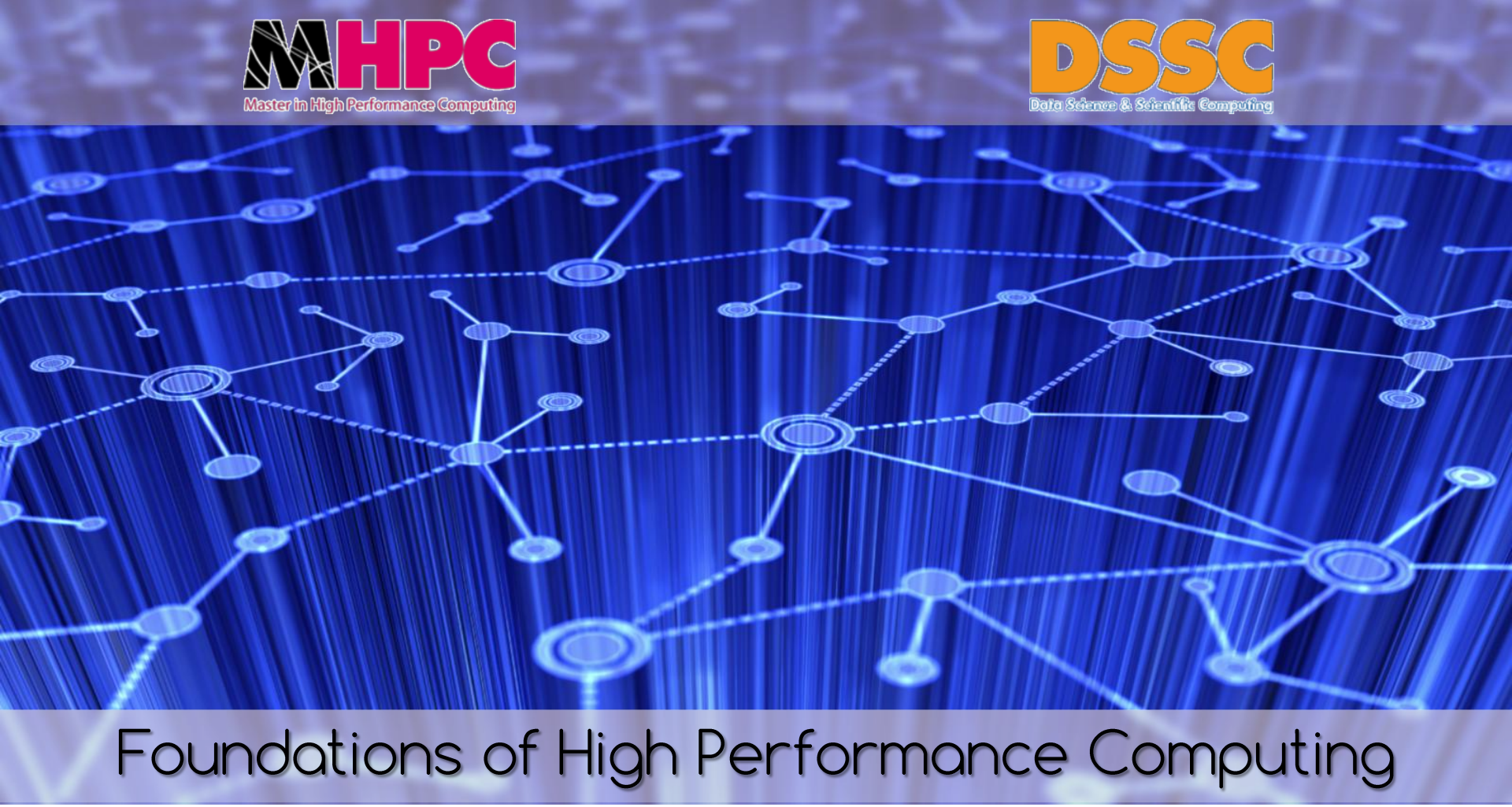


Approaching the EXA-Scale epoch

Why this talk ?

Both **data-intensive** and **computation-intensive** applications will be **forced** to face an epocal **major shift** in the computation paradigm, which indeed started several years ago.

Why this talk ?

"**CRUCIAL PROBLEMS** that we can only hope to address computationally **REQUIRE US TO DELIVER EFFECTIVE COMPUTING POWER ORDERS-OF-MAGNITUDE GREATER THAN WE CAN DEPLOY TODAY**."

DOE's Office of Science, 2012

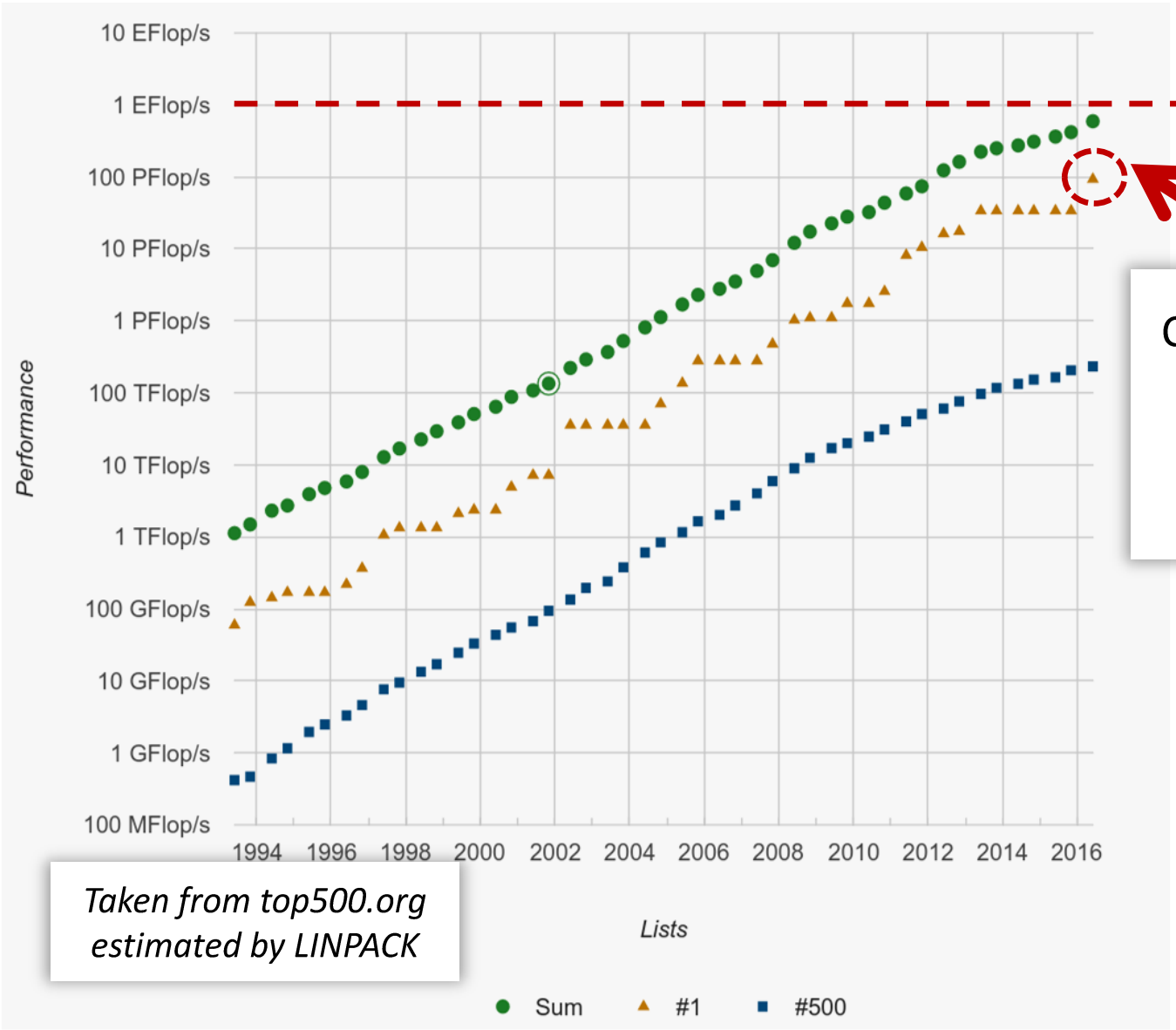
Why Exa-Scale ?

"EXA-scale" is the necessary upscale step that HPC needs to achieve in the next (few) years.

Basically, it is defined as the frontier of a **sustained** performance around **10^{18} flop/s**.

There are profound consequences on the way we design, write and optimize scientific and data-intensive codes.

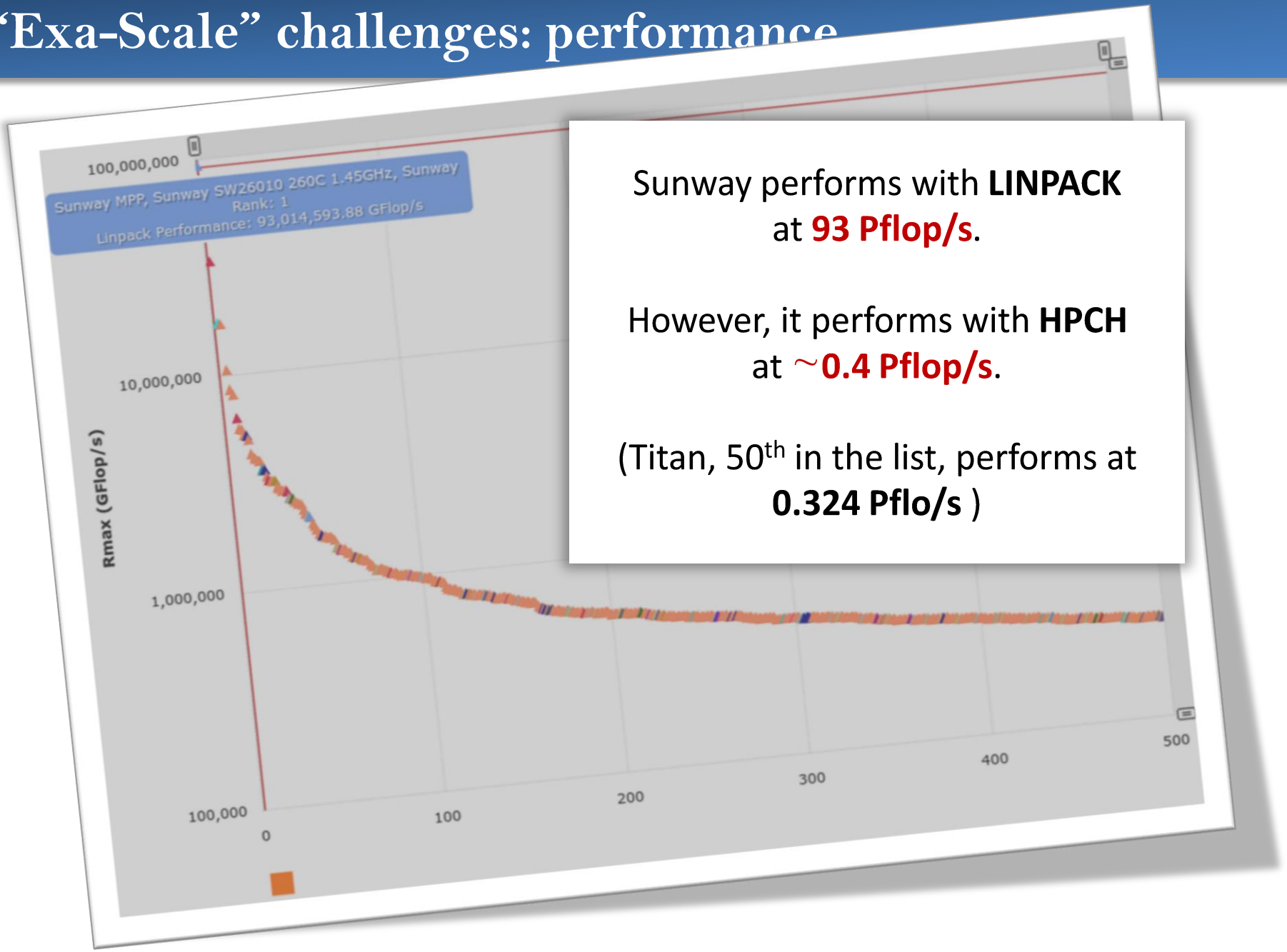
“Exa-Scale” challenges: performance



China’s SunWay has a theoretical *peak performance* of **125Pflops**

Taken from top500.org estimated by LINPACK

"Exa-Scale" challenges: performance



Sunway performs with **LINPACK** at **93 Pflop/s**.

