Ventures into Power-Aware computer architecture design

R. Iris Bahar

School of Engineering
Brown University

BROWN
School of Engineering

THE PRE-DAWN OF POWER-AWARE PROCESSOR DESIGN

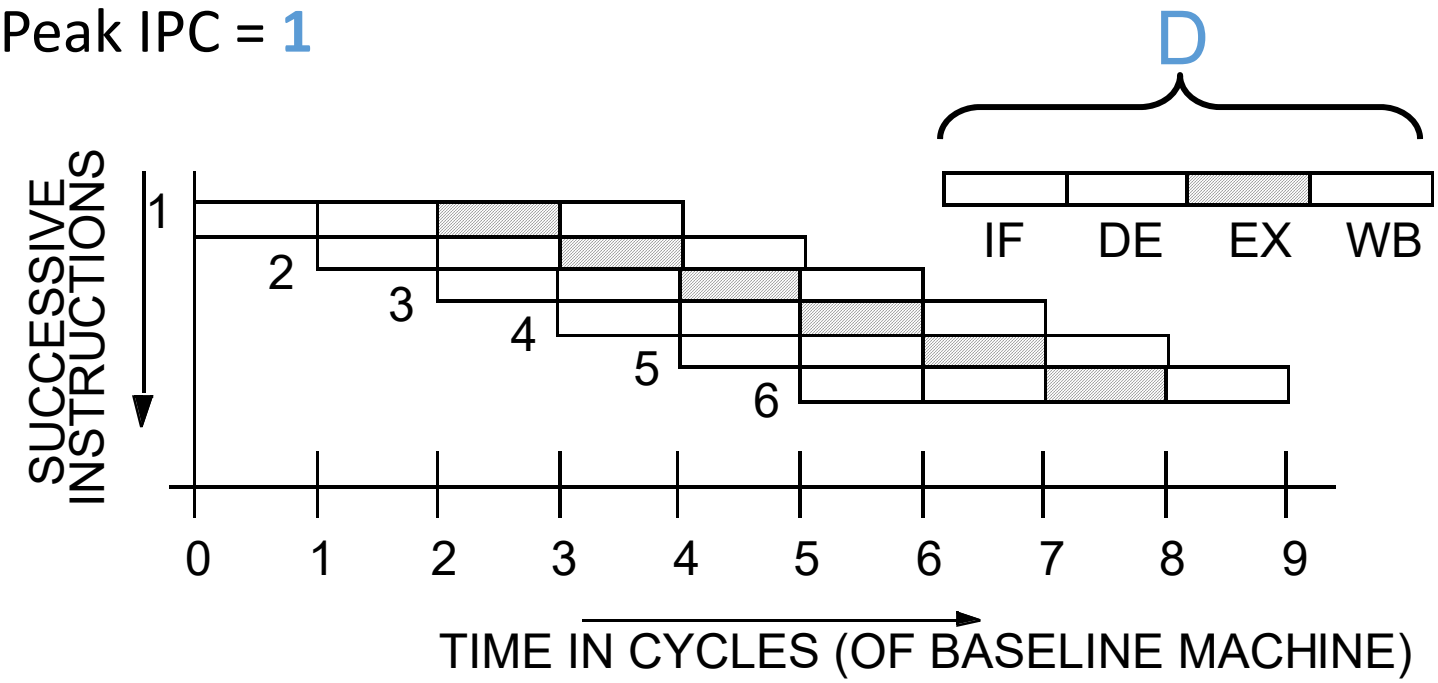
Architectures for Instruction-Level Parallelism

Scalar Pipeline (baseline)

Pipeline Depth = **D**

Operation Latency = **1**

Peak IPC = **1**



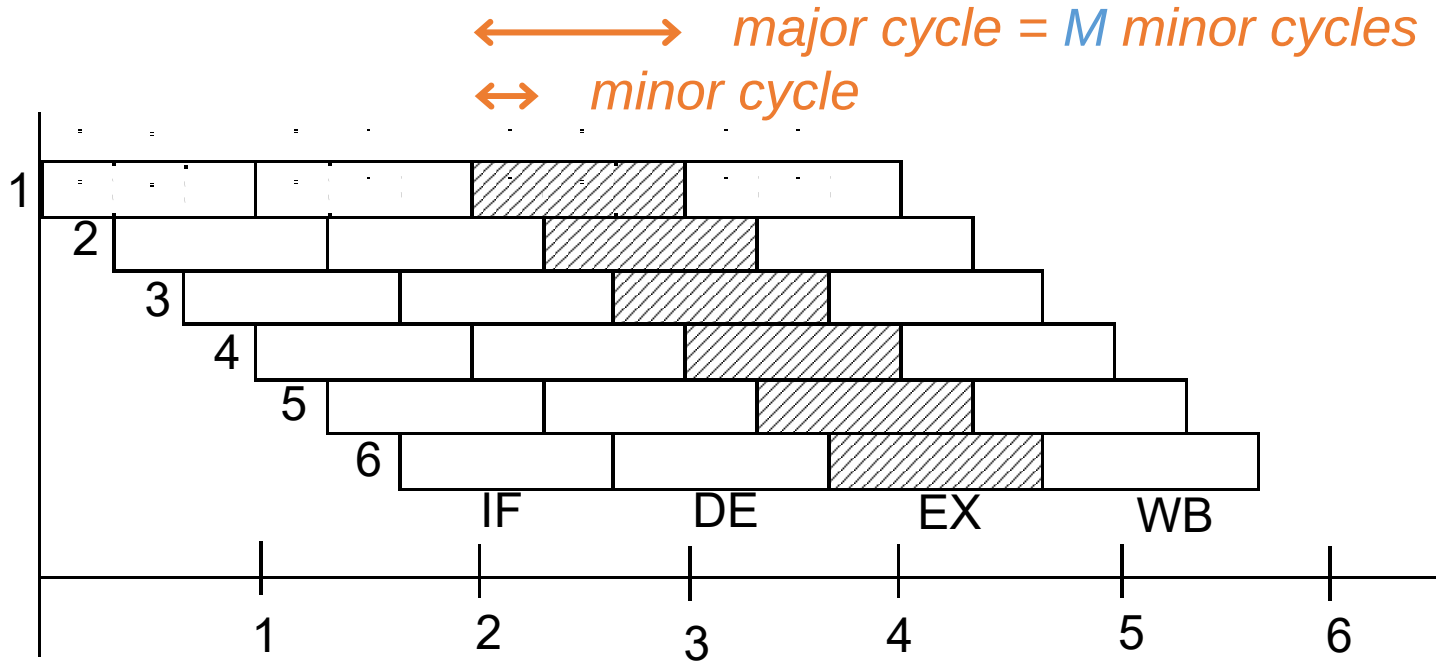
SUPERPIPELINED MACHINE

Superpipelined Execution

Instruction Parallelism = *DxM*

Operation Latency = *M minor cycles*

Peak IPC = *1 per minor cycle* *(M per baseline cycle)*



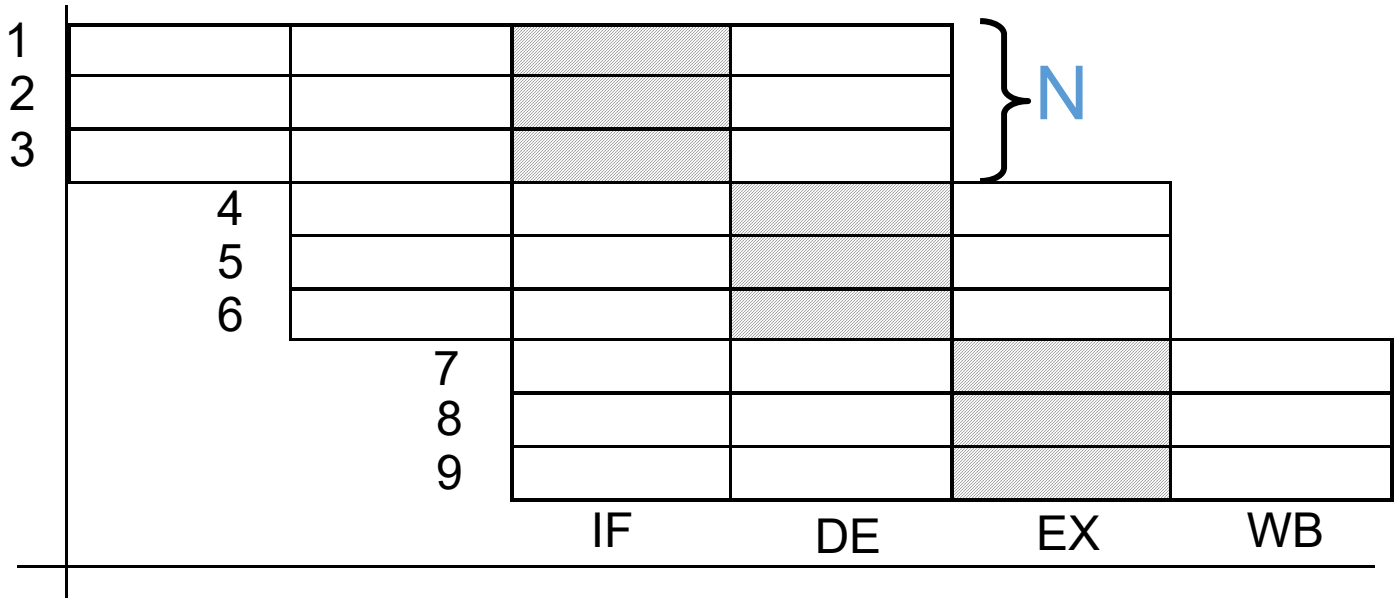
SUPERSCALAR MACHINES

Superscalar (Pipelined) Execution

Instruction Parallelism = *DxN*

Operation Latency = *1 baseline cycles*

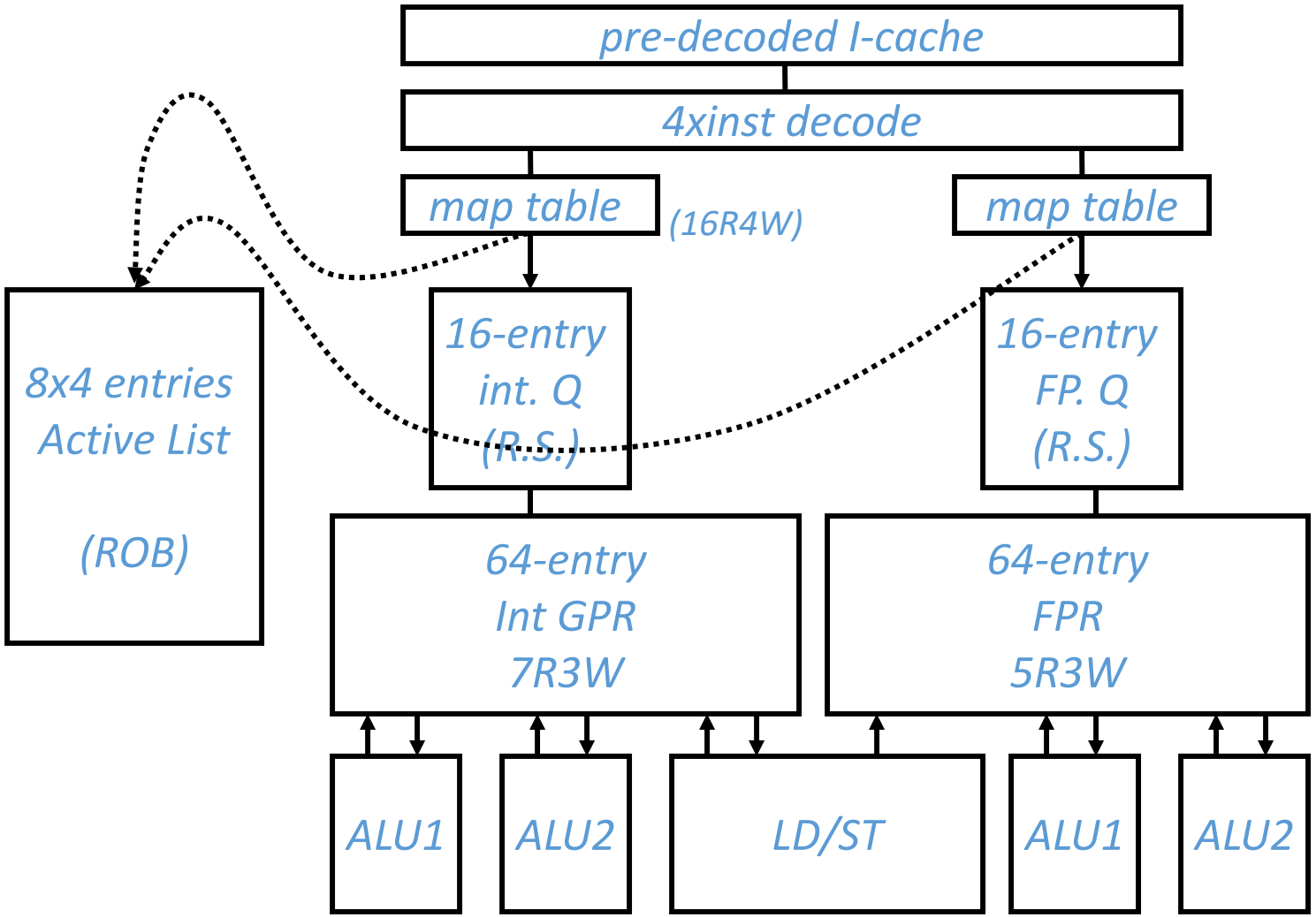
Peak IPC = *N per baseline cycle*



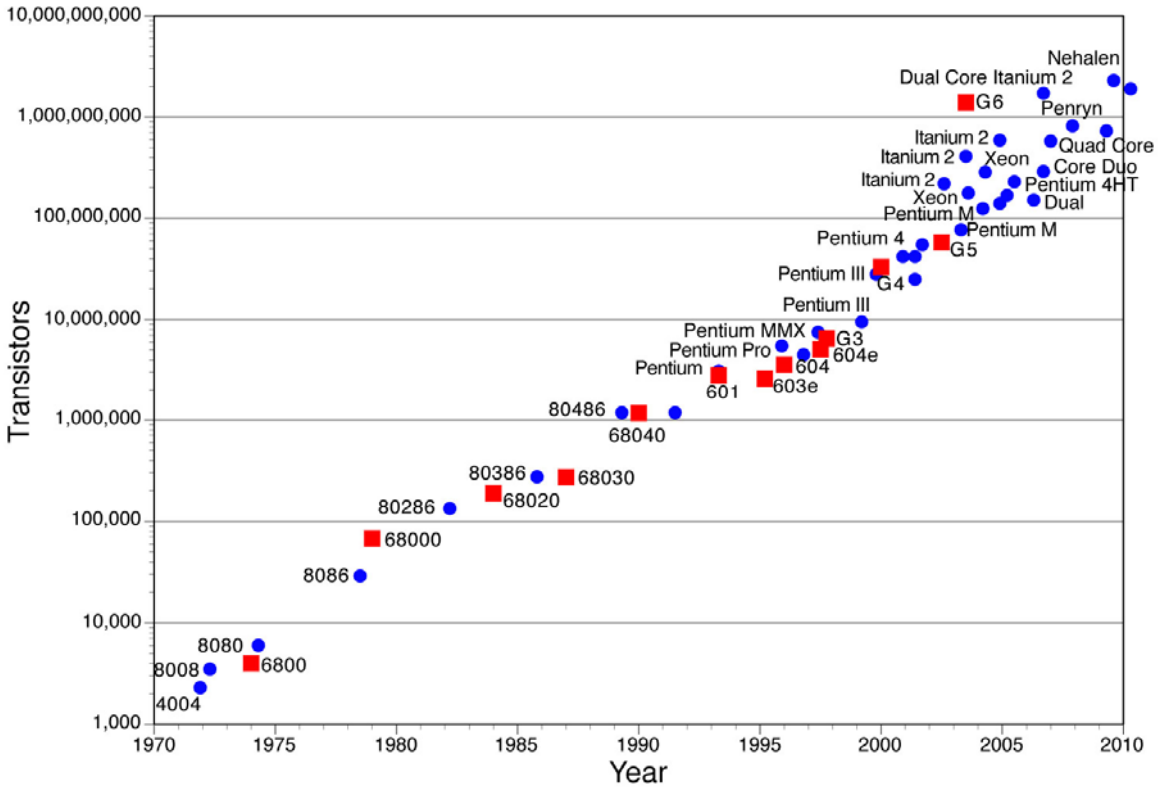
Out of Order Execution

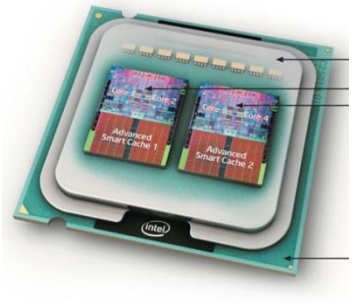
- Execute instr. based on “data flow” rather than program order.
- **Basic idea**:
  - Fetch fills up a window of instructions.
  - Of all instructions in the window, look for ones ready to execute:
    - All the data on which the instructions are dependent are ready
    - Resources are available.
- As soon as instruction is executed, it needs to signal to its dependent instructions that the input is ready.
  - Triggers “wake-up” and “instruction select” for next cycle
- **Advantages:**
  - Help exploit Instruction Level Parallelism (ILP)
  - Help cover latencies (e.g., cache miss, divide)

