Commit

MO801

Why a new stage is necessary?

- Both in-order and out-of-order processors can have instructions reaching write-back stage out of order
- Processors need to handle exceptions
- Processors need to handle branch miss-prediction
- The commit stage helps all these issues

Processor states

- Architectural state
  - Registers that should be accessed and visible through all processor (committed values)
- Speculative state
  - Data that should be used with caution (speculatively)
  - Processor needs to keep track of all speculative state
  - Processor should be able to undo all speculative state and its consequences
  - Committed speculative state $\rightarrow$ Architectural state

RISC vs CISC

- CISC instructions are broken into micro-operations
- In general, micro-operations should commit together
  - There are a few exceptions in x86 like memory copy instructions

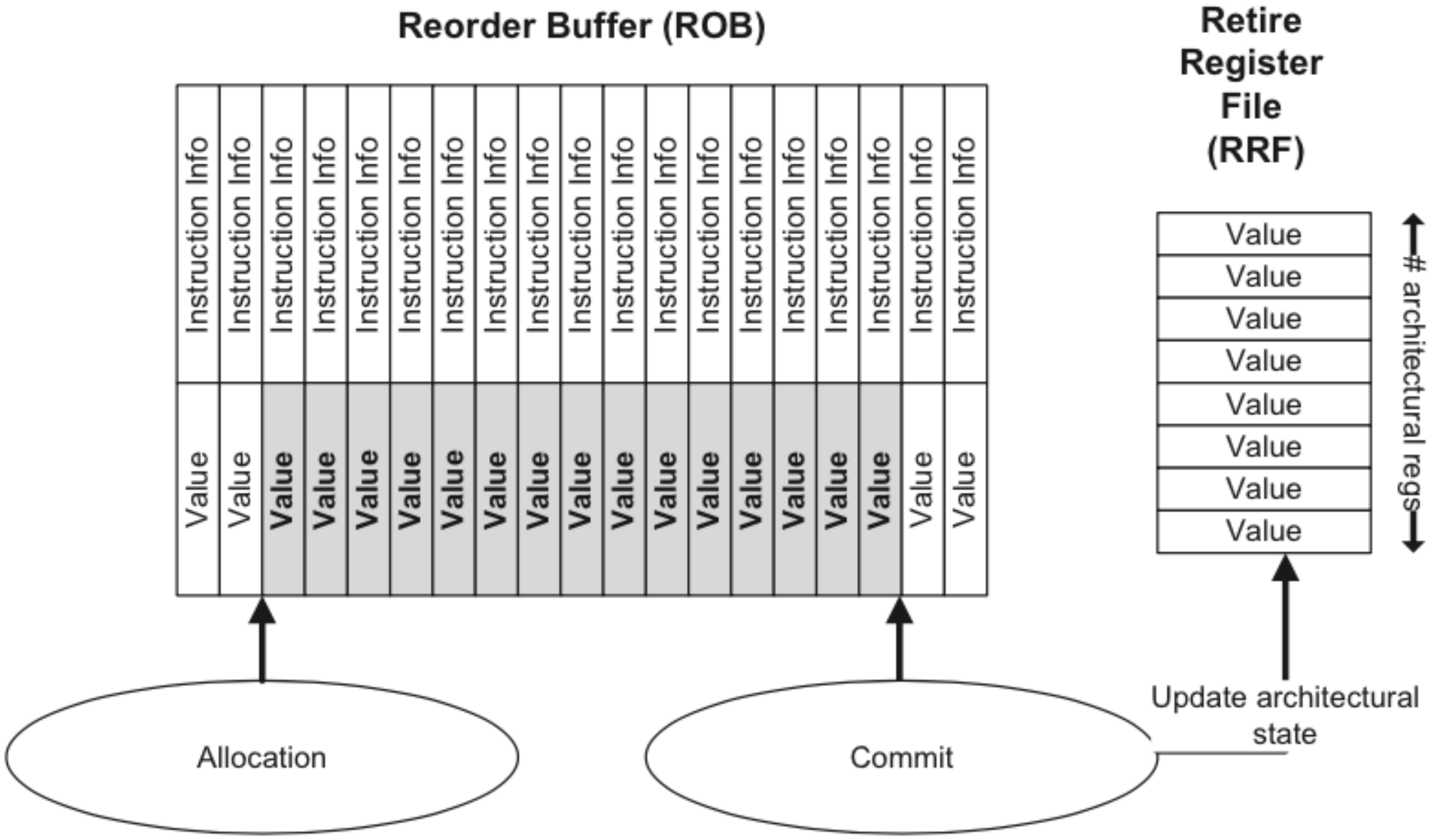
Reclaiming Resources

- Reorder Buffer (ROB) and Memory Order Buffer (MOB) entries are reclaimed at commit
- Physical registers can be reclaimed if they are not going to be used

