Workshop on Recent Trends in Procesor Architecture – OpenSPARC

March 17, 2007

Ramesh Iyer

Sr. Engineering Manager

Shrenik Mehta

Senior Director,

Frontend Technologies & OpenSPARC Program

David Weaver

Sr. Staff Engineer, UltraSPARC Architect

Jhy-Chun (JC) Wang

Sr. Staff Engineer

Systems Group

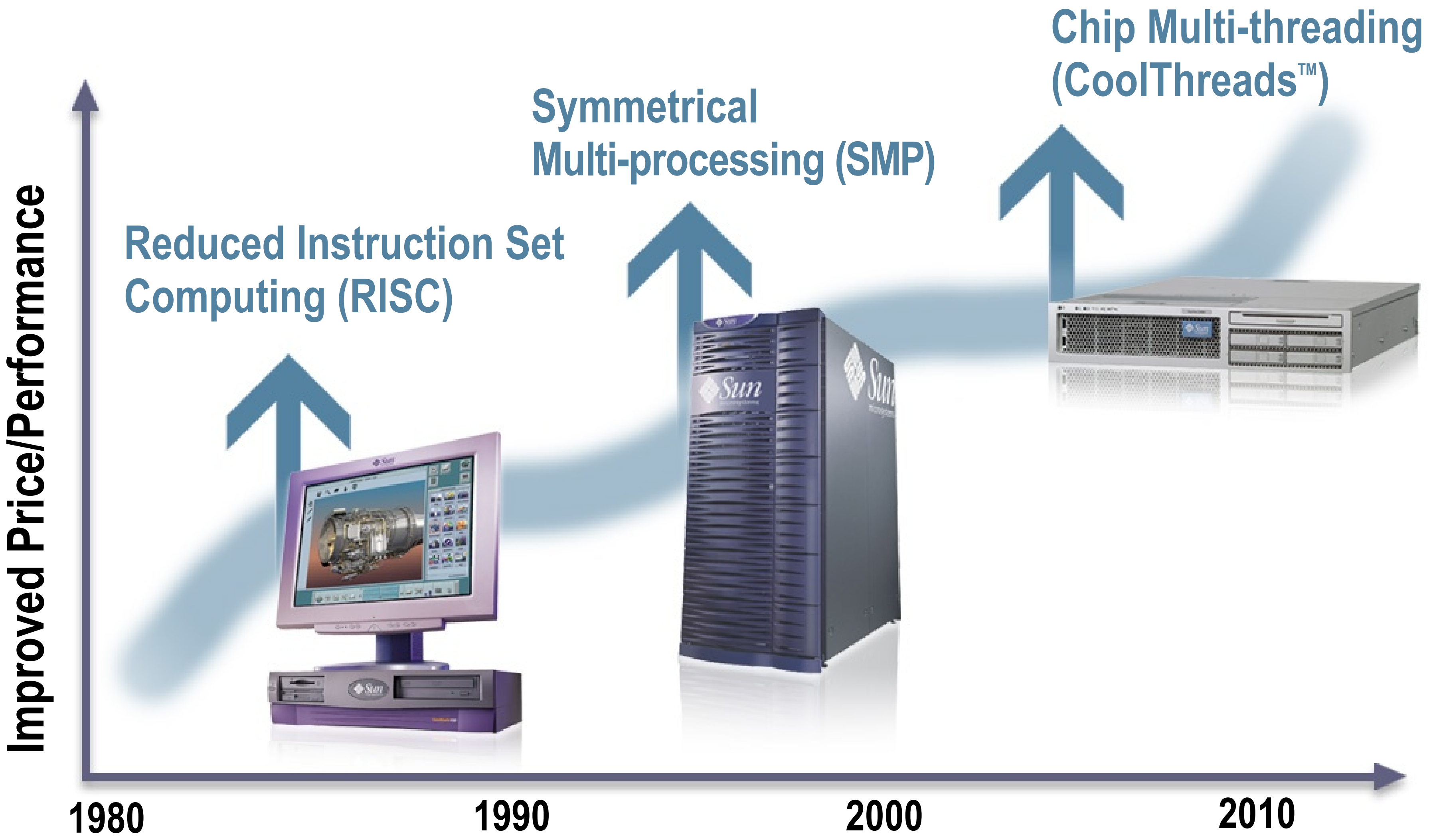
Sun Microsystems, Inc.

www.opensparc.net

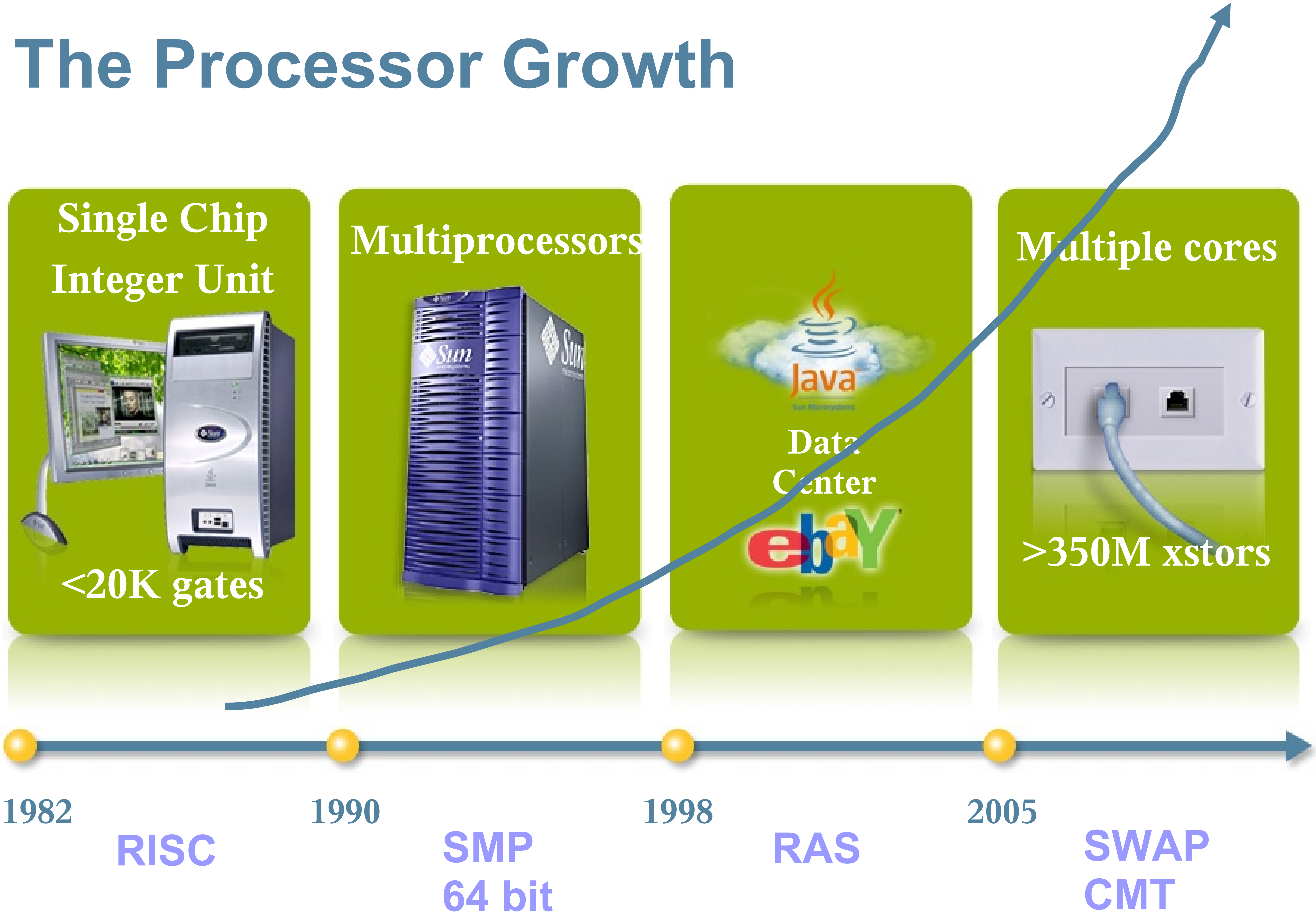
Agenda

1. Chip Multi-Threading (CMT) Era
2. Microarchitecture of OpenSPARC T1
3. OpenSPARC T1 Program
4. SPARC Architecture
5. OpenSPARC in Academia
6. OpenSPARC T1 simulators
7. Hypervisor and OS porting
8. Compiler Optimizations and tools
9. Community Participation

Making the Right Waves


Chip Multi-threading (CoolThreads™)
Symmetrical Multi-processing (SMP)
Reduced Instruction Set Computing (RISC)
Improved Price/Performance
1980
1990
2000
2010

The Processor Growth



Single Chip Integer Unit

<20K gates

Multiprocessors

Data Center

Multiple cores

>350M xstors

1982

RISC

1990

SMP
64 bit

1998

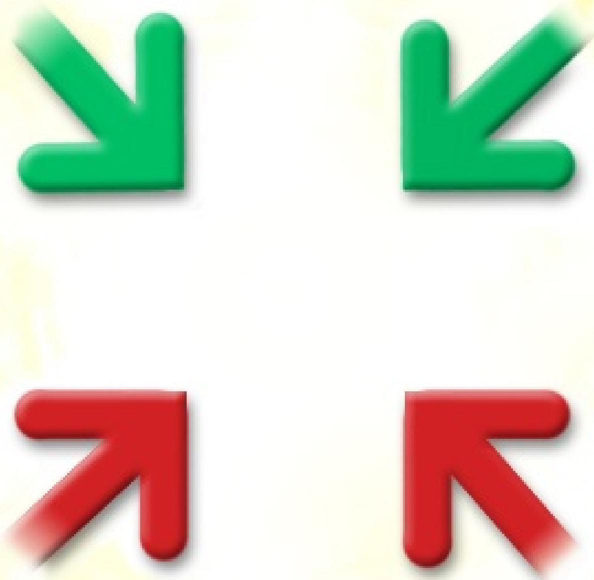
RAS

2005

SWAP
CMT

Source: Sun Network San Francisco, NC03Q3, Sep. 17, 2003

The Big Bang Is Happening— Four Converging Trends



Network Computing Is Thread Rich

Web services, Java™ applications, database transactions, ERP . . .

Moore's Law

A fraction of the die can already build a good processor core; how am I going to use a billion transistors?

Worsening Memory Latency

It's approaching 1000s of CPU cycles! Friend or foe?

Growing Complexity of Processor Design

Forcing a rethinking of processor architecture – modularity, less is more, time-to-market

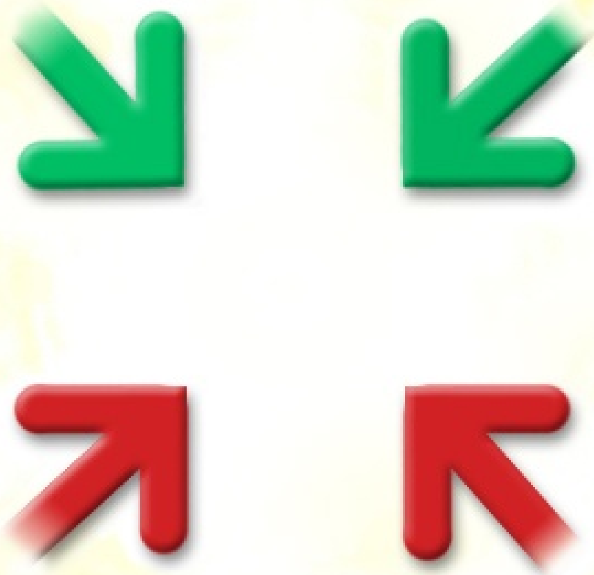
The Big Bang *Has* Happened—Four Converging Trends

Network Computing Is Thread Rich

Web services, Java™ applications, database transactions, ERP . . .

Moore's Law

A fraction of the die can already build a good processor core; how am I going to use a billion transistors?



Worsening Memory Latency

It's approaching 1000s of CPU cycles! Friend or foe?

Growing Complexity of Processor Design

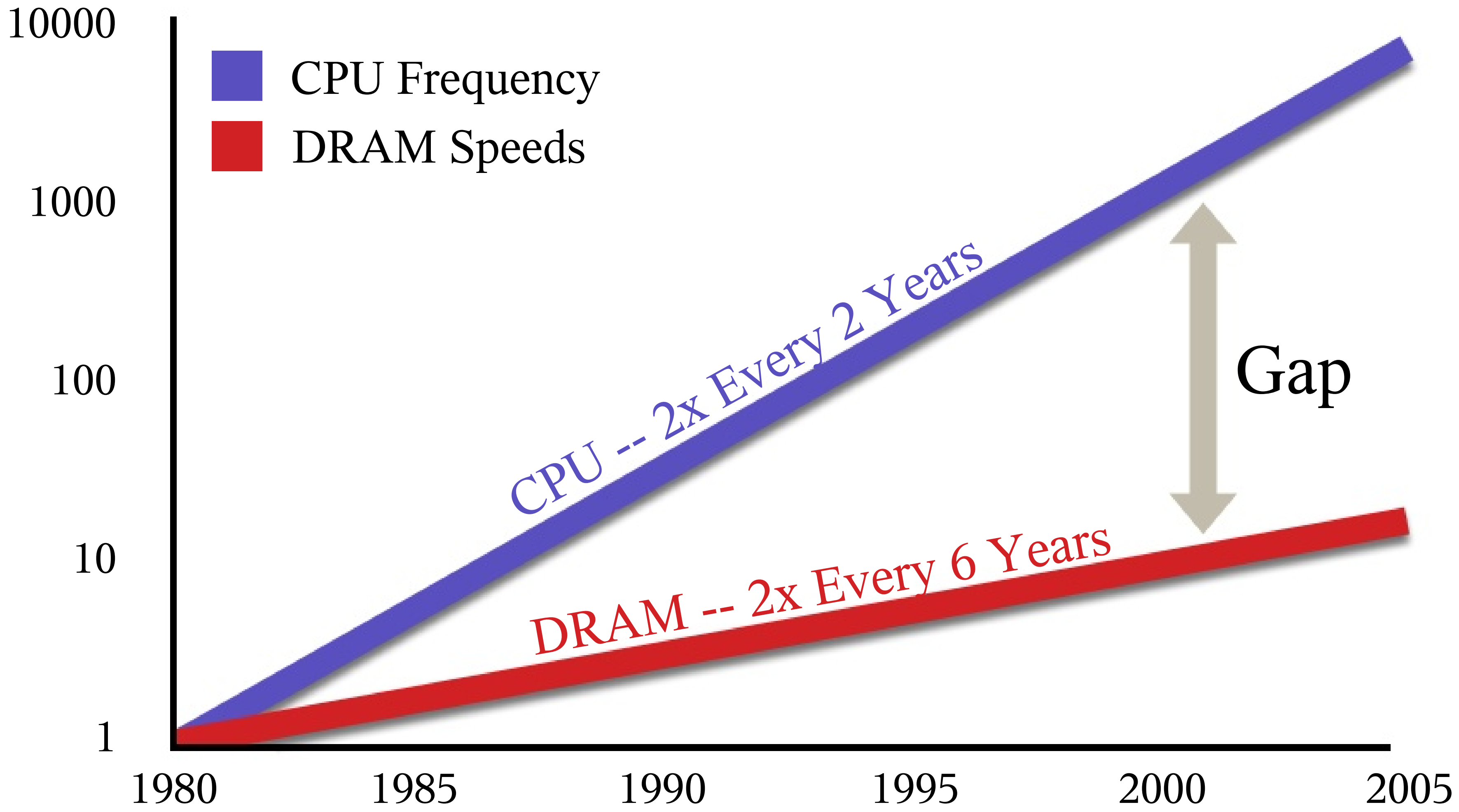
Forcing a rethinking of processor architecture – modularity, less is more, time-to-market

Attributes of Commercial Workloads

	Web Services			Client Server		Data Warehouse
Attribute	TIER1 Web (Web99)	TIER2 App Serv (JBB)	TIER3 Data (TPC-C)	SAP 2T	SAP 3T (DB)	DSS (TPC-H)
Application Category	Web Server	Server Java	OLTP	ERP	ERP	DSS
Instruction-level Parallelism	Low	Low	Low	Medium	Low	High
Thread-level Parallelism	High	High	High	High	High	High
Instruction/Data Working Set	Large	Large	Large	Medium	Large	Large
Data Sharing	Low	Medium	High	Medium	High	Medium

Memory Bottleneck

Relative Performance



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

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

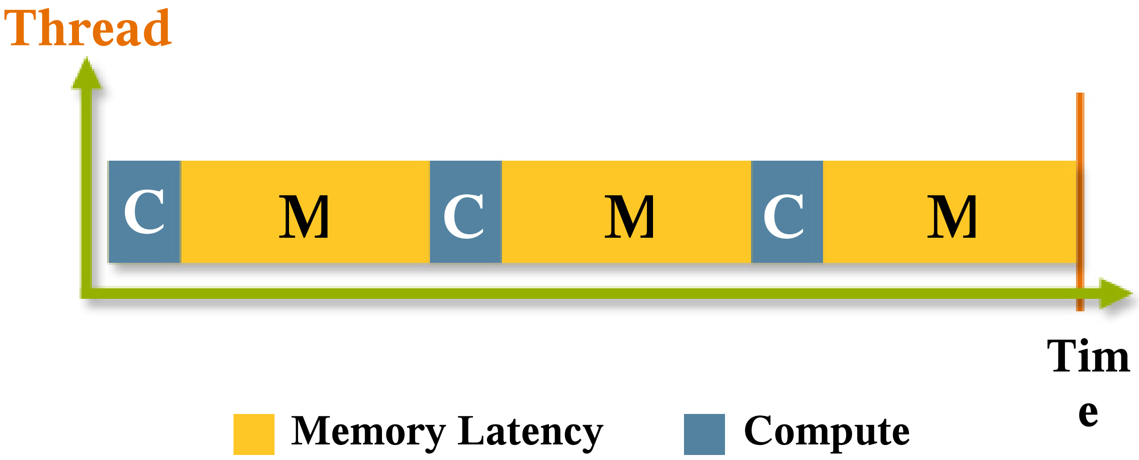
Single Threading

Up to 85% Cycles Waiting for Memory

HURRY UP AND WAIT!

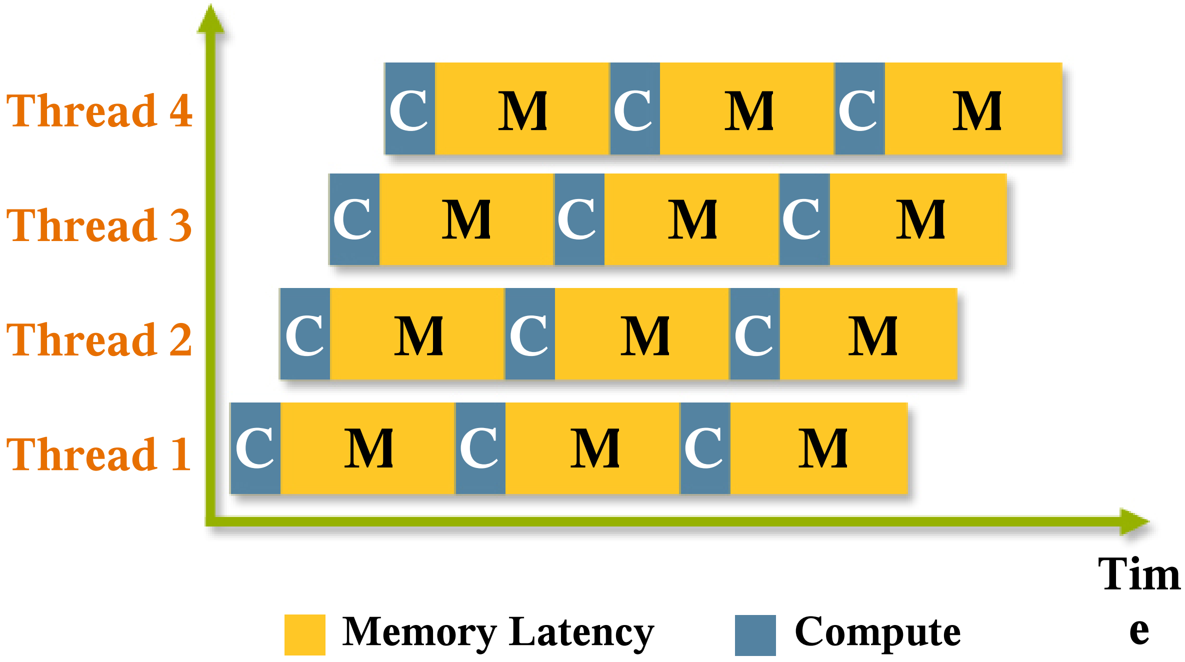
Single Threaded Performance

Typical Processor Utilization:15–25%



The Power of CMT - CoolThreads

UltraSPARC T1 Core Utilization: Up to 85%



Thread 4: C M C M C M

Thread 3: C M C M C M

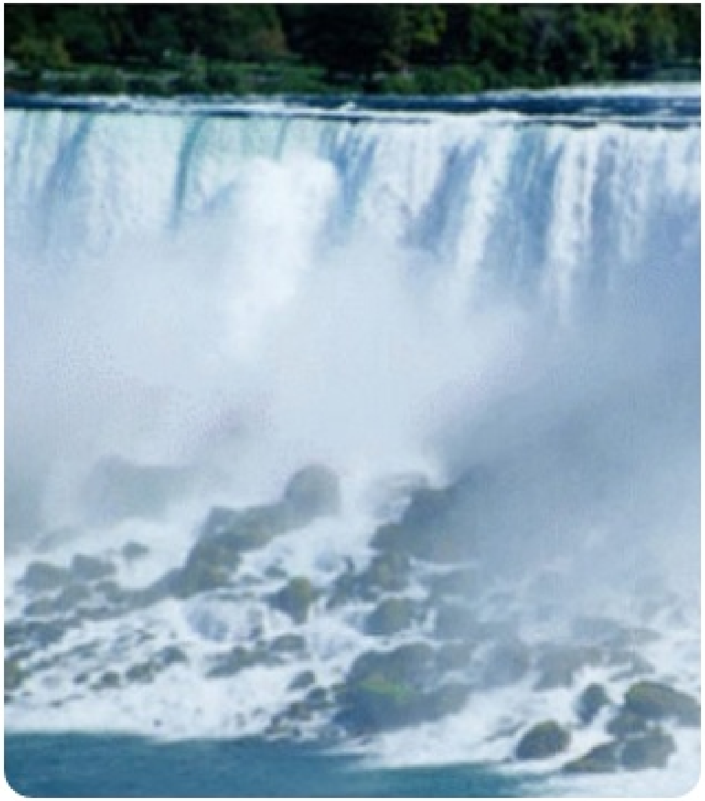
Thread 2: C M C M C M

Thread 1: C M C M C M

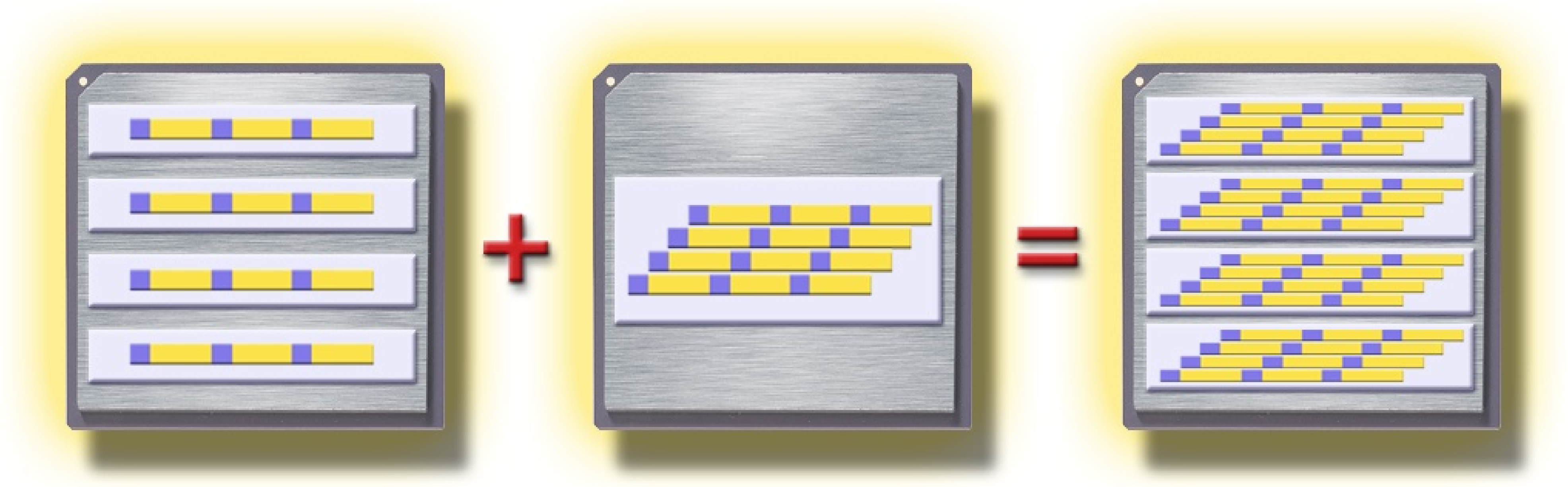
Tim e

🟨 Memory Latency 🟦 Compute

Chip Multi-threaded (CMT) Performance



Chip Multi-Threading (CMT) to the rescue



CMP
(chip multiprocessing)

HMT
(hardware multithreading)

CMT
(chip multithreading)

n cores per processor

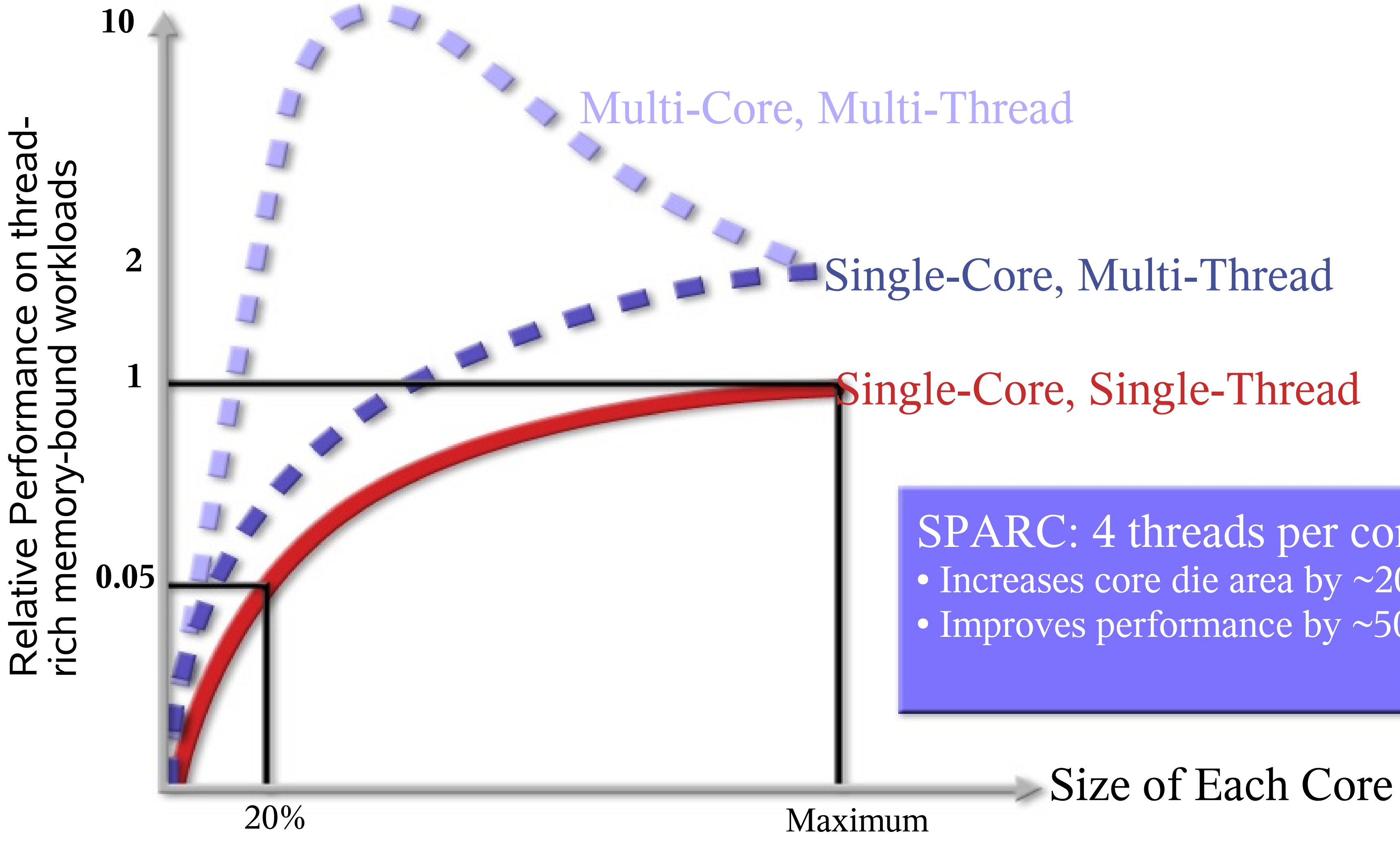
m strands per core

n x m threads per processor

www.opensparc.net

Why CMT Works

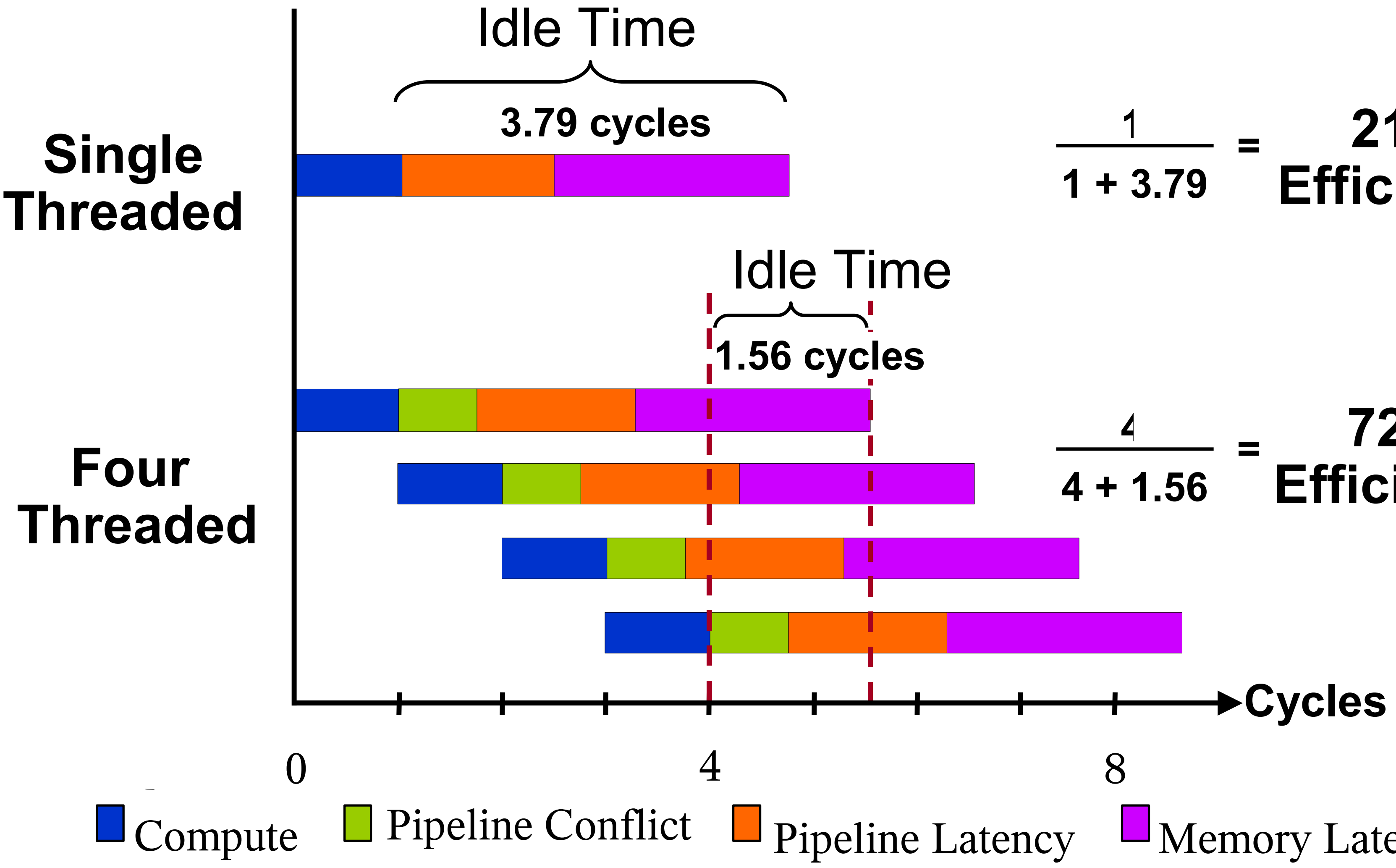
Goal: “100% Resource Utilization” *(given a fixed die size)*



SPARC: 4 threads per core

- Increases core die area by ~20%
- Improves performance by ~50–100%

Example - SpecJBB Execution Efficiency



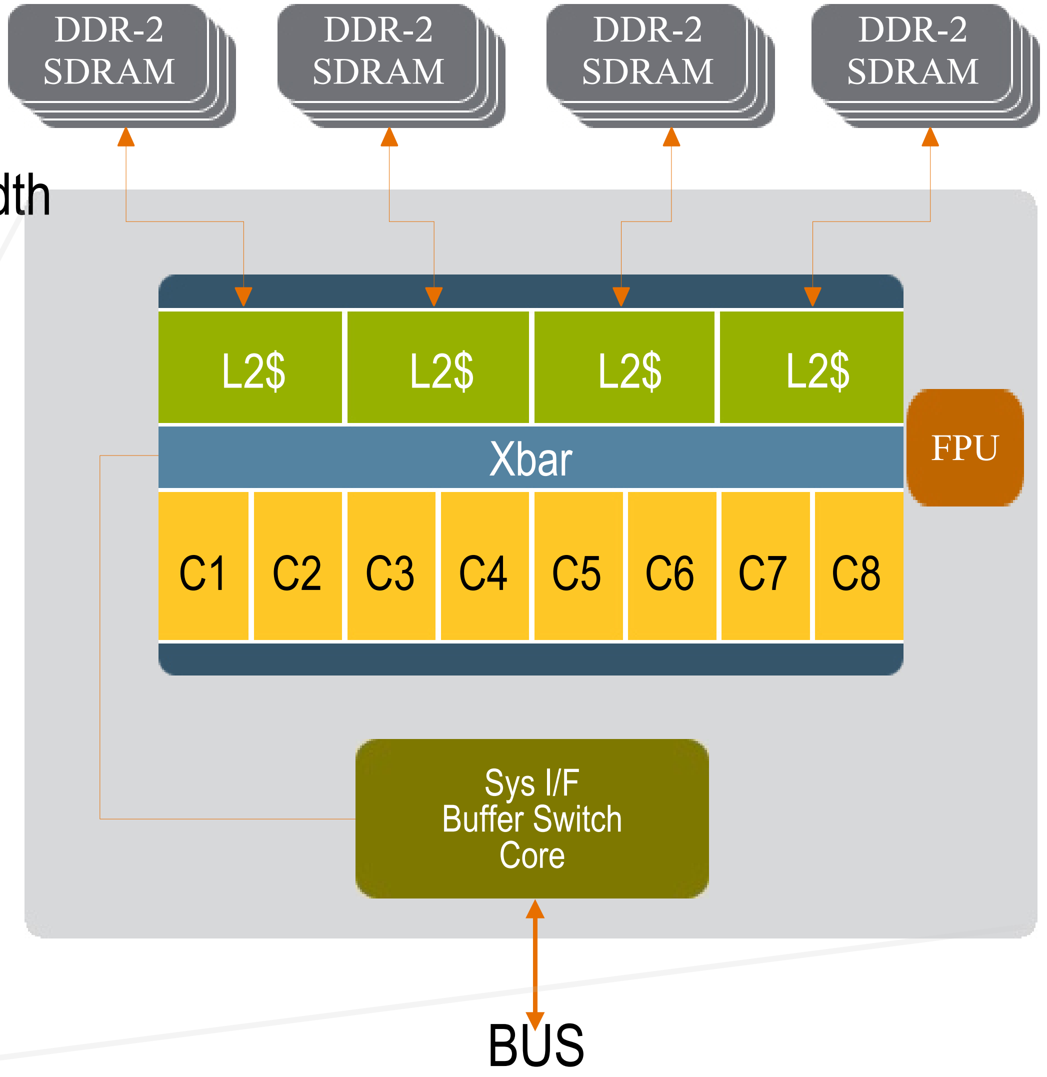
$$\frac{1}{1 + 3.79} =$$ **21% Efficiency**

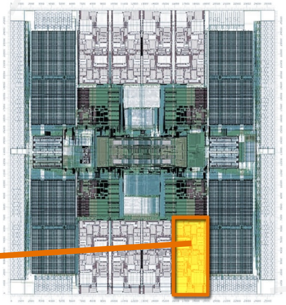
$$\frac{4}{4 + 1.56} =$$ **72% Efficiency**

A. S. Leon *et al*., "A Power-Efficient High Throughput 32-Thread SPARC Processor," ISSCC06, Paper 5.1

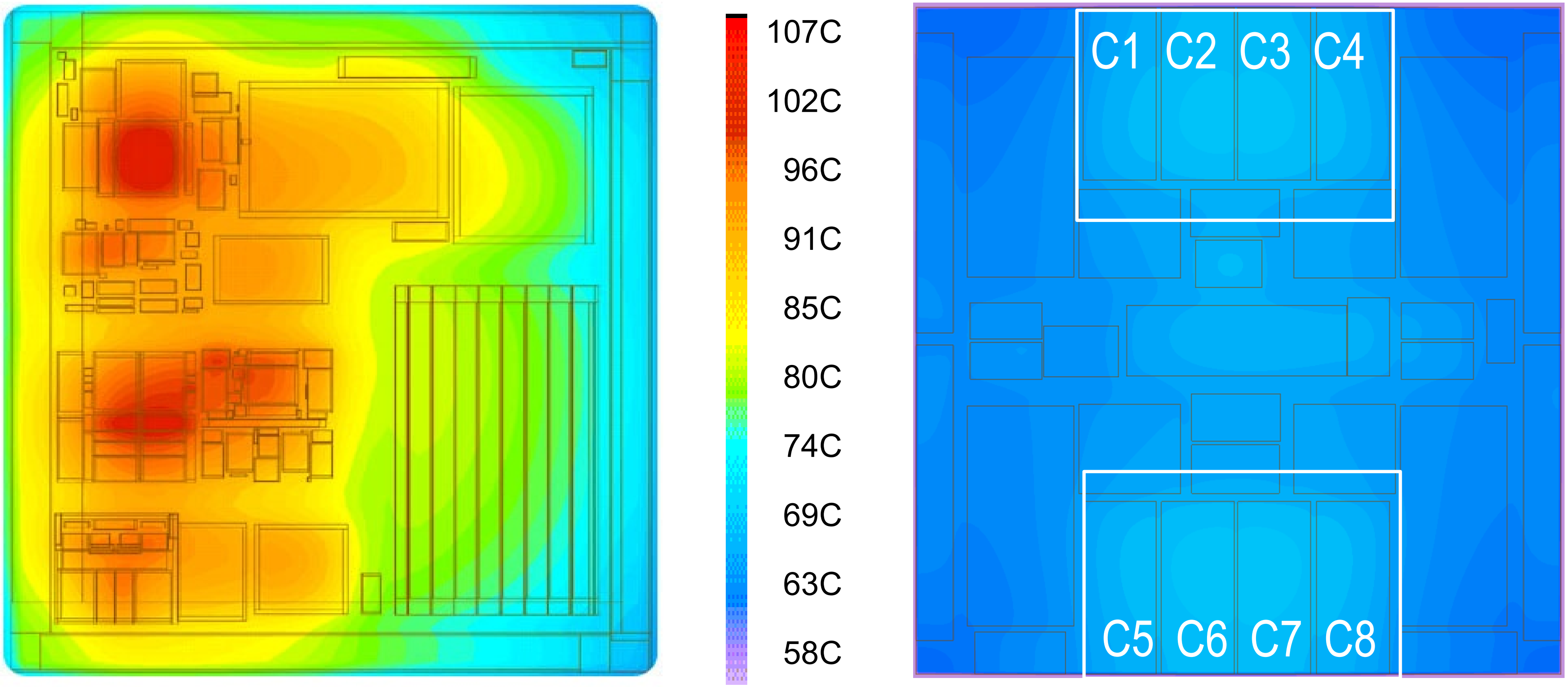
UltraSPARC T1 Processor

- SPARC V9 (Level 1) implementation
- Up to eight 4-threaded cores (32 simultaneous threads)
- All cores connected through high bandwidth (134.4GB/s) crossbar switch
- High-bandwidth, 12-way associative 3MB Level-2 cache on chip
- 4 DDR2 channels (23GB/s)
- Power : < 80W
- ~300M transistors
- 378 sq. mm die





