CASPER

CS230: Digital Logic Design and Computer Architecture

Lecture 14: Mitigating Control Hazards

https://www.cse.iitb.ac.in/~biswa/courses/CS230/autumn23/main.html

https://www.cse.iitb.ac.in/~biswa/

What and Where? Control Hazard

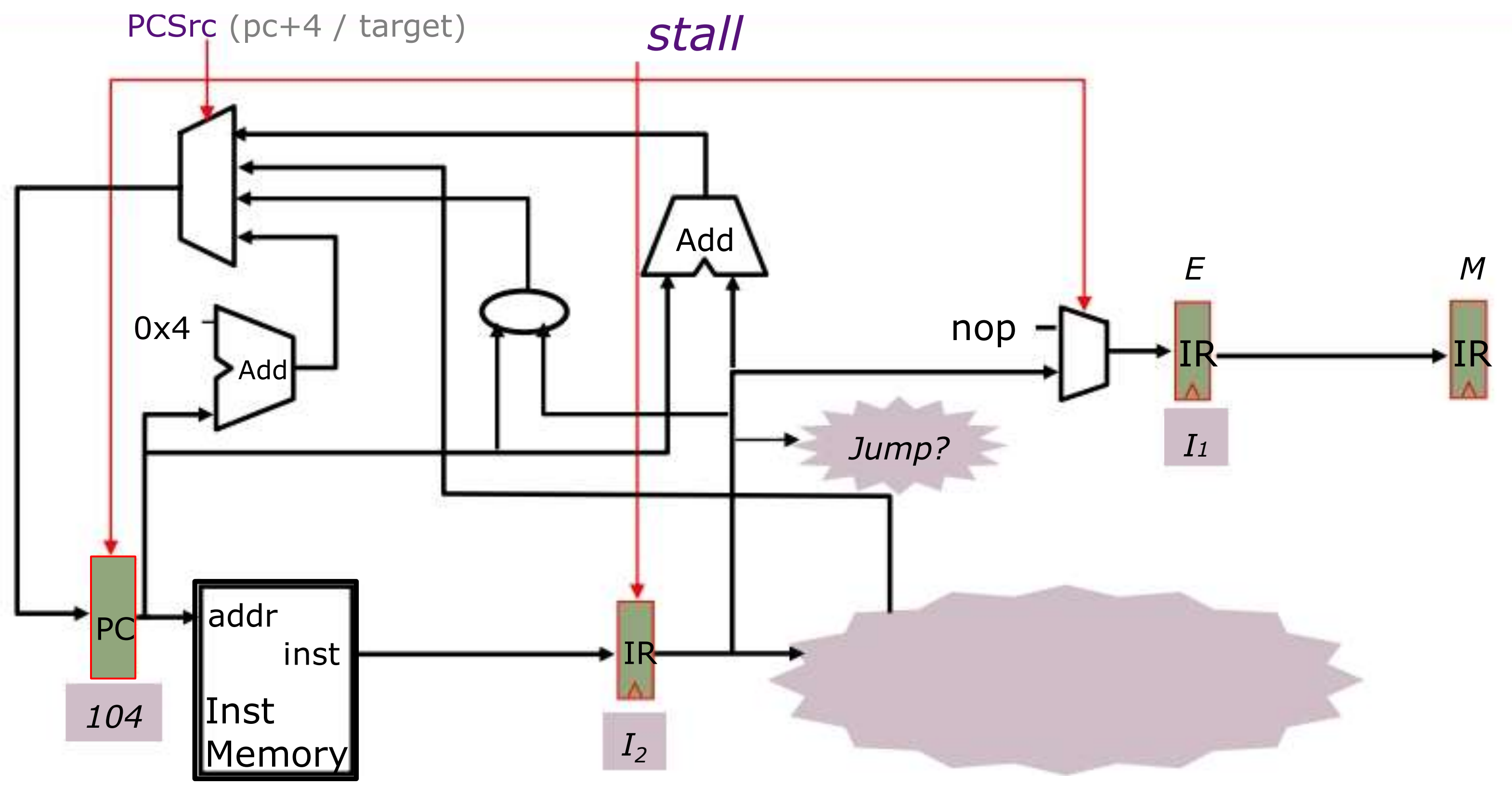
What do we need to calculate next PC?

- For Jumps
  - Opcode, offset, and PC
- For Jump Register
  - Opcode and register value
- For Conditional Branches
  - Opcode, offset, PC, and register (for condition)
- For all others
  - Opcode and PC

In what stage do we know these?

- PC - Fetch
- Opcode, offset - Decode (or Fetch?)
- Register value - Decode
- Branch condition ((rs)==0) - Execute (or Decode?)

Speculate, PC=PC+4



I1	096	ADD
I2	100	J304
I3	~~104~~	~~ADD~~
I4	304	ADD

What happens on mis-speculation, i.e., when next instruction is not PC+4?

kill *How? Insert NOPs*

Conditional branches

```
I1    096    ADD
I2    100    BEQZ r1 200
I3    104    ADD
I4    304    ADD
```

Branch condition is not known until the execute stage

Instructions between a branch instruction and the target are
in the wrong-path if the branch is not taken

Again (stalls/NOPs)

	time t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	WB_1				
(I_2) 100: BEQZ 200		IF_2	ID_2	EX_2	MA_2	WB_2			
(I_3) 104: ADD			IF_3	ID_3	nop	nop	nop		
(I_4) 108:				IF_4	nop	nop	nop	nop	
(I_5) 304: ADD					IF_5	ID_5	EX_5	MA_5	WB_5

Resource Usage

	time t0	t1	t2	t3	t4	t5	t6	t7	. . . .
IF	I_1	I_2	I_3	I_4	I_5				
ID		I_1	I_2	I_3	nop	I_5			
EX			I_1	I_2	nop	nop	I_5		
MA				I_1	I_2	nop	nop	I_5	
WB					I_1	I_2	nop	nop	I_5

Branches: Taken/Not Taken and Target

Instruction	*Taken known?*	*Target known?*
J	After Inst. Decode	After Inst. Decode
BEQZ/BNEZ	After Inst. Execute	After Inst. Execute

what action should be taken in the decode stage?

