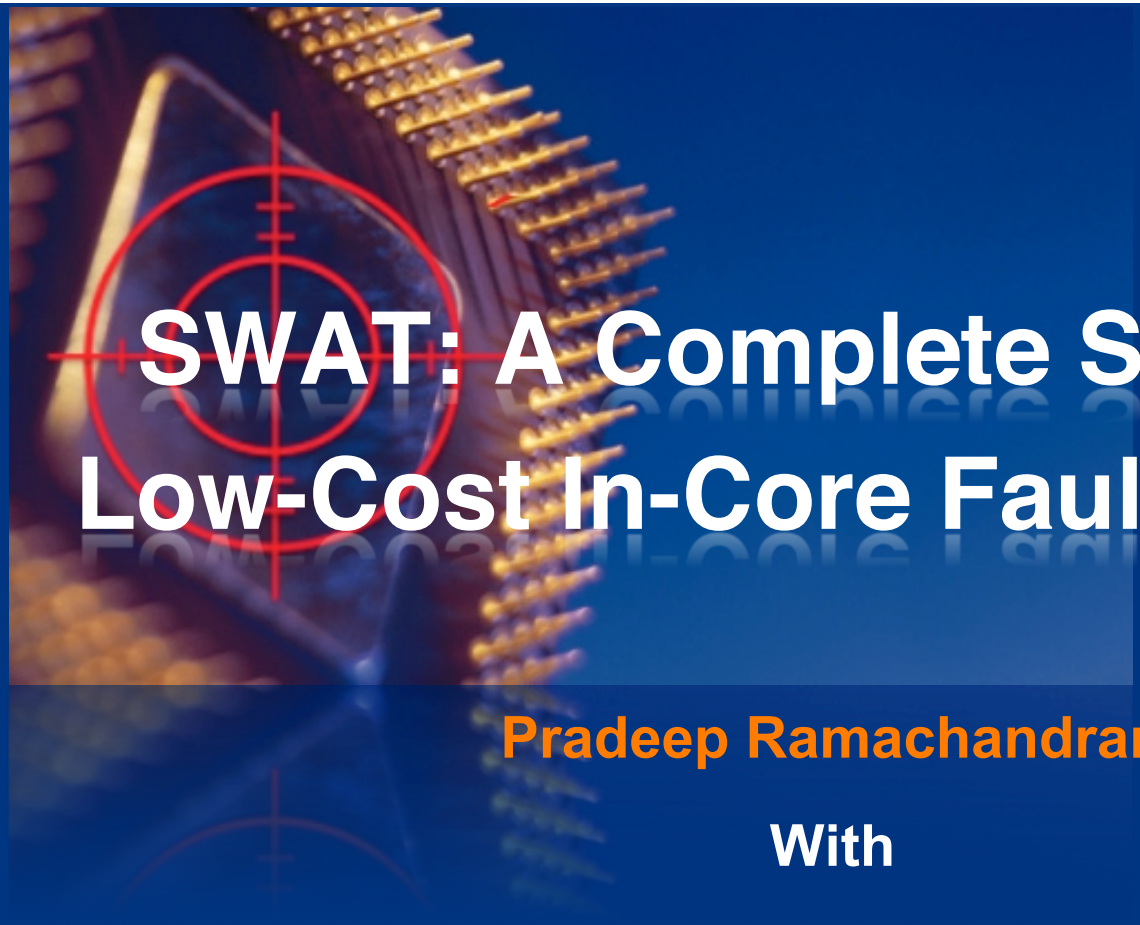


SWAT: A Complete Solution for Low-Cost In-Core Fault Resiliency

Pradeep Ramachandran

With

Manlap Li, Siva Hari, Lei chen, Byn Choi, Swarup Sahoo, Rob Smolinski,

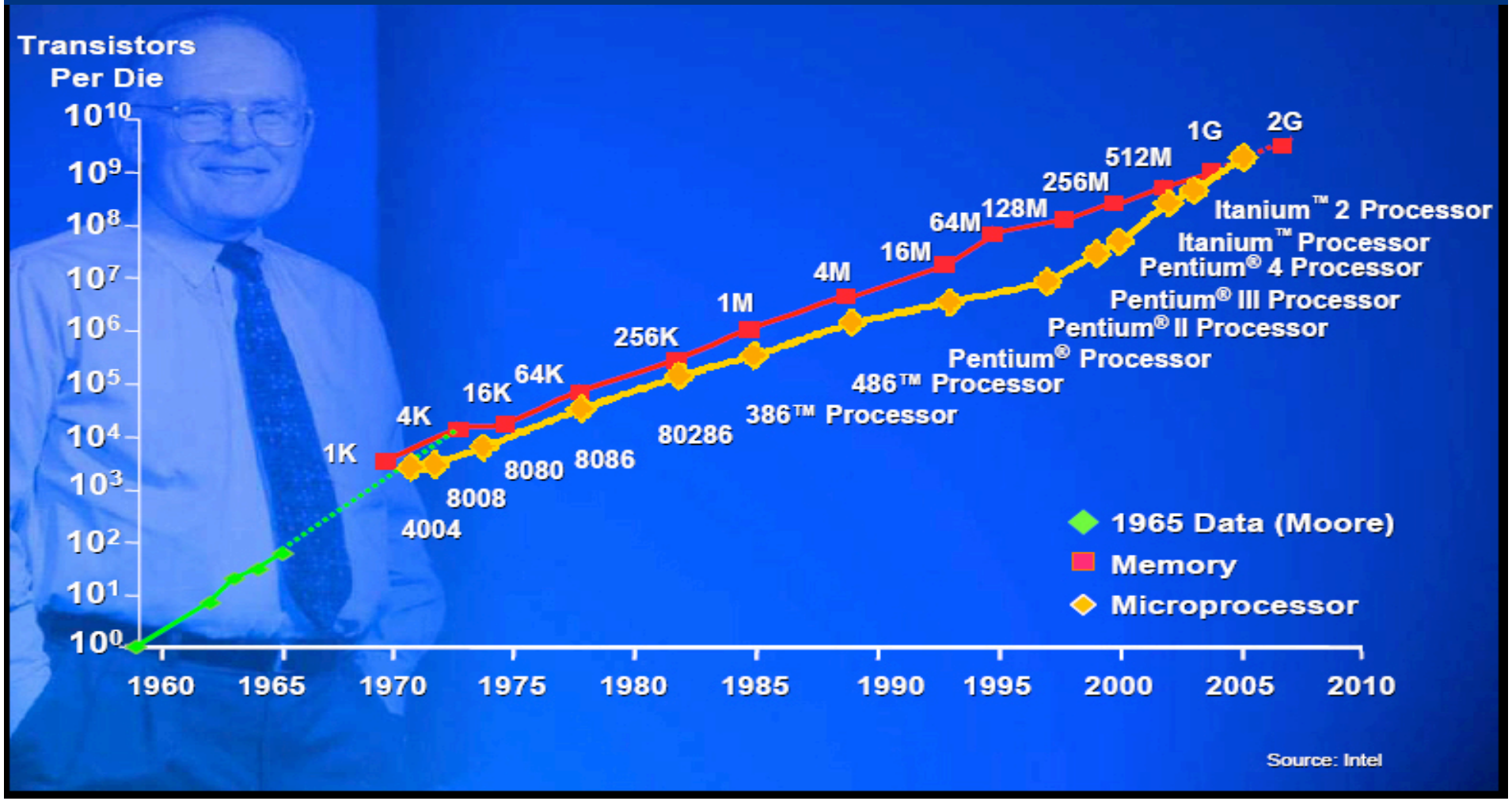
Sarita Adve, Vikram Adve, Yuanyuan Zhou

Department of Computer Science

University of Illinois at Urbana Champaign

swat@cs.uiuc.edu

The Glorious Age of Moore's Law



- **2X transistors ⇒ 2X performance every 18 months**
  - **But every coin has two sides!**

Reliability – The Dark Side of Moore's Law

- More transistors with smaller feature sizes ⇒ more failures
  - Physical limitations in manufacturing, testing, etc.
- Transistor failures cause losses of billions on dollars

"As technology scales further, new challenges will emerge, … These problems will inevitably lead to **inherent unreliability in components,** posing serious design and test challenges."

Shekhar Borkar, Intel fellow

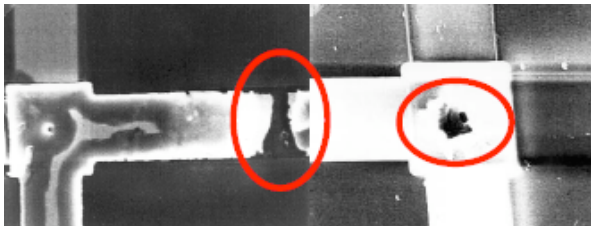
"Statistically, the large number of components will lead to **reliability, aging, and defect limitations** that could no longer be eliminated through margins or overdesign."

Ajith Amerasekera, TI fellow

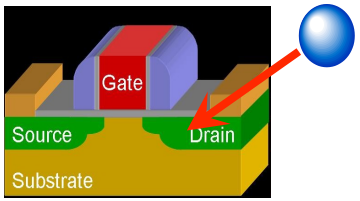
- Need to guarantee reliable operations from unreliable components
  - Identified by ITRS as a grand challenge in the late CMOS era

Handling Failures In-the-Field

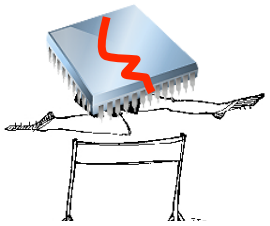
- **Hardware will fail in-the-field due to several reasons**



Wear-out
(Devices are weaker)



Transient errors
(High-energy particles)



