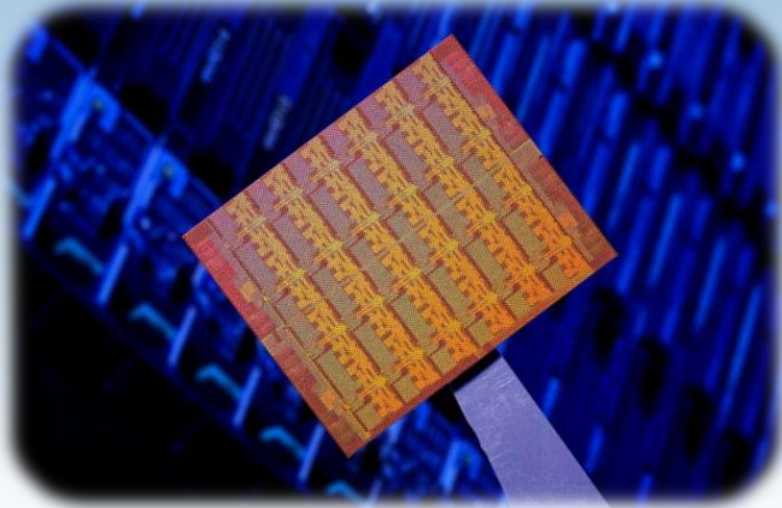


“Single-chip Cloud Computer”

An experimental many-core processor from Intel Labs



Jim Held
Intel Fellow & Director
Tera-scale Computing Research

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

Motivations for SCC

- Many-core processor research
 - High-performance power-efficient fabric
 - Fine-grain power management
 - Message-based programming support
- Parallel Programming research
 - Better support for scale-out model servers
 - > Operating system, communication architecture
 - Scale-out programming model for client
 - > Programming languages, runtimes

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

SCC Architecture and Design Overview

Jason Howard
Advanced Microprocessor Research
Intel Labs



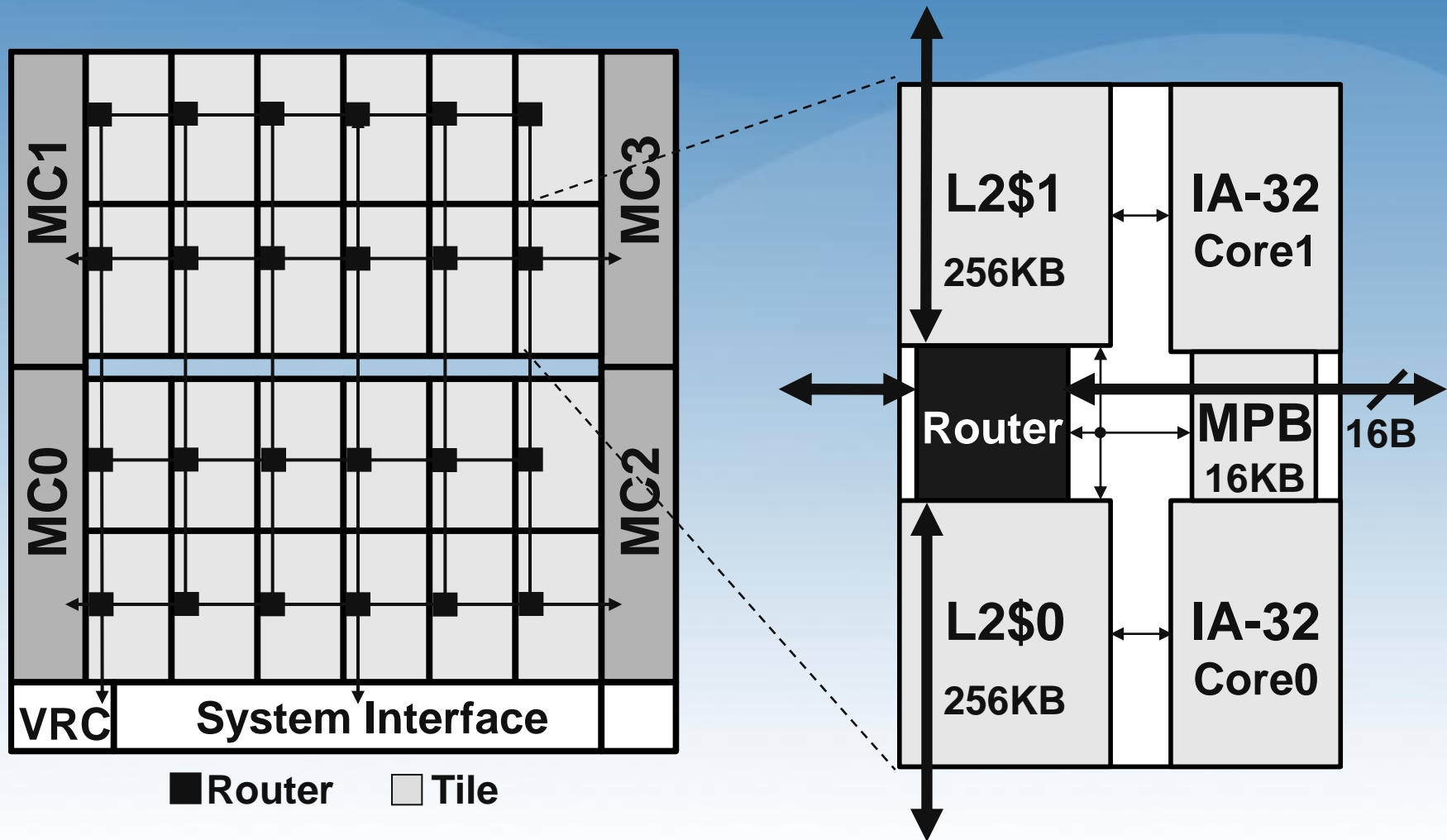
Agenda

- Feature set
- Architecture overview
 - Core
 - Interconnect Fabric
 - Memory model & Message passing
 - I/O and System Overview
- Design Overview
 - Tiled design methodology
 - Clocking
 - Power management
- Results
- Summary

SCC Feature set

- First Si with 48 iA cores on a single die
- Power envelope 125W Core @1GHz, Mesh @2GHz
- Message passing architecture
 - > No coherent shared memory
 - > Proof of Concept for scalable solution for many core
- Next generation 2D mesh interconnect
 - > Bisection B/W 1.5Tb/s to 2Tb/s, avg. power 6W to 12W
- Fine grain dynamic power management
 - > Off-die VRs

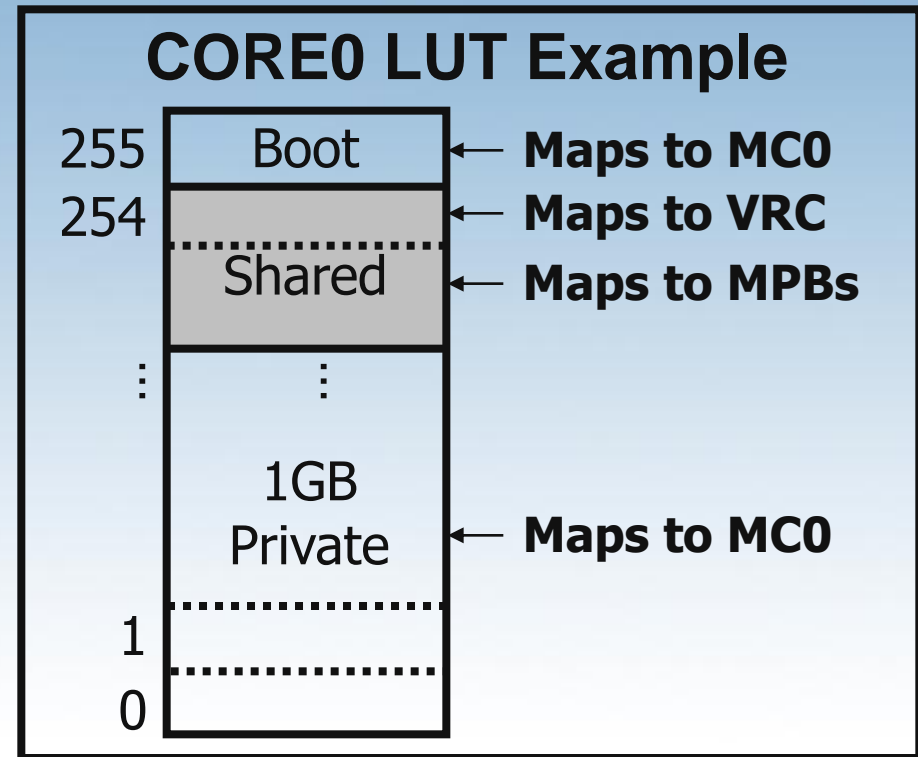
Die Architecture



2 core clusters in 6x4 2-D mesh

Core Memory Management

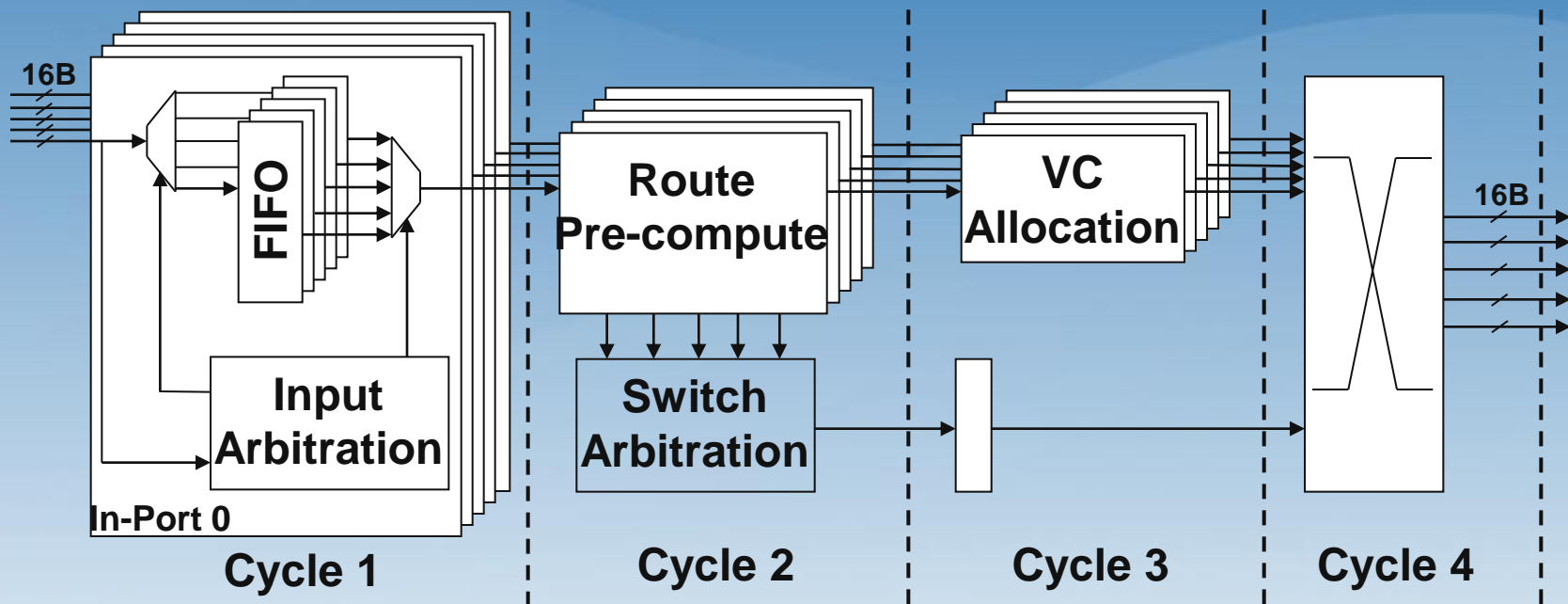
- Core cache coherency is restricted to private memory space
 - Maintaining cache coherency for shared memory space is under software control
- Each core has an address Look Up Table (LUT) extension
 - Provides address translation and routing information
- LUT must fit within the core and memory controller constraints
- LUT boundaries are dynamically programmed



On-Die 2D Mesh

- 16B wide data links + 2B sideband
 - > Target frequency: 2GHz
 - > Bisection bandwidth: 2 Tb/s
 - > Latency: 4 cycles (2ns)
- 2 message classes and 8 VCs
- Low power circuit techniques
 - > Sleep, clock gating, voltage control, low power RF
 - > Low power 5 port crossbar design
- Speculative VC allocation
- Route pre-computation
- Single cycle switch allocation

Router Architecture

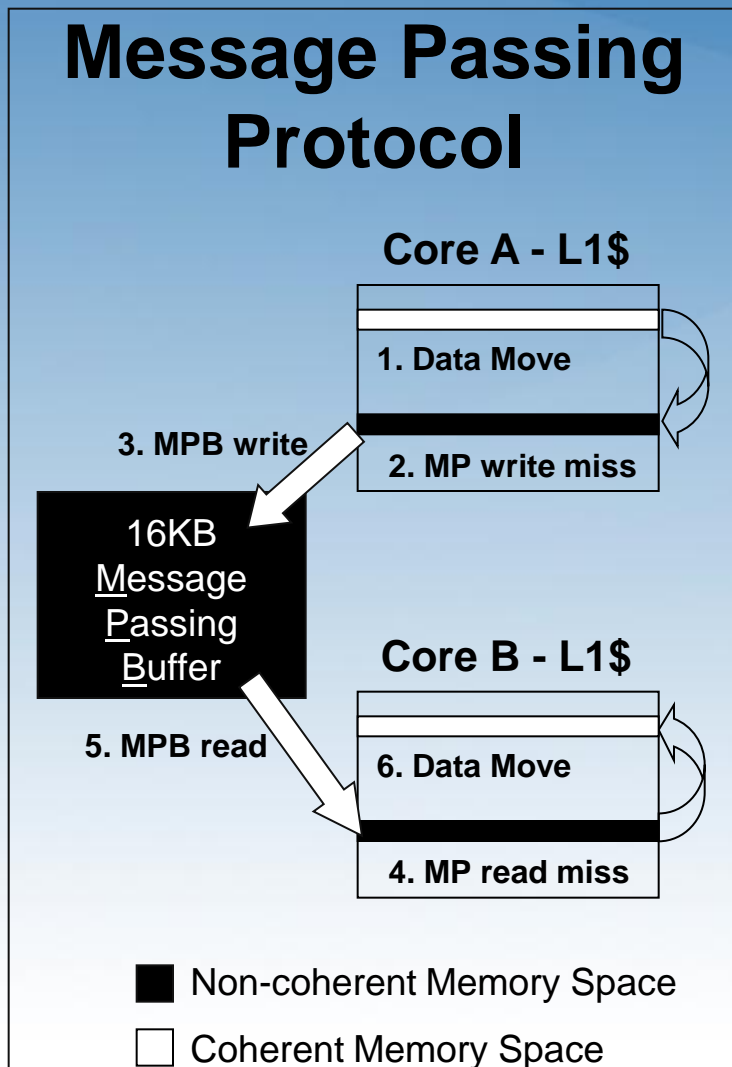


Frequency	2GHz @ 1.1V
Latency	4 cycles
Link Width	16 Bytes
Bandwidth	64GB/s per link
Architecture	8 VCs over 2 MCs
Power Consumption	500mW @ 50°C

