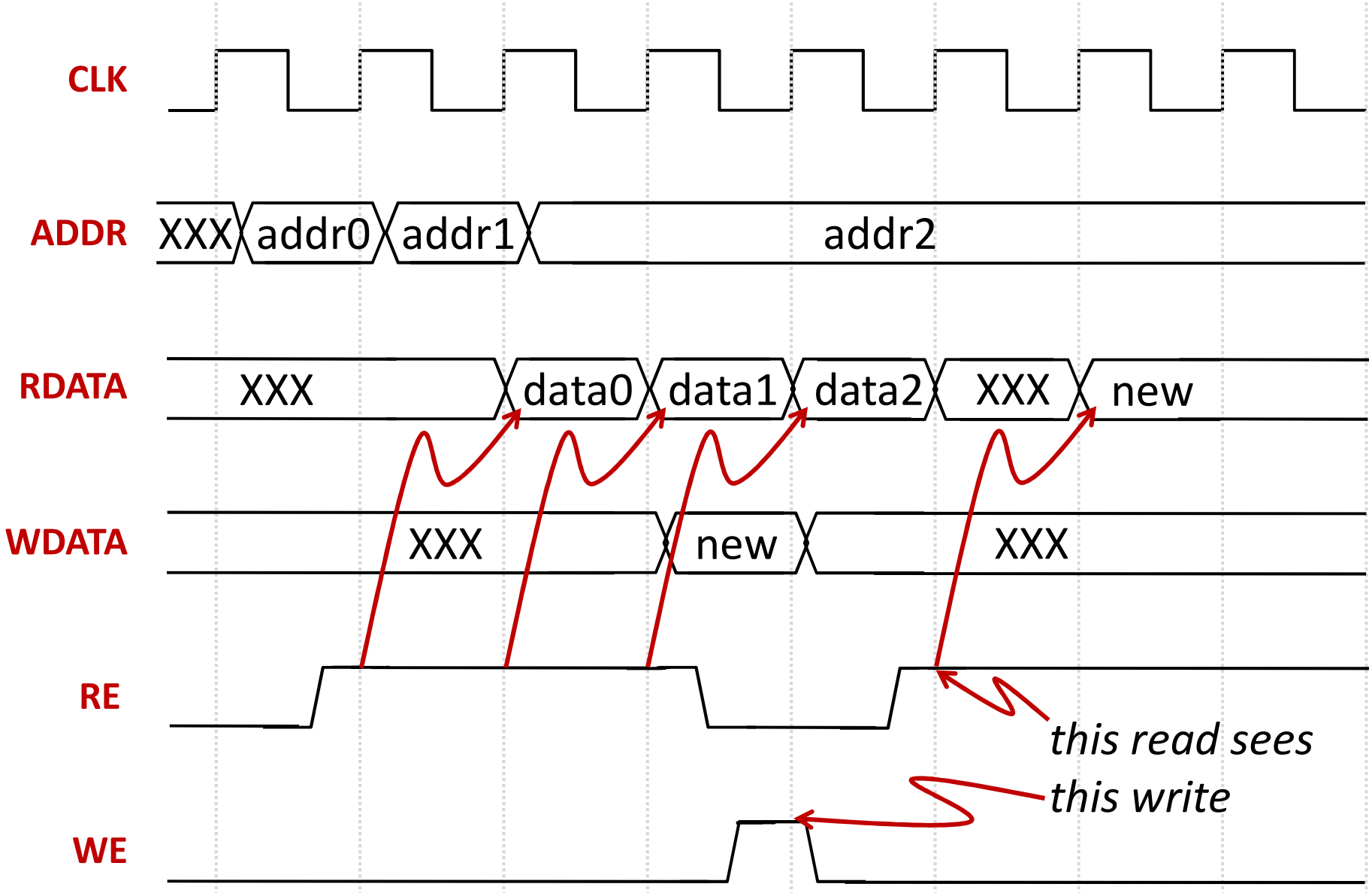
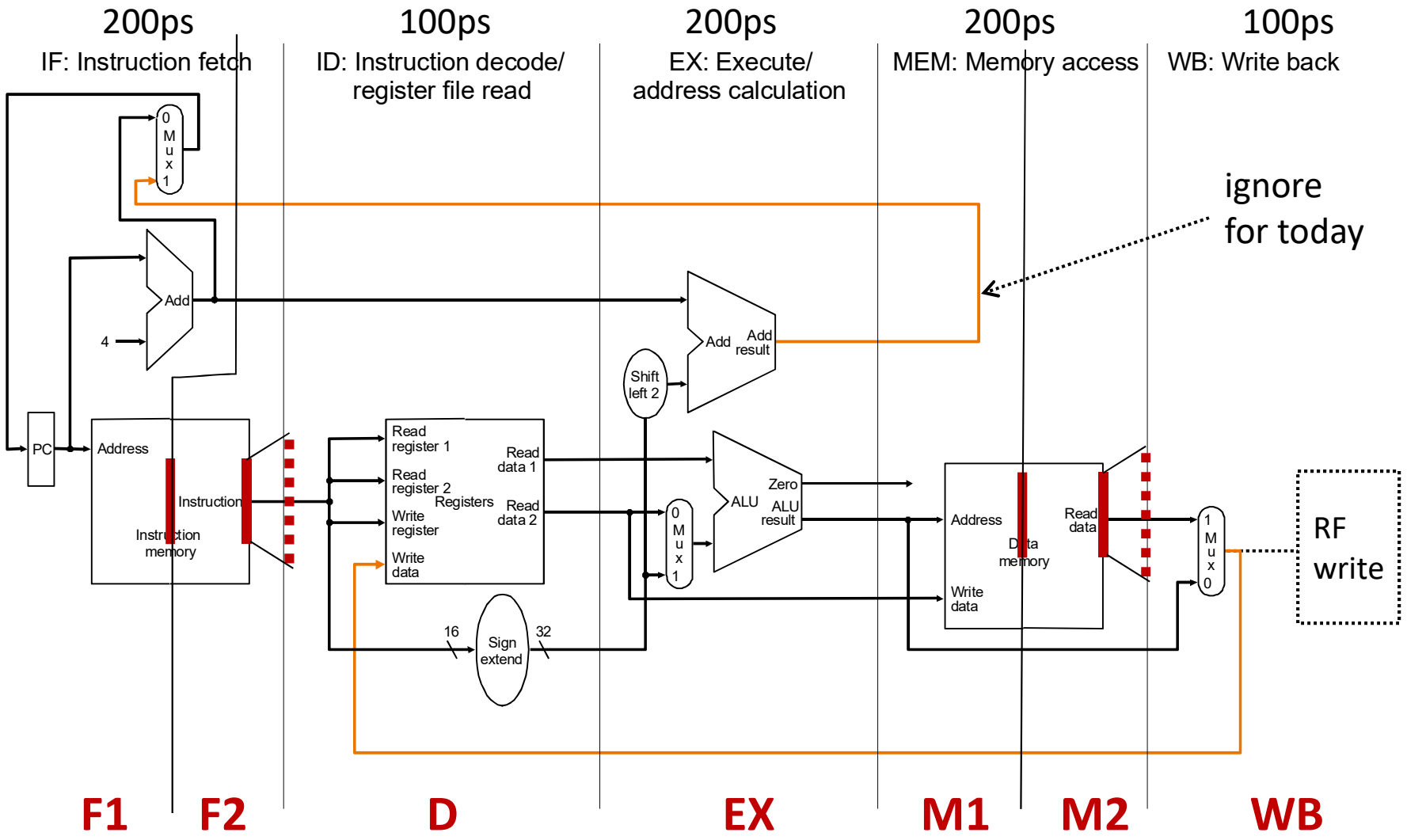
18-447 Lecture 9: Control Hazard and Resolution

James C. Hoe
Department of ECE
Carnegie Mellon University

2-Cycle Synchronous Mem Read/Write



2024 Lab 2 with 7 stages



Housekeeping

- Your goal today
  - "simple" control flow resolution in in-order pipelines
  - there is more fun to come on this
- Notices
  - HW 2, due Mon 2/19
  - Lab 2, status check wk6, due wk7 (Handout #7)

