Lecture-7 (Branch Predictors)
CS422-Spring 2020

Biswa@CSE-IITK

Welcome to the World of Predictors

Impact of a Branch?

Average dynamic instruction mix of SPEC CPU 2017 [Limaye and Adegbiya , ISPASS'18]:

	SPECint	SPECfp
Branches	19 %	11 %
Loads	24 %	26 %
Stores	10 %	7 %
Other	47 %	56 %

SPECint17: *perlbench, gcc, mcf, omnetpp, xalancbmk, x264, deepsjeng, leela, exchange2, xz*
SPECfp17: *bwaves, cactus, lbm, wrf, pop2, imagick, nab, fotonik3d, roms*

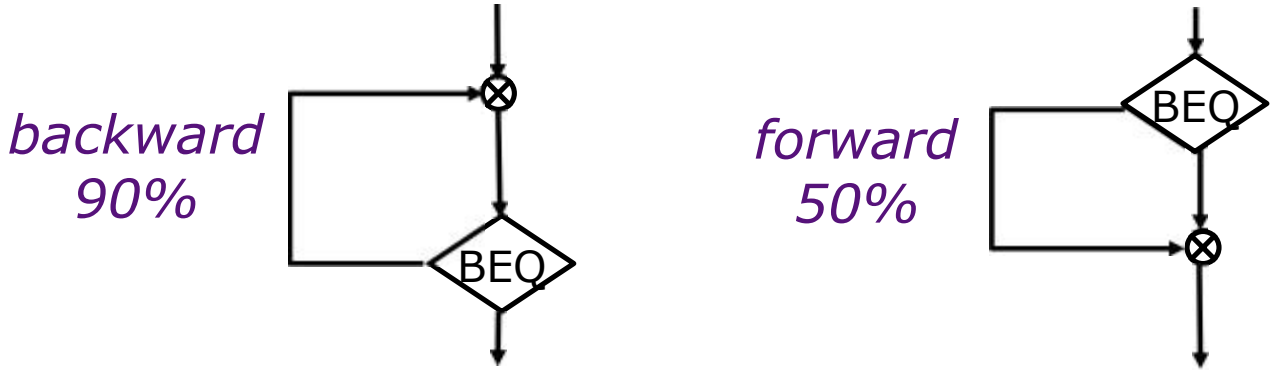
What is the average run length *between branches?*

Roughly 5-10 instructions

Branches and Jumps

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

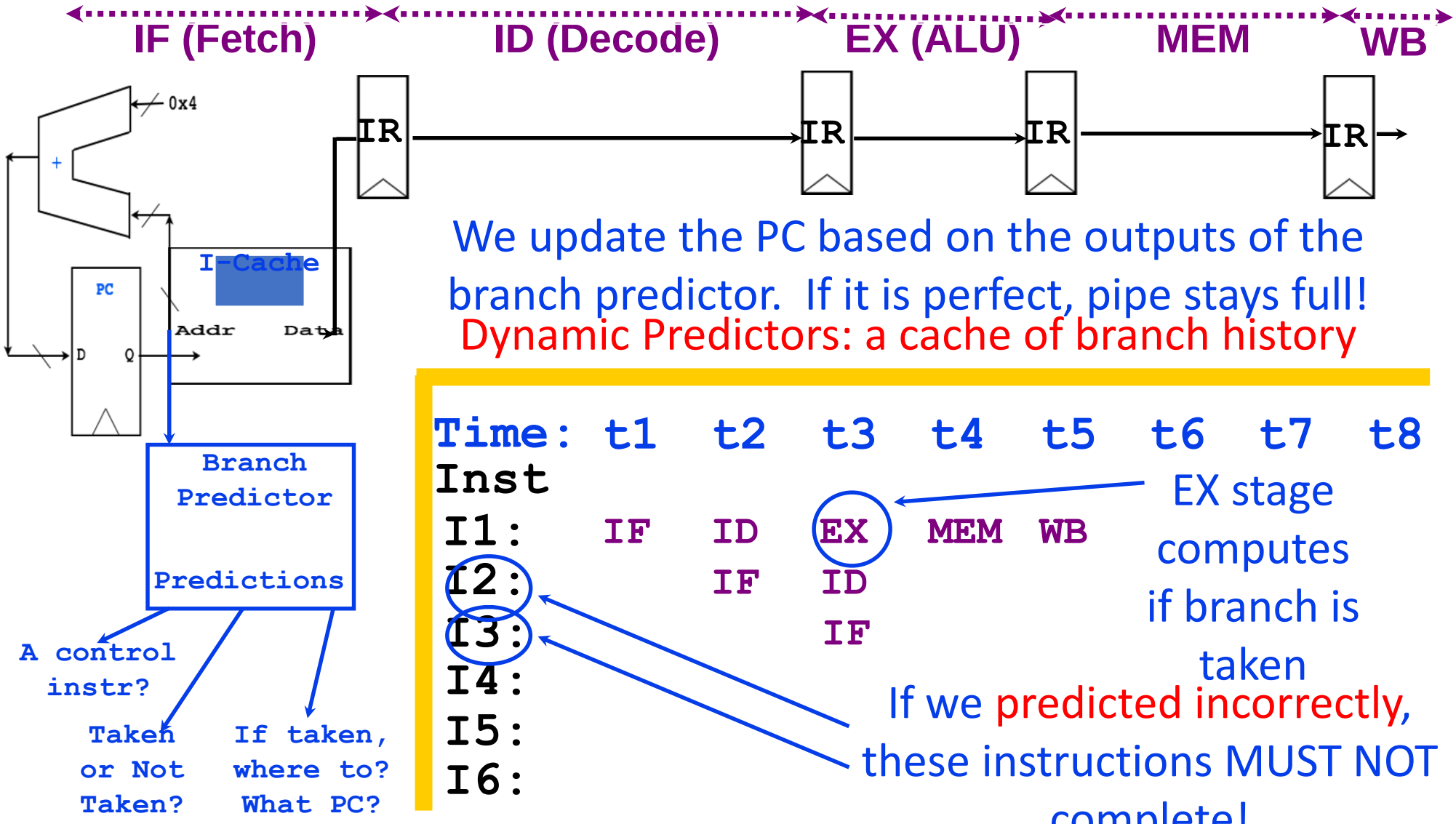
Static Branch Prediction



ISA can attach preferred direction semantics to branches,
e.g., Motorola MC88110
bne0 *(preferred taken)* beq0 *(not taken)*

ISA can allow arbitrary choice of statically predicted direction,
e.g., HP PA-RISC, Intel IA-64
typically reported as ~80% accurate

Dynamic Branch Predictor



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!

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

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

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 → 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

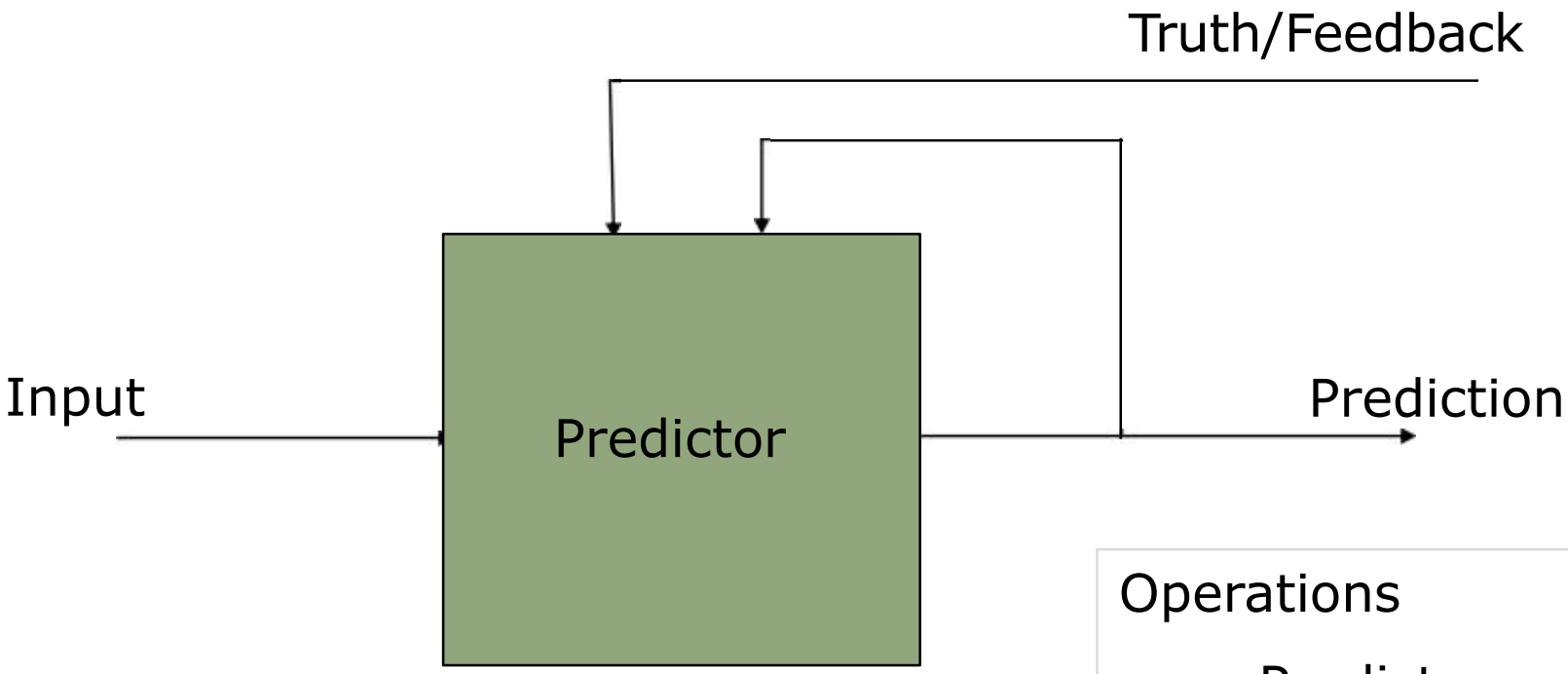
    + It can adapt to dynamic changes in branch behavior

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

- Disadvantages

  -- More complex (requires additional hardware)

