Multicore and Parallelism

Prof. Hakim Weatherspoon

CS 3410, Spring 2015

Computer Science

Cornell University

P & H Chapter 4.10, 1.7, 1.8, 5.10, 6

Announcements

- HW2 Review Sessions!
  - Saturday, April 18th, Hollister B14@7pm
  - Tuesday, April 21st, Hollister B14@7pm
- Lab3 is due yesterday!
  - Wednesday, April 15th
- PA3 started this week!
  - The Lord of the Cache!
  - Due Friday, April 24th

Announcements

- HW2-P5 (Pre-Lab4) due next week!
  - Monday, April 20th
  - Don’t forget to submit on CMS!
  - Designed for you to look over the code and understand virtual memory before coming to Lab 4.

Announcements

- Prelim 2 is on April 30th at 7 PM at Statler Hall!

- If you have a conflict e-mail me:

deniz@cs.cornell.edu

Announcements

Next three weeks

- Week 12 (Apr 21): Lab4 due in-class, Proj3 due Fri, HW2 due Sat
- Week 13 (Apr 28): Proj4 release, Prelim2
- Week 14 (May 5): Proj3 tournament Mon, Proj4 design doc due

Final Project for class

- Week 15 (May 12): Proj4 due Wed

Today

Many ways to improve performance

- Instruction Level Parallelism
- Multicore
  - Performance in multicore

Next 2 lectures: synchronization

xkcd/619


IT TOOK A LOT OF WORK, BUT THIS LATEST LINUX PATCH ENABLES SUPPORT FOR MACHINES WITH 4,096 CPUs, UP FROM THE OLD LIMIT OF 1,024.
DO YOU HAVE SUPPORT FOR SMOOTH FULL-SCREEN FLASH VIDEO YET?
NO, BUT WHO USES THAT?

Pitfall: Amdahl's Law

Execution time after improvement =

$$\frac{\text{affected execution time}}{\text{amount of improvement}} + \text{execution time unaffected}$$

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

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

Scaling Example

Workload: sum of 10 scalars, and 10 × 10 matrix sum

- Speed up from 10 to 100 processors?

Single processor: Time = $(10 + 100) \times t_{add}$

10 processors

100 processors

.

Scaling Example

What if matrix size is 100×100?

Single processor: Time = $(10 + 10000) \times t_{add}$

10 processors

100 processors

•

Goals for Today

How to improve System Performance?

- Instruction Level Parallelism (ILP)
- Multicore
  - Increase clock frequency vs multicore
- Beware of Amdahls Law

Next time:

- Concurrency, programming, and synchronization

Problem Statement

Q: How to improve system performance?

→ Increase CPU clock rate?

→ But I/O speeds are limited

Disk, Memory, Networks, etc.

Recall: Amdahl's Law

Solution: Parallelism

Instruction-Level Parallelism (ILP)

Pipelining: execute multiple instructions in parallel

Q: How to get more instruction level parallelism?

A: Deeper pipeline

- E.g. 250MHz 1-stage; 500Mhz 2-stage; 1GHz 4-stage; 4GHz 16-stage

Pipeline depth limited by…

- max clock speed (less work per stage $\Rightarrow$ shorter clock cycle)
- min unit of work
- dependencies, hazards / forwarding logic

Instruction-Level Parallelism (ILP)

Pipelining: execute multiple instructions in parallel

Q: How to get more instruction level parallelism?

Static Multiple Issue

Static Multiple Issue

a.k.a. Very Long Instruction Word (VLIW)

Compiler groups instructions to be issued together

- Packages them into “issue slots”

Q: How does HW detect and resolve hazards?

MIPS with Static Dual Issue

Two-issue packets

- One ALU/branch instruction
- One load/store instruction
- 64-bit aligned
  - ALU/branch, then load/store
  - Pad an unused instruction with nop

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

Scheduling Example

Schedule this for dual-issue MIPS

```
Loop: lw   $t0, 0($s1)       # $t0=array element
      addu $t0, $t0, $s2     # add scalar in $s2
      sw   $t0, 0($s1)       # store result
      addi $s1, $s1,–4       # decrement pointer
      bne  $s1, $zero, Loop # branch $s1!=0
```

	ALU/branch	Load/store	cycle

Scheduling Example

Compiler scheduling for dual-issue MIPS…

```
Loop:  lw   $t0, 0($s1)          # $t0 = A[i]
       lw   $t1, 4($s1)          # $t1 = A[i+1]
       addu $t0, $t0, $s2        # add $s2
       addu $t1, $t1, $s2        # add $s2
       sw   $t0, 0($s1)          # store A[i]
       sw   $t1, 4($s1)          # store A[i+1]
       addi $s1, $s1, +8         # increment pointer
       bne  $s1, $s3, TOP        # continue if $s1!=end
```

```
      ALU/branch slot        Load/store slot       cycle
Loop: nop                    lw   $t0,  0($s1)       1
      nop                    lw   $t1,  4($s1)       2
      addu $t0, $t0, $s2     nop                     3
      addu $t1, $t1, $s2     sw   $t0,  0($s1)       4
      addi $s1, $s1, +8      sw   $t1,  4($s1)       5
      bne  $s1, $s3, TOP     nop                     6
```

Scheduling Example

Compiler scheduling for dual-issue MIPS…

```
Loop: lw   $t0, 0($s1)          # $t0 = A[i]
      lw   $t1, 4($s1)          # $t1 = A[i+1]
      addu $t0, $t0, $s2        # add $s2
      addu $t1, $t1, $s2        # add $s2
      sw   $t0, 0($s1)          # store A[i]
      sw   $t1, 4($s1)          # store A[i+1]
      addi $s1, $s1, +8         # increment pointer
      bne  $s1, $s3, TOP        # continue if $s1!=end
```

	ALU/branch slot	Load/store slot	cycle
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1, +8	lw $t1, 4($s1)	2
	addu $t0, $t0, $s2	nop	3
	addu $t1, $t1, $s2	sw $t0, -8($s1)	4
	bne $s1, $s3, Loop	sw $t1, -4($s1)	5

