Branch Prediction

Courtesy : Dr Onur Mutlu

Slides Adapted by: Dr Sparsh Mittal

Branch Prediction

- How do we keep the pipeline full in the presence of branches?
  - Guess the next instruction when a branch is fetched
  - Requires guessing the direction and target of a branch

- Branches can be unconditional or conditional

- For conditional branches, we need to predict
  - Whether the branch will be taken (=branch direction prediction)
  - Branch target address (if taken) (=branch target address prediction)

BTB

- 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
  - The structure is called Branch Target Buffer (BTB) or Branch Target Address Cache

In remainder of PPT, we only discuss branch direction prediction

Useful terms

- Backward branch: target address lower than branch PC
- Biased branches: are either taken or not-taken most of the time

Some Direction Prediction Techniques

- 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

Static Branch Prediction (I)

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

- Always taken
  - No direction prediction
  - Better accuracy: ~60-70% (for conditional branches)
    - 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 (II)

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

\+ Per branch prediction (more accurate than schemes in previous slide)

-- Requires hint bits in the branch instruction format

-- Cannot distinguish between such cases:

TTTTTTTTTTNNNNNNNNNN → 50% taken-rate

TNTNTNTNTNTNTNTNTNTN → 50% taken-rate

-- Accuracy depends on representativeness of profile input

-- Requires compiler analysis and ISA extension

Static Branch Prediction (III)

- Program-analysis based
  - Idea: Use heuristics based on program analysis to determine statically-predicted direction
  - Example loop heuristic: Predict a branch guarding a loop execution as taken (i.e., execute the loop)
  - Pointer and FP comparisons: Predict not equal
  - Branch on error: predict not taken

\+ Does not require profiling

-- Heuristics may not be representative or good

-- Requires compiler analysis and ISA support

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
  - Indicates which direction branch went last time it executed
  - Need one bit per branch

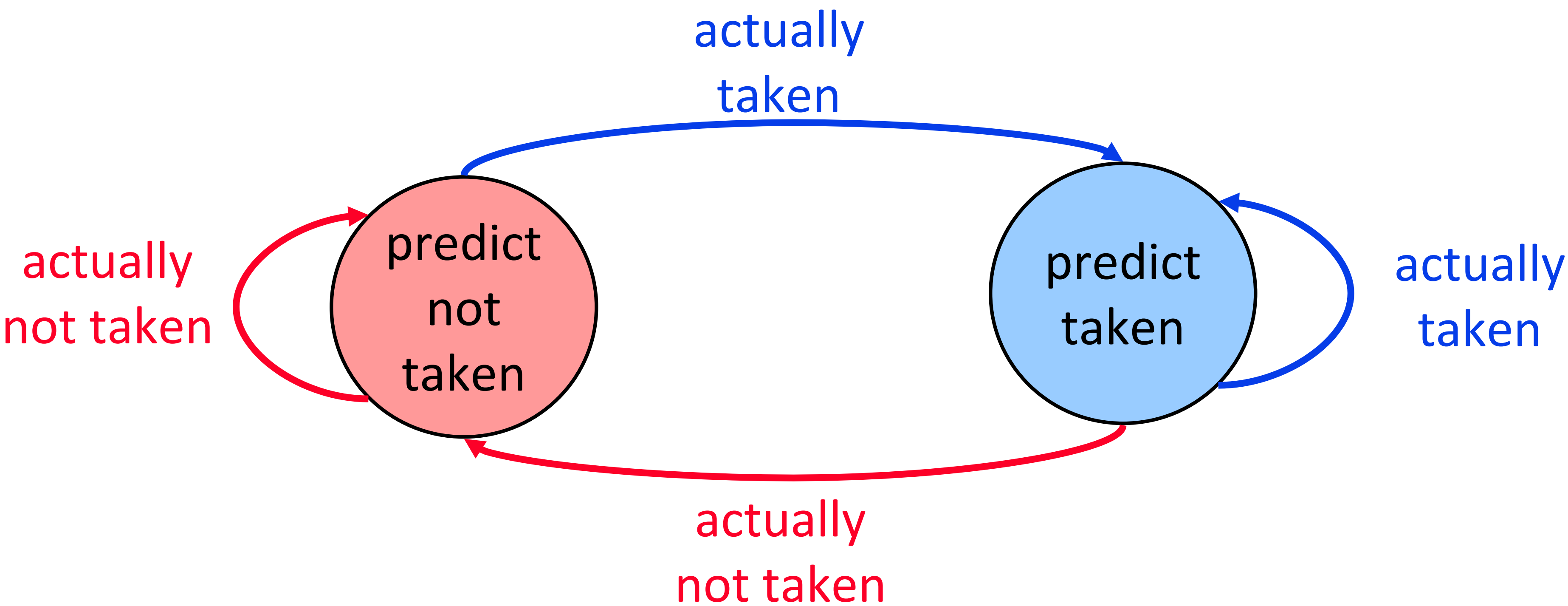
- Always mispredicts the last iteration and the first iteration of a loop branch (N= number of iterations)
  - Accuracy for a loop with N iterations = (N-2)/N

+ Good for loop branches for loops with large N

-- Poor for loop branches for loops will small N

TNTNTNTNTNTNTNTNTNTN → 0% accuracy

State Machine for Last-Time Prediction



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: 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

