MultiCore Design: OpenSPARC T1 & T2

Architecture, OS, Compilers, Tools & Open Source Community

Presented by:
Aman Joshi
Director, IC Tools & Solutions
MicroElectronics Division
Sun Microsystems, Inc.

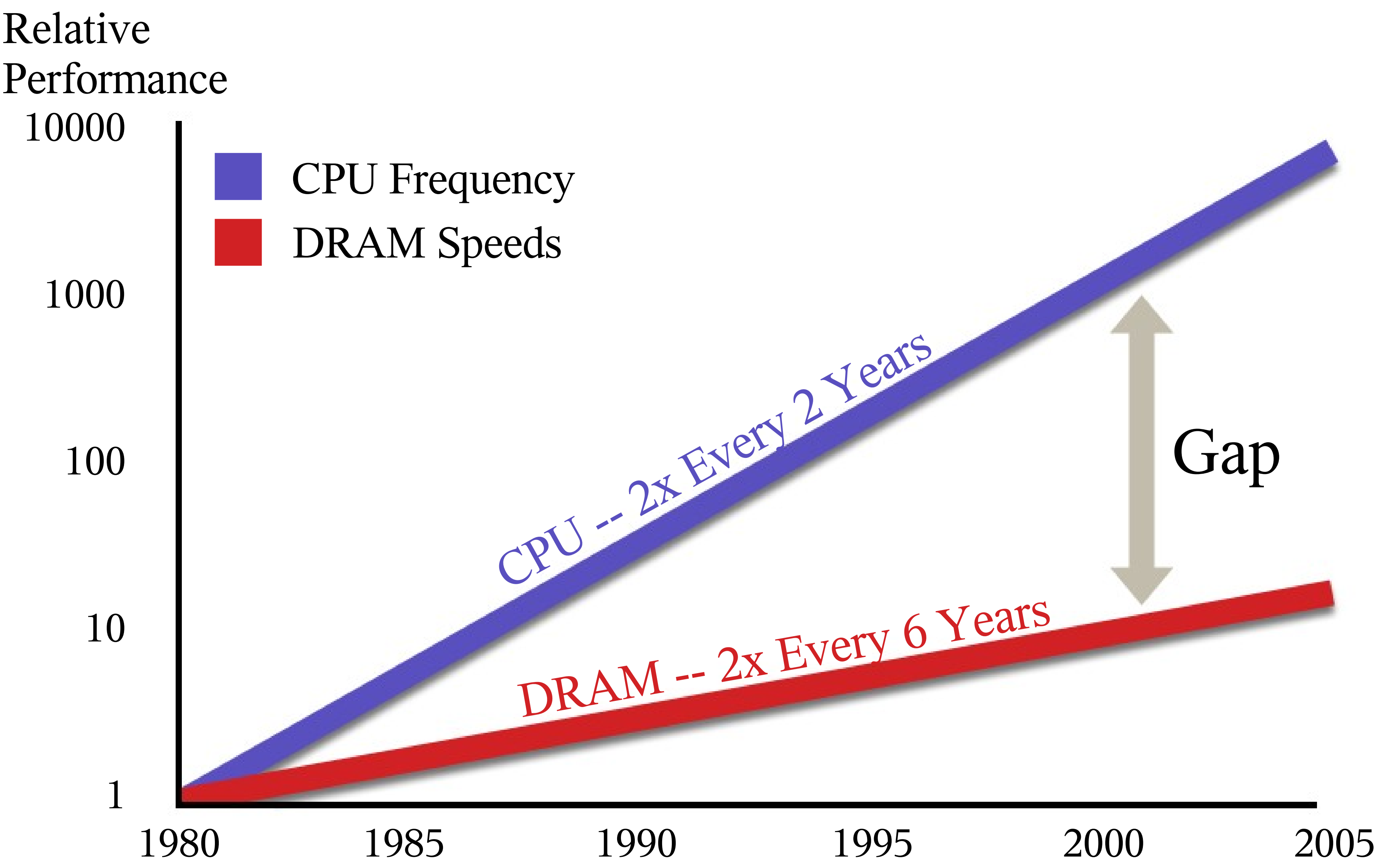
Acknowledgements:
David Weaver, Jhy-chun Wang, Paul Jordan
and others

- *OpenSPARC . net*

Agenda

1. Chip Multi-Threading (CMT) Era
2. OpenSPARC Program
3. Open Source T1 Overview
4. OpenSPARC T1, T2, T2 Plus
5. OpenSPARC Core on FPGA
6. Sun's Chip Design Tools & Flows
7. OpenSPARC Simulators
8. Hypervisor & Virtualization
9. Compiler Optimizations and tools
10. Community Participation

Memory Bottleneck

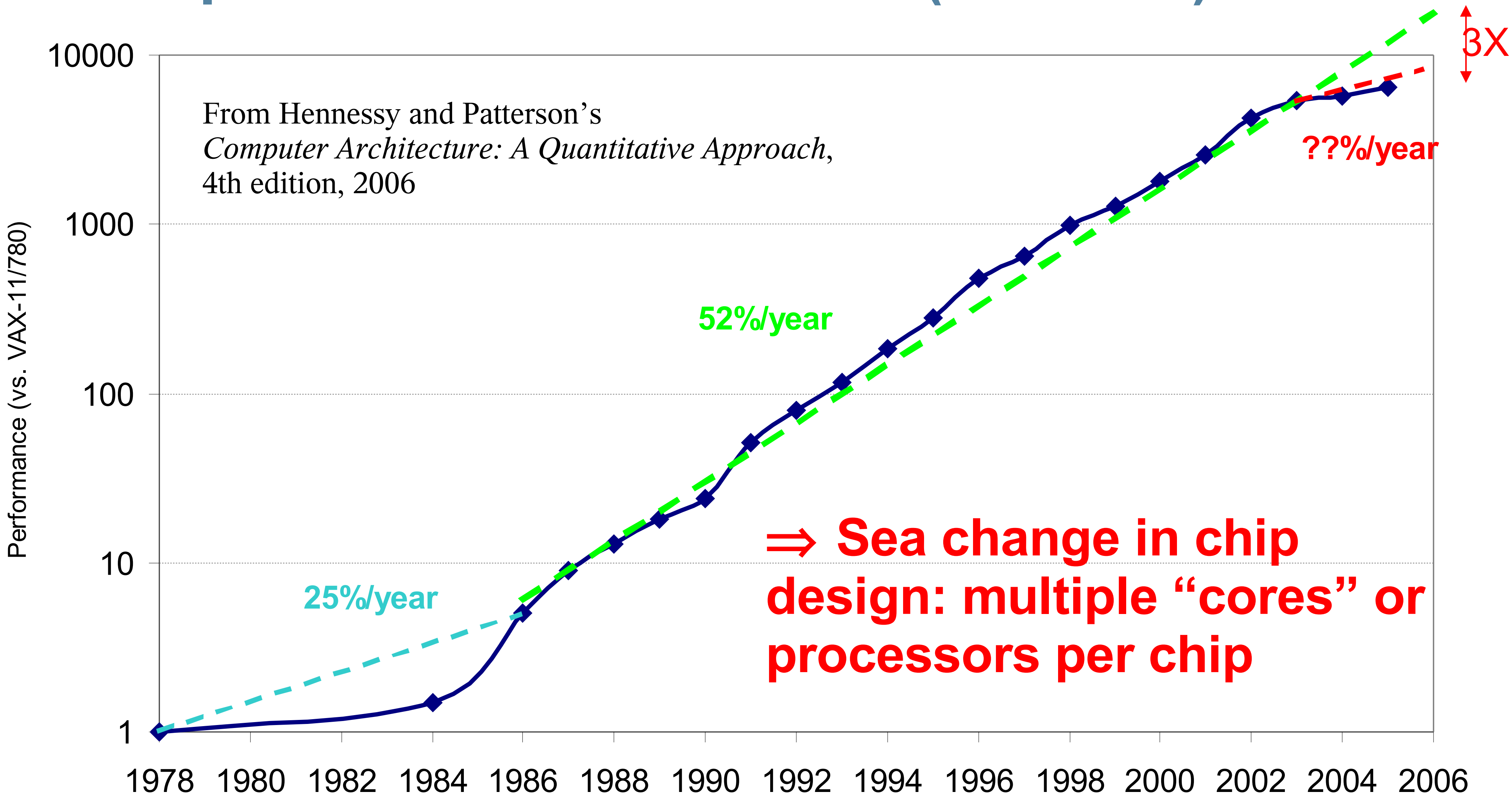


Source: Sun World Wide Analyst Conference Feb. 25, 2003

“Hitting walls” in Processor Design

- Clock frequency
  - frequency increases tapering off, in new semiconductor processes
  - high frequencies => *power* issues
- Memory latency (not instruction execution speed) dominating most application times
- Processor designs for high single-thread performance are becoming *highly* complex, therefore:
  - expense and/or time-to-market suffer
  - verification increasingly difficult
  - more complexity => more circuitry => increased power ... for diminishing performance returns

Uniprocessor Performance (SPECint)



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

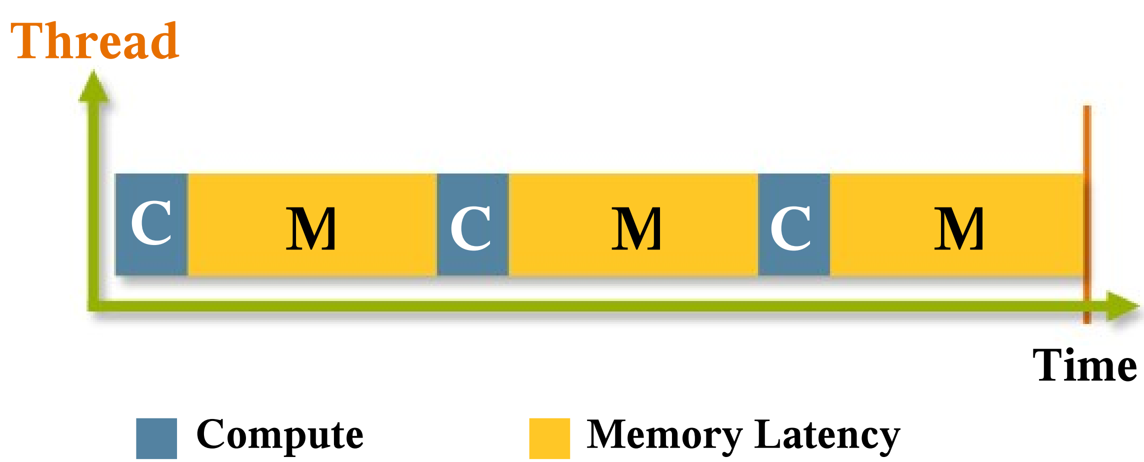
Source: David Patterson presentation at MultiCore Expo, March 2006

Single Threading

Up to 85% Cycles Spent Waiting for Memory

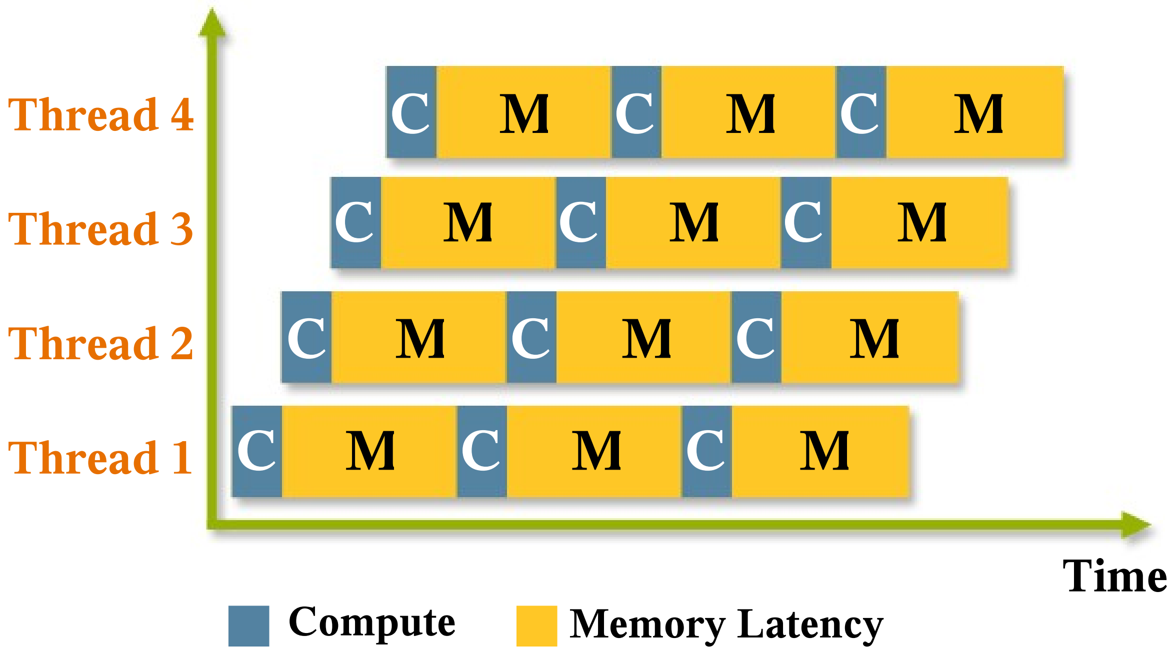
Single Threaded Performance

Typical Utilization of Processor: 15–25%



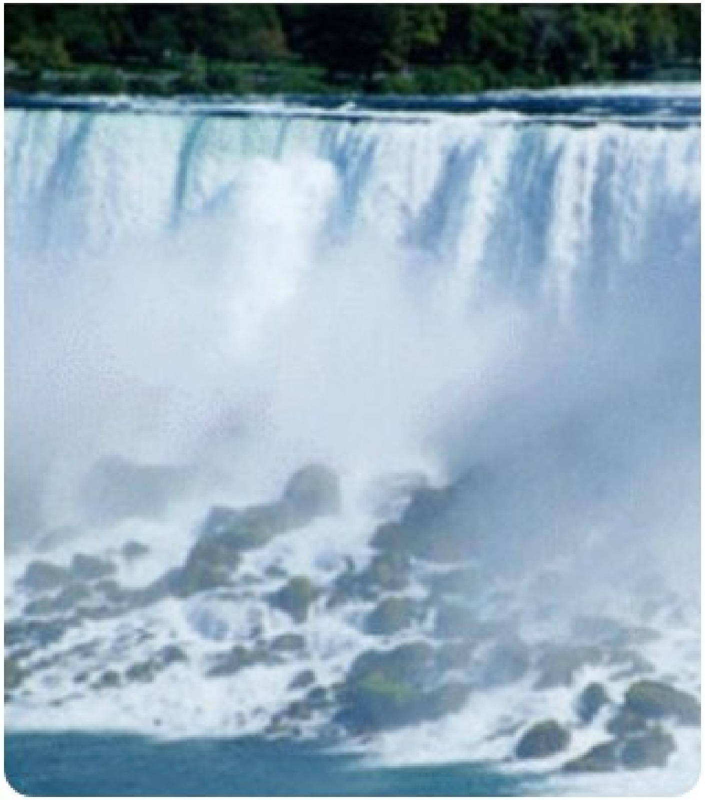
Hardware Multi-Threading (HMT)

