CSCI654 Advanced Computer Architecture

When Software meets Hardware Faults

Hao Han

hhan@cs.wm.edu

7 April 2009

Some slides are adapted from talks of "SWAT"[ASPLOS'08], "SymPIFIED" [DSN'08], "Trace-based diagnosis"[DSN'08], and "Likely program invariants"[DSN'08]

Outline

- Motivation
- Background
- Research points
  - Program verification: SymPLFIED
  - Error detection: SWAT
- Experimental methodology (see report)
- Limitations
- Conclusion

Motivation

- **Goal**: highly reliable systems
- Conventional illusion: fault-free hardware devices to software

⇒ Can not only focus on software bugs of programs

- Hardware faults will happen in the field
  - Traditional solutions: (1) Hardware redundancy (2) special circuits to verify hardware

⇒ Too expensive: area, power, and so on

Today: Re-think about the reliability problem when considering hardware faults, especially in the core

Background - Location of H/W faults

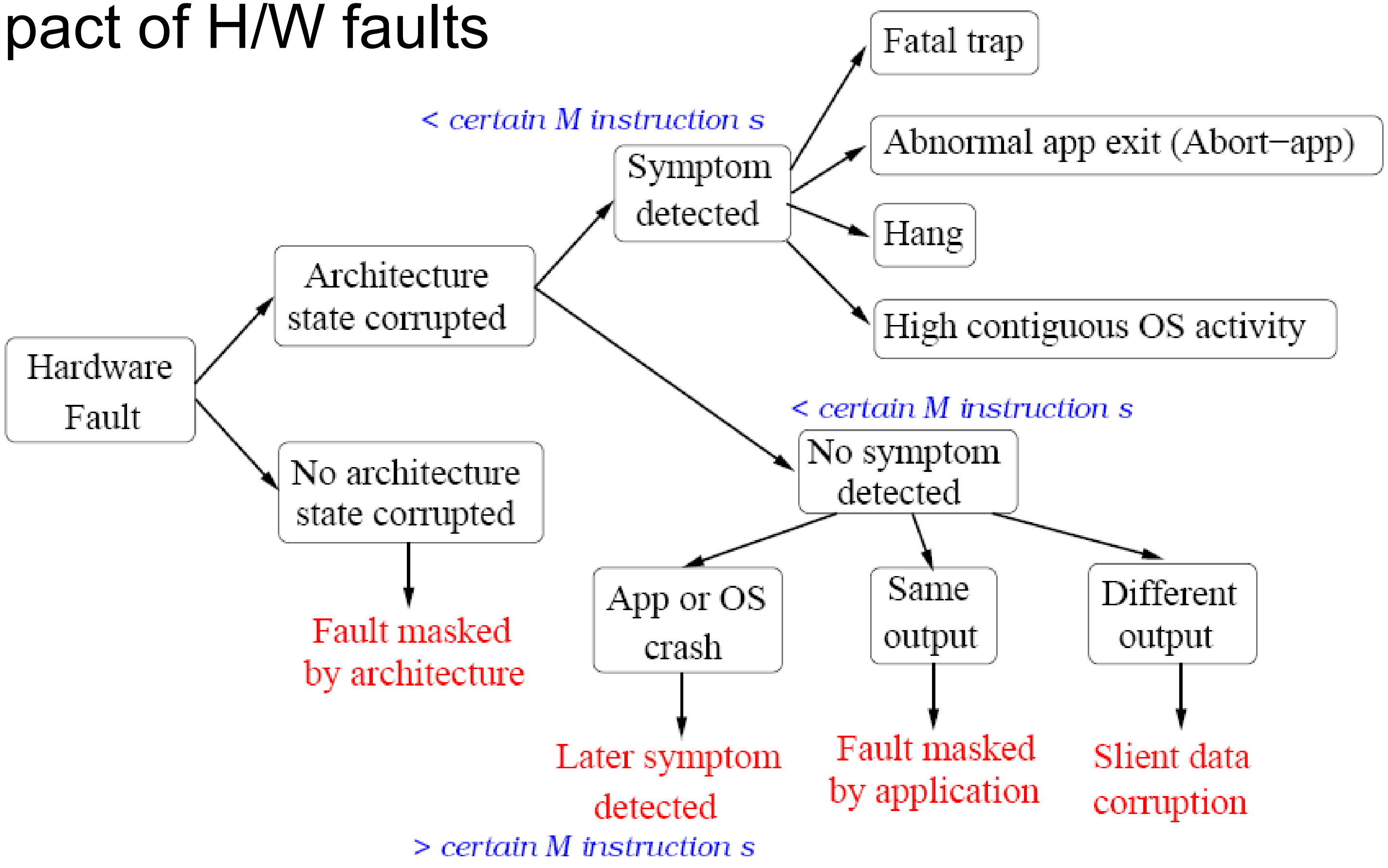
Microarchitectural structure	Faults
Instruction decoder	*Decoding instruction is corrupted*
Integer ALU	*Output latch of one of the ALUs*
FP ALU	*Output latch of one of the ALUs*
Address or data bus	*Bus of register, cache, memory*
Physical reg file	*Physical regs in the reg file*
Reorder buffer (ROB)	*Src/dest reg of instr in ROB entry*
Address gen unit (AGEN)	*Virtual address generated by the unit*
Register alias table (RAT)	*Logical -> phys map of a logical reg*

Background - Hardware Faults

- Category of H/W faults:

  (1) permanent (2) transient (3) intermittent

- Impact of H/W faults


Hardware Fault
Architecture state corrupted
No architecture state corrupted
Fault masked by architecture
< certain M instruction s
Symptom detected
Fatal trap
Abnormal app exit (Abort−app)
Hang
High contiguous OS activity
< certain M instruction s
No symptom detected
App or OS crash
Same output
Different output
Later symptom detected
> certain M instruction s
Fault masked by application
Slient data corruption

