FRASCAL HPC Day

General Introduction

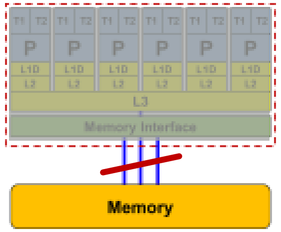
https://go-nhr.de/FRASCAL23

Agenda

- General intro, Q&A
  - Computer architecture intro, bottlenecks (all)
  - Performance vs. scalability, scaling laws (GHa)
  - NHR@FAU clusters + file systems (MW)
- Performance assessment with tools
  - Typical performance patterns (GHa)
  - ClusterCockpit job monitoring (MW)
  - likwid-perfctr (TG)
    - Demo: analyzing a preconditioned CG solver
- Introduction to the Intel Trace Analyzer and Collector (GHa)
  - Demo: analyzing a simple ray tracer code
- Hints and strategies for code performance and scalability optimization (GHa)

Quiz

- What is “memory bandwidth”?
  Rate of data transfer between main memory (RAM) and CPU chip.
  Typical CPU $b_S \approx 30 \dots 300\ \mathrm{GB/s}$, GPU $b_S \approx 0.8\ \dots\ 2.5\ \mathrm{TB/s}$



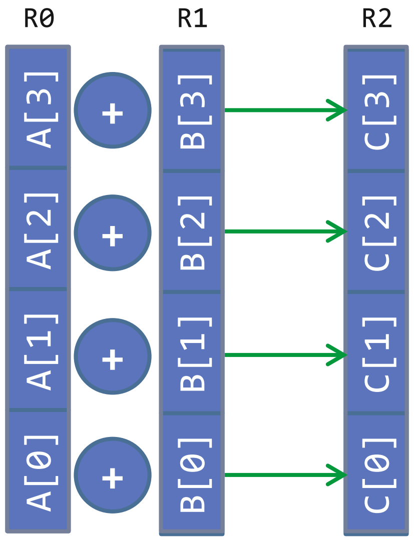
- What is “pipelining” in computing?
  An instruction execution unit on the core that executes a certain task in several simple sub-steps. The stages of the pipeline can act in parallel on several instructions at once.

- What is “superscalarity”?
  Multiple instructions can be finished in parallel each cycle.

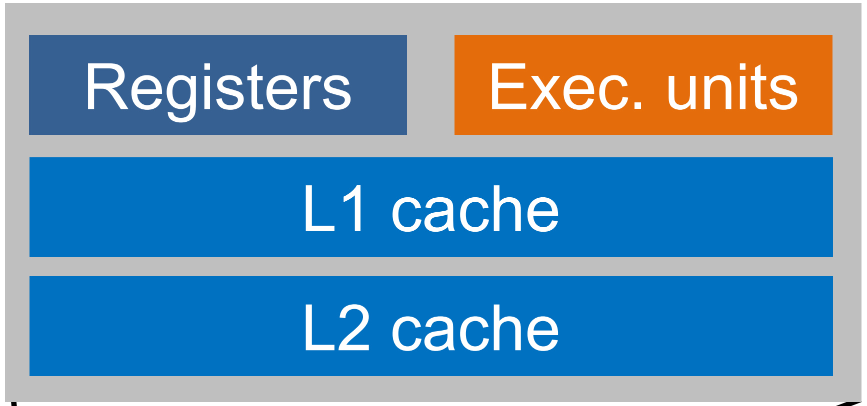
Quiz

- What is a register?
  A storage unit in the CPU core that can take one single value (a few values in case of SIMD). Operands for computations reside in registers.

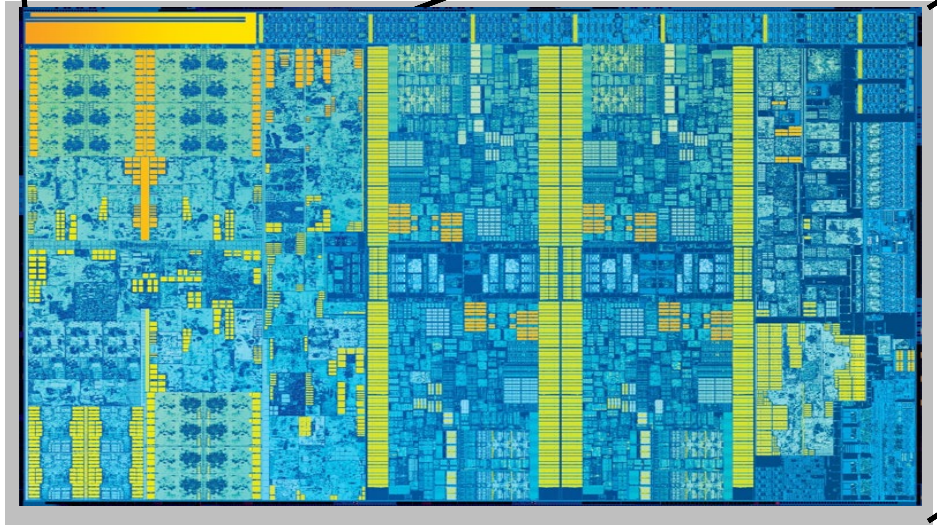
- What is “SIMD”?
  Single **I**nstruction **M**ultiple **D**ata.
  Data-parallel load/store and execution units.



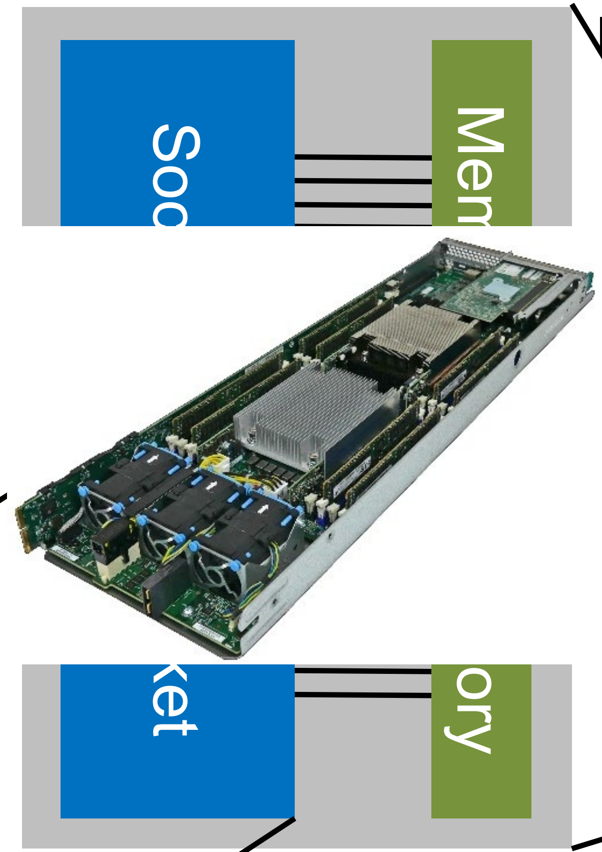
Anatomy of a (CPU) compute cluster



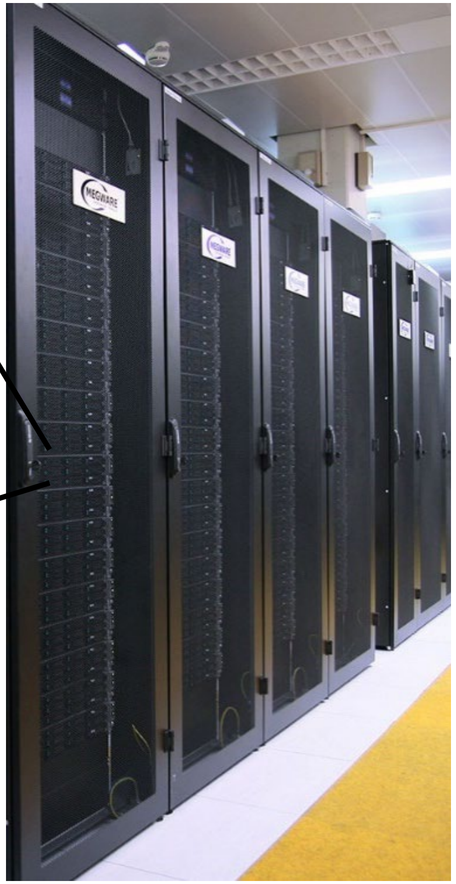
Core



Chip (up to 64 Cores)

