Lecture 9: Pipelined CPU Implementation: Control Hazards and Resolutions

Jangwoo Kim (Seoul National University)
jangwoo@snu.ac.kr

Announcement

- Are you reading the textbook?
  - Finish P&H Chapter 4! **No Read → No Hope!**
- Are you doing the project?
  - Understand the reference solution! **No Project → No Hope!**
- Homework #3 to be posted
  - **Due: 4/19**
- Mid-term Exam
  - **Mid-term 4/20 (Friday, 6:00pm – forever?)**

Review: Data Dependence

Data dependence

$r_3 \leftarrow r_1$ op r_2
$r_5 \leftarrow r_3$ op r_4

Read-after-Write (RAW)

Anti-dependence

$r_3 \leftarrow r_1$ op r_2
$r_1 \leftarrow r_4$ op r_5

Write-after-Read (WAR)

Output-dependence

$r_3 \leftarrow r_1$ op r_2
$r_5 \leftarrow r_3$ op r_4
$r_3 \leftarrow r_6$ op r_7

Write-after-Write (WAW)

We discuss control-flow dependence today!

Instruction Ordering/Dependencies

- Data Dependence
  - True dependence or Read after Write (RAW)
    Instruction must wait for all required input operands
  - Anti-Dependence or Write after Read (WAR)
    Later write must not affect a still-pending earlier read
  - Output dependence or Write after Write (WAW)
    Earlier write must not affect an already-finished later write

- Control Dependence (or Procedural Dependence)
  - All instructions are dependent by control flow
  - Every instruction uses and sets the PC

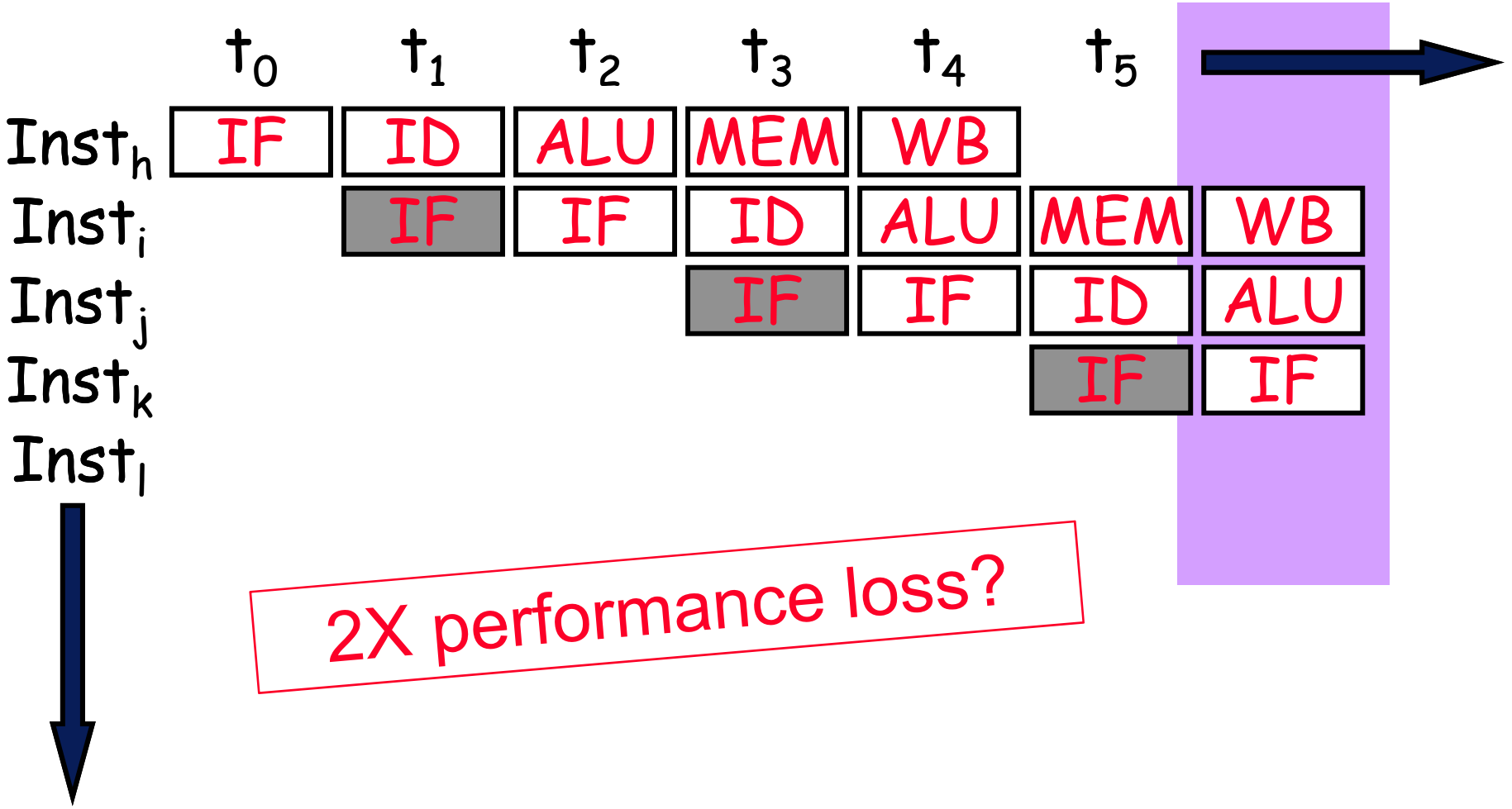
  In other words, control dependence is **data dependence** on the PC

