MSWAT: Hardware Fault Detection and Diagnosis for Multicore Systems

Siva Kumar Sastry Hari, Man-Lap (Alex) Li,
Pradeep Ramachandran, Byn Choi, Sarita Adve

Department of Computer Science
University of Illinois at Urbana-Champaign
swat@cs.uiuc.edu

Motivation

- **Goal**
  - **Hardware resilience with low overhead**
- **Previous Work**
  - **SWAT – low-cost fault detection and diagnosis**
  - **For single-threaded workloads**
- **This work**
  - **Fault detection and diagnosis for multithreaded apps**

SWAT Background

SWAT Observations

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

SWAT Approach

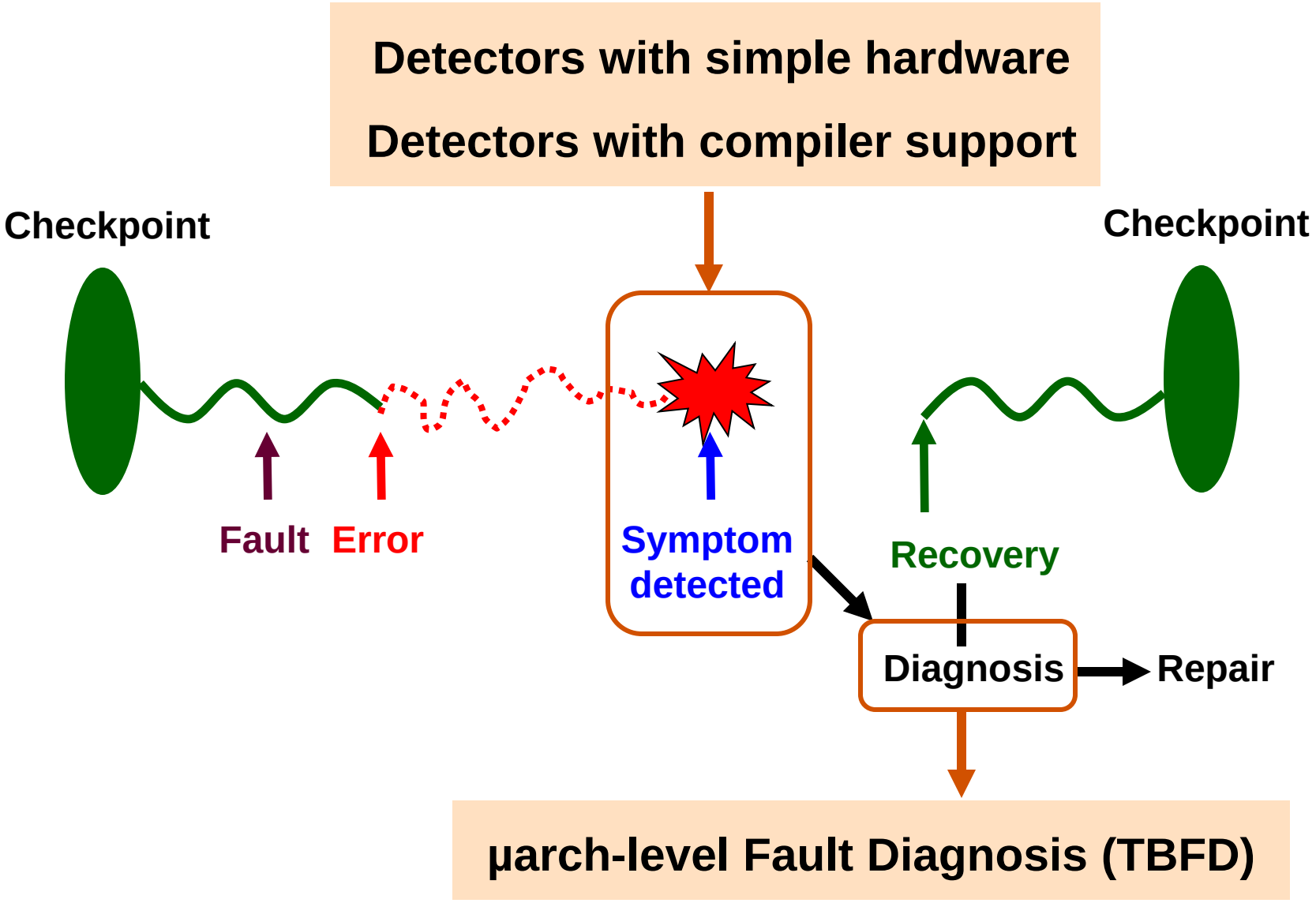
⇒ Watch for software anomalies (symptoms)

