# Dynamic Branch Prediction Techniques

# Reminder:
# Branch Prediction Techniques

- Static Branch Prediction Techniques: The actions for a branch are fixed for each branch during the entire execution
- Dynamic Branch Prediction Techniques: The decisions regarding branch prediction can change for the same branch, during program execution

# Dynamic Branch Prediction

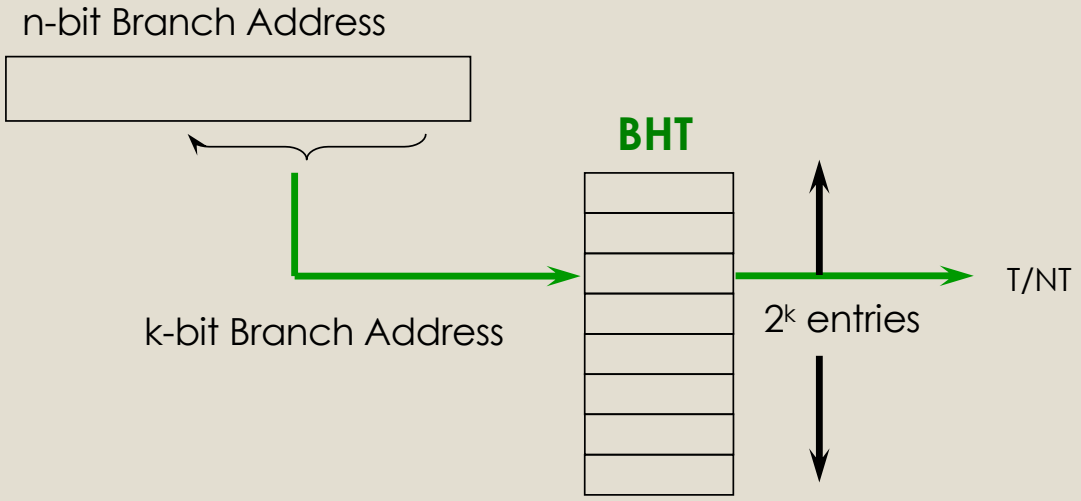
- Basic Idea:  To use the past branch behavior to predict the future
- Dynamically predict the outcome of a branch:
  - the prediction will depend on the behavior of the branch at run time
  - It can change if the branch changes its behavior during execution
    → advantage over static prediction

# Prediction influenced by Branch History

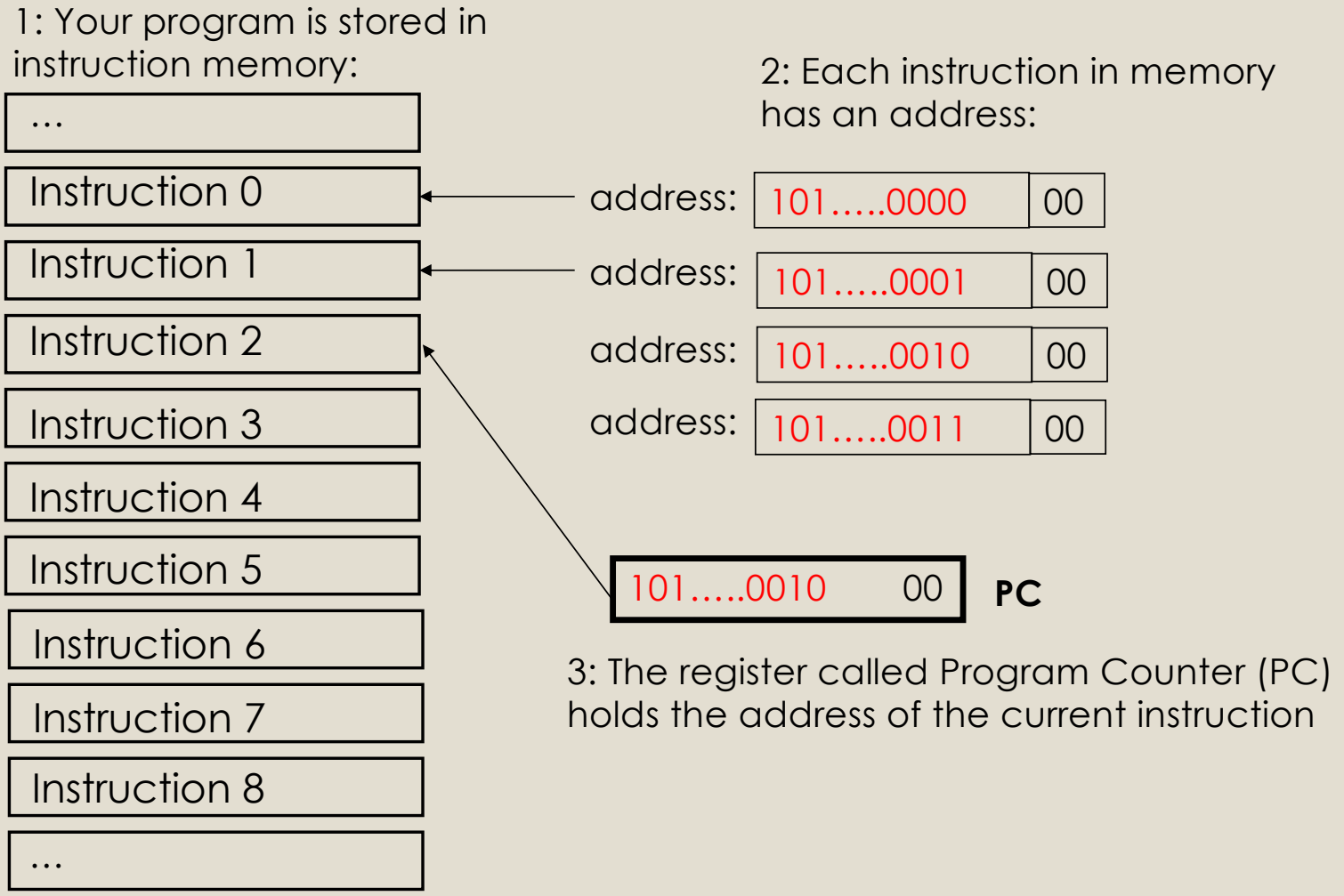
- If the prediction will take into account the past behaviour of the branch
- How do I know past behaviour? → I need to store the BRANCH HISTORY
- In a BRANCH HISTORY TABLE (BHT)

# Branch History Table

- Small table containing 1 bit for each branch entry
  - says whether the branch was recently taken (1) or not (0)
  - the table is indexed by the lower portion of the address of the branch instruction