MIPS R10000 (1996)



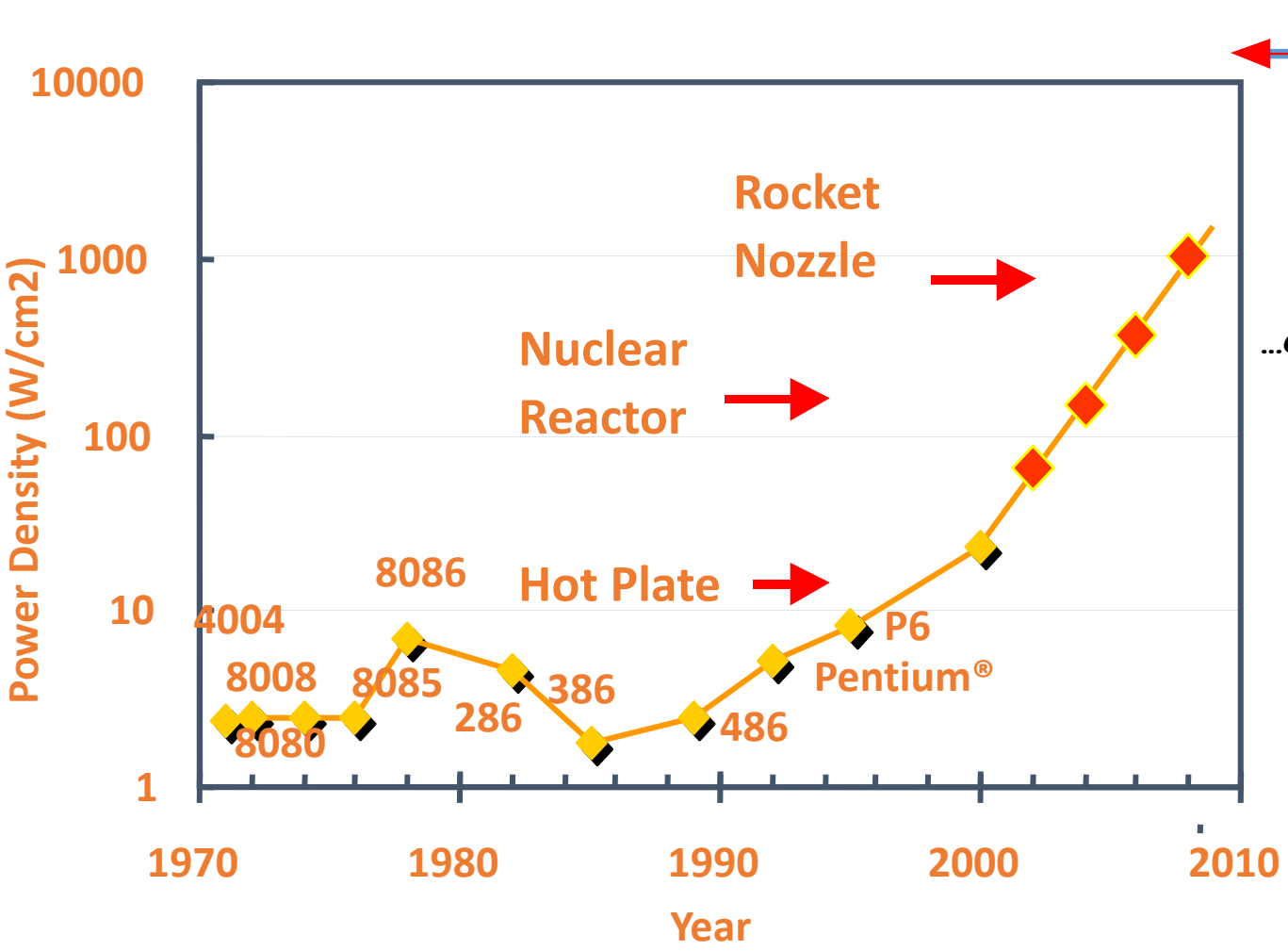
MOORE'S LAW IN ACTION



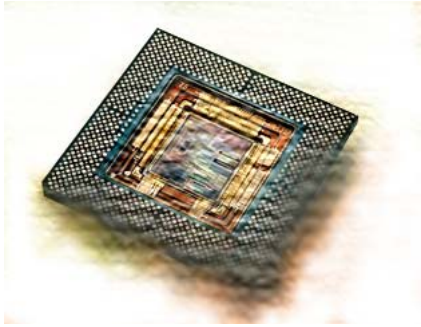


- 2300 transistors, 1 MHz clock (Intel 4004) - 1971
- 16 Million transistors, 900MHz (Ultra Sparc III) - 2001
- 42 Million, 1.5 GHz clock (Intel P4) - 2001
- 140 Million transistors, 500MHz (HP PA-8500) - 1998

CHIP POWER DENSITY



…chips might become hot…



Source: Borkar, De Intel®

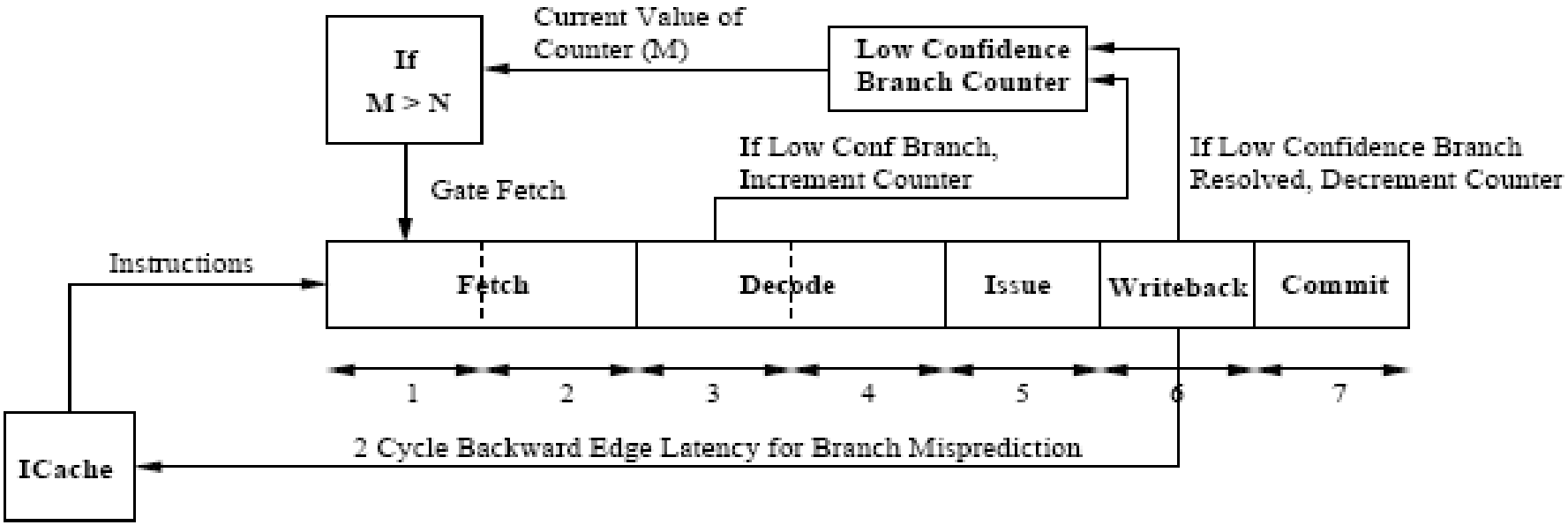
The Cost of Speculation

- How does the processor find enough "ready" instructions?
  - Look beyond branch boundaries
  - Use fairly sophisticated branch prediction mechanisms to guess target and direction.
  - Front end fills instruction window with instructions down path of predicted branch
- What happens if the branch prediction is wrong?
- What is the cost in terms of performance and power of keeping these "wrong path" instructions in the instruction window?

Reducing Mis-speculated Instructions to Save Power

- Executing wrong path instruction is a waste of power
  - Does nothing to improve effective IPC either
- ***IDEA:*** if branch prediction becomes too speculative, don't bother continuing to fetch instructions past branches
  - Fetch unit stops reading new instructions from the cache
  - Instruction window does not take new instructions
  - Instruction execution rate may slow, but only until predicted branches have been resolved
- How do we know if an instruction flow has become "too speculative"?

PIPELINE GATING (ISCA 1998 MANNE ET AL.)



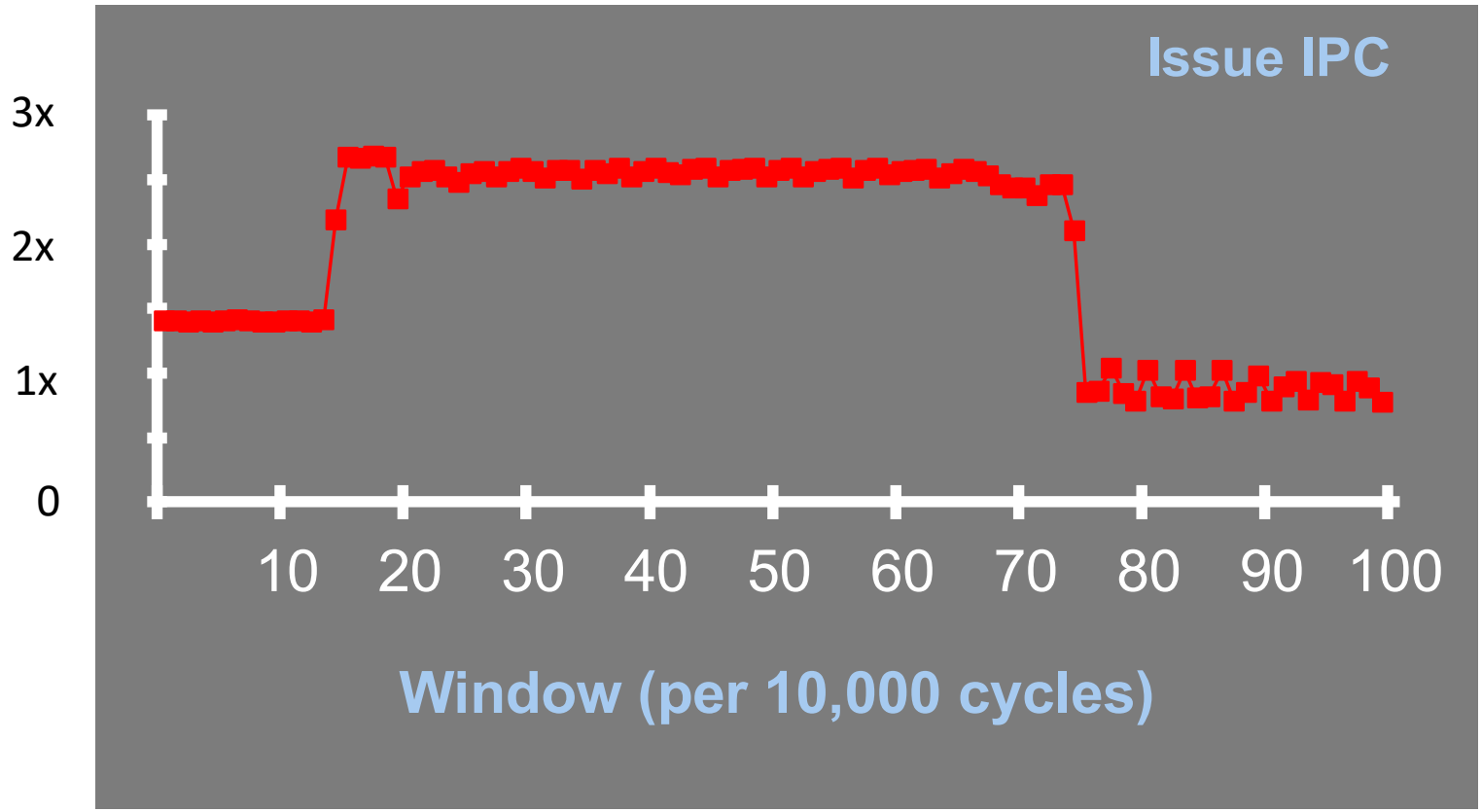
- Low-confidence branch counter records # of unresolved branches that reported as low-confidence.
- The processor gates off instruction fetch if there are more than *N* unresolved low-confident branches in the pipeline.

ISCA 2013 Influential Paper Award !

Do We Need Wide Issue Machines?

