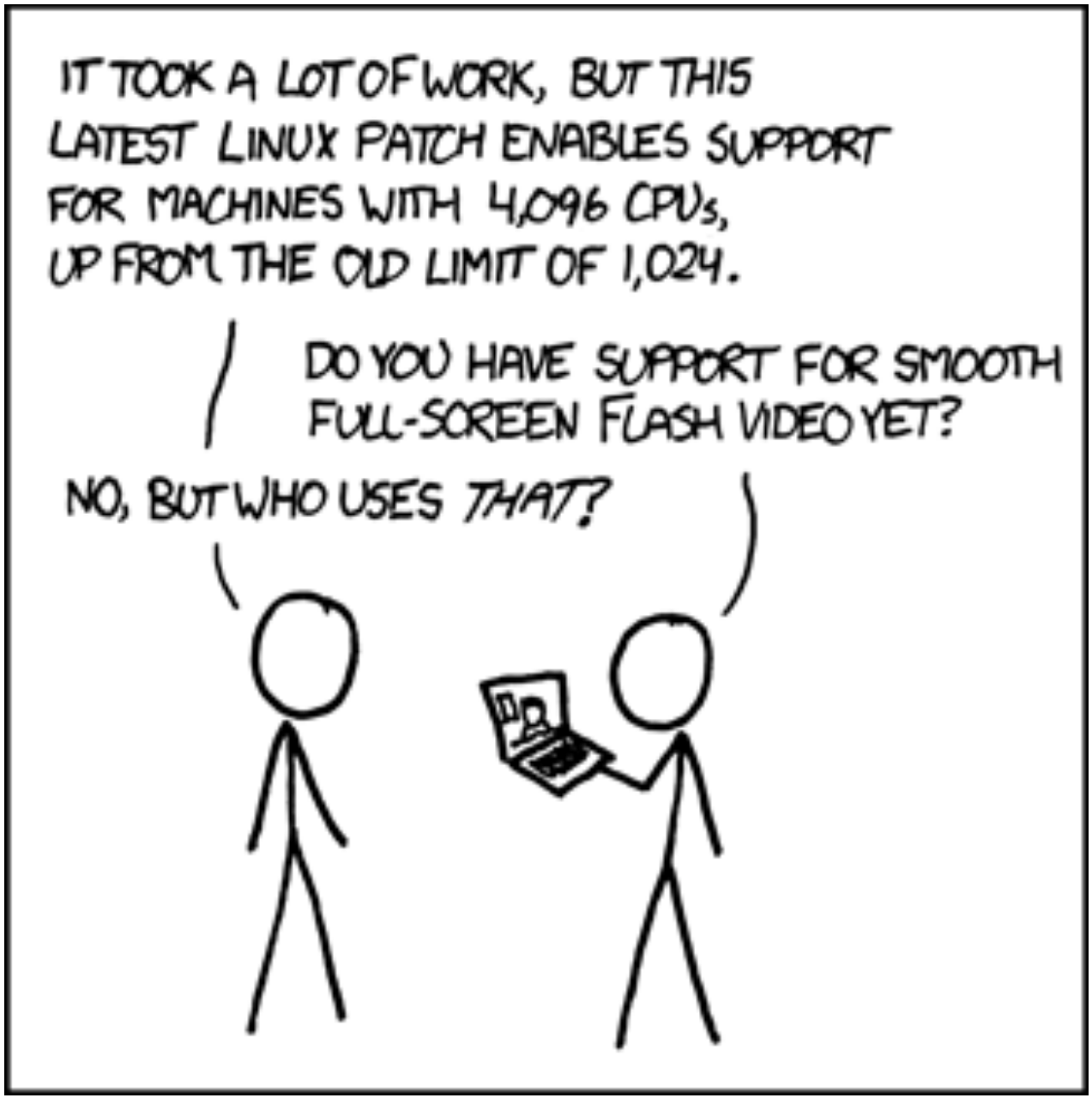

IT TOOK A LOT OF WORK, BUT THIS LATEST LINUX PATCH ENABLES SUPPORT FOR MACHINES WITH 4,096 CPUs, UP FROM THE OLD LIMIT OF 1,024.
DO YOU HAVE SUPPORT FOR SMOOTH FULL-SCREEN FLASH VIDEO YET?
NO, BUT WHO USES THAT?

Multicore & Parallel Processing

Guest Lecture: Kevin Walsh
CS 3410, Spring 2011
Computer Science
Cornell University

Execution

Execution time after improvement =

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

+ execution time unaffected

Amdahl's Law

Q: How to improve system performance?

→

→

Recall: Amdahl's Law

Solution: Parallelism

Pipelining: execute multiple instructions in parallel

Q: How to get more instruction level parallelism?

