# Design for Test and Resilience

Jacob Abraham

EE 382M.8 VLSI-II

Spring 2017

February 21, 2017

# Topics

- Design for test
- Built-in self test
- Testing mixed-signal components
- Silicon debug
- Fault tolerance and Resilience
- Effect of soft errors
- Evaluating system resiliency – fault/error injection
- Control-flow checking
- Cross-layer resilience
- Application-level resilience

# Dealing with Hardware Defects

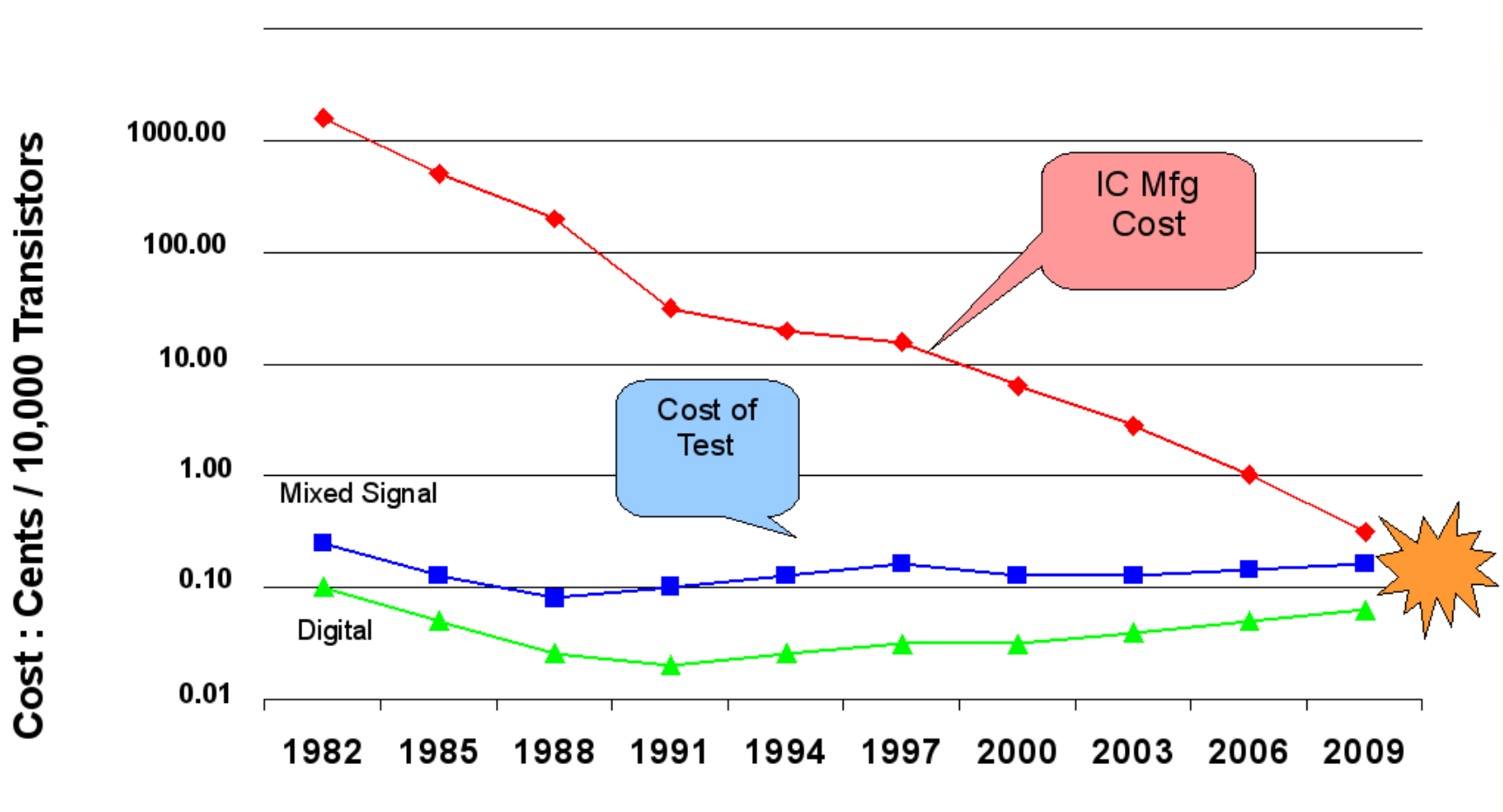
## Defects introduced during manufacturing

- Screen defective parts with *tests* at the end of the manufacturing process
- Additional circuitry added during design to facilitate test *Design for Test (DfT)*
- Need to be able to detect "hard" defects (shorts, opens, etc.) as well as "soft" defects (which may result in only timing errors)

## Defects occurring during operation

- Design system to tolerate the incorrect results or to be "resilient"
- Metal migration, threshold shifts (due to NBTI), etc.
- Bit-flips (soft errors) (due to cosmic radiation)
- Incorrect logic value sampled (due to timing problems)

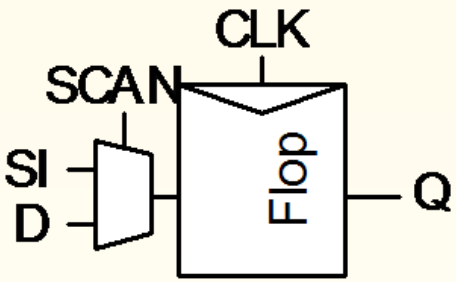
# Manufacturing Test Costs

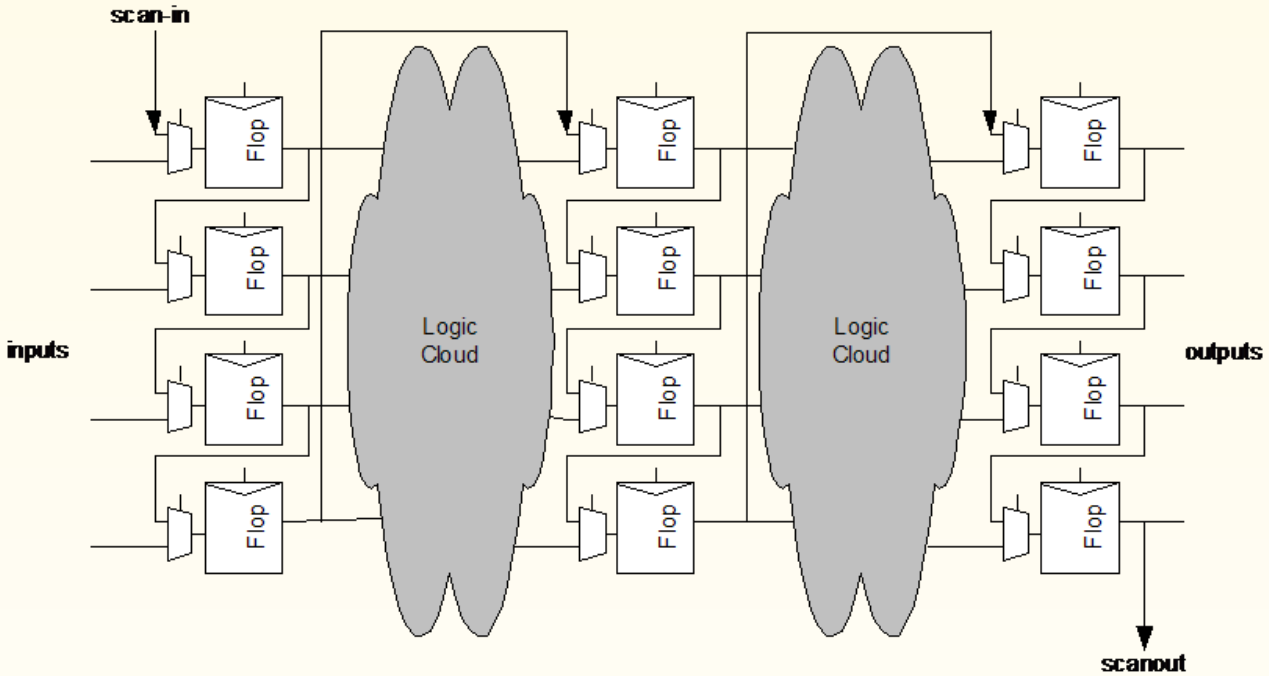



Why doesn't the cost of testing a transistor scale like the cost of manufacturing the transistor?

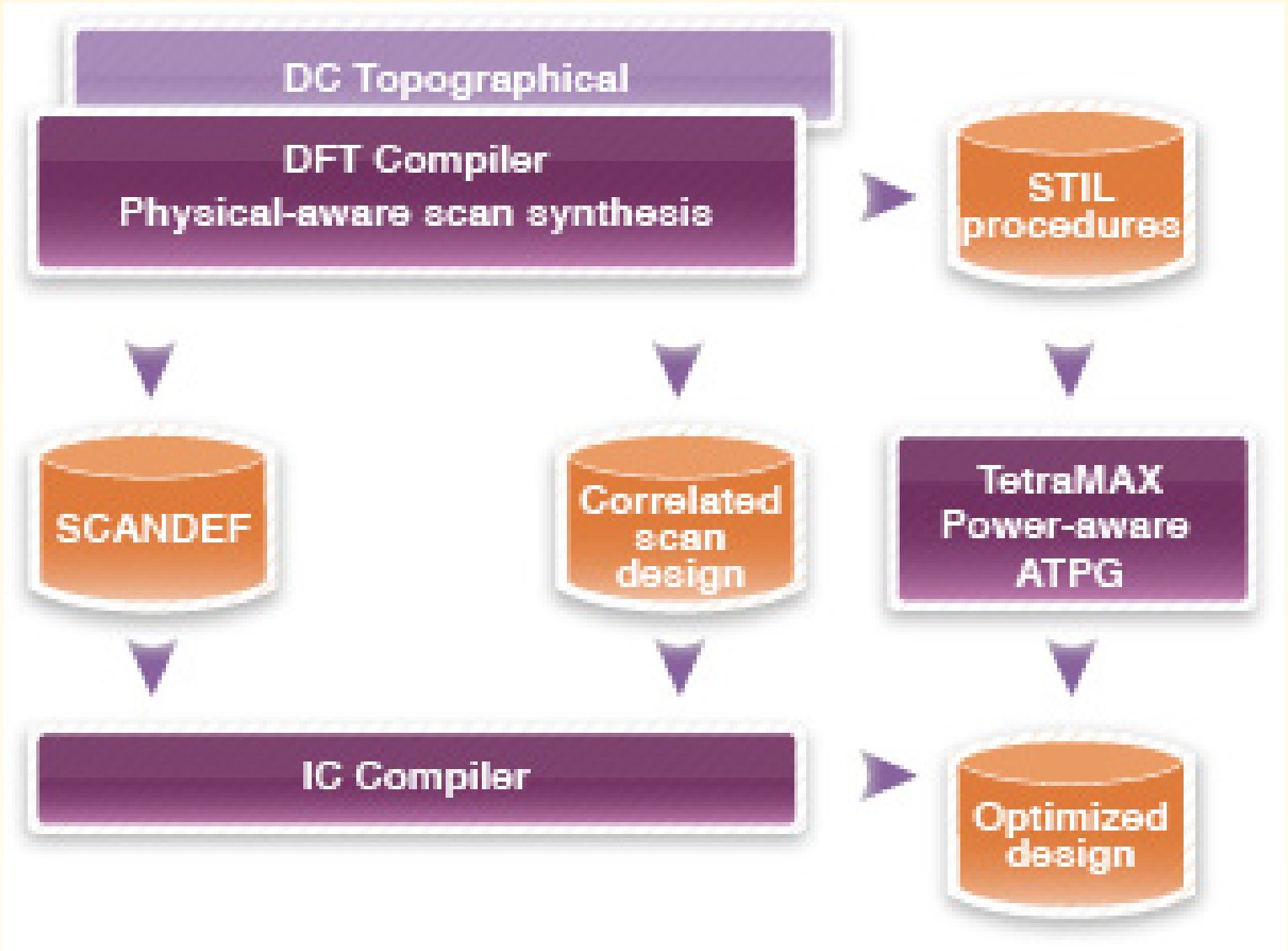
# Scan Chains

- Convert each flip-flop to a scan register
  - Only costs one extra multiplexer
- Normal mode: flip-flops behave as usual
- Scan mode: flip-flops behave as shift register
- Contents of flops can be scanned out and new values scanned in



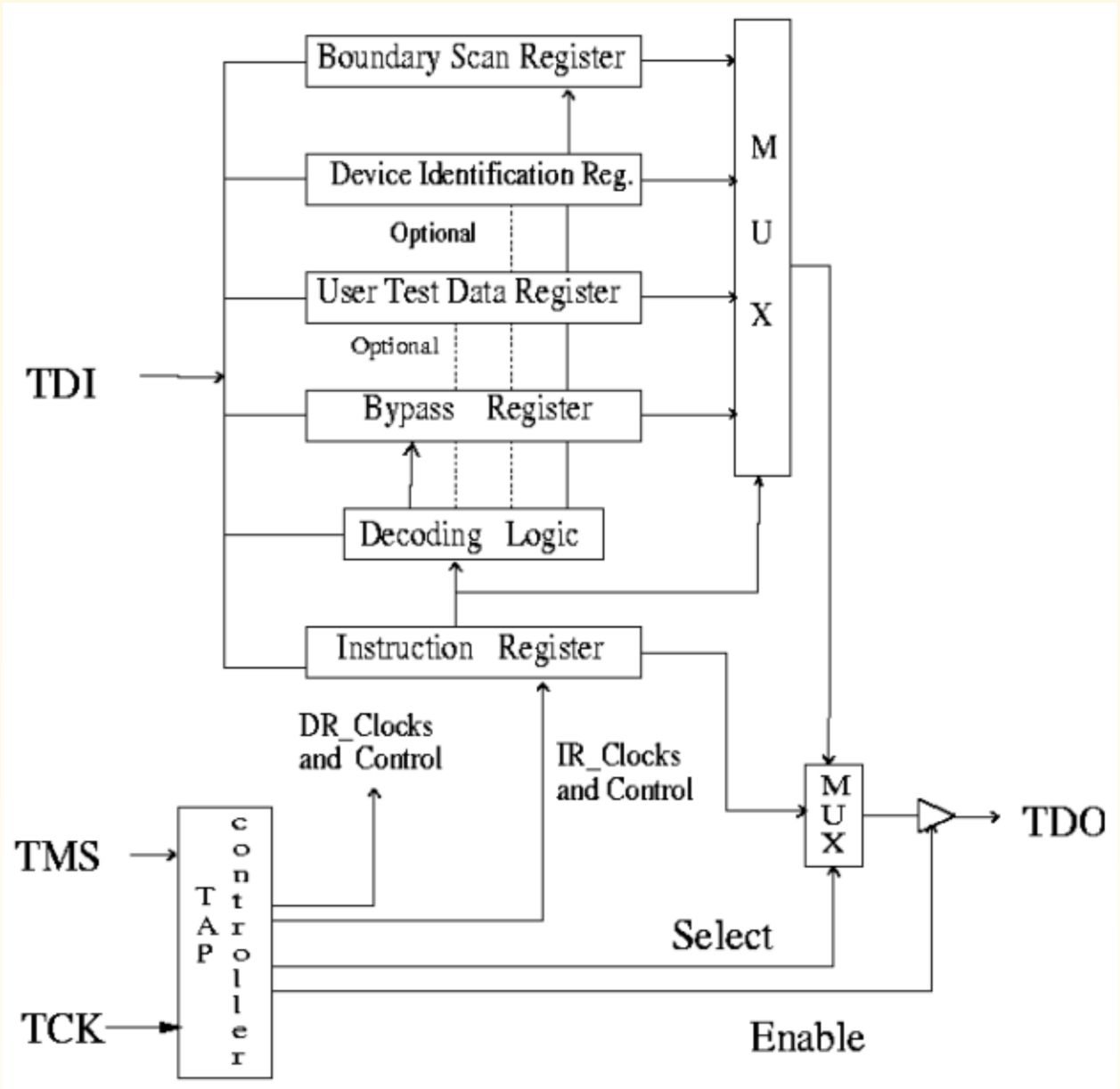



# Synthesis of Scan Logic – DFT Compiler




Source: Synopsys

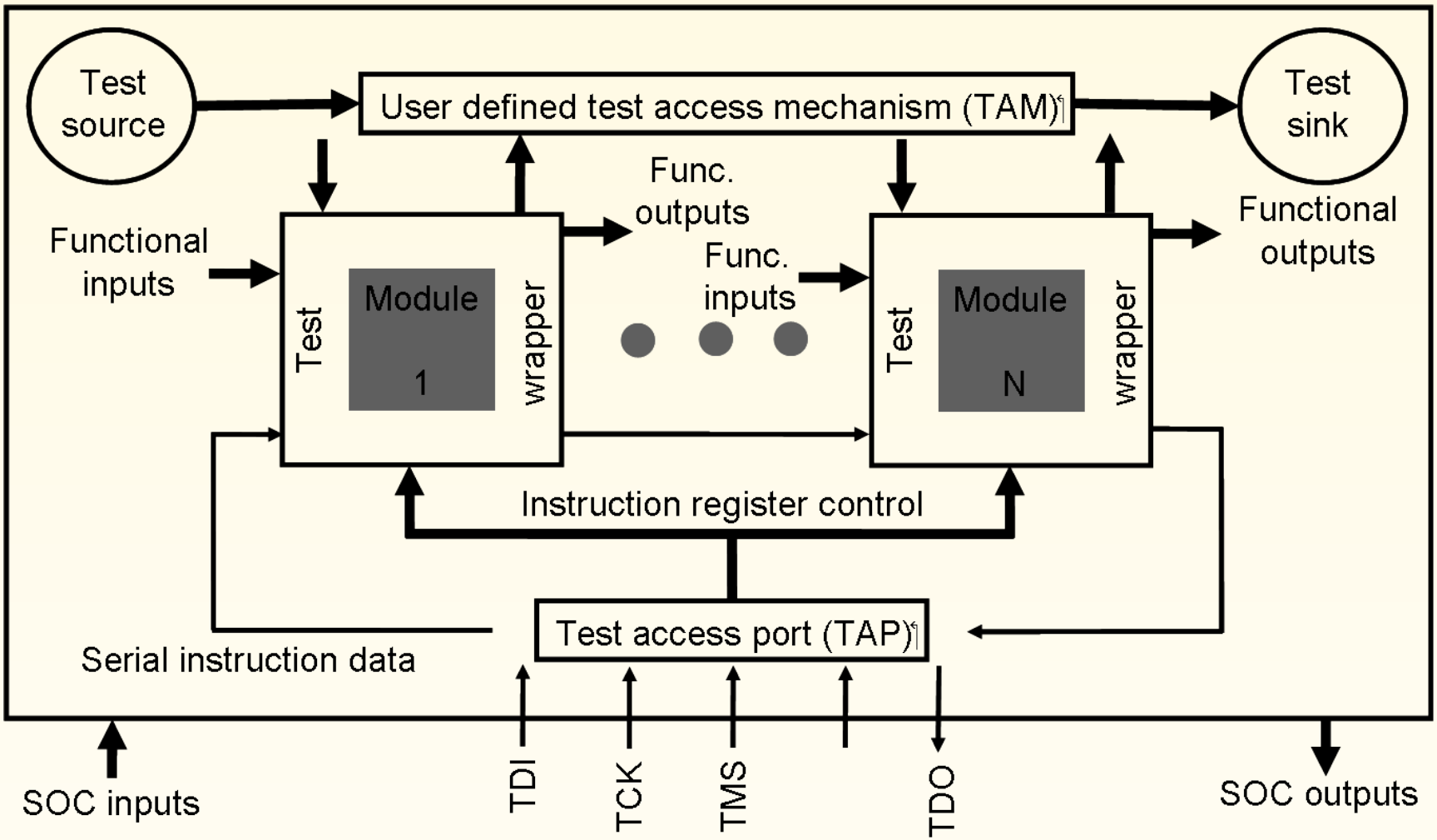
# Boundary Scan (IEEE 1149.1)



# Boundary Scan Interface

- Boundary scan is accessed through five pins
  - TCK: test clock
  - TMS: test mode select
  - TDI: test data in
  - TDO: test data out
  - TRST*: test reset (optional)
- Chips with internal scan chains can access the chains through boundary scan for unified test strategy

# DFT Architecture for SoCs



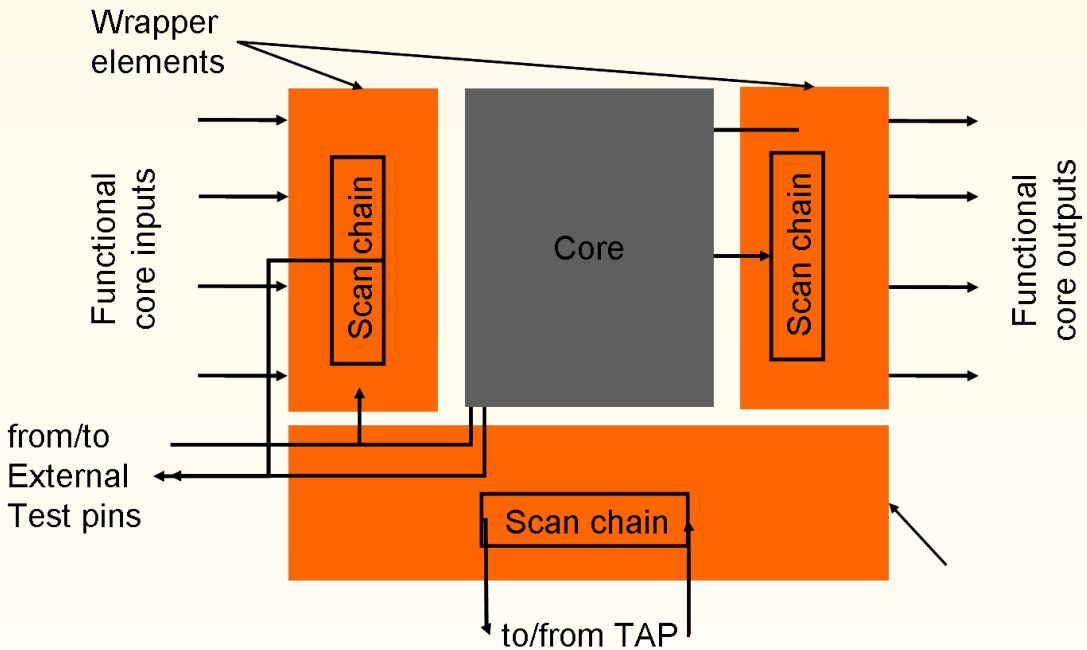

Source: Bushnell and Agarwal

# Test Wrapper for an Embedded Core

Logic added around embedded core to provide test access

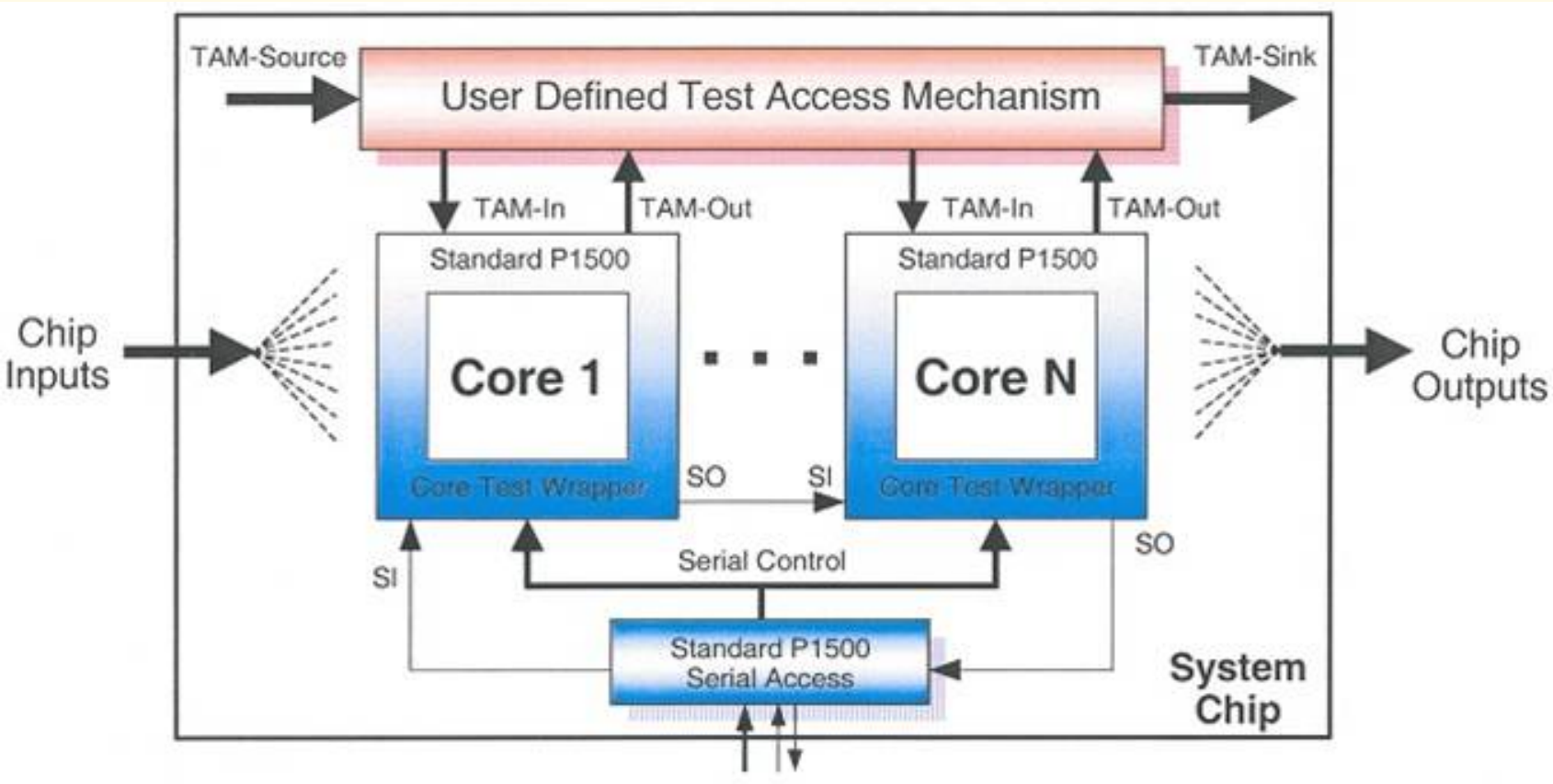
## Test wrapper modes for core terminals

