

Multicore and Parallelism

Prof. Hakim Weatherspoon

CS 3410, Spring 2015

Computer Science

Cornell University

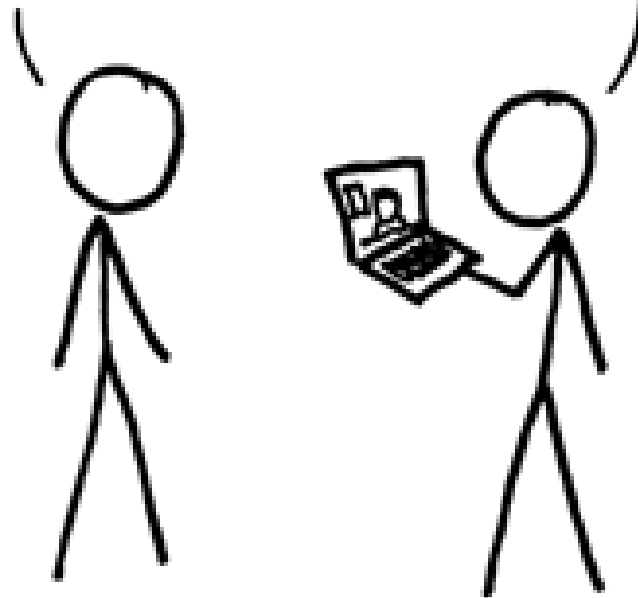
P & H Chapter 4.10, 1.7, 1.8, 5.10, 6

xkcd/619

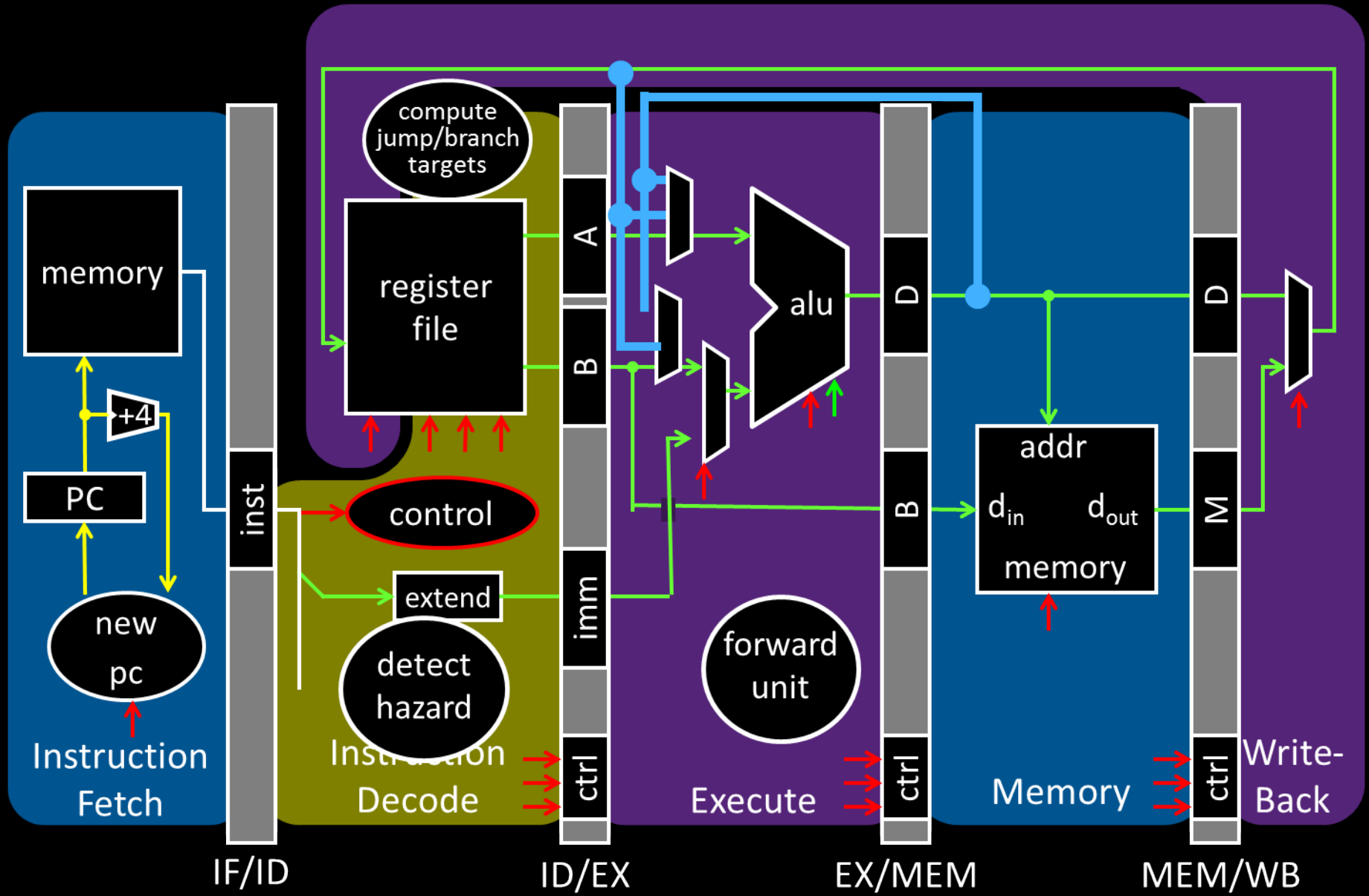
IT TOOK A LOT OF WORK, BUT THIS LATEST LINUX PATCH ENABLES SUPPORT FOR MACHINES WITH 4,096 CPUs, UP FROM THE OLD LIMIT OF 1,024.

DO YOU HAVE SUPPORT FOR SMOOTH FULL-SCREEN FLASH VIDEO YET?

NO, BUT WHO USES THAT?

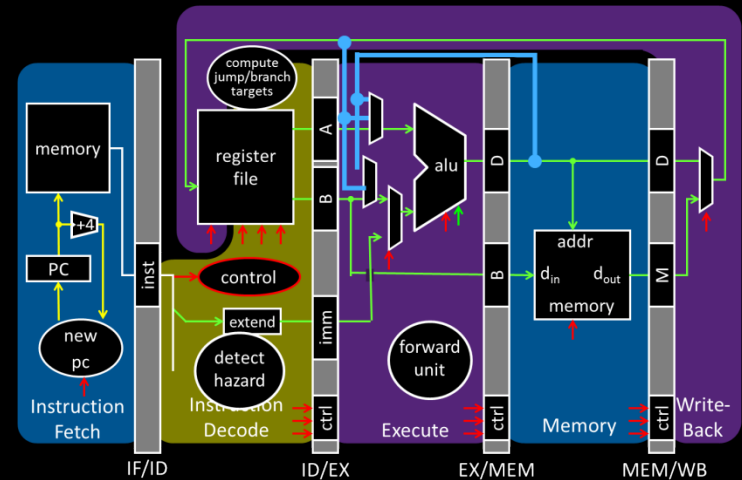
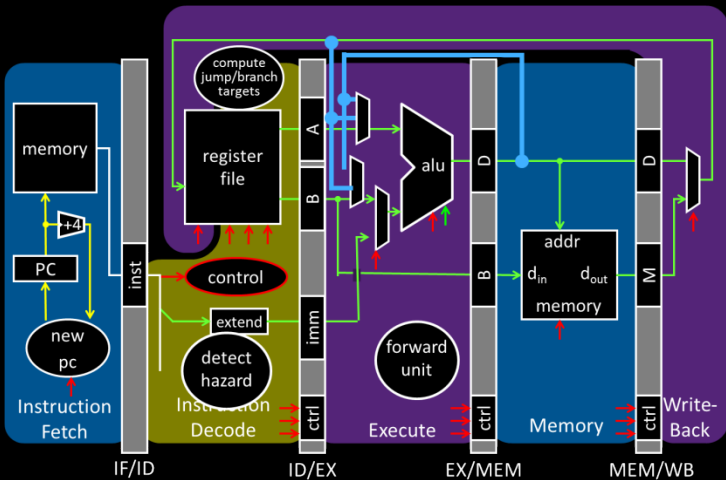
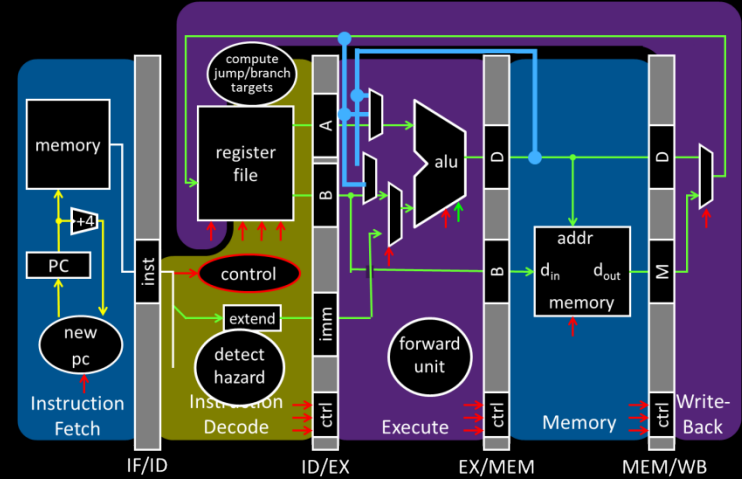
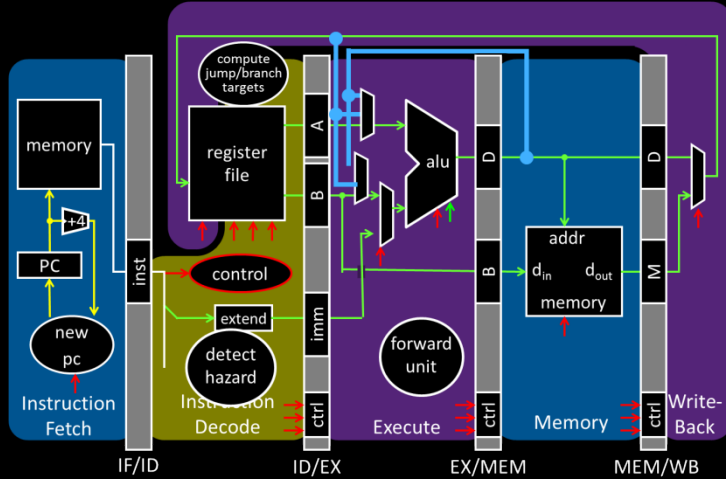