A: Deeper pipeline

Pipeline depth limited by…

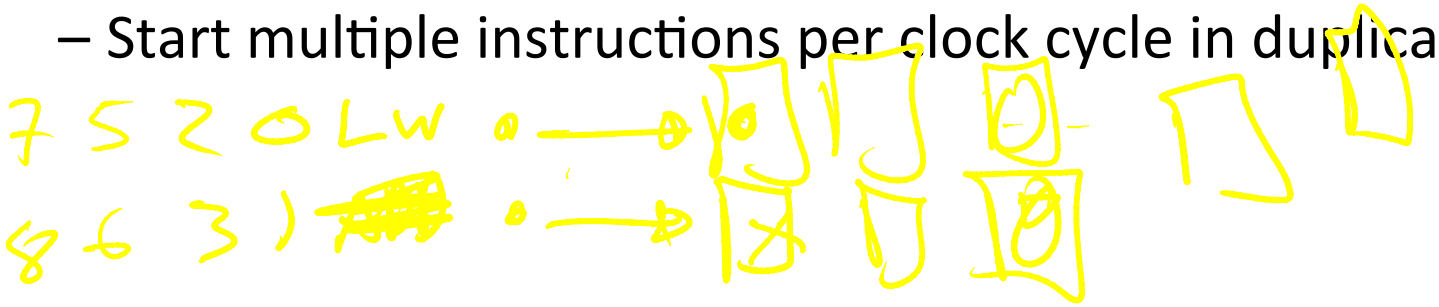
- max clock speed (less work per stage ⇒ shorter clock cycle)
- min unit of work
- dependencies, hazards / forwarding logic

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



Hazards. – Redefine – Delay slots etc

Branches – Always Even offset from Even address

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)

Compiler scheduling for dual-issue MIPS…

```
TOP:   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
```

8 cycles

	ALU/branch slot	Load/store slot	cycle
TOP:	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

5 cycles

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

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

Look Ahead

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

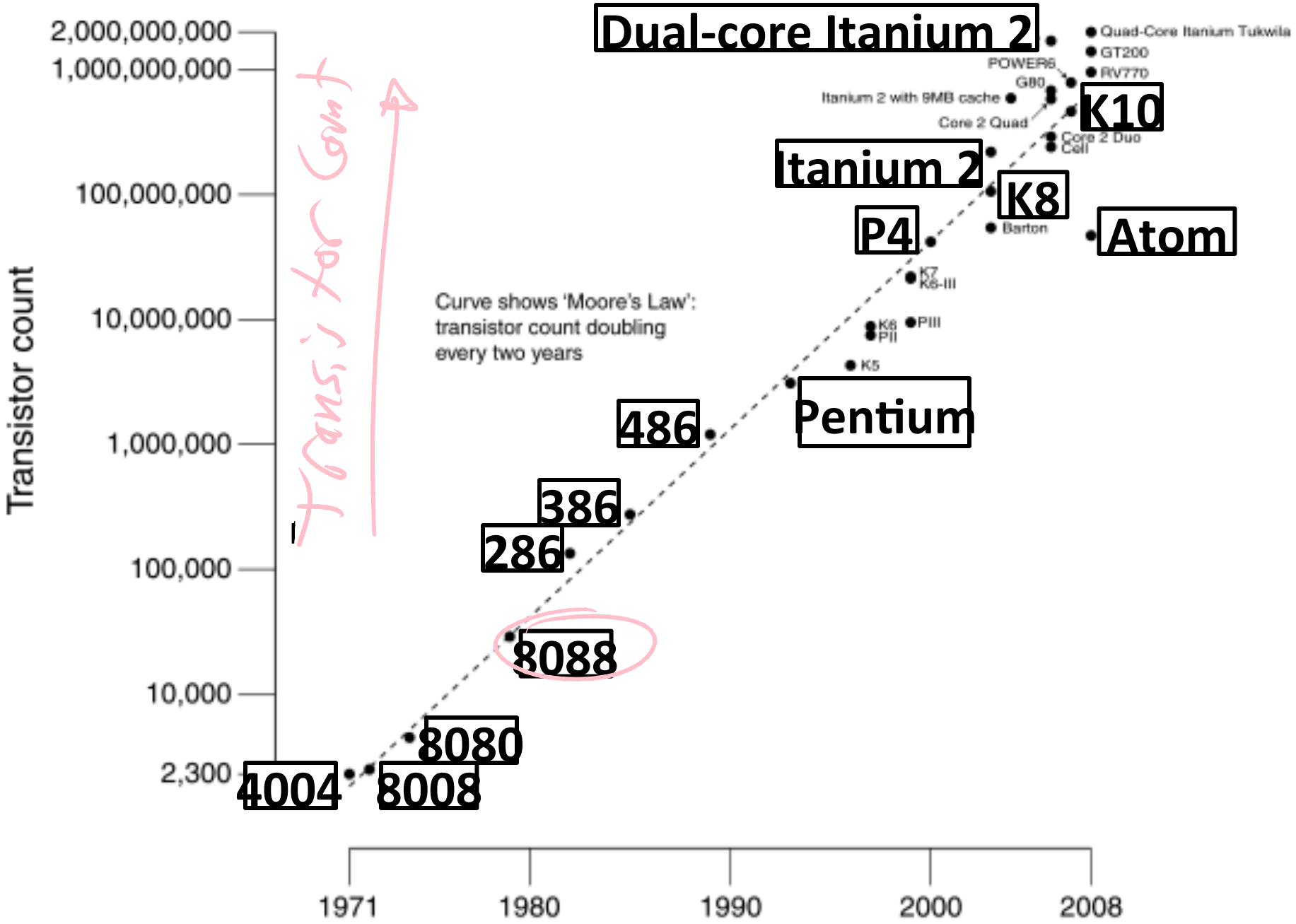
Q: Does multiple issue / ILP cost much?