Faster Can Be Cooler



Single-Core Processor

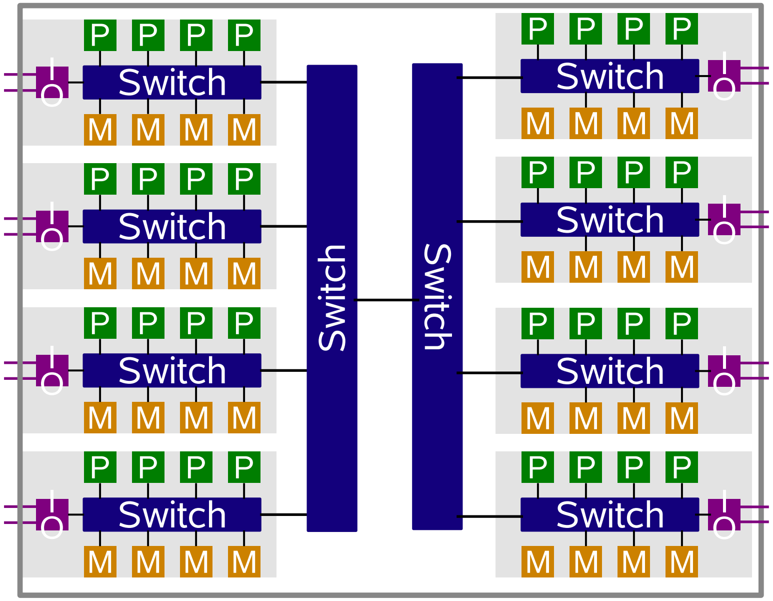
CMT Processor

(Not to Scale)

CMT: On-chip = High Bandwidth

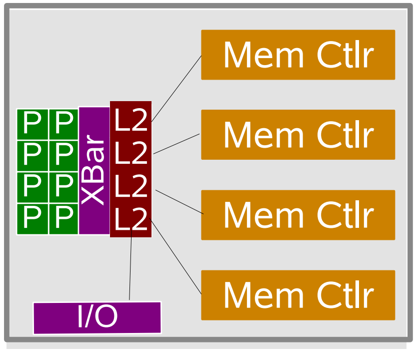
32-thread Traditional SMP System

Example: Typical SMP Machine Configuration



32-thread OpenSPARC T1 Processor

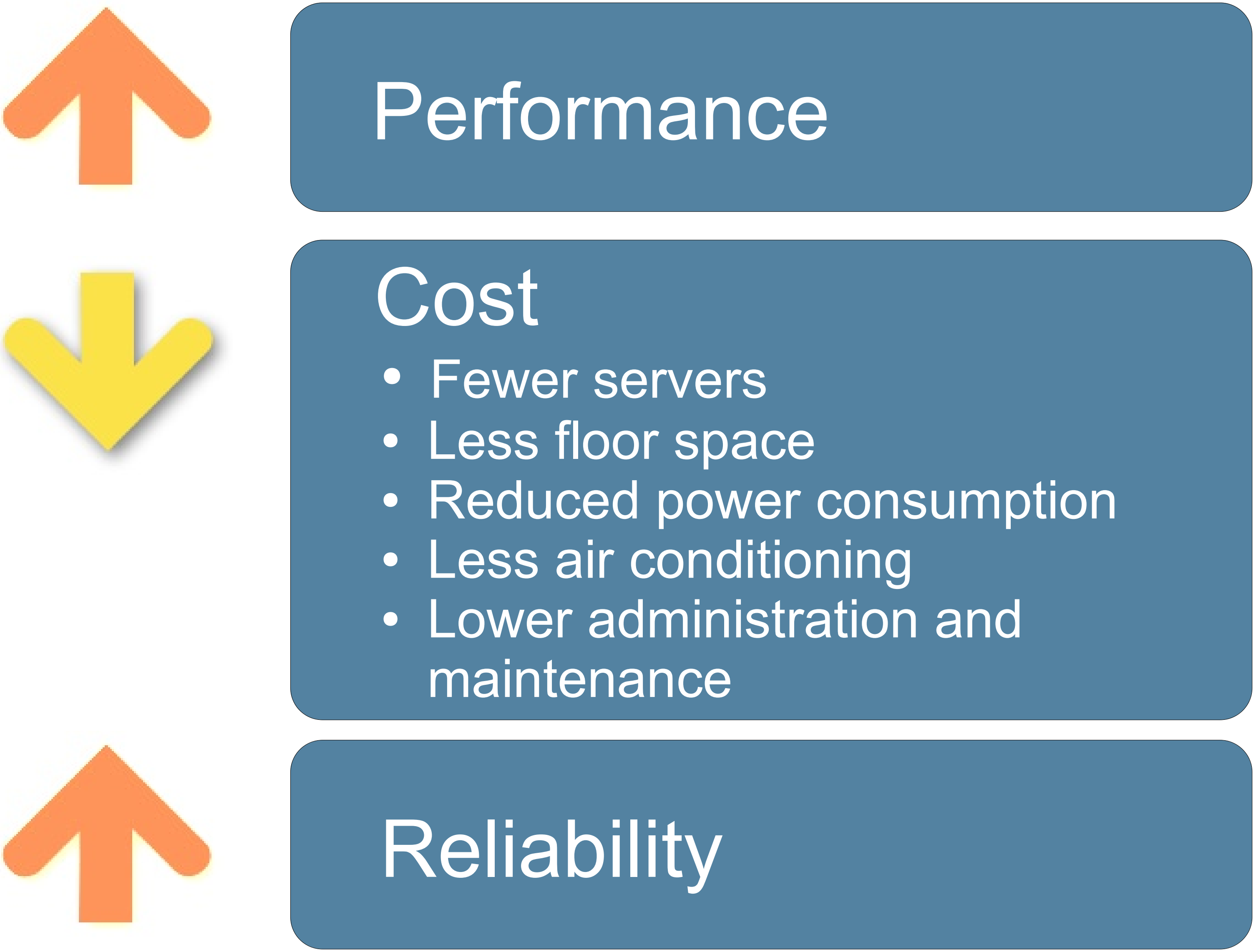
One motherboard, *no* switch ASICs



Direct crossbar interconnect

-- Lower cost
-- better RAS
-- lower BTUs,
-- lower and uniform latency,
-- greater and uniform bandwidth. . .

CMT Benefits



Performance

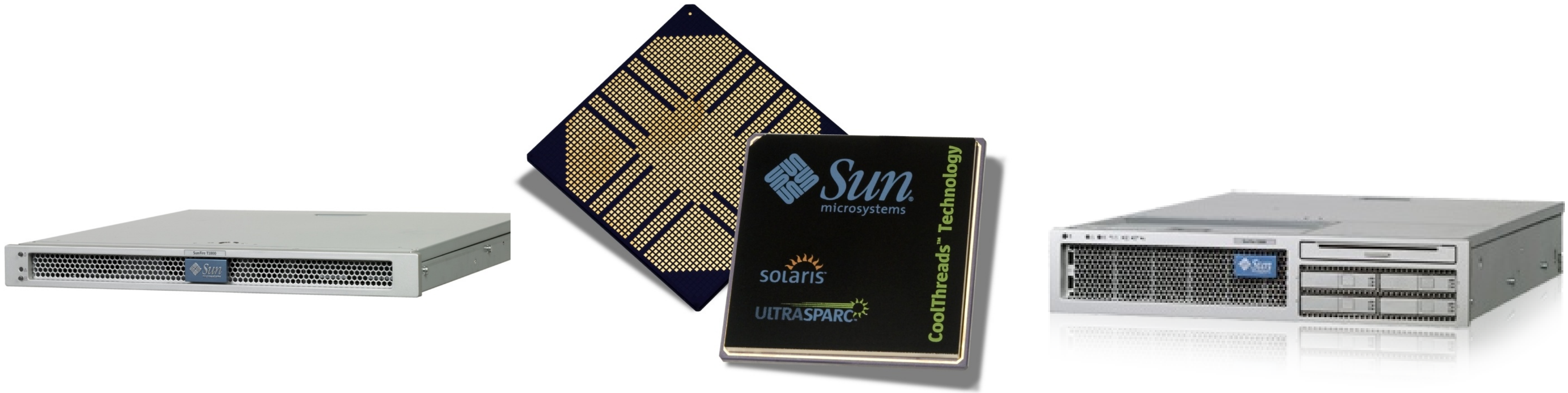
Cost

- Fewer servers
- Less floor space
- Reduced power consumption
- Less air conditioning
- Lower administration and maintenance

Reliability

CMT Pays Off with CoolThreads™ Technology

Sun Fire T1000 and T2000



- Up to 5x the performance
- As low as 1/5 the energy
- As small as 1/4 the size

*See disclosures

CoolThreads Servers are a Hit

NETWORKWORLD

"...performance in several profiles unmatched for the power and space it consumes."

InformationWeek

"These servers could save companies millions."

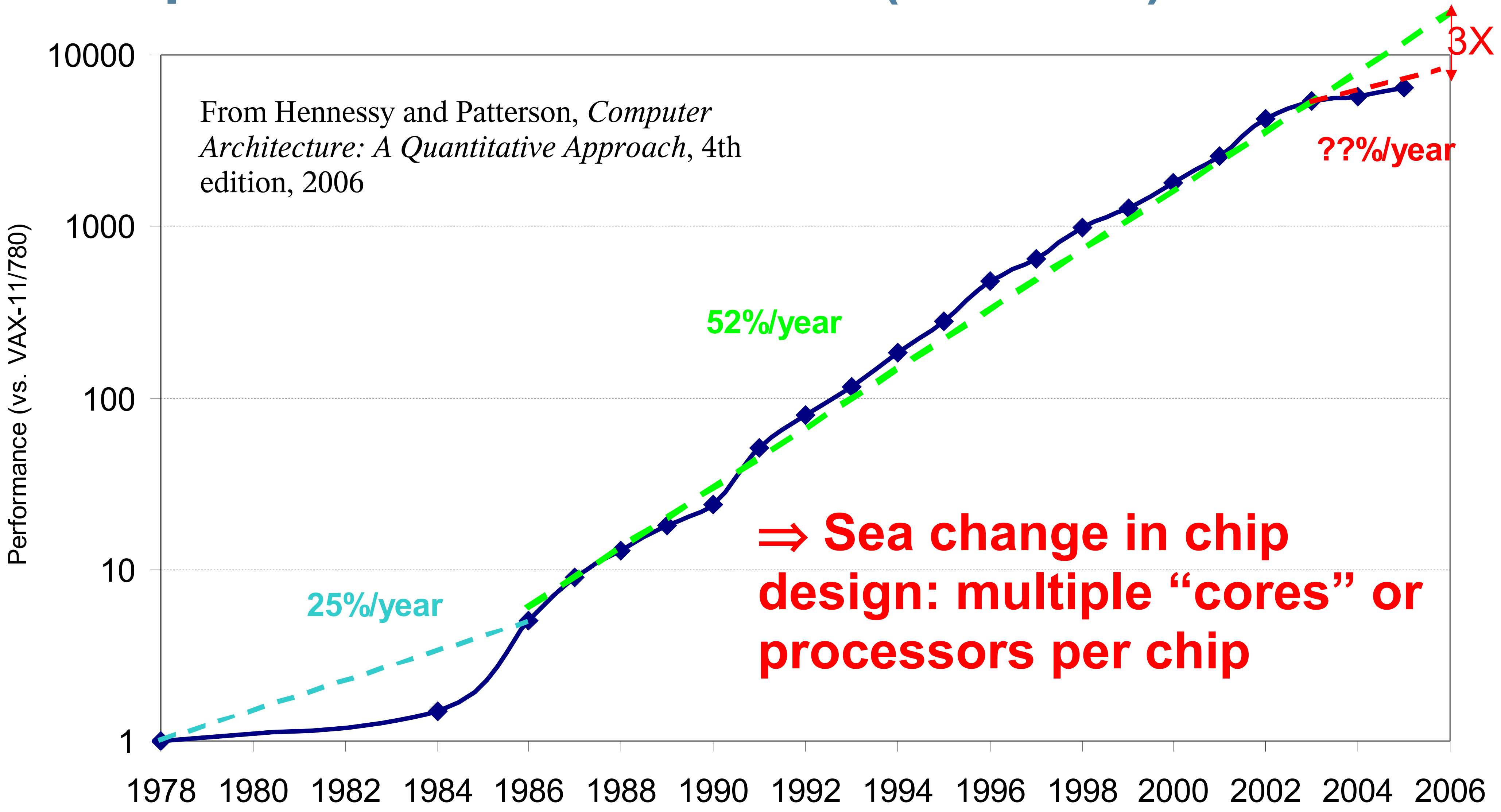
The McGraw-Hill Companies BusinessWeek

"...the machines put Sun at the cutting edge of one of the chip industry's biggest trends... multi-core systems."

InfoWorld

"...the UltraSparc T1 is truly a revolutionary processor."

Uniprocessor Performance (SPECint)



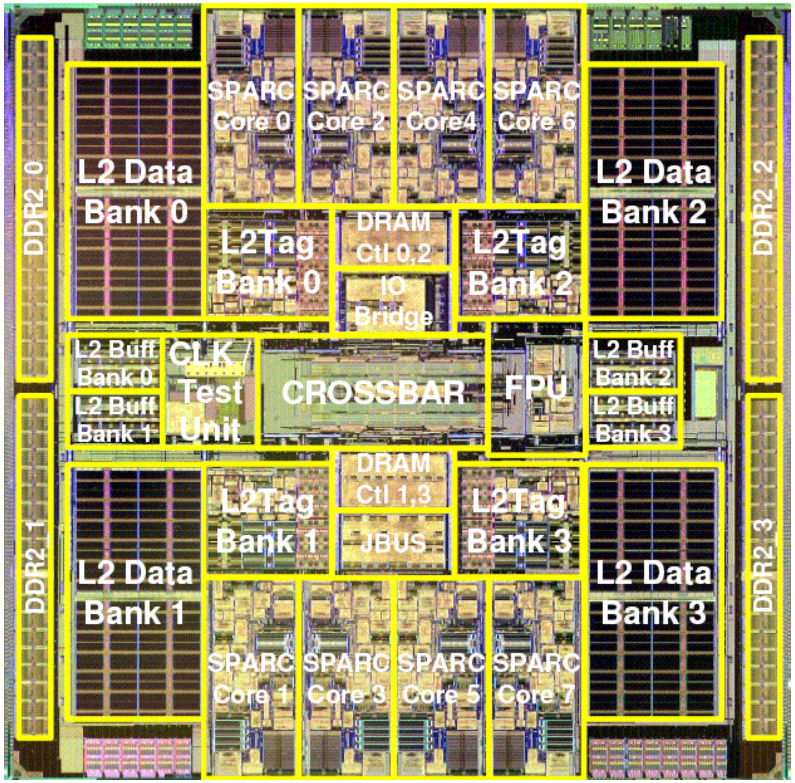
From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, 2006

⇒ **Sea change in chip design: multiple "cores" or processors per chip**

- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: ??%/year 2002 to present**

Source: David Patterson presentation at MultiCore Expo 2006

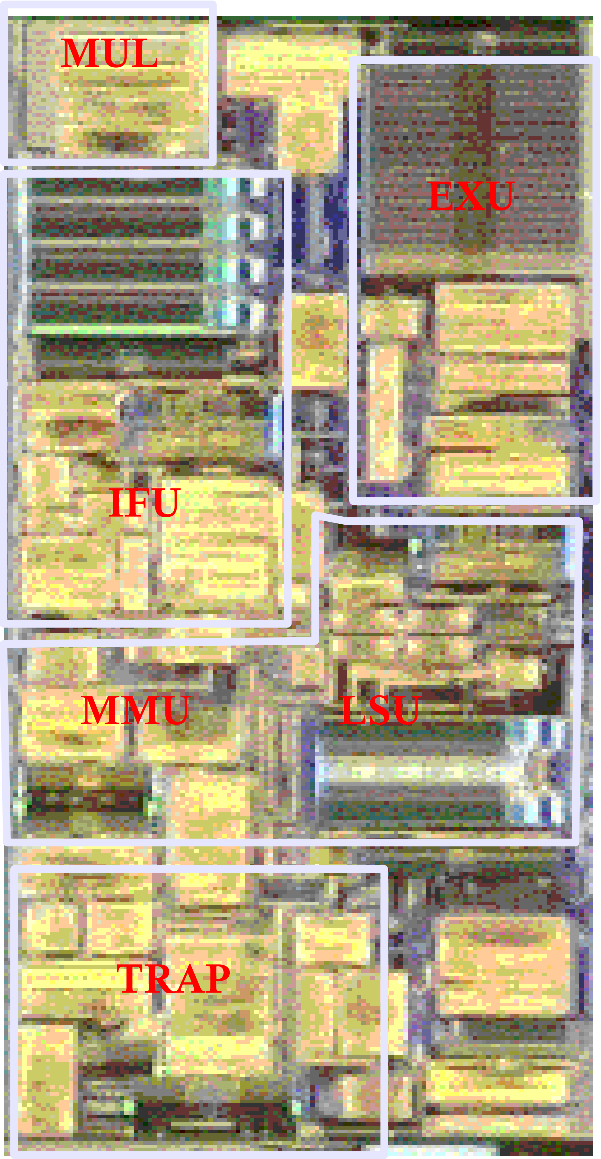
UltraSPARC-T1: Choices & Benefits



- Simple core (6-stage, only 11mm^2 in 90nm), 1 FPU
  ⟶ maximum # of cores/threads on die
  ⟶ pipeline built from scratch, useful for multiple generations
  ⟶ modular, flexible design ... ***scalable*** (up and down)
- Caches, DRAM channels shared across cores
  → better area utilization
- Shared L2 cache
  ⟶ cost of coherence misses decrease by order of magnitude
  ⟶ enables highly efficient multi-threaded software
- On-die memory controllers
  → reduce miss latency
- Crossbar switch
  → good for b/w, latency, functional verification

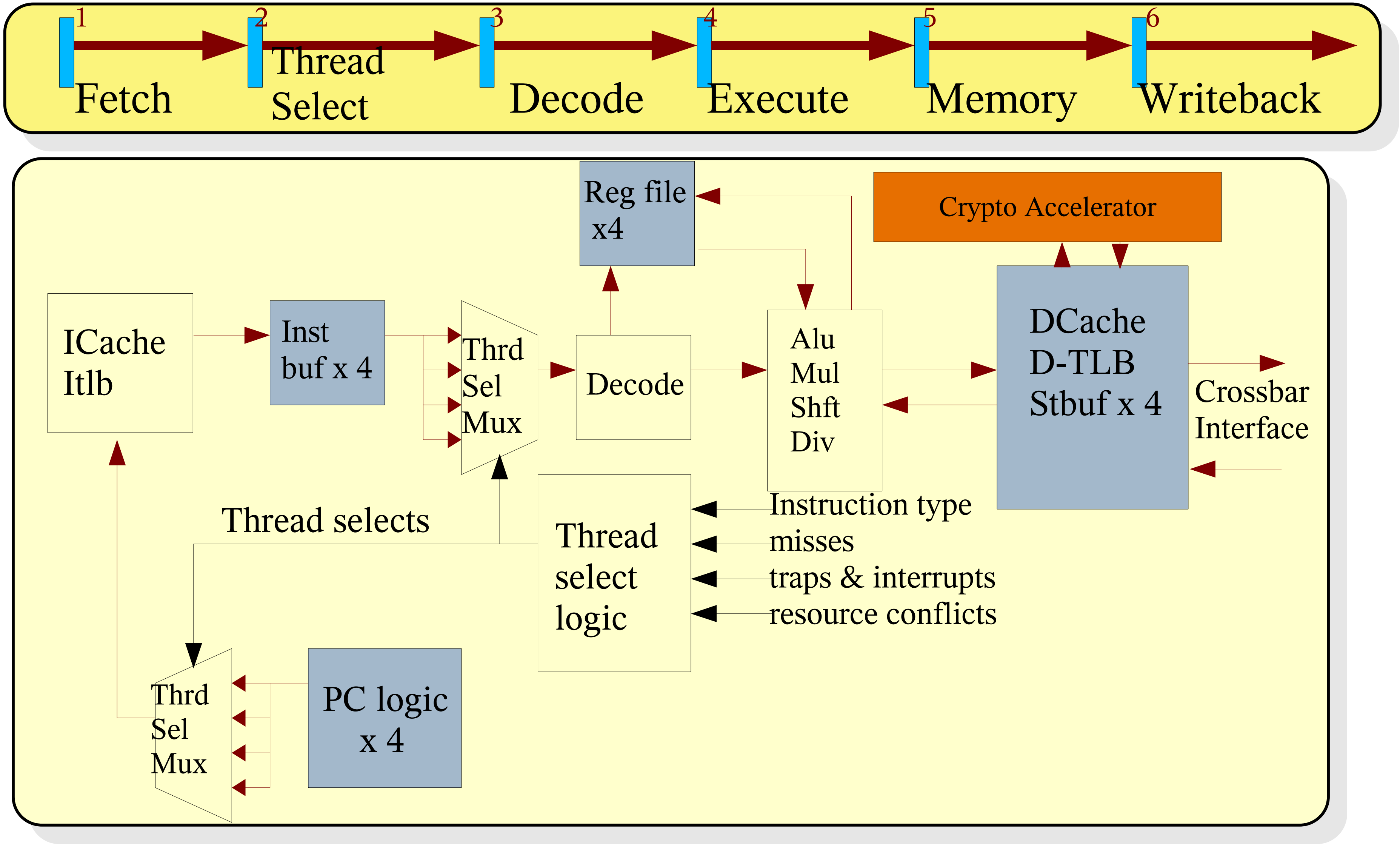
For reference: in 90nm technology, included 8 cores, 32 threads, and only dissipate 70^W

UltraSPARC-T1 Processor Core



- Four threads per core
- Single issue 6 stage pipeline
- 16KB I-Cache, 8KB D-Cache
- Unique resources per thread
  - Registers
  - Portions of I-fetch datapath
  - Store and Miss buffers
- Resources shared by 4 threads
  - Caches, TLBs, Execution Units
  - Pipeline registers and DP
- Core Area = 11mm2 in 90nm
- MT adds ~20% area to core

UltraSPARC T1 Processor Core Pipeline



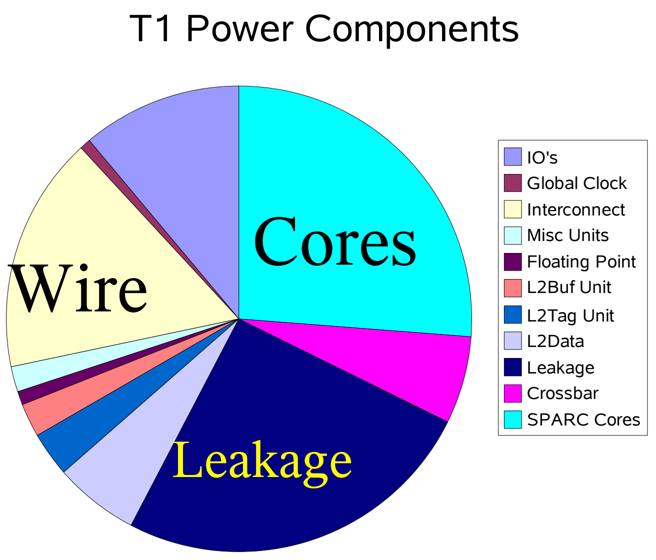
...blue units are replicated *per thread* on core

Thread Selection Policy

- Every cycle, switch among available (ready to run) threads
  - priority given to least-recently-executed thread
- Thread becomes not-ready-to-run due to:
  - Long latency operation like load, branch, mul, or div
  - Pipeline stall such as cache miss, trap, or resource conflict
- Loads are speculated as cache hits, and the thread is switched in with lower priority

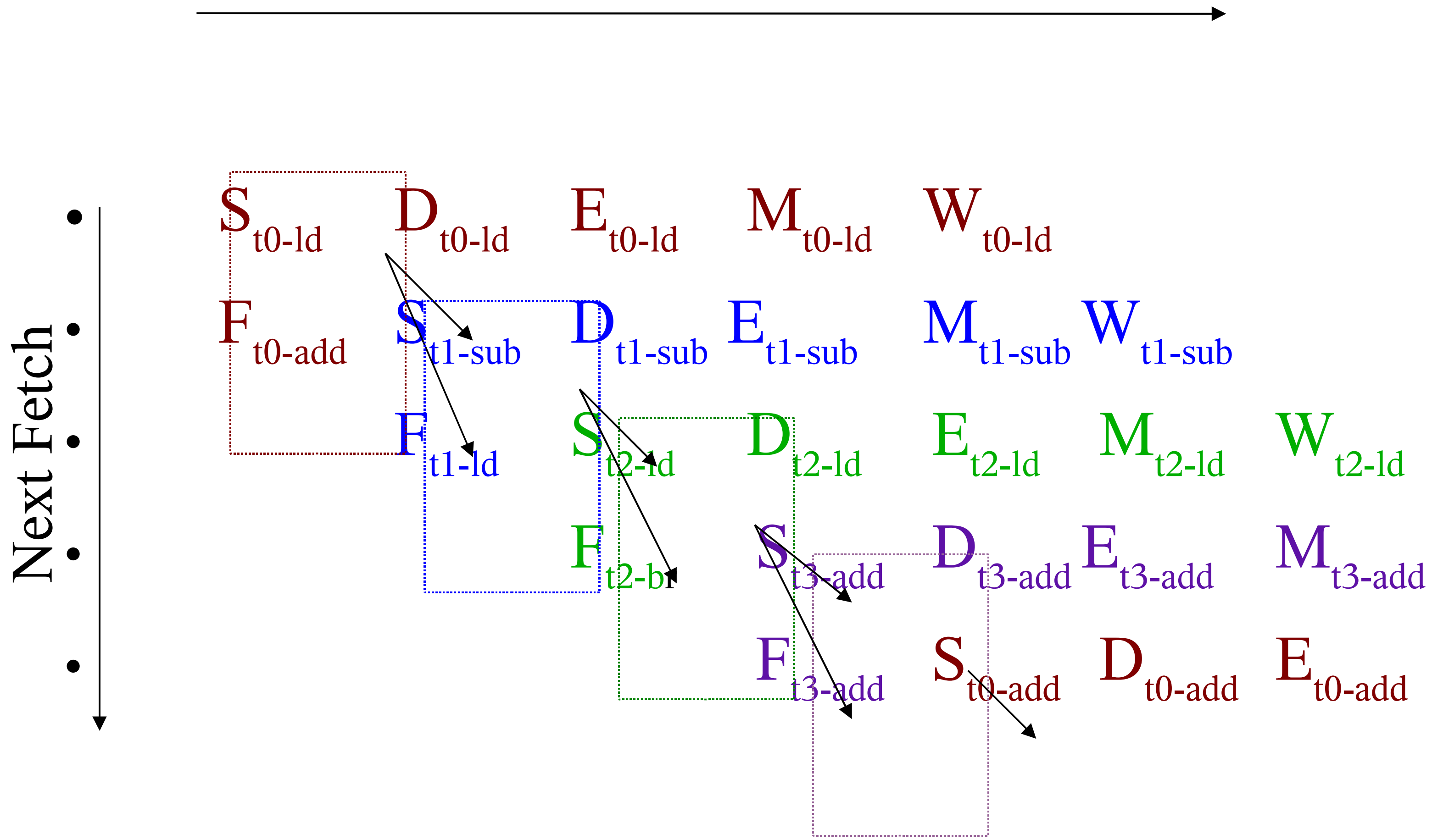
T1 Power

- Power Efficient Architecture
  - Single issue, in-order six stage pipeline
  - Minimal speculation, predication or branch prediction
- Thermal monitoring for power throttling
  - 3 external power throttle pins
    - Controlled by thermal diodes
    - Stall cycles injected, affecting all threads
  - Memory throttling
    - Open page limit
- Design Implementation
  - Fully static design
  - Fine granularity clock gating
    - Limited clock issue on stall, FGU
    - Limited L2 Cache & Memory clock gating
  - Wire classes optimized for power * delay



Microarchitecture details of the UltraSPARC -T1 CPU

Thread Selection – All Threads Ready

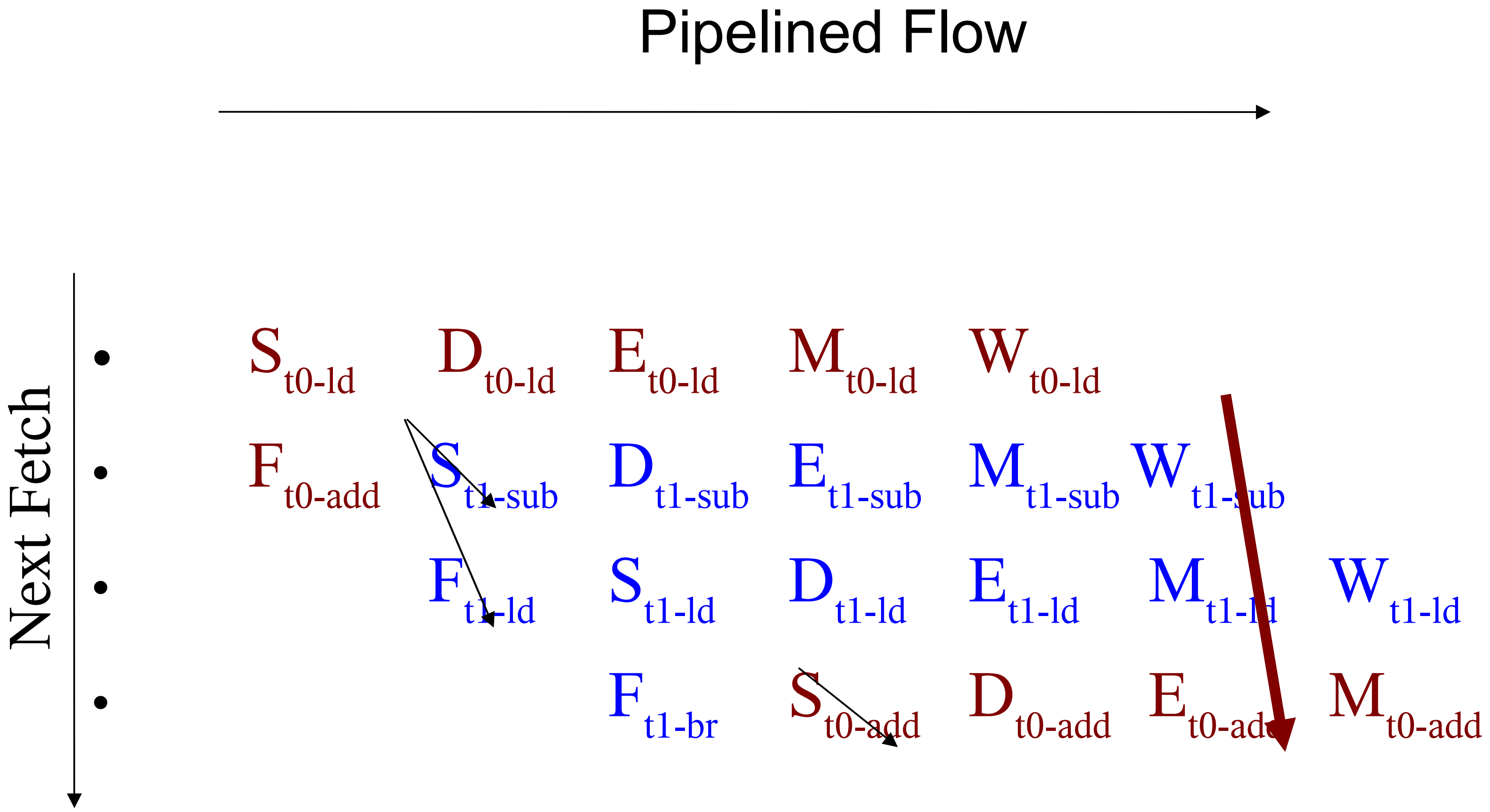


Pipelined Flow

Next Fetch

$S_{t0\text{-}ld}$	$D_{t0\text{-}ld}$	$E_{t0\text{-}ld}$	$M_{t0\text{-}ld}$	$W_{t0\text{-}ld}$		
$F_{t0\text{-}add}$	$S_{t1\text{-}sub}$	$D_{t1\text{-}sub}$	$E_{t1\text{-}sub}$	$M_{t1\text{-}sub}$	$W_{t1\text{-}sub}$	
	$F_{t1\text{-}ld}$	$S_{t2\text{-}ld}$	$D_{t2\text{-}ld}$	$E_{t2\text{-}ld}$	$M_{t2\text{-}ld}$	$W_{t2\text{-}ld}$
		$F_{t2\text{-}br}$	$S_{t3\text{-}add}$	$D_{t3\text{-}add}$	$E_{t3\text{-}add}$	$M_{t3\text{-}add}$
			$F_{t3\text{-}add}$	$S_{t0\text{-}add}$	$D_{t0\text{-}add}$	$E_{t0\text{-}add}$

Thread Selection – Two Threads Ready



•	$S_{t0\text{-}ld}$	$D_{t0\text{-}ld}$	$E_{t0\text{-}ld}$	$M_{t0\text{-}ld}$	$W_{t0\text{-}ld}$		
•	$F_{t0\text{-}add}$	$S_{t1\text{-}sub}$	$D_{t1\text{-}sub}$	$E_{t1\text{-}sub}$	$M_{t1\text{-}sub}$	$W_{t1\text{-}sub}$	
•		$F_{t1\text{-}ld}$	$S_{t1\text{-}ld}$	$D_{t1\text{-}ld}$	$E_{t1\text{-}ld}$	$M_{t1\text{-}ld}$	$W_{t1\text{-}ld}$
•			$F_{t1\text{-}br}$	$S_{t0\text{-}add}$	$D_{t0\text{-}add}$	$E_{t0\text{-}add}$	$M_{t0\text{-}add}$

Thread '0' is speculatively switched in before cache hit information is available, in time for the 'load' to bypass data to the 'add'

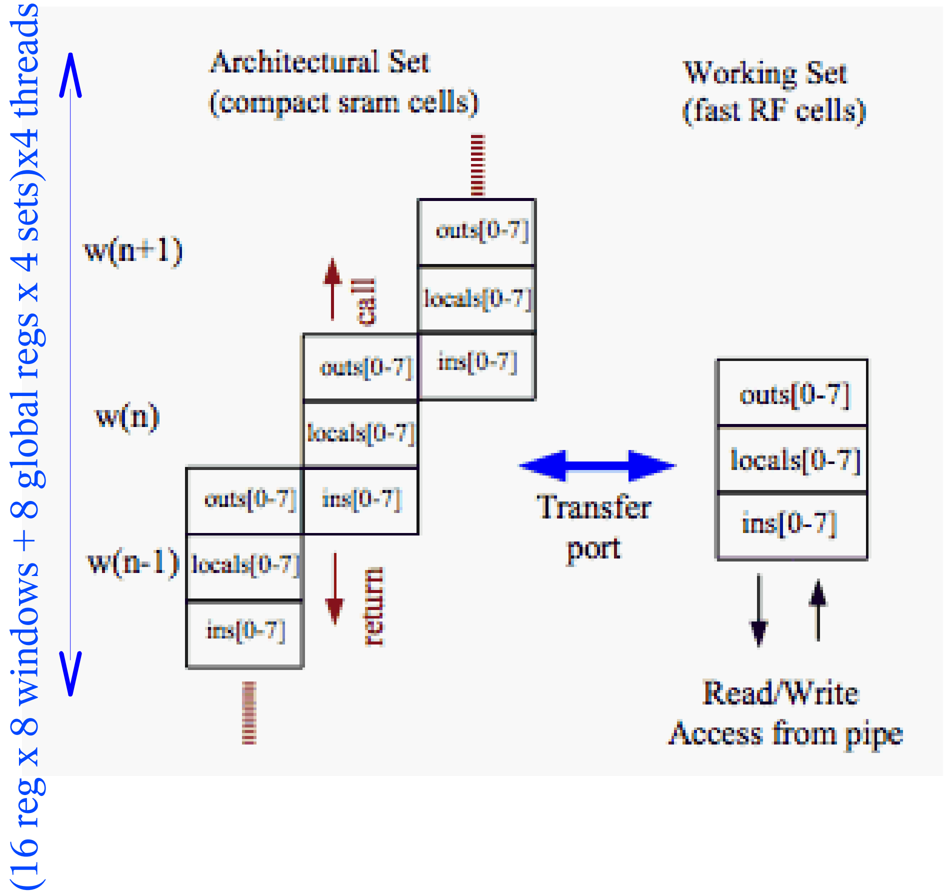
Instruction Fetch/Switch/Decode Unit(IFU)

- I-cache complex
  - 16KB data, 4ways, 32B line size
  - Single ported Instruction Tag.
  - Dual ported(1R/1W) Valid bit array to hold Cache line state of valid/invalid
  - Invalidates access Vbit array not Instruction Tag
  - Pseudo-random replacement
- Fully Associative Instruction TLB
  - 64 entries, Page sizes: 8k,64k, 4M, 256M
  - Pseudo LRU replacement.
  - Multiple hits in TLB prevented by doing autodemap on fill

IFU Functions (cont'd)

- 2 instructions fetched each cycle, though only one is issued/clk. Reduces I$ activity and allows opportunistic line fill.
- 1 outstanding miss/thread, and 4 per core. Duplicate misses do not request to L2
- PC's, NPC's for all live instructions in machine maintained in IFU

Windowed Integer Register File



- 5KB 3R/2W/1T structure
  - 640 64b regs with ECC!
- Only the 32 registers from current window is visible to thread
- Window changing in background under thread switch. Other threads continue to access IRF
- Compact design with 6T cells for architectural set & multi ported cell for working set.
- Single cycle R/W access

Execution Units

- Single ALU and Shifter. ALU reused for Branch Address and Virtual Address Calculation
- Integer Multiplier
  - 5 clock latency, throughput of ½ per cycle for area saving
  - Contains accumulate function for Mod Arithmetic.
  - 1 integer mul allowed outstanding per core.
  - Multiplier shared between Core Pipe and Modular Arithmetic unit on a round robin basis.
- Simple non restoring divider, with one divide outstanding per core.
- Thread issuing a MUL/DIV will rollback and switch out if another thread is occupying the mul/div units.

Load Store Unit(LSU)

- D-Cache complex
  - > 8KB data, 4ways, 16B line size
  - > Single ported Data Tag.
  - > Dual ported(1R/1W) Valid bit array to hold Cache line state of valid/invalid
  - > Invalidates access Vbit array but not Data Tag
  - > Pseudo-random replacement
  - > Loads are allocating, stores are non allocating.
- DTLB: common macro to ITLB(64 entry FA)
- 8 entry store buffer per thread, unified into single 32 entry array, with RAW bypassing.

