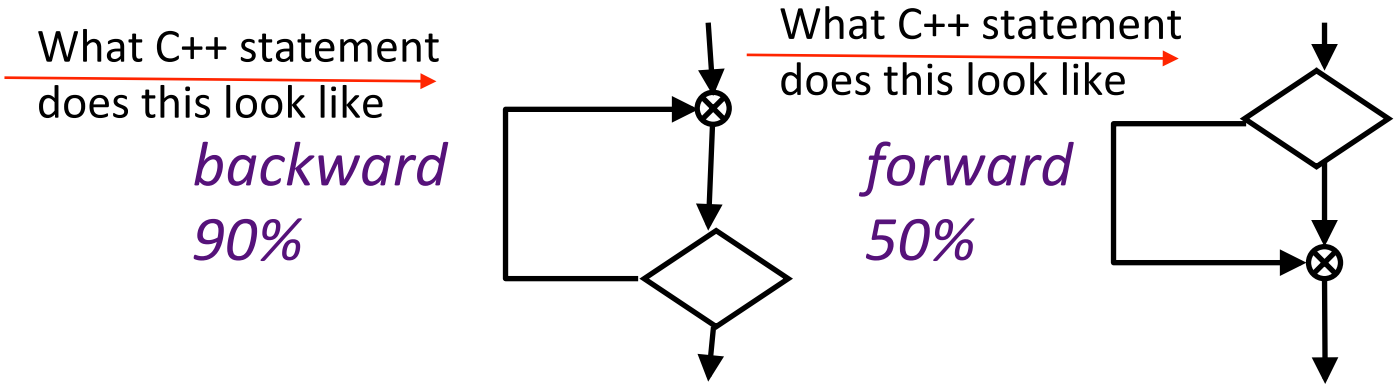
Branch Prediction

- *Motivation:*
  - Branch penalties limit performance of deeply pipelined processors
  - Modern branch predictors have high accuracy
  - (>95%) and can reduce branch penalties significantly

- *Required hardware support:*
  - *Prediction structures:*
    - Branch history tables, branch target buffers, etc.
  - *Mispredict recovery mechanisms:*
    - *Keep result computation separate from commit*
    - Kill instructions following branch in pipeline
    - Restore state to that following branch

Static Branch Prediction

Overall probability a branch is taken is ~60-70% but:



ISA can attach preferred direction semantics to branches, e.g., Motorola MC88110

bne0 *(preferred taken)* beq0 *(not taken)*

Dynamic Branch Prediction
learning based on past behavior

- Temporal correlation (time)
  - If I tell you that a certain branch was taken last time, does this help?
  - The way a branch resolves may be a good predictor of the way it will resolve at the next execution

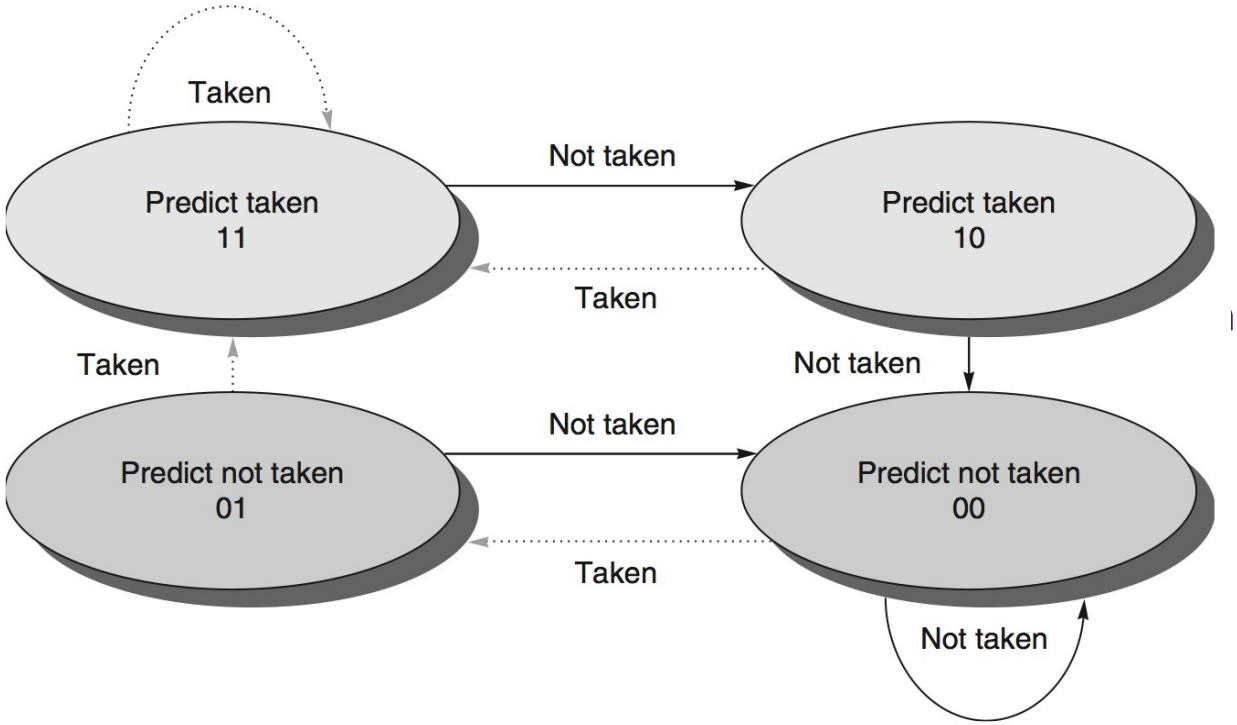
- Spatial correlation (space)
  - Several branches may resolve in a highly correlated manner
  - For instance, a preferred path of execution