- For each input terminal – external test mode for interconnect test, internal test mode for testing logic inside core
- For each output terminal – normal mode, as well as external test mode for interconnect test, internal test mode to observe core outputs



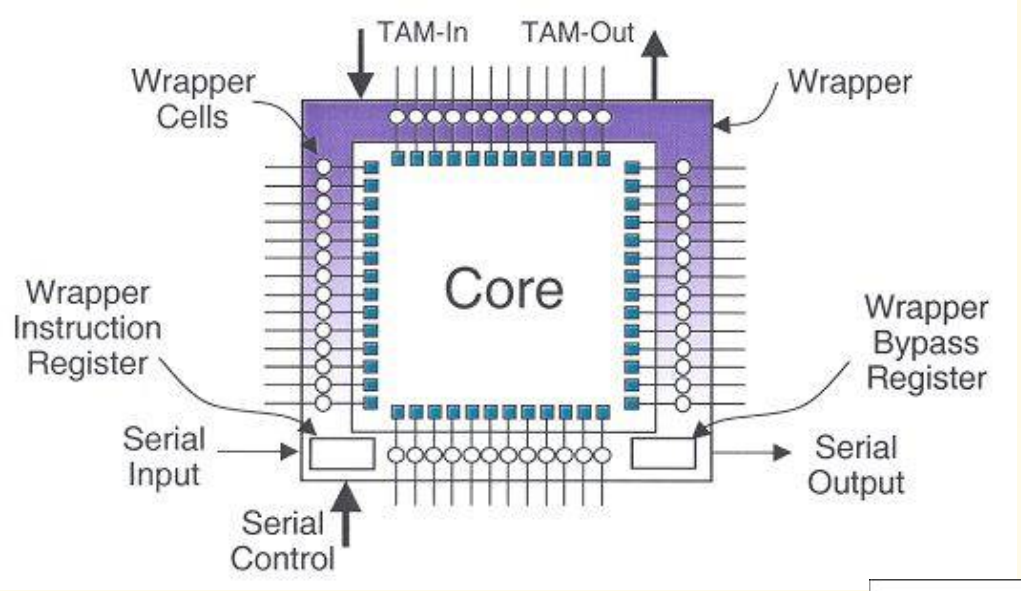
# IEEE Standard P1500

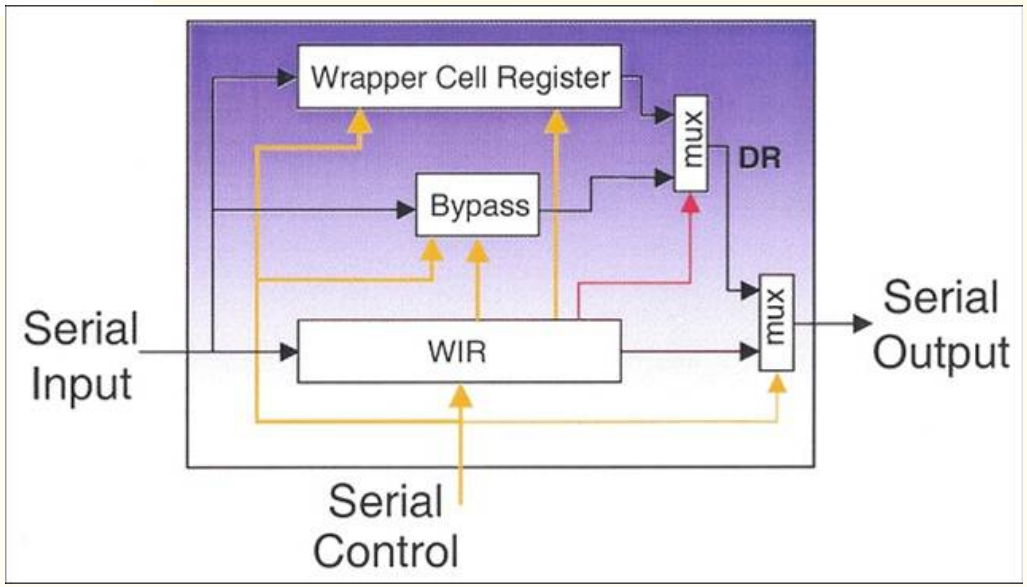
Core test interface between embedded core and system chip, supports test reuse, testability of system interconnect and logic
Improves efficiency of test between core users and core providers



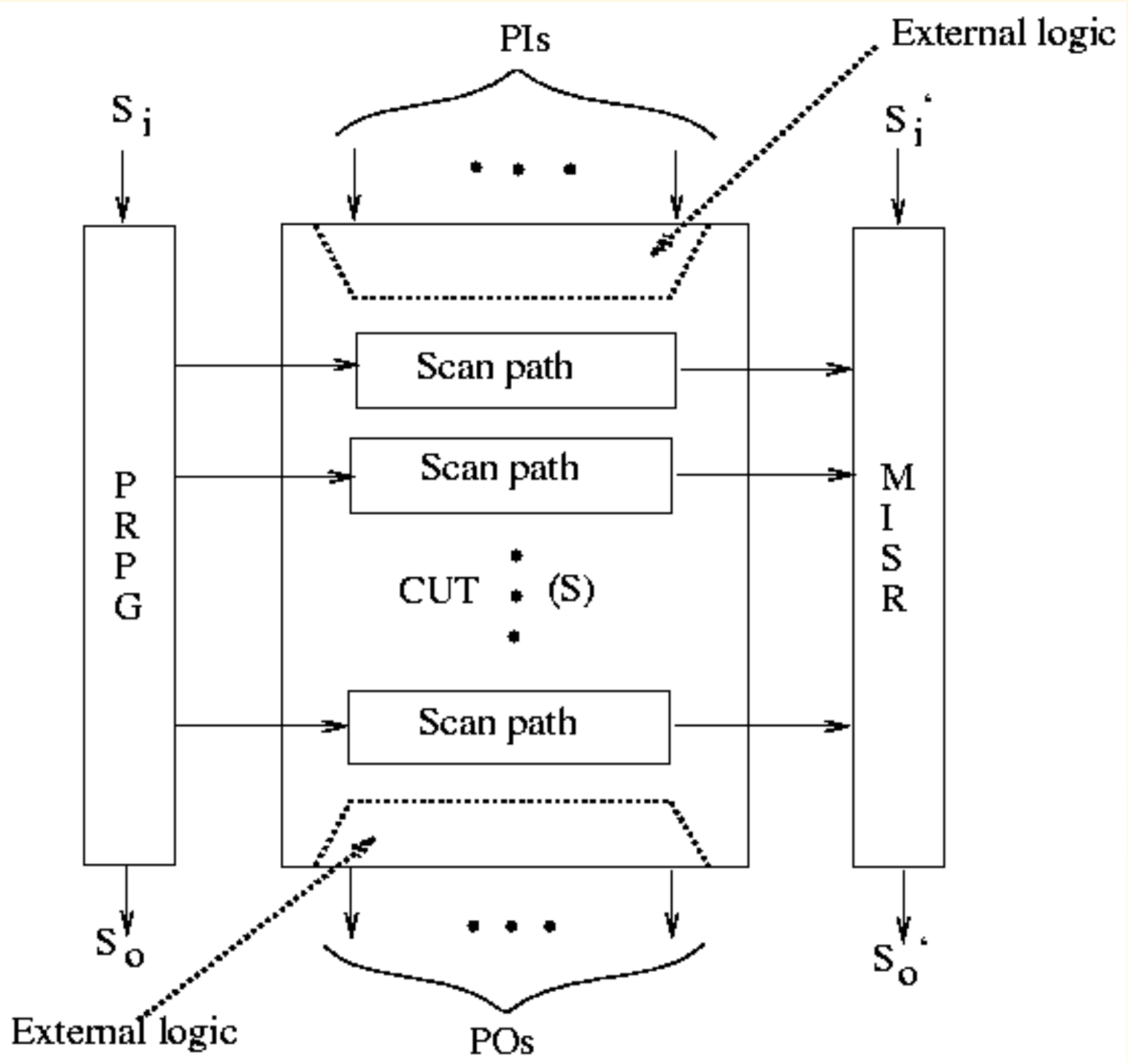

Source: H. Kerkhoff

# Wrapper Cells and Wrapper Registers for P1500



# Example of Built-In Self-Test (BIST)




Technique called
STUMPS (from IBM)

# Issues with Built-In Self Test

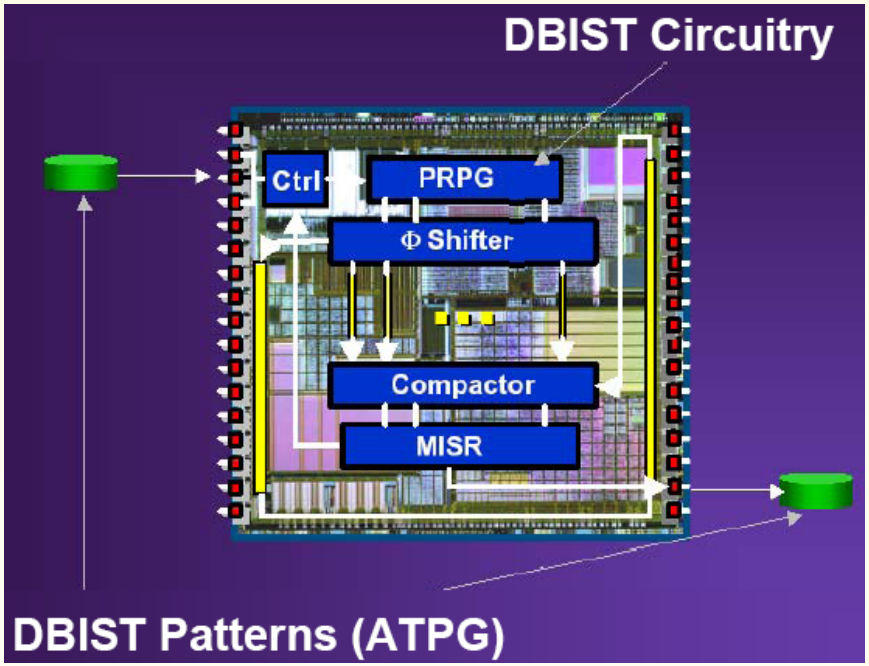
Technique can run test sequences at operating frequencies and capture results within the chip

- Pseudorandom pattern generators, signature analyzers

Can also use weighted random patterns or deterministic patterns

Example (Synopsys): (Deterministic Logic BIST

**Problems:**
**Hardware overhead**
**Test power**
**Non-functional modes during test**



# Industry Issues with Processor Testing

Concern with detecting real defects

- Small delay defects due to process variations, power droops and capacitive coupling
- Cause a shift in the speed of the part

Problems with logic BIST (same issues with scan AC tests)

- Overheads on chip
- False paths tested
- Test operating conditions different from normal operating modes

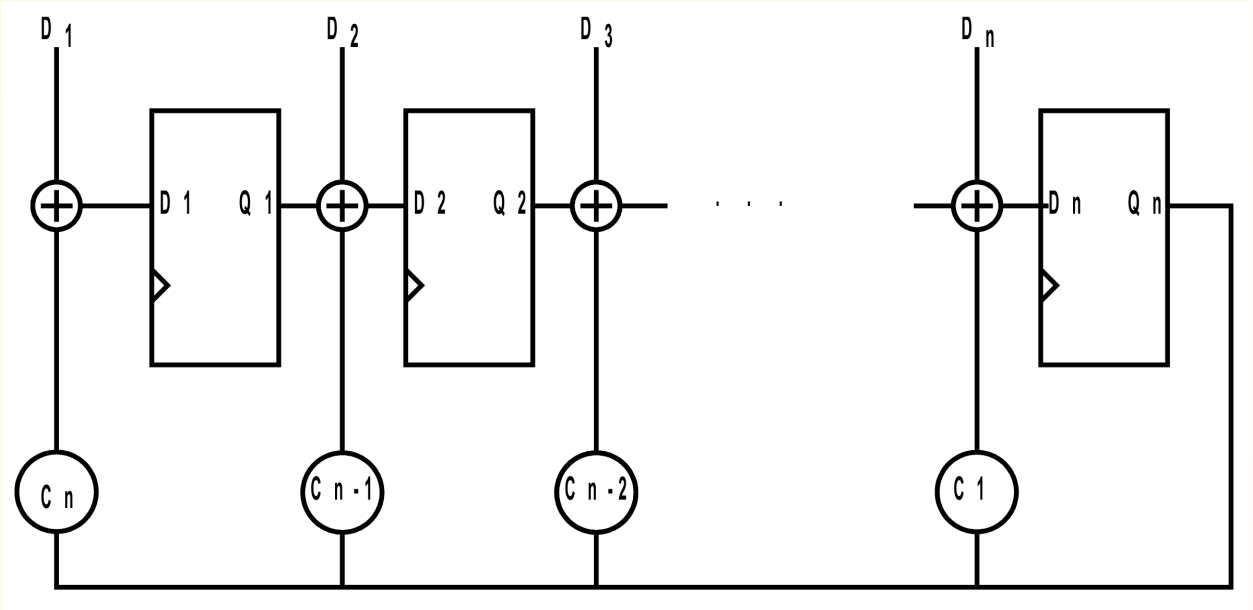
# Software-Based (Native-Mode) Self Test for Processors

- Why not use functional capabilities of processors to replace BIST hardware?
  - No additional hardware
- Reduce test costs by using low-cost testers
- Increase coverage of delay defects and increase yield by testing native
- No issues with excessive power consumption during test

Developed at University of Texas (Int'l Test Conference 1998)

Application to processors at Intel (Int'l Test Conference 2002)

# Native-Mode Signature Compression




Pseudo-code for software signature analyzer

```
// S: general register or memory, holds signature
for each register R_i to be compressed
{
    Shift_Right_Through_Carry(R_i);
    if (Carry) {S = XOR(S, feedback_polynomial)};
    S = XOR(S, R_i);
}
```

# Intel Functional BIST

Functional Random Instruction Testing at Speed (FRITS) applied to Itanium processor family and Pentium 4 line (ITC 2002)

Tests (kernels) are instruction sequences

- Kernels loaded into cache and executed in real time during test application
- They generate and execute pseudo-random or directed sequences of machine code

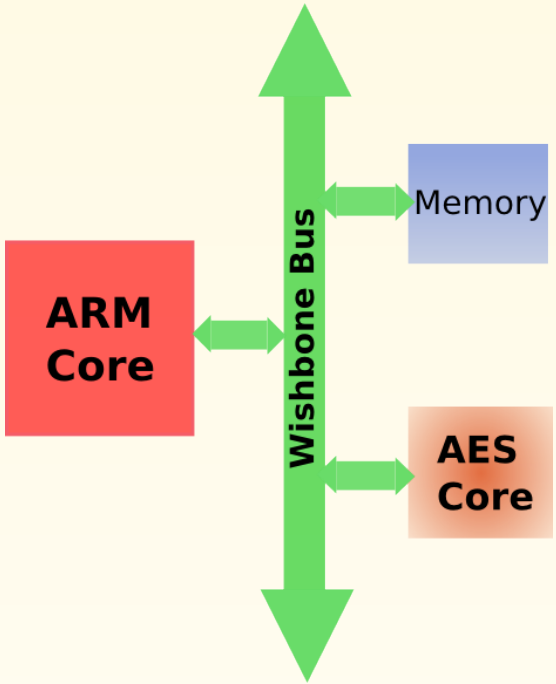
On Pentium-4, FRITS

- added 5% unique coverage to manual tests
- screened 10% – 15% of chips which passed wafer sort/package tests, but failed system tests
- enabled low-cost testers: 40% increase in defect screening on structural tester
- Kernels execute 20 loops in $\approx$ 8 mSecs

# Test of SoC Cores using Embedded Processor

Wishbone and 128-bit AES designs from opencores.org

Validation vectors: random values encrypted/decrypted




AES Core

| | |
|---|---|
| Inputs | 69 |
| Outputs | 33 |
| Combinational primitives | 9225 |
| Sequential primitives | 1119 |
| Stuck-at faults | 64070 |

Result of Mapping AES tests to ARM instructions (one case)

| | Size (bytes) | Fault coverage(%) | Original Coverage(%) | No. of Cycles | Original Cycles |
|---|---|---|---|---|---|
| Test | 9128 | 90.15 | 90.35 | 7816 | 7435 |

Source: Gurumurthy *et al.*, 2008

# Testing SoCs with Embedded Analog/Mixed-Signal/RF Modules

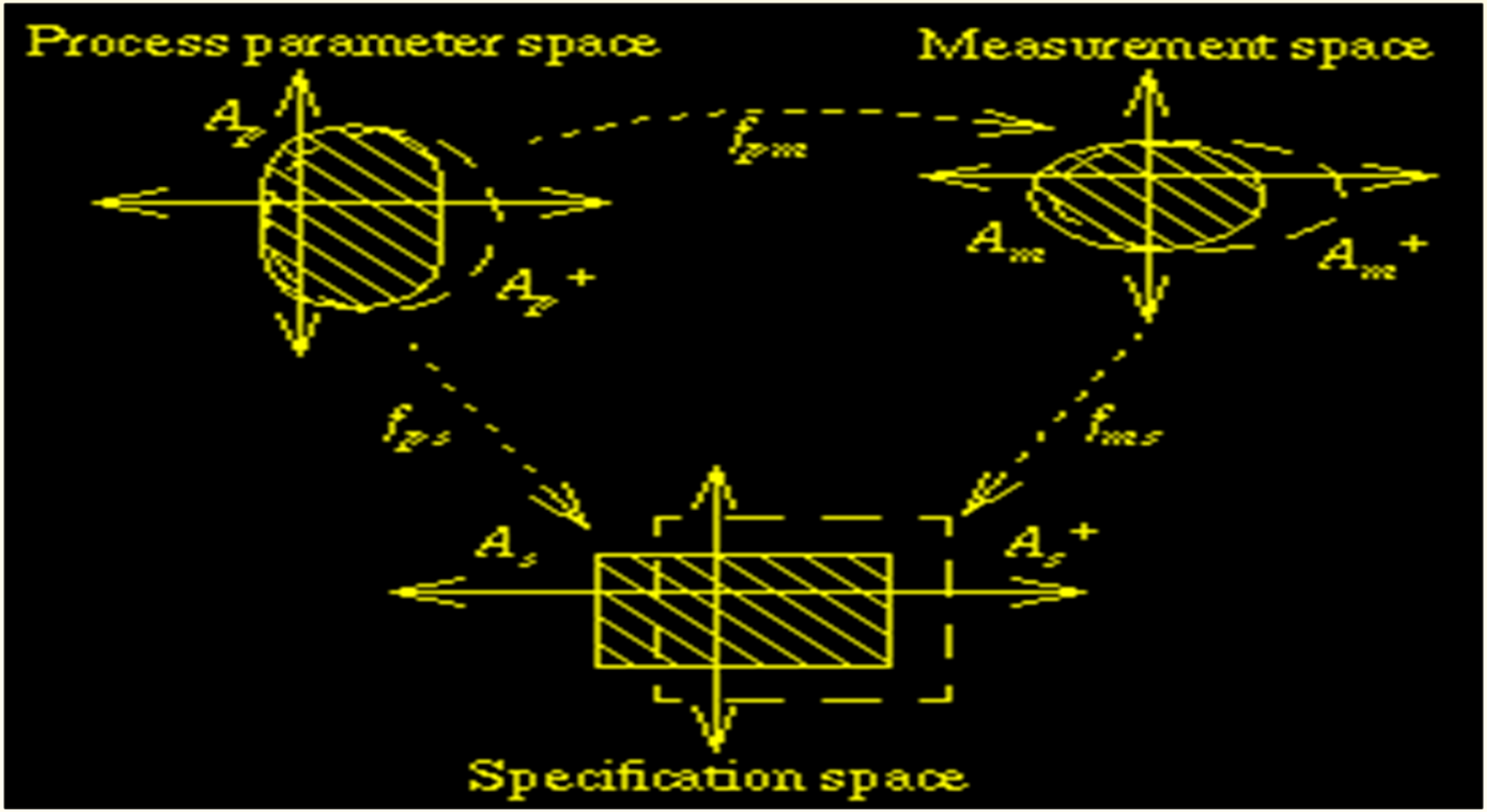
## Analog/mixed-signal test

- Customers want a guarantee of specifications
- A defect may or may not affect the desired behavior of a chip
- Tests are for the specifications, not for defects
- Most analog/mixed-signal blocks are part of larger systems
- Need to develop new directions in testing embedded analog/mixed-signal blocks

## Reducing the cost of testing embedded mixed-signal modules

- Develop indirect (alternate) test techniques to reduce the cost of testing for all specifications

# "Alternate" Tests




Source: A. Chatterjee

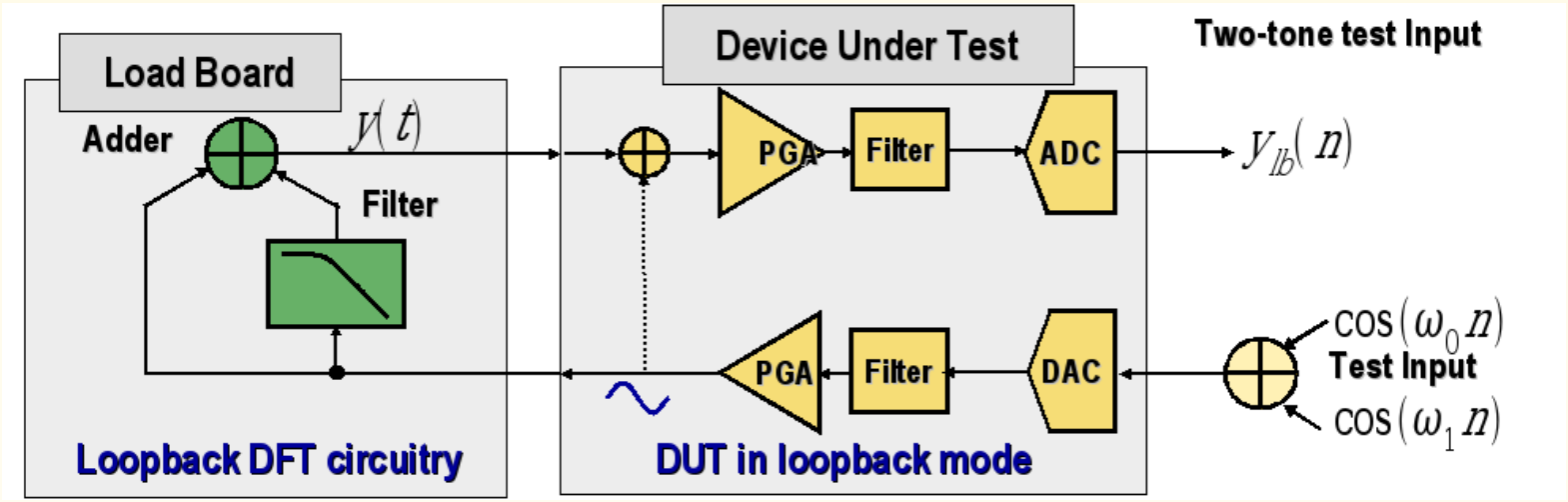
# Loopback + DFT Scheme for ADC/DAC System Test