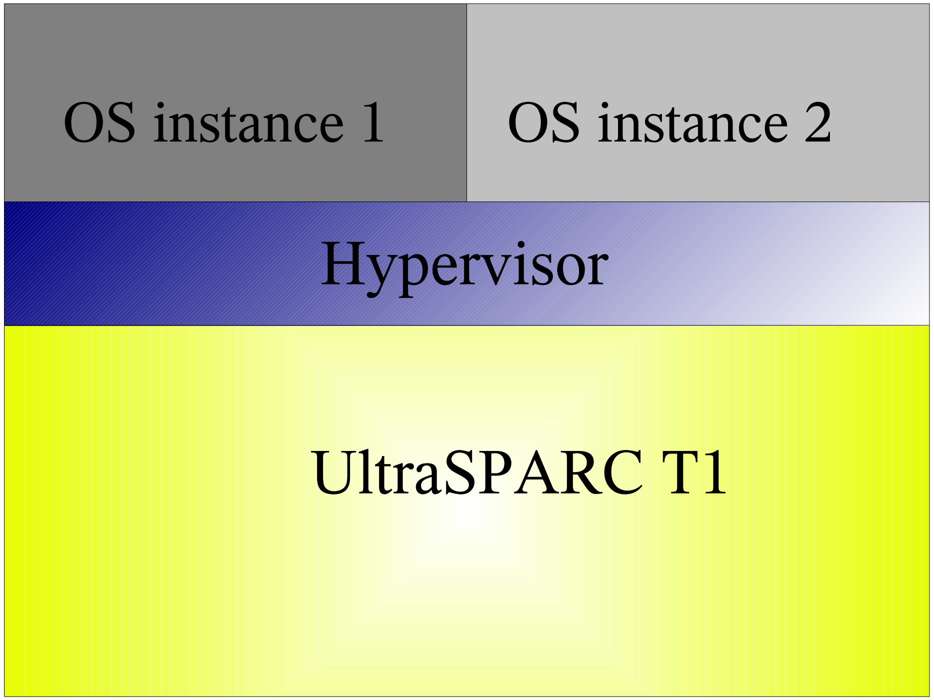
LSU(cont'd)

- Single load per thread outstanding. Duplicate request for the same line not sent to L2
- Crossbar interface
  - > LSU prioritizes requests to the crossbar for FPops, Streaming ops, I and D misses, stores and interrupts etc.
  - > Request priority: imiss>ldmiss>stores,{fpu,strm,interrupt}.
  - > Packet assembly for pcx.
- Handles returns from crossbar and maintains order for cache updates and invalidates.

Other Functions

- Support for 6 trap levels. Traps cause pipeline flush and thread switch until trap PC is available
- Support for upto 64 pending interrupts per thread
- Floating Point
  - FP registers and decode located within core
  - On detecting an Fpop
    - The thread switches out
    - Fpop is further decoded and FRF is read
    - Fpop with operands are packetized and shipped over the crossbar to the FPU
    - Computation done in FPU and result returned via crossbar
    - Writeback completed to FRF and thread restart

Virtualization

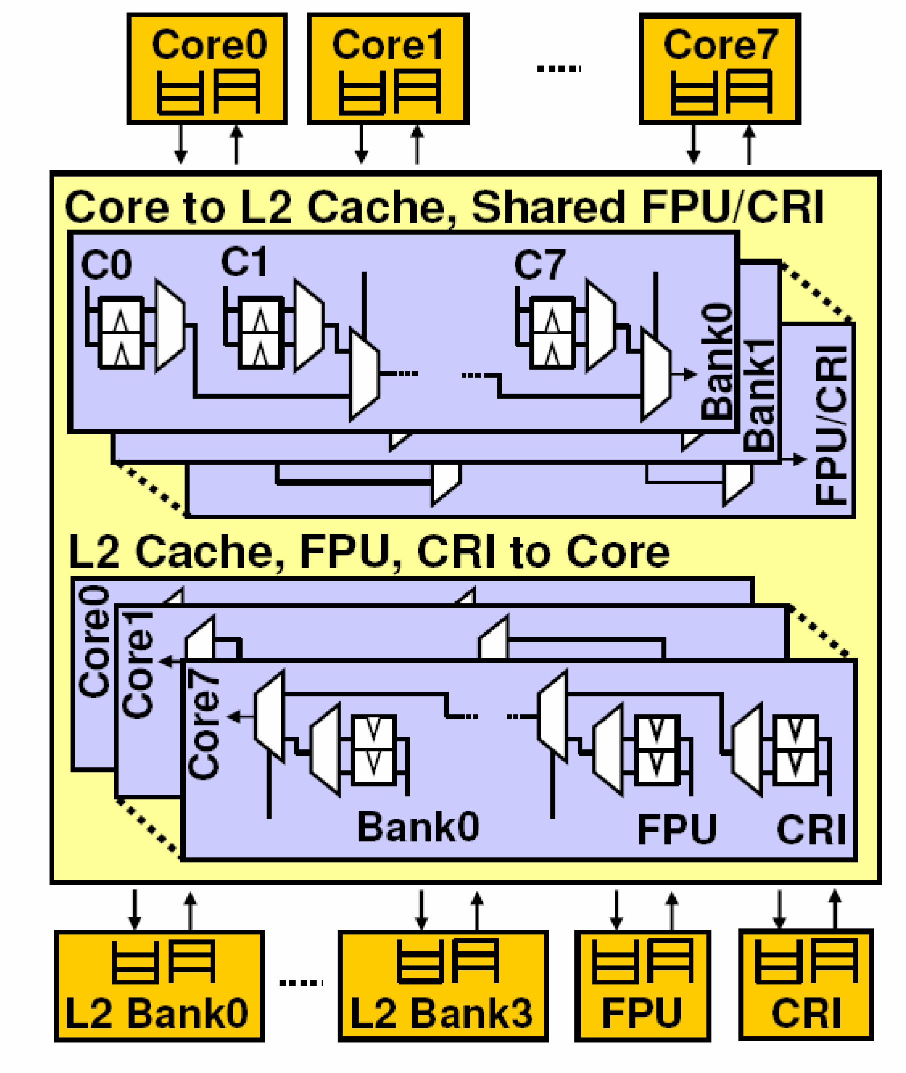


- Hypervisor layer virtualizes CPU
- Multiple OS instances
- Better RAS as failures in one domain do not affect other domain
- Improved OS portability to newer hardware

Virtualization on UltraSPARC T1

- Implementation on UltraSPARC-T1
  - Hypervisor uses Physical Addresses
  - Supervisor sees 'Real Addresses' – a PA abstraction
  - VA translated to 'RA' and then PA. Niagara MMU and TLB provides h/w support.
  - Upto 8 partitions can be supported. 3Bit partion ID is part of TLB translation checks
  - Additional trap level added for hypervisor use

Crossbar



- Each requestor queues upto 2 packets per destination.
- 3 stage pipeline: Request, Arbitrate and Transmit
- Centralised arbitration with oldest requestor getting priority
- Core to cache bus optimized for address + doubleword store
- Cache to core bus optimized for 16B line fill. 32B I$ line fill delivered in 2 back to back clks

L2 Cache

- 3MB, 4-way banked, 12way SA, Writeback
- 64B line size, 64B interleaved between banks
- Pipeline latency: 8 clks for Load, 9 clks for I-miss, with critical chunk returned first
- 16 outstanding misses per bank -> 64 total
- Coherence maintained by shadowing L1 tags in an L2 directory structure.
- L2 is point of global visibility. DMA from IO is serialised wrt traffic from cores in L2

L2 Cache – Directory

- Directory shadows L1 tags
  - L1 set index and L2 bank interleaving is such that ¼ of L1 entries come from an L2 bank
  - On an L1 miss, the L1 replacement way and set index identify the physical location of the tag which will be updated by miss address
  - On a store, directory will be cammed.
    - Directory entries collated by set so only 64 entries need to be cammed. Scheme is quite power efficient
    - Invalidates are a pointer to the physical location in the L1, eliminating the need for a tag lookup in L1

Coherence/Ordering

- Loads update directory & fill the L1 on return
- Stores are non allocating in L1
  - Two flavors of stores: TSO, RMO.
    One TSO store outstanding to L2 per thread to preserve store ordering. No such limitation on RMO stores
  - No tag check done at store buffer insert
  - Stores check directory and determine L1 hit.
  - Directory sends store ack/inv to core
  - Store update happens to D$ on store ack
- Crossbar orders responses across cache banks

On Chip Mem Controller

- 4 independent DDRII DRAM channels
- Can supports memory size of upto 128GB
- 25GB/s peak bandwidth
- Schedules across 8 rds + 8 writes
- Can be programmed to 2 channel mode in reduced configuration
- 128+16b interface, chipkill support, nibble error correction, byte error detection
- Designed to work from 125-200Mhz

New wave requires rethinking everything

Why not open source hardware?

www.opensparc.net

OpenSPARC™

It’s about
Participation

World's First Open Source Microprocessor

OpenSPARC.net

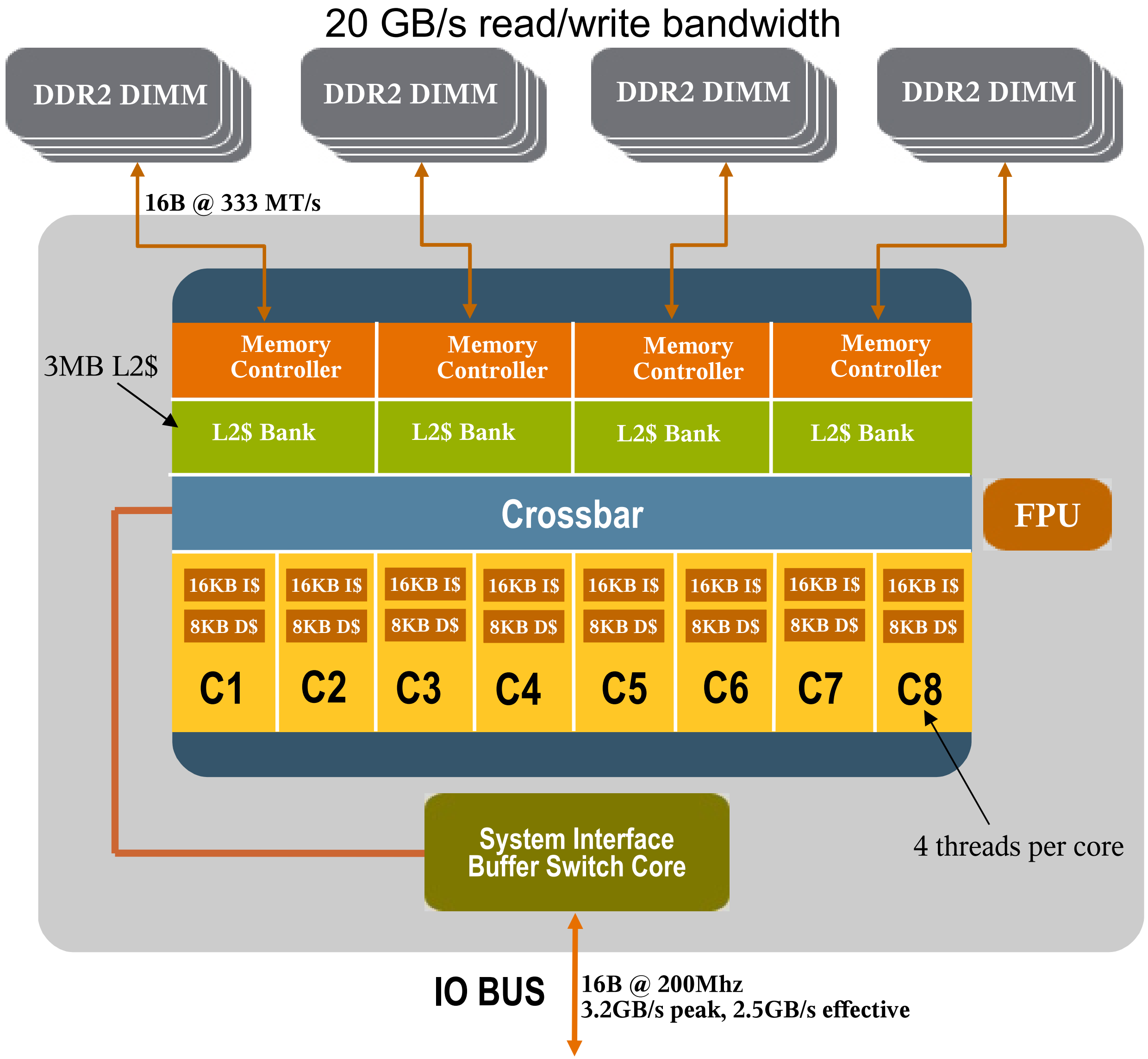
- Governed by GPL (2)
- Complete chip architecture
- Register Transfer Logic (RTL)
- Hypervisor API
- Verification suite and architectural models
- Simulation model for Solaris bringup on s/w
- 14 million lines of code

Linux

ubuntu

BSD

Get the Source ... Start Innovating!

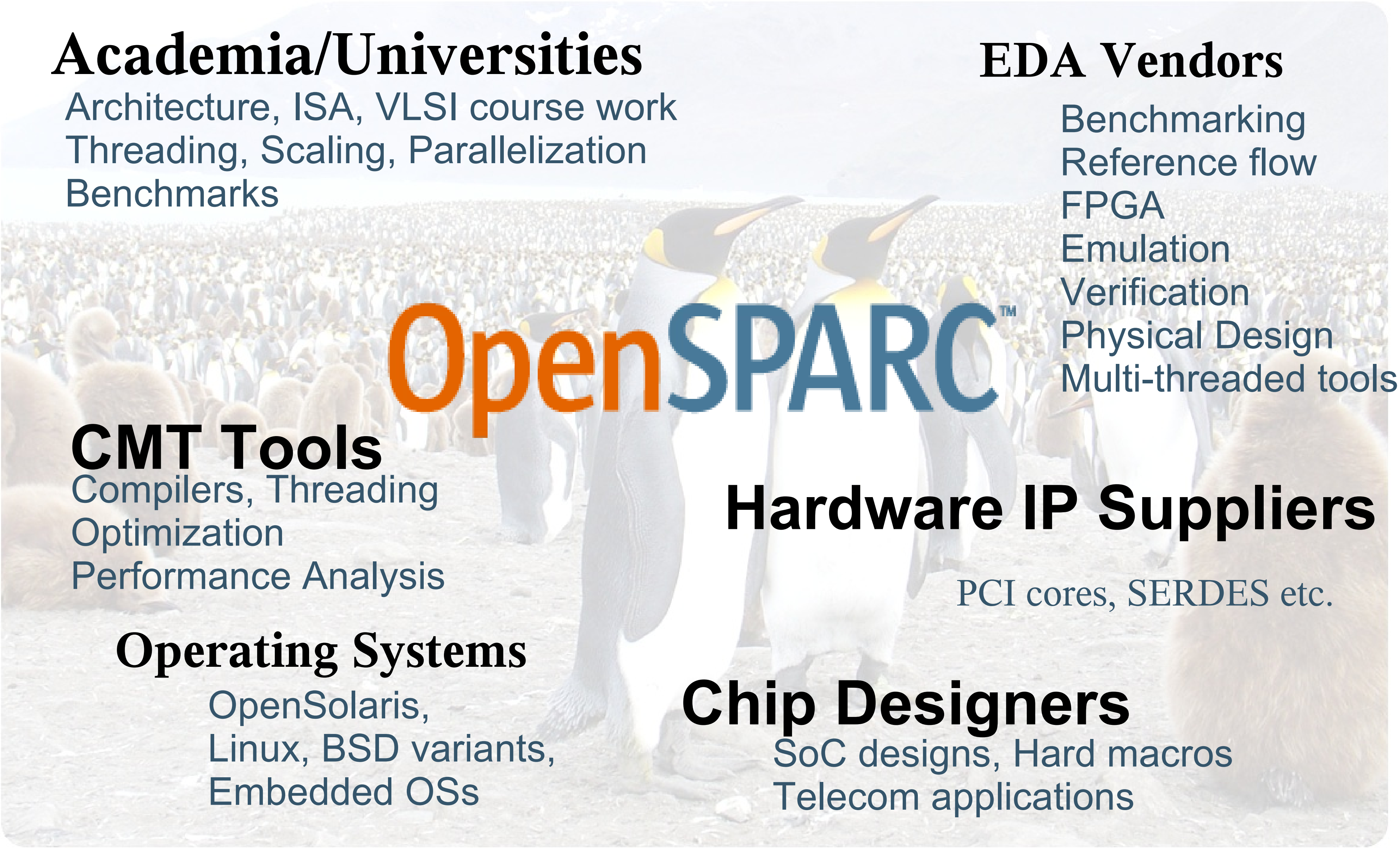


Things you can do:

- use as is
- add/delete threads
- add/delete cores
- add new instructions
- change or add FPUs
- add custom coprocessors
- add video/graphics
- add network interface
- change memory interface
- change I/O interface
- change cache/mem interface
- etc...

Innovate anywhere – within it or outside it

OpenSPARC Communities



Academia/Universities

Architecture, ISA, VLSI course work
Threading, Scaling, Parallelization
Benchmarks

EDA Vendors

Benchmarking
Reference flow
FPGA
Emulation
Verification
Physical Design
Multi-threaded tools

CMT Tools

Compilers, Threading
Optimization
Performance Analysis

Hardware IP Suppliers

PCI cores, SERDES etc.

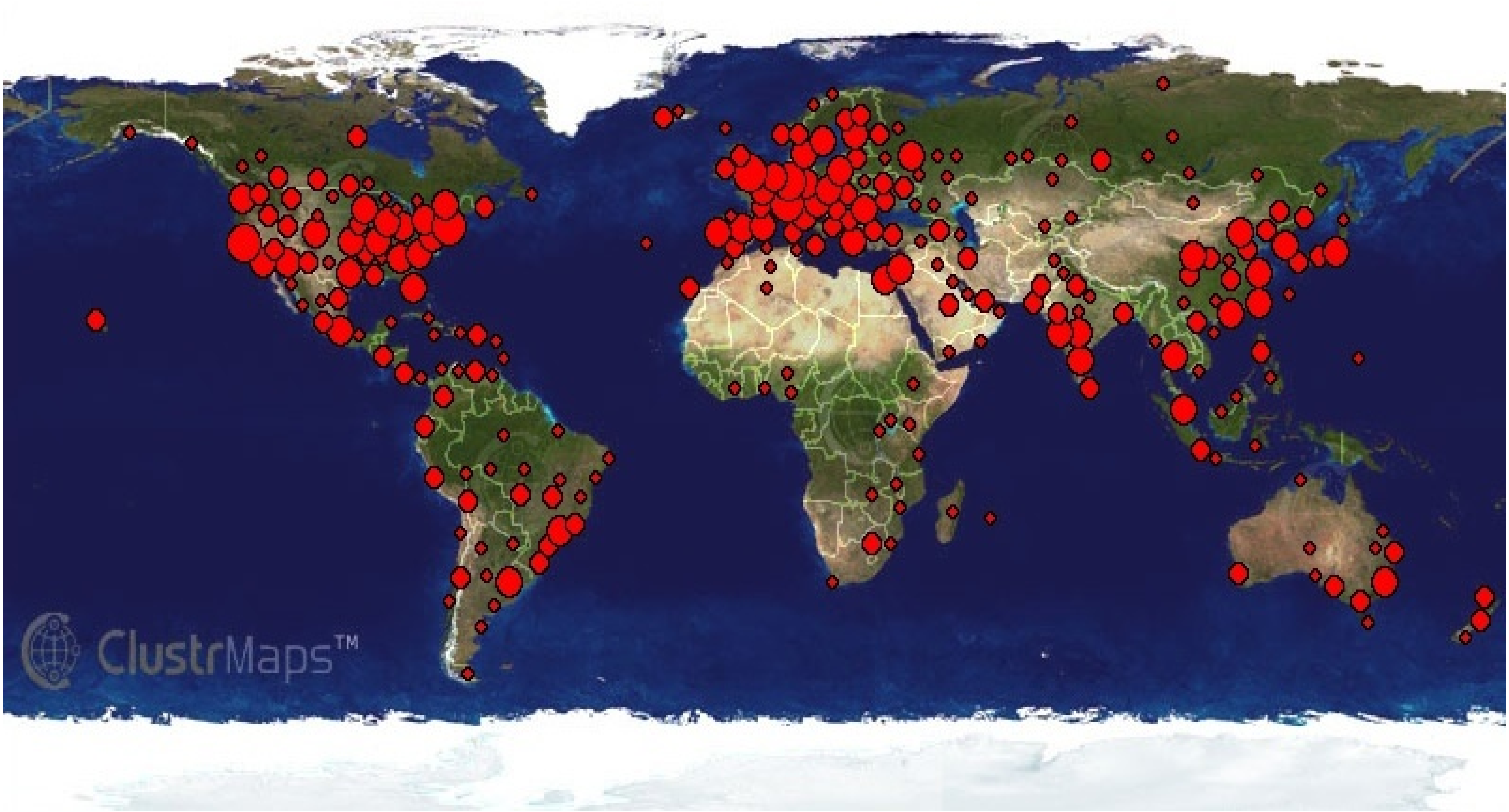
Operating Systems

OpenSolaris,
Linux, BSD variants,
Embedded OSs

Chip Designers

SoC designs, Hard macros
Telecom applications

OpenSPARC momentum



Innovation Happens Everywhere

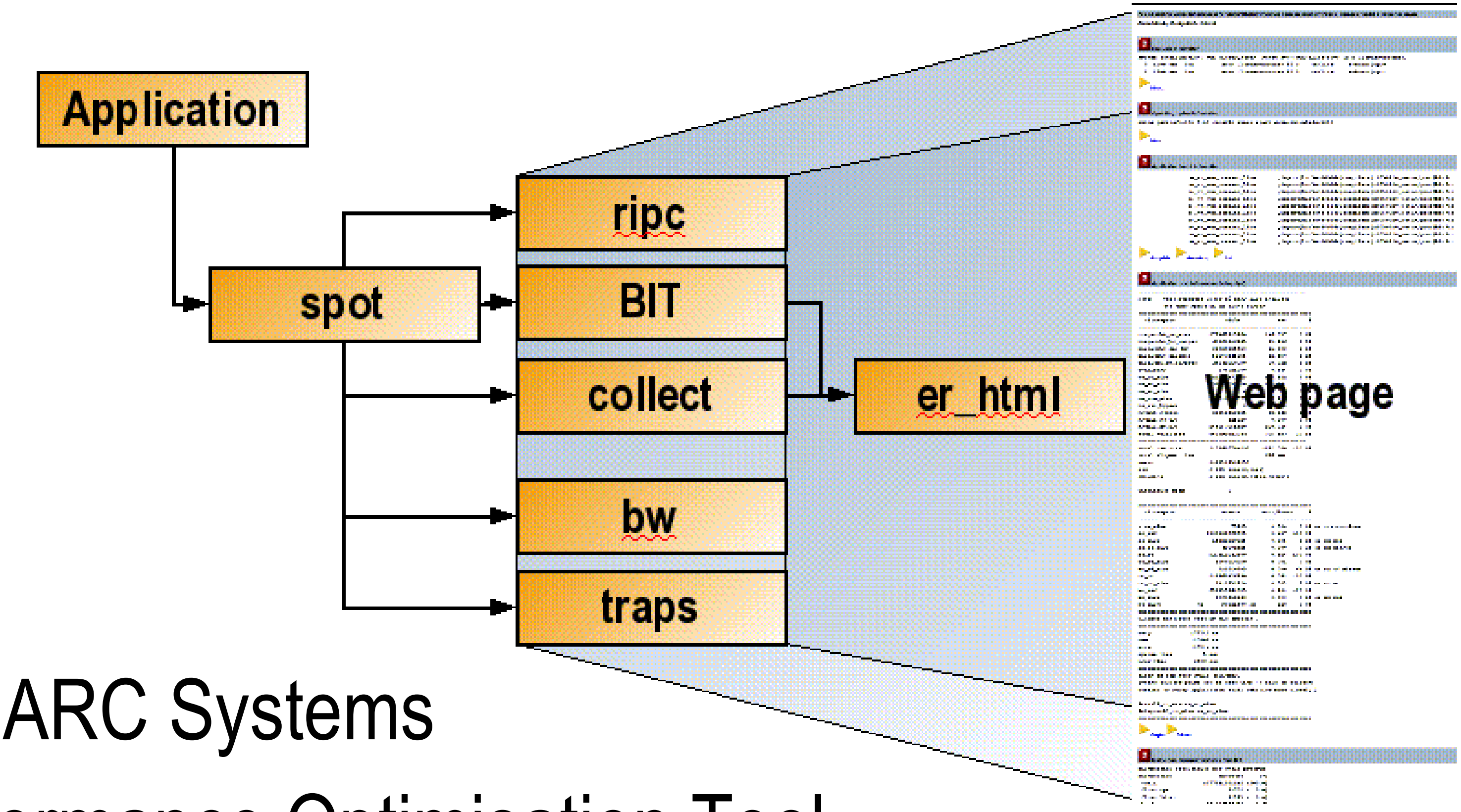
www.opensparc.net

OpenSPARC community achievements

- Single core (S1) design released by Simply RISC based in Italy (less than 6 months of effort)
- David Miller ported Linux in less than 6 weeks to T2000 system
- Cadence uses OpenSPARC for benchmarking of two generation of hardware accelelrators
- John Hennessy and David Patterson's fourth edition of “Computer architecture” book includes section on T1
- UCSC professor Jose Renau releases 65nm synthesis results
- Collaborative effort on RAMP (build 1000 core system)
- > 4500 downloads.

Cool tools for SPARC systems

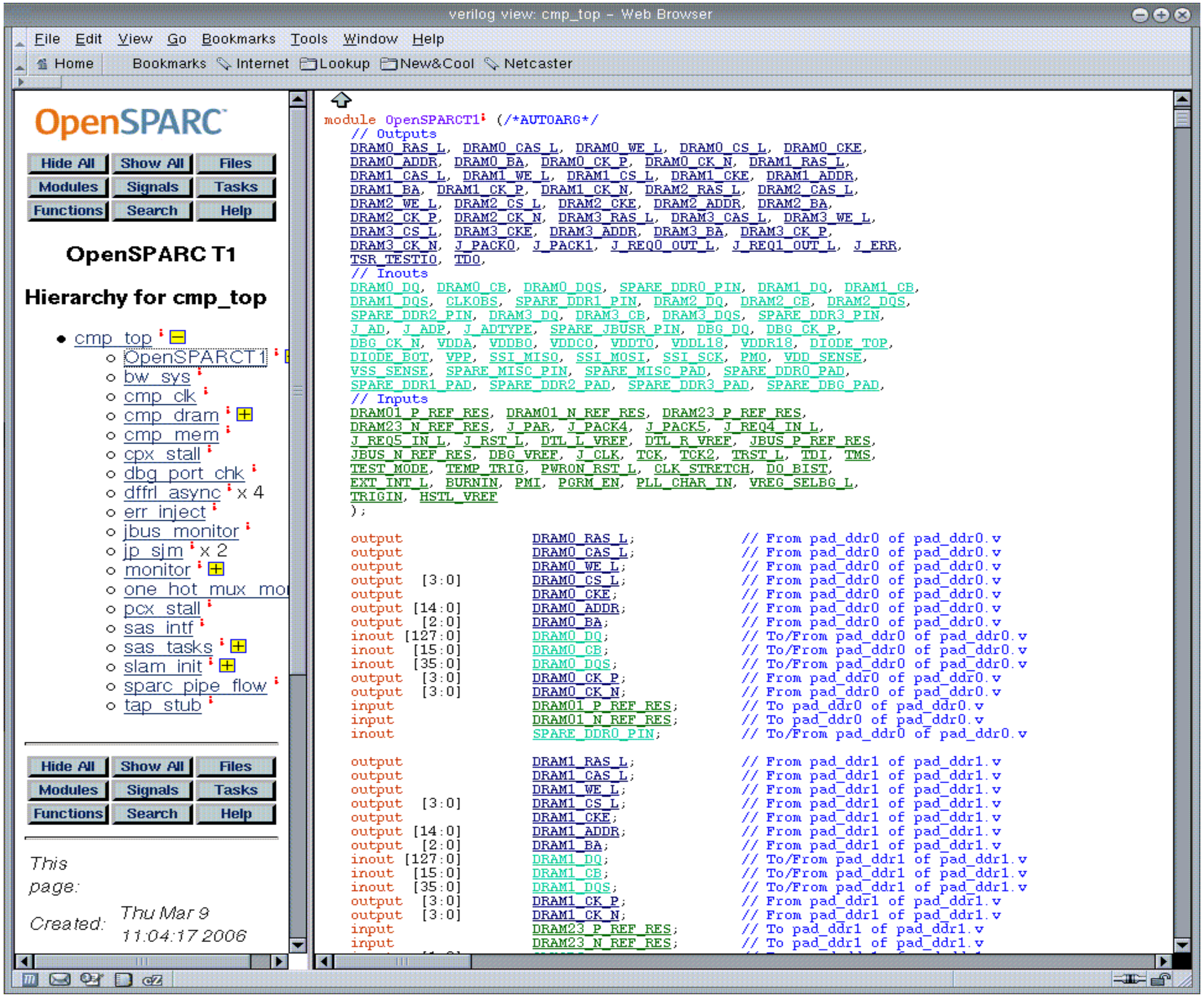
http://cooltools.sunsource.net/



- GCC for SPARC Systems
- Simple Performance Optimisation Tool
- Automatic Tuning and Troubleshooting Tool

OpenSPARC.net: Get the Source

- Browse online
- Sort by:
  - > Modules
  - > Signals
  - > Files
  - > Tasks
  - > Functions
- or Download the .tar files


verilog view: cmp_top - Web Browser
File Edit View Go Bookmarks Tools Window Help
Home Bookmarks Internet Lookup New&Cool Netcaster
OpenSPARC
Hide All
Show All
Files
Modules
Signals
Tasks
Functions
Search
Help

OpenSPARC T1

Hierarchy for cmp_top

- cmp_top
  - OpenSPARCT1
  - bw_sys
  - cmp_clk
  - cmp_dram
  - cmp_mem
  - cpx_stall
  - dbg_port_chk
  - dffrl_async x 4
  - err_inject
  - jbus_monitor
  - jp_sjm x 2
  - monitor
  - one_hot_mux_mo
  - pcx_stall
  - sas_intf
  - sas_tasks
  - slam_init
  - sparc_pipe_flow
  - tap_stub

This page:

Created: *Thu Mar 9 11:04:17 2006*

```
module OpenSPARCT1 (/*AUTOARG*/
   // Outputs
   DRAM0_RAS_L, DRAM0_CAS_L, DRAM0_WE_L, DRAM0_CS_L, DRAM0_CKE,
   DRAM0_ADDR, DRAM0_BA, DRAM0_CK_P, DRAM0_CK_N, DRAM1_RAS_L,
   DRAM1_CAS_L, DRAM1_WE_L, DRAM1_CS_L, DRAM1_CKE, DRAM1_ADDR,
   DRAM1_BA, DRAM1_CK_P, DRAM1_CK_N, DRAM2_RAS_L, DRAM2_CAS_L,
   DRAM2_WE_L, DRAM2_CS_L, DRAM2_CKE, DRAM2_ADDR, DRAM2_BA,
   DRAM2_CK_P, DRAM2_CK_N, DRAM3_RAS_L, DRAM3_CAS_L, DRAM3_WE_L,
   DRAM3_CS_L, DRAM3_CKE, DRAM3_ADDR, DRAM3_BA, DRAM3_CK_P,
   DRAM3_CK_N, J_PACK0, J_PACK1, J_REQ0_OUT_L, J_REQ1_OUT_L, J_ERR,
   TSR_TESTIO, TDO,
   // Inouts
   DRAM0_DQ, DRAM0_CB, DRAM0_DQS, SPARE_DDR0_PIN, DRAM1_DQ, DRAM1_CB,
   DRAM1_DQS, CLKOBS, SPARE_DDR1_PIN, DRAM2_DQ, DRAM2_CB, DRAM2_DQS,
   SPARE_DDR2_PIN, DRAM3_DQ, DRAM3_CB, DRAM3_DQS, SPARE_DDR3_PIN,
   J_AD, J_ADP, J_ADTYPE, SPARE_JBUSR_PIN, DBG_DQ, DBG_CK_P,
   DBG_CK_N, VDDA, VDDBO, VDDCO, VDDTO, VDDL18, VDDR18, DIODE_TOP,
   DIODE_BOT, VPP, SSI_MISO, SSI_MOSI, SSI_SCK, PMO, VDD_SENSE,
   VSS_SENSE, SPARE_MISC_PIN, SPARE_MISC_PAD, SPARE_DDR0_PAD,
   SPARE_DDR1_PAD, SPARE_DDR2_PAD, SPARE_DDR3_PAD, SPARE_DBG_PAD,
   // Inputs
   DRAM01_P_REF_RES, DRAM01_N_REF_RES, DRAM23_P_REF_RES,
   DRAM23_N_REF_RES, J_PAR, J_PACK4, J_PACK5, J_REQ4_IN_L,
   J_REQ5_IN_L, J_RST_L, DTL_L_VREF, DTL_R_VREF, JBUS_P_REF_RES,
   JBUS_N_REF_RES, DBG_VREF, J_CLK, TCK, TCK2, TRST_L, TDI, TMS,
   TEST_MODE, TEMP_TRIG, PWRON_RST_L, CLK_STRETCH, DO_BIST,
   EXT_INT_L, BURNIN, PMI, PGRM_EN, PLL_CHAR_IN, VREG_SELBG_L,
   TRIGIN, HSTL_VREF
   );

   output              DRAM0_RAS_L;            // From pad_ddr0 of pad_ddr0.v
   output              DRAM0_CAS_L;            // From pad_ddr0 of pad_ddr0.v
   output              DRAM0_WE_L;             // From pad_ddr0 of pad_ddr0.v
   output   [3:0]      DRAM0_CS_L;             // From pad_ddr0 of pad_ddr0.v
   output              DRAM0_CKE;              // From pad_ddr0 of pad_ddr0.v
   output  [14:0]      DRAM0_ADDR;             // From pad_ddr0 of pad_ddr0.v
   output   [2:0]      DRAM0_BA;               // From pad_ddr0 of pad_ddr0.v
   inout  [127:0]      DRAM0_DQ;               // To/From pad_ddr0 of pad_ddr0.v
   inout   [15:0]      DRAM0_CB;               // To/From pad_ddr0 of pad_ddr0.v
   inout   [35:0]      DRAM0_DQS;              // To/From pad_ddr0 of pad_ddr0.v
   output   [3:0]      DRAM0_CK_P;             // From pad_ddr0 of pad_ddr0.v
   output   [3:0]      DRAM0_CK_N;             // From pad_ddr0 of pad_ddr0.v
   input               DRAM01_P_REF_RES;       // To pad_ddr0 of pad_ddr0.v
   input               DRAM01_N_REF_RES;       // To pad_ddr0 of pad_ddr0.v
   inout               SPARE_DDR0_PIN;         // To/From pad_ddr0 of pad_ddr0.v

   output              DRAM1_RAS_L;            // From pad_ddr1 of pad_ddr1.v
   output              DRAM1_CAS_L;            // From pad_ddr1 of pad_ddr1.v
   output              DRAM1_WE_L;             // From pad_ddr1 of pad_ddr1.v
   output   [3:0]      DRAM1_CS_L;             // From pad_ddr1 of pad_ddr1.v
   output              DRAM1_CKE;              // From pad_ddr1 of pad_ddr1.v
   output  [14:0]      DRAM1_ADDR;             // From pad_ddr1 of pad_ddr1.v
   output   [2:0]      DRAM1_BA;               // From pad_ddr1 of pad_ddr1.v
   inout  [127:0]      DRAM1_DQ;               // To/From pad_ddr1 of pad_ddr1.v
   inout   [15:0]      DRAM1_CB;               // To/From pad_ddr1 of pad_ddr1.v
   inout   [35:0]      DRAM1_DQS;              // To/From pad_ddr1 of pad_ddr1.v
   output   [3:0]      DRAM1_CK_P;             // From pad_ddr1 of pad_ddr1.v
   output   [3:0]      DRAM1_CK_N;             // From pad_ddr1 of pad_ddr1.v
   input               DRAM23_P_REF_RES;       // To pad_ddr1 of pad_ddr1.v
   input               DRAM23_N_REF_RES;       // To pad_ddr1 of pad_ddr1.v
```

64 Bits. 32 Threads. Free.

Imagine what's next...

"Sun's decision to release Verilog source code for the UltraSPARC hardware design under a free software license is a historic step. Sun is showing its profound understanding of the forces shaping our technological future in making this decision.

-- Eben Moglen, founding director of the Software Freedom Law Center

"The free world welcomes Sun's decision to use the Free Software Foundation's GNU GPL for the freeing of OpenSPARC. We'd love to see other hardware companies follow in Sun's footsteps."

-- Richard Stallman, Free Software Foundation

Legal Substantiation – Benchmarks

- Sun Fire T2000 (8 cores, 1 chip) 14,001 SPECweb2005. IBM p5 550 (4 cores, 2 chips) 7881 SPECweb2005. IBM eServer Xseries x346 (2 cores, 2 chips) 4348 SPECweb2005. SPEC, SPECweb reg tm of Standard Performance Evaluation Corporation. Sun Fire T2000 results submitted to SPEC. Other results from www.spec.org as of December 6, 2005.
- SPECjAppServer2004 with BEA - Sun Fire T2000 (8 cores, 1 chip) 615.64 JOPS@Standard. SPECjAppServer2004 Sun Fire rx4600 (4 cores, 4 chip) 471.28 JOPS@Standard. SPEC, SPECjAppServer reg tm of Standard Performance Evaluation Corporation. Sun Fire T2000 results submitted to SPEC. Other results from www.spec.org as of 12/06/2005.
- SPECjAppServer2004 with Sun Java System Application Server. SPECjAppServer2004 Sun Fire T2000 (8 cores, 1 chip) 436.71 JOPS@Standard. SPECjAppServer2004 HPrx4640 (4 cores, 4 chip) 471.28 JOPS@Standard. SPEC, SPECjAppServer reg tm of Standard Performance Evaluation Corporation. Sun Fire T2000 results submitted to SPEC. Other results from www.spec.org as of 12/06/2005. Sun HW+SW application tier cost = $37,484.95, appl cost per JOP = $85.83. HP HW+SW application tier cost = $140,537.88, appl cost per JOP = $298.20 HP Bill of Material from http://www.spec.org/jAppServer2004/results/res2005q3/jAppServer2004-20050913-00016.html BEA pricing from http://www.awaretechnologies.com/BEA/index.html.System pricing dated 11/29/05
- Sun Fire T1000 Server (1 chip, 8 cores, 1-way) 51,540 bops, 12,885 bops/JVM. IBM x346 (2 chip, 4 cores, 4-way) 39,585 bops, 39,585 bops/JVM. IBM p520 (1 chip, 2 cores, 2-way) 32,820 bops, 32,820 bops/JVM. Dell SC1425 (2 chip, 2 cores, 2-way) 24,208 bops, 24,208 bops/JVM. SPEC, SPECjbb reg tm of Standard PerformanceEvaluation Corporation. Sun Fire T1000 results submitted to SPEC. Other resultss as of 12/6/2005 on www.spec.org

Legal Substantiation – Benchmarks (Cont'd)

- NotesBench R6iNotes Sun Fire T2000 (1x1200 MHz UltraSPARC T1, 32GB), 4 partitions, Solaris[TM] 10, Lotus[R] Domino 7.0, 19,000 users, $4.36 per user, 16,061 NotesMark tpm, 400 ms avg rt.   NotesBench R6iNotes IBM x346 (2 x 3.4 GHz Xeon processors, 8GB), 1 partition, SuSE Linux 8, Lotus[R] DominoR6.5.3, 6,50 users, $9.07 per user, 5,109 NotesMark tpm, 569 ms avg rt.More info www.notesbench.org

- Portal tests conducted on Sun Fire Sun Fire T2000 configured with 6 cores, 1.0GHz UltraSPARC T1 processor and 16GB memory. Dell 6650 configured with 4 x Intel Xeon processors at 2GHz and 4GB memory. 1 processor was active to run the test. Internal test using SLAMD Distributed Load Generation Engine AE & Mercury Load Runner. Both systems were installed with Solaris 10 OS and Sun Java Portal Server 7. Test date: 11/14/05.