"PC" Data Hazard Analysis

	R/I-Type	LW	SW	Br	J	Jr
IF	use	use	use	use	use	use
ID	produce	produce	produce		produce	produce
EX				produce		
MEM						
WB						

- PC hazard distance is at least 1
- Does that mean we must stall after every instruction?
  - IF stage can't know which PC to fetch next until the current PC is fetched and decoded

Control Hazard by Stalling



For non-control-flow instructions

HPCS
High Performance Computer System Lab

Simplest Branch Prediction → Predict not-taken or PC+4

- Rather than waiting for true-dependence on PC to resolve, just **guess** nextPC = PC+4 to keep fetching every cycle

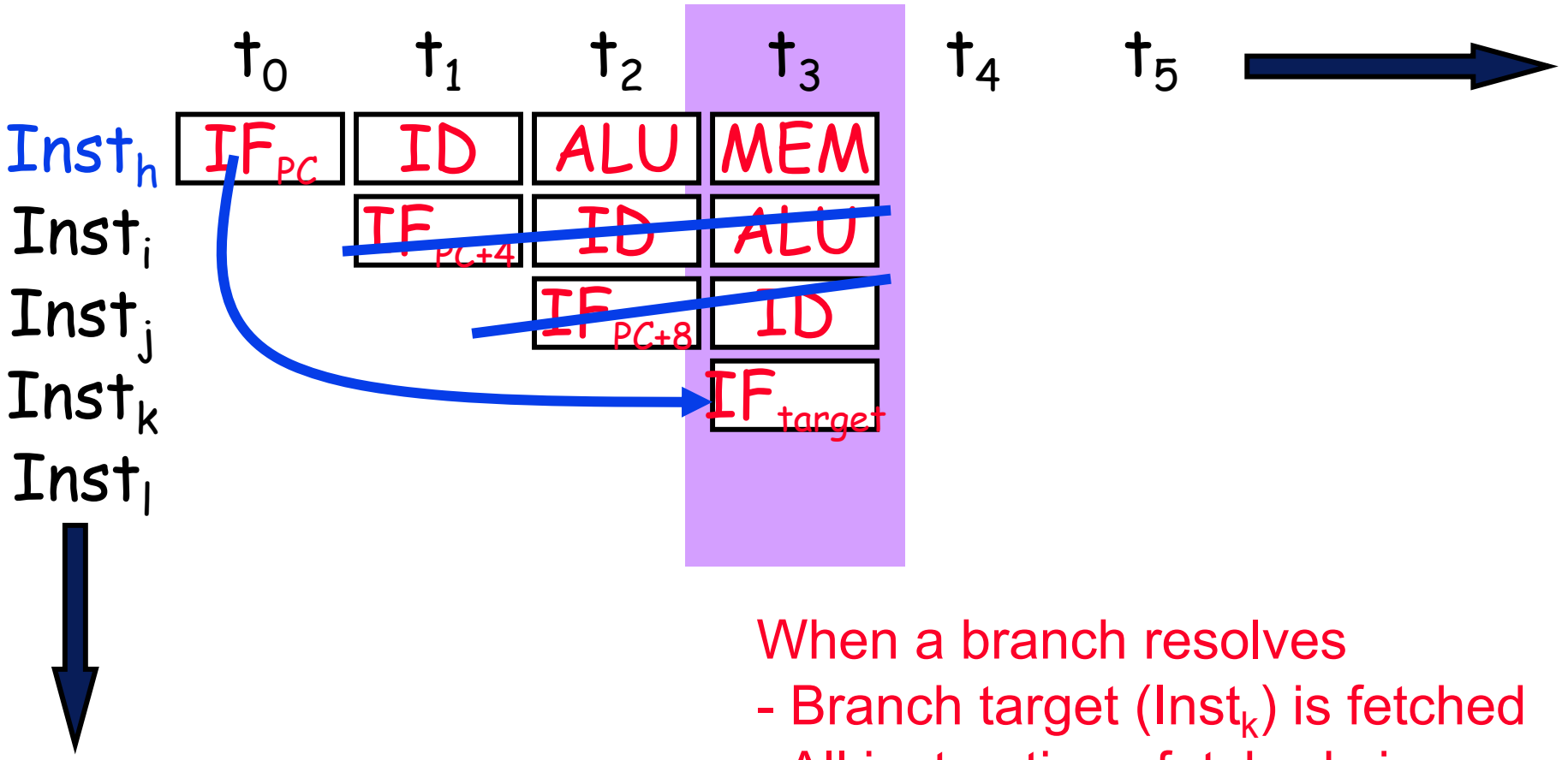
Is this a good guess? What do you lose if you guessed incorrectly?