Design Bugs

… and so on

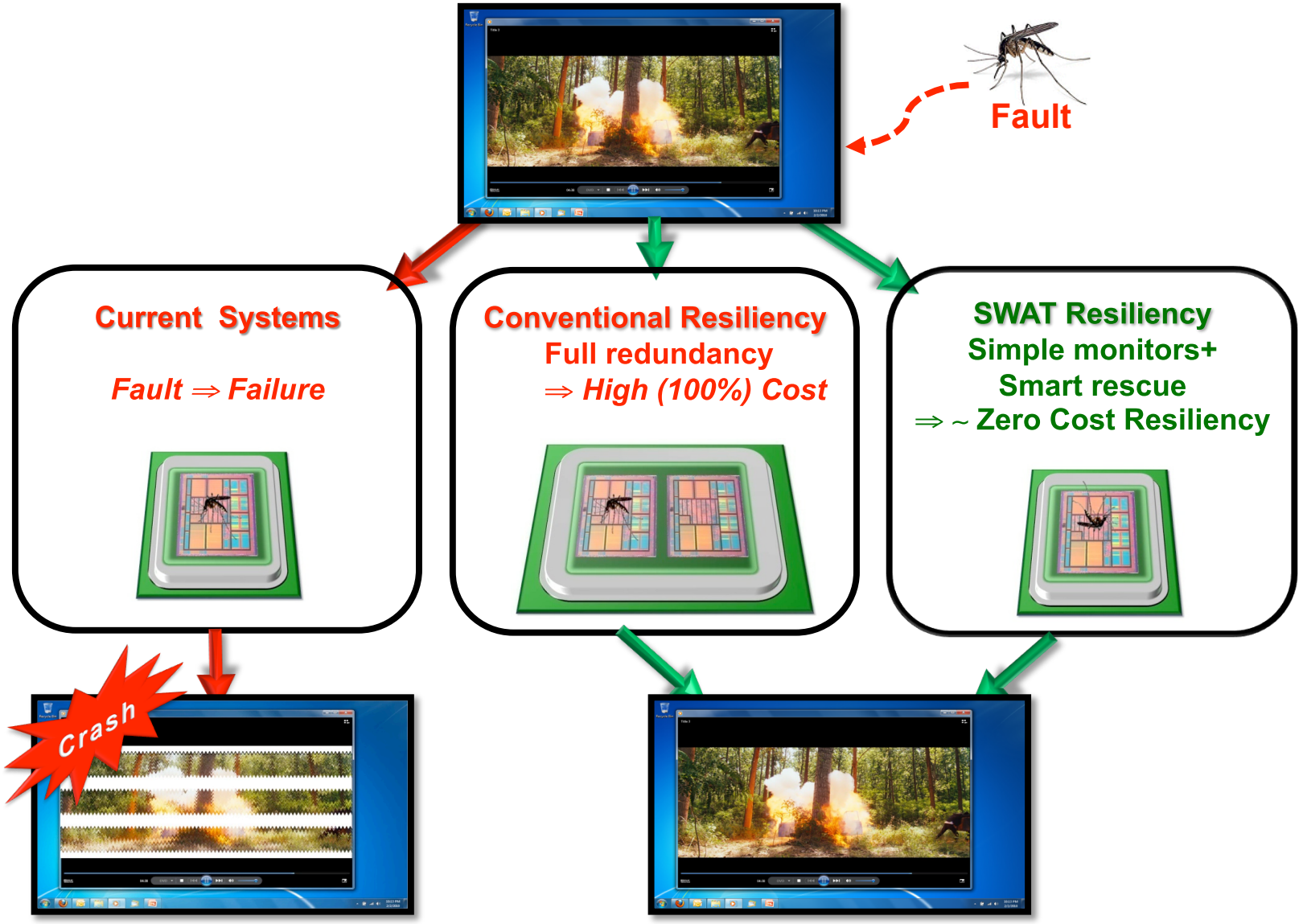
⇒**Need in-field detection, diagnosis, recovery, repair**

- **Reliability problem pervasive across many markets**
  - **Traditional redundancy solutions (e.g., nMR) too expensive**

⇒ **Need low-cost solutions for multiple failure sources**

∗ **Must incur low area, performance, power overhead**

SWAT: A Low-Cost Reliability Solution



Observations

- Need handle only hardware faults that propagate to software
- Fault-free case remains common, must be optimized

⇒ SWAT: SoftWare Anomaly Treatment

⇒ Detect software anomalies, HW support for recovery

– Zero to low overhead “always-on” monitors

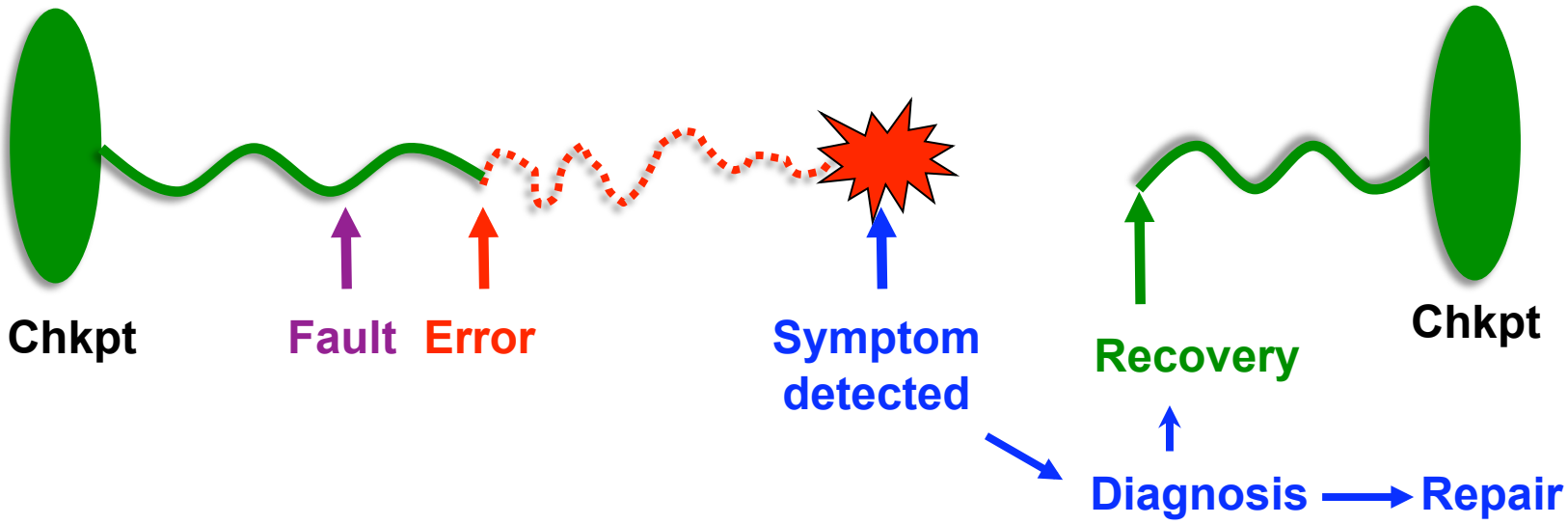
Diagnose cause after symptom detected

– May incur high overhead, but rarely invoked

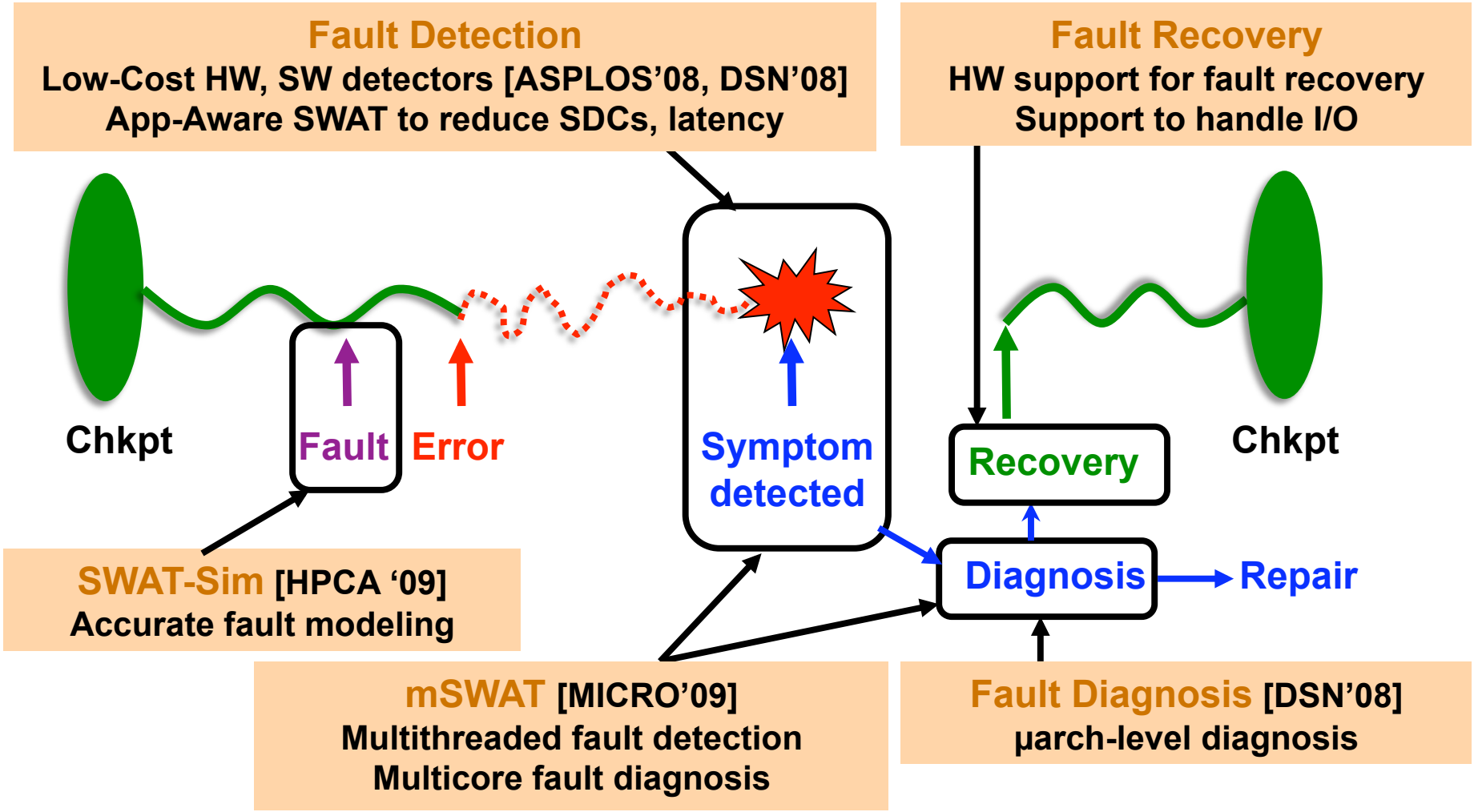
Advantages of SWAT

- **Handles faults that matter,** ignore fault-model, masked faults
- **Low, amortized overheads** by optimizing for common case
  - Potential to exploit SW reliability solutions
- **Customizable and flexible** to system needs
- **Holistic systems view enables novel solutions**
  - Synergistic detection, diagnosis, recovery solutions
- **Beyond hardware reliability**
  - Potential application to post-silicon test and debug