- Crypto Processing RSA & DSA sign operations @ 1024-bit

Legal Substantiation – Benchmarks (Cont'd)

- Two-tier SAP ECC 5.0 Standard Sales and Distribution (SD) benchmark Sun Fire T2000 (1 processor, 8 cores, 32 threads) 1.2 GHz UltraSPARC T1, 32 GB mem, 950 SD benchmark users, 1.91 sec avg resp, MaxDB 7.5 database, Solaris 10. SAP certification number was not available at press time, please see: www.sap.com/benchmark. Benchmark data submitted for approval: 950 SD Users (Sales &Distribution), Ave. dialog resp. time: 1.91 seconds, Throughput: Fully processed order line items/hour:95,670, Dialog steps/hour: 287,000, SAPS: 4,780, Average DB req. time (dia/upd): 0.080 sec / 0.157 sec, CPU utilization of central server: 99%, central server OS: Solaris 10, RDBMS: MaxDB 7.5, SAP ECC

  Release: 5.0, Configuration: Sun Fire Model T2000, 1 processor/ 8 cores / 32 threads, UltraSPARC T1, 1200 MHz, 64 KB(D) + 128 KB(I) L1 cache, 3 MB L2 cache, 32 GB main memory. Two-tier SAP SD Standard Application Benchmark Release 4.70 (64-bit)results for the HP ProLiant DL580 (4-way, 4 procs, 4 cores, 4 threads) included 4x 3.33 Ghz Xeon, 32 GB mem, 937 SAP SD Benchmark users,1.96 sec avg response time, Cert#2005012, running Microsoft® Windows Server

  2003, Enterprise x64 Edition (64-bit) and Microsoft SQL Server 2000 Enterprise Edition (32-bit), certified on March 29, 2005. Two-tier SAP SD Standard Application Benchmark Release 4.70 (64-bit) results for the HP rx4640 (4-way, 4 procs, 4 cores, 4 threads) included 4x 1.5 Ghz Itanium2, 32 GB mem, 880 SAP SD Benchmark users, 1.89 sec avg response time, Cert#2004030, running HP-UX 11i,Oracle 9i certified on June 4, 2004. More information on SAP Benchmark results can be found at www.sap.com/benchmark

Agenda

1. Chip Multi-Threading (CMT) Era
2. Microarchitecture of OpenSPARC T1
3. OpenSPARC T1 Program
4. SPARC Architecture
5. OpenSPARC in Academia
6. OpenSPARC T1 simulators
7. Hypervisor and OS porting
8. Compiler Optimizations and tools
9. Community Participation

SPARC Architecture Generations

Generations of SPARC

- **SPARC V8** (S.I., 1987): 32-bit

- **SPARC V9** (S.I., 1994): 64-bit addr+data
  - UltraSPARC I, 1995 – VIS-1 instructions
  - UltraSPARC III, ~2000 – VIS-2 instructions
  - UltraSPARC IV, ~2004 – basic CMT

- **UltraSPARC Architecture 2005** (Sun,2005): full CMT, hyperprivileged mode
  - UltraSPARC T1, 2005 ⟵[OpenSPARC T1]

Specification Differences, V9 → UA 2005

- Formatting improvements

- More complete and more precise than SPARC V9; for example:
  - lists the specific conditions under which each exception may be raised, for every instruction
  - clarifies relative trap priorities
  - closes many old implementation dependencies
  - specifies many extensions to architecture

- Document Design:
  - Architecture Spec + Implementation Supplements

Architecture Extensions, V9 → UA 2005

- Sun’s VIS1 and VIS2 instructions
- GSR register
- Privileged register-window management instructions ALLCLEAN, OTHERW, NORMALW, and INVALW
- “Deferred” traps split into two categories
  - SPARC V9 deferred traps are now "resumable deferred" traps

Architecture Changes, V9 → UA 2005

- **Hyperprivileged** mode has been added, including:
  - several hyperprivileged registers
  - a few hyperprivileged instructions
    - notably RDHPR and WRHPR (hyperprivileged register access)
  - effects on the Tcc instruction
  - effects on the trap model
  - SIR instruction is now hyperprivileged
  - VER register is now the hyperprivileged (**H**VER)
  - full control of Chip MultiThreading (CMT) features

Architecture Changes, Earlier UltraSPARCs → UA 2005

- For Block Store instructions, an intermediate "zero" state is allowed to be observed during execution

Feature Classification in UA 2005

- Architectural features are now classified and tagged
  - Software Class (letter)
  - Implementation Class (digit)
  - allows smooth long-term architectural evolution (addition and deprecation of features)

Why Hyperprivileged Mode?

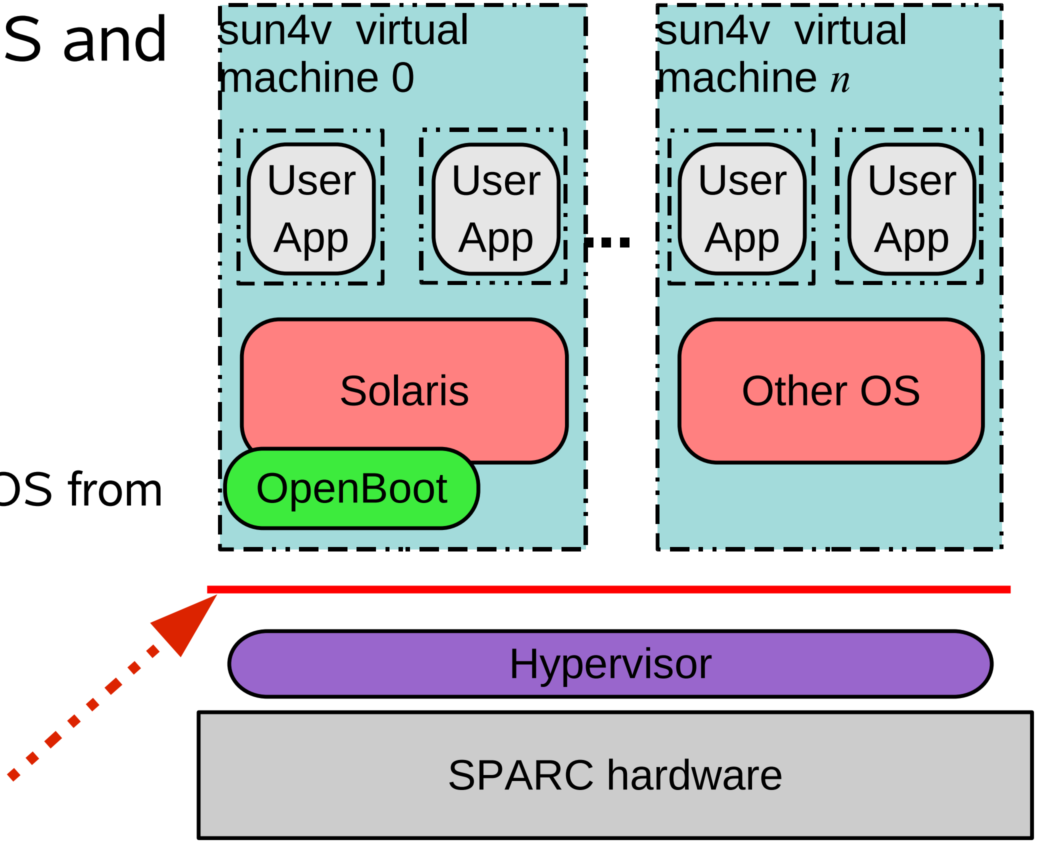
- Allows running multiple simultaneous guest OSs
  - (and/or multiple versions of the same OS)
- Allows running older OS (that uses hypervisor API) on newer hardware, without need to port the OS
- Simplifies OS ports (Linux in 2 months!)
- Allows implementation of logical domains
- Allows *virtualization*

Why Virtualization?

- Insulates higher levels of software from underlying hardware, by adding another software abstraction layer
  - > Protects customers' investment in application software from changes in underlying software (OS)
  - > Buying new, faster HW no longer requires running a new version of the OS
- Allows ability to "oversubscribe" resources (run multiple top-level software)

Virtualization

- Thin software layer between OS and platform hardware
- Para-virtualized OS
- Hypervisor + sun4v interface
  - Virtualizes machine HW and isolates OS from register-level
  - Delivered with *platform*, not with OS
  - Not itself an OS



OpenSPARC -- What’s Available

OpenSPARC T1

- Complete Solution
  - Full implementation -- CPU core, FPU
  - Tools – Verification suite, Simulation, Performance, Compiler optimization tools
  - Multiple OpenSource Operating Systems: Solaris 10, Linux, FreeBSD, etc
- All Open Source on the web
  - from OpenSPARC.net and additional web sites
- Actively enabling community for Open Sourcing of hardware and software

What’s Available – for HW Engineering

- RTL (Verilog) of OpenSPARC T1 design
  (8 cores, 32 threads – ***14 million lines of code!***)
- RTL for *reduced* OpenSPARC, for FPGA
- Synthesis scripts for RTL
- Verification test suites
- UltraSPARC Architecture 2005 spec
- UltraSPARC T1 implementation spec
- Full OpenSPARC simulation environment
- “CoolTools”, including Sun Studio software, SPARC-optimized GCC compiler, development tools, ATS, etc

What’s Available – for SW Engineering

- Architecture and Performance Modeling Package, including:
  - SAS – Instruction-accurate SPARC Architecture Simulator (includes source code)
  - SAM – SPARC instruction-accurate full-system simulator (includes source code)
  - Solaris Images for simulation: Solaris 10, Hypervisor, OBP
  - Legion – SPARC full-system simulation model for Software Developers (includes source code)
  - Hypervisor source code
  - Documentation

What’s Available – other sources

- OpenSolaris (OpenSolaris.org)
- Linux ports for T1-based systems:
  - Ubuntu
  - Gentoo
  - Wind River Linux
  - FreeBSD
- “Simply RISC” processor design based on OpenSPARC (SRisc.com)
- New Hennesey & Patterson book, Chap 4
- ...etc...

FPGA Implementations

FPGA Implementation

- Initial version released May 2006

  (on OpenSparc.net website)

  > full 8-core, 32-thread

  > First-cut implementation;
  > not yet optimized for Area/Timing

  > Synplicity scripts for Xilinx/Altera FPGAs

- Reduced version released Mar 2007 – Release 1.4

  > single-core, single-thread

  > Reduced I$/D$/TLB

  > Optimizations for Area

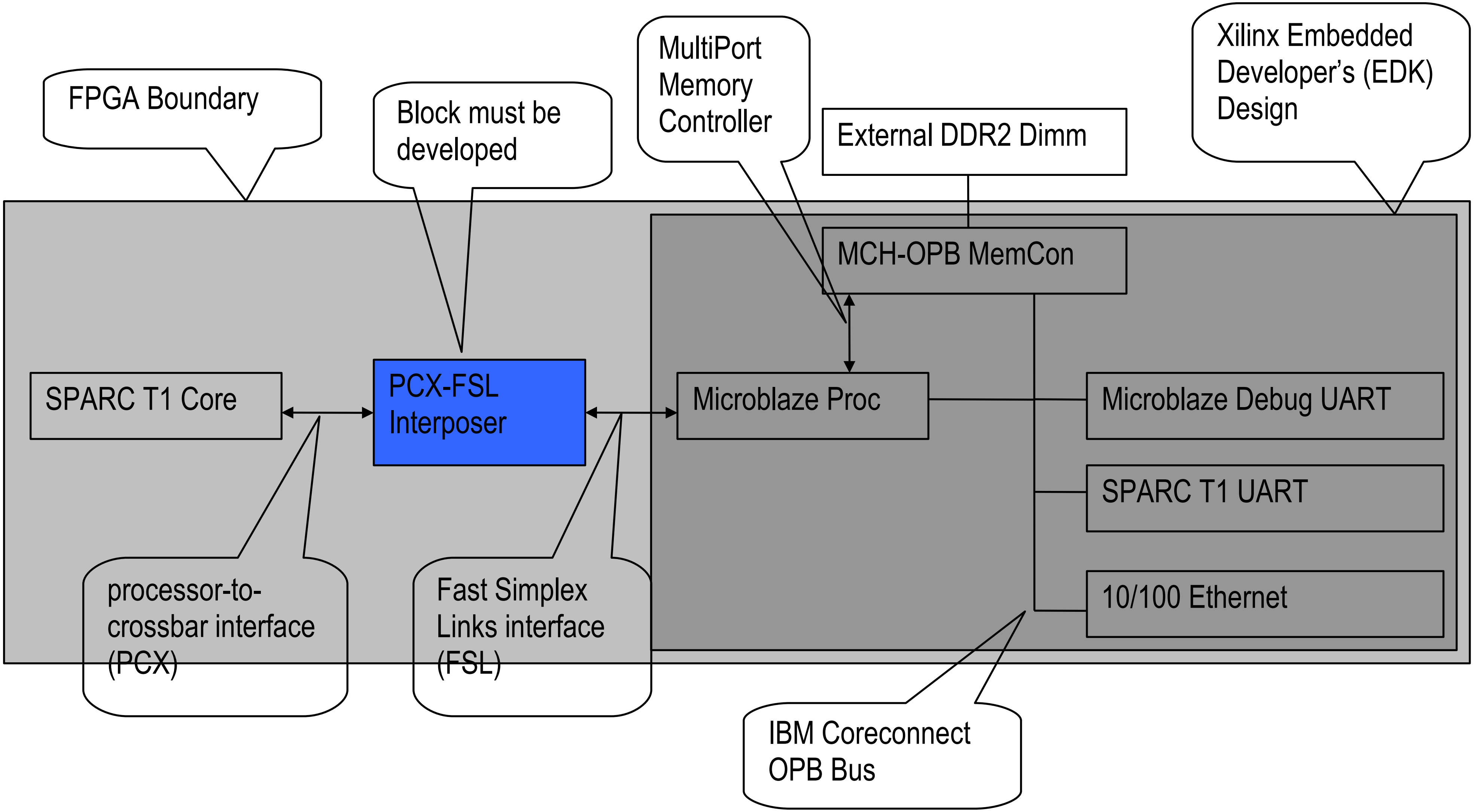
OpenSPARC FPGA Implementation

- Single core, single thread implementation of T1
  - Small, clean and modular FPGA implementation
    - About 39K 4-input LUTs, 123 BRAMs (synplicity on Virtex{2/2Pro/4})
    - Synchronous, no latches or gated clocks
    - Better utilization of FPGA resources (BRAMs, Multiplier)
  - Functionally equivalent to custom implementation, except
    - 8 entry Fully Associative TLB as opposed to 64 entry
    - Removed Crypto unit (modular arithmetic operations)

Single Thread T1 on FPGAs

- Functionally stable
  - > Passing mini and full regressions
- Completely routed
  - > No timing violations
  - > Easily meets 20ns (50MHz) cycle time
- Expandable to more threads
  - > Reasonable overhead for most blocks (~30% for 4 threads)
  - > Some bottlenecks exist (Multi-port register files)

System Block Diagram – T1 on FPGA

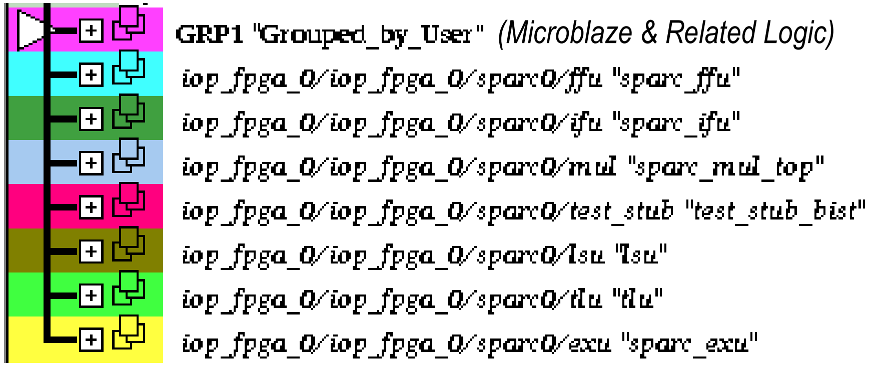


System Theory of Operation – T1 on FPGA

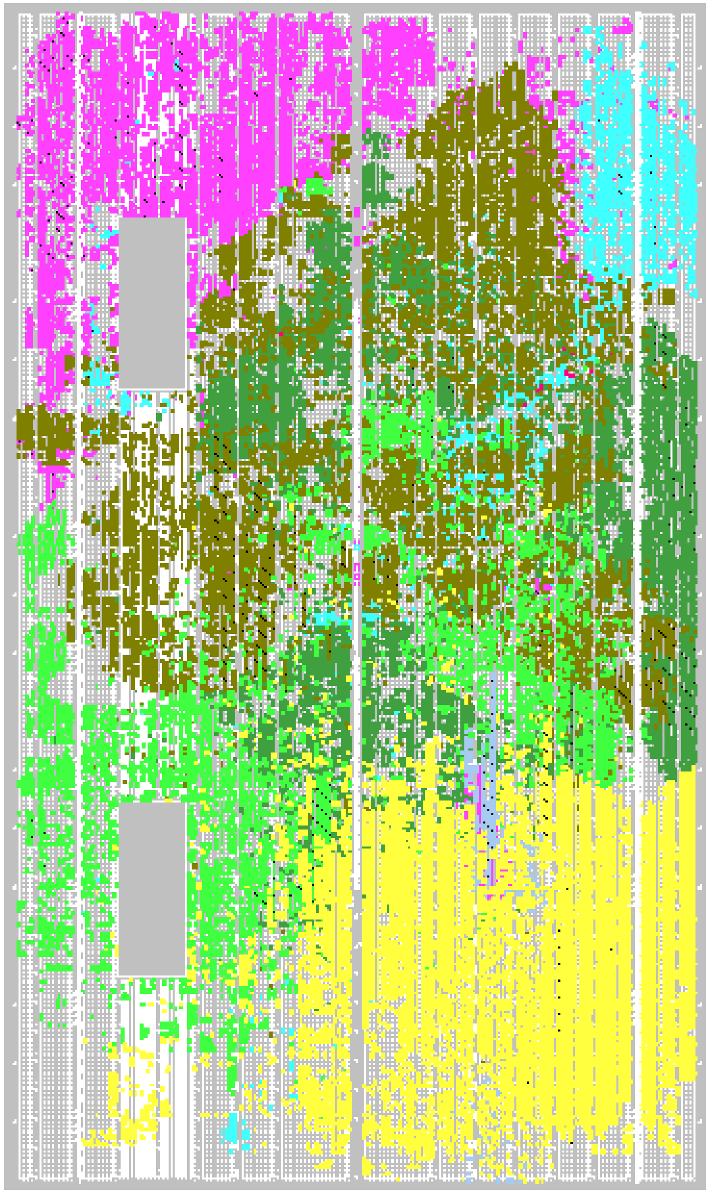
- OpenSPARC T1 core communicates exclusively via the processor-to-crossbar interface (PCX)
  - PCX is a packet based interface
- Microblaze softcore will sit in a polling loop and accept these packets, perform any protocol conversion, and forward them to the appropriate peripheral
  - Could even implement floating point operations via the Microblaze FPU unit
- Microblaze will also poll (or accept interrupts from) the peripherals, convert the info to a PCX packet, and forward it to the PCX interface
  - Microblaze has its own UART for its own diagnostic input/output

Implementation Results

- XC4VFX100-11FF1152 FPGA
  - 42,649/84,352 LUT4s (50%)
  - 131/376 BRAM-16kbits (34%)
  - 50MHz operation
    - Have not attempted any faster
  - Synplicity Synthesis: 25 minutes
  - Place and Route: 42 minutes



🟪 GRP1 "Grouped_by_User" *(Microblaze & Related Logic)*
🟦 iop_fpga_0/iop_fpga_0/sparc0/ffu "sparc_ffu"
🟩 iop_fpga_0/iop_fpga_0/sparc0/ifu "sparc_ifu"
🟦 iop_fpga_0/iop_fpga_0/sparc0/mul "sparc_mul_top"
🟥 iop_fpga_0/iop_fpga_0/sparc0/test_stub "test_stub_bist"
🟫 iop_fpga_0/iop_fpga_0/sparc0/lsu "lsu"
🟩 iop_fpga_0/iop_fpga_0/sparc0/tlu "tlu"
🟨 iop_fpga_0/iop_fpga_0/sparc0/exu "sparc_exu"



Preliminary Virtex5 Results

- Virtex5 xc5vlx1 10tff1 136
  - Same as FPGA in RAMP Bee3 board
- 30,508 6-input LUTs used out of 69,120 (44%)
- 119 used out of148 BRAM-36kbits (80%)
  - Working through mapping issues…
- 50MHz placed and routed design
  - Have not attempted any faster

OpenSPARC FPGA HW Roadmap

- Current reference design occupies about 45% of XC4V100FX FPGA. This design includes:
  - > Single core, single thread of OpenSPARC T1
  - > Microblaze to communicate with peripherals (DRAM, Ethernet)
  - > Glue logic to connect T1 core with Microblaze

- More design paths exist, e.g.

  1) Two single thread cores in single FPGA

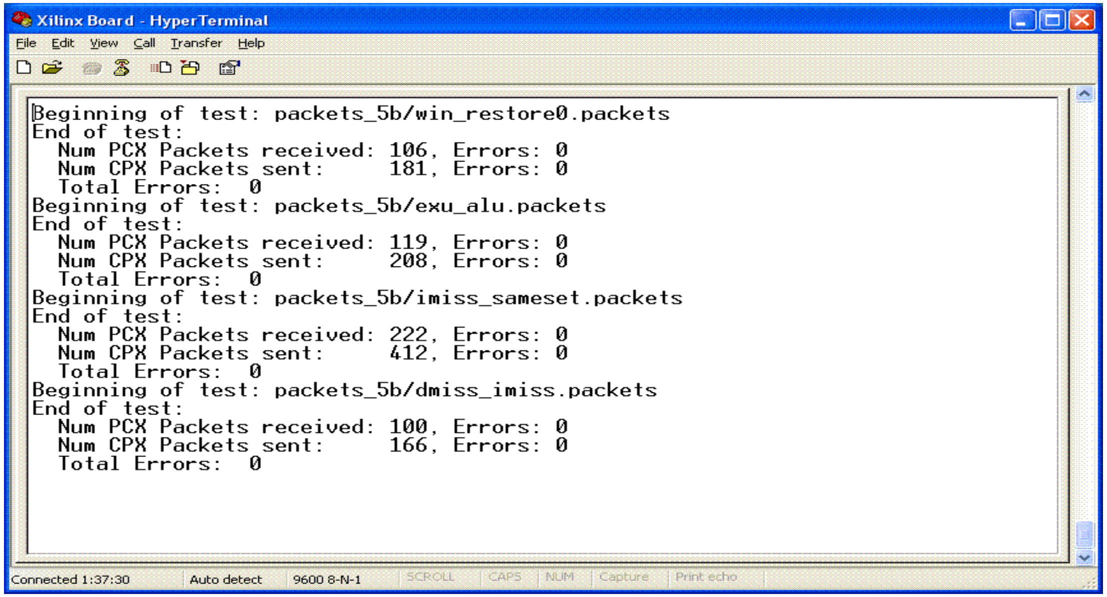
  2) Up to 4 threads per FPGA

OpenSPARC FPGA SW Roadmap

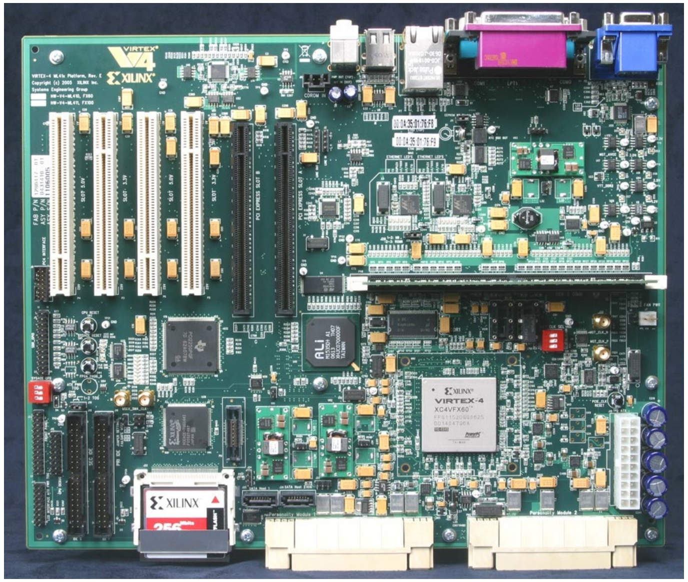
- Boot Solaris and Linux on a single thread FPGA version of the design
  - > Include support for all packet types with Microblaze
  - > Hypervisor changes to support this variant of T1
    - > Reduction in TLB size
  - > Device driver support for the system
  - > Emulation routines in OS for floating point ops
    - > Mainly for ISA compliance

FPGA Reference Design

- ml410 board with Virtex4-100 FPGA (aka ml411)
  - > Bit file and elf is stored on CompactFlash card
- Each design is a hardware implementation of one regression suite test
  - > Microblaze soft-core sends the test packets to the OpenSPARC core and verifies the return packets



```
Beginning of test: packets_5b/win_restore0.packets
End of test:
   Num PCX Packets received: 106, Errors: 0
   Num CPX Packets sent:     181, Errors: 0
   Total Errors:  0
Beginning of test: packets_5b/exu_alu.packets
End of test:
   Num PCX Packets received: 119, Errors: 0
   Num CPX Packets sent:     208, Errors: 0
   Total Errors:  0
Beginning of test: packets_5b/imiss_sameset.packets
End of test:
   Num PCX Packets received: 222, Errors: 0
   Num CPX Packets sent:     412, Errors: 0
   Total Errors:  0
Beginning of test: packets_5b/dmiss_imiss.packets
End of test:
   Num PCX Packets received: 100, Errors: 0
   Num CPX Packets sent:     166, Errors: 0
   Total Errors:  0
```



Operating Systems for OpenSPARC T1

Solaris on UltraSPARC T1

- Solaris 10 (and beyond) run on UltraSPARC T1
- Run on top of Hypervisor (“sun4v”) layer
- Fully supported by Sun and OpenSolaris

Linux Ports to date

- Sun T1000 support putback to kernel.org
  - > Bulk of support for UltraSPARC/OpenSPARC T1
  - > putback by David Miller, approx Dec 2005
  - > in 2.6.17 Linux kernel
  - > runs on top of Hypervisor

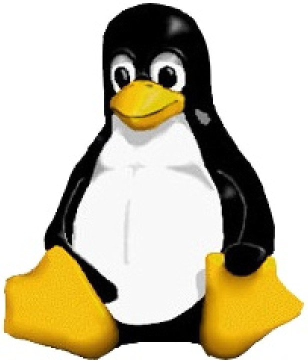
- Full Ubuntu distribution (announced ~Spring 2006)
- Gentoo Distribution (announced August 2006)
- Wind River Linux (announced October 2006)
  - > “carrier-grade” Linux, notably for Telecom applications

Linux on UltraSPARC T1

- Ubuntu 6.06 LTS support on UltraSPARC T1-based T1000/T2000
- Expands innovation and choice for developers & customers

per Colm MacCárthaigh, Senior IT Administrator at HEANet:

> “As a Linux and Apache developer, the prospect of running Ubuntu GNU/Linux – a rock solid operating system - on a CoolThreads system is exciting. I'm impressed with the innovation that's coming out of Sun these days and look for more good things going forward ”



*BSD on OpenSPARC T1

- FreeBSD port for UltraSPARC T1 announced Nov 2006

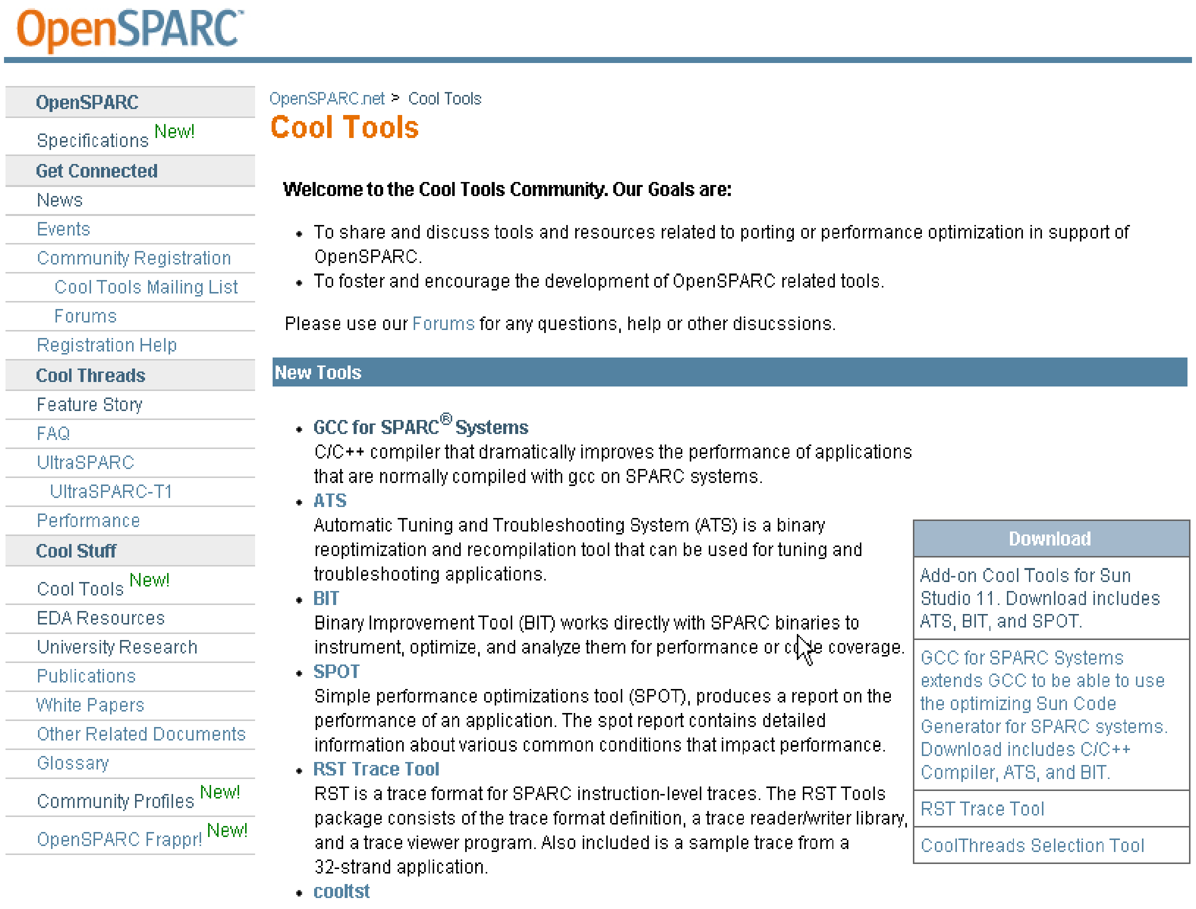
- Other *BSD ports are underway

OpenSPARC.net: Find Cool Tools

- Your resource for developer tools – FREE !
  - **GCC**
    SPARC systems highly optimized
  - **SPOT**
    Simple Performance Optimization Tool
  - **RST Trace**
  - **ATS**
    Automatic Tuning System

And –

Share *your* tools with the community at this site



OpenSPARC.net > Cool Tools

Cool Tools

Welcome to the Cool Tools Community. Our Goals are:

- To share and discuss tools and resources related to porting or performance optimization in support of OpenSPARC.
- To foster and encourage the development of OpenSPARC related tools.

Please use our Forums for any questions, help or other disucssions.

New Tools

- **GCC for SPARC® Systems**
  C/C++ compiler that dramatically improves the performance of applications that are normally compiled with gcc on SPARC systems.
- **ATS**
  Automatic Tuning and Troubleshooting System (ATS) is a binary reoptimization and recompilation tool that can be used for tuning and troubleshooting applications.
- **BIT**
  Binary Improvement Tool (BIT) works directly with SPARC binaries to instrument, optimize, and analyze them for performance or code coverage.
- **SPOT**
  Simple performance optimizations tool (SPOT), produces a report on the performance of an application. The spot report contains detailed information about various common conditions that impact performance.
- **RST Trace Tool**
  RST is a trace format for SPARC instruction-level traces. The RST Tools package consists of the trace format definition, a trace reader/writer library, and a trace viewer program. Also included is a sample trace from a 32-strand application.
- **cooltst**

Download

Add-on Cool Tools for Sun Studio 11. Download includes ATS, BIT, and SPOT.

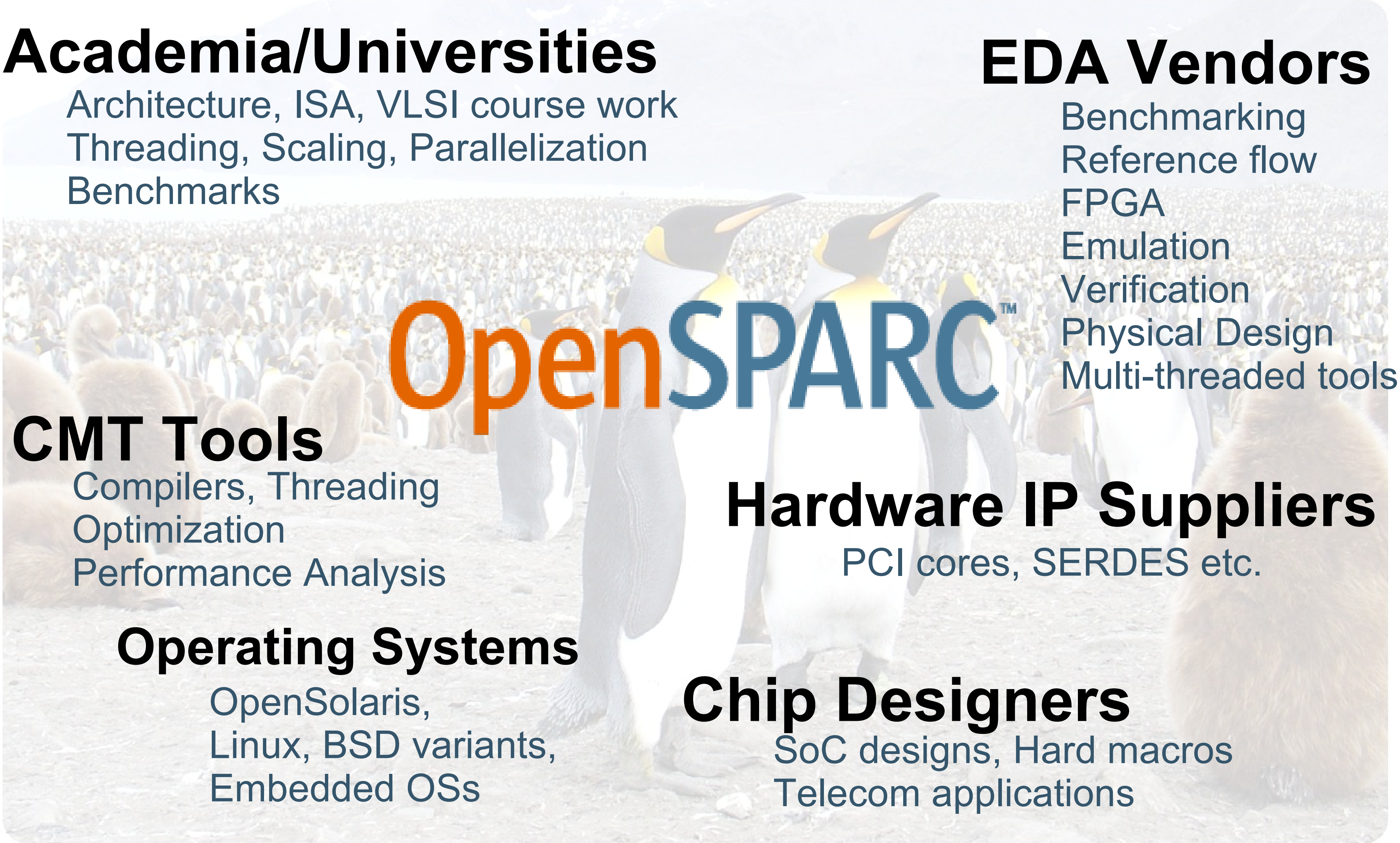
GCC for SPARC Systems extends GCC to be able to use the optimizing Sun Code Generator for SPARC systems. Download includes C/C++ Compiler, ATS, and BIT.

RST Trace Tool

CoolThreads Selection Tool

OpenSPARC Community and Governance

OpenSPARC Community Groups



Academia/Universities

Architecture, ISA, VLSI course work
Threading, Scaling, Parallelization
Benchmarks

EDA Vendors

Benchmarking
Reference flow
FPGA
Emulation
Verification
Physical Design
Multi-threaded tools

CMT Tools

Compilers, Threading
Optimization
Performance Analysis

Hardware IP Suppliers

PCI cores, SERDES etc.

Operating Systems

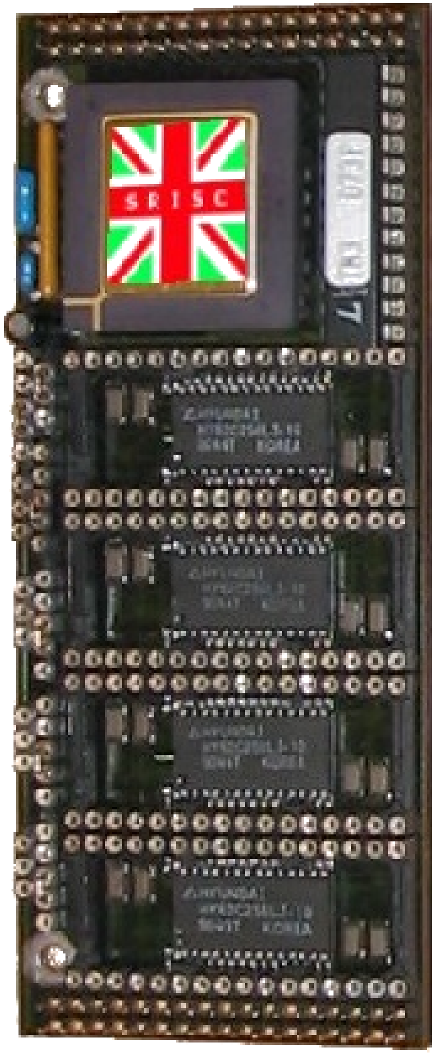
OpenSolaris,
Linux, BSD variants,
Embedded OSs

Chip Designers

SoC designs, Hard macros
Telecom applications

OpenSPARC Grows the Community

- Simply RISC “S1”
  - Single-core version of UltraSPARC T1
  - Targets small embedded devices
  - Runs Solaris and Linux
  - Design also released under GPL
- Allows Sun to grow the SPARC community by virtue of having great technology and not by handing out money



“Due to the collaborative nature of the GPL license Simply RISC plans to add new features to the S1 Core and test them extensively over the next months with the help of the community.”

http://www.srisc.com

OpenSPARC Governance Board

- Initial Advisory Board announced Sept 2006
  - 3 Community members:
    - Nathan Brookwood, industry analyst (Insight64)
    - Jose Renau, Univ. of California at Santa Cruz
    - Robert Ober, Fellow, CTO Office, LSI Logic
  - 2 members from Sun:
    - Simon Phipps, Chief Open-Source Officer
    - David Weaver, Sr. Staff Engineer, UltraSPARC Architecture
- Governance Board
  - Advisory Board became initial Governance Board Jan'07
  - New Board to be elected from Community in a year

OpenSPARC in Academia

Example Uses for OpenSPARC (1)

- Create variations from the basic design
  - more/fewer cores
  - more/fewer threads per core
  - add new instructions
  - add video/graphics
  - add network interface
  - change cache/memory interface
  - change I/O interface

Example Uses for OpenSPARC (2)

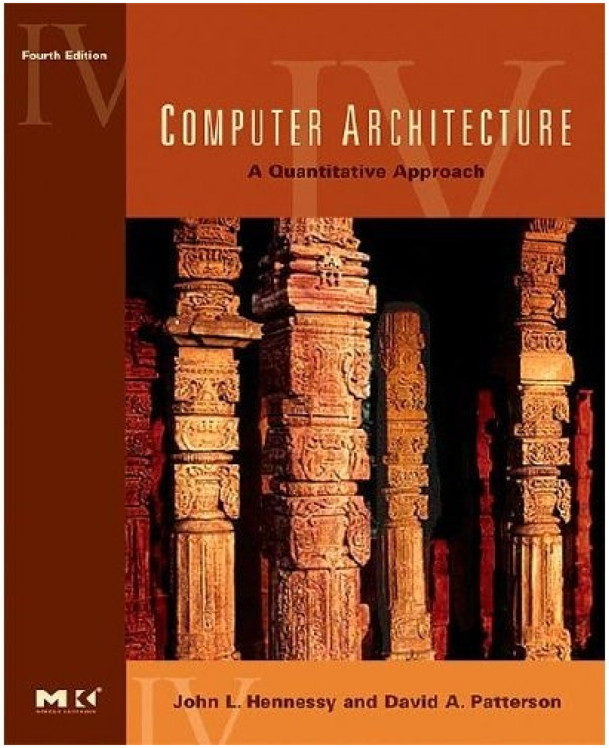
- Experimental processor designs
  - highly threaded, high-bandwith network processor
  - add more FPUs, for highly threaded HPC processing node
  - add crytographic processing elements, for high-bandwidth crypto engine
  - add coprocessors for specialized functions
  - research into optimizing useful work done per watt of power consumed (efficiency)
  - computer architecture research - add/remove instructions, new operating modes
  - port tools to other hardware and/or OS platforms (x86/x64, Linux, others)

Example Uses for OpenSPARC (3)

- Starting point for lab courses
  - a known-good design that can be modified for lab projects in computer architecture or VLSI design courses
- Real-world input to test robustness of CAD tools and simulators developed at Univ.
  - major industry CAD tool vendors already doing this!
- Burn derivative processors into FPGAs
  - quick design iterations
  - high-speed emulation
- Trigger spin-off/start-up ventures?

