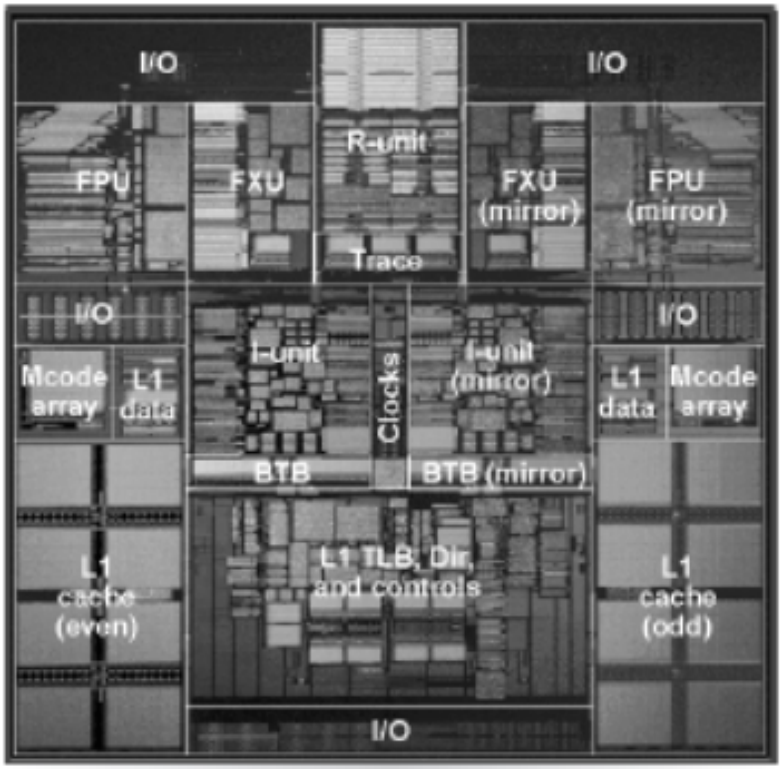
Transient Fault Detection and Reducing Transient Error Rate



Jose Lugo-Martinez

CSE 240C: Advanced Microarchitecture

Prof. Steven Swanson

Outline

- **Motivation**
  - What are transient faults?
- **Hardware Fault Detection**
  - Lockstepping
    - NonStop Himalaya
  - Hardware Transient Fault Detection via SMT
    - SRT
- **Reducing Transient Error Rate**
  - Reducing SDC
  - Reducing False DUE
- **Conclusions**

Transient (Soft-error) Faults Arise

- Alpha and beta particles from packaging material and/or neutrons from cosmic rays that:
  - Invert bit stored in SRAM cell, dynamic latch, or gate
- Probability of transient faults is low—typically less than one fault per year per thousand computers
- Big assumption – transient faults persist for only a short duration

Motivation

- Modern microprocessor are susceptible to hardware transient faults due to:
- Increasing number of transistors
- Decreasing feature sizes
- Reduced chip voltages and noise margins
- Increasing number of processors
- No practical absorbent for cosmic rays

Hardware Fault Detection (HFD)

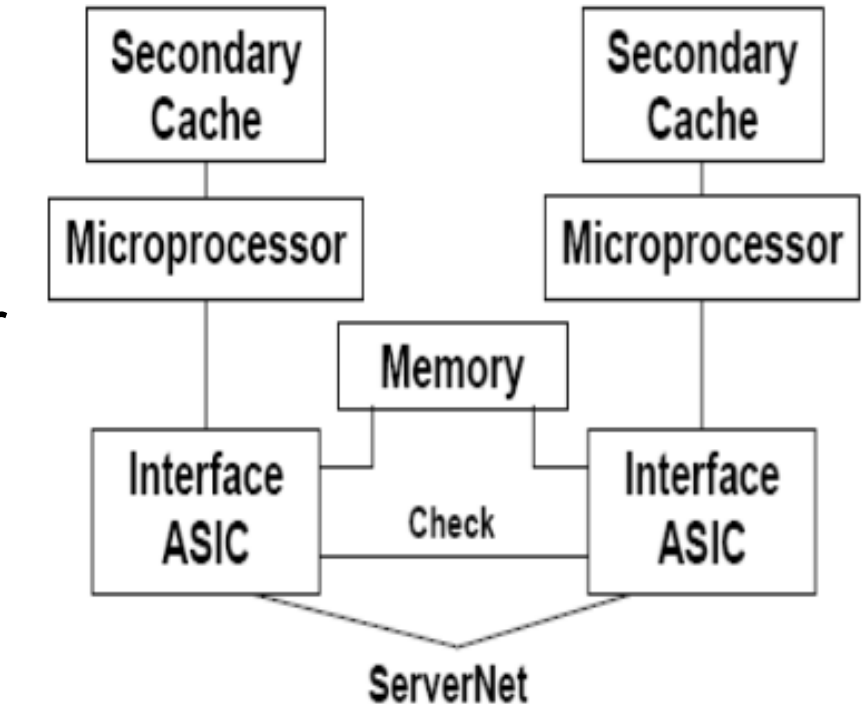
- HFD involves a combination of:
  - Time redundancy
    - (Execute same instruction twice in same hardware)
  - Space redundancy
    - (Execute same instruction on duplicate hardware)
  - Information redundancy
    - (Parity, ECC, etc.)

Previous Approaches for Fault Detection

- Complete hardware replication (lockstepping) … only for mission-critical systems
  - Examples
    - NonStop Himalaya (Next Slide)
    - IBM S/390 G5
- Parity and ECC for large components like caches, memories, etc
- Self checking circuits
- Re-computing with Shifted Operands

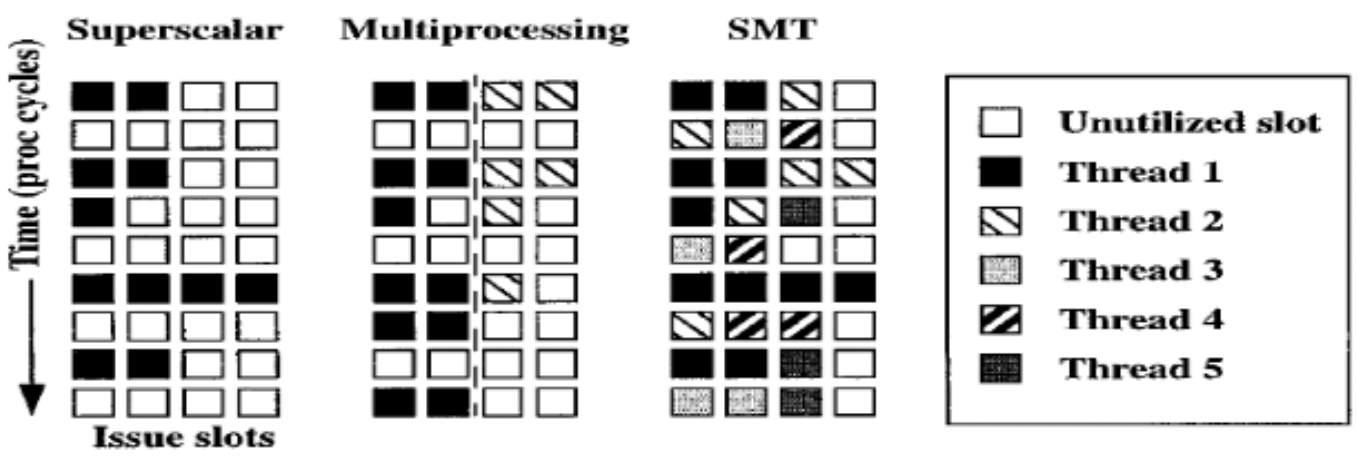
Fault Detection via Lockstepping Microprocessors in the NonStop Himalaya System

- Detect faults by running identical copies of cycle-synchronized microprocessors.
- Each cycle, feed identical inputs to microprocessors, and checker compares outputs.
- If output mismatch, checker flags an error and initiates a software recovery sequence.