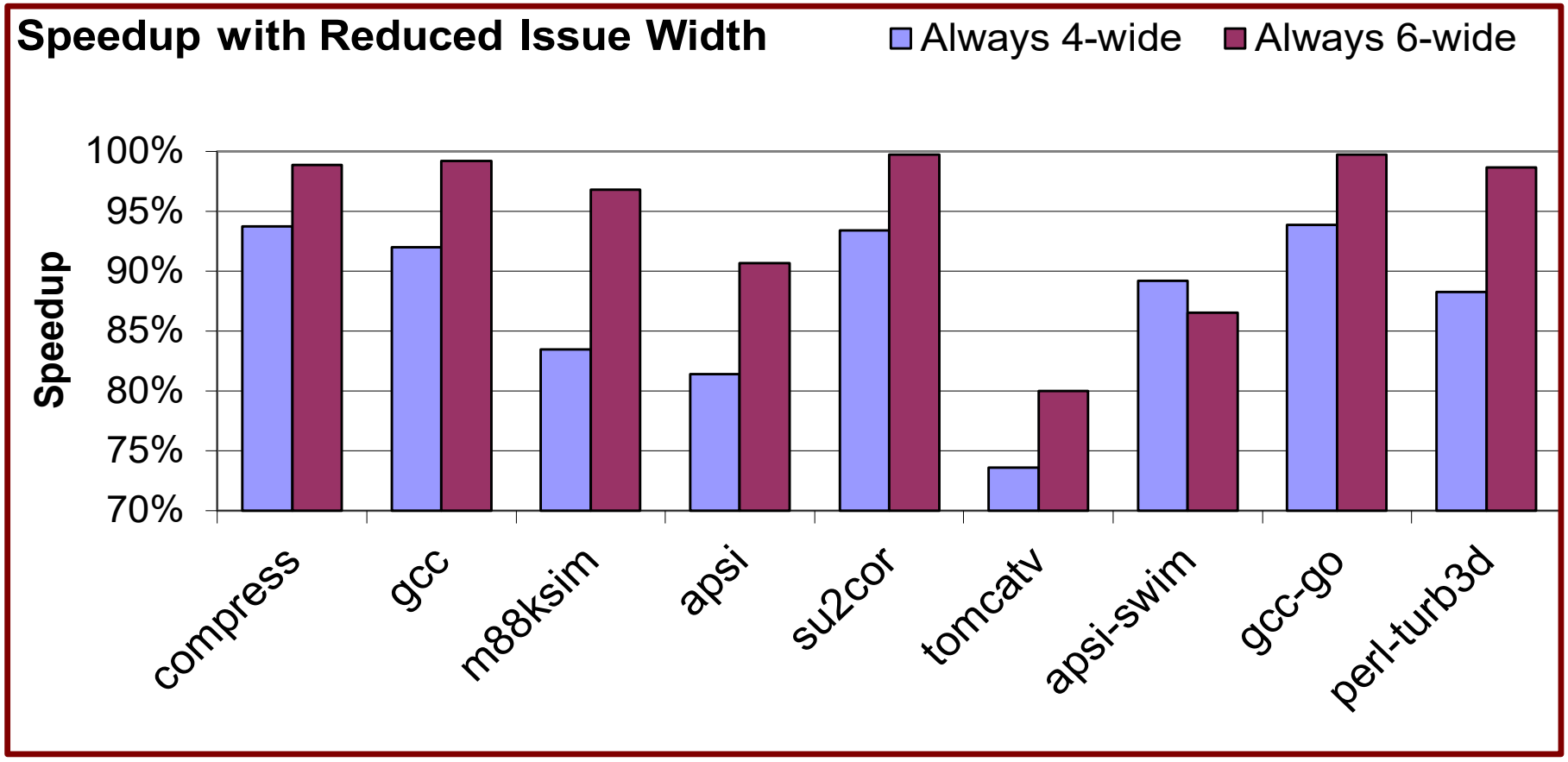
- Many programs never achieve IPC values close to the maximum issue of the machine
  - Branch Misprediction
  - Dependency Chains
  - Cache Misses
- Overall IPC is not indicative of superscalar needs of a program

VARYING PROGRAM NEEDS



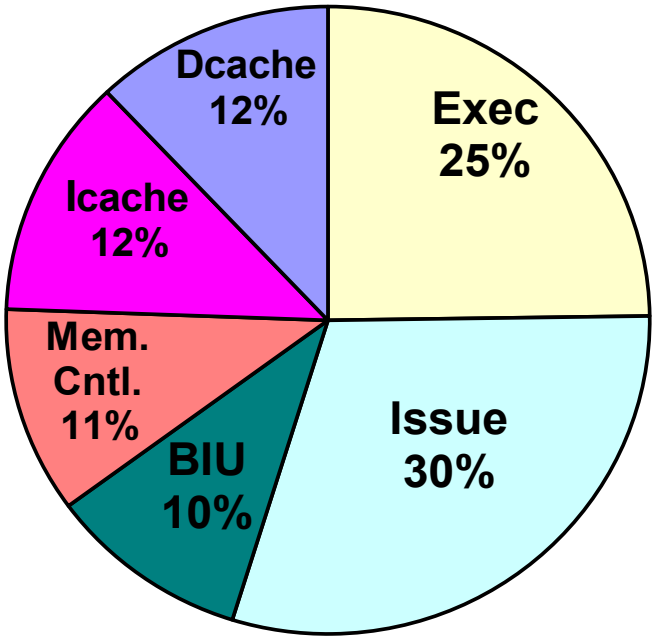
- Nearly 3X difference in IPC across successive snapshots of the program execution

IMPACT OF REDUCED ISSUE WIDTH

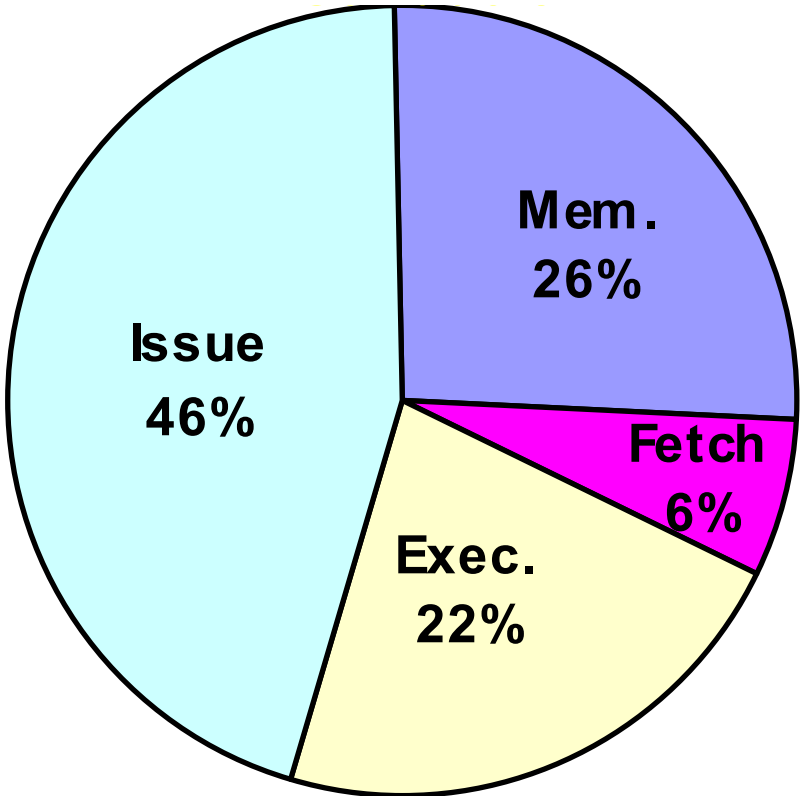


MICROARCHITECTURAL DECISIONS IMPACT POWER

21264 Power Distribution



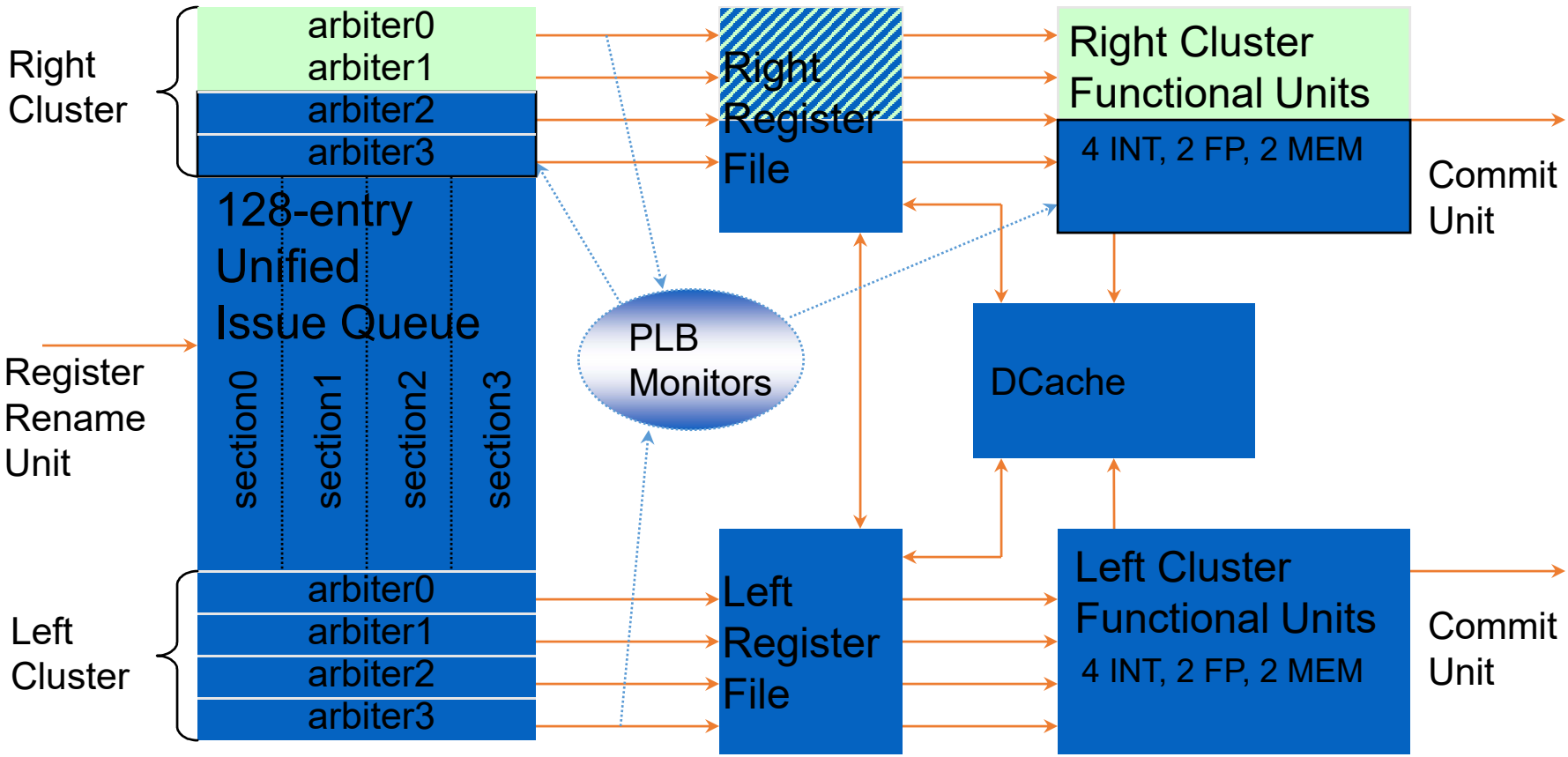
21464 Power Distribution



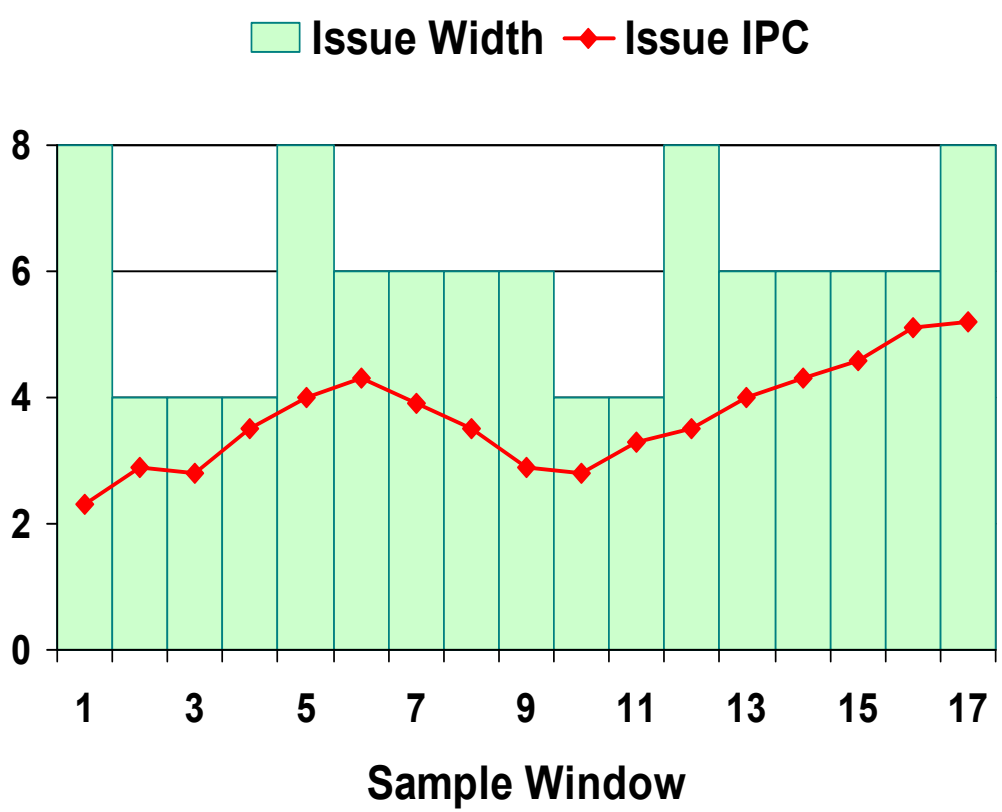
Pipeline Balancing (ISCA 2001, Bahar, Manne)

- Monitor the varying issue requirements of each program
  - Overall issue rate
  - Floating point issue rate
  - History of past behavior
- **IDEA:** Tune processor issue and execution resources according to the needs of the program
- **Goal:** Reduce power, retain performance