Utilization: Up to 85%*

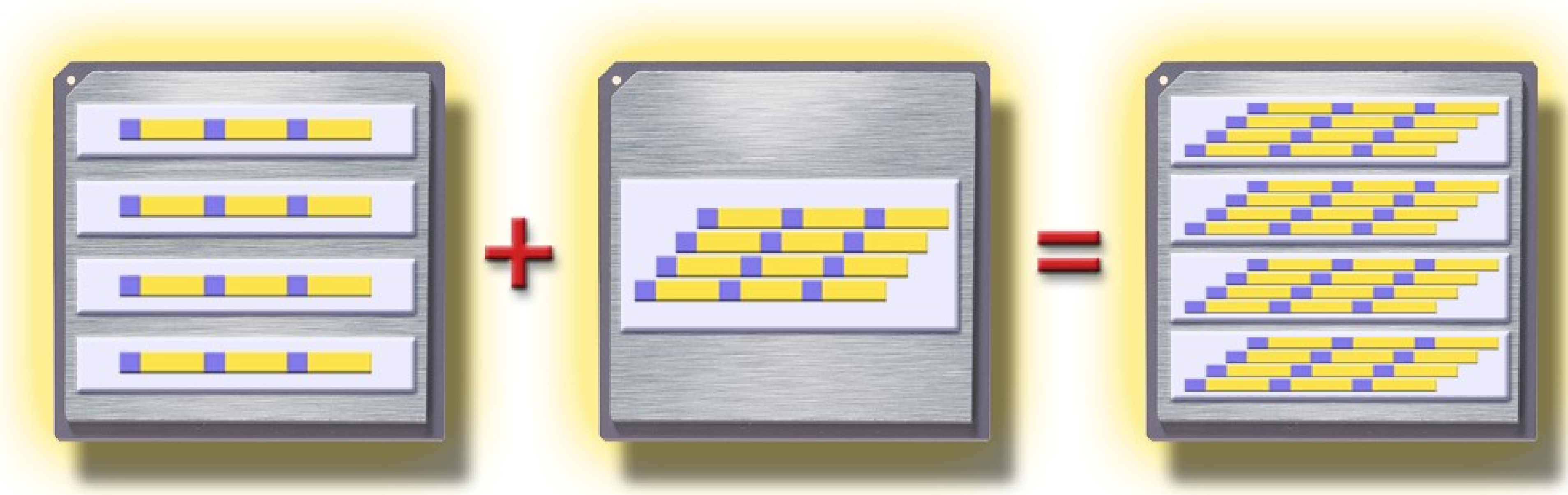


* based on example of UltraSPARC T1

Multi-threaded Performance



Chip Multi-Threading (CMT)



CMP
(Chip MultiProcessing, a.k.a. "multicore")

n cores per processor

HMT
(Hardware Multithreading)

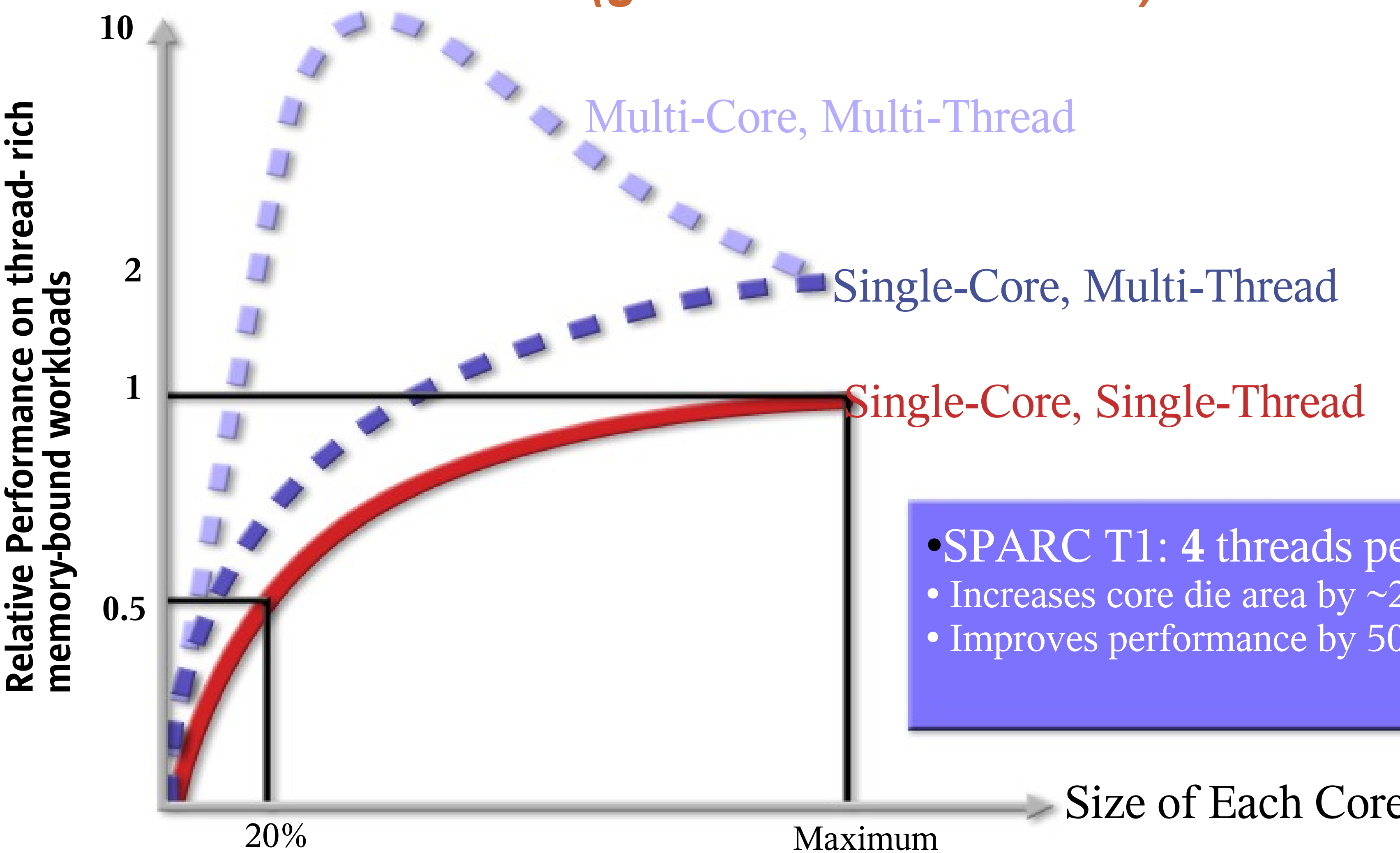
m threads per core

CMT
(Chip MultiThreading)

n x *m* threads per processor

Why CMT Works

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



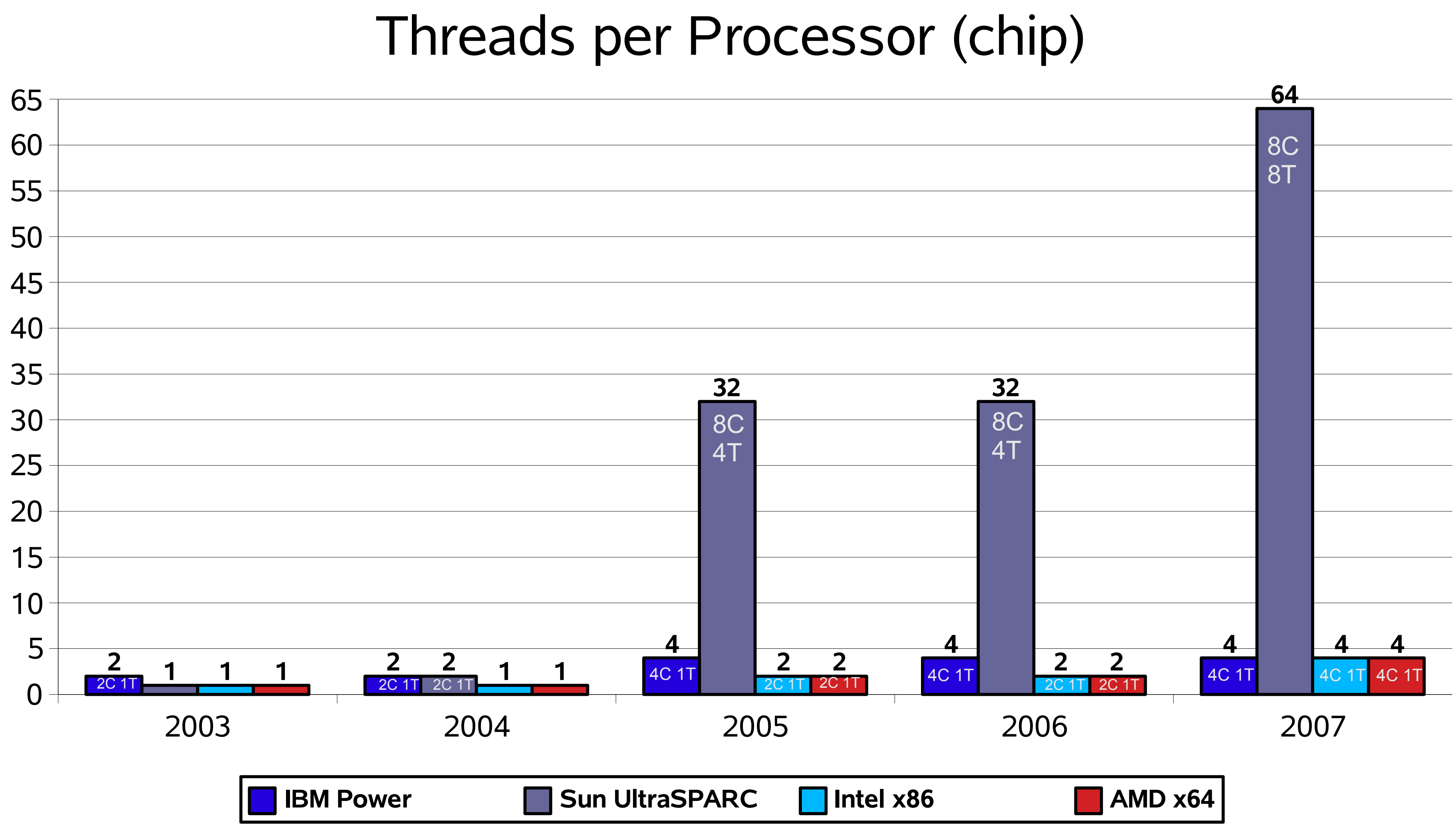
- SPARC T1: **4** threads per core
  - Increases core die area by ~20%
  - Improves performance by 50–100%

Major shift in processor design

- **<u>FROM</u>** *single-thread performance*
  - ever-increasing clock rate
  - IPC (e.g. superscalar, out-of-order) and ILP (Instruction Level Parallelism) - high power consumption
  - cross-CPU communication through bus/memory
  - running a single OS

- **<u>TO</u>** *multi-threaded performance*
  - high thread count (TLP)
  - high throughput
  - high efficiency (performance/power)
  - high inter-CPU(strand) bandwidth
  - virtualization and multiple guest OSs

The Tidal Wave of CMT is Building



OS & Compilers Playing "Catch up"

- A tiny handful of Operating Systems scale well to hundreds of threads*
  - generally, those previously used for 100+ processor SMPs
- Most only scale up to a few (4-8) threads
  - generally, those previously targeted at desktop systems
- Improving auto-parallelization
  - to automatically fork threads to take advantage of CMT
- Need more work on both
  - totally automatic parallelization
  - parallelization with directives (e.g. OpenMP)

* including Solaris

Applications Playing “Catch up”

- Application software is generally *waaaay* behind the CMT curve
- **Good** news:
  many Java apps are inherently multi-threaded
- **Mediocre** news:
  smarter compilers will help many apps
- **Bad** news:
  - some apps require *rewriting* to perform well in the CMT age
  - most programmers aren’t used to thinking in terms of executing concurrent threads

Academic Curricula Opportunities

- Train students in software implications of CMT & multi-core architectures on:
  - operating system design
  - compiler/tools design
  - application design

- Train processor architects on *real-world* tradeoffs
  - performance/complexity vs. power consumption
  - performance vs. *time to market!*
    - additional performance *only* worthwhile if it can be implemented quickly enough
    - 1 month delay trades away ~5% of performance
  - **Verification** takes *twice* the time/effort/$ of **design**
    - so make the design easier to verify

