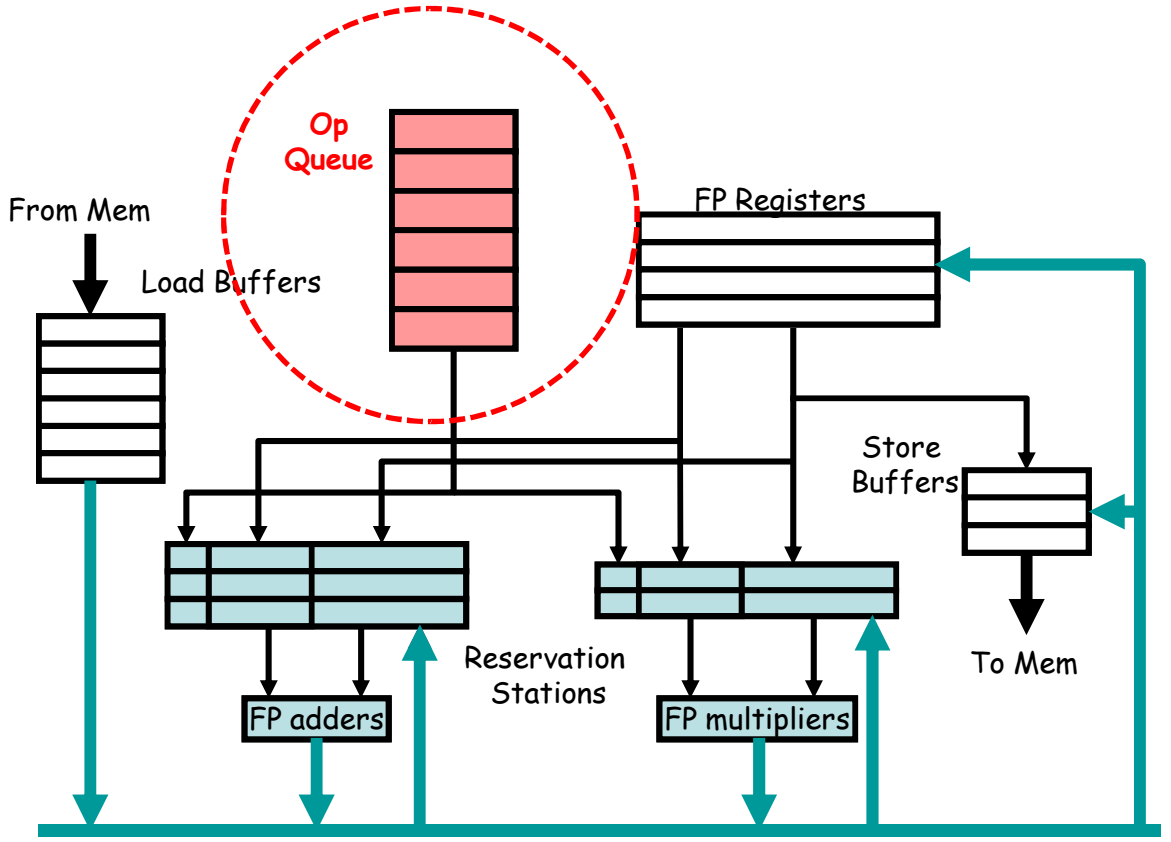
Branchs tend to increase the CPI!

- Need to sustain the incoming instruction flow
  - Fetch determines a lower bound to CPI
  - 1 for *scalar* processors
- How *essential* are control dependencies?



Basic techniques

- Code restructuring
  - e.g. loop unrolling to reduce the number of branches
- Architectural techniques:
  - Delayed branch
  - Static *predict untaken*

- as pipelines get **longer**, the branch penalty gets **higher**

Static prediction

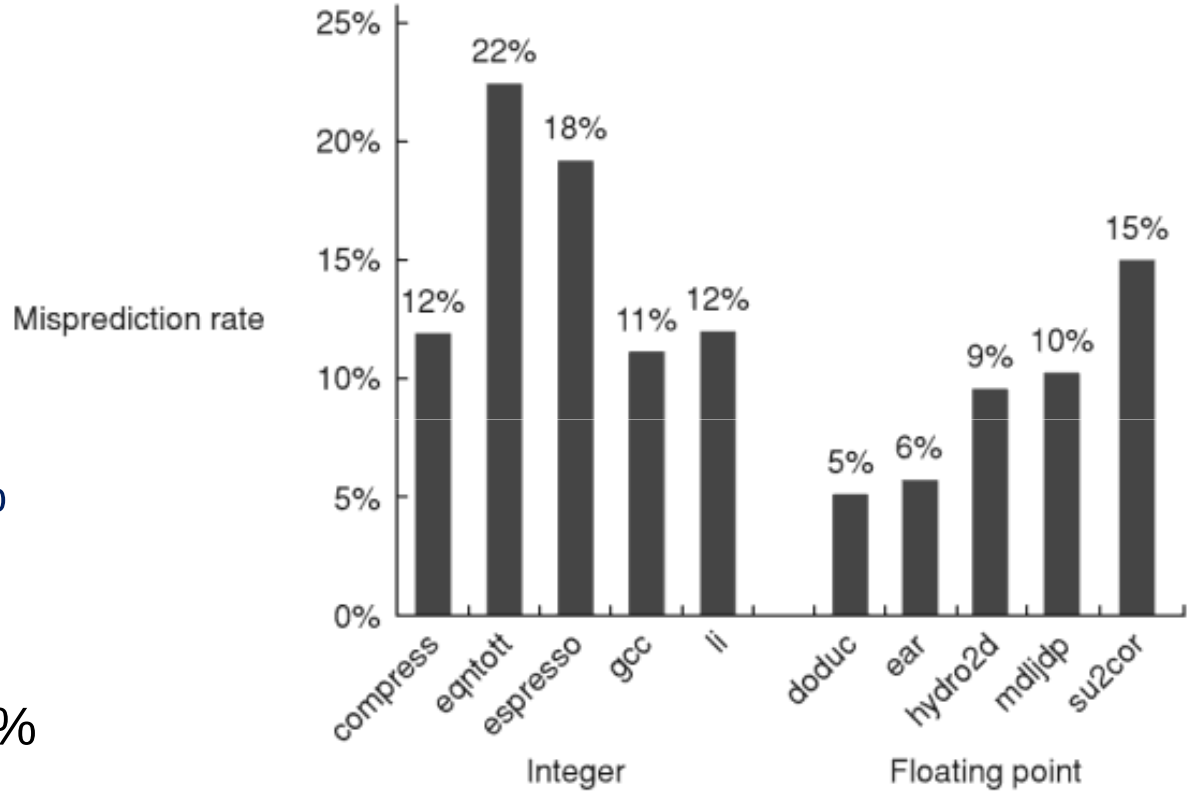
- How essential are control dependencies?
  - *i.e.* when can we hide them?

- *First approach*: profile the code and statically predict all branches
  - All branches are either always predicted taken or predicted untaken
  - E.g., SPEC92 benchmarks → untaken frequency is 34%

    …with a large variance (accuracy of predict taken ranges between 91% and 41% for the different programs in the suite)
- *Second approach*: support individual branch prediction
  - Profile single branches separately
  - processor support needed (e.g. a branch hint field in the instr.)