Provide dynamic performance parameters of individual signal paths

- Avoid yield loss due to fault masking

DFT circuitry – analog filter and adder on loadboard or on chip

- Characterize harmonic distortion and noise parameters




H. Shin *et al.*, Best Paper Award at VTS 2006

# Validation: Hardware Measurements

Broadband modem IC

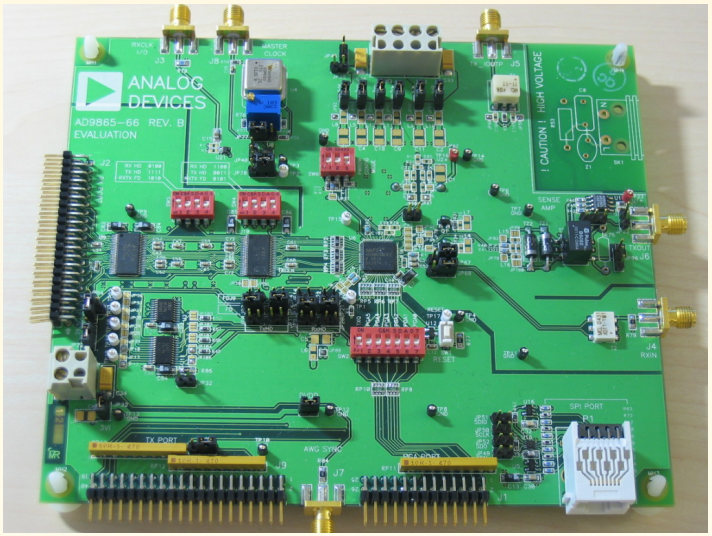
- Tx/Rx data rates up to 80MSPS

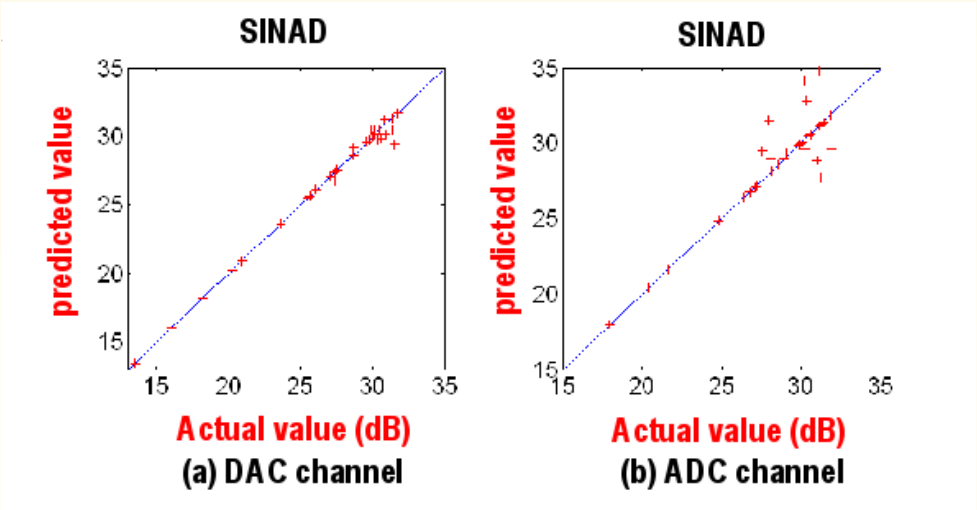
Programmable 3-pole filter

- Bypassed in normal mode

Faults injected by

- Reconfiguring TX/RX gain
- Sweeping supply voltage, input amplitude

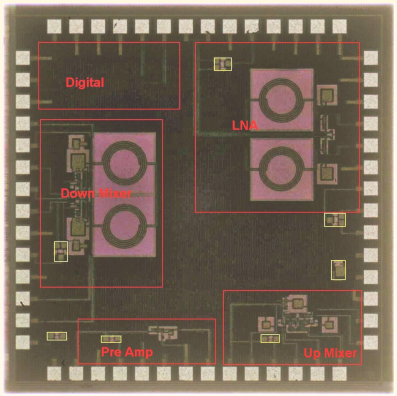


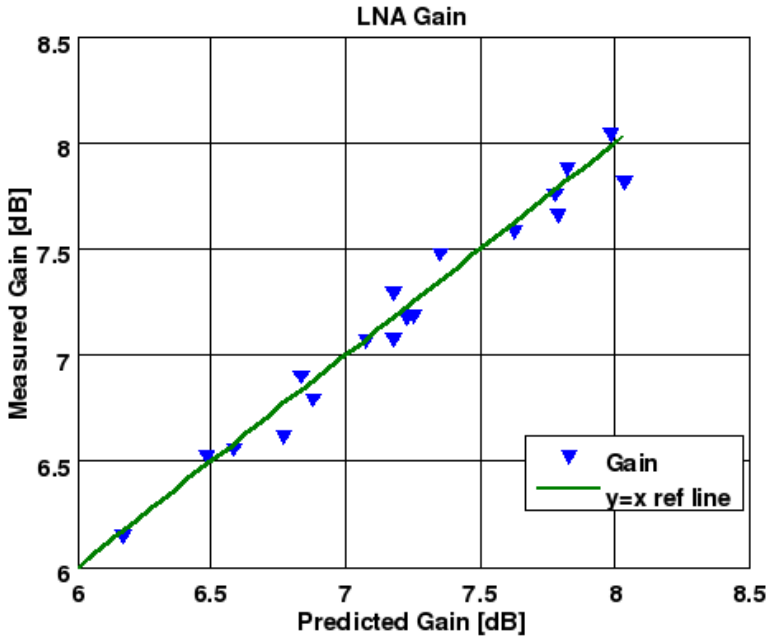
# On-chip sensors for low-cost RF test

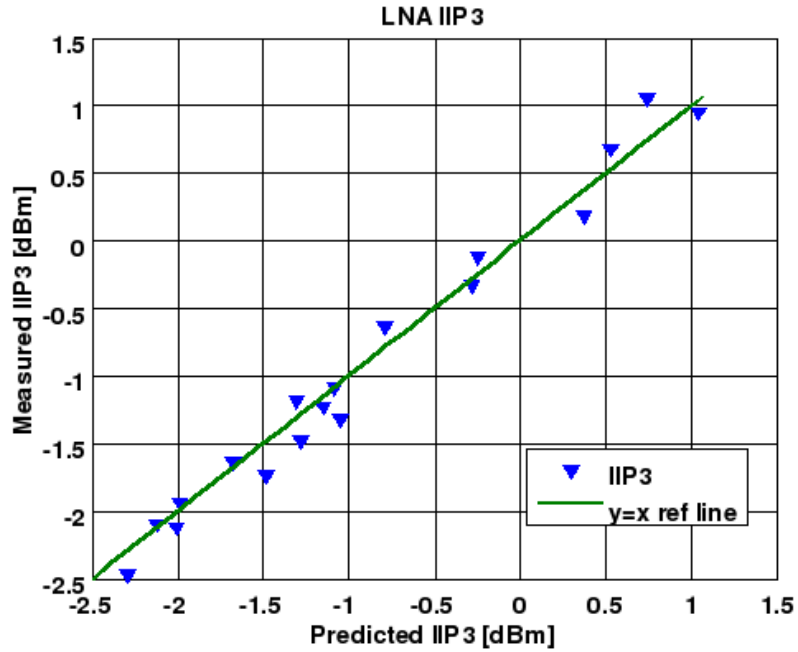
940 MHz RF Transceiver (0.18$\mu$ CMOS)

**10 MHz output used to predict specifications**

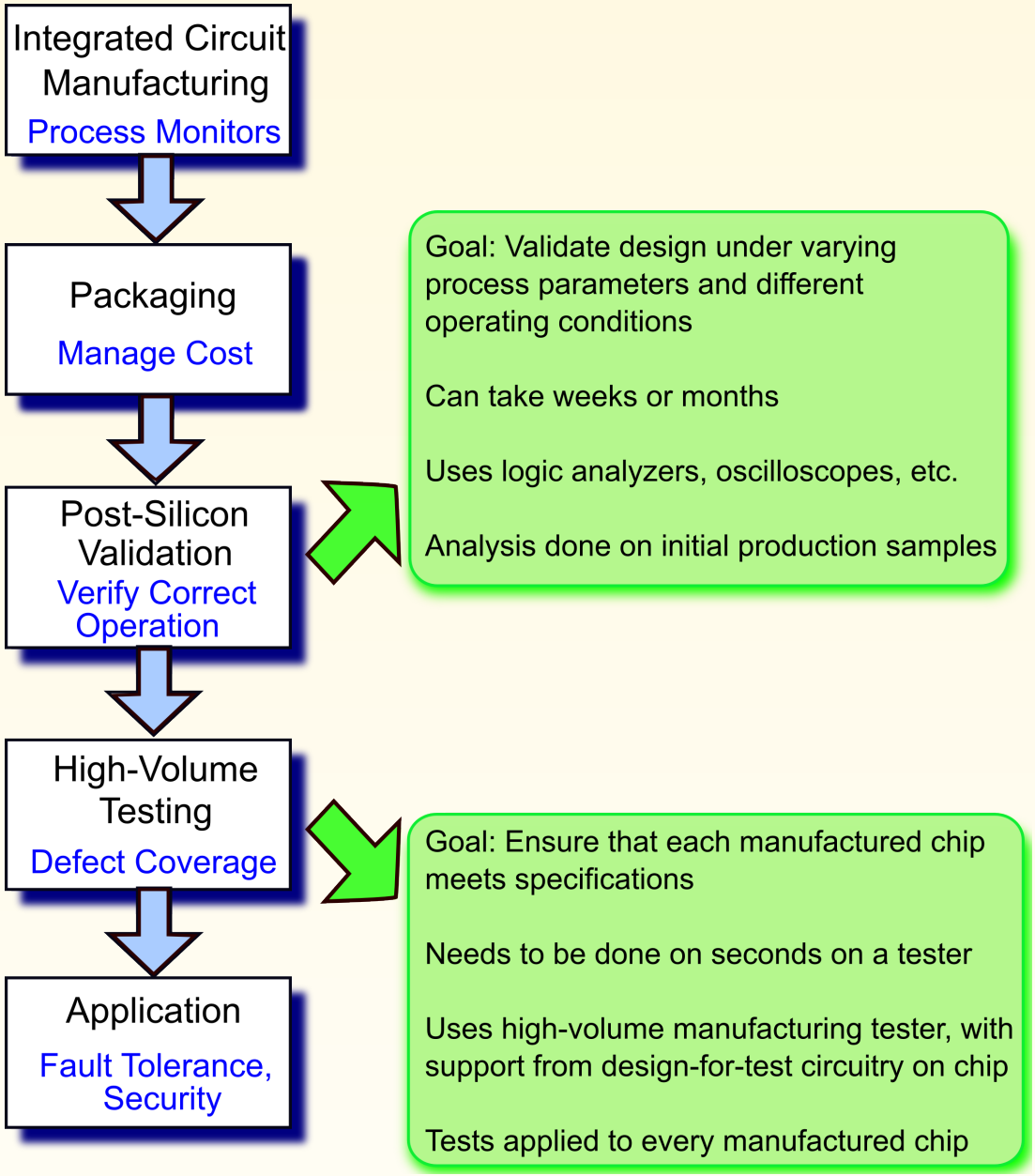



Predicted versus Measured Results for LNA Gain and LNA IIP3

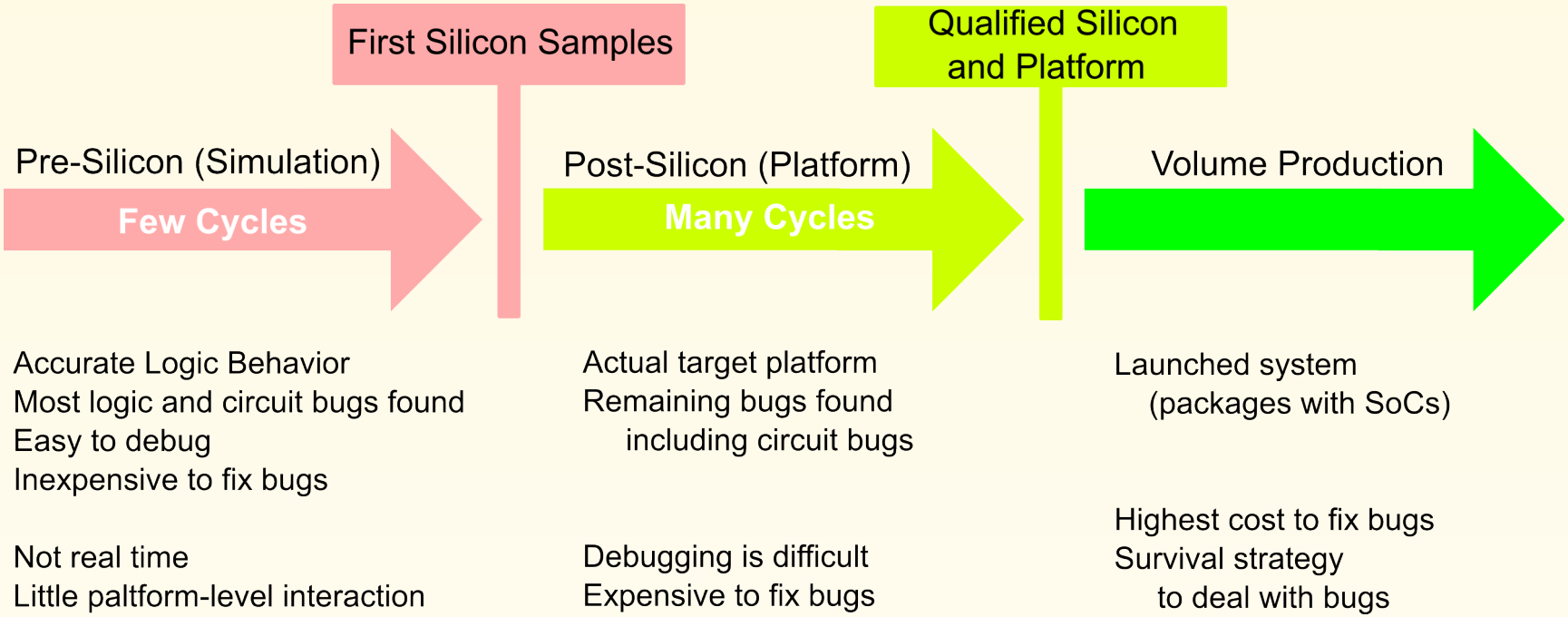


# Post-Silicon Validation and Manufacturing Test



# Post-Silicon Validation Domains




Bugs decline in number over development cycle, but cost to fix them increases

Source: N. Hakim, Intel

# Where Bugs are Found

## Functional bugs (also known as "logic bugs")

- Exist in all manufactured parts (Metric: DPM (defects per million), fatal logic bugs result in 1M DPM)
- 98% found before tape out, 2% post-silicon

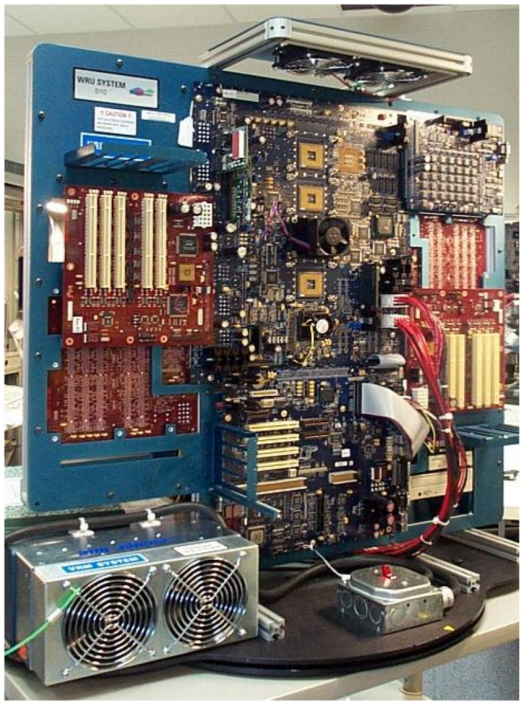
## Circuit bugs

- Not all parts exhibit failures ($<$ 1M DPM)
- Variable with Voltage, Temperature, Frequency, process and component age
- Computation limits to simulation (not real time) limits extent of variation combinations which can be simulated
- 90% found pre-silicon, 10% post-silicon

Source: N. Hakim, Intel

# Platform Validation Infrastructure

**Tests generated in server farm**



**Tests executed on validation platform**

**Failures debugged using logic analyzer**

Source: N. Hakim, Intel

# Classes of Tests

## Functional bugs in micro-architecture

- Weighted random instructions
- Architectural simulation patterns
- Random power state transitions
- Directed tests for corner cases
- Multi-core/multi-processor tests
- Tests of virtualization system

## Memory subsystem

- Memory channel activation
- Tests for multiple cores/processors
- Directed tests for memory paging, cache coherence

# Circuit Bugs

Affect DPM – not all die behave the same way

## Timing convergence bugs

- Speed path: circuit operates too slow
- Min-delay: circuit operating too fast (hold times)
- Race: circuit fails due to timing of multiple converging signals

## Analog bugs

- Primarily occur in I/O buffers, PLLs, and thermal sensors
- Silicon does not operate in accordance with predicted (simulated) circuit behavior

- Fundamentals for circuit bug hunting
  - Need sufficiently large population of devices
  - Need to vary environmental conditions
  - Need to stimulate stressful system behavior
  - Stimulus is generally functional – failures look just like functional failures

# Circuit Bug Root Causes

## On-die signal integrity

- Cross-coupling induced noise
- Droop-event induced noise

## Power delivery integrity

- High dynamic current events
- Clock gating often results in high dynamic current

## Clock domain crossing

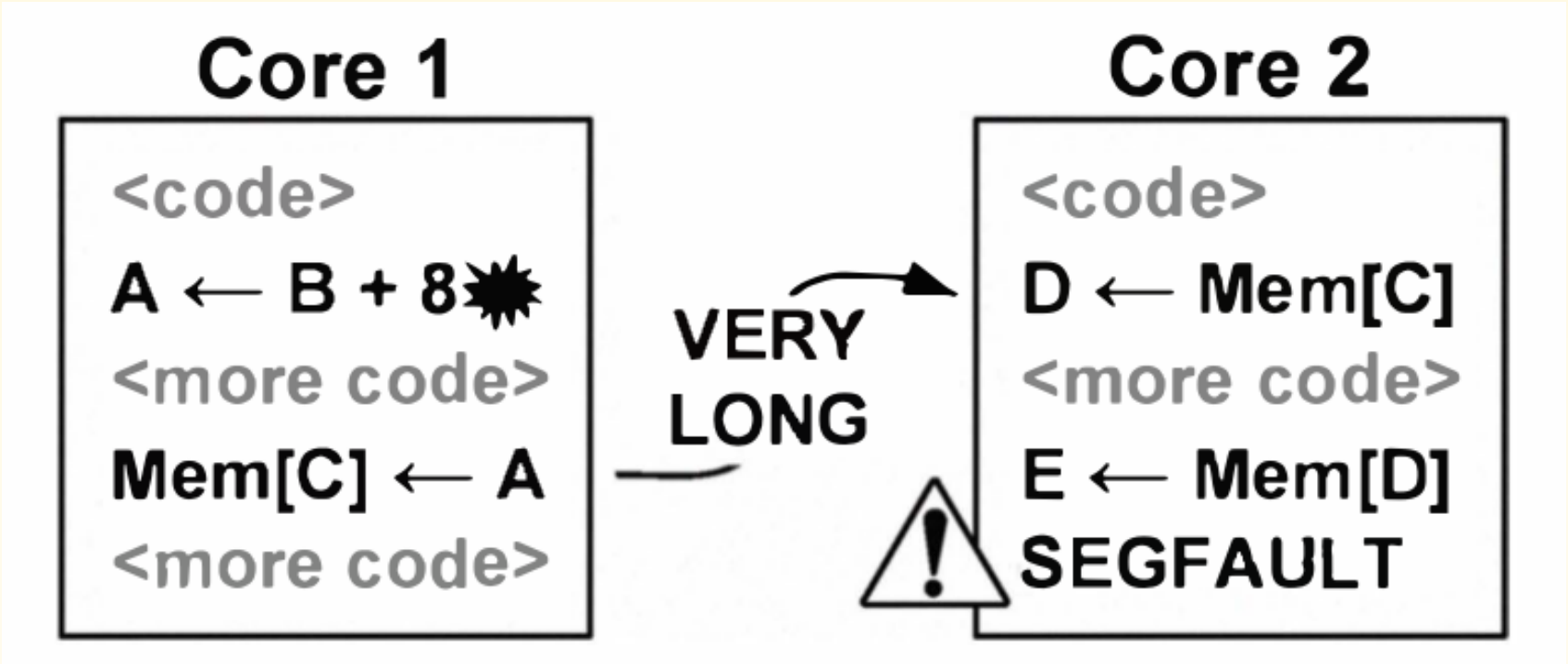
- May cause synchronization and timing issues

## Process, Voltage, Temperature

- Power state transistors
- Silicon process variation

Source: N. Hakim, Intel

# Long Error Detection Latencies in Practice

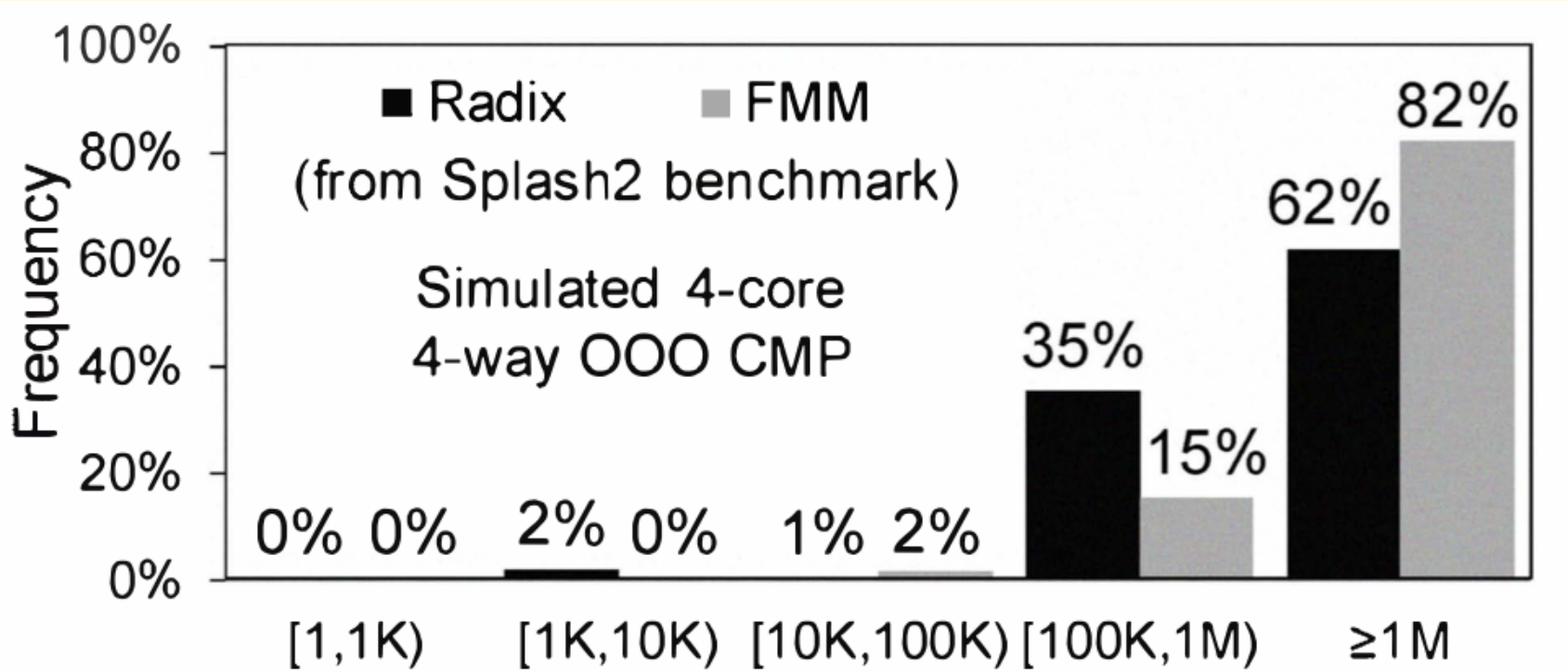



- Inter-core interactions result in long error detection latencies

Source: Hong *et al.*, 2010

# Distribution of Inter-Core Store-to-Load Latencies

- Splash2 benchmarks executed on a simulated 4-core, 4-way, out-of-order MIPS processor




