



고려대학교
KOREA UNIVERSITY

COM515 Advanced Computer Architecture

Lecture 4. Branch Prediction

Prof. Taeweon Suh

Computer Science Education

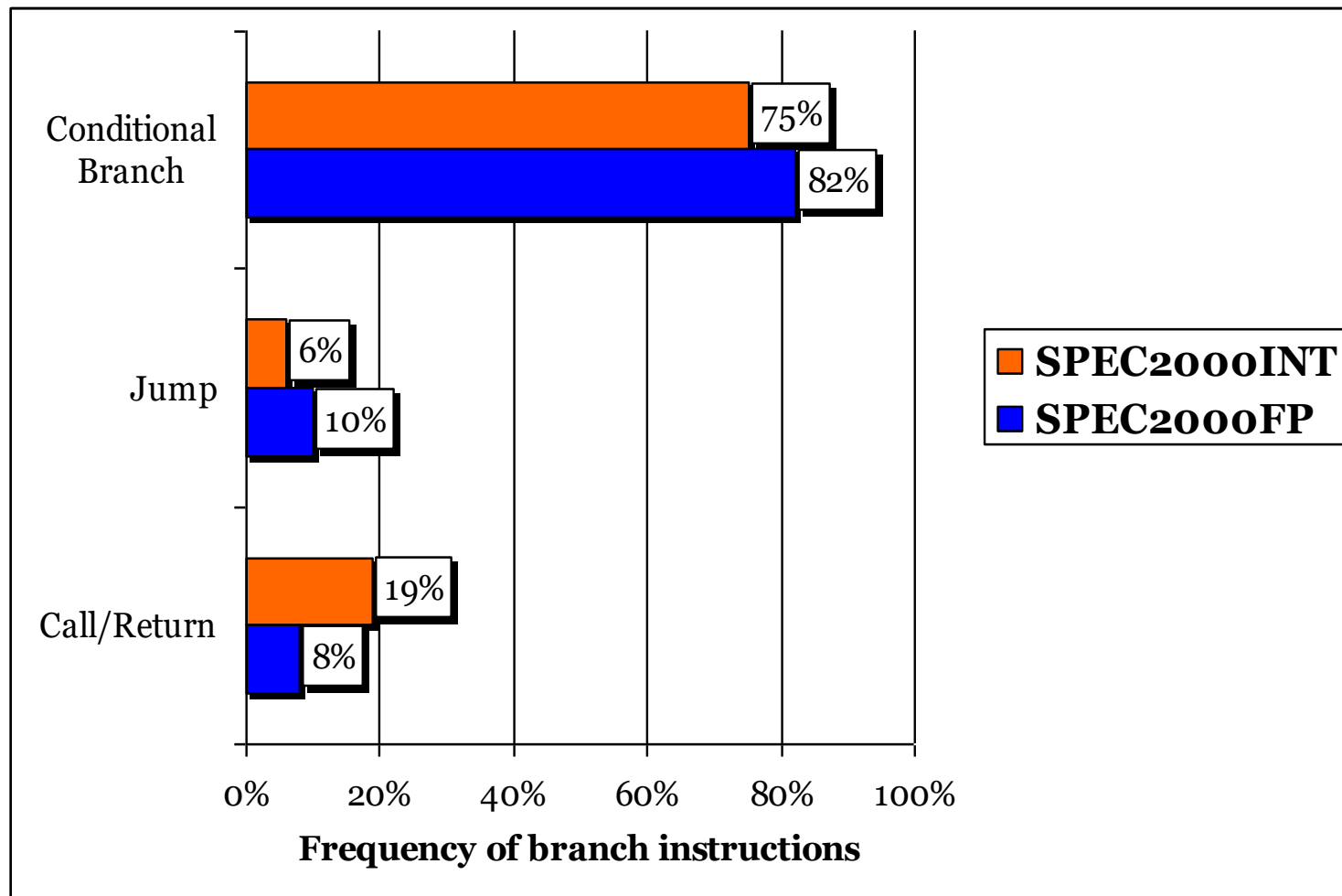
Korea University

Predict What?



- Direction (1-bit)
 - Single direction for unconditional jumps and calls/returns
 - Binary for conditional branches
- Target (32-bit or 64-bit addresses)
 - Some are easy
 - One: Uni-directional jumps
 - Two: Fall through (Not Taken) vs. Taken
 - Many: Function Pointer or Indirect Jump (e.g. jr r31)

Categorizing Branches



Source: H&P using Alpha

Branch Misprediction

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

PC Next	PC Fetch	Drive	Alloc	Rename	Queue	Schedule	Dispatch	Reg File	Exec	Flags	Br	Resolve
---------	----------	-------	-------	--------	-------	----------	----------	----------	------	-------	----	---------

Single Issue

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Branch Misprediction

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

PC Next	PC Fetch	Drive	Alloc	Rename	Queue	Schedule	Dispatch	Reg File	Exec	Flags	Br	Resolve
---------	----------	-------	-------	--------	-------	----------	----------	----------	------	-------	----	---------

Single Issue



↑
Mispredict

Branch Misprediction

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

PC Next	PC Fetch	Drive	Alloc	Rename	Queue	Schedule	Dispatch	Reg File	Exec	Flags	Br	Resolve
---------	----------	-------	-------	--------	-------	----------	----------	----------	------	-------	----	---------

Single Issue (flush entailed instructions and refetch)



↑
Mispredict

Branch Misprediction

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

PC Next	PC Fetch	Drive	Alloc	Rename	Queue	Schedule	Dispatch	Reg File	Exec	Flags	Br	Resolve
---------	----------	-------	-------	--------	-------	----------	----------	----------	------	-------	----	---------

Single Issue



Fetch the correct path

Branch Misprediction

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

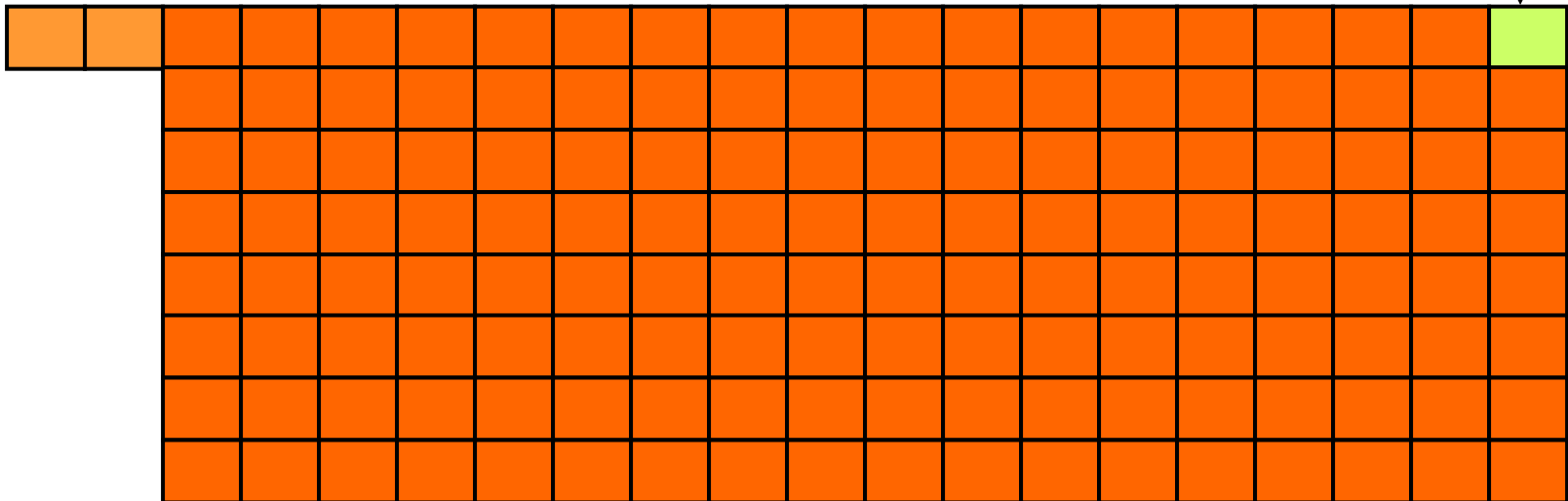
PC Next	PC Fetch	Drive	Alloc	Rename	Queue	Schedule	Dispatch	Reg File	Exec	Flags	Br	Resolve
---------	----------	-------	-------	--------	-------	----------	----------	----------	------	-------	----	---------

Single Issue



Mispredict

8-issue Superscalar Processor (Worst case)



Why Branch is Predictable?

```
for (i=0; i<100; i++) {  
    ....  
}
```

```
    addi r10, r0, 100  
    addi r1, r0, r0  
  
L1: ... ..  
    ... ..  
    addi r1, r1, 1  
    bne r1, r10, L1  
    ... ..
```

```
if (aa==2)  
    aa = 0;  
if (bb==2)  
    bb = 0;  
if (aa!=bb)  
    ....
```

```
    addi r2, r0, 2  
    bne r10, r2, L_bb  
    xor r10, r10, r10  
    j    L_exit  
  
L_bb: bne r11, r2, L_xx  
    xor r11, r11, r11  
    j    L_exit  
  
L_xx: beq r10, r11, L_exit  
    ...  
L_exit:
```

Control Speculation

- Execute instruction beyond a branch before the branch is resolved → Performance
- Speculative execution
- What if mis-speculated? need
 - Recovery mechanism
 - Squash instructions on the incorrect path
- Branch prediction: Dynamic vs. Static
- What to predict?

Static Branch Prediction

- Uni-directional, always predict taken
- Backward taken, Forward not taken
 - Need offset information
- Compiler hints with branch annotation
- Static predication is used as a fall-back technique in some processors with dynamic branch when there is not any information for dynamic predictors to use
- Example
 - Pentium 4 introduced static hints to branches
 - Pentium 4 uses it as a fall-back – instruction prefixes can be added before a branch instruction
 - 0x3E – statically predict a branch as taken
 - 0x2E – statically predict a branch as not taken

Simplest Dynamic Branch Predictor

- Prediction based on latest outcome
- Index by some bits in the branch PC
 - Aliasing

```
for (i=0; i<100; i++) {  
    ....  
}
```

```
0x40010100  addi  r10, r0, 100
```

```
0x40010104  addi  r1,  r1, r0
```

```
0x40010108  L1:
```

```
... ..
```

```
... ..
```

```
0x40010A04  addi  r1, r1,  1
```

```
0x40010A08  bne   r1, r10, L1
```

