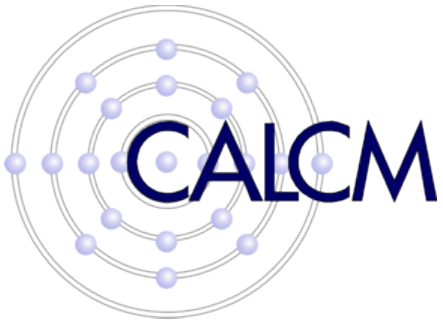
TRUSS: Reliable, Scalable Server Architectures

Babak Falsafi and James Hoe

Team: Eric Chung, Brian Gold, Jangwoo Kim, Eriko Nurvitadhi and Jared Smolens



TRUSS

Computer Architecture Lab

Carnegie Mellon

http://www.ece.cmu.edu/~truss

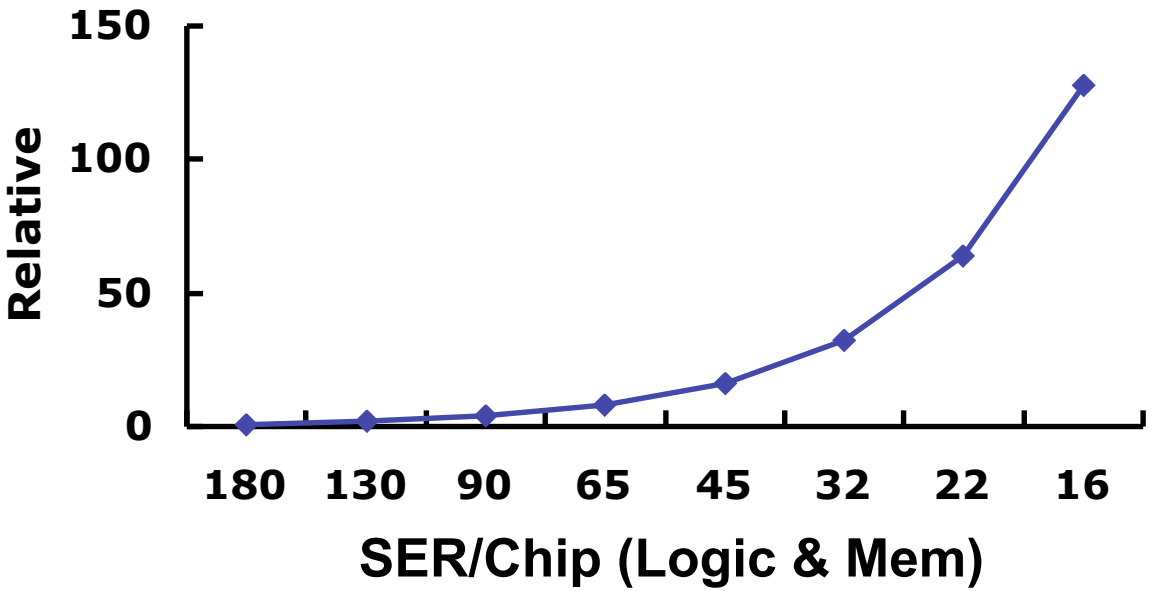
Scaling Trends by 2015
[Borkar, IEEE Micro'05 on Reliability]

- **We'll have 100B transistors on a chip**
  - … but 20B will be unusable
  - … another 10B will fail over time
  - … and at any point, some bit could flip spuriously
- **Reliable chips from unreliable devices – how?**
  - Harden all transistors? Not practical
  - Change all software? Not feasible

Need solutions at all levels of system stack

Sources of Error: Transient

- **Scaling — increasing density, decreasing charge**
- **Processor pipeline remains unprotected**
  - Complex, timing-critical datapath — can't do ECC

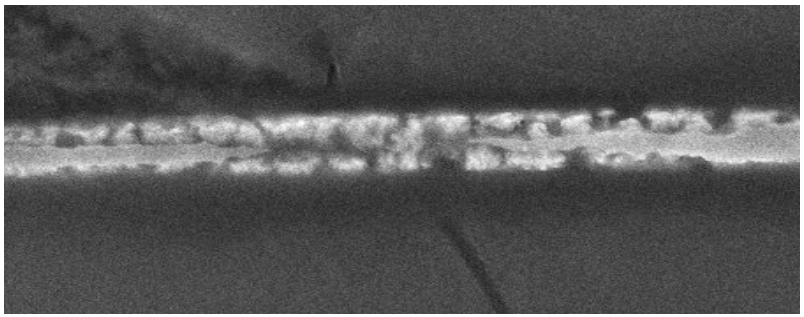


Source: Borkar, Intel

Exponential increase in bitflips!

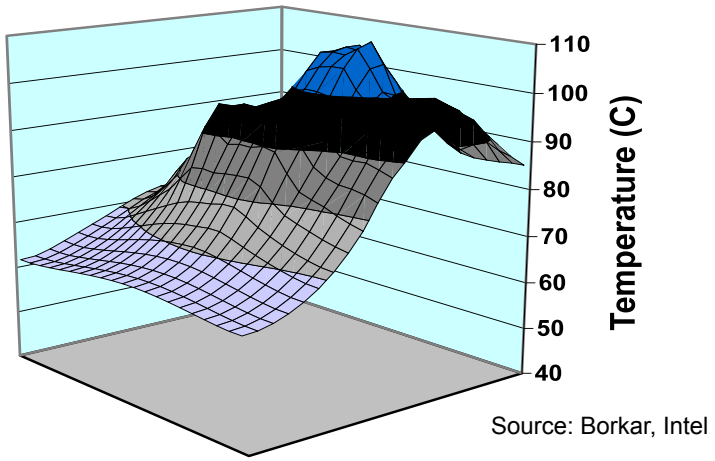
Sources of Error: Lifetime

- **Time-dependent variability**
  - Degradation: electromigration, oxide breakdown, etc.
  - Thermal: transistors switch slower in hot spots



Source:Zörner

Electromigration



Source: Borkar, Intel

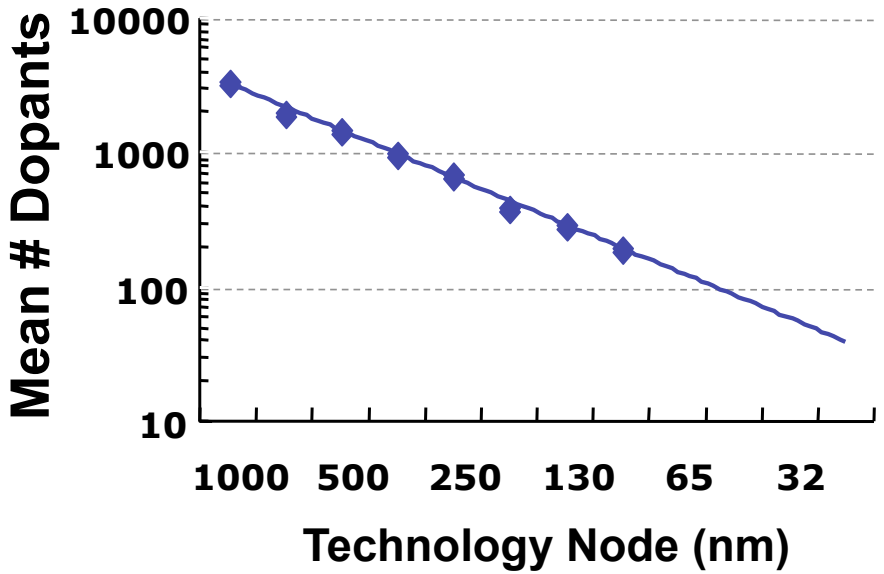
Temperature hot spots

Accelerated chip failure!

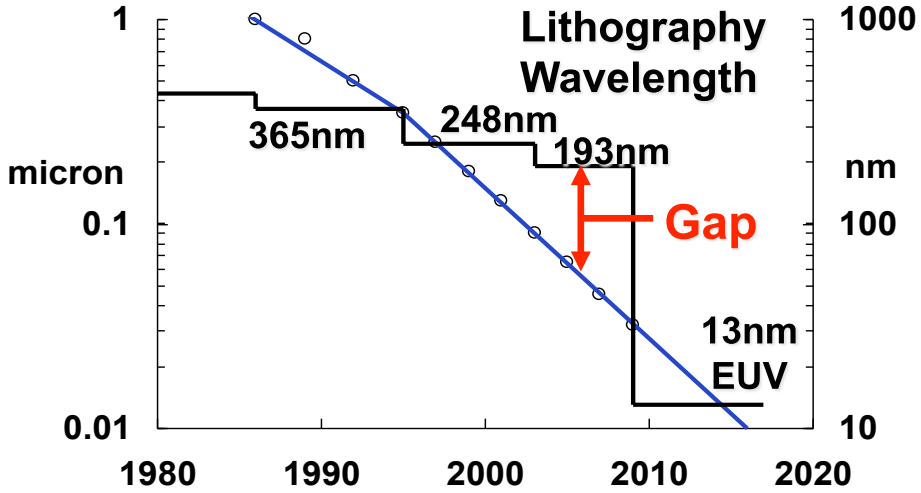
Sources of Error: Manufacturing

- Increasing variability at manufacture

Random Dopant Fluctuations



Sub-wavelength Lithography



Sources: Borkar/Bohr, Intel

- Burn-in testing not practical

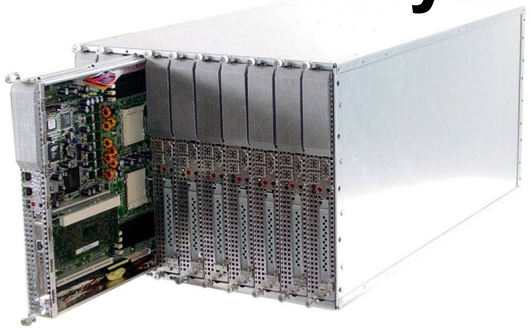
Dramatic increase in defect density!