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

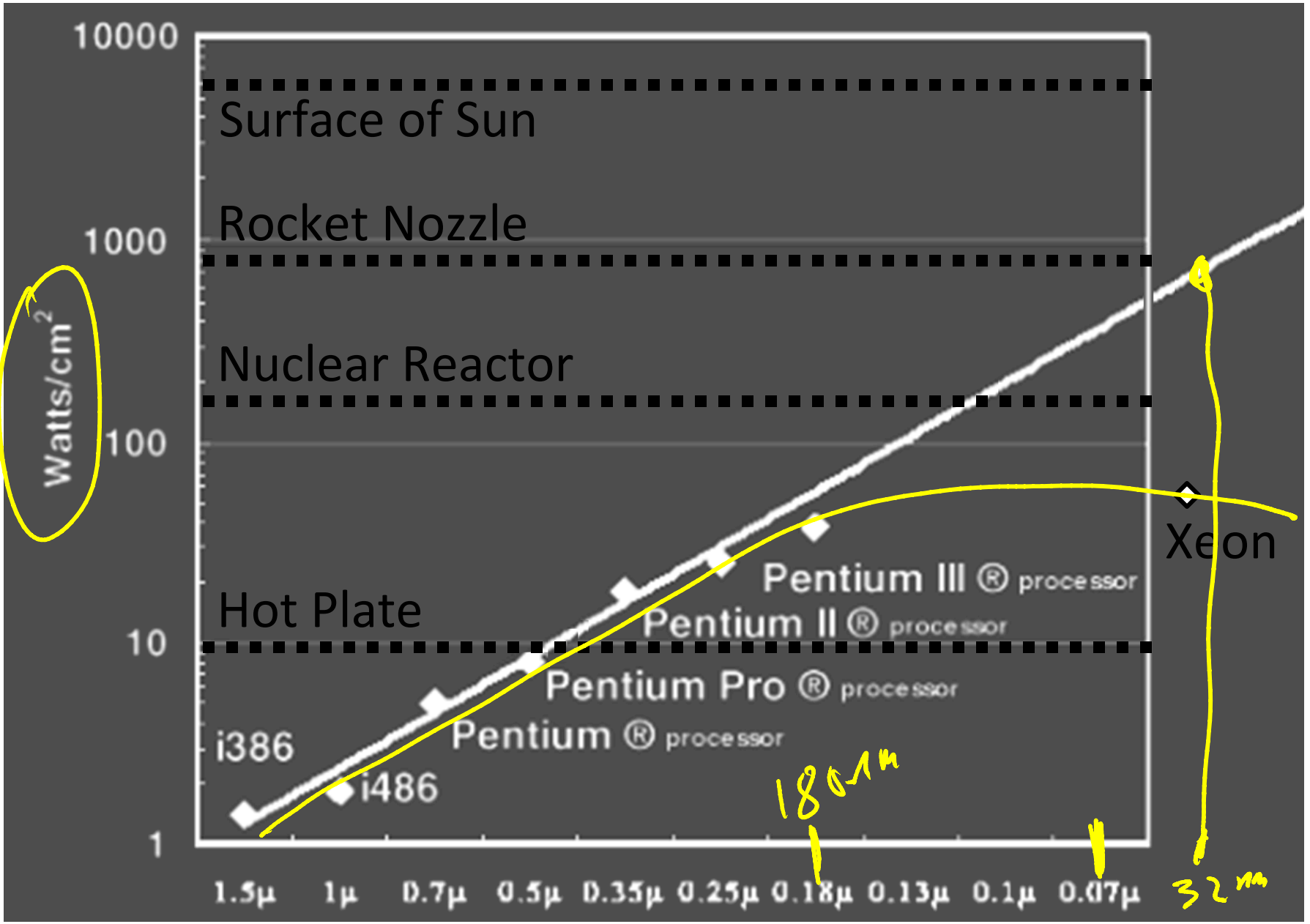
→ Multiple simpler cores may be better?