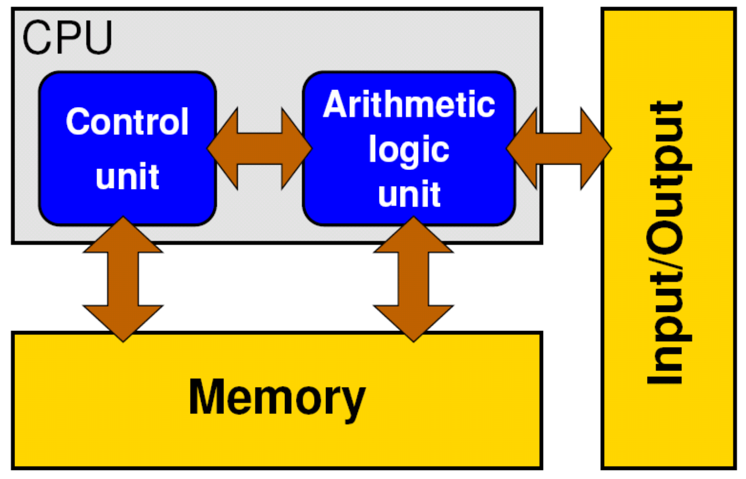


Node (2 sockets + memory + I/O, possibly multiple chips per socket)

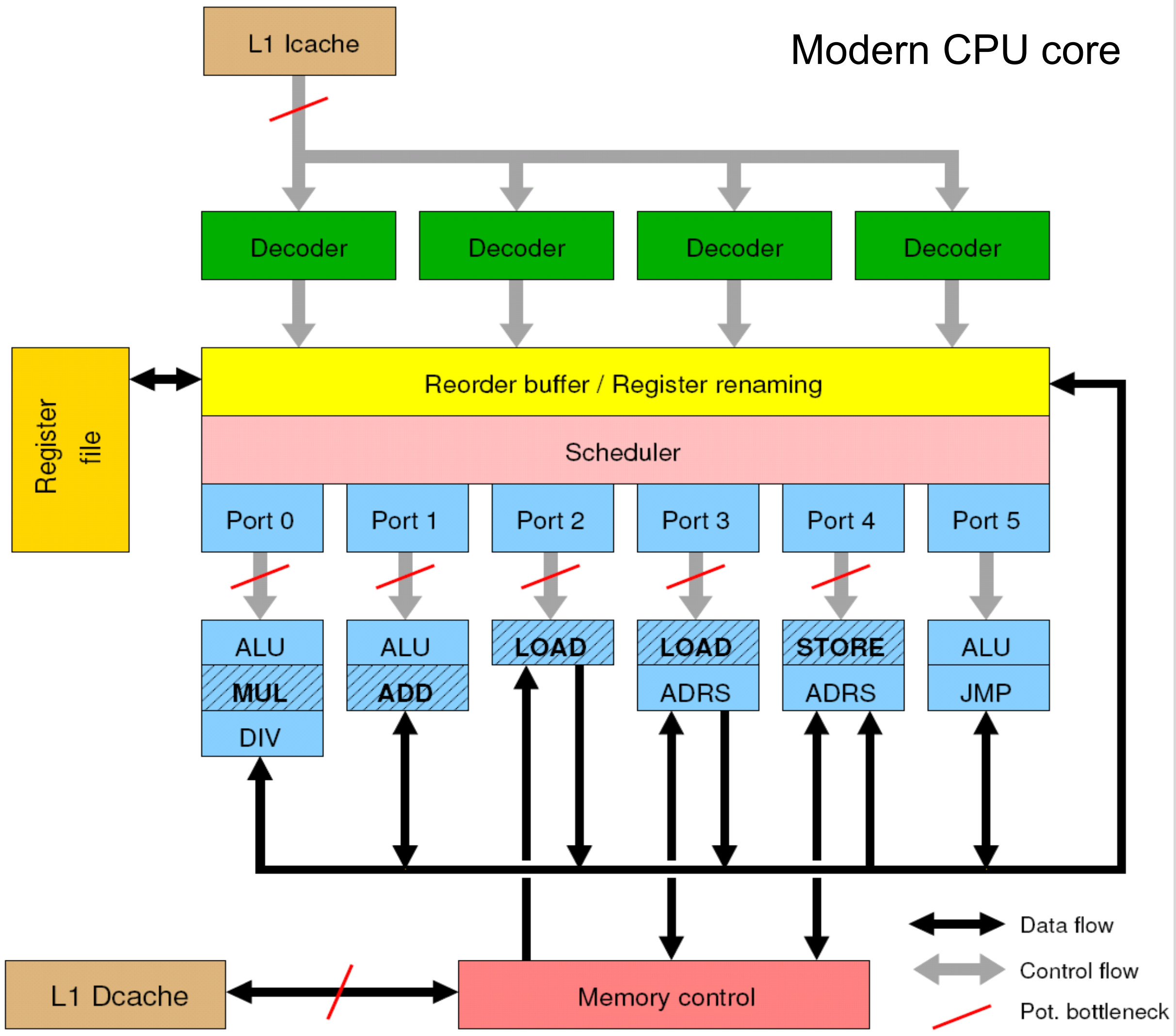


Supercomputer (many nodes, high-performance network, storage)

