Computer Architecture

14. Multicore System

Jianhua Li
College of Computer and Information
Hefei University of Technology

How to improve performance?

- We have looked at
  - Pipelining
  - Pipeline optimization

- To speed up:
  - Deeper pipelining
  - Make the clock run faster
  - Parallelism
    - It's necessary for performance

Why Multicore?

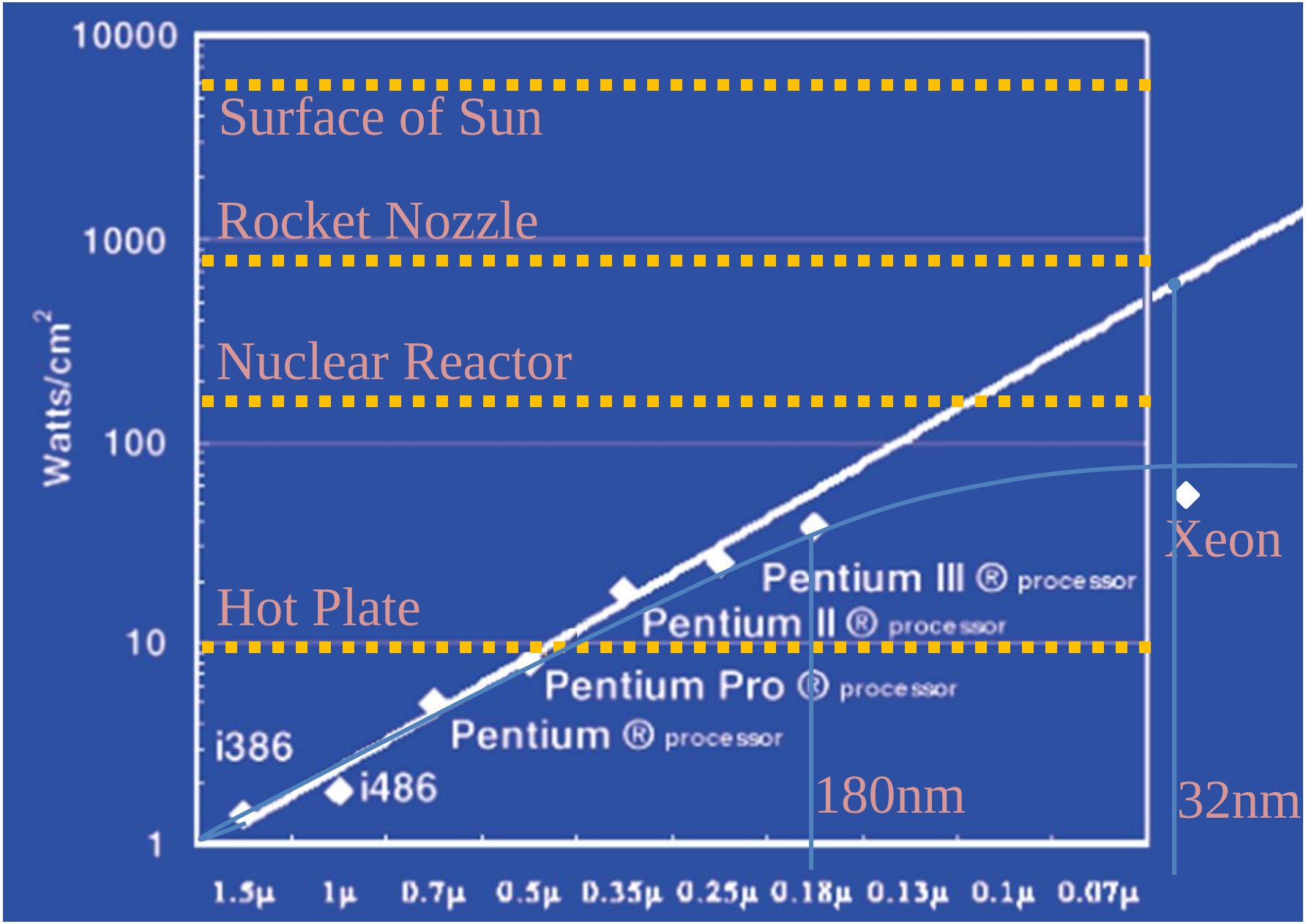
- Moore's law
  - A law about transistors
  - Smaller means more transistors per die
  - And smaller means faster too

