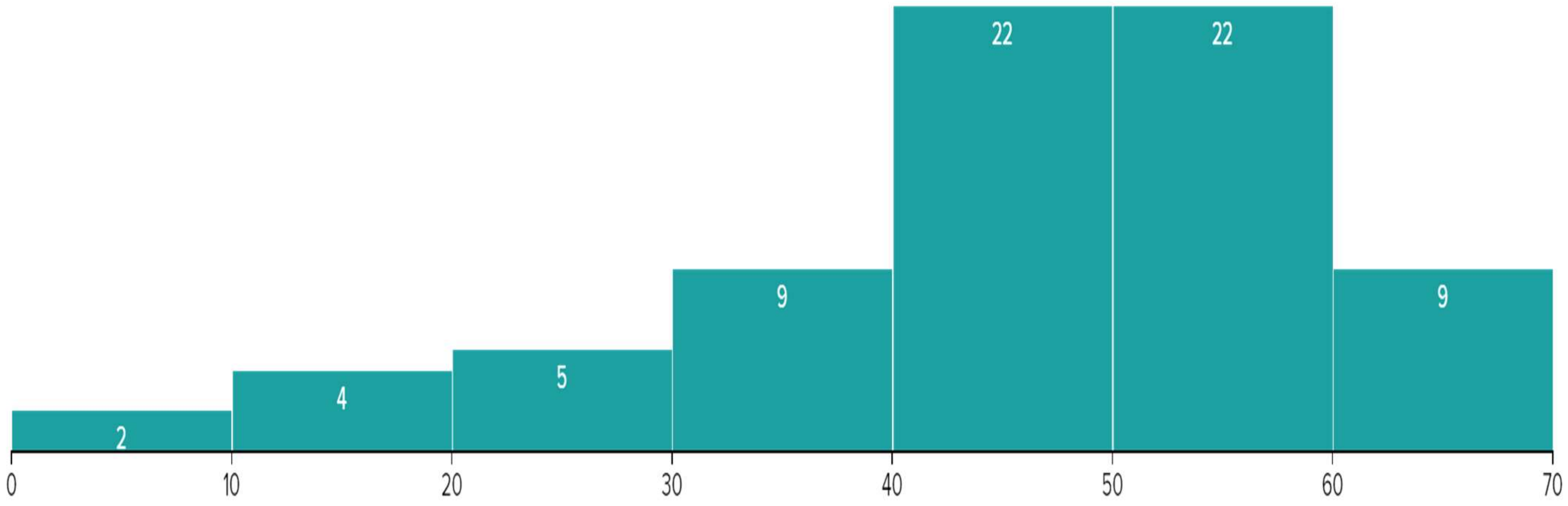
18-447 Lecture 20: ILP to Multicores

James C. Hoe

Department of ECE

Carnegie Mellon University

Midterm 2 Class Distribution



MINIMUM	MEDIAN	MAXIMUM	MEAN	STD DEV
6.0	47.5	70.0	45.26	14.41

Midterm 2 Summary Statistics

	1:TLB	2:except	3:energy	4:poll	5:draw	6:trace	7:table	total
possible	6	8	12	8	19	12	14	70
average	4.8	3.3	5.8	4.6	8.2	9.3	9.3	45.3
stdev	1.8	2.8	4.1	3.0	2.1	3.3	5.6	14.4
max	6.0	8.0	12.0	8.0	10.0	12.0	14.0	70.0
median	6.0	4.0	7.0	5.5	8.0	11.0	14.0	47.5
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.0

Housekeeping

- Your goal today
  - transition from sequential to parallel
  - enjoy (only first part, before OOO, on 447 exam)
- Notices
  - get going on Lab 4, now 3 weeks left
  - Handout #17: HW6, due Monday 5/2 noon
- Readings (advanced optional)
  - MIPS R10K Superscalar Microprocessor, Yeager
  - Synthesis Lectures: *Processor Microarchitecture: An Implementation Perspective, 2010*
  - Superscalar Club!!

Parallelism Defined

- T_1 (work measured in time):
  - time to do work with 1 PE
- T_∞ (critical path):
  - time to do work with infinite PEs
  - T_∞ bounded by dataflow dependence
- Average parallelism:

$$P_{avg} = T_1 / T_\infty$$

let's call **p** *concurrency*

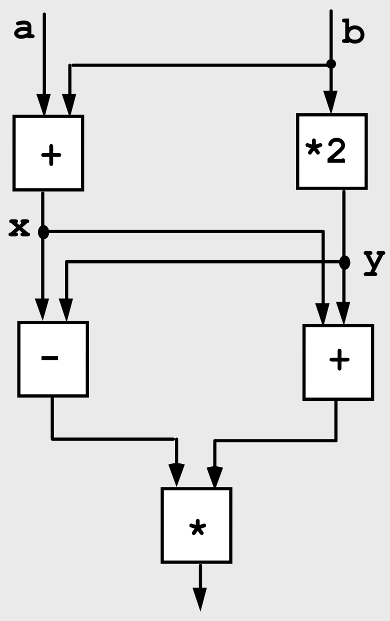
- For a system with **p** PEs

$$T_p \geq \max\{ T_1/p, T_\infty \}$$

- When $P_{avg} \gg p$

$$T_p \approx T_1/p$$, aka "linear speedup"

x = a + b;
y = b * 2
z =(x-y) * (x+y)

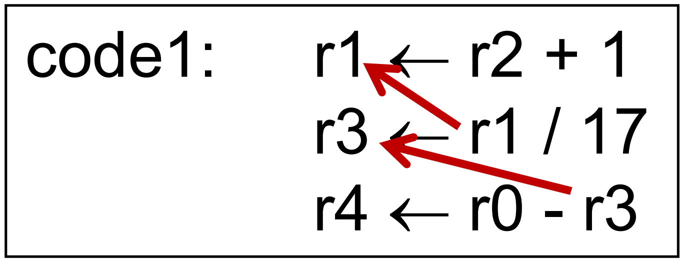


ILP: Instruction-Level Parallelism

- Average **ILP** = T_1 / T_∞
  = no. instruction / no. cyc required

code1: **ILP** = 1
i.e., must execute serially

code2: **ILP** = 3
i.e., can execute at the same time



```
code1:   r1 ← r2 + 1
         r3 ← r1 / 17
         r4 ← r0 - r3
```

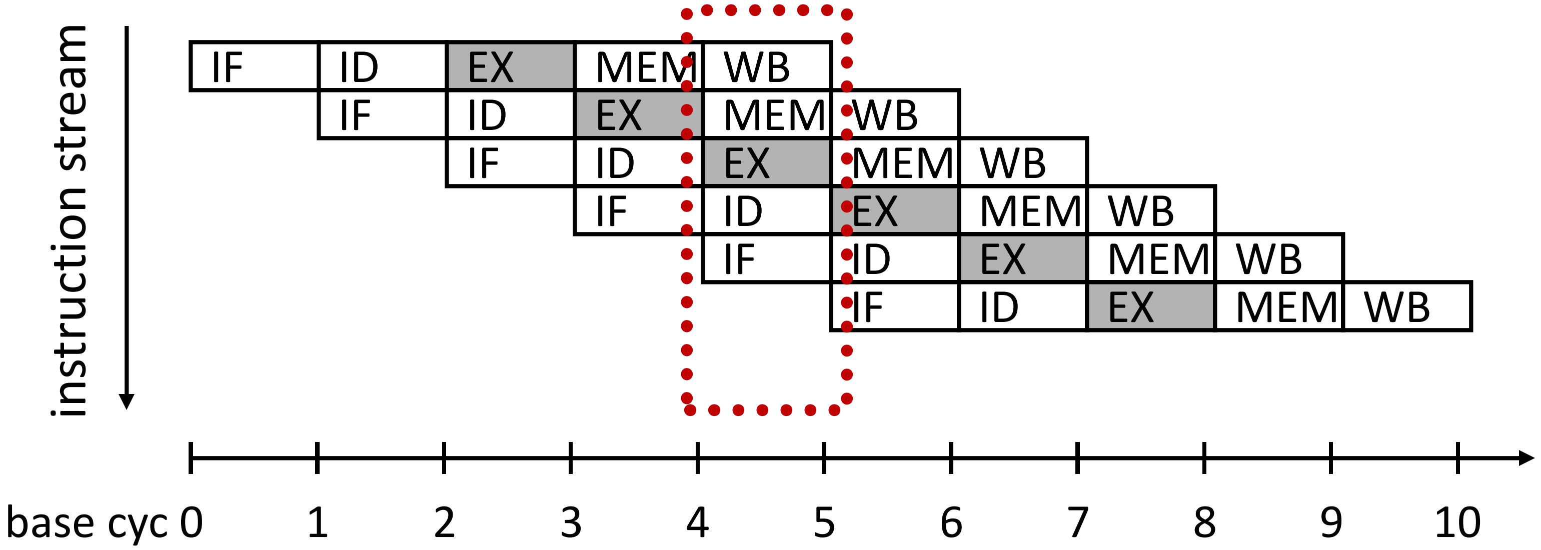
```
code2:   r1 ← r2 + 1
         r3 ← r9 / 17
         r4 ← r0 - r10
```