SWAT Operations

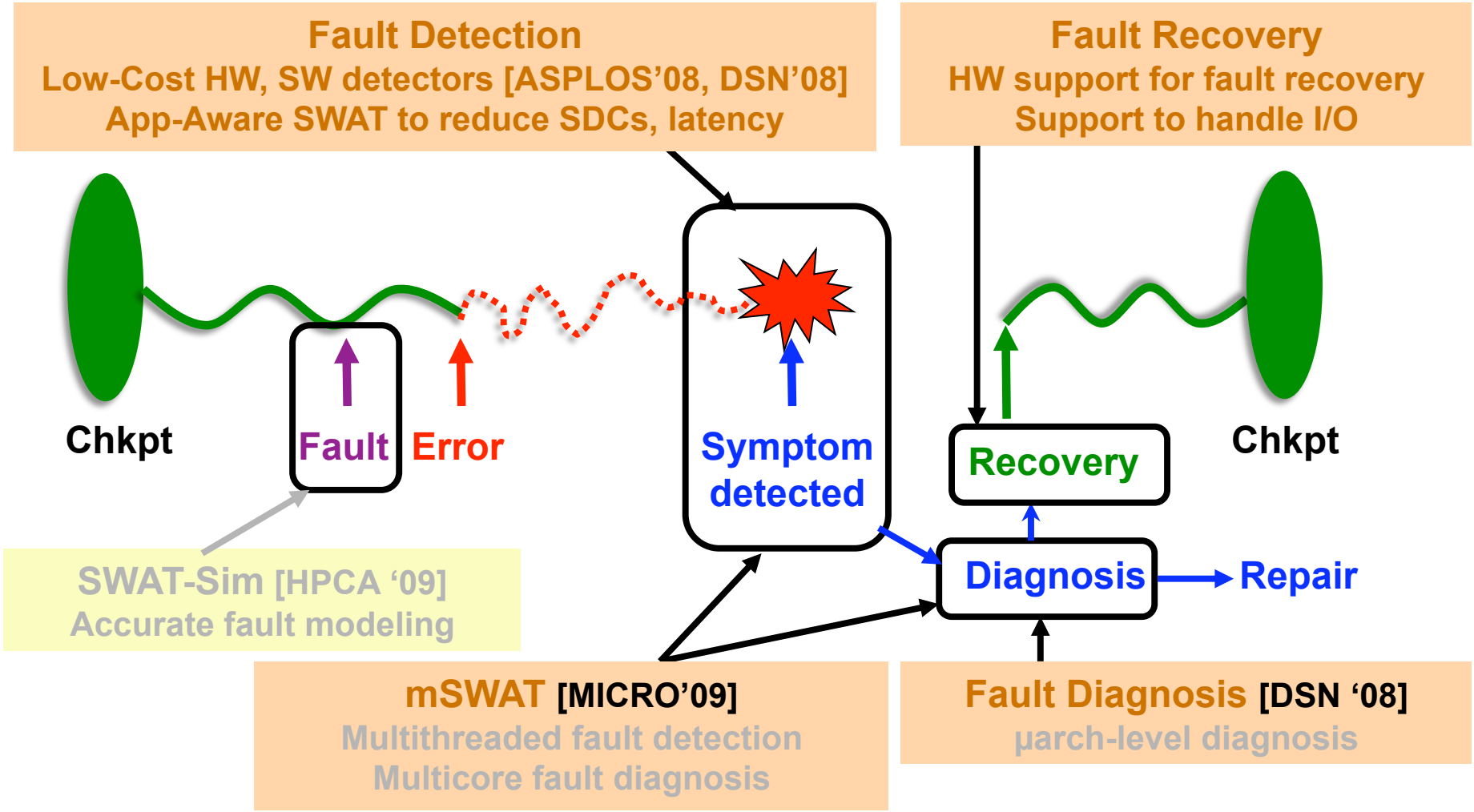


SWAT Contributions



- Overall, SWAT is a complete solution for In-core HW faults
  - Demonstrated on a wide spectrum of workloads

This Talk

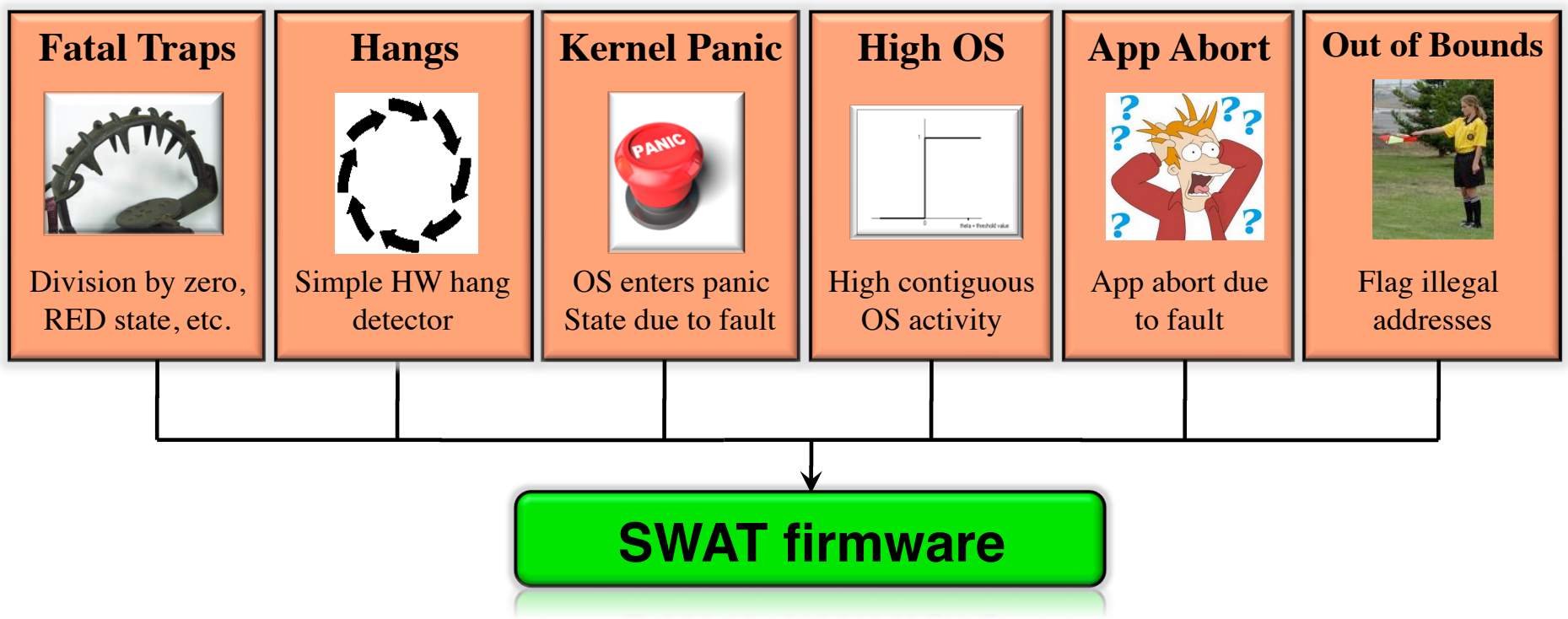

Fault Detection
Low-Cost HW, SW detectors [ASPLOS'08, DSN'08]
App-Aware SWAT to reduce SDCs, latency
Fault Recovery
HW support for fault recovery
Support to handle I/O
Chkpt
Fault
Error
Symptom detected
Recovery
Chkpt
Diagnosis
Repair
SWAT-Sim [HPCA '09]
Accurate fault modeling
mSWAT [MICRO'09]
Multithreaded fault detection
Multicore fault diagnosis
Fault Diagnosis [DSN '08]
µarch-level diagnosis

Outline

- Introduction to SWAT
- **Fault Detection**
- **Fault Recovery**
- **Fault Diagnosis**
- **Conclusions and Future Work**

SWAT Fault Detection

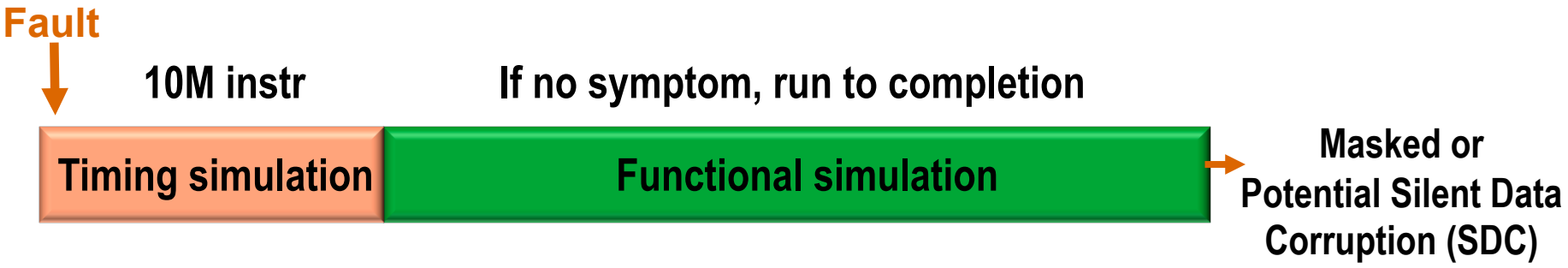
- **Simple detectors that observe anomalous SW behavior**



- **Incur very low hardware area ⇒ low cost detectors**
- **Incur near-zero perf overhead in fault-free operation**

Evaluating SWAT Detectors

- Full-system simulation with modern out-of-order processor
  - Simics functional + GEMS timing simulator
- Apps: Mix of multimedia, I/O intensive and compute intensive
  - Faults injected at different points in app execution
- µarch-level fault injections (single fault model)
  - Stuck-at, transient faults in latches of 8 µarch units
  - ~48,000 total faults ⇒ statistically significant

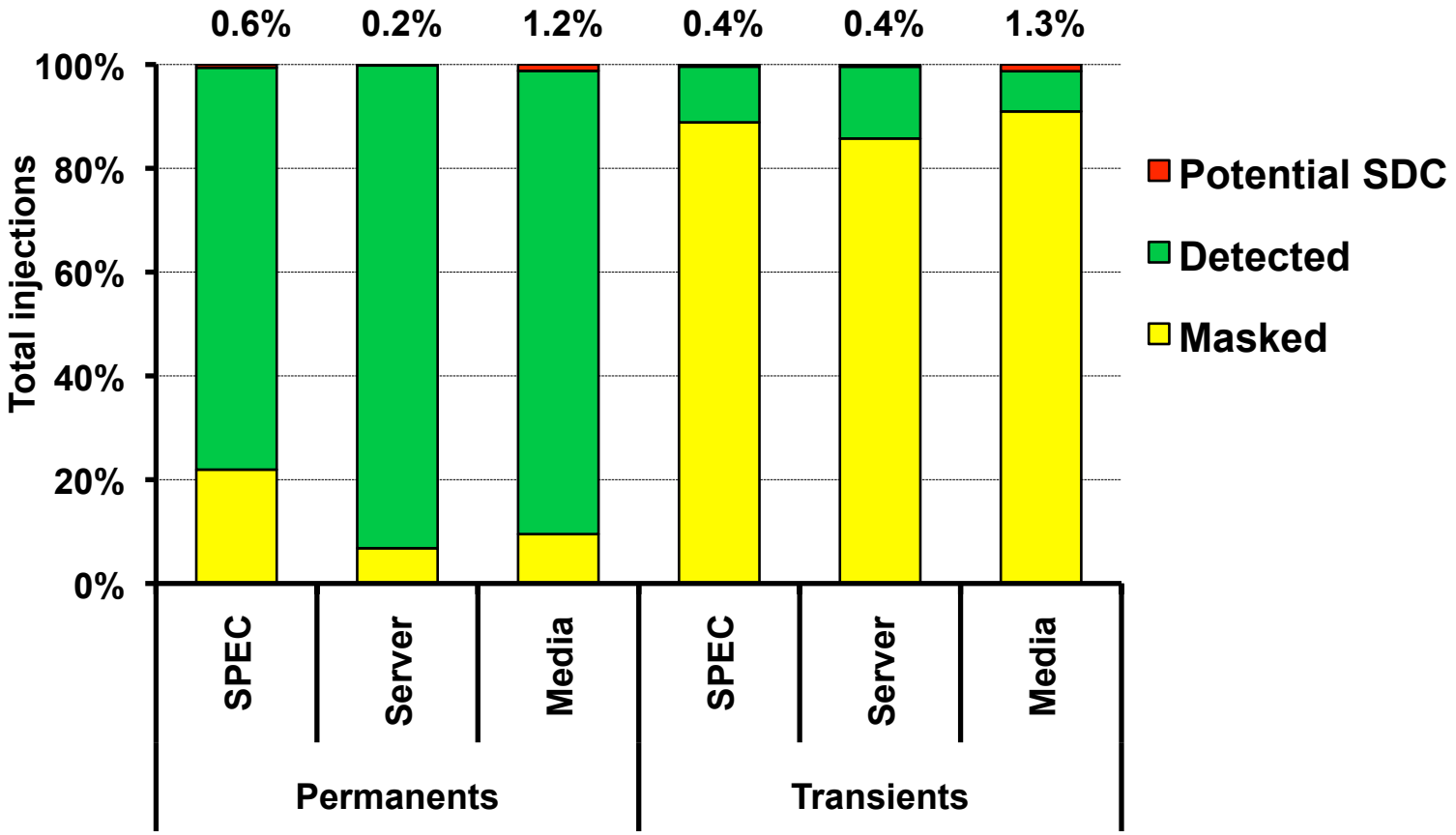


Metrics for Fault Detection

- **Potential SDC rate**
  - **SDC ⇒ undetected fault that changes app output**
  - **All unmasked, undetected faults are "potential" SDCs**
    - **"Potential" as some such faults may be tolerated**