General-purpose cache based microprocessor core



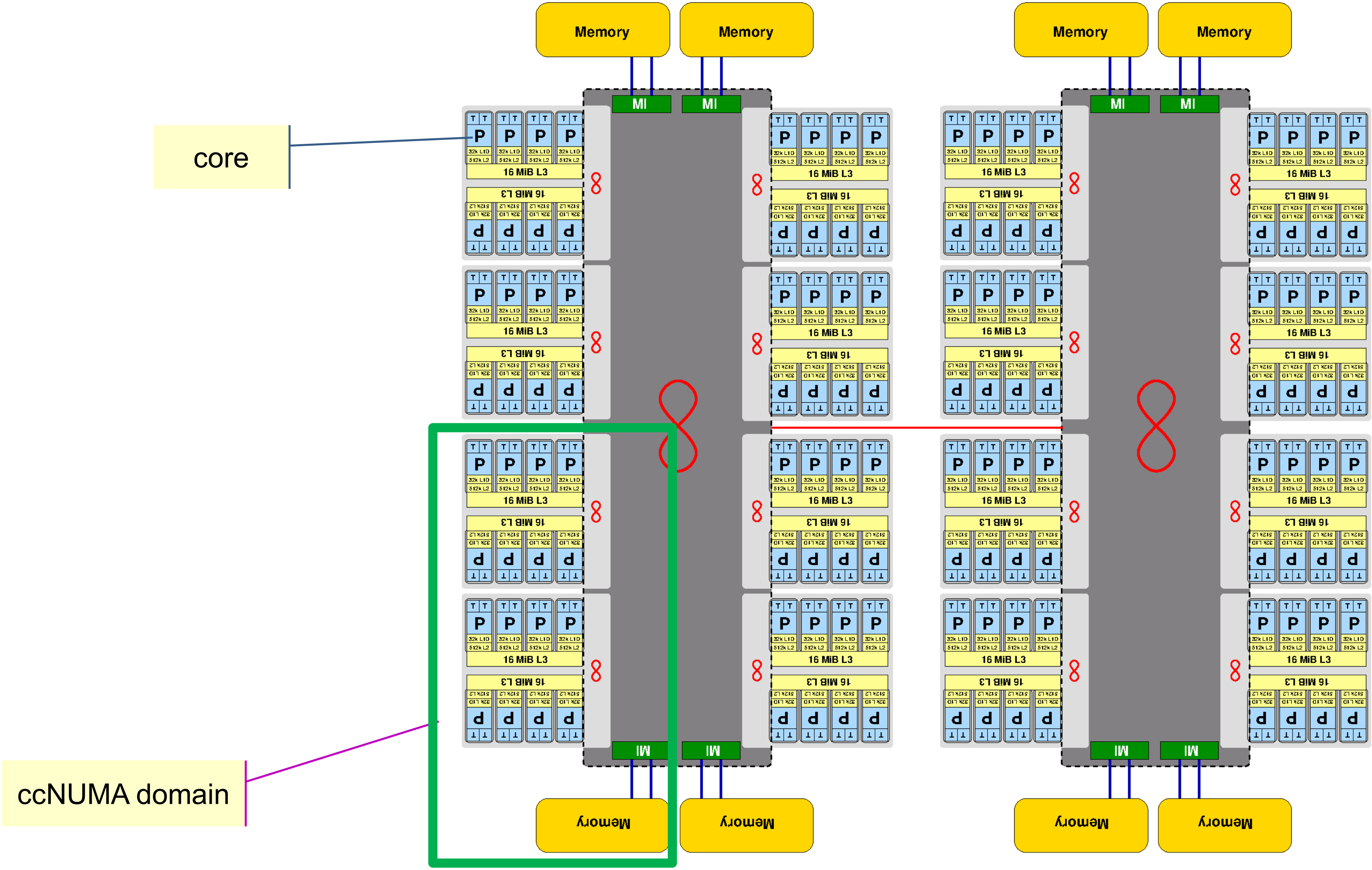
Stored-program computer



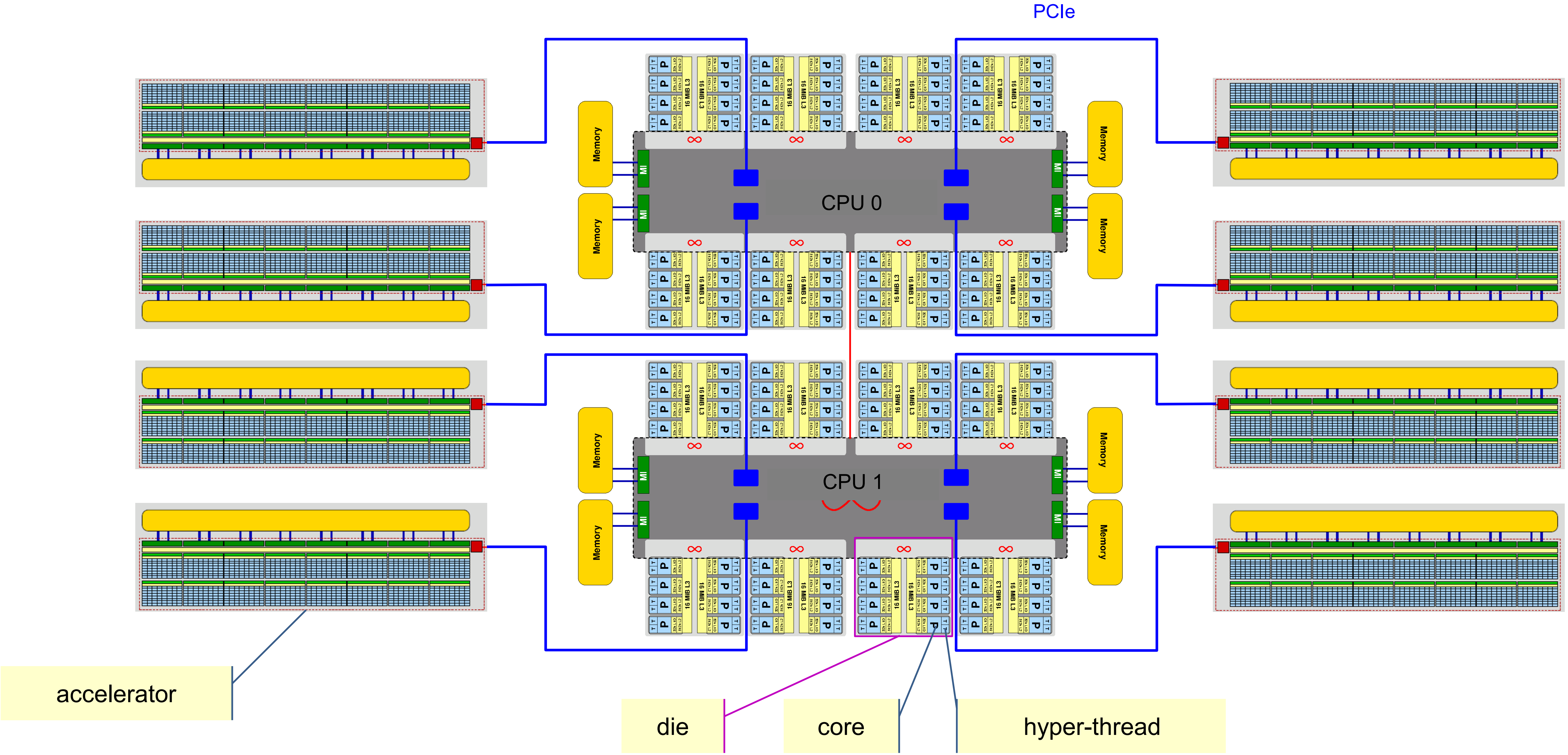
- Implements "Stored Program Computer" concept
- Similar designs on all modern systems
- (Still) multiple potential bottlenecks

The clock cycle is the "heartbeat" of the core

A modern CPU compute node (AMD Zen2 “Rome”)



Adding accelerators to the node



Quiz

- What is “network latency”?

  The time it takes to set up a data transfer over a network connection. Typically 1-3 μs (InfiniBand) or a few 100 ns (intra-node)

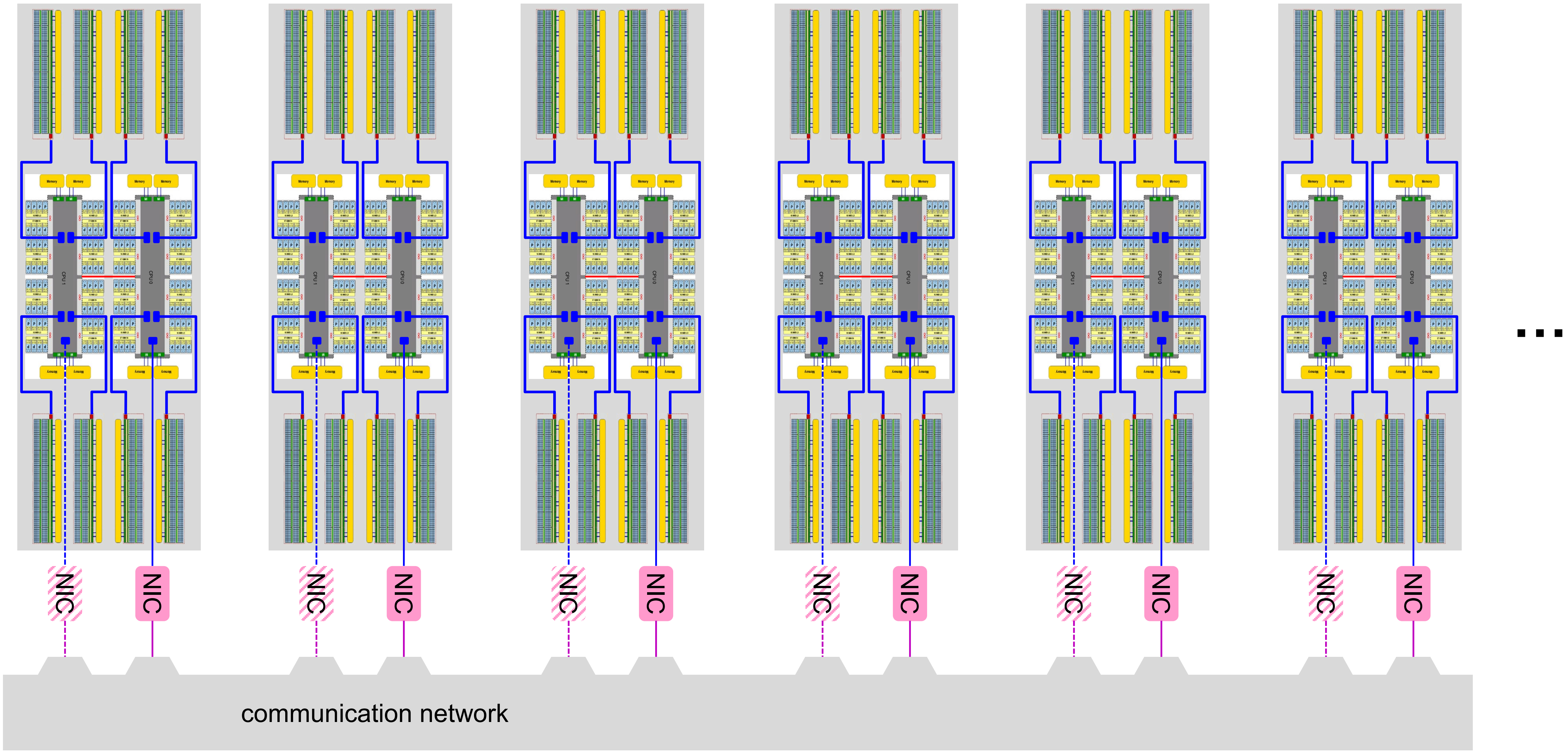
  Transfer time for package of size V: $T = \lambda + \frac{V}{B}$, where λ is the latency and B is the bandwidth of the connection

- What does the following code do?:

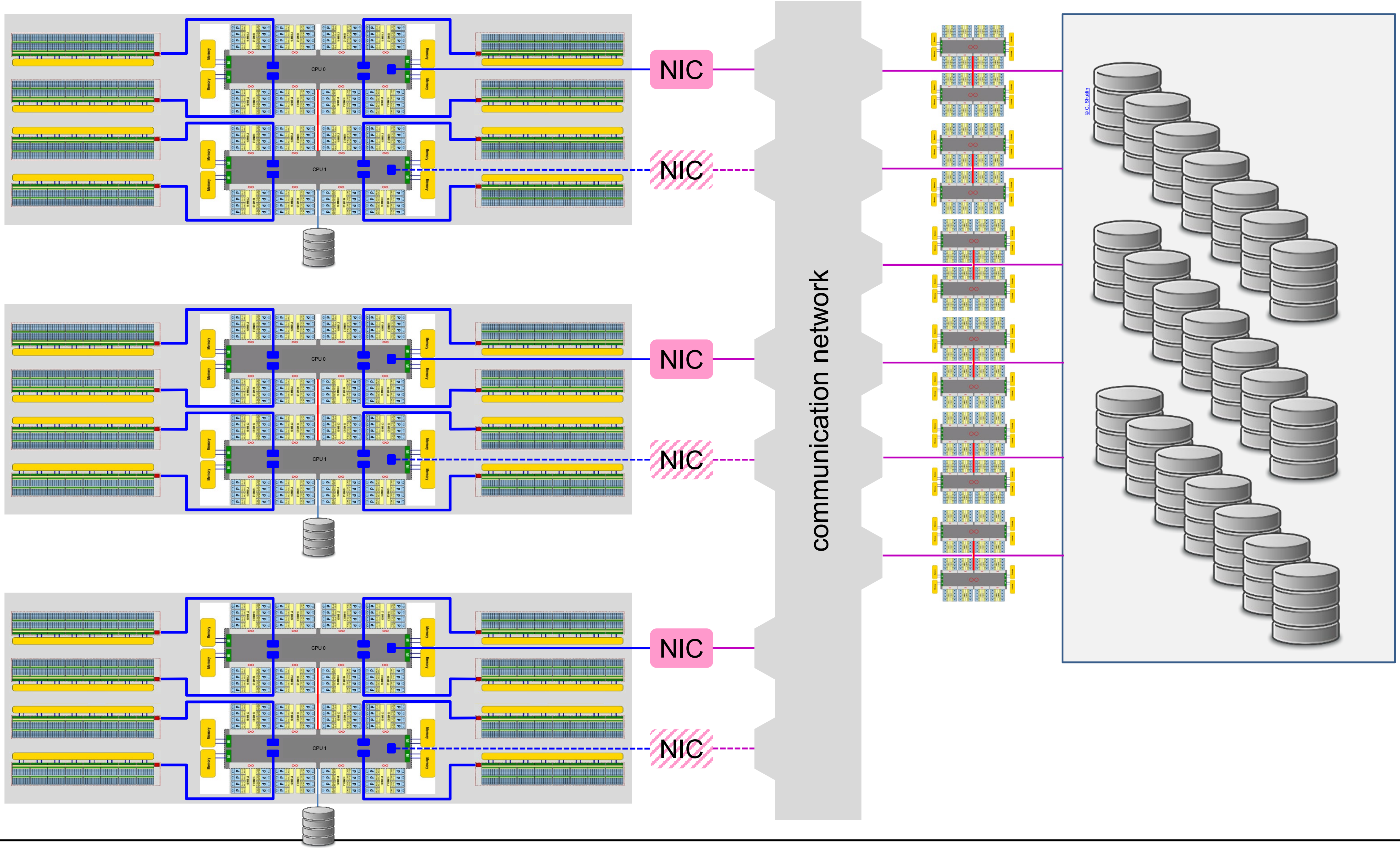
```
MPI_Isend(&buf, …, &request);
do_some_work();
MPI_Wait(&request,…);
```

It looks like work and communication will overlap, but in practice this depends on many factors

Turning it into a cluster



Adding permanent storage

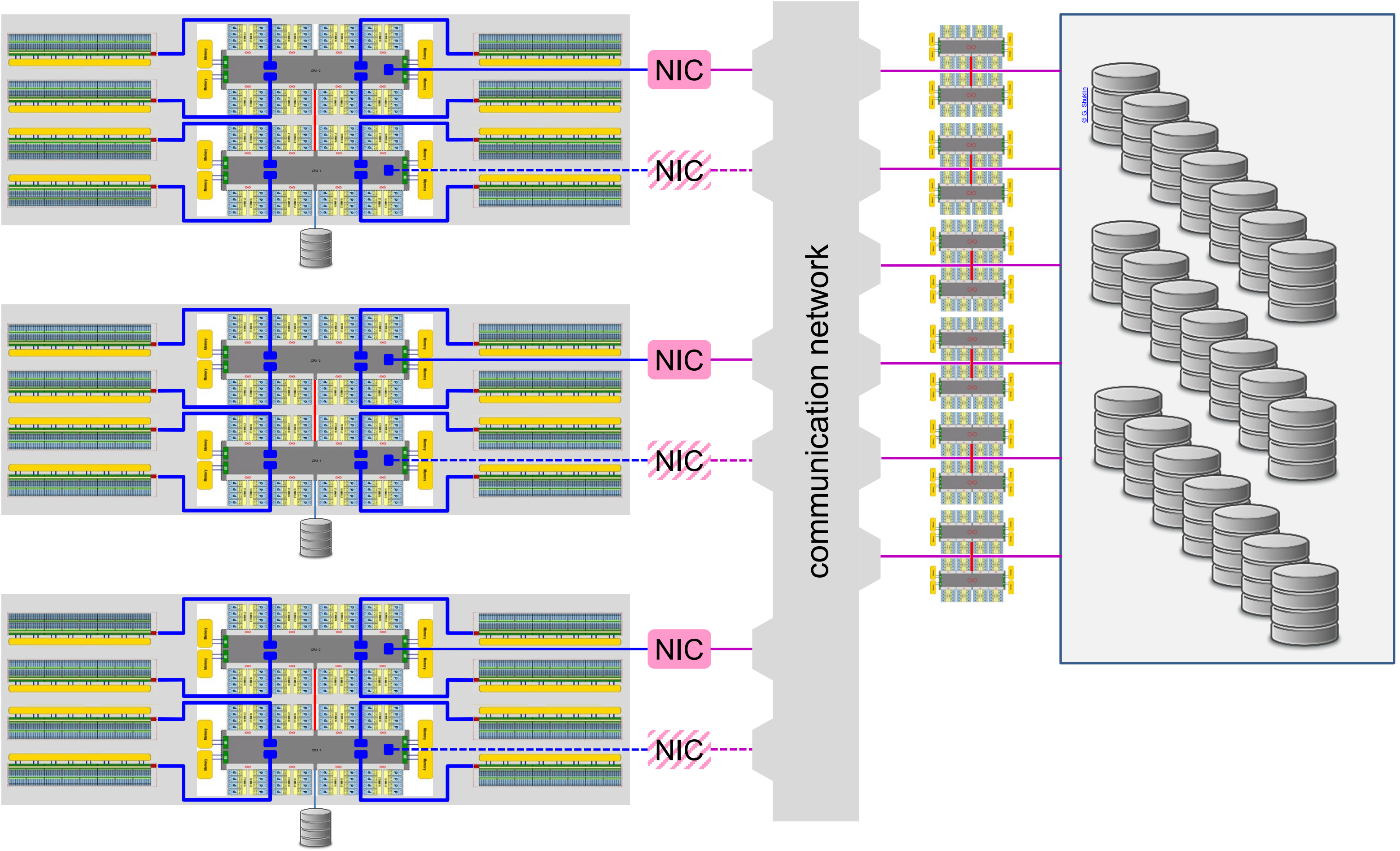


The crucial questions

Questions

- What are the hardware components that limit the performance of my code?
- What software properties limit the performance of my code?

- How should I know?
- What can I do about it?

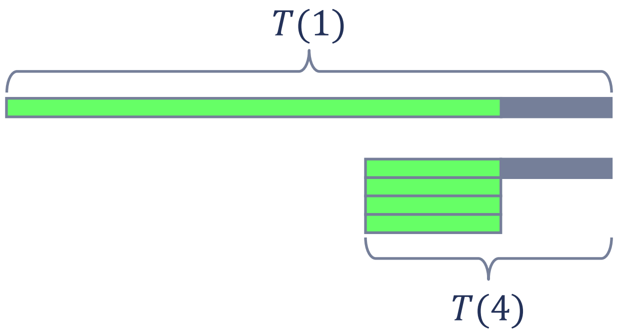


Quiz

- What is "strong scaling" vs. "weak scaling"?
  Strong scaling: more resources (compute units), same problem size
  Weak scaling: problem size scales with resources

- What is "Amdahl's Law"?

$$S_p = \frac{T(1)}{T(N)} = \frac{1}{s + \frac{1-s}{N}}$$



- "*My code shows a speedup of 1000x on 1024 CPUs, so it's really efficient*." Any thoughts?
  Speedup and performance are different metrics. The code could scale perfectly but still make inefficient use of hardware resources (compute units, memory bandwidth)

What is “performance”?