Dynamic Branch Prediction

- 1-bit prediction scheme
  - Low-portion address as address for a one-bit flag for Taken or NotTaken historically
  - Simple
- 2-bit prediction
  - Miss twice to change

Branch Prediction Bits

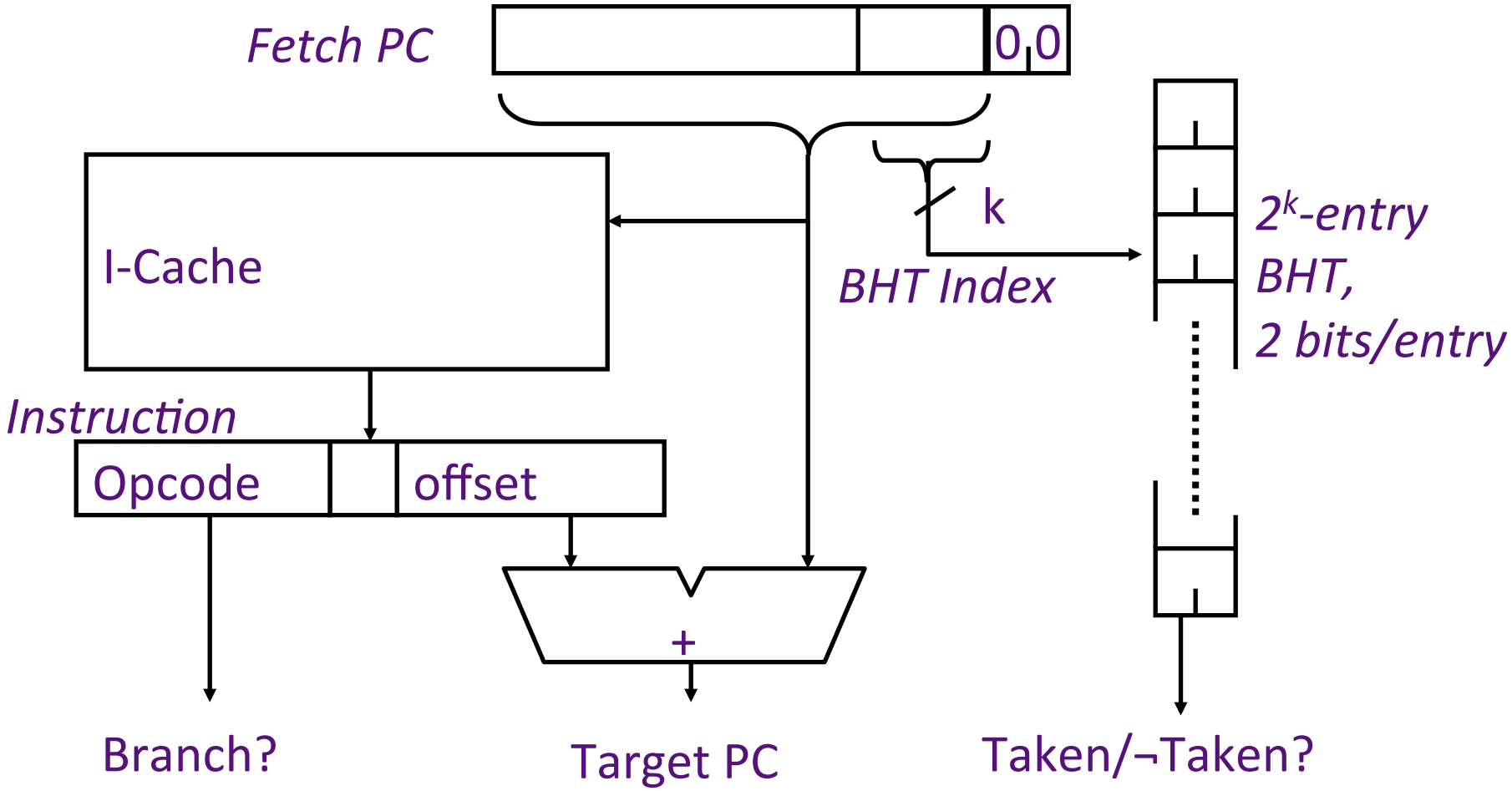
- Assume 2 BP bits per instruction
- Change the prediction after two consecutive mistakes!