- **Detection Latency**
  - **Latency between arch state corruption and detection**
    - **Arch state = registers + memory**
  - **Long detection latencies impede fault recovery**

Potential SDC rate for HW Faults



- **SWAT detectors effective for hardware faults**
- **What fraction of Potential SDCs are *true* SDCs?**

Application-Aware SDC Analysis

- SDC ⇒ undetected faults that corrupt app output
- But, many applications can tolerate faults
  - Client may detect fault and retry request
  - Application may perform fault-tolerant computations
    - E.g., Same cost place & route, acceptable PSNR, etc.

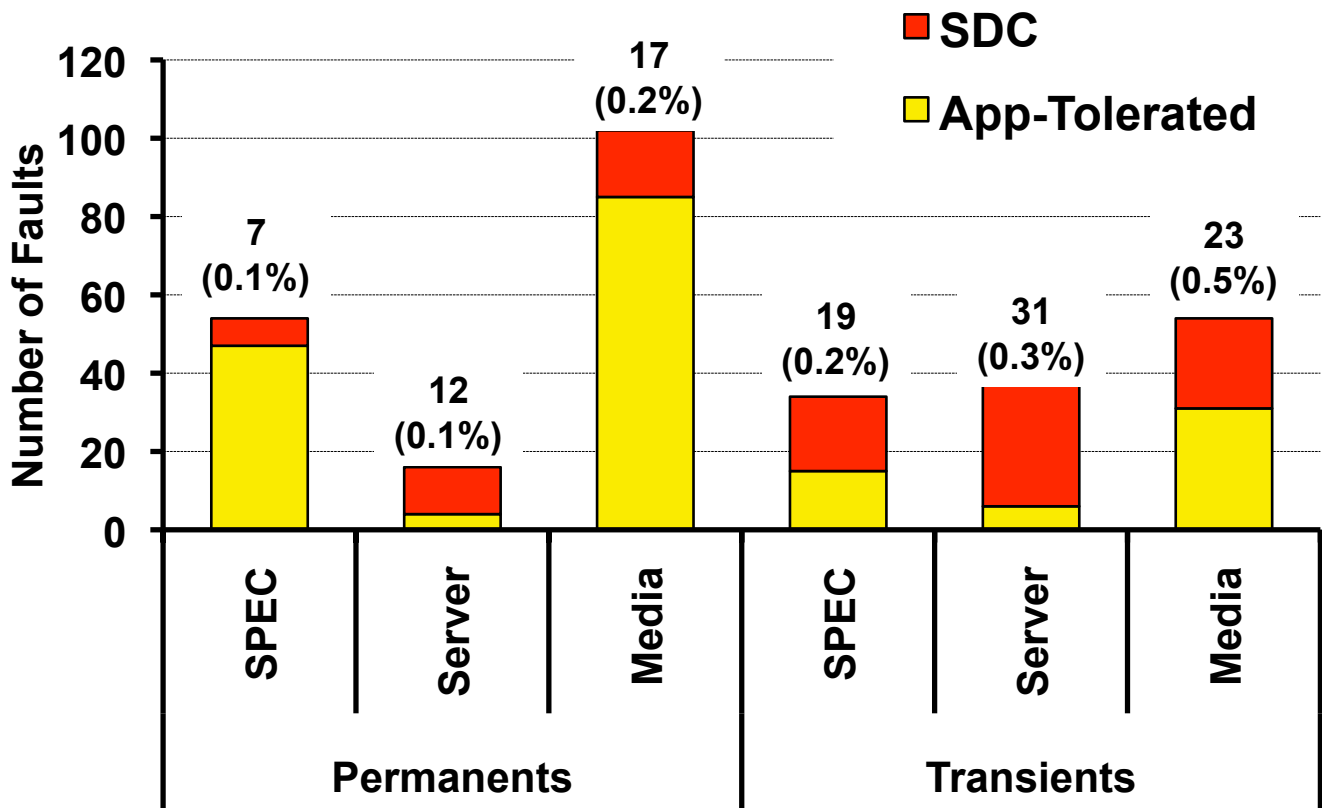
⇒ Not all undetected faults may result in SDCs

  - For each application, define notion of fault tolerance

should not?

- SWAT detectors ~~cannot~~ detect such acceptable changes

True SDCs in SWAT

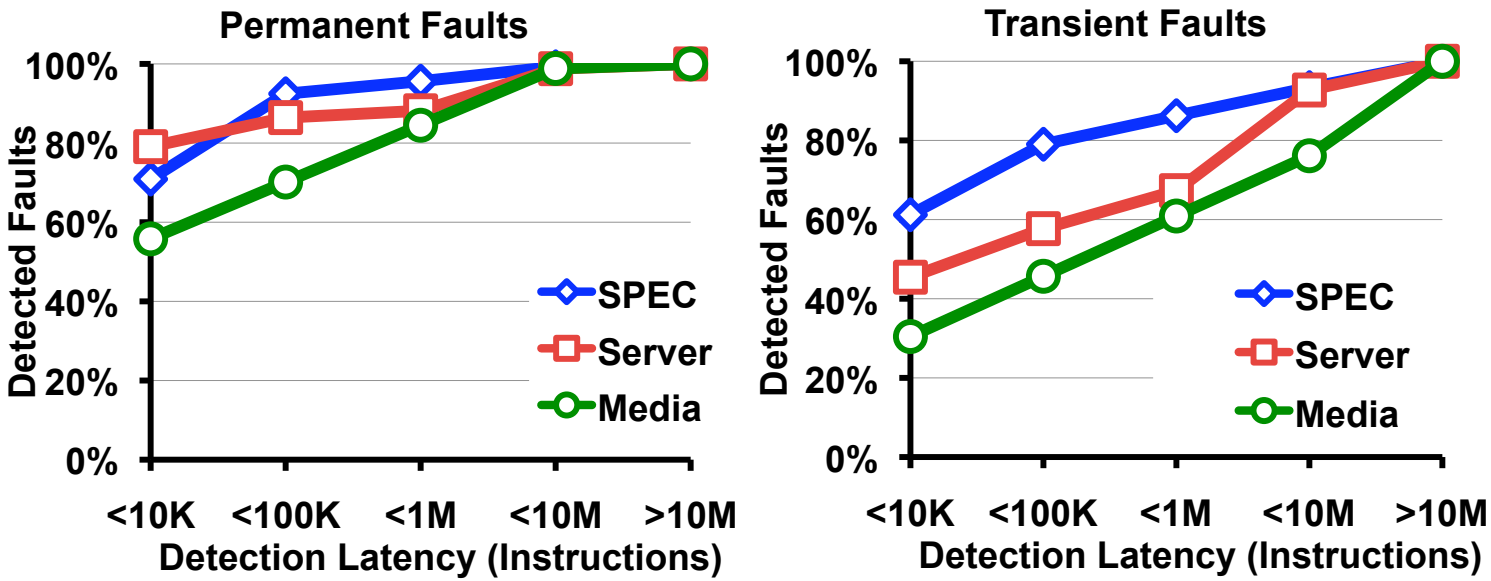


- **Only 109/48,000 faults are true SDCs (0.2% of injected faults)**
  - **63% of potential SDCs tolerated by app**

⇒ Simple SWAT detectors achieve low SDC rates

Detection Latency

- **Detection latency dictates chkpt intervals, recovery**



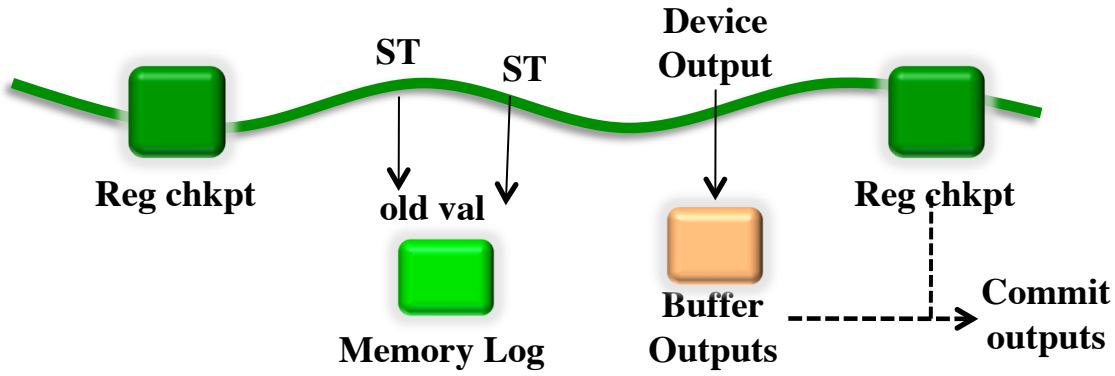
- **>98% of all faults detected in under 10M instructions**
  - **Prior work claims such faults are recoverable in HW**
  - **Our analysis follows with recovery**

Outline

- Introduction to SWAT
- Fault Detection
- **Fault Recovery**
- **Fault Diagnosis**
- **Conclusions and Future Work**

SWAT Recovery

- SWAT uses checkpointing & rollback for fault recovery
  - "Always-on" ⇒ must incur minimal overhead
- Recovery components
  - *Arch state*: Reg chkpt, mem undo log [SafetyNet, ReVive]
  - *Device state*: Reset device, restore driver [Nooks]
  - *Input Replay*: Rely on higher level protocols [ReVive I/O]
  - *Outputs*: Delay until guaranteed fault-free



Output Buffering

- External outputs cannot be rolled back after detection

⇒ Delay external outputs until guaranteed to be fault-free

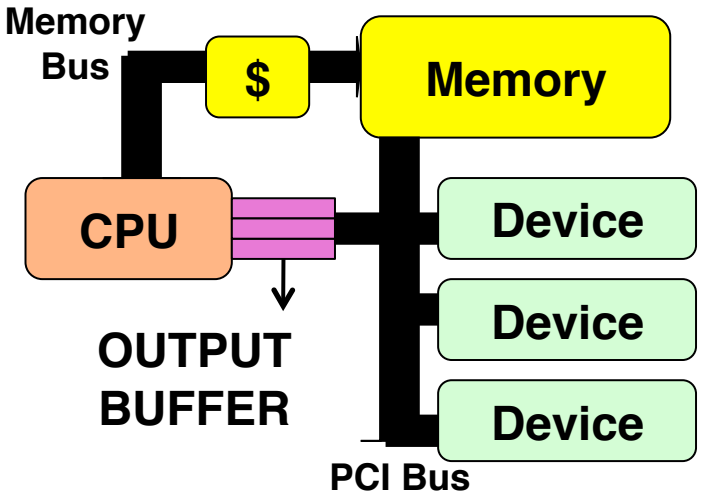
- *Previous solution*: Buffer outputs in dedicated SW [Revive I/O]
  - ✓ No HW changes, exploit output semantics
  - ⊘ Outputs vulnerable to in-core faults, SW complex

- *Our solution*: Buffer external outputs in dedicated HW
  - Low-level stores delayed ⇒ high overhead?
  - Does HW buffering require device HW changes?
  - How to reduce vulnerability of buffered outputs?

