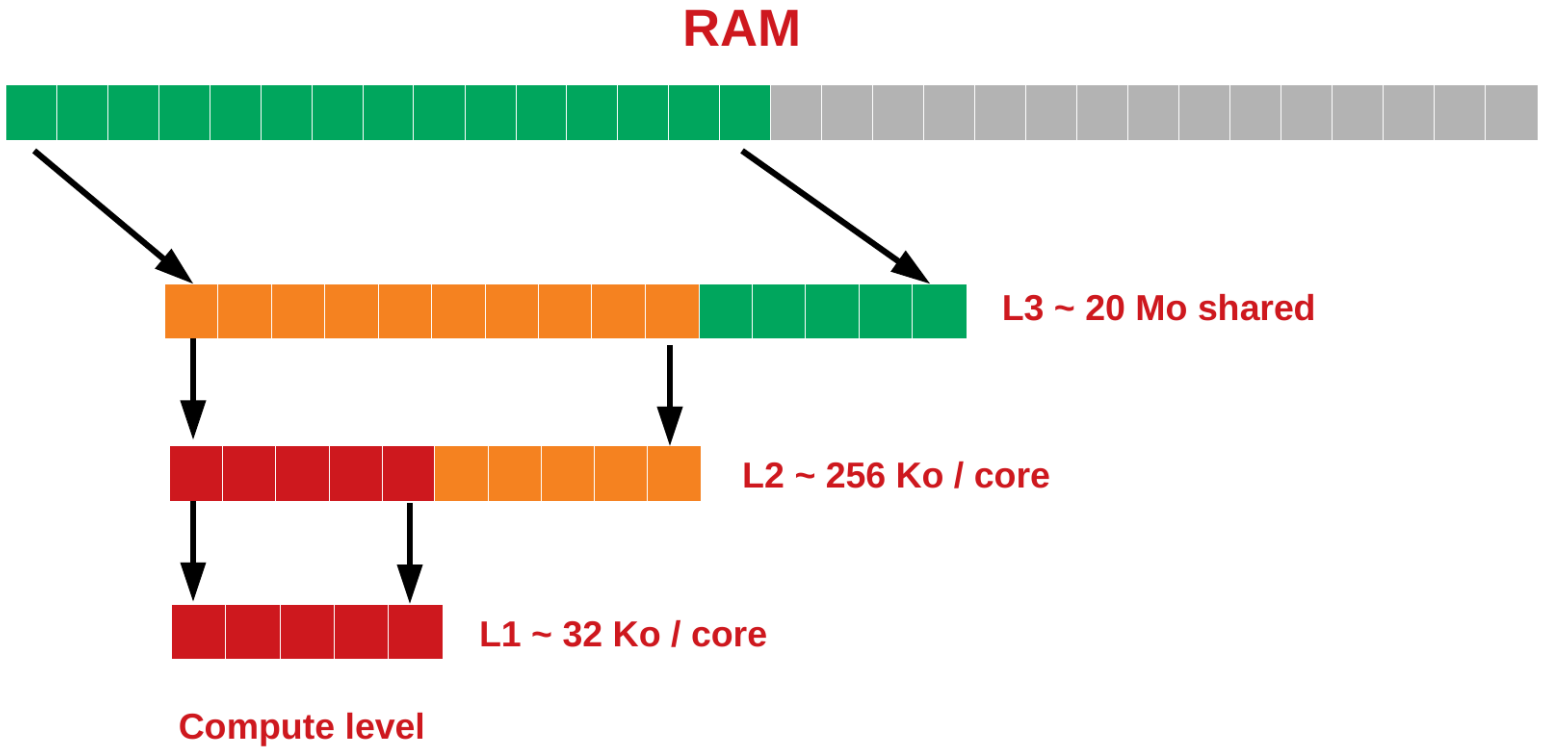
*We note that it is possible to use “linked list” so the data are continuous only for one dimension. While it can ease programming, it usually has strong adverse effects on performances.*

# Data access and “cache miss”

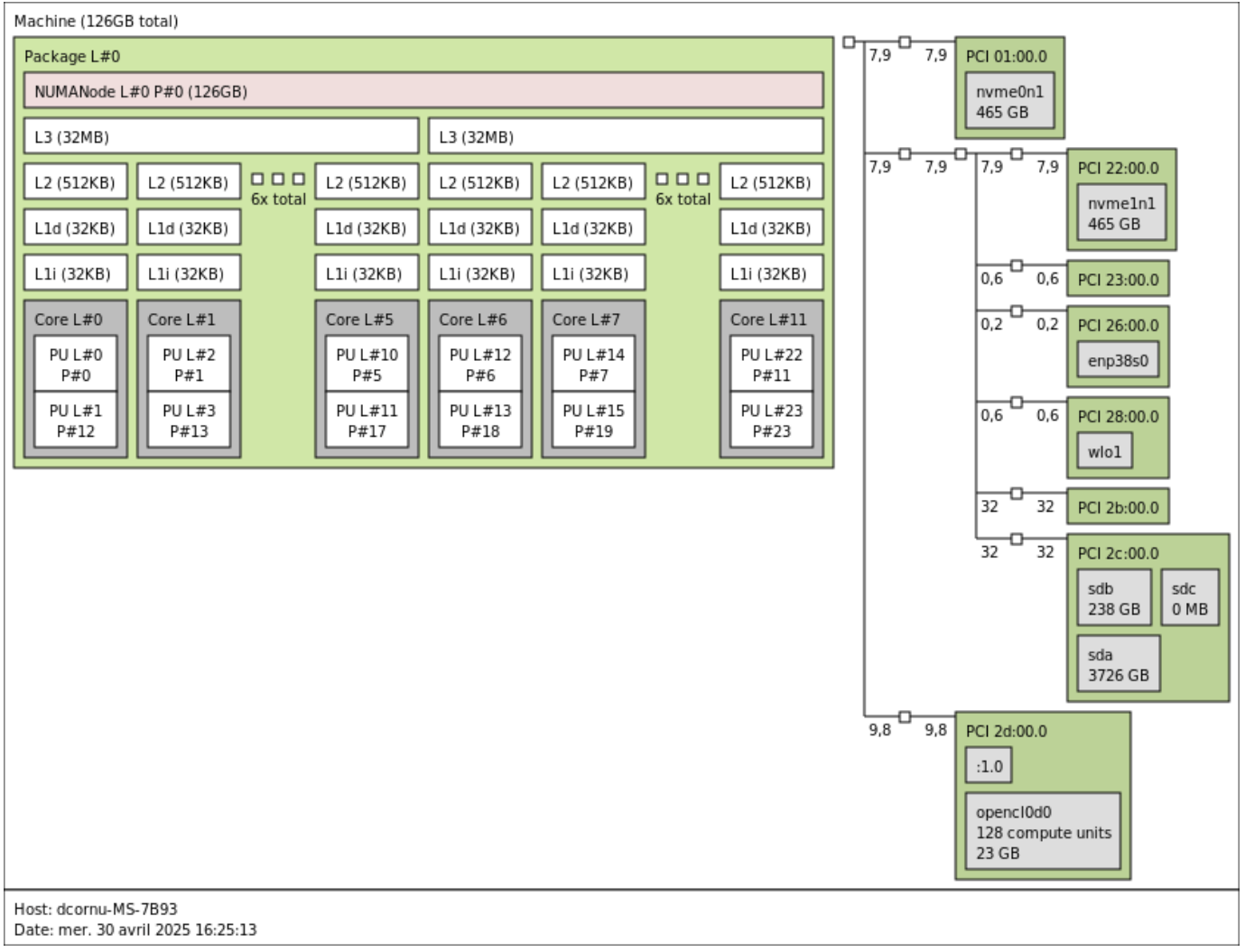
CPUs are equipped with « caches » that are **successive levels of faster and smaller memory.**

Accessing a specific data element from the RAM will actually **load a full chunk of data into the caches starting from this element.** This way, if the next operation requires the next element in memory, it can simply use the cached value instead of loading it from the RAM.

Breaking this access scheme will result in a **cache miss**, which means that data has to be read from the RAM more than necessary. This is typically the case when accessing a 2D array in the wrong order.



# Memory hierarchy and topography




System memory hierarchy usually refers to the different levels of system memory and how they are interconnected.

On this topography, we can see that the **RAM is shared** for all subsystems.

On our test system (Ryzen 9 5900X), the 12 cores are spread over two CCX that have a **distinct shared L3 cache**. Then, **L2 and L1 caches are dedicated to each core.**

**The effective topography of the system always impacts performance.**

# Let the compiler do the work!

**The compiler is the best friend you can have regarding optimization.**

CPUs are complicated and comprise **a lot of advanced instructions** (see CISC and RISC) that can perform complex operations in a **reduced amount of cycles.** Using them explicitly require meticulous programming.

Luckily, we can ask the compiler to do the work of converting simple codes to strongly optimized ones using all the available instructions. For this the compiler has to analyze large parts of the code to identify dependencies. It will then perform only the authorized optimizations.

While individual aspects of optimization can be specified, we usually rely on **the general -O flag**, with 3 different levels from -O1 to -O3.