BP state:

(*predict* take/¬take) x (*last prediction* right/wrong)

Branch History Table



4K-entry BHT, 2 bits/entry, ~80-90% correct predictions

Exploiting Spatial Correlation

Yeh and Patt, 1992

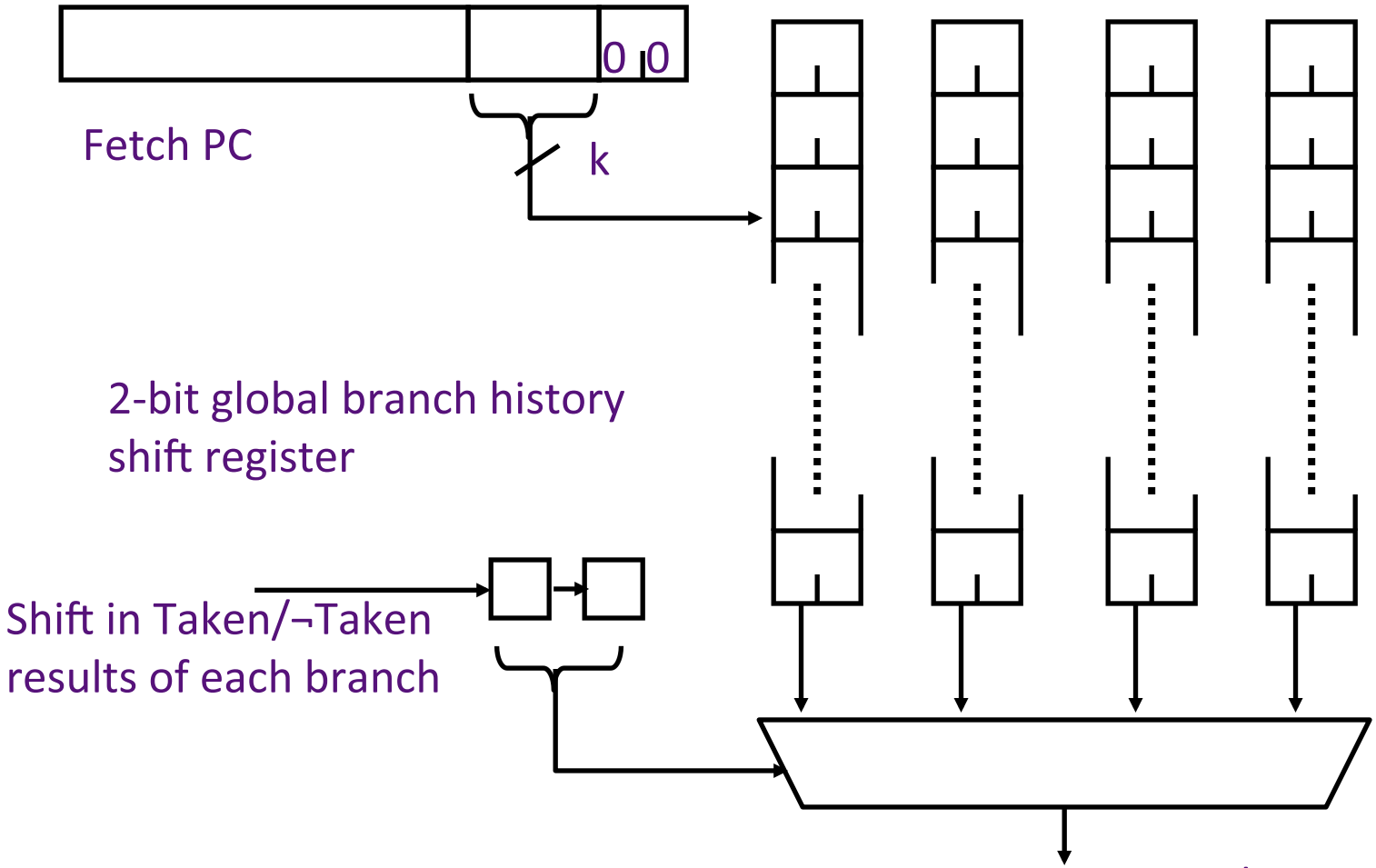
```
if (x[i] < 7) then
   y += 1;
if (x[i] < 5) then
   c -= 4;
```

