Lecture - 4

Dynamic Branch Prediction and High Performance Instruction Delivery

Dr. Soner Onder
CS 4431
Michigan Technological University

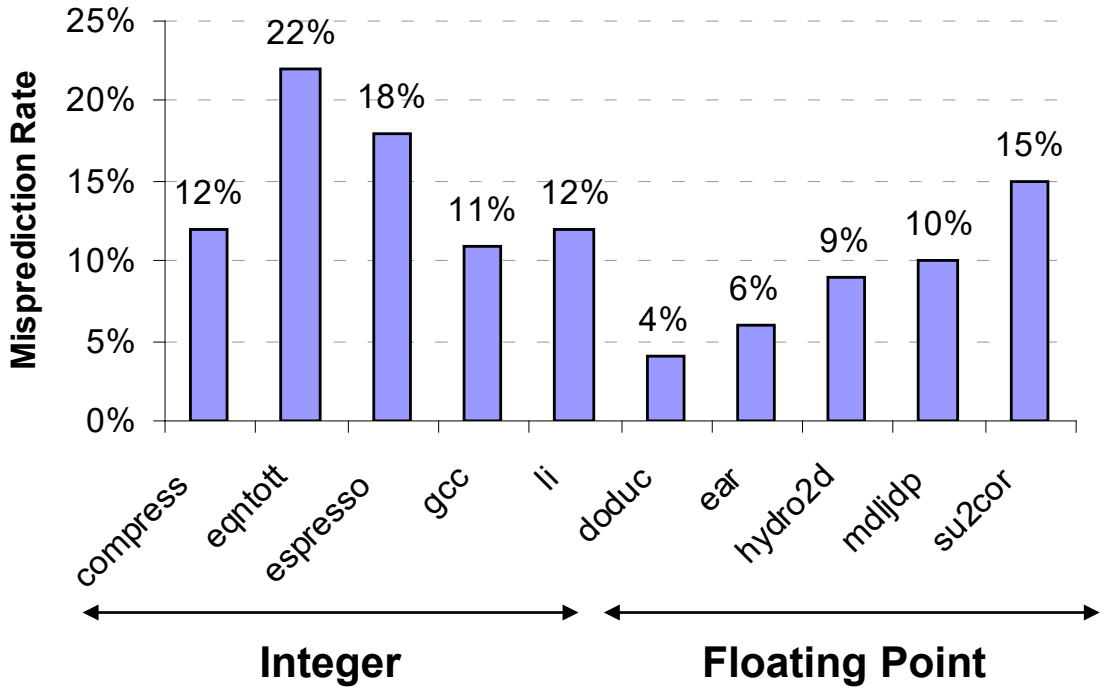
Dynamic Branch Prediction

- Why does prediction work?
  - Underlying algorithm has regularities
  - Data that is being operated on has regularities
  - Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems
  - There are a small number of important branches in programs which have dynamic behavior

Static Branch Prediction

- To reorder code around branches, need to predict branch statically when compiling.
- Simplest scheme is to predict a branch as taken
  - Average misprediction = untaken branch frequency = 34% SPEC

• More accurate scheme predicts branches using profile information collected from earlier runs, and modify prediction based on last run:

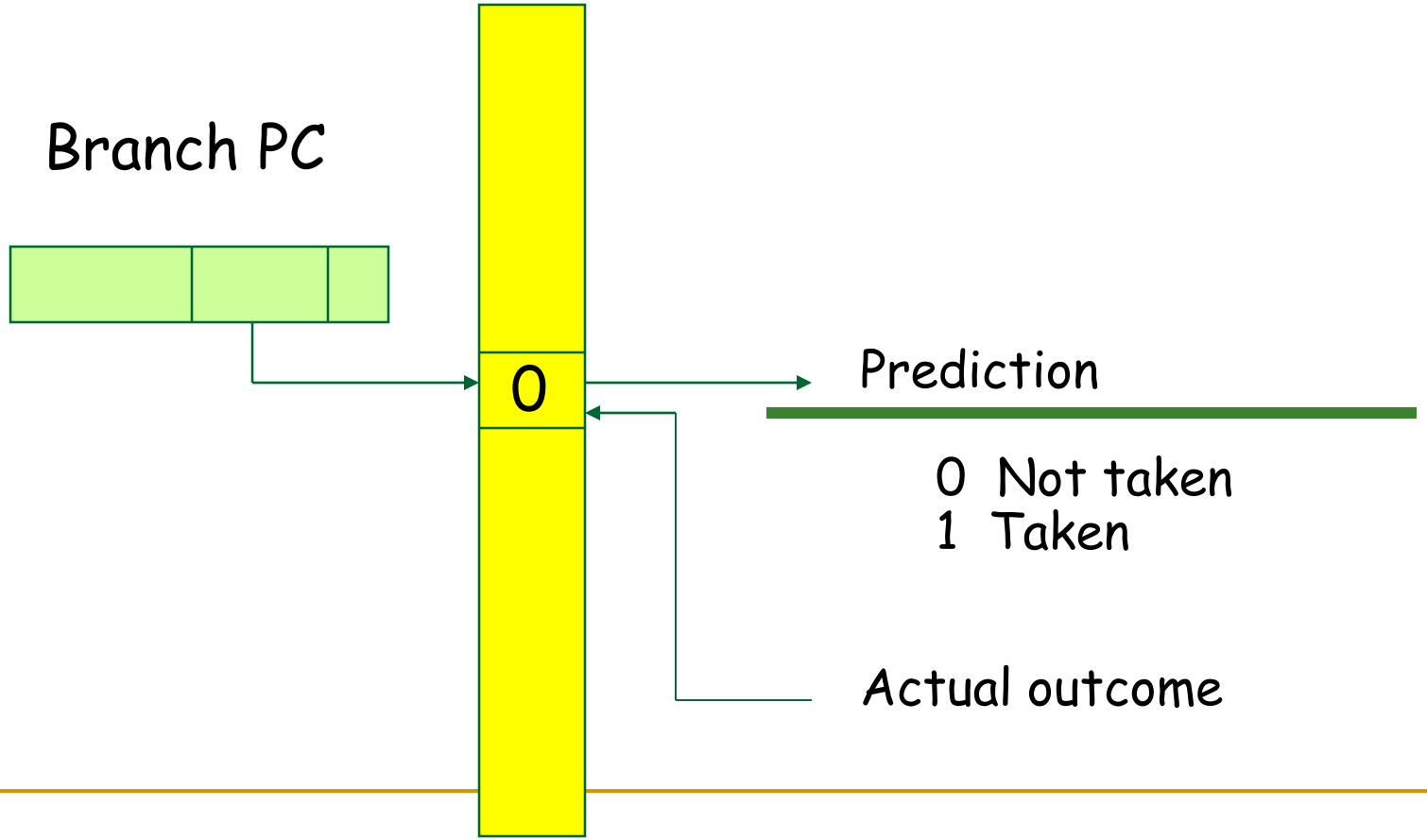


Dynamic Branch Prediction

- Performance = f(accuracy, cost of misprediction)
- Branch History Table: Lower bits of PC address index table of 1-bit values
  - Says whether or not branch taken last time
  - No address check

A Simple Branch Predictor

- Accessed early in the pipeline using the branch instruction PC
- Updated using the actual outcome.