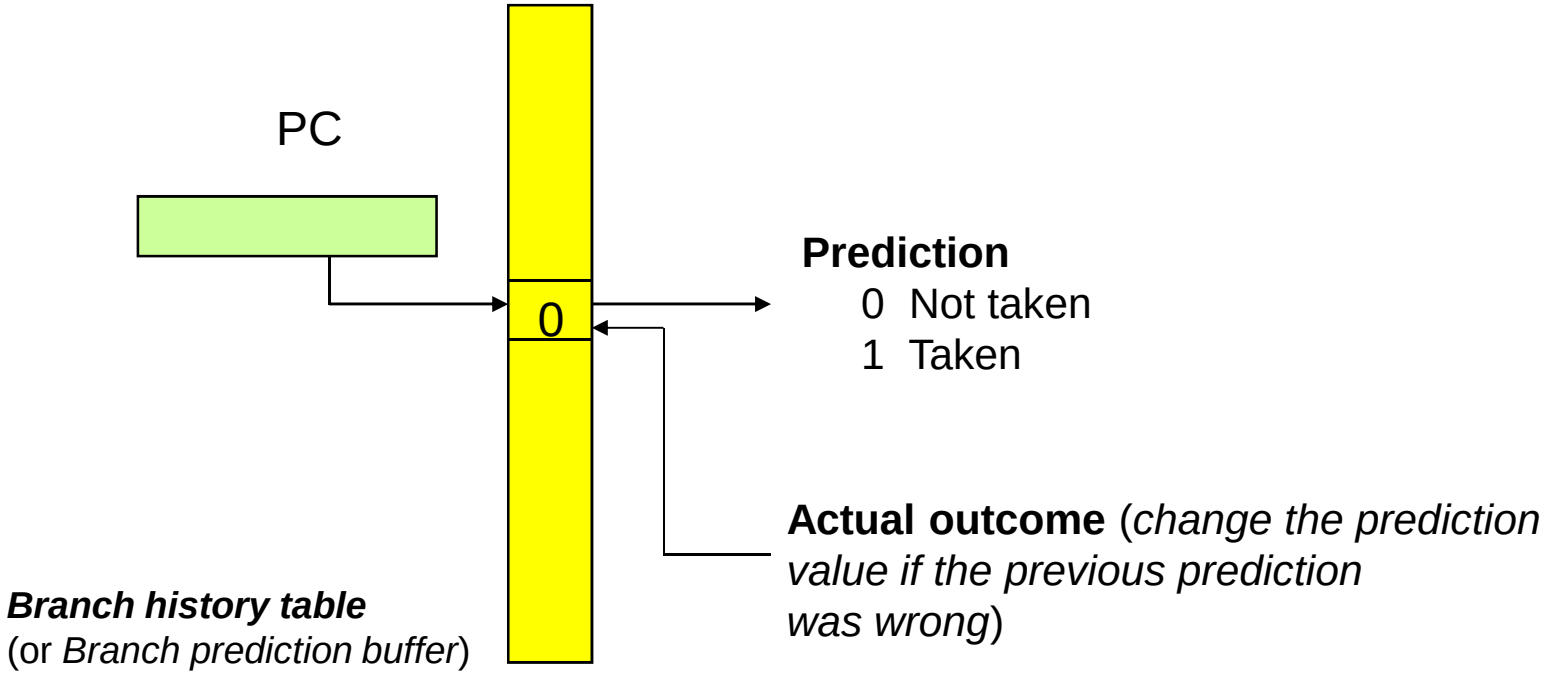
Profiling branches

- SPEC92
- Individual branch hint
- **FP** programs:
  - Average misprediction 9%
- **INT** programs:
  - Average misprediction 15%



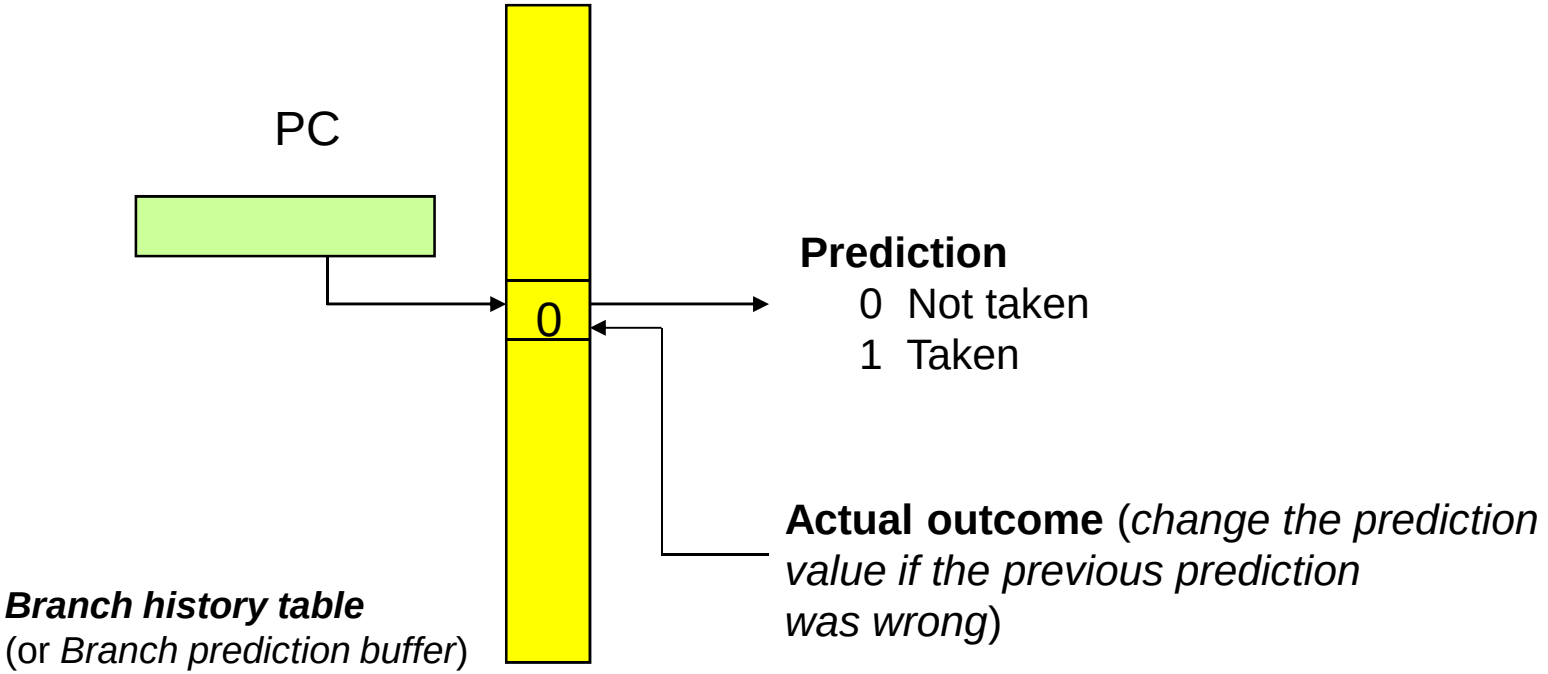
Dynamic prediction: one-bit predictor

- Associate a prediction bit to *each branch* in the program
- Each branch instrution has a single bit associated to it telling the processor whether to jump or not
- Prediciton *can be changed* during execution