Source: Hong *et al.*, 2010

# Quick Error Detection (QED) Tests

- QED tests obtained by transforming existing post-silicon validation tests into new tests with significantly lower error latencies
- Enabled by a variety of *QED Transfomations*
- Transformations done with software-only or hardware-only changes
- Allow flexible tradeoffs between error detection latency, coverage (percentage of bugs detected) and complexity (additional software or hardware modifications required)

**Initial focus on electrical bugs**

- These bugs are very time consuming to debug
- Modeled as bit-flips of storage elements (flip-flops)

# QED Transformations

Based on concurrent error detection techniques

## Error detection by duplicated instructions for validation (EDDI-V)

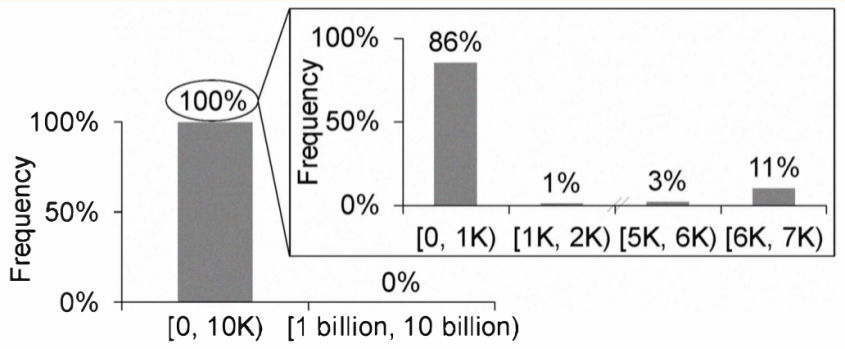
- Strategically duplicates instructions and compares their results
- Each "block" of instructions is duplicated and a check is inserted to compare the results of the two blocks

## Redundant Multi-Threading for Validation (RMT-V)

- RMT-V executes the original instructions on one thread and uses an additional thread to execute the duplicated check instructions
- The two threads can be simultaneously executed on different cores
- Additional instructions in main thread to transmit its results to the check thread

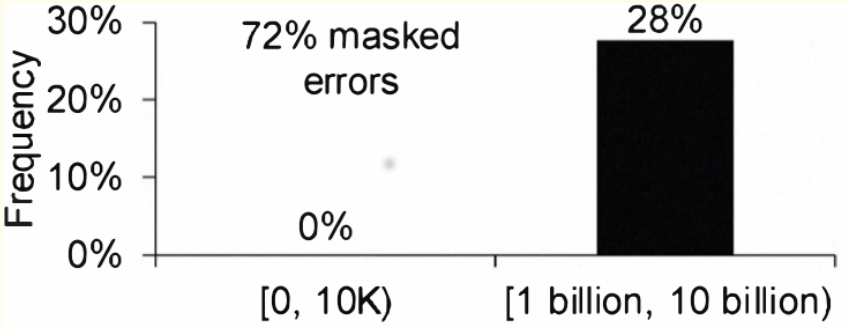
# Error Detection Latency Results with Linpack

Linpack test using QED checks




Execution Cycles

Linpack test using only end-result-checks




Execution Cycles

# Dealing with Incorrect Results: "Fault Tolerance" vs. "Resilience"

## Fault Tolerance

- Errors (due to faults) detected and corrected, fault located, reconfiguration around faulty unit
- System designed to tolerate classes of faults
- User does not see anything wrong (except perhaps an additional delay)
- Service does not suffer any down time

## Resilience

- User may see errors during the service, but the final results are correct
- System requires on-line error detection, but may use checkpoints, retry, etc., to achieve resilience
- Ability to deal with "unknown" faults
- Service may be down intermittently

# Detecting Incorrect Results – Historical Perspective

## Dionysius Lardner – **Checking Results, Diversity**

"The most certain and effectual check upon errors which arise in the process of computation, is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if they make their computations by different methods,"
*Edinburgh Review*, No. CXX, July 1834

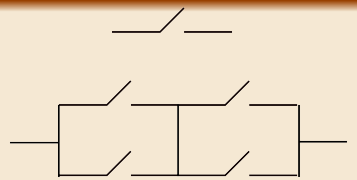
## Japanese Proverb – **Resilience**

The bamboo that bends is stronger than the oak that resists

# Key Techniques for Reliability

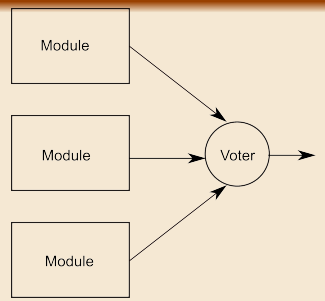
## Switch (Circuit) Level

- Shannon, 1956
- Reliable circuits using less reliable relays



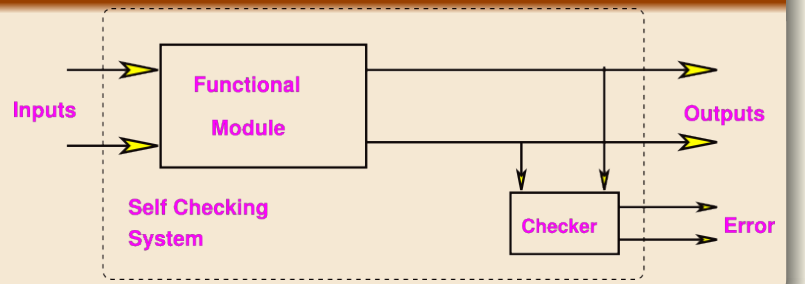
## Module Level

- von Neumann, 1956
- Reliable systems from unreliable components



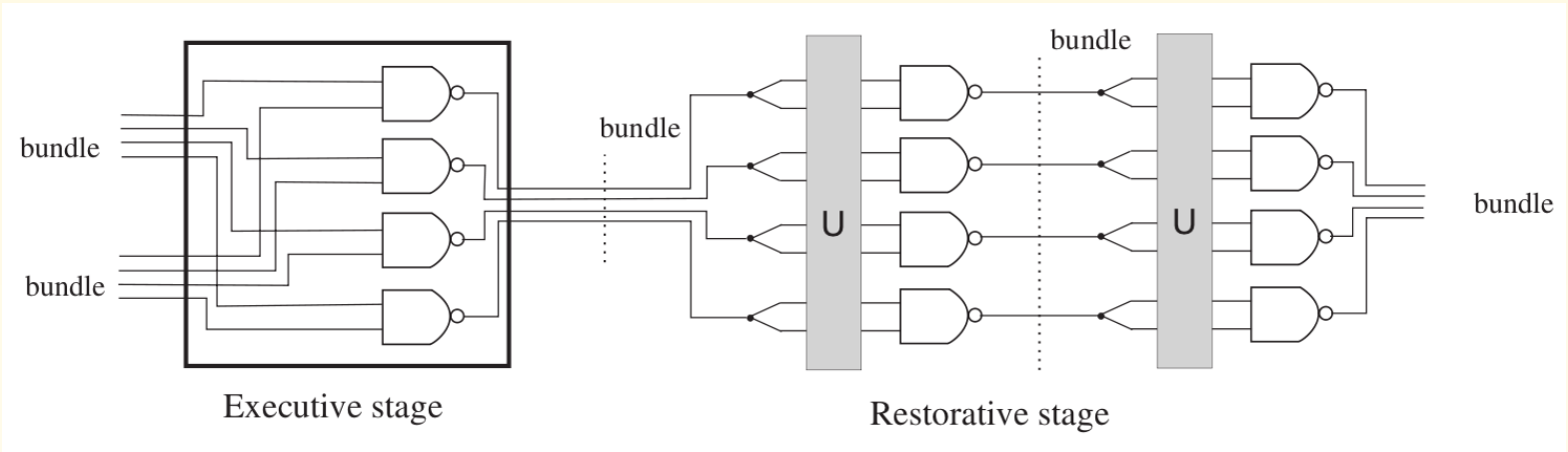

## Module Level

- Anderson and Metze, 1971
- Self-Checking Circuits



# NAND Multiplexing – Application to Nanotechnology

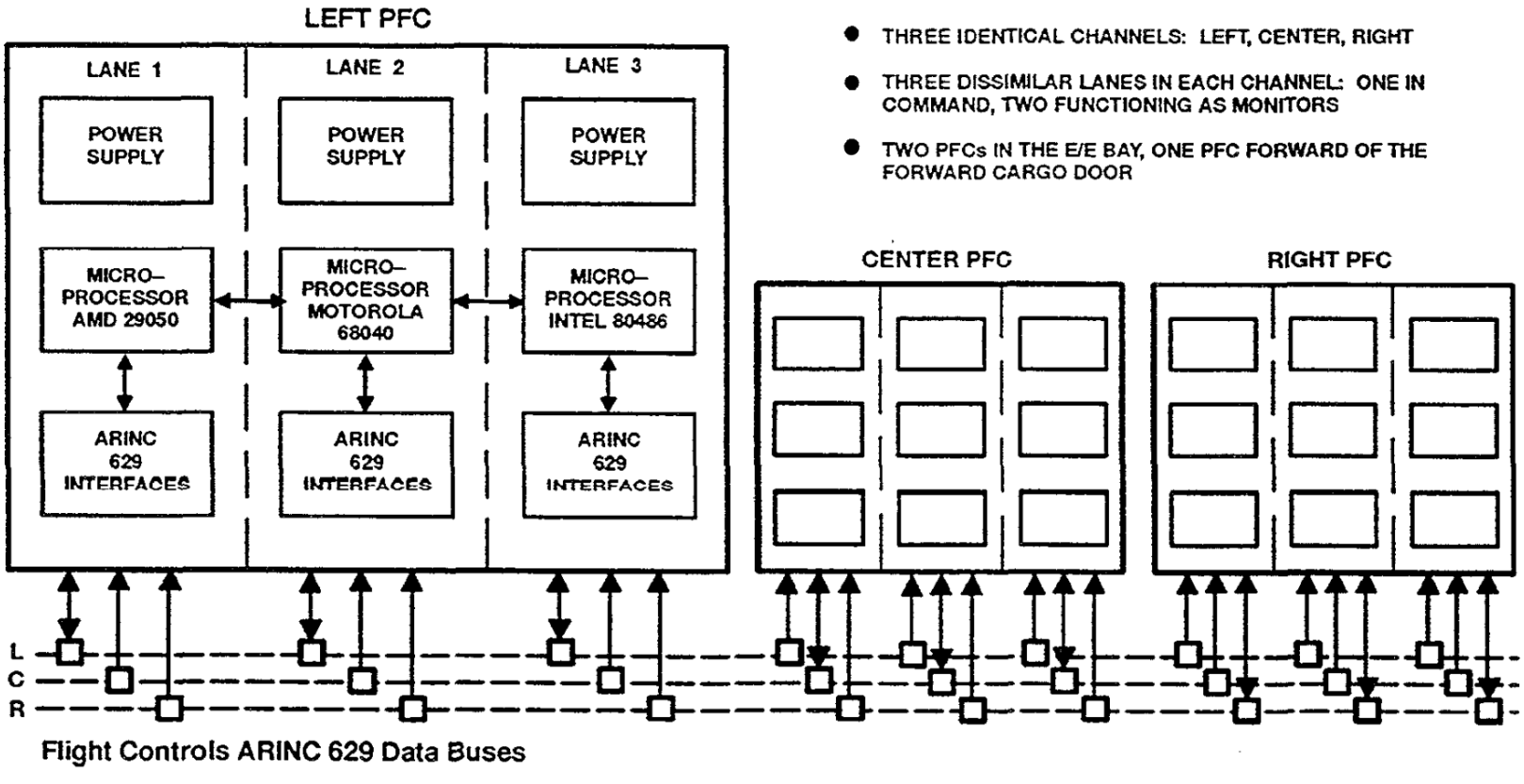
Proposed in the 1956 von Neumann paper, now being proposed for logic implemented in nanotechnologies




## Approach

- Similar to NMR, but instead of voting to decide the output, it is carried out in a *bundle*
  - Executive stage – performs operations
  - Restorative stage – reduces degradation caused by errors from the executive stage

# Boeing 777 Primary Flight Computer

# Example of Resilience – Single Engine Airplane

## From Malibu Jetprop pilot's operating handbook

- If loss of power occurs at altitude, trim the aircraft for best gliding angle (90 KIAS) and look for a suitable field.
- At best glide angle, no wind, with the engine stopped and the propeller feathered, the aircraft will travel approximately 2 miles for each thousand feet of altitude.

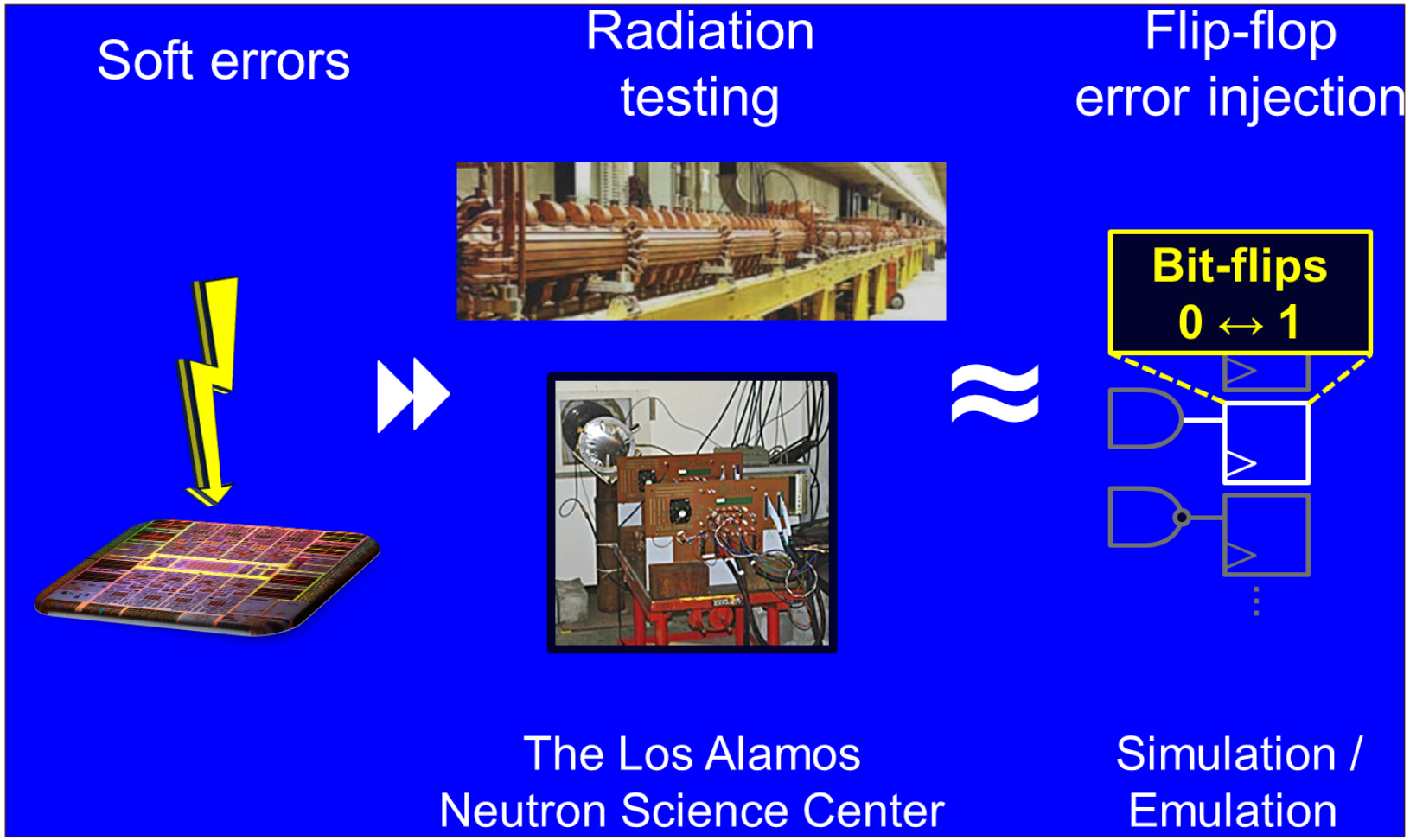
# Example of Resilience – Single Engine Airplane

## From Malibu Jetprop pilot's operating handbook

- If loss of power occurs at altitude, trim the aircraft for best gliding angle (90 KIAS) and look for a suitable field.
- At best glide angle, no wind, with the engine stopped and the propeller feathered, the aircraft will travel approximately 2 miles for each thousand feet of altitude.

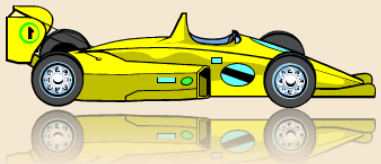
# Evaluating the Effects of Soft Errors

## Radiation-induced soft errors and ways of evaluating them



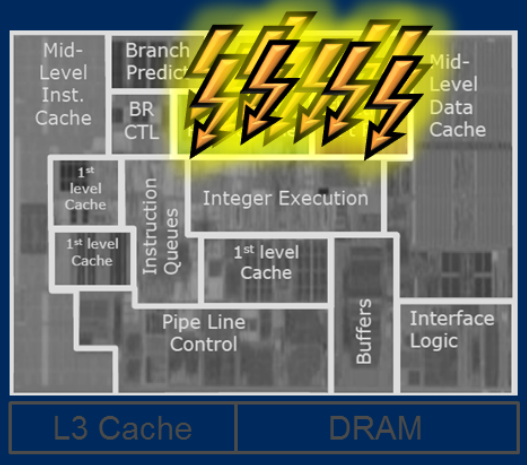
# Error Injection Tradeoffs

## Simulation speeds



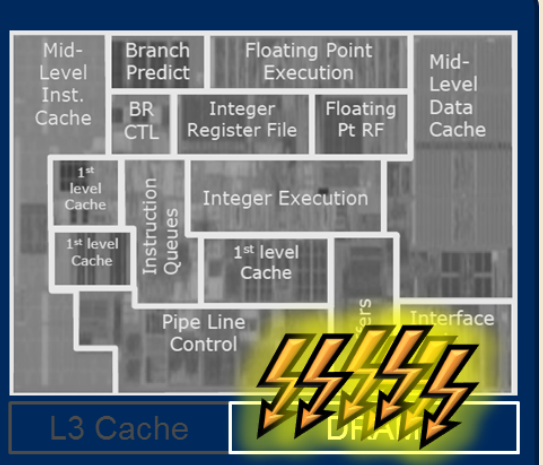

**Flip-flop**

$10^2$ cycles / sec




**Architectural register**

$10^7$ cycles / sec




**Program variable**

$10^9$ cycles / sec

# Error Injection Study

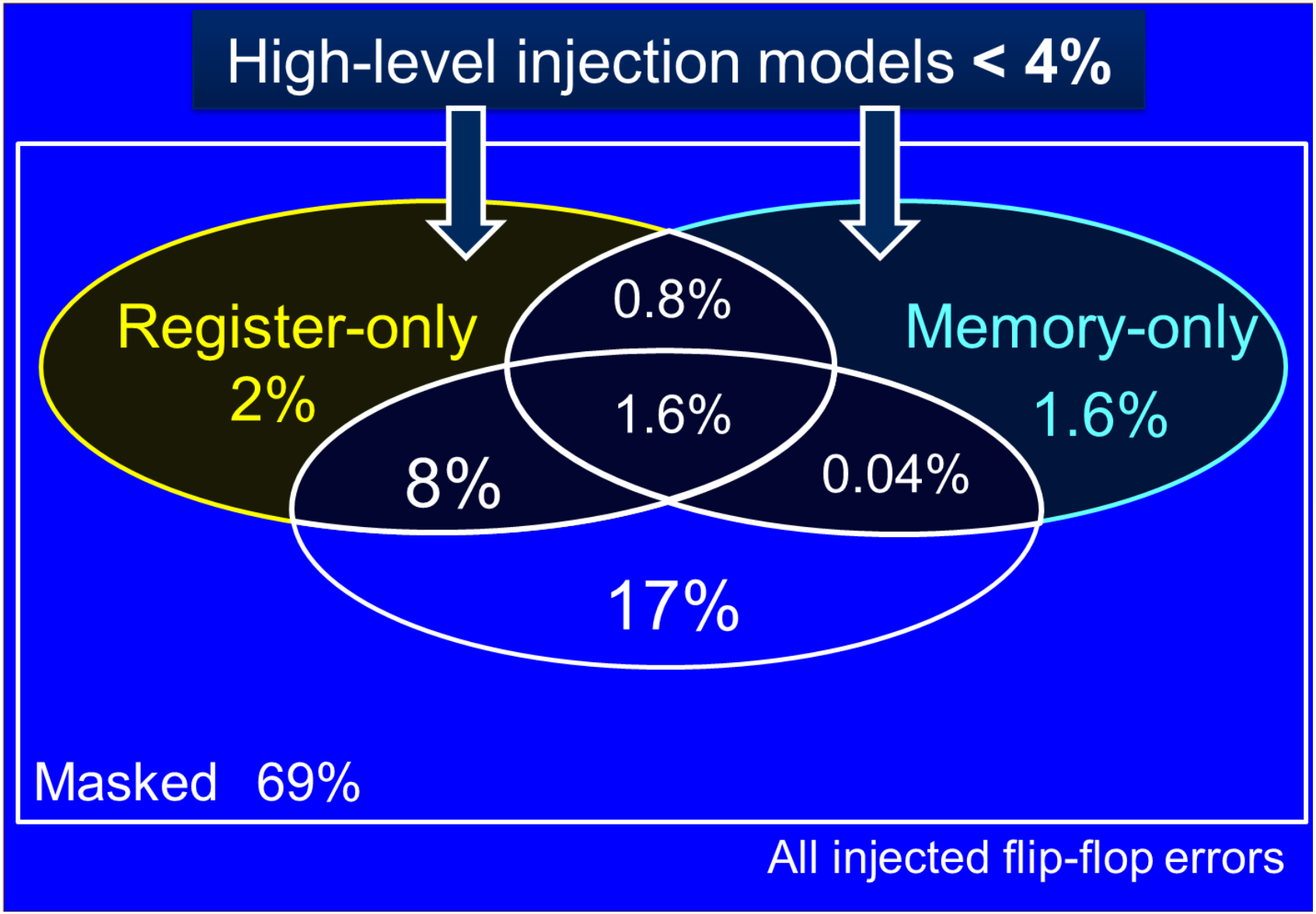
## Joint research project, University of Texas at Austin and Stanford University

- Designs
  - LEON3 (in-order, single-issue)
  - ALPHA (out-of-order, superscalar)
- SPEC 2000 applications
- Bit flip in logic-level flip-flops
- 6 million error injection samples

