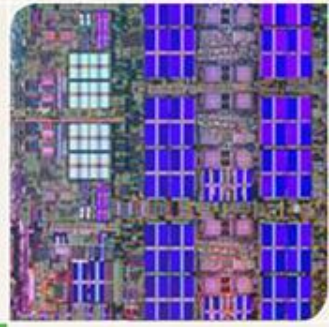
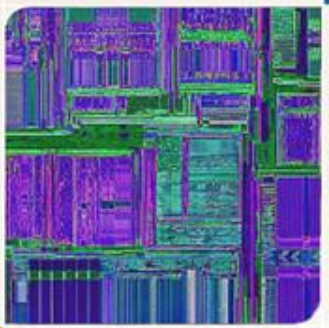
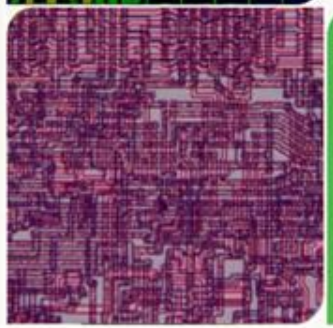



CS4290/CS6290

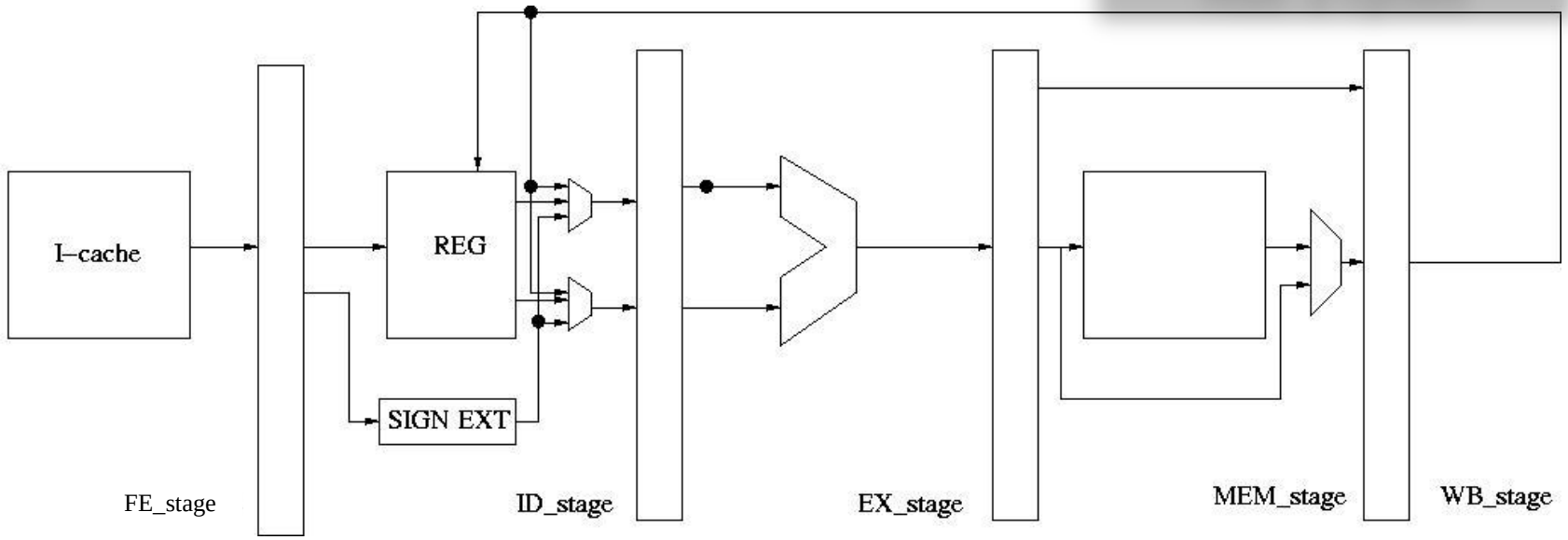
Fall 2011

Prof. Hyesoon Kim

Georgia Tech College of Computing

Dependent Instructions: dst data is available at WB

Add: 2 cycles



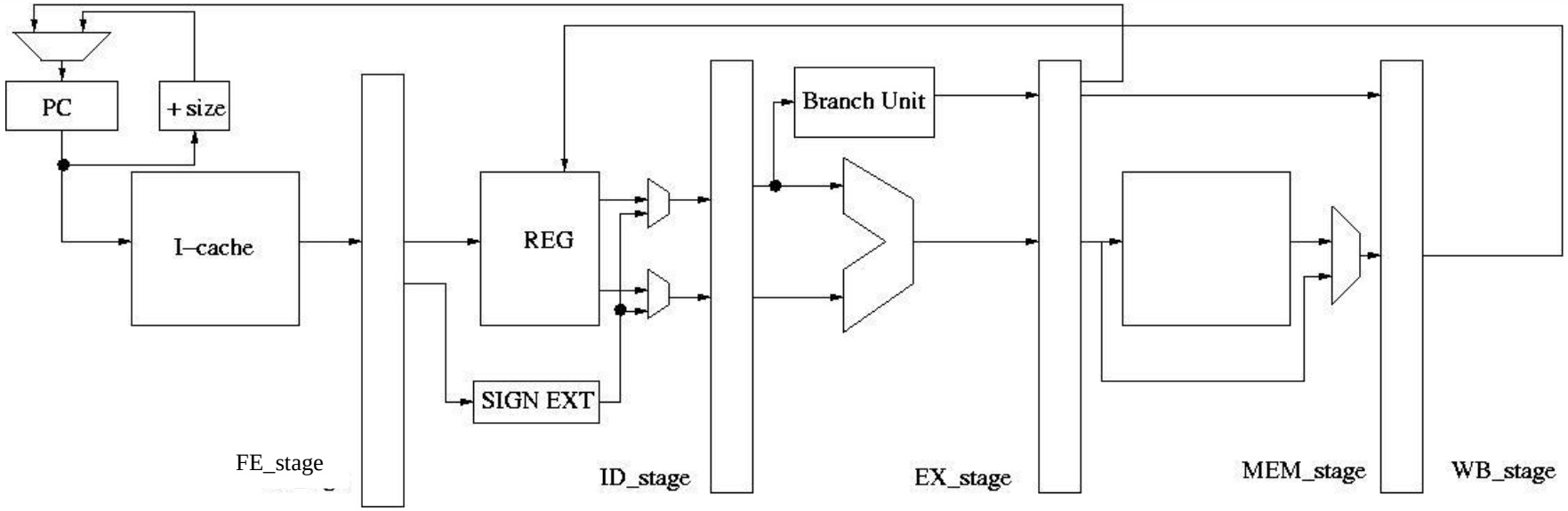

```
add  r1, r2, r3
sub  r4, r1, r3
mul  r5, r2, r3
```