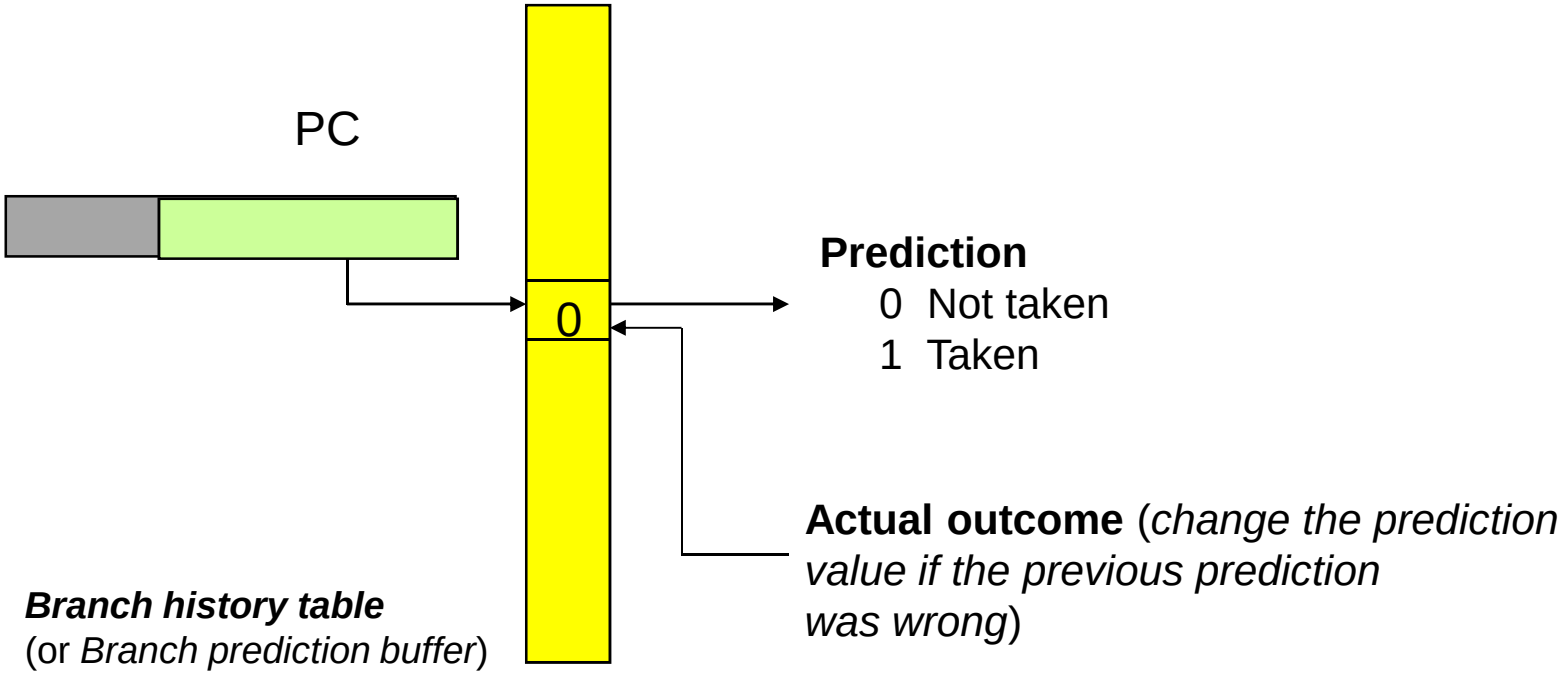
Dynamic prediction: one-bit predictor

- If prediction is correct …
  - Continue normal execution – no wasted cycles
- If prediction is incorrect (*misprediction*) …
  - Flush the instructions that were incorrectly fetched – wasted cycles
  - Update prediction bit and target address for future use



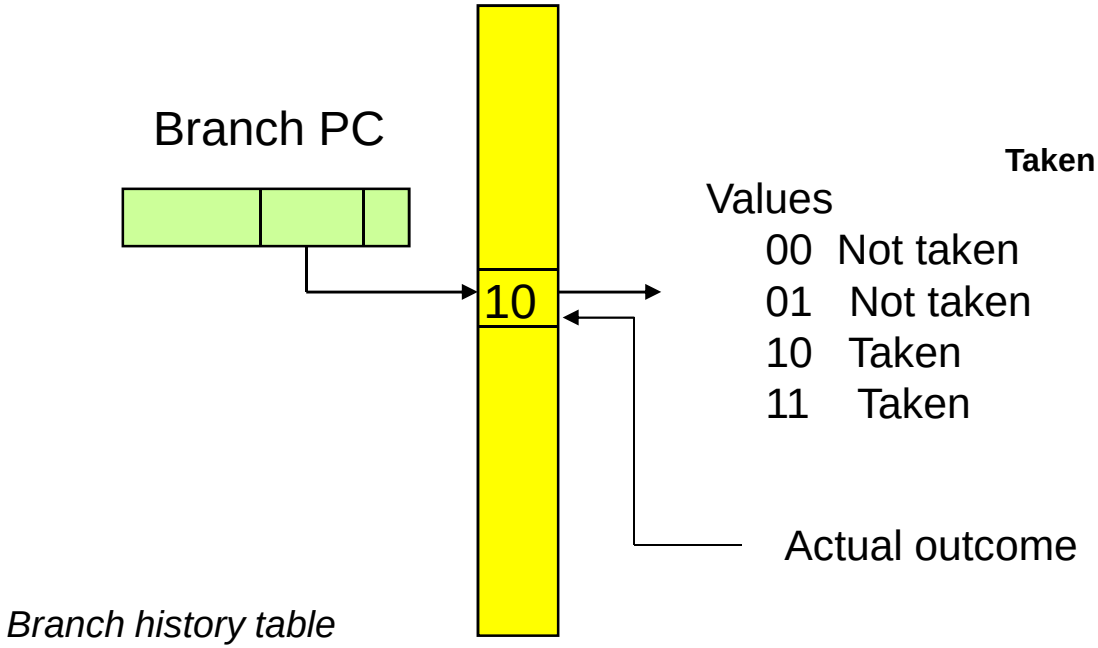
Dynamic prediction: one-bit predictor

- The branch prediction buffer acts as a cache
  - addressed by the lower part of the PC (e.g. the 12 l.s.bits)
- Limited entries → *aliasing* possible (same cell for two PCs)
  - the prediction is likely to be wrong, but incorrect predictions should be hanbled in any case, so no additional HW is needed
  - Make sure the table can contain a sufficient number of branches