Message Passing on SCC

- Message passing is done through shared memory space
- Two classes of shared memory:
 - Off-die, DRAM: Uncachable shared memory ... results in high latency message passing
 - On-die, message passing buffers (MPB) ... low latency message passing
 - > On-die dedicated message buffers placed in each tile to improve message passing performance
 - > Message bandwidth improved to 1 GB/s

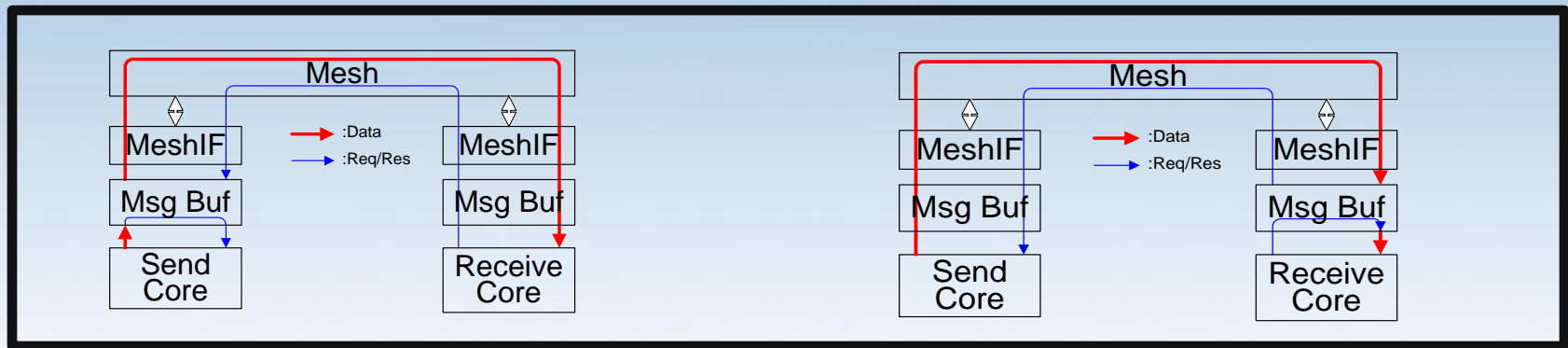
Message Passing Protocol



- Cores communicate through small fast messages
 - L1 to L1 data transfers
 - New Message Passing Data Type (MPDT)
- Message passing Buffer (MPB) – 16KB
 - 1 MPB per tile for 384KB of on-die shared memory
 - MPB size coincides with L1 caches

Dedicated Message Buffers

- Cache line transfers into L1 cache of receiving core implemented through on-die message passing buffers
- Each tile has 16KB MPB
- Part of the shared memory space is statically mapped into MPB in each tile rather than into memory controller
- Messages larger than MPB can still go out to main memory



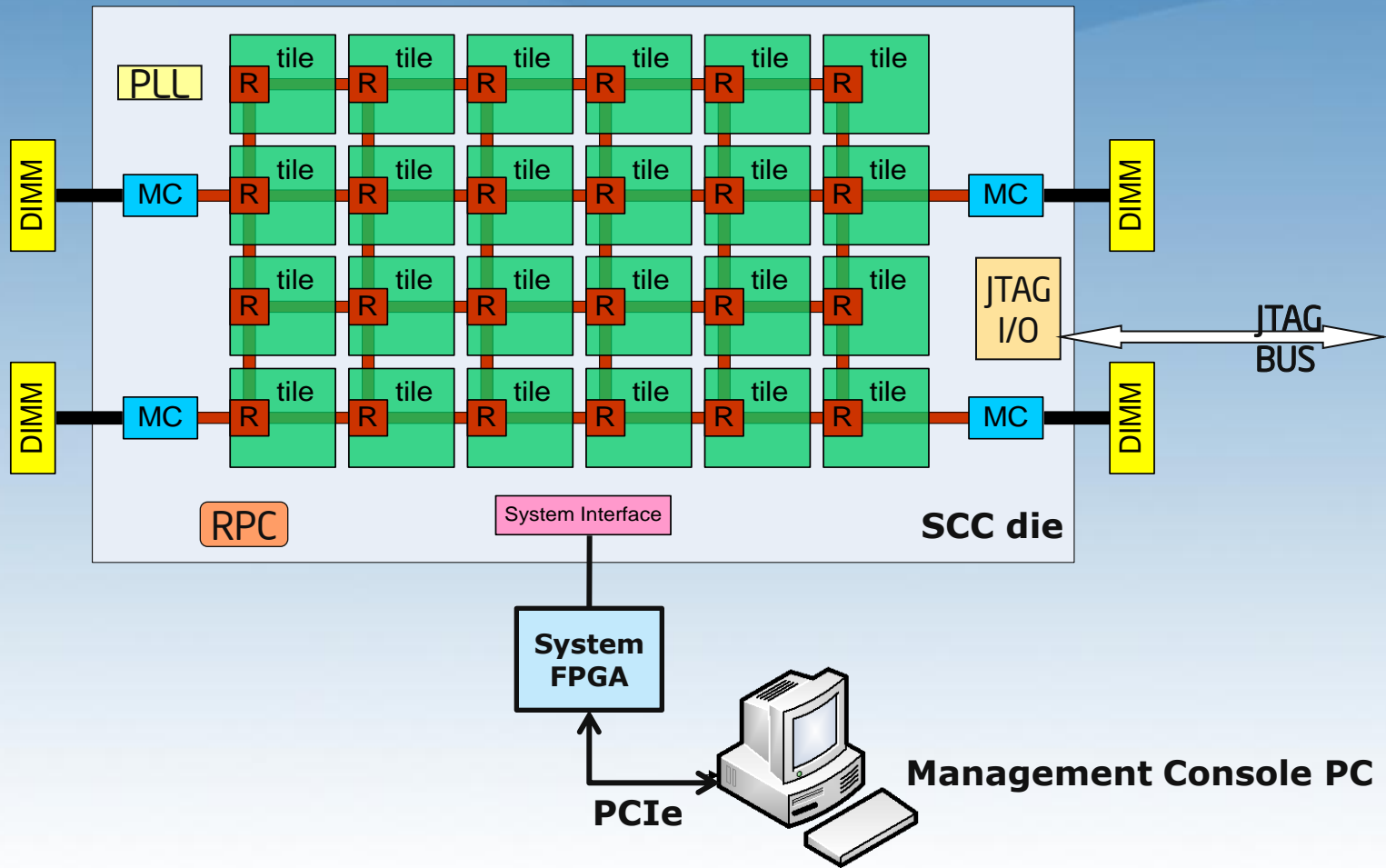
Local write, remote read

Remote write, local read

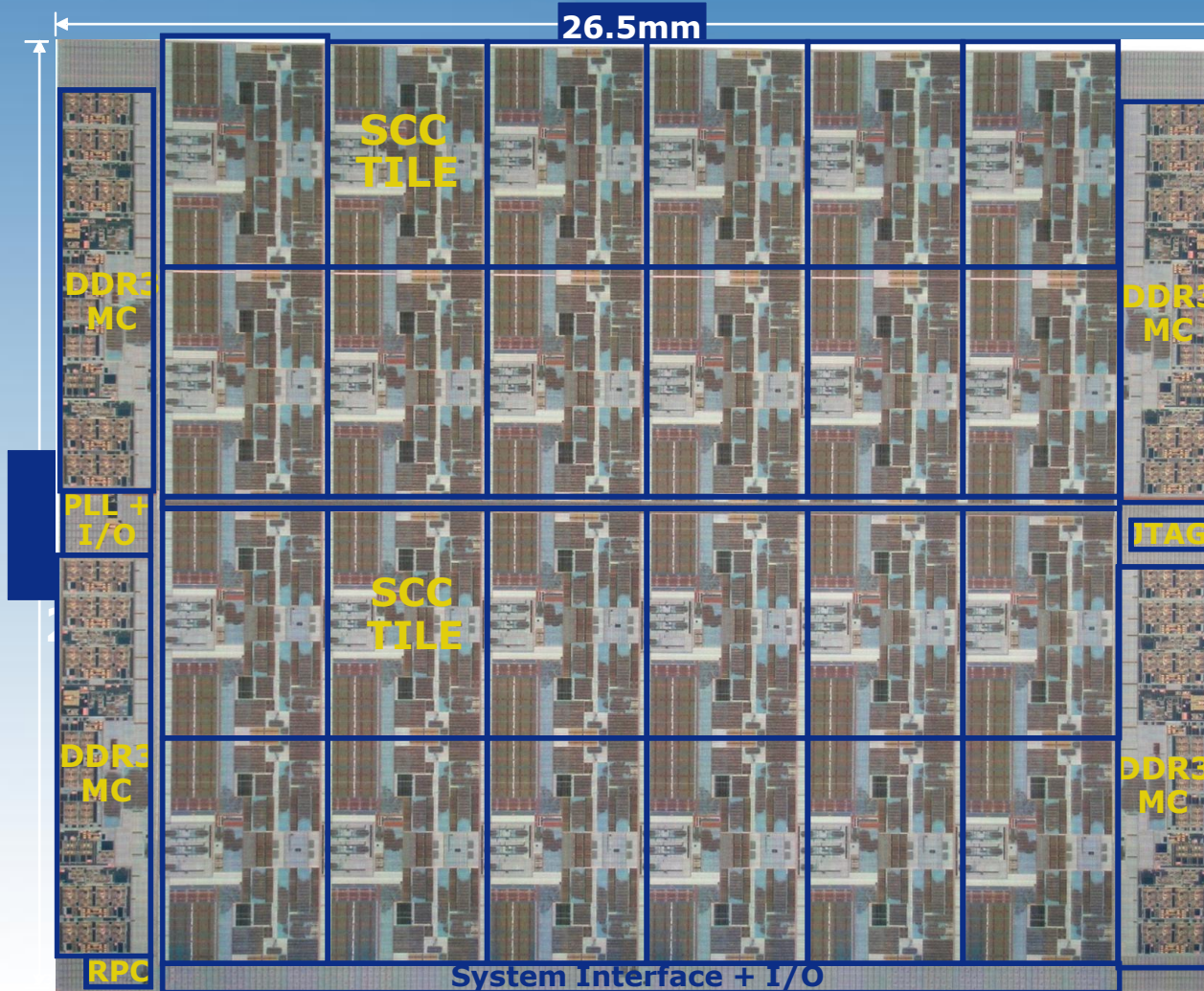
System Interface

- JTAG access to config system while in reset/debug
 - > Done on Power Reset from Management Console PC
 - > Configuring memory controller etc.
 - > Reset cores with default configuration
- Management Console PC can use Mem-mapped registers to modify default behavior
 - > Configuration and voltage control registers
 - > Message passing buffers
 - > Memory mapping
- Preload image and reset rather than PC bootstrap
 - > BIOS & firmware a work in progress

SCC system overview

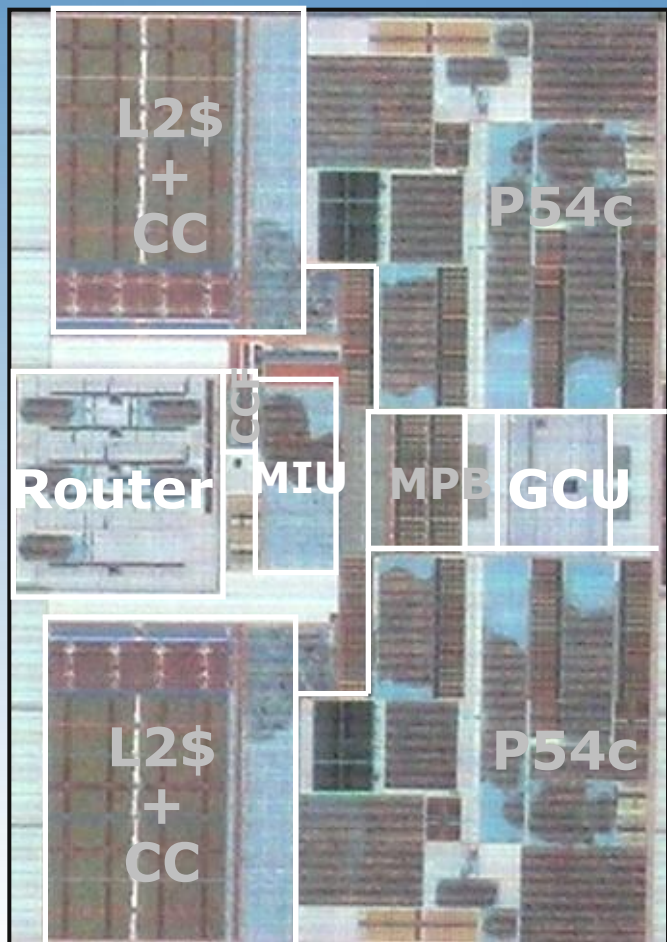


SCC full chip



Technology	45nm Process
Interconnect	1 Poly, 9 Metal (Cu)
Transistors	Die: 1.3B, Tile: 48M
Tile Area	18.7mm ²
Die Area	567.1mm ²

