COMP3710 (Class # 5176)
Special Topics in Computer Science
Computer Microarchitecture

Convener: Shoaib Akram

shoaib.akram@anu.edu.au

Australian National University

Plan

Week 4: Data and branch hazards, branch prediction

Week 4: Correlating predictors (via an example)

Week 5: Hybrid, Neural, and Tag-based predictors

Week 5: BTBs, Exception handling, Multiscalar Pipelines

Week 5: Move towards Out-of-Order

Reasons for Mispredictions

Unseen (cold) branches → training time

- *Relearning due to phase behavior*
- *2^n history patterns for a BHR of size n*

Randomized/cryptographic algorithms

Lack of information

- Global history cannot see local correlation
- Not enough history bits

Negative interference/aliasing

- Two branches with opposite bias map to the same entry in the PHT
- Contrast with neutral interference (similar bias)

Types of Aliasing

The three-C's model: Compulsory Capacity Conflict

Compulsory aliasing

- First use of address-history pair (approx. 1% mispredictions)

Capacity aliasing

- Size of working set is greater than the size of PHT
- 128-entry PHT, 129 branches in the program

Conflict aliasing

- Two different address-history pairs map to the same PHT entry
- 128-entry PHT, 2 branches in the program with addresses 131 and 259

Interference-Reducing Predictors

The next two predictors try to reduce the negative interference due to a shared PHT

gskewed Predictor

Bi-Mode Predictor

Branch Filtering

gskewed

Operation

- Divide the PHT into multiple banks
- Each bank is indexed with a different hash
- Combine the results with a majority function
- Total update: update all PHTs with the correct outcome
- Partial update: Do not update the mispredicted bank if overall prediction is correct

Intuition: *If two branch-history pairs conflict in one PHT, then they are unlikely to conflict in the other two PHTs*

Is gshare not sufficient?

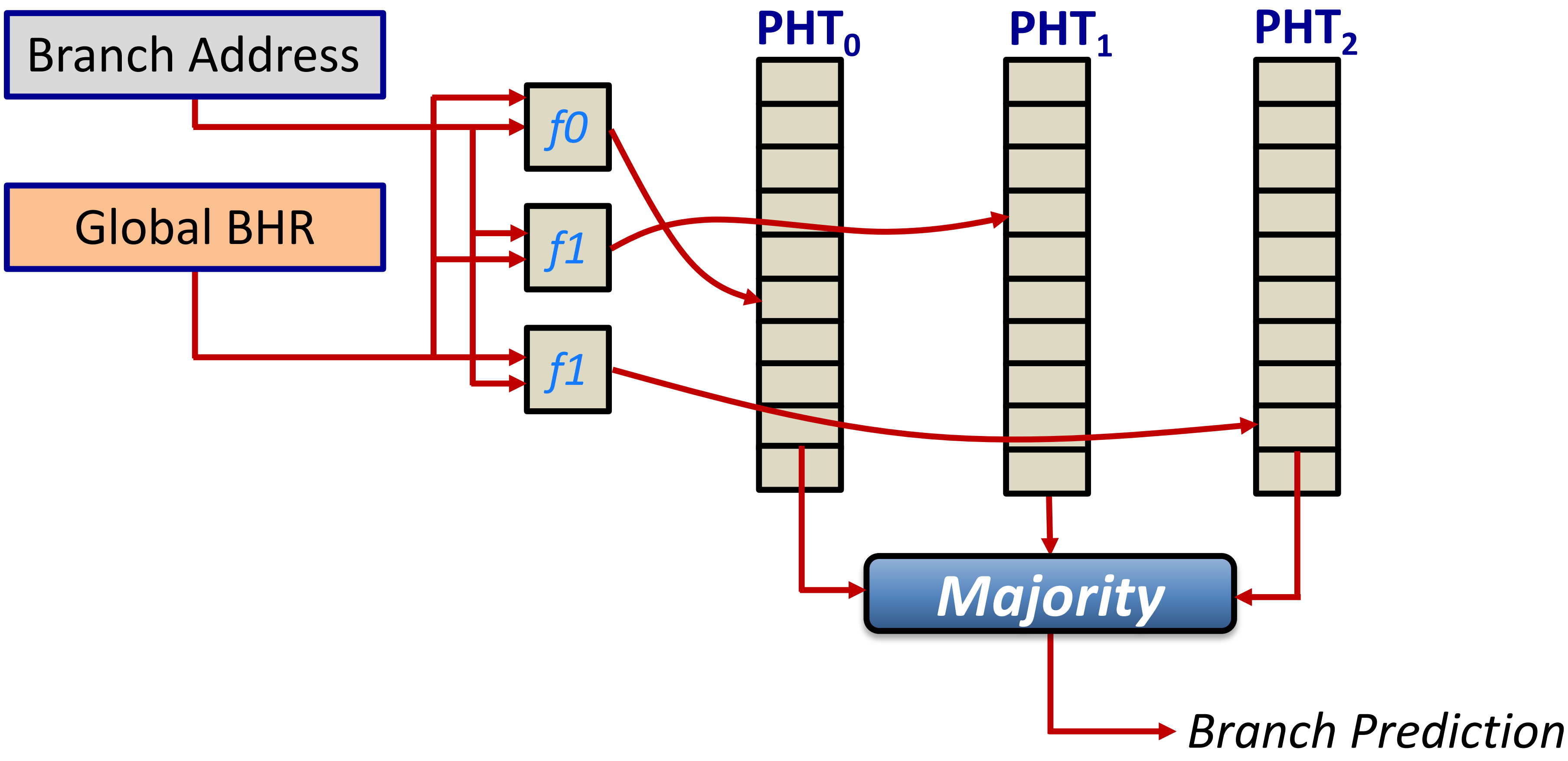
Consider the following branch-history pairs:

BHR	0	1	1	0
PC	1	1	0	0
Index	1	0	1	0

BHR	1	1	0	1
PC	0	1	1	1
Index	1	0	1	0

gshare

gskewed Predictor



gskewed Predictor

$f_0(x,y) = H(y) \veebar H^{-1}(x) \veebar x$

$f_1(x,y) = H(y) \veebar H^{-1}(x) \veebar y$

$f_2(x,y) = H^{-1}(y) \veebar H(x) \veebar x$

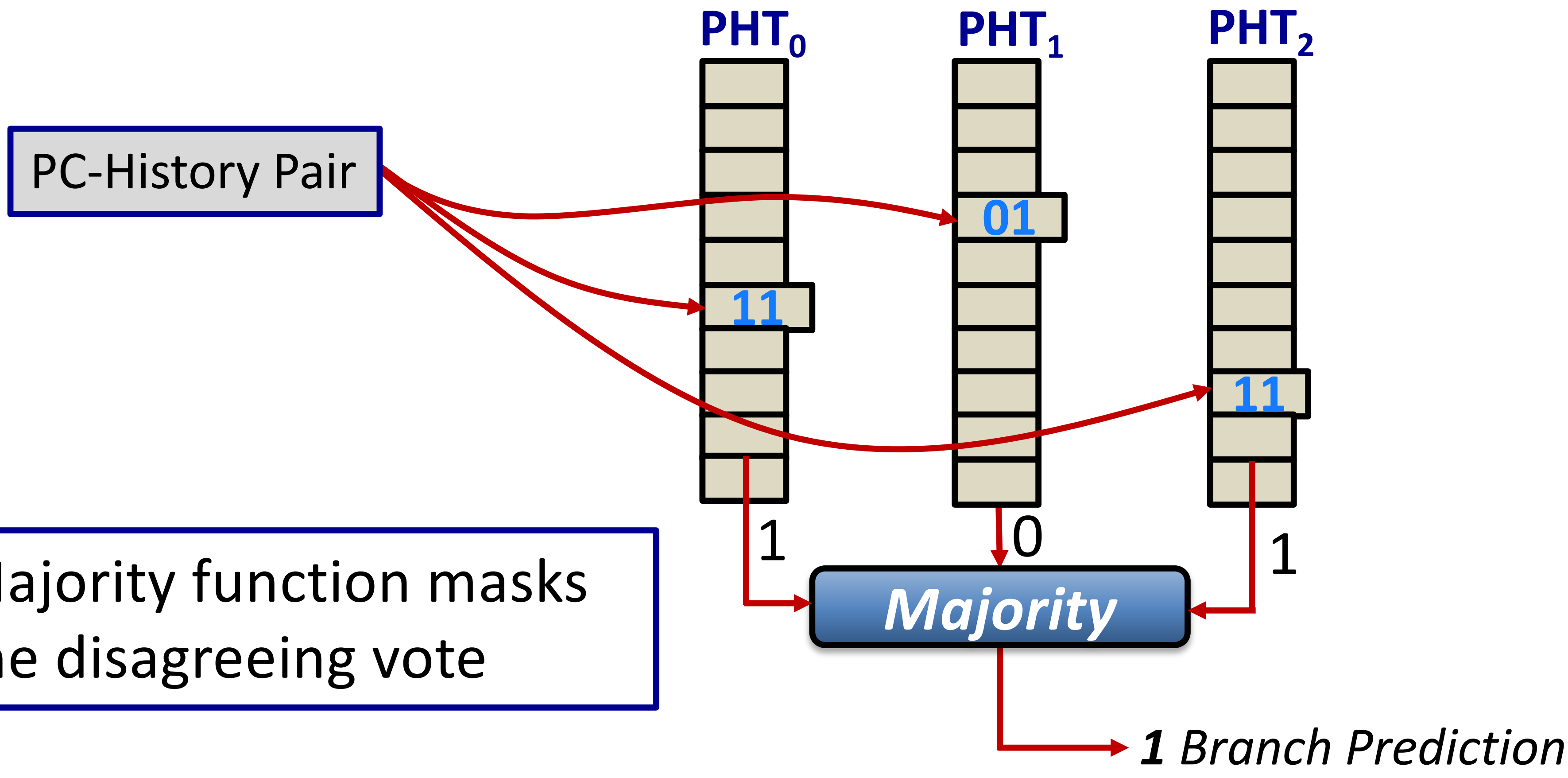
- (x,y) are n-bits long
- H^{-1} is inverse of H

$f_0(x1) = f_0(x2)$ then $f_1(x1)$ **!=** $f_1(x2)$ and $f_2(x1)$ **!=** $f_2(x2)$

$H(b_n, b_{n-1},\ldots,b_3,b_2,b_1) = (b_n \text{ XOR } b_1, b_n, b_{n-1},\ldots,b_3,b_2)$