*These optimizations depend on the compiler! We only list them for GCC here.*

*From -O1 to -O3*

```
-fauto-inc-dec
-fbranch-count-reg
-fcombine-stack-adjustments
-fcompare-elim
-fcprop-registers
-fdce
-fdefer-pop
-fdelayed-branch
-fdse
-fforward-propagate
-fguess-branch-probability
-fif-conversion2
-fif-conversion
-finline-functions-called-once
-fipa-pure-const
-fipa-profile
-fipa-reference
-fmerge-constants
-fmove-loop-invariants
-freorder-blocks
-fshrink-wrap
-fshrink-wrap-separate
-fsplit-wide-types
-fssa-backprop
-fssa-phiopt
-ftree-bit-ccp
-ftree-ccp
-ftree-ch
-ftree-coalesce-vars
-ftree-copy-prop
-ftree-dce
-ftree-dominator-opts
-ftree-dse
-ftree-forwprop
-ftree-fre
-ftree-phiprop
-ftree-sink
-ftree-slsr
-ftree-sra
-ftree-pta
-ftree-ter
-funit-at-a-time
```

*From -O2 to -O3*

```
-fthread-jumps
-falign-functions  -falign-jumps
-falign-loops  -falign-labels
-fcaller-saves
-fcrossjumping
-fcse-follow-jumps  -fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize -fdevirtualize-speculatively
-fexpensive-optimizations
-fgcse  -fgcse-lm
-fhoist-adjacent-loads
-finline-small-functions
-findirect-inlining
-fipa-cp
-fipa-bit-cp
-fipa-vrp
-fipa-sra
-fipa-icf
-fisolate-erroneous-paths-dereference
-flra-remat
-foptimize-sibling-calls
-foptimize-strlen
-fpartial-inlining
-fpeephole2
-freorder-blocks-algorithm=stc
-freorder-blocks-and-partition -freorder-functions
-frerun-cse-after-loop
-fsched-interblock  -fsched-spec
-fschedule-insns  -fschedule-insns2
-fstore-merging
-fstrict-aliasing -fstrict-overflow
-ftree-builtin-call-dce
-ftree-switch-conversion -ftree-tail-merge
-fcode-hoisting
-ftree-pre
-ftree-vrp
-fipa-ra
```

*-O3 only*

```
-finline-functions
-funswitch-loops
-fpredictive-commoning
-fgcse-after-reload
-ftree-loop-vectorize
-ftree-loop-distribute-patterns
-fsplit-paths
-ftree-slp-vectorize
-fvect-cost-model
-ftree-partial-pre
-fpeel-loops
-fipa-cp-clone
```

# The matrix multiplication operation

$$C(i, j) = \sum_k A(i, k) \times B(k, j)$$

**Matrix operations are of uttermost importance**

**→ They appear in a lot of computing problems!**

**→ They represent the vast majority of the computations done by AI models.**

Matrix operations have received much **optimization attention.** Chip manufacturers even built **dedicated hardware and instructions** for this operation!

The default algorithm **complexity scales with the cube of the matrix side size**
→ if M=N=K, it scales in $\Theta(N^3)$.

*Algorithms with a lower complexity exist, but they often work in a way that makes them challenging to optimize for classical computing hardware.*

Many libraries are dedicated to matrix **multiplication or other linear algebra operation (BLAS) acceleration** using various types of hardware:

**OpenBLAS, IntelMKL, MAGMA, CuBLAS, rocBLAS, ...**

# First practical work: Matrix multiplication optimization

$$C(i, j) = \sum_k A(i, k) \times B(k, j)$$

## PW objectives and strucuture:

- Implement and benchmark a naive matrix multiplication in Python. Compare to built-in implementations in Python and Numpy.
- Explore the effect of compilation on execution time, first with JIT Python compilation then moving to a C implementation
- Implement increasingly complex version of the matmul operation in C that makes proper use of the system design and memory hierarchy.
- Compare different implementations against the reference low-level OpenBLAS library.
- Explore CPU parallelization with OpenMP and its impact on energy consumption.
- Discover matmul usage on GPU with examples of basic and advances implementation and compare the energy consumption against CPU.

*This PW mostly follows Algorithmica by Sergey Slotin.*

# Optimization: Memory continuity

$$C(i, j) = \sum_k A(i, k) \times B(k, j)$$

**The naive implementation is composed of 3 loops.** The first two over the i and j index specify the coordinates of a cell in the C matrix, and the last one indexed by k spans over the shared dimension between A and B. This naive implementation is memory bandwidth limited as the accessed elements are not continuous, **inducing cache miss.**

$$C(i, j) = \sum_k A^T(k, i) \times B(k, j)$$

The first main optimization that can be done is to **transpose the matrix that is responsible for the cache-miss**, the A matrix in our case (with column-major encoding).

# Optimization: SIMD vectorized operations

When **accessing continuous data in memory,** the operations can be vectorized using specific instruction sets that follows the **SIMD principle**.

E.g., in **matmul_v3** the main instruction **`C[j*M+i] += A`$^{T}$`[i*K+k] * B[j*K+k]`** can be expressed as a **SIMD operation** using the **FMA (Fused Multiply Add) instruction from the AVX2** set that will multiply, add, and round using a reduced number of CPU cycles.

This type of operation works on **vector data structures**:

```
typedef float vec __attribute__ (( vector_size(32) ));
```

Using AVX-2, **the vector size is 256 bit, so it can store 8 floats of 32 bits each**.

When doing operations on vector data, the compiler will automatically use the vectorized FMA operation.

**An explicit vectorized matmul that uses vectorized versions of A and B and then does the accumulation on the individual vectors is presented in matmul_v4.**

# Optimization: CPU register re-use

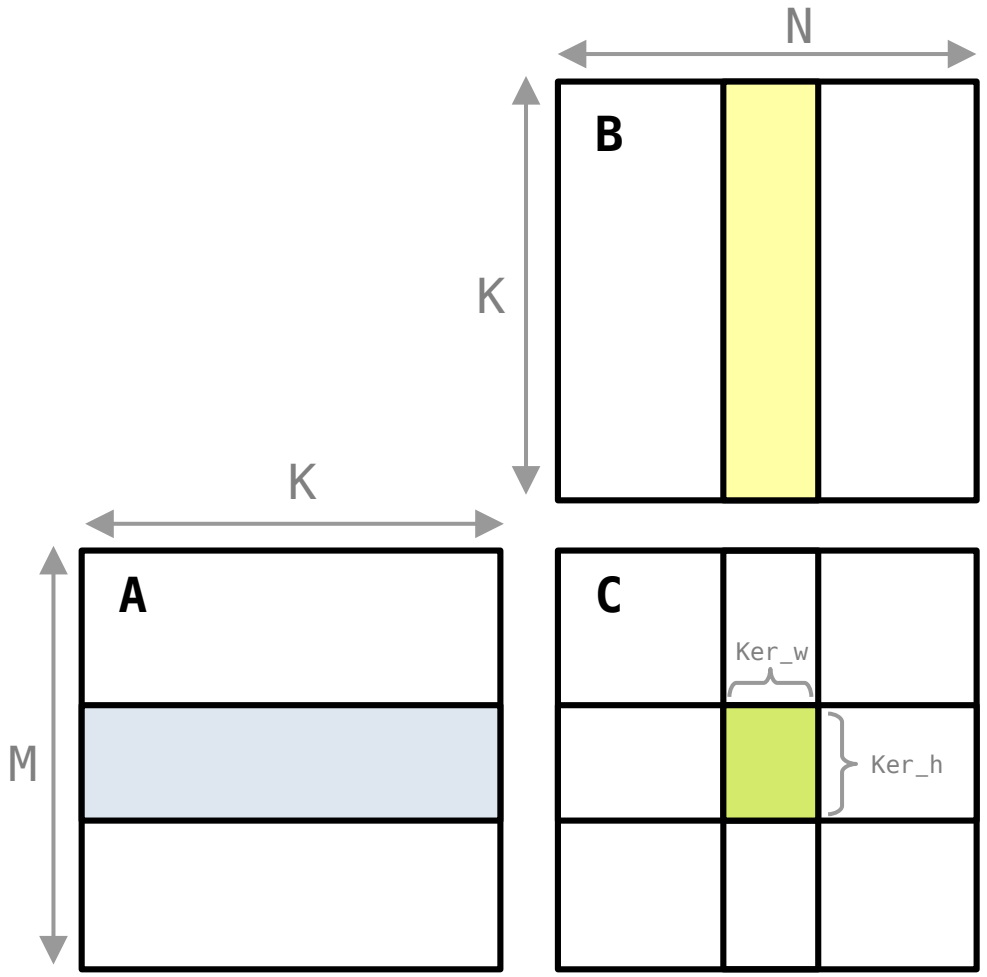
Even with continuous memory access, the implementation is likely to remain **limited by the memory bandwidth** due to the **high number of data reading and writing** that need to be done in contrast to compute operations.

**We must reduce data movement and maximize the use of CPU caches while letting the compiler do the hard work as much as possible.**

One approach is to ensure that data that are loaded into the **CPU register are reused as much as possible** before being replaced by others.

This can be combined with a **SIMD vectorized computation inside a kernel** that is tasked to compute the result for a sub-matrix **`C[i:i+di][j:j+dj]`.**

# Optimization: SIMD vectorized kernel computation



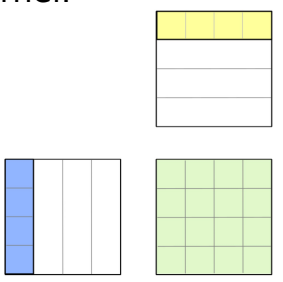

Instead of accumulating the product of all the K elements for a row in A and a column in B at once, we can use a progressive accumulator. The objective here is to reuse loaded data for all the computations they contribute to instead of reloading all elements for filling each cell of C.

The strategy is based on a kernel of size ker_h (rows of A), and ker_w (columns of B). For each value of k, the kernel takes a vertical vector in A and an horizontal vector of B inside the sub regions defined by the kernel.

Doing the dot product of these two vector we can recover their contributions to each cell of C and accumulate it into temporary memory space of the size of the kernel.

Accumulate version (strong re-use!)

$$
\begin{aligned}
C_{0,0} &= A_{0,0}B_{0,0} + A_{0,1}B_{1,0} + A_{0,2}B_{2,0} \\
C_{0,1} &= A_{0,0}B_{0,1} + A_{0,1}B_{1,1} + A_{0,2}B_{2,1} \\
C_{0,2} &= A_{0,0}B_{0,2} + A_{0,1}B_{1,2} + A_{0,2}B_{2,2} \\
C_{1,0} &= A_{1,0}B_{0,0} + A_{1,1}B_{1,0} + A_{1,2}B_{2,0} \\
C_{1,1} &= A_{1,0}B_{0,1} + A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\
C_{1,2} &= A_{1,0}B_{0,2} + A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\
C_{2,0} &= A_{2,0}B_{0,0} + A_{2,1}B_{1,0} + A_{2,2}B_{2,0} \\
C_{2,1} &= A_{2,0}B_{0,1} + A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\
C_{2,2} &= A_{2,0}B_{0,2} + A_{2,1}B_{1,2} + A_{2,2}B_{2,2}
\end{aligned}
$$



Naive version (no re-use)

# Optimization: SIMD vectorized kernel computation

Inside the kernel, for a given value of K we must compute a set of dot products that can simply be written as: **`C[i,j] += A[i] * B[j]`**

This can also be expressed as a SIMD FMA instruction. To **saturate the execution port** of this instruction we must work on at least 10 registers (instruction-level parallelism).

