Branch Prediction

Control Dependence

- Question: What should the fetch PC be in the next cycle?
- If the instruction that is fetched is a control-flow instruction:
  - How do we determine the next Fetch PC?
- In fact, how do we even know whether or not the fetched instruction is a control-flow instruction?

How to Handle Control Dependences

- **Critical to keep the pipeline full with correct sequence of dynamic instructions.**

- Potential solutions if the instruction is a control-flow instruction:

- Stall the pipeline until we know the next fetch address
- Guess the next fetch address (branch prediction)
- Employ delayed branching (branch delay slot)
- Do something else (fine-grained multithreading)
- Eliminate control-flow instructions (predicated execution)
- Fetch from both possible paths (if you know the addresses of both possible paths) (multipath execution)

The Branch Problem

- **Control flow instructions (branches) are frequent**
  - **In a program, 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

- If we are fetching W instructions per cycle (i.e., if the pipeline is W wide)
  - A branch mis-prediction leads to **N x W** wasted instruction slots

Importance of The Branch Problem

- Assume N = 20 (20 pipe stages), W = 5 (5 wide fetch)
- Assume: 1 out of 5 instructions is a branch
- Assume: Each 5 instruction-block ends with a branch
- Assume: Branch direction is determined at the final pipe stage.

- How long does it take to fetch 500 instructions?
  - 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

Simplest: Always Guess NextPC = PC + 4

- Always predict the next sequential instruction is the next instruction to be executed
- This is a form of next fetch address prediction (and branch prediction)

- How can you make this more effective?

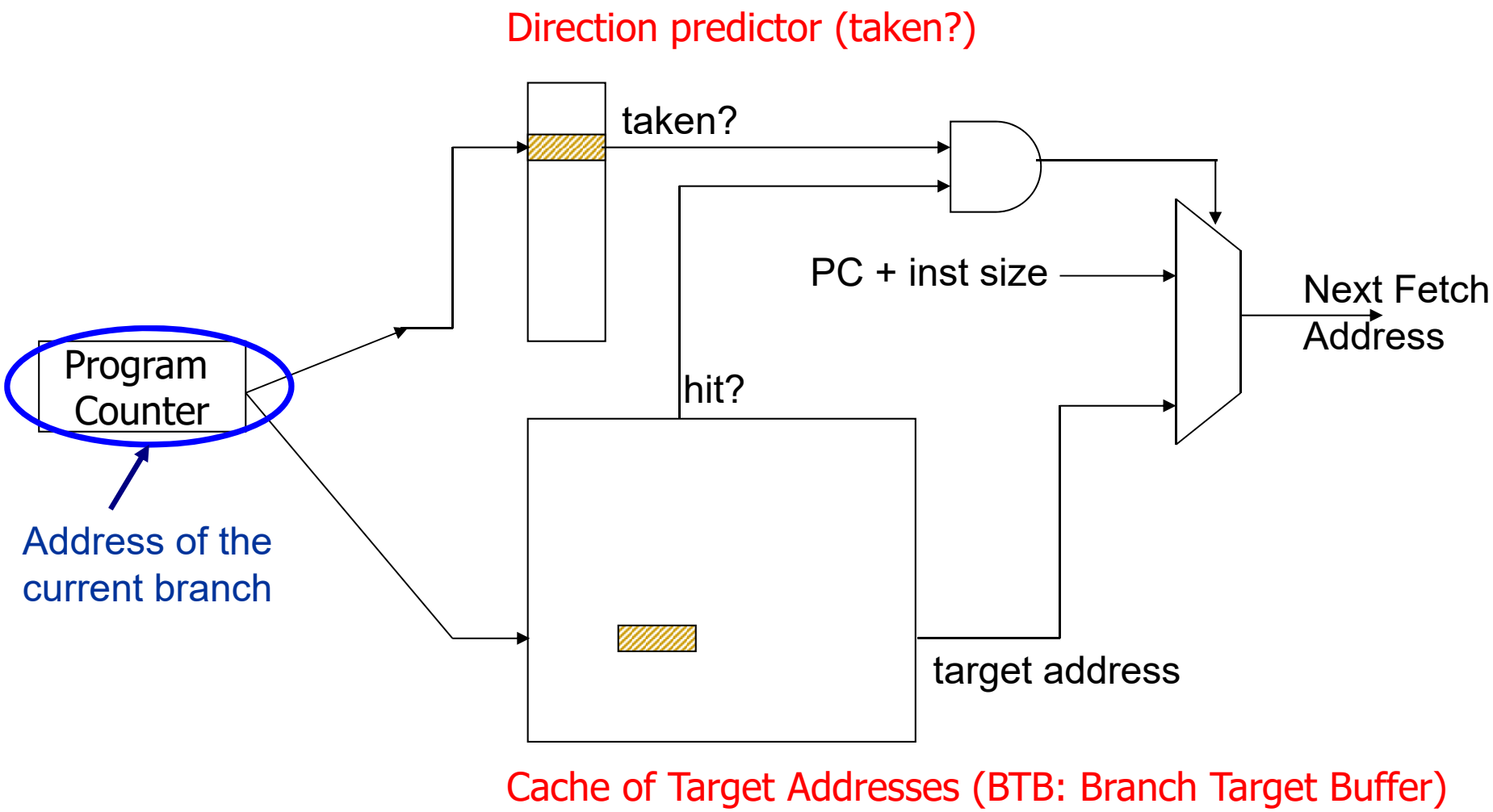
Branch Prediction (A Bit More Enhanced)

- 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



Three Things to Be Predicted

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

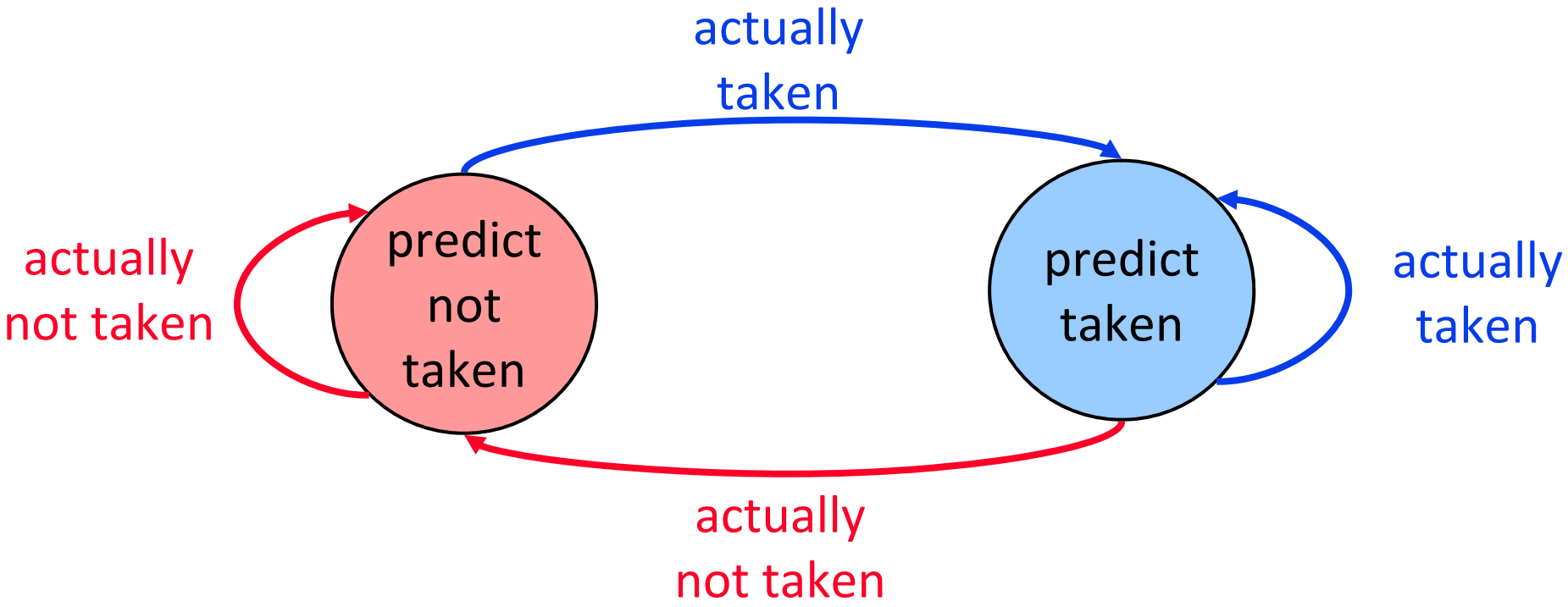
- Third (3.) can be accomplished using a BTB
  - Remember target address computed last time branch was executed
- First (1.) can be accomplished using a BTB
  - If BTB provides a target address for the program counter, then it must be a branch
  - Or, we can store "branch metadata" bits in instruction cache/memory → partially decoded instruction stored in I-cache
- **Second (2.): How do we predict the direction?**