- But: need to worry about power too…

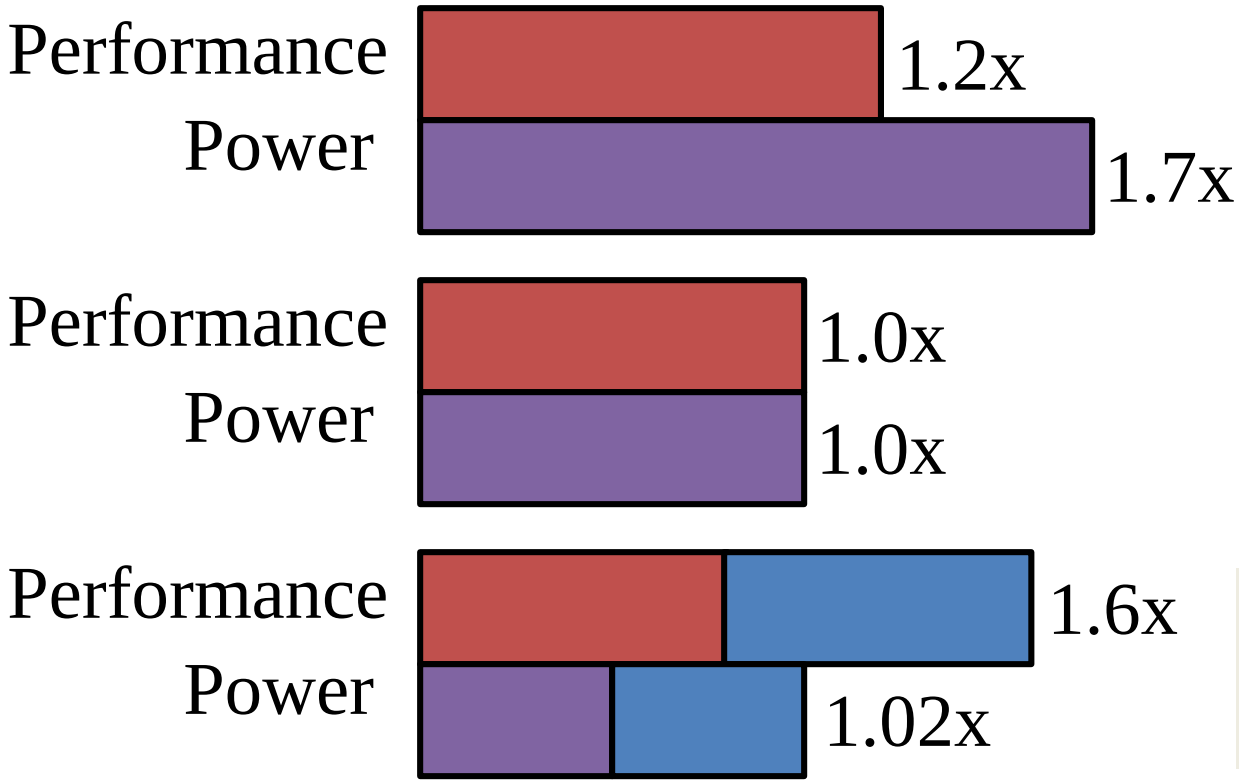
Power Wall

- Power = capacitance * voltage^2 * frequency
- Reducing voltage helps (a lot)
- Better cooling helps
- The power wall
  - We can't reduce voltage further
    - leakage power
  - We can't remove more heat
    - limited by air

Power Limits



Why Multicore?