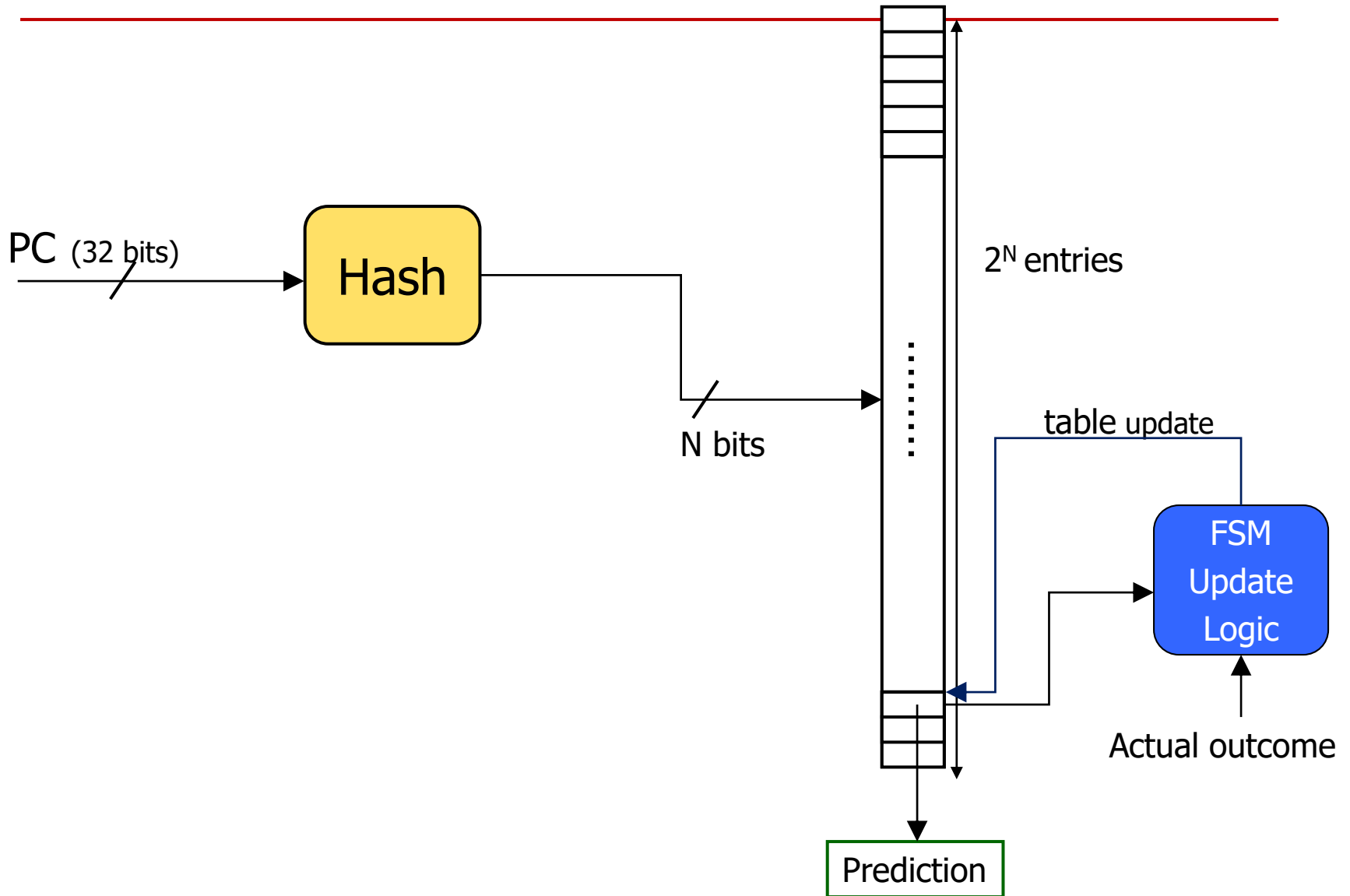
```
... ..
```

NT
T
T
NT
T
.
.
.
T
NT
NT

1-bit
Branch
History
Table

How accurate?

Typical Table Organization



Simplest Dynamic Branch Predictor

```
for (i=0; i<100; i++) {  
    if (a[i] == 0) {  
        ...  
    }  
    ...  
}
```

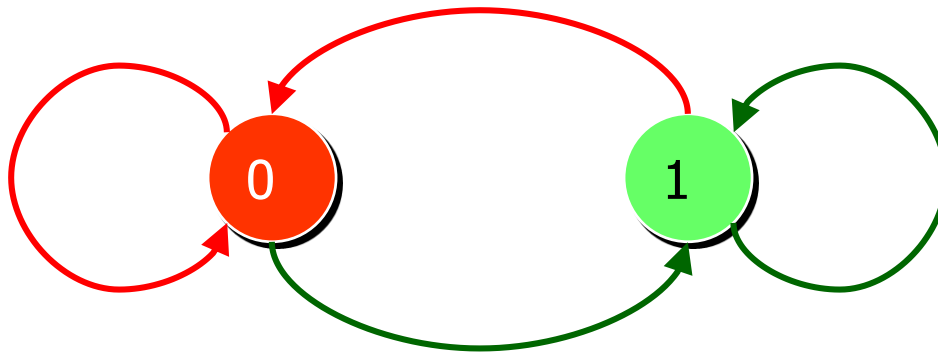
```
0x40010100  addi  r10, r0, 100  
0x40010104  addi  r1,  r1, r0  
L1:  
0x40010108  add   r21, r20, r1  
0x4001010c  lw    r2, (r21)  
0x40010110  beq   r2, r0, L2  
... ..  
0x40010210  j     L3  
L2:  
... ..  
L3:  
0x40010B0c  addi  r1, r1, 1  
0x40010B10  bne   r1, r10, L1
```

NT
T
T
NT
T
.
.
T
NT
NT

1-bit
Branch
History
Table

FSM of the Simplest Predictor

- A 2-state machine
- Change mind fast

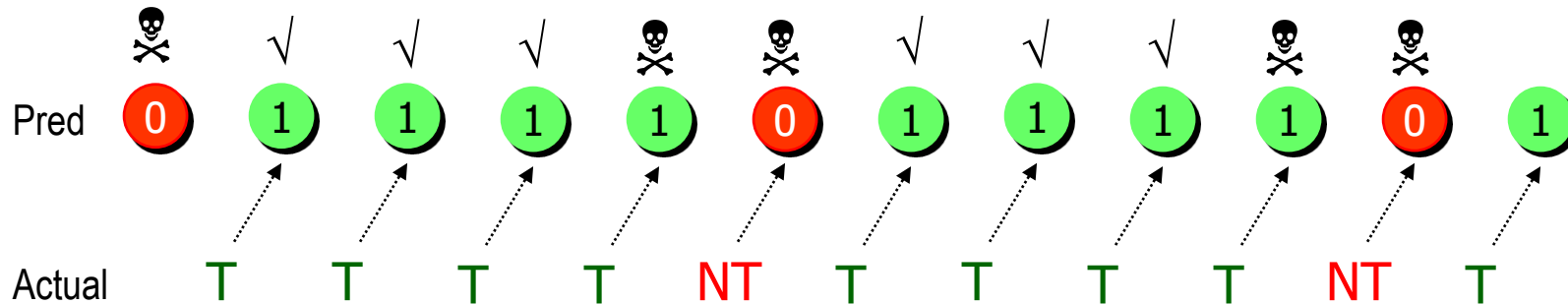


- If branch taken
- If branch not taken
- 0 Predict not taken
- 1 Predict taken

Example using 1-bit branch history table

```
for (i=0; i<4; i++) {  
    ....  
}
```

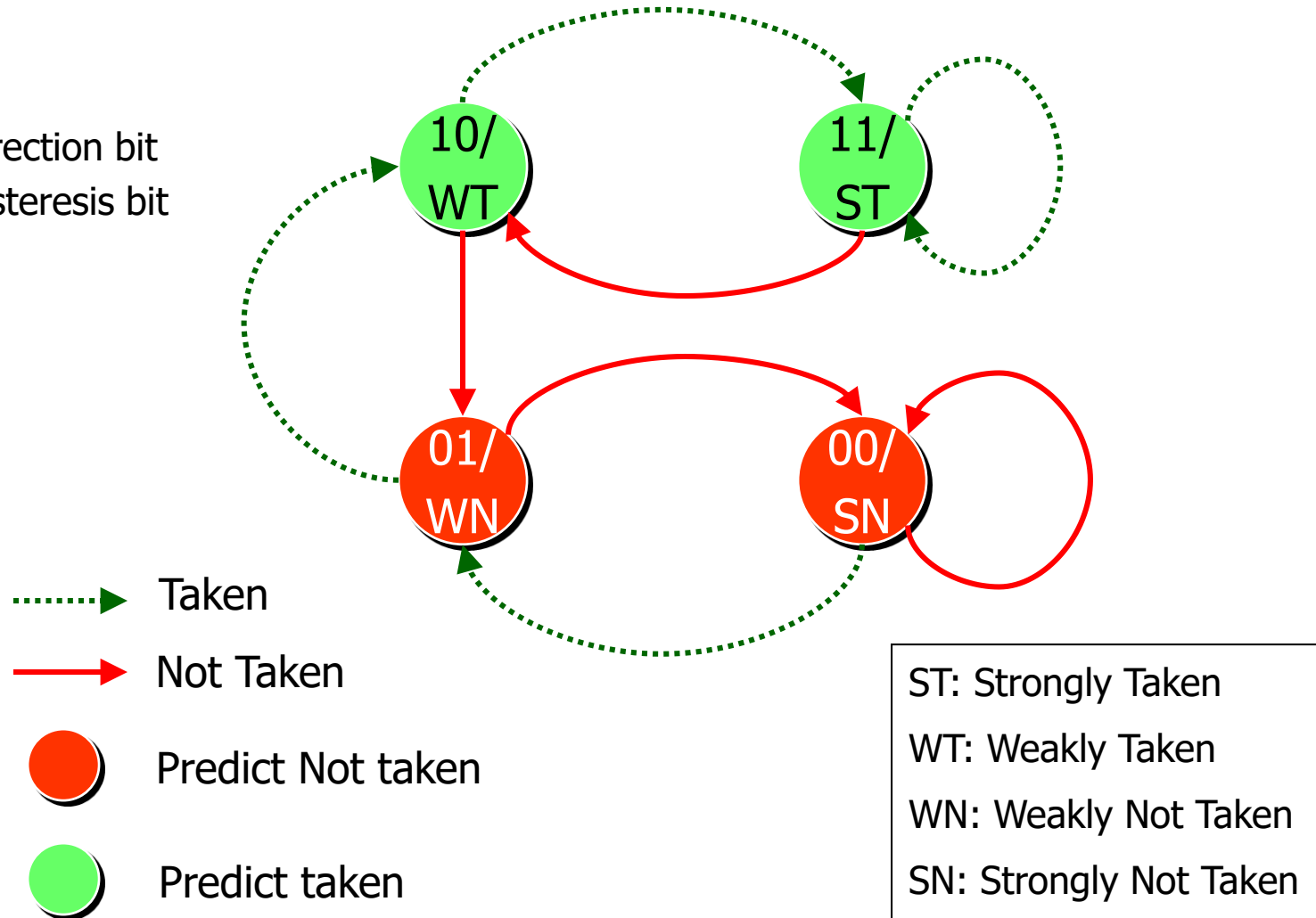
```
addi r10, r0, 4  
addi r1, r1, r0  
L1:  
... ..  
addi r1, r1, 1  
bne r1, r10, L1
```



