OpenSPARC – An Open Platform for Hardware Reliability Experimentation

Ishwar Parulkar and Alan Wood
Sun Microsystems, Inc.
James C. Hoe and Babak Falsafi
Carnegie Mellon University
Sarita V. Adve and Josep Torrellas
University of Illinois at Urbana-Champaign
Subhasish Mitra
Stanford University

IEEE SELSE 4 - March 26, 2008

Outline

1. Chip Multi-threading (CMT)
2. OpenSPARC T2 and T1 processors
3. Reliability in OpenSPARC processors
4. What is available in OpenSPARC
5. Current university research using OpenSPARC
6. Future research directions

World's First 64-bit Open Source Microprocessor

OpenSPARC.net

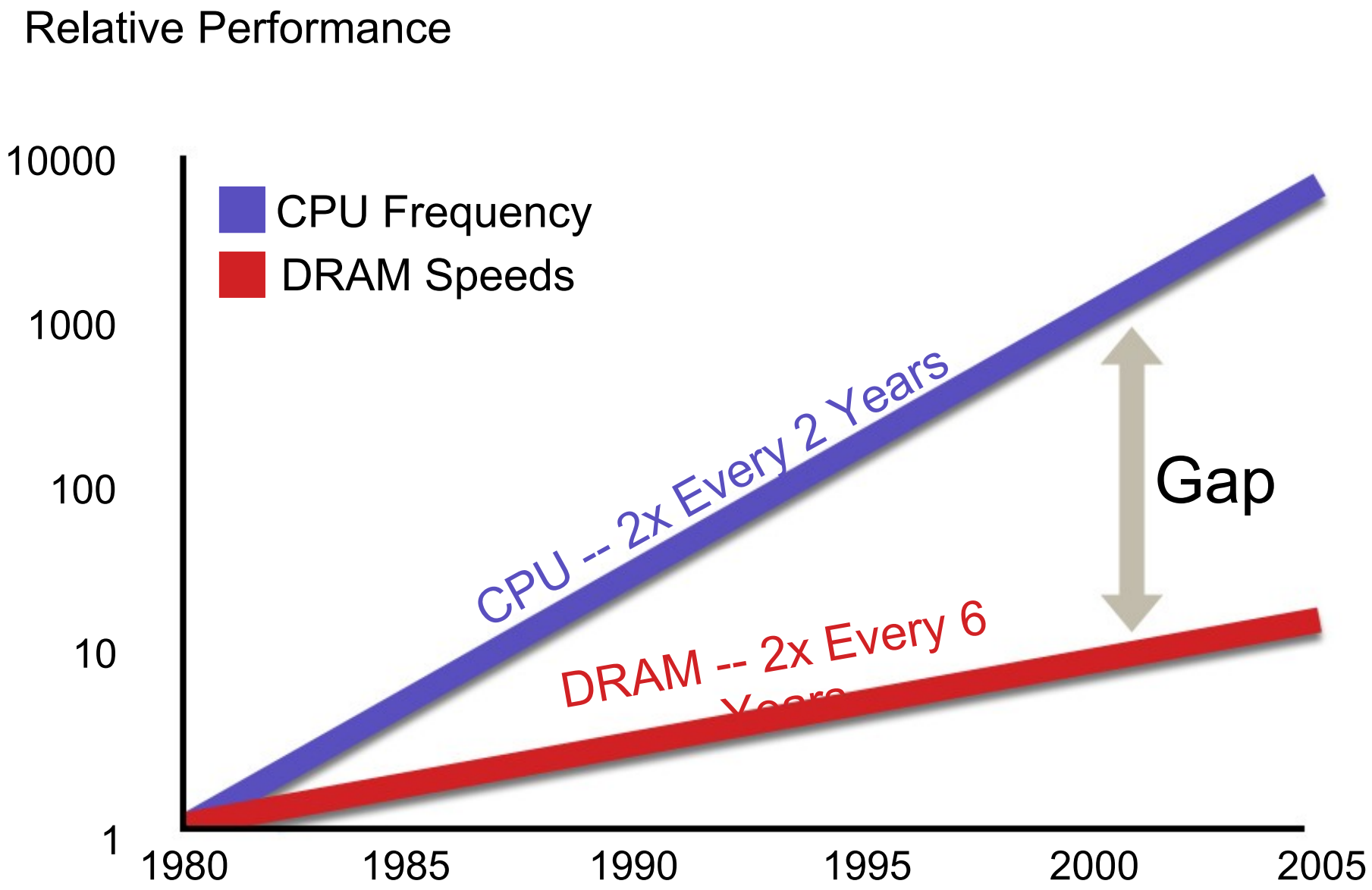
- Governed by GPLv2
- Complete processor architecture & implementation
- Register Transfer Level (RTL)
- Hypervisor API
- Verification suite and architectural models
- Simulation model for operating system bringup on s/w

solaris™ OpenSPARC™

Linux

Chip Multithreading (CMT)

Memory Bottleneck



Source: Sun World Wide Analyst Conference Feb. 25, 2003

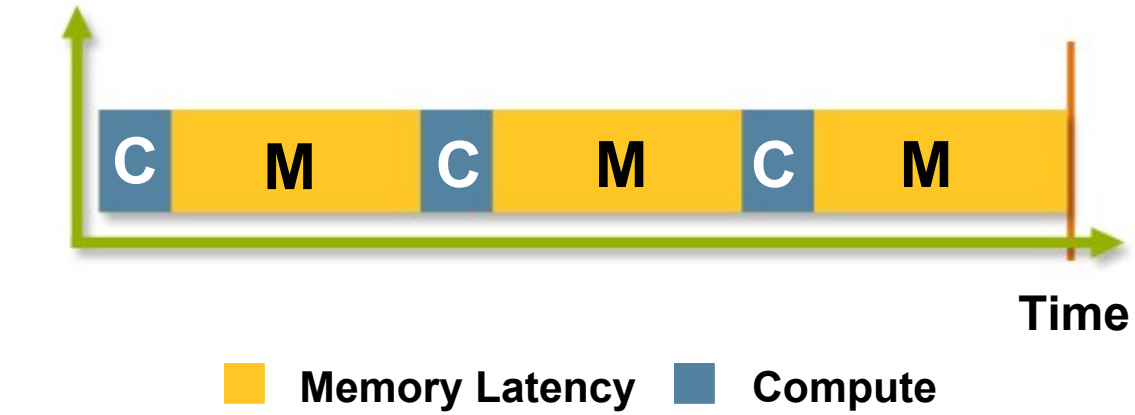
Single Threading

Up to 85% Cycles Waiting for Memory

HURRY UP AND WAIT!