gskewed Predictor

Used in Alpha EV8
Never realized



Majority function masks the disagreeing vote

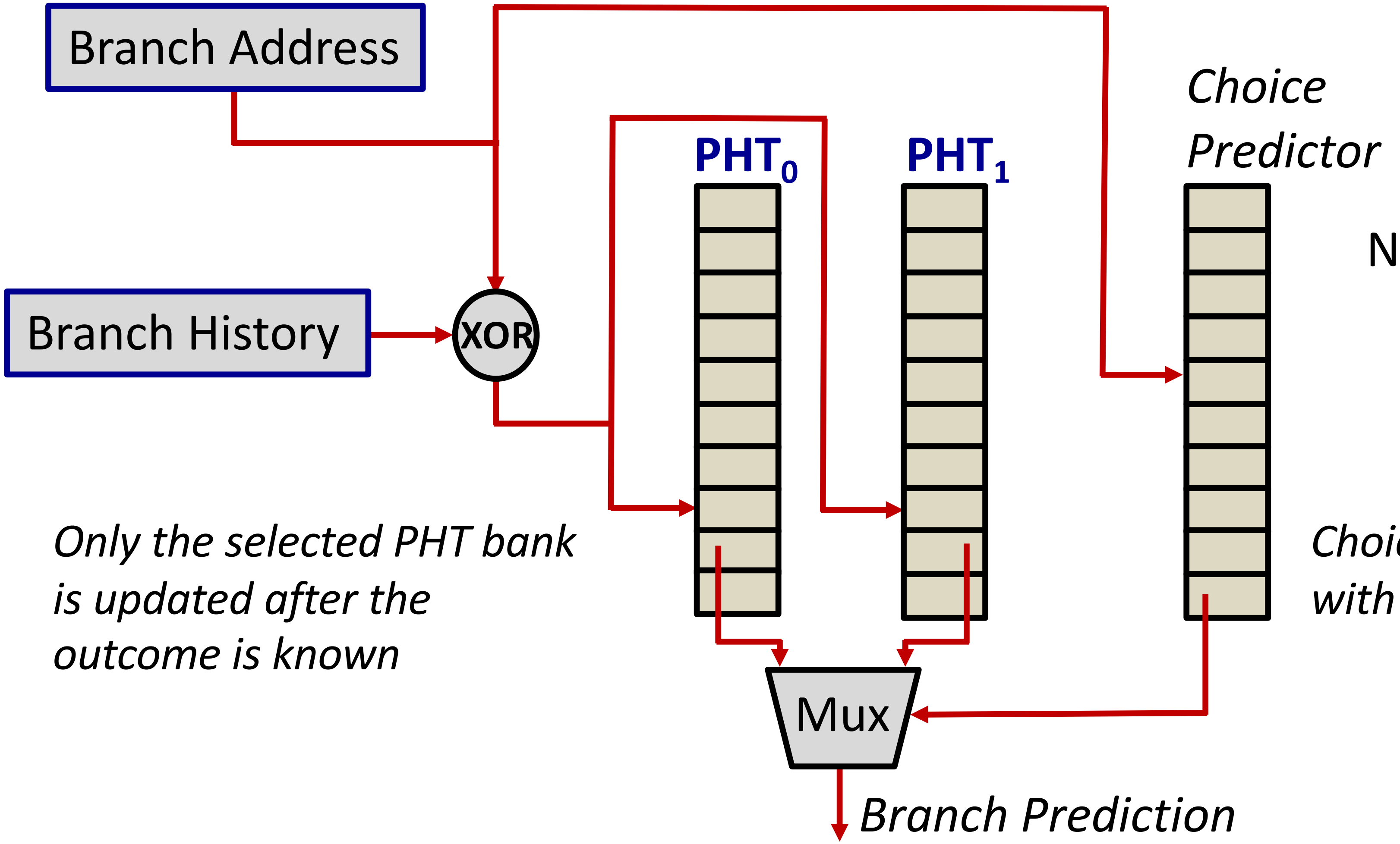
Bi-Mode Predictor

Operation

- Split branches into two groups (ST and SNT)
- Use two PHTs (direction predictors) and index with the same address-history hash
- ST branches map to one PHT, and SNT branches to the other
- A meta-predictor (*choice predictor*) selects the PHT bank
- Index the choice predictor with the branch address

Intuition: *Branches have a bias (ST or SNT). Separating them into two PHT mitigates –ve interference. If two branches map to the same entry in the PHT, they are unlikely to harm each other*

Bi-Mode Predictor



NNNNNNNTNNTN

- Which PHT?
- Entries of PHT?

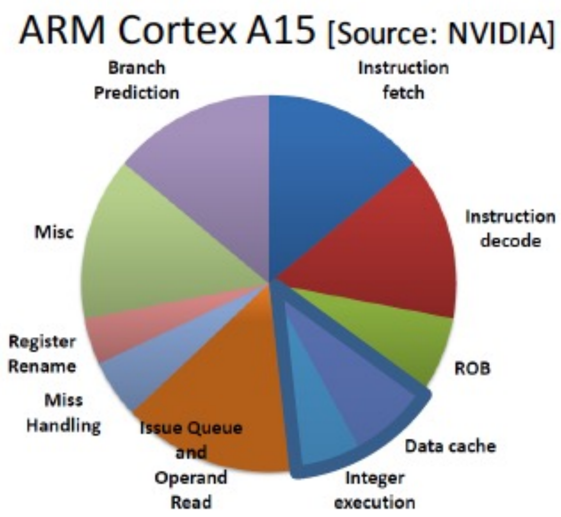
Only the selected PHT bank is updated after the outcome is known

Choice predictor is updated with the branch outcome

Bi-Mode Predictor

15% of Cortex A15 power

Sizing more complicated as one needs to tune PHT sizes and that of the choice predictor



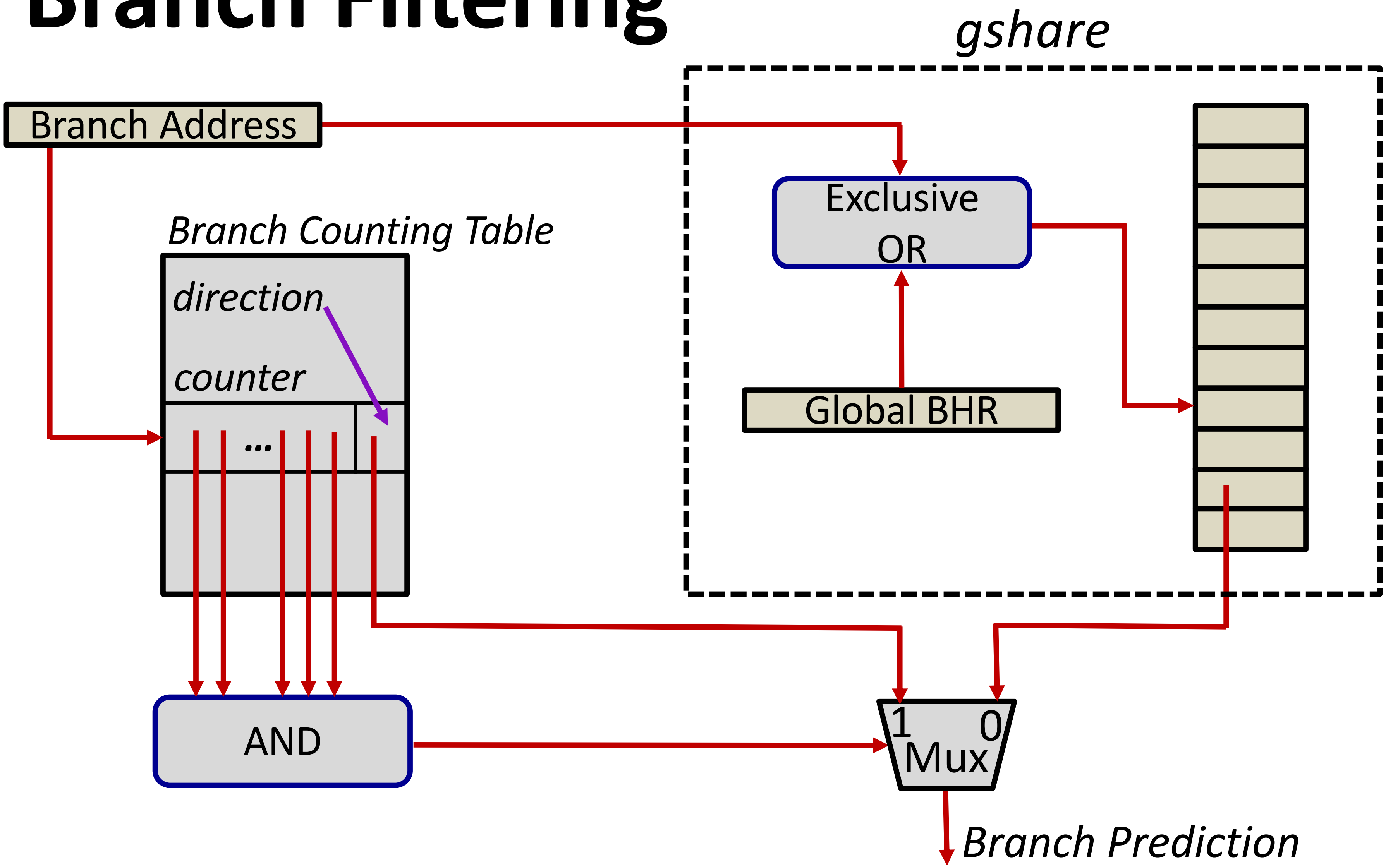
Branch Filtering

Intuition: *Reduce the # branches stored in the PHT by removing highly biased branches from the PHT*

Operation

- Track how many times a branch has gone in the same direction
- Beyond a threshold, a branch is “filtered” and no longer updates the PHT
- If the direction changes, reset the counter, and note the new direction

Branch Filtering



Alternative Context Predictors

Tradeoffs in choosing the branch prediction context

- Local or global history
- Length of branch history register
- How many bits of the branch address?