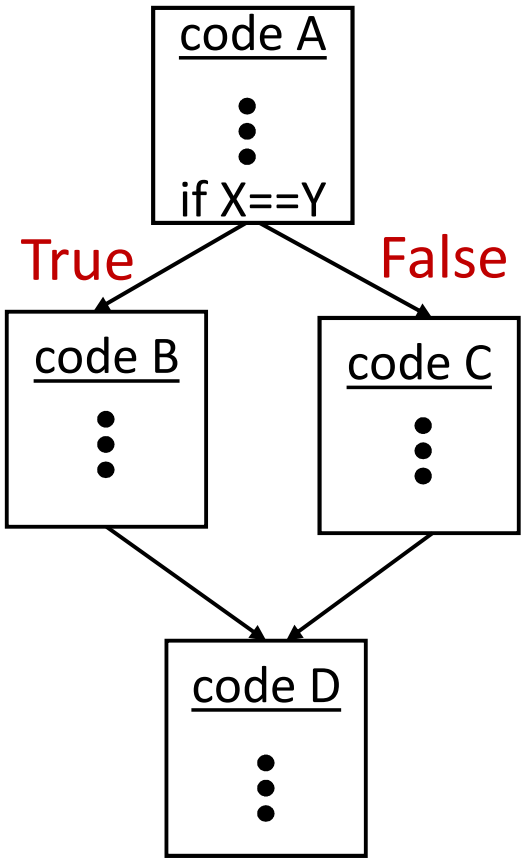
    Look (with your brain!!!) at the counter values
- Readings
  - P&H Ch 4

Control Dependence

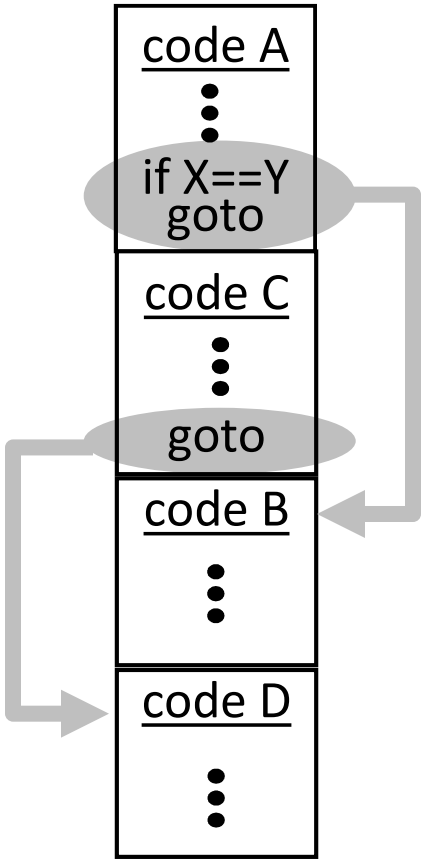
- C-Code

```
{ code A }
if X==Y then
    { code B }
else
    { code C }
{ code D }
```

Control Flow Graph



Assembly Code (linearized)



At ISA-level, control dependence == “data dependence on PC”

Applying Hazard Analysis on PC

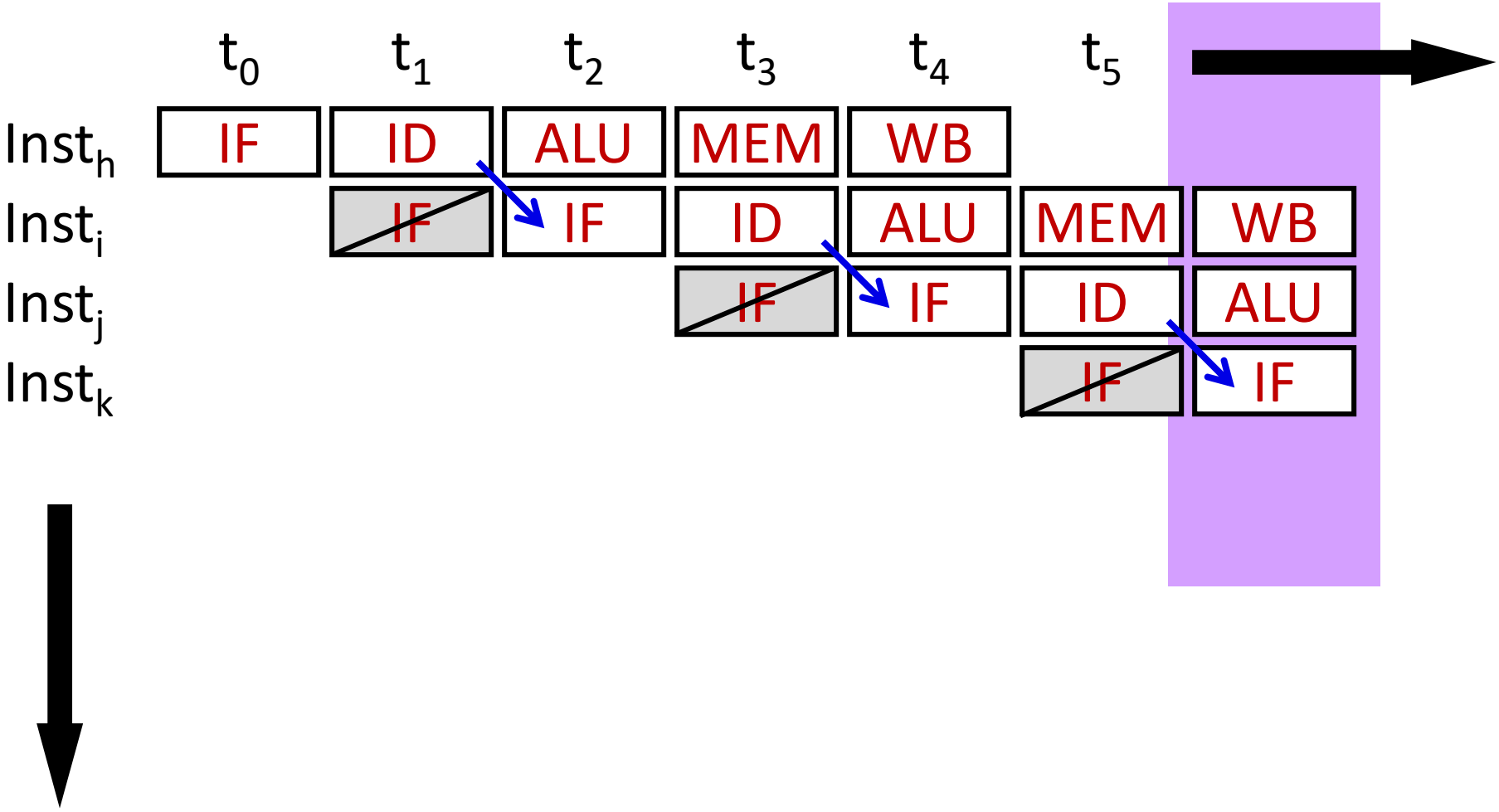
	R/I-Type	LW	SW	Bxx	Jal	Jalr
IF	**use**	**use**	**use**	**use**	**use**	**use**
ID	**produce**	**produce**	**produce**			
EX				**produce**	**produce**	**produce**
MEM						
WB						

- All instructions read and write PC
- PC dependence distance is exactly 1
- PC hazard distance in 5-stage is at least 1