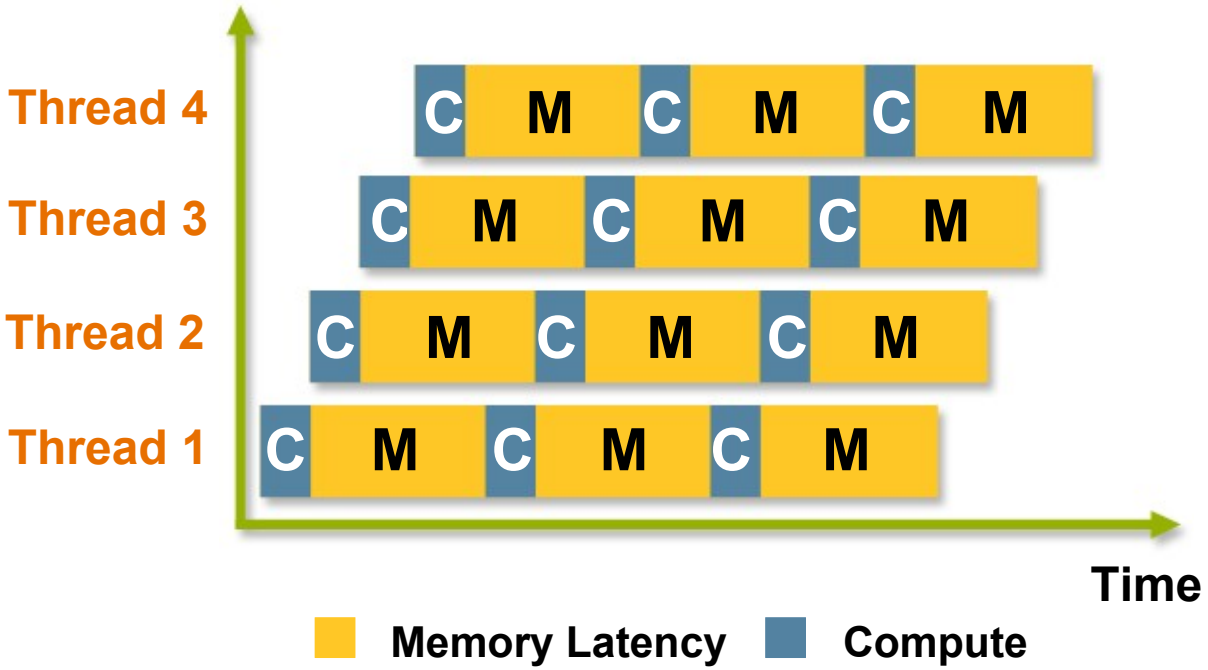
Single Threaded Performance

Typical Processor Utilization:15–25%

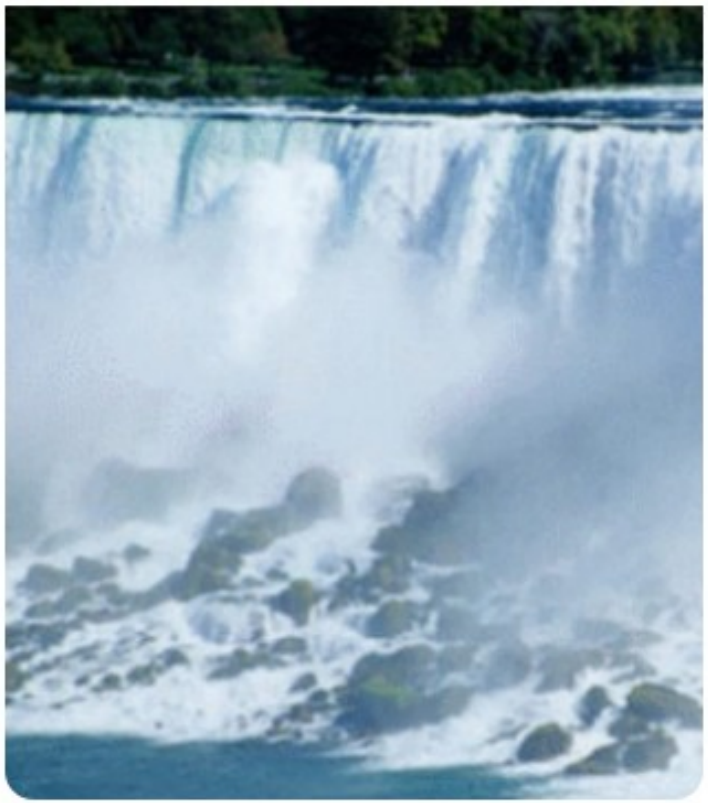


The Power of CMT

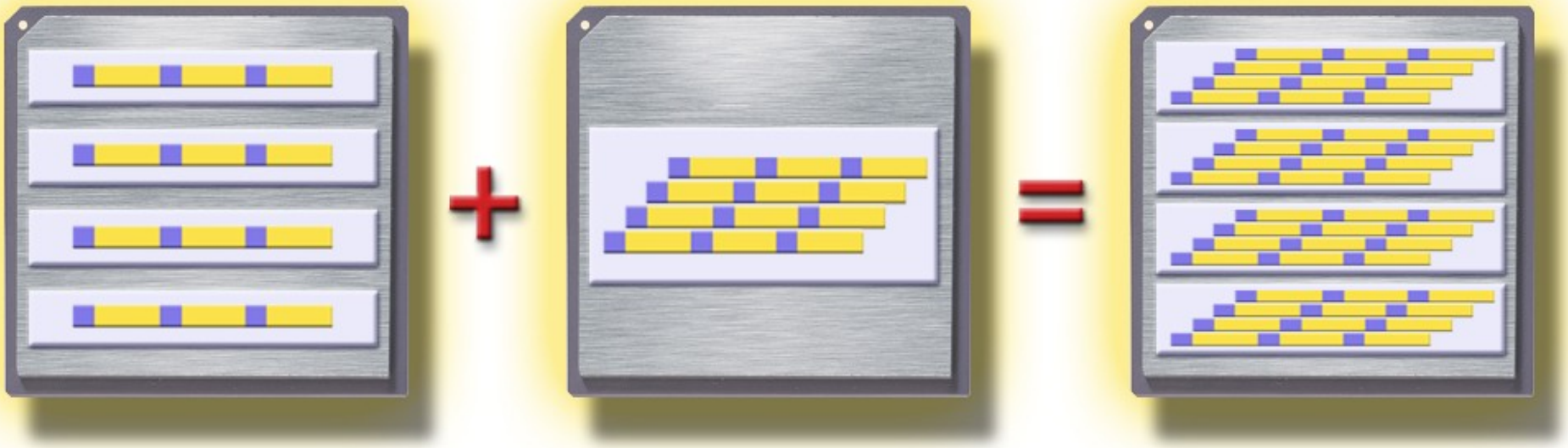
UltraSPARC T1 core Processor Utilization: Up to 85%



Chip Multi-threaded (CMT) Performance



Chip Multi-Threading (CMT)



CMP
(chip multiprocessing)

n cores per processor

HMT
(hardware multithreading)

m threads per core

CMT
(chip multithreading)

n x *m* threads per processor

CMT Paradigm Shift!

CMT technology allows simple, compact system designs, which deliver:

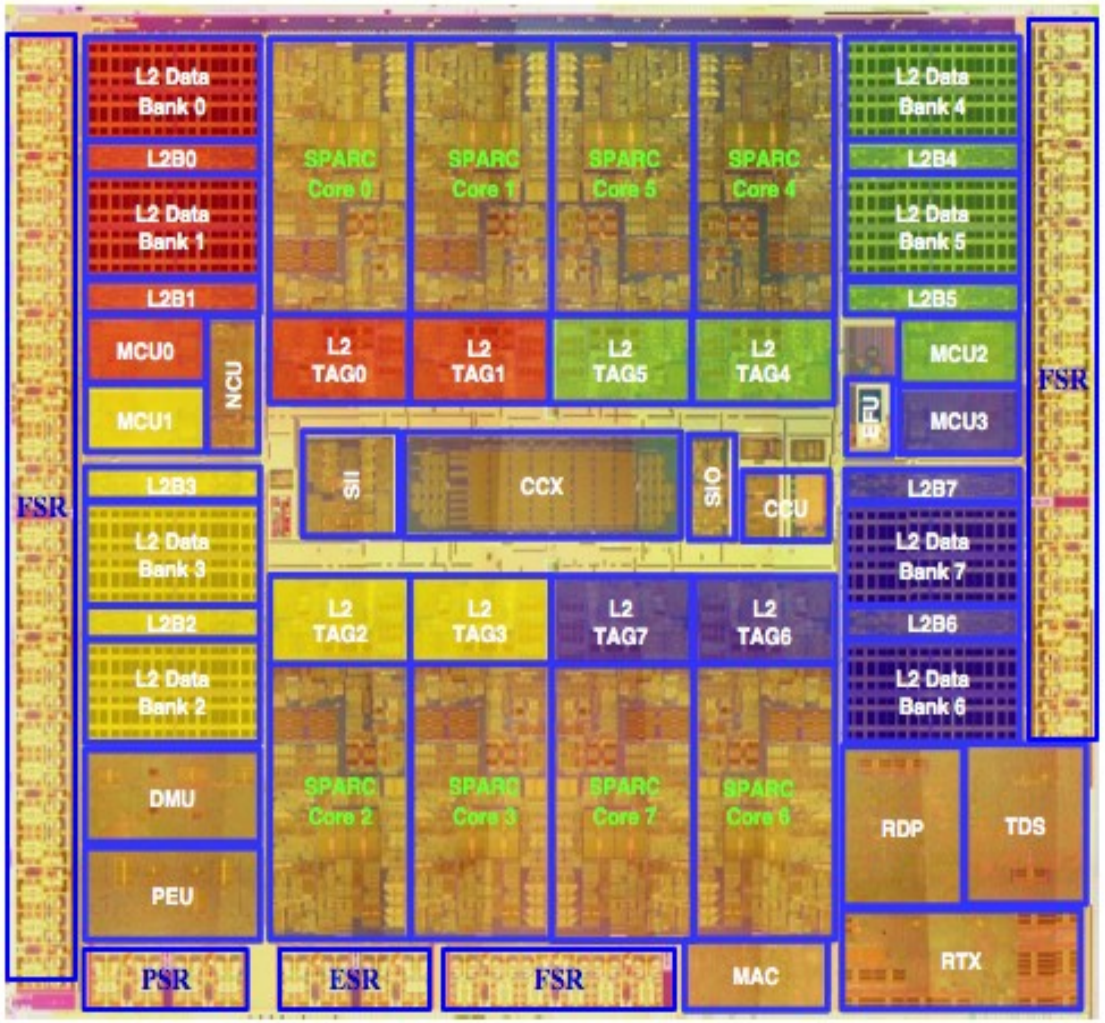
- Higher reliability
- Better performance
- Lower cost
- Faster Installation
- More efficient energy use
- Lower HVAC cost
- Faster time-to-repair
- ... and more

