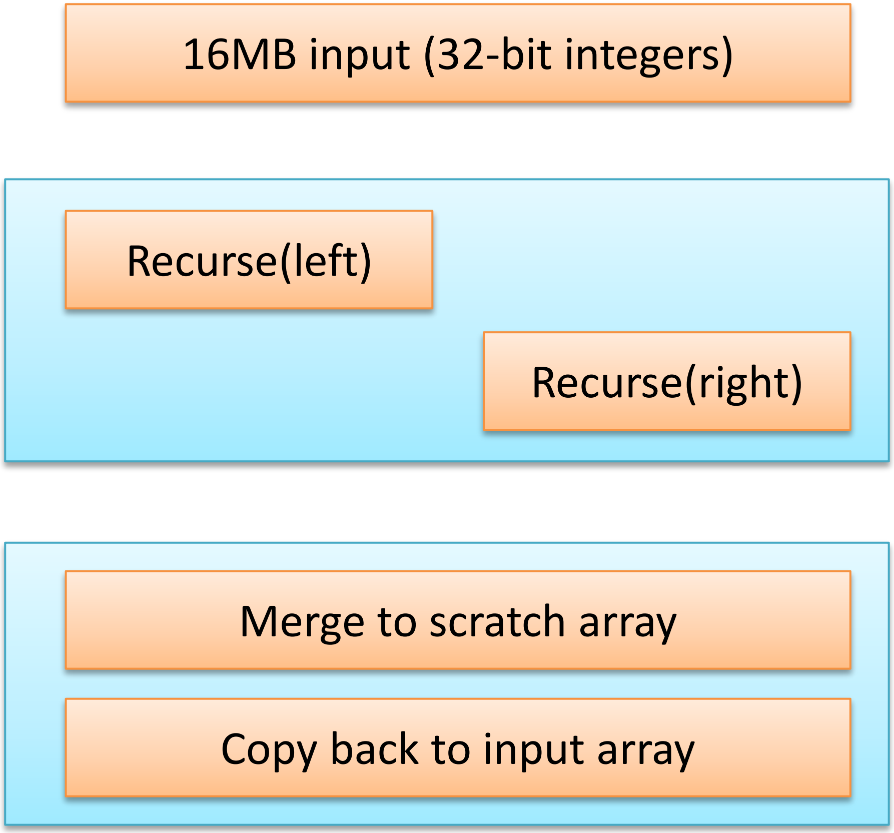
Multicore Programming

Parallel Hardware and Performance

8 Nov 2010 (Part 1)

Peter Sewell Jaroslav Ševčík Tim Harris

Merge sort



16MB input (32-bit integers)

Recurse(left)

Recurse(right)

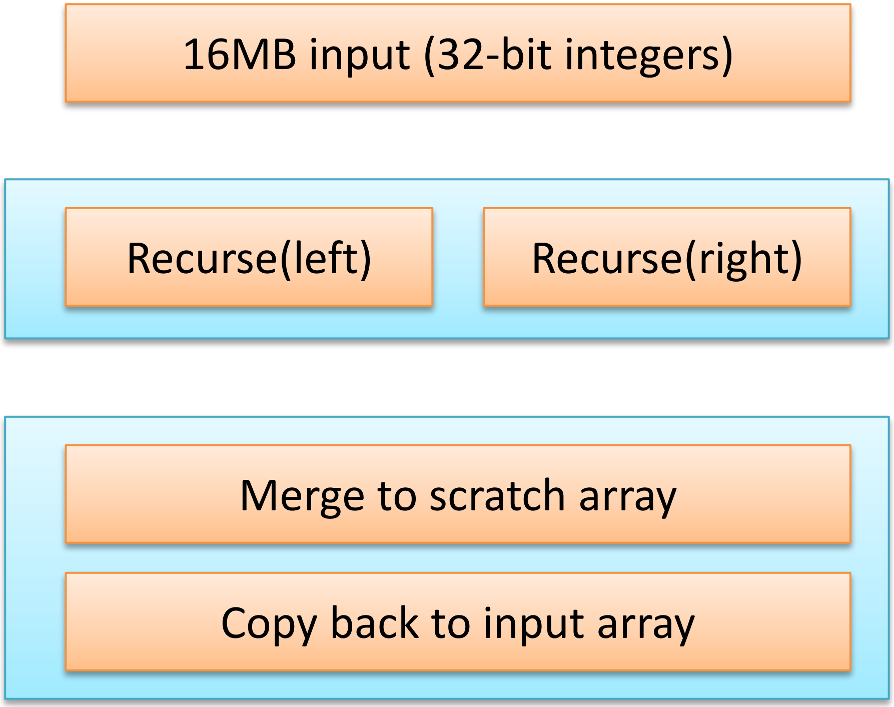
Merge to scratch array

Copy back to input array

~98% execution time

~2% execution time

Merge sort, dual core



16MB input (32-bit integers)

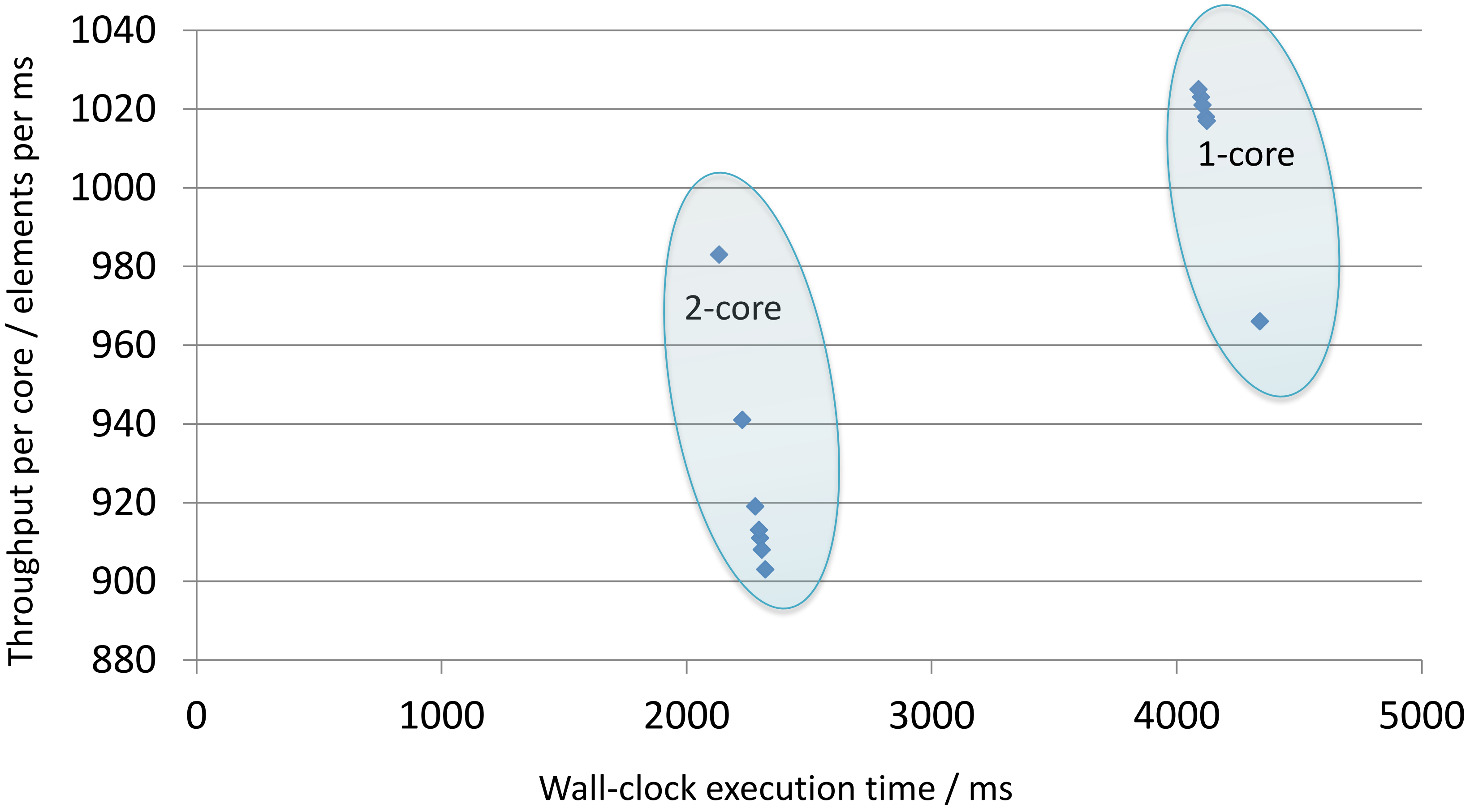
Recurse(left)

Recurse(right)

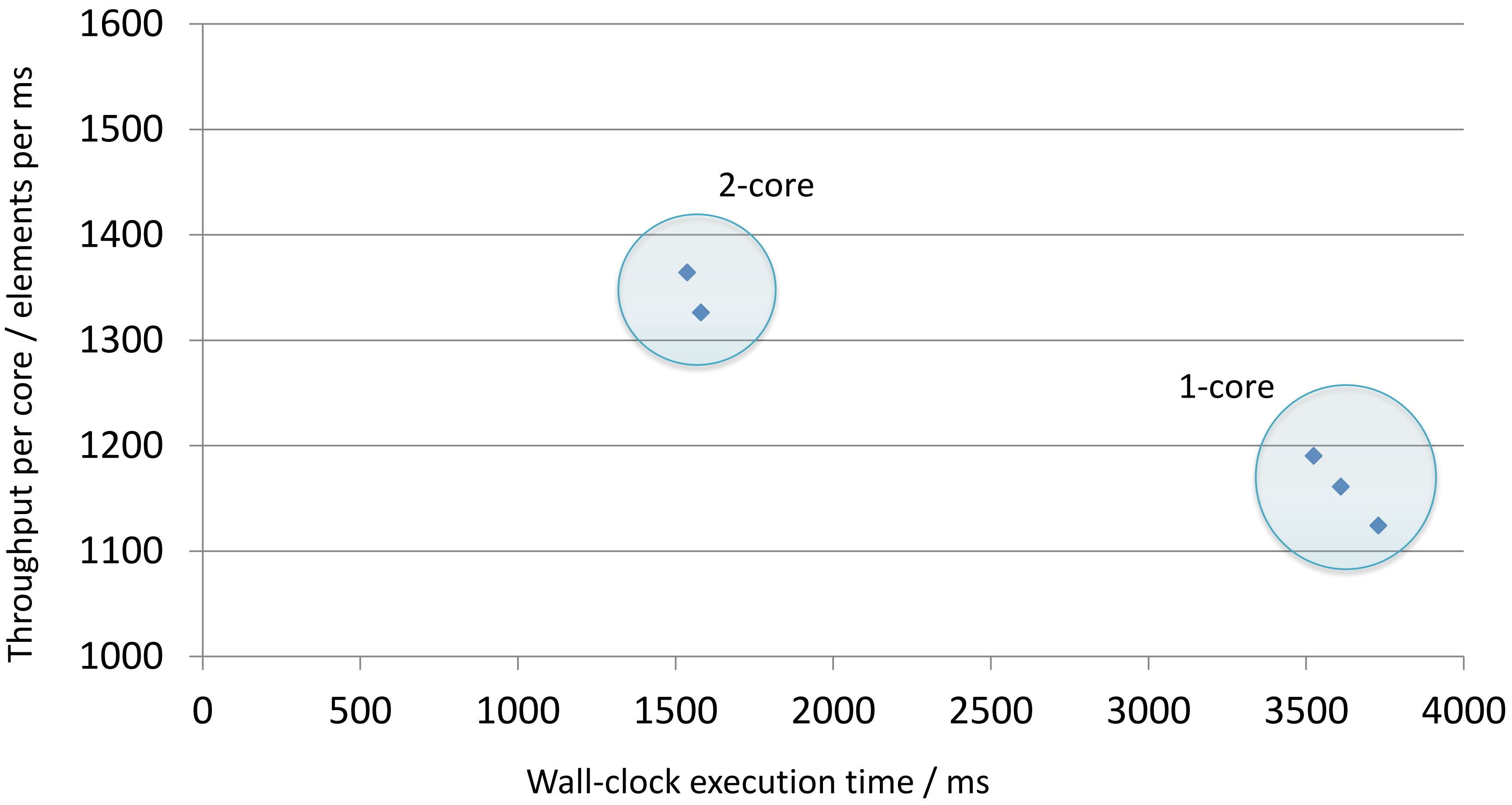
Merge to scratch array

Copy back to input array

T7300 dual-core laptop



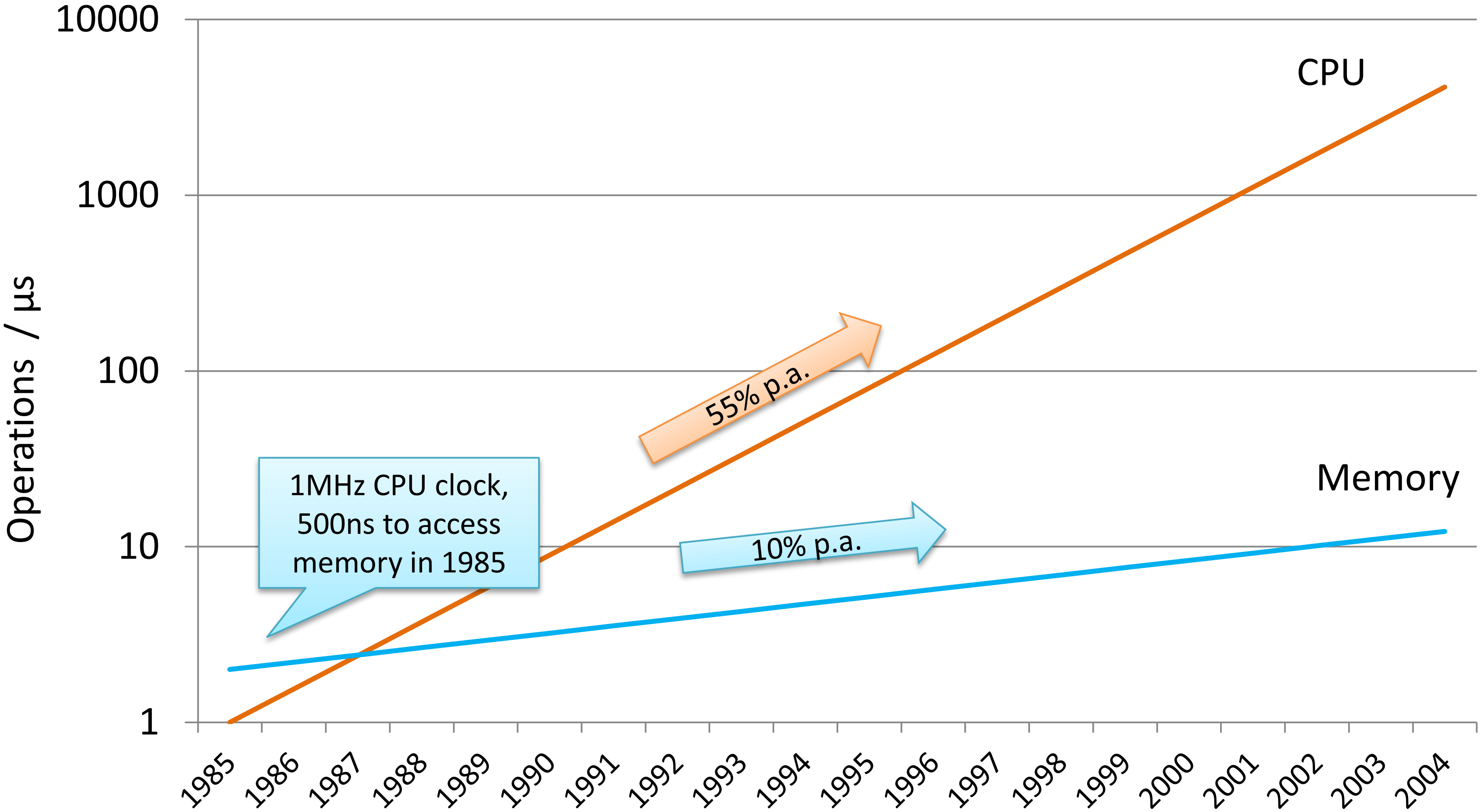
AMD Phenom 3-core



The power wall

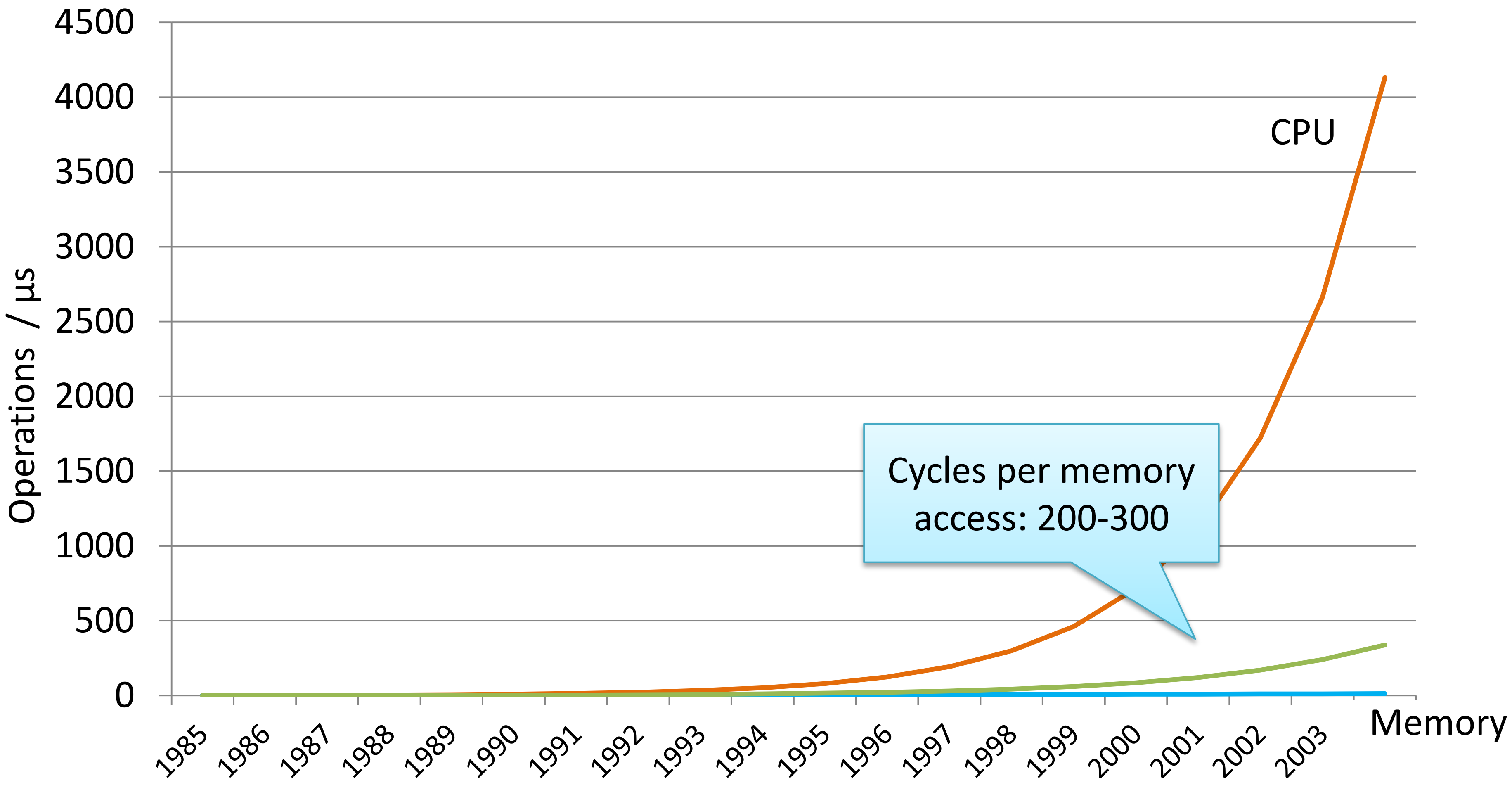
- Moore's law: area of transistor is about 50% smaller each generation
  - A given chip area accommodates twice as many transistors
- $P_{dyn} = \alpha f V^2$
  - Shrinking process technology (constant field scaling) allows reducing V to partially counteract increasing f
  - V cannot be reduced arbitrarily
    - Halving V more than halves max f (for a given transistor)
    - Physical limits
- $P_{leak} = V(I_{sub} + I_{ox})$
  - Reduce I_{sub} (sub-threshold leakage): turn off component, increase threshold volatage (reduces max f)
  - Reduce I_{ox} (gate-oxide leakage): increase oxide thickness (but it needs to decrease with process scale)

The memory wall



1MHz CPU clock, 500ns to access memory in 1985

The memory wall



Cycles per memory access: 200-300

The ILP wall

- ILP = "Instruction level parallelism"
- Implicit parallelism between instructions in a single thread
- Identified by the hardware
  - Speculate past memory accesses
  - Speculate past control transfer
- Diminishing returns

Power wall + ILP wall + memory wall = brick wall

- Power wall means we can’t just clock processors faster any longer
- Memory wall means that many workload’s perf is dominated by memory access times
- ILP wall means we can’t find extra work to keep functional units busy while waiting for memory accesses

Why parallelism?

Amdahl’s law

Why asymmetric performance?

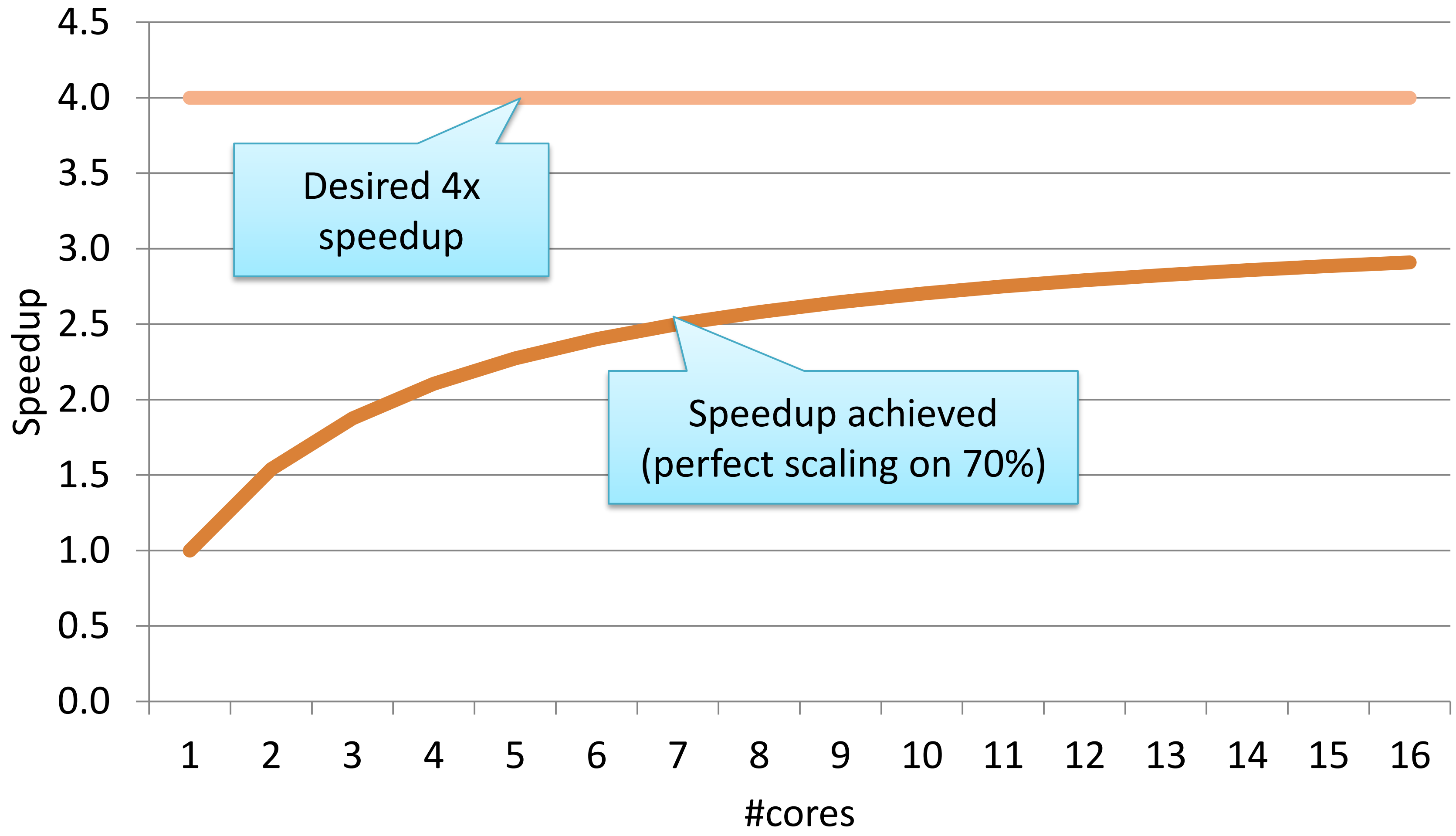
Parallel algorithms

Multi-processing hardware

Amdahl's law

- "Sorting takes 70% of the execution time of a sequential program. You replace the sorting algorithm with one that scales perfectly on multi-core hardware. On a machine with *n* cores, how many cores do you need to use to get a 4x speed-up on the overall algorithm?"