Two-Bit Counter (2BC) 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

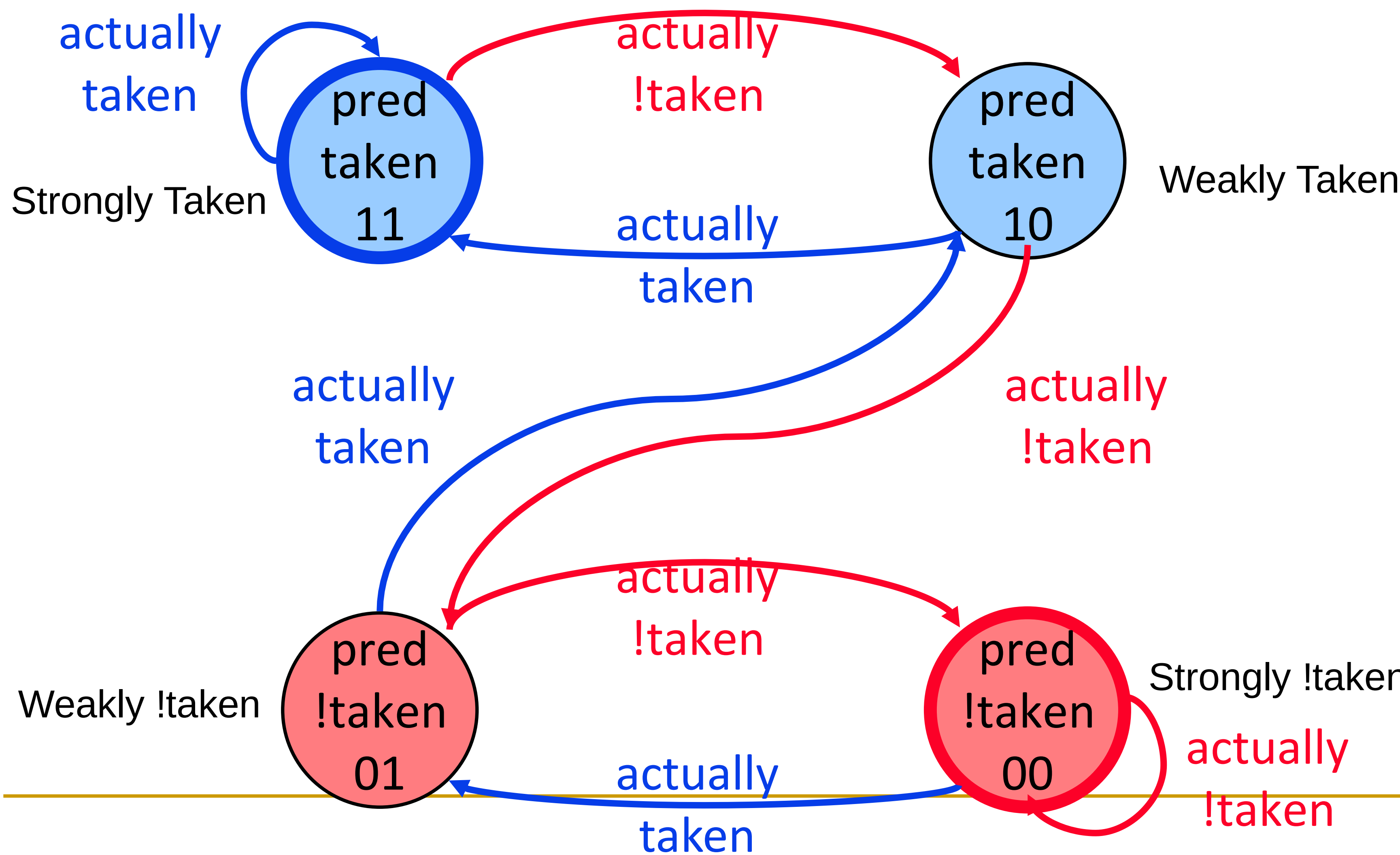
- TNTNTNTNTNTNTNTNTNTN → 50% accuracy
  (assuming counter initialized to weakly taken)

