Averting a System and Software Induced Energy Crisis

Bill Gervasi

July 2024

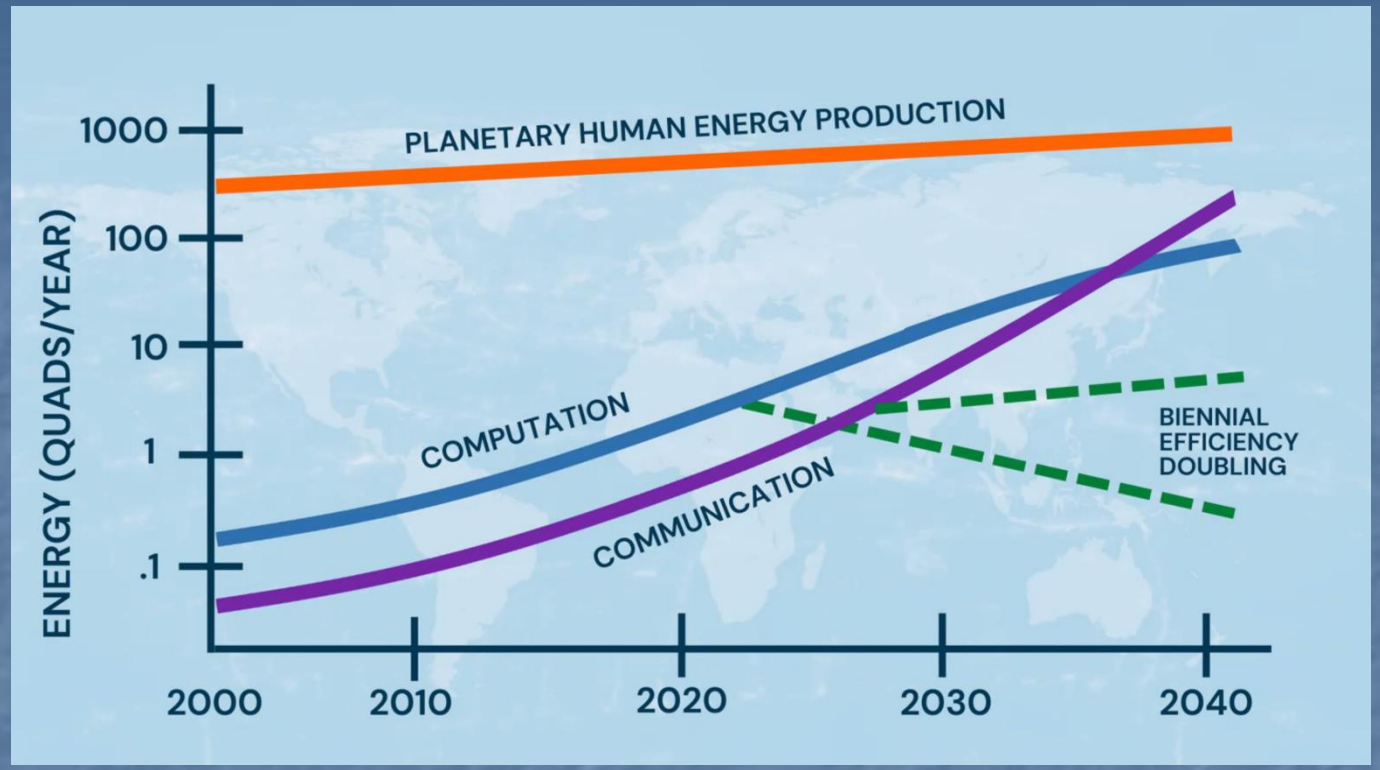
Bill Gervasi

bilge@discobolusdesigns.com

- Computer Science Nerd
- Chairman of JEDEC Committees since 1996
- Principal Systems Architect for Wolley Inc.
- Memory Industry Consultant
- National Engagements
  - US Department of Energy
  - National Science Foundation
- Trustee, BRDG Bridge to Connect

EES2 ENERGY EFFICIENCY SCALING

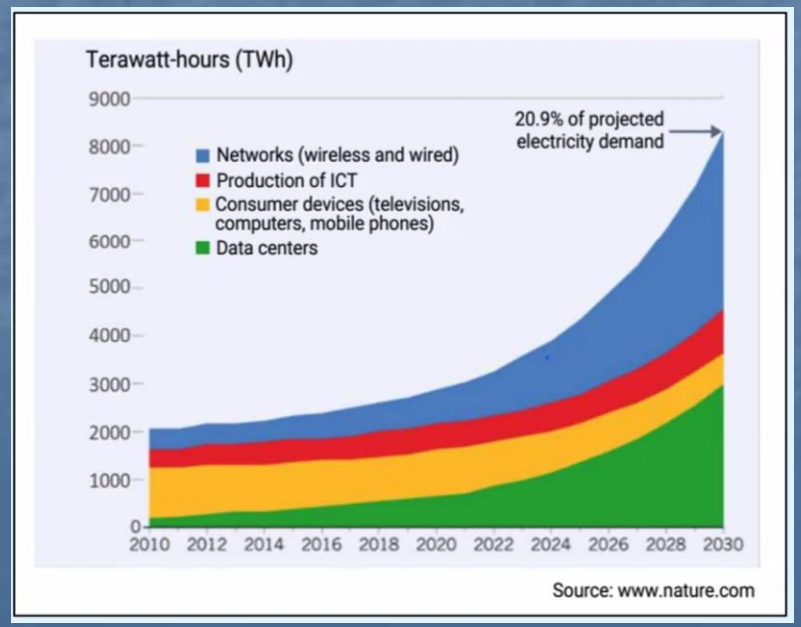
Designing for energy efficiency is a growing concern



On the current trajectory of energy use versus energy production,

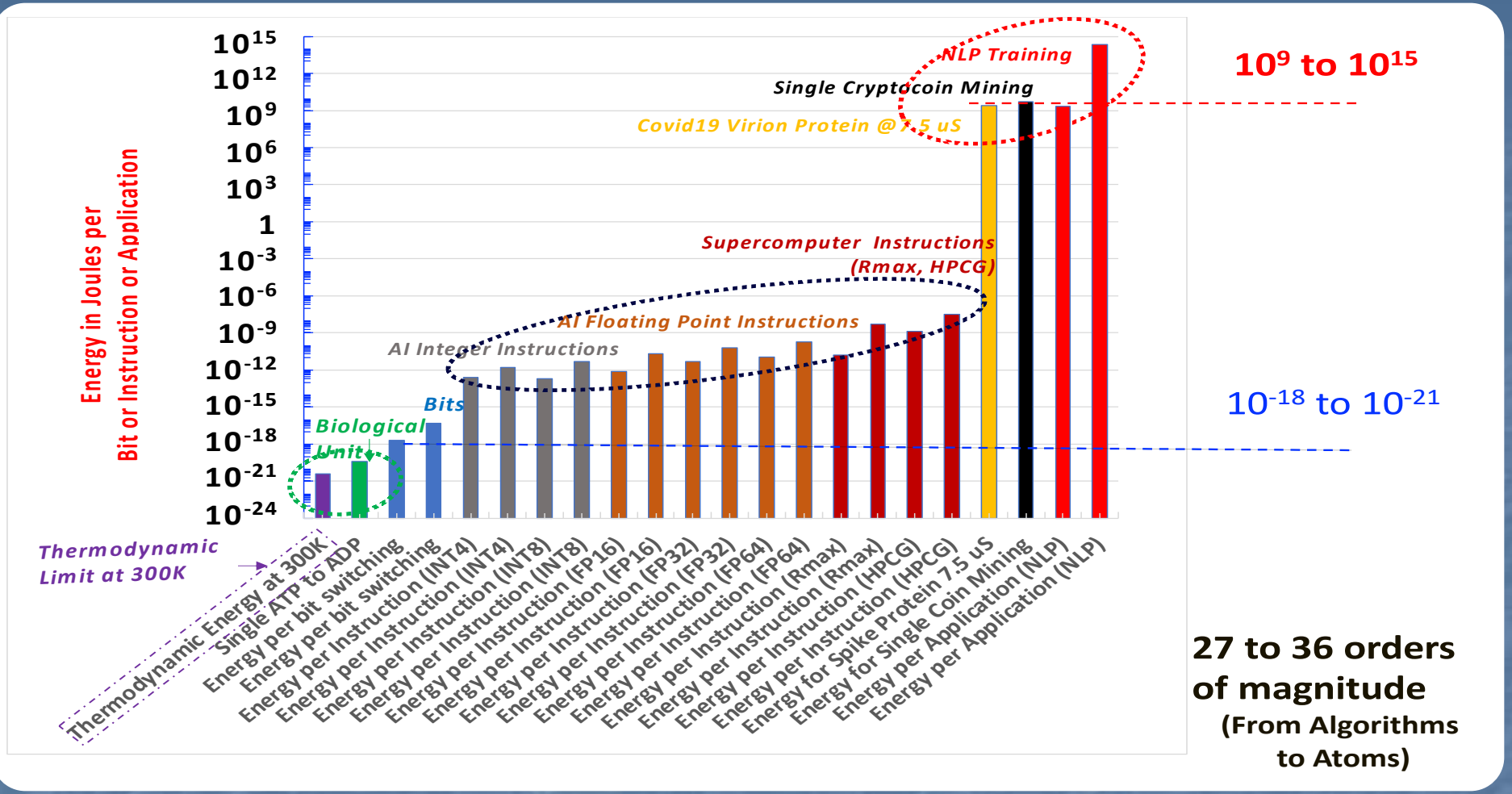
THESE CROSS OVER ~ 2055

EES2 program goal is **1000X** improvement in energy efficiency over the next **20 years**



Where is all that power going?

Operation	Energy per bit
Wireless data	10 – 30μJ
Internet: access	40 – 80nJ
Internet: routing	20nJ
Internet: optical WDM links	3nJ
Reading DRAM	5pJ
Communicating off chip	1 – 20 pJ
Data link multiplexing and timing circuits	~ 2 pJ
Communicating across chip	600 fJ
Floating point operation	100fJ
Energy in DRAM cell	10fJ
Switching CMOS gate	~50aJ – 3fJ
1 electron at 1V, or 1 photon @1eV	0.16aJ (160zJ)



10^{9} to 10^{15}

10^{-18} to 10^{-21}

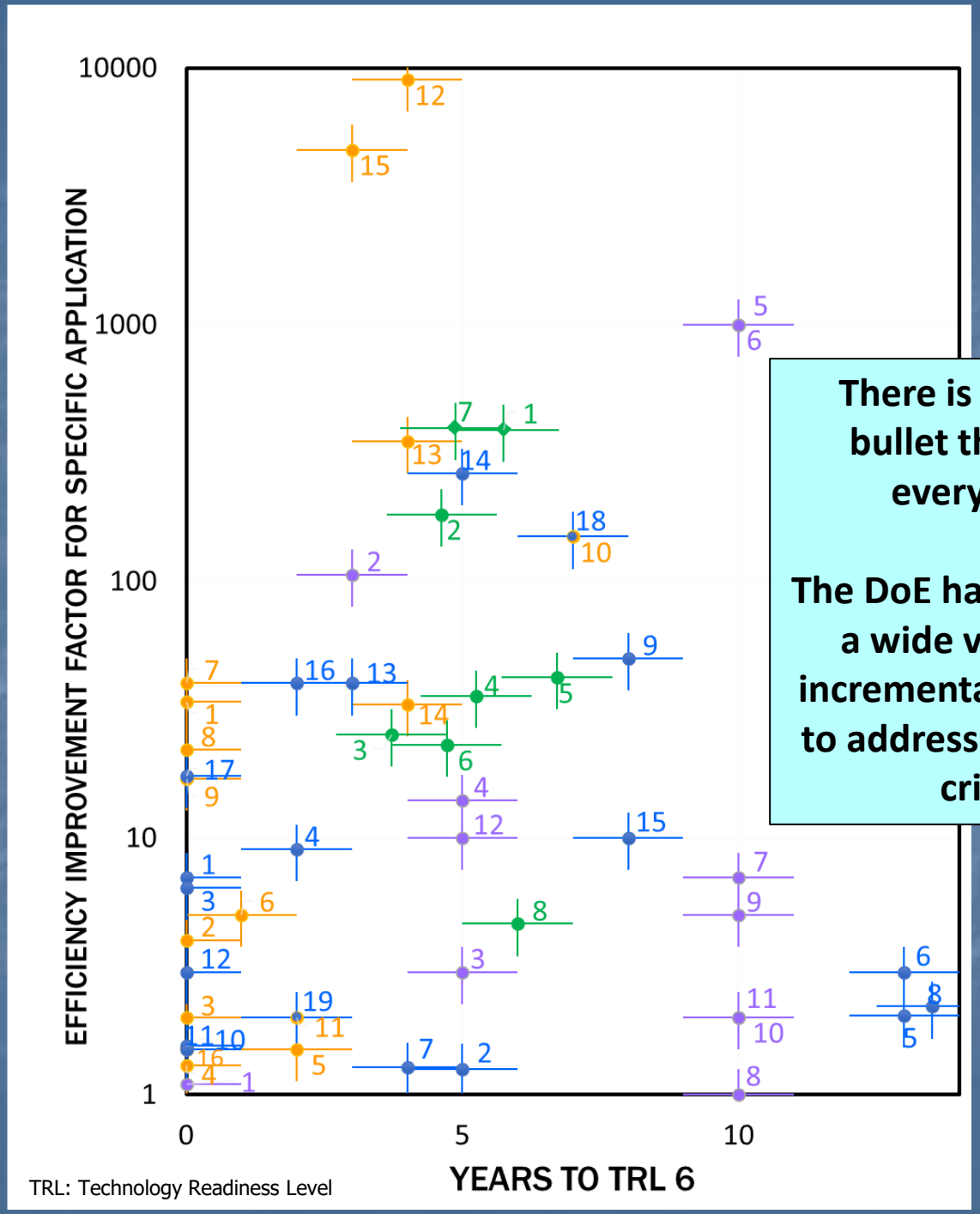
27 to 36 orders of magnitude
(From Algorithms to Atoms)

Part of the looming energy crisis are **fundamental inefficiencies** of applications and programming languages

Python programming is orders of magnitude less energy efficient than C programming (ChatGPT is Python-based)

Cryptocurrency in particular consumes ≥0.5% of **world** energy resources already

P.S. This is not good...



TRL: Technology Readiness Level

There is no silver bullet that fixes everything

The DoE has identified a wide variety of incremental solutions to address the power crisis

Circuits and Architectures

1 ReRAM vs NAND
2 STTRAM vs NAND
3 NRAM vs DRAM
4 ReRAM vs DRAM
5 CNT NVM
6 Metis SRAM
7 Molecular dynamics ASIC
8 FPGAs for machine vision
9 SRAM stacked 3D DNN accelerator
10 MIV stacked ReRAM
11 HBM Cache
12 Neuromorphic memcapacitive devices
13 Neuromorphic memristor matrix multiplier
14 Neuromorphic asynchronous computing
15 CMOS SRAM CIM
16 CXL optimized DDR5