Performance metric:

$$P = \frac{\text{Work}}{\text{Time}}$$

Work:

- # of **flops** (+ - * /)
- # of lattice site updates
- # of images processed
- ns of simulated time
- # of iterations
- “Solving the problem”...

Time:

“Wall-clock time”

Parallel performance

Performance is generated by parallelism!

$$P = P_{core} \times (\#\ \text{cores})$$

$$P_{socket} \times (\#\ \text{sockets})$$

$$P_{GPU} \times (\#\ \text{GPUs})$$

$$P_{node} \times (\#\ \text{nodes})$$

$$P_{sub-cluster} \times (\#\ \text{sub}-\text{clusters})$$

"scaling baselines" → P_{core}, P_{socket}, P_{GPU}, P_{node}, $P_{sub-cluster}$

Speedup

"How much faster can I compute with n times as much resources?"

cores

GPUs

nodes

sockets

...

$$S(n) = \frac{P(n)}{P(1)}$$

Best case (sort of): $S(n) = n$

Usual case: $S(n) < n$

Worst-case scenario: $S(n) < 1$

Parallel efficiency:

$$\varepsilon(n) = \frac{S(n)}{n}$$

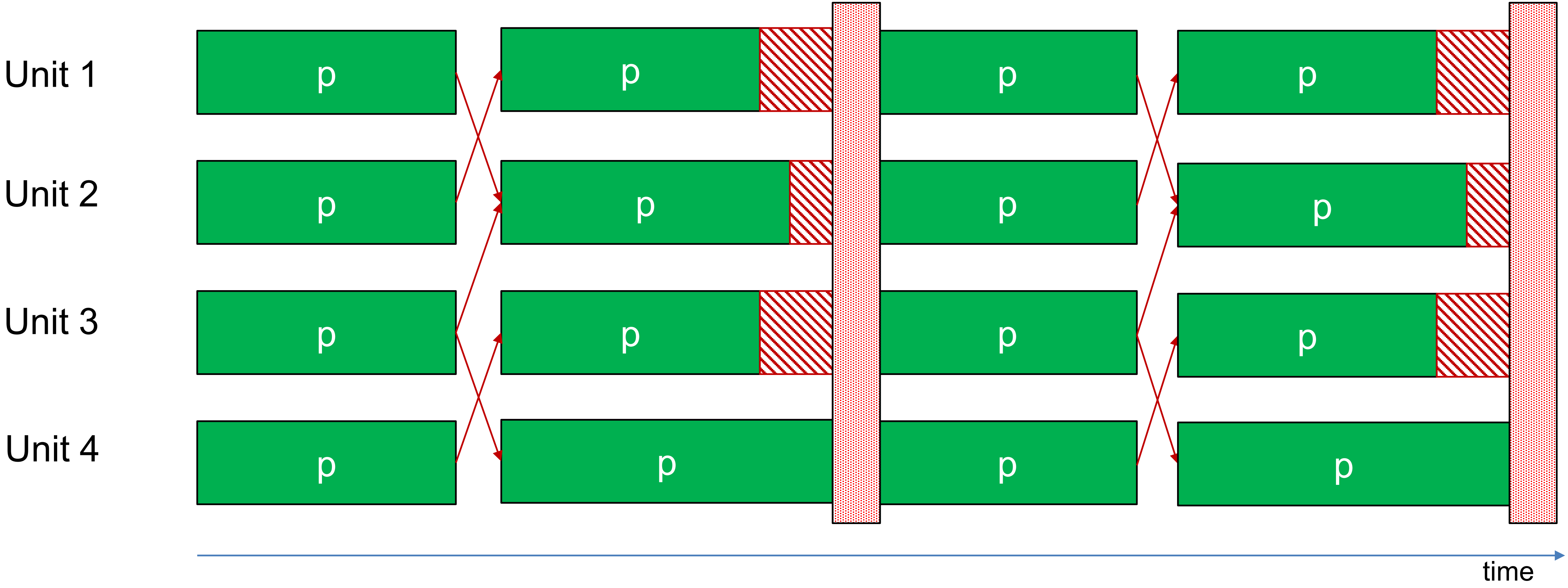
Quiz

- What basic roadblocks exist for scaling?
  - Structural impediments
    - Load imbalance
    - Communication overhead
    - Synchronization overhead
    - Redundant work
  - Hardware limitations
    - Memory (also cache) bandwidth saturation
    - Network contention
    - I/O contention

- Can I make my code scale better by slowing it down?
  Absolutely, if communication and synchronization overhead are relevant.
  But you shouldn’t.

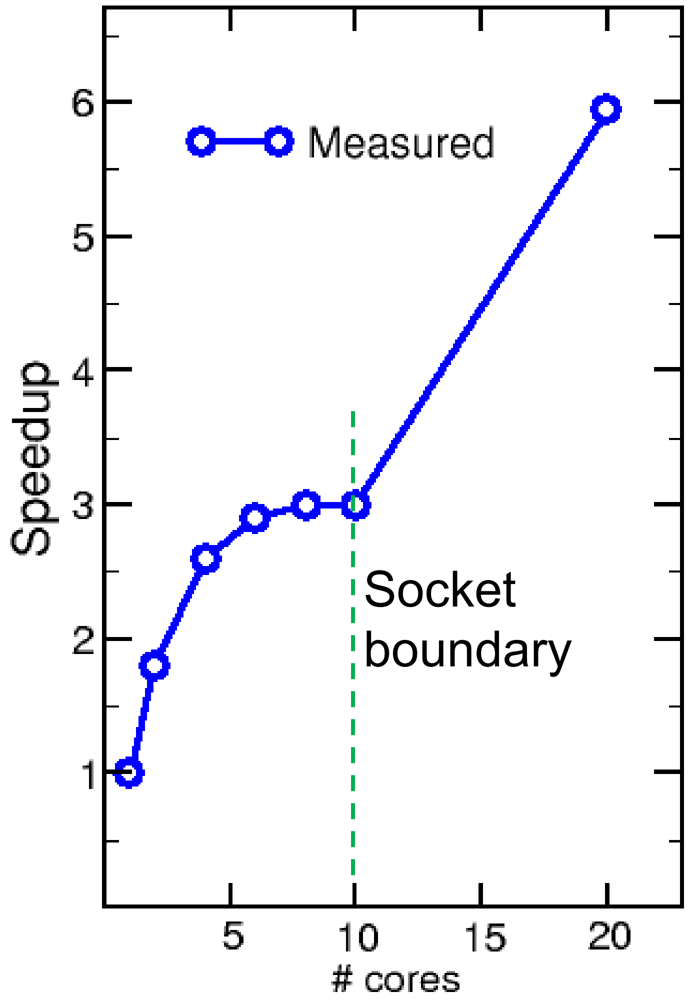
"Structural" scaling roadblocks

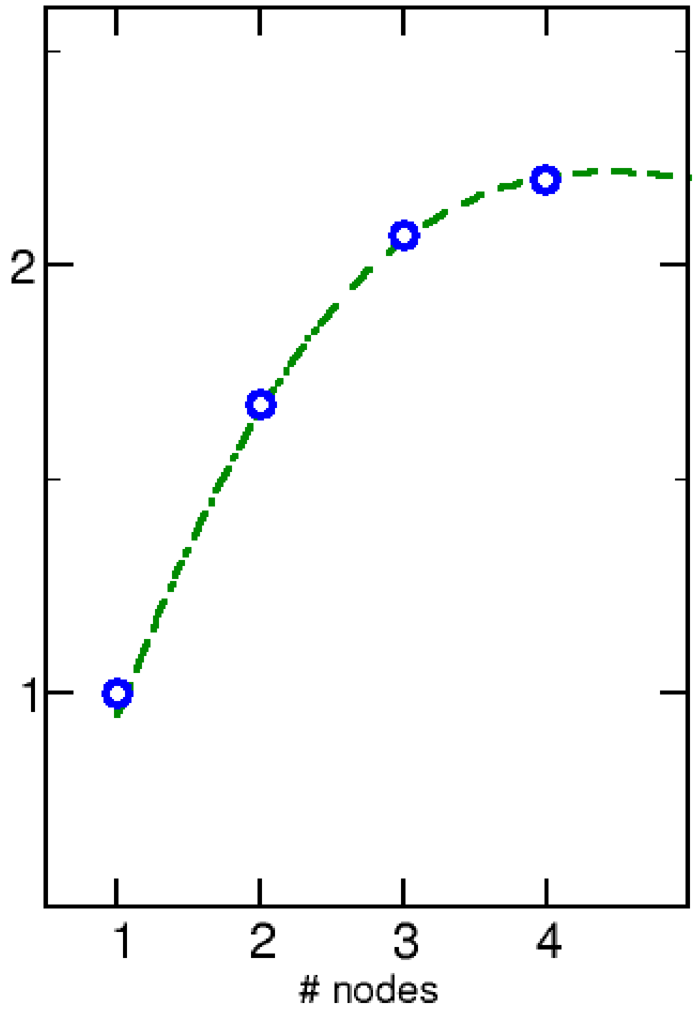
Communication, synchronization, work imbalance