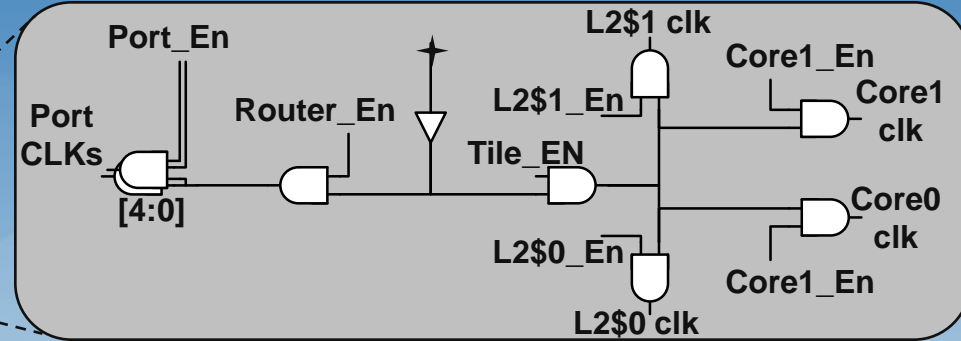
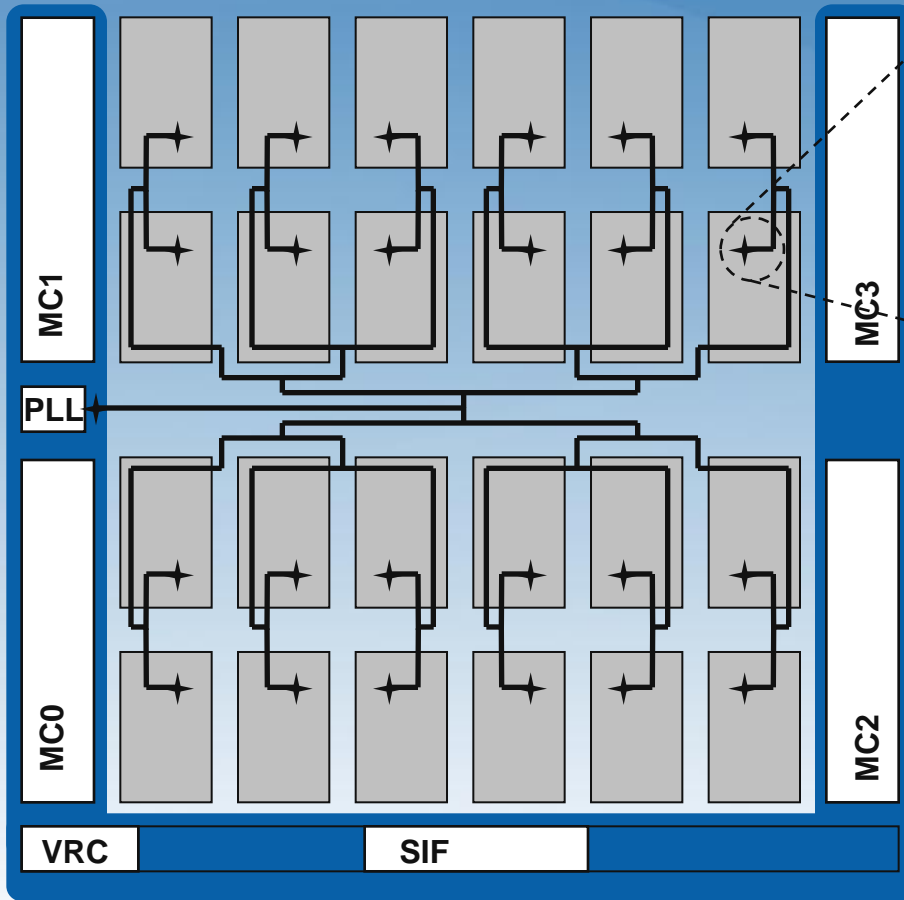
SCC Dual-core Tile



- 2 P54C cores (16K L1\$/core)
- 256K L2\$ per core
- 8K Message passing buffer
- Clock Crossing FIFOs b/w Mesh interface unit and Router

- Tile area 18.7mm^2
- Core are 3.9mm^2
- Cores and uncore units @1GHz
- Router @2GHz

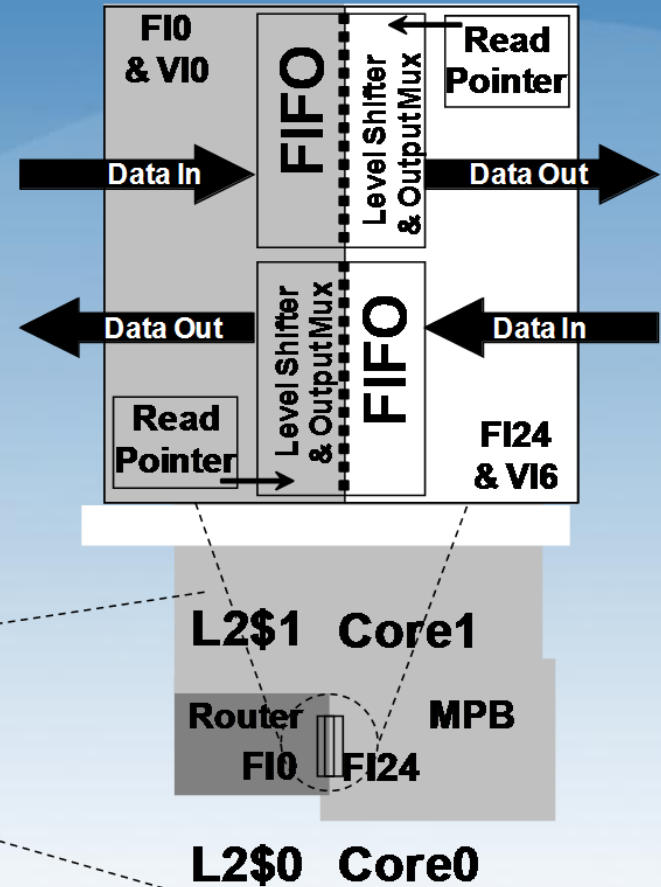
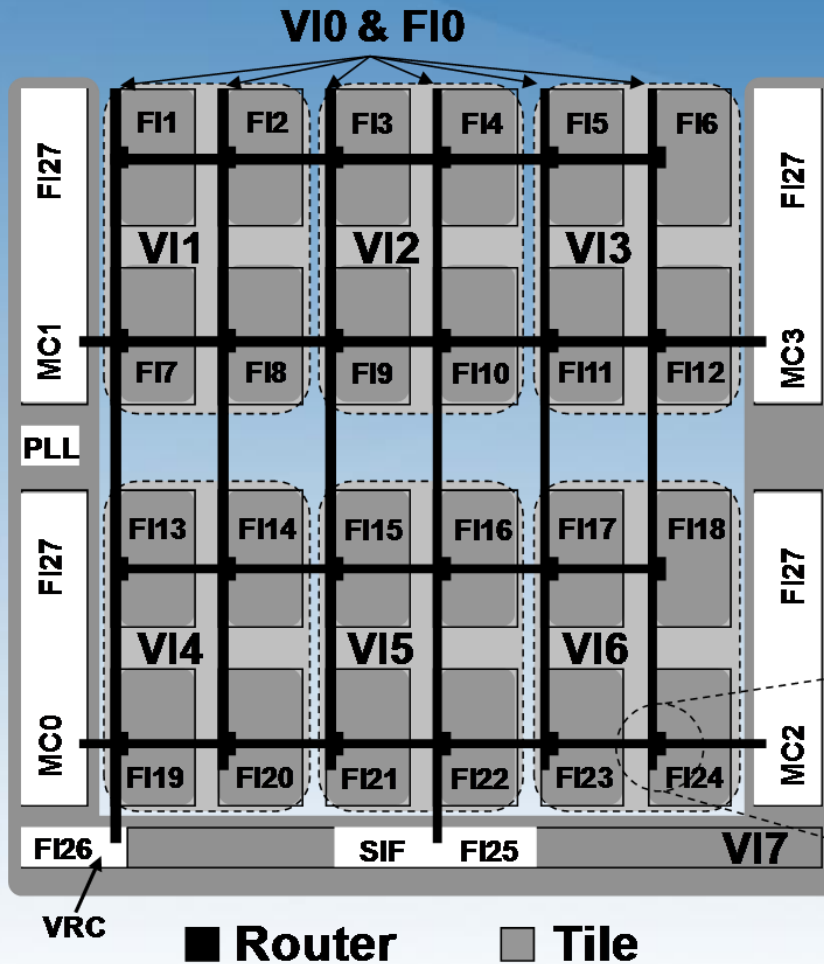
Clock Distribution



- Balanced H-tree clock distribution
- Designed to provide 4GHz clock to tile entry points
- Simulated skew for adjacent tiles - 5ps
- Cross die skew irrelevant

■ Router ■ Tile
+ Clock Gating

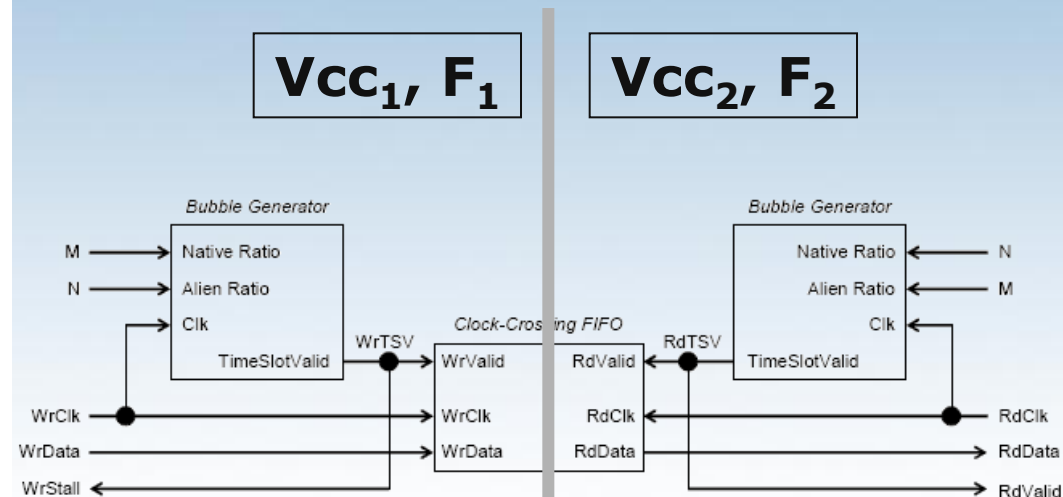
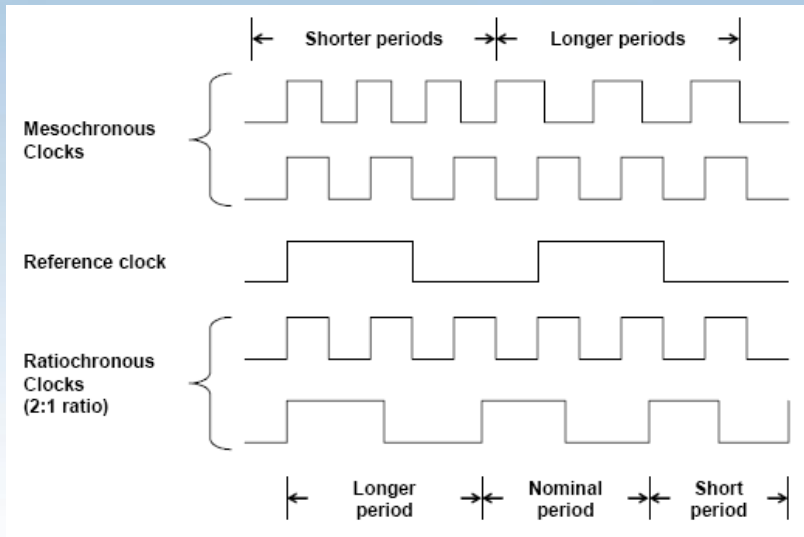
Voltage and Frequency islands



27 Frequency Islands (FI) 8 Voltage Islands (VI)

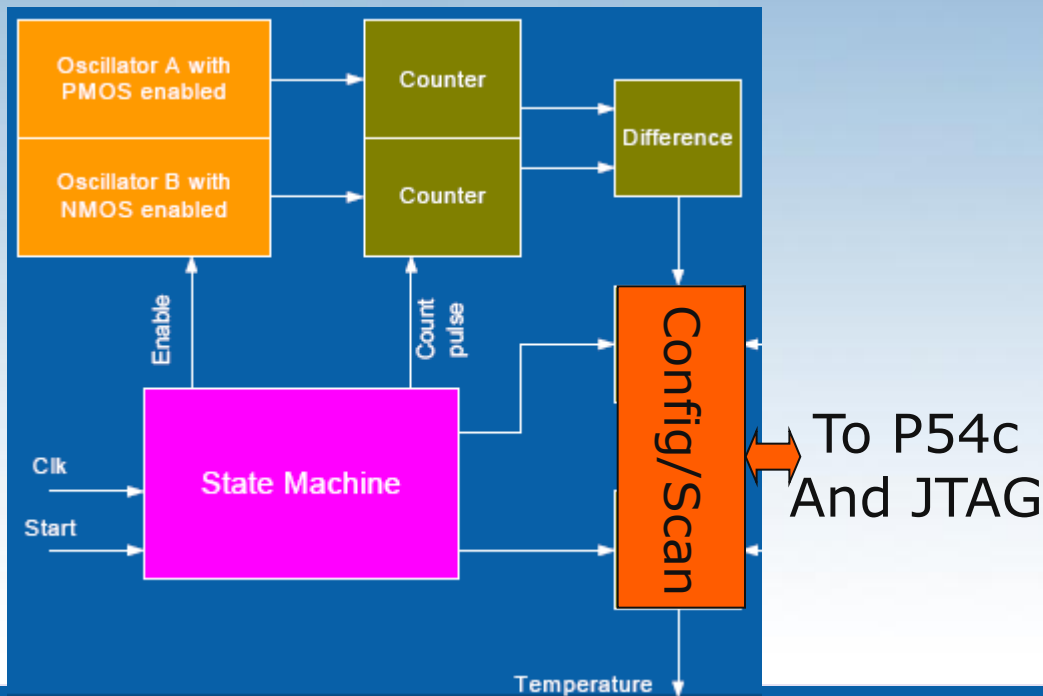
SCC Clock Crossing FIFO (CCF)

