# HARDWARE SPECULATION

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

# Announcement

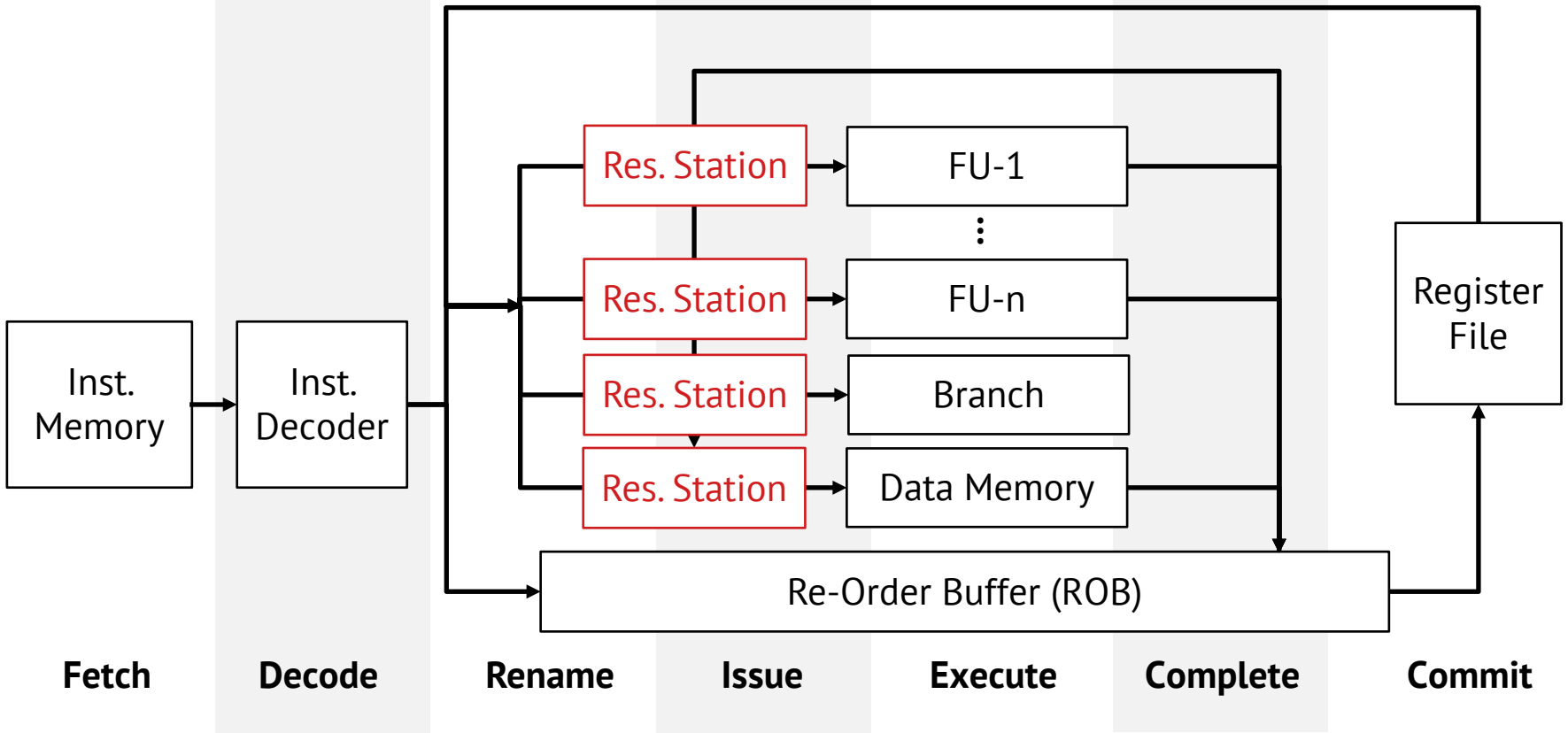
- Homework 5
  - Apparently, the easiest one so far ☺
  - Do not miss the deadline!
- Homework 6
  - ** Postponed till after Fall Break **

# Recall: Out-of-Order Execution

- Producer-consumer chains on the fly
  - Register renaming: remove anti-/output-dependences via register tags
  - Limited by the number of instructions in the instruction window (ROB)
- Out-of-order issue (dispatch)
  - Broadcast tags to waiting instructions
  - Wake up ready instructions and select among them
- Out-of-order execute/complete
- In-order fetch/decode and commit