Simultaneous Multithreading (SMT) in a Nutshell

- Multiple threads from the same or different processes execute simultaneously through the pipeline
- Dynamic partitioning of resources reduces waste



Fault Detection via SMT

- Complete redundancy without complete replication
- Leverages idle hardware already on chip
- Uses inter-thread “communication” to decrease execution time
- Require less hardware - it can use time and information redundancy in places where space redundancy is not critical.

Previous Work (AR-SMT)

- First paper to use SMT for HFD
- Fault detection through time/space redundancy
- Two copies of the program run as separate threads sharing hardware resources
- Dynamic instruction scheduling enables efficient resource utilization

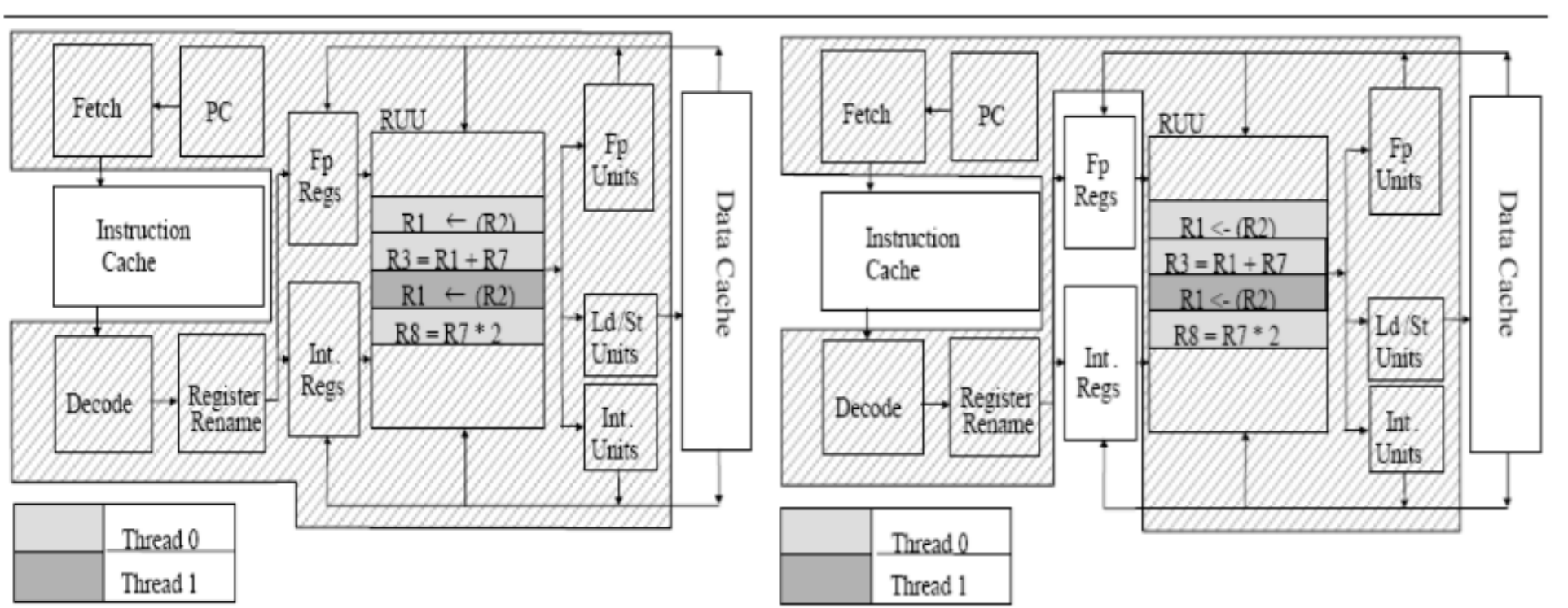
Transient Fault Detection via SMT Paper

- Analyzes performance impact of fault tolerance of Simultaneous and Redundant Threading (SRT)
- Introduces *Sphere of Replication* concept
- Input replication mechanism
- Architecture for output comparison
- Slack fetch and branch outcome queue mechanism

SRT Overview

- SRT = SMT + Fault Detection
- Advantages
  - Piggyback on an SMT processor with “little” extra hardware
  - Better performance than complete replication
  - Lower cost
- Challenges
  - Lockstepping very difficult with SRT
  - Must carefully fetch/schedule instructions from redundant threads

Sphere of Replication

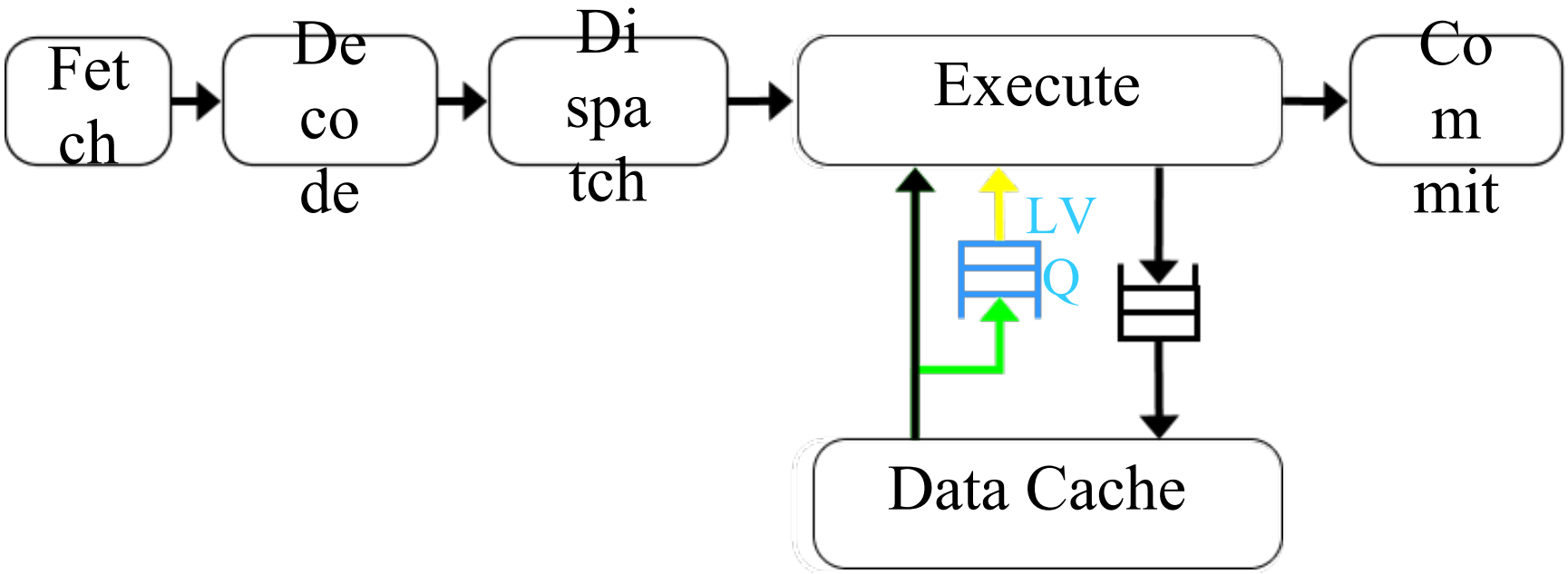


- Size of sphere of replication
  - Two alternatives – with and without register file
  - Instruction and data caches kept outside

