

# Multicore and Parallel Processing

**Hakim Weatherspoon**

**CS 3410, Spring 2012**

Computer Science

Cornell University

# Administrivia

---

FlameWar Games Night Next Friday, April 27<sup>th</sup>

- 5pm in Upson B17
- Please come, eat, drink and have fun

No Lab4 or Lab Section *next* week!

# Administrivia

---

PA3: FlameWar is due next Monday, April 23<sup>rd</sup>

- The goal is to have fun with it
- Recitations today will talk about it

HW6 Due next Tuesday, April 24<sup>th</sup>

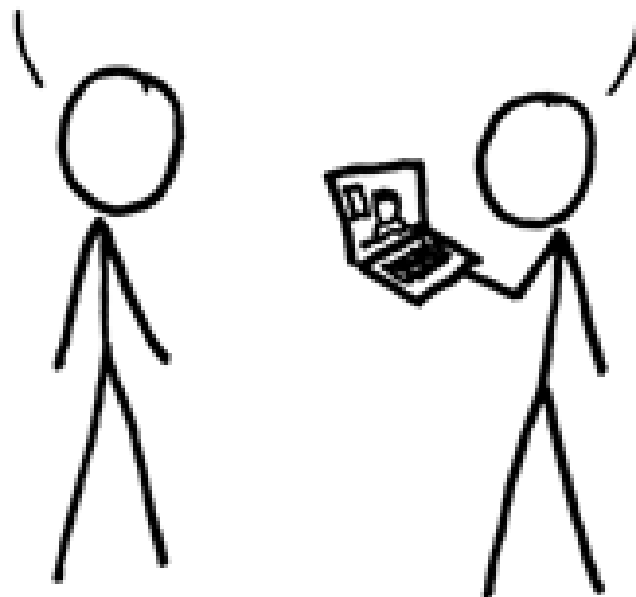
Prelim3 next Thursday, April 26<sup>th</sup>

# 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 =  
affected execution time

---

amount of improvement

+ 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

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

# Scaling Example

---

Workload: sum of 10 scalars, and  $10 \times 10$  matrix sum

- Speed up from 10 to 100 processors?

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

10 processors

100 processors

Assumes load can be balanced across processors

# Scaling Example

---

What if matrix size is  $100 \times 100$ ?

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

10 processors

100 processors

Assuming load balanced



# 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**?

A: Multiple issue pipeline

- Start multiple instructions per clock cycle in duplicate stages

# 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?

A: It doesn't.

→ Simple HW, assumes compiler avoids hazards

Example: Static Dual-Issue 32-bit MIPS

- Instructions come in pairs (64-bit aligned)
  - One ALU/branch instruction (or nop)
  - One load/store instruction (or nop)

# 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
Loop:	nop	lw    \$t0, 0(\$s1)	1
	addi  \$s1, \$s1, -4	nop	2
	addu  \$t0, \$t0, \$s2	nop	3
	bne   \$s1, \$zero, Loop	sw    \$t0, 4(\$s1)	4

# 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
	<b>addi \$s1, \$s1, +8</b>	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

# 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

# Does Multiple Issue Work?

---

Q: Does multiple issue / ILP work?

A: Kind of... but not as much as we'd like

Limiting factors?

- Programs dependencies
- Hard to detect dependencies → be conservative
  - e.g. Pointer Aliasing: `A[0] += 1; B[0] *= 2;`
- Hard to expose parallelism
  - Can only issue a few instructions ahead of PC
- Structural limits
  - Memory delays and limited bandwidth
- Hard to keep pipelines full