\+ Better prediction accuracy

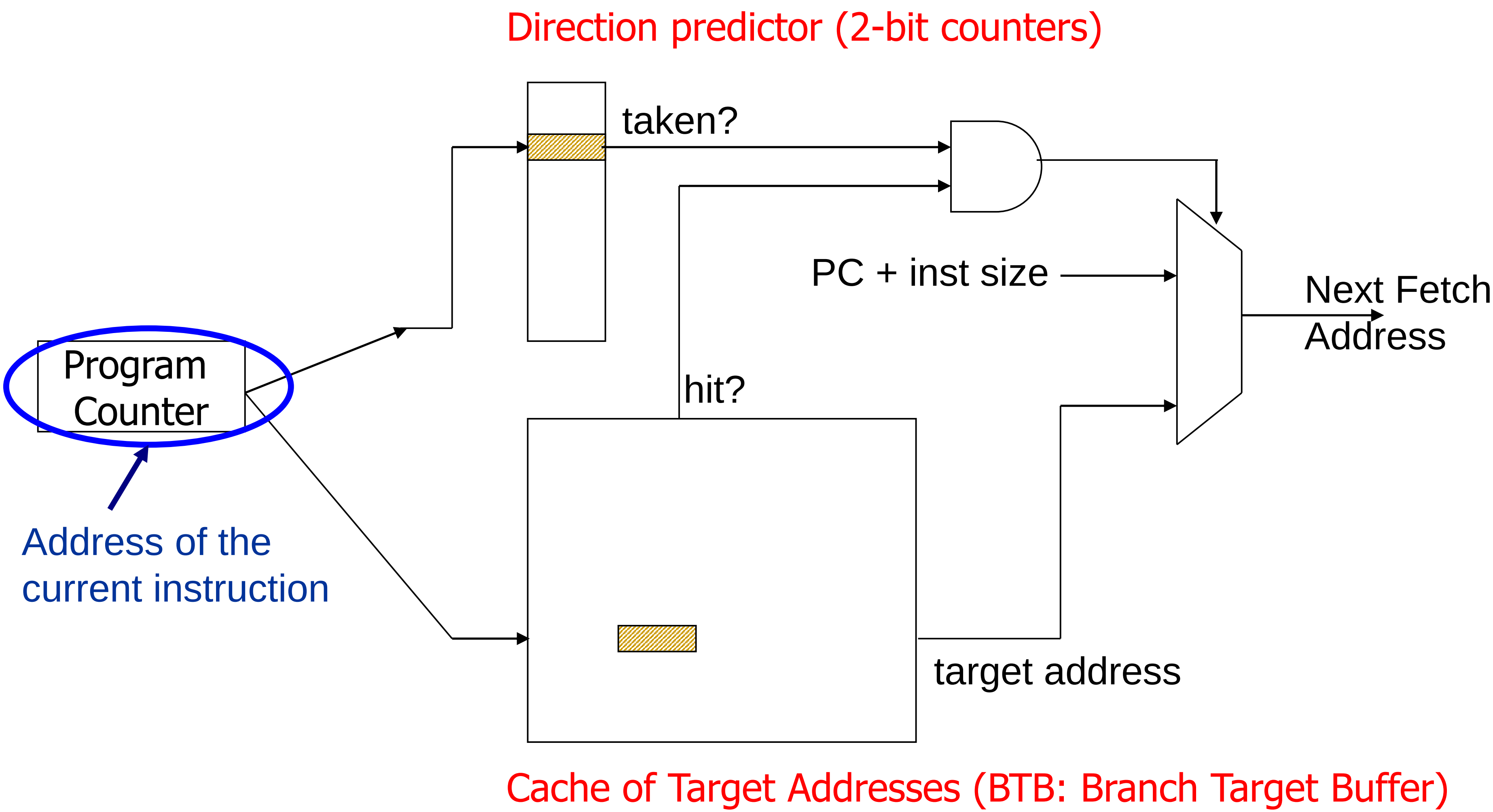
-- More hardware cost

State Machine for 2-bit Saturating Counter

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



One-Level Branch Predictor



Can We Do Better?

- Last-time and 2-bit counter 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

Global Branch Correlation (I)

- Recently executed branch outcomes in the execution path is 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

Global Branch Correlation (III)

```
if (aa==2)                    // B1
      aa=0;
if (bb==2)                    // B2
      bb=0;
if (aa!=bb) {                 // B3
      ….
  }
```

If B1 is true (i.e., aa==0@B3) and B2 is true (i.e. bb=0@B3) then B3 is certainly false