Limits of Static Scheduling

Compiler scheduling for dual-issue MIPS…

```
lw   $t0, 0($s1)          # load A
addi $t0, $t0, +1         # increment A
sw   $t0, 0($s1)          # store A
lw   $t0, 0($s2)          # load B
addi $t0, $t0, +1         # increment B
sw   $t0, 0($s2)          # store B
```

ALU/branch slot	Load/store slot	cycle
nop	lw $t0, 0($s1)	1
nop	nop	2
addi $t0, $t0, +1	nop	3
nop	sw $t0, 0($s1)	4
nop	lw $t0, 0($s2)	5
nop	nop	6
addi $t0, $t0, +1	nop	7
nop	sw $t0, 0($s2)	8

Limits of Static Scheduling

Compiler scheduling for dual-issue MIPS…

```
lw   $t0, 0($s1)          # load A
addi $t0, $t0, +1         # increment A
sw   $t0, 0($s1)          # store A
lw   $t1, 0($s2)          # load B
addi $t1, $t1, +1         # increment B
sw   $t0, 0($s2)          # store B
```

ALU/branch slot	Load/store slot	cycle
nop	lw $t0, 0($s1)	1
nop	lw $t1, 0($s2)	2
addi $t0, $t0, +1	nop	3
addi $t1, $t1, +1	sw $t0, 0($s1)	4
nop	sw $t1, 0($s2)	5

Dynamic Multiple Issue

Dynamic Multiple Issue

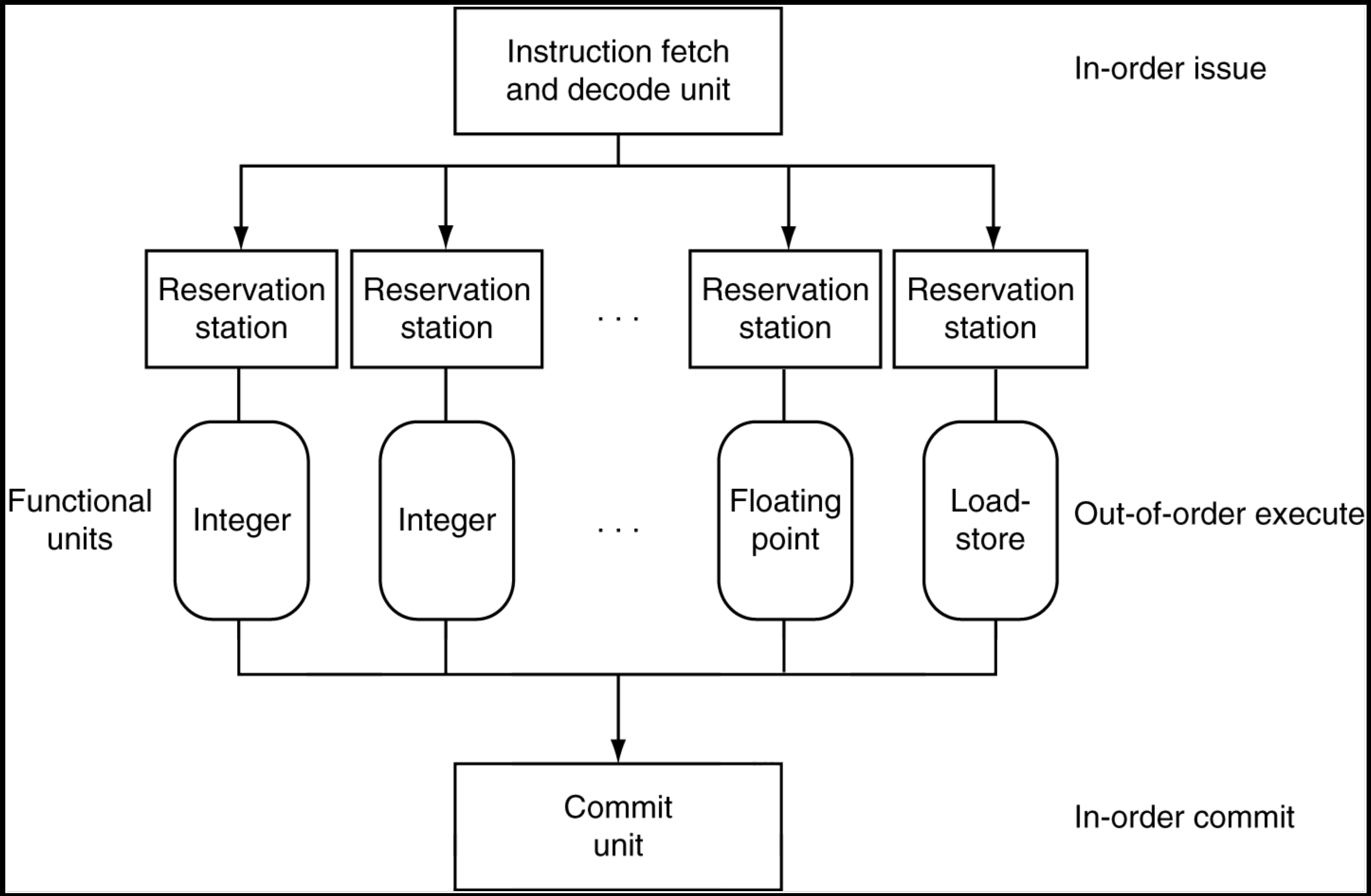
a.k.a. SuperScalar Processor (c.f. Intel)

- CPU examines instruction stream and chooses multiple instructions to issue each cycle
- Compiler can help by reordering instructions….
- … but CPU is responsible for resolving hazards

Even better: Speculation/Out-of-order Execution

- Execute instructions as early as possible
- Aggressive register renaming
- Guess results of branches, loads, etc.
- Roll back if guesses were wrong
- Don’t commit results until all previous insts. are retired