Superscalar Speculative Out-of-Order Execution

Exploiting ILP for Performance

Scalar in-order pipeline with forwarding

- operation latency (**OL**)= **1** base cycle
- peak **IPC** = **1** *// no concurrency*
- require **ILP** ≥ **1** to avoid stall

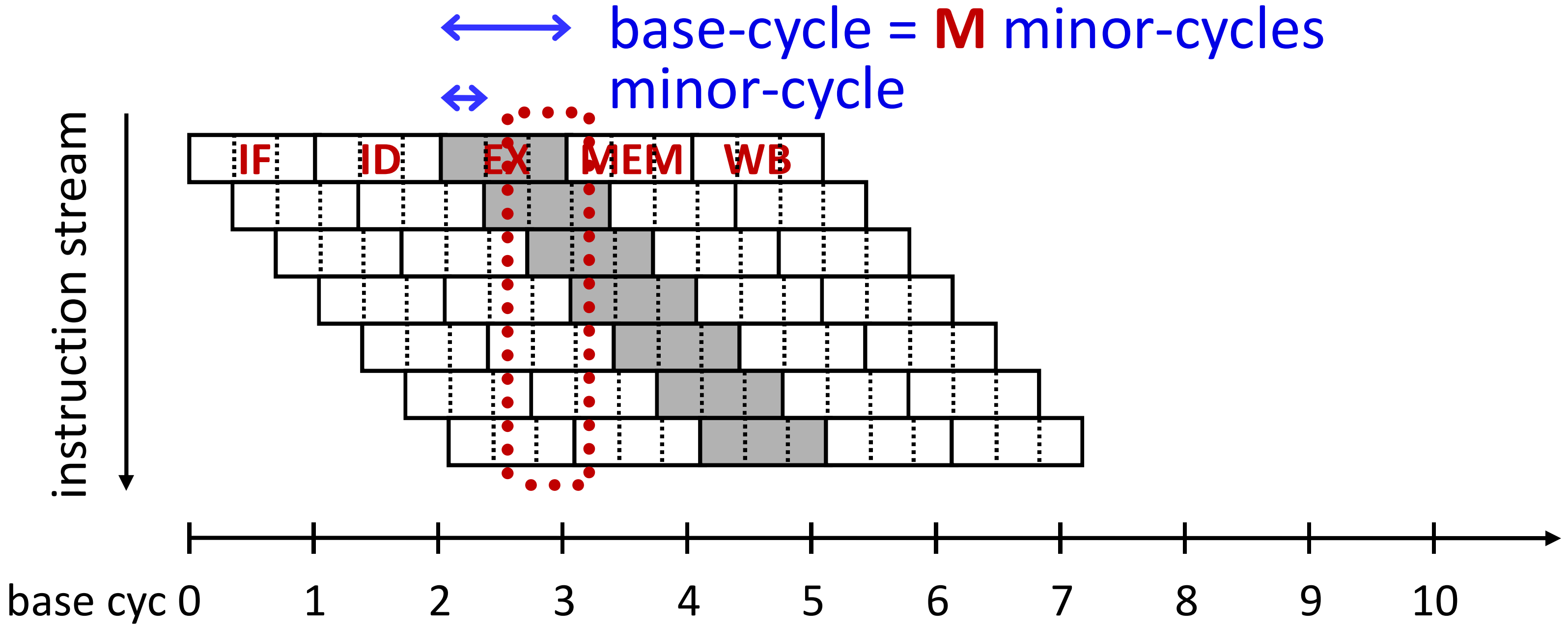


Superpipelined Execution

OL = M minor-cycle; same as 1 base cycle

peak IPC = 1 per minor-cycle *// has concurrency though*

required ILP ≥ M



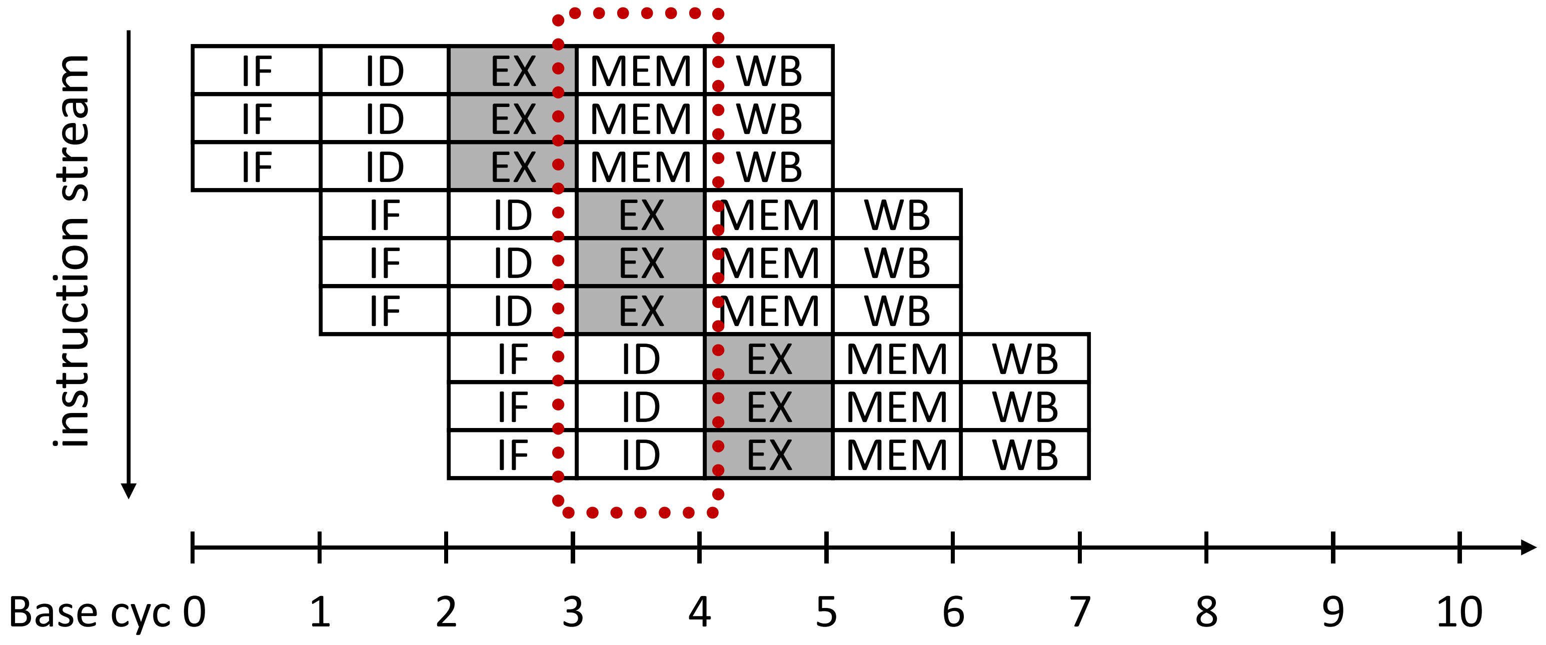
Achieving full performance requires always finding M "independent" instructions in a row