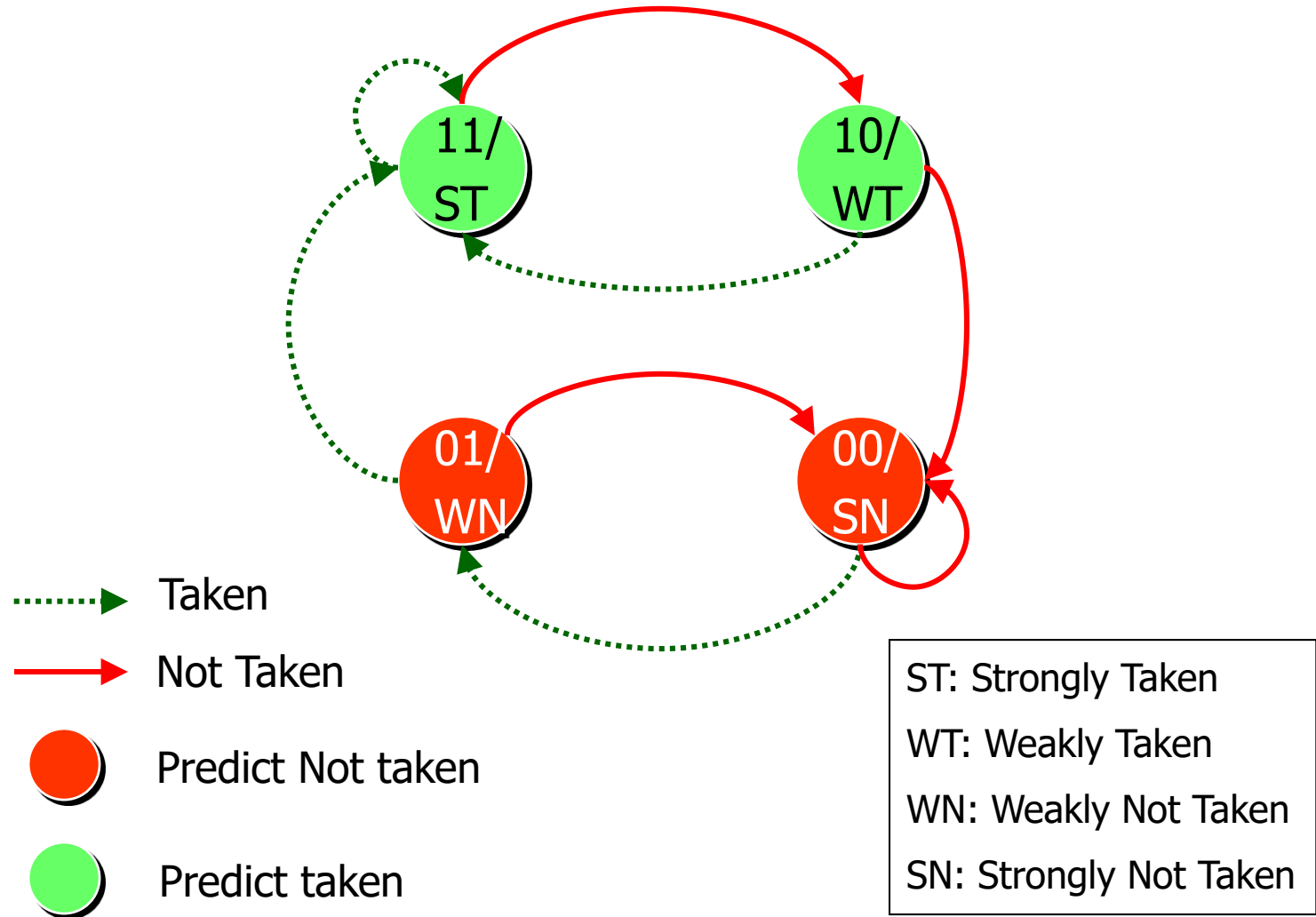
60% accuracy

2-bit Saturating Up/Down Counter Predictor

MSB: Direction bit
LSB: Hysteresis bit



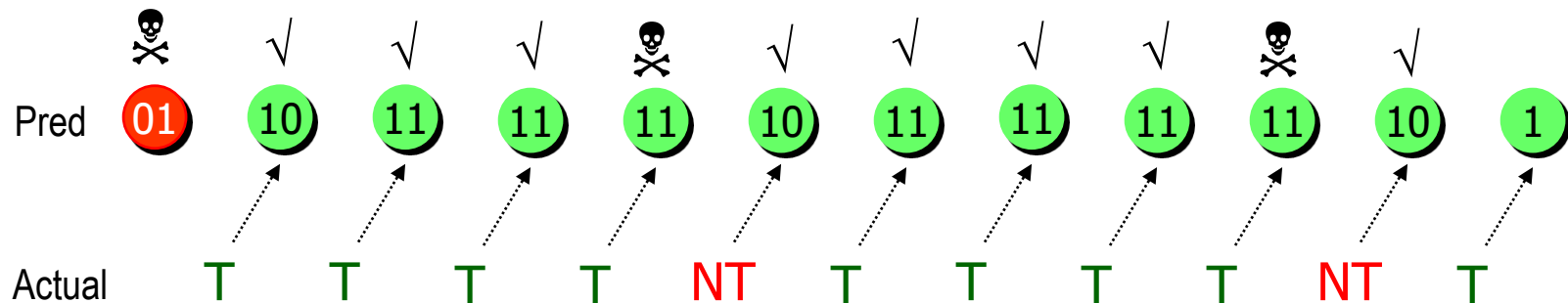
2-bit Counter Predictor (Another Scheme)



Example using 2-bit up/down counter

```
for (i=0; i<4; i++) {  
    ....  
}
```

```
addi r10, r0, 4  
addi r1, r1, r0  
L1:  
... ..  
addi r1, r1, 1  
bne r1, r10, L1
```

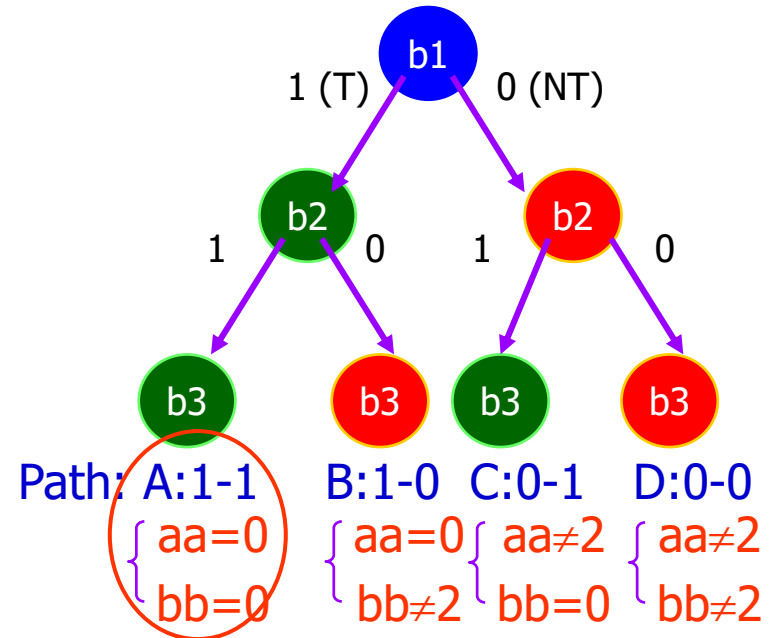


80% accuracy

Branch Correlation

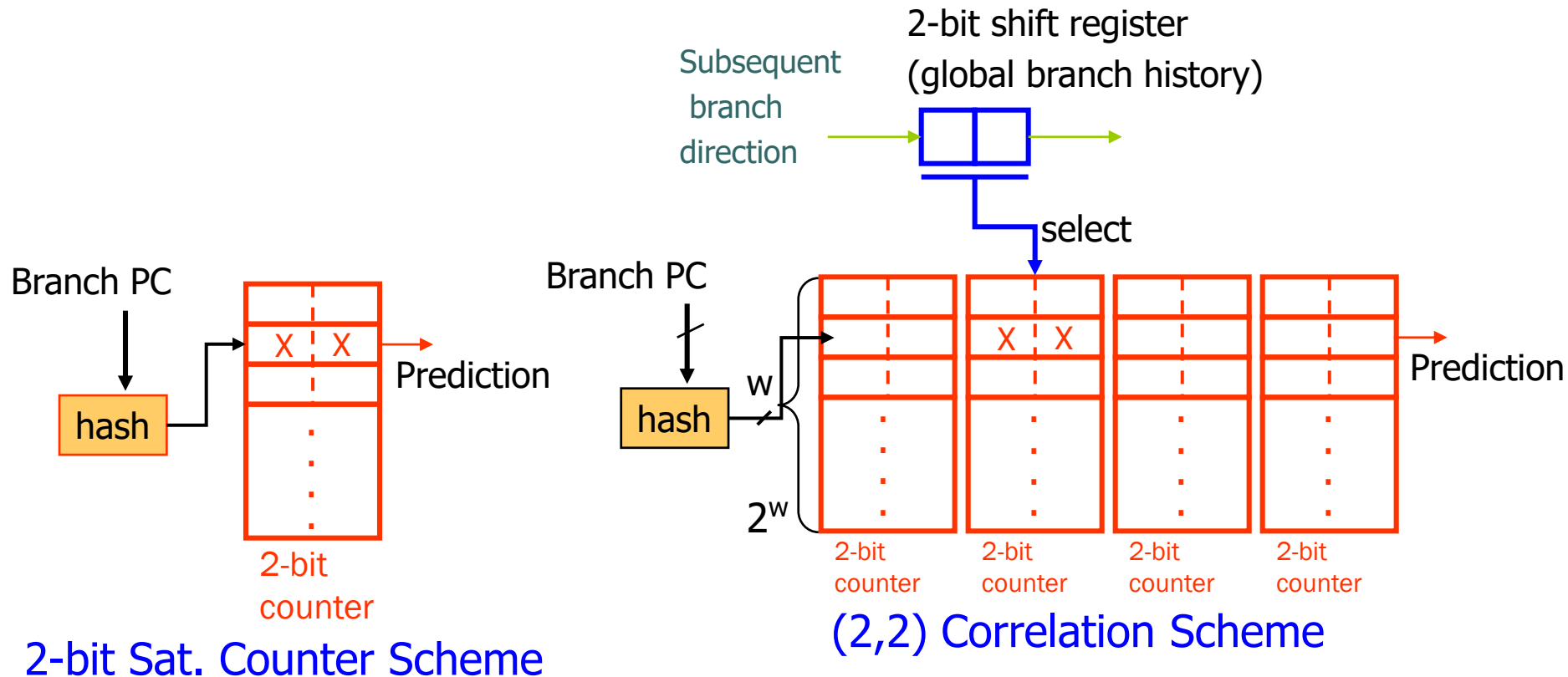
Code Snippet

```
if (aa==2)          // b1
    aa = 0;
if (bb==2)          // b2
    bb = 0;
if (aa!=bb) {       // b3
    .....
}
```



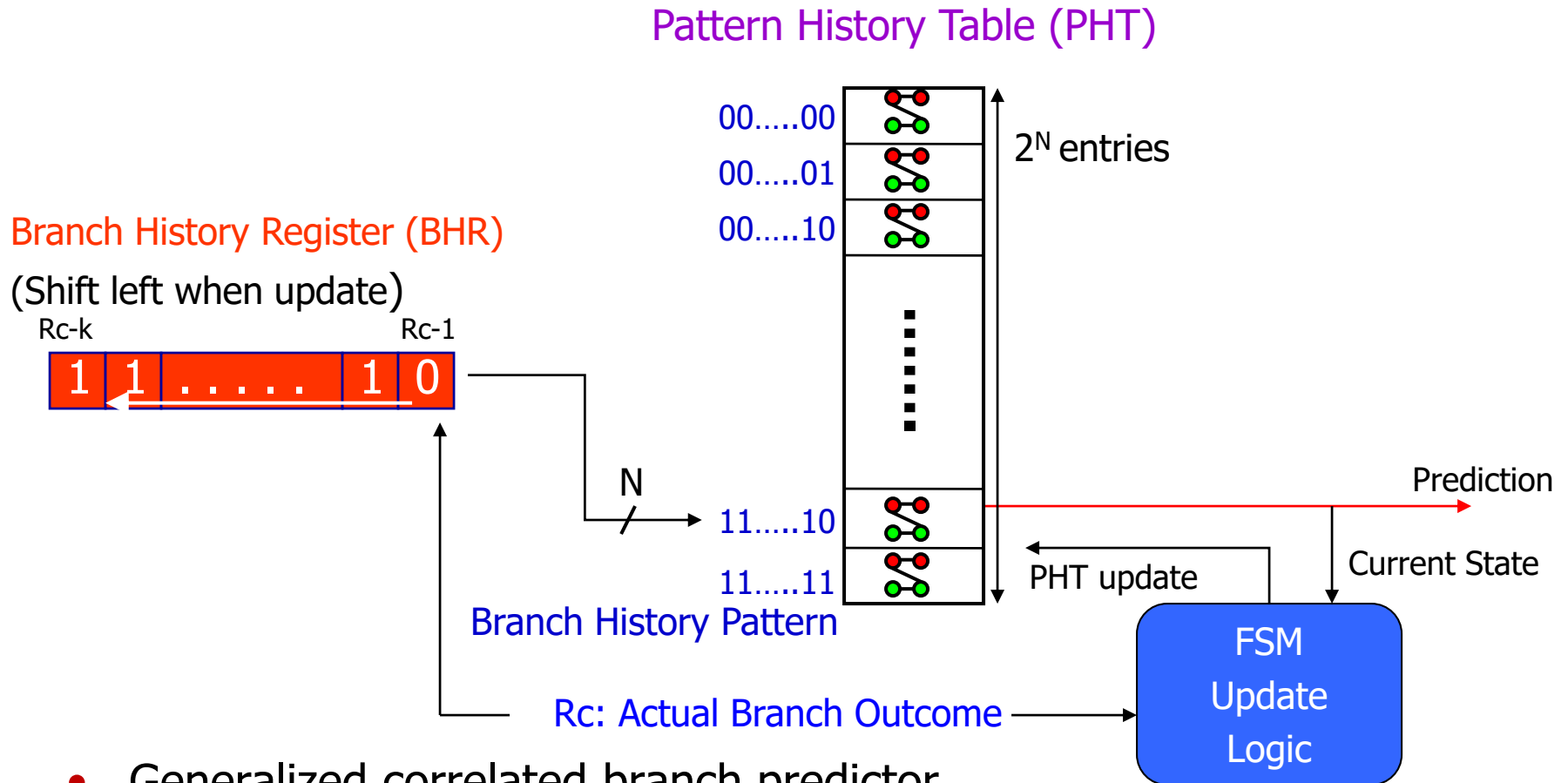
- Branch direction
 - Not independent
 - Correlated to the path taken
- Example: Path 1-1 of b3 can be surely known beforehand
- Track path using a 2-bit register