Motivation: *Can we combine all of the above into a single context? Can we use per-branch-type information? Can we use additional information to form context?*

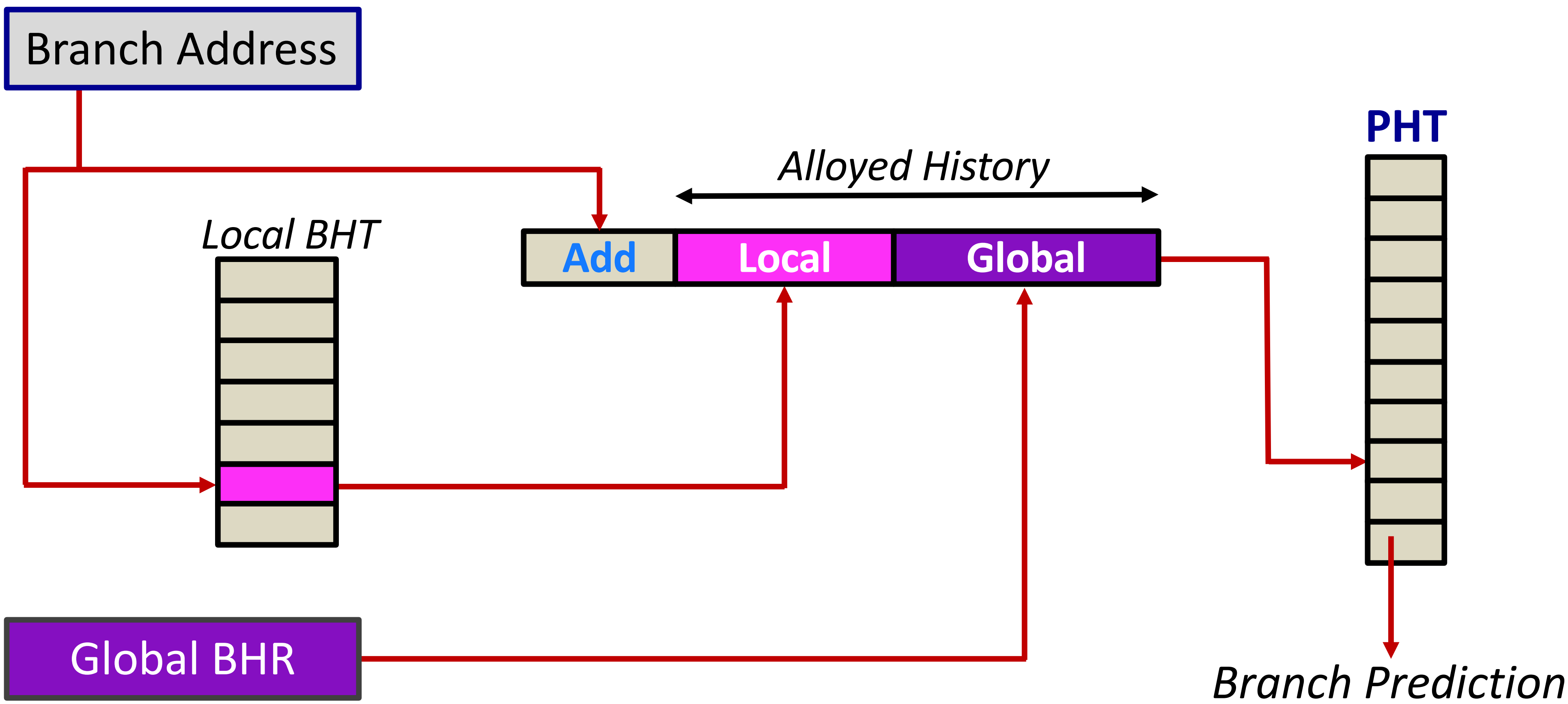
Alloyed History Predictor

Some mispredictions are due to

- Wrong type of history (*wrong-history misprediction*)
- Some branches prefer local, some global, and some both

Motivation: *Distinguish the local and global correlations with the same structure*

Alloyed History Predictor



Loop Counting Predictors

If we want to accurately predict loops, what size BHR do we need for a loop that iterates n times?

PHT size is exponential in the history length

Loop predictor in Pentium M

- # iterations (limit)
- Current count
- Direction
- Can detect 11101110 and 00010001

Sum: Assembly

Example of NNNNNNNNNNT pattern

C code:

```c
int i;
int sum = 0;

for (i = 0; i < 10; i = i + 1) {
    sum = sum + i;
}
```

ARM Assembly code

; R0 = i, R1 = sum

```
        MOV     R1,     #0
        MOV     R0,     #0
FOR
        CMP     R0,     #10
        BGE     DONE
        ADD     R1,     R1,     R0
        ADD     R0,     R0,     #1
        B       FOR
DONE
```

check termination condition to break out of the loop if condition is met (BGE DONE)

keep iterating by branching back (B FOR)