Algorithms and Software

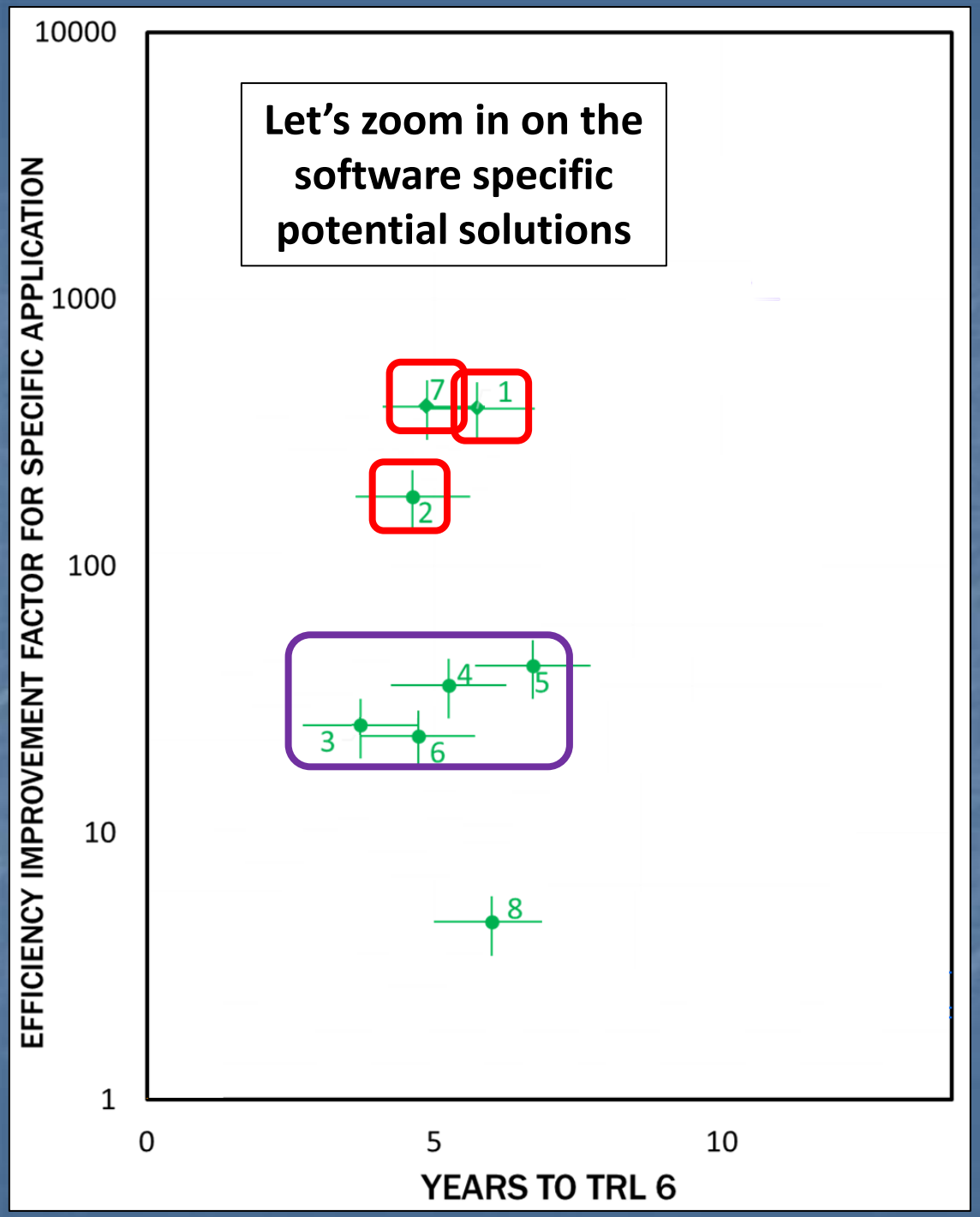
1 Reduced energy for ML algorithms
2 Algorithm-specific energy (tooling)
3 Algorithm-specific energy (benchmarking)
4 Languages, compilers, and runtime systems
5 Communication protocols
6 Homomorphic encryption
7 Software for emerging architectures
8 Computational reliability

Advanced Packaging & Heterogeneous Integration

1 LMP solder with polymer
2 Nanostructured thermal interface surface
3 CNT TIM
4 Graphene TIM
5 Graphene interconnects
6 CNT interconnects
7 Rh/Ir interconnect
8 CNT for 3D ICs
9 3D IC MIVs
10 Feveros
11 TSV for 3D IC
12 Hybrid bonding (Cu-Cu)
13 Optical off-chip interconnect
14 Optical on-chip interconnect
15 Optical bus
16 UCIe chiplet standard
17 3D stacked SRAM
18 MIV stacked ReRAM
19 HBM on logic

Materials and Devices

1 Si-GAA
2 CNT Memory
3 CNTFET (Logic)
4 TFET
5 Spintronic memory
6 FeFET (Flash)
7 Analog devices for neuromorphic computing
8 FeFET (SRAM)
9 Contact & interconnect
10 Novel ILD
11 Spintronic logic
12 2D materials



Algorithms and Software

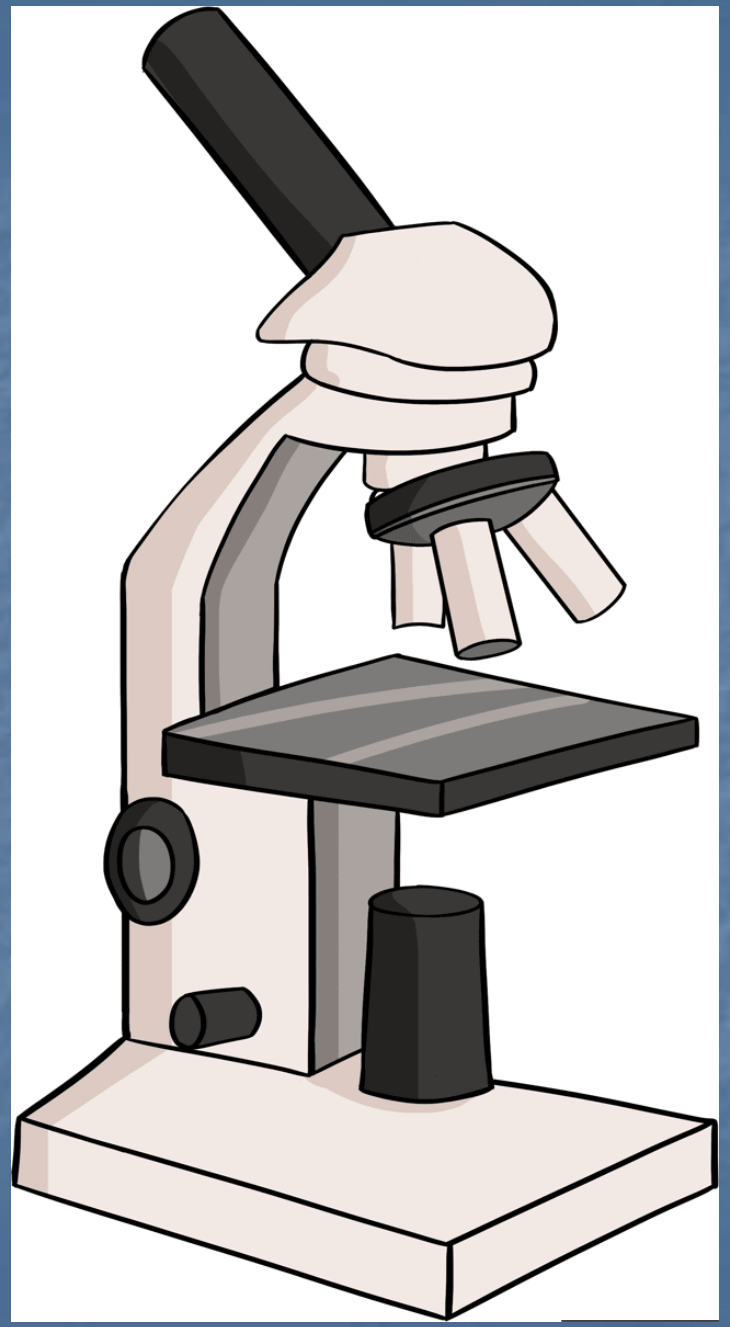
1. Reduced energy for ML algorithms
2. Algorithm-specific energy (tooling)
3. Algorithm-specific energy (benchmarking)
4. Languages, compilers, and runtime systems
5. Communication protocols
6. Homomorphic encryption
7. Software for emerging architectures
8. Computational reliability

As a software nerd
turned hardware geek

let’s take a dive into computer architecture

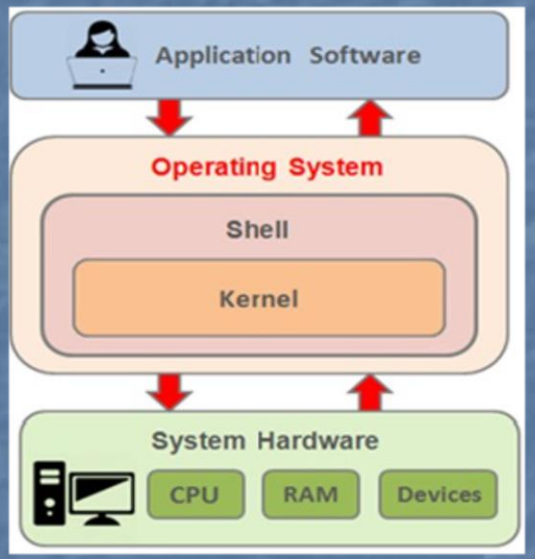
examine how the software and hardware work together

to see what we can do for our planet



Memory

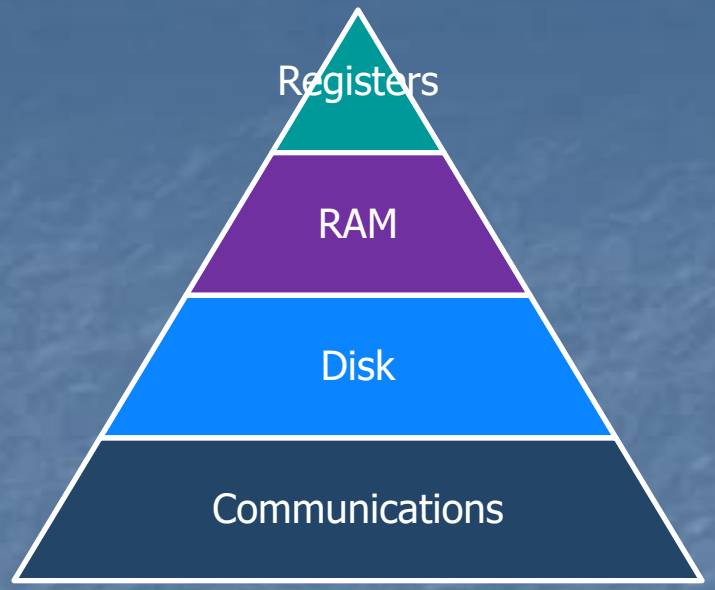
Storage



Fundamentals of Computing

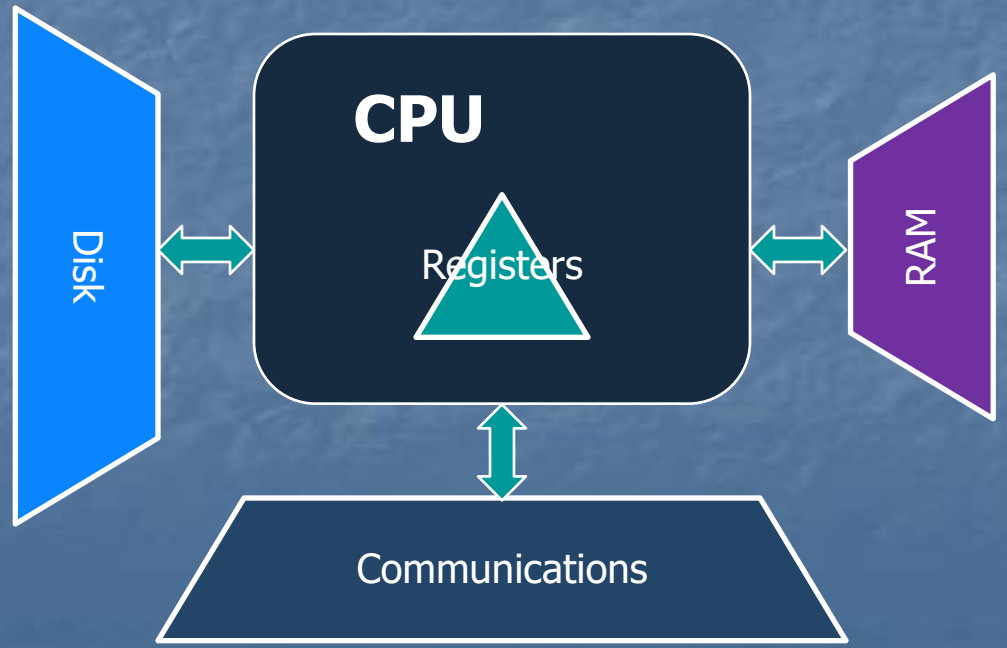
Processing

Communications

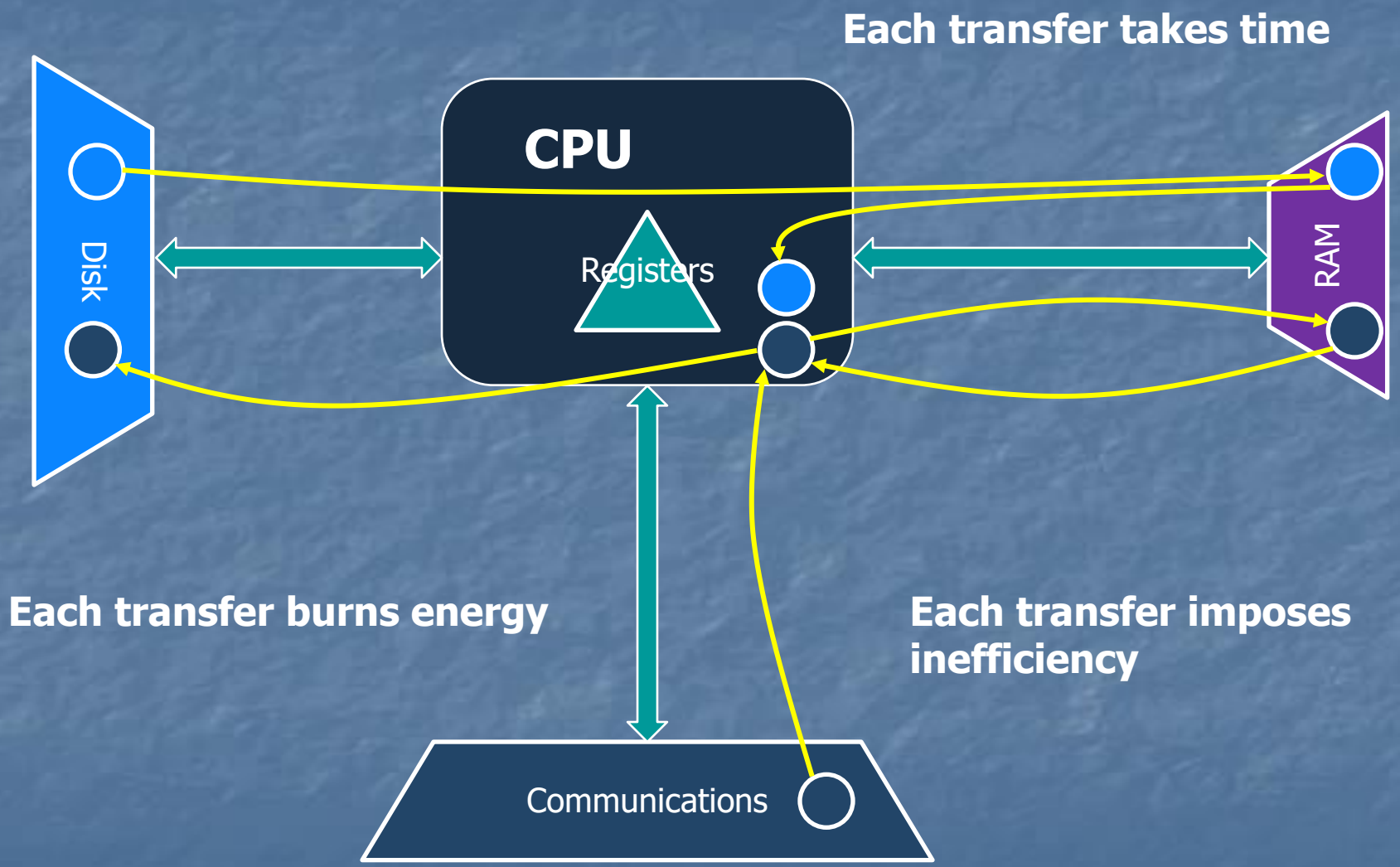


Traditional picture of computing resources helps visualize relative speed, latency, capacity, and cost

However, it does not adequately describe the requisite electrical interfaces, each of which is a distinct source of issues

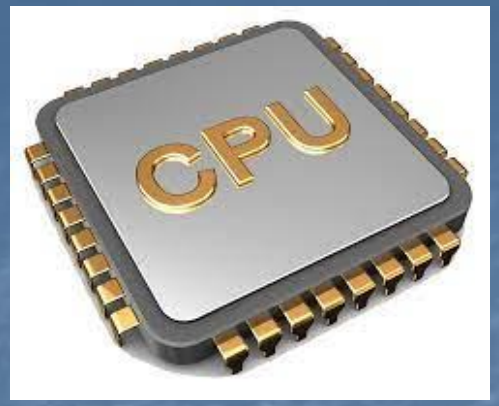


Simplified but realistic case of program execution and data movement



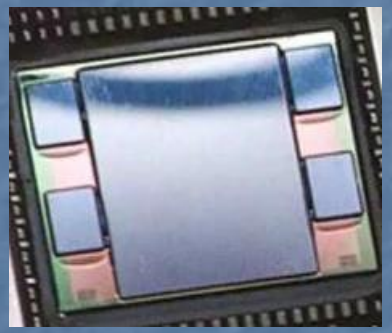
Typical application flow

1. App read from disk through CPU to RAM
2. App read from RAM to CPU for execution
3. Info read from I/O through CPU and written to RAM
4. App reads RAM to process
5. App writes results to disk



INT8
INT16
FP16
FP32
FP64

CPU registers have an intrinsic waste with various size data types



CPUs have all added caches for recently accessed data

Industry standard is 64 bytes per cache line

64 Byte Cache Line

If an application needs a yes or no answer (1 bit)

But accesses a cache line (64 bytes)