Superscalar (Inorder) Execution

OL = **1** base cycle

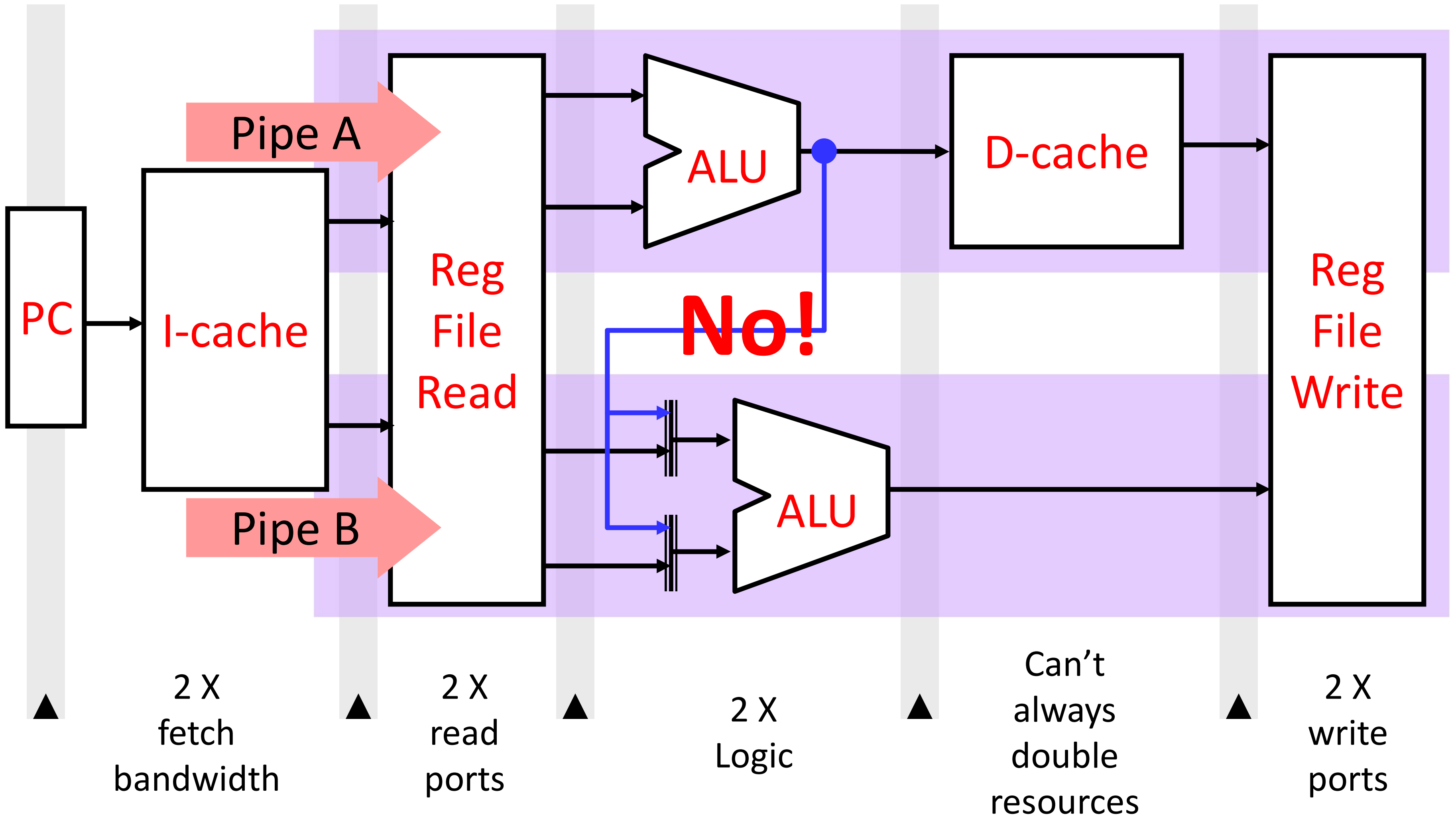
peak **IPC** = **N**

required **ILP** ≥ **N**



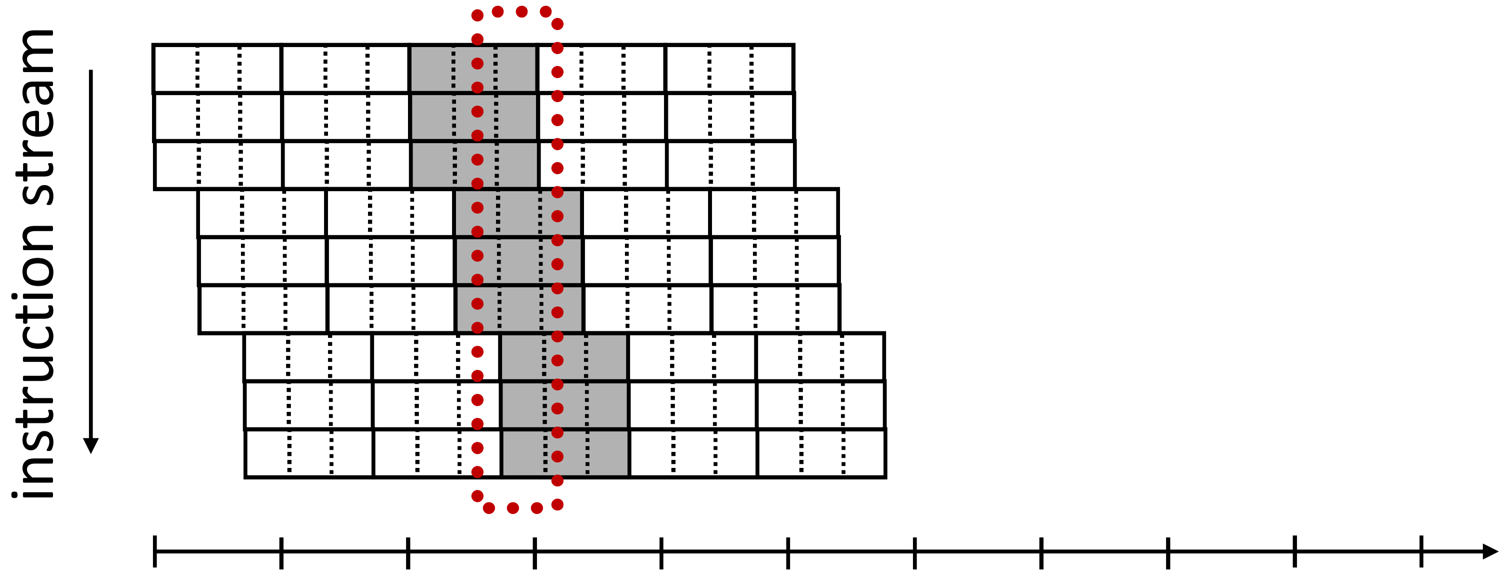
Achieving full performance requires finding **N** "independent" instructions on every cycle

Lab 4: 2-way, In-order Superscalar



Limitations of Inorder Pipeline

- Achieved **IPC** of inorder pipelines degrades rapidly as **N**x**M** approaches **ILP**
- Despite high concurrency potential, pipeline never full due to frequent dependency stalls!!