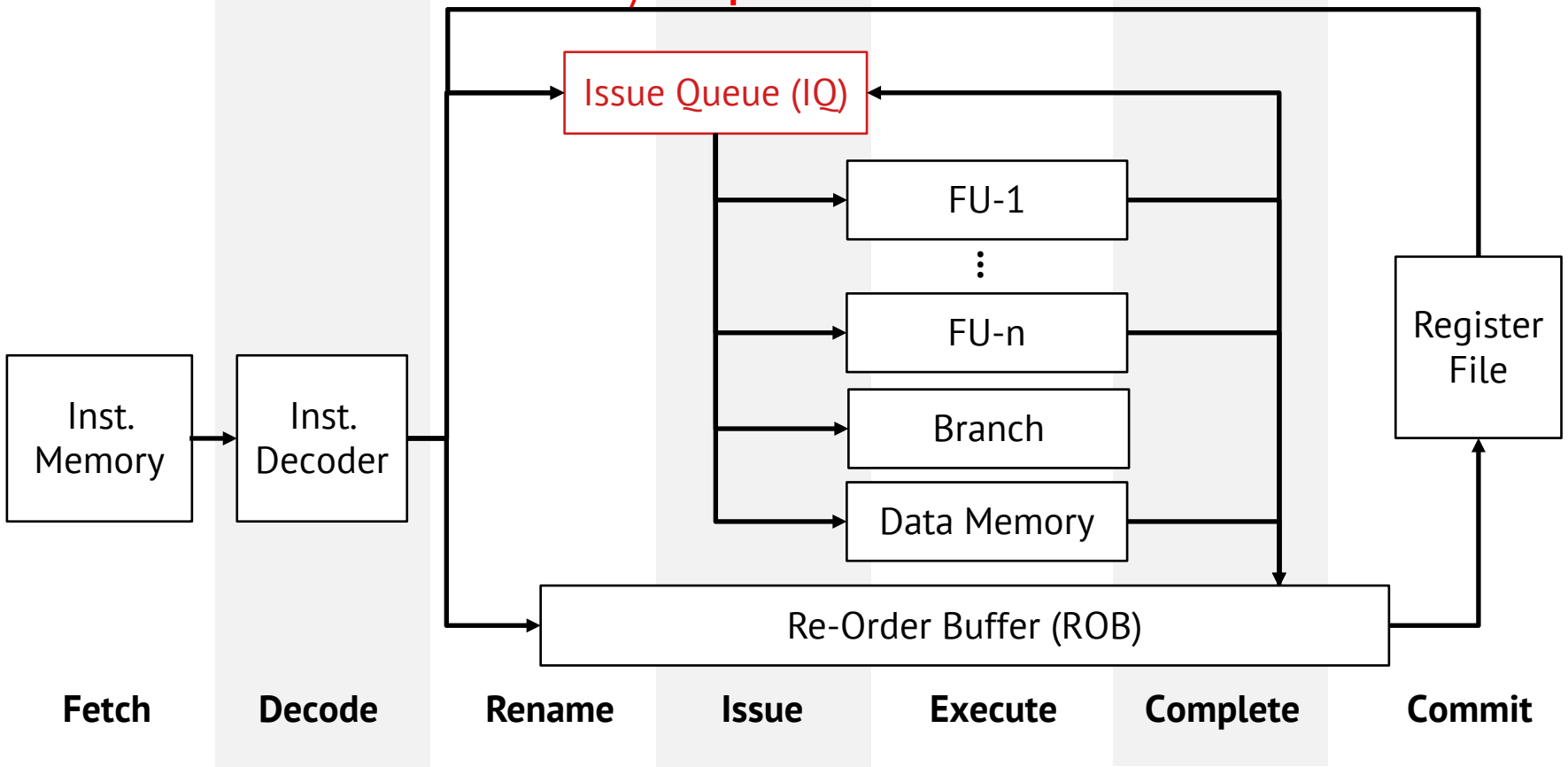
# Out-of-Order Pipelines

- Distributed reservation stations
  - In-order issue/dispatch



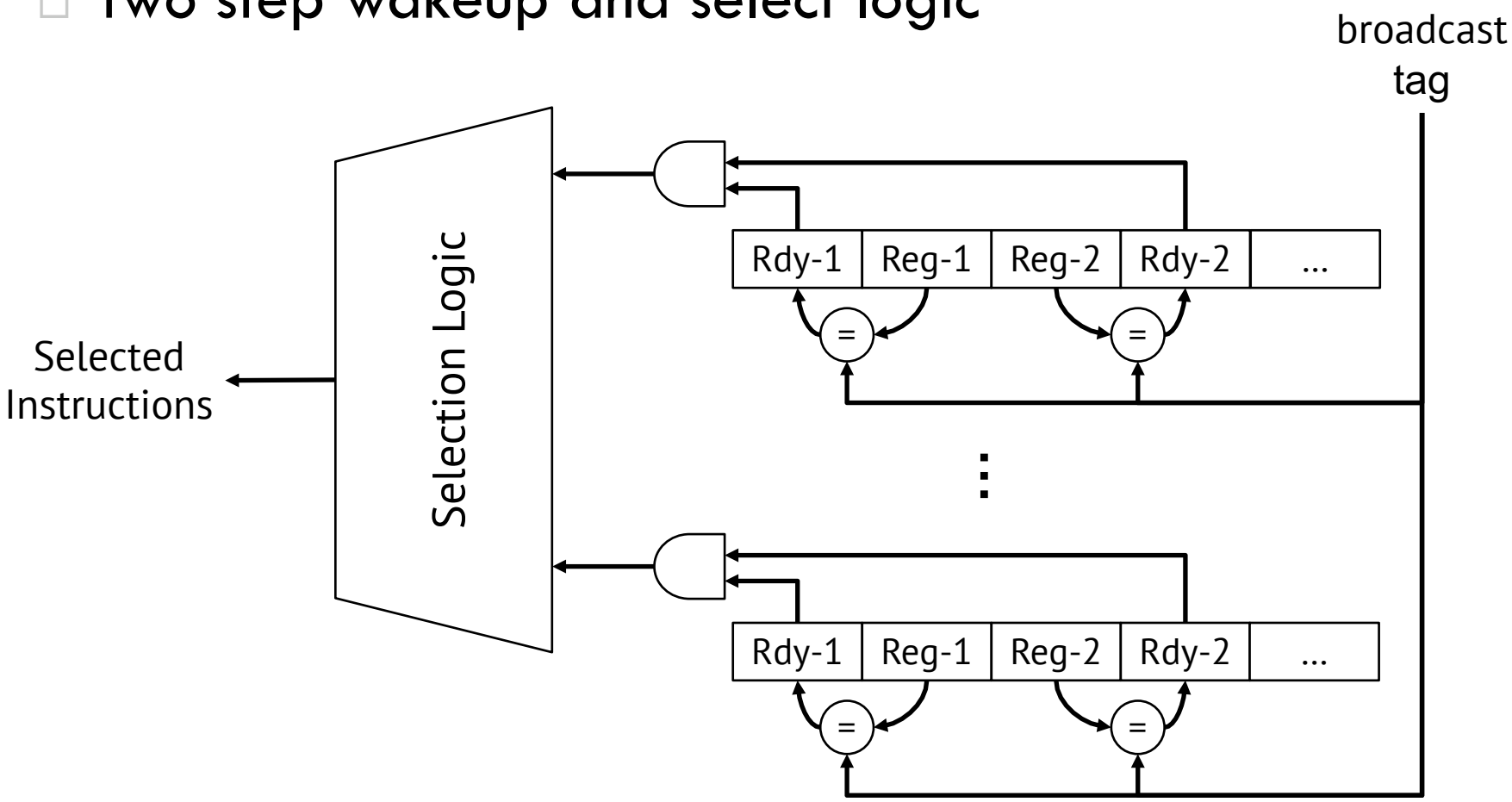
# Out-of-Order Pipelines

- Out of order issue/dispatch to functional units
  - Out-of-order issue/dispatch