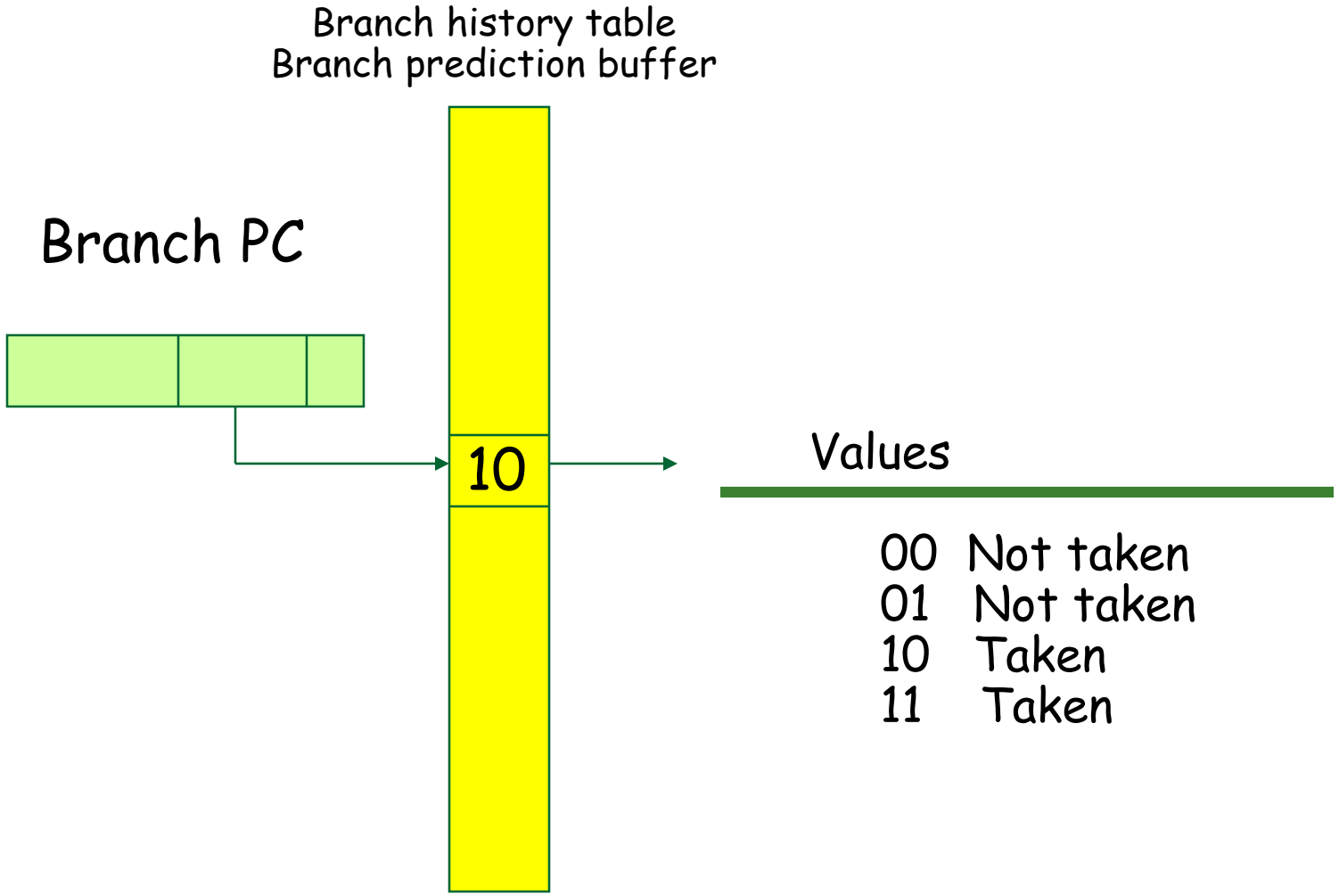
A Simple Branch Predictor

- What happens when we see the sequence of branches :
  - T N T N T N T N T N
  - (T T T T T T T T T N) *
  - N N N T N N N T
- What is the branch misprediction rate for each of the cases assuming the predictor is initialized to zero?

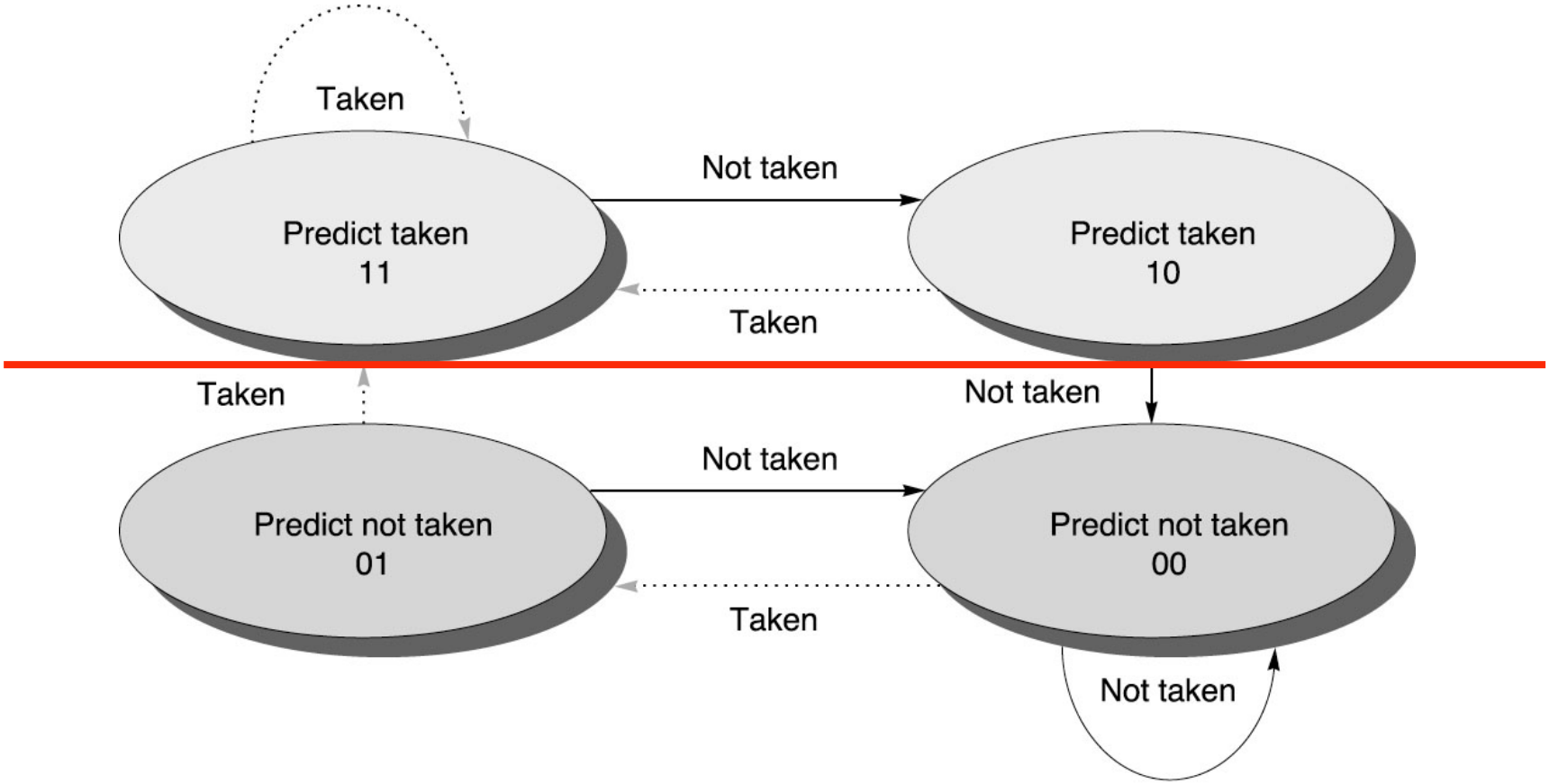
Dynamic Branch Prediction

- Problem: in a loop, 1-bit BHT will cause two mispredictions (avg is 9 iterations before exit):
  - End of loop case, when it exits instead of looping as before
  - First time through loop on next time through code, when it predicts exit instead of looping

Two bit branch prediction

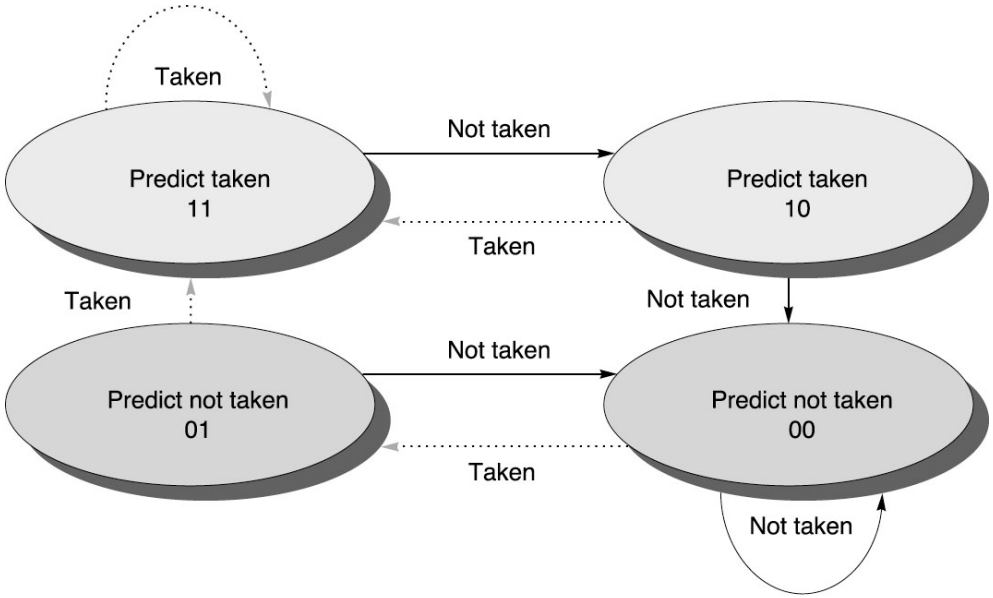


Two-bit predictor state diagram



Two-bit predictor

- A branch must miss twice before the prediction is changed.
- It is a specialization of an n-bit saturating counter scheme.