Out-of-Order Execution

- **ILP** is scope dependent

$$\left.\begin{array}{l} \mathbf{ILP}=1 \left\{\begin{array}{l} r1 \leftarrow r2 + 1 \\ r3 \leftarrow r1 / 17 \\ r4 \leftarrow r0 - r3 \end{array}\right. \\ r11 \leftarrow r12 + 1 \\ r13 \leftarrow r19 / 17 \\ r14 \leftarrow r0 - r20 \end{array}\right\} \mathbf{ILP}=2$$

Accessing **ILP**=2 requires not only (1) larger scheduling window but also (2) out-of-order execution

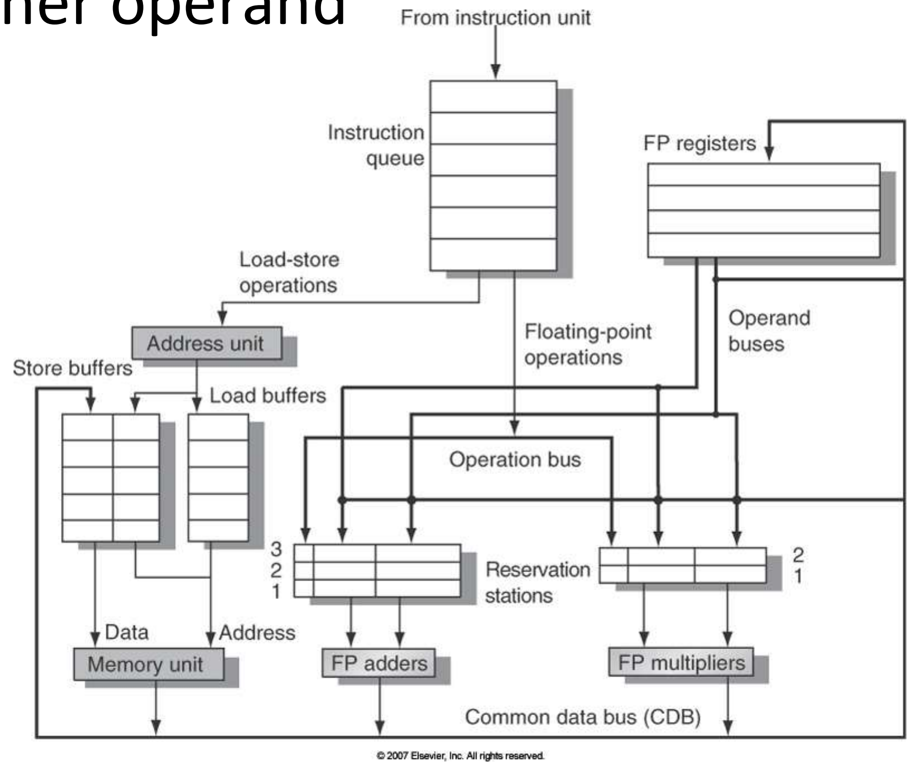
Superscalar Speculative Out-of-Order Execution

Instruction Micro-Dataflow

- Maintain a buffer of many pending instructions, a.k.a. reservation stations (**RS**s)
  - wait for functional unit to be free
  - wait for required input operands to be available
- Decouple execution order from who is first in line (program order)
  - select inst's in **RS** whose operands are available
  - give preference to older instructions (heuristical)
- A completing instruction (producer) signals dependent instructions (consumer) of operand availability *(producer-push resolution)*

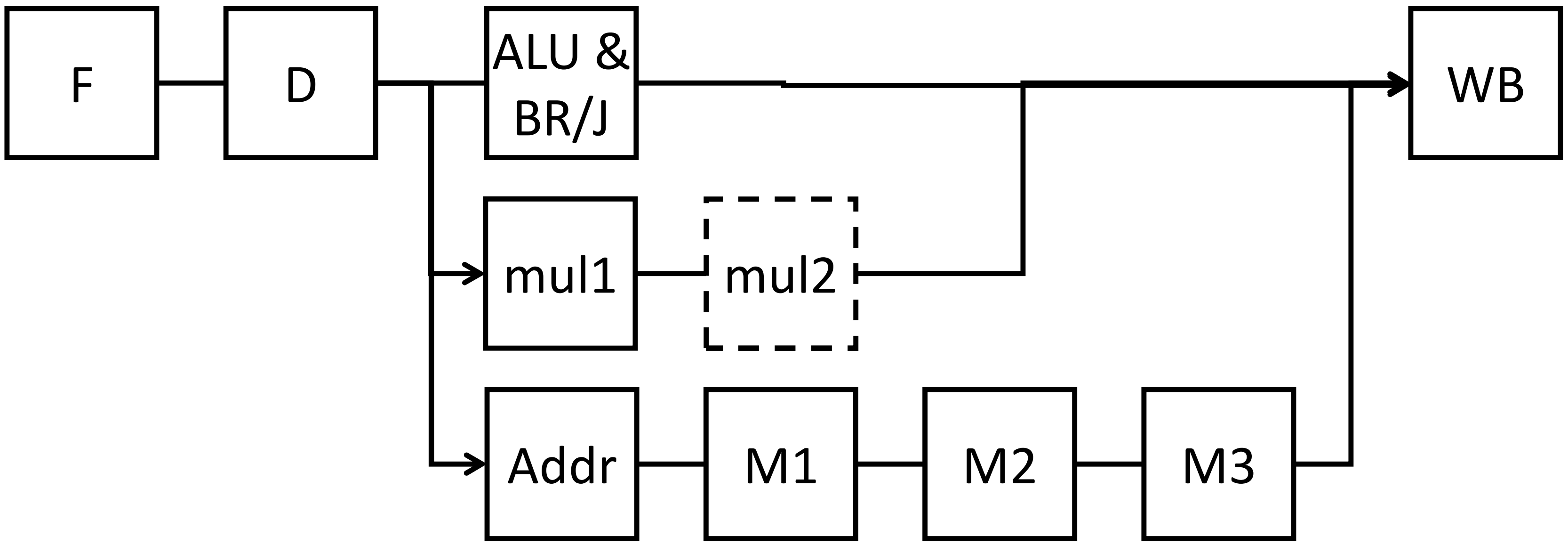
Tomasulo's Algorithm [IBM 360/91, 1967]

- Dispatch an instruction to a **RS** slot after decode
  - decode received from RF either operand value or placeholder **RS-tag**
  - mark RF dest with **RS-tag** of current inst's **RS** slot
- Inst in **RS** can issue when all operand values ready
- Completing instruction, in addition to updating RF dest, broadcast its **RS-tag** and value to all **RS** slots
- **RS** slot holding matching **RS-tag** placeholder pickup value



WAW and WAR

- No WAW and WAR before because
  - single write stage
  - write stage at the end *(later than any read stage)*
  - in-order progression in pipeline



Removing False Dependencies

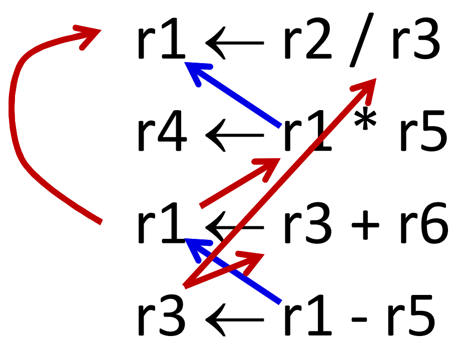
- With out-of-order execution comes WAW and WAR hazards
- Anti and output dependencies are false dependencies on register names rather than data

$$\begin{tikzcd}
r_3 & \leftarrow & r_1 & \text{op} & r_2 \\
r_5 & \leftarrow & r_3 & \text{op} & r_4 \\
r_3 \arrow[uu, bend left] \arrow[urr] & \leftarrow & r_6 & \text{op} & r_7
\end{tikzcd}$$

- With infinite number of registers, anti and output dependencies avoidable by using a new register for each new value

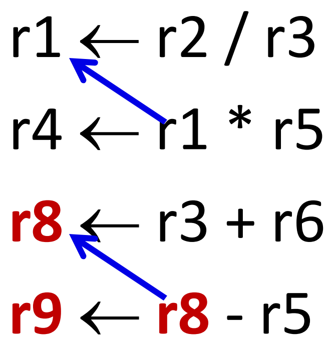
Register Renaming: Example

Original



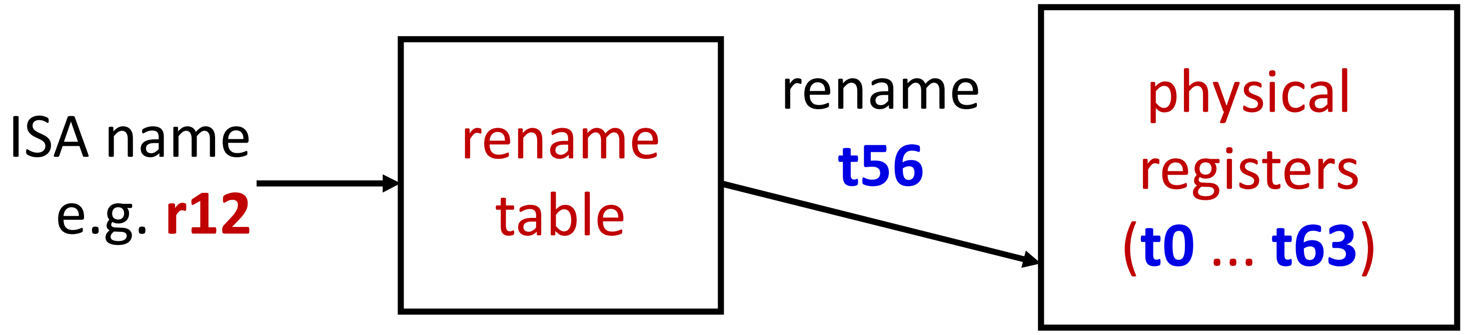
```
r1 ← r2 / r3
r4 ← r1 * r5
r1 ← r3 + r6
r3 ← r1 - r5
```