Sum: Alternative Approach

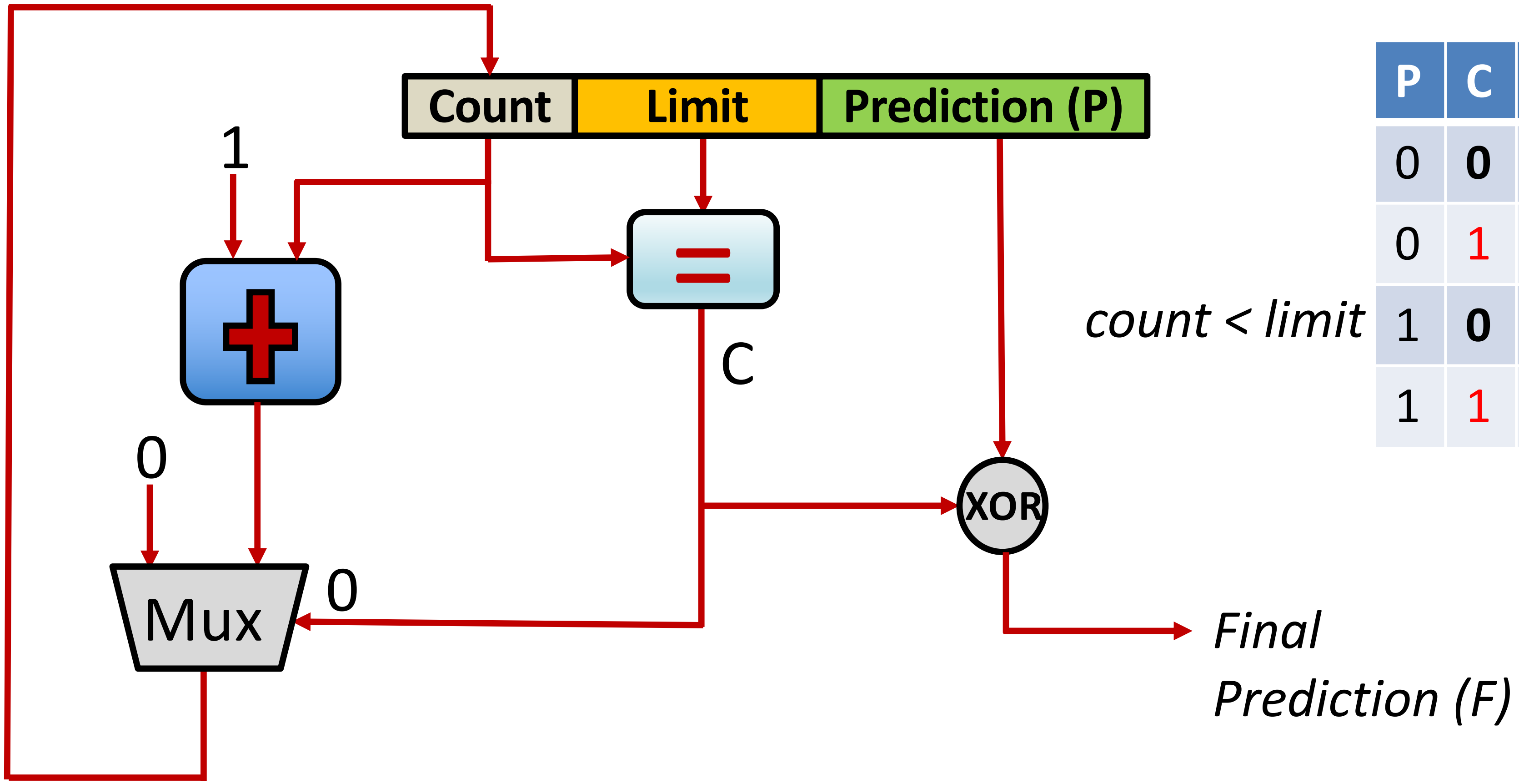
```
        MOV     R1,     #0
        MOV     R0,     #0
COND
        CMP     R0,     #10
        BLT     FOR
        B       DONE
FOR
        ADD     R1,     R1,     R0
        ADD     R0,     R0,     #1
        B       COND
DONE
```

Example of TTTTTTTTTTN pattern

- More faithfully follow the for loop semantics in C
- Use BLT instead of BGE

Loop Counting Predictors

The Pentium-M Loop Predictor Table (One entry)



count < limit

P	C	F
0	**0**	0
0	1	1
1	**0**	1
1	1	0

The Perceptron Predictor

Motivation: Increasing the # history bits

- Exponentially increases the PHT size
- Many patterns are irrelevant (training noise)

Question: Can we use more history bits without the exponential increase in area?

- Use perceptron for training the branch predictor
- Use branch history as a feature vector (*not index*)

https://www.youtube.com/watch?v=5g0TPrxKK6o&ab_channel=Udacity

Perceptron

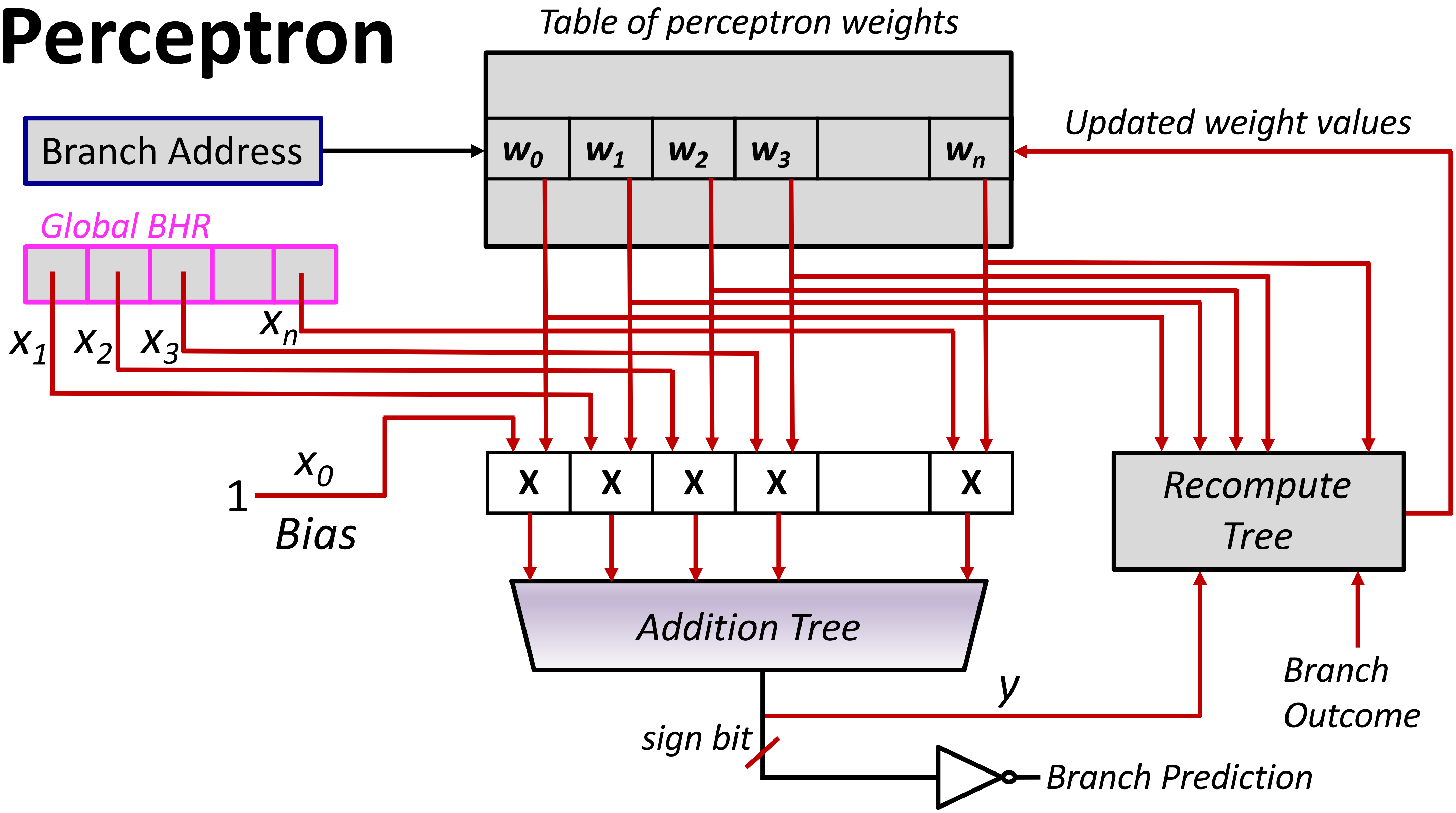


Table of perceptron weights
Branch Address
w0
w1
w2
w3
wn
Updated weight values
Global BHR
x1
x2
x3
xn
x0
1
Bias
X
X
X
X
X
Recompute
Tree
Addition Tree
y
sign bit
Branch
Outcome
Branch Prediction