Dual-core Itanium 2
Quad-Core Itanium Tukwila
GT200
RV770
POWER6
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
PII
PIII
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

Moore's law

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

But: Power consumption growing too…


10000
1000
100
10
1
Watts/cm²
Surface of Sun
Rocket Nozzle
Nuclear Reactor
Hot Plate
Xeon
Pentium III ® processor
Pentium II ® processor
Pentium Pro ® processor
Pentium ® processor
i386
i486
180nm
32nm
1.5µ
1µ
0.7µ
0.5µ
0.35µ
0.25µ
0.18µ
0.13µ
0.1µ
0.07µ

Power = capacitance * voltage^2 * frequency

In practice: Power ~ voltage^3

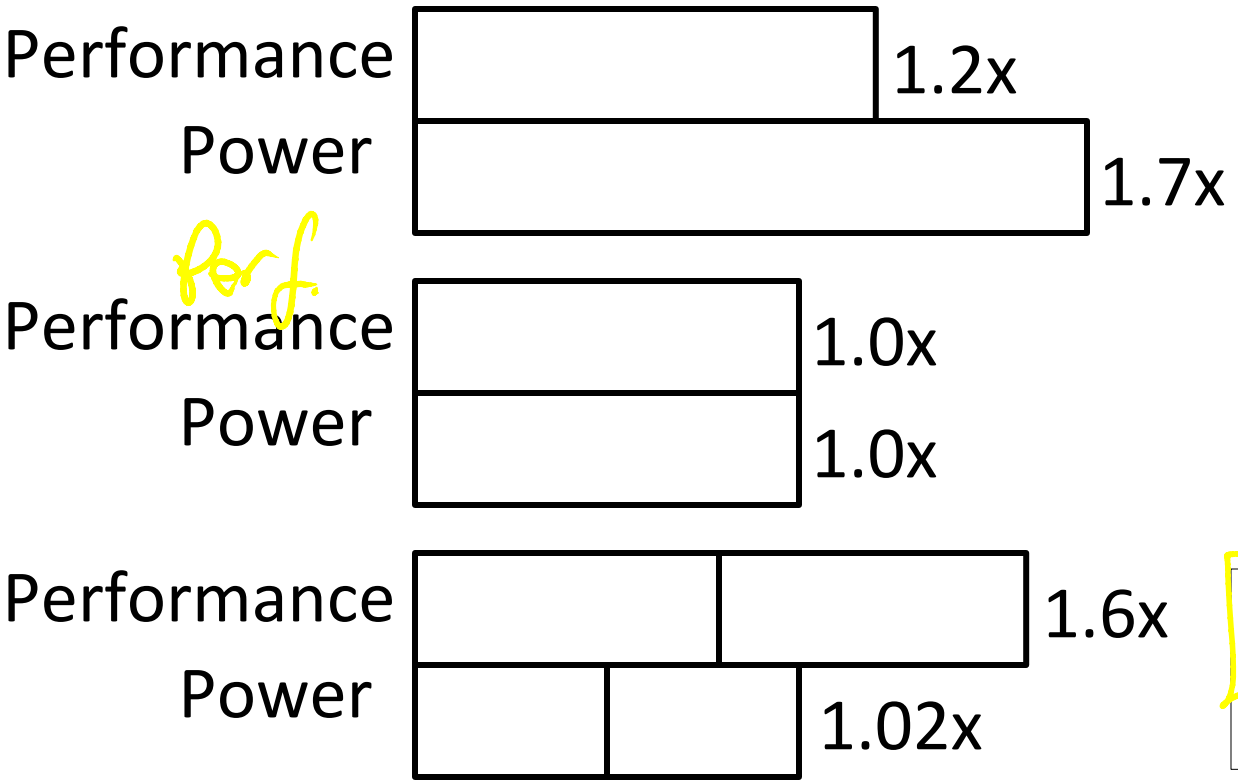
Reducing voltage helps (a lot)