FE	L	ID	L	EX	L	MEM	L	WB	L
add	add								
sub	sub	add	add						
mul	sub	sub	add	add					
mul	sub	sub		add	add				
mul	sub	sub			add	add			
mul	mul	sub	sub				add		

(Lab#1)

```
        br    target    0x800
        add r1, r2,r3   0x804

target sub r2,r3,r4    0x900
```



cycle	PC (latch)	FE	ID	EX	MEM	WB
1	0x800	br				
2	0x804	add	br			
3	0x804	add		br		
4	0x804	add			br	
5	0x900	sub				br
6	0x904	add	sub			

What to do with branches?

- -Eliminate branches
  - Predication (more on later)
- Delayed branch slot
  - SPARC, MIPS
- Dual-path execution (more on later)
- Or predict? ☺



Control Dependencies

- Branches are very frequent
  - Approx. 20% of all instructions
- Can not wait until we know where it goes
  - Long pipelines
    - Branch outcome is known after B cycles
    - No scheduling past the branch until outcome known
  - Superscalars (e.g., 4-way)
    - Branch every cycle or so!
    - One cycle of work, then bubbles for ~B cycles?



Surviving Branches: Prediction

- Predict Branches
  - And predict them well!
- Fetch, decode, etc. on the predicted path
  - Option 1: No execution until branch is resolved
  - Option 2: Execute anyway (speculation)
- Recover from mispredictions
  - Restart fetch from correct path



Branch Prediction

- Need to know two things
  - Whether the branch is taken or not (direction)
  - The target address if it is taken (target)
- Direct jumps, Function calls: unconditional branches
  - Direction known (always taken), target easy to compute
- Conditional Branches (typically PC-relative)
  - Direction difficult to predict, target easy to compute
- Indirect jumps, function returns
  - Direction known (always taken), target difficult



Branch Prediction: Direction

- Needed for conditional branches
  - Most branches are of this type
- Many, many kinds of predictors for this
  - Static: fixed rule, or compiler annotation (e.g. br.bwh (branch whether hint. IA-64))
  - Dynamic: hardware prediction
- Dynamic prediction usually history-based
  - Example: predict direction is the same as the last time this branch was executed



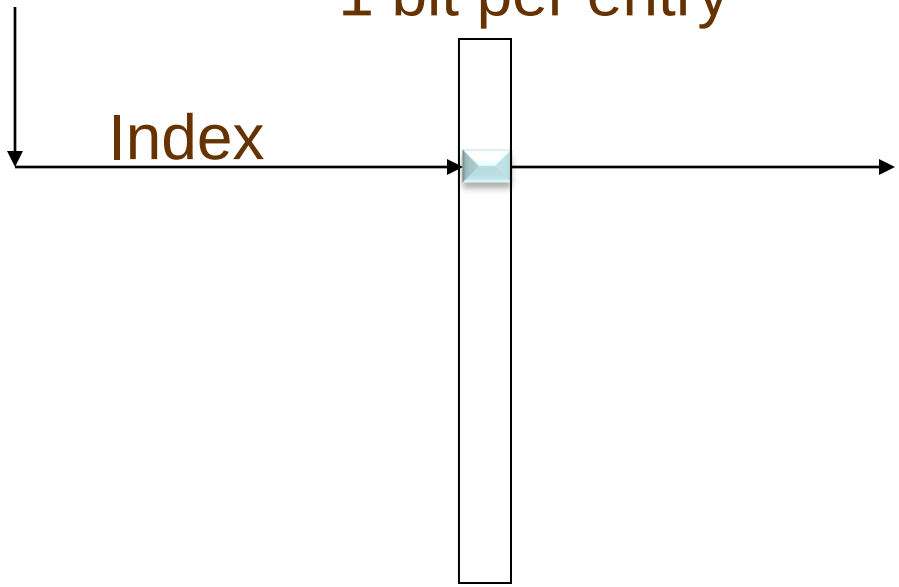
Static Prediction

- Always predict NT
  - easy to implement
  - 30-40% accuracy … not so good
- Always predict T
  - 60-70% accuracy
- BTFNT
  - loops usually have a few iterations, so this is like always predicting that the loop is taken
  - don't know target until decode