Big Picture: Multicore and Parallelism



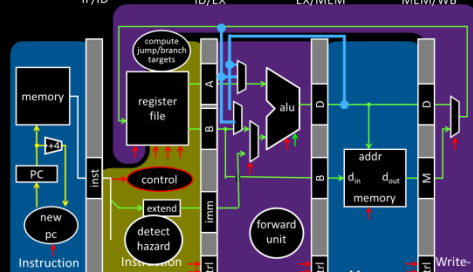
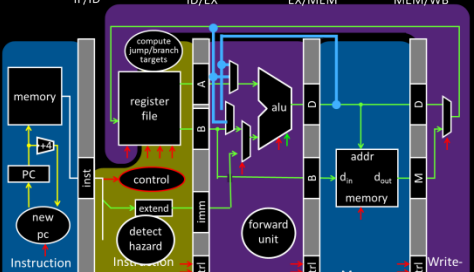
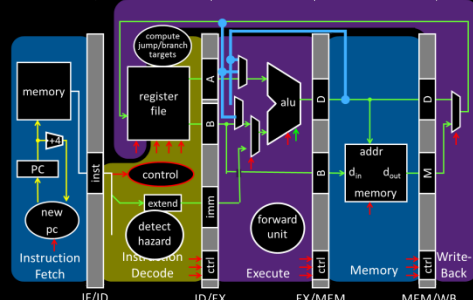
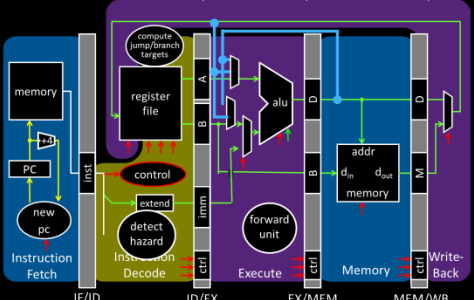
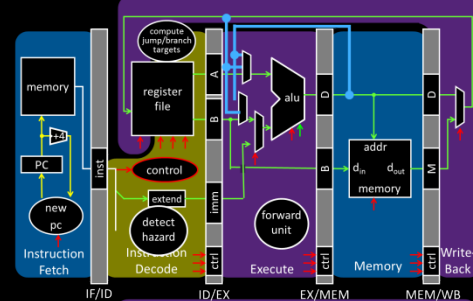
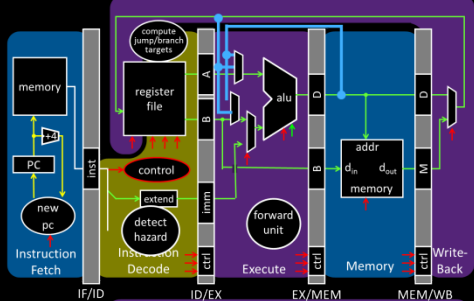
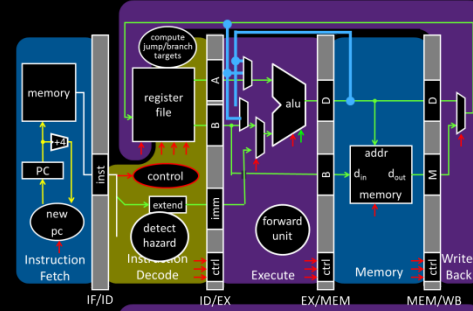
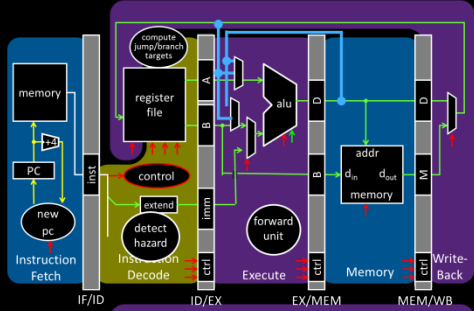
Big Picture: Multicore and Parallelism

Why do I need *four* computing cores on my phone?!



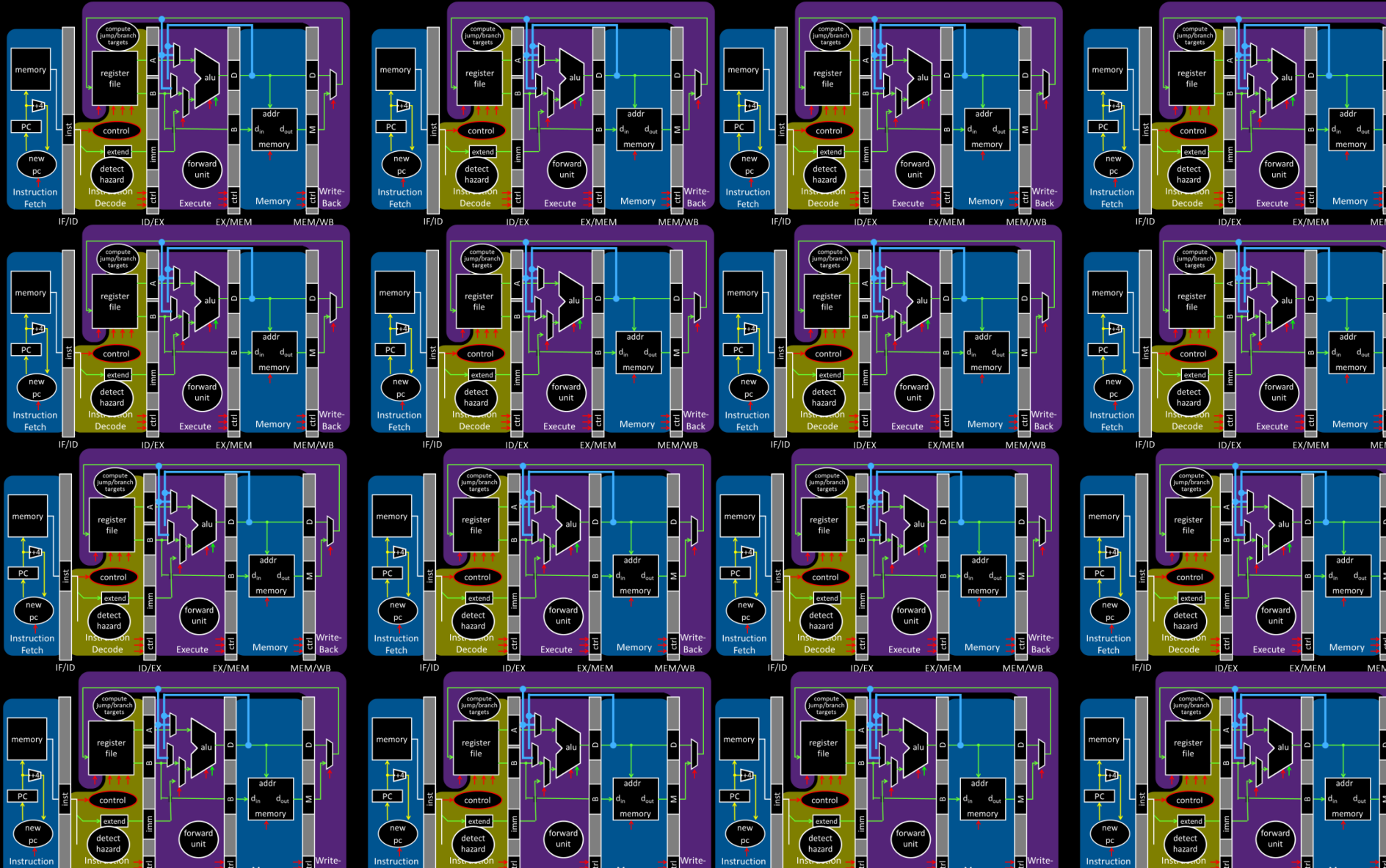
Big Picture: Multicore and Parallelism

Why do I need *eight* computing cores on my phone?!