Can we add an ALU in the decode stage?

What else can be done? Compiler?

Delayed branch: Define branch to take place AFTER a following instruction(used to be in early RISC processors)

branch instruction

sequential successor$_1$

sequential successor$_2$

....

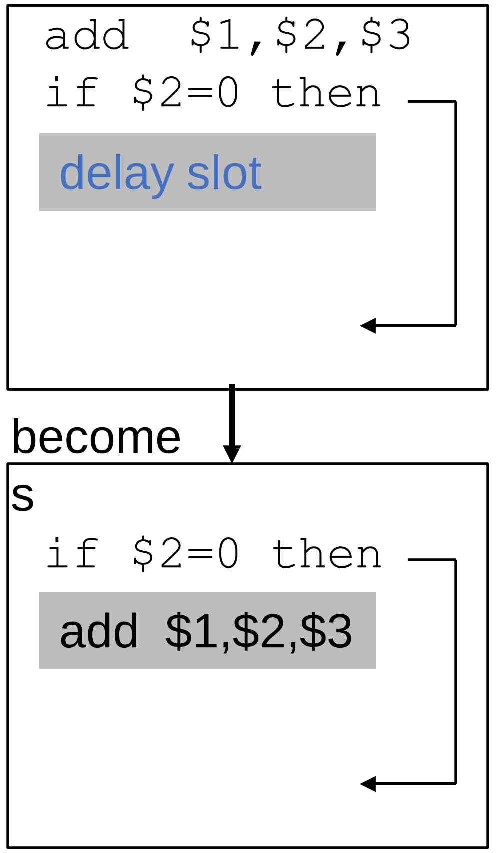
sequential successor$_n$

Branch delay of length *n*

branch target if taken

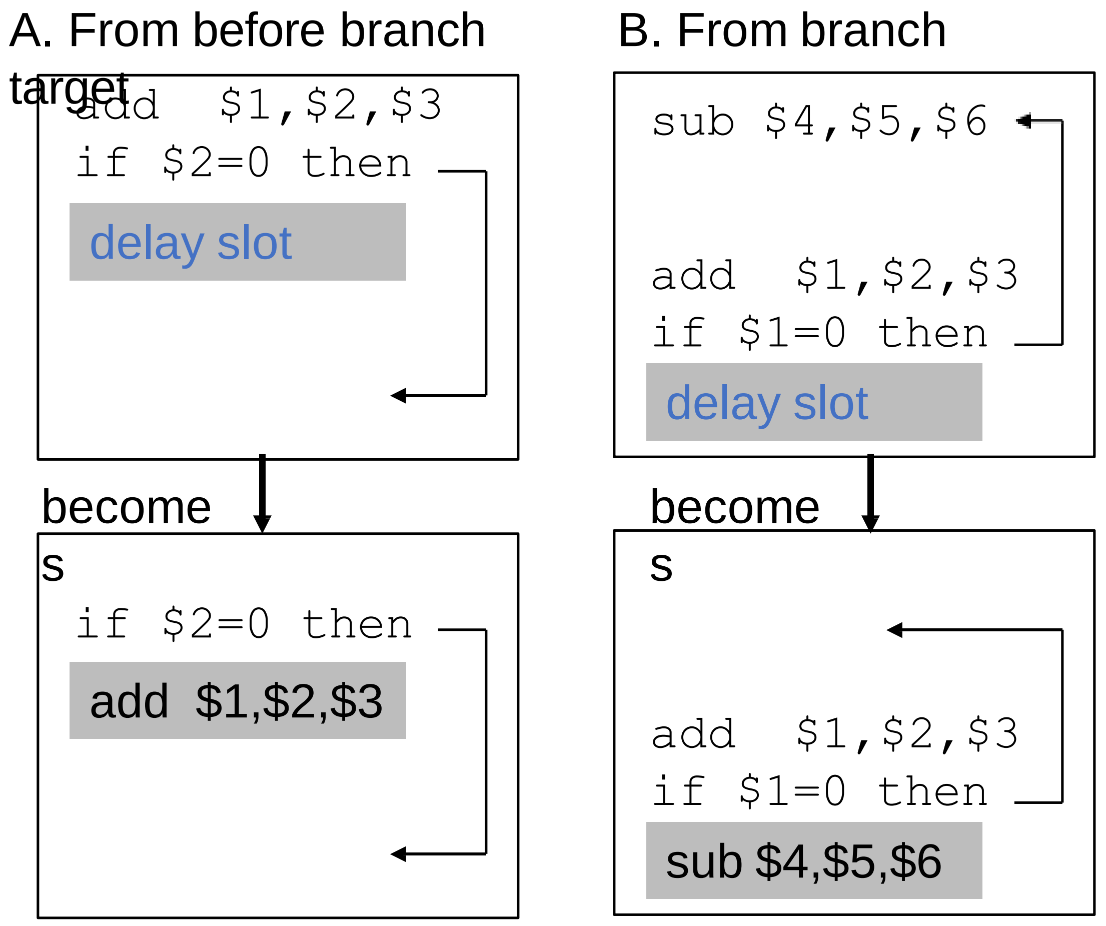
Scheduling Branch Delay Slots

A. From before branch



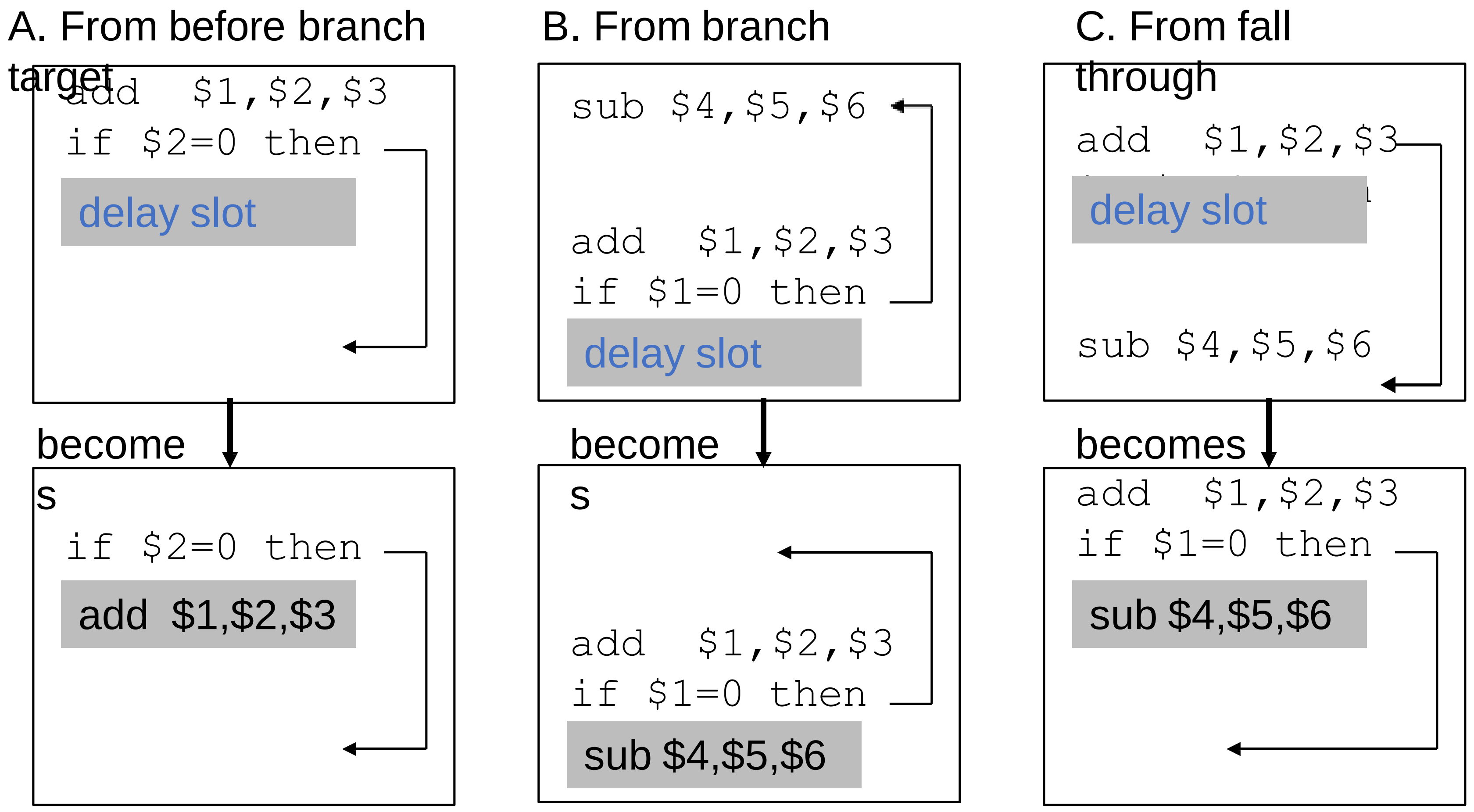
A is the best choice, fills delay slot & reduces instruction count (IC)

Scheduling Branch Delay Slots



A is the best choice, fills delay slot & reduces instruction count (IC)

Scheduling Branch Delay Slots



A. From before branch target

```
add  $1,$2,$3
if $2=0 then
delay slot
```

becomes

```
if $2=0 then
add $1,$2,$3
```

B. From branch

```
