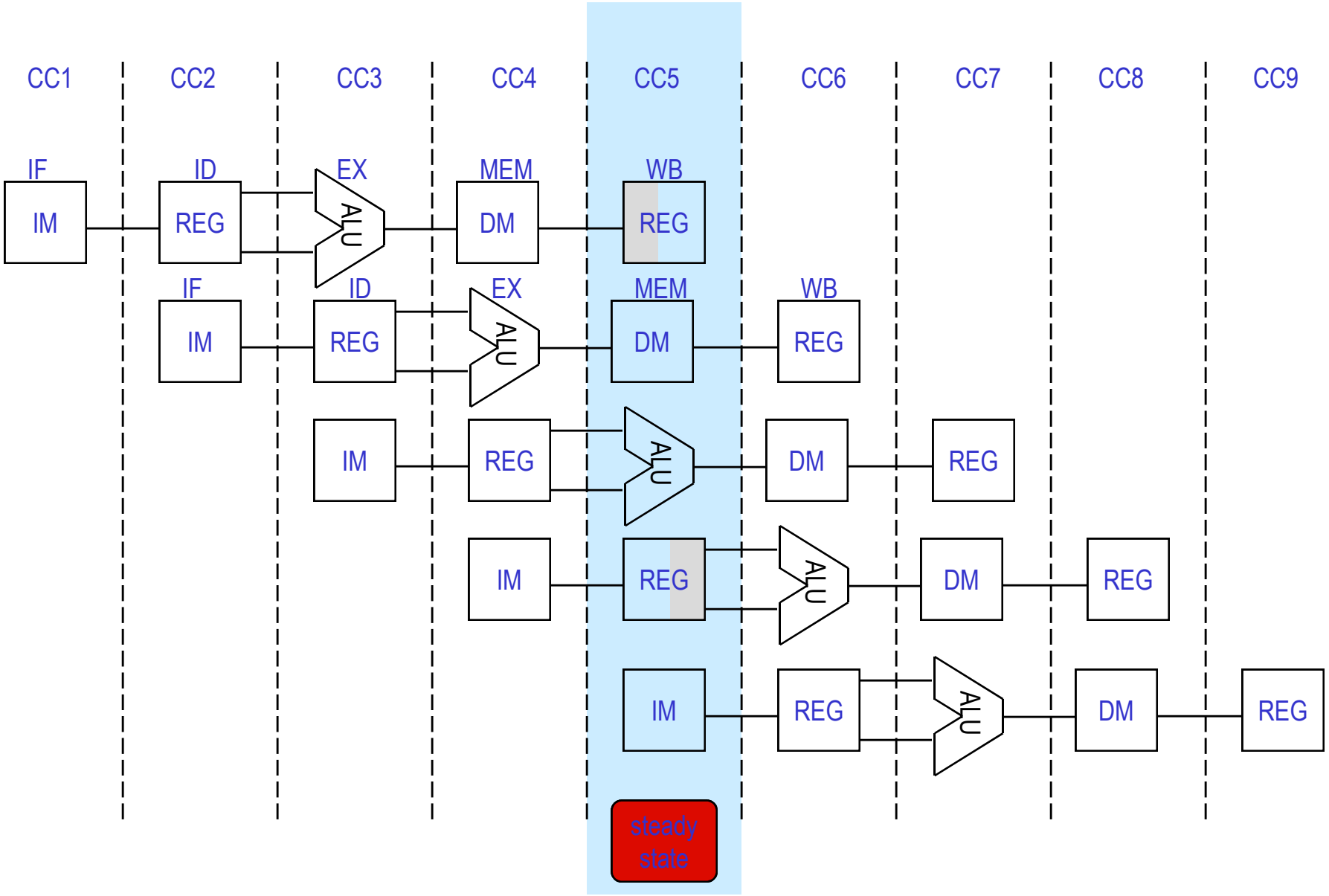
Lecture 9:
Pipeline – Control Hazard

Review: Pipelined Execution Timing

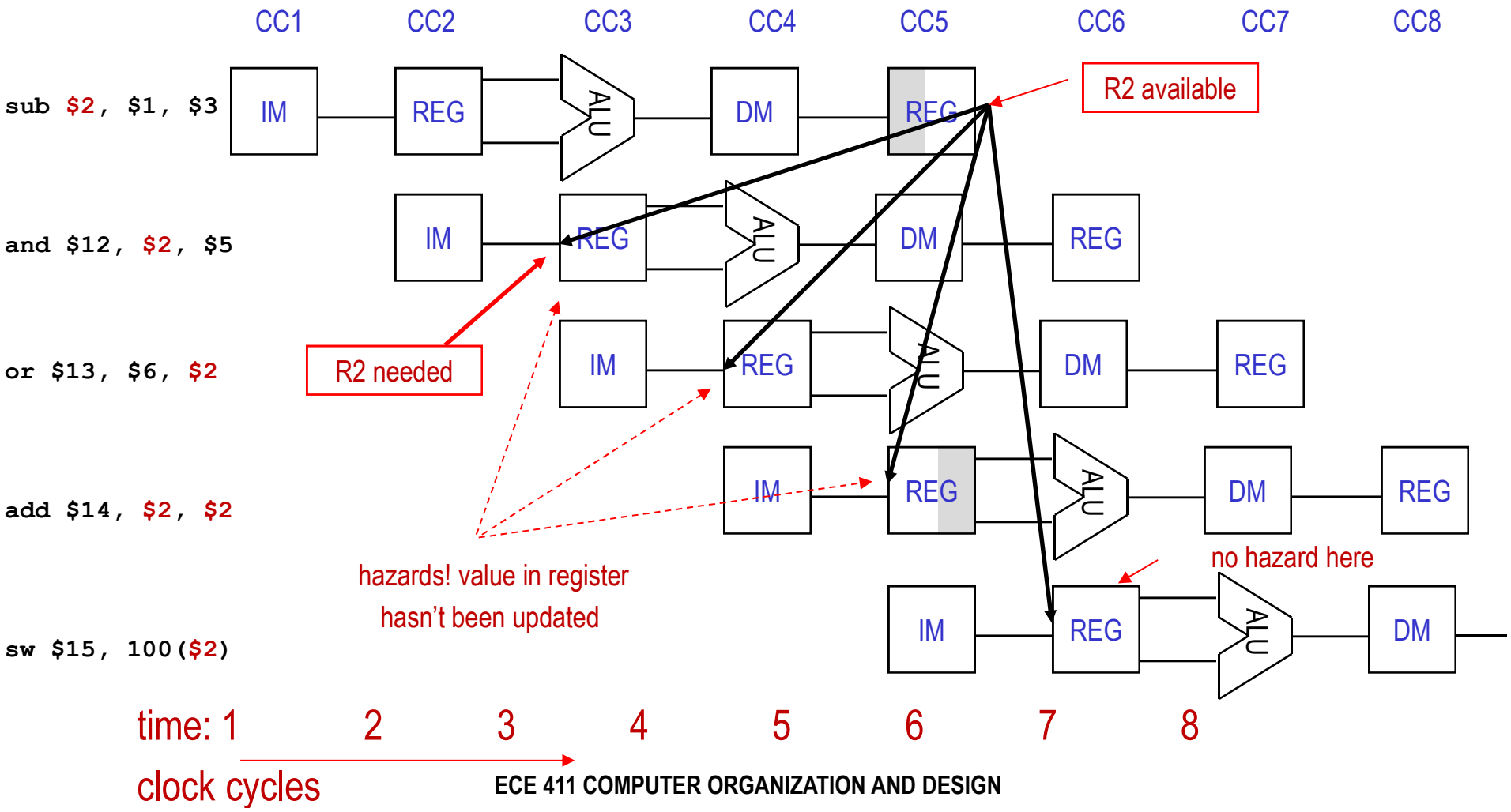


Review: Hazards

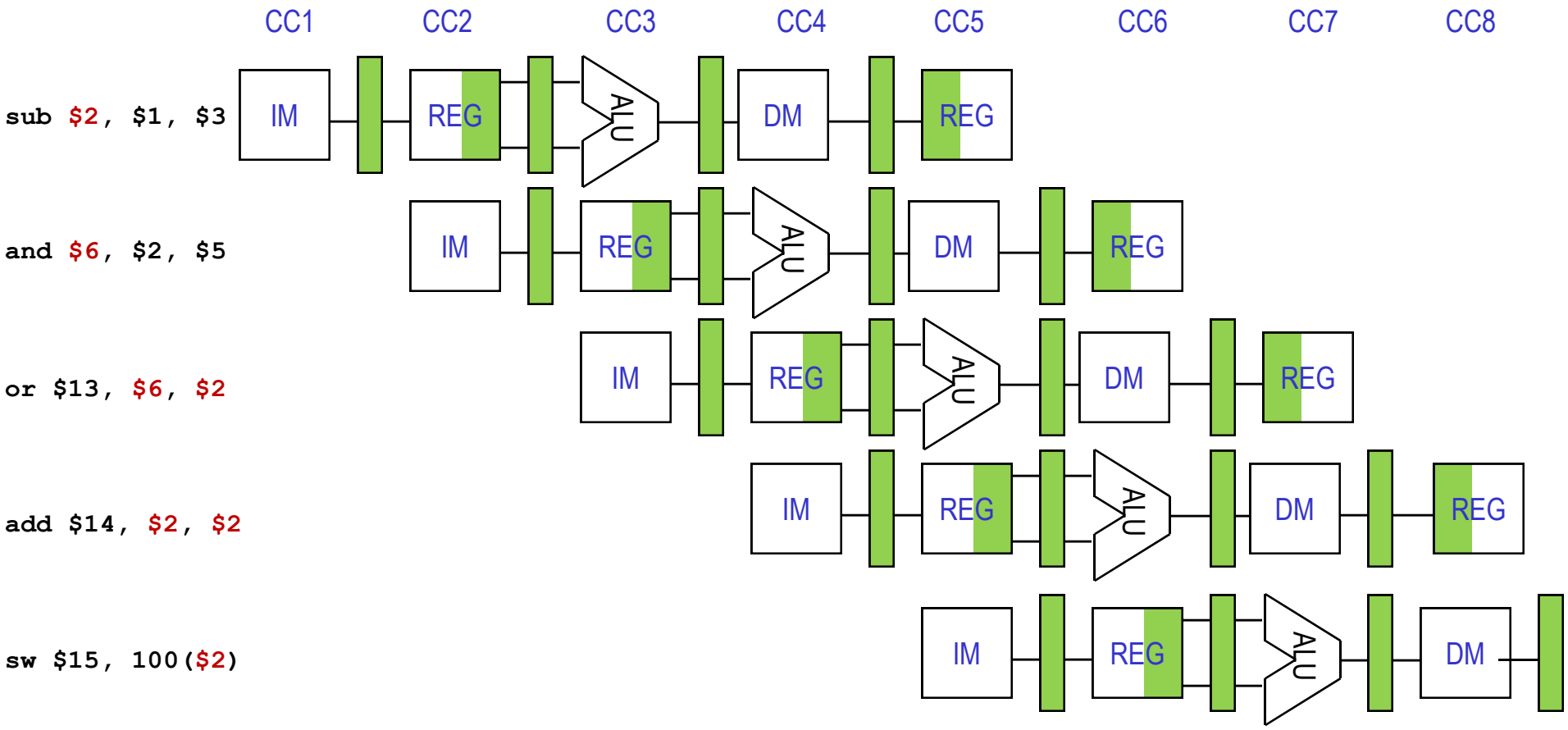
- situations that prevent starting the next instruction in the next cycle
  - ✓ structure hazards
    - a required resource is busy
  - ✓ data hazard
    - need to wait for previous instruction to complete its data read/write
  - ✓ control hazard
    - deciding on control action depends on previous instruction