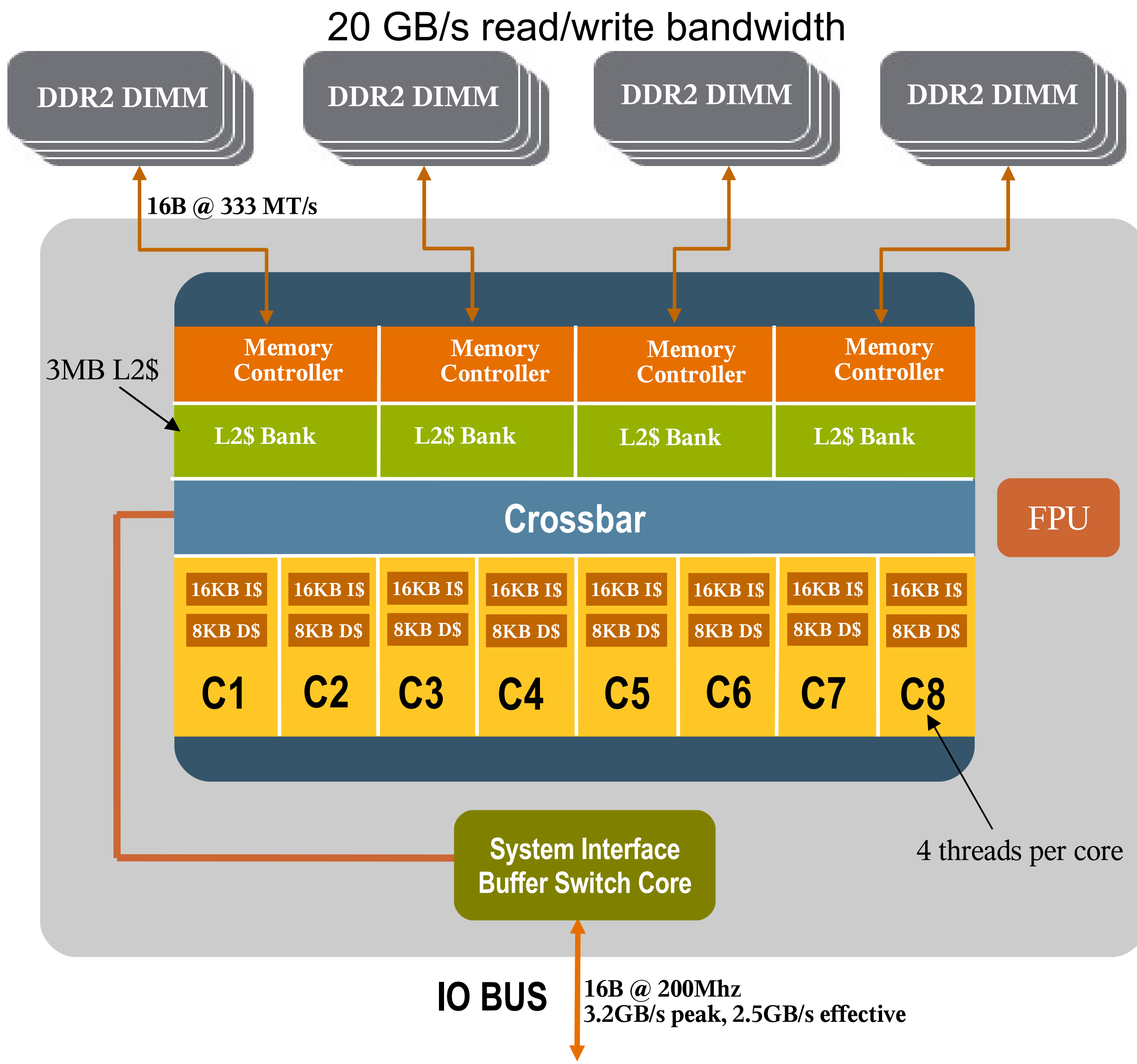
OpenSPARC Program

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

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 momentum: >7000 downloads

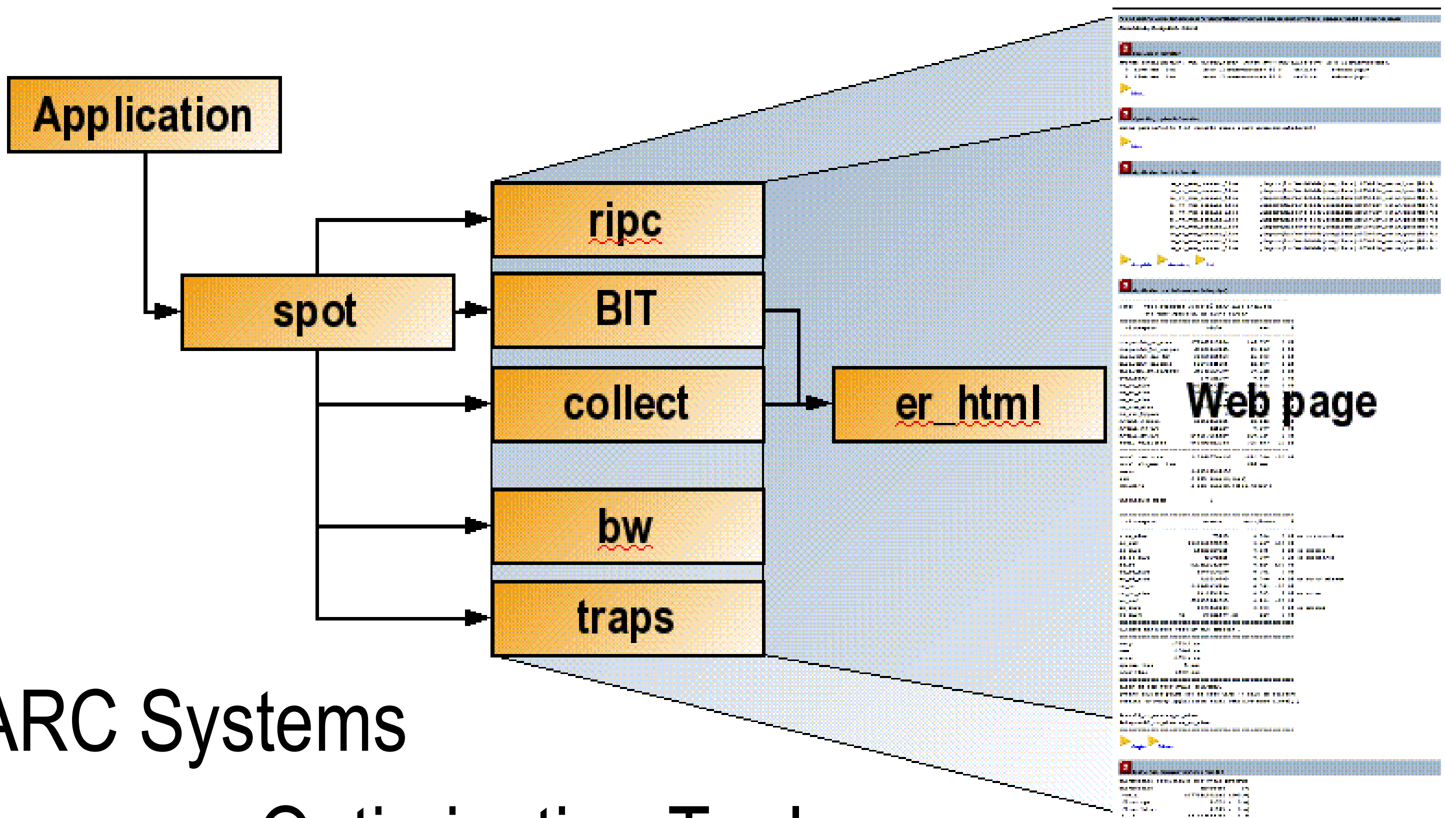
Innovation Happens Everywhere

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 released 65nm synthesis results
- Collaborative effort on RAMP (build 1000 core system)
- > 7000 downloads.

Cool tools for SPARC systems

http://cooltools.sunsource.net/

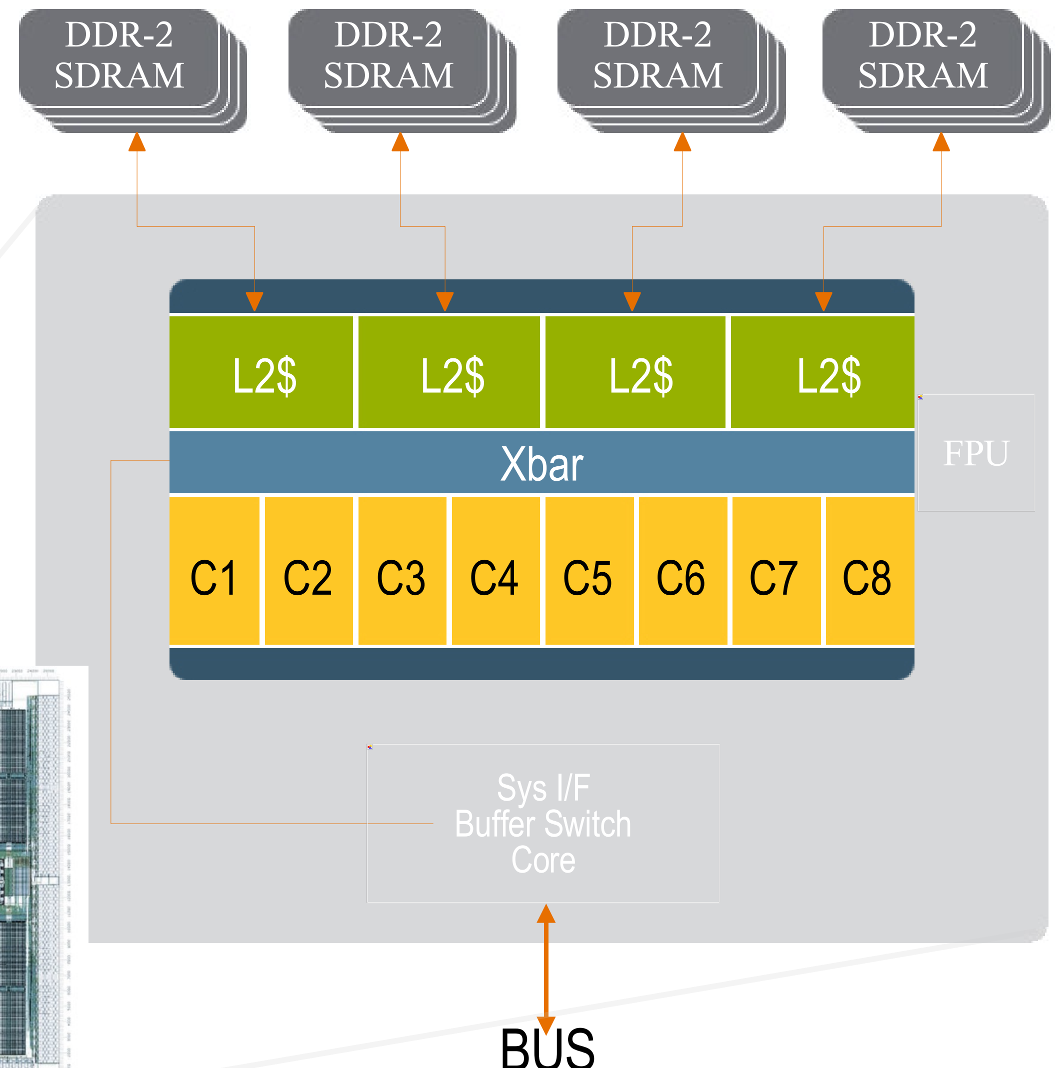


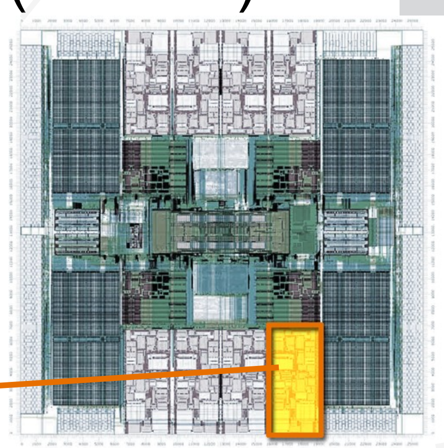
- GCC for SPARC Systems
- Simple Performance Optimization Tool
- Automatic Tuning and Troubleshooting Tool