Direction Prediction

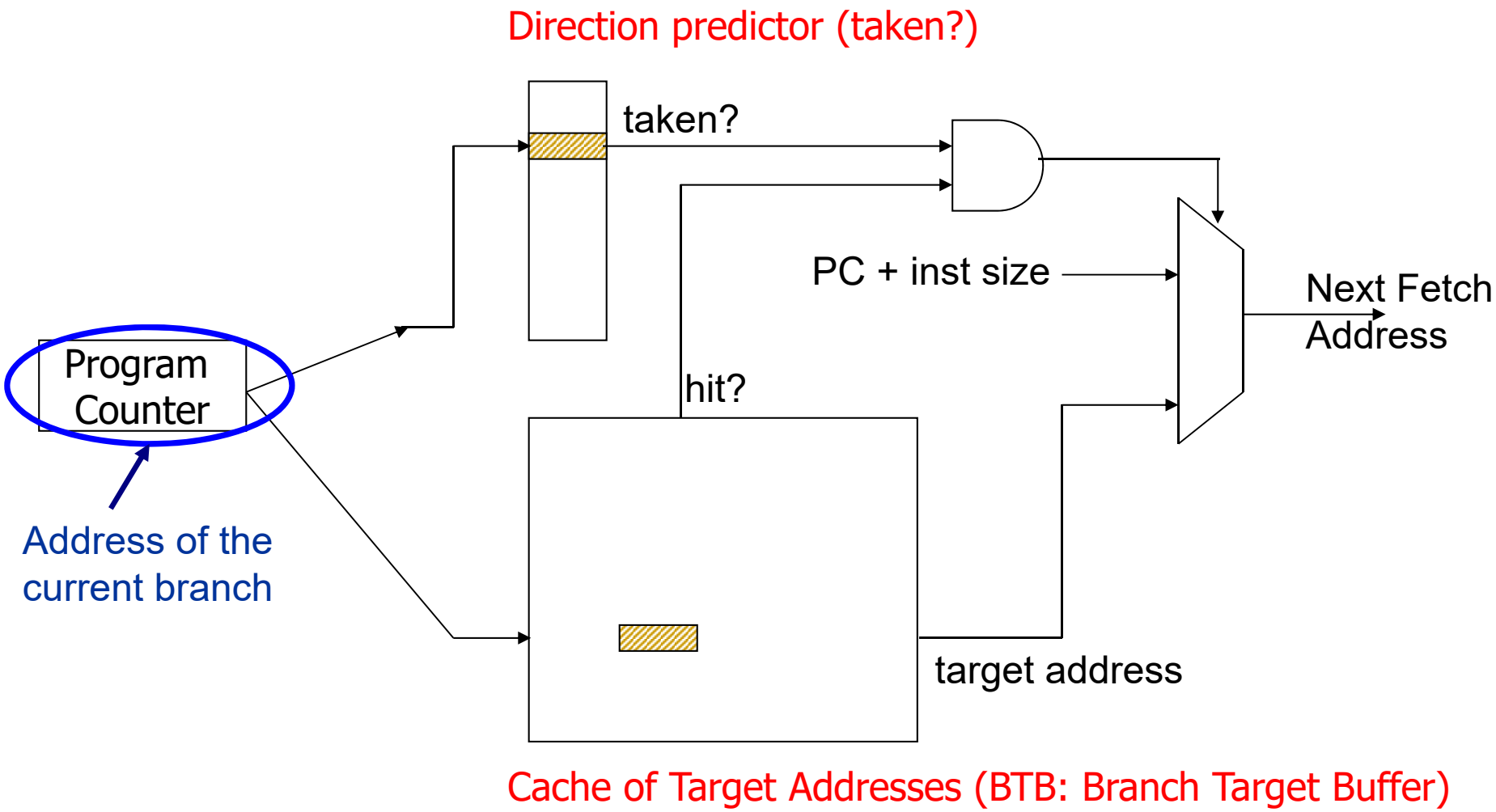
- Compile time (static)
  - Always not taken
  - Always taken
  - BTFN (Backward taken, forward not taken)
  - Profile based (likely direction)
  - Program analysis based (likely direction)

- Run time (dynamic)
  - Last time prediction (single-bit)
  - Two-bit counter based prediction
  - Two-level prediction (global vs. local)
  - Hybrid

Review: State Machine for Last-Time Prediction



Fetch Stage with BTB and Direction Prediction



Review: Improving the Last Time Predictor

- Problem: A last-time predictor changes its prediction from T→NT or NT→T too quickly
  - even though the branch may be mostly taken or mostly not taken

- Solution Idea: Add **hysteresis** to the predictor so that prediction does not change on a single different outcome
  - Use two bits to track the history of predictions for a branch instead of a single bit
  - Can have 2 states for T or NT instead of 1 state for each

- Smith, "A Study of Branch Prediction Strategies," ISCA 1981.

Review: Two-Bit Counter Based Prediction

- Each branch associated with a two-bit counter
- One more bit provides hysteresis
- A strong prediction does not change with one single different outcome

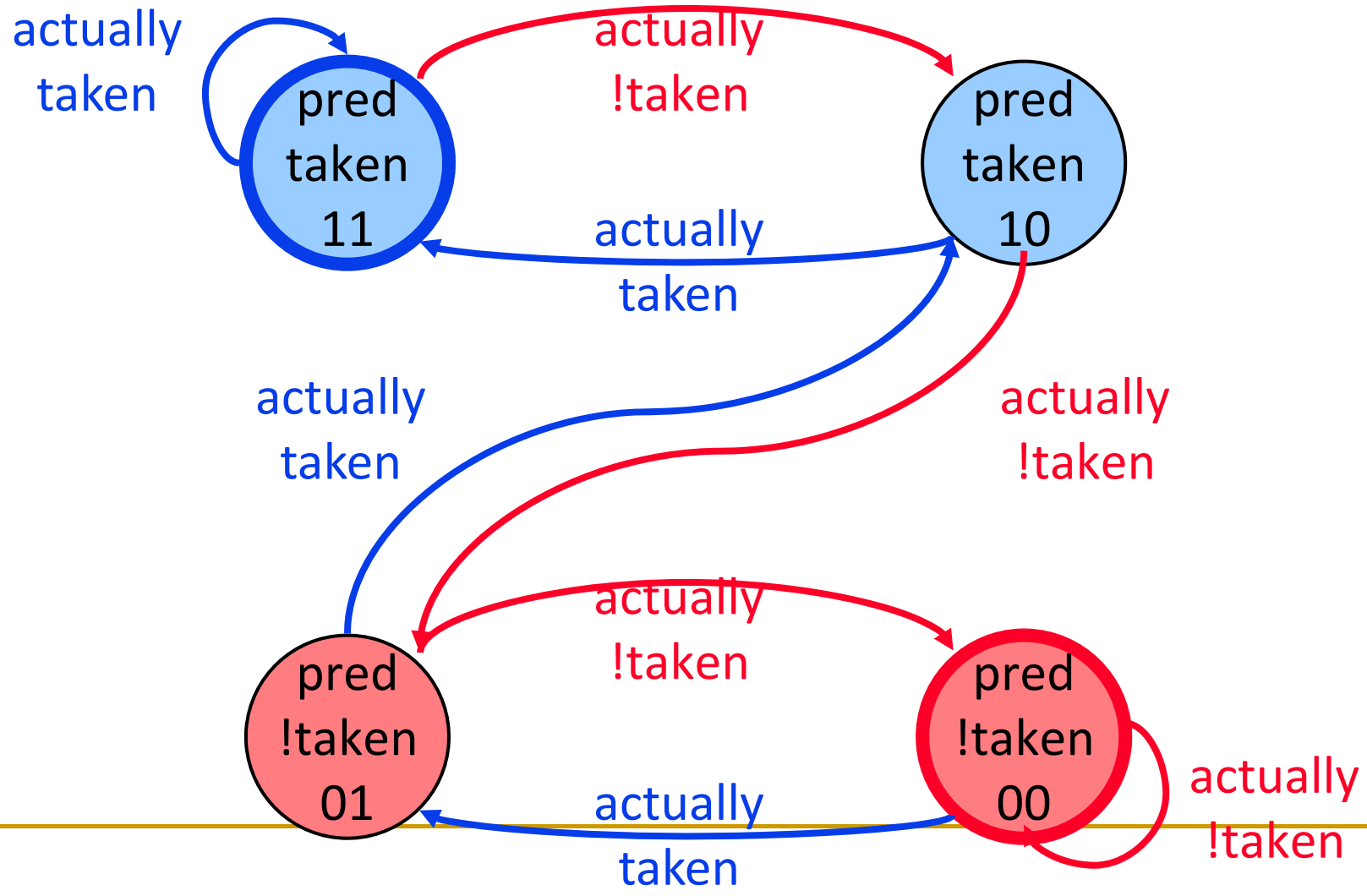
- Accuracy for a loop with N iterations = (N-1)/N

