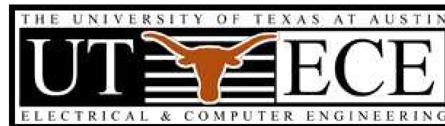
Fault Tolerance with

Microarchitecture-Based Introspection

Moinuddin K. Qureshi Onur Mutlu Yale N. Patt

Department of Electrical and Computer Engineering

The University of Texas at Austin

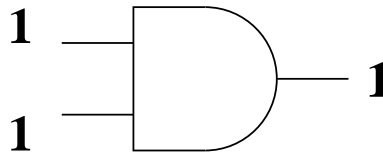


The Problem of Transient Faults

A transient fault occurs when a cosmic particle strikes and inverts the logical state of a device.

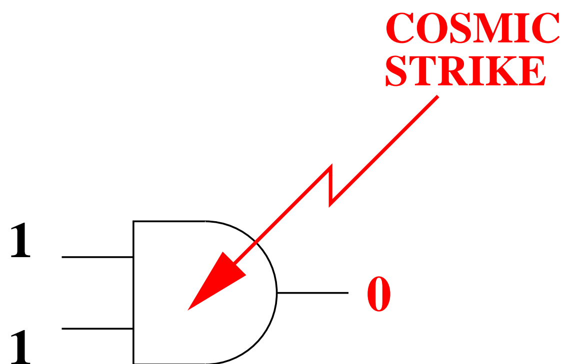
The Problem of Transient Faults

A transient fault occurs when a cosmic particle strikes and inverts the logical state of a device.



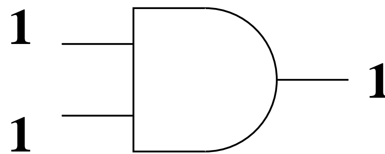
The Problem of Transient Faults

A transient fault occurs when a cosmic particle strikes and inverts the logical state of a device.



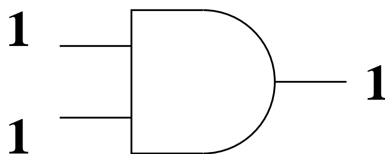
The Problem of Transient Faults

A transient fault occurs when a cosmic particle strikes and inverts the logical state of a device.



The Problem of Transient Faults

A transient fault occurs when a cosmic particle strikes and inverts the logical state of a device.



Can cause a fully verified, correct design to perform incorrectly.

The Need for Cost-Effective Transient-Fault Tolerance

- The rate of transient faults is expected to increase significantly. Processors will need some form of fault tolerance.
- However, different applications have different reliability requirements (e.g. server-apps v/s games). Users who do not require high reliability may not want to pay the overhead.
- Fault tolerant mechanisms with low hardware cost are attractive because they allow the designs to be used for a wide variety of applications.

Background on Fault Tolerance

- Storage structures are easy to protect with parity or ECC.
- Logic structures often need redundant execution:
  - Space redundancy
  - Time redundancy
- Space redundancy has high hardware overhead.
- Time redundancy has low hardware overhead but high performance overhead.

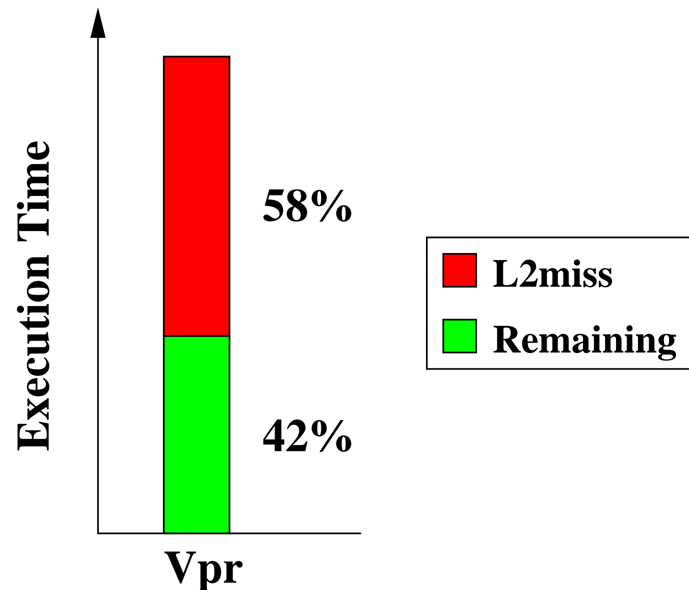
The performance overhead of time redundancy can be reduced if redundant execution is performed during idle periods.

Exploiting the Memory Wall for Idle Periods

- A long-latency miss typically stalls instruction processing.
- A significant proportion of execution time is spent waiting for memory.

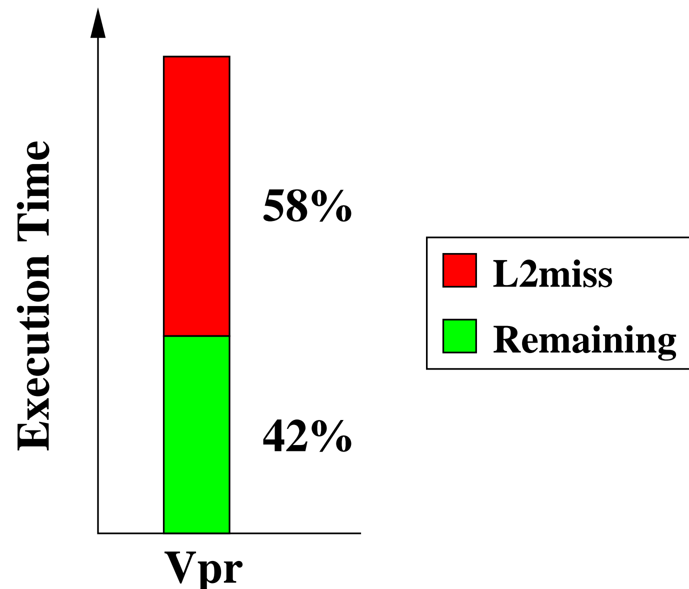
Exploiting the Memory Wall for Idle Periods

- A long-latency miss typically stalls instruction processing.
- A significant proportion of execution time is spent waiting for memory.



Exploiting the Memory Wall for Idle Periods

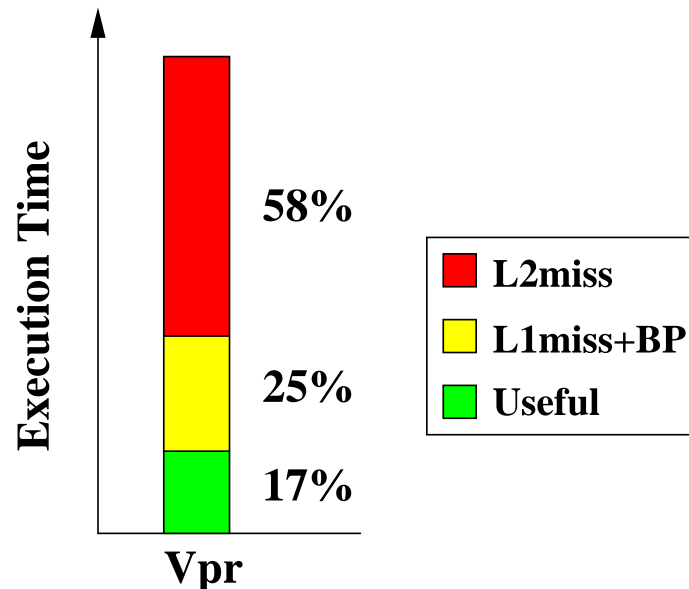
- A long-latency miss typically stalls instruction processing.
- A significant proportion of execution time is spent waiting for memory.



Redundant execution can leverage load values and branch prediction from primary execution.

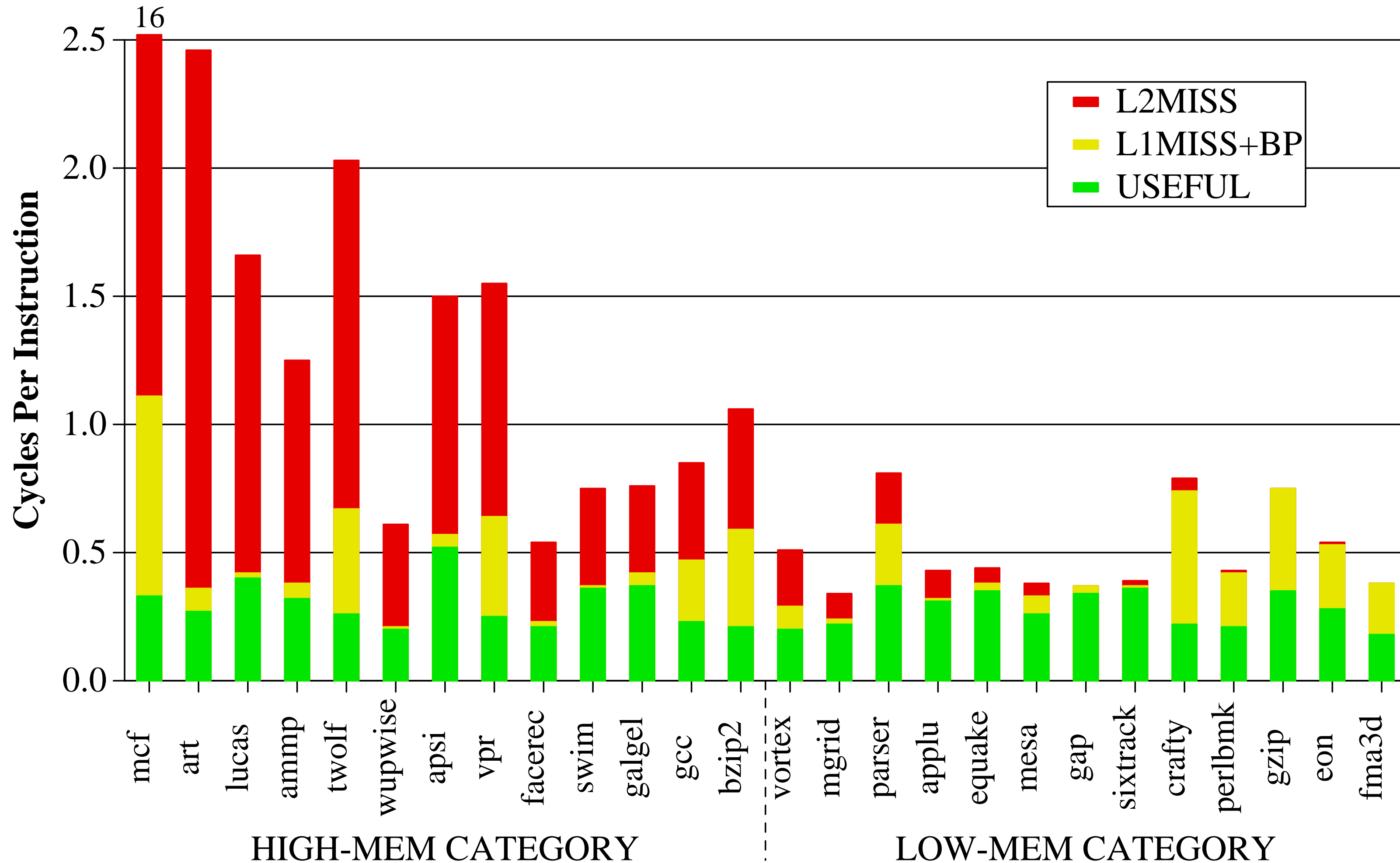
Exploiting the Memory Wall for Idle Periods

- A long-latency miss typically stalls instruction processing.
- A significant proportion of execution time is spent waiting for memory.

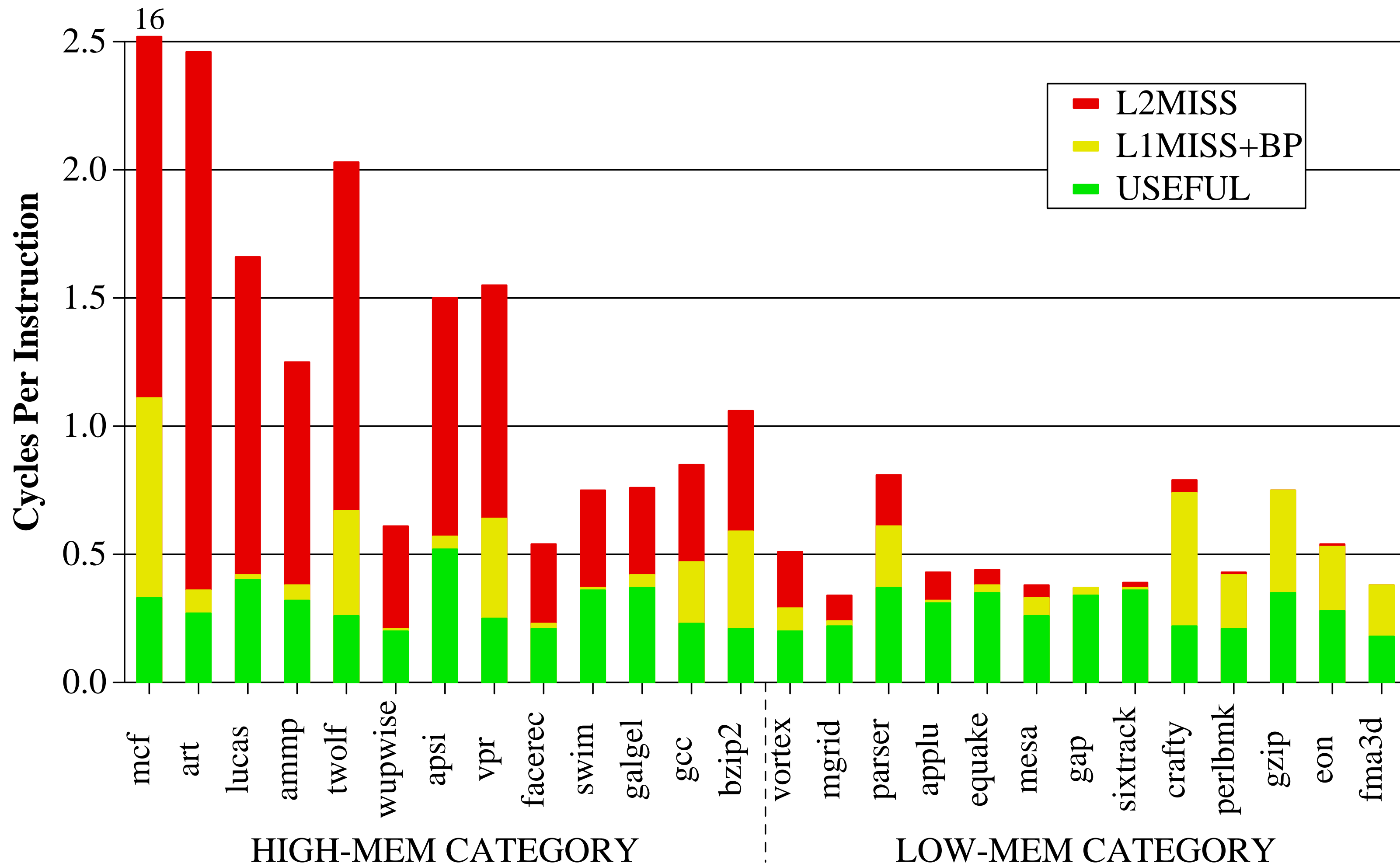


There is enough idle time to re-execute the application!