However, it performs with **HPCH** at **~0.4 Pflop/s**.

(Titan, 50th in the list, performs at **0.324 Pflo/s**)

“Exa-Scale” challenges

Message I

Exascale is the achievement of a **sustained performance around 10^{18} Pflops.**

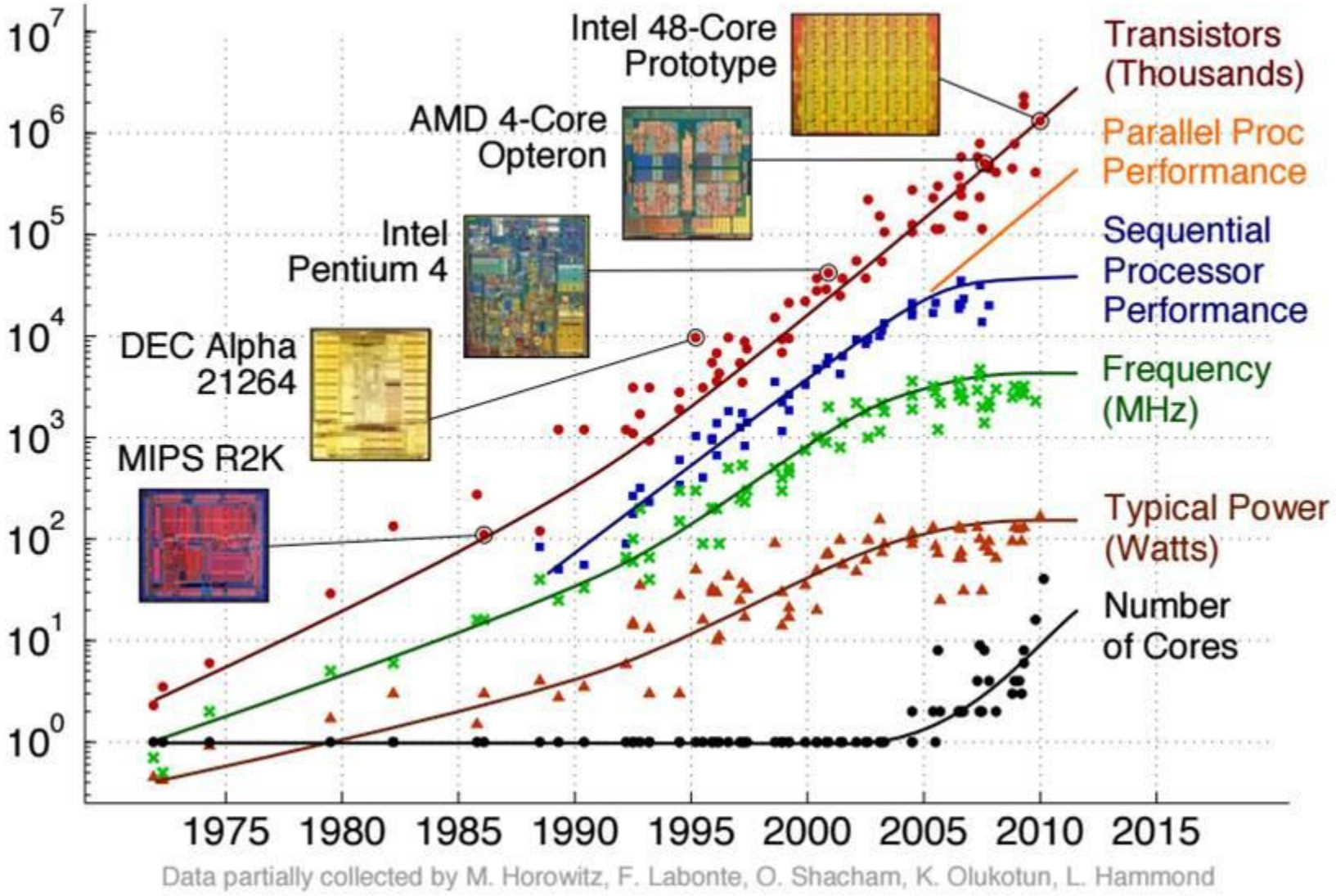
It is a **relative term** pointing to **1000-fold better capability** than that representative of the *petascale*.

Question

Is this achievement dependent only on improvements in **hardware technology** ?

i.e: shall **our codes** plug-in as they are, and just run faster, saturating a exa-scale machine ?

Flash back: why there is no more "free-lunch" ?

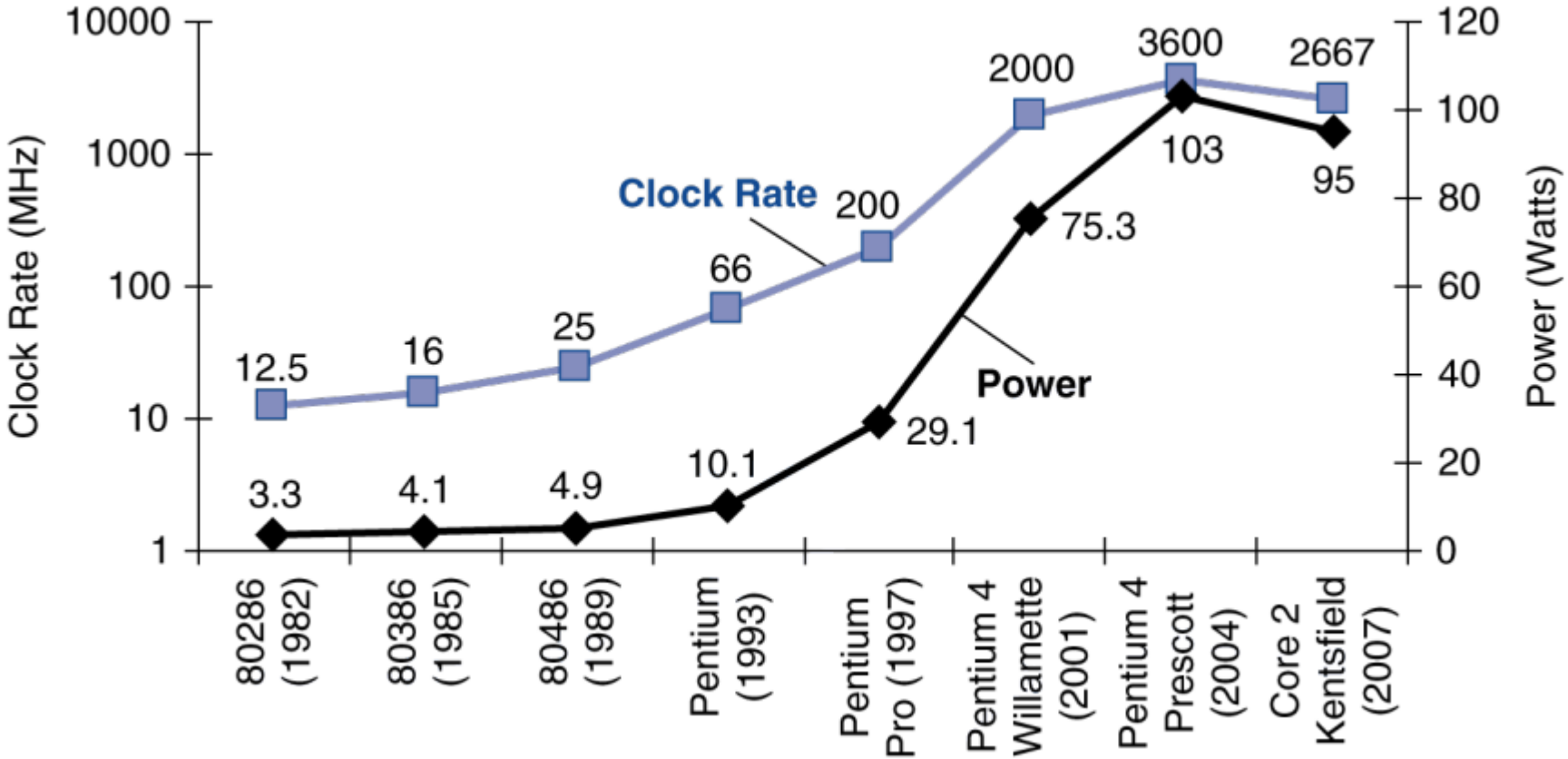


Flash back: why there is no more "free-lunch" ?

Moore's law	Dennard's scaling (MOSFET)
Manifacturing cost/area is costant while the transistors' dimension halves every 1-2 years **→ The number of transistors doubles in a CPU every 1-2 years**	- Voltage, Capacitance, Current scale with λ as $1/\lambda$ - Transistor power scales as $1/\lambda^2$ **→ Power density remains constant**

$$P \propto C \cdot V^2 \cdot f$$

Flash back: why there is no more "free-lunch" ?