Research Points

- Program verification under hardware faults

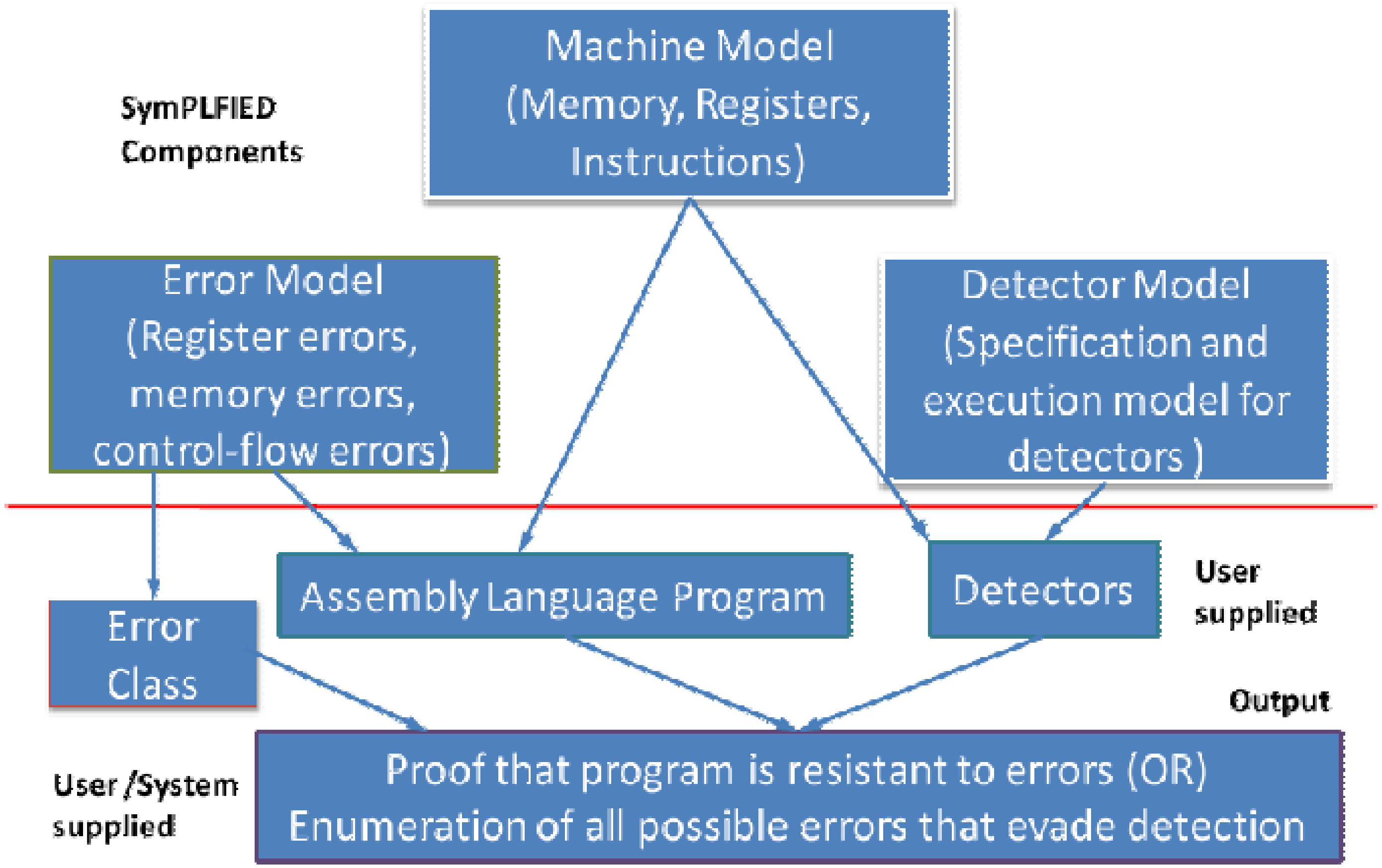
  SymPLFIED [DSN'08] (Best paper award)

- Error detection for hardware faults with low cost

SWAT
- SWAT [ASPLOS '08]
- Trace-Based Fault Diagnosis [DSN'08]
- Likely Program Invariants [DSN'08]
- Accurate Fault Models [HPCA'09]

SymPLFIED [DSN'08]

Goal: A formal framework to evaluate the effects of hardware faults on arbitrary programs independent of the detection mechanism


SymPLFIED Components
Machine Model (Memory, Registers, Instructions)
Error Model (Register errors, memory errors, control-flow errors)
Detector Model (Specification and execution model for detectors)
Error Class
Assembly Language Program
Detectors
User supplied
Output
User /System supplied
Proof that program is resistant to errors (OR) Enumeration of all possible errors that evade detection