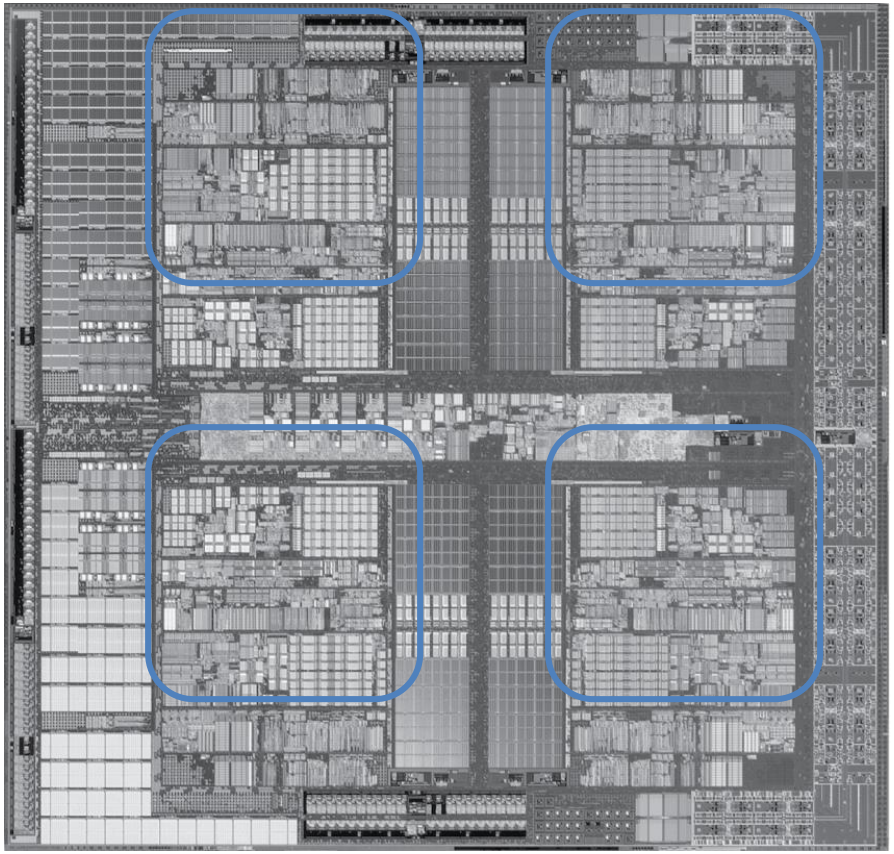
Single-Core
Overclocked +20%

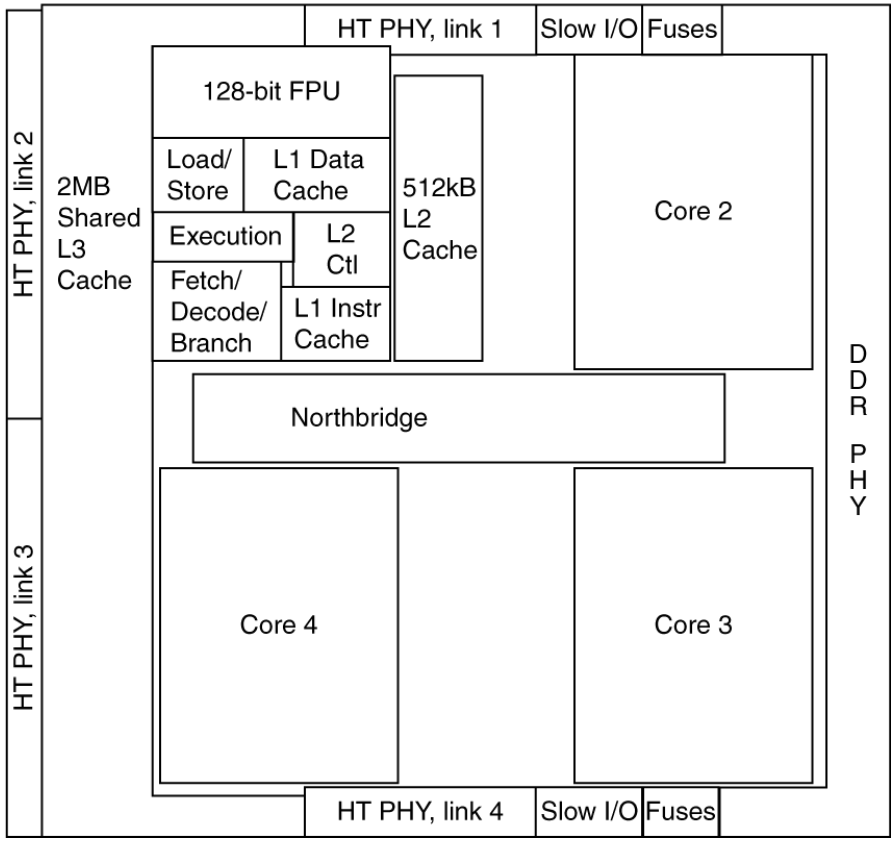
Single-Core

Dual-Core
Underclocked -20%

Inside the Processor

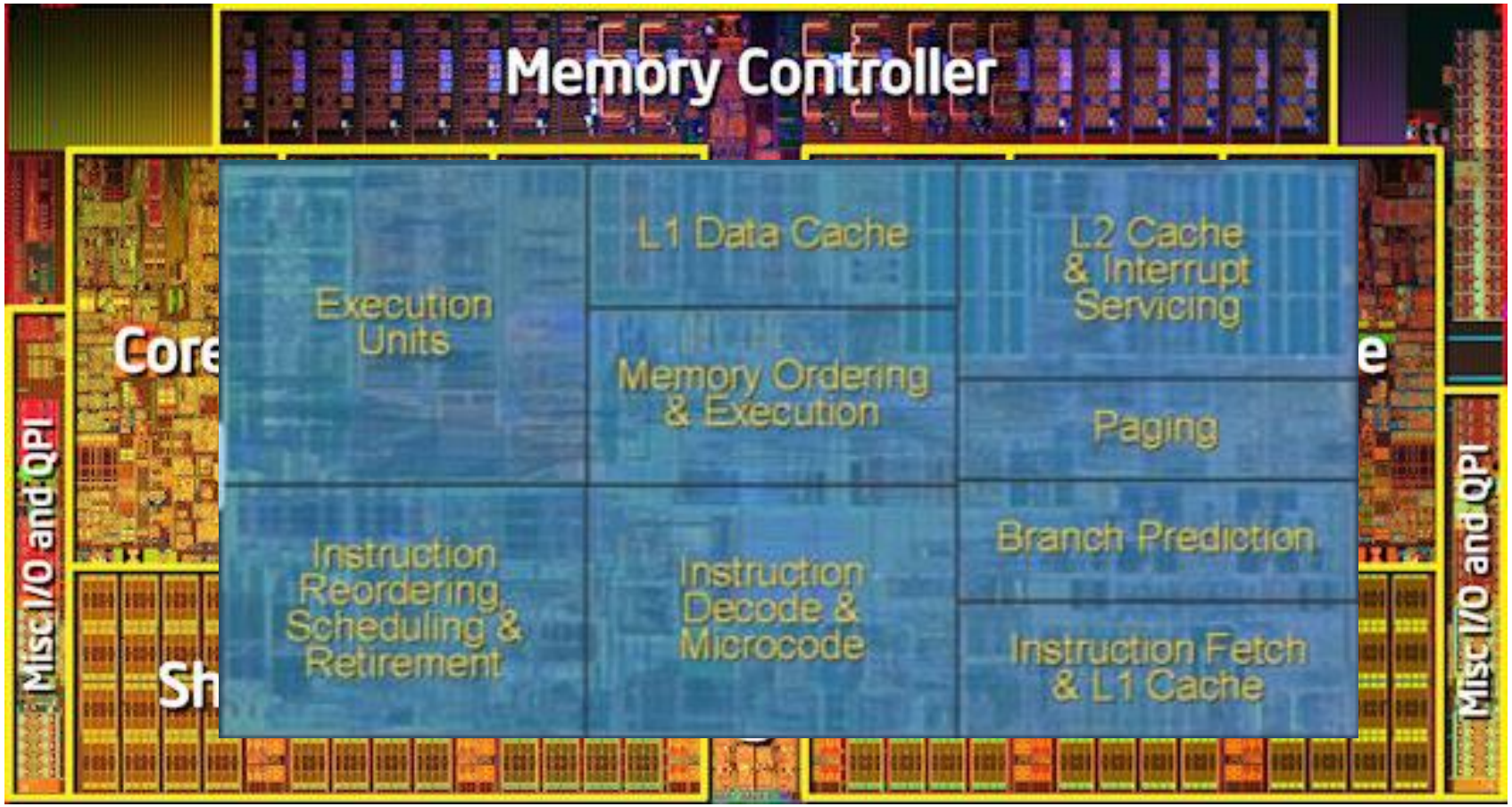
- AMD Barcelona Quad-Core: 4 processor cores