  TNTNTNTNTNTNTNTNTNTNTN → 50% accuracy

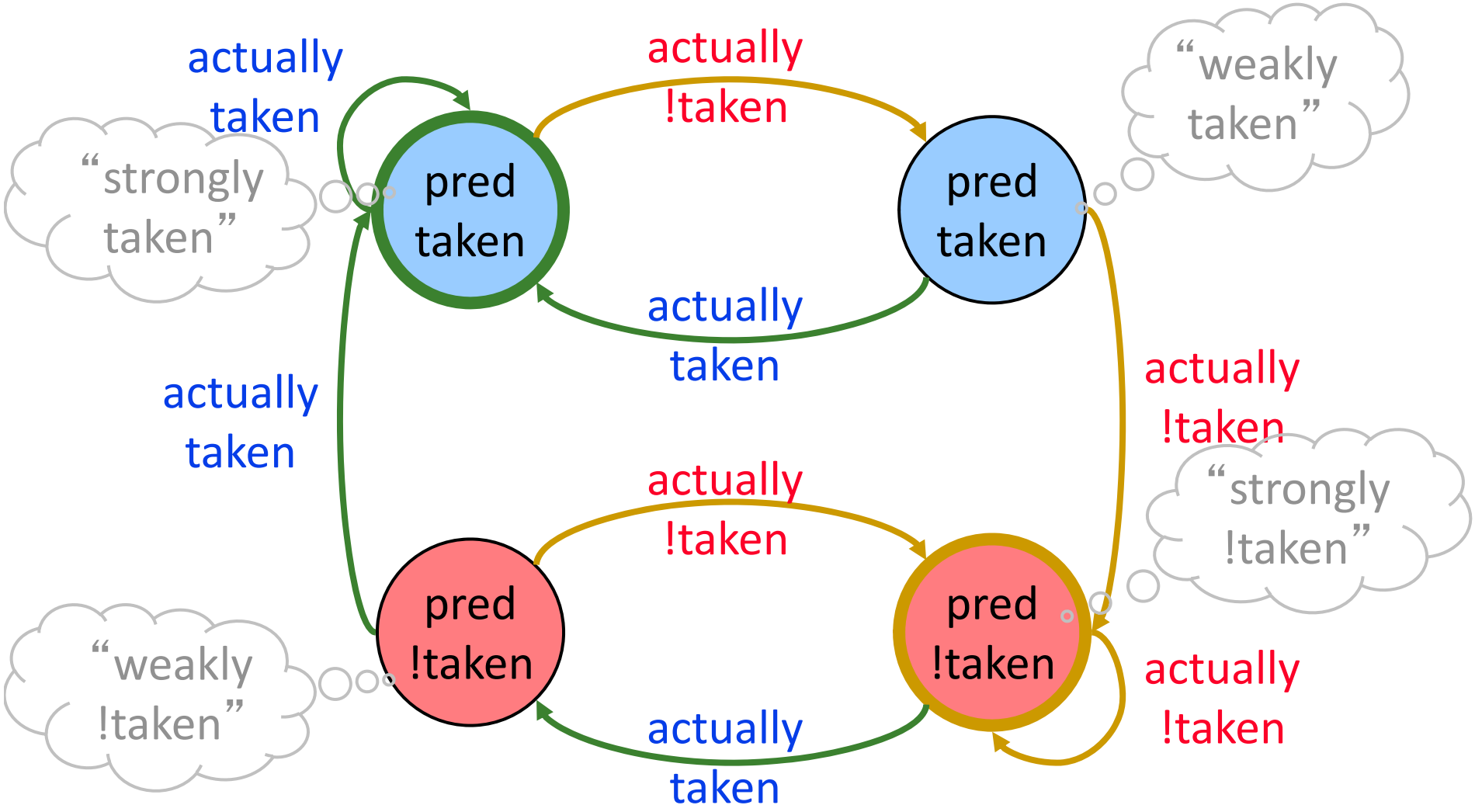
(assuming counter initialized to weakly taken)

Review: State Machine for 2-bit Counter

- Counter using *saturating arithmetic*
  - Arithmetic with maximum and minimum values