Big Picture: Multicore and Parallelism

Why do I need *sixteen* computing cores on my phone?!



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

- Multiply can be parallized
- How much improvement do we need in the multiply performance to get 5x overall improvement?
(a) 2x (b) 10x (c) 100x (d) 1000x (e) not possible

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

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

$$20 = \frac{80}{n} + 20 \quad - \text{ Can't be done!}$$

Scaling Example

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

- Speed up from 10 to 100 processors?

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

10 processors

- Time = $100/10 \times t_{\text{add}} + 10 \times t_{\text{add}} = 20 \times t_{\text{add}}$
- Speedup = $110/20 = 5.5$

100 processors

- Time = $100/100 \times t_{\text{add}} + 10 \times t_{\text{add}} = 11 \times t_{\text{add}}$
- Speedup = $110/11 = 10$

Assumes load can be balanced across processors

Scaling Example

What if matrix size is 100×100 ?

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

10 processors

- Time = $10 \times t_{\text{add}} + 10000/10 \times t_{\text{add}} = 1010 \times t_{\text{add}}$
- Speedup = $10010/1010 = 9.9$

100 processors

- Time = $10 \times t_{\text{add}} + 10000/100 \times t_{\text{add}} = 110 \times t_{\text{add}}$
- Speedup = $10010/110 = 91$

Assuming load balanced

Takeaway

Unfortunately, we cannot not obtain unlimited scaling (speedup) by adding unlimited parallel resources, eventual performance is dominated by a component needing to be executed sequentially.

Amdahl's Law is a caution about this diminishing return

Announcements

- HW2 Review Sessions!
 - Saturday, April 18th, Hollister B14@7pm
 - Tuesday, April 21st, Hollister B14@7pm
- Lab3 is due yesterday!
 - Wednesday, April 15th
 - *In theory, theory and practice are the same. In practice, they are not. –Albert Einstein*
- 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

Goals for Today

How to improve System Performance?

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

Next 2 lectures:

- Cache coherency, synchronization, Concurrency, and programming

How to improve performance?

We have looked at

- Pipelining
- To speed up:
 - Deeper pipelining
 - Make the clock run faster
 - Parallelism
 - Not a luxury, a necessity

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

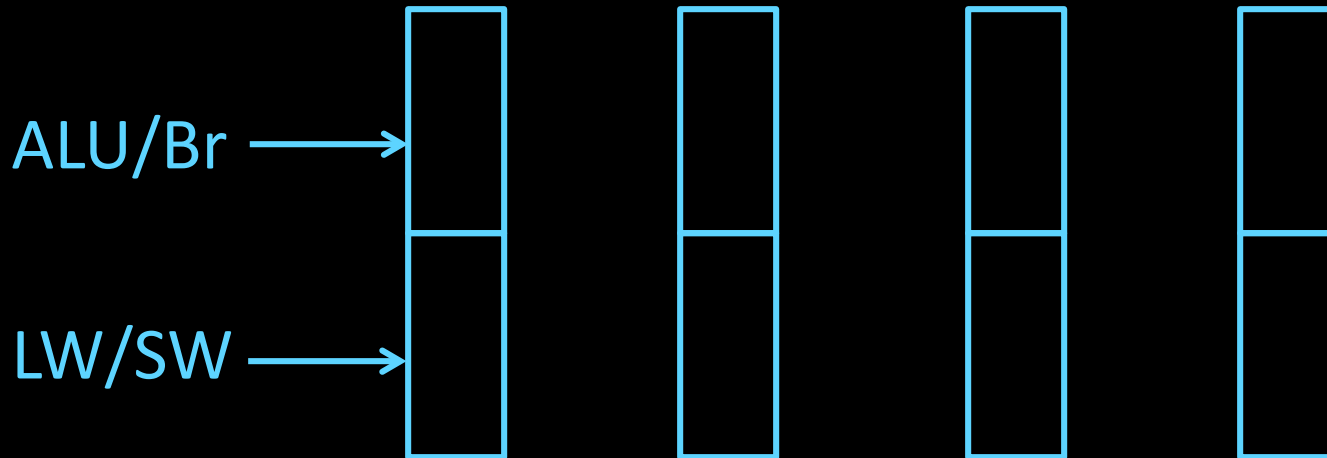
Instruction-Level Parallelism (ILP)

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



Multiple issue pipeline

Static multiple issue

aka Very Long Instruction Word

Decisions made by compiler

Dynamic multiple issue

Decisions made on the fly

Cost: More execute hardware

Reading/writing register files: more ports

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

$$\frac{5 \text{ instructions}}{4 \text{ cycles}} = \text{IPC} = 1.25$$

$$\frac{4 \text{ cycles}}{5 \text{ instructions}} = \text{CPI} = 0.8$$

Speculation

Reorder instructions

To fill the issue slot with useful work

Complicated: exceptions may occur

Optimizations to make it work

Move instructions to fill in nops

Need to track hazards and dependencies

Loop unrolling

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    $t0, -8($s1)     # store A[i]
      sw    $t1, -4($s1)     # store A[i+1]
      addi  $s1, $s1, +8     # increment pointer
      bne   $s1, $s3, Loop   # 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, Loop	nop	6

$$\frac{6 \text{ cycles}}{8 \text{ instructions}} = \text{CPI} = 0.75$$

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    $t0, -8($s1)      # store A[i]
      sw    $t1, -4($s1)      # store A[i+1]
      addi  $s1, $s1, +8      # increment pointer
      bne   $s1, $s3, Loop    # 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

$$\frac{5 \text{ cycles}}{8 \text{ instructions}} = \text{CPI} = 0.625$$

Limits of Static Scheduling

Compiler scheduling for dual-issue MIPS...

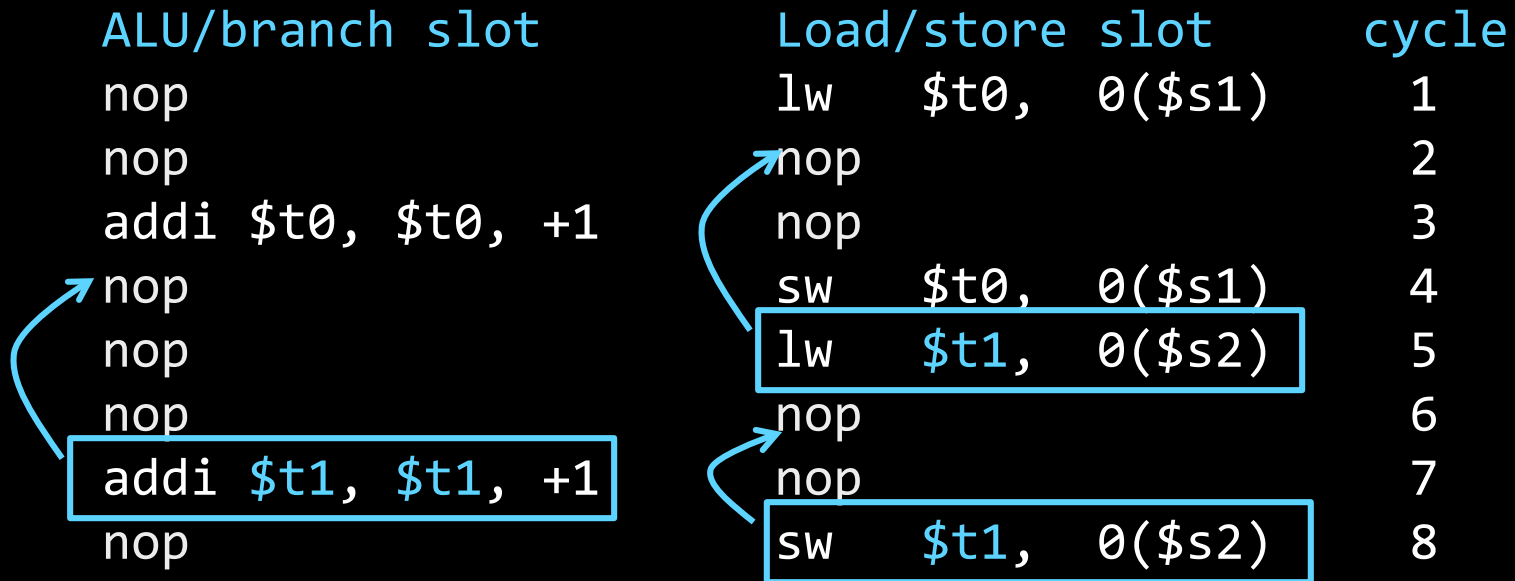
```
lw    $t0, 0($s1)      # load A
addi  $t0, $t0, +1     # increment A
sw    $t0, 0($s1)      # store A
lw    $t0, 0($s2)      # load B
addi  $t0, $t0, +1     # increment B
sw    $t0, 0($s2)      # store B
```