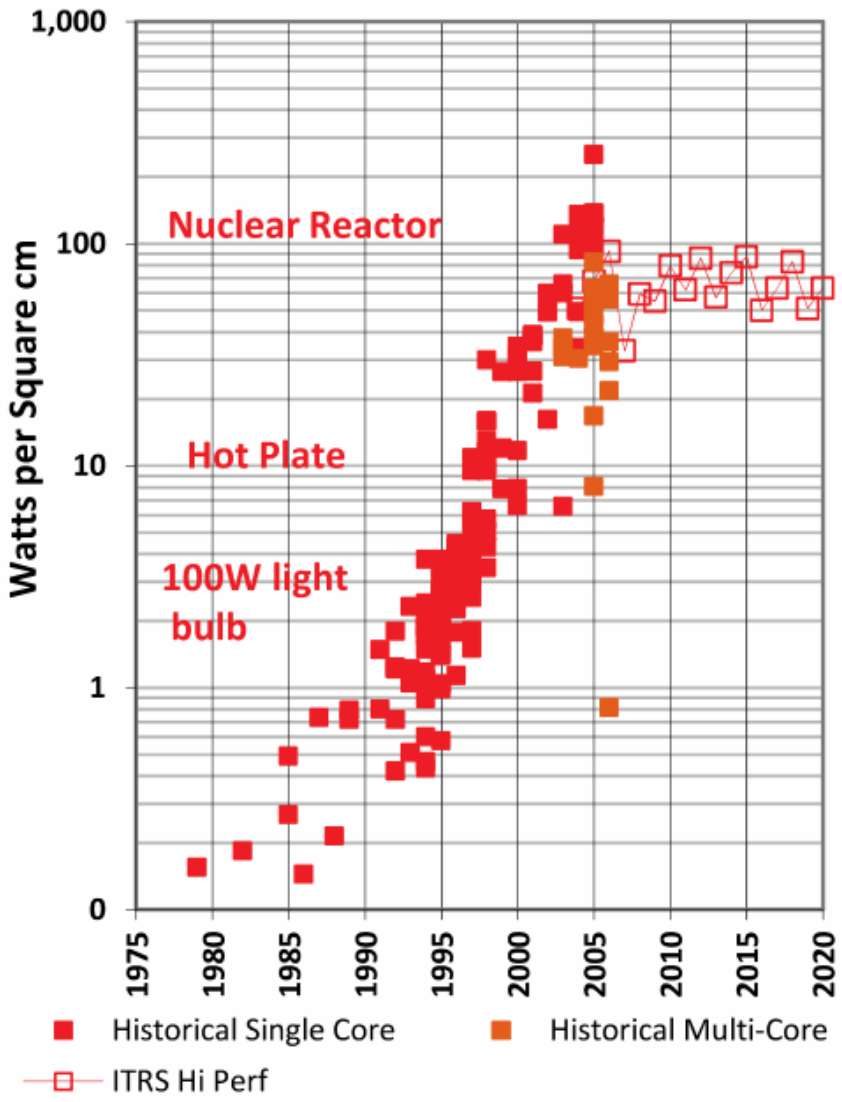
$$P \propto C \cdot V^2 \cdot f$$

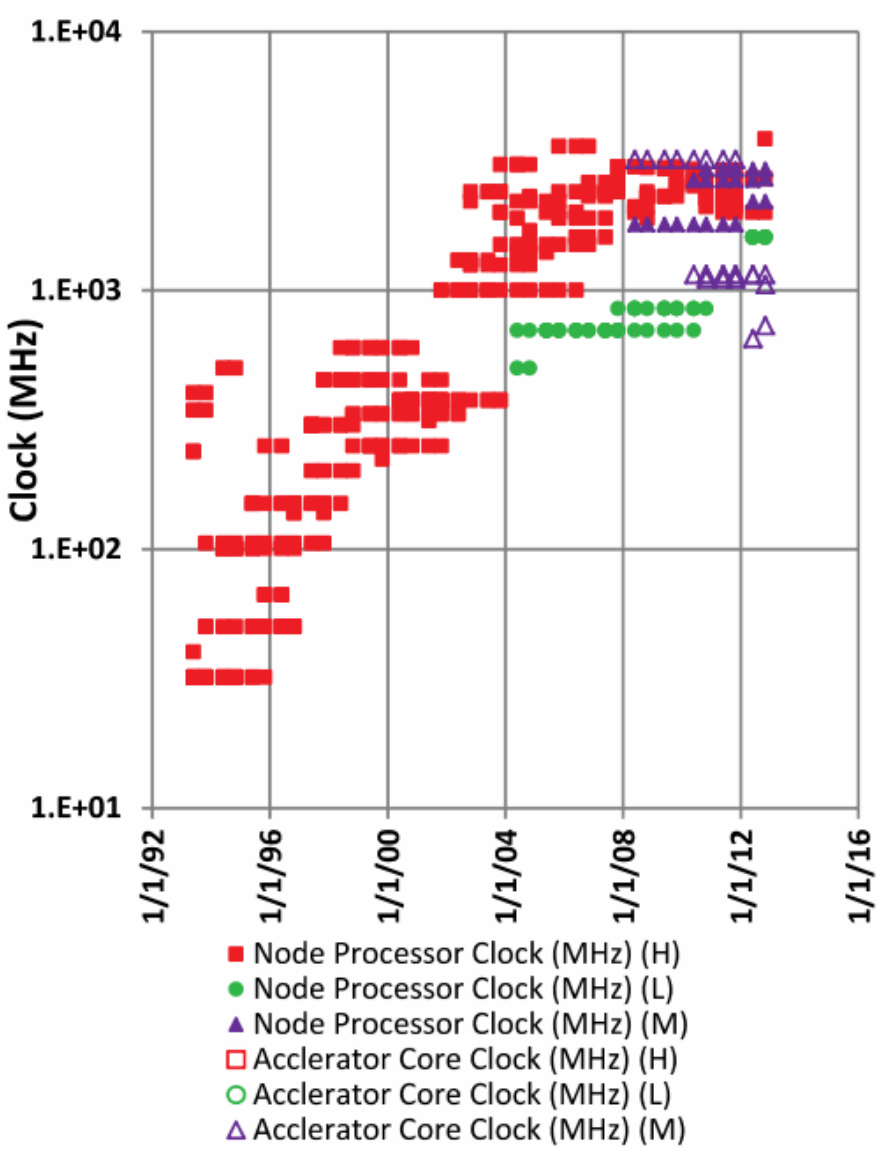
$\times \sim 30$

5V → 1V

$\times \sim 1000$

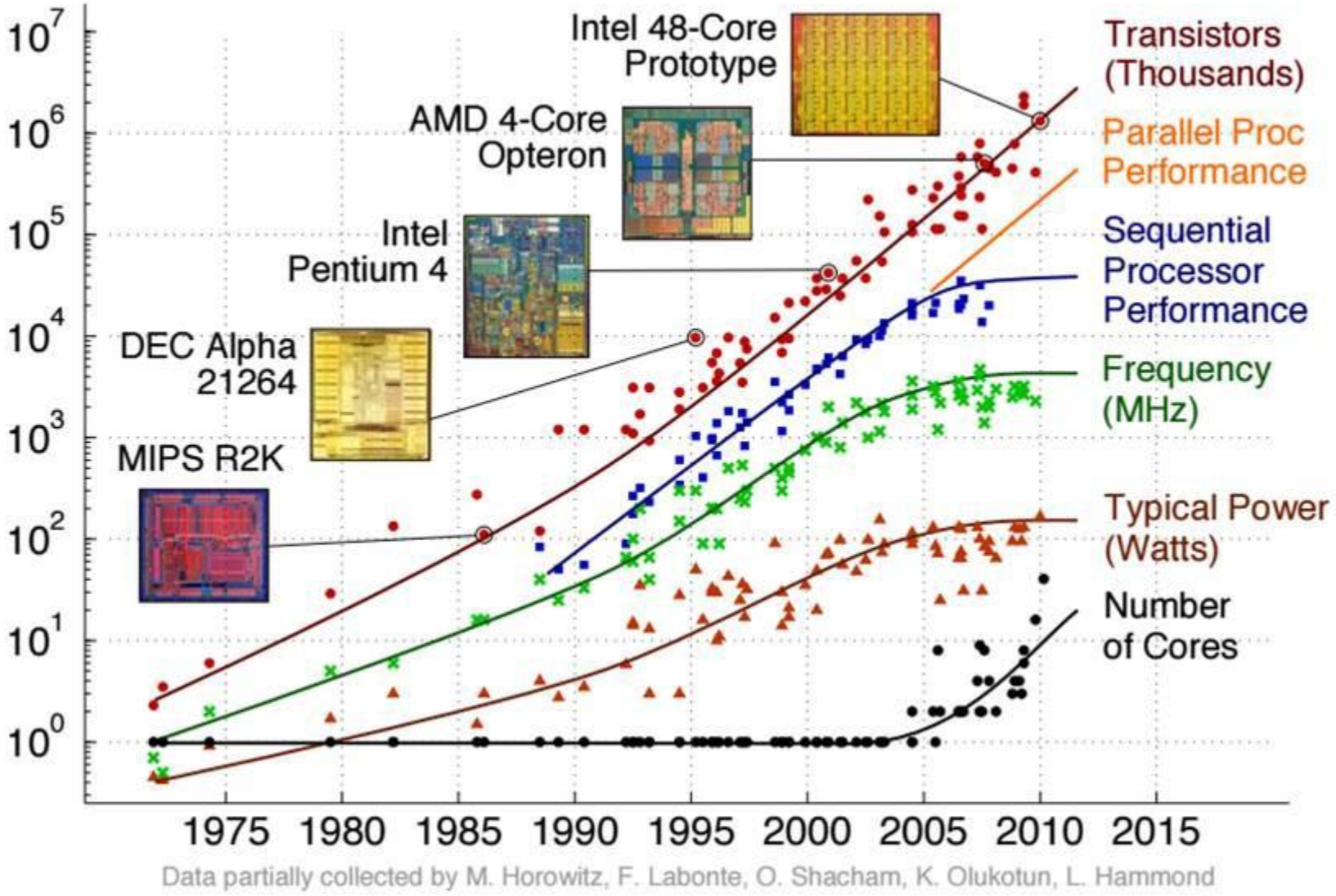
Flash back: why there is no more "free-lunch" ?





Source: Kogge and Shalf, IEEE CISE

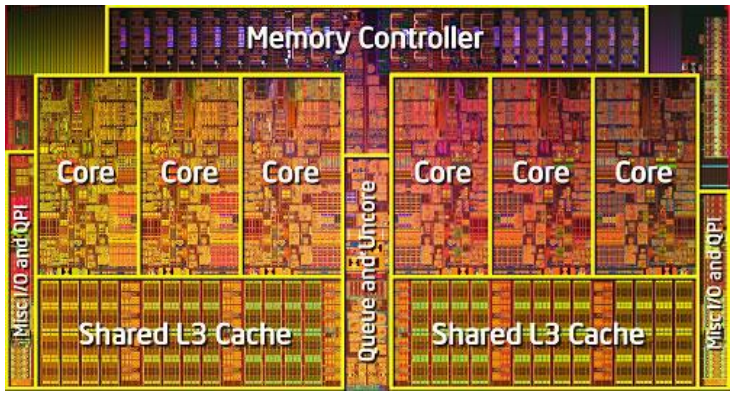
Flash back: why there is no more "free-lunch" ?



Back to the future

Message II

Many-cores CPUs are here to stay



- Concurrency-based model programming (which is different than both *parallel* and *ILP*): means work subdivision in as many independent tasks as possibile

- Specialized, heterogeneous cores

- Multiple memory hierarchies

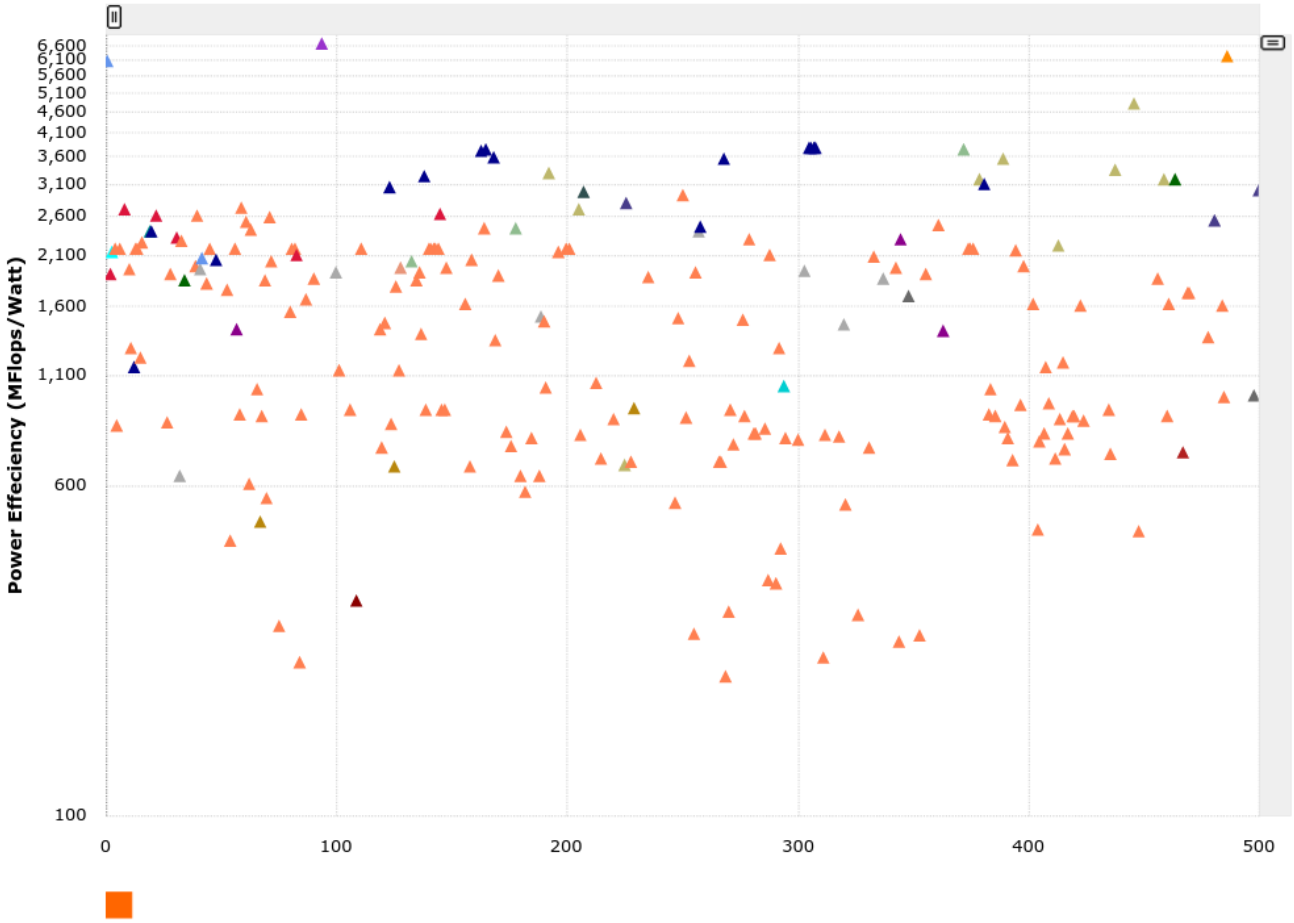
“Exa-Scale” challenges: energy consumption

Sunway performs for some apps at **≈10 Pflop/s** consuming **≈18MW**.

