Branch Prediction and Speculative Execution

Arvind
Computer Science and Artificial Intelligence Laboratory
M.I.T.

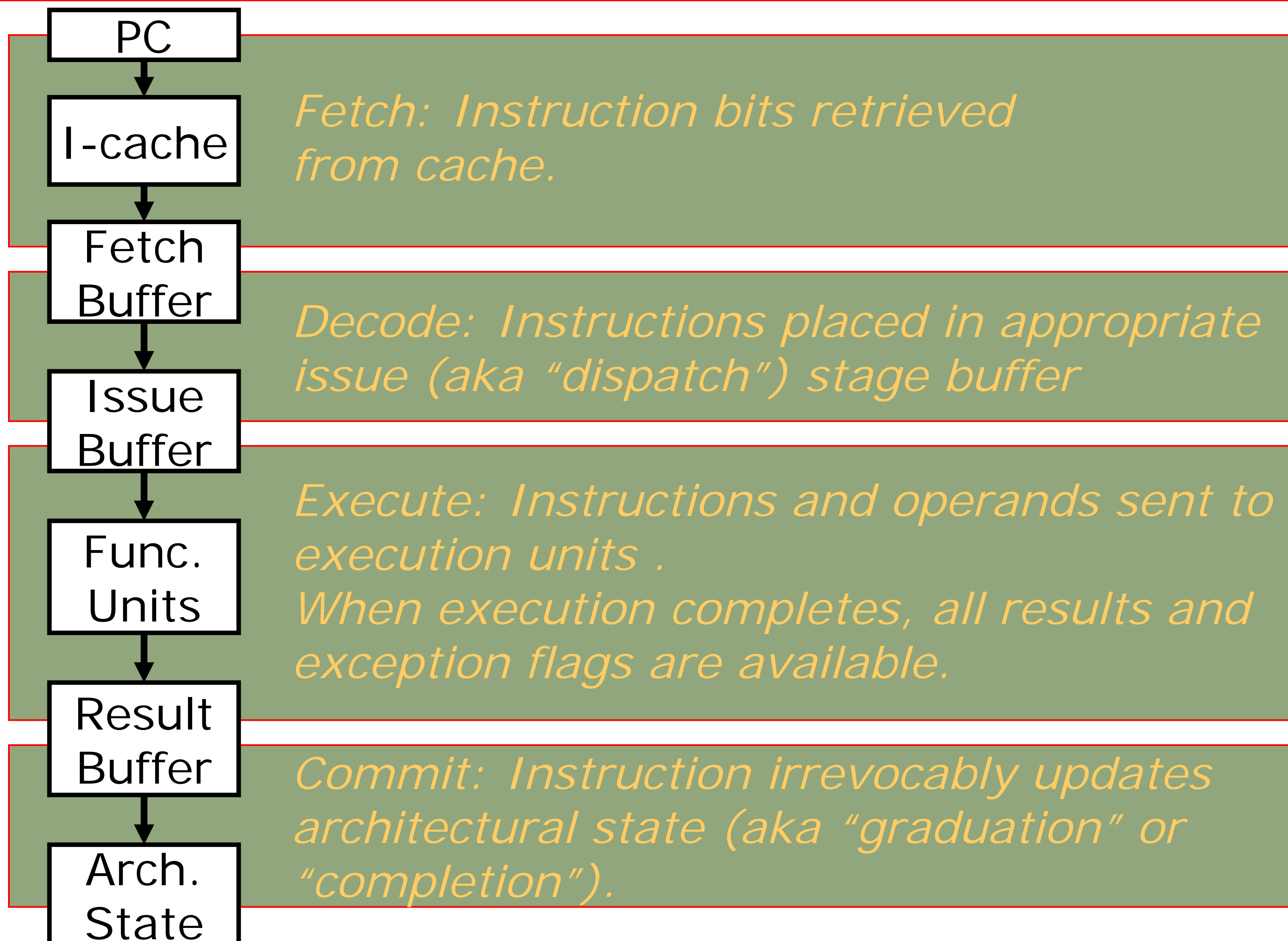
Based on the material prepared by Krste Asanovic and Arvind

Outline

- Control transfer penalty ←
- Branch prediction schemes
- Branch misprediction recovery schemes