Waste = 99.8%

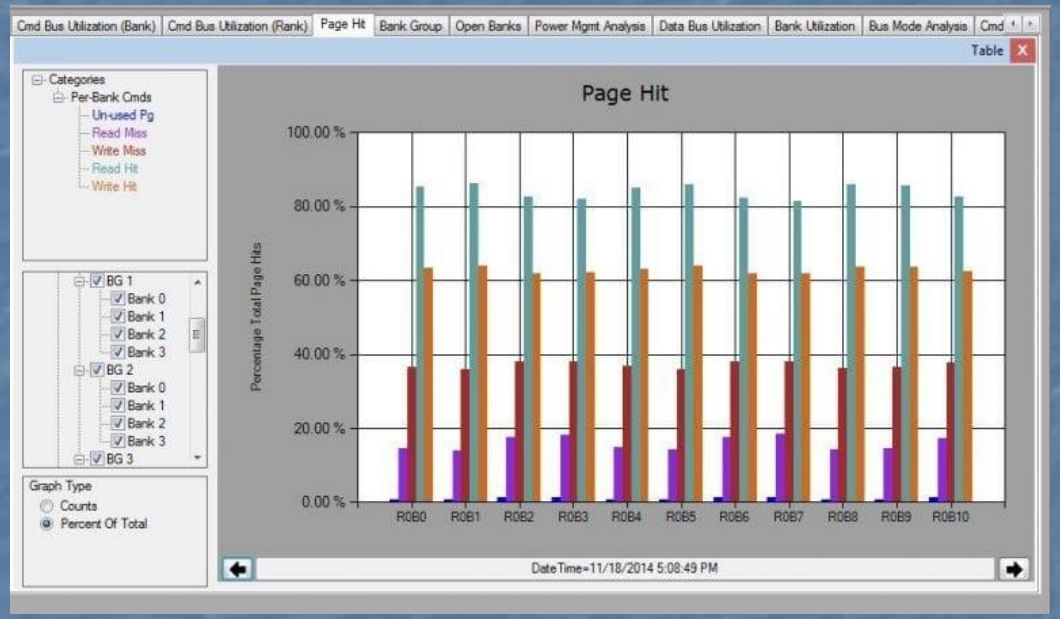
Discrepancy between cache line size and data item size creates significant wasted data access

L1: 96% hit rate, 1 cycle access
L2: 95% hit rate, 25 cycles access
L3: 98% hit rate, 80 cycles access

The good news: near-CPU caches do have high hit rates
(reduces waste from unnecessary accesses)

By the time an access gets to the local DRAM, though, hit rates start to drop dramatically

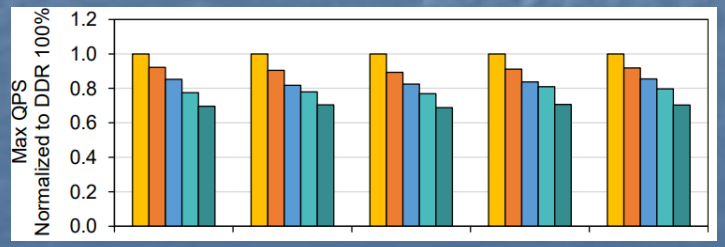
Read hit ~82%
Write hit ~62%



A question I have posed that CPU guys refuse to answer:

How much performance gain are we getting for each watt expended?

ESPECIALLY when it comes to speculative operations



Access to remote memory drops even further, especially with increased thread count

Hit rate ~65%

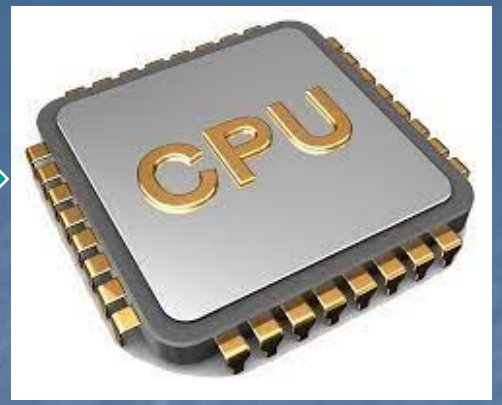
…and this is before memory pooling…

https://www.futureplus.com/blog/critical-memory-performance-metrics-for-ddr4-systems-page-hit-analysis

https://arxiv.org/pdf/2303.15375#:~:text=Meanwhile%2C%20as%20the%20block%20size%20increases%20beyond,latency%20begins%20to%20dominate%20the%20p99%20latency.

HDD/SSD

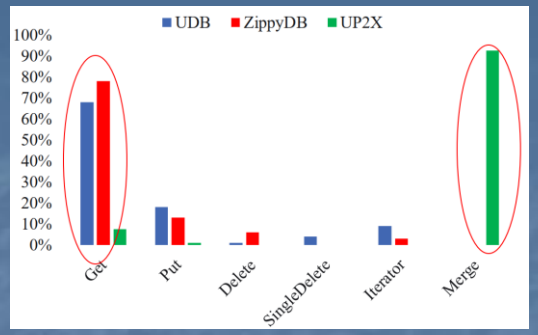
4KB block



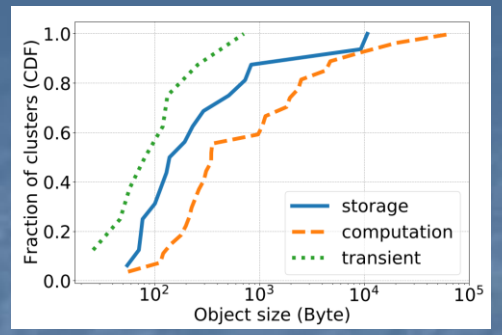
Disk/SSD access

64B cache line

Facebook RocksDB



X (Twitter) Twemcache



The average key size (AVG-K), the standard deviation of key size (SD-K), the average value size (AVG-V), and the standard deviation of value size (SD-V) of UDB, ZippyDB, and UP2X (in bytes)

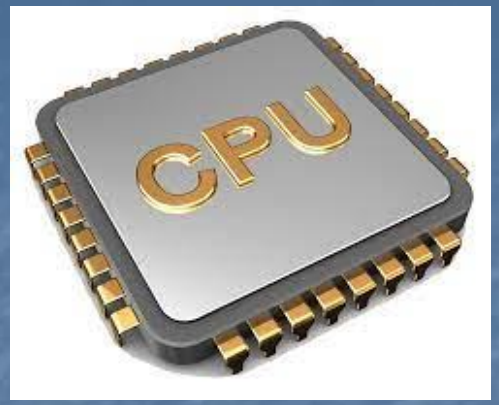
	AVG-K	SD-K	AVG-V	SD-V
UDB	27.1	2.6	126.7	22.1
ZippyDB	47.9	3.7	42.9	26.1
UP2X	10.45	1.4	46.8	11.6

Typical disk block transfer size is 4KB

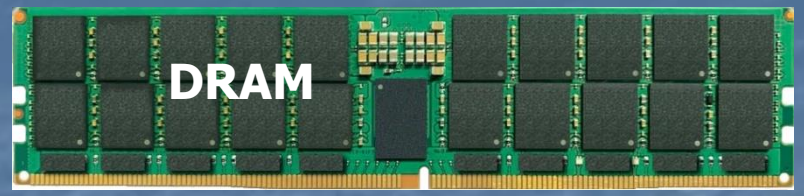
Average number of bytes actually used is 100

This is Best Case… even worse if the block is cached

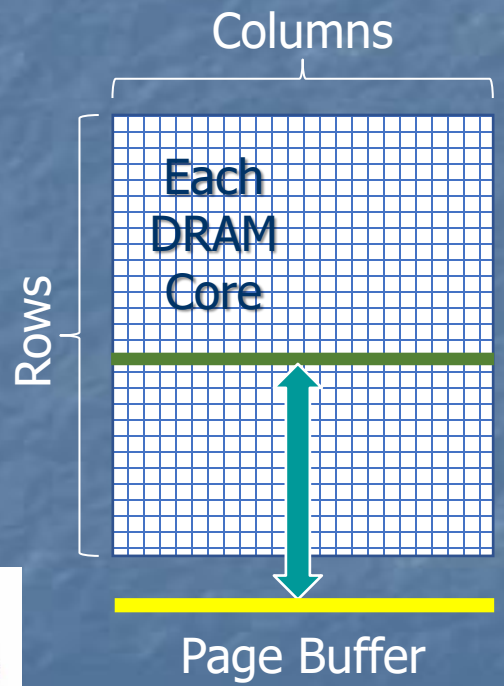
Waste = 97.5%



64 byte
cache line



10KB block
X 2



RAMs are grouped in 10s to form a "rank"

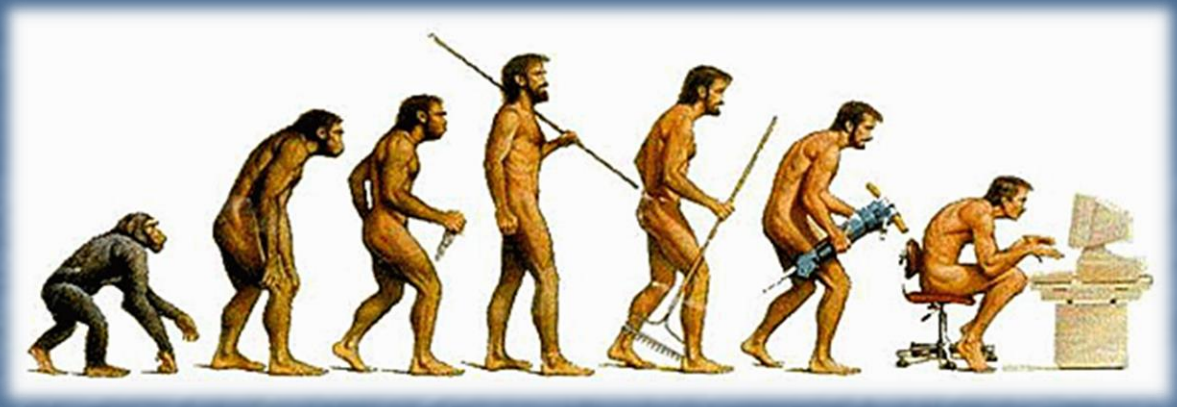
Each RAM has a 1KB page buffer size (access granularity)

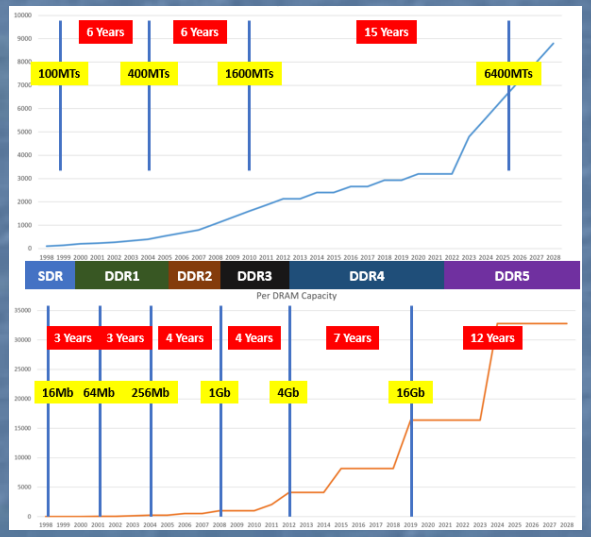
Activations are destructive and data rewrite is needed

Therefore, every data access requires 20KB of data movement

Waste = 99.7%

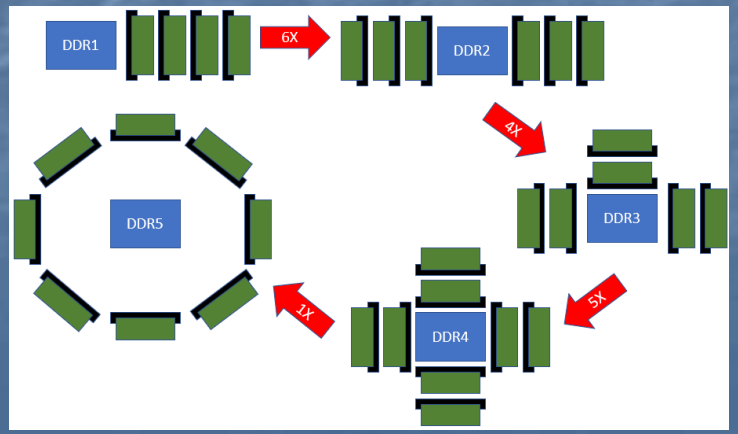
Computer systems continue to evolve



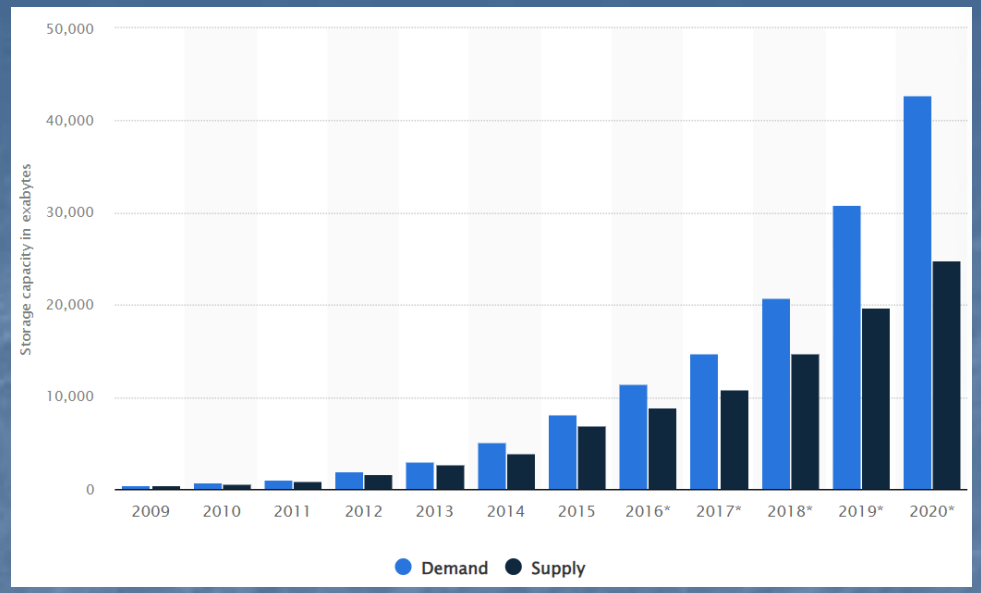


Development of faster, higher capacity memories slowed

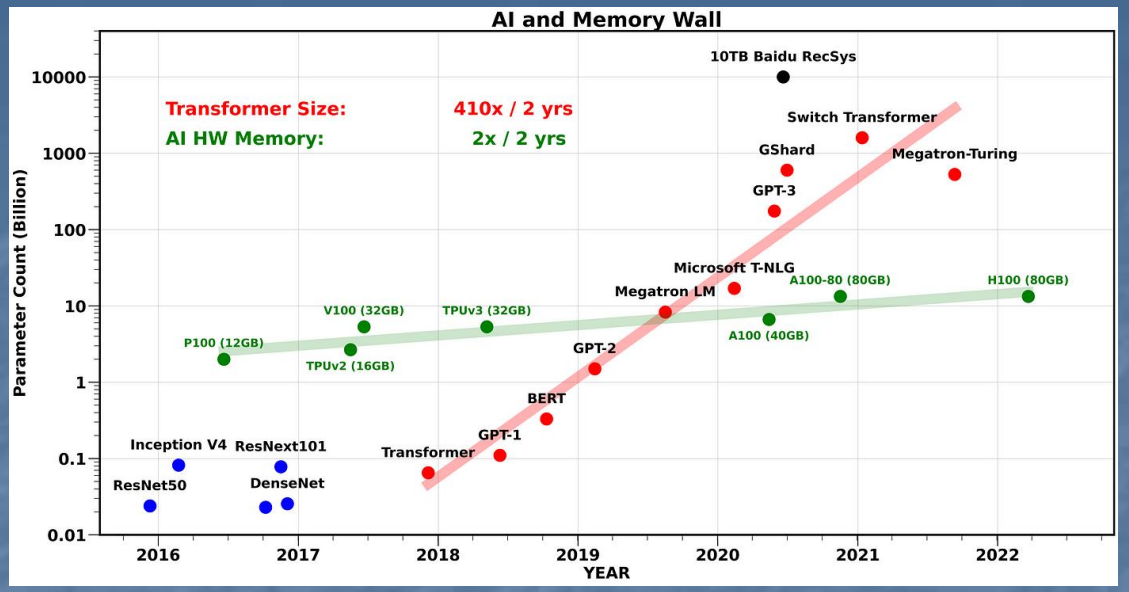
Electrical challenges with supporting more memory got worse resulting in fewer memory slots



