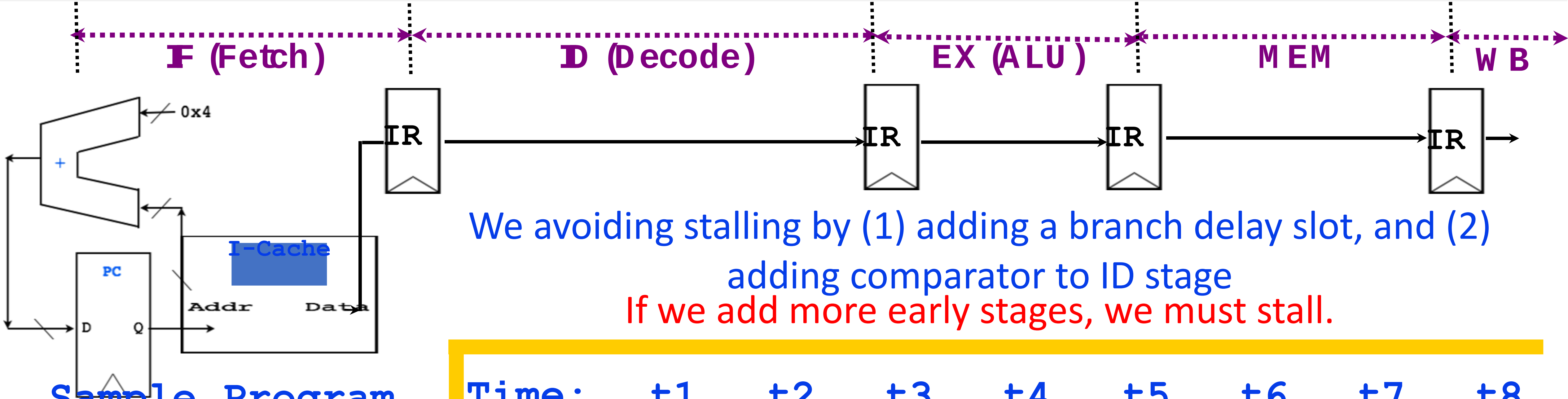
Lecture-9 (Branch Prediction)
CS422-Spring 2018

Biswa@CSE-IITK

Remember This



We avoiding stalling by (1) adding a branch delay slot, and (2) adding comparator to ID stage

If we add more early stages, we must stall.

Sample Program (ISA w/o branch delay slot)

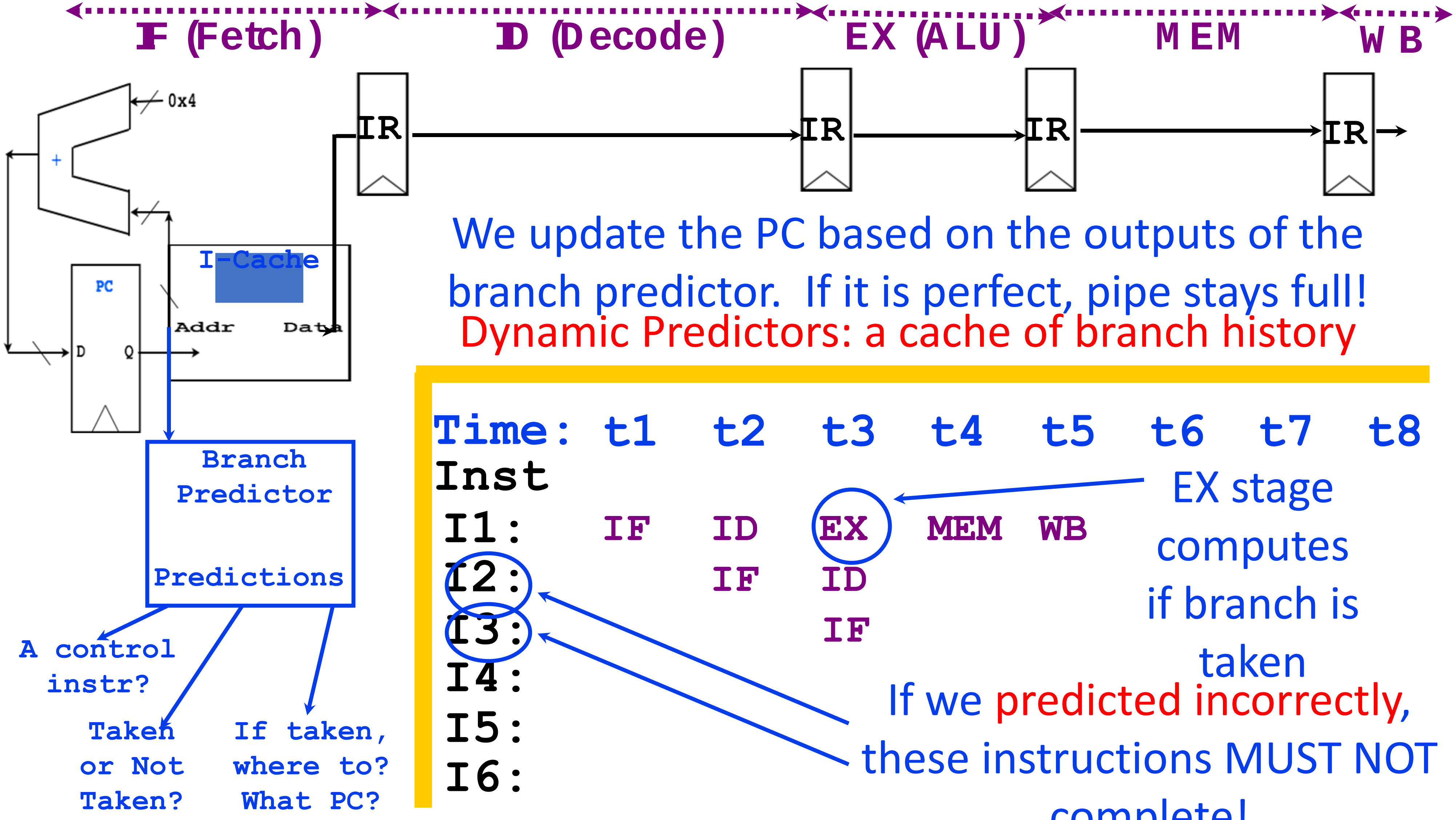
```
I1:   BEQ R4,R3,25
I2:   AND R6,R5,R4
I3:   SUB R1,R9,R8
```

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!

Welcome to Branch Prediction



We update the PC based on the outputs of the branch predictor. If it is perfect, pipe stays full!

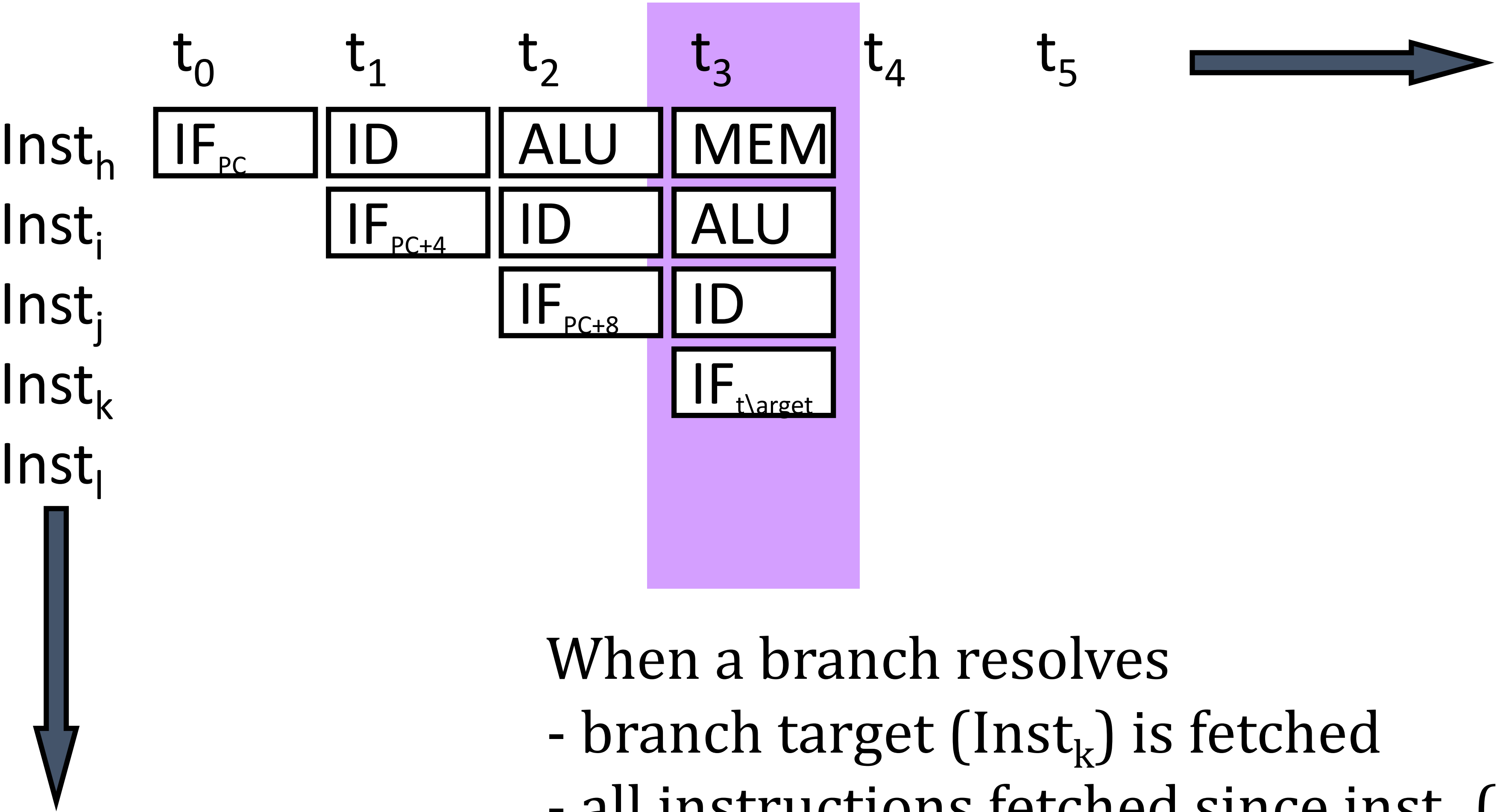
Dynamic Predictors: a cache of branch history