  $\Rightarrow$ Yes, there is RAW hazard

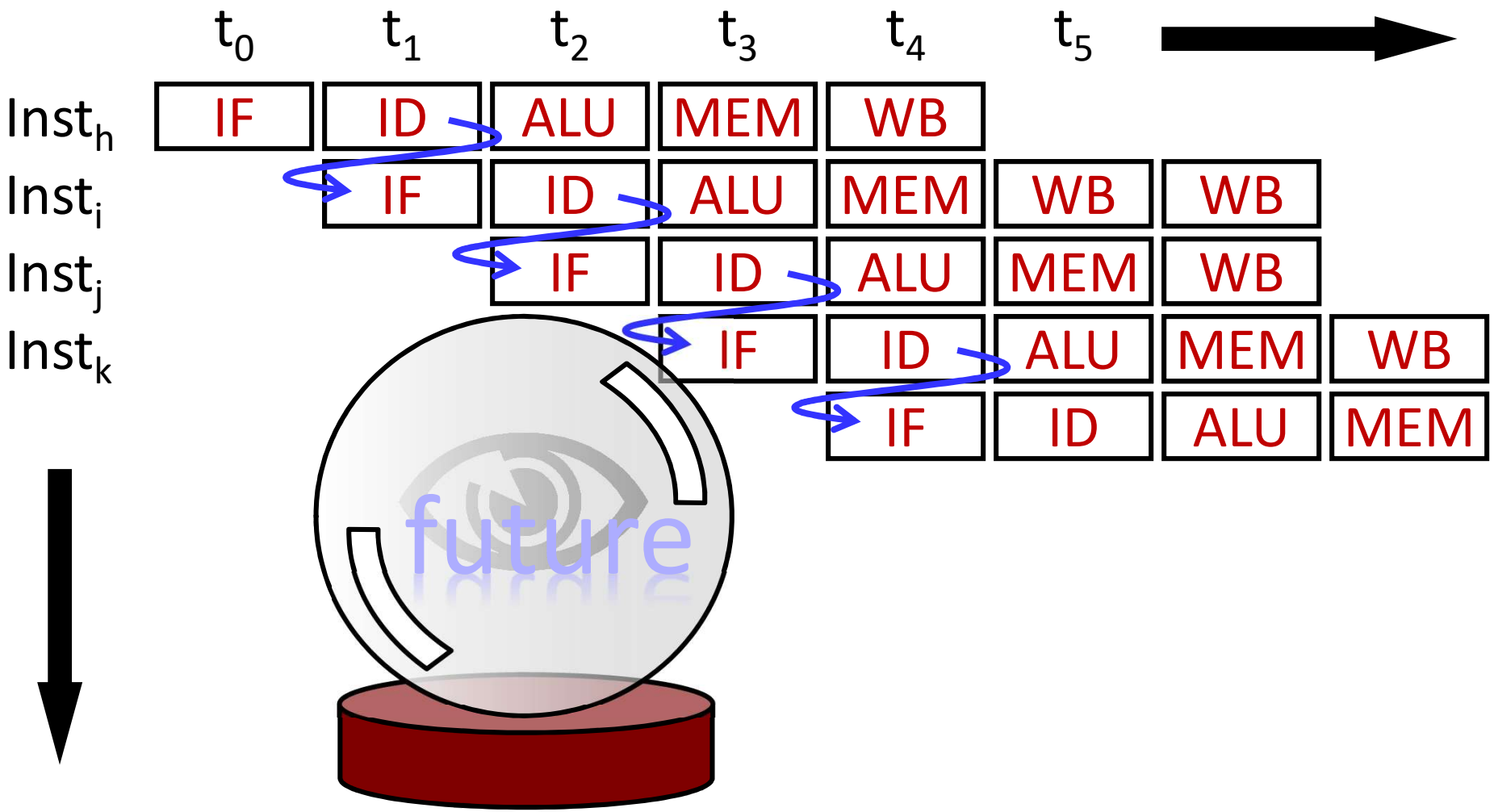
  $\Rightarrow$ Can't eliminate by forwarding; so must stall

Resolve Control Hazard by Stalling

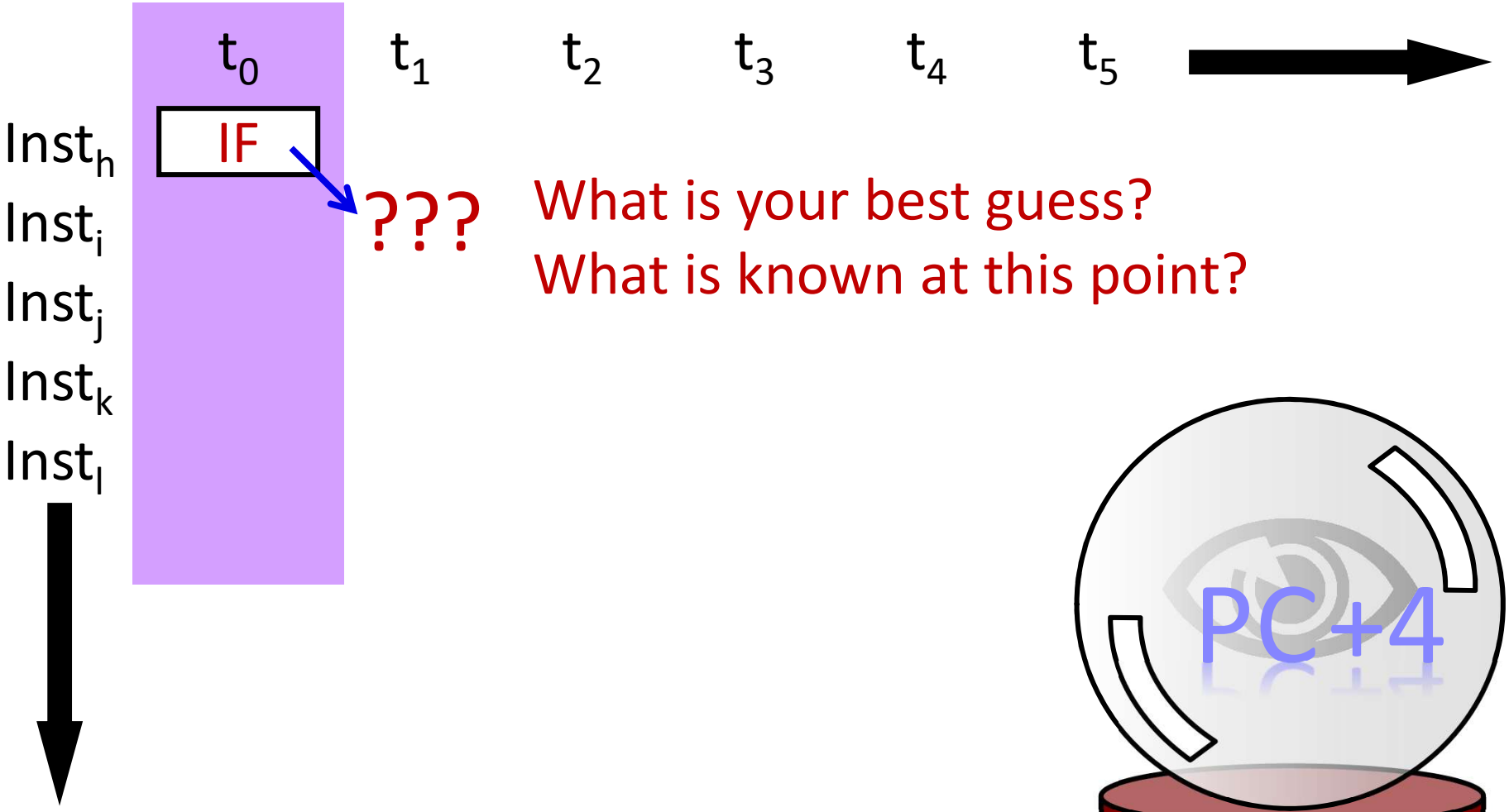


Note: this is if decoding to non-control-flow; BR resolves in EX

Only 1 way to beat "true" dependence



Resolve Control Hazard by Guessing



Control Speculation for Dummies

- Guess nextPC = PC+4 to keep fetching every cycle

  Is this a good guess?