Review: Data Hazards

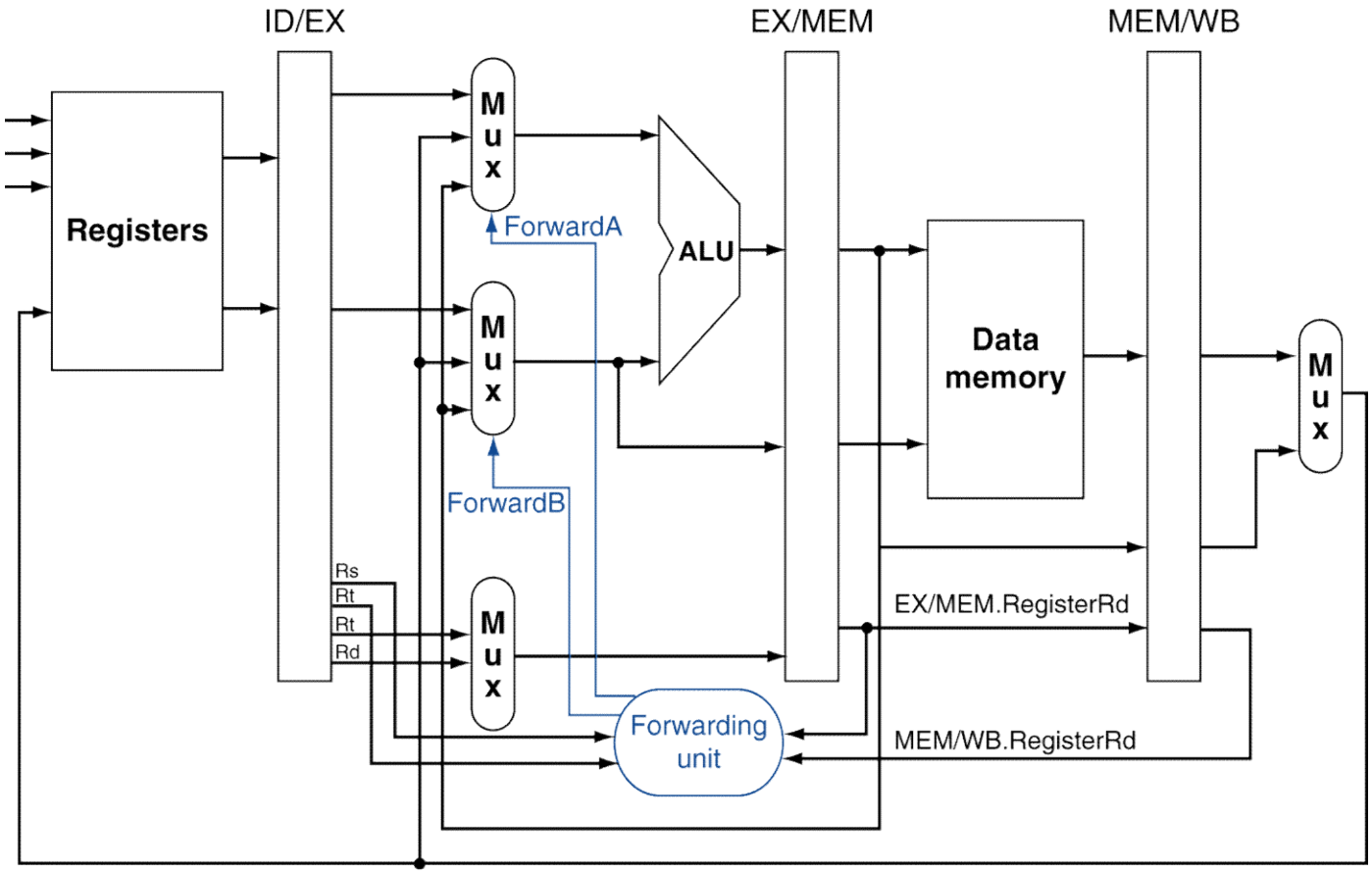
- when a result is needed in the pipeline before it is available, a data hazard occurs



Review: Reducing Data Hazards: Forwarding



Review: Forwarding Paths



b. With forwarding

Types of Branches

	Conditional	Unconditional
Direct	if - then- else for loops (bez, bnez, etc)	procedure calls (jal) goto (j)
Indirect		return (jr) virtual function lookup function pointers (jalr)

Control Hazard due to Branches

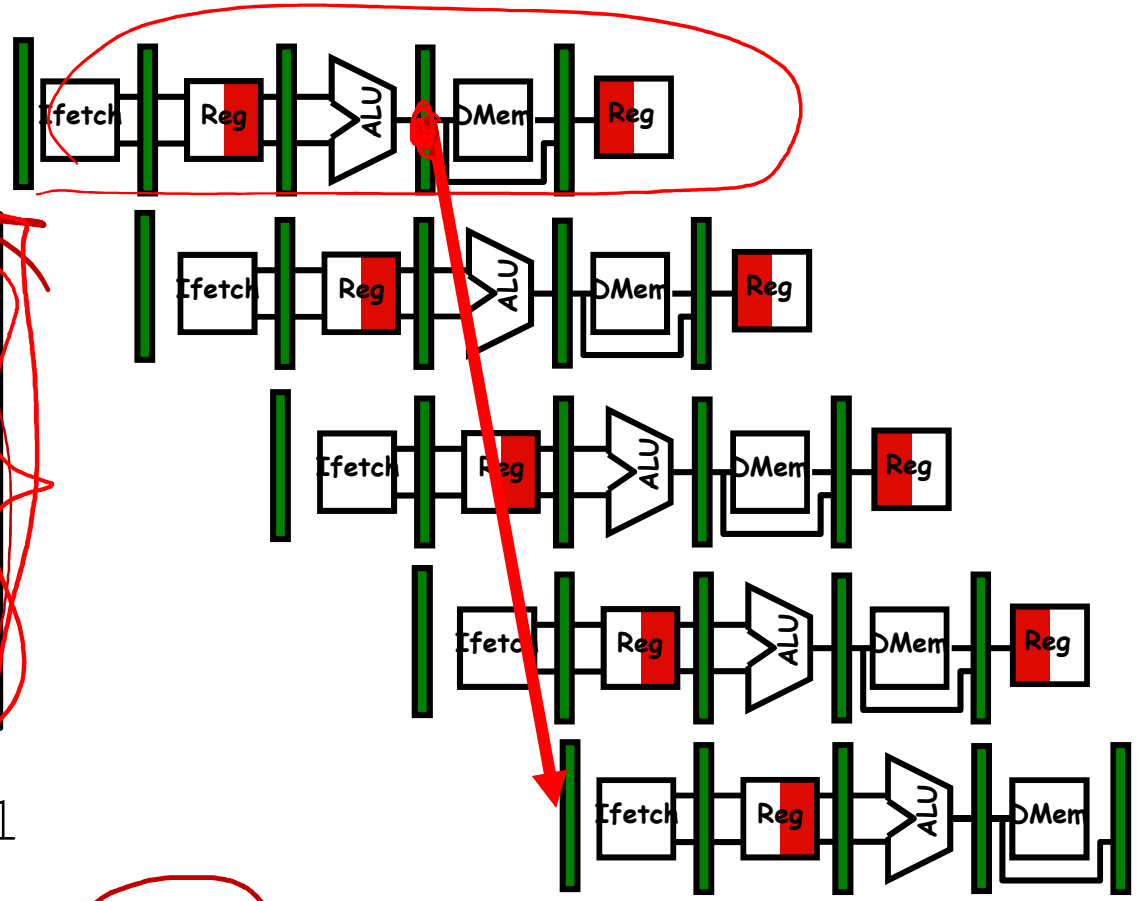
```
10: beq r1,r3,36

14: and r2,r3,r5

18: or  r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11
```



what do you do with the 3 instructions in between?

how do you do it?

where is the "commit"?

CPI

CPI = 1

20% branches

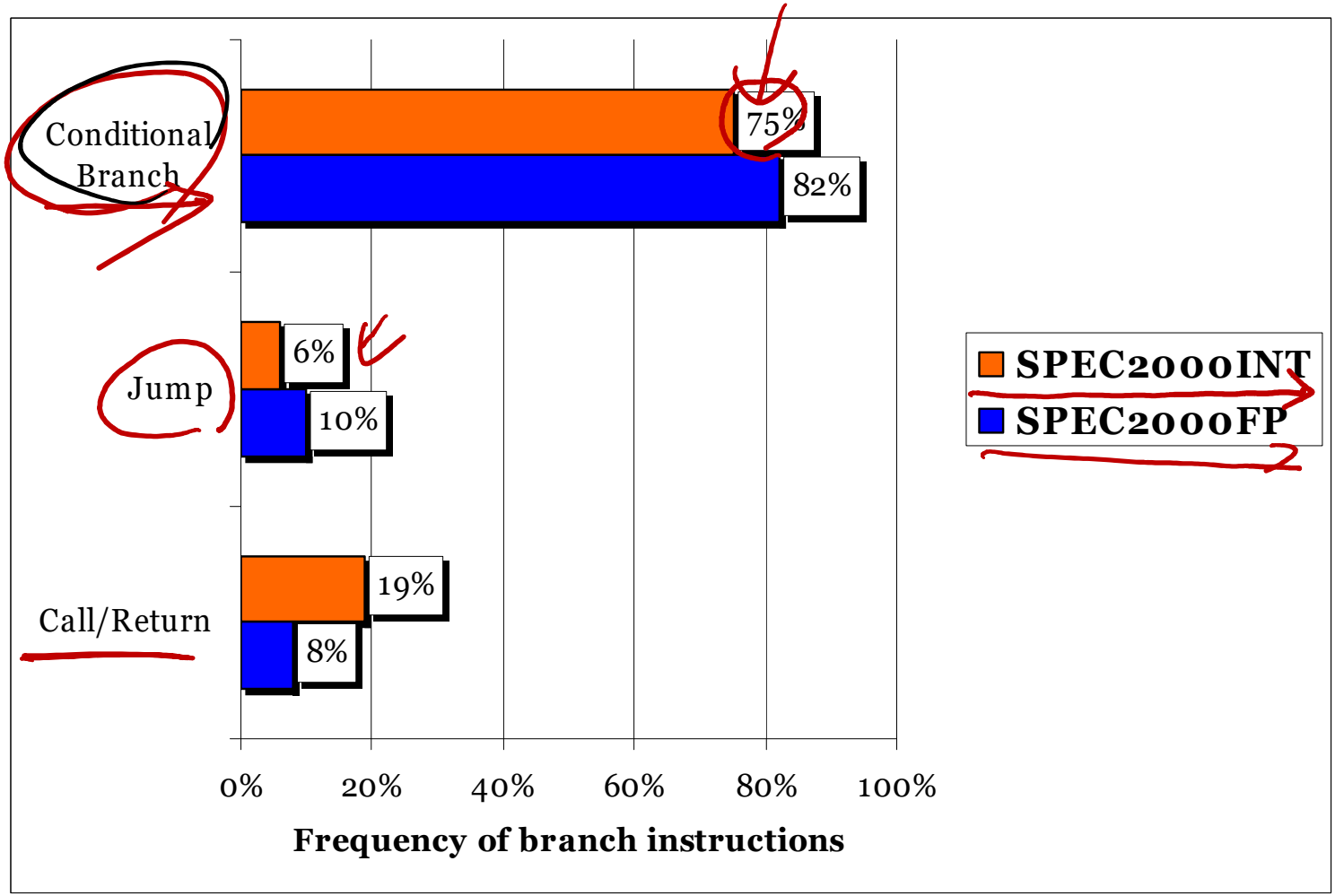
3 cycles

ADD	%
LD	%
ST	%

0.8 x 1 + 0.2 x 3

0.8 + 0.6 = 1.4

Categorizing Branches



Source: H&P using Alpha

Branch Hazard Resolutions

#1: stall until branch direction is clear (☹)