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 we predicted incorrectly, these instructions MUST NOT complete!

If always PC+4?

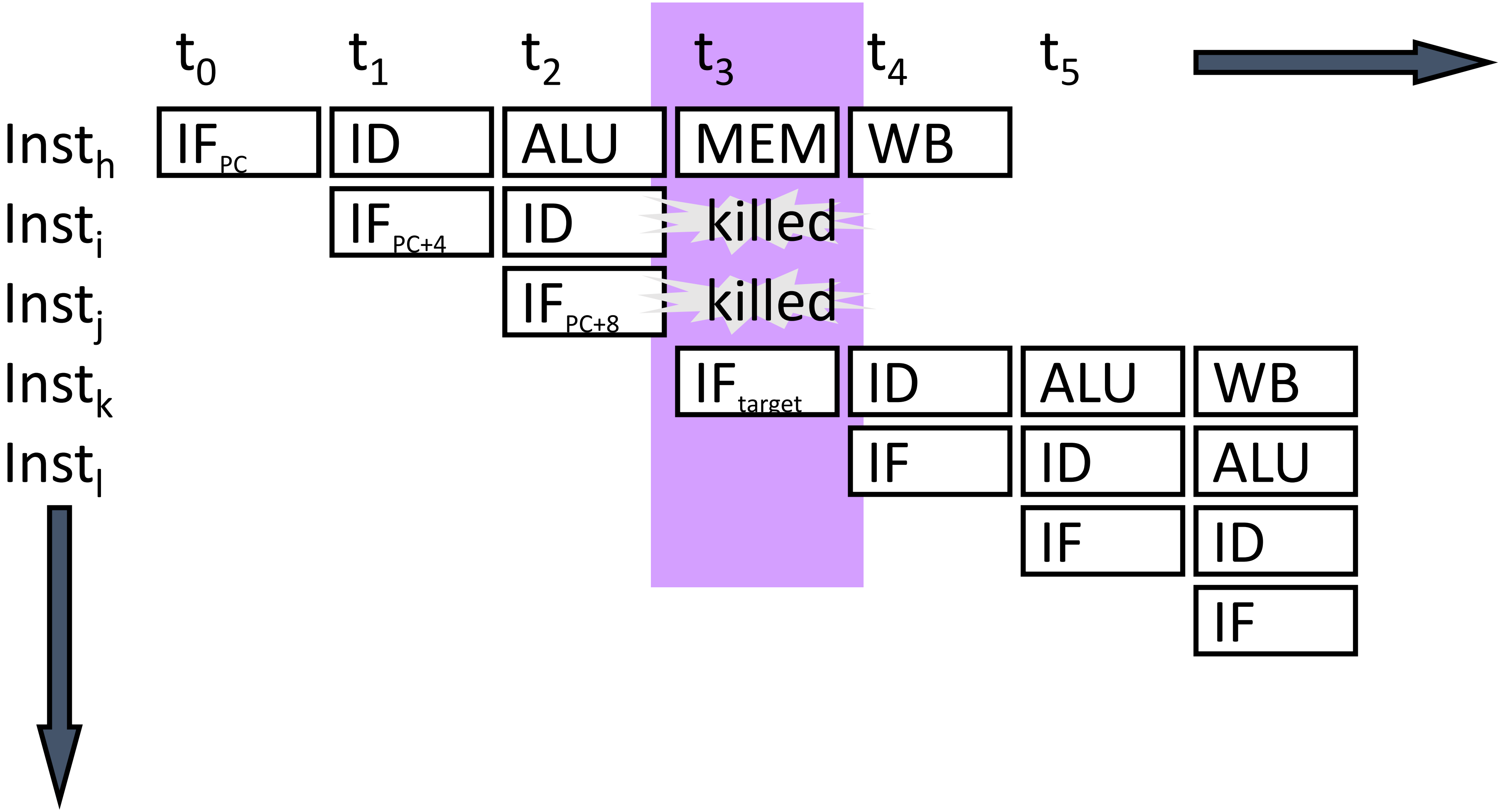


	t_0	t_1	t_2	t_3	t_4	t_5
Inst_h	IF_{PC}	ID	ALU	MEM		
Inst_i		$\text{IF}_{\text{PC+4}}$	ID	ALU		
Inst_j			$\text{IF}_{\text{PC+8}}$	ID		
Inst_k				$\text{IF}_{\text{t\textbackslash arget}}$		
Inst_l						

When a branch resolves

- branch target (Inst_k) is fetched
- all instructions fetched since inst_h (so called "wrong-path" instructions) must be flushed

Flush on a Mispred.



	t_0	t_1	t_2	t_3	t_4	t_5	
$Inst_h$	IF_{PC}	ID	ALU	MEM	WB		
$Inst_i$		IF_{PC+4}	ID	killed			
$Inst_j$			IF_{PC+8}	killed			
$Inst_k$				IF_{target}	ID	ALU	WB
$Inst_l$					IF	ID	ALU
						IF	ID
							IF