OpenSPARC T1
Processor Overview

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



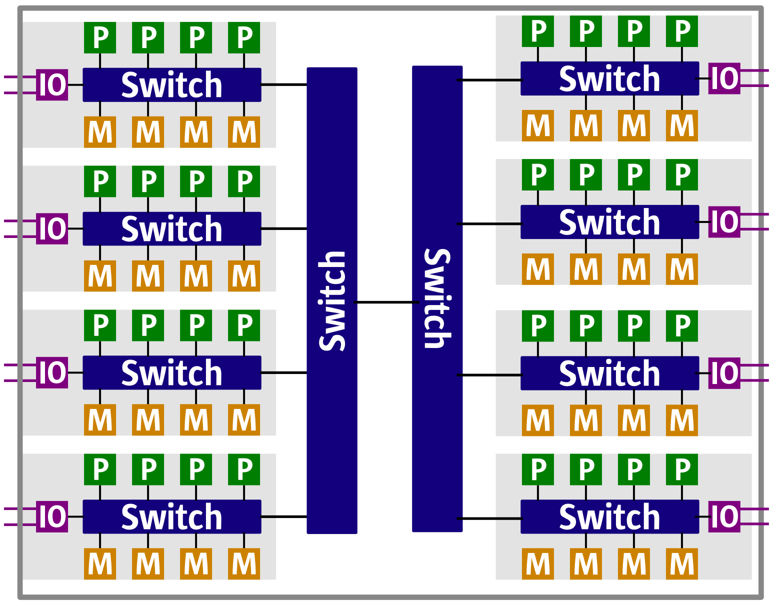


1 of 8 Cores

CMT: On-chip = High Bandwidth

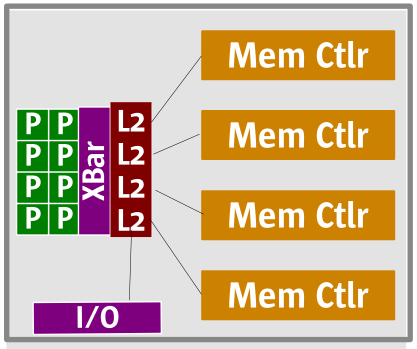
32-thread Traditional SMP System

Example: Typical SMP Machine Configuration



32-thread OpenSPARC T1 Processor

One motherboard, no switch ASICs

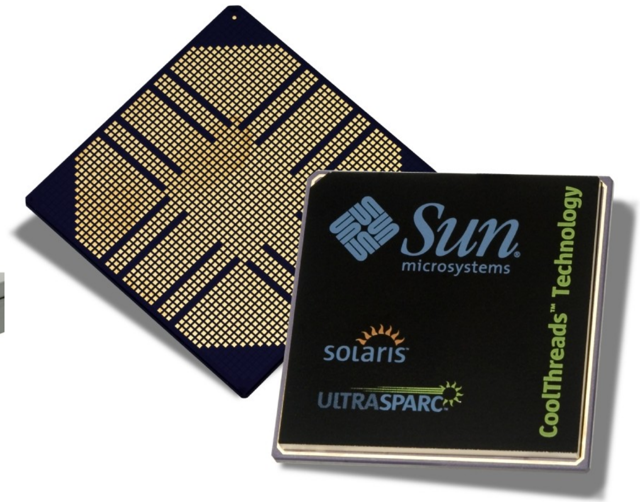


Direct crossbar interconnect

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

CMT Pays Off with CoolThreads™ Technology

Sun Fire T1000

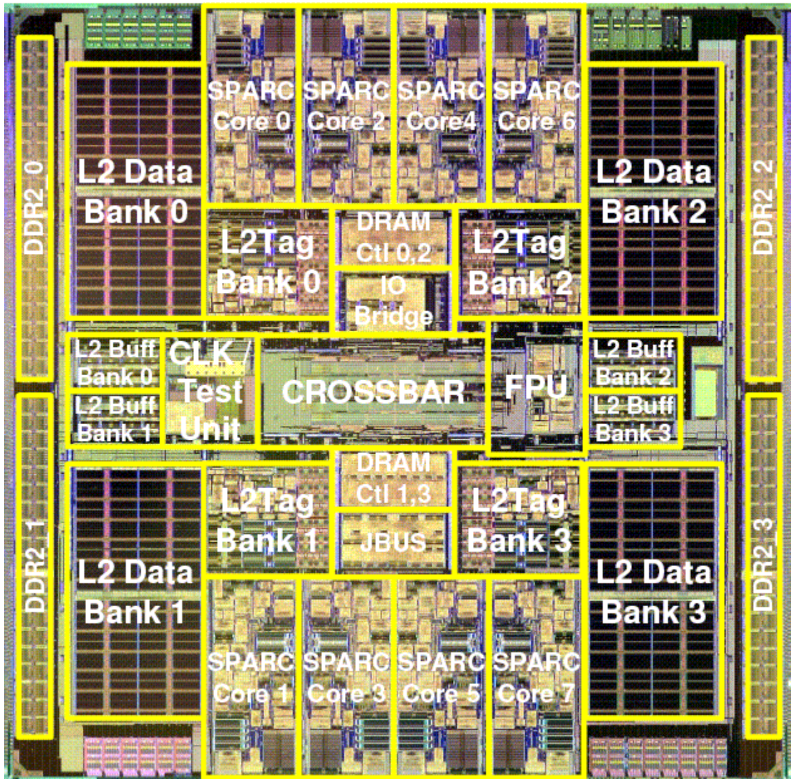


Sun Fire T2000

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

*See disclosures

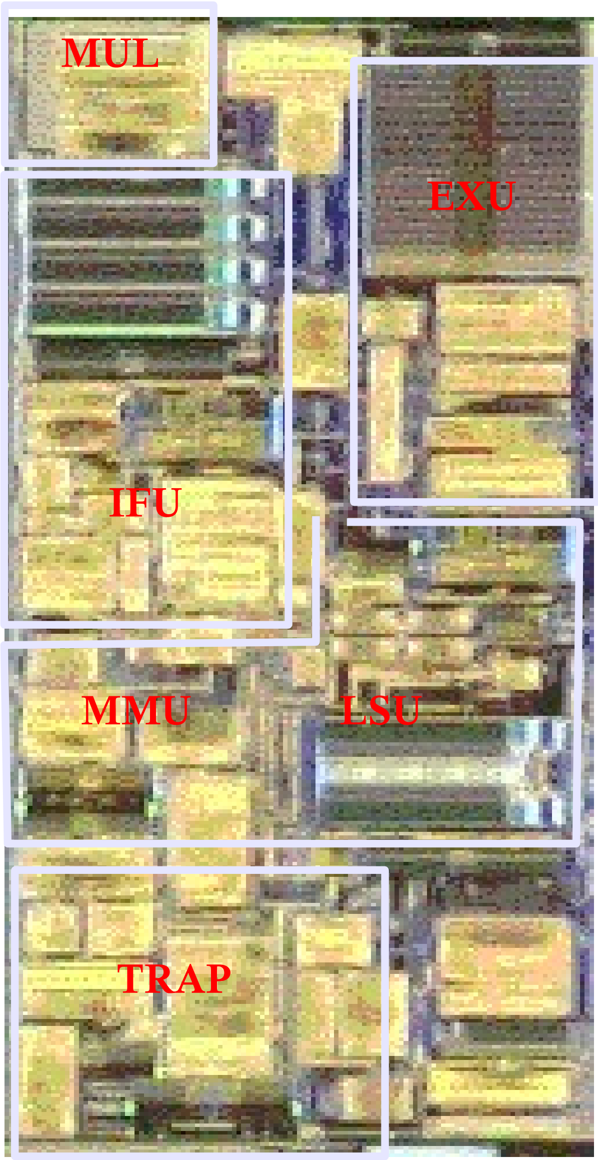
UltraSPARC-T1: Choices & Benefits



- 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

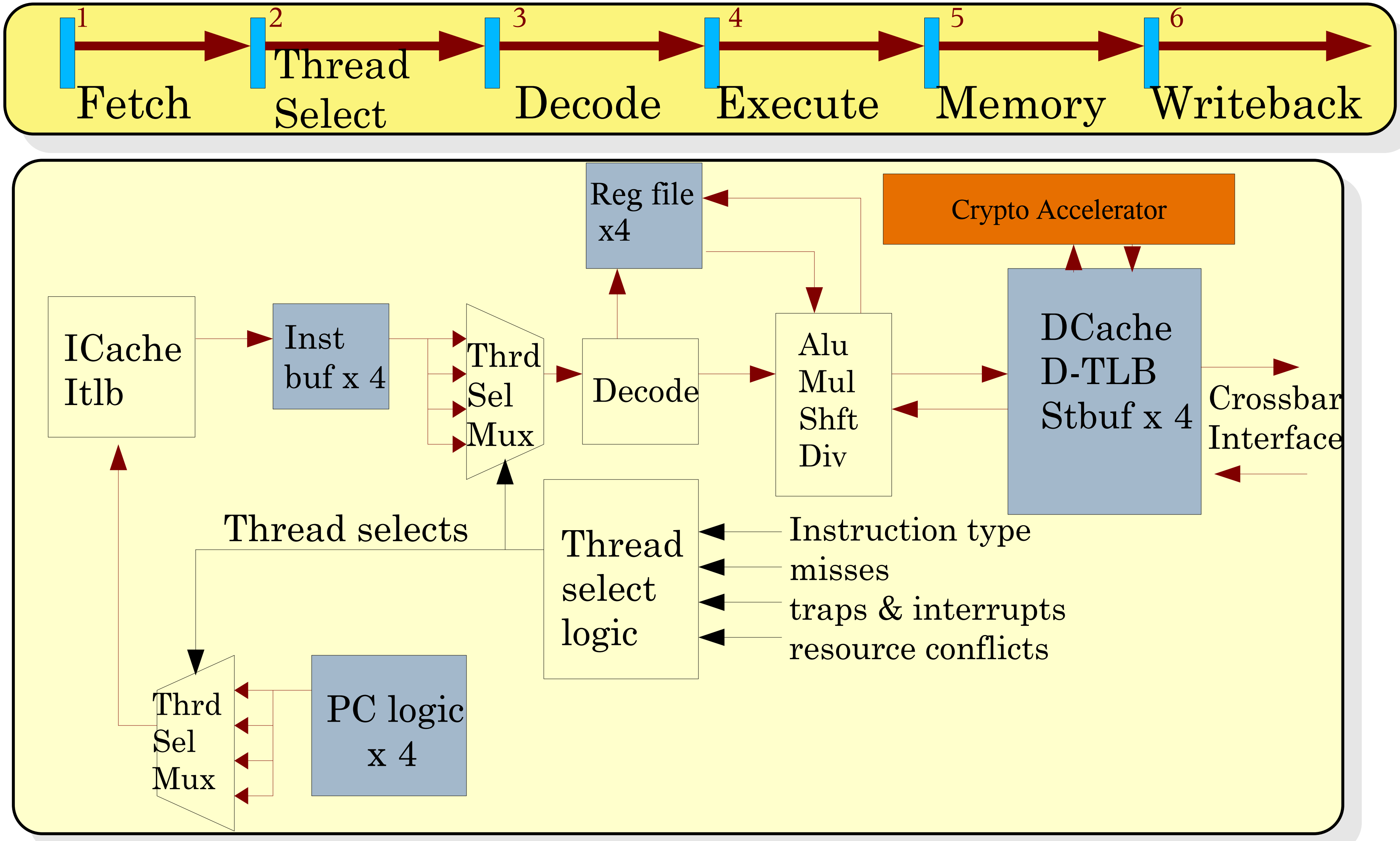
For reference: in 90nm technology, included 8 cores, 32 threads, and only dissipate 70 W

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

OpenSPARC T2

A Highly Threaded
Open Source
Server-on-a-Chip
64 Threads in “Huron” system

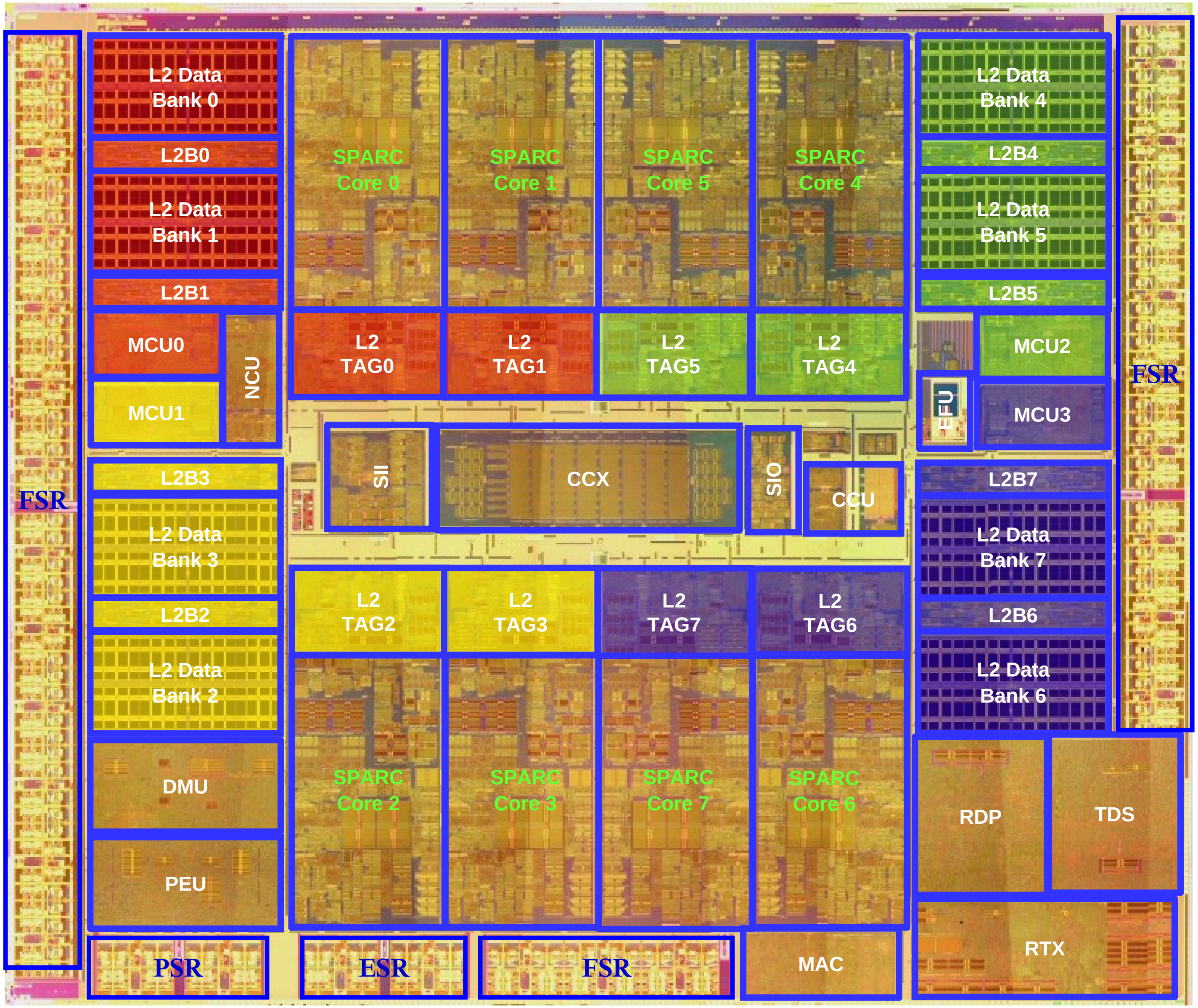
OpenSPARC T2 Plus

T2 + Coherence support
for upto 2+ sockets in a system
128 Threads in “Maramba”

OpenSPARC T2 Chip Goals

- Double throughput versus OpenSPARC T1
  - Doubling cores versus increasing threads per core
  - Utilization of execution units
- Improve throughput / watt
- Improve single-thread performance
- Improve floating-point performance
- Maintain SPARC binary compatibility

UltraSPARC T2 Overview



UltraSPARC T2 Die Photo

- 8 SPARC cores, 8 threads each, 64 threads total
- Shared 4MB L2, 8 banks, 16 way associative
- Four dual-channel FBDIMM memory controllers
- Full 8x9 crossbar connects cores to L2 banks / SIU and vice versa
- SIU connects I/O to memory
- T2+ removed 2 MCUs and NIU and adds SMP Coherency

OpenSPARC T1 to T2 Core Changes

- Increase threads from 4 to 8 in each core
- Increase execution units from 1 to 2 in each core
- Floating-point and Graphics Unit in each core
- New pipe stage: pick
  - Choose 2 threads out of 8 to execute each cycle
- Instruction buffers after L1 instruction cache for each thread
- Increase set associativity of L1 instruction cache to 8
- Increase size of fully associative DTLB from 64 to 128 entries
- Hardware tablewalk for ITLB and DTLB misses
- Speculate branches not taken

OpenSPARC T1 to T2 Chip Changes

- Increase L2 banks from 4 to 8
  - 15 percent performance loss with only 4 banks and 64 threads
- FBDIMM memory interface replaces DDR2
  - Saves pins
  - Improved bandwidth
    - 42 GB/sec read
    - 21 GB/sec write
  - Improved capacity (512 GB)
- RAS changes (to match T1 FIT rate)

Core Power Management

- Minimal speculation
  - > Next sequential I$ line prefetch
  - > Predict branches not-taken
  - > Predict loads hit in D$
  - > Pick independent instructions after loads
  - > Hardware tablewalk search control
- Extensive clock gating
  - > Datapath
  - > Control blocks
  - > Arrays
- External power throttling
  - > Add stall cycles at decode stage

Core Reliability and Serviceability

- Extensive RAS features
  - Parity-protection on I$, D$ tags and data, ITLB, DTLB CAM and data, store buffer address
  - ECC on integer RF, floating-point RF, store buffer data, trap stack, other internal arrays
- Combination of hardware and software correction flows
  - Hardware re-fetch for I$, D$
  - ECC inside the core is corrected by software

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

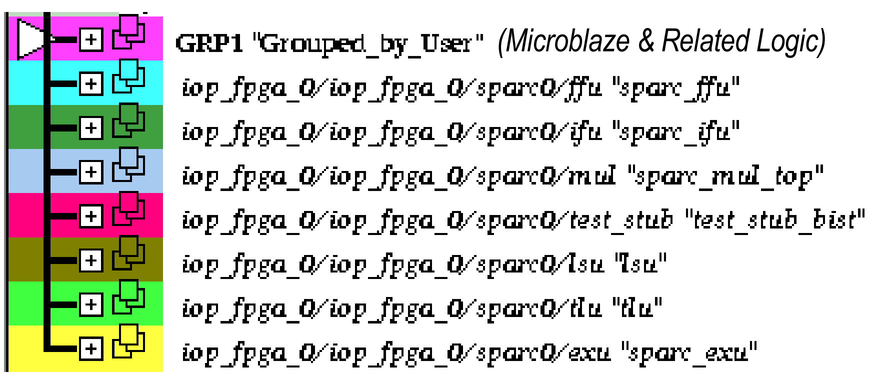
- **Initial version released May 2006**
  (on OpenSparc.net website)
  - > full 8-core, 32-thread
  - > First-cut implementation;
    not yet optimized for Area/Timing
  - > Synplicity scripts for Xilinx/Altera FPGAs
- **Reduced version released Mar 2007 – Release 1.4**
  - > Single-core, single-thread
  - > Reduced size TLB
  - > Optimizations for Area

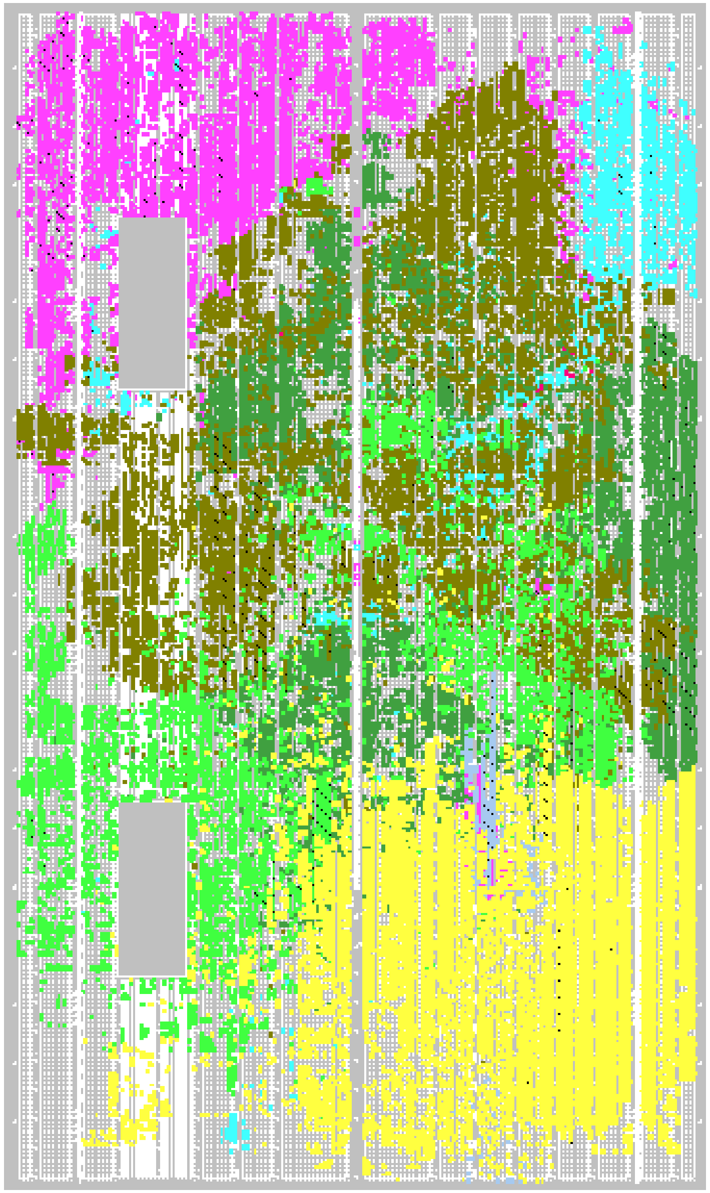
OpenSPARC FPGA Implementation

- **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)