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    $t1, 0($s2)      # store B
```



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    $t1, 0($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

Problem: What if **\$s1 and \$s2** are equal (*aliasing*)? Won't work

Dynamic Multiple Issue

Dynamic Multiple Issue

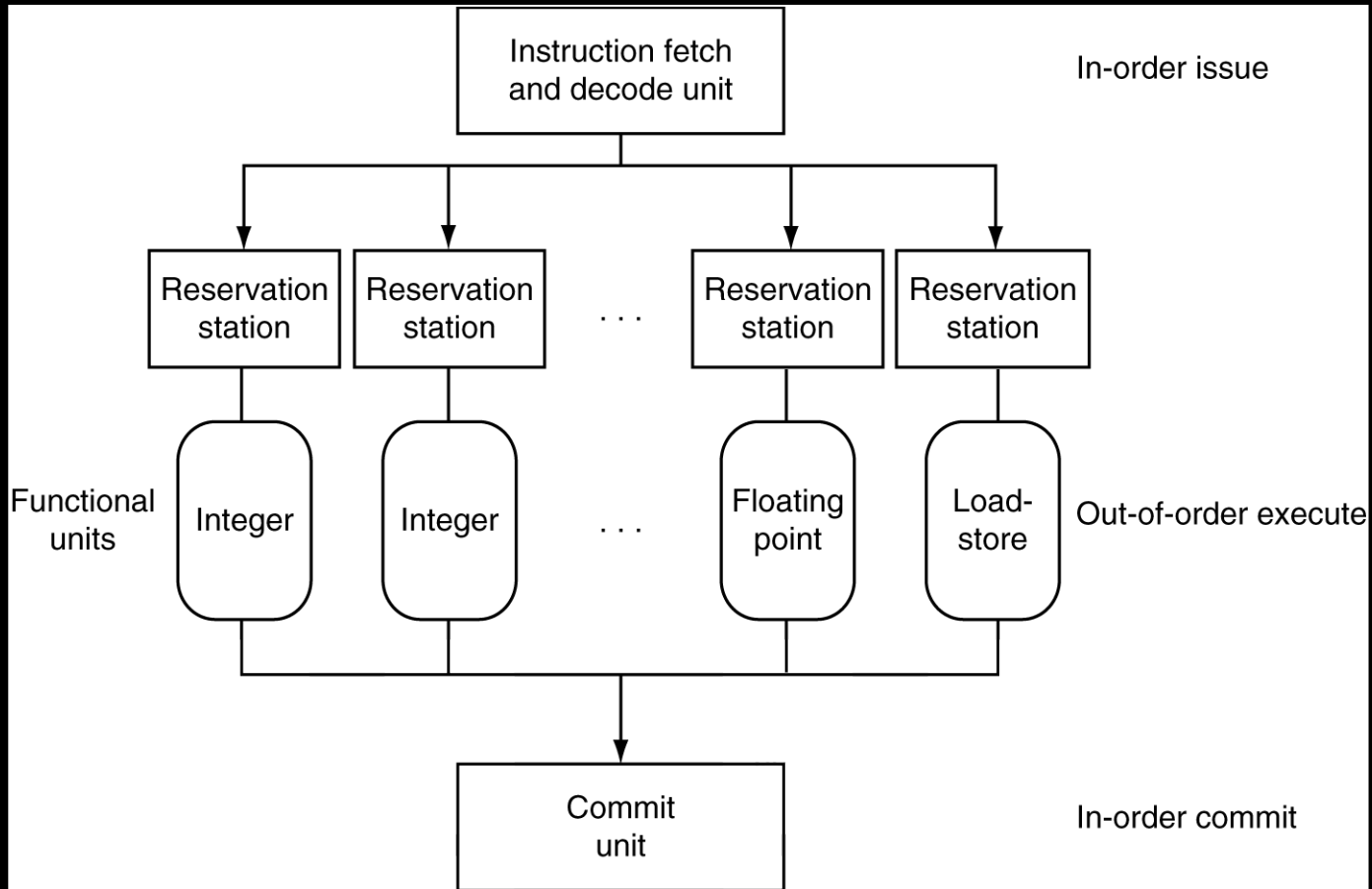
a.k.a. SuperScalar Processor (c.f. Intel)

- 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?

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

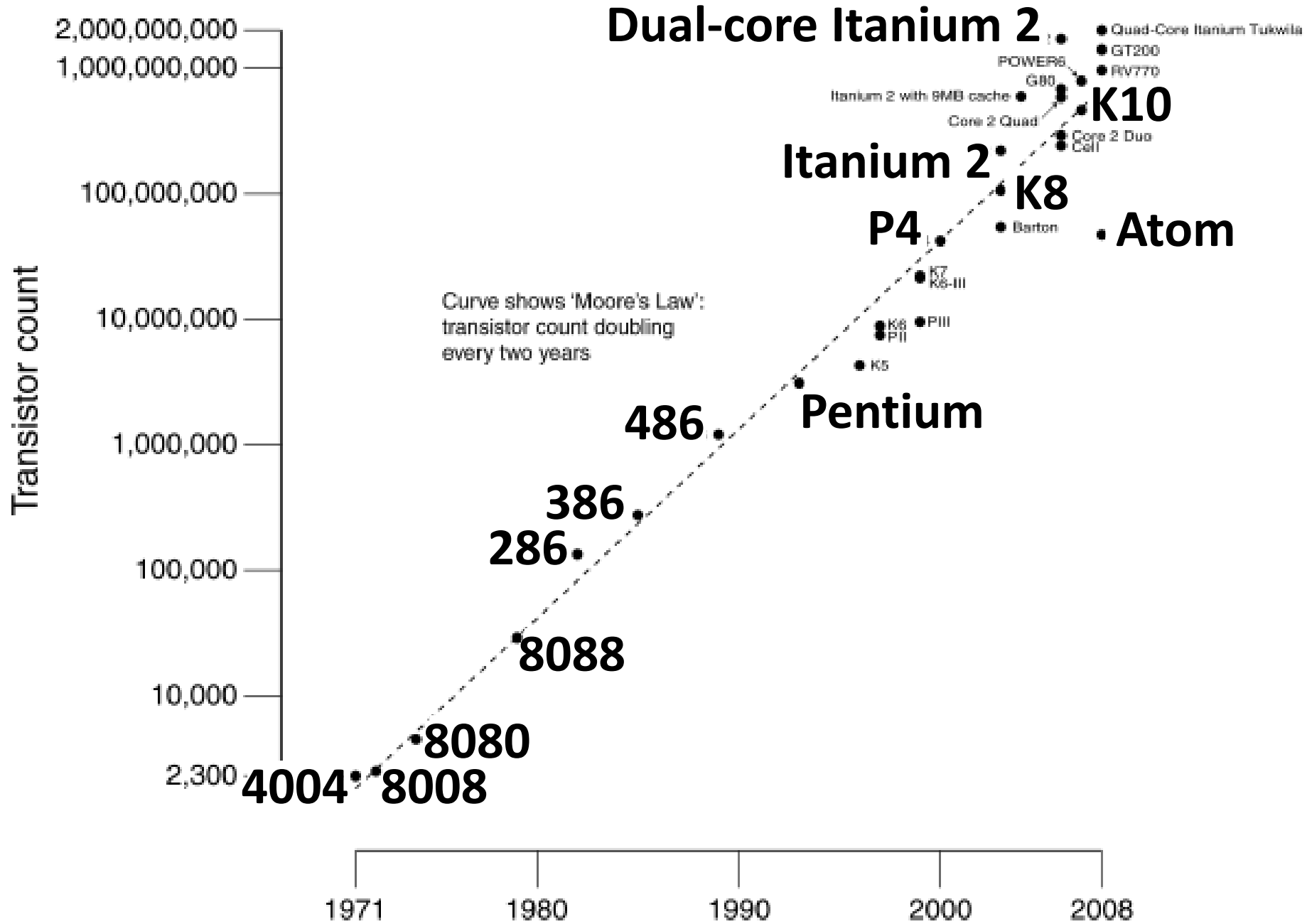
Q: Does multiple issue / ILP cost much?

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
Core i5 Nehal	2010	3300MHz	14	4	Yes	1	87W
Core i5 Ivy Br	2012	3400MHz	14	4	Yes	8	77W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