One-Bit Branch Predictor: Last-time predictor

K bits of branch instruction address

Branch history table of 2^K entries, 1 bit per entry

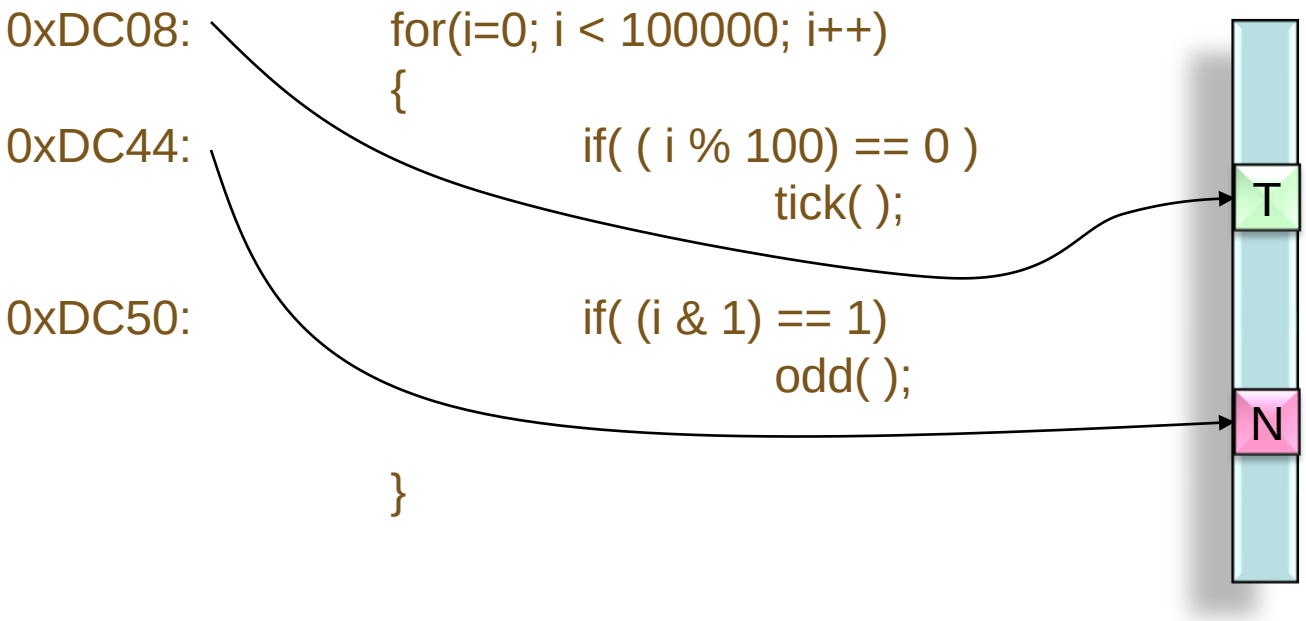


Use this entry to predict this branch:

0: predict not taken
1: predict taken

When branch direction resolved, go back into the table and update entry: 0 if not taken, 1 if taken

One-Bit Branch Predictor (cont'd)





The Bit Is Not Enough!

- Example: short loop (8 iterations)
  - Taken 7 times, then not taken once
  - Not-taken mispredicted (was taken previously)