$Inst_h$ is a branch

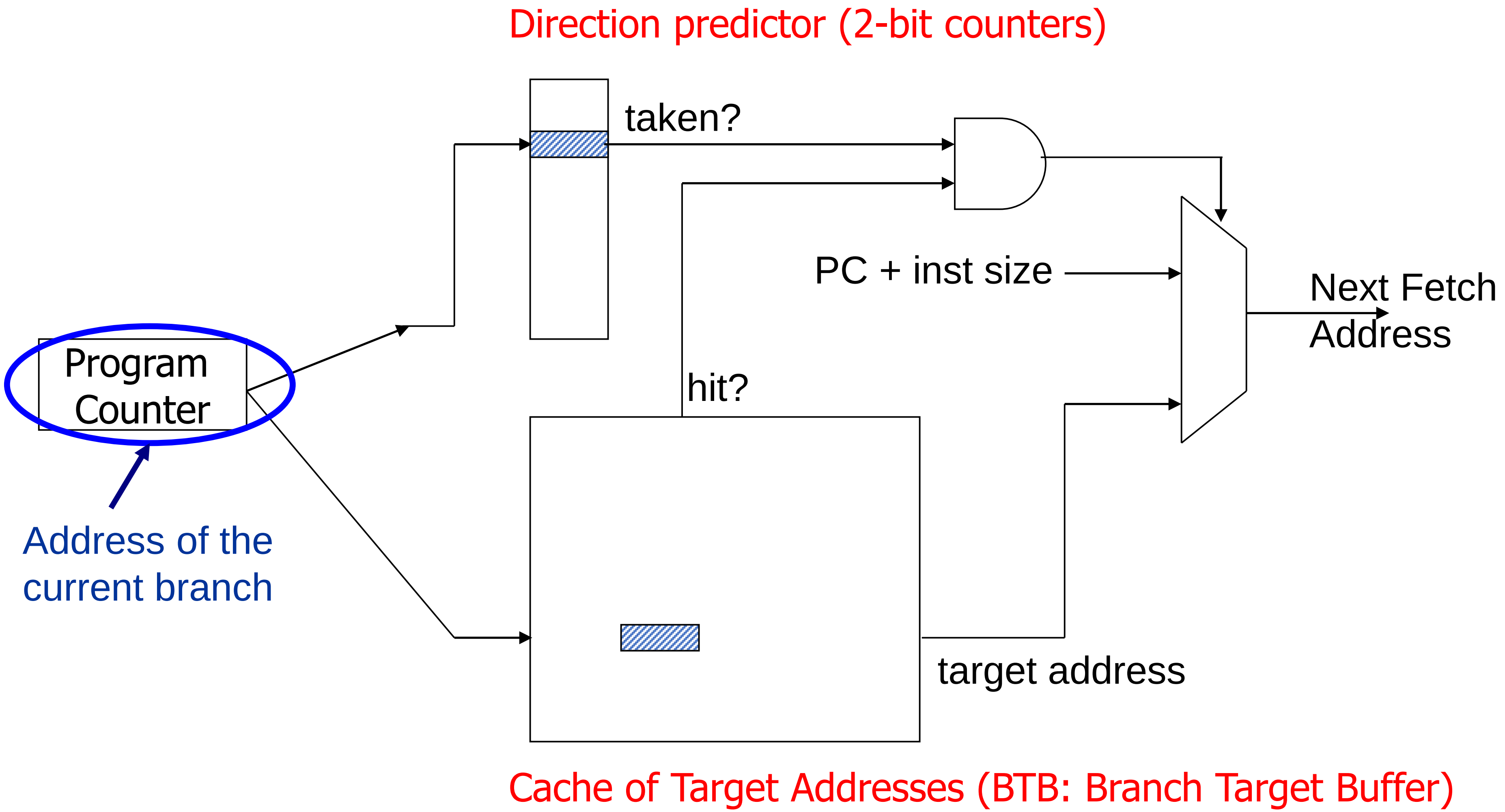
Branch Prediction

- Idea: Predict the next fetch address (to be used in the next cycle)

- Requires three things to be predicted at fetch stage:
  - Whether the fetched instruction is a branch
  - (Conditional) branch direction
  - Branch target address (if taken)

- Observation: Target address remains the same for a conditional direct branch across dynamic instances
  - Idea: Store the target address from previous instance and access it with the PC
  - Called Branch Target Buffer (BTB) or Branch Target Address Cache

Fetch Stage with BTB and Direction Prediction


Direction predictor (2-bit counters)
taken?
PC + inst size
Next Fetch Address
Program Counter
hit?
Address of the current branch
target address
Cache of Target Addresses (BTB: Branch Target Buffer)

Always taken CPI = [1 + (0.20*0.3) * 2] = 1.12 (70% of branches taken)

Static Branch Prediction

- Always not-taken
  - Simple to implement: no need for BTB, no direction prediction
  - Low accuracy: ~30-40%

- Always taken
  - No direction prediction
  - Better accuracy: ~60-70%
    - Backward branches (i.e. loop branches) are usually taken

- Backward taken, forward not taken (BTFN)
  - Predict backward (loop) branches as taken, others not-taken

Static Branch Prediction

- Profile-based
  - Idea: Compiler determines likely direction for each branch using profile run. Encodes that direction as a hint bit in the branch instruction format.

+ Per branch prediction (more accurate than schemes in previous slide) → accurate if profile is representative!

-- Requires hint bits in the branch instruction format

-- Accuracy depends on dynamic branch behavior:

TTTTTTTTTTNNNNNNNNNN → 50% accuracy
TNTNTNTNTNTNTNTNTNTN → 50% accuracy

-- Accuracy depends on the representativeness of profile input set

Dynamic Branch Prediction

- Idea: Predict branches based on dynamic information (collected at run-time)

- Advantages

  + Prediction based on history of the execution of branches

    + It can adapt to dynamic changes in branch behavior

  + No need for static profiling: input set representativeness problem goes away

- Disadvantages

  -- More complex (requires additional hardware)

Last-Time Predictor

- Last time predictor
  - Single bit per branch (stored in BTB)
  - Indicates which direction branch went last time it executed

  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

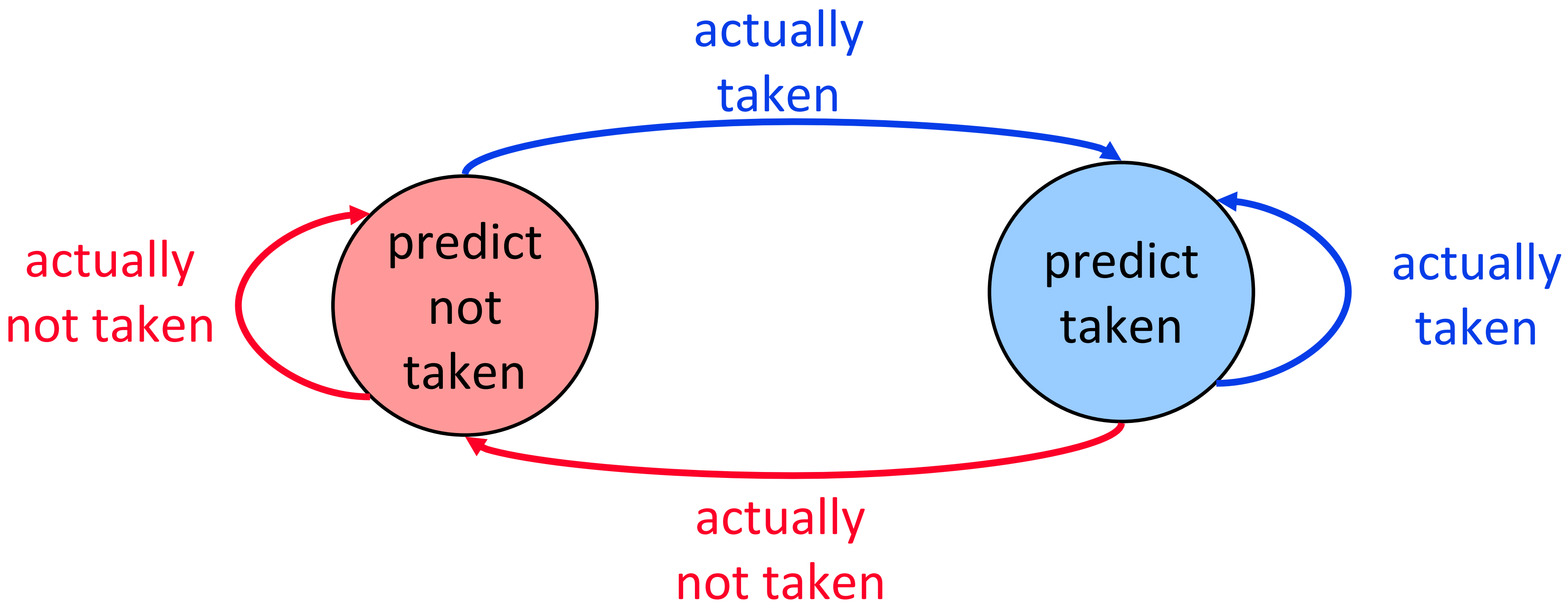
+ Loop branches for loops with large number of iterations

-- Loop branches for loops will small number of iterations

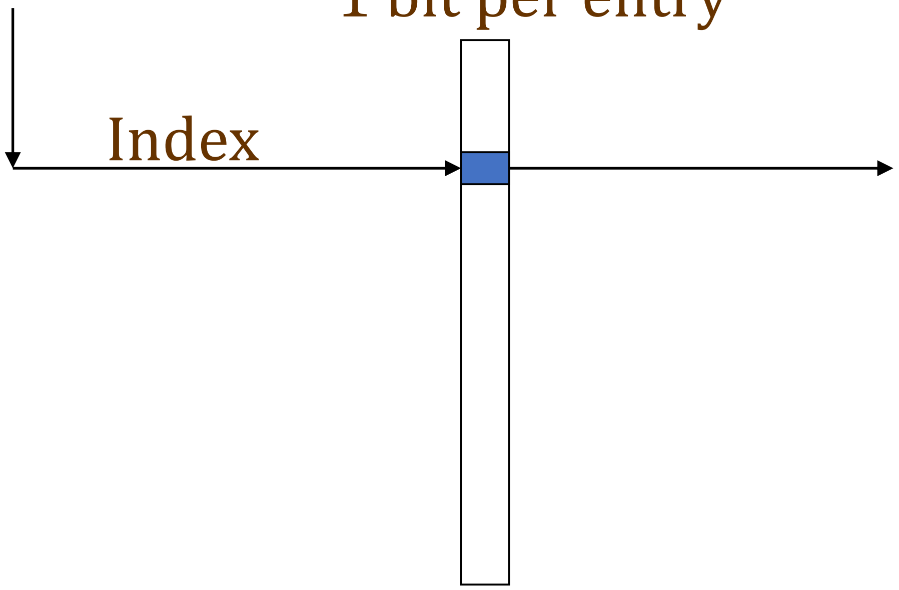
TNTNTNTNTNTNTNTNTNTN → 0% accuracy

Last-time predictor CPI = [1 + (0.20*0.15) * 2] = 1.06 (Assuming 85% accuracy)