Current Solutions not Sufficient

IBM z900

- **Want Reliability**
  - Both transient and permanent error
- **Want Software transparency**
  - Port existing SMP code base

HP Himalaya



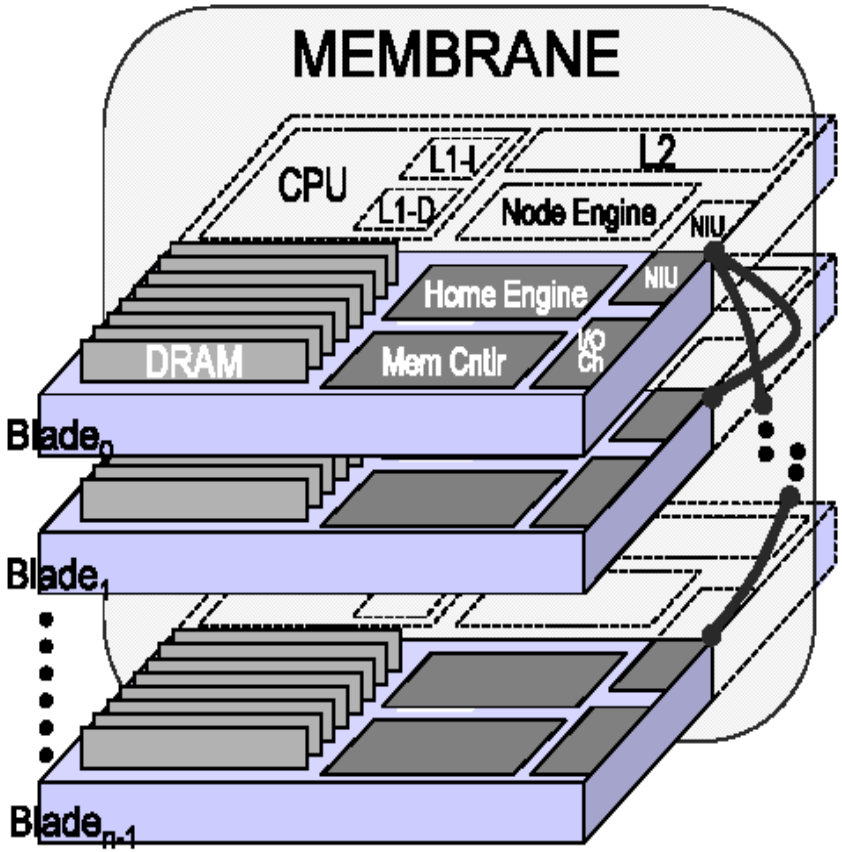
- **Want Scalability**
  - Both in cost and performance

No solution combines all three

Our Solution: TRUSS
[IEEE Micro'05 on reliability]

Reliable Server

- Distributed shared memory
  - Cost/performance scalable
  - Software transparent
- Tolerates
  - Multi-bit soft errors
  - Single component failure
- Available/serviceable



Prototype ca 2007

Contributions

Protecting Processors

1. Lightweight detection (fingerprinting)
   - Bounds latency, limits bandwidth needed for detection
2. Distributed redundancy
   - Enables soft- and hard-error tolerance

Protecting Memory

3. Distributed parity (DRUM)
   - Tolerates node failure and multi-bit soft error

Outline

- Overview
- **Computation redundancy**
  - Fingerprinting
  - Distributed redundancy
- **Memory protection**
  - Distributed parity
- **Summary**

Methodology: Infrastructure

SimFlex [SIGMETRICS'04]

- Full-system simulation of MP (boots Linux & Solaris)
- Real server software: (e.g., DB2 & Oracle)
- Statistically sampled timing models
  - Microarchitectural measurement in minutes
- Component-based design
  - Allows virtual prototyping with FPGA components

Publicly available at

http://www.ece.cmu.edu/~simflex

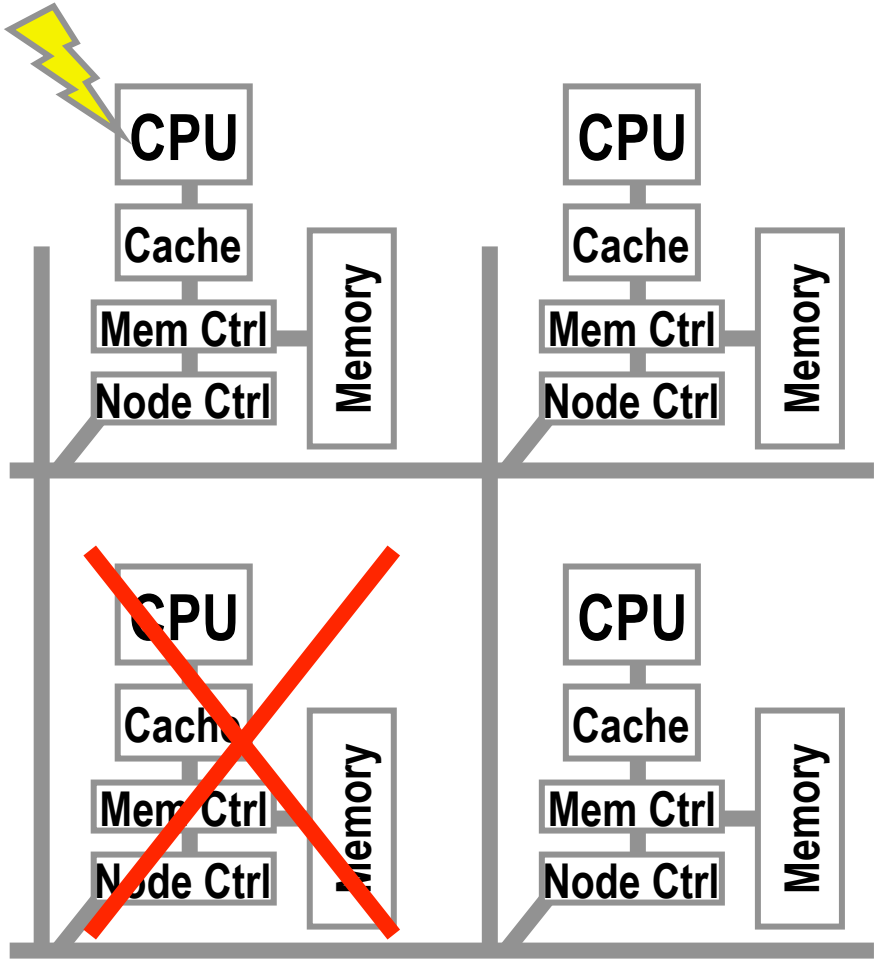
TRUSS Fault Model

Transient Faults

- Cosmic rays, alpha particles, …
- Tolerate any single soft error

Permanent faults

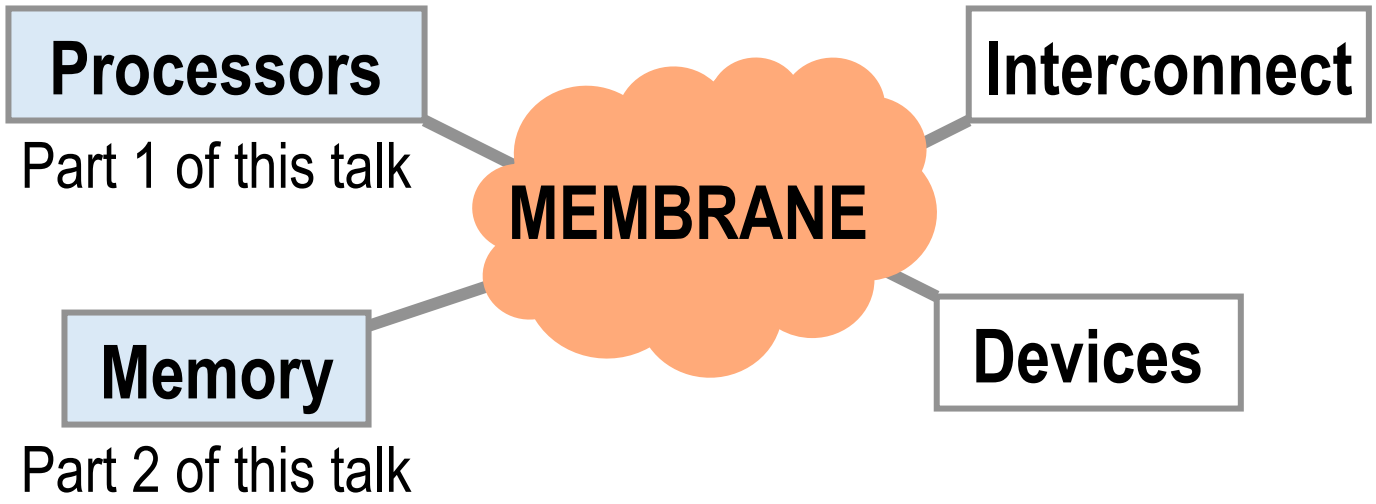
- Device wearout, thermal cycling, …
- Tolerate up to node failure



How to reason about faults in complex system?

MEMBRANE: A Fault "Barrier"

- **Decompose system into domains**
  - Each domain must detect, recover errors locally
  - Errors cannot propagate across domains