- 6 entry deep FIFO, 144-bits wide
- Built-in Voltage translation: M, N ratios and pointer separation scanned in
- Key Benefit: independent mesh & tile frequency

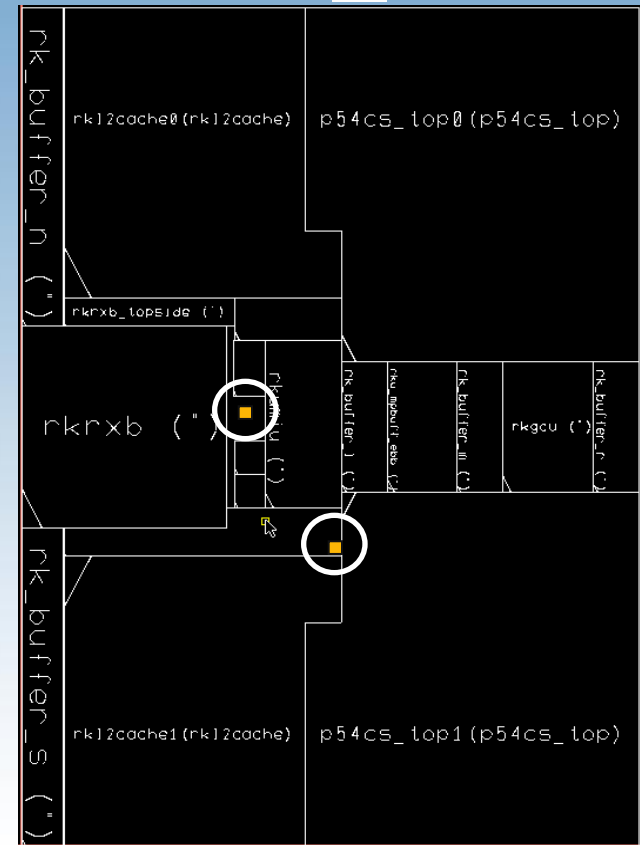


Closed Loop Thermal Management

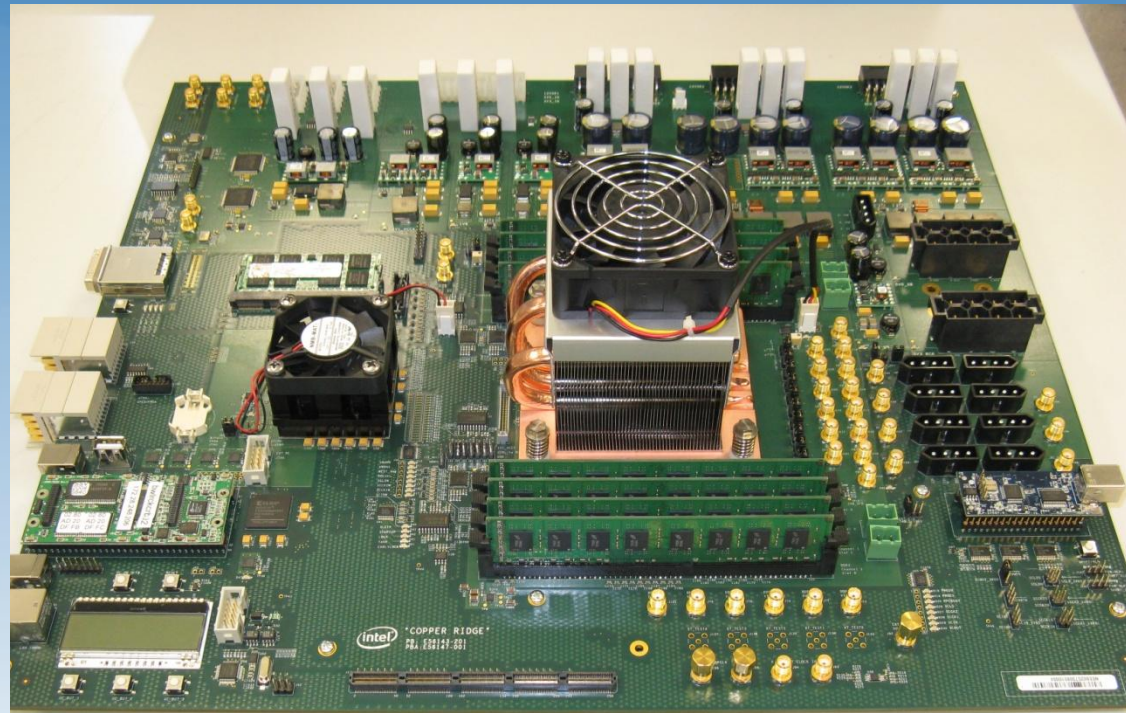
- Network of digital temperature sensors
 - 2 per Tile, 48 total
 - Programmable 13-bit counters via FSM
 - Outputs written to Config registers
 - > Readable by any P54c core for DVFS
 - Readable by core/JTAG and via 2D mesh/SIF



SCC_TILE

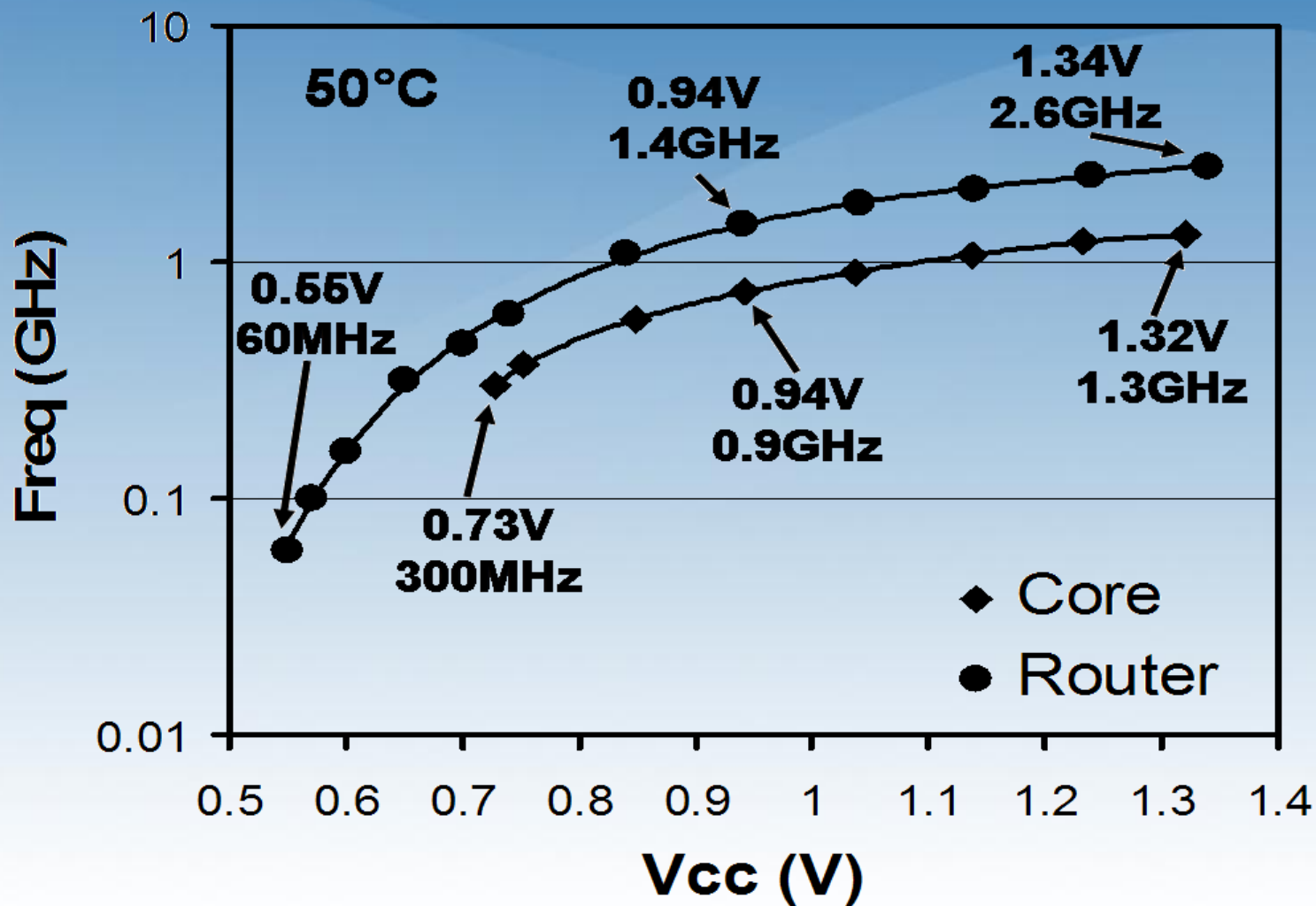


Package and Test Board

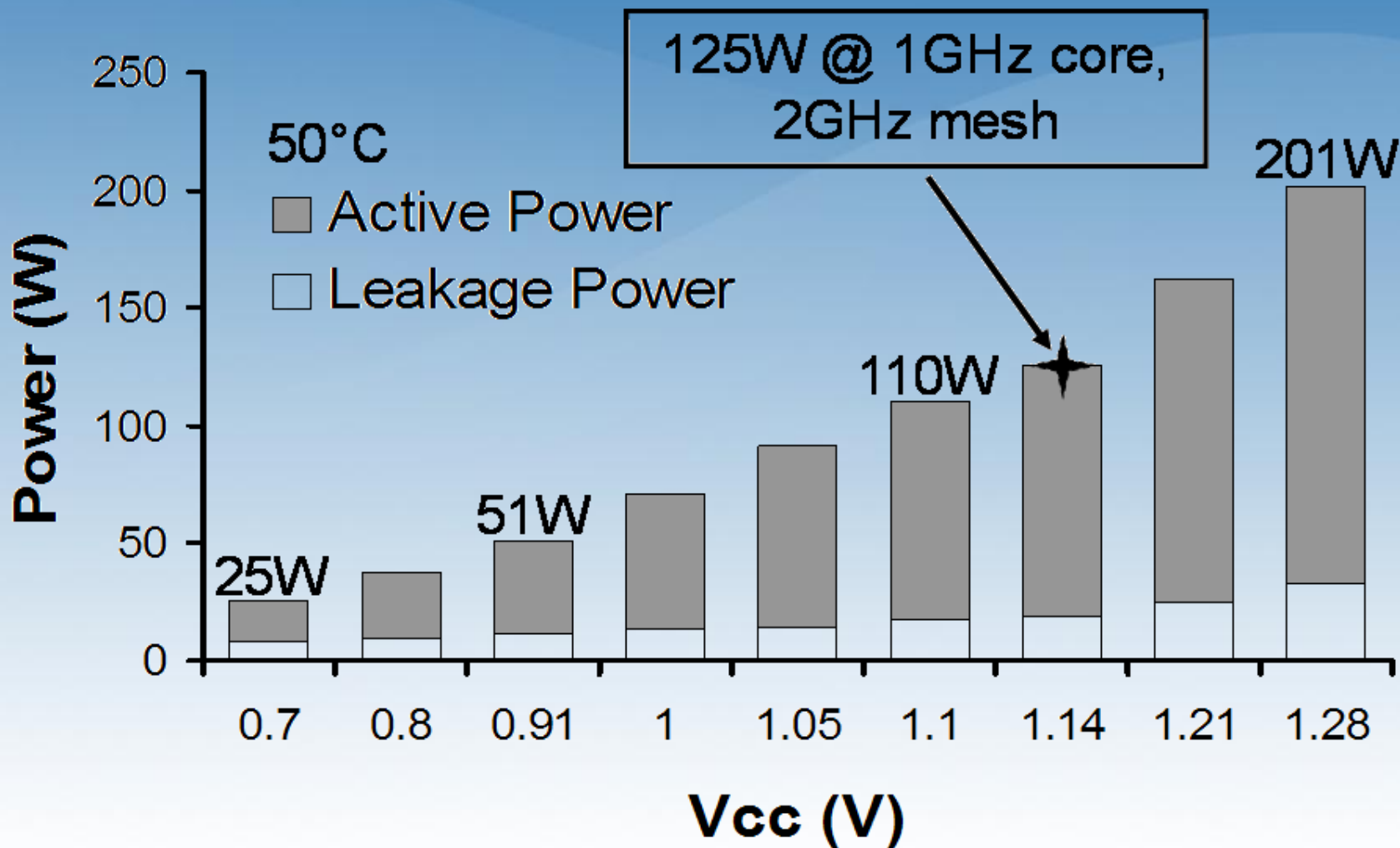


Technology	45nm Process
Package	1567 pin LGA package
	14 layers (5-4-5)
Signals	970 pins