Our 8-Wide Issue Processor Model

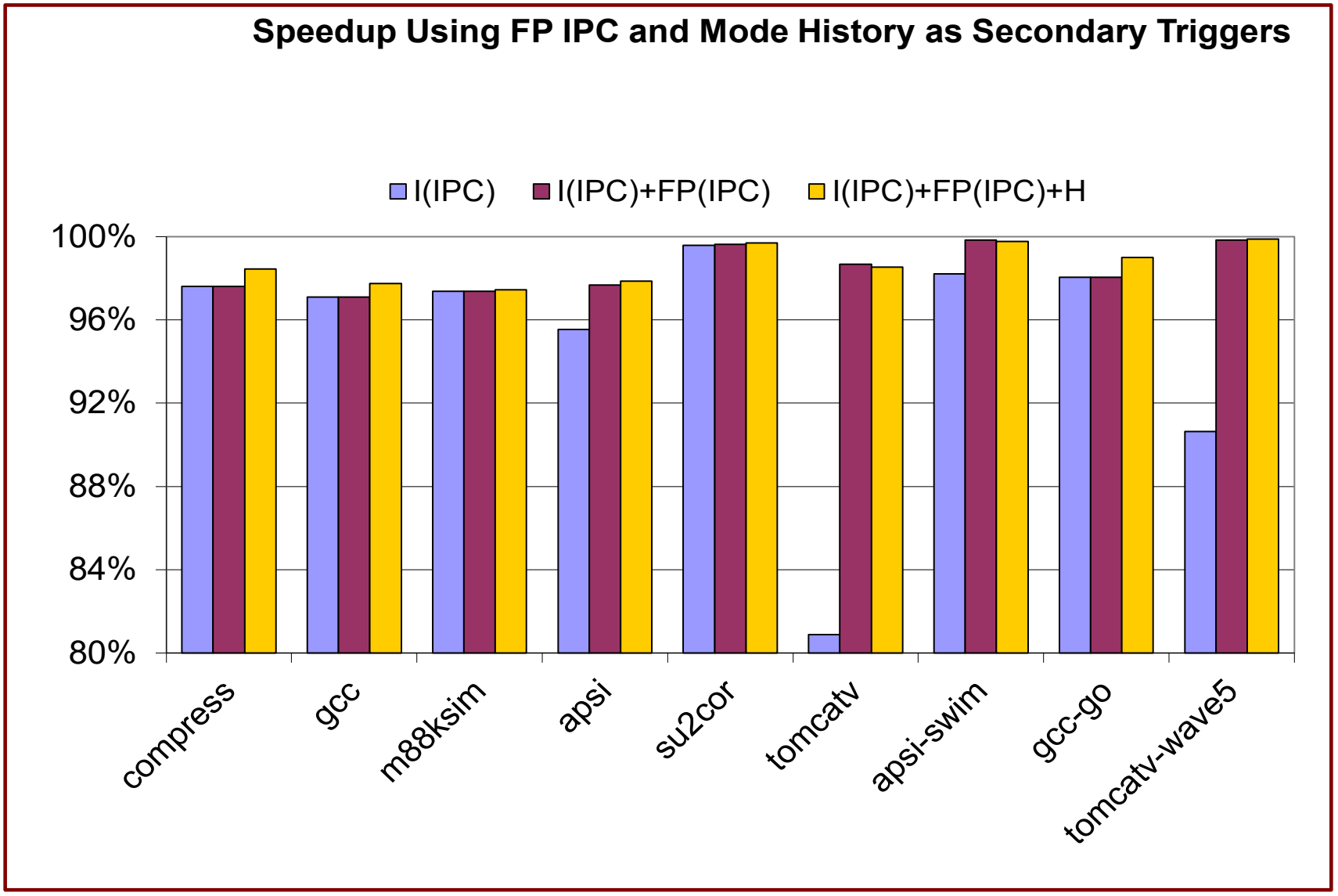


TRIGGERING PLB BY TRACKING ISSUE IPC



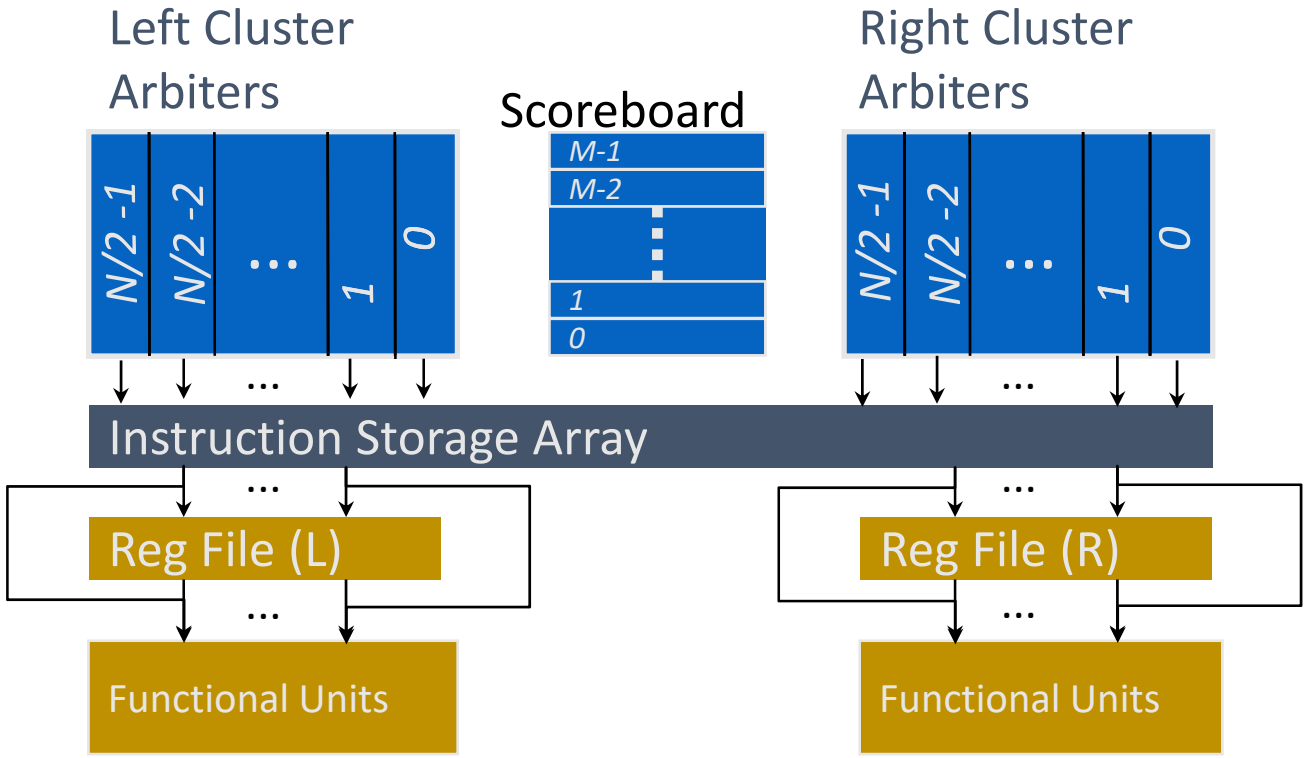
- Switch to 4-, 6-, or 8-wide issue depending on issue IPC of previous sampling window

PRIMARY AND SECONDARY TRIGGERS

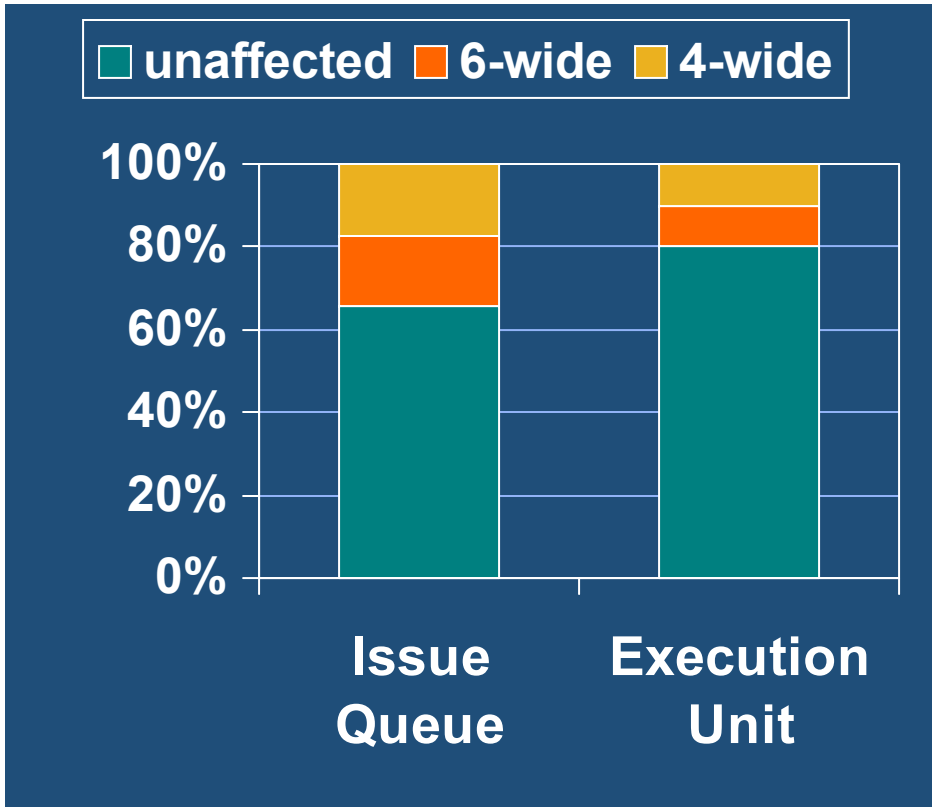


IQ Design and Power Dissipation

- Goal: Issue **N** instructions out of **M** entries in IQ
- IQ power is relative to the size of **N** and **M**

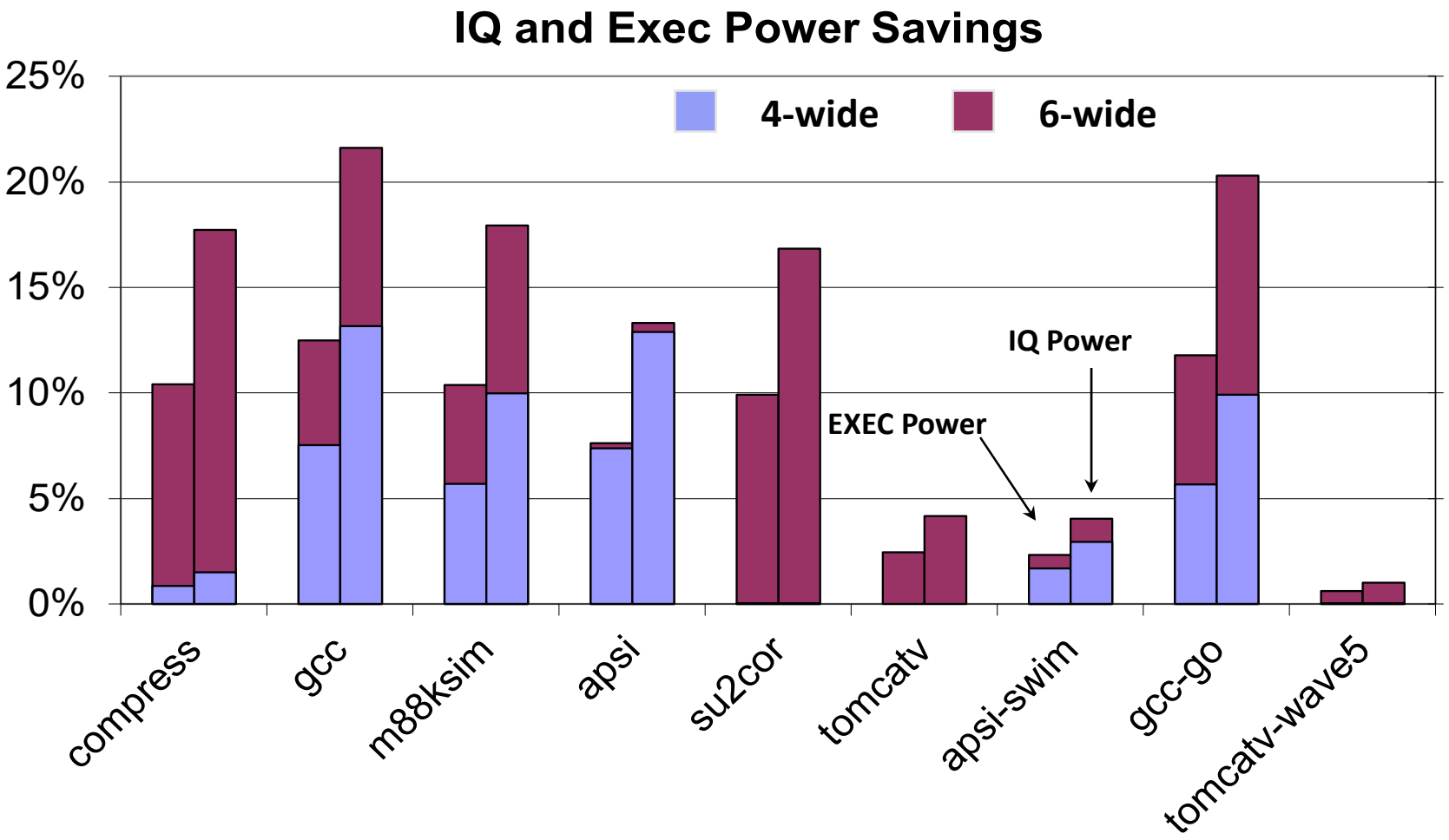


PER COMPONENT POWER ESTIMATES

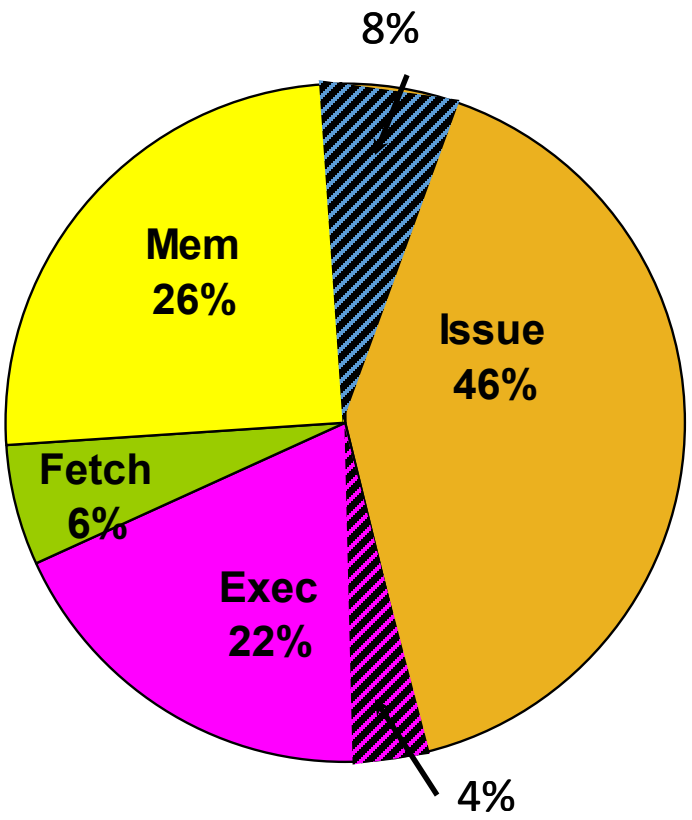


- Up to 35% of Issue Queue power is saved
  - Arbiter enable signals
  - Bid logic
  - Selection logic
- Up to 20% of Execution Unit power is saved
  - Disable clocks in unused portions

COMPONENT POWER REDUCTION WITH PLB



Chip-Wide Power Estimates



- 17-35% of IQ power saved
  - % of Total power: 4-8%
- 10-20% of Execute Unit power saved
  - % of Total power: 2-4%
- Up to 12% of total power saved using PLB

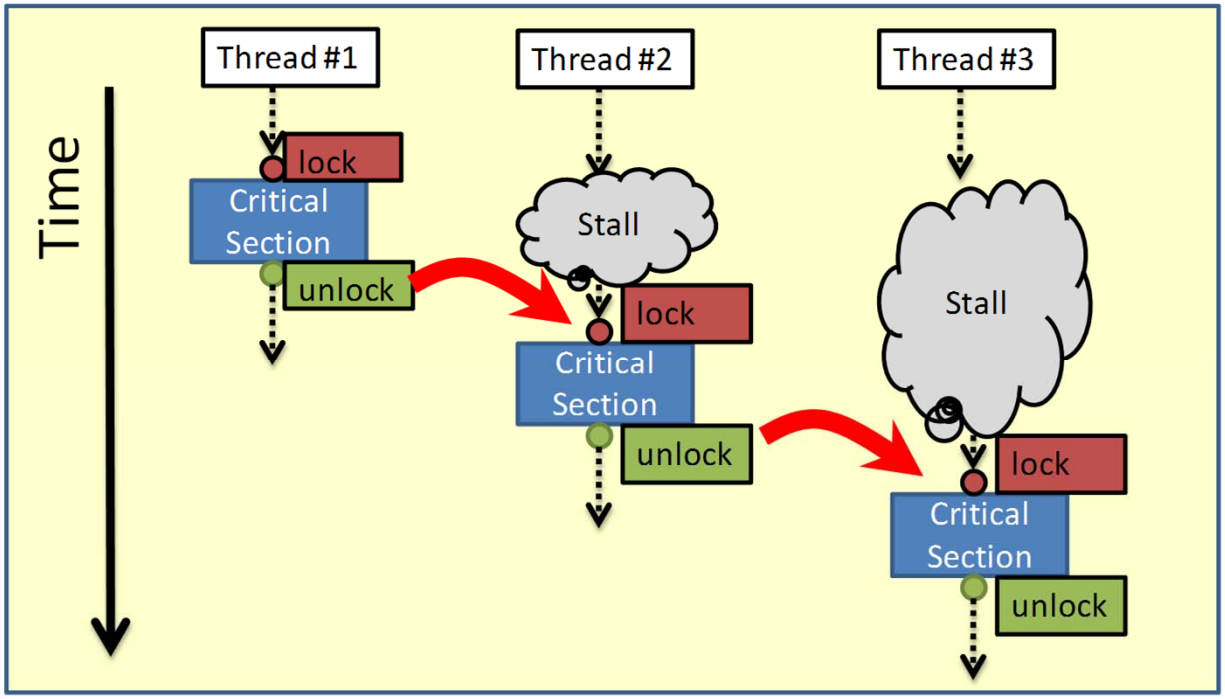
Energy-Efficient Transactional Memory for Embedded Systems