sub $4,$5,$6

add  $1,$2,$3
if $1=0 then
delay slot
```

becomes

```
add  $1,$2,$3
if $1=0 then
sub $4,$5,$6
```

C. From fall through

```
add  $1,$2,$3
delay slot

sub $4,$5,$6
```

becomes

```
add  $1,$2,$3
if $1=0 then
sub $4,$5,$6
```

A is the best choice

Word of Caution!

Do not put a branch
in the branch delay slot ☹

Stalls and Performance

For a program with N instructions and S stall cycles,

Average CPI = $\dfrac{N}{N}$

Stalls and Performance

For a program with N instructions and S stall cycles,

$$\text{Average CPI} = \frac{N+S}{N}$$

New Pipeline Speedup

$$\text{Pipeline Speedup} = \frac{\text{Pipeline Depth}}{1+\text{pipeline stalls because of branches}}$$

Pipeline stalls (branches) = Branch frequency X penalty

Summary

Data Hazards

Bypassing/forwarding

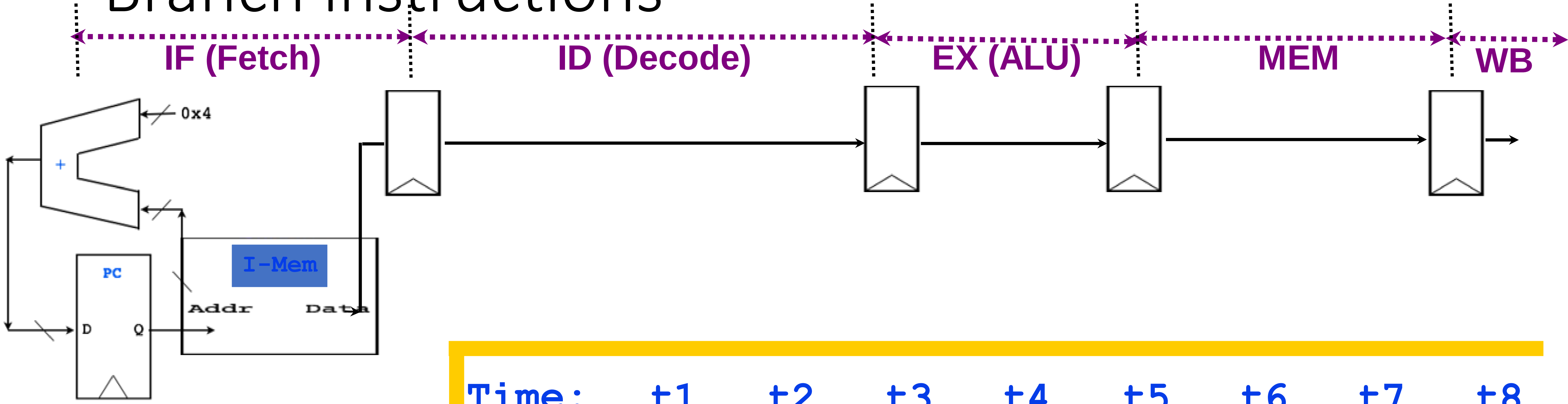
Stalls (NOPs) – if no scope for bypassing