Enables optimized protection for each domain

(1) Protecting Processors

Redundant execution within core

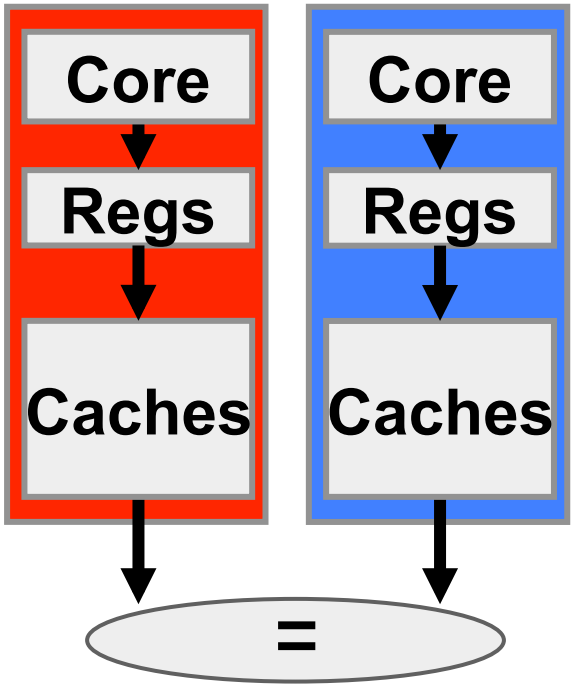
- Detect/recover soft errors

Redundant execution across chips

- Tolerate soft and hard errors

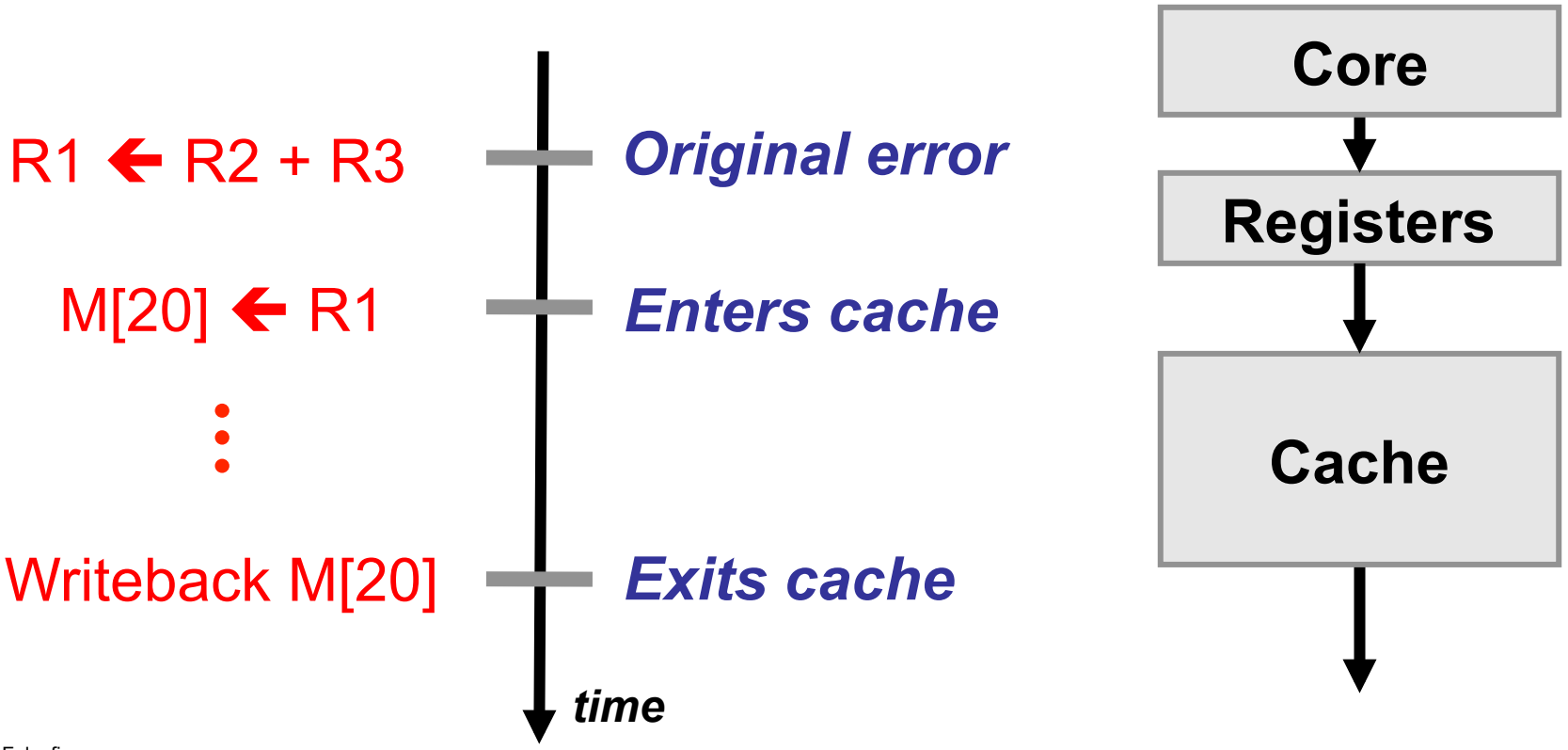
Key challenges

- How to detect errors?
  - Need low latency, low bandwidth
- How to coordinate execution?
  - Maintain redundant instruction stream

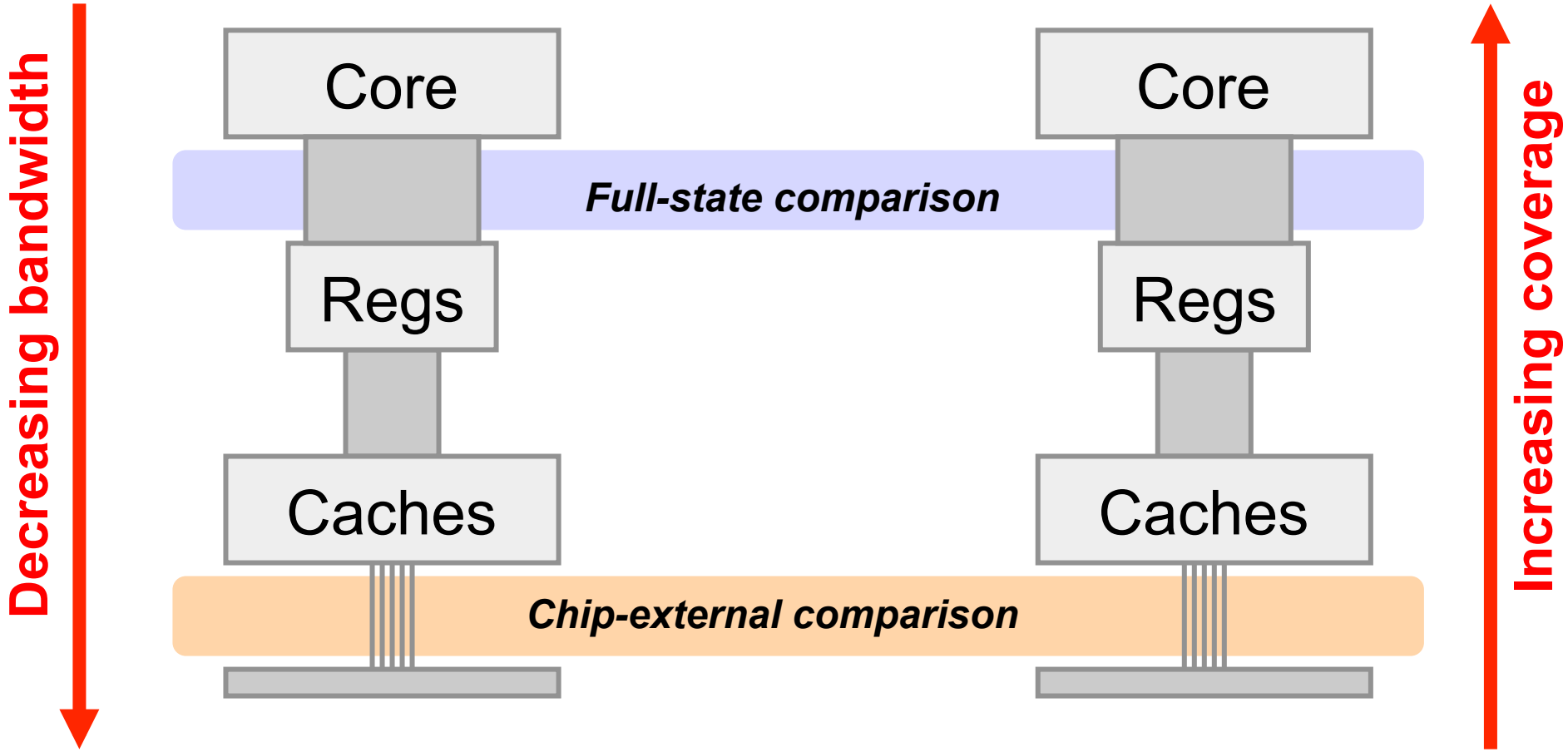


Error Detection: Latency

- **Existing solution: compare chip-external traffic**
  - Errors can hide in cache for millions of instructions
  - Recovery harder with longer detection latencies



Error Detection: Tradeoffs



Want high coverage with low bandwidth

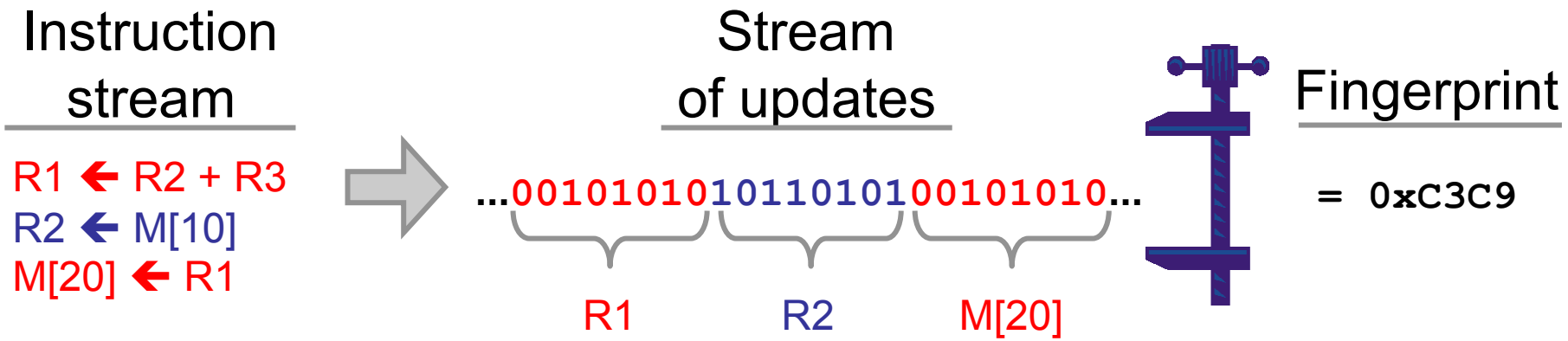
Error Detection: Fingerprinting

[IEEE MICRO top pick'04]

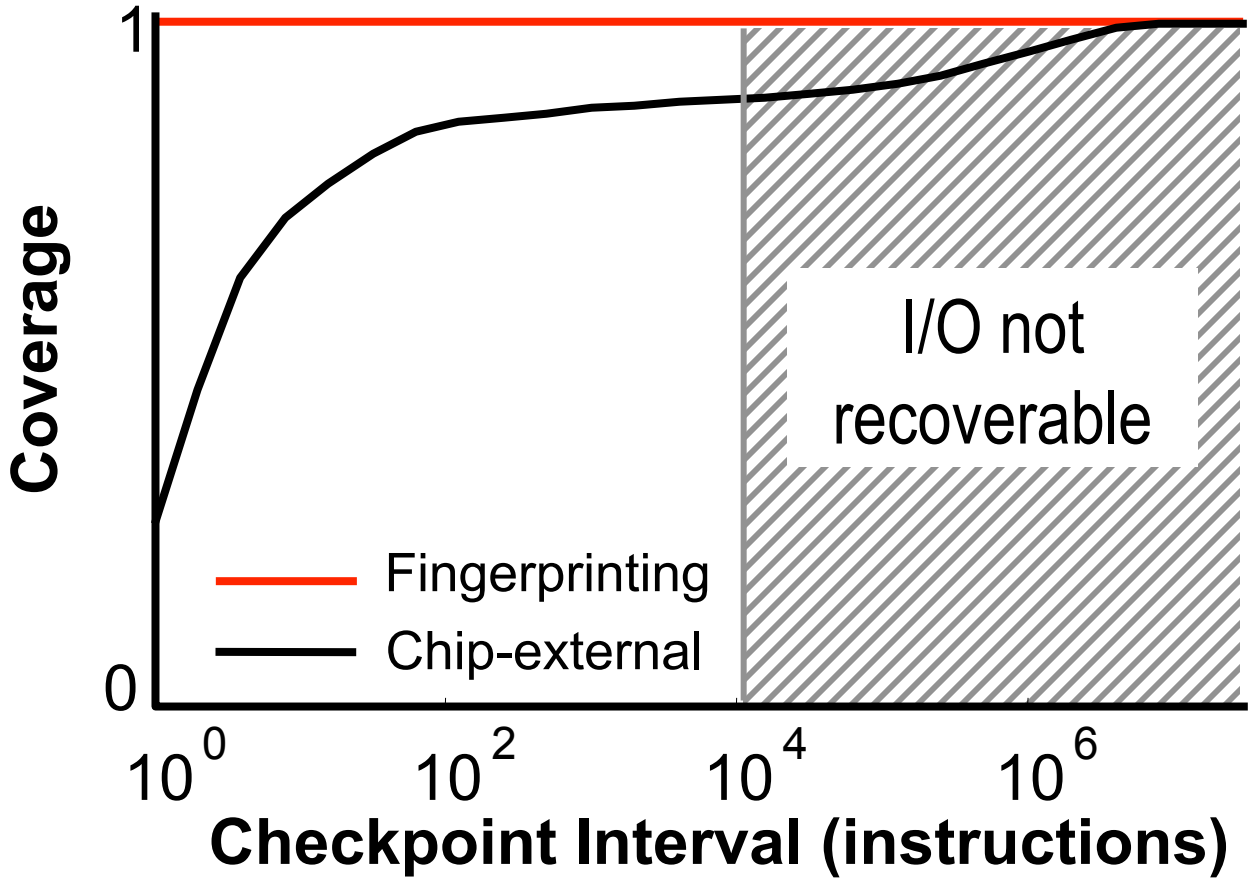
- Hash updates to architectural state
- Fingerprints compared across redundant nodes

