18-447 Lecture 20: ILP to Multicores

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today
  - transition from sequential to parallel
  - enjoy (only first part, before OOO, on 447 exam)
- Notices
  - Midterm Regrades past due
  - HW4, due 4/8
  - Lab 4, less than 4 weeks to go
  - **NO LECUTRE on MONDAY**
- Readings (advanced optional)
  - MIPS R10K Superscalar Microprocessor, Yeager
  - Synthesis Lectures: *Processor Microarchitecture: An Implementation Perspective, 2010*

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*

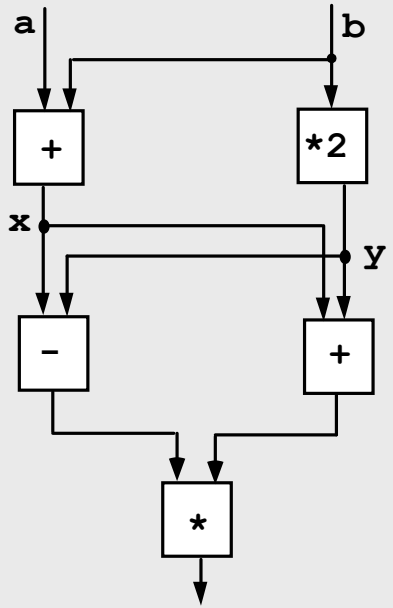
- For a system with **p** PEs

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

- When $P_{avg} >> p$

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

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



[Shiloach&Vishkin]

ILP: Instruction-Level Parallelism

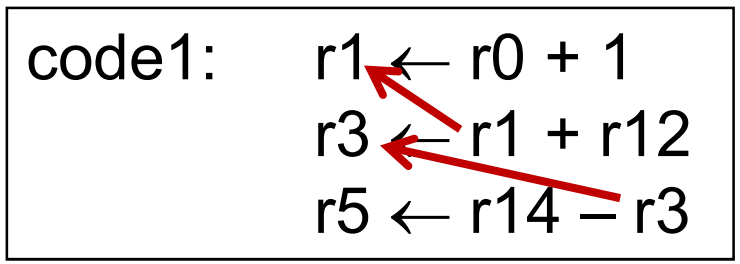
- Average **ILP** = T_1 / T_∞
  = no. instruction / no. cyc required