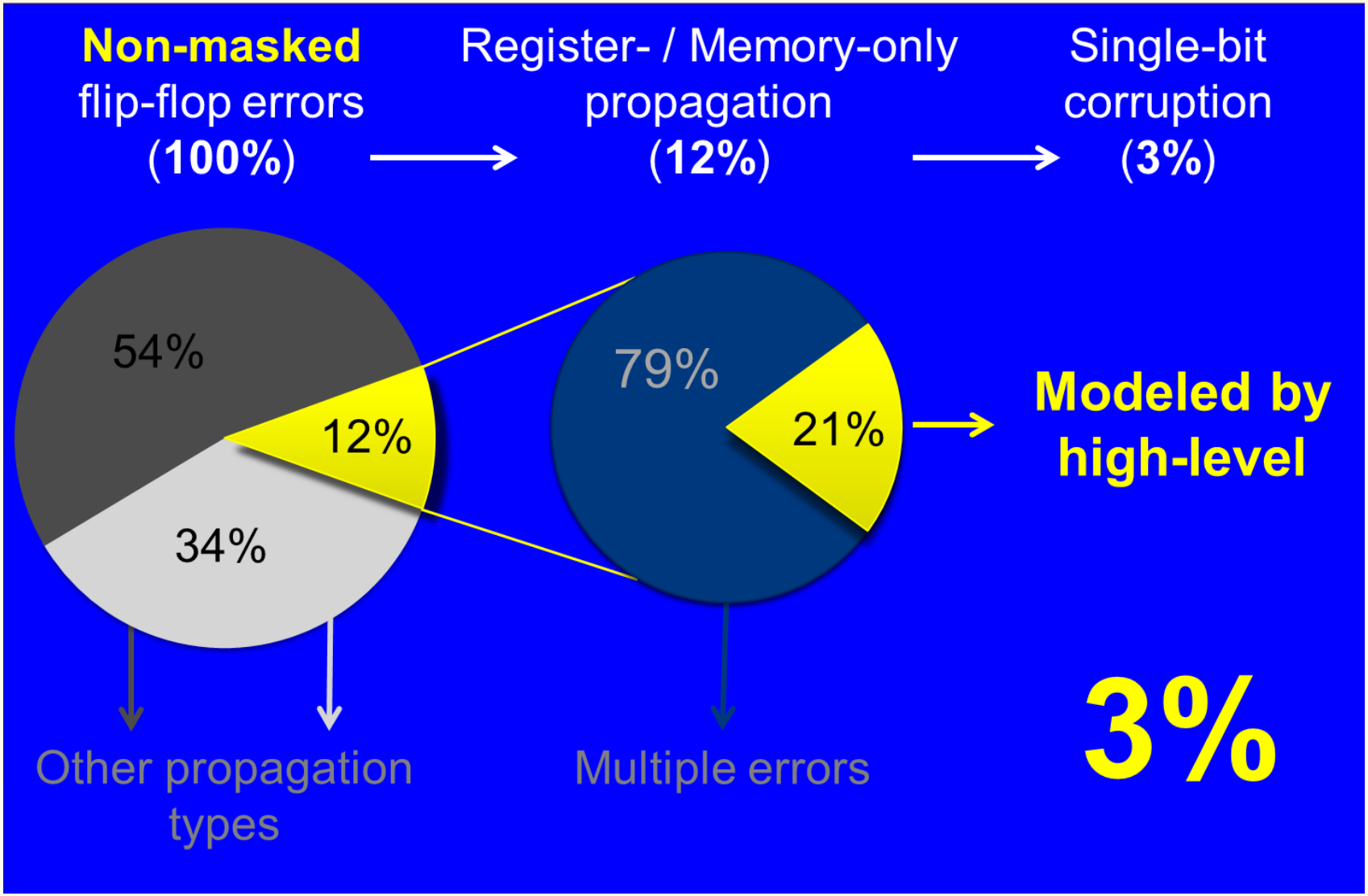
## Outcomes

- Vanished
- **Output Mismatch**
- Unexpected Termination
- Hang

# Tracking Flip-Flop Error Propagation



# Limitations of High-Level Error Injection



# Lessons Learned

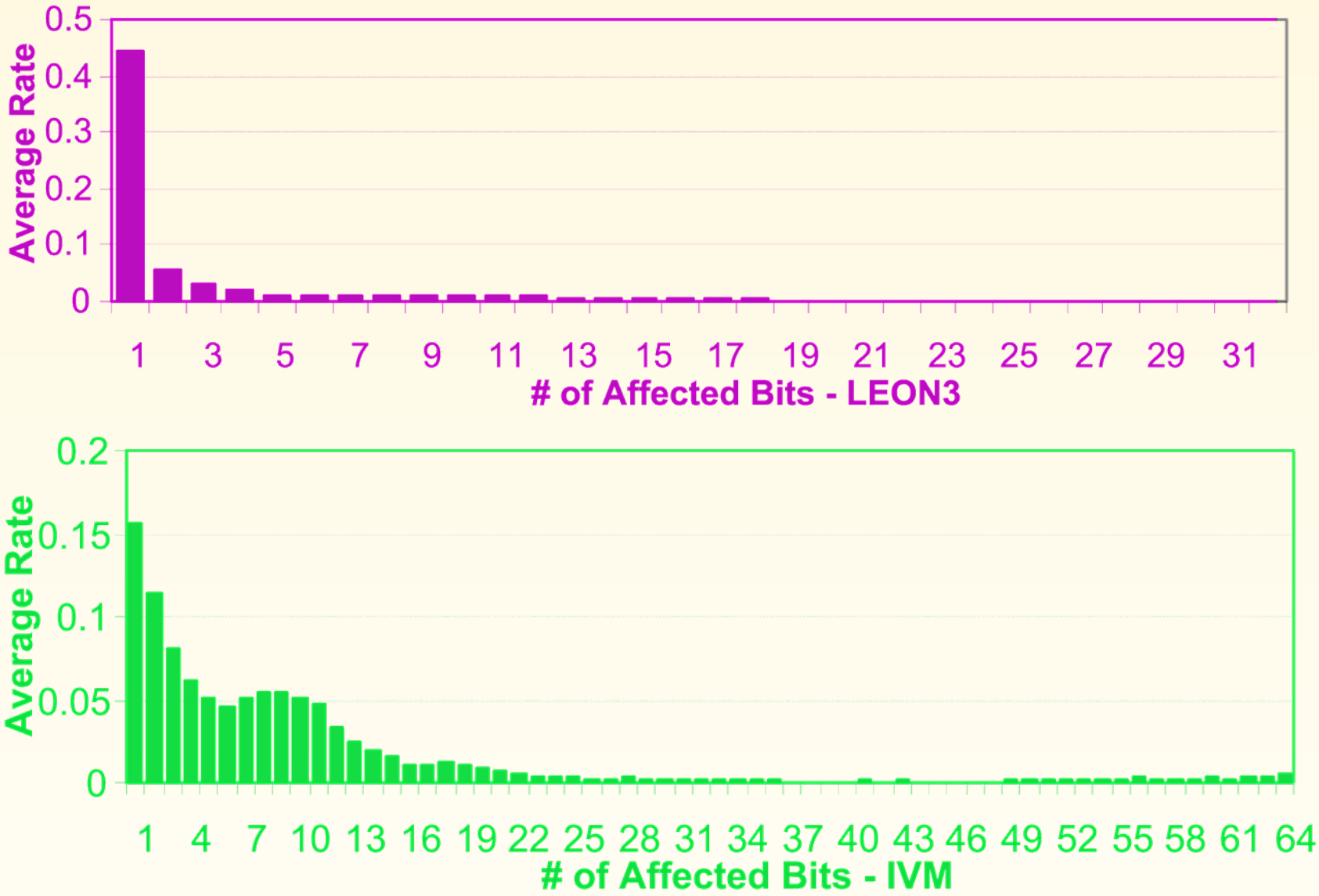
## High-Level Error Injections Inaccurate

- How inaccurate?
  - **Up to 45 X**
  - Neither optimistic or pessimistic
- Why inaccurate?
  - **Only 3% of flip-flop errors modeled**

## Future directions

- Better high-level models
- Improved simulation techniques – hybrid, hierarchical
- Approximation of high-level error probabilities
- Beyond soft errors

# Need High-Level Models for Behavior Under Errors




[Mirkhani 2014]

# FIESTA: Fault Injection for Embedded System Target Applications (Krishnamurthy *et al.*, 1992)

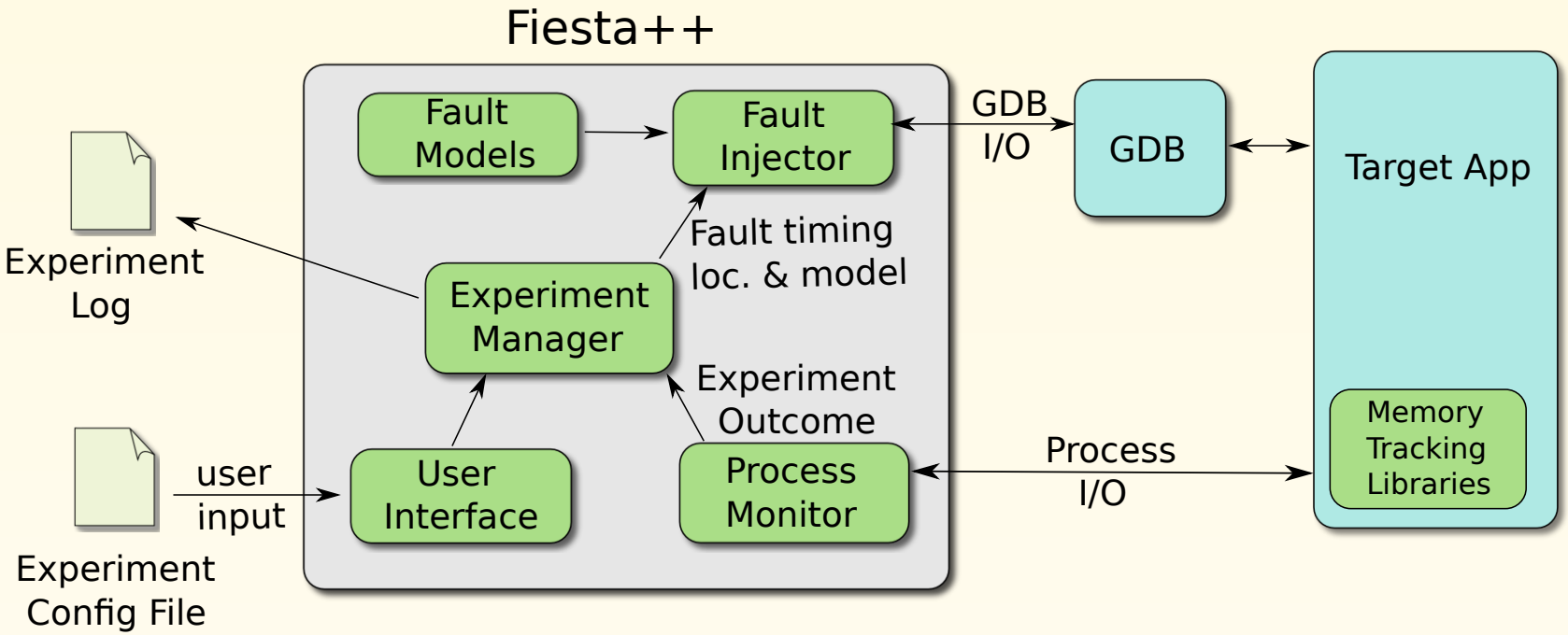
## Build fault injection on top of existing infrastructure

- Open architectures
- Support injection with "real-time" debuggers
- Initial system configuration
  - Host: Solaris 2.5.1, SUN 4
  - Target: VxWorks 5.3, MC68040
- Target embedded system applications (control computers)

## Evolution of FERRARI Tool, 1992

- Errors due to hardware faults emulated using software routines

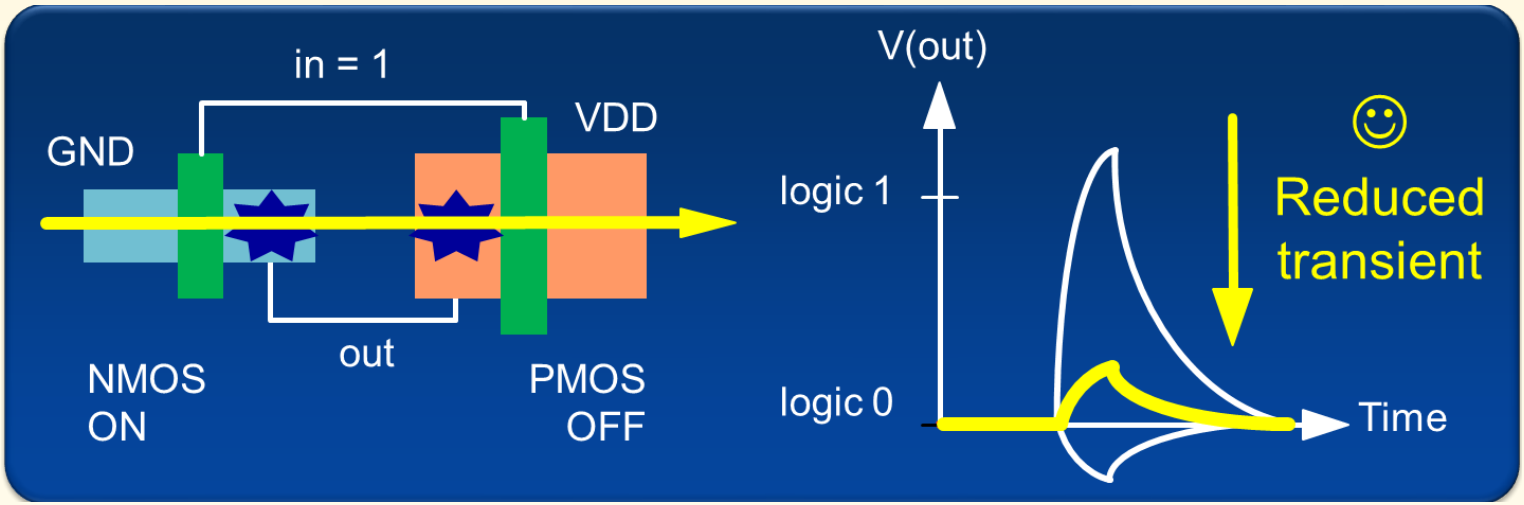
# Fiesta++: GDB Based Software Fault Injector (Chaudhari, 2011) – Open Source Software




- Uses GDB, but can be extended to any debugger
- Injection experiments can be carried out with the entire software stack in place
- Programmable models for fault injection of both hardware faults/errors and software bugs

# Effective Circuit-Level Resilience

LEAP: **L**ayout design through **E**rror-**A**ware transistor **P**ositioning



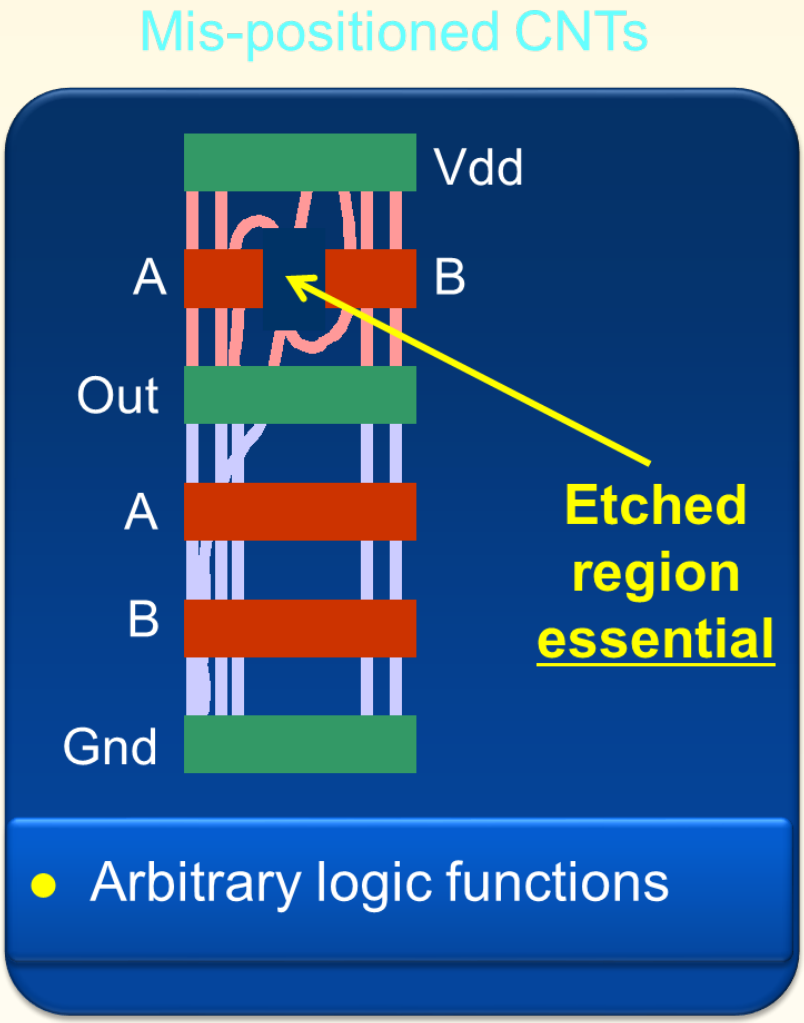

Radiation beam expts.: 40nm, 28nm, 20nm, 14nm (Bulk/SOI)

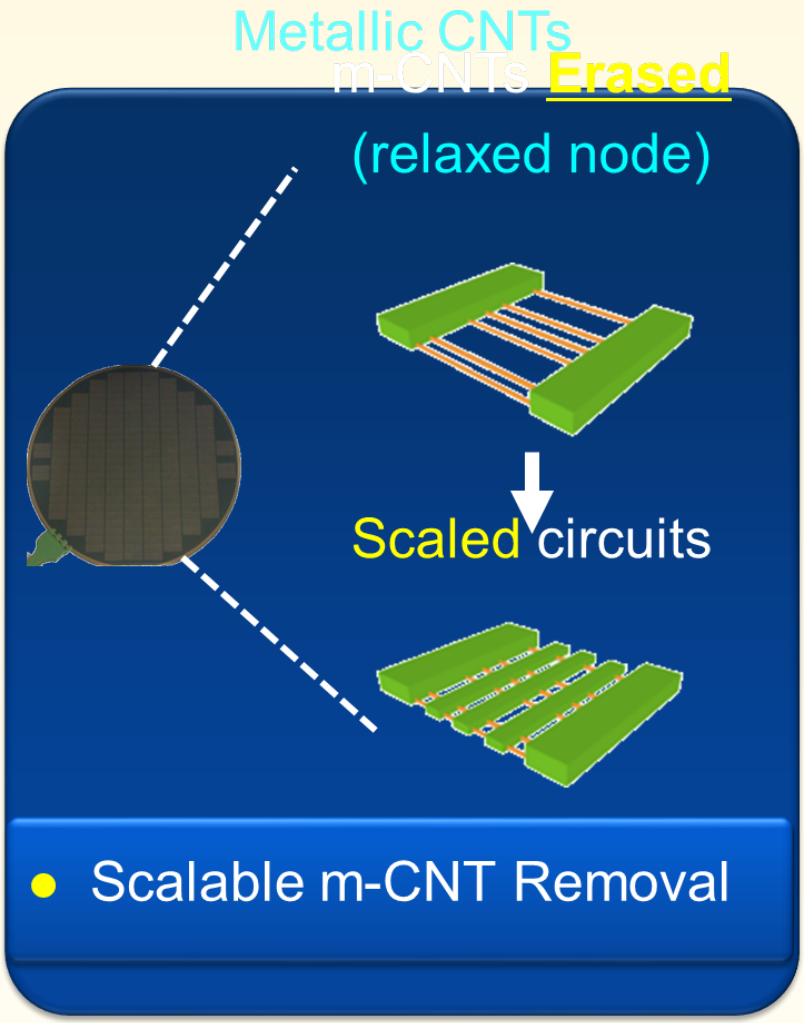
## VDD: nominal, near-threshold

| Flip-flop | SER | Area | Power | Delay | Energy |
|---|---|---|---|---|---|
| Baseline | 1 | 1 | 1 | 1 | 1 |
| LEAP-DICE | $2 \times 10^{-4}$ | 2 | 1.8 | 1 | 1.8 |

[Lee IRPS 10; Lilja IEEE T. Nucl. Sci. 13, SEE 16; Quinn NSREC 15, REDW 15; Turowski SEE 15]

# Imperfection-Immune Carbon Nanotubes (CNTs)




[Patil Symp. VLSI Tech. 08, TCAD 09, Shulaker IEDM 15]

# CNT Computer




[Shulaker, Nature 13

# Detecting Control Flow Errors

## Low-cost applications; example: embedded, mobile Systems

- Need to detect control-flow errors without high overhead in either hardware or performance
- Supplement data checks

## Security: intrusions and attacks

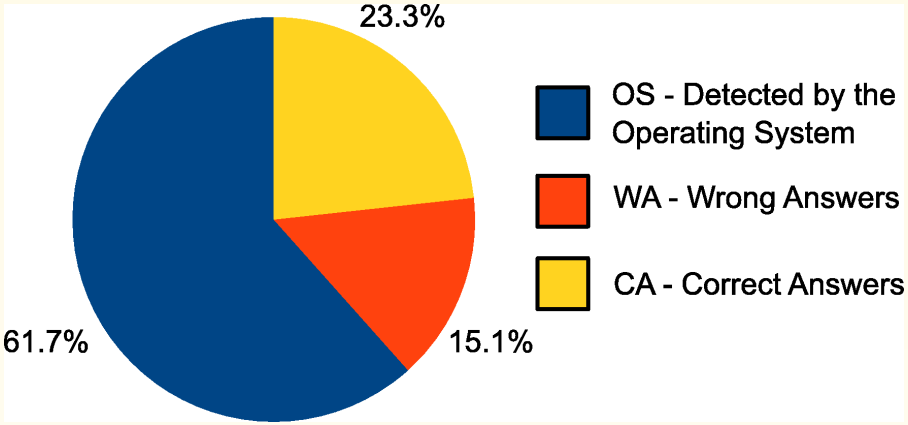
- Any security violation which changes the program flow will be detected with a control flow check

## Control Flow Errors

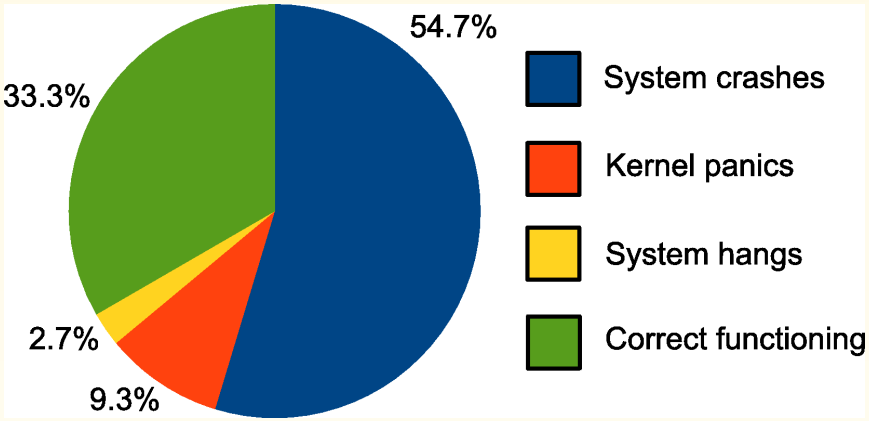
- Execution of incorrect sequence of instructions
  - Can be caused by transient or permanent faults
  - 30% to 50% in RISC systems
- Detection methodologies
  - Hardware redundancy: high cost
  - Software-based checking: high performance overhead