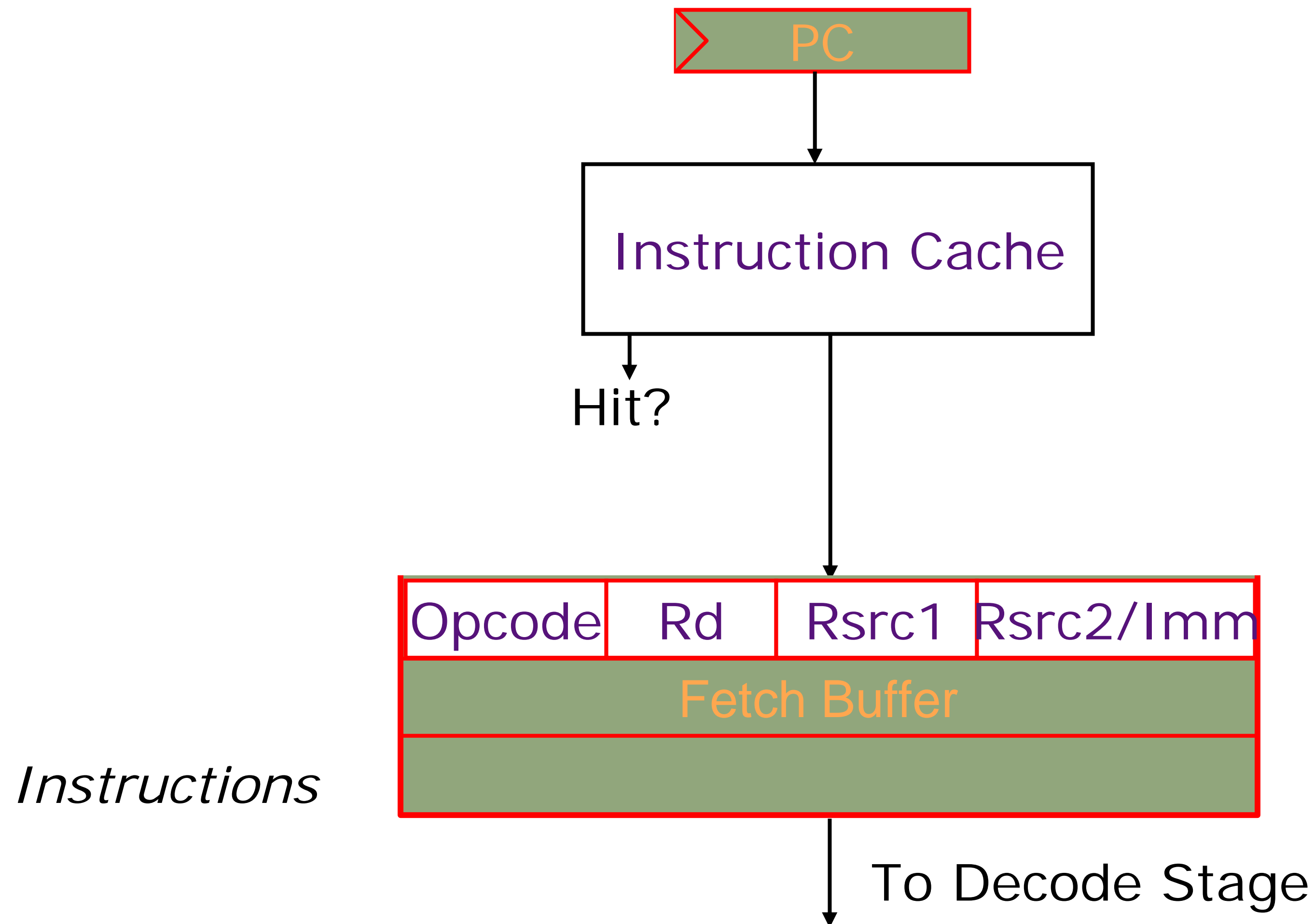
CSAIL

Phases of Instruction Execution



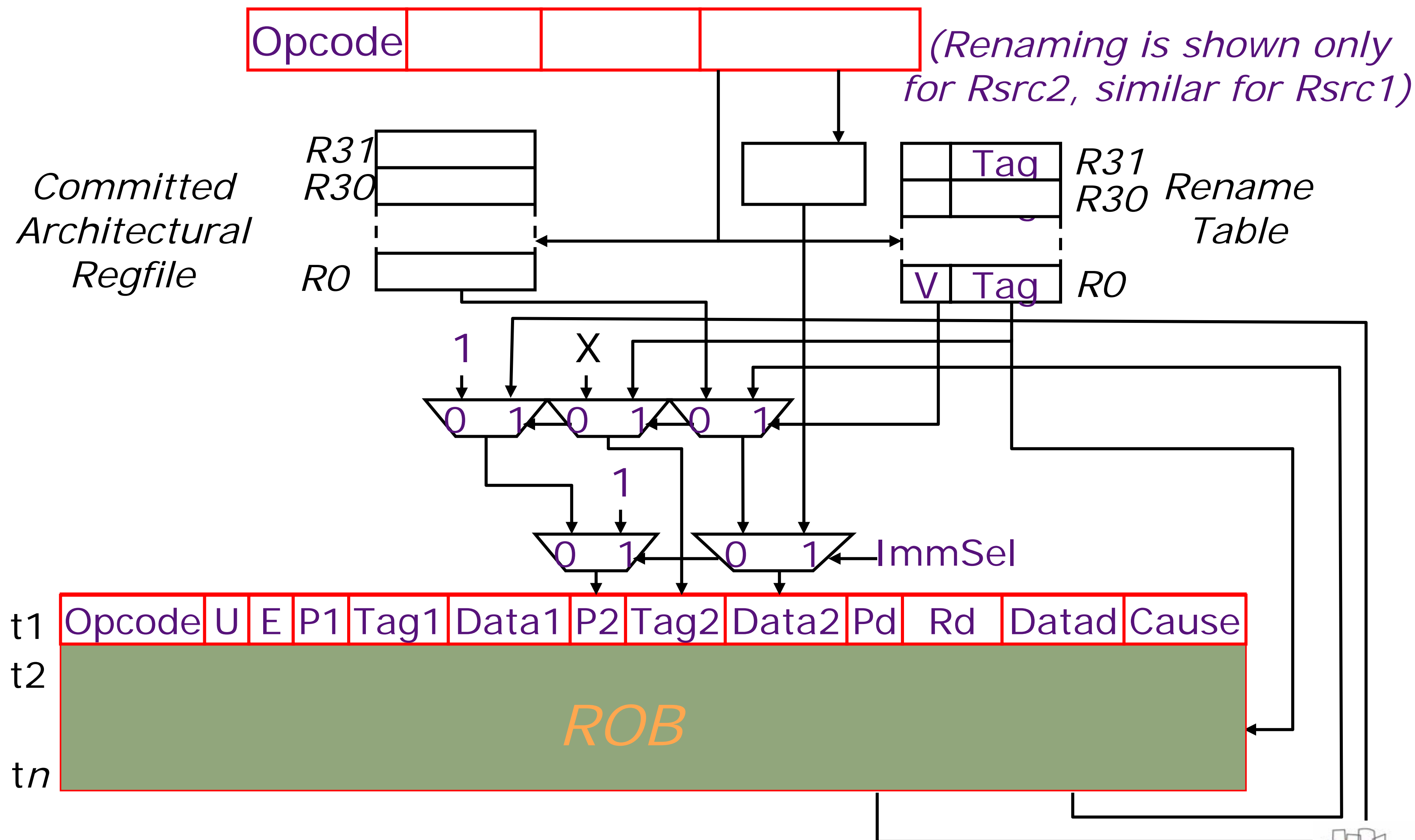
CSAIL

Fetch Stage



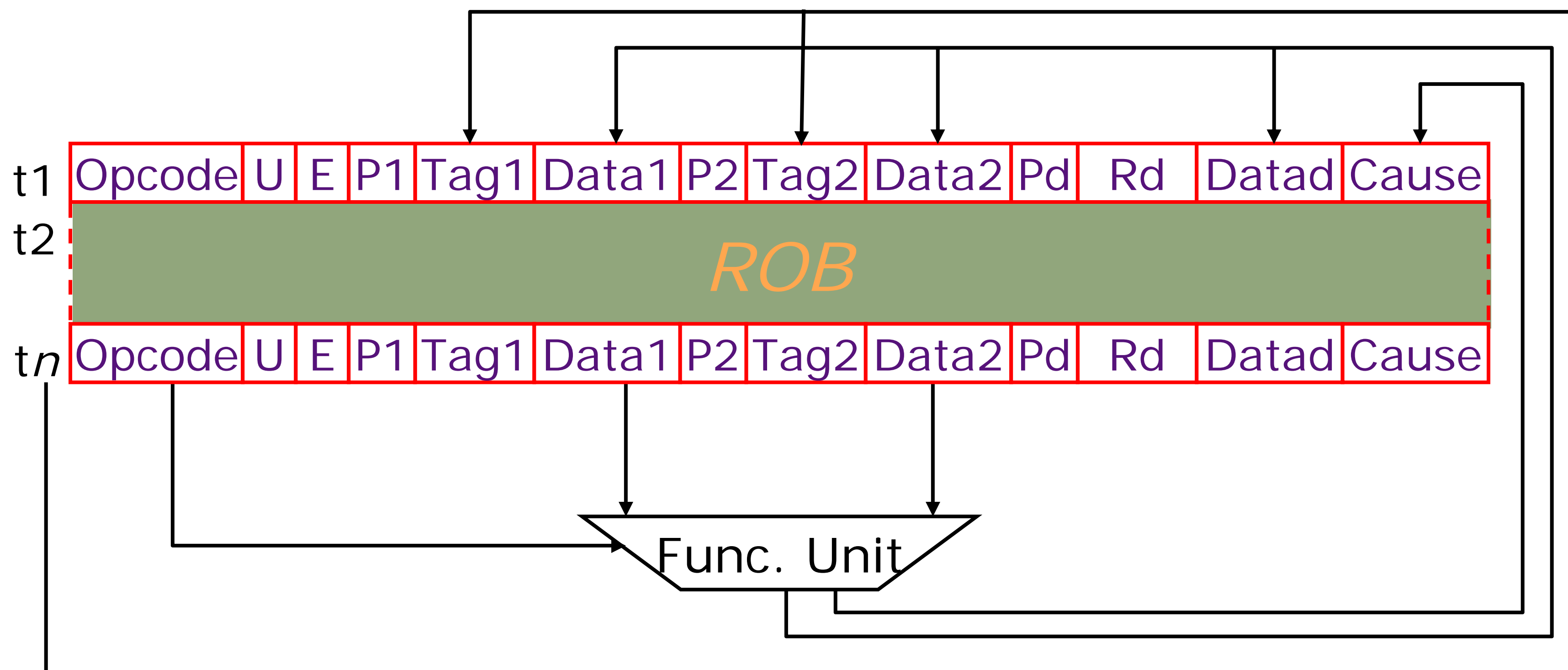
CSAIL

Decode & Rename Stage



Execute Stage

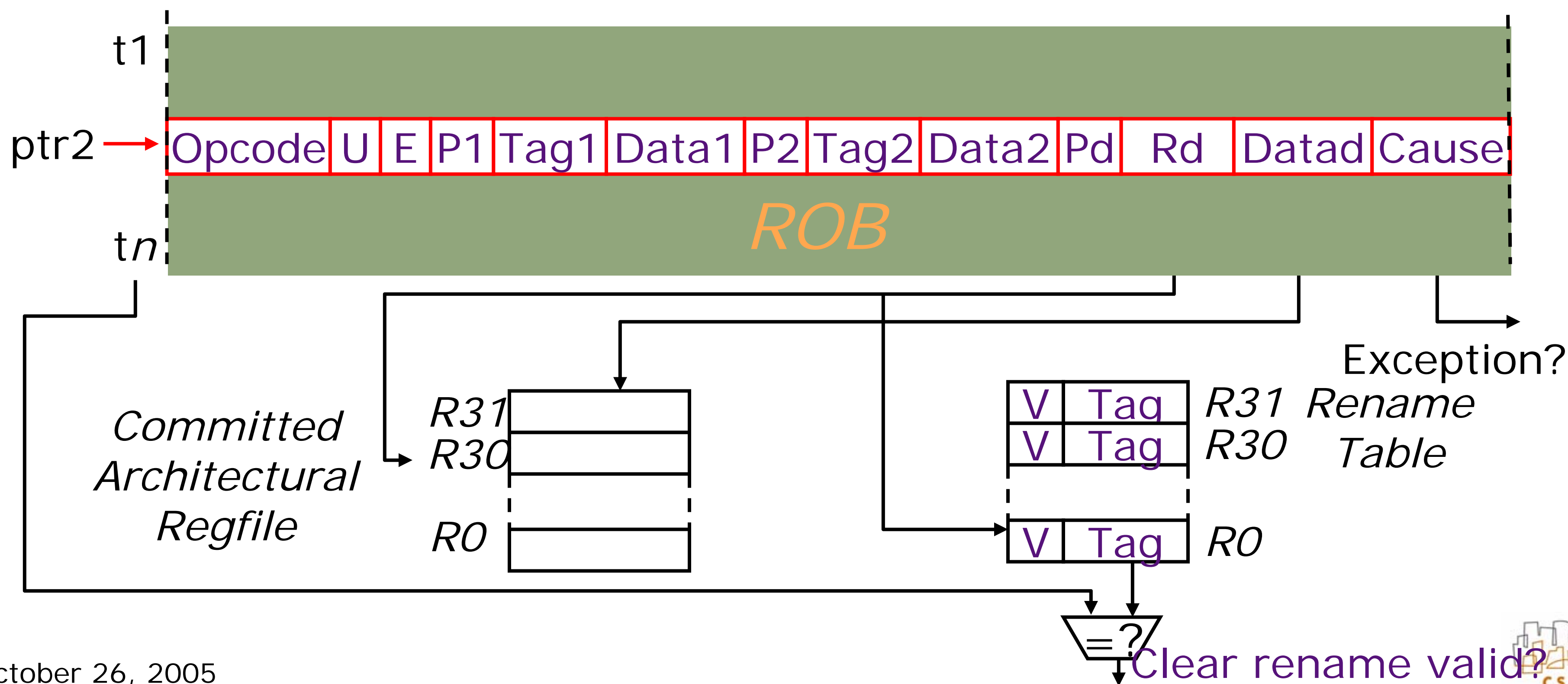
- Arbiter selects one ready instruction (P1=1 AND P2=1) to execute
- Instruction reads operands from ROB, executes, and broadcasts tag and result to waiting instructions in ROB. Also saves result and exception flags for commit.



