ECE 252 / CPS 220
Advanced Computer Architecture I

Lecture 10
Instruction-Level Parallelism – Part 4

Benjamin Lee
Electrical and Computer Engineering
Duke University

www.duke.edu/~bcl15
www.duke.edu/~bcl15/class/class_ece252fall11.html

ECE252 Administrivia

4 October – Homework #2 Due

- Use blackboard forum for questions
- Attend office hours with questions
- Email for separate meetings

4 October – Class Discussion

Roughly one reading per class. Do not wait until the day before!

1. Srinivasan et al. "Optimizing pipelines for power and performance"
2. Mahlke et al. "A comparison of full and partial predicated execution support for ILP processors"
3. Palacharla et al. "Complexity-effective superscalar processors"
4. Yeh et al. "Two-level adaptive training branch prediction"

ECE252 Administrivia

6 October – Midterm Exam

- 75 minutes, in-class
- Closed book, closed notes exam

1. **Performance metrics** – performance, power, yield
2. **Technology** – trends that changed architectural design
3. **History** – Instruction sets (accumulator, stack, index, general-purpose)
4. **CISC** – microprogramming, writing microprogram fragments
5. **Pipelining** – Performance, hazards and ways to resolve them
6. **Instruction-level Parallelism** – mechanisms to dynamically detect data dependences and to manage instruction flow (Scoreboard, Tomasulo, Physical Register File)
7. **Speculative Execution** – exception handling, branch prediction
8. **Readings** – High-level questions, not details

Branch Prediction

Motivation

-- Branch penalties limit performance of deeply pipelined processors

-- Modern branch predictors have high accuracy (>95%) and can significantly reduce branch penalties

Hardware Support

-- Prediction structures: branch history tables, branch target buffer, etc.

-- Mispredict recovery mechanisms:

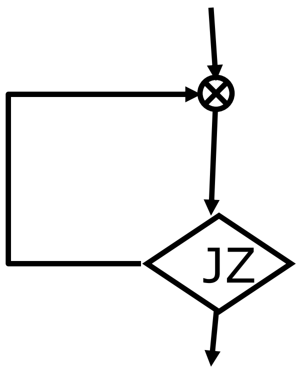
-- Separate instruction execution and instruction commit

-- Kill instructions following branch in pipeline

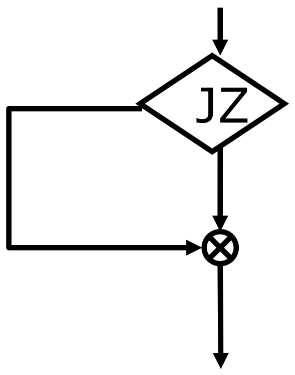
-- Restore architectural state to correct path of execution

Static Branch Prediction

backward
90%



forward
50%



On average, probability a branch is taken is 60-70%.
But branch direction is a good predictor.

ISA can attach preferred direction semantics to branches (e.g., Motorola MC8810, bne0 prefers taken, beq0 prefers not taken).

ISA can allow choice of statically predicted direction (e.g., Intel IA-64). Can be 80% accurate.

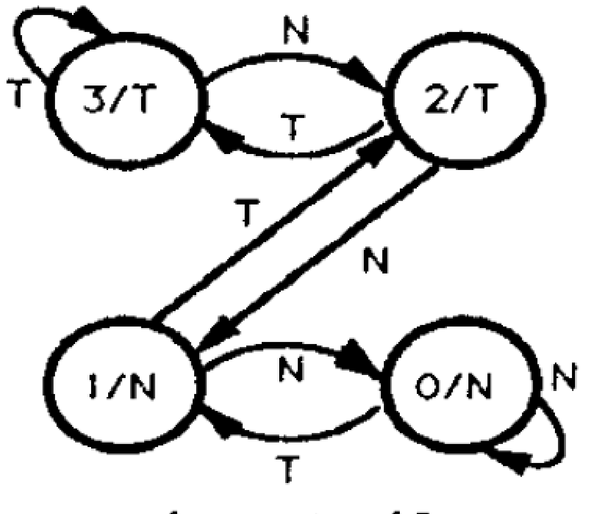
Dynamic Branch Prediction

Learn from 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 (preferred path of execution in the application)