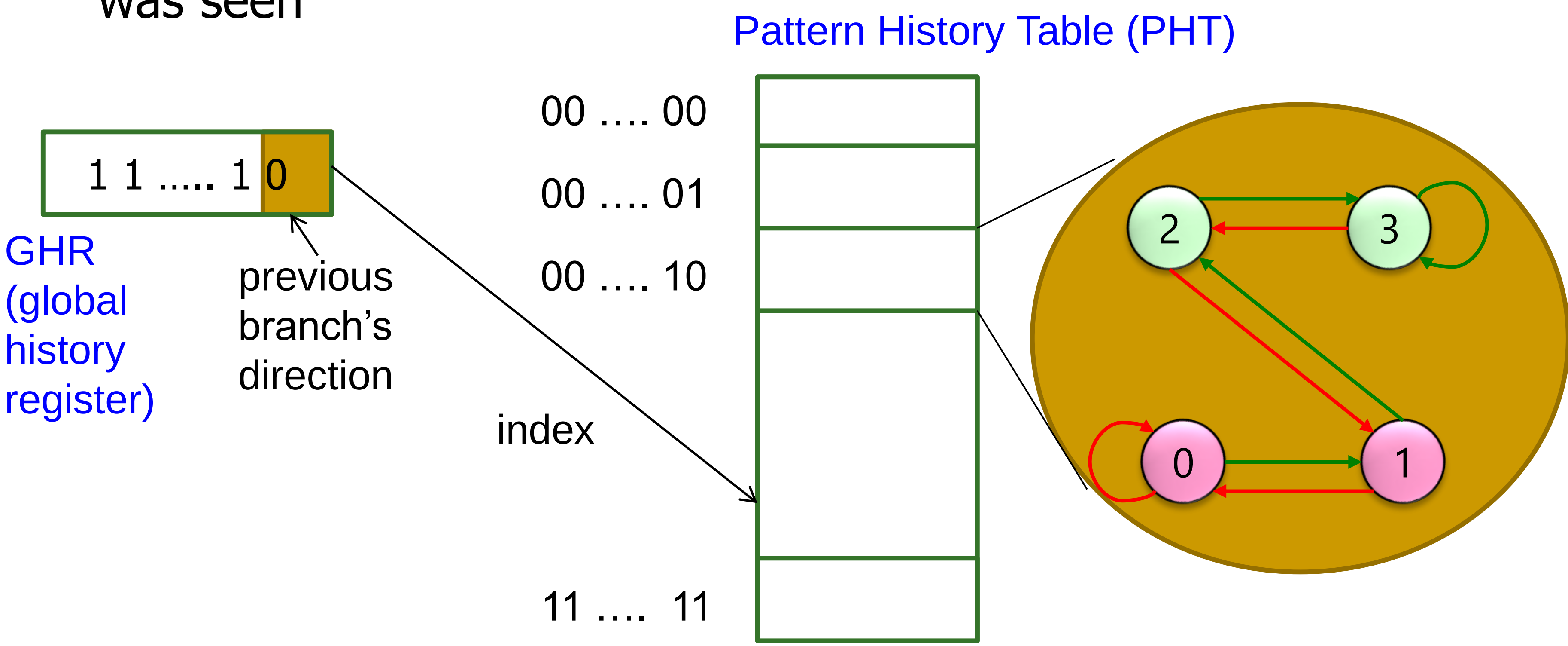
Capturing Global Branch Correlation

- 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

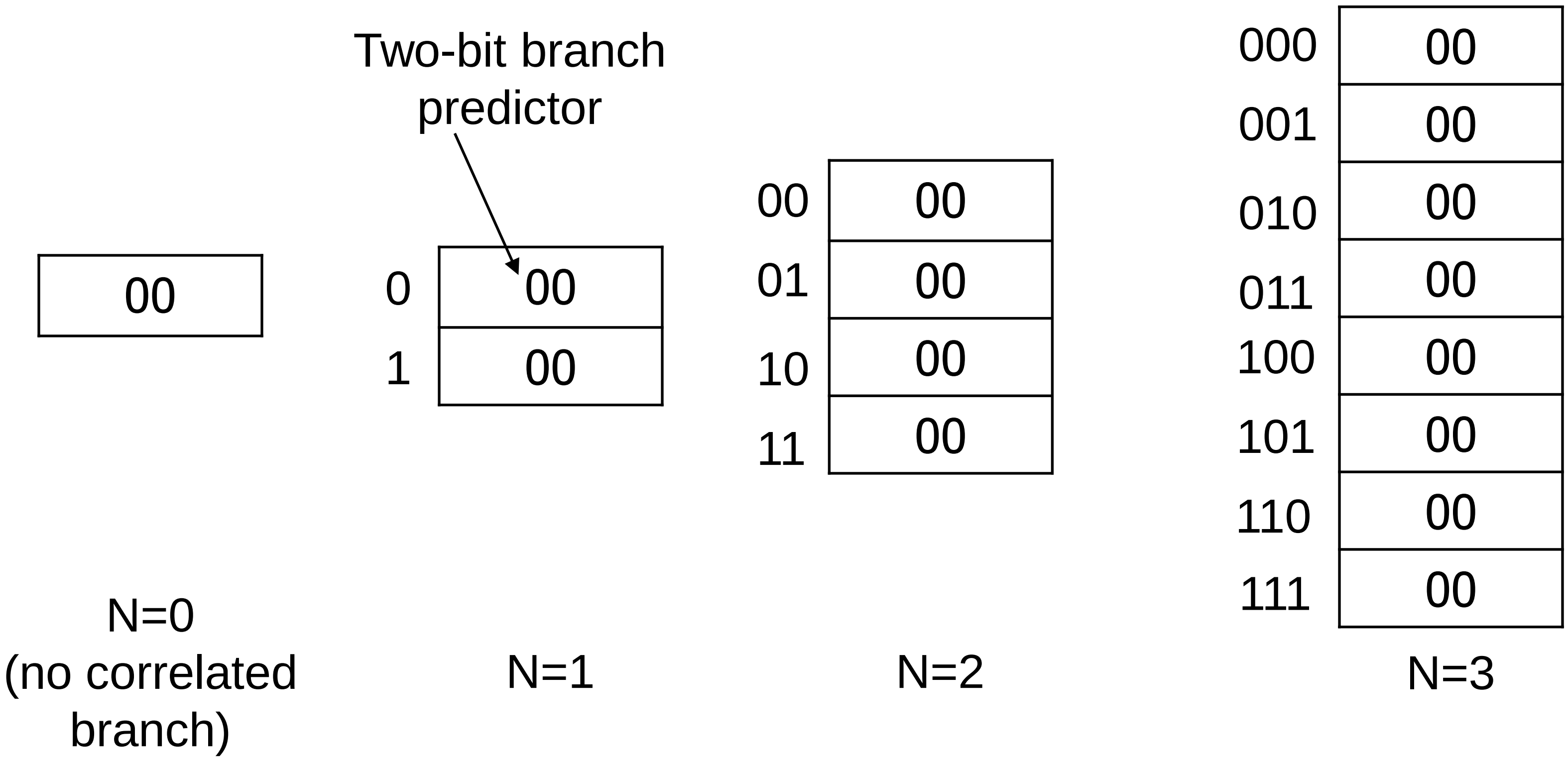
- Implementation:
  - Keep track of the "global T/NT history" of all branches in a register → Global History Register (GHR)
  - Use GHR to index into a table that recorded the outcome that was seen for each GHR value in the recent past → Pattern History Table (table of 2-bit counters)
- Called Global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

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



Examples (N= number of previous branches which are correlated)



Example of working

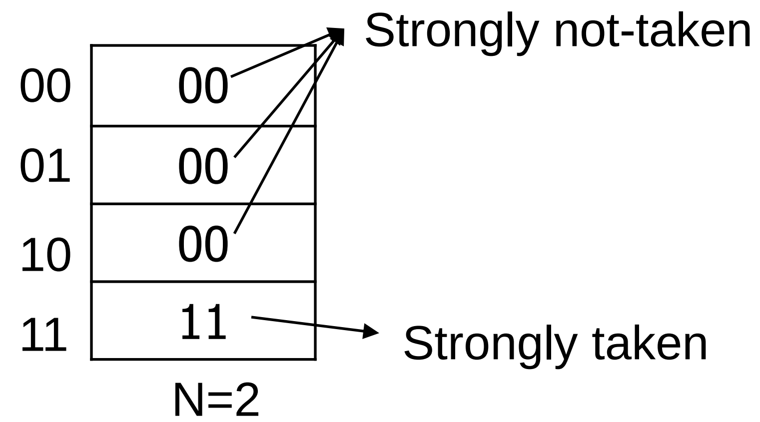
branch Y: if (cond1)

...

branch Z: if (cond2)

...

branch X: if (cond1 AND cond2)



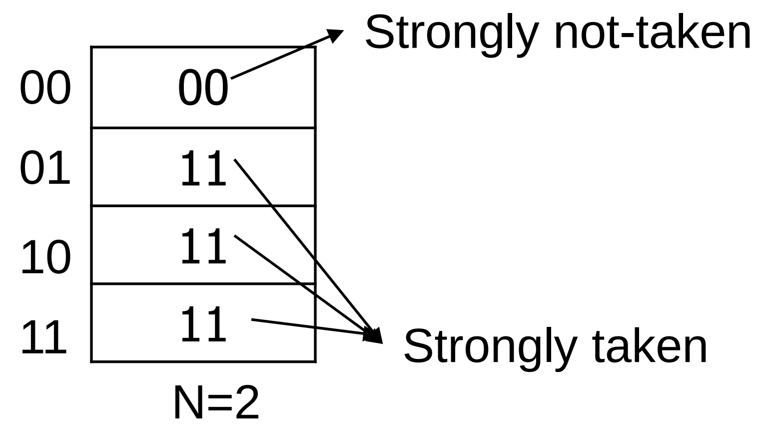
branch Y: if (cond1)

...

branch Z: if (cond2)

...

branch X: if (cond1 OR cond2)



Example of working

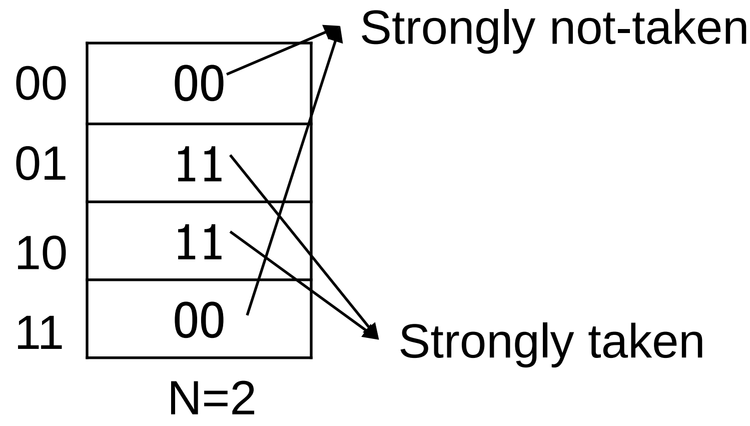
branch Y: if (cond1)

...