Simply rescaling to Eflop/s, it would consume **≈1.8GW**.

The exa-scale goal is to reach Eflop/s at **20MW** of electric power, i.e.
50 Gflops/W

Rule of thumb:
1MW = $1M / yr



"Exa-Scale" challenges: energy consumption

What dominates the energy consumption in computation ?

Operation	*pJoules*
64bits FP 28nm CMOS	12
32bits integer operations on 28nm CMOS	3
64bits FP single-issue in-order core	200
64bits multiple-issue out-of-order core	1000
reading 32bits instruction from 32KB cache	20
reading 64bits operands from DRAM	2000

“Exa-Scale” challenges

Message III

Moving memory is among most expensive operations,
x100 or more than a 64bits FP instruction.

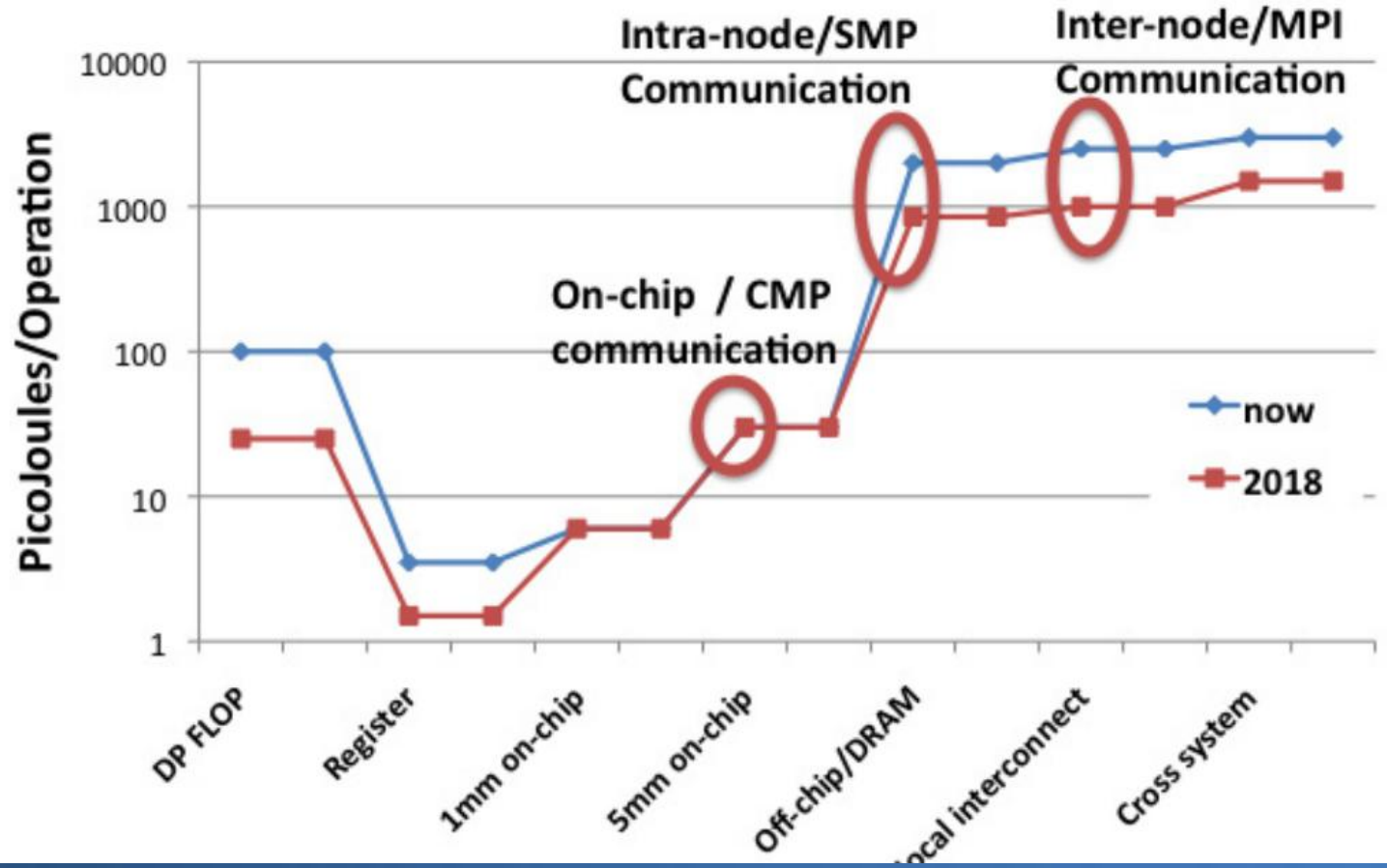
By 2018 FP will cost **10 pJ on 11nm chips**,
while reading from DRAM will still cost >**1000pJ**.

A **10 Tflop** chip will require **100W**.
It shall take **2000W** of power to supply
memory bandwidth for a modest **Bytes/FP of 0.2**

"Exa-Scale" challenges: memory capacity

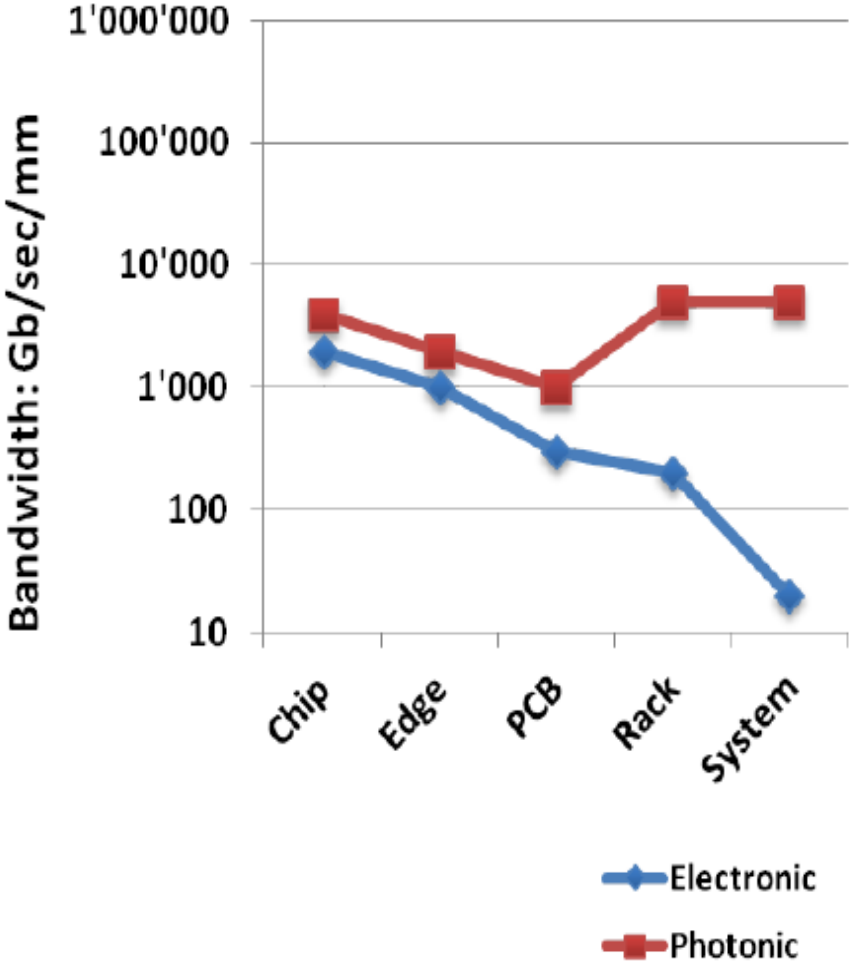
We can engineer far more floating point capability onto a chip than can reasonably be used by an application.

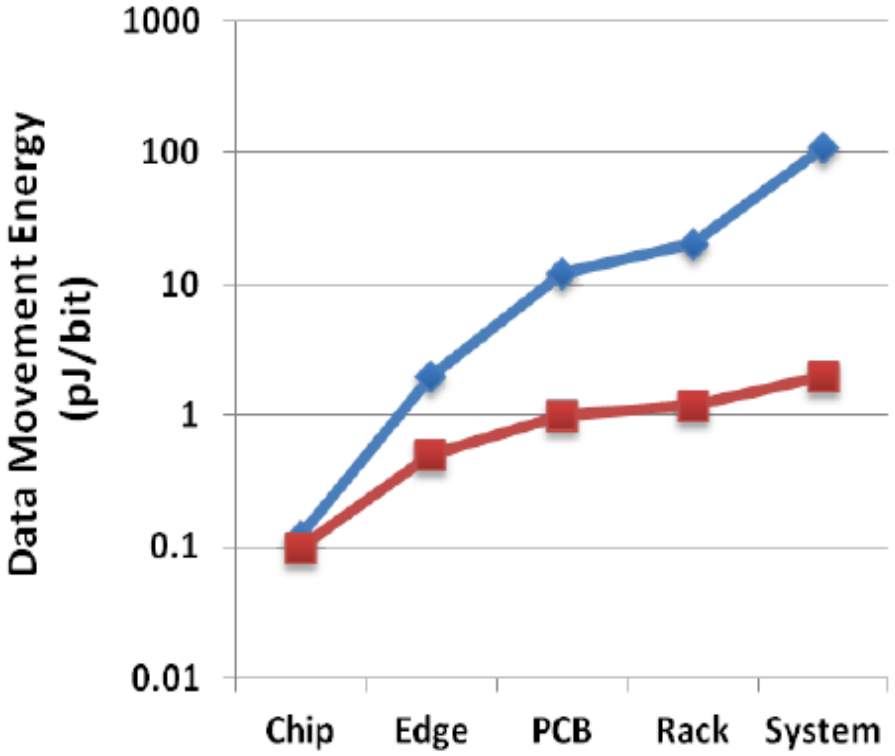
Data movement presents the most daunting engineering and computer architecture challenge.