#2: static branch prediction

- ✓ predict branch Not Taken (fall through, as shown in previous slide)
  - execute successor instructions in sequence
  - "squash" instructions in pipeline if branch actually taken
  - PC+4 already calculated, so use it to get next instruction
- ✓ predict branch Taken
  - but haven't calculated branch target address
  - might incur 1 cycle branch penalty
  - other machines: branch target known before outcome

#3: dynamic branch prediction

- ✓ will talk about it later today

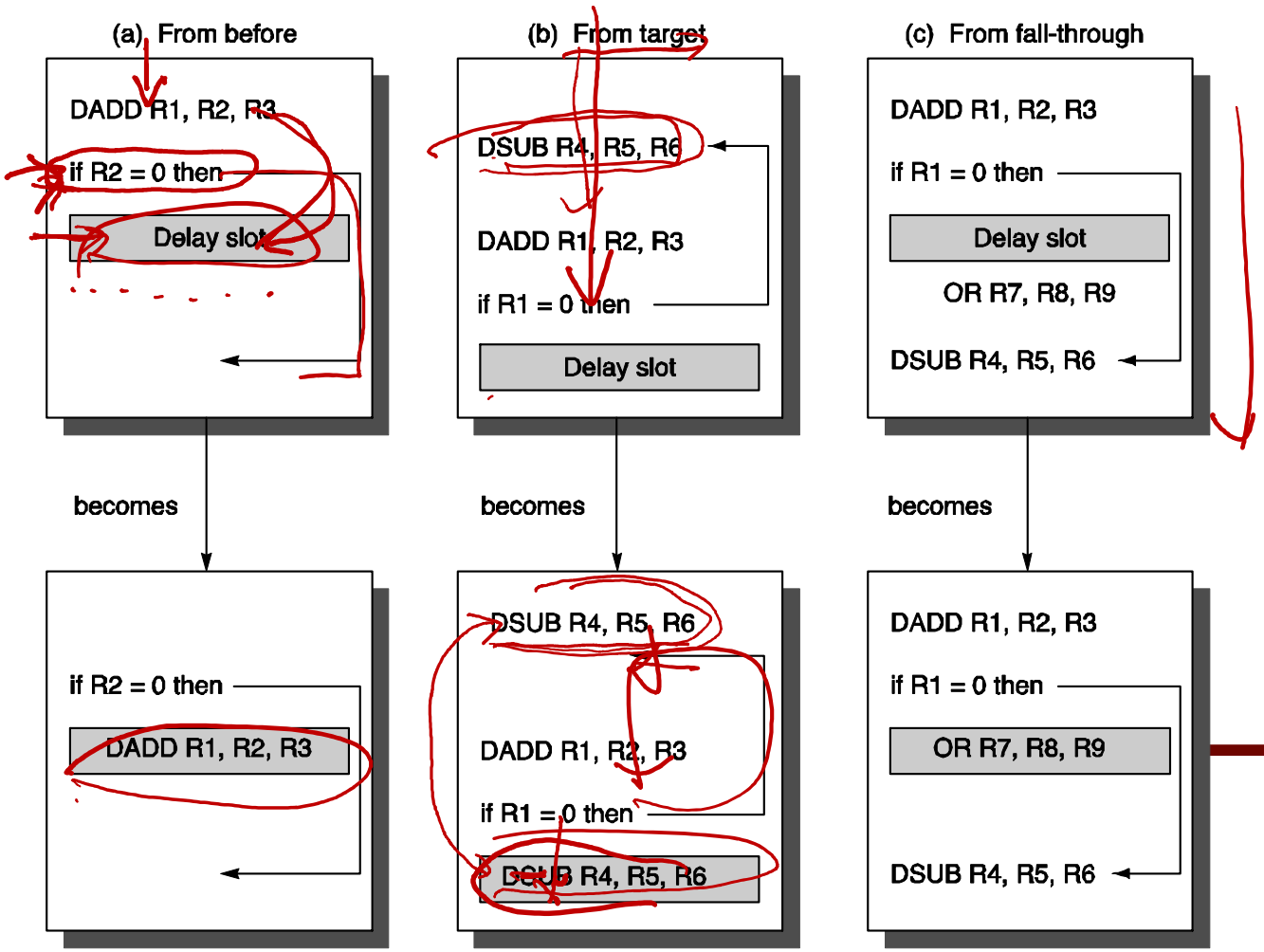
Alternative Branch Hazard Resolutions

- #4 delayed branch
  - ✓ define branch to take place after a following instruction

```
branch instruction
sequential successor1
sequential successor2
........
sequential successorn
branch target if taken
```

  - ✓ 1 slot delay allows proper decision and branch target address in 5 stage pipeline (next page)

Filling Branch Delay Slot



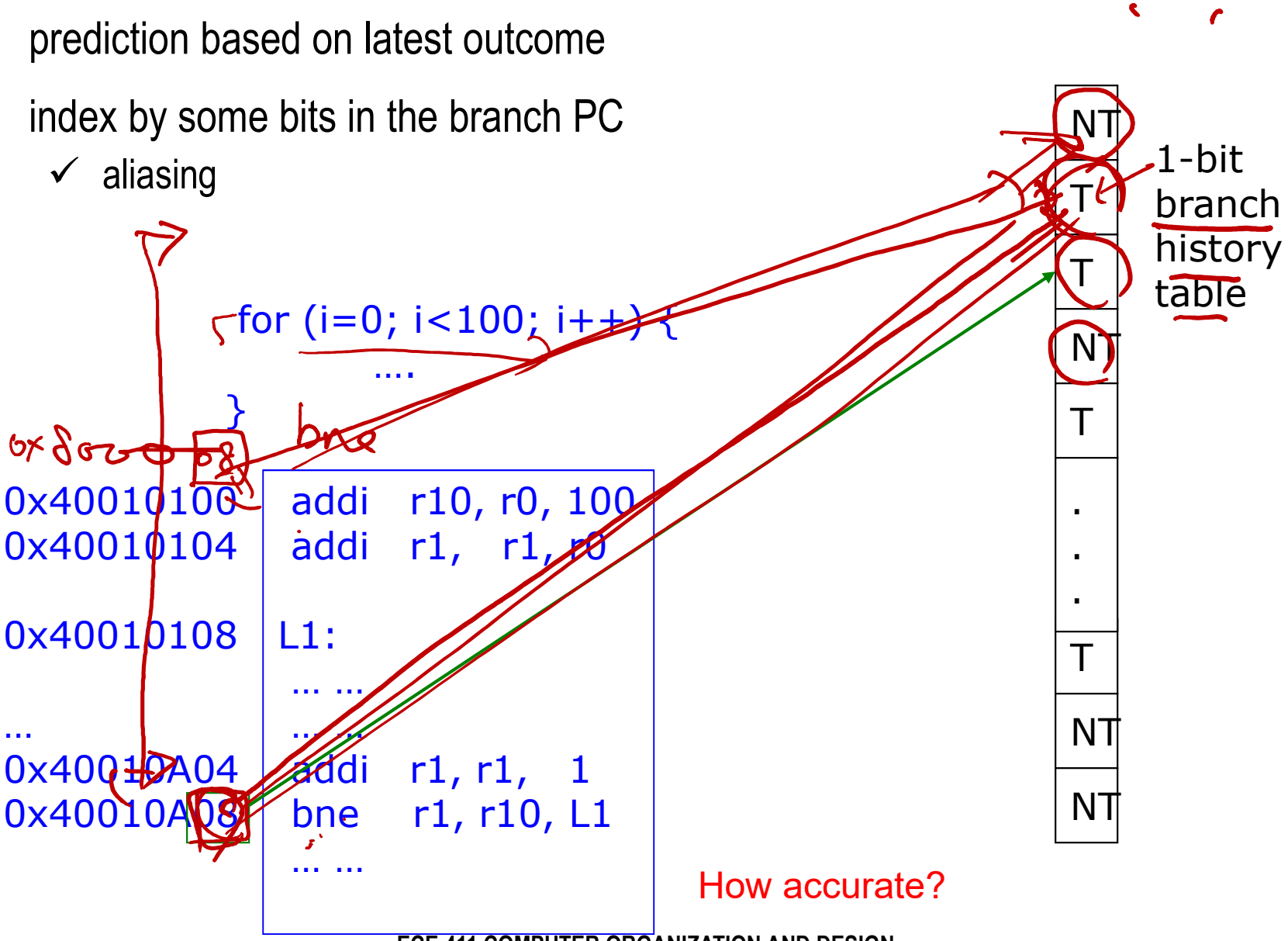
make sure R7 will not be used in taken path before redefined

Predict What?