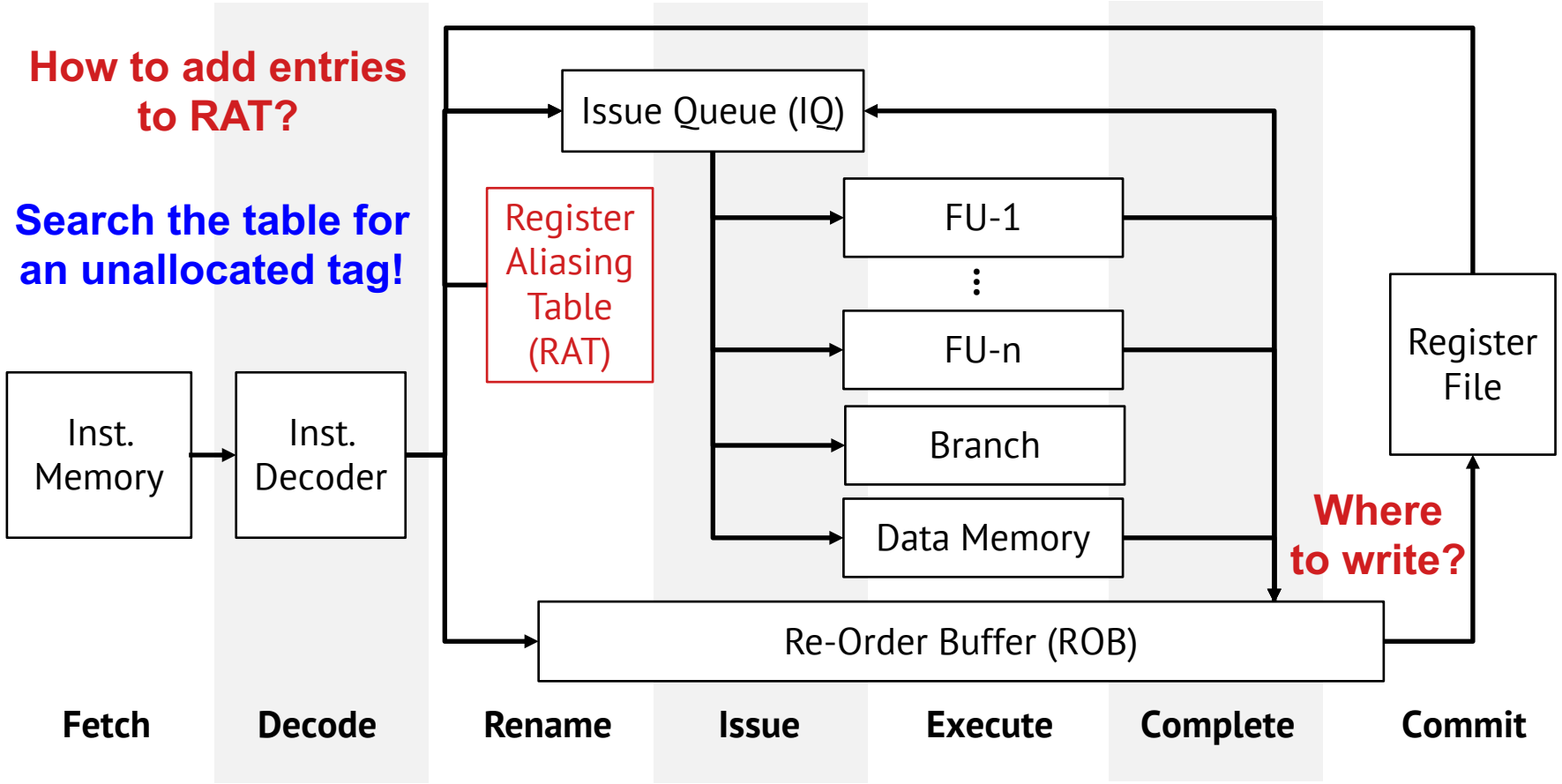
# Out-of-Order Issue Queue

- Two step wakeup and select logic



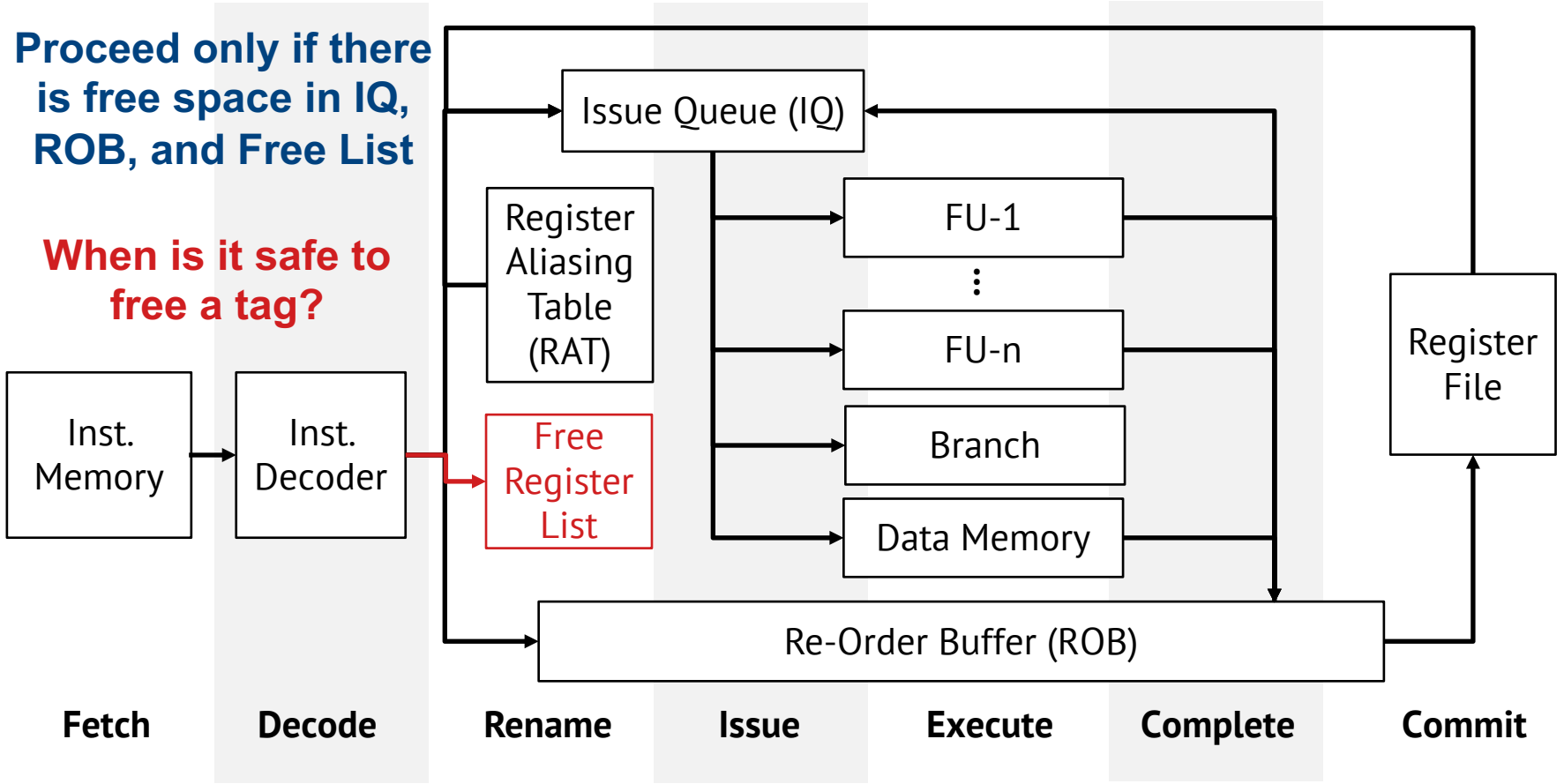
# Register Renaming

- Register aliasing table for fast lookup



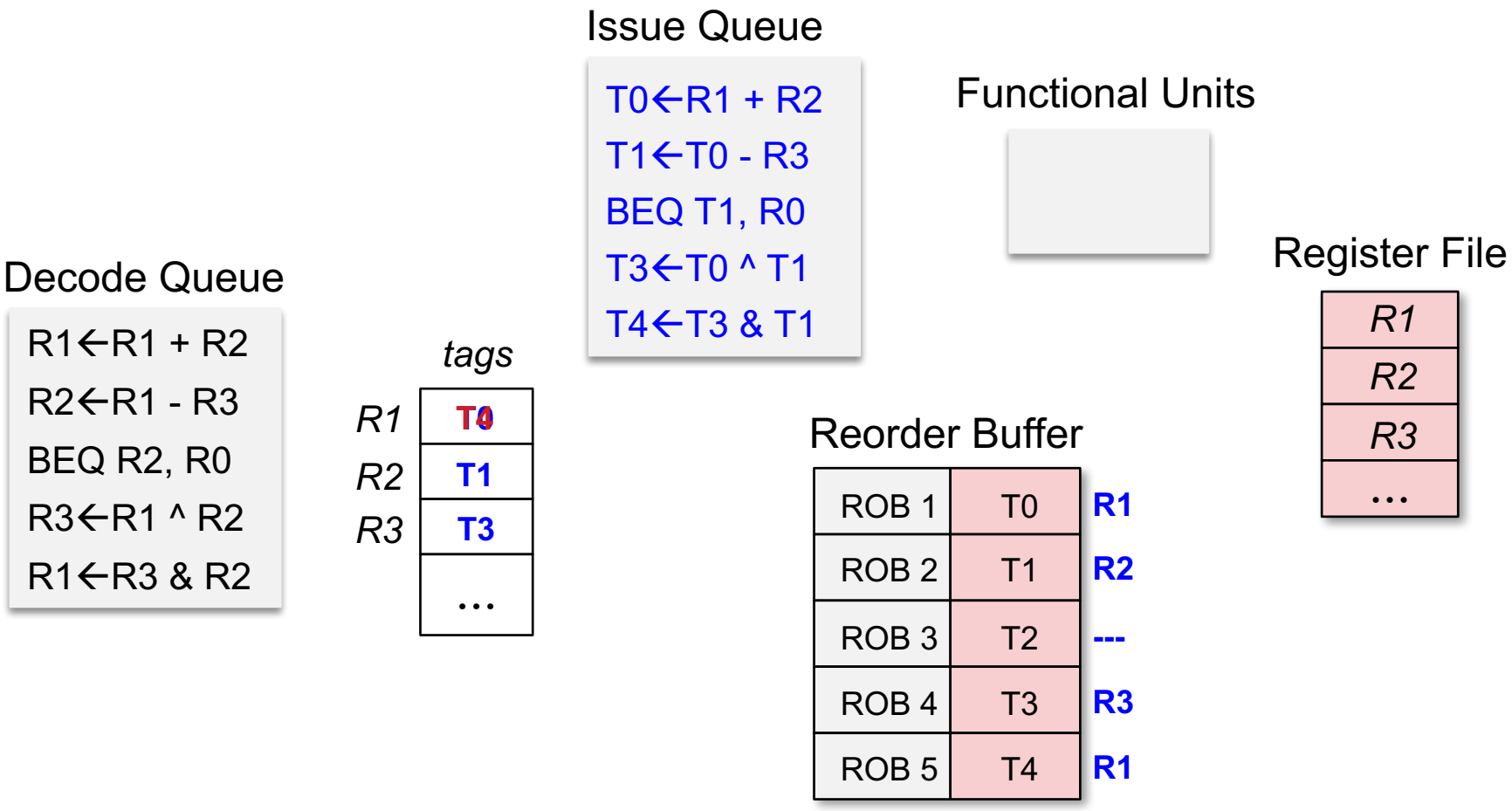
# Register Renaming

- Free register list for fast register renaming