Conceptual Design Flow of SymPLFIED

Techniques of SymPFLIED

- Model error propagation by representingl errors in program as abstract symbol

  <symbolic execution>

  – Represents all kinds of faults

  – Avoids explosion of exhaustive fault injection

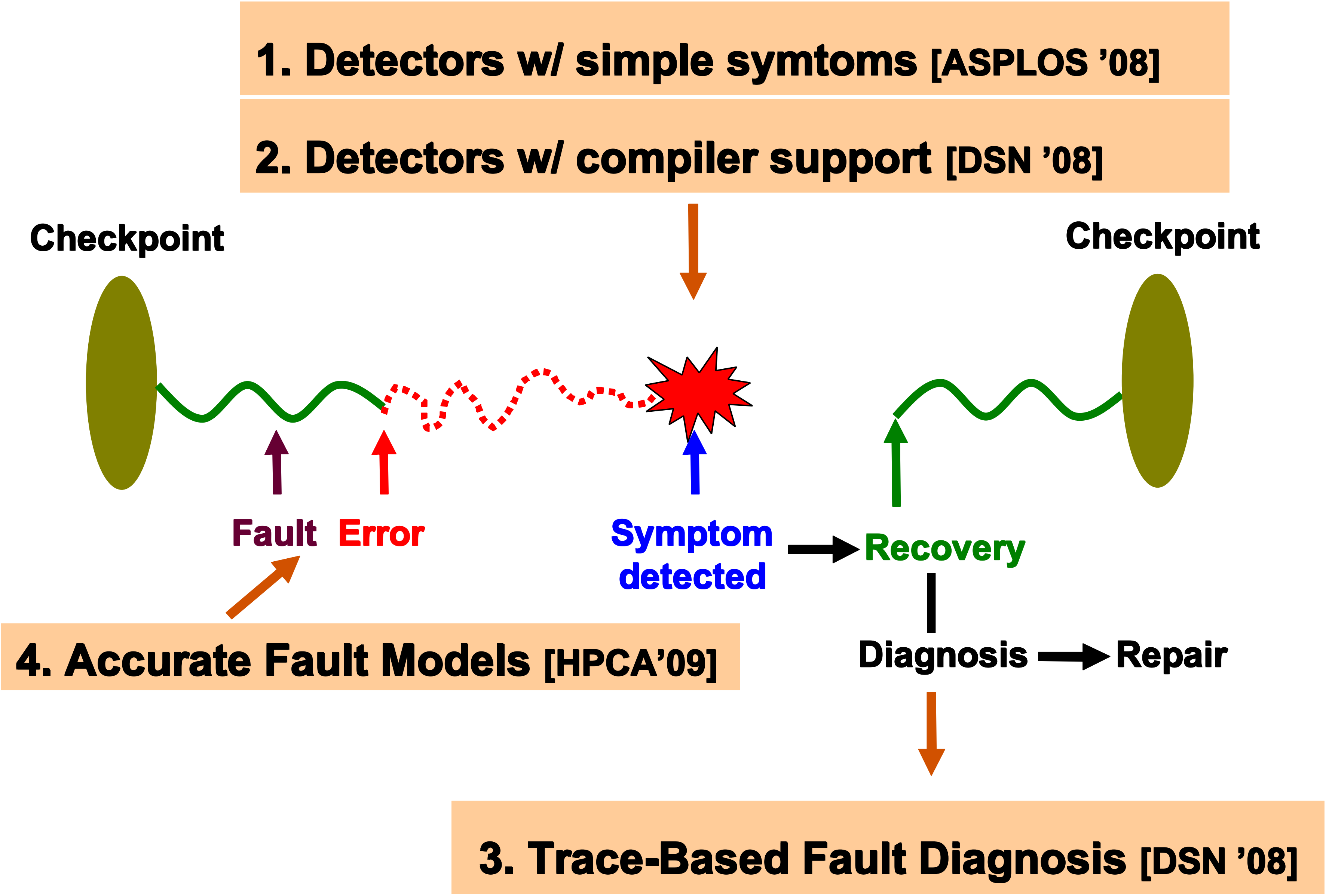
- Automatically search possible values of symoblic error that escape from detecion and cause SDC

  <model checking>

  – Bounded model checking using satisfiability solving

SWAT System

- Assumptions:
  - Multicore system where a fault-free core is always available
  - Checkpoint/rollback mechanism
- Goals:
  - Provide low-cost software-level detection methods for permanent hardware fault, and low-level diagnosis for recovery and possibly repair/reconfiguration
- SWAT components
  - Detection: Symptoms of software for detecting
  - Diagnosis: Identify the source of faulty unit