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



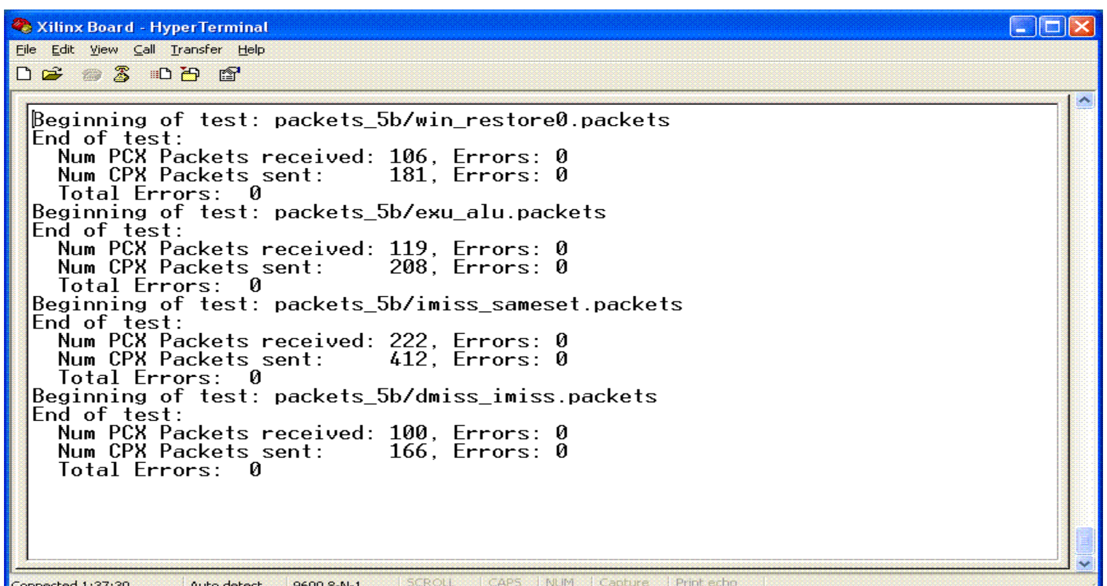


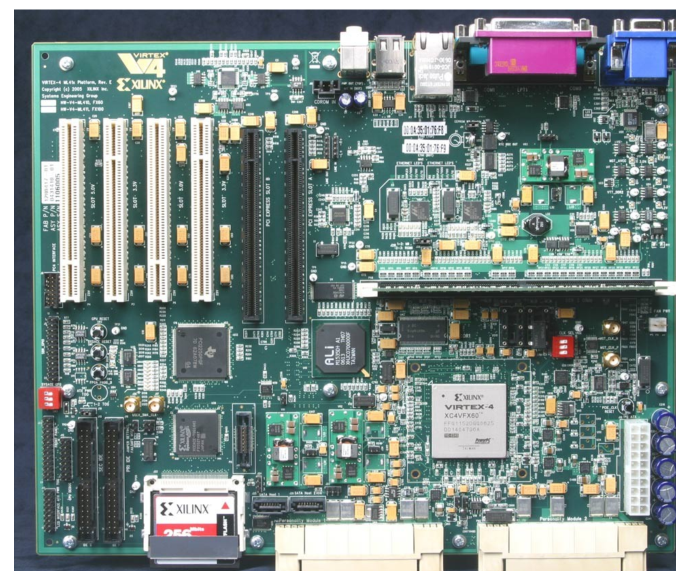
Preliminary Virtex5 Results

- 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

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





Operating Systems

Solaris on UltraSPARC T1

- Solaris 10 (and beyond) run on UltraSPARC T1&T2
- 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

*BSD on OpenSPARC T1

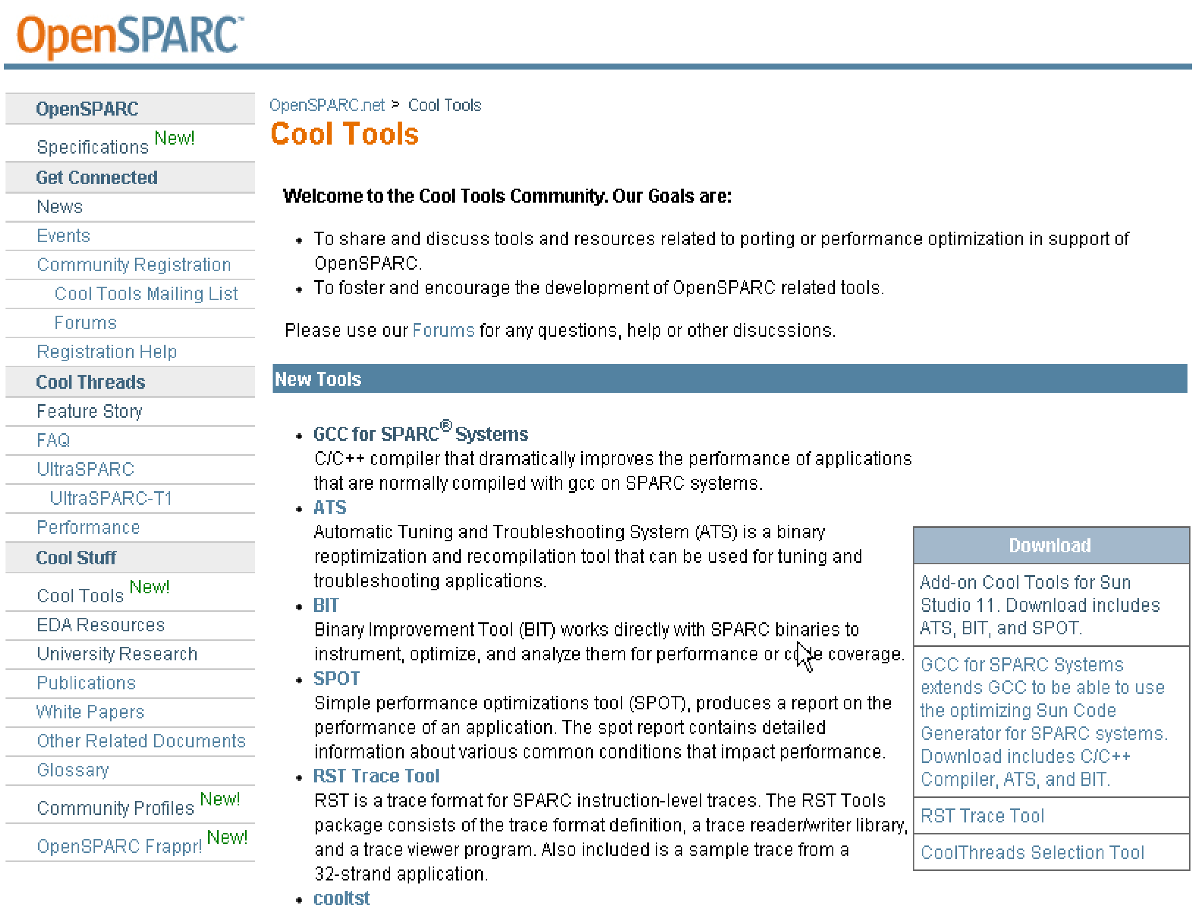
- FreeBSD port for UltraSPARC T1 announced Nov 2006

- 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 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

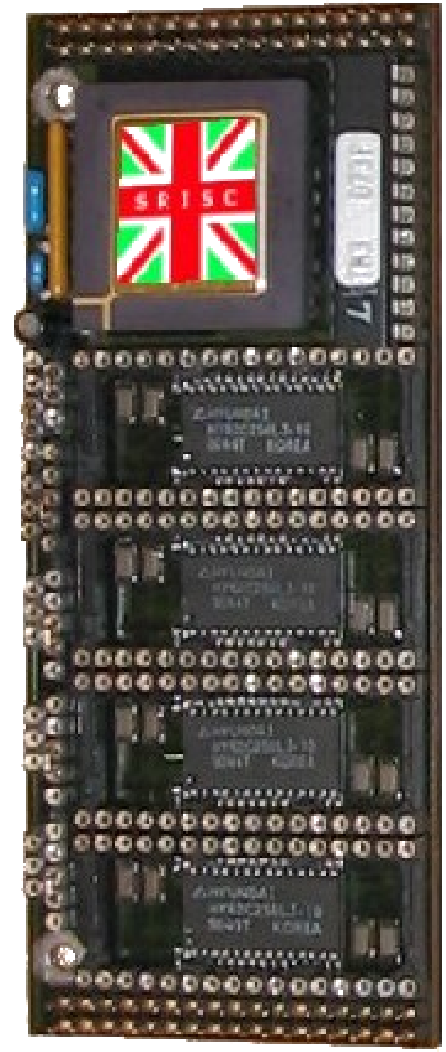
OpenSolaris,
Linux, BSD variants,
Embedded OSs

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
  - > Upgraded to v1.5 of T1 & FPGA downloads available
- 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

OpenSPARC Contest

- The OpenSPARC Community Innovation Awards Contest
  - > Part of Sun's **$1 Million** Open Source Community Innovation Awards Program,
  - > **$175,000** of the $1 Million total prize.
- The OpenSPARC Contest awards categories and award amounts are as follows:

  * A. Grand Prize: **$35,000** (+ **$20K** for category award)

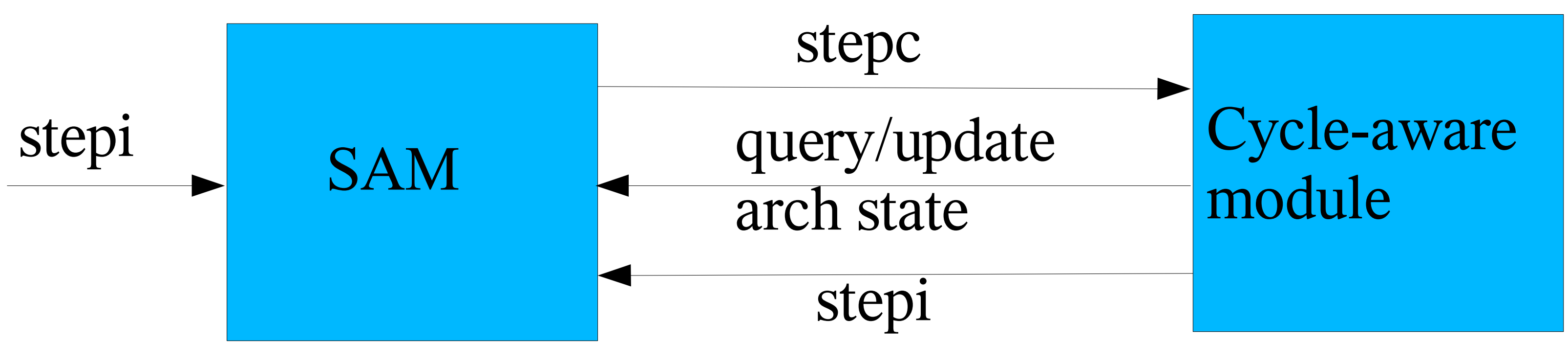
  * B. First Prizes: ($20,000 each category)
- Please read details on www.opensparc.net

OpenSPARC Arch Tools Download

- SAM: SPARC Architecture Model: instruction-accurate SPARC full-system simulator
- SAS/NAS: instruction-accurate SPARC arch. Simulator
- Rstracer: a loadable trace module
- Binary images for simulation: Solaris 10, Hypervisor, OBP, etc
- Legion: SPARC full-system simulator for firmware and software development
- Hypervisor source code
- Documentation

Correlate With Cycle-Aware Module

- Use SAM's loadable module mechanism
- Create callback functions between modules
- Cycle-aware module maintains cycle-related state



Virtualization:

The sun4v Operating Environment

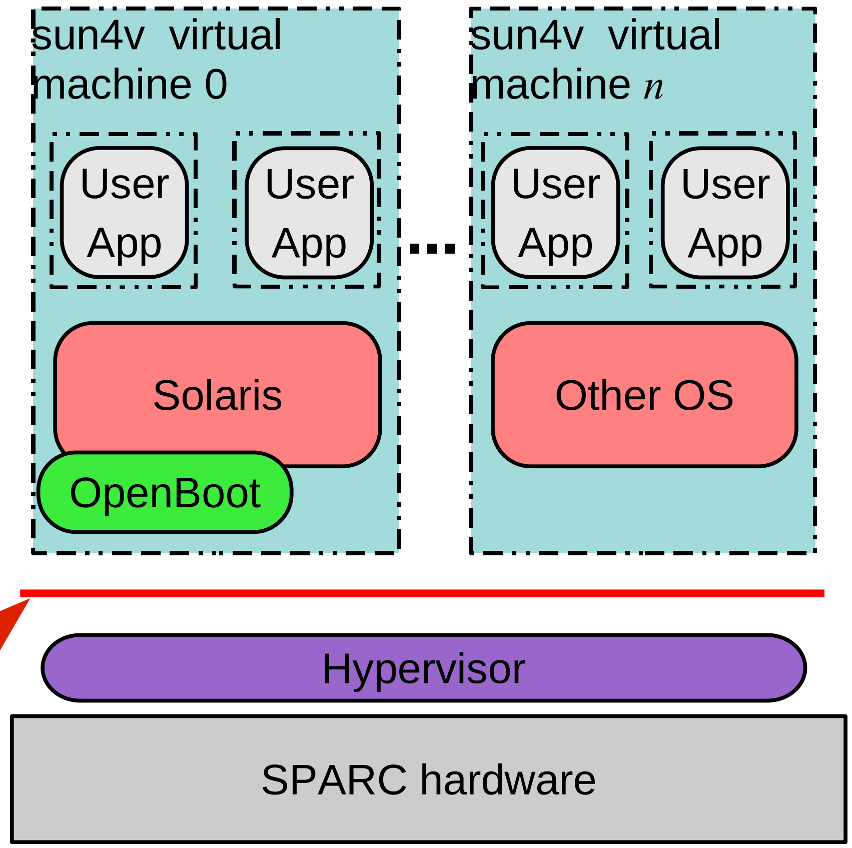
(or)

“Your OS on the T1 Hypervisor”