# Power Efficiency

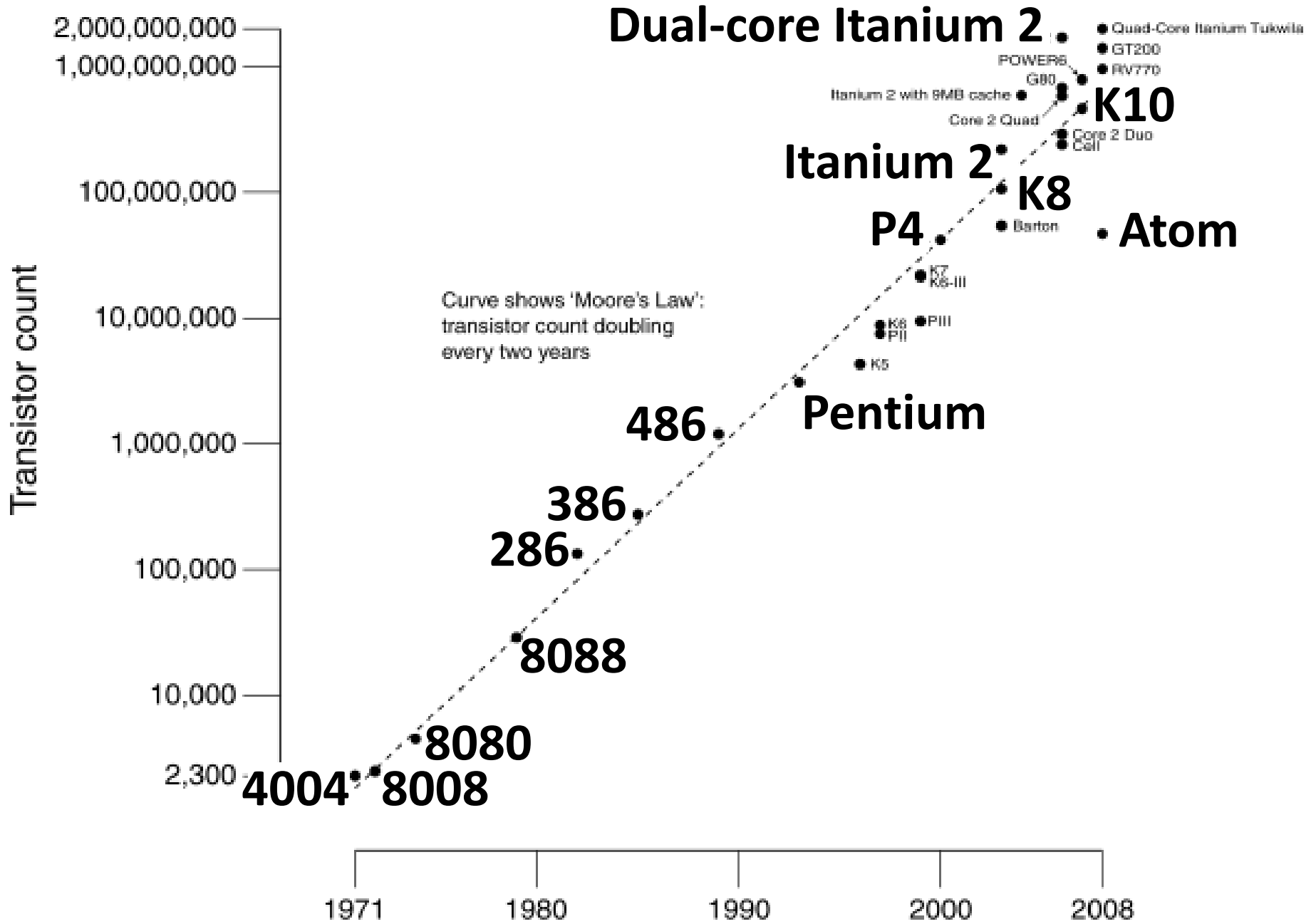
Q: Does multiple issue / ILP cost much?

A: Yes.

→ Dynamic issue and speculation requires power

CPU	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

→ Multiple simpler cores may be better?