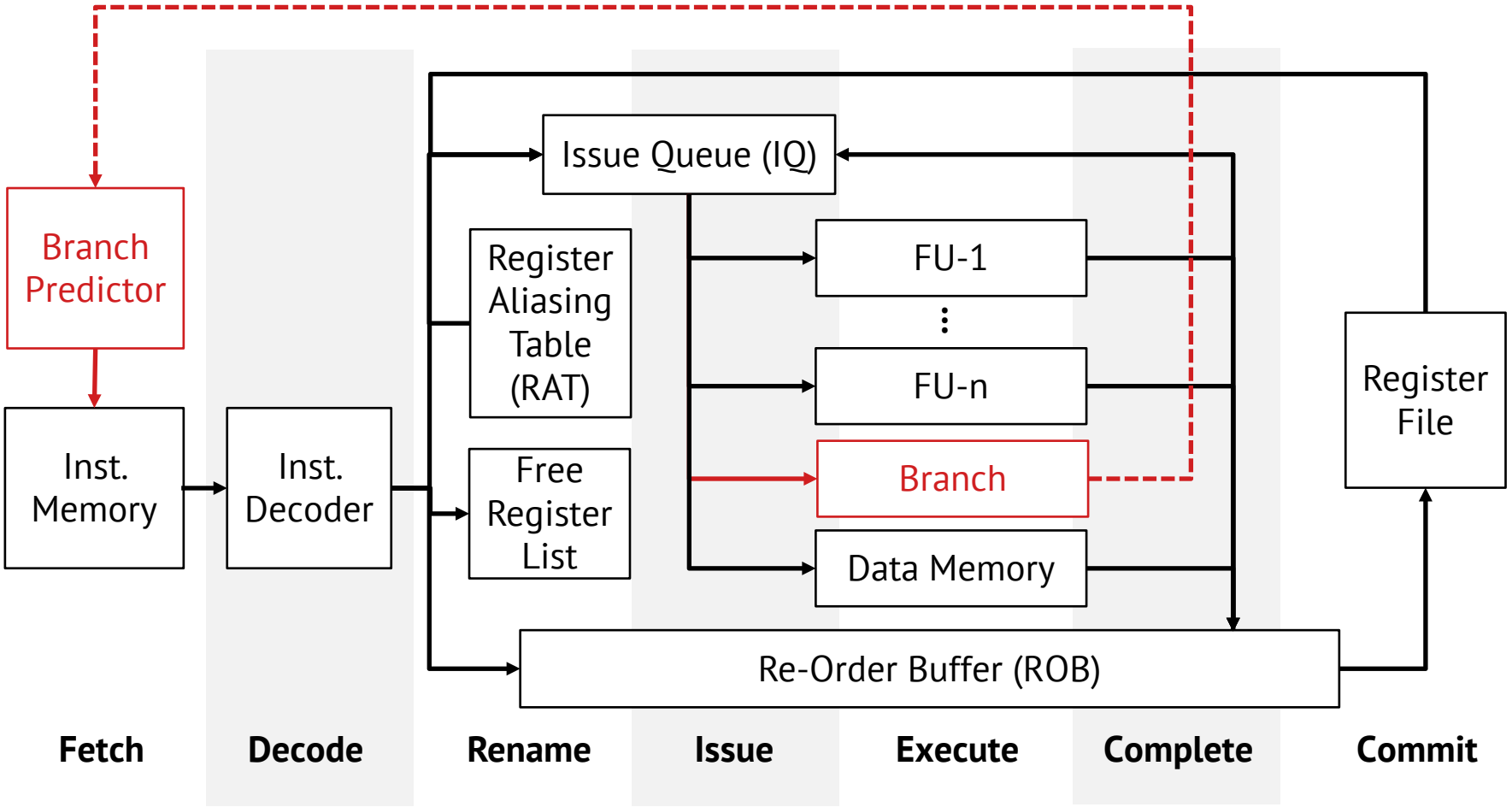
# Register Renaming Example

- Where values are stored?



# Branch Recovery

- How to handle branches?



# Revisit Branch Prediction

- Problem: find the average number of stall cycles caused by branches in a pipeline, where branch misprediction