✓ Bounded error detection latency

✓ Reduced comparison bandwidth



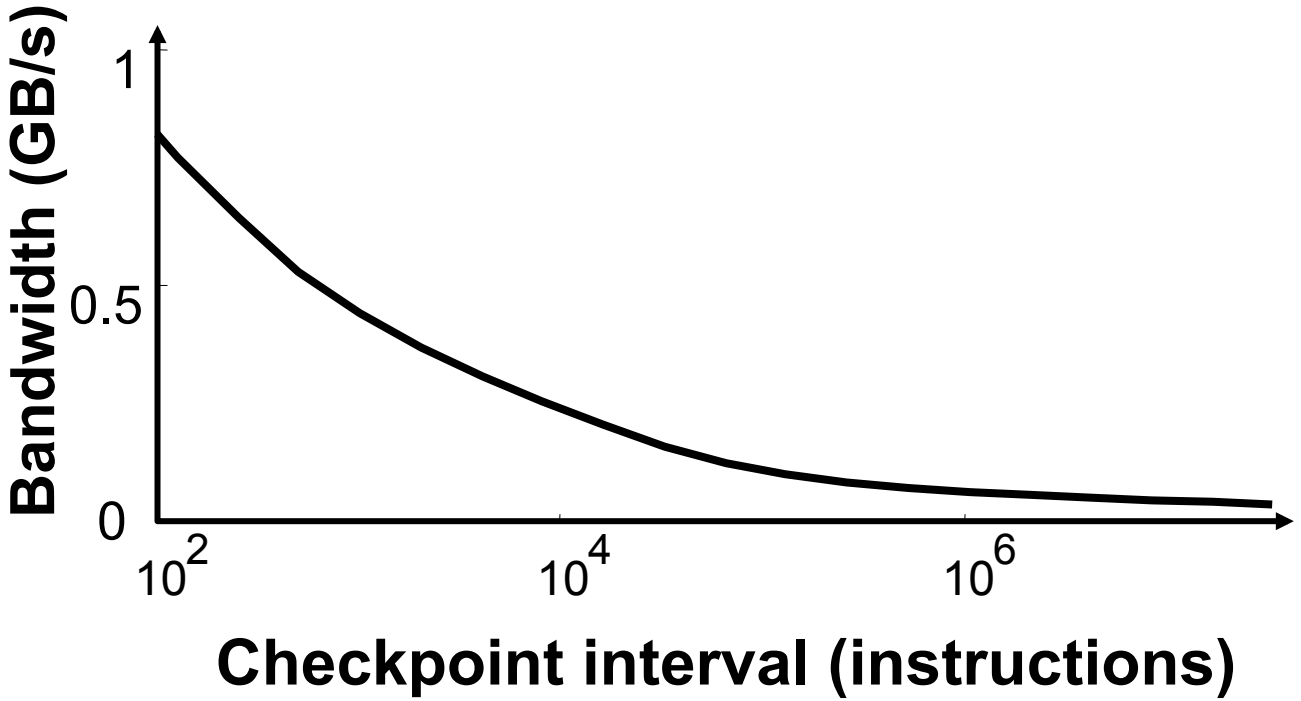
Error Detection: Coverage



- *16-bit (CRC) fingerprint → near perfect coverage*
- *Chip-external → acceptable coverage for >1M*

Error Detection: Bandwidth

- Differential comparison over interval



16-bit fingerprint < 150KB/s for 14K checkpoint intervals

Full-state band. unreasonable for small intervals

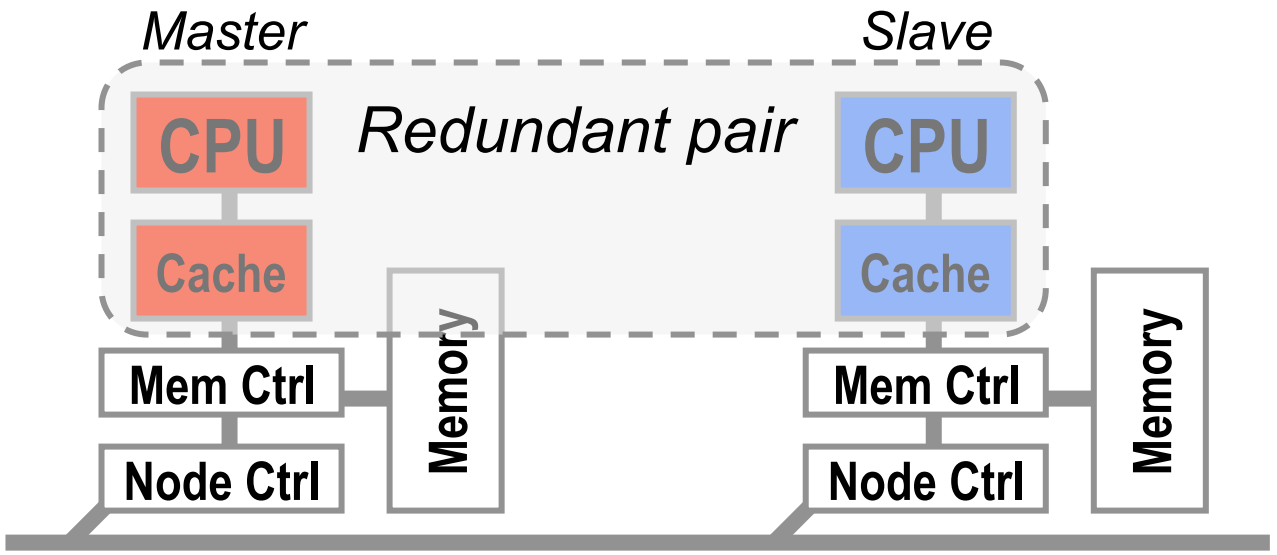
Coordination Across Nodes

Lockstep across system

- Logical lockstep of two physical processors (master / slave)

Two key challenges

- In-bound: Observe identical input data/timing
- Out-bound: Coordinate result (fingerprint) comparison



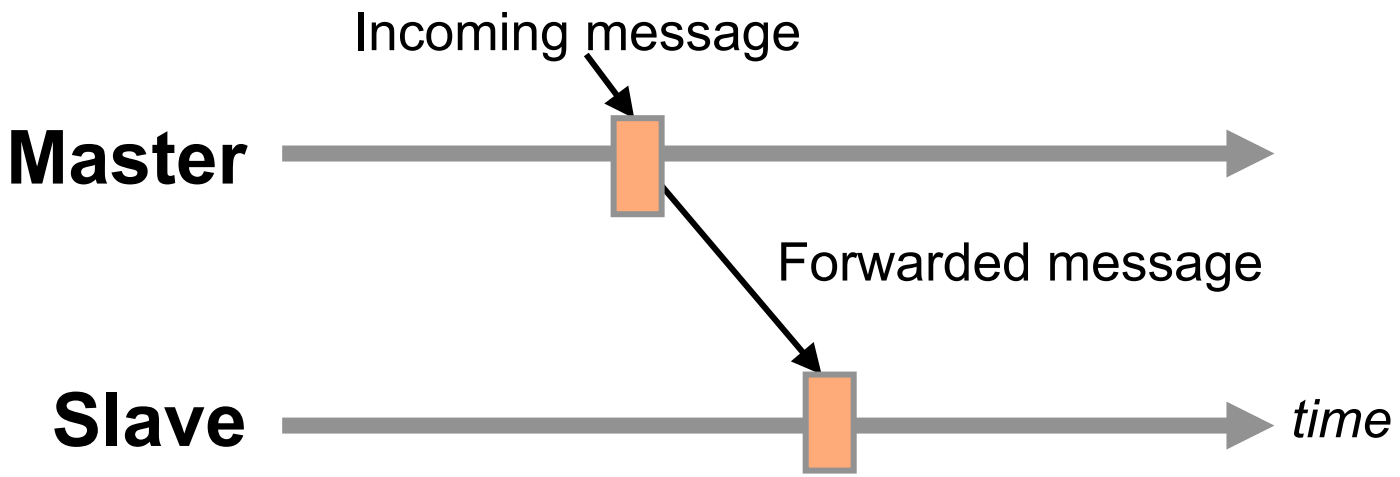
Coordination: Incoming Events

Need processor pair to observe identical inputs

- Cache refills, external interrupts, etc.

Master/slave arrangement

- Master receives all inputs and forwards to slave



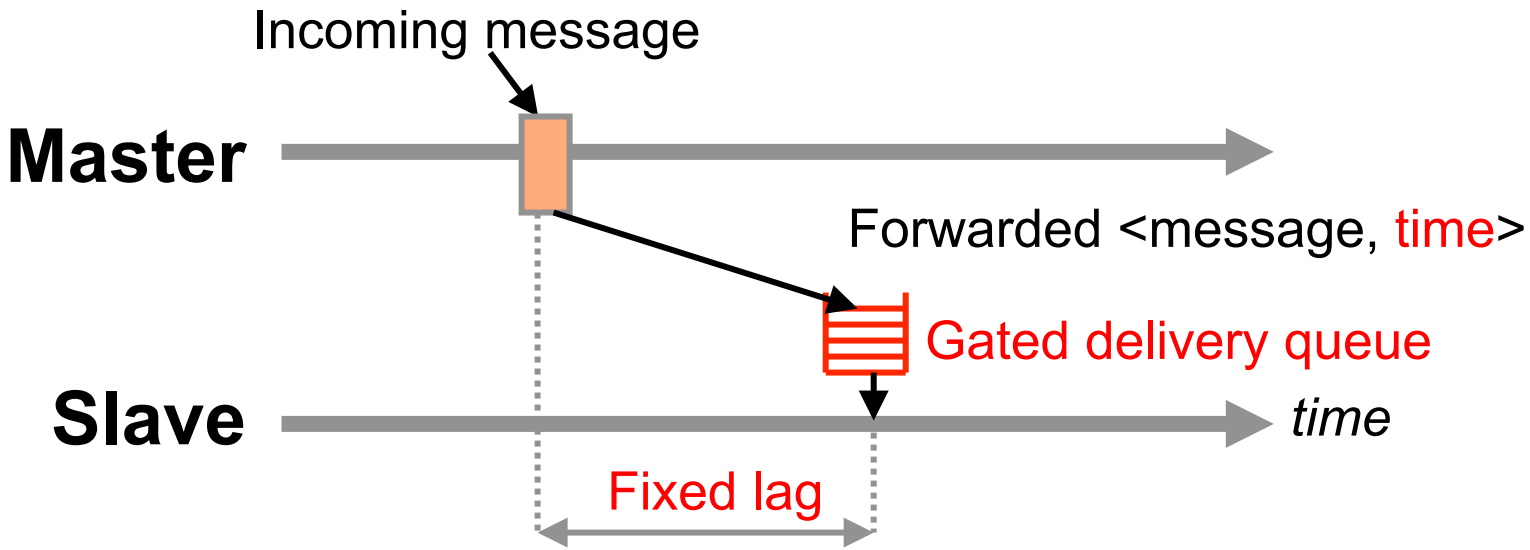
Replicated values, but also need replicated timing…