Everybody has changed to multi-core (CMP) and/or chip multi-threaded (CMT) processors:

- Sun(CMT), IBM(CMT), Intel(CMP), AMD(CMP)

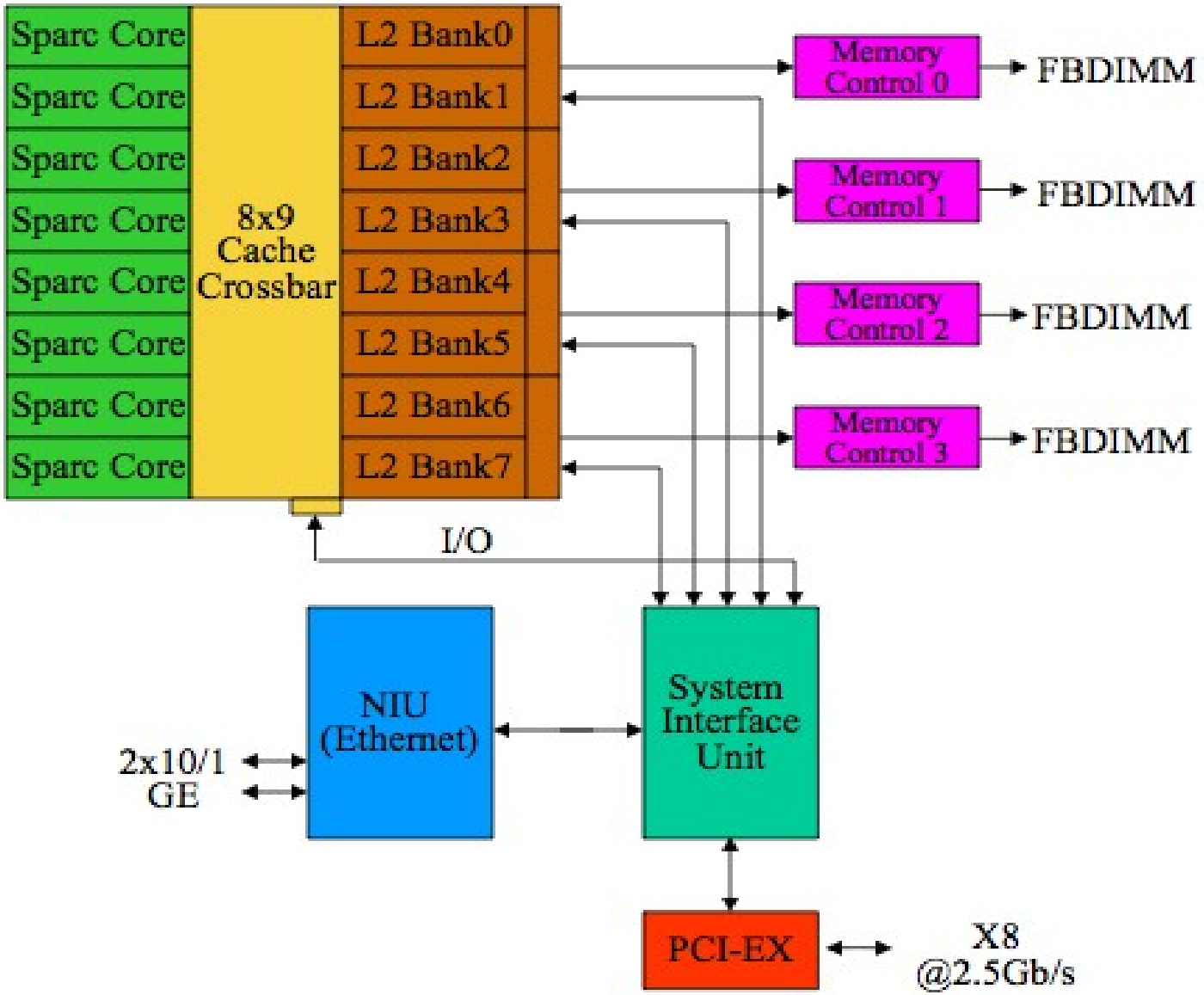
UltraSPARC T2 and T1 CMT Processors

UltraSPARC T2
Die Photo



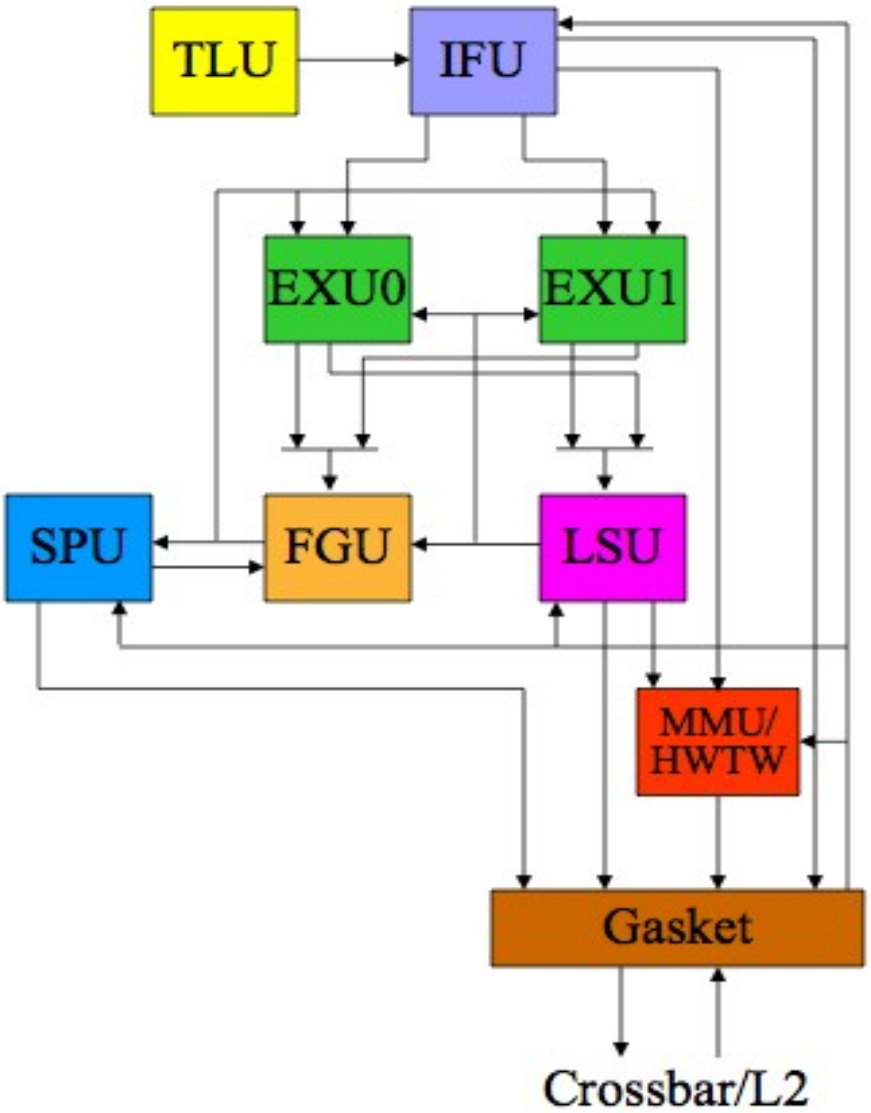
- 8 SPARC cores, 8 threads each
- Shared 4MB L2, 8 banks, 16-way associative
- Four dual-channel FBDIMM memory controllers
- Two 10/1 Gb Enet ports w/onboard packet classification and filtering
- One PCI-E x8 port
- Cryptograhic coprocessor on chip
- 1831 pins, 711 signal I/0
- 342mm^2 die in 65nm