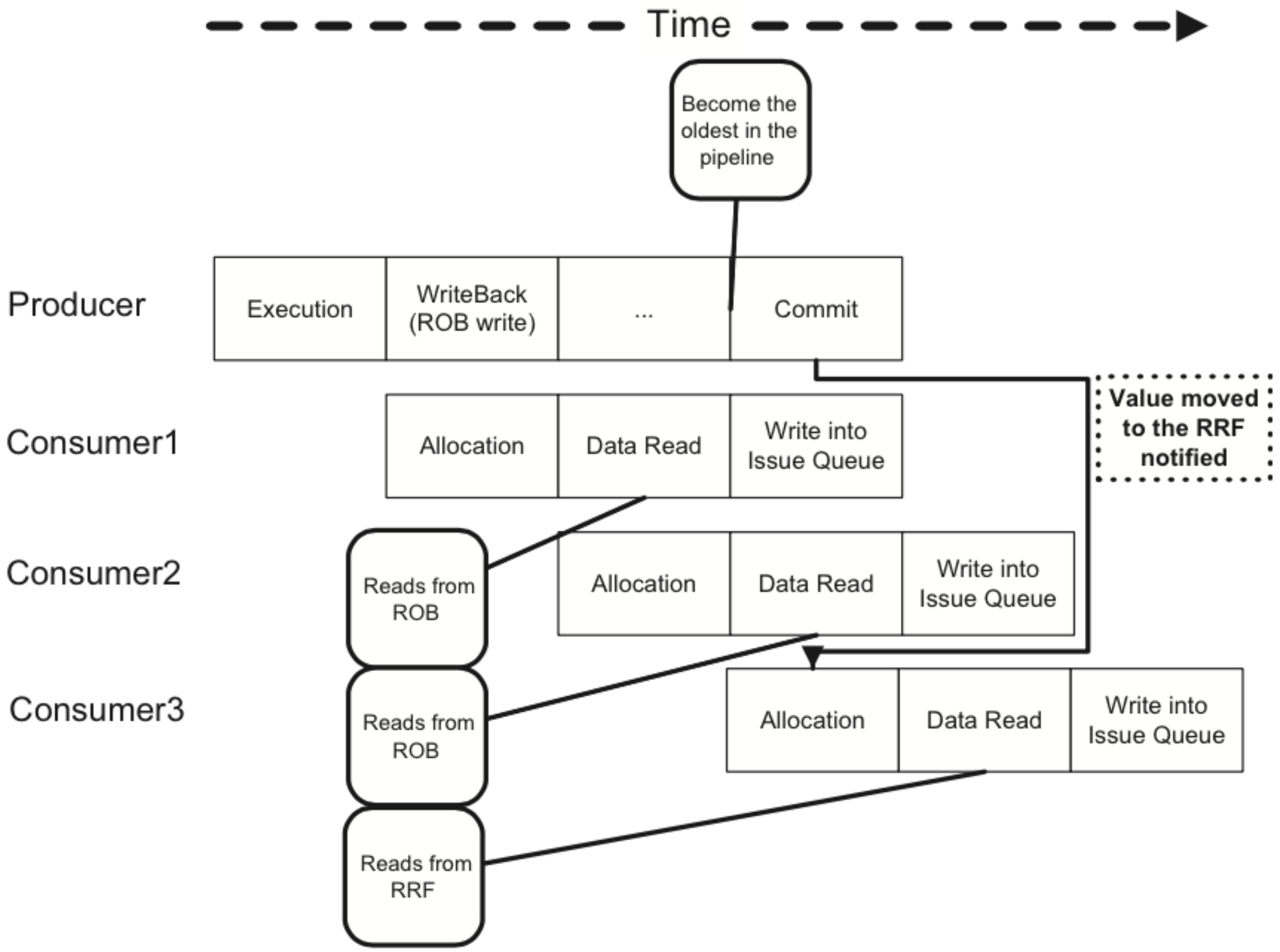
Managing Architectural States

- Stores can only update memory at commit stage
  - Loads need to check the store buffer
- Two approaches to keep track of architectural states
  - Reorder Buffer (ROB) and Retire Register File (RRF)
  - Merged Register File (MRF)

Retire Register File



RRF in use


Time
Become the oldest in the pipeline
Producer
Execution
WriteBack (ROB write)
...
Commit
Value moved to the RRF notified
Consumer1
Allocation
Data Read
Write into Issue Queue
Consumer2
Reads from ROB
Allocation
Data Read
Write into Issue Queue
Consumer3
Reads from ROB
Allocation
Data Read
Write into Issue Queue
Reads from RRF

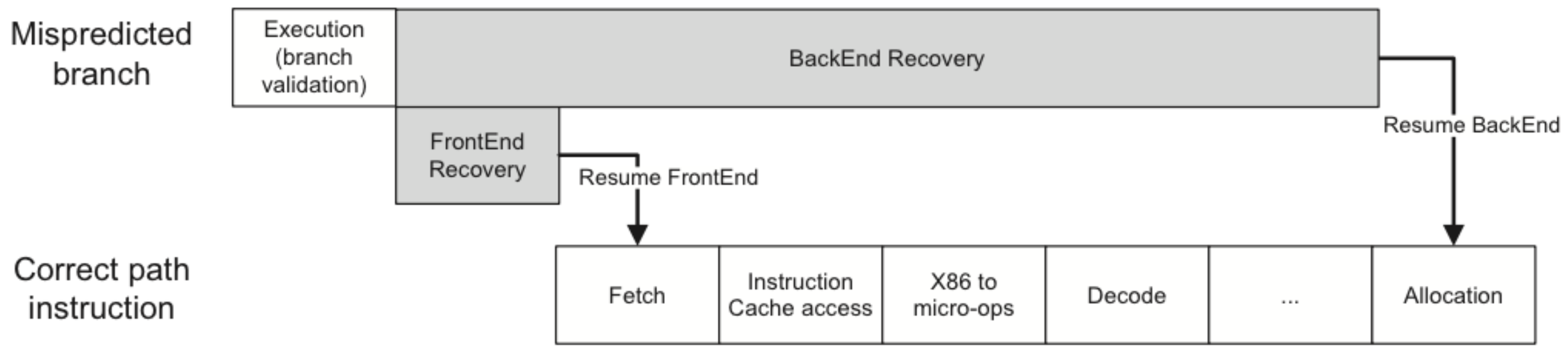
Merged Register File

- One unique register file to hold Architectural State and Speculative State
- Requires a list of Speculative values
- Values are not moved from one place to another
- Better for processors where instructions read their source operands after issue
- Centralized vs Decentralized scheme
  - ROB
  - MRF
  - Multiple pipelines

Recovering…

- Make sure that the instruction is not speculative in case of exception
- Two distinct tasks:
  - Front end recovery
    - Flush intermediate buffers
    - Restore branch predictor history
    - Update PC
  - Back end recovery
    - Remove all speculative instructions
    - Restore renaming tables
    - Reclaim architectural state storage

Pipeline for branch recovery



Handling Branch Misprediction

- ROB-Based architecture with RRF
  - Wait for the branch to commit
  - Use RRF to restore architectural state (renaming table)
- Merged Register File
  - Use a log to detect speculative values
  - May take time and can be optimized
  - Adjust renaming table and physical register identifiers

Exceptions

- Wait until commit stage
  - This guarantees that the instruction is not speculative
- All architectural states are stable now
- Flush all in-flight instructions
- Recovery the architectural states
- Redirect the front-end to the exception handler