# Why Multicore?

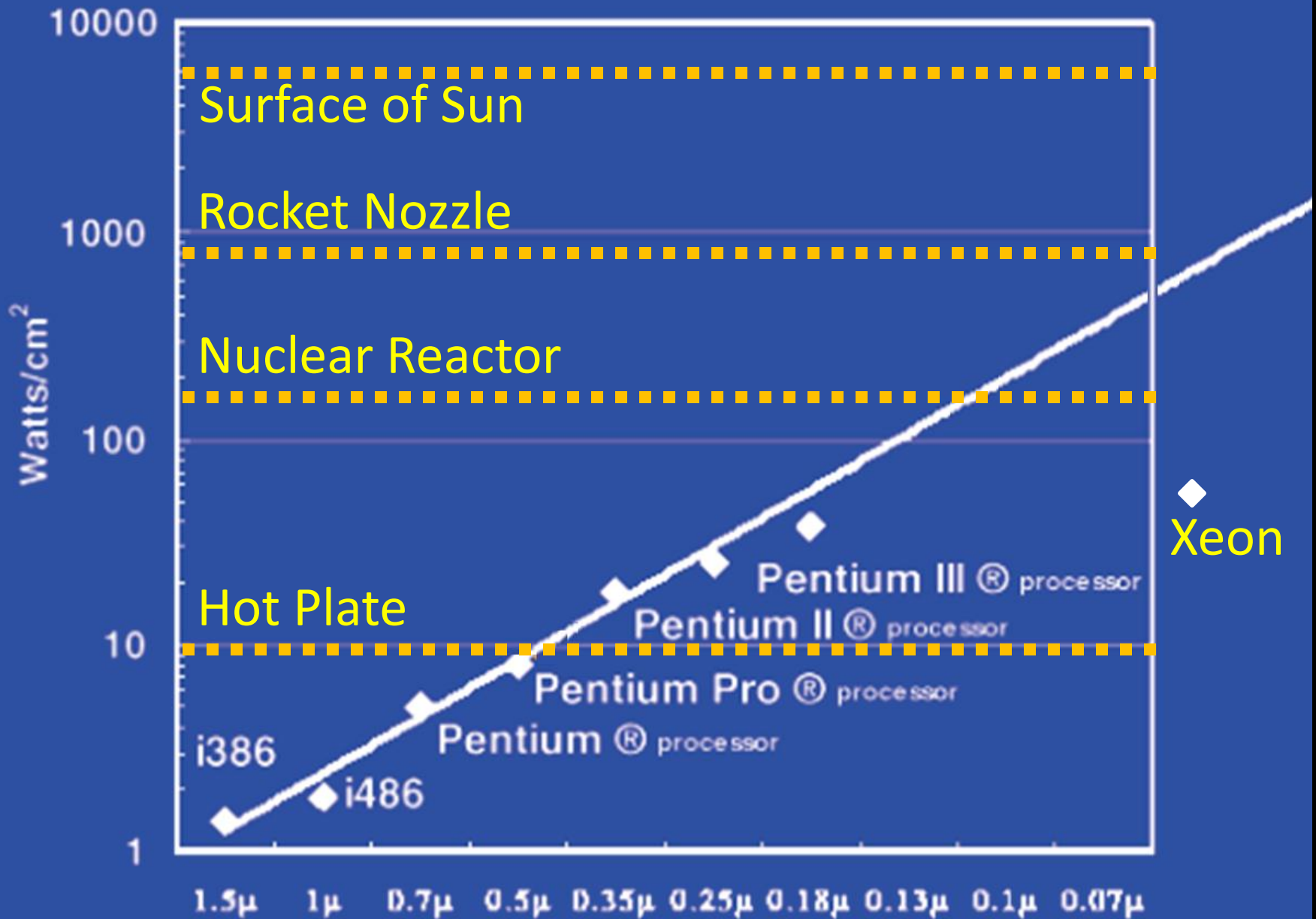
---

## 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<sup>2</sup> \* frequency

In practice: Power ~ voltage<sup>3</sup>

Reducing voltage helps (a lot)