Correlated Branch Predictor [PanSoRahmeh'92]



- (M,N) correlation scheme
 - M: shift register size (# bits)
 - N: N-bit counter

Two-Level Branch Predictor [YehPatt91,92,93]



- Generalized correlated branch predictor
- 1st level keeps branch history in Branch History Register (BHR)
- 2nd level segregates pattern history in Pattern History Table (PHT)

Branch History Register

- An N-bit Shift Register = 2^N patterns in PHT
- Shift-in branch outcomes
 - 1 $\Rightarrow$ taken
 - 0 $\Rightarrow$ not taken
- First-in First-Out
- BHR can be
 - Global
 - Per-set
 - Local (Per-address)

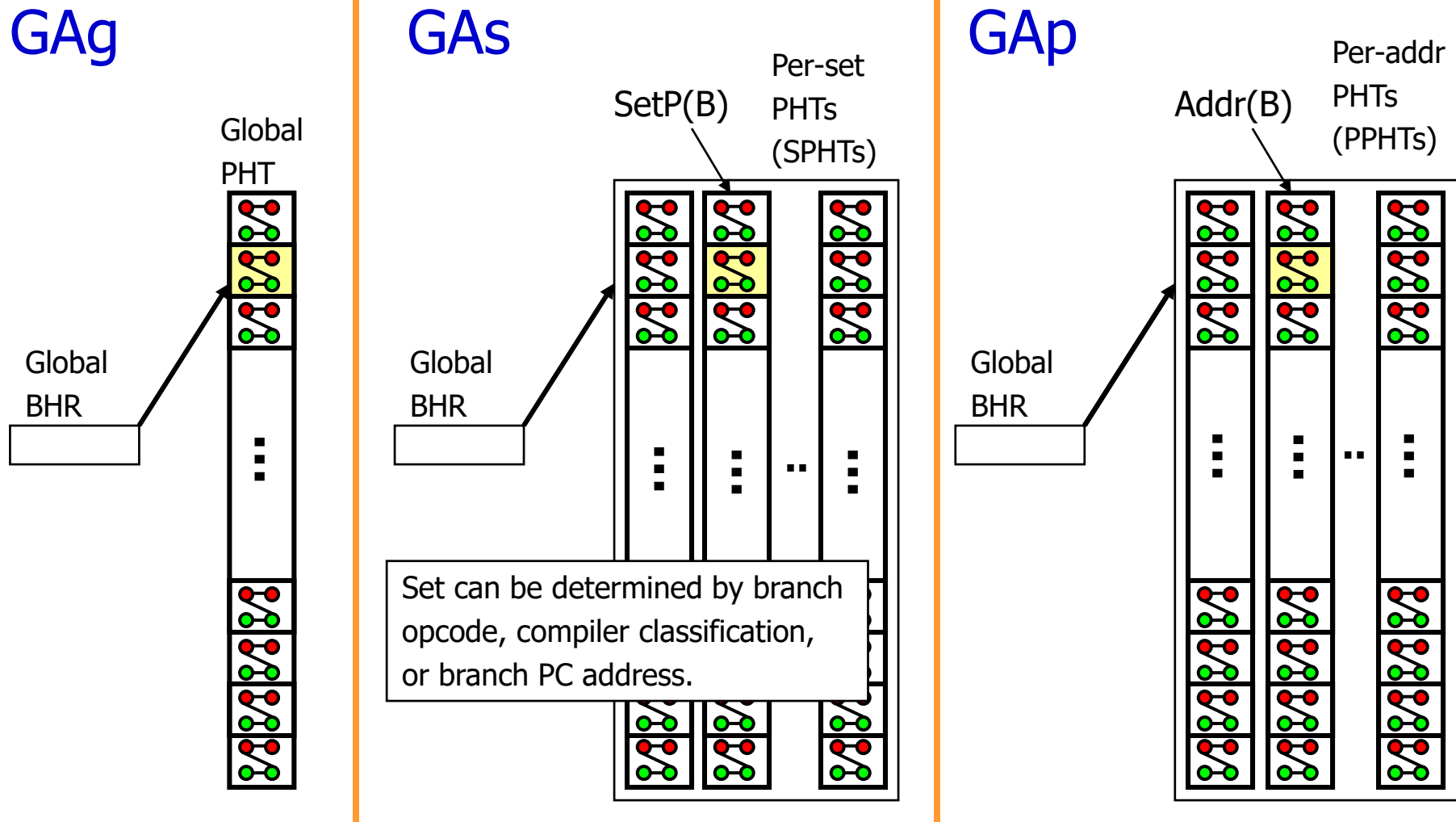
Pattern History Table

- 2^N entries addressed by N-bit BHR
- Each entry keeps a counter (2-bit or more) for prediction
 - Counter update: the same as 2-bit counter
 - Can be initialized in alternate patterns (01, 10, 01, 10, ..)
- Alias (or interference) problem

Variation & Reference		Description
GAg	[2]	Global Adaptive Branch Prediction using one global pattern history table.
GAs	[4]	Global Adaptive Branch Prediction using per-set pattern history tables.
GAp	[4]	Global Adaptive Branch Prediction using per-address pattern history tables.
PAG	[1]	Per-address Adaptive Branch Prediction using one global pattern history table.
PAs	-	Per-address Adaptive Branch Prediction using per-set pattern history tables.
PAp	[2]	Per-address Adaptive Branch Prediction using per-address pattern history tables.
SAG	-	Per-Set Adaptive Branch Prediction using one global pattern history table.
SAs	-	Per-Set Adaptive Branch Prediction using per-set pattern history tables.
SAP	-	Per-Set Adaptive Branch Prediction using per-address pattern history tables.

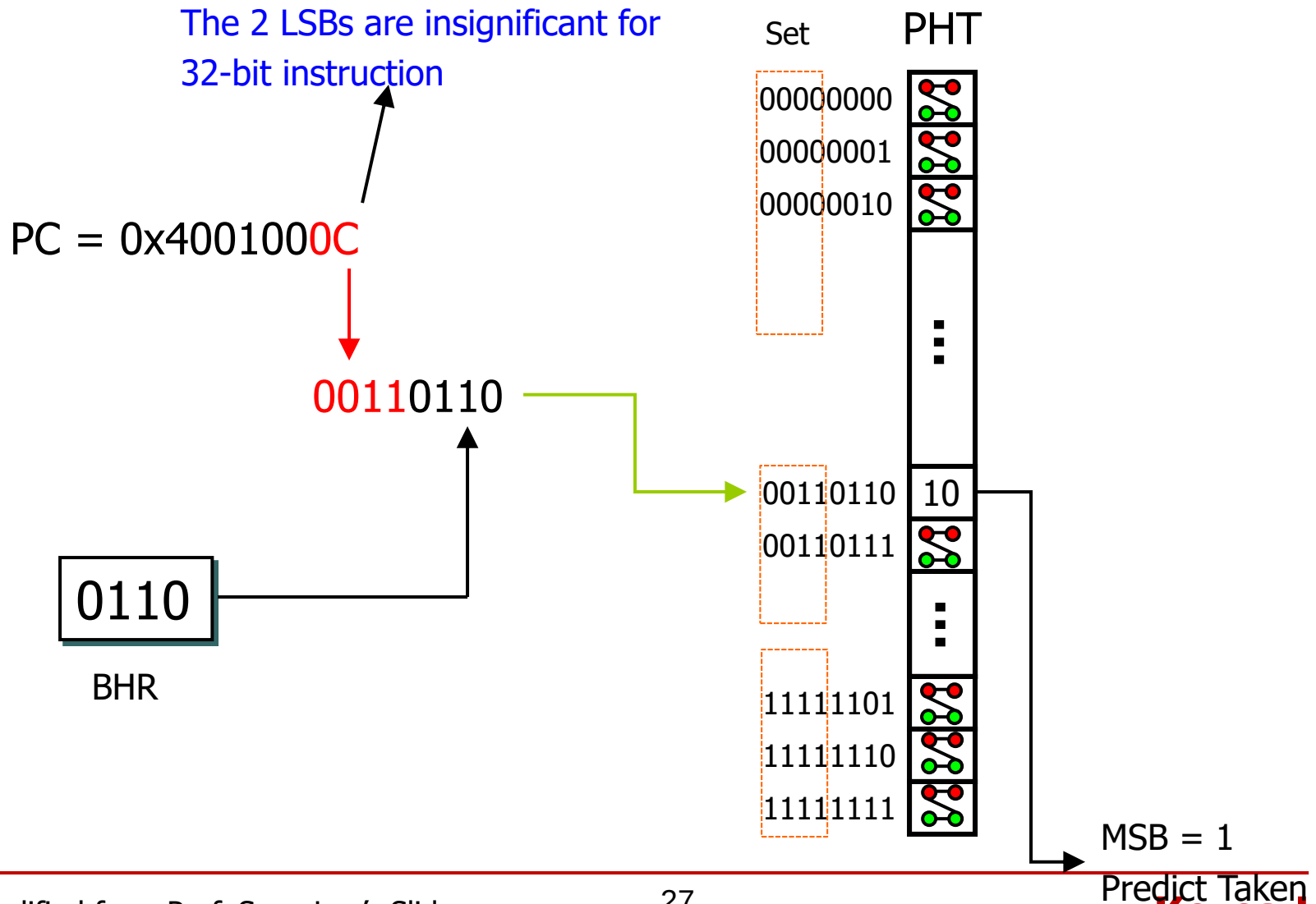
Source: [A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History](#) by Yeh and Patt, 1993