Motivation



Motivation



Can we design a time redundancy technique that utilizes memory waiting time for redundant execution?

Outline

- Introduction
- Concept of Introspection
- Implementation of MBI
- Evaluation of MBI
- Summary

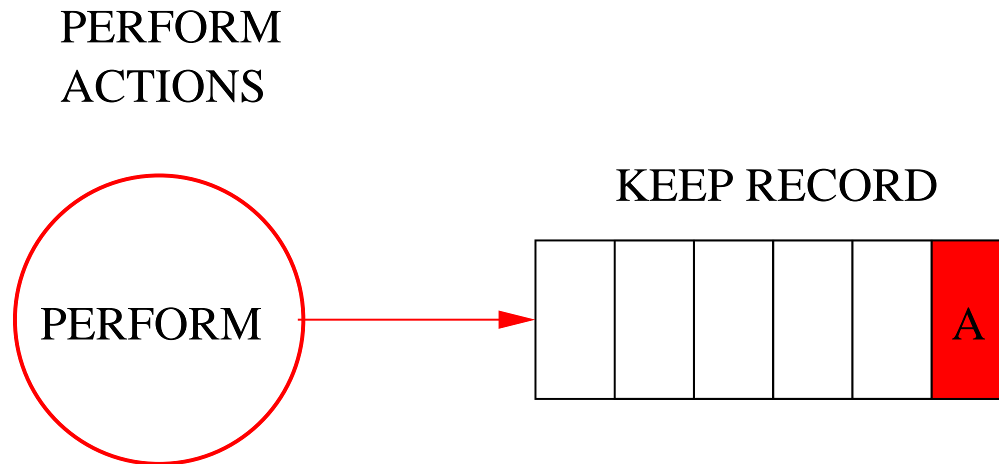
The Concept of Introspection

An example from the human domain:

PERFORM
ACTIONS

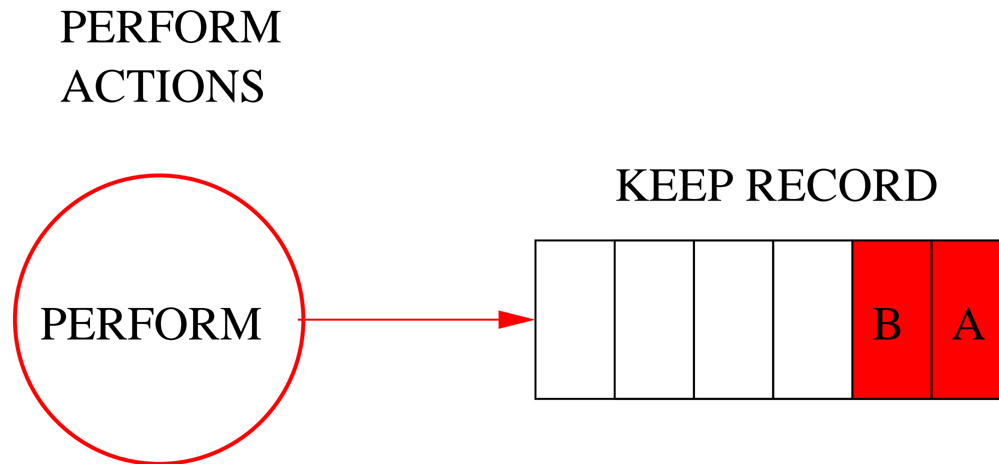
The Concept of Introspection

An example from the human domain:



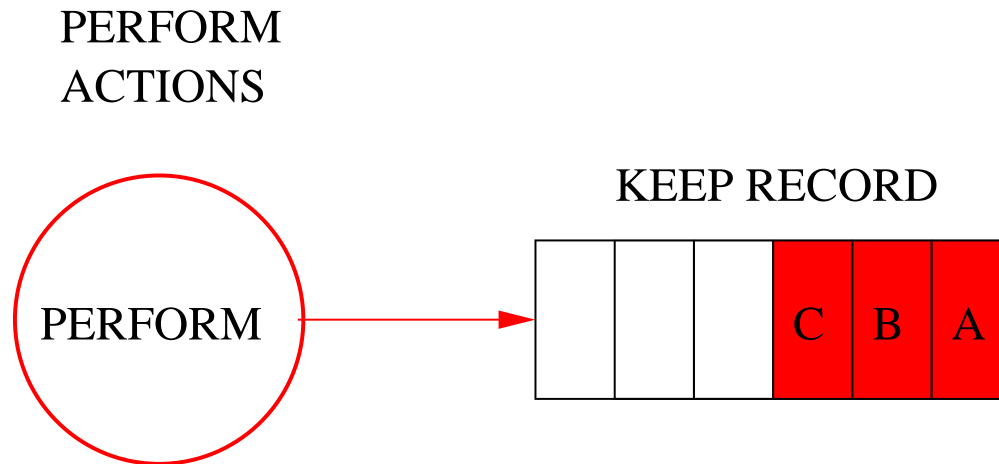
The Concept of Introspection

An example from the human domain:



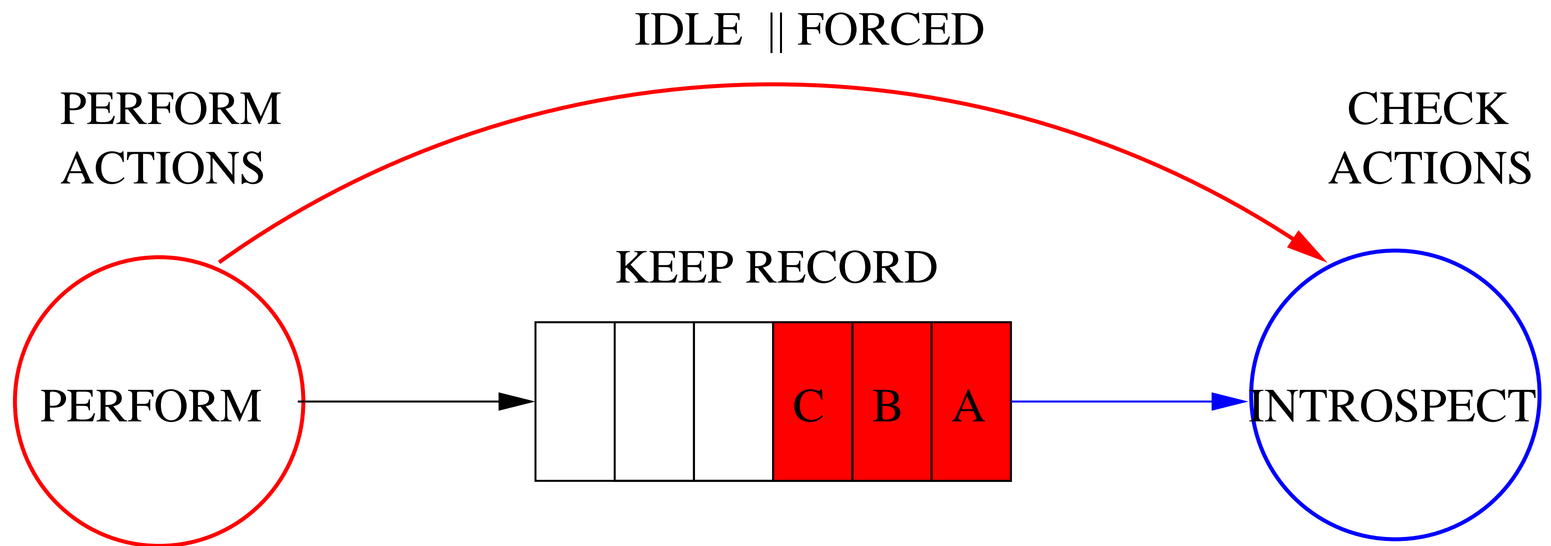
The Concept of Introspection

An example from the human domain:



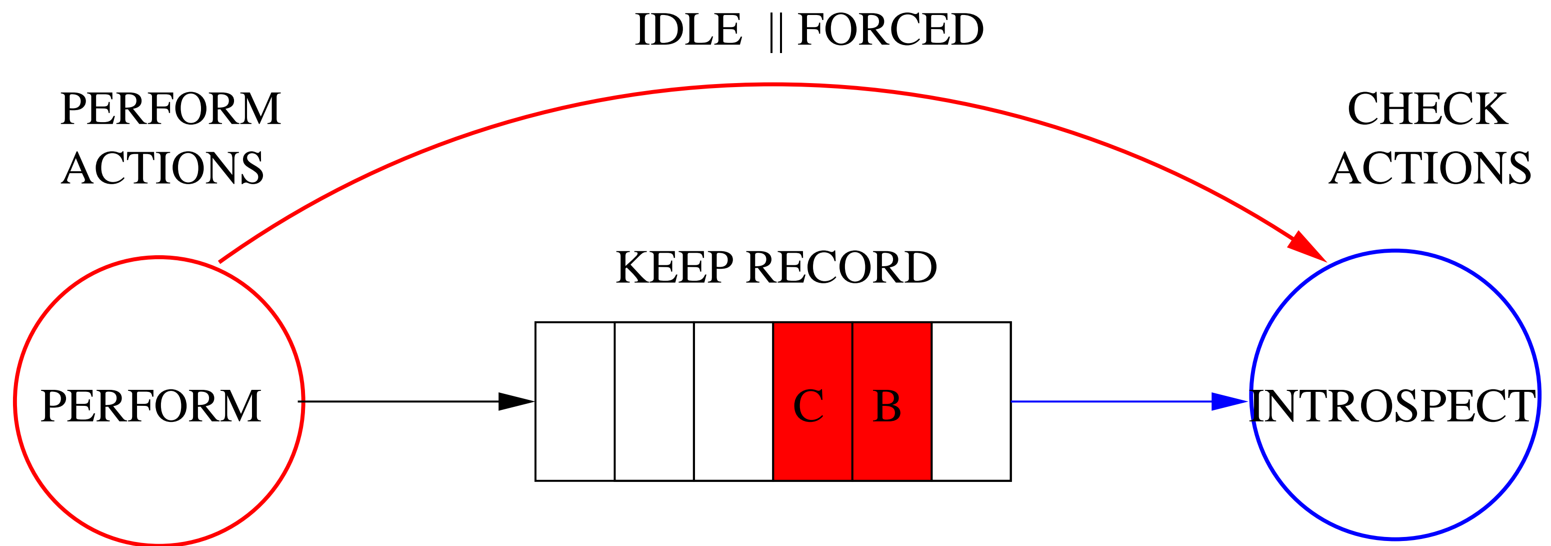
The Concept of Introspection

An example from the human domain:



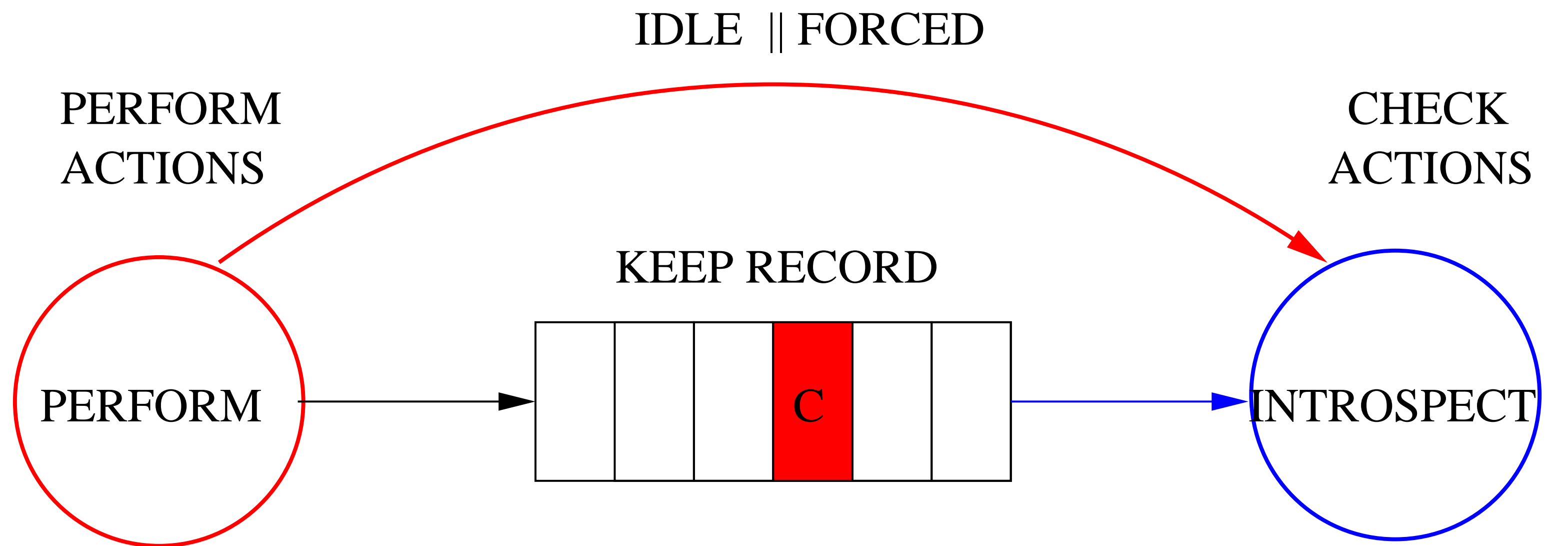
The Concept of Introspection

An example from the human domain:



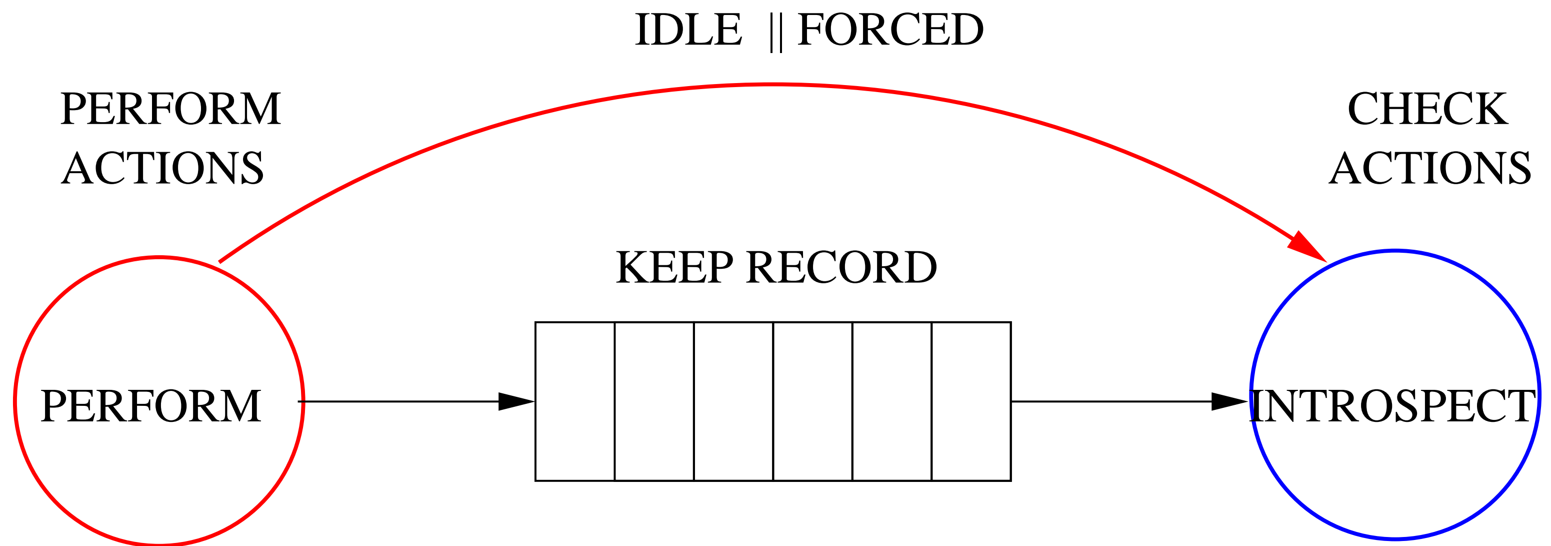
The Concept of Introspection

An example from the human domain:



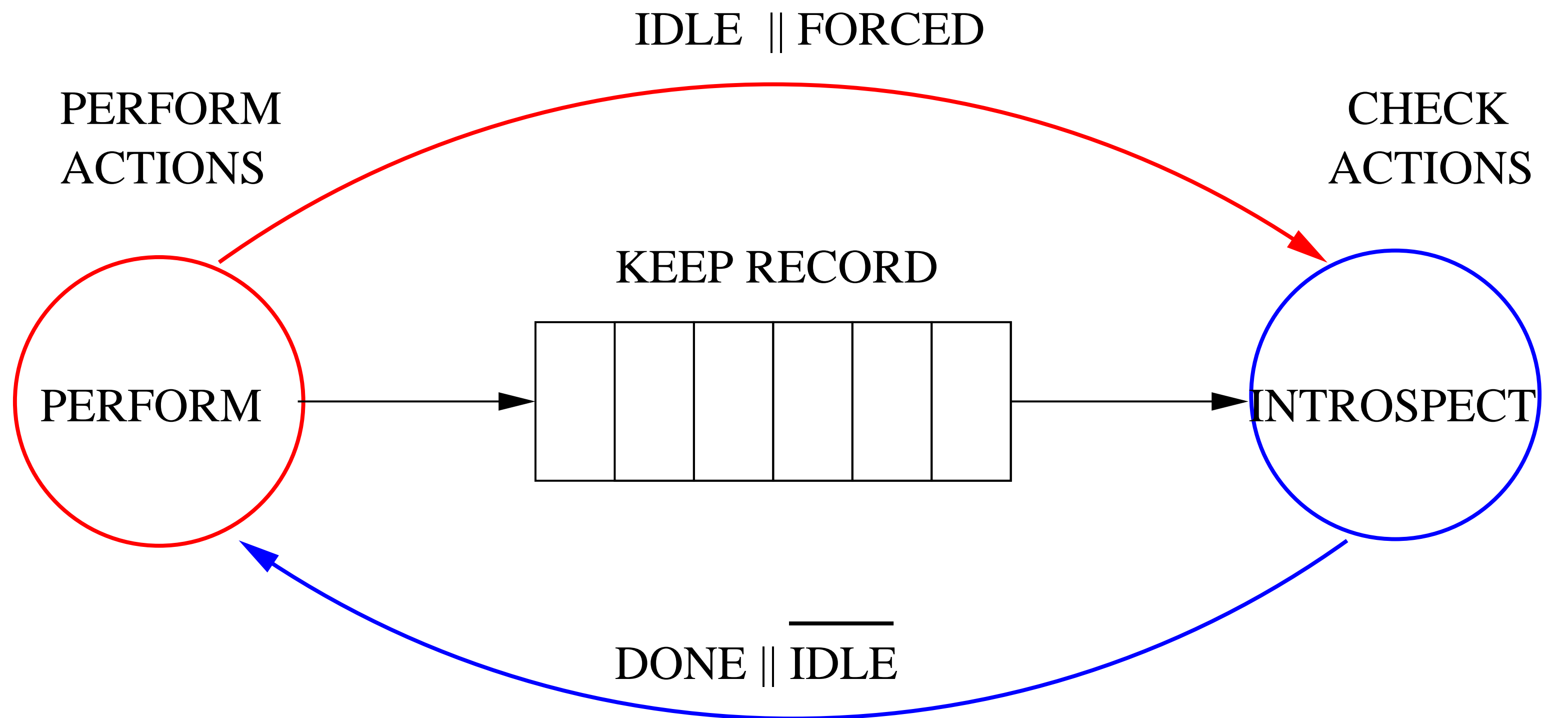
The Concept of Introspection

An example from the human domain:



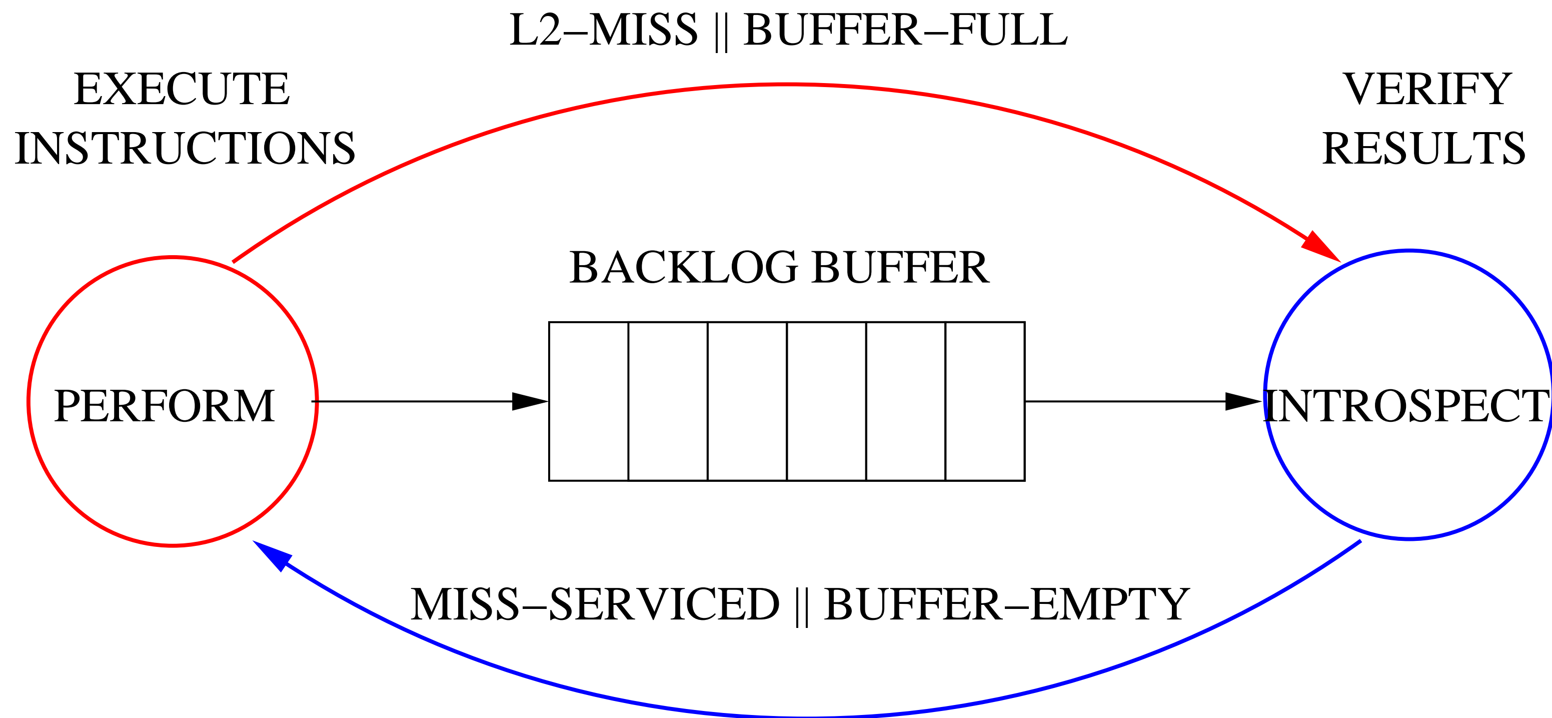
The Concept of Introspection

An example from the human domain:



The Concept of Introspection

Extending it to the microarchitecture domain:

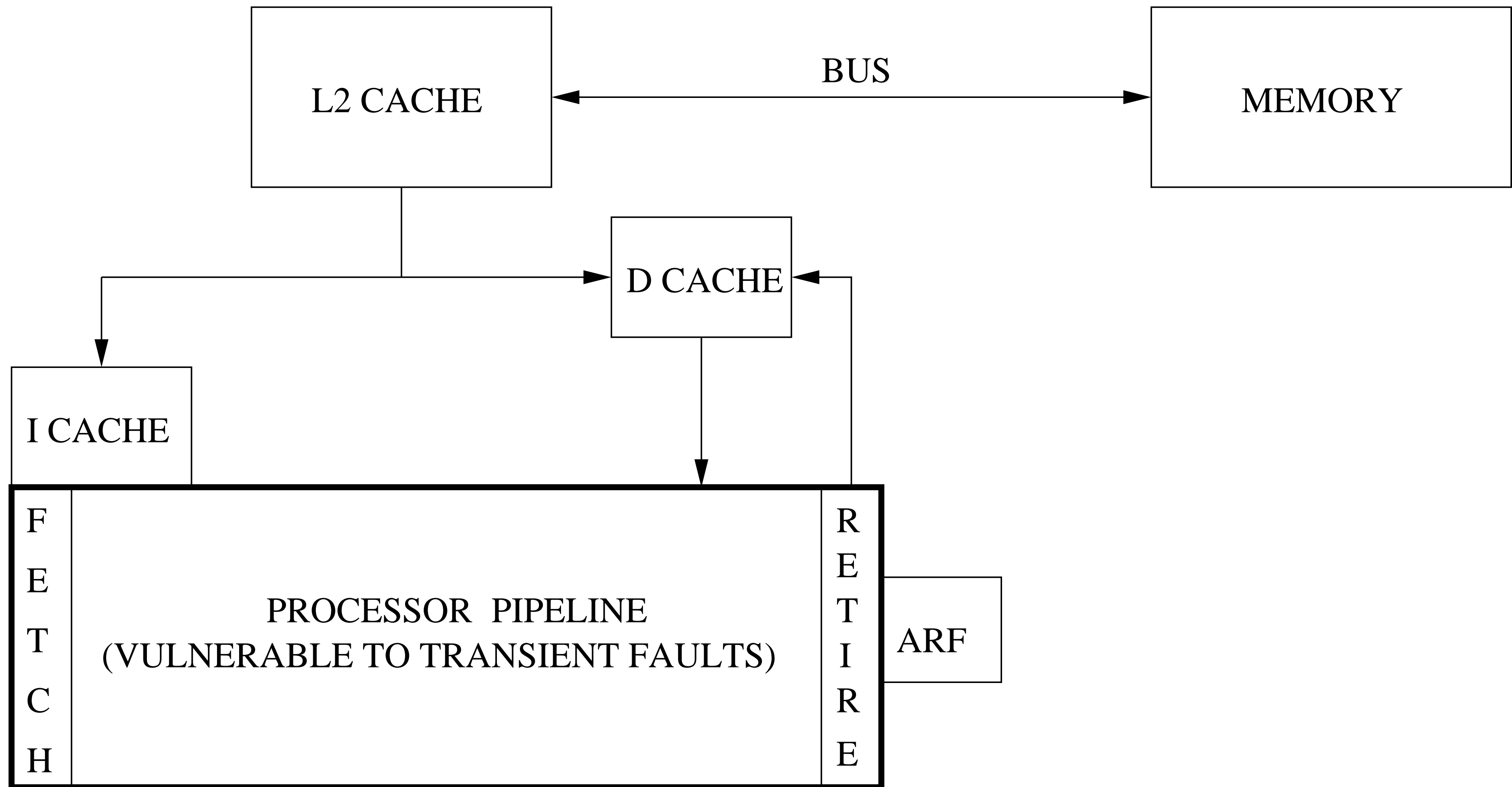


Microarchitecture-Based Introspection (MBI)

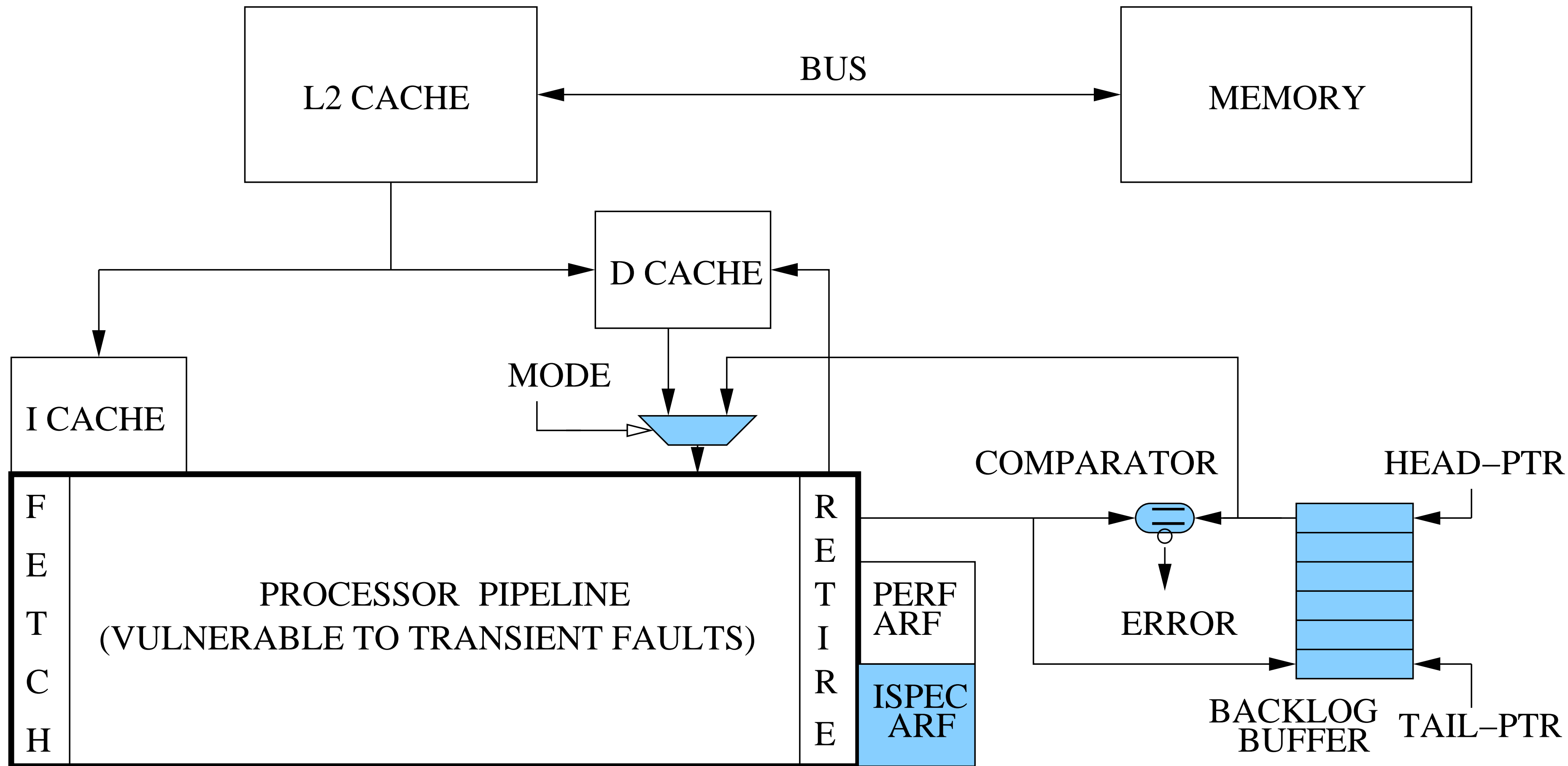
Outline

- Introduction
- Concept of Introspection
- Implementation of MBI
- Evaluation of MBI
- Summary