→ Multiple simpler cores may be better?



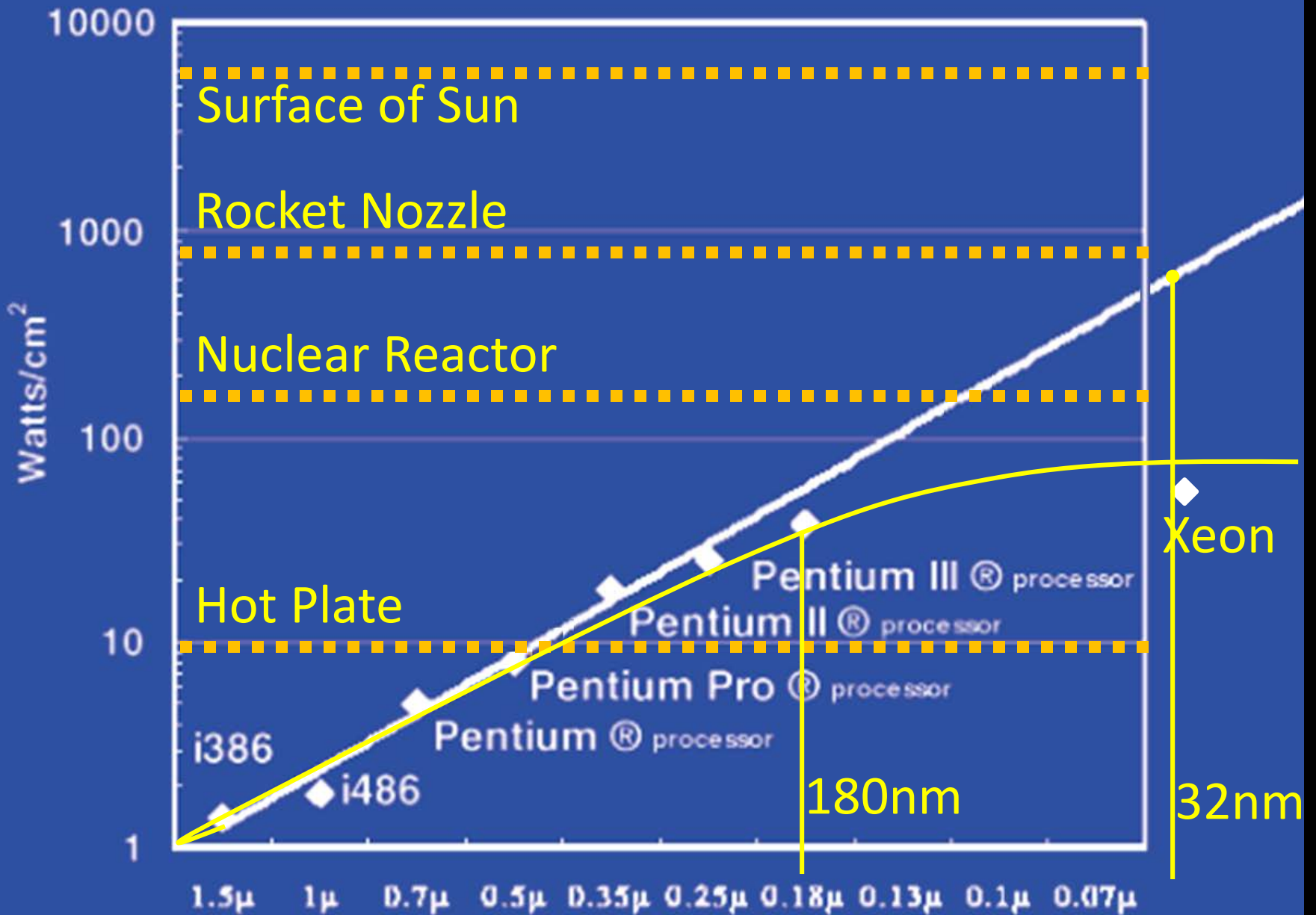
Why Multicore?

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 * voltage² * frequency

In practice: Power ~ voltage³

Lower Frequency

Reducing voltage helps (a lot)

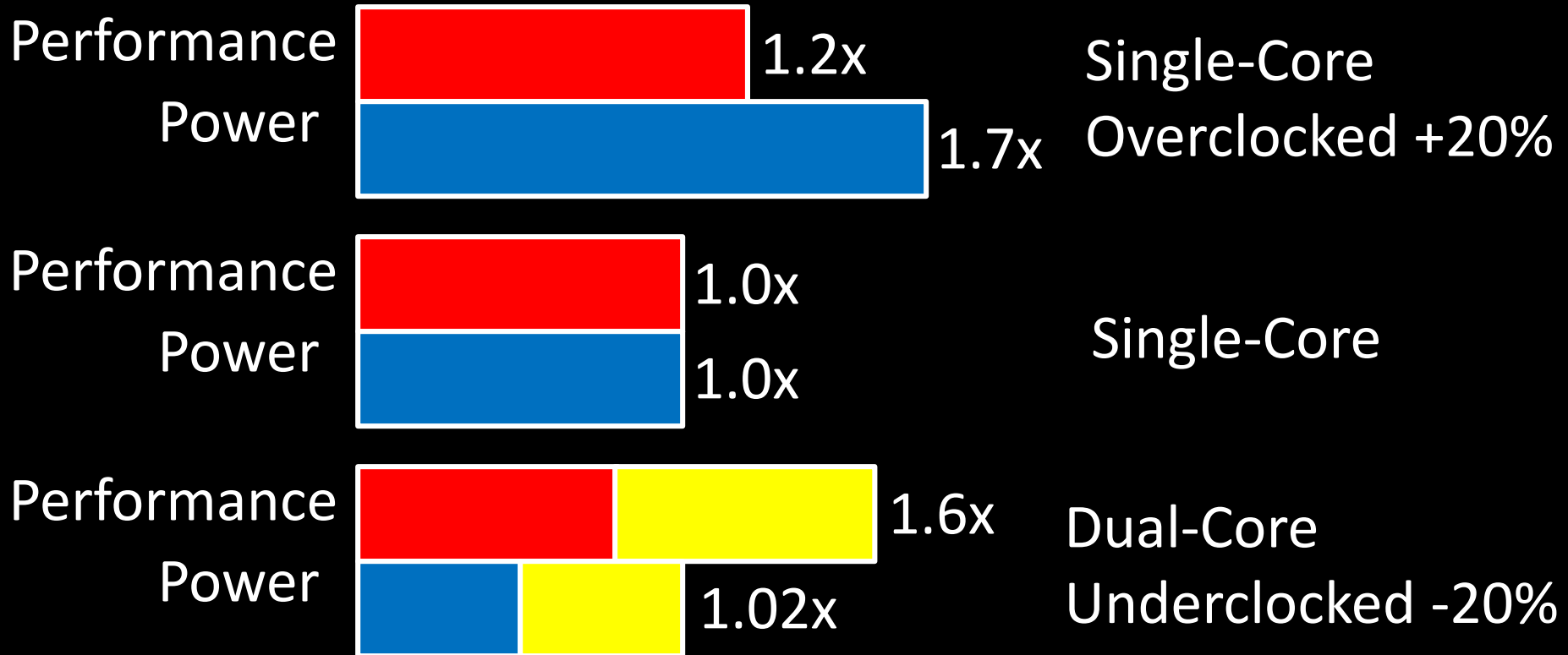
... so does reducing clock speed

Better cooling helps

The power wall

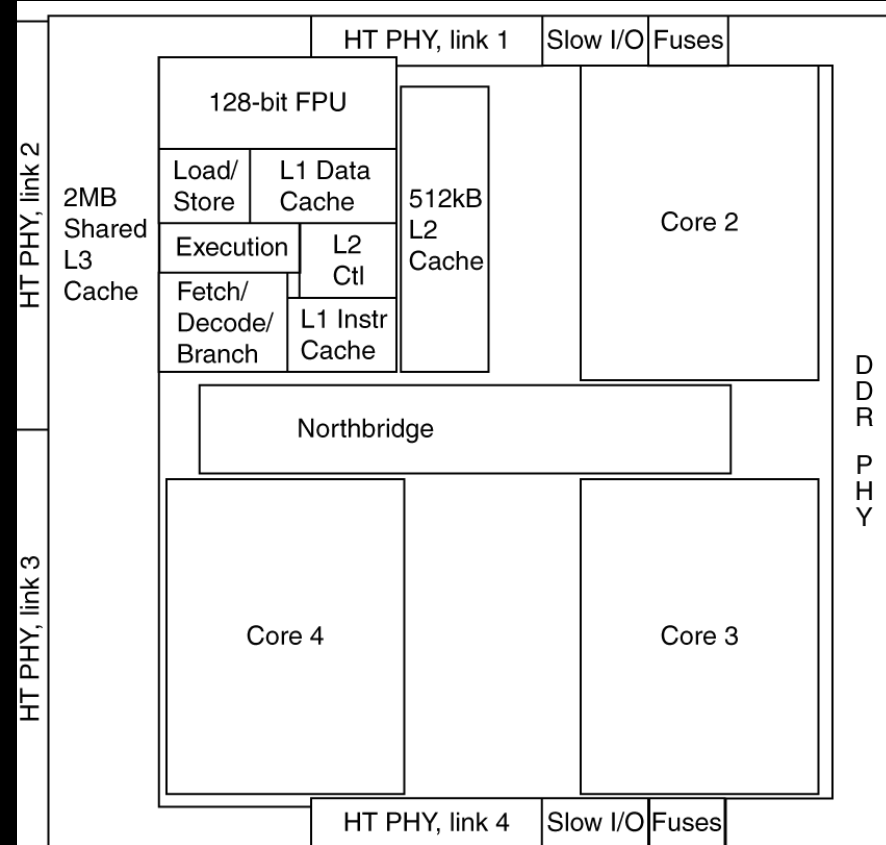
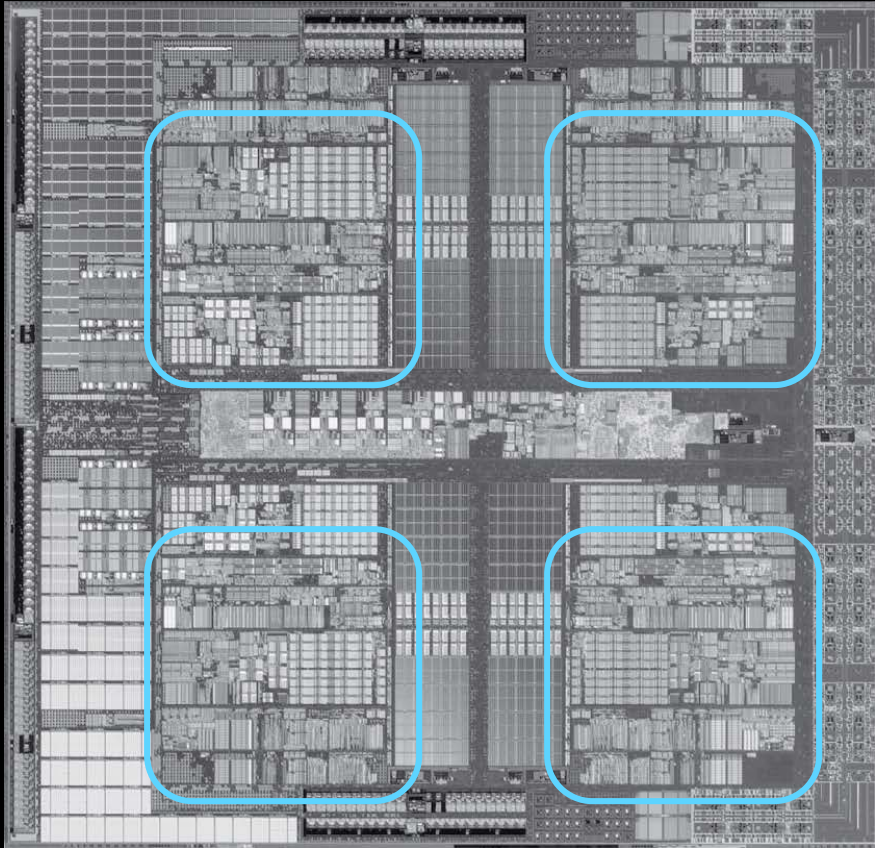
- We can't reduce voltage further
- We can't remove more heat

Why Multicore?

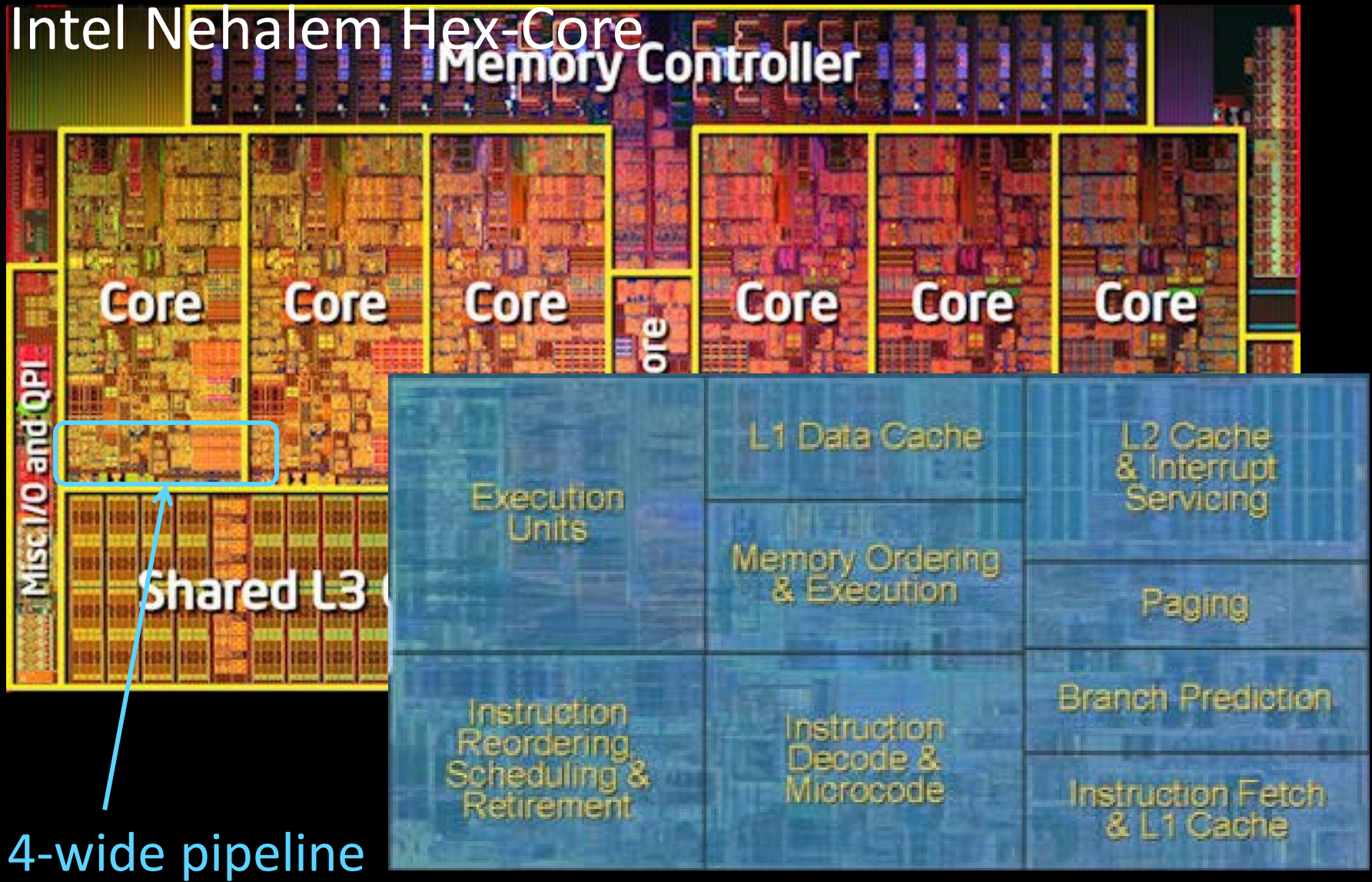


Inside the Processor

AMD Barcelona Quad-Core · 4 processor cores



Inside the Processor



Multicore is a necessity. How can we get better performance?

Hardware multithreading

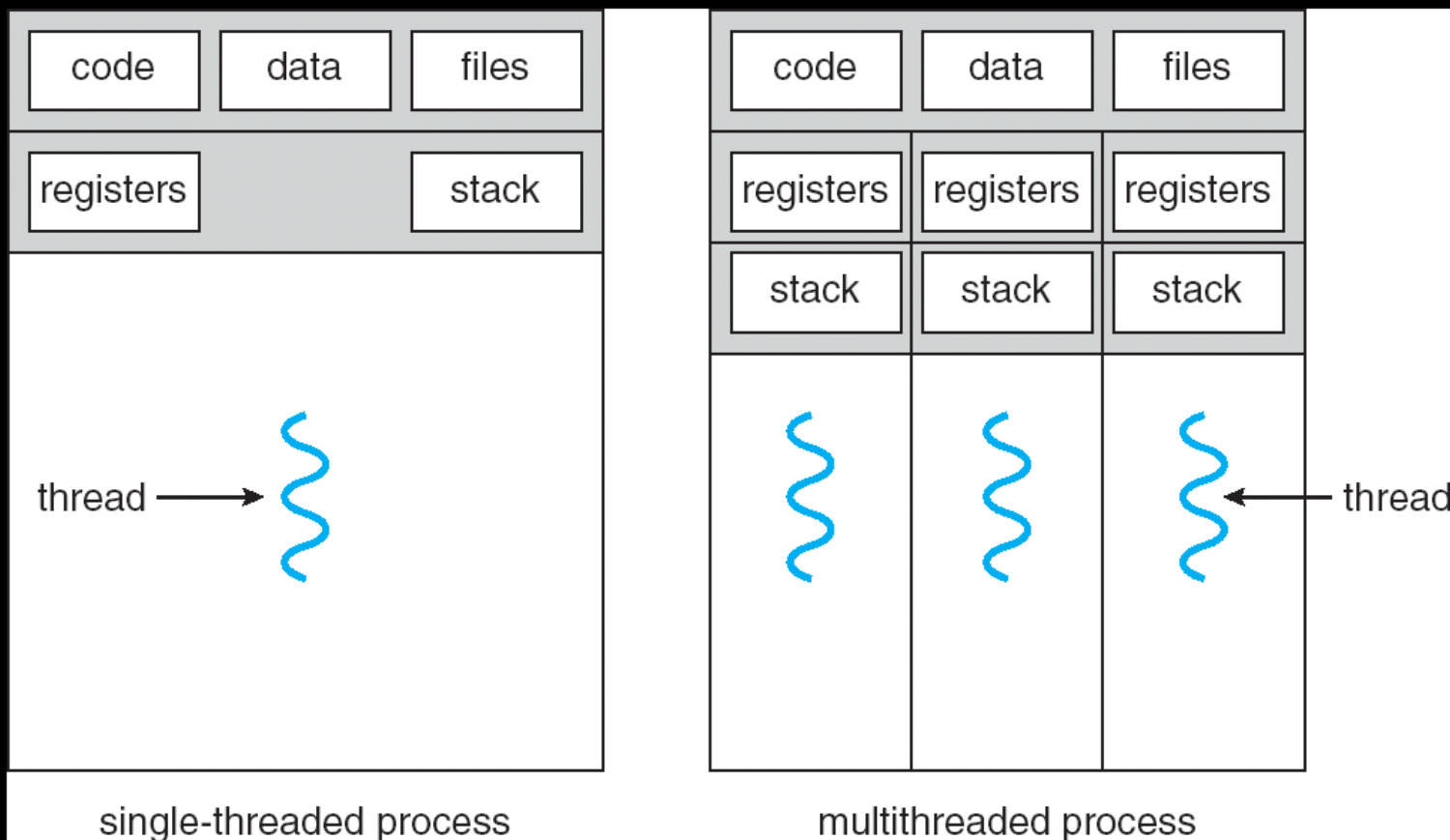
Hardware multithreading