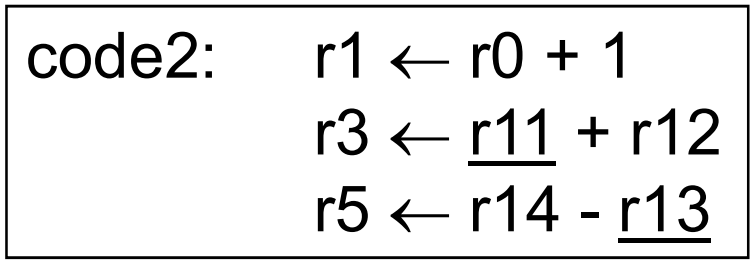
code1: **ILP** = 1

i.e., must execute serially

code2: **ILP** = 3

i.e., can execute at the same time



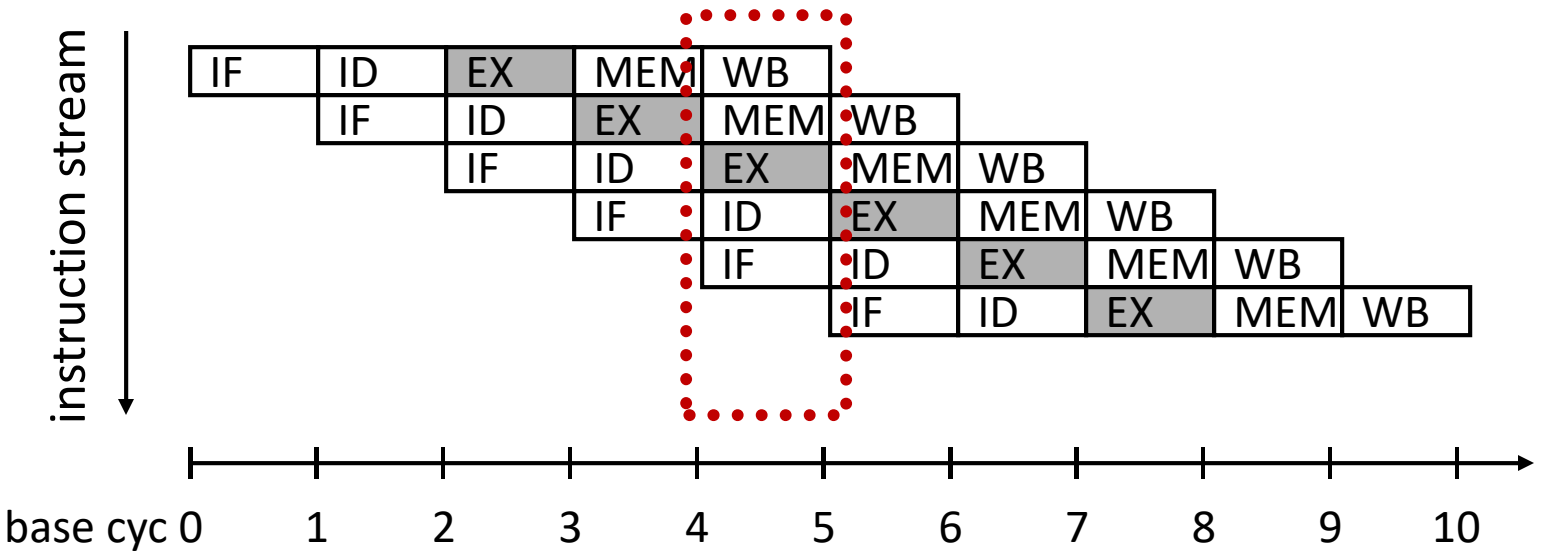


Add Superscalar to Pipelining to get more Performance from ILP

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

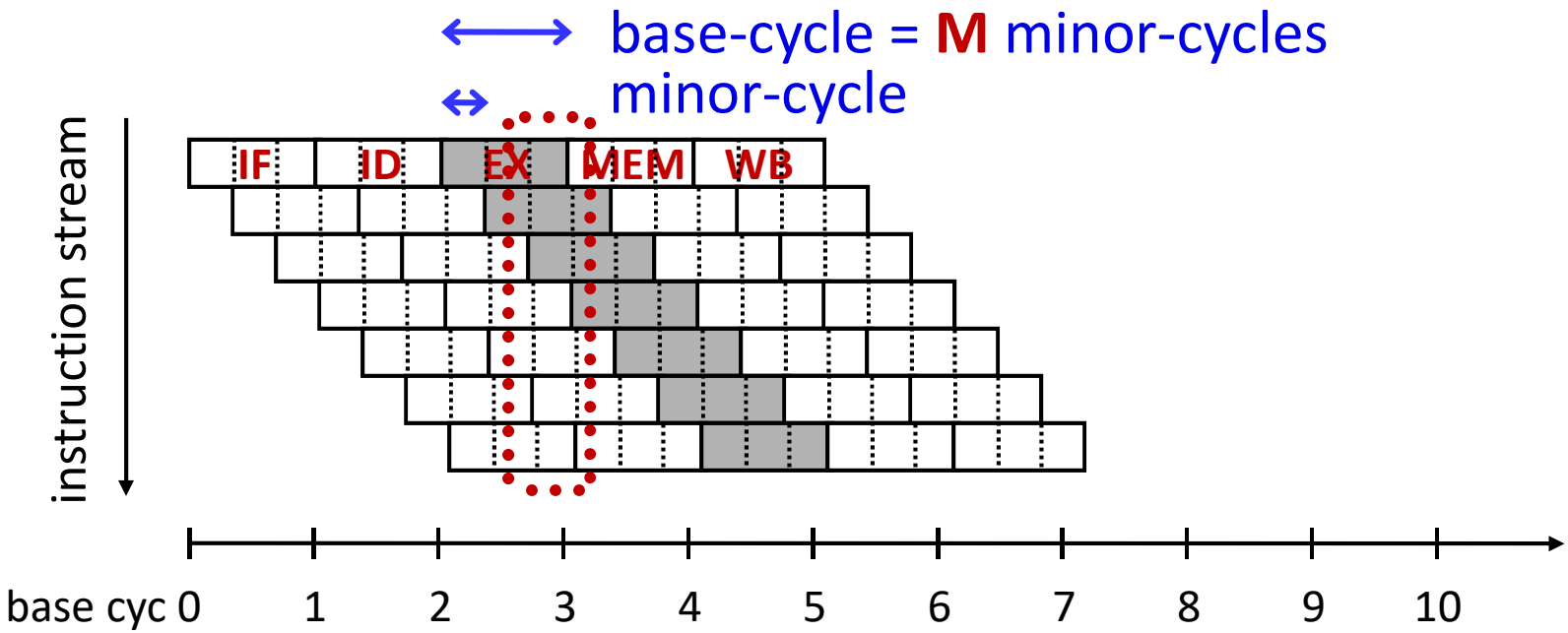


Superpipelined Execution

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

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

required **ILP** ≥ **M**



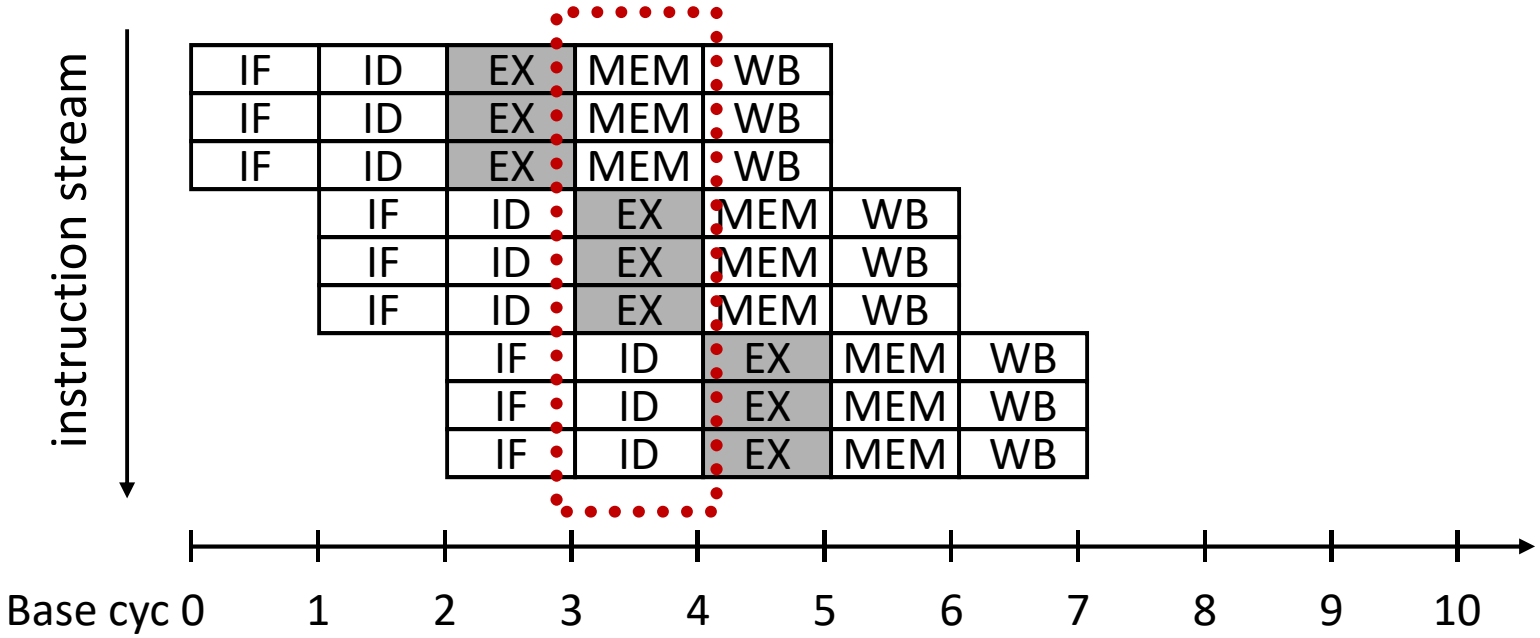
Achieving full performance requires always finding **M** "independent" instructions in a row