2-bit Branch Predictor



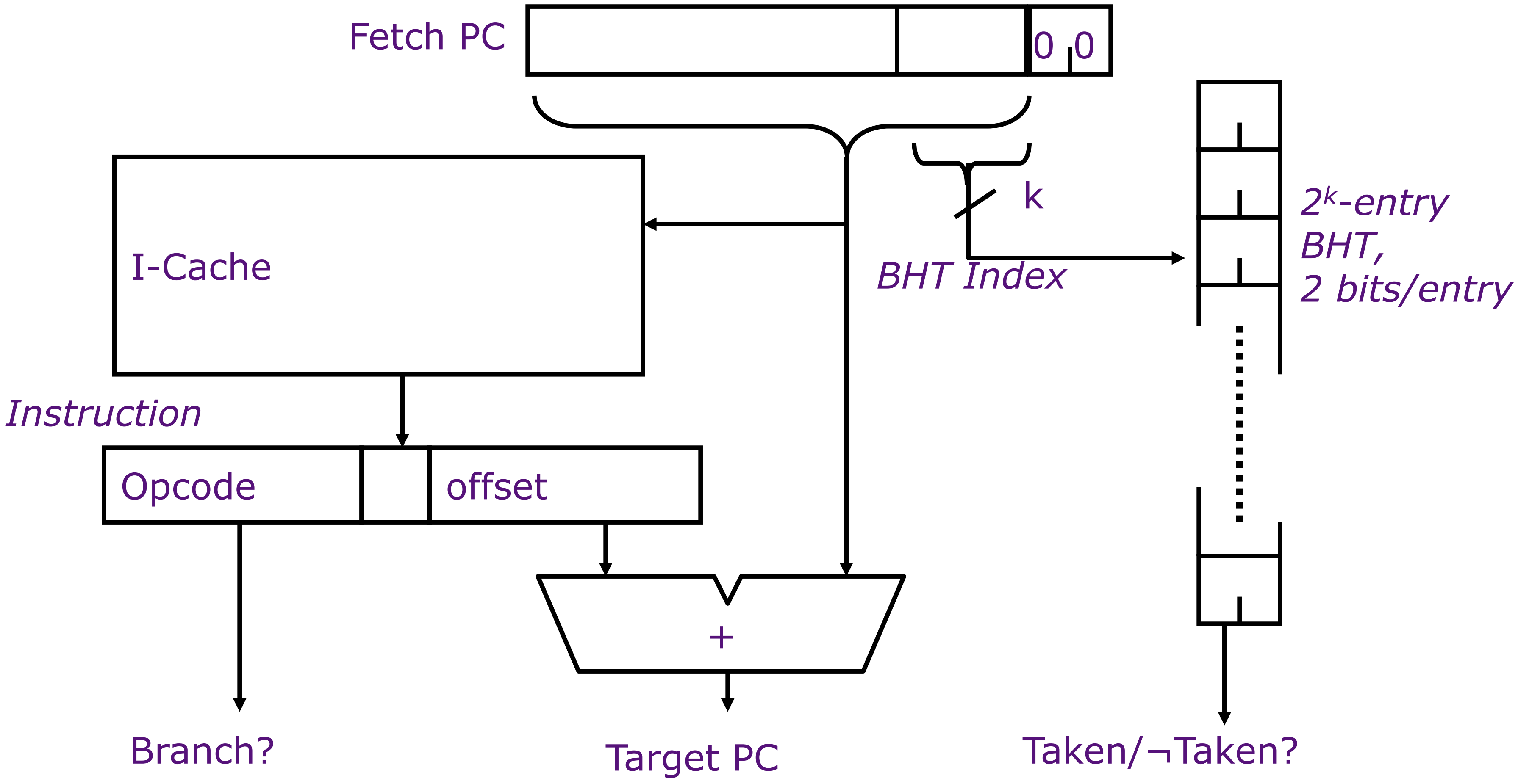
Automaton A2
(2-bit Saturating Up-down Counter)

Use two-bit saturating counter.