# Effect of Control Flow Errors

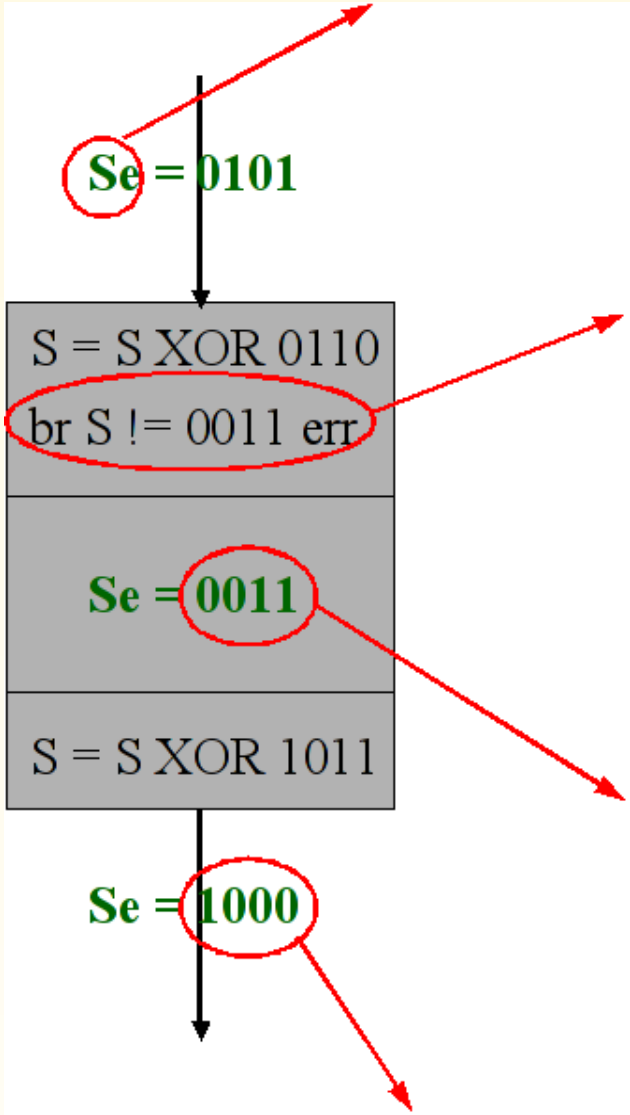
On application software




On Linux OS



# CEDA – Control Flow Error Detection through Assertions



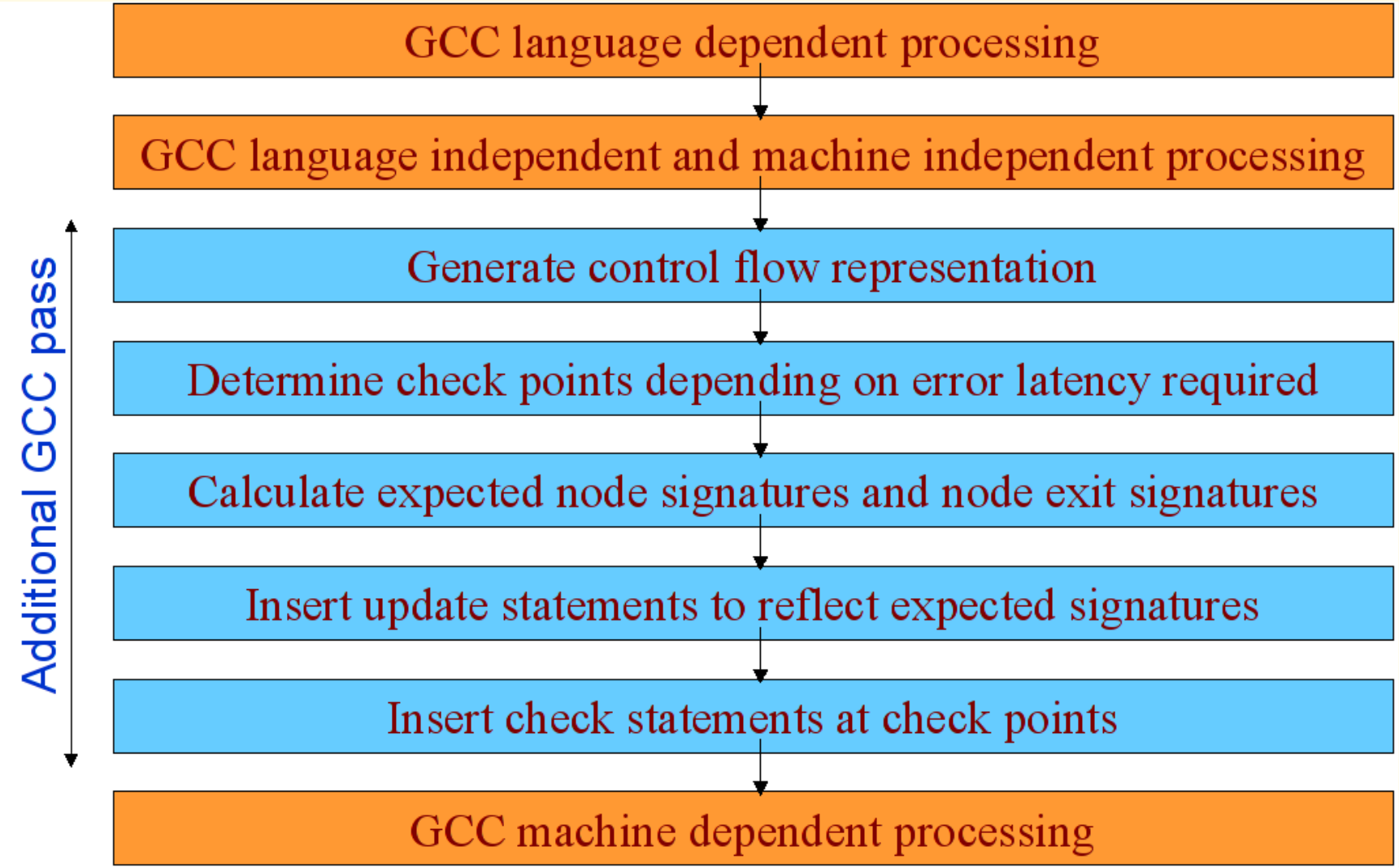

Se: expected value of S at each point in the program (calculated at compile time)

Check point: S is checked against its expected value (detects CFE if one occurred, not required inside every node)

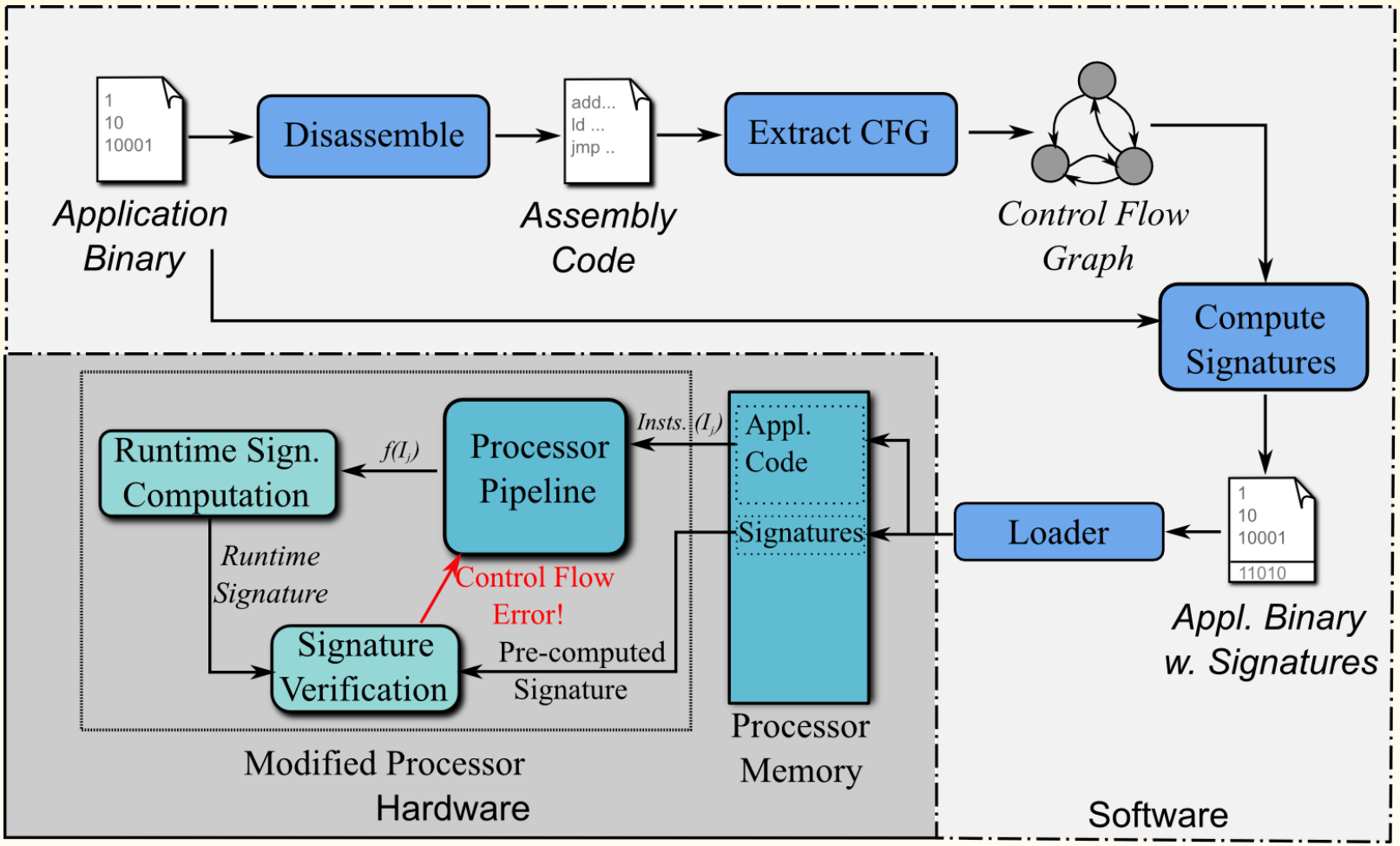
Node signature: expected value of S inside a node

Node exit signature: expected value of S immediately after exiting a node

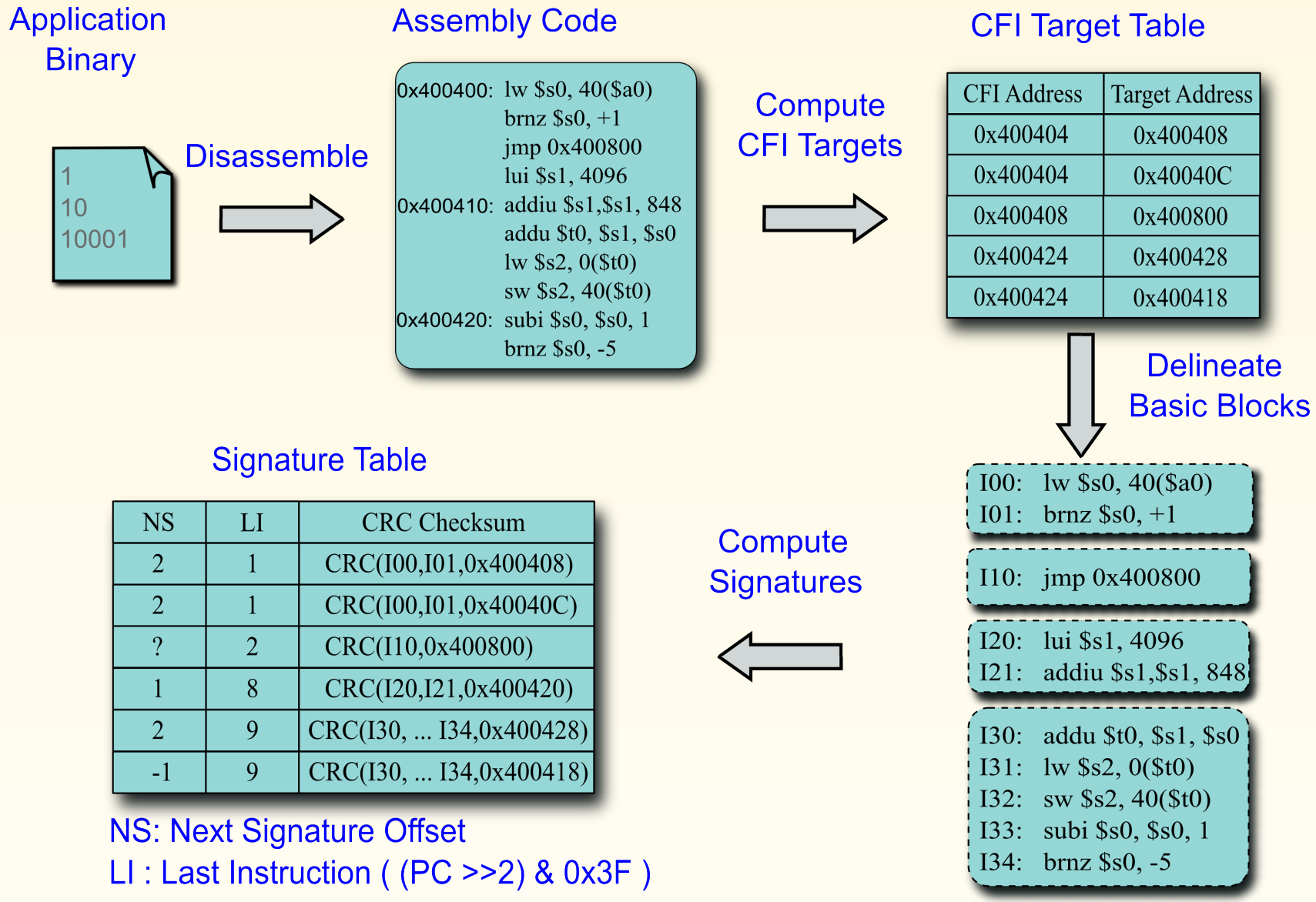
# Integration with GCC



# Framework for Hardware Control Flow Monitoring



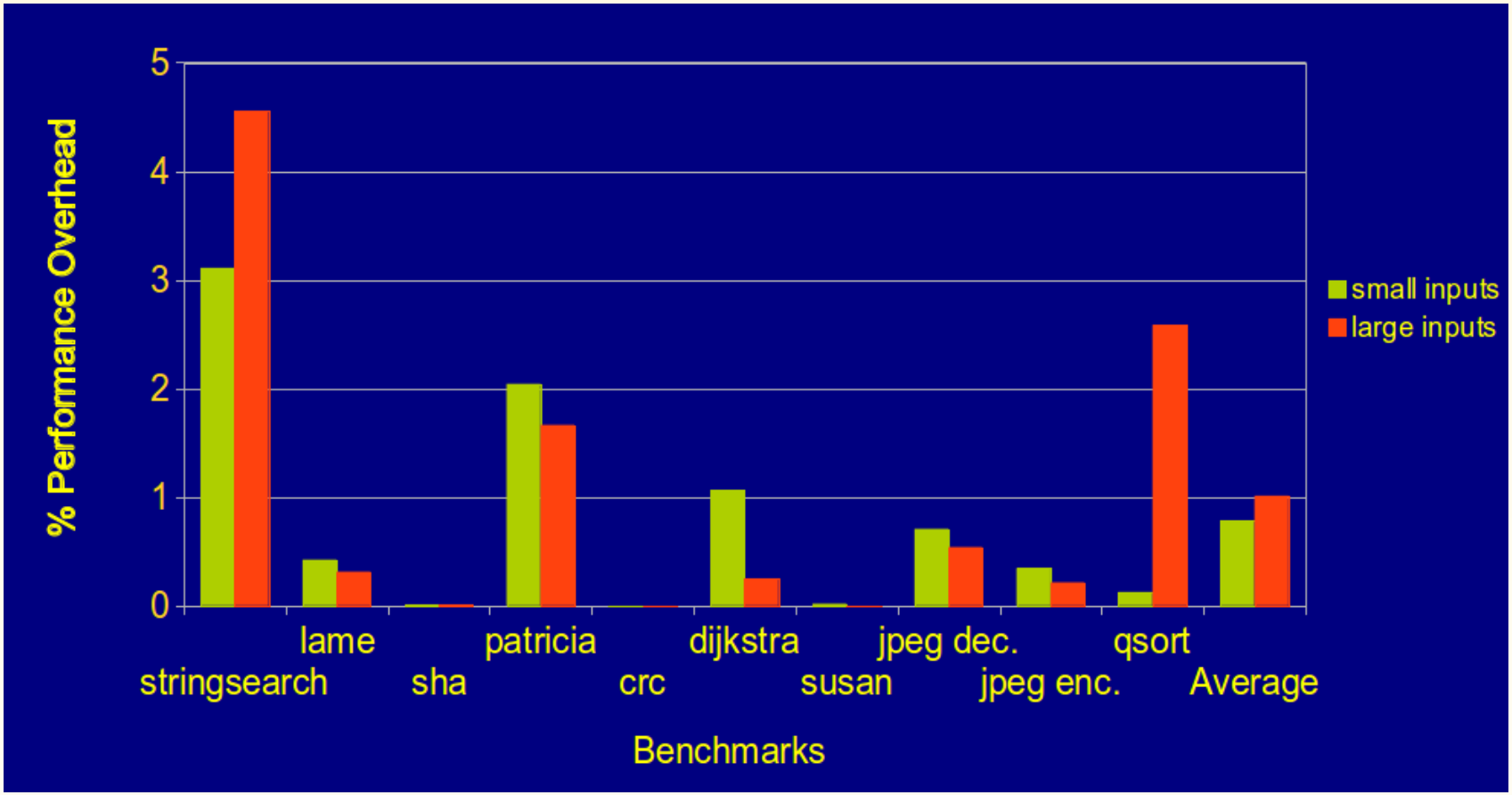
# Signature Computation



# Performance Overhead

MIPS 4Kc processor used as benchmark

In 180nm technology, total area overhead = 5.8%



# Application-Level Fault Tolerance

Reduce the cost of fault tolerance by looking at computations at a higher level

**Algorithm-Based Fault Tolerance (ABFT), (Huang and Abraham, 1984)**

- Encode data at a high level (application level)
- Design algorithm to operate on encoded input data and produce encoded output data
- Distribute computation tasks among multiple computation units, so that failure of a unit affects only a portion of the output data, enabling the correct data to be recovered

Very general fault model: A computation unit can produce any arbitrary logical output under failure

Communication paths checked using coding techniques

# Application to Matrix Operations

Add an extra row and an extra column to encode the matrix

A is an $n \times m$ matrix and $e^T = [111 \ldots 1]$

Column Checksum Matrix,

$$A_c = \left[\frac{A}{e^T A}\right]$$

Row Checksum Matrix, $A_r = [A \mid Ae]$

Full Checksum Matrix,

$$A_f = \left[\frac{A}{e^T A} \mid \frac{Ae}{e^T Ae}\right]$$

# Example of ABFT Applied to Matrices

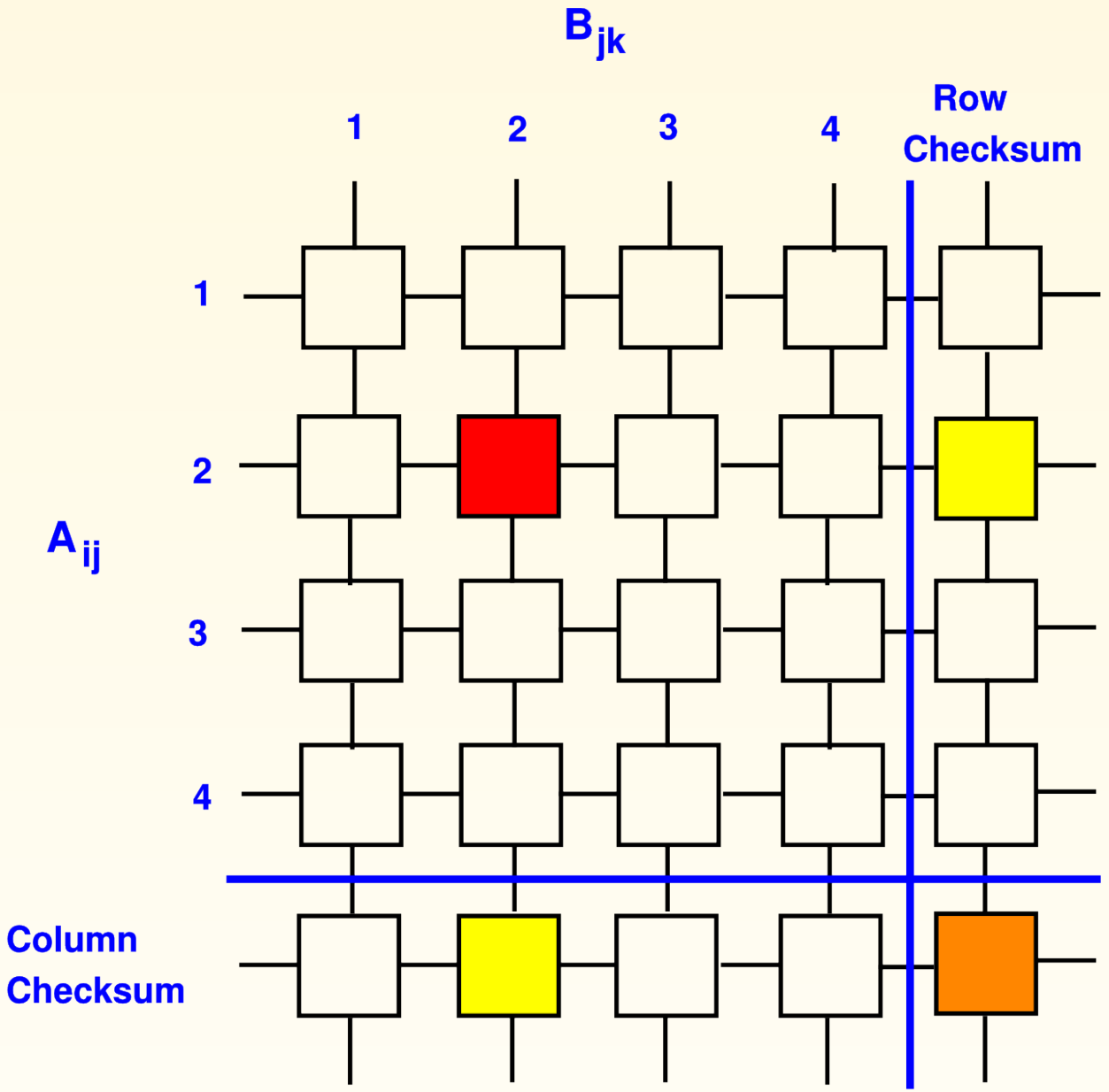
$$A = \begin{bmatrix} 2 & 3 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \end{bmatrix}$$

$$Ac = \begin{bmatrix} 2 & 3 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \\ 17 & 20 & 24 \end{bmatrix}$$

$$A_r = \begin{bmatrix} 2 & 3 & 5 & 10 \\ 6 & 7 & 8 & 21 \\ 9 & 10 & 11 & 20 \end{bmatrix}$$

$$A_f = \left[\begin{array}{ccc|c} 2 & 3 & 5 & 10 \\ 6 & 7 & 8 & 21 \\ 9 & 10 & 11 & 20 \\ \hline 17 & 20 & 24 & 61 \end{array}\right]$$

# Illustration of Application to Matrix Operations



# Properties of Checksum Matrices

The full checksum is a type of product code

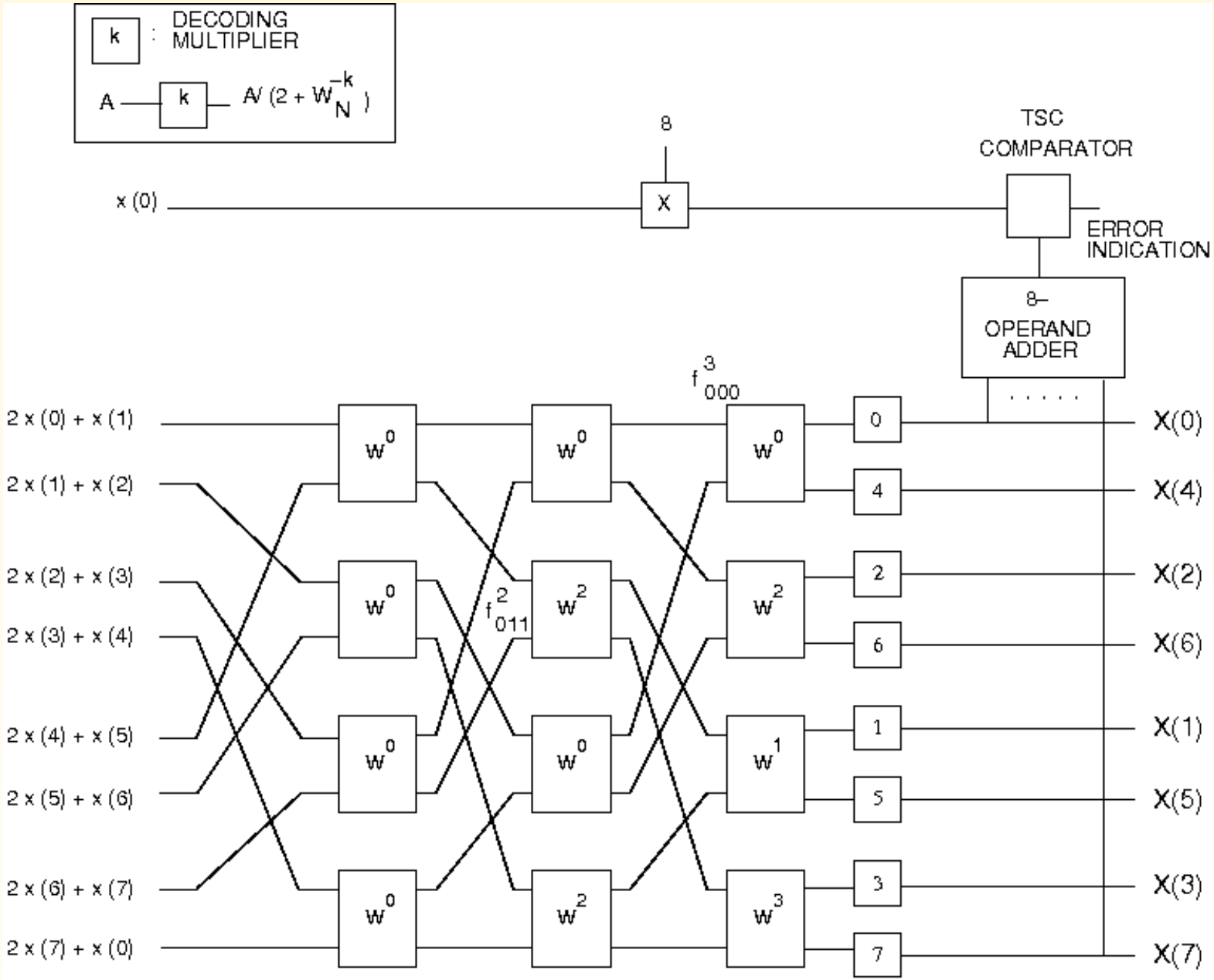
The Matrix Distance between two matrices is defined as the number of elements in which they differ

If two matrices are unequal, their full checksum matrices will have a distance of at least 4

**Checksum Property is Preserved By**

- Matrix multiplication
- Matrix addition
- Scalar product
- Matrix transpose
- LU-decomposition

# Encoding Data to Detect Errors in FFT Networks



# Reliable High-Performance Computing (HPC)

## Issues with checkpointing

- Writing data to stable storage involves high cost
- Diskless checkpointing has been studied as a solution
- "Application–oblivious" checkpointing of message passing applications suffer from scaling issues
- Can incur considerable message passing performance penalties in some cases

## Checkpointing-based applications do not scale

- As the number of processors increases, the overall reliability of the system decreases
- Then, checkpointing-restart is not a viable solution
- Scalable applications require revovery time to decrease as the number of processors increases

Source: Bosilca, Delmas, Dongarra and Langou, 2008.