penalty is 20 cycles, branch predictor accuracy is 90%, and branch target buffer hit rate is 80%. Every fifth instruction is a branch; 30% of branches are actually taken.

# Revisit Branch Prediction

- Problem: find the average number of stall cycles caused by branches in a pipeline, where branch misprediction penalty is 20 cycles, branch predictor accuracy is 90%, and branch target buffer hit rate is 80%. Every fifth instruction is a branch; 30% of branches are actually taken.

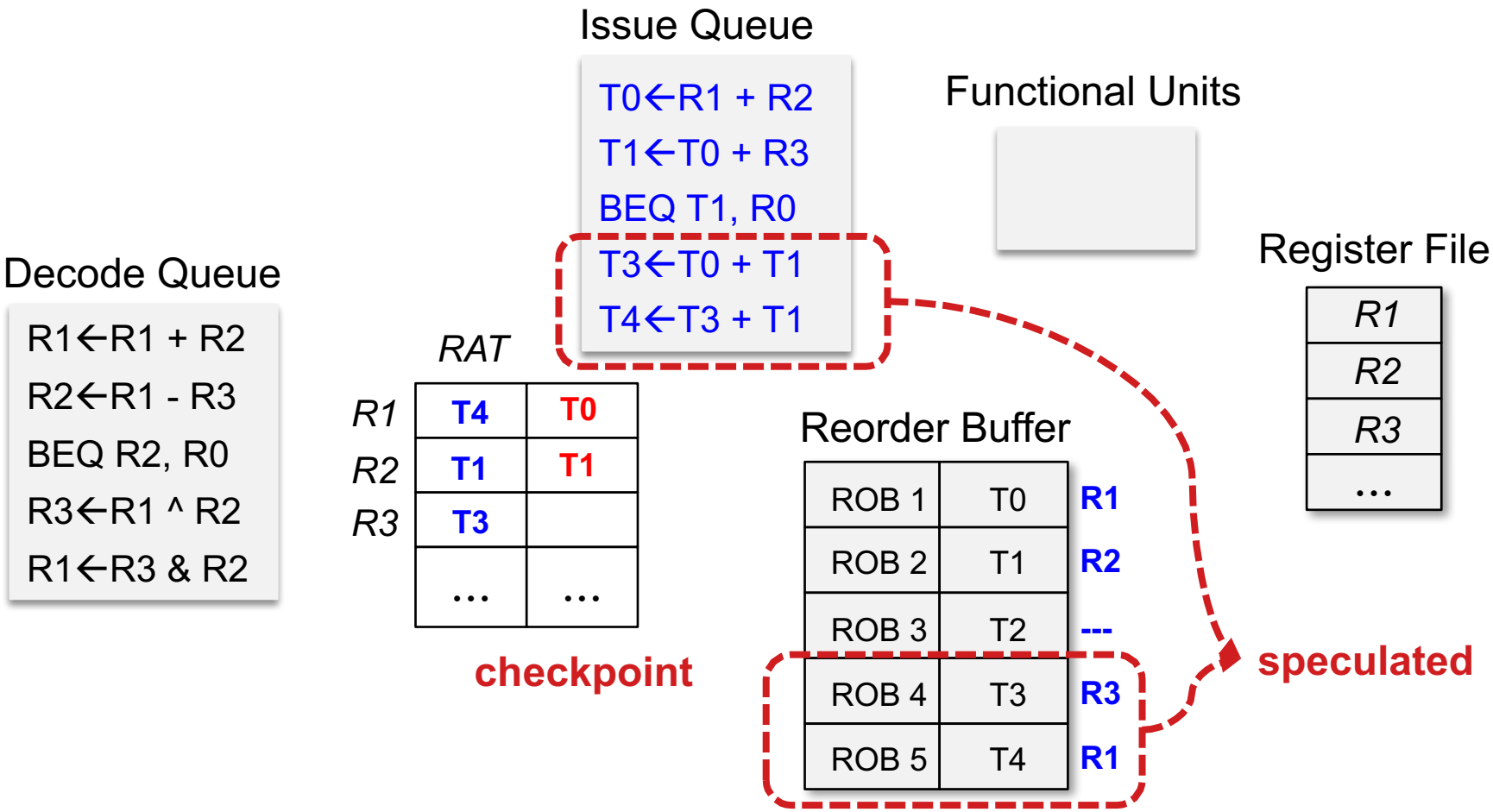
  - Average misses = 1- (0.3x0.9x0.8+0.7x0.9) = 0.151
  - Average stalls = 20x0.2x0.151 = 0.6