Next state

State	N-T	T
00	00	01
01	00	11
10	00	11
11	10	11

N-Bit saturating counter

- 0 - 2n-1 possible values:
  - 0000
  - 0001
  - 0010
  - 0011
  - 0100
  - 0101
  - 0110
  - 0111

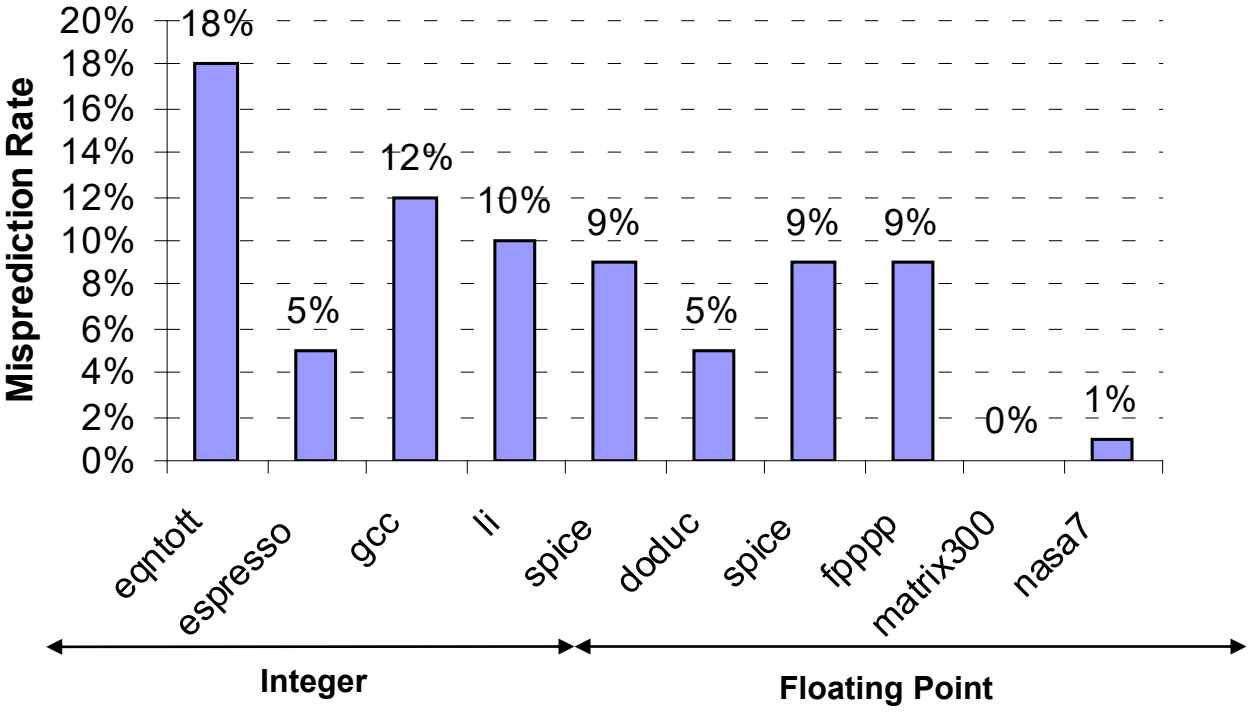
  PREDICT Not Taken
  - 1000
  - 1001
  - 1010
  - 1011
  - 1100
  - 1101
  - 1110
  - 1111

  PREDICT Taken

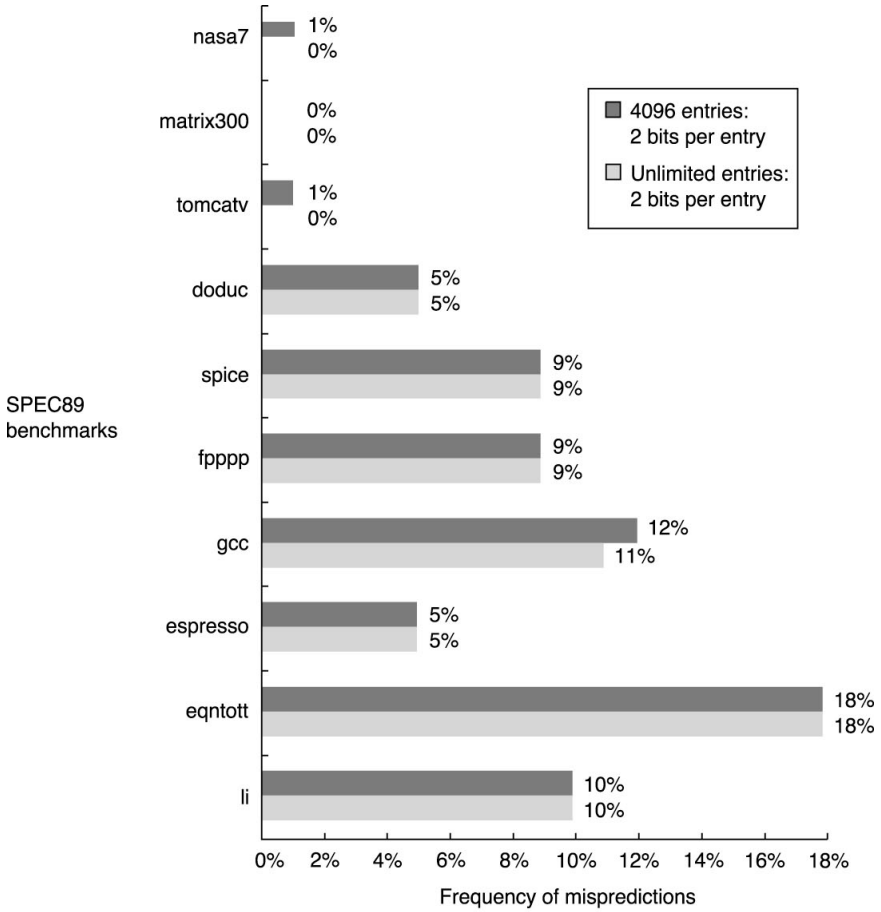
- Increment upon taken
- Decrement upon not taken

BHT Accuracy

- Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch when indexing the table
- 4096 entry table:



Spec89 Prediction Accuracy, infinite buffer



Correlating (Two-Level) Branch Predictors

- Consider the sequence:
  - if (aa == 2) aa=0;
  - if (bb == 2) bb=0;
  - if (aa != bb) { . . . .}

What can you say about the behavior of the last branch with respect to the prior two branches?

- MIPS Assembly:

```
aa is in $1, bb is in $2
subi $3,$1,#2
bnez $3,L1      ; aa!=2
add $1,$0,$0
```

- L1:

```
subi $3,$2,#2
bnez $3,L2      ; bb != 2
add $2,$0,$0
```

- L2:

```
subi $3,$1,$2
beqz  $3,L3     ; aa==bb
```

Correlating Branch Predictors

How can we capture the behavior of last n branches and adjust the prediction of the current branch accordingly?

Answer:

Use an n bit shift register, and shift the behavior of each branch to this register as they become known.

How many possible values will our shift register have?

Imagine there are many tables and select the table you want to use based on the value of the shift register.