Inside the Processor

- Intel Nehalem Hex-Core



Hyperthreading

- Hyperthreads (Intel)
  - Illusion of multiple cores on a single core

- Switch between hardware threads for stalls
  - Fine grained and coarse grained

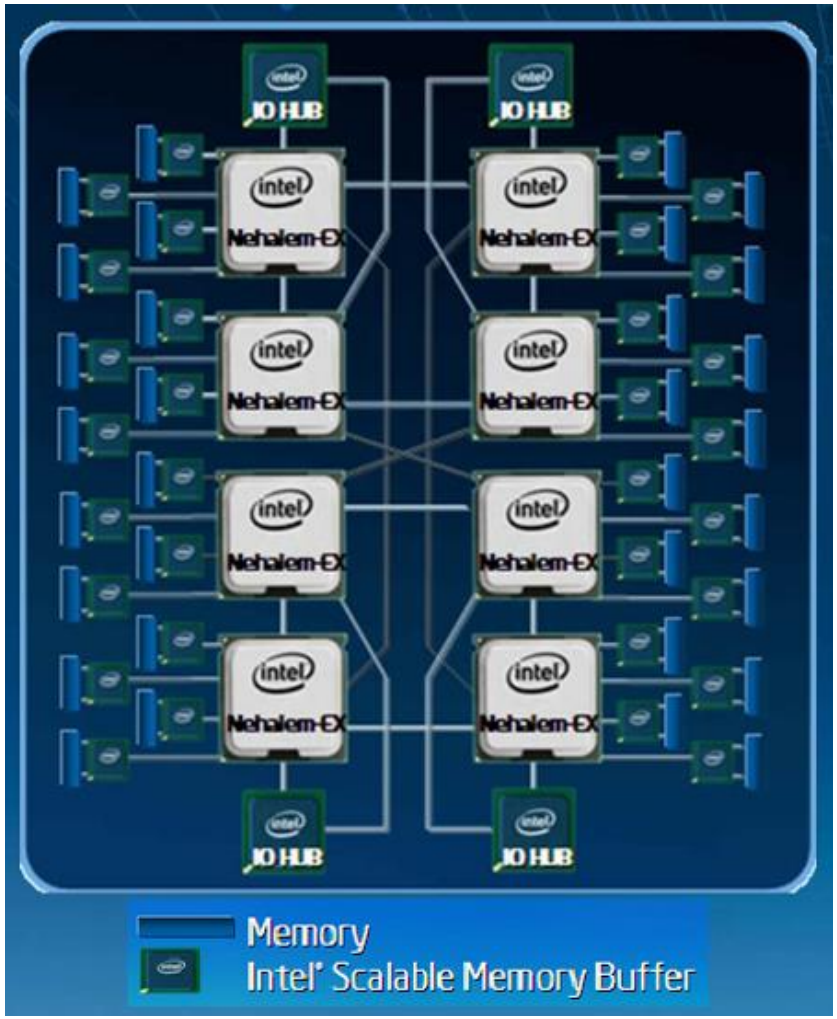
Hyperthreading

	Multi-Core	Multi-Issue	HT
• Programs:	*N*	*1*	*N*
• Num. Pipelines:	*N*	*1*	1
• Pipeline Width:	*1*	*N*	*N*

- Hyperthreads
  - HT = Multi-Issue + extra PCs and registers – dependency logic
  - HT = MultiCore – redundant functional units + hazard avoidance

- Hyperthreads (Intel)
  - Illusion of multiple cores on a single core
  - Easy to keep HT pipelines full + share functional units

Example: All of the above



8 multiprocessors
4 core per multiprocessor
2 HT per core

Dynamic multi-issue: 4 issue
Pipeline depth: 16

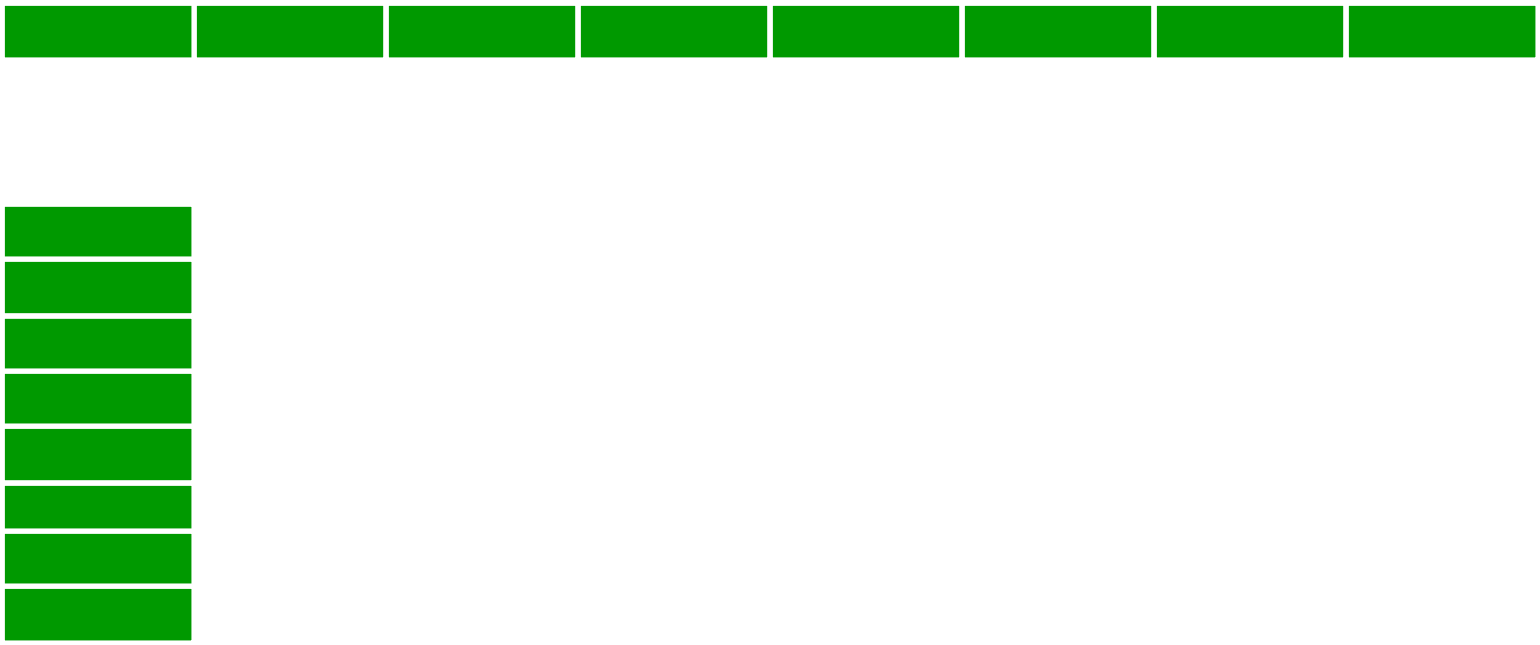
Note: each processor may have multiple processing cores, so this is an example of a **multiprocessor, multicore, hyperthreaded** system

Parallel Programming

- Q: So lets just all use multicore from now on!
- A: Software must be written as parallel program
- Multicore difficulties
  - Partitioning work, balancing load
  - Coordination & synchronization
  - Communication overhead
  - How do you write parallel programs?
    - ... without knowing exact underlying architecture?