IT managers got very sad

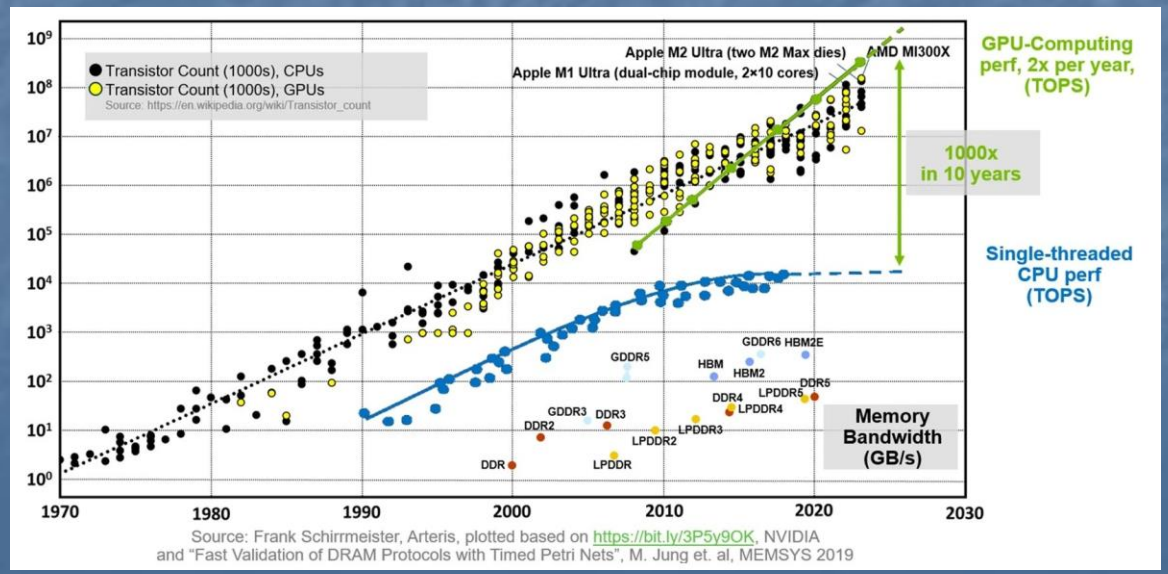


Demand for media outstrips supply



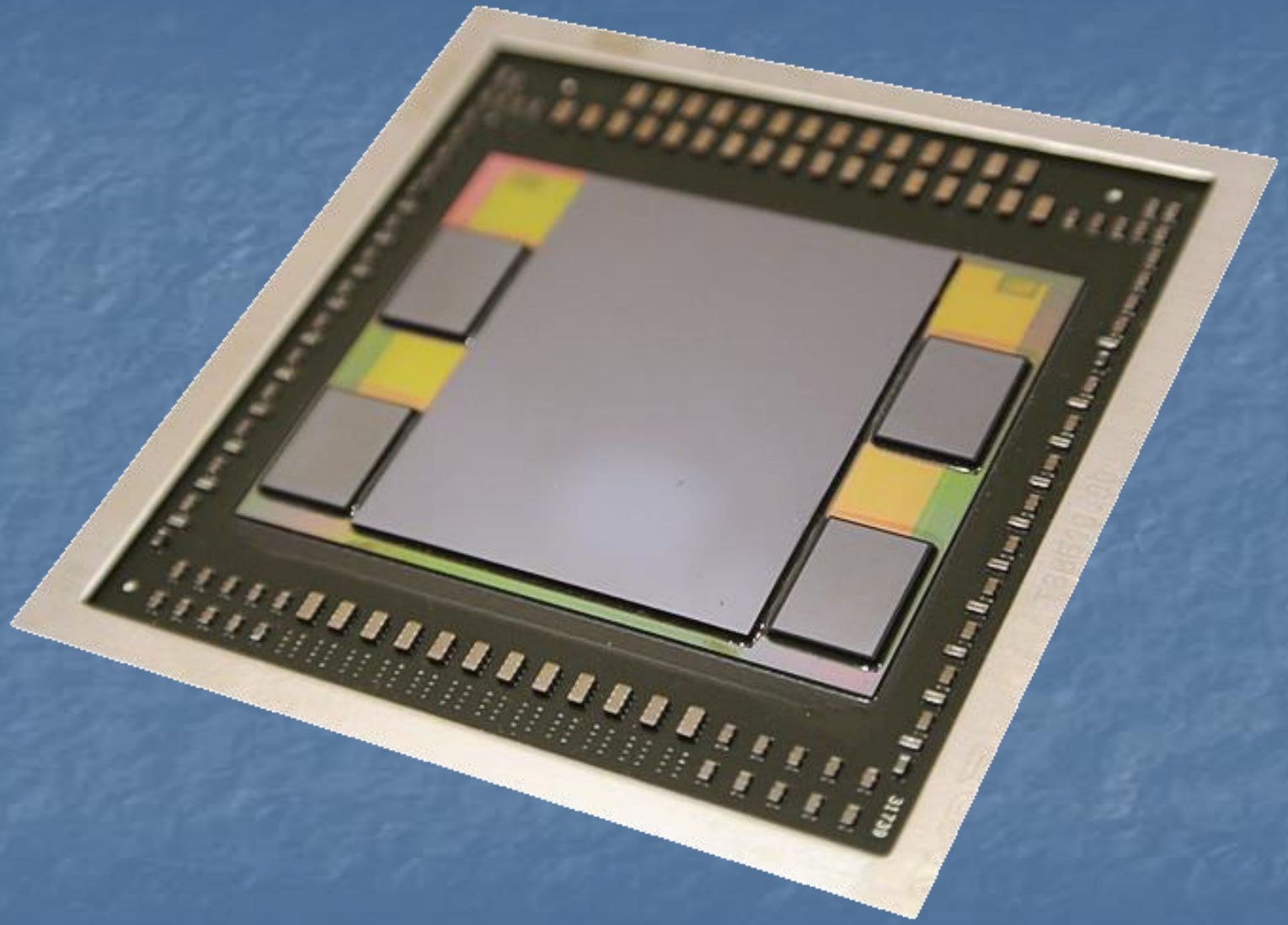
AI algorithms largely drive new demand

Media speeds can't keep up with demand



How is the industry addressing the need for more capacity and higher bandwidth?

One way is to add memory to the processor



High Bandwidth Memory (HBM)

Added to most new processors

Limited capacity (16GB per part)

Limited mm distance from the CPU

Fast yet relatively **low power**

Very $$$$expensive

Or you can steal your neighbor's memory!

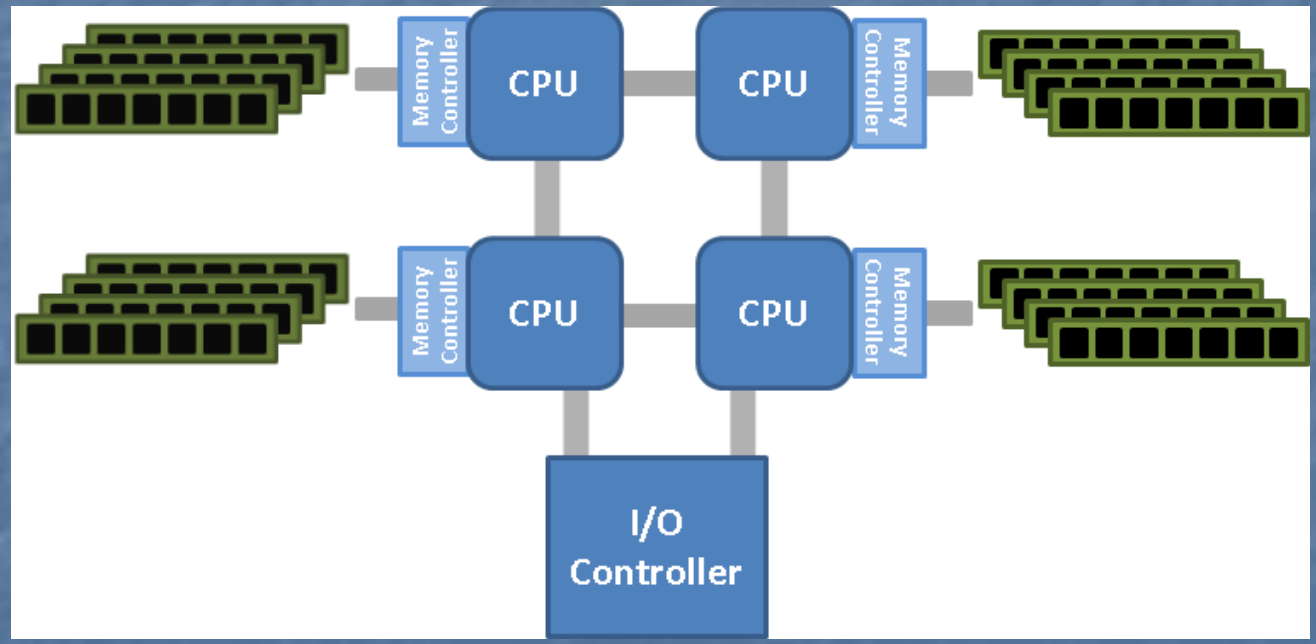
Resource Pooling via NUMA

Non-Uniform Memory Architectures (NUMA) have been common ways to pool memory resources

Buses such as HyperTransport and Ultra Path Interconnect have been around for decades

These NUMAs created a tier of resources

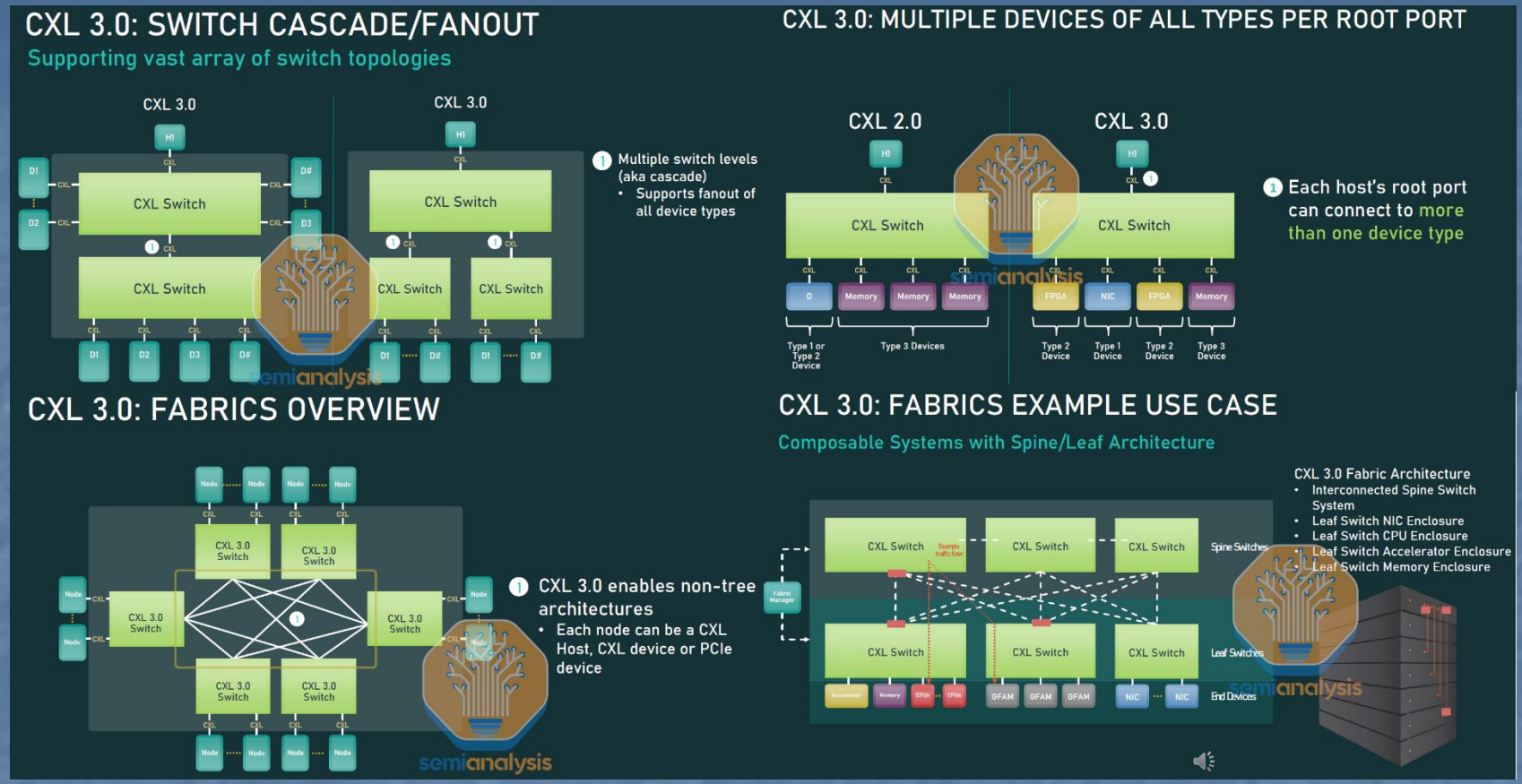
- Fastest memory attached to CPU
- Slower memory one hop away
- Slowest memory two hops away



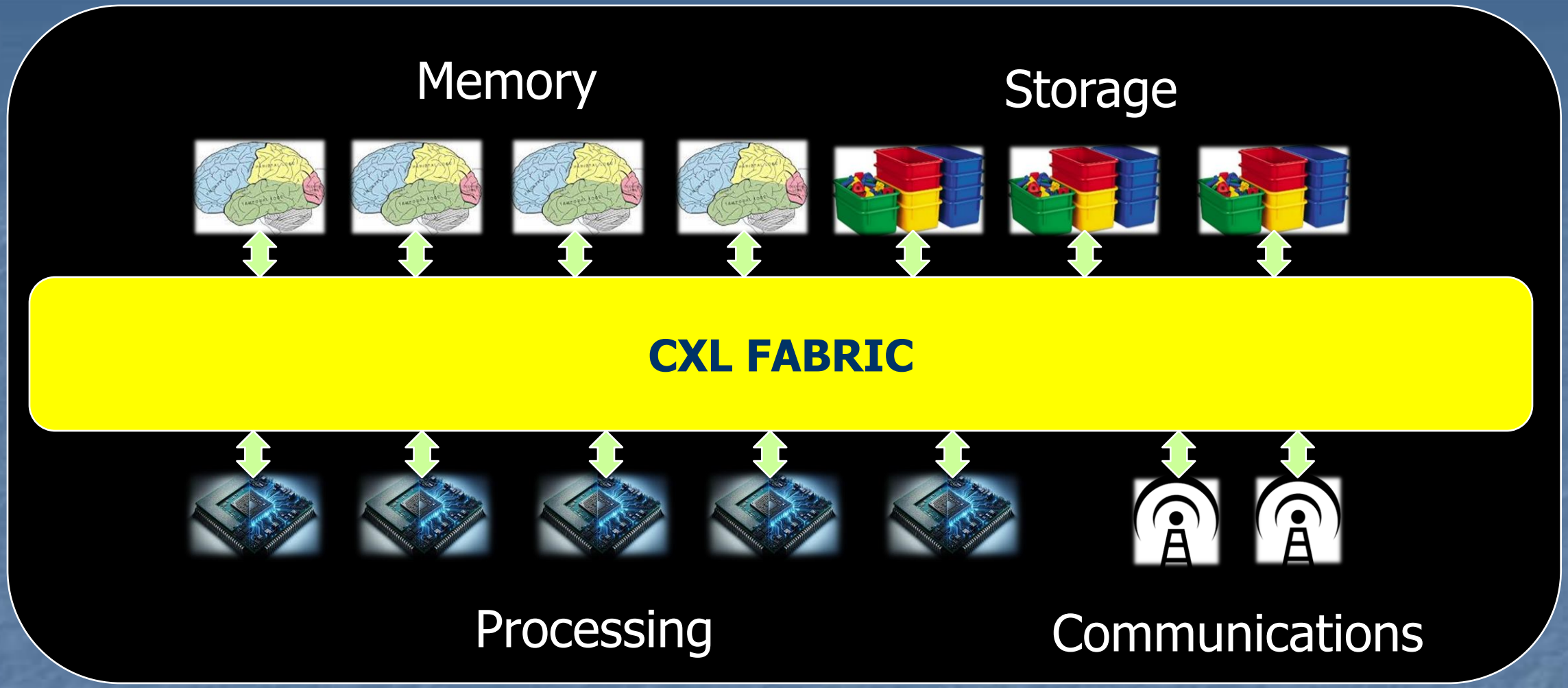
Proprietary NUMA buses inhibited the industry for fabric based computer complexes