Superscalar (Inorder) Execution

OL = **1** base cycle

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

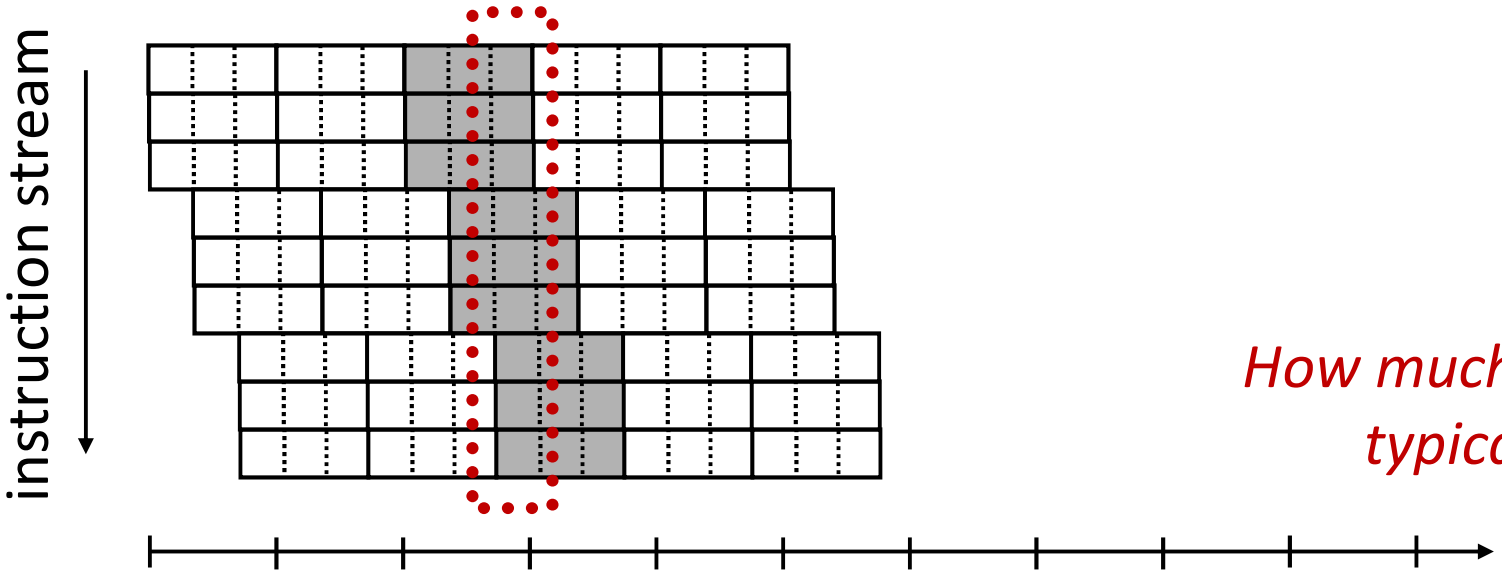
required **ILP** ≥ **N**



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

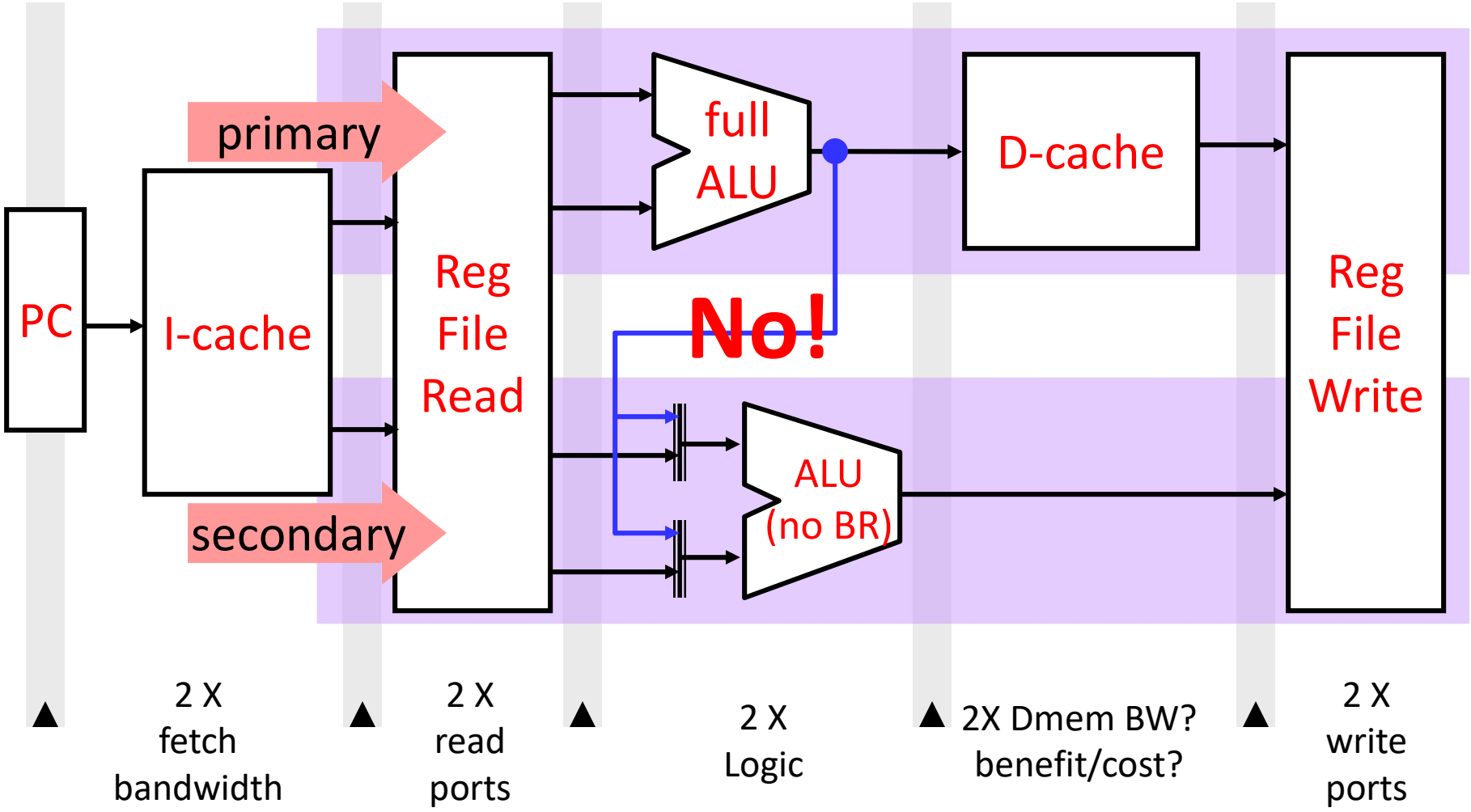
Limitations of Inorder Pipeline

- 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!!



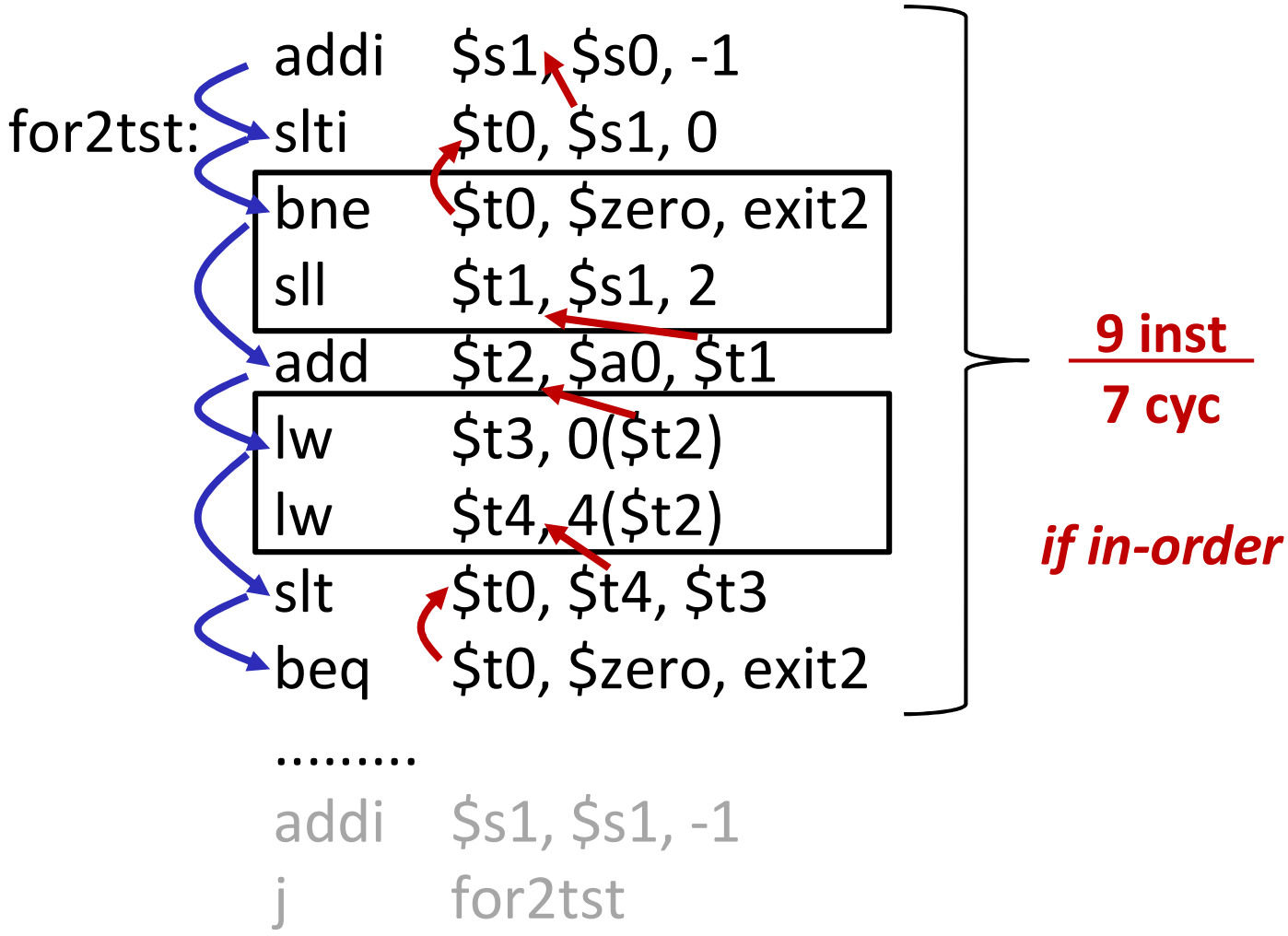
How much ILP typically?