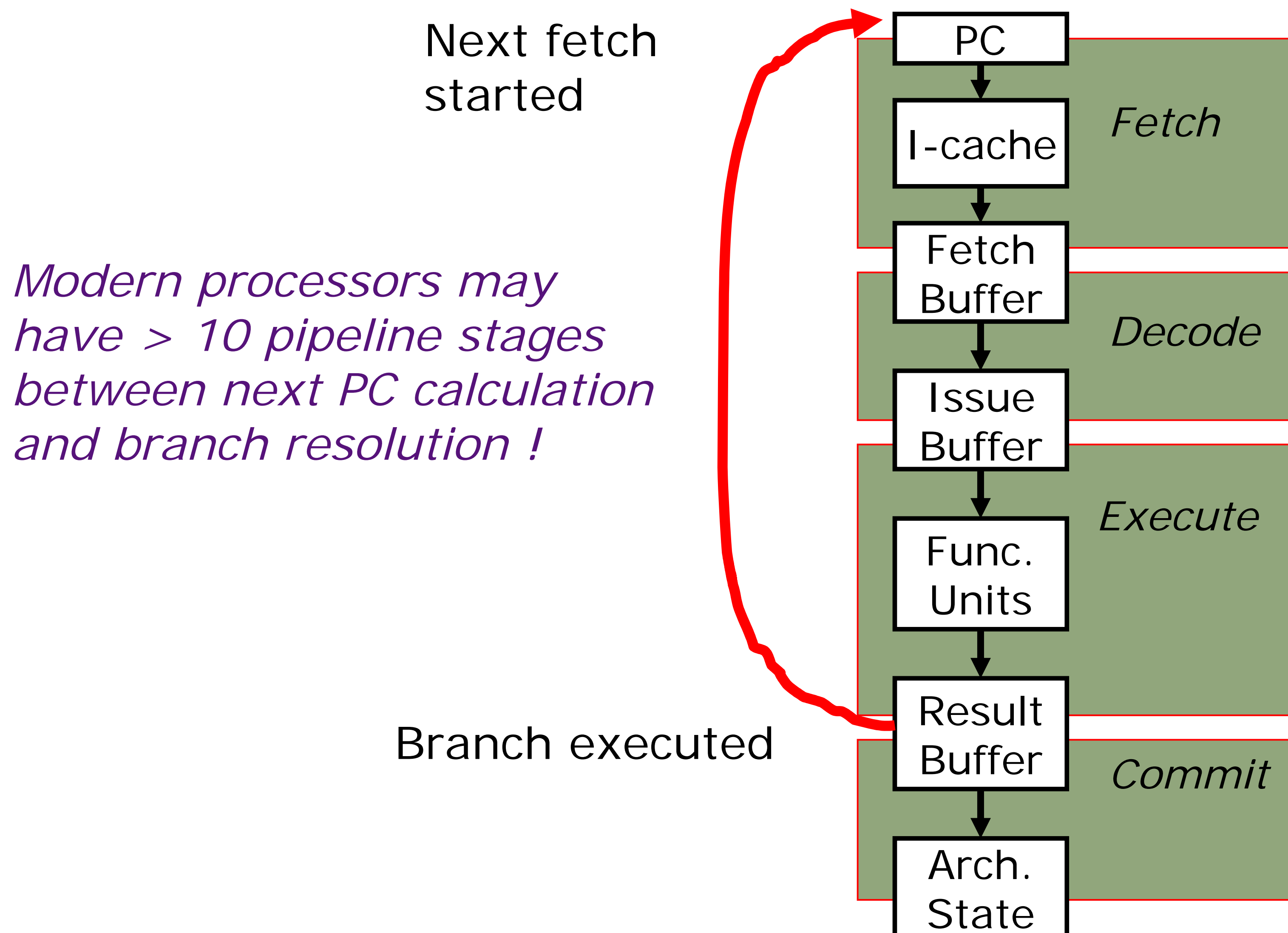
CSAIL

Commit Stage

- When instruction at ptr2 (commit point) has completed, write back result to architectural state and check for exceptions
- Check if rename table entry for architectural register written matches tag, if so, clear valid bit in rename table



Branch Penalty



Average Run-Length between Branches

Average dynamic instruction mix from SPEC92:

	SPECint92	SPECfp92
ALU	39 %	13 %
FPU Add		20 %
FPU Mult		13 %
load	26 %	23 %
store	9 %	9 %
branch	16 %	8 %
other	10 %	12 %

SPECint92: *compress, eqntott, espresso, gcc, li*
SPECfp92: *doduc, ear, hydro2d, mdijdp2, su2cor*

What is the average *run length* between branches

CSAIL

Reducing Control Transfer Penalties

Software solution

- *loop unrolling*
  Increases the run length
- *instruction scheduling*
  Compute the branch condition as early as possible (limited)

Hardware solution

- *delay slots*
  replaces pipeline bubbles with useful work (requires software cooperation)
- *branch prediction & speculative execution* of instructions beyond the branch

CSAIL

MIPS Branches and Jumps

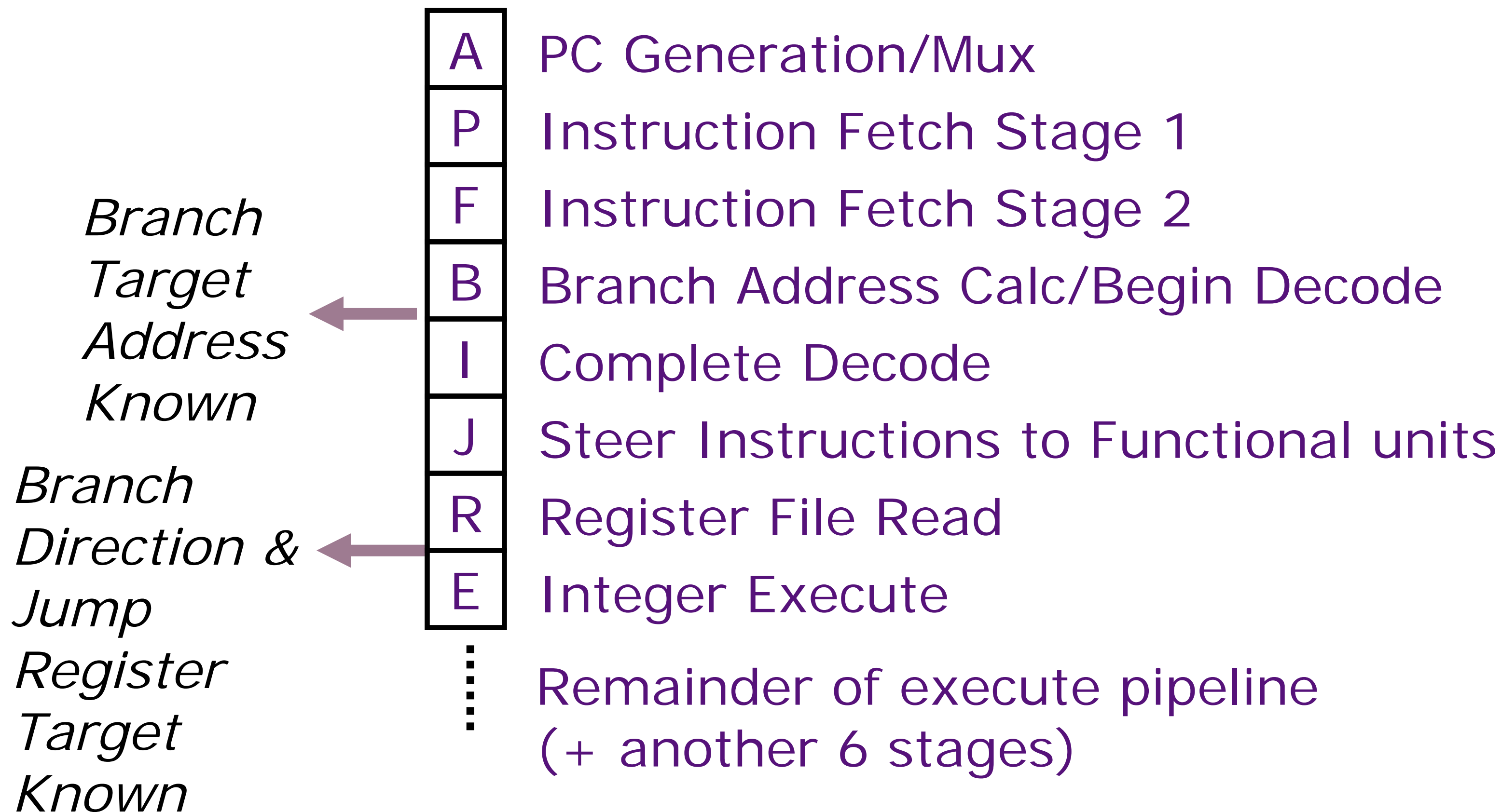
Need to know (or guess) both target address and whether the branch/jump is taken or not

Instruction	*Taken known?*	*Target known?*
BEQZ/BNEZ	After Reg. Fetch	After Inst. Fetch
J	Always Taken	After Inst. Fetch
JR	Always Taken	After Reg. Fetch

CSAIL

Branch Penalties in Modern Pipelines

UltraSPARC-III instruction fetch pipeline stages
(in-order issue, 4-way superscalar, 750MHz, 2000)



Outline

- Control transfer penalty
- Branch prediction schemes ←
- Branch misprediction recovery schemes

CSAIL

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.