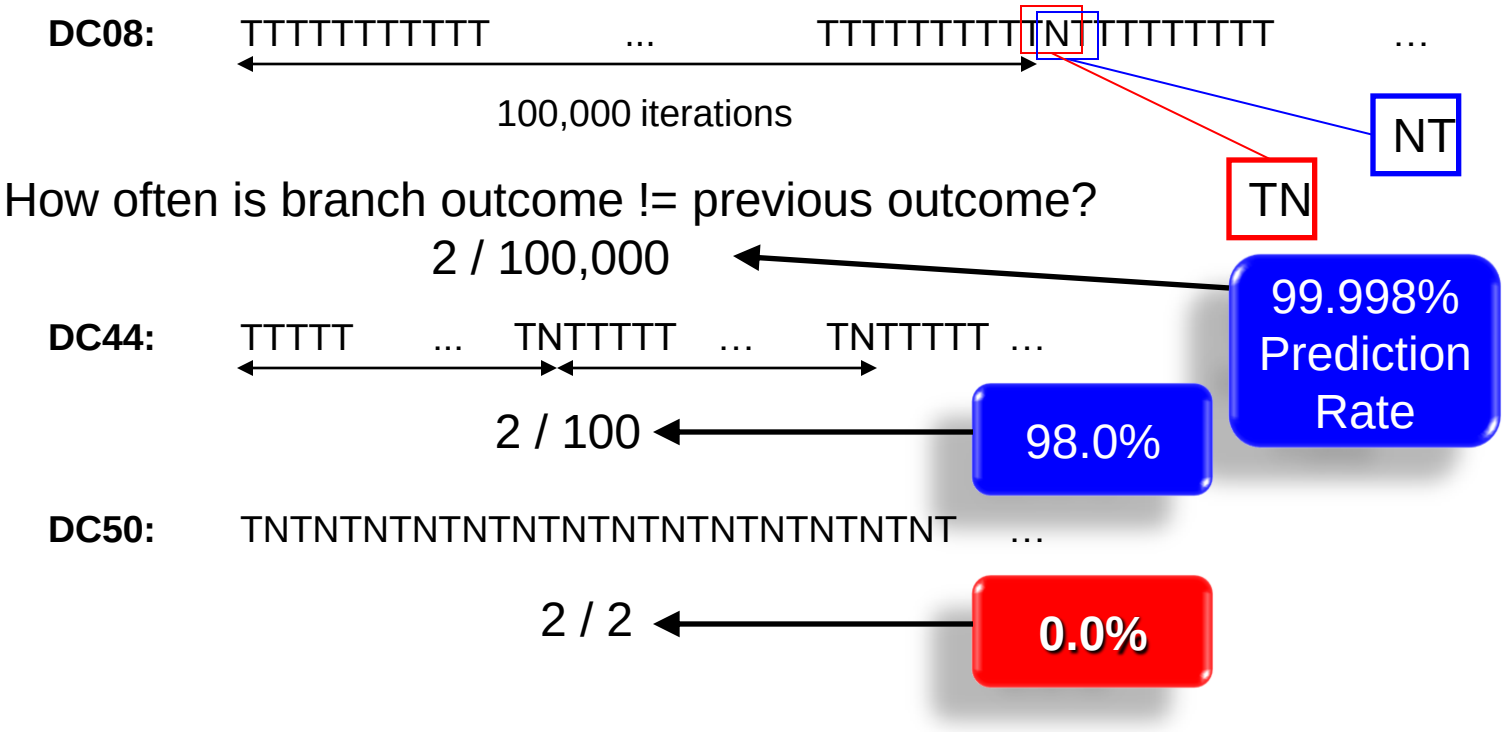
  Act: TTTTTTTNTTTTTTTNTTTTTTTNT…

  Pred: XTTTTTTTNTTTTTTNTTTTTTTN

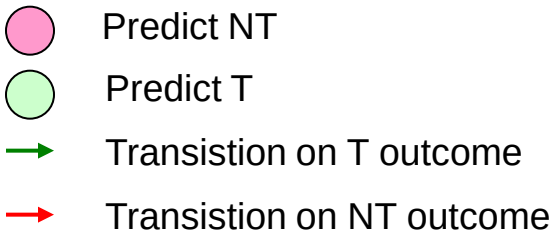
  Corr: XoooooMMooooooMMooooooMM

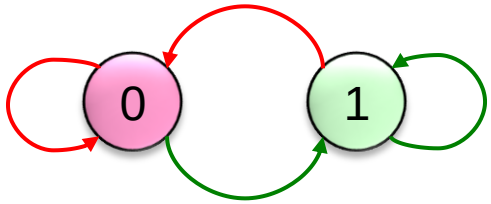
  Misprediction rate: 2/8 = 25%
- Execute the same loop again
  - First always mispredicted (previous outcome was not taken)
  - Then 6 predicted correctly
  - Then last one mispredicted again
- Each fluke/anomaly in a stable pattern results in two mispredicts per loop

Examples

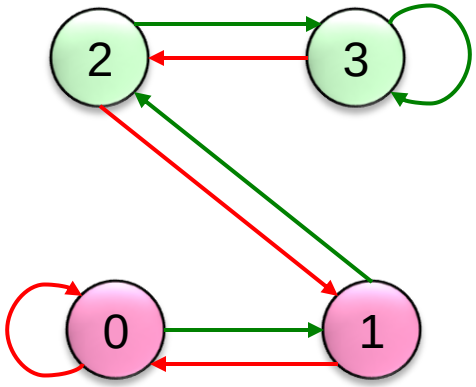

DC08:
TTTTTTTTTTT ... TTTTTTTTTTNTTTTTTTTT …
100,000 iterations
NT
TN
How often is branch outcome != previous outcome?
2 / 100,000
99.998% Prediction Rate
DC44:
TTTTT ... TNTTTTT … TNTTTTT …
2 / 100
98.0%
DC50:
TNTNTNTNTNTNTNTNTNTNTNTNTNTNT …
2 / 2
0.0%