Global History Schemes

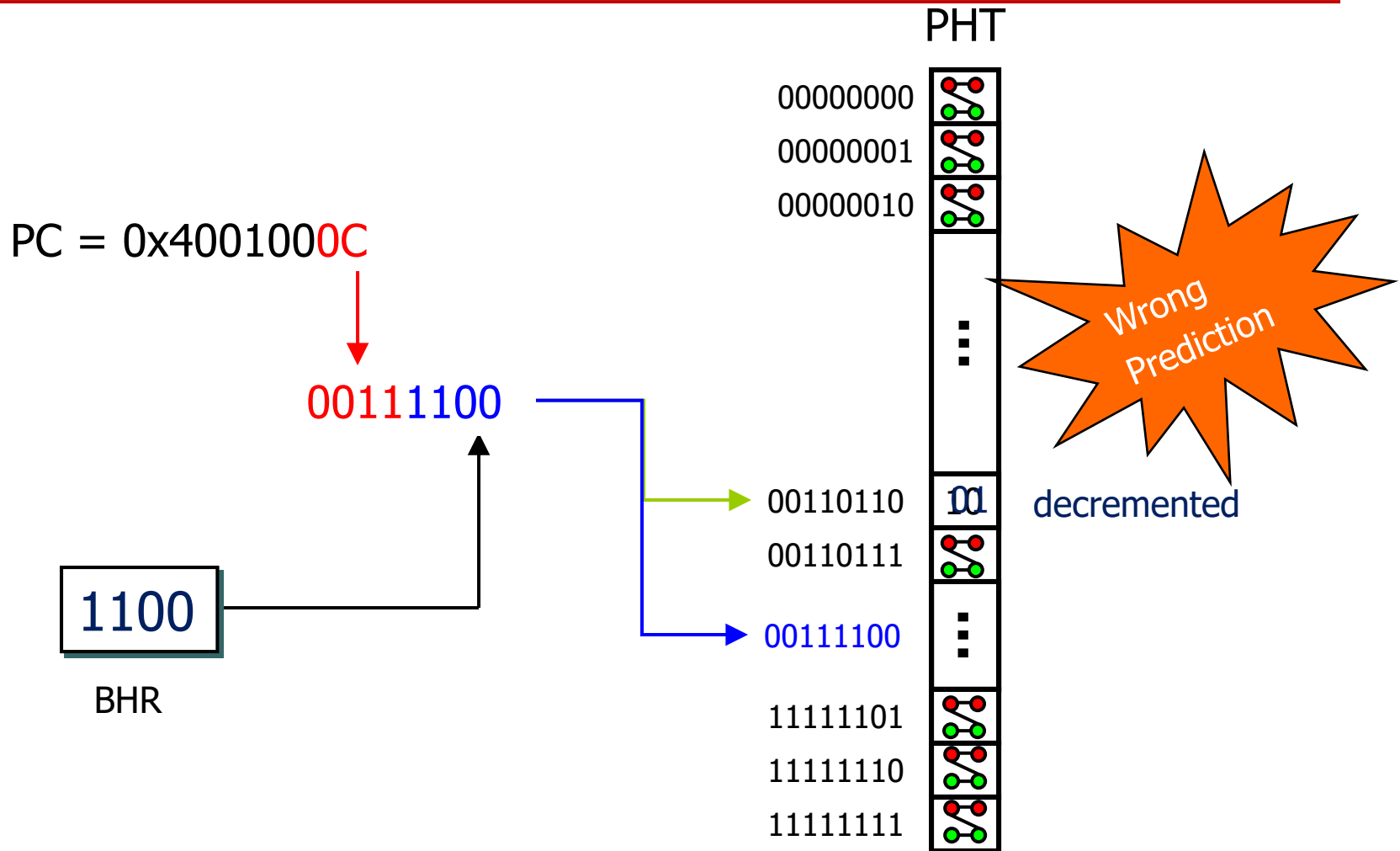


* [PanSoRahmeh'92] similar to GAp

GAs Two-Level Branch Prediction

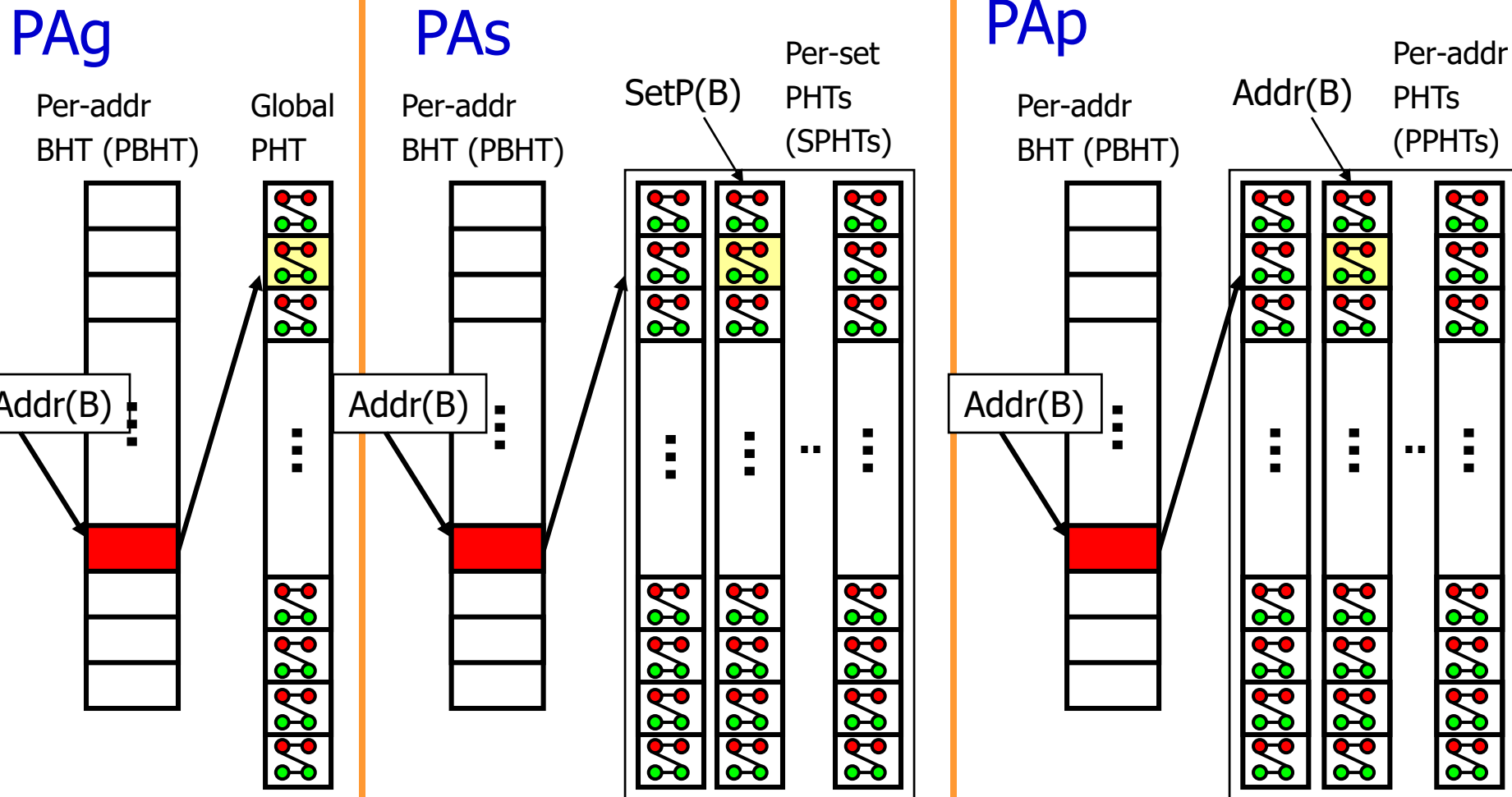


Predictor Update (Actually, Not Taken)



- Update Predictor after branch is resolved

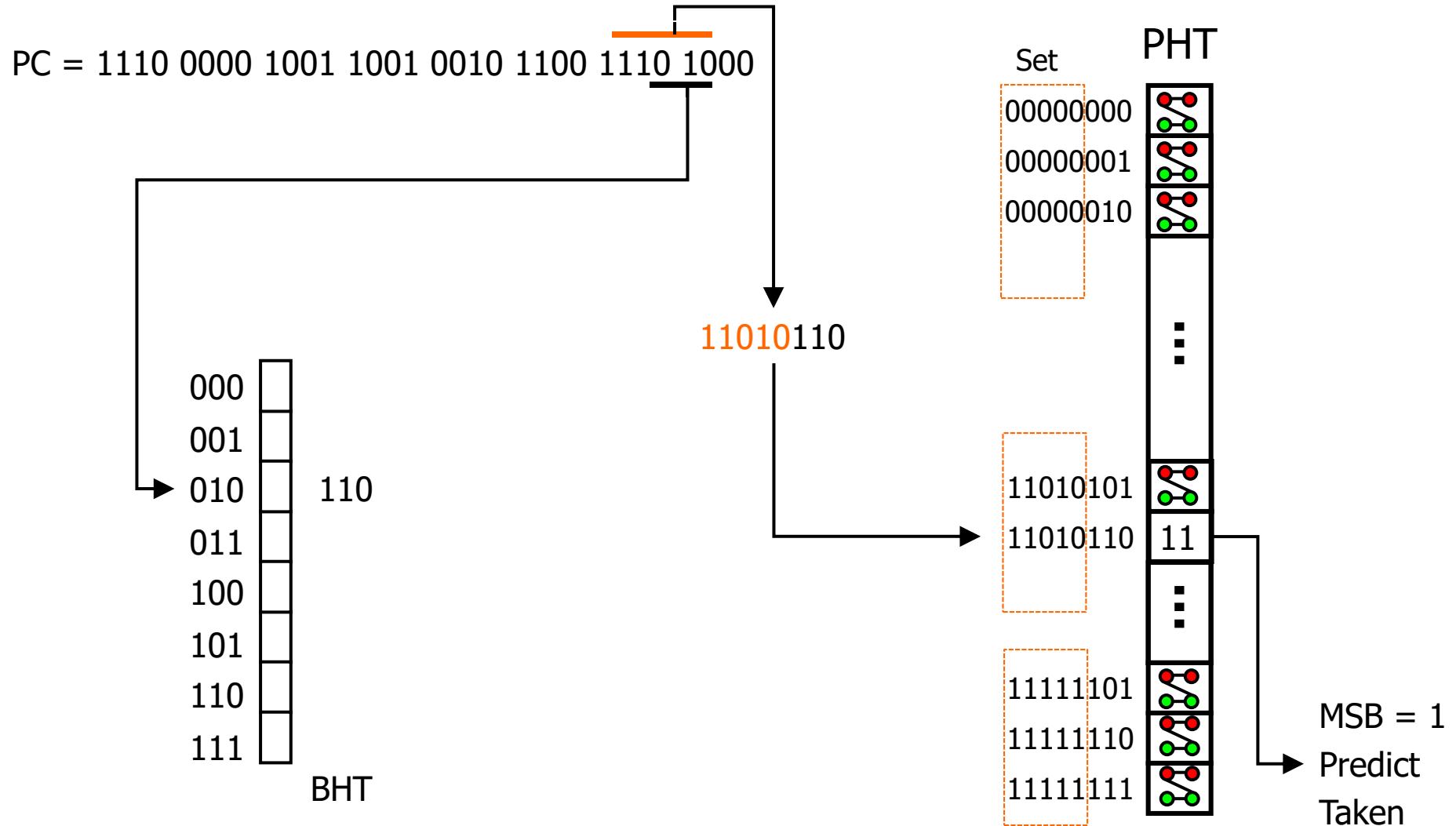
Per-Address History Schemes



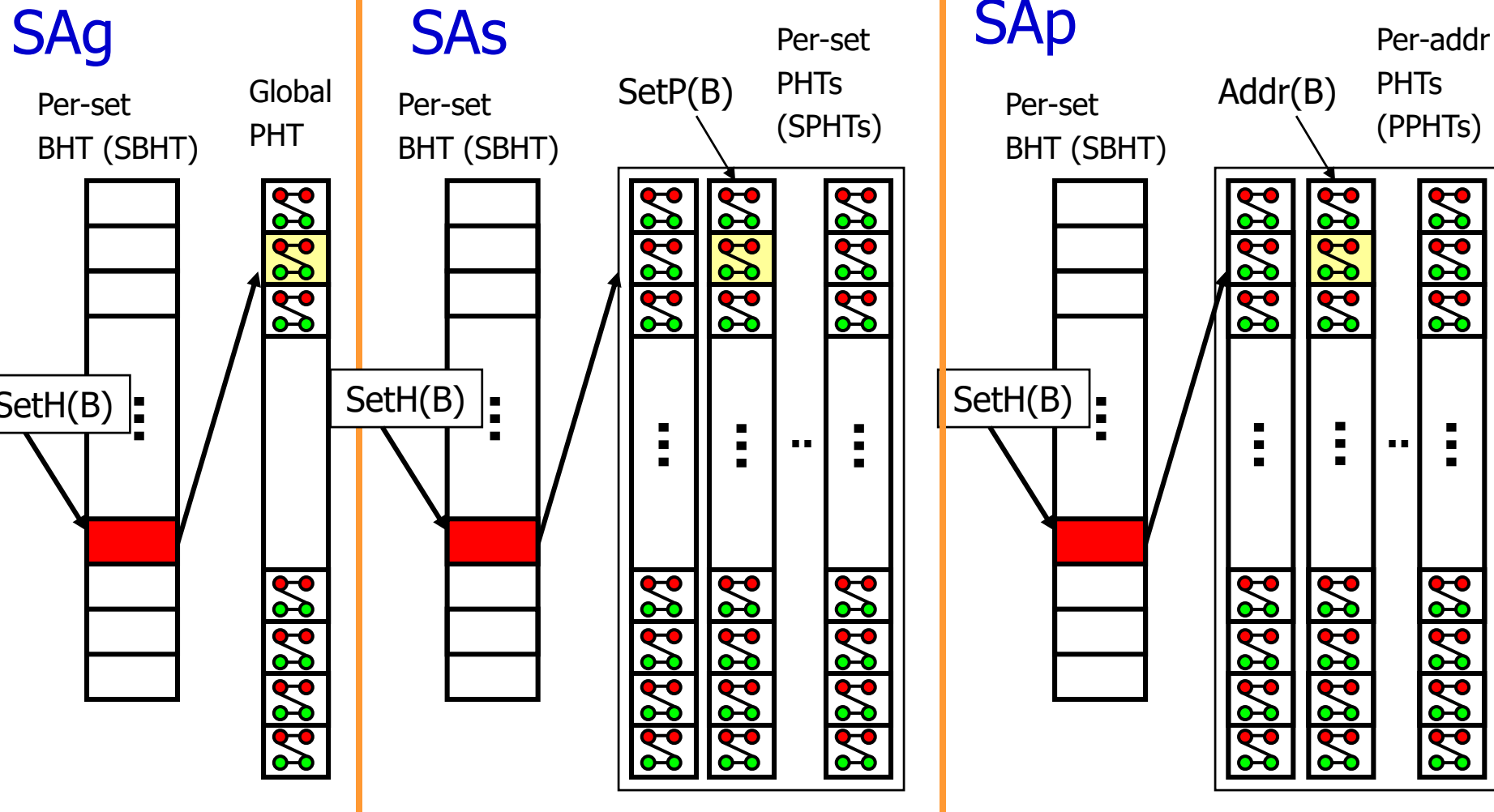
Ex: Alpha 21264's local predictor

Ex: P6, Itanium

PAs Two-Level Branch Predictor



Per-Set History Schemes



PHT Indexing

Branch addr	Global history	Gselect 4/4	
00000000	00000001	00000001	
00000000	00000000	00000000	
11111111	00000000	11110000	
11111111	10000000	11110000	