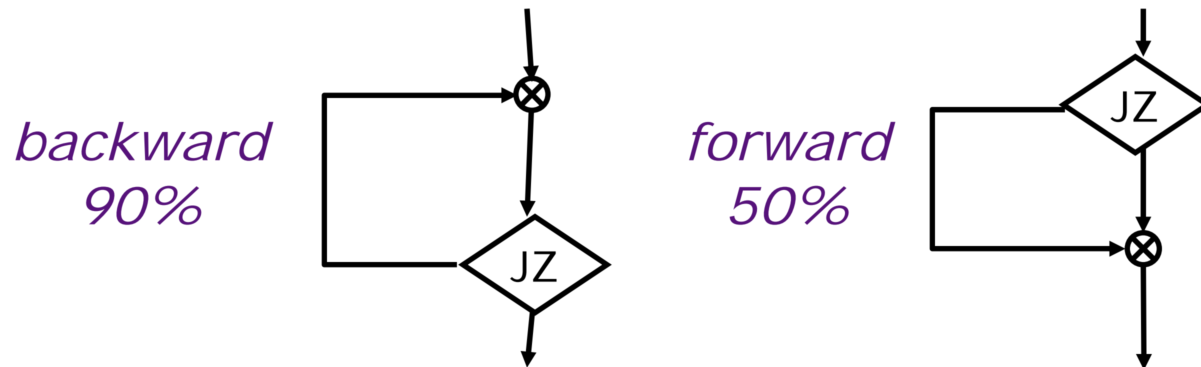
Mispredict recovery mechanisms:

- In-order machines: kill instructions following branch in pipeline
- Out-of-order machines: shadow registers and memory buffers for each speculated branch

CSAIL

Static Branch Prediction

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



ISA can attach additional semantics to branches about *preferred direction,* e.g., Motorola MC88110
bne0 *(preferred taken)* beq0 *(not taken)*

ISA can allow arbitrary choice of statically predicted direction (HP PA-RISC, Intel IA-64)

CSAIL

Dynamic Branch Prediction
learning based on past behavior

Temporal correlation

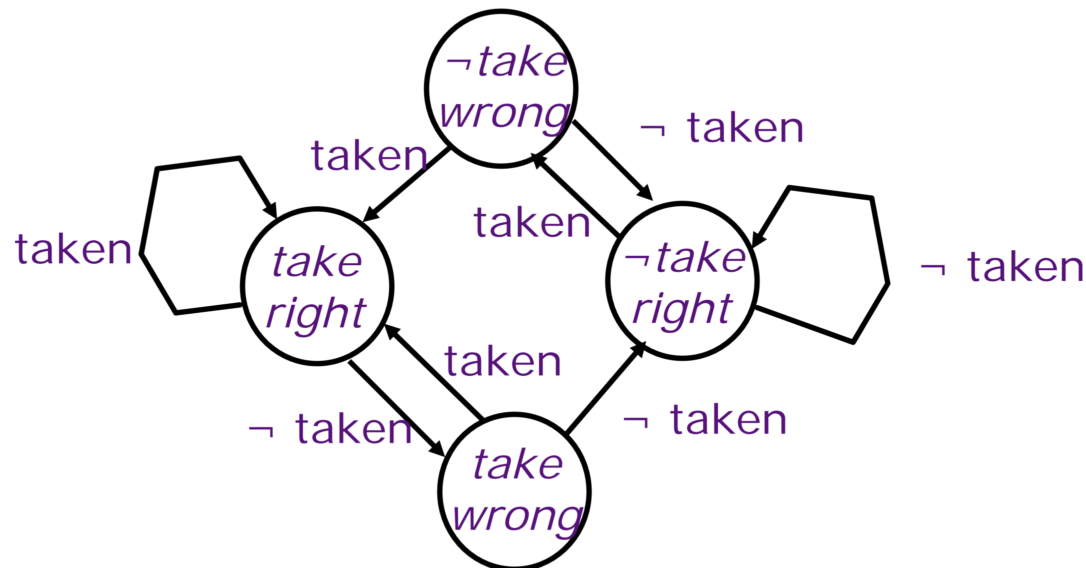
The way a branch resolves may be a good predictor of the way it will resolve at the next execution

Spatial correlation

Several branches may resolve in a highly correlated manner *(a preferred path of execution)*

Branch Prediction Bits

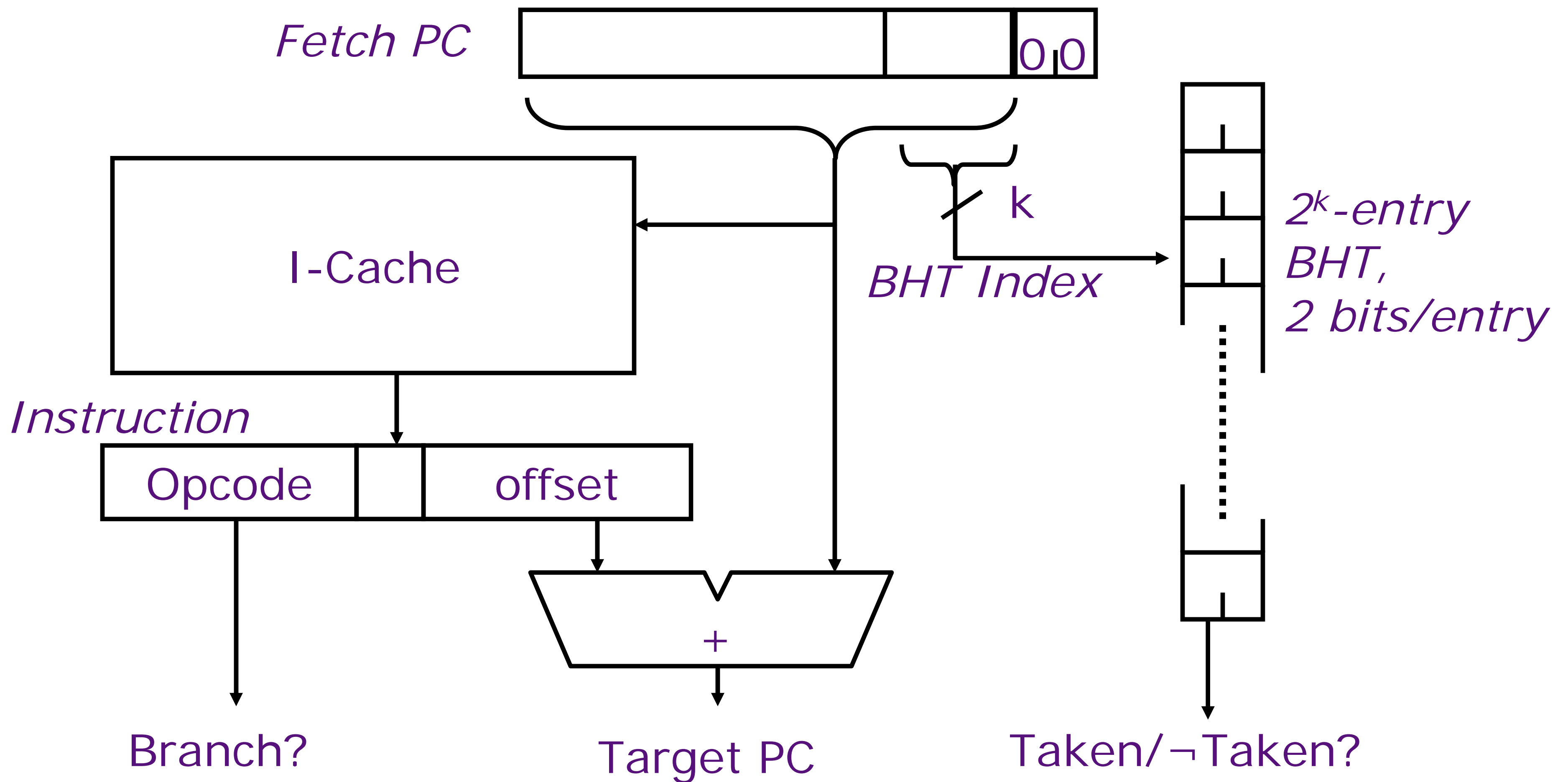
- Assume 2 BP bits per instruction
- Change the prediction after two consecutive mistakes!



BP state:

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

Branch History Table

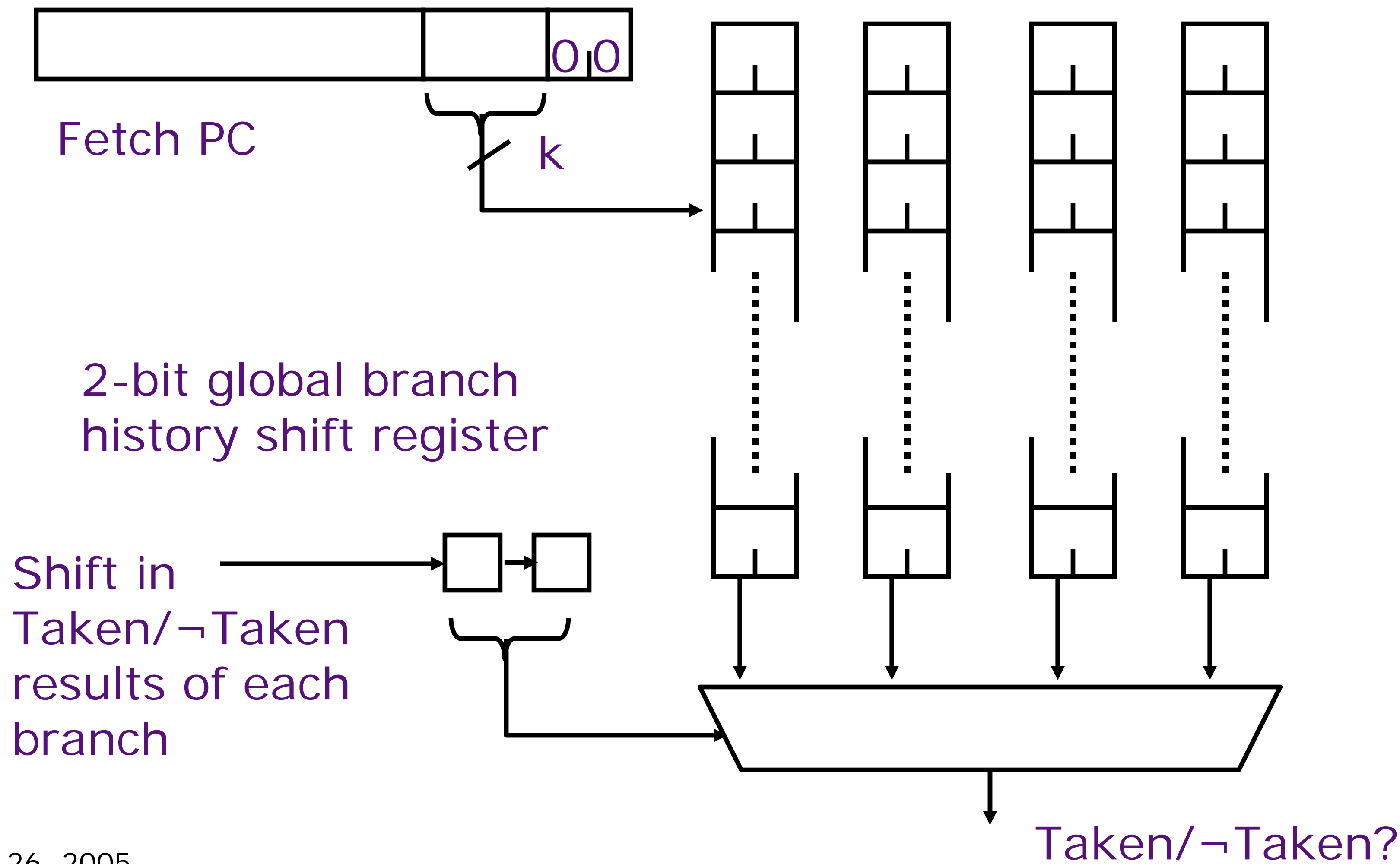


4K-entry BHT, 2 bits/entry, ~80-90% correct predictions

CSAIL

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)



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 bit: H records the direction of the last branch executed by the processor

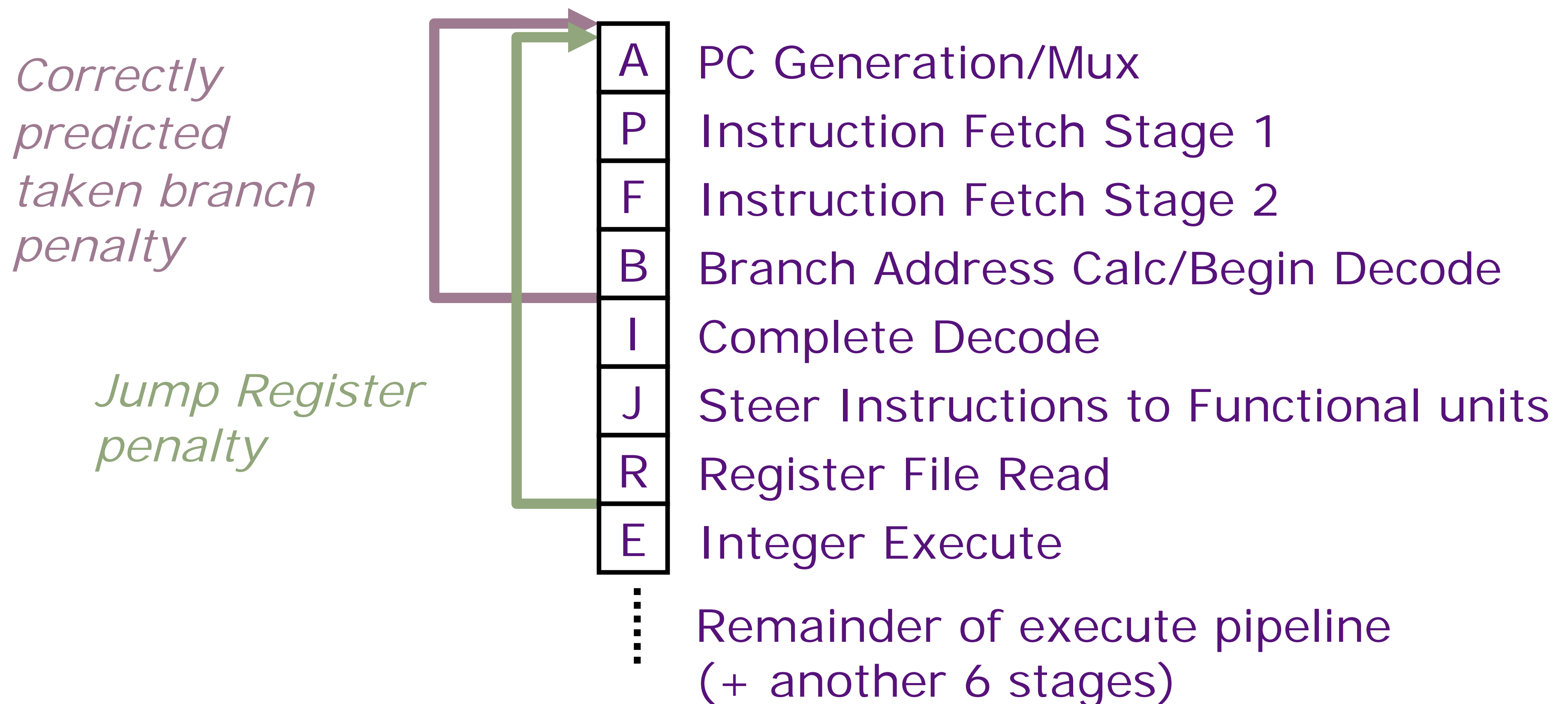
Two sets of BHT bits (BHT0 & BHT1) per branch instruction

H = 0 (not taken) $\Rightarrow$ consult BHT0
H = 1 (taken) $\Rightarrow$ consult BHT1

CSAIL

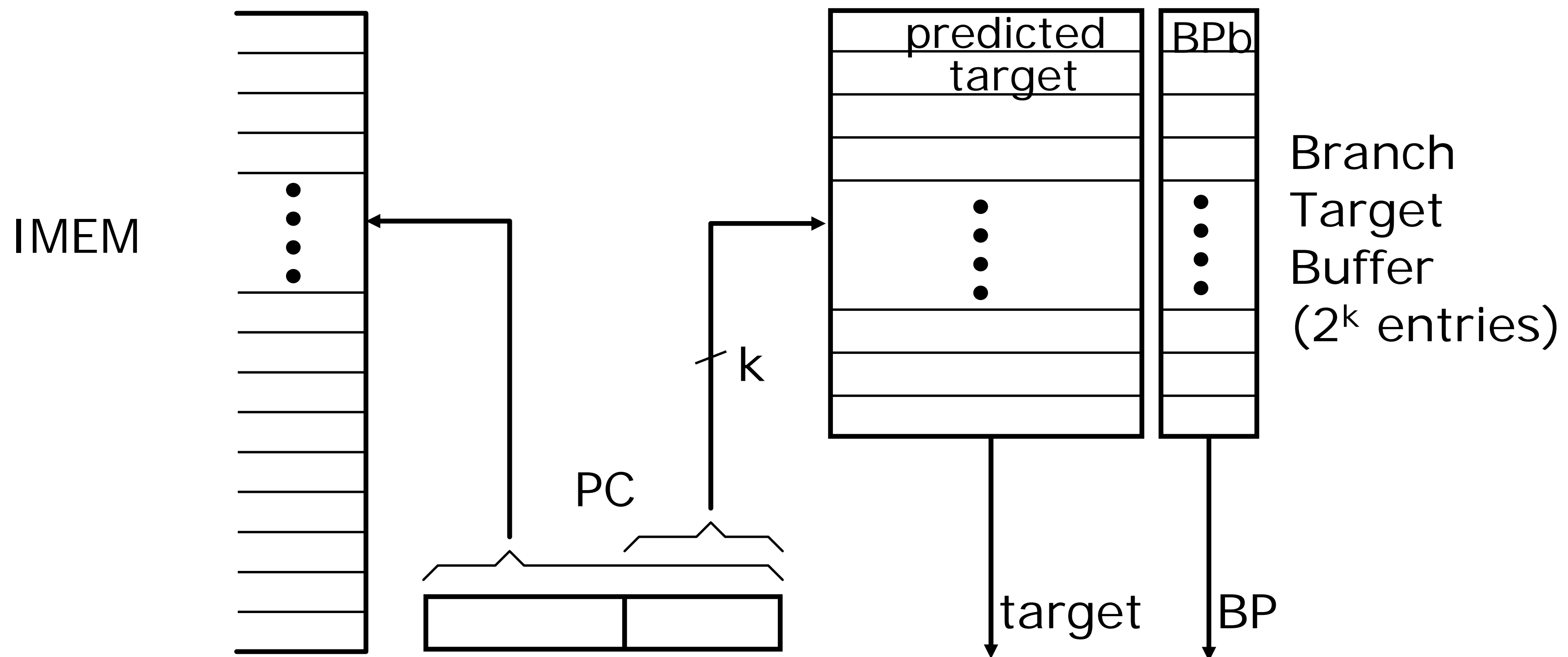
Limitations of BHTs

Cannot redirect fetch stream until after branch instruction is fetched and decoded, and target address determined



UltraSPARC-III fetch pipeline

Branch Target Buffer

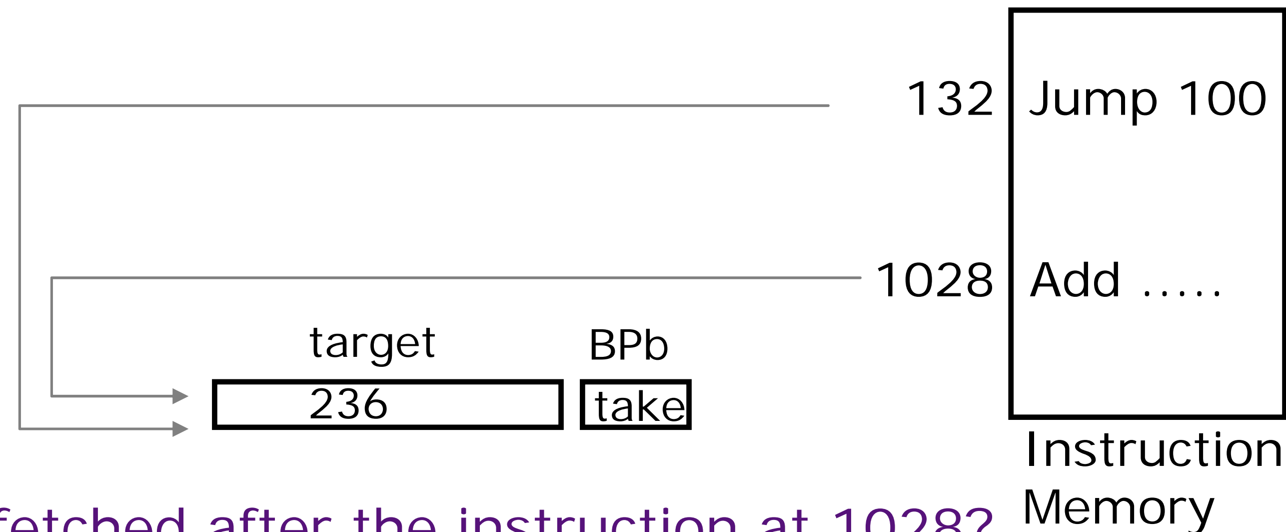


BP bits are stored with the predicted target address.