Predictor as a Black Box



Operations

- Predict
- Update

Prediction as a feedback control process

Learning

Temporal correlation

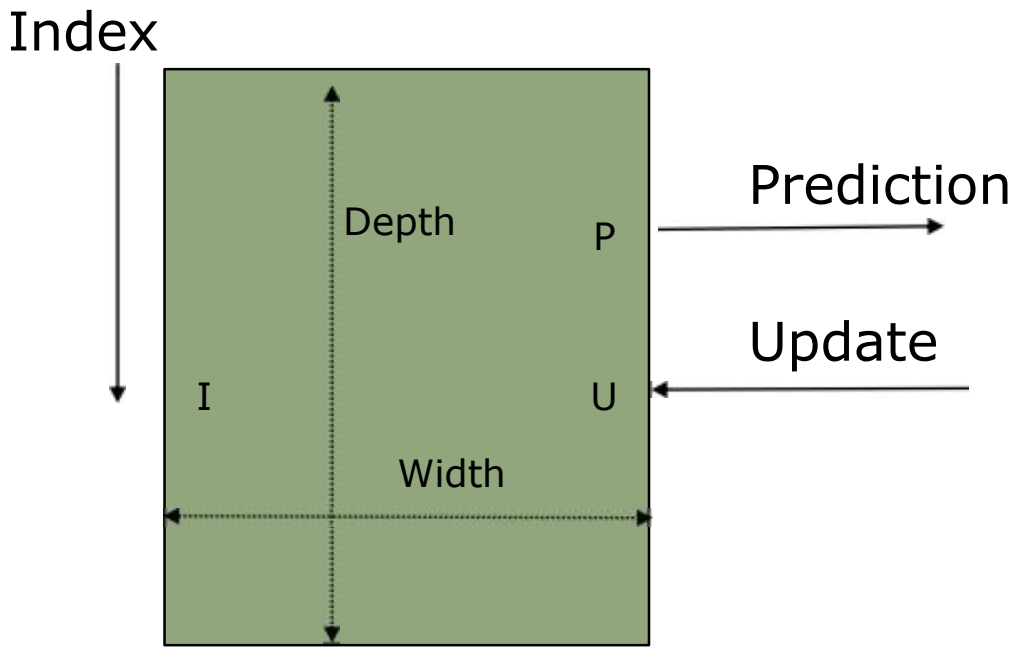
The way a branch resolves may be a good predictor of the way it will resolve at the next execution

Spatial correlation

Several branches may resolve in a highly correlated manner *(a preferred path of execution)*

Primitive

- Indexed table holding values
- Operations
  - Predict
  - Update



- Algebraic notation

$$\text{Prediction} = P[\text{Width}, \text{Depth}](\text{Index}; \text{Update})$$

Simplest One: Last-Time Predictor

- Last time predictor
  - Indicates which direction branch went last time it executed

    TTTTTTTTTTNNNNNNNNNN → 90% accuracy

- Always mis-predicts the last iteration and the first iteration of a loop branch
  - Accuracy for a loop with N iterations = (N-2)/N

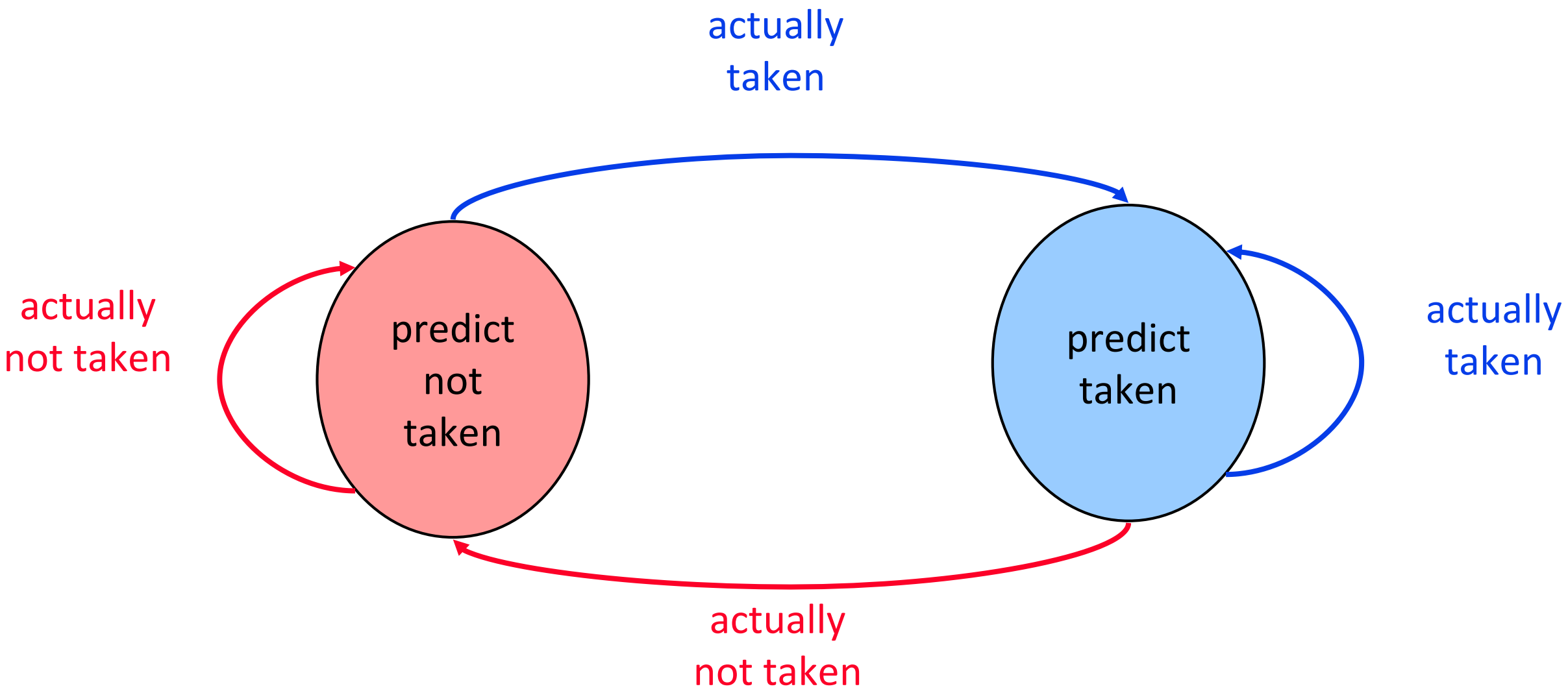
+ Loop branches for loops with large number of iterations

-- 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 20% instructions are branches and branch predictor has 85% accuracy)*