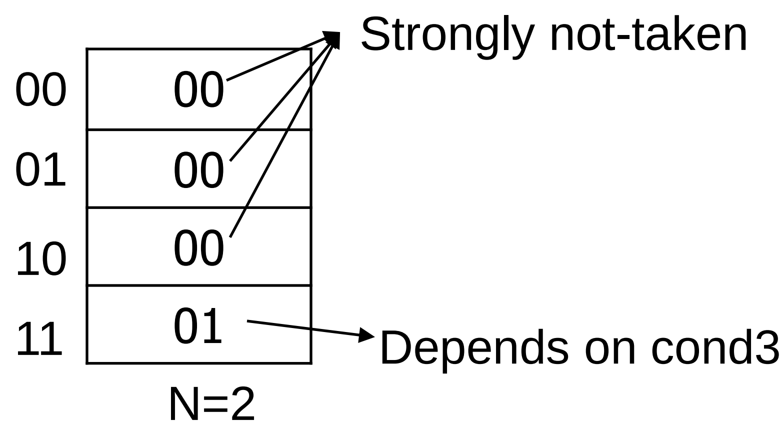
branch Z: if (cond2)

...

branch X: if (cond1 XOR cond2)

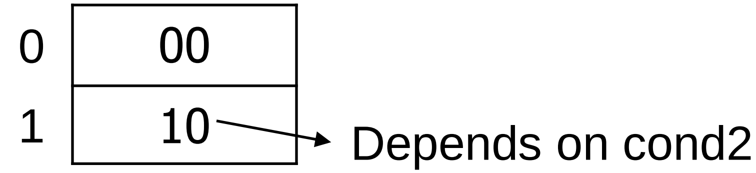


Branch Y: if (cond1)
…
Branch Z: if (cond2)
…
Branch X: if(cond1 AND cond2 and cond3)

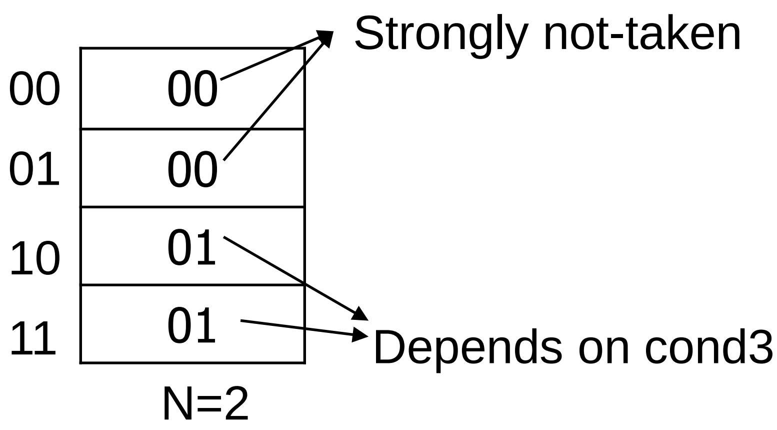


Example of working

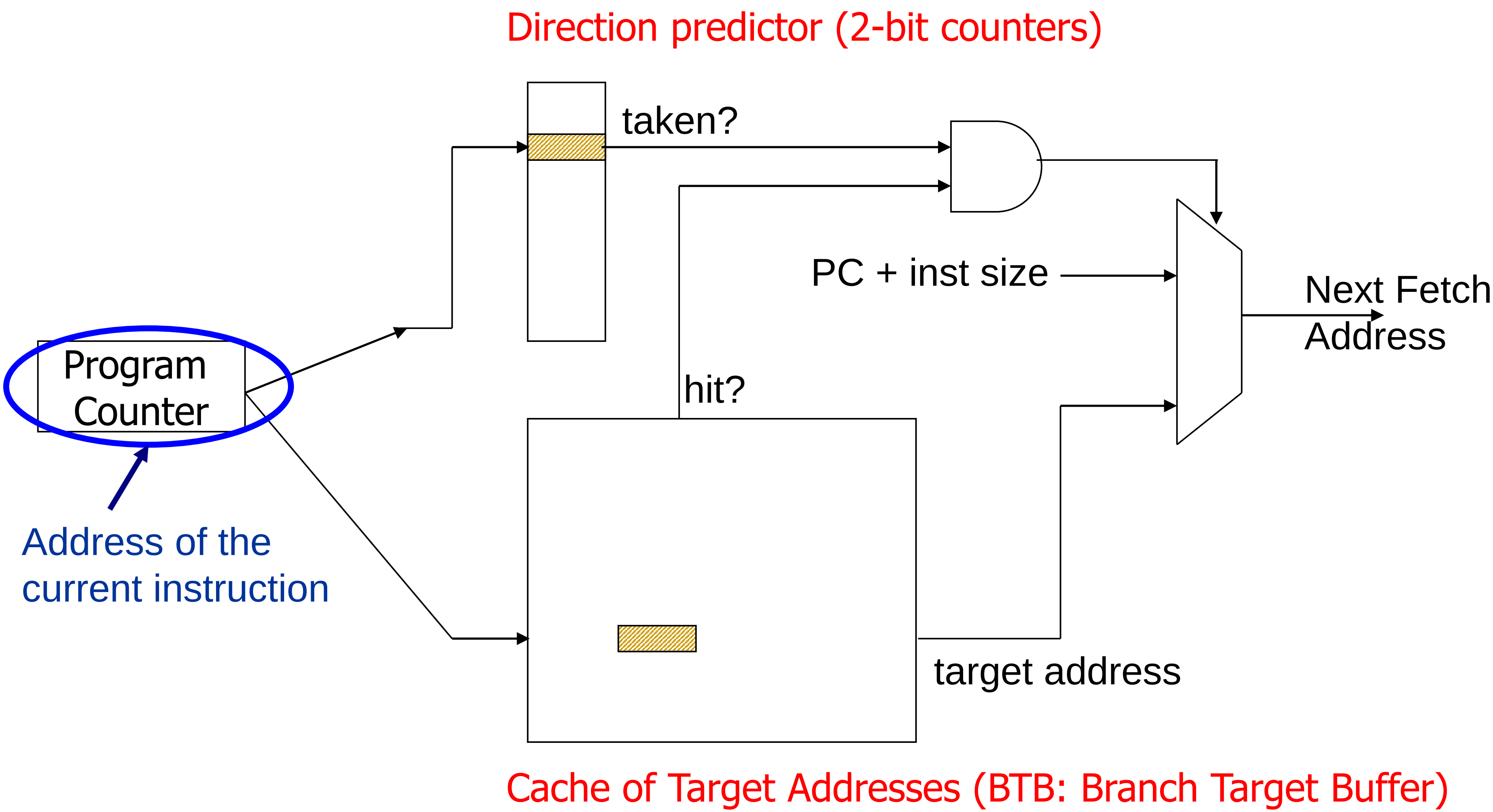
```
if (cond1)
...
if (cond1 AND cond2)
```