Control hazards

Speculate, PC=PC+4, kill the wrong path

Delayed branch with the help of branch delay slots, new pipeline speedup

Branch instructions

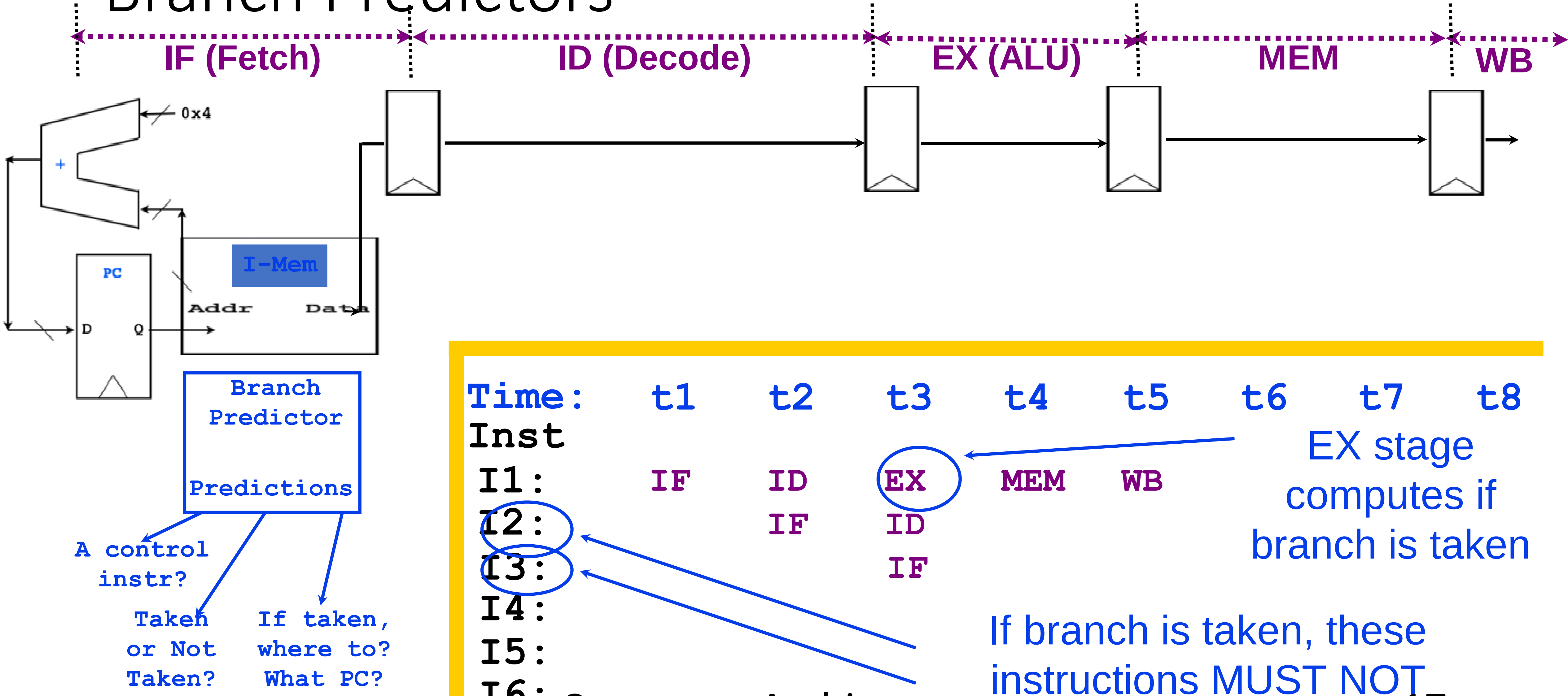


Time:	t1	t2	t3	t4	t5	t6	t7	t8
Inst								
I1:	IF	ID	EX	MEM	WB			
I2:		IF	ID					
I3:			IF					
I4:								
I5:								
I6:								

EX stage computes if branch is taken

If branch is taken, these instructions MUST NOT complete!

Branch Predictors



Time:	t1	t2	t3	t4	t5	t6	t7	t8
Inst								
I1:	IF	ID	EX	MEM	WB			
I2:		IF	ID					
I3:			IF					
I4:								
I5:								
I6:								

EX stage computes if branch is taken

If branch is taken, these instructions MUST NOT complete!

A quick recap

What if PC=PC+4? Not TRUE

Flush/kill all the instructions in the wrong path.

Branch Prediction: 10K Feet View

Predict whether the next PC is a branch PC, at the fetch stage?

Predict whether the next PC is a branch PC, at the fetch stage?

If branch, will it be taken?

Predict whether the next PC is a branch PC, at the fetch stage?

If branch, will it be taken?

If taken, what is the target address?

Predict whether the next PC is a branch PC, at the fetch stage?

If branch, will it be taken?

If taken, what is the target address?

How?

Predict whether the next PC is a branch PC, at the fetch stage?

If branch, will it be taken?

If taken, what is the target address?

How?