Changes prediction after two consecutive mistakes.

- <u>Temporal Correlation</u>
- Branch can be taken (T), not-taken (N)
- 4 states (0, 1, 2, 3), each with corresponding prediction (T/N)
- Arcs correspond to resolved branch decision (T/N)

Branch History Table (BHT)

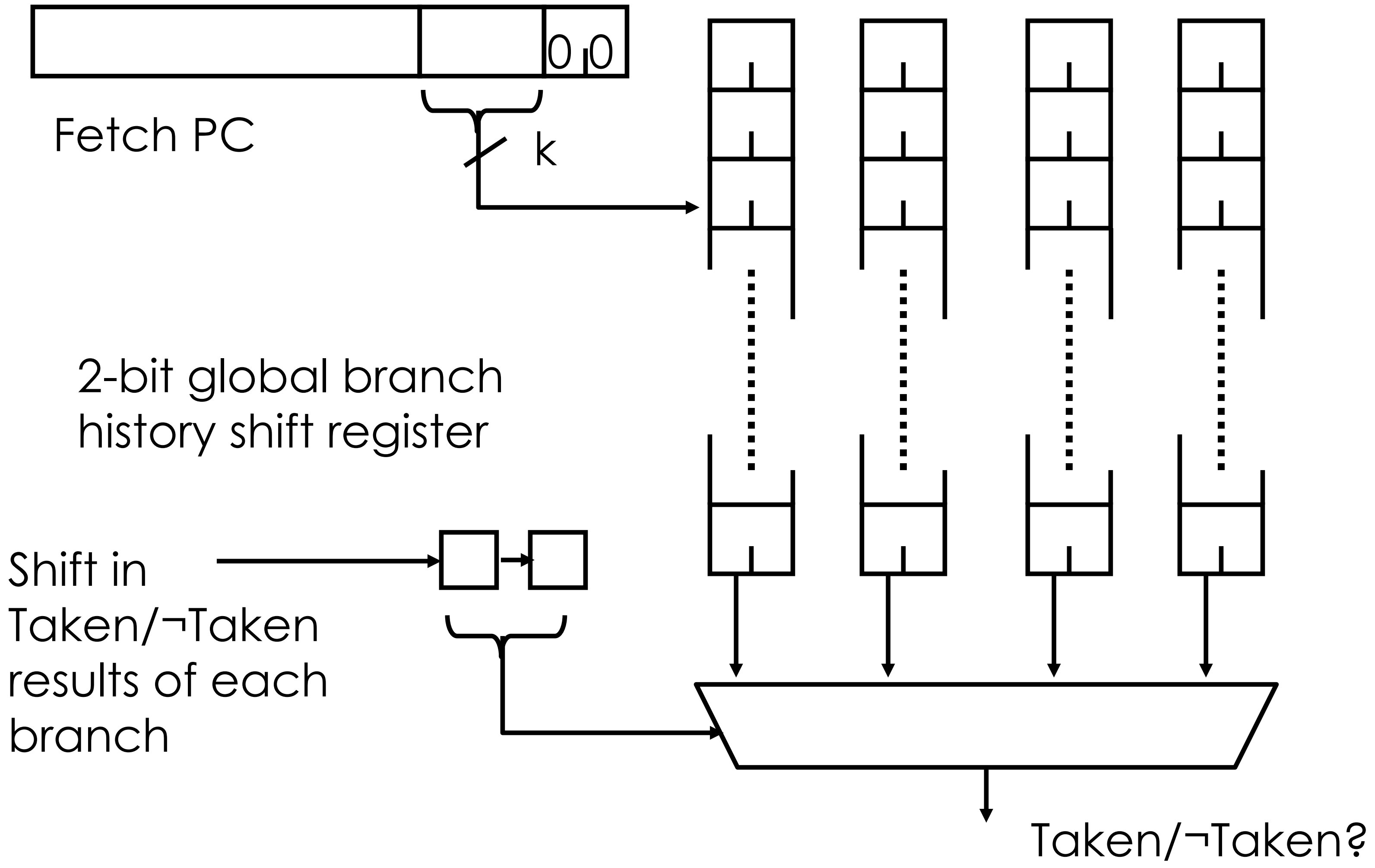