Renamed



```
r1 ← r2 / r3
r4 ← r1 * r5
r8 ← r3 + r6
r9 ← r8 - r5
```

On-the-fly HW Register Renaming



- Maintain mapping from ISA reg. names to physical registers
- When decoding an instruction that updates 'r_x':
  - allocate unused physical register t_y to hold inst result
  - set new mapping from 'r_x' to t_y
  - younger instructions using 'r_x' as input finds t_y
- De-allocate a physical register for reuse when it is never needed again?

^^^^^when is this exactly?

$r1 \leftarrow r2 / r3$

$r4 \leftarrow r1 * r5$

$r1 \leftarrow r3 + r6$

Superscalar Speculative Out-of-Order Execution

Control Speculation

- For want of a large window of instructions
  - if 14% of avg. instruction mix is control flow, what is average distance between control flow?
  - instruction fetch must make multiple levels of branch predictions (condition and target) to fetch far ahead of execution and commit
- Modern CPUs can have over 100 instructions in out-of-order execution scope

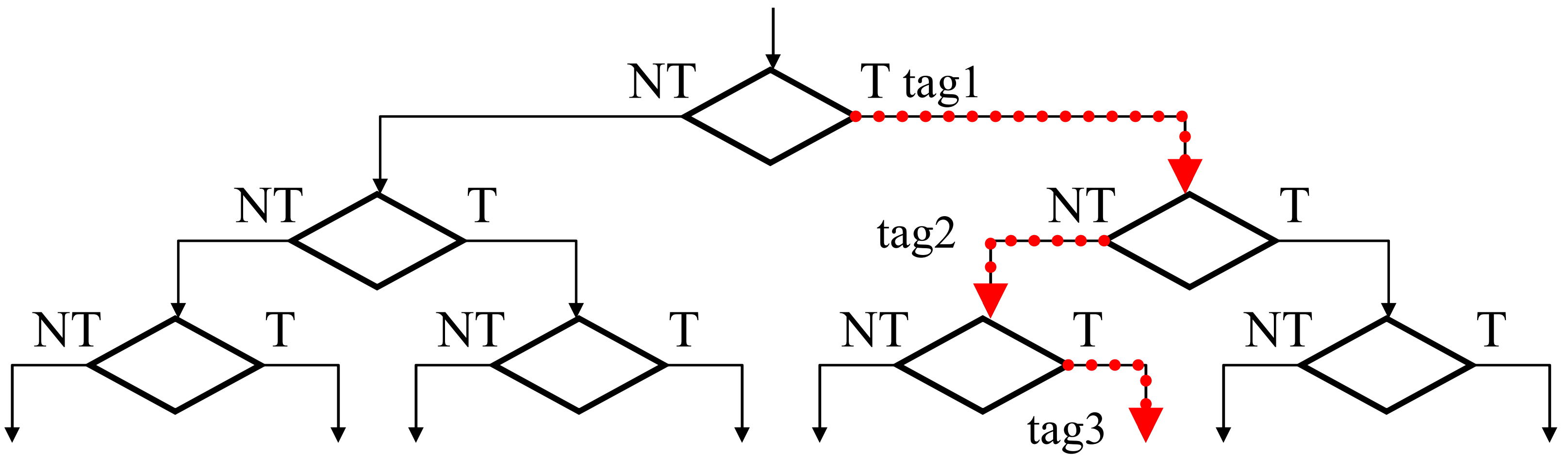
- **Question:**
  - **how much more ILP is uncovered with look ahead**
  - **how much useful work is done during look ahead**

Ans: not much and not much

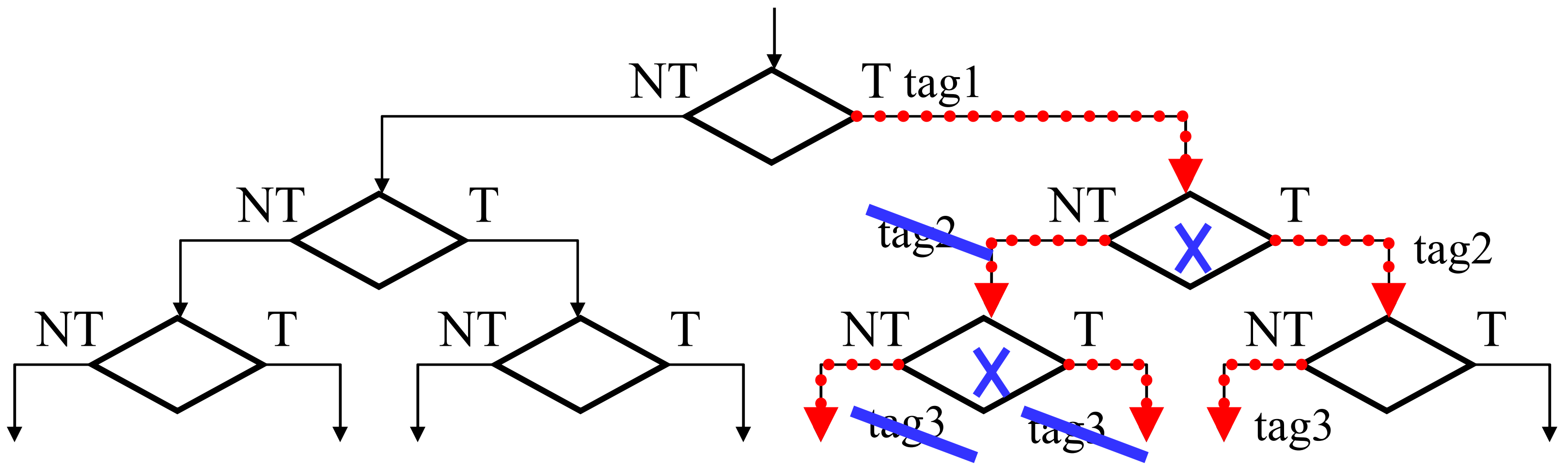
Speculative Out-of-order Execution

- A mispredicted branch after resolution must be rewound and restarted **ASAP**!
- Much trickier than 5-stage pipeline . . .
  - can rewind to an intermediate speculative state
  - a rewound branch could still be speculative and itself be discarded by another rewind!
  - rewind must reestablish both architectural state (register value) and microarchitecture state (e.g., rename table)
  - rewind/restart must be fast (not infrequent)
- Also need to rewind on exceptions . . . .but easier

Nested Control Flow Speculation

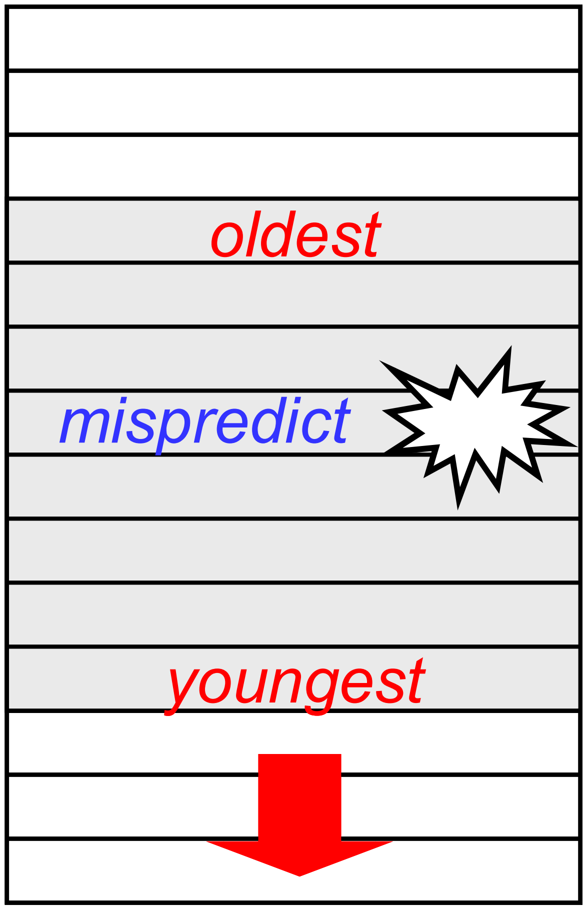


Mis-speculation Recovery can be Speculative



Instruction Reorder Buffer (ROB)