“Exa-Scale” challenges: memory capacity

..unless more efficient memory technologies are developed





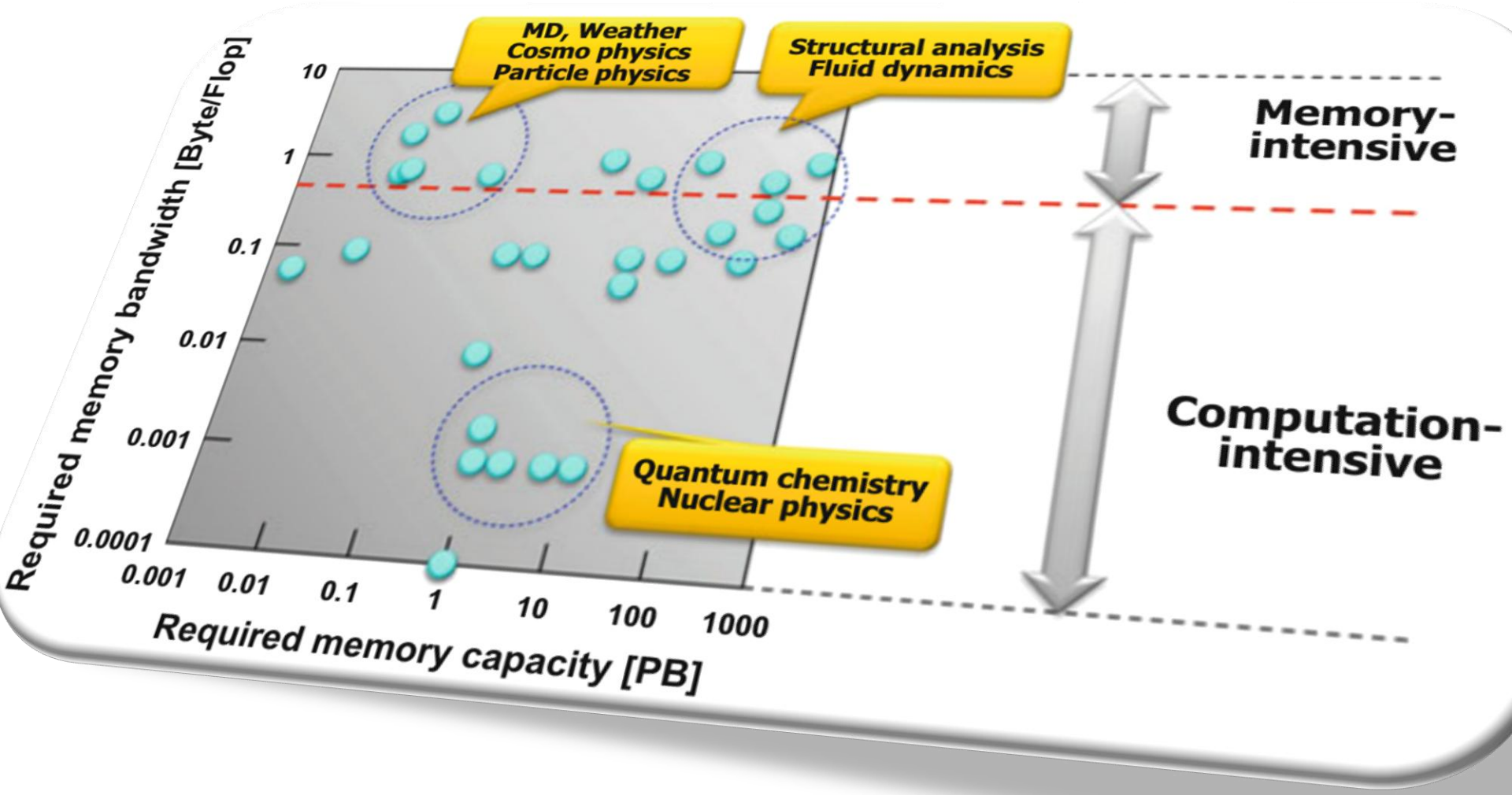
"Exa-Scale" challenges

Message IV

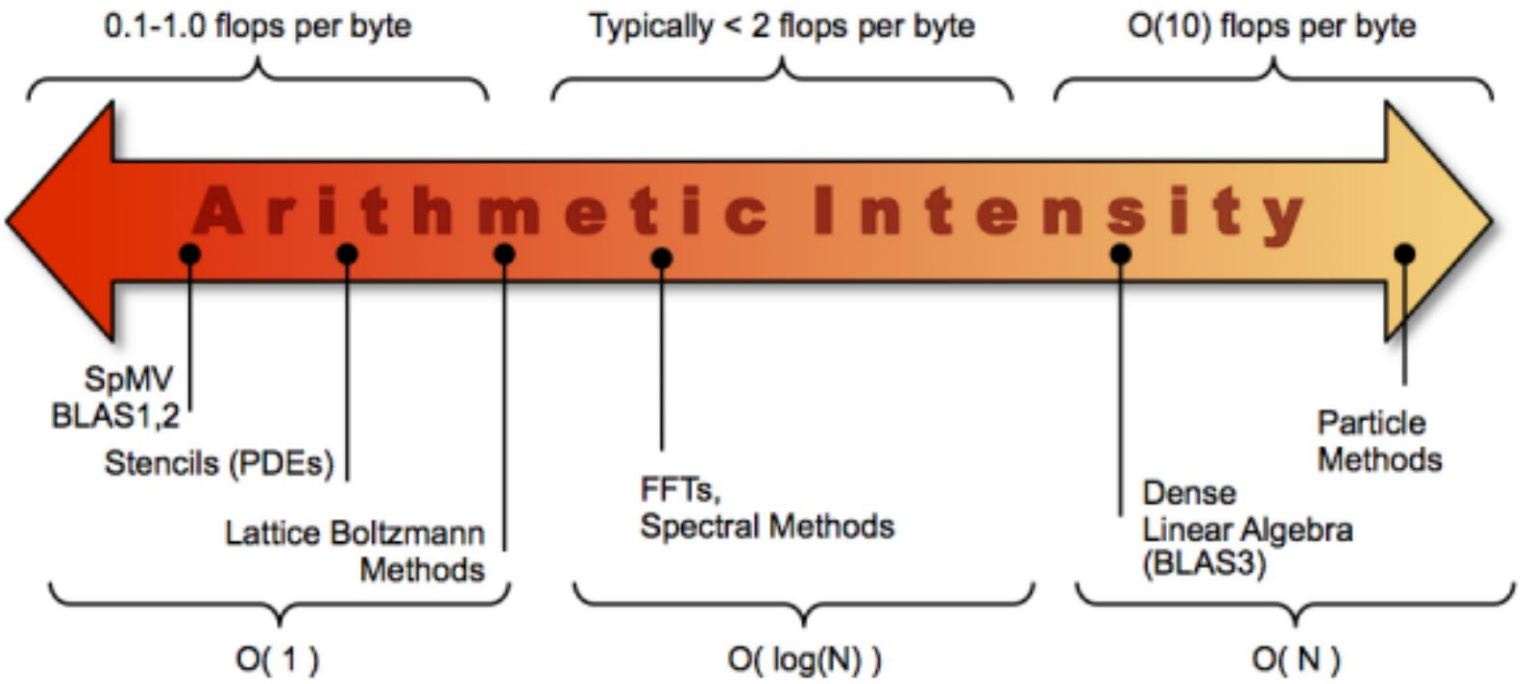
Memory Capacity is critical to applications.

- weak scaling (enlarge your problem, stay efficient)
- in-memory checkpoints
- message logging/replay for resilience
- algorithms that buy performance by using data structures that may not be minimal in their memory footprint.

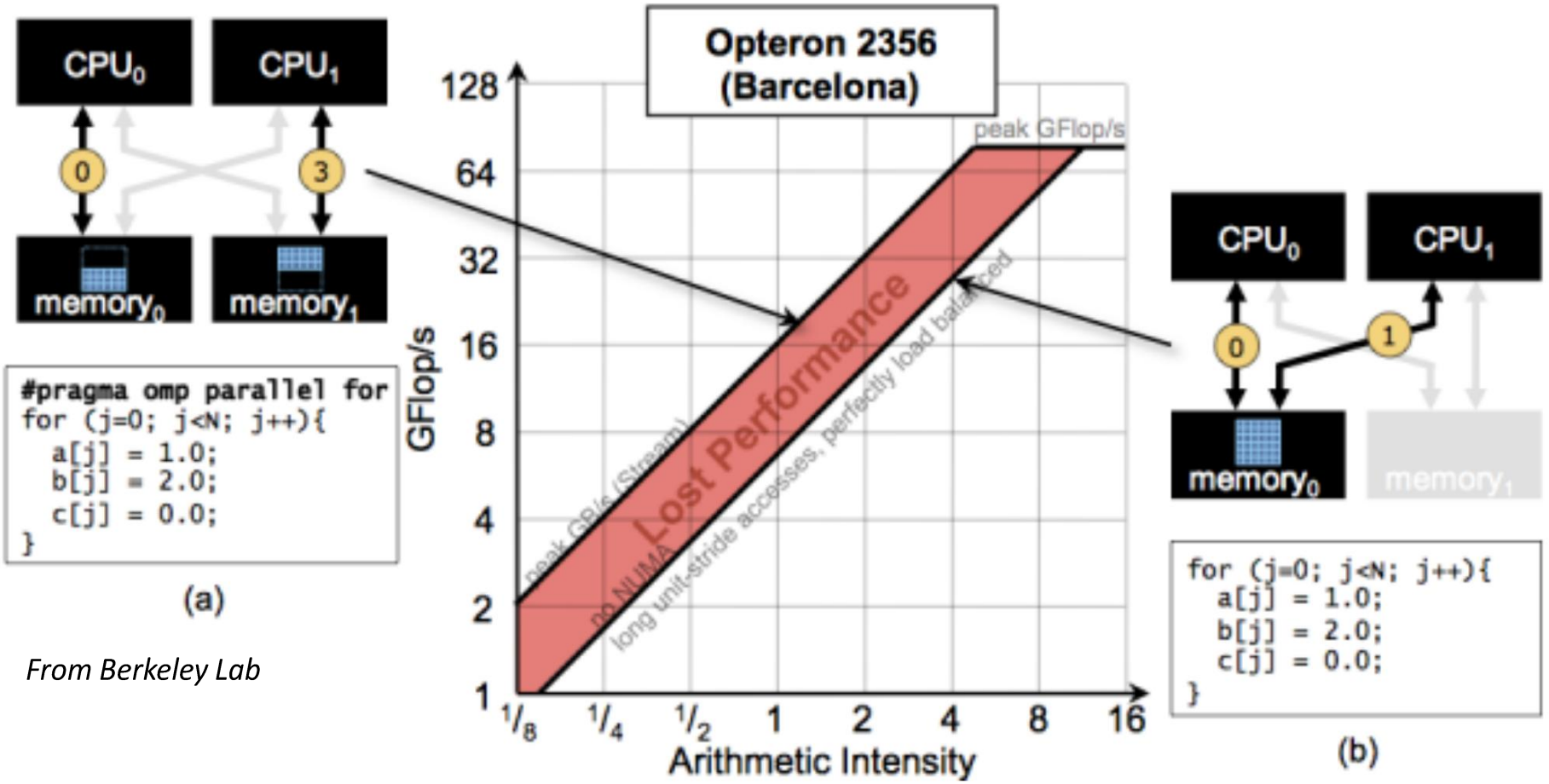
Memory- vs Computation - intensity



Memory- vs Computation - intensity



Impact of NUMA on computation efficiency



From Berkeley Lab

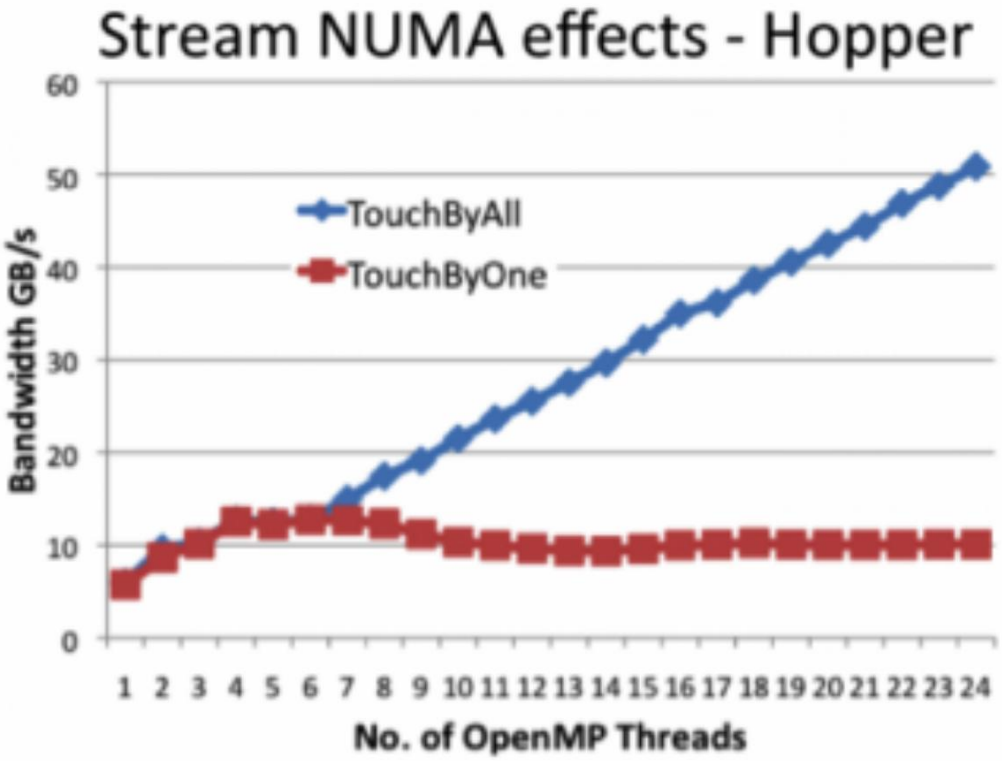
Depending on the locality of data and on where the threads are running, the memory bandwidth can change a lot.