BHT is an array of 2-bit branch predictors, indexed by branch PC

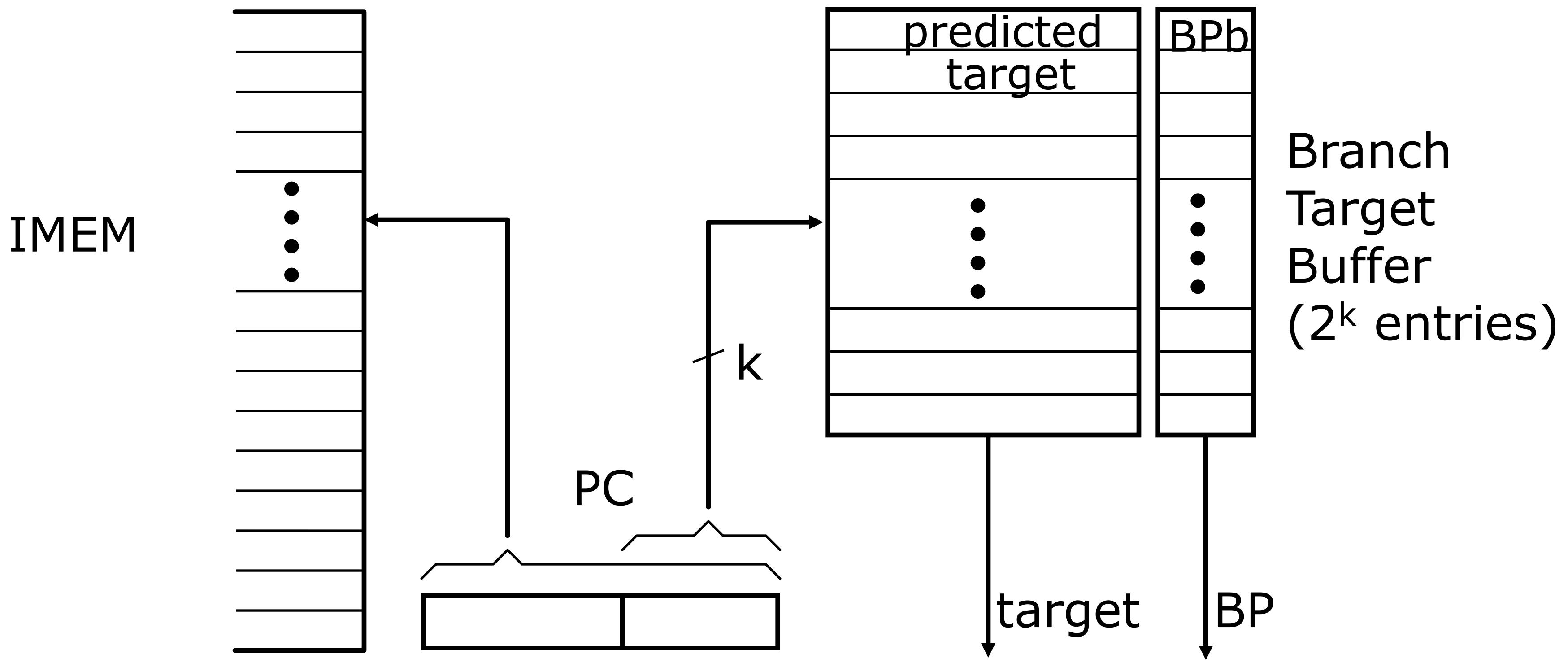
4K-entry branch history table, 80-90% accurate

Two-Level Branch Prediction

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


0 0
Fetch PC
k
2-bit global branch history shift register
Shift in Taken/¬Taken results of each branch
Taken/¬Taken?