Architecture of HW Output Buffer

- CPU communicates with devices through I/O loads & stores
- HW buffer delays outputs until next chkpt

  ⇒ Committed outputs verified fault-free

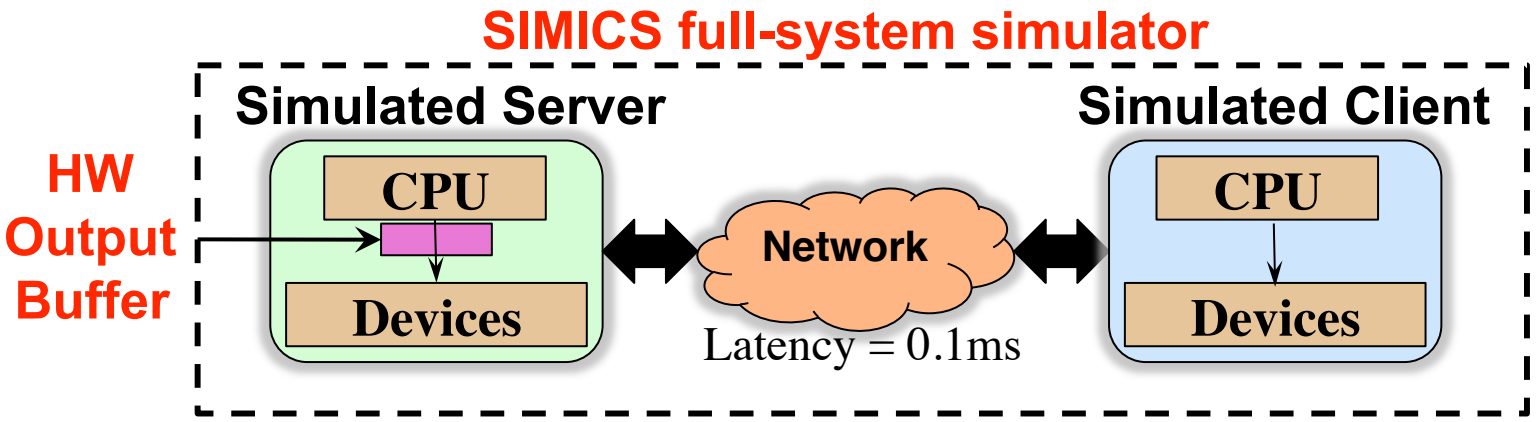


- Requires no changes to device HW
- Simple buffer design, outputs ECC checked

  ⇒Outputs protected from faults

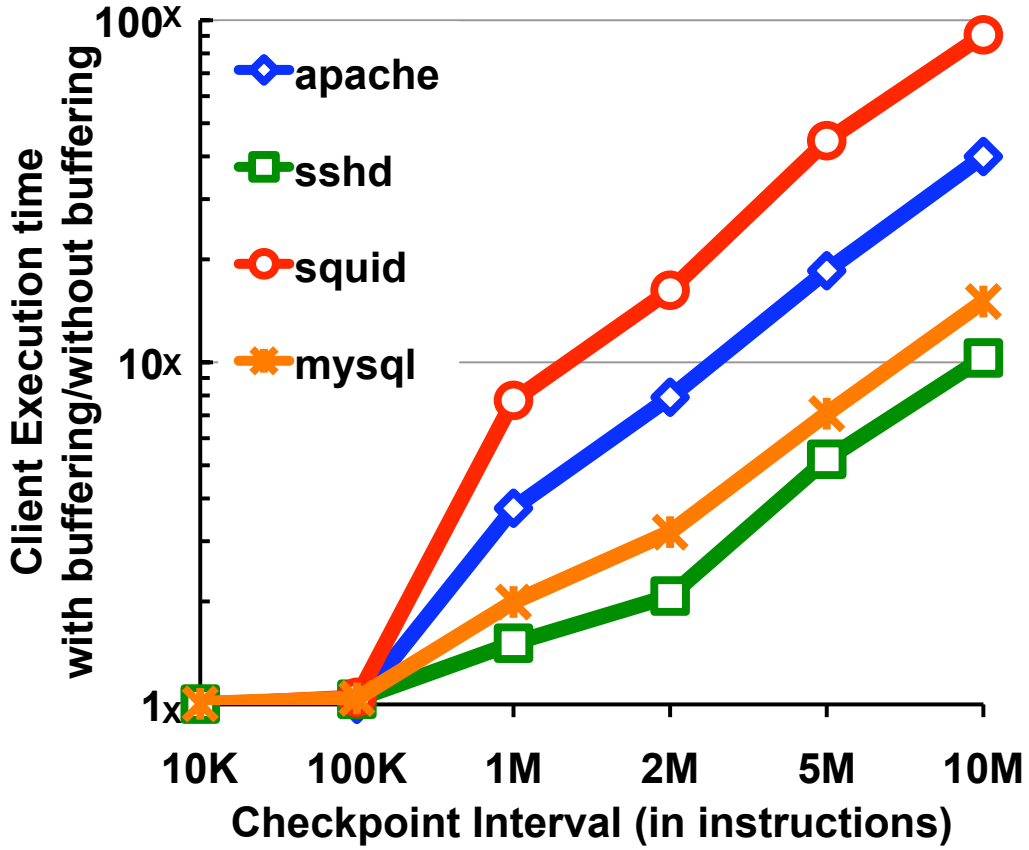
Measuring Fault-free Overheads

- HW output buffering incurs fault-free overhead
  - Outputs to clients delayed ⇒ performance overhead
  - HW to store buffered outputs ⇒ area overhead
- Evaluation setup: Output buffering on simulated server



- Focused on I/O intensive workloads to study fault recovery
  - sshd, apache, mysql, squid w/ multithreaded requests

Performance Overhead on Client from Buffering



- Chkpt interval ≤ 100K inst ⇒ <5% perf, <2KB area overhead
- Interval of millions of instr ⇒ perf overheads up to 100X!

Connecting Detection and Recovery

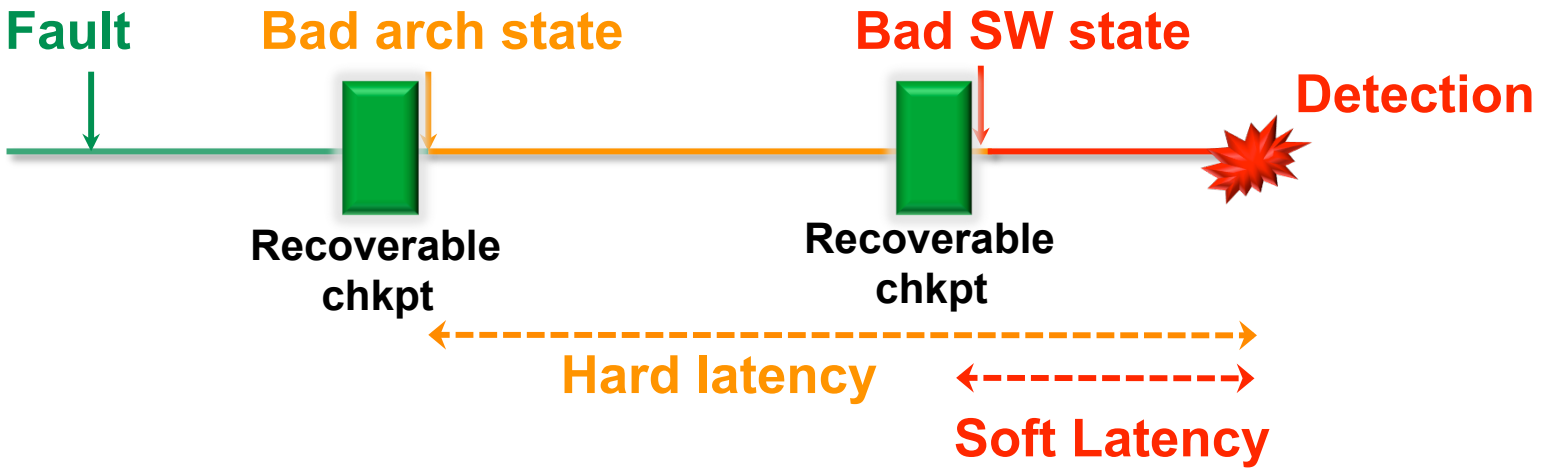
- Recovery results ⇒ chkpt interval ≤100K instrs
- Detection results ⇒ only 80% detected in 100K instrs

- Need to reduce latency to enable practical solution
  - Shortcoming identified only when components combined
    - Commonly ignored in prior work

- Goal: Reduce detection latency, enable low-cost solution
- Strategy: Redefine latency from a recovery stand-point