... so does reducing clock speed

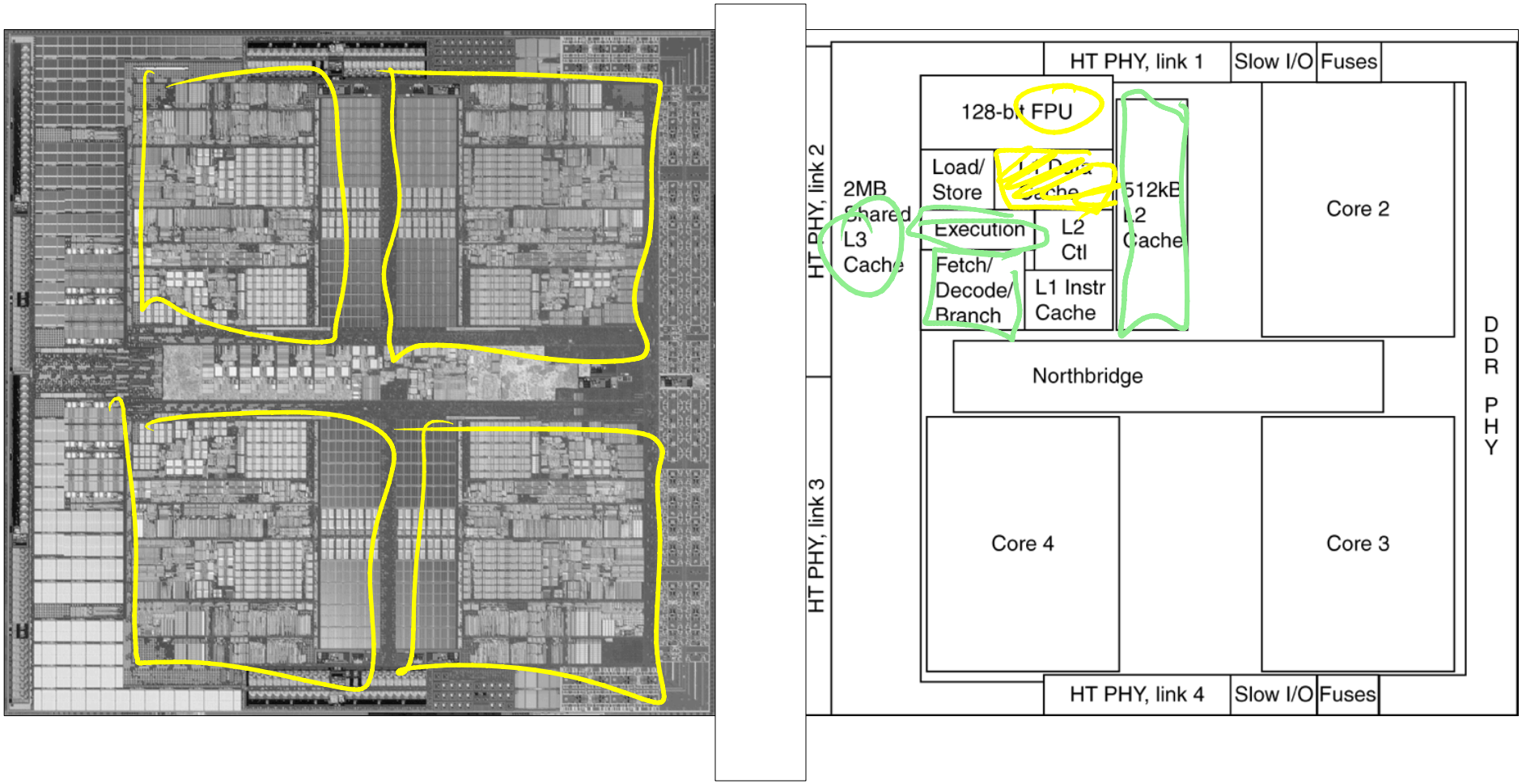
Better cooling helps

The power wall

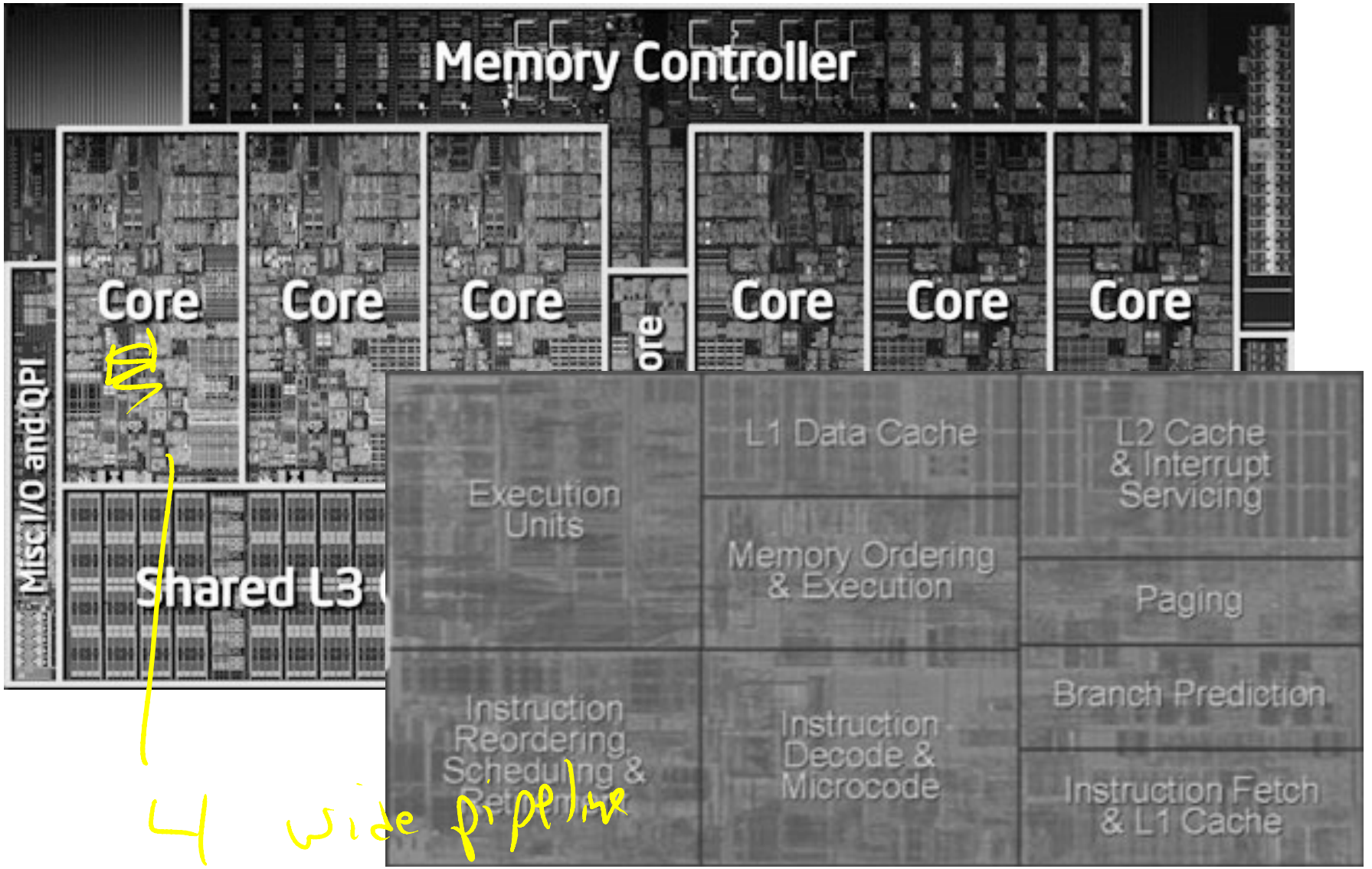
- We can't reduce voltage further
- We can't remove more heat