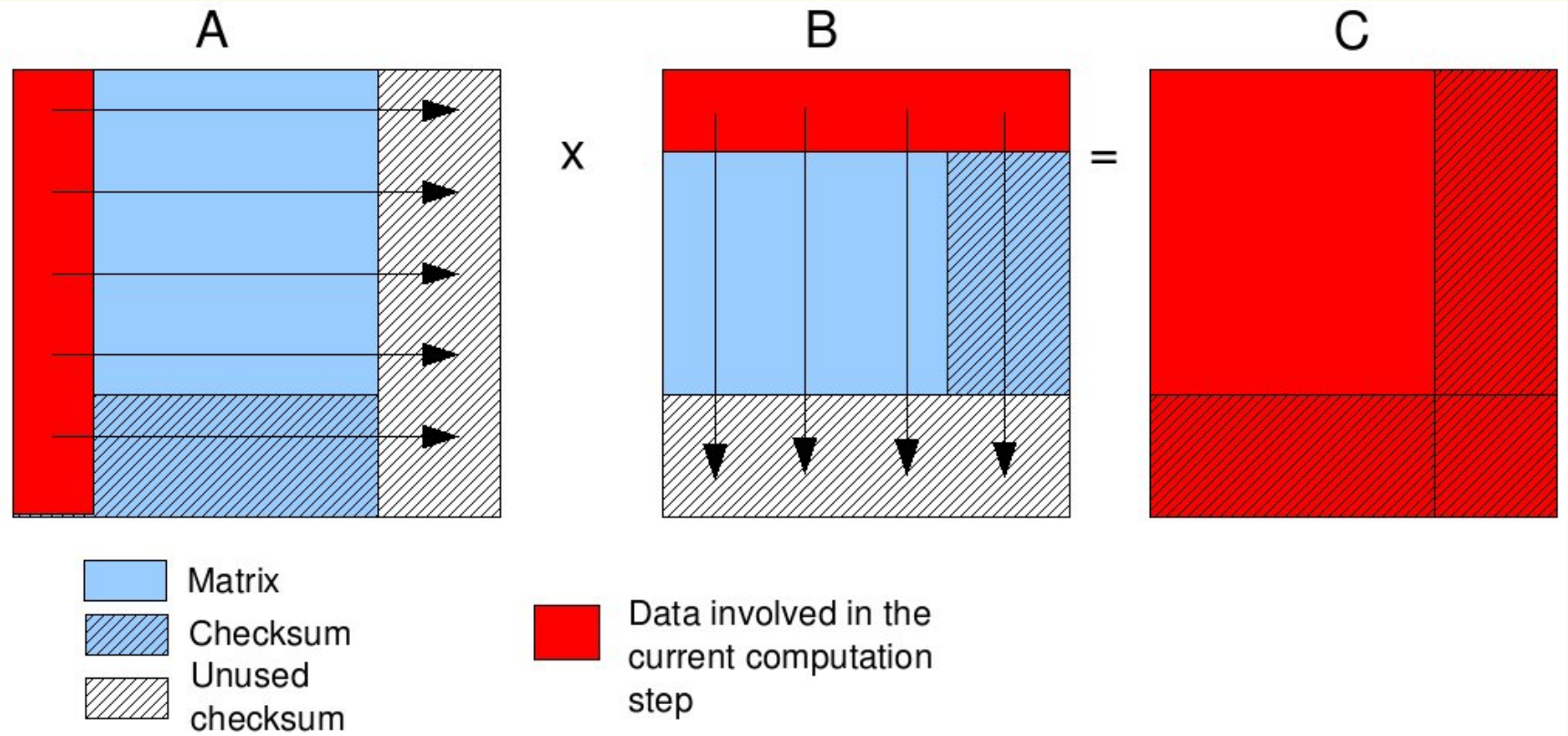
# Use of ABFT in High-Performance Computing

- Algorithm-based fault tolerance has been extended to the parallel distributed context
- The classic algorithm corrects errors at the end of the matrix multiplication operation
  - Not ideal in HPC, where error correction should be done immediately after a failure
- Applied to matrix-matrix subroutine (PDGEMM)
  - Matrix-matrix multiplication is a kernel of fundamental importance to obtain efficient linear algebra subroutines
  - Application does not respond well to memory exclusion techniques, and so standard checkpointing techniques perform poorly
- Can encapsulate all the fault tolerance needed by the linear algebra subroutines in ScaLAPACK in a fault-tolerant *Basic Linear Algebra Subroutines* (BLAS)

Source: Bosilca, Delmas, Dongarra and Langou, 2008.

# Illustration of Checksum Calculations in Linear Algebra Routines

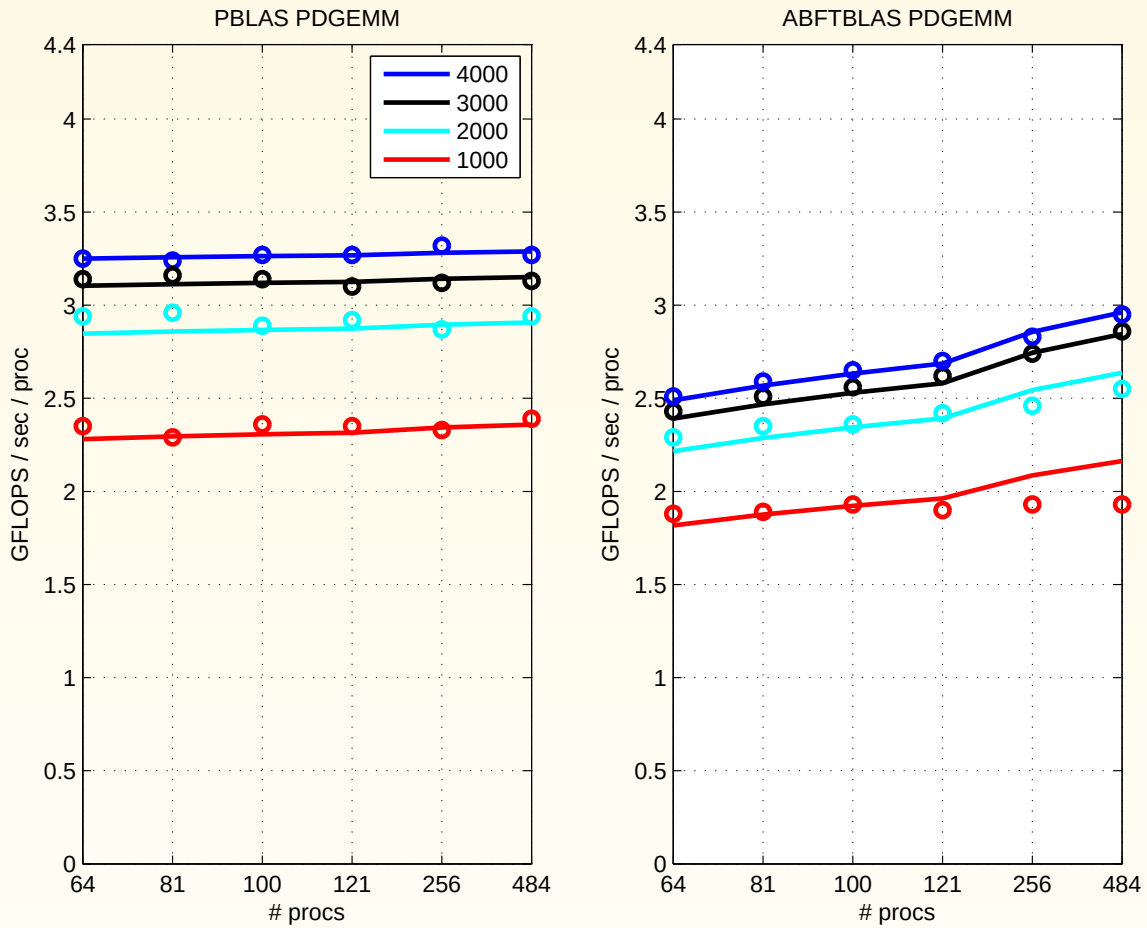
ABFT applied to DGEMM




Source: Bosilca, Delmas, Dongarra and Langou, 2008.

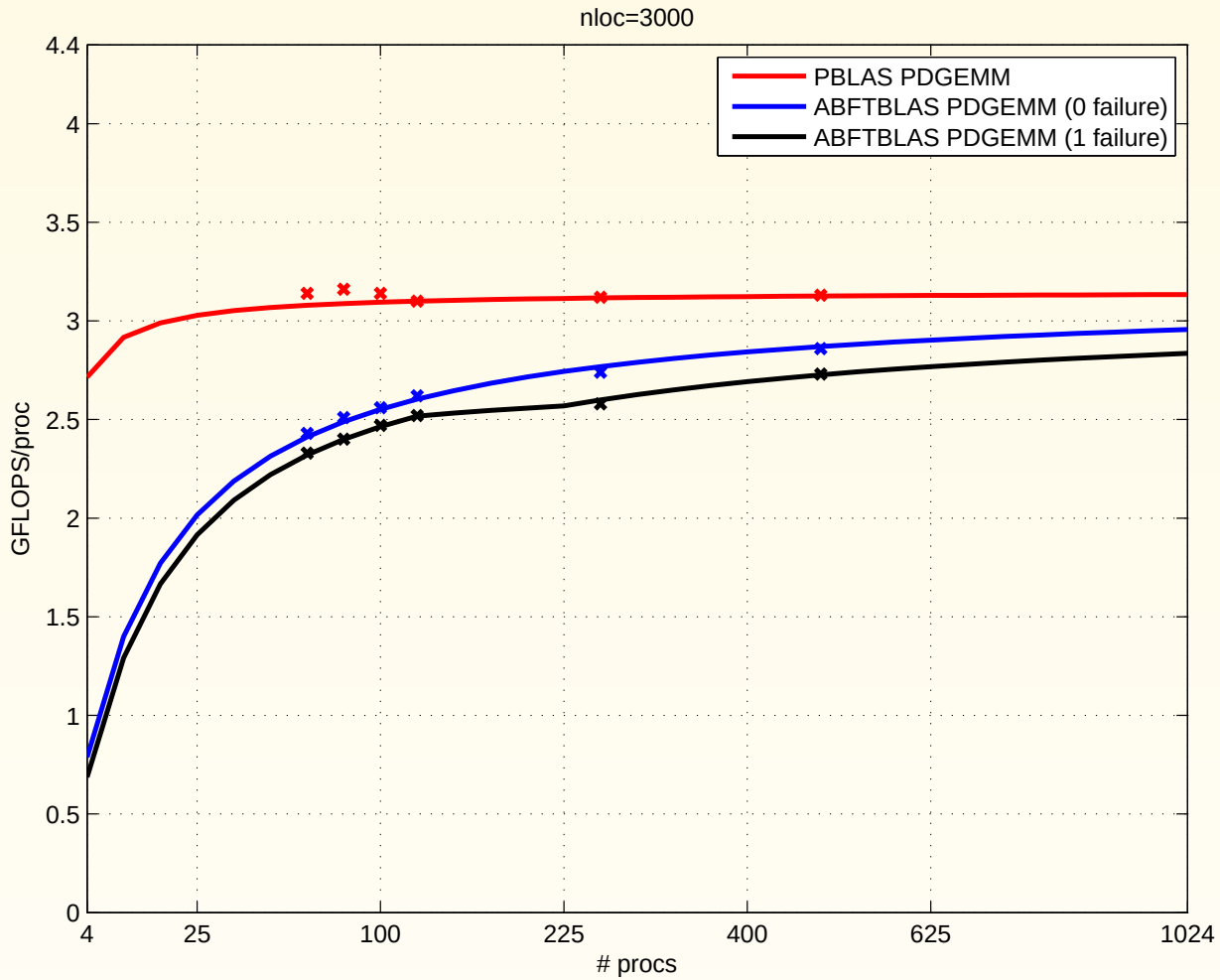
# Performance Comparison

Performance (GFLOPS/sec/proc) of PBLAS PDGEMM (left) and ABFT BLAS PDGEMM with 0 failure (right)
The solid lines represent model while the circles represent experimental points



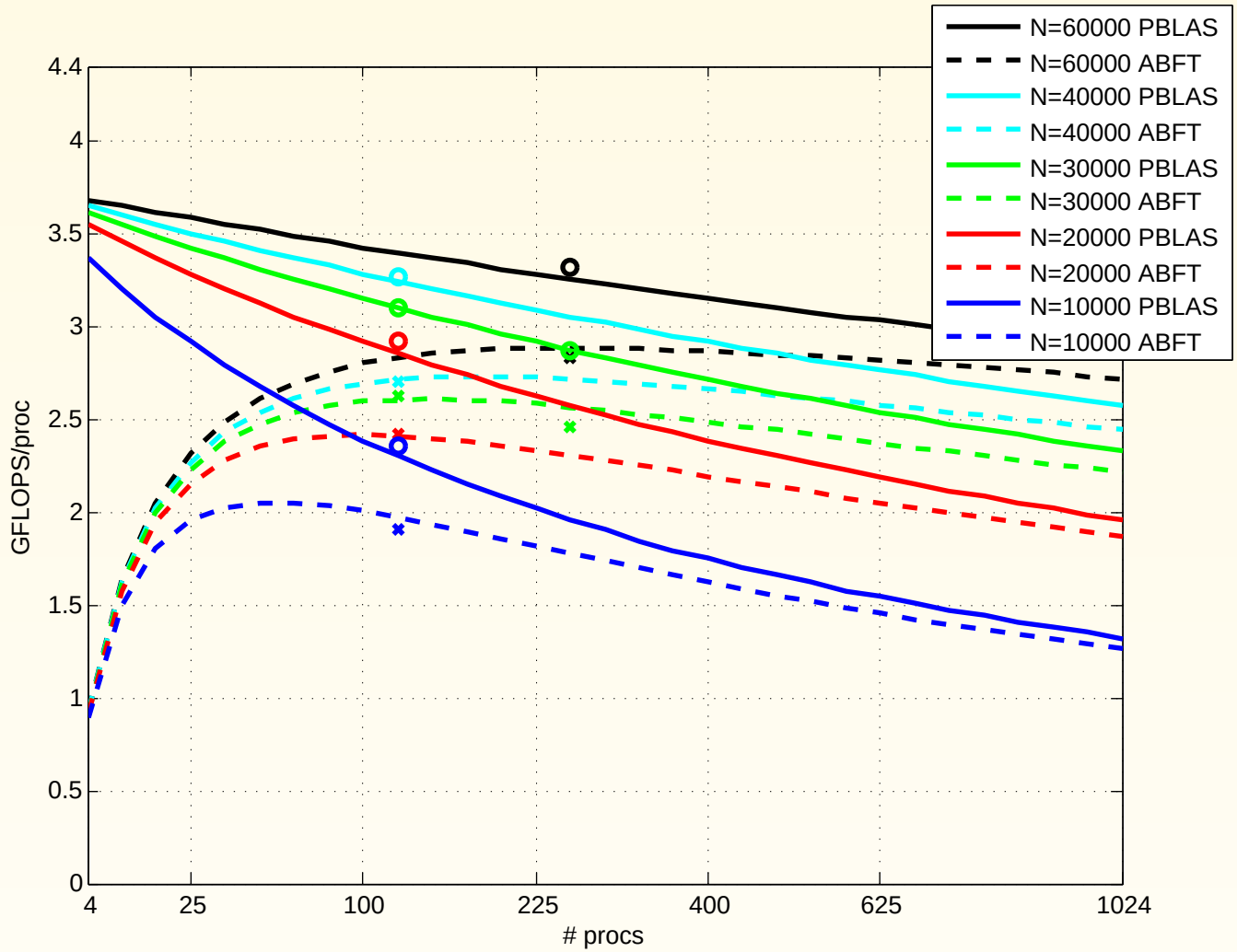
# Performance Under Failure

Performance (GFLOPS/sec/proc) of PBLAS PDGEMM, ABFT BLAS PDGEMM (0 failure), and ABFT BLAS PDGEMM (1 failure)



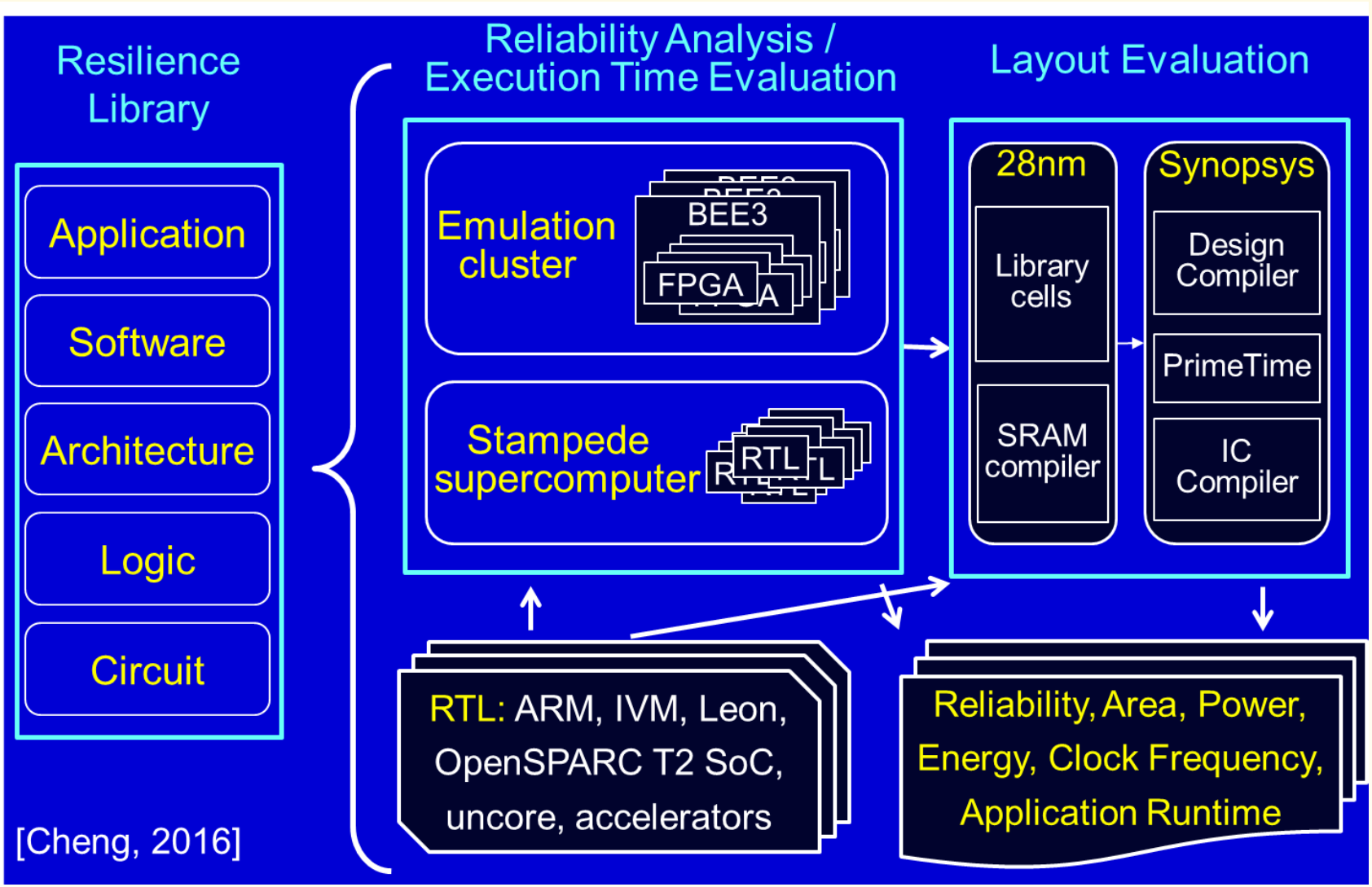
# Performance of PBLAS PDGEMM and ABFT PDGEMM

Performance (GFLOPS/sec/processor) for PBLAS PDGEMM (plain) and performance for ABFT PDGEMM (dashed)



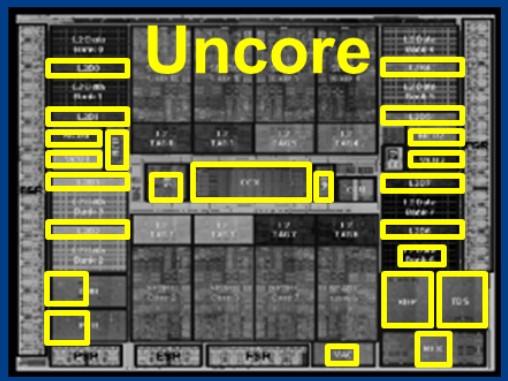
# Cross-Layer Resilience

CLEAR: **C**ross-**L**ayer **E**xploration for **A**rchitecting **R**esilience

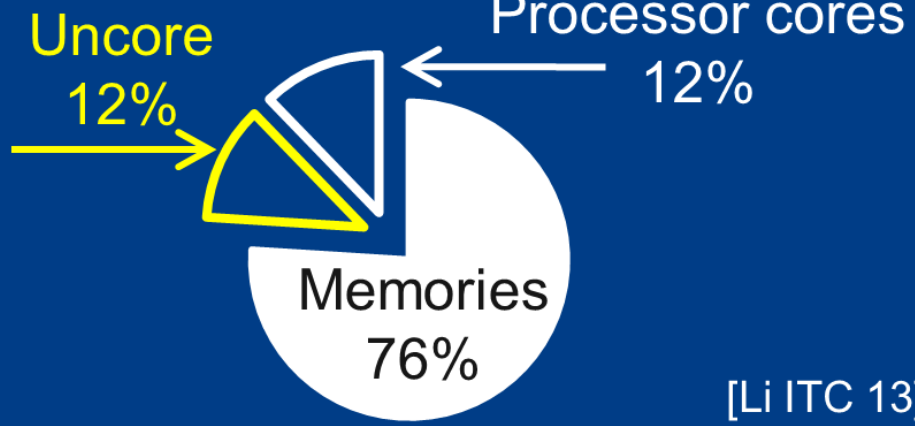


# "Uncore" Components

OpenSPARC T2 SoC

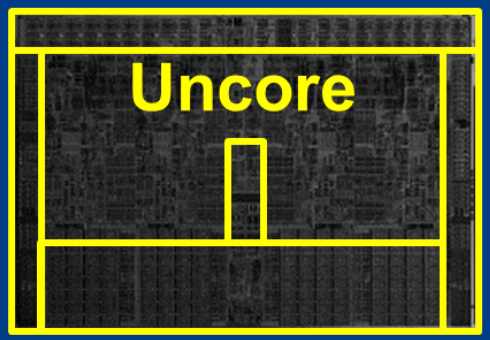



**Area**




[Li ITC 13]