MULTICORE ARCHITECTURES

- Like GP systems, embedded platforms have turned to multicore architectures
  - Better scalability and design cycle time
- Issues:
  - Required High Level Of Parallelism
  - Synchronization Must be Efficient
- What's Special about Embedded Multicore:
  - Simplicity! (small caches, no OS support)
  - Power matters (as much as throughput)

MULTIPROCESSOR SYNCHRONIZATION

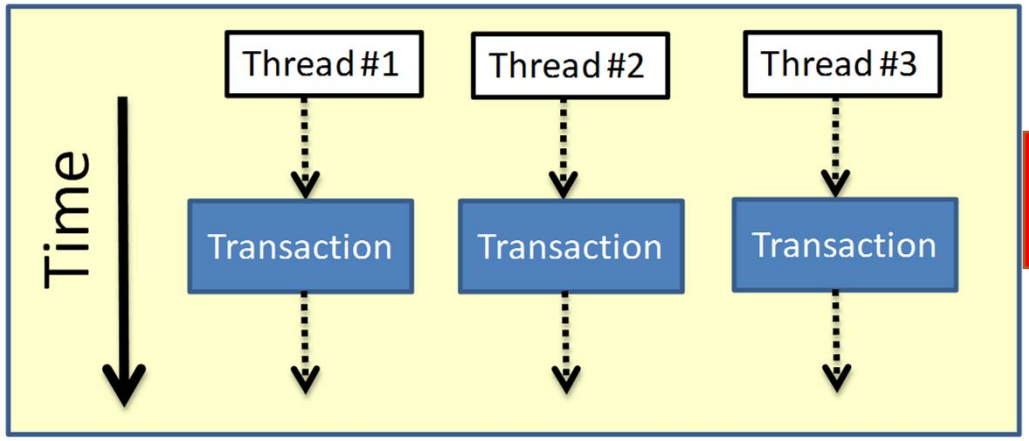
- Focus: Shared Memory Model (widely accepted)
- Common Approach: Lock-based Synchronization
  - **Idea**: mutual-exclusive access to shared data (i.e., critical sections)



No Concurrency

TRANSACTIONAL PROGRAMMING

- Key Features:
  - Optimistic (i.e., concurrent) execution of critical sections
  - Threads run in isolation
  - Atomicity (roll-back in case of conflict)



❑ Higher Performance than Fine-grained Locking (*without* reliability issues)

TM for Embedded Systems

- Software-TM schemes (STM or HW-accelerated STM):
  - Flexible, but heavyweight
  - Require OS support
  - Too slow (and power hungry...)

- Hardware-TM schemes (HTM):
  - Less flexible, but **simple**, **power-efficient**, and higher performance
  - Transaction sizes may be bounded, but...
  - For embedded systems, resource requirements are well understood

  Full HW-TM seems a natural design choice for embedded systems

❑ ***New challenges*** :
- ➢ Peculiar Memory Hierarchy (eg. tightly-coupled private memories)
- ➢ Limited resources (eg. smaller caches, no OS)
- ➢ Energy Constrained environment

1ST WORK IN EMBEDDED HTM: SOC-TM

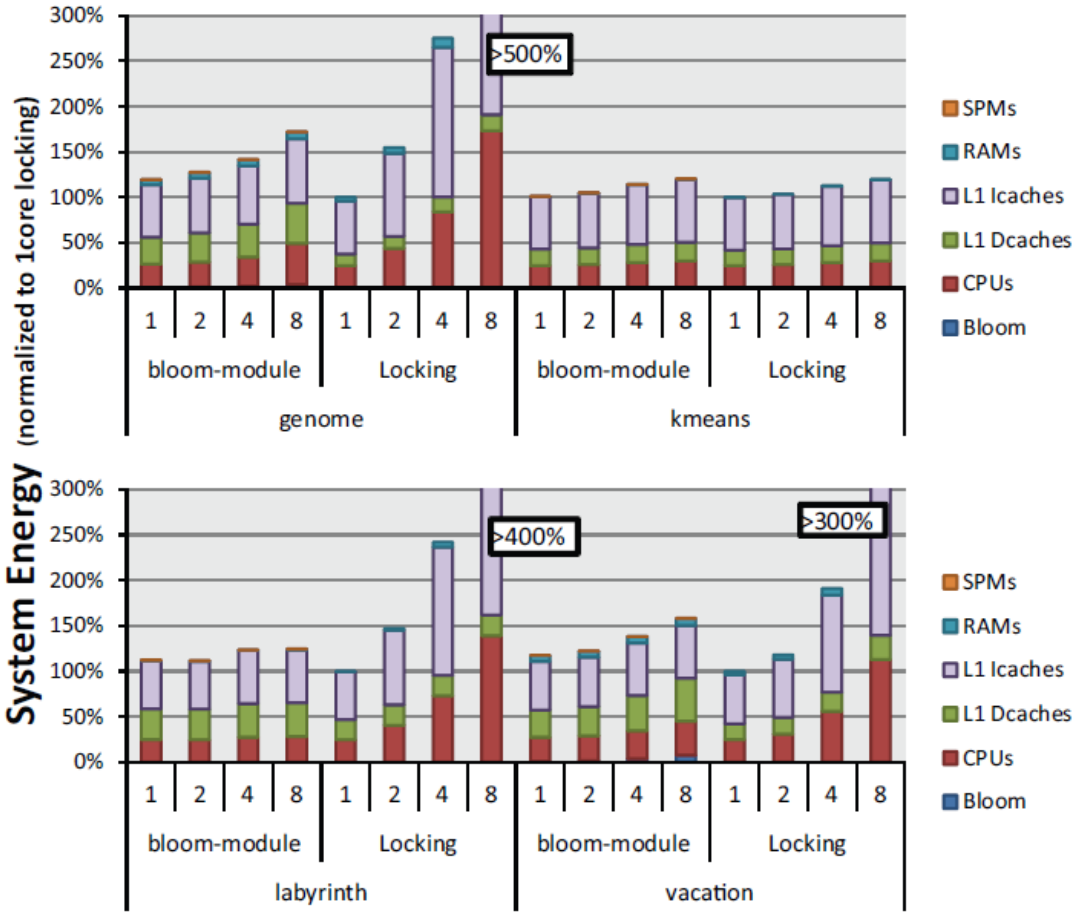
- Simplify contention management
  - Disentangle HTM from cache coherence
  - Bloom Module for conflict detection
  - More flexibility for conflict resolution policies
- Create an integrated HW/SW solution for transactional programming on MP-SoCs
  - Expands on Transactional OpenMP (OpenTM)
  - Adds support for speculative task- and data-level parallelism

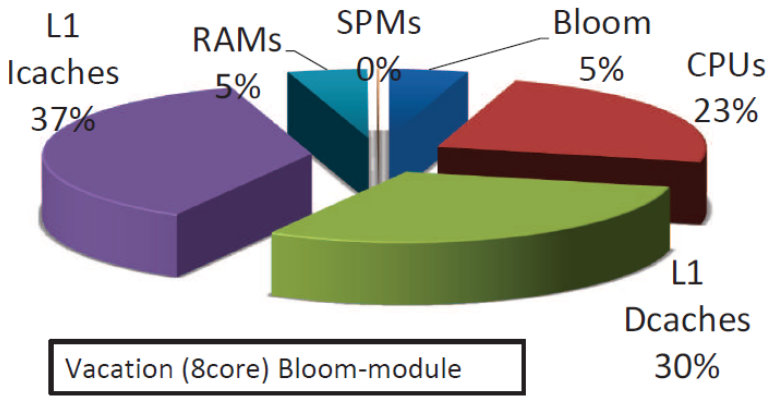
Simplifying Conflict Detection

- Originally, conflict detection was through the cache coherency protocol of each core
  - Requires modification to MESI protocol
  - Hard to exercise control over conflict resolution policy
- **Bloom Module:** centralized module for read/write conflict detection
- Key Features:
  - Decoupled from Cache Coherency
  - Keeps Tx histories in per-core Bloom signatures
  - Detects conflicts by comparing bits in the signatures
  - Programmable (Any algorithm can be used by the module to decide which core to abort)

EXPERIMENTAL RESULTS

- STAMP benchmarks
  - energy consumption comparison with locks





2ND WORK: EMBEDDED–TM

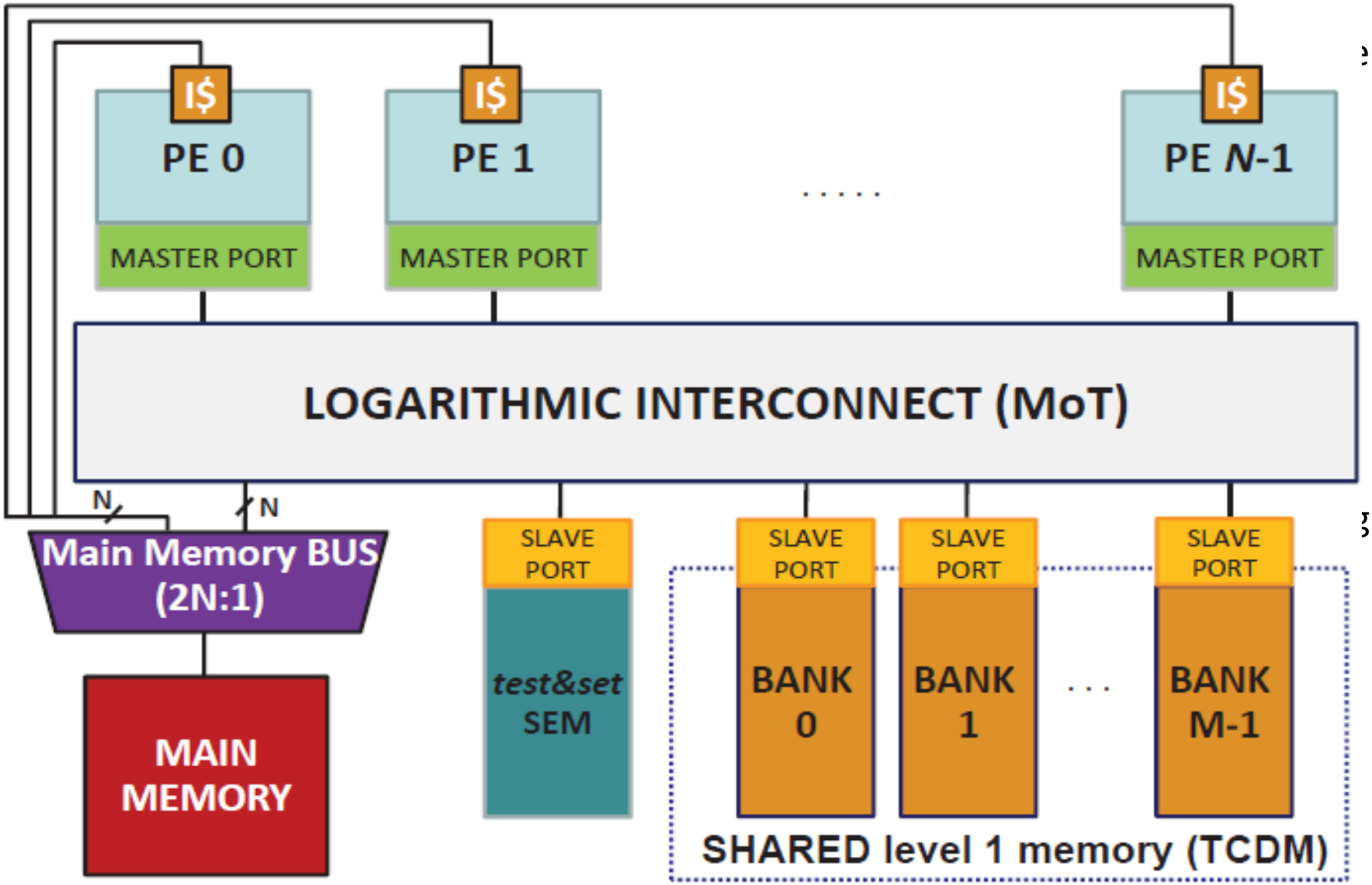
- **Limitations of SoC-TM**
  - Used a *centralized* unit to snoop on the bus traffic.
  - Bus-based architecture: one transaction/cycle.

- **Next Work:** First implementation of HTM for cluster-based NUMA embedded systems without cache-coherence.

- Lack of cache-coherence brings major challenges in HTM design.

- **Contributions:**
  - Building from scratch a self-contained HTM scheme.
  - Propose novel hardware support for tracking memory accesses.
  - Single-cluster but extendable to multi-clusters.
  - Performance evaluation.

TARGET ARCHITECTURE



EMBEDDED-TM IMPLEMENTATION

- **In our case:** Logical Interconnect allows multiple transactions/cycle. Snooping ➔ bottleneck, limits scalability.

- Multiple per-bank **Transactional Support Modules (TSM).**
  - Distribute conflict detection/resolution across TCDM banks.

Distributed Transactional Memory Control.

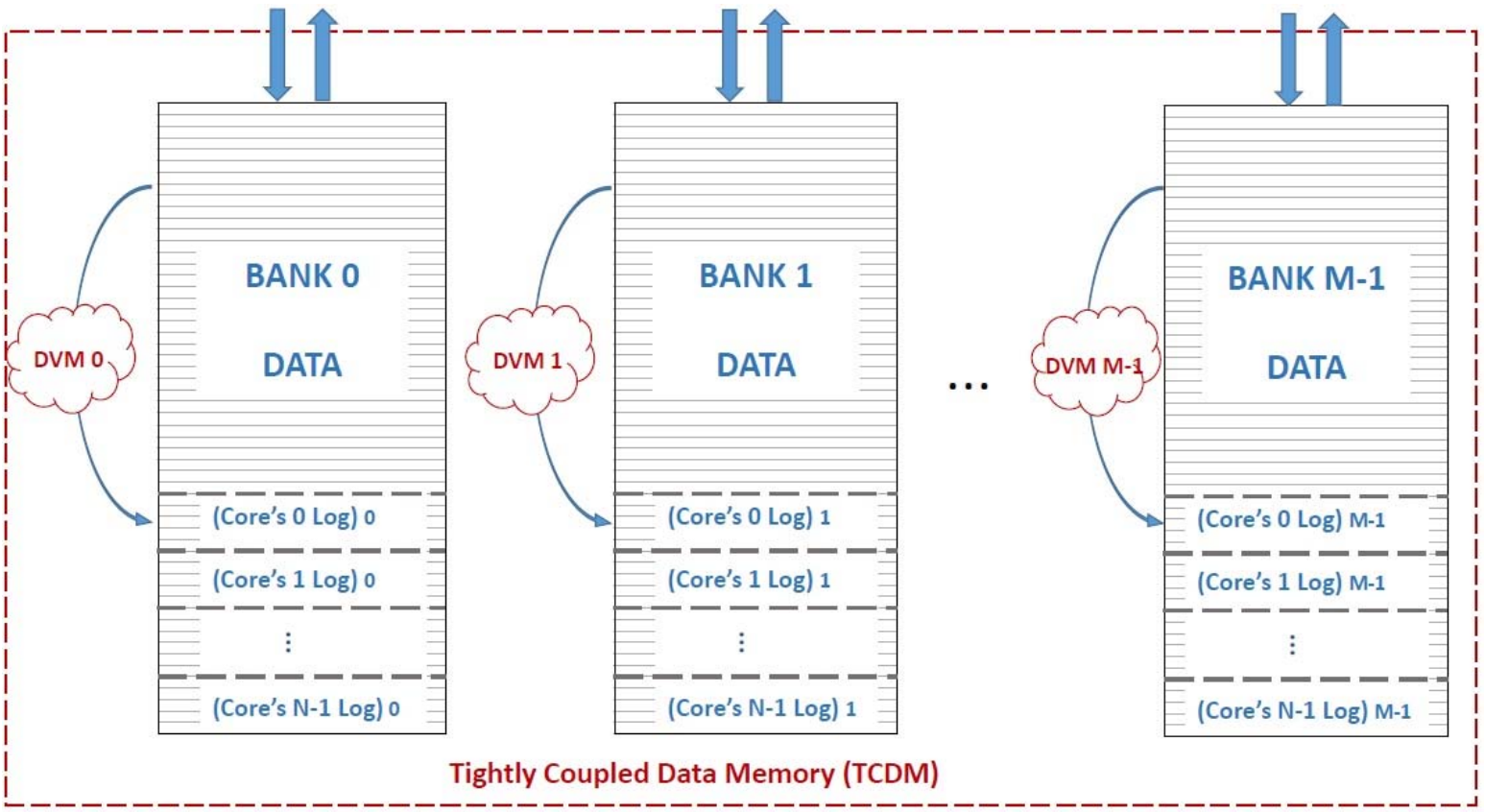
- Our TM support design:
  - a) Transactional Bookkeeping
  - b) Data Versioning
  - c) Rollback Mechanism.