Coordination: Time Synch

Problem: pair separated by variable-latency link

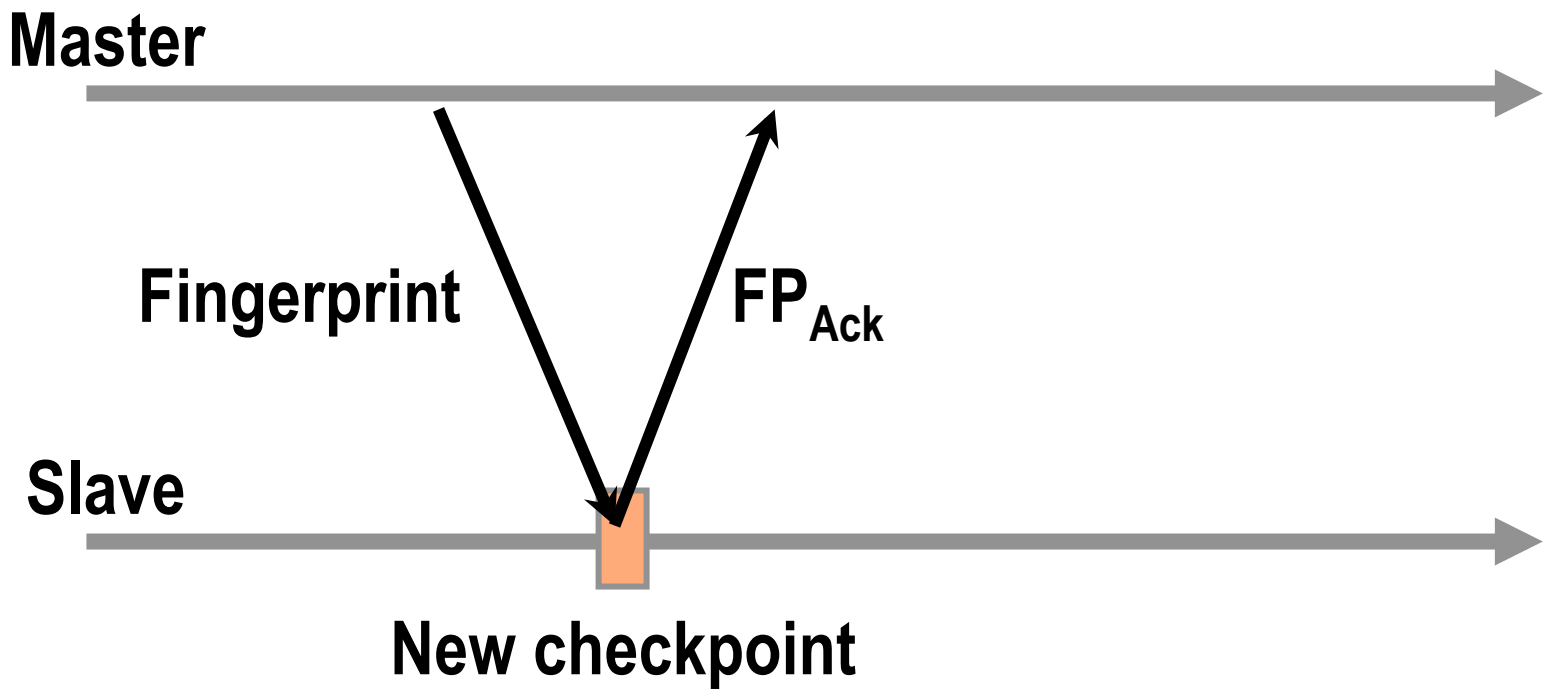
Key idea: use timestamp counter as *logical* clock

- Record message arrival time at master
- Buffer and replay at slave after fixed lag



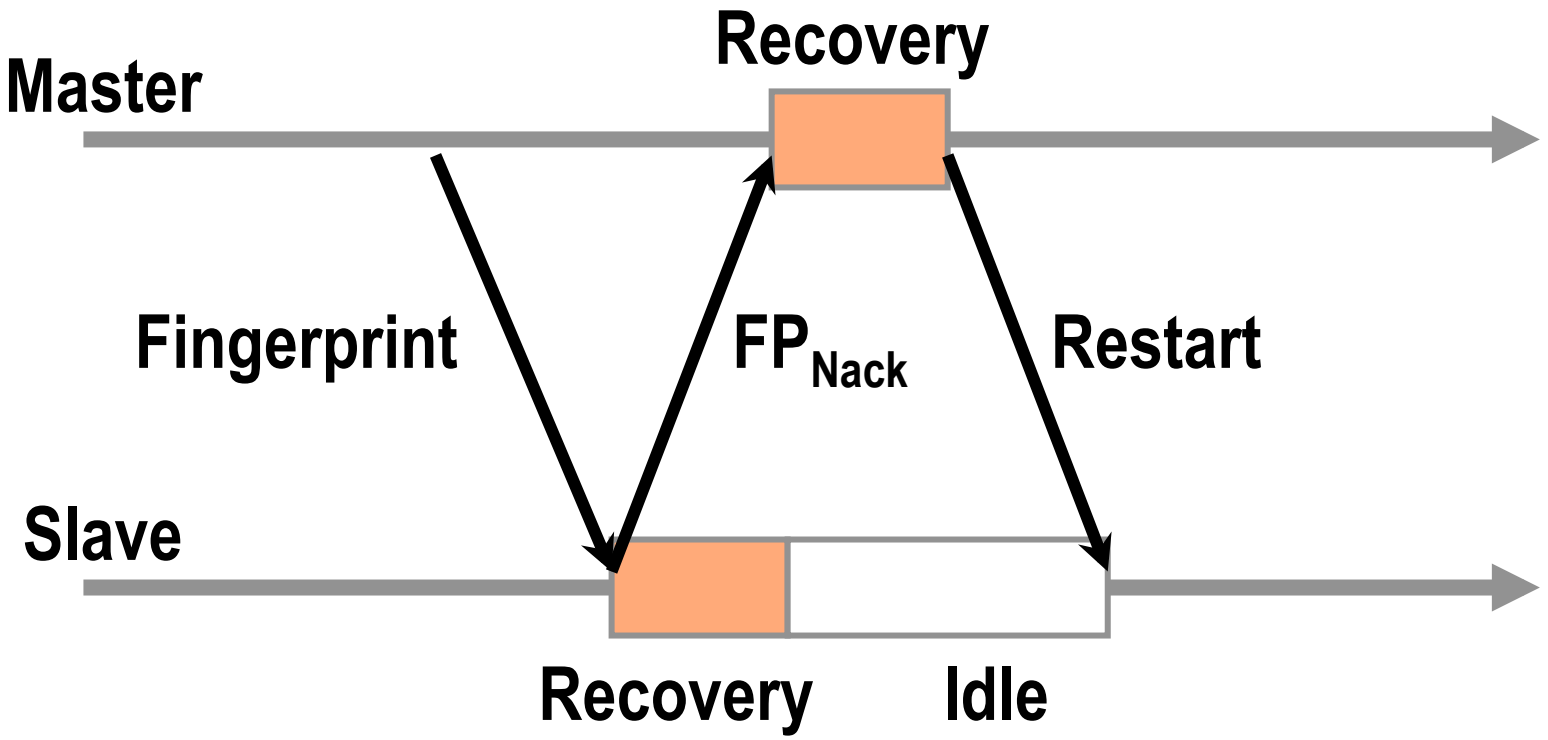
Obviates precise physical clock synchronization