Intel i7 quad-core SoC



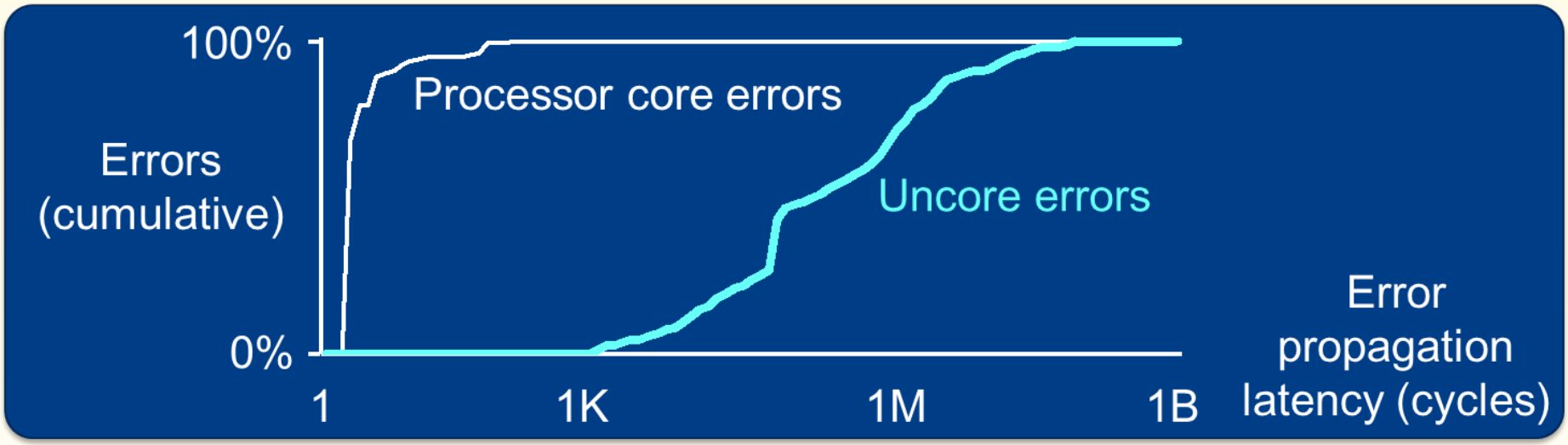

**Power**

| Processor cores 60.2% | Uncore 39.8% |
|---|---|

[Gupta USENIX 12]

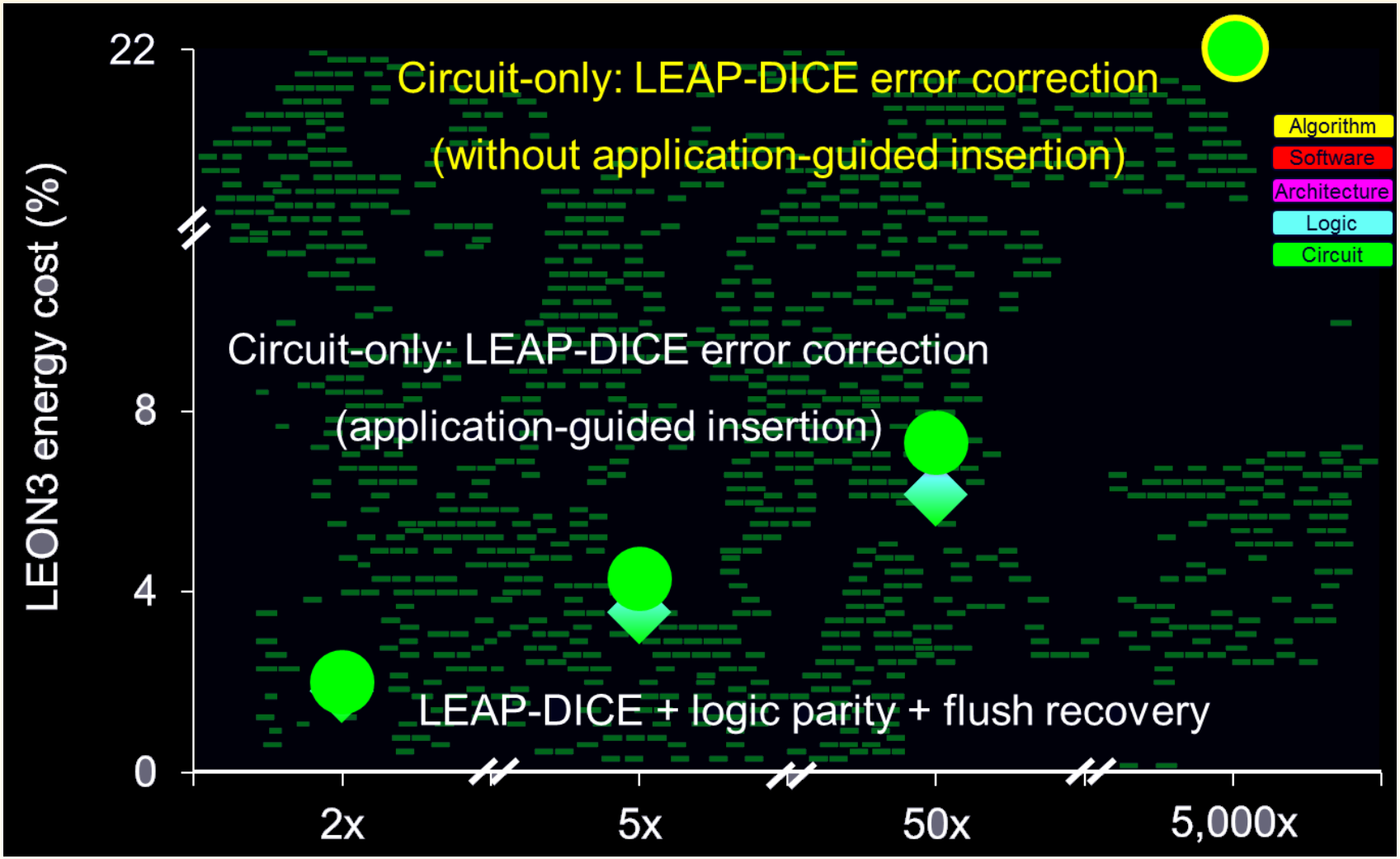
# Uncore Soft Errors: First Extensive Study

- Fast and accurate error injection
  - 20,000X speedup vs. RTL
- Reliability impact: similar for processor cores and uncore portion
  - But, very long error propagation latency

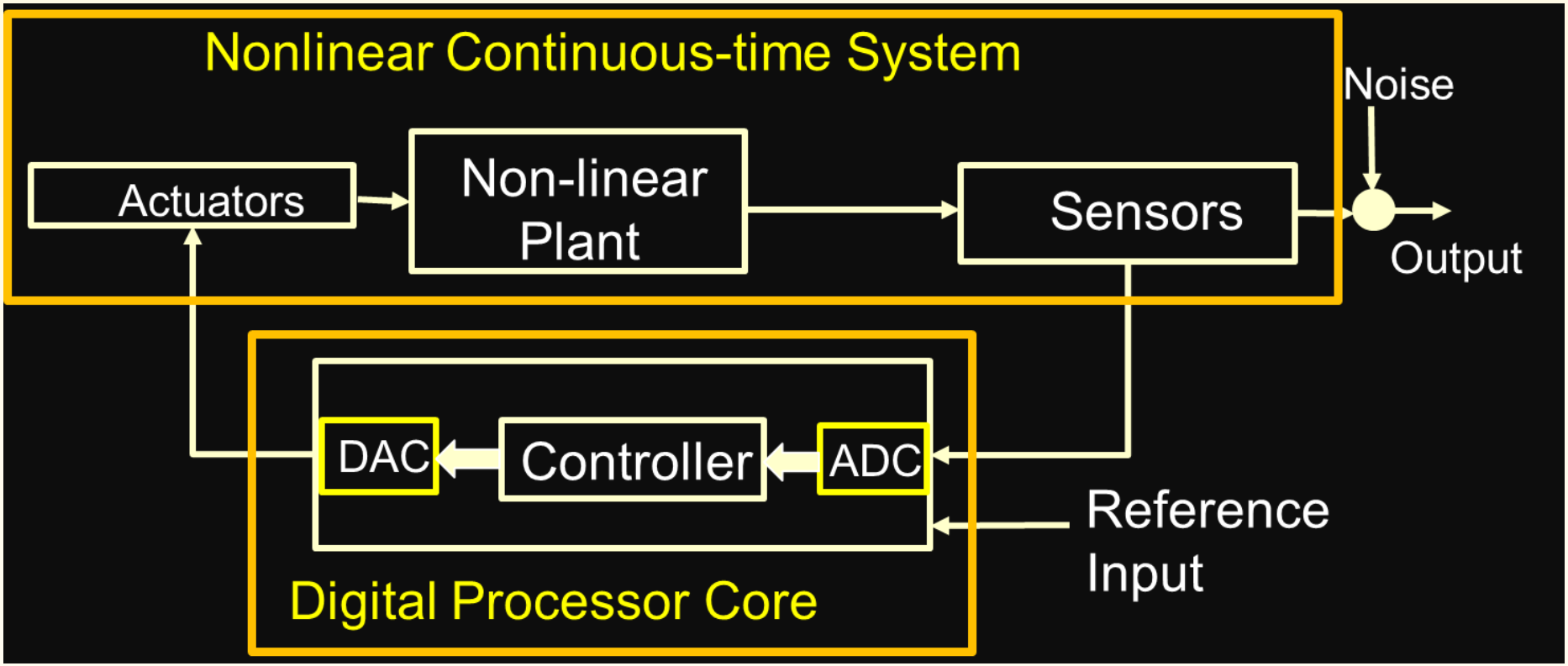



[Cho DAC 15]

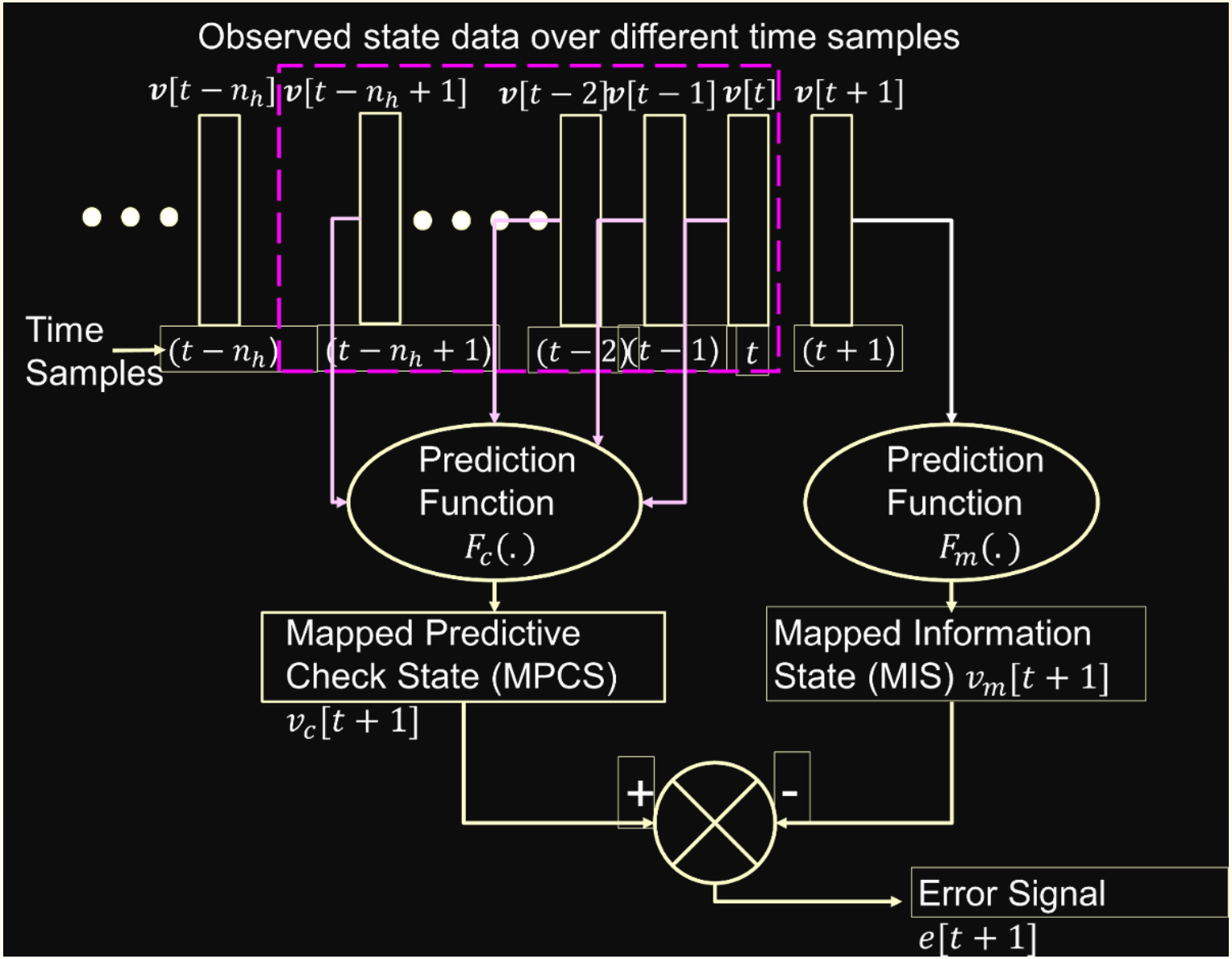
# Circuit-only (Application-guided): Highly Effective



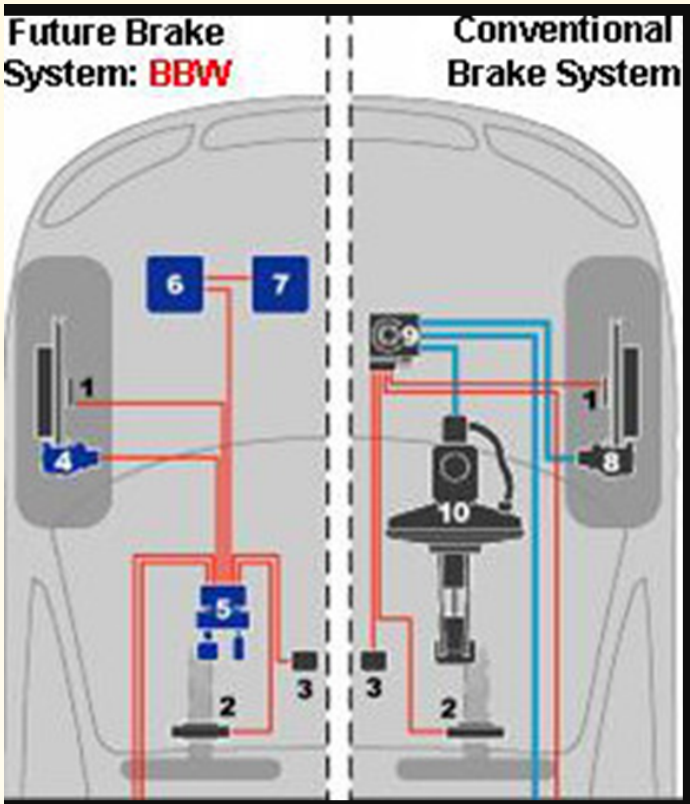
# Resilient Control Systems



# Error Detection Methodology

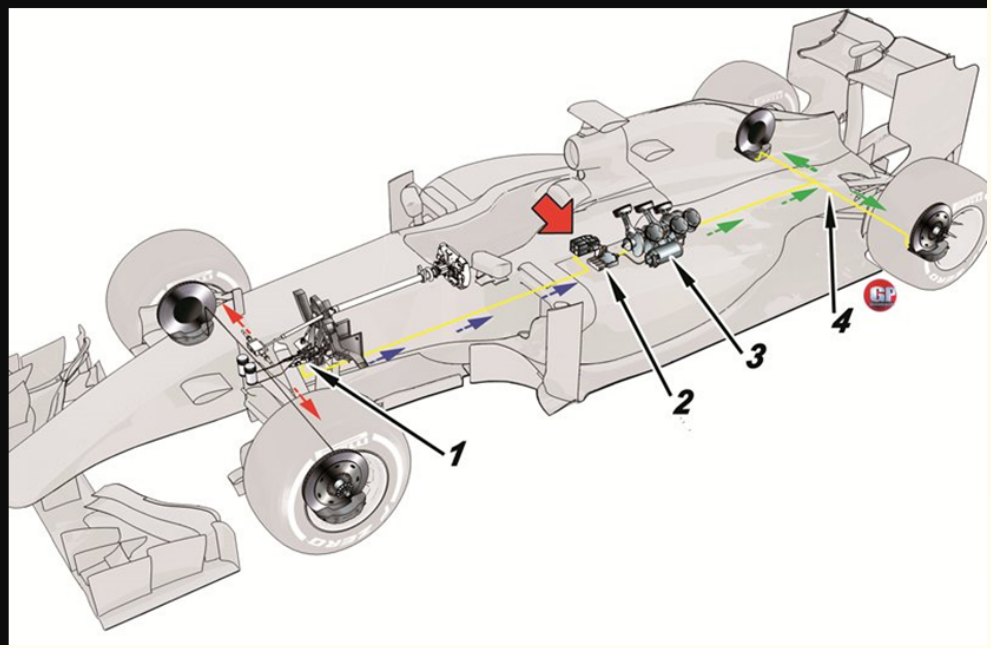


# Brake by Wire



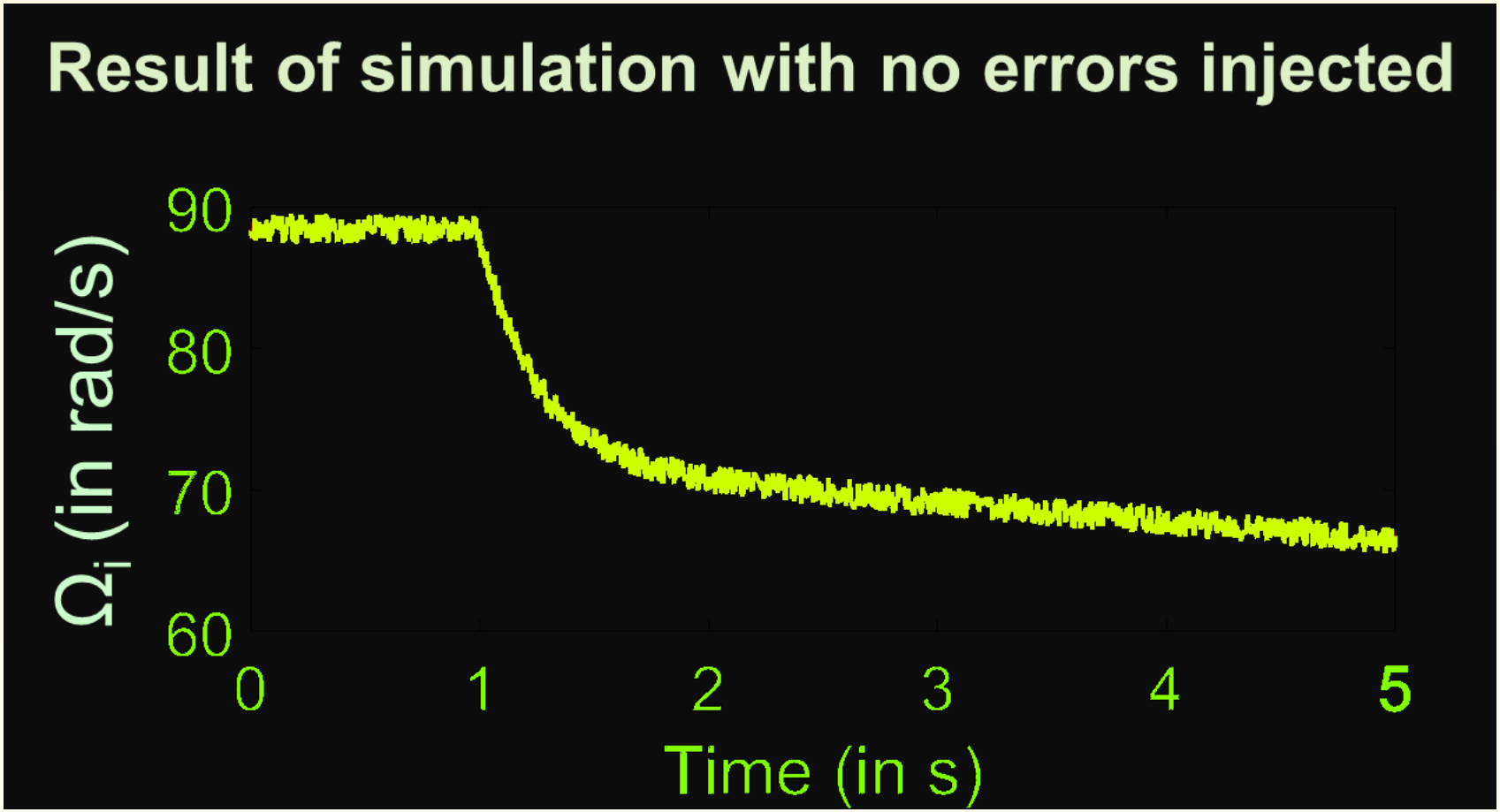

**Lower costs, improved stopping distances**

**Electromechanical brakes (vs. hydraulic**

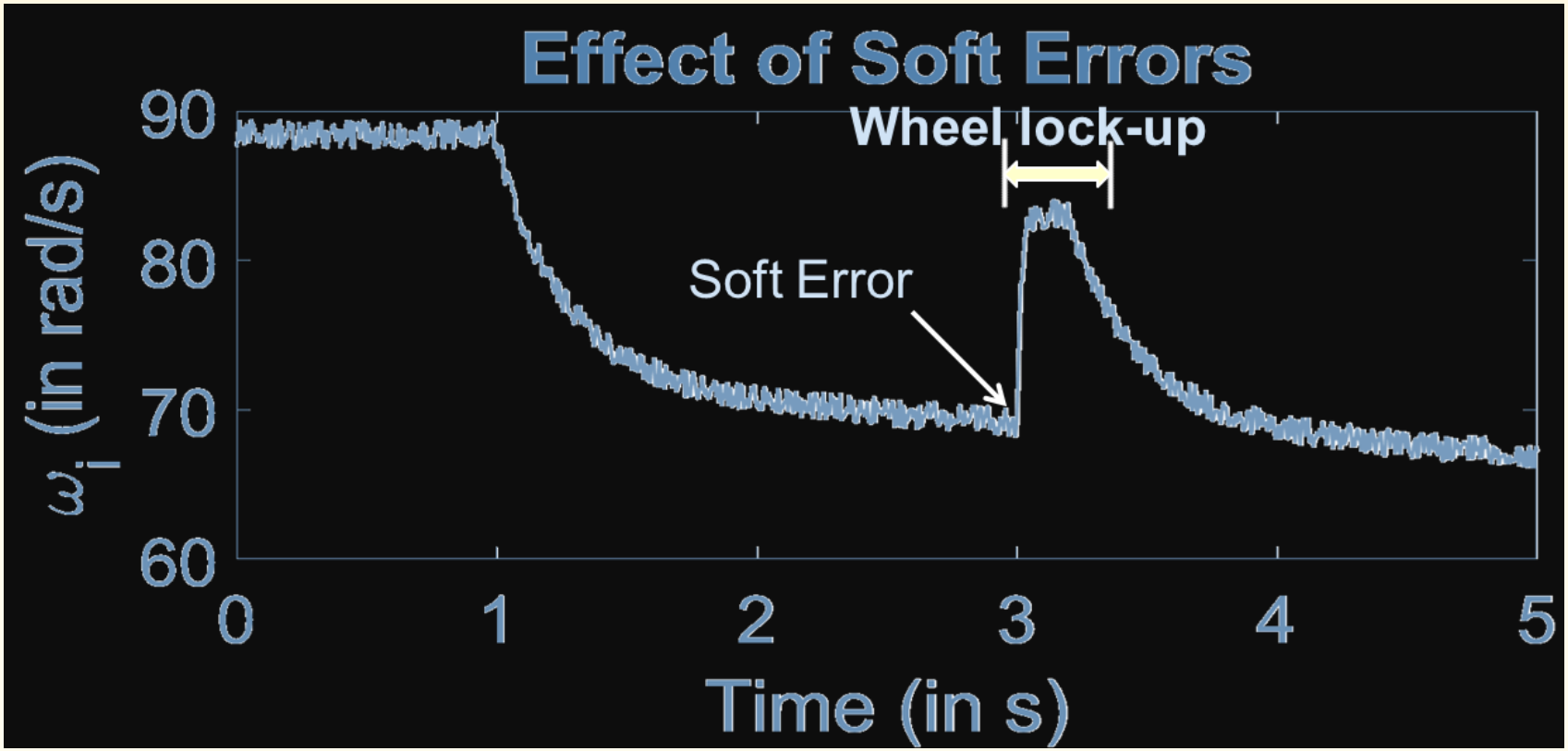


# Error Detection and Correction in Non-Linear Control System

Brake by Wire algorithm executing on embedded processor



# Error Detection and Correction in Brake by Wire System




## Approach to dealing with soft errors

- When error is detected, results of the control loop (output to actuator) ignored for a few cycles till no error is seen.

# Achieving Resilience

## Explore possible solutions – from circuit and logic levels, through architecture and algorithm levels

- Special-purpose solutions are most efficient, and circuit level techniques may be ideal
- For general-purpose systems, application-level techniques may be better
- Software allows flexible solutions at the application level

## Detection is key at the application level

- Detect errors in results of computations
- Application-level results are, ultimately, what are important

## Ensure correct results at the application level

- Appropriate checks at different levels of the design
- High-level checks tend to have lower overheads