Branch Target Buffer (BTB) – v1



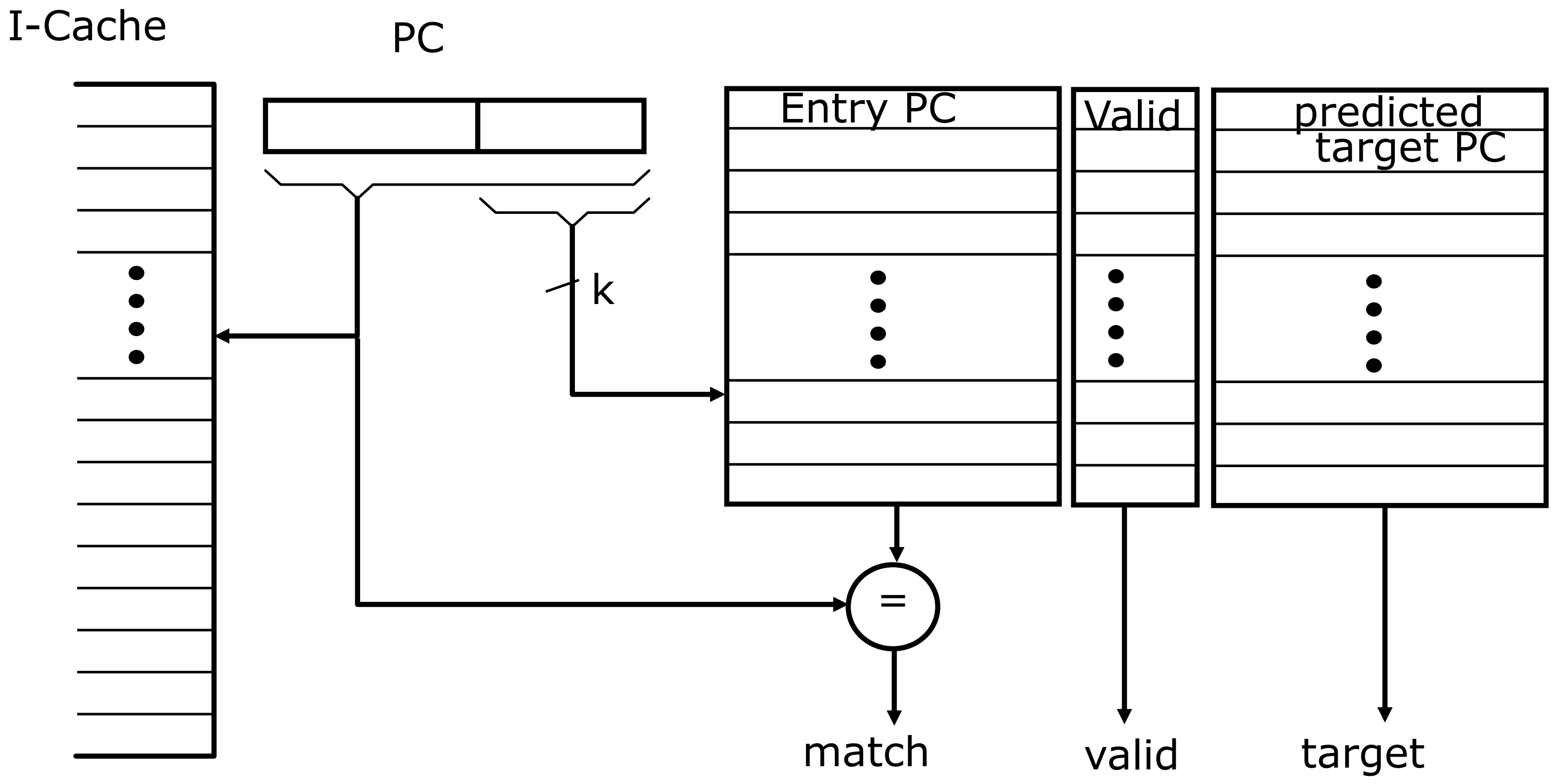
BHT only predicts branch direction (taken, not taken). Cannot redirect instruction flow until after branch target determined.