Core & Router Fmax

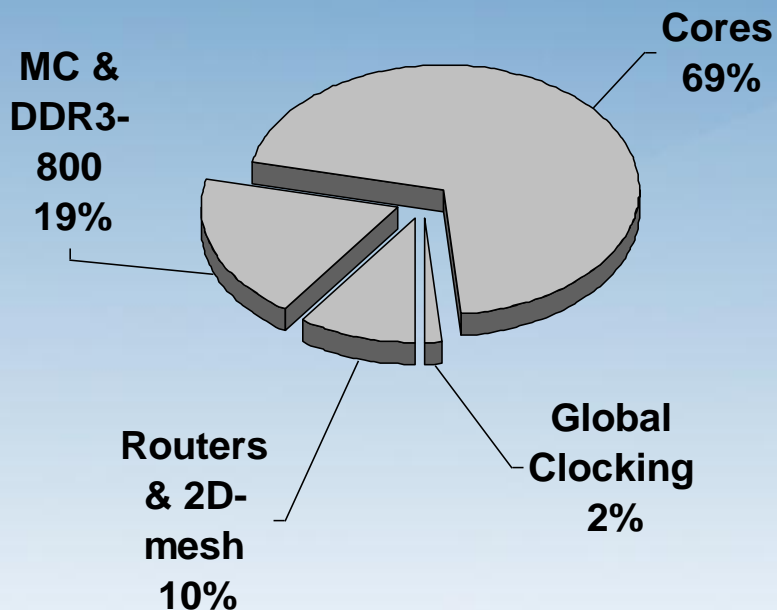


Measured full chip power



Power breakdown

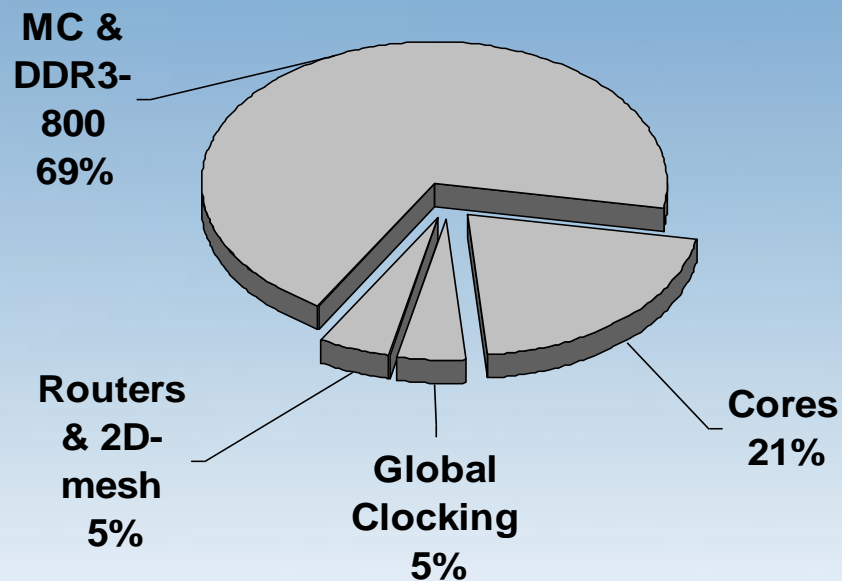
Full Power Breakdown Total -125.3W



Clocking: 1.9W Routers: 12.1W
Cores: 87.7W MCs: 23.6W

Cores-1GHz, Mesh-2GHz, 1.14V, 50°C

Low Power Breakdown Total - 24.7W



Clocking: 1.2W Routers: 1.2W
Cores: 5.1W MCs: 17.2W

Cores-125MHz, Mesh-250MHz, 0.7V, 50°C

Summary

- A 48 IA-32 core processor in 45nm CMOS
 - Second generation 2D-mesh network
 - 4 DDR3 channels in a 6×4
 - Highest level of IA-32 integration
- New message passing HW for increased performance
 - 384KB of on-die shared memory
 - Message passing memory type
- Power management employs 8VIs and 28FIs for DVFS
- Chip dissipates between 25W and 125W as performance scales
 - 25W at 0.7, 125MHz core, 250MHz mesh and 50°C
 - 125W at 1.14V, 1GHz core, 2GHz mesh and 50°C

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

sccKit

Software framework for SCC Platform

Michael Riepen, Michael Konow
Germany Research Center



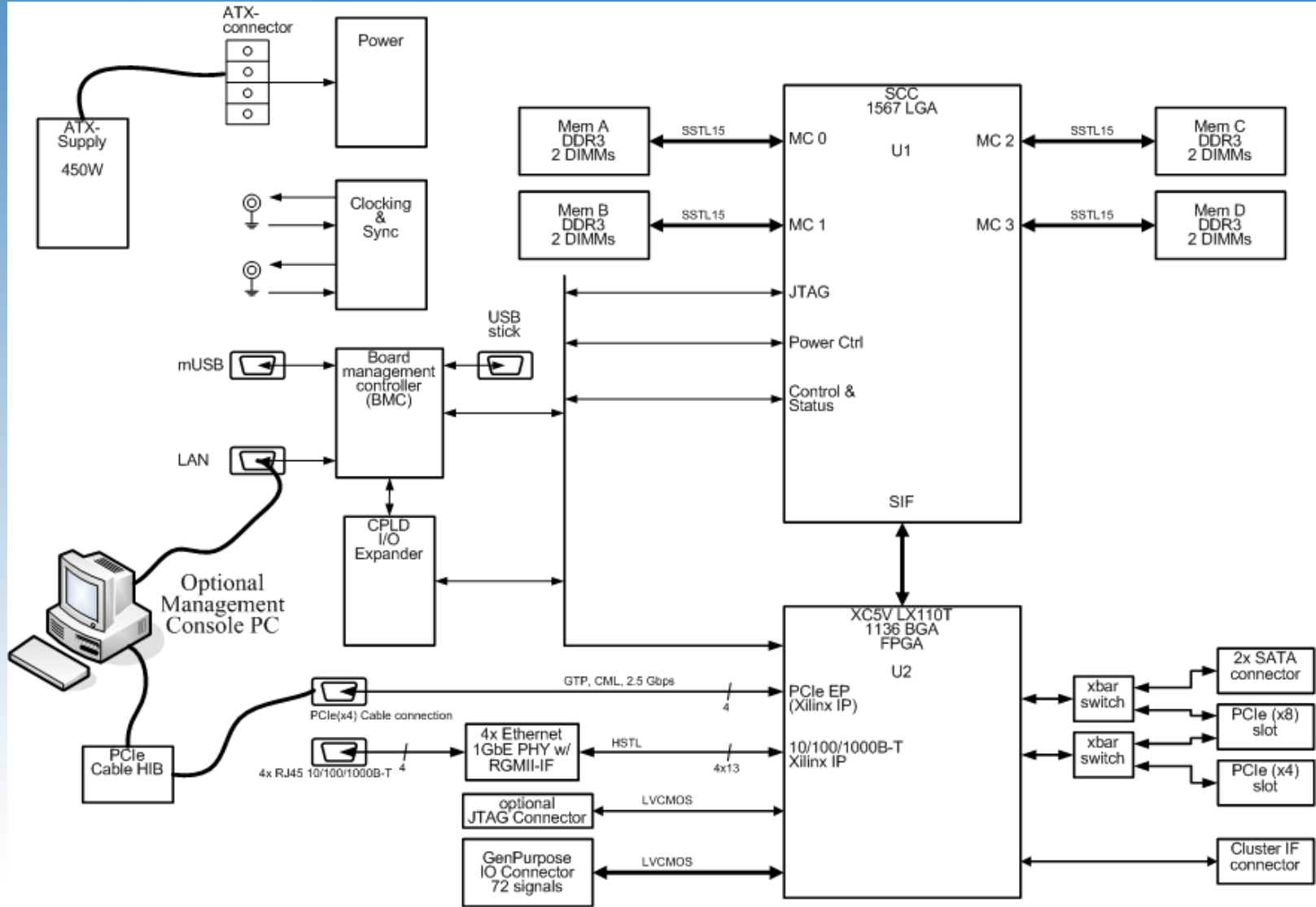
Content

- Platform
 - Platform overview
 - System Interface FPGA bitstream
 - Board management controller (BMC) firmware
- SCC Software
 - Customized Linux
 - bareMetalC
- Management Console PC Software
 - PCIe driver with integrated TCP/IP driver
 - Programming API for communication with SCC platform
 - GUI for interaction with SCC platform
 - Command line tools for interaction with SCC platform

Content

- Platform
 - Platform overview
 - System Interface FPGA bitstream
 - Board management controller (BMC) firmware
- SCC Software
 - Customized Linux
 - bareMetalC
- Management Console PC Software
 - PCIe driver with integrated TCP/IP driver
 - Programming API for communication with SCC platform
 - GUI for interaction with SCC platform
 - Command line tools for interaction with SCC platform

SCC Platform Board Overview



SCC “Chipset”

- System Interface FPGA
 - Connects to SCC Mesh interconnect
 - IO capabilities like PCIe, Ethernet & SATA
 - Bitstream is part of sccKit distribution
- Board Management Controller (BMC)
 - JTAG interface for Clocking, Power etc.
 - USB Stick with FPGA bitstream
 - Network interface for User interaction via Telnet
 - Status monitoring
 - Firmware is part of sccKit distribution

Content

- Platform
 - Platform overview
 - System Interface FPGA bitstream
 - Board management controller (BMC) firmware
- SCC Software
 - Customized Linux
 - bareMetalC
- Management Console PC Software
 - PCIe driver with integrated TCP/IP driver
 - Programming API for communication with SCC platform
 - GUI for interaction with SCC platform
 - Command line tools for interaction with SCC platform

SCC Linux Build

The sccKit comes with a custom Linux build which can be used to execute own applications:

- Kernel 2.6.16 with Busybox 1.15.1
- Booting w/o BIOS possible (Kernel mods)
- Dropbear ssh
- On-die TCP/IP network drivers
- Off-die TCP/IP driver for connection to management console including NFS service.
- Drivers for low level access to SCC specific hardware (e.g. MPB).

SCC Linux Apps

- Cross-compilers for Pentium processor compatible IA cores available (C++, Fortran)
- Write own low level device drivers for deeper dives.
- Cross compiled MPI2 including iTAC trace analyzer available.

Creating own SCC binaries

- It is also possible to write software that directly executes on SCC cores such as an operating system.
- C++ based programming framework “bareMetalC” is available and allows direct access to all dedicated SCC hardware features (e.g. MPB).

Up sides	Down sides
Direct access to low level features of SCC.	Limited IO capabilities.
No overhead from OS.	Harder to debug.
Full flexibility.	Low level coding w/o OS

bareMetalC Apps

- bareMetalC used for bring-up and production tests (e.g. BIST test)
- Useful for creation of own low level apps (e.g. customized OS)
- SCC communication environment (RCCE) with MPI-like API available including several applications

Content

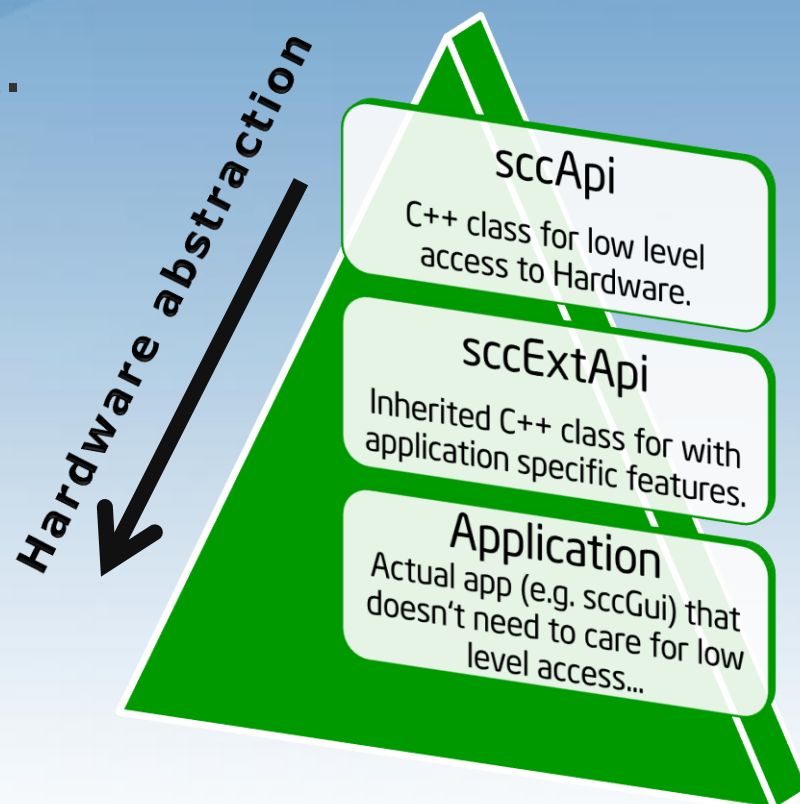
- Platform
 - Platform overview
 - System Interface FPGA bitstream
 - Board management controller (BMC) firmware
- SCC Software
 - Customized Linux
 - bareMetalC
- Management Console PC Software
 - PCIe driver with integrated TCP/IP driver
 - Programming API for communication with SCC platform
 - GUI for interaction with SCC platform
 - Command line tools for interaction with SCC platform

PCIe driver with Ethernet

- Management Console PC comes with PCIe driver that provides:
 - TCP/IP connection to SCC
 - Connection to Management Console PC applications.
 - Access to all memory and register locations of SCC.

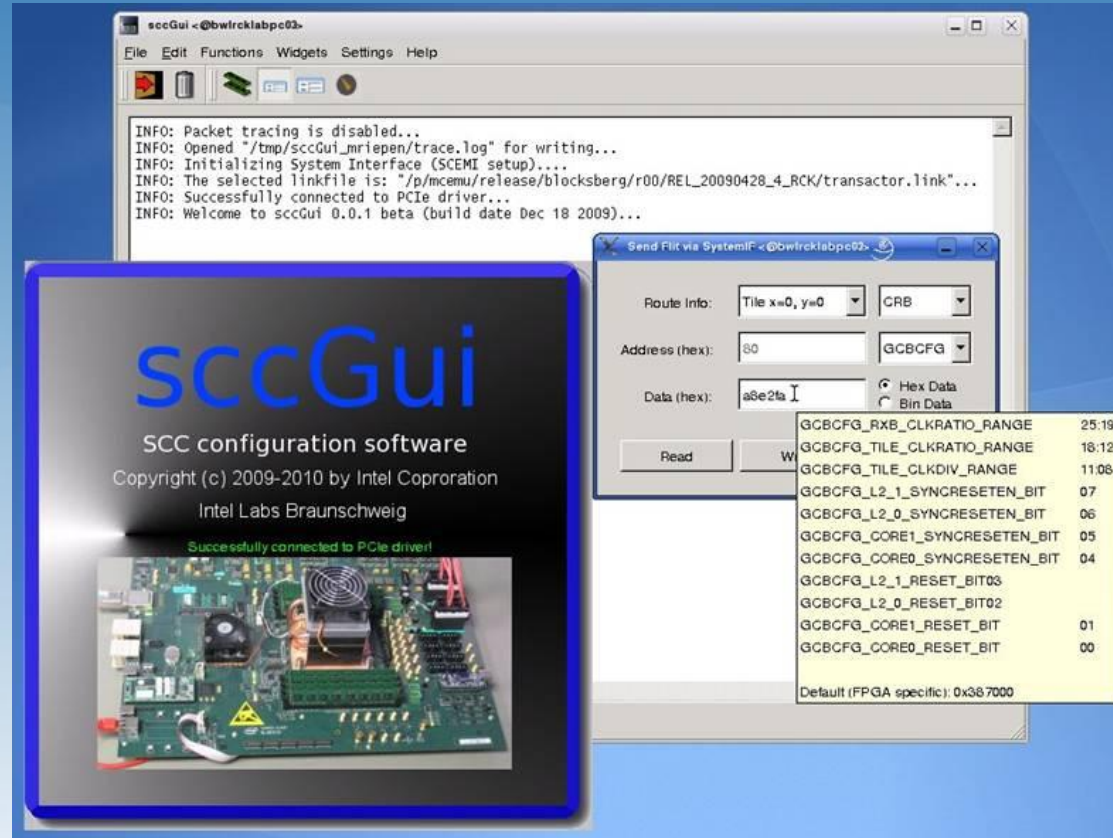
Creating Management Console PC Apps

- Written in C++ making use of Nokia Qt cross-platform application and UI framework.
- Low level API (sccApi) with access to SCC and board management controller via PCIe.
- Code of sccGui as well as command line tools is available as code example. These tools use and extend the low level API.