OpenSPARC in the Curriculum

- UltraSPARC T1 is now the “putting it all together” example of multiprocessors in the world's most iconic textbook on Computer Architecture
- Chapter on Intel Itanium was removed altogether
  - Placed on the companion CD because, according to Dr. Patterson, “it wasn't worth the paper”



Computer Architecture:
A Quantitative Approach, 4th ed.
by John Hennessy and David Patterson

...Students are emerging from university Computer Science programs already understanding SPARC!

Processor Performance

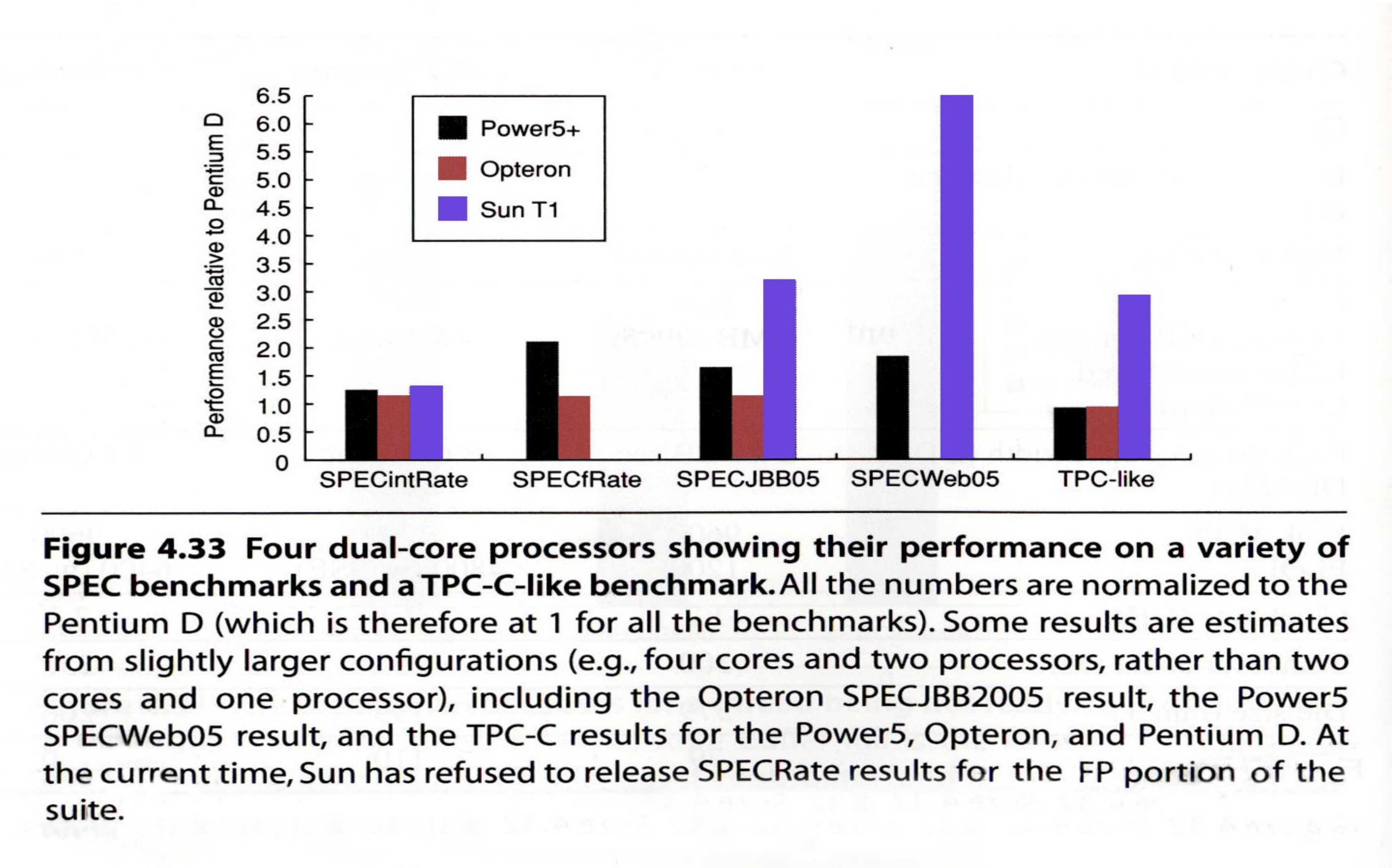


Figure 4.33 Four dual-core processors showing their performance on a variety of SPEC benchmarks and a TPC-C-like benchmark. All the numbers are normalized to the Pentium D (which is therefore at 1 for all the benchmarks). Some results are estimates from slightly larger configurations (e.g., four cores and two processors, rather than two cores and one processor), including the Opteron SPECJBB2005 result, the Power5 SPECWeb05 result, and the TPC-C results for the Power5, Opteron, and Pentium D. At the current time, Sun has refused to release SPECRate results for the FP portion of the suite.

Source: Computer Architecture, 4th ed. John Hennessy & David Patterson

Processor Performance

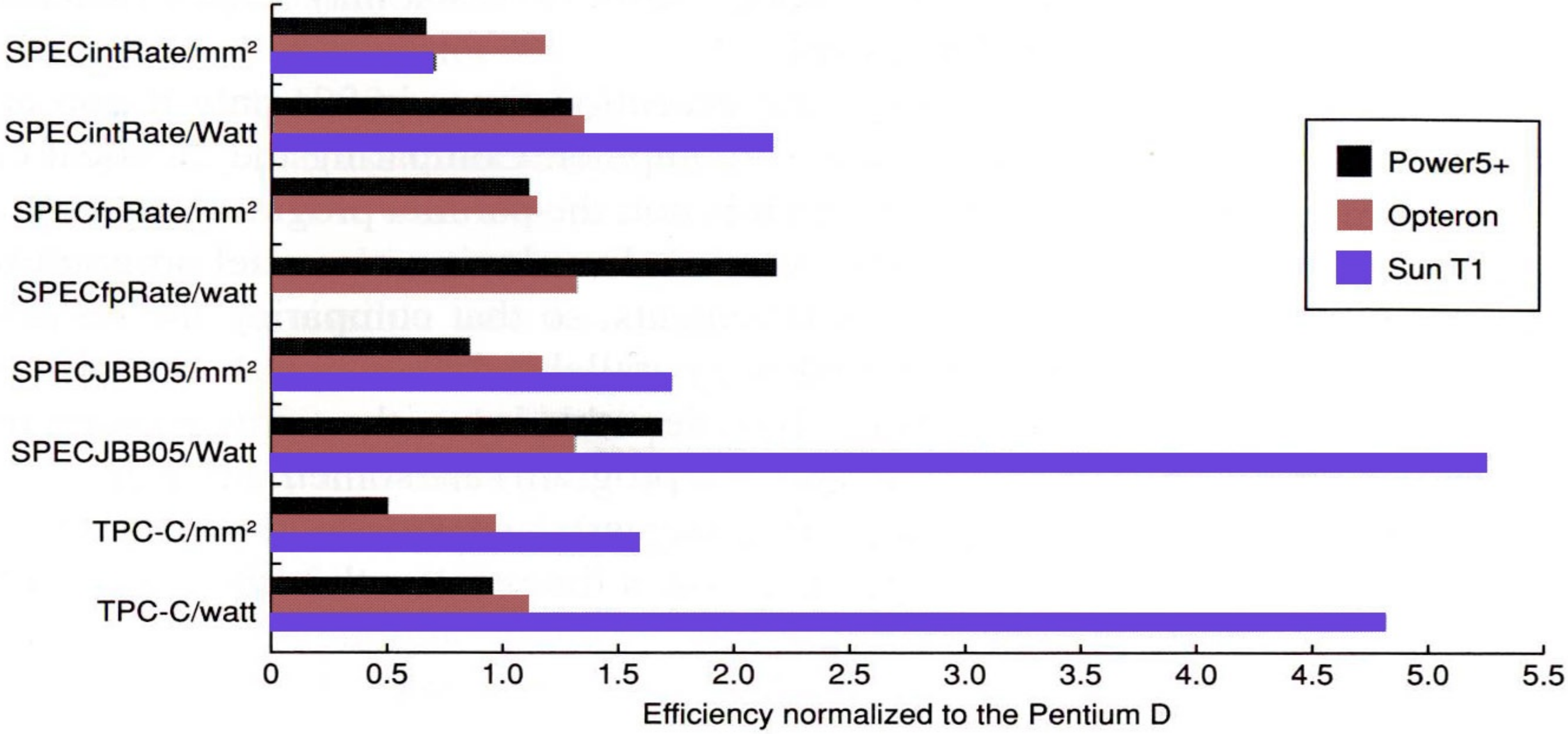


Figure 4.34 Performance efficiency on SPECRate for four dual-core processors, normalized to the Pentium D metric (which is always 1).

Source: Computer Architecture, 4th ed. John Hennessy & David Patterson

University Programs

- Sun supports/encourages academic use of OpenSPARC
  - Collaborations
  - Centers of Excellence (CoE)
    - First OpenSPARC CoE announced Feb 2007
    - more to come
  - technology access, greater equipment discounts, equipment grants, publicity and prestige that aids in obtaining other grants, other support
- For university program info, contact:
  `David.Weaver@sun.com`

Call for Action

- Participate in OpenSPARC community
  - Download, Innovate, Contribute

http://OpenSPARC.net

- Academia: apply for OpenSPARC University collaboration or Center of Excellence programs

OpenSPARC SAM-T1 Simulator

March15, 2007

Jhy-Chun Wang
Sun Microsystems

jhy-chun.wang@sun.com

Agenda

- OpenSPARC SAM-T1 overview
- RTL co-simulation
- Full-system simulation
- CPU & device model interface
- User interface
- Trace
- Disk image
- Function extension

OpenSPARC T1 Arch Tools Download

- http://opensparc-t1.sunsource.net/download_sw.html
- OpenSPARCT1-Arch_1.3.tar.bz
  - SAM: instruction-accurate SPARC full-system simulator
  - SAS: instruction-accurate SPARC arch. simulator
  - Binary images for simulation: Solaris 10, Hypervisor, OBP, etc
  - Legion: SPARC full-system simulator for software development
  - Hypervisor source code
  - Documentation

What is SAM

- **S**PARC **A**rchitecture **M**odel
- SPARC architecture golden reference model
- SPARC full-system simulator
- Functional simulator, no timing, no cache
- Usage
  - RTL verification
  - Solaris boot disk validation
  - Device model development
  - Software development
  - Benchmark tracing

Accurate CPU Simulation

- SAM models the functions of a SPARC cpu
  - SPARC instruciton set
  - memory management, TLB
  - ASI translation
  - privileged and hyperprivileged protection
  - accurate trap priority
  - IEEE floating-point instructions behavior and exceptions
- Used to verify RTL design

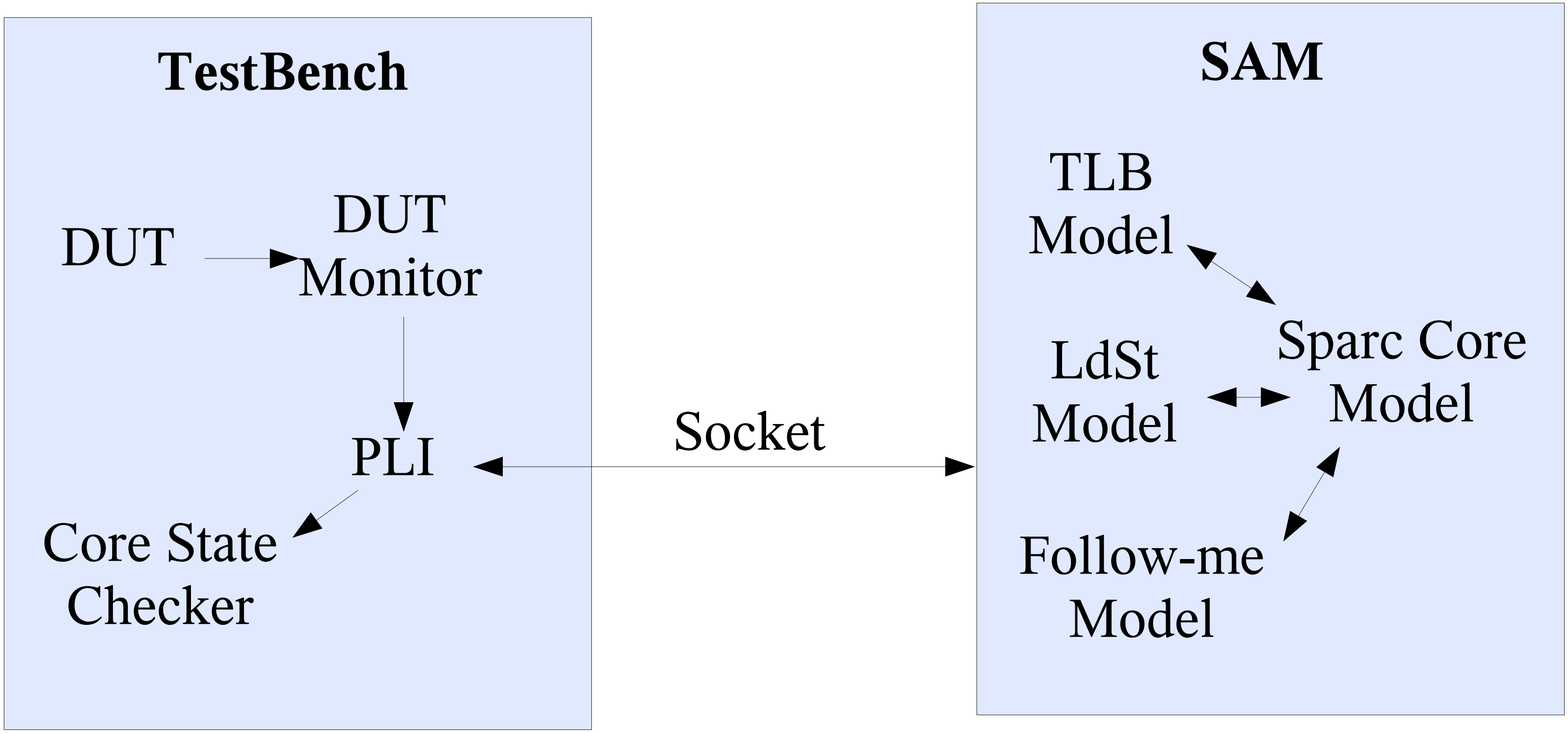
Functional Simulator

- No timing information
- Behavior will not match RTL exactly because of timing issues:
  - TLB replacement
  - deferred and disrupting traps
  - memory barrier
- No cache modeling

RTL Co-simulation

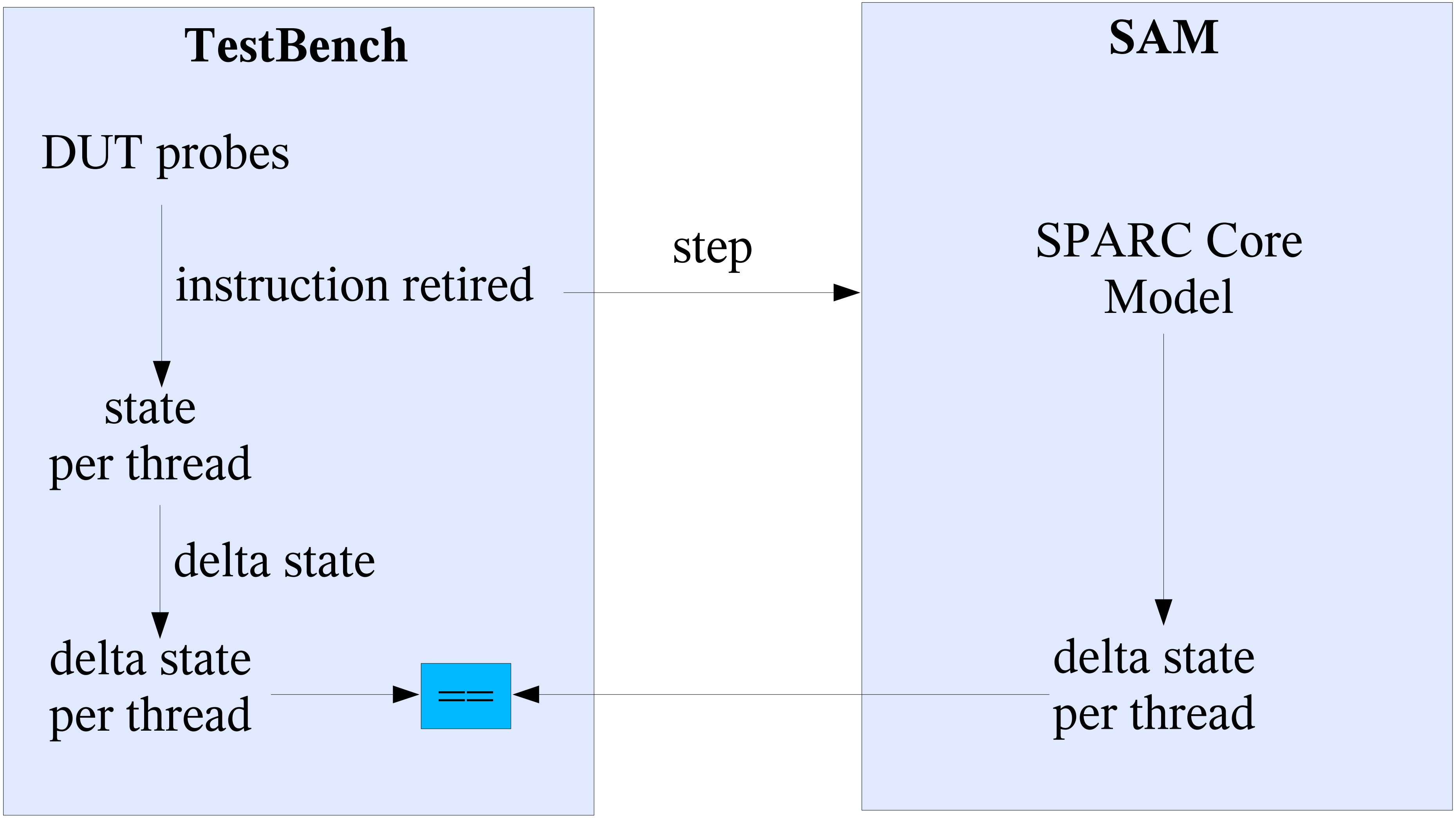
- Verify RTL by concurrently running the same instructions in RTL and SAM, and cross-checking architectural state instr-by-instr
- RTL overrides SAM's behavior to correct timing mismatches
  - TLB replacement
  - deferred and disrupting traps
  - memory barrier
  - execution latency, e.g., crypto operations
- Co-simulation is run continuously through RR
  - high level of fidelity to hardware

RTL Verification Co-simulation



* DUT: device under test, i.e., RTL

RTL Verification State Checking



TestBench

DUT probes

instruction retired

state
per thread

delta state

delta state
per thread

==

step

SAM

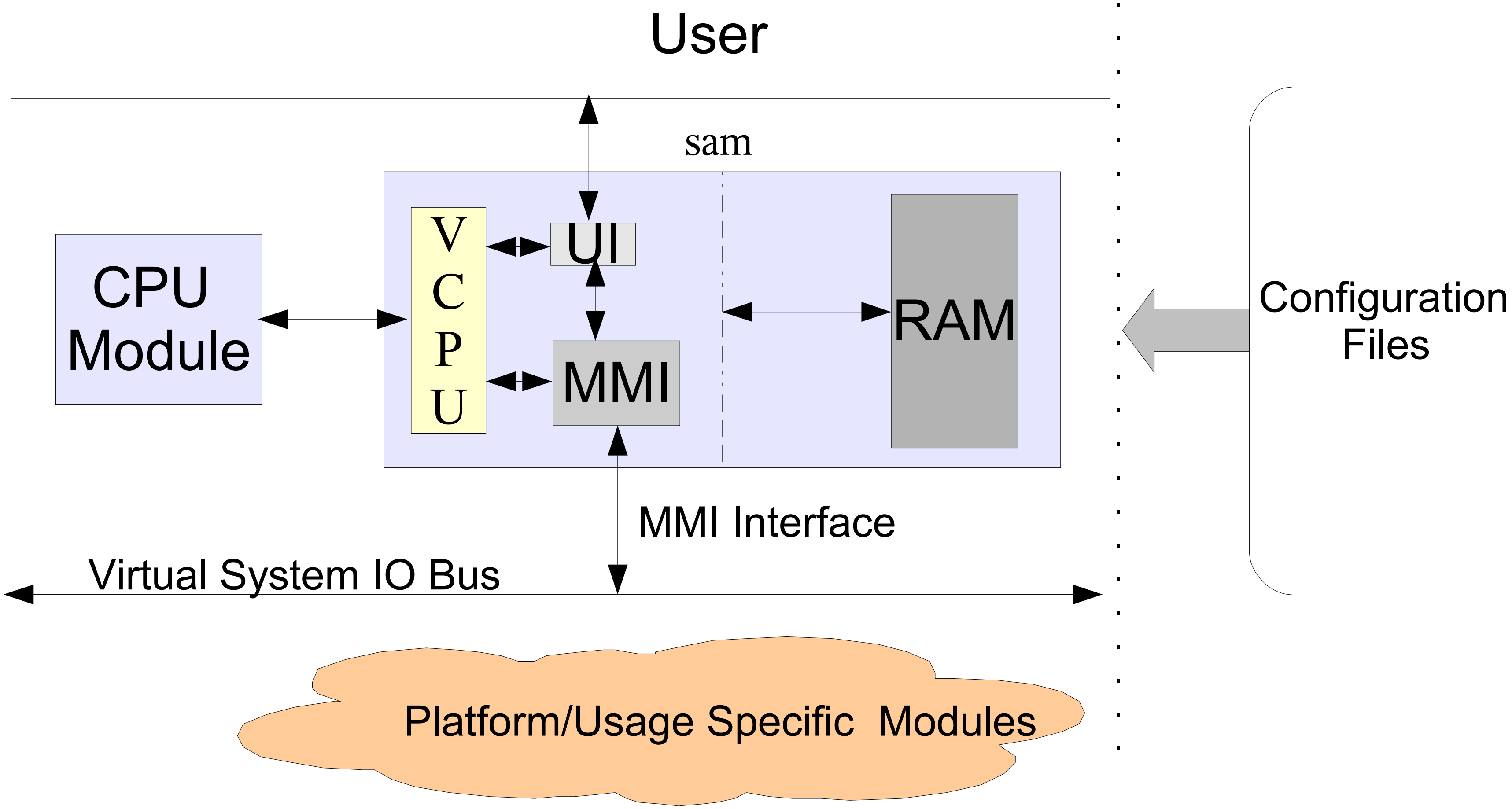
SPARC Core
Model

delta state
per thread

Full-system Simulation

- System framework with cpus and devices
- Use the same cpu module as in RTL verification, same high degree of fidelity
- Collection of dynamically loadable device modules
- Runtime configuration
- Checkpoint and restart
- Benchmark tracing
- Availability
  - > Full-system simulation available on SPARC platform
  - > RTL co-simulation available on SPARC and x86 platforms

Full-system Simulation



Configuration File

- Basic configuration
  - simulated RAM size
  - simulated MIPS
  - number of simulated cpus
- Specify the type of device and cpu modules to be loaded
- Configuration parameters for each loaded module
- Architecture state setup

Sample Configuration

- conf ramsize 64M
- sysconf cpu name=cpu0 cpu-type=SUNW,UltraSPARC-T1
- sysconf cpu name=cpu1 cpu-type=SUNW,UltraSPARC-T1
- sysconf dumbserial serial1 type=GUEST startpa=0x9f10000000 endpa=0x9f1000004f
- load bin disk.s10hw2 0x1f40000000
- load bin nvram1 0x1f11000000
- load bin 1up-hv.bin 0x1f12080000
- load bin 1up-md.bin 0x1f12000000
- load bin reset.bin 0xfff0000000
- setreg pc 0xfff0000020
- setreg npc 0xfff0000024

VCPU: Virtual CPU Interface

- Interface between cpu model and system framework
- Control instruction execution
- Interface memory access
- Interface I/O activity
- Interface interrupt handling
- Access architecture registers and state

MMI: Modular device Model Interface

- Device model implemented as loadable module
- Map device instance to phys I/O address
- Perform DMA
- Trigger interrupts
- Model device hierarchy
- Define model specific user commands
- Interface to share device models with other system simulators

User Interface

- Allow users fine control of execution
  - penable, pdisable, stepi, run, stop
- Set breakpoints by PC and strand(s)
  - break, enable, disable, delete
- Architectural state is readable and writable
  - registers, TLB's, ASI's, ASR's
    - read-reg, read-fp-reg-i, pregs
  - memory
    - get, set
- Dynamically load/unload device and trace modules
  - mod load, mod unload

User Interface ...

- Control trace collection
  - > mod load analyzer rstracer.so
  - > rstrace -o out-file -n #instr
- Checkpoint and restart
  - > dump
- New commands defined by loadable modules
- Use python as underlying scripting processor, allow native python statements from command prompt

Tracing

- Rstracer: a loadable trace module
- Collect architecture state
  - instructions
  - traps
  - TLB updates
  - DMAs
- Write RST records to output trace file(s)
- Data is compressed on-the-fly by RSTZIP
- Tools to examine & analyze trace files
- Trace data can be fed into performance models (pipeline, cache model)

Trace Collection

- stop
- mod load analyzer rstracer.so
- rstrace -o output-file -n #instr
- run N
- rstrace off
- mod unload analyzer

Trace Analysis

- trconv, the “swiss army knife” of rst tool
- Process data by
  - cpu#
  - record range
  - PC/EA range
  - summary
- Use together with rstunzip to process trace data

Sample Trace Records

- `rstunzip trace_file | trconv -c`

```
Counted: 20451321 records
      9999998 instruction recs
      1 header recs
      1 traceinfo recs
      1071 tlb recs
      ...
      2288 trap recs
      ...
      8687727 regval recs
      ...
```

Sample Trace Records ...

- `rstunzip trace_file | trconv -i`

```
RST trace format (stdin)
================
                              User/                                                       Branch
 Rec # Type                   Priv          PC              Disassembly                   Taken        EA
   49 instr   : cpuid=0 p [0x0000000001063254] or %o2, 0x80, %o1
   67 instr   : cpuid=0 p [0x0000000001063258] stb %o1, [%o0 + 0x4b]                                   [0x0000070002503e4b]
   68 instr   : cpuid=0 p [0x000000000106325c] call 0x1069764                       T          [0x0000000001069764]
   71 instr   : cpuid=0 p [0x0000000001063260] ldx [%fp + 0x7f7], %o0                           [0x000000000180b7b8]
   73 instr   : cpuid=0 p [0x0000000001069764] jmpl %o7 + 8, %g0                     T          [0x0000000001063264]
```

Modify Disk Image

- Use lofiadm to add/remove/modify files in disk image
  - with root access
  - lofiadm -a /absolute-path/disk_image
  - (assume /dev/lofi/1 is the returned value)
  - mount /dev/lofi/1 /mnt
  - cd /mnt
  - add/remove/modify files in /mnt
  - umount /mnt
  - lofiadm -d /dev/lofi/1

Transfer File to/from Simulator

- Transfer file to/from simulator without lofiadm
- Transfer speed is slow, only good for small file
- First boot the Solaris disk image up to shell prompt
  - enter ^] (ctrl-right-bracket) to get netcons> prompt
  - get_file <path-sim-file> <path-dest-file>
  - put_file <path-src-file> <path-sim-file>
  - cksum path-sim-file
  - cksum path-src/dest-file

Save Modified Disk Image

- During initialization
  - UI(load): loading <disk1> memory image
  - loading disk1, base addr 0x1f4000000, size 0x200000
- Sync up file system first
  - `sync`
  - `halt`
    - syncing file system... done
    - Program terminated
- `stop`
- `memdump new_disk 0x1f4000000 0x200000`
- use lofiadm/mount to examine files

Function Extension

- Add/remove/modify
  - instructions
  - ASI's
  - traps
  - TLB
  - memory access
- Change system configuration

Basic Source Code Structure

- Mimic SAM-T1, 1-cpu x 8-core x 4-thread
- sam-t1/src/riesling-cm/riesling/src:
  - system/Ni/Ni_System.cc
  - cpu/Ni/Ni_Cpu.cc
  - core/Ni/Ni_Core.cc
  - strand/Ni/Ni_Strand.cc
  - mmu/Ni/Ni_Mmu.cc
  - trap/Ni/Ni_Trap.cc
  - asi/Ni/Ni_Asi.xml

Instruction Execution Flow

- Determine PC
- ITLB
- Fetch instruction
- Decode instruction
- Execute instruction
- (DTLB)
- Retire instruction

Instruction Execution: Ni_Strand::step()

- handle pending interrupt
- ITLB: Ni_Mmu::handleInstr()
- fetch, decode: Ni_InstructionWord
- execute: Ni_InstructionEmulator::execute(), Ni_Fgu::execute()
- DTLB: Ni_Mmu::handleData()
- handle trap: Ni_InstructionEmulator::handleTrap()

ITLB: Ni_Mmu::handleInstr()

- check translation bypassing
- check trap violation:
  - > *mem_address_not_aligned*
  - > *instruction_access_exception*
  - > *fast_instruction_access_MMU_miss*
  - > *inst_real_translation_miss*
  - > etc
- address translation

Decode: Ni_InstructionWord

- decode op/rd/rs1/rs2/etc
- check illegal_instruction violation
  - wrong instruction syntax
  - non-zero reserved bits
  - etc
- check privileged_opcode violation

Execute: Ni_InstructionEmulator::execute()

- map instruction to corresponding exec_inst()
  - exec_retry()
  - exec_rdasr()
- basic SPARC instructions are in sam-t1/src/riesling-cm/riesling/src/strand/bcore
- some functions are in SPARC assembly for performance reason
- x86 version is available in bcore/v9_inst_c.c

DTLB: Ni_Mmu::handleData()

- check translation bypassing
- check trap violation:
  - mem_address_not_aligned
  - privileged_action
  - VA_watchpoint
  - data_access_exception
  - fast_data_access_MMU_miss
  - data_real_translation_miss
  - fast_data_access_protection
  - etc
- address translation

Handle Trap: Ni_InstructionEmulator::handleTrap()

- Normal traps
  - privileged traps
  - hyperprivileged traps
- RED_state traps
  - nonreset traps
  - POR, WMR, XIR, SIR
- Ensure correct trap priority
- Update trap-related architecture state

ASI Handling

- asi/Ni/Ni_Asi.xml
- asiReadHandler() & asiWriteHandler()

```
<asi>
   <name>ASI_I_TLB_DATA_ACCESS</name>
   <value>0x55</value>
   <access>RW</access>
   <priv>HYPER</priv>
   <handler>
            <class>Riesling::Ni_Mmu</class>
            <id>Riesling::Ni_Mmu::I_TLB_DATA_ACCESS</id>
   </handler>
   <start_va>0x0</start_va>
   <end_va>0x7f8</end_va>
</asi>
```

Major Extension Points

- Ni_InstructionWord
- Ni_InstructionEmulator
- Ni_Mmu
- Ni_Trap
- Ni_Asi.xml

open

64 bit, 32 threads, free

http://OpenSPARC.net

Jhy-Chun Wang
Sun Microsystems

Legion

- Full-system simulator for firmware and software development
- Implement enough architecture state to boot up Solaris
- Share the same disk image and binary files (Hypervisor/OBP/reset/etc) with SAM
- Startup script run_legion.sh
- Available configuration: 1-thread, 2-thread, 32-thread

```
run_legion.sh 1/2/32  [options]
```

Legion Runtime Options

- Available options
  - `-debug debug_bits`
  - `-t #physical_cpu`
  - `-h`
- Debug options
  - 0x2: PC & instruction
  - 0x8000: hypervisor calls
  - 0x20000: exceptions, XIR
  - 0x100000: TLB miss
  - 0x400000: trap, TSTATE
  - etc

Legion Runtime Display

- Use ~ (tilde) on 'guest console' window to dump out architecture state
  - ~z: exit simulation
  - ~i: dump I-TLB content
  - ~d: dump D-TLB content
  - ~b: toggle debug enable bits
  - etc

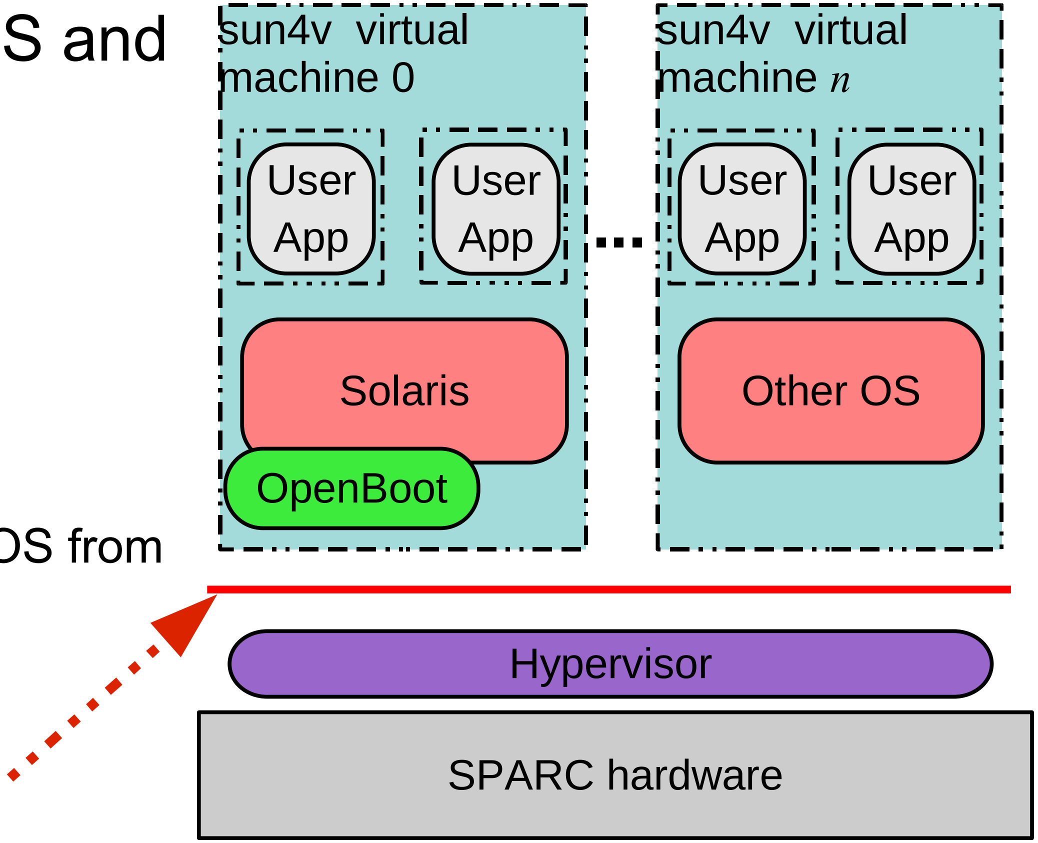
The sun4v Operating Environment

(aka “Your OS on the T1 Hypervisor”)

David Weaver

Virtual Machine for SPARC

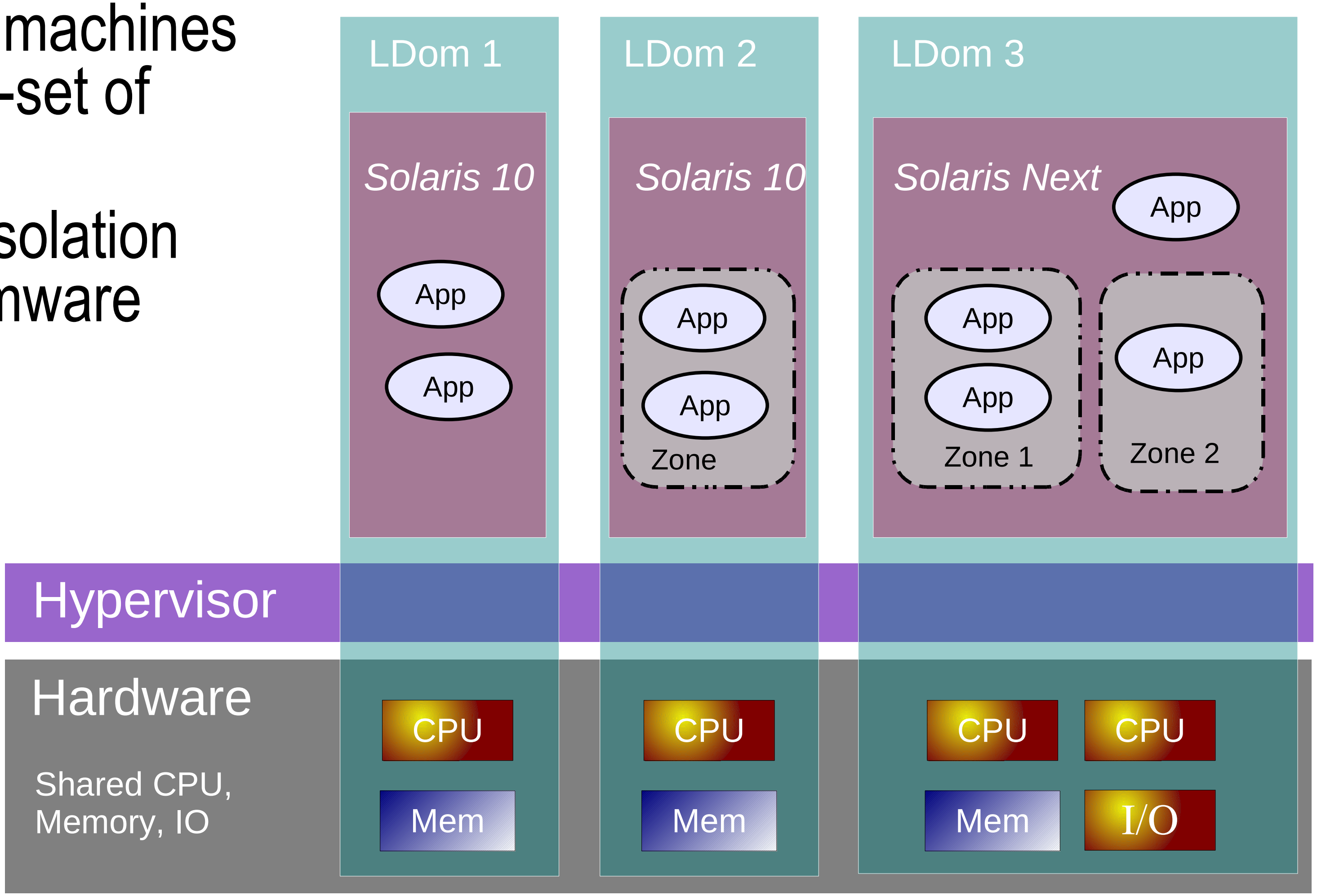
- Thin software layer between OS and platform hardware
- Para-virtualised OS
- Hypervisor + sun4v interface
  - Virtualises machine HW and isolates OS from register-level
  - Delivered with platform not OS
  - Not itself an OS



stable interface “**sun4v**”