... so does reducing clock speed

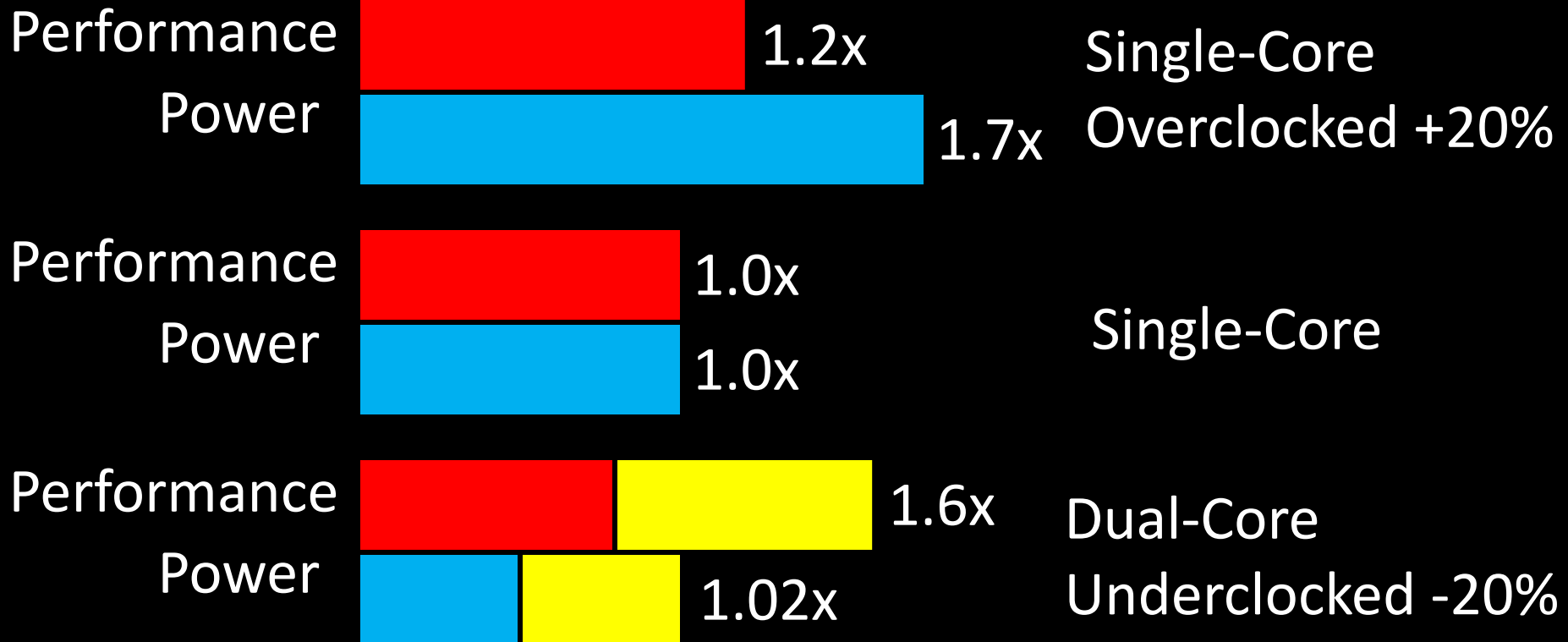
Better cooling helps

The **power wall**