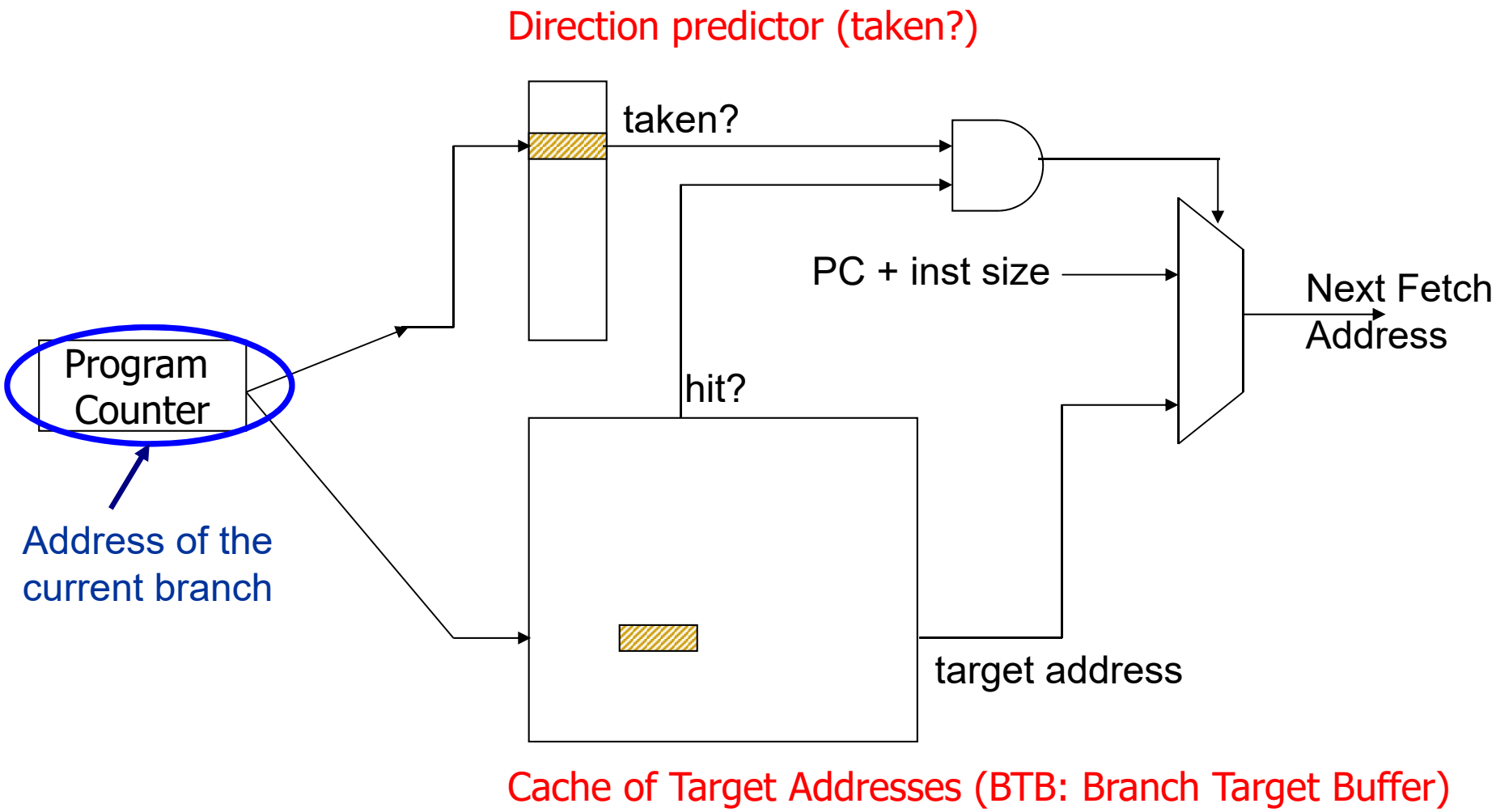


Review: Hysteresis Using a 2-bit Counter



Change prediction after 2 consecutive mistakes

Fetch Stage with BTB and Direction Prediction



Is This Good Enough?

- ~85-90% accuracy for **many** programs with 2-bit counter based prediction (also called bimodal prediction)
- Is this good enough?
- How big is the branch problem?

Let's Do the Exercise Again

- Assume N = 20 (20 pipe stages), W = 5 (5 wide fetch)
- Assume: 1 out of 5 instructions is a branch
- Assume: Each 5 instruction-block ends with a branch
- Assume: Branch direction is determined at the final pipe stage.

- How long does it take to fetch 500 instructions?
  - 100% accuracy
    - 100 cycles (all instructions fetched on the correct path)
    - No wasted work
  - 95% accuracy
    - 100 (correct path) + 20 * 5 (wrong path) = 200 cycles
    - 100% extra instructions fetched
  - 90% accuracy
    - 100 (correct path) + 20 * 10 (wrong path) = 300 cycles
    - 200% extra instructions fetched
  - 85% accuracy
    - 100 (correct path) + 20 * 15 (wrong path) = 400 cycles
    - 300% extra instructions fetched

Can We Do Better: Two-Level Prediction

- Last-time and 2BC predictors exploit "last-time" predictability

- Realization 1: A branch's outcome can be correlated with other branches' outcomes
  - Global branch correlation

- Realization 2: A branch's outcome can be correlated with past outcomes of the same branch (other than the outcome of the branch "last-time" it was executed)
  - Local branch correlation

Yeh and Patt, "Two-Level Adaptive Training Branch Prediction," MICRO 1991.

Global Branch Correlation (I)

- Recently executed branch outcomes in the execution path are correlated with the outcome of the next branch

```
if (cond1)
...
if (cond1 AND cond2)
```

- If first branch not taken, second also not taken

```
branch Y: if (cond1) a = 2;
...
branch X: if (a == 0)
```

- If first branch taken, second definitely not taken

Global Branch Correlation (II)

```
branch Y: if (cond1)
...
branch Z: if (cond2)
...
branch X: if (cond1 AND cond2)
```

- If Y and Z both taken, then X also taken
- If Y or Z not taken, then X also not taken

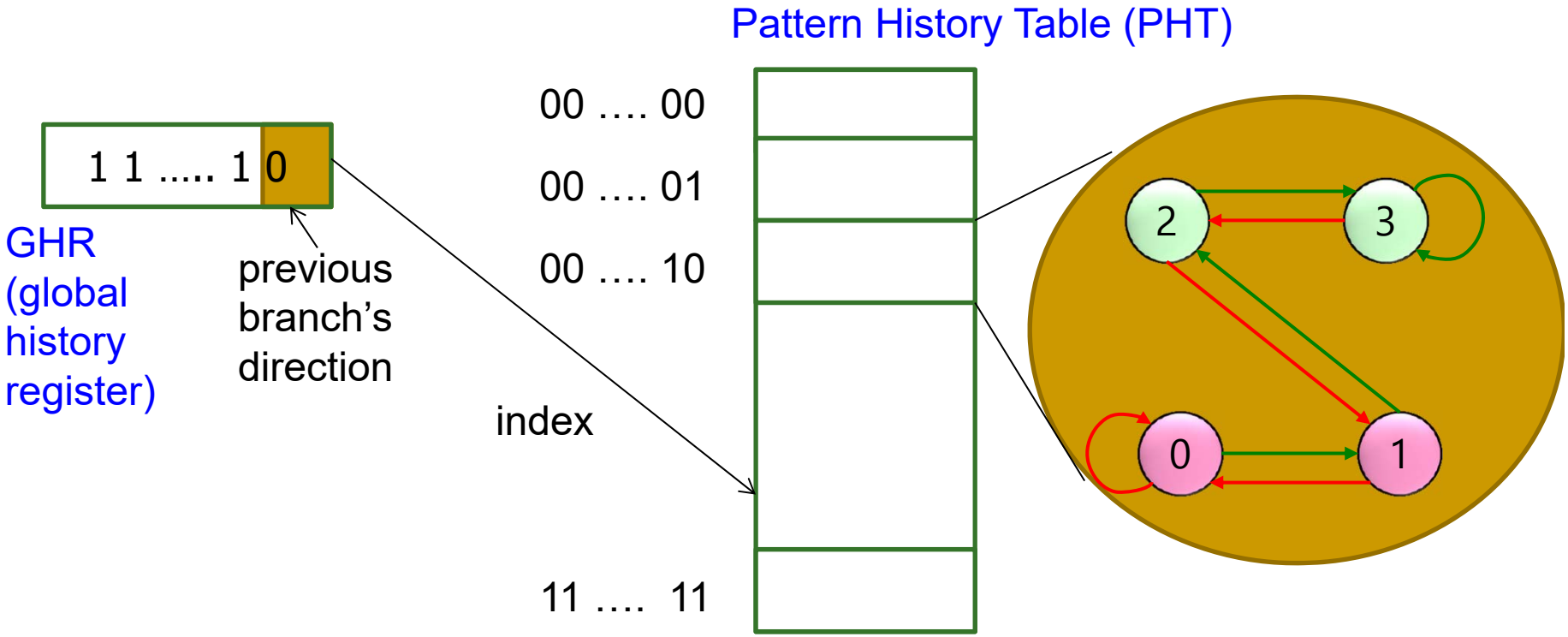
Capturing Global Branch Correlation

- Idea: Associate branch outcomes with "global T/NT history" of all branches
- Make a prediction based on the outcome of the branch the last time the same global branch history was encountered
- Implementation:
  - Keep track of the "global T/NT history" of all branches in a register → Global History Register (GHR)
  - Use GHR to index into a table that recorded the outcome that was seen for each GHR value in the recent past → Pattern History Table (table of 2-bit counters)
- Global history/branch predictor
- Uses two levels of history (GHR + history at that GHR)

Yeh and Patt, "Two-Level Adaptive Training Branch Prediction," MICRO 1991.