Dynamic Multiple Issue

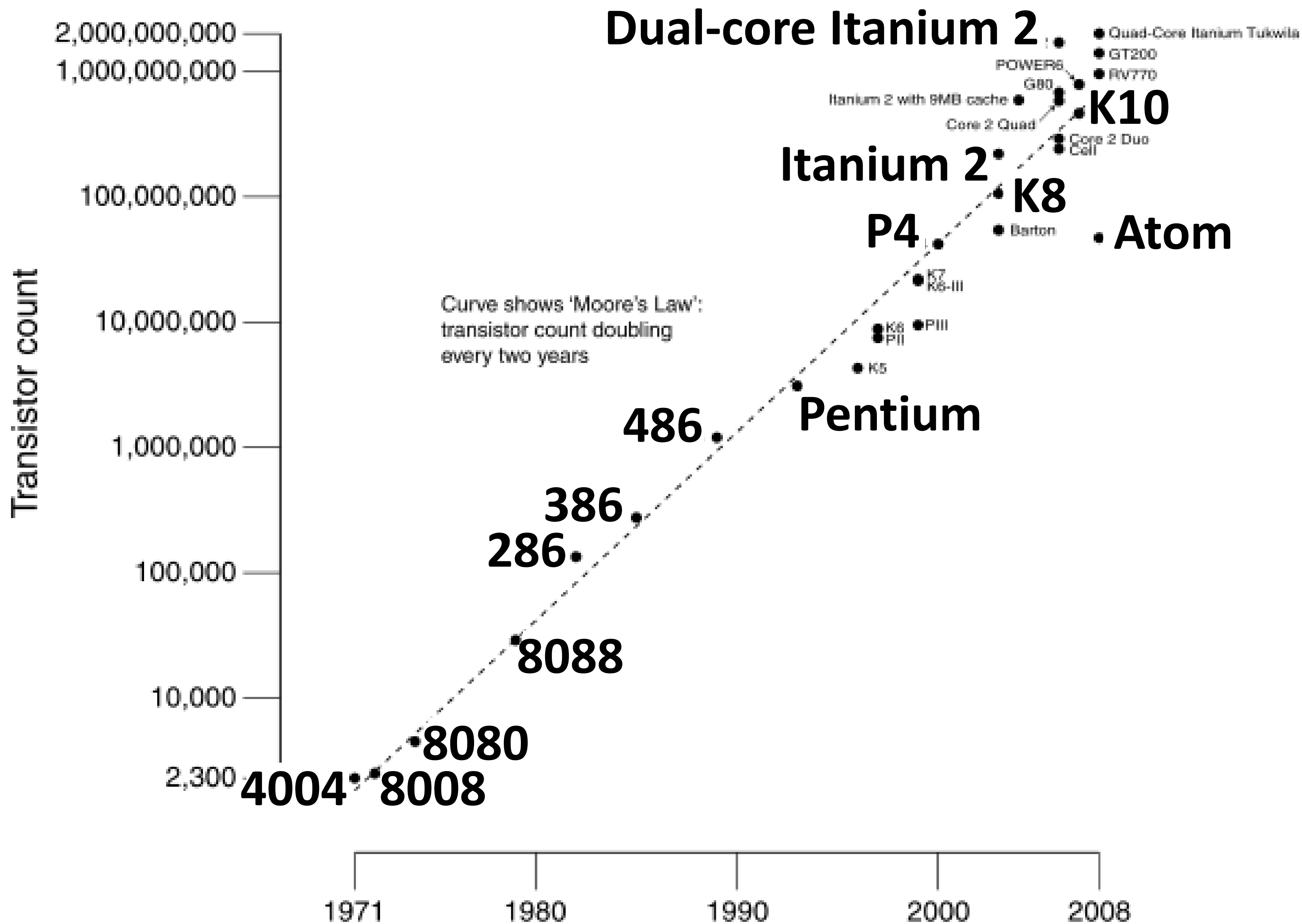


Does Multiple Issue Work?

Q: Does multiple issue / ILP work?

Power Efficiency

Q: Does multiple issue / ILP cost much?


Dual-core Itanium 2
Quad-Core Itanium Tukwila
GT200
POWER6
RV770
G80
Itanium 2 with 9MB cache
K10
Core 2 Quad
Core 2 Duo
Cell
Itanium 2
K8
P4
Barton
Atom
K7
K6-III
Curve shows 'Moore's Law':
transistor count doubling
every two years
K6
PIII
PII
K5
Pentium
486
386
286
8088
8080
4004
8008
Transistor count
2,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
1971
1980
1990
2000
2008