**The maximum number vector type register in AVX2 is 16**, so we define a kernel with **ker_w = 16 and ker_h = 6**, for a total of 16/8 x 6 = 12 vector registers to keep a margin.

**For each position k, the kernel must contain two loops:**

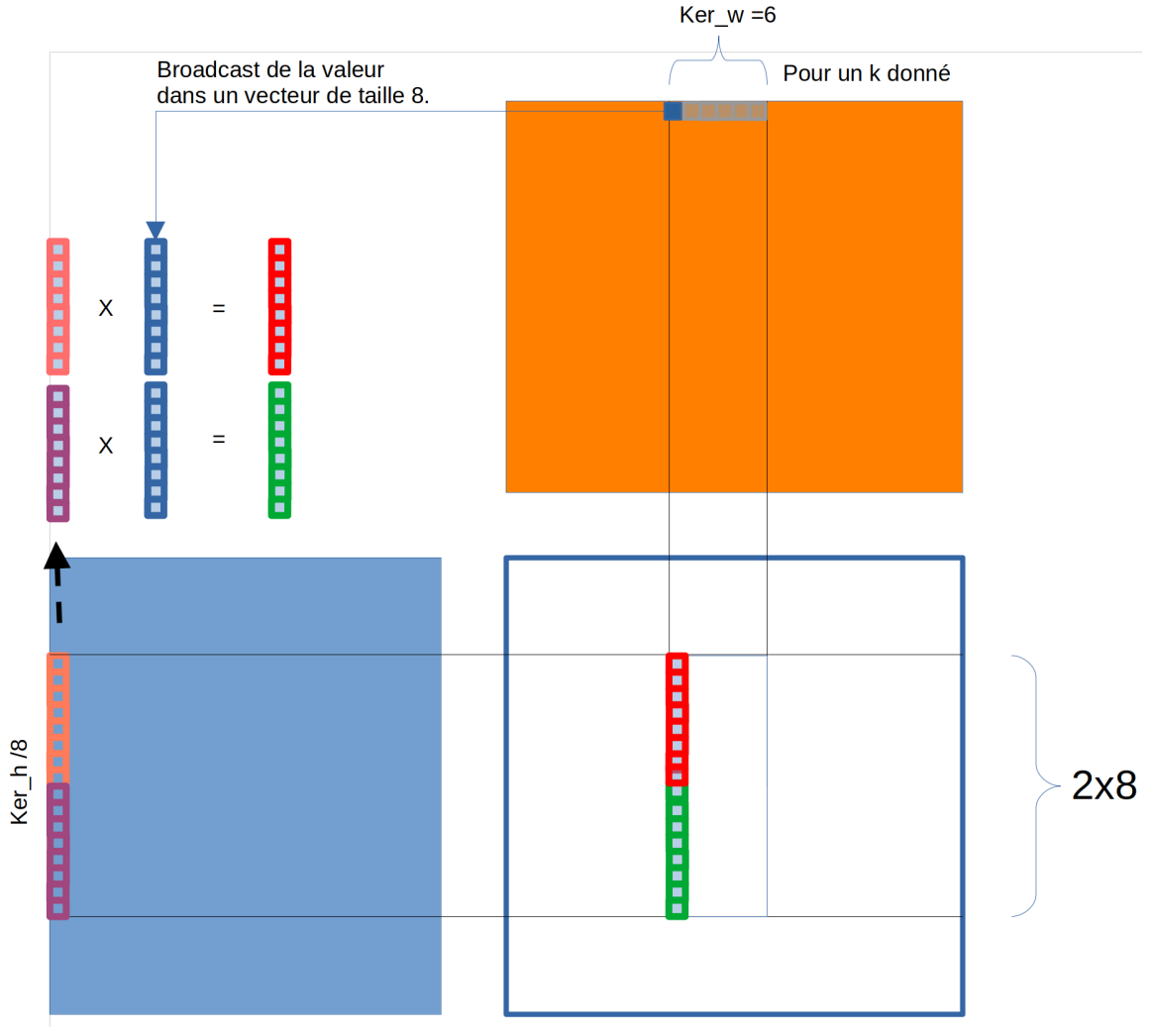
1) A loop over the 6 element of the sub-row from B. At each step, the current value must be broadcast to a vector with 8 identical values.

2) A loop over the the two vectors that contains the 16 elements of the sub column from A.

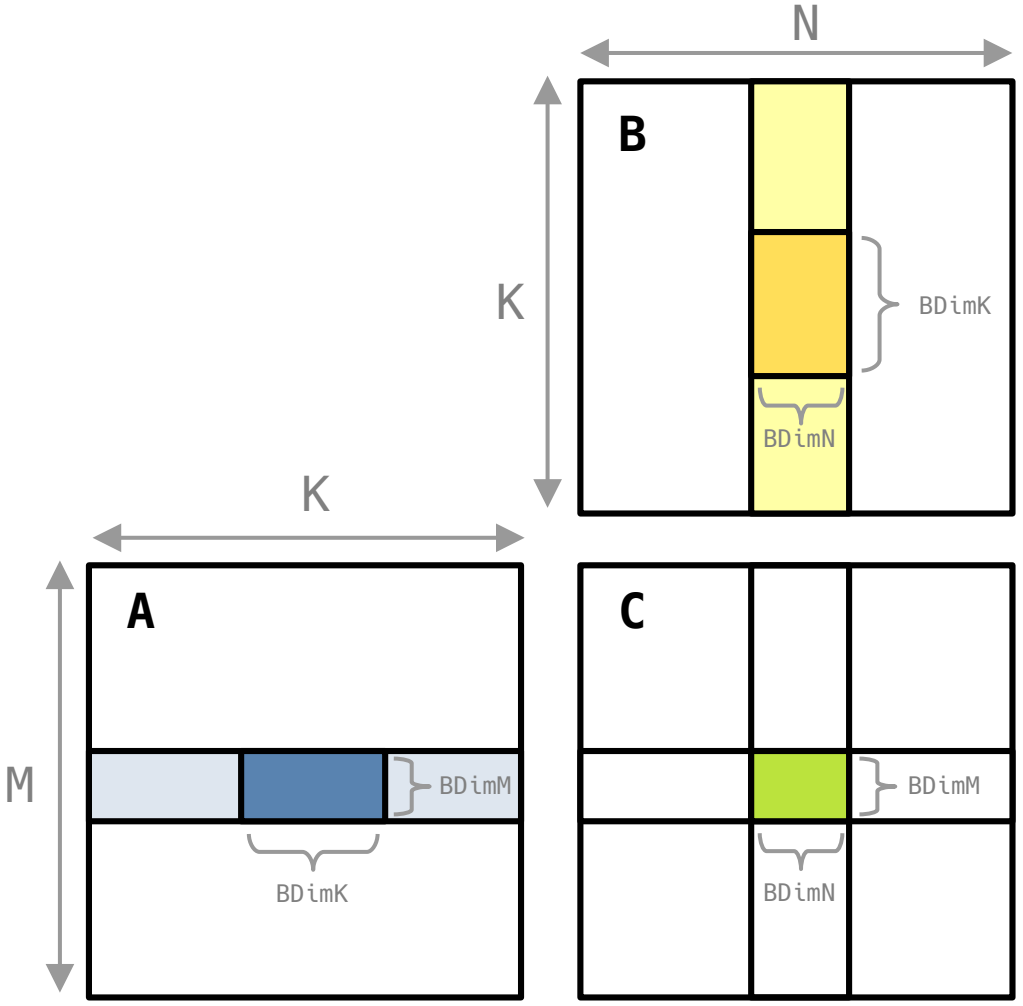
3) The two selected vectors can be multiplied to accumulate 8 products in the kernel corresponding to 8 different positions in **`C[i:i+8,j] += A[i:i+8] * B[j]`**

**This kernel itself must be called in a double loop for all its possible positions in C.**

# Optimization: SIMD vectorized kernel computation



# Optimization: Fully blocked version with kernel




**The previous version is still bandwidth limited!** To further optimize we must maximize re-use of the data loaded in the different CPU caches by working on **blocks of the matrices**. This is similar to a divide-and-conquer strategy.