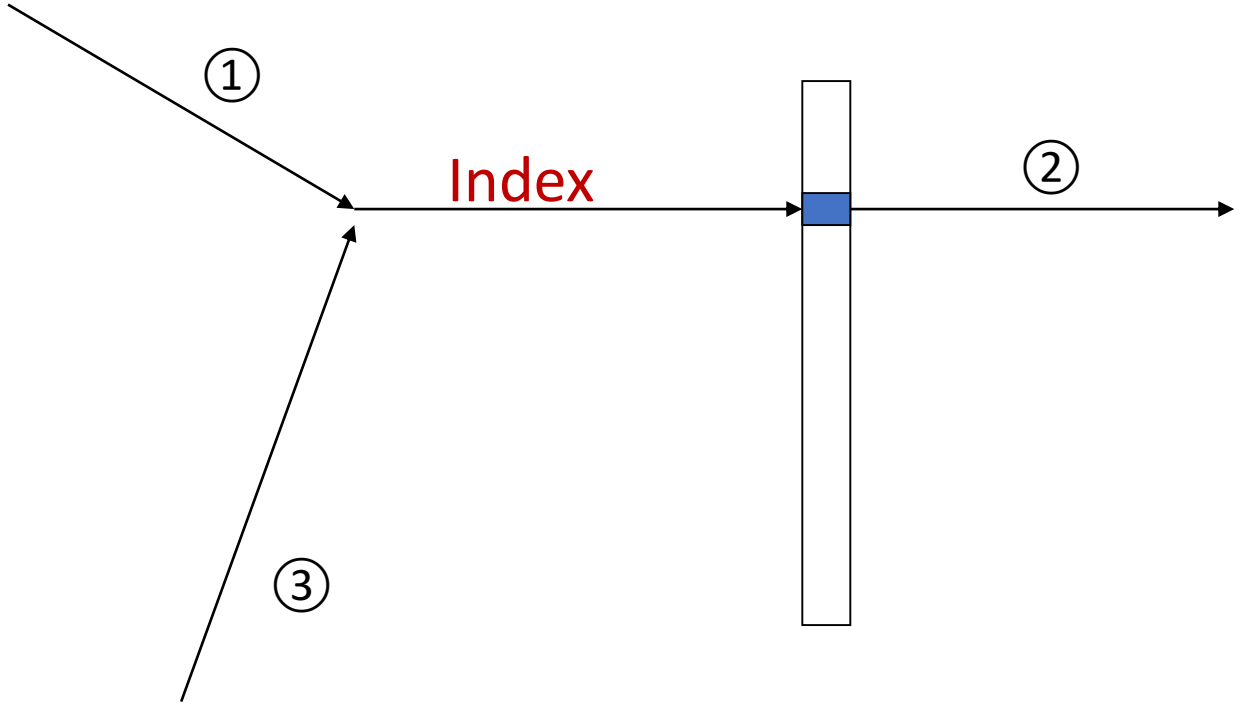
Last-Time


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

Last-time Predictor: The hardware

K bits of branch instruction address

Branch history table of 2^K entries, 1 bit per entry

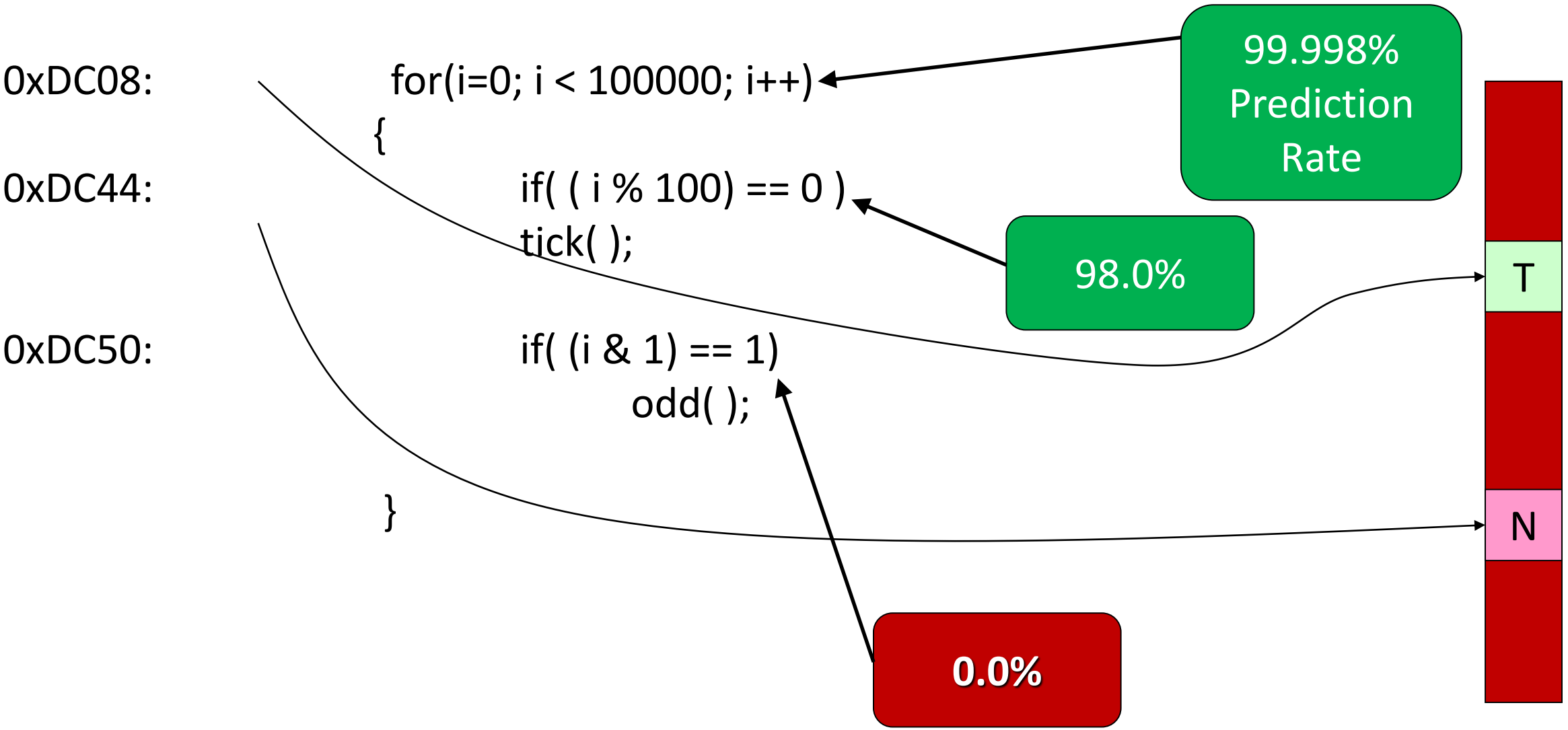


Use this entry to predict

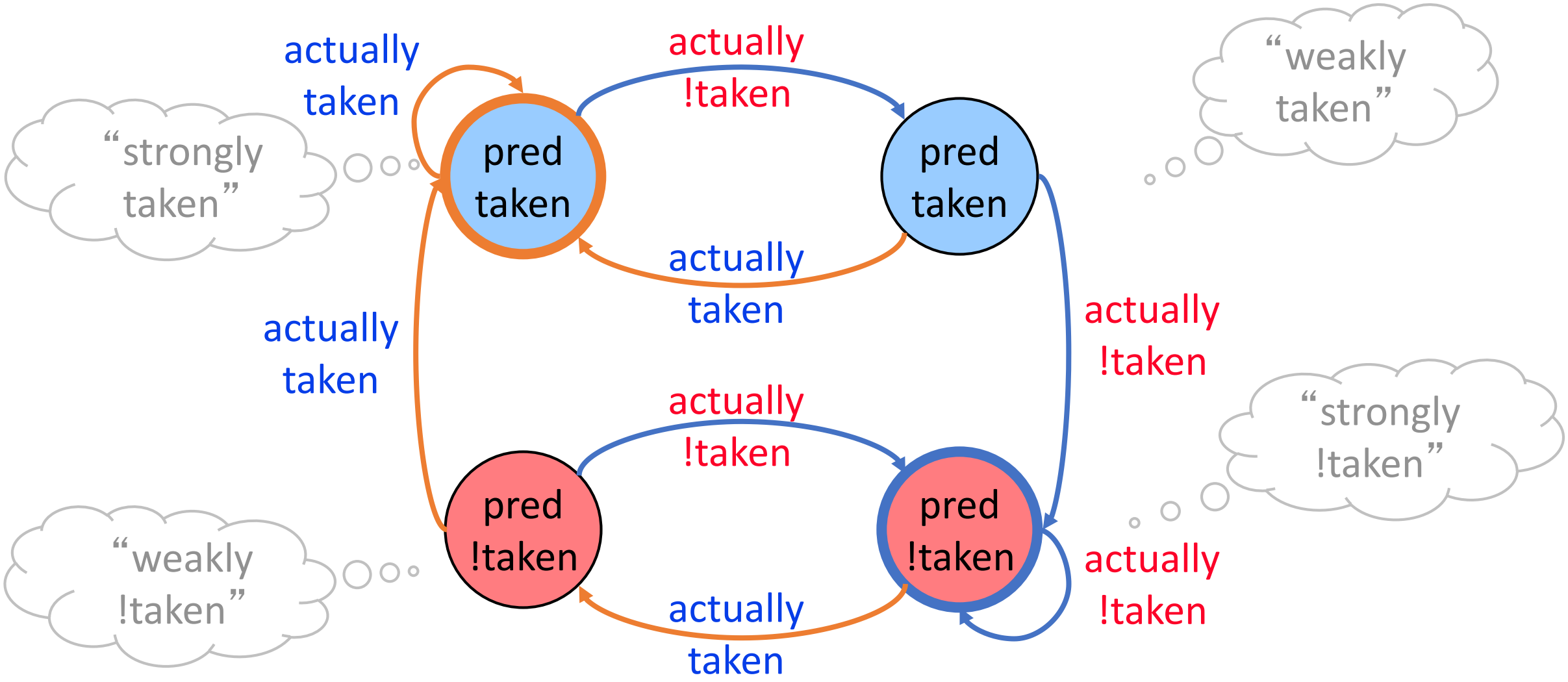
0: predict not taken
1: predict taken

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

Example: Predict!!



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?

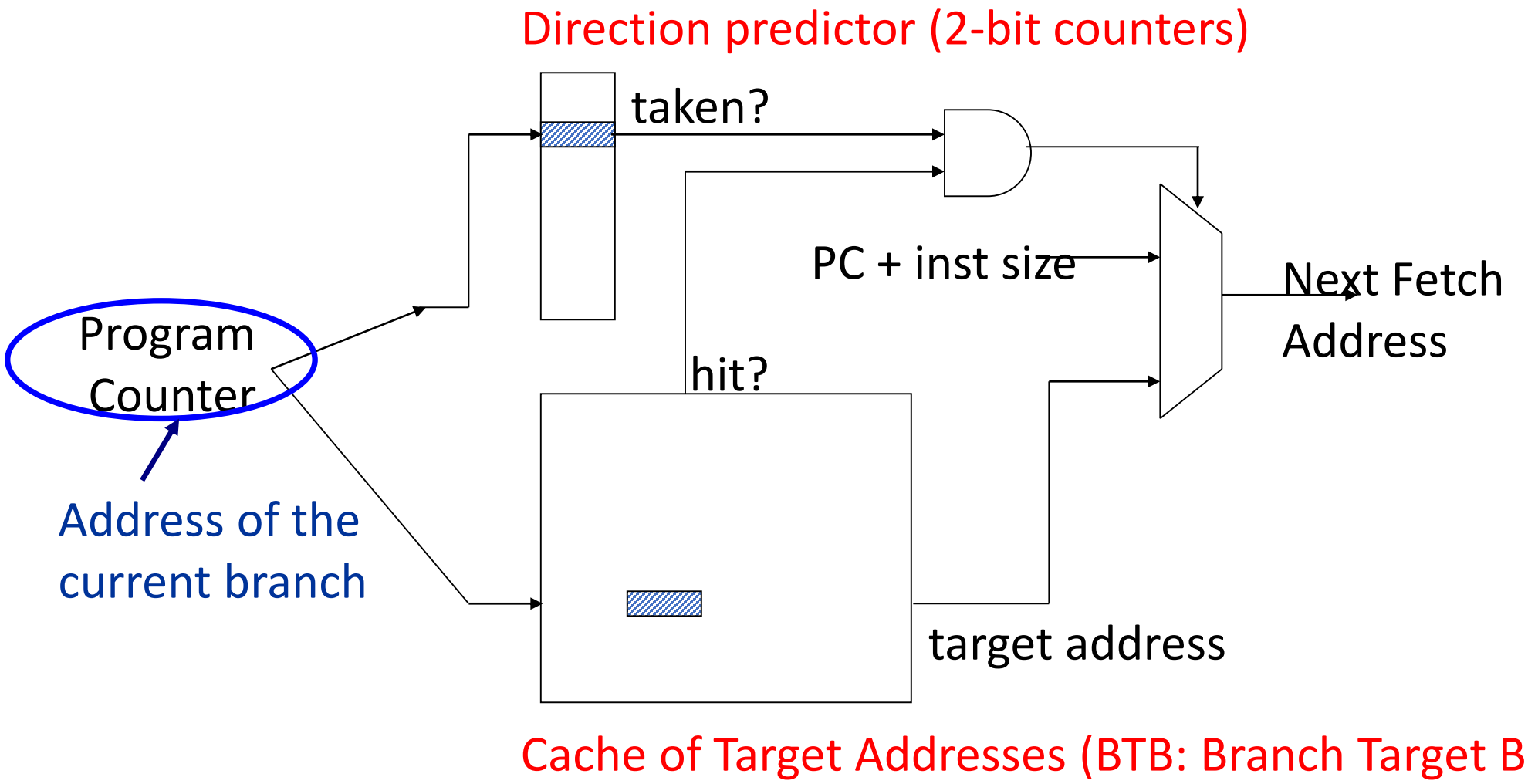
- Problem: Need to determine the **next fetch address** when the branch is fetched (to avoid a pipeline bubble)

Is This Enough?

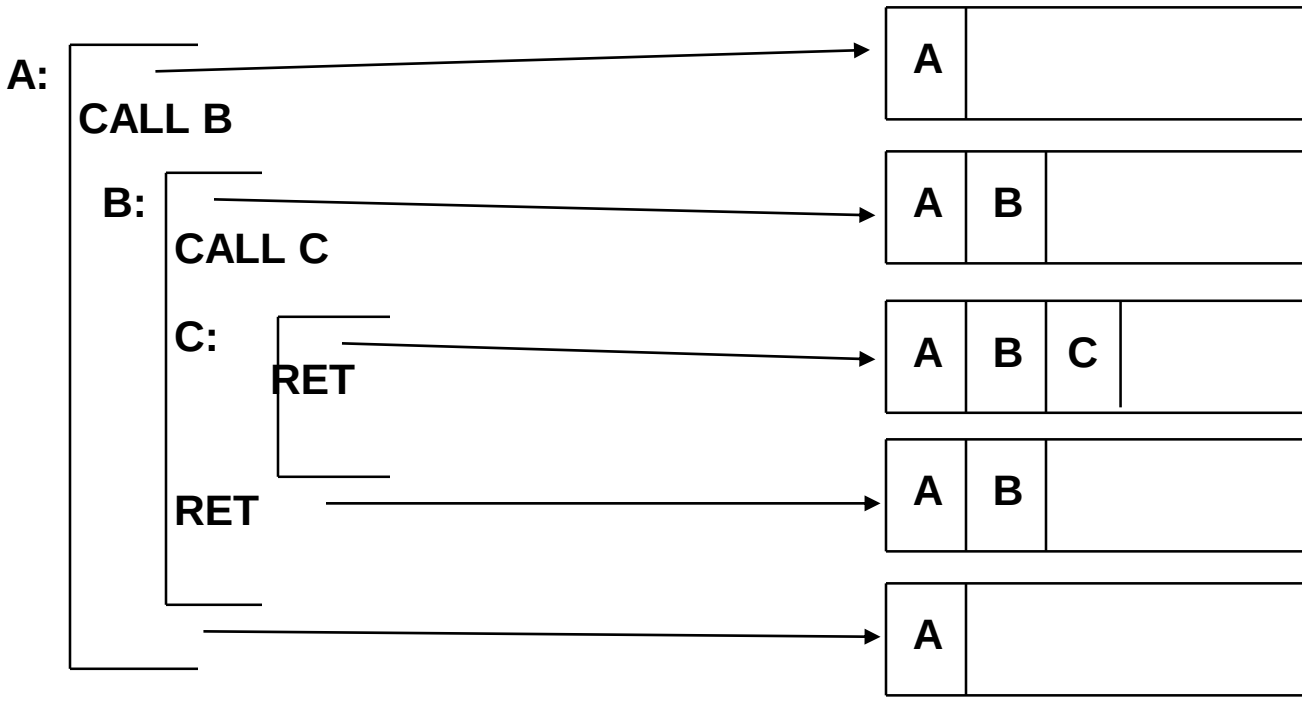
- Assume a pipeline with 20-cycle branch resolution latency