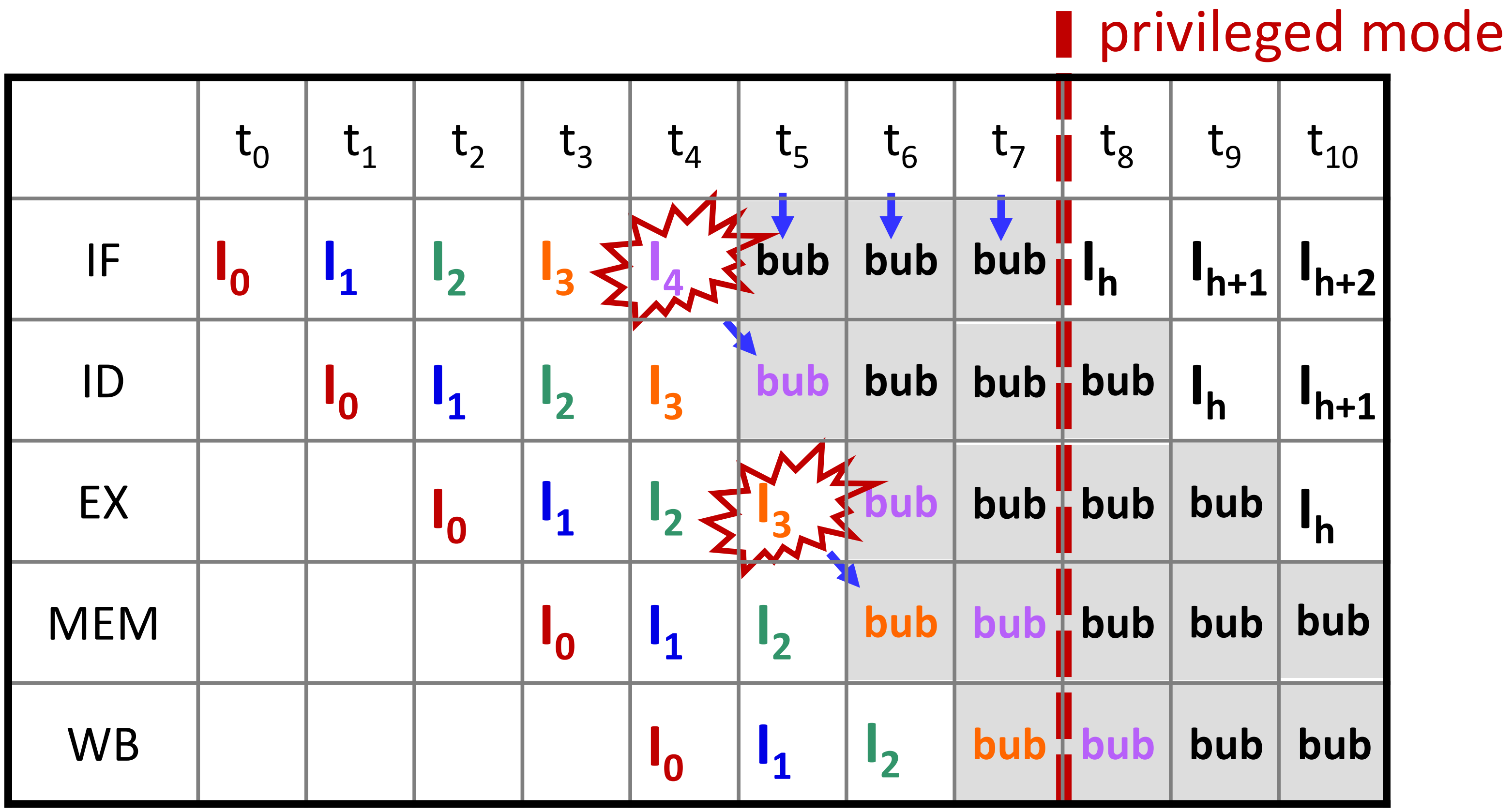
- Program-order bookkeeping (circular buffer)
  - instructions enter and leave in program order
  - tracks 10s to 100s of in-flight instructions in different stages of execution
- Dynamic juggling of state and dependency
  - oldest finished instruction "commit" architectural state updates on exit
  - all ROB entries considered "speculative" due to potential for exceptions and mispredictions



In-order vs Speculative State

- In-order state:
  - cumulative architectural effects of all instructions committed in-order so far
  - can never be undone!!
- Speculative state, as viewed by a given inst in **ROB**
  - in-order state + effects of older inst's in **ROB**
  - effects of some older inst's may be pending
- Speculative state effects must be reversible
  - remember both in-order and speculative values for an RF register (may have multiple speculative values)
  - store inst updates memory only at commit time
- Discard younger speculative state to rewind execution to oldest remaining inst in **ROB**