UltraSPARC T2
Block Diagram



UltraSPARC T2

Sparc Core Block Diagram



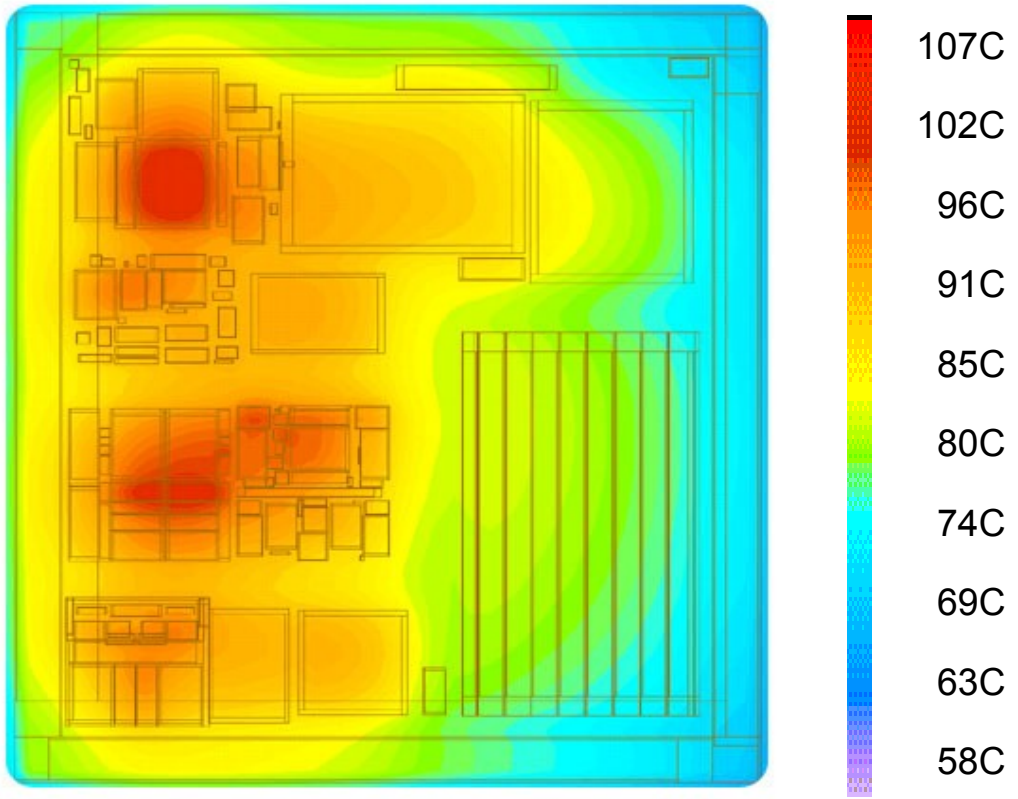
- IFU – Instruction Fetch Unit
  - 16 KB I$, 32B lines, 8-way SA
  - 64-entry fully-associative ITLB
- EXU0/1 – Integer Execution Units
  - 4 threads share each unit
  - 8 register windows/thread
  - 160 IRF entries/thread
- LSU – Load/Store Unit
  - 8 threads share LSU
  - 8KB D$, 16B lines, 4-way SA
  - 128-entry fully-associative DTLB
- FGU – Floating-Point/Graphics Unit
  - 8 threads share FGU
  - 32 FRF entries/thread
- SPU – Stream Processing Unit
  - Cryptographic coprocessor
- TLU – Trap Logic Unit
  - Updates machine state, handles exceptions and interrupts
- MMU – Memory Management Unit
  - Hardware tablewalk (HWTW)
  - 8KB, 64KB, 4MB, 256MB pages

UltraSPARC T2 Reliability

- Extensive error detection and correction
  - Parity protection on I$, D$ tags and data, ITLB, DTLB, CAM and data, modular arithmetic, store address buffer
  - ECC on integer RF, floating point RF, store data buffer, trap stack, L2$ and other internal arrays
- Combination of hardware and software correction flows
  - Hardware re-fetch for I$ and D$
  - Software recovery for other errors
  - Offlining of a thread, group of threads or physical core
- Hardware error injection for verification
- Selective disabling of detection and reporting for bringup

UltraSPARC T2 Reliability Faster Can Be *Cooler* (1)

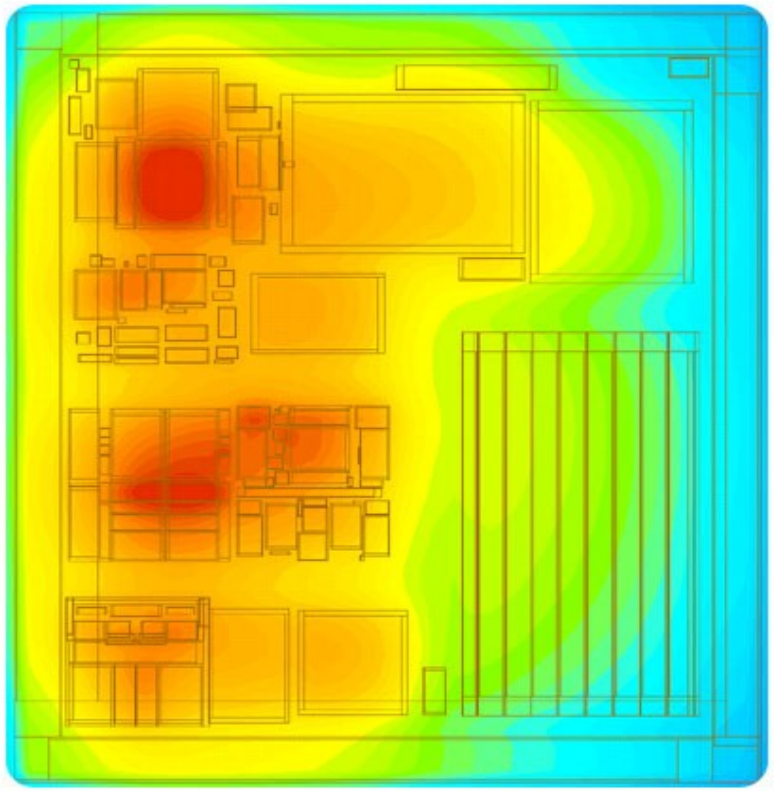
Single-Core Processor

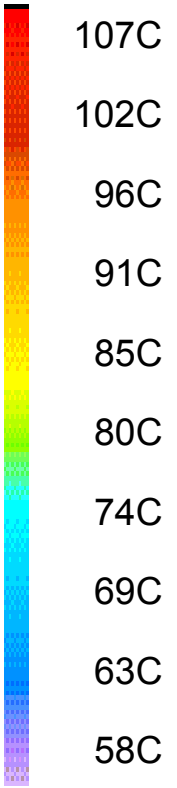


(Not to Scale)

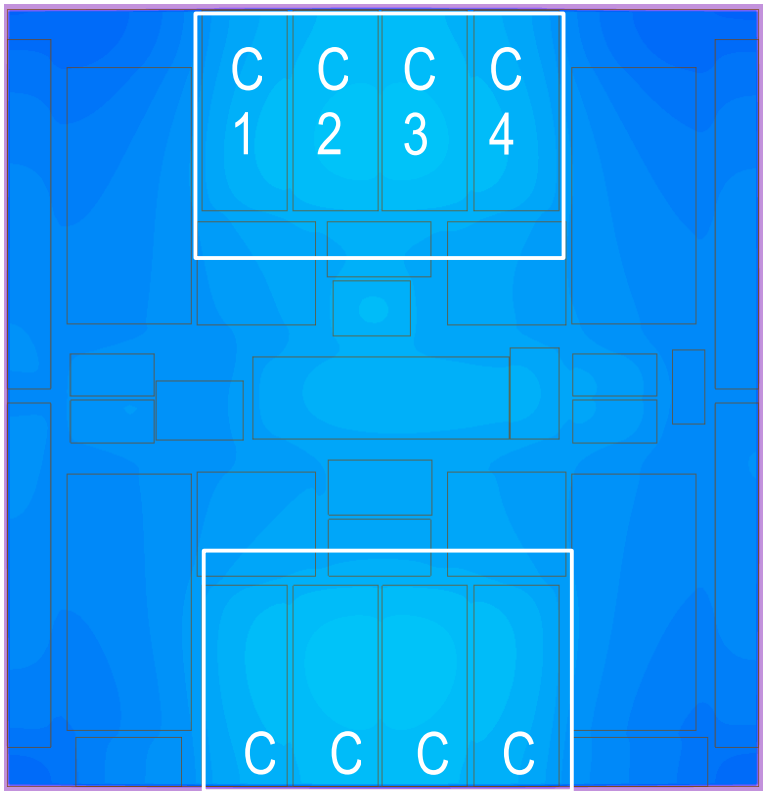
UltraSPARC T2 Reliability
Faster Can Be *Cooler* (2)