Two Bits are Better Than One



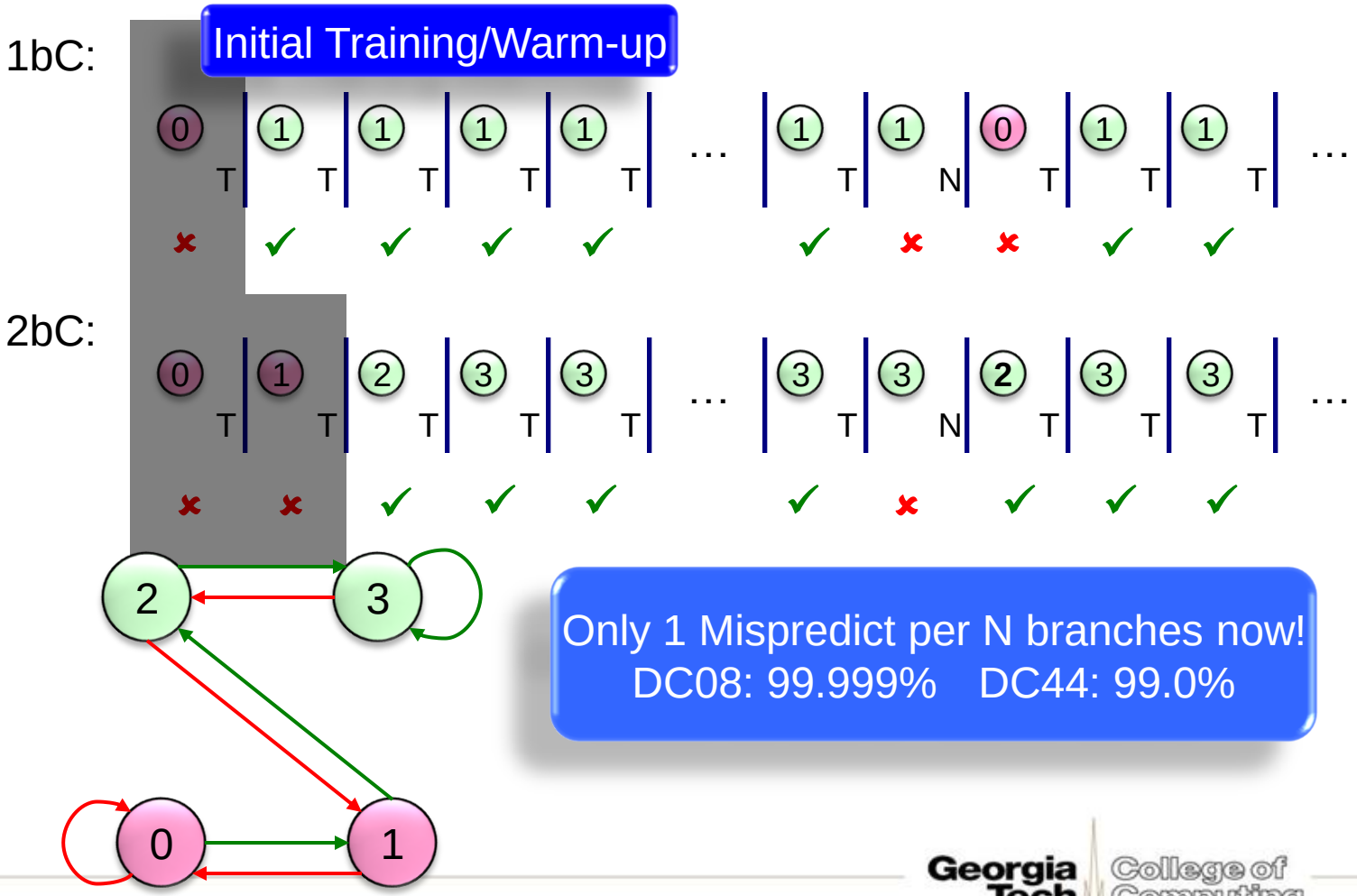


FSM for Last-time Prediction

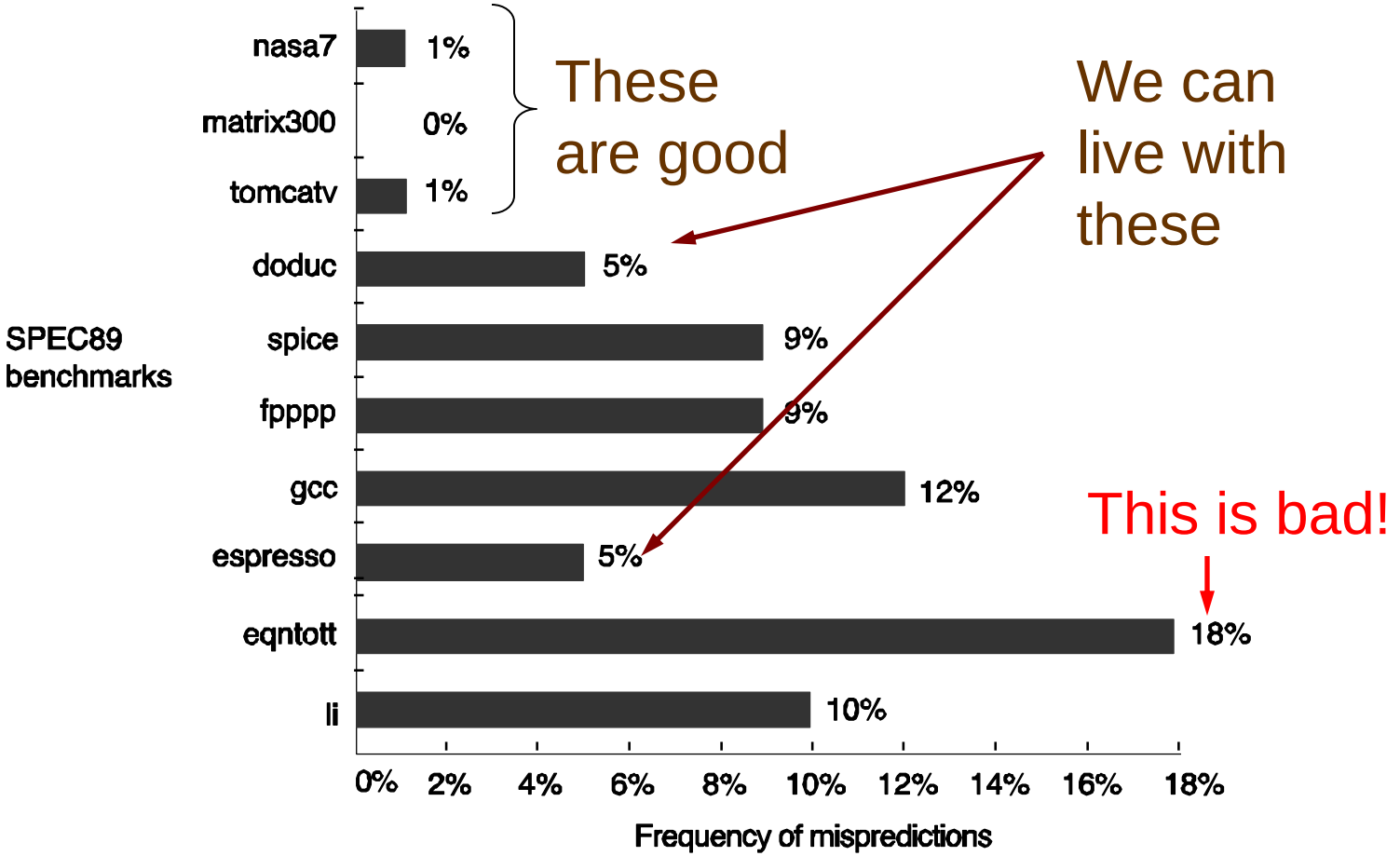


FSM for 2bC (**2**-**b**it **C**ounter)