Logical Domains

- Partitioning capability
  - > Create virtual machines each with sub-set of resources
  - > Protection & Isolation using HW+firmware combination

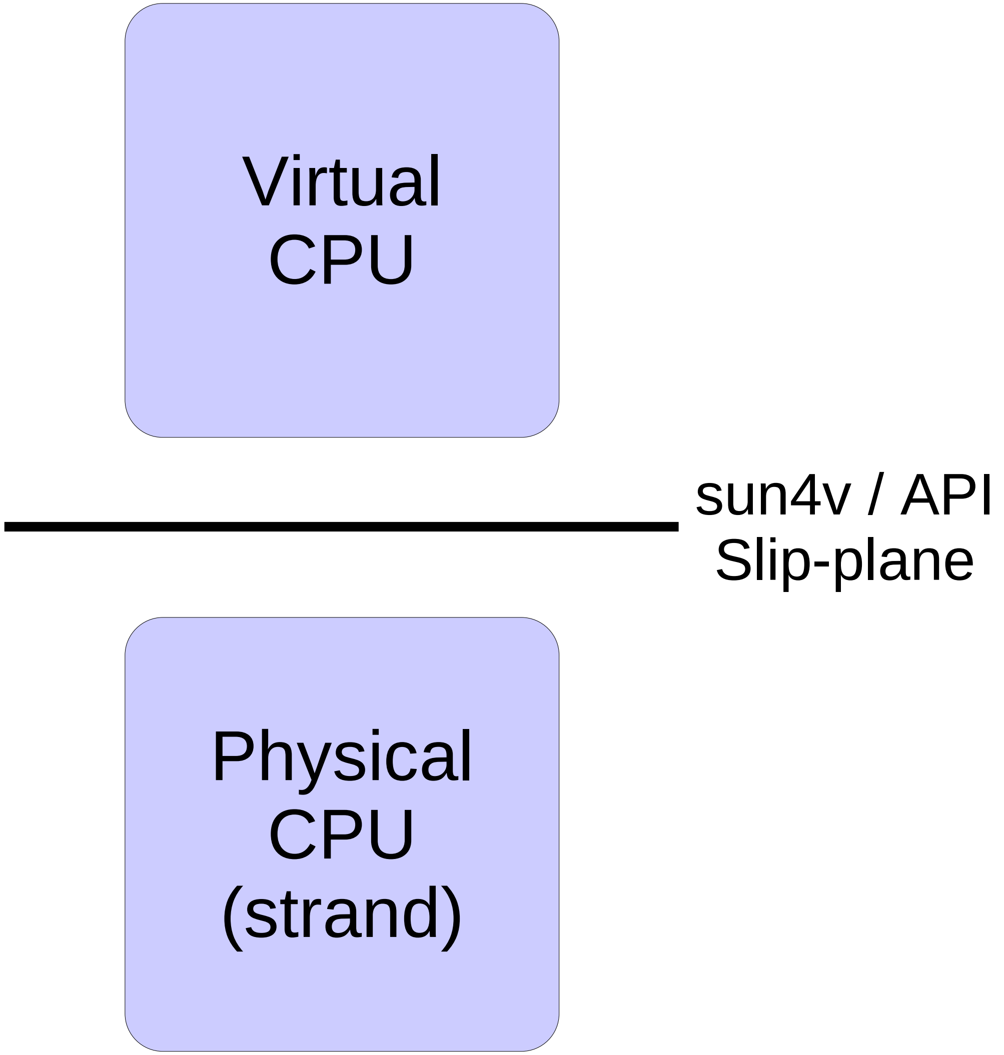


Topics

- CPU changes
- Memory management
- I/O
- Interrupts
  - > x-cpu & devices
- Multiple Domains
- Additional Topics
  - > Error handling
  - > Machine Description
  - > Boot process

Basic Principles

- Ability to rebind virtual resources to physical components at any time
- Minimal state held in Hypervisor to describe guest OS
- *Never* trust Guest OS



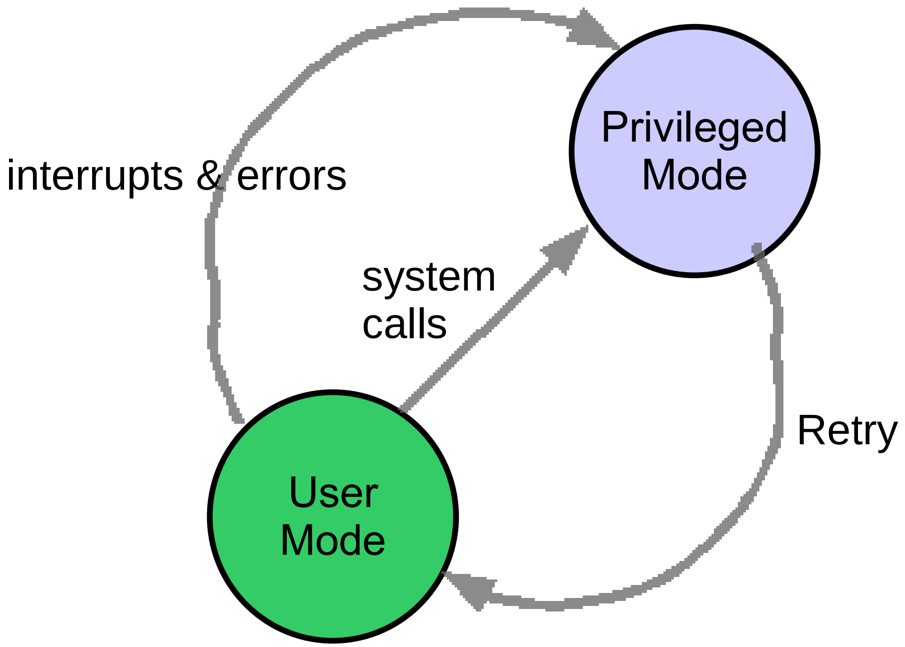
Virtual CPU

sun4v / API Slip-plane

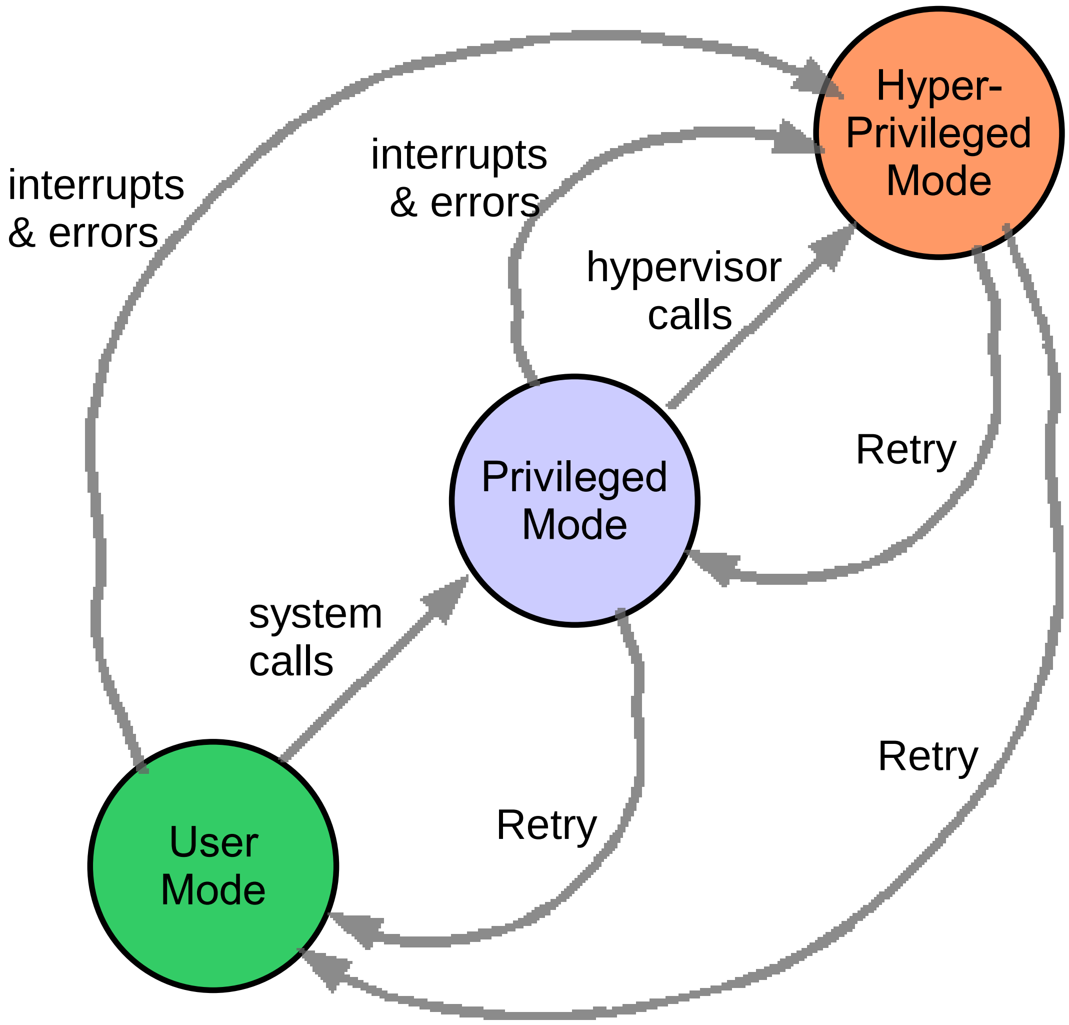
Physical CPU (strand)

Legacy SPARC execution mode

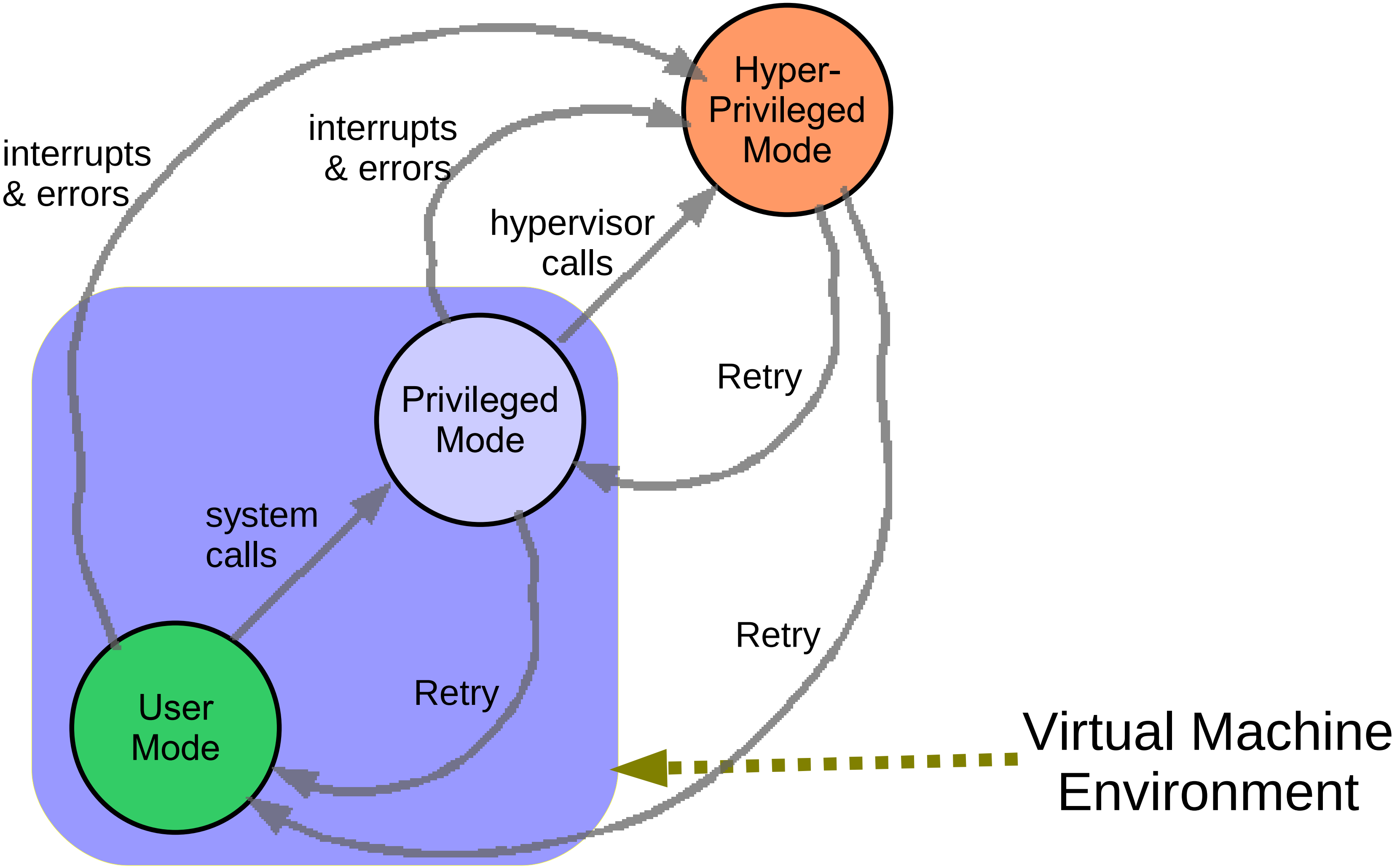
- Existing sun4u chips



New SPARC Execution mode



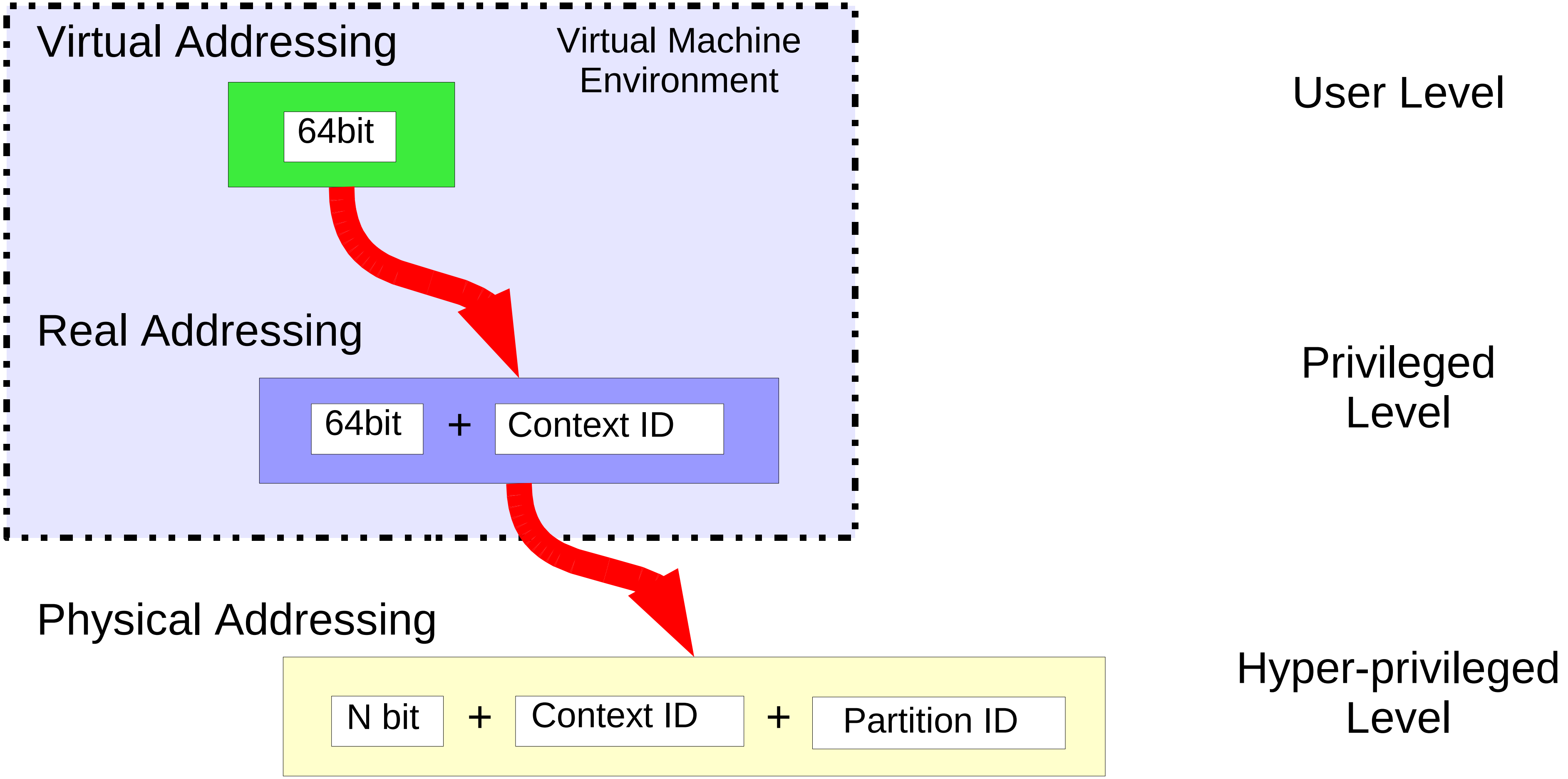
New SPARC Execution mode



Privileged mode constrained

- Close derivative of legacy privileged mode, but:
  - > No access to diagnostic registers
  - > No access to MMU control registers
  - > No access to interrupt control registers
  - > No access to I/O-MMU control registers
  - > All replaced by Hypervisor API calls
- UltraSPARC-ness remains with minor changes
  - > timer tick registers
  - > softint registers etc.
  - > trap-levels & global registers etc.
  - > register window spill/fill

Translation hierarchy



Translation management

- Guest OS defines a “fault-status” area of memory for each virtual CPU (vCPU)
  - Hypervisor fills in info for each MMU exception
- UltraSPARC does not use page-tables
  - traditionally a software loaded TLB
- Hypervisor APIs to support direct software management of TLB entries
  - map, demap
  - Simple guests like OBP can use this

Translation Storage Buffers

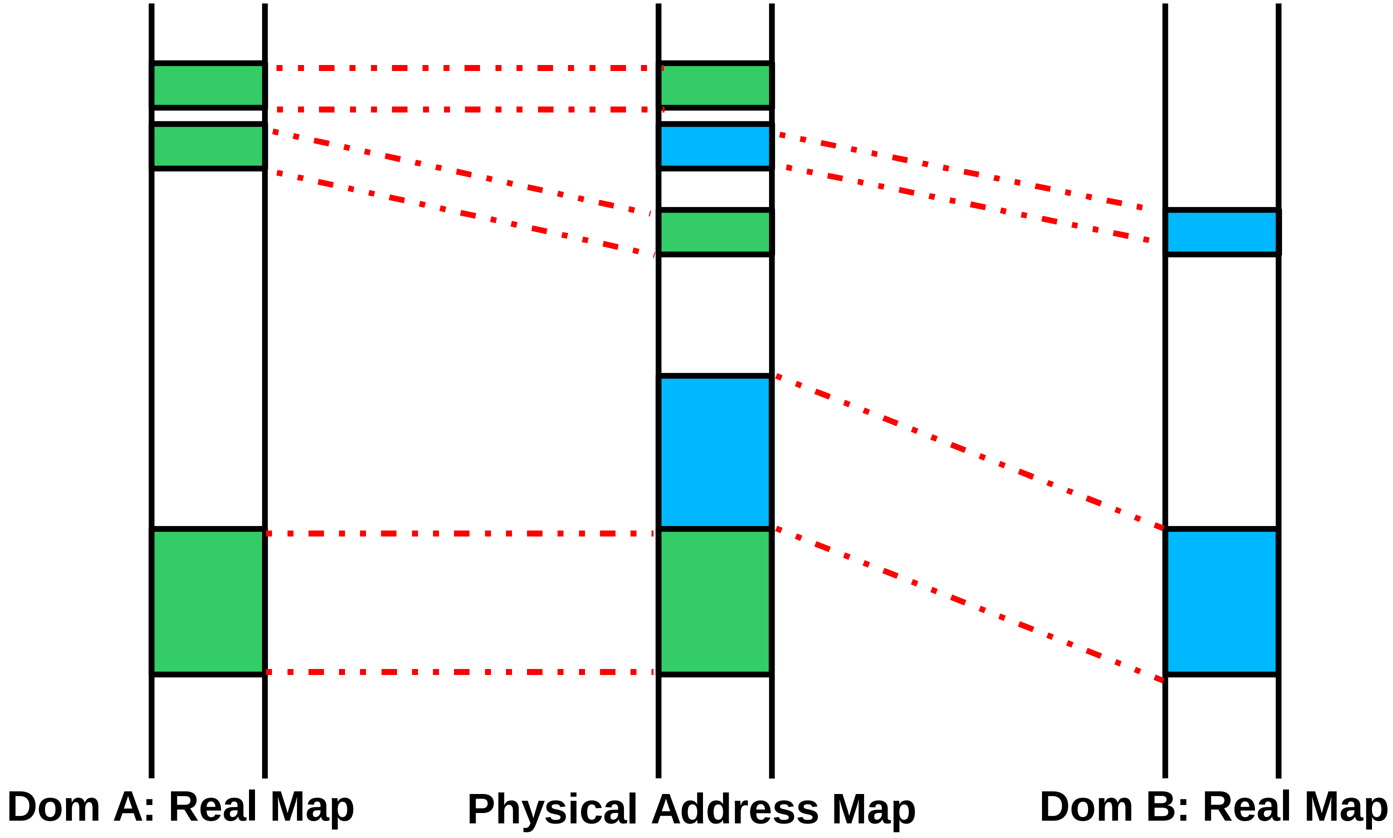
- Guest OS managed cache of translations stored in memory
  - Guest allocates memory for buffer
  - Guest places translation mappings into buffer when needed
  - Hypervisor fetches from this cache into TLBs

- Guest specifies virtual -> real mappings
  - Hypervisor translates real->physical to load into TLB
  - TLB holds virtual -> physical mappings

- Multiple TSBs used simultaneously for multiple page sizes and contexts

Address space control

- Hypervisor limits access to memory (and devices) -- creating partitions (logical domains)



Virtual I/O devices

- Provided via Hypervisor
  - e.g. Console - getchar / putchar API calls
  - Hypervisor generates virtual interrupts

Physical I/O devices

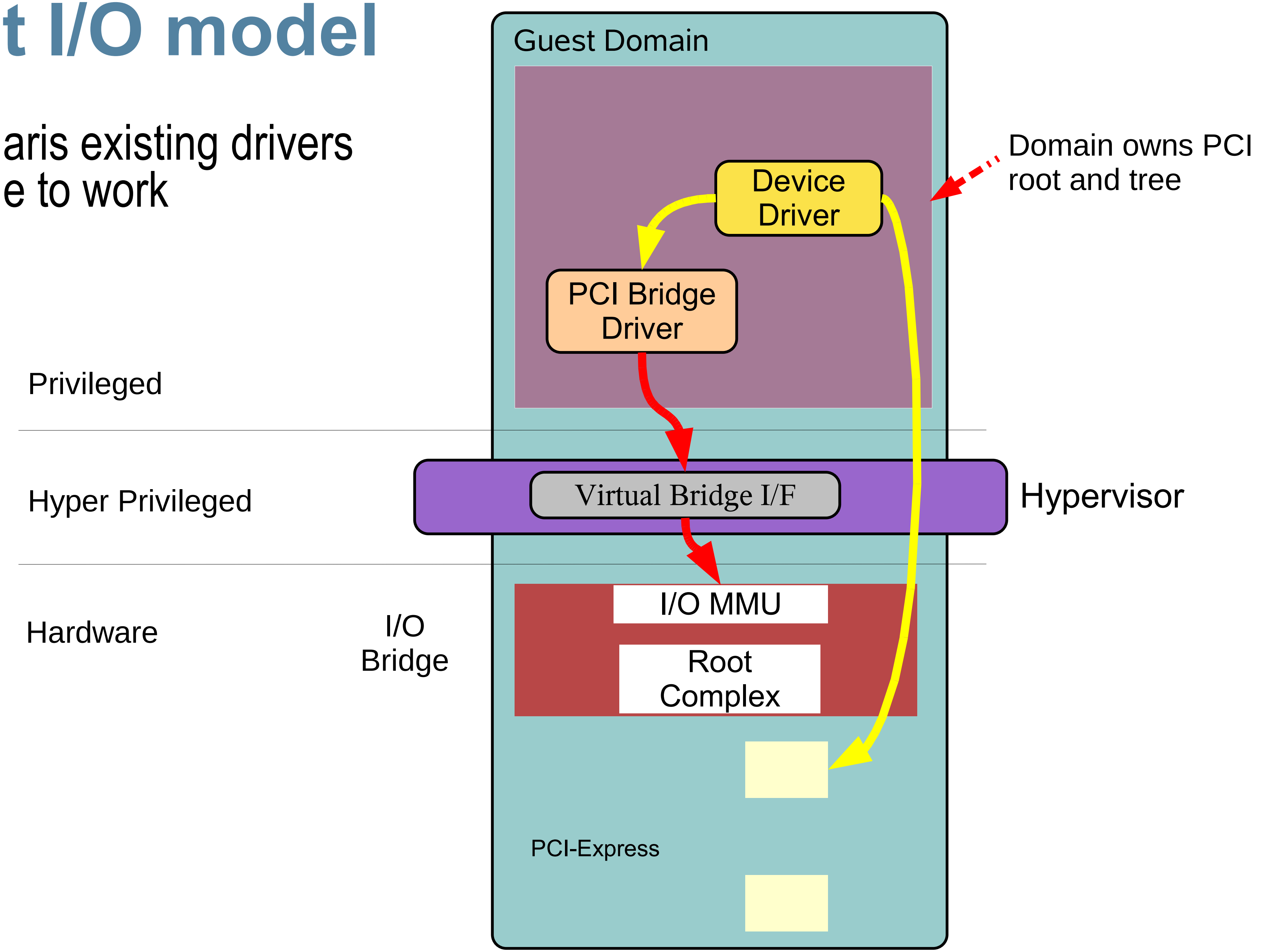
- PCI-Express root complex mapped into real address space of guest domain

- Direct access to device registers
  - OBP probes and configures bus and devices

- I/O Bridge and I/O MMU configuration virtualized by hypervisor APIs
  - Ensures that I/O MMU translations are validated by hypervisor
  - Device interrupts are virtualized for delivery

Direct I/O model

- For Solaris existing drivers continue to work



Interrupt and event delivery

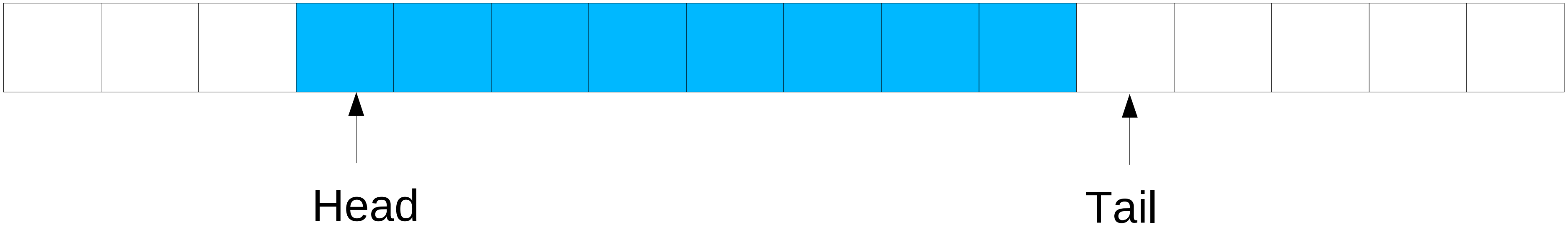
- Device interrupts and error events need a mechanism to asynchronously cause and exception for a virtual CPU

- Typically also require some data to identify reason and source and notification

- How to do this in an abstract manner?
  - What if the virtual CPU is not currently bound to a physical CPU?
  - Can't block physical interrupt source until virtual CPU is rescheduled

Interrupts & CPU mondos

- Delivered to privileged mode via in-memory FIFO queues
  - cpu-mondo, dev-mondo, resumable & non-resumable error queues
- 64-byte entries carry cause information (interrupt numbers)
- Head and Tail offsets available as CPU registers to privileged code
  - Tail manipulated by hypervisor, head by guest OS
  - For either queue, head != tail causes trap

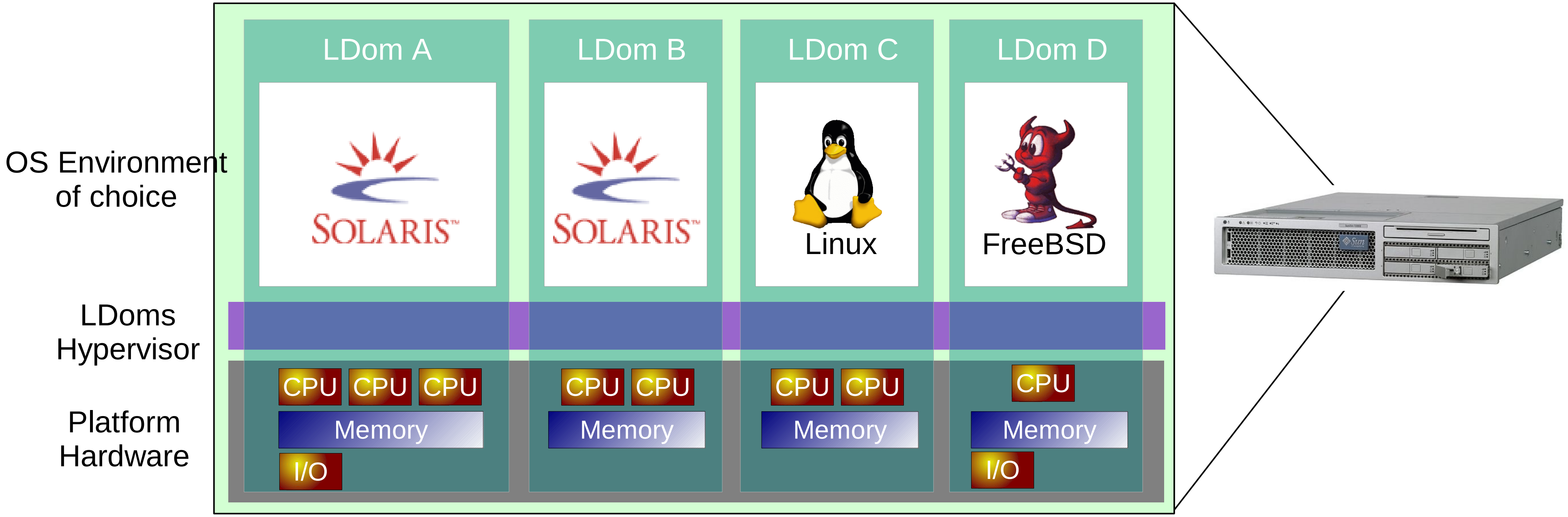


Queue constraints

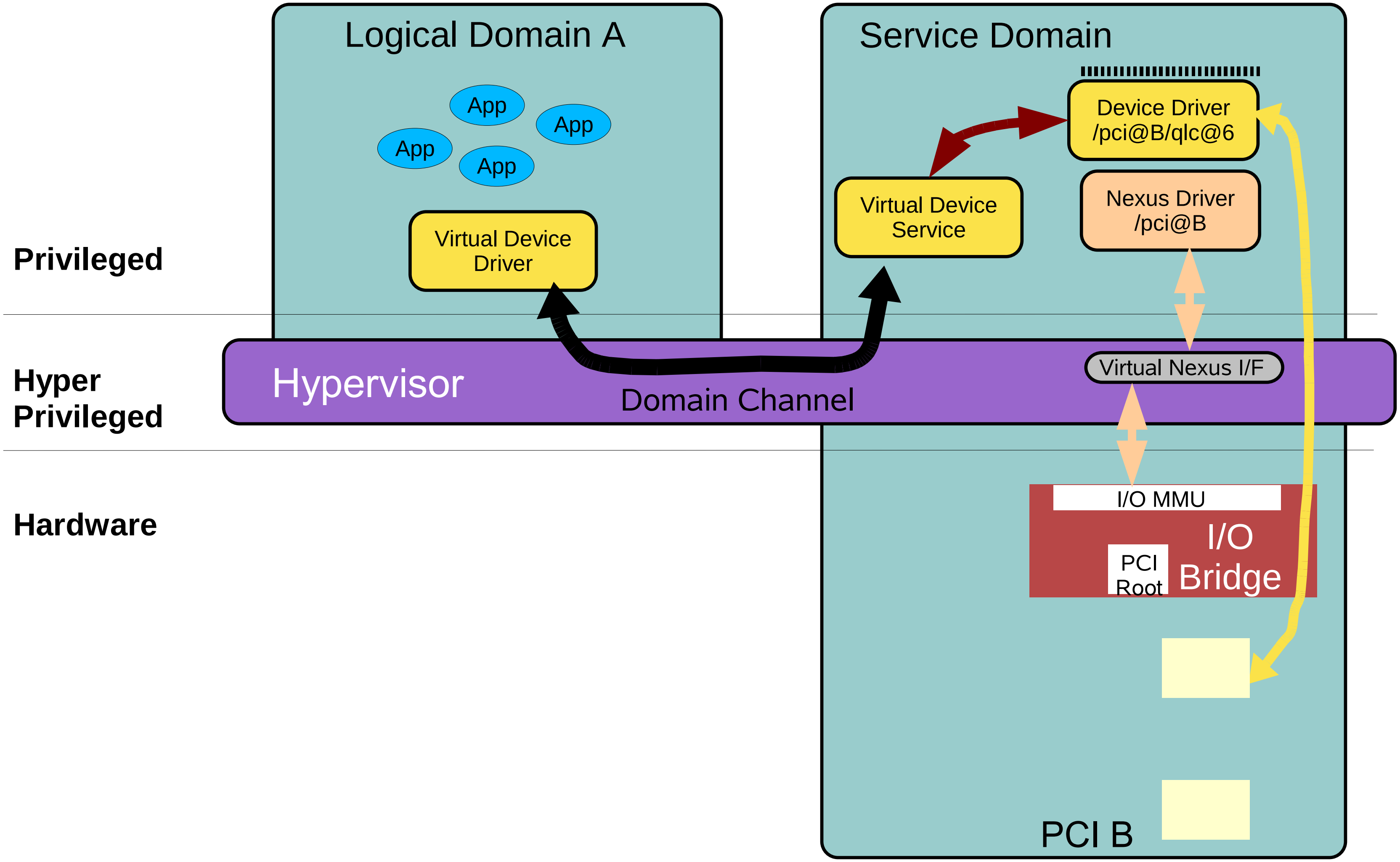
- Must be a power-of-2 number of entries, minimum of 2
  - Entries always 64 bytes in size
  - (tail+1)%size == head defines full state
- Must be aligned on a real address boundary identical to Q size
  - Designed to make hardware mondo delivery easier
- May have queues defined for each virtual CPU
  - dev_mondo queue must be sized for all possible interrupt sources
  - dev_mondo queue may never contain more than one entry for same source
- Hypervisor API to send 64Byte mondo to CPU queue
  - Used for CPU to CPU x-calls
  - Queue may fill and sender's API call fails

Logical Domaining Technology

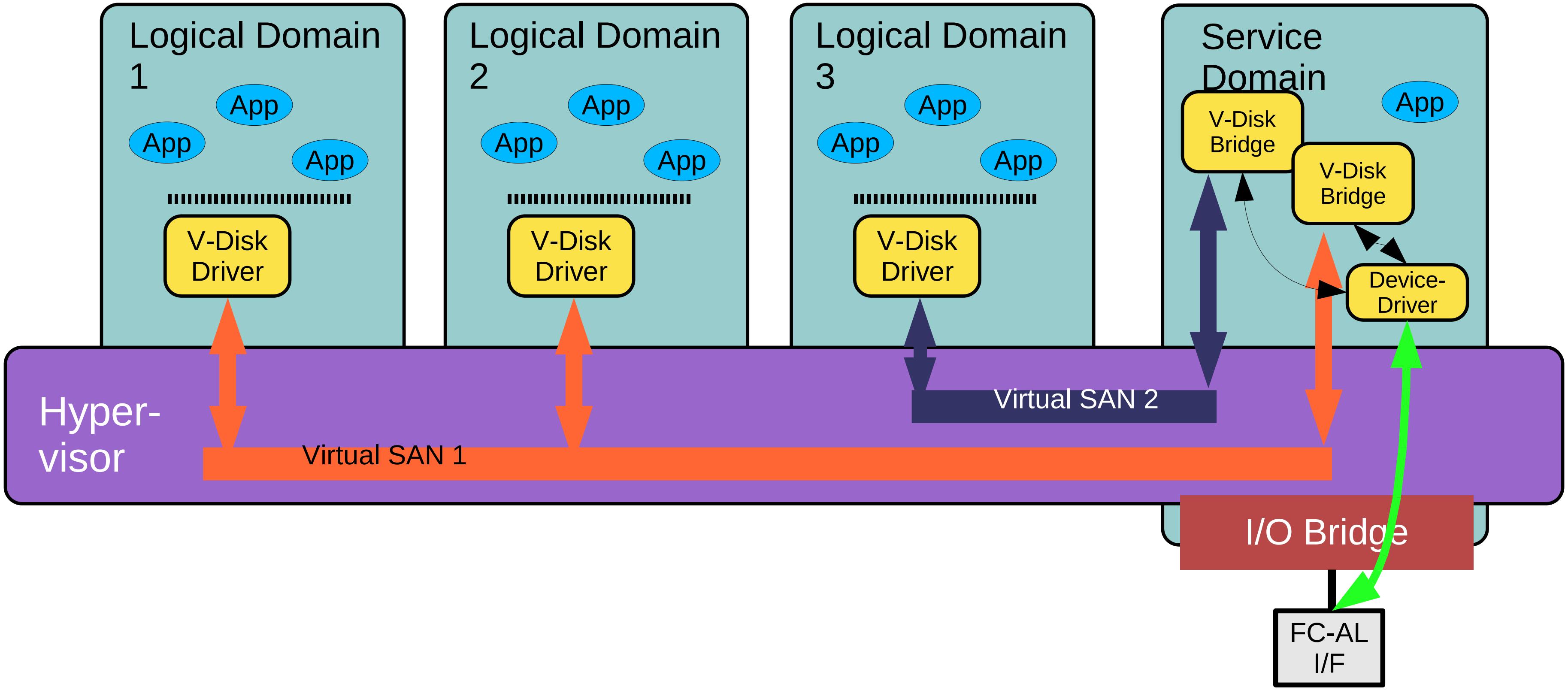
- Virtualization and partitioning of resources
  - > Each domain is a full virtual machine, with a dynamically re-configurable sub-set of machine resources, and its own independing OS instance
  - > Protection & isolation via SPARC hardware and Ldoms firmware