Scaling baselines: Some resources do not scale

Scaling across cores, sockets, nodes

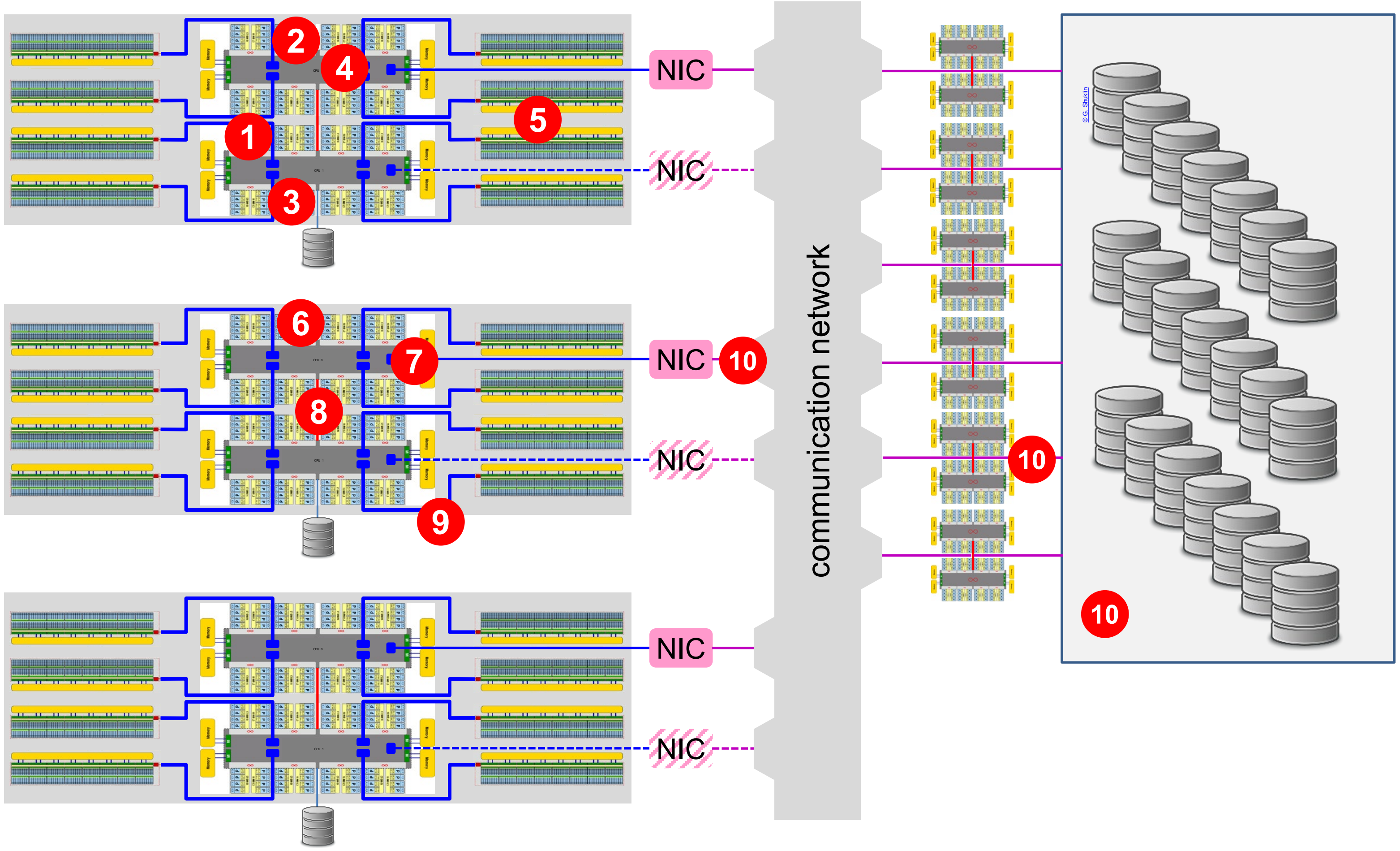




Does this code “scale”?

Scalablility of hardware components

Parallel and shared resources within a shared-memory node



Parallel resources:

- Execution units 1
- Cores 2
- Inner cache levels 3
- Sockets / memory domains 4
- Multiple accelerators 5

Shared resources:

- Outer cache levels 6
- Memory bus per socket 7
- Intersocket link 8
- PCIe bus(es) 9
- Other I/O resources 10

How does your application react to all of those details?

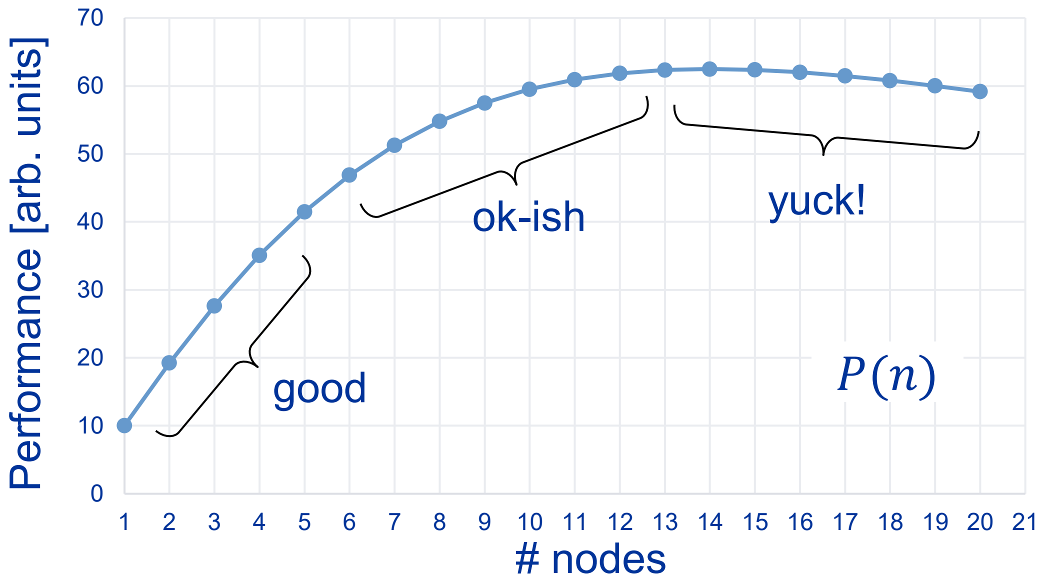
So what should I do?

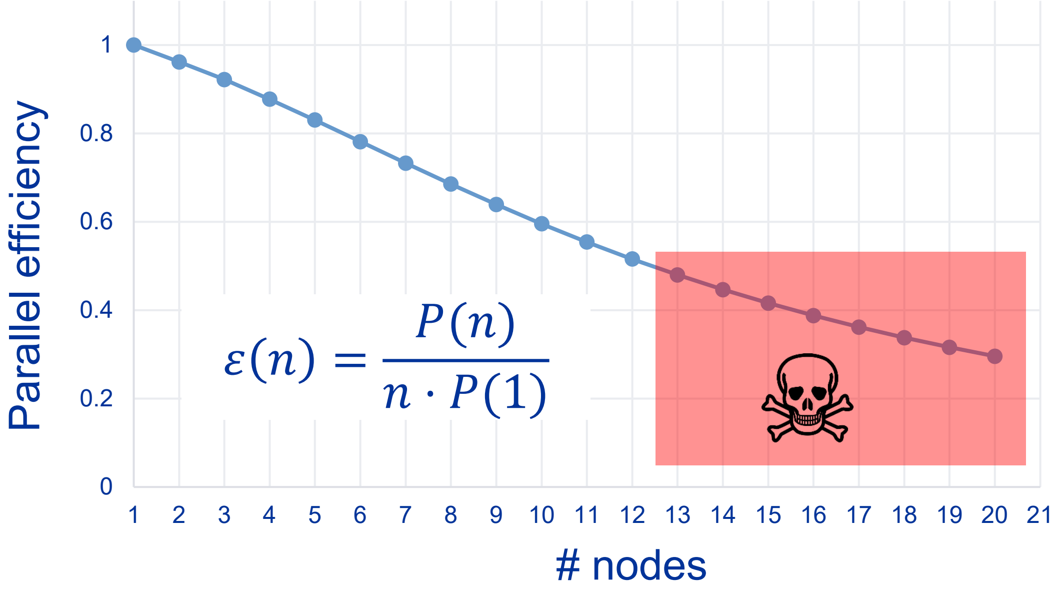
Assess the scaling properties of your code by benchmarking

- Scaling baseline: Basic allocation unit (node, GPU) first, then others
- Less than 50% efficiency is a blatant waste of resources

If you change the input (geometry, model, data set size), scaling will probably change, too

- Repeat scaling runs after significant changes to setup





What about performance (vs. scaling)?