- Increase utilization with low overhead

Switch between hardware threads for stalls

What is a thread?

Process includes multiple threads, code, data and OS state



Hardware multithreading

Fine grained vs. Coarse grained hardware multithreading

Simultaneous multithreading (SMT)

Hyperthreads (Intel simultaneous multithreading)

- Need to hide latency

Hardware multithreading

Fine grained vs. Coarse grained hardware multithreading

Fine grained multithreading

Switch on each cycle

Pros: Can hide very short stalls

Cons: Slows down every thread

Coarse grained multithreading

Switch only on quite long stalls

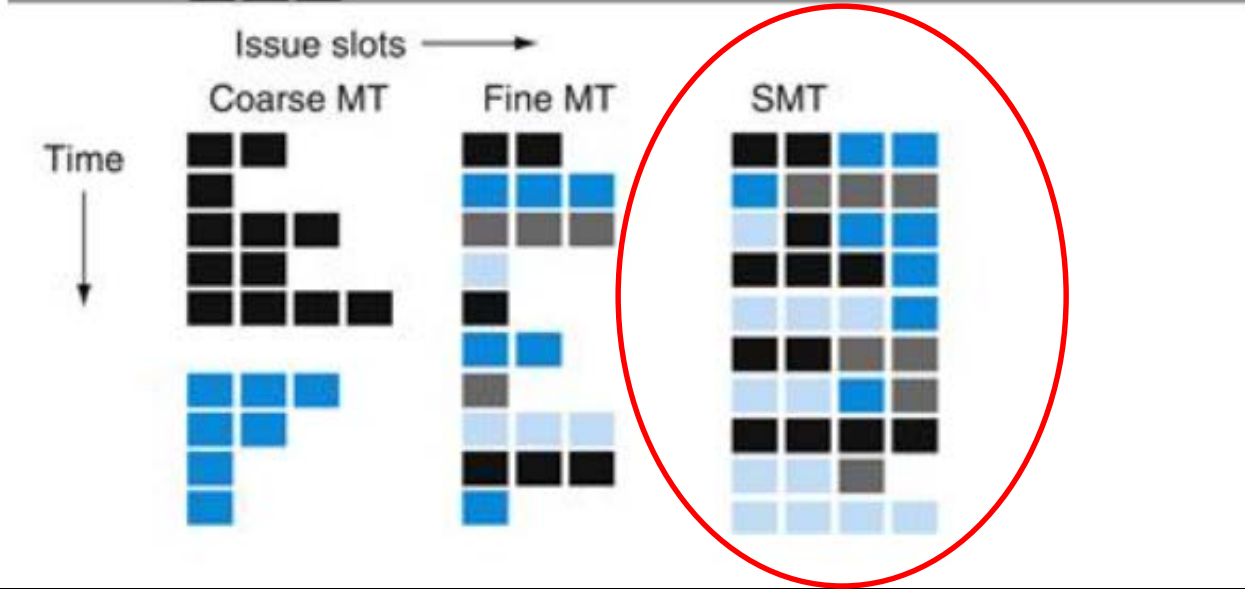
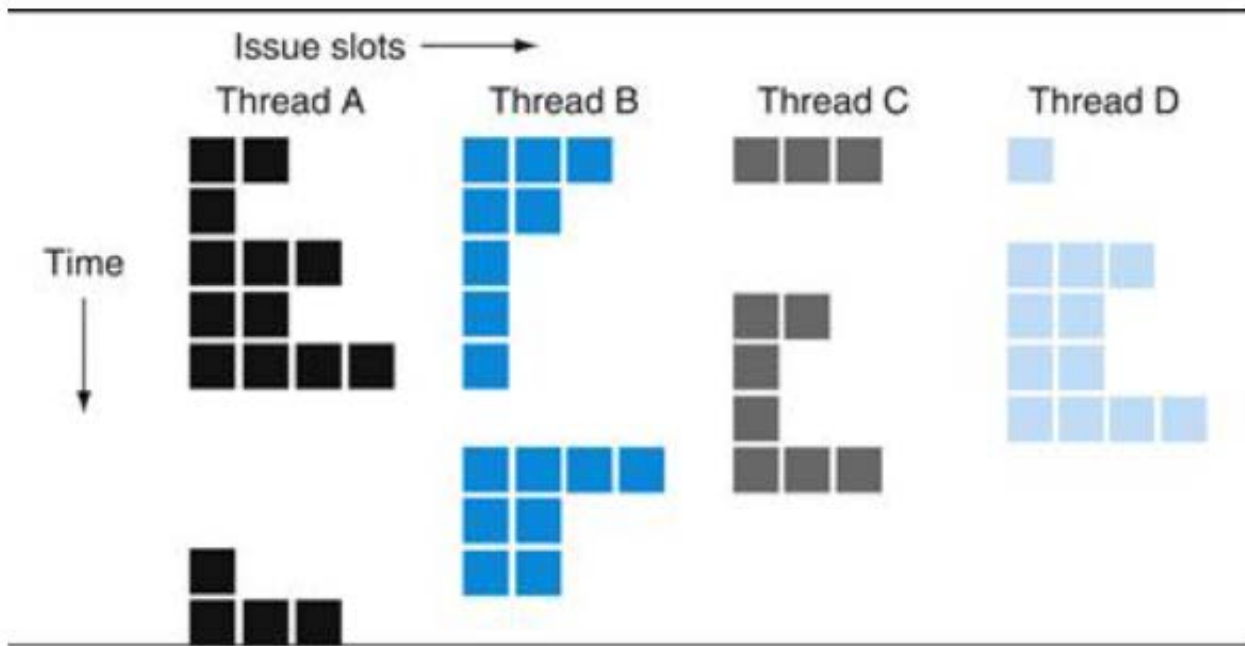
Pros: removes need for very fast switches

Cons: flush pipeline, short stalls not handled

Simultaneous multithreading

SMT