Single-Core Processor





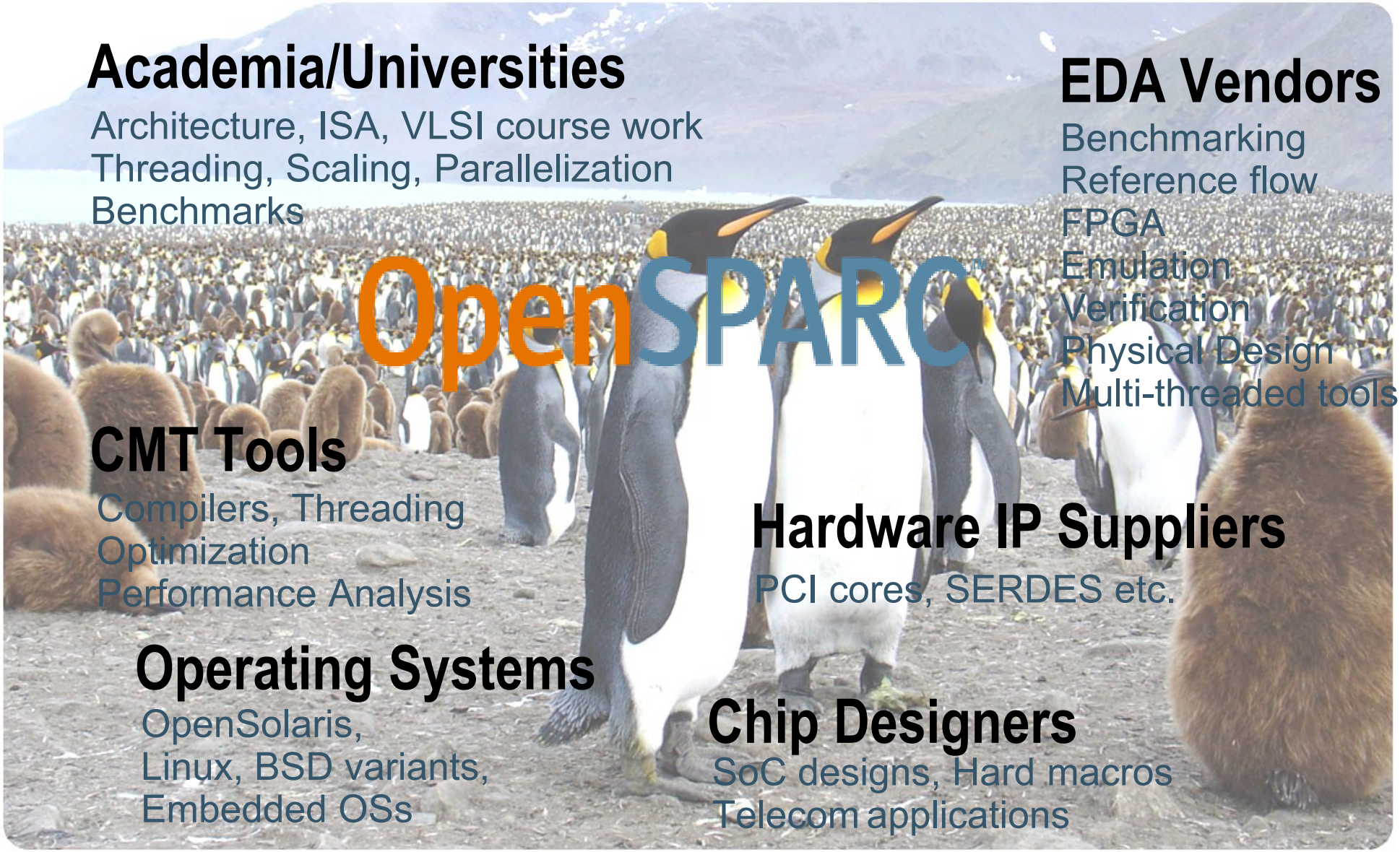
T2 Processor



(Not to Scale)

OpenSPARC

OpenSPARC Communities



What's Available in OpenSPARC 1. Chip design and verification

- UltraSPARC Architecture 2005 spec
- UltraSPARC T2/T1 implementation spec
- Full RTL (Verilog) of OpenSPARC T2/T1
  (8 cores, 64/32 threads – ***more than 4 million lines of code!***)
- Verification test suites
- Full OpenSPARC simulation environment
- Synthesis scripts for RTL
- FPGA implementation support
  - Reduced (to fit capacity), synthesizable version of RTL
  - Synplicity scripts for FPGA synthesis

What's Available in OpenSPARC
2. Architecture and performance modeling

- SAM – SPARC Architectural Model (including source code)
- Legion – Instruction accurate simulator (incl. source code)
- OBP – Open Boot PROM source code
- Hypervisor source code
- Solaris images for simulation
- RST Trace Tool – trace format for SPARC instruction-level traces

What's Available in OpenSPARC
3. Tools for tuning and debug

- ATS – Binary reoptimization and recompilation tool for tuning and troubleshooting applications
- Corestat – Online monitoring of core and FPU utilization
- Discover – Runtime detection of programming errors in allocating and using program memory
- Thread Analyzer – Checking of multi-threaded programming errors such as data races and deadlocks
- More...

What's Available in OpenSPARC 4. Tools for software developers

- Sun Studio 12 – C, C++, Fortran compilers for Solaris/Linux combined with Netbeans, etc.
- BIT – Binary Improvement Tool analyzes and optimizes SPARC binaries for performance and code coverage
- SPOT – produces detailed report on conditions that impact performance of an application
- Source code analysis tool to identify incompatible APIs between Solaris and Linux to speed up migration
- More...

University research in hardware reliability using OpenSPARC

Architectural Fingerprints

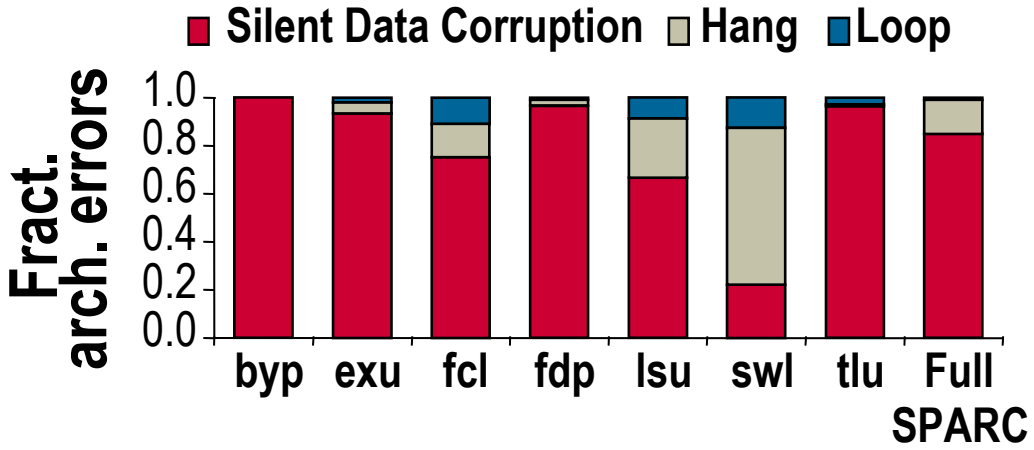
Problem: Error detection for the processor pipeline (soft, wearout, …)