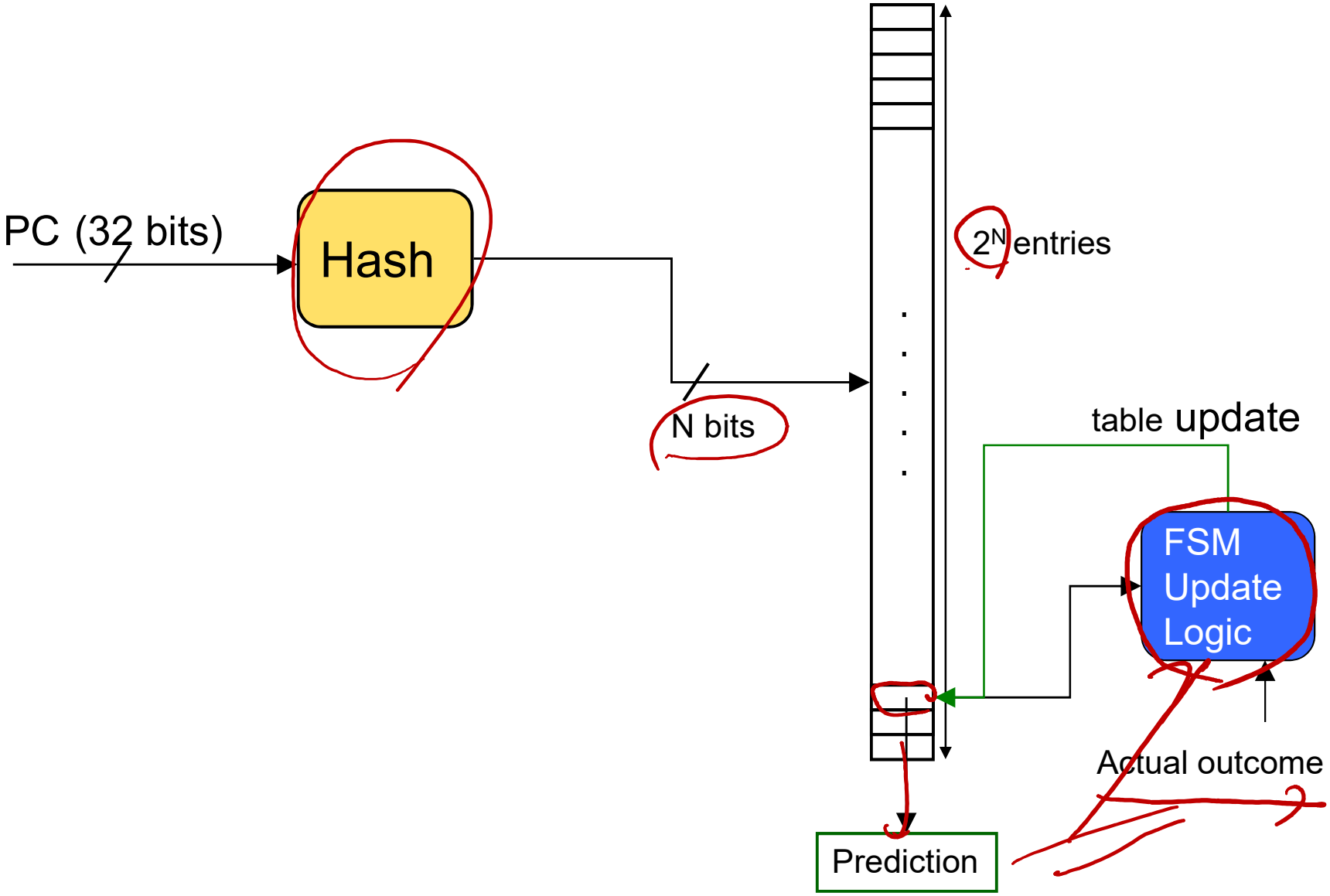
- direction (1-bit)
  - ✓ single direction for unconditional jumps and calls/returns
  - ✓ binary for conditional branches
- target (32-bit or 64-bit addresses)
  - ✓ one
    - uni-directional jumps
  - ✓ two
    - fall through (not Taken) vs. taken
  - ✓ many:
    - function pointer or indirect jump (e.g., jr r31)

Simplest Dynamic Branch Predictor

- prediction based on latest outcome
- index by some bits in the branch PC
  - ✓ aliasing

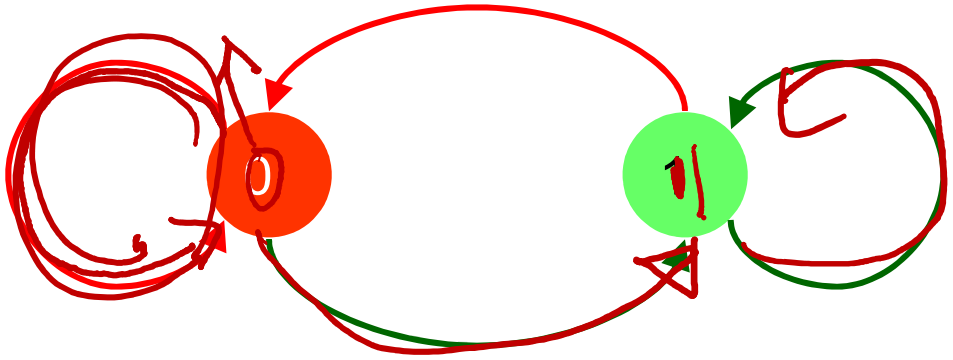


Typical Table Organization



FSM of the Simplest Predictor

- a 2-state machine
- change mind fast



If branch taken

If branch not taken

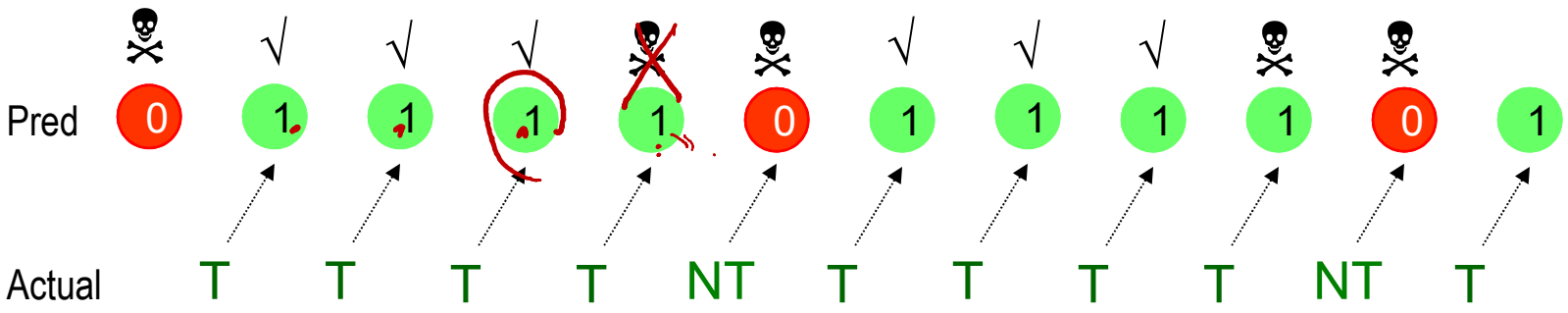
0 Predict not taken

1 Predict taken

Example using 1-bit branch history table

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

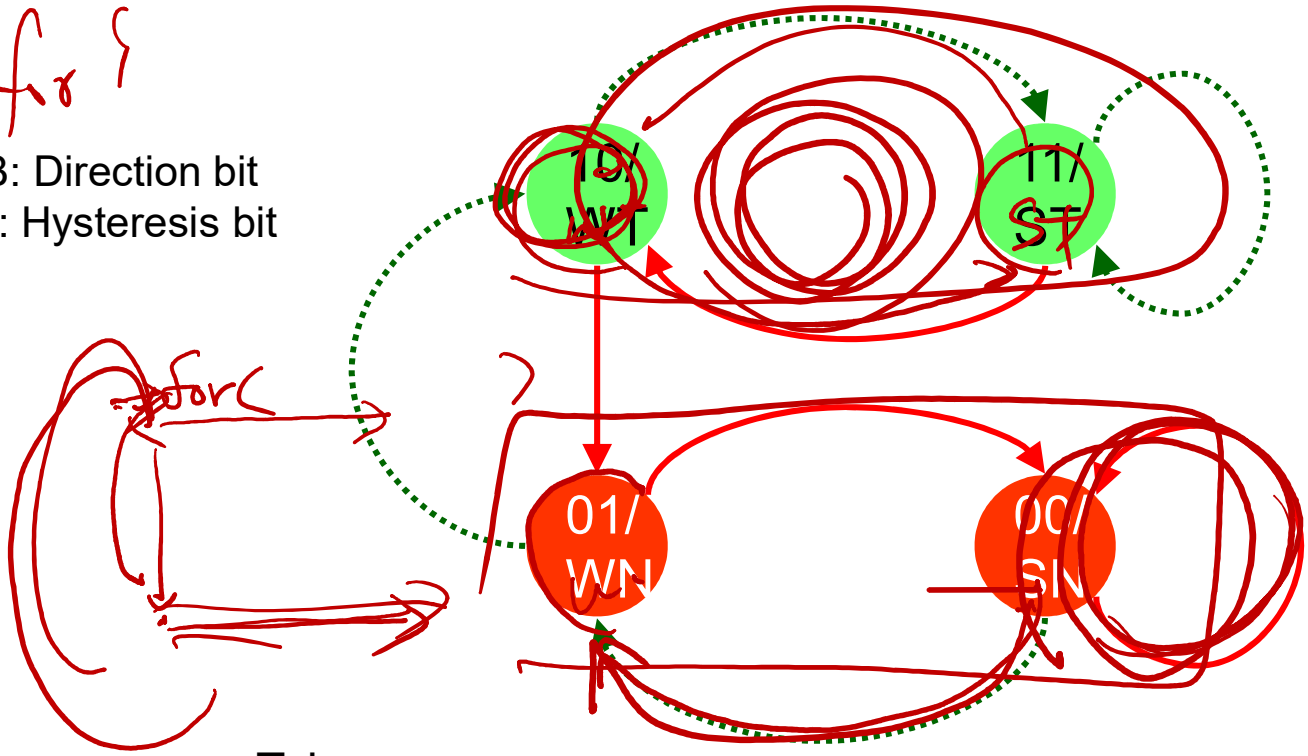
```
 addi   r10, r0, 4
 addi   r1,   r1, r0
L1:
 … …
 addi   r1, r1,   1
 bne    r1, r10, L1
```