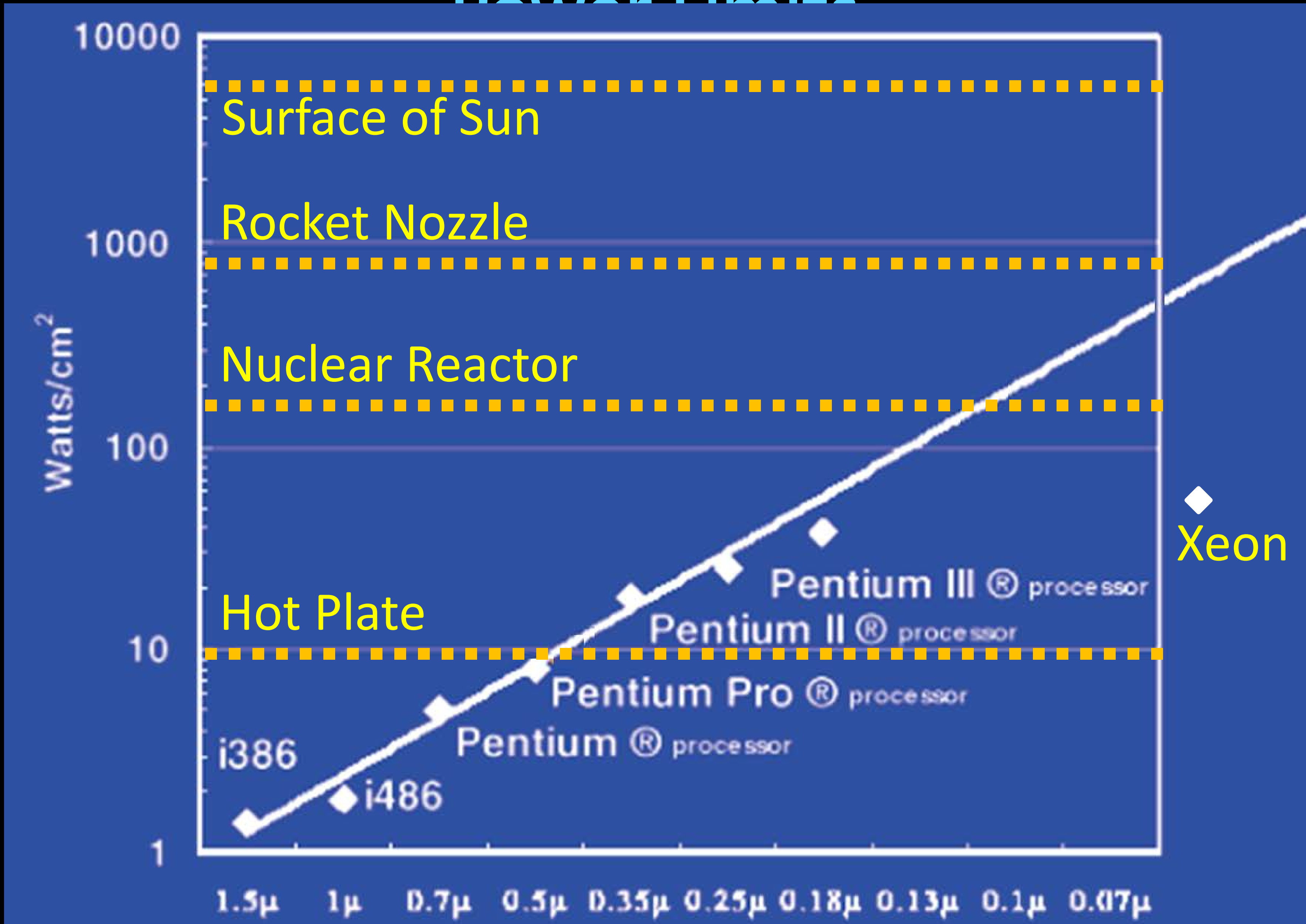
Why Multicore?

Moore's law

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

But: Power consumption growing too…

Power Limits



Power Wall

Power = capacitance * voltage^2 * frequency

In practice: Power ~ voltage^3

Reducing voltage helps (a lot)