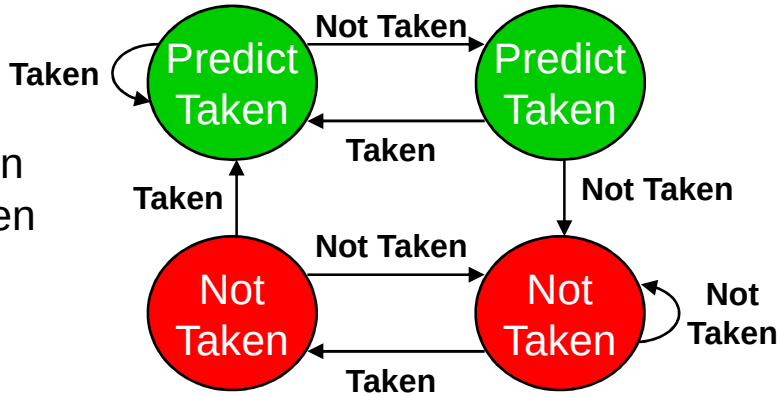


Dynamic prediction: 2-bit predictor

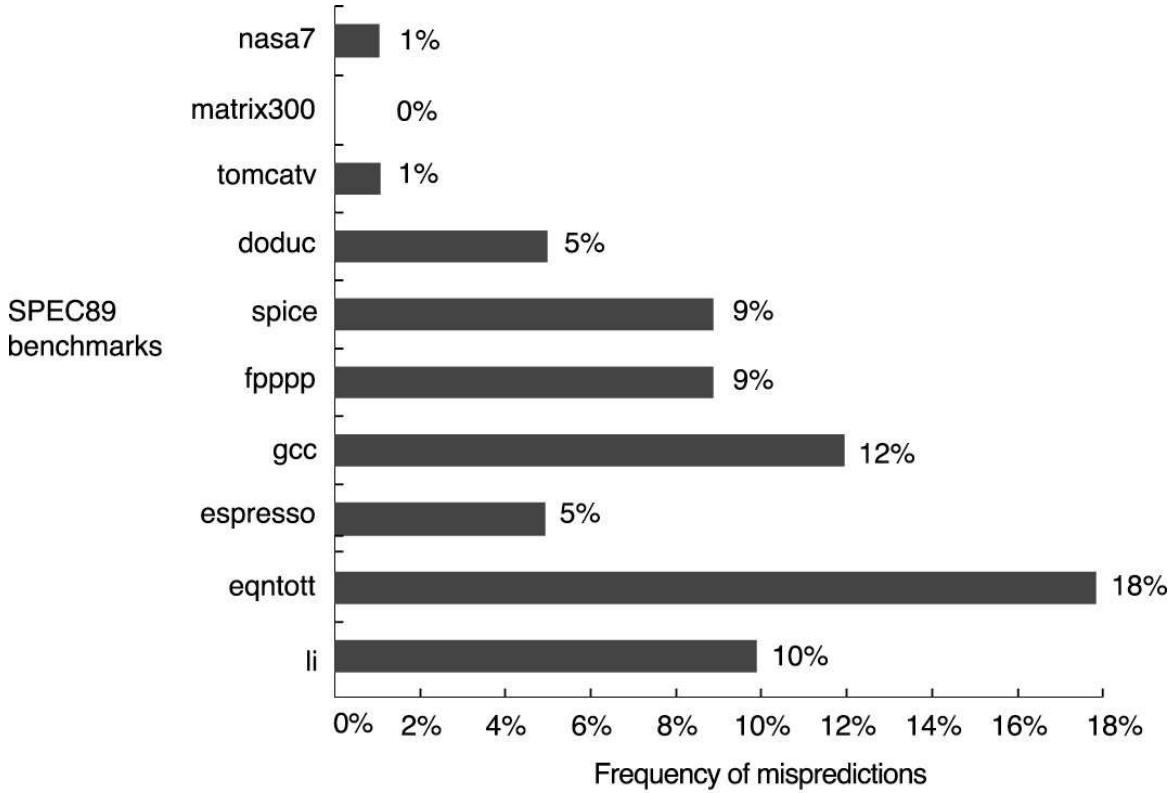
- Matches closer the structure of typical code. *Why*?
  - Think of nested cycles in the code!
  - The internal branch will be **untaken** once every ***n* taken**
  - Add "*inertia*" to the decision about changing the direction of the prediction: only change decision *after two* wrong predictions



Branch history table
(or *Branch prediction buffer*)

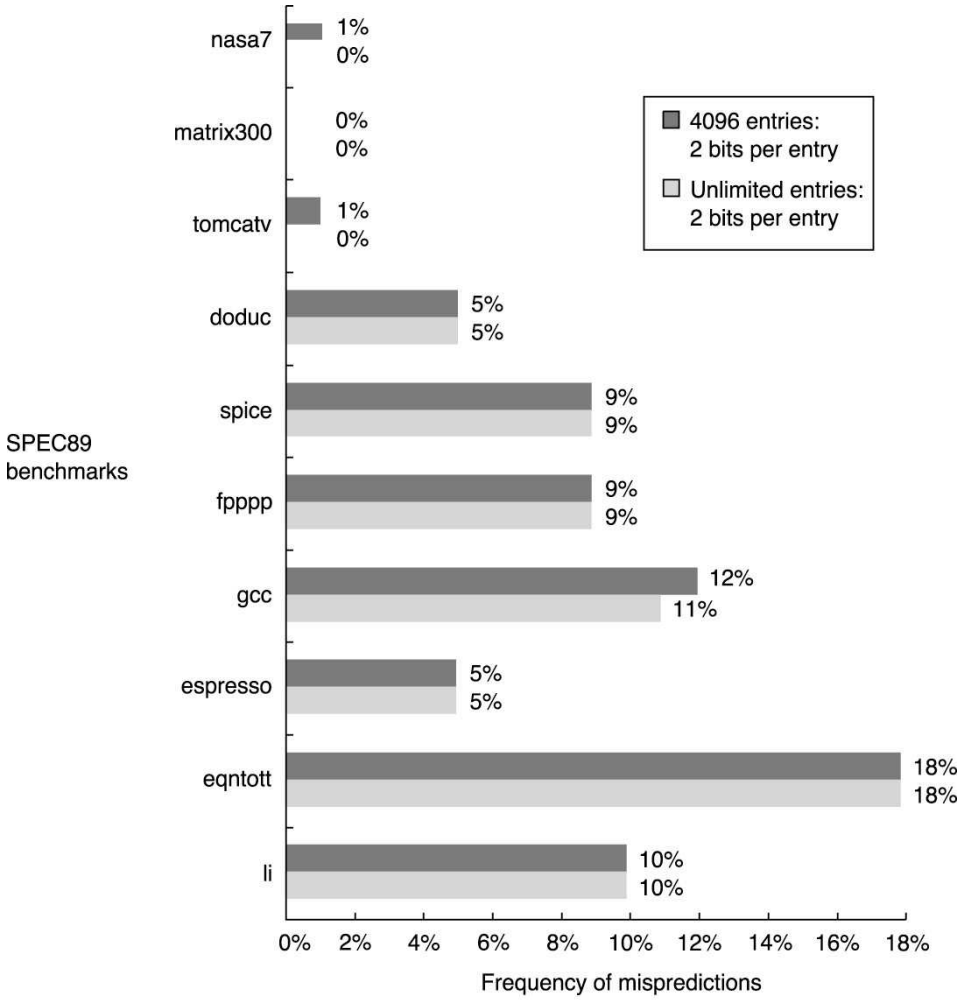


SPEC89: 4096 Entry 2-bit prediction buffer



SPEC89: Prediction Accuracy, infinite buffer

- How far can we get by increasing the buffer size?
- SPEC89 Benchmarks:
  - Aliasing doesn't affect the performance too much with a 4096-entry table



N-Bit saturating counter

- Generalize 2-bit predictors

0 - 2^n-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

Correlating 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 R1, bb is in R2

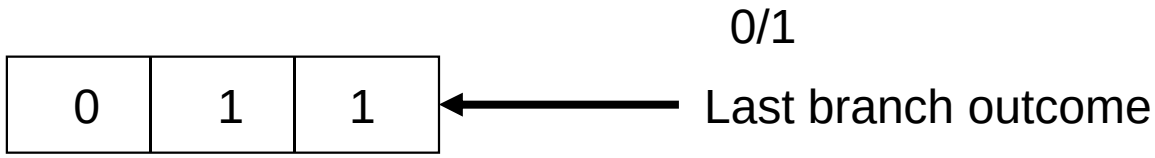
```
    DSUBI R3,R1,#2
    BNEZ R3,L1      ; aa!=2
    DADD R1,R0,R0
L1:DSUBI R3,R2,#2
    BNEZ R3,L2      ; bb != 2
    DADD R2,R0,R0
L2:DSUBI R3,R1,R2
    BEQZ  R3,L3     ; aa==bb
```