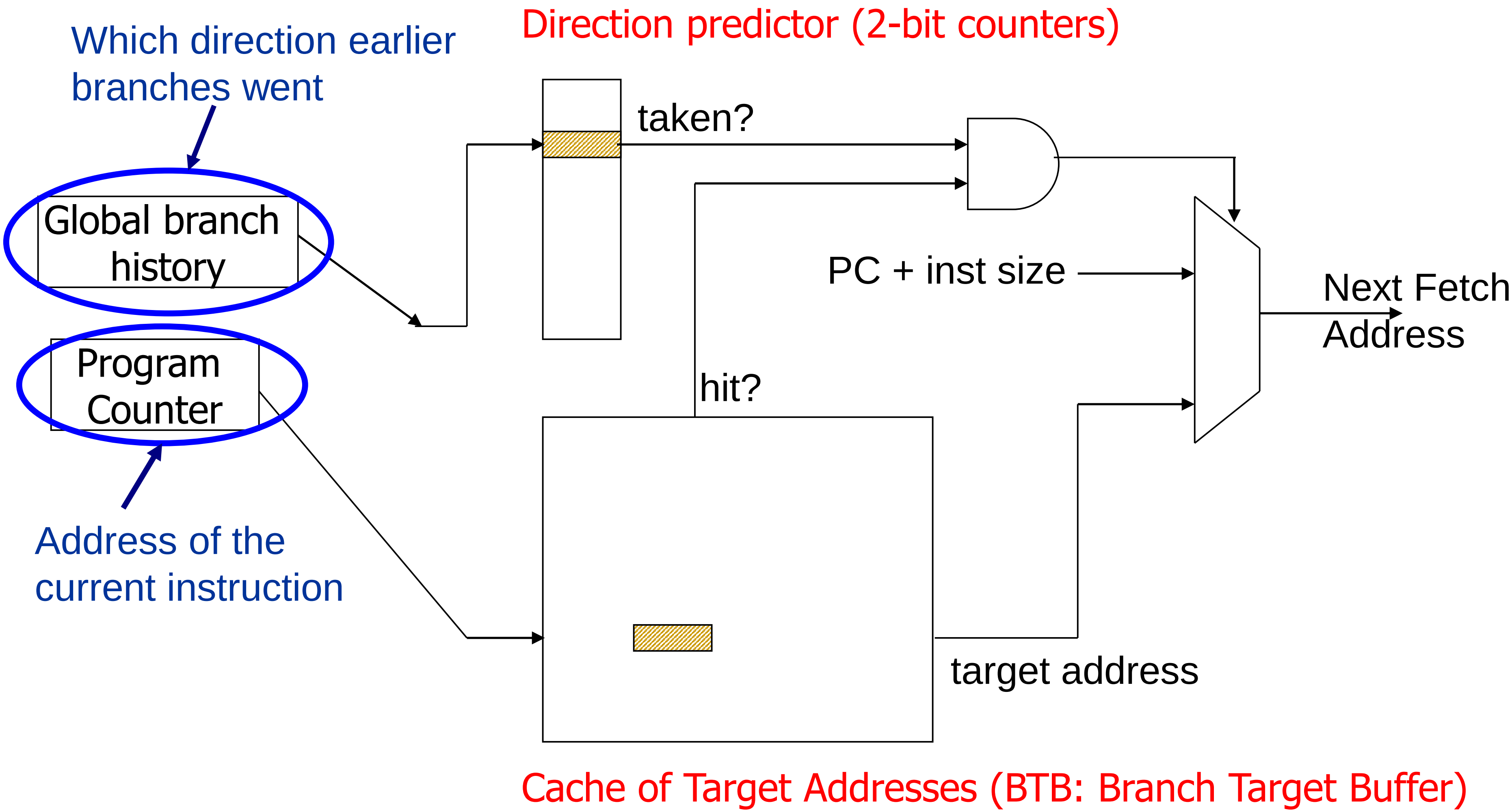
Branch Y: if (cond1)
…
Branch Z: if (cond2)
…
Branch X: if(cond1 and cond3)