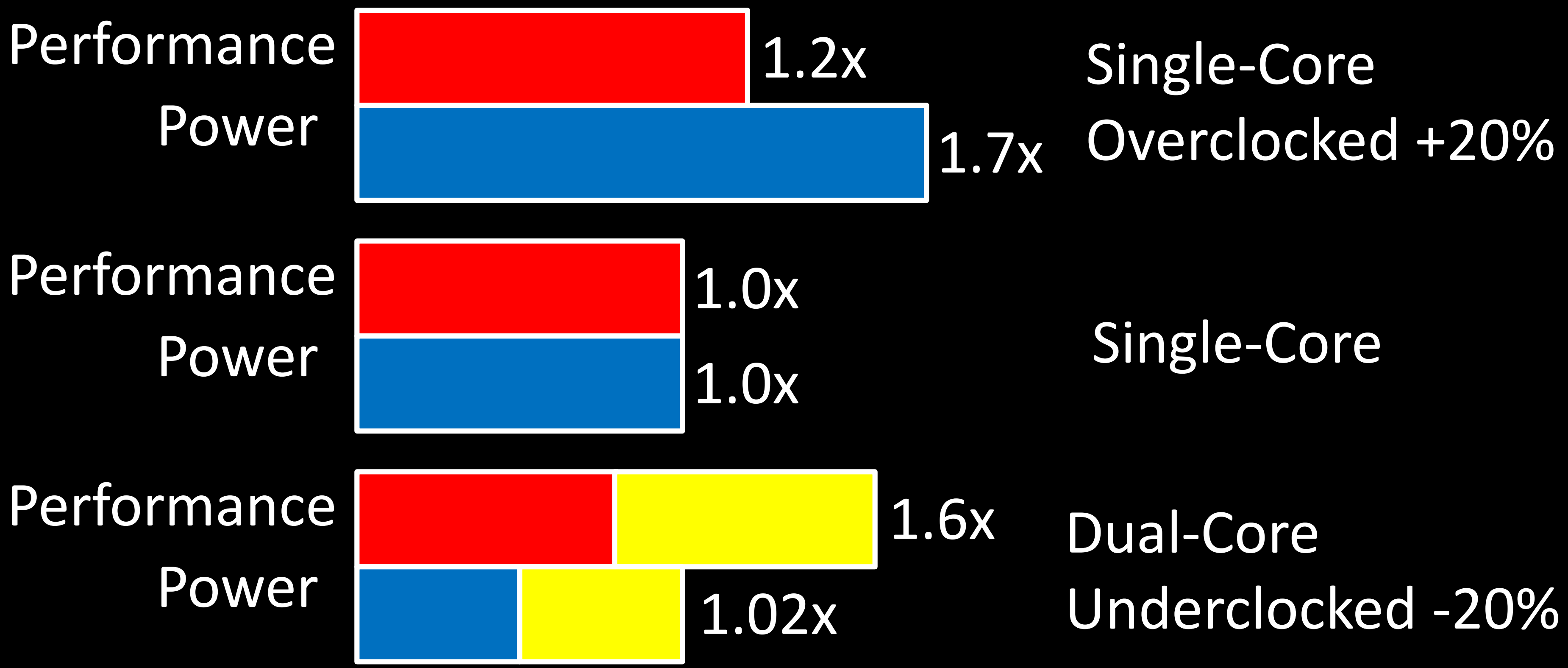
... so does reducing clock speed

Better cooling helps

The power wall

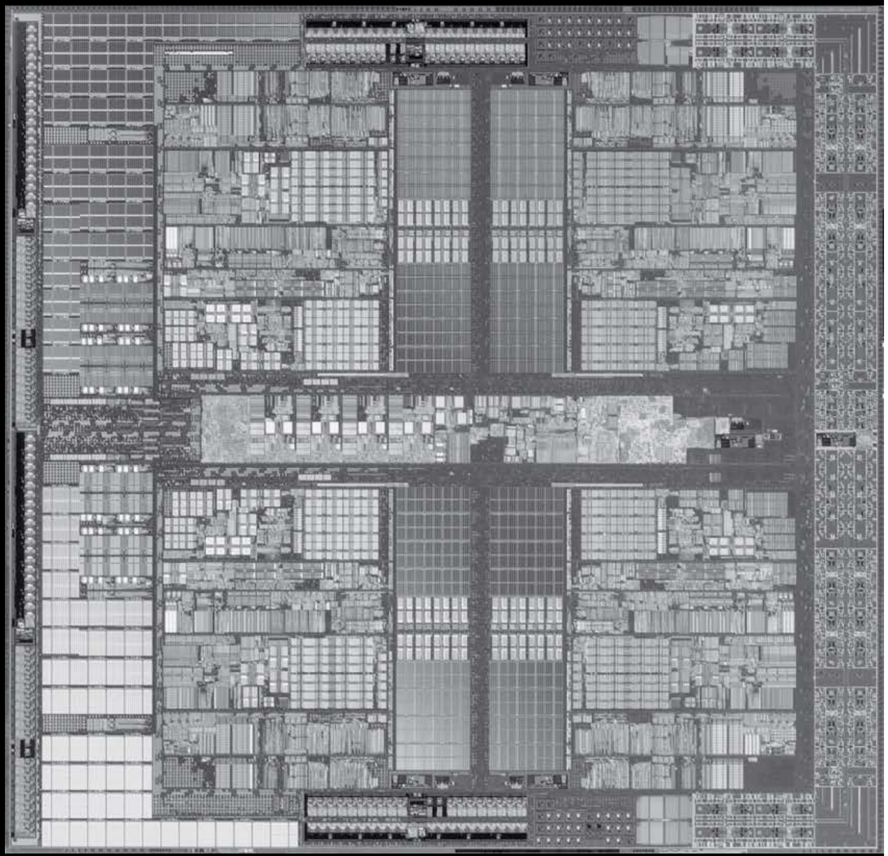
- We can't reduce voltage further
- We can't remove more heat

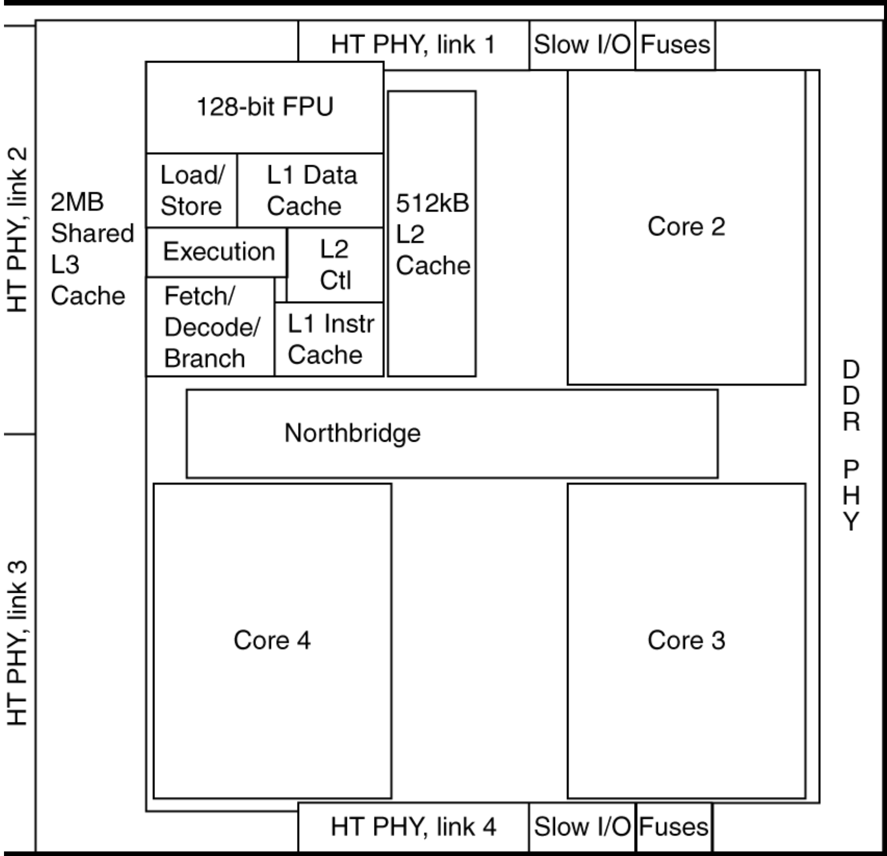
Why Multicore?