Zero to low overhead "always-on" monitors

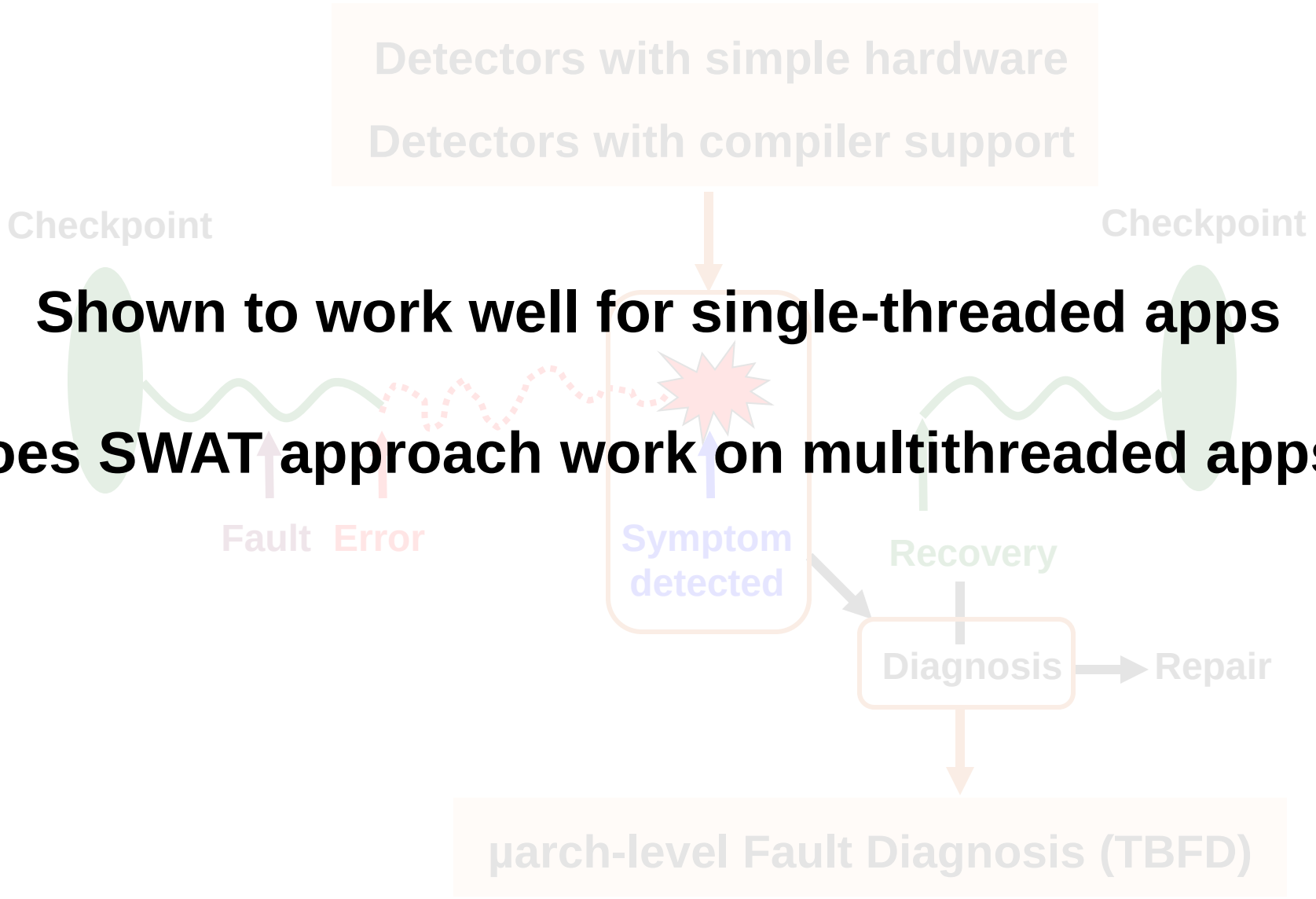
Diagnose cause after symptom detected

May incur high overhead, but rarely invoked

SWAT Framework Components



Challenge

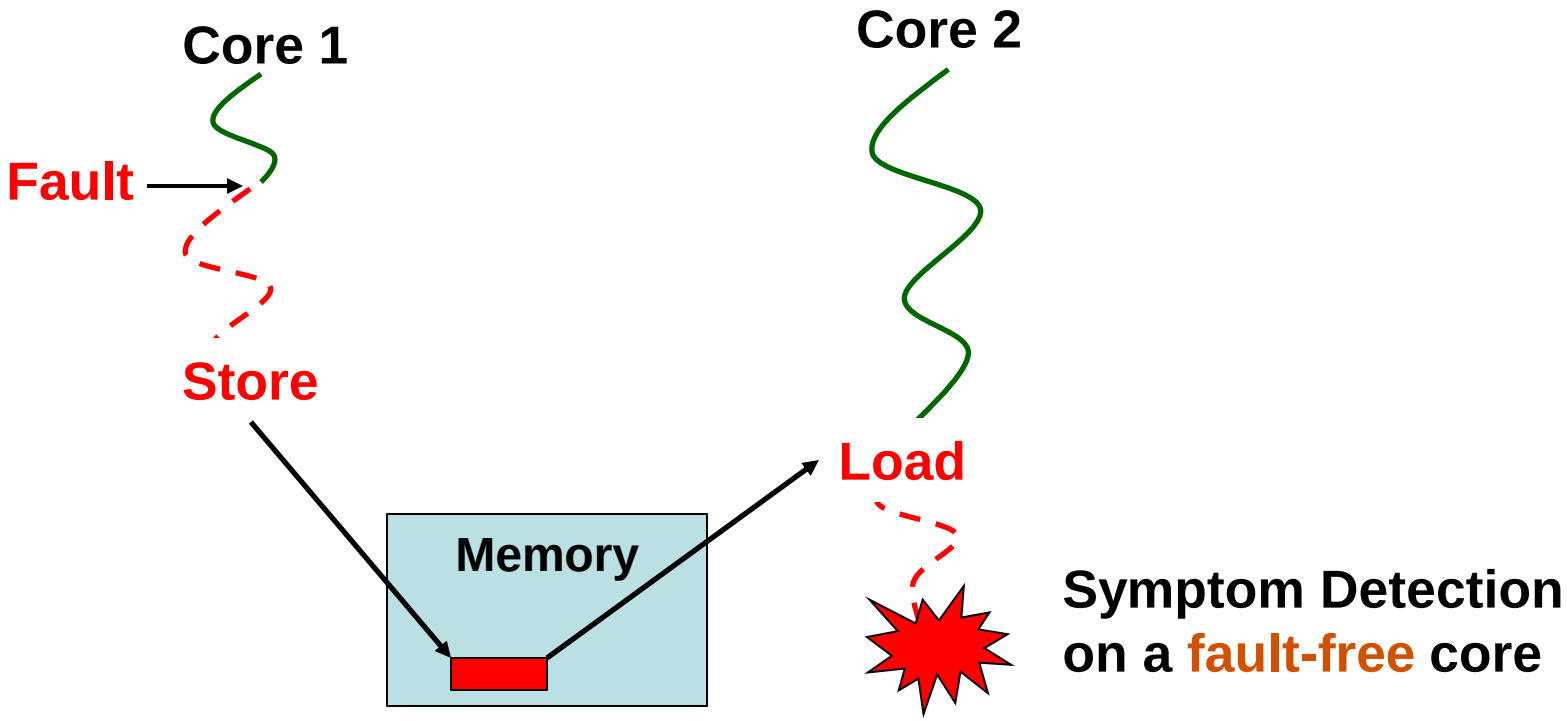


Shown to work well for single-threaded apps

Does SWAT approach work on multithreaded apps?

Multithreaded Applications

- **Multithreaded apps share data among threads**



- **Symptom causing core may not be faulty**
- **Need to diagnose faulty core**

Contributions

- **Evaluate SWAT detectors on multithreaded apps**
  - **High fault coverage for multithreaded workloads too**
  - **Observed symptom from fault-free cores**

- **Novel fault diagnosis for multithreaded apps**
  - **Identifies the faulty core despite fault propagation**
  - **Provides high diagnosability**

Outline

- Motivation
- **MSWAT Detection**
- **MSWAT Diagnosis**
- **Results**
- **Summary and Advantages**
- **Future Work**

SWAT Hardware Fault Detection

- **Low-cost monitors to detect anomalous software behavior**

- **Fatal traps detected by hardware**
  - **Division by Zero, RED State, etc.**
- **Hangs detected using simple hardware hang detector**
- **High OS activity using performance counters**
  - **Typical OS invocations take 10s or 100s of instructions**

MSWAT Fault Detection

- **New symptom: Panic detected when kernel panics**
  - **Detected using hardware debug registers**