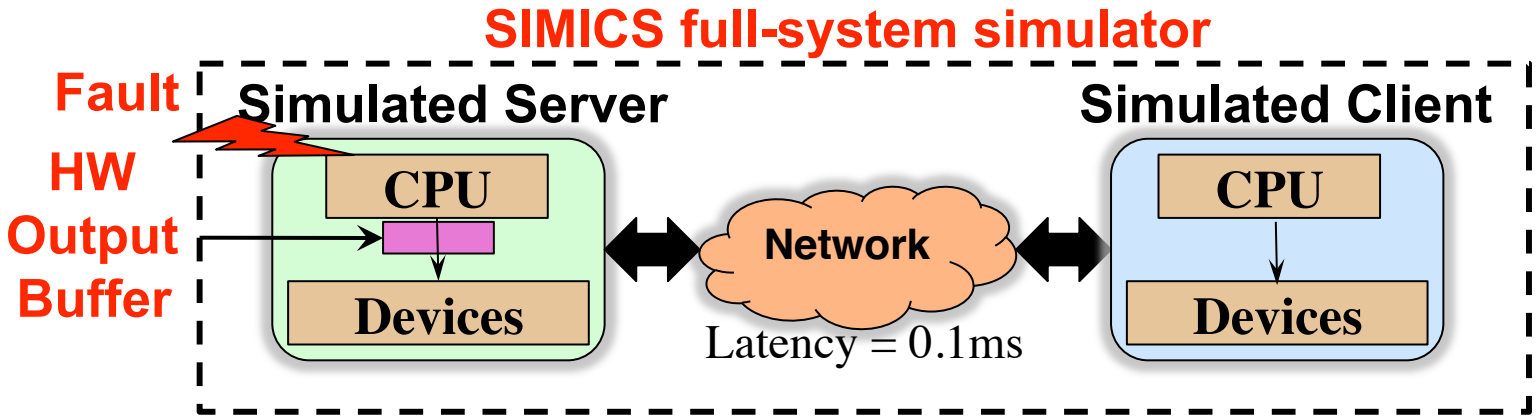
A New Definition for Detection Latency

- Traditional defn. = arch state corruption to detection
  - Fault corrupts arch state ⇒ system unrecoverable?
- But software may tolerate some corruptions!
  - E.g., *a* used only for *a>0* changes from *5* to *10*
- New defn. = SW state corruption to detection
  - Chkpt intervals should be based on new definition



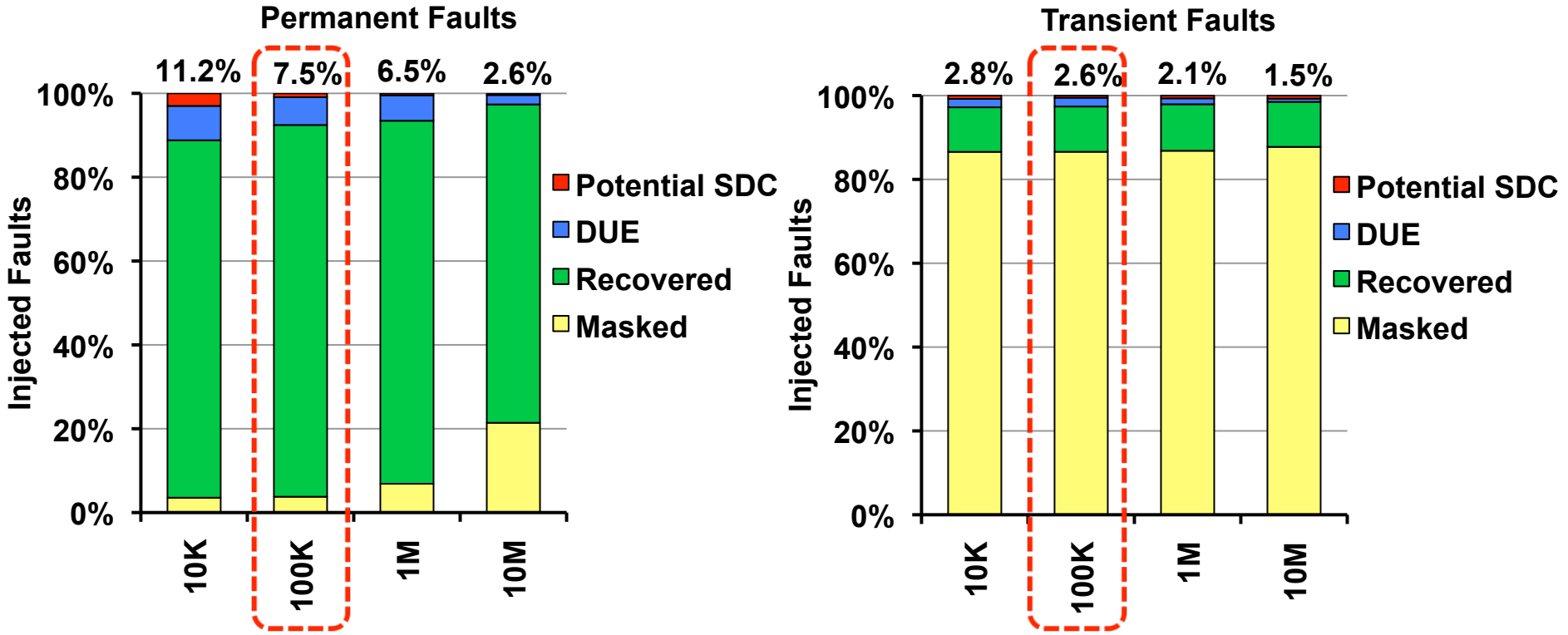
Evaluating SWAT Detection + Recovery

- μarch-level fault injections into simulated server CPU
  - Server CPU and Memory timing simulated with GEMS



- Detection: Simulate faults for 10M instructions with SWAT
- Recovery: Restore system with different chkpt intervals
  - Rollback CPU & mem, buffer outputs, restore devices
  - *Only required for evaluation, not in real system*

SWAT Detection + Recovery Results



- **95% of perm, trans faults masked or recovered at 100K inst**
  - **Only 44 faults (out of ~18K injected) are real SDCs**

⇒ **SWAT strategy effective for perm, trans HW faults**

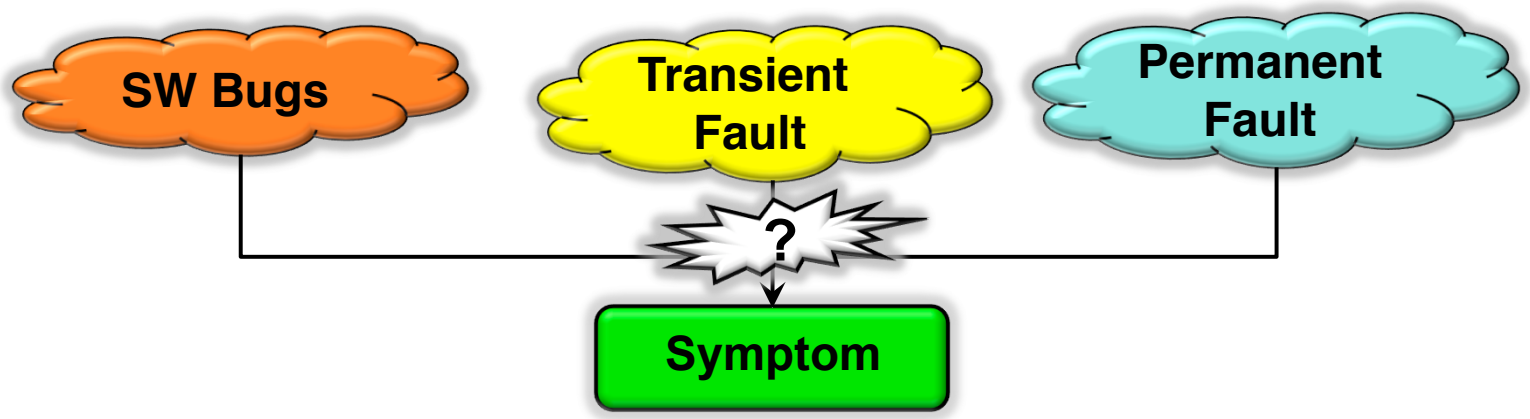
- **Low SDC rate, high recoverability, low overheads**

Outline

- Introduction to SWAT
- Fault Detection
- Fault Recovery
- **Fault Diagnosis - Overview**
- **Conclusions and Future Work**

Fault Diagnosis

- Symptom-based detection is cheap but
  - Need to diagnose root cause of fault



- Diagnosis leverages rollback/replay of SWAT recovery
- Simple rollback, replay to distinguish SW bugs, transients
- In the case of permanent faults …
  - Identify faulty core in multicore, refine to μarch-level

mSWAT: Multicore SWAT Diagnosis

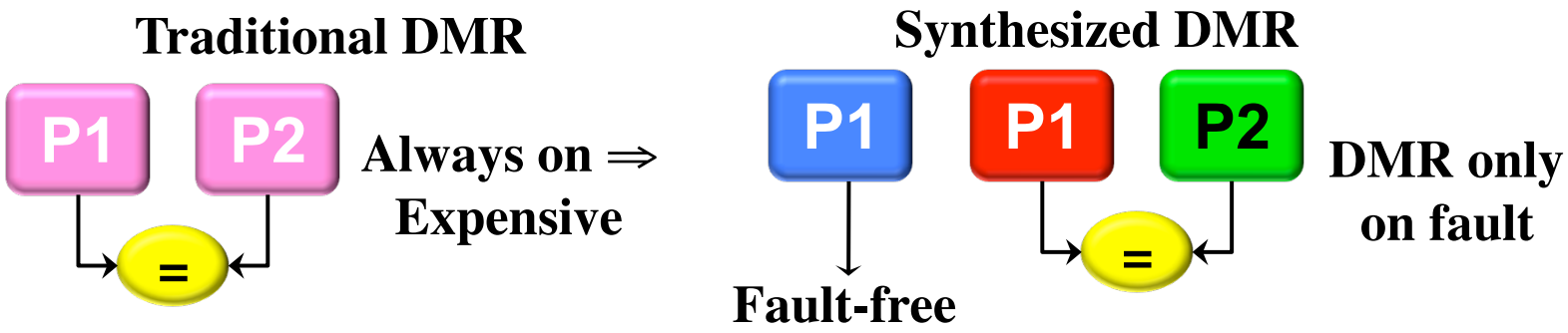
- Challenges
  - Deterministic replay of multithread app expensive
  - Fault migrates through data sharing
    - ⇒ Symptom from fault-free core, no known good core!