Input Replication

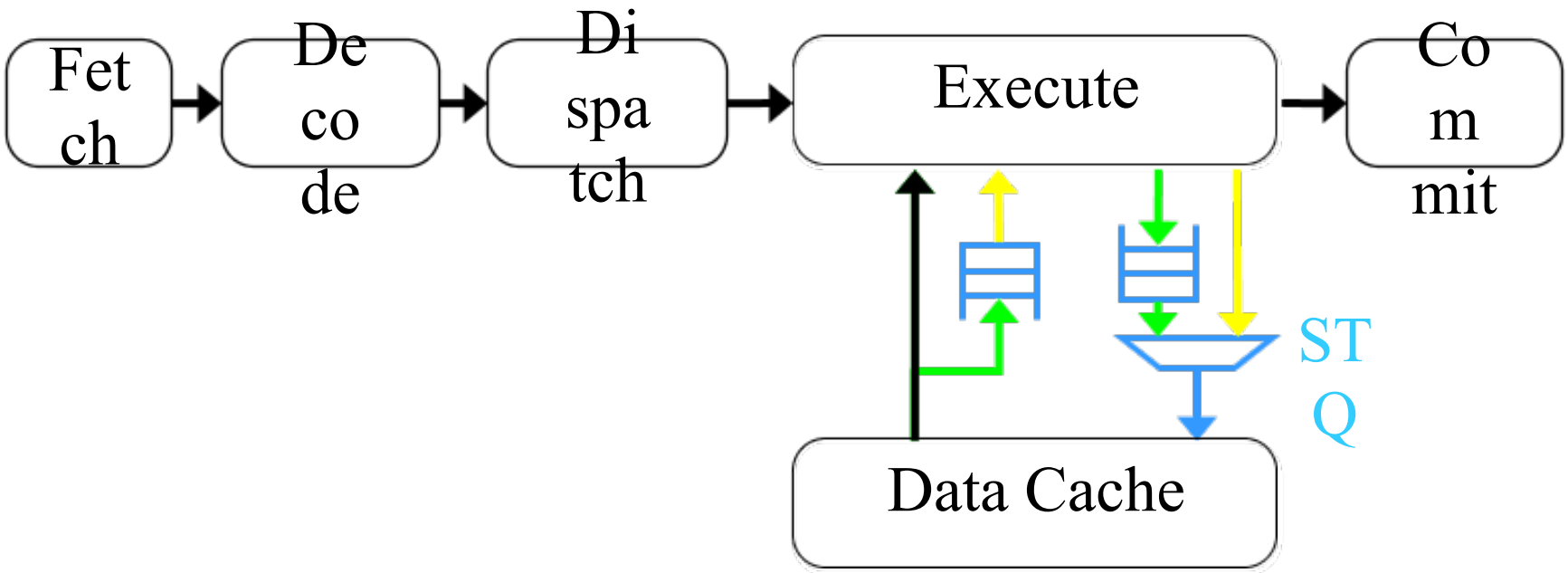
- Guarantee that both threads received same inputs and follow same path
- Instructions
  - Can't be self-modified
- Cached load data
  - Out-of-order execution issue, multiprocessor cache coherence issues
- Uncached load data
  - Must synchronize
- External interrupts
  - Stall lead thread and deliver interrupt synchronously
  - Record interrupt delivery point and deliver later

Load Value Queue (LVQ)



- Load Value Queue (LVQ)
  - Keep threads on same path despite I/O or MP writes
  - Out-of-order load issue possible

Output Comparison

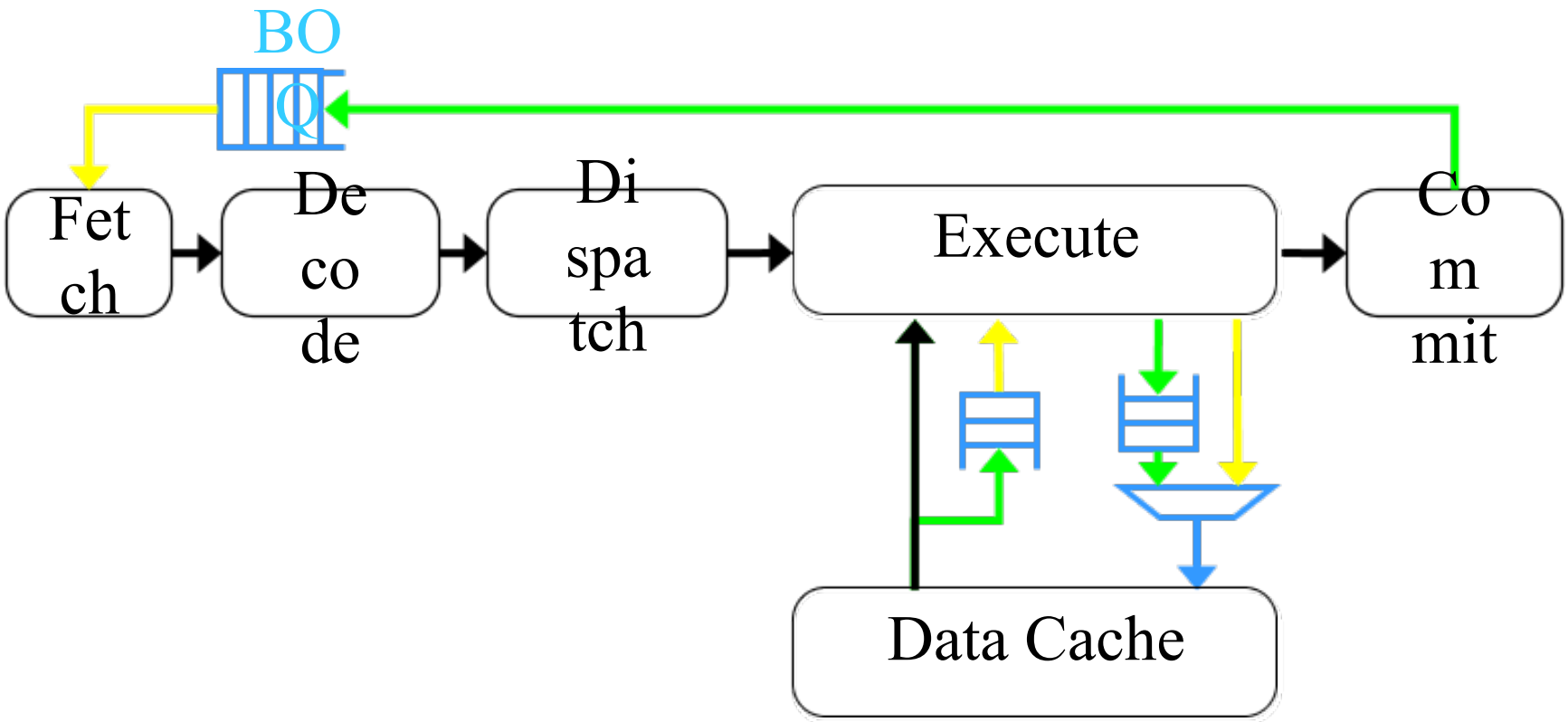


- Store Queue Comparator
  - Compares outputs to data cache
  - Catch faults before propagating to rest of system