Amdahl's law, f=70%



Desired 4x speedup

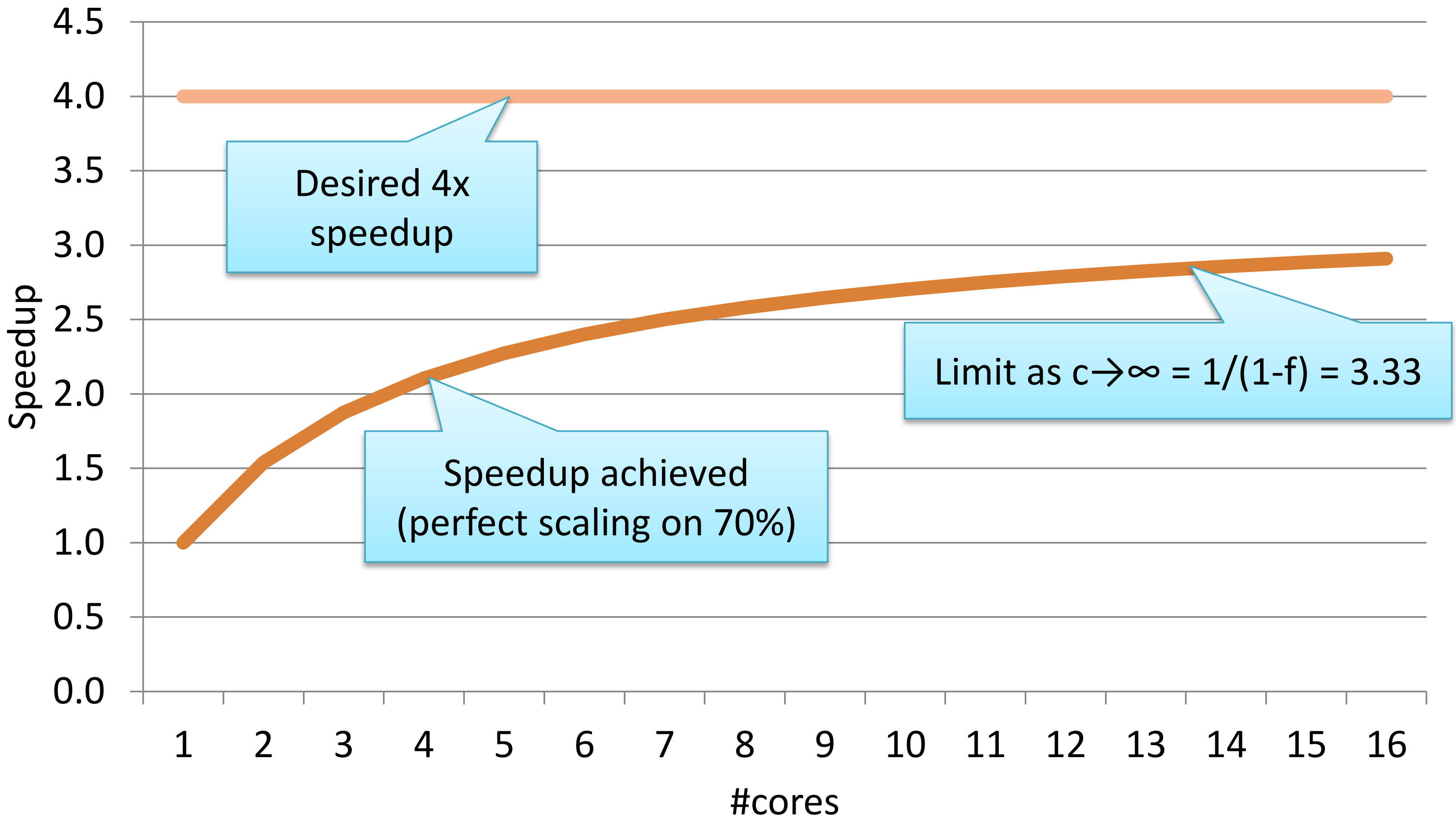
Speedup achieved (perfect scaling on 70%)

Amdahl's law, f=70%

$$MaxSpeedup(f,c) = \frac{1}{(1-f) + {}^{f}/_{c}}$$

f = fraction of code speedup applies to
c = number of cores used

Amdahl’s law, f=70%

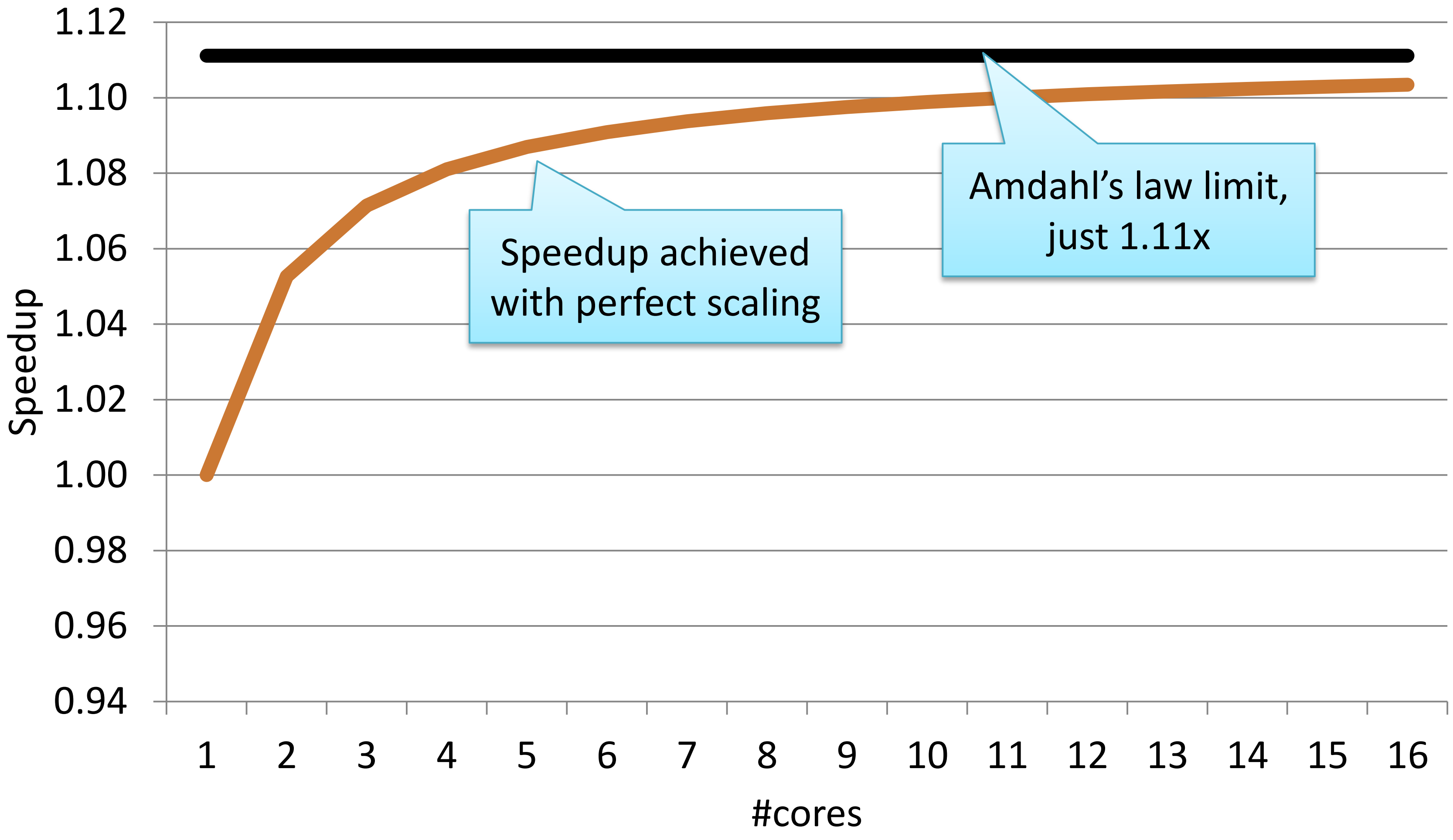


Desired 4x speedup

Limit as c→∞ = 1/(1-f) = 3.33

Speedup achieved (perfect scaling on 70%)

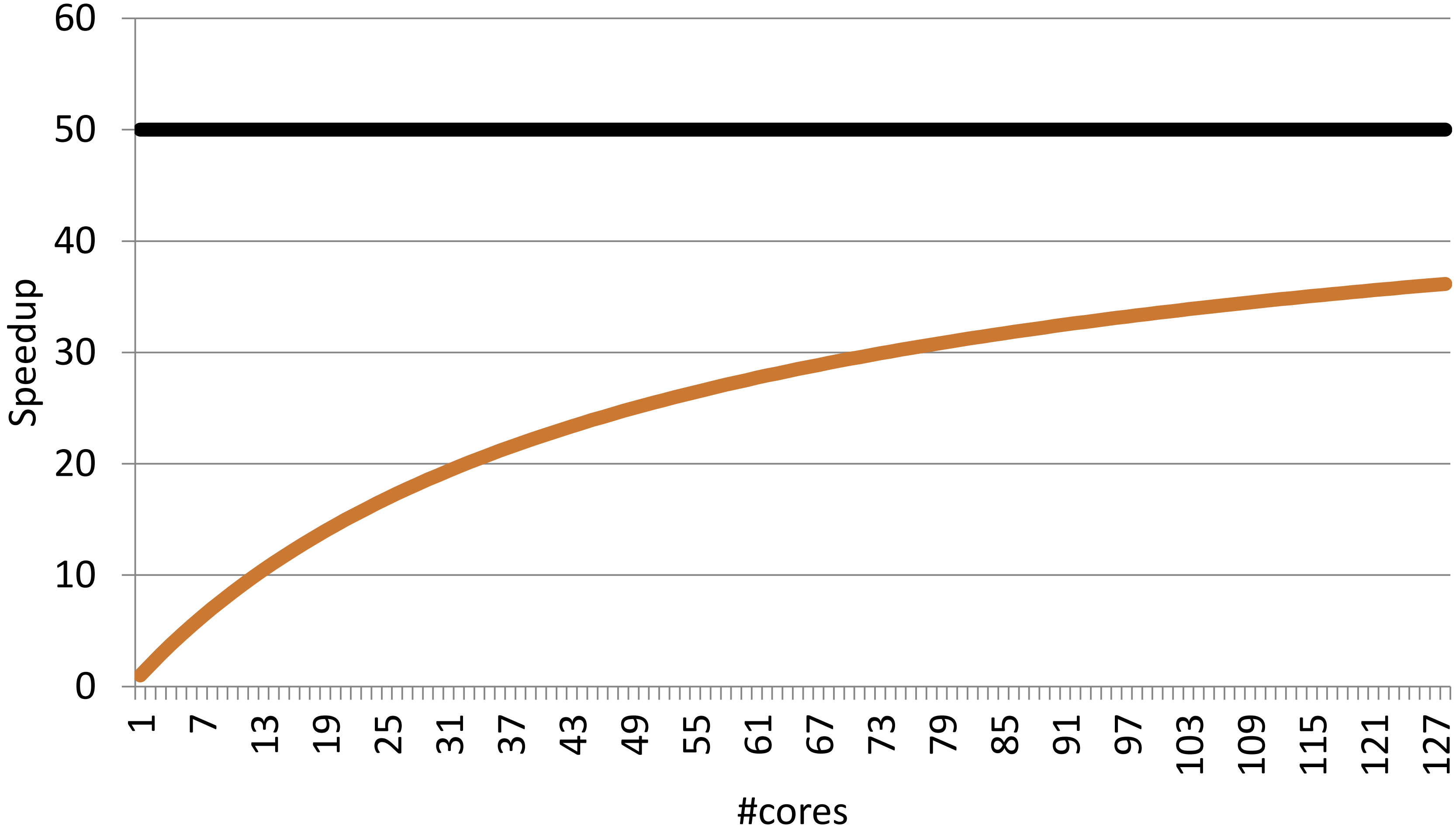
Amdahl’s law, f=10%



Speedup achieved with perfect scaling

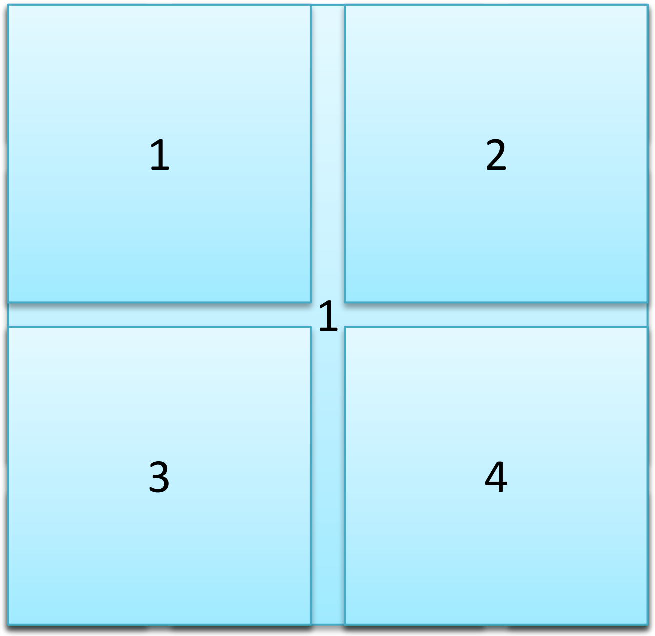
Amdahl’s law limit, just 1.11x

Amdahl's law, f=98%

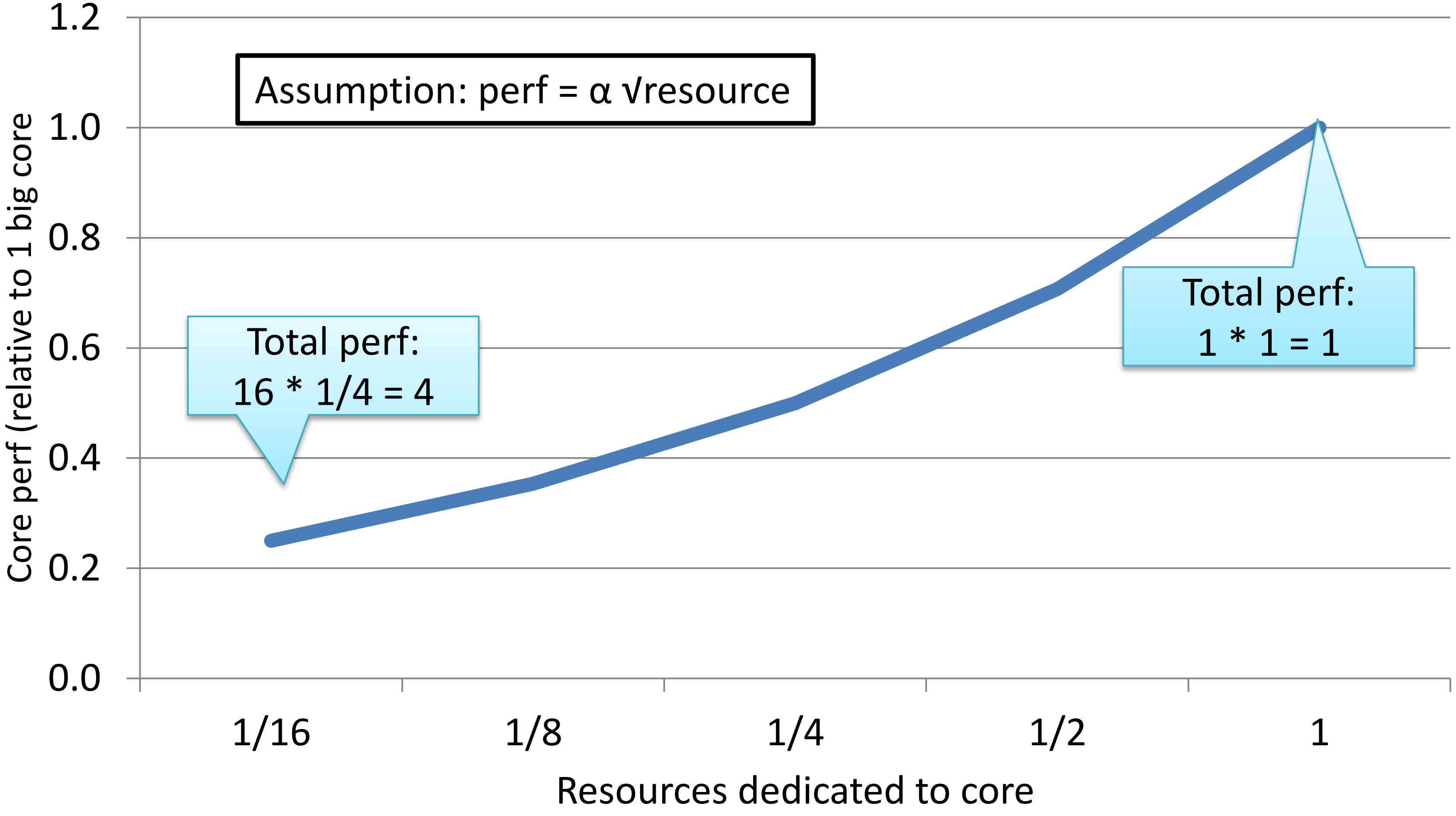


Amdahl’s law & multi-core

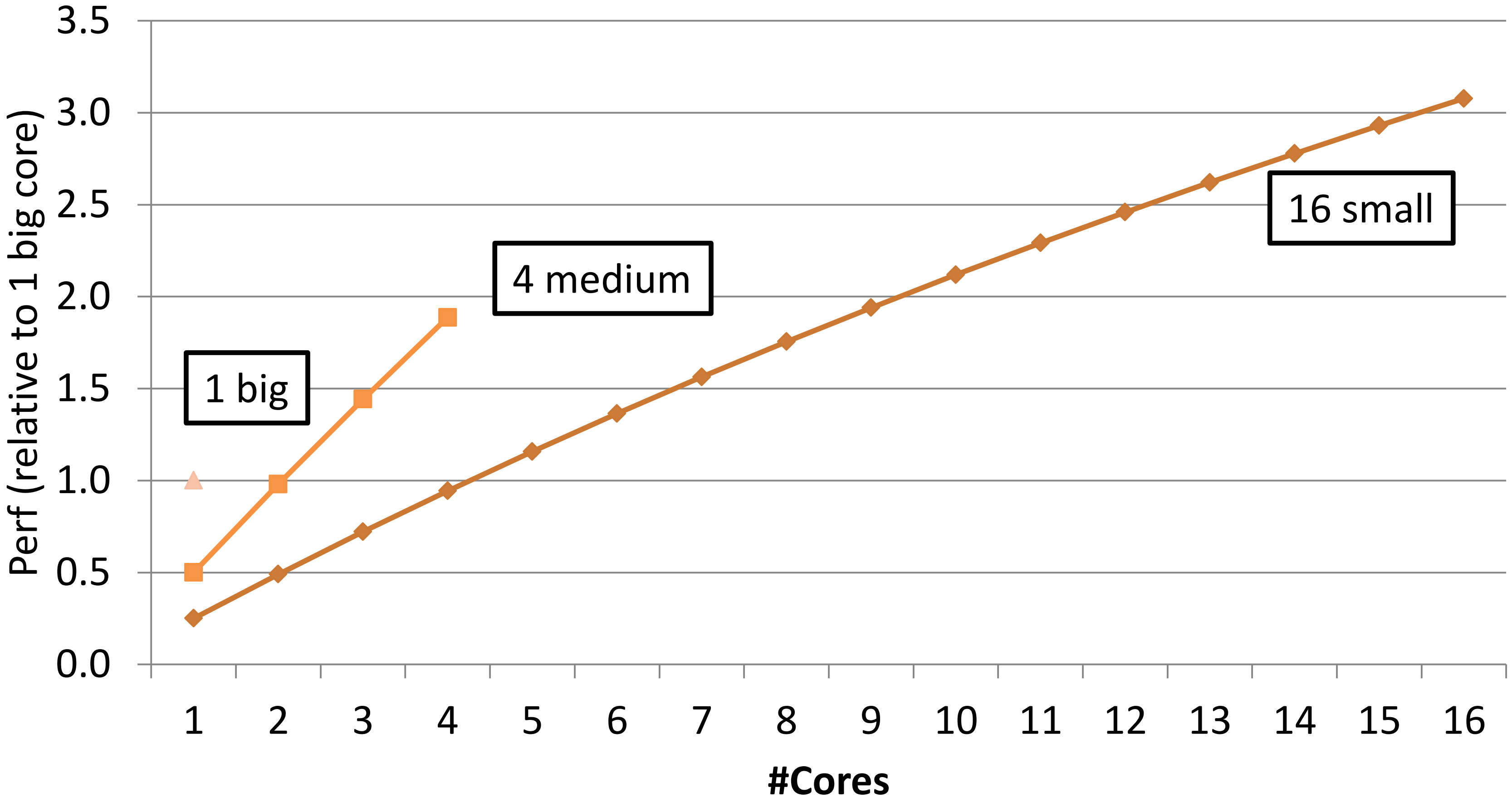
Suppose that the same h/w budget (space or power) can make us:



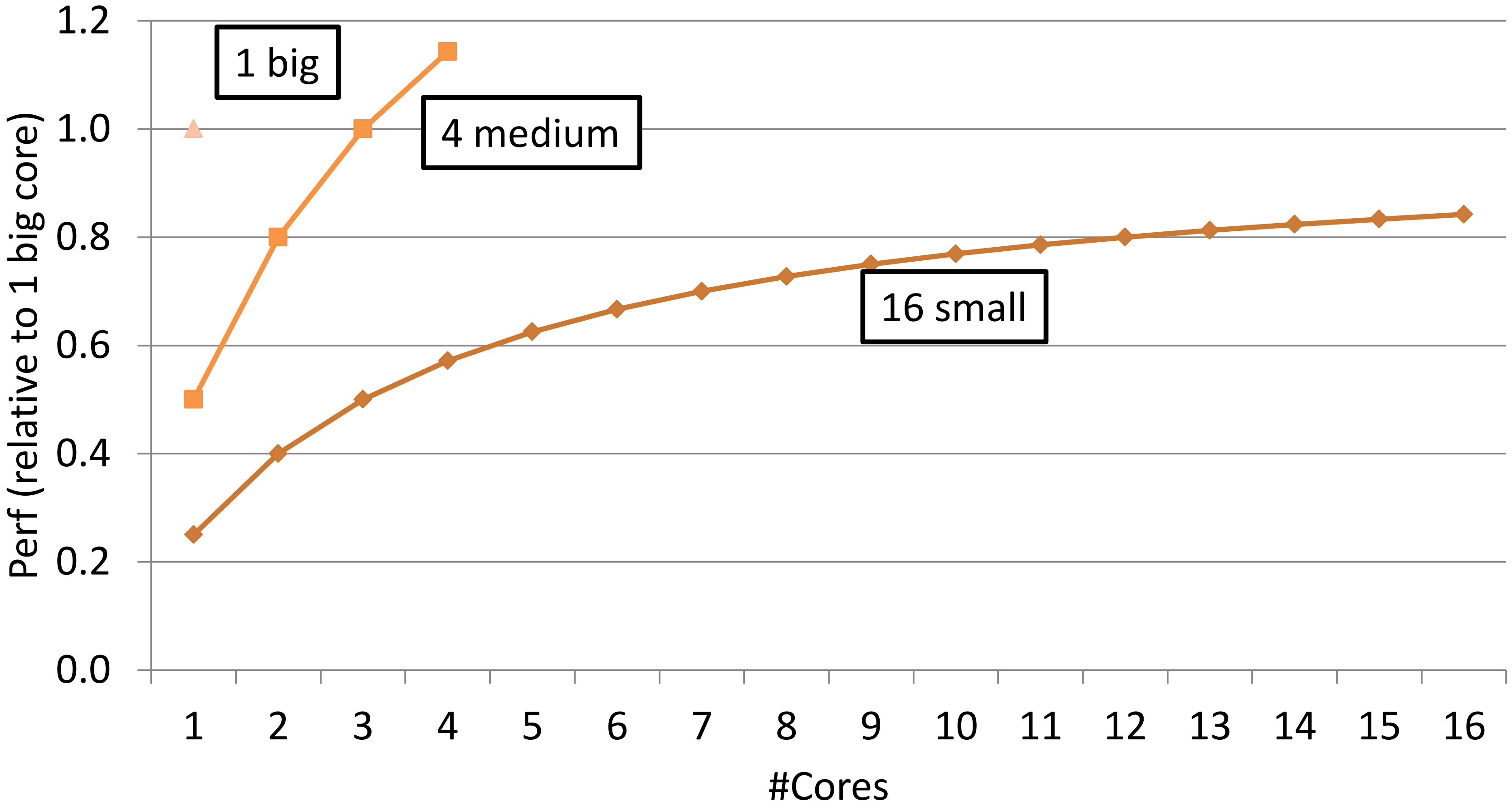
Perf of big & small cores



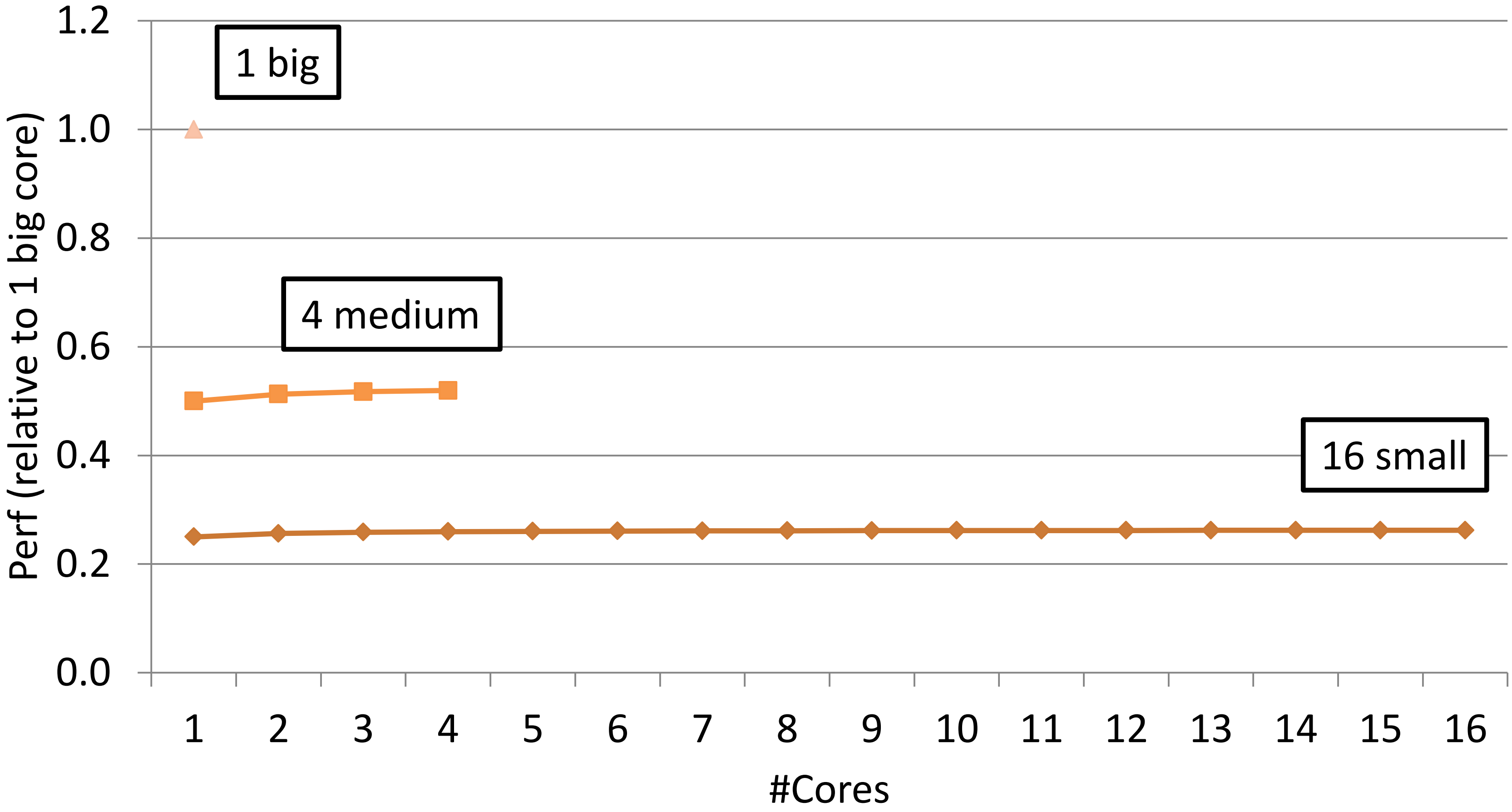
Amdahl’s law, f=98%



Amdahl's law, f=75%



Amdahl's law, f=5%



Why parallelism?

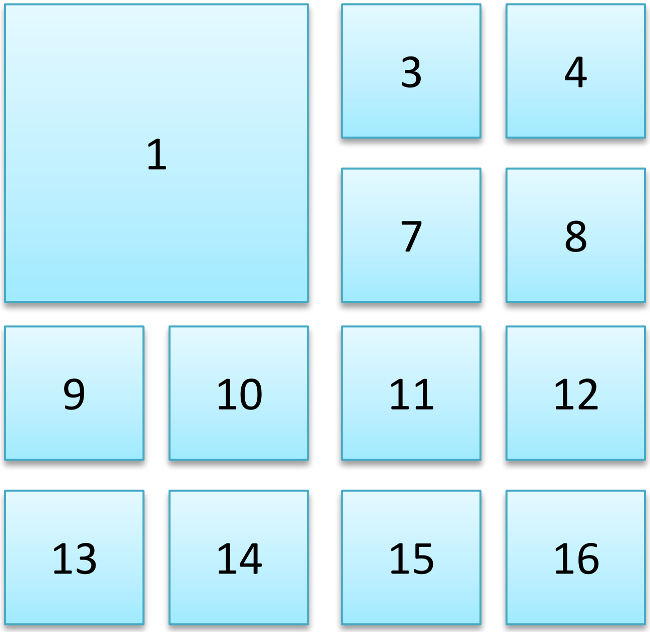
Amdahl’s law

Why asymmetric performance?