- Only **~20%** of the instruction mix is control flow
  - ~50 % of "forward" control flow (i.e., if-then-else) is taken
  - ~90% of "backward" control flow (i.e., loop back) is taken

  Overall, typically ~70% taken and **~30%** not taken
  [Lee and Smith, 1984]

- Expect "nextPC = PC+4" ~86% of the time, but what about the remaining 14%?

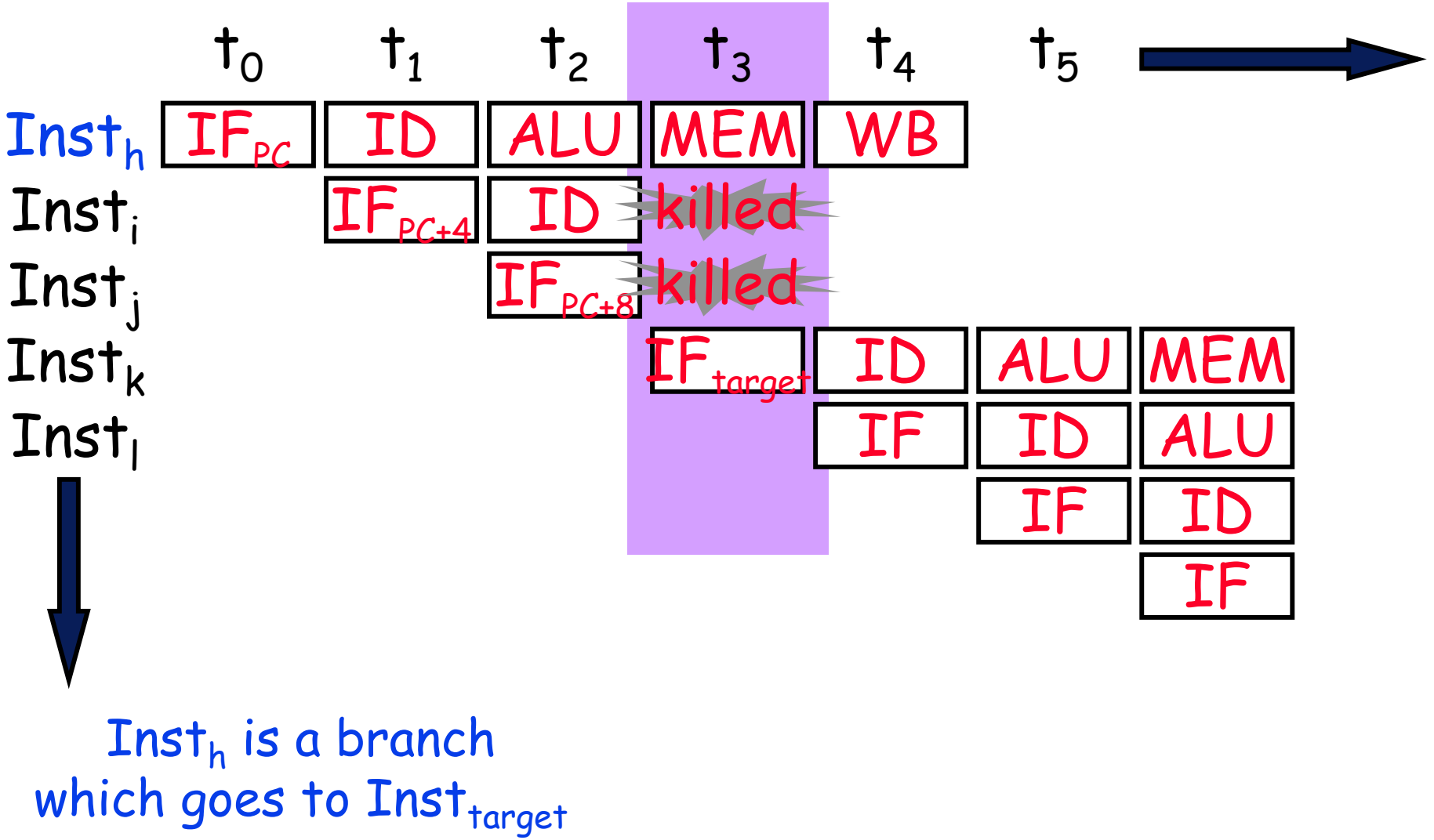
Control Speculation: PC+4



Insth is a branch which goes to Insttarget

When a branch resolves
- Branch target ($Inst_k$) is fetched
- All instructions fetched since $inst_h$ (so called "**wrong-path**" instructions) must be flushed

Pipeline Flush on Misprediction



$Inst_h$ is a branch
which goes to $Inst_{target}$

Pipeline Flush on Misprediction

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
IF	h	i	j	k	l	m	n				
ID		h	i	bub	k	l	m	n			
EX			h	bub	bub	k	l	m	n		
MEM				h	bub	bub	k	l	m	n	
WB					h	bub	bub	k	l	m	n

Branch resolved

[Textbook says 3 cycle loss. Why?]

Performance Impact (PC+4 prediction)