Correlated Branch Prediction

- Idea: record m most recently executed branches as taken or not taken, and use that pattern to select the proper n-bit branch history table
- In general, (m,n) predictor means record last m branches to select between 2m history tables, each with n-bit counters
  - Thus, old 2-bit BHT is a (0,2) predictor
- Global Branch History: m-bit shift register keeping T/NT status of last m branches.
- Each entry in table has m n-bit predictors.

Correlating Branches

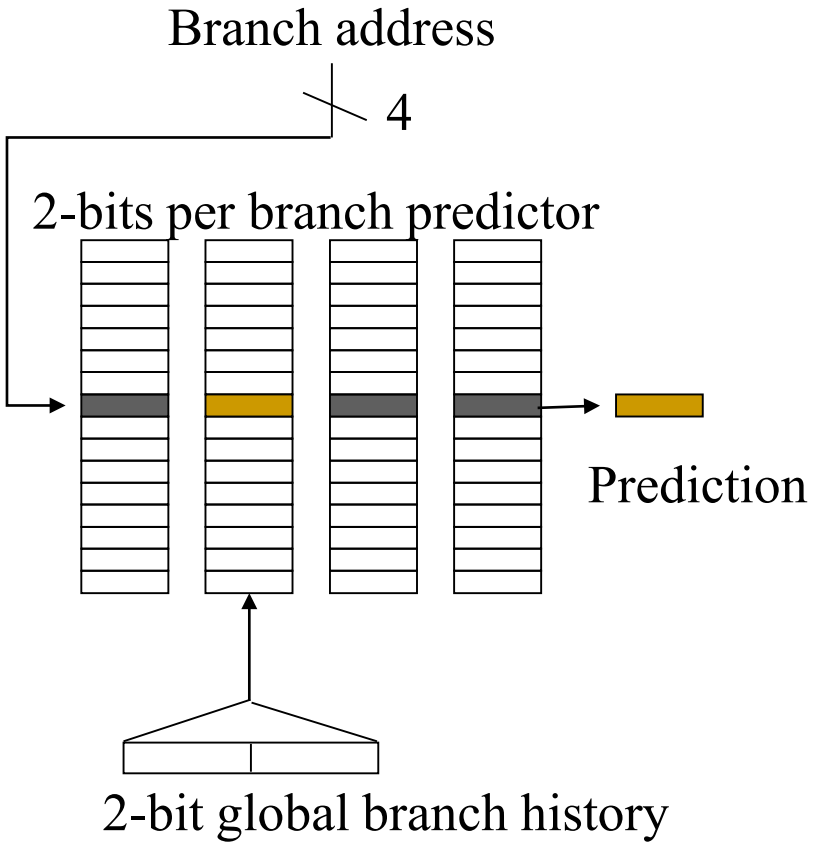
(2,2) predictor

–Behavior of recent branches selects between four predictions of next branch, updating just that prediction

2 Bits of global history means we look at T/NT behavior of last two branches to predict THIS branch.

The buffer can be implemented as a one dimensional array. How?

(m,n) predictor uses behavior of last m branches to choose from 2m predictors each being an n-bit predictor.



Correlating Branch Predictors

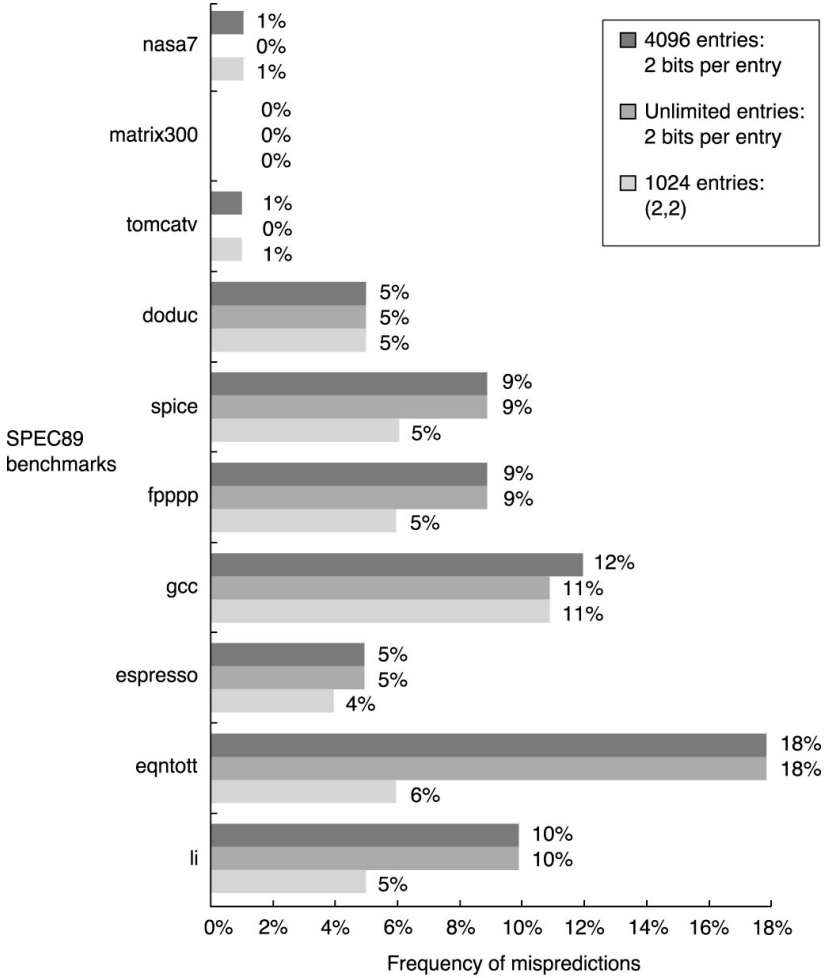
(m,n) predictor uses behavior of last m branches to choose from 2^m predictor each being an n-bit predictor. How many bits are there in a (0,2) branch predictor that has 4K entries selected by the branch address?

2^0x2x4K = 8K.

How many bits does the example predictor have?