Hybrid Branch Predictors

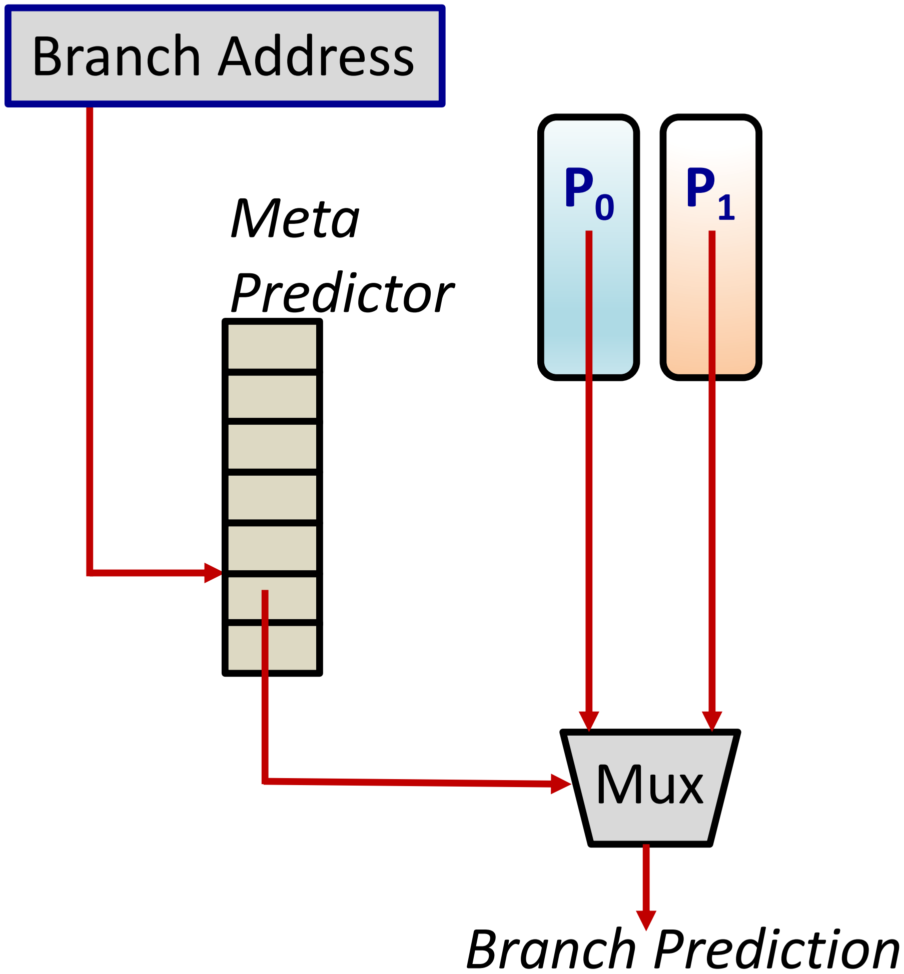
Motivation: *Programs contain a mix of branch types. Different branches may be strongly correlated with different types of history (i.e., global vs local)*

Hybrid branch predictors employ two or more single-scheme branch prediction algorithms

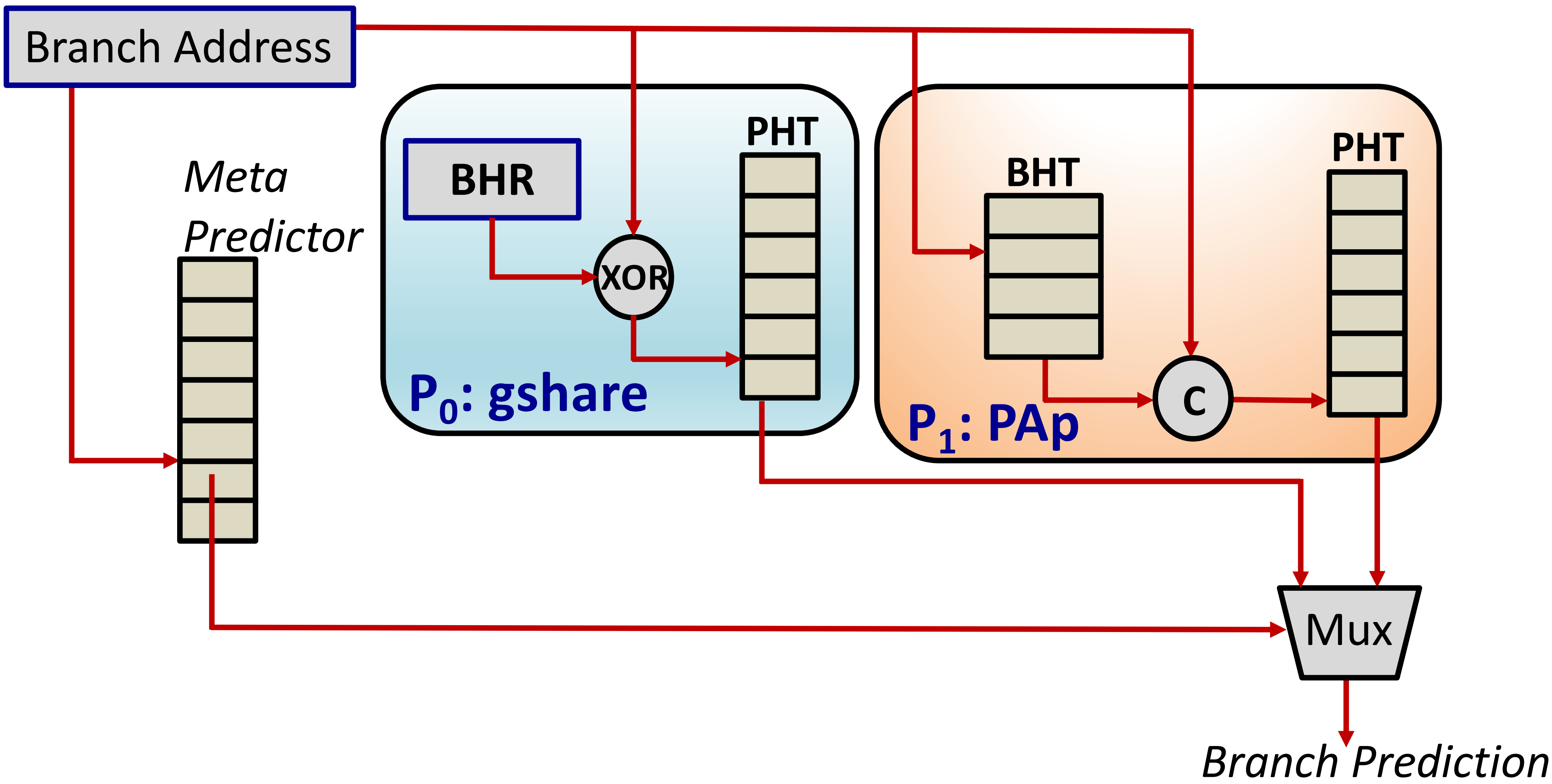
- Combine multiple predictions to make one final prediction