- ~20% of the instruction mix is control flow
  - ~50 % of "forward" control flow taken (if-then-else)
  - ~90% of "backward" control flow taken (end-of-loop)

  Over all, typically ~70% taken and ~30% not taken

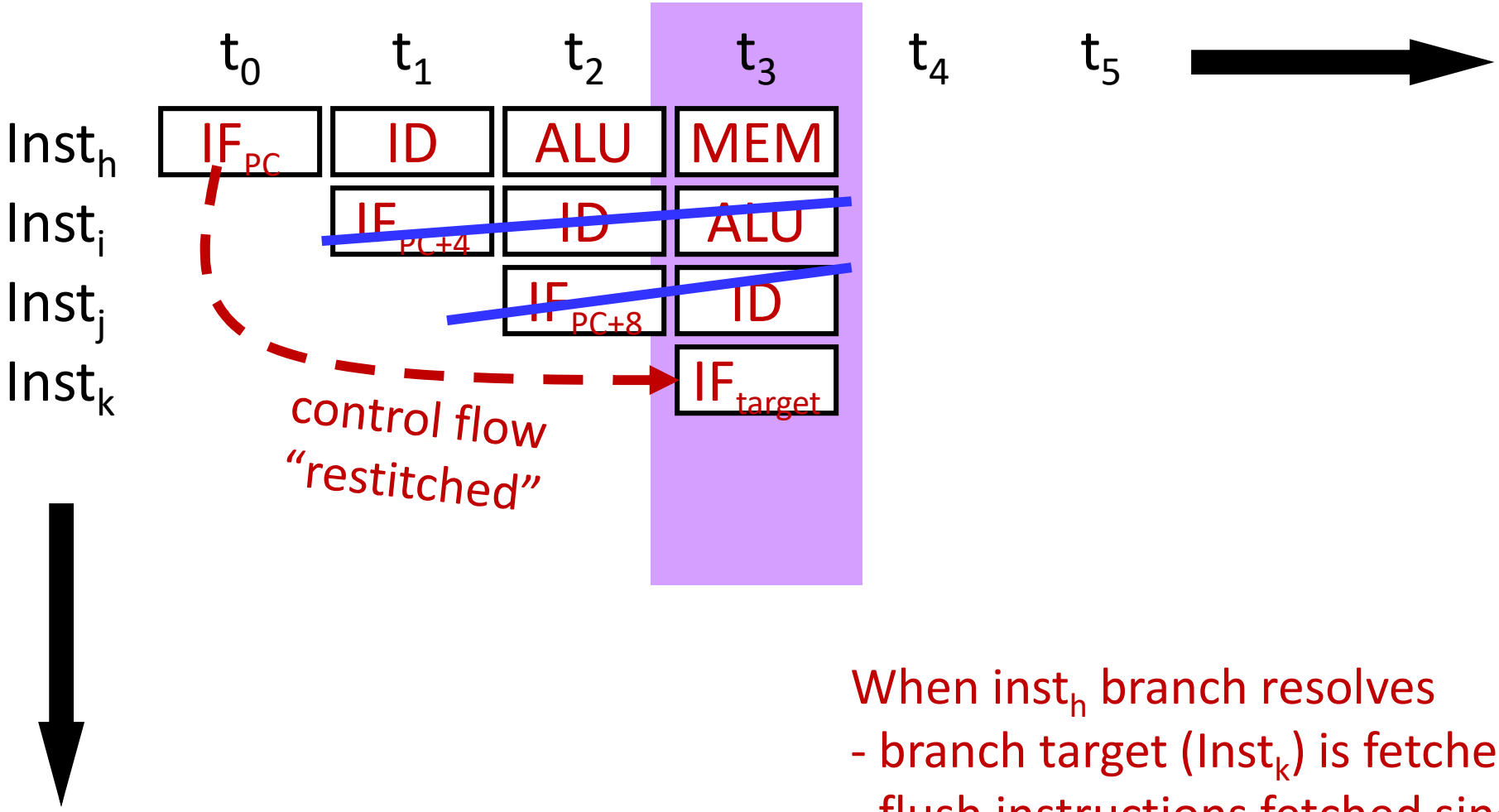
  [Lee and Smith, 1984]

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

  What do you do when wrong?

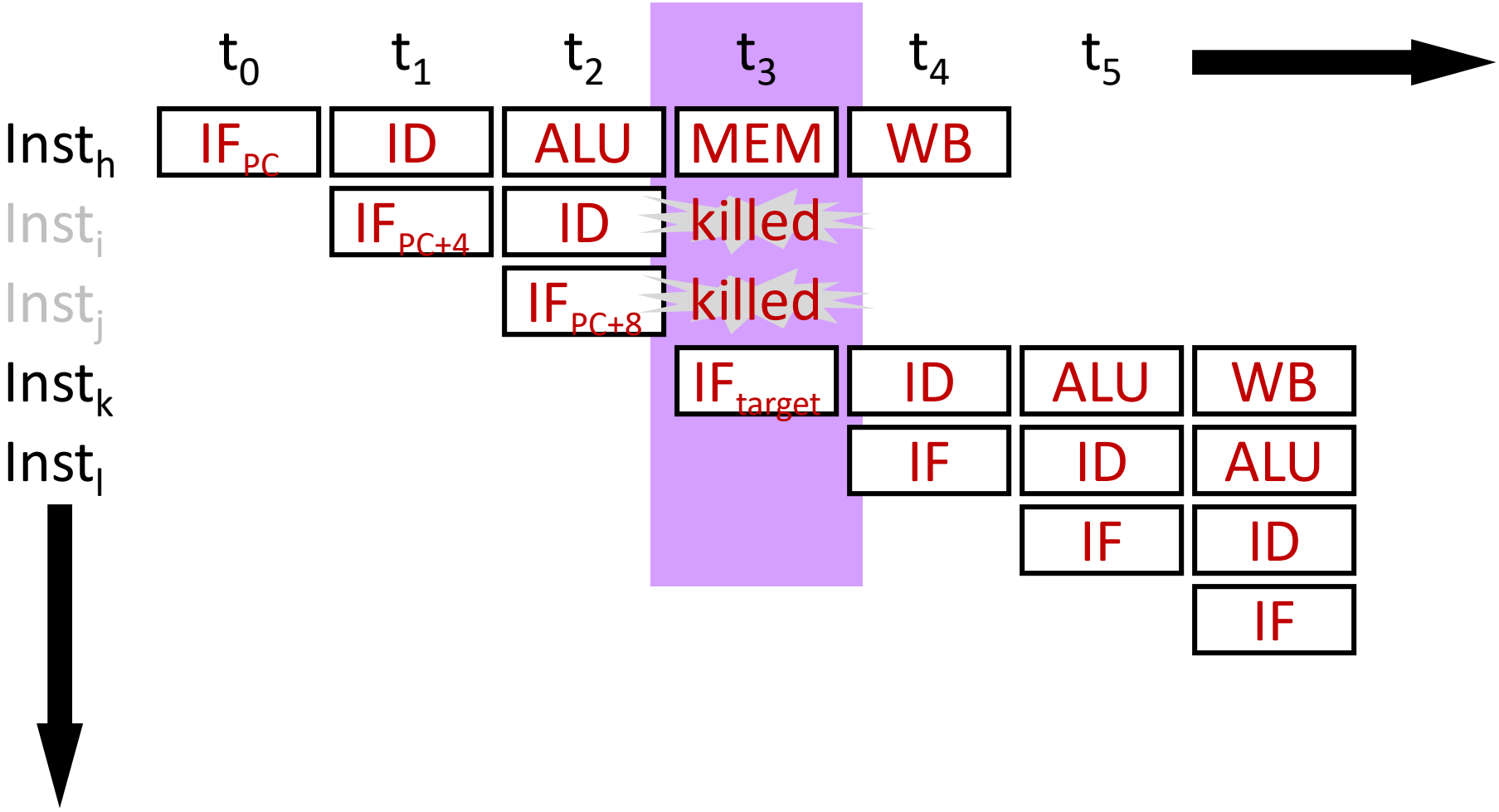
  What do you lose when wrong?