Last-Time


actually taken
predict not taken
actually not taken
predict taken
actually taken
actually not taken

Last-time



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

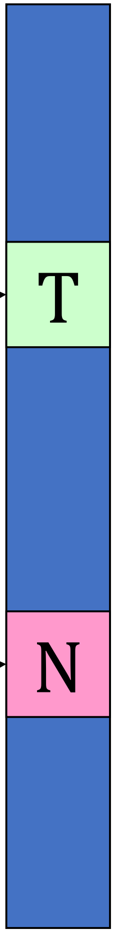
When branch direction resolved, go back into the table and update entry: 0 if not taken, 1 if taken

Example

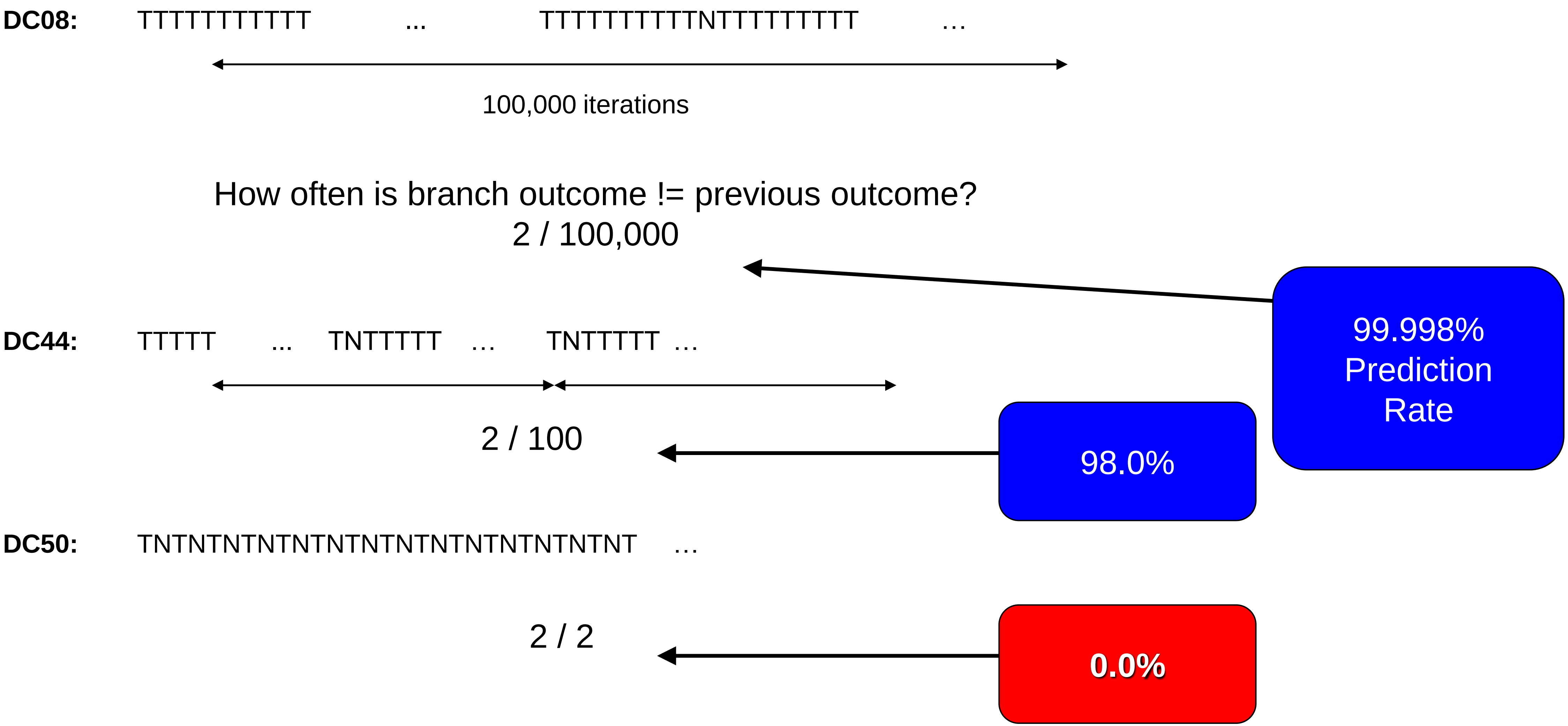
```
0xDC08:          for(i=0; i < 100000; i++)
                 {
0xDC44:                 if( ( i % 100) == 0 )
                        tick( );

0xDC50:                 if( (i & 1) == 1)
                                odd( );

             }
```



Example



DC08: TTTTTTTTTTT ... TTTTTTTTTTNTTTTTTTTT …

100,000 iterations

How often is branch outcome != previous outcome?
2 / 100,000

99.998% Prediction Rate

DC44: TTTTT ... TNTTTTT … TNTTTTT …

2 / 100

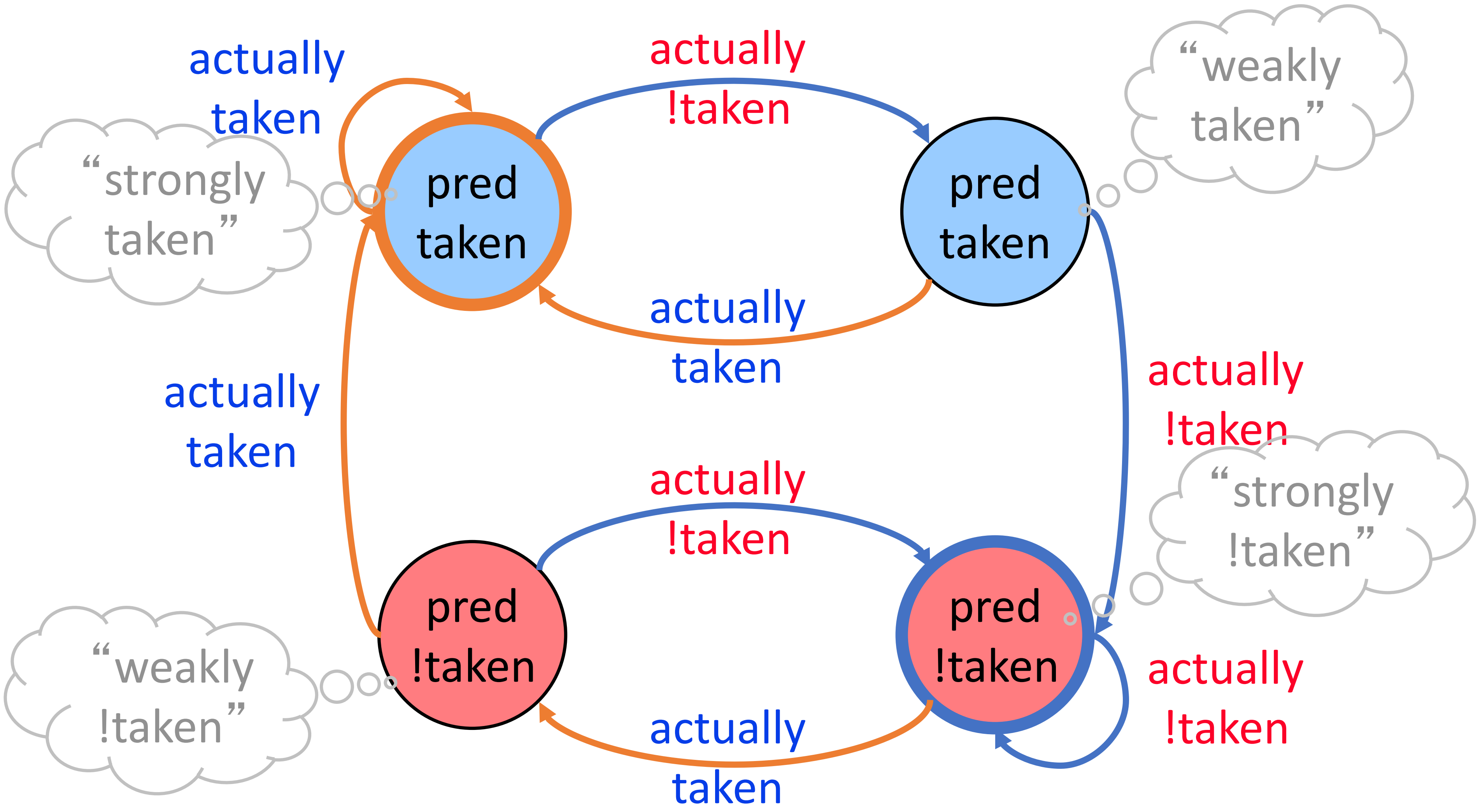
98.0%

DC50: TNTNTNTNTNTNTNTNTNTNTNTNTNTNT …

2 / 2

0.0%

Change Predictor after 2 Mistakes



Is This Enough

- Control flow instructions (branches) are frequent
  - 15-25% of all instructions