1. **Detectors w/ simple symtoms [ASPLOS '08]**

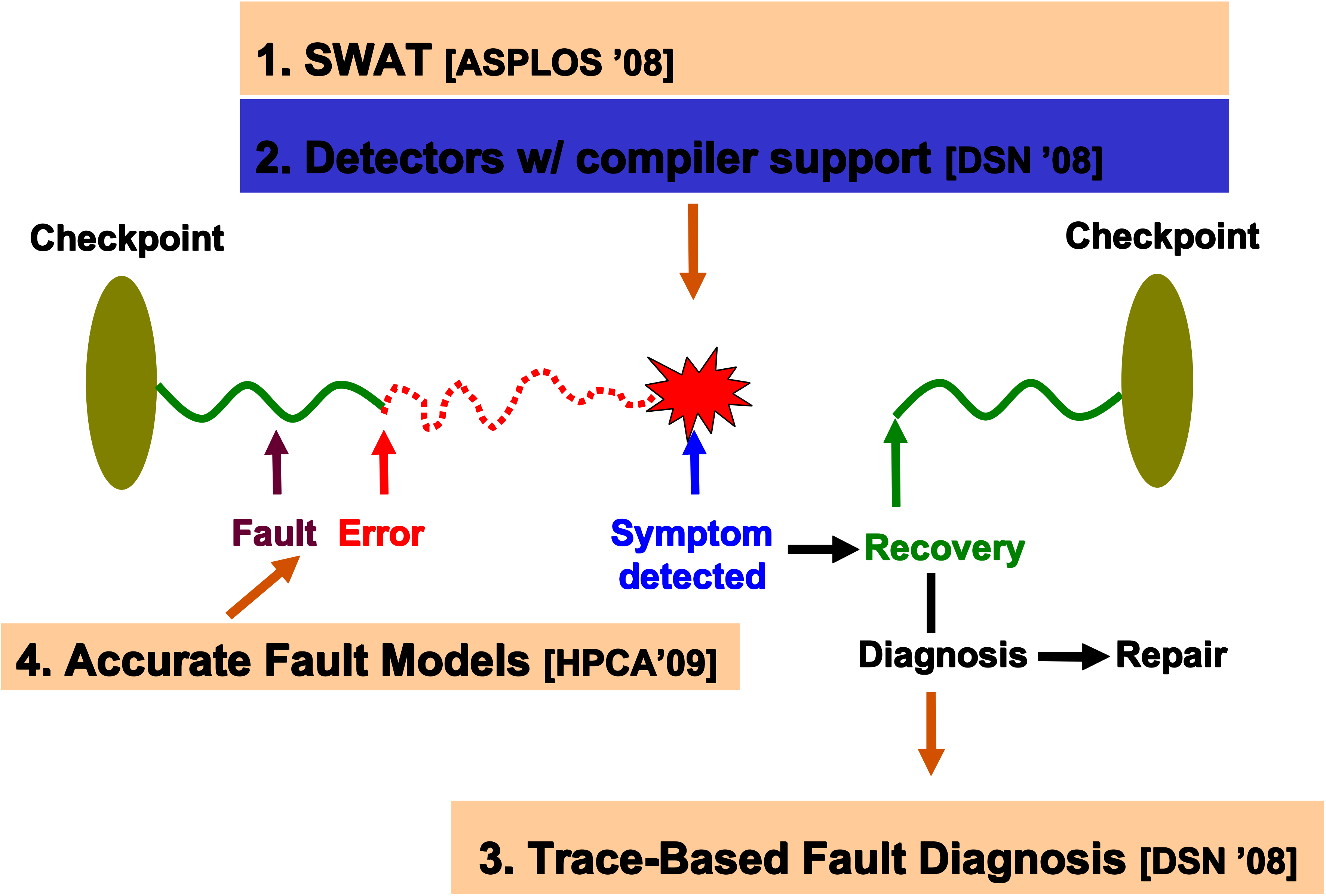
2. **Detectors w/ compiler support [DSN '08]**

4. **Accurate Fault Models [HPCA'09]**

3. **Trace-Based Fault Diagnosis [DSN '08]**

Simple Symptoms

- Observe anomalous symptoms for fault detection
  - Incur low overheads for “always-on” detectors
  - Minimal support from hardware, no software support
- Anomalous symptoms
  - **Fatal hardware traps**
    - For example, division by zero, RED State, etc.
  - **Abnormal application exit**, indicated by OS
    - For example, application terminates due to segmentation fault
  - **Hangs**
    - The whole system becomes unresponsive
    - Detected by setting up counter
  - **High OS activity**
    - Monitoring the amount of time the execution remains in the OS, without returning to the application



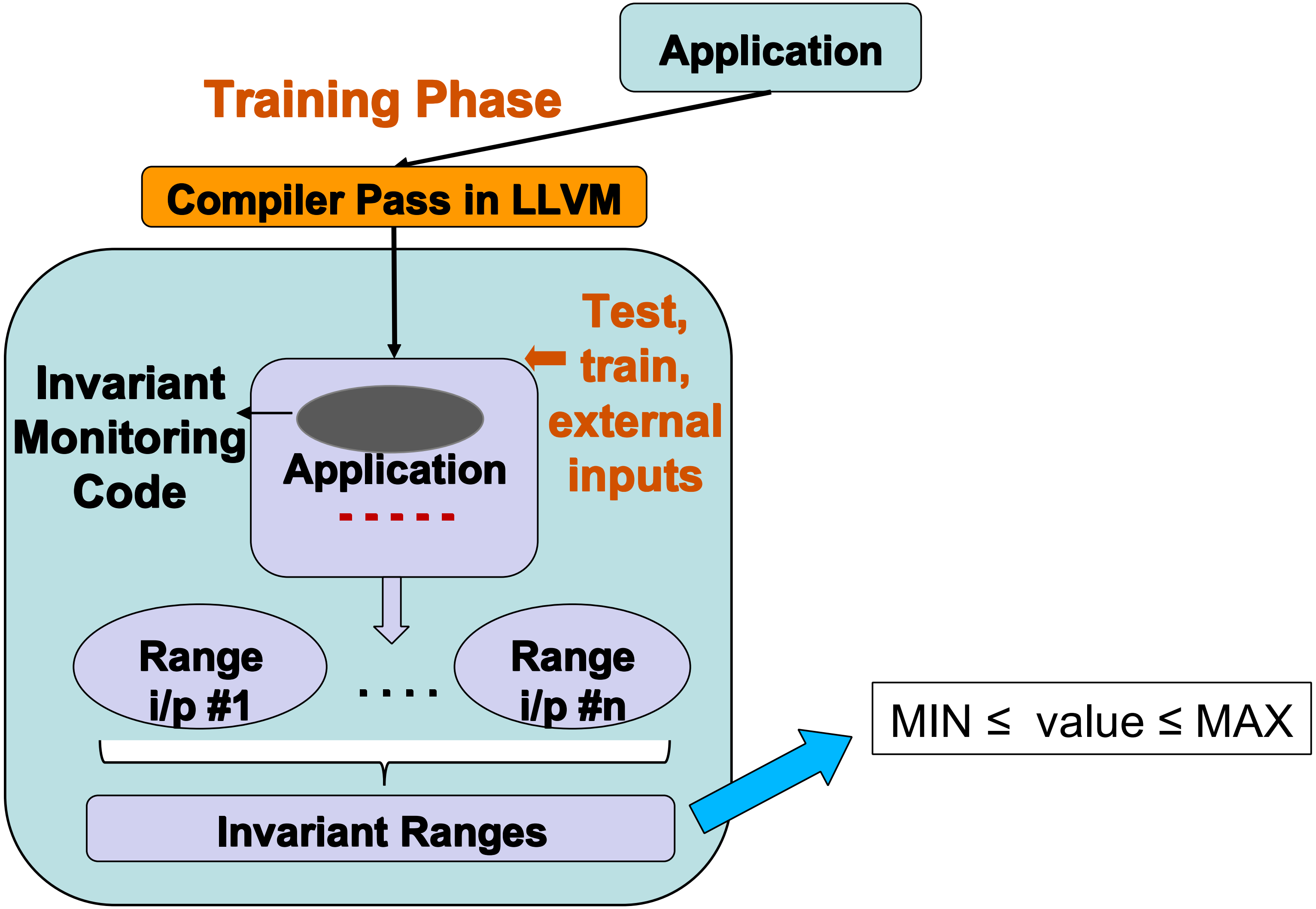
1. **SWAT** [ASPLOS '08]