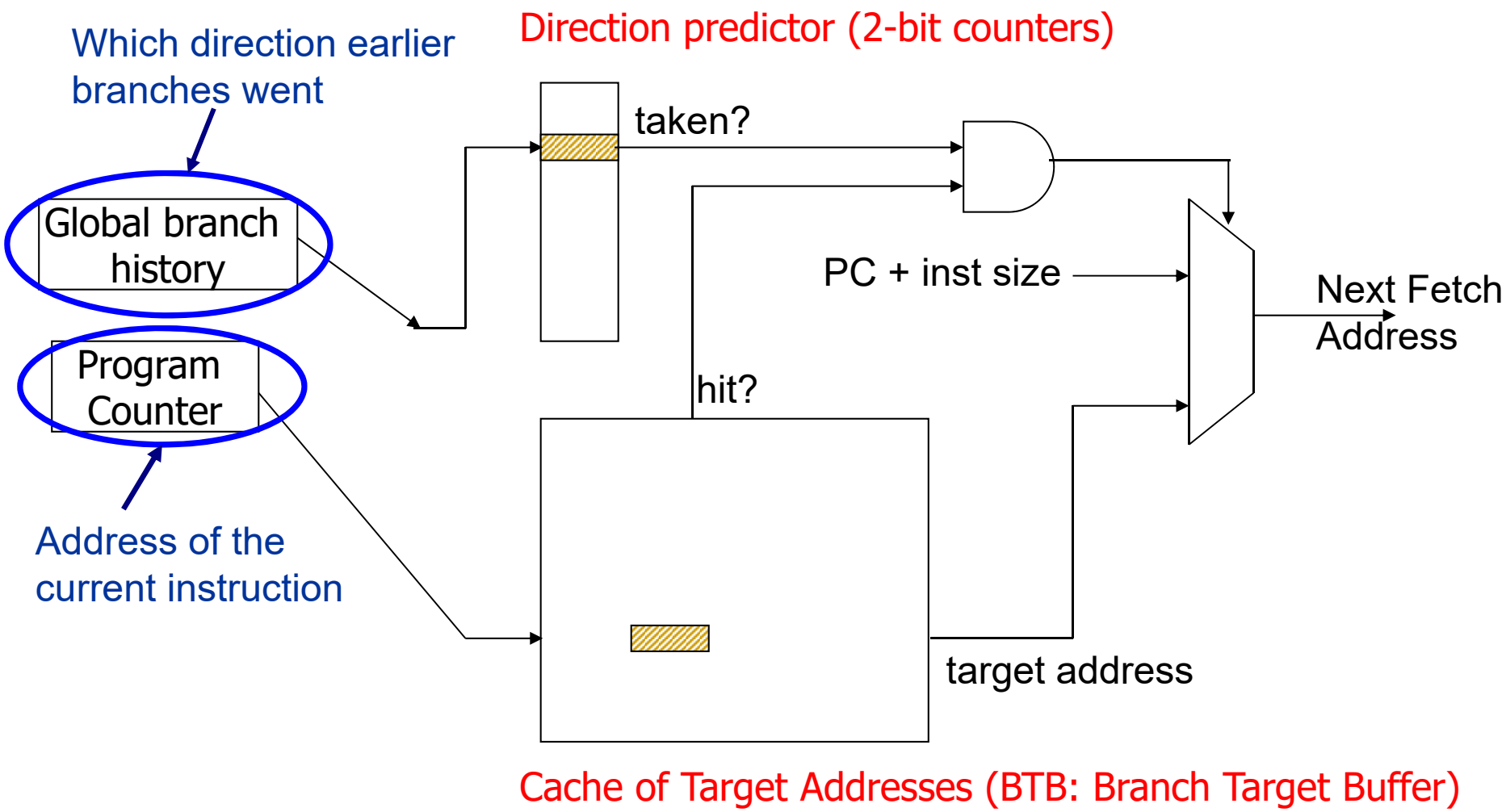
Two Level Global Branch Prediction

- 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



Yeh and Patt, "Two-Level Adaptive Training Branch Prediction," MICRO 1991.

Two-Level Global History Branch Predictor

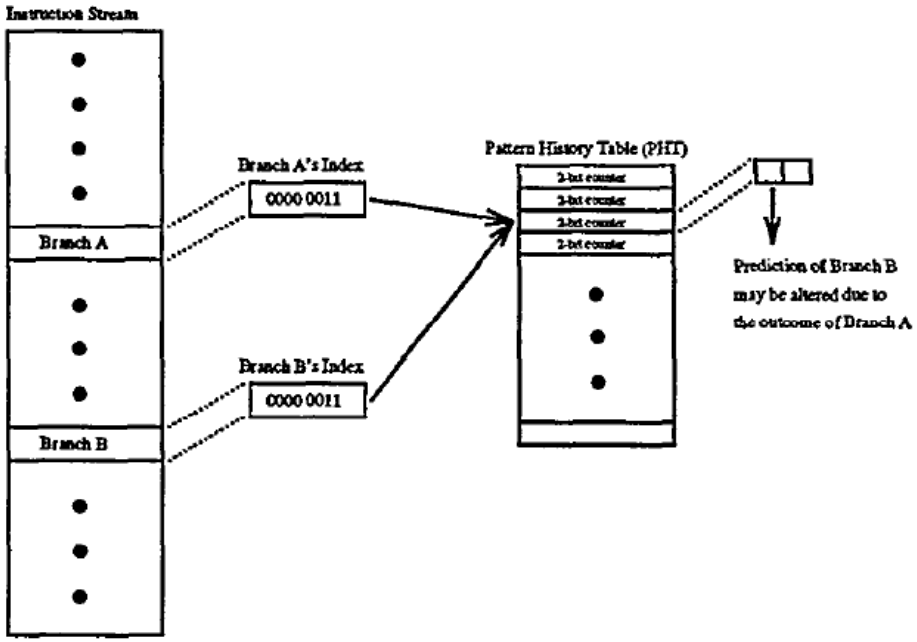


Example: Intel Pentium Pro Branch Predictor

- Two level global branch predictor
- 4-bit global history register
- Multiple pattern history tables (of 2 bit counters)
  - Which pattern history table to use is determined by lower order bits of the branch address

An Issue: Interference in the PHTs

- 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?

Reducing Interference in PHTs (I)

1. Increase size of PHT

2. Branch filtering
   - Predict highly-biased branches separately so that they do not consume PHT entries
   - E.g., static prediction or BTB based prediction

3. Hashing/index-randomization
   - Gshare
   - Gskew

Biased Branches and Branch Filtering

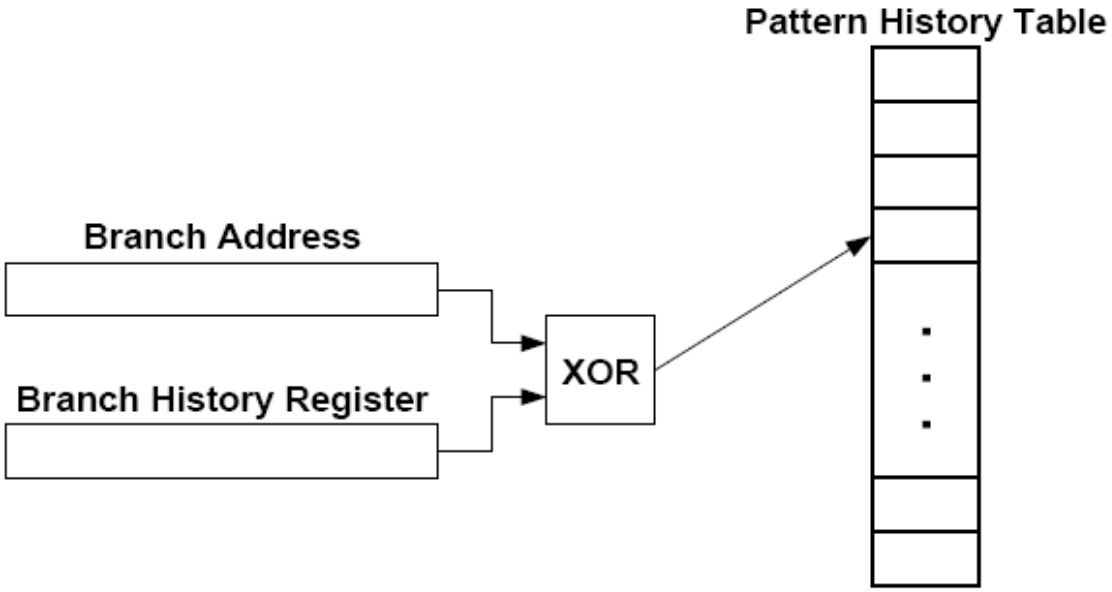
- Observation: Many branches are biased in one direction (e.g., 99% taken)
- Problem: These branches *pollute* the branch prediction structures → make the prediction of other branches difficult by causing "interference" in branch prediction tables and history registers
- Solution: Detect such biased branches, and predict them with a simpler predictor (e.g., last time, static, …)
- Chang et al., "Branch classification: a new mechanism for improving branch predictor performance," MICRO 1994.