- How long does it take to fetch 100 5-instruction blocks (500 instructions)?
  - Assume 1 out of 5 instructions is a branch, fetch width of 5, each 5 instruction block ends in 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

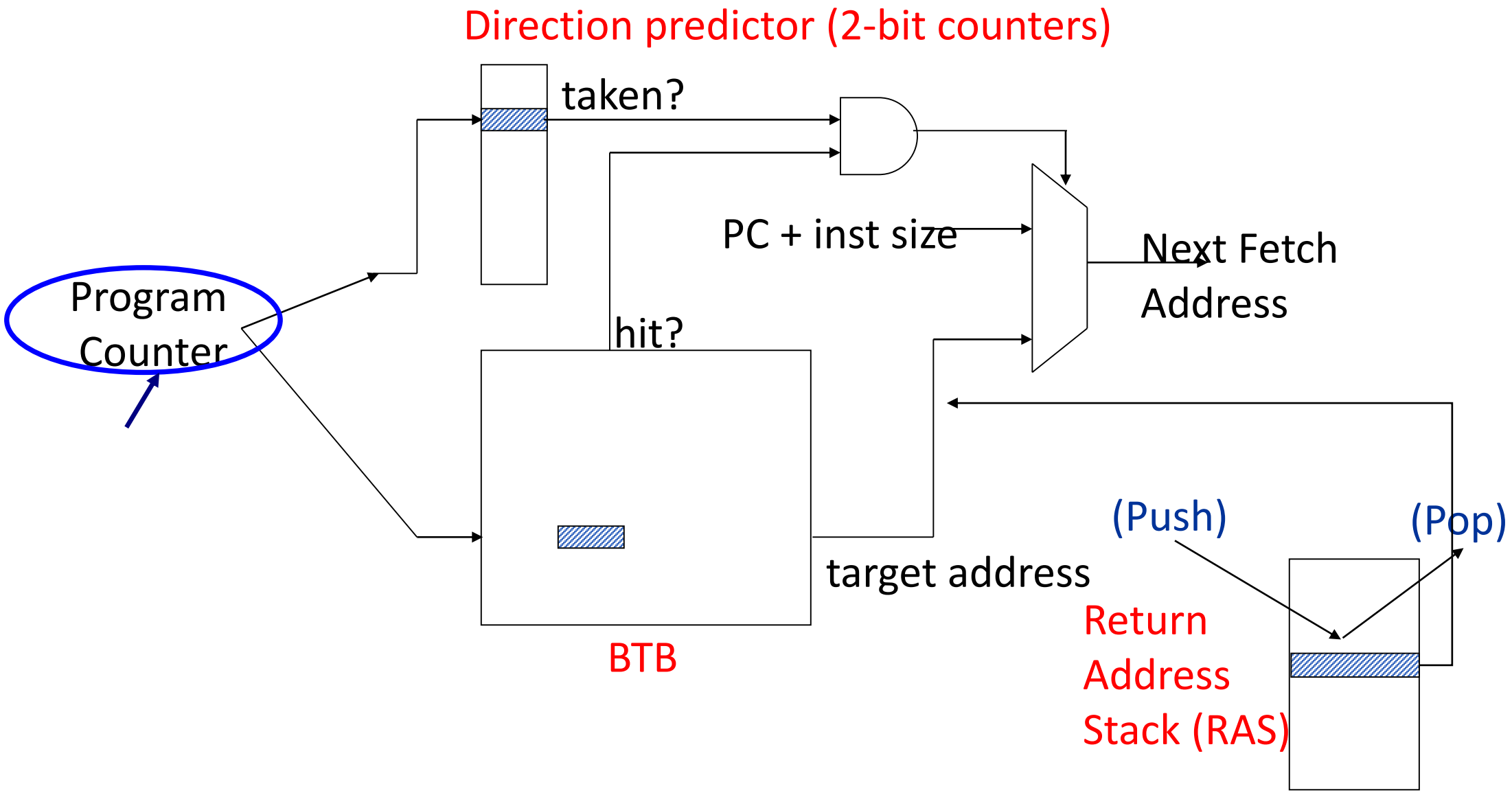
Fetch Stage with BTB and Direction Prediction



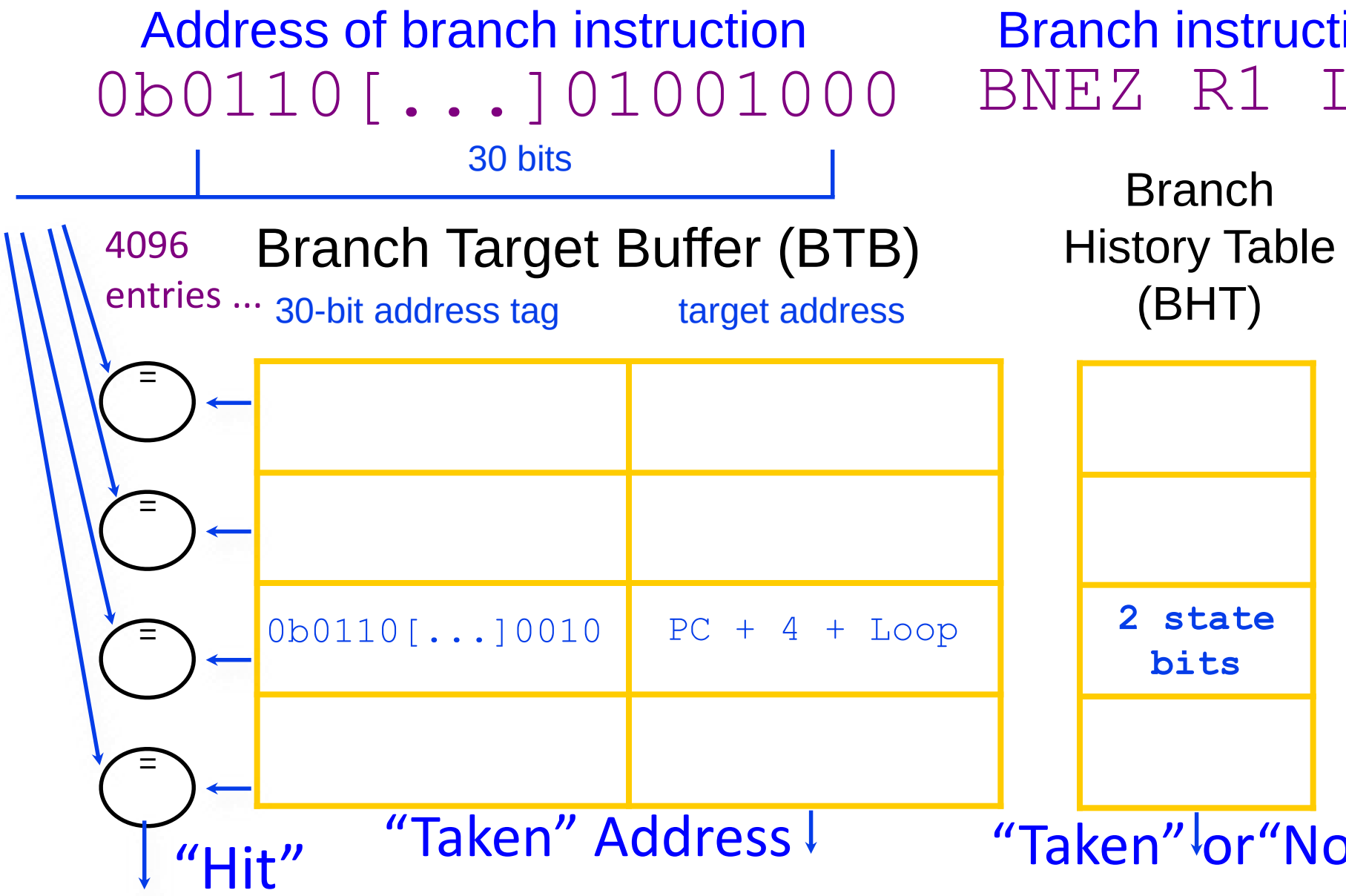
In Some Cases: No Need of BTB Access



Let's Revisit



BTB (Why 30-bit Tag?)

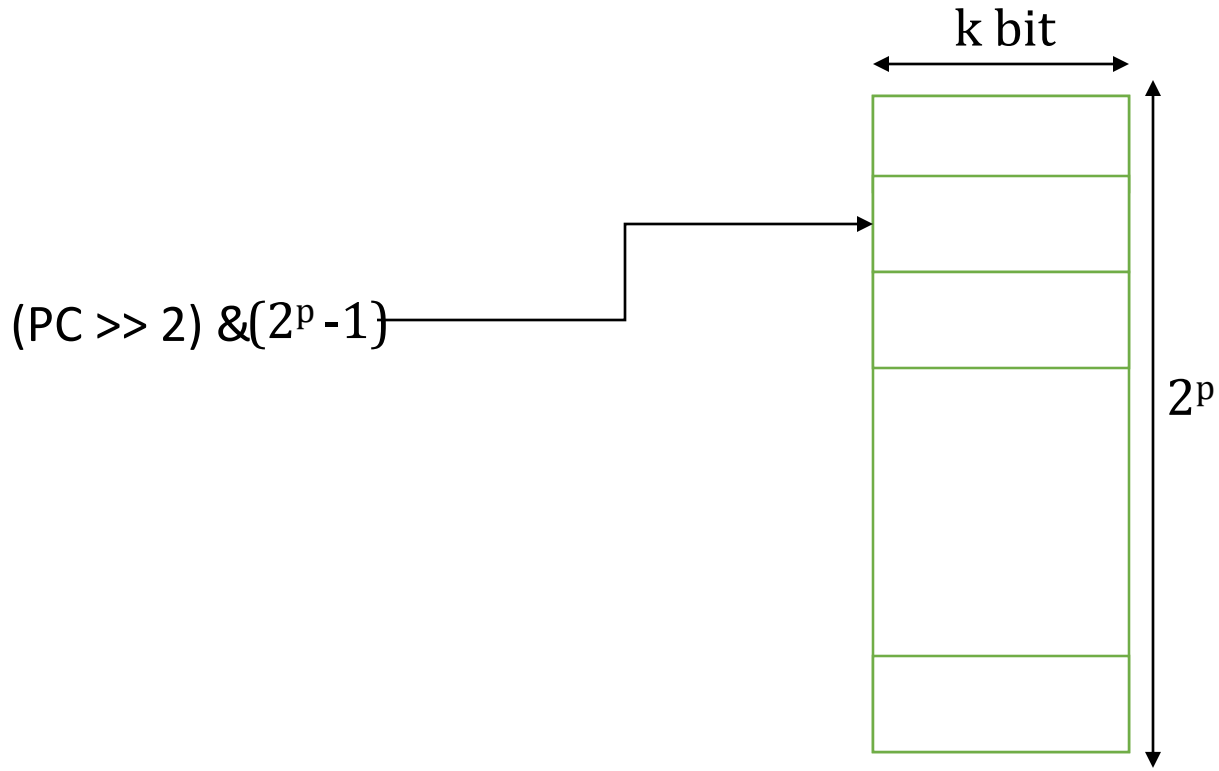


Drawn as fully associative to focus on the essentials.