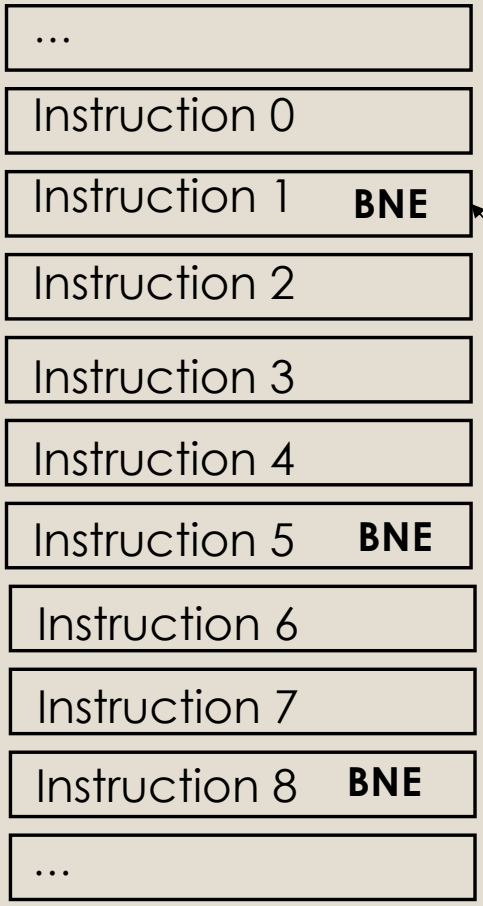


# Understanding the Branch History Table (1)



# Understanding the Branch History Table (2)

1: Some of the instructions in the program are branches – some are not




address: 101…..0000 00

address: 101…..0001 00

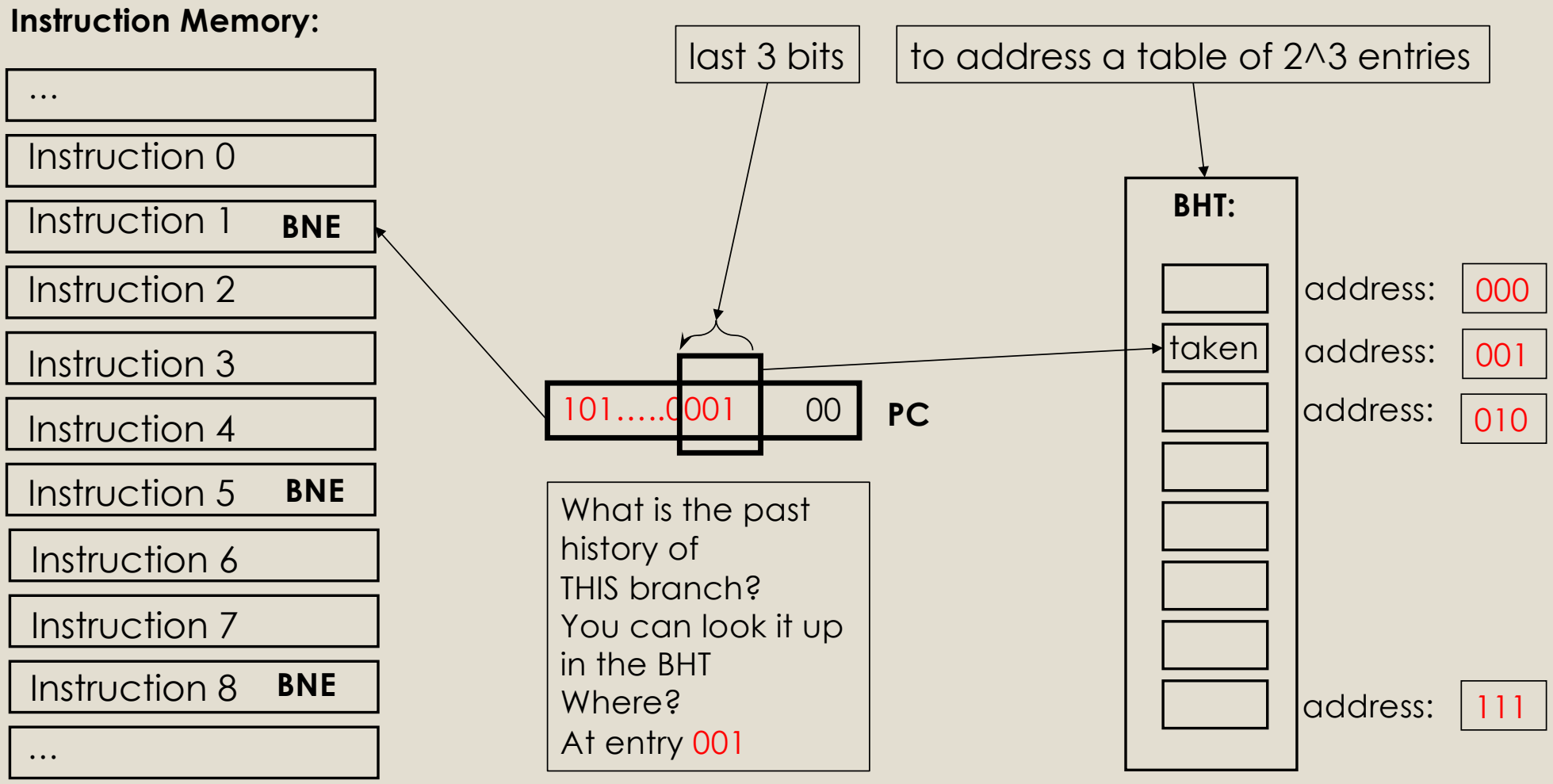
101…..0001 00 **PC**

2: The PC holds the address of a branch

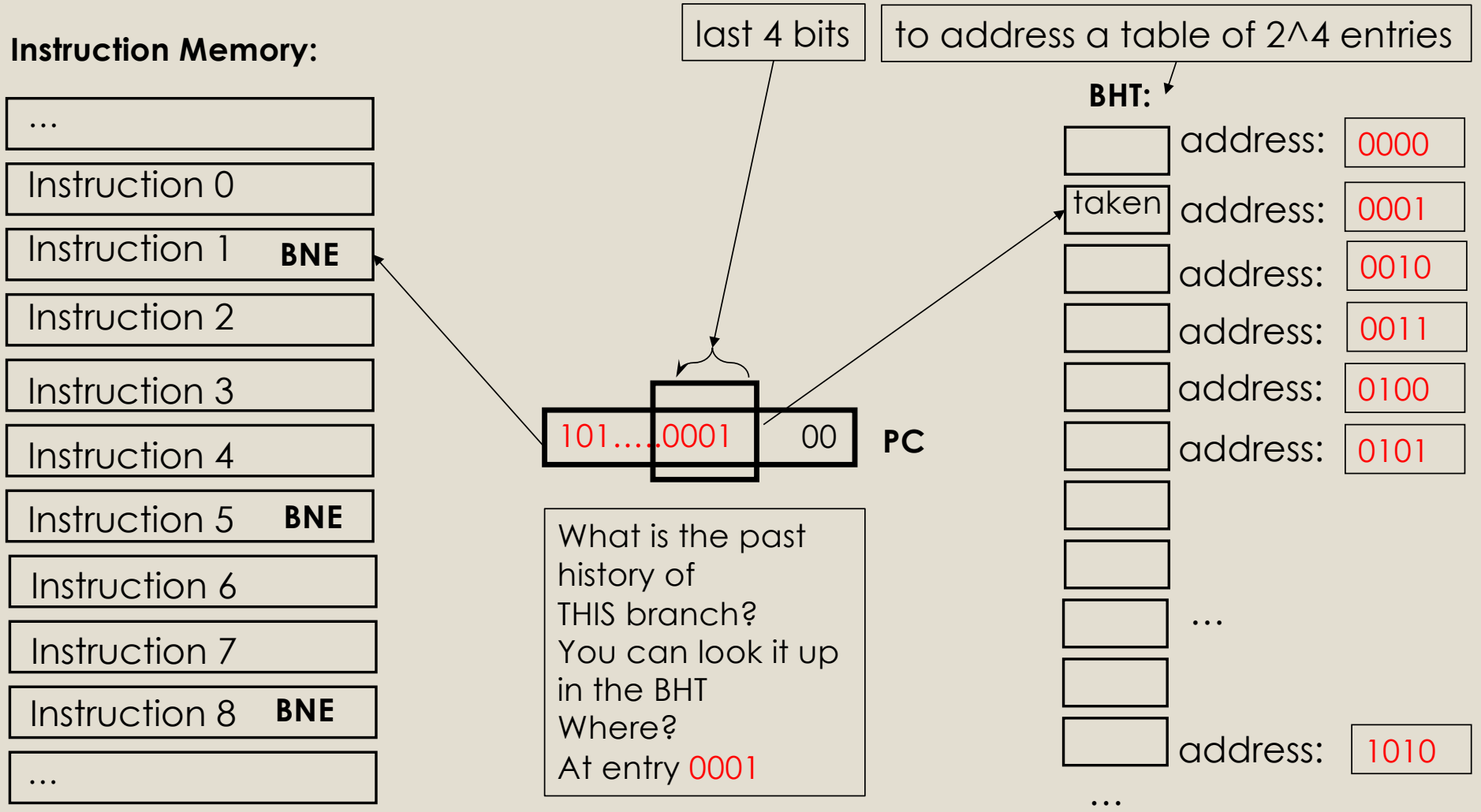
3: What is the past history of THIS particular branch?

4: It is held in the BHT (Branch History Table)

# Understanding the Branch History Table (3)



# Understanding the Branch History Table (4)



# Exercise

- Assume that a program contains 7 branches and that those branches are stored in instruction memory at the following addresses:

1. 001000001000
2. 001000001100
3. 001000011000
4. 001001011100
5. 001001110000
6. 001001111000
7. 001001111100

- Tell me, for each branch, which entry of the BHT it is mapped to, for a BHT of 4 entries, of 8 entries, or of 16 entries

# Exercise

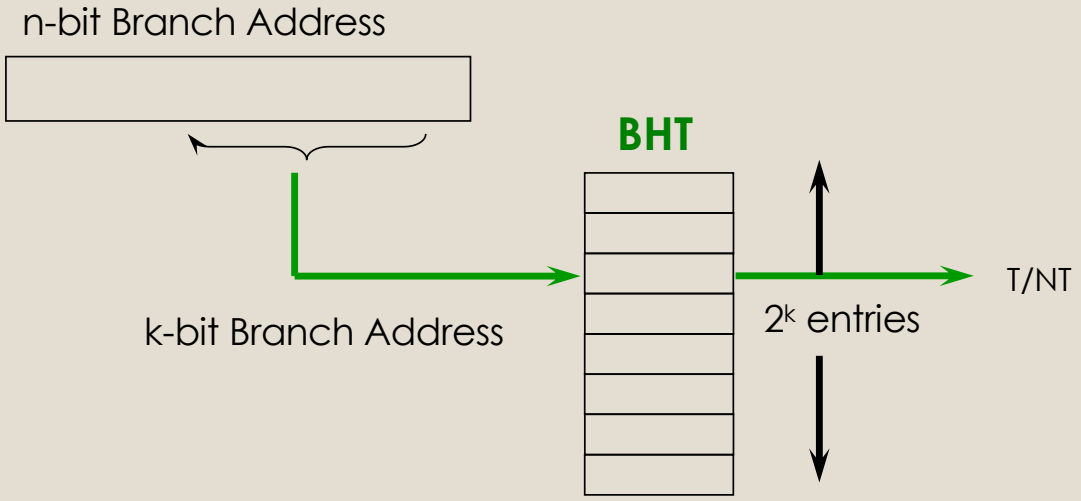
- Assume that a program contains 7 branches and that those branches are stored in instruction memory at the following addresses:

1. 001000001000
2. 001000001100
3. 001000011000
4. 001001011100
5. 001001110000
6. 001001111000
7. 001001111100

- Tell me, for each branch, which entry of the BHT it is mapped to, for a BHT of 4 entries, of 8 entries, or of 16 entries

# Branch History Table

- Small table containing 1 bit for each branch entry
  - says whether the branch was recently taken (1) or not (0)
  - the table is indexed by the lower portion of the address of the branch instruction



# Prediction

- BHT is a table containing 1 bit for each branch entry
  - says whether the branch was recently taken or not
- That bit (taken or not taken) is used as the prediction for this branch this time round
- A prediction is a hint that is assumed to be correct: fetching begins in the predicted direction
- If the hint turns out to be wrong, the prediction bit is inverted and stored back. The pipeline is flushed and the correct sequence is executed
- The prediction bit could have been put there by another branch with the same low-order address bits