Parallel algorithms

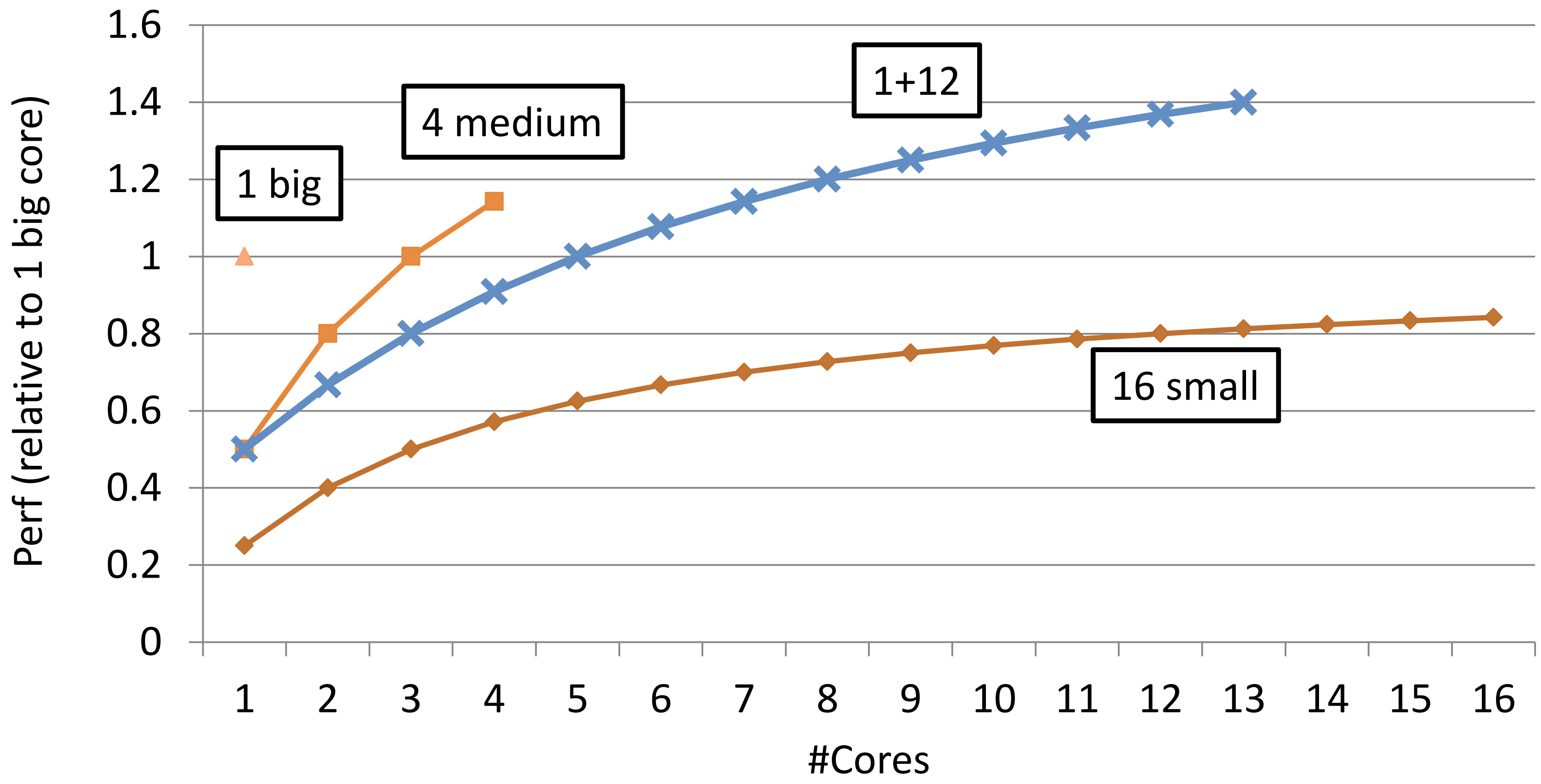
Multi-processing hardware

Asymmetric chips

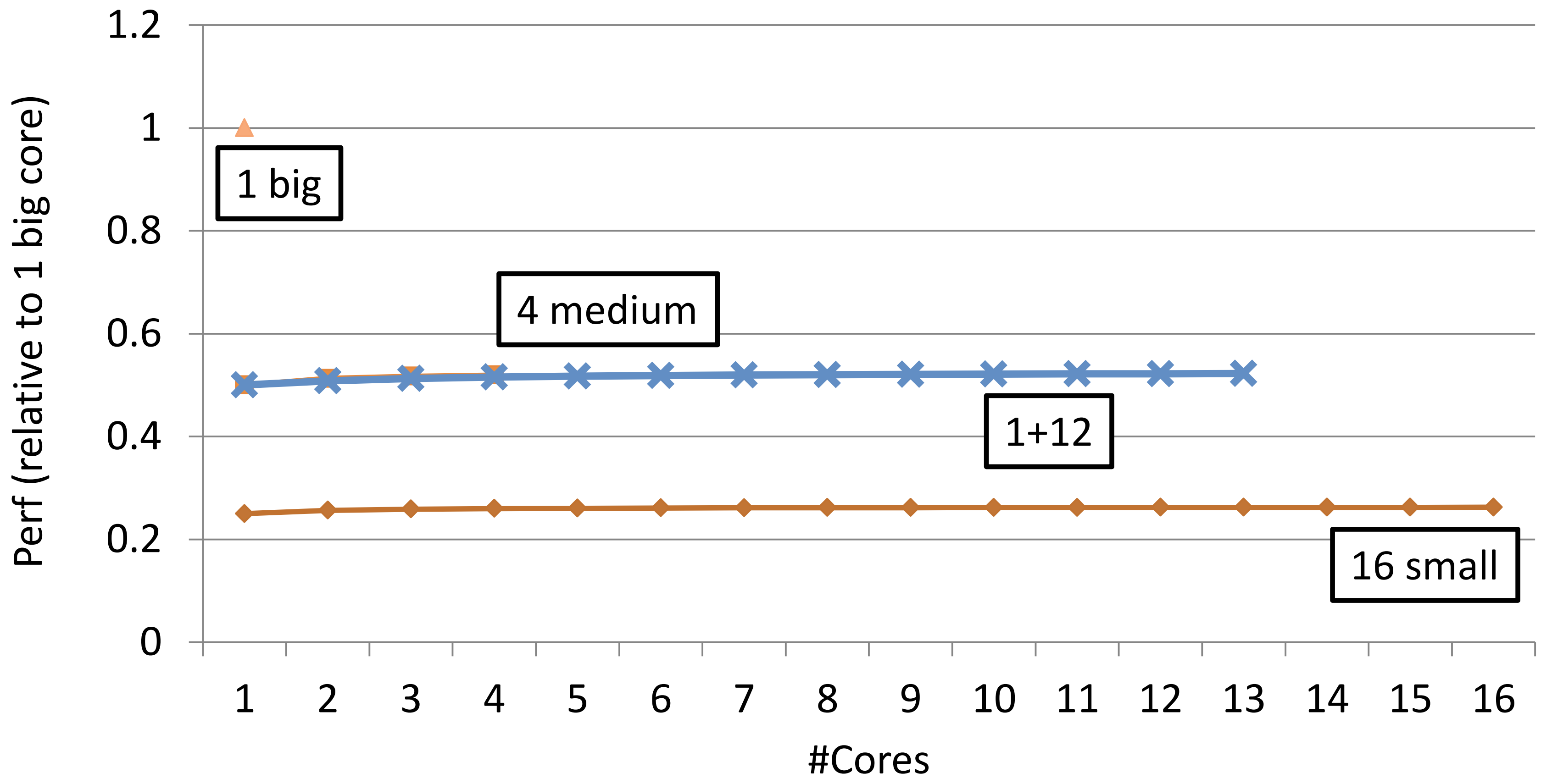


1 3 4
7 8
9 10 11 12
13 14 15 16

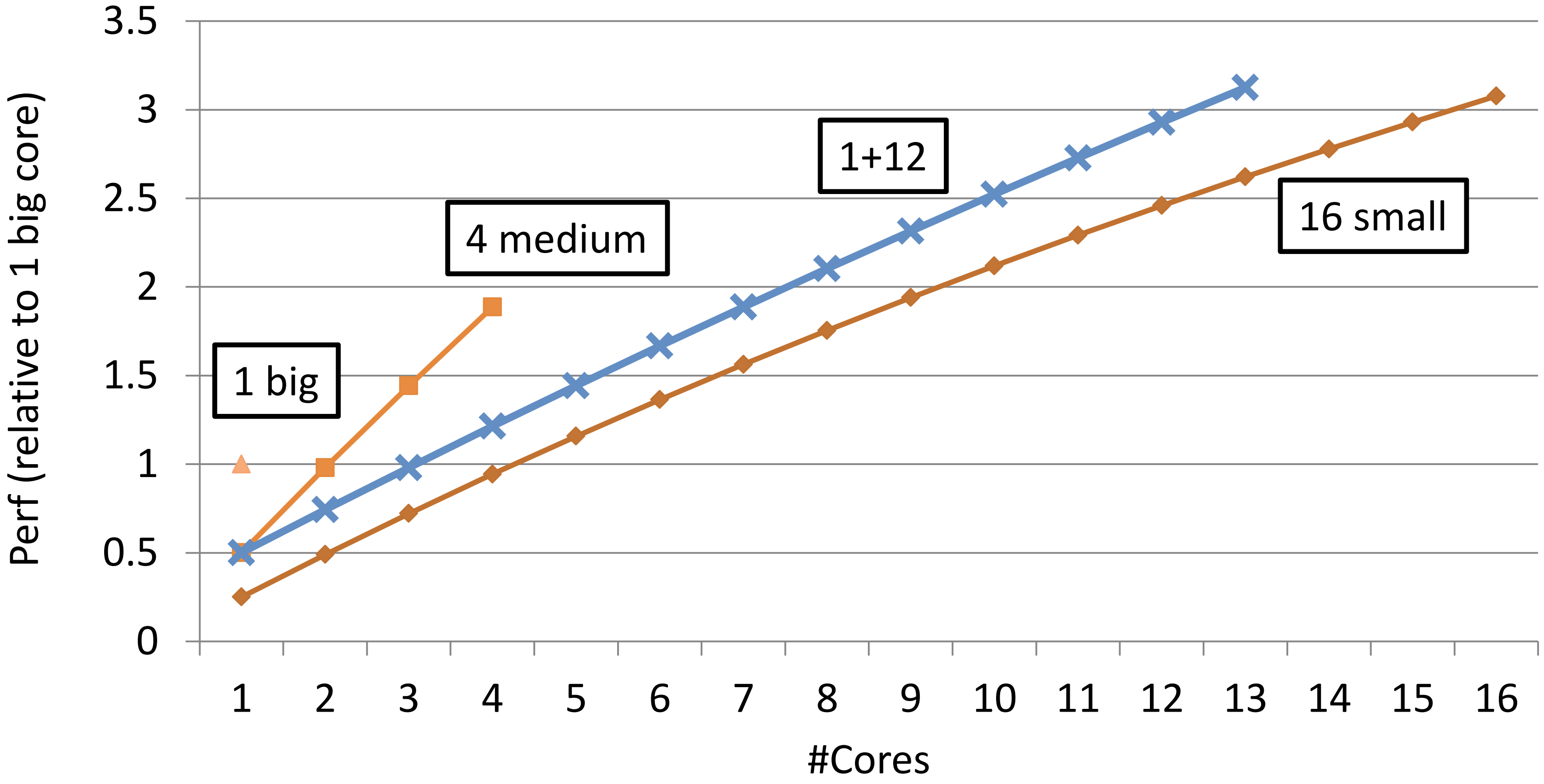
Amdahl's law, f=75%



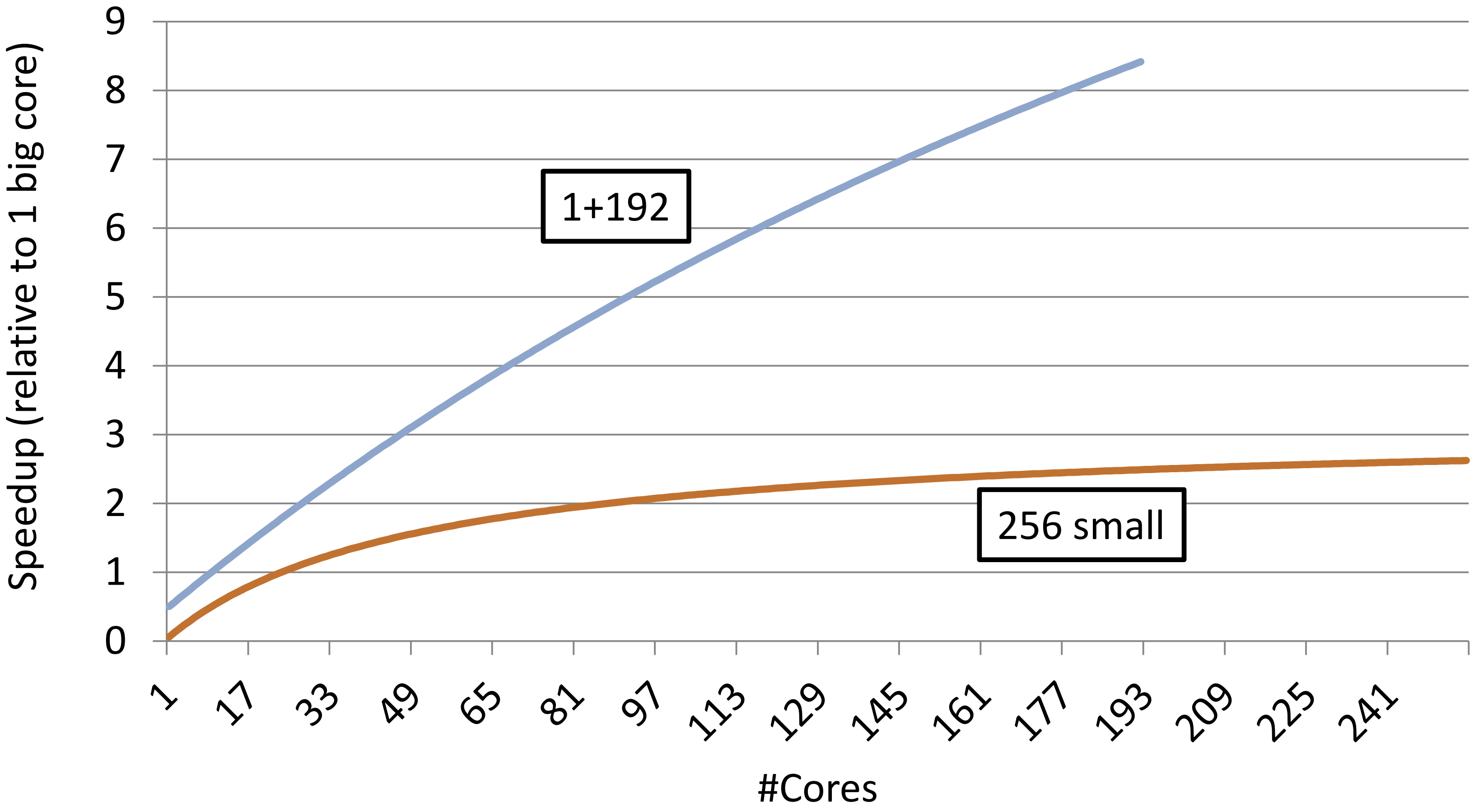
Amdahl’s law, f=5%



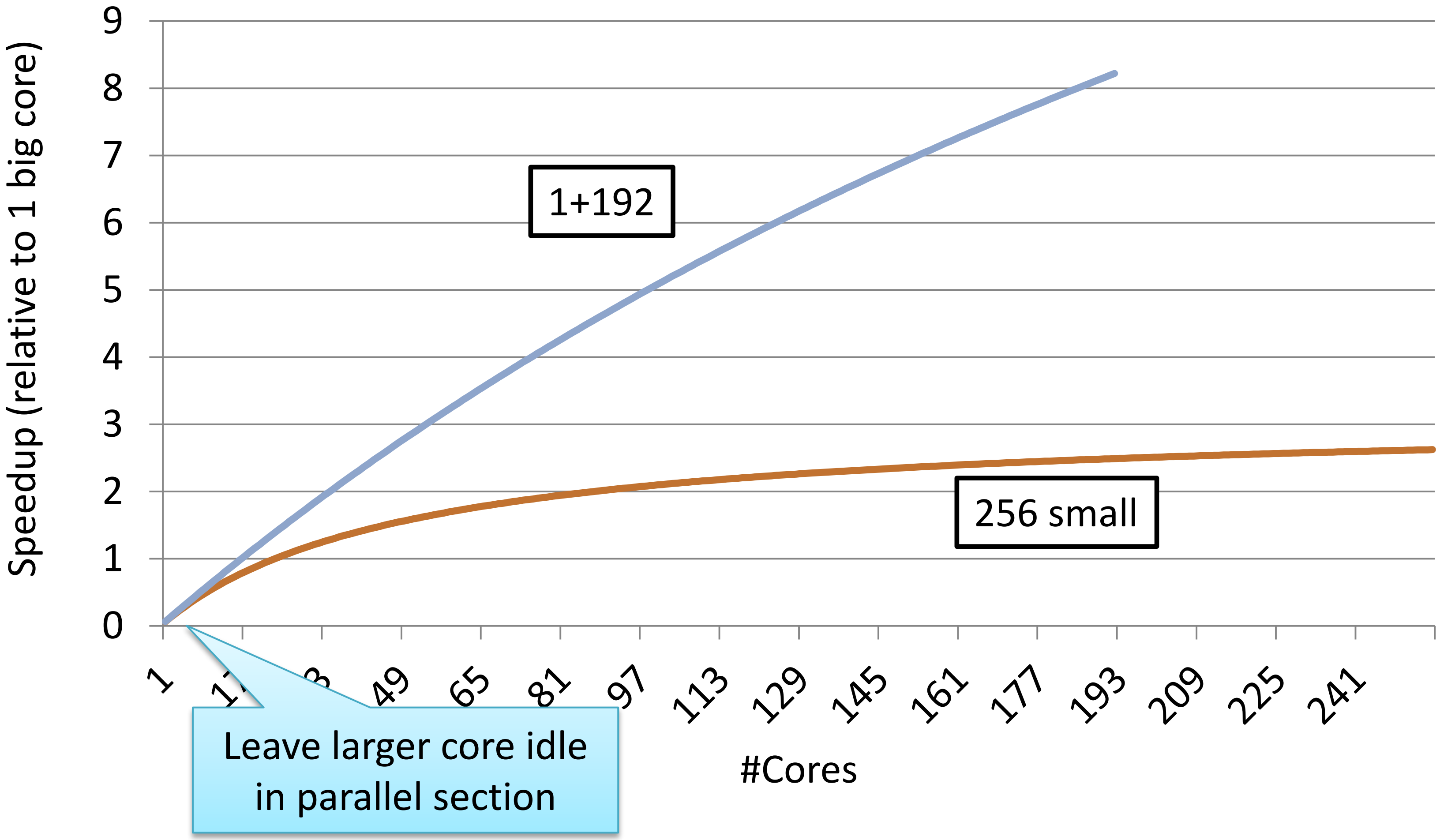
Amdahl's law, f=98%



Amdahl’s law, f=98%



Amdahl's law, f=98%



Leave larger core idle in parallel section

Why parallelism?

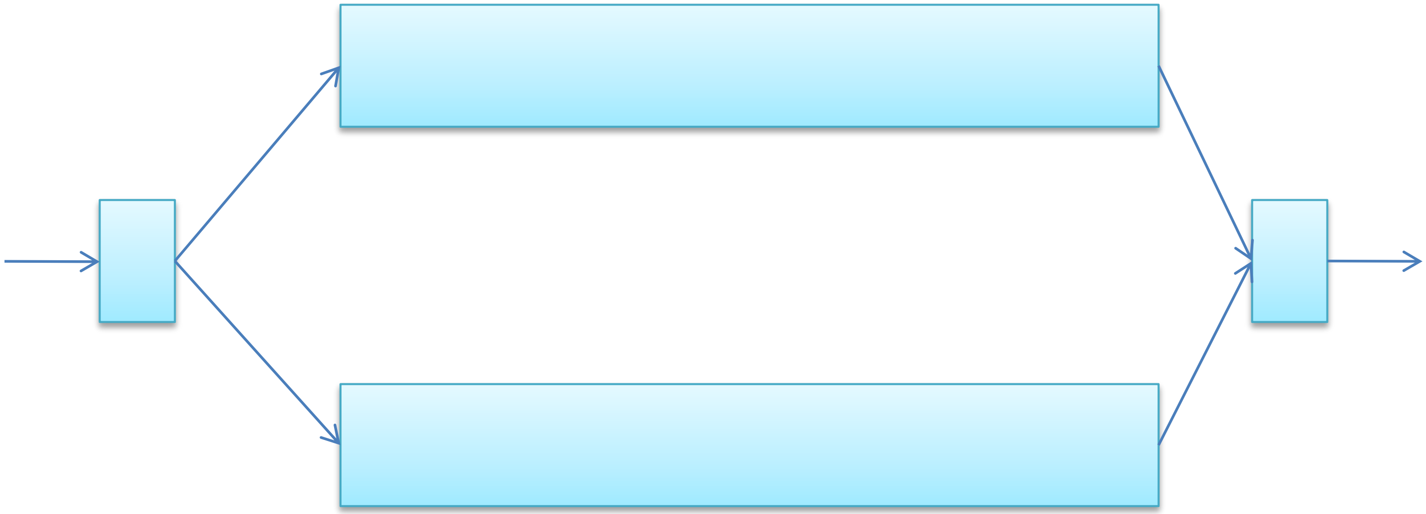
Amdahl’s law

Why asymmetric performance?

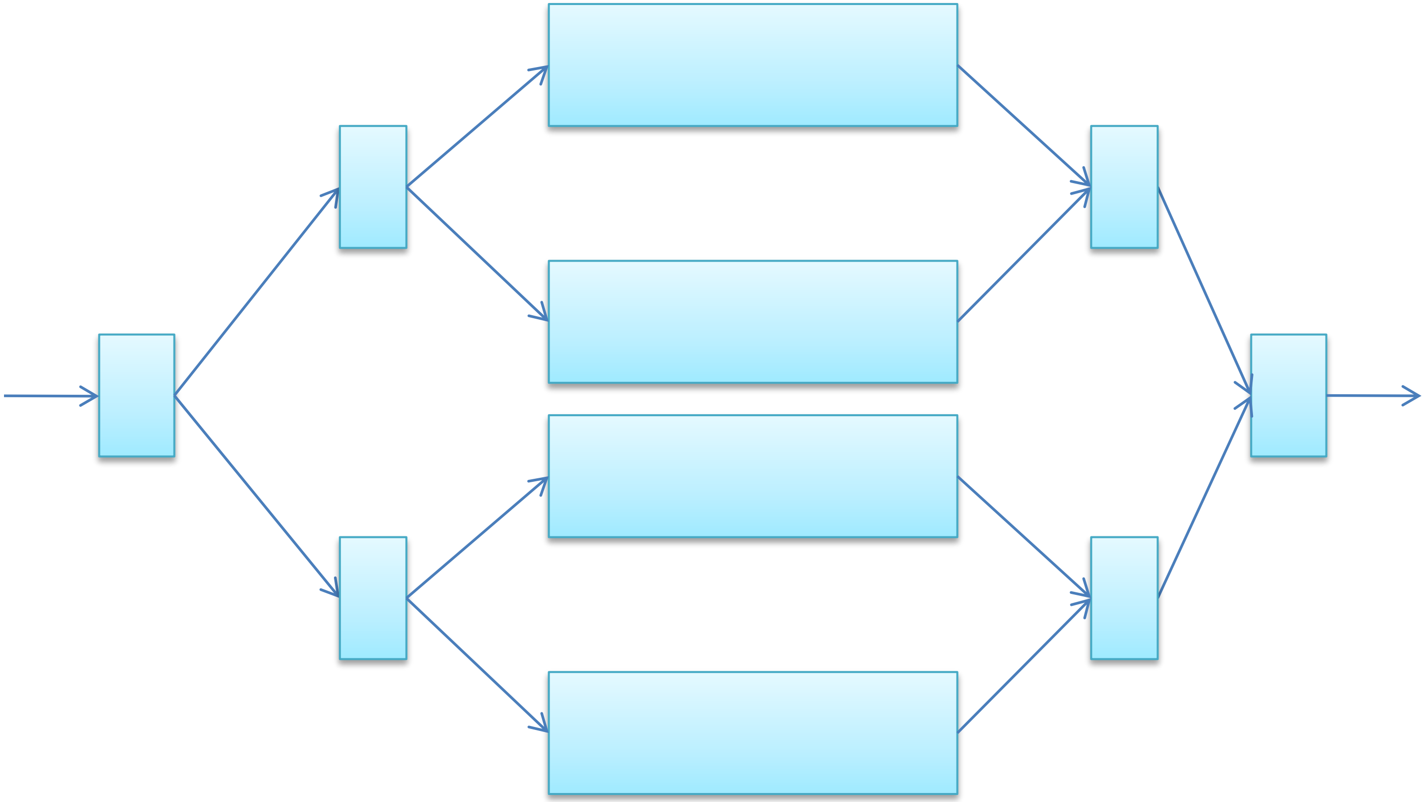
Parallel algorithms

Multi-processing hardware

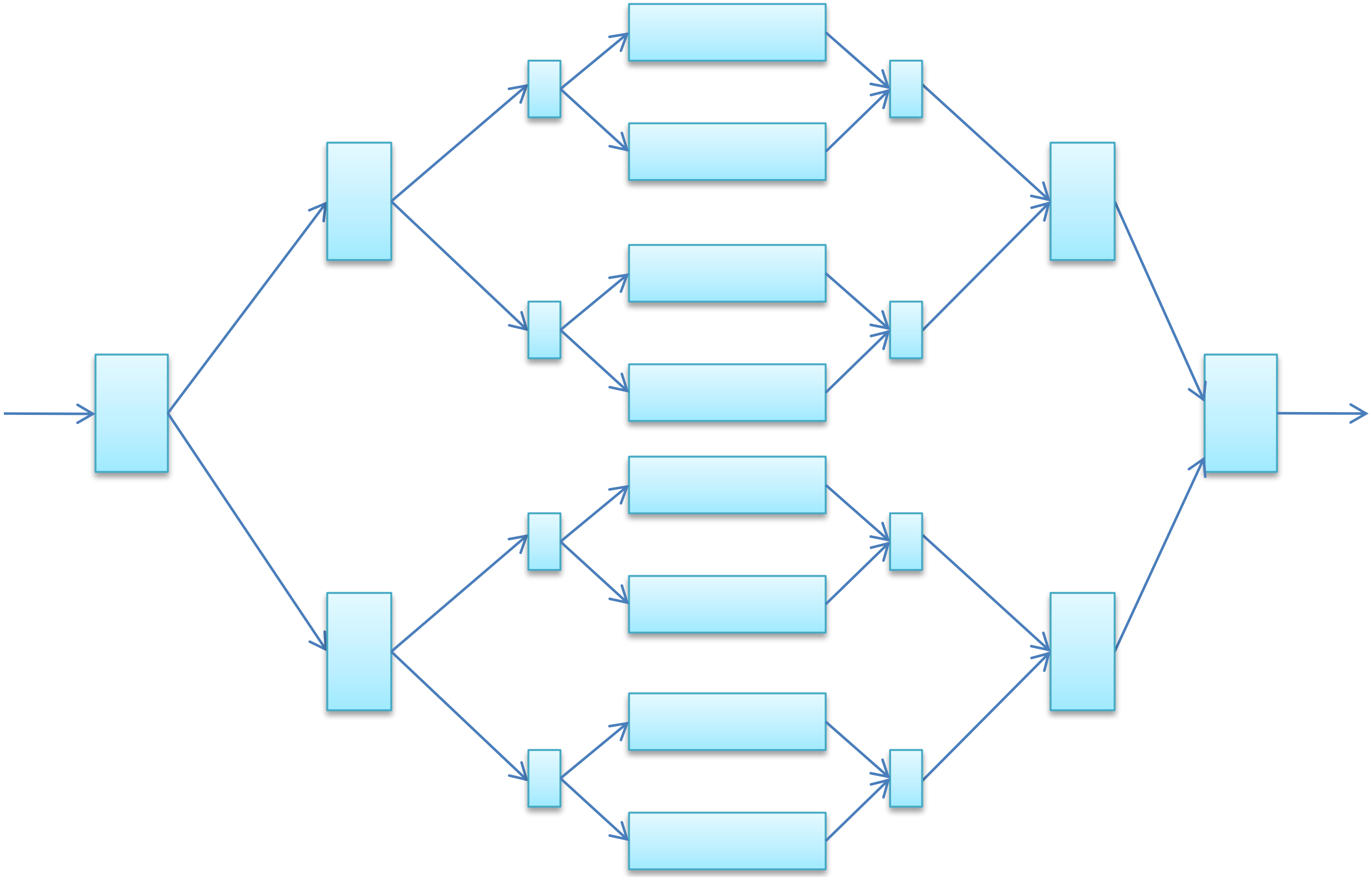
Merge sort (2-core)



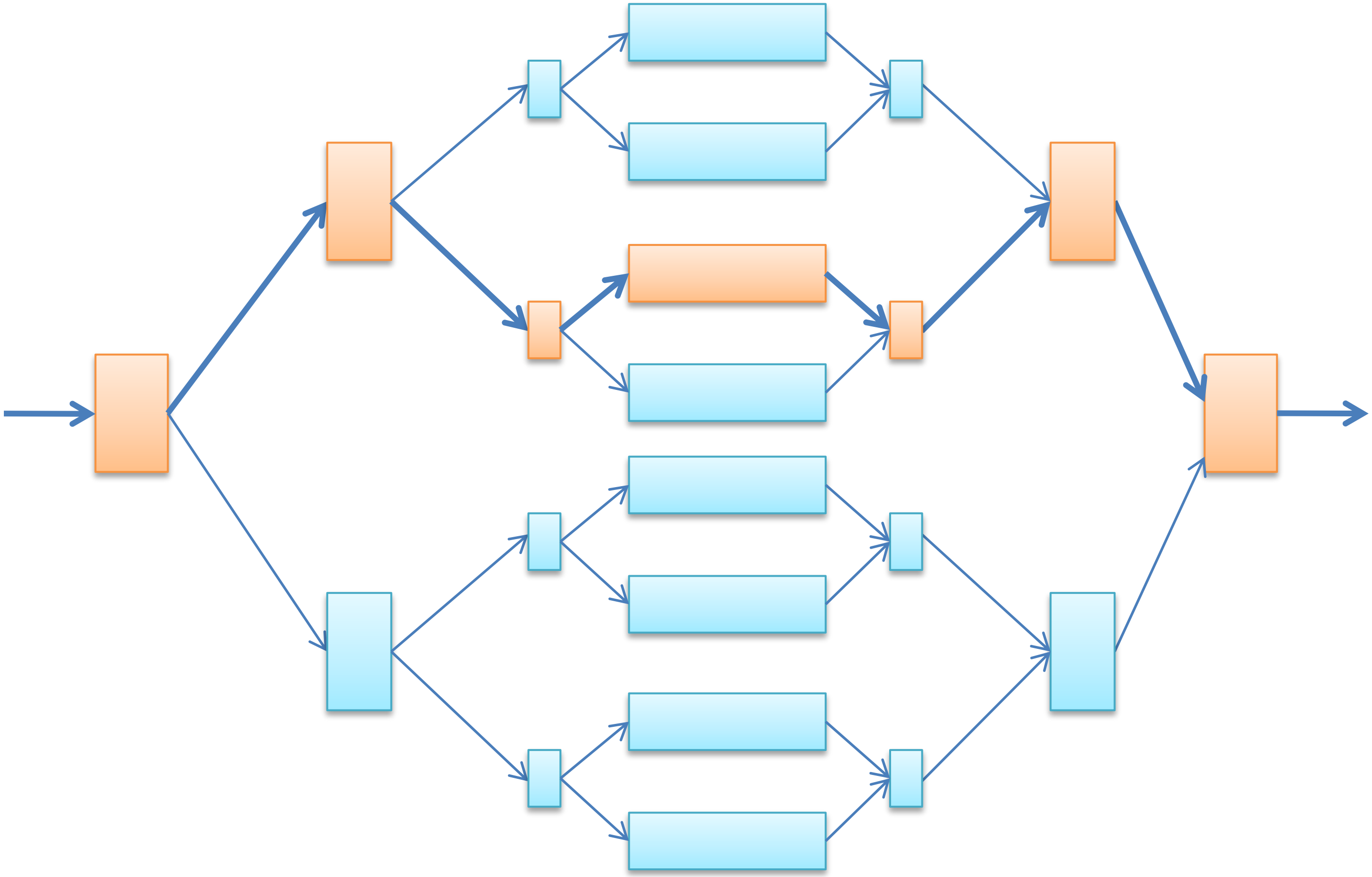
Merge sort (4-core)



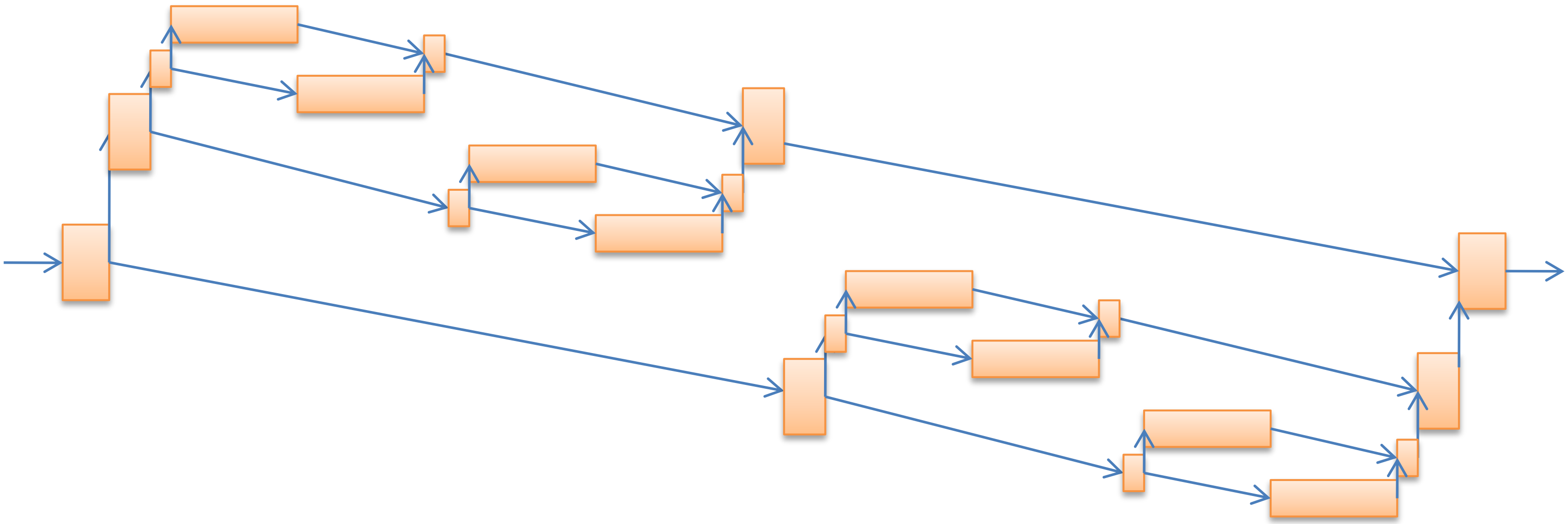
Merge sort (8-core)



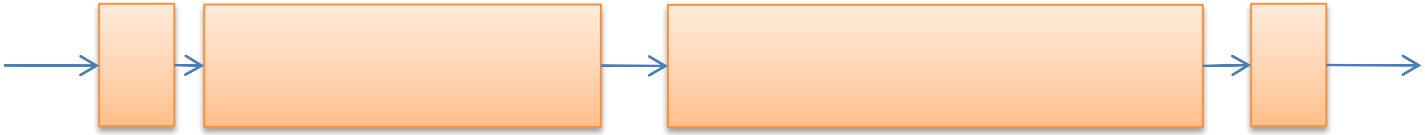
T_∞ (span): critical path length



T_1 (work): time to run sequentially



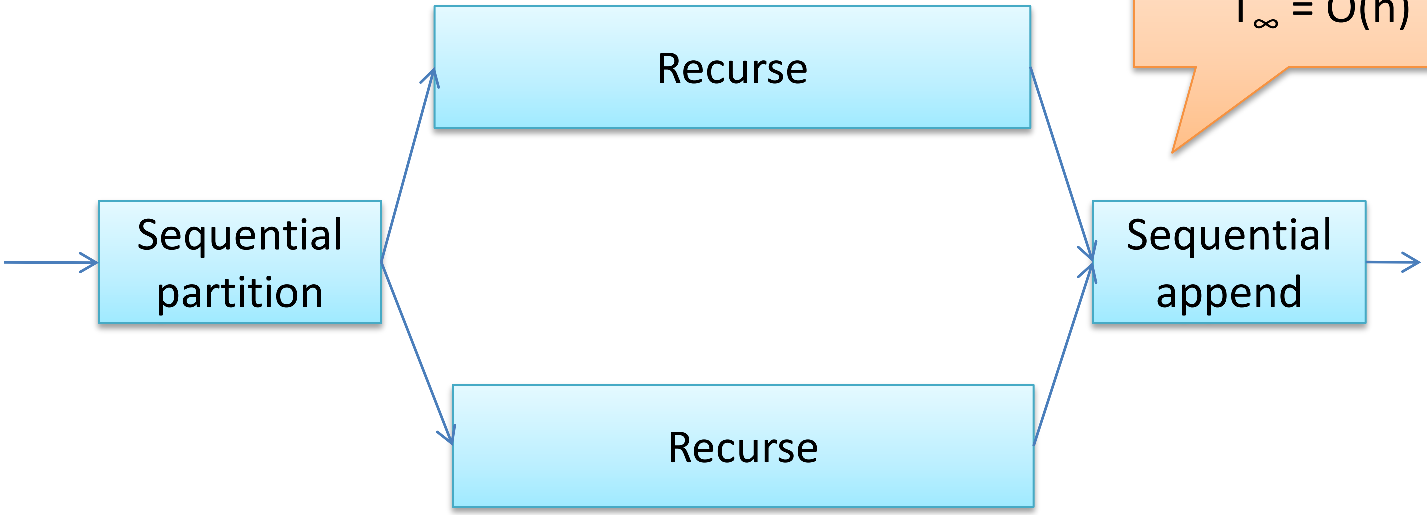
T_{serial}: optimized sequential code



A good multi-core parallel algorithm

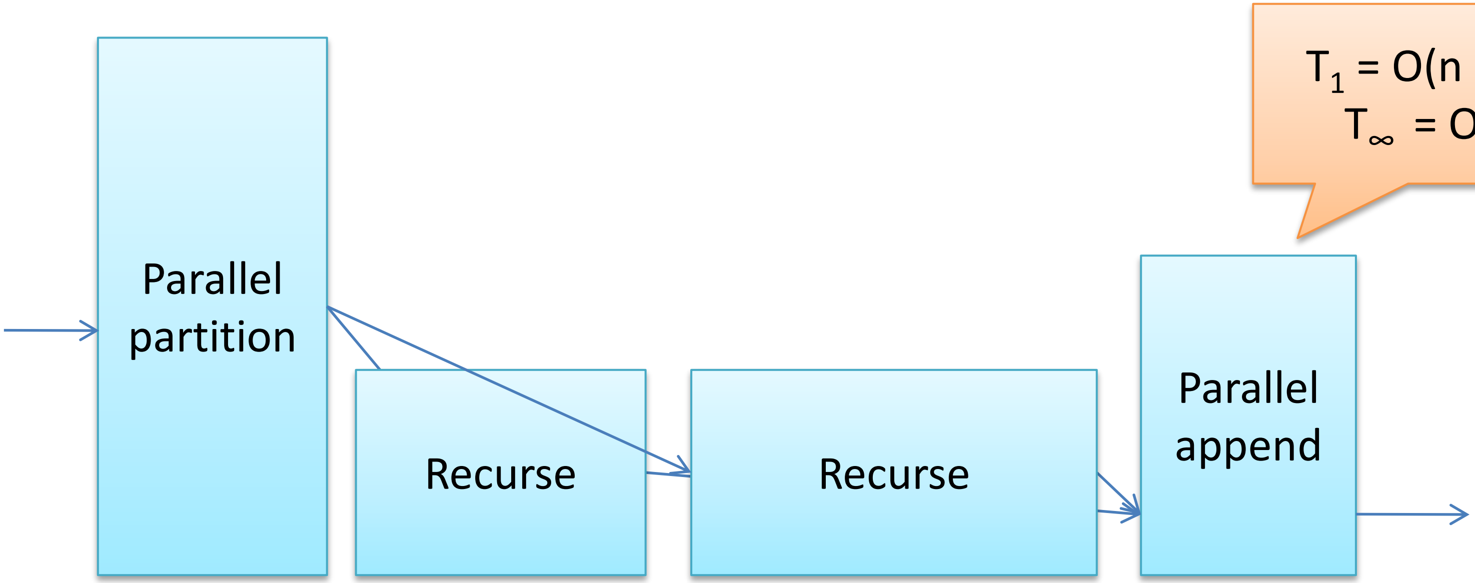
- T_1/T_{serial} is low
  - What we lose on sequential performance we must make up through parallelism
  - Resource availability may limit the ability to do that
- T_∞ grows slowly with the problem size
  - We tackle bigger problems by using more cores, not by running for longer