For decades, it was impractical to create a connection fabric for resources

In March 2019, all that changed with the introduction of CXL



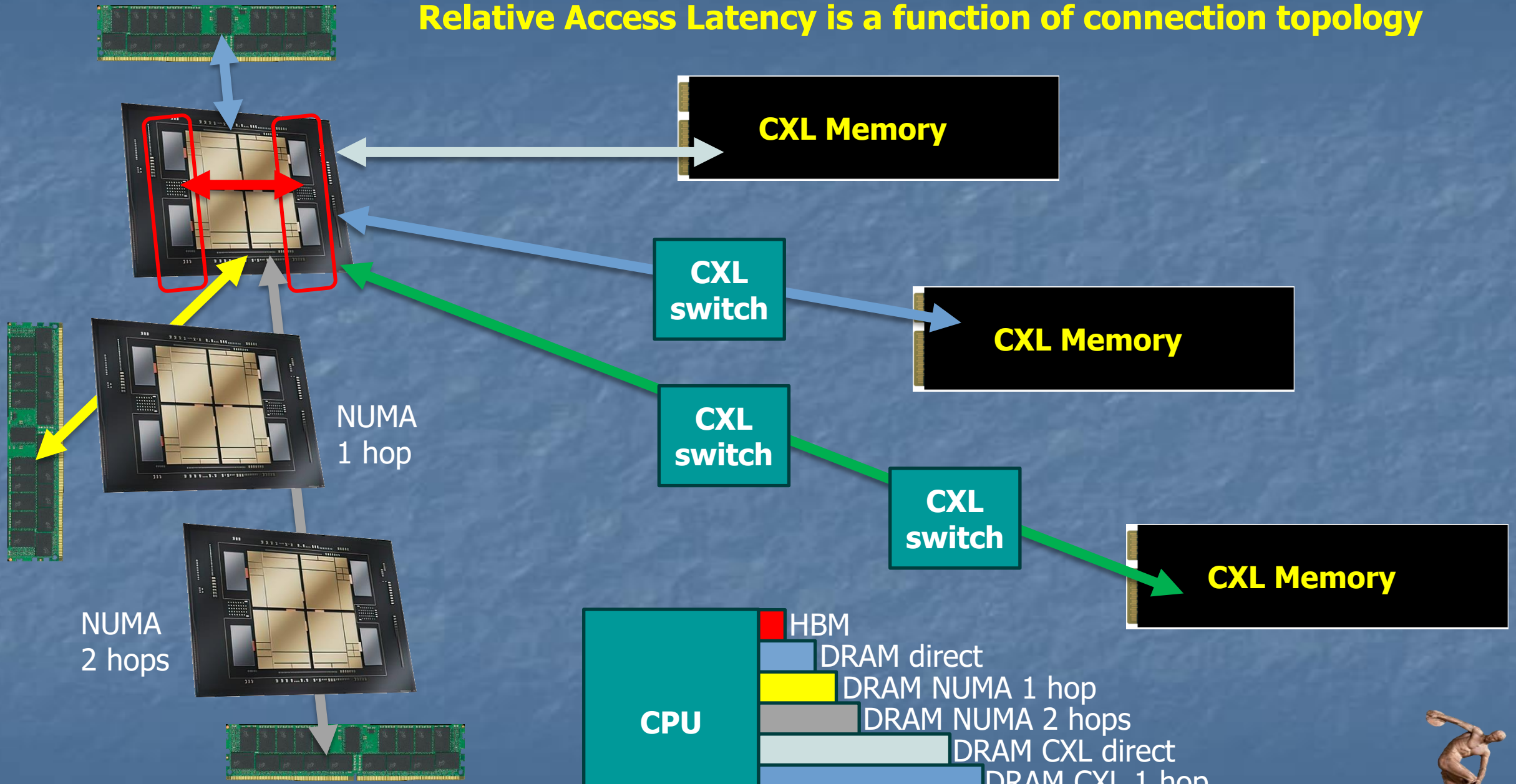
CXL: Compute Express Link changed systems architecture



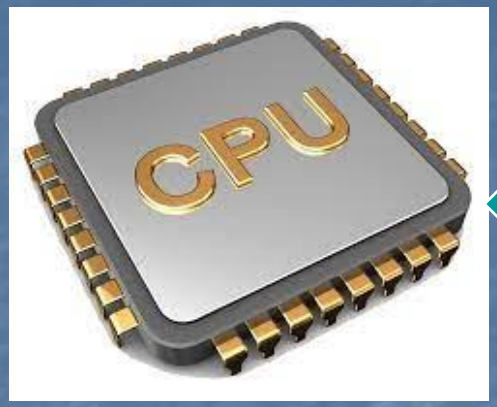
CXL virtualized all resources into a consistent address space

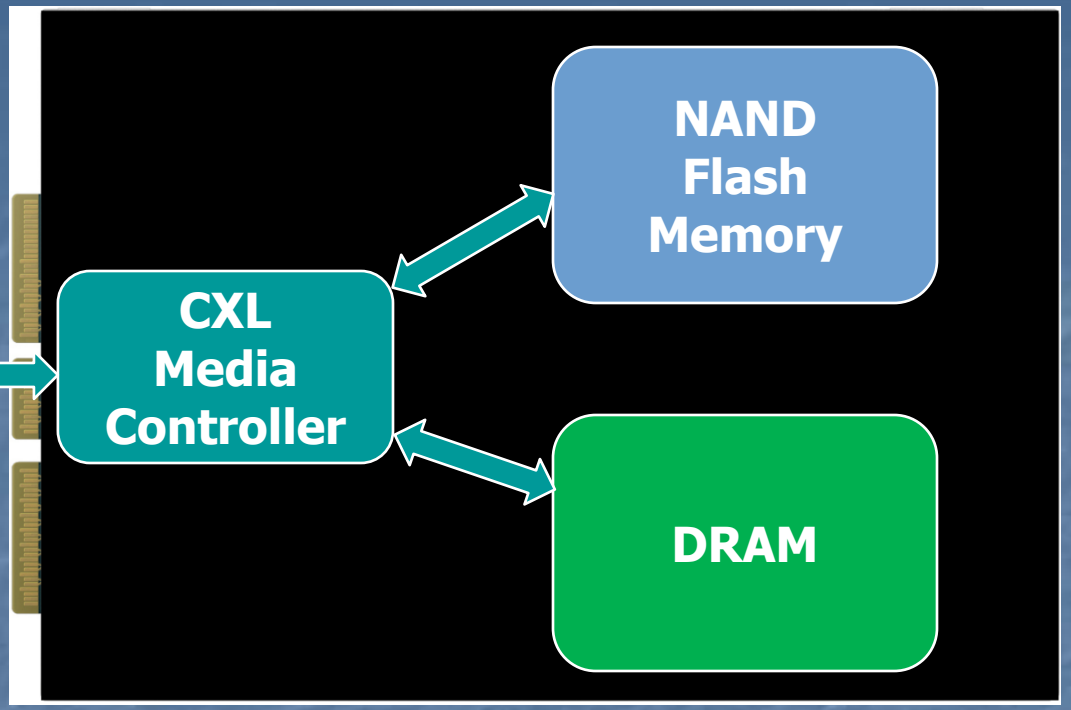
IT can mix and match processing, memory, storage, comms

Relative Access Latency is a function of connection topology



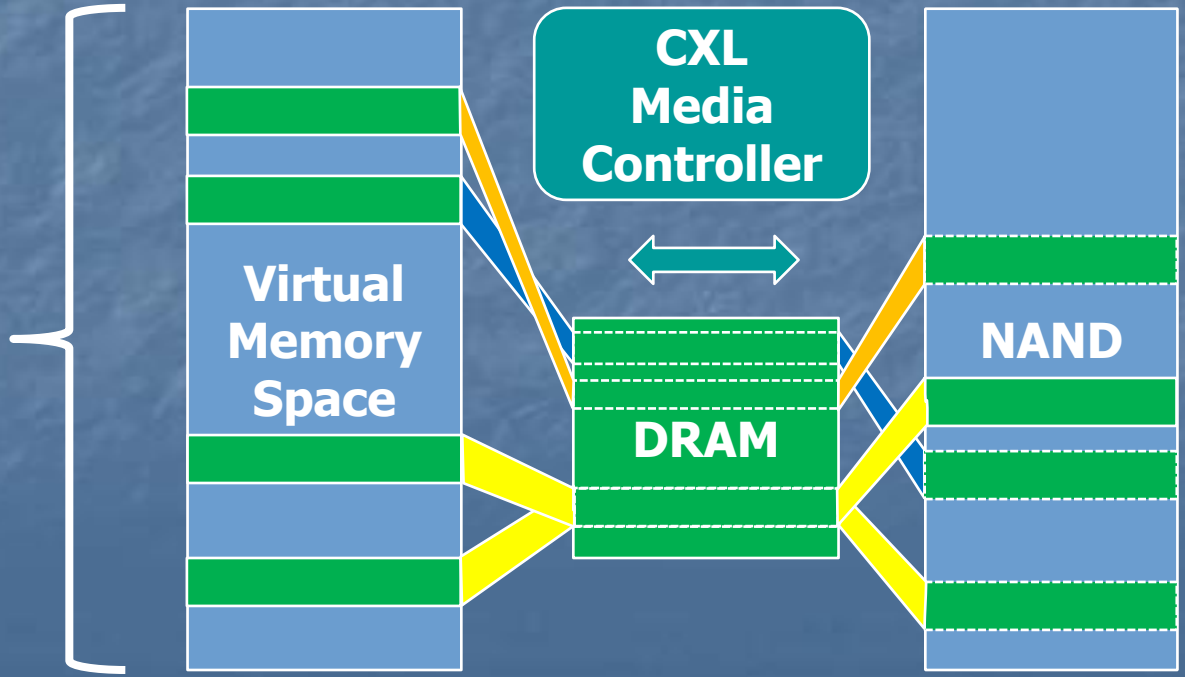
Hybrid Memory: A Virtualization

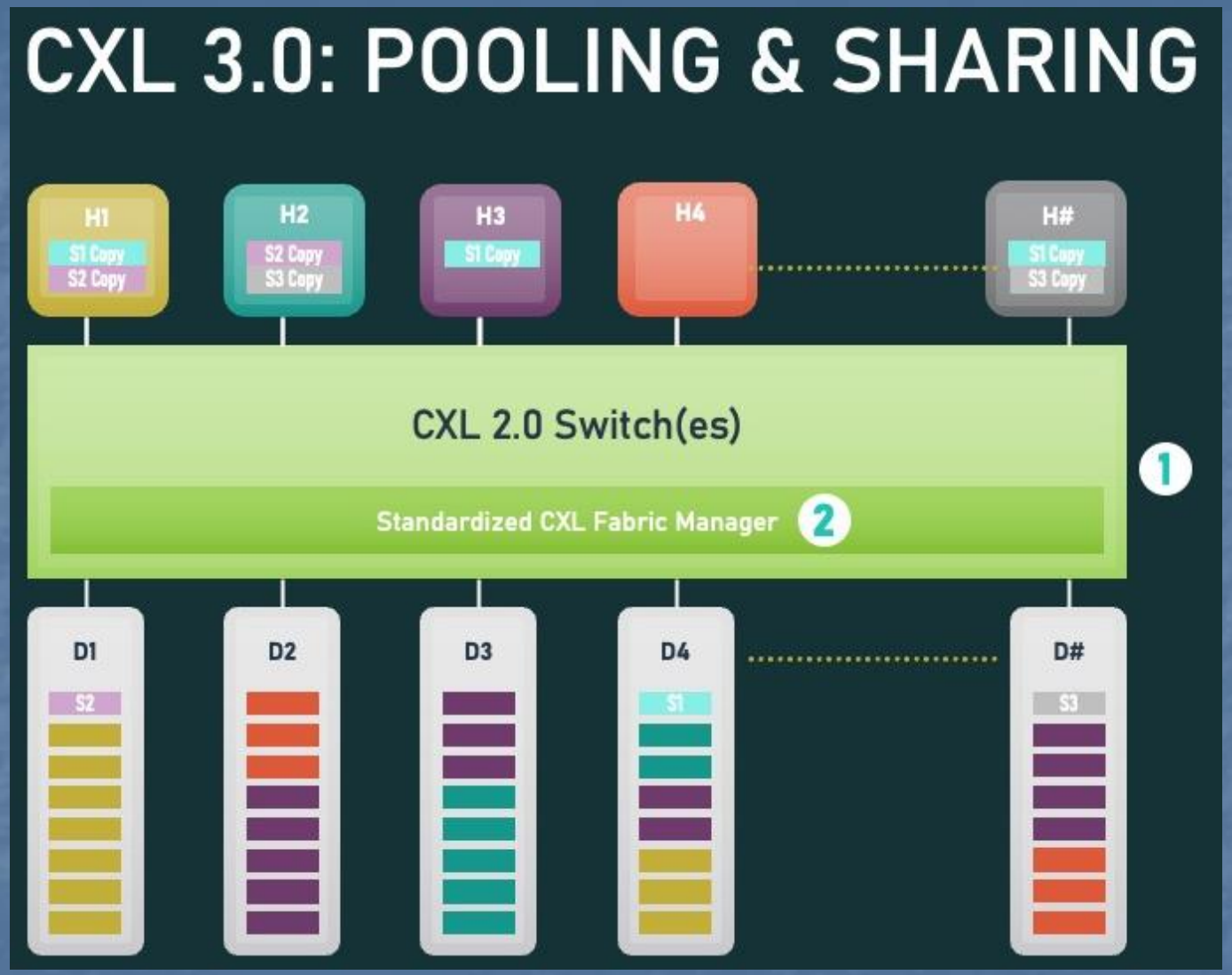




CPU sees a **hybrid DRAM + NAND** module as a linear RAM with the larger capacity of the NAND

As the DRAM resource is depleted, can be flushed and another block can replace it



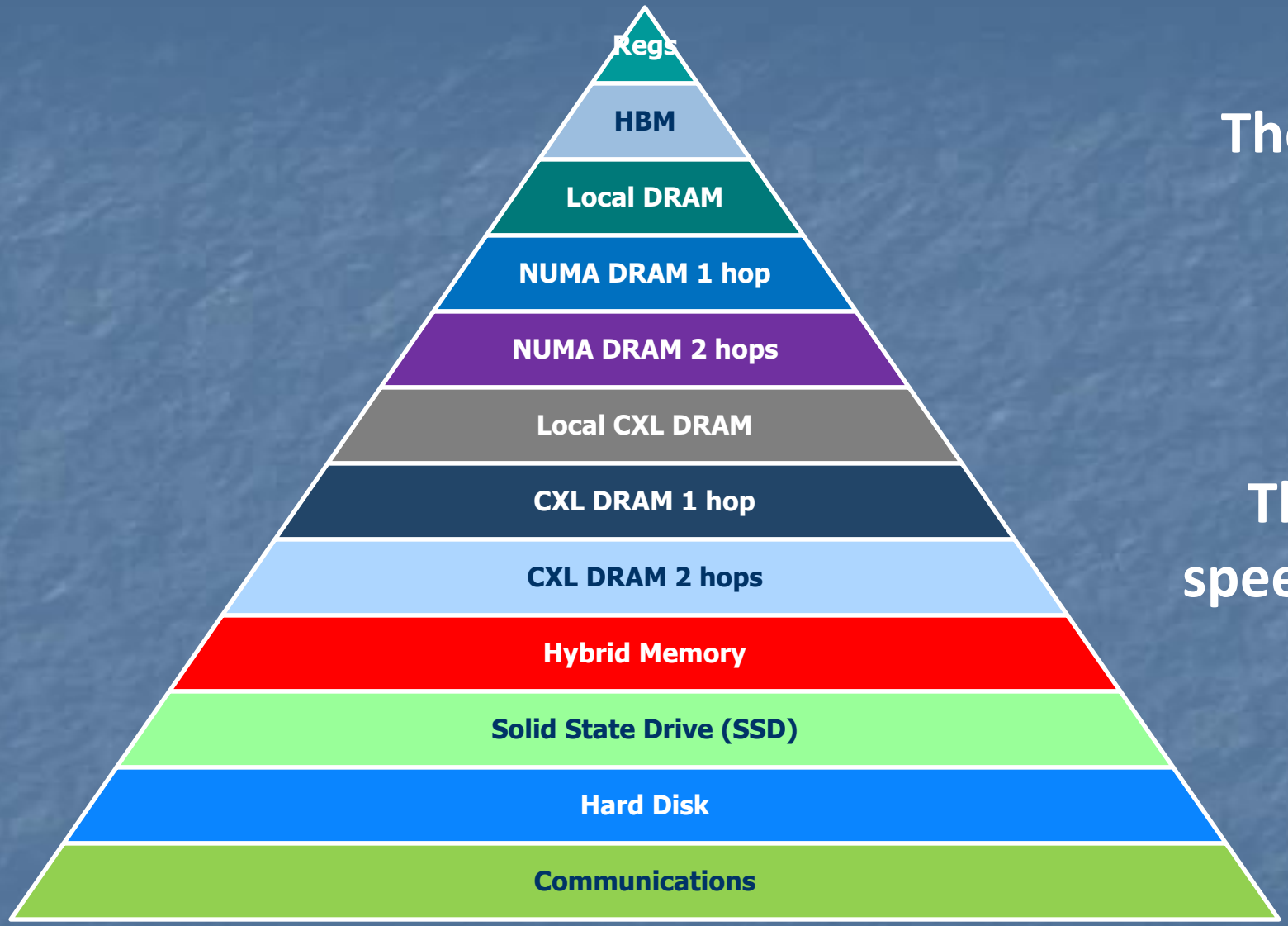


The CXL fabric can be used to improve memory usage

Memory pooling allows each memory resource to be broken into chunks and allocated to different processors