60% accuracy

2-bit Sat. Up/Down Counter Predictor

MSB: Direction bit
LSB: Hysteresis bit



Taken

Not Taken

Predict Not taken

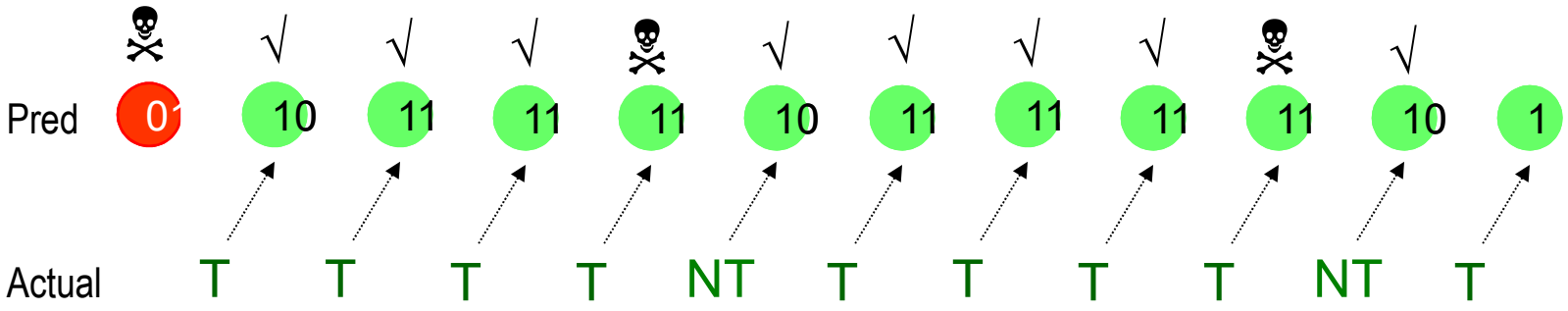
Predict taken

ST: Strongly Taken
WT: Weakly Taken
WN: Weakly Not Taken
SN: Strongly Not Taken

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
```



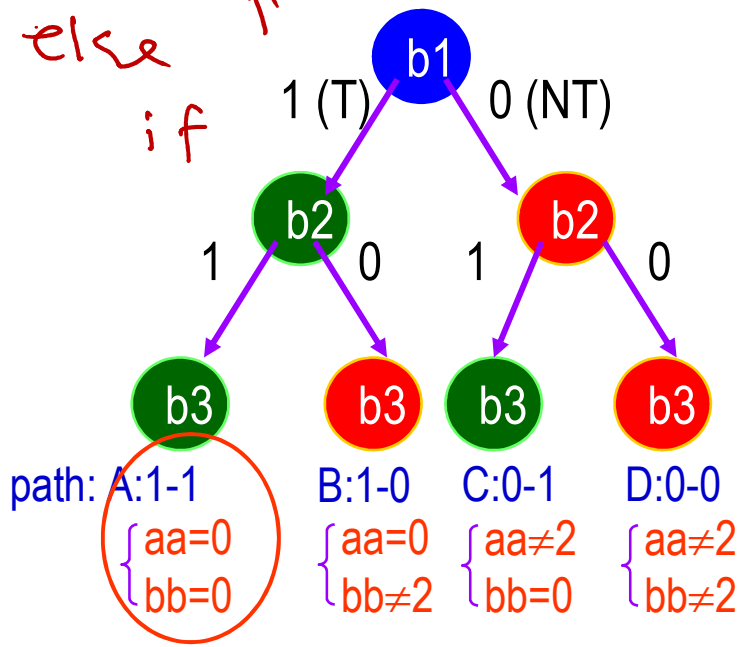
80% accuracy

Branch Correlation

- 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

code snippet

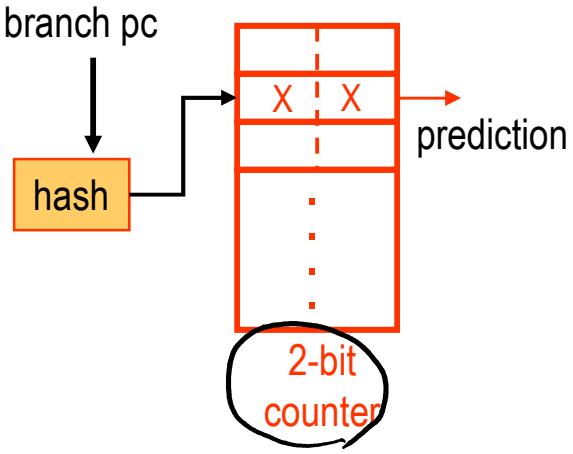
```
if (aa==2)      // b1
    aa = 0;
if (bb==2)      // b2
    bb = 0;
if (aa!=bb) {   // b3
    ....
}
```



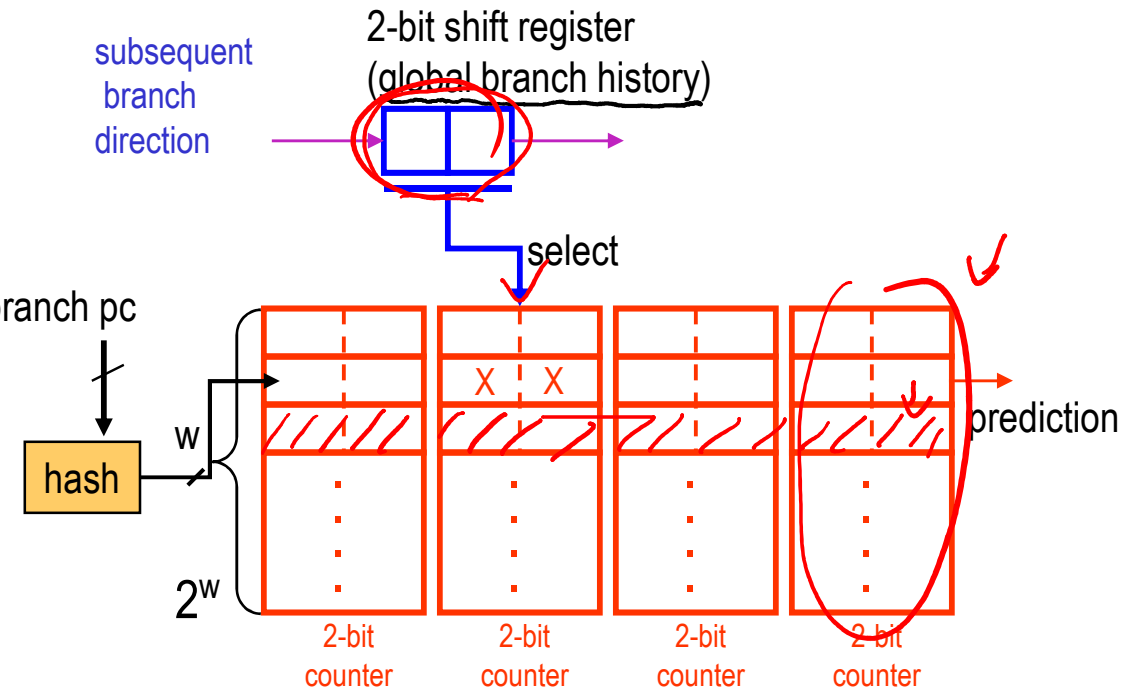
Correlated Branch Predictor

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

[PanSoRahmeh'92]



2-bit sat. counter scheme

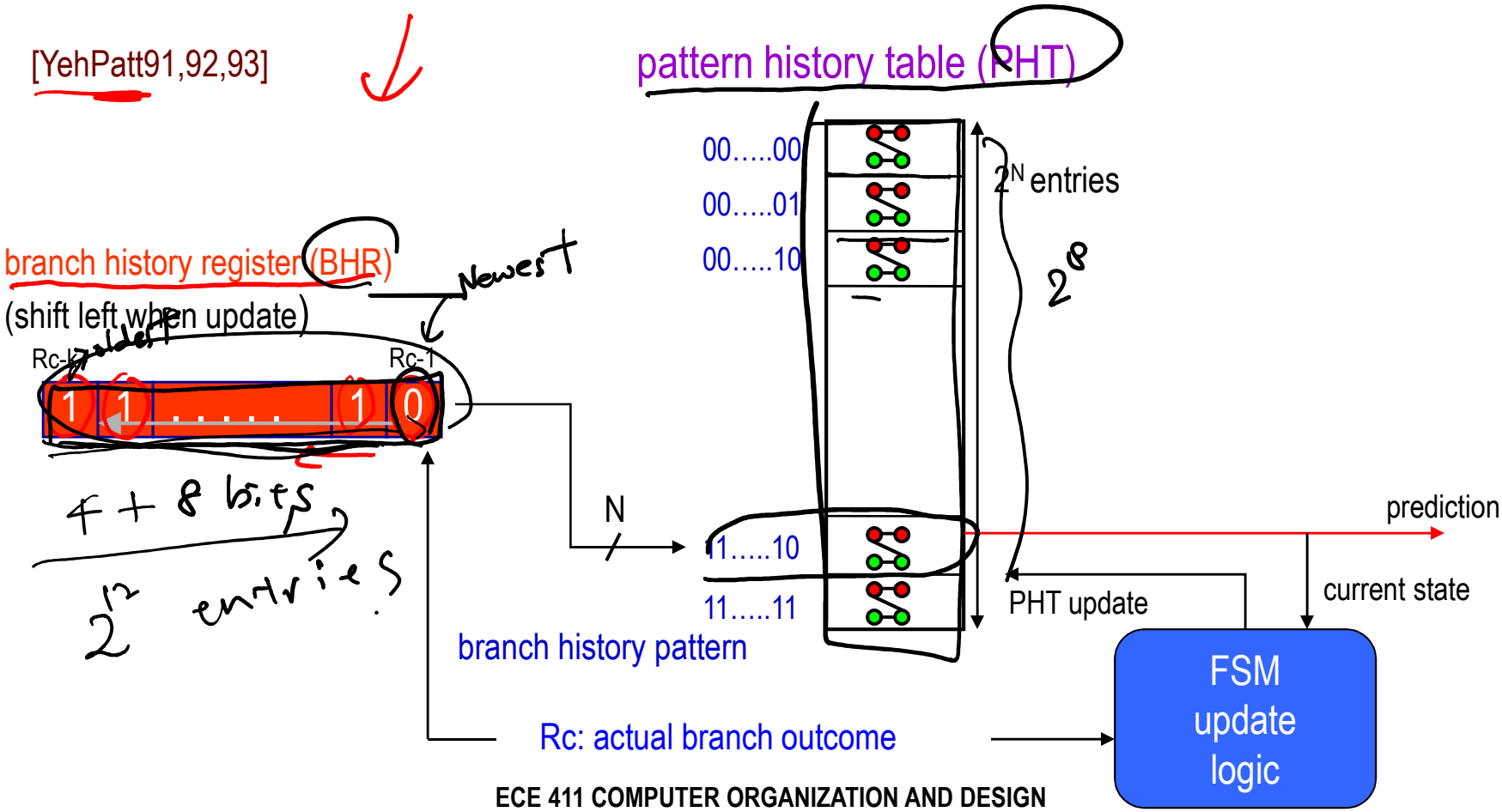


(2,2) correlation scheme

Two-Level Branch Predictor

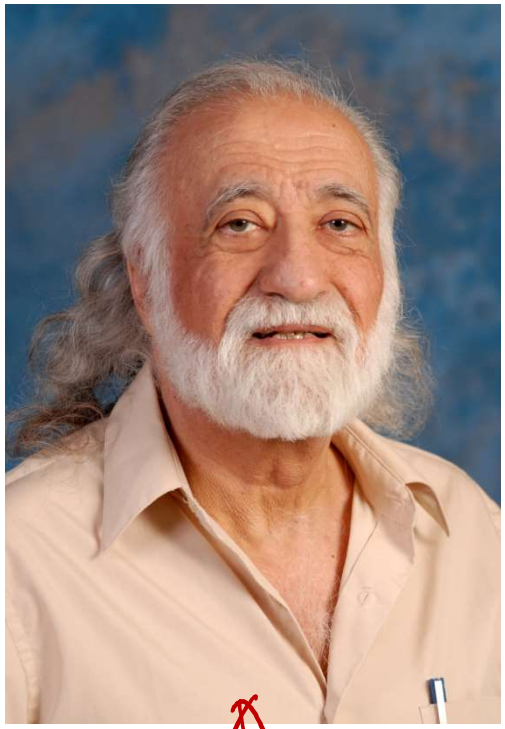
- generalized correlated branch predictor
  - ✓ 1st level keeps branch history in branch hist reg (BHR)
  - ✓ 2nd level keeps pattern history in pattern hist. tab. (PHT)

[YehPatt91,92,93]



Branch History Register

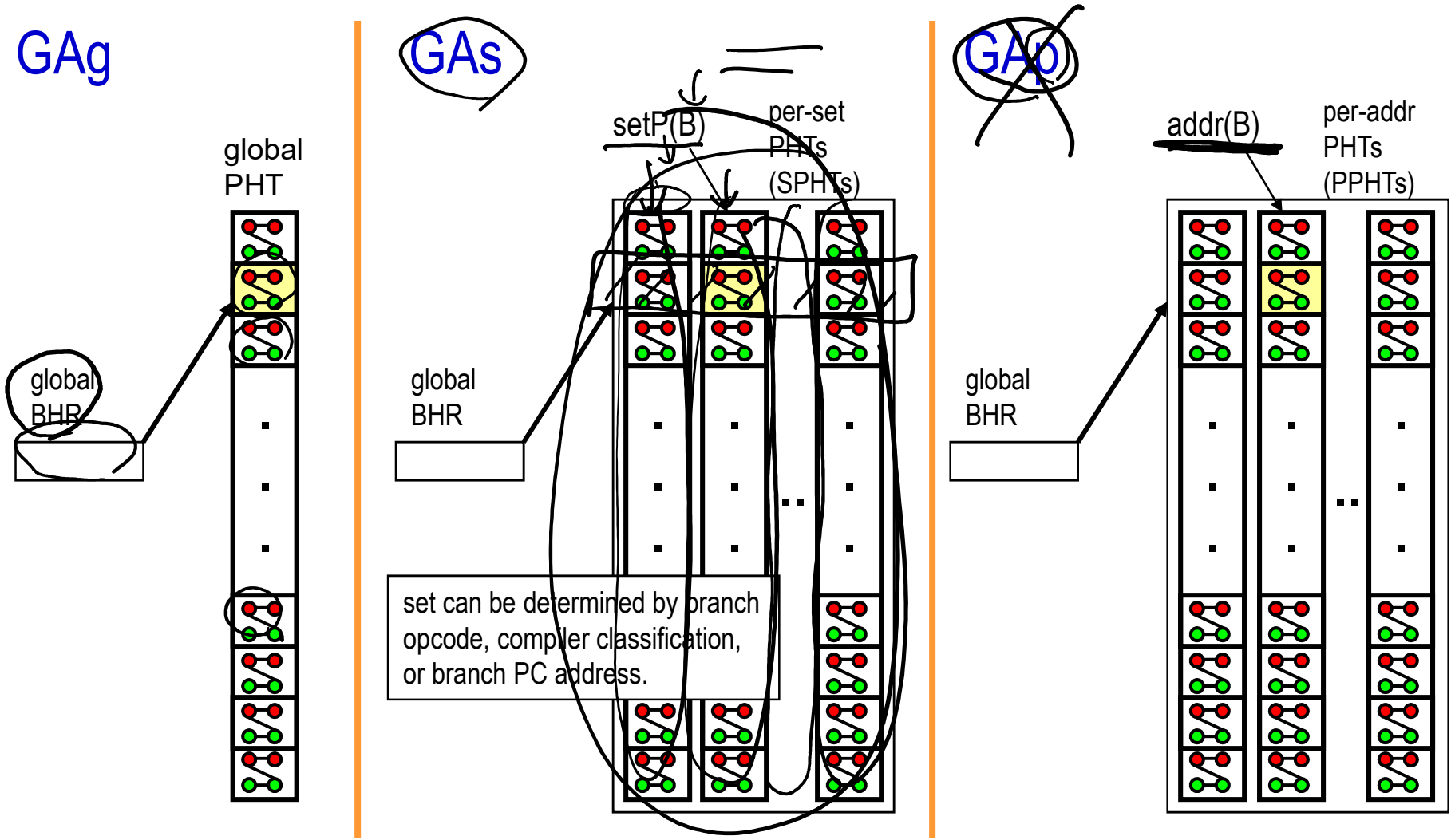
- N-bit shift register = 2^n patterns in pht
- shift-in branch outcomes
  - ✓ 1 ⇒ taken