Quick-sort on "good" input



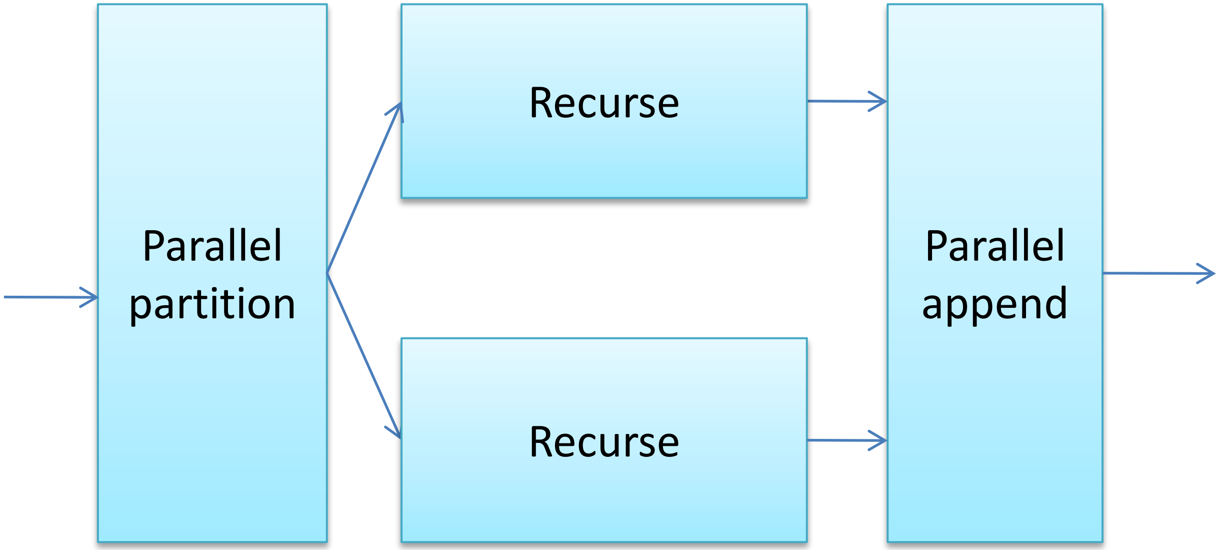
T_1 = O(n logn)
T_∞ = O(n)

Quick-sort on “good” input



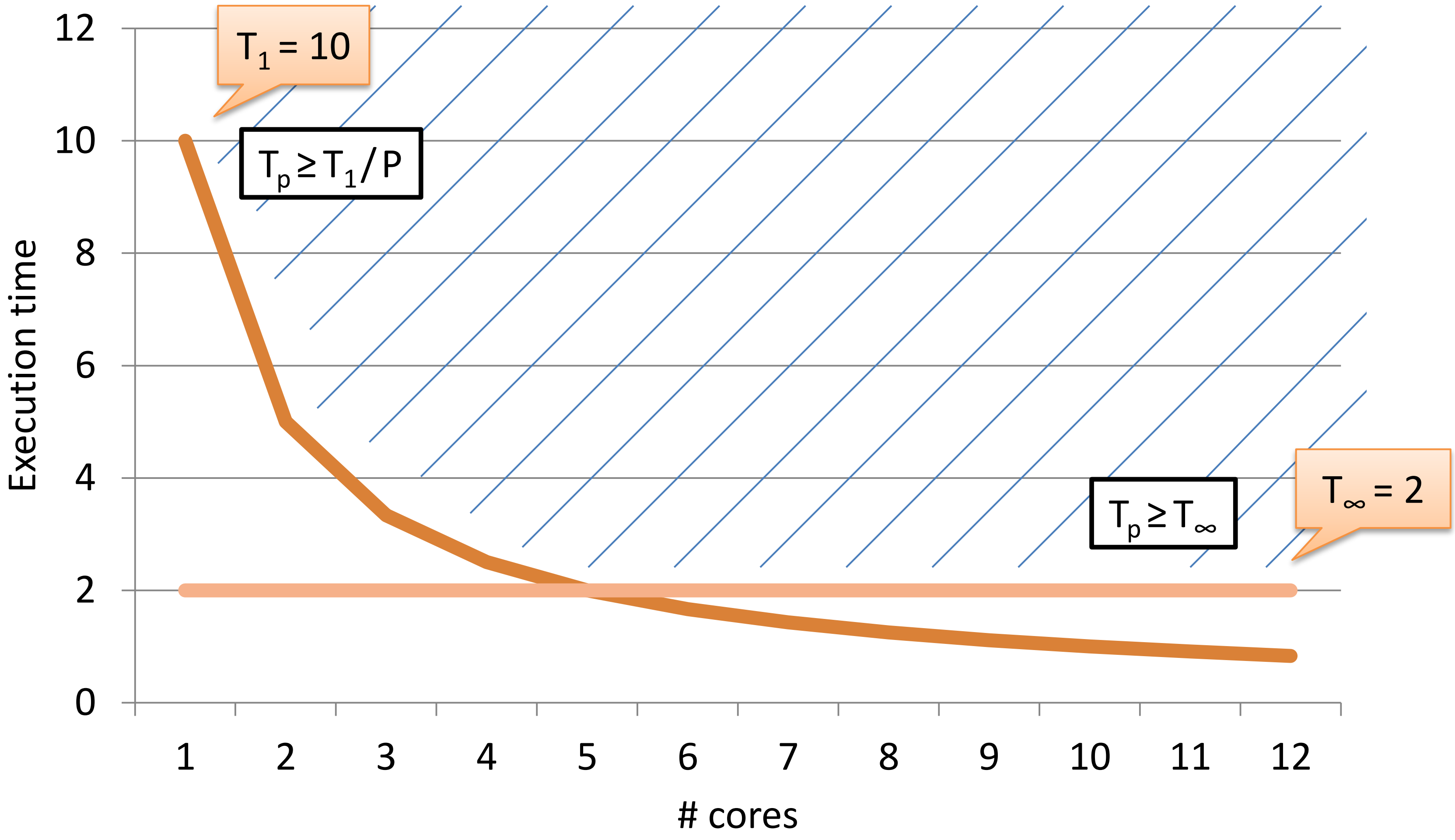
T_1 = O(n logn)
T_∞ = O(n)

Quick-sort on “good” input

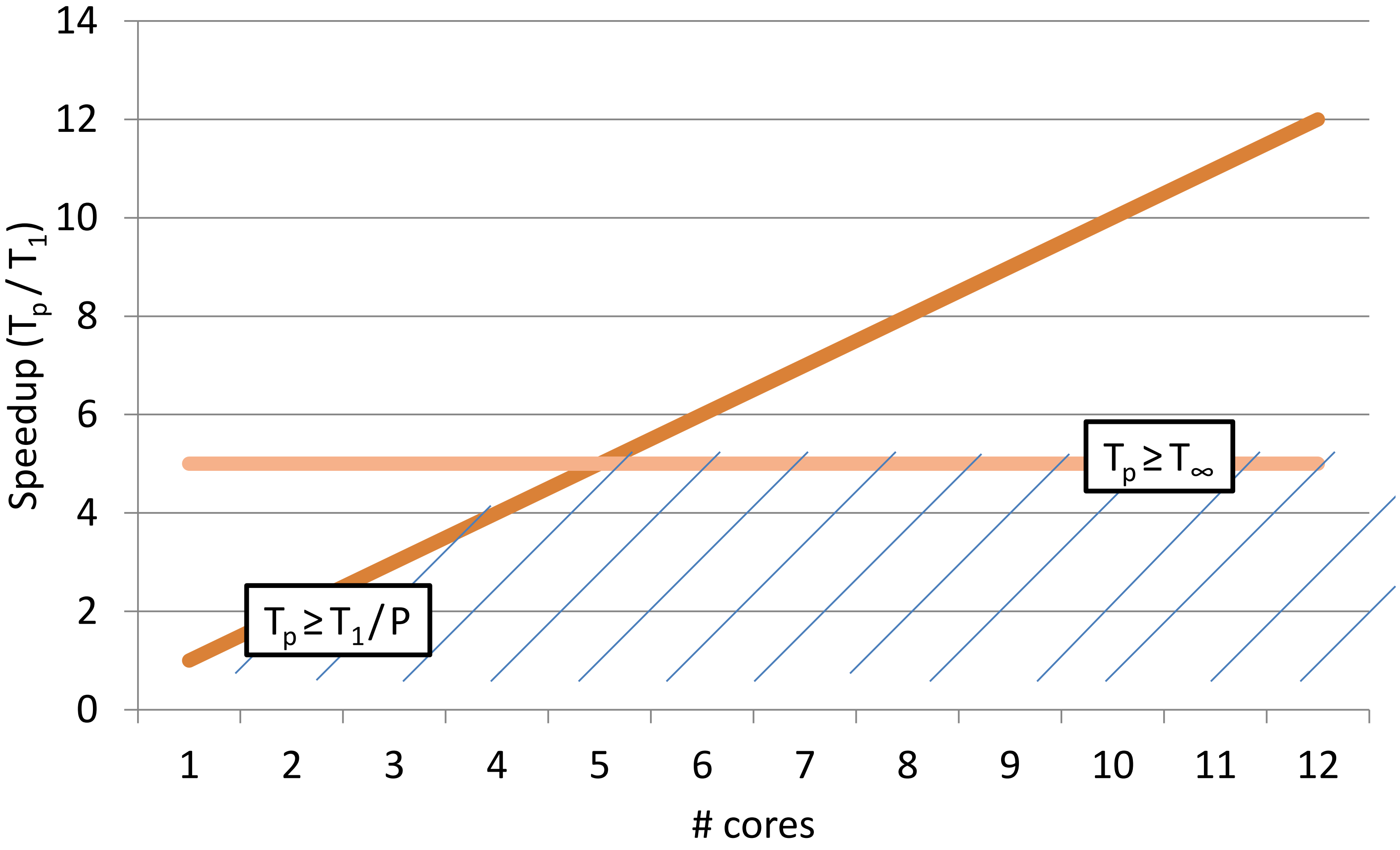


$T_1 = O(n \log n)$
$T_\infty = O(\log^2 n)$

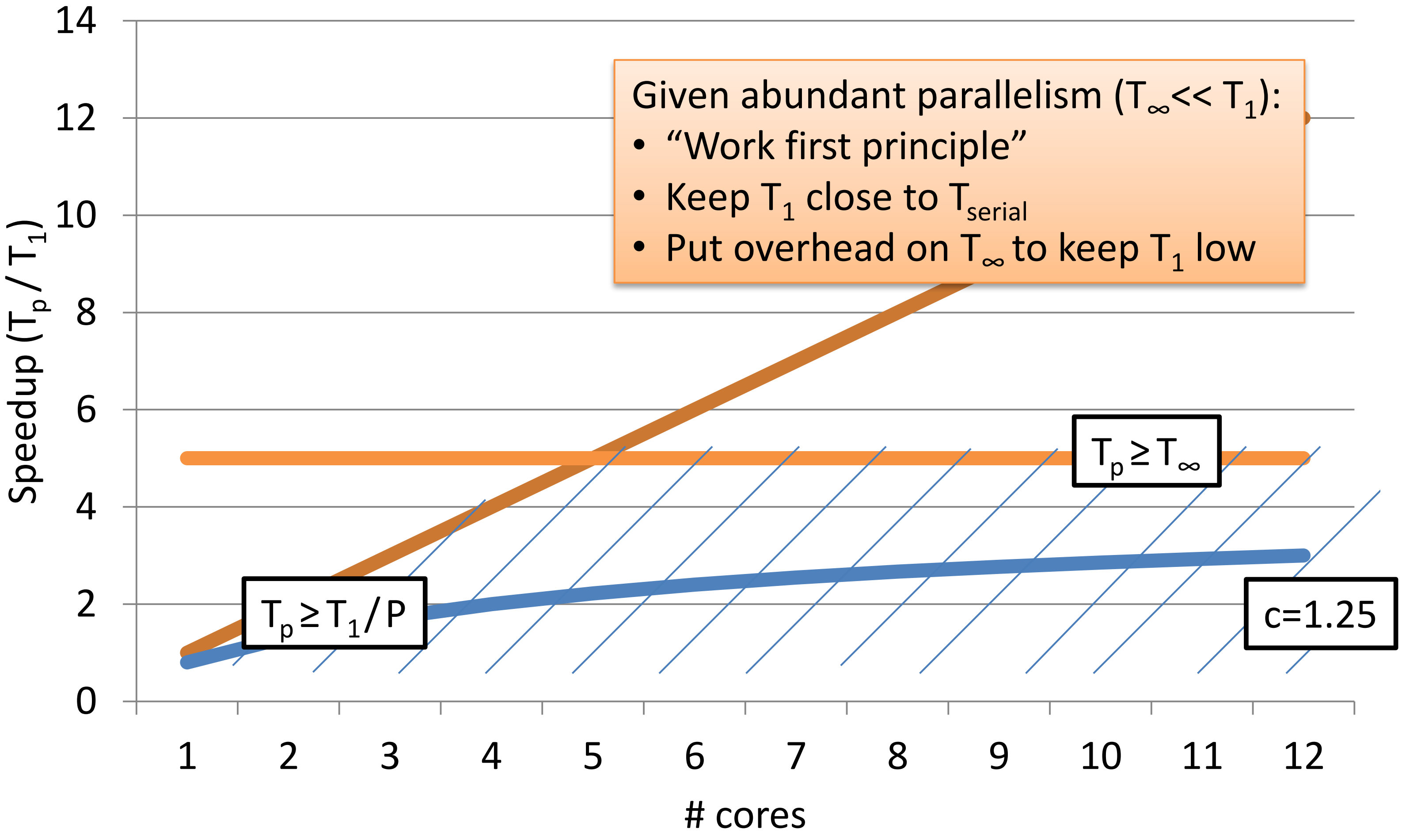
Scheduling from a DAG



Scheduling from a DAG



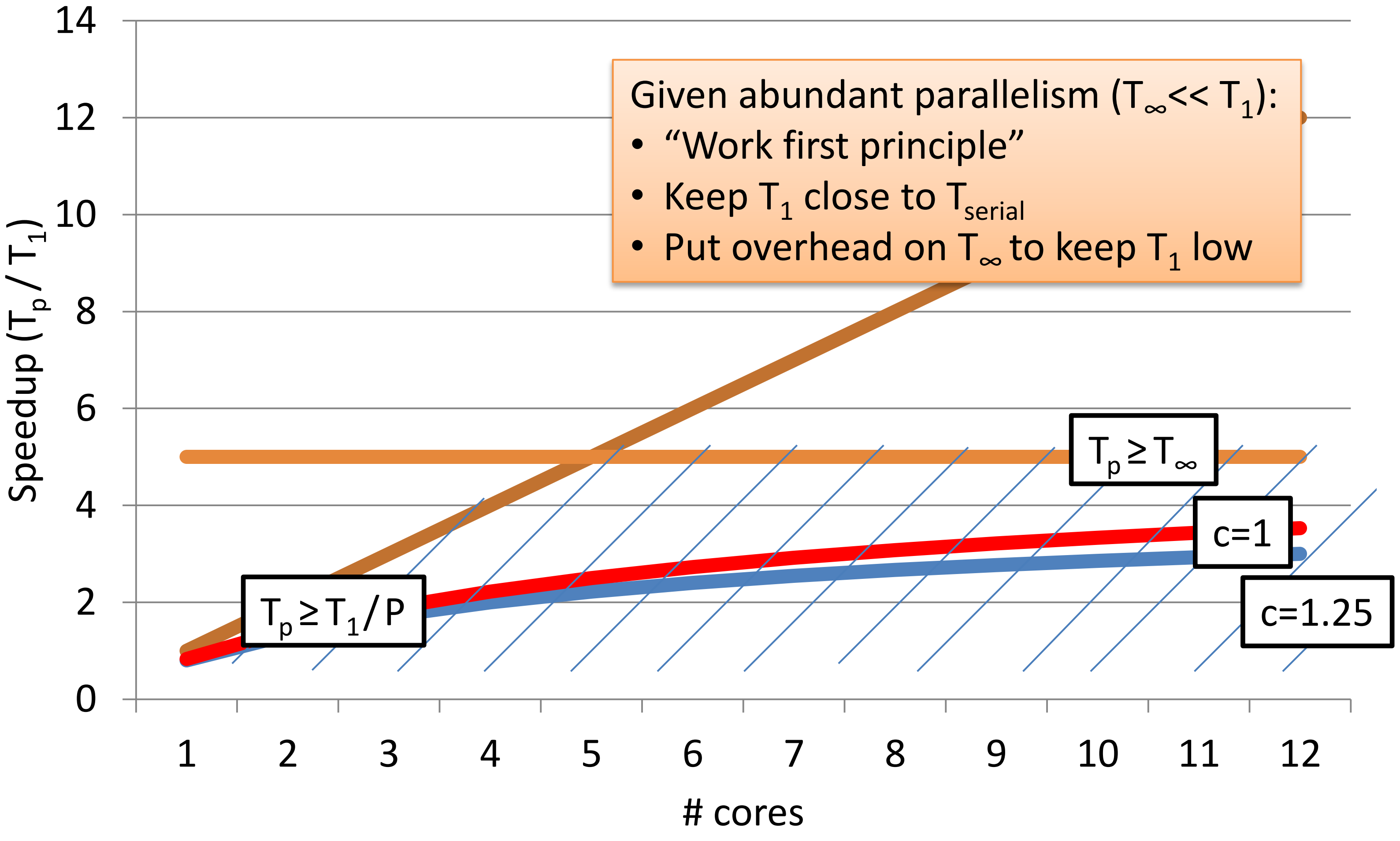
In CILK: $T_p \approx T_1 / p + c\,T_\infty$



Given abundant parallelism (T_∞<< T_1):

- “Work first principle”
- Keep T_1 close to T_{serial}
- Put overhead on T_∞ to keep T_1 low

In CILK: $T_p \approx T_1 / p + c\,T_\infty$



Given abundant parallelism ($T_\infty << T_1$):

- "Work first principle"
- Keep T_1 close to T_{serial}
- Put overhead on T_∞ to keep T_1 low

Why parallelism?

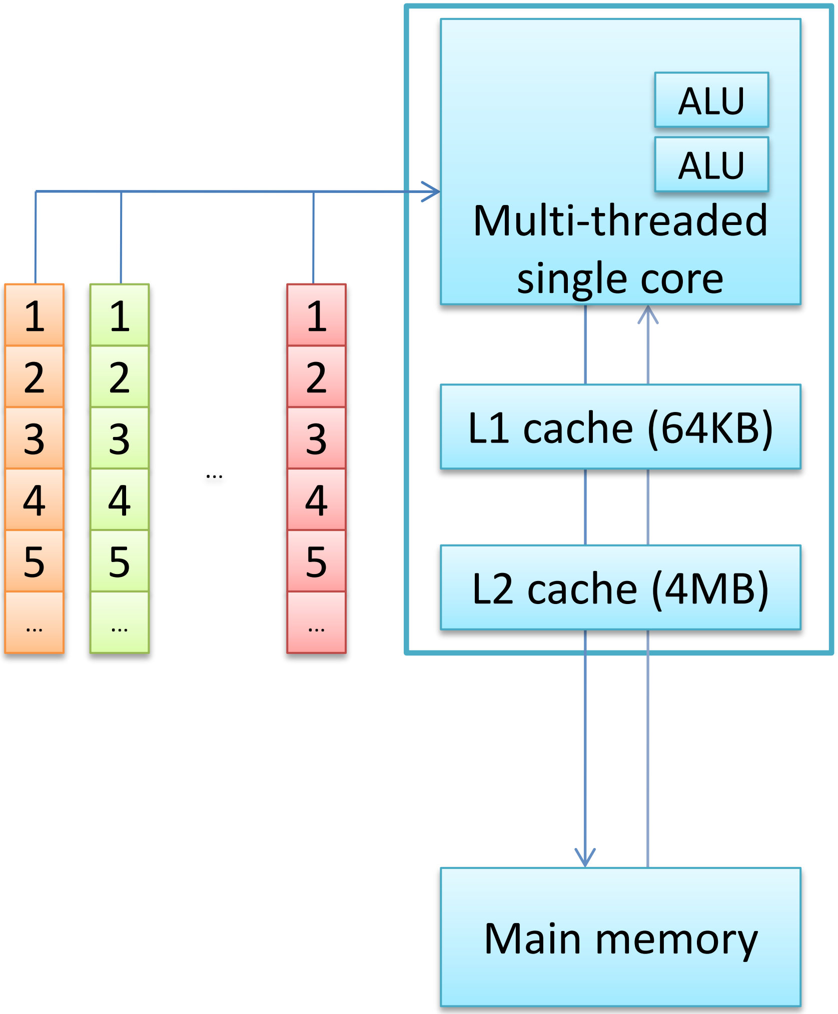
Amdahl’s law

Why asymmetric performance?