Store target with branch predictions.

During fetch: If (BP == taken) then nPC=target, else nPC=PC+4

Later: update BHT, BTB

Branch Target Buffer (BTB) – v2



Keep both branch PC and target PC in the BTB

If match fails, PC+4 is fetched

Only taken branches and jumps held in BTB

Mispredict Recovery

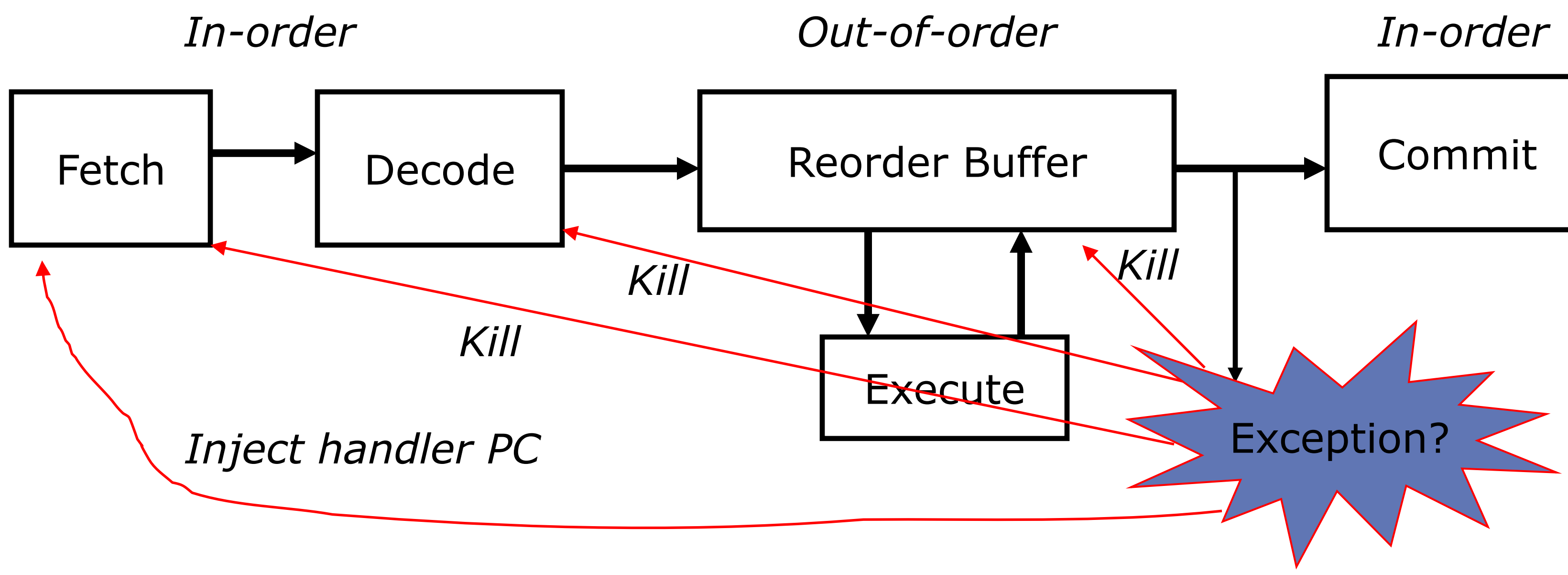
In-order execution

- No instruction following branch can commit before branch resolves
- Kill all instructions in pipeline behind mis-predicted branch