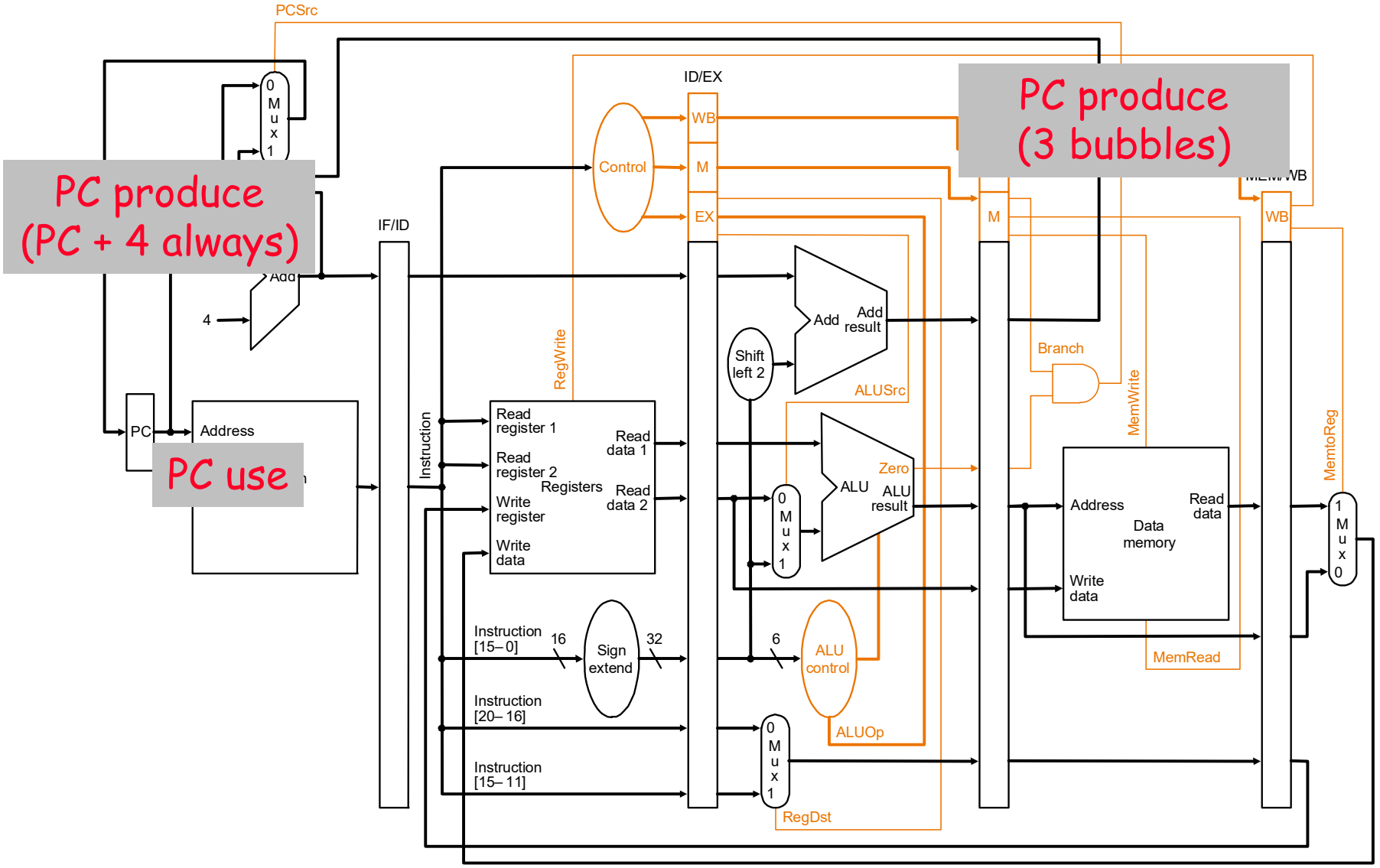
- Correct guess → No penalty **~86%** of the time
- Incorrect guess → 2 bubbles (or 3 bubbles in textbook)
- Assume
  - No data hazards
  - 20% control flow instructions
  - 70% of control flow instructions are taken
  - IPC = 1 / [1 + (0.2*0.7) * (2 or 3)] =
    = 1 / [1 + **0.14** * (2 or 3)] = 1 / 1.28 = 0.78 or 0.70