# Shortcoming of:
# 1-bit Branch History Table

# Shortcoming of:
# 1-bit Branch History Table

Even if a branch is almost always taken, we will likely predict incorrectly twice, rather than once, when it is not taken

because of the flipped bit caused by the misprediction

# Why:

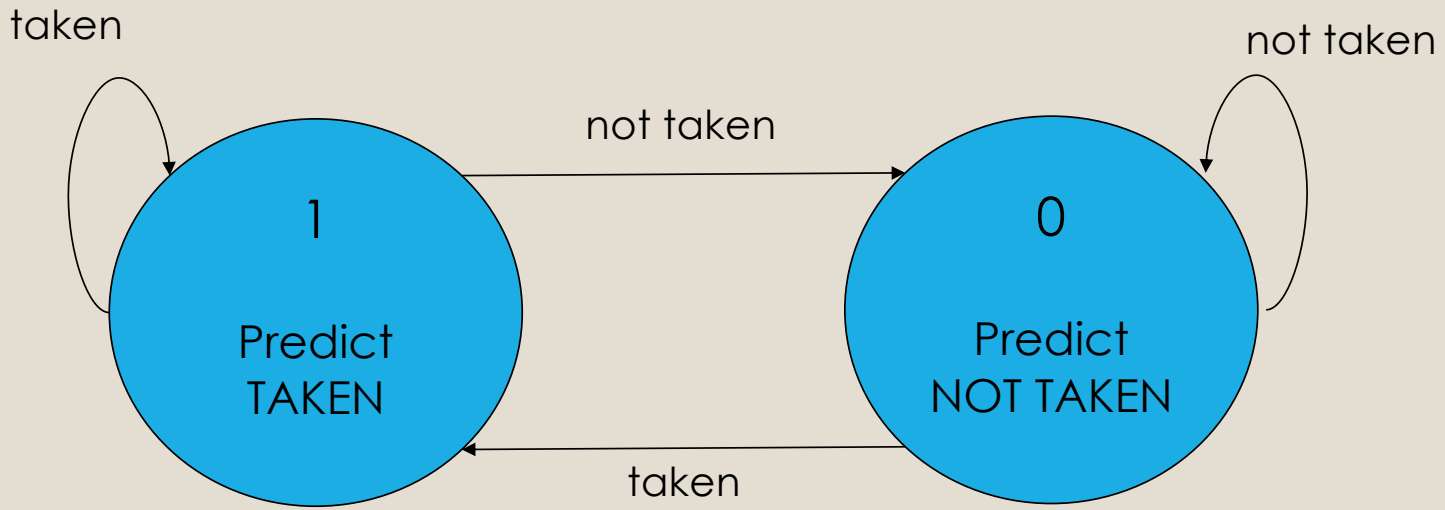
| Predicted outcome | | | Effective outcome |
|---|---|---|---|
| | | | taken |
| taken | 1 | | taken |
| taken | 1 | | taken |
| taken | 1 | | taken |
| taken | 1 | | taken |
| taken | 1 | | taken |
| taken | 1 | | taken |
| taken | 1 | MISPR | not taken |
| not taken | 0 | MISPR | taken |
| taken | 1 | | taken |

# How to solve it?

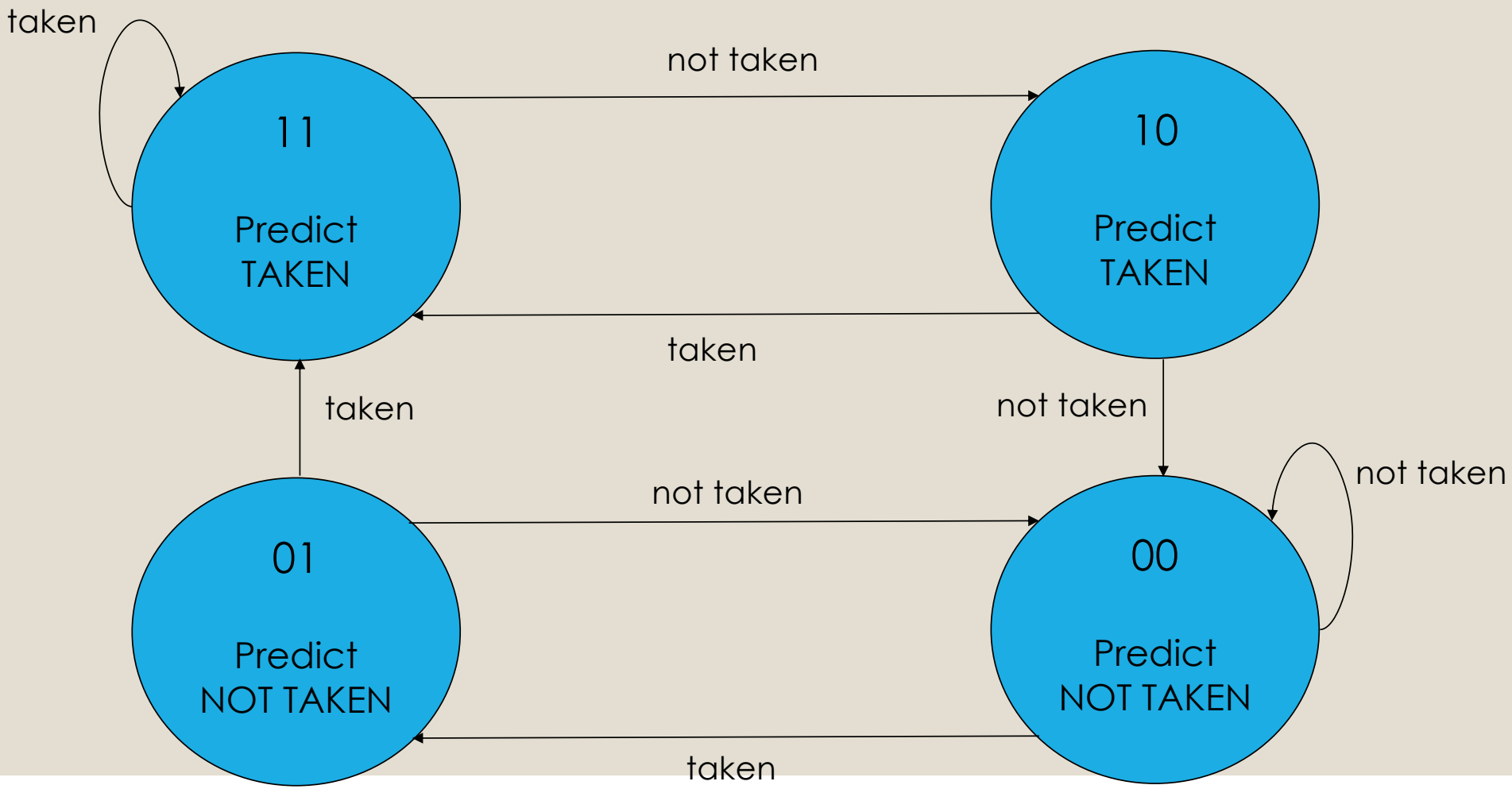
- How can we avoid mispredicting twice, for the common behaviour of the previous slide (which happens, for example, in most loops)

- Use more bits!