Parallel algorithms

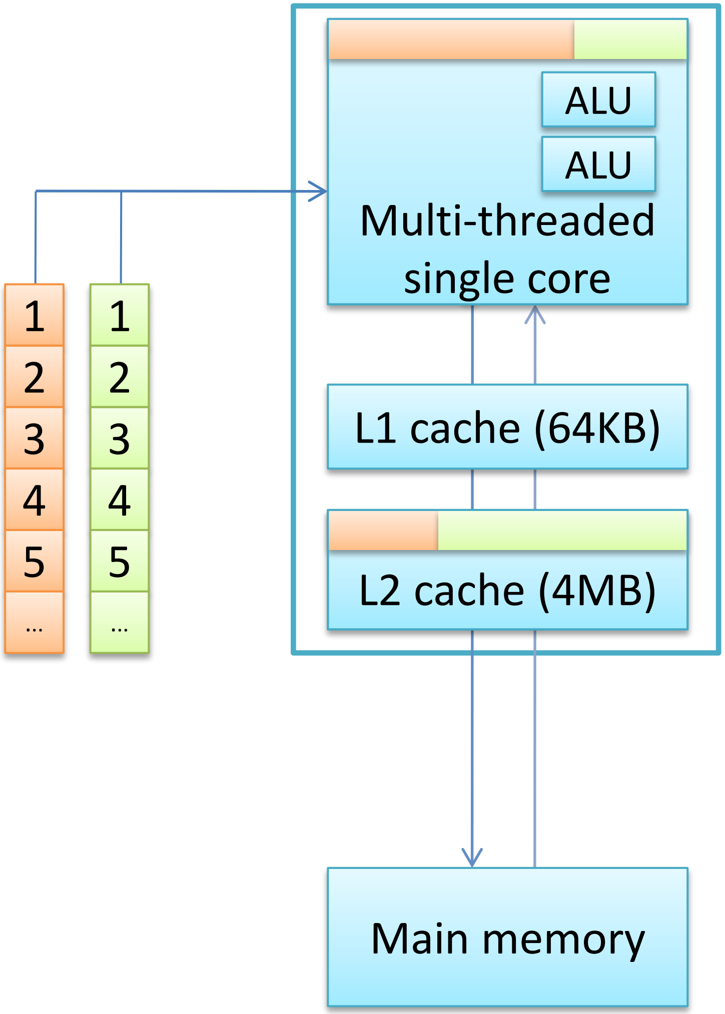
Multi-processing hardware

Multi-threaded h/w



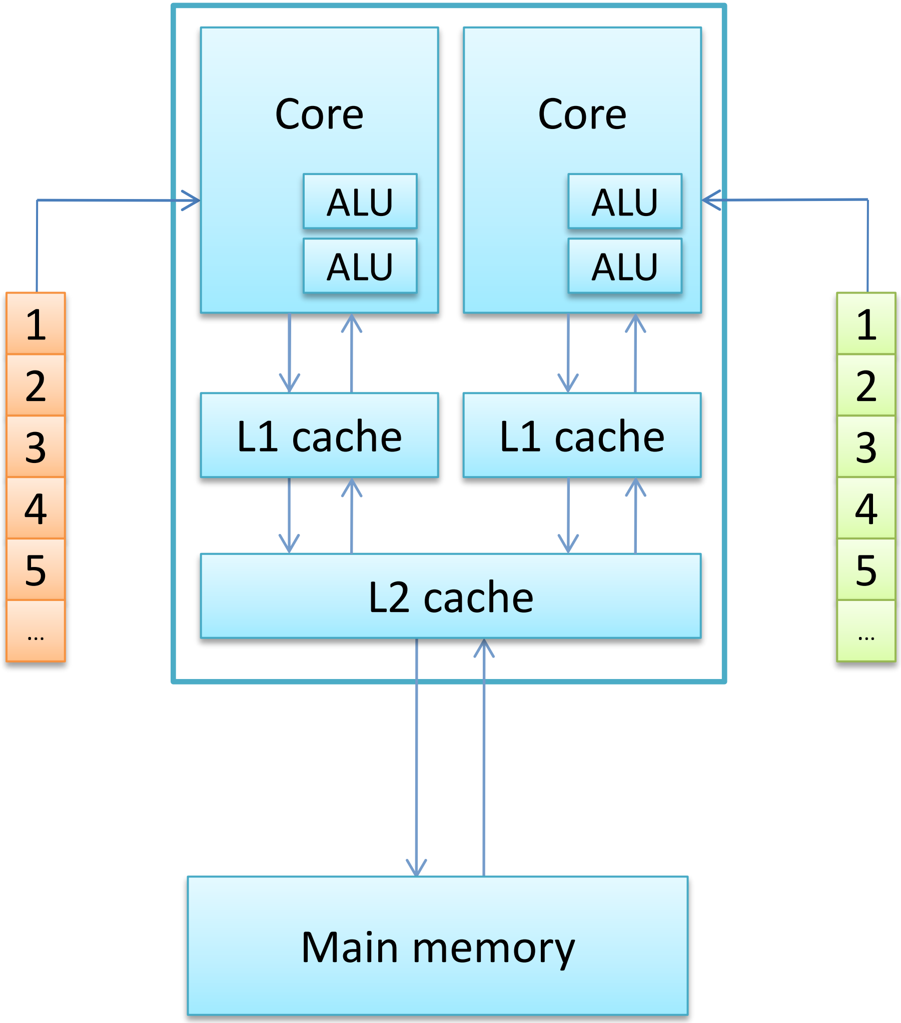
- Multiple threads in a workload with:
  - Poor spatial locality
  - Frequent memory accesses