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

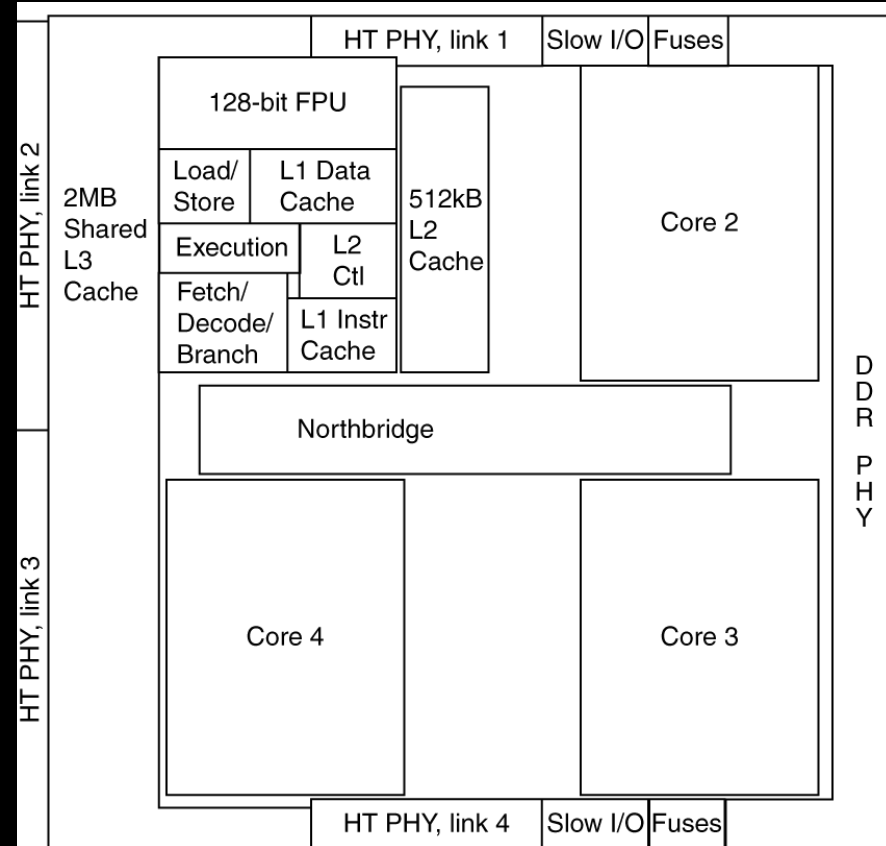
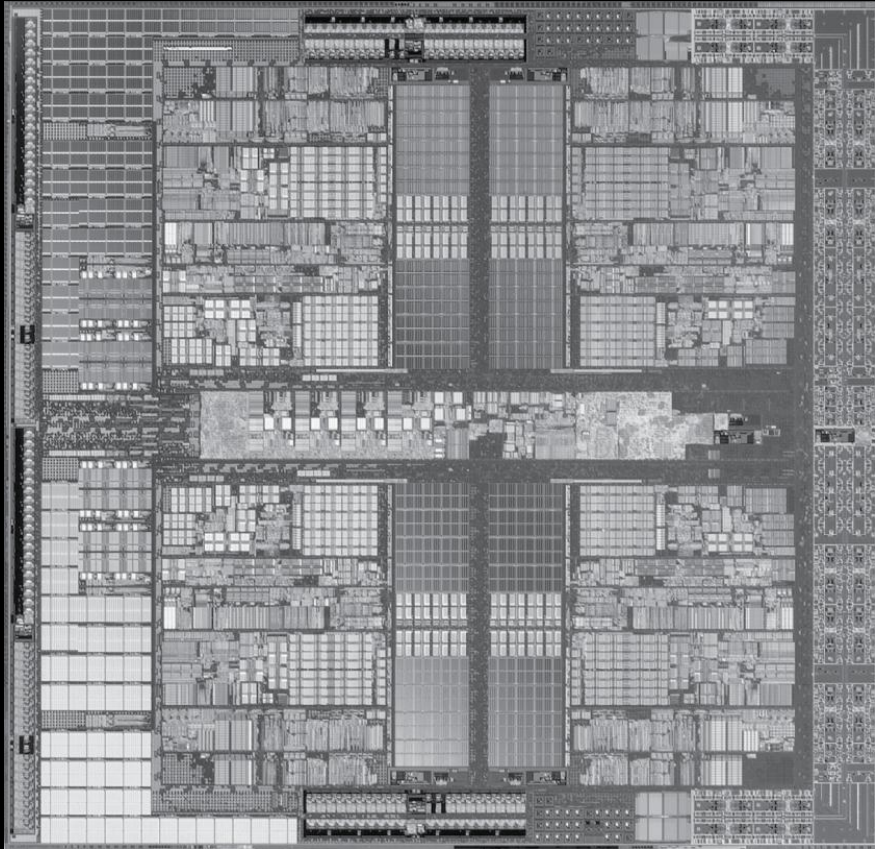
# Why Multicore?