The above example illustrates the impact of first-touch data allocation in OpenMP

Impact of NUMA on computation efficiency

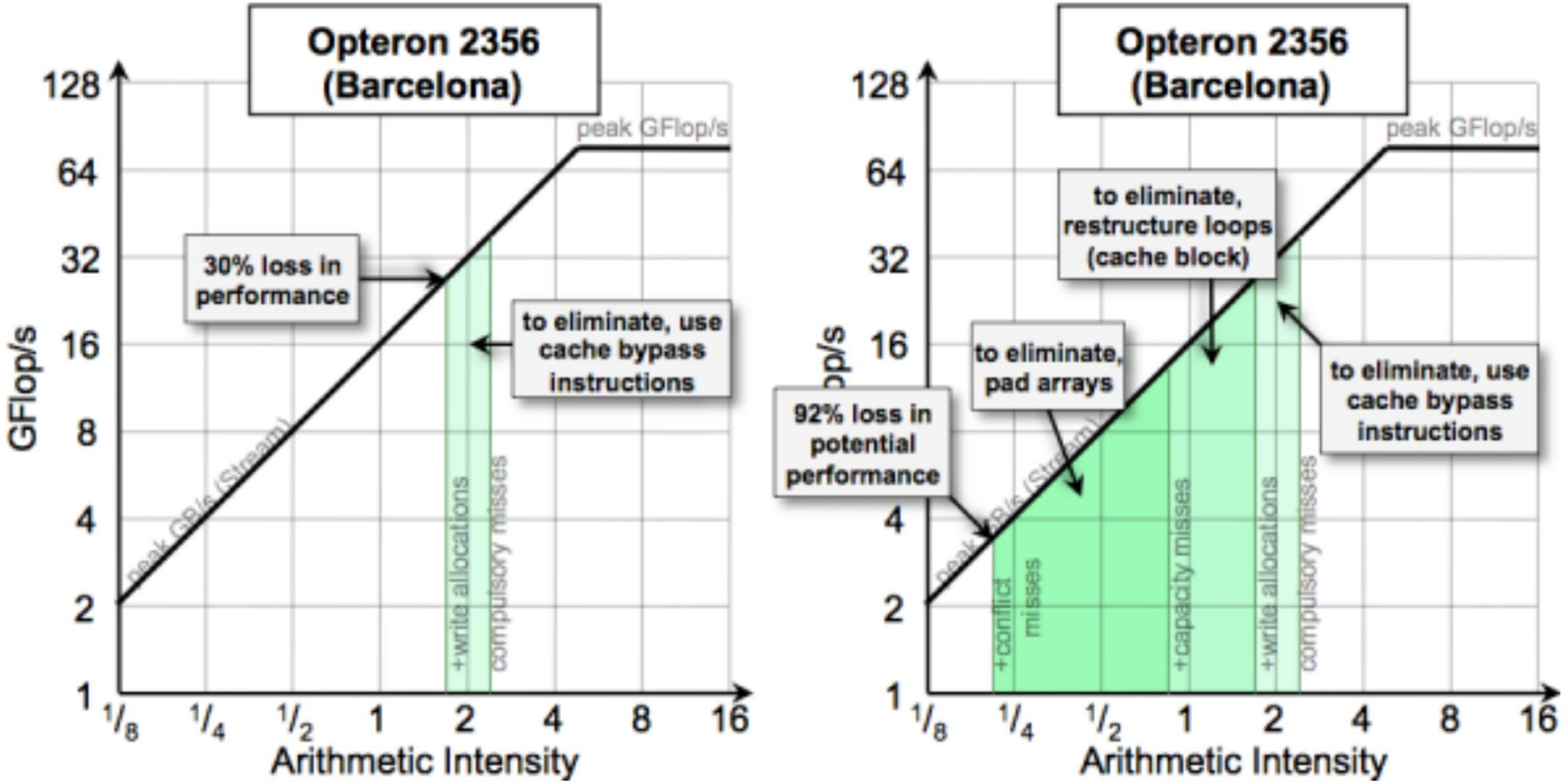
```
! Initialization
#pragma omp parallel for
 for (j=0; j<VectorSize; j++) {
 a[j] = 1.0; b[j] = 2.0; c[j] = 0.0;}

! Compute
#pragma omp parallel for
 for (j=0; j<VectorSize; j++) {
 a[j]=b[j]+d*c[j];}
```

Taken from NERSC; try on your own machine



The effect of «first-touch»

Impact of NUMA on computation efficiency

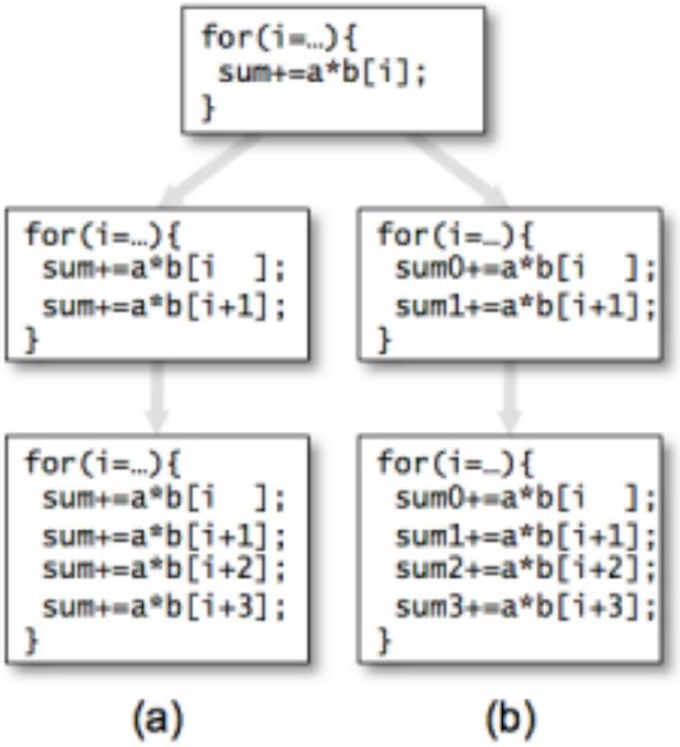


From Berkeley Lab

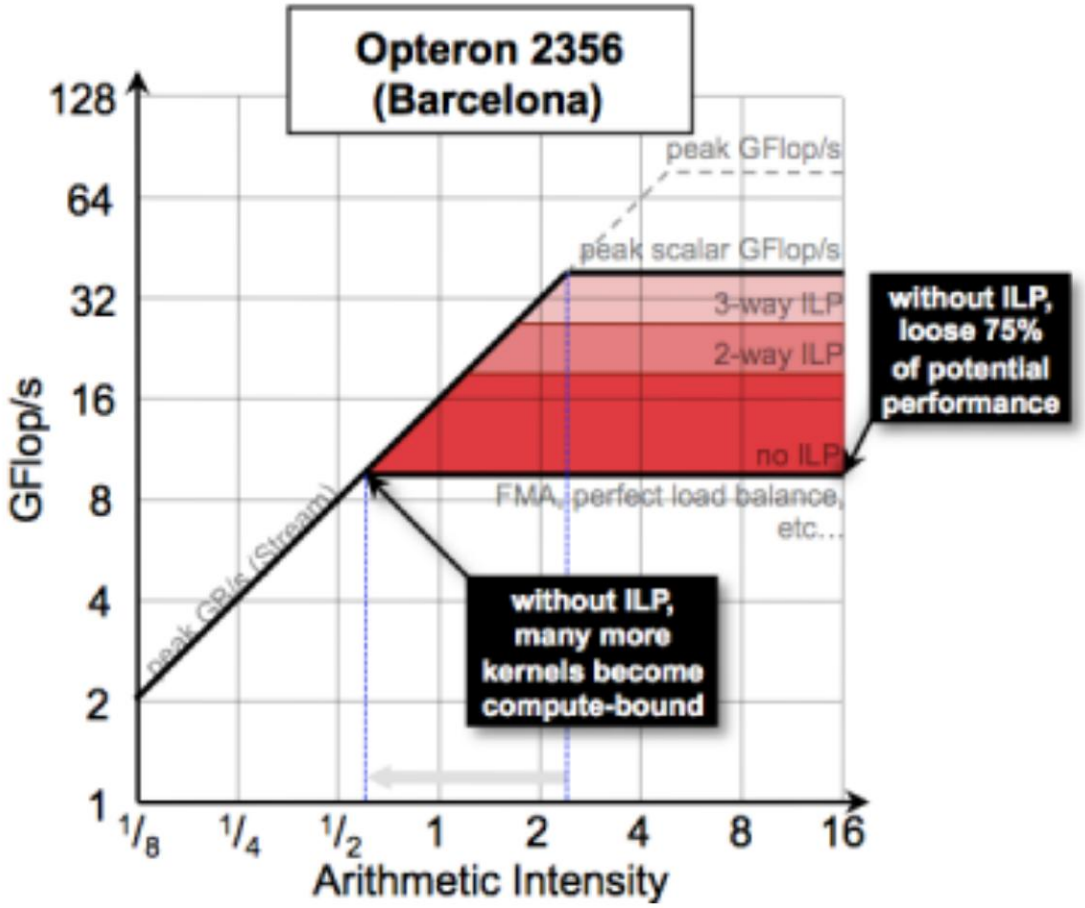
Wrong use of **cache** may result in a catastrophe.

Caches capacity and misses and useless write-allocations can result in redundant data movements

Impact of ILP on computation efficiency



From Berkeley Lab



In order that pipelines and ILP are an advantage, you have to shape your code so not to obstruct the compiler.

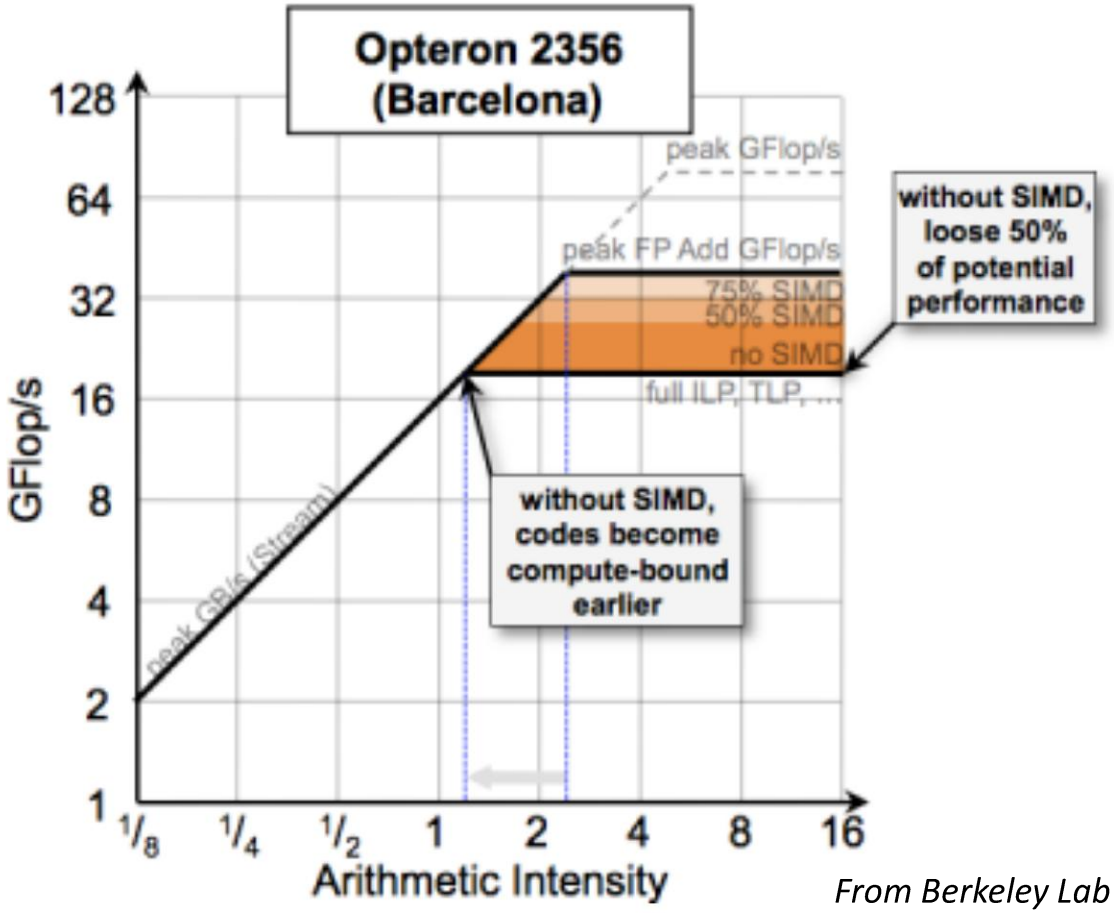
Important for compute-intensive, not that much for memory-intensive