Memory sharing allows multiple processors to share a memory chunk

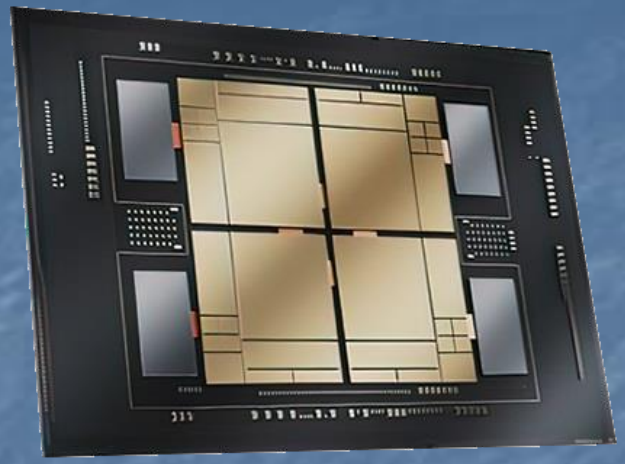
Example: Multiple **AI engines may share** a machine learning data set

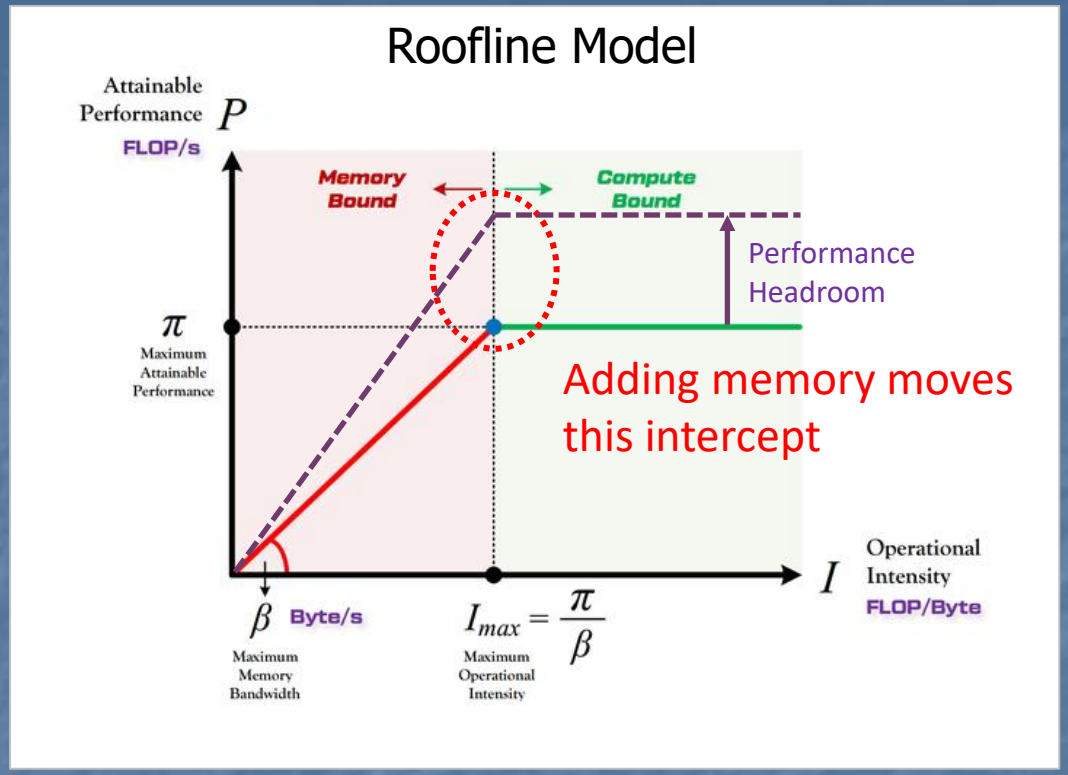


The resource tier map got more complicated

…but…

The same factors apply:
speed, latency, capacity, cost





Artificial intelligence is particularly sensitive to memory capacity

Once an AI application hits the intensity wall, performance peaks

Some Large Language Model applications are splitting into Small Language Models to avoid the intensity wall

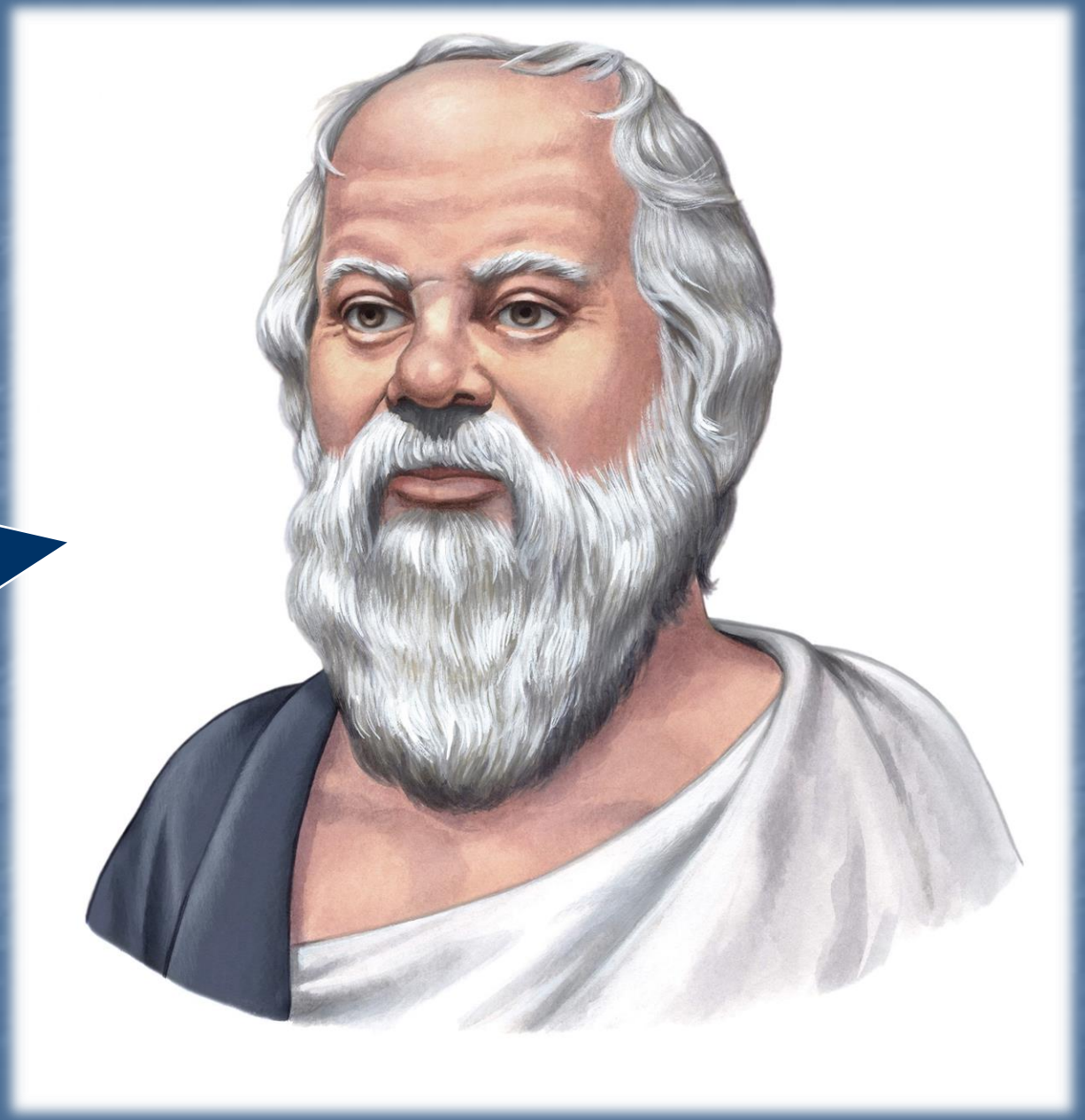
Inference is not as good with SLMs

Adding memory allows LLMs to process complex data sets

CXL memory pooling is a great asset

Όλο αυτό το υλικό!

Τι πρέπει να κάνει ένας σπασίκλας λογισμικού?



* All this hardware stuff!

What's a software nerd to do?

Personal story

In the 1980s I wrote the boot code for this voicemail system

I wrote the memory test algorithm in PL/M (C-like programming language)

Execution time = 27 minutes

I dumped the output of the PL/M compiler to assembly and tweaked the code

Execution time = 7 minutes

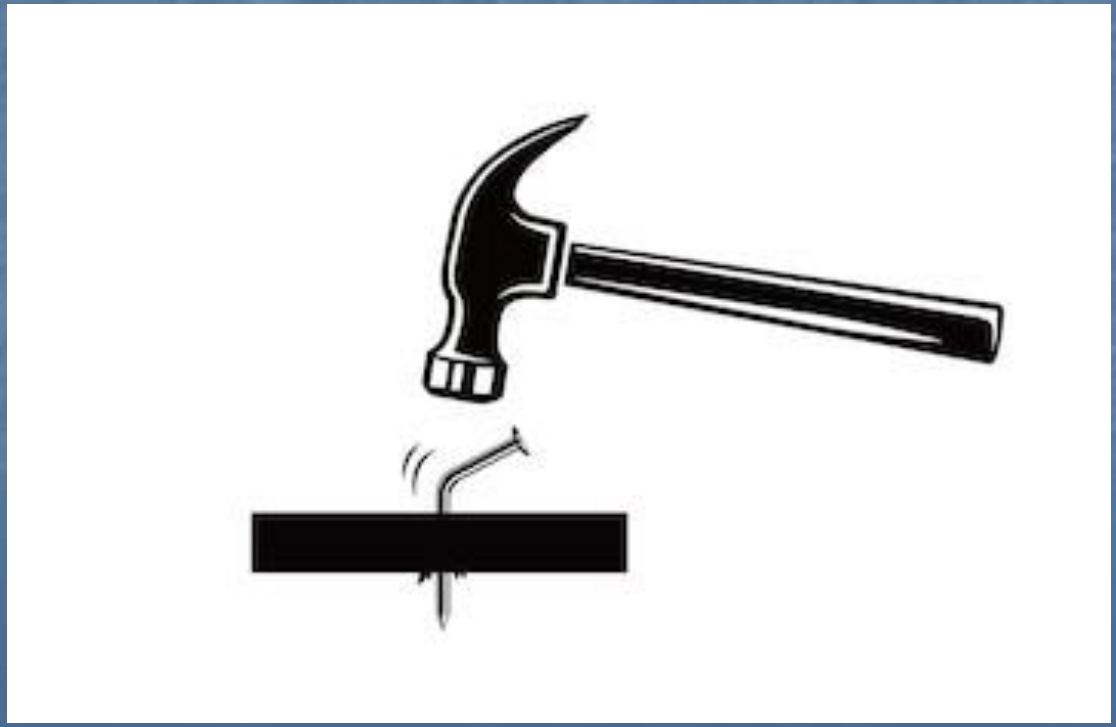
I used clock counting and loop unrolling to recode

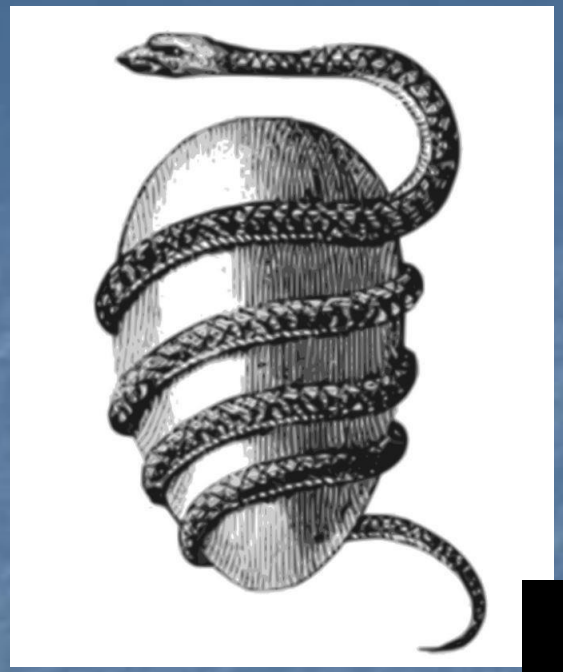
Execution time = 7 seconds (231X better than PL/M)

Lesson learned: high level languages can cause great evil

One of my favorite sayings:

"When all you have is a hammer,
everything looks like a nail."





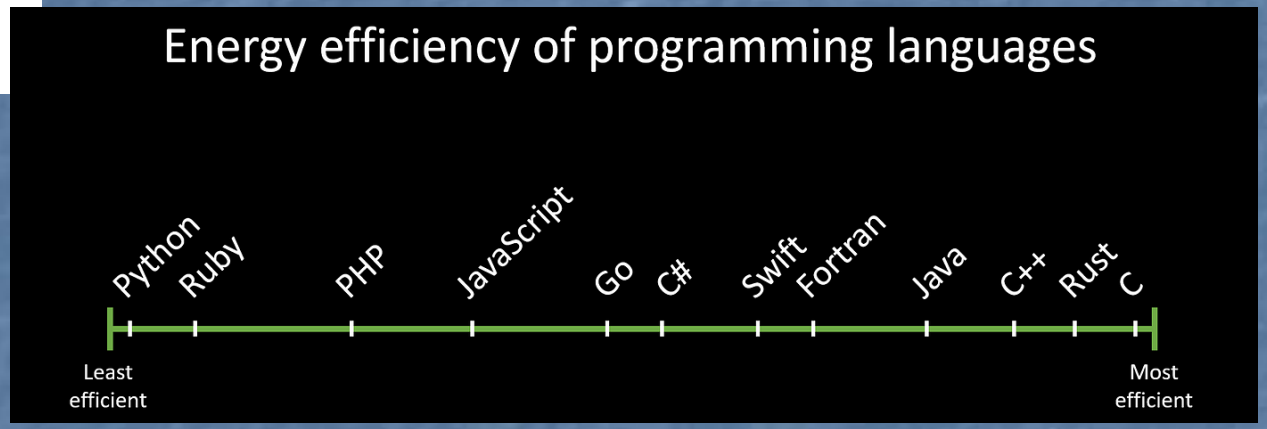
"Python Meets Planet Earth"

Sometimes the development language is fixed

When you have a choice, consider efficiency

Python may be great for AI

but it is overkill for a calendar program



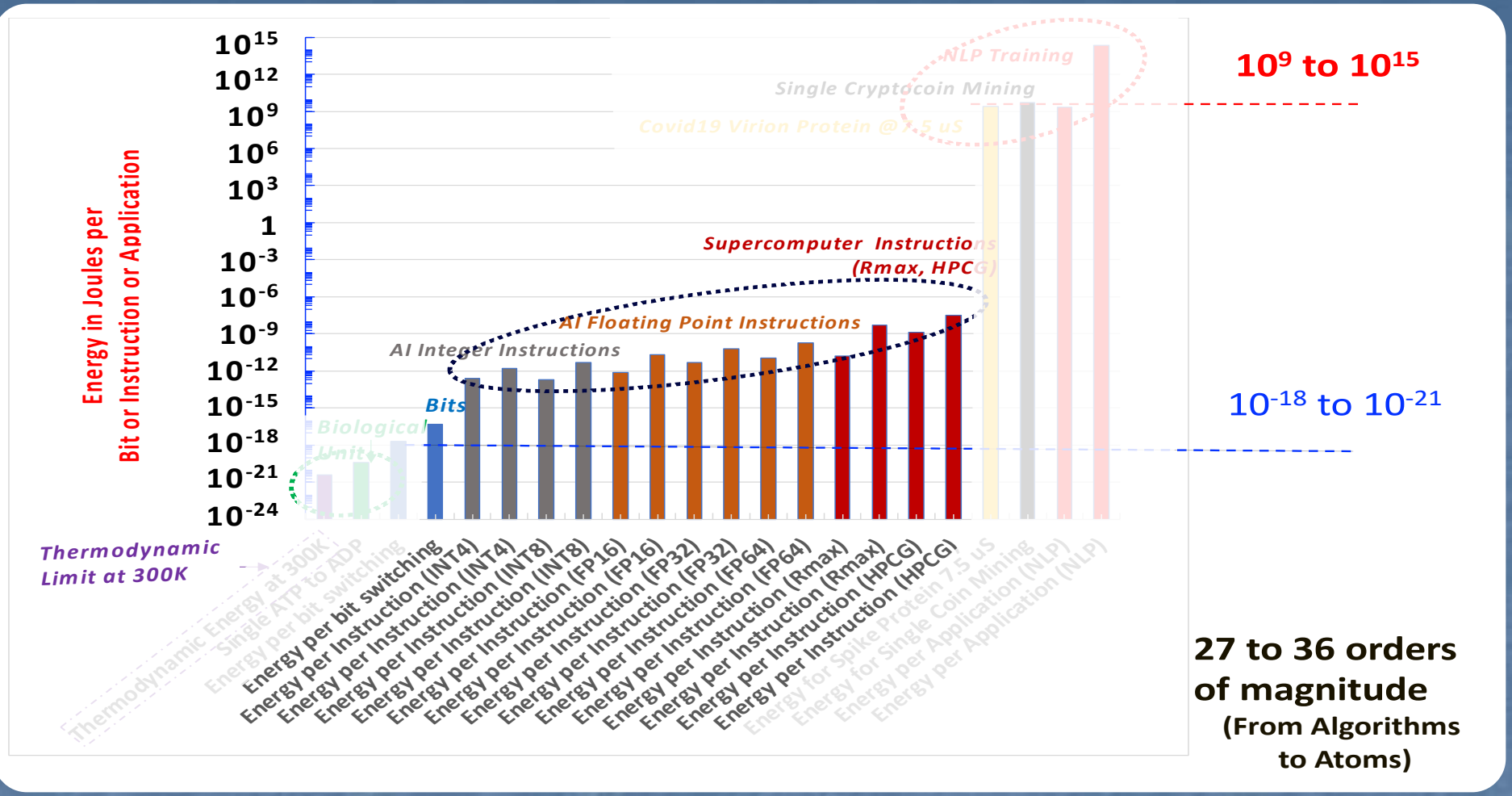
Call to action includes BETTER COMPILERS