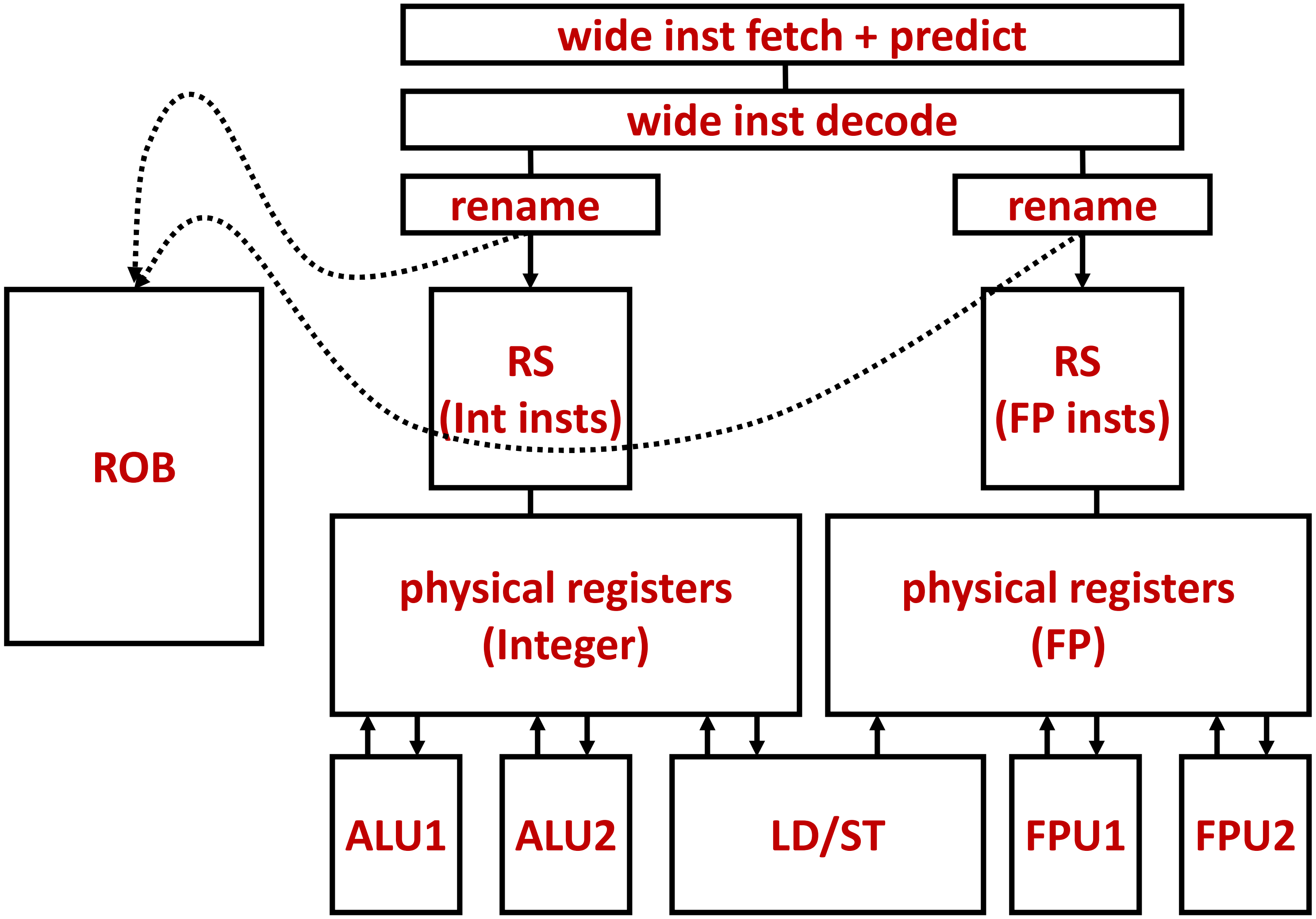
You have seen this in your pipeline



	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
IF	I_0	I_1	I_2	I_3	I_4	bub	bub	bub	I_h	I_{h+1}	I_{h+2}
ID		I_0	I_1	I_2	I_3	bub	bub	bub	bub	I_h	I_{h+1}
EX			I_0	I_1	I_2	I_3	bub	bub	bub	bub	I_h
MEM				I_0	I_1	I_2	bub	bub	bub	bub	bub
WB					I_0	I_1	I_2	bub	bub	bub	bub

- Flush the pipeline at anytime to start from after last instruction committed
- Decode in ID needs to consider pending "speculative" state updates in EX/MEM/WB, on top of committed RF states

Superscalar Speculative OOO All Together



Read [Yeager 1996, IEEE Micro] if you are interested

Truth about Superscalar Speculative OOO

- If memory speed kept up with core speed, we would still be building in-order pipelines
- But, by 2005 we were seeing

  e.g., Intel P4 at 4+GHz

- 16KB L1 D-cache
  - t_1 = 4 cyc int (9 cycle fp)
- 1024KB L2 D-cache
  - t_2 = 18 cyc int (18 cyc fp)
- Main memory
  - t_3 = ~ 50ns or 180 cyc

- Speculative OOO has really been about
  - finding independent work to do after cache hit&miss
  - getting to future cache misses as early as possible
  - overlapping multiple cache misses for BW

At the 2005 Peak of Superscalar OOO

	Alpha 21364	AMD Opteron	Intel Xeon	IBM Power5	MIPS R14000	Intel Itanium2
clock (GHz)	1.30	2.4	**3.6**	1.9	0.6	1.6
issue rate	4	3 (x86)	3 (rop)	**8**	4	8
pipeline int/fp	7/9	9/11	**22/24**	12/17	6	8
inst in flight	80	72(rop)	126 rop	**200**	48	inorder
rename reg	48+41	36+36	128	48/40	32/32	**328**
transistor (10^6)	135	106	125	276	7.2	**592**
power (W)	**155**	86	103	120	16	130
SPECint 2000	904	1,566	1,521	1,398	483	**1,590**
SPECfp 2000	1279	1,591	1,504	2,576	499	**2,712**

Microprocessor Report, December 2004

At peak minus 5 years

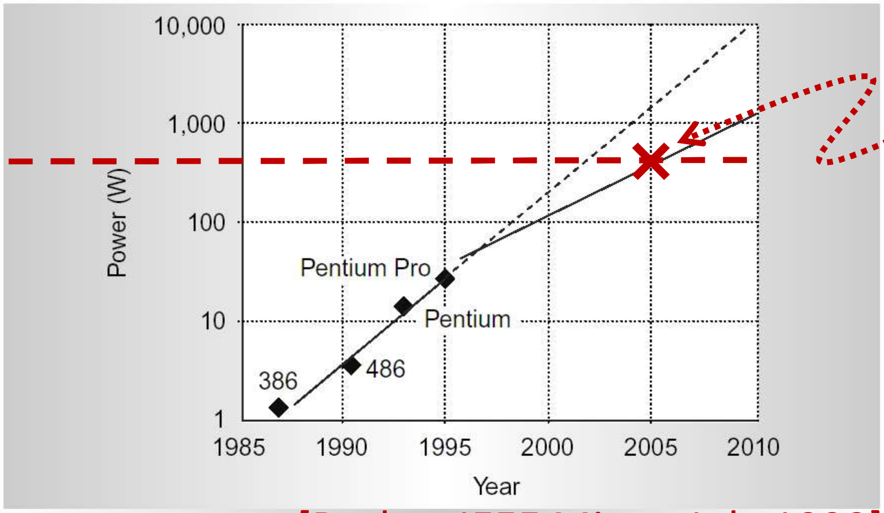
	Alpha 21264	AMD Athlon	Intel P4	MIPS R12000	IBM Power3	HP PA8600	SUN Ultra3
clock (MHz)	833	1200	**1500**	400	450	552	900
issue rate	4	3 (x86)	3 (rop)	4	4	4	4
pipeline int/fp	7/9	9/11	**22/24**	6	7/8	7/9	14//15
inst in flight	80	72(rop)	**126** rop	48	32	56	inorder
rename reg	48+41	36+36	**128**	32+32	16+24	56	inorder
transistor (10^6)	15.4	37	42	7.2	23	**130**	29
power (W)	75	**76**	55	25	36	60	65
SPECint 2000	518		**524**	320	286	417	438
SPECfp 2000	**590**	304	549	319	356	400	427

Microprocessor Report, December 2000

Performance (In)efficiency

- To hit "expected" performance target
  - push frequency harder by deepening pipelines
  - used the 2x transistors to build more complicated microarchitectures so fast/deep pipelines don't stall (i.e., caches, BP, superscalar, out-of-order)
- The consequence of performance inefficiency is

limit of economical cooling [ITRS]