Coordination: Detection



Slave keeps checkpoint: on-chip record of changed register/memory state

Coordination: Recovery



Rollback-recovery to last checkpoint upon detection

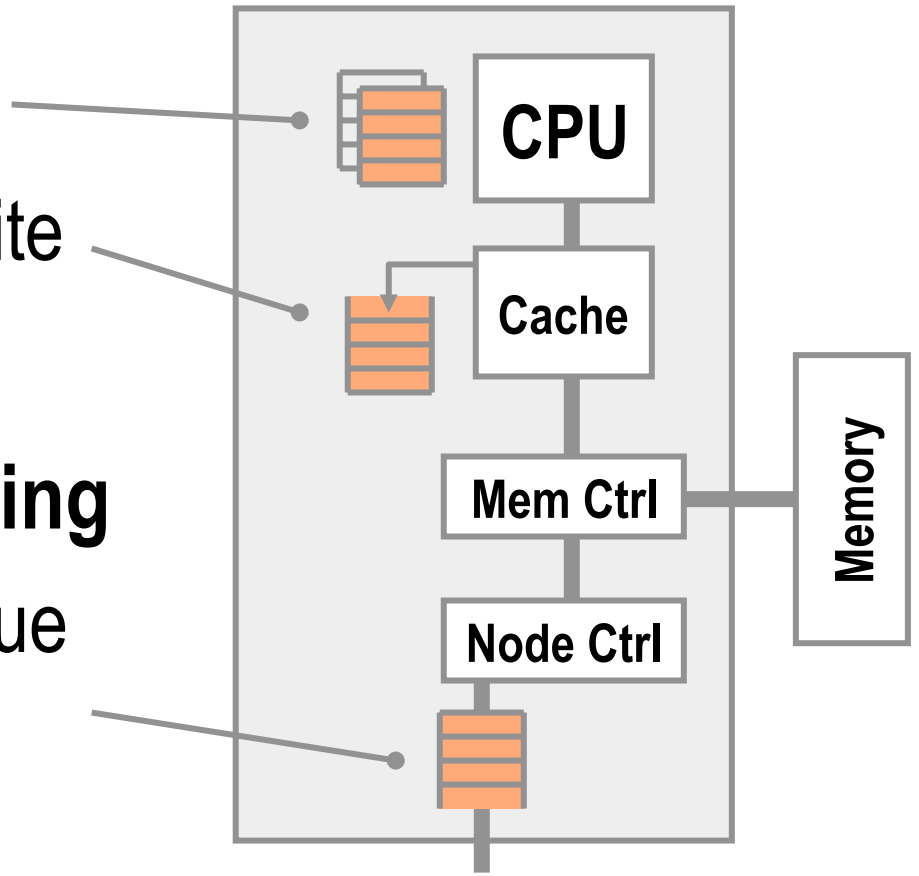
Coordination: Hardware

Lightweight logging

- Register file: copy
- Cache: copy-on-write
  - 256-entry FIFO

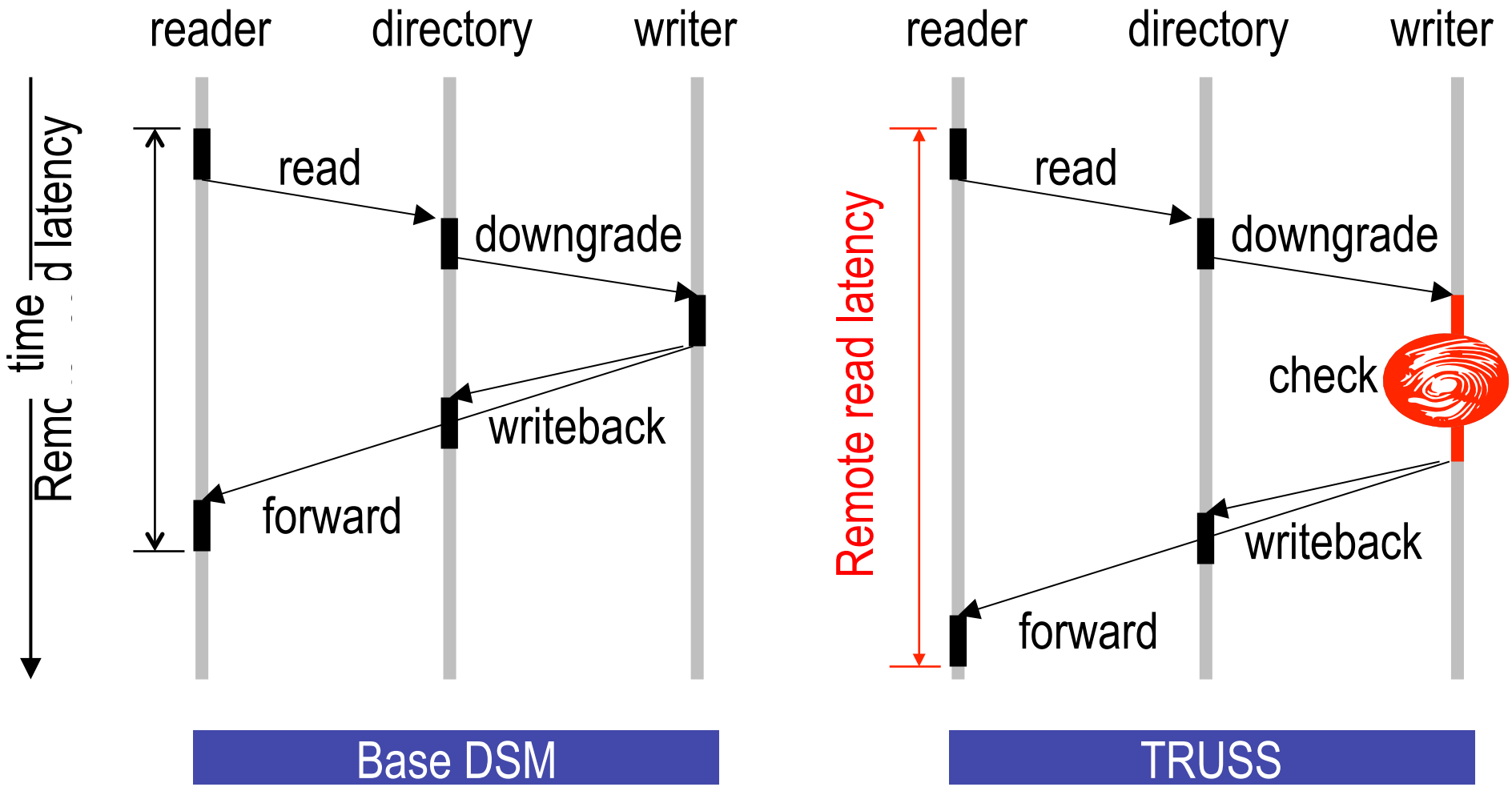
Master/Slave messaging

- Gated delivery queue
  - 16-entry FIFO



Coordination and recovery with little extra hardware

Coordination: Performance Impact



Longer read time when data "dirty" in cache

Coordination: Validation Filter

Observation: most dirty data known correct

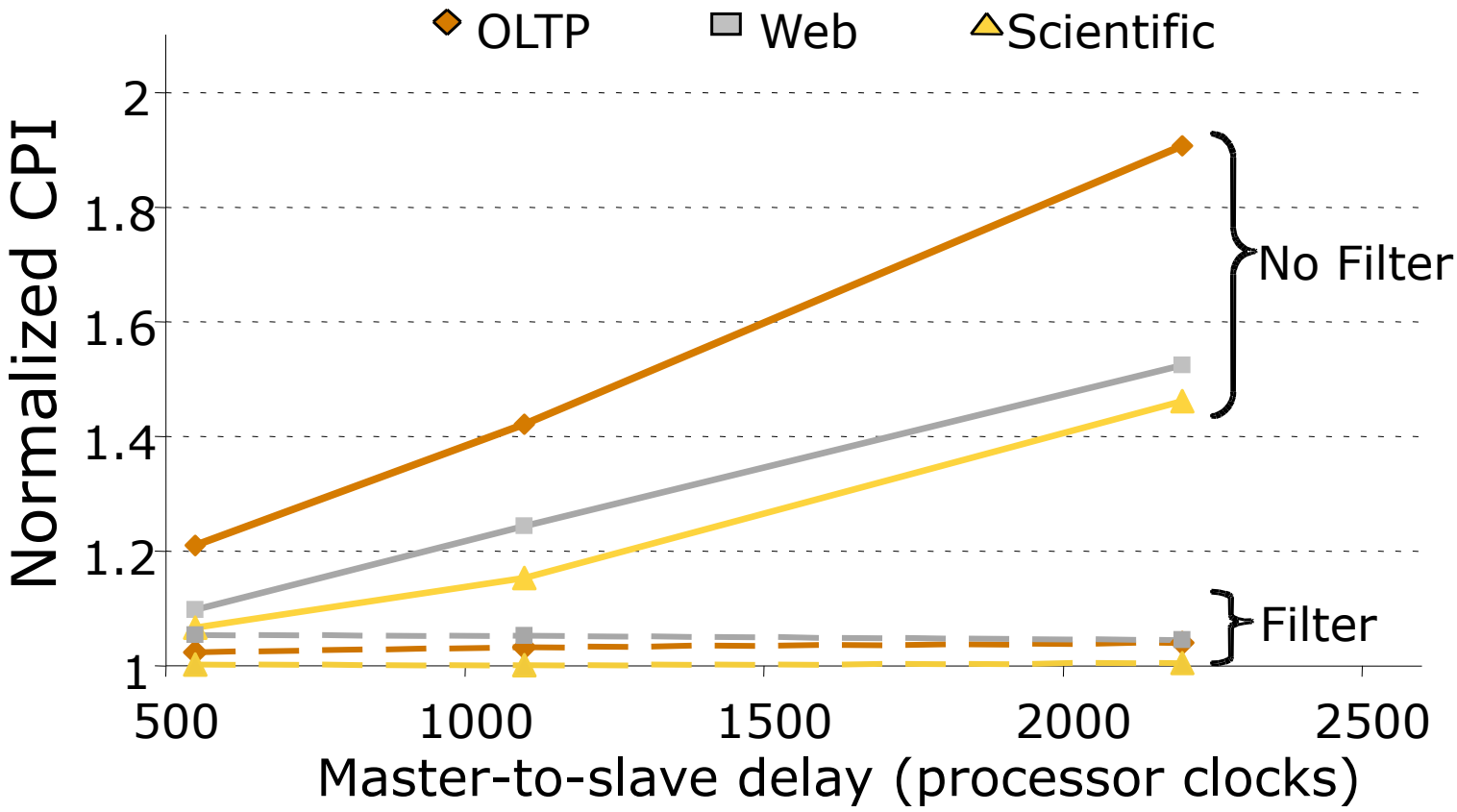
- Results were in previous fingerprint comparison
- Similar to RegionScout, Jetty, etc.

Solution: validation filter

- Tracks recent stores not yet fingerprinted
- Searched on snoop
  - Match? Explicitly check with slave (delay)
  - No match? Forward data directly (no delay)

Small filter (64 entries) sufficient

Performance Sensitivity to Lag

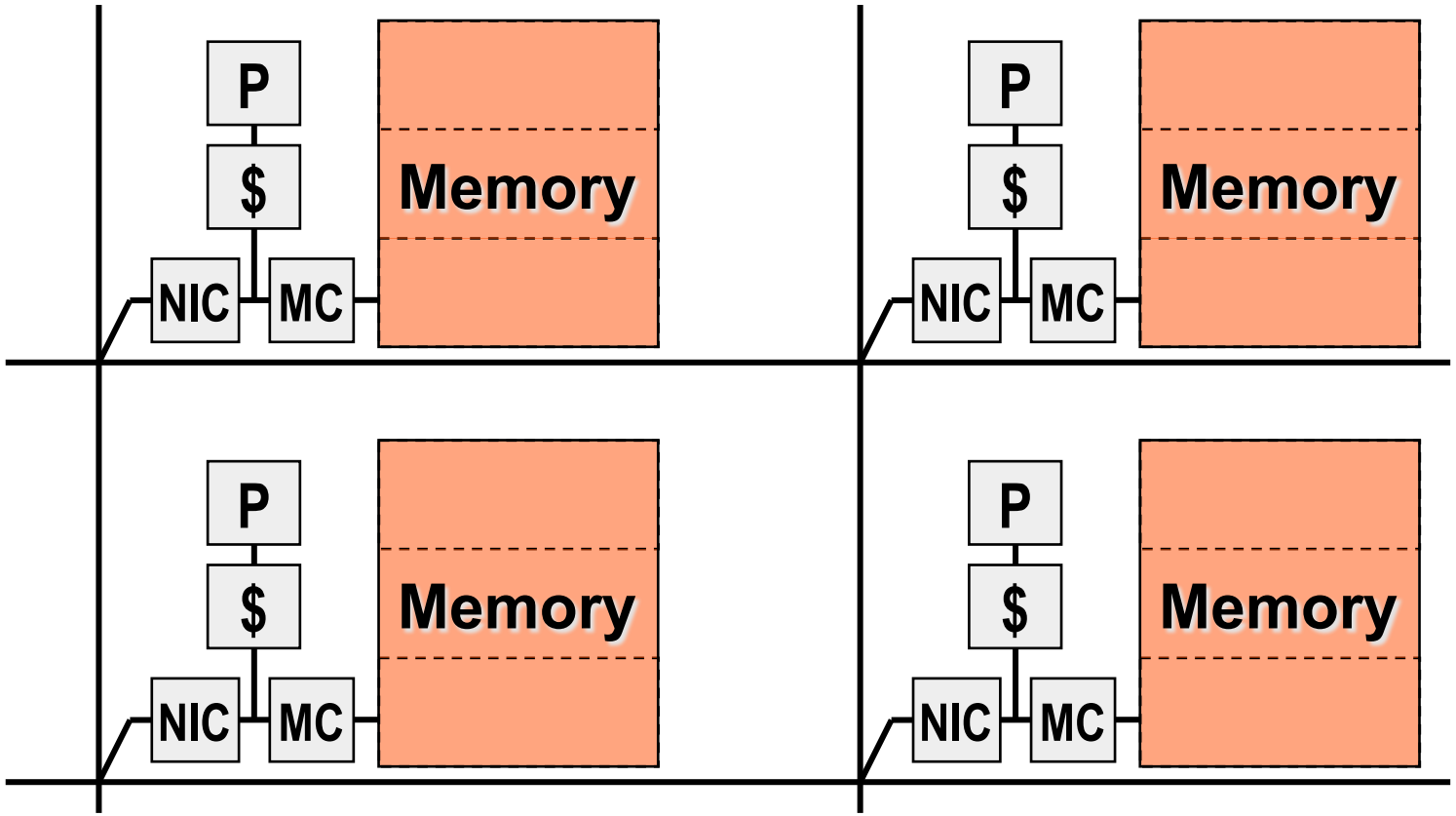


- ***Large performance impact as lag increases***
- ***Filter removes sensitivity, permits scaling***