A) TRANSACTIONAL BOOKKEEPING

- Mechanism that keeps track of transactional readers/writers in memory.
- Each bank's TSM intercepts all memory traffic to bank. Upon conflict, it notifies conflicted cores.
- For each data line: Multiple Readers, Only one writer/owner.
- Write to a location read by another core ➔ **Conflict!**
  Read to a location read by other cores ➔ **No Conflict!**
- Each bank keeps track of readers/writers per data line through per-bank array of k r-bit vectors,
  $r = 1 + \log_2 N + N$, N: # cores in cluster, k: # data lines per bank,

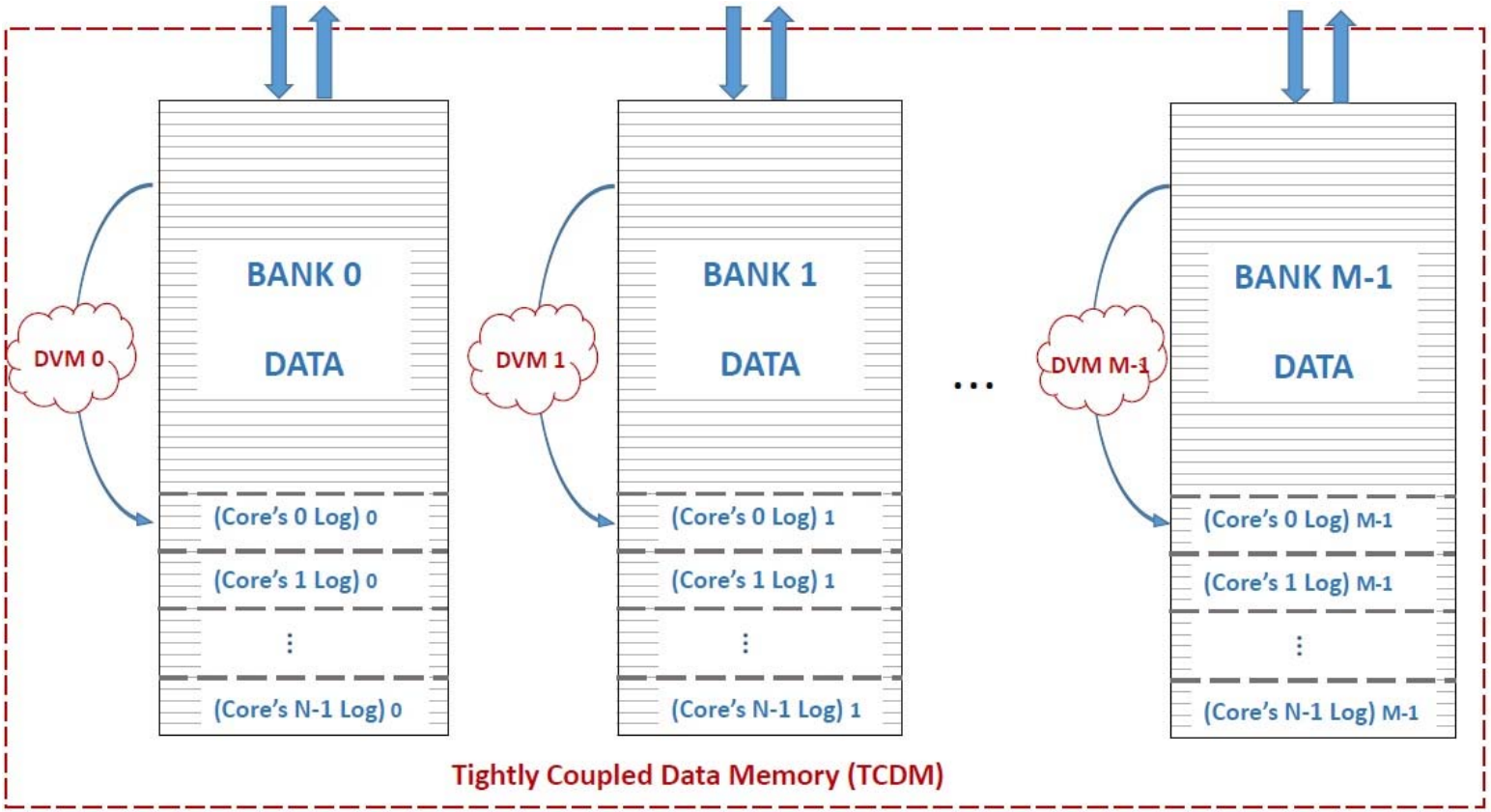
	Owner Bit	Writer ID	Readers (1 bit per core)
Time t2:	0	0 0 0 0	0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0
Bit #:	20	19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

B) Data Versioning



- Original data saved in ***Logs***, distributed among TCDM banks.
- In each bank: fixed-size log space per core.
- Each core's log includes entries for modified data for that bank only
- **Data Versioning Modules (DVM)** monitor accesses for its bank only.

B) DATA VERSIONING

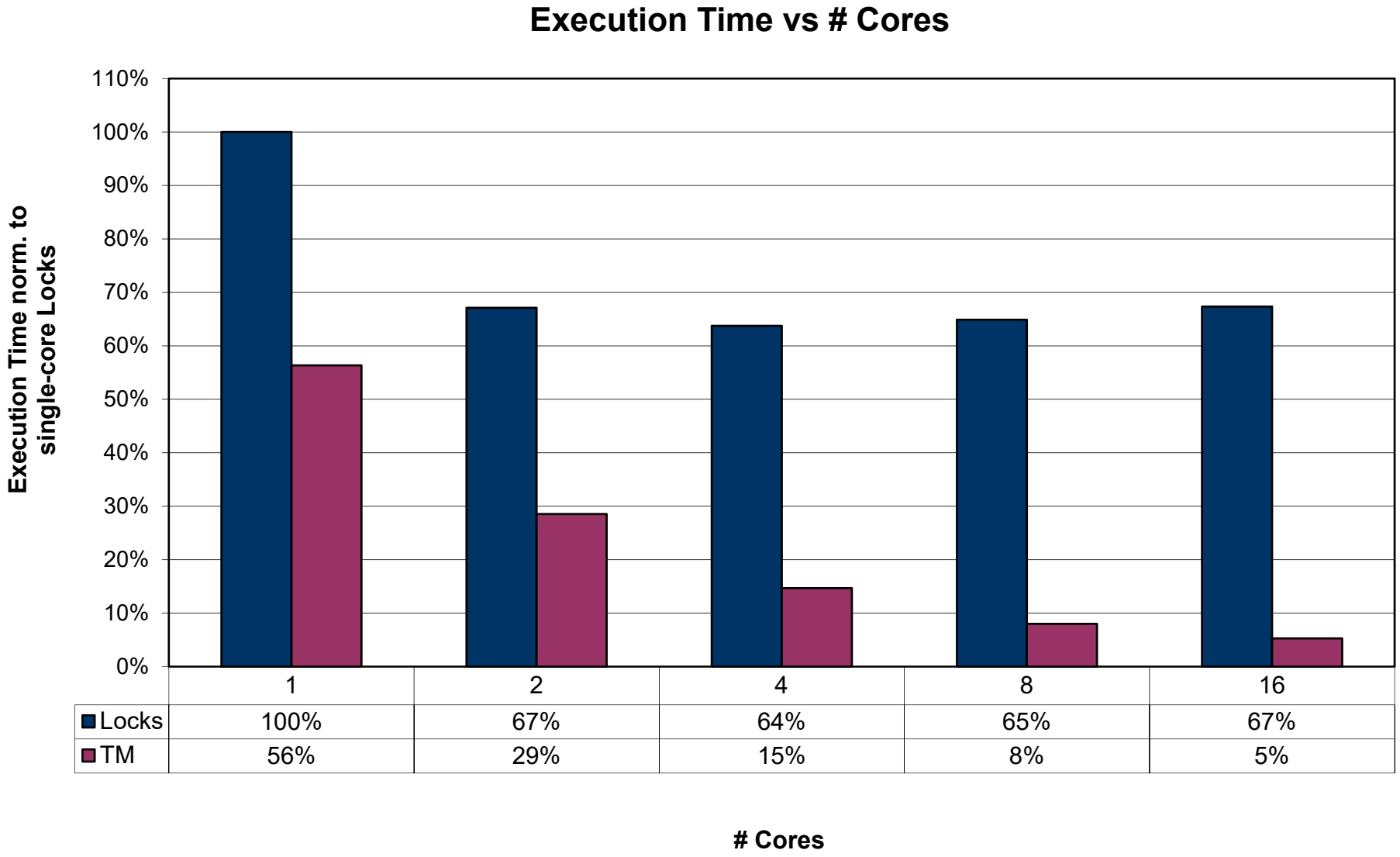


- ***Transaction begins:*** save state of core (PC, registers, SP, stack)
- ***On a write***: DVM checks log for entry, creating new entry only once.
- ***On a data conflict:*** Each bank's log is traversed. Data restored.
- ***Transaction completes with no error:*** logs are cleared.

C) ROLLBACK MECHANISM

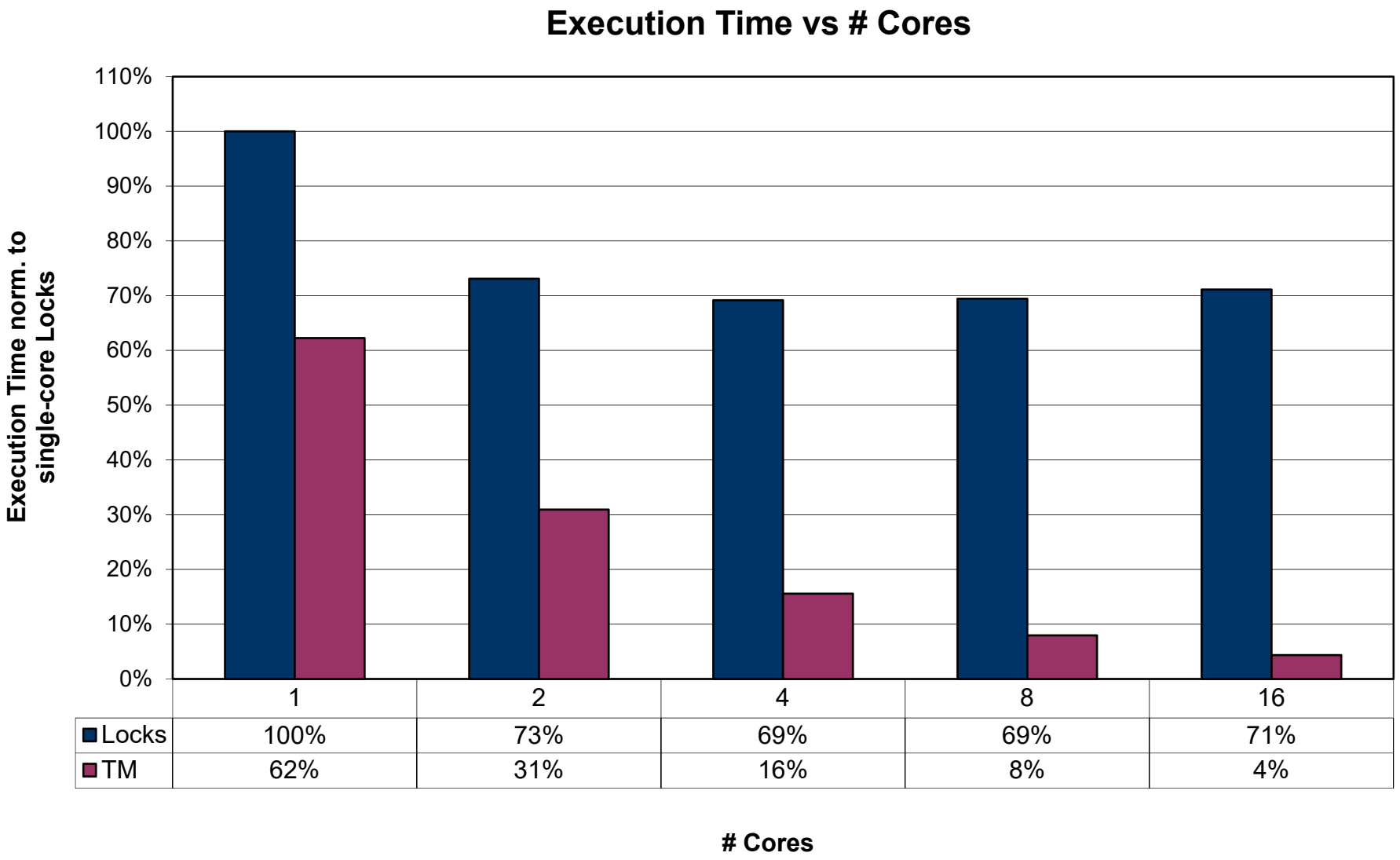
- Conflicts are detected while transaction executes
  - We choose to abort the “requester”
  - Triggers a call to *abort_transaction*() function
  - Restores saved state from logs of all banks in TCDM for requester transaction
  - Aborted transaction tries again after some backoff period