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

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?
Can we avoid these bubbles?

BTB should be for Control Transfer instructions only

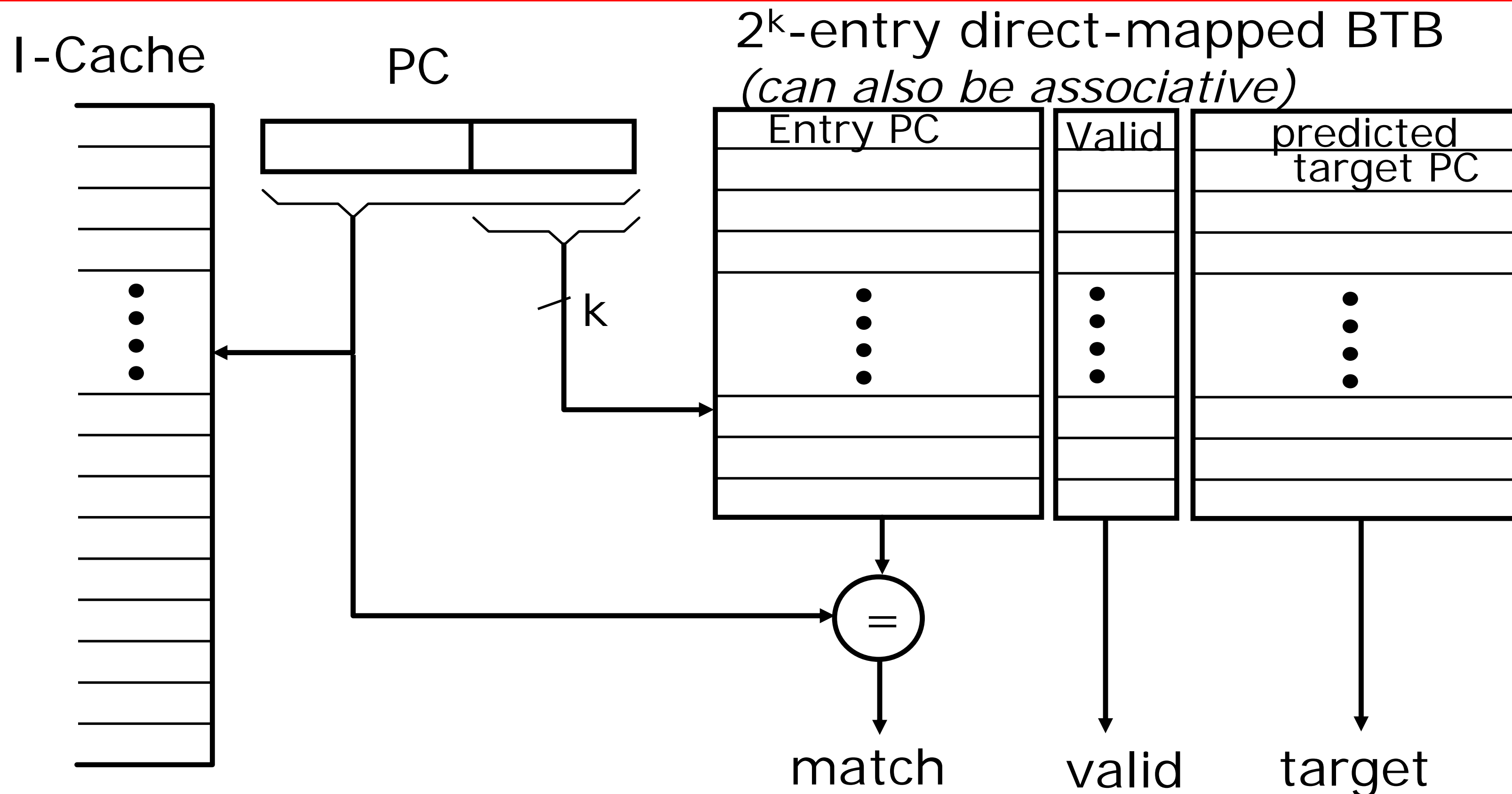
BTB contains useful information for branch and jump instructions only

$\Rightarrow$ it should not be updated for other instructions

For all other instructions the next PC is (PC)+4 !

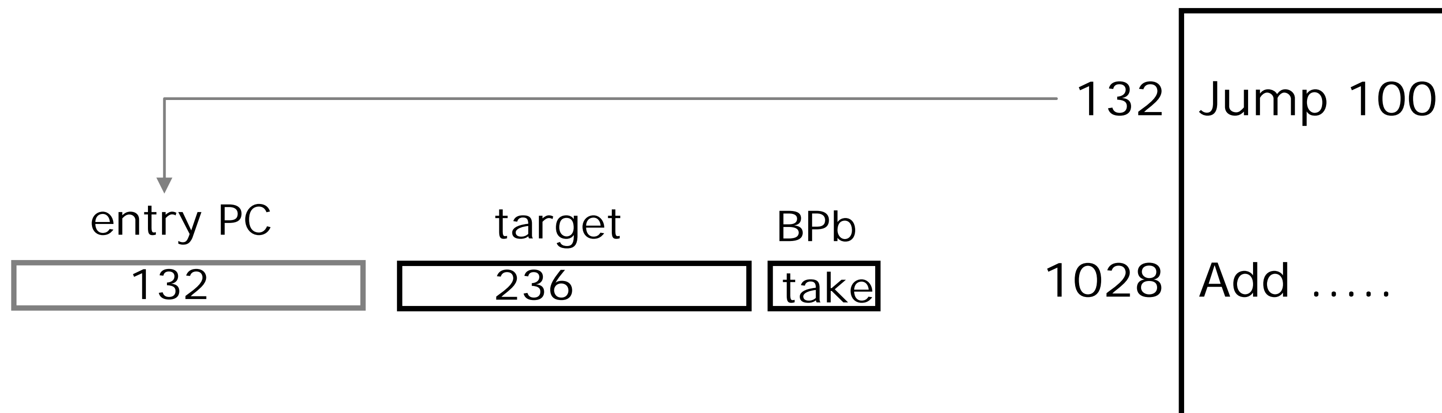
How to achieve this effect without decoding the instruction?

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

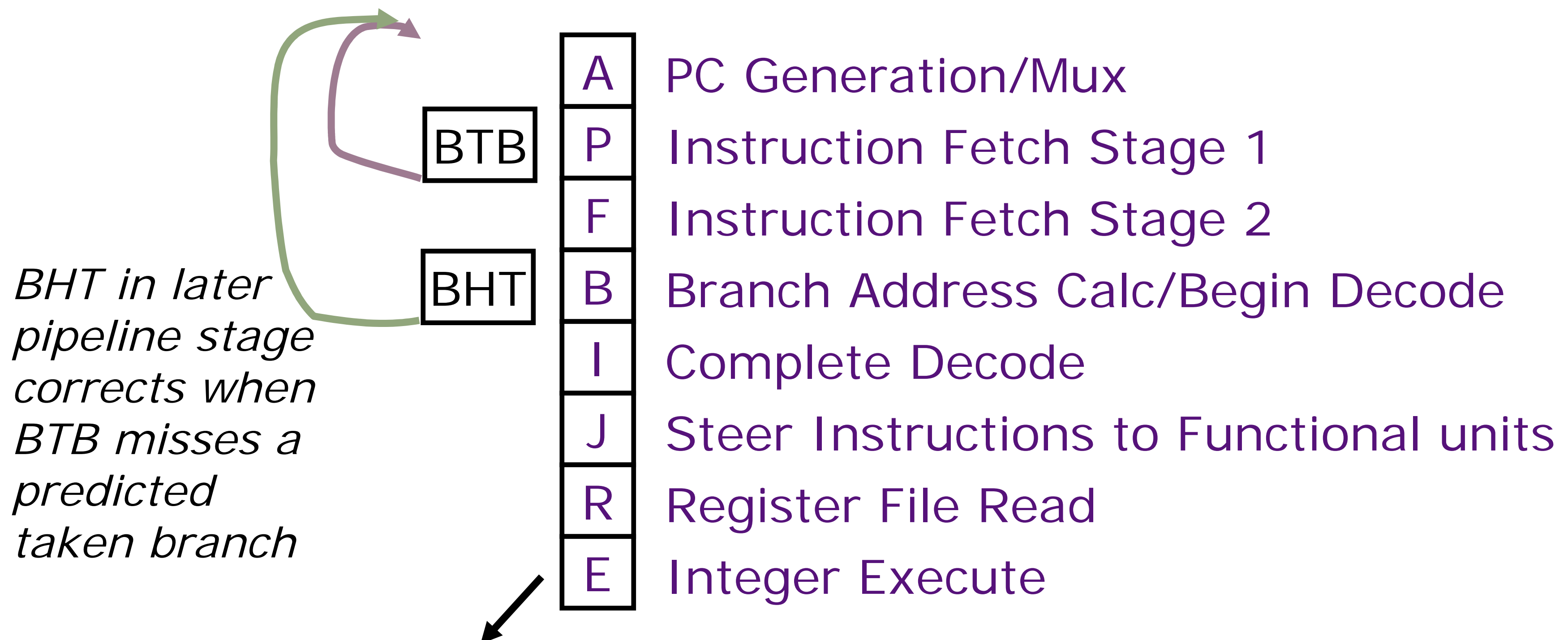
Consulting BTB Before Decoding



- The match for PC=1028 fails and 1028+4 is fetched
  ⇒ *eliminates false predictions after ALU instructions*
- BTB contains entries only for control transfer instructions
  ⇒ *more room to store branch targets*

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

CSAIL

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 different call sites!*

How well does BTB work for each of these cases?