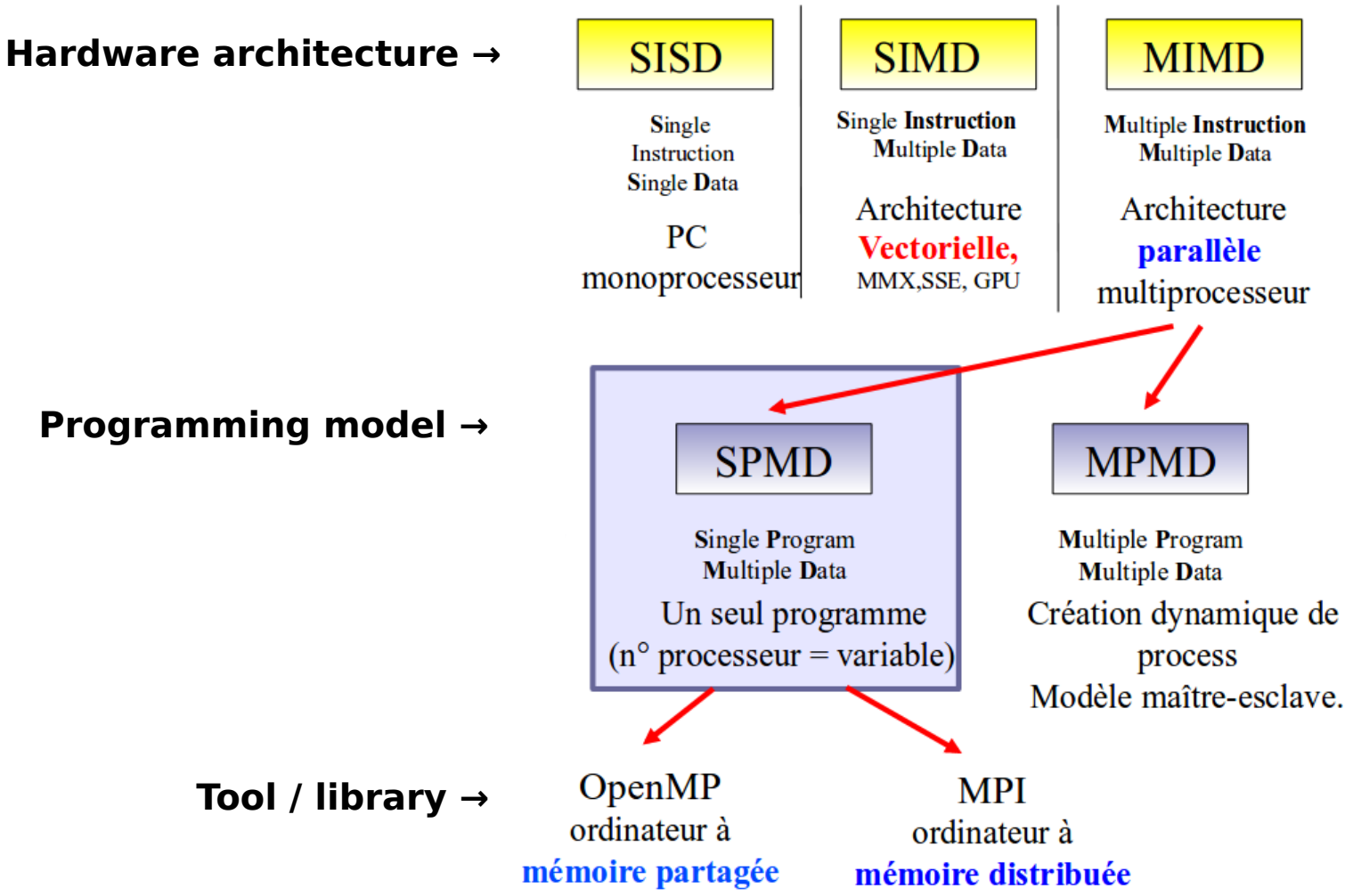
To ensure we work on a subproblem that fits in the caches, we can define a **subregion of C (BDimN, BDimN)** with a size that is a multiple of our kernel dimension. Then instead of working on the whole K dimension at one, we define a **third dimension BDimK** to work only on **sub-regions of A and B**.

Since we don't work on all K at once, the kernel must be modified to take the **start and end k positions** of the current block in the K dimension, and **accumulate the results in C** rather than setting the values.

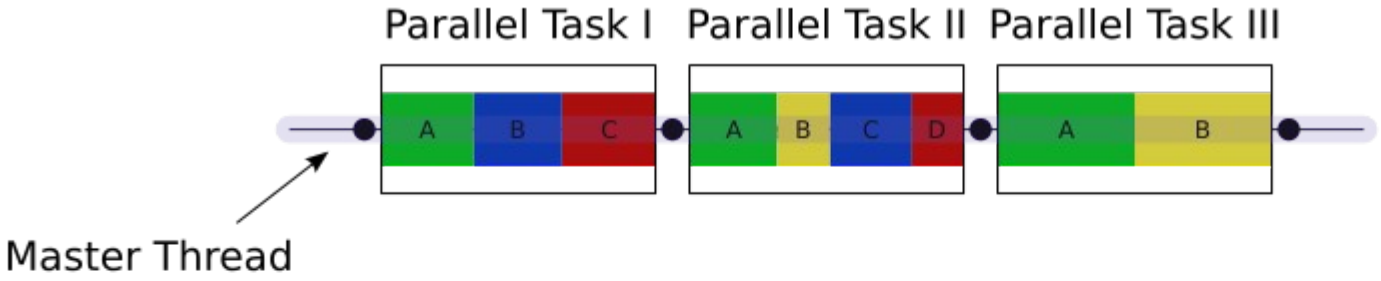
Now, the full matumul operation can be done by looping over sub-blocks with:

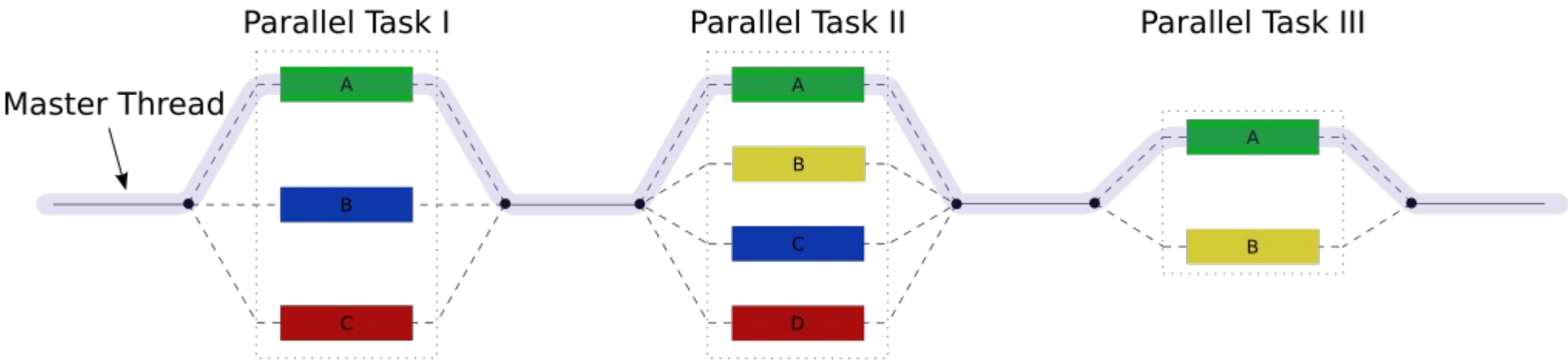
- A first loop over BdimM
- A second loop over BdimN
- A third loop over BdimK
- A double loop over the possible position of the kernel in the selected C subregion

# CPU parallelization paradigms



# OpenMP principle




**OpenMP « threads » are logical entities and does not refer directly to CPU cores or CPU threads (with Hyper Threading)**

**→ There can be less or more OpenMP threads than cores**

# OpenMP syntax

**OpenMP is used in the form of « pre-processing » directives.**

These lines will be replaced by actual code lines at the start of the compiling.

```
#pragma omp parallel shared(…) private(…)
{
    [code inside the parallel region]
    #pragma omp for schedule(TYPE,N)
    for(i=0, i<X; i++){
        [In loop code]
    }
}
```

OpenMP directives are only taken into account when adding a specific compilation flag.

**For gcc it is `-fopenmp`**

The number of threads X in each parallel is defined outside of the code by setting an environment variable of the system, which can be done with:

```
export OMP_NUM_THREADS=X
```

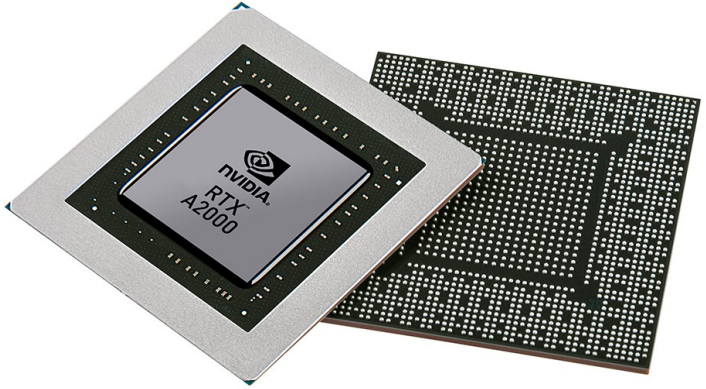
**The logical threads are distributed on the system, occupying physical core and threads.**

# What about computing on GPUs?

**A GPU (Graphical Processing Unit) is a massively parallel computing chip dedicated to SIMD-like operations (thousands of cores).** Most image processing algorithms apply the same transformation to millions of pixels, following a SIMD formalism.

GPU chips have the same form factor as CPUs but they usually come as a dedicated daughter board with their own large cooling system due to a higher power draw than CPUs!

**GPUs are not suited for all tasks, but they pack a huge amount of computing power for tasks they were designed for, which includes matrix multiplication!**




# Nvidia H100 GPU spec-sheet (AI dedicated)

| Graphics Processor | |
|---|---|
| GPU Name: | GH100 |
| Architecture: | Hopper |
| Foundry: | TSMC |
| Process Size: | 4 nm |
| Transistors: | 80,000 million |
| Density: | 98.3M / mm² |
| Die Size: | 814 mm² |

| Graphics Features | |
|---|---|
| DirectX: | N/A |
| OpenGL: | N/A |
| OpenCL: | 3.0 |
| Vulkan: | N/A |
| CUDA: | 9.0 |
| Shader Model: | N/A |

| Graphics Card | |
|---|---|
| Release Date: | Mar 21st, 2023 |
| Generation: | Tesla Hopper (Hxx) |
| Predecessor: | Tesla Ada |
| Production: | Active |
| Bus Interface: | PCIe 5.0 x16 |