1) REDBLACK



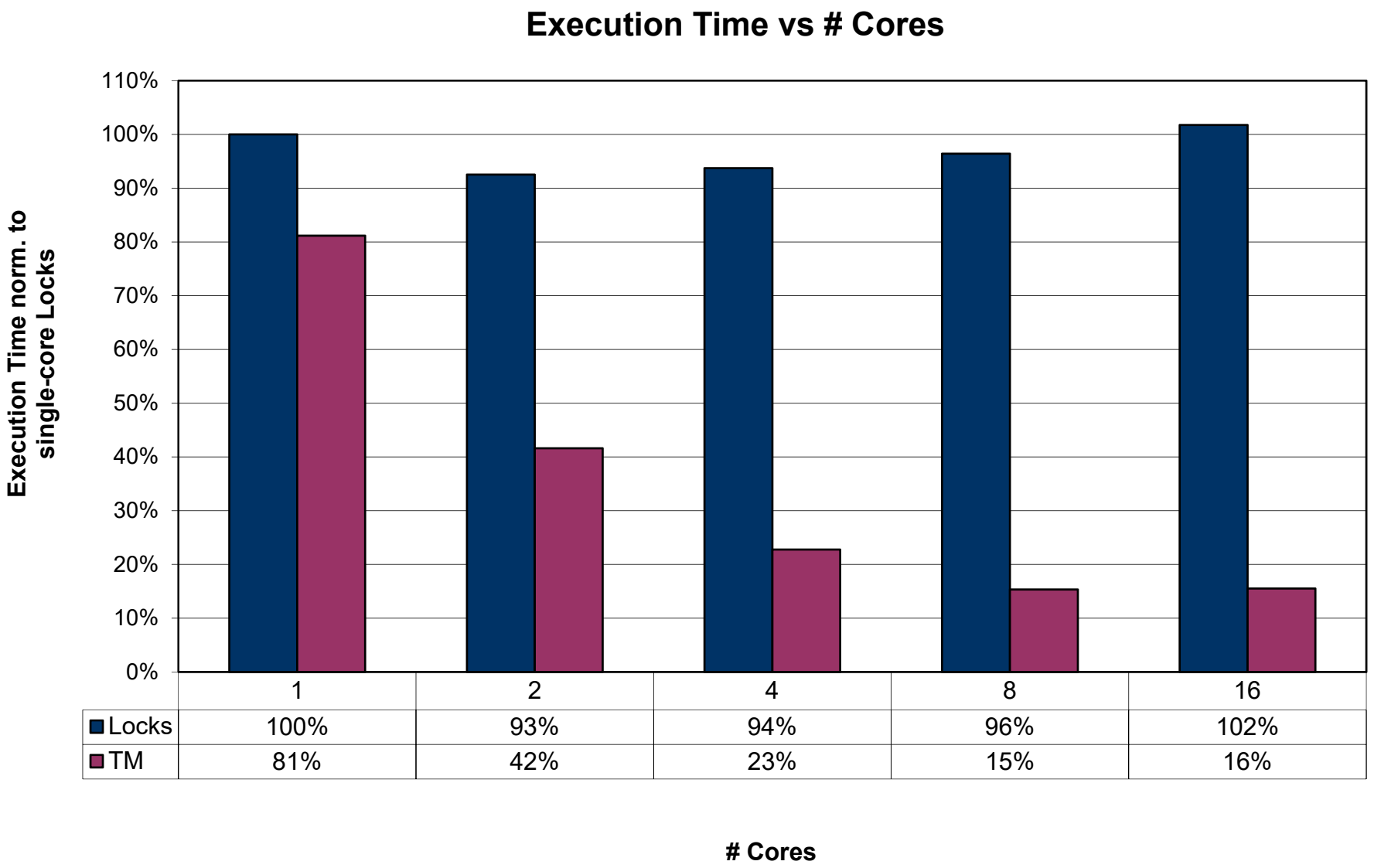
	1	2	4	8	16
Locks	100%	67%	64%	65%	67%
TM	56%	29%	15%	8%	5%

2) SKIPLIST



	1	2	4	8	16
Locks	100%	73%	69%	69%	71%
TM	62%	31%	16%	8%	4%

3) GENOME



	1	2	4	8	16
Locks	100%	93%	94%	96%	102%
TM	81%	42%	23%	15%	16%

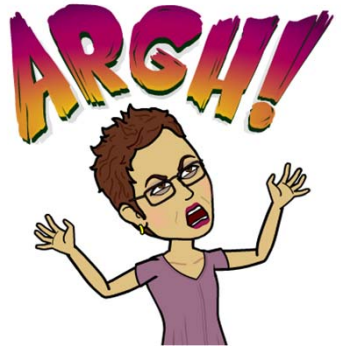
EXPLOITING HARDWARE TRANSACTIONAL MEMORY FOR ERROR TOLERANCE AND ENERGY EFFICIENCY

VARIABILITY-INDUCED ERRORS

- Shrinking transistor sizes has left devices more susceptible to variability
  - **Static Variations**: lead to performance/power mismatches
  - **Dynamic variations**: aging, voltage drops, temperature fluctuations
- **Solution:** Conservative guardbands on the operating frequency/voltage

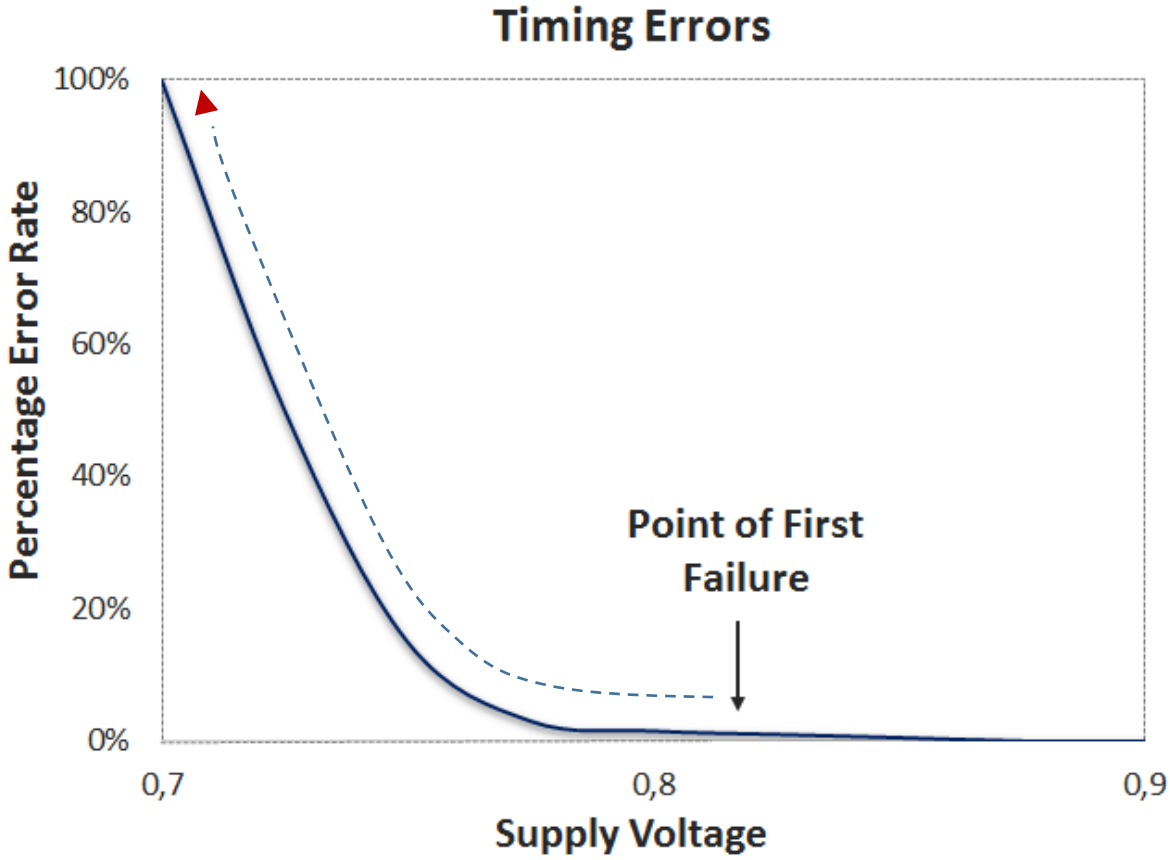
Performance ↓ ***Energy*** ↑

- **Reduce guardbands?**
  - Intermittent timing errors
  - Critical operating point (COP)



A) Intermittent timing errors

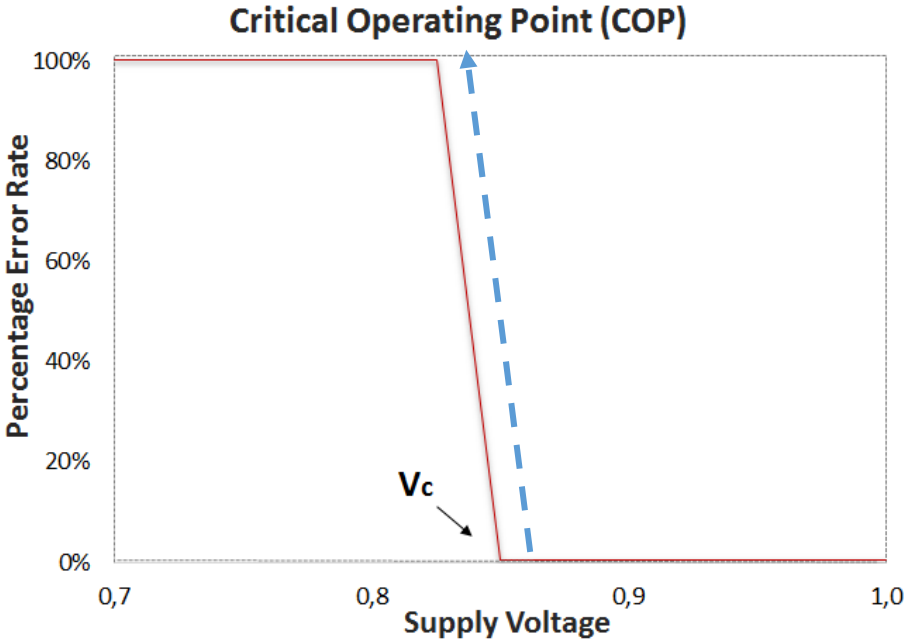
Intermittent Timing Errors cause erroneous instructions with wrong outputs or incorrect control flow.



B) Critical Operating Point

Critical Operating point (COP): (Vc, Fc) pair that if surpassed:

⇨ **Massive instruction failures.**



COP Hypothesis : In large CMOS circuits, there exists a critical operating frequency F_c and a critical voltage V_c for a fixed temperature **T**, such that:

- Any $F > F_c$ causes massive errors.
- Any $V < V_c$ causes massive errors.
- Any $F < F_c$ and $V > V_c$,
  no errors occur.

COP might change over space/time.

In a many-core environment:

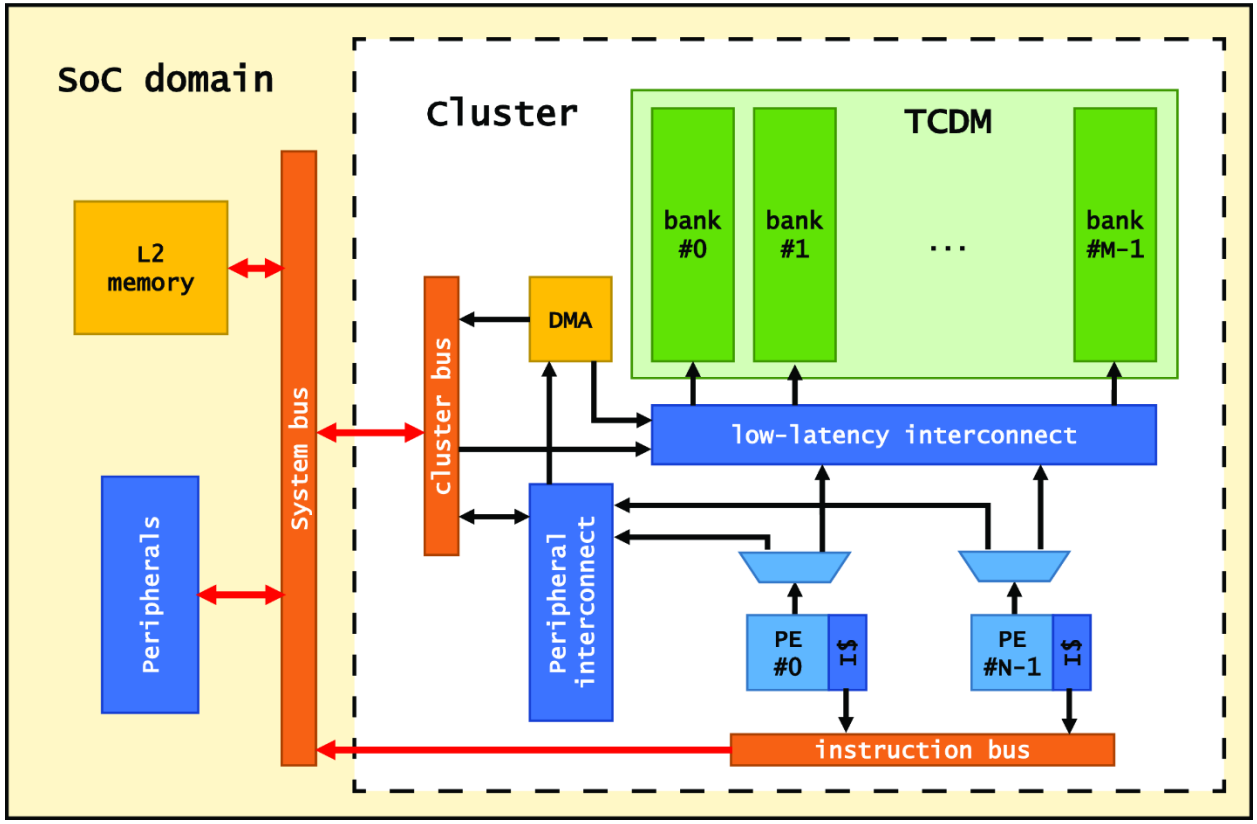
- COP might differ among cores
- Become unsafe.

CONTRIBUTIONS

- We need a recovery mechanism to **avoid guardbands** and operate at lower voltage to **save energy**.
- **Edge-TM:** HW/SW scheme for **error resilience** and **energy efficiency:**
  - Scales down the voltage to save energy
  - Monitors errors
  - **Hardware Transactional Memory (HTM)** for error recovery (Intermittent timing errors and COP).
- Our approach:
  - Enables operation at **highly reduced supply voltage margins.**
  - **Forward progress**.
  - **Low overhead**.