2^2x2x16=128 bits

Correlating predictor performance

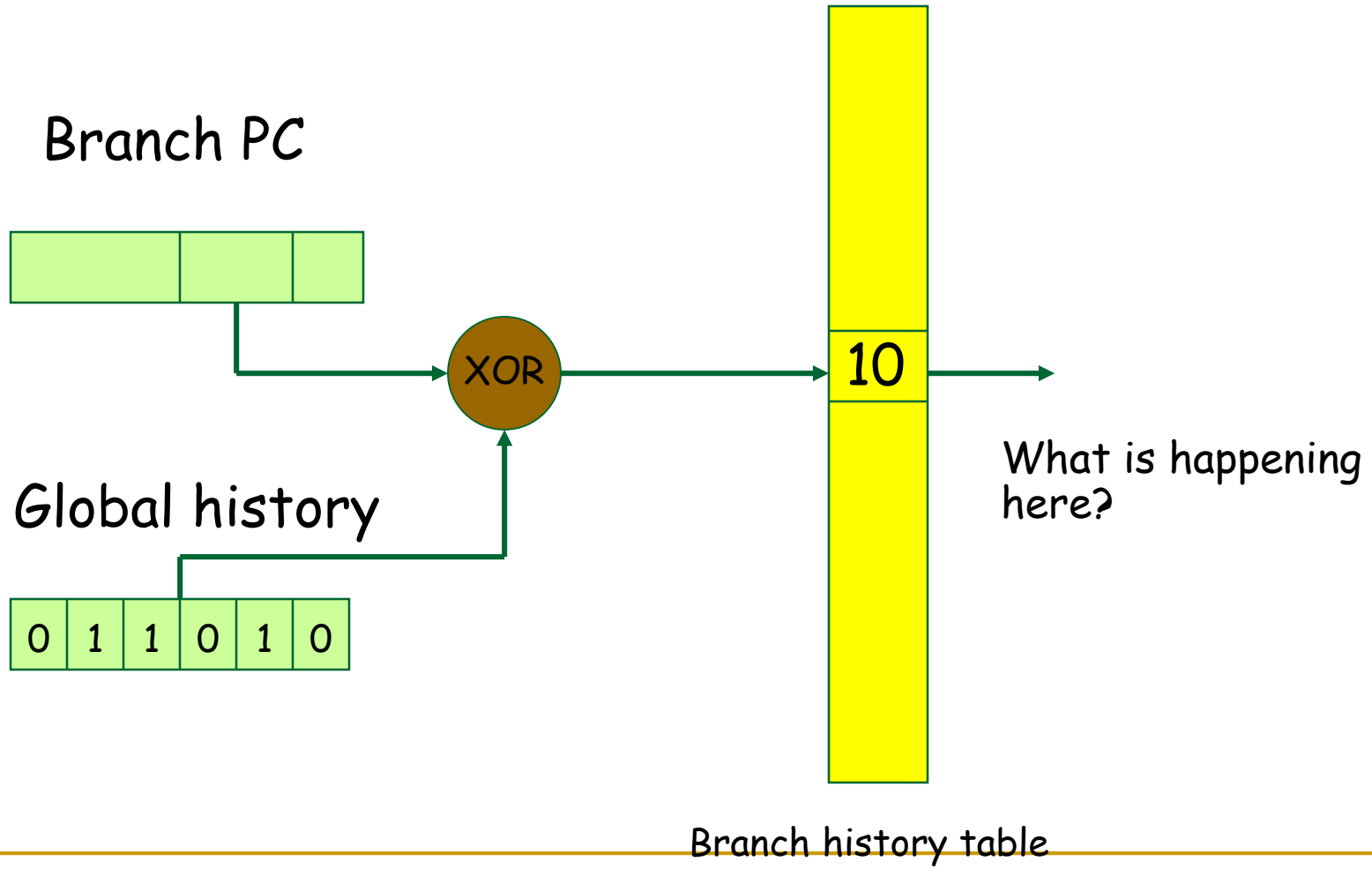


What can you say about the performance of a (2,2) branch predictor?

Why is this the case?

Which benchmarks show this behavior? Why?

Gshare Correlating predictor



Gshare Correlating predictor

Quick Homework:

Show the working of a gshare predictor that uses two bits of global history, and having 16 entries.

For this, enumerate the possible PC values (i.e., the portion of the PC that is used to index the table) and the global history values. Record the number of times each table entry is referenced. If you like, you may write a small C program for this purpose.

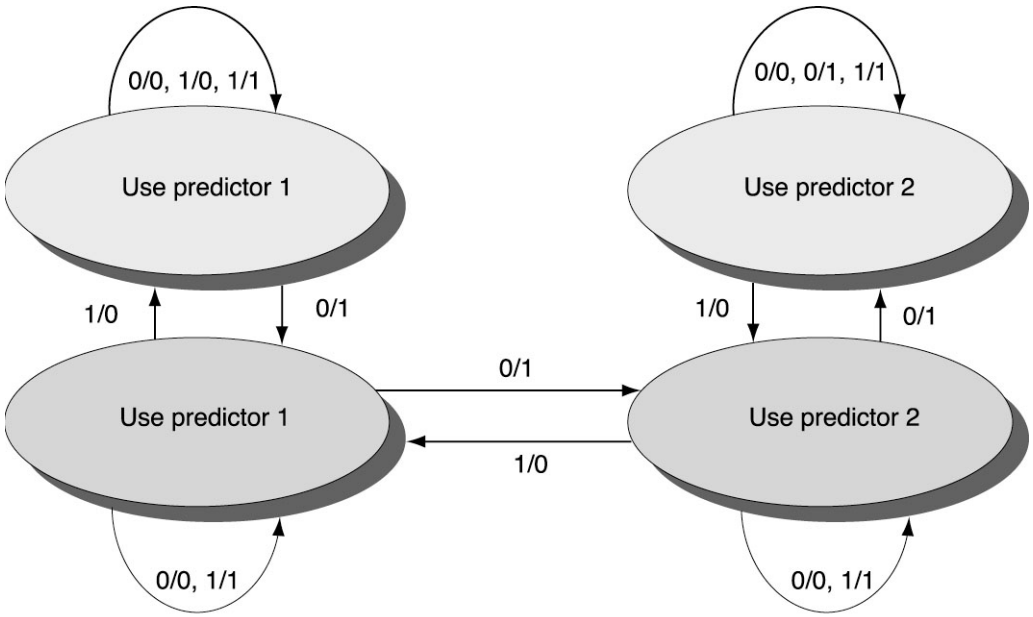
Are there collisions?

Is this harmful?

Hybrid predictors

- The basic idea is to use a META predictor to select among multiple predictors.
- Example:
- Local predictors are better in some branches.
- Global predictors are better in utilizing correlation.
- Use a predictor to select the better predictor.

Tournament Predictors



n/m means :
n left predictor
m right predictor

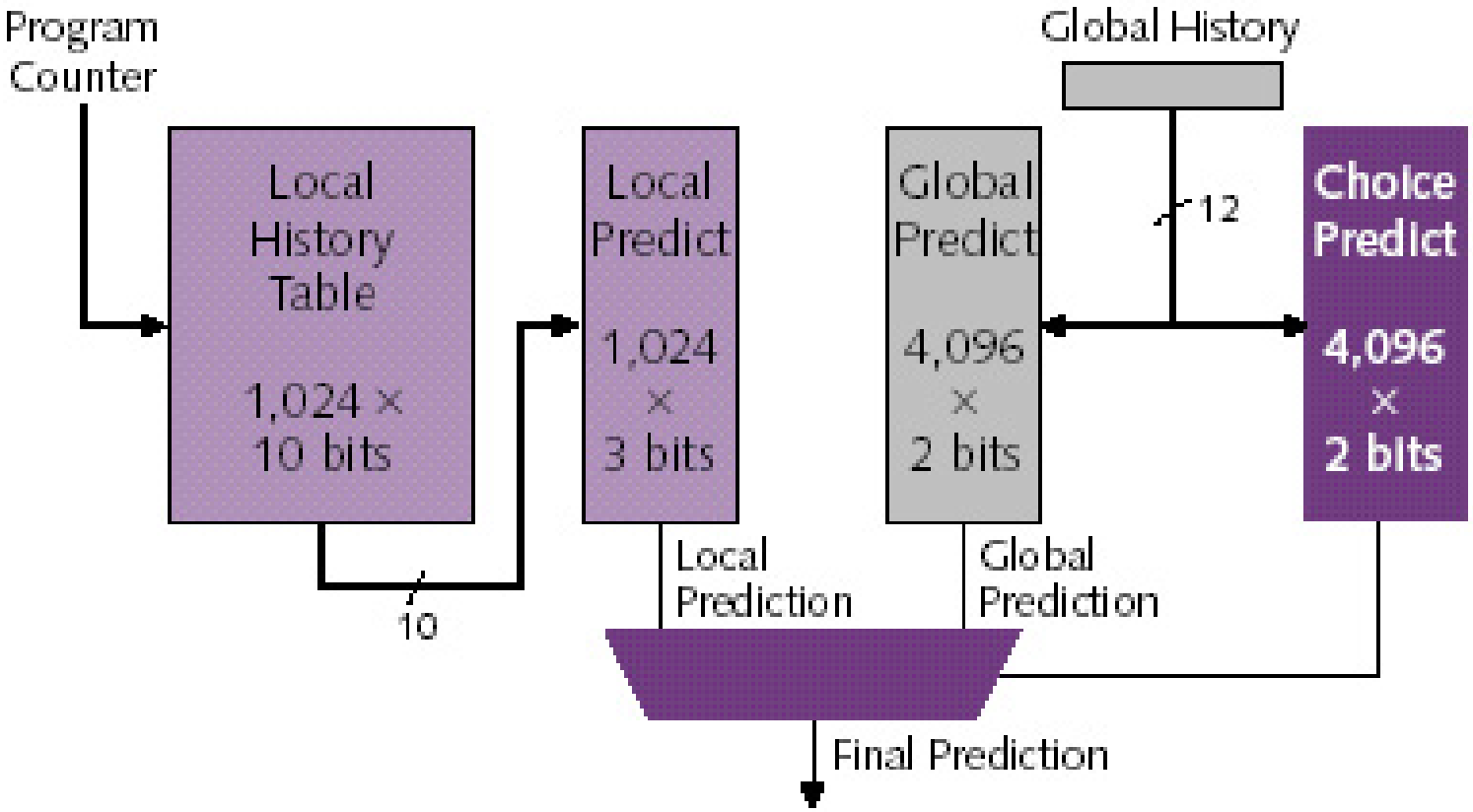
0: Incorrect
1: Correct

A predictor must be twice incorrect before we switch to the other one.