- Leverages multi-issue pipeline with dynamic instruction scheduling and ILP
- Exploits functional unit parallelism better than single threads
- Always running multiple instructions from multiple threads
 - No cost of context switching
 - Uses dynamic scheduling and register renaming through reservation stations
- Can use all functional units very efficiently



Hyperthreading

Multi-Core vs. Multi-Issue vs. HT

Programs:	N	1	N
Num. Pipelines:	N	1	1
Pipeline Width:	1	N	N

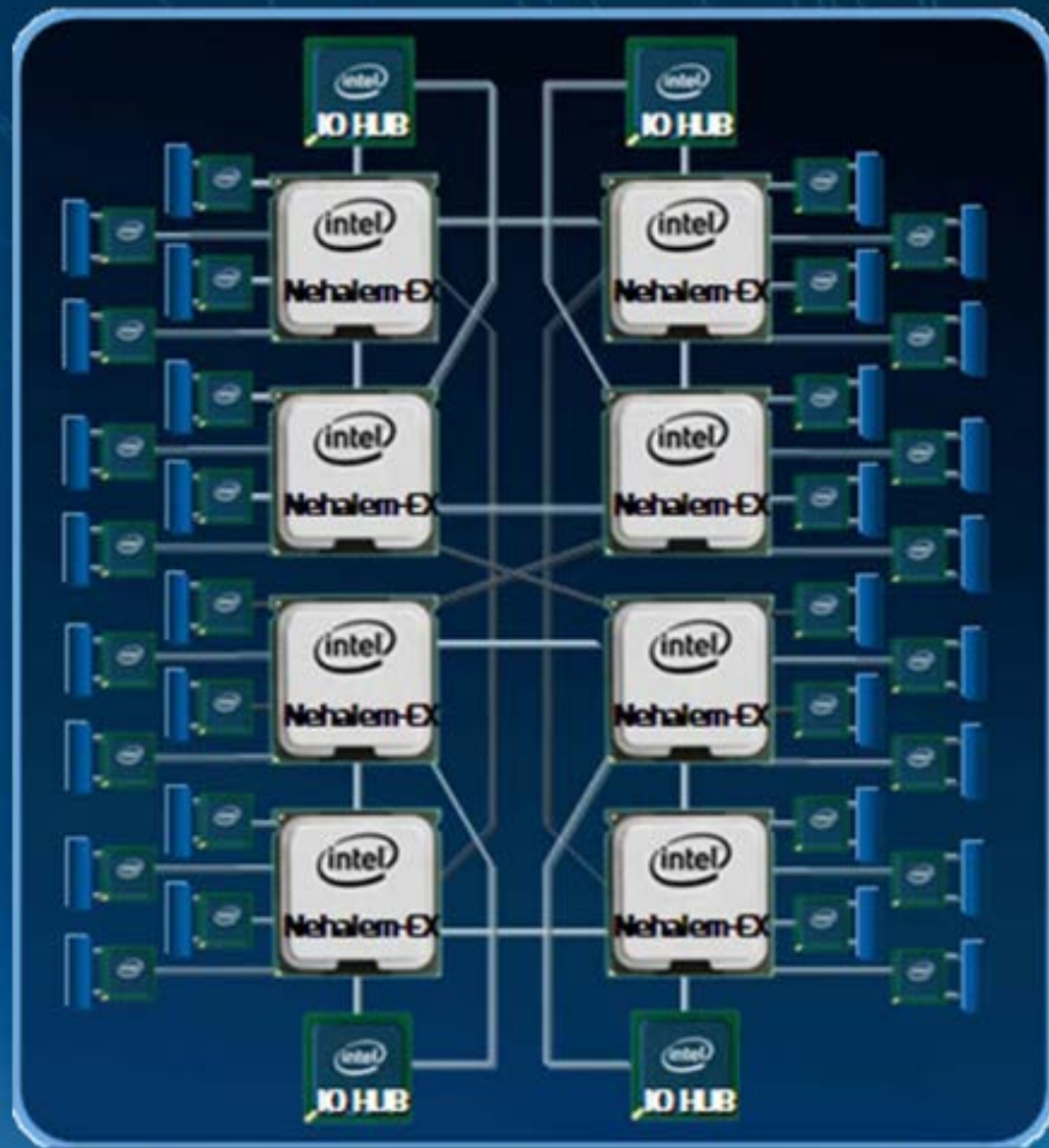
Hyperthreads

- HT = MultiIssue + 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 die (aka 8 sockets)
4 core per socket
2 HT per core

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

Memory
Intel Scalable Memory Buffer

Parallel Programming

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