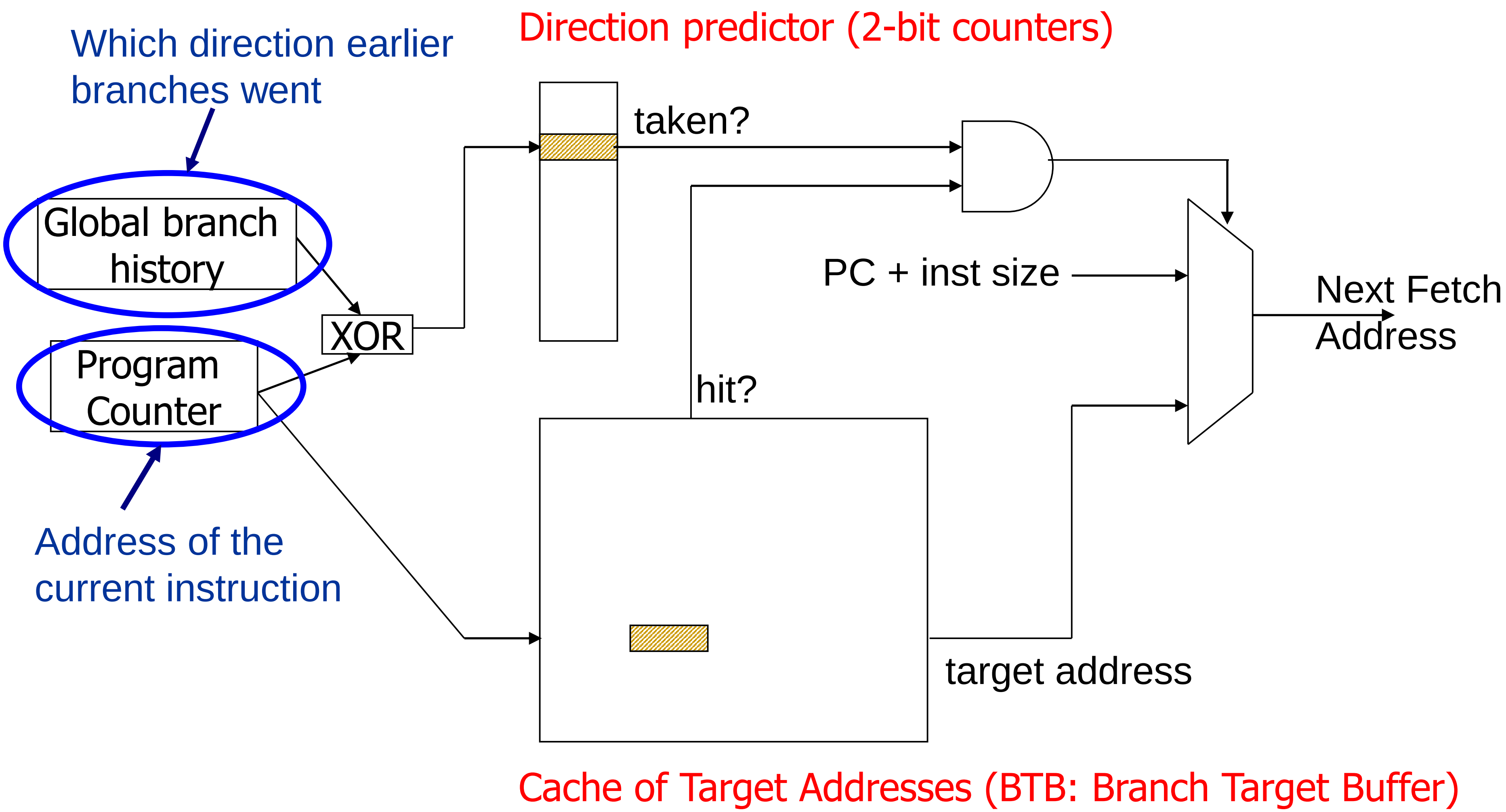
Review: One-Level Branch Predictor



Two-Level Global History Branch Predictor



Two-Level Gshare Branch Predictor



Can We Do Better?

- 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

Local Branch Correlation

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

- If loop test done at end of for loop, that branch will execute the pattern $(1110)^n$
- 1 and 0 show taken and not-taken
- If we know the direction of the branch in previous 3 execution, we can accurately predict the next branch direction