2. **Detectors w/ compiler support** [DSN '08]

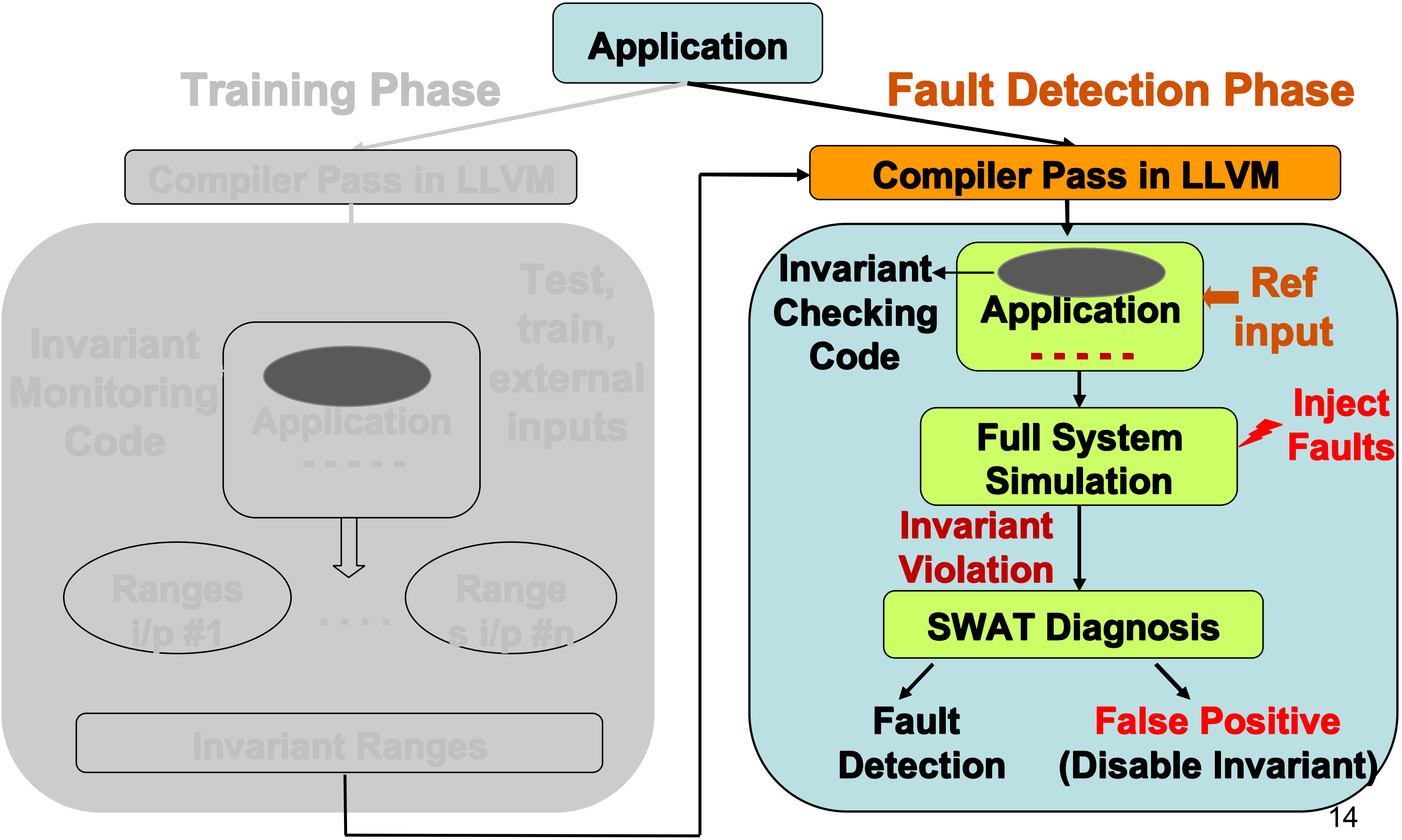
4. **Accurate Fault Models** [HPCA'09]