Example


1bC:
Initial Training/Warm-up
0 T 1 T 1 T 1 T 1 T … 1 T 1 N 0 T 1 T 1 T …
✘ ✓ ✓ ✓ ✓ ✓ ✘ ✘ ✓ ✓
2bC:
0 T 1 T 2 T 3 T 3 T … 3 T 3 N 2 T 3 T 3 T …
✘ ✘ ✓ ✓ ✓ ✓ ✘ ✓ ✓ ✓
2
3
0
1
Only 1 Mispredict per N branches now!
DC08: 99.999% DC44: 99.0%

Still Not Good Enough

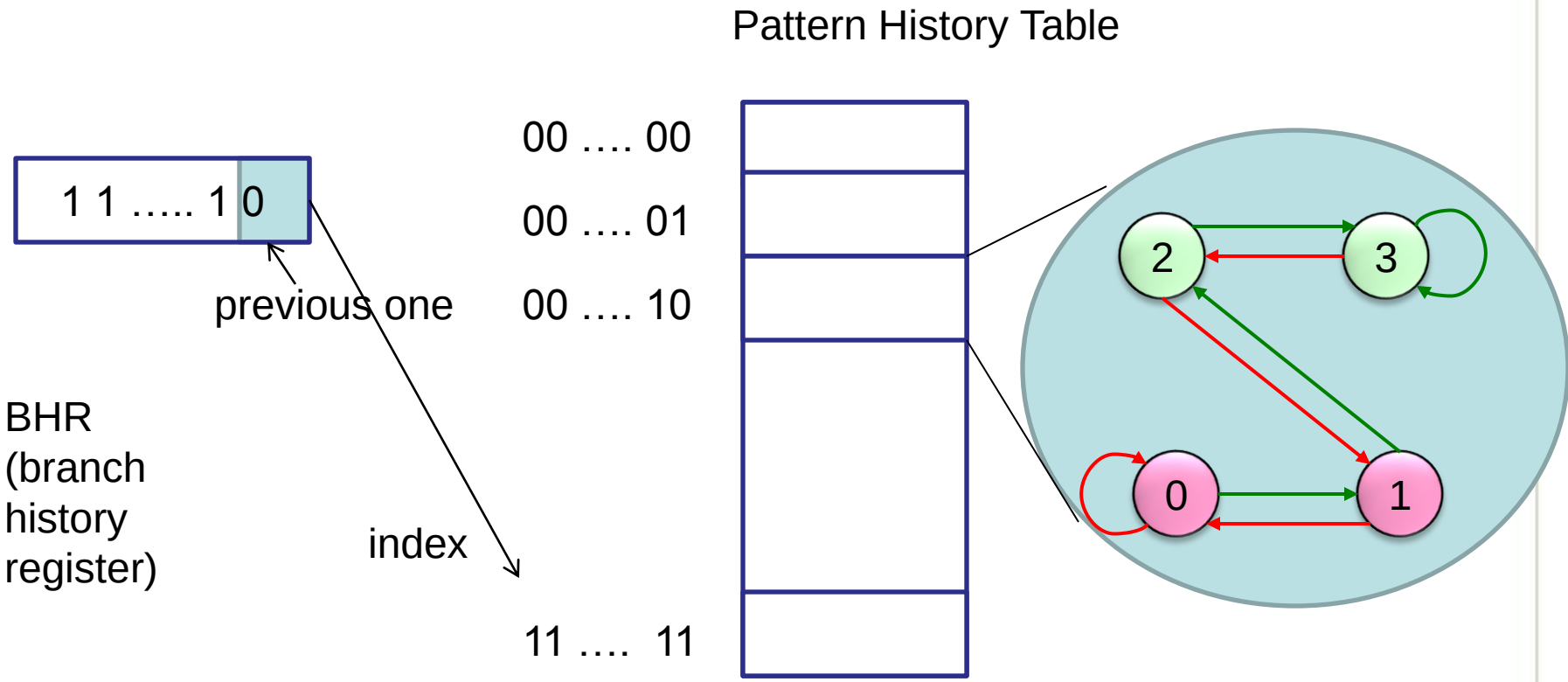




Importance of Branches

- 98% → 99%
  - Who cares?
  - Actually, it's 2% misprediction rate → 1%
  - That's a halving of the number of mispredictions
- So what?
  - If a pipeline can fetch 5 instructions at a cycle and the branch resolution time is 20 cycles
  - To Fetch 500 instructions
  - 100 accuracy : 100 cycles
  - 98 accuracy:
    - 100 (correctly fetch) + 20 (misprediction)*10 = 300 cycles
  - 99 accuracy
    - 100 (correctly fetch) + 20 misprediction *5 = 200 cycles

Two-level Branch Predictor

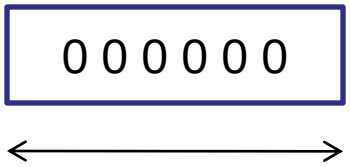


Yeh&patt'92

BHR (Branch History Register)