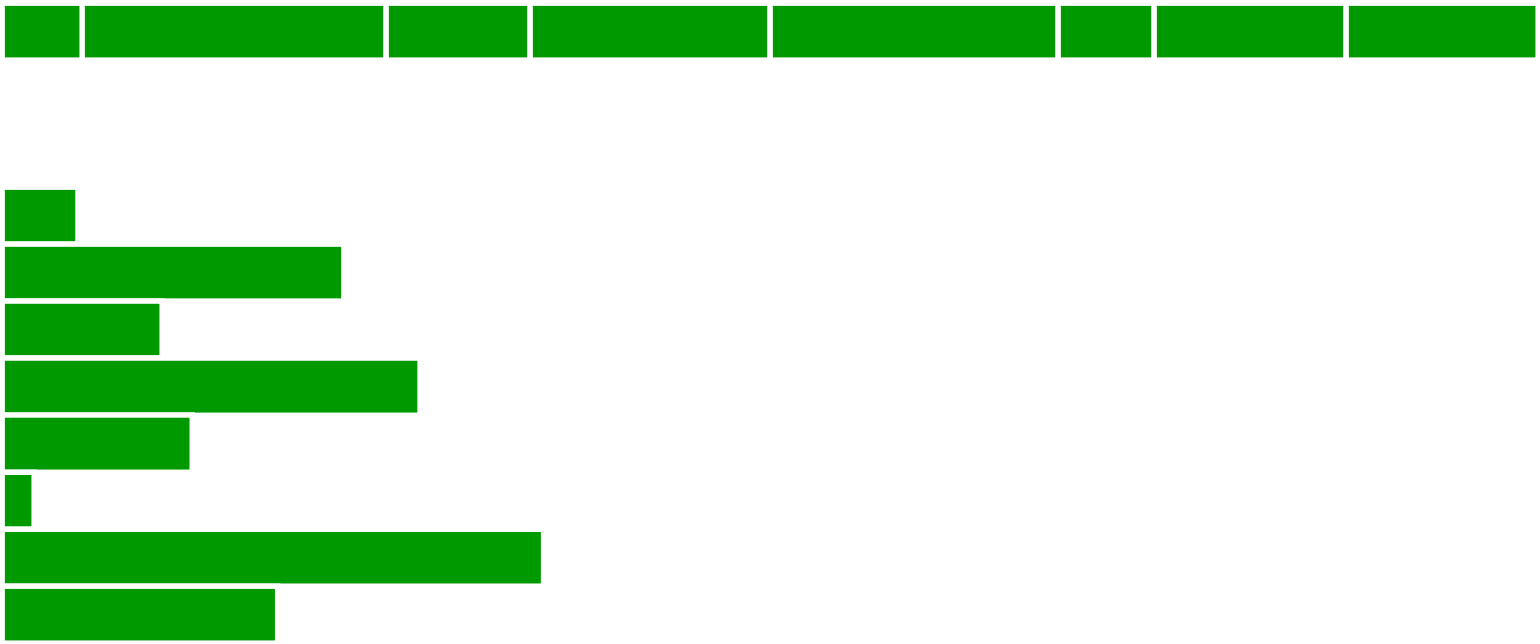
Work Partitioning

- Partition work so all cores have something to do



Load Balancing

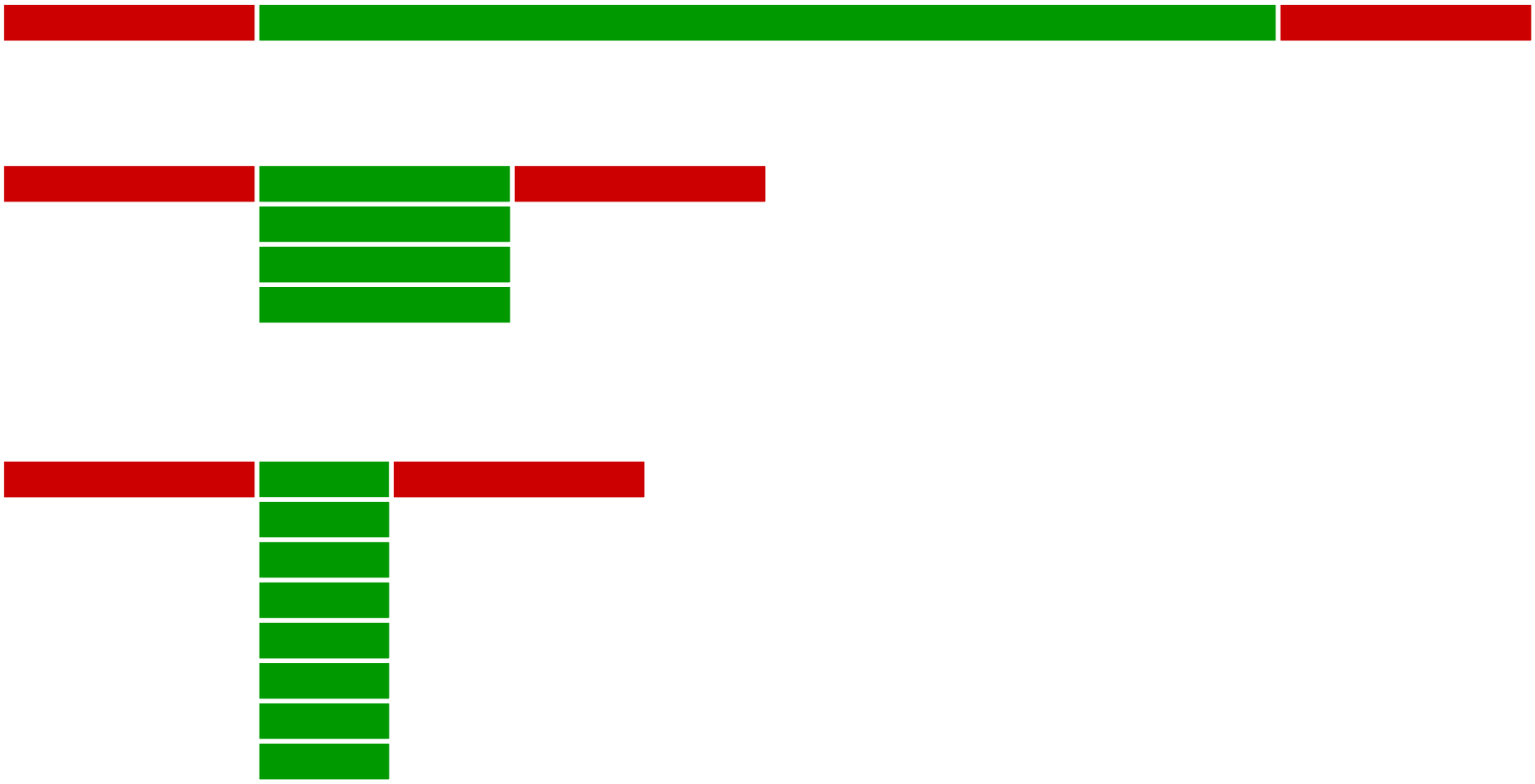
- Need to partition so all cores are actually working