- Problem: Next fetch address after a control-flow instruction is not determined after N cycles in a pipelined processor
  - N cycles: (minimum) branch resolution latency
  - Stalling on a branch wastes instruction processing bandwidth (i.e. reduces IPC)

- How do we keep the pipeline full after a branch?
- Problem: Need to determine the **next fetch address** when the branch is fetched (to avoid a pipeline bubble)

Is This Enough?

- Assume a pipeline with 20-cycle branch resolution latency

- How long does it take to fetch 100 instructions?
  - Assume 1 out of 5 instructions is a branch
  - 100% accuracy
    - 100 cycles (all instructions fetched on the correct path)
    - No wasted work
  - 99% accuracy
    - 100 (correct path) + 20 (wrong path) = 120 cycles
    - 20% extra instructions fetched
  - 98% accuracy
    - 100 (correct path) + 20 * 2 (wrong path) = 140 cycles
    - 40% extra instructions fetched
  - 95% accuracy
    - 100 (correct path) + 20 * 5 (wrong path) = 200 cycles
    - 100% extra instructions fetched

Who Cares ?

- 98% → 99%
  - Who cares?
  - Actually, it's 2% misprediction rate → 1%
  - That's a halving of the number of mispredictions
- So what?
  - Halving the miss rate doubles the number of useful instructions that we can try to extract ILP from
  - Piazaa + 2

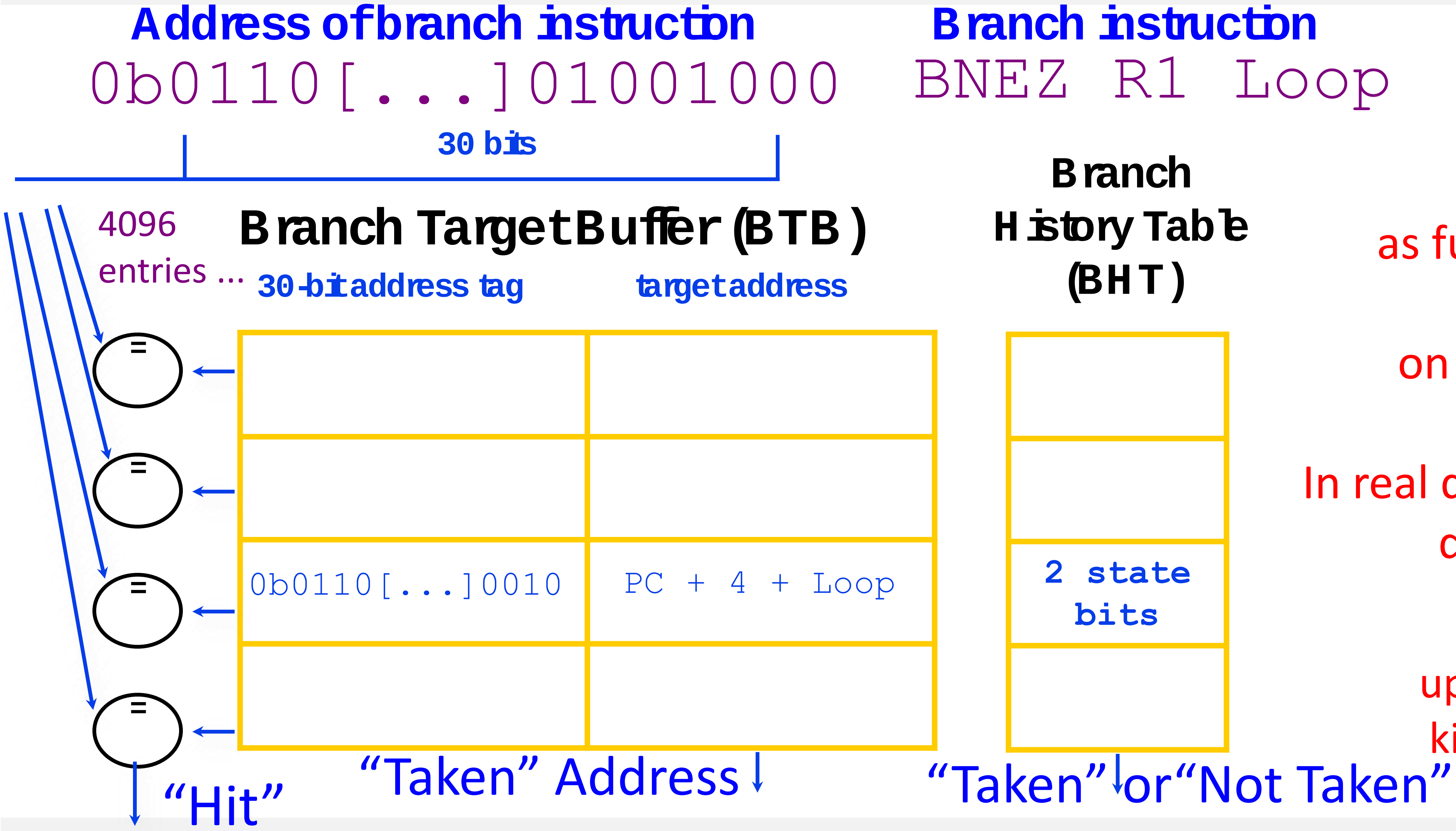
Local History & Global History

- Local Behavior
  - What is the predicted direction of Branch A given the outcomes of previous instances of Branch A?

- Global Behavior
  - What is the predicted direction of Branch Z given the outcomes of *all** previous branches A, B, …, X and Y?

* number of previous branches tracked limited by the history length

BTB (What about JALR ? Why 30-bit Tag?)



Address of branch instruction

0b0110[...]01001000

Branch instruction

BNEZ R1 Loop

30 bits

4096 entries ...

Branch Target Buffer (BTB)

30-bit address tag

target address

0b0110[...]0010	PC + 4 + Loop

Branch History Table (BHT)

2 state bits

"Hit"

"Taken" Address

"Taken" or "Not Taken"

Drawn as fully associative to focus on the essentials.

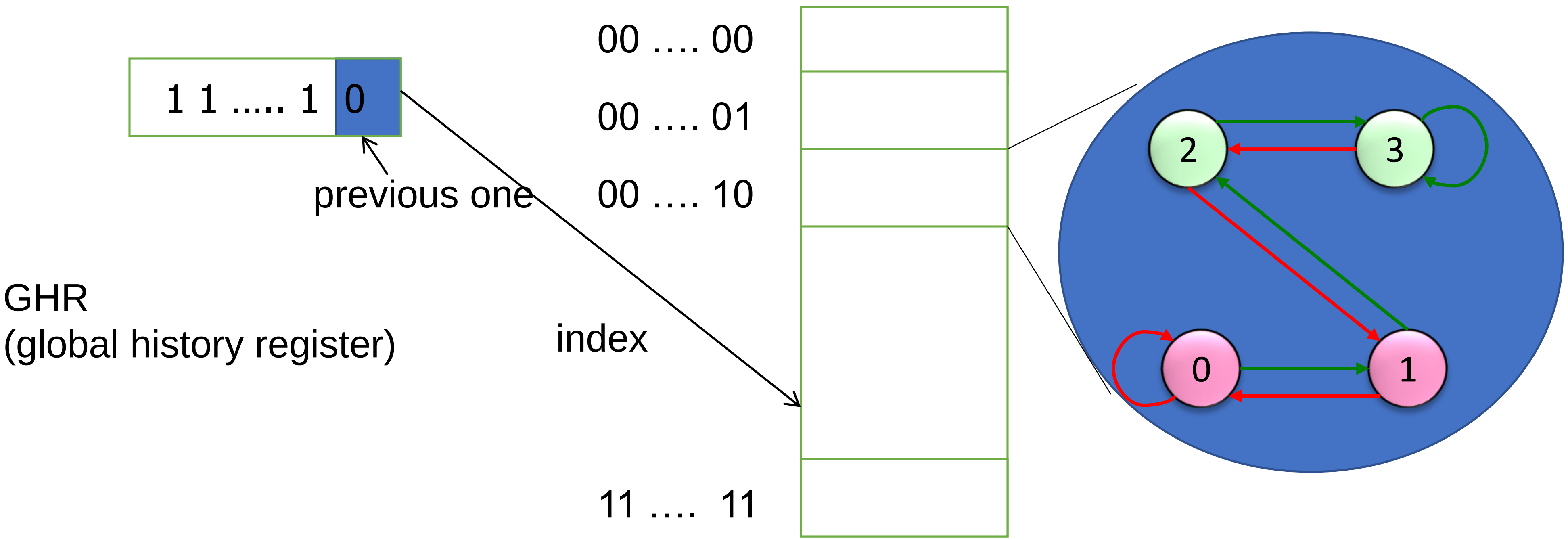
In real designs, always direct-mapped.