Additional Structures



Additional Structures



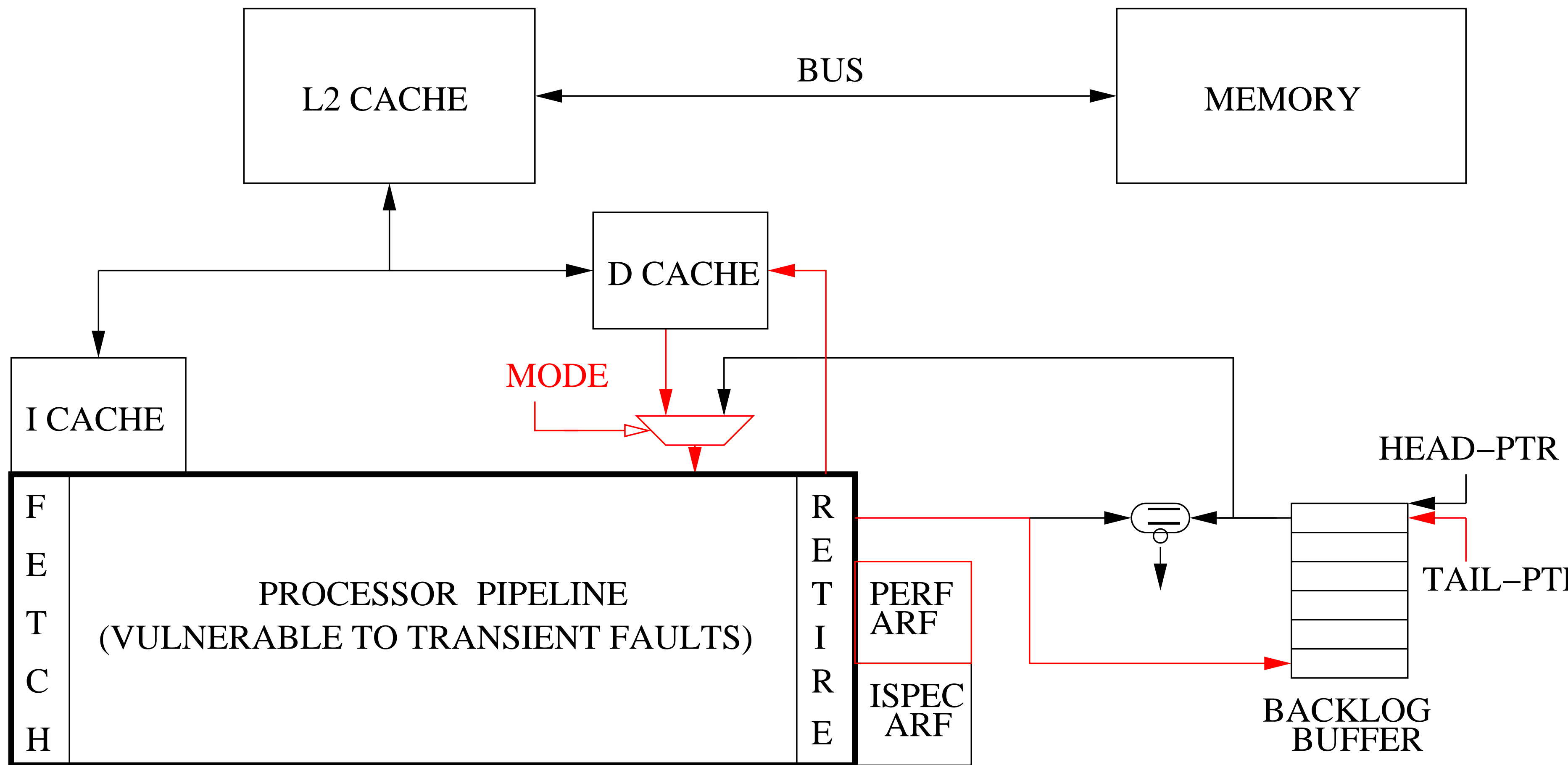
Mechanism

Two modes: Performance and Introspection

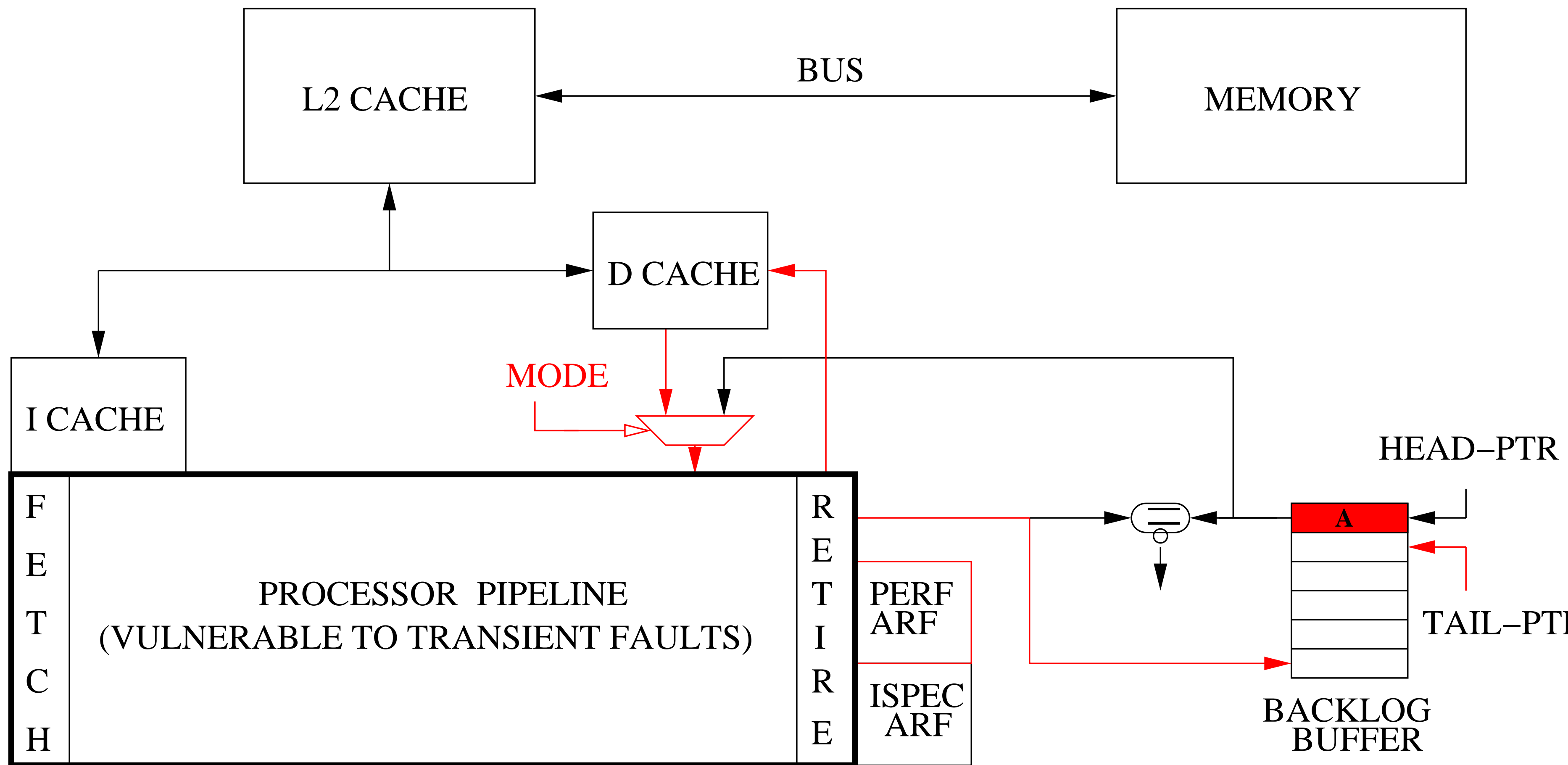
Four cases:

- Operation in performance mode
- Switching from performance mode to introspection mode
- Operation in introspection mode
- Switching from introspection mode to performance mode