Control Speculation: PC+4



When $inst_h$ branch resolves
- branch target ($Inst_k$) is fetched
- flush instructions fetched since $inst_h$ ("wrong-path")

Pipeline Flush on Misprediction



$Inst_h$ is a taken branch; $Inst_i$ and $Inst_j$ fetched but not executed

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

Performance Impact

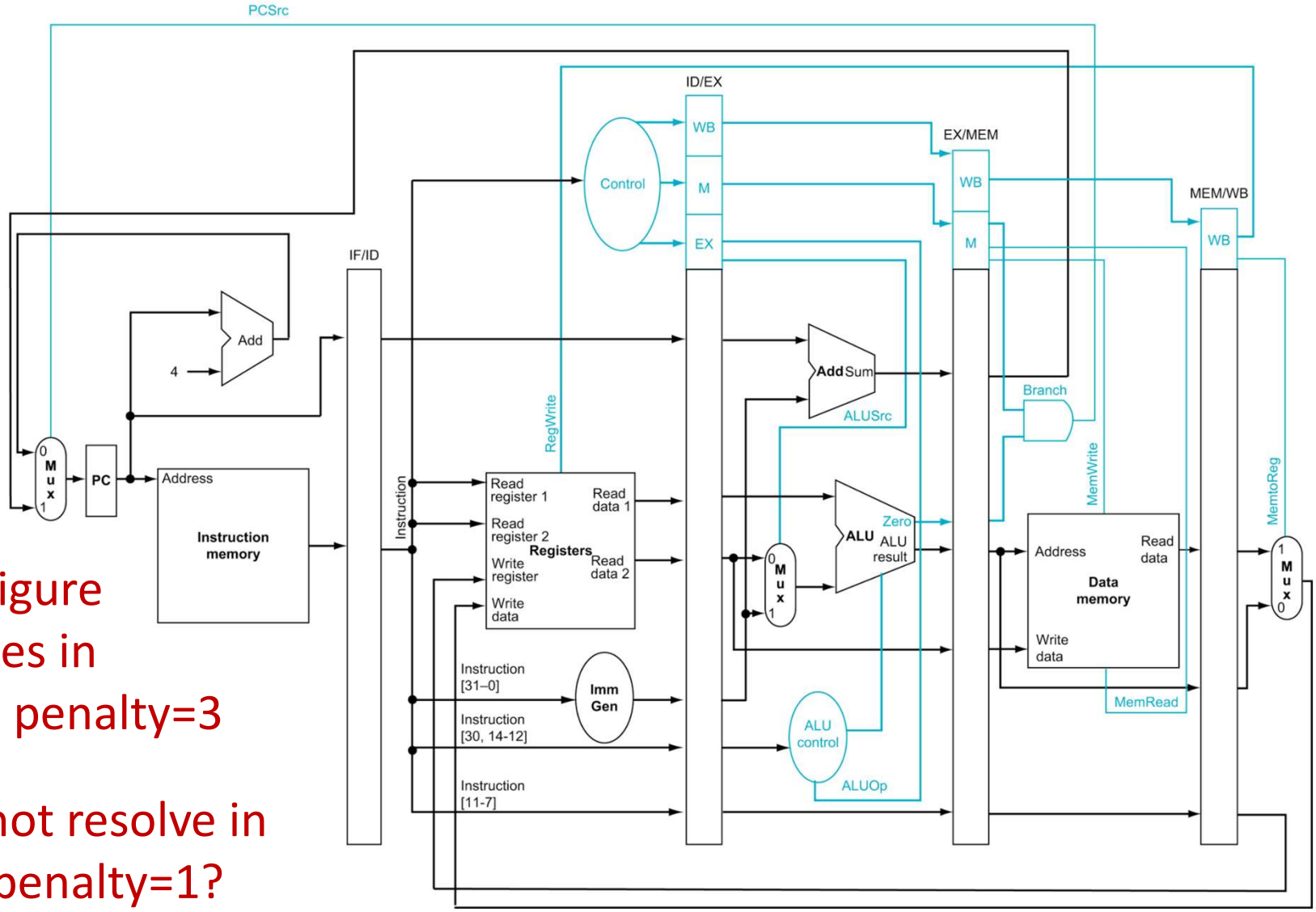
- Correct guess ⇒ no penalty most of the time!!
- Incorrect guess ⇒ 2 bubbles
- Assume
  - no data hazard stalls
  - 20% control flow instructions
  - 70% of control flow instructions are taken
  - IPC = 1 / [1 + (0.20*0.7) * 2] =

    = 1 / [1 + 0.14 * 2] = 1 / 1.28 = 0.78

misprediction rate *misprediction penalty*

How to reduce the two penalty terms?

Reducing Mispredict Penalty



P&H figure resolves in MEM, penalty=3

Why not resolve in ID so penalty=1?

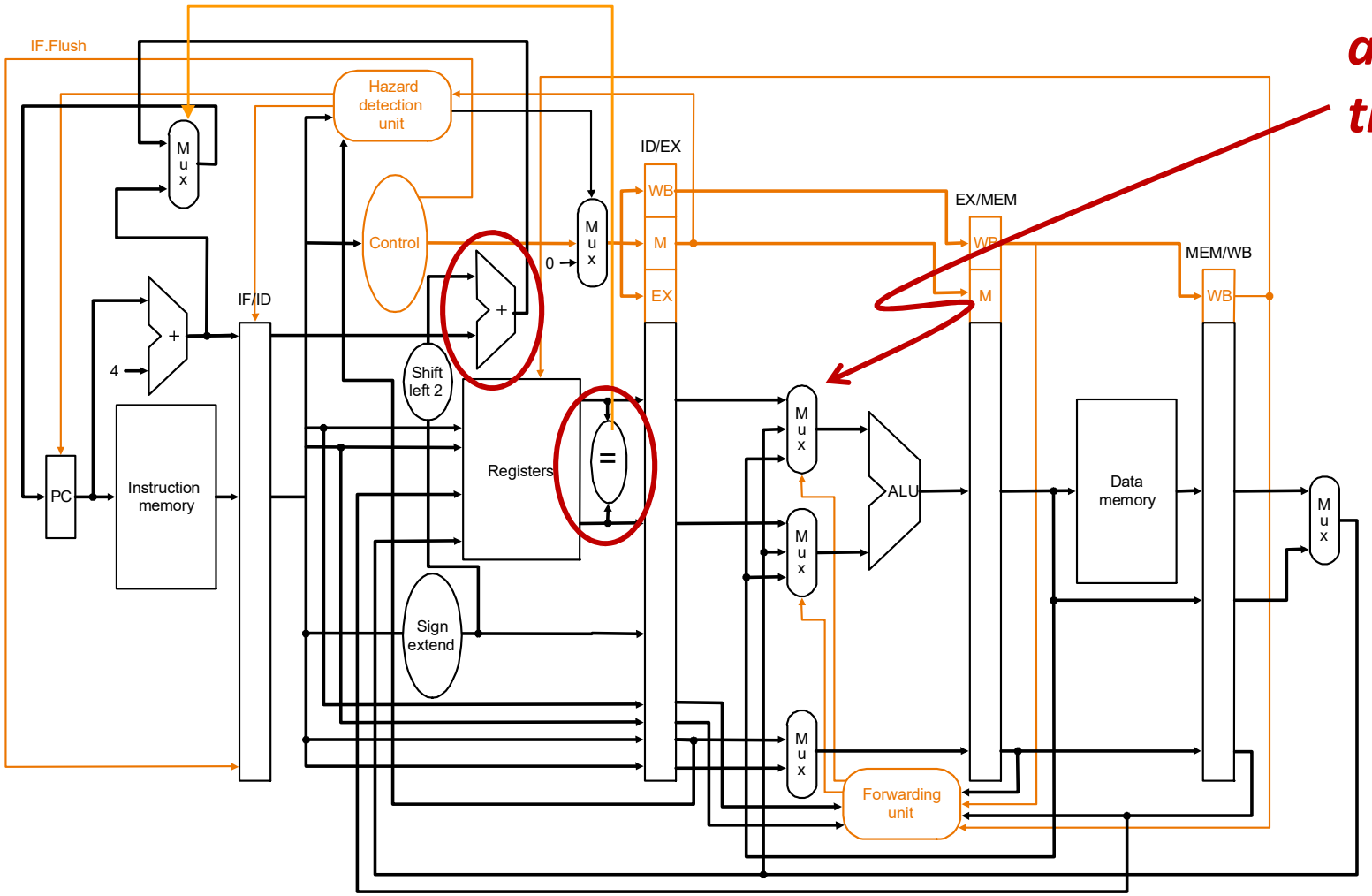
MIPS R2000 ISA Control Flow Design

- Simple address calculation based on IR 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!