If first condition false, second condition also false

History register, H, records the direction of the last N branches executed by the processor

Two-Level Branch Predictor

Pentium Pro uses the result from the last two branches to select one of the four sets of BHT bits (~95% correct)



Speculating Both Directions

- An alternative to branch prediction is to execute both directions of a branch speculatively
  - resource requirement is proportional to the number of concurrent speculative executions
  - only half the resources engage in useful work when both directions of a branch are executed speculatively
  - branch prediction takes less resources than speculative execution of both paths
- With accurate branch prediction, it is more cost effective to dedicate all resources to the predicted direction!
  - What would you choose with 80% accuracy?

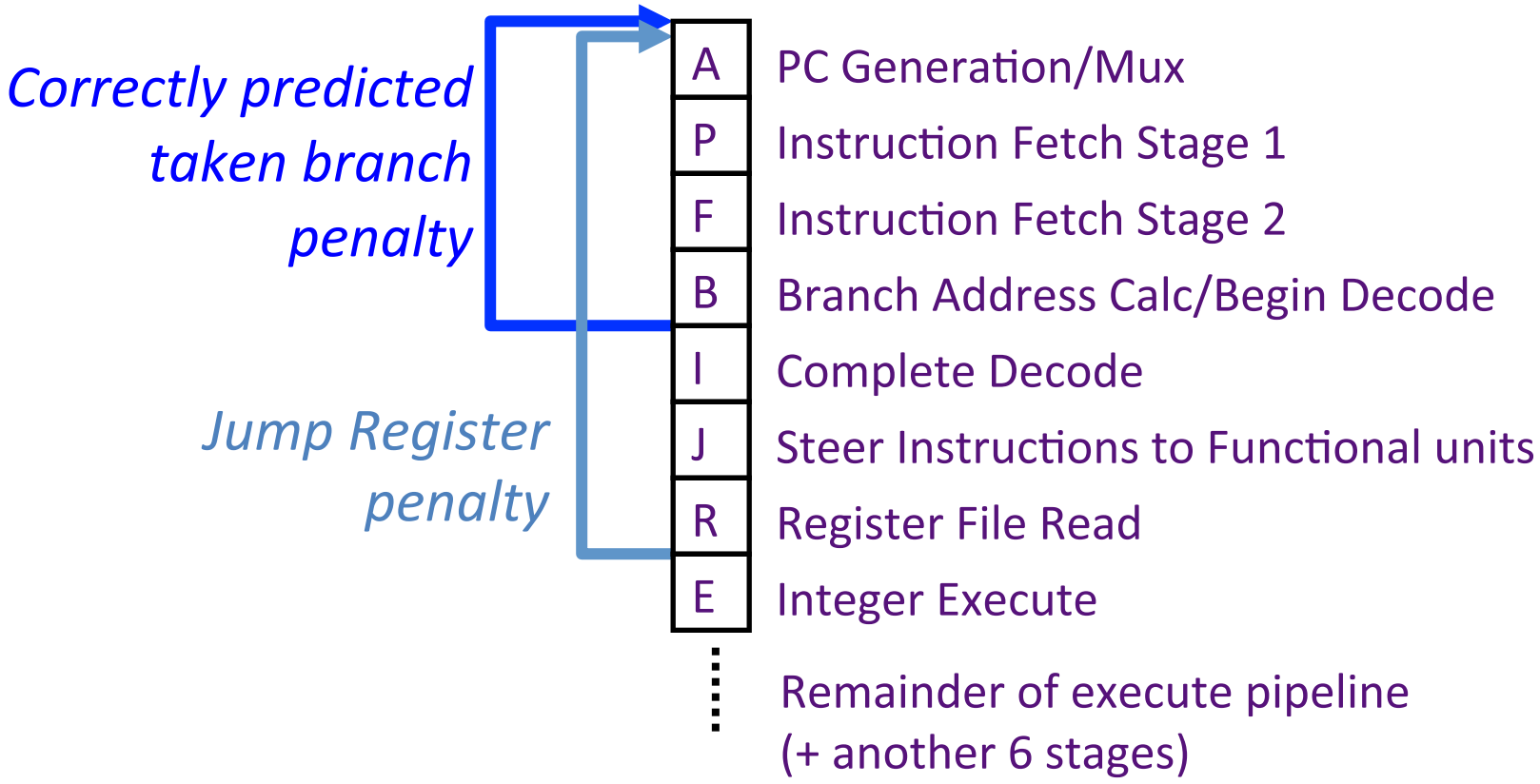
Are We Missing Something?