CSAIL

Subroutine Return Stack

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

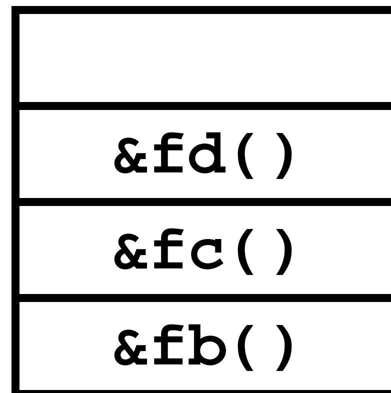
```
fa() { fb(); }

fb() { fc(); }

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

Push call address when function call executed

Pop return address when subroutine return decoded



k entries (typically k=8-16)

Outline

- Control transfer penalty
- Branch prediction schemes
- Branch misprediction recovery schemes ←

Five-minute break to stretch your legs

CSAIL

Mispredict Recovery

In-order execution machines:

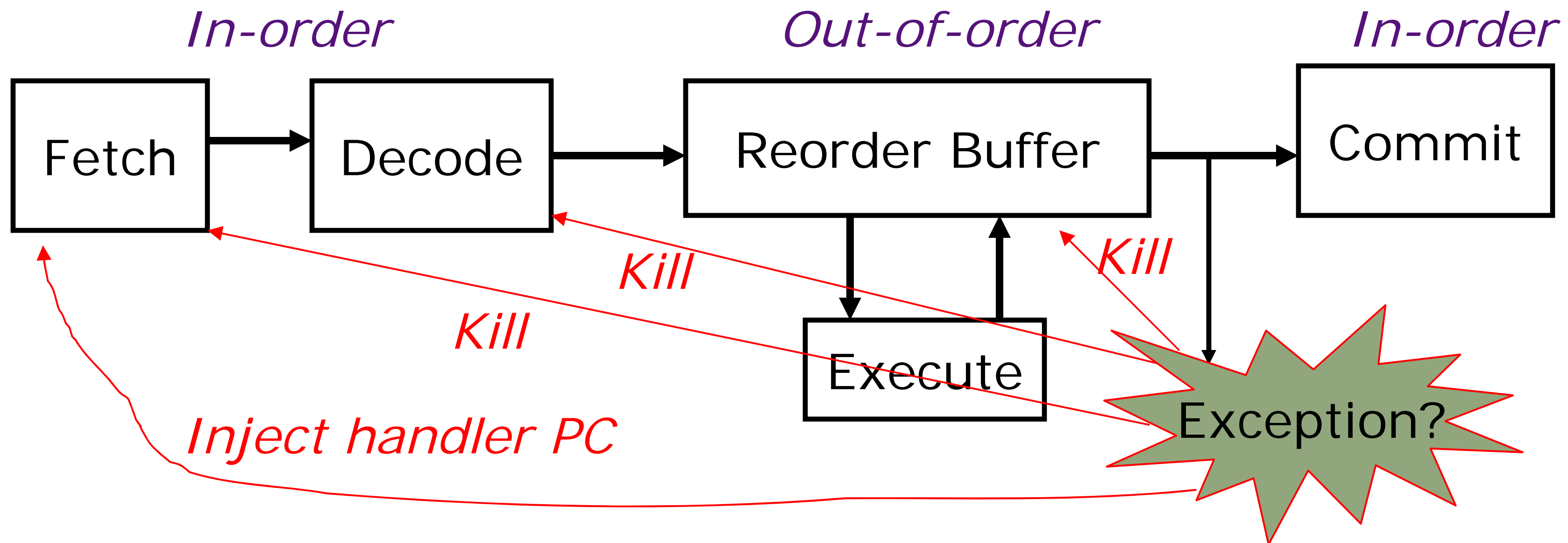
- Assume no instruction issued after branch can write-back before branch resolves
- Kill all instructions in pipeline behind mispredicted branch

Out-of-order execution?

- Multiple instructions following branch in program order can complete before branch resolves

CSAIL

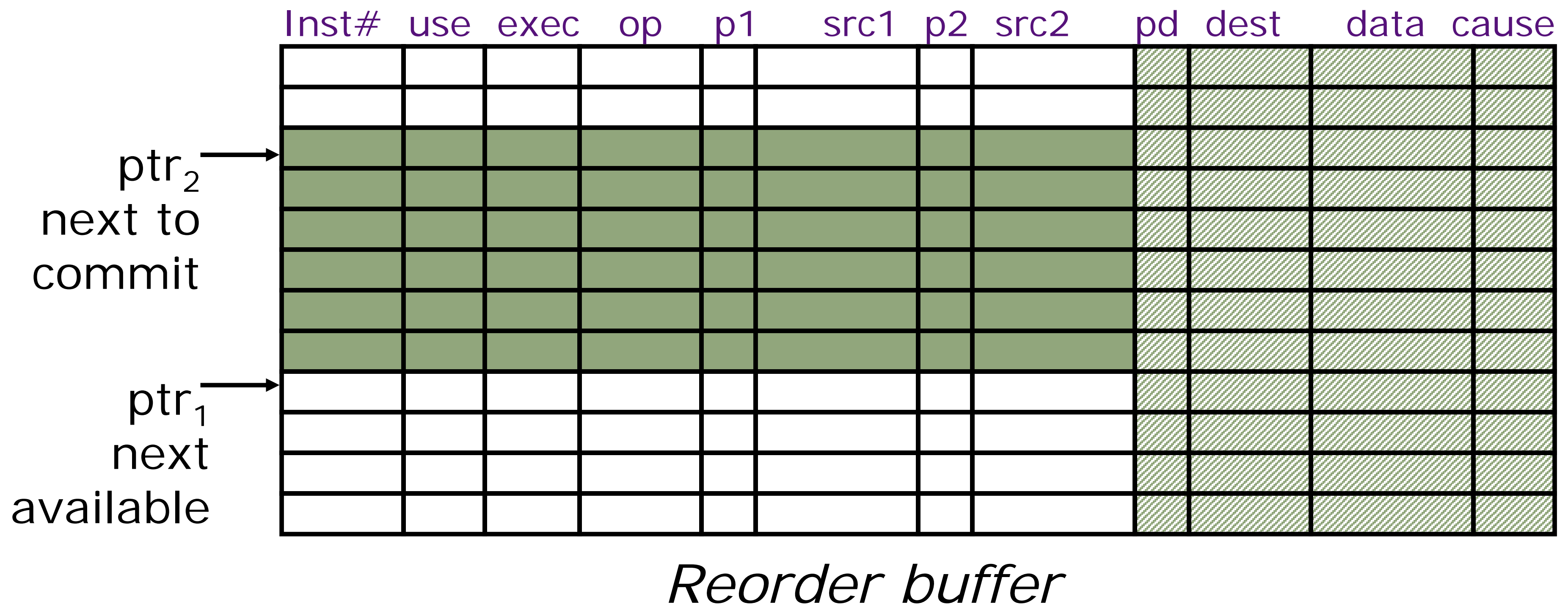
In-Order Commit for Precise Exceptions



- Instructions fetched and decoded into instruction reorder buffer in-order
- Execution is out-of-order (⇒ out-of-order completion)
- *Commit* (write-back to architectural state, i.e., regfile & memory, is in-order