E.g., 2-way, In-order Superscalar



Sample Assembly [P&H]

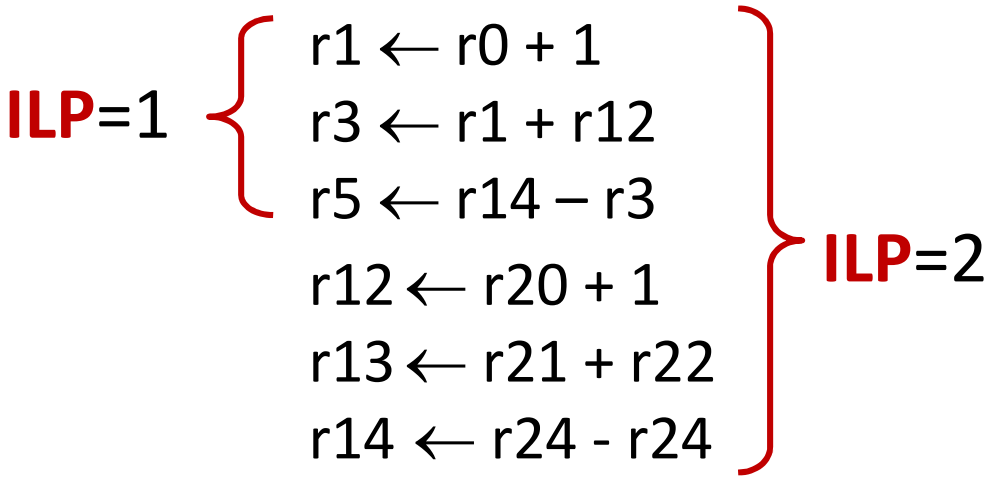
for (j=i-1; j>=0 && v[j] > v[j+1]; j-=1) { ..... }



Assume all inst 1 cyc

Out-of-Order Execution for ILP

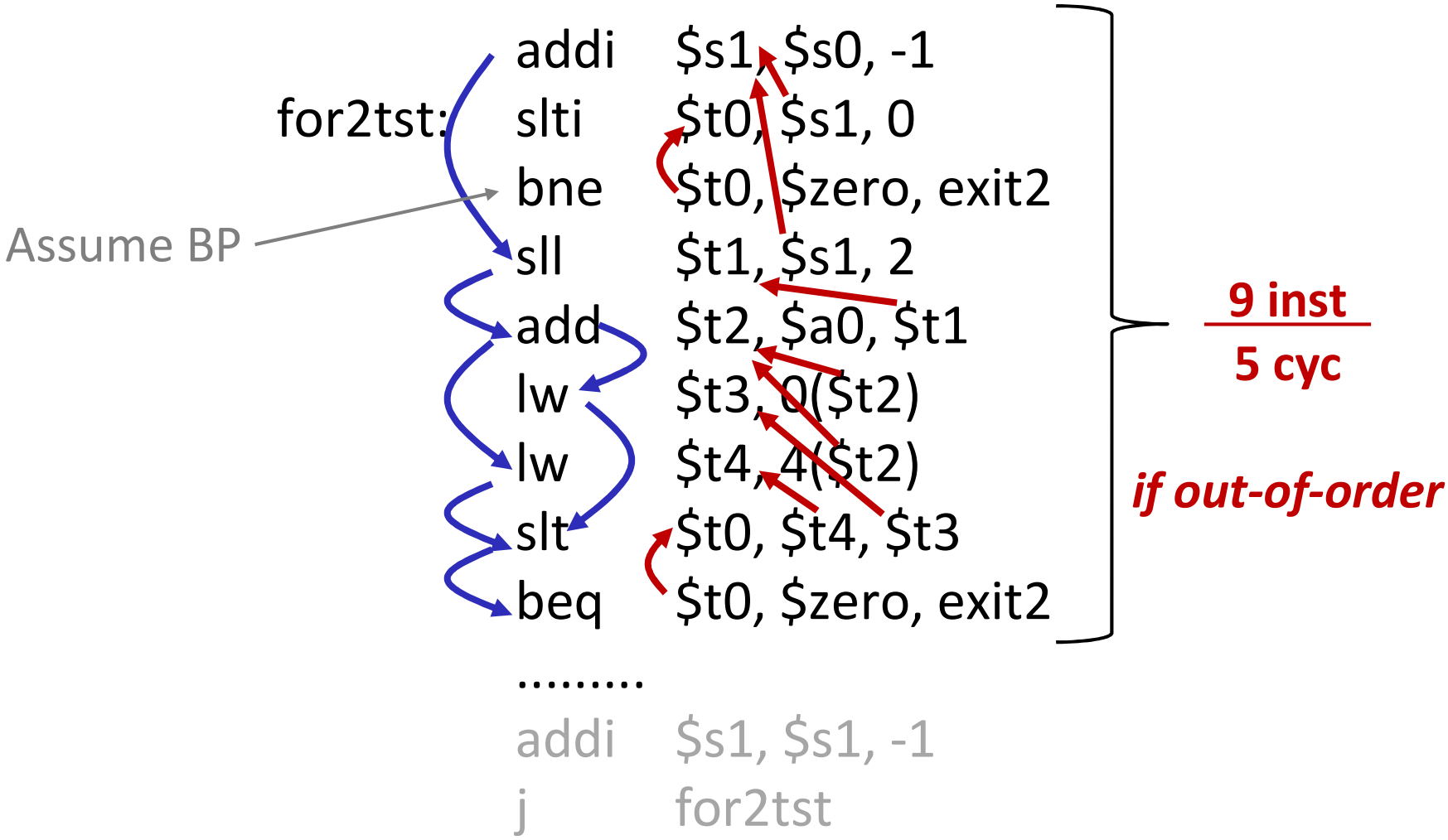
- **ILP** is scope dependent



Accessing **ILP**=2 requires not only (1) larger scheduling window but also (2) out-of-order execution and (3) handling WAW and WAR

Sample Assembly [P&H]

for (j=i-1; j>=0 && v[j] > v[j+1]; j-=1) { ..... }



Assume all inst 1 cyc

Pass this point not on exams

For more, go read "Synthesis Lectures: Processor Microarchitecture: An Implementation Perspective," 2010

Add Out-of-Order to Superscalar to get more ILP

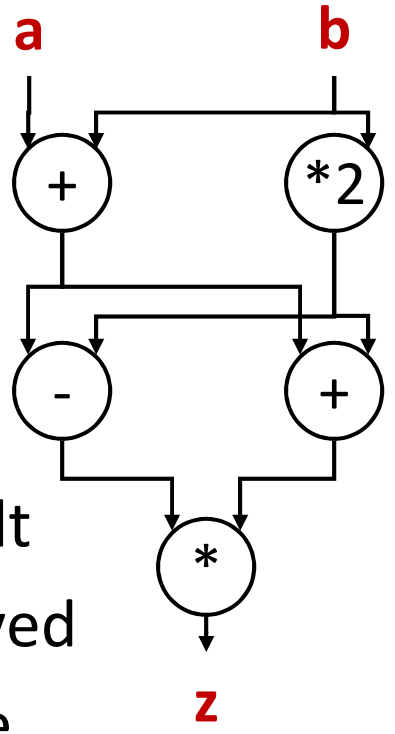
How much more?