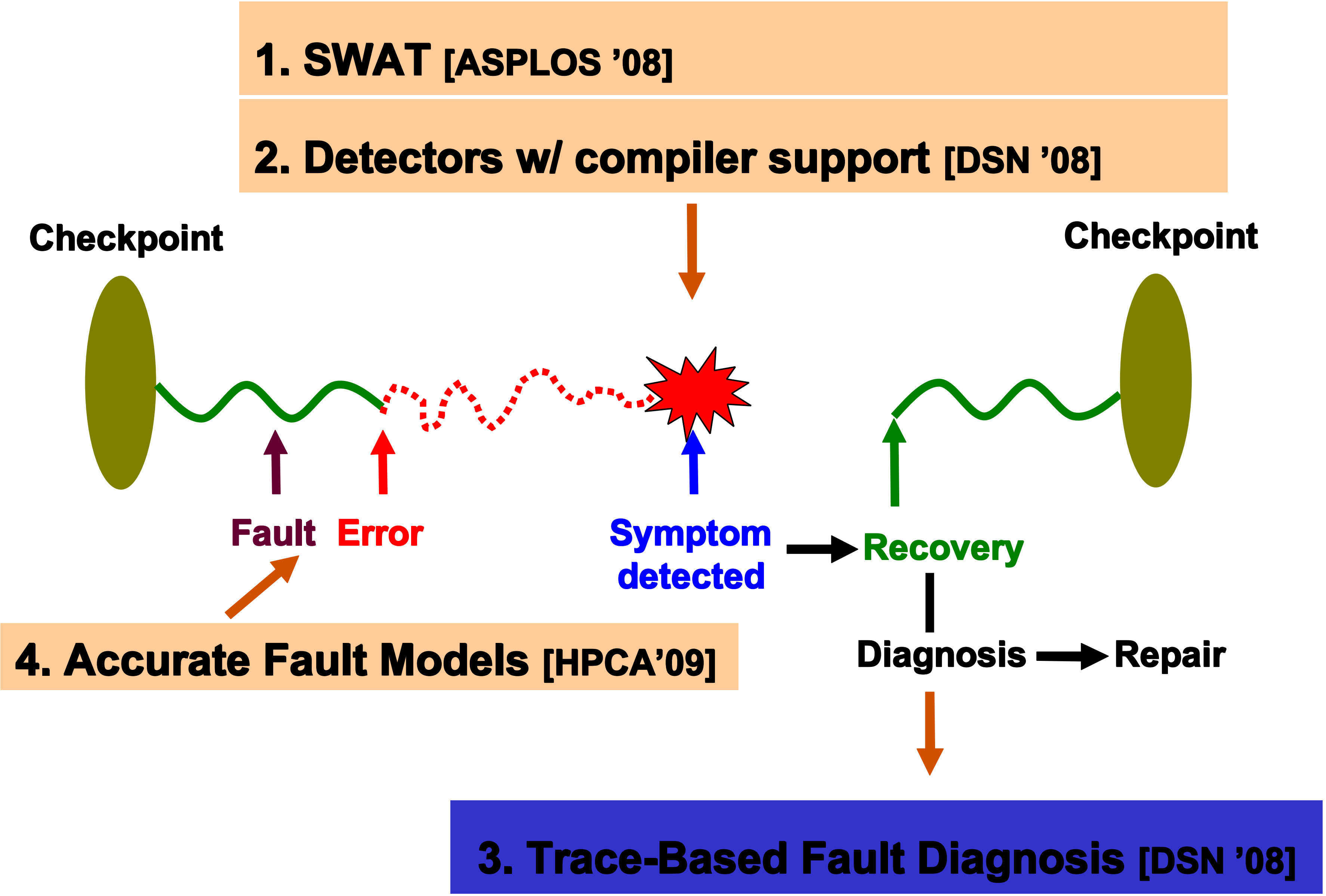
3. **Trace-Based Fault Diagnosis** [DSN '08]

Likely Program Invariant



Likely Program Invariant





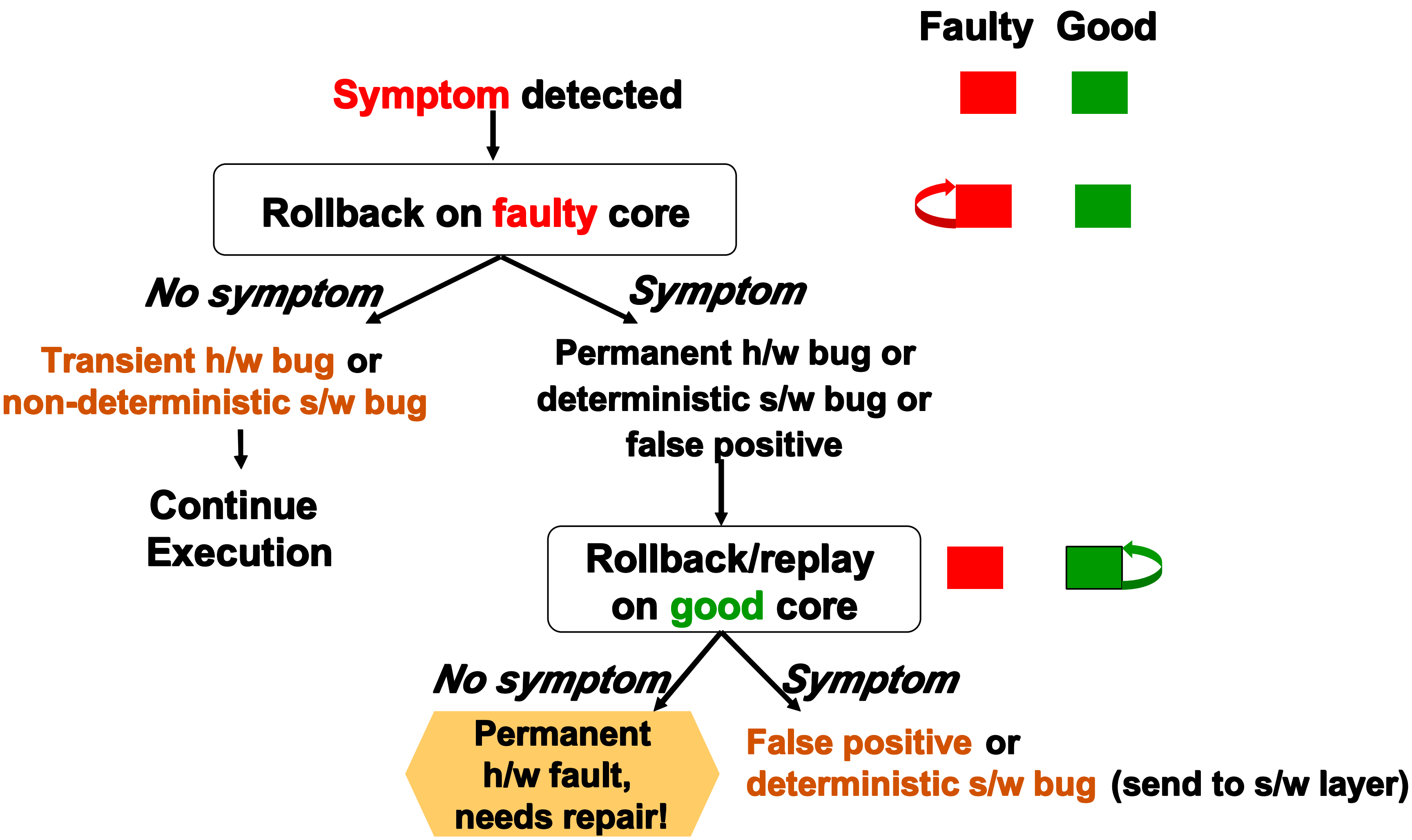
1. SWAT [ASPLOS '08]