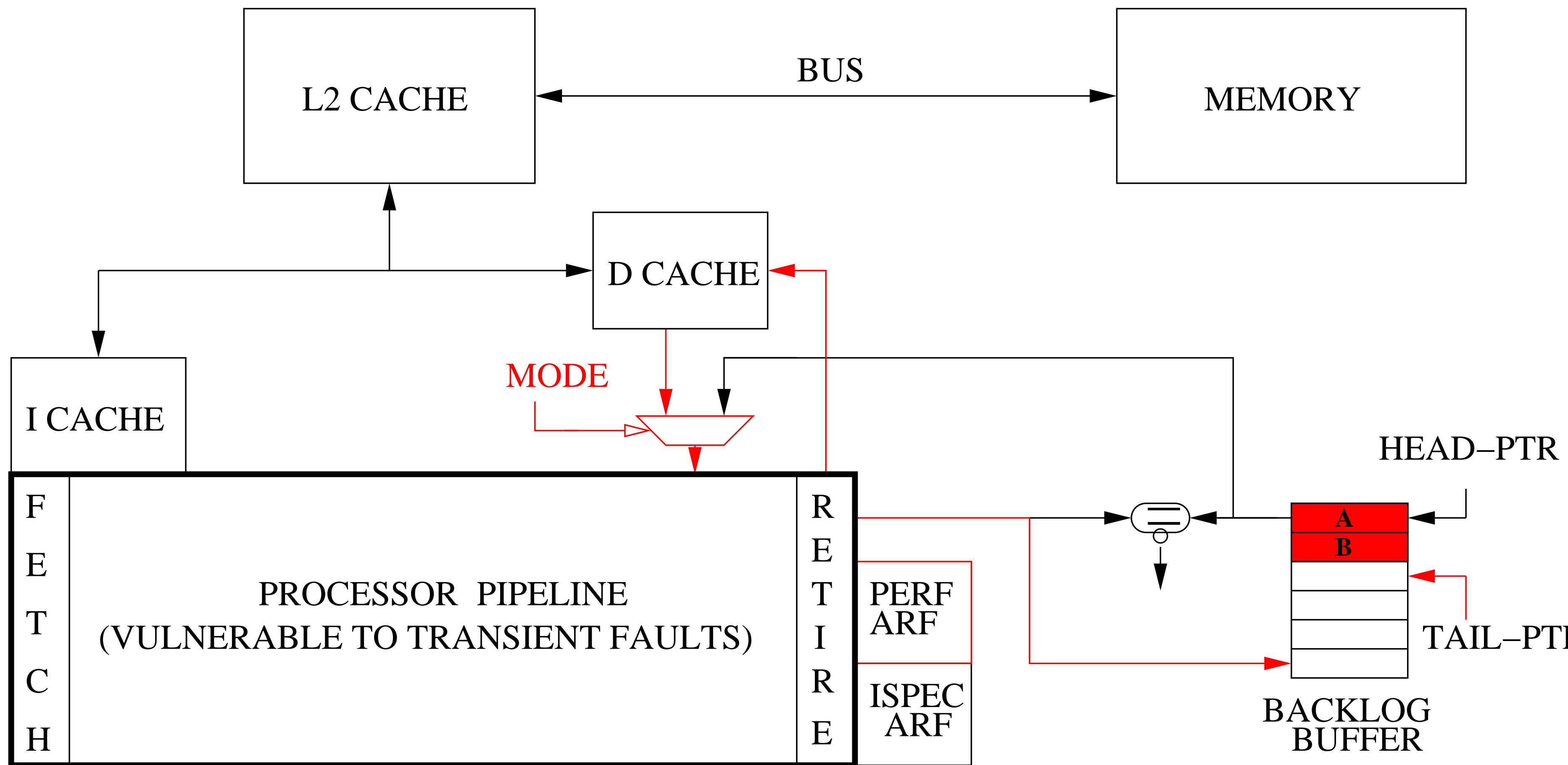
Operation in Performance Mode



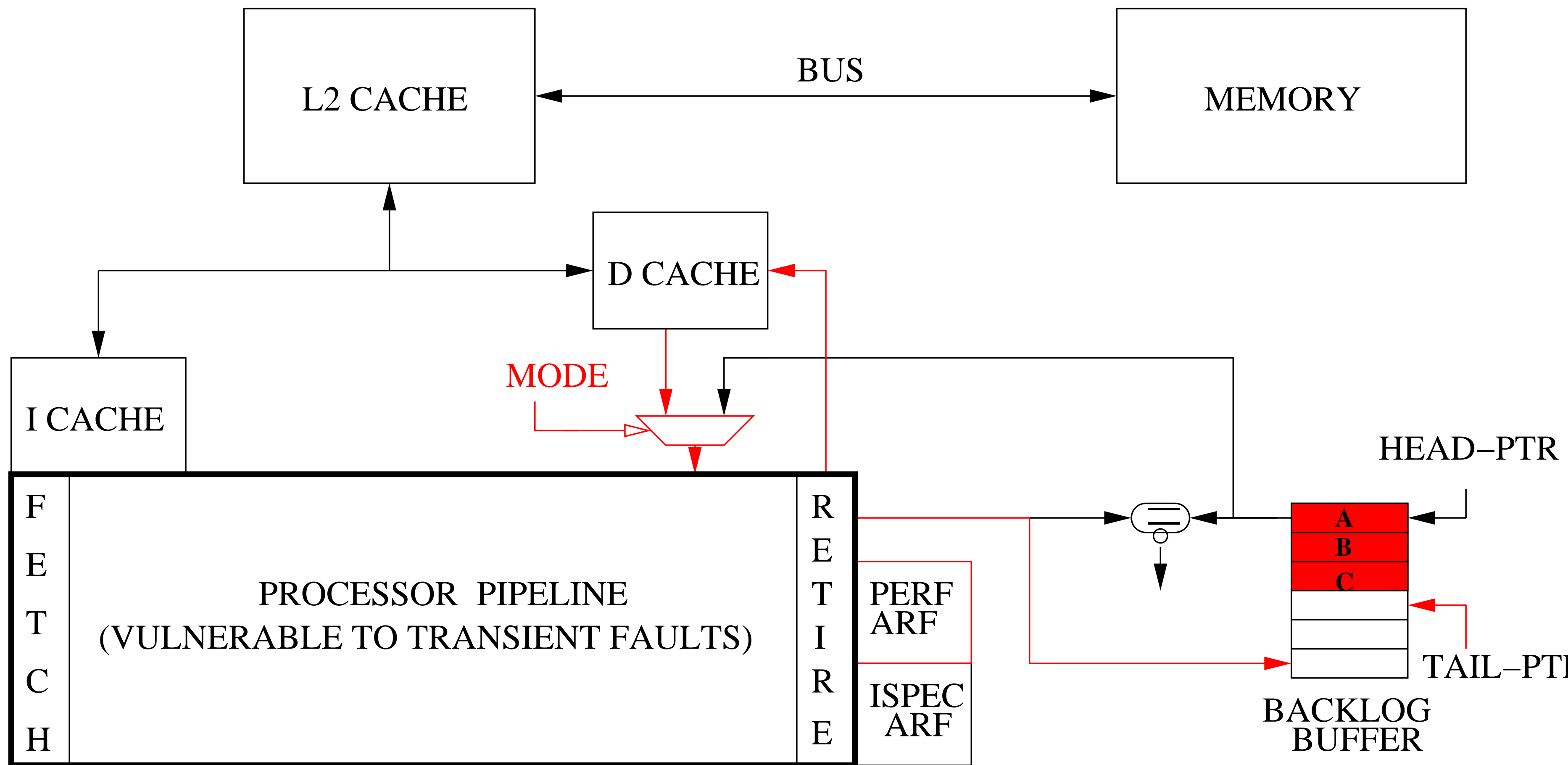
Operation in Performance Mode



Operation in Performance Mode



Operation in Performance Mode

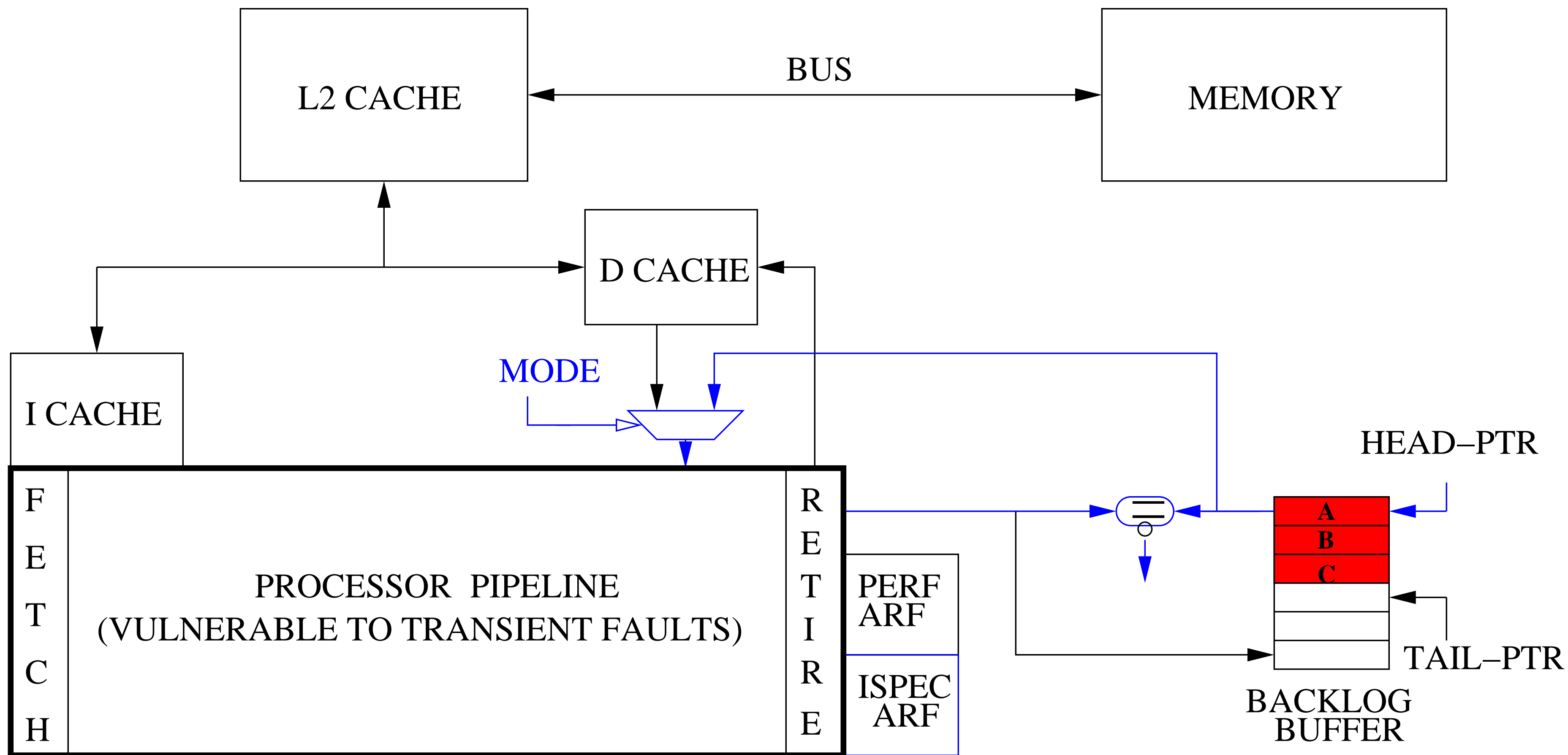


Switching from Performance Mode to Introspection Mode

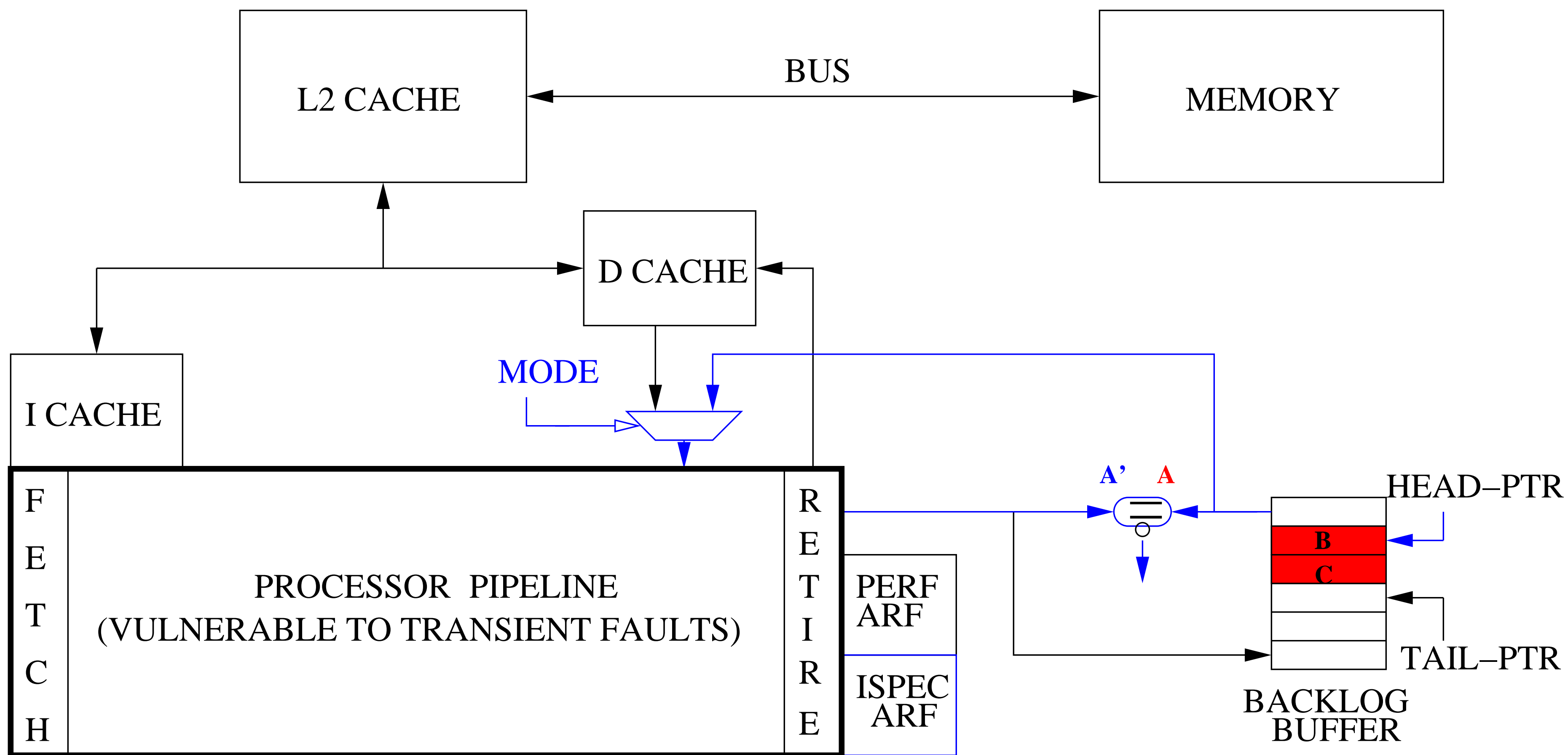
- Enter Introspection on:
  - Long latency L2 miss (wait for 30 cycles)
  - Backlog buffer full
  - System call or Interaction with I/O

- The process involves:
  - Flushing the pipeline
  - Switching the mode bit
  - Fetching instructions from the PC of ISPEC-ARF

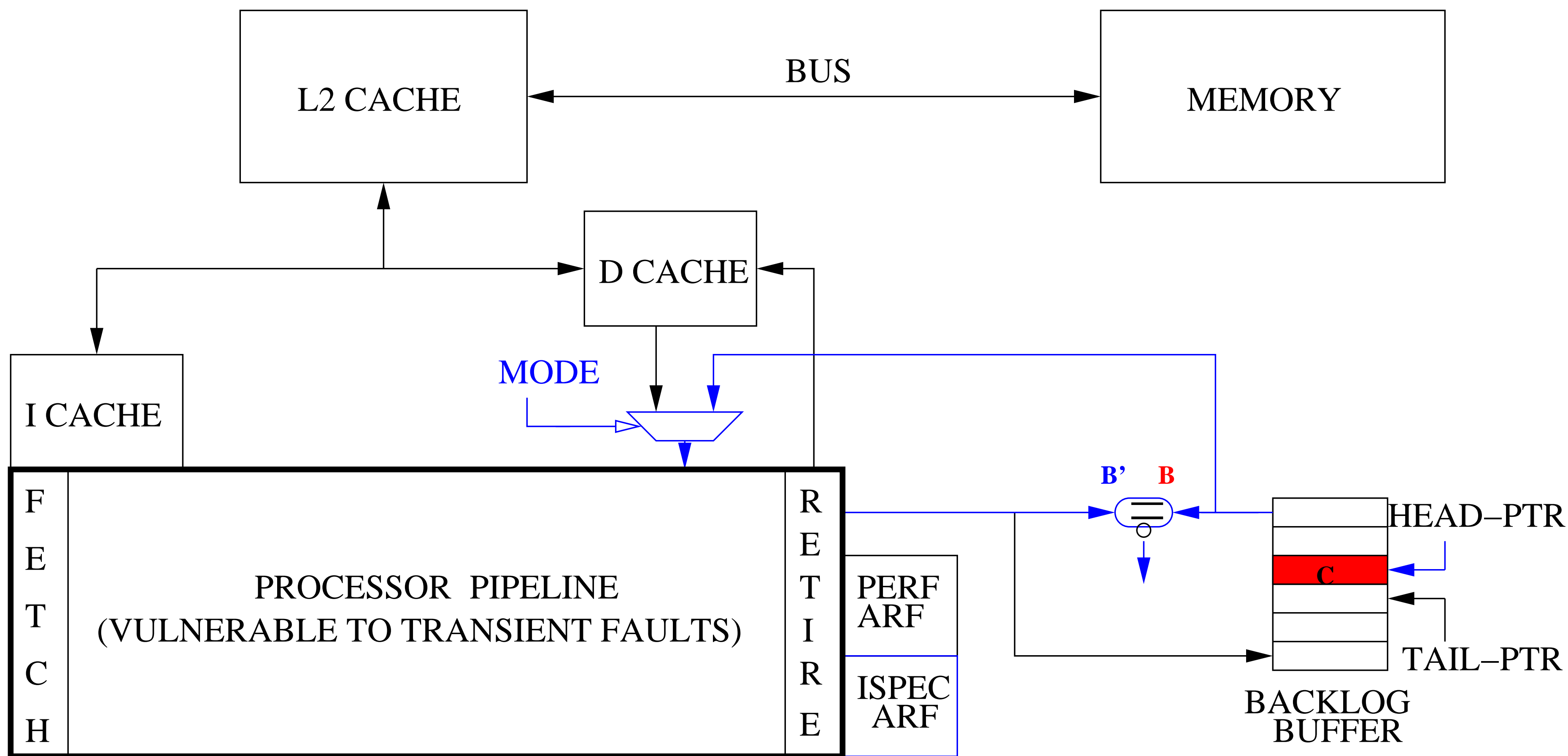
Operation in Introspection Mode



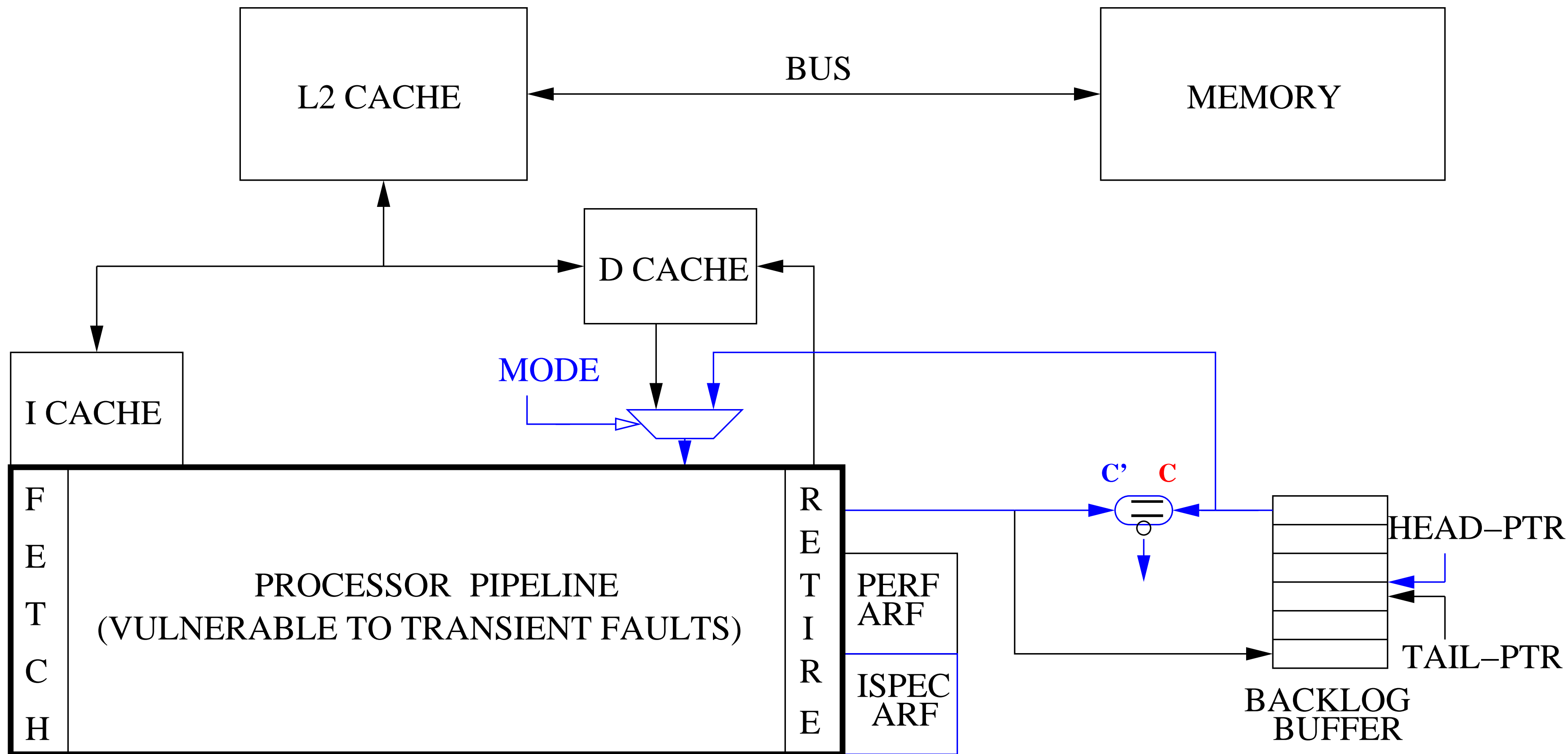
Operation in Introspection Mode



Operation in Introspection Mode



Operation in Introspection Mode



Switching from Introspection Mode to Performance Mode

- Exit Introspection mode when:
  - L2 miss is serviced
  - Backlog buffer is empty