- **SWAT-like detectors provide high coverage**

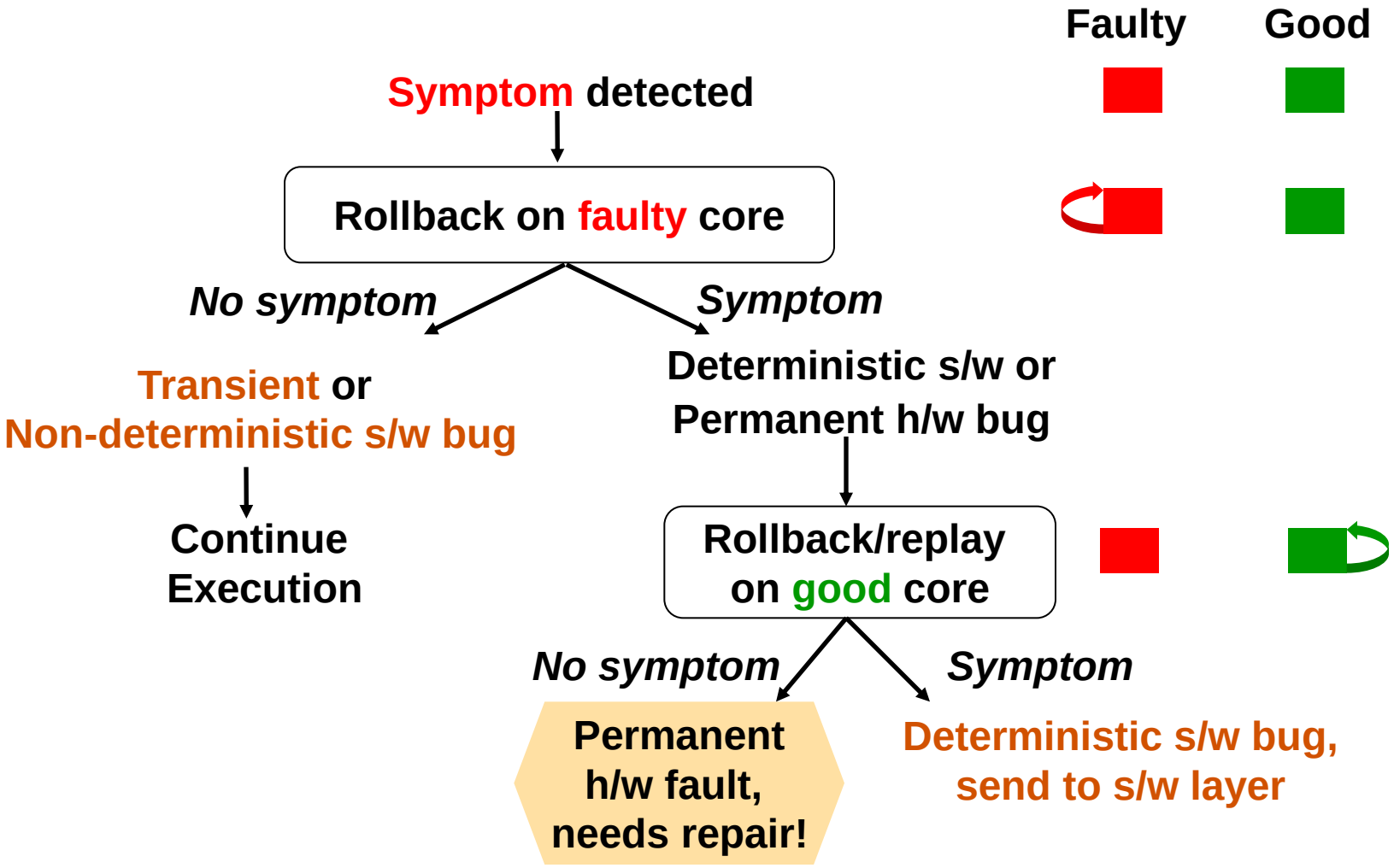
Fault Diagnosis

- **After detection, invoke diagnosis to identify the faulty core**

- **Replay fault activating execution**

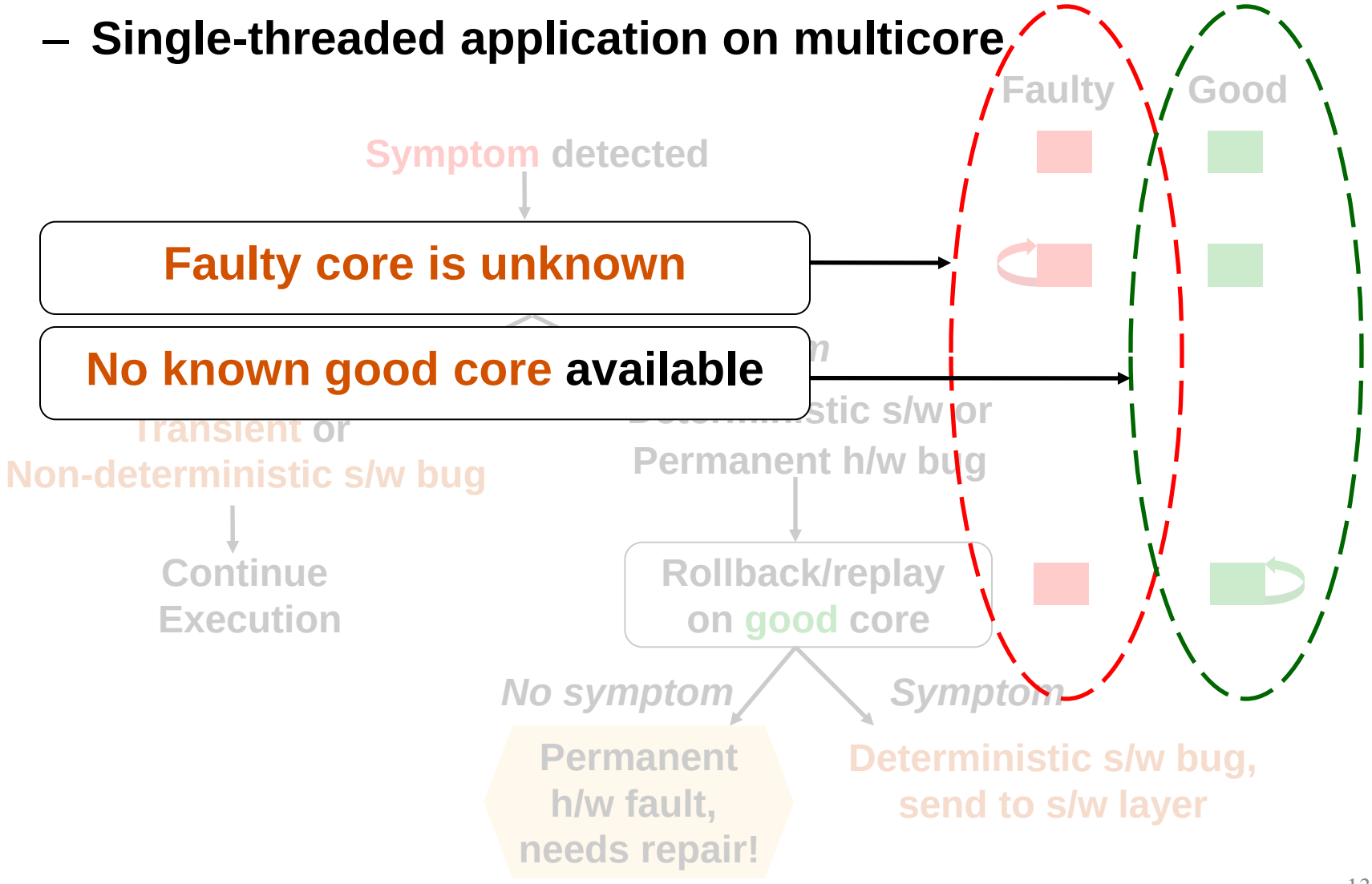
SWAT Fault Diagnosis

- **Rollback/replay on same/different core**
  - **Single-threaded application on multicore**



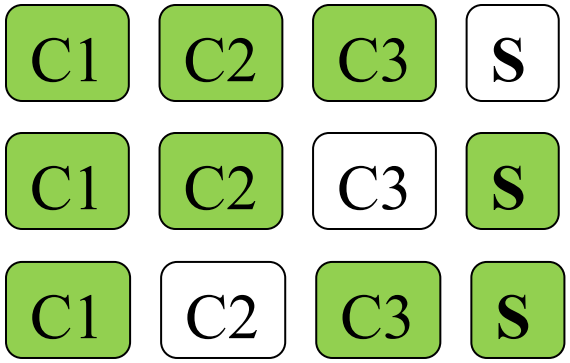
SWAT Fault Diagnosis

- **Rollback/replay on same/different core**
  - **Single-threaded application on multicore**



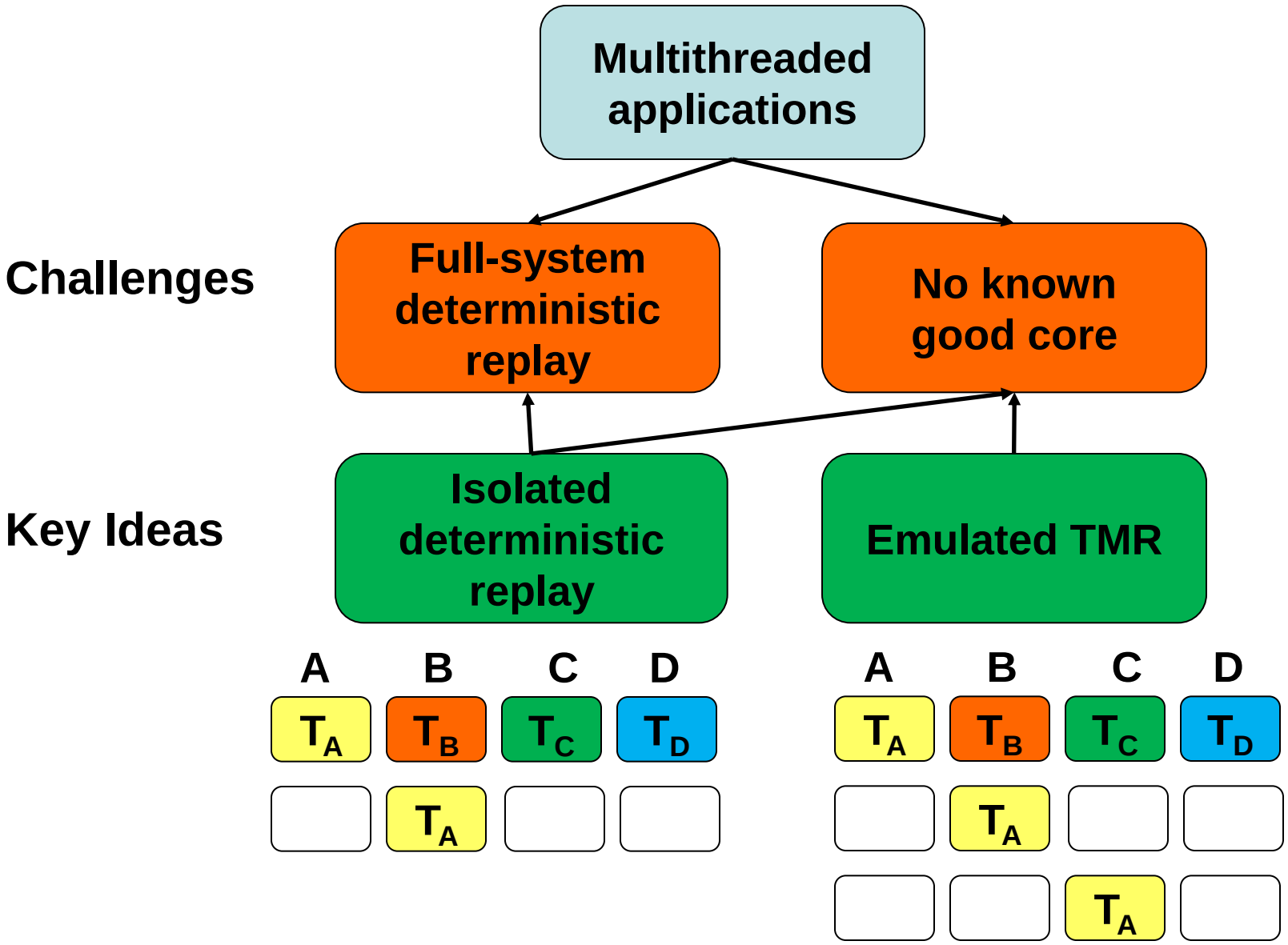
Extending SWAT Diagnosis to Multithreaded Apps

- **Naïve extension – N known good cores to replay the trace**
  - 👎 **Too expensive – area**
  - 👎 **Requires full-system deterministic replay**
- **Simple optimization – One spare core**