Tournament Predictors

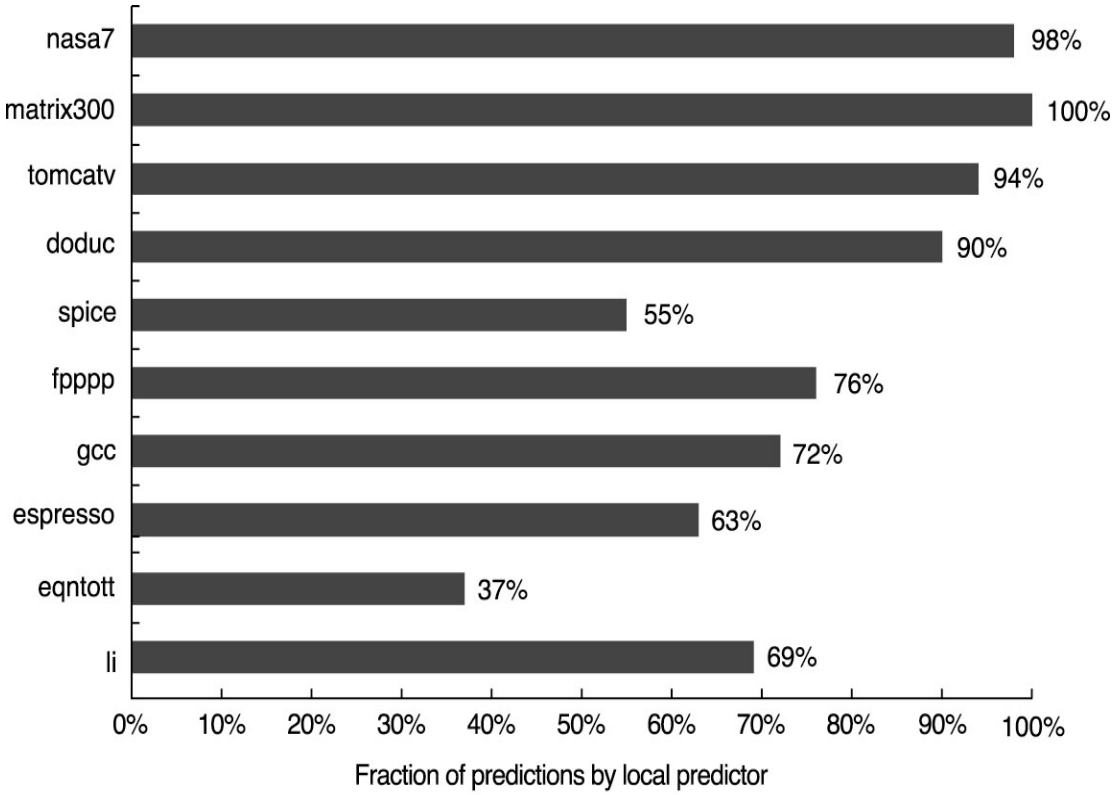
- Tournament predictor using, say, 4K 2-bit counters indexed by local branch address. Chooses between:
- Global predictor
  - 4K entries index by history of last 12 branches (212 = 4K)
  - Each entry is a standard 2-bit predictor
- Local predictor
  - Local history table: 1024 10-bit entries recording last 10 branches, index by branch address
  - The pattern of the last 10 occurrences of that particular branch used to index table of 1K entries with 3-bit saturating counters

Alpha 21264 Branch Prediction Mechanism



Source: Microprocessor Report, 10/28/96

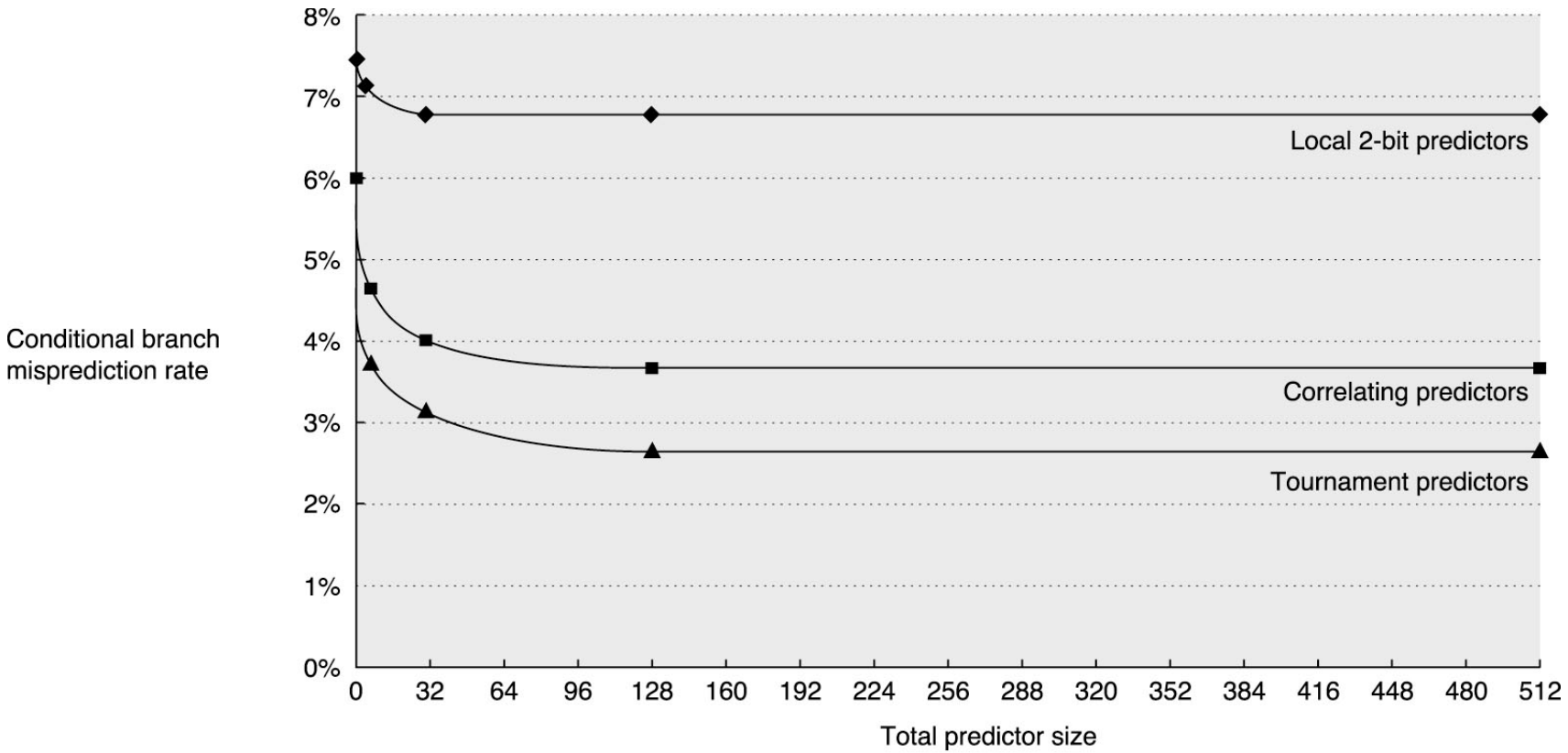
Fraction of predictions coming from the local predictor