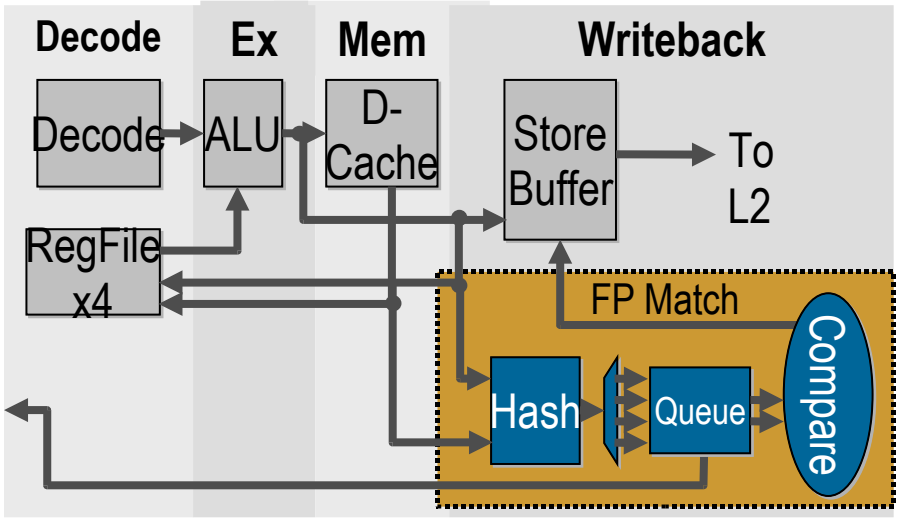
Solution: Architectural fingerprints

- Summarize retiring architectural updates into compact hash (regs, stores)
- Periodically compare hash with reference (another core, previous execution)

Results:

- Multithreaded OpenSPARC T1 RTL implementation — less than 4% area overhead
- Scalable to wide-issue superscalar BW
- Soft fault injection: effective detection for errors propagated to arch. state





FIRST – Detecting Emerging Wearout Faults

Problem: Detecting device wearout during soft breakdown stage

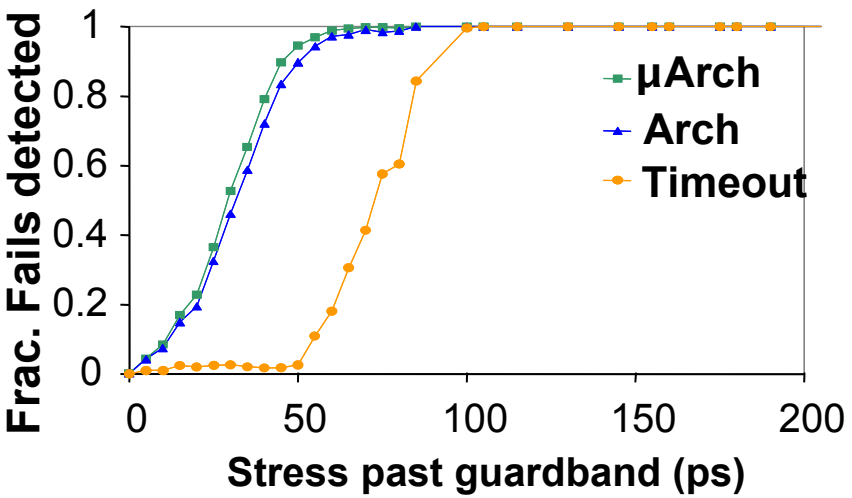
- Faults initially hidden by guardbands & masking

Solution: Periodically test processor cores for signs of growing wearout

- Reduce freq./voltage guardbands until marginal
- Test w/Arch. or µArch. fingerprints
- Observe fails at incr. conservative conditions

Results:

- Wearout fault injection in OpenSPARC
- Arch. and µArch. fingerprints **equivalent** for wide-spread wearout
- µArch. needed for isolated wearout



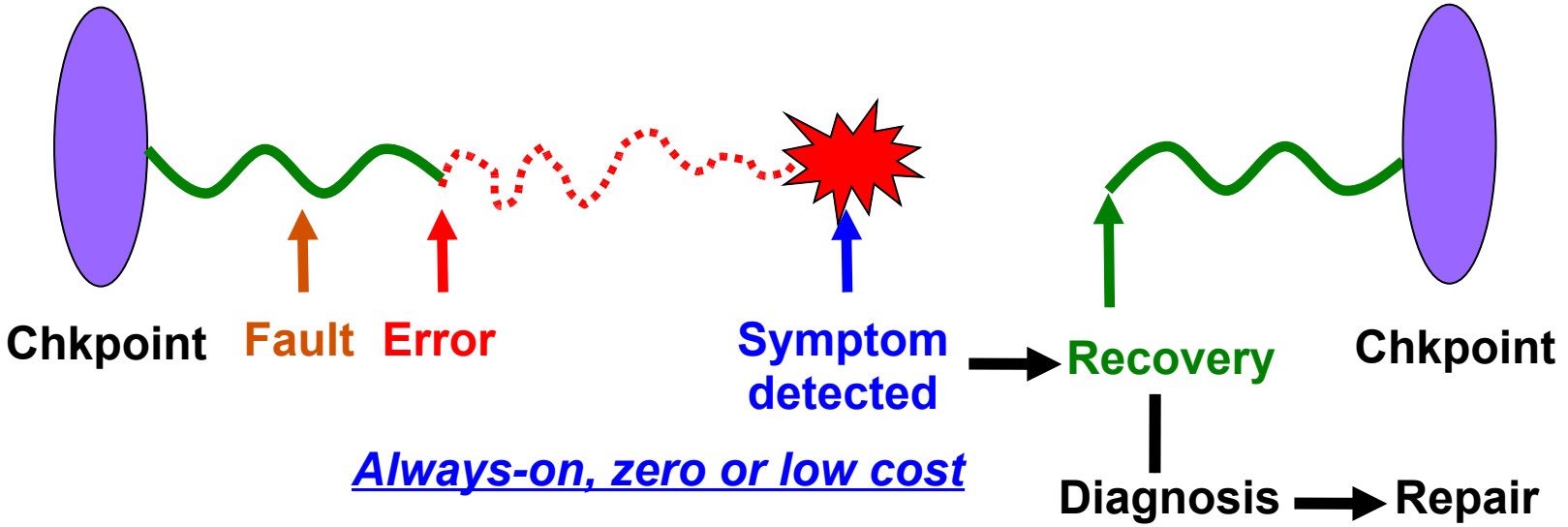
SWAT – SoftWare Anomaly Treatment

Motivation

Low cost solutions needed for in-field detection, diagnosis, recovery and repair for failures due to aging, soft errors inadequate burn-in, design defects, …

SWAT Framework Components

- Detection: Software symptoms, minimal backup hardware
- Recovery: Software/hardware checkpoint and rollback
- Diagnosis: Firmware-controlled rollback/replay on multicore
- Repair/reconfiguration: Redundant, reconfigurable hardware



Always-on, zero or low cost

May have high overhead, rarely invoked

SWAT – Status and Ongoing Work

- Status
  - Detection techniques with > 95% coverage for most structures [ASPLOS'08, SELSE'08, DSN'08]
  - Microarchitecture level, firmware-driven diagnosis with > 97% coverage [SELSE'08, DSN'08]
  - So far, used microarchitecture-level fault injection in simulation
- Ongoing/future work with OpenSPARC
  - Gate-level fault modeling
  - Hypervisor implementation

SWAT – Ongoing Work
High-level fault models and validation

- Goals
  - Understand how gate level faults propagate to microarch & s/w
  - Abstract fault models at microarchitecture level
  - Evaluate reliability solutions and validate results
- Methodology
  - Perform fault injections at gate level
  - For better simulation speed
    - Hierarchical integration of microarchitecture level full system simulator with lower-level simulation of faulty unit
    - Using OpenSPARC Verilog model

SWAT – Future Work
Hypervisor implementation

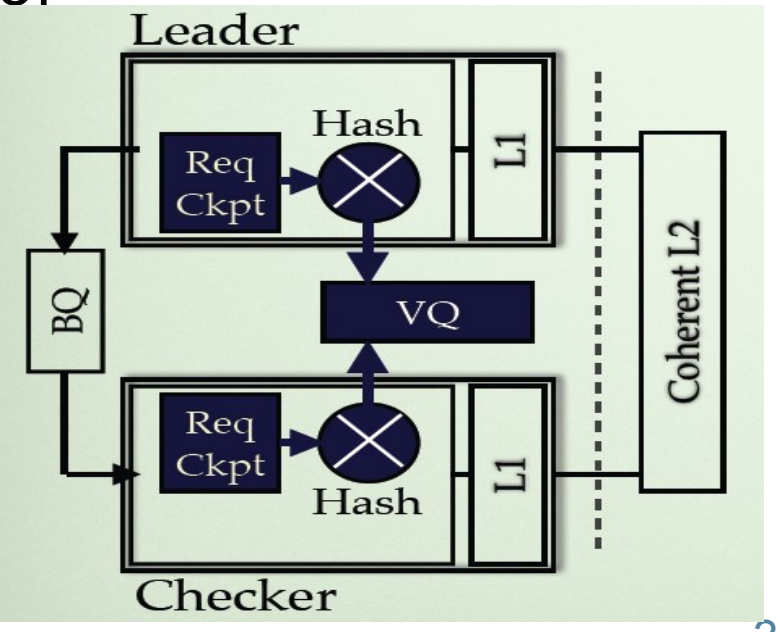
- Plan to use OpenSPARC hypervisor to prototype and evaluate firmware part of SWAT
- Methodology
  - Leverage, extend interface between hypervisor/hardware and hypervisor/OS
  - Extend hypervisor for functionality
  - Use for error detection, recovery, diagnosis, repair

VARIUS – Process Parameter Variation

- Problem:
  - Parameter variation in present and future multicore chips
- Goals:
  - Model parameter variation and resulting timing errors
  - Design multicore microarchitectures to detect and tolerate variation-induced errors
  - Develop new microarchitectural techniques to mitigate variation and variation-induced errors.