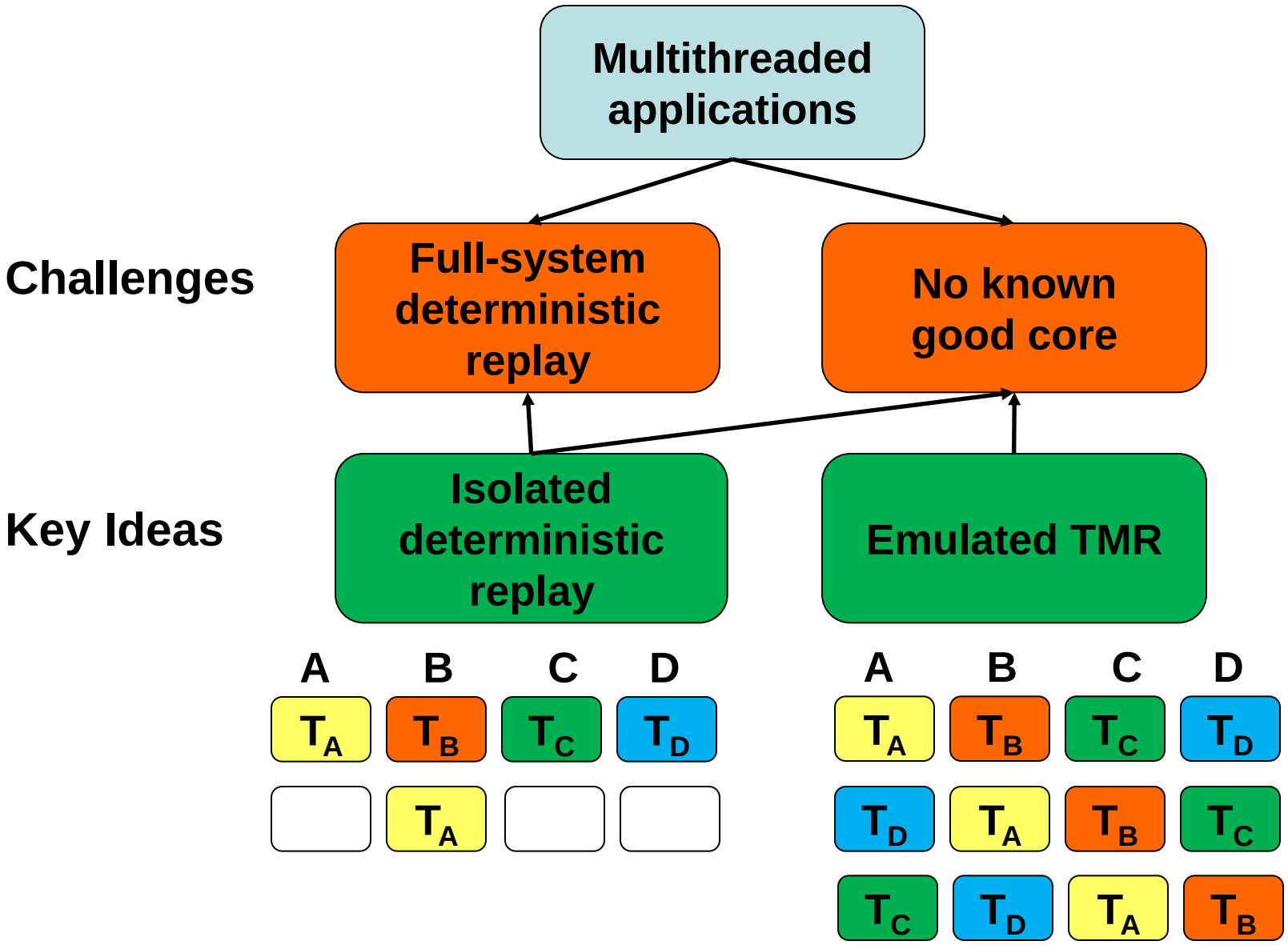


  - 👎 **Not Scalable, requires N full-system deterministic replays**
  - 👎 **Requires a spare core**
  - 👎 **Single point of failure**