Initialization value (0 or 1)

Old history New history



History length

1 : branch is taken
0: branch is not-taken

New BHR = old BHR<<1 | (br_dir)

Example

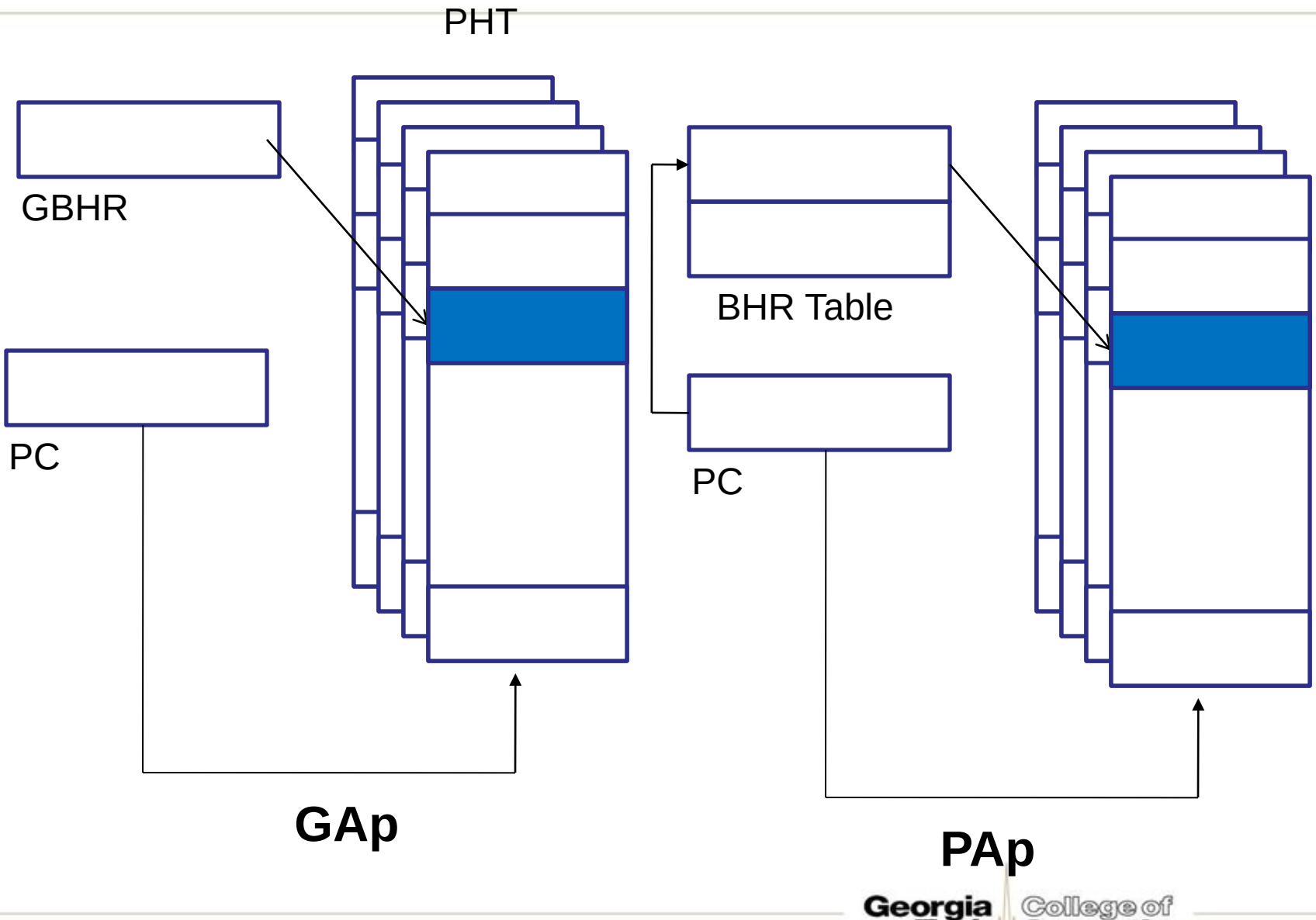
BHR: 00000

Br1 : taken → BHR 00001
Br 2: not-taken → BHR 00010
Br 3: taken → BHR 00101

Two-level Predictor Classification

- Yeh and Patt 3-letter naming scheme
  - Type of history collected
    - **G** (global), **P** (per branch), **S** (per set)
  - PHT type
    - **A** (adaptive), **S** (static)
  - PHT organization
    - **g** (global), **p** (per branch), **s** (per set)

Some Two-level Predictors



Global vs. Local Branch History

- Local Behavior
  - What is the predicted direction of Branch A given the outcomes of previous instances of Branch A?
- Global Behavior
  - What is the predicted direction of Branch Z given the outcomes of *all** previous branches A, B, …, X and Y?

* number of previous branches tracked limited by the history length

Why does Global Predictor Work?