Virtual Machine for SPARC

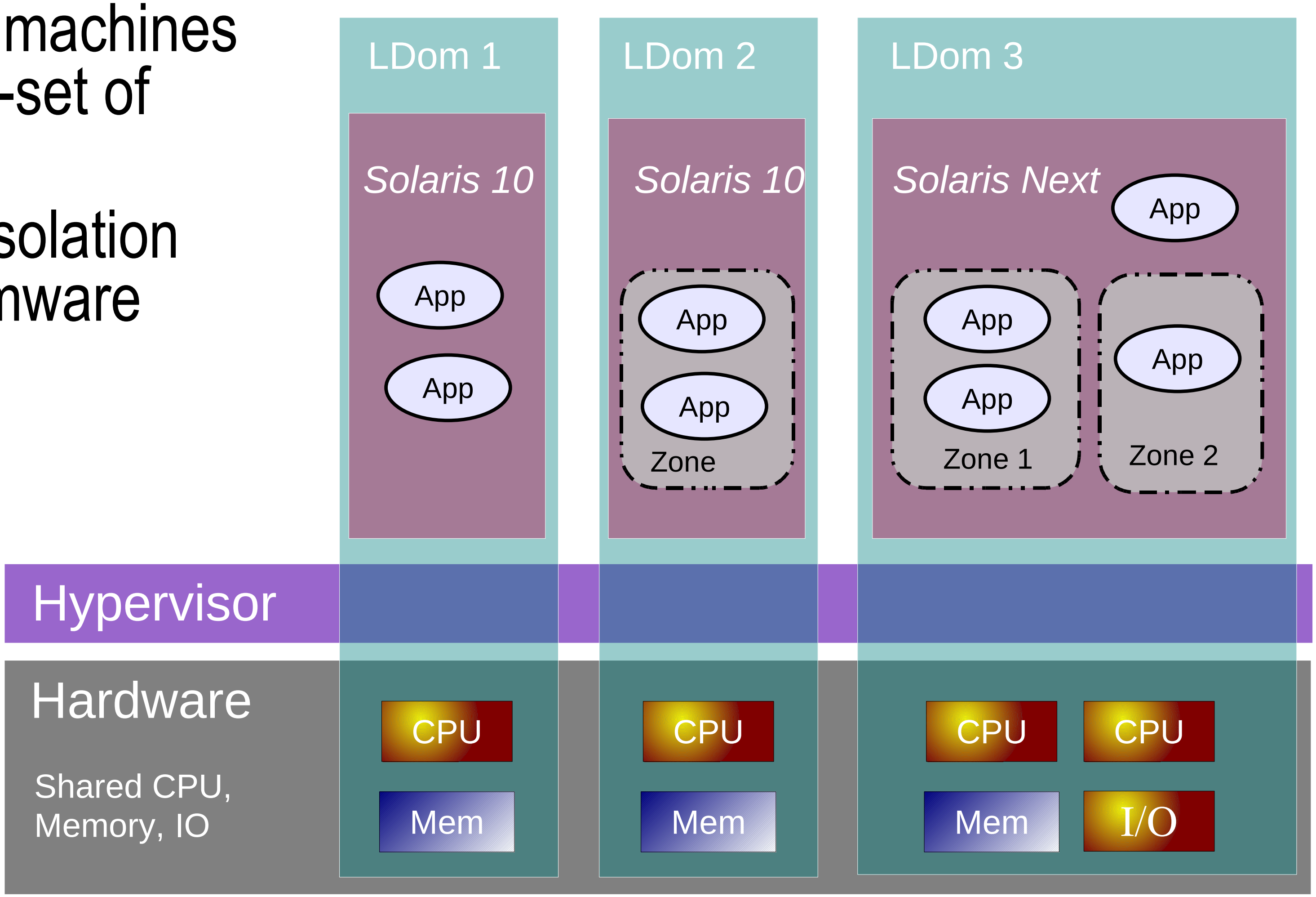
- Thin software layer between OS and platform hardware
- Para-virtualised OS
- Hypervisor + sun4v interface
  - Virtualises machine HW and isolates OS from register-level
  - Delivered with platform not OS
  - Not itself an OS

stable interface "**sun4v**"



Logical Domains

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



Why Hyperprivileged Mode?

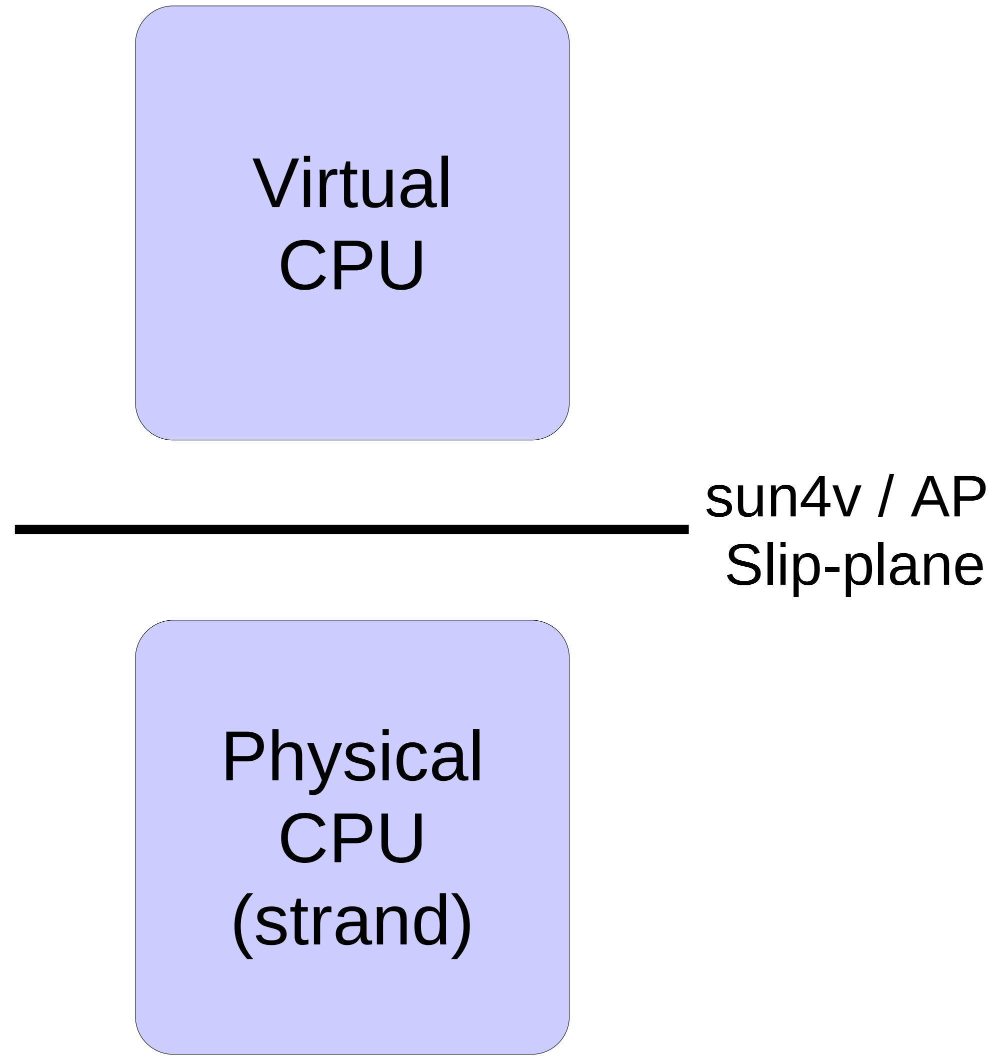
- Allows running multiple simultaneous guest OSs
  - > (and/or multiple versions of the same OS)
- Allows running older OS (that uses hypervisor API) on newer hardware, without need to port the OS
- Simplifies OS ports (Linux in 2 months!)
- Allows implementation of logical domains (LDOMs)
- Allows *virtualization*

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)

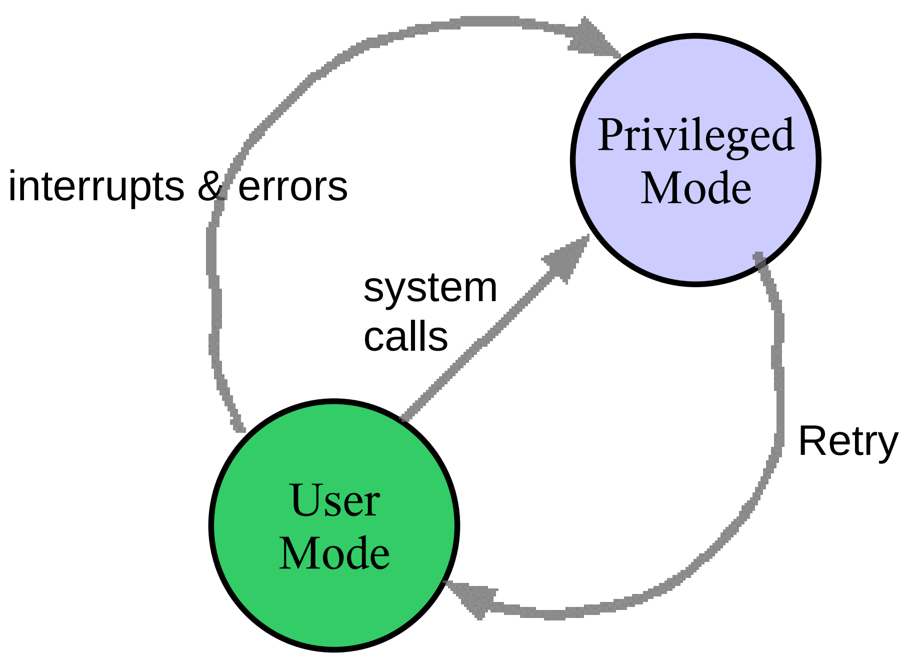
Basic Principles

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

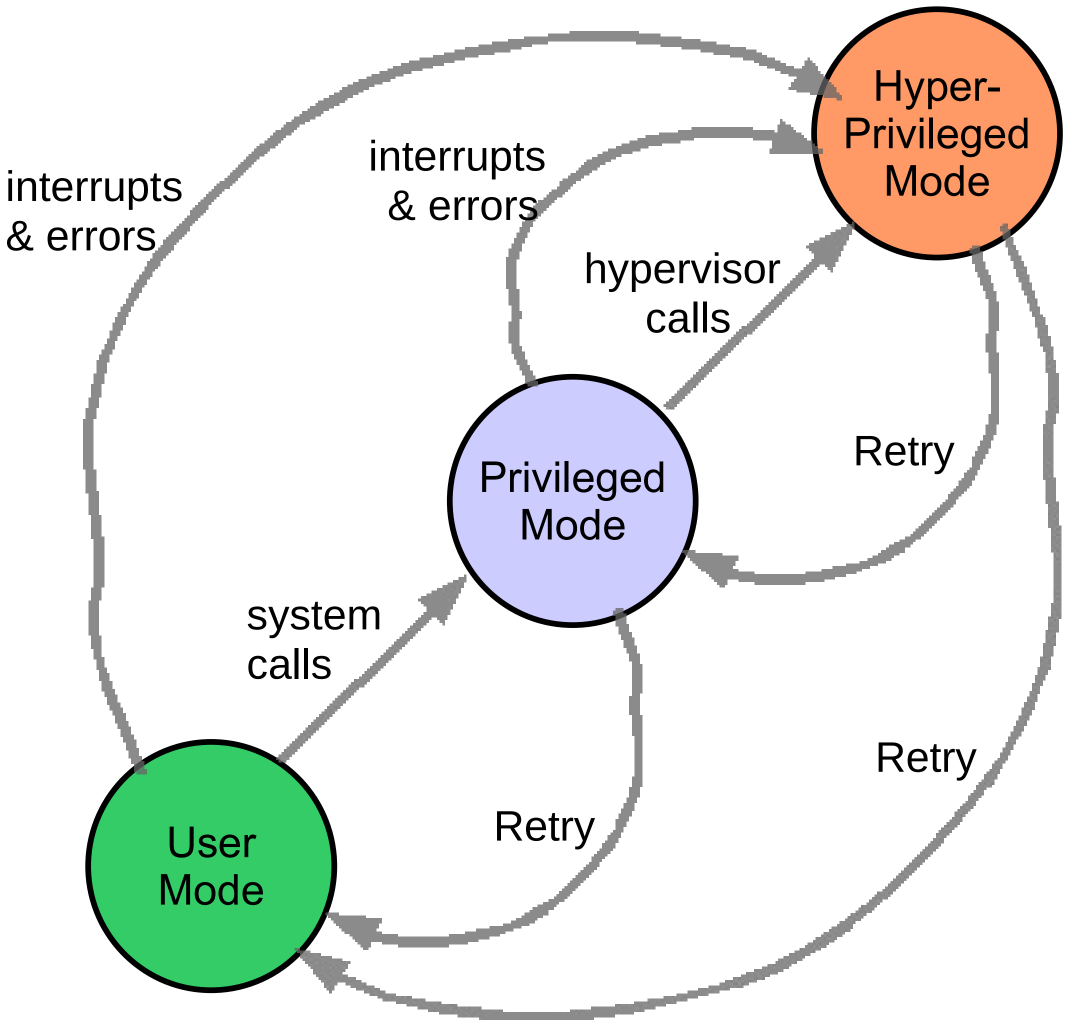


Legacy SPARC execution mode

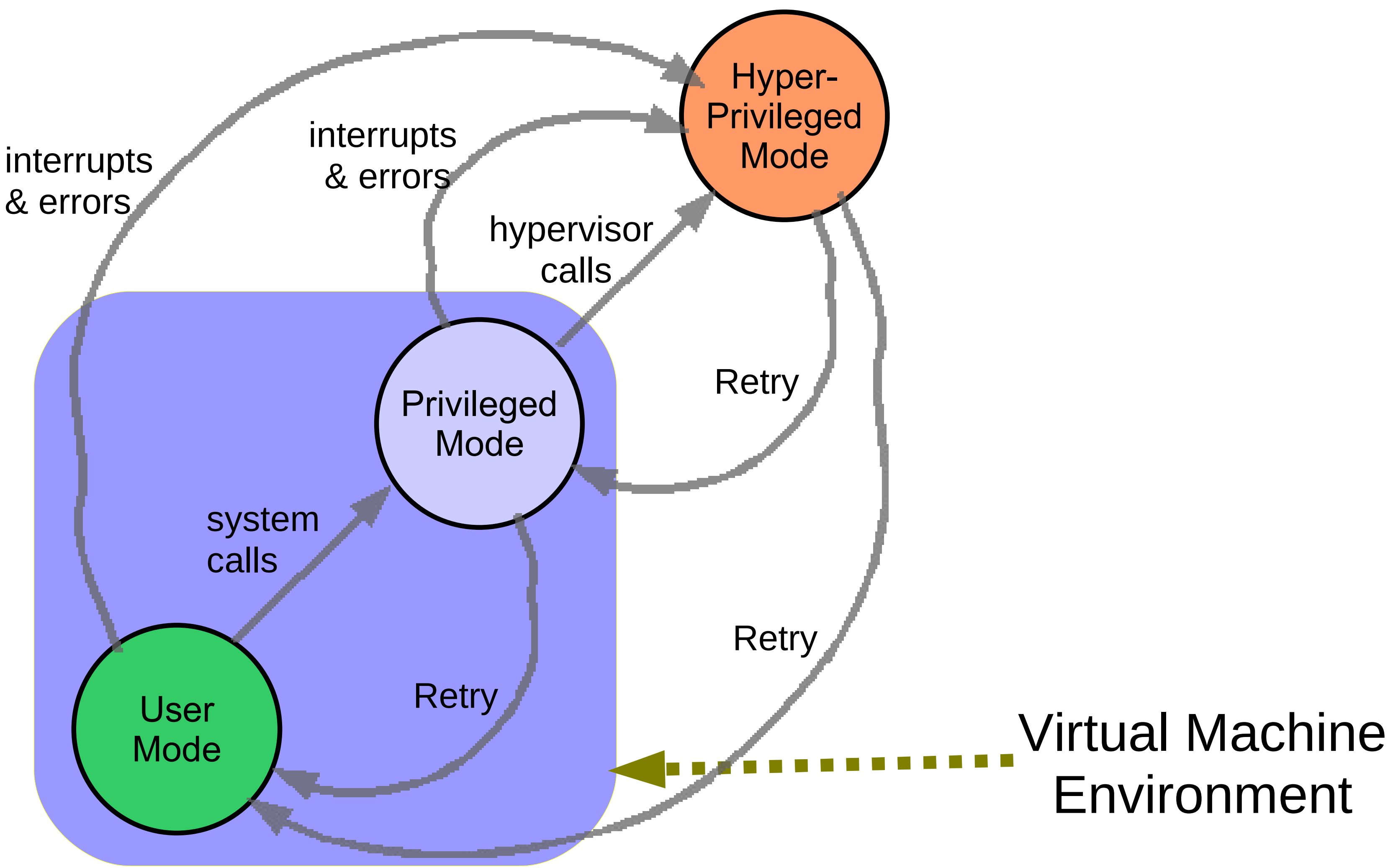
- Older sun4u chips (UltraSPARC I, II, III, IV)