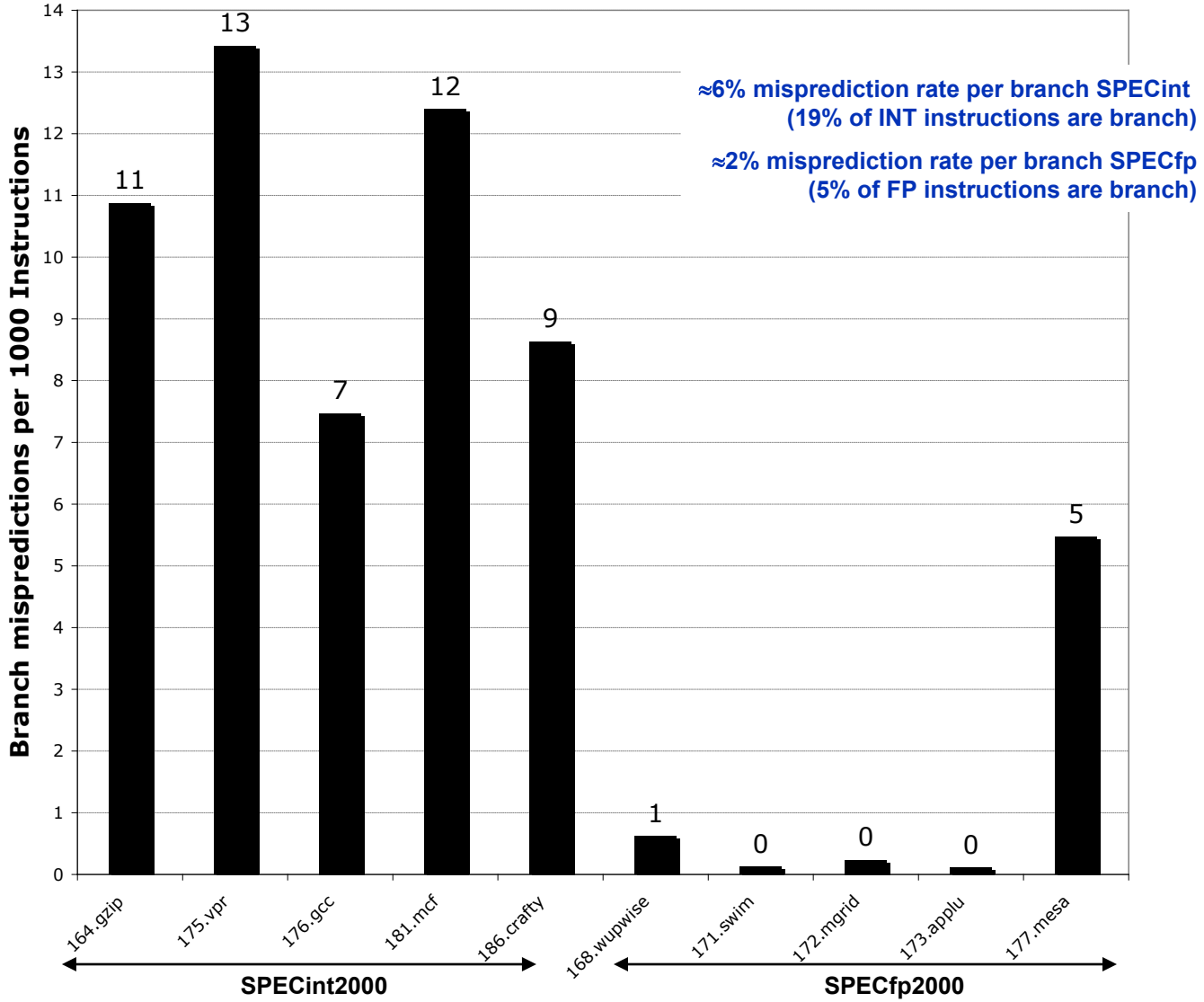
The Tournament predictor selects between a local 2-bit predictor and a 2-bit Gshare predictor.

Each predictor has 1024 entries each 2 bits for a total of 64 K bits.

Misprediction rates



Pentium 4 Misprediction Rate (per 1000 instructions)



Dynamic Branch Prediction Summary

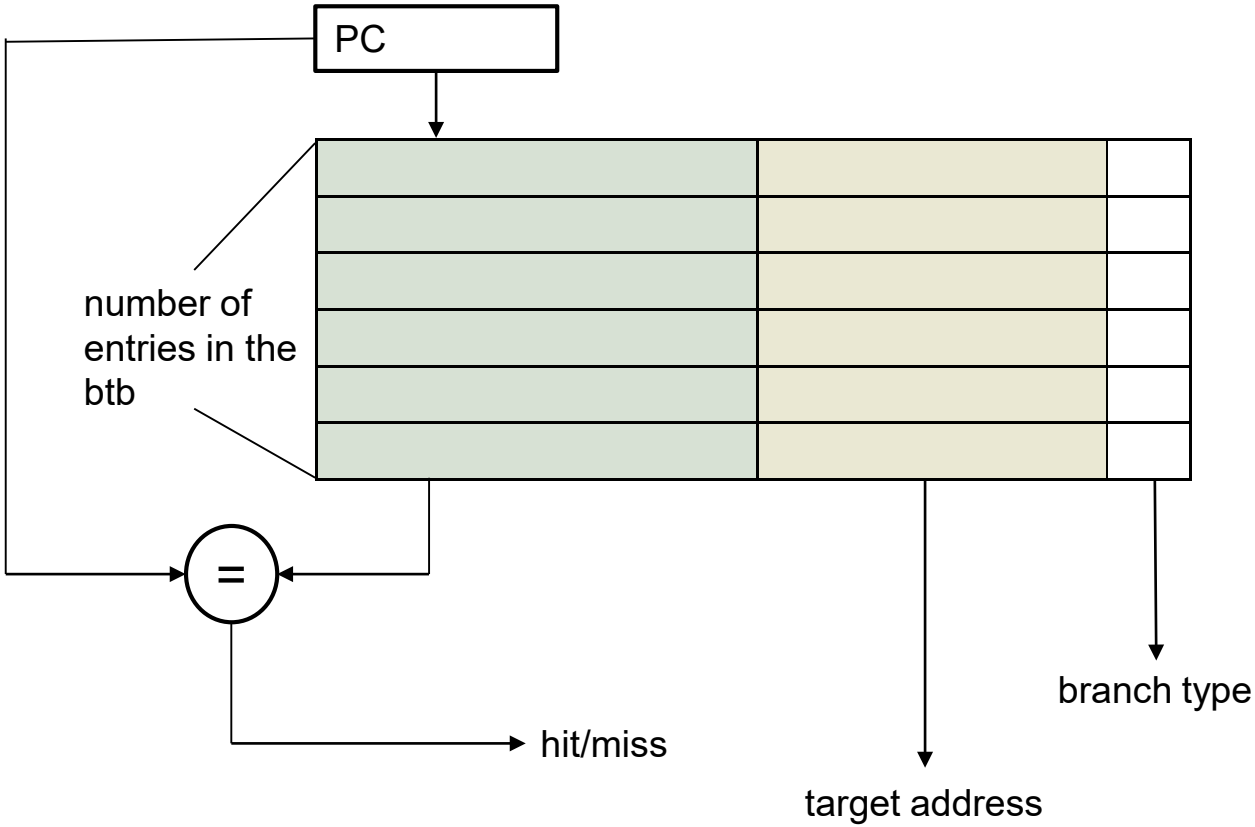
- Prediction becoming important part of execution
- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch
  - Either different branches (GA)
  - Or different executions of the same branch (PA)

- Tournament predictors take insight to next level, by using multiple predictors
  - usually one based on global information and one based on local information, and combining them with a selector
  - In 2006, tournament predictors using ≈ 30K bits are in processors like the Power5 and Pentium 4
- Branch Target Buffer: include branch address & prediction, discuss next

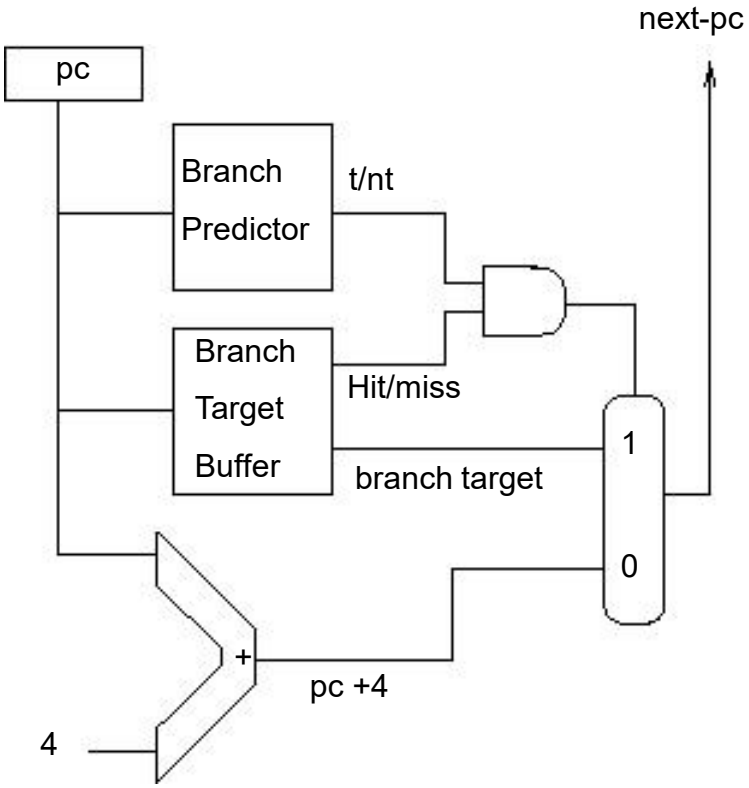
Branch Target Buffers

- We need to know the branch target address as well as the direction of the branch.
- We need to supply the branch target *before* decoding the current instruction!
- Don’t worry there is a simple way to achieve this. It is called a BTB.