And What for?

von Neuman vs Dataflow

- Consider a von Neumann program
  - What is the significance of the program order?
  - What is the significance of the storage locations?

```
v := a + b;
w := b * 2;
x := v - w;
y := v + w;
z := x * y;
```



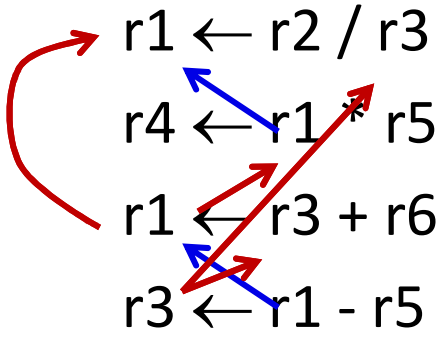
- Dataflow program instruction ordering implied by data dependence
  - instruction specifies who receives the result
  - instruction executes when operands received
  - no program counter, no intermediate state

[dataflow figure and example from Arvind]

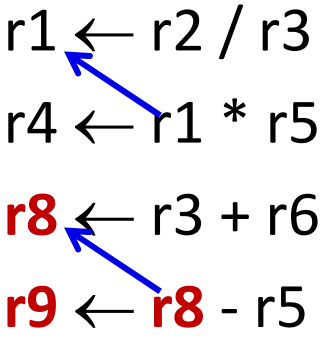
Recall

Converting von Neumann to Dataflow: Register Renaming

Original



Renamed



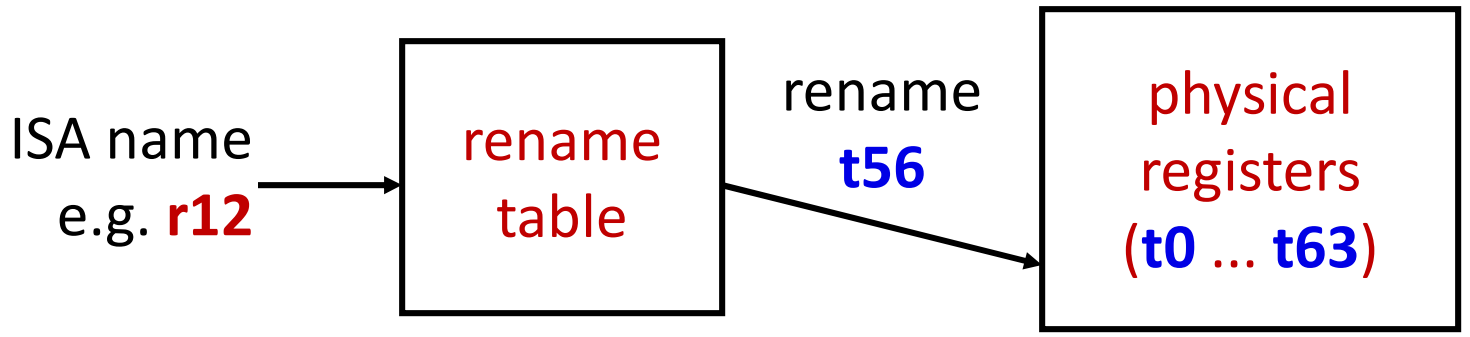
- von Neumann model relies on finite state to convey producer-consumer dependence

  $\Rightarrow$ must reuse state in correct order

- If infinite registers, use a new register for each new value $\Rightarrow$ no WAW, no WAR, just dataflow left

Can μarch fake infinite registers?

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 ← r2 / r3
r4 ← r1 * r5
r1 ← r3 + r6

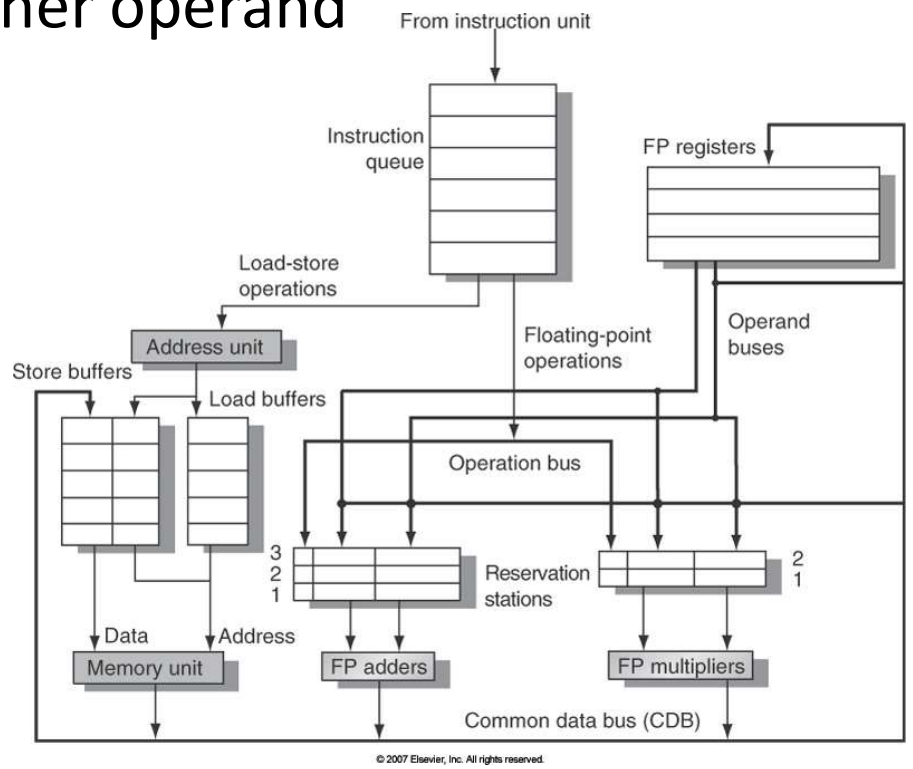
Register Micro-Dataflow

- Maintain a buffer of many pending renamed 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

Memory also out-of-order but much harder

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



Control Dependence also Limits ILP

- Average basic block size is only 6~7 instructions
- In-order pipelines used branch prediction to cover delay from fetch to branch resolution

  Expect 1 or less BP in flight in 5-stage pipeline

- Out-of-order pipelines must speculate past many levels of BP to present a large out-of-order window of instructions!

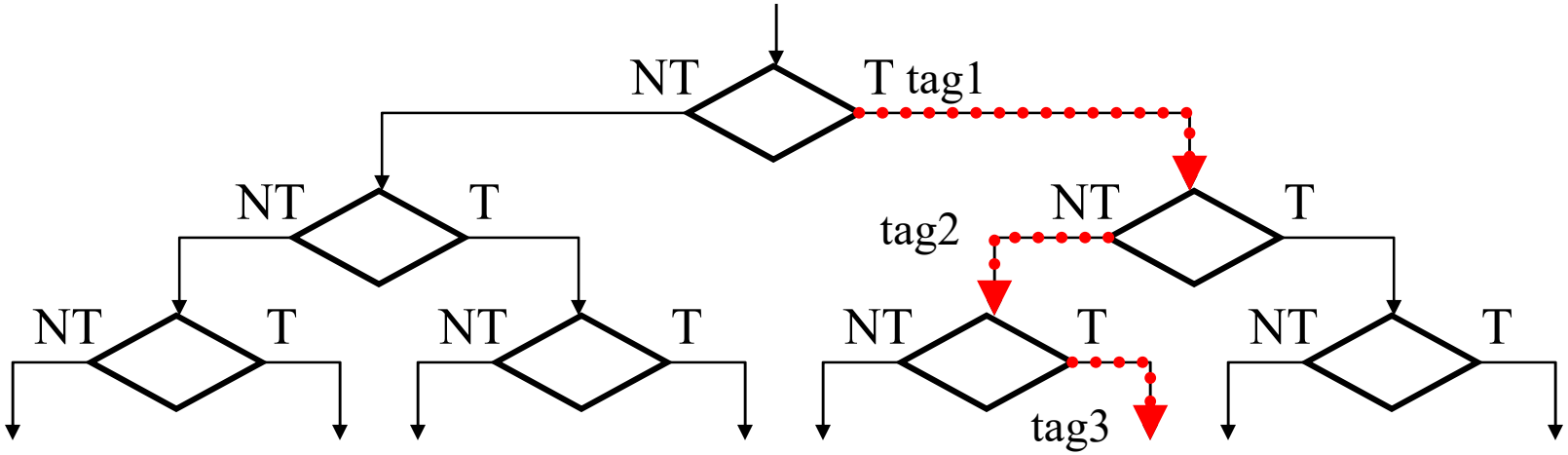
  Over 100 insts out-of-order in modern superscalar OOO CPUs

Add Speculation to Superscalar OOO to get still more ILP (??)