Performance
Power
1.2x
1.7x
Single-Core
Overclocked +20%
Perf.
CPU Freq.
Performance
Power
1.0x
1.0x
Single-Core
Performance
Power
1.6x
1.02x
Dual-Core
Underclocked -20%

AMD Barcelona Quad-Core: 4 processor cores



Intel Nehalem Hex-Core

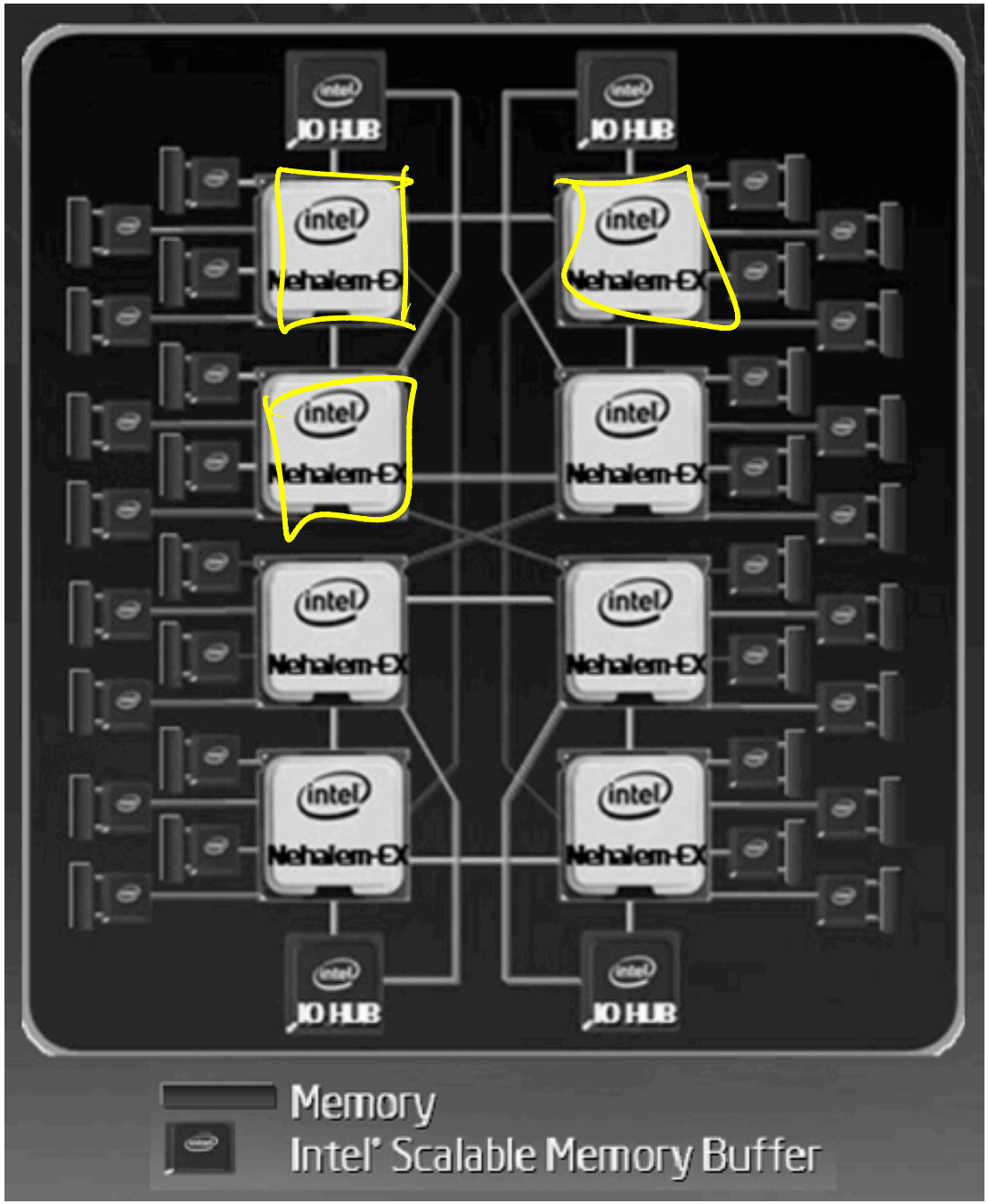


Multi-Core vs. Multi-Issue vs. HT

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

Hyperthreads (Intel)