Branch Target Buffers



Basic fetch unit using a branch predictor and a btb



If the BTB hits and the predictor predicts taken, then target address coming off the BTB is used as the next-pc. Otherwise, current pc+4 will become the next-pc.

RTL:

```
pc4 = pc + 4
taken = bp[pc]
target = btb[pc].target
hit = btb[pc].hit
next_pc = if hit & taken then target else pc4
```

Return Address Predictors

```
Procedure foo()
{
Important stuff
return;  {It really is jr $31}
}


for i=1; i < 100000; i++)
{
  foo();
}
```

What can you say about the prediction accuracy of BTB for the jr instructions?

Return Address Predictors

```
Procedure foo()
{
Important stuff
return;  {It really is jr $31}
}


for i=1; i < 50000; i++)
{
  foo();
  foo();
}
```

What can you say about the prediction accuracy of BTB for the jr instructions?

Return Address Predictors

Use a stack:

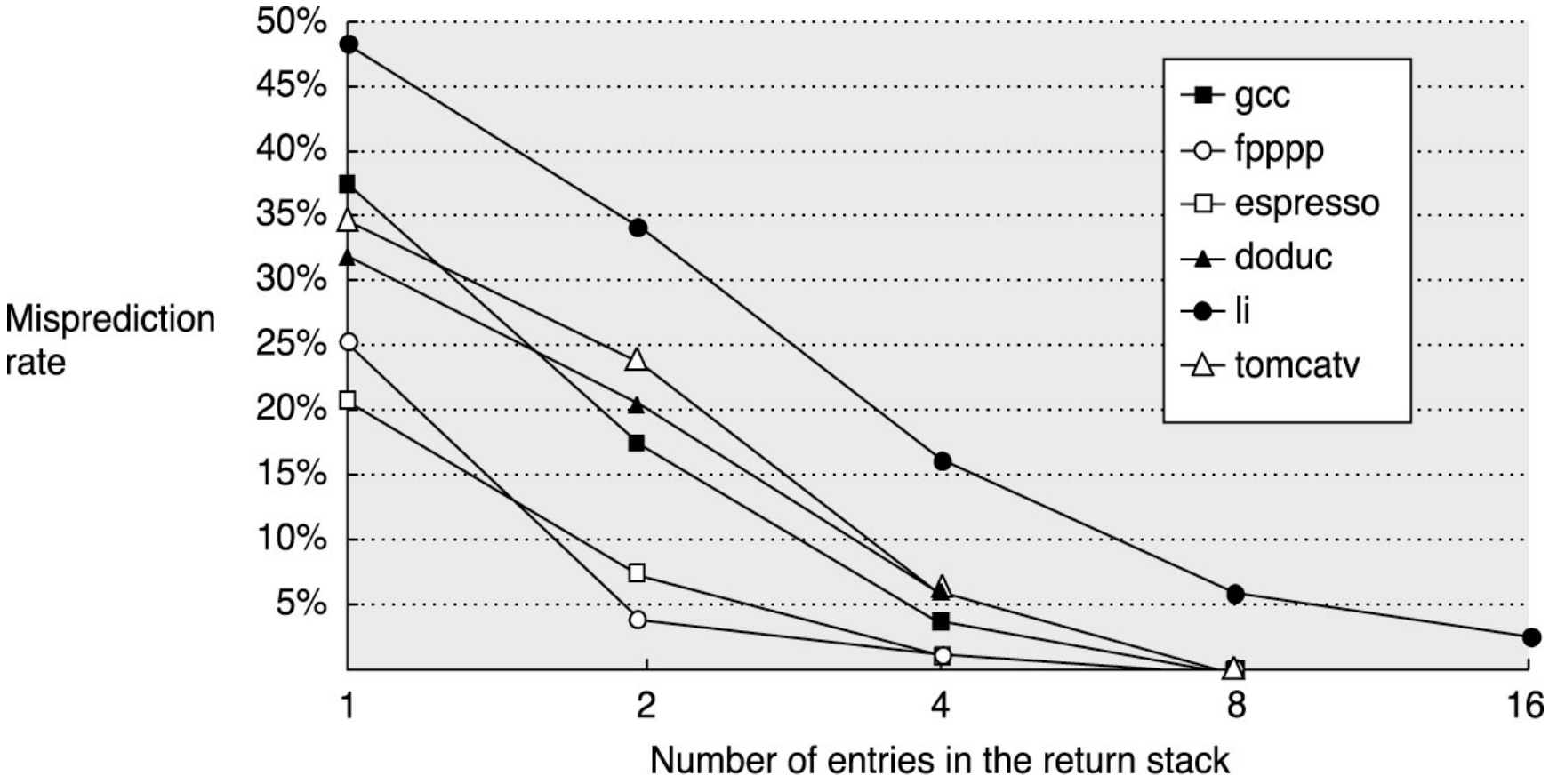
call (I.e. jal to a subroutine) push the return address onto the stack.
return (I.e. jr $31) pop the address from the stack.

Discard the bottom entry if overflow.

What can you say about the prediction accuracy of BTB for the jr instructions if we have an infinite stack depth?

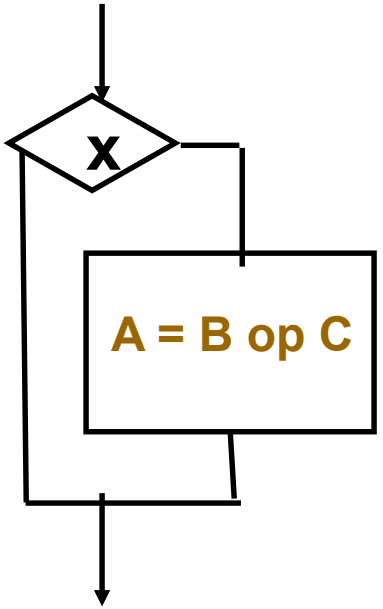
How about a limited stack depth?

Return Address Predictors



Instruction predication

- Avoid branch prediction by turning branches into conditionally executed instructions:
- if (x) then A = B op C else NOP
  - If false, then neither store result nor cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr.
  - IA-64: 64 1-bit condition fields selected so conditional execution of any instruction



Conditional Move Instructions

Example

```
if x < y then
   a=a + 1
else
   a=a * 2
```

Code Sequence

```
lw   r11,x
lw   r12,y
slt  r3,r11,r12

lw   r7,a
addi r8,1
sll  r9,r7,1
cmov r9,r8,r3
sw   r9,a
```

Full predication

Example

```
if x < y then
   a=a + 1
else
   a=a * 2
```

```
p = x < y;
p: a = a + 1;
!p: a = a * 2;
```

Code sequence:

```
lw   T: r11,x
lw   T: r12,y
slt  T: r3,r11,r12
lw   T : r7,a
addi r3: r8,r7,1
sll   r3: r8,r7,1
sw   T : r9,a
```

Instruction predication

- Drawbacks to conditional instructions
  - Still takes a clock even if “annulled”
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline

Dynamic Branch Prediction Summary

- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch
- Branch Target Buffer: include branch address & prediction
- Return address predictor: Works well for most procedure calls.
- Predicated Execution can reduce number of branches as well as number of mispredicted branches.

Spring in Calumet