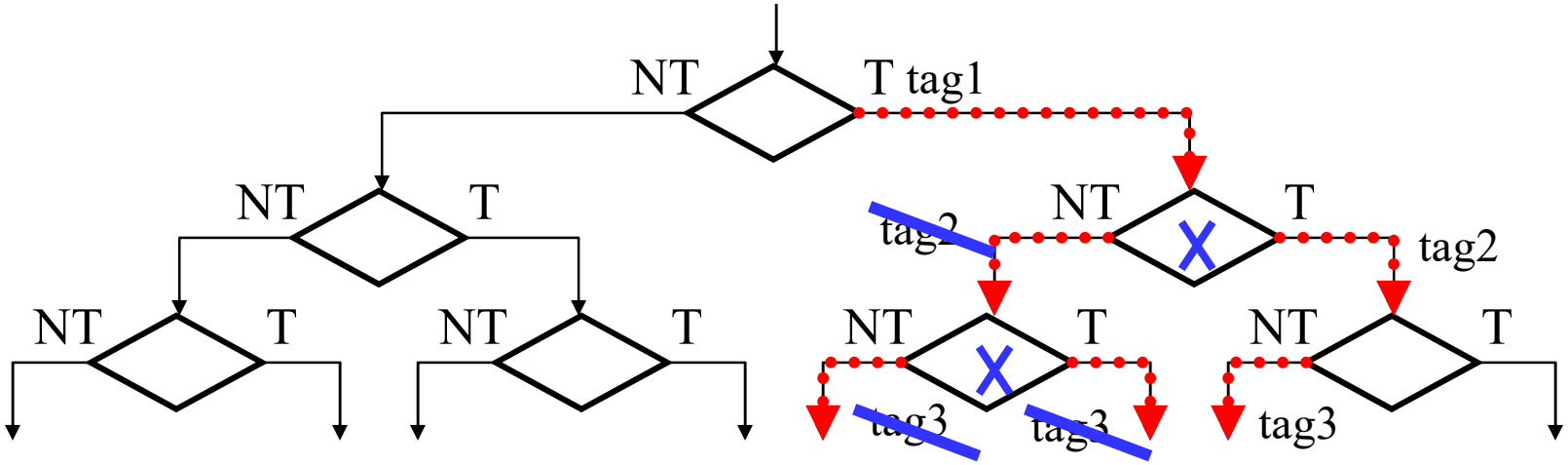
how much more ILP can be uncovered?
how much useful work will be done?

Ans: not much and not much

Nested Control Flow Speculation



Mis-speculation Recovery can be Speculative

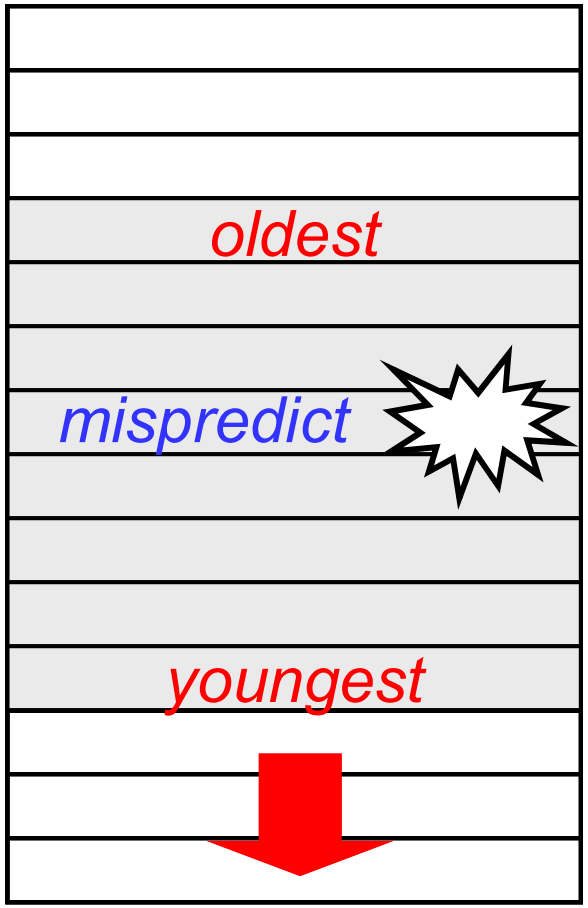


Speculative Out-of-order Execution

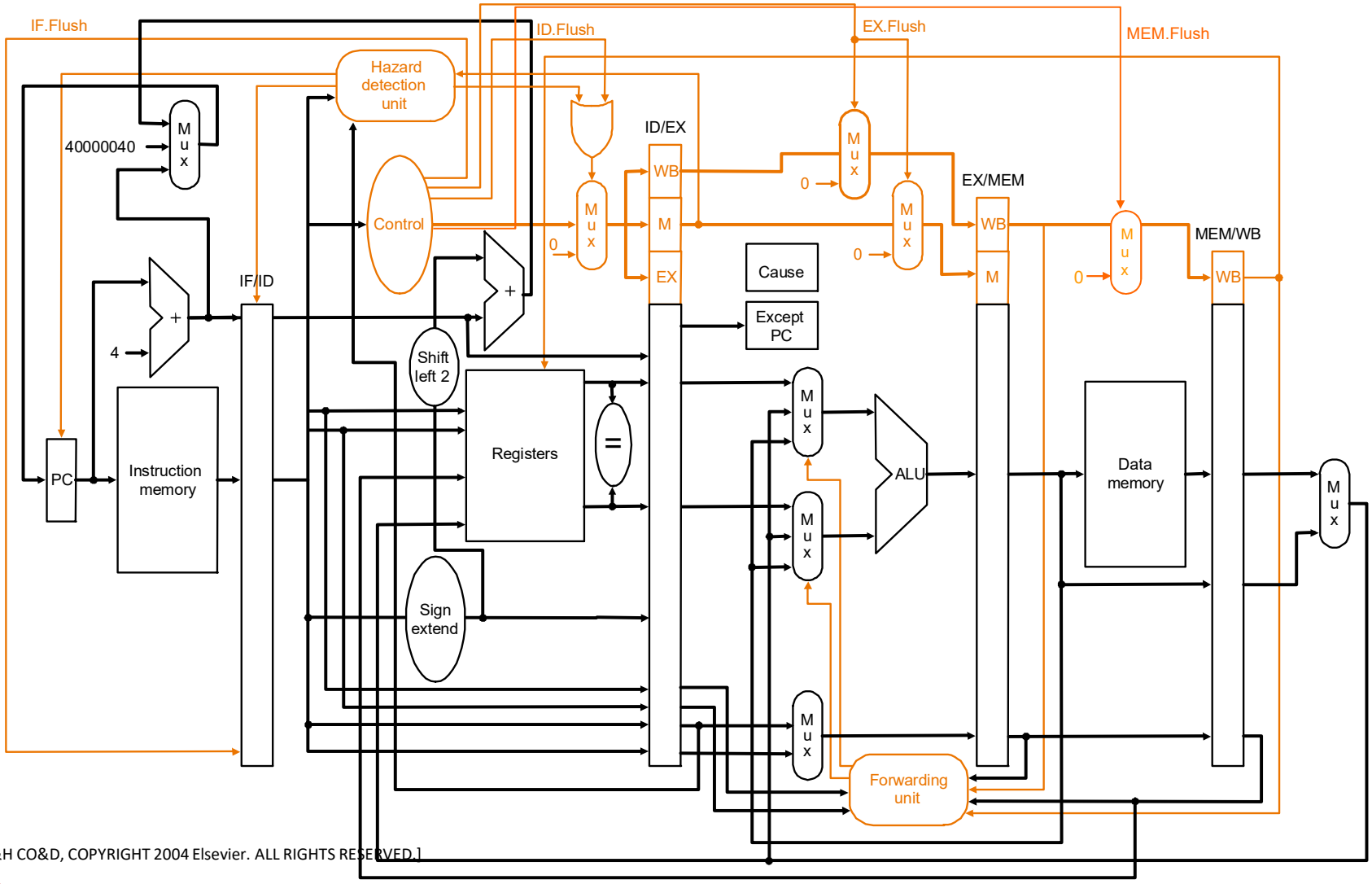
- A mispredicted branch after resolution must be rewound and restarted **ASAP**!
- Much trickier than 5-stage pipeline . . .
  - can rewind to an intermediate speculative state
  - a rewound branch could still be speculative and itself be discarded by another rewind!
  - rewind must reestablish both 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