Insufficient
History

- Tradeoff between more history bits and address bits
- Too many bits needed in Gselect $\Rightarrow$ sparse table entries

Gshare Branch Predictor [McFarling93]

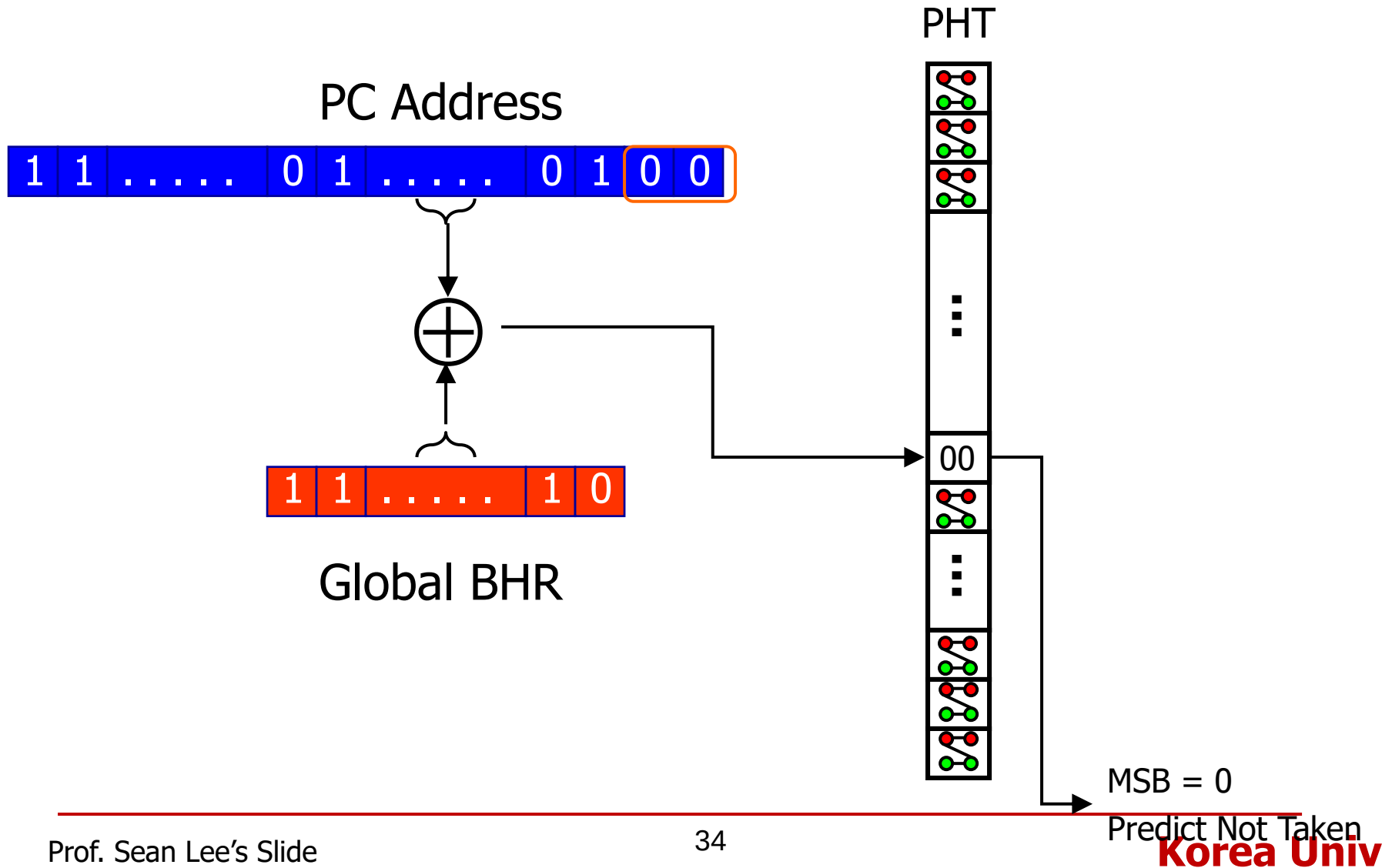
Branch addr	Global history	Gselect 4/4	Gshare 8/8
00000000	00000001	00000001	00000001
00000000	00000000	00000000	00000000
11111111	00000000	11110000	11111111
11111111	10000000	11110000	01111111

Gselect 4/4: Index PHT by **concatenate** low order 4 bits

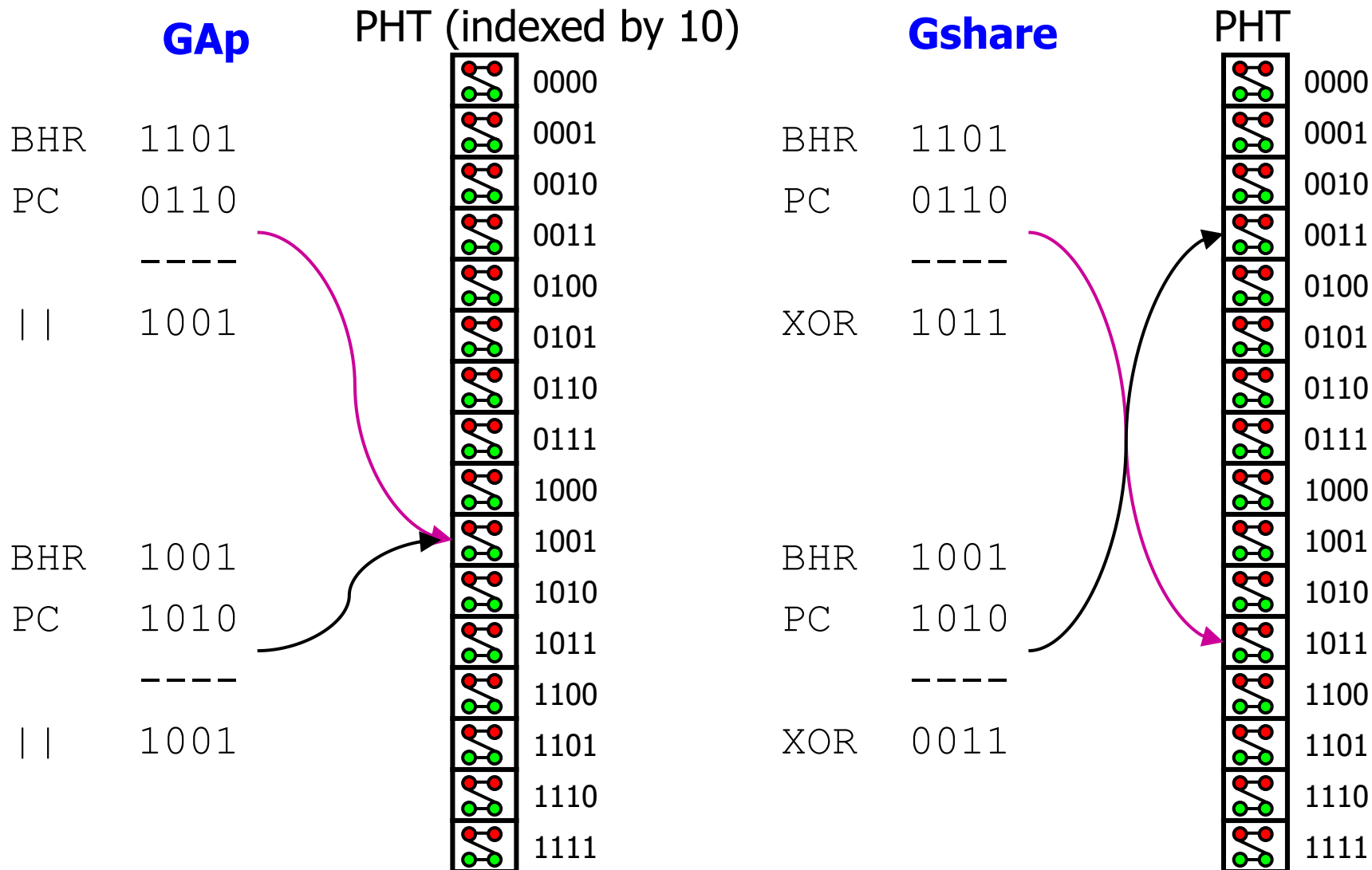
Gshare 8/8: Index PHT by {Branch address $\oplus$ Global history}

- Tradeoff between more history bits and address bits
- Too many bits needed in Gselect $\Rightarrow$ sparse table entries
- Gshare $\Rightarrow$ Not to lose global history bits
- Ex: AMD Athlon, MIPS R12000, Sun MAJC, Broadcom SiByte's SB-1