Explicit ISA design choices to make possible branch resolution in ID of a 5-stage pipeline

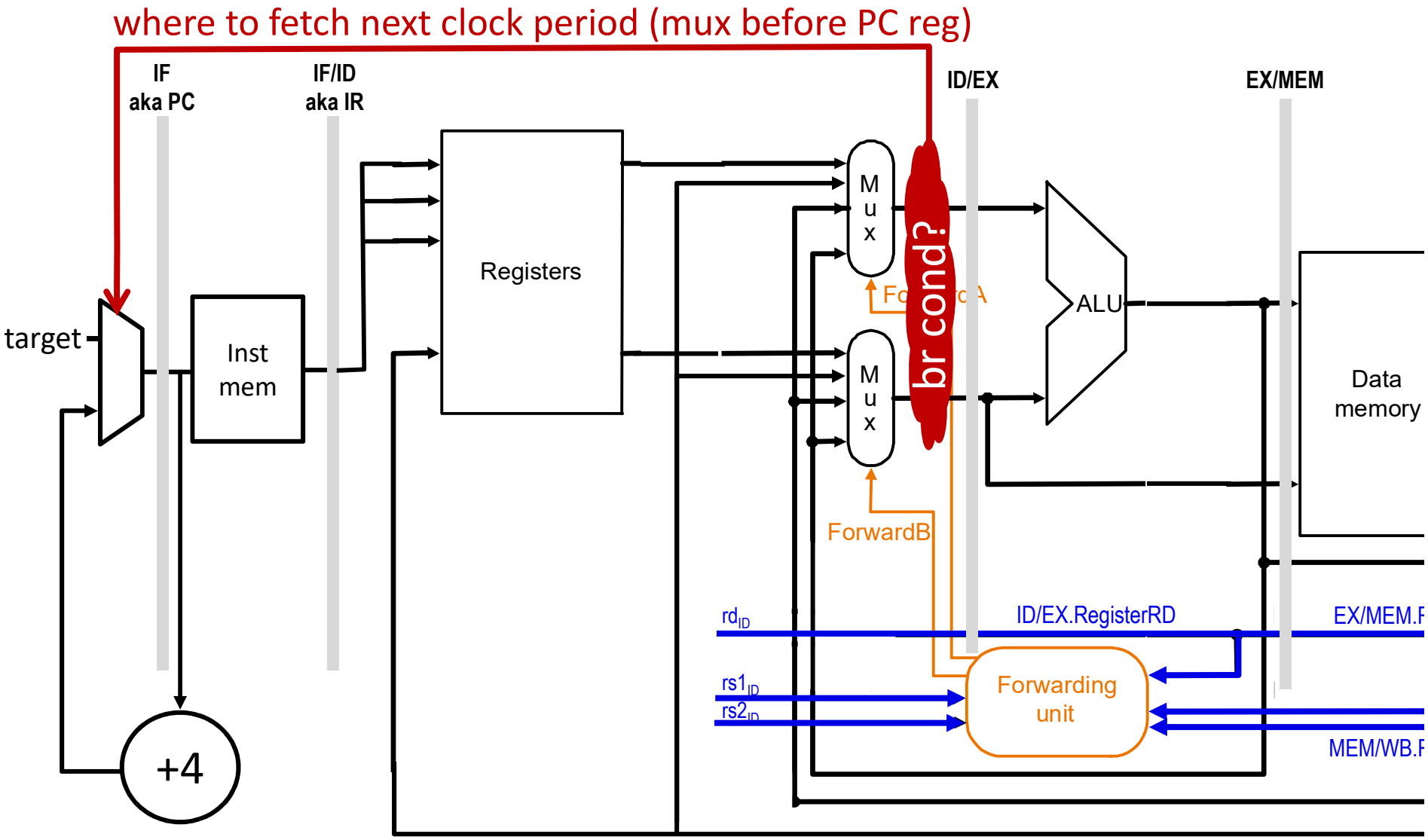
Branch Resolved in ID



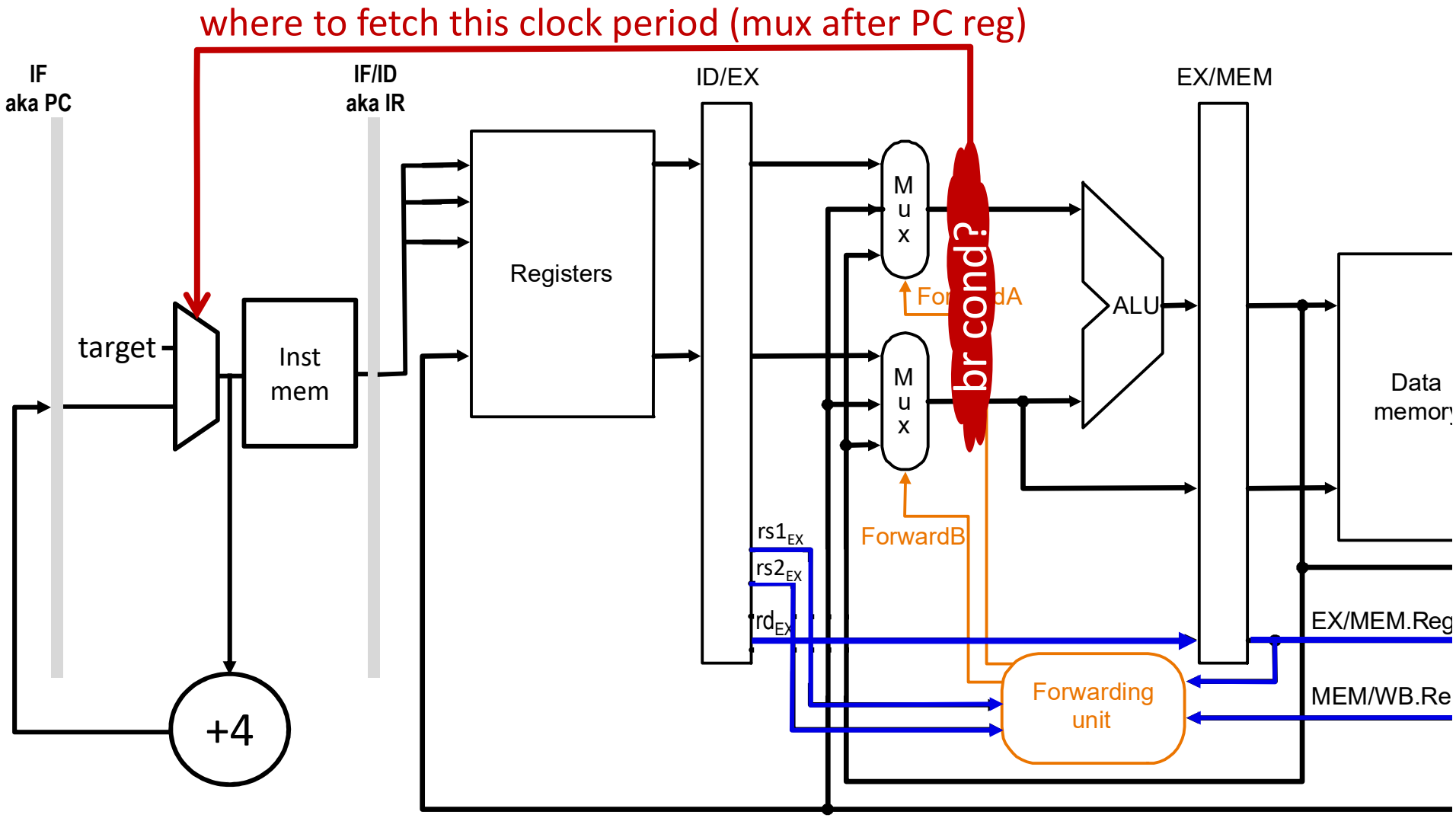
[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