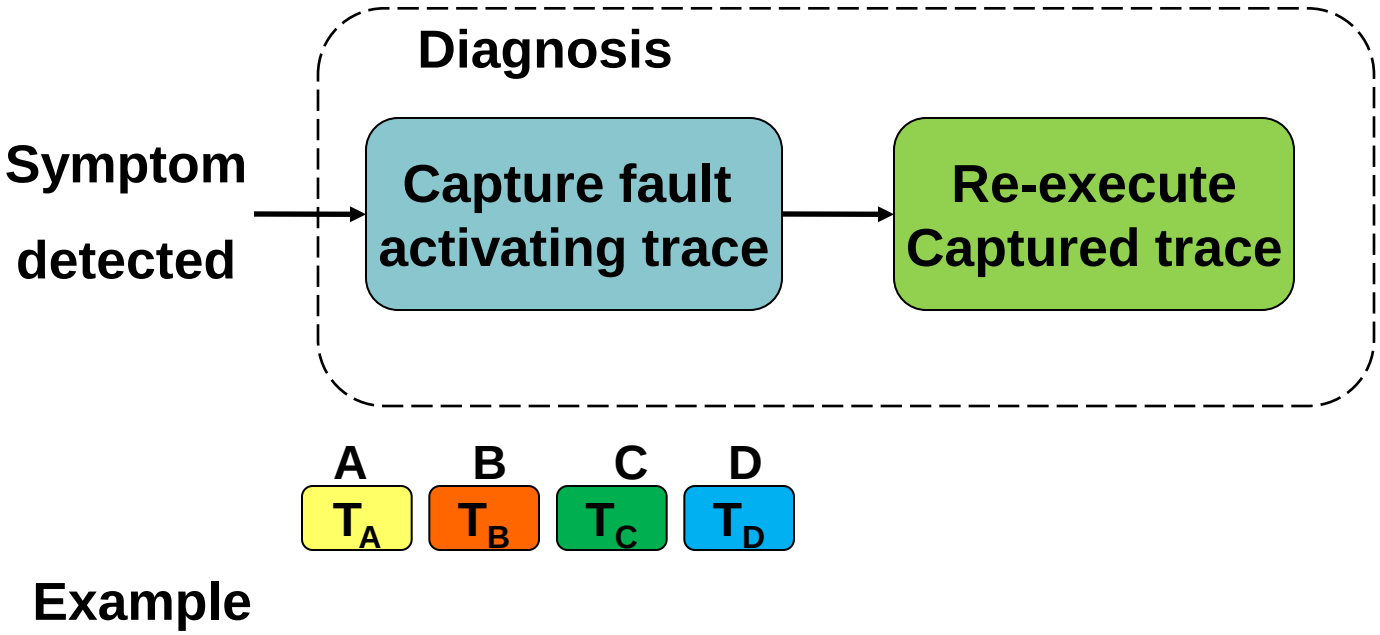
MSWAT Diagnosis - Key Ideas



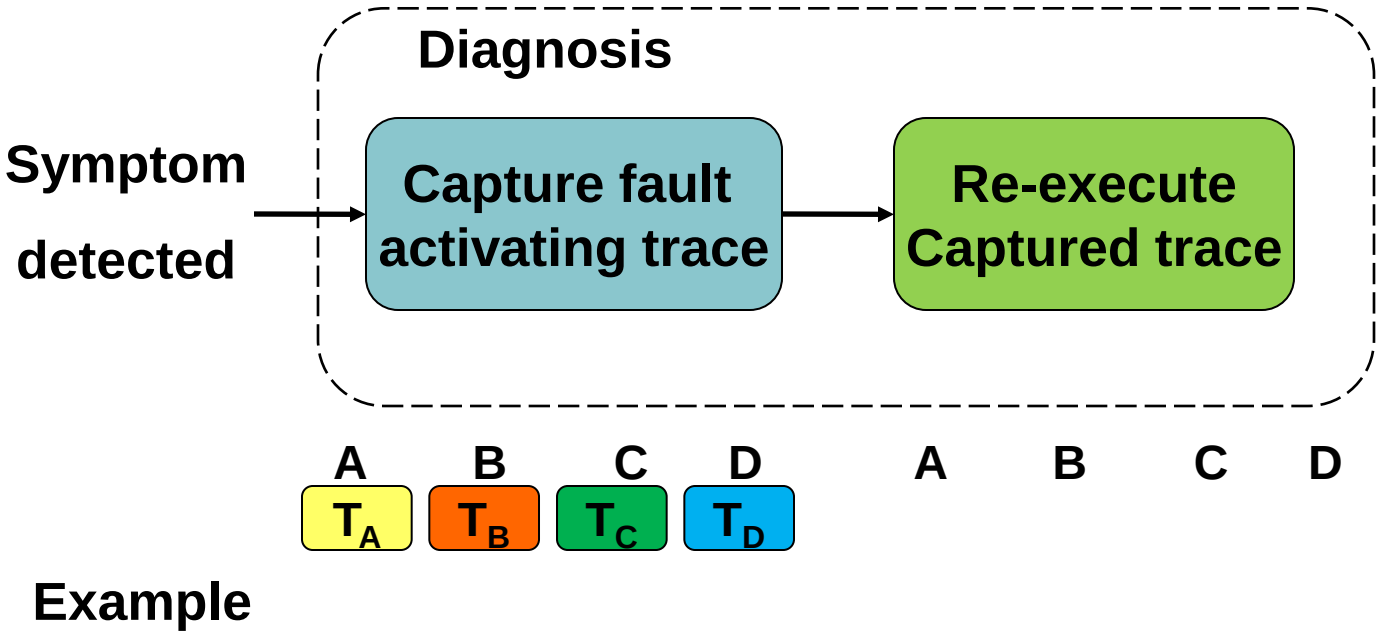
MSWAT Diagnosis - Key Ideas



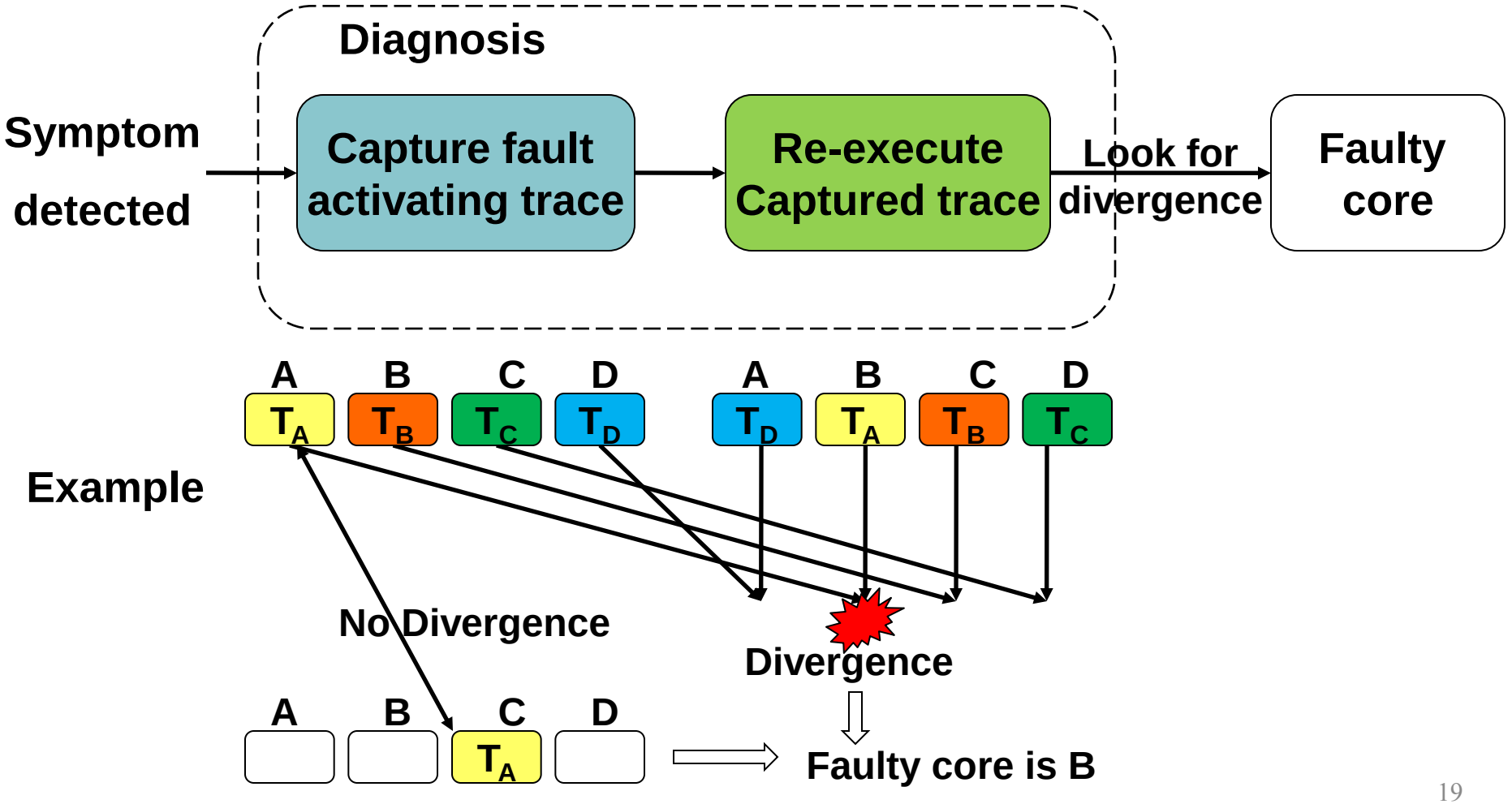
Multicore Fault Diagnosis Algorithm



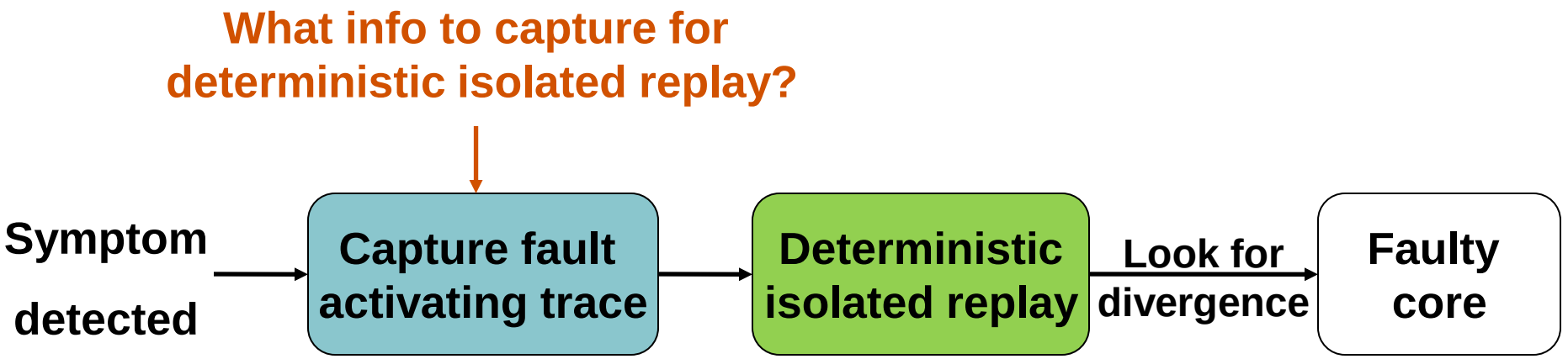
Multicore Fault Diagnosis Algorithm



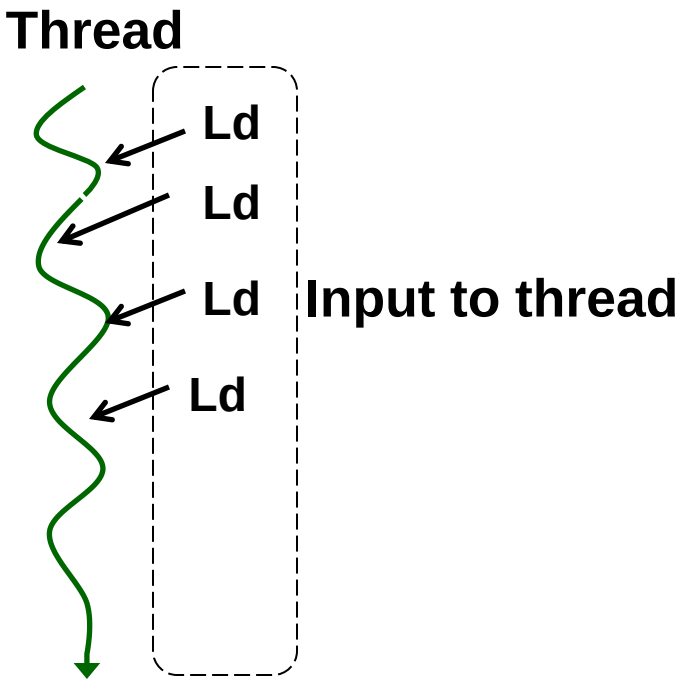
Multicore Fault Diagnosis Algorithm



Multicore Fault Diagnosis Algorithm

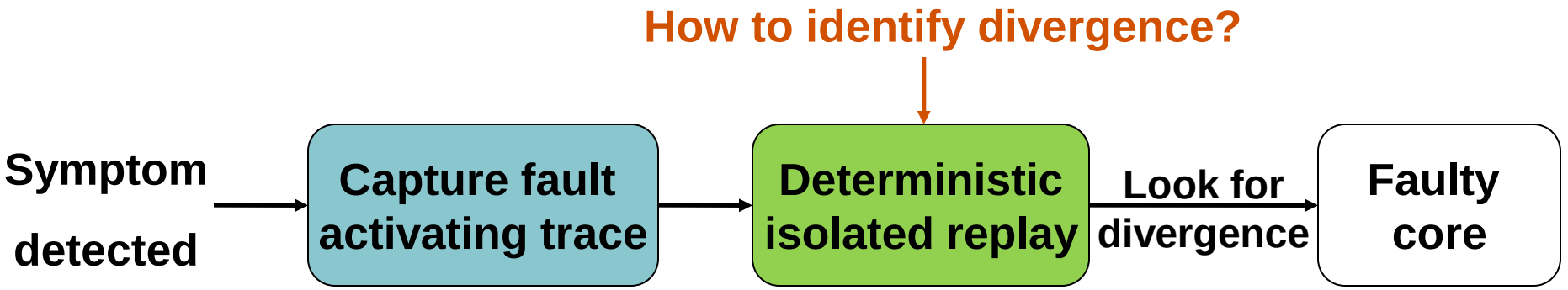


Enabling Deterministic Isolated Replay



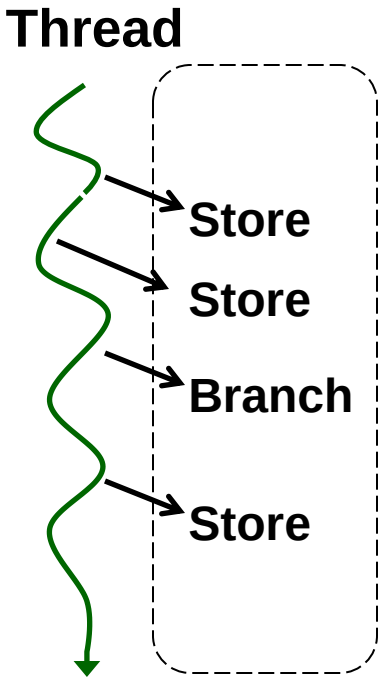
- Capturing input to thread is sufficient for deterministic replay
  - Record all retiring loads
- Enables isolated replay of each thread

Multicore Fault Diagnosis Algorithm



Identifying Divergence

- Comparing all instructions ⇒ Large buffer requirement
- Faults corrupt software through
  - Memory and control instructions