Impact of SIMD on computation efficiency

```
for(i=…){
 sum0+=b[i  ];
 sum1+=b[i+1];
 sum2+=b[i+2];
 sum3+=b[i+3];
}
```

```
for(i=…){
 sum0=_mm_add_sd(sum0,…b[i  ]…);
 sum1=_mm_add_sd(sum1,…b[i+1]…);
 sum2=_mm_add_sd(sum2,…b[i+2]…);
 sum3=_mm_add_sd(sum3,…b[i+3]…);
}
```

```
for(i=…){
 sum01=_mm_add_pd(sum01,…b[i  ]…);
 sum23=_mm_add_pd(sum23,…b[i+2]…);
 sum45=_mm_add_pd(sum45,…b[i+4]…);
 sum67=_mm_add_pd(sum67,…b[i+6]…);
}
```



From Berkeley Lab

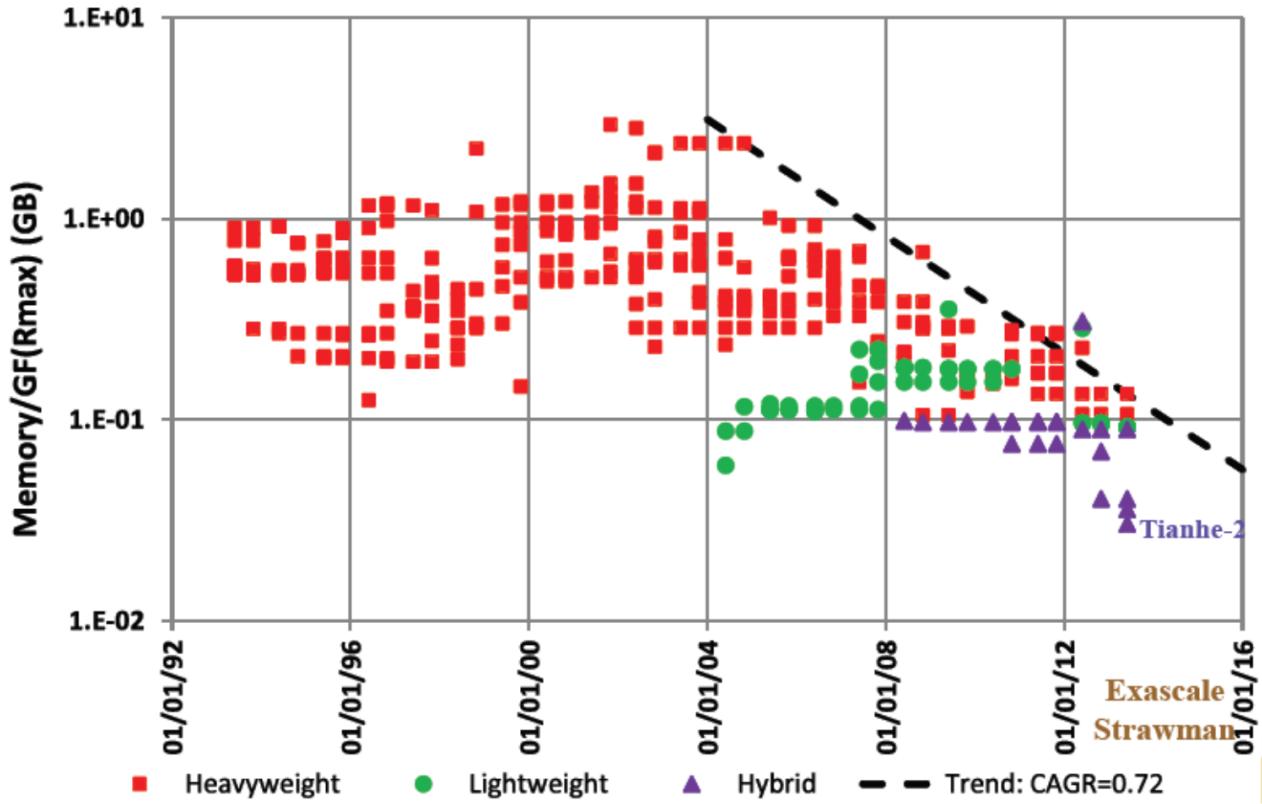
As for ILP, it may be highly dependent on you whether or not the compiler is able to attain peak performance.

Or you may choose to use intrinsics function directly.

“Exa-Scale” challenges

The machines at the top of the TOP500 **do not have sufficient memory to match historical requirements of 1B/Flop**, and the situation is getting worse.

This is a big change: it places the burden increasingly on **strong-scaling** of applications for performance, rather than on **weak-scaling** like in tera-scale era.



"Exa-Scale" challenges

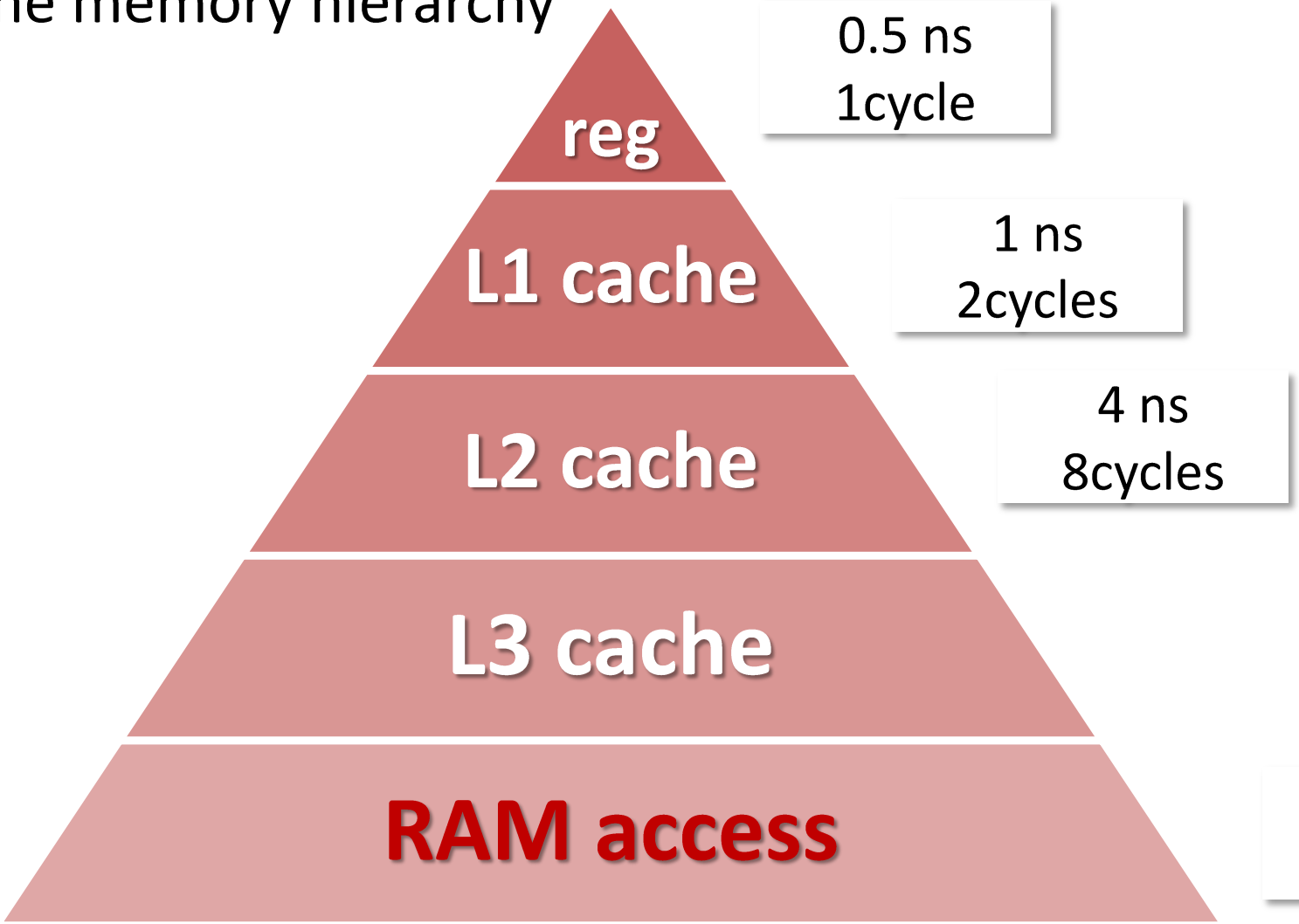
Memory power consumption $\propto$ Bw $\times$ Length^2 / Area

	AMD Radeon R9 290X	NVIDIA GeForce GTX 980 Ti	AMD Radeon R9 Fury X	Samsung's 4-Stack HBM2 based on 8 Gb DRAMs	Theoretical GDDR5X 256-bit sub-system
Total Capacity	4 GB	6 GB	4 GB	16 GB	8 GB
Bandwidth Per Pin	5 Gb/s	7 Gb/s	1 Gb/s	2 Gb/s	10 Gb/s
Number of Chips/Stacks	16	12	4	4	8
Bandwidth Per Chip/Stack	20 GB/s	28 GB/s	128 GB/s	256 GB/s	40 GB/s
Effective Bus Width	512-bit	384-bit	4096-bit	4096-bit	256-bit
Total Bandwidth	320 GB/s	336 GB/s	512 GB/s	1 TB/s	320 GB/s
Estimated DRAM Power Consumption	30W	31.5W	14.6W	n/a	20W

Feeding 1B / flop for 10^{18} flop/s ~28 MW ~60 MW

“Exa-Scale” challenges

The memory hierarchy



“Exa-Scale” challenges

1st example: 1 level of cache

Average Access Time

$$t_c \times H_c + (1 - H_c) \times (t_{ram} + t_c) = t_c + (1 - H_c) \times T_{ram}$$

99% of L1 hit

AAT = 1 + 0.01 × 100 = **2 cycles**

97% of L1 hit

AAT = 1 + 0.03 × 100 = **4 cycles**

2nd example: 2 levels of cache

Average Access Time

$$t_{c1} \times H_{c1} + (2 - H_{c1} - H_{c2}) \times t_{c2} + (1 - H_{c2}) \times T_{ram}$$