Amdahl's Law

- If tasks have a serial part and a parallel part…
- Example:
  - step 1: divide input data into n pieces
  - step 2: do work on each piece
  - step 3: combine all results
- Recall: Amdahl's Law
- As number of cores increases …
  - time to execute parallel part? goes to zero
  - time to execute serial part? Remains the same
  - *Serial part eventually dominates*

Amdahl's Law



Pitfall: Amdahl's Law

affected execution time

amount of improvement

\+ execution time unaffected

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

Example: multiply accounts for 80s out of 100s

- How much improvement do we need in the multiply performance to get 5× overall improvement?

$$20 = 80/n + 20$$

Can't be done!

Scaling Example

- Workload: sum of 10 scalars, and 10 × 10 matrix sum
  - Speed up from 10 to 100 processors?
- Single processor: Time
- 10 processors
  - Time =
  - Speedup =
- 100 processors
  - Time =
  - Speedup =
- Assumes load can be balanced across processors

Scaling Example

- What if matrix size is 100 × 100?
- Single processor: Time = (10 + 10000) × t_{add}
- 10 processors
  - Time =
  - Speedup =
- 100 processors
  - Time =
  - Speedup =
- Assuming load balanced

Scaling

- Strong scaling vs. weak scaling
- Strong scaling: scales with same problem size
- Weak scaling: scales with increased problem size

Parallel Programming

- Q: So lets just all use multicore from now on!
- A: Software must be written as parallel program
- Multicore difficulties
  - Partitioning work
  - Coordination & synchronization
  - Communications overhead
  - Balancing load over cores

  HW
  - How do you write parallel programs?
    - ... without knowing exact underlying architecture?

SW
Your
career…

Synchronization

Parallelism and Synchronization

- How do I take advantage of multiple processors; *parallelism*?
- How do I write (**correct**) parallel programs, *cache coherency and synchronization*?

- What primitives do I need to implement correct parallel programs?

Topics

- Understand Cache Coherency
- Synchronizing parallel programs
  - Atomic Instructions
  - HW support for synchronization
- How to write parallel programs
  - Threads and processes
  - Critical sections, race conditions, and mutexes

Parallelism and Synchronization

- Cache Coherency Problem: What happens when two or more processors cache ***shared*** data?

Parallelism and Synchronization

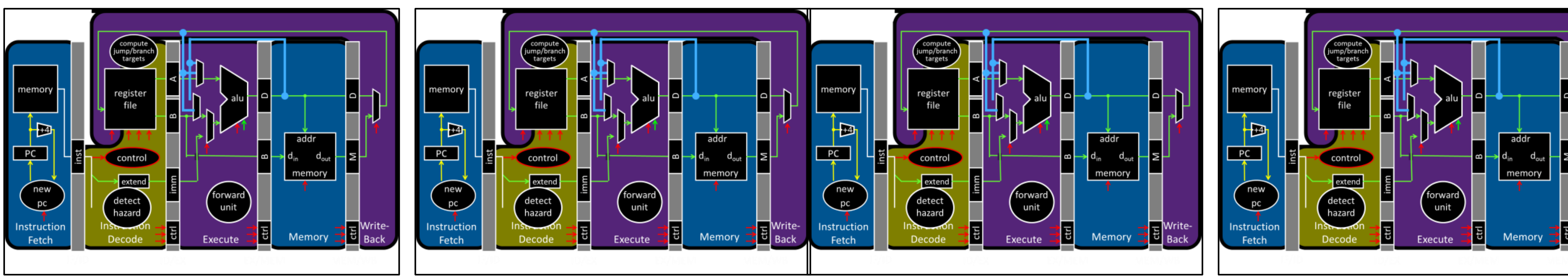
- Cache Coherency Problem: What happens when two or more processors cache *shared* data?