# In case of 1 bit we had:

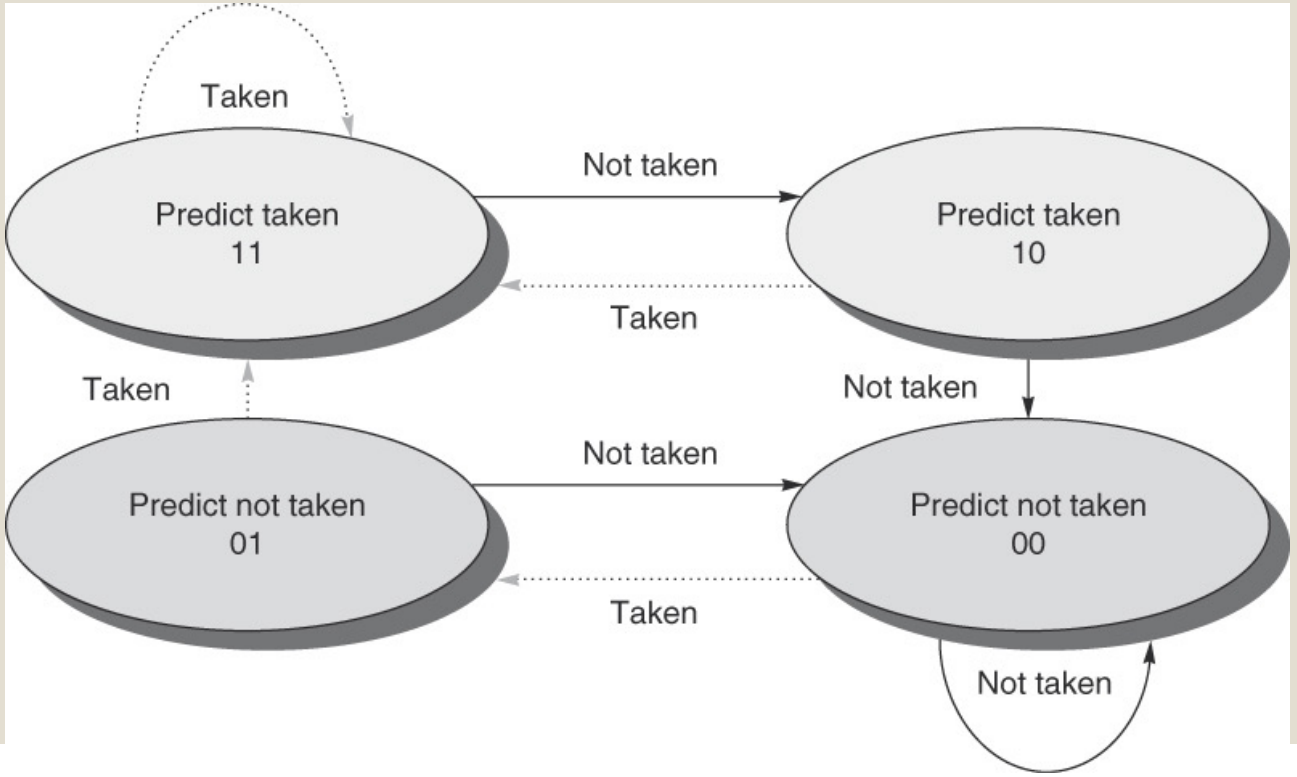


# What if we have 2 bits?



# 2-bit Branch History Table

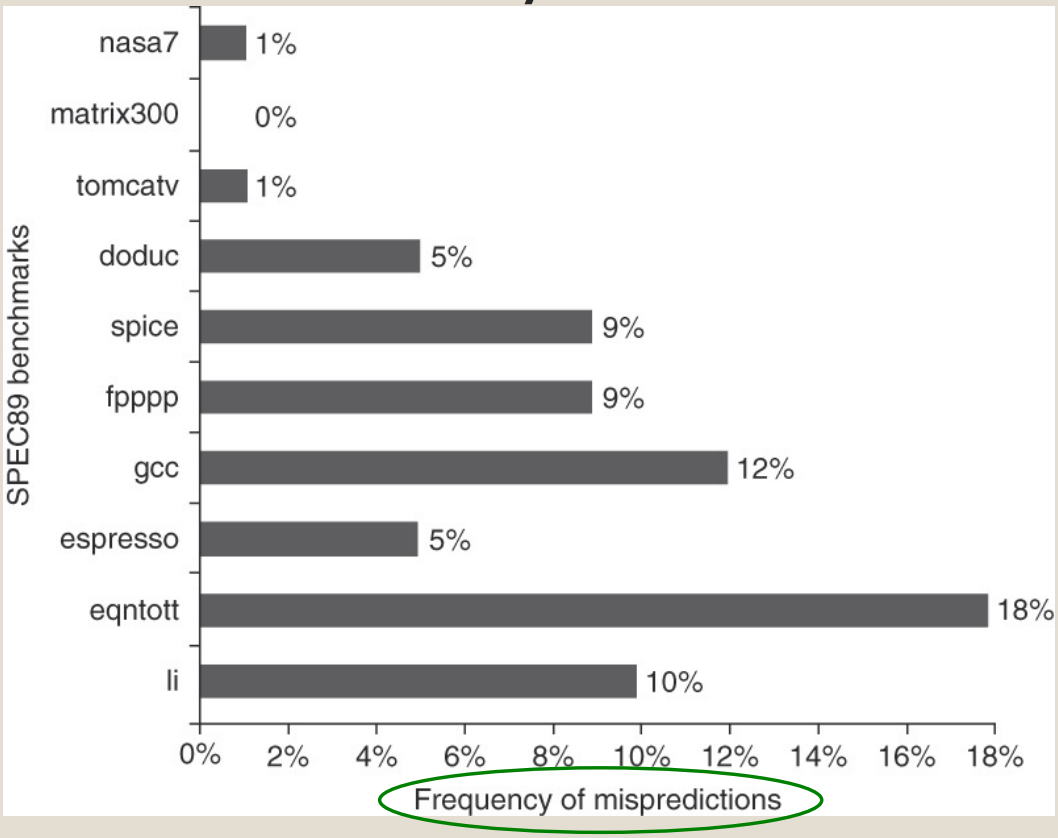
- The problem is solved by using a 2-bit BHT



# Now:

| Predicted outcome | | | Effective outcome |
|---|---|---|---|
| | | | taken |
| taken | 11 | | taken |
| taken | 11 | | taken |
| taken | 11 | | taken |
| taken | 11 | | taken |
| taken | 11 | | taken |
| taken | 11 | | taken |
| taken | 11 | MISPR | not taken |
| taken | 10 | | taken |
| taken | 11 | | taken |

# Accuracy of 2-bit Branch History Table




For a 4096-entry 2-bit prediction (IBM Power architecture)

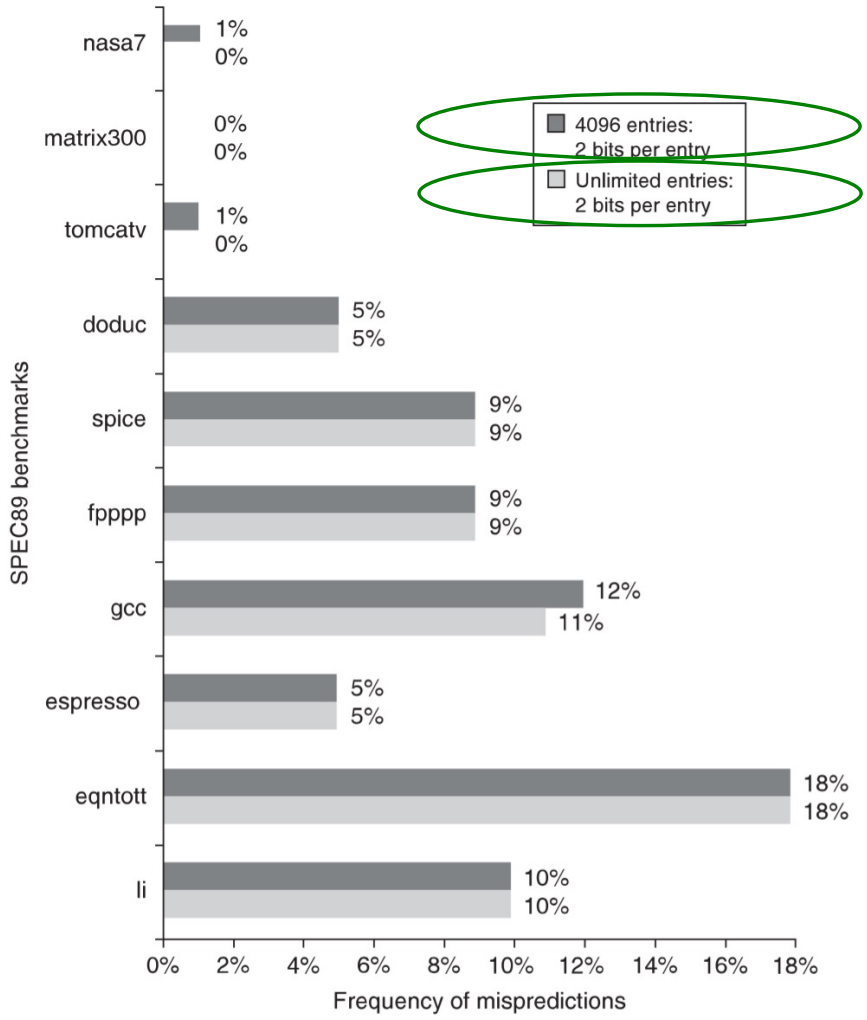



**Figure C.20 Prediction accuracy of a 4096-entry 2-bit prediction buffer versus an infinite buffer for the SPEC89 benchmarks.** Although these data are for an older version of a subset of the SPEC benchmarks, the results would be comparable for newer versions with perhaps as many as 8K entries needed to match an infinite 2-bit predictor.

# Exercises:

- Given a certain branch, and assuming no other branch in the program is conflicting with this branch for space in the BHT,
- assuming the BHT is initialized with all 1s (1 = taken),
- calculate the percentage of mispredictions, both in case of a 1-bit BHT and of a 2-bit BHT,
- when the program goes through the branch 10 times, and the branch outcome sequence for these ten times is:

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Case 0) | NT | NT | NT | NT | NT | NT | NT | NT | NT | NT |
| Case 1) | T | T | T | T | T | NT | T | T | T | T |
| Case 2) | T | NT | T | NT | T | NT | T | NT | T | NT |

# Let's start with Case 0) :

- Given a certain branch, and assuming no other branch in the program is conflicting with this branch for space in the BHT,
- assuming the BHT is initialized with all 1s (1 = taken),
- calculate the percentage of mispredictions, both in case of a 1-bit BHT and of a 2-bit BHT,
- when the program goes through the branch 10 times, and the branch outcome sequence for these ten times is:

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Case 0) | NT | NT | NT | NT | NT | NT | NT | NT | NT | NT |

# Let's start with Case 0) :

- Given a certain branch, and assuming no other branch in the program is conflicting with this branch for space in the BHT,
- assuming the BHT is initialized with all 1s (1 = taken),
- calculate the percentage of mispredictions, both in case of a 1-bit BHT and of a 2-bit BHT,
- when the program goes through the branch 10 times, and the branch outcome sequence for these ten times is:

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Case 0) | NT | NT | NT | NT | NT | NT | NT | NT | NT | NT |

1-bit BHT: Misprediction rate: 1/10

# Let's start with Case 0) :

- Given a certain branch, and assuming no other branch in the program is conflicting with this branch for space in the BHT,
- assuming the BHT is initialized with all 1s (1 = taken),
- calculate the percentage of mispredictions, both in case of a 1-bit BHT and of a 2-bit BHT,
- when the program goes through the branch 10 times, and the branch outcome sequence for these ten times is:

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Case 0) | NT | NT | NT | NT | NT | NT | NT | NT | NT | NT |

2-bit BHT: Misprediction rate: 2/10

# You do it for Case 1) and for Case 2)

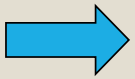
# Correlating Branch Predictors

# Correlating Branch Predictors

- Both 1-bit or 2-bit BHT use only the recent behavior of a single branch to predict future behaviour
- Basic Idea: let's look at the behaviour of *other* branches too, rather than just the branch we are trying to predict

# Example of Correlating Branches

```
    if(a==2) a = 0;   b1
L1: if(b==2) b = 0;   b2
L2: if(a!=b) {…};     b3
L3:
```



```
    subi r3,r1,2
    bnez r3,L1     ;b1
    add  r1,r0,r0 ;(set a=0)
L1: subi r3,r2,2
    bnez r3,L2     ;b2
    add  r2,r0,r0 ;(set b=0)
L2: sub  r3,r1,r2
    beqz r3,L3;    ;b3
L3:
```

Is there a **correlation** among the behaviour of branches **b1**, **b2**, **b3**?