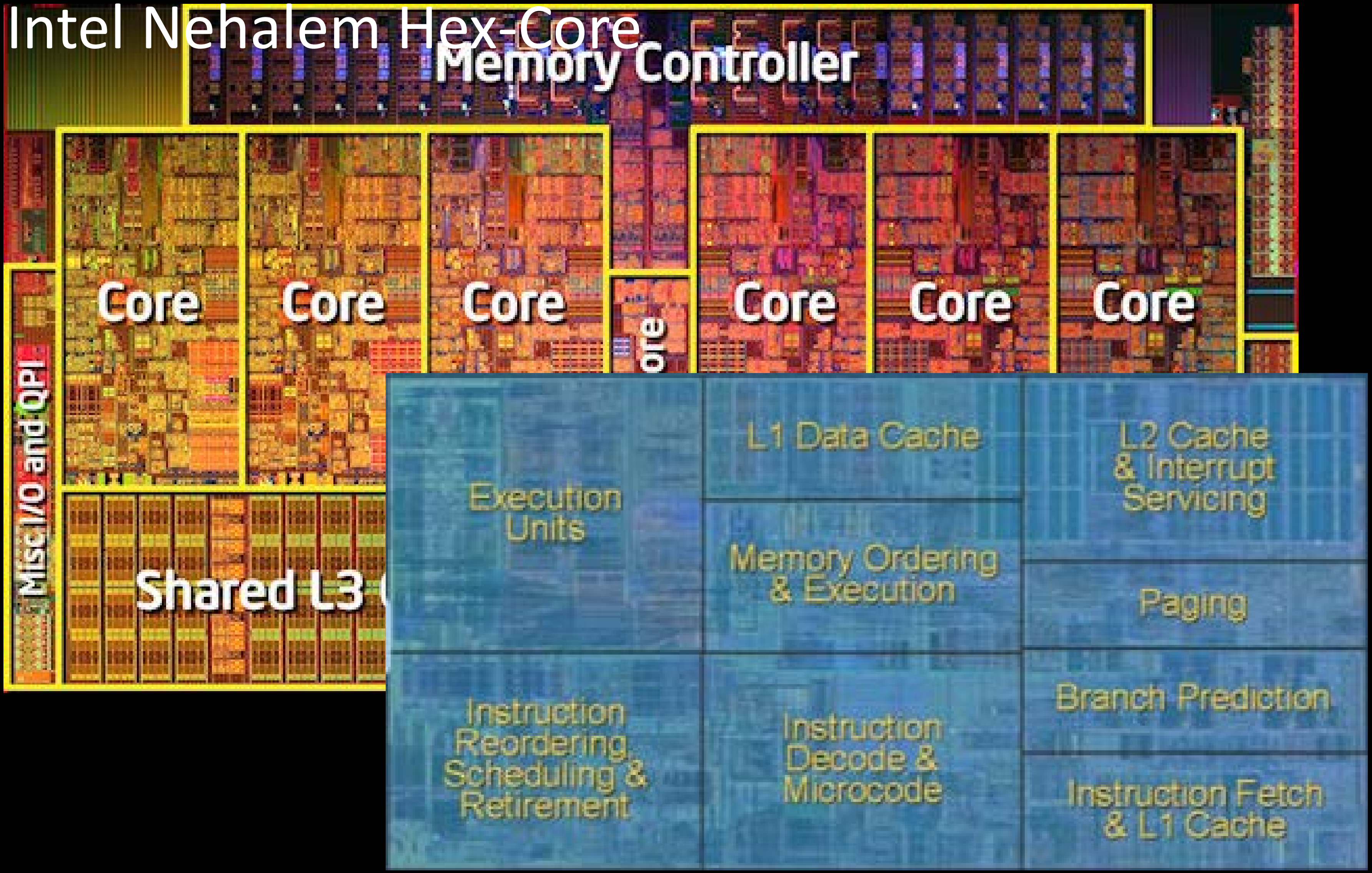
Inside the Processor

AMD Barcelona Quad-Core: 4 processor cores





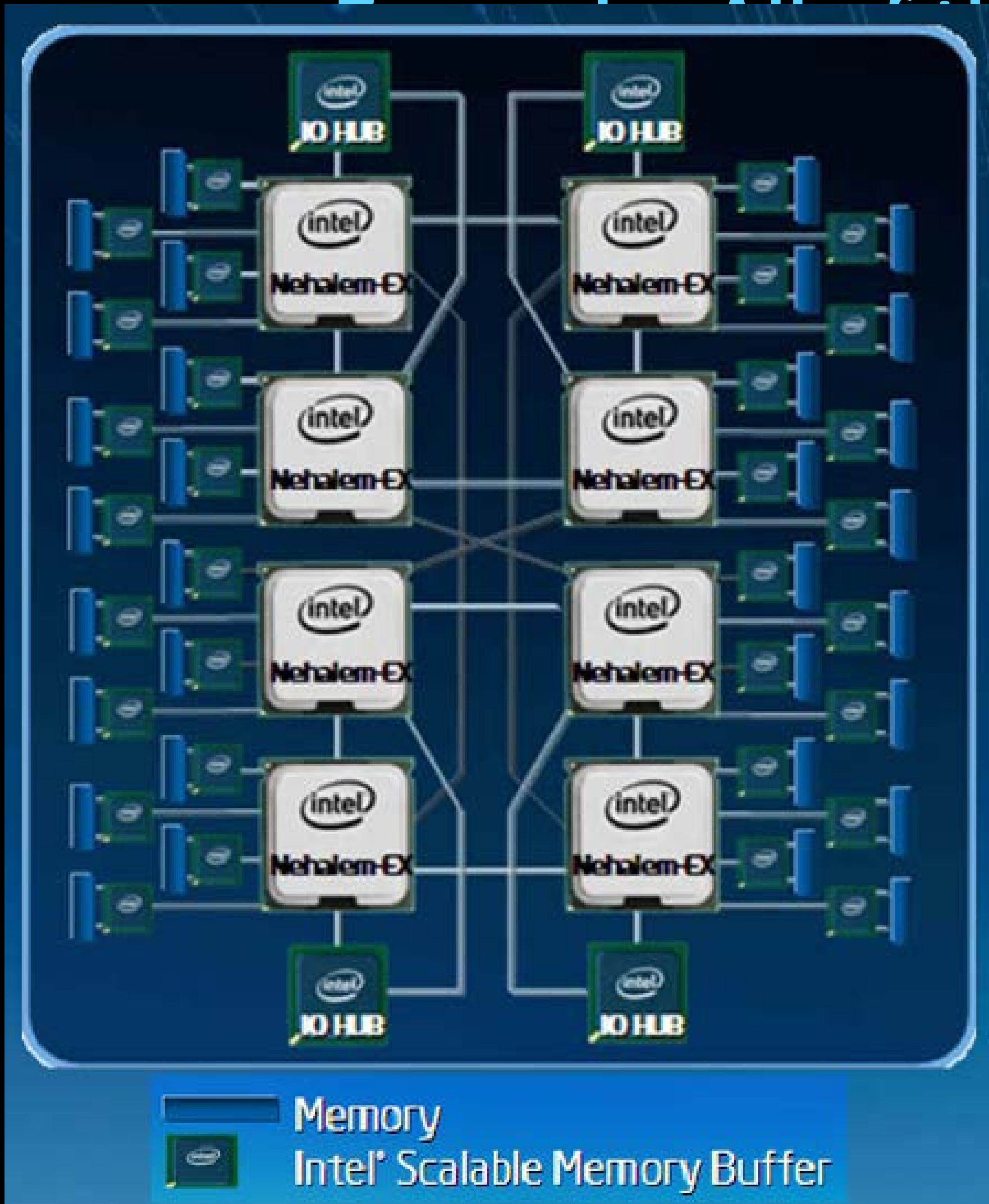
Inside the Processor



Hyperthreading

	Multi-Core vs.	Multi-Issue vs.	HT
Programs:			
Num. Pipelines:			
Pipeline Width:			