- "Good" scaling does not mean that your code is fast
- It may still be that it makes bad use of the available main resources
  - Computational performance
  - Memory bandwidth

- Clustercockpit monitoring to the rescue
  - https://monitoring.nhr.fau.de
  - HPC Café (January 2023) on ClusterCockpit and the HPC Portal: https://www.fau.tv/clip/id/46327

Quiz

- How can I compute the peak performance of a CPU or a GPU?
  Multiply the amount of available resources on each level, e.g.:

  (SIMD width) x (#FP instr/cy) x (2) x (# cores) x (clock frequency)

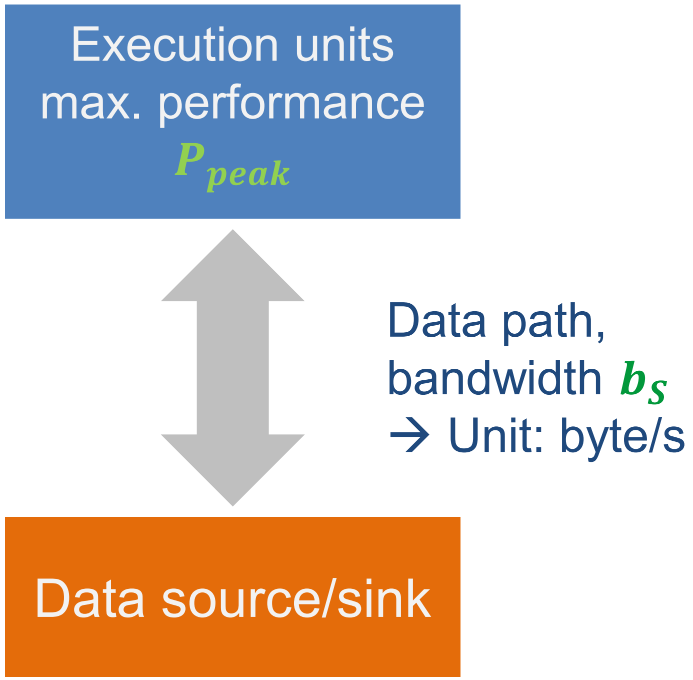
- How can I know the memory bandwidth of my CPU or GPU
  Run a streaming benchmark (e.g., STREAM Triad) to measure it

- What is the “Roofline Model”?
  A simple analytic performance model, which assumes that a loop’s performance is limited either by memory data transfer or by code execution, whichever takes longer

The Roofline Model

A simple performance model for loops

Simplistic view of the hardware:



Simplistic view of the software:

```
do i = 1,<sufficient>
  <complicated stuff doing
    N flops causing
    V bytes of data transfer>
enddo
```

Computational intensity $I = \frac{N}{V}$
→ Unit: flop/byte

Also in use: Code balance $B_c = \frac{V}{N}$
→ Unit: byte/flop

Other metrics for work are possible

Naïve Roofline Model

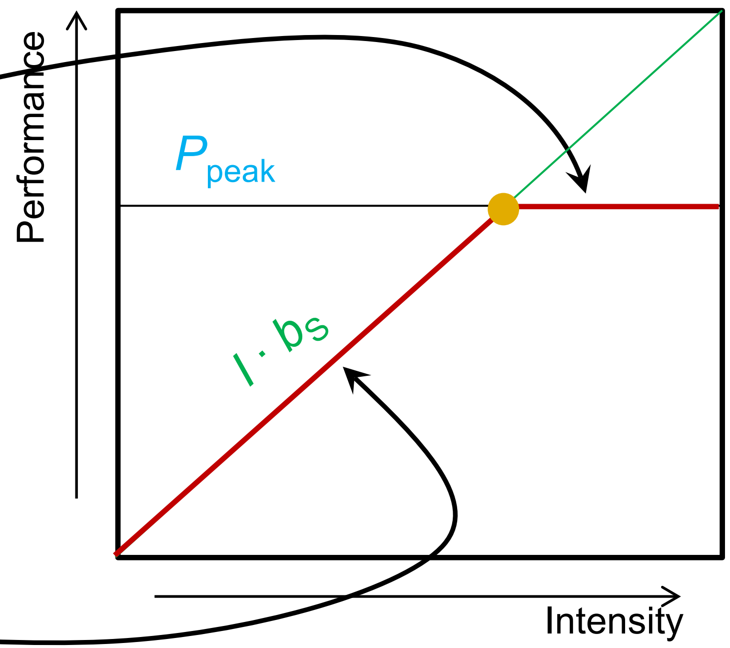
How fast can tasks be processed at most? ***P*** [flop/s]

The bottleneck is either
- The execution of work: P_{peak} [flop/s]
- The data path: $I \cdot b_S$ [flop/byte x byte/s]

$$P = \min(P_{\text{peak}}, I \cdot b_S)$$

This is the "Naïve Roofline Model"
- High intensity: P limited by execution
- Low intensity: P limited by data transfer
- "Knee" at $P_{peak} = I \cdot b_S$: Best use of resources

- Roofline is an "optimistic" model (think "light speed")



Roofline: application model and machine model

Apply the naive Roofline model in practice

- Machine parameter #1: Peak performance: $P_{peak}\left[\frac{F}{s}\right]$
- Machine parameter #2: Memory bandwidth: $b_S\left[\frac{B}{s}\right]$
- Code characteristic: Computational intensity: $I\left[\frac{F}{B}\right]$

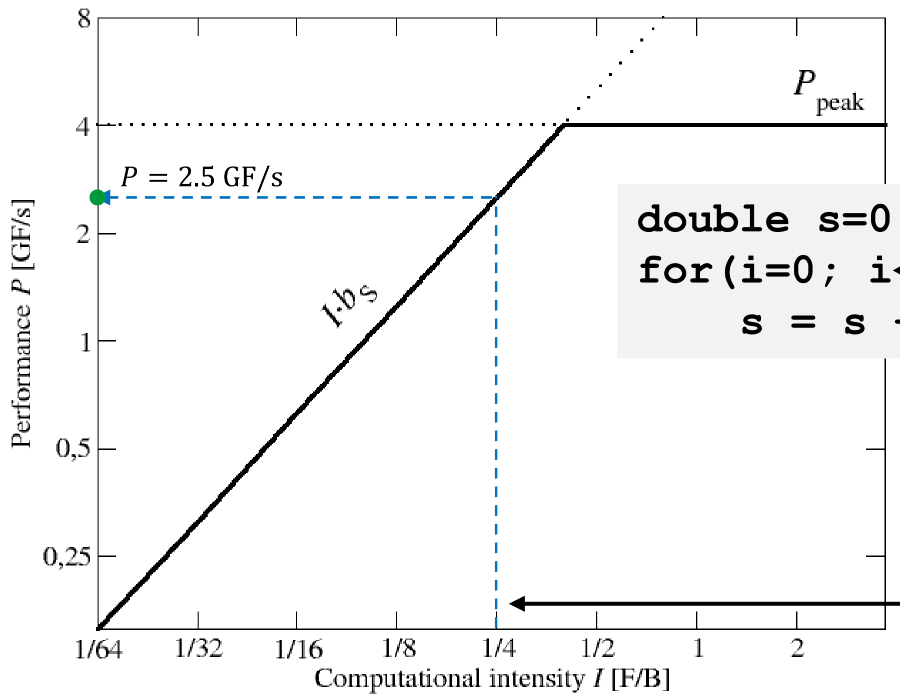
(Peak performance and memory bandwidth: Machine model; computational intensity: Application model)