99% of L1 hit, 99% of L2 hit

AAT = 1 + 0.02 × 2 + 0.01 × 100 = **2.04 cycles**

97% of L1 hit, 99% of L2 hit

AAT = 1 + 0.04 × 2 + 0.01 × 100 = **2.08 cycles**

97% of L1 hit, 97% of L2 hit

AAT = 1 + 0.06 × 2 + 0.03 × 100 = **4.12 cycles**

"Exa-Scale" challenges

3rd example: CPU @ 2GHz, base CPI = 1

Miss rate/instruction = 2%

RAM access time = 100ns

1 level of cache

Miss penalty = 100ns/0.5ns = 200 cycles

Effective CPI = 1 + 0.02 × 200 = **5**

2 levels of cache

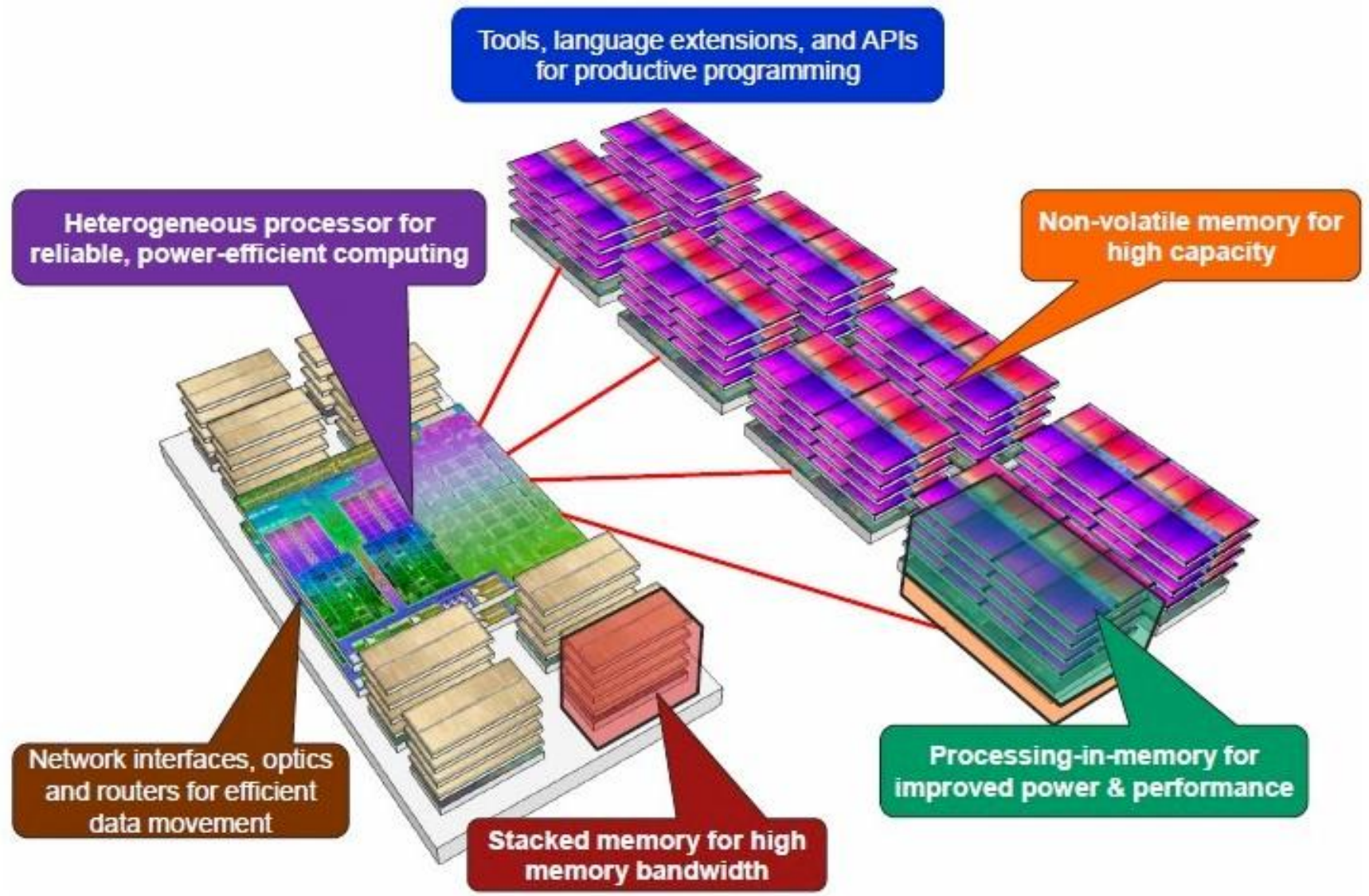
L2 access time = 5ns ; Miss rate = 1%

L1 Miss penalty = 5ns/0.5ns = 10 cycles

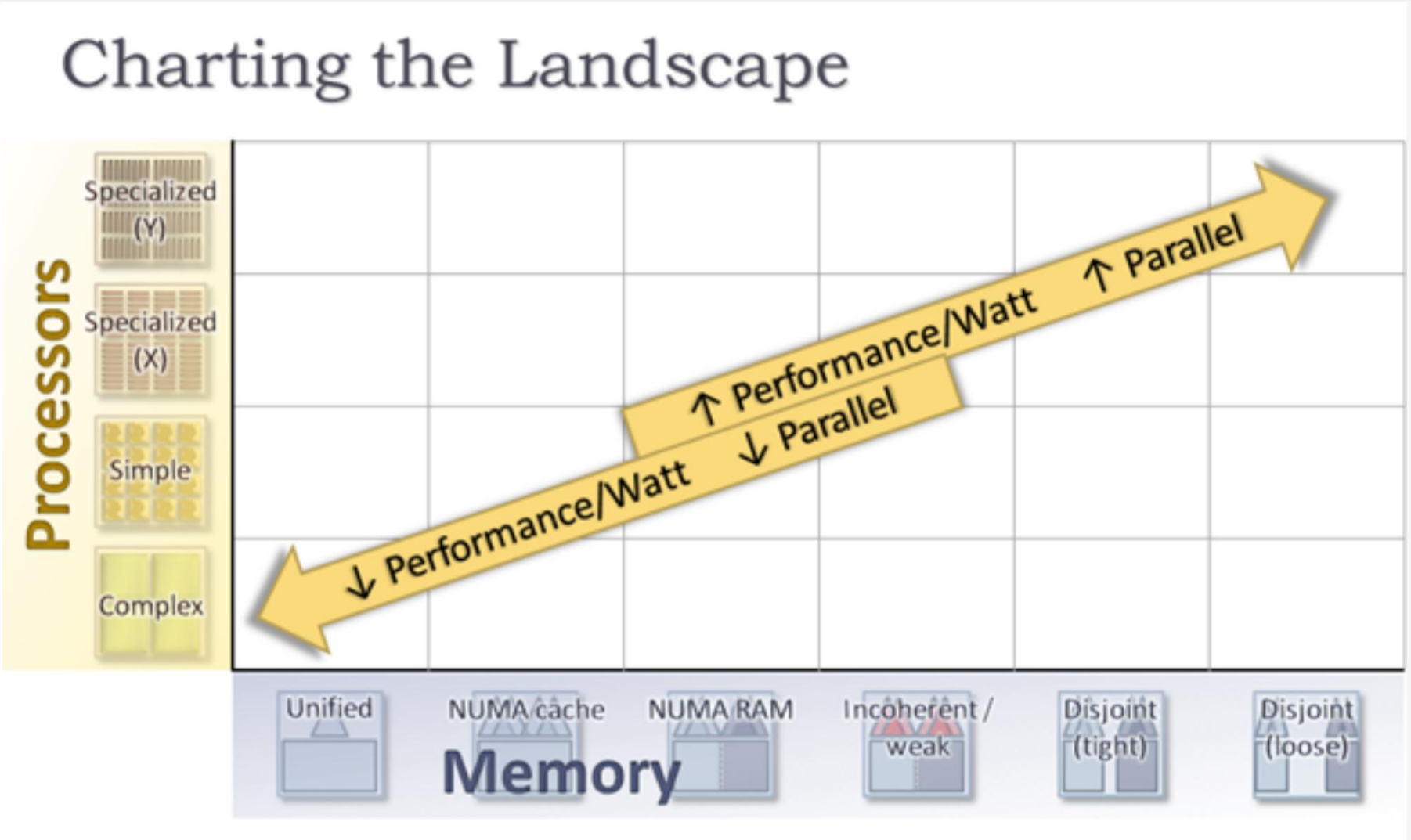
L2 Miss penalty = 200cycles

Effective CPI = 1 + 0.02 × 10 + 0.01 × 200 = **3.2**

Some Exa-Scale facts : architectural view



Some Exa-Scale facts : computation/data view



By far, no more a Von Neumann machine…

Some Exa-Scale facts

POWER WALL

- Cores need to be as simple as possible
- **Code optimization** becomes fundamental
- Concurrency programming → **software design**
- Cores' **specialization** must be exploited
- **Billion-way parallelism**

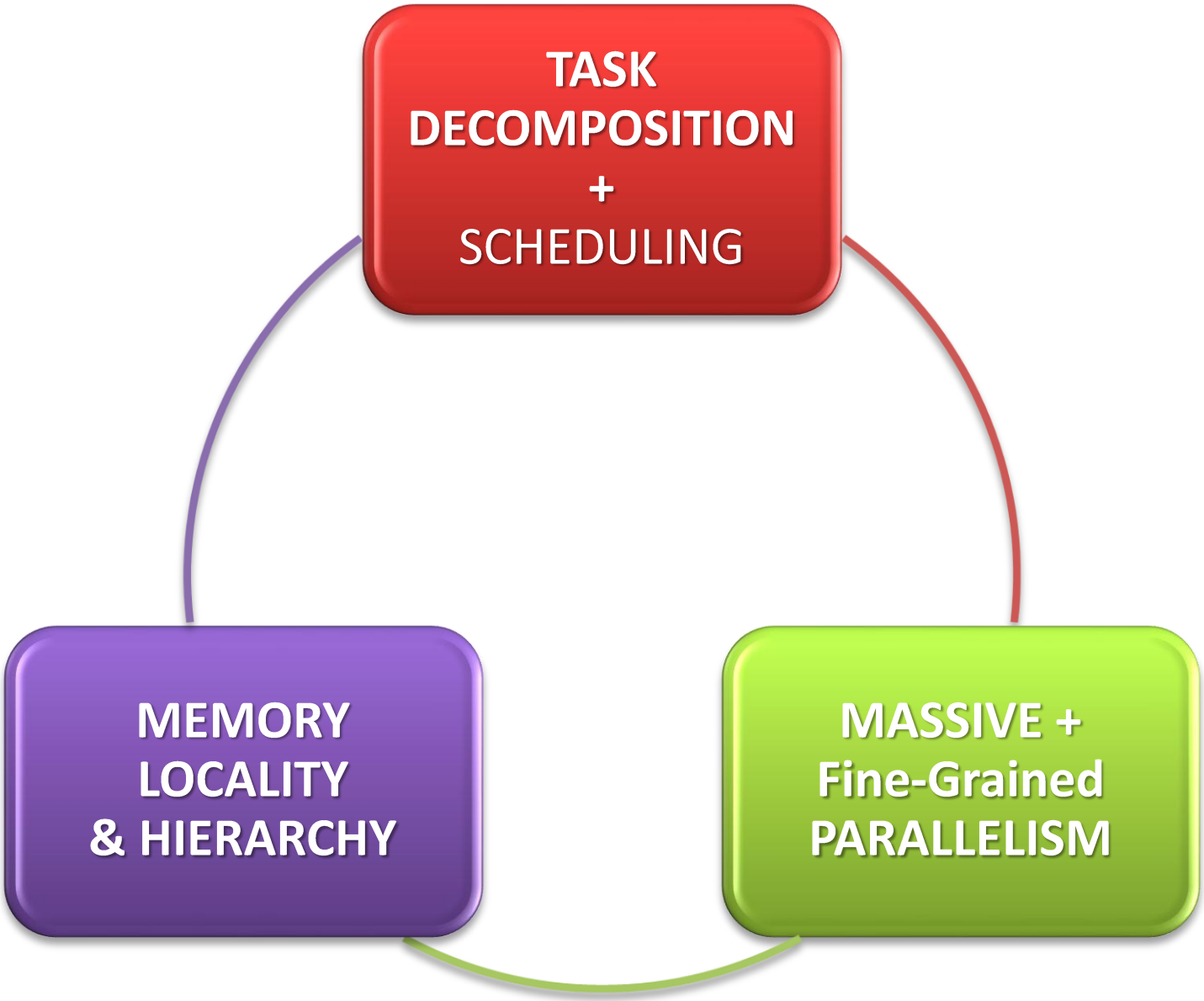
MEMORY WALL

- Memory power wall + spatial constraints + cost constraints → very small memory with high bw
- **Extreme multi-level NUMA hierarchy**: L1 -> L2 -> L3 -> local RAM (shared) -> non-local RAM -> distant RAM
- Possible PGAS paradigm
- **Data locality by design is mandatory**

ILP WALL

- Concurrency programming + careful threadization, do not rely on automatic pipelining

Some consequences for developers



Some consequences for developers