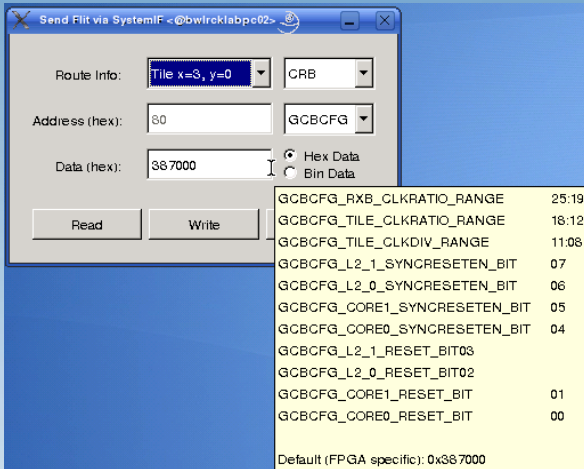


sccGui

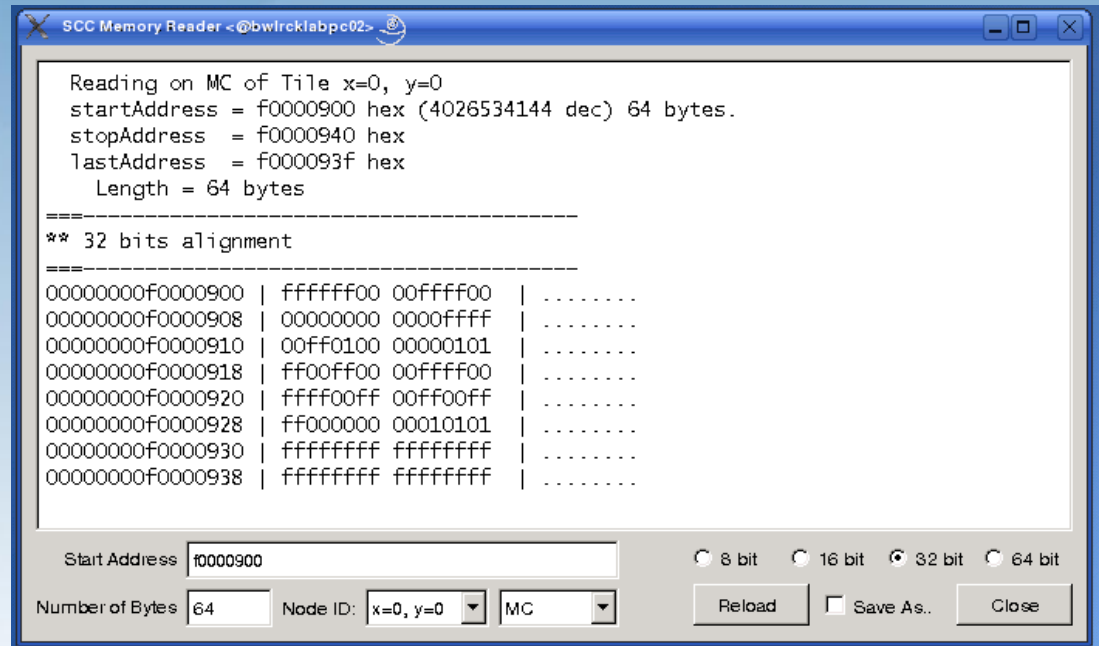
- Read and write system memory and registers.
- Boot OS or other workloads (e.g. bareMetalC).
- Open SSH connections to booted Linux cores
- Performance meter
- Initialize Platform via Board Management Controller.



sccGui for debugging



Modify registers



Read system memory

sccBoot & sccReset

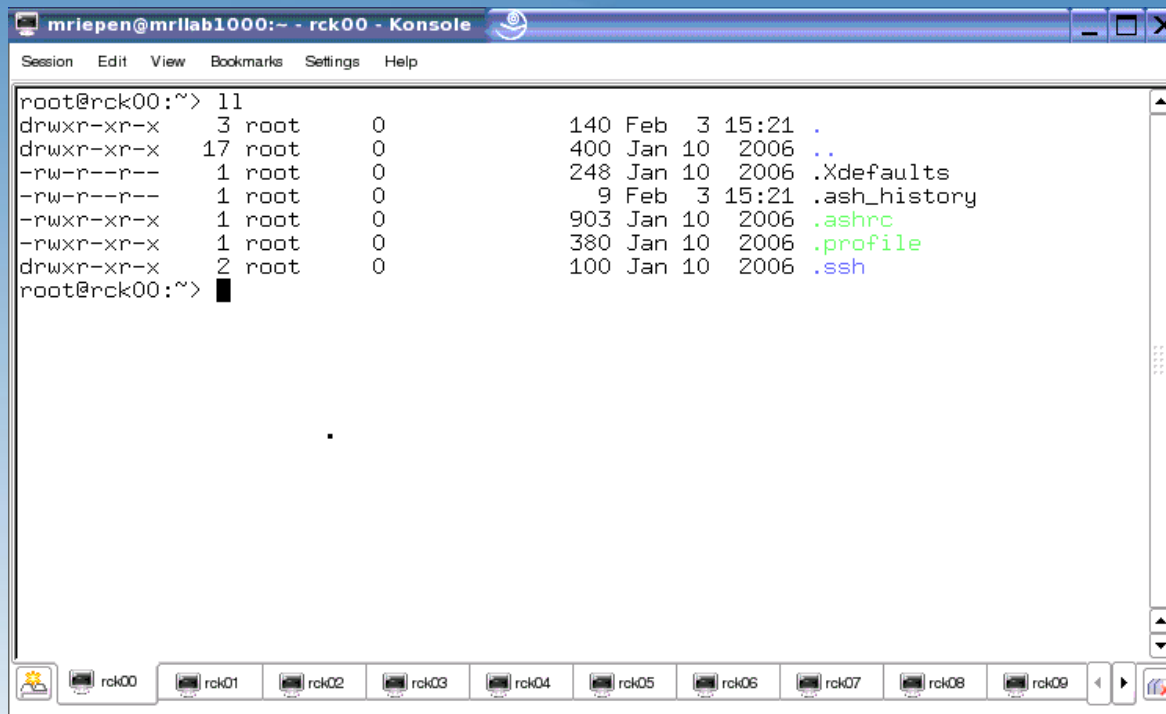
- sccBoot:

A command-line tool that allows to boot Linux on selected cores and to check the status (“which cores are currently booted”).

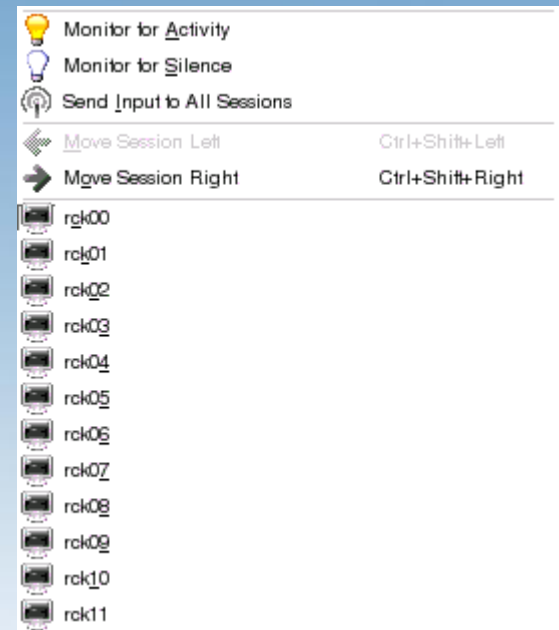
- sccReset:

A command-line tool that allows to reset selected SCC cores.

sccKonsole



```
mrieppen@mrlab1000:~ - rck00 - Konsole
Session Edit View Bookmarks Settings Help
root@rck00:~> ll
drwxr-xr-x  3 root    0      140 Feb  3 15:21 .
drwxr-xr-x 17 root    0      400 Jan 10 2006 ..
-rw-r--r--  1 root    0      248 Jan 10 2006 .Xdefaults
-rw-r--r--  1 root    0         9 Feb  3 15:21 .ash_history
-rwxr-xr-x  1 root    0     903 Jan 10 2006 .ashrc
-rwxr-xr-x  1 root    0     380 Jan 10 2006 .profile
drwxr-xr-x  2 root    0     100 Jan 10 2006 .ssh
root@rck00:~> █
```



- Monitor for Activity
- Monitor for Silence
- Send Input to All Sessions
- Move Session Left (Ctrl+Shift+Left)
- Move Session Right (Ctrl+Shift+Right)
- rck00
- rck01
- rck02
- rck03
- rck04
- rck05
- rck06
- rck07
- rck08
- rck09
- rck10
- rck11

- Regular konsole, with automatic login to selected cores.
- Enables broadcasting amongst shells.

Future

- Linux will be the default OS for self-contained booting
- Self-boot firmware is in preparation

Let's shape the future
together!

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 Buffet Lunch – Informal discussions**
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

RCCCE

A small library for many-core communication

Rob van der Wijngaart
Software and Services Group

Tim Mattson
Intel Labs

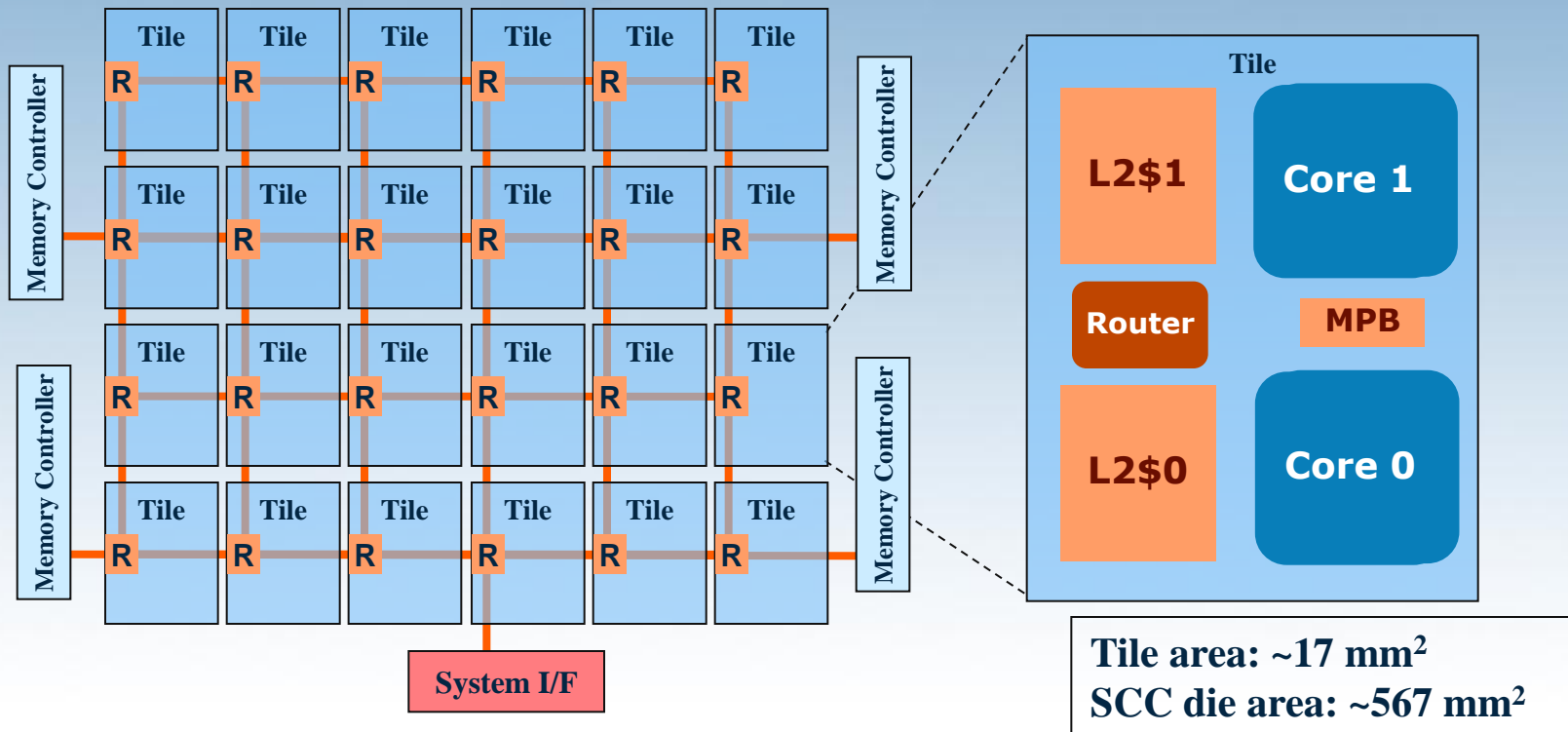


Agenda

- • Views of SCC: HW, SW and Platforms
- RCCE: A communication environment for application programmers.
- Benchmarks and Results
- Power management