Machine properties:

$$P_{peak} = 4\,\frac{\text{GF}}{\text{s}}$$

$$b_S = 10\,\frac{\text{GB}}{\text{s}}$$

Application property: I

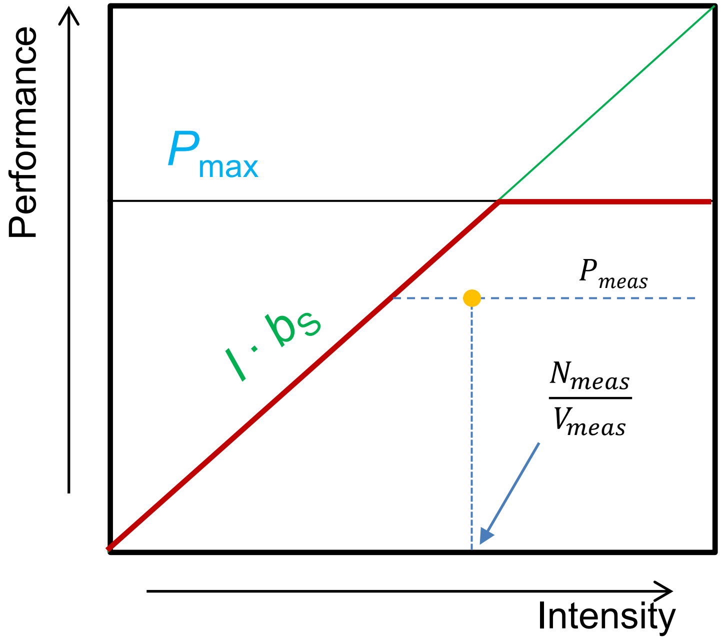



```
double s=0, a[];
for(i=0; i<N; ++i) {
    s = s + a[i] * a[i];}
```

$$I = \frac{2\,F}{8\,B} = 0.25\,{}^{F}/_{B}$$

Diagnostic modeling

- What if we cannot predict the intensity/balance?
  - Code very complicated
  - Code not available
  - Parameters unknown
  - Doubts about correctness of analysis
- Measure data volume V_{meas} (and work N_{meas})
  - Hardware performance counters
  - Tools: likwid-perfctr, PAPI, Intel Vtune,…
- Insights + benefits
  - Compare analytic model and measurement → validate model
  - Can be applied (semi-)automatically
  - Useful in performance monitoring of user jobs on clusters



Identifying problems: Typical performance patterns

Performance patterns 1: low-hanging fruits

- Too many/too few nodes allocated
- Load >#cores per node
- Non-usage of allocated GPU

Probably an oversight, or you copied a script without proper adaptations.

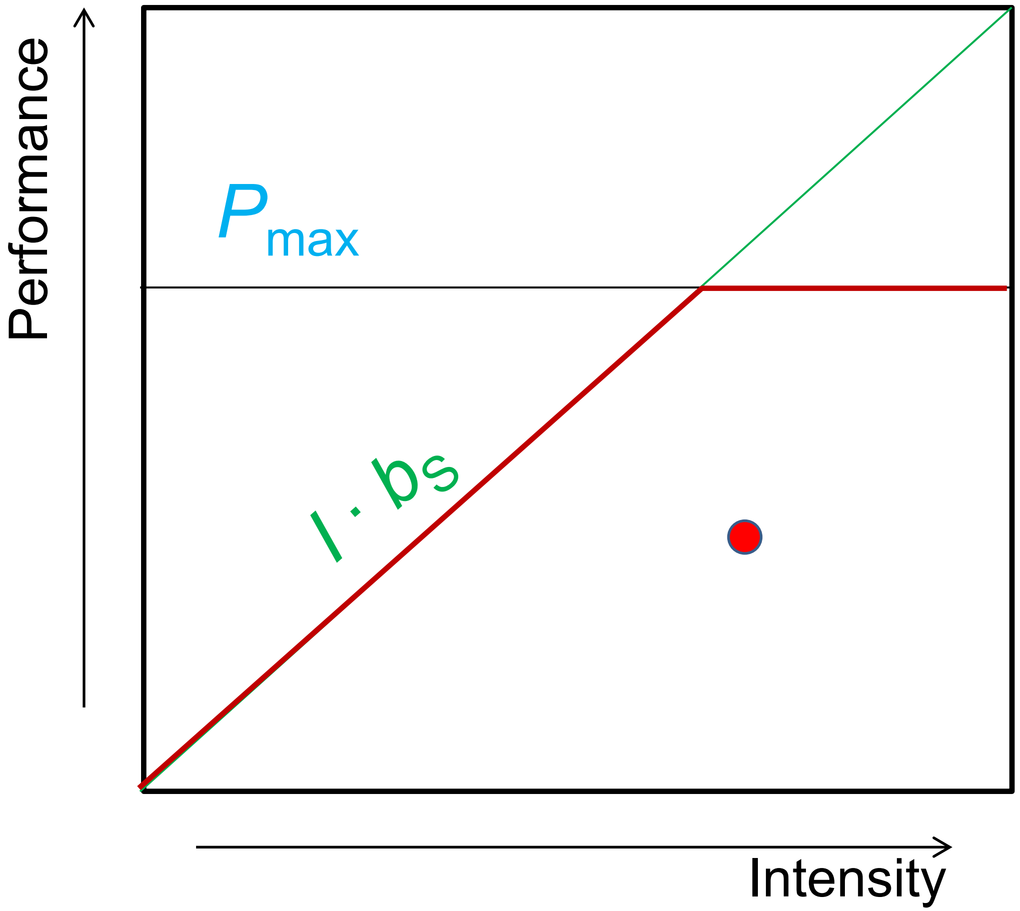
Easy solution: Fix your job script

Performance patterns 2: bad hardware utilization

- Far away from Roofline in diagnostic Roofline plot
  → no large fraction of memBW
  → no large fraction of peak

Possible reasons?

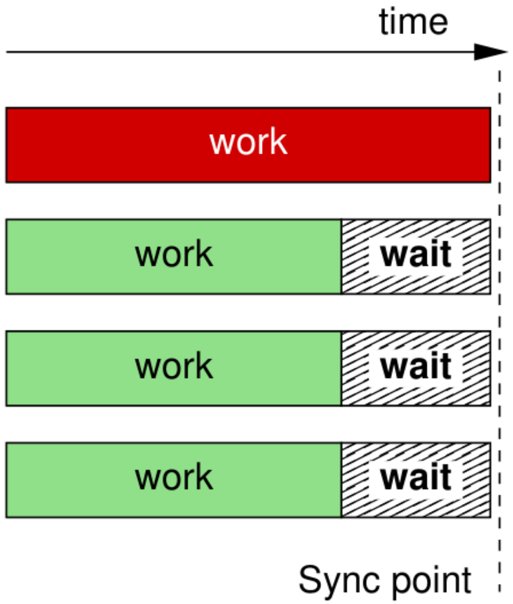
- "Invisible performance ceiling"
- Load imbalance
- Bad memory access patterns
- Large overhead from I/O or communication/synchronization
- Anything from previous slide

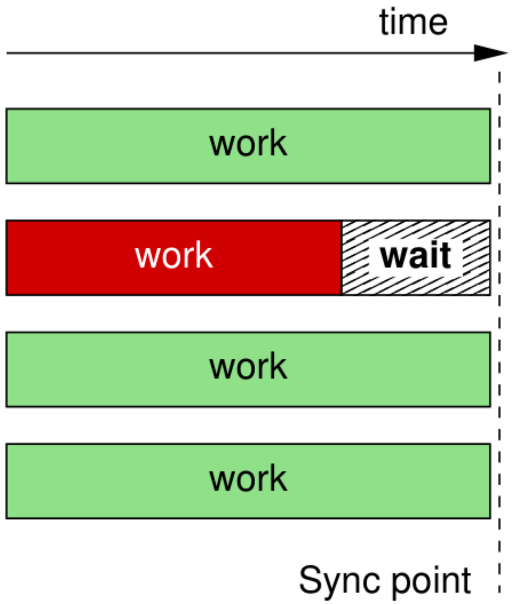


Performance patterns 2: bad hardware utilization

- Low vectorization ratio
  Low ratio of vectorized (SIMD) vs. scalar instructions; not necessarily bad
  - Some codes just cannot be vectorized
  - If hardware utilization is still good, you might not care
  - If SIMD pays off, a factor of up to 8x (DP) might be achievable

- Load imbalance (actually, execution time imbalance)
  Should usually be fixed; however, memory-bound code is more forgiving towards load imbalance (why?)
  Caveat: Two extreme cases!





Performance patterns 3: I/O

- High IB package rate
  - IB latency is in the low-µs range; hundreds of millions of IB packages per second are thus near the limit
  - Remedy: Communicate less ☺, aggregation
  - Probably you are just using too many nodes/processes

- High NFS rate

  Some codes write to NFS-mounted volumes frequently; a “fat” server can take up to 500 MB/sec

- Fine-grained, high-frequency I/O

  Rapid-fire I/O requests can overload the metadata servers and severely slow down the shared file system for all users