  - Comparing all retiring store and branch is sufficient



Hardware Cost

- **The first replay is native execution**
  - **Minor support for collection of trace**

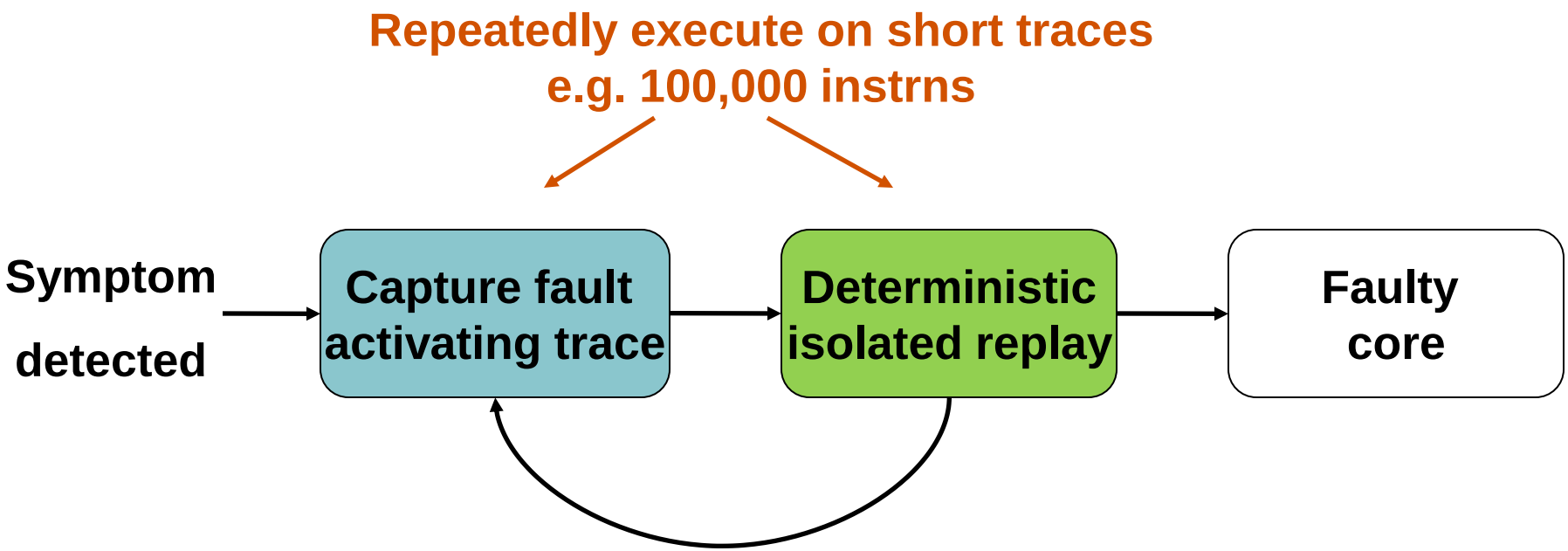
- **Deterministic replay is firmware emulated**
  - **Requires minimal hardware support**
  - **Replay threads in isolation**

    ⇒ **No need to capture memory orderings**

Trace Buffer Size

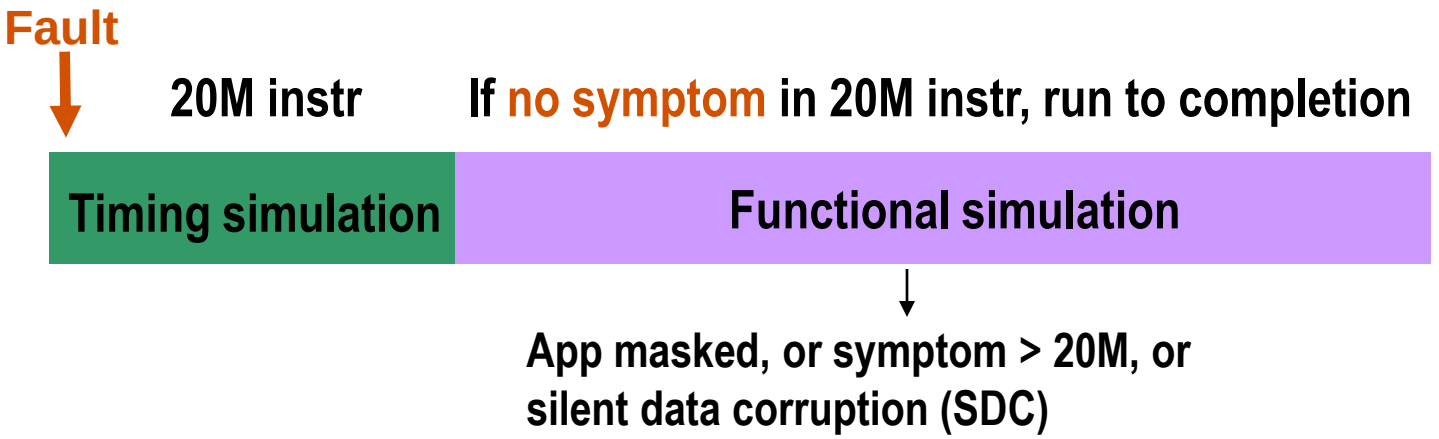
- **Long detection latency ⇒ large trace buffers (8MB/core)**
  - **Need to reduce the size requirement**

  ⇒ Iterative Diagnosis Algorithm



Experimental Methodology

- Microarchitecture-level fault injection
  - GEMS timing models + Simics full-system simulation
  - Six multithreaded applications on OpenSolaris
- Permanent fault models
  - Stuck-at faults in latches of 7 μarch structures
- Simulate impact of fault in detail for 20M instructions