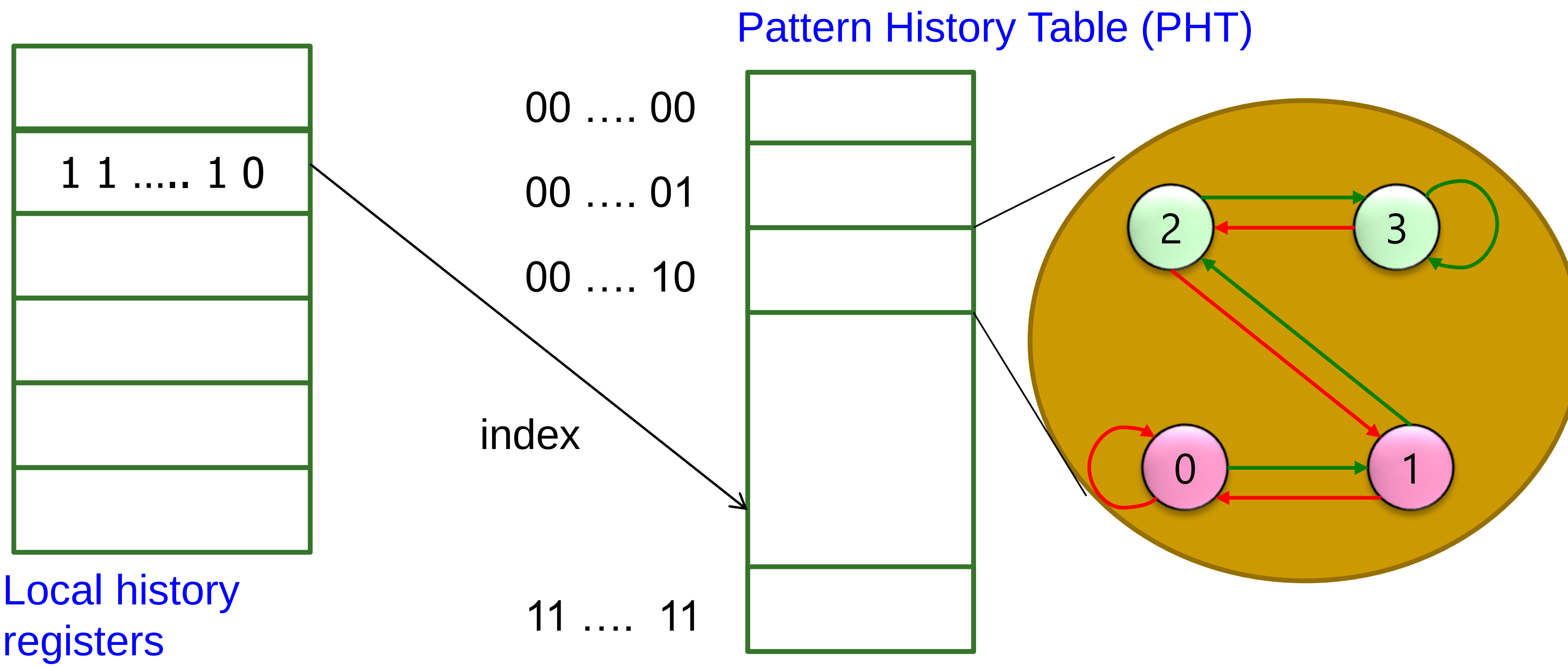
Capturing Local Branch Correlation

- 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)

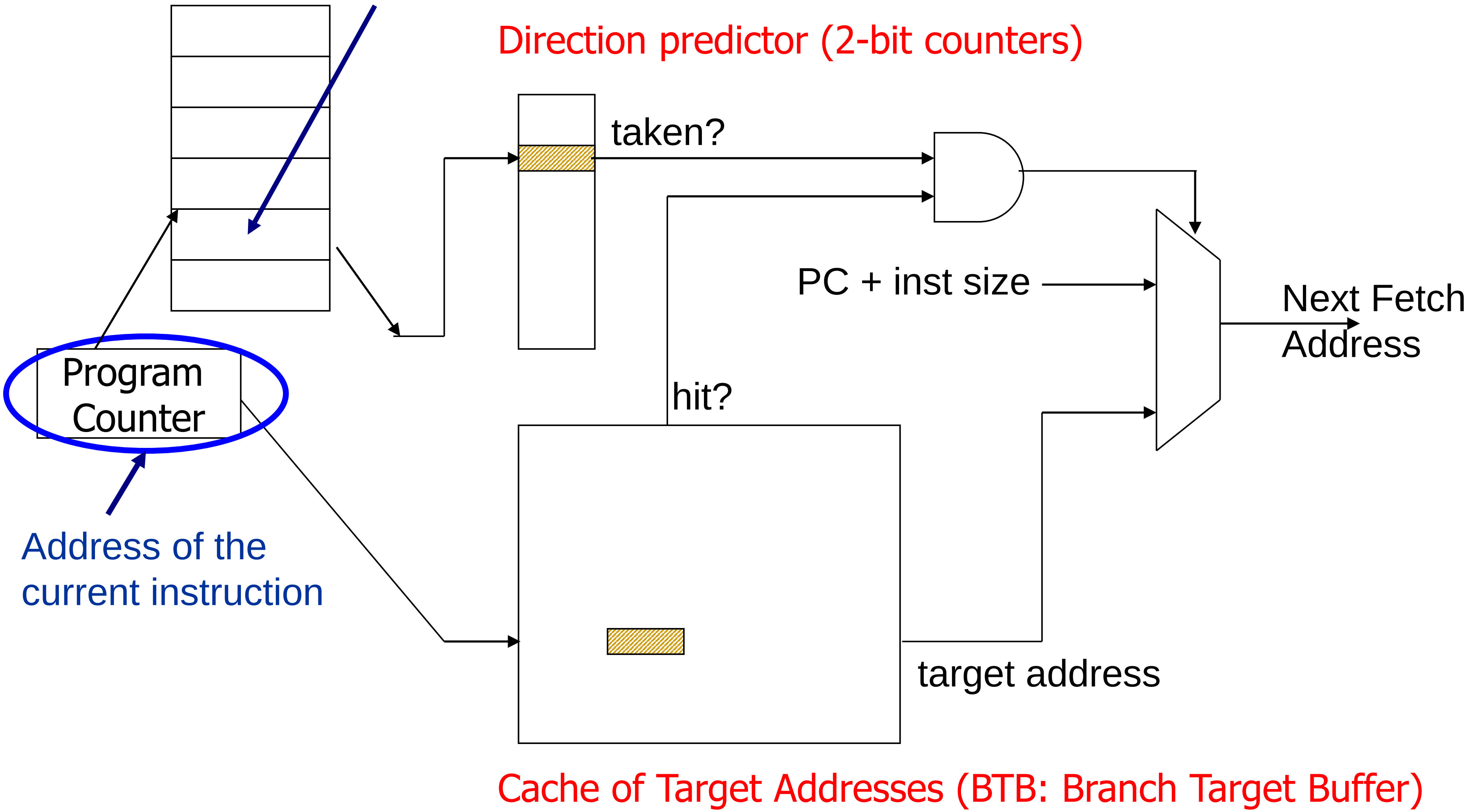
Two Level Local Branch Prediction

- 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



Two-Level Local History Branch Predictor

Which directions earlier instances of *this branch* went



Understanding diff b/w local & global correlation

Consider an array[N] which is filled with truly random integers
Now consider the following program.

```
for (int i=0; i<N; i++) { /* B1 */
val = array[i];
if (val % 20 == 0) { /* B2 */
sum += val;}
if (val % 7 == 0) { /* B3 */
sum += val;}
if (val % 140 == 0) { /* B4 */
sum += val;}
}
```

Understanding diff b/w local & global correlation

- Is any of these branches (B1 to B4) locally correlated? If yes, which one(s)?
- Ans: Only B1.
- Explanation: B2, B3, B4 are not locally correlated. Similar to successive outcomes of a die, an element being a multiple of N (N is 20, 7, and 140, respectively for B2, B3, and B4) has no connection on whether the next element is also a multiple of N.
- Is any of these branches (B1 to B4) globally correlated? If yes, which one(s)?
- B4 is correlated with B2 and B3. 140 is a common multiple of 20 and 7.

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

Biased Branches

- 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, …)
- Do not put their information in the branch history register

References

- S. Mittal, "**A Survey of Techniques for Dynamic Branch Prediction**" *Concurrency and Computation: Practice and Experience, 2018.* (PDF)