  - ✓ 0 ⇒ not taken
- first-in first-out
- BHR can be
  - ✓ global
  - ✓ per-set
  - ✓ local (per-address)



Pattern History Table

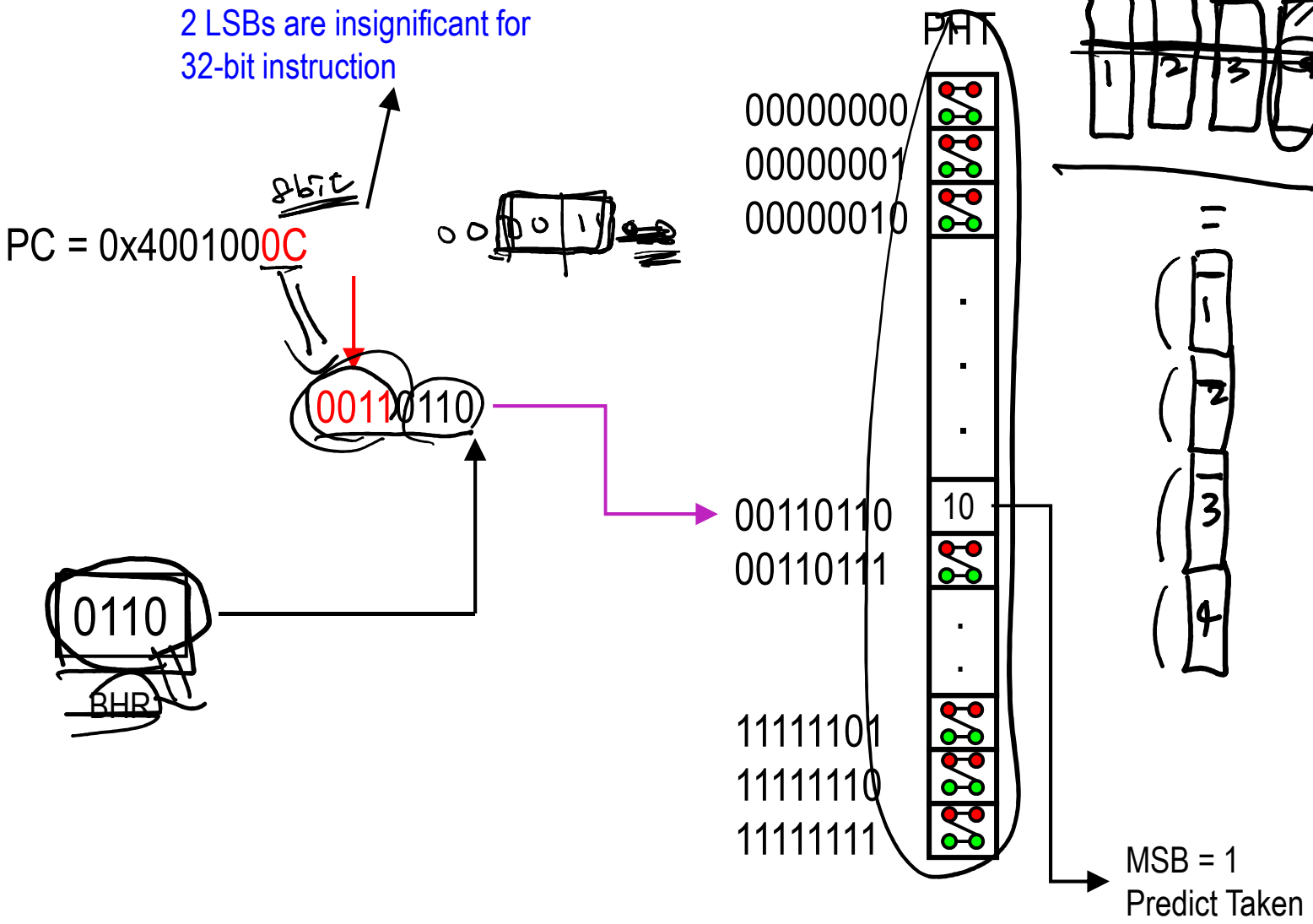
- 2^N entries addressed by N-bit BHR
- each entry keeps a counter (2-bit or more) for prediction
  - ✓ counter update: the same as 2-bit counter
  - ✓ can be initialized in alternate patterns (01, 10, 01, 10, ..)
- alias (or interference) problem

Global History Schemes



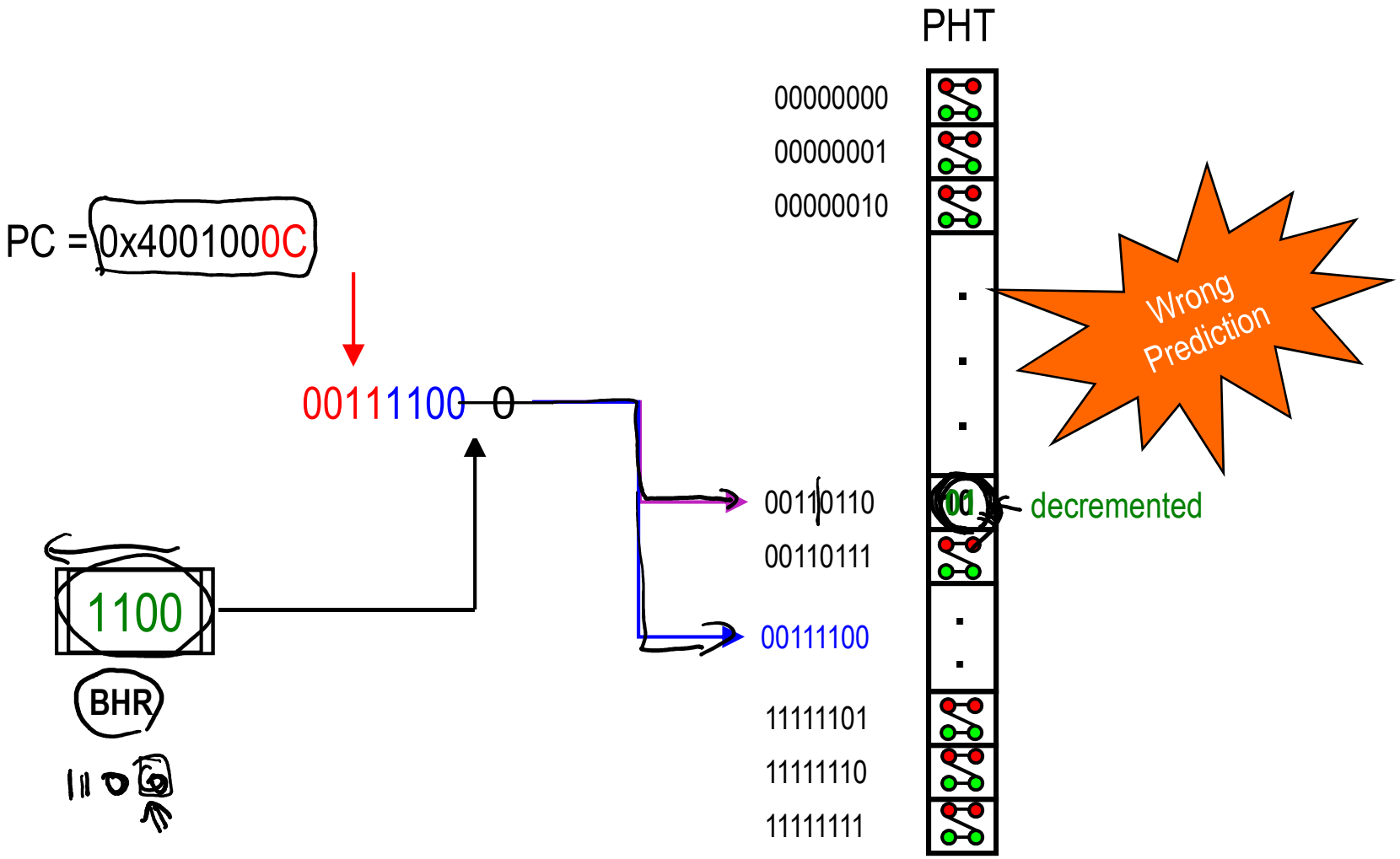
* [PanSoRahmeh'92] similar to GAp

GAs Two-Level Branch Prediction

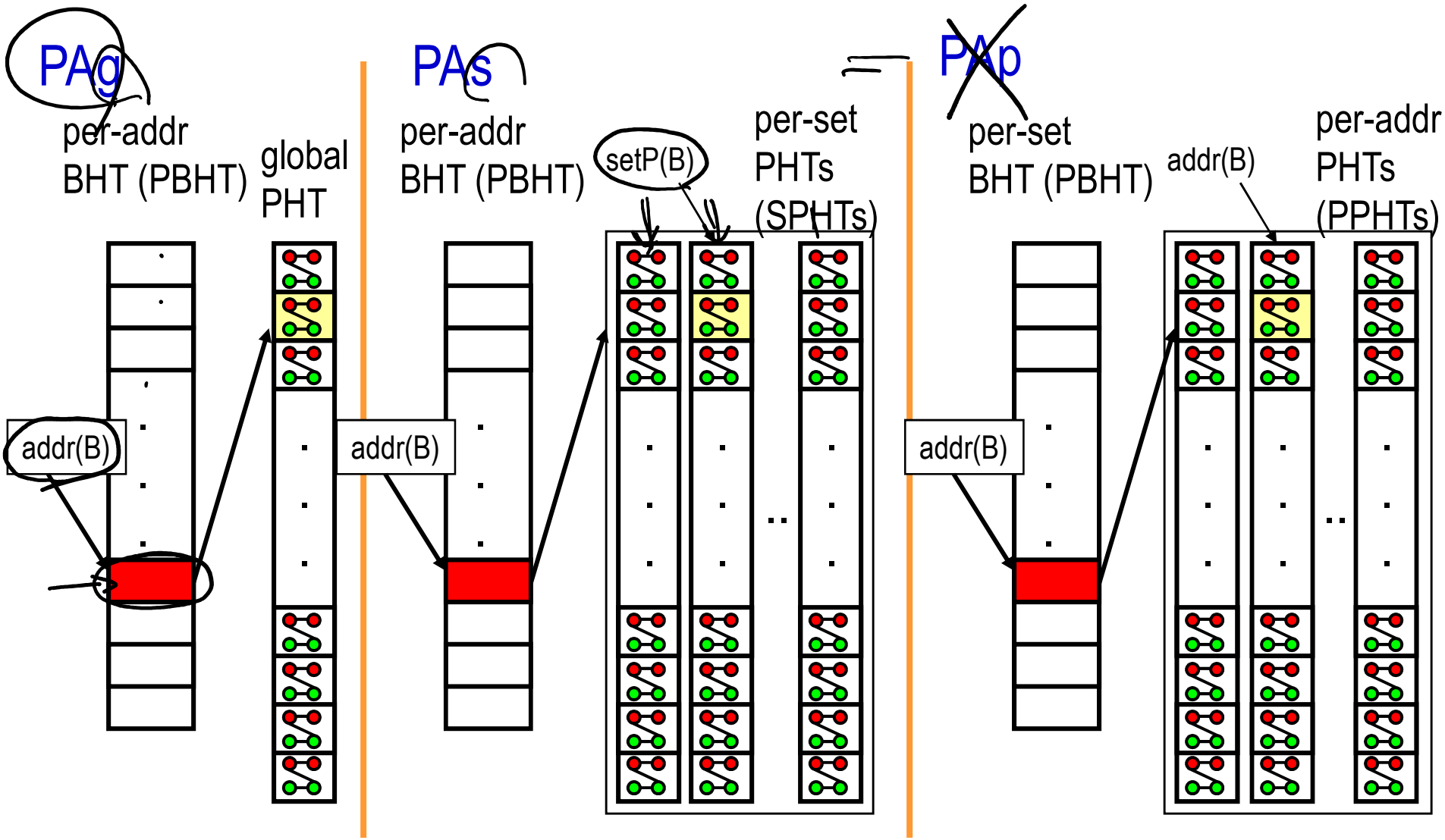


Predictor Update (Actually, Not Taken)

- update predictor after branch is resolved



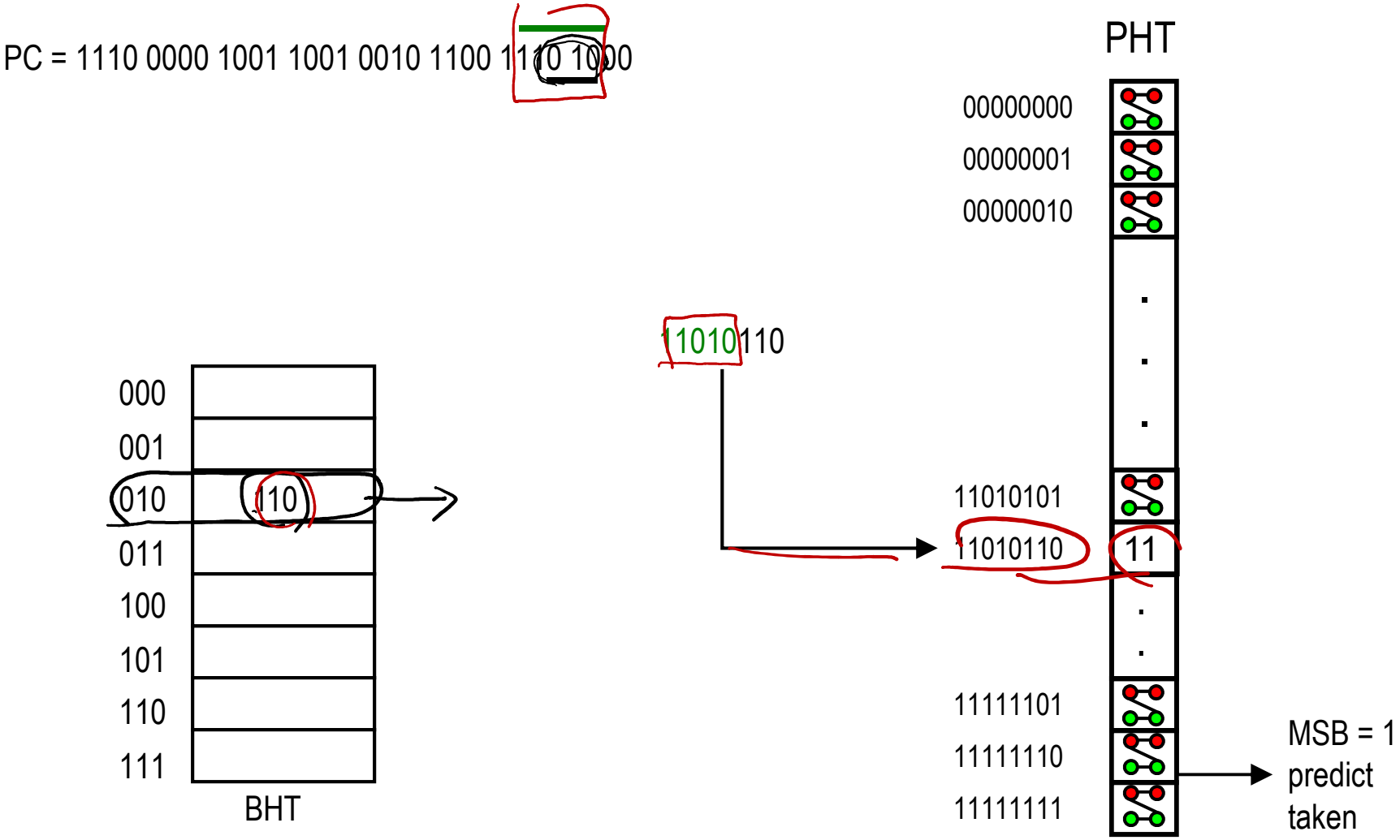
Per-Address History Schemes



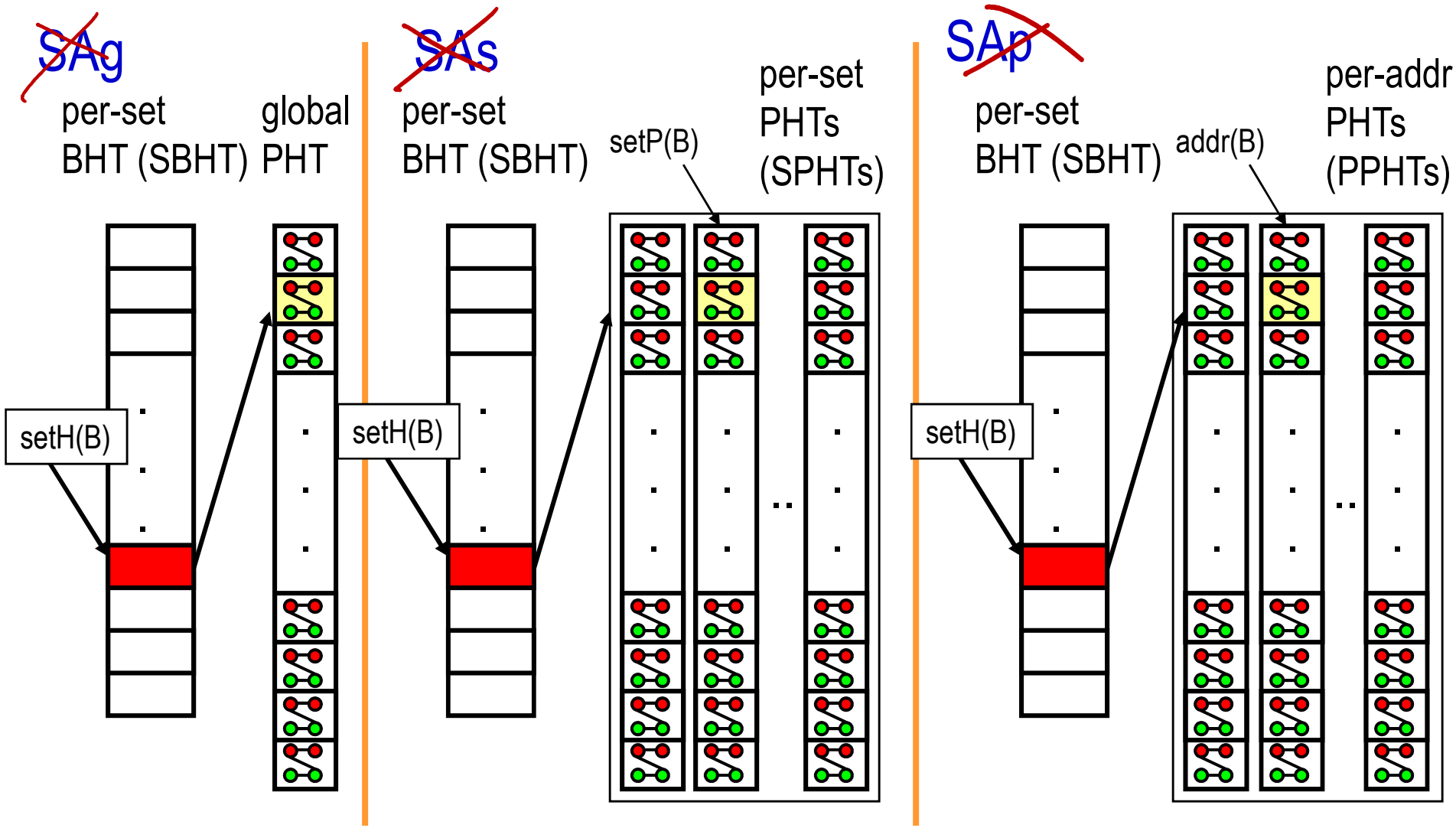
Alpha 21264's local predictor

P6, Itanium

PAs Two-Level Branch Predictor



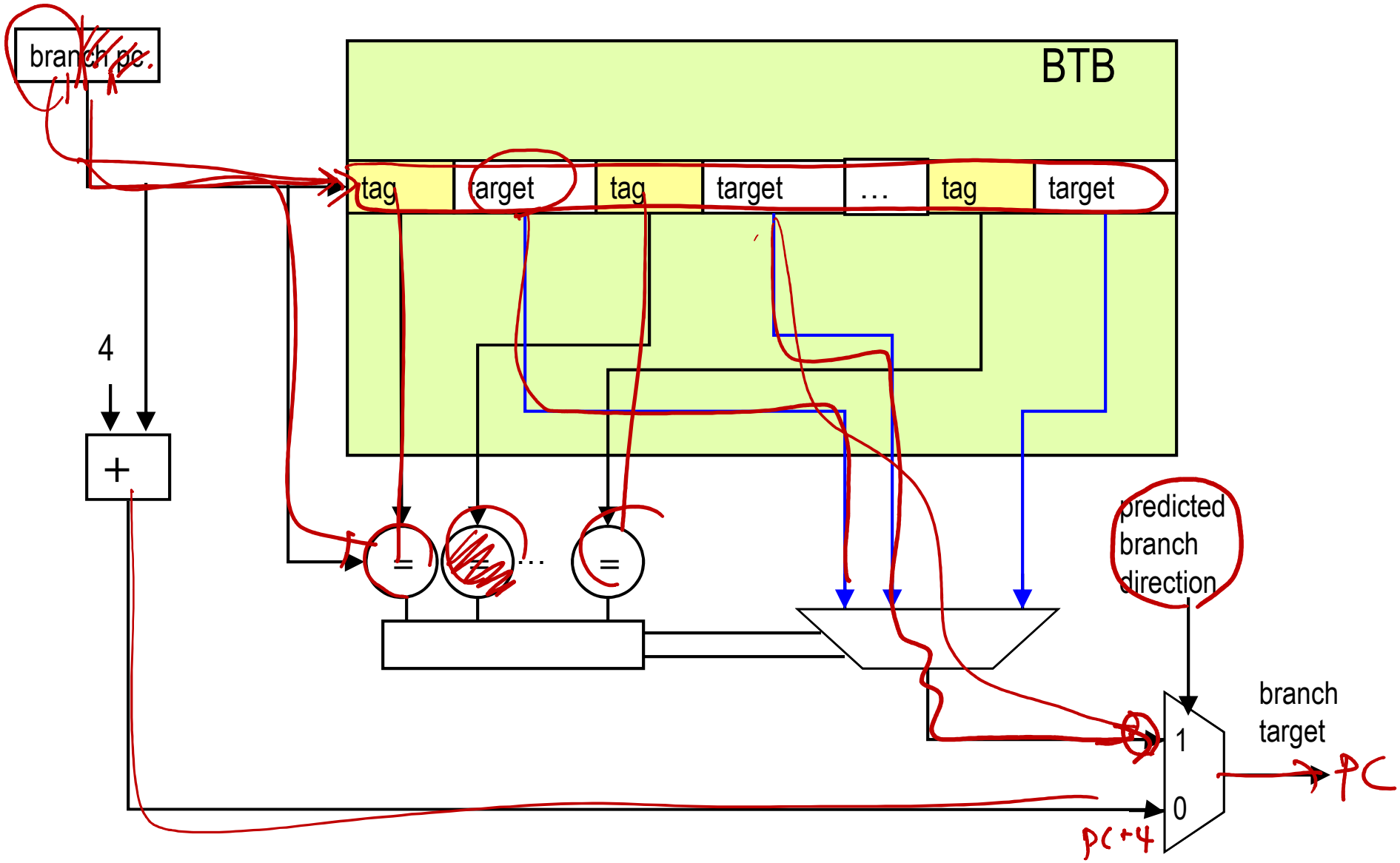
Per-Set History Schemes



BTB Operation

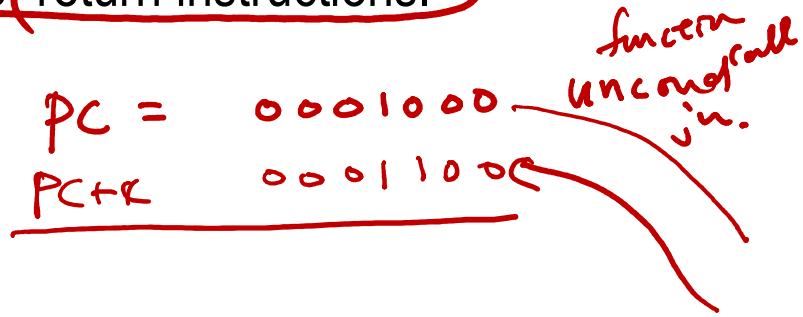
- use PC (all bits) for lookup
  - ✓ Match implies this is a branch
- if match and predict bits: taken, set PC to predicted PC
- if branch predict wrong: must recover (same as branch hazards we've already seen)
- if decode indicates branch w/ no BTB match, two choices:
  - ✓ look up prediction now and act on it
  - ✓ just predict not taken (= PC+4)
- when branch resolved, update BTB (at least prediction bits, maybe more)

Branch Target Buffer (BTB)



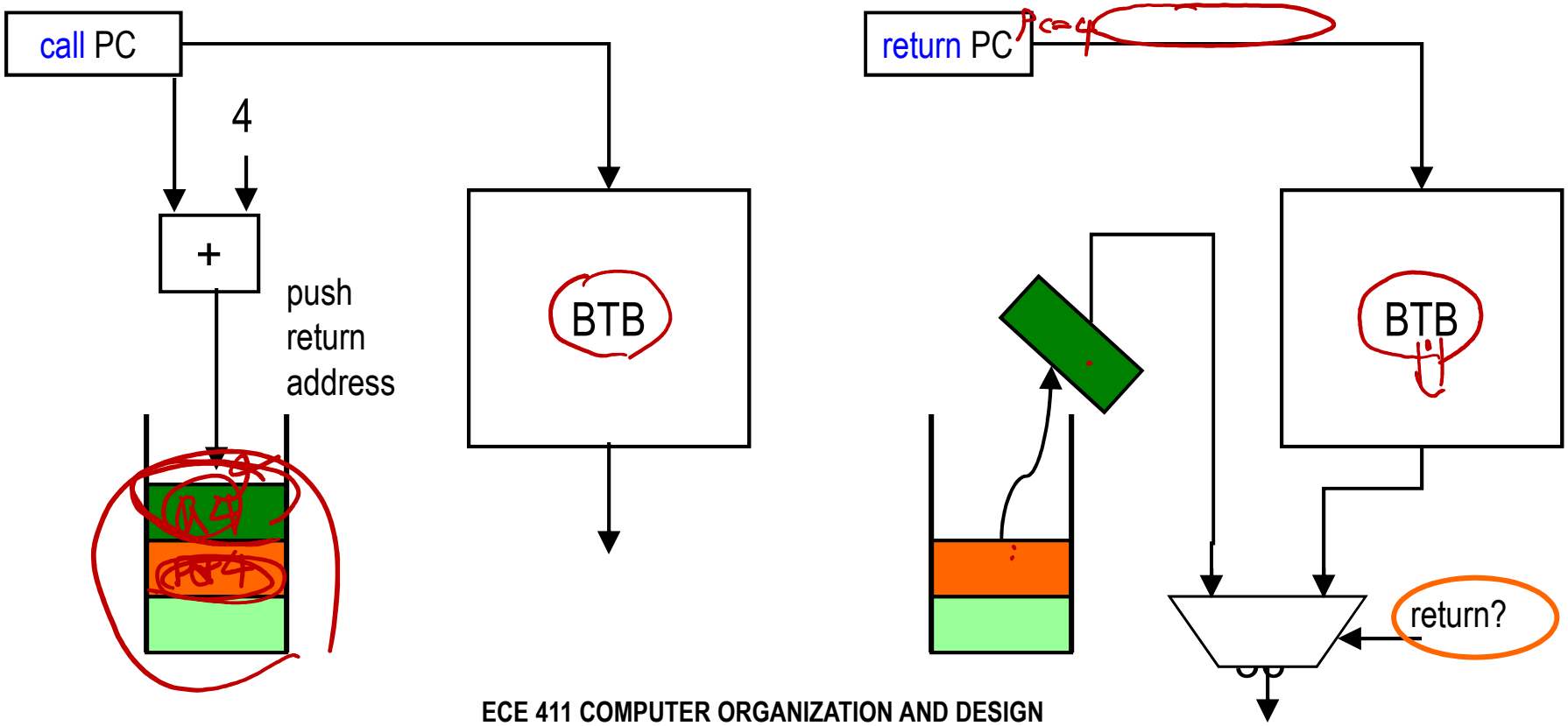
What about indirect jumps/returns?

- branch predictor does really well with conditional jumps
- BTB does really well with unconditional jumps (jump, jal, etc.)
- indirect jumps often jump to different destinations, even from the same instruction. Indirect jumps most often used for return instructions.
- procedure calls and returns
  - ✓ calls are always taken
  - ✓ return address almost always known
- return address stack (RAS)
  - ✓ on a procedure call, push the address of the instruction after the call onto the stack



Return Address Stack (RAS)

- 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



Calculating the Cost of Branches

Factors to consider:

- branch frequency (every 4-6 instructions)
- correct prediction rate
  - ✓ 1 bit: ~ 80% to 85%
  - ✓ 2 bit: ~ high 80s to 90%
  - ✓ correlated branch prediction: ~95%
- misprediction penalty w/ (multiply by the instruction width)
  - ✓ Alpha 21164: 5 cycles; 21264: 7 cycles
  - ✓ UltraSPARC 1: 4 cycles
  - ✓ Pentium Pro: at least 9 cycles, 15 on average
- or misfetch penalty,