---



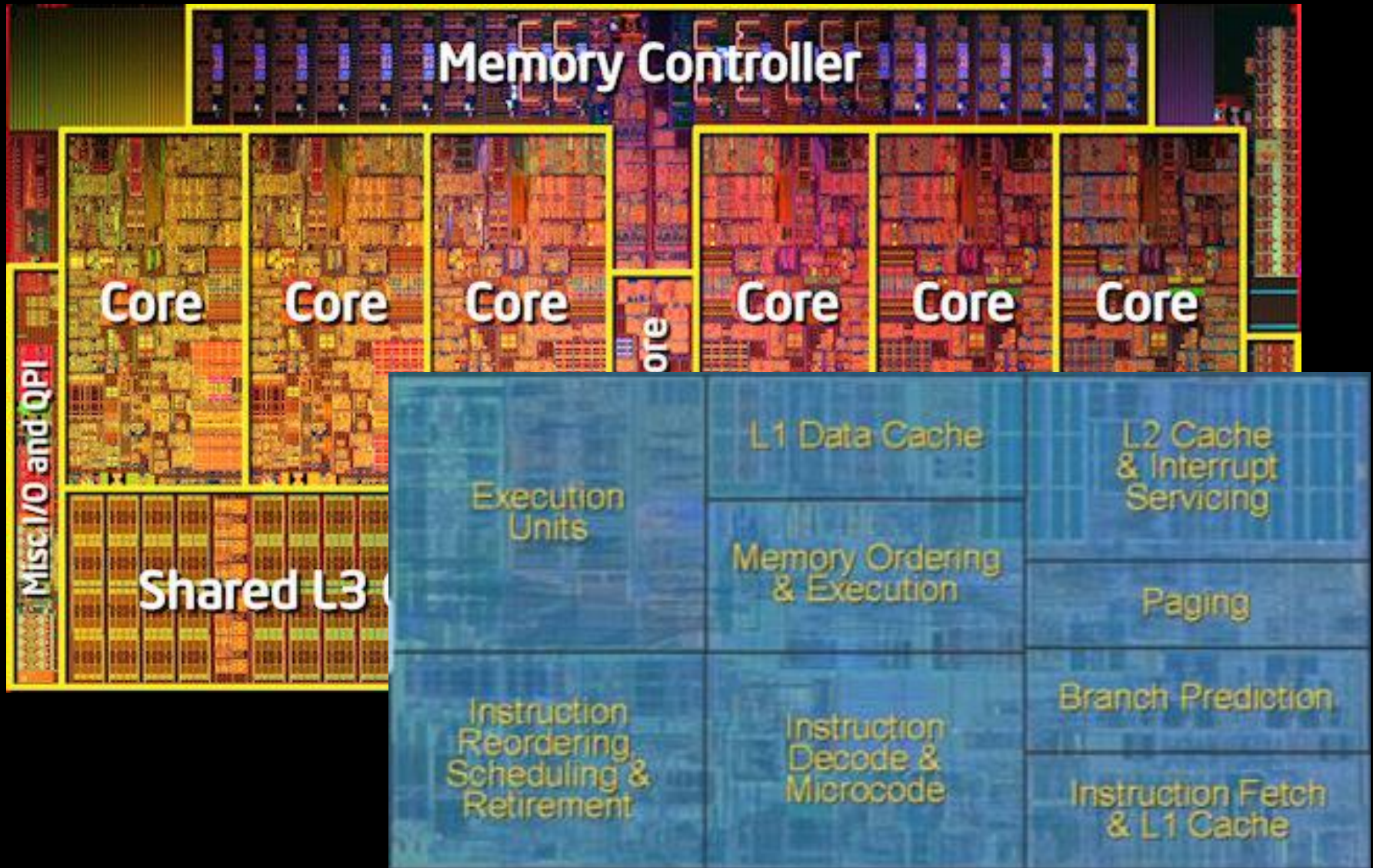
# Inside the Processor

## AMD Barcelona Quad-Core: 4 processor cores



# Inside the Processor

## Intel