In real designs, always direct-mapped.

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

No History based Branch Predictor



Bimodal predictor: Good for biased branches

Local History & Global History

- Local Behavior

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

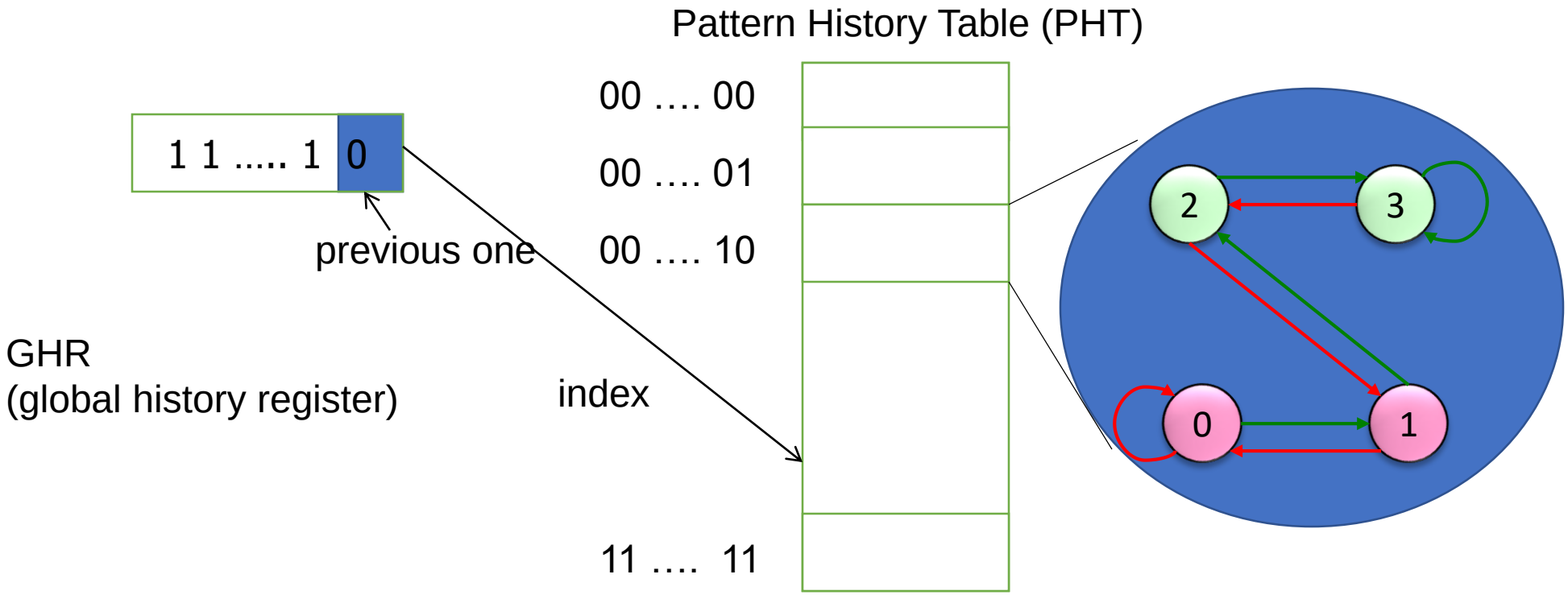
- Global Behavior

  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

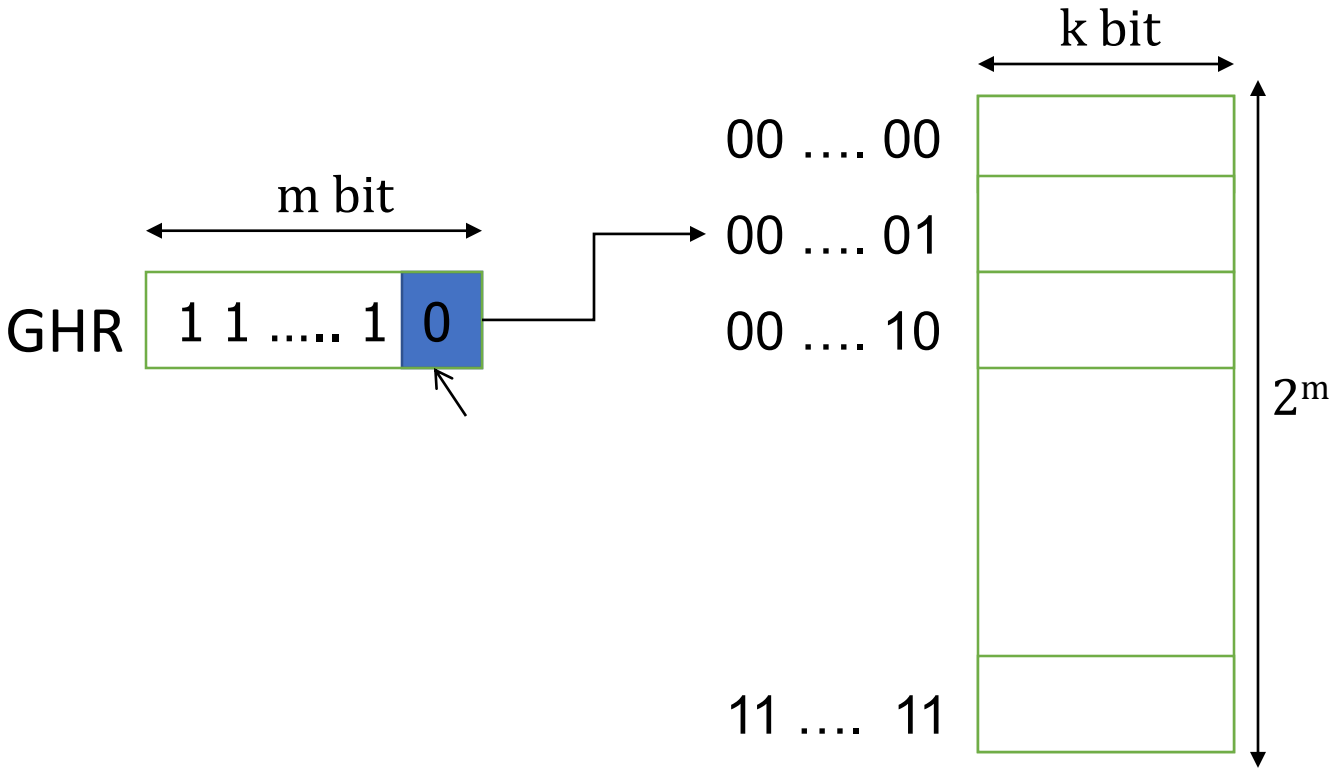
Two Level Global Branch Prediction [MICRO '91]

- 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

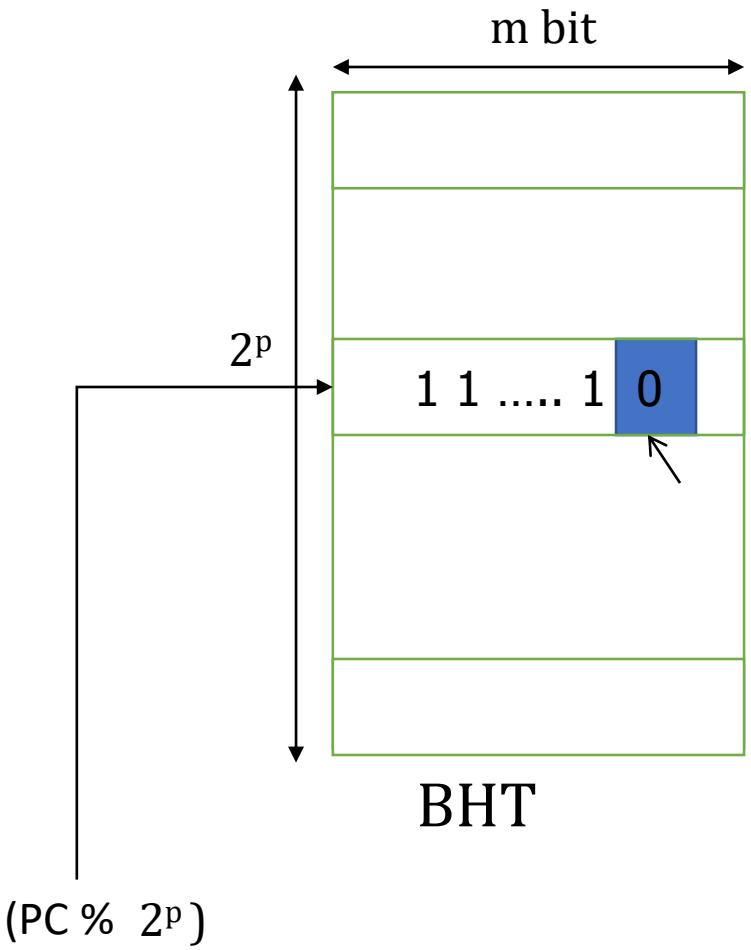


PHT

- Table of saturating counters



Set of Branches – One Register



Interference in Tables

- Sharing the PHTs between histories/branches leads to interference
  - Different branches map to the same PHT entry and modify it
  - Interference can be positive, negative, or neutral
- Interference can be eliminated by dedicating a PHT per branch

  -- Too much hardware cost
- How else can you eliminate

or reduce interference?