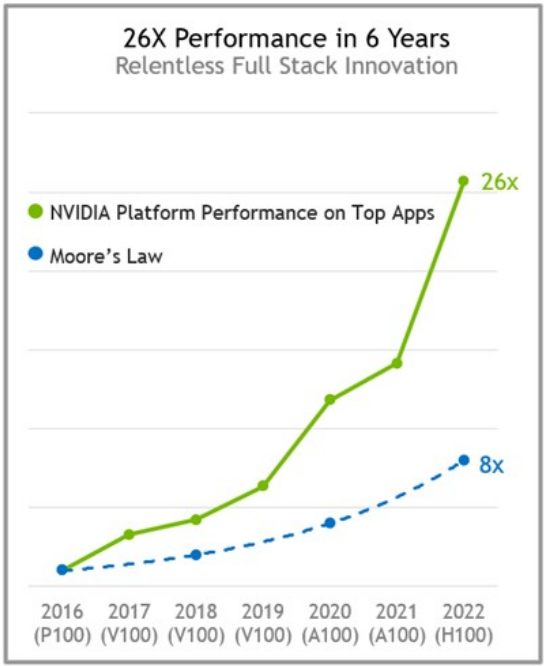
| Board Design | |
|---|---|
| Slot Width: | Dual-slot |
| Length: | 268 mm 10.6 inches |
| Width: | 111 mm 4.4 inches |
| TDP: | 350 W |
| Suggested PSU: | 750 W |
| Outputs: | No outputs |
| Power Connectors: | 1x 16-pin |
| Board Number: | P1010 SKU 200 |

| Clock Speeds | |
|---|---|
| Base Clock: | 1095 MHz |
| Boost Clock: | 1755 MHz |
| Memory Clock: | 1593 MHz 3.2 Gbps effective |

| Render Config | |
|---|---|
| Shading Units: | 14592 |
| TMUs: | 456 |
| ROPs: | 24 |
| SM Count: | 114 |
| Tensor Cores: | 456 |
| L1 Cache: | 256 KB (per SM) |
| L2 Cache: | 50 MB |

| Memory | |
|---|---|
| Memory Size: | 80 GB |
| Memory Type: | HBM2e |
| Memory Bus: | 5120 bit |
| Bandwidth: | 2,039 GB/s |

| Theoretical Performance | |
|---|---|
| Pixel Rate: | 42.12 GPixel/s |
| Texture Rate: | 800.3 GTexel/s |
| FP16 (half): | 204.9 TFLOPS (4:1) |
| FP32 (float): | 51.22 TFLOPS |
| FP64 (double): | 25.61 TFLOPS (1:2) |




*From Nvidia*

*From Techpowerup*

# GPU architecture in theory

The GPU is specialized for highly parallel computations and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control. The schematic Figure 1 shows an example distribution of chip resources for a CPU versus a GPU.

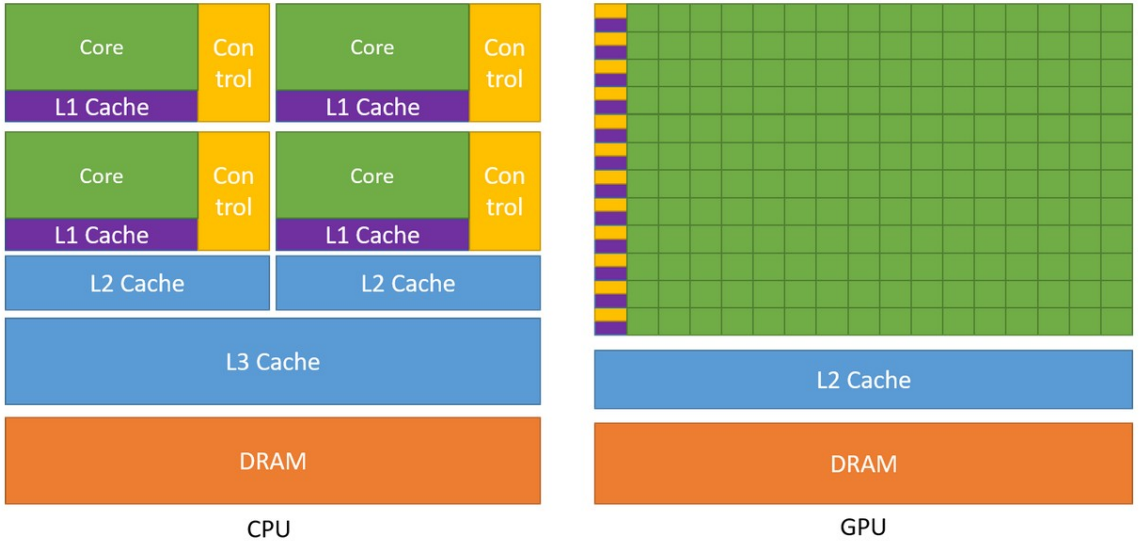


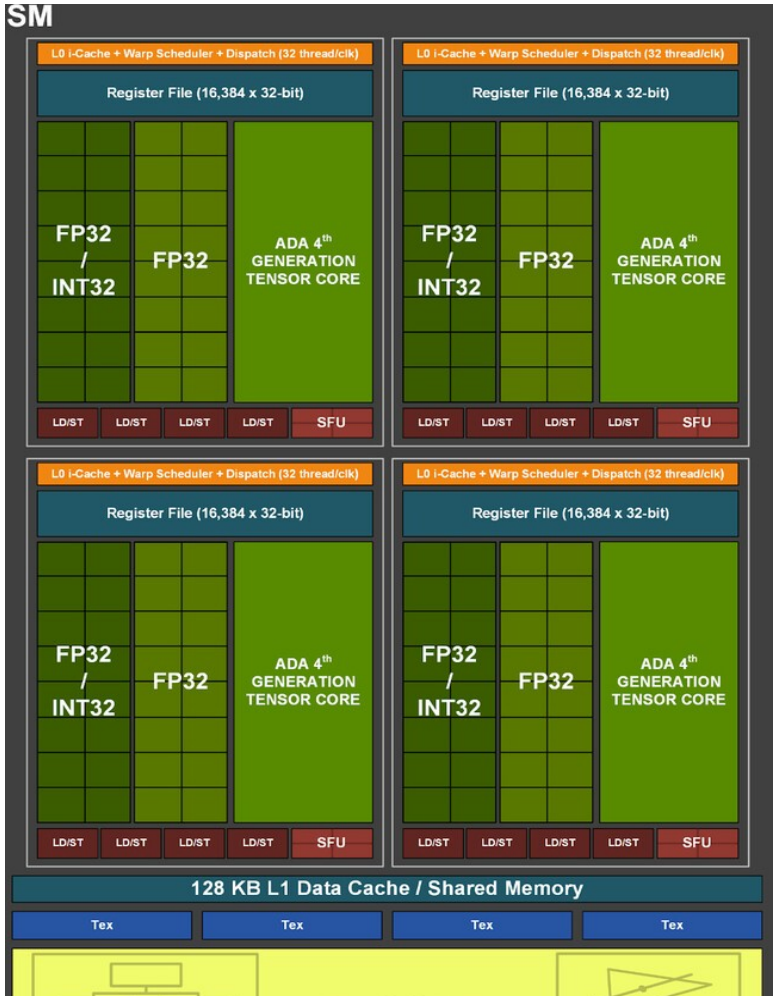

Figure 1: *The GPU Devotes More Transistors to Data Processing*

*From Nvidia CUDA programming guide*

# GPU architecture in the real world


*Ex. of an Nvidia AD102 chip (Ada Lovelace) – used in RTX 4090 and RTX 6000 Ada*

# GPU architecture in details




Nvidia GPUs are decomposed into Streaming Multiprocessors (SMs) each containing an on-chip SRAM block that serves both as L1 data cache and a programmer managed **“shared memory”** space.

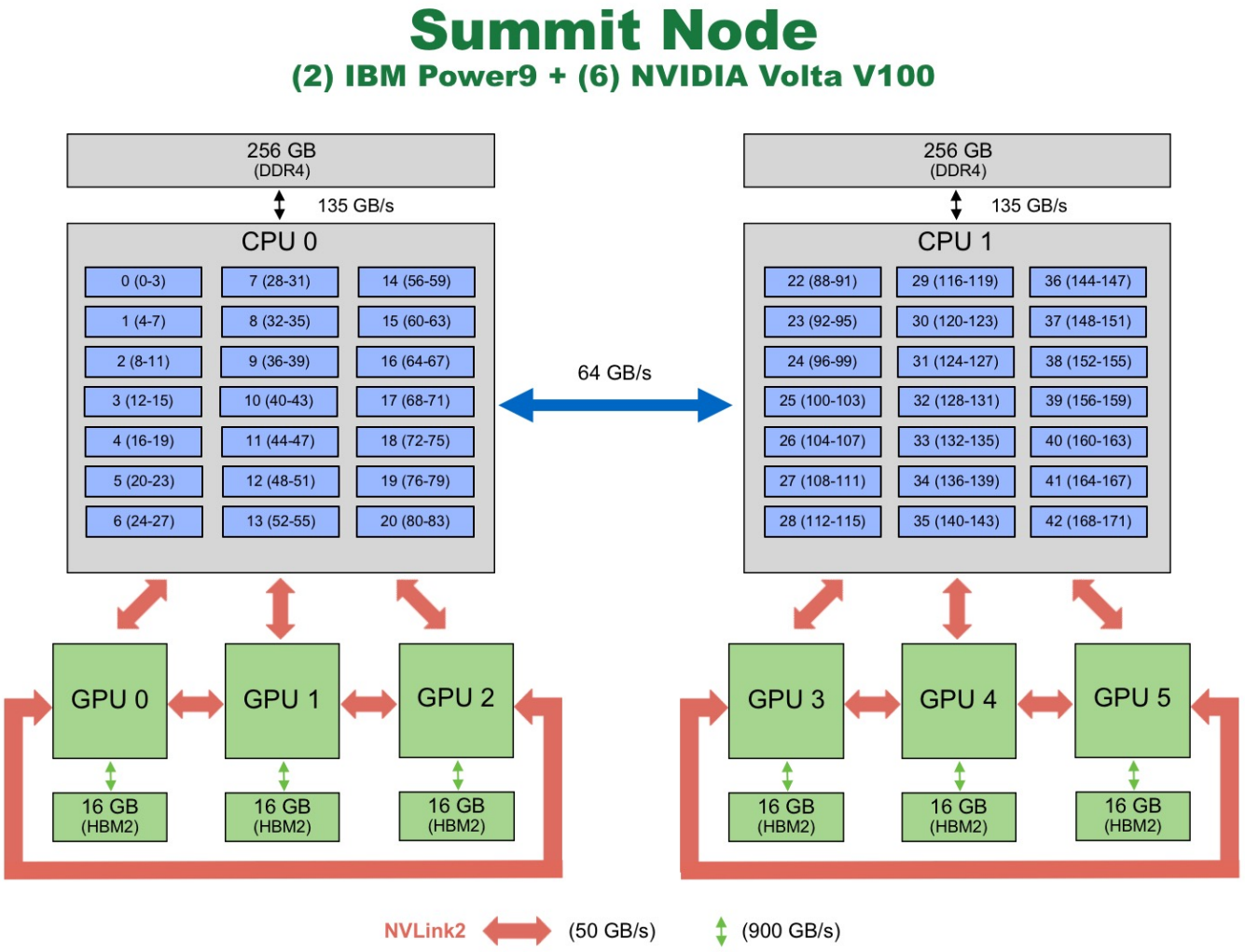
Each SM contains multiple independent warp schedulers, each with its own independent register file and dedicated execution units. Depending on the specific architecture, they can also be accompanied by special compute units dedicated to specific tasks, like a Tensor Core.