Multi-threaded h/w

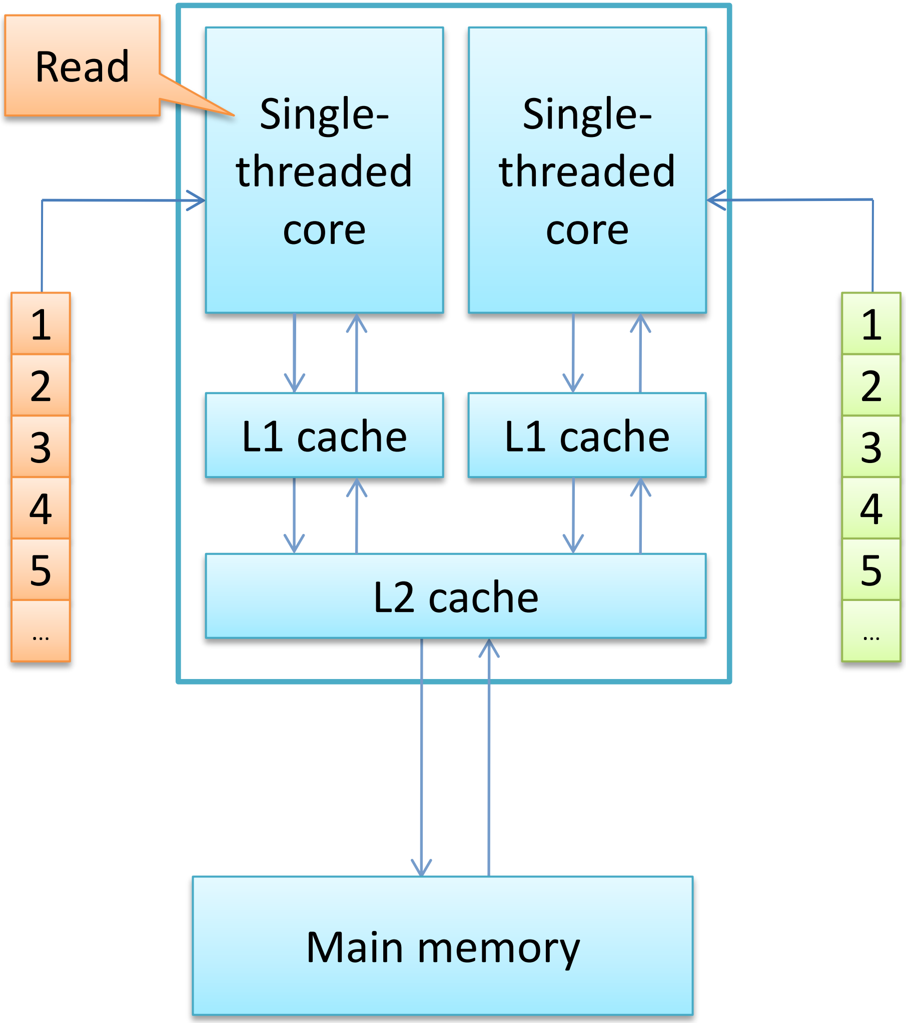


- Multiple threads with synergistic resource needs

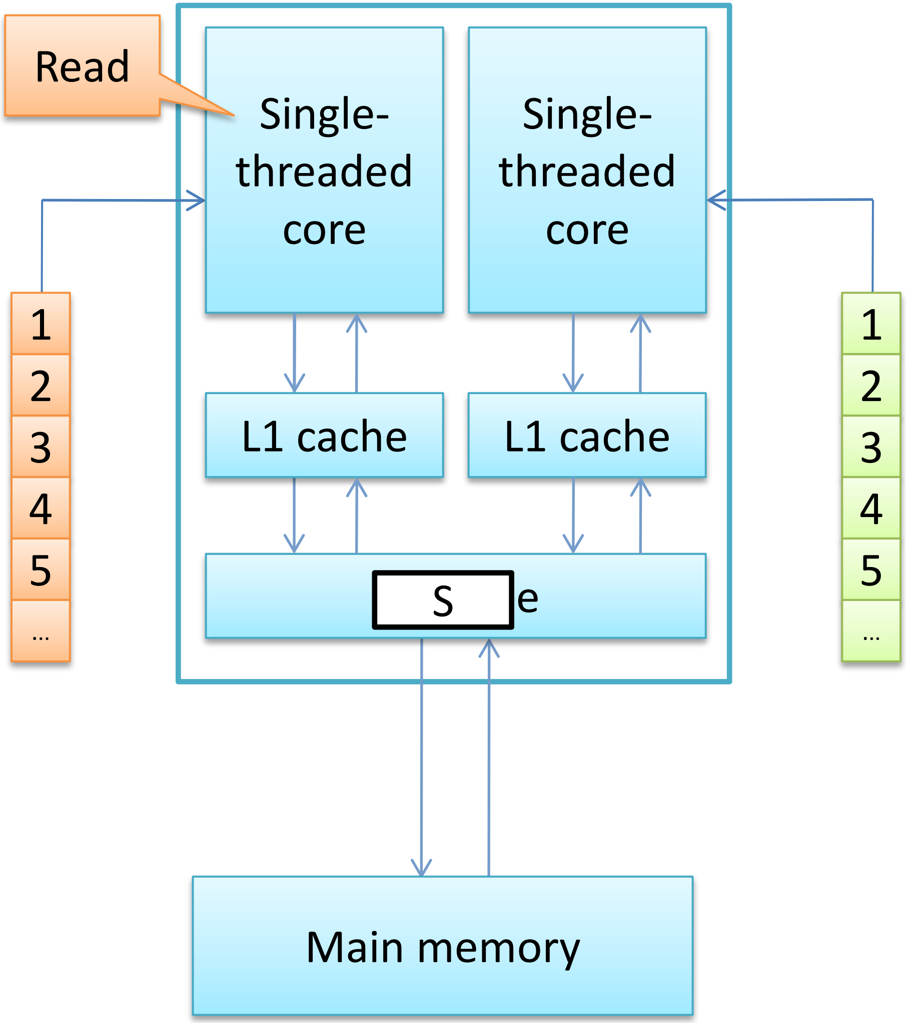
Multi-core h/w – common L2



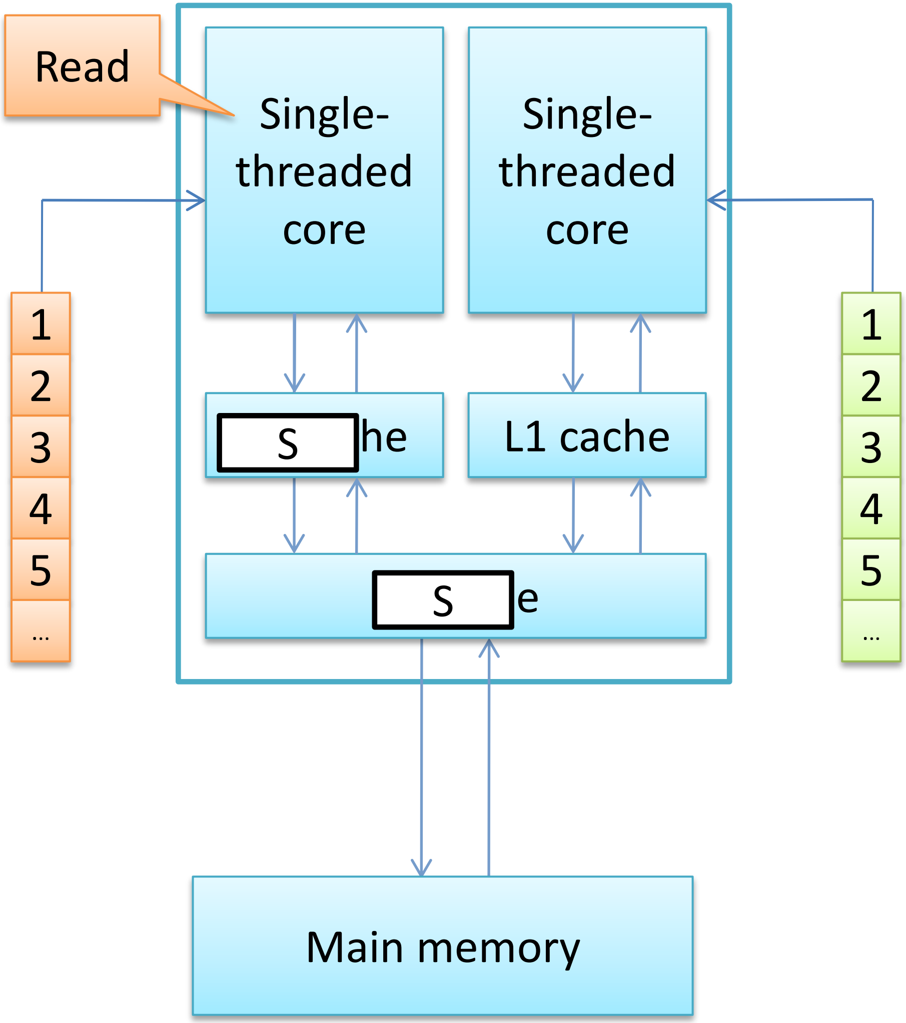
Multi-core h/w – common L2



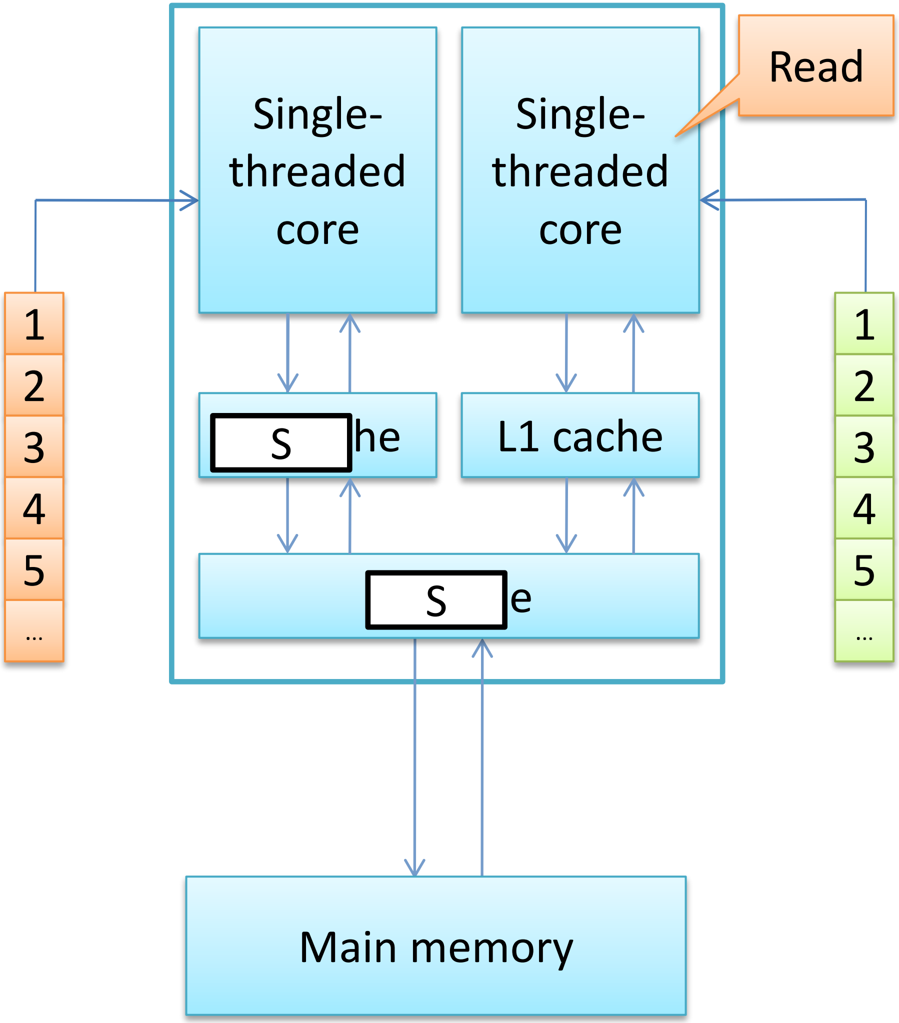
Multi-core h/w – common L2



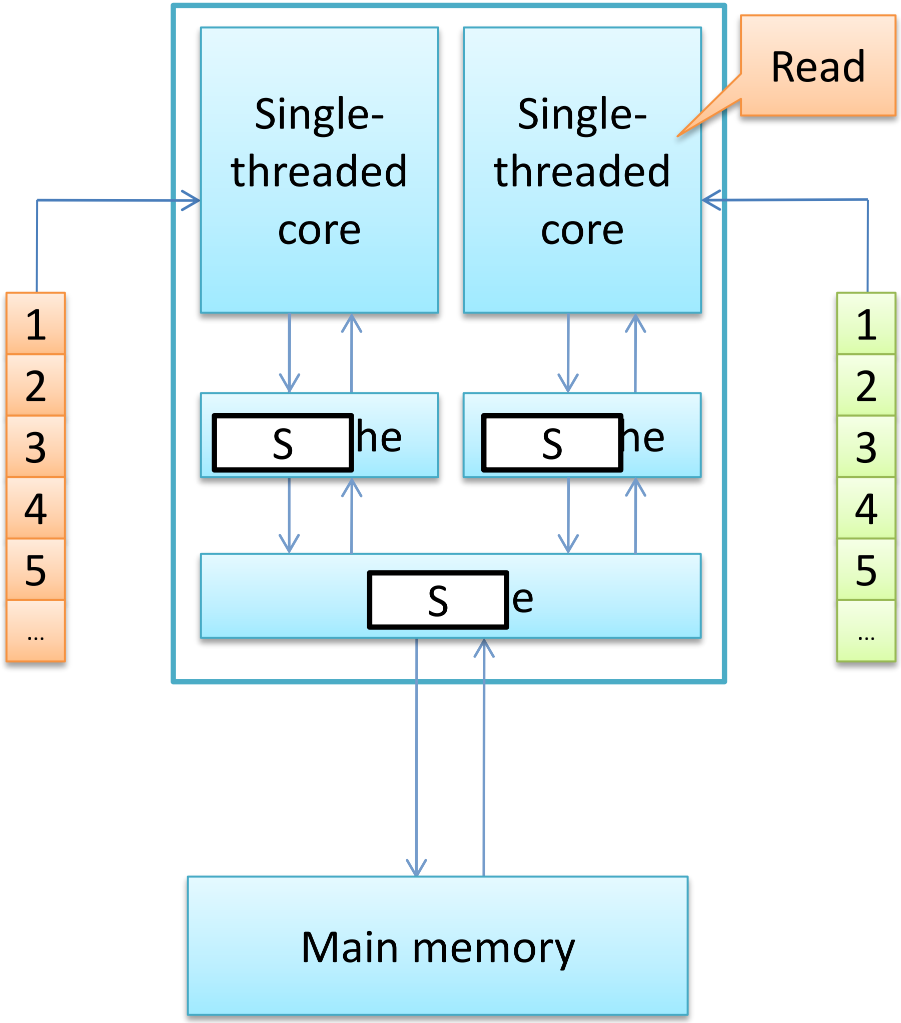
Multi-core h/w – common L2



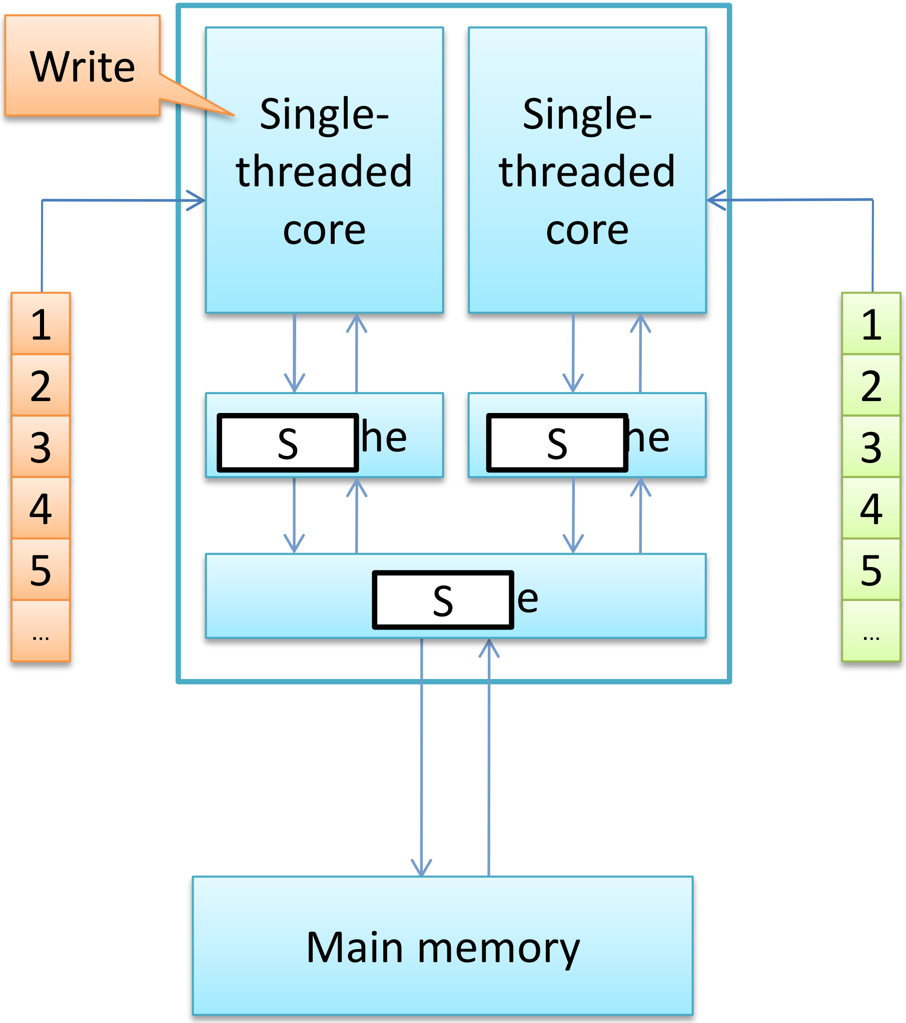
Multi-core h/w – common L2



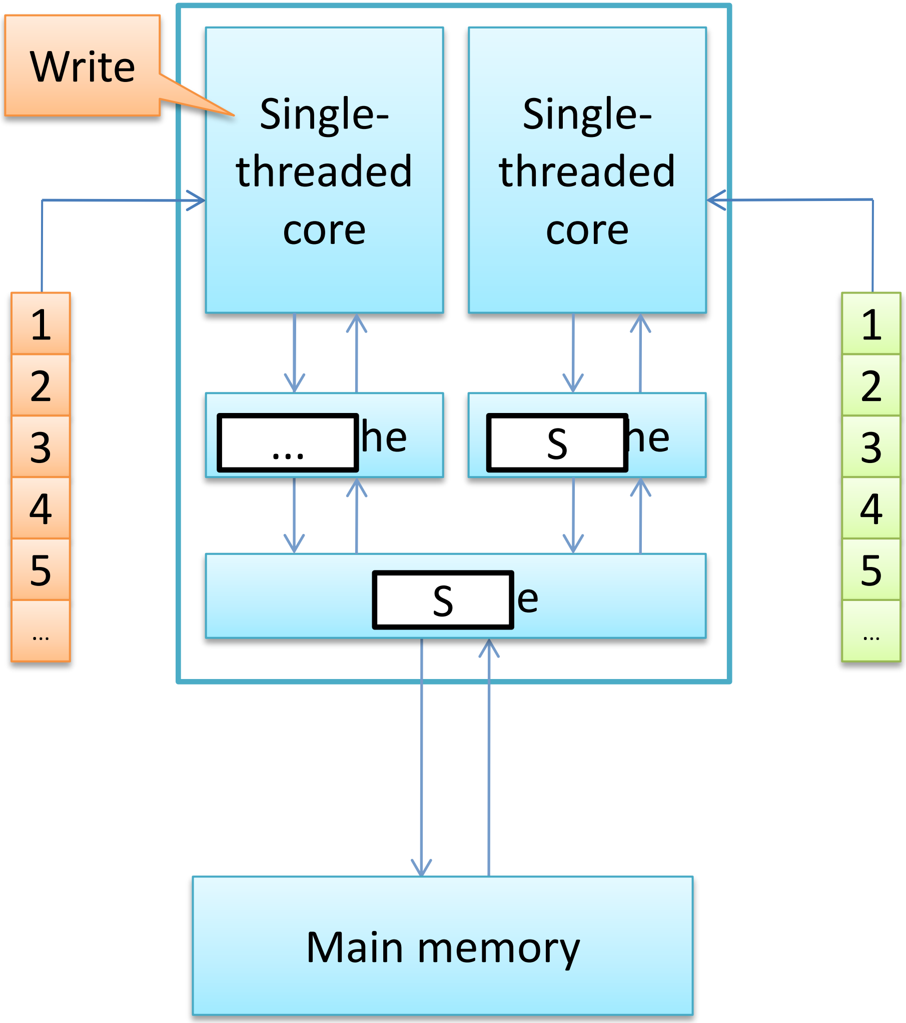
Multi-core h/w – common L2



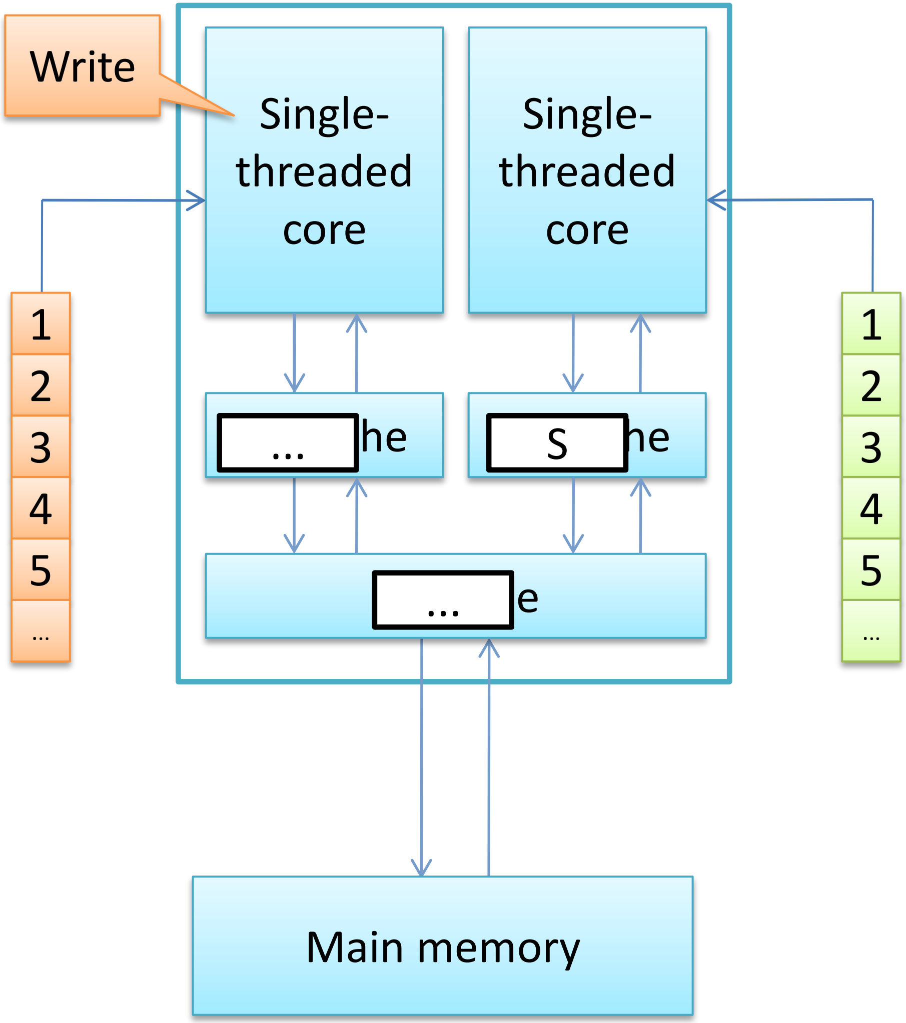
Multi-core h/w – common L2



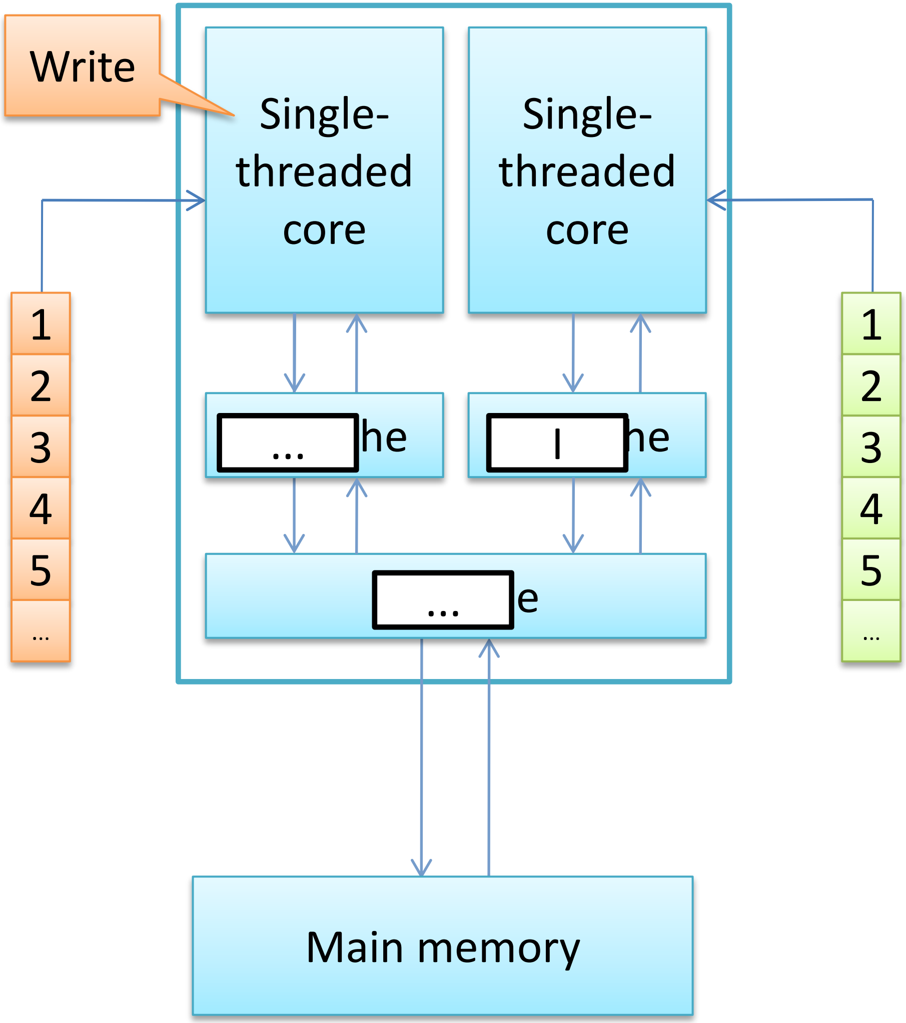
Multi-core h/w – common L2



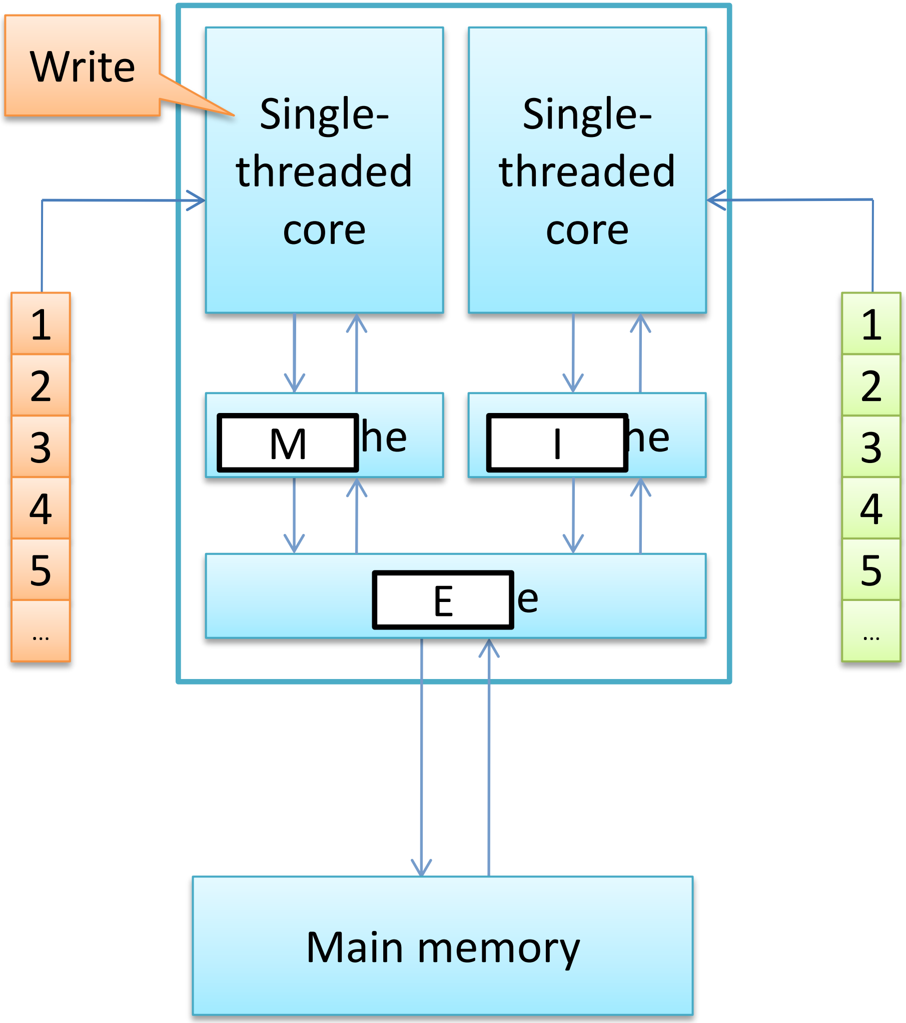
Multi-core h/w – common L2



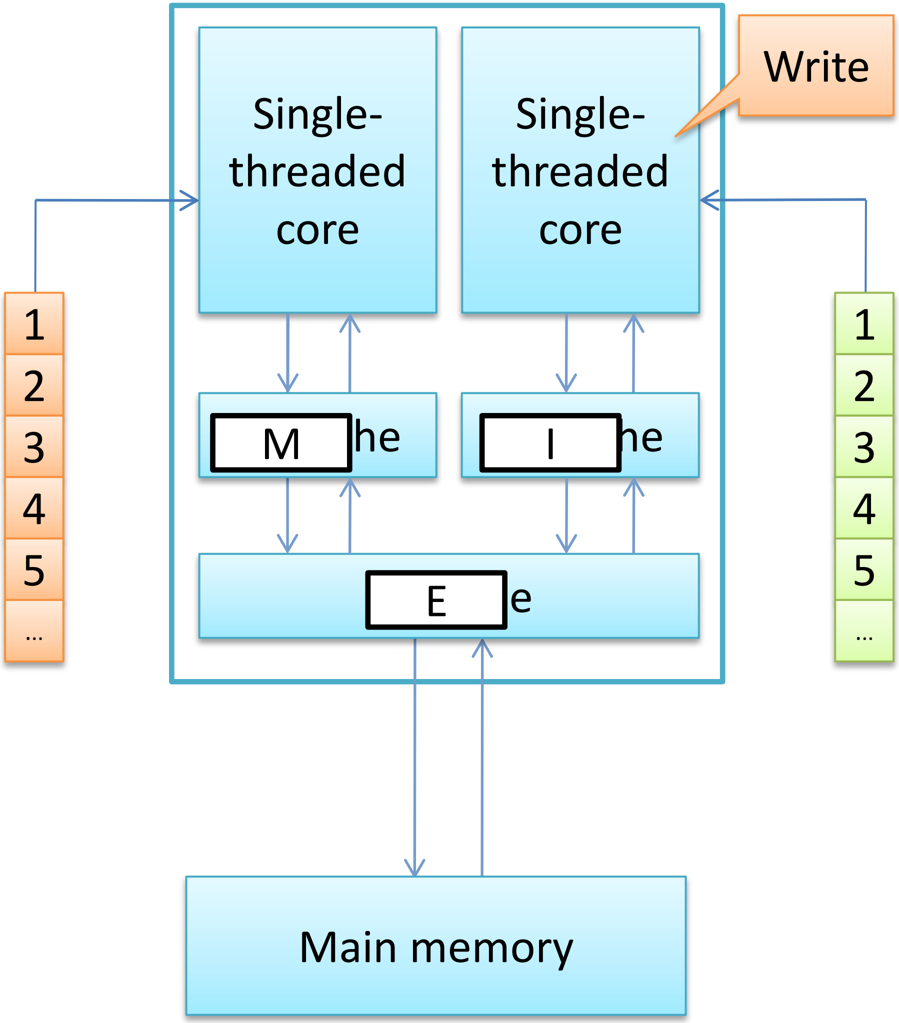
Multi-core h/w – common L2



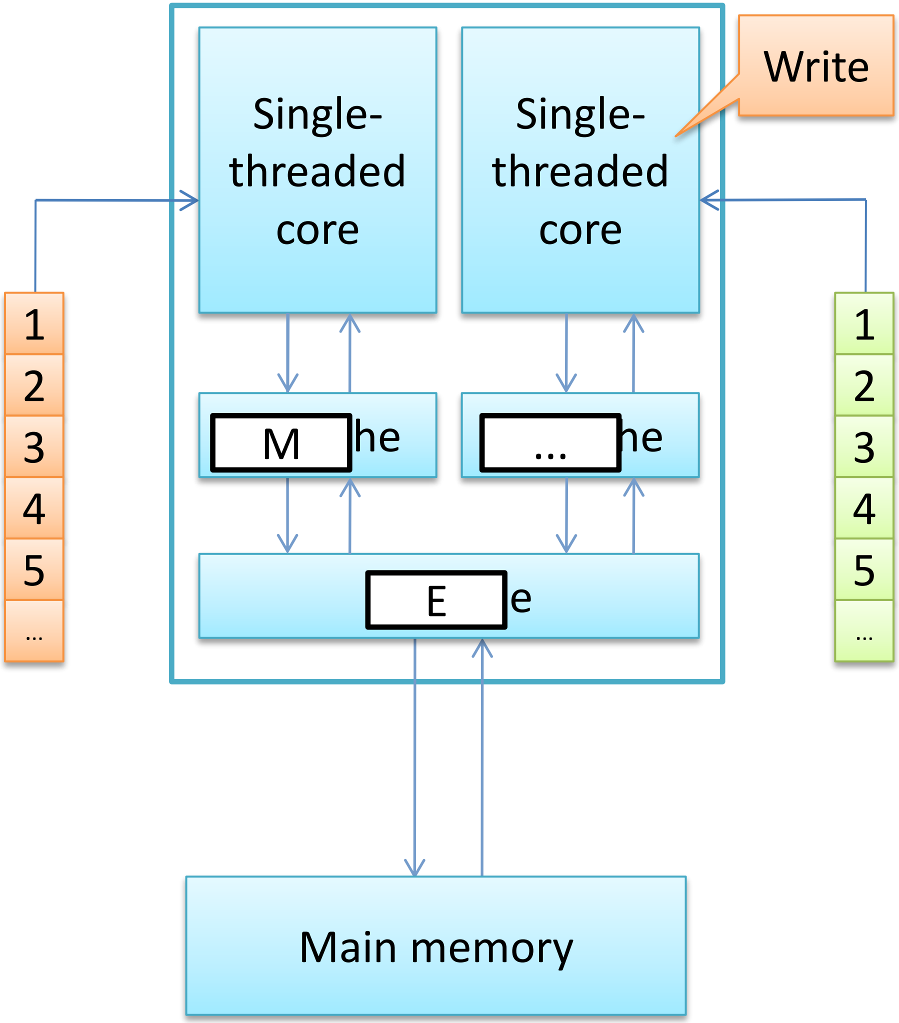
Multi-core h/w – common L2



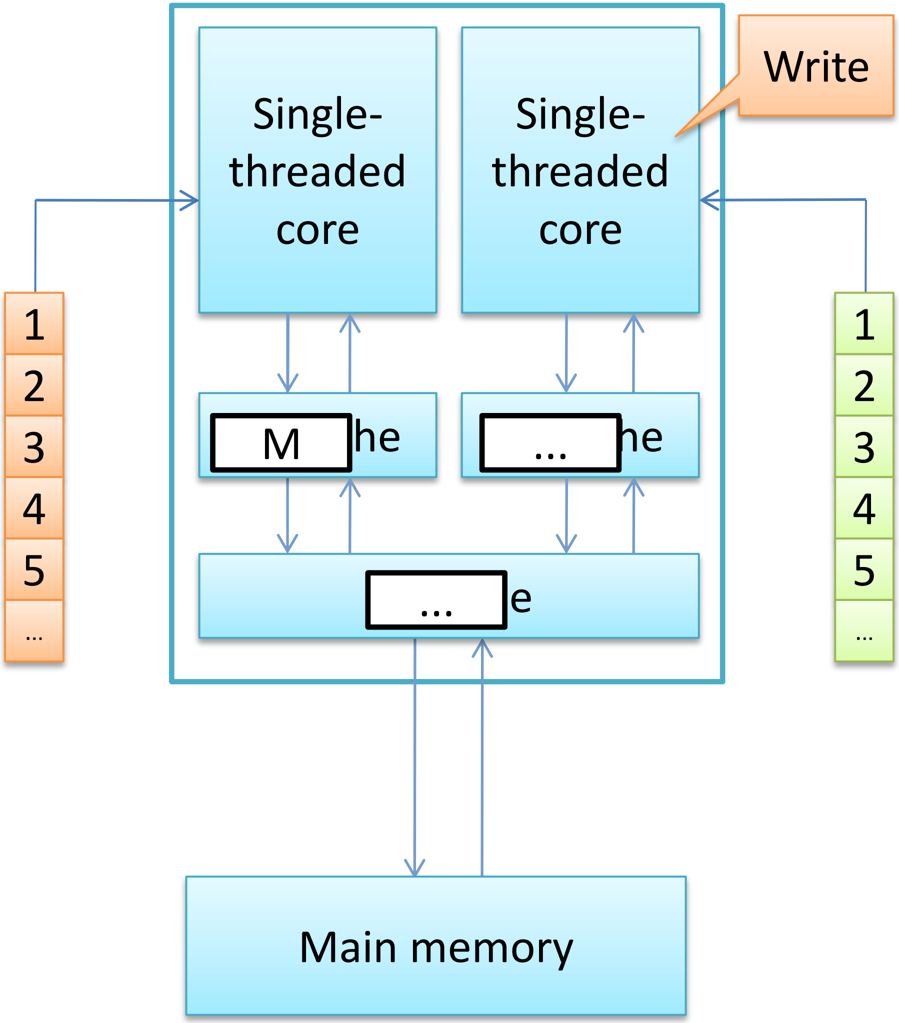
Multi-core h/w – common L2



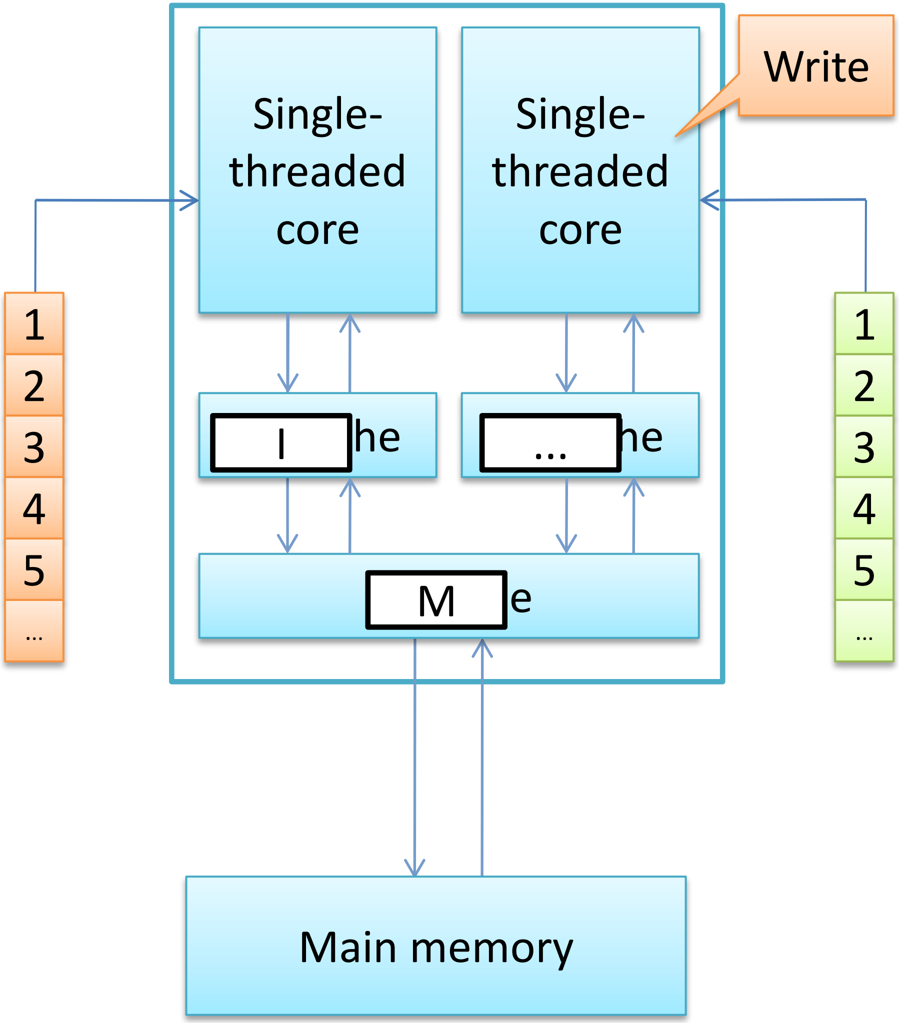
Multi-core h/w – common L2



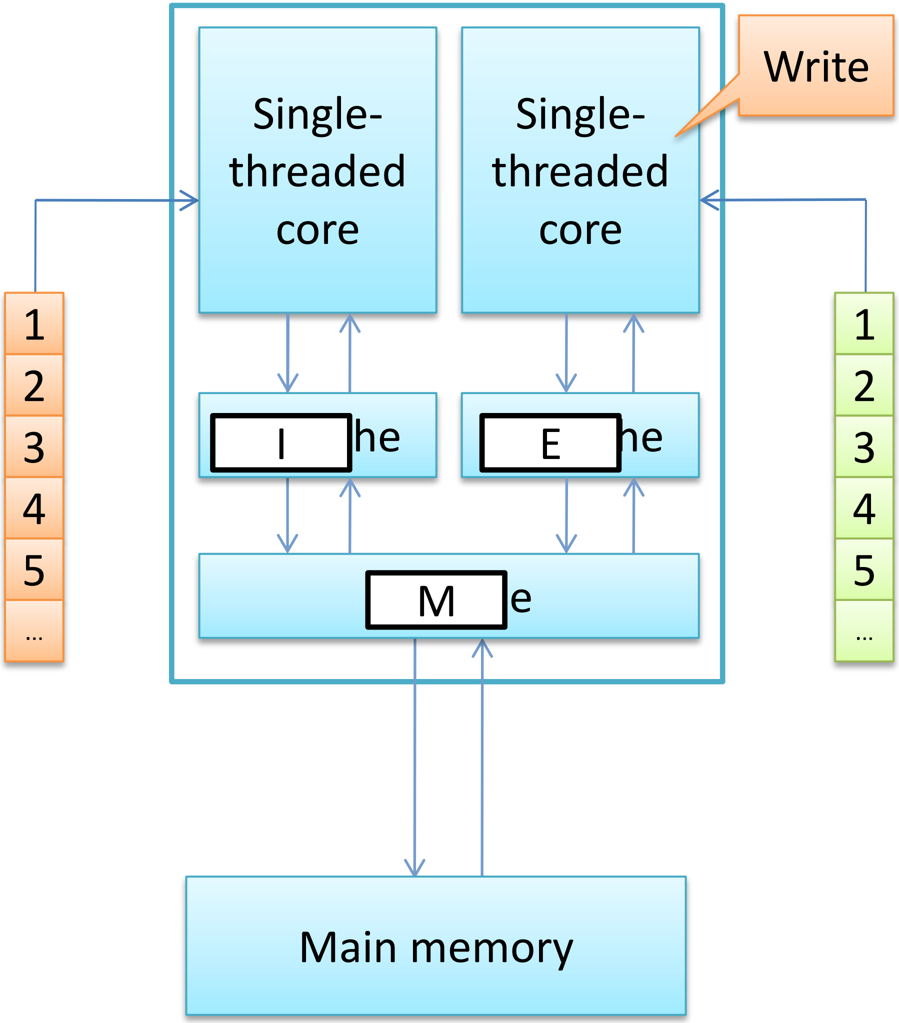
Multi-core h/w – common L2



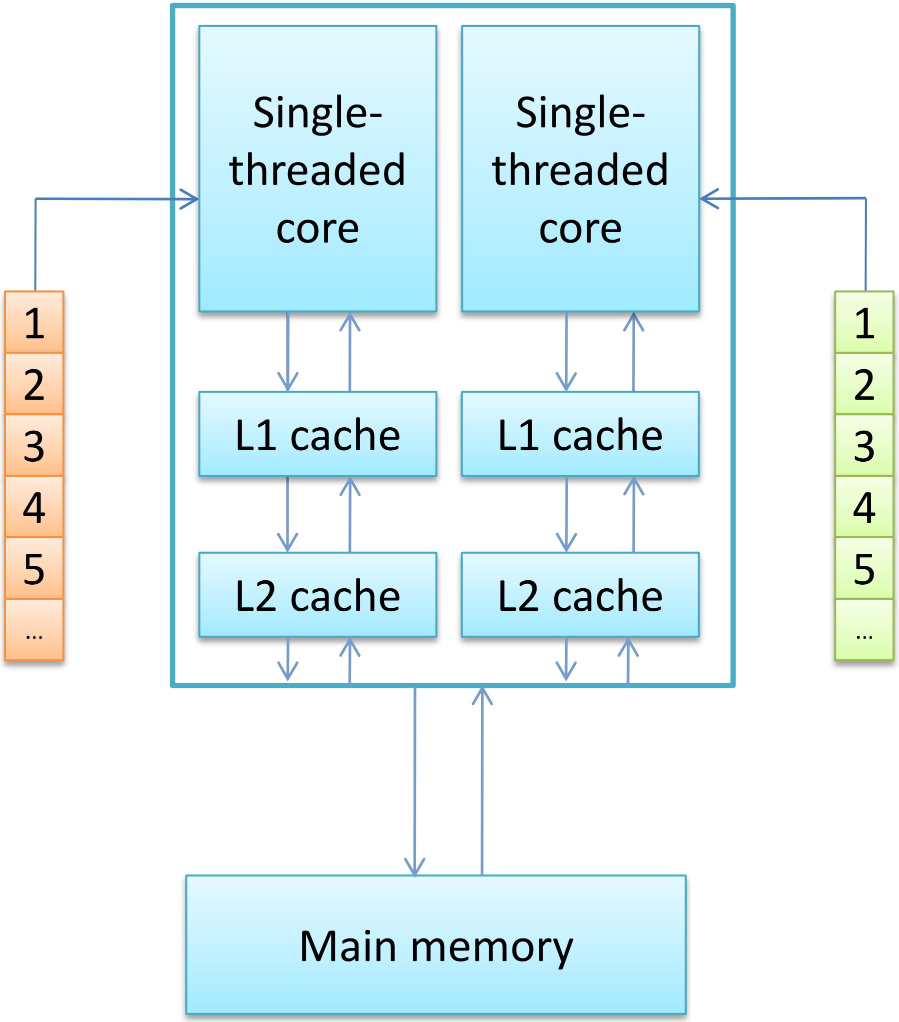
Multi-core h/w – common L2