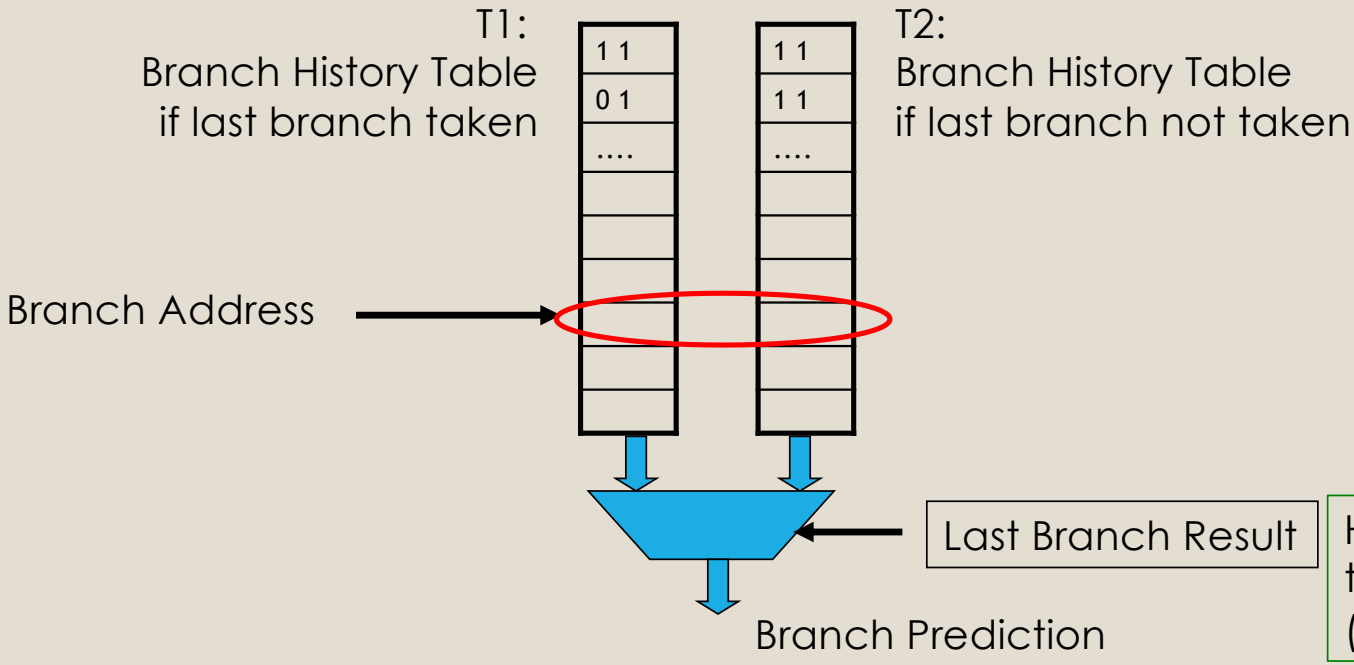
Branch **b3** is correlated to previous branches **b1** and **b2**.
In fact: if the previous branches are both *not taken*, (then: a = 0, and b = 0)
then **b3** will be *taken* **(**because now **a==b)**

# Correlating Branch Predictors

- Branch predictors that use the behaviour of other branches to make a prediction are called Correlating Predictors or 2-level Predictors

- Example: a (1,2) Correlating Predictors means a 2-bit predictor with 1 bit of correlation: the behavior of the most recent branch is used to choose among a pair of 2-bit branch predictors

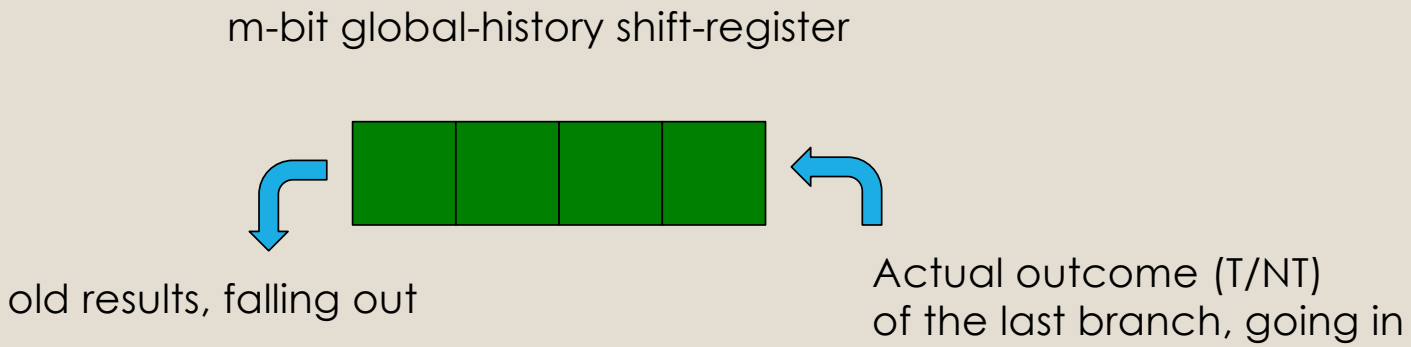
# Correlating Branch Predictors: Example




How do we keep
the last branch result(s)?
(next slide)

# The global history of the last m branches

can be recorded in a simple m-bit shift register




This info is then used to access the correct branch history table

# (m,n) correlating (or global) predictors

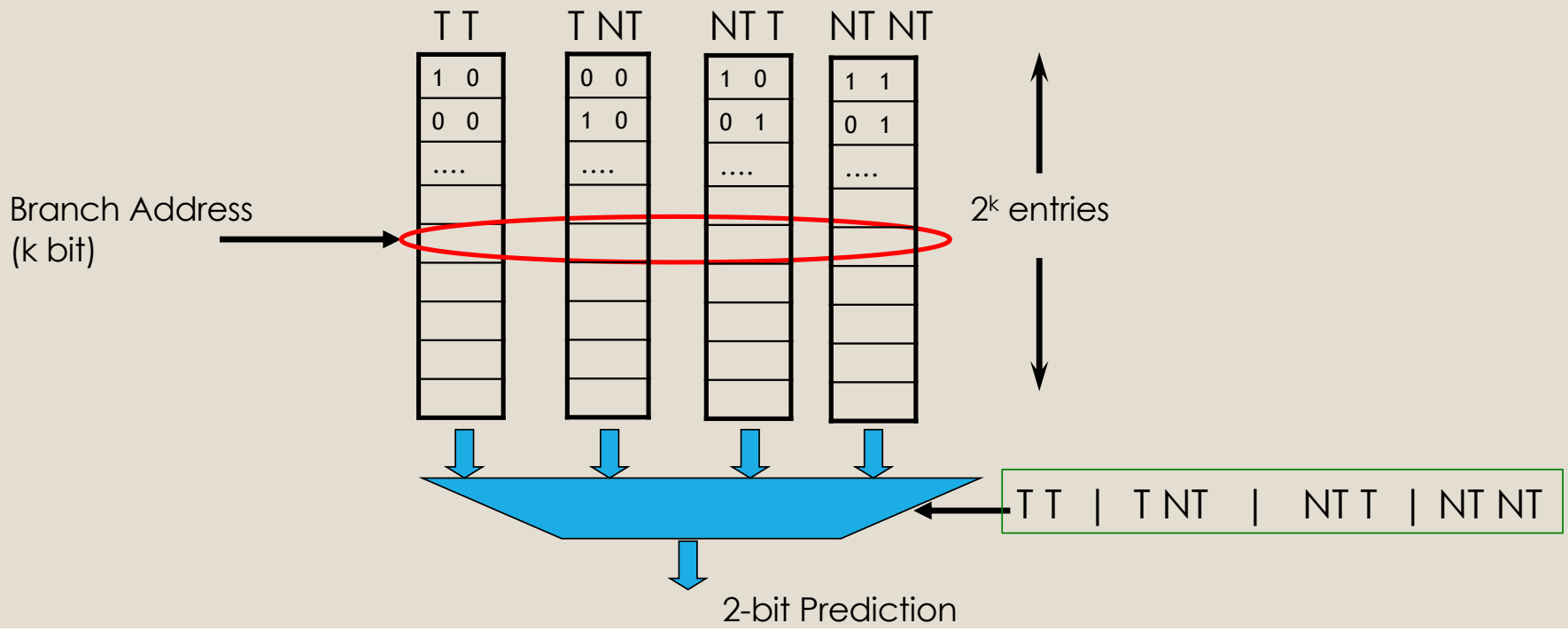
- In general, an (m, n) correlating predictor records the behaviour of the most recent m branches to choose from $2^m$ BHTs, each of which is an n-bit predictor

# An (m,n) = (2,2) correlating predictor

- $2^m = 2^2 = 4$ different BHTs (4 possibilities for the 2 most recent branches: TT, TNT, NTT, NTNT)
- Each BHT is a 2 bit predictor

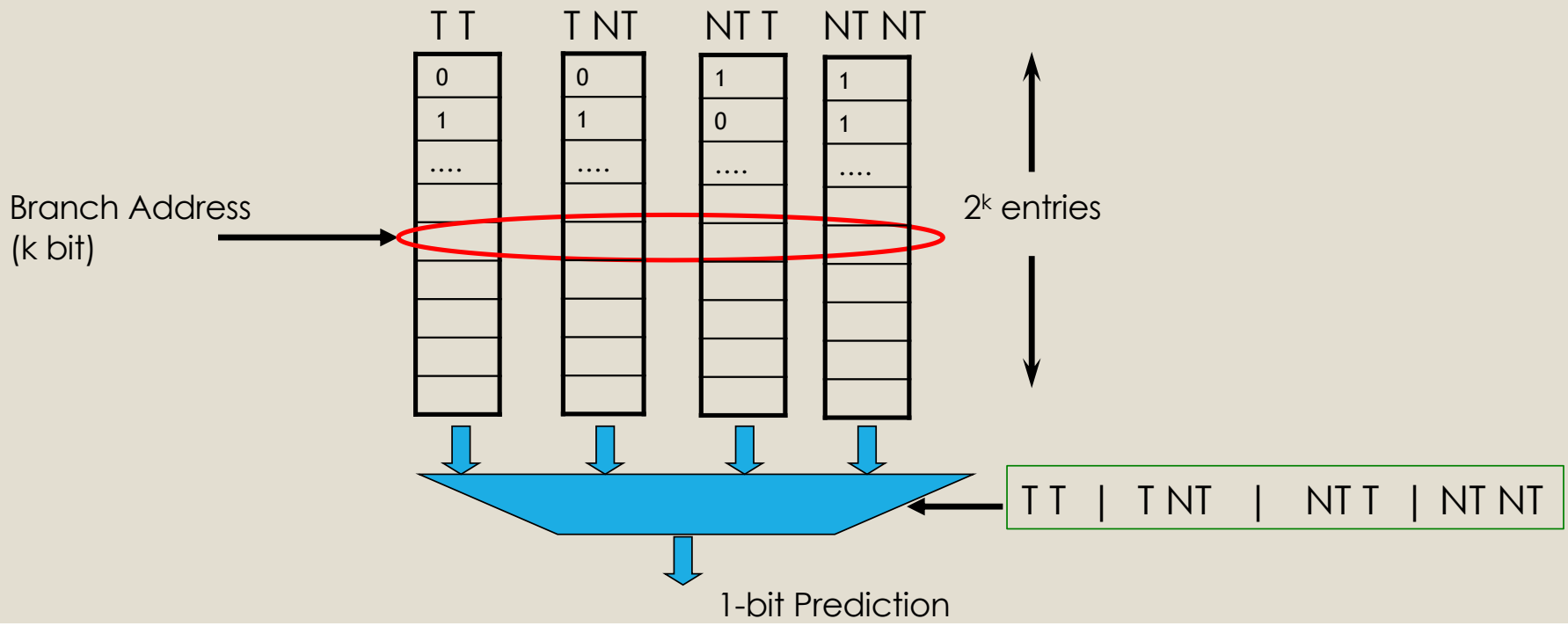
# An (m,n) = (2,2) correlating predictor