- i.e. the view of memory held by two different processors is through their individual caches

- As a result, processors can see different (incoherent) values to the *same* memory location

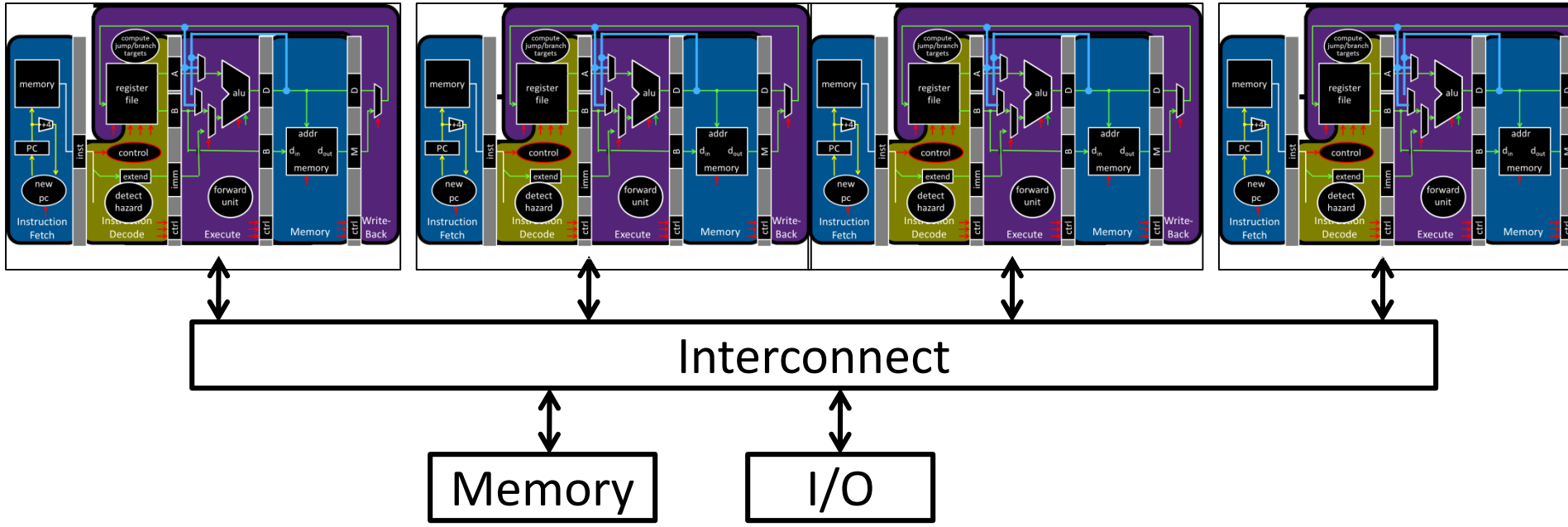
Parallelism and Synchronization

- Each processor core has its own L1 cache



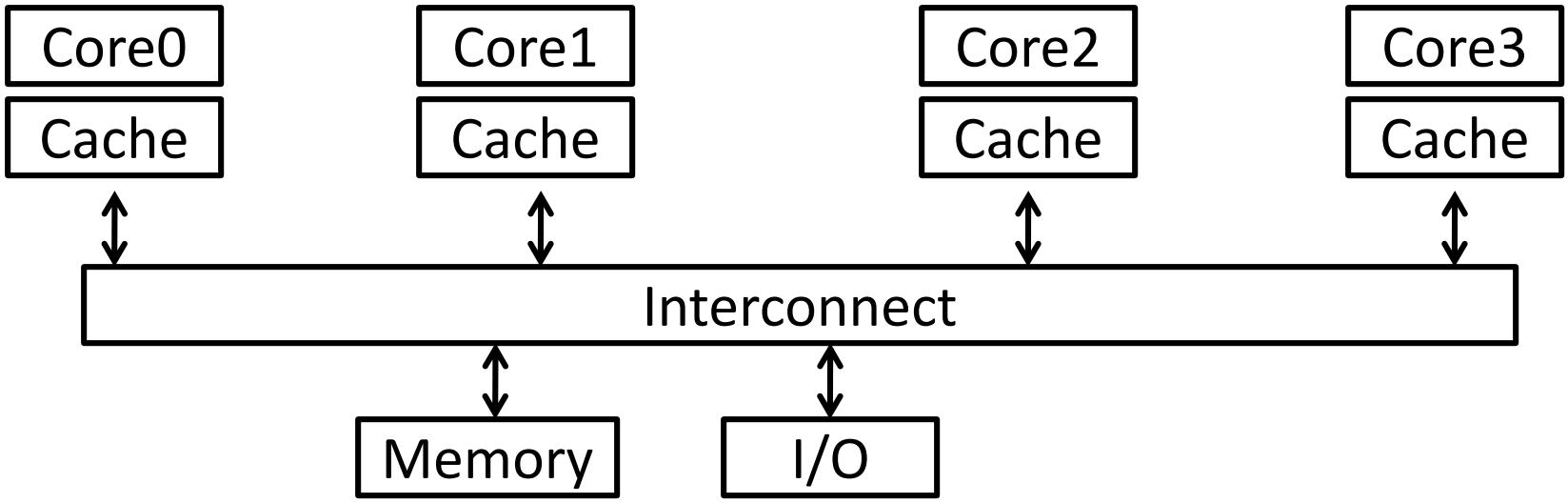
Parallelism and Synchronization

- Each processor core has its own L1 cache



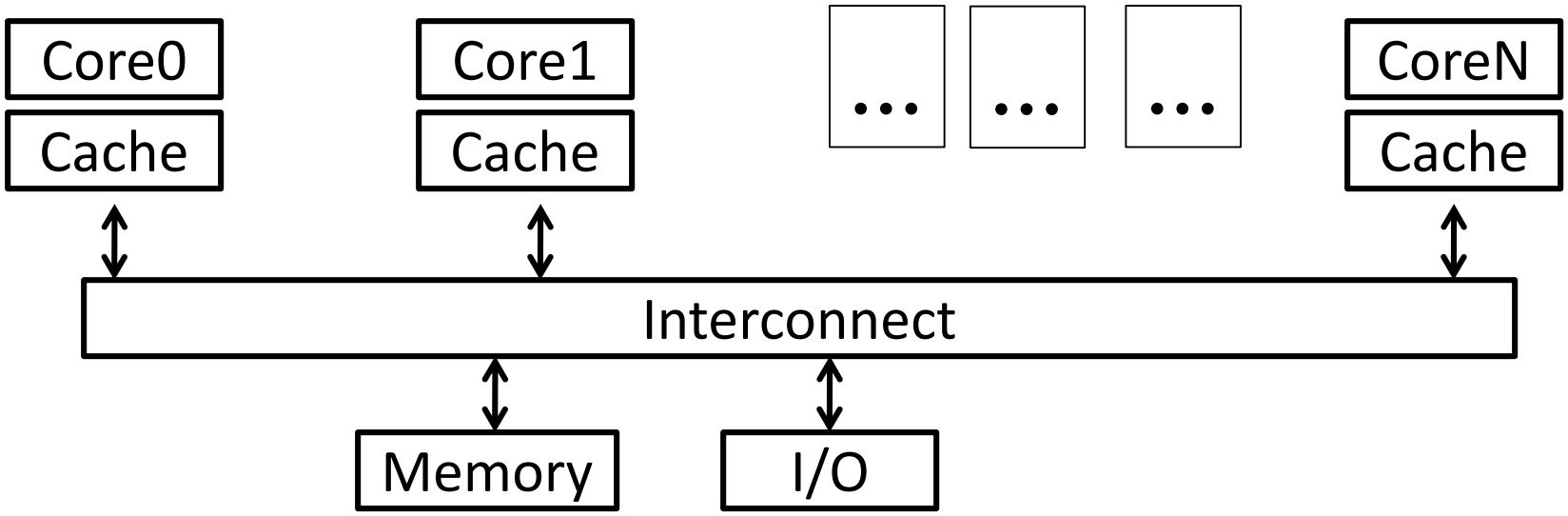
Shared Memory Multiprocessors

- Shared Memory Multiprocessor (SMP)
  - Typical (today): 2 – 8 cores each
  - HW provides *single physical address* space for all processors
  - Assume uniform memory access (ignore NUMA)



Shared Memory Multiprocessors

- Shared Memory Multiprocessor (SMP)
  - Typical (today): 2 – 8 cores each
  - HW provides *single physical address* space for all processors
  - Assume uniform memory access (ignore NUMA)



Cache Coherency Problem

- Thread A (on Core0)

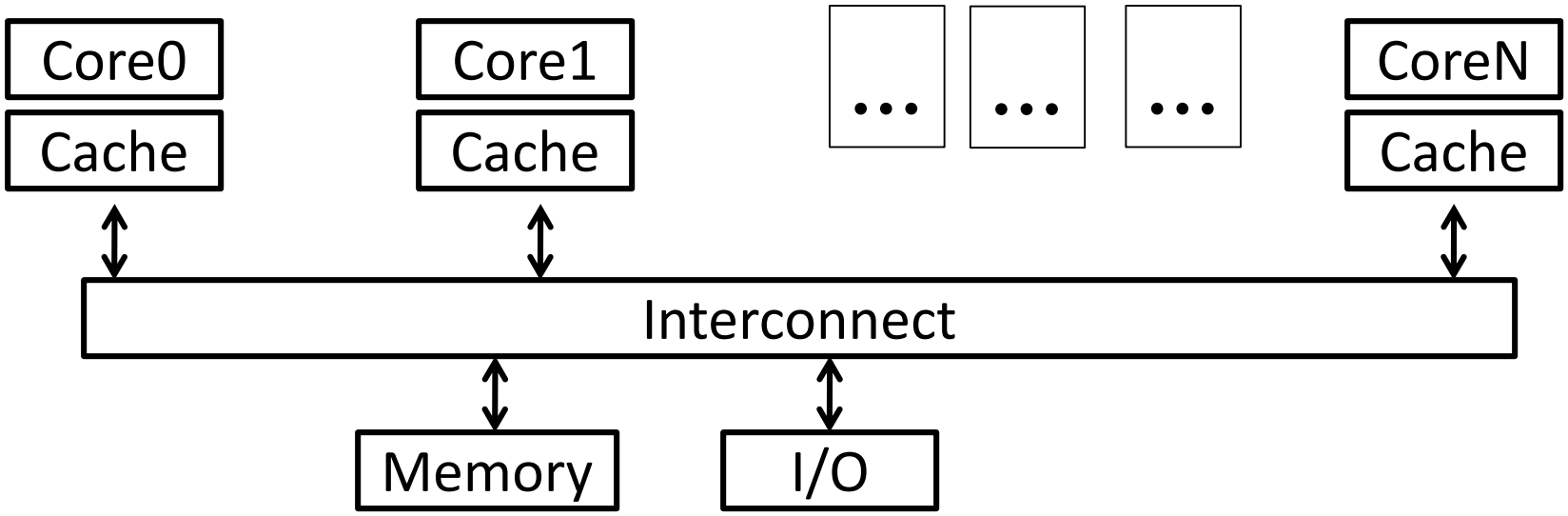