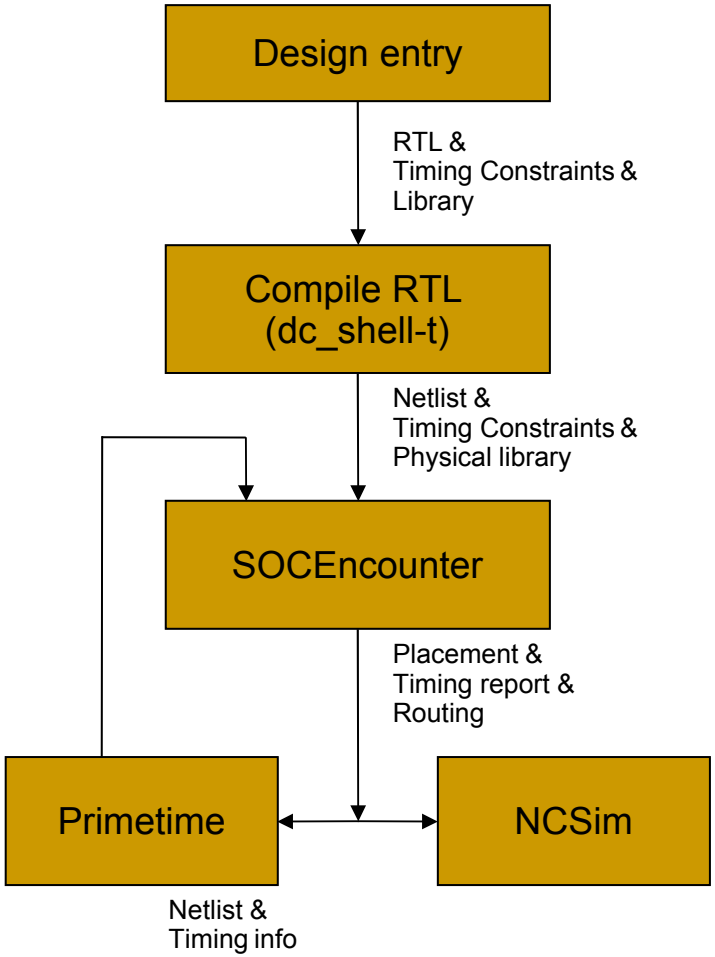
VARIUS – Process Parameter Variation
Accomplishments

- VARIUS model of parameter variation and resulting timing errors for microarchitects [TSM08]
- ReCycle: Pipeline rebalance under process variation [ISCA07]
- Fine-grain adaptive body bias (ABB) to mitigate variation in multicores [MICRO07]
- Workload scheduling and DVFS power management in multicores under variation [ISCA08]
- Paceline: Core pairing for reliability under process variation [PACT07]



VARIUS – Process Parameter Variation
Using OpenSPARC

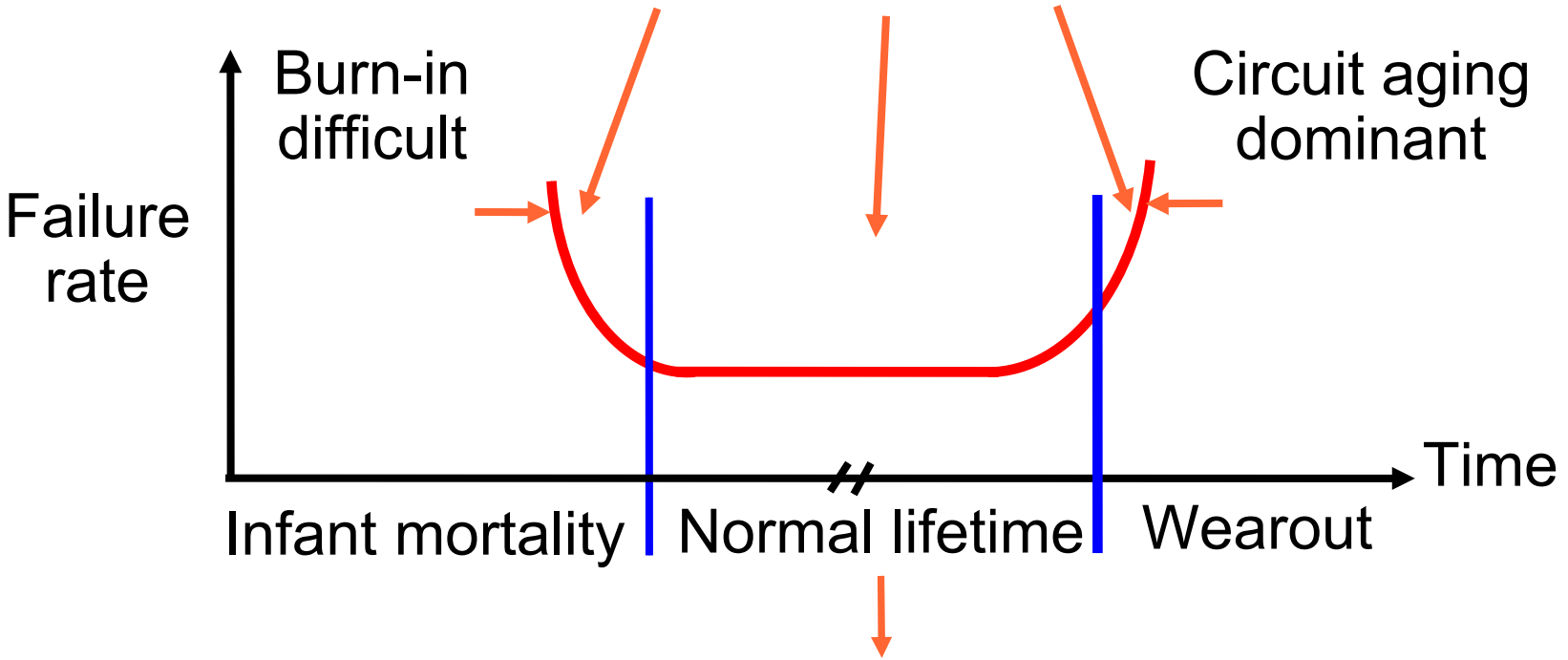
- Goal: Get insights into the effect of parameter variation on a real processor
  - Measure the distribution of the path delays
  - Apply the variation model
- Evaluation Flow:
  - *Synopsys dc_shell-t*:
    - compile RTL to gate-level netlist
  - Cadence SOCEncounter
    - Floorplan, Placement, Routing, Timing analysis
  - Synopsys Primetime
    - Static timing analysis & timing debugging
  - Cadence NCSim
    - Simulation