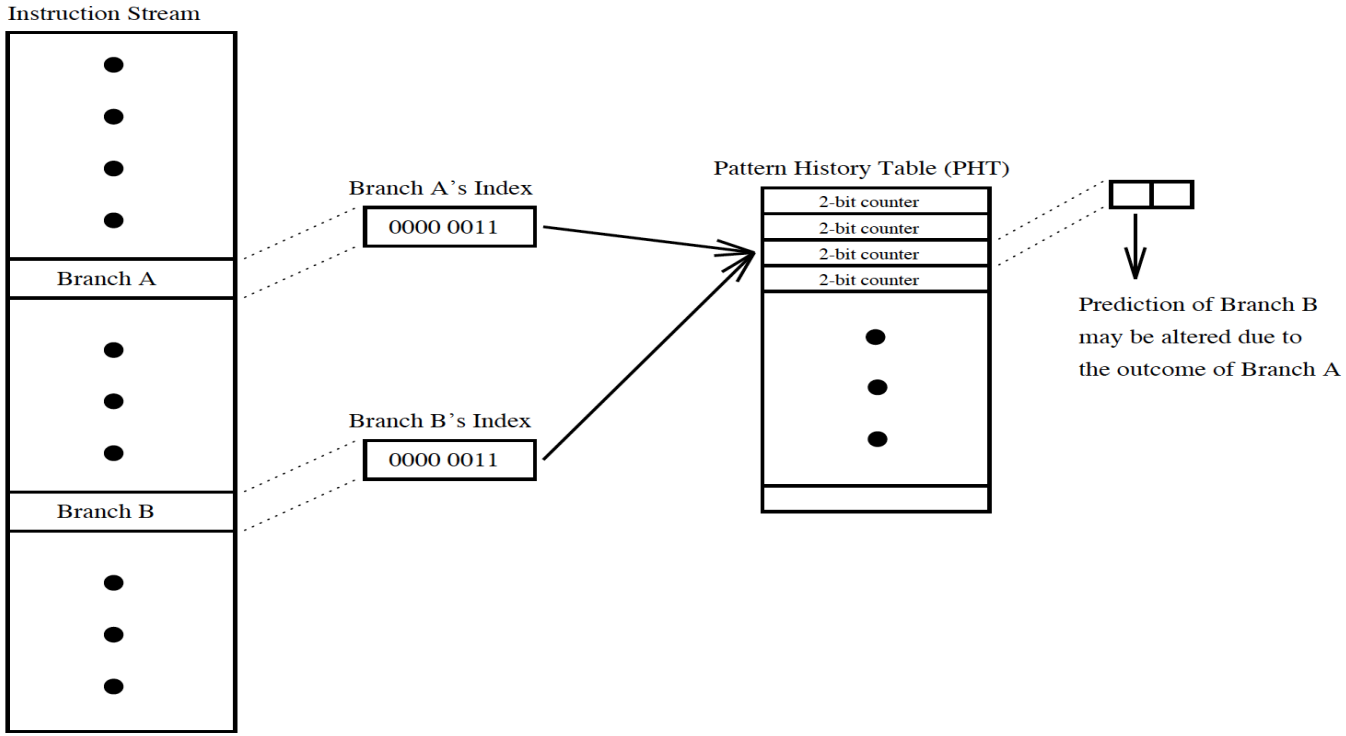
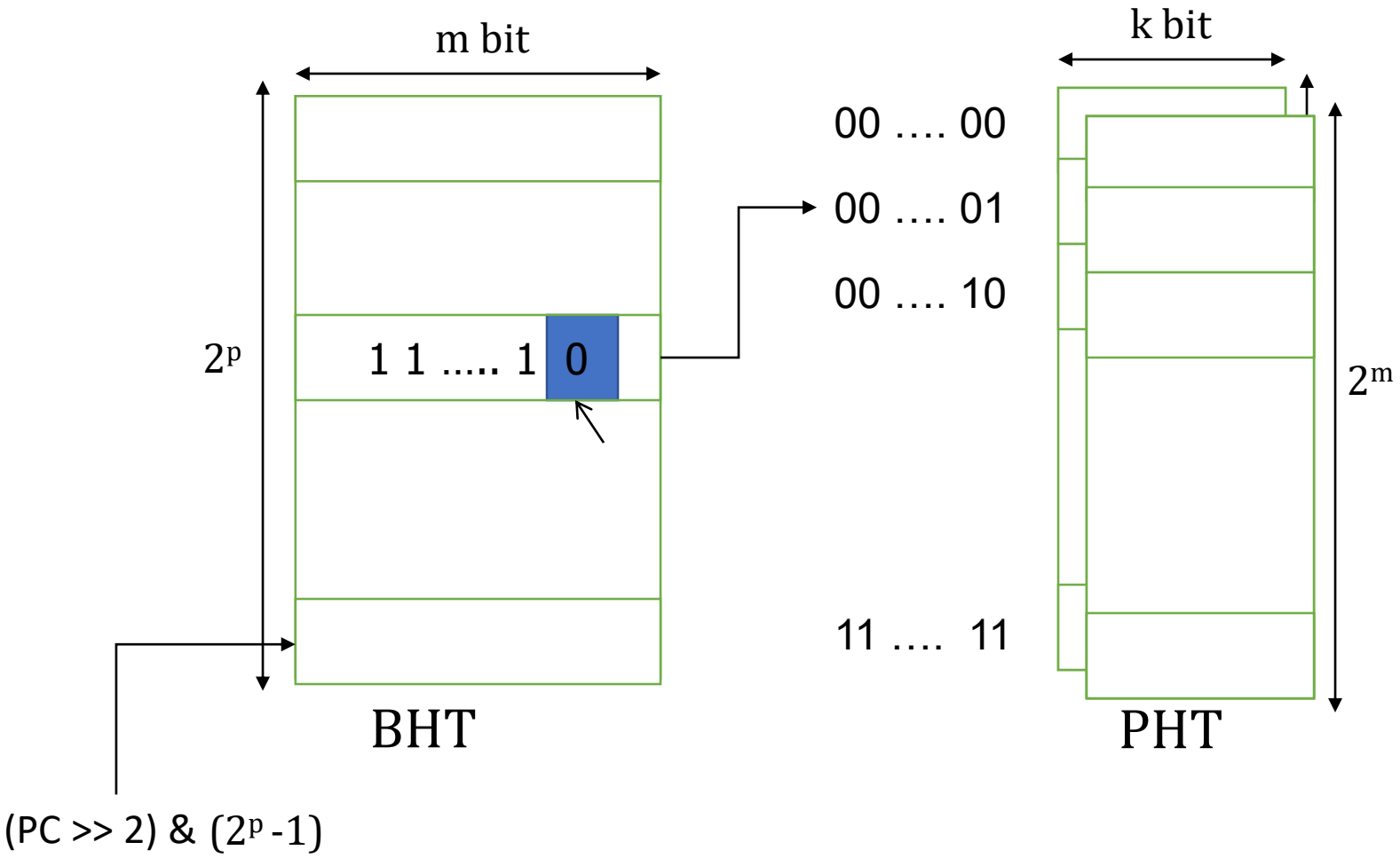
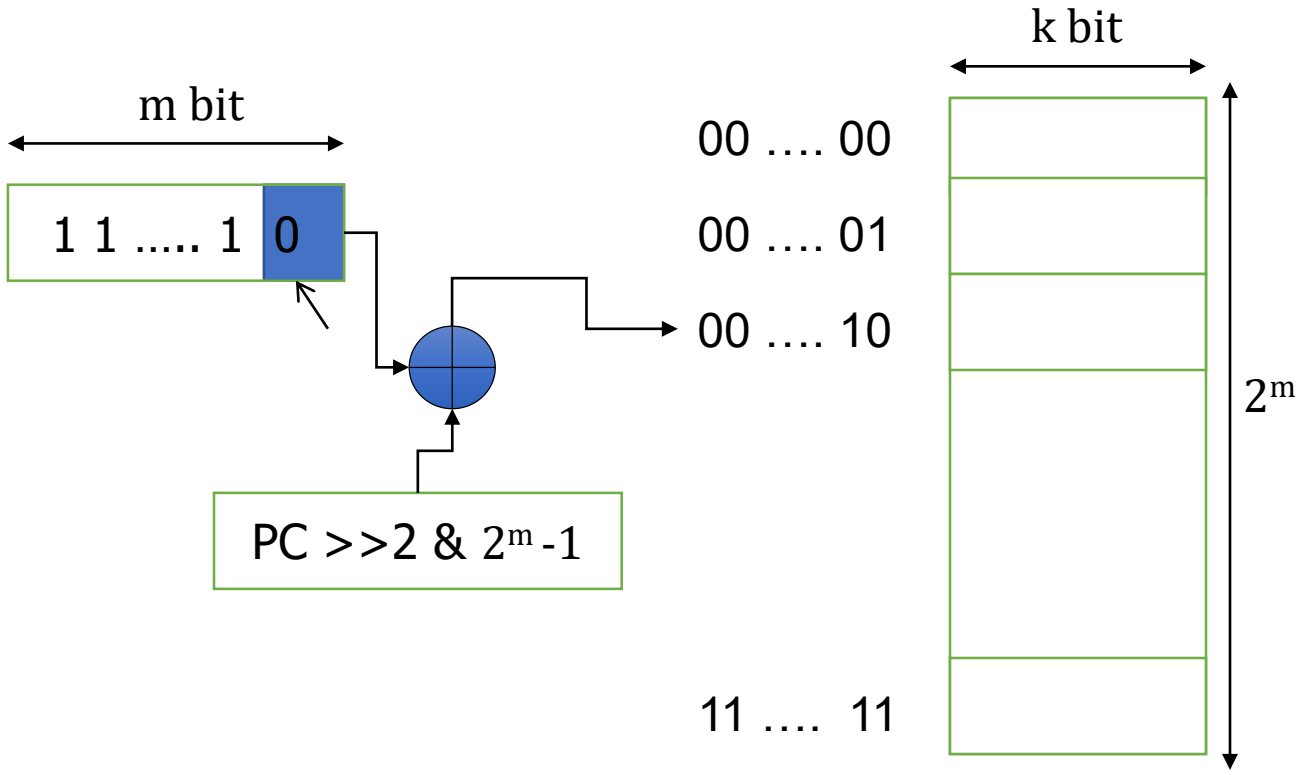


Figure 2: Interference in a two-level predictor.