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

Forwarding (v1): extend critical path

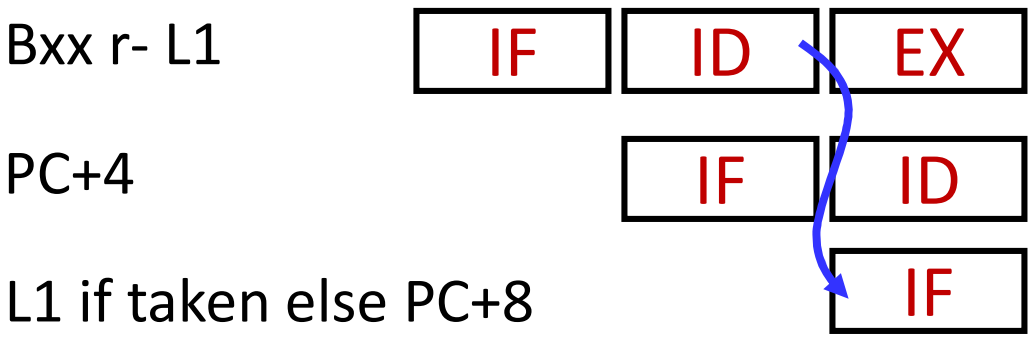


Forwarding (v2): retiming hack



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

MIPS Branch Delay Slot



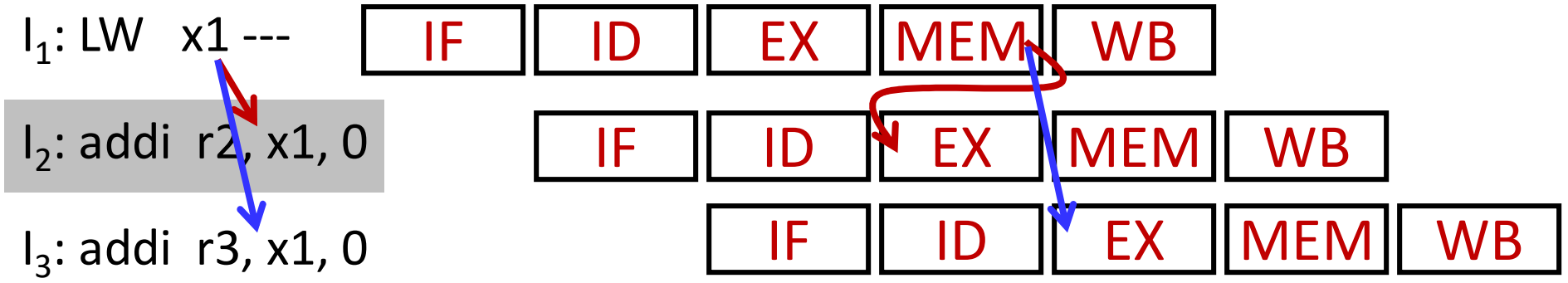
- Throwing PC+4 away cost 1 bubble; letting PC+4 finish won't hurt performance . . . . . .
- R2000 jump/branch has 1 inst. architectural latency
  - PC+4 after jump/branch always executed

    no need for pipeline flush logic
  - if delay slot always do useful work, effective IPC=1
  - ~80% of "delay slots" can be filled by compilers

unfilled nop

IPC = 1 / [1 + (0.2*0.2) * 1] = 0.96

Also MIPS Load "Delay Slot"



- R2000 defined LW with arch. latency of 1 inst
  - invalid for I_2 (in LW's delay slot) to ask for LW's result
  - any dependence on LW at least distance 2
- Delay slot vs dynamic stalling
  - fill with an independent instruction (no difference)
  - if not, fill with a NOP (no difference)
- **MIPS**=**M**icroproc. without **I**nterlocked **P**ipeline **S**tages

Delay slots good idea? ***non-atomic, μarch specific***

Performance Impact

- Correct guess ⇒ no penalty most of the time!!
- Incorrect guess ⇒ 2 bubbles; 1 if resolve in ID
- Assume
  - no data hazard stalls
  - 20% control flow instructions
  - 70% of control flow instructions are taken
  - IPC = 1 / [1 + (0.20*0.7) * 2] =

    = 1 / [1 + 0.14 * 2] = 1 / 1.28 = 0.78

how about?

misprediction rate

misprediction penalty

0.87 resolve in ID;
0.96 MIPS delay slot

Need to do more?

In case you needed motivation

Basic Pentium III Processor Misprediction Pipeline

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium 4 Processor Misprediction Pipeline

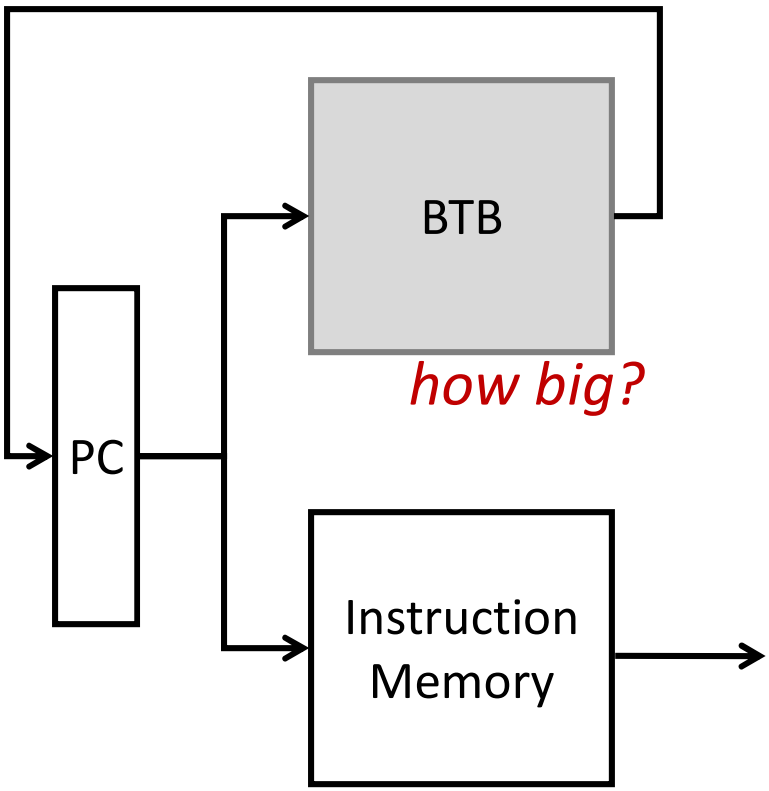
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP		TC Fetch		Drive	Alloc	Rename		Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive

[The Microarchitecture of the Pentium 4 Processor, Intel Technology Journal, 2001]

Can we make better guesses?
(for when it is not MIPS or 5-stage)

- For control-flow instructions
  - why not always guess taken since 70% correct
  - need to know taken target to be helpful
- For non-control-flow instructions
  - can't do better than guessing nextPC=PC+4
  - still tricky since must guess before knowing it is control-flow or non-control-flow
- Guess nextPC from current PC alone, and fast!
- Fortunately
  - **instruction at same PC doesn't change**
  - **PC-offset target doesn't changes**
  - **okay to be wrong some of the time**

Branch Target Buffer (magic version)



- BTB
  - a giant table indexed by PC
  - returns the "guess" for nextPC
- When seeing a PC first time, after decoding, record in BTB . . .
  - PC + 4 if ALU/LD/ST
  - PC+offset if Branch or Jump
  - ?? if Jump Indirect
- Effectively guessing branches are always taken (and where to)

IPC = 1 / [1 + (0.20*0.3) * 2]