Interpreters are great for prototyping but are power pigs

Variable size matters

Orders of magnitude differences in power used

Don't be lazy



There are many APIs for accessing memory and storage

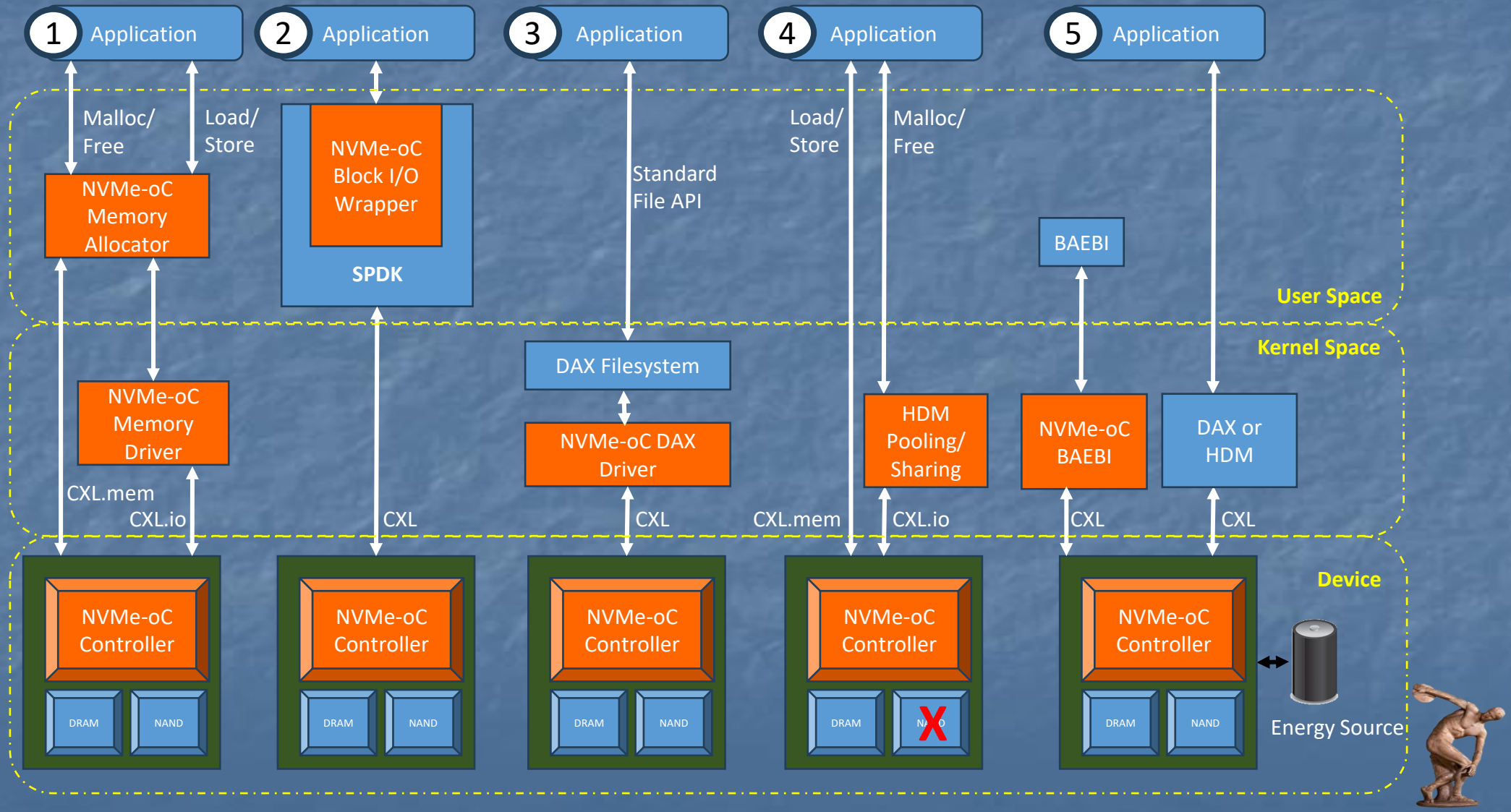
All have different uses and different efficiencies

Every application may have legacy mechanisms

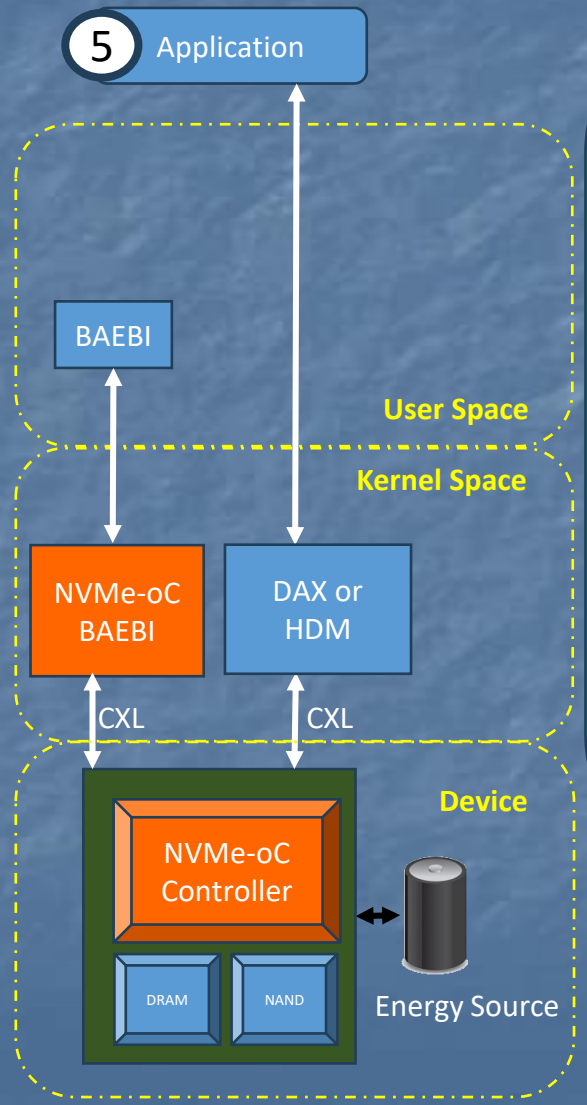
New applications can use newer more efficient methods

E.G., DAX mode memory mapped filesystems

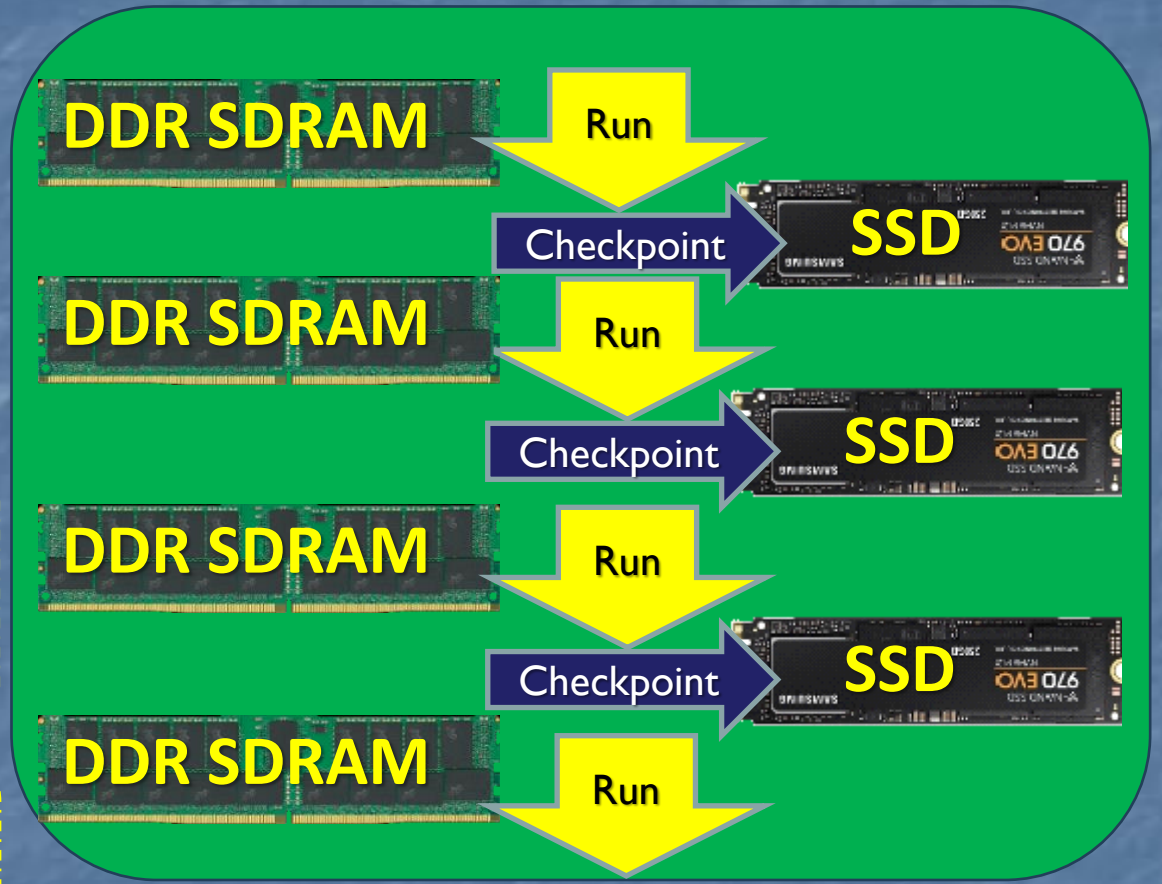
Avoid OS traps to improve efficiency

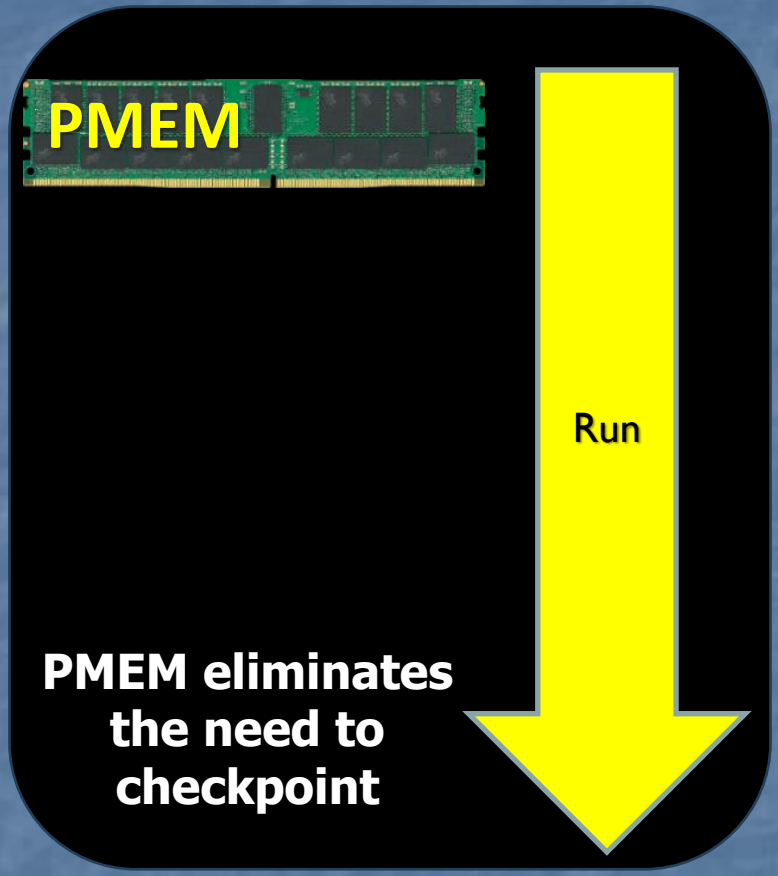


Persistent memory is not just about data integrity



Applications are forced to checkpoint contents periodically because of volatile DRAM





Checkpointing consumes ~8% of system throughput and power on average

Matrix math is heavily deployed in AI algorithms and Python programming

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Adding a parameter can explode the number of calculations that must be executed

Many parameters are not used in some calculations

A lot of rows or columns are one of three values:

-1

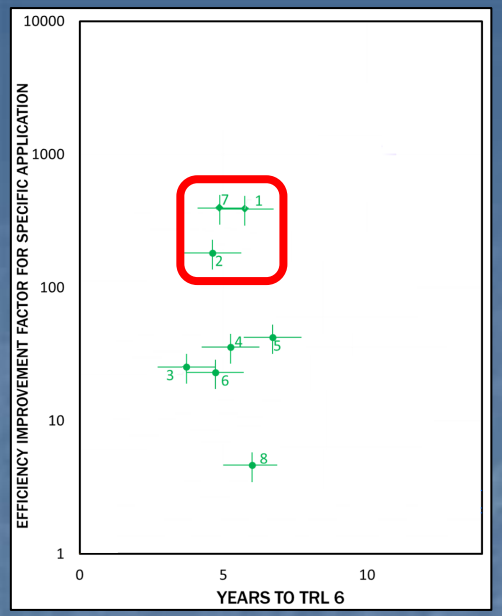
0

+1

Conclusions:

1. A lot of efficiency can be gained by avoiding under-utilized parameters
2. Picking appropriate data types can reduce processing overhead
3. Low-hanging fruit: compressed memory for sparse matrices

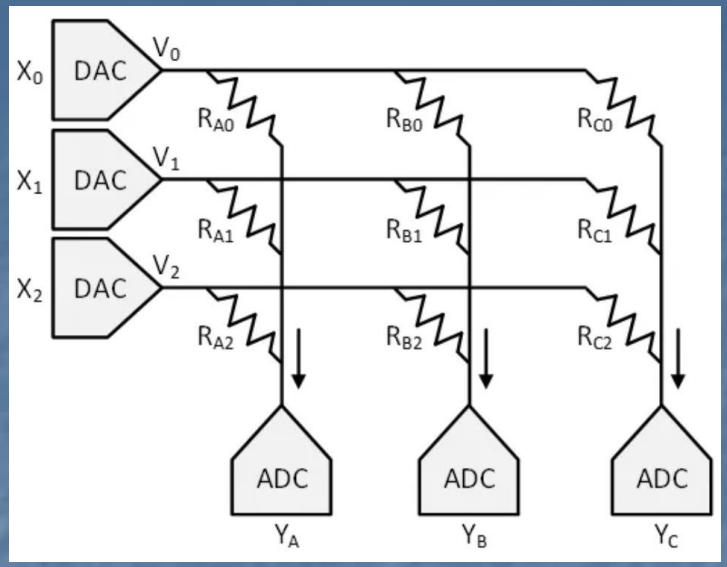
Example of combined HW/SW solution



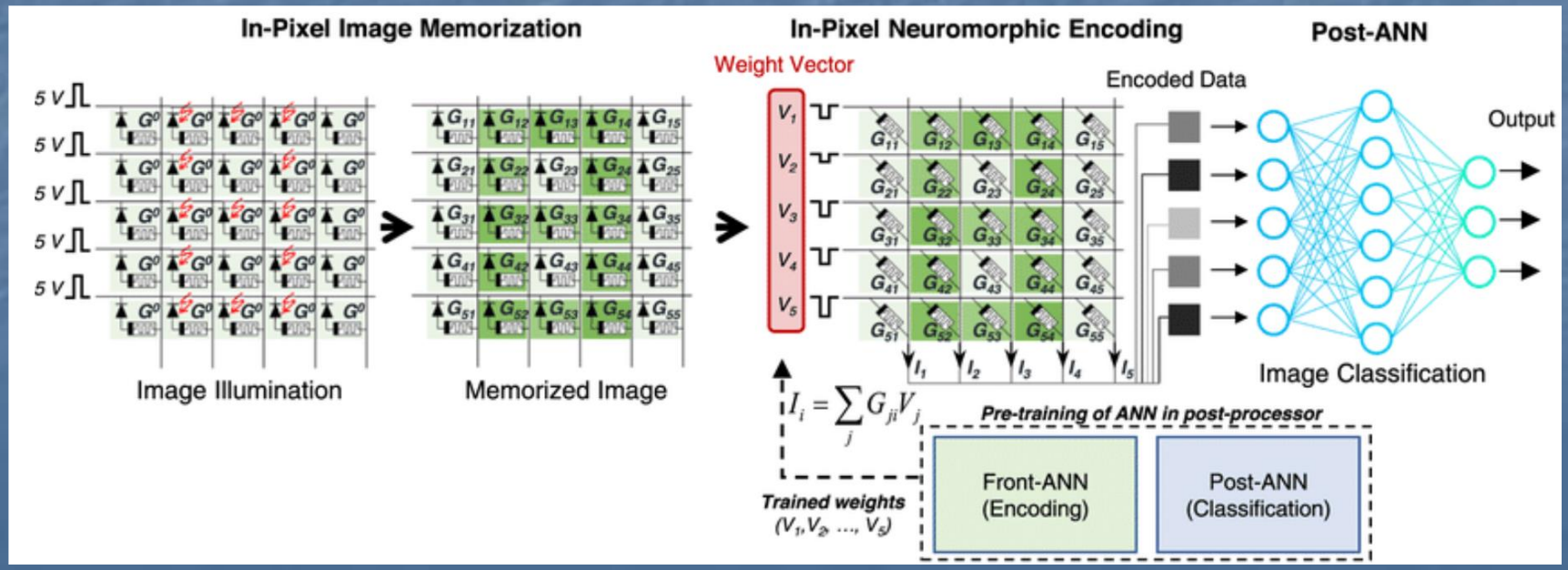
Algorithms and Software

1. Reduced energy for ML algorithms
2. Algorithm-specific energy (tooling)
3. Algorithm-specific energy (benchmarking)
4. Languages, compilers, and runtime systems
5. Communication protocols
6. Homomorphic encryption
7. Software for emerging architectures
8. Computational reliability