- Knowing whether a branch is taken or not is great, but what else do we need to know about it?

Branch target address

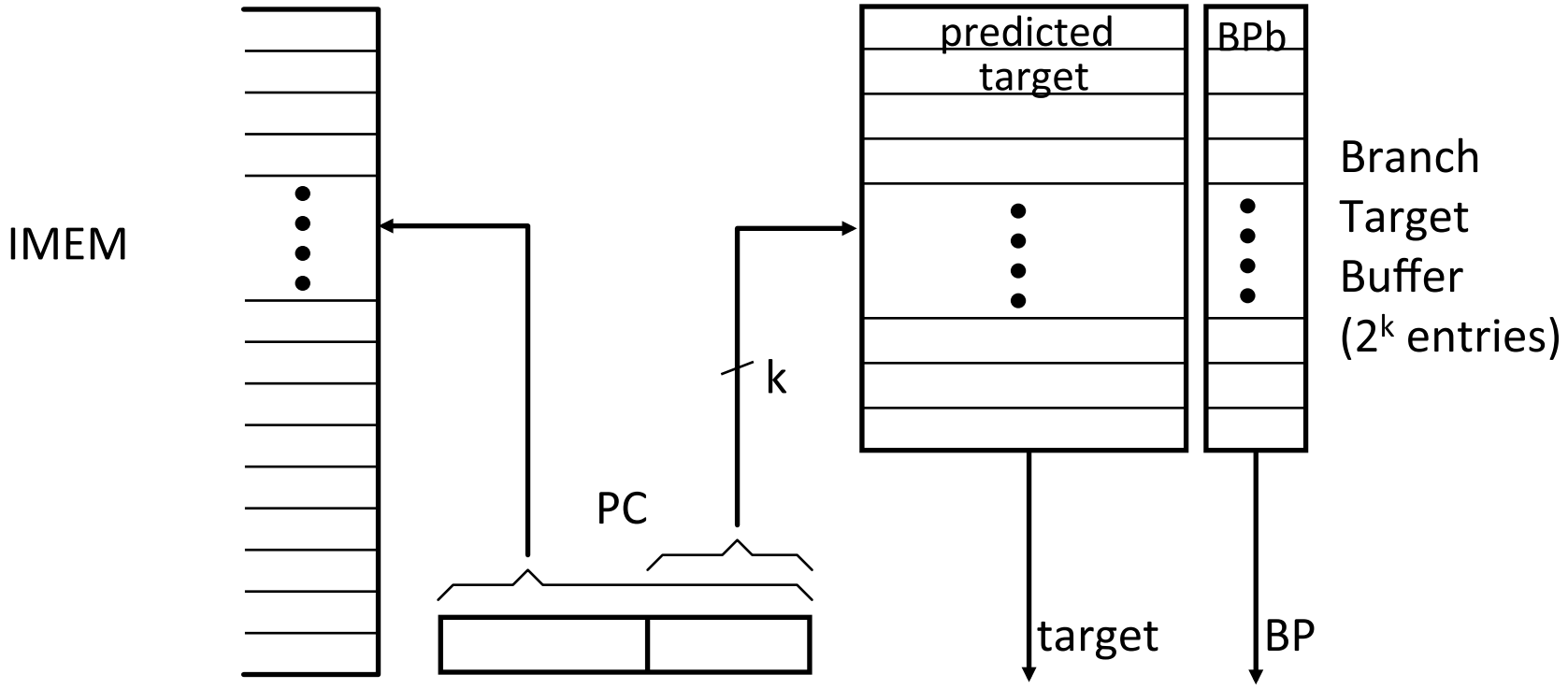
Limitations of BHTs

Only predicts branch direction. Therefore, cannot redirect fetch stream until after branch target is determined.