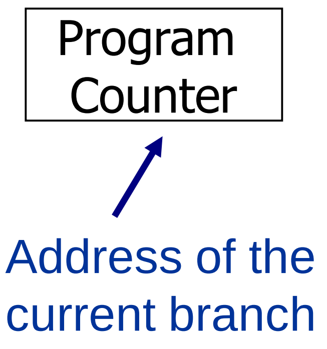
We know whether it is a branch PC or not in the decode stage. Oh no ☹

Branch Predictor: A bit deeper

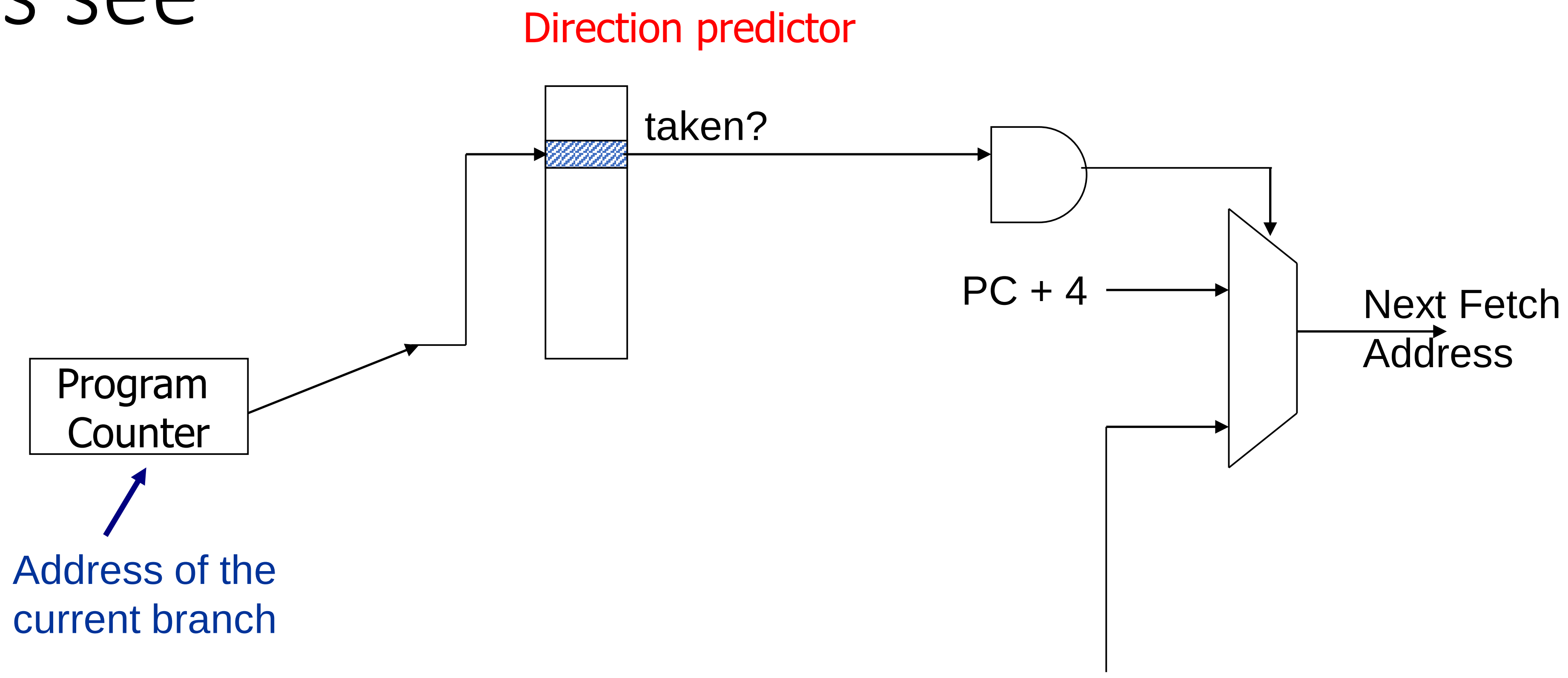
Three tasks

1. Is the PC a branch/jump? YES/NO
2. If Yes, can we predict the direction? Taken or not-taken
3. If taken, can we predict the target address?