Correlating Branch Predictors

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

Answer:

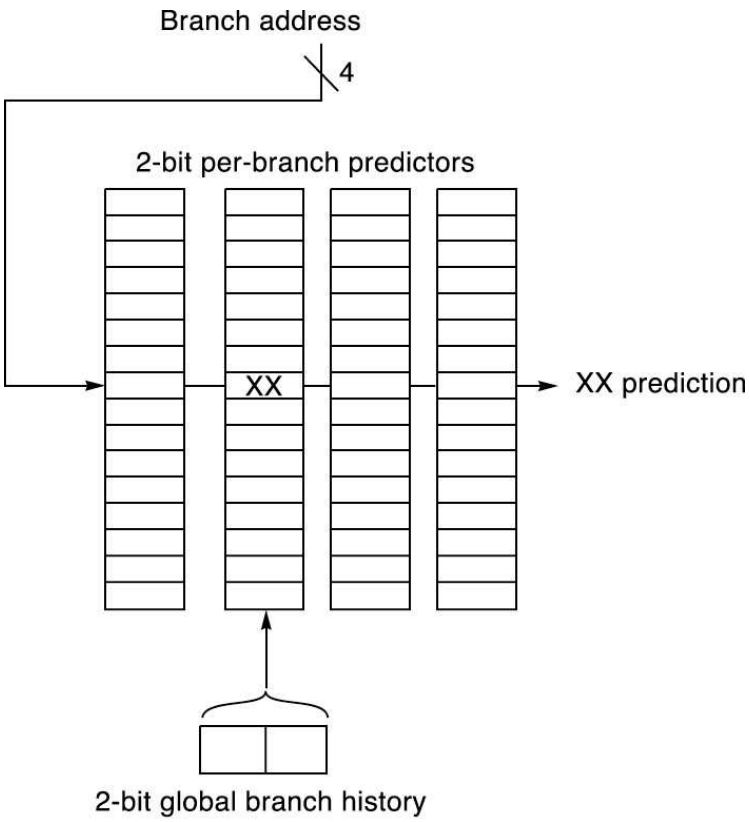
Use an *n* bit shift register, and shift the behavior of each branch to this register as they become known.



How many possible values will our shift register have?

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

Correlating Branch Predictors



2 Bits of ***global history*** means we look at T/NT behavior of last two branches to determine the behavior of 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 2^m predictor each being an *n*-bit predictor.

Correlating Branch Predictors

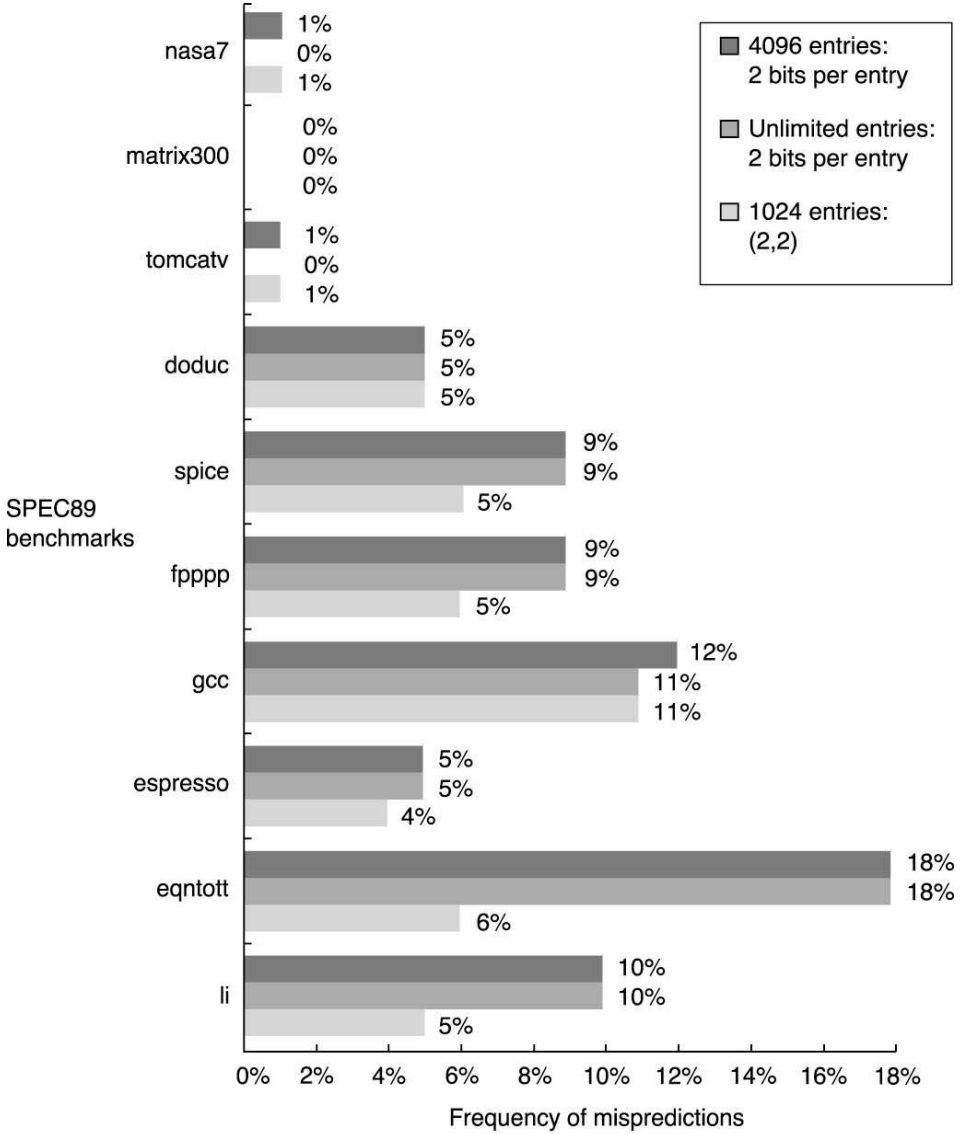
(m,n) predictor uses behavior of last *m* branches to choose from 2^m predictor each being an *n*-bit predictor.
How many bits are there in a *(0,2)* branch predictor that has **4K** entries selected by the branch address?