UltraSPARC-III fetch pipeline

Branch Target Buffer

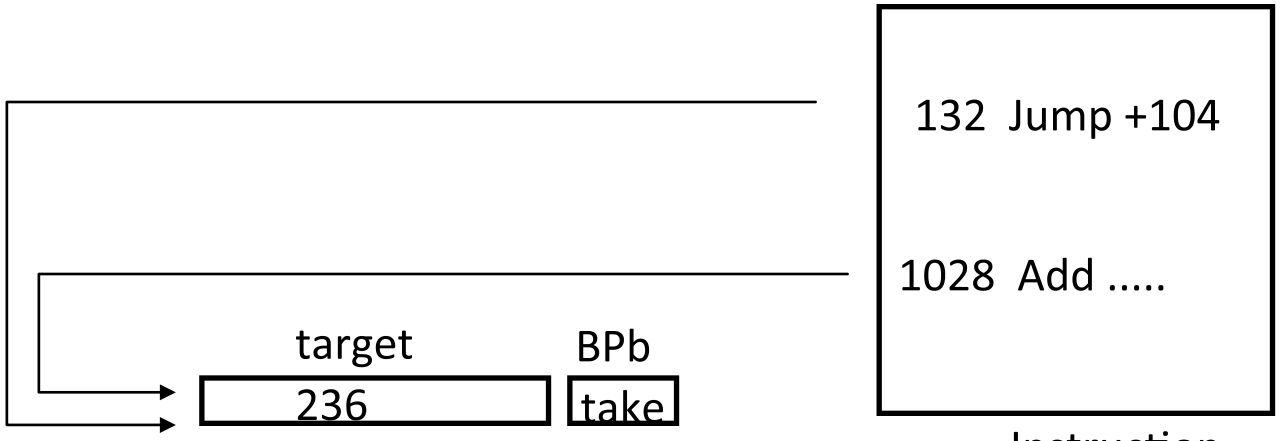


BP bits are stored with the predicted target address.

IF stage: *If (BP=taken) then nPC=target else nPC=PC+4*
Later: *check prediction, if wrong then kill the instruction and update BTB & BPb else update BPb*

Address Collisions (Mis-Prediction)

Assume a 128-entry BTB



What will be fetched after the instruction at 1028?

BTB prediction = 236

Correct target = 1032

=> *kill* PC=236 and *fetch* PC=1032

Is this a common occurrence?

BTB is only for Control Instructions

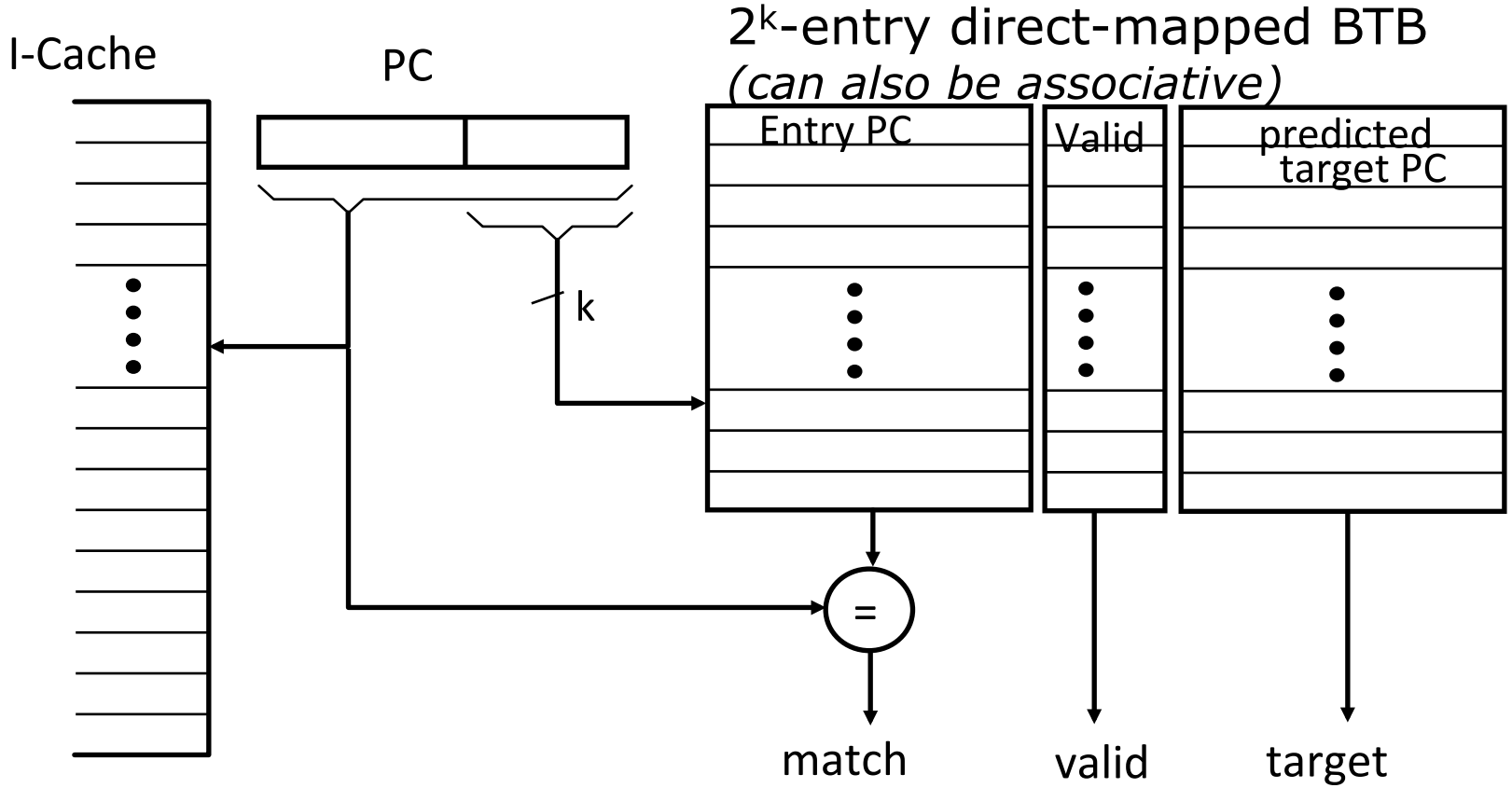
- Is even branch prediction fast enough to avoid bubbles?
- When do we index the BTB?
  - i.e., what state is the branch in, in order to avoid bubbles?