Outline

- Overview
- Computation redundancy
  - Fingerprinting
  - Distributed redundancy
- **Memory protection**
  - Distributed parity
- **Summary**

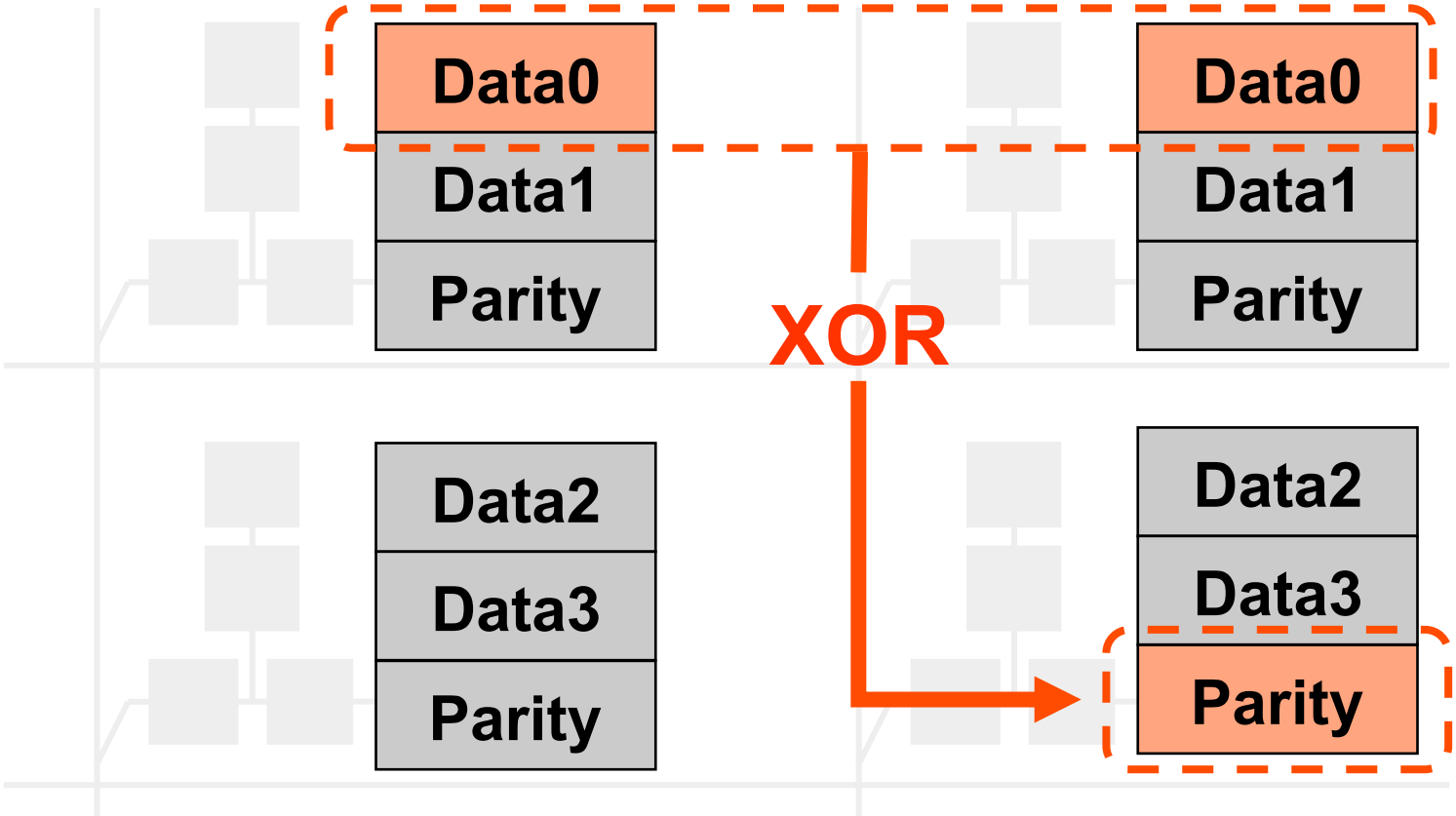
(2) DRAM in Scalable Servers



- Physical address space distributed across nodes

Lose one node ➔ Memory lost on all nodes

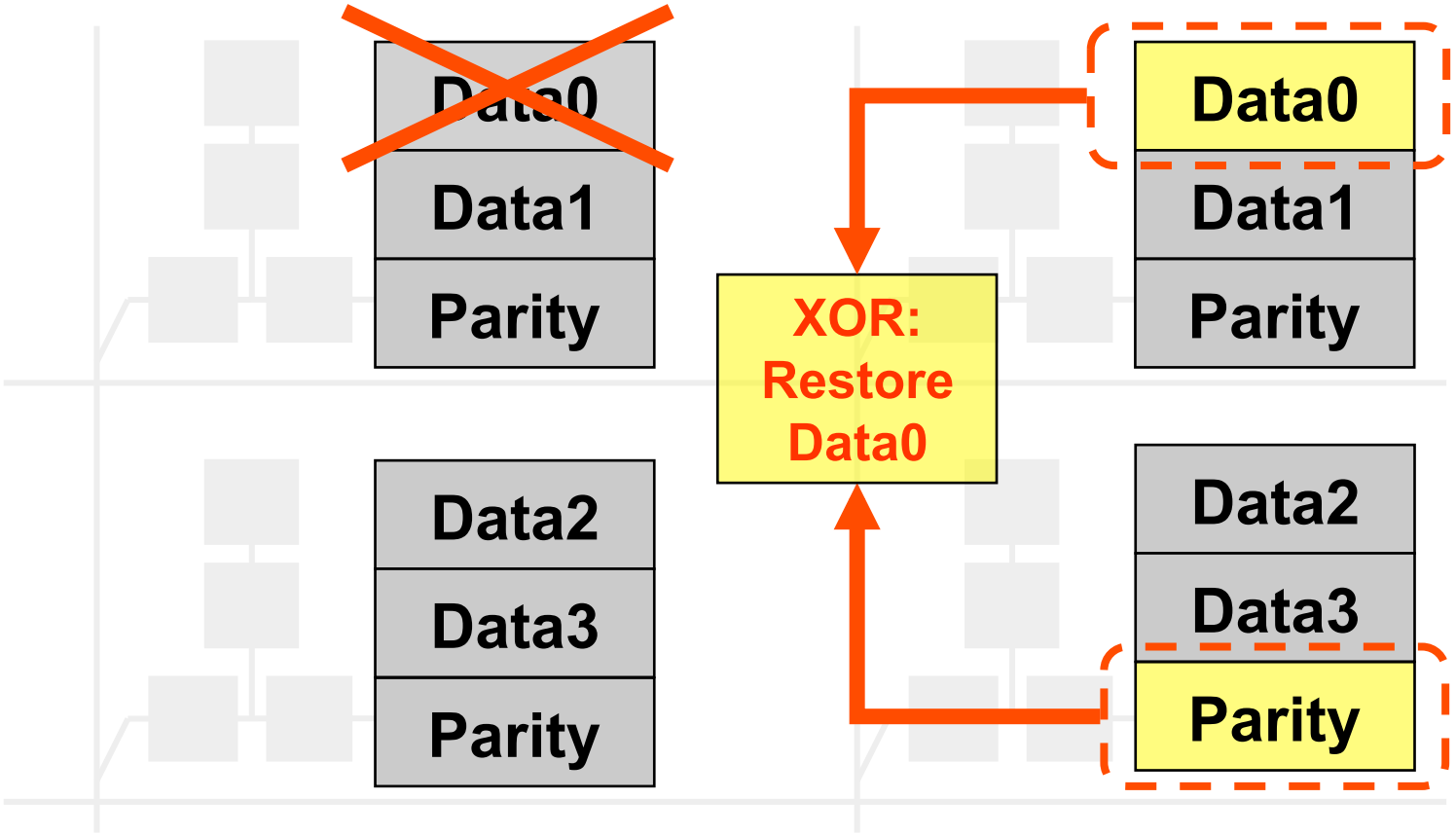
DRUM: Distributed Memory Parity



- Parity for group updated on writebacks
- Use ECC to detect soft error, distributed parity to correct