Slack Fetch

- Maintain constant lag between thread's execution
- Lead thread updates branch and data predictors
- Lead thread prefetches loads
- Traditional SMT ICount fetch policy is modified to maintain slack

Branch Outcome Queue (BOQ)



- Branch Outcome Queue
  - Forward leading-thread branch targets to trailing fetch
  - 100% prediction accuracy in absence of faults

Results

- Baseline Characterization
  - ORH-Dual two pipelines, each with half the resources
  - SMT-Dual replicated threads with no detection hardware
- ORH and SMT-Dual 32% slower than SMT-Single

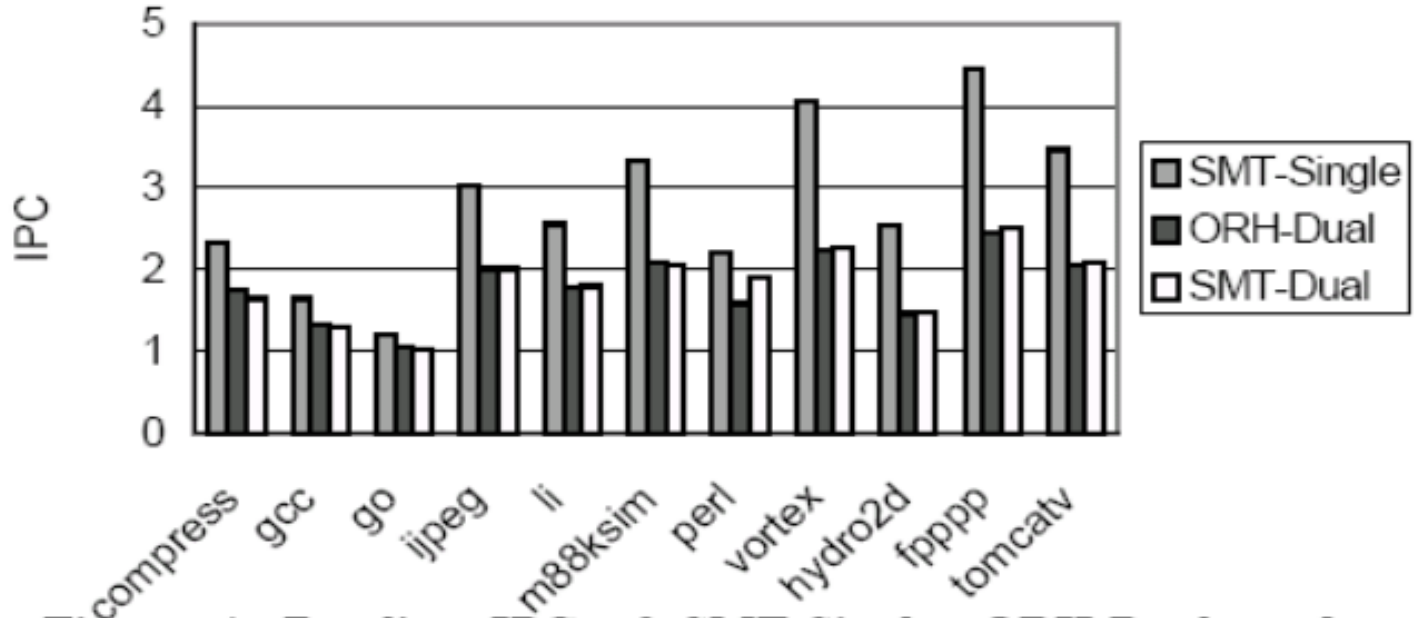


Figure 4. Baseline IPC of SMT-Single, ORH-Dual, and SMT-Dual.

Overall Results

- Speedup of SRT processor with 256 slack fetch, branch outcome queue with 128 entries, 64-entry store buffer, and 64-entry load value queue.
- SRT demonstrates a 16% speedup on average (up to 29%) over a lockstepping processor with the “same” hardware

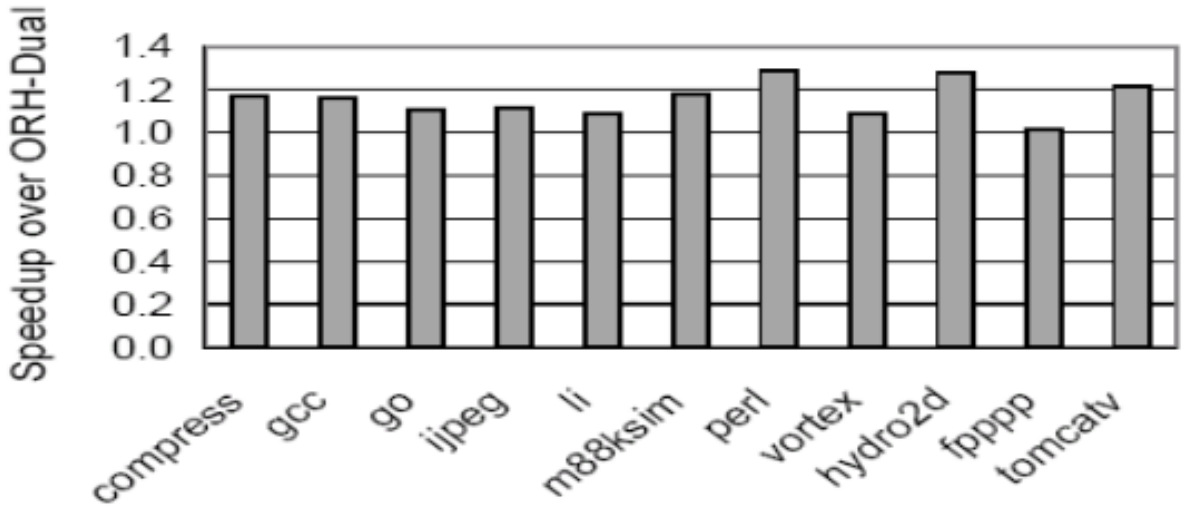


Figure 9. Speedup of sf256+boq128+sb64+lvq64 over ORH-Dual.

Conclusions

- SRT processor can provide similar transient fault coverage than cycle-by-cycle lockstepping, but with "superior" performance.