- **BTB contains useful information for branch and jump instructions only**
  => Do not update it for other instructions

- For all other instructions the next PC is PC+4 !

- *How to achieve this effect without decoding the instruction?*

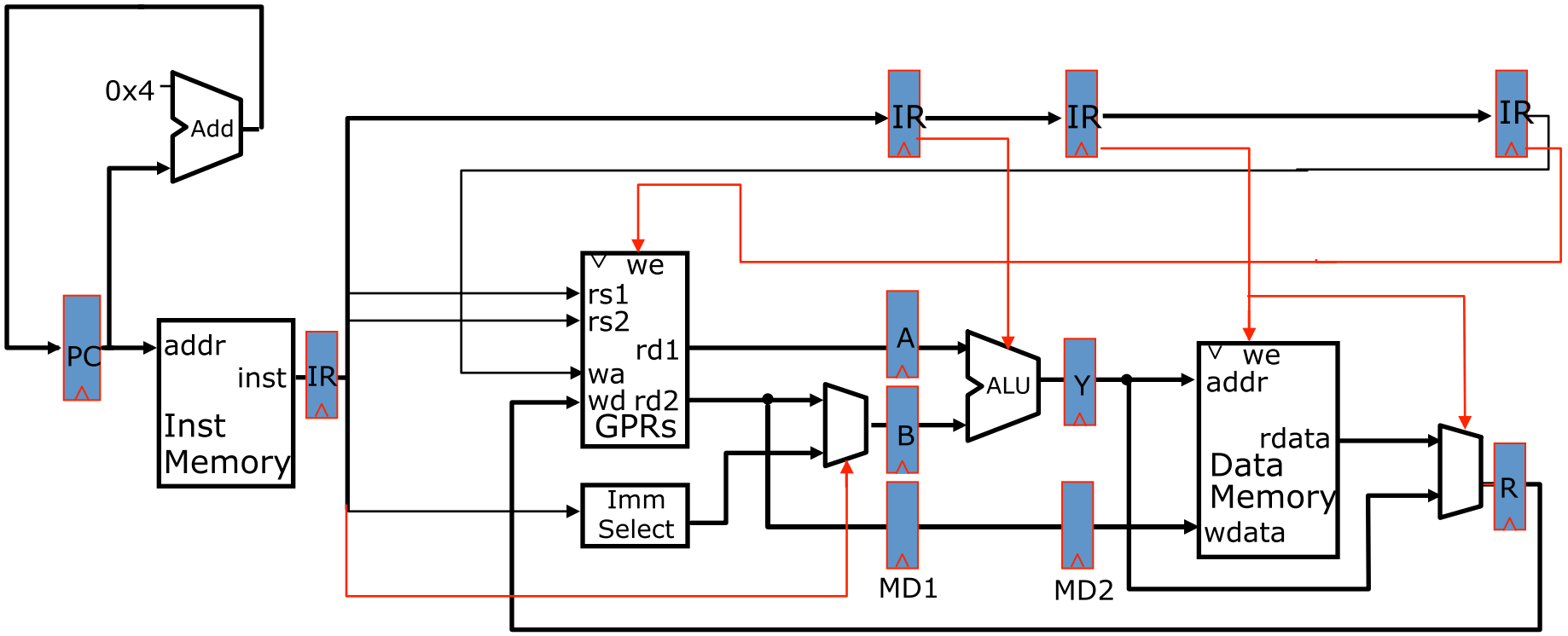
Branch Target Buffer (BTB)