$2^0 \times 2 \times 4K = 8K$

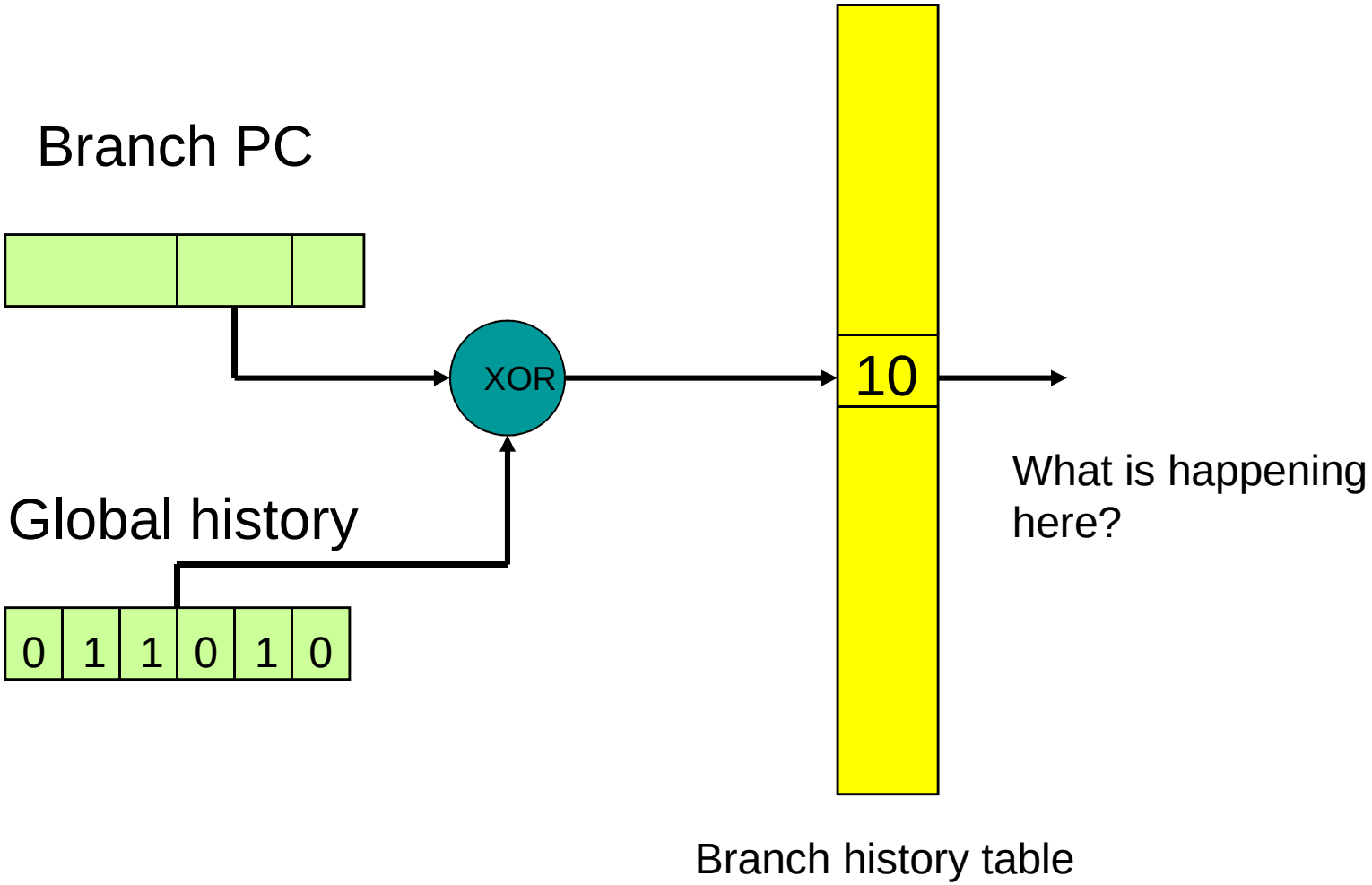
How many bits the example predictor has?

$2^2 \times 2 \times 16 = 128$ bits

Correlating predictors performance



Gshare Correlating predictor



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.

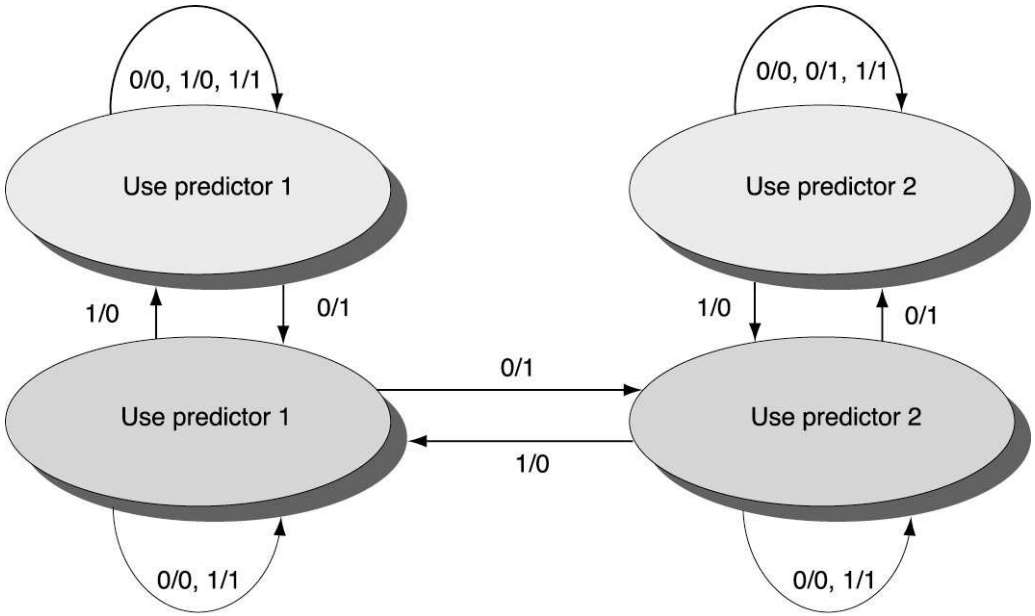
Use a predictor to select the better predictor.

Tournament Predictors

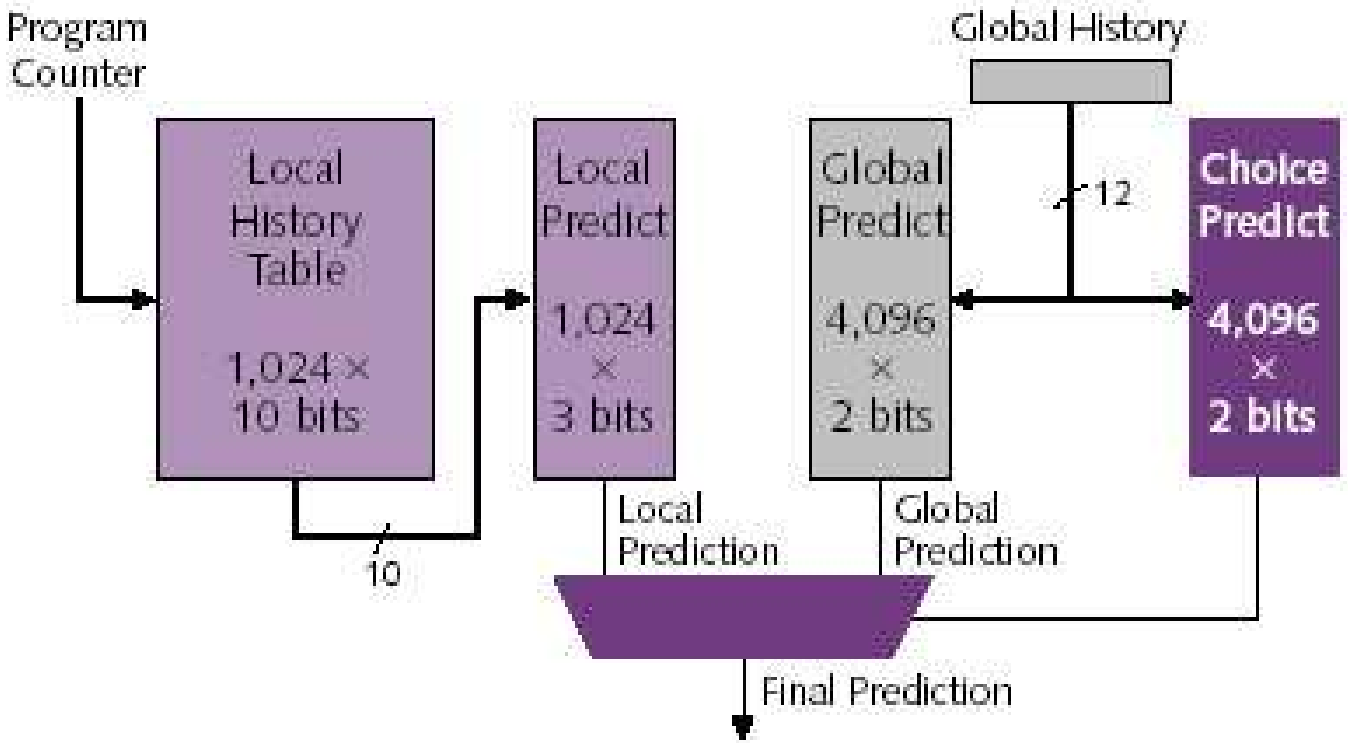
*Predict **which predictor** to use*

Use a 2-bit predictor for that:

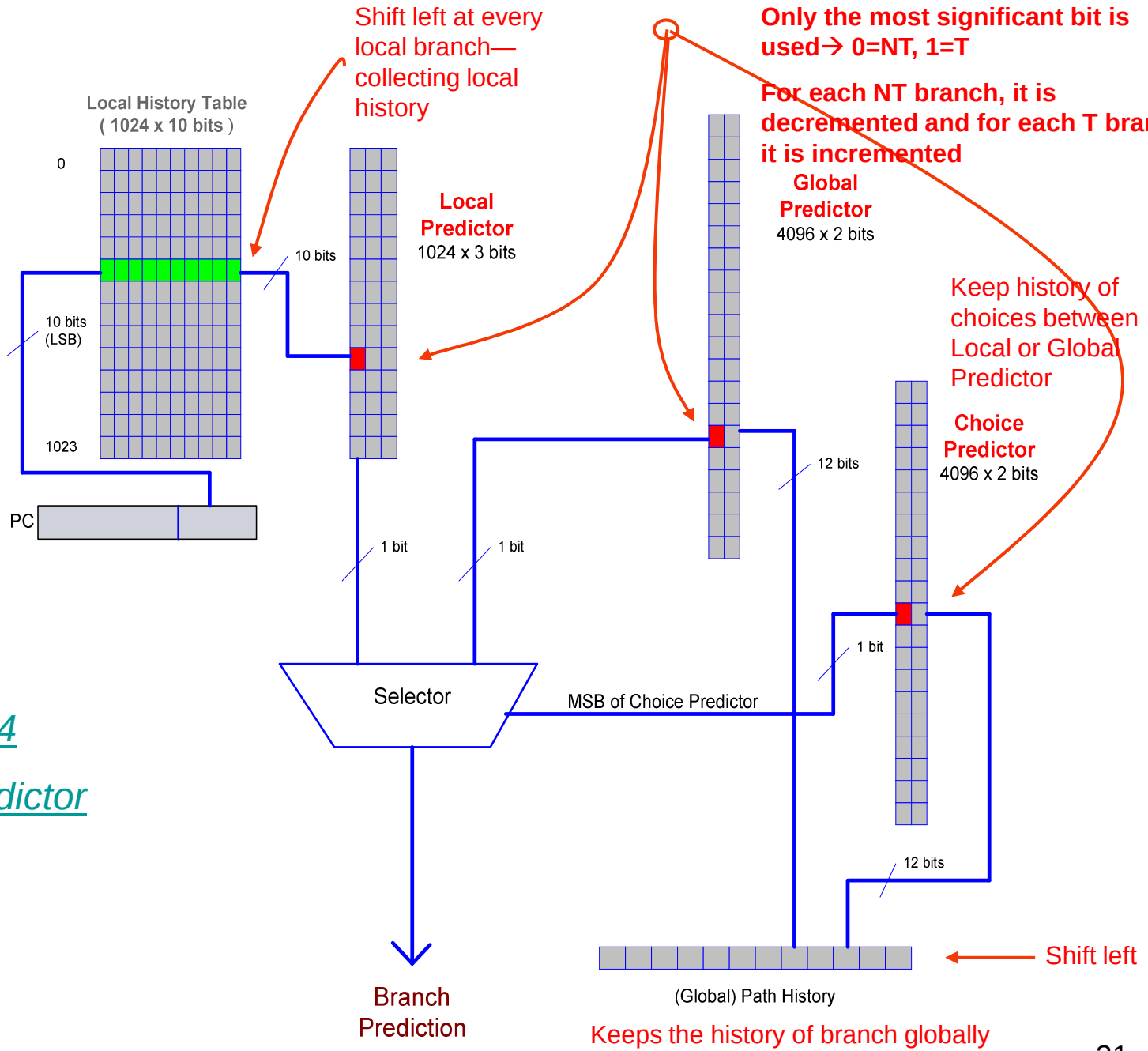
A predictor must be twice incorrect before we switch to the other one.



Alpha 21264 Branch Prediction



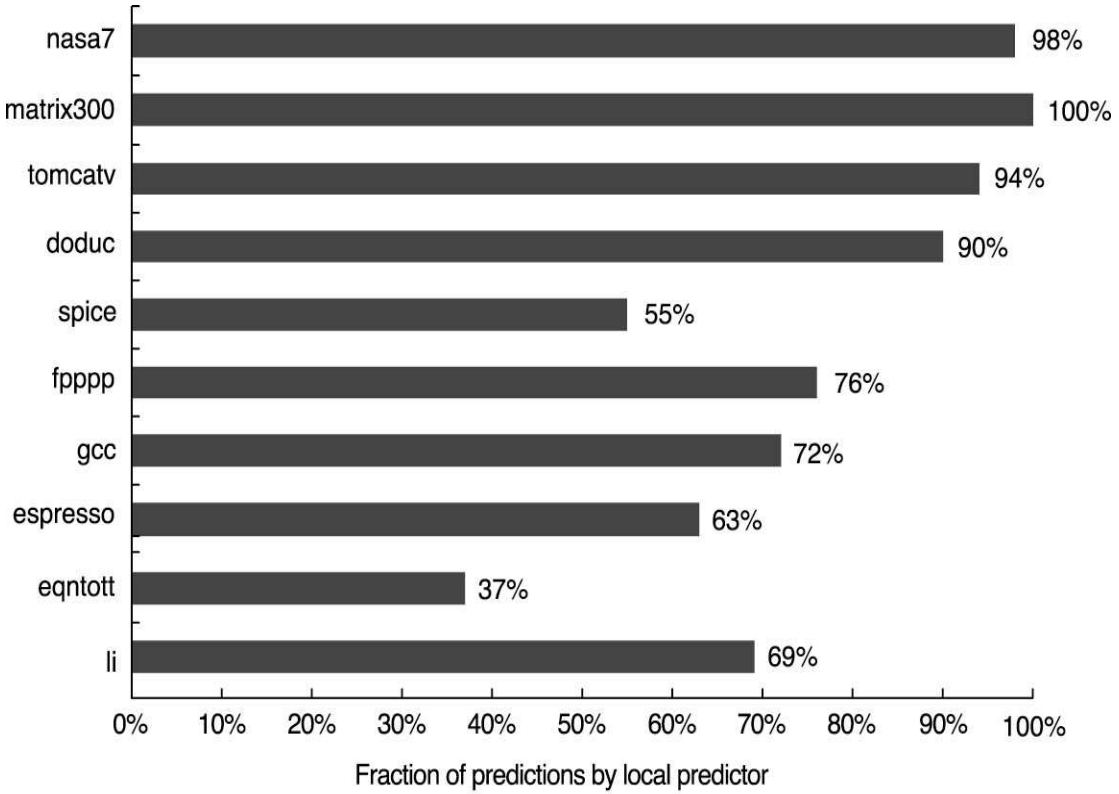
- Rather complex structure
- Representative of predictors used in current processors, e.g. Pentium4