New SPARC Execution mode



New SPARC Execution mode

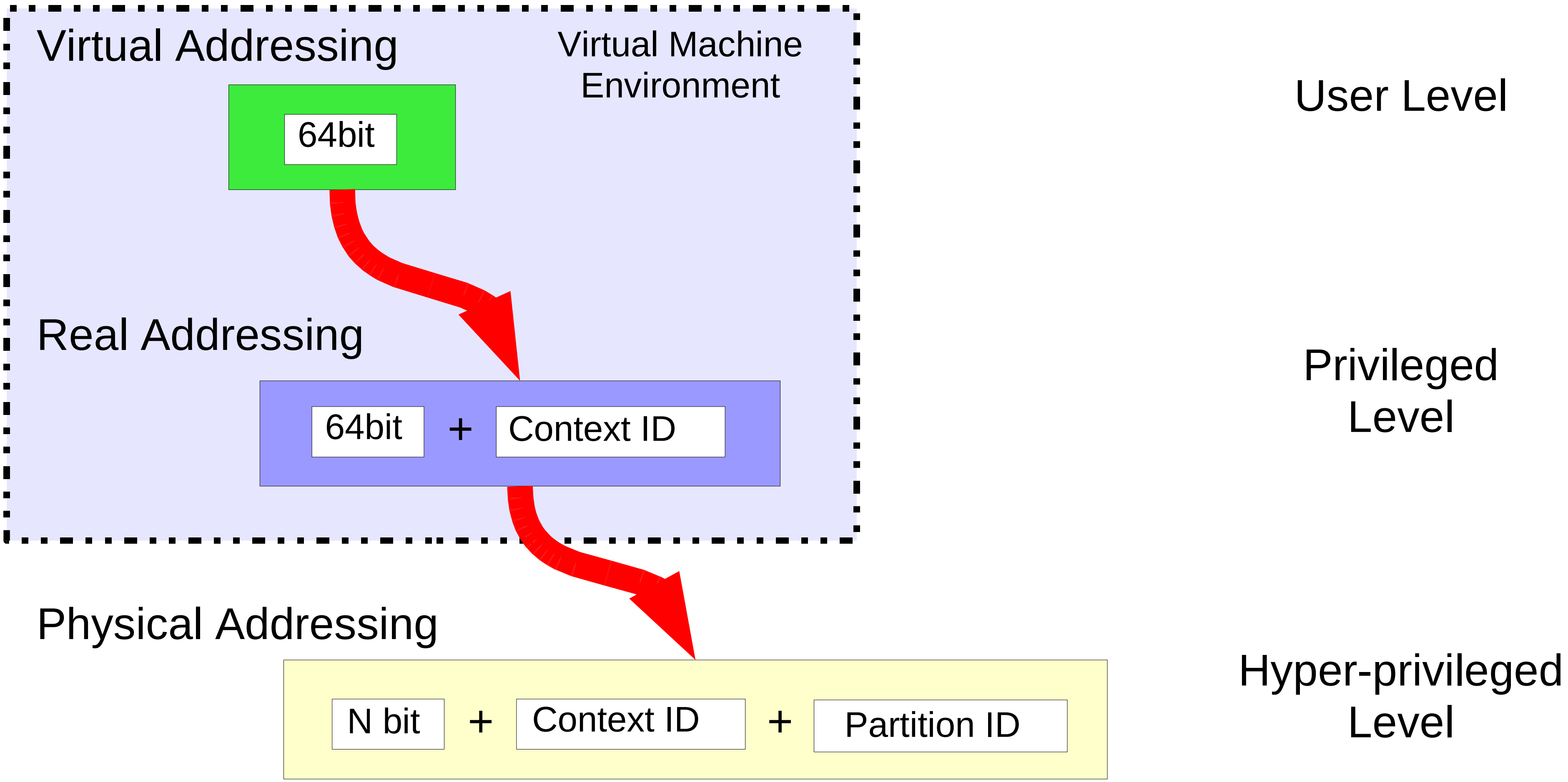


Virtualization on UltraSPARC T1/T2

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

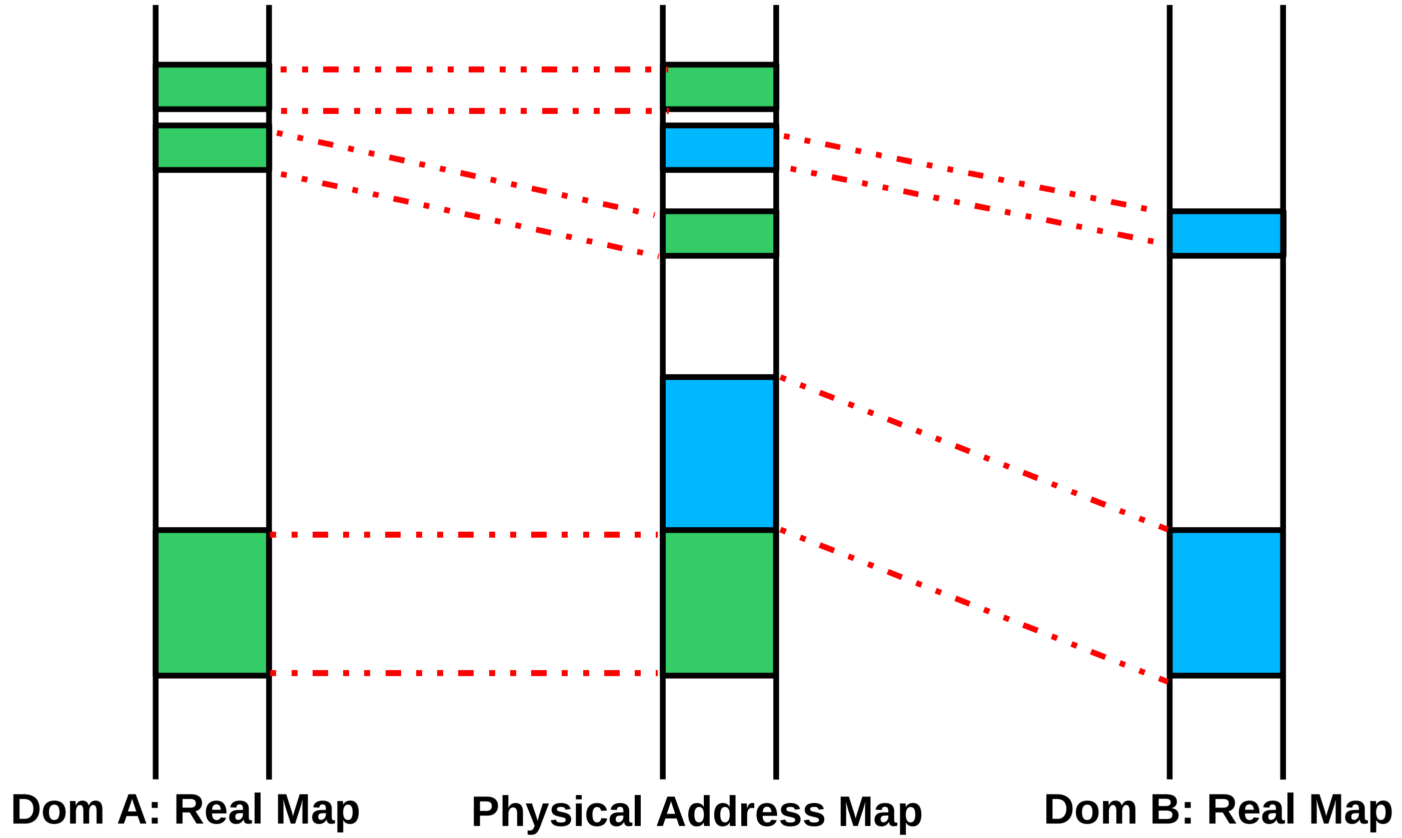
* supervisor = privileged-mode software = operating system
(for example, Solaris, Linux, *BSD, ...)

Translation hierarchy



Address space control

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



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

Virtual I/O devices

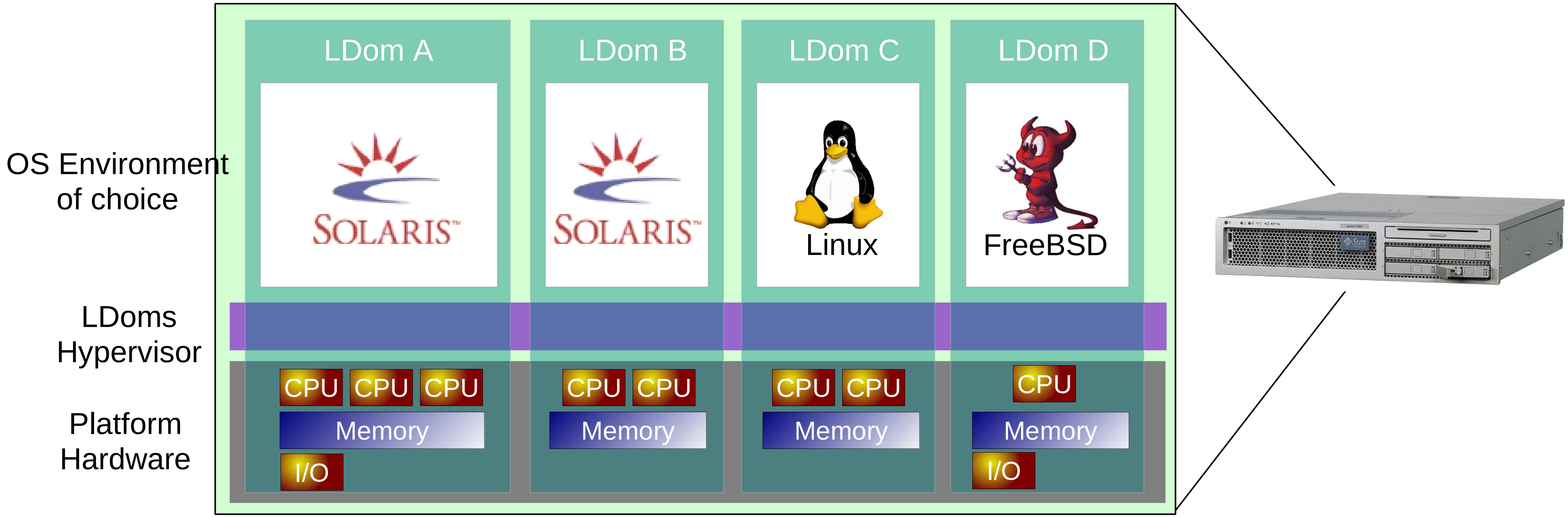
- Provided via Hypervisor
  - e.g. Console - getchar / putchar API calls
  - Hypervisor generates virtual interrupts

Physical I/O devices

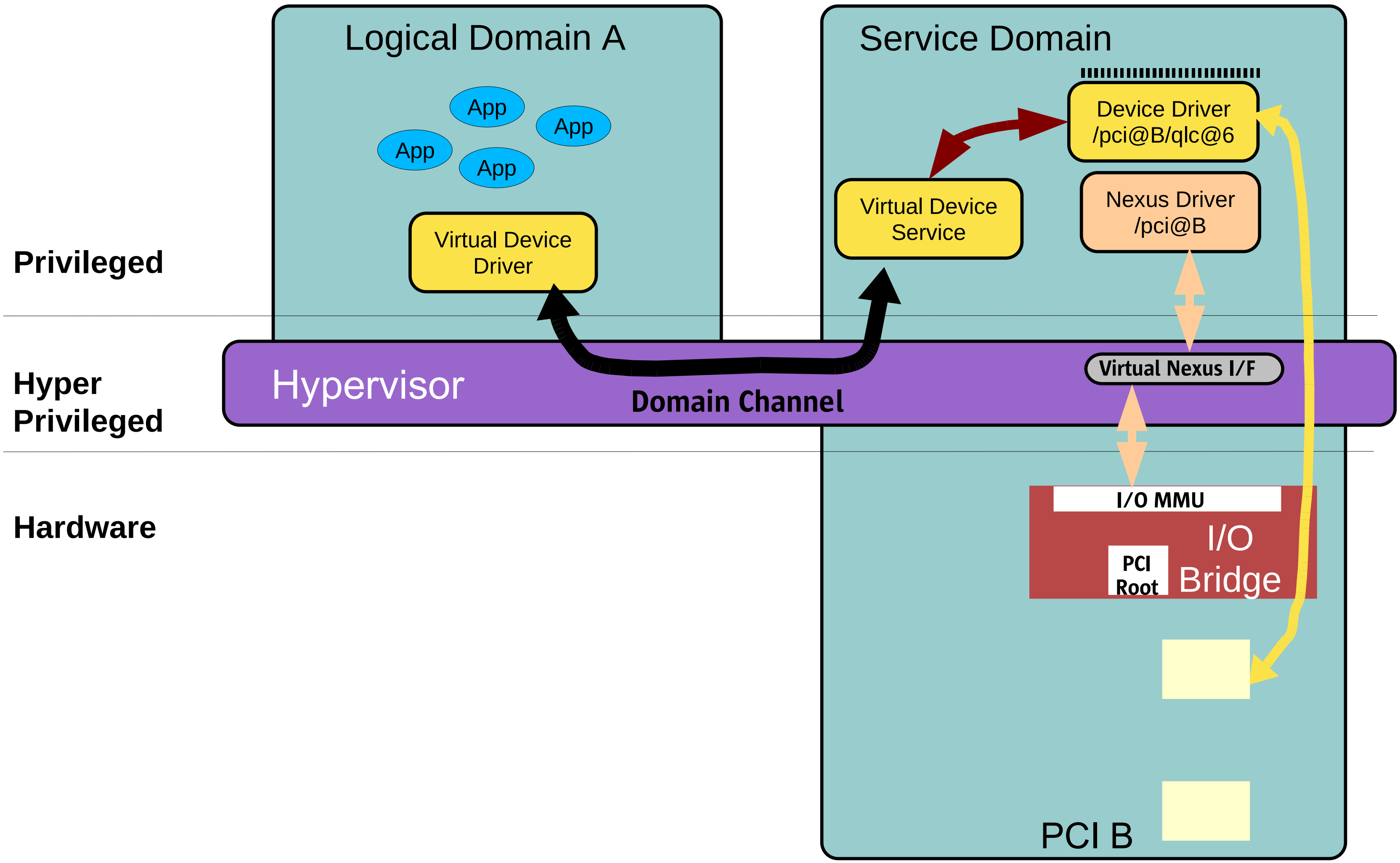
- PCI-Express root complex mapped into real address space of guest domain
- Direct access to device registers
  - OBP probes and configures bus and devices
- I/O Bridge and I/O MMU configuration virtualized by hypervisor APIs
  - Ensures that I/O MMU translations are validated by hypervisor
  - Device interrupts are virtualized for delivery