Reducing Interference: Gshare

- Idea 1: Randomize the indexing function into the PHT such that probability of two branches mapping to the same entry reduces
  - Gshare predictor: GHR hashed with the Branch PC

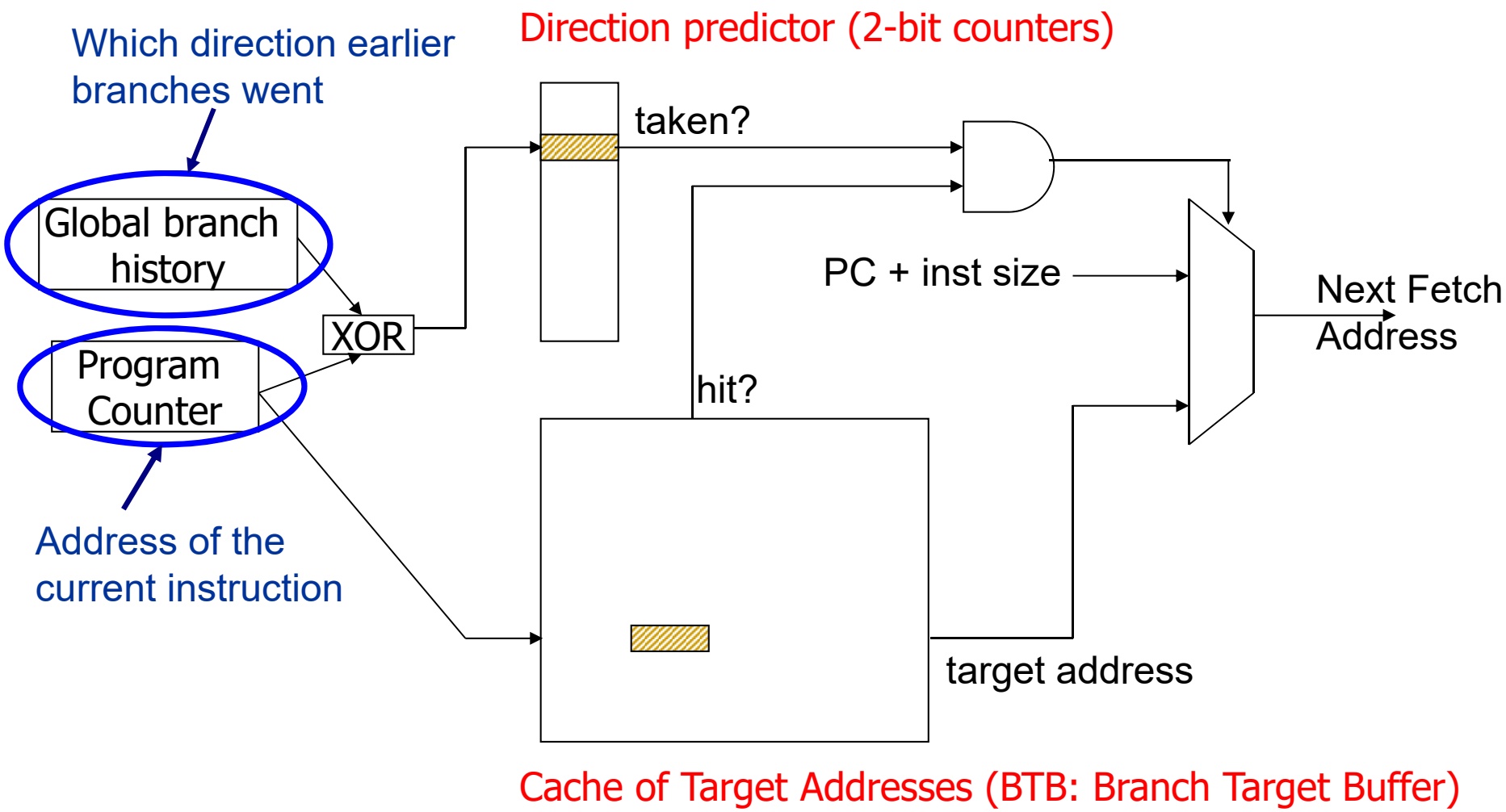
  + Better utilization of PHT + More context information

  - Increases access latency



  - McFarling, "Combining Branch Predictors," DEC WRL Tech Report, 1993.

Two-Level Gshare Branch Predictor

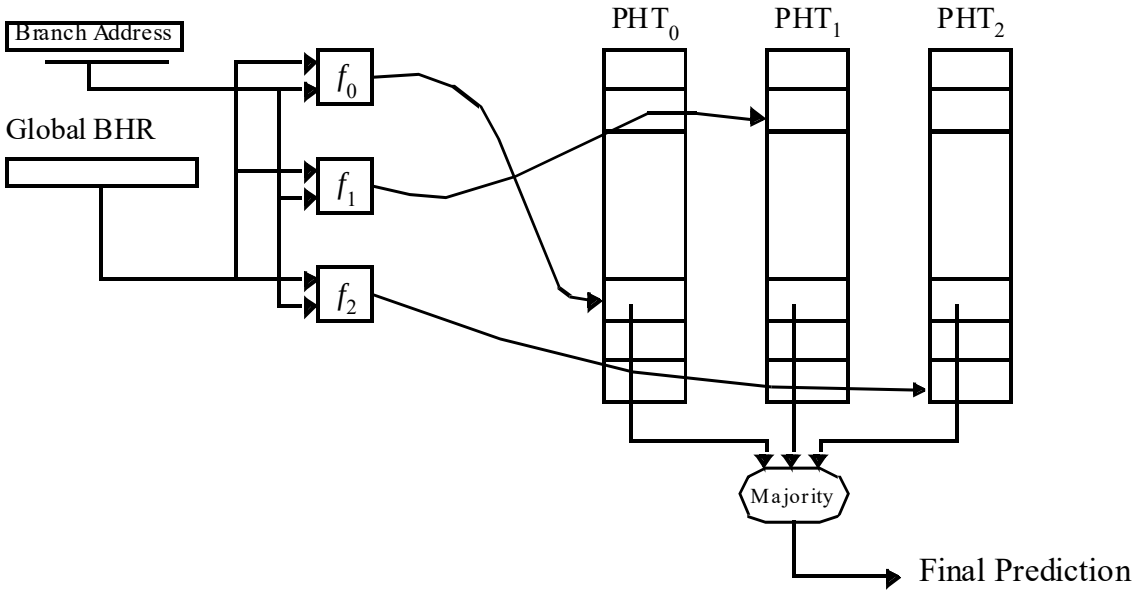


Reducing Interference: Gskew

- Idea 2: Gskew predictor
  - Multiple PHTs
  - **Each indexed with a different type of hash function**
  - Final prediction is a majority vote

+ Distributes interference patterns in a more randomized way (interfering patterns less likely in different PHTs at the same time)

-- More complexity (due to multiple PHTs, hash functions)



Seznec, "An optimized 2bcgskew branch predictor," IRISA Tech Report 1993.

Michaud, "Trading conflict and capacity aliasing in conditional branch predictors," ISCA 1997

Can We Do Better: Two-Level Prediction

- Last-time and 2BC predictors exploit only "last-time" predictability for a given branch
- Realization 1: A branch's outcome can be correlated with other branches' outcomes
  - Global branch correlation
- Realization 2: A branch's outcome can be correlated with past outcomes of the same branch (in addition to the outcome of the branch "last-time" it was executed)
  - Local branch correlation

Yeh and Patt, "Two-Level Adaptive Training Branch Prediction," MICRO 1991.