Probability of a wrong guess

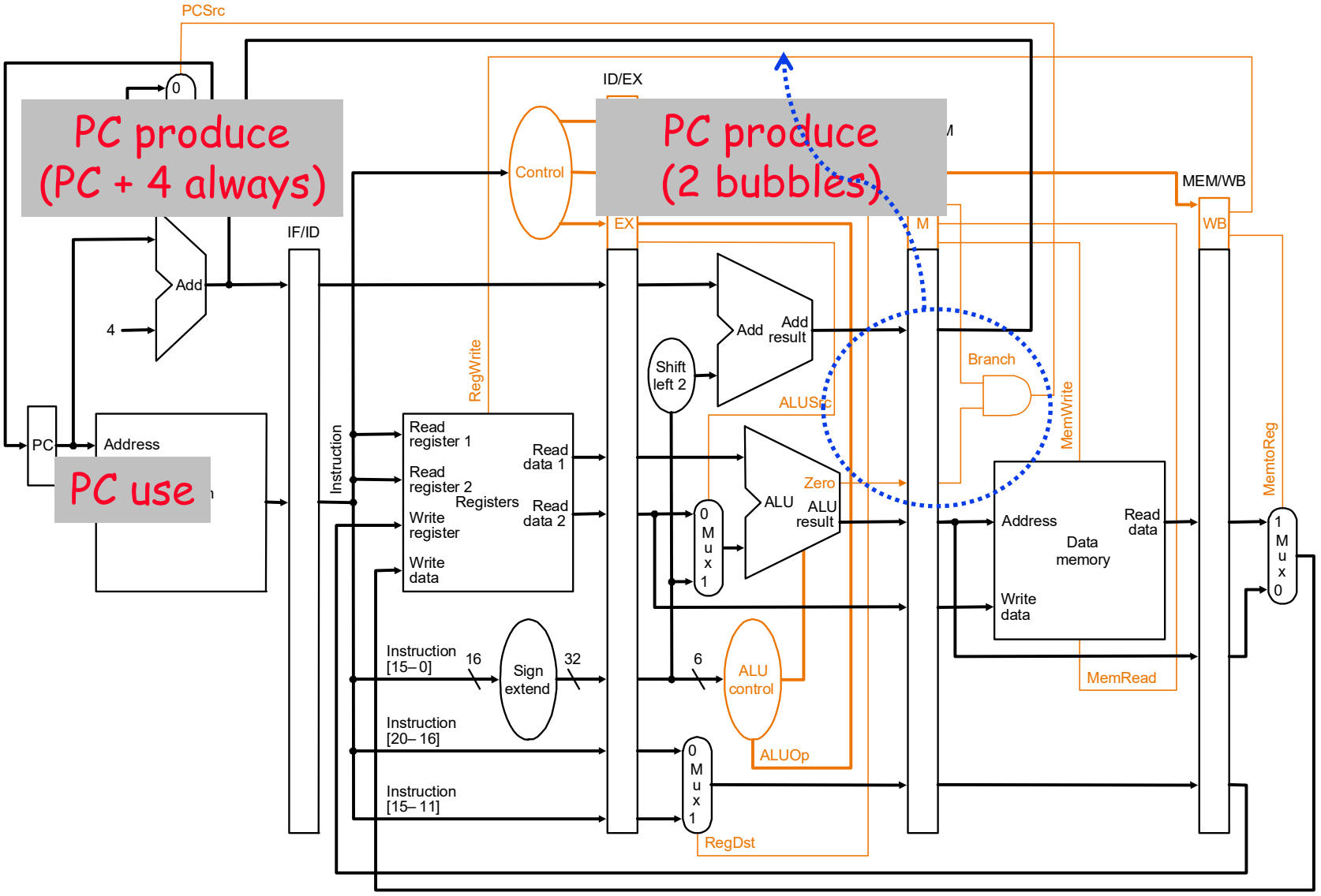
Penalty for a wrong guess

Can we reduce either of the two penalty terms?