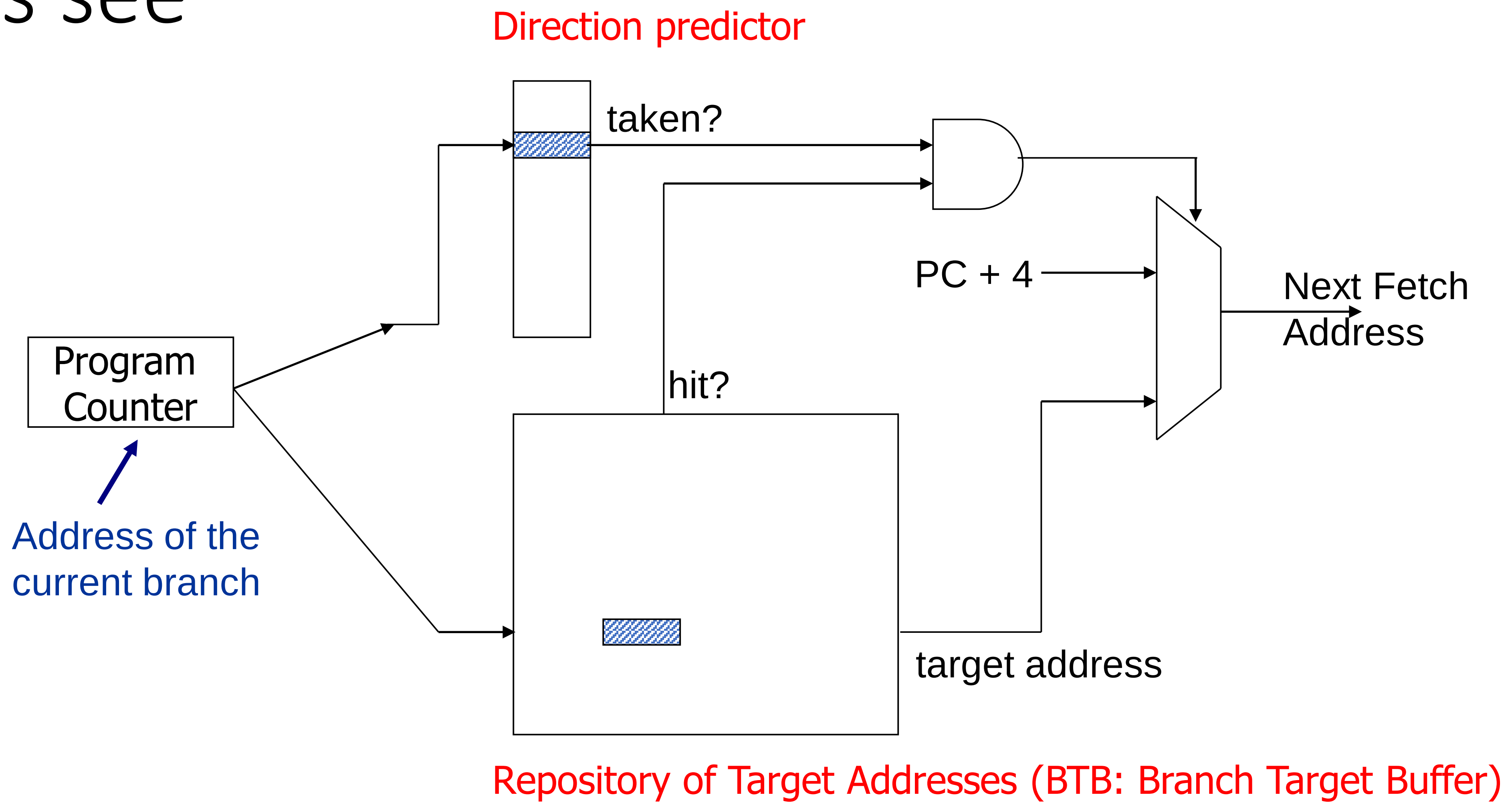
Let’s see



Let's see


Direction predictor
taken?
PC + 4
Next Fetch Address
Program Counter
Address of the current branch

Let’s see


Direction predictor
taken?
PC + 4
Next Fetch Address
Program Counter
hit?
Address of the current branch
target address
Repository of Target Addresses (BTB: Branch Target Buffer)

Static (compiler) Direction Prediction Techniques

Always not-taken: Simple to implement: no need for BTB, no direction prediction

Low accuracy: ~30-40%

Always taken: No direction prediction, we need BTB though

Better accuracy: ~60-70%

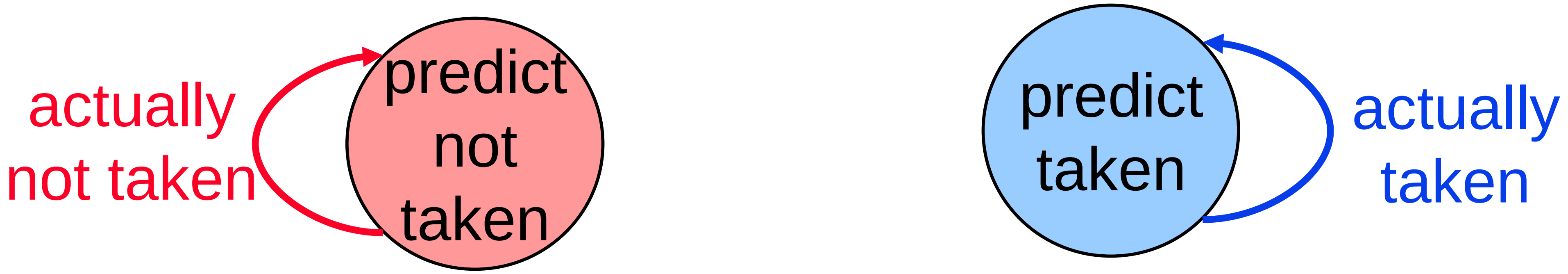
Backward branches (i.e., loop branches) are usually taken

Dynamic Predictors

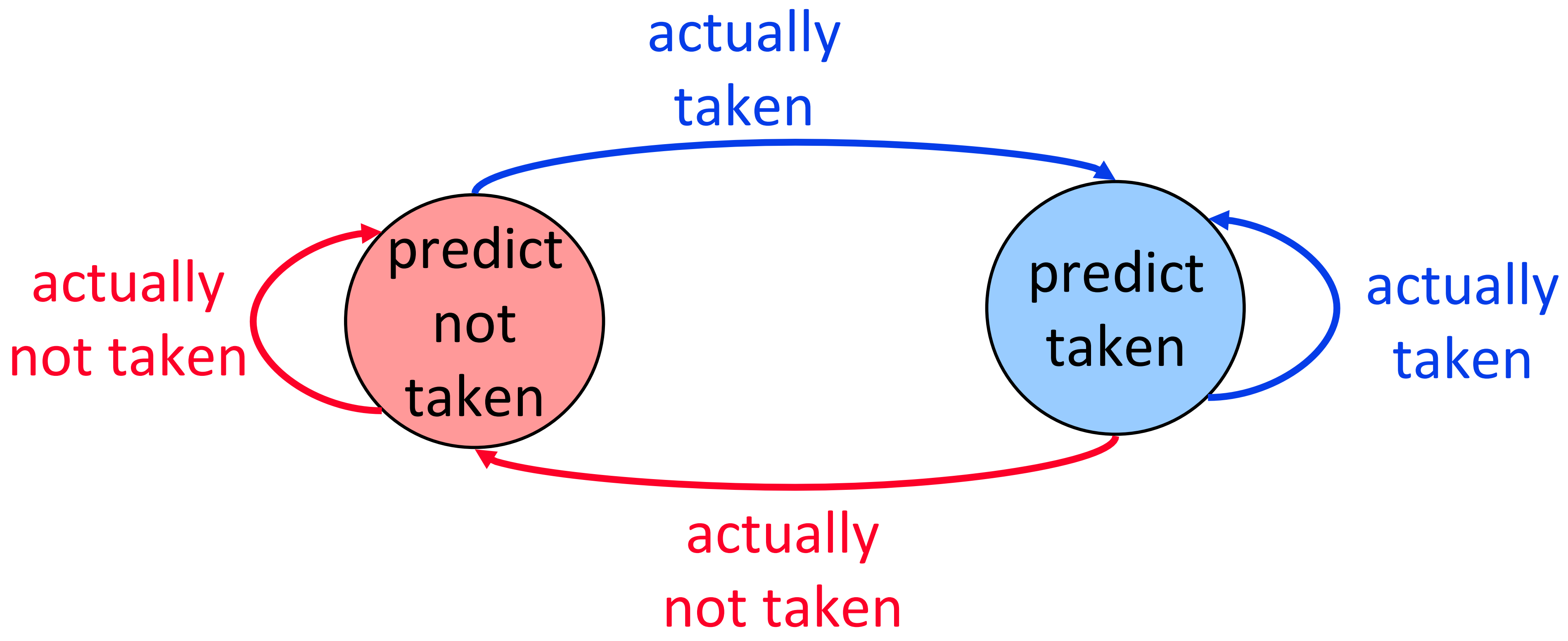
Microarchitectural way of predicting it.