- Key Ideas
  - Enable isolated deterministic replay
  - Emulate TMR for diagnosis

- Results
  - Faulty core diagnosed in >95% of detected faults
    - ∗ All faults detected on fault-free cores diagnosed
  - >93% of faults diagnosed within 100K instructions

TBFD: *μ*arch-level Fault Refinement

- Disabling core wasteful ⇒ need to refine to μarch-level
- Faulty core identified ⇒ fault-free cores available
- Strategy: Replay on good, faulty core & compare μarch inv.
  - Synthesize DMR for fault diagnosis



- Results: >98% of faults diagnosed to faulty μarch unit

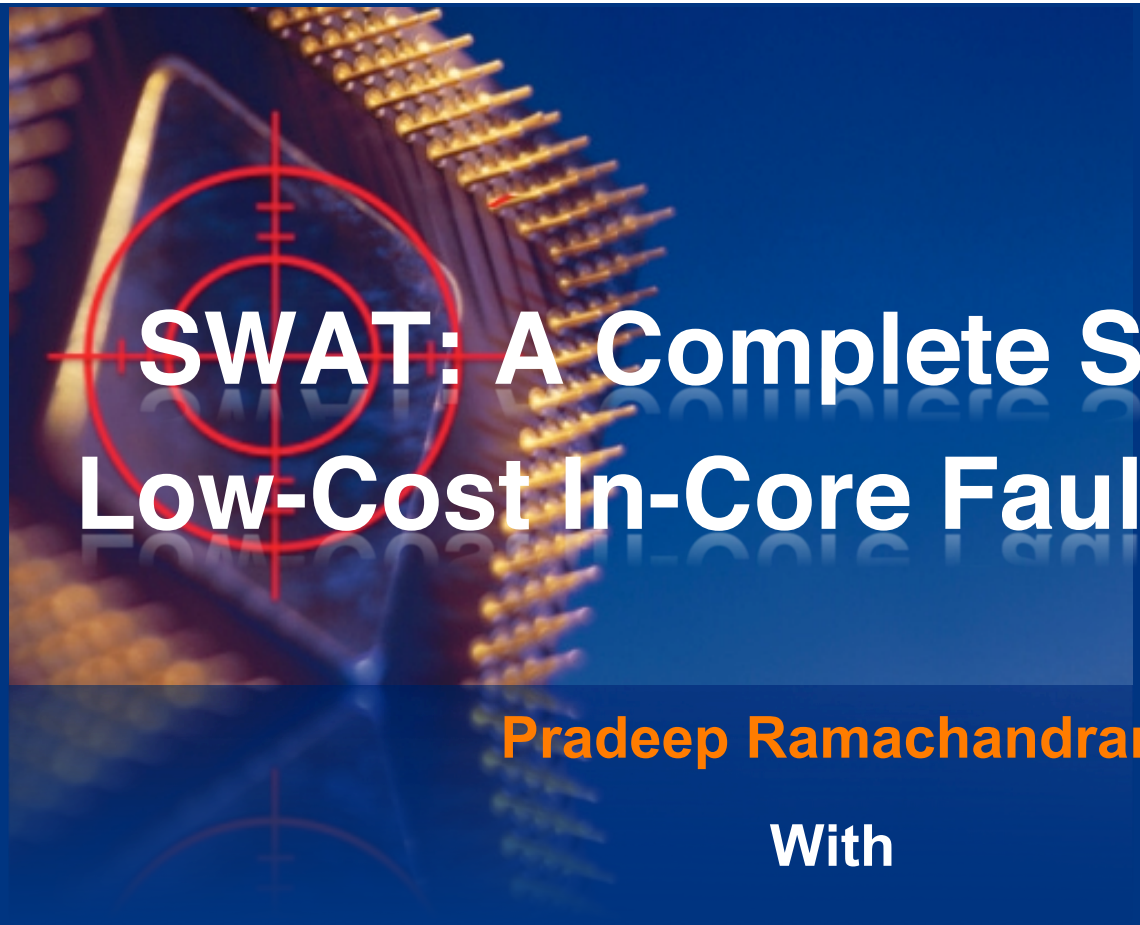
⇒ Diagnosis effective and invisible to end-user

Conclusion

- SWAT strategy demonstrated on permanent, transient faults
- Detection: Simple HW detectors effective
  - Low SDC rate, detection latency, fault-free overheads
- Recovery: HW support for recovery, output buffering
  - High recoverability, minimal impact on fault-free exec
- Diagnosis: Identify faulty core, μarch-level diagnosis
  - Effective diagnosis in the rare event of a fault
- Overall, SWAT ⇒ Complete solution for in-core HW faults

Ongoing and Future Work

- **Prototyping SWAT on FPGA for real-world evaluation**
  - **Implementation on OpenSPARC FPGA**
    - **Collaboration with Univ. of Michigan CrashTest**
- **Formal framework for why and where SWAT works**
  - **Reduce SDCs even further through more detectors**
  - **Identify hard-to-detect codes, provide SDC bounds**
    - **Collaboration with Intel**
- **Application-level resiliency**
  - **Systematic exploitation for detection and recovery**
- **Expanding SWAT to other types of faults**
  - **Faults in off-core components, other fault models**



SWAT: A Complete Solution for Low-Cost In-Core Fault Resiliency

Pradeep Ramachandran

With

Manlap Li, Siva Hari, Lei chen, Byn Choi, Swarup Sahoo, Rob Smolinski,

Sarita Adve, Vikram Adve, Yuanyuan Zhou

Department of Computer Science

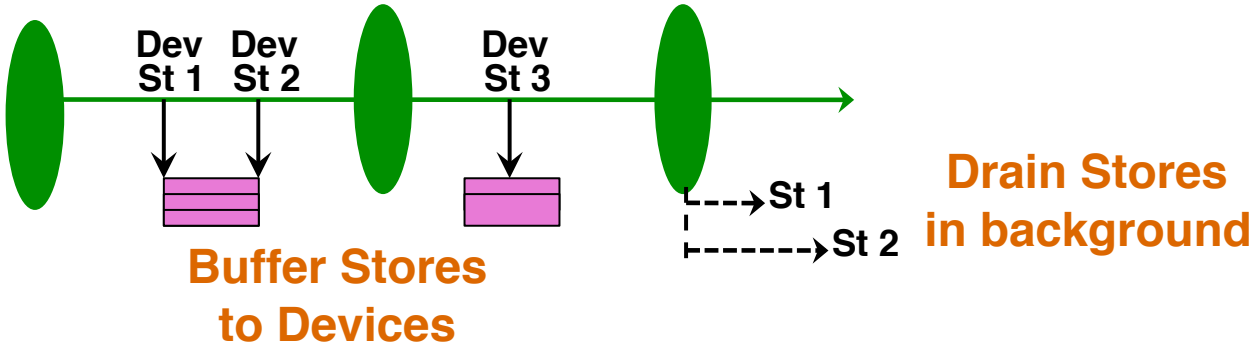
University of Illinois at Urbana Champaign