- The process involves:
  - Flushing the pipeline
  - Switching the mode bit
  - Fetching instructions from the PC of PERF-ARF

Outline

- Introduction
- Concept of Introspection
- Implementation of MBI
- Evaluation of MBI
- Summary

Methodology

- Pipeline: 20 stages, 8-wide issue with out-of-order execution
- L2 cache: Unified, 1MB, 15-cycle hit
- Memory: 400 cycles (32 banks)
- Prefetcher: Streaming with up to 32 streams

- Backlog Buffer contains 2k entries

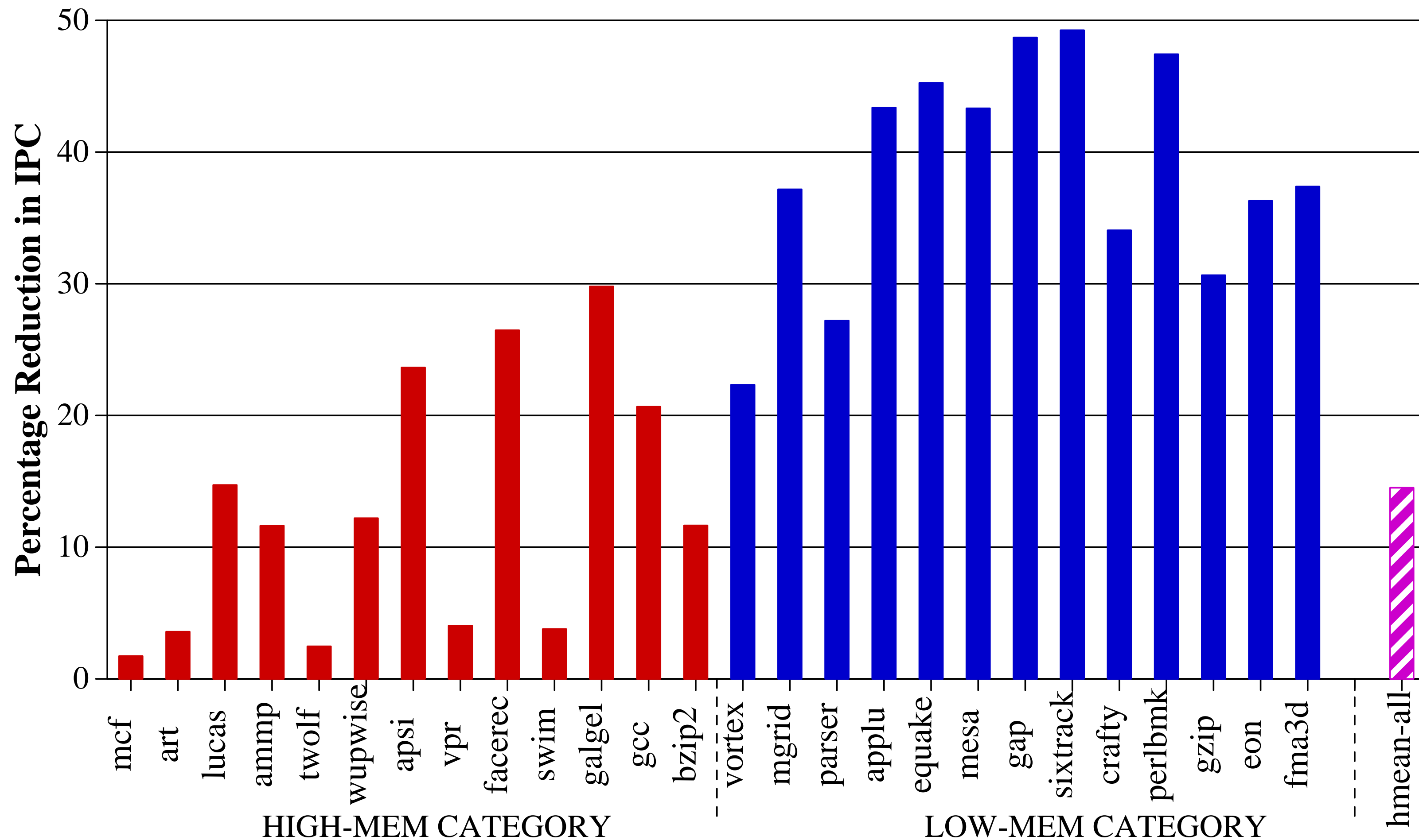
- Benchmarks: SPEC CPU2000

Performance Overhead

- Forced-Introspection penalty
- Pipeline-Fill penalty

Performance Overhead

- Forced-Introspection penalty
- Pipeline-Fill penalty



Other Overheads: Storage and Error-Detection Latency

- Storage overhead:
  Depends on the Number of Entries in the Backlog Buffer

Num. Entries	Storage Cost	IPC overhead
1k	8.5kB	16.1%
2k	16.5kB	14.5%
4k	32.5kB	13.9%

- Error Detection Latency
  - Average: 682 cycles
  - Worst case: 36183 cycles
    (impact depends on the error correction mechanism)

Outline

- Introduction
- Concept of Introspection
- Implementation of MBI
- Evaluation of MBI
- Summary

Summary

- Transient fault rate is increasing. Processors will need some form of fault tolerance.
- Fault tolerant mechanisms with low hardware cost are attractive because they allow the designs to be used for a wide variety of applications.
- MBI utilizes the idle time during long latency cache misses for redundant execution. MBI has low hardware overhead and provides coverage for the entire pipeline.
- The time redundancy of MBI incurs an average IPC overhead of 14.5%.

Questions

Extra Slides

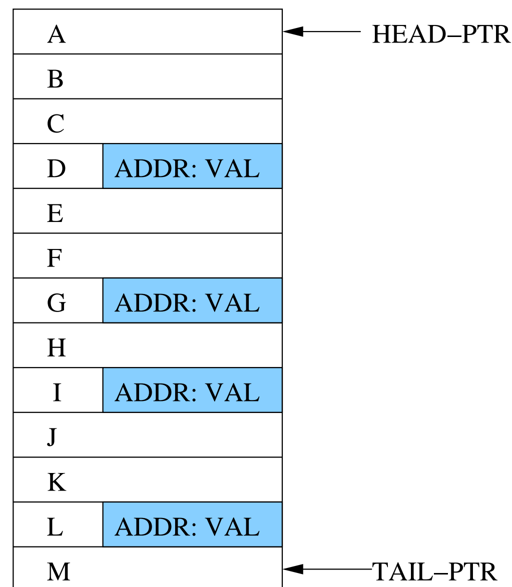
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...

Implementation of MBI: Recovery

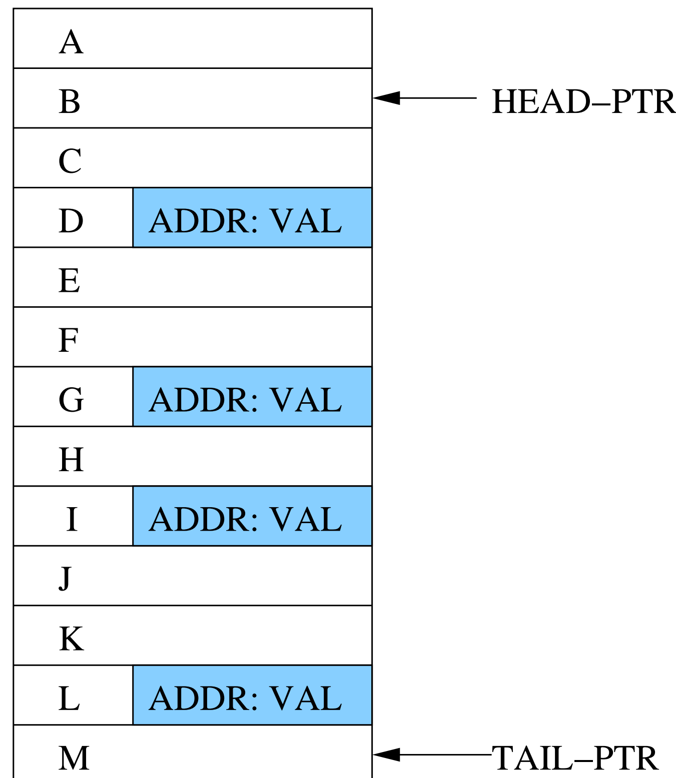
MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...

Recover register state by copying ISPEC-ARF to PERF-ARF
Recover memory state by undoing younger store operations.



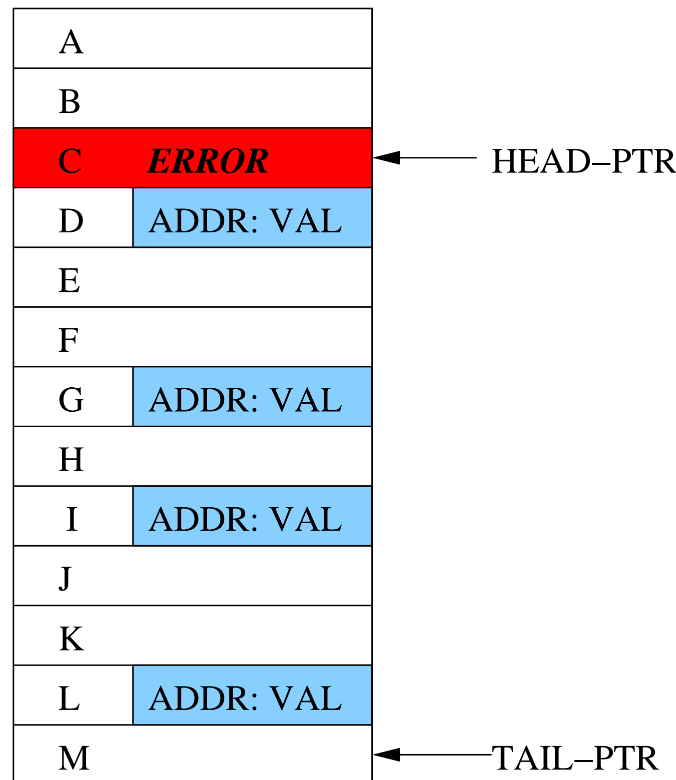
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



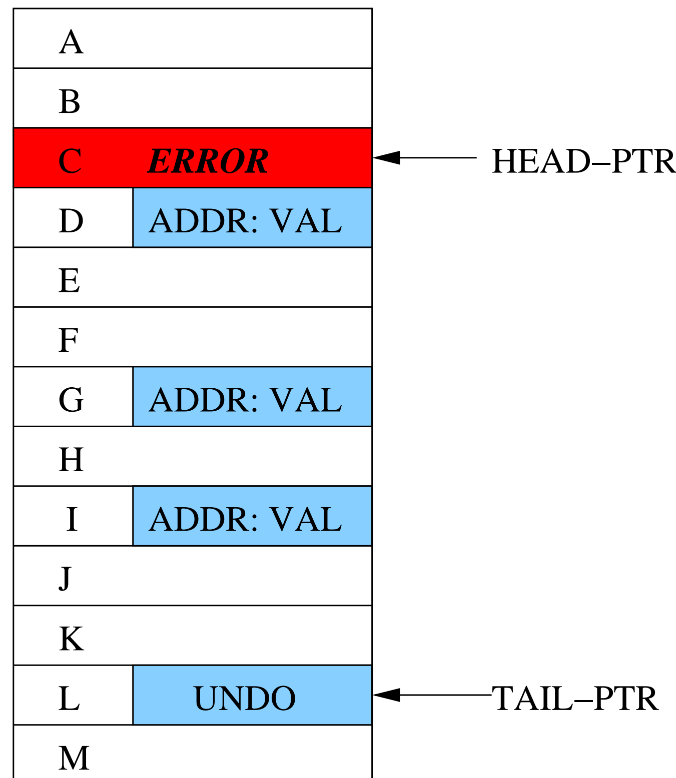
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



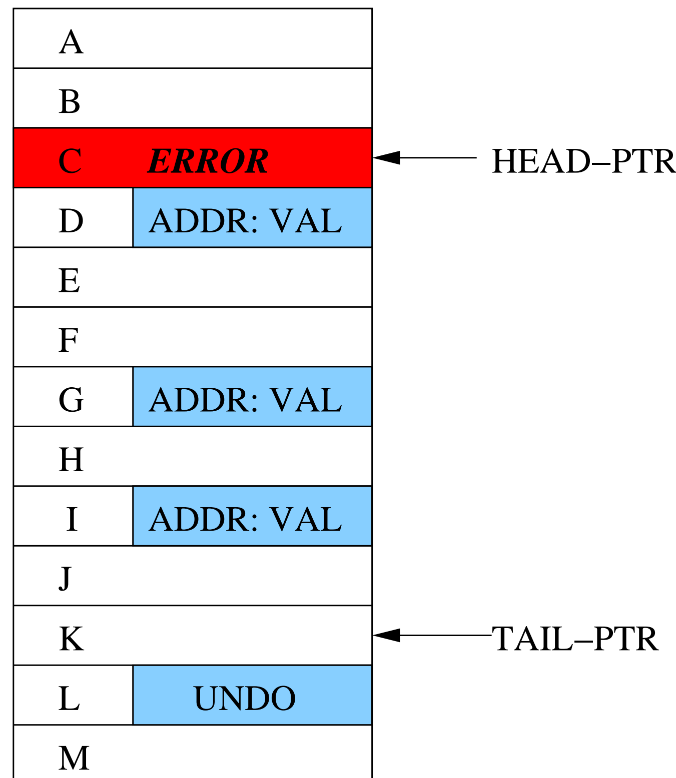
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