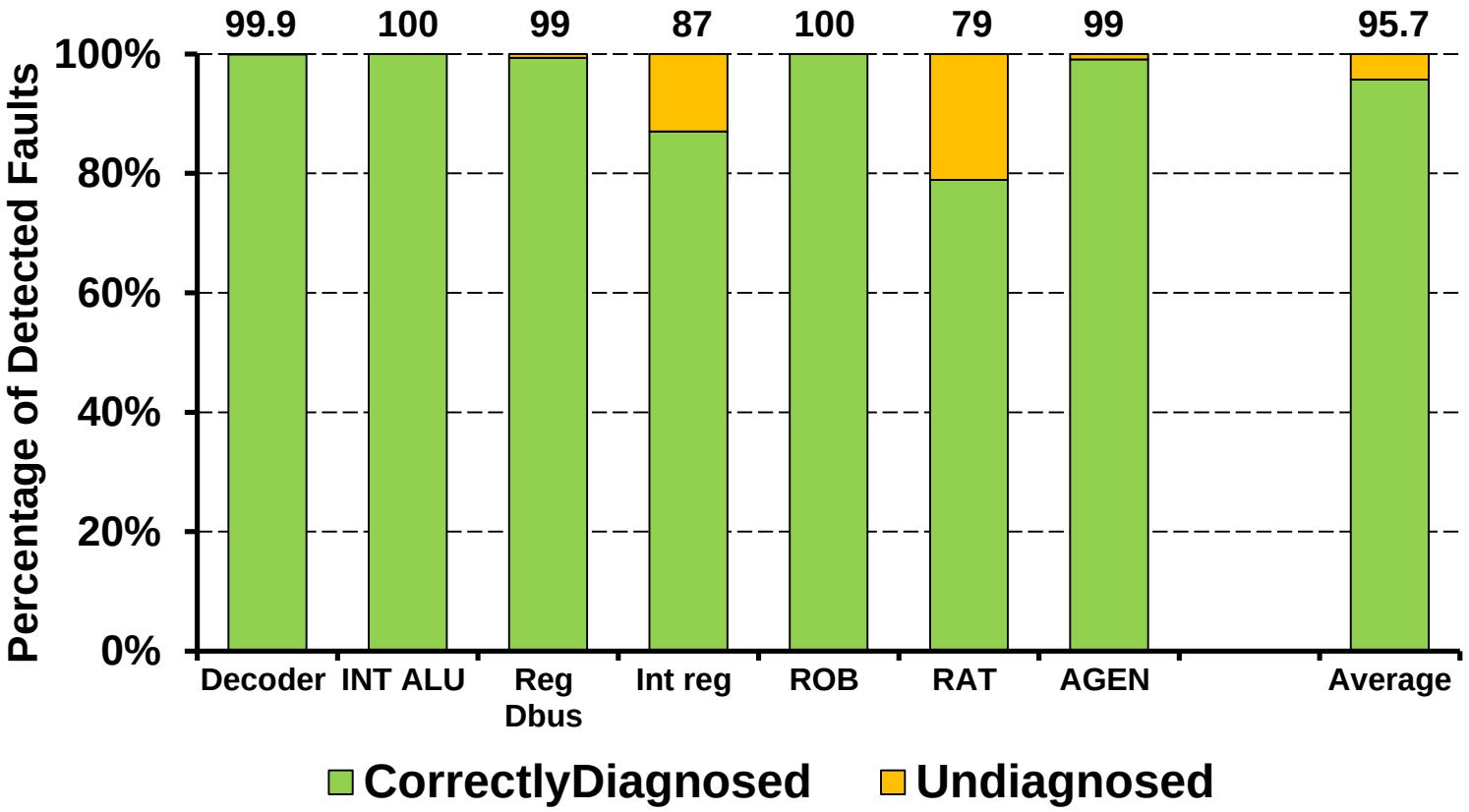
Experimental Methodology

- Iterative algorithm with 100,000 instrns in each iteration
  - Until divergence or 20M instrns
- Deterministic replay is native execution
  - not firmware emulated

Results: MSWAT Fault Detection

- **Coverage:**
  - **Over 98% faults detected**
  - **Only 0.2% give Silent Data Corruptions (SDCs)**
- **Low SDC rate of 0.4% for transient faults as well**
- **12% of detections occur in fault-free core**
  - **Data sharing propagates faults from faulty to fault-free core**

Results: MSWAT Fault Diagnosis (1/2)



- **Over 95% of detected faults are successfully diagnosed**
- **All faults detected in fault-free core are diagnosed**

Results: MSWAT Fault Diagnosis (2/2)

- **Diagnosis Latency**
  - **97% diagnosed <10 million cycles (10ms in 1GHz system)**
  - **93% of these were diagnosed in 1 iteration**
    - **Showing the effectiveness of iterative approach**

- **Trace Buffer size**
  - **96% require <200KB/core of loadLog & compareLog**
    - **Trace buffer can easily fit in L2 or L3 cache**

MSWAT Summary and Advantages

- **Detection**
  - **Coverage over 98% with low SDC rate of 0.2%**
- **Diagnosis**
  - **High diagnosability over 95% with low diagnosis latency**
  - **Firmware based replay reduces hw overhead**
  - **Scalable - maximum of 3 replays for any system**
  - **Iterative approach significantly reduces**
    - **Trace buffer size (8MB/core → 400KB/core)**
    - **Diagnosis latency**

Future Work

- Extending this study to server applications
- Off-core faults
- Post-silicon debug and test
  - Use faulty trace as test vector
- Validation on FPGA (w/ Michigan)

Thank you

Backup

Hardware support

- **Detection**
  - **Simple hardware detectors**
  - **Hw support to ensure correct invocation of firmware**

- **Diagnosis**
  - **Small hardware buffer for memory backed trace buffer**
  - **Minor design changes to capture retiring instrns**
  - **Hw checks to prevent trace corruption of good cores**

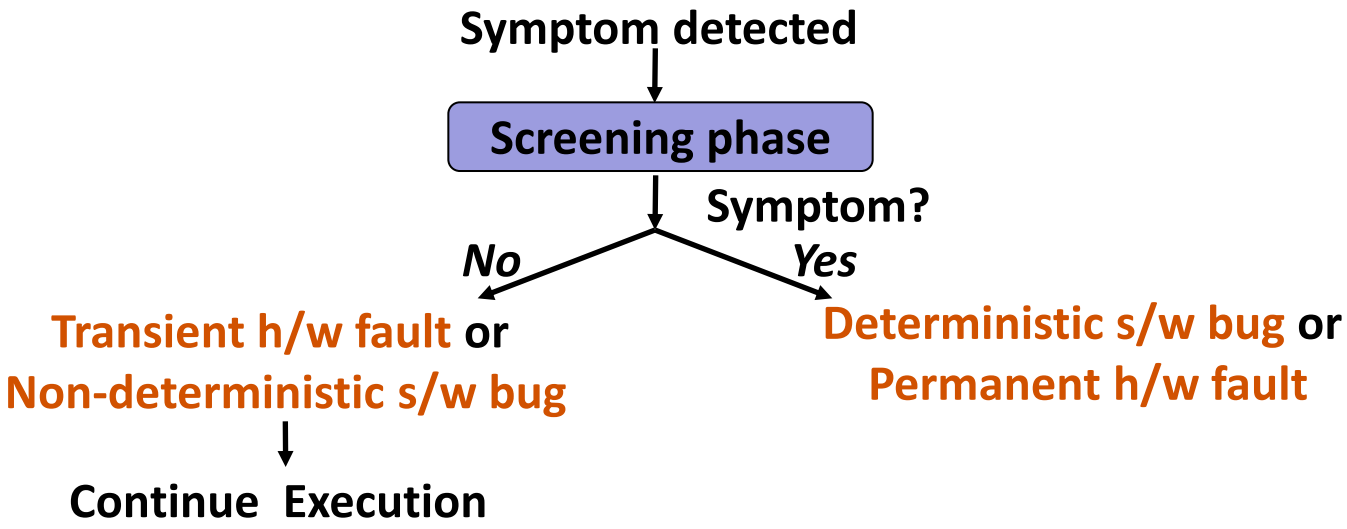
Trace Buffer

- Collect *loadLog* and *compareLog* as a merged trace buffer

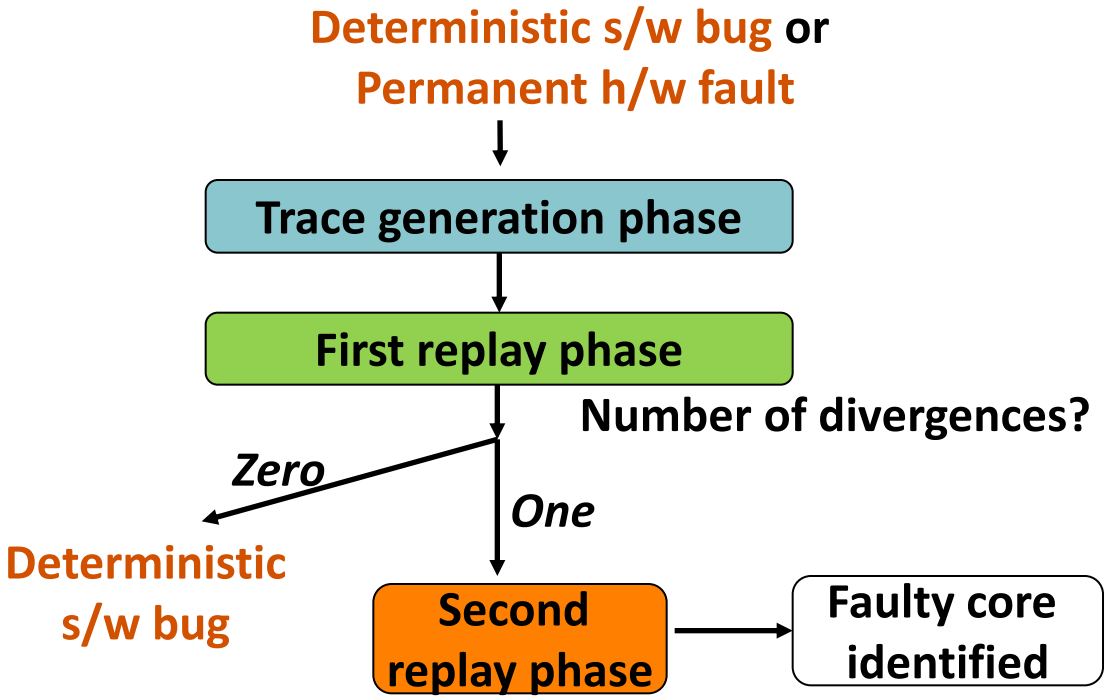
- A small FIFO that is memory backed
  - Minimizes hardware cost
  - Diagnosis can tolerate small performance slack
  - Similar to one used in BugNet and SWAT's TBFD