2. Detectors w/ compiler support [DSN '08]

3. Trace-Based Fault Diagnosis [DSN '08]

4. Accurate Fault Models [HPCA'09]

Diagnosis: first step



Faulty Good

🟥 🟩

Symptom detected

Rollback on faulty core

No symptom

Transient h/w bug or non-deterministic s/w bug

Continue Execution

Symptom

Permanent h/w bug or deterministic s/w bug or false positive

Rollback/replay on good core

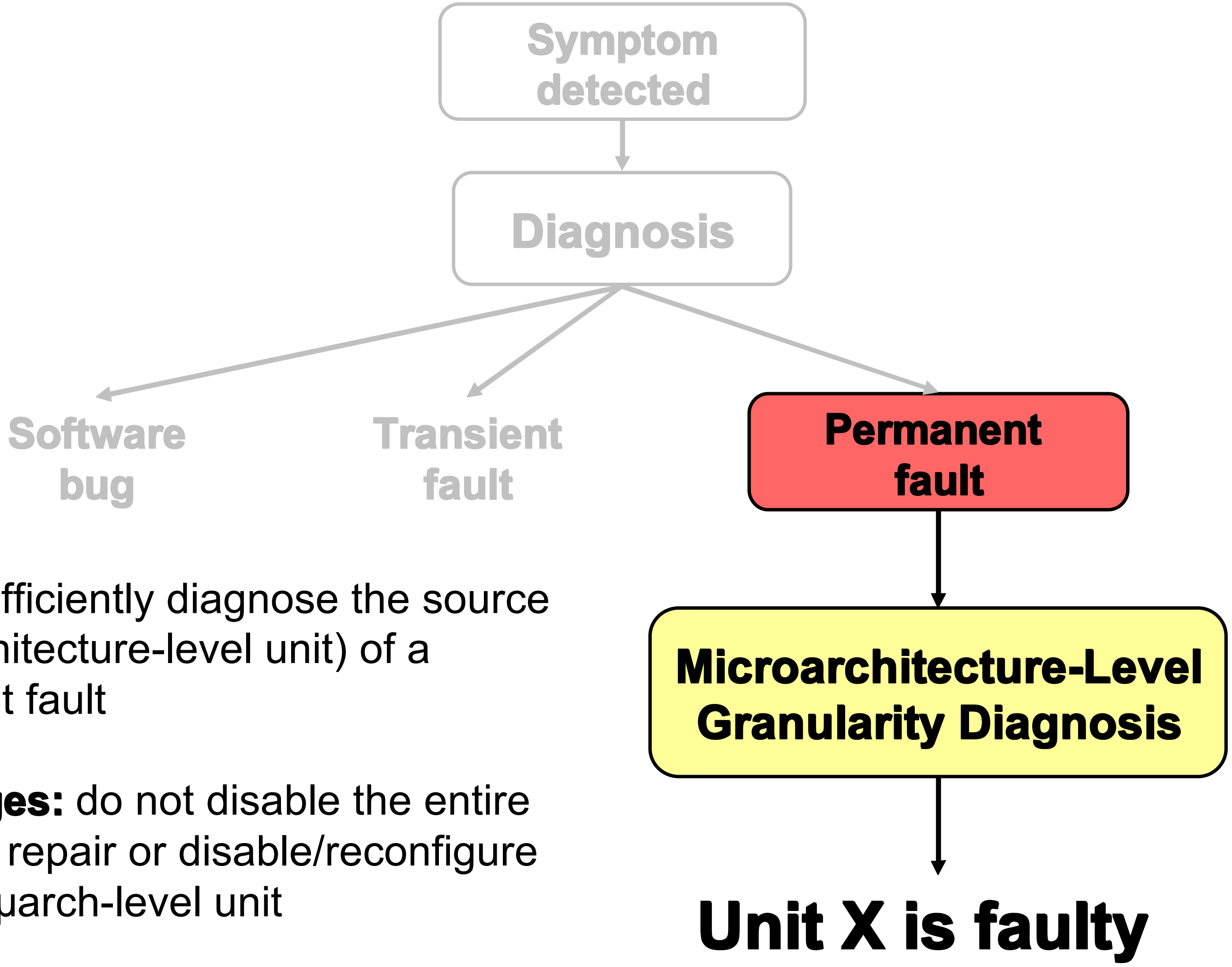
No symptom

Permanent h/w fault, needs repair!