Local Branch Correlation

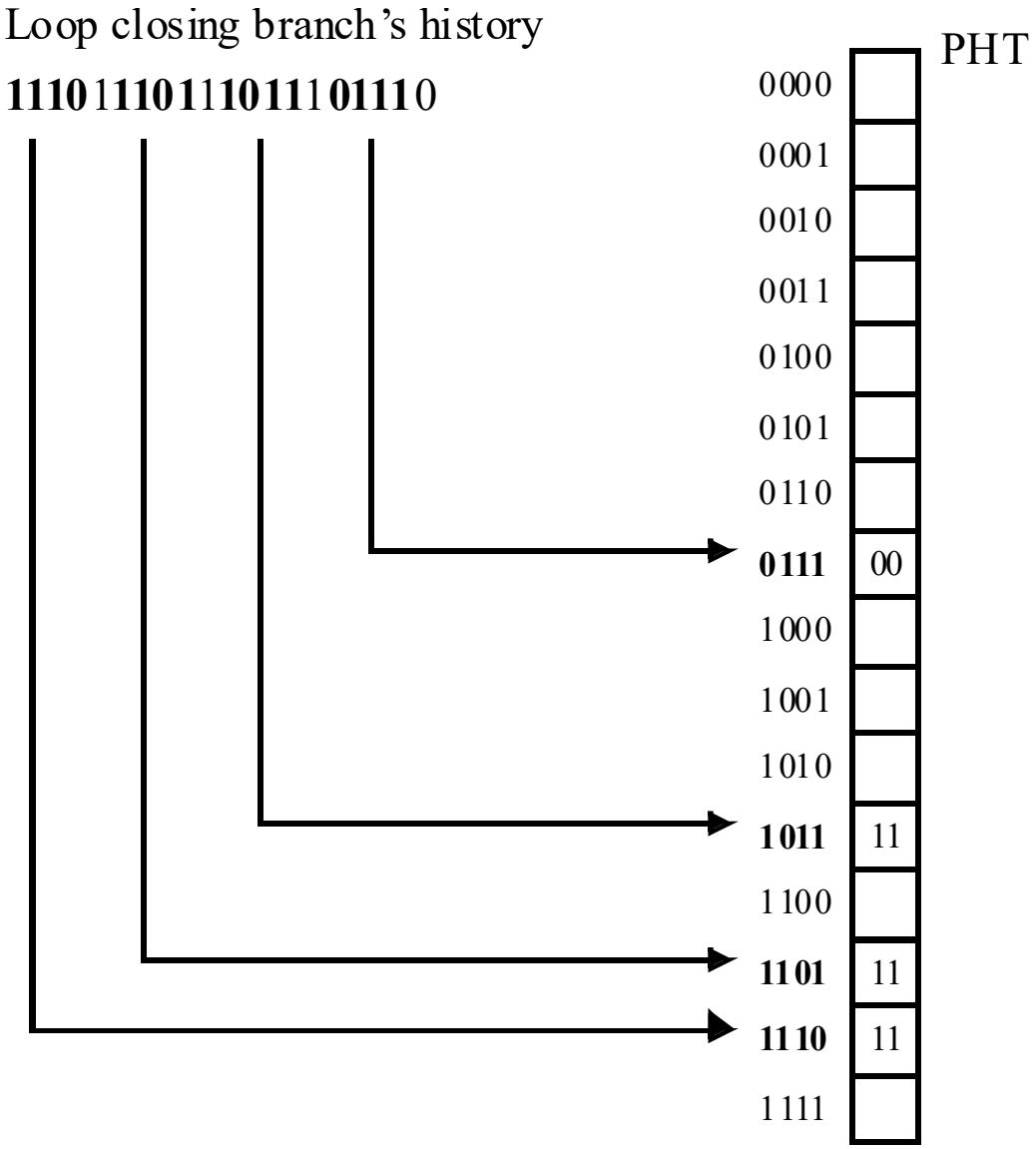
```
for (i=1; i<=4; i++) { }
```

If the loop test is done at the end of the body, the corresponding branch will execute the pattern $(1110)^n$, where 1 and 0 represent taken and not taken respectively, and n is the number of times the loop is executed. Clearly, if we knew the direction this branch had gone on the previous three executions, then we could always be able to predict the next branch direction.

- McFarling, "Combining Branch Predictors," DEC WRL TR 1993.

More Motivation for Local History

- To predict a loop branch "perfectly", we want to identify the last iteration of the loop
- By having a separate PHT entry for each local history, we can distinguish different iterations of a loop
- Works for "short" loops



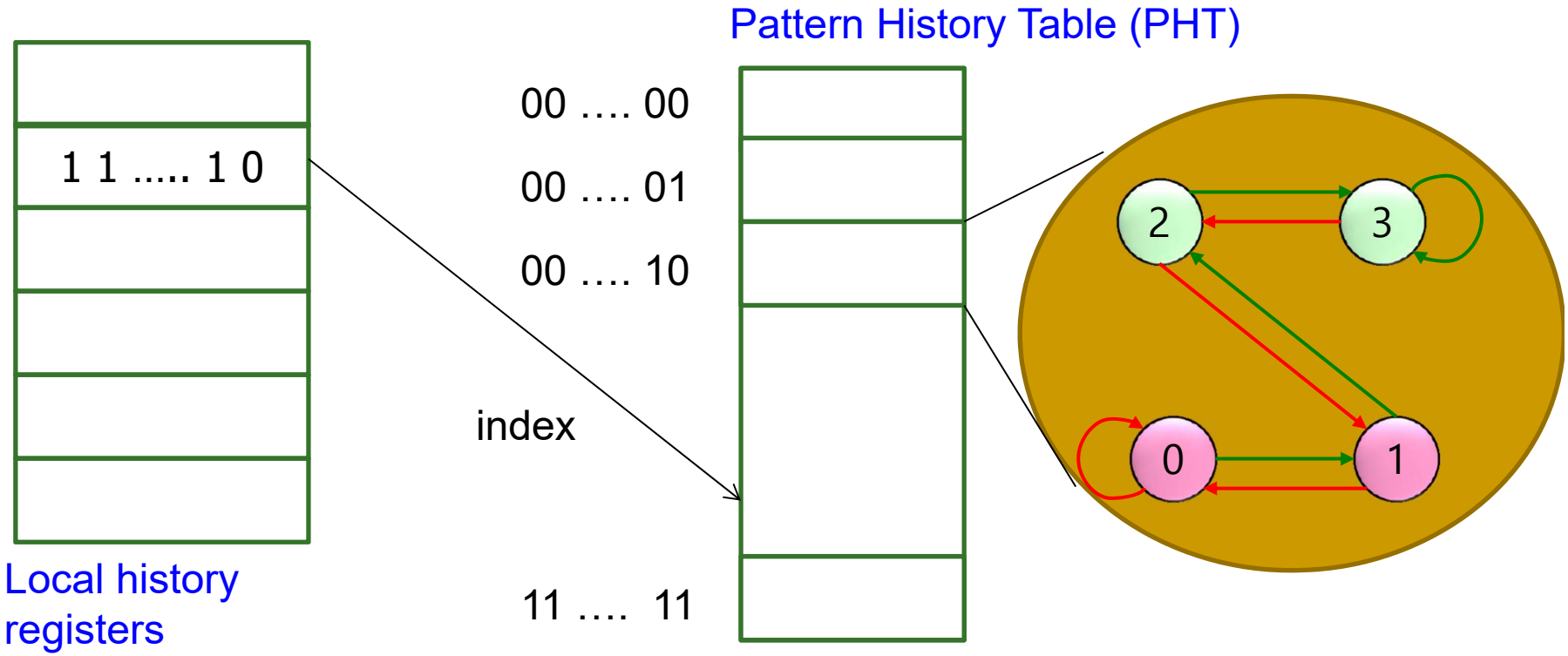
Capturing Local Branch Correlation

- Idea: Have a per-branch history register
  - Associate the predicted outcome of a branch with "T/NT history" of the same branch
- Make a prediction based on the outcome of the branch the last time the same local branch history was encountered

- Called the local history/branch predictor
- Uses two levels of history (Per-branch history register + history at that history register value)