Top Level Hardware Architecture

- 6x4 mesh 2 Pentium™ P54c cores per tile
- 256KB L2 Cache, 16KB shared MPB per tile
- 4 iMCs, 16-64 GB total memory

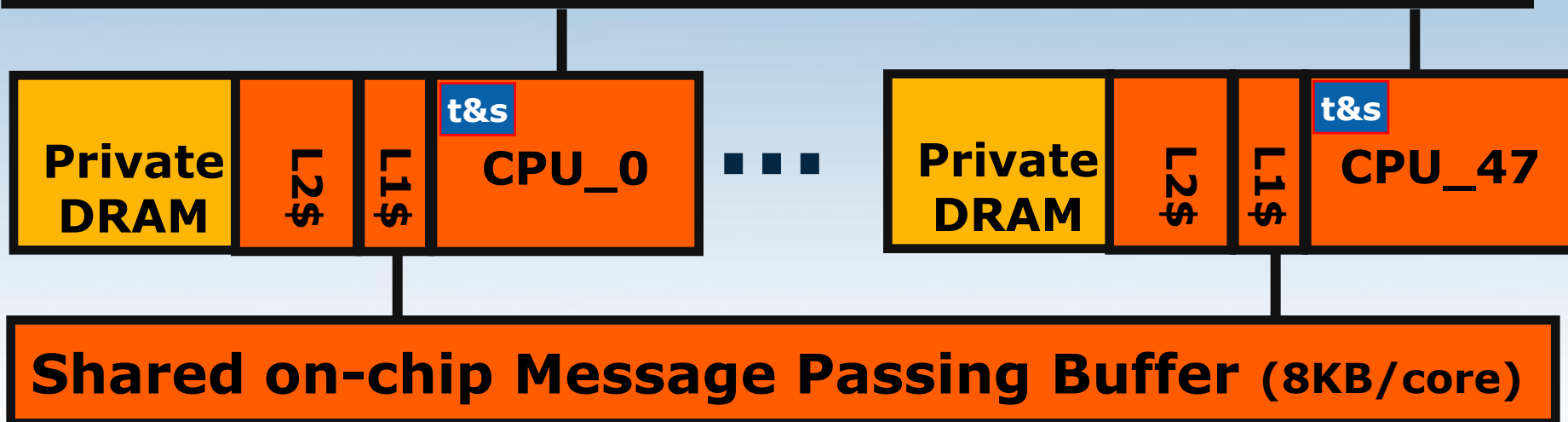



R = router, iMC = integrated Memory Controller, MPB = message passing buffer

Programmer's view of SCC

- 48 x86 cores with the familiar x86 memory model for Private DRAM
- 3 memory spaces, with fast message passing between cores
( /  means on/off-chip)

Shared off-chip DRAM (variable size)



 Shared test and set register

SCC Software research goals

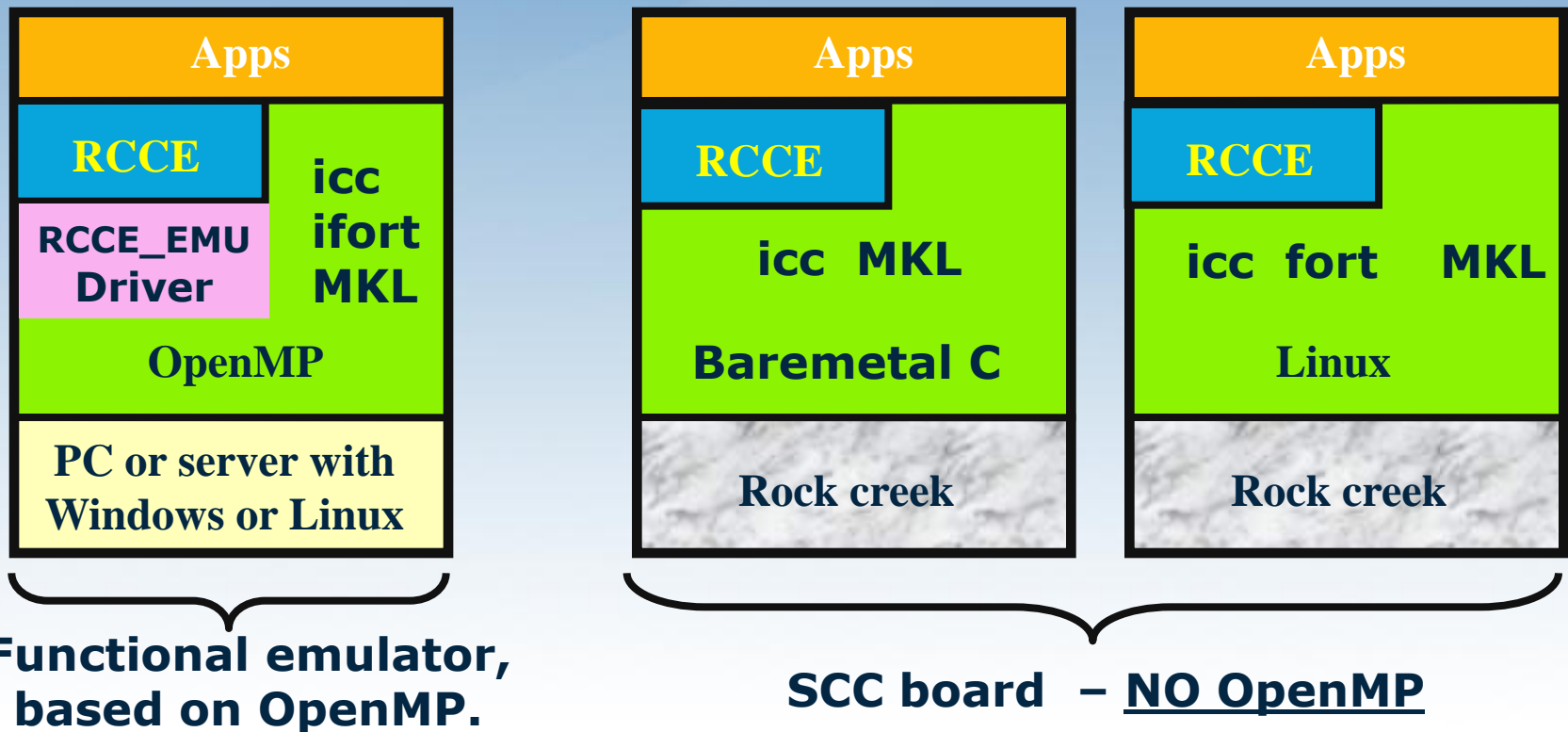
- Understand programmability and application scalability of many-core chips.
- Answer question “what can you do with a many-core chip that has (some) shared non-cache-coherent memory?”
- Study usage models and techniques for software controlled power management
- Sample software for other programming model and applications researchers (industry partners, Flame group at UT Austin, UPCRC, YOU ...)

Our research resulted in a light weight, compact, low latency communication library called RCCE (pronounced “Rocky”)



SCC Platforms

- Three platforms for SCC and RCCE
 - Functional emulator (on top of OpenMP)
 - SCC board with two “OS Flavors” ... Linux or Baremetal (i.e. no OS)



RCCE supports greatest common denominator between the three platforms

Agenda

- Views of SCC: HW, SW and Platforms
- • RCCE: A communication environment for application programmers.
- Benchmarks and Results
- Power management

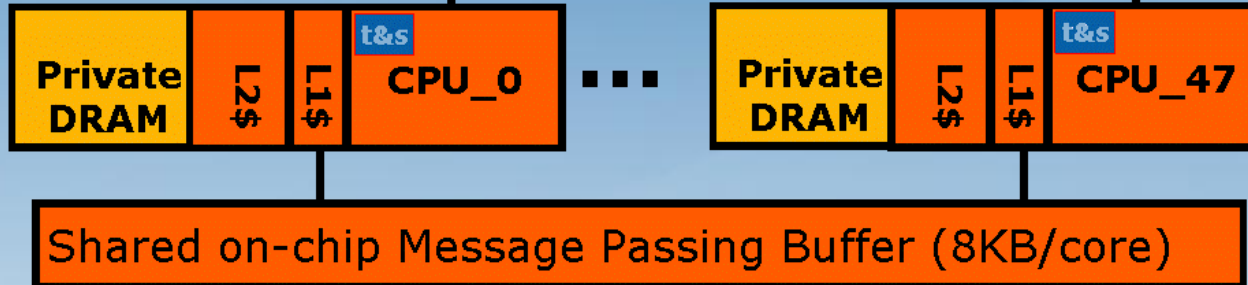
High level view of RCCE

- RCCE is a compact, lightweight communication environment.
 - SCC and RCCE were designed together side by side:
 - > ... a true HW/SW co-design project.
- RCCE is a research vehicle to understand how message passing APIs map onto many core chips.
- RCCE is for experienced parallel programmers willing to work close to the hardware.
- RCCE Execution Model:
 - Static SPMD:
 - > identical UEs created together when a program starts (this is a standard approach familiar to message passing programmers)

UE: Unit of Execution ... a software entity that advances a program counter (e.g. process or thread).

How does RCCE work? Part 1

Shared off-chip DRAM (variable size)



Message passing buffer memory is special ... of type MPBT

Cached in L1, L2 bypassed. Not coherent between cores

Data cached on read, not write. Single cycle op to invalidate all MPBT in L1 ... Note this is not a flush

Consequences of MPBT properties:

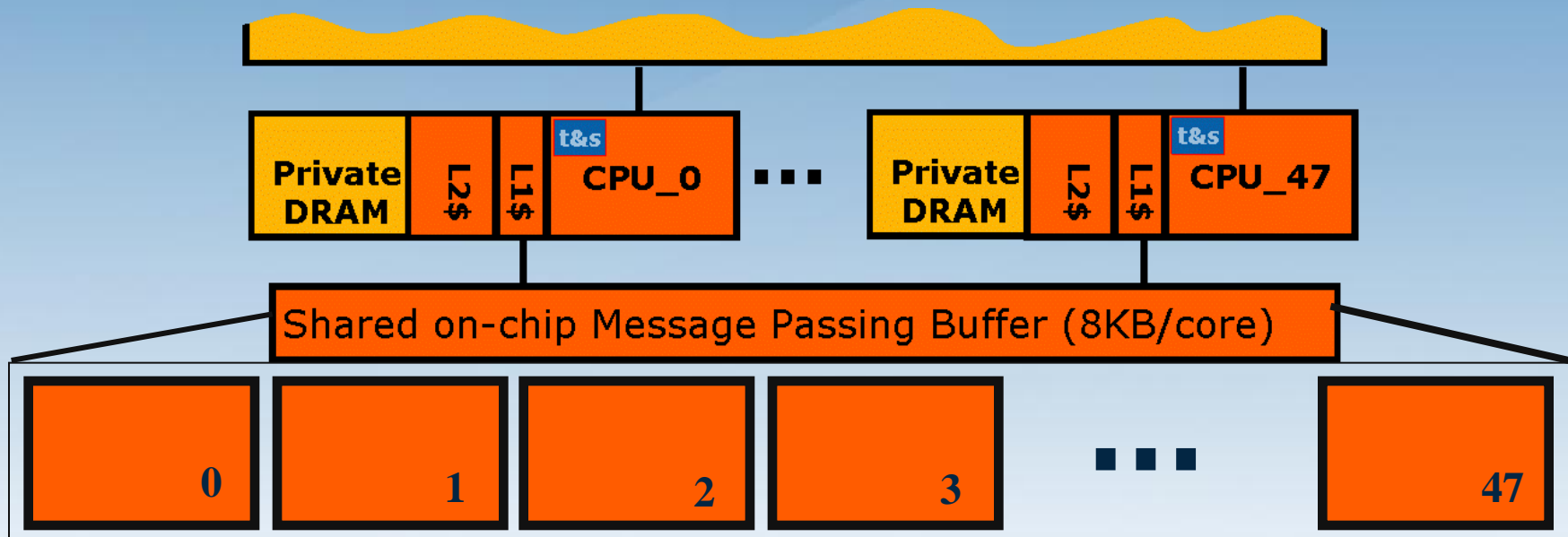
- If data changed by another core and image still in L1, read returns stale data.
 - **Solution: Invalidate before read.**
- L1 has write-combining buffer; write incomplete line? expect trouble!
 - **Solution: don't. Always push whole cache lines**
- If image of line to be written already in L1, write will not go to memory.
 - **Solution: invalidate before write.**

Discourage user operations on data in MPB. Use only as a data movement area managed by RCCE ... Invalidate early, invalidate often