Reducing Mispredict Penalty



Reducing Mispredict Penalty



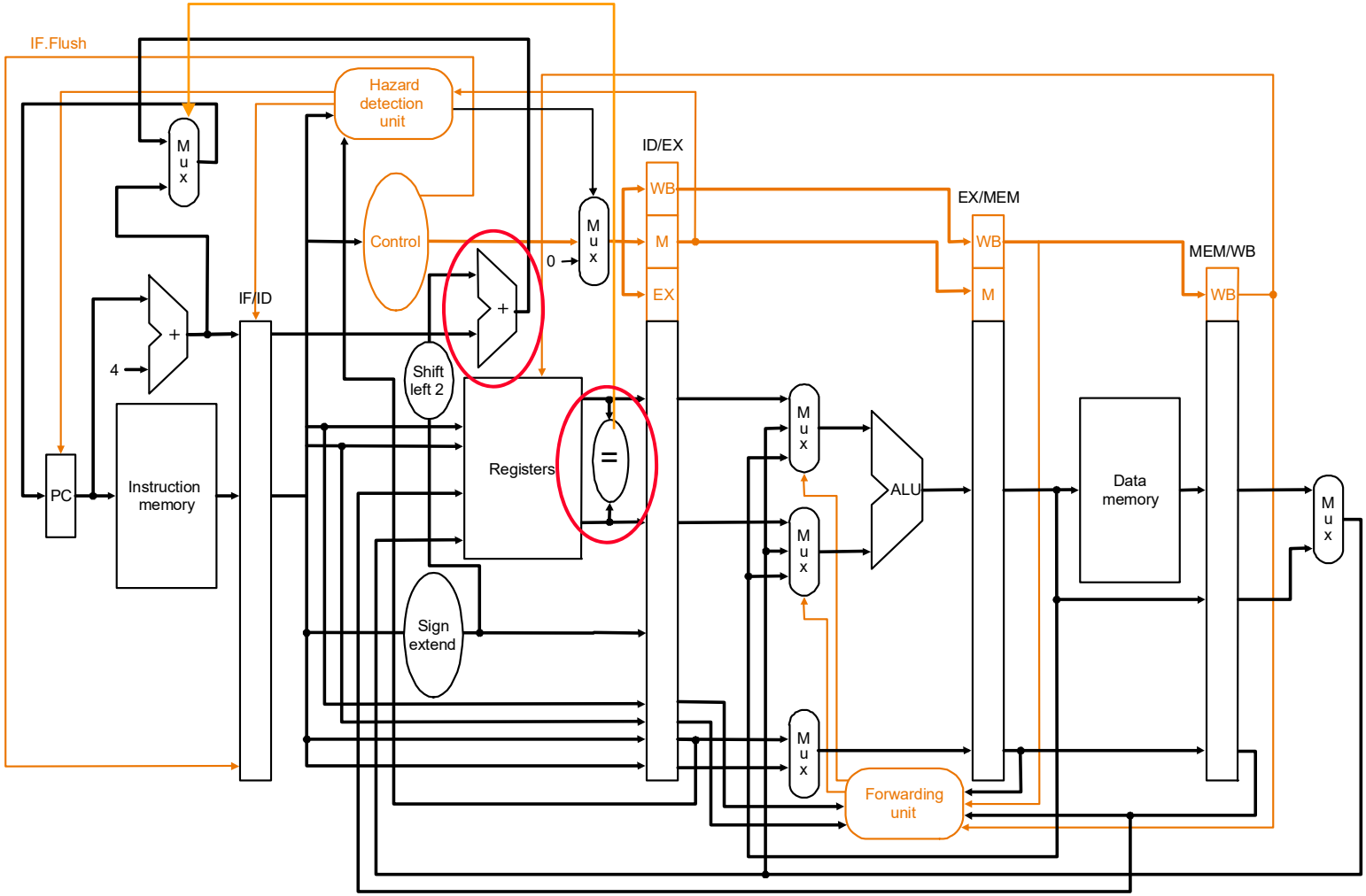
IPC = 1 / [1 + (0.2*0.7) * 2] = 0.78

MIPS R2000 Control Flow Design (for an early branch resolution)

- Simple address calculation based on instruction only
  - Branch PC-offset: 16-bit full-addition + 14-bit half-addition