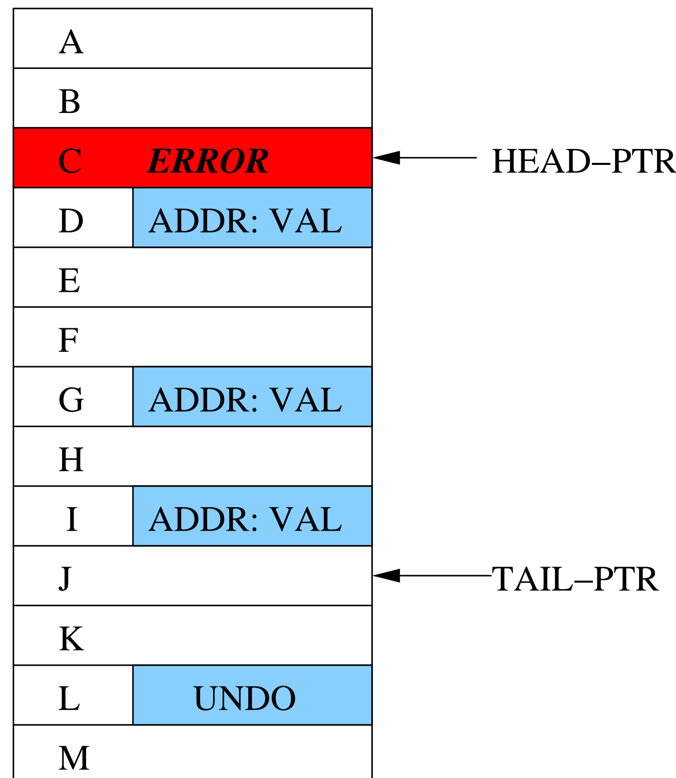
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



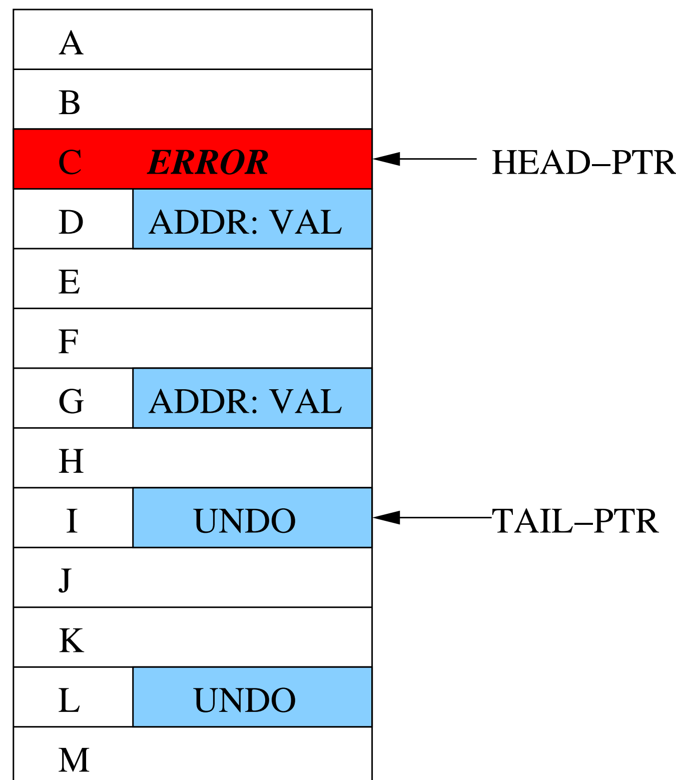
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



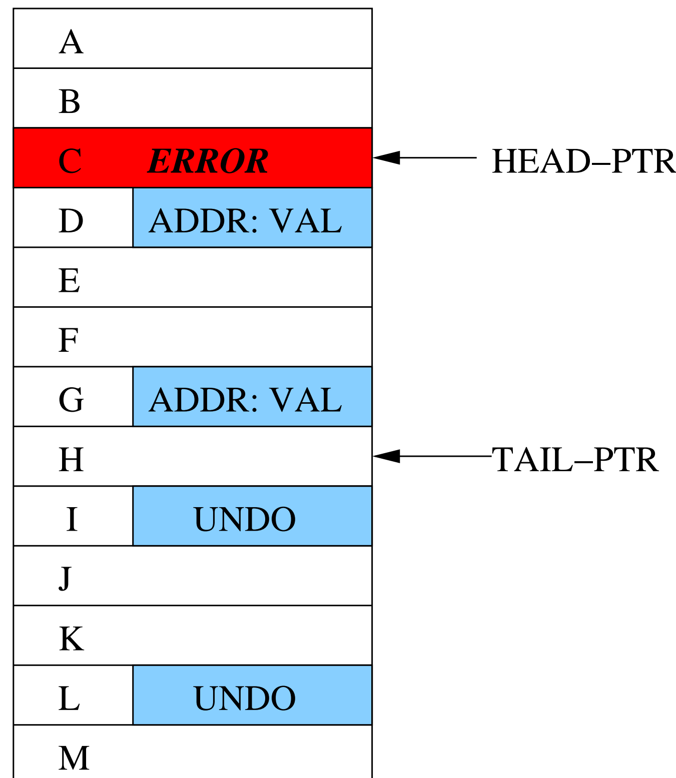
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



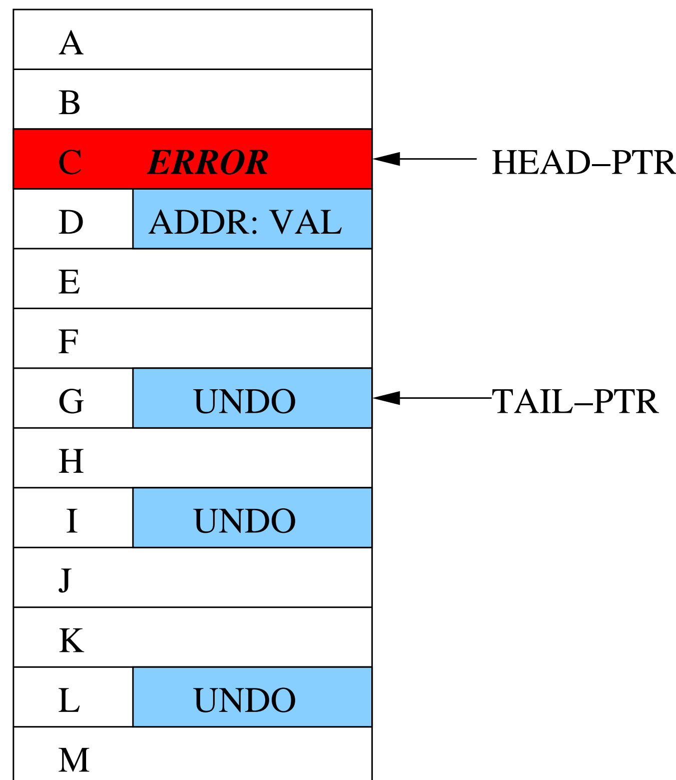
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



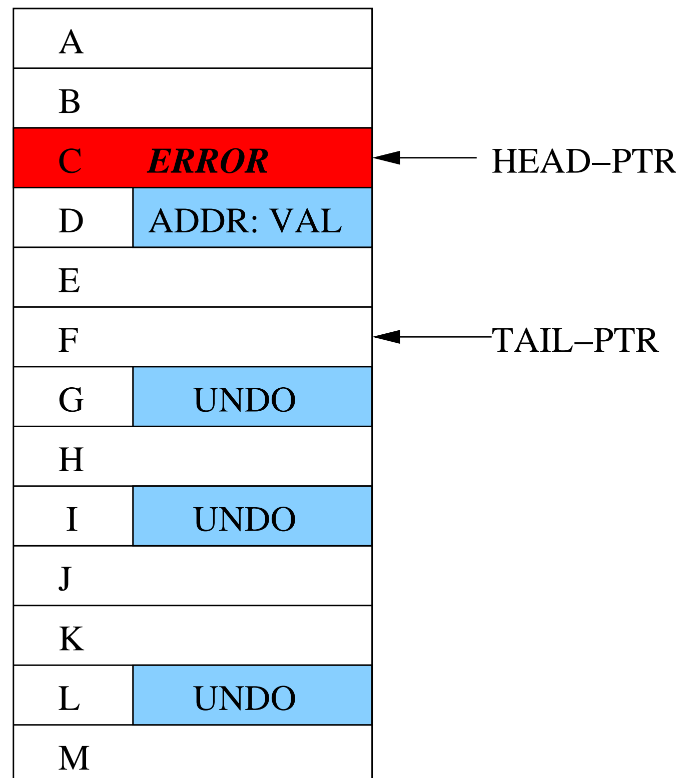
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



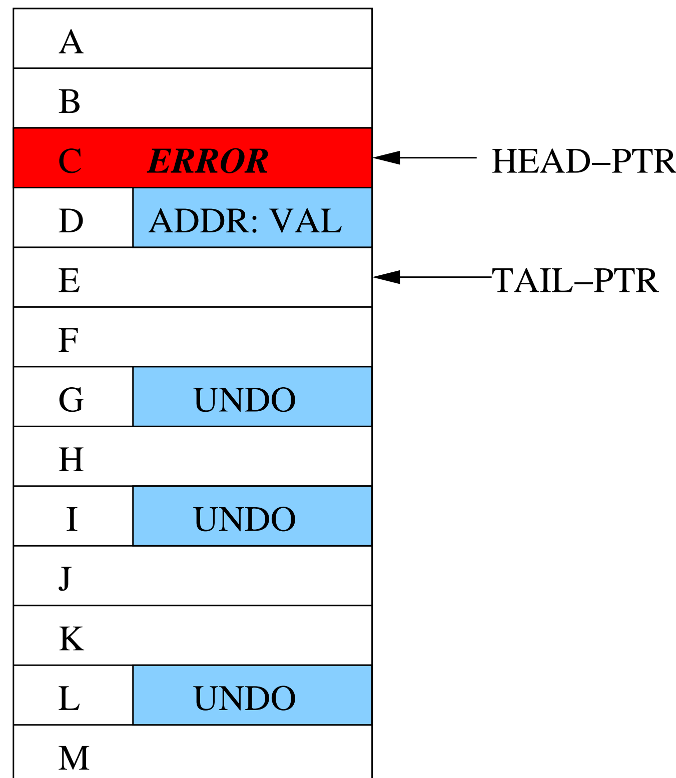
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



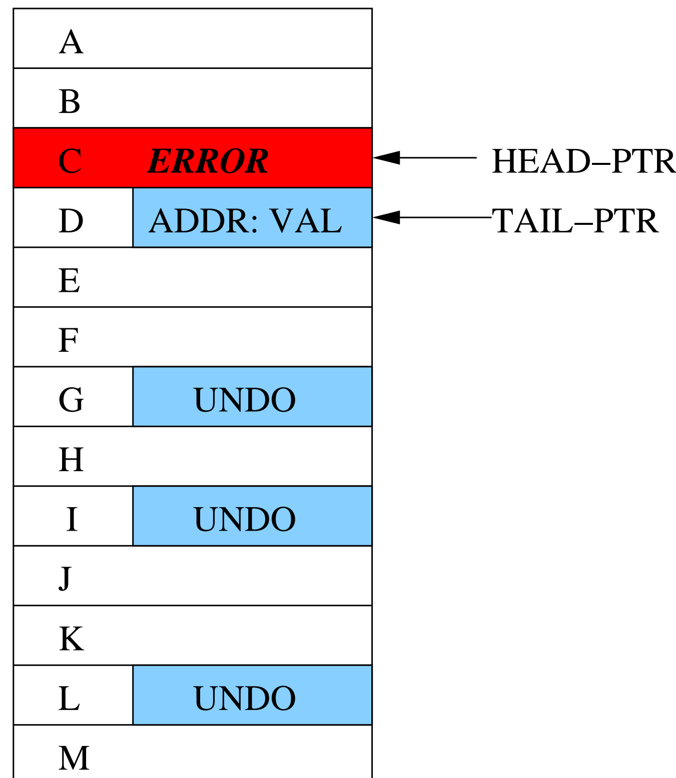
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



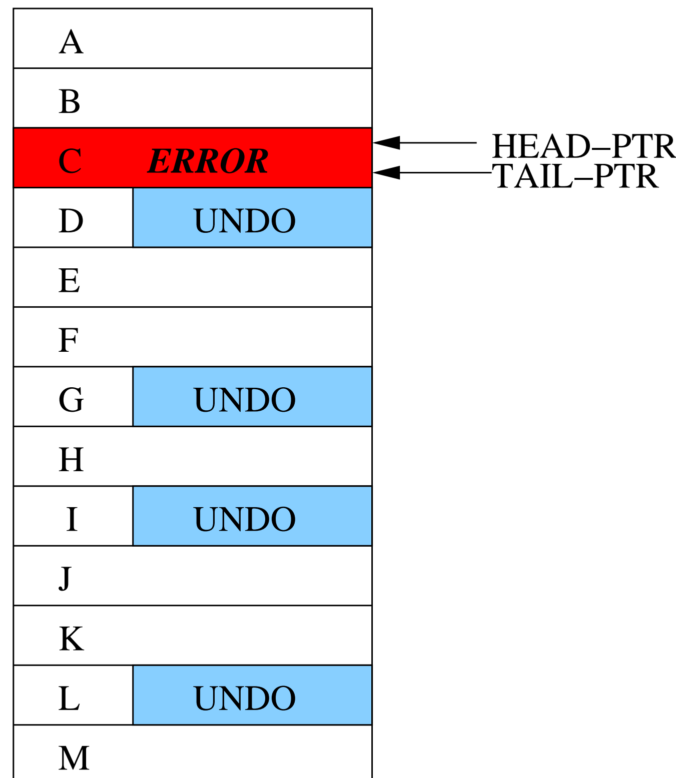
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



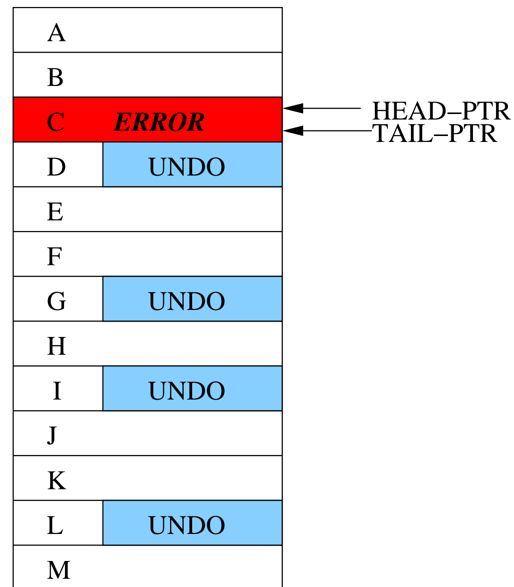
Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



Implementation of MBI: Recovery

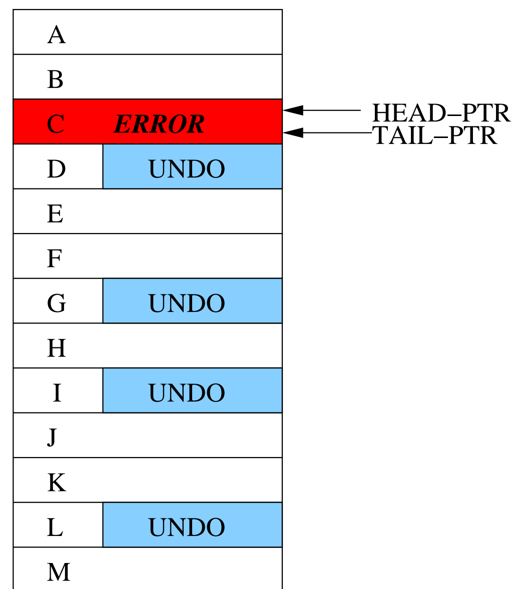
MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



No effect of the error causing instruction or any later instruction is architecturally visible.