Example: All of the above



Parallel Programming

Q: So lets just all use multicore from now on!

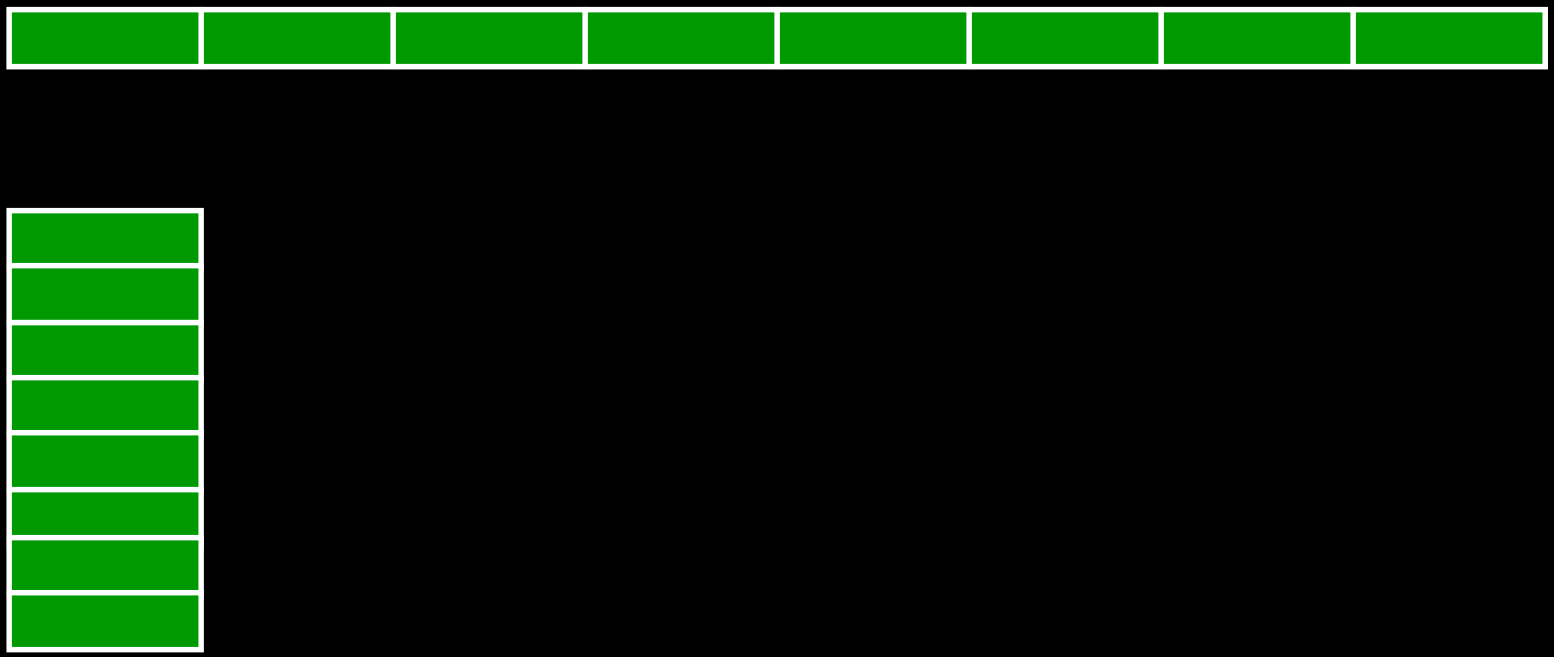
A: Software must be written as parallel program

Multicore difficulties

- Partitioning work
- Coordination & synchronization
- Communications overhead
- Balancing load over cores
- How do you write parallel programs?
  - ... without knowing exact underlying architecture?