At EX stage, update BTB/BHT, kill instructions, if necessary,

Two Level Global Branch Prediction [MICRO ‘91]

- First level: Global branch history register (N bits)
  - The direction of last N branches
- Second level: Table of saturating counters for each history entry
  - The direction the branch took the last time the same history was seen

Pattern History Table (PHT)



1 1 ..... 1 0

previous one

GHR
(global history register)

index

00 …. 00

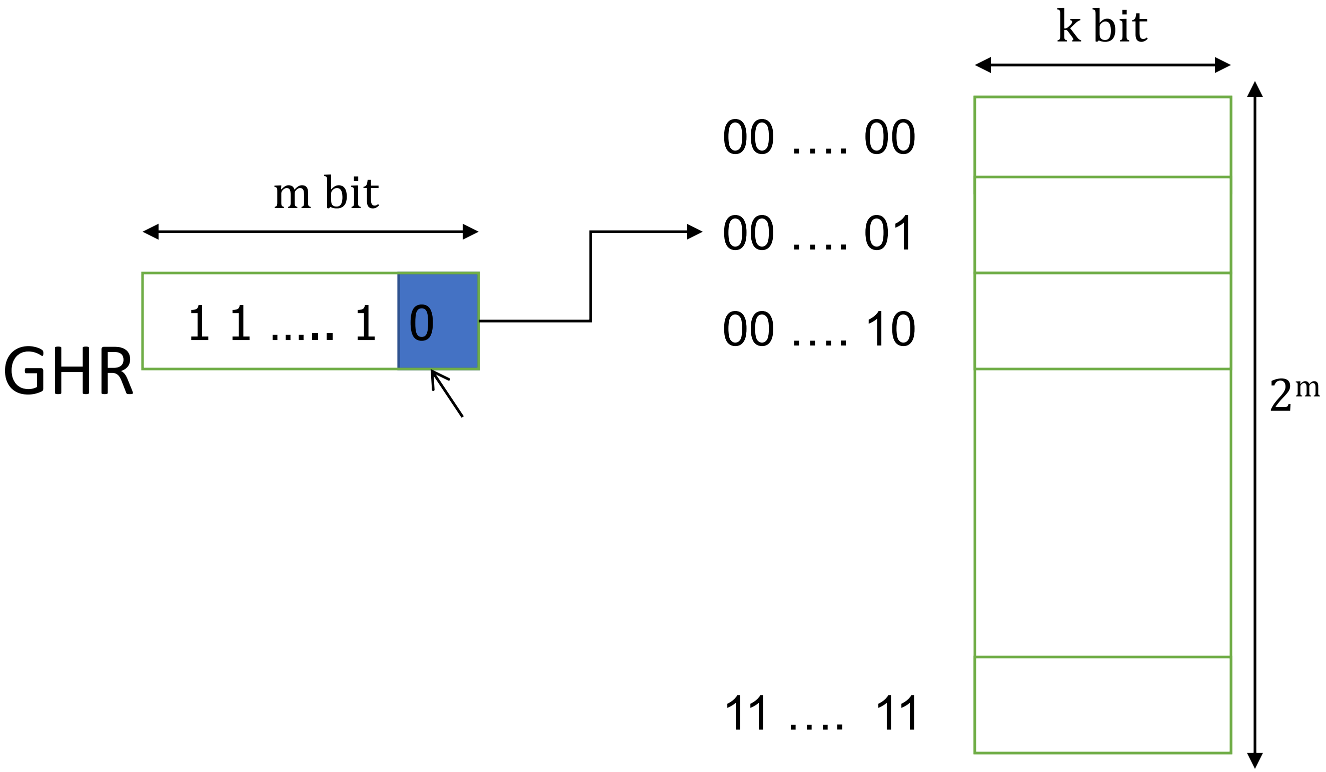
00 …. 01

00 …. 10

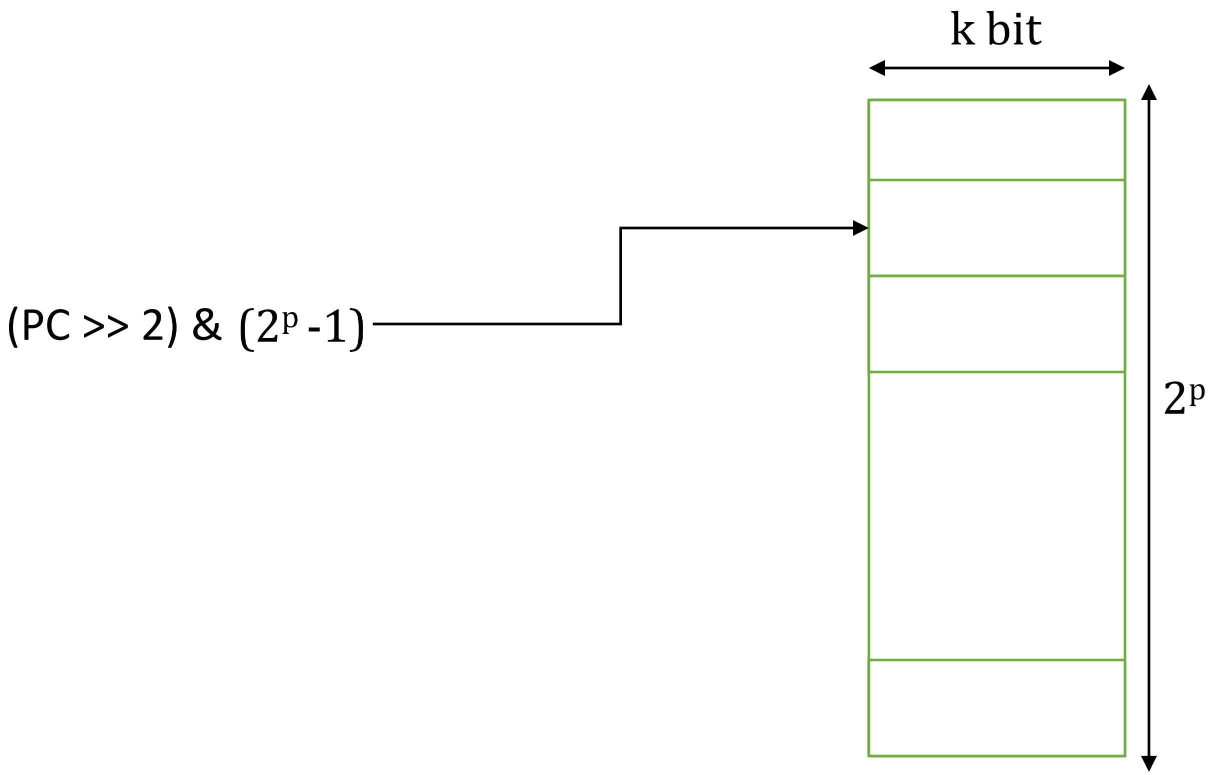
11 …. 11

PHT

- Table of saturating counters

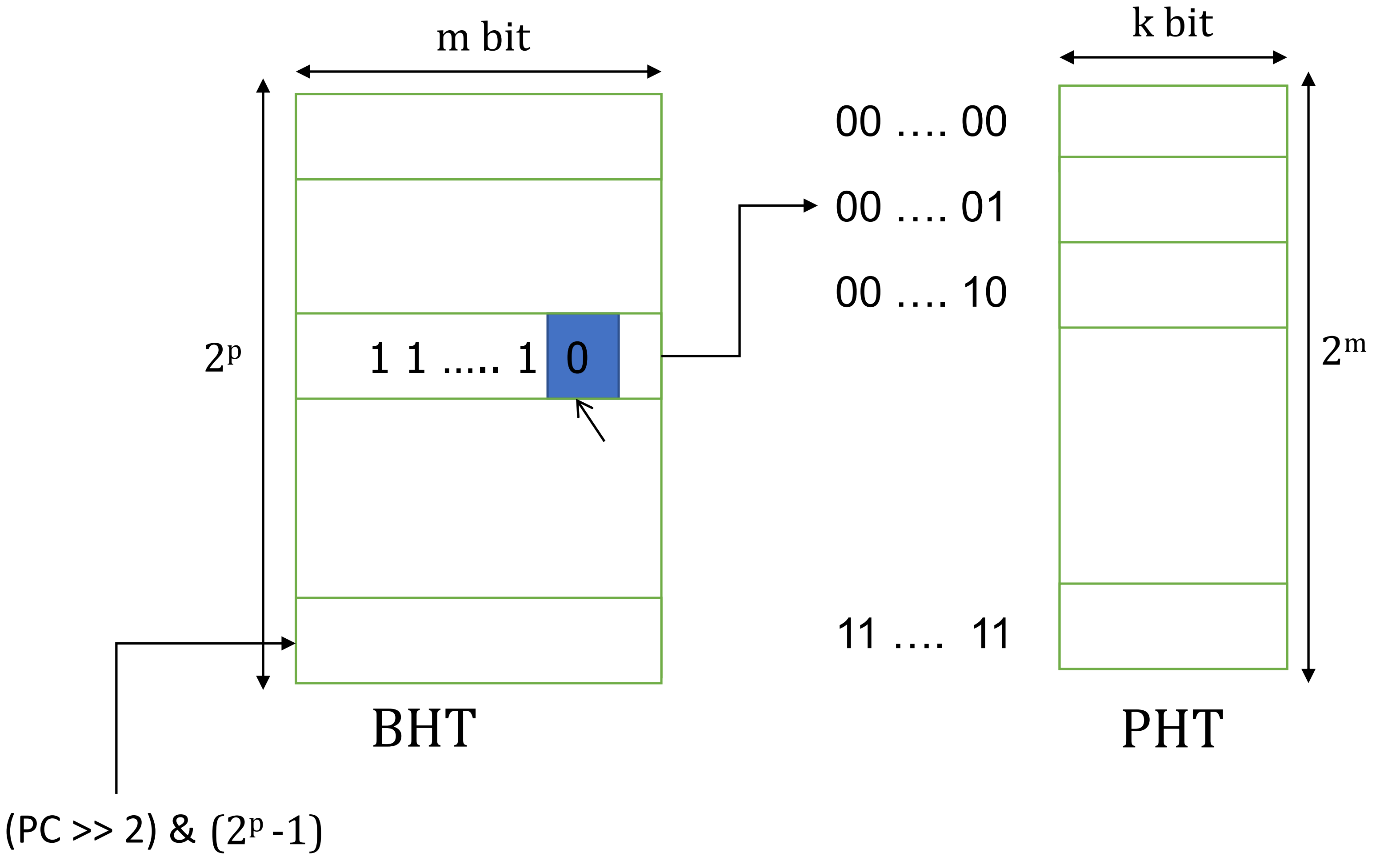


What about – NO GHR?



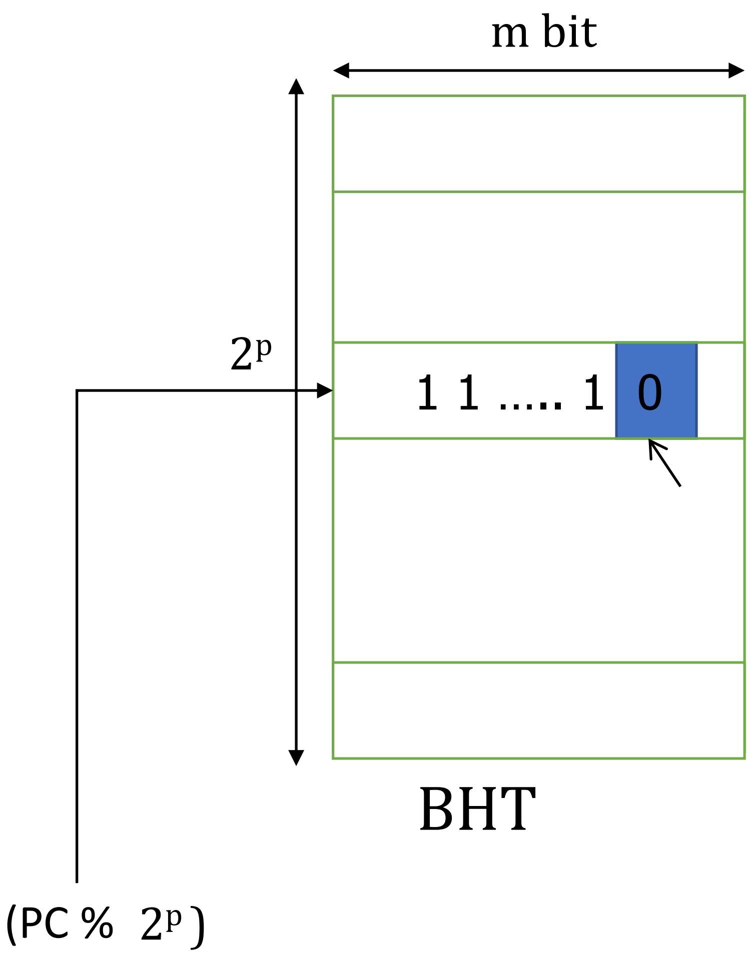
Bimodal predictor: Good for biased branches

GHR per Branch (Gain/Loss?)

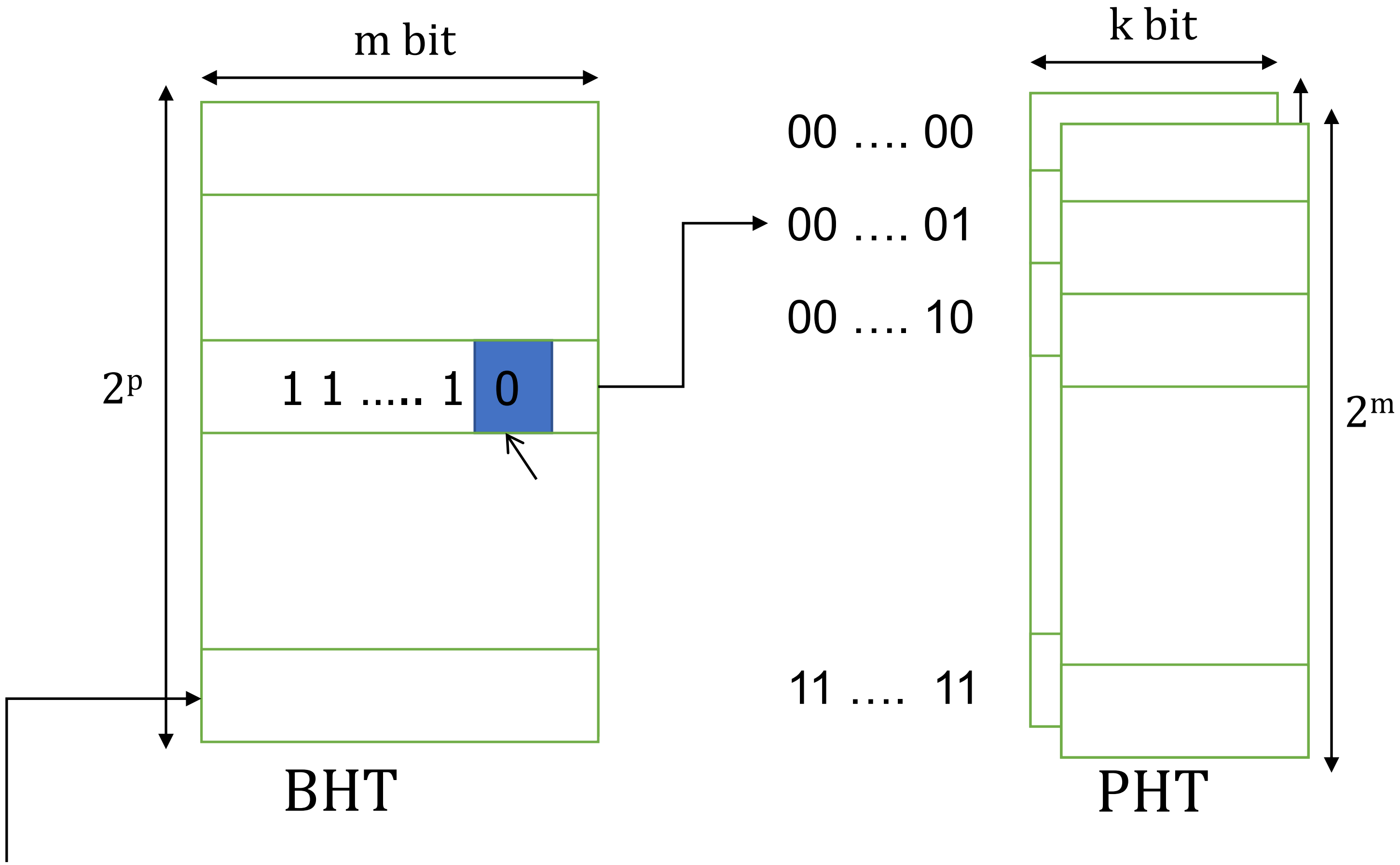


How large: k ? Mostly K=2, m =12, how large m?