- Keep both the branch PC and target PC in the BTB
- PC+4 is fetched if match fails
- Only *taken* branches and jumps held in BTB
- Next PC determined *before* branch fetched and decoded

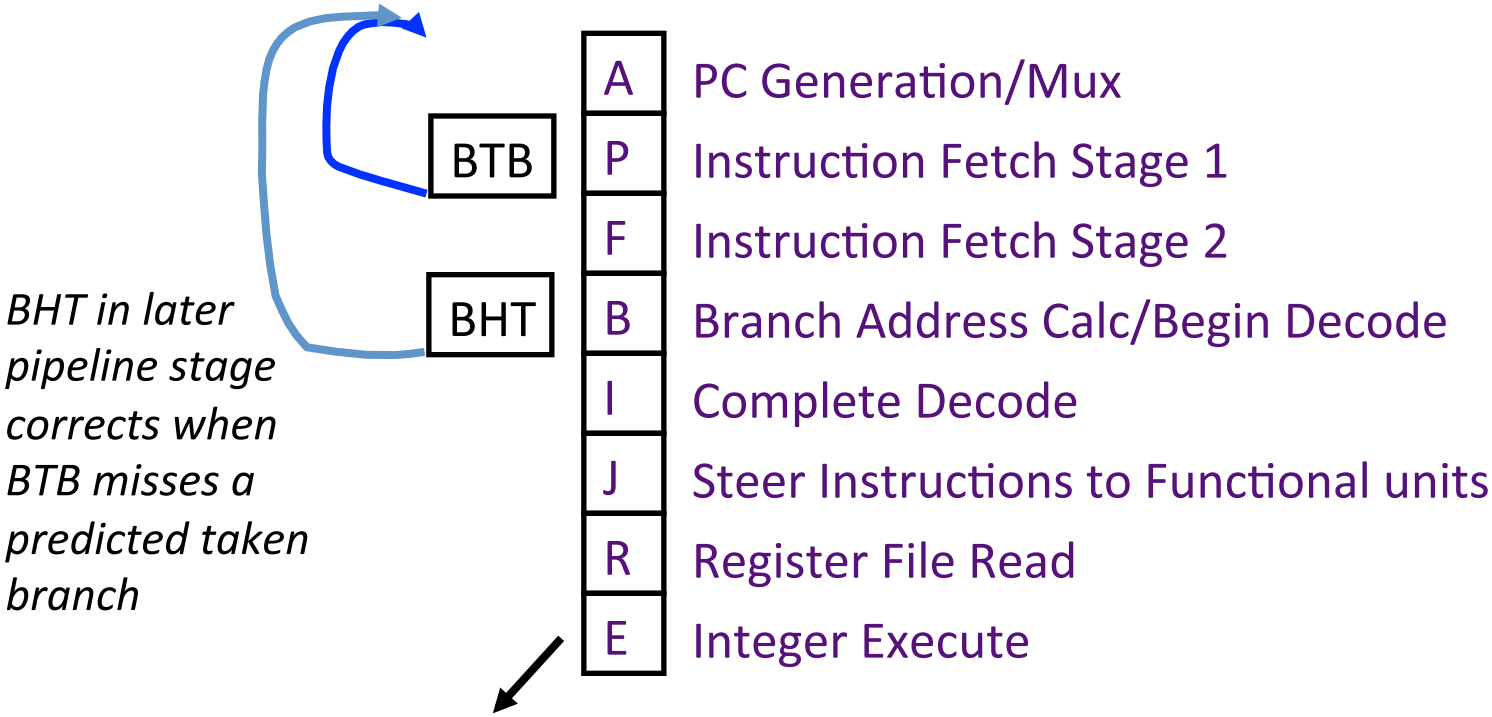
Are We Missing Something? (2)

- When do we update the BTB or BHT?



Combining BTB and BHT

- BTB entries are considerably more expensive than BHT, but can redirect fetches at earlier stage in pipeline and can accelerate indirect branches (JR)
- BHT can hold many more entries and is more accurate



BTB/BHT only updated after branch resolves in E stage

Uses of Jump Register (JR)

- Switch statements (jump to address of matching case)

  BTB works well if same case used repeatedly

- Dynamic function call (jump to run-time function address)

  BTB works well if same function usually called, (e.g., in C++ programming, when objects have same type in virtual function call)

- Subroutine returns (jump to return address)

  BTB works well if usually return to the same place

  ⇒ *Often one function called from many distinct call sites!*

How well does BTB work for each of these cases?

Subroutine Return Stack

Small structure to accelerate JR for subroutine returns, typically much more accurate than BTBs.

```
fa() { fb(); }
fb() { fc(); }
fc() { fd(); }
```