- However, in a later publication "*Detailed Design and Evaluation of Redundant Multithreading Alternatives* " the benefits of SRT are not as great as those reported in this paper when using a detailed model
  - 30% and 32% degradation seen on single thread and multithread workloads, respectively

Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor

Classification of Possible Faults Outcomes Bits in Microprocessor

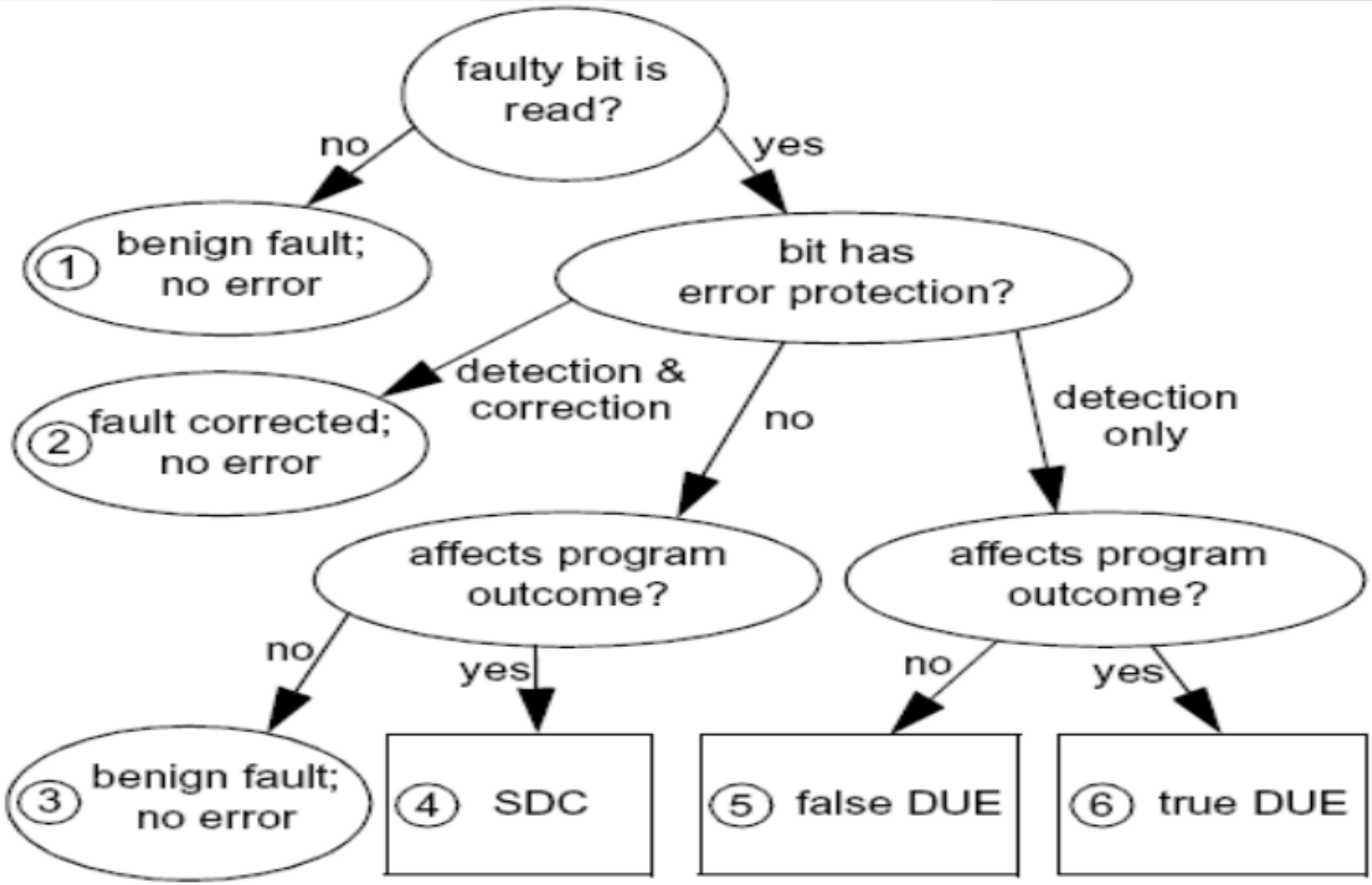


Figure 1. Classification of the possible outcomes of a faulty bit in a microprocessor. SDC = silent data corruption. DUE = detected unrecoverable error.

Instruction Queue's SDC AVF

- The SDC architectural vulnerability factor (AVF) of a structure is the average of the SDC AVFs of all cells in that structure.
- Mukherjee, et al. computed an SDC AVF of 28% for an unprotected instruction queue in an Itanium 2-like microprocessor.
- SDC AVF => probability that a strike affecting the device propagates to program output

Reducing Silent Data Corruption (SDC)

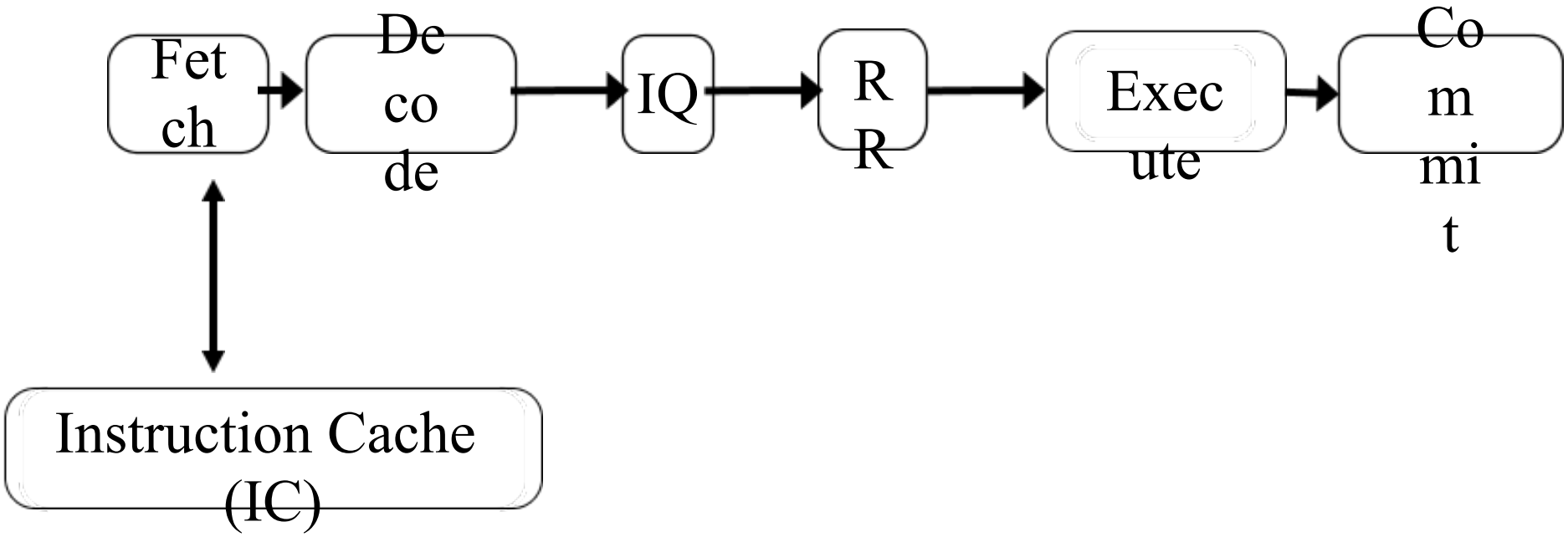
- Previous approaches
  - Change process technology (fully depleted *Silicon on Insulator*)
  - Circuit technology (radiation hardened shells)
  - Error detection

- Proposed approach: Reduce architecturally correct execution, ACE, (*i.e.* any execution that generates results consistent with the correct operation of the system as observed by a user) object exposure to radiation
  - Squash instruction queue on stalls
  - MITF – measures trade off between performance and error rate
    - MITF tells us how many instructions a processor will commit, on average, between two errors.
    - A higher MITF implies a greater amount of work done between errors.