Symptom

False positive or deterministic s/w bug (send to s/w layer)

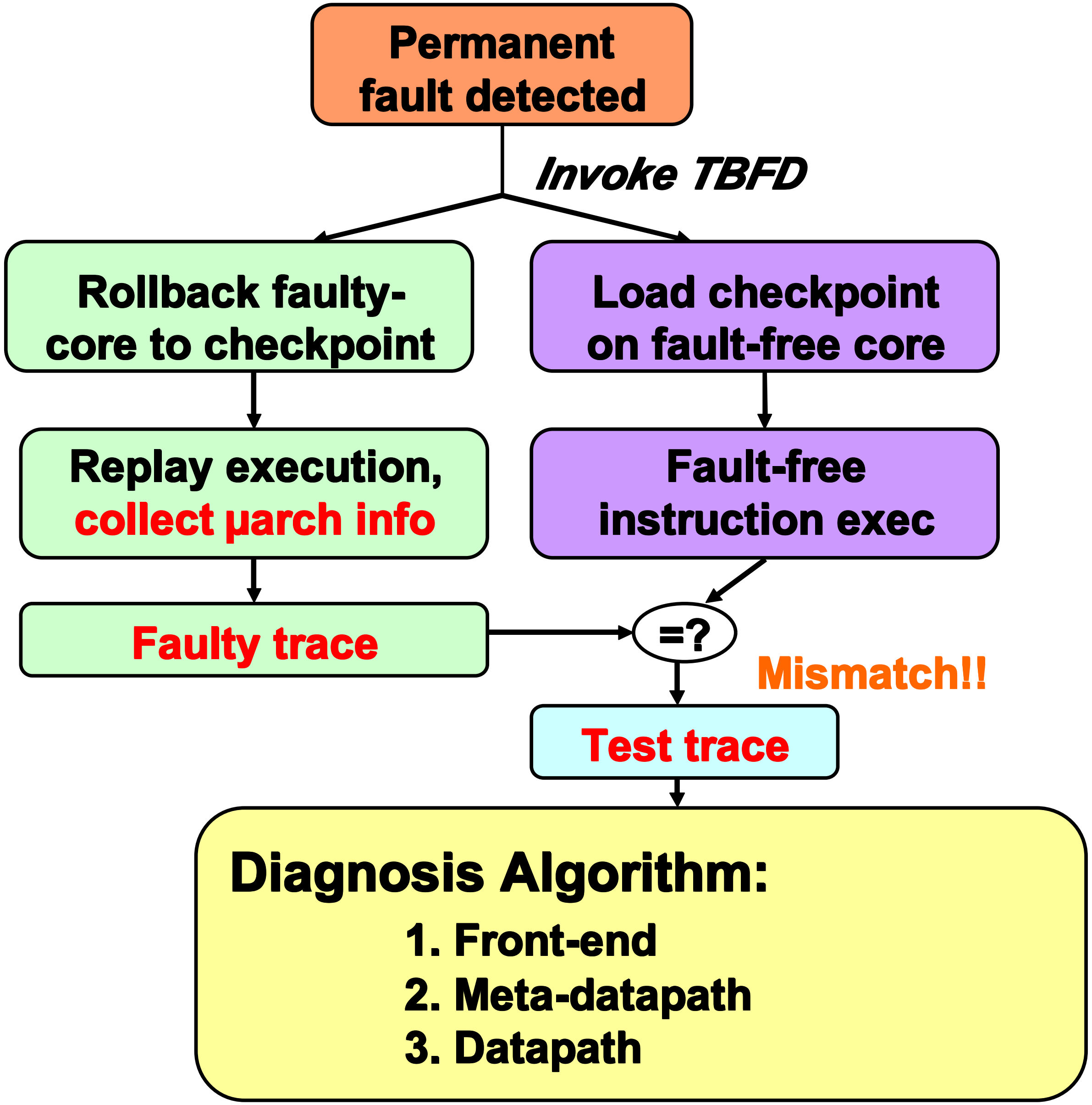
Diagnosis: second step



Goal: to efficiently diagnose the source (microarchitecture-level unit) of a permanent fault

Advantages: do not disable the entire core, only repair or disable/reconfigure the faulty µarch-level unit

Trace-Based Fault Diagnosis (TBFD)


Permanent fault detected
Invoke TBFD
Rollback faulty-core to checkpoint
Load checkpoint on fault-free core
Replay execution, collect μarch info
Fault-free instruction exec
Faulty trace
=?
Mismatch!!
Test trace
Diagnosis Algorithm:
1. Front-end
2. Meta-datapath
3. Datapath

- Faults in front-end is related to Instruction Decoder;
- Fault in meta-datapath indicates faults in ROB or RAT;
- Faults in datapath is related to ALU, data bus, and register file.

Limitations

- Do not consider the off-core faults, such as faults in crossbar
- Most work only considers single error for simplicity, but in practice hardware faults can be multi-types and multi-sources
- Pure software level detection has inherent shortcomings, hybrid method (combining hardware and software) may be a better choice
- SWAT is passive scheme, need more aggressive detection method

...

Conclusion

- Verifying program and detecting hardware faults are vital for reliable system
- For SymPLFIED
  - ✓ Verify programs automatically with symbolic execution and model checking
- For SWAT
  - ✓ High-level detection, low-level diagnosis
  - ✓ Treats hardware faults as software bugs
  - ✓ Handles all faults that matter, and oblivious to masked faults