Logical Domaining Technology

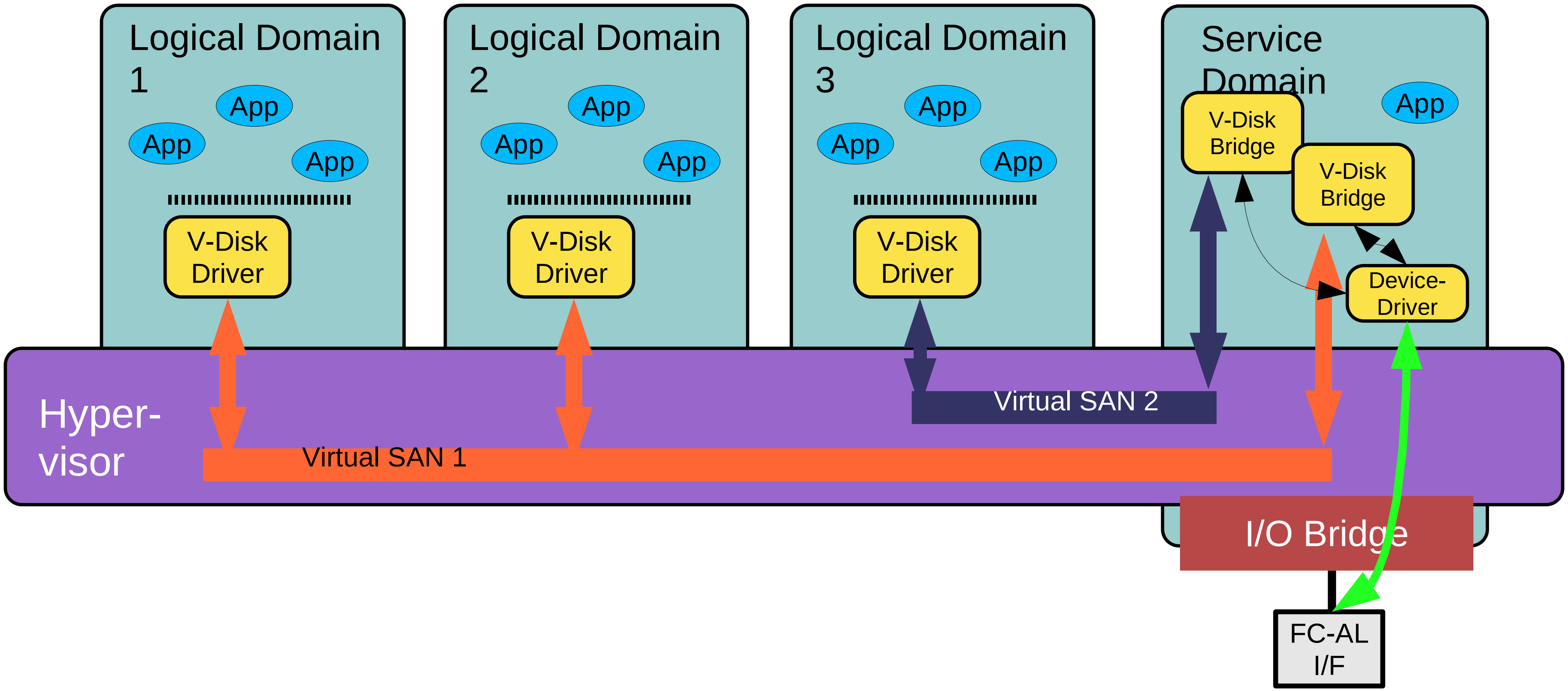
- Virtualization and partitioning of resources
  - > Each domain is a full virtual machine, with a dynamically re-configurable sub-set of machine resources, and its own independing OS instance
  - > Protection & isolation via SPARC hardware and Ldoms firmware



Virtualized I/O

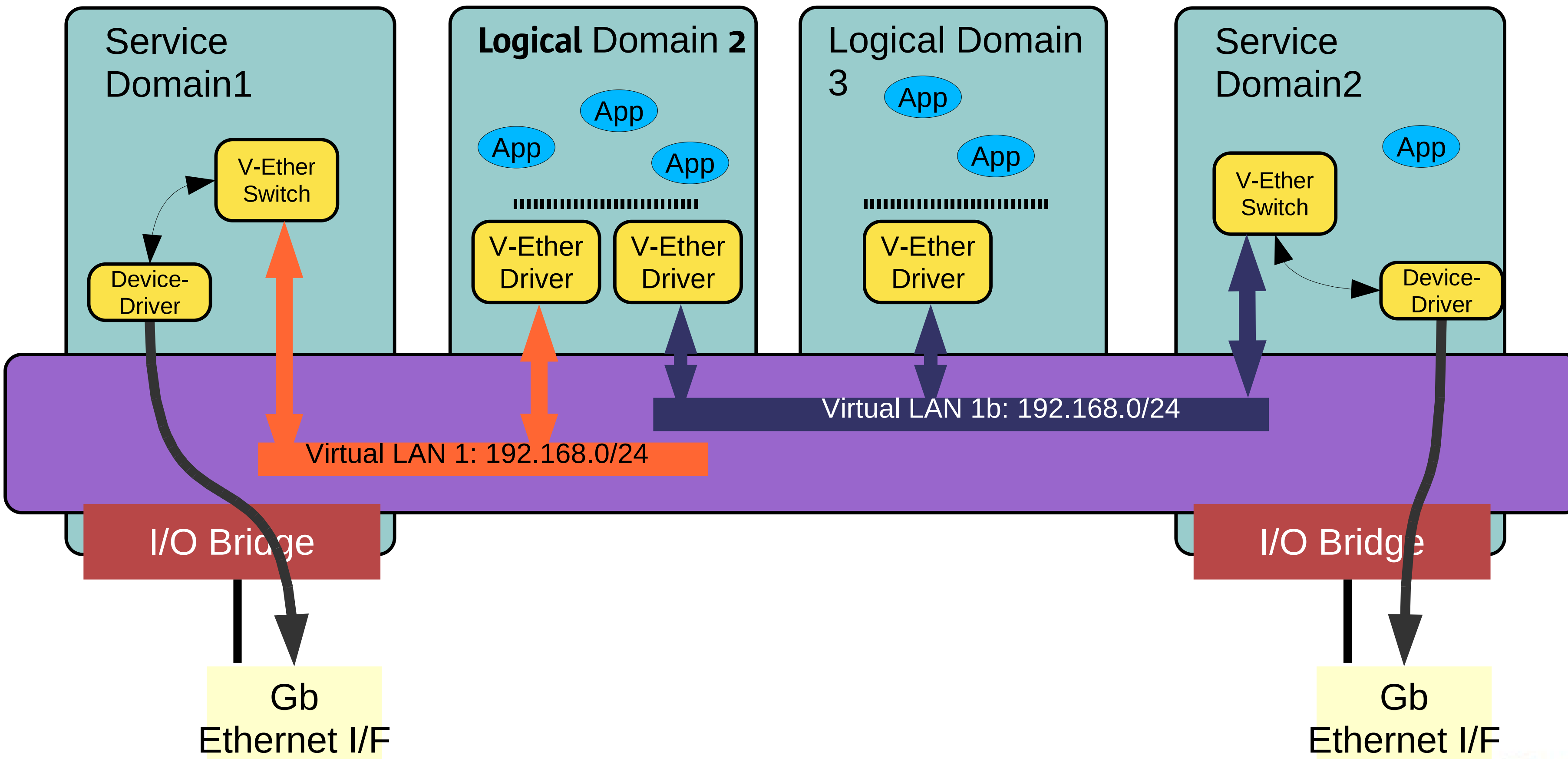


•Virtual (Block) Disk device & server



•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 HyperVisor
  - > 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

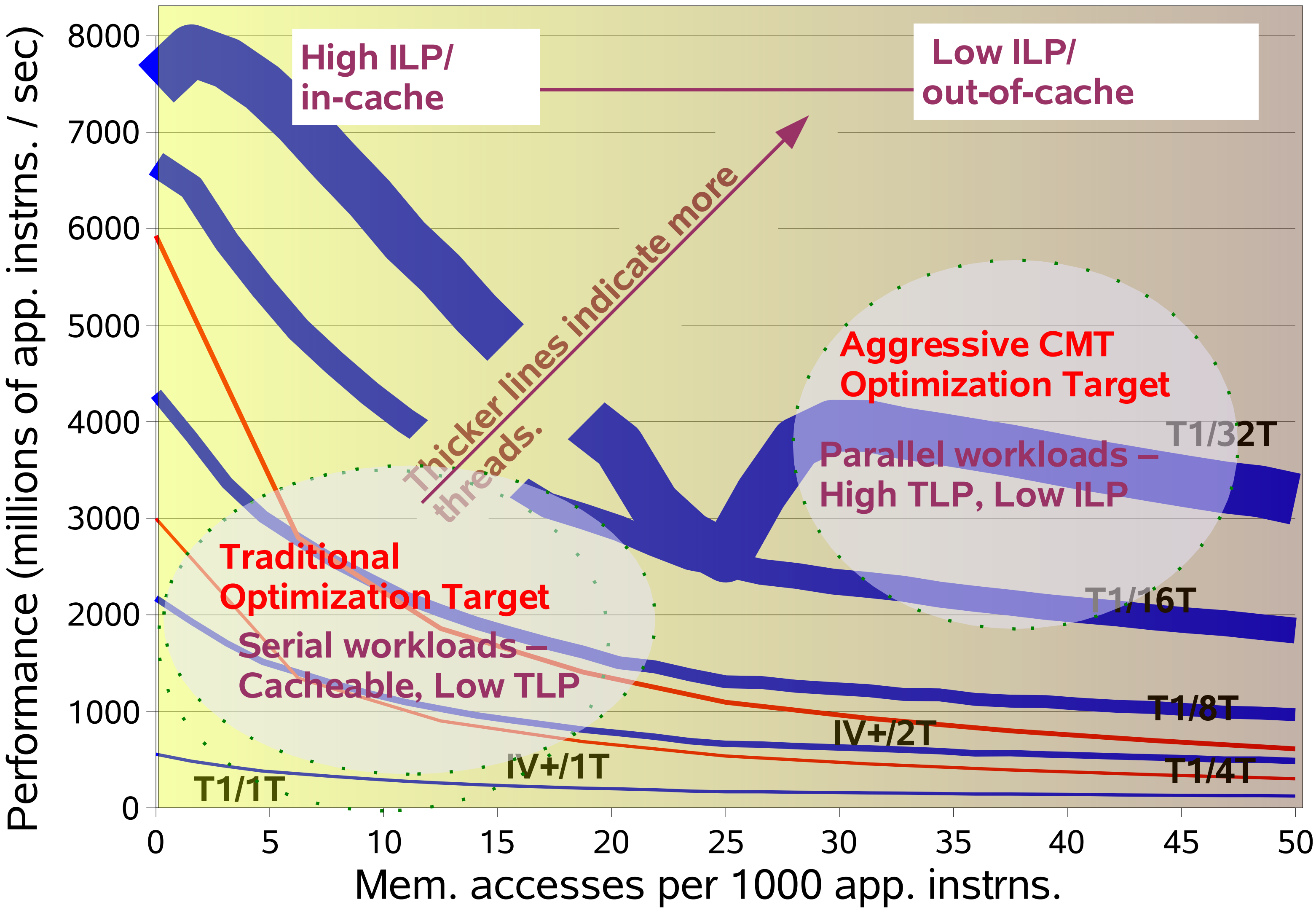
Summary

- 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

OpenSPARC T1

Compiler Optimizations

Traditional vs. Aggressive CMT



Designing for ILP vs. TLP

ILP =Instruction Level Parallization
TLP=Thread Level Parallization

- 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 an IPC of 1-0.25@1.25GHz

Much easier to find

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
- Compiler also provides a "-fast" single switch

```
$ 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.

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

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

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

Simplifying Performance Optimization

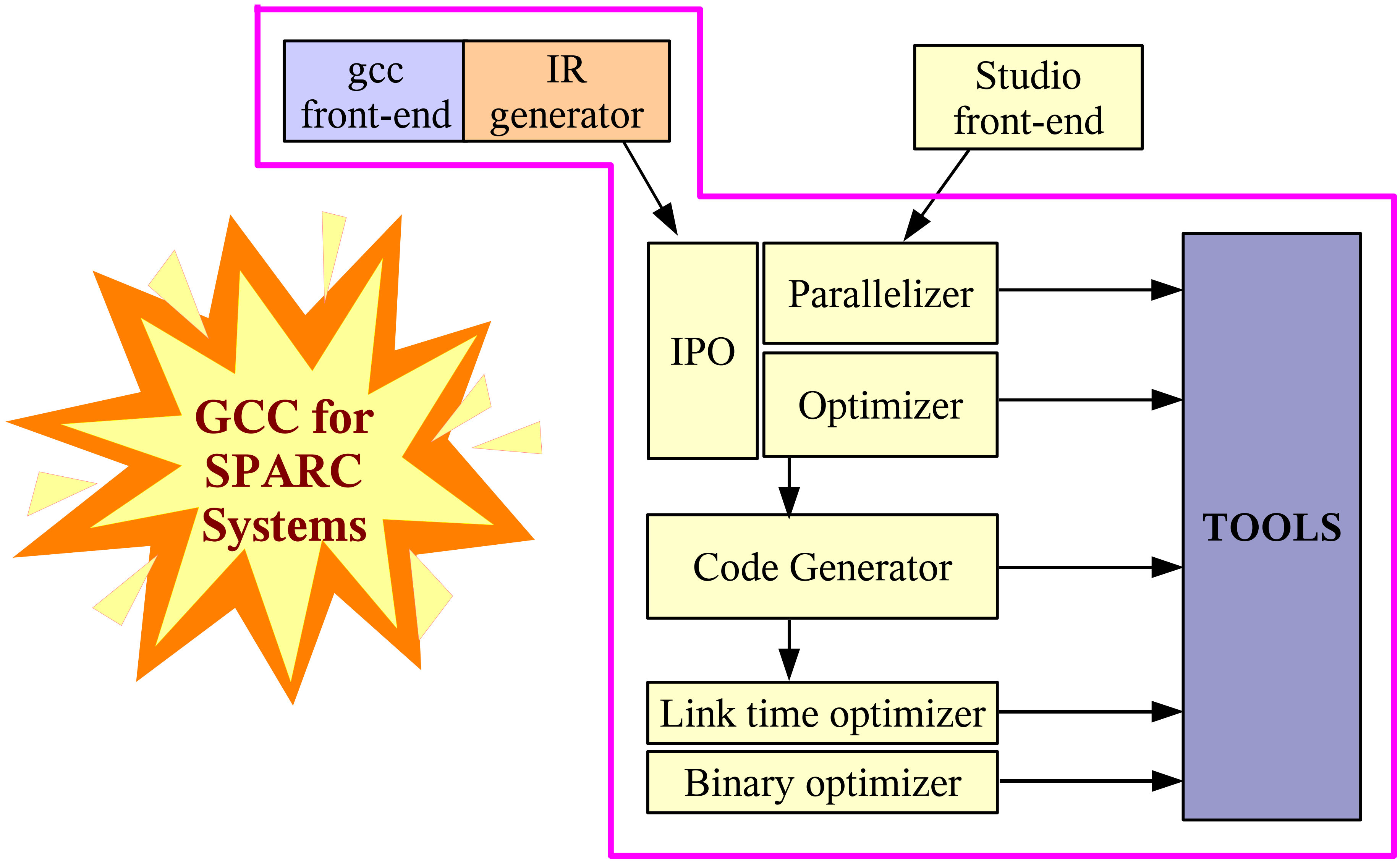
- SPOT – A Simple Performance Optimization Tool
  - Produces a report on a code's execution
  - Exposes common causes of performance loss
  - Very easy to use
- SPOT reports contain hyperlinked profiles
  - Makes it easy to navigate from performance issue to source to assembly
  - For maximum information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

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 GCC users!

Making the Connection



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

OpenSPARC
Community Participation

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)