```
for(int i = 0, i < 5; i++){
        x = x + 1;
}
```

Thread B (on Core1)

```
for(int j = 0; j < 5; j++) {
        x = x + 1;
}
```

- What will the value of x be after both loops finish?

Start: x = 0



Cache Coherency Problem

- Thread A (on Core0)

```
for(int i = 0, i < 5; i++) {
        x = x + 1;
}
```

Thread B (on Core1)

```
for(int j = 0; j < 5; j++) {
        x = x + 1;
}
```

Cache Coherency Problem

- Thread A (on Core0)

```
for(int i = 0, i < 5; i++) {
        LW $t0, addr(x)

        ADDIU $t0, $t0, 1

        SW $t0, addr(x)
}
```

Thread B (on Core1)

```
for(int j = 0; j < 5; j++) {
        LW $t0, addr(x)

        ADDIU $t0, $t0, 1

        SW $t0, addr(x)
}
```

Cache Coherency Problem

- Thread A (on Core0)

```
for(int i = 0, i < 5; i++) {
    x = x + 1;
}
```

Thread B (on Core1)

```
for(int j = 0; j < 5; j++) {
    x = x + 1;
}
```

- What will the value of x be after both loops finish?

a) 6

b) 8

c) 10

d) All of the above

e) None of the above