  - Jump PC-offset: concatenation only
- Simple branch condition based on RF
  - One register relative (>, <, =) to 0
  - Equality between 2 registers

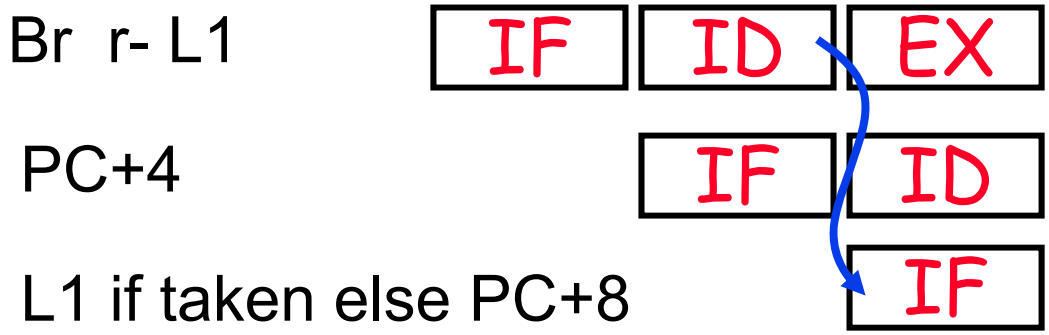
  No addition/subtraction necessary!
- An explicit ISA design choice to enable **branch resolution in ID** of a 5-stage pipeline!