Nehalem Hex-Core



# Hyperthreading

---

## Multi-Core vs. Multi-Issue <sup>vs. HT</sup>

Programs:

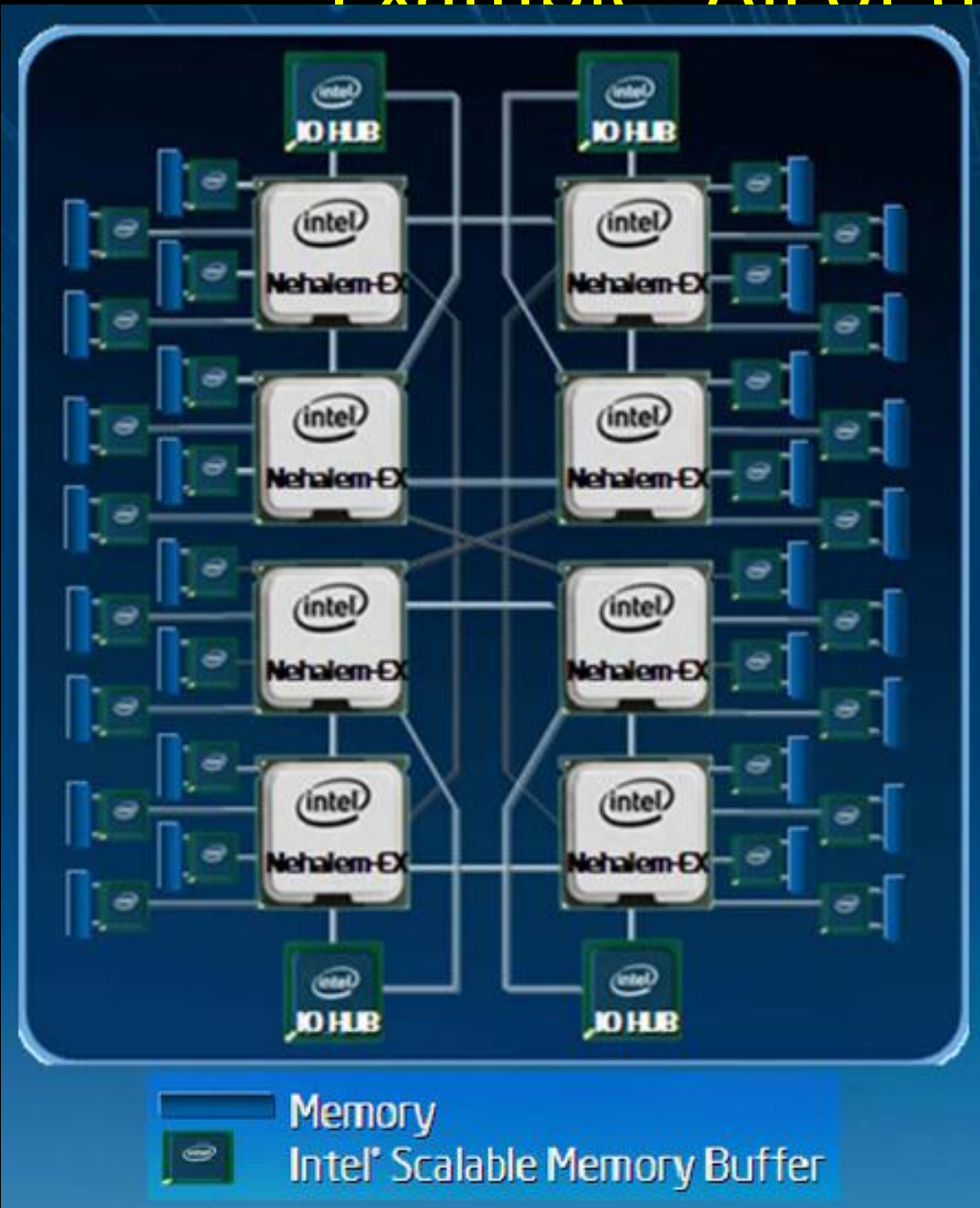
Num. Pipelines:

Pipeline Width:

### Hyperthreads (Intel)

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

# 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

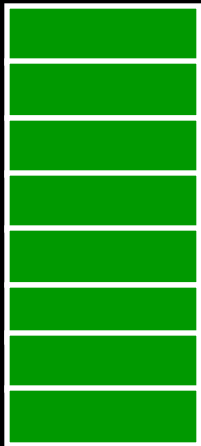
## Multicore difficulties

- 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

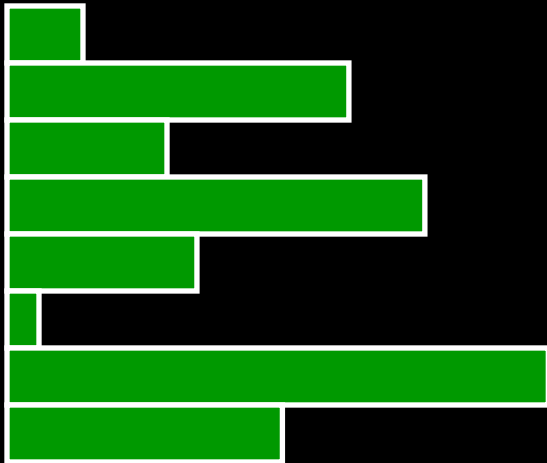




# Load Balancing

## 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

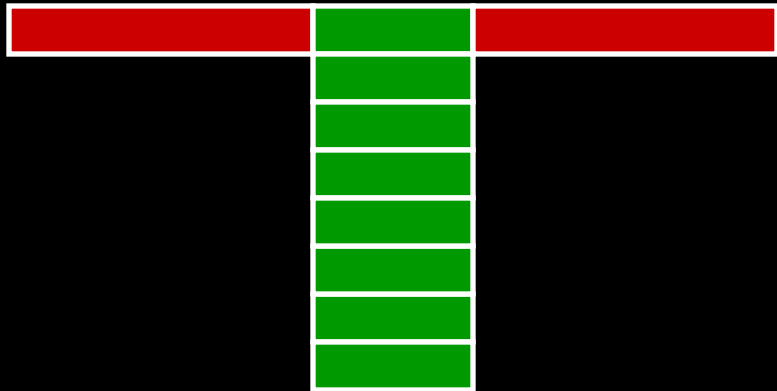
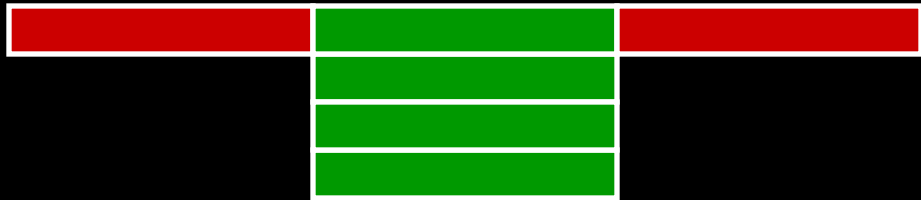
Recall: **Amdahl's Law**

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!

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?