Out-of-order execution

- Multiple instructions following branch can complete before one branch resolves

In-order Commit



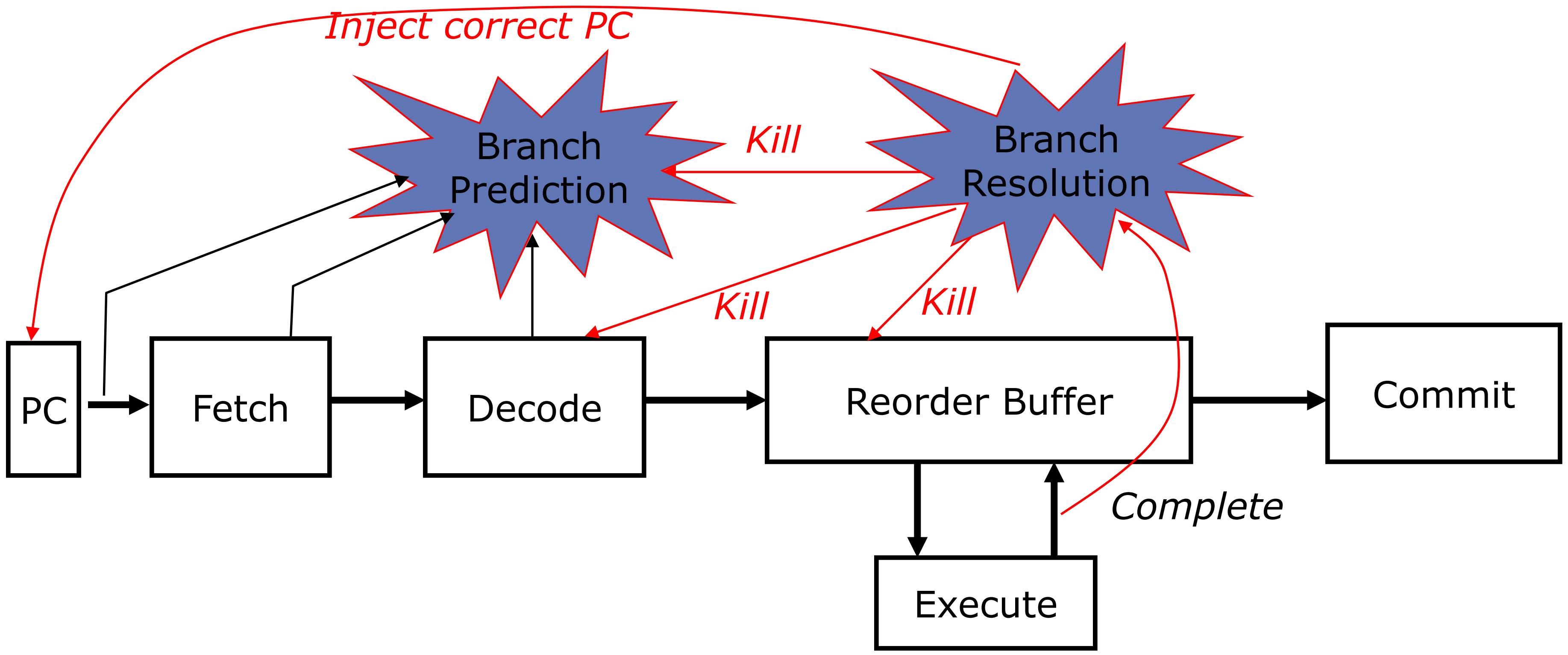
-- Instructions fetched, decoded in-order (entering the reorder buffer -- ROB)

-- Instructions executed out-of-order

-- Instructions commit in-order (write back to architectural state)