# Speculated Execution

- Problem: branch may significantly limit performance
  - consumer of a load or long latency instructions
- Solution: speculative instruction execution
  - Fetch and decode instructions speculatively
  - Issue and execute speculative instructions
  - Branch resolution
    - Nullify the impact of speculative instructions if mispredicted
    - Commit speculative instructions (writes to register file/memory) only if prediction was correct

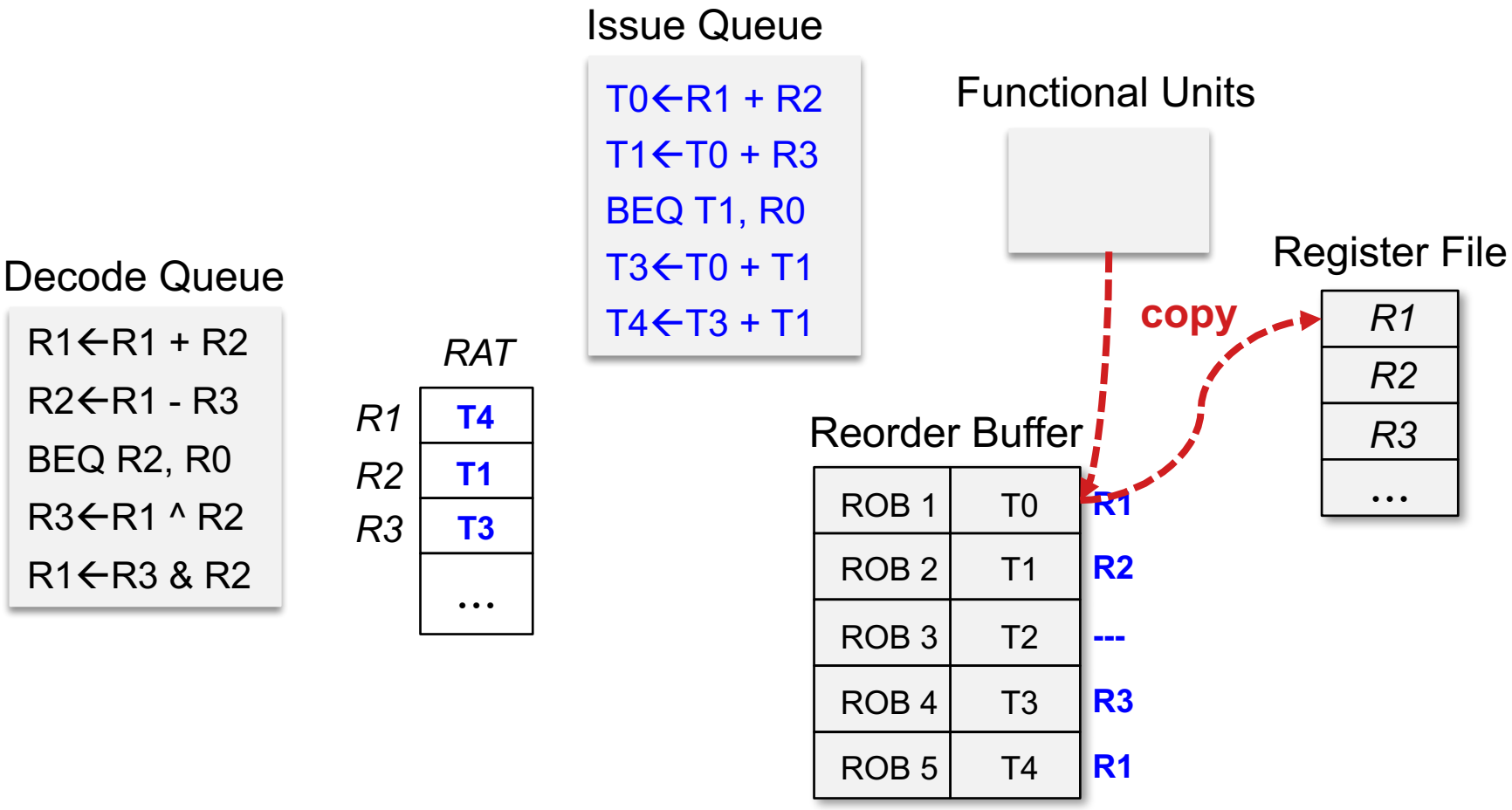
# Branch Recovery

- Squash all mispredicted entries



# Physical Register File

- Avoid copying register values multiple times



# Physical Register File

- Avoid copying register values multiple times

Note1: only a subset of the Phy. Reg. file is committed at any time.