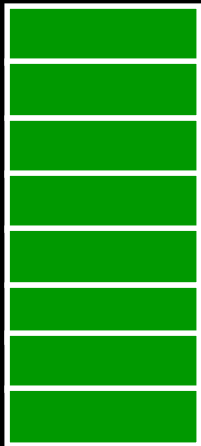
A: Software must be written as parallel program

Multicore difficulties

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

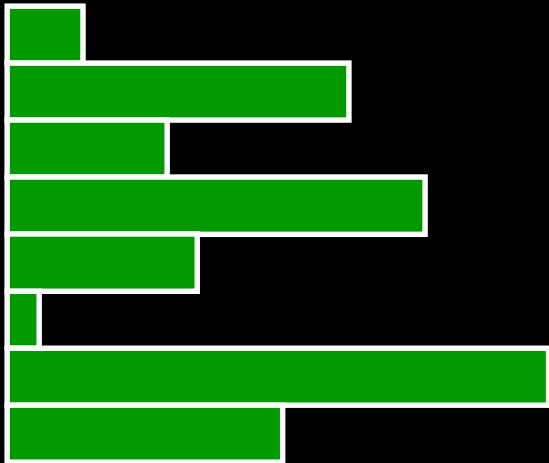
Work Partitioning

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

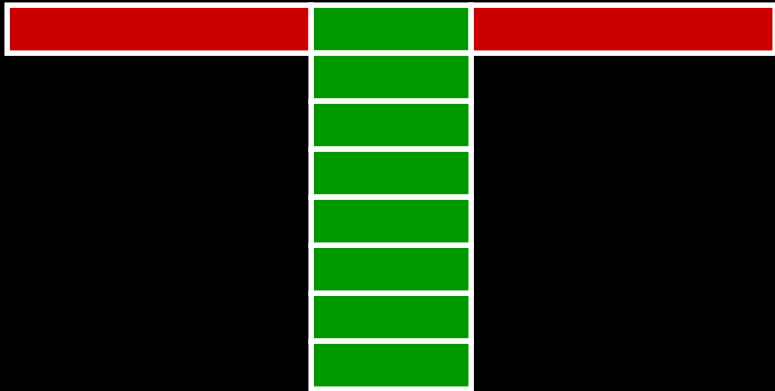
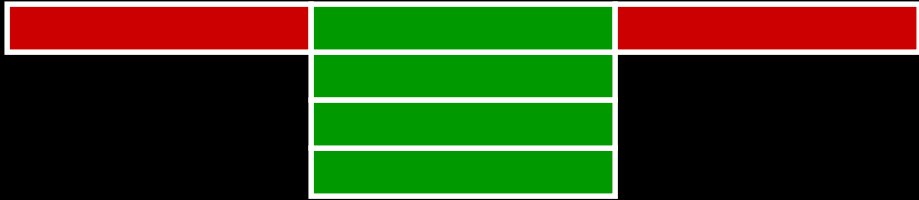
step 3: combine all results

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



Parallelism is a necessity

Necessity, not luxury

Power wall

Not easy to get performance out of

Many solutions

Pipelining

Multi-issue

Hyperthreading

Multicore

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 HW
- Balancing load over cores
- How do you write parallel programs?
 - ... without knowing exact underlying architecture?

SW
Your
career...