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



8 dies
4 cores per die
2 HT

4 wide MIMD issue pipeline

16 pipeline stages

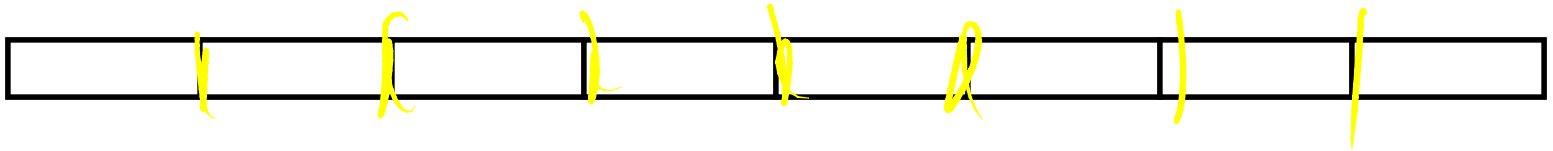
Q: So lets just all use multicore from now on!

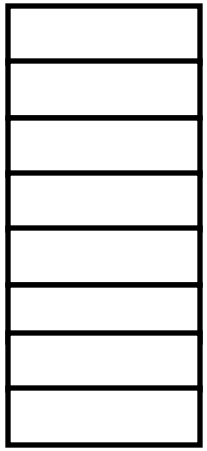
A: Software must be written as parallel program

Multicore difficulties

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

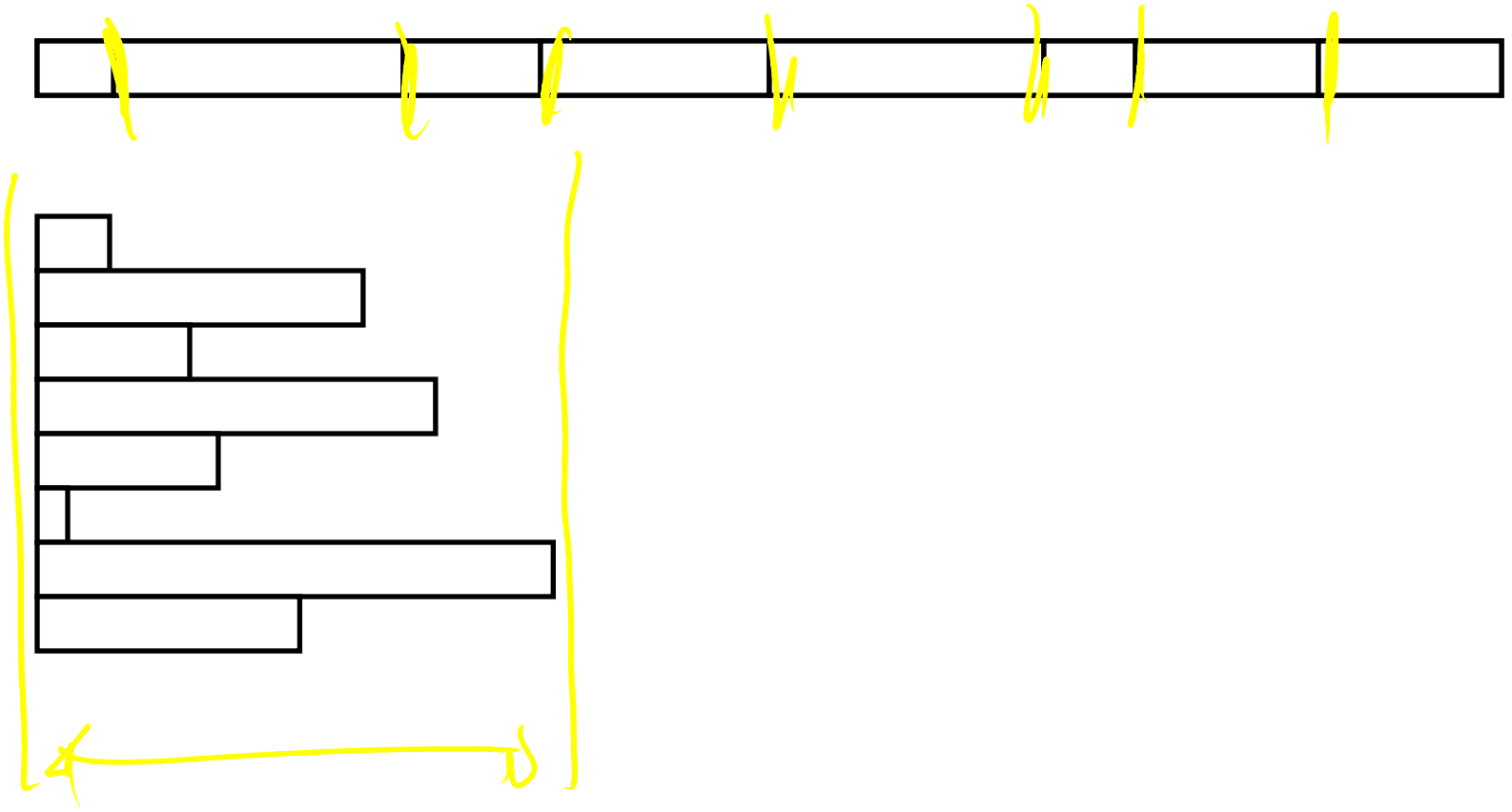
Partition work so all cores have something to do