Overview of Reducing Exposure to Radiation

- Main idea: Reduce the time instructions sit in the queue
- How?
  - Trigger on cache miss
- Action?
  - Squash all instructions in the queue on load miss,
    - Because they examine an in-order machine, squashing should have minimal impact on performance.
    - At the same time, it should lower the AVF by reducing the exposure of instructions to neutron and alpha strikes

Reducing Exposure to Radiation in IQ



- Increase IPC: fetch aggressively from IC to IQ
- Reduce SDC AVF: prevent instructions from sitting needlessly in IQ
- Net benefit if we improve MITF (proportional to IPC / AVF)

Results for Reducing Exposure to Radiation

$$\text{MITF} \approx \frac{\text{IPC}}{\text{SDC AVF}}$$

Design Point	**IPC**	**SDC AVF**	**IPC/SDC AVF**	**MITF Improvement**
No Squashing	1.21	29%	4.1	0%
Squash on L1 Miss	1.19	22%	5.6	37%
Squash on L0 Miss	1.09	19%	5.7	39%

Classification of Possible Faults Outcomes Bits in Microprocessor

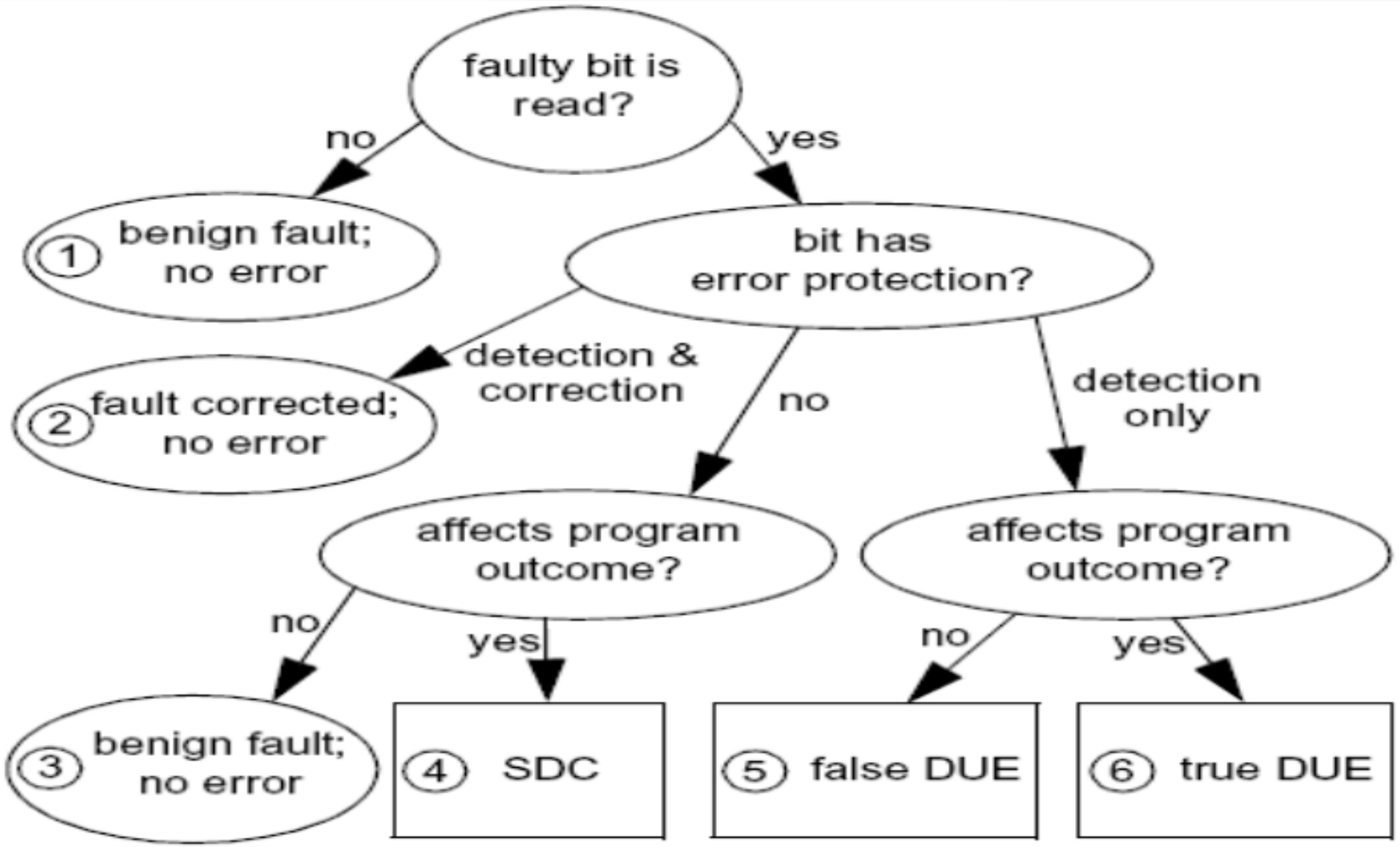


Figure 1. Classification of the possible outcomes of a faulty bit in a microprocessor. SDC = silent data corruption. DUE = detected unrecoverable error.