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



You have seen something like this



[P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

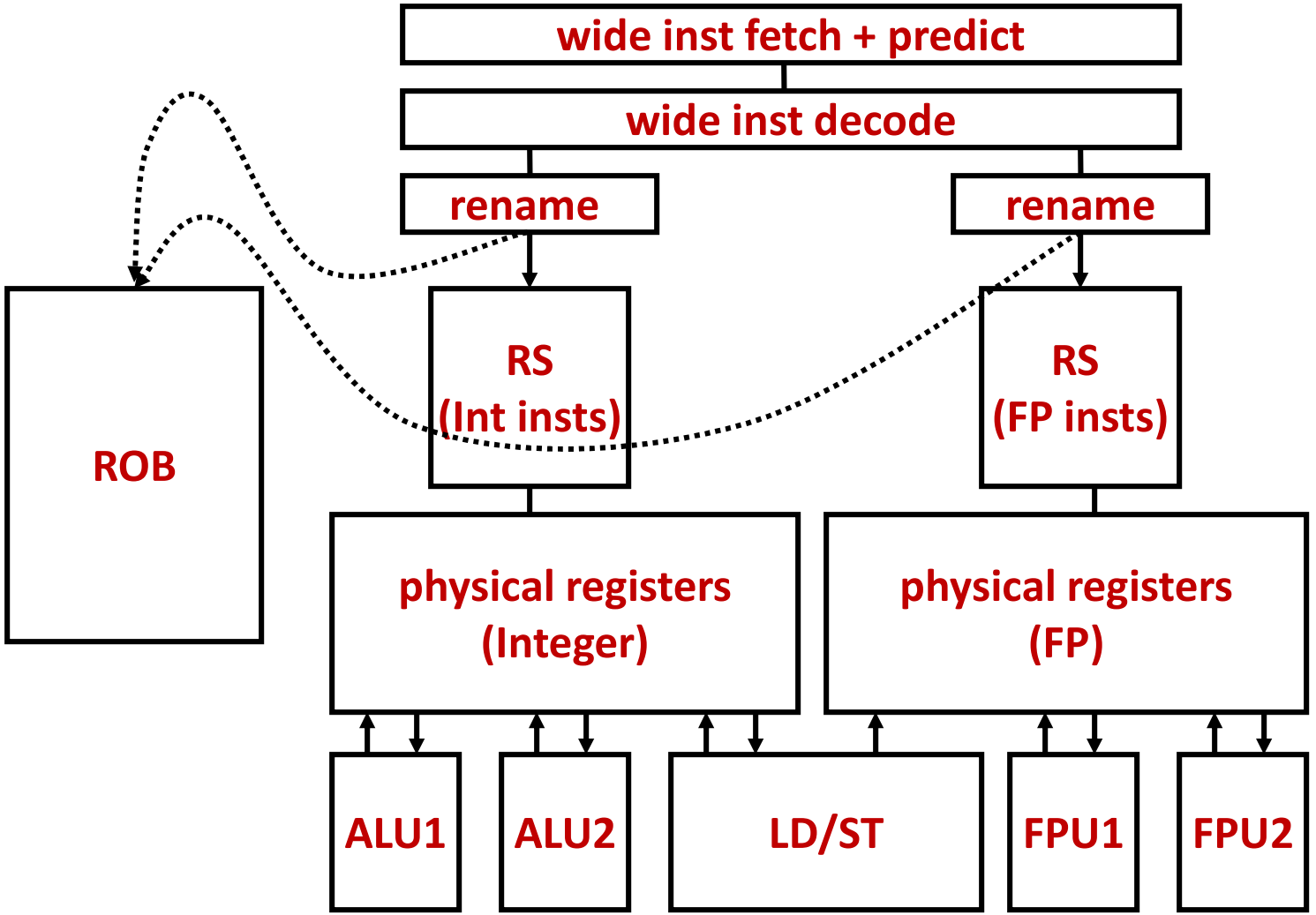
Recall

Where is "the current instruction"?

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**

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 (aka MLP)

At the 2005 Peak of Superscalar OOO

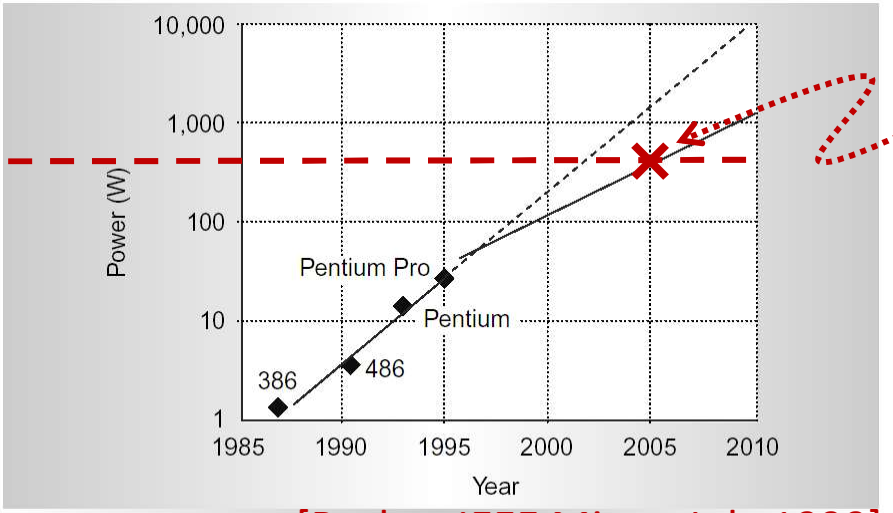
	Alpha 21364	AMD Opteron	Intel Xeon	IBM Power5	MIPS R14000	Intel Itanium2
clock (GHz)	1.30	2.4	**3.6**	1.9	0.6	1.6
issue rate	4	3 (x86)	3 (rop)	**8**	4	8
pipeline int/fp	7/9	9/11	**22/24**	12/17	6	8
inst in flight	80	72(rop)	126 rop	**200**	48	inorder
rename reg	48+41	36+36	128	48/40	32/32	**328**
transistor (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

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]