Virtualized I/O

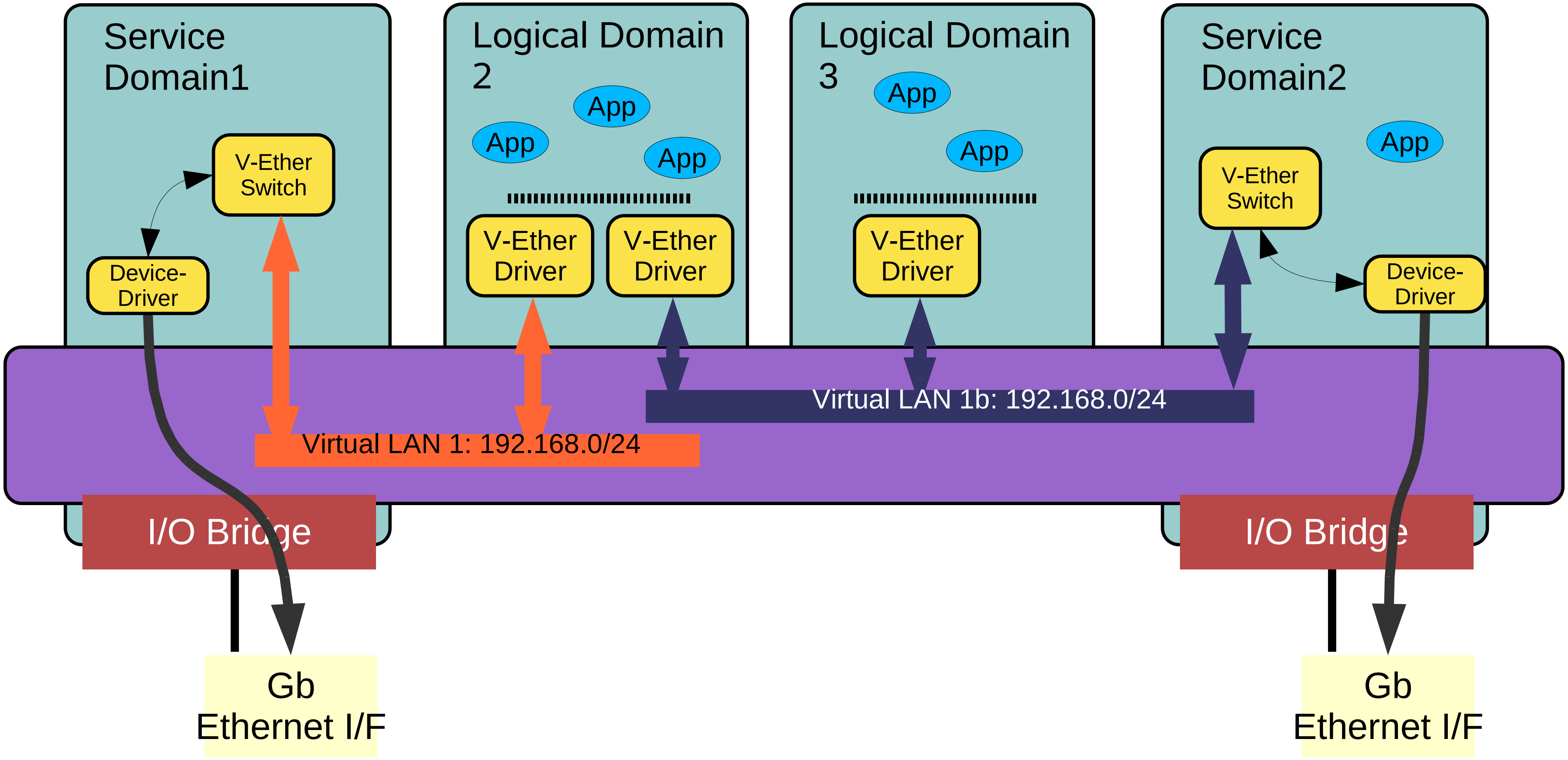


Virtual (Block) Disk device & server



Redundancy; Multi-path virtual I/O

- Virtualised devices can be used for redundant fail-over if guest OS supports it



Domain Manager

- One manager per host HV
  - Application that controls Hypervisor and its LDom

- Exposes external CLI & XML control interfaces

- Maps Domains to physical resources
  - Constraint engine
  - Heuristic binding of LDoms to resources
    - Assists with performance optimisation
    - Assists with handling failures and blacklisting

Dynamic Reconfiguration

- Hypervisor has ability to dynamically shrink or grow LDoms upon demand
- Simply add/remove cpus, memory & I/O
  - Ability to cope with this without rebooting depends on guest OS capabilities
  - Guest OS indicates its capabilities to the domain manager
- Opportunity to improve utilisation by balancing resources between domains

Summary

- Specifications & code published:
  - http://www.opensparc.net
  - http://www.opensolaris.net
- “Legion” instruction level simulator available to assist with code development
  - Provides level of code execution visibility not possible on actual hardware
  - Source code available on http://www.opensparc.net
- Contact alias:
  - hypervisor@sun.com

Additional Topics

- Error handling
- Machine Descriptions
- Boot process

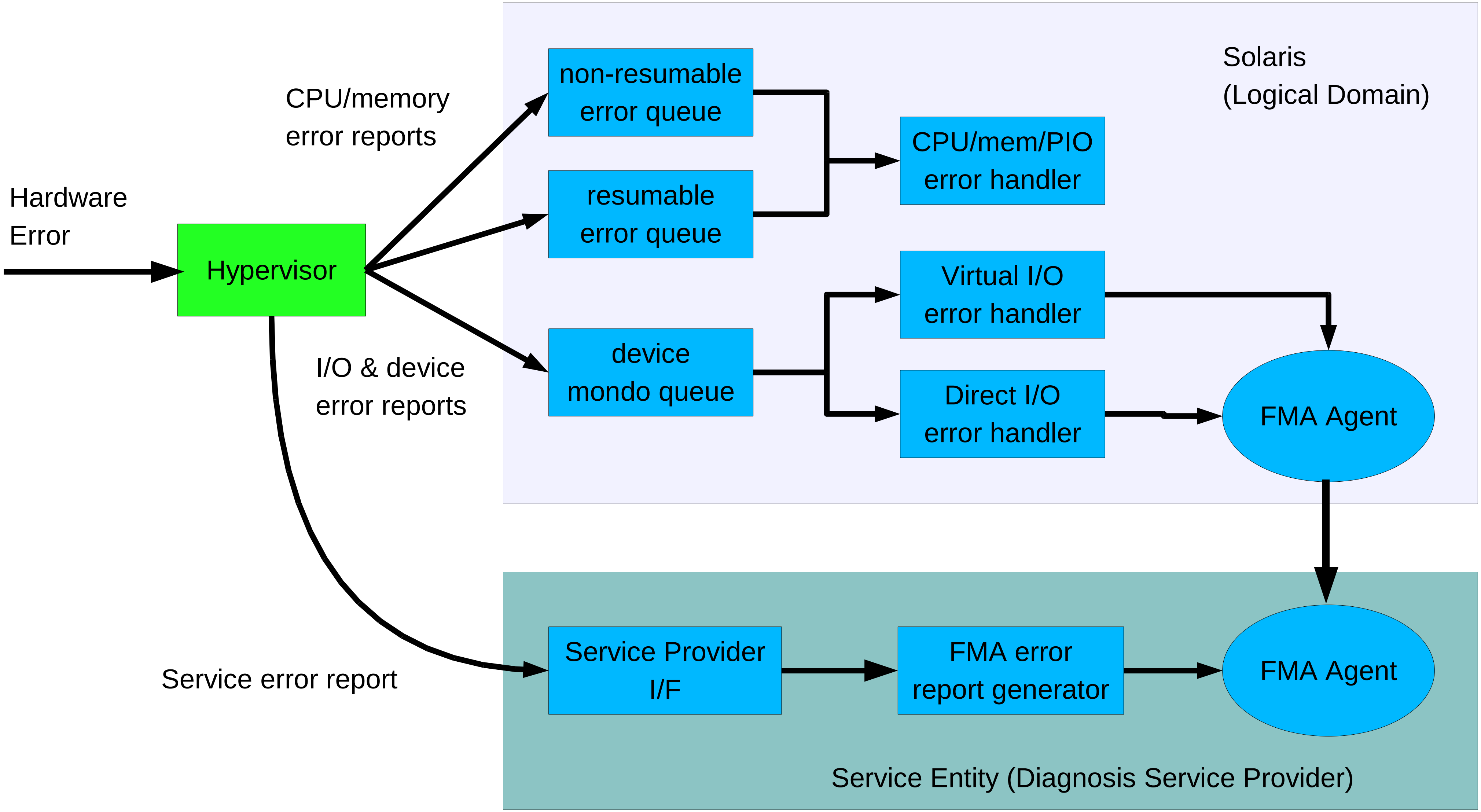
Error delivery philosophy

- Hypervisor handles and abstracts underlying hardware errors
- All errors logged on service processor
- Guest OSs are told about impact of error
  - No point in informing guest about correctable errors
- Legacy OS should be able to run on new platform

Error handling

- Two simple classificiations; “after handling the error, can I ...”
  - 1. Resume execution of what I was doing, or
  - 2. Can't resume execution ... some policy to handle this
- Simplest error handlers:
  - 1. Retry
  - 2. Panic
- 2 in-memory queues associated with these types, similar to interrupts
- Queue entries contain error reports distilled by hypervisor
- Hypervisor creates reports and attempts to correct errors when possible

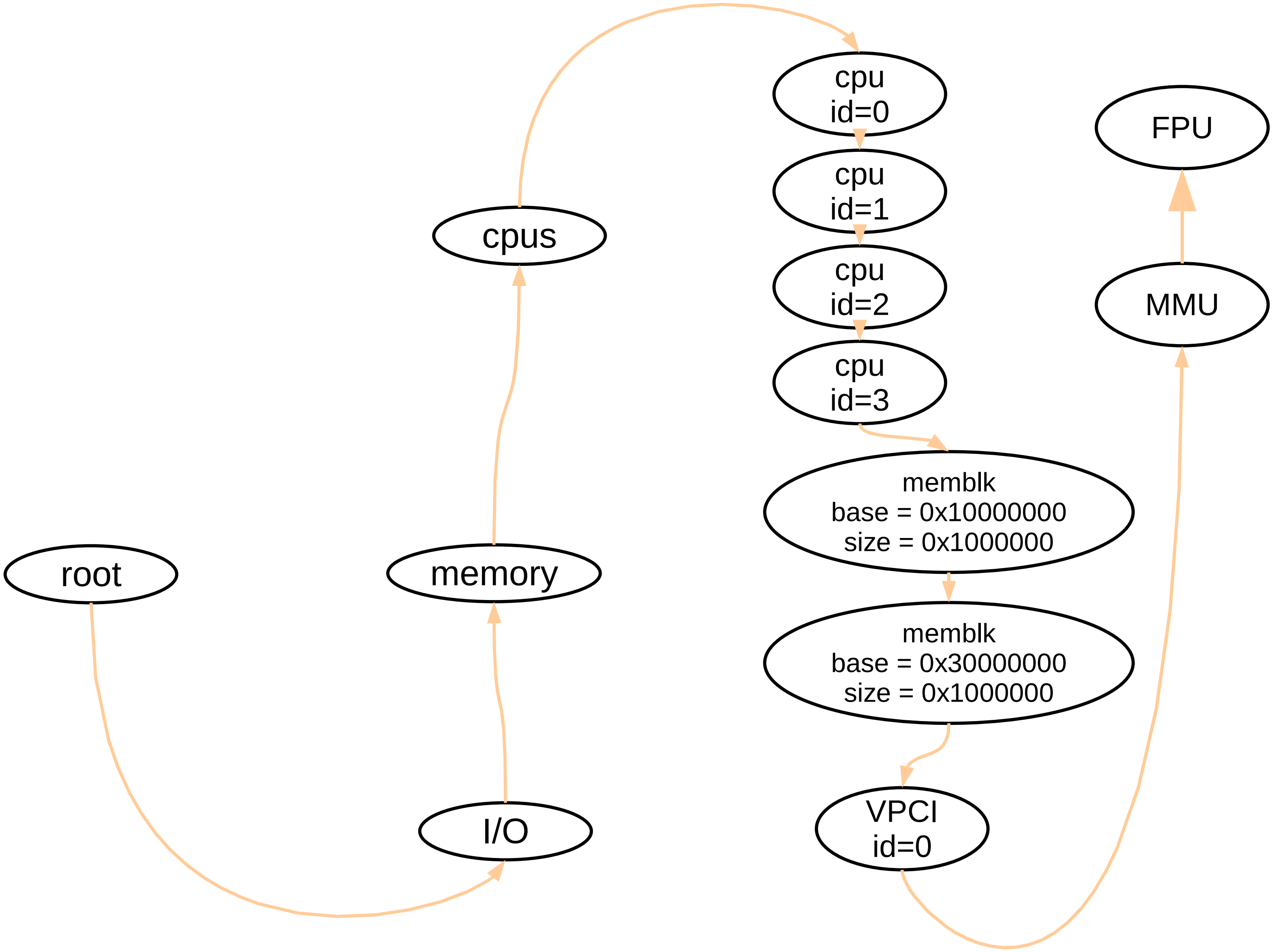
Error handling agents



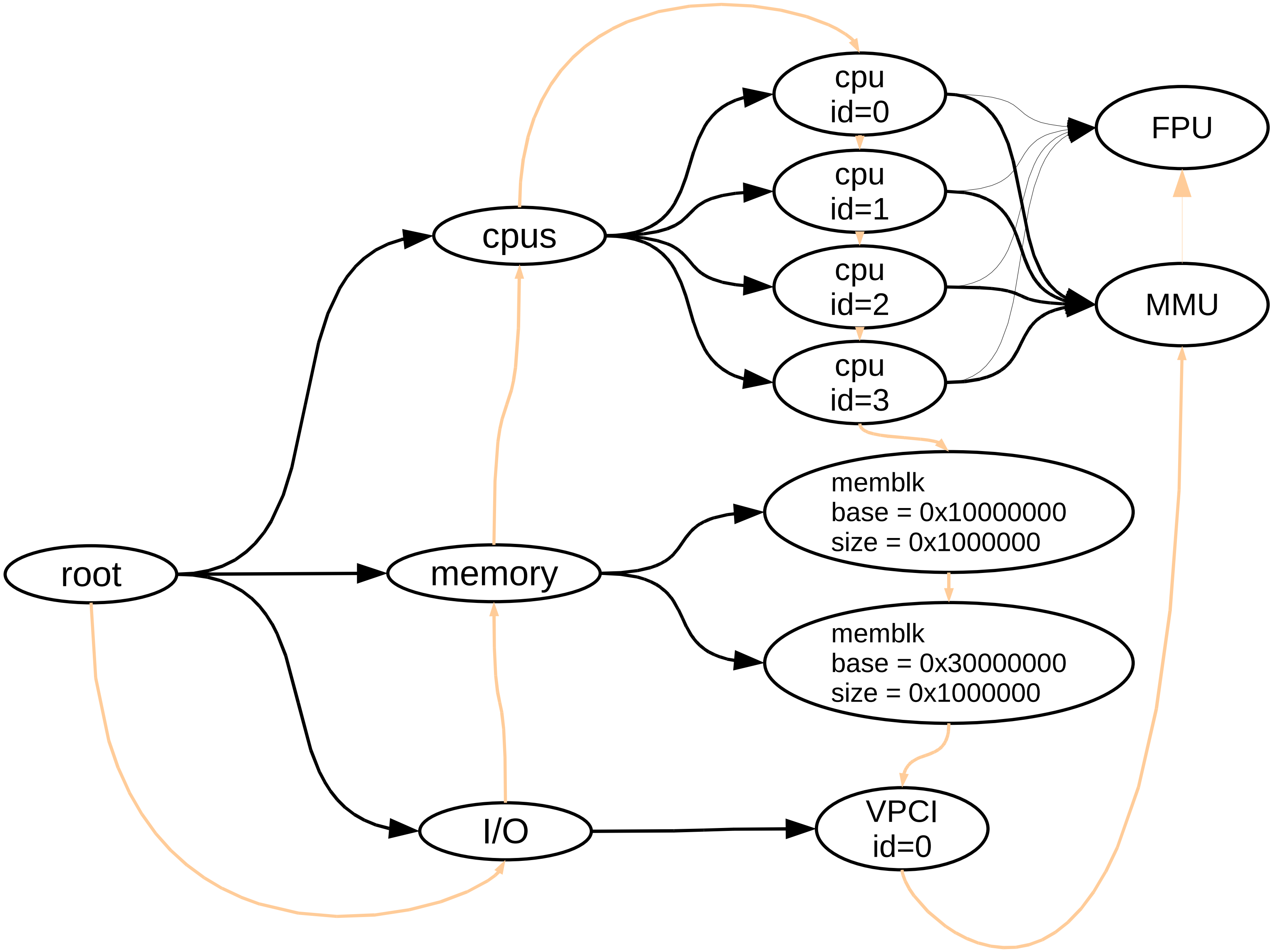
Machine description

- How the OS Inside a virtual machine finds its resources
- Trivial list of nodes that detail the contents of each domain
  - CPUs, Blocks of memory, I/O devices, I/O Ports etc.
- Nodes are also inter-linked to form a DAG to convey more advanced information for guest OSs that care
  - e.g. cache sharing, NUMA memory latencies etc.
- Key requirements;
  - Very simple to parse by the simplest of guests
  - Convey very complex information for guest OSs that care
  - A guest need not understand all the information presented
    - e.g. old OS running on a new platform

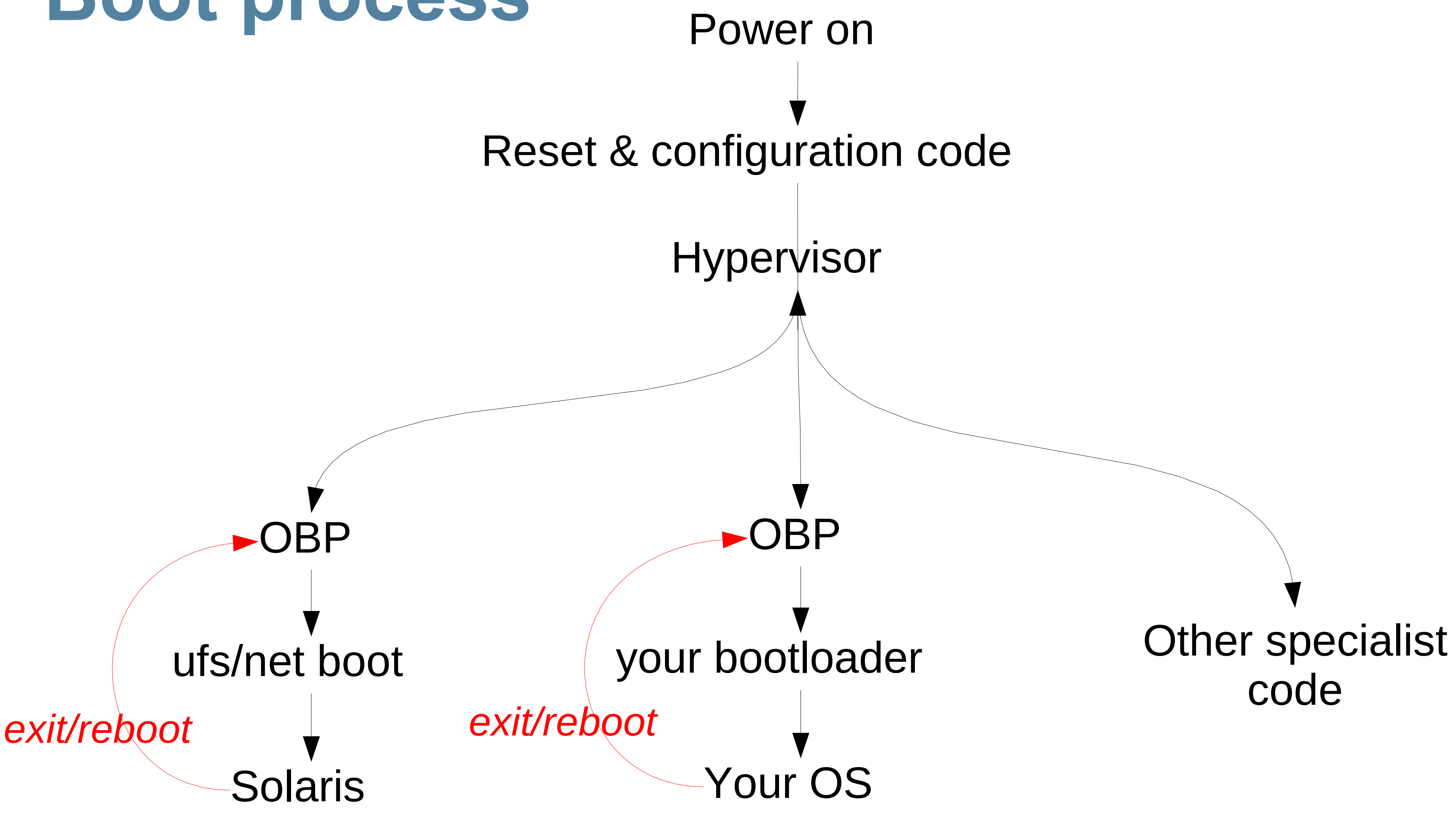
Simple list of nodes



Also arranged as a DAG



Boot process



Power on

Reset & configuration code

Hypervisor

OBP

ufs/net boot

Solaris

exit/reboot

OBP

your bootloader

Your OS

exit/reboot

Other specialist code

開放的
열린
مفتوح
libre
मुक्त
ಮುಕ್ತ
livre
libero
ముక్త
开放的
açık
open
nyílt
⠕⠏⠑⠝
פתוח
オープン
livre
ανοικτό
offen
otevřený
öppen
открытый
வெளிப்படை

open

64 bit, 32 threads, free

OpenSPARC T1 Tutorial

Compiler Optimizations

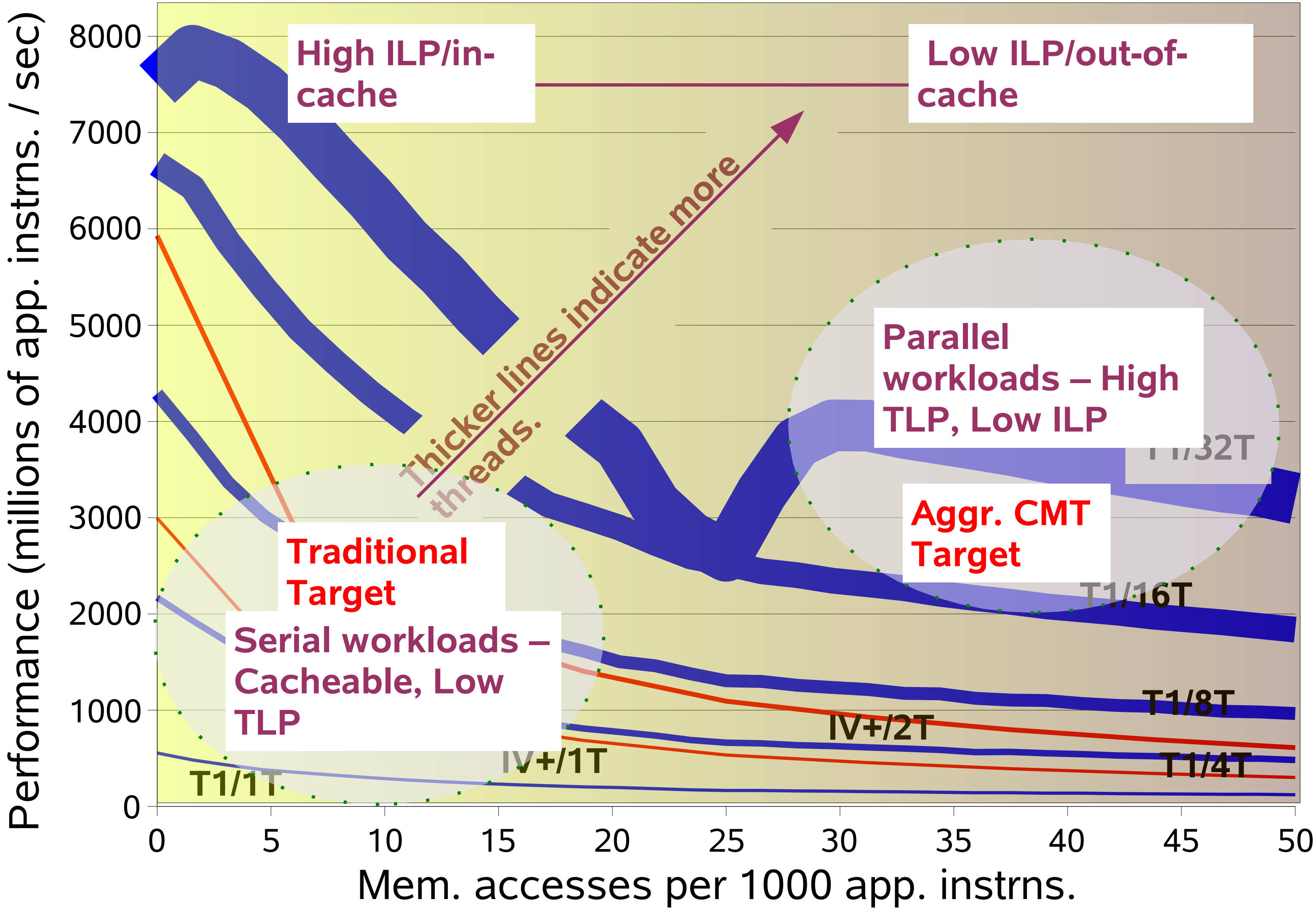
Jhy-Chun (JC) Wang
Senior Staff Engineer

jhy-chun.wang@sun.com

Agenda

- Background and Context
- Traditional (serial) optimization
- Parallelization
  - Automatic, OpenMP, Libraries ...
- Analysis
- Embracing gcc users
- Summary

Traditional vs. Aggressive CMT



High ILP/in-cache

Low ILP/out-of-cache

Thicker lines indicate more threads.

Parallel workloads – High TLP, Low ILP

Aggr. CMT Target

Traditional Target

Serial workloads – Cacheable, Low TLP

Designing for ILP vs. TLP

- Want to build a CPU with ~10BIPS capability?

- Option A
  - Build a superscalar dual-core design
  - Run the chip at 2.5GHz
  - Look for 1-2 threads with an IPC of 4-2@2.5GHz

Rare in most codes

- Option B
  - Build a 1-issue 8 core, 32-thread design
  - Run the chip at 1.25GHz
  - Look for 8-32 threads with and IPC of 1-0.25@1.25GHz

Much easier to find

Synergies Within a System

- Hardware
  - > Adequate cache/memory, I/O, and networking bandwidth, plus RAS for large, parallel workloads
- Operating System
  - > Reliable and scalable OS for optimal management of parallel threads
- Developer Tools
  - > Compilers and tools to make application development easy and efficient

Focus of this section:
C/C++/Fortran Compilers & Tools
Free download: cooltools.sunsource.net

Basic Optimization

- An easy (naïve) start:
  ```
  $ cc foo.c
  ```
  - No optimization (or very limited optimization)

Basic Optimization

- An easy (naïve) start:

  -
    ```
    $ cc foo.c
    ```
    - No optimization (or very limited optimization)
- A little better

  -
    ```
    $ cc -O foo.c
    ```
    - Optimization turned on at default level

Optimization Bag

Comm. Sub. Elim.
Dead Code Elim.
Loop Transformations
Instruction scheduling
Register allocation
Invariant hoisting
Peephole
....

Basic Optimization

- An easy (naïve) start:
  - `$ cc foo.c`
    - No optimization (or very limited optimization)
- A little better
  - `$ cc -O foo.c`
    - Optimization turned on at default level
- Even better
  - `$ cc -xO4 foo.c`
    - Optimization turned on at a high level
- What next?

Optimization Bag

Comm. Sub. Elim.
Dead Code Elim.
Loop Transformations
Instruction scheduling
Register allocation
Invariant hoisting
Peephole
....

Guiding/Controlling Optimizations

- Numerous advanced optimizations in the compiler
- Controls exist to leverage/guide most optimizations
  - > Inlining, inter-procedural analysis, profile feedback, alias analysis, target system selection, prefetching, pragmas/directives ....
- Significant benefits can be obtained by carefully selecting and tuning available optimizations

```
$ cc -xO4 -xinline=foo,no%bar -xprefetch_level=3 \
        -xchip=ultraT1 program.c
```

Besides -O4, suggests that routine foo() be inlined and bar() not be inlined in program.c, turns on aggressive prefetching, and targets the T1 chip.

An All in One Flag?

- The -fast option includes various flags designed for a fairly aggressive build
  - Easy to start with -fast and add options after it to modify the behavior as desired
  - Fragment from makefile might look like this:

```
ISA        = -xarch=v8plus
CFLAGS     = -fast $(ISA) ....
LDFLAGS    = -fast $(ISA) ....
```

Think “-fast”!

What did it do to my code?

- The compiler commentary explains how the source code was optimized
  - Build with “-g” added (does not disable optimizations)
  - Get commentary with er_src command
  - See documentation for details

- Improves understanding and helps user optimize
  - User can derive hints on further options to use (or not use)
  - User can derive hints on adding pragmas that might help
  - User can derive hints on what reorganization might help

Example 1 – Loop Scheduling

```
for (j=1; j<n; j++)
  a[j] = a[j-1] + 4.0*b[j]*c[j] +
         b[j]*b[j] + c[j]*c[j] + 6.0;
```

```
cc -fast -xchip=ultra4 -g -c loop.c
er_src -source foo 1 loop.o
```

```
L-tag L1 scheduled with steady-state cycle count = 4
L-tag L1 unrolled 4 times
L-tag L1 has 2 loads, 1 stores, 3 prefetches, \
  4 FPadds, 3 FPmuls, and 0 FPdivs per iteration
L-tag L1 has 0 int-loads, 0 int-stores, 5 alu-ops, \
  0 muls, 0 int-divs and 0 shifts per iteration
Source loop below has tag L1
     7.   for (j=1; j<n; j++)
     8.     a[j] = a[j-1] + 4.0*b[j]*c[j] +
                   b[j]*b[j] + c[j]*c[j] + 6.0;
```

- 1 load eliminated
- 1 fpmul eliminated
- unrolled 4 times
- optimally scheduled
  - resource limit = 4
  - dependence limit = 4
  - achieved schedule = 4
- 3 prefetches inserted

Example 2 – IPO, Pointers, IF's

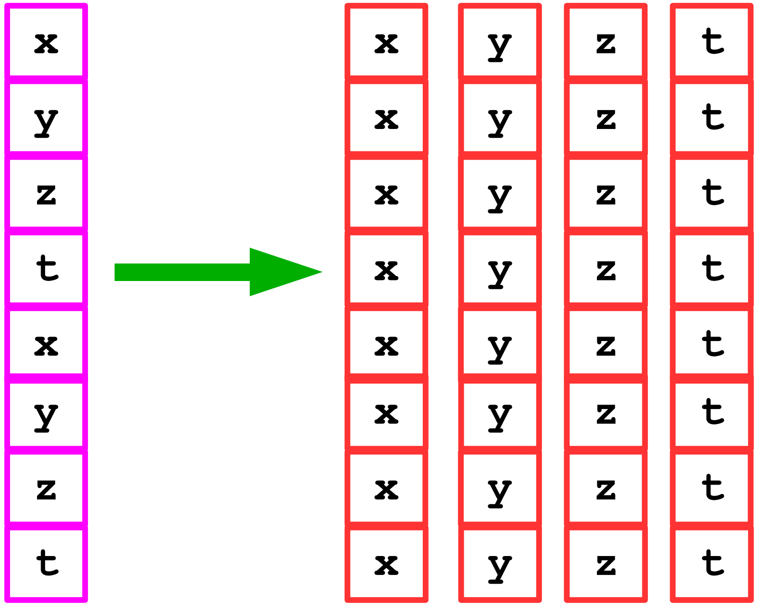
```
void propagate(int *p, int *q, int *r) {
  int x;
...

  x = *p;
...
  if (x < 50) {
    ...r1...
    split(p,q);
    ...r2...
    if (x < 100) {
      merge(q,r);
    }
  }
...
```

- Note `x` is a local variable
- If it can be proved that:
  - cond1 => cond2
  - `x` is not modifiable in split
  - `x` is not modified in r1
  - `x` is not modified in r2
- then:
  - the second `if` is eliminated
- Involves
  - Pointer analysis
  - Inter-procedural analysis
  - Conditional relationships

- **Complexity and “code rot” can cause such scenarios**
- **Second conditional optimized away by the compiler**

Example 3 – Whole Program Mode

```
setup(p);
for (i=0; i<STEPS; i++) {
    transform_x(p);
    transform_y(p);
    transform_z(p);
    transform_t(p);
}
report(p);
```



- Original source has 32 byte struct
- Program malloc's for large vector
- All hot segments touch one field
- Ends up with poor cache behavior
  - 32 byte stride, 25% utilization
- With whole program analysis:
  - Compiler splits the vector
  - Generates four vectors
  - Hot segments get 8 byte stride
  - 100% cache block utilization
  - Performance is improved

Parallelization: Automatic

- Compiler does the parallelization *automatically*
  - > Just use the -xautopar option
  - > No other user action required
- Automatic parallelization targets loop nests
  - > Works synergistically with loop transformations
  - > Steadily improving - handles many complex cases now
- Thread count controlled by environment variable
- Two versions generated (if profitability cannot be statically determined)
  - > Run time selection between serial and parallel versions
  - > Serial version used if work/thread is too low

Example – Autopar + Multiple Transforms

```
...
first(m,n);
second(m,n);
...
```

```
for (j=0; j<n; j++)
  for (i=0; i<m; i++)
    a[i][j] = b[i][j] + c[i][j];
```

```
for (i=1; i<m-1; i++)
  for (j=1; j<n-1; j++)
    b[i][j] = 0.5*c[i][j];
```

- Top level routine has two calls

- Loop in first()

- Loop in second()

- Routines inlined
- Loop nest in first() interchanged
- Loop nest in first() peeled
- Fused with loop nest in second()
- Loops parallelized at outer level
- Inner loop pipelined
- Prefetches inserted

➔ Difficult to understand final code

➔ Use commentary, other options

Parallelization: OpenMP

- It is an industry standard (www.openmp.org)
  - Supported by a large number of compilers
  - OpenMP code is portable
  - Directives can be ignored for serial/unsupported systems
- Requires little programming effort
  - Can start with just a handful of directives
  - Applications can be parallelized incrementally
- Good performance and scalability possible
  - Depends ultimately on the code, compiler, and system
  - CMT-friendly shared-memory parallelism leveraged

Example 1 - Loop

```
for (i=0; i<n; i++)
  a[index[i]] = a[i] + 2;
```

- -xautopar used
- Not parallelized
  - Unsafe

```
$ cc -xO4 -xautopar -xloopinfo loop.c
"loop.c", line 8: not parallelized, unsafe dependence (a)
```

```
#pragma omp parallel for shared(n,a) private(i)
for (i=0; i<n; i++)
  a[index[i]] = a[i] + 2;
```

- Pragma added
- -xopenmp used
- Parallelized

```
$ cc -xO4 -xopenmp -xloopinfo loop.c
"loop.c", line 9: PARALLELIZED, user pragma used
$ OMP_NUM_THREADS=4    // Controls number of threads
$ a.out
```

Example 2 - Sections

```
#pragma omp sections
    {
#pragma omp section
      foo1();
#pragma omp section
      foo2();
#pragma omp section
      foo3();
#pragma omp section
      foo4();
    }
```

- It is not just for loops
- Arbitrary pieces easily parallelized
- Must be legal, of course
- In this example:
  - foo1() - foo4() run in parallel

Value-Added Features

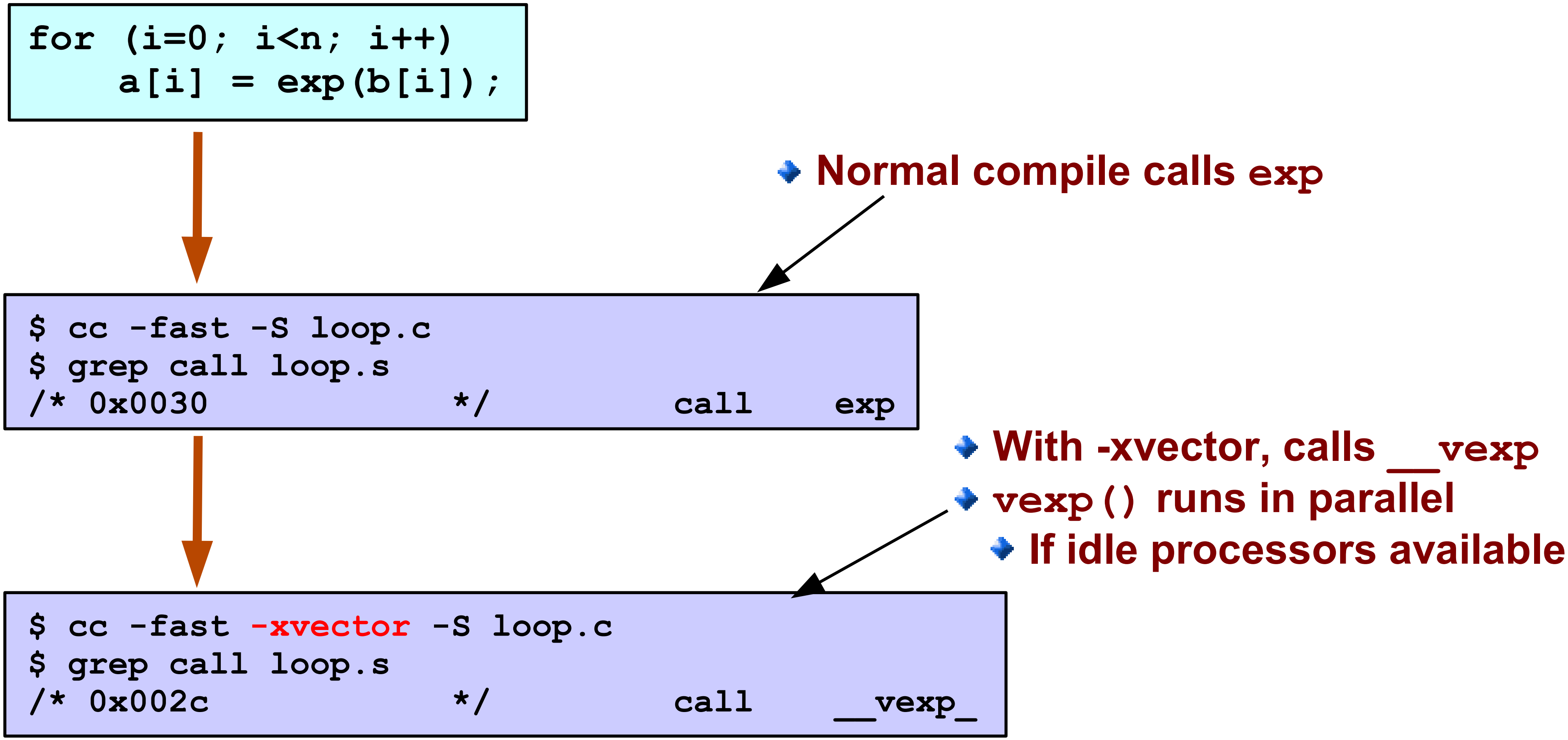
- OpenMP version 2.5 fully supported
  - Includes support for nested parallelism
- Performance tuned for OpenSPARC systems
- Idle thread behavior can be controlled
- Static and runtime error checking
- OpenMP debugging using dbx
- OpenMP performance profiling
- Autoscoping
  - The compiler can assist the user with scoping the variables

C Autoscoping Example

```
$ cc -fast -g -c -xopenmp -xloopinfo -xvpara loop.c
$ er_src -cc parallel -src loop.c loop.o
...
...
   Source OpenMP region below has tag R1
   Variables autoscoped as SHARED in R1: b, c, n, a
   Private variables in R1: i
   Shared variables in R1: a, b, c, n
     8. #pragma omp parallel for default(__auto)

   L1 parallelized by explicit user directive
     9.    for (i=0; i<n; i++)
    10.      a[i] = a[i] + 2*b[i] + c[i];
    11. }
```

- Compiler commentary lists the autoscoping done

Parallelization: Math Library



```
for (i=0; i<n; i++)
     a[i] = exp(b[i]);
```

```
$ cc -fast -S loop.c
$ grep call loop.s
/* 0x0030                 */          call     exp
```

- Normal compile calls `exp`

```
$ cc -fast -xvector -S loop.c
$ grep call loop.s
/* 0x002c                 */          call     __vexp_
```

- With -xvector, calls `__vexp`
- `vexp()` runs in parallel
  - If idle processors available

Loops may be split to enable calling vector functions.