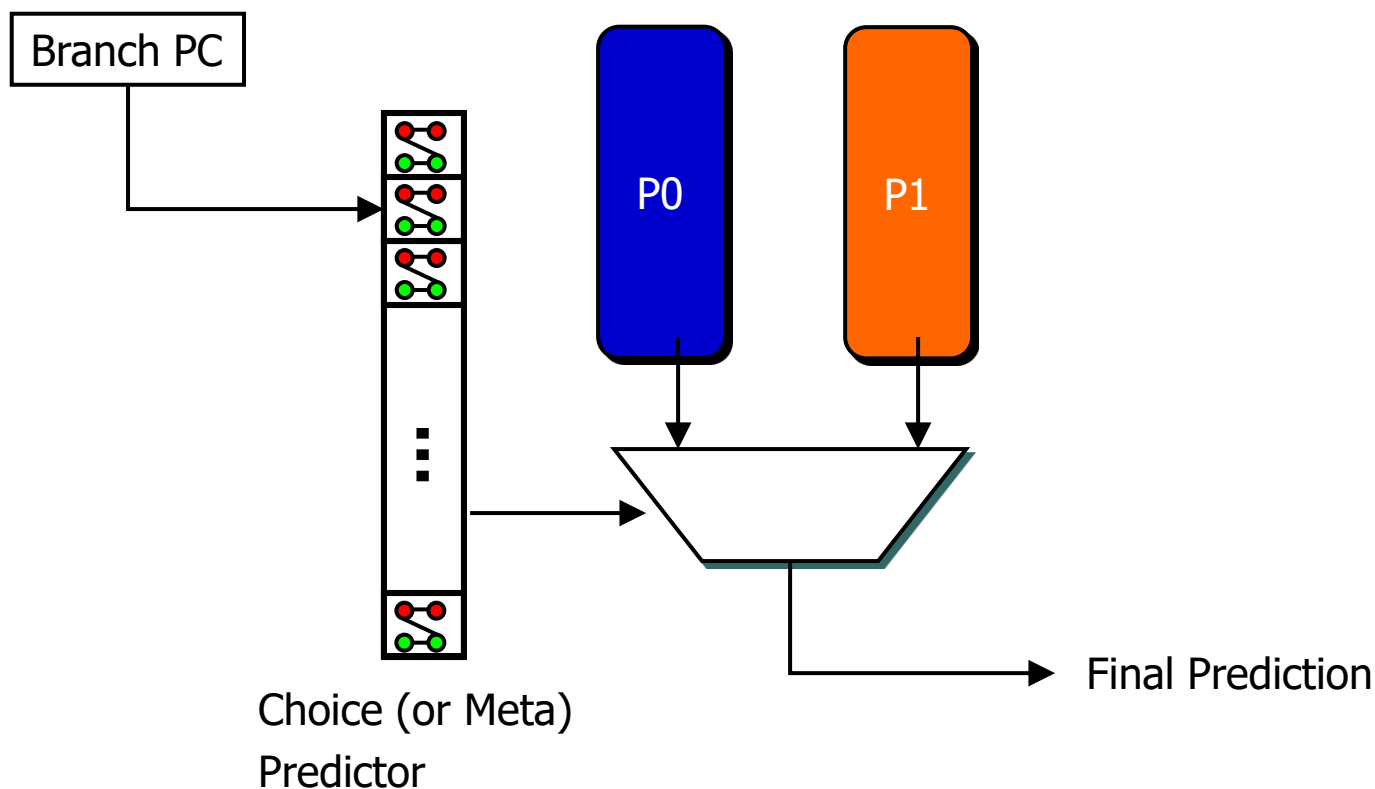
Gshare Branch Predictor



Aliasing Example



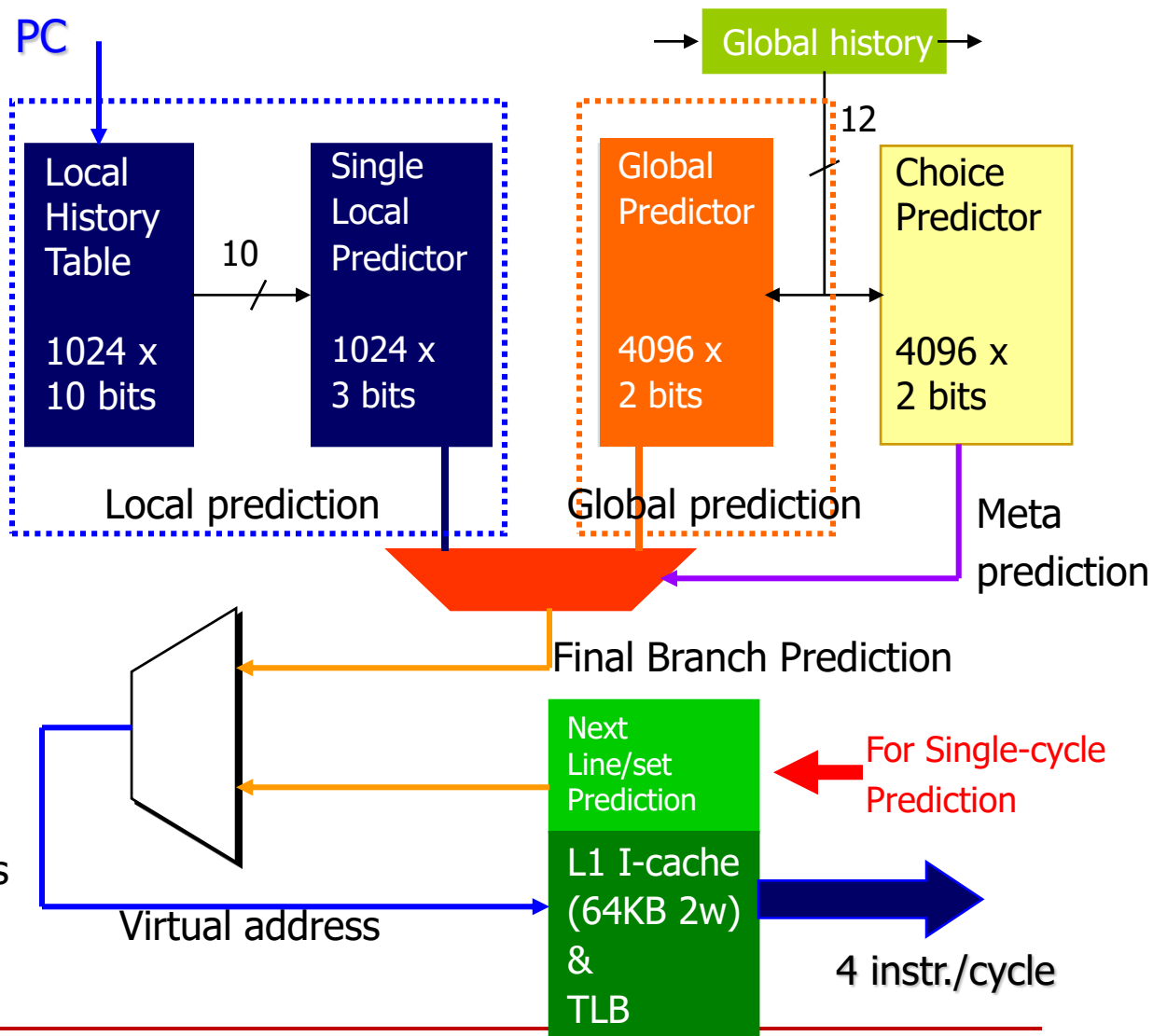
Hybrid Branch Predictor [McFarling93]



- Some branches correlated to global history, some correlated to local history
- Only update the meta-predictor when 2 predictors disagree

Alpha 21264 (EV6) Hybrid Predictor

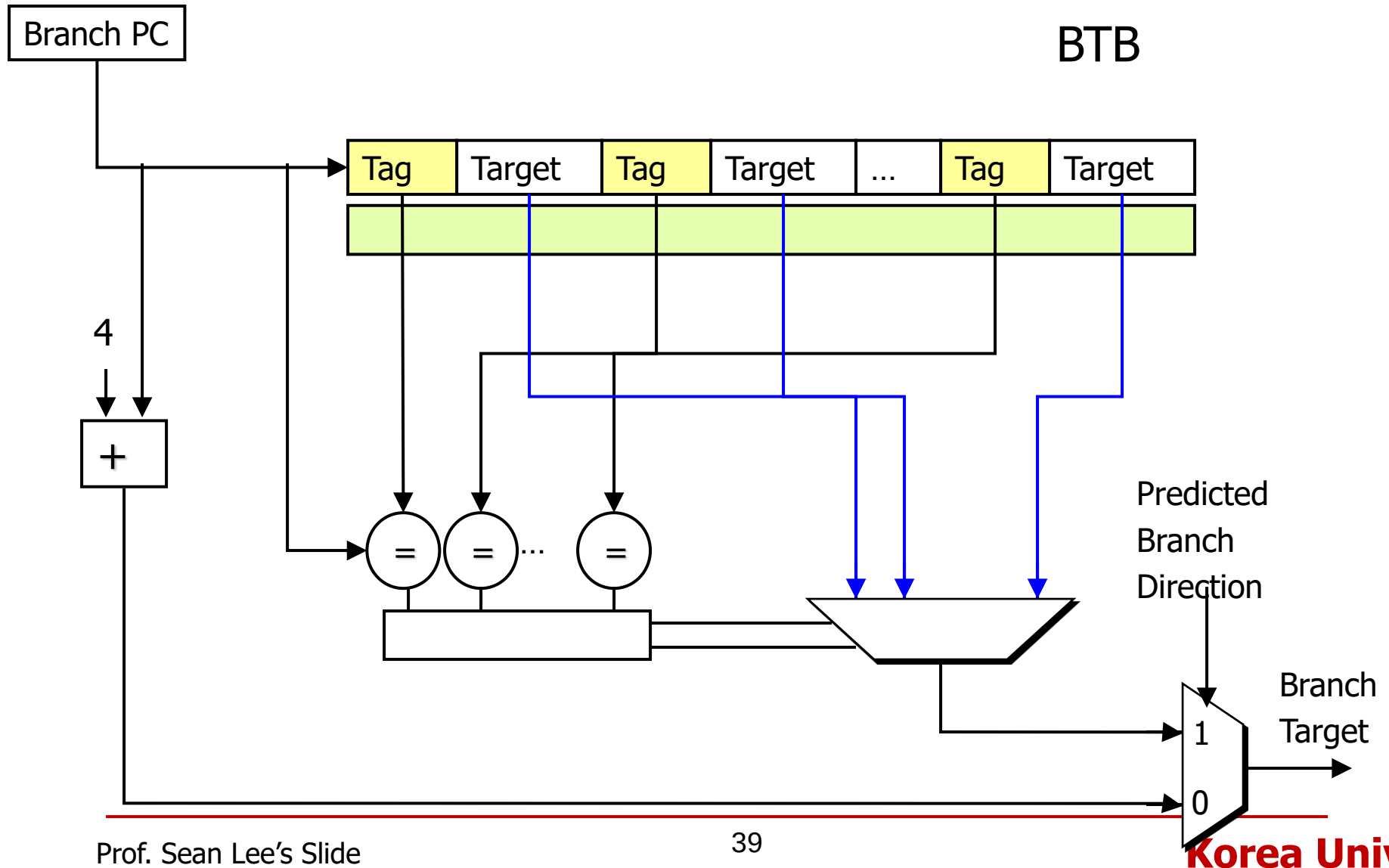
- A “**tournament branch predictor**”
- Multi-predictor scheme w/
 - **Local predictor** (~PAg)
 - Self-correlation
 - **Global predictor**
 - Inter-correlation
 - **Choice predictor** as the **decision maker**: a 2-bit sat. counter to credit either local or global predictors.
- Die size impact
 - History info tables ~2%
 - BTB ~ 2.7% (associated with I-\$ on a per-line basis)
- 2 cycle latency, we will discuss more later



Branch Target Prediction

- Try the easy ones first
 - Direct jumps
 - Call/Return
 - Conditional branch (bi-directional)
- Branch Target Buffer (BTB)
- Return Address Stack (RAS)

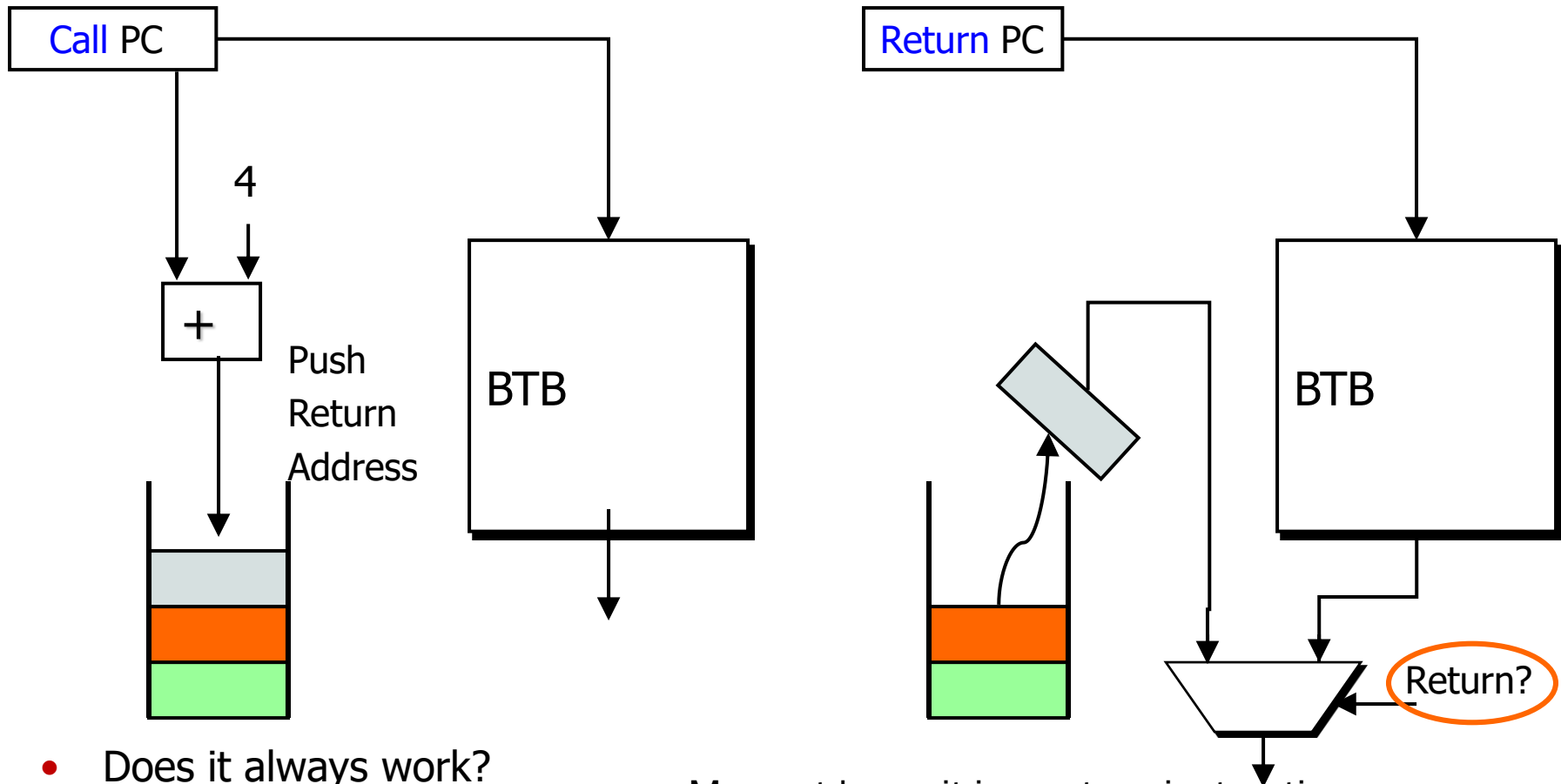
Branch Target Buffer (BTB)