swat@cs.uiuc.edu

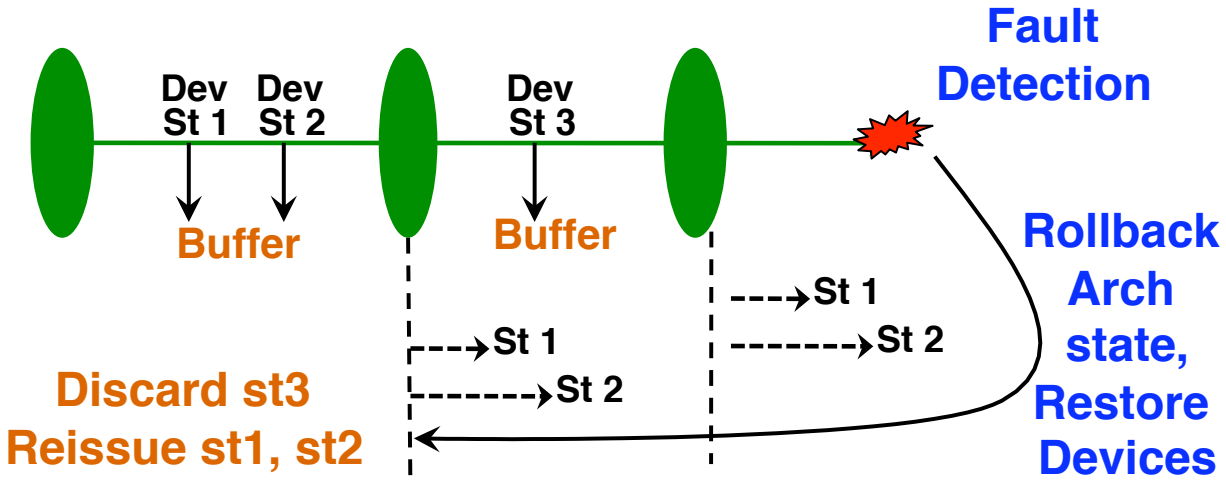
Backup Slides

Operations of HW Output Buffer

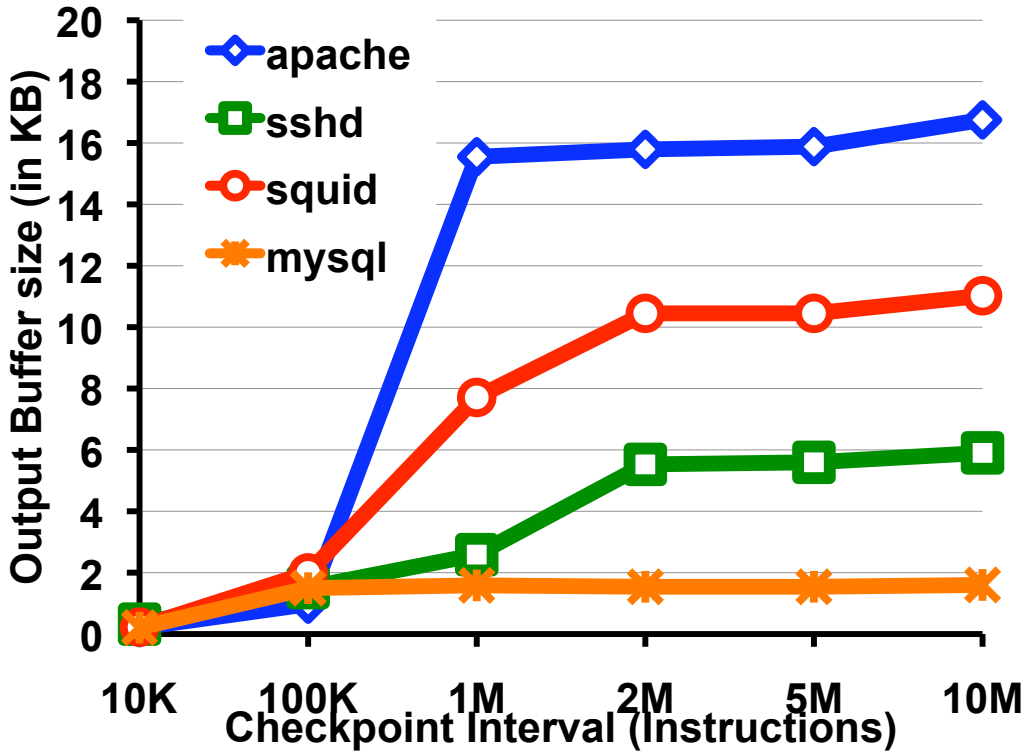
- **Fault-free Operation**



- **Recovery Operation**



Area Overhead from Buffering

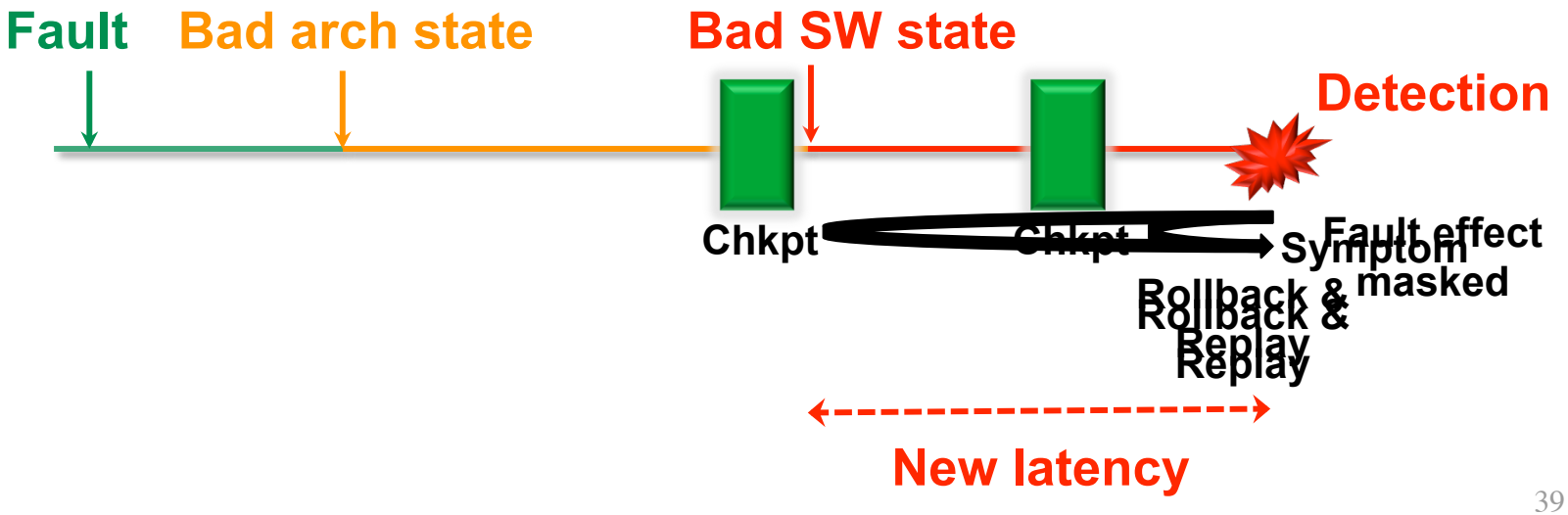


- **Small HW buffers for intervals of under ≤100K instructions**
  - **~2KB buffer for 100K instr ⇒ implementable on-core**

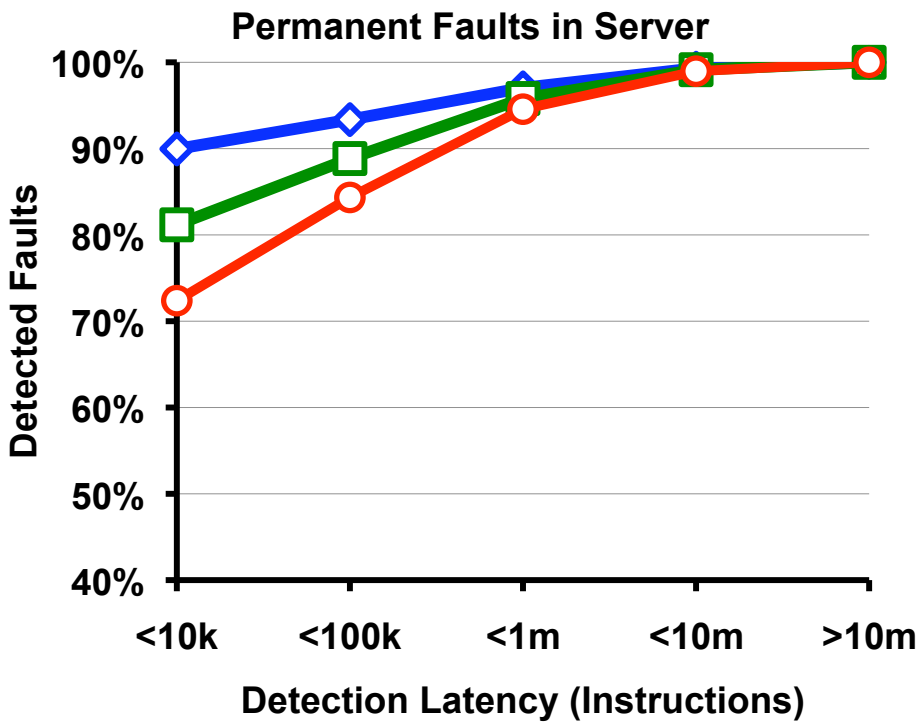
⇒ Checkpoint interval ≤ 100K instr for practical recovery

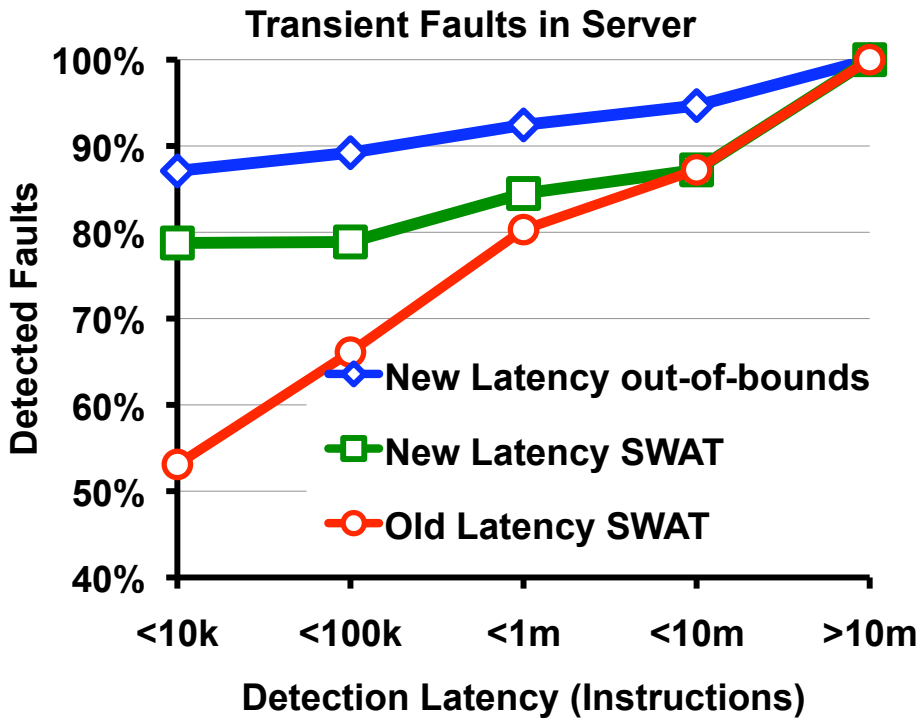
Measuring Detection Latency

- New detection latency = SW state corruption to detection
- But identifying SW state corruption is hard!
  - Need to know how faulty value used by application
  - If faulty value affects output, then SW state corrupted
- Measure latency by rolling back to older checkpoints
  - Only for analysis, not required in real system



Detection Latency - Server



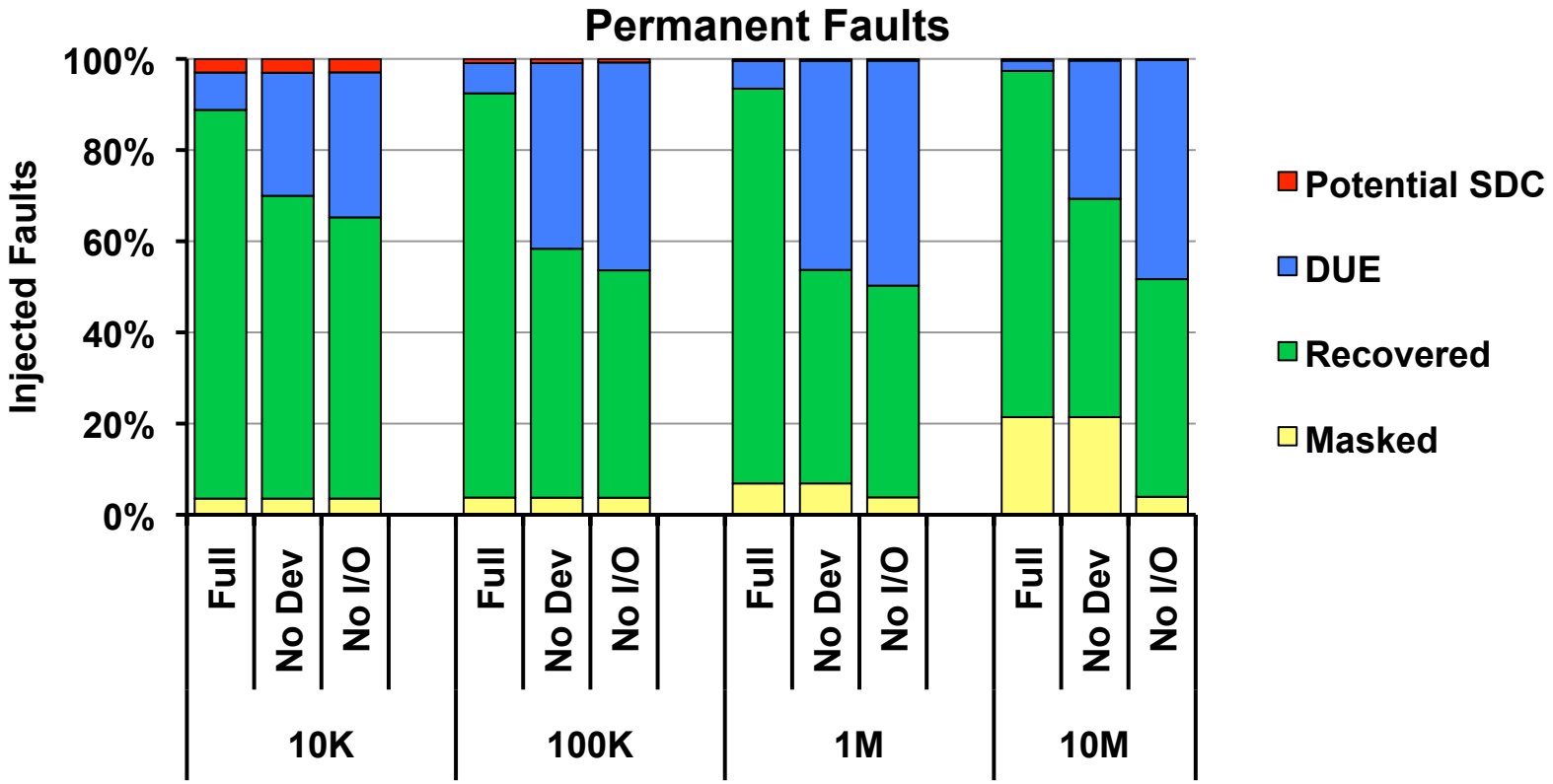


- **Measuring new latency important to study recovery**
- **New techniques significantly reduce detection latency**
  - **>90% of faults detected in <100K instructions**
- **Reduced detection latency impacts recoverability**

Overheads from Other Chkpt Components

- **Register checkpoint**
  - **Negligible overheads at 100K chkpt interval [SafetyNet]**
- **Memory logging**
  - **Max log size at 100K chkpt interval = 450KB**
    - **Can be collected with small HW that is memory backed**
  - **<1% perf overheads at 100K chkpt interval [SafetyNet]**
- **Device chkpt**
  - **<1% average perf overhead for 3 sample drivers [Nooks]**

Importance of I/O for Fault Recovery



- **No device recovery, output buffering ↓ recoverability 89%**

⇒ Critical components are required for recovery