- Potential problem:
  - Faulty core can corrupt trace buffer of other cores
  - One solution:
    - H/W bounds check – a core writes only to its trace region

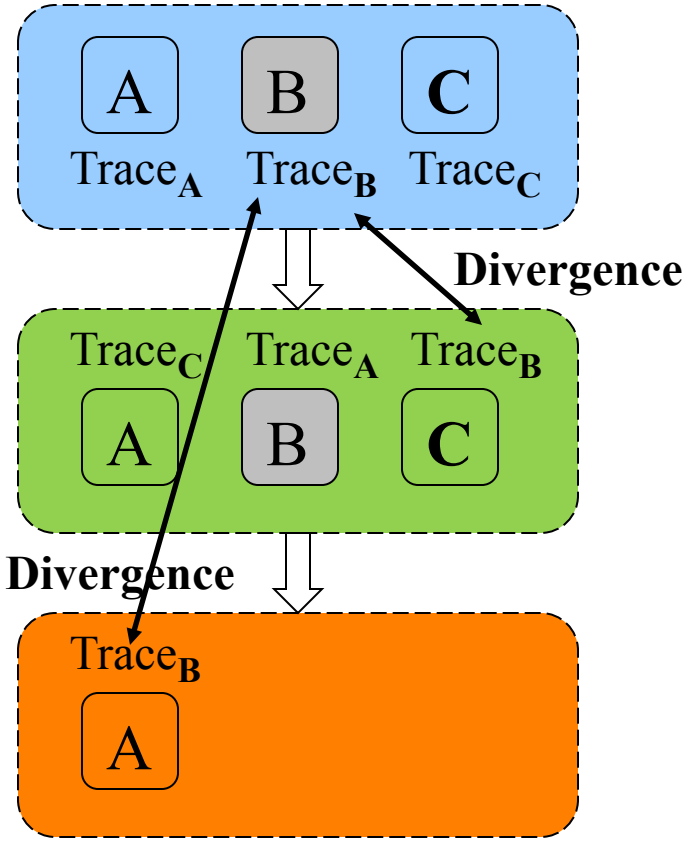
Transient vs. SW Bug vs. Permanent Fault



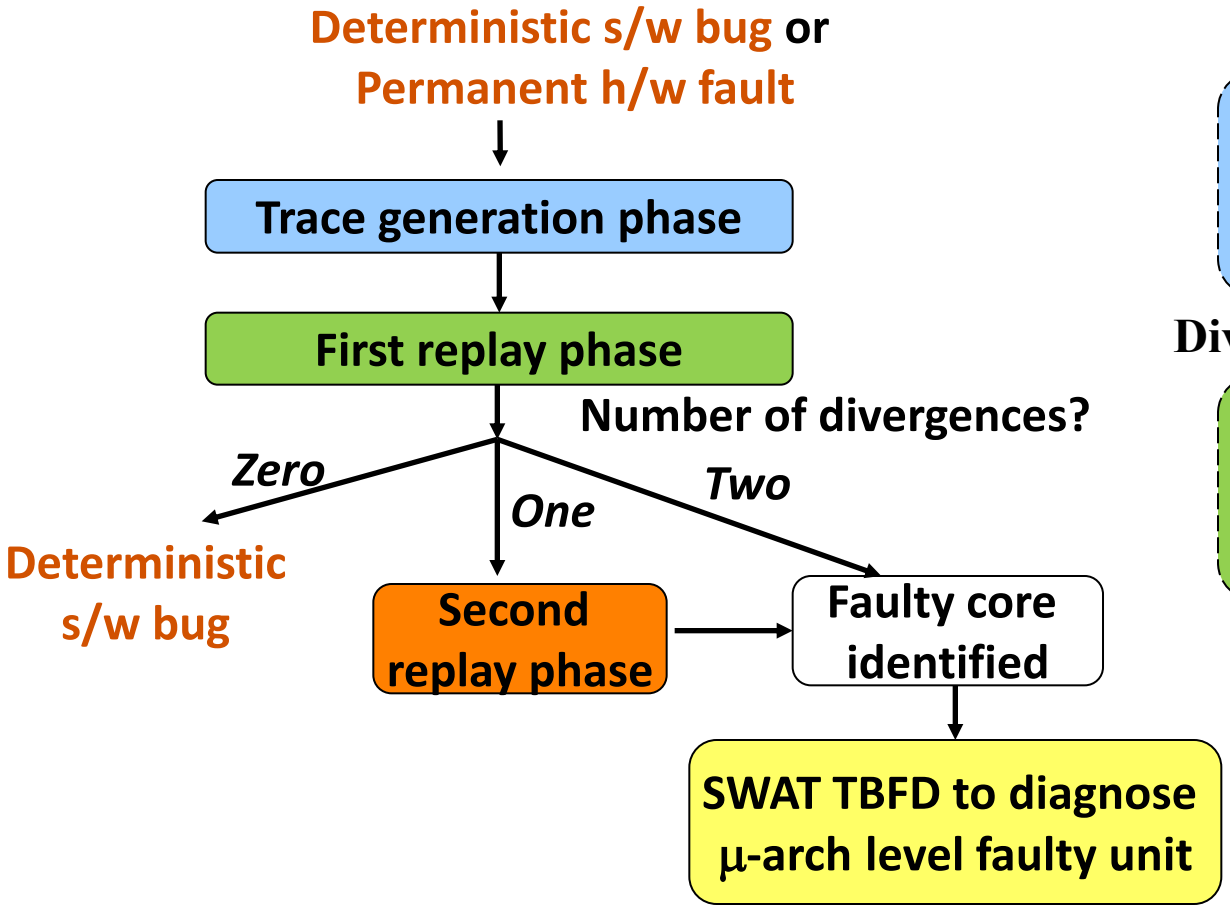
Multicore Fault Diagnosis Algorithm



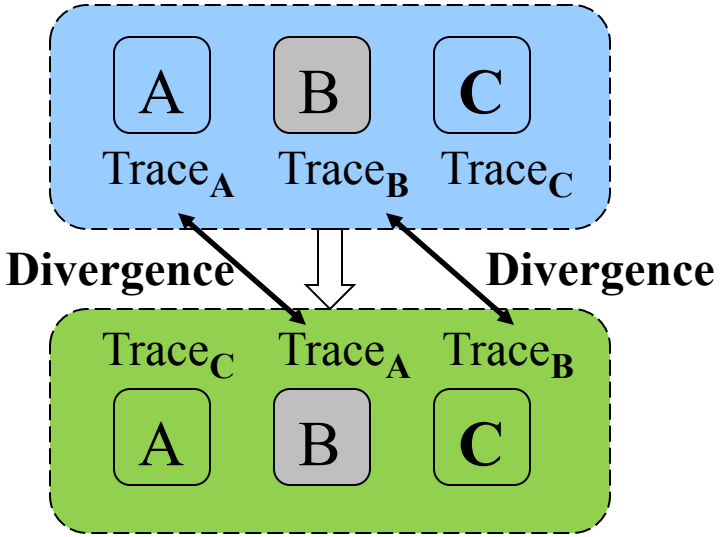
Example: A three core system



Multicore Fault Diagnosis Algorithm



Example: A three core system



Reliability of firmware

- **SWAT philosophy**
  - **Low hw overhead ⇒ firmware based implementation**
- **How to guarantee correct execution of firmware on faulty hw?**
- **Detection**
  - **Hw support ensures correct invocation of firmware**
- **Diagnosis**
  - **Use hw check to not corrupt trace buffers of other cores**
  - **Diagnosis outcome checked by two cores**
    - **Prevents faulty core from subverting the process**