EDGE-TM ARCHITECTURE DESIGN

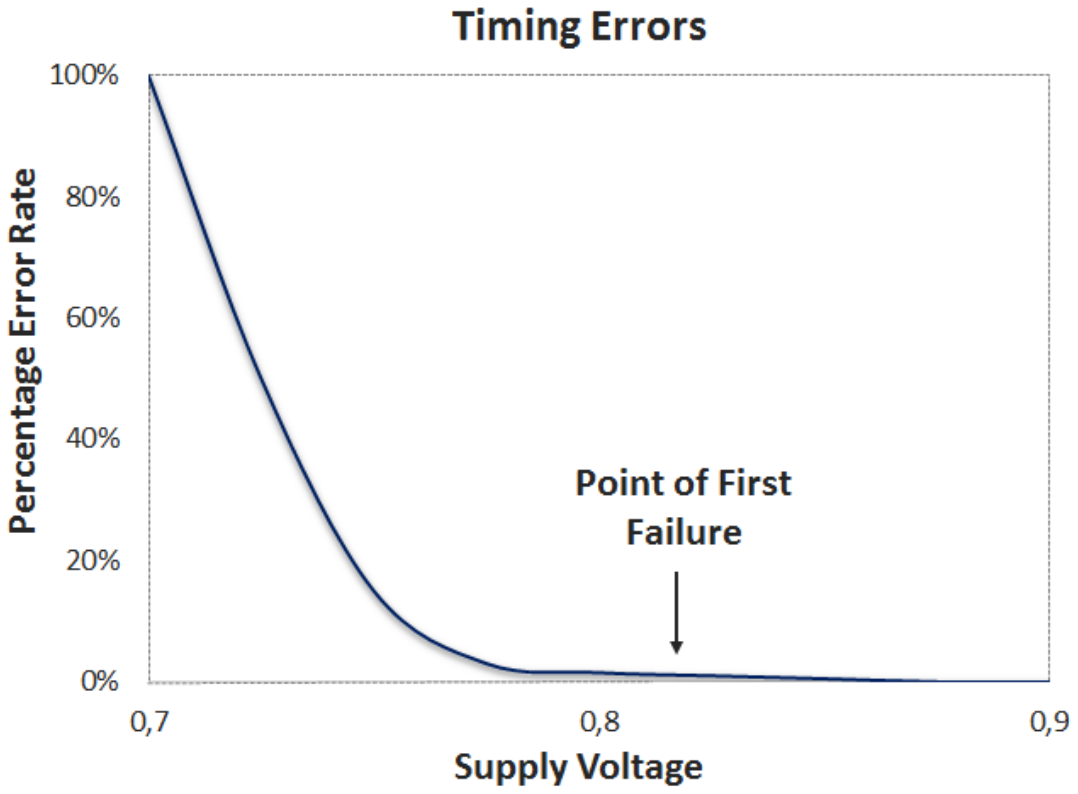
- **Parallel Ultra-Low Power Platform (PULP):** Cluster-based shared memory embedded architecture.



- **Edge-TM:**
  - Hardware extensions for error tolerance: a) Error Detection, b) Error Correction.
  - Software directed DVS error management policies for energy efficiency.

A) Error Detection

- **Error Detection Sequential (EDS):** Error detection circuitry for each core.
- Probabilistic Error Model (Fojtik et al.): Expected error rate vs. Supply Voltage for intermittent timing errors.



B) Error Correction

- **HTM:** A speculative execution mechanism, used for synchronization in multicore environments.

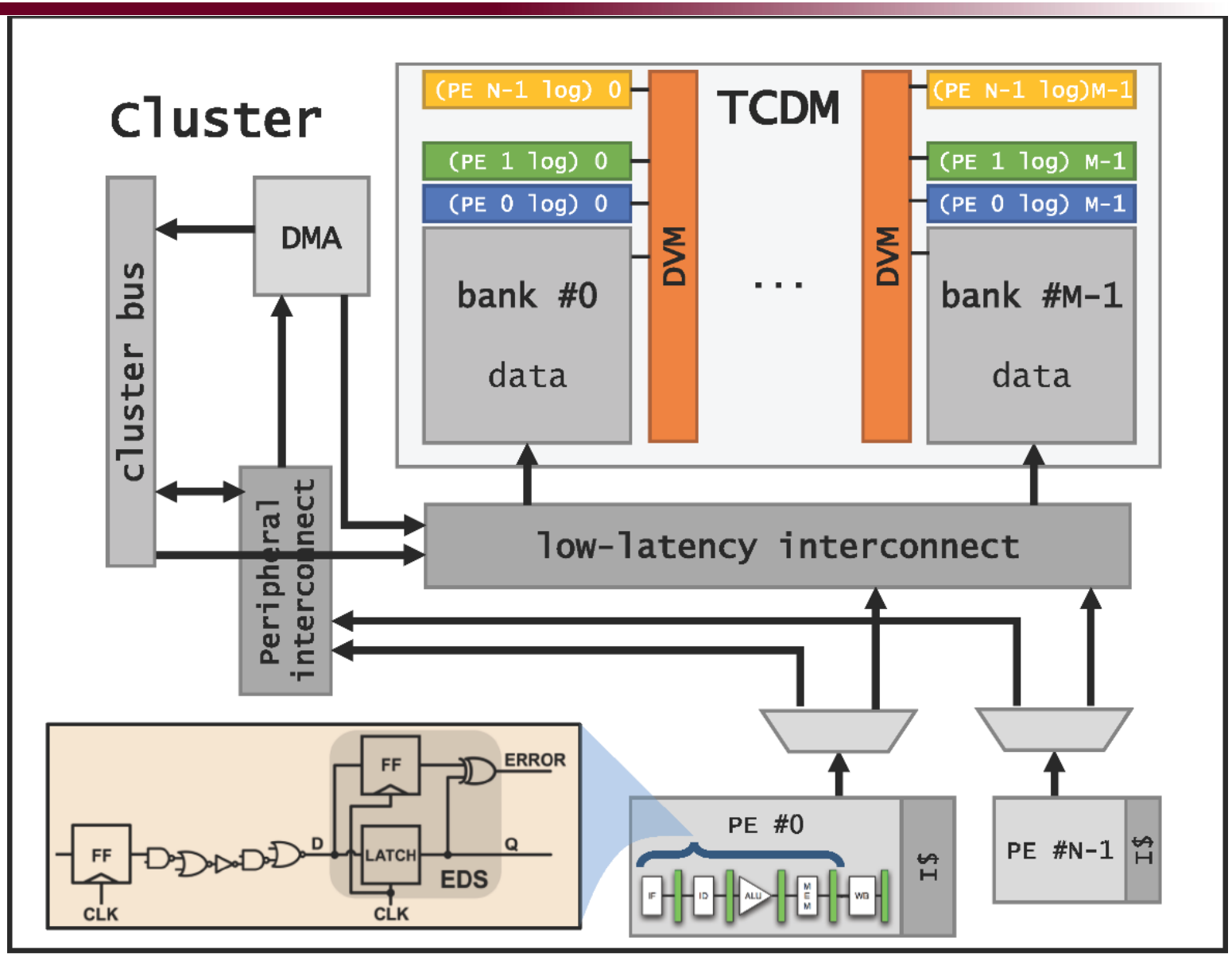
- Traditionally HTM requires:

  1. Bookkeeping **2.** Data versioning **3.** Checkpointing/Rollback

- **Data Versioning:** Fast distributed logging scheme saves original data.

- **Checkpointing/Rollback:**
  - Transaction boundaries are defined using OpenMP constructs for parallelism.
  - Transaction Starts: A snapshot of the processor state is taken.
  - If error occurs: **Transaction Aborts**. State and Logs are restored.
  - If no error occurs: **Transaction Commits**.

PULP CLUSTER EXTENDED FOR ERROR TOLERANCE



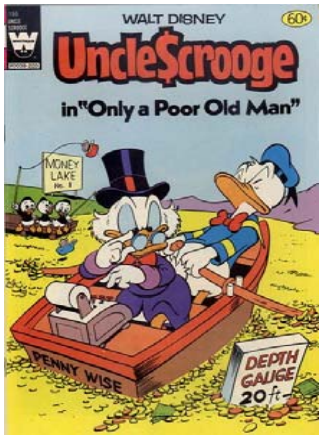
EDGE–TM DVS ERROR MANAGEMENT POLICIES

a) Point of First Failure (PoFF) policy: Operates just above the edge of failure.

- Lowers voltage until the first failure. Then voltage is step increased.

b) Thrifty Uncle- Reckless Nephew (TURN) policy:

- Lowers voltage beyond the point of first failure.
- Adjusts voltage based on aborts and commits.



Trade-off: Further voltage scaling saves energy but increases abort rate

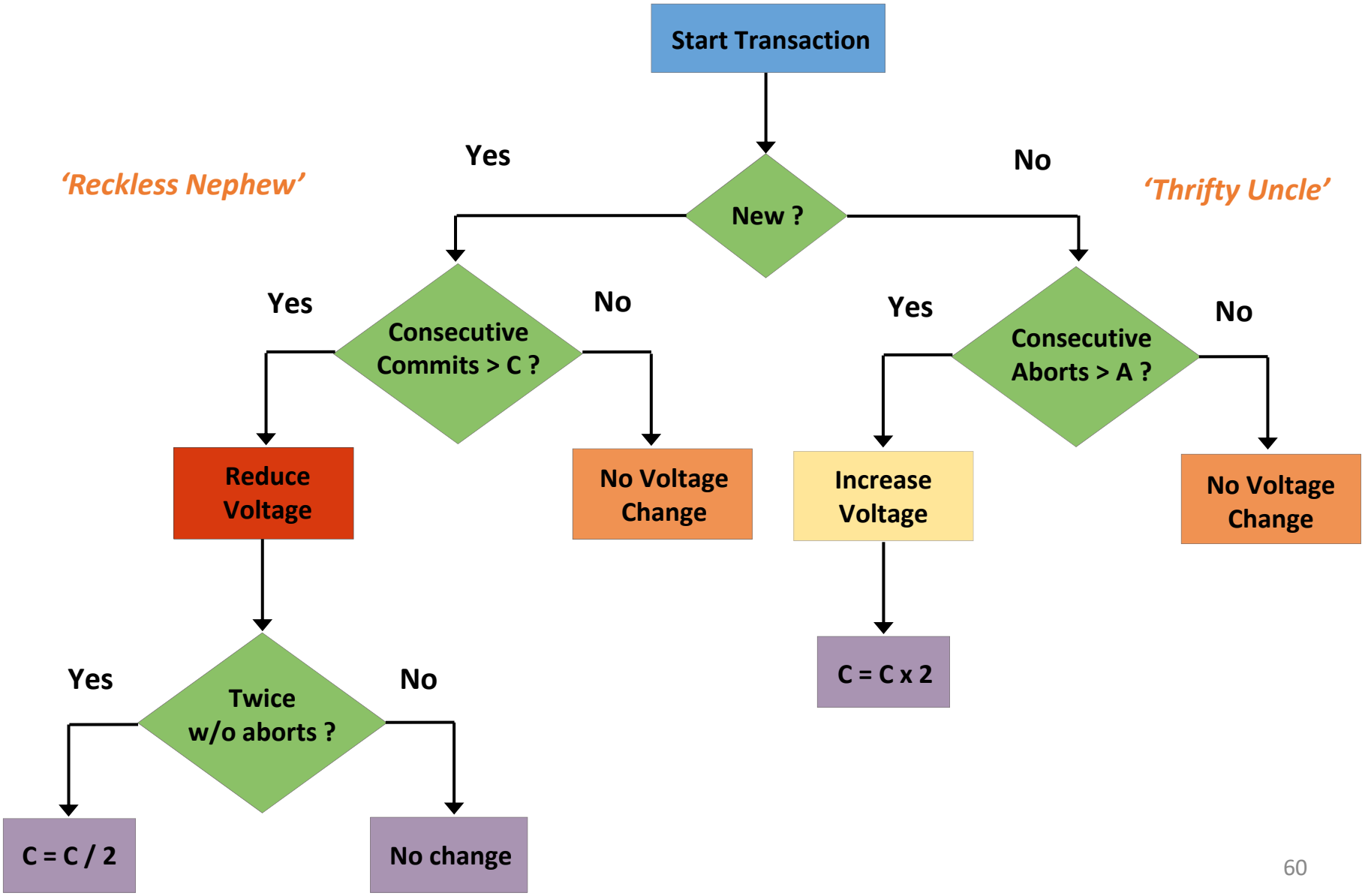
⟹ Energy and time is wasted in recovery/re-execution.

- Two parts:

1) Reckless Nephew: Scales down voltage for further energy savings.

2) Thrifty Uncle: Moderates energy loss due to aborts by setting up a threshold for voltage scaling based on **number of consecutive aborts: A and commits: C**.

THRIFTY UNCLE/RECKLESS NEPHEW POLICY



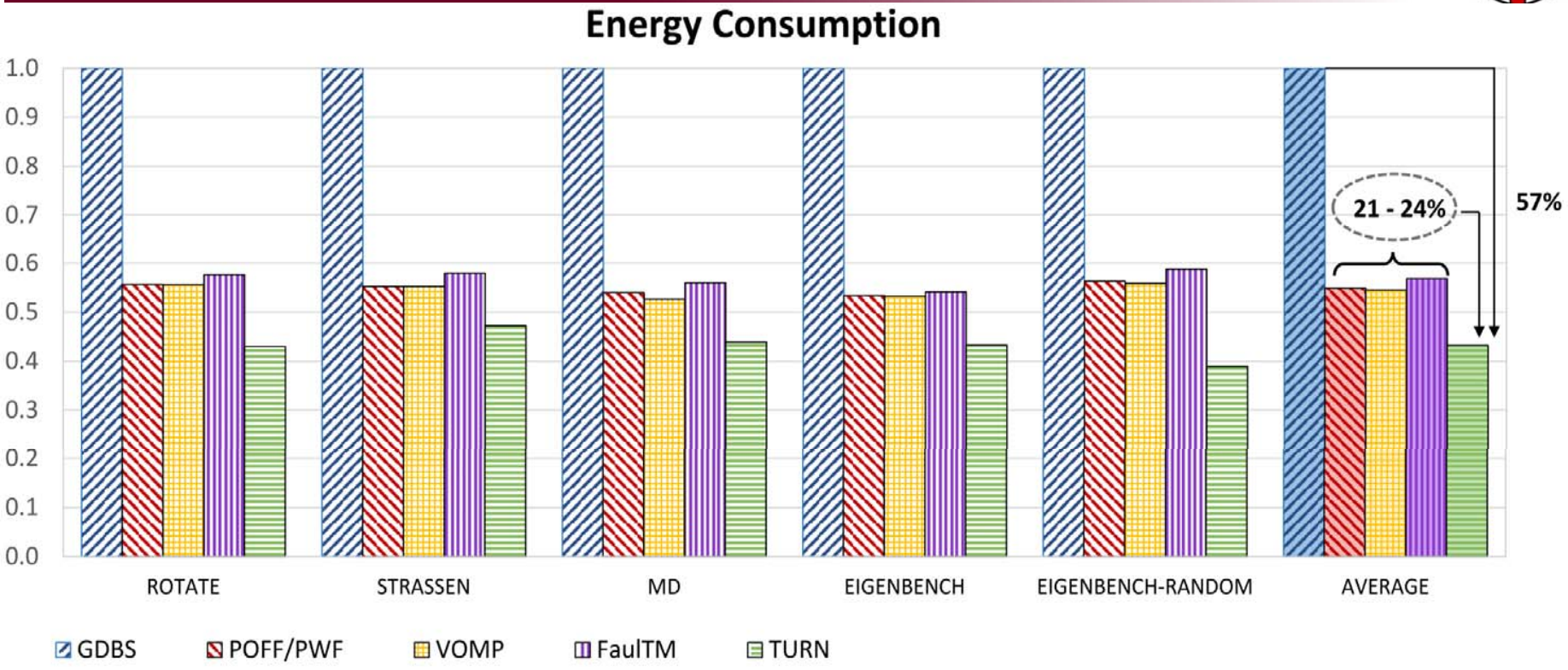
EXPERIMENTAL SETUP

- Power/Performance numbers from STMicroelectronics 28nm UTB FD-SOI implementation of PULP.

- Comparison of Edge-TM policies (**POFF** and **TURN**) with related work:
  - **VOMP:** Vulnerability-aware, error-tolerant task scheduling (Rahimi et al.)
  - **FaulTM:** Revisiting transactional memory for fault tolerance (Yalcin et al.)
  - **PWF:** Revisiting transactional memory for timing error-tolerance (Papagiannopoulou et al.)

- **GDBS:** Uses guardbands and keeps voltage steady (Baseline technique).

RESULTS: ENERGY CONSUMPTION



- **TURN** vs **POFF**/**PWF**/**VOMP**: 21%
- **TURN** vs **FAULTM**: 24%
- **TURN** vs **GDBS**: 57%
- **POFF** vs **GDBS**: 45%