**Warps are logical groups of 32 threads.** A warp scheduler is then tasked to issue warp-level instructions to its SM (distributing it to the appropriate execution unit, possibly over multiple GPU clock cycles).

At the scale of the whole GPU, all SMs usually share a single L2 cache, and they all have access to the same dedicated VRAM on the GPU board.

*Ex. of an Nvidia AD102 chip (Ada Lovelace) – used in RTX 4090 and RTX 6000 Ada*

# Distribution in GPU clusters



# General Purpose GPU programming

There are not many GPU designing and manufacturing companies, and even fewer that allow GPGPU programming. **Mainly two brands for this, Nvidia and AMD.**

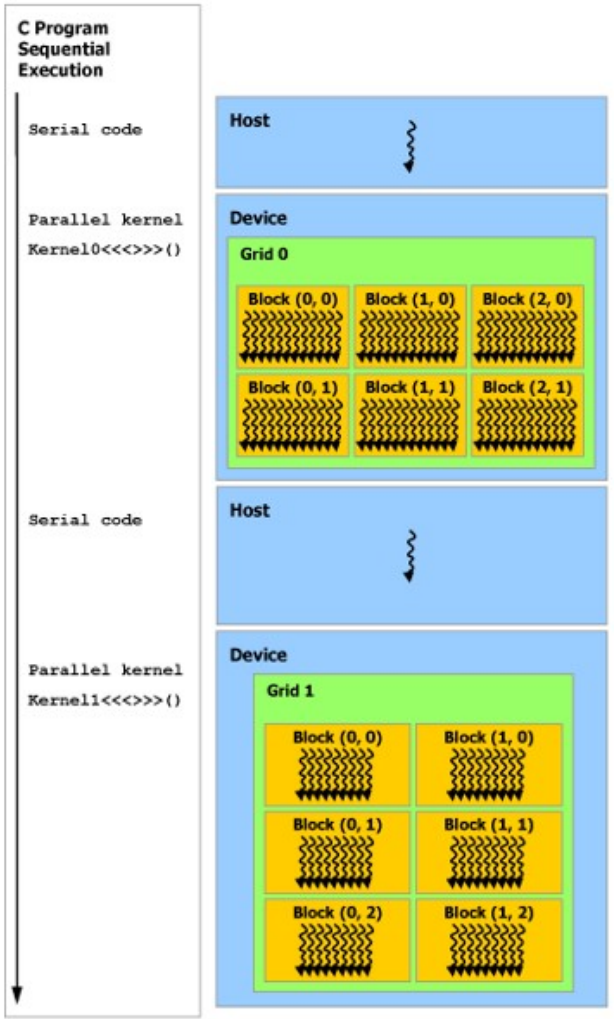
**Programming on GPUs usually requires using specialized programming languages and/or libraries to construct parallel kernels.**

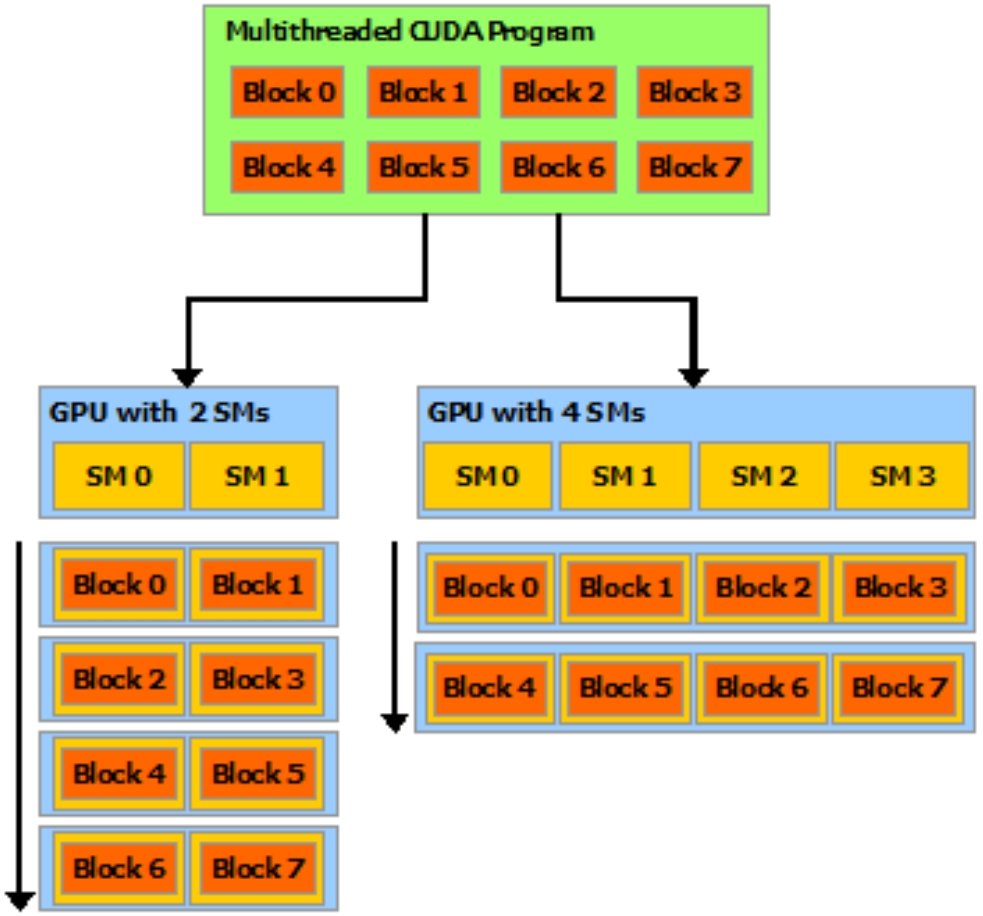


The NVIDIA CUDA solution mostly dominates GPU programming. However, it is **limited to running only on Nvidia GPUs**. Moreover, CUDA is not open source. **Nvidia is dominating the AI/ML domain** with dedicated hardware and technologies.

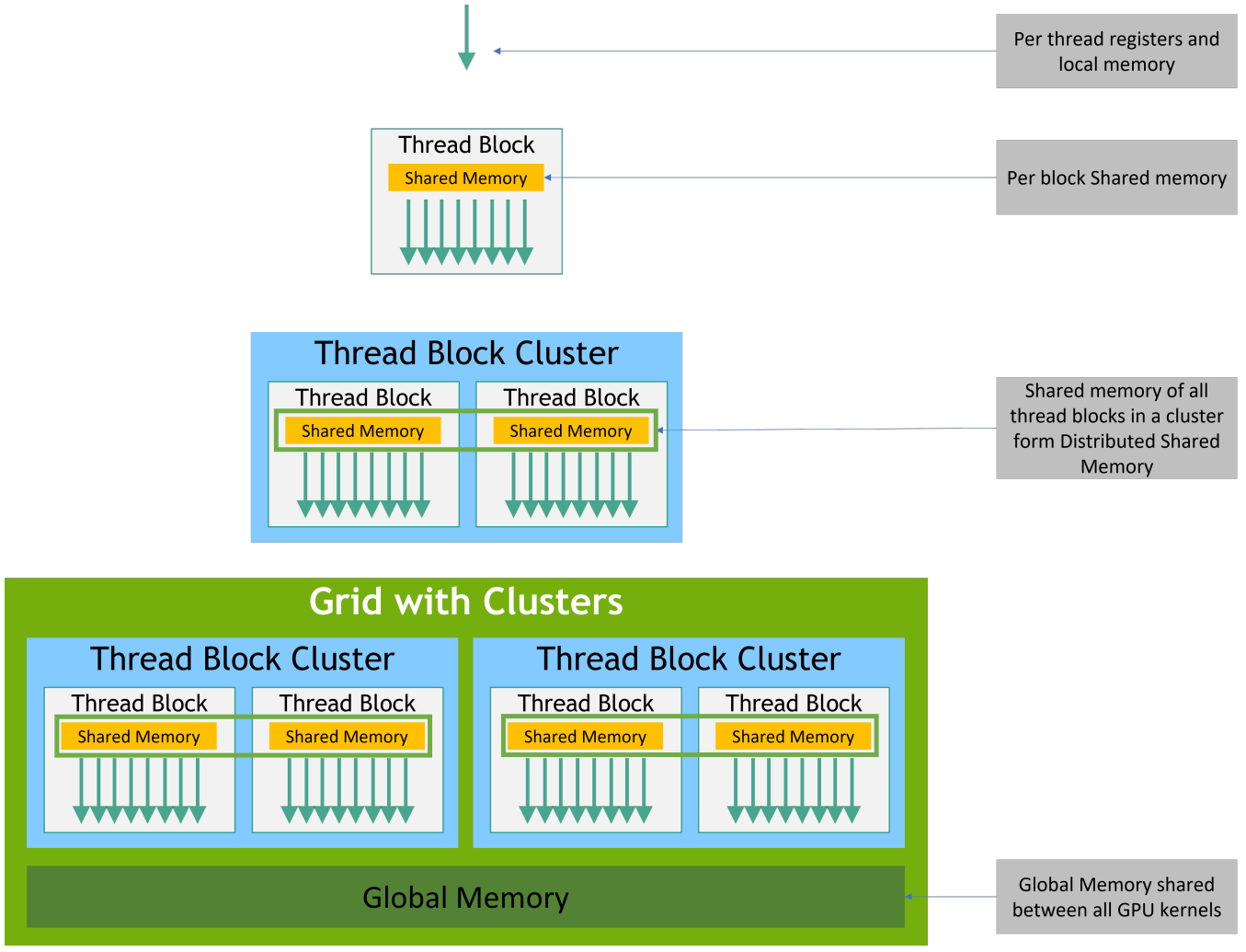
In contrast, AMD has always worked on open-source GPGPU development tools (OpenCL and now ROCm), and ensures that the produced code can run on a **broader variety of hardware** (see HIP).