McFarling (1993) proposed the multi-scheme **tournament predictor**

Tournament Predictor



Tournament Predictor

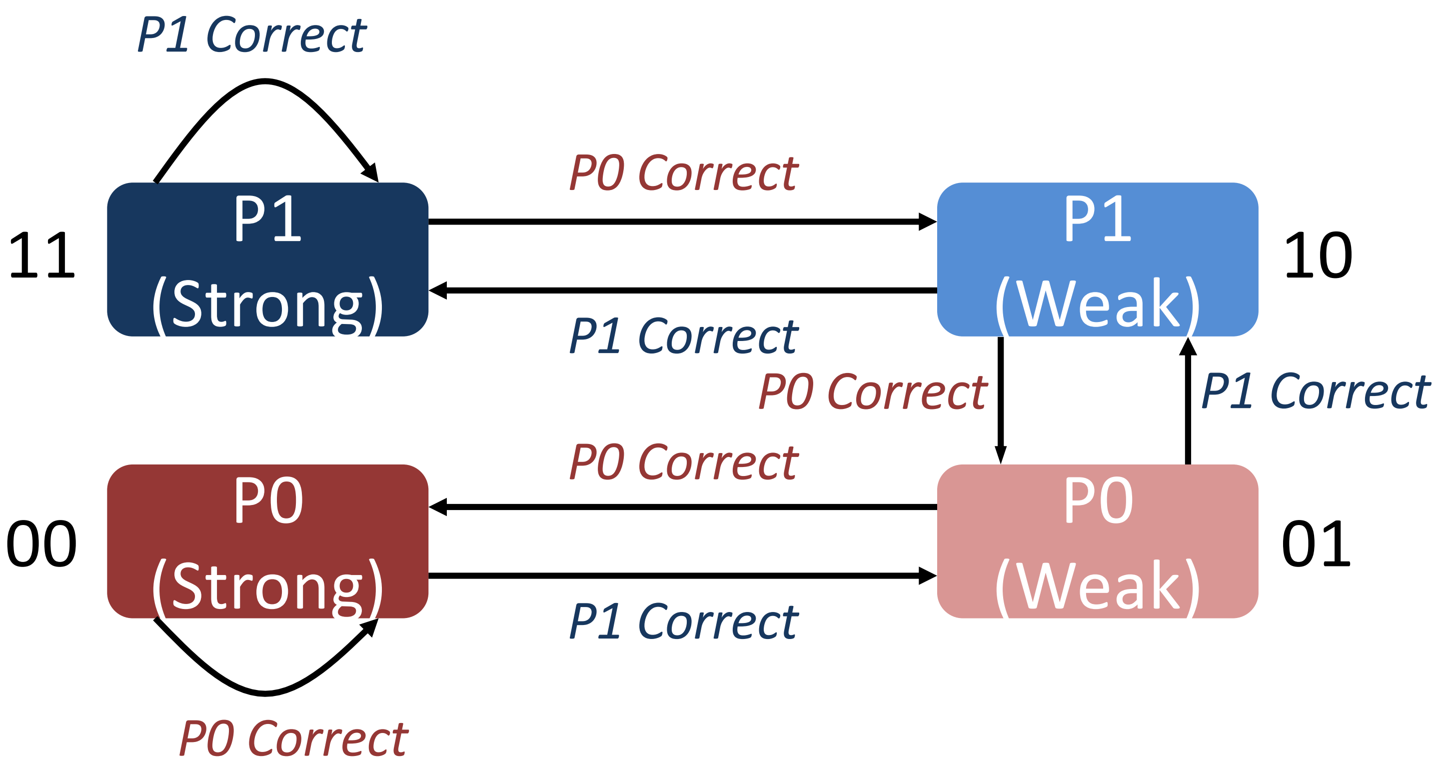


Operation

- After the branch outcome is available, P_0 and P_1 are updated according to their respective update rules
- The meta-predictor is structurally identical to $Smith_2$, the update rules (state transitions) are different
- The meta-predictor is indexed by the low-order bits of the branch address
  - It makes a prediction which predictor will be correct
- Meta prediction of 0 indicates that P_0 should be used
- Meta prediction of 1 indicates that P_1 should be used
- Meta-prediction is made from the most significant bit of the counter