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

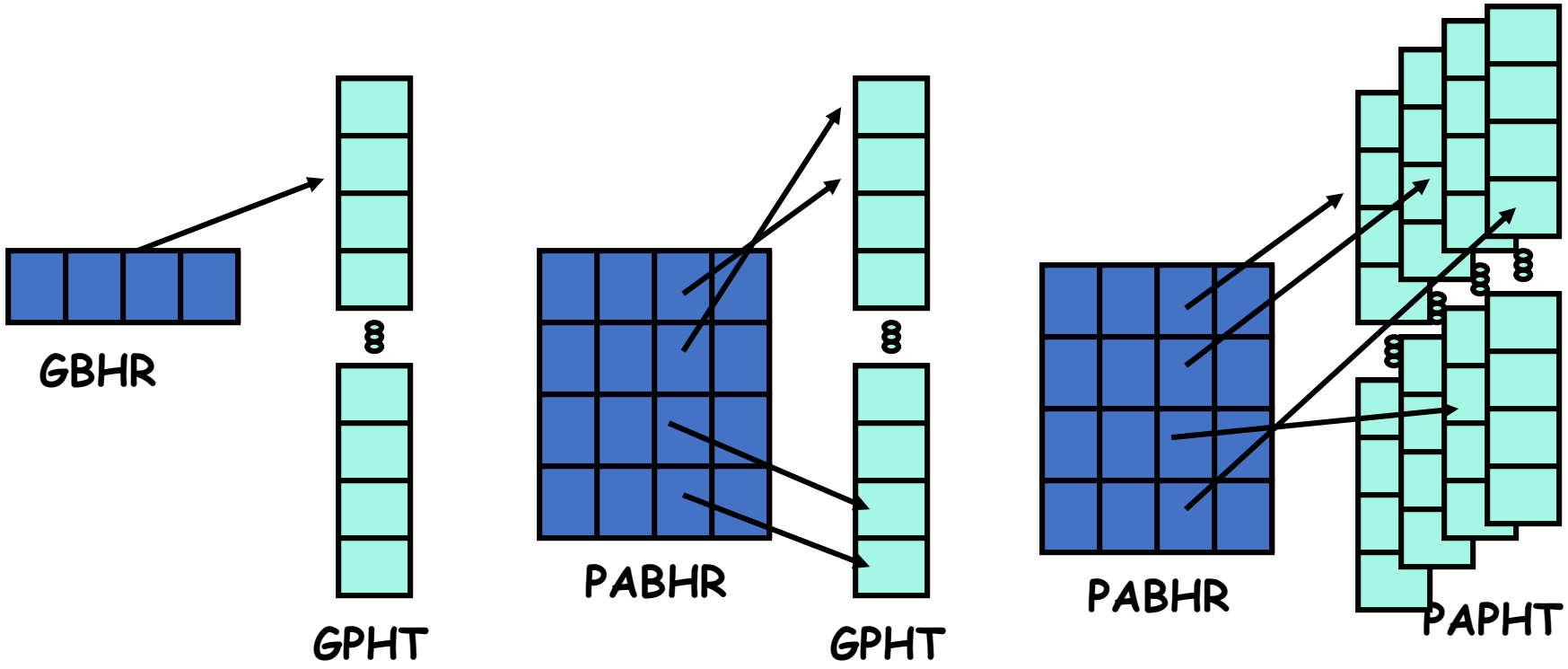


GShare



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

Y & P Classification [MICRO 91]



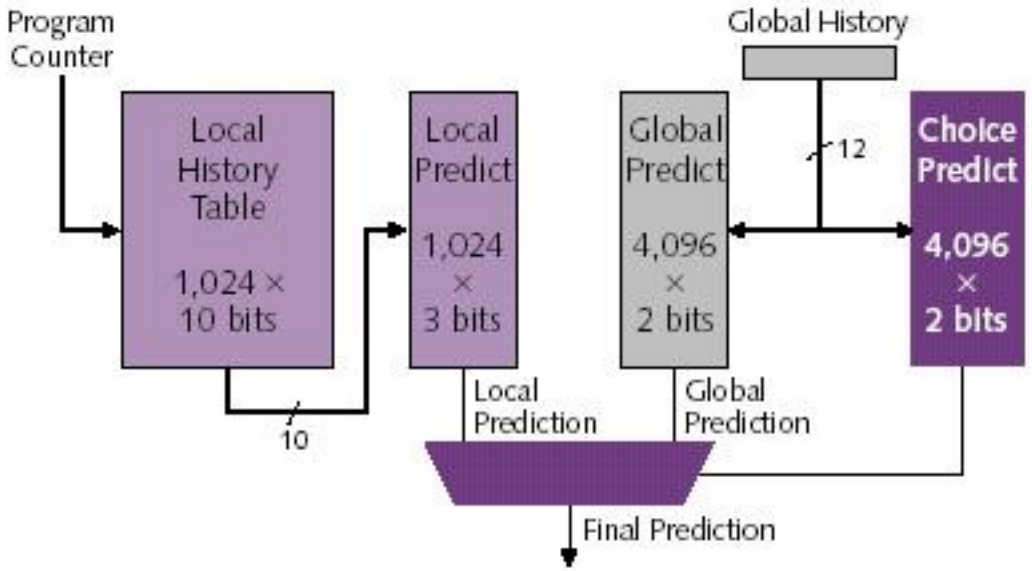
GAg

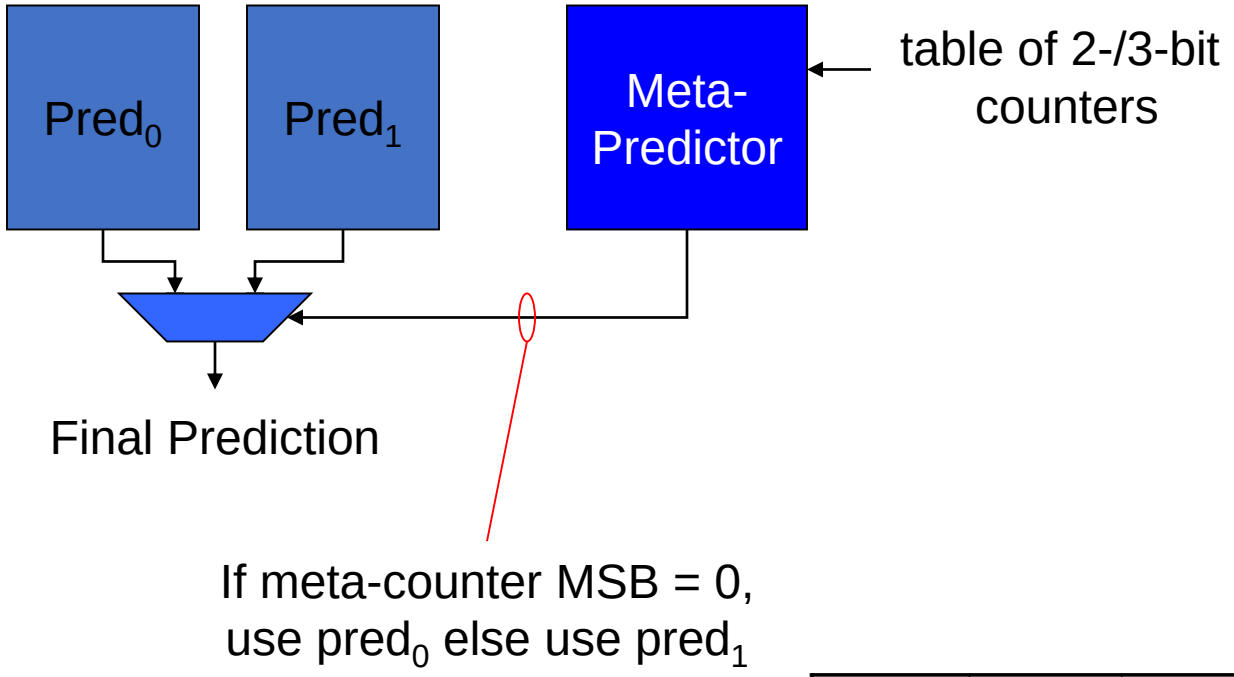
PAg (SAg?)

PAp

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

Tournament Predictor





If meta-counter MSB = 0,
use $pred_0$ else use $pred_1$

$Pred_0$	$Pred_1$	Meta Update
✗	✗	---
✗	✓	Inc
✓	✗	Dec
✓	✓	---

Some Other Predictors

- Loop branch detector and predictor
  - Loop iteration count detector/predictor
  - Works well for loops, where iteration count is predictable
  - Used in Intel Pentium M
- Perceptron branch predictor
  - Learns the *direction correlations* between individual branches
  - Assigns weights to correlations
  - Jimenez and Lin, "Dynamic Branch Prediction with Perceptrons," HPCA 2001.
- Hybrid history length based predictor
  - Uses different tables with different history lengths
  - Seznec, "Analysis of the O-Geometric History Length branch predictor," ISCA 2005.

Intel Pentium M Predictors

The advanced branch prediction in the Pentium M processor is based on the Intel Pentium® 4 processor's [6] branch predictor. On top of that, two additional predictors to capture special program flows, were added: a Loop Detector and an Indirect Branch Predictor.

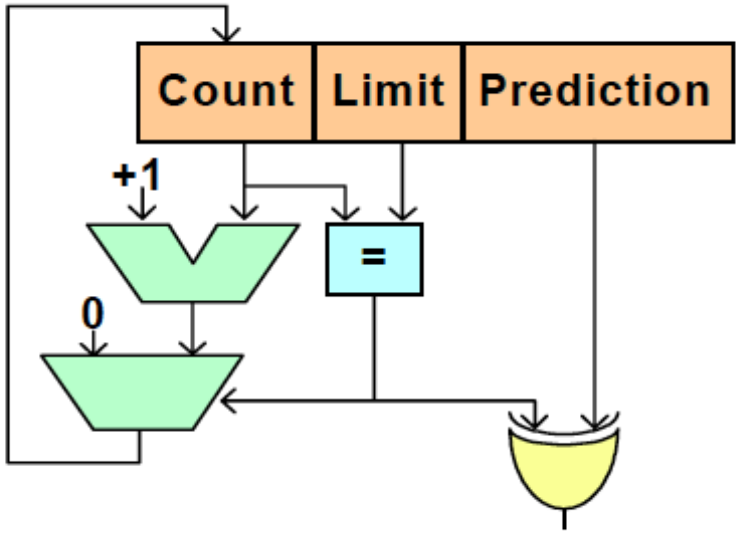


Figure 2: The Loop Detector logic

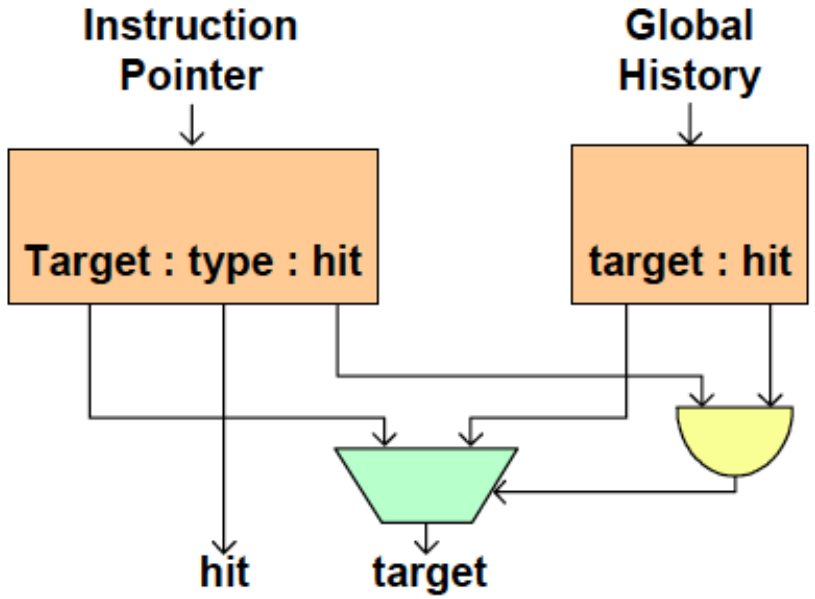


Figure 3: The Indirect Branch Predictor logic

Gochman et al.,
"The Intel Pentium M Processor: Microarchitecture and Performance,"
Intel Technology Journal, May 2003.