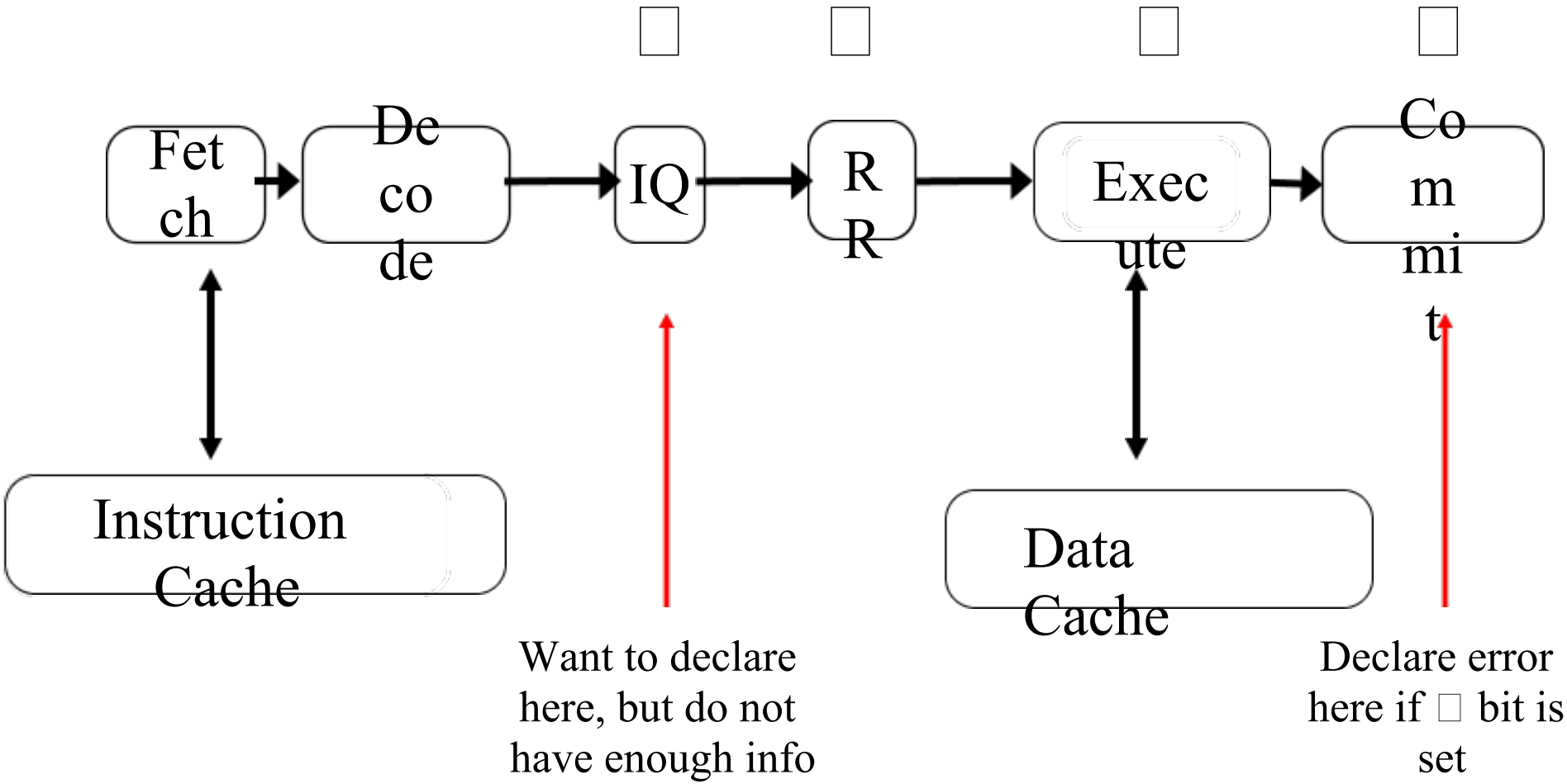
Reducing False Detected Unrecoverable Errors (DUE)

- The false DUE AVF is of 33%.
- Idea
  - Modify pipeline's error detection logic to mark affected instructions and data as possibly incorrect rather than immediately signaling an error. If determine later that the possibly incorrect value could have affected the program's output then signal an error.
- Techniques?
  - π bit (Possibly Incorrect bit)
  - anti-π bit

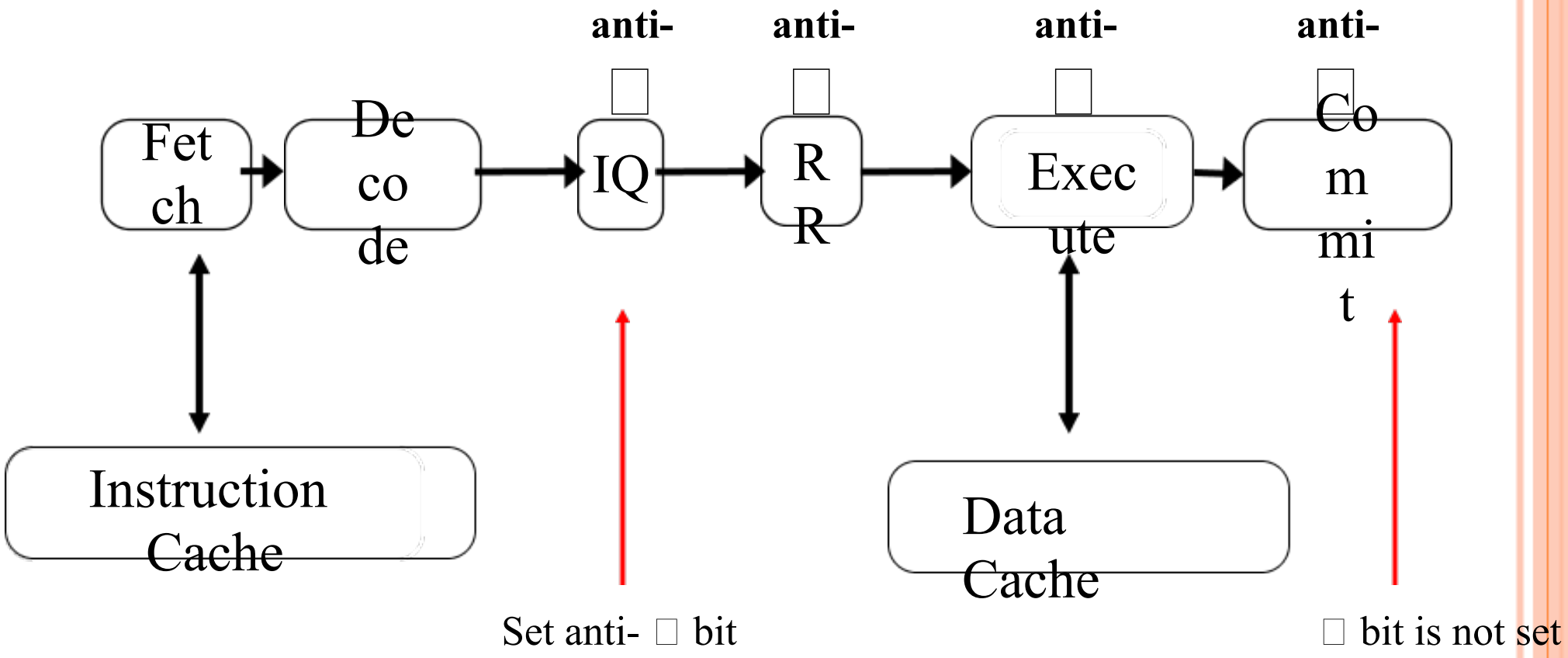
Sources of False DUE Events

- Instructions with uncommitted results
  - wrong-path, predicated-false
  - Solution: bit until commit
- Instruction types neutral to errors
  - no-ops, pre-fetches, branch predict hints
  - Solution: anti- bit
- Dynamically dead instructions
  - instructions whose results will not be used in future
  - Solution: bit after they commit