Set of Branches – One Register

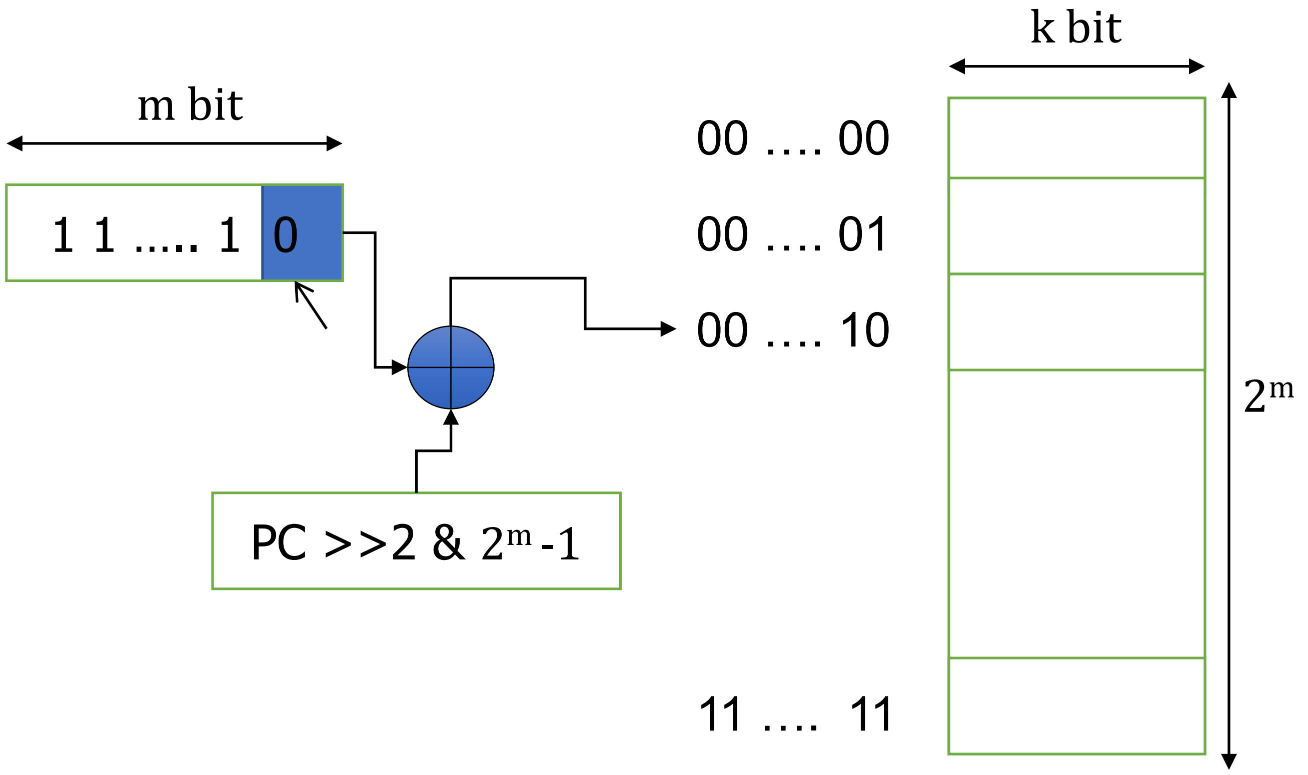


What if One Branch -> One History -> One PHT ?



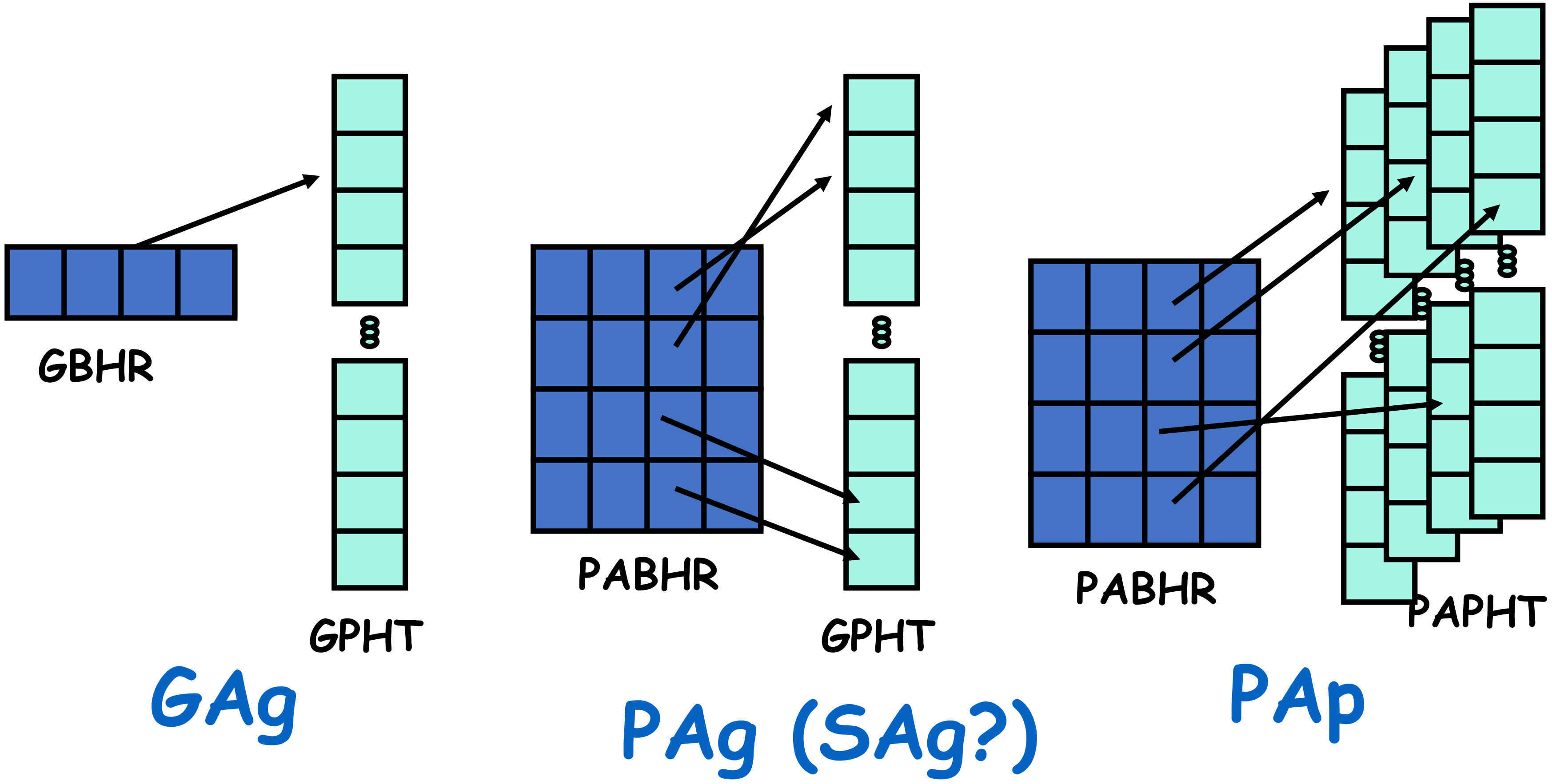
(PC >> 2) & (2^p -1)

GShare



For a given history and for a given branch (PC) counters are trained

Y & P Classification [MICRO 91]



- GAg: Global History Register, Global History Table
- PAg: Per-Address History Register, Global History Table
- PAp: Per-Address History Register, Per-Address History Table