Parity distributed across memory nodes

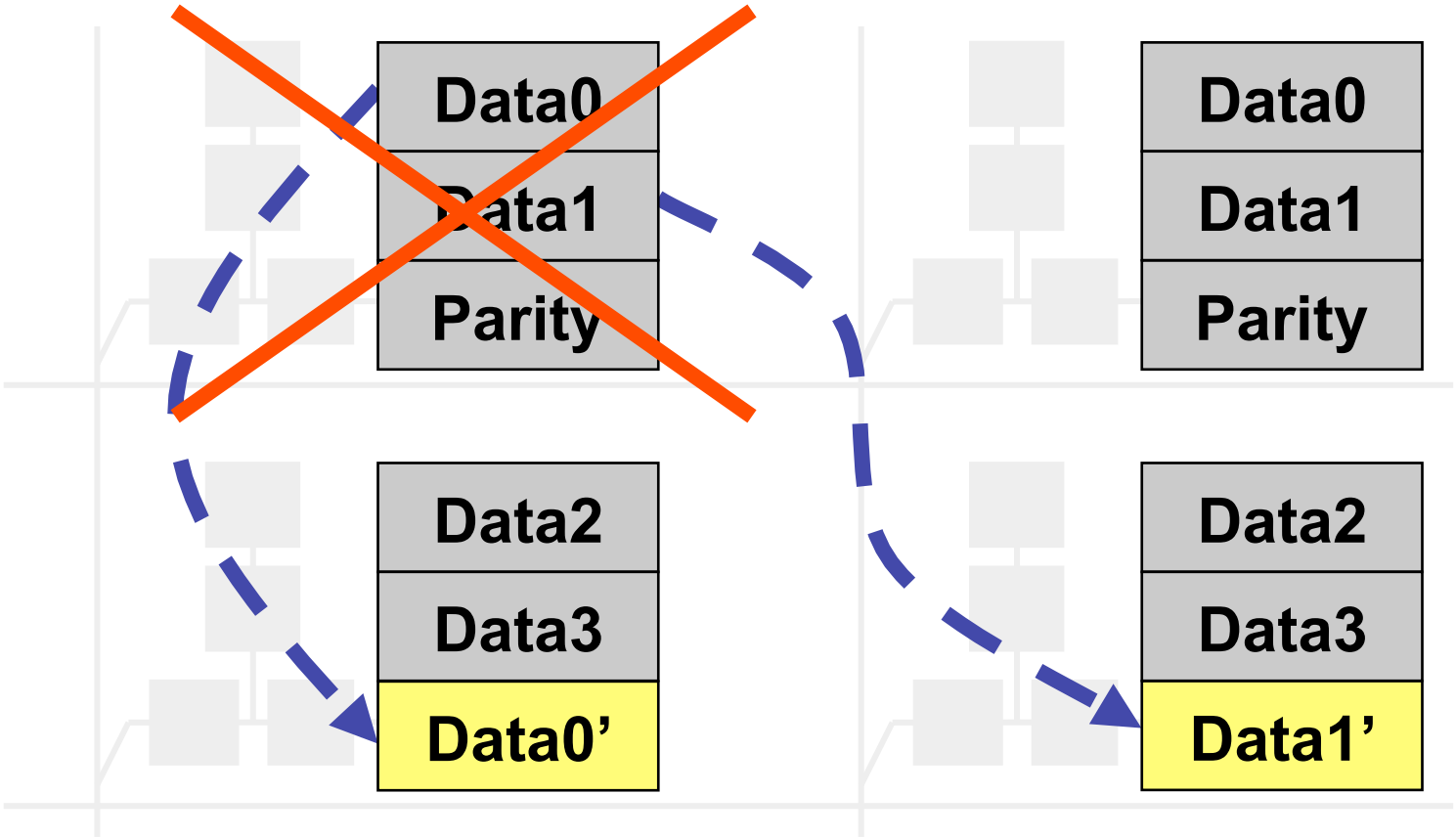
Degraded Mode Recovery



- Reconstruct lost data using remaining data+parity on demand
- Performance overhead grows w/parity group size

Restores data from soft errors and node failures

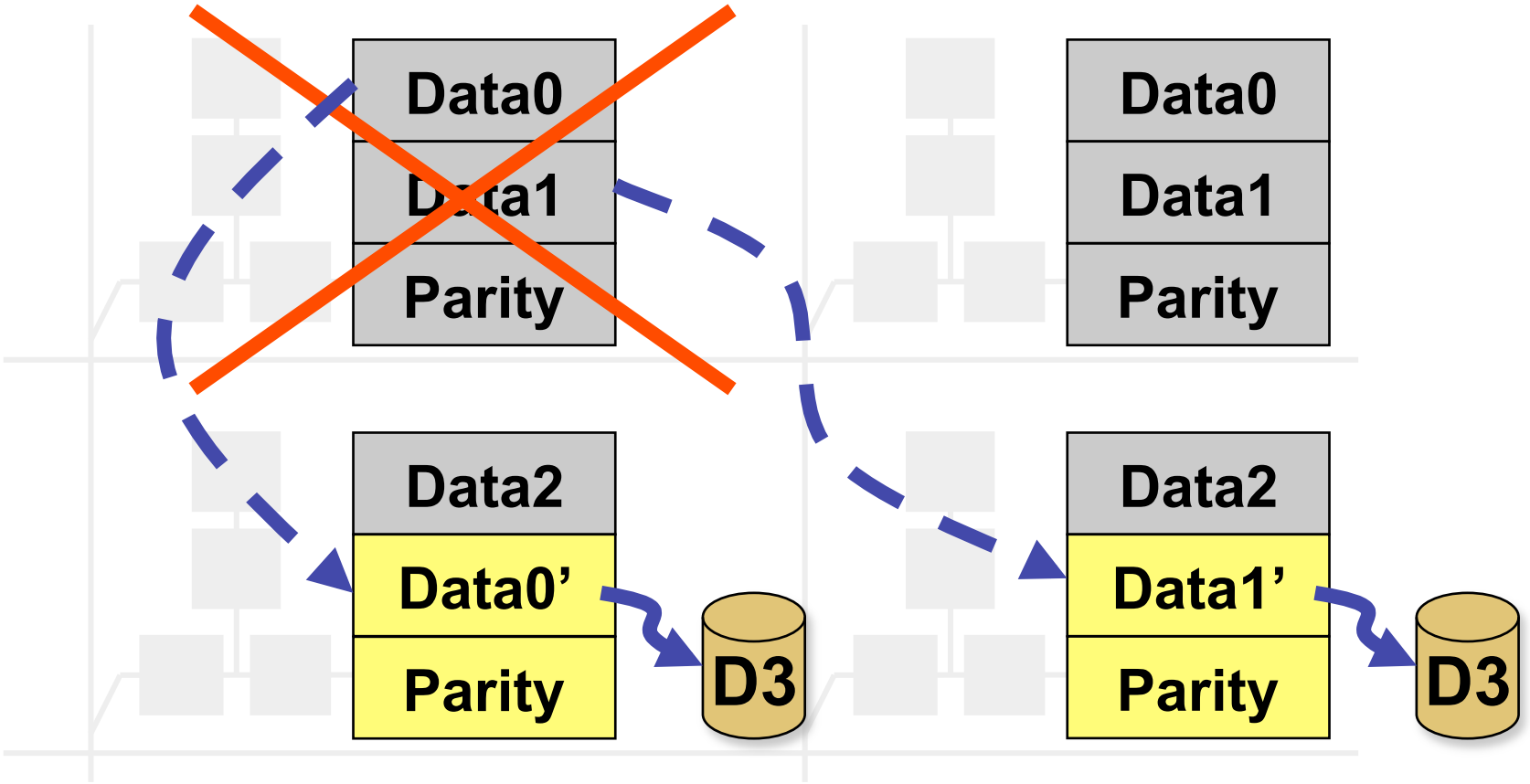
Node Failure: Parity Swap



- Reconstruct data to parity group locations
- Transparently remap Phys. Addrs. to reconstructed data

Fast execution after recovery, but no further error protection

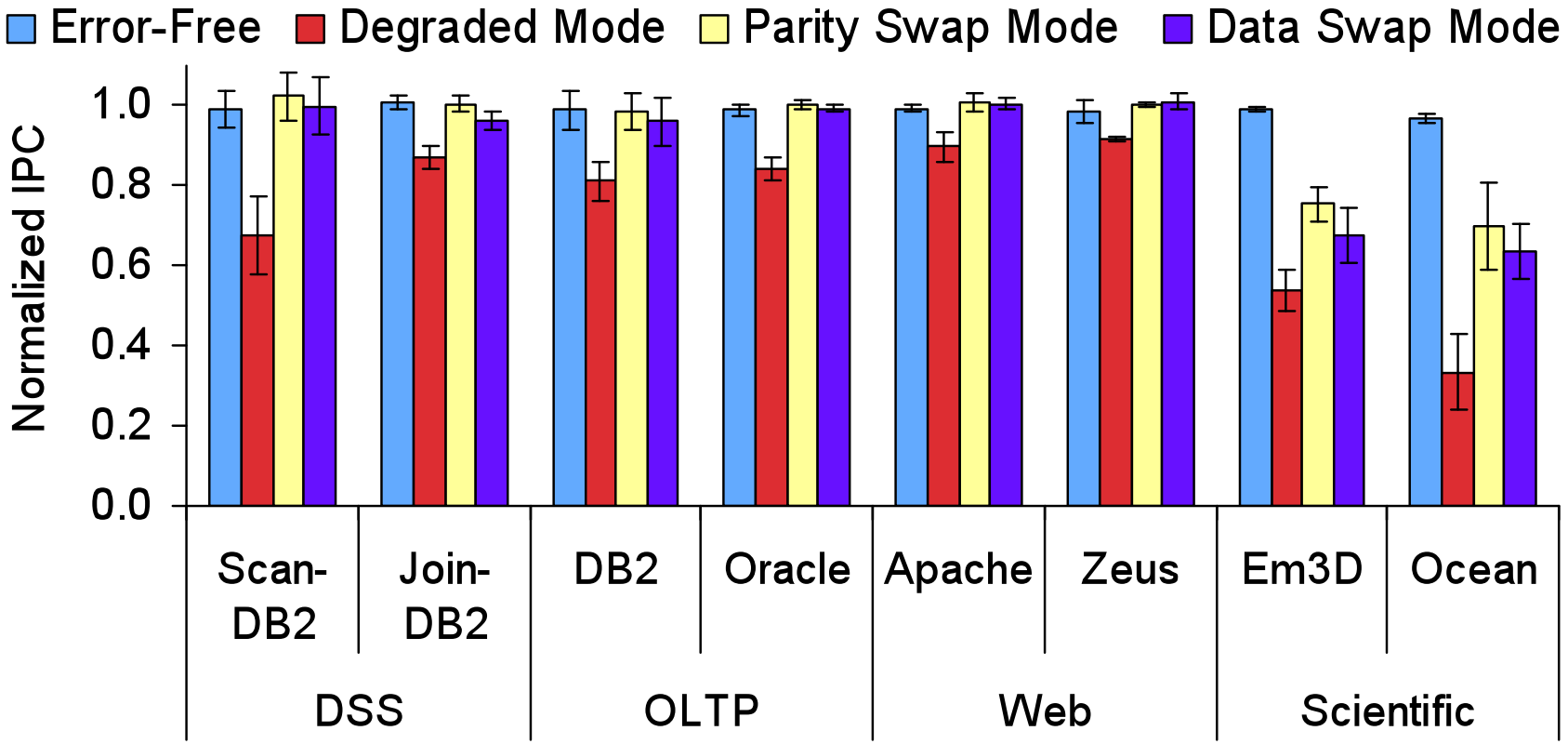
Node Failure: Data Swap



- OS swaps group to disk
- Reconstruct lost data to the freed group

Reduced address space, but maintains parity protection for all data

DRUM Performance



- ***Error-free: Low-overhead parity update for error-free & data swap***
- ***Lost node: Parity and data swap eliminate degraded costs***

DRUM Summary

Distributed memory parity in a DSM

- Physical addresses transparently remapped
- Recovery modes trade performance, complexity, and protection
- Minimal error-free performance impact

Protection for memory node failures

- Preserves data over any single node failure
- Subsumes other techniques for hard/soft memory errors

Related Work

Processor

- Redundant threading (DIVA, SRT[R]/CRT[R], Ditto, SHREC, ….)
- Fail-stop detection/global checkpointing (ReVive)

→ No solution for both permanent and soft error

Memory

- IBM ChipKill, Compaq MemoryRAID, DRAM Mirroring

  → No cross node solution
- ReVive's distributed parity

  → Directory locking overhead upon parity update

No complete solution for soft and hard errors

Summary

TRUSS goals:

- Reliable, scalable server architecture
- Software transparency

Enablers:

- Lightweight detection and recovery
- Fault barrier and coord./comm. Protocols
- Distributed memory parity

Sponsors:

- Intel, NSF, MARCO/C2S2, CyLab

TRUSS

Computer Architecture Lab (CALCM)

Carnegie Mellon University

http://www.ece.cmu.edu/~truss