# CUDA programming model (SIMT)



# CUDA Memory Hierarchy



# Simple CUDA kernel example

```cuda
// Device code
__global__ void VecAdd(float* A, float* B, float* C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}

// Host code
int main()
{
    int N = ...;
    size_t size = N * sizeof(float);

    // Allocate input vectors h_A and h_B in host memory
    float* h_A = (float*)malloc(size);
    float* h_B = (float*)malloc(size);
    float* h_C = (float*)malloc(size);

    // Initialize input vectors
    ...

    // Allocate vectors in device memory
    float* d_A;
    cudaMalloc(&d_A, size);
    float* d_B;
    cudaMalloc(&d_B, size);
    float* d_C;
    cudaMalloc(&d_C, size);

    // Copy vectors from host memory to device memory
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // Invoke kernel
    int threadsPerBlock = 256;
    int blocksPerGrid =
            (N + threadsPerBlock - 1) / threadsPerBlock;
    VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);

    // Copy result from device memory to host memory
    // h_C contains the result in host memory
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    // Free device memory
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    // Free host memory
    ...
}
```

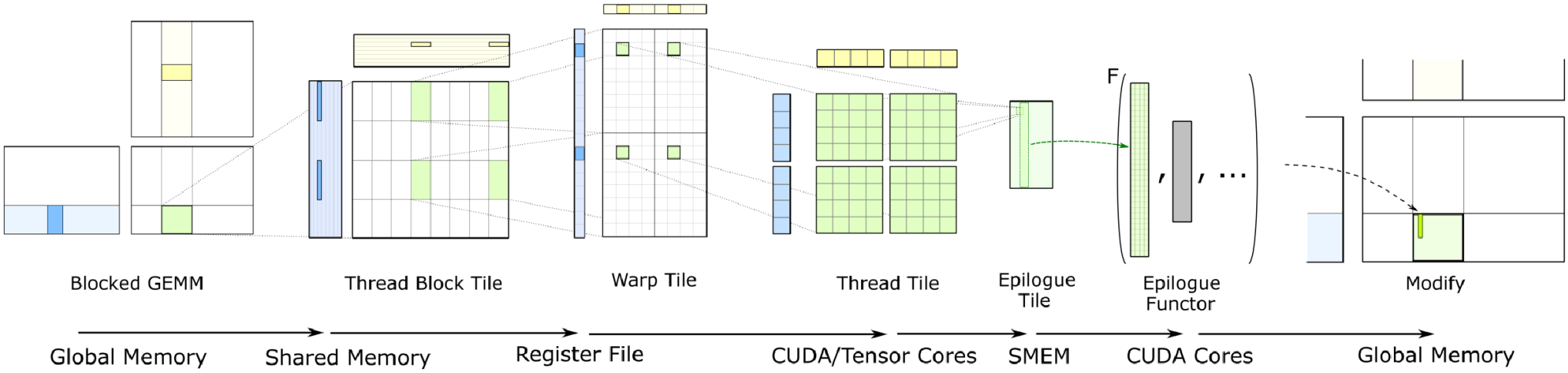
The GPU is controlled by the main program, which allocates GPU memory, moves data to the GPU, and launches kernels.

**Inside a kernel, all the code is executed by each thread (SIMT).** Specific variables are used to recover the IDs corresponding to the thread's position within the block and the block grid.

With these basic elements, it would be possible to design a naive matrix multiplication kernel. However, optimizing it to achieve good performance is much more tedious and requires more effort for handling the memory hierarchy.

Like OpenBLAS, CUDA provides the **high-level cuBLAS library**, which allows calls to optimized matrix multiplication operations that take full advantage of Nvidia's GPU capabilities.

# Optimized GEMM construction for GPU




GemmGlobalIteratorAb
Transformer
GemmSharedStoreTileAb

GemmSharedLoadTile{A,B}

fma, dp4a, hfma2
WMMA

Transformer
GemmSharedStoreTileD
GemmSharedLoadTileD

GemmGlobalIteratorC
Functor
GemmGlobalIteratorD

Global Load Stream

Shared Load Stream

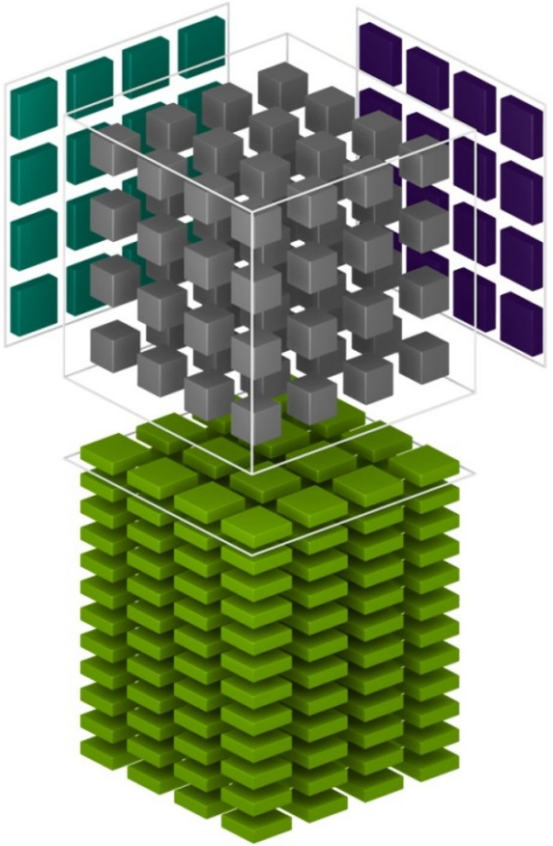
Matrix Multiply

Epilogue

*From CuTLASS documentation*

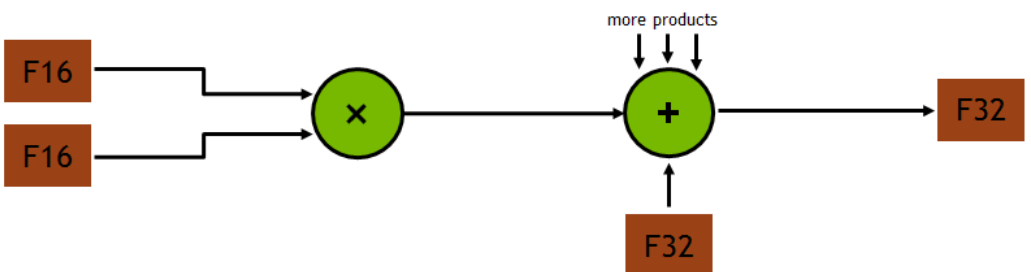
# Nvidia Tensor Cores

**Optimized Warp Matrix Multiply Add (WMMA) instructions !**
**CuBLAS can be set to use tensor core through the gemmEX function.**



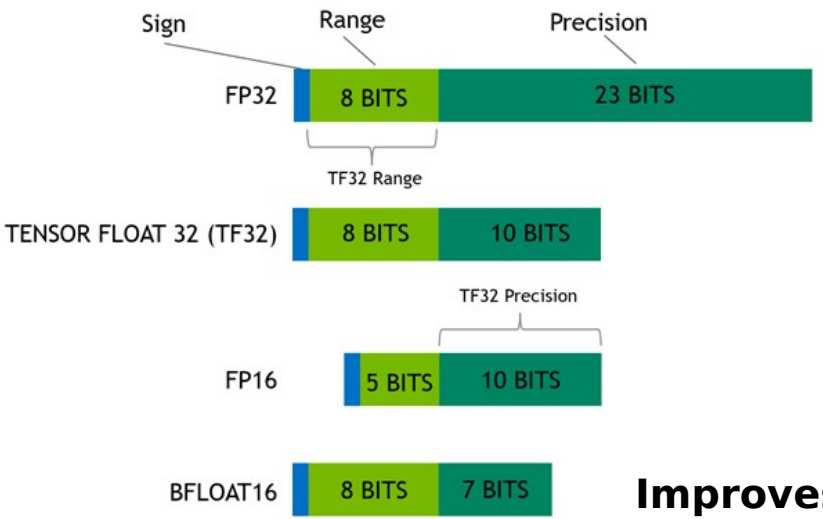
$$D = \underbrace{\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}}_{\text{FP16 or FP32}} \underbrace{\begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}}_{\text{FP16}} + \underbrace{\begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}}_{\text{FP16}}$$