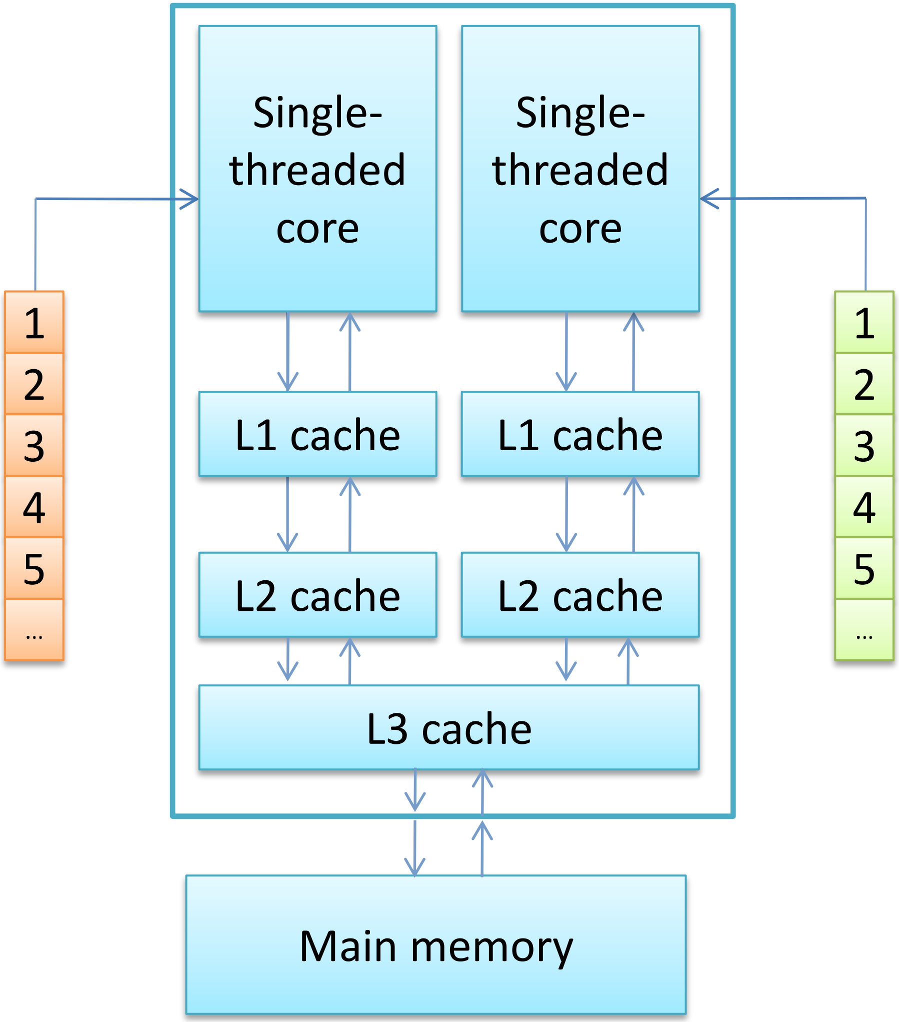
Multi-core h/w – common L2



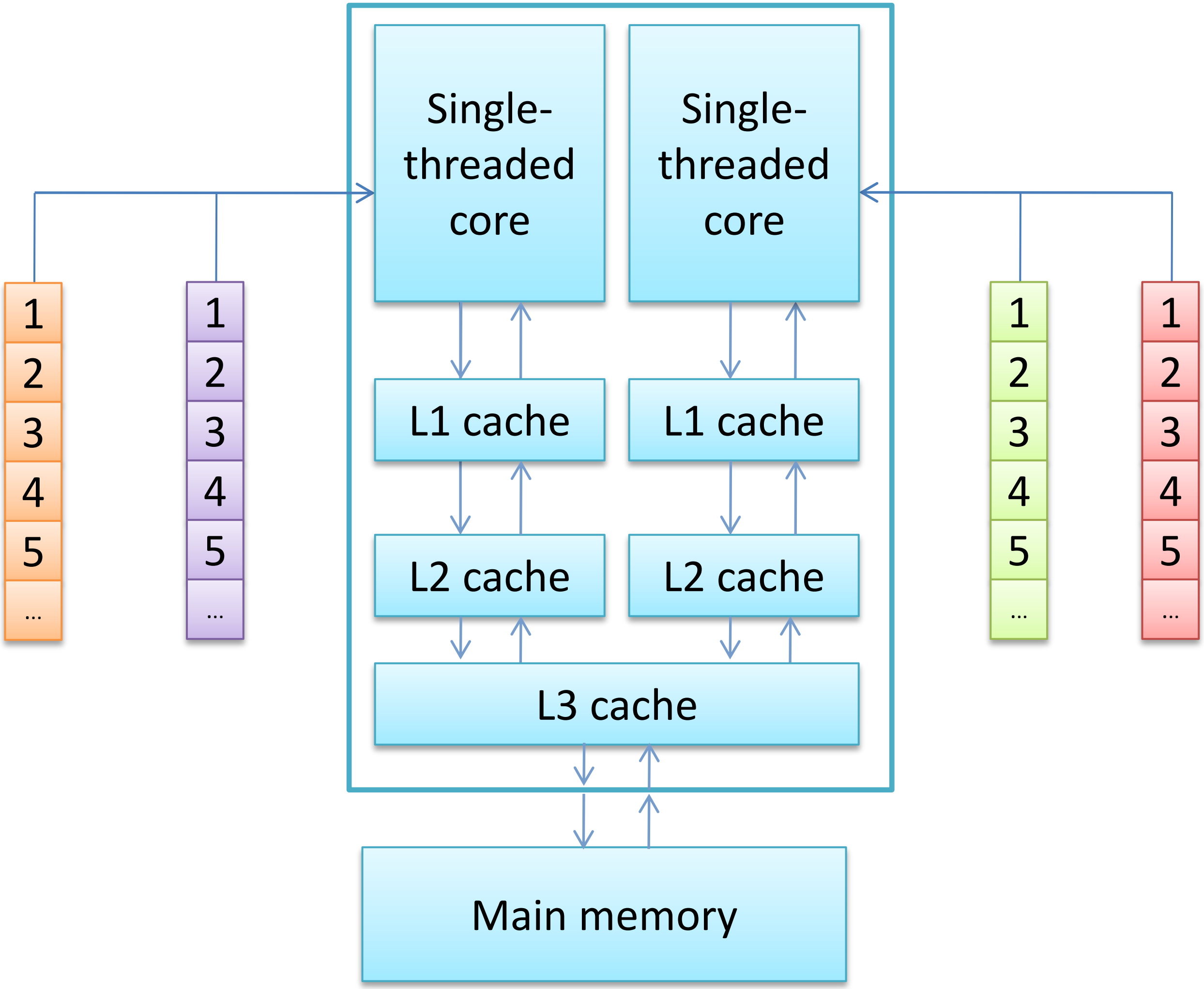
Multi-core h/w – separate L2



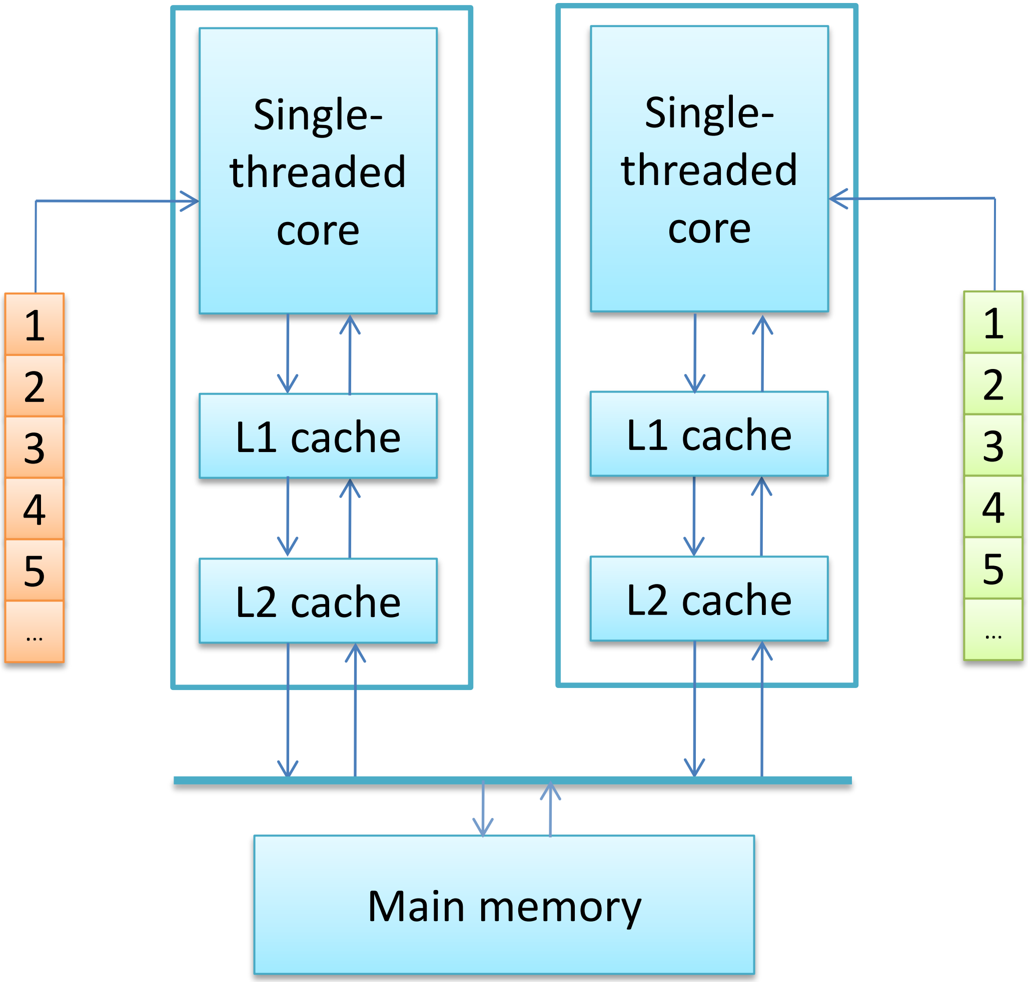
Multi-core h/w – additional L3



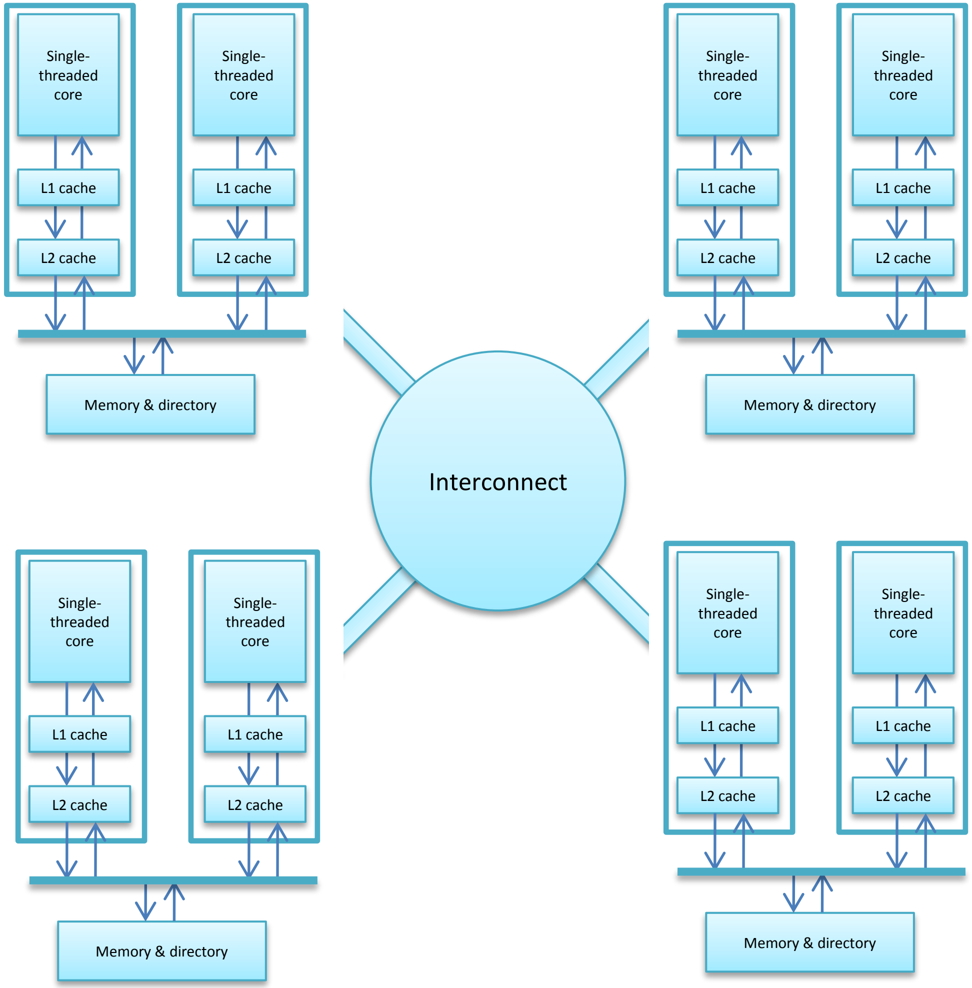
Multi-threaded multi-core h/w



SMP multiprocessor



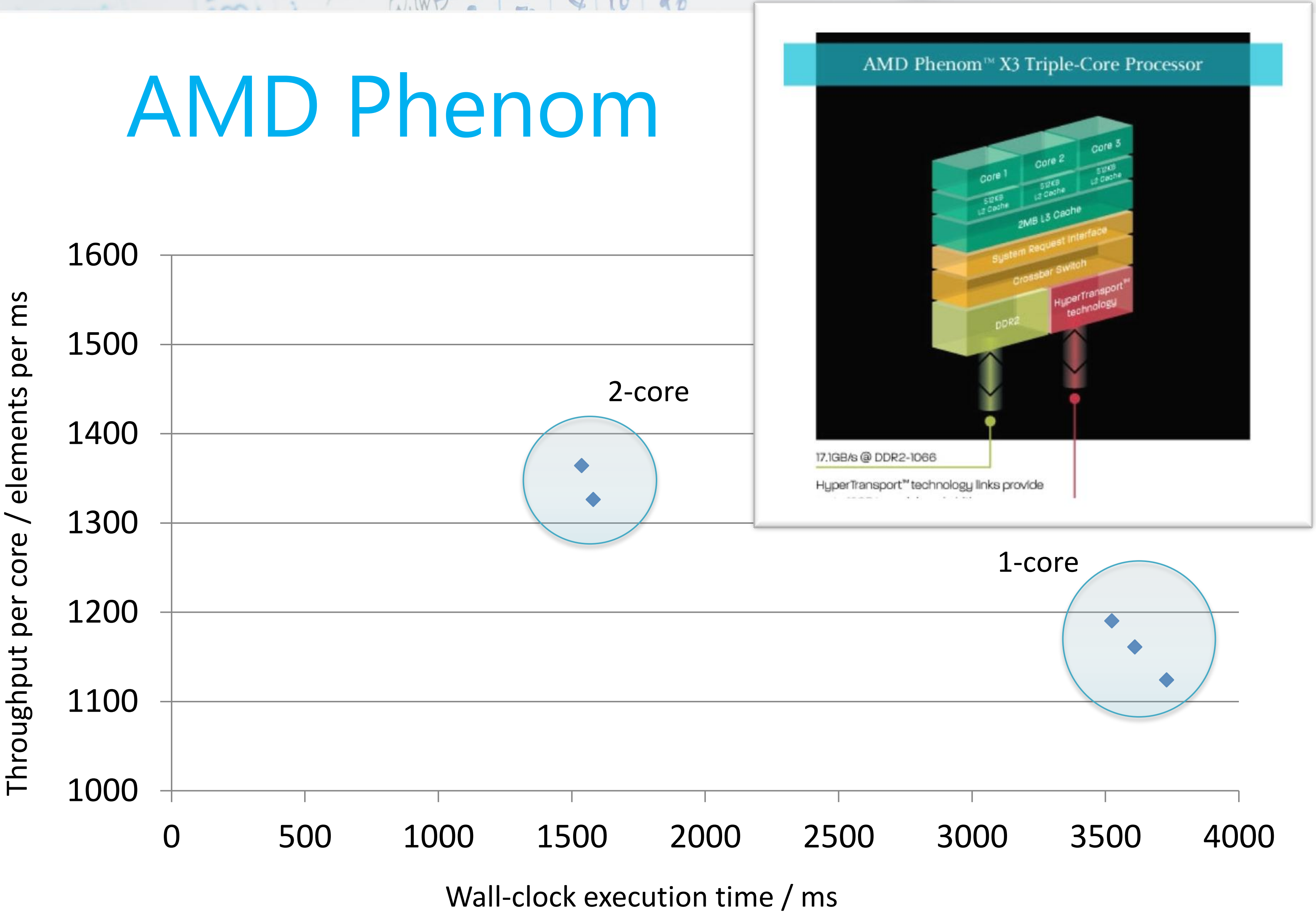
NUMA multiprocessor



Three kinds of parallel hardware

- Multi-threaded cores
  - Increase utilization of a core or memory b/w
  - Peak ops/cycle fixed
- Multiple cores
  - Increase ops/cycle
  - Don’t necessarily scale caches and off-chip resources proportionately
- Multi-processor machines
  - Increase ops/cycle
  - Often scale cache & memory capacities and b/w proportionately

AMD Phenom


AMD Phenom™ X3 Triple-Core Processor
Core 1
Core 2
Core 3
512KB L2 Cache
512KB L2 Cache
512KB L2 Cache
2MB L3 Cache
System Request Interface
Crossbar Switch
DDR2
HyperTransport™ technology
17.1GB/s @ DDR2-1066
HyperTransport™ technology links provide
Throughput per core / elements per ms
1600
1500
1400
1300
1200
1100
1000
0
500
1000
1500
2000
2500
3000
3500
4000
Wall-clock execution time / ms
2-core
1-core