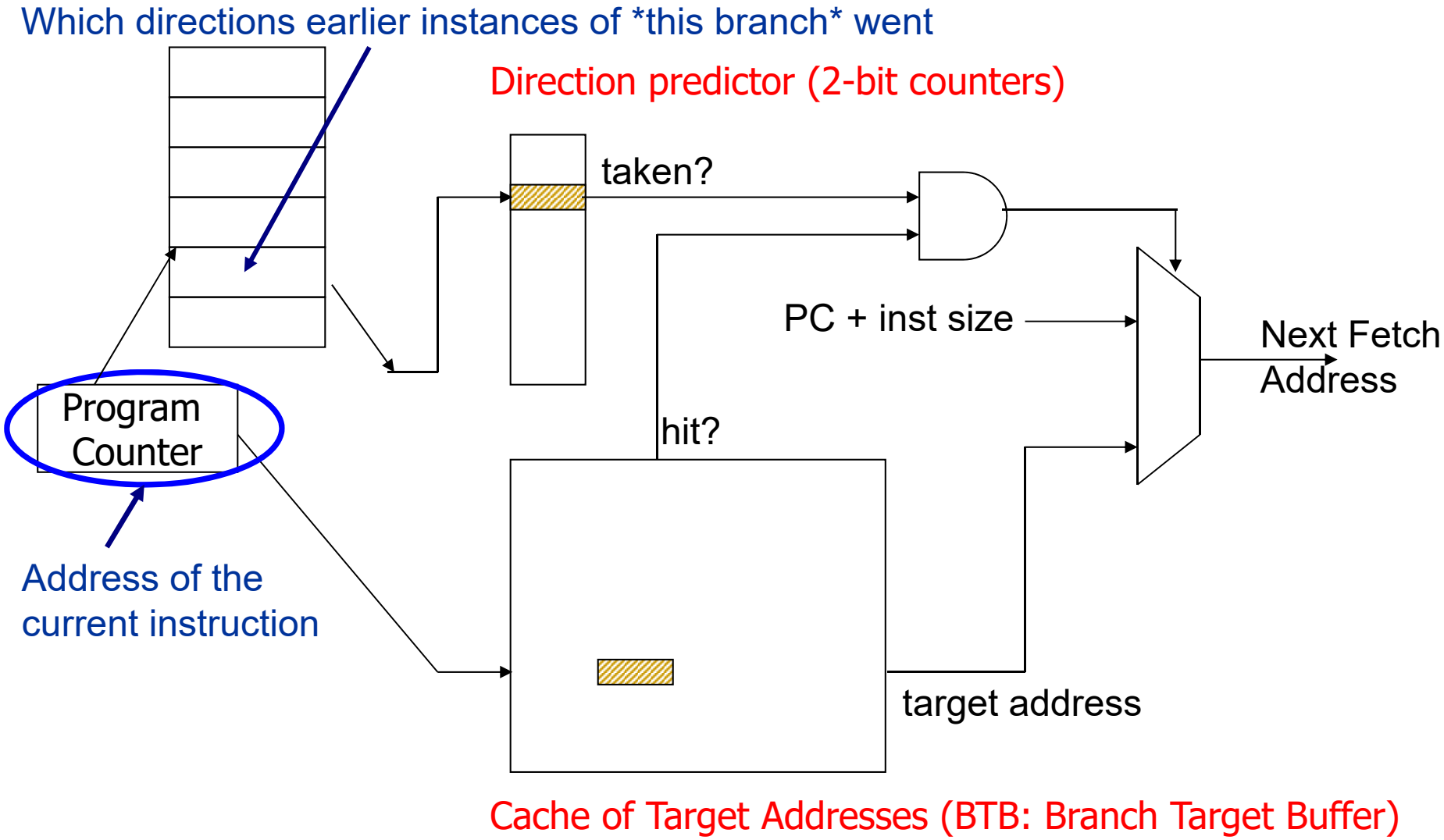
Two Level Local Branch Prediction

- First level: A set of local history registers (N bits each)
  - Select the history register based on the PC of the branch
- Second level: Table of saturating counters for each history entry
  - The direction the branch took the last time the same history was seen



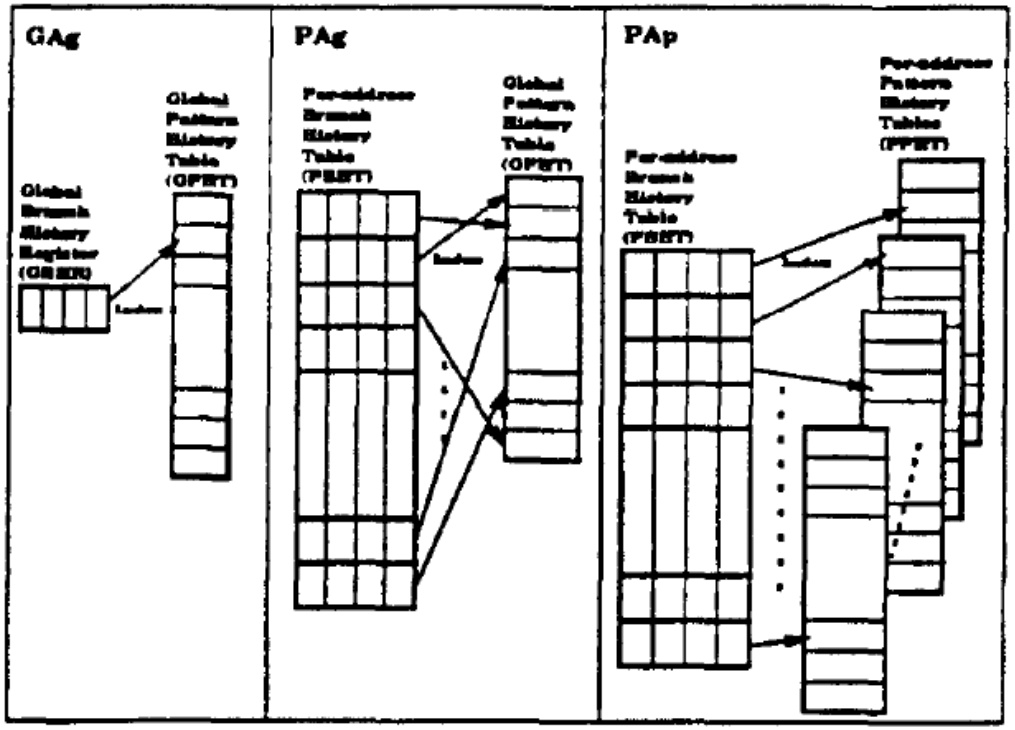
Yeh and Patt, "Two-Level Adaptive Training Branch Prediction," MICRO 1991.

Two-Level Local History Branch Predictor



Two-Level Predictor Taxonomy

- BHR can be global (G), per set of branches (S), or per branch (P)
- PHT counters can be adaptive (A) or static (S)
- PHT can be global (g), per set of branches (s), or per branch (p)



- Yeh and Patt, "Two-Level Adaptive Training Branch Prediction," MICRO 1991.

Can We Do Even Better?

- Predictability of branches varies

- Some branches are more predictable using local history
- Some using global
- For others, a simple two-bit counter is enough
- Yet for others, a bit is enough

- Observation: There is heterogeneity in predictability behavior of branches
  - No one-size fits all branch prediction algorithm for all branches

- Idea: Exploit that heterogeneity by designing heterogeneous branch predictors

Hybrid Branch Predictors

- Idea: Use more than one type of predictor (i.e., multiple algorithms) and select the "best" prediction
  - E.g., hybrid of 2-bit counters and global predictor

- Advantages:

  + Better accuracy: different predictors are better for different branches

  + Reduced warmup time (faster-warmup predictor used until the slower-warmup predictor warms up)

- Disadvantages:

  -- Need "meta-predictor" or "selector"

  -- Longer access latency

  - McFarling, "Combining Branch Predictors," DEC WRL Tech Report, 1993.

Are We Done w/ Branch Prediction?

- Hybrid branch predictors work well
  - E.g., 90-97% prediction accuracy on average

- Some "difficult" workloads still suffer, though!
  - E.g., gcc
  - Max IPC with tournament prediction: 9
  - Max IPC with perfect prediction: 35