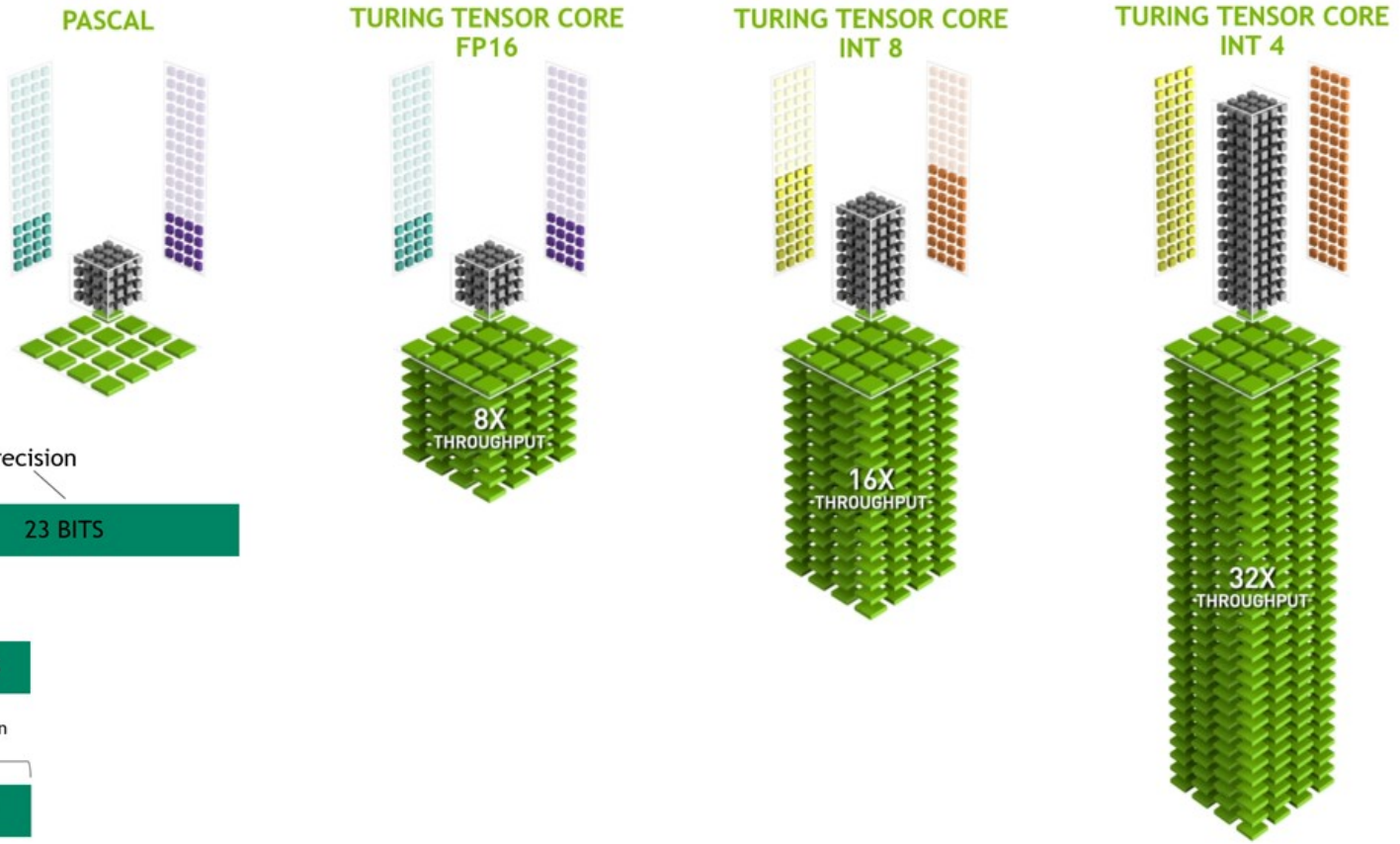
**FP16 storage/input** | **Full precision product** | **Sum with FP32 accumulator** | **Convert to FP32 result**



# Nvidia Tensor Cores reduced quantization

$$1.2345 = \underbrace{12345}_{\text{mantissa}} \times 10^{\overbrace{-4}^{\text{exponent}}}$$

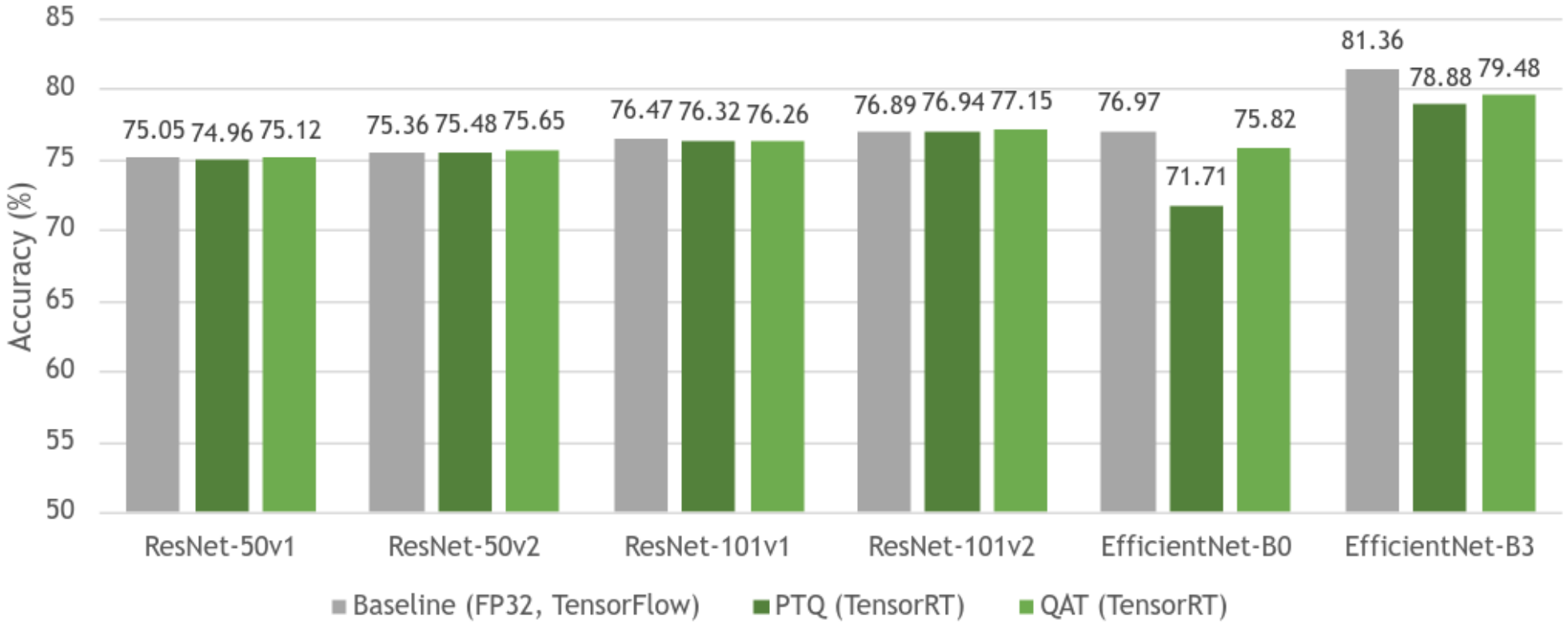



**Improves compute throughput AND reduces data size,**
**→ reduce memory footprint and required bandwidth**

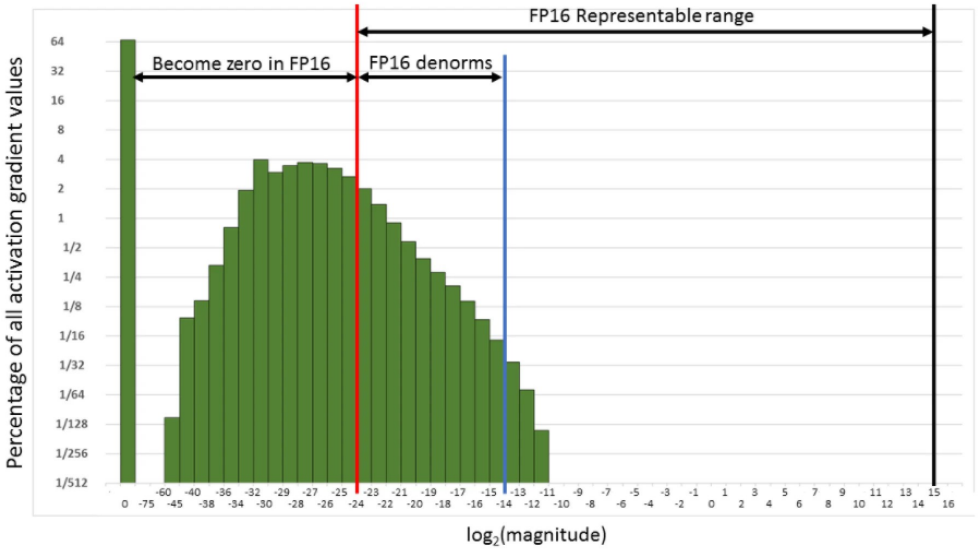
# Quantization effect on AI models accuracy

**All models quantized to INT8, accuracy for ImageNET-2012**




*PTQ = Post training quantization*
*QAT = Quantization aware training*

# Mixed precision training

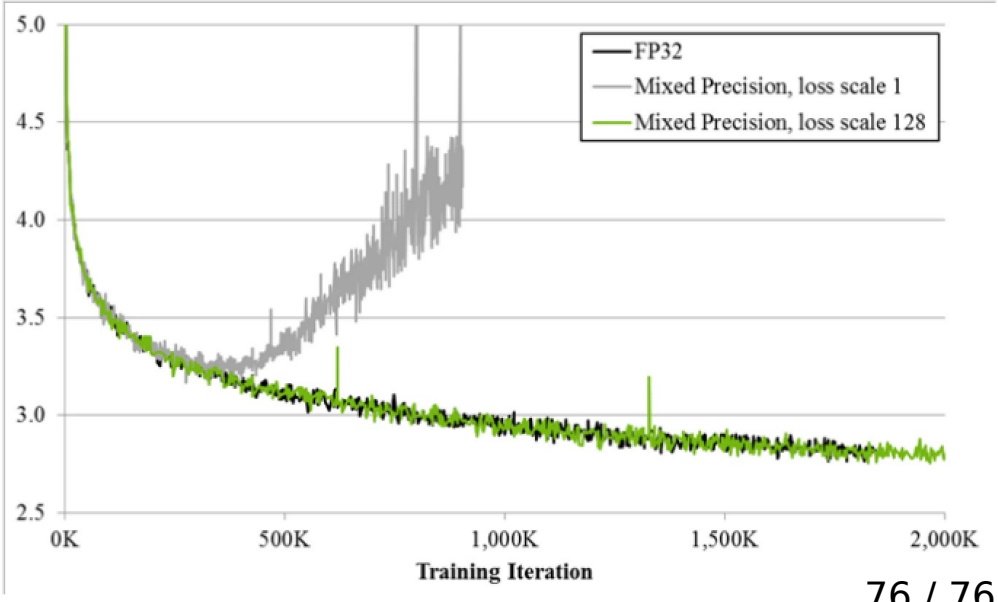



Reduced-bit-count variables have smaller representable ranges. This can lead to severe gradient vanishing.

This problem can first be mitigated by preserving an FP32 copy of the weights to accumulate the low-resolution updates.

To further reduce this issue, a scaling is applied to the output loss. All propagated values are naturally scaled, so they are more likely to be in the proper range.

At weight update time, the correction is scaled down by the same factor.




*From Nvidia « Mixed precision training »*