- Branches are correlated

Branch X: if (cond1)

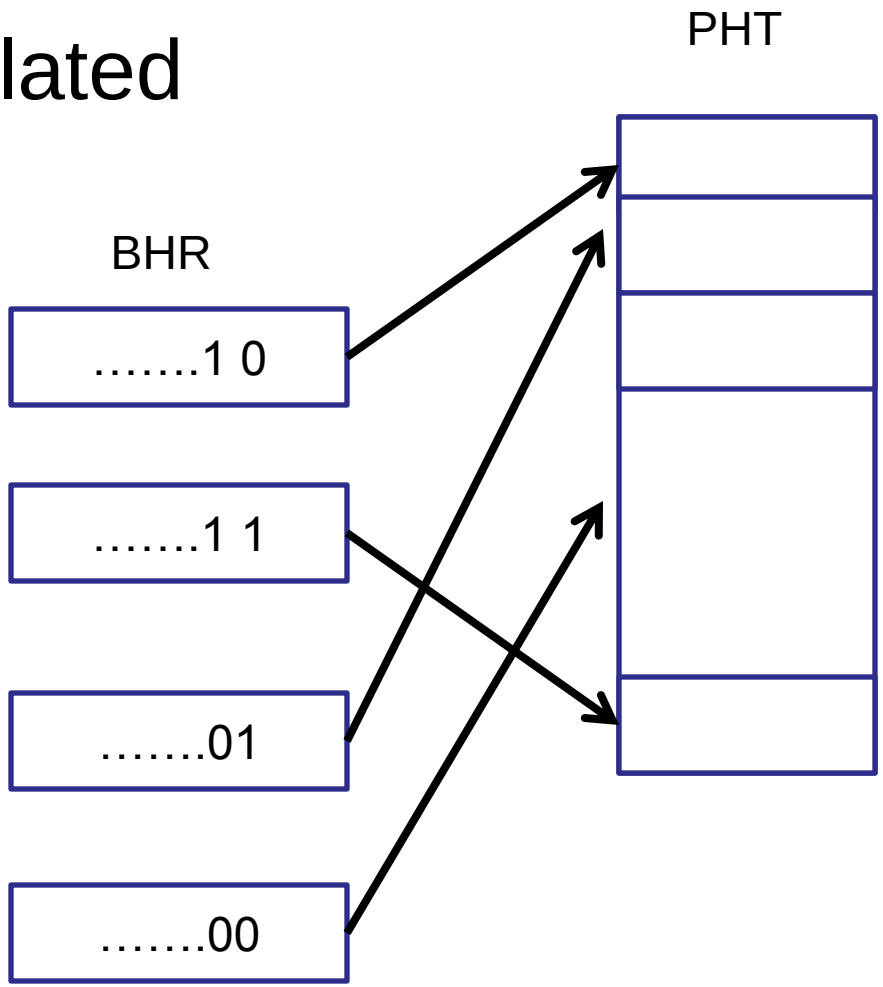
….

Branch Y: if (cond 2)

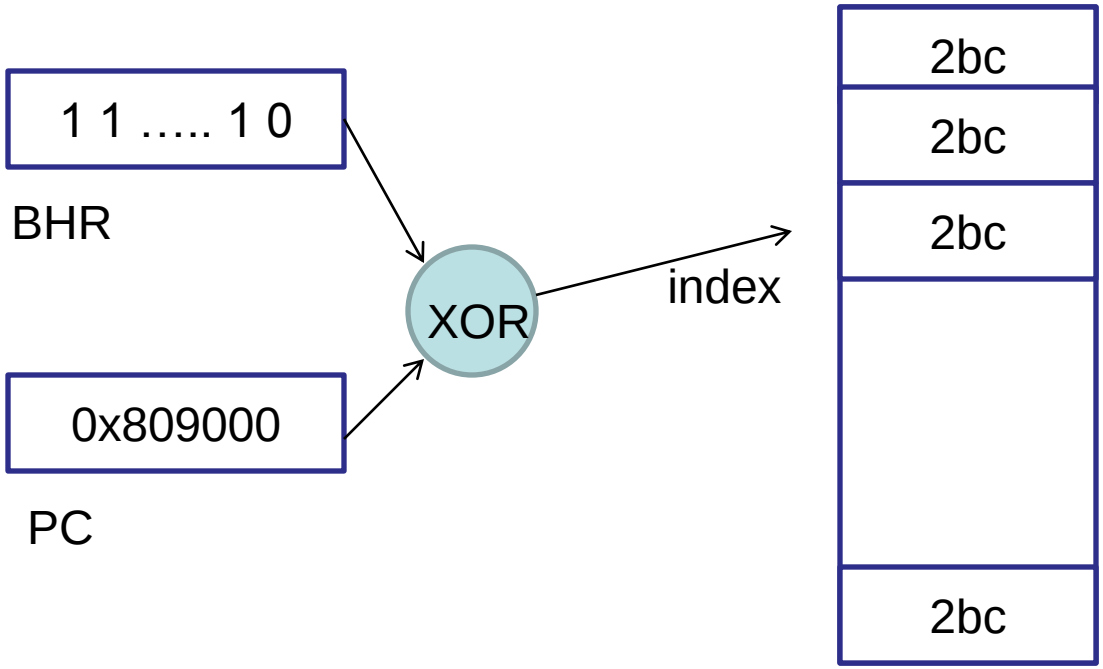
….

Branch Z : if (cond 1 and cond 2)

Branch X	Branch Y	Branch Z
1	0	0
1	1	1
0	1	0
0	0	0



Gshare Branch Predictor



McFarling'93

Predictor size: 2^(history length)*2bit



G-SHARE Algorithms

```
predict_func(pc, actual_dir)
{
  index = PC xor BHR
   taken = 2bit_counters[index] > 2 ? 1 : 0
   correctly_predictied = (actual_dir == taken) ? 1 : 0  // stats
}

updated_func(pc, actual_dir)
{
  index = PC xor BHR
  if (actual_dir) SAT_INC( 2bit_counter[index] )
  else SAT_DEC ( 2bit_counter[index] )
  BHR = BHR << 1 | actual_dir
}
```