Parallelization: Performance Library

```
do jj = 1,n,nb
   call zip(b(1,jj),%val(nb))
   do ii = 1,n,na
      call zip(a(1,ii),%val(na))
      do jjj = jj,jj+nb-1,3
         do iii = ii,ii+na-1,3
             call getts()
            call foo(a,b,c,iii,jjj)
             call gette()
         end do
      end do
       call gettp()
   end do
end do
```

+ There's more code behind!

```
call dgemm('T','N',n,n,n,1.d0,a,num,b,num,0.d0,d,num)
```

```
$ f90 -fast main.f \
  hrtime.o -lsunperf
$ a.out
 Mflops:  2640
$ PARALLEL=8
$ a.out
 Mflops:  19736
```

- No coding sweat
- No debugging pains
- No tuning headaches
- Great performance!
- Portable!
- And parallel!

Parallelization: Media/Graphics Library

```
for (n = 0; n < dlen; n ++) {
  tmp = 0;
  for (k = 0; k < flen; k ++)
    tmp += fir[k] * src[n+k];
  dst[n] = (vis_s16) (tmp >> 16);
}
```

```
vis_write_gsr(0);
da = (vis_u8 *) dst;
dp = (vis_d64 *) ((vis_u32) da & (~7));
off = (vis_u32) dp - (vis_u32) da;
dend = da + 2 * dlen - 1;
emask = vis_edge16(da, dend);
sa = (vis_u8 *) src;
num = ((vis_u32)dend>>3) - ((vis_u32)da>>3) + 1;
for (n = 0; n < num; n ++) {
  ss = sa;
  rdh = vis_fzero(); rdl = vis_fzero();
  for (k = 0; k < flen; k ++) {
    sp = (vis_d64 *) vis_alignaddr(ss, off);
    s0 = sp[0]; s1 = sp[1];
    sd = vis_faligndata(s0,
    ...
```

+ There's more code!

- No need to write VIS
- No need to write MMX
- Just use mediaLib functions
- Perf. gain of ~6X (avg.)
- “C” version exists

  (can run on any system)

Performance Analysis

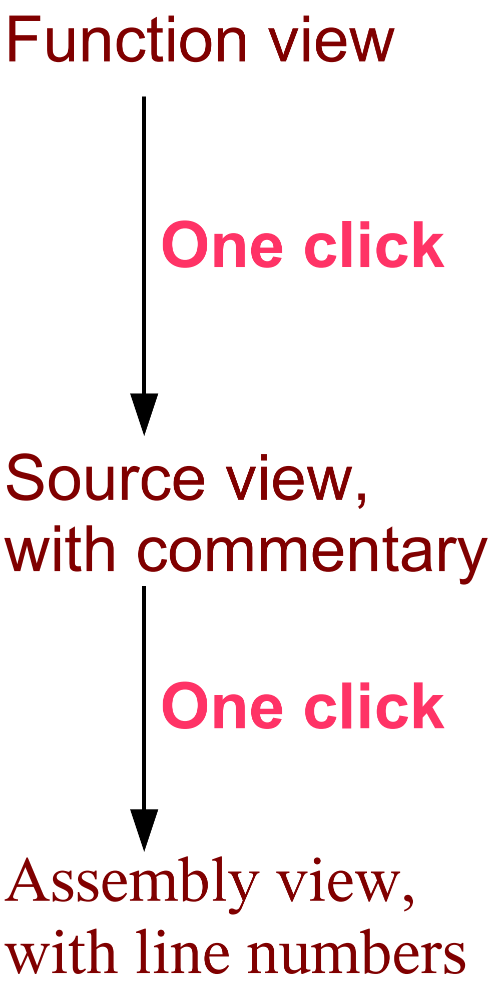
- Analyzer – an advanced performance analysis tool
- Intuitive GUI interface
- Clock based statistical profiling
- HW counter based statistical profiling
- Can relate data to function, source, assembly level
- Integrated with compiler commentary
- Dataspace and memoryspace profiling
- Enhanced OpenMP support

Analysis Example: Memory Bottleneck

```
Excl.       Incl.       Excl.     Incl.      Name
Instr_cnt   Instr_cnt   L3_miss   L3_miss
Events      Events      Events    Events
2452671487  2452671487  96803274  96803274   <Total>
2327052822  2327052822  93803181  93803181   loop
 125618665  2452671487   3000093  96803274   main
         0  2452671487         0  96803274   _start
```

```
...compiler commentary here...
            0           0         0         0
  11.   for (i=0; i<n; i++)
## 2327052822  2327052822  93803181  93803181
  12.     t += a[index[i]];
```

```
      9200436     9200436         0         0
  [13]    10908:  ldd         [%o0 + %l0], %f4
            0           0         0         0
  [13]    1090c:  faddd       %f2, %f10, %f14
## 1154977323  1154977323  93803181  93803181
  [13]    10910:  sll         %g5, 3, %g4
```



Function view

One click

Source view, with commentary

One click

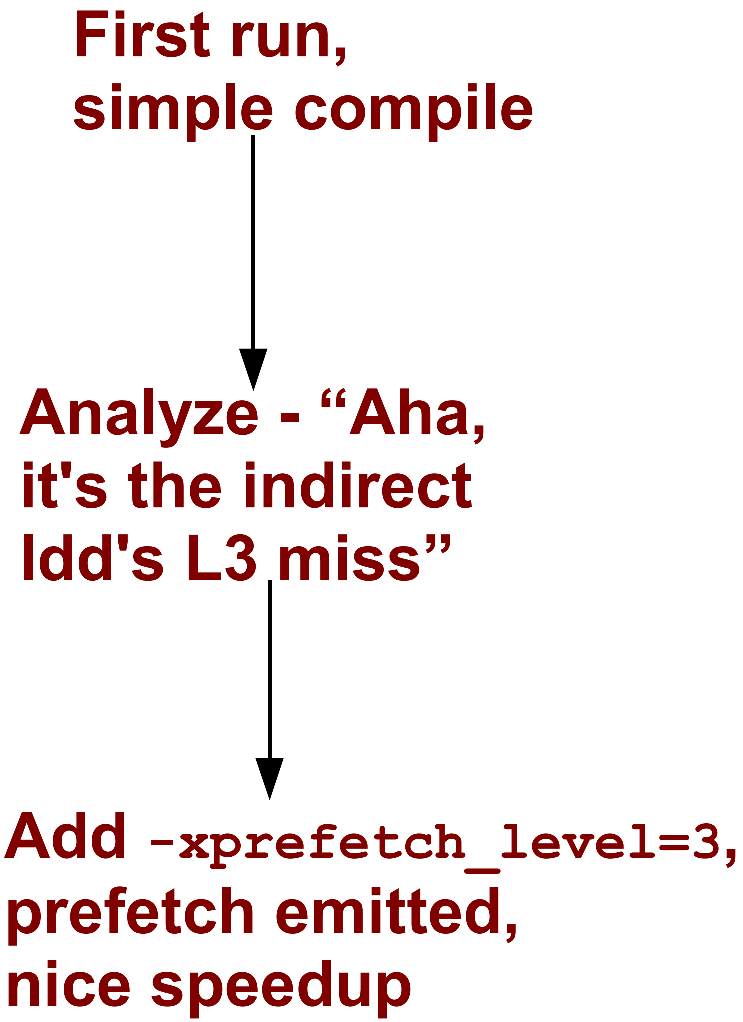
Assembly view, with line numbers

Analysis Example Continued

```
$ cc -fast -g main.c loop.c
main.c:
loop.c:
$ time a.out

real     0m17.49s
user     0m16.24s
sys      0m0.97s
```

```
$ collect -h Instr_cnt,h,L3_miss,h a.out
$ analyzer
```

```
$ cc -fast -g -xprefetch_level=3 main.c loop.c
main.c:
loop.c:
$ time a.out

real     0m6.79s
user     0m5.54s
sys      0m0.97s
```



First run, simple compile

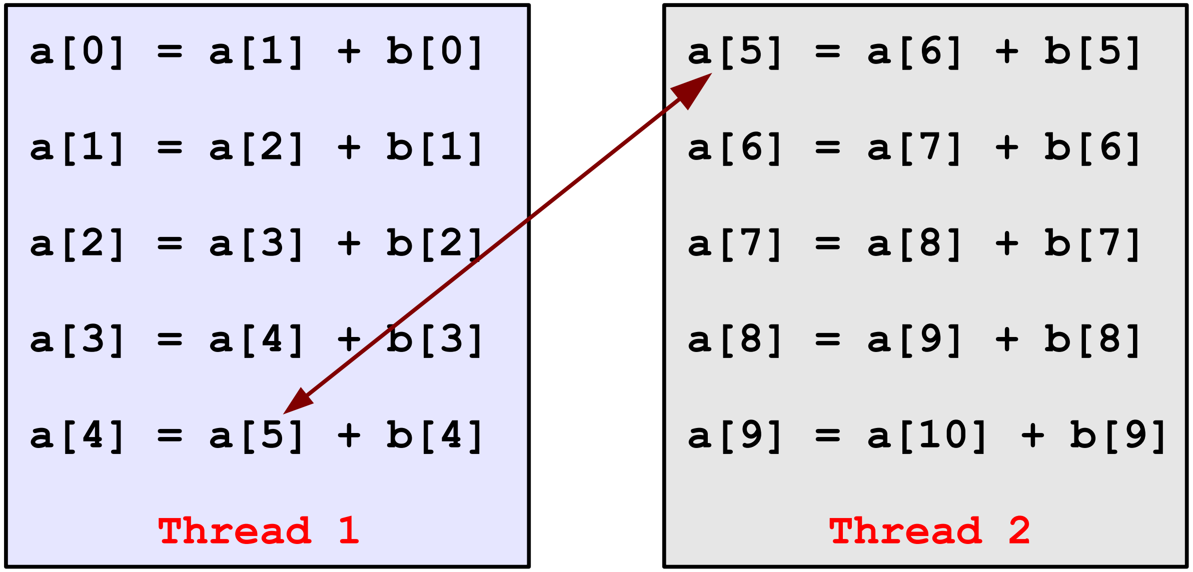
Analyze - “Aha, it's the indirect ldd's L3 miss”

Add `-xprefetch_level=3`, prefetch emitted, nice speedup

Race Detection: What is a race?

```
for (i=0; i<n; i++)
  a[i] = a[i+1] + b[i];
```

Sequential execution: Results are deterministic
Parallel execution: Results non-deterministic



```
a[0] = a[1] + b[0]
a[1] = a[2] + b[1]
a[2] = a[3] + b[2]
a[3] = a[4] + b[3]
a[4] = a[5] + b[4]
```

Thread 1

```
a[5] = a[6] + b[5]
a[6] = a[7] + b[6]
a[7] = a[8] + b[7]
a[8] = a[9] + b[8]
a[9] = a[10] + b[9]
```

Thread 2

Example:
- Thread 1 executes i=0-4
- Thread 2 executes i=5-9
- Thread 2 might write a[5] before thread 1 reads it.
- Final value of a[4] wrong!

This is a data race.

DRDT: A Data Race Detection Tool

Using DRDT – Step 1

- Compile a program for instrumentation.
  - Add “`-xinstrument=datarace`” to the compiler/linker options.

```
% cc -xinstrument=datarace -mt a1.c a2.c a3.c

% cc -xinstrument=datarace -xopenmp omp1.c omp2.c
```

Using DRDT – Step 2

- Run the application under *collect* (analyzer) with the –`r` option
  - Similar to using *collect* for performance analysis

```
% collect -r on a.out arg1 arg2
```

  - This will create a data file that stores the race detection results
- Significant slowdown (> 50X) can occur when using DRDT
  - Use a small input data set for a short run

Using DRDT – Step 3

- Check the result
  - Use *er_print* for command line interface.
    - For a summary report

```
% er_print -races test.1.er
```

    - For a detailed report on one particular data race detected

```
% er_print -rdetail 3 test.1.er
```

  - Use *rdt* for a GUI

```
% rdt test.1.er
```

Example of DRDT Output

```
% er_print -races test.1.er

Total Races:  2 Experiment:  test.1.er

Race #1, Vaddr: 0x212c0
      Access 1: Read,  work + 0x000000A0,
                       line 42 in "pthr_prime.c"
      Access 2: Write, work + 0x000000DC,
                       line 44 in "pthr_prime.c"
  Total Traces: 3

Race #2, Vaddr: 0x212c0
      Access 1: Write, work + 0x000000DC,
                       line 44 in "pthr_prime.c"
      Access 2: Write, work + 0x000000DC,
                       line 44 in "pthr_prime.c"
  Total Traces: 2
```

Analyzing & Improving Binaries

- BIT - A tool that operates *reliably* on binaries
- Can instrument and collect information for analysis
- Can create a new binary with improved performance
  - Focusses on rearranging code to better use the I-cache
  - Works best on large, complex applications
- Build with
  - Option -xbinopt=prepare
  - Use -O1 or higher optimization level

BIT: Examining Code Coverage

```
$ cc -fast -xbinopt=prepare *.c -lm         // Build for BIT
...
and.c:
build-disjuncts.c:
extract-links.c:
...
$ bit instrument a.out                      // Instrument the a.out
$ a.out.instr 2.1.dict -batch < input       // Run a.out.instr
$ bit coverage a.out                        // Analyze coverage
Creating experiment database test.1.er ...
BIT Code Coverage
Total Functions: 350
Covered Functions: 216
Function Coverage: 61.7%
Total Basic Blocks: 6,041
Covered Basic Blocks: 3,969
Basic Block Coverage: 65.7%
Total Basic Block Executions: 3,955,536,194
Average Executions per Basic Block: 654,781.69
Total Instructions: 27,606
Covered Instructions: 17,680
Instruction Coverage: 64.0%
Total Instruction Executions: 18,866,478,764
Average Executions per Instruction: 683,419.50
```

Simplifying Performance Optimisation

- SPOT – A Simple Performance Optimisation Tool
  - Produces a report on a code's execution
  - Exposes common causes of performance loss
  - Very easy to use
- SPOT reports contain hyperlinked profiles
  - Makes it easy to navigate from performance issue to source to assembly
  - For maximum information
    - Add -g (-g0 for C++)
    - Use -O1 or higher
    - Include -xbinopt=prepare

Using SPOT

```
$ cc -fast -xbinopt=prepare -g *.c -lm                 // Build the code
...
and.c:
build-disjuncts.c:
extract-links.c:
...
$ spot -X a.out 2.1.dict -batch < input > /dev/null    // Run SPOT
Copying spot resources
Collect machine statistics
Collect application details
Collect ipc data using ripc
Collect data using BIT
Output ifreq data from bit
Collect bandwidth data
Collect traps data
Collect HW counter profile data
Collect data for Rstall_IU_use & Re_DC_miss
Collect data for Rstall_storeQ & Cycle_cnt
Collect data for Dispatch0_2nd_br & Cycle_cnt
Generating html output for HW counter profile data
Collect clock-based profiling data
Generating html output for time profile data
Done collecting, tidying up reports
$
```

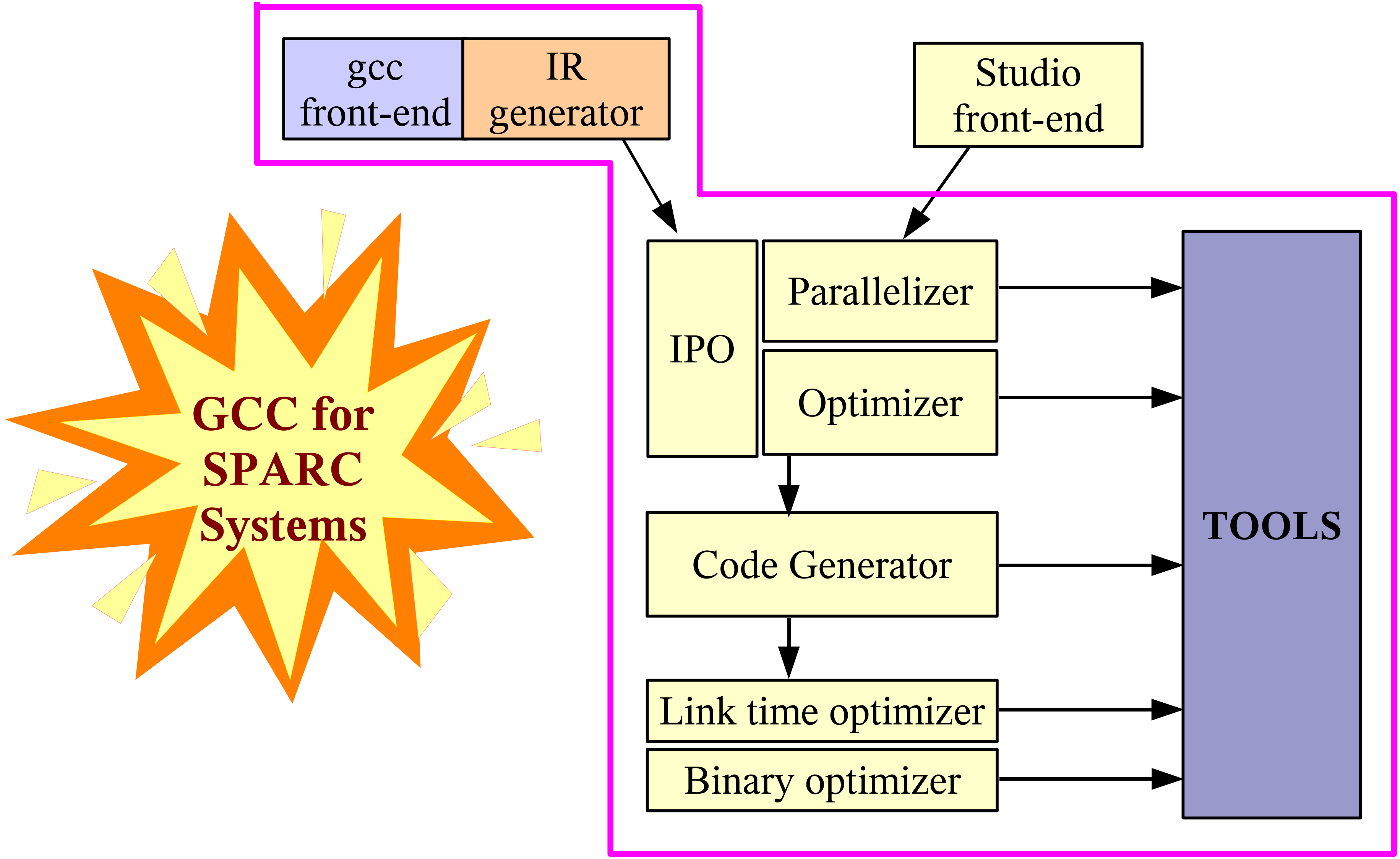
- SPOT runs the code many times
- Approx. 20X longer (in this mode)
- Invokes other tools to collect data
- Generates comprehensive report

Embracing OpenSPARC/gcc Users

- Many developers use gcc
  - Want to use the same compiler for different platforms
  - Use gcc language extensions
  - Familiar with & feel comfortable with gcc
  - Migration to Studio is, or is viewed as being, difficult

Would be nice to bring the features of Studio to these users.

Making the Connection



gcc front-end

IR generator

Studio front-end

IPO

Parallelizer

Optimizer

Code Generator

Link time optimizer

Binary optimizer

TOOLS

GCC for SPARC Systems

Key Features

- Transparent to gcc users
  - > Feature compatible with gcc
  - > Debuggable with gdb and dbx
- Improved performance
  - > Through advanced optimizations tuned to SPARC systems
  - > Extra optimizations such as -xipo, -xprefetch, -xprofile
- Higher reliability

Summary

- A rich collection of compilers and tools is available to OpenSPARC developers
  - Components are thread-aware and work synergestically
  - Reliable, with advanced optimizations and parallelization
  - Excellent multi-threaded analysis and debugging tools
- These tools are all free and can be downloaded from:

http://cooltools.sunsource.net

開放的
열린
مفتوح
libre
मुक्त
ಮುಕ್ತ
livre
libero
ముక్త
开放的
açık
open
nyílt
פתוח
オープン
livre
ανοικτό
offen
otevřený
öppen
открытый
வெளிப்படை

open

64 bit, 32 threads, free

Cool Tools – Automatic Tuning and Trouble Shooting Systems (ATS)

David Weaver

Agenda

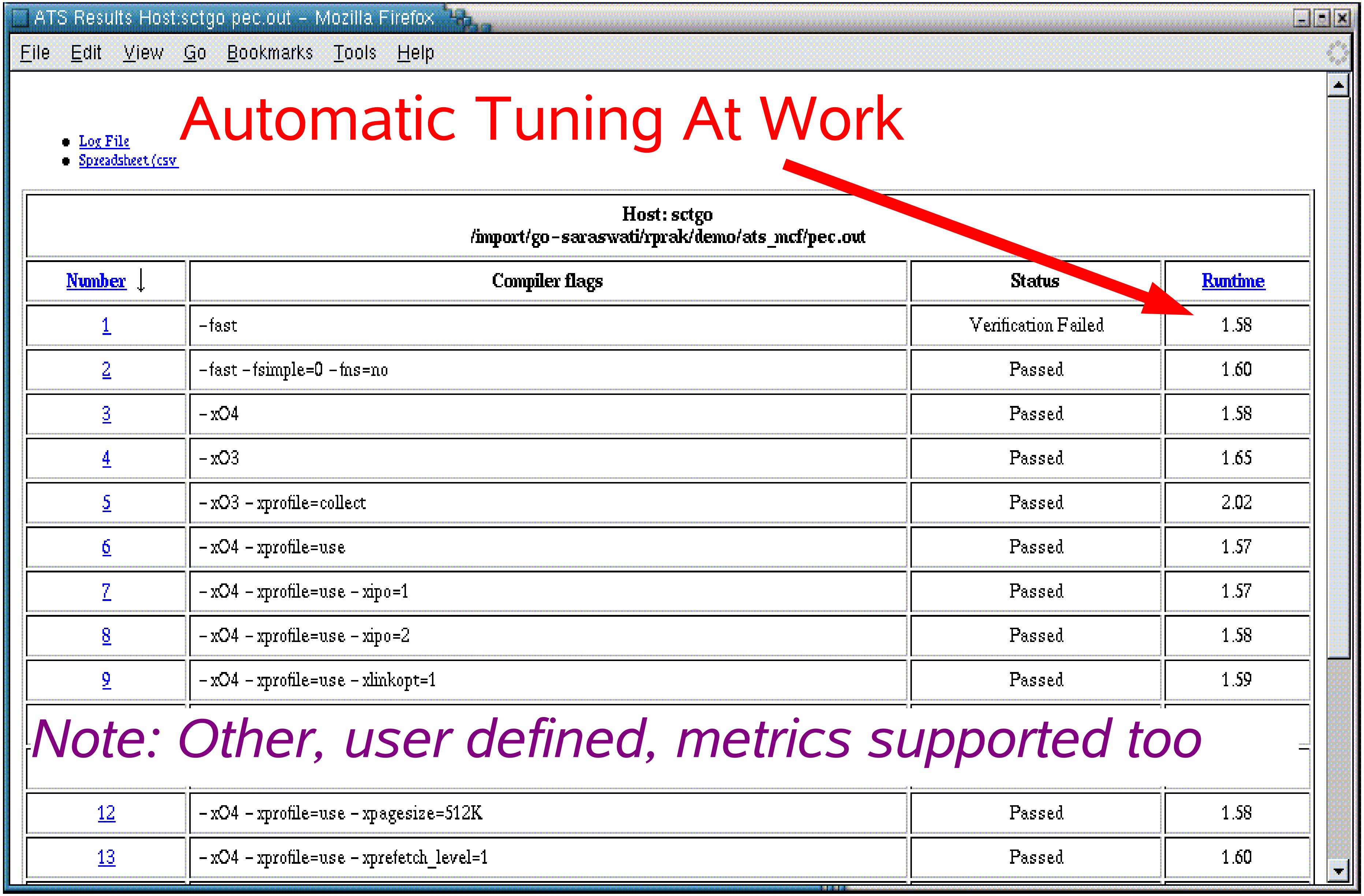
- Automatic Tuning and Troubleshooting System
  - > Build the best binaries
  - > Bonus: troubleshooting
- Unified Solaris Binary
  - > Deliver the best-fit binary at run-time

Automatic Tuning and Troubleshooting System (ATS)

Automatic Tuning Through ATS

```
% ats a.out
```

Automatic Tuning At Work


ATS Results Host:sctgo pec.out - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

- Log File
- Spreadsheet (csv

Host: sctgo
/import/go-saraswati/rprak/demo/ats_mcf/pec.out

Number ↓	Compiler flags	Status	Runtime
1	-fast	Verification Failed	1.58
2	-fast -fsimple=0 -fns=no	Passed	1.60
3	-xO4	Passed	1.58
4	-xO3	Passed	1.65
5	-xO3 -xprofile=collect	Passed	2.02
6	-xO4 -xprofile=use	Passed	1.57
7	-xO4 -xprofile=use -xipo=1	Passed	1.57
8	-xO4 -xprofile=use -xipo=2	Passed	1.58
9	-xO4 -xprofile=use -xlinkopt=1	Passed	1.59
12	-xO4 -xprofile=use -xpagesize=512K	Passed	1.58
13	-xO4 -xprofile=use -xprefetch_level=1	Passed	1.60

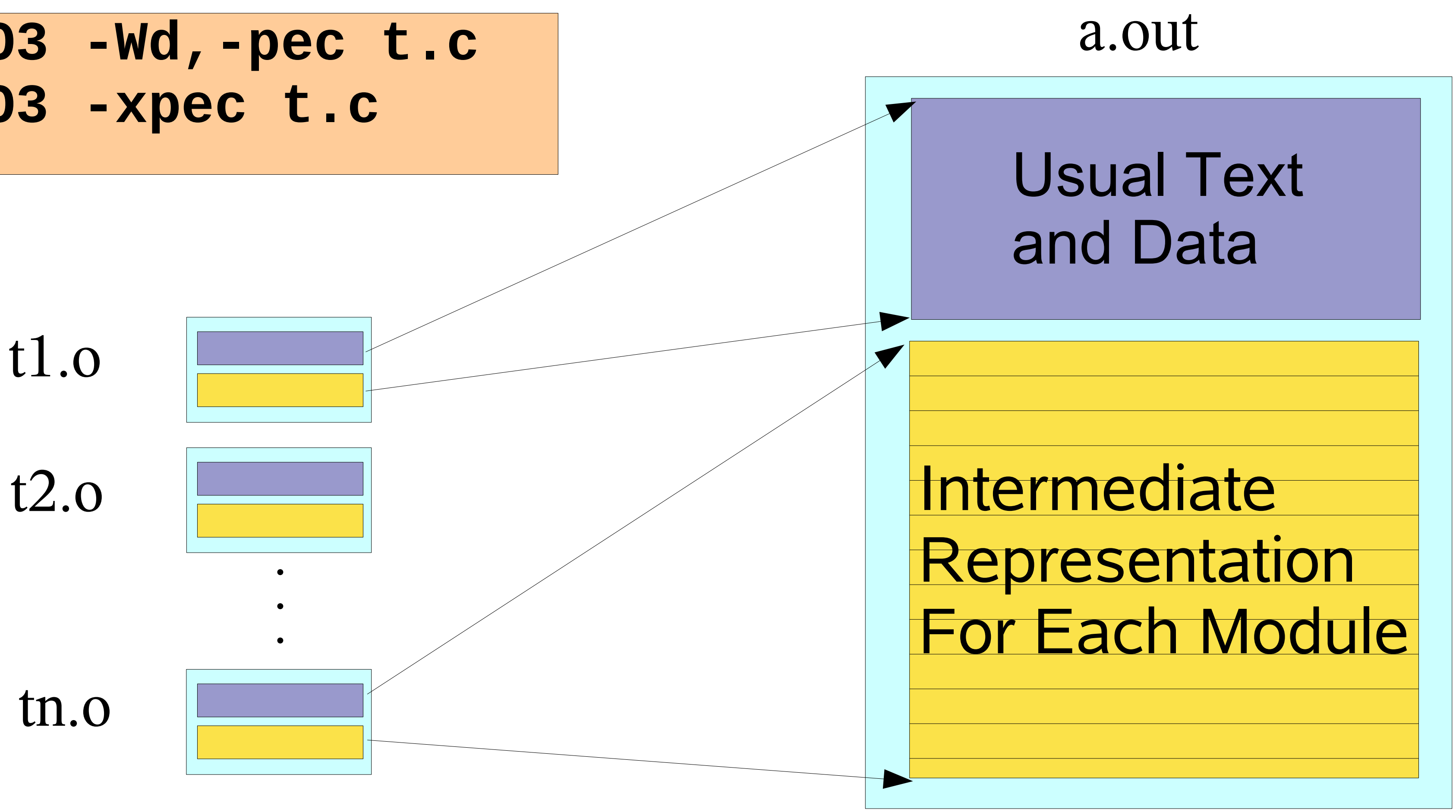
Note: Other, user defined, metrics supported too

How Does It Work? -- ATS uses PEC

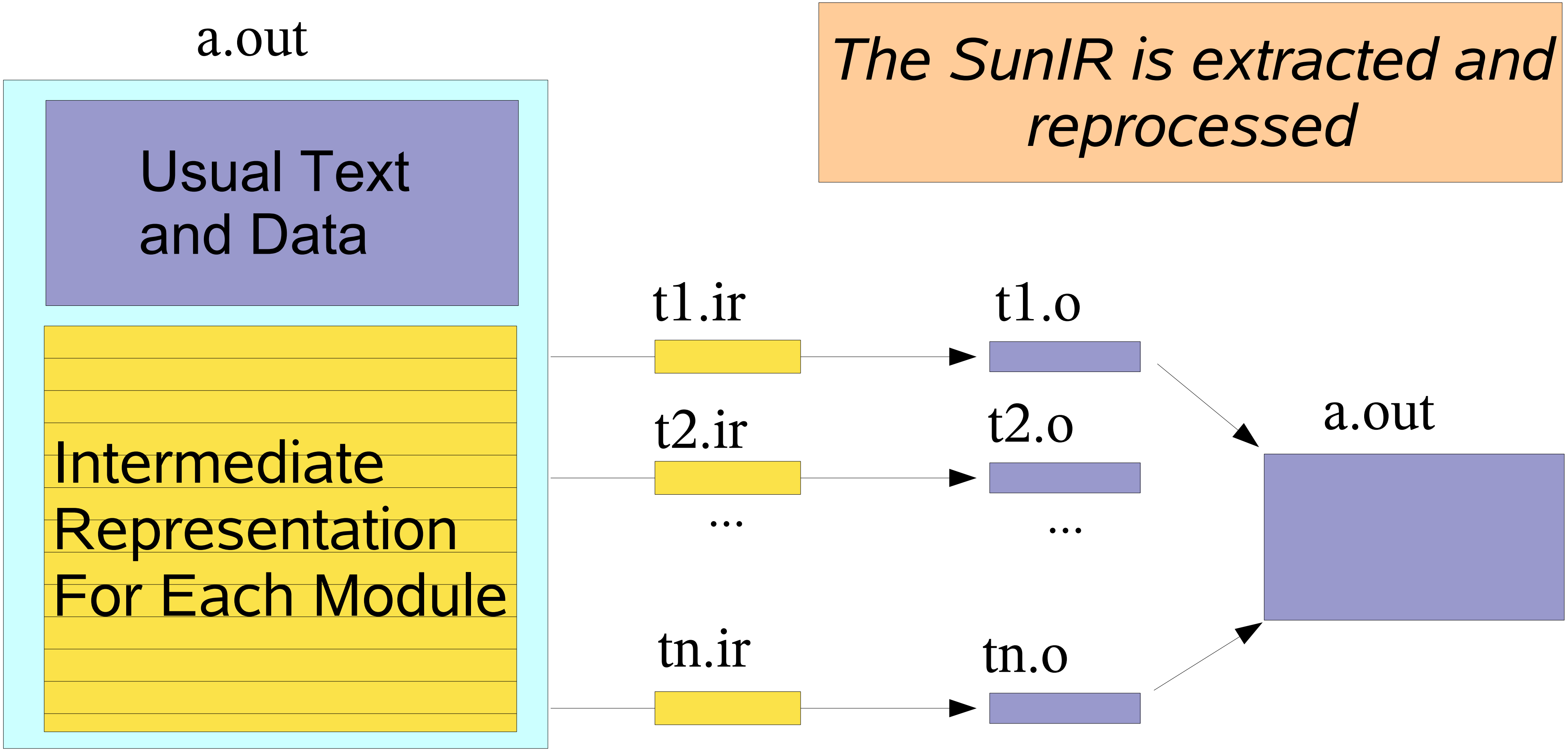
PEC = Portable Execution Code

(SunIR is kept in the binary)

```
% cc -xO3 -Wd,-pec t.c
% gcc -O3 -xpec t.c
```



Recompiling Binaries Through PEC



The SunIR is extracted and reprocessed

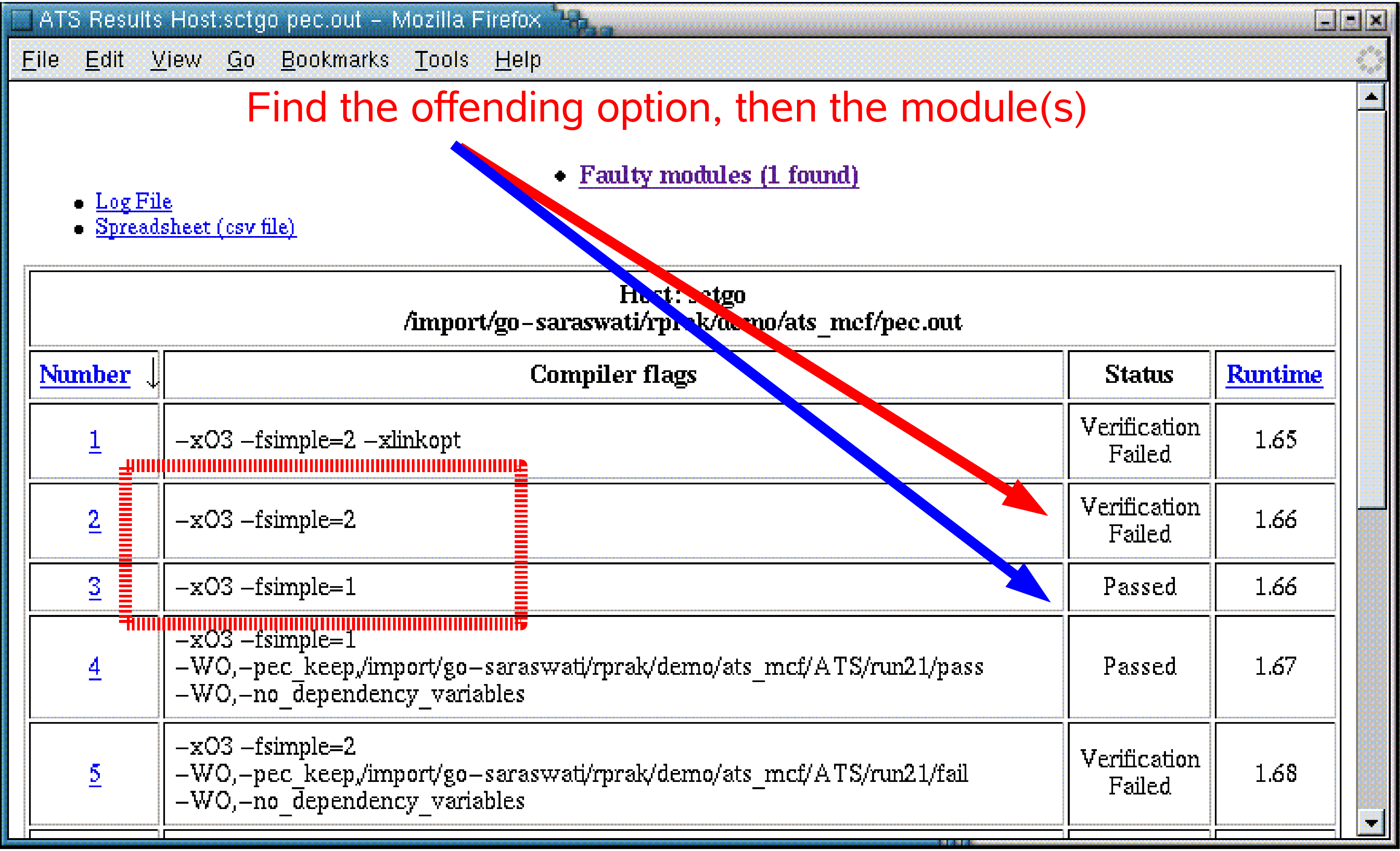
Dramatic reduction in compile time

Troubleshooting - Findbug

- Find the offending options
- Then find the offending module

```
% ats \
   -i 'script:findbug -xO3 -fsimple=2 -xlinkopt' \
   a.out
```

Find the offending option, then the module(s)



- Faulty modules (1 found)

- Log File
- Spreadsheet (csv file)

Host: sctgo
/import/go-saraswati/rprak/demo/ats_mcf/pec.out

Number ↓	Compiler flags	Status	Runtime
1	–xO3 –fsimple=2 –xlinkopt	Verification Failed	1.65
2	–xO3 –fsimple=2	Verification Failed	1.66
3	–xO3 –fsimple=1	Passed	1.66
4	–xO3 –fsimple=1 –WO,–pec_keep,/import/go–saraswati/rprak/demo/ats_mcf/ATS/run21/pass –WO,–no_dependency_variables	Passed	1.67
5	–xO3 –fsimple=2 –WO,–pec_keep,/import/go–saraswati/rprak/demo/ats_mcf/ATS/run21/fail –WO,–no_dependency_variables	Verification Failed	1.68

Unified Solaris Binary

Making A Unified Solaris Binary

- Make platform specific versions of a binary
- Add them to a unified solaris binary
- Example:

```
% mkusb -c ultra3  build-ultra3/a.out  -o a.out

% mkusb -c ultraT1 build-ultraT1/a.out -a a.out
```

Running A Unified Solaris Binary

- Just as you would run a normal binary

```
% a.out
Hello World!
```

```
% setenv USB_VERBOSE 1

% a.out
## BEST MATCHING BINARY ultra3
Hello World!
```

Making Another Unified Solaris Binary

- Example:

```
% mkusb -c sparc sparc.out   -o a.out
% mkusb -c x86   x86.out     -a a.out
% mkusb -c ultra4 ultra4.out -a a.out
```

How Does It Work?

- Binaries are compressed, encoded and added to the Unified Solaris Binary
- At runtime
  - Run machine is recognized
  - Best matching binary is extracted and cached into a directory and executed
  - For example, on an ultra3cu, order of perference
    - ultra3cu ultra3i ultra3iplus ultra3
      ultra2 ultra2i ultra2e
      ultra sparc
- When run again, the *cached* binary is used

開
放
的
열린
مفتوح
libre
मुक्त
ಮುಕ್ತ
livre
libero
ముక్త
开放的
açık
open
nyílt
⠕⠏⠑⠝
פתוח
オープン
livre
ανοικτό
offen
otevřený
öppen
открытый
வெளிப்படை

open

64 bit, 32 threads, free

http://OpenSPARC.net

OpenSPARC participation

- Community Registration:
  - > http://www.sunsource.net/servlets/Join After registration and confirming password, you can join the mailing lists: http://www.sunsource.net.servlets/ProjectMailingListsList
- Forums:
  - > http://forum.java.sun.com/category.jspa?categoryID=120 (separate registration required for posting)

OpenSPARC participation

- Add your university (or company) to the marketplace: http://www.opensparc.net/community-marketplace/
- Send us your profile and we'll post it: http://www.opensparc.net/profiles/
- Add yourself to our Frappr!!: http://wwwopensparc.net/frappr.html
- Contribute to our OpenSPARC Book: http://wiki.opensparc.net/bin/view.pl/Main/Webhome (separate registration required for editing)