- $2^m = 2^2 = 4$ different BHTs (4 possibilities for the 2 most recent branches: TT, TNT, NTT, NTNT)
- Each BHT is a 2 bit predictor



# An (m,n) = (2,1) correlating predictor

- $2^m = 2^2 = 4$ different BHTs (4 possibilities for the 2 most recent branches: TT, TNT, NTT, NTNT)
- Each BHT is a 1 bit predictor

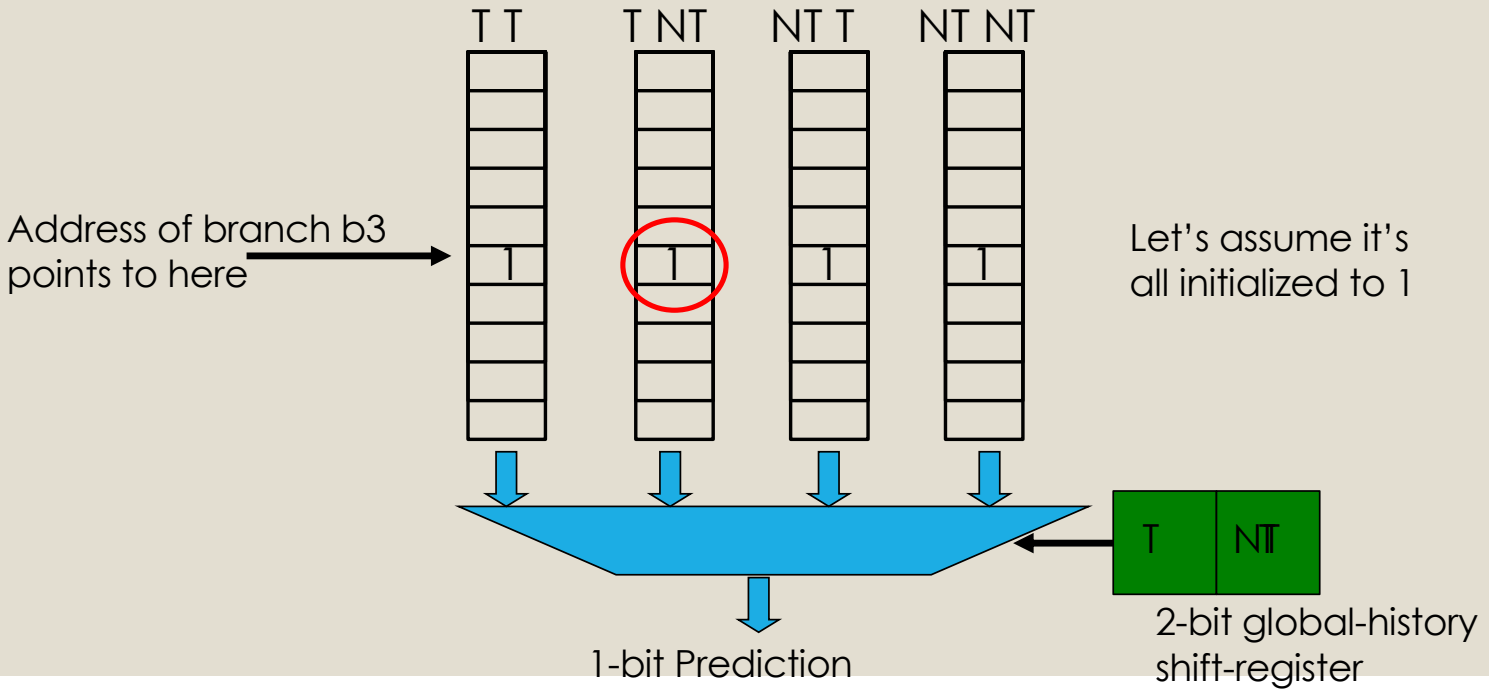


REAL outcome:

| | | |
|---|---|---|
| T | `if(a==2) a = 0;` | **b1** |
| NT | `if(b==2) b = 0;` | **b2** |
| NT | `if(a!=b) {…};` | **b3** |

PREDICT TAKEN

MISPREDICTED

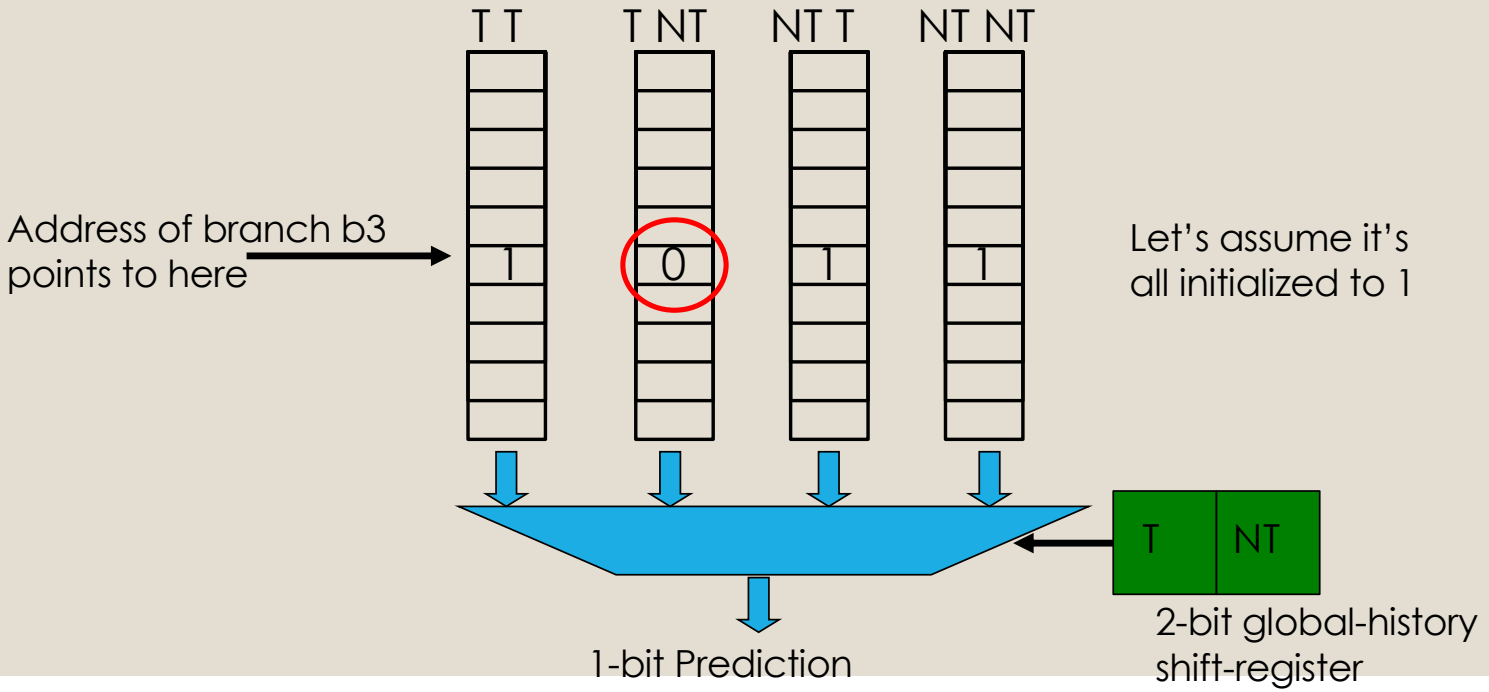


REAL outcome

| | | |
|---|---|---|
| T | `if(a==2) a = 0;` | **b1** |
| NT | `if(b==2) b = 0;` | **b2** |
| NT | `if(a!=b) {…};` | **b3** |