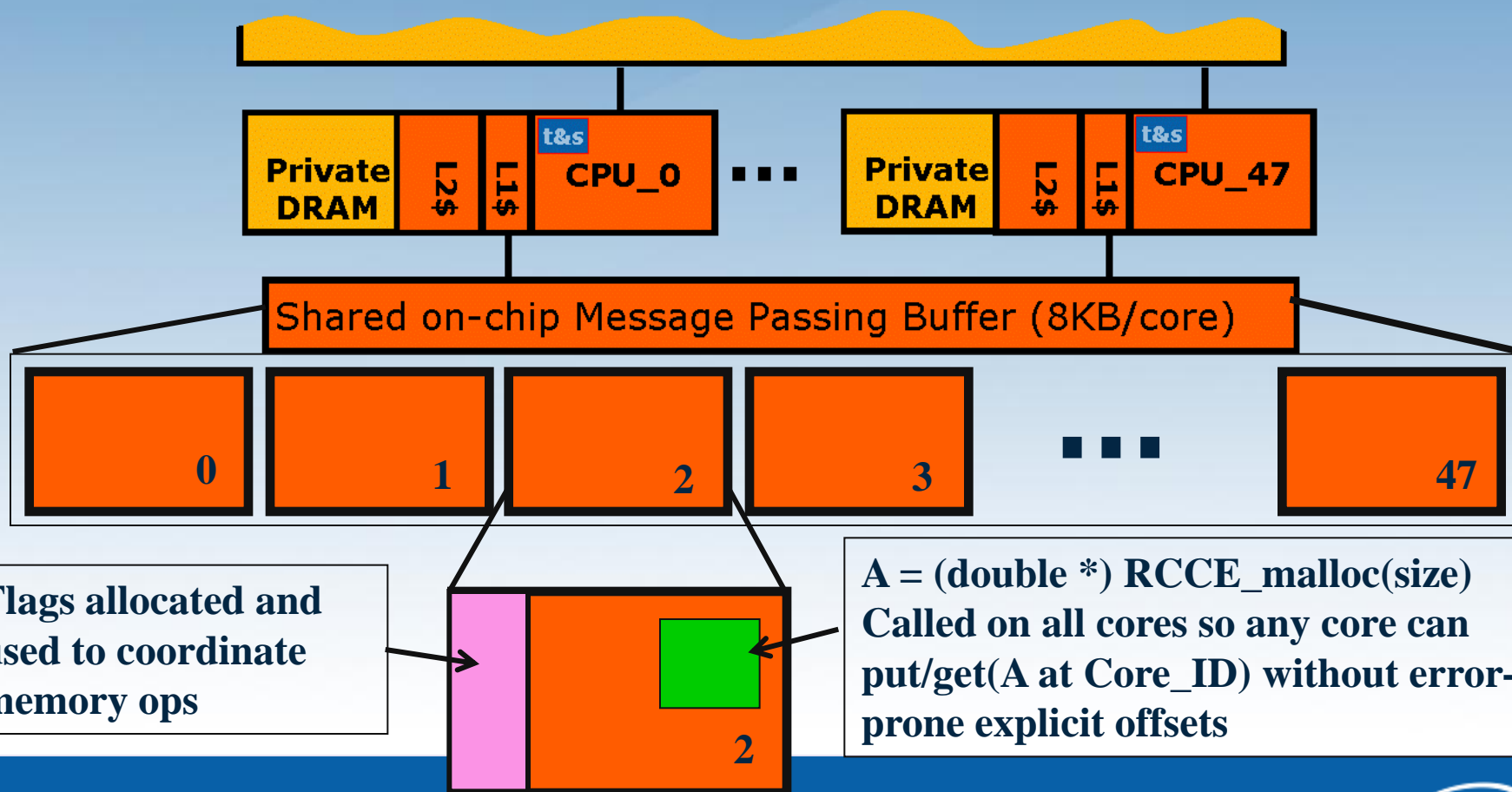
How does RCCE work? Part 2

- Treat Msg Pass Buf (MPB) as 48 smaller buffers ... one per core.



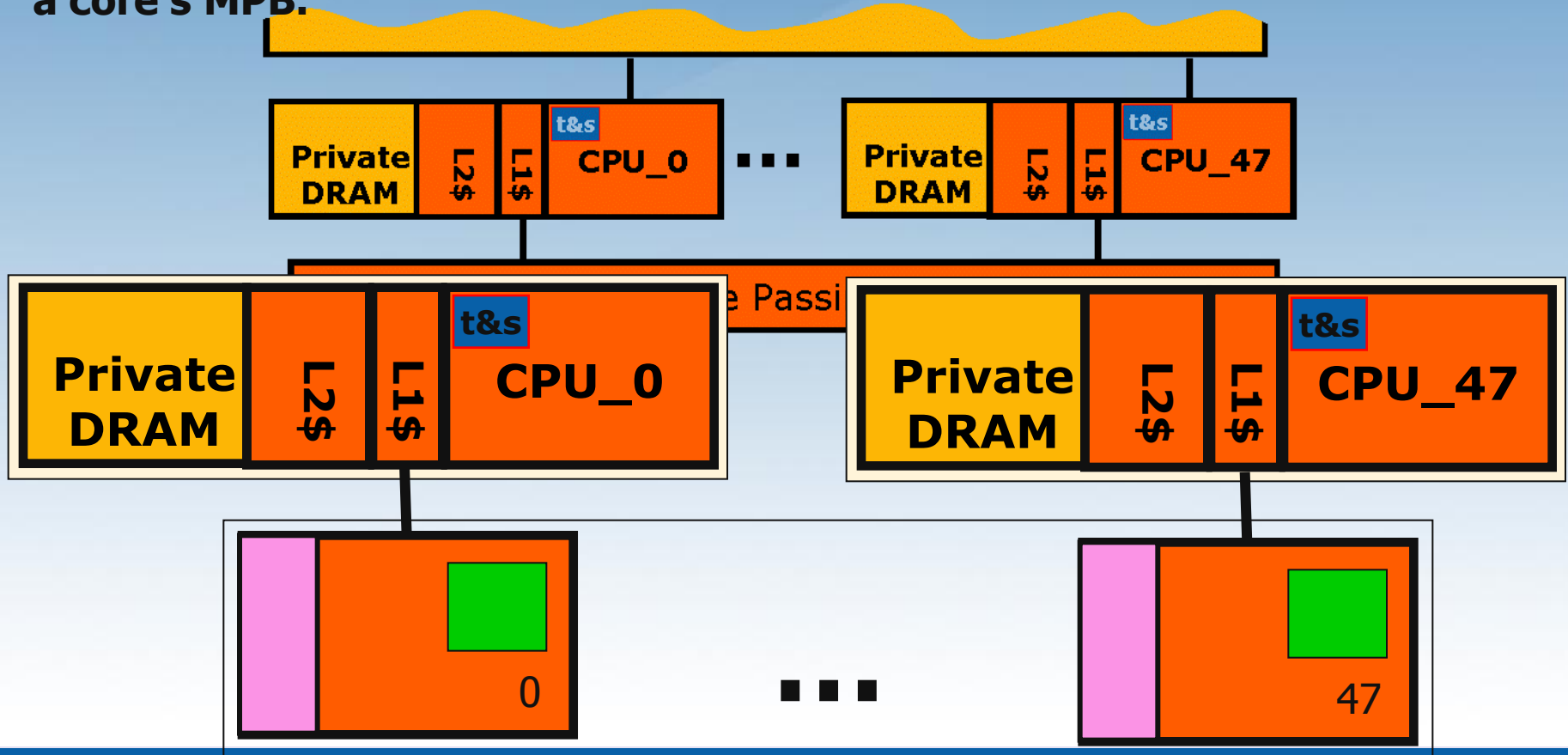
How does RCCE work? Part 2

- Treat Msg Pass Buf (MPB) as 48 smaller buffers ... one per core.
- Symmetric name space ... Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.



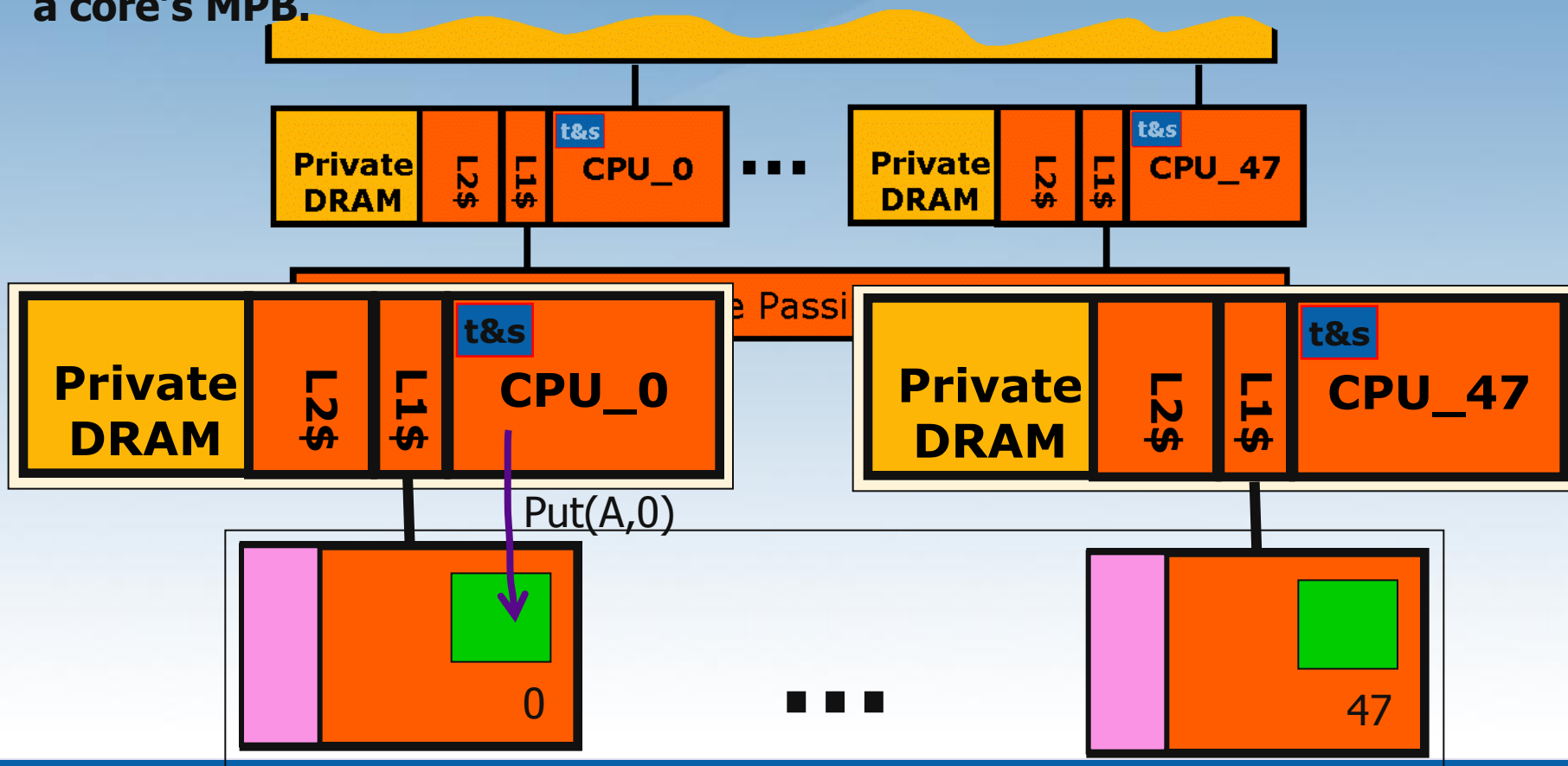
How does RCCE work? Part 3

- The foundation of RCCE is a one-sided put/get interface.
- **Symmetric name space ... Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.**



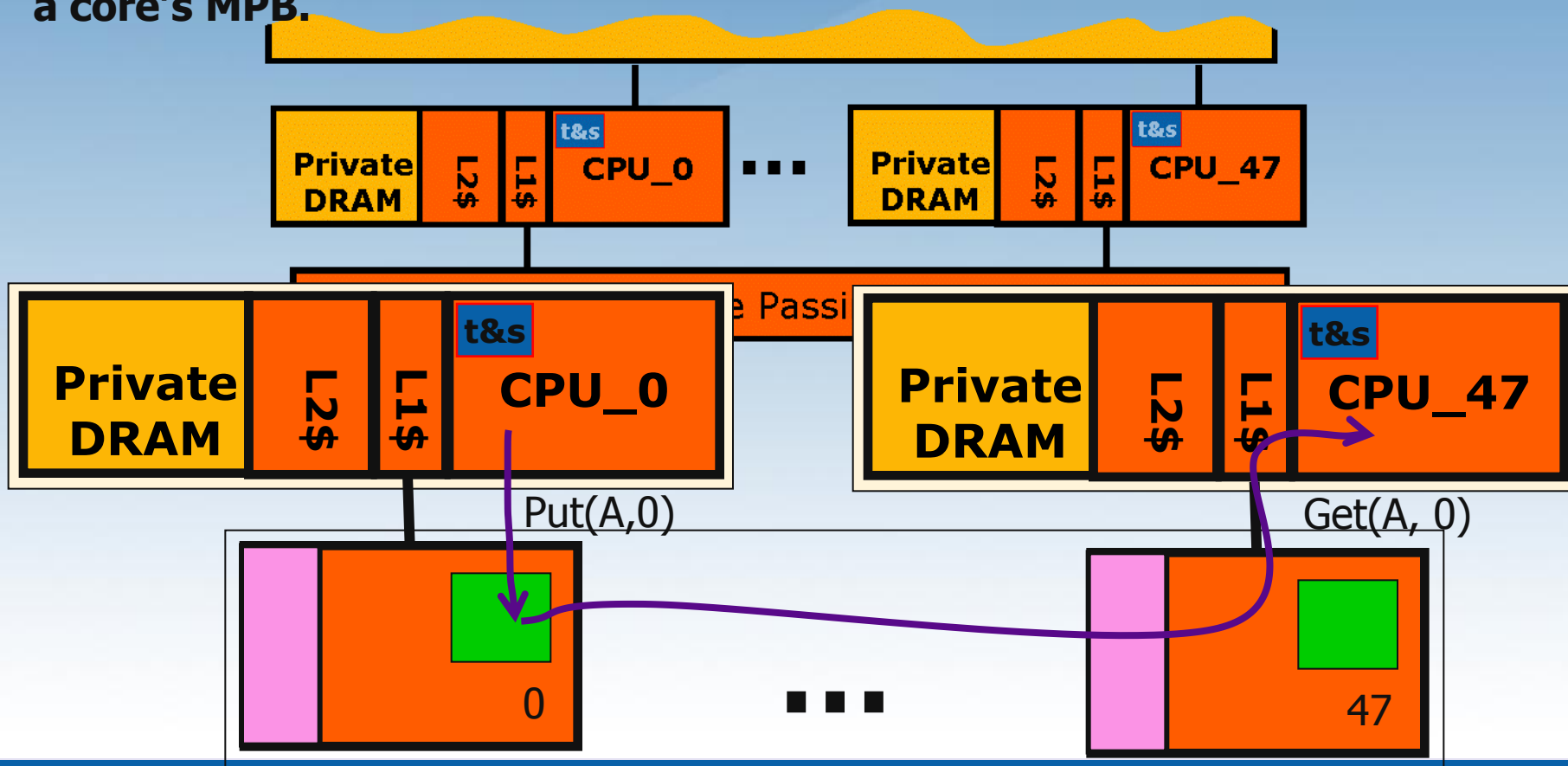
How does RCCE work? Part 3

- The foundation of RCCE is a one-sided put/get interface.
- **Symmetric name space ... Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.**