MIPS R2000: Branch Resolved in ID



IPC = 1 / [1 + (0.2*0.7) * 1] = 0.88

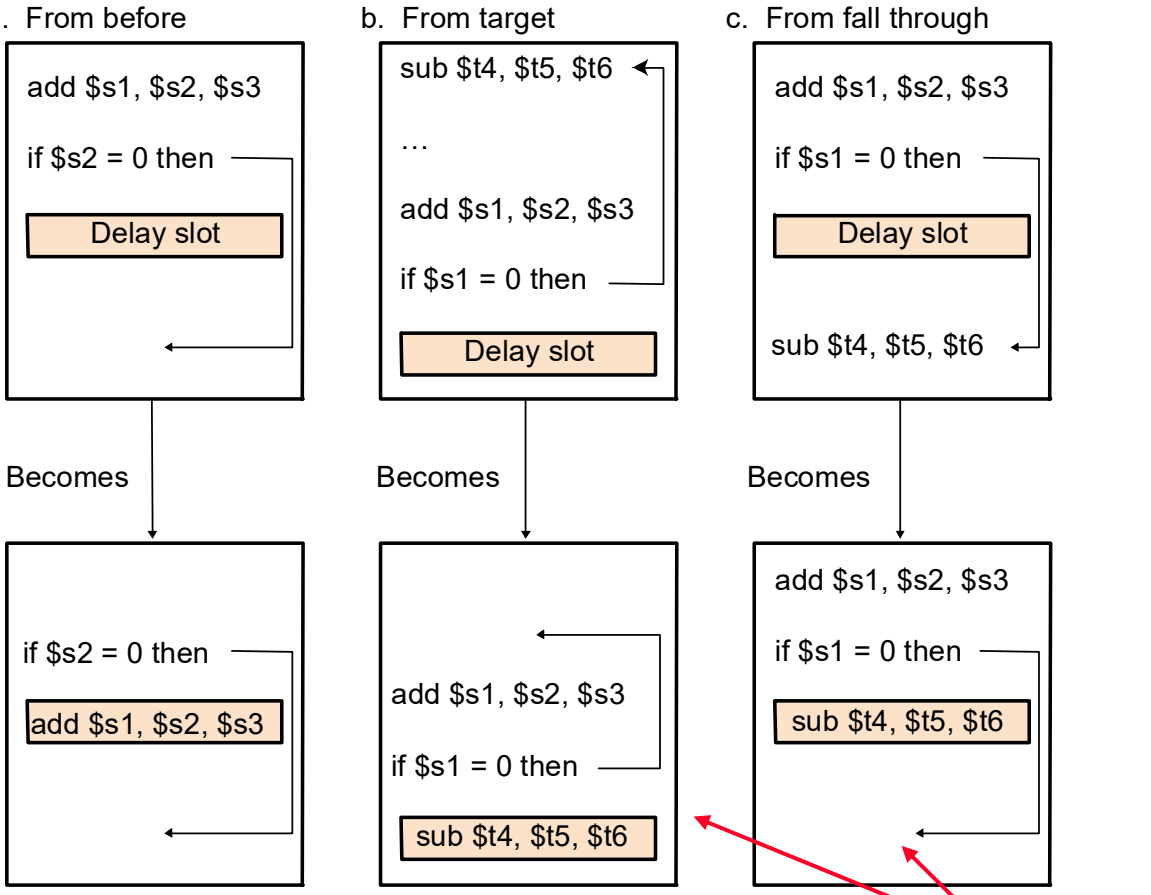
Branch Delay Slots in MIPS R2000



- PC+4 is already in the pipeline
  - Throwing PC+4 away costs 1 bubble
  - Letting PC+4 finish to the end won't hurt performance
- R2000 defined branches to have an architectural latency of 1 instruction
  - 1st instruction immediately after a branch is always executed
  - Branch target takes effect on the 2nd instruction
  - If we can find an instruction to fill the delay slot, we get ideal IPC of 1 without branch prediction or a pipeline flush logic
  - ~80% of delay slots can be filled automatically by compilers

Filling Delay Slots by Static Reordering Transformation

Reordering data-independent instructions does not change the program's correctness (no RAW, WAR, WAW)



Within the same basic block

Safe??

[Based on figures from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Final Data Hazard Analysis (with Forwarding)

	R/I-Type	LW	SW	Br	J	Jr
IF						
ID				use		use
EX	use produce	use	use	use		
MEM		produce				
WB						

- With forwarding, hazard distance is 0 except for RAW dependence on LW where it is 1
- Load delay slot semantics ensures a dependent instruction to be at least distance 2

Final PC Hazard Analysis

	R/I-Type	LW	SW	Br	J	Jr
IF	use (produce)	use (produce)	use (produce)	use	use	use
ID				produce	produce	produce
EX						
MEM						
WB						

If MIPS R2K

- Hazard distance on a taken branch is 1
- Again, branch delay slot semantics ensures a dependent instruction to be at least distance 2

Hazard distance is greater in modern CPUs. Why?

Question?

Announcements: *Homework #2 will be collected (due: 4/12)*
Homework #3 will be posted soon (due: 4/19)
1st mid-term on 4/20

Reading: *Finish reading P&H Ch.4*

Handouts: *None*