Simple one: Last time predictor

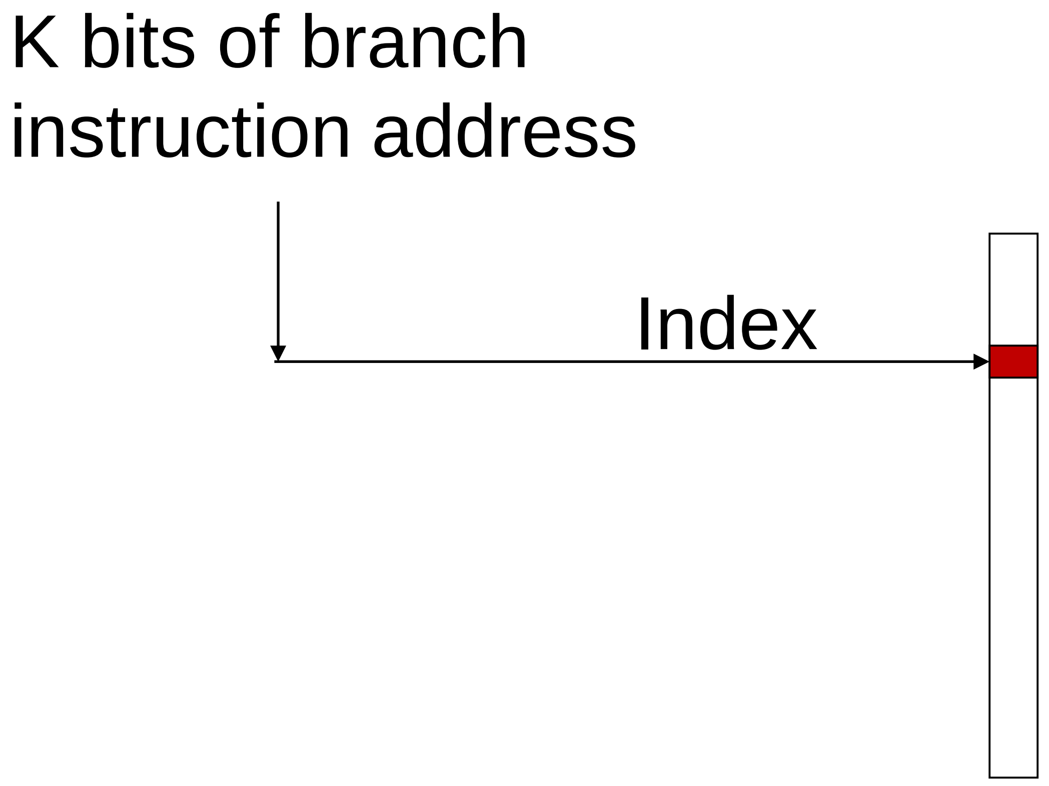
Last-time predictor



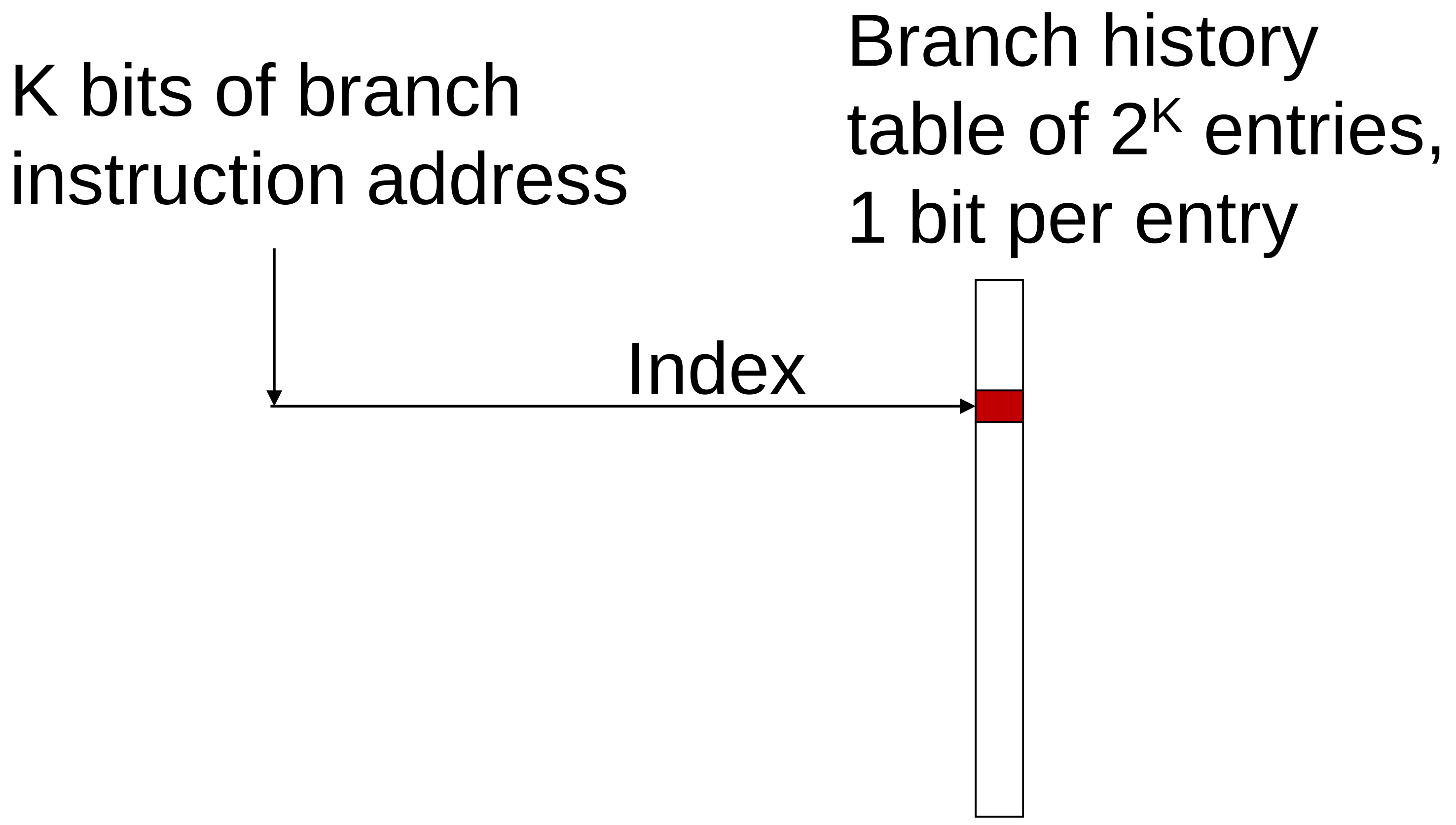
Last-time predictor



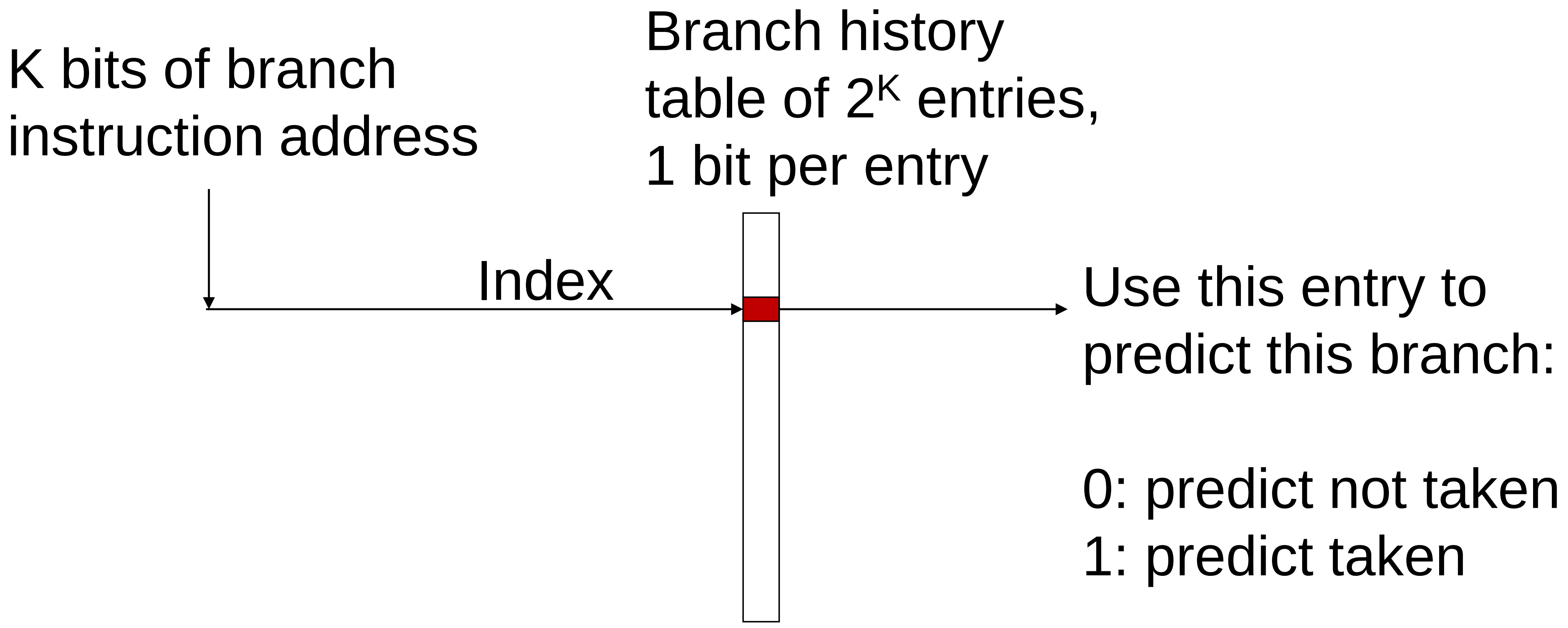
Implementation



Implementation


K bits of branch instruction address
Branch history table of 2^K entries, 1 bit per entry
Index

Implementation


K bits of branch instruction address
Branch history table of 2^K entries, 1 bit per entry
Index
Use this entry to predict this branch:
0: predict not taken
1: predict taken

Performance of Last-time predictor

TTTTTTTTTTNNNNNNNNNN - 90% accuracy

Always mispredicts the last iteration and the first iteration of a loop branch

Accuracy for a loop with N iterations = (N-2)/N

+ Loop branches for loops with large number of iterations

-- Loop branches for loops will small number of iterations

Performance contd.

TNTNTNTNTNTNTNTNTNTN → 0% accuracy

20% of all instructions are branches, 85% accuracy

Last-time predictor CPI =

[1 + (0.20*0.15) * 2] =

1.06 (minimum two stalls to resolve a branch)