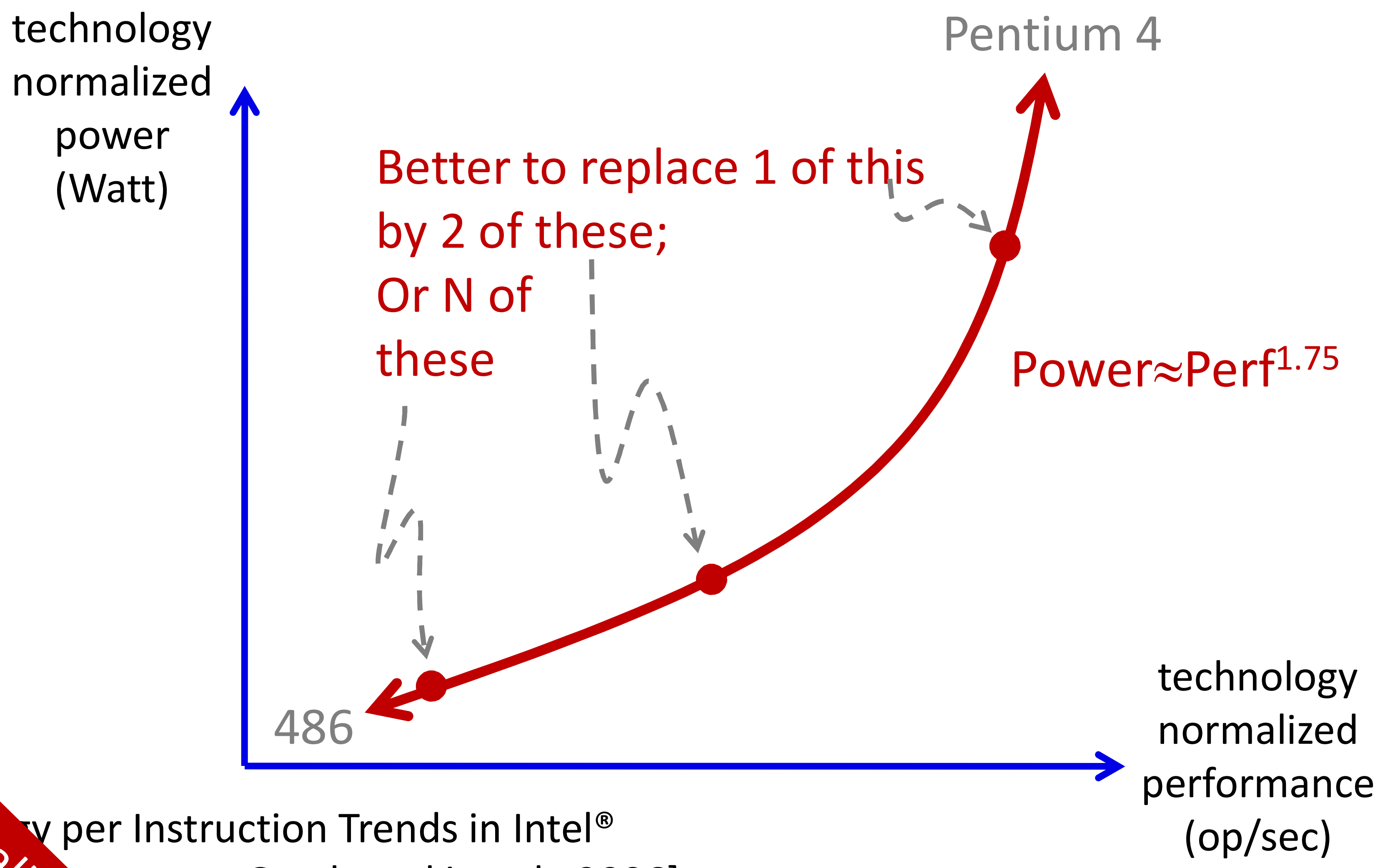
2005, Intel P4 Tehas 150W

Figure 8. Power dissipation projections.

[Borkar, IEEE Micro, July 1999]

Recall

Efficiency of Parallel Processing



Recall

[y per Instruction Trends in Intel®
Mi cessors, Grochowski et al., 2006]

At peak plus 1 year

	AMD 285	Intel 5160	Intel 965	Intel Itanium2	IBM P5+	MIPS R16000	SUN Ultra4
cores/threads	**2x1**	**2x2**	**2x2**	**2x2**	**2x2**	1x1	**2x1**
clock (GHz)	2.6	3.03	**3.73**	1.6	2.3	0.7	1.8
issue rate	3 (x86)	4 (rop)	3 (rop)	6	**8**	4	4
pipeline depth	11	14	**31**	8	17	6	14
inst in flight	72(rop)	96(rop)	126(rop)	inorder	**200**	48	inorder
on-chip$ (MB)	2x1	4	2x2	**2x13**	1.9	0.064	2
transistor (10^6)	233	291	376	**1700**	276	7.2	295
power (W)	95	80	**130**	104	100	17	90
SPECint 2000 per core	**1942**	(1556*)	1870	1474	1820	560	1300
SPECfp 2000 per core	2260	(1694+)	2232	3017	**3369**	580	1800

*3086/+2884 according to www.spec.org

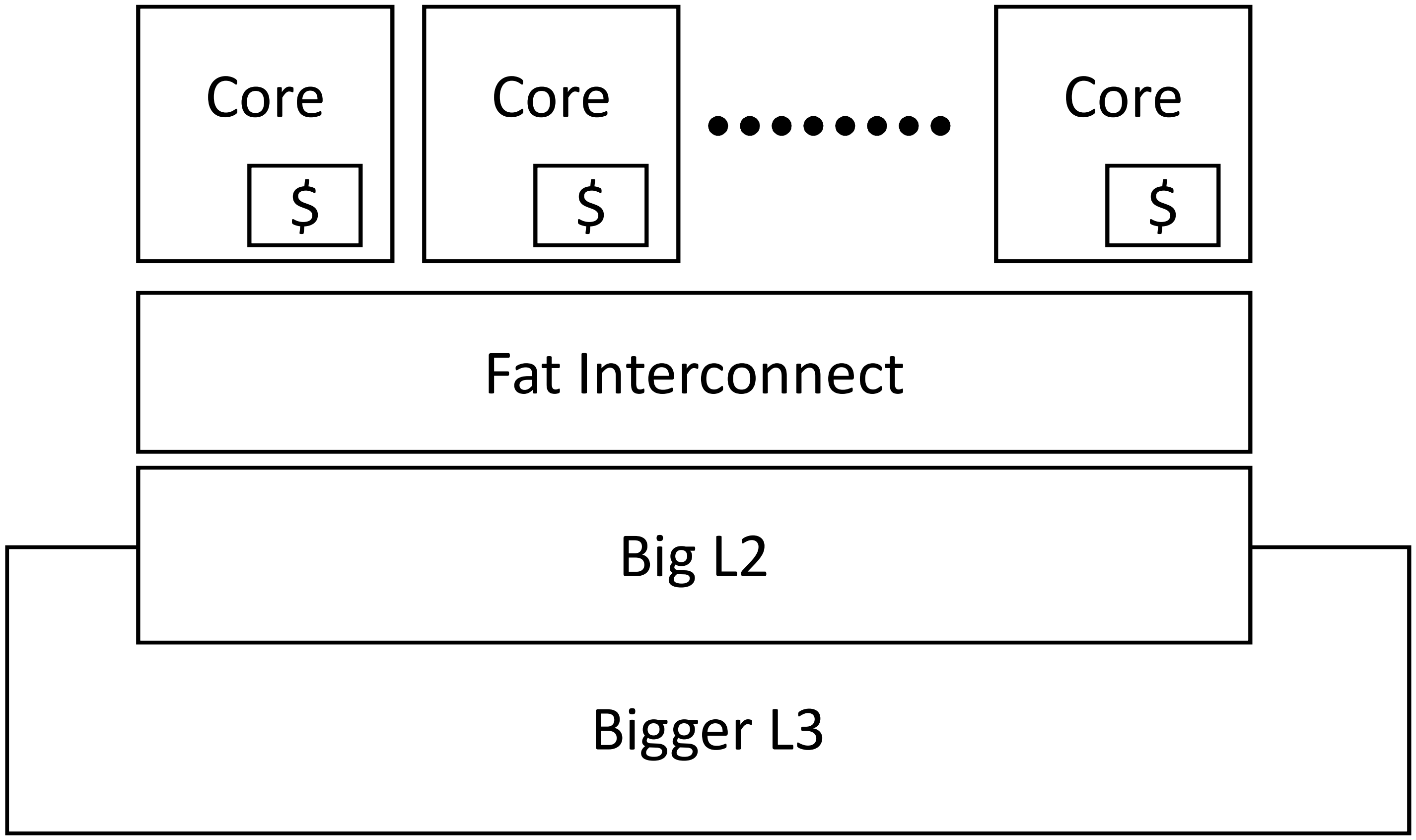
Microprocessor Report, Aug 2006

At peak plus 3 years

	AMD Opteron 8360SE	Intel Xeon X7460	Intel Itanium 9050	IBM P5	IBM P6	Fijitsu SPARC 7	SUN T2
cores/threads	4x1	**6x1**	2x2	2x2	2x2	4x2	**8x8**
clock (GHz)	2.5	2.67	1.60	2.2	**5**	2.52	1.8
issue rate	3 (x86)	4 (rop)	6	5	**7**	4	2
pipeline depth	12/**17**	14	8	15	13	15	8/12
out-of-order	72(rop)	96(rop)	inorder	**200**	limited	64	inorder
on-chip$ (MB)	2+2	**9+16**	1+12	1.92	8	6	4
transistor (10^6)	463	**1900**	1720	276	790	600	503
power max(W)	105	130	104	100	>100	**135**	95
SPECint 2006 per-core/total	14.4/170	**22**/274	14.5/1534	10.5/197	15.8/1837	10.5/**2088**	--/142
SPECfp 2006 per-core/total	18.5/156	22/142	17.3/1671	12.9/229	20.1/1822	**25.0**/**1861**	--/111

Microprocessor Report, Oct 2008

On to Mainstream Parallelism in Multicores and Manycores



Remember, we got here because we need to compute <u>faster while using less energy per operation</u>