Work Partitioning

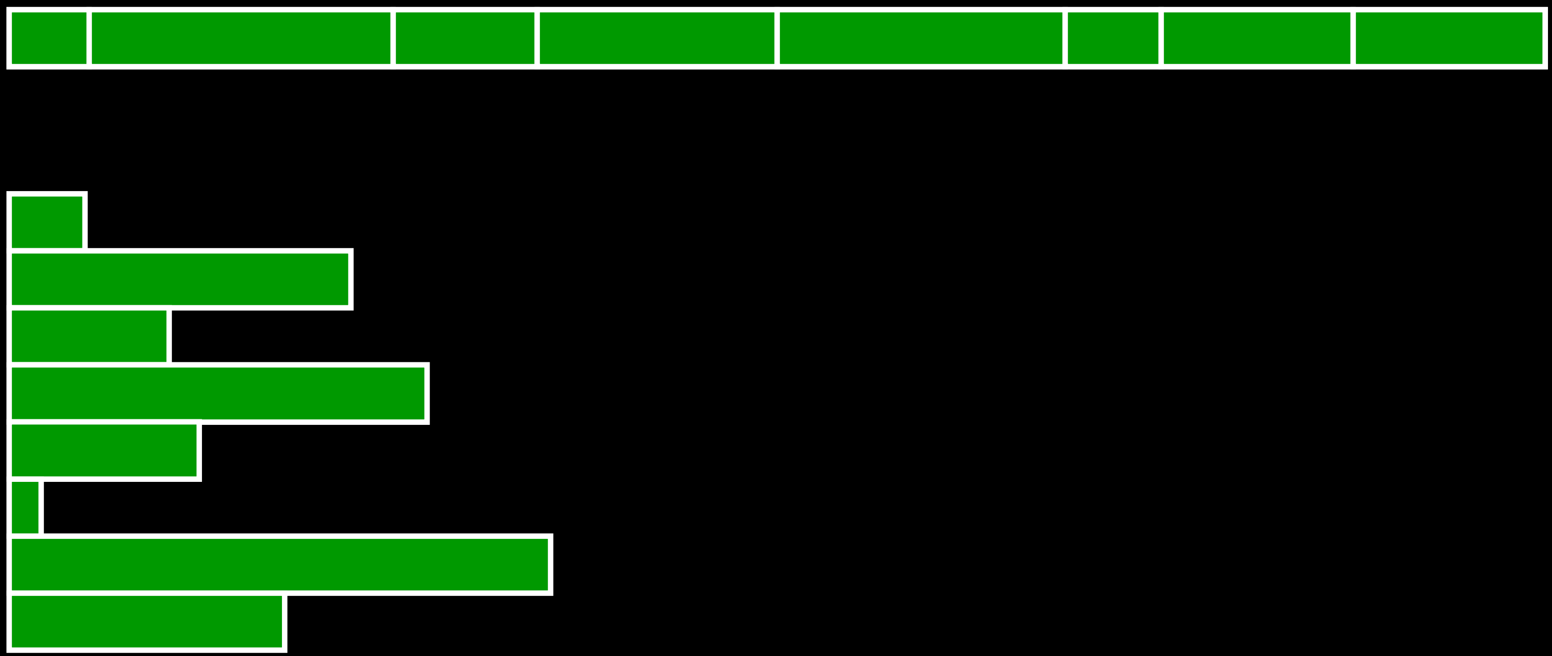
Partition work so all cores have something to do



Load Balancing

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

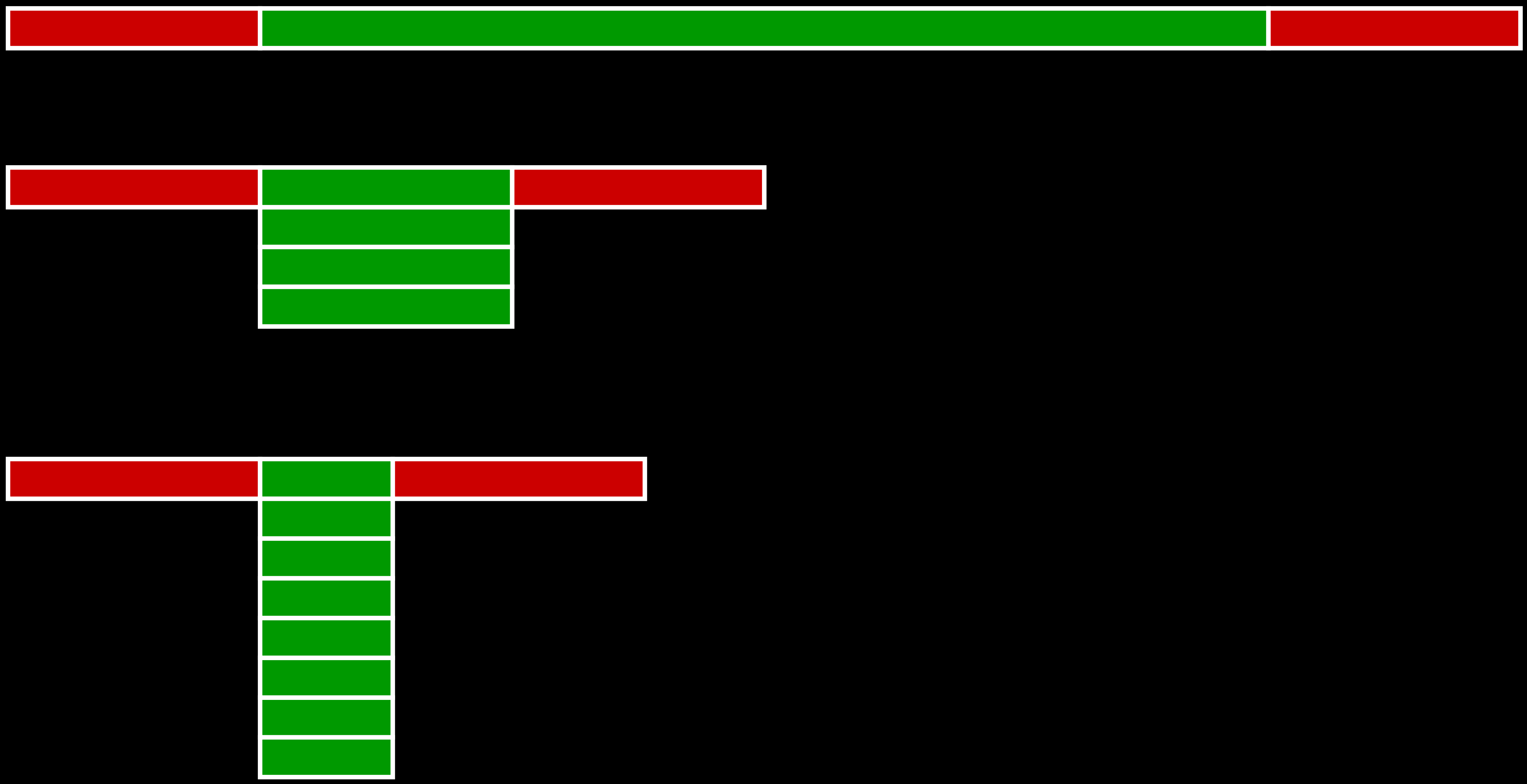
step 3: combine all results

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



Parallel Programming

Q: So lets just all use multicore from now on!

Multicore difficulties

- Partitioning work
- Coordination & synchronization
- Communications overhead
- Balancing load over cores
- How do you write parallel programs?
  - ... without knowing exact underlying architecture?