Hardware multiply-accumulate units in NPUs

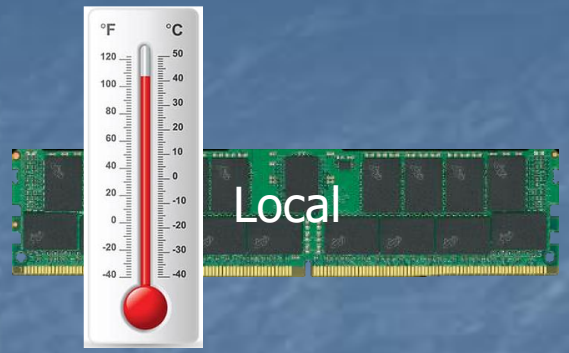


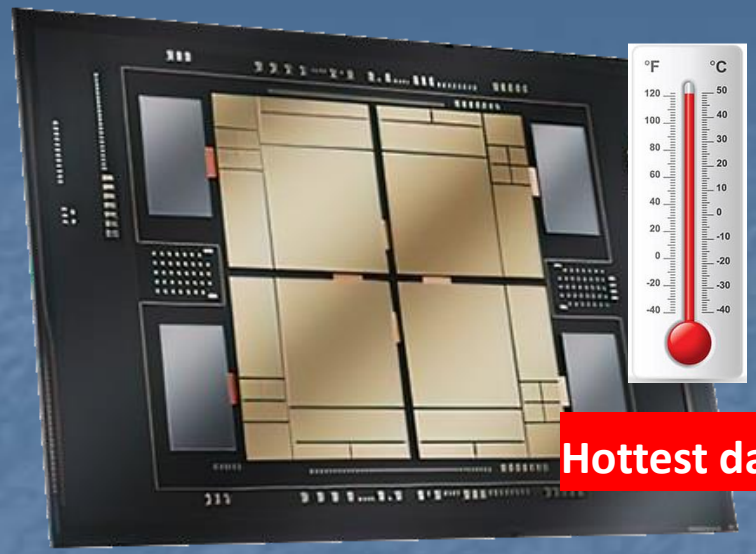
Reduces the total energy used by eliminating repetition



Consider the **temperature** of your data

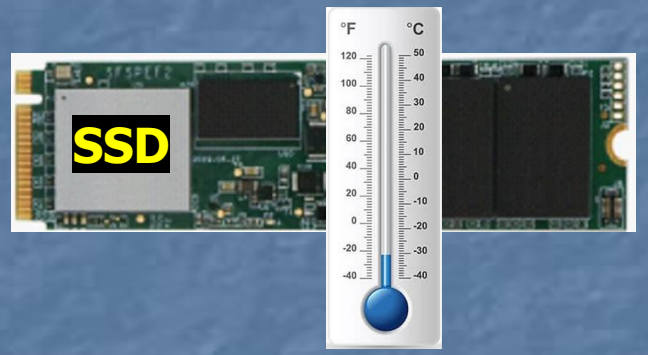
Coldest data

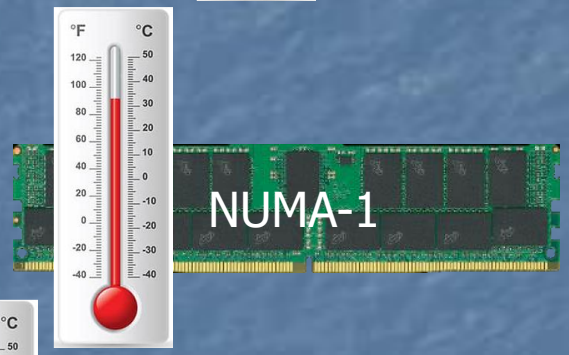


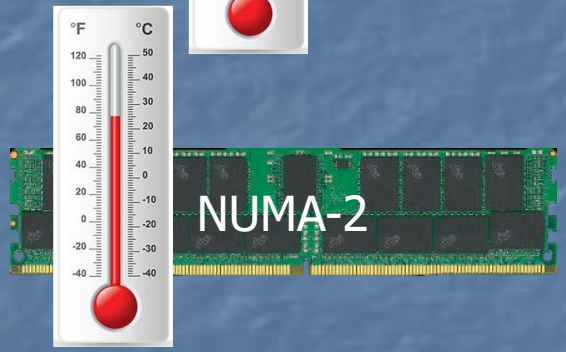


Hottest data

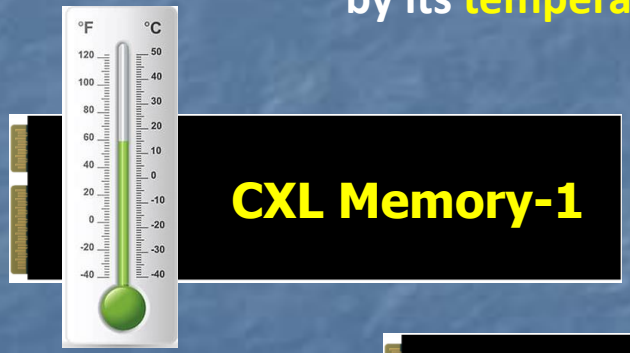
Network

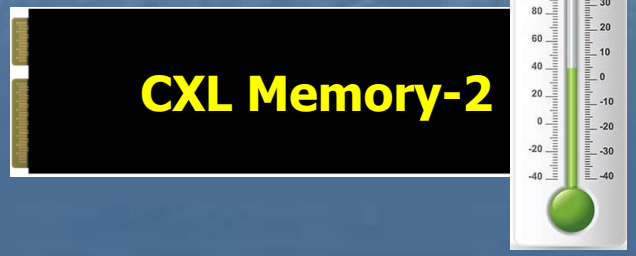






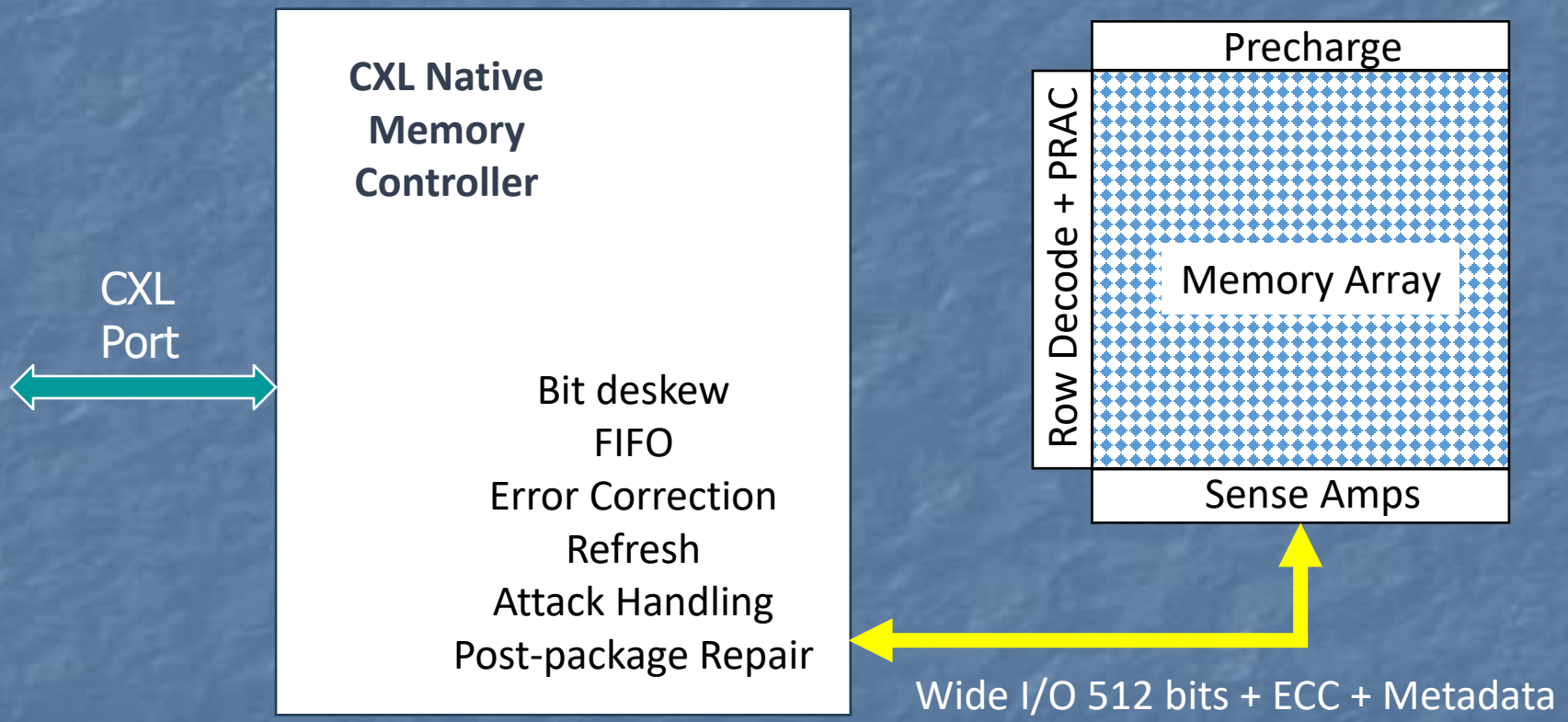
Map your data into the appropriate memory tier by its temperature rating





What's this nerd doing to make things better?

Rethinking RAM design



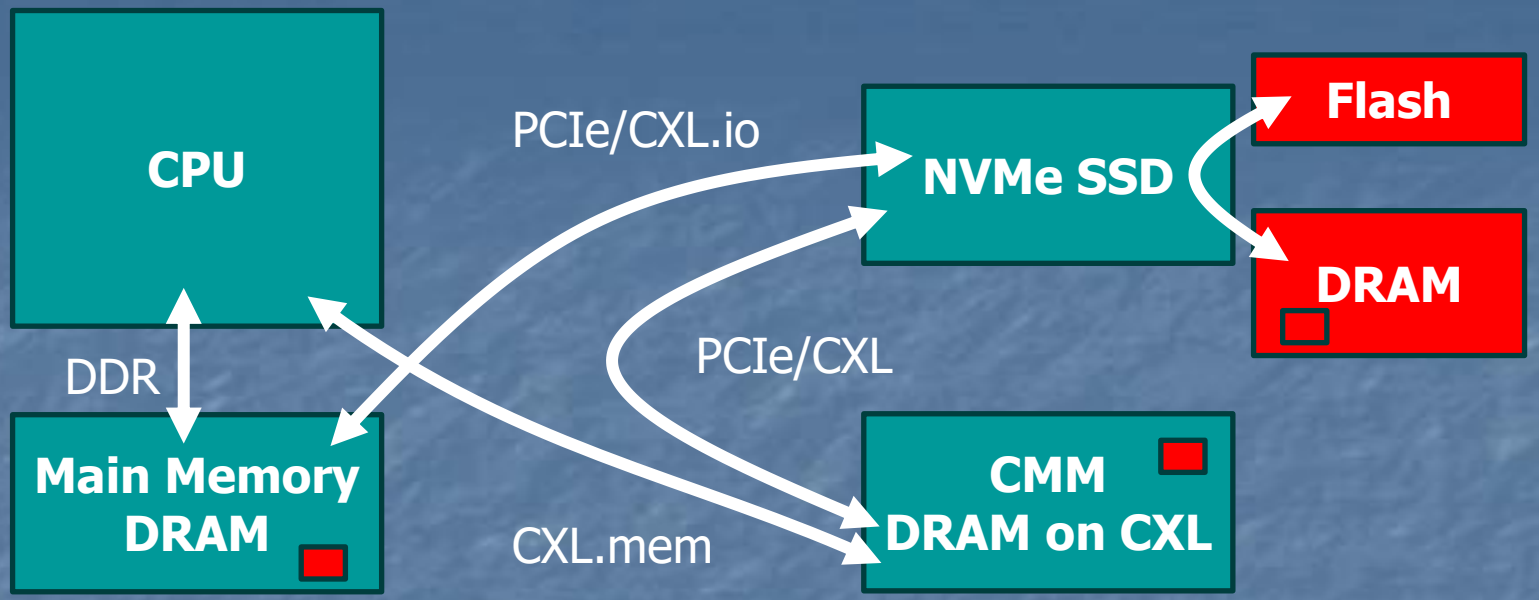
Internal memory bus is 512 bits wide

= 64 bytes = one cache line

Activation + rewrite still required

Resulting efficiency is 50%

This is a 2000X improvement over current DRAM rank approach



Current systems separate storage and memory

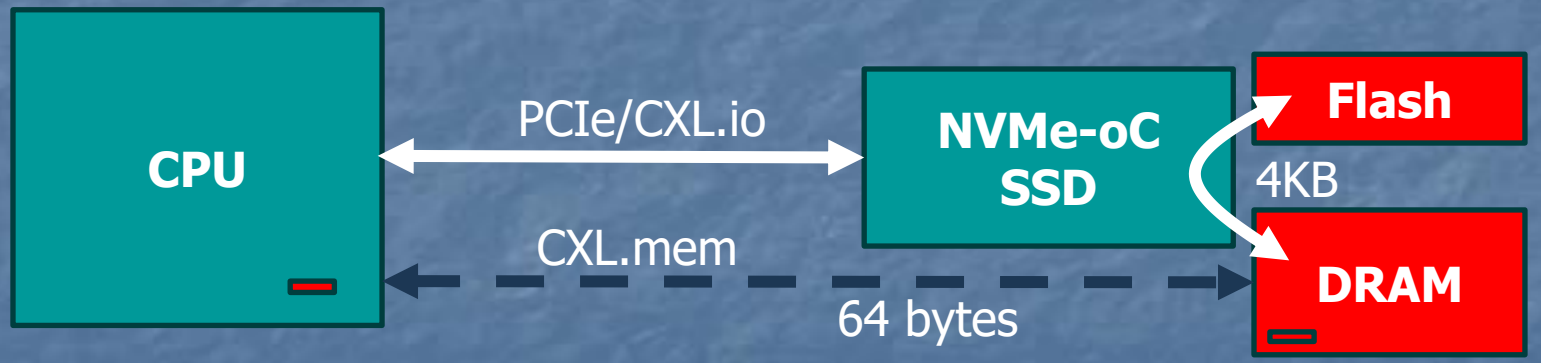
Data is copied between storage and memory as needed in 4KB blocks

With 100 bytes used, efficiency is 2.5%

New high efficiency design combines storage and memory

4KB transfers kept on the module

CPU accesses only the data it needs in cache line size chunks



With average access of 100 bytes, efficiency increased to 78% (31X better)

Key Takeaways

- Hardware matters and affects your application
- Architectures are evolving and making your life complicated
- Programmers can help us deal with the energy crisis
  - Choose the right language for the job
  - We need better compilers
  - Optimizing a matrix can have a huge impact
  - Be hardware aware; know your tiers
  - Put data where it belongs
  - Not all APIs are the same – some are more efficient, e.g., no OS traps
  - Persistent memory enables rethinking process integrity
- I'm still trying to figure out how to implement sparse matrices – on the fly memory compression?

We are part of the problem so we must be part of the solution

We have an obligation to bring the next generation into the process

BRDG
bridge to connect

Mentoring STEM students in "beyond the job description" skills

https://www.bridge-to-connect.org/

CHIPS funding coming to have students help with efficiency projects

What Else Can We Do?

- Audience feedback…

Thank You

Bill Gervasi

bilge@discobolusdesigns.com