Return Address Stack (RAS)

- Different call sites make return address hard to predict
 - Printf() being called by many callers
 - The target of “return” instruction in printf() is a moving target
- A hardware stack (LIFO)
 - Call will push return address on the stack
 - Return uses the prediction off of TOS

Return Address Stack



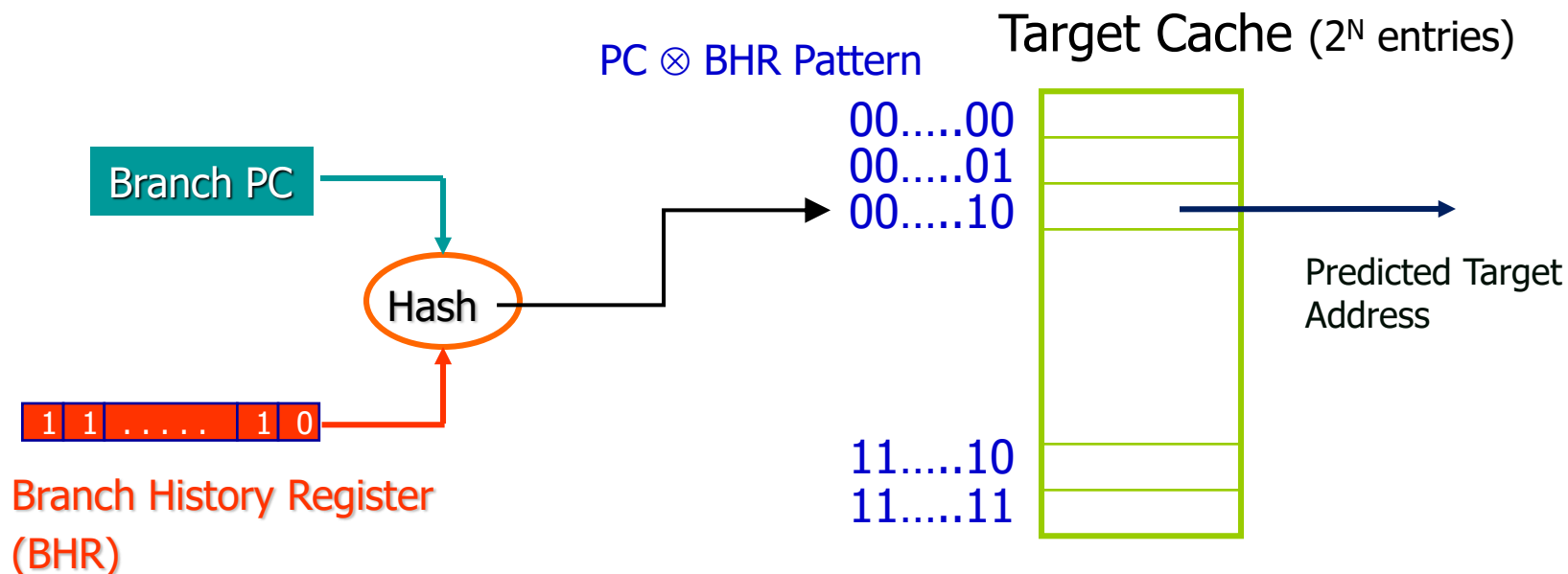
- Does it always work?
 - Call depth
 - Setjmp/Longjmp
 - Speculative call?

- May not know it is a return instruction prior to decoding
 - Rely on BTB for speculation
 - Fix once recognize Return

Indirect Jump

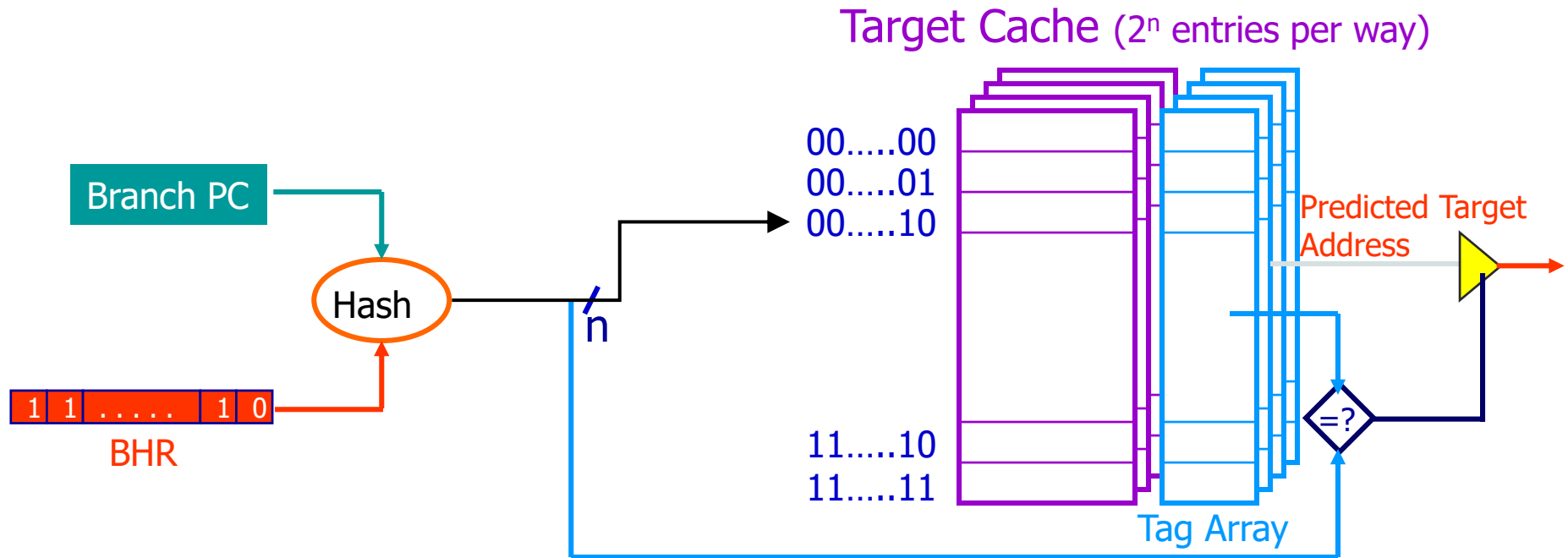
- Need Target Prediction
 - Many (potentially 2^{30} for 32-bit machine)
 - In reality, not so many
 - Similar to predicting values
- Tagless Target Prediction
- Tagged Target Prediction

Tagless Target Prediction [ChangHaoPatt'97]



- Modify the PHT to be a “Target Cache”
 - (indirect jump) ? (from target cache) : (from BTB)
- Alias?

Tagged Target Prediction [ChangHaoPatt'97]



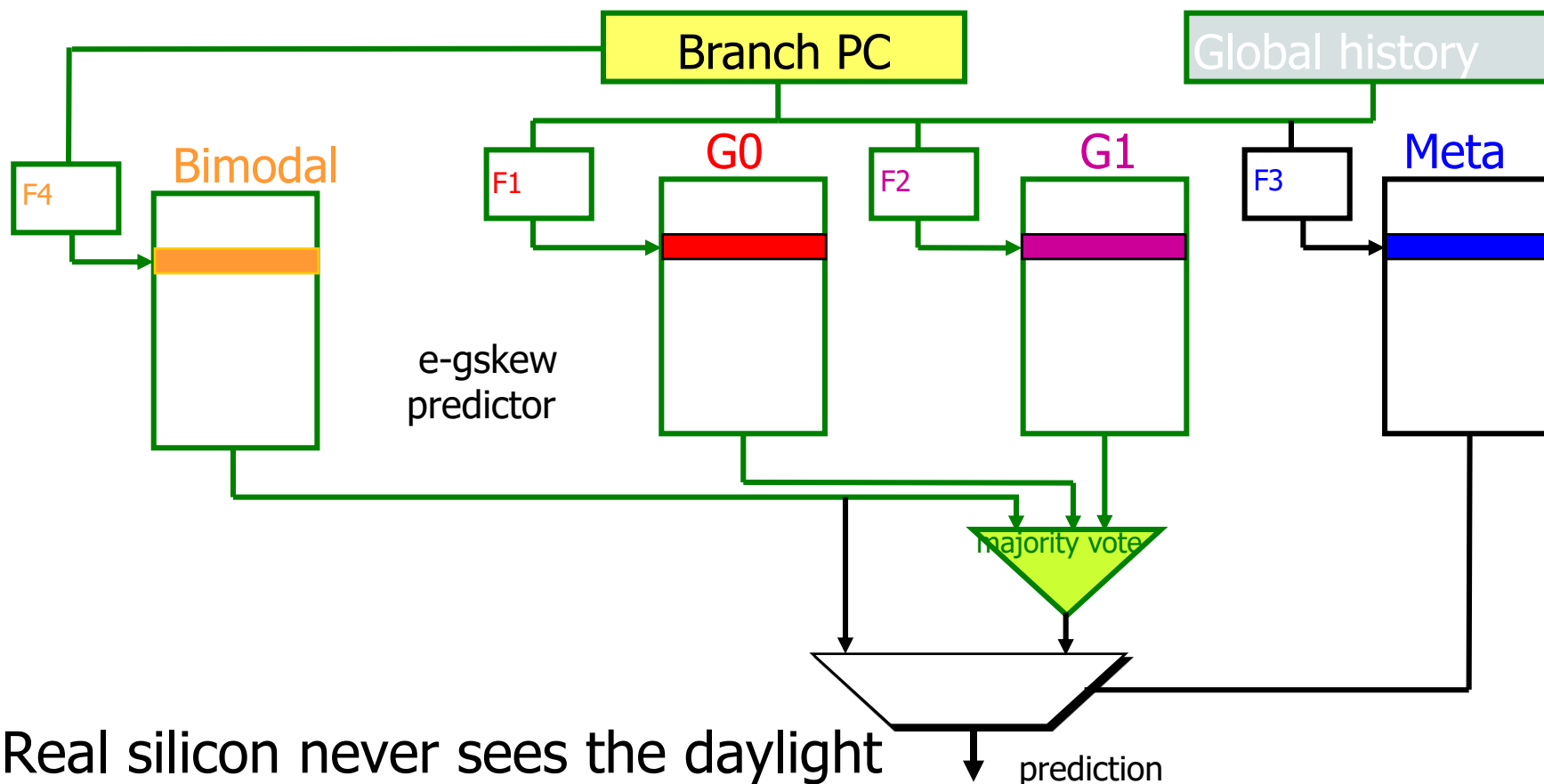
- To reduce aliasing with set-associative target cache
- Use branch PC and/or history for tags

Multiple Branch Prediction

- For a really wide machine
 - Across several basic blocks
 - Need to predict multiple branches per cycle
- How to fetch non-contiguous instructions in one cycle?
- Prediction accuracy extremely critical (will be reduced geometrically)

Backup Slides

Alpha EV8 Branch Predictor



- Real silicon never sees the daylight
- Use a 2Bc-gskew predictor (one form of enhanced gskew)
 - Bimodal predictor used as (1) static biased predictor and (2) part of e-gskew predictor
 - Global predictors G0 and G1 are part of e-gskew predictor
 - Table sizes: 352Kbits in total (208Kbits for prediction table; 144Kbits for hysteresis table.)