Wrong Path Instructions: □ Bit Solution



No-Ops: Anti-□ Bit Solution

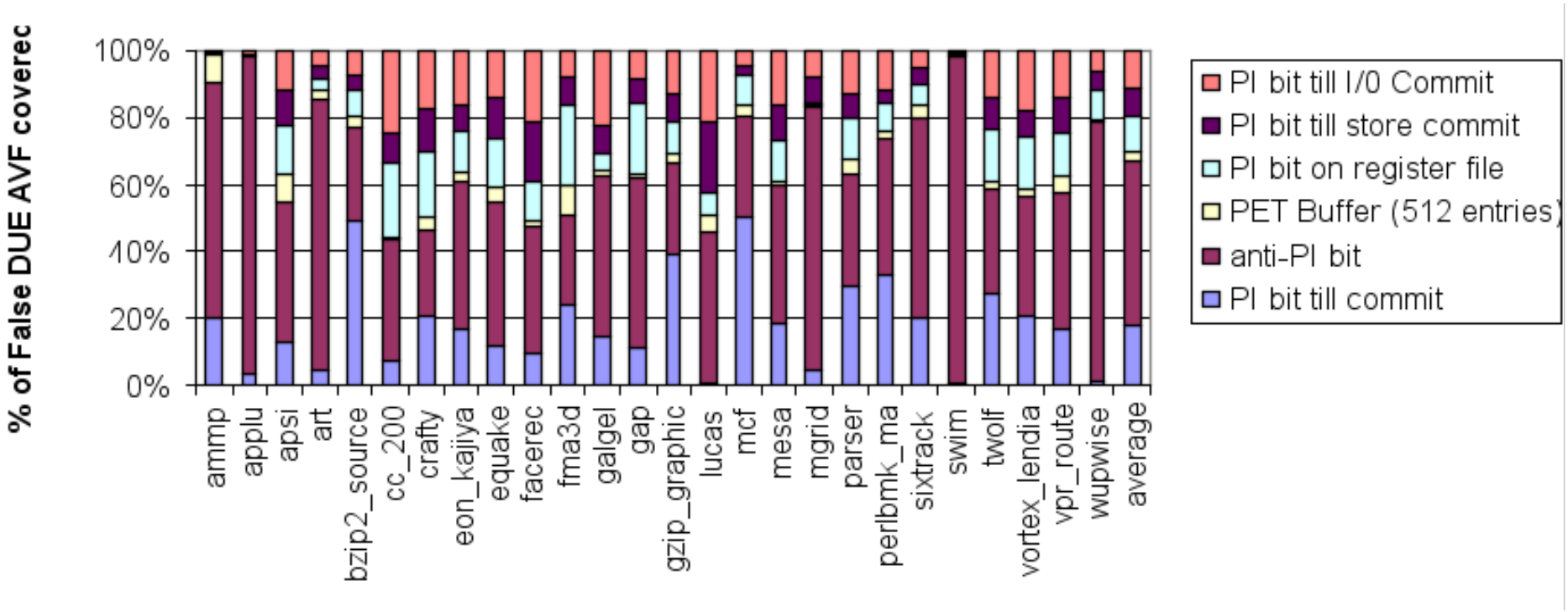


- If anti-□ bit is set, do not flag □ bit

Dynamically Dead Instructions

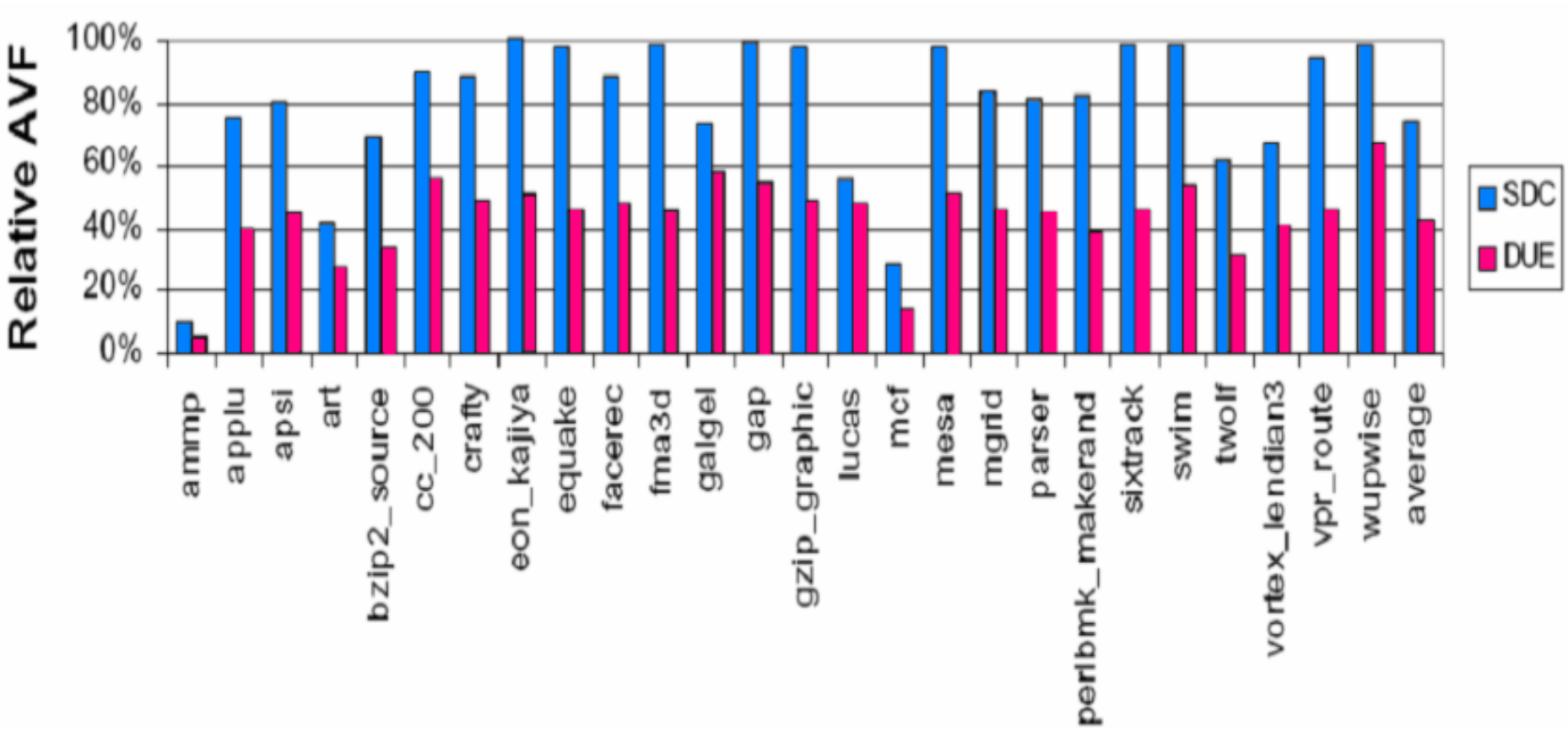
- Carry □ bit through to register
- Declare the error on load, if □ bit is set
- If register is not read (dynamically dead), then no false DUE

Results for Reducing False DUE



- Pi-bit till commit -> reduces by 18%
- Anti-pi bit -> reduces by 60% for FP and 35% for integer
- Pi-bit on register file -> reduces by 11%
- Pi-bit till store commit -> reduces by 8%
- Pi-bit till I/O commit -> reduces by 12%

Result for Combining Both Techniques



- Average 26% reduction in SDC AVF
  - ammp – 90% reduction with 7% decrease in IPC (because instructions queued behind few critical cache misses)
- Average 57% reduction in DUE AVF with 2% decrease in IPC
- DUE MITF increase by 15%

Conclusions

- Reducing SDC
  - Keep instructions in protected memory for as long as possible
- Reducing False DUE
  - Reduce false errors (π bit, anti-π bit)