Load Balancing

Need to partition so all cores are actually working



If tasks have a serial part and a parallel part…

Example:

step 1: divide input data into *n* pieces

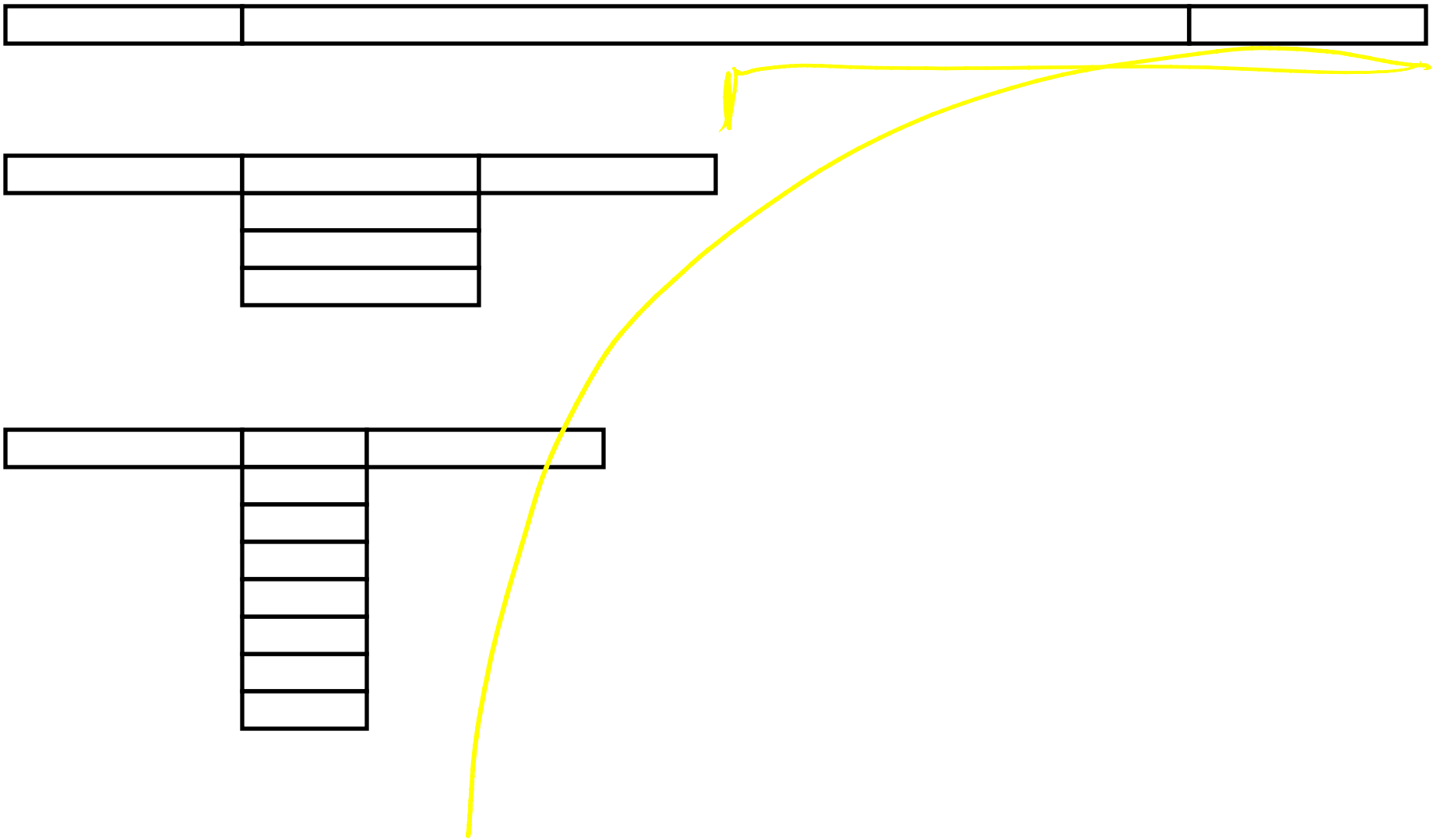
step 2: do work on each piece

step 3: combine all results

Recall: Amdahl's Law

As number of cores increases …

- time to execute parallel part?
- time to execute serial part?



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?