Temporary storage needed in ROB to hold results before commit

Extensions for Precise Exceptions



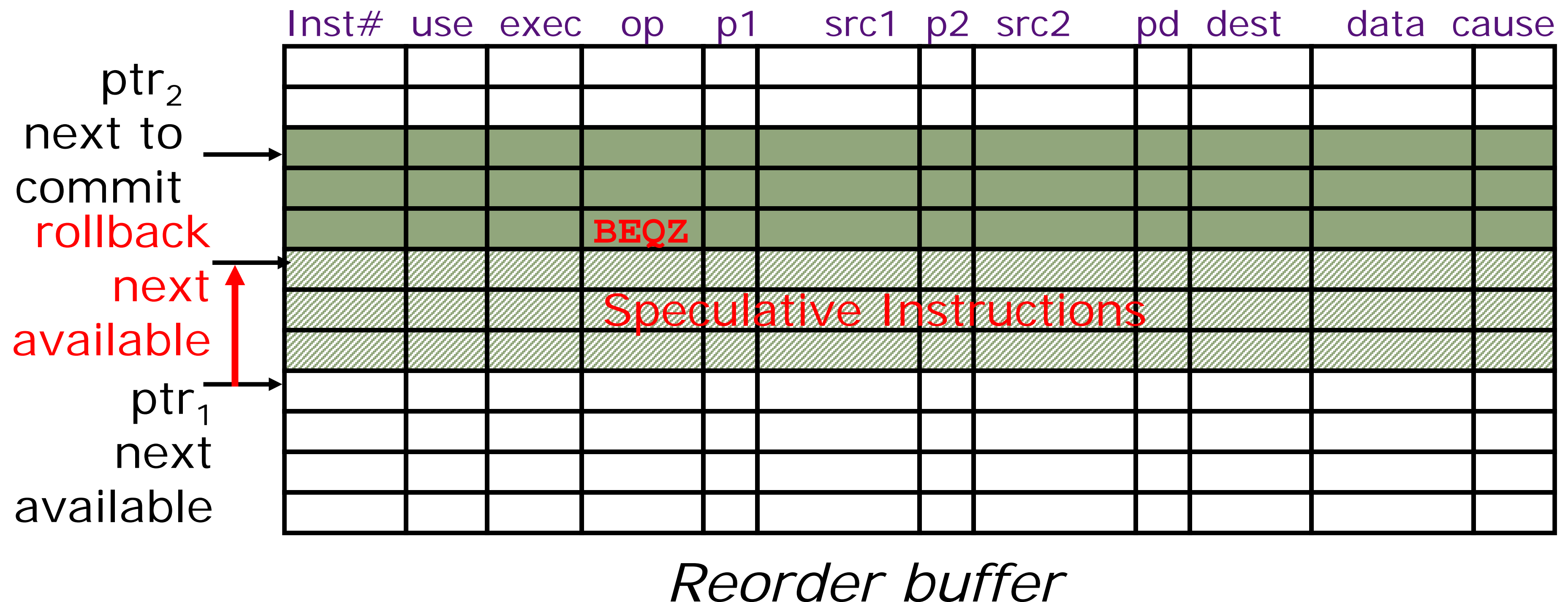
Reorder buffer

- add <pd, dest, data, cause> fields in the instruction template
- commit instructions to reg file and memory in program order ⇒ buffers can be maintained circularly
- on exception, clear reorder buffer by resetting $ptr_1 = ptr_2$

  (stores must wait for commit before updating memory)

CSAIL

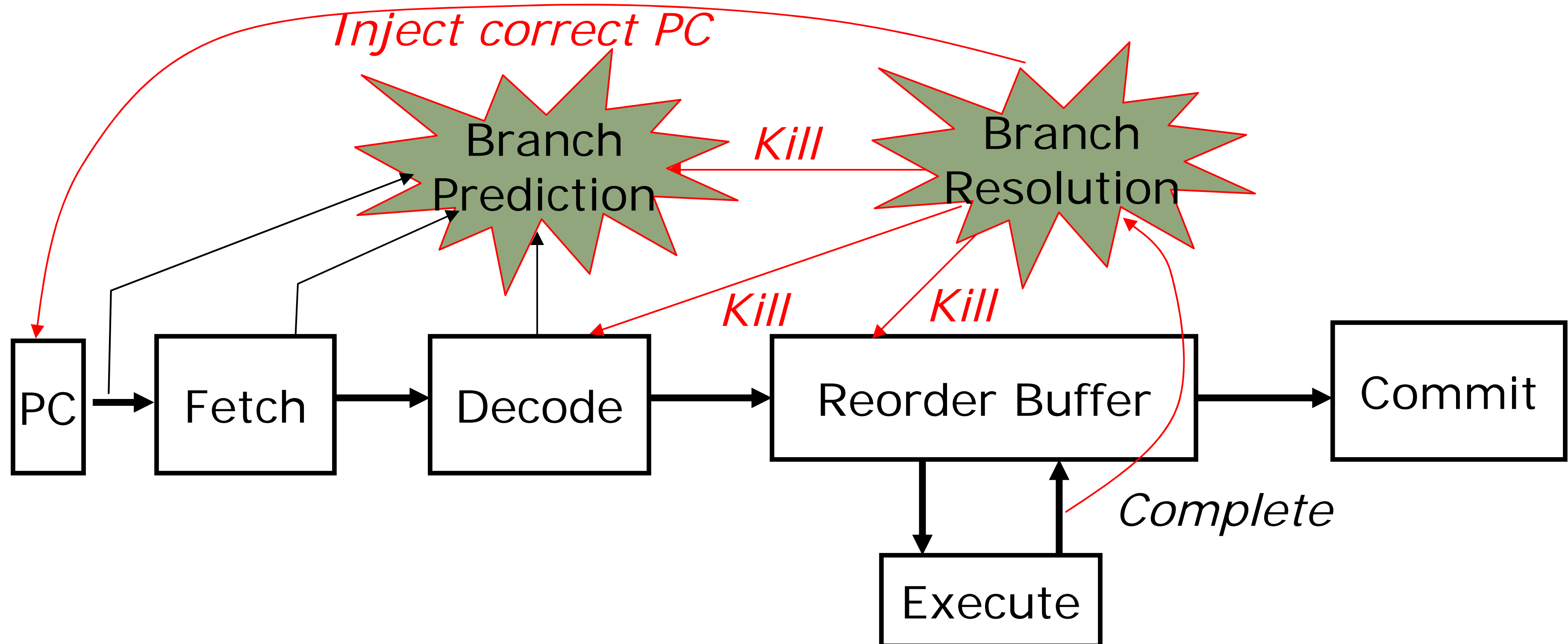
Branch Misprediction Recovery



On mispredict

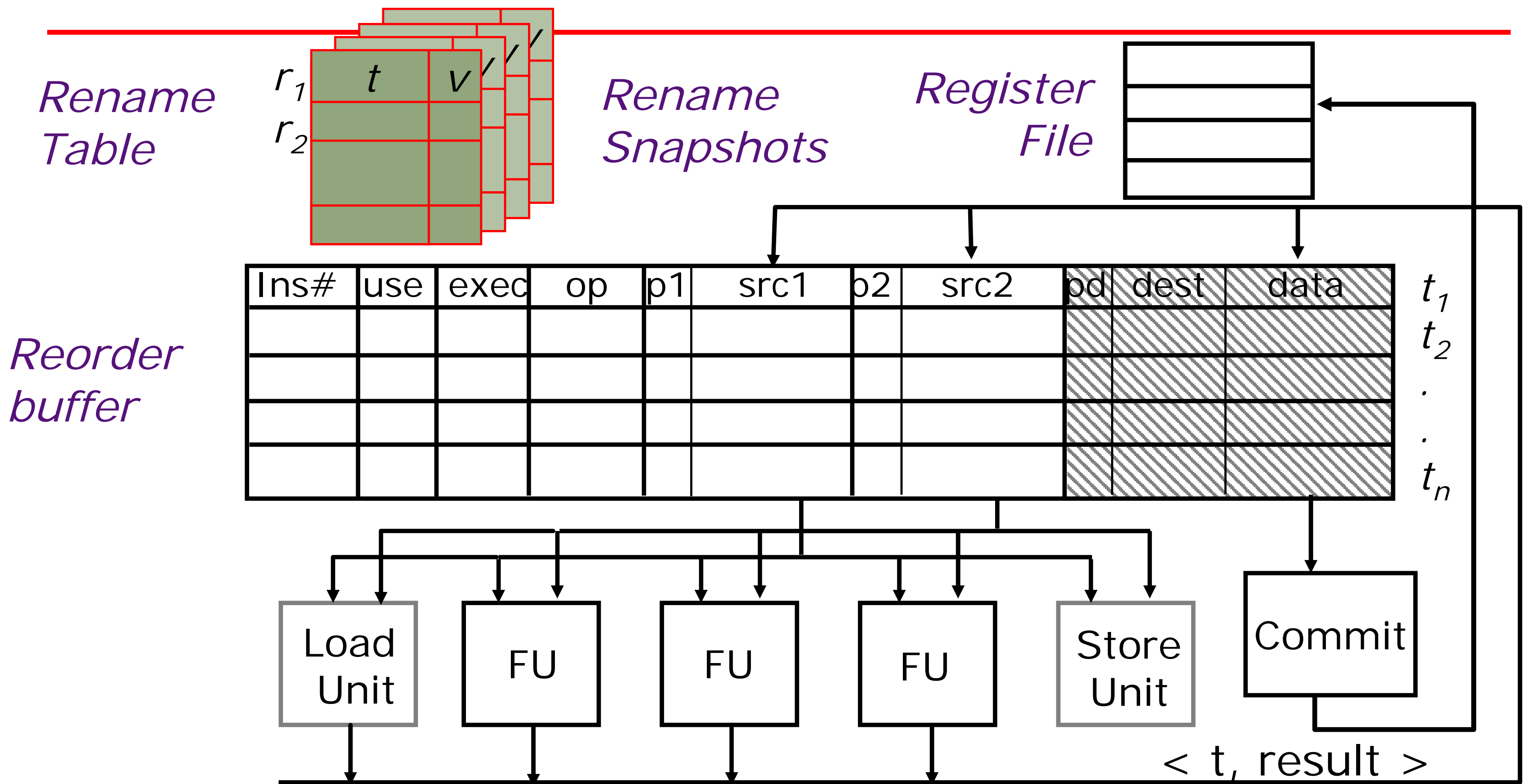
- Roll back “next available” pointer to just after branch
- Reset use bits
- Flush mis-speculated instructions from pipelines
- Restart fetch on correct branch path

Branch Misprediction in Pipeline



- Can have multiple unresolved branches in ROB
- Can resolve branches out-of-order by killing all the instructions in ROB that follow a mispredicted branch

Recovering Renaming Table



Take snapshot of register rename table at each predicted branch, recover earlier snapshot if branch mispredicted

CSAIL

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

CSAIL

Thank you !