CASP – Concurrent Autonomous Chip Self-test using Stored Patterns

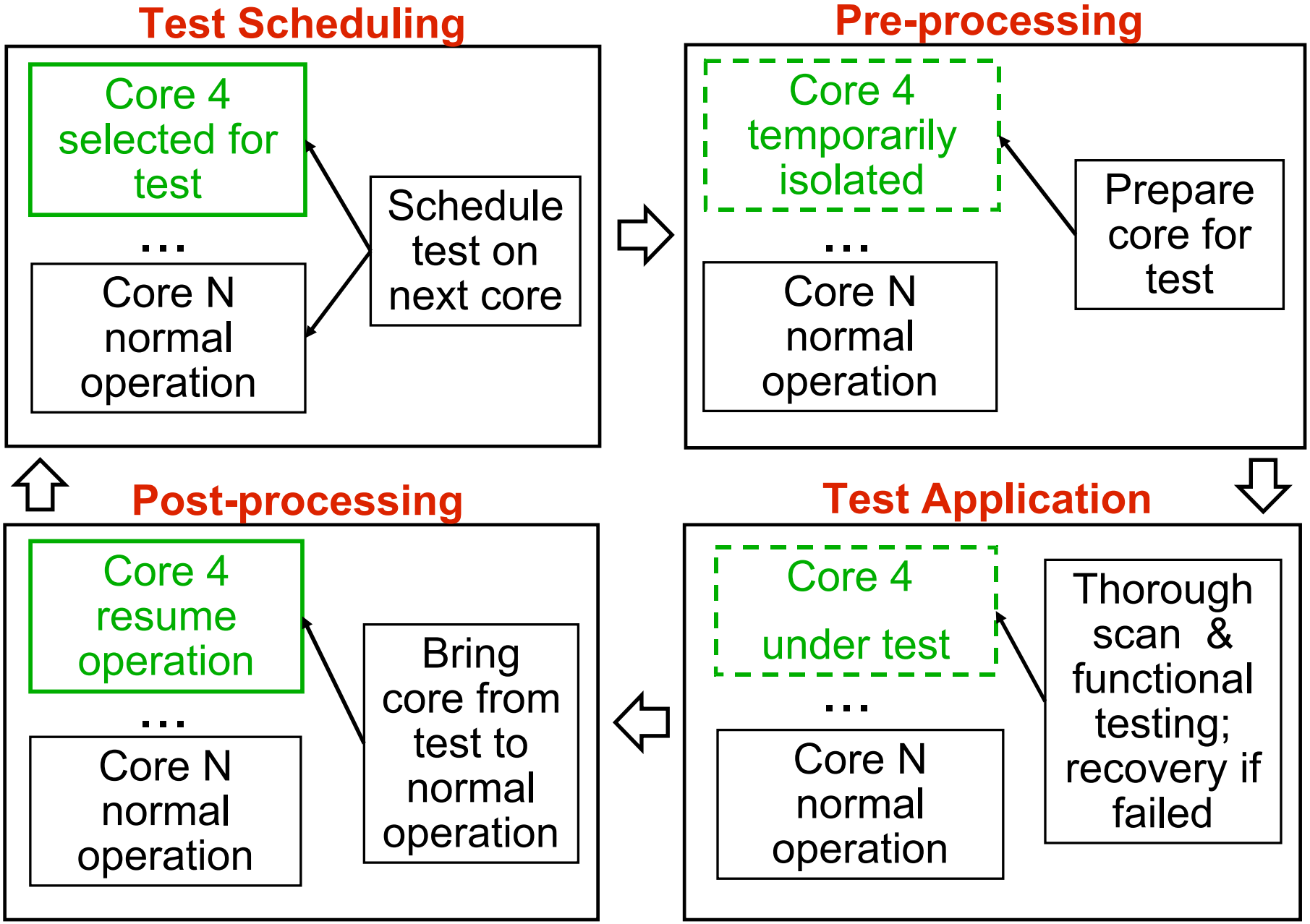
Motivation

Solution:
EXTREMELY THOROUGH
online self-test



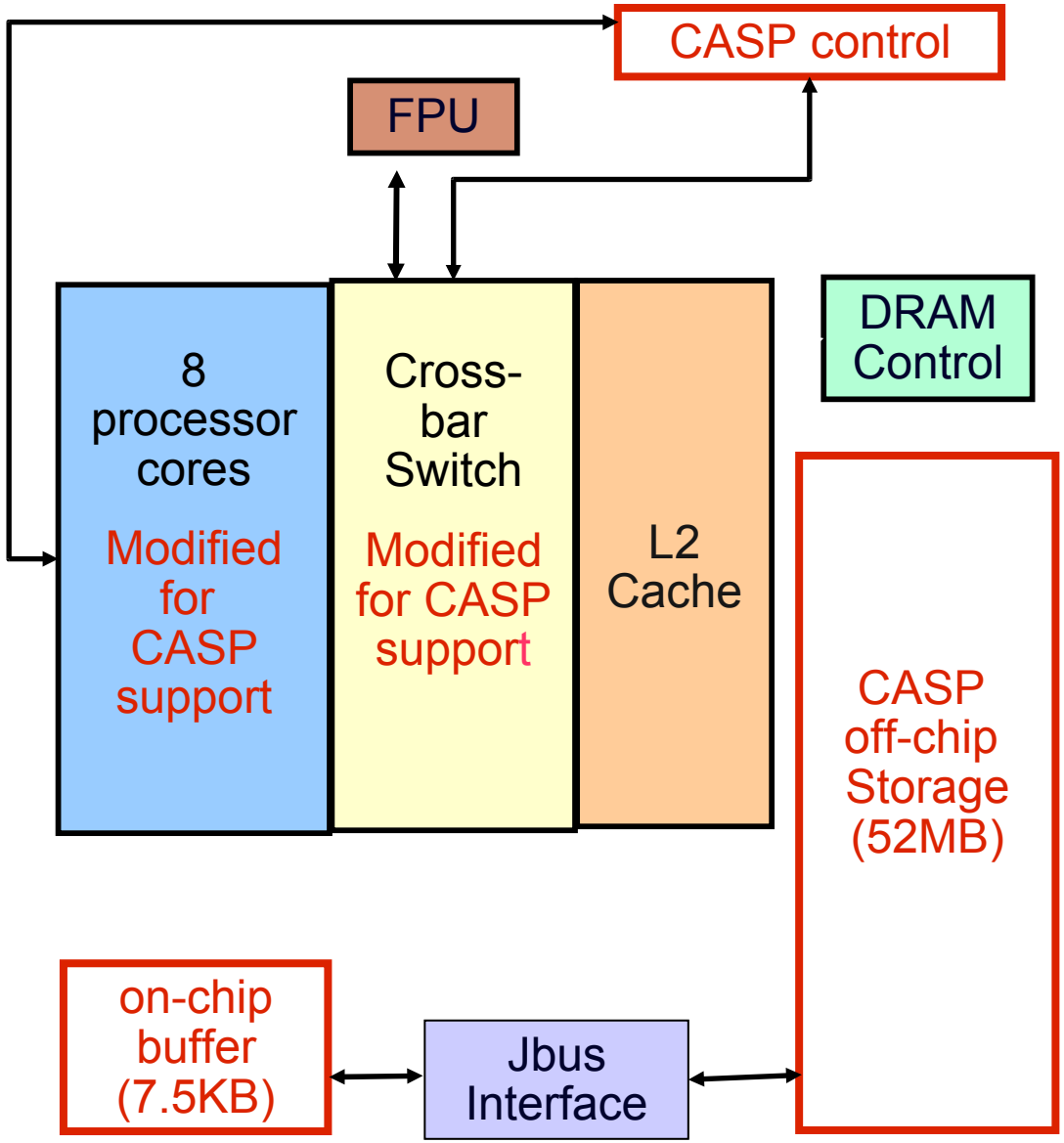
Soft errors: effective techniques exist

CASP – Test Flow



OpenSPARC Modifications for CASP

- CASP Controller
- On-chip buffer for scan test data
- Architectural modfications
  - *Before* a core is tested
    - stalling/draining pipeline
    - disabling communication with core under test
    - saving critical state
    - invalidating D$
  - *After* a core is tested
    - restoring critical state
    - enabling communication with core under test
    - restarting pipeline
- 8000 lines of new Verilog code
- Verification regression used to simulate normal operation of chip



Future research possibilities in hardware reliability using OpenSPARC

Future research possibilities

- Using CMT hardware resources for error detection and recovery
  - cores, threads, structures used by cores/threads
- Understanding errors in the context of CMT architectural constructs
  - thread arbitration and scheduling
  - speculative threading
- Validate error management solutions using a state-of-the-art microprocessor design

Future research possibilities

- Study impact of reliability solutions on microprocessor performance
  - use performance tools available in OpenSPARC
- Firmware and software solutions for hardware reliability
  - FPGA implementation and T1000/2000 servers with Solaris/Hypervisor source and other tools
- Study impact of error detectors in processor on chip level and application failure rates
  - enable error detection selectively, use simulators
- Several more...

Conclusions

- OpenSPARC is an open source community based around UltraSPARC T1 and T2 CMT microprocessors
- OpenSPARC provides a rich, state-of-the-art infrastructure for research in hardware reliability
- Many universities are actively using OpenSPARC in their research, with a lot of success
- There is a lot more research in hardware reliability that can be done using OpenSPARC

Acknowledgment

We would like to acknowledge the ***students*** (past and present) from Carnegie Mellon University, University of Illinois at U-C and Stanford University who contributed to the research described in this presentation.