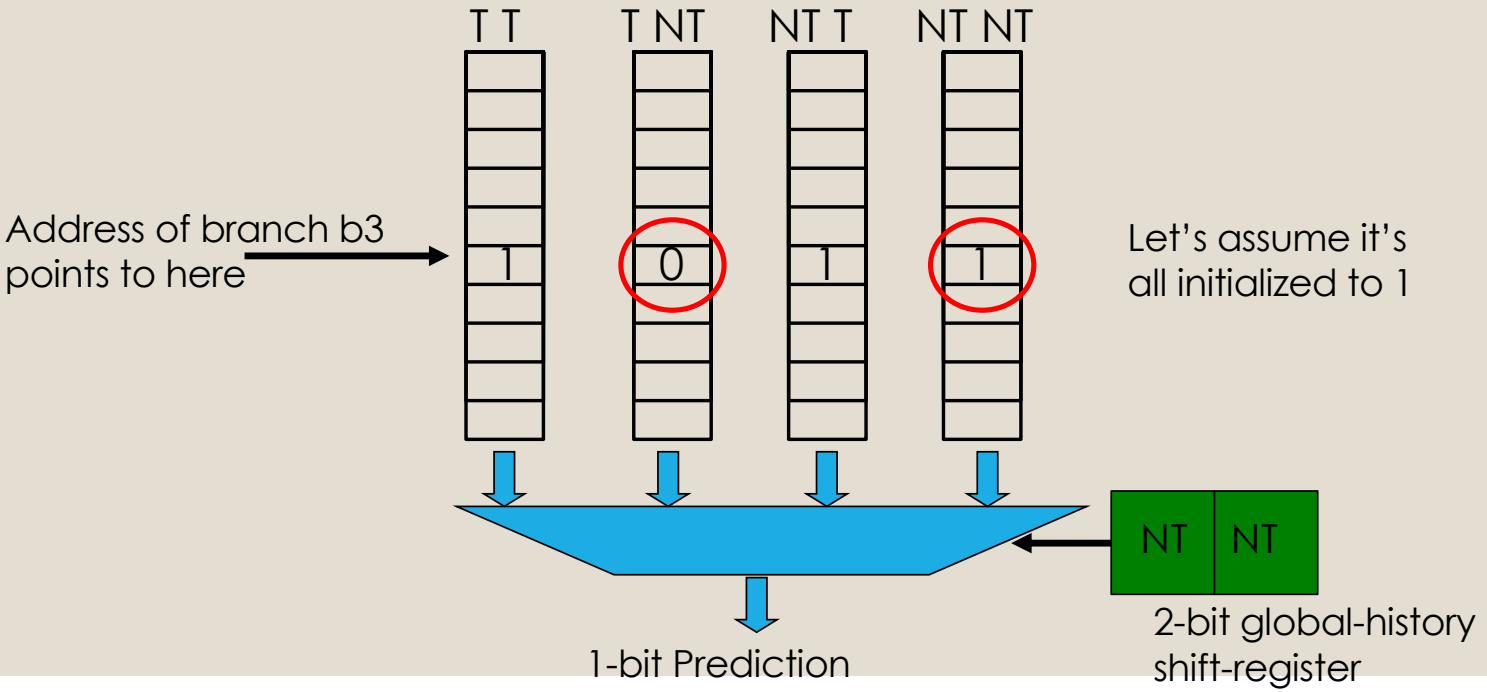
PREDICT TAKEN

MISPREDICTED



REAL outcome:

| | | |
|---|---|---|
| NT | `if(a==2) a = 0;` | **b1** |
| NT | `if(b==2) b = 0;` | **b2** |
| T | `if(a!=b) {…};` | **b3** PREDICT TAKEN |



# Exercise

```
if(a==2) {…};   b1
if(b==2) {…};   b2
if(a!=b) {…};   b3
if(a==4) {…};   b4
```

We'll analyze the behaviour of various predictors, for this snippet of code according to the branch outcome given in the next slide

# Exercise

- A snippet of program has a sequence of four branches: b1, b2, b3, b4
- During five subsequent executions of this snippet of code, the outcome of these four branches are the ones written in the table below
- Assume that no branch in the code alias, i.e. they all point to different locations in the BHTs
- Also assume that the control flow is such that the four branches are indeed always executed in this sequence. So: b1b2b3b4 .... b1b2b3b4 ... b1b2b3b4 ... etc

| | t=1 | t=2 | t=3 | t=4 | t=5 |
|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT |
| b3 | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT |

# Exercise 1) Calculate:

- What are the bits stored in the BHTs
  - where: at the line pointed to by branch b3
  - when: at the end of each execution of branch b3
- [Assume a (2,2) branch predictor, initially filled with 1s]
- Also calculate the misprediction rate, again for branch b3

| | t=1 | t=2 | t=3 | t=4 | t=5 |
|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT |
| b3 | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT |

# Exercise 1) Answer

| | | T T | T NT | NT T | NT NT |
|---|---|---|---|---|---|
| t=1 | b3 | 11 | 11 | 11 | 11 |
| t=2 | b3 | 10 | 11 | 11 | 11 |
| t=3 | b3 | 10 | 11 | 11 | 11 |
| t=4 | b3 | 11 | 11 | 11 | 11 |
| t=5 | b3 | 11 | 11 | 11 | 10 |

- What are the bits stored in the BHTs
  - where: at the line pointed to by branch b3
  - when: at the end of each execution of branch b3
- [Assume a (2,2) branch predictor, initially filled with 1s]
- Also calculate the misprediction rate, again for branch b3

| | t=1 | t=2 | t=3 | t=4 | t=5 |
|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT |
| b3 | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT |

misprediction rate = 2/5

# Exercise 2) Now you do it. Calculate:

- What are the bits stored in the BHTs
  - where: at the line pointed to by branch **b4**
  - when: at the end of each execution of branch b4
- [Assume the same (2,2) branch predictor, initially filled with 1s]

- Also calculate the misprediction rate, again for branch **b4**

| | t=1 | t=2 | t=3 | t=4 | t=5 |
|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT |
| b3 | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT |

# Exercise 3) Calculate:

- What are the bits stored in the BHTs
  - where: at the line pointed to by branch **b4**
  - when: at the end of each execution of branch b4
- [Now for a **(3,1)** branch predictor, initially filled with 1s]
- Also calculate the misprediction rate, again for branch **b4**

| | t=1 | t=2 | t=3 | t=4 | t=5 |
|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT |
| b3 | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT |

# I started exercise 3) in class

- Note that during class I made a mistake ☺ because at time t=2 I recorded a lookup in the wrong BHT
- Below is the correct version. the lookup at time t=2 happens in column: T T NT

| | T T T | T T NT | T NT T | T NT NT | NT T T | NT T NT | NT NT T | NT NT NT |
|---|---|---|---|---|---|---|---|---|
| b1 | | | | | | | | |
| b2 | | | | | | | | |
| b3 | | | | | | | | |
| b4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

| T | T | NT |
|---|---|---|

| | t=1 | t=2 | t=3 | t=4 | t=5 |
|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT |
| b3 | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT |

| | | T T T | T T NT | T NT T | T NT NT | NT T T | NT T NT | NT NT T | NT NT NT |
|---|---|---|---|---|---|---|---|---|---|
| t=1 | b4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| t=2 | b4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| t=3 | b4 | ... | | | | | | | |
| t=4 | b4 | ... | | | | | | | |
| t=5 | b4 | .... | | | | | | | |

Now you can continue this exercise, so that you fill up the next 3 iterations (t=3, t=4, t=5). The answer is in the next slide.

# Exercise 3) Answer:

| | | T T T | T T NT | T NT T | T NT NT | NT T T | NT T NT | NT NT T | NT NT NT |
|---|---|---|---|---|---|---|---|---|---|
| t=1 | b4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| t=2 | b4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| t=3 | b4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| t=4 | b4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| t=5 | b4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |

- What are the bits stored in the BHTs
  - where: at the line pointed to by branch **b4**
  - when: at the end of each execution of branch b4
- [Now for a **(3,1)** branch predictor, initially filled with 1s]
- Also calculate the misprediction rate, again for branch **b4**

| | t=1 | t=2 | t=3 | t=4 | t=5 |
|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT |
| b3 | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT |

misprediction rate = 2/5

# Exercise 4) Now you do it

- Continue the simulation of the previous slide [so, for a (3,1) branch predictor], for five more iterations of these branches. Therefore for t = 6,7,8,9,10
- for t = 6,7,8,9,10 , assume exactly the same outcome as before, repeated, i.e., as shown in the table below
- What are the bits stored in the BHTs
  - where: again, at the lines pointed to by branch **b4**
  - when: again, at the end of each execution of branch b4
- What is the misprediction rate now?
- (you will see that the predictor has started to learn … )

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 |
|---|---|---|---|---|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT | T | T | T | T | NT |
| b3 | T | NT | T | T | NT | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT | NT | T | NT | T | NT |

# Exercise 5) Now you do it

- For the same table in the previous slide, also reported here
- But now assume a (0,1) predictor,
- i.e. a 1-bit predictor with no correlation
- and assume, as usual, that the BHT is initially filled with 1s

- What is, in this case, the misprediction rate for branch b4?

- (you will see that correlation was indeed helping … )

| | t=1 | t=2 | t=3 | t=4 | t=5 | t=6 | t=7 | t=8 | t=9 | t=10 |
|---|---|---|---|---|---|---|---|---|---|---|
| b1 | NT | T | NT | T | NT | NT | T | NT | T | NT |
| b2 | T | T | T | T | NT | T | T | T | T | NT |
| b3 | T | NT | T | T | NT | T | NT | T | T | NT |
| b4 | NT | T | NT | T | NT | NT | T | NT | T | NT |

# REMINDER: performance before correlation

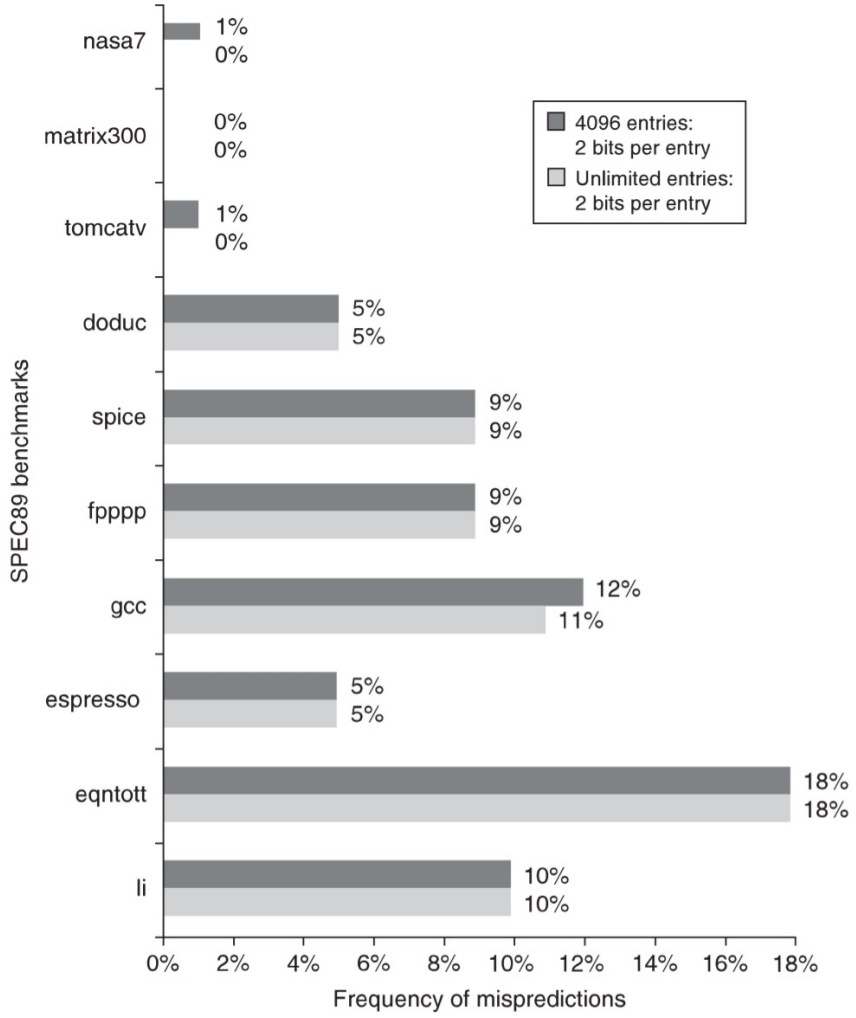



**Figure C.20 Prediction accuracy of a 4096-entry 2-bit prediction buffer versus an infinite buffer for the SPEC89 benchmarks.** Although these data are for an older version of a subset of the SPEC benchmarks, the results would be comparable for newer versions with perhaps as many as 8K entries needed to match an infinite 2-bit predictor.