= 0.89

If not taken

Locality Principle to the Rescue

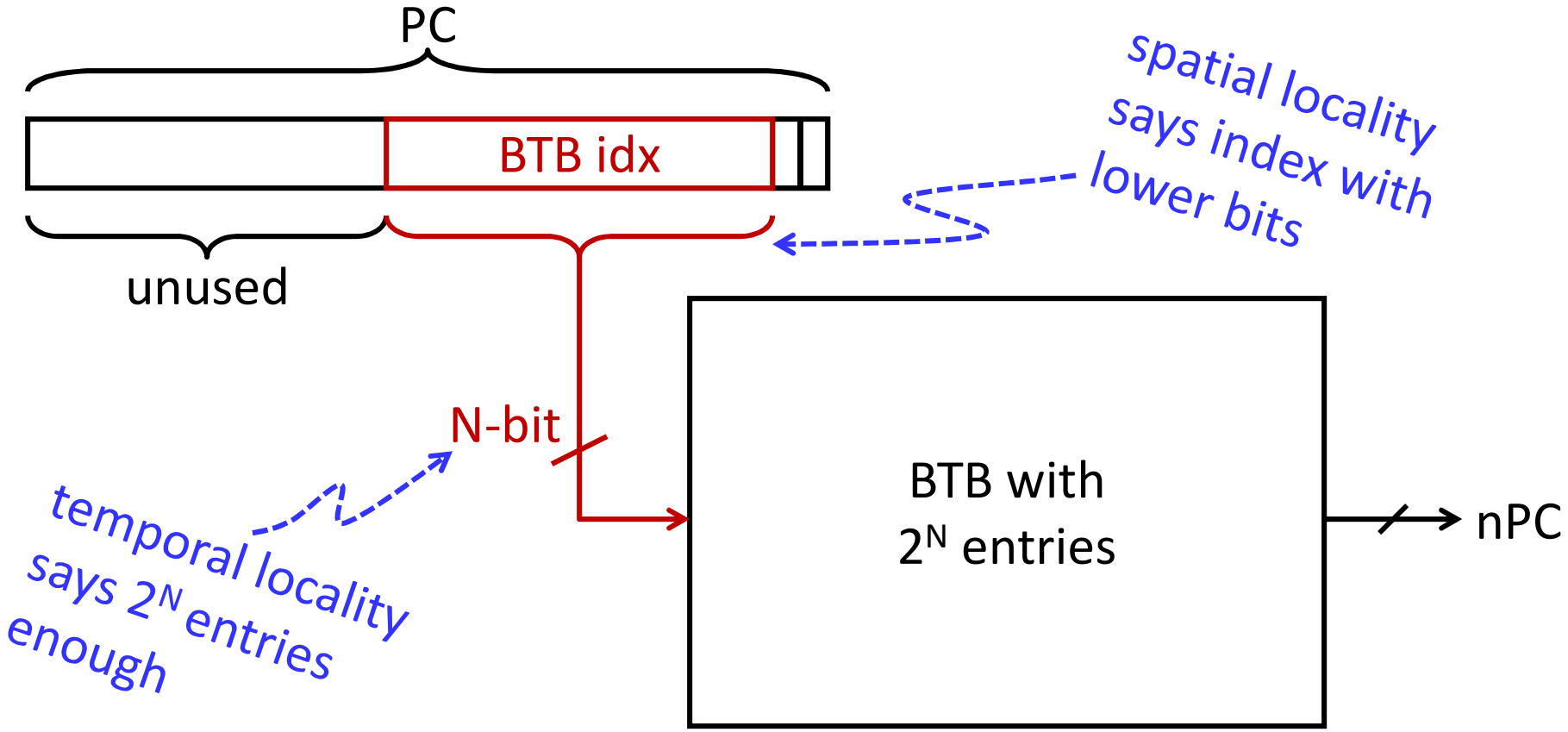
- **Temporal:** after accessing **A**, how many other distinct addresses before accessing **A** again?
- **Spatial:** after accessing **A**, how many other distinct addresses before accessing **B**?
- "Typical" programs have strong locality in memory references—instruction and data

  we put them there ... BB, loops, arrays, structs ...
- **Corollary:** a program with strong temporal and spatial locality access only a compact "**working set**" at any point in time

fewer is more "local"

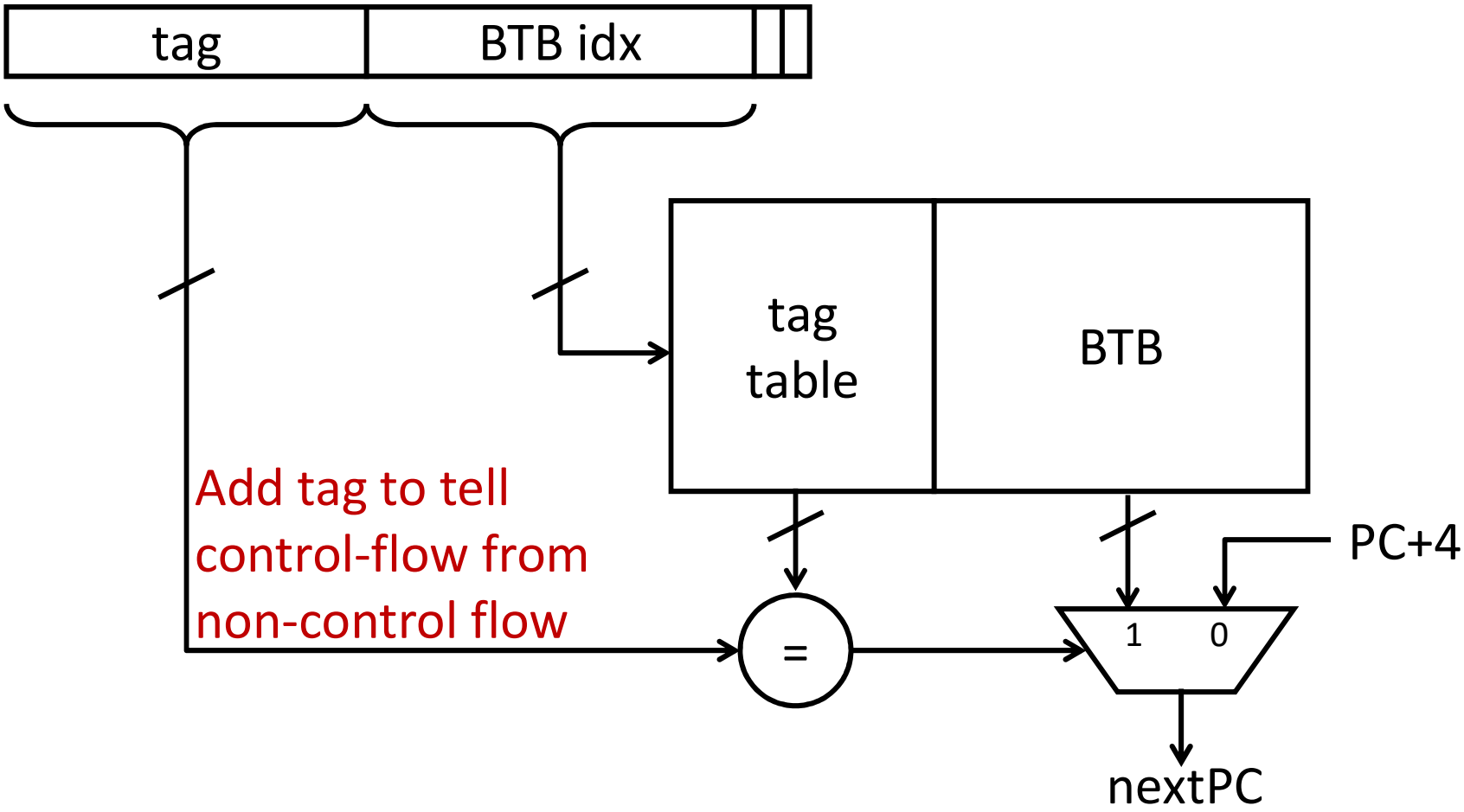
⇒ **just need BTB big enough for *hot* instructions**

Smaller BTB by Hashing



- "Hash" PC into a 2^N entry table
- What happens when two "hot" instructions collide? *No problem, as long as infrequent*

Even Smaller BTB after Tagging



Only hold control-flow instructions (save 80% storage)

Update tag and BTB for new branch after collision

Final 5-stage RISC Datapath & Control