-- Temporary storage needed in ROB to hold results before commit

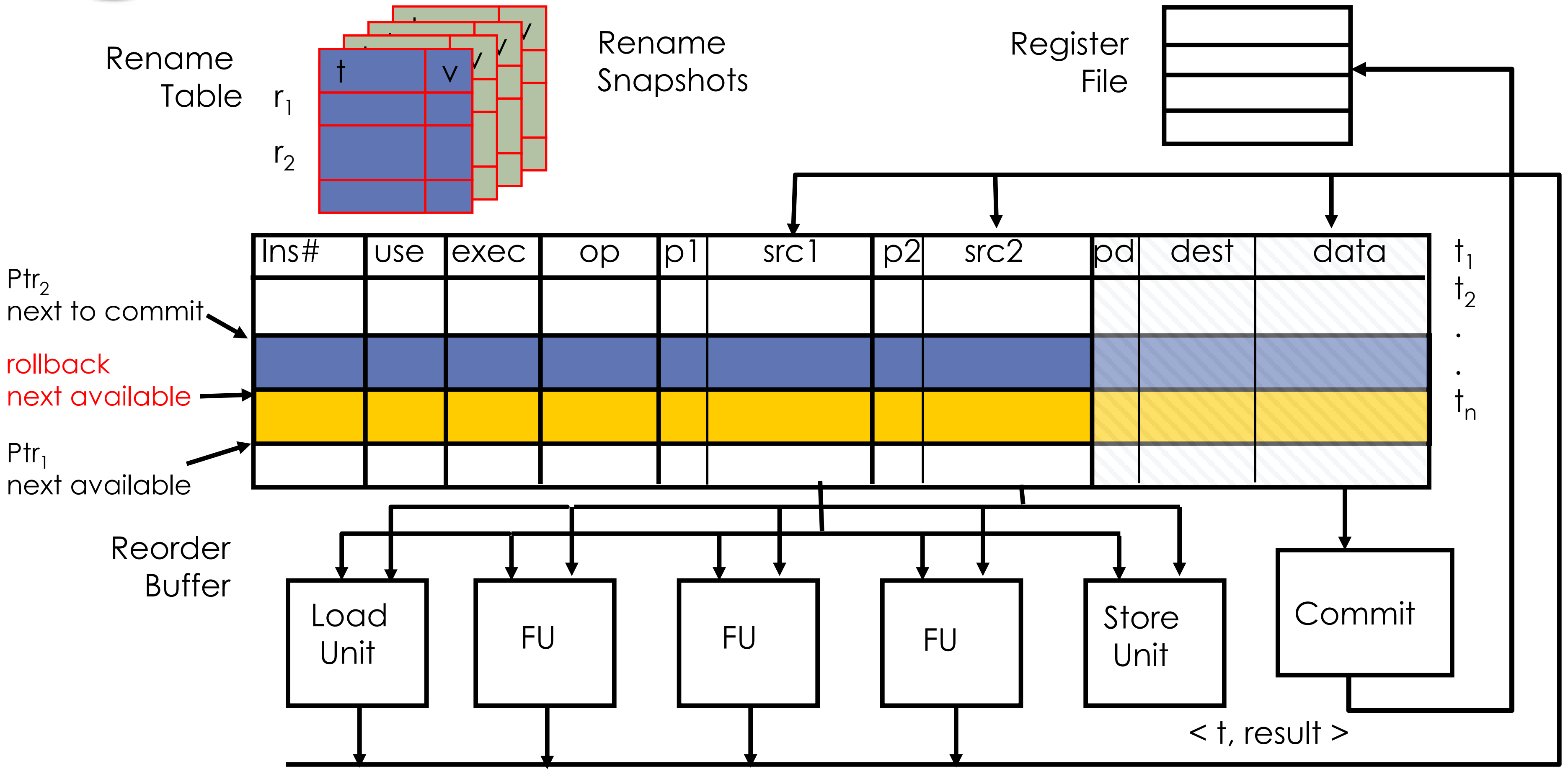
Branch Misprediction in Pipeline



-- Can have multiple unresolved branches in reorder buffer -- ROB

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

Mispredict Recovery



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

Acknowledgements

These slides contain material developed and copyright by

- Arvind (MIT)
- Krste Asanovic (MIT/UCB)
- Joel Emer (Intel/MIT)
- James Hoe (CMU)
- John Kubiatowicz (UCB)
- Alvin Lebeck (Duke)
- David Patterson (UCB)
- Daniel Sorin (Duke)