# Performance

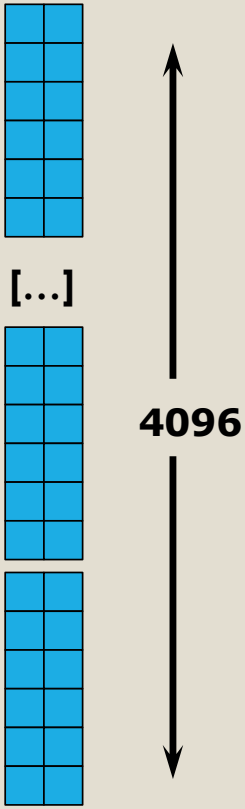
- Compare performance of correlating branch predictors
  - Also called global predictors

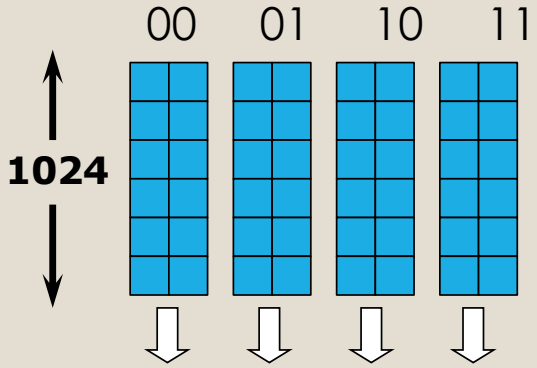
- With non-correlating
  - Also called local predictors

A local predictor (no global history) of 2 bits per entry
is a (0,2) predictor
$2^0$ = 1 BHT, and 2 bits per entry in the BHT)

# (0,2) x 4096 vs (2,2) x 1024

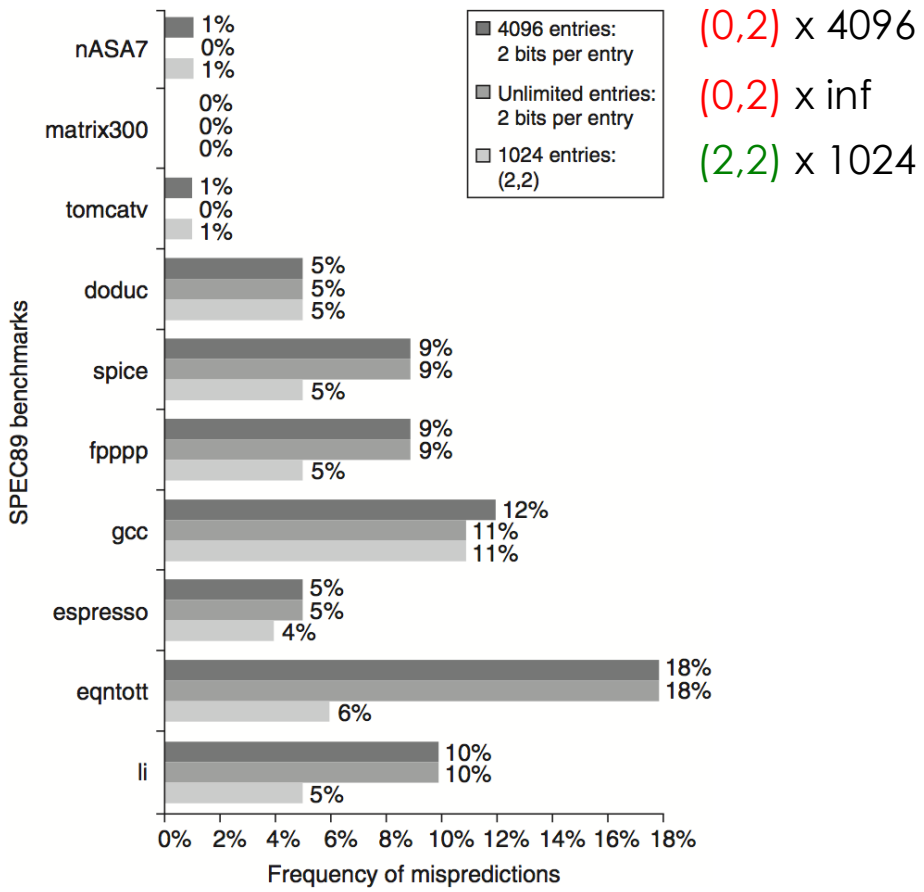
a local (0 global history) x 4096 entries needs the same amount of bits as a global one (2-bits global history) with a fourth (1/ $2^2$) of the entries




# Performance of correlating branch predictors






(0,2) x 4096

(0,2) x inf

(2,2) x 1024

**Figure 3.3 Comparison of 2-bit predictors.** A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating 2-bit predictor with unlimited entries and a 2-bit predictor with 2 bits of global history and a total of 1024 entries. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks would show similar differences in accuracy.

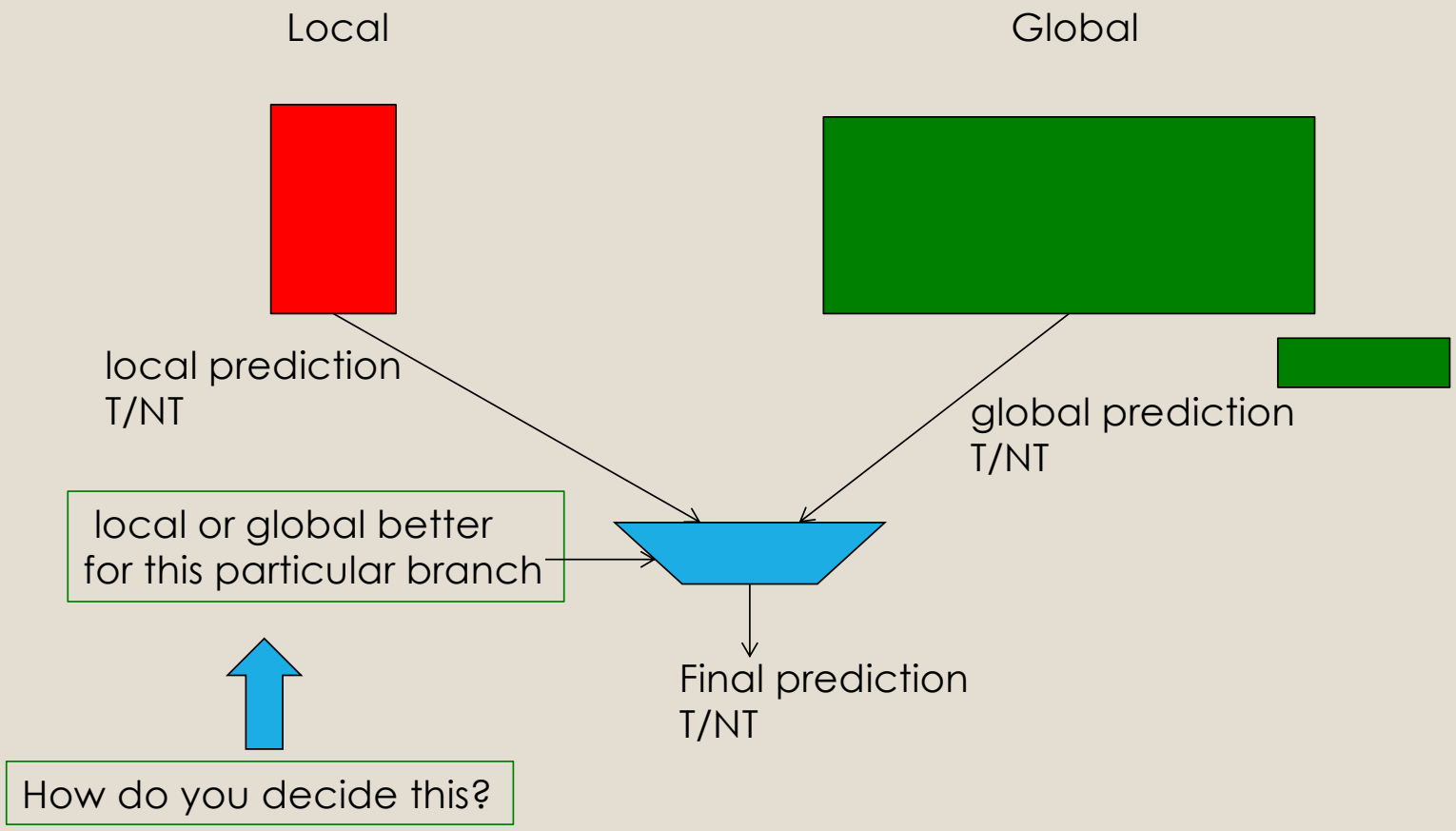
# Problem

- Some branches are indeed correlated
  - They benefit from correlating predictors
- Some are not
  - The right prediction gets spoiled by the correlating predictor

# Tournament predictors

- Combine local and global predictors
- and select the right prediction for each particular branch
- (so if one branch has correlation, use correlation, if one does not, use local prediction)

# Tournament predictors



# How do you decide local/global

- You need an indicator of which has been the best predictor for this branch in the past
  - Use a 2-bit saturating counter for each branch: increase for one predictor, decrease for the other
    - e.g. if the local has been the best predictor last time(s), increase, if the global, decrease

# Performance