Alpha 21264

Branch Predictor

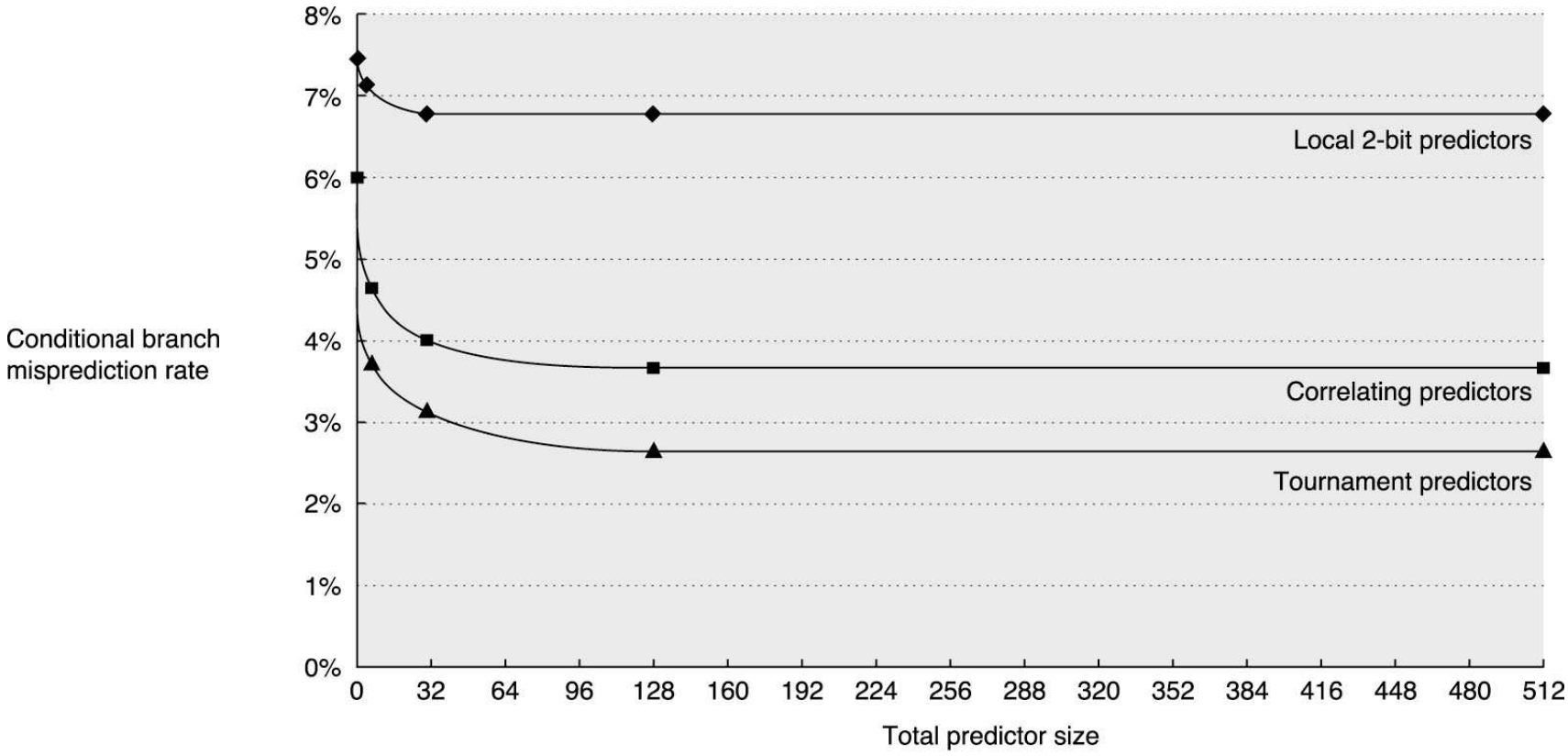
Fraction of predictions coming from the local predictor



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.

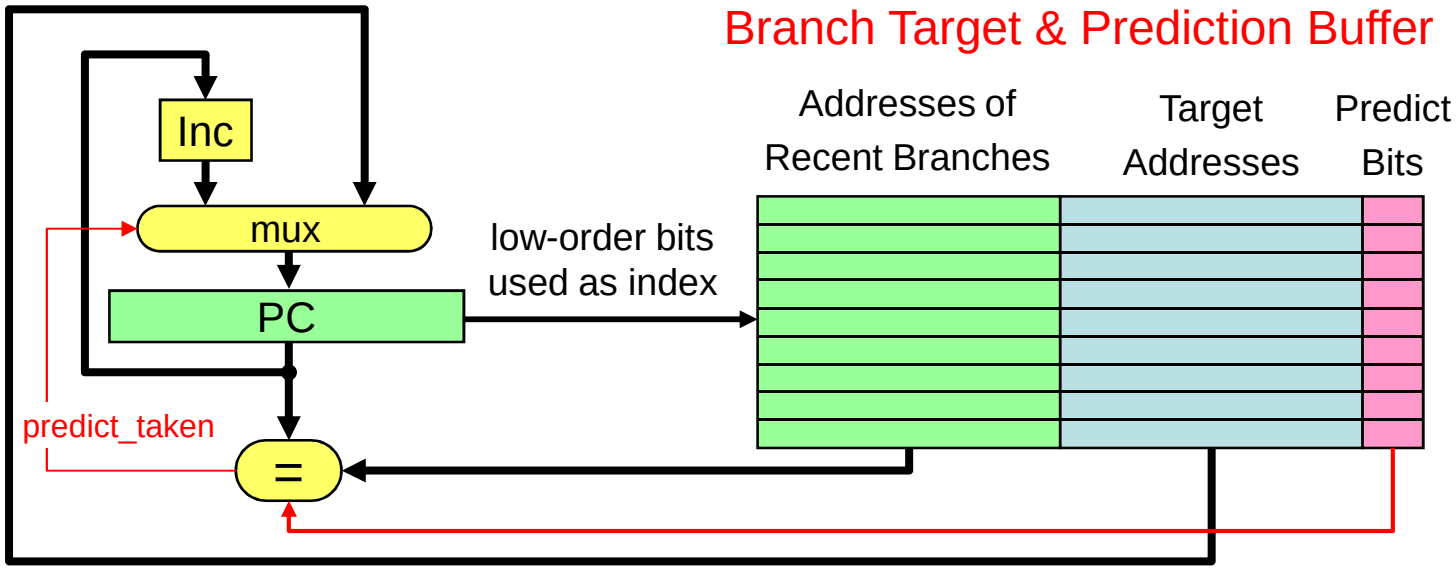
Misprediction rates



Zero-Delayed Branch

- Disadvantages of delayed branch
  - Branch delay can increase to multiple cycles in deeper pipelines
  - Branch delay slots must be filled with useful instructions or no-op
- How can we achieve zero-delay for a taken branch?
  - Branch target address is computed in the ID stage
- Solution
  - Check the PC to see if the instruction being fetched is a branch
  - Store the branch target address in a branch buffer in the IF stage
  - If branch is predicted taken then
    - Next PC = branch target fetched from branch target buffer
  - Otherwise, if branch is predicted not taken then Next PC = PC+4

Branch Target and Prediction Buffer



The steps involved in handling an instruction with a branch-target buffer