  - ✓ have the correct prediction but not know the target address yet (may also apply to unconditional branches)

Calculating the Cost of Branches

What is the probability that a branch is taken?

- Given:
  - ✓ 20% of branches are unconditional branches
  - ✓ of conditional branches,
    - 66% branch forward & are evenly split between taken & not taken
    - the rest branch backwards & are almost always taken

Calculating the Cost of Branches

What is the contribution to CPI of conditional branch stalls, given:

- ✓ 15% branch frequency
- ✓ a BTB for conditional branches only w/
  - 10% miss rate
  - 3-cycle miss penalty
  - 92% prediction accuracy
  - 7 cycle misprediction penalty
- ✓ base CPI is 1

BTB result	Prediction	Frequency (per instruction)	Penalty (cycles)	*Stalls*
miss	--	.15 * .10 = .015	3	.045
hit	correct	.15 * .90 * .92 = .124	0	0
hit	incorrect	.15 * .90 * .08 = .011	7	.076
Total contribution to CPI				**.121**

1.4 CPI = 1.121

Announcement

- this's lecture: pipeline
  - ✓ Ch. 4.8 (HP1), C.26—C30 (HP2),
  - ✓ cf. paper – T.-Y. Yeh and Y.N. Patt, "Two-Level Adaptive Training Branch Prediction," International Symposium on Microarchitecutre, 1991; https://dl.acm.org/citation.cfm?id=123475
- MP assignment
  - ✓ MP2 checkpoint due on 2/18 5pm