Implementation of MBI: Recovery

MBI is primarily a fault detection mechanism.
Possible recovery mechanisms include fail-safe and checkpointing.
Can also use undo mechanism...



Needs special handling in case of multiprocessors.
Data about error detection latency is here

Related work

- Dual modulo redundancy: [e.g. IBM-z390, Compaq-Himalaya]
  - High hardware cost

Related work

- Dual modulo redundancy: [e.g. IBM-z390, Compaq-Himalaya]
  - High hardware cost
- DIVA [Austin MICRO'99]
  - Needs a checker processor – substantial investment
  - Does not provide full coverage for the pipeline

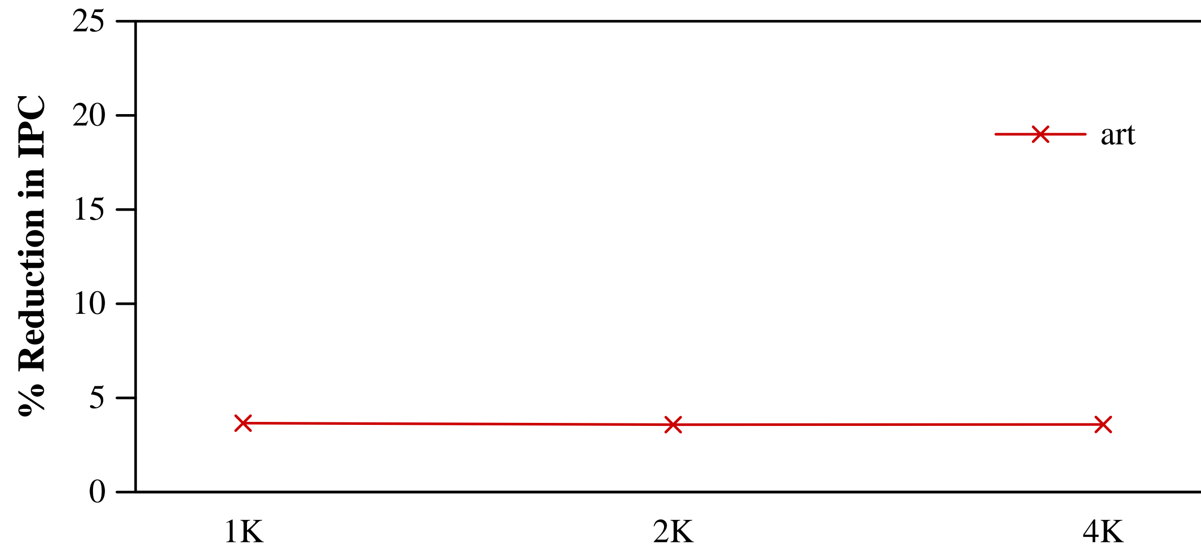
Related work

- Dual modulo redundancy: [e.g. IBM-z390, Compaq-Himalaya]
  - High hardware cost
- DIVA [Austin MICRO'99]
  - Needs a checker processor – substantial investment
  - Does not provide full coverage for the pipeline
- Software based techniques: [EDDI, SWIFT, etc.]
  - May need recompilation (not always possible)
  - Cannot take advantage of run-time information (e.g. cache miss)

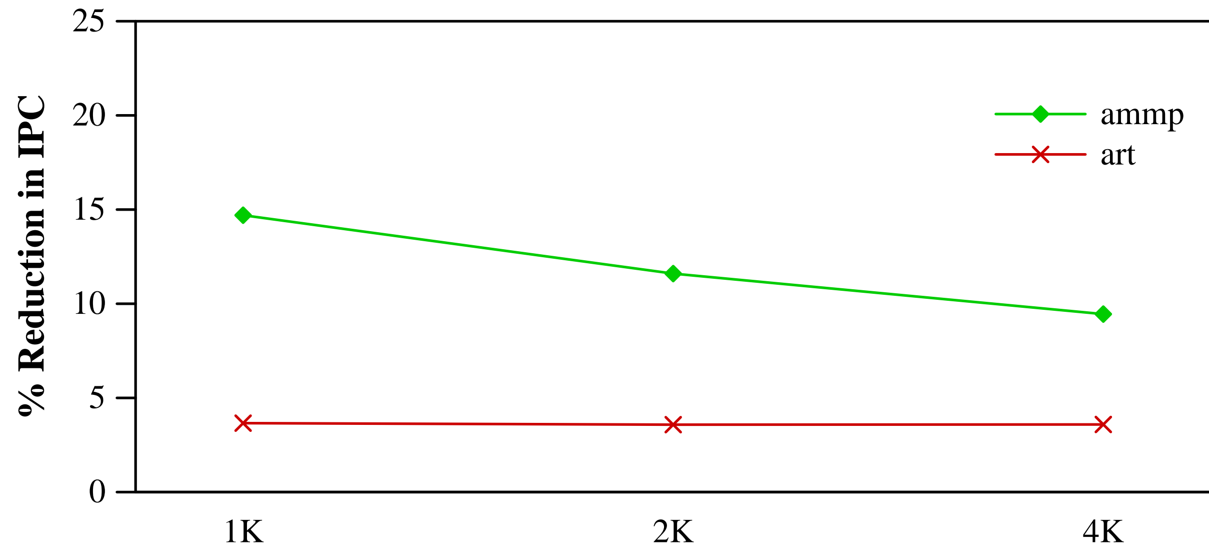
Related work

- Dual modulo redundancy: [e.g. IBM-z390, Compaq-Himalaya]
  - High hardware cost
- DIVA [Austin MICRO'99]
  - Needs a checker processor – substantial investment
  - Does not provide full coverage for the pipeline
- Software based techniques: [EDDI, SWIFT, etc.]
  - May need recompilation (not always possible)
  - Cannot take advantage of run-time information (e.g. cache miss)
- SMT based techniques: [e.g. RMT - ISCA'00]
  - Need a fine-grained multi-threaded machine
  - Reduces throughput

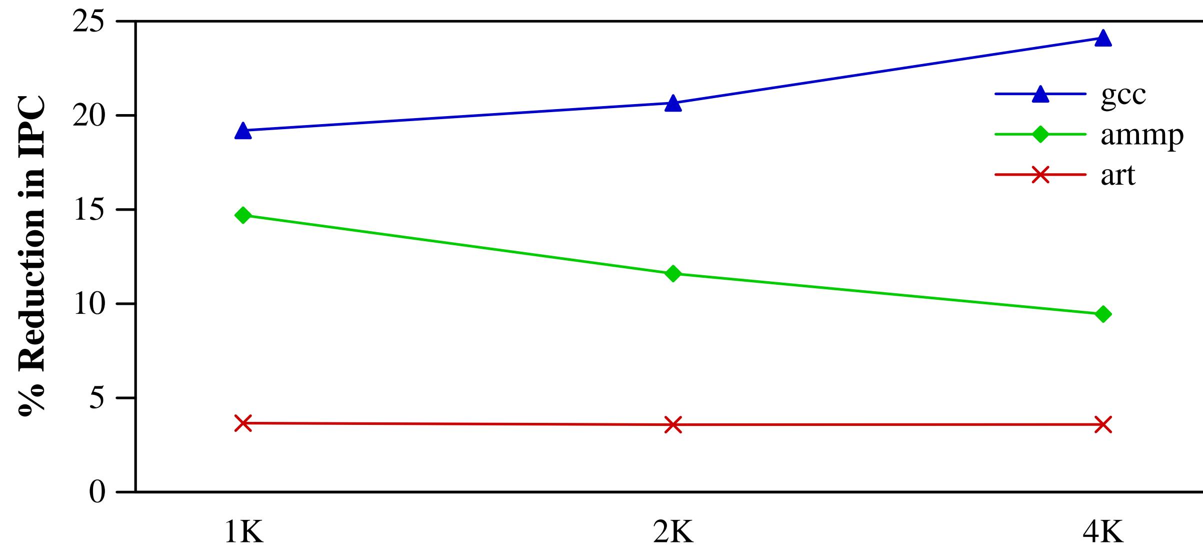
Sensitivity to Entries in the Backlog Buffer



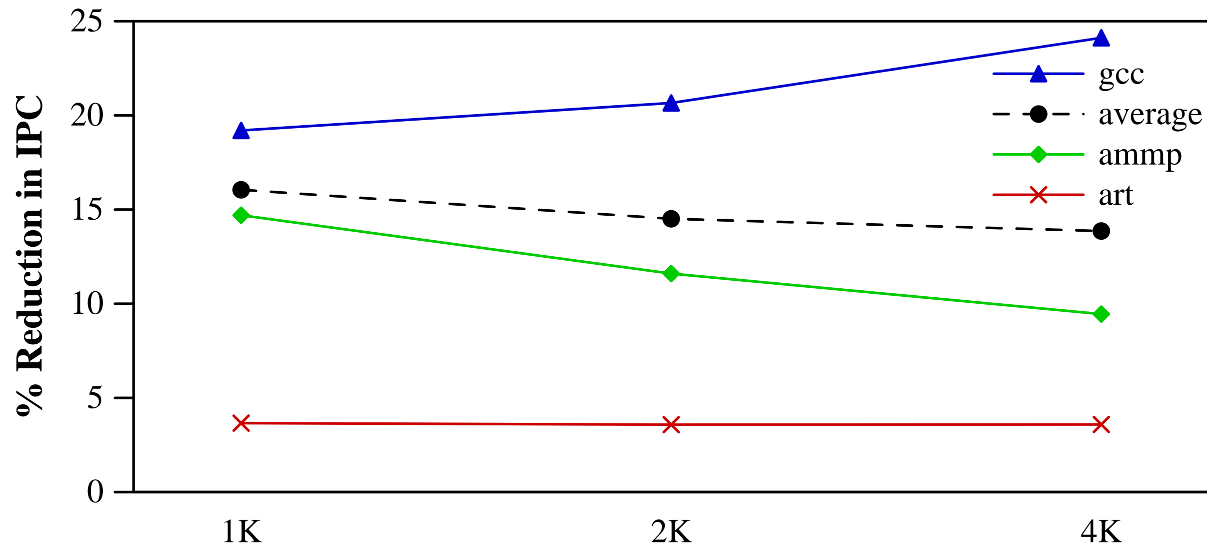
Sensitivity to Entries in the Backlog Buffer



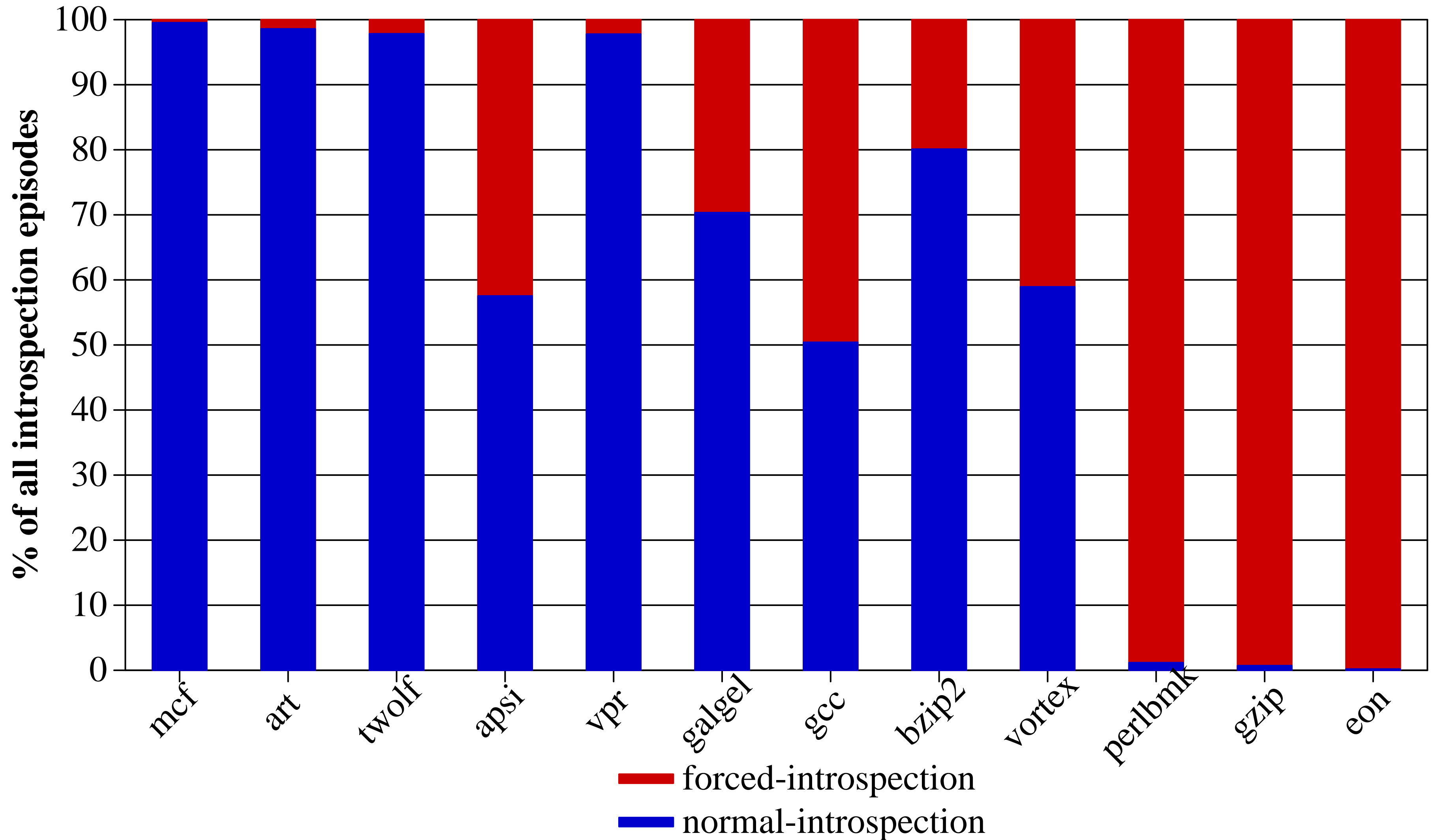
Sensitivity to Entries in the Backlog Buffer



Sensitivity to Entries in the Backlog Buffer



Breakdown of Introspection Episodes



Coverage of Different Types of Introspection