Tournament Meta-Predictor

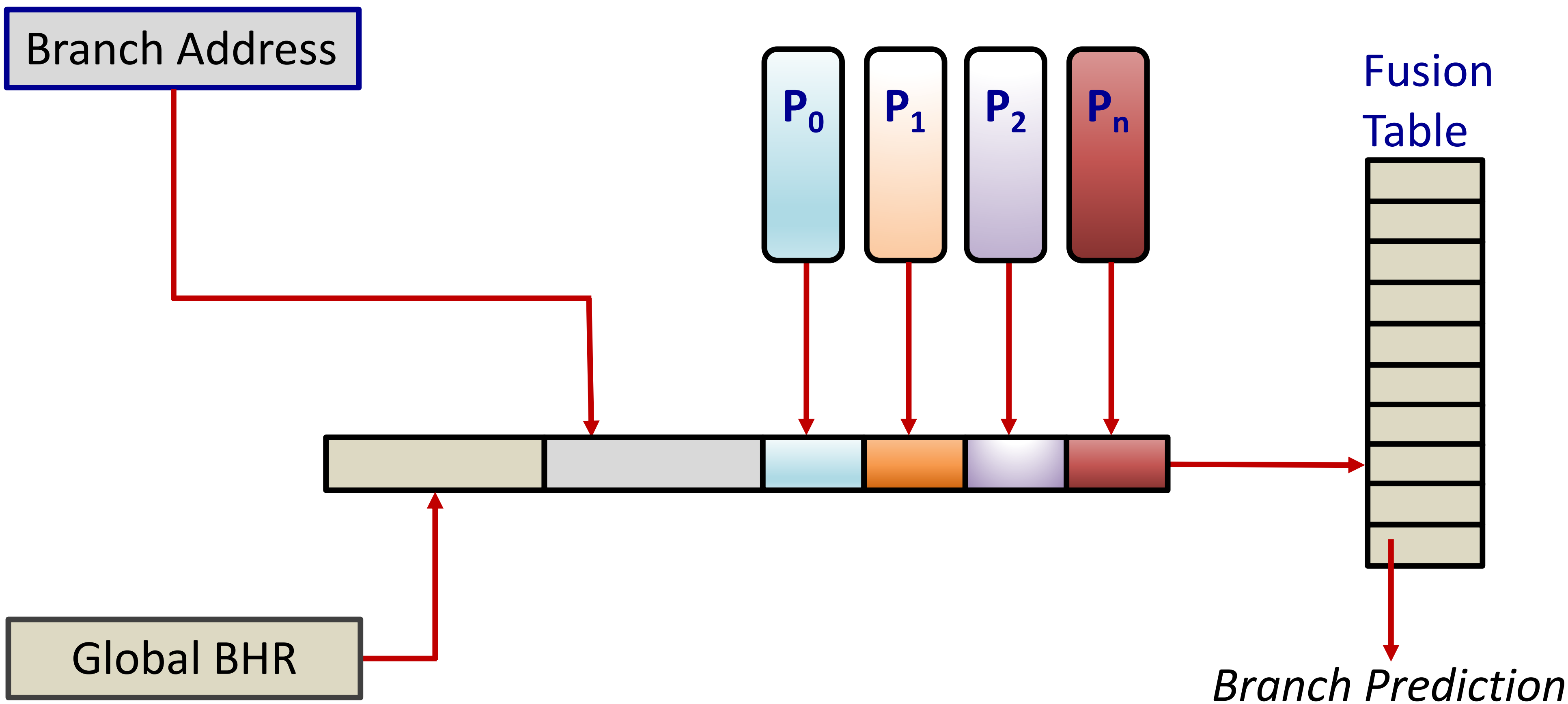
P0/P1 both correct/incorrect: state unchanged



Fusion-Based Hybrid Predictor

Motivation: *Do not throw away the output from any predictor*

Fusion-Based Hybrid Predictor



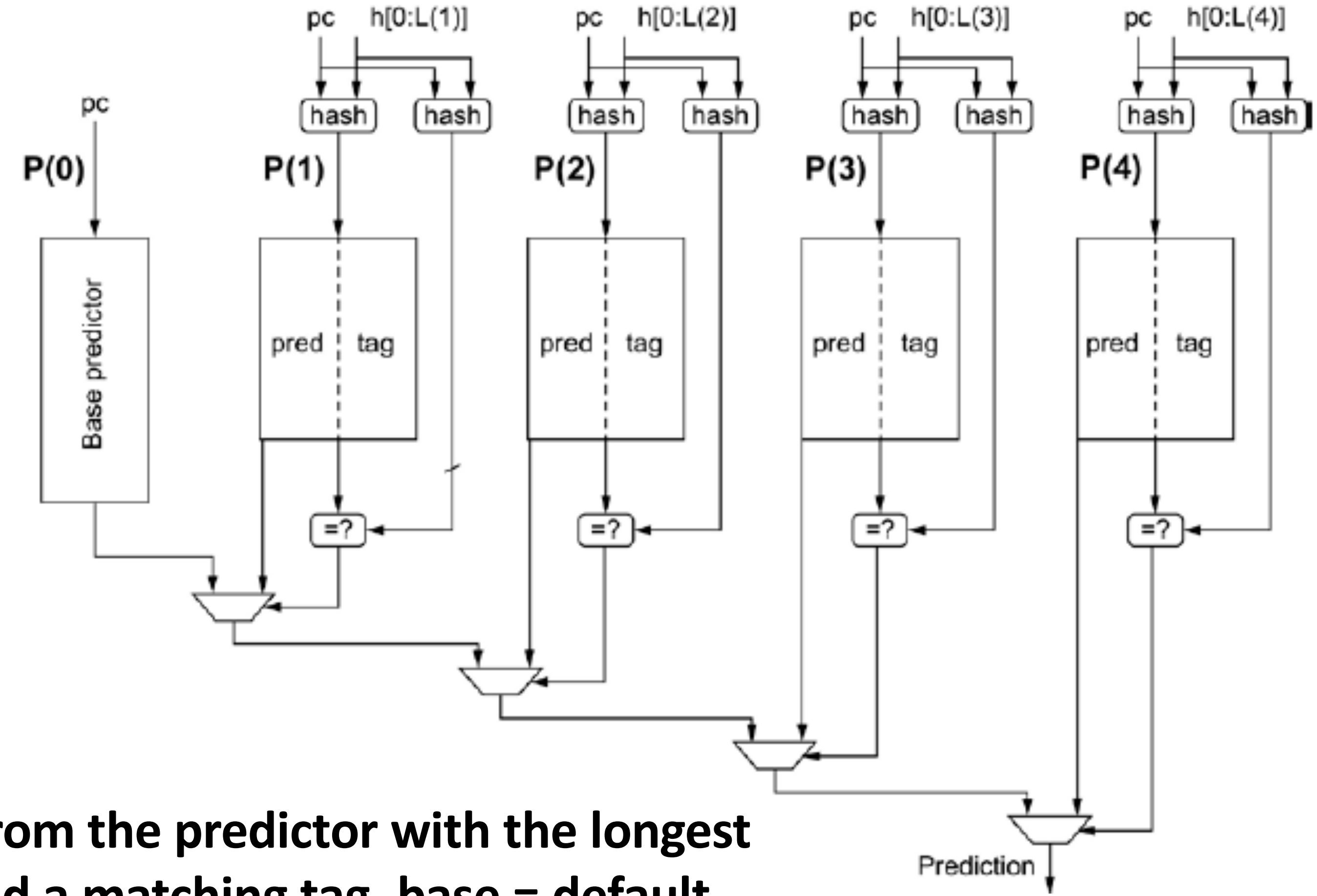
TAGE

TAgged **GE**ometric Predictors
(state-of-the-art)

Two key innovations

- Use multiple history lengths
  - *History lengths make a geometric series*
- Use tags to alleviate aliasing

Tagged Hybrid Predictors



Take prediction from the predictor with the longest branch history and a matching tag, base = default.