Issue Queue

P0←R1 + R2
P1←P0 + R3
BEQ P1, R0
P3←P0 + P1
P4←P3 + P1

Functional Units

Retire RAT

| | |
|---|---|
| *P0* | *R1* |
| *P1* | *R2* |
| | *R3* |
| … | |

Decode Queue

R1←R1 + R2
R2←R1 - R3
BEQ R2, R0
R3←R1 ^ R2
R1←R3 & R2

*Front RAT*

| | |
|---|---|
| *R1* | **P4** |
| *R2* | **P1** |
| *R3* | **P3** |
| | … |

Reorder Buffer

| | |
|---|---|
| ROB 1 | **R1,P0** |
| ROB 2 | **R2,P1** |
| ROB 3 | **---** |
| ROB 4 | **R3,P3** |
| ROB 5 | **R1,P4** |

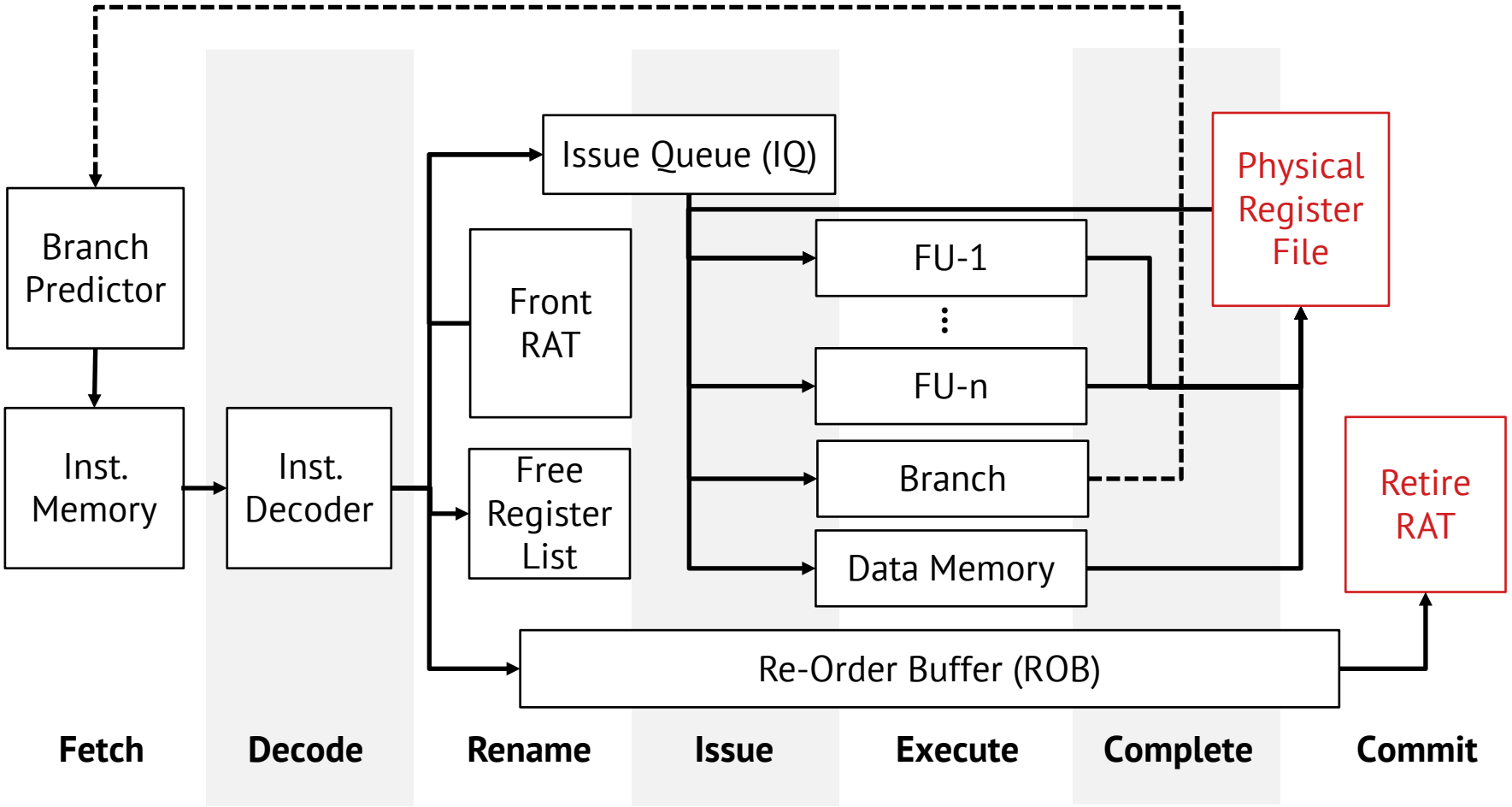
Phy. Reg. File

| |
|---|
| P0 |
| P1 |
| P2 |
| P3 |
| P4 |
| P5 |
| … |

Note2: no need for storing values in ROB or IQ

# Double RAT Architecture

- What is the size of ROB?



# Physical Register Release

- Example: when is it safe to free p30 (R1)?

```
ADD  R1, R2, R3
SUB   R2, R1, R3
…
ADD  R3, R1, R2
…
SUB  R1, R3, R2
ADD R2, R1, R3
```