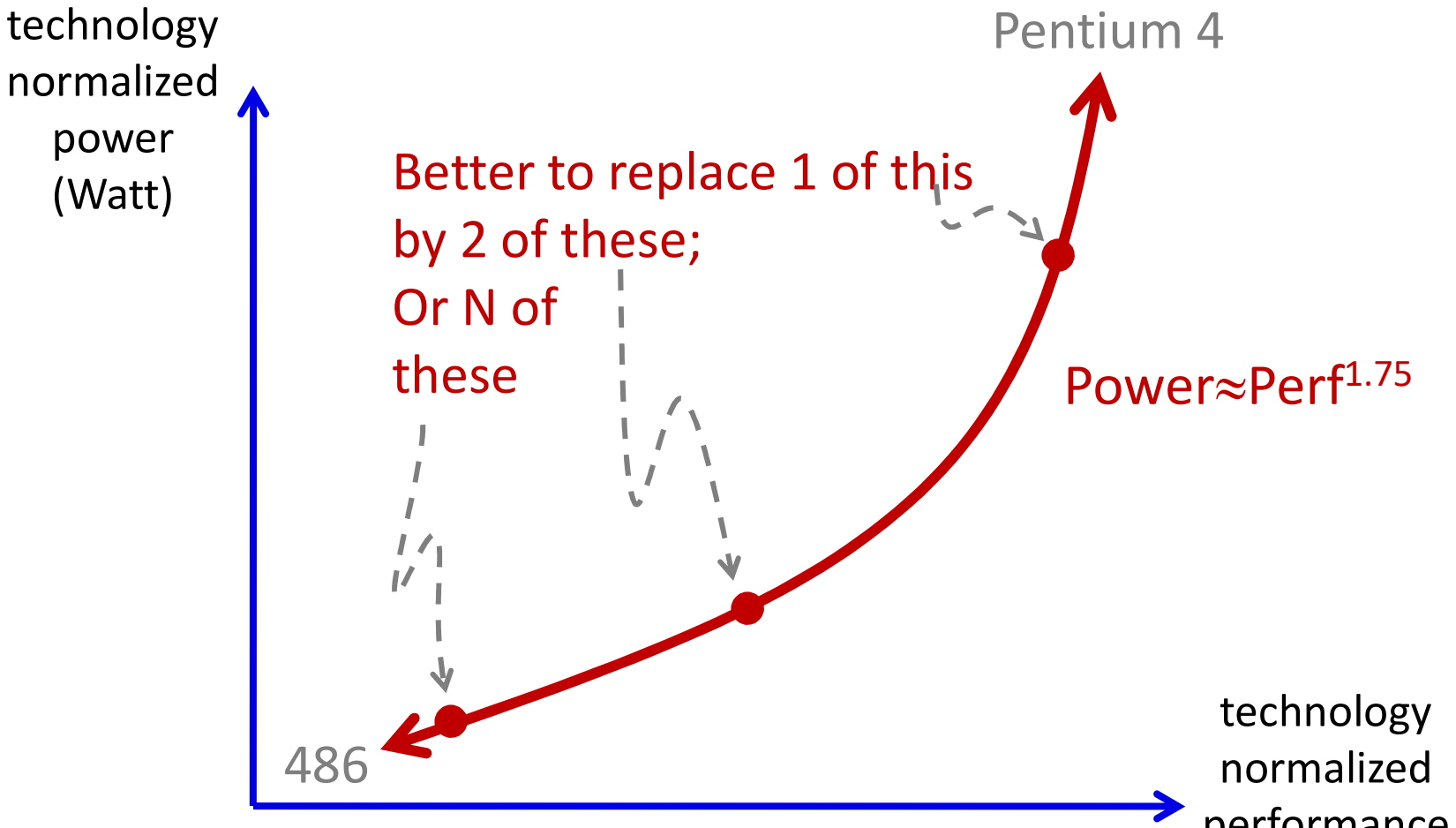
2005, Intel P4 Tehas 150W

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

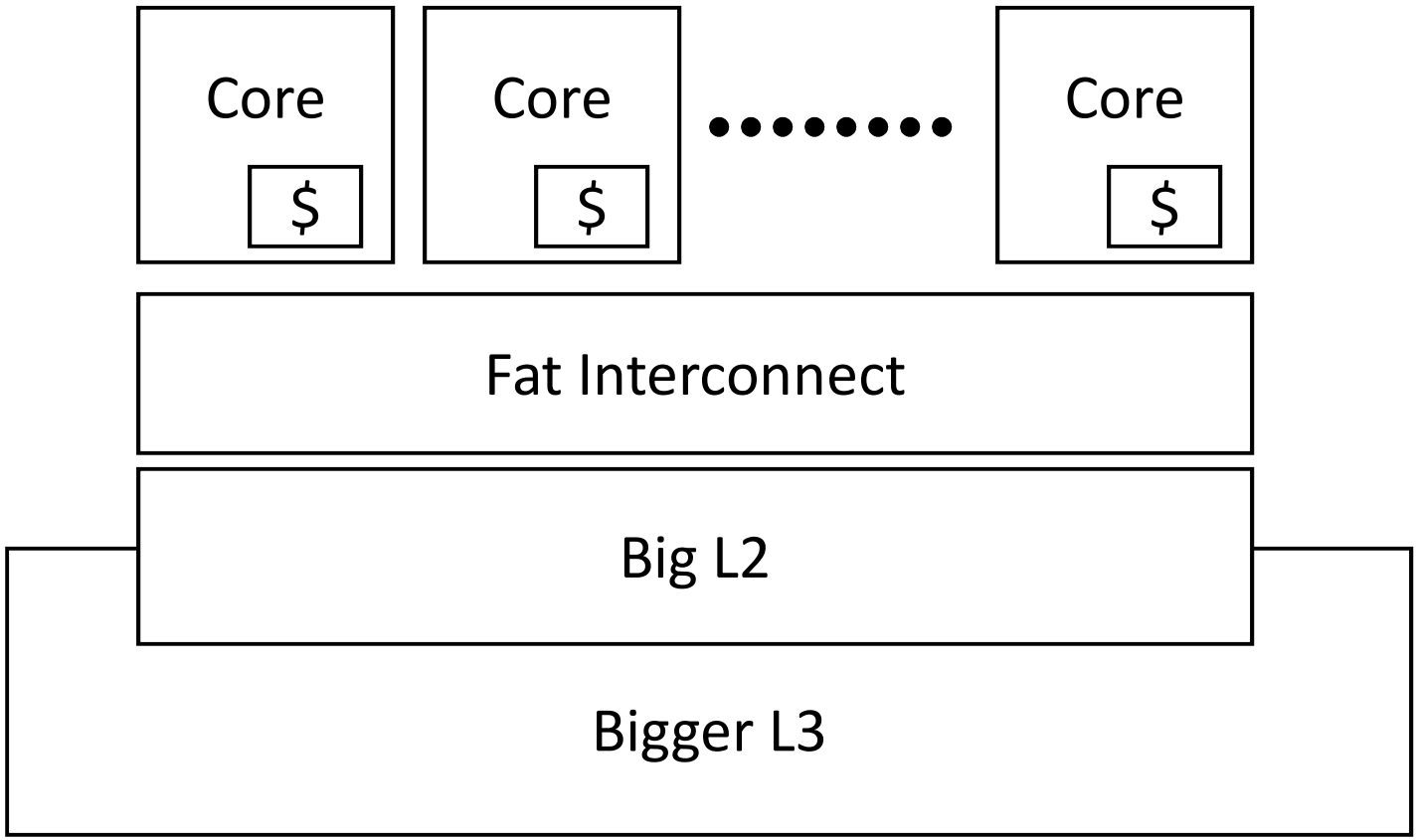
Microprocessor Report, Aug 2006

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 faster while using less energy per operation