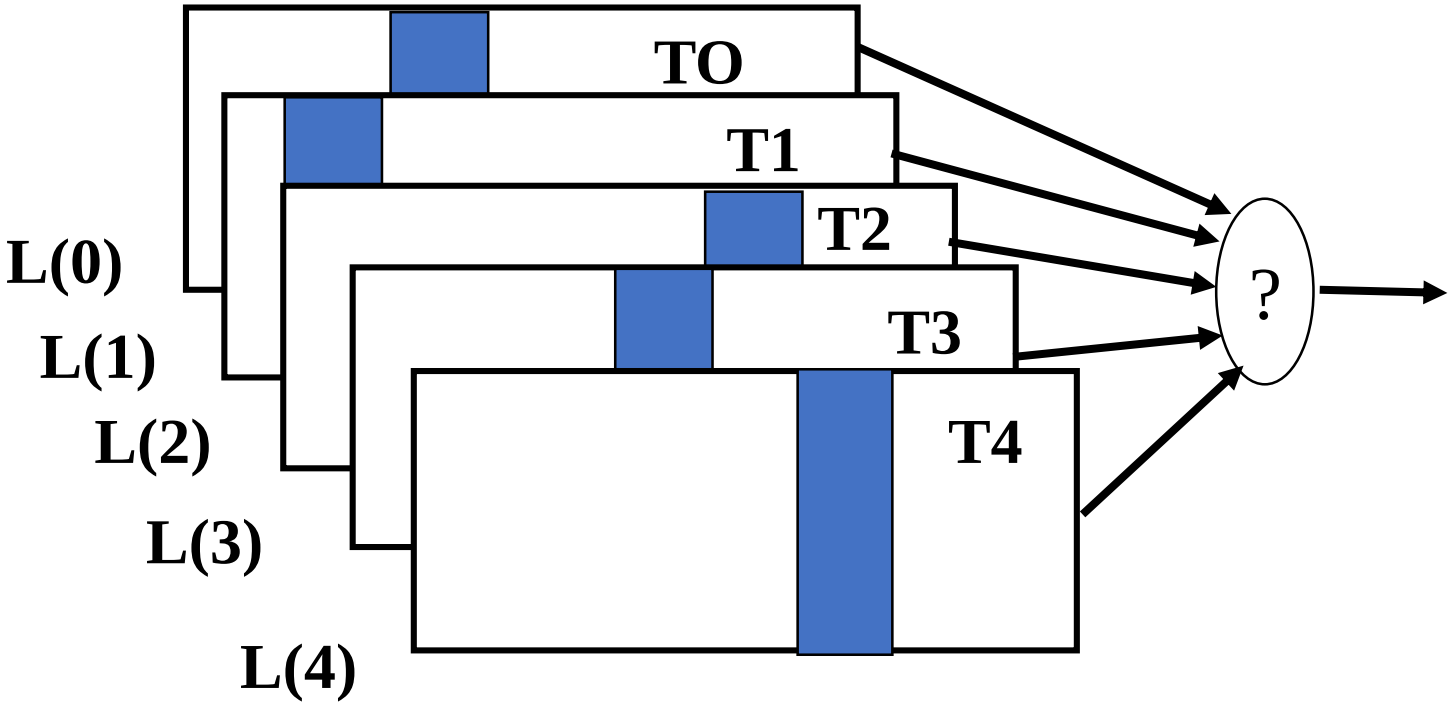
State-of-the-Art

State of the art: Neural vs. TAGE

1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1996: Vary history length
1998: Cache exceptions
2001: Neural predictor
2004: PPM
2006: TAGE
2016: Still TAGE vs Neural

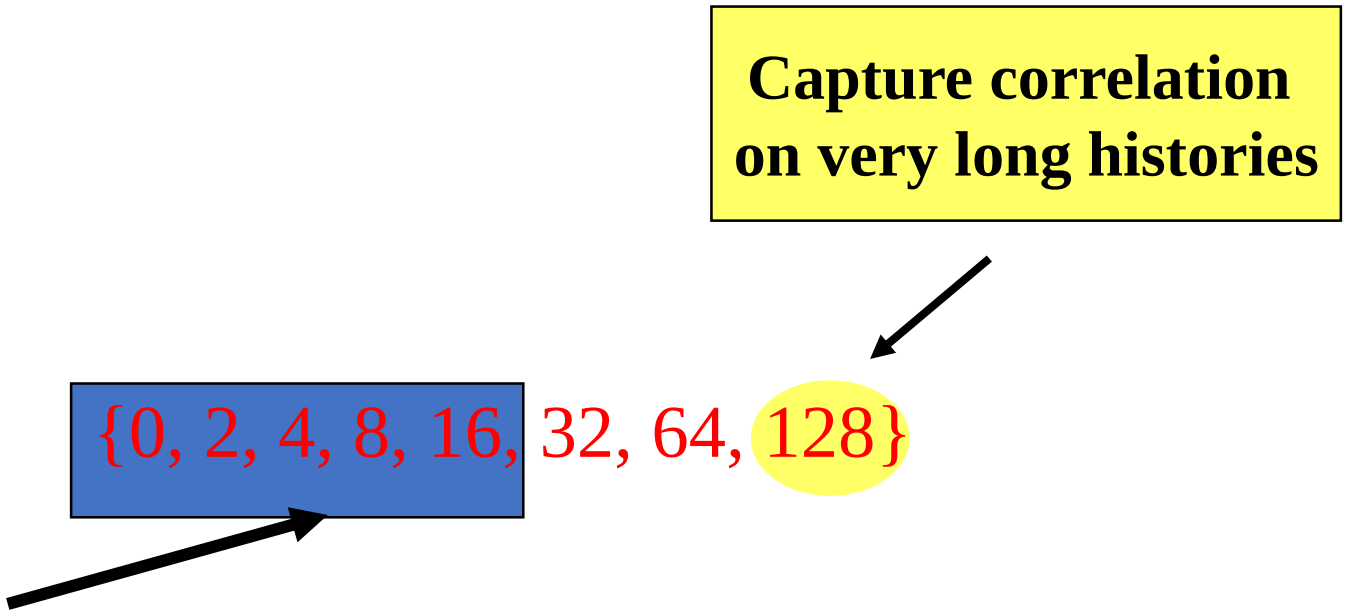
- Neural: AMD, Samsung
- TAGE: Intel?, ARM?
- Similarity
  - Many sources or “features”
- Key difference: how to combine them
  - TAGE: Override via partial match
  - Neural: integrate + threshold
- Every CBP is a cage match
  - Andre Seznec vs. Daniel Jimenez

The TAGE



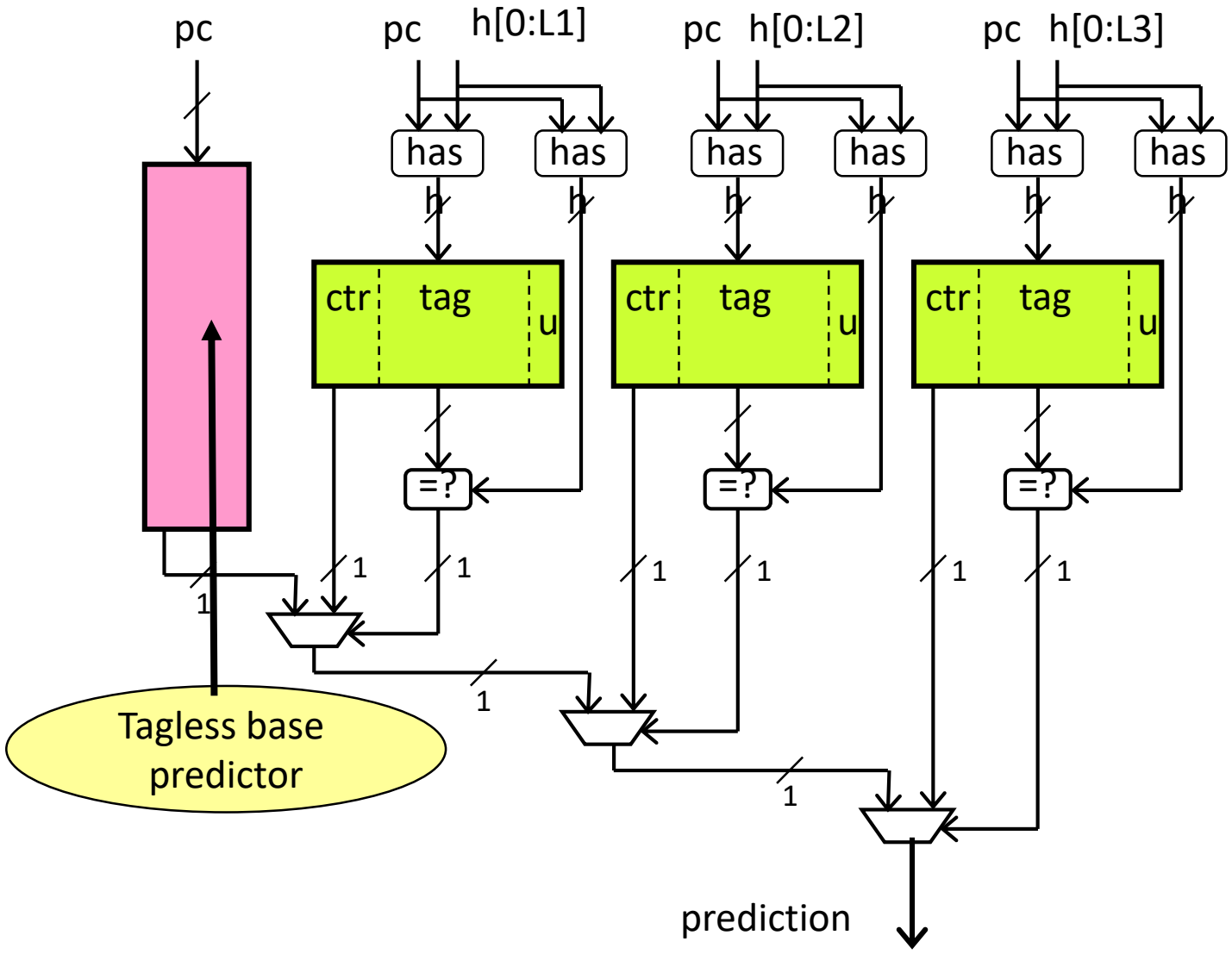
The TAGE

The set of history lengths forms a geometric series



What is important: L(i)-L(i-1) is drastically increasing

Micro-architecture



Moinuddin Qureshi @mointweets · Aug 29, 2019

Replying to @RunBaconHockey

Will do -- I meant that after his paper was published, we almost stopped publishing Branch Prediction papers at top tier conferences. Yes, I agree it is a very important topic (and hence I chaired CBP-2014).

Impact

- *Seznec's TAGE paper from 2006 had a huge impact on processor designs but it also resulted in raising the bar so high that the number of publications in this area reduced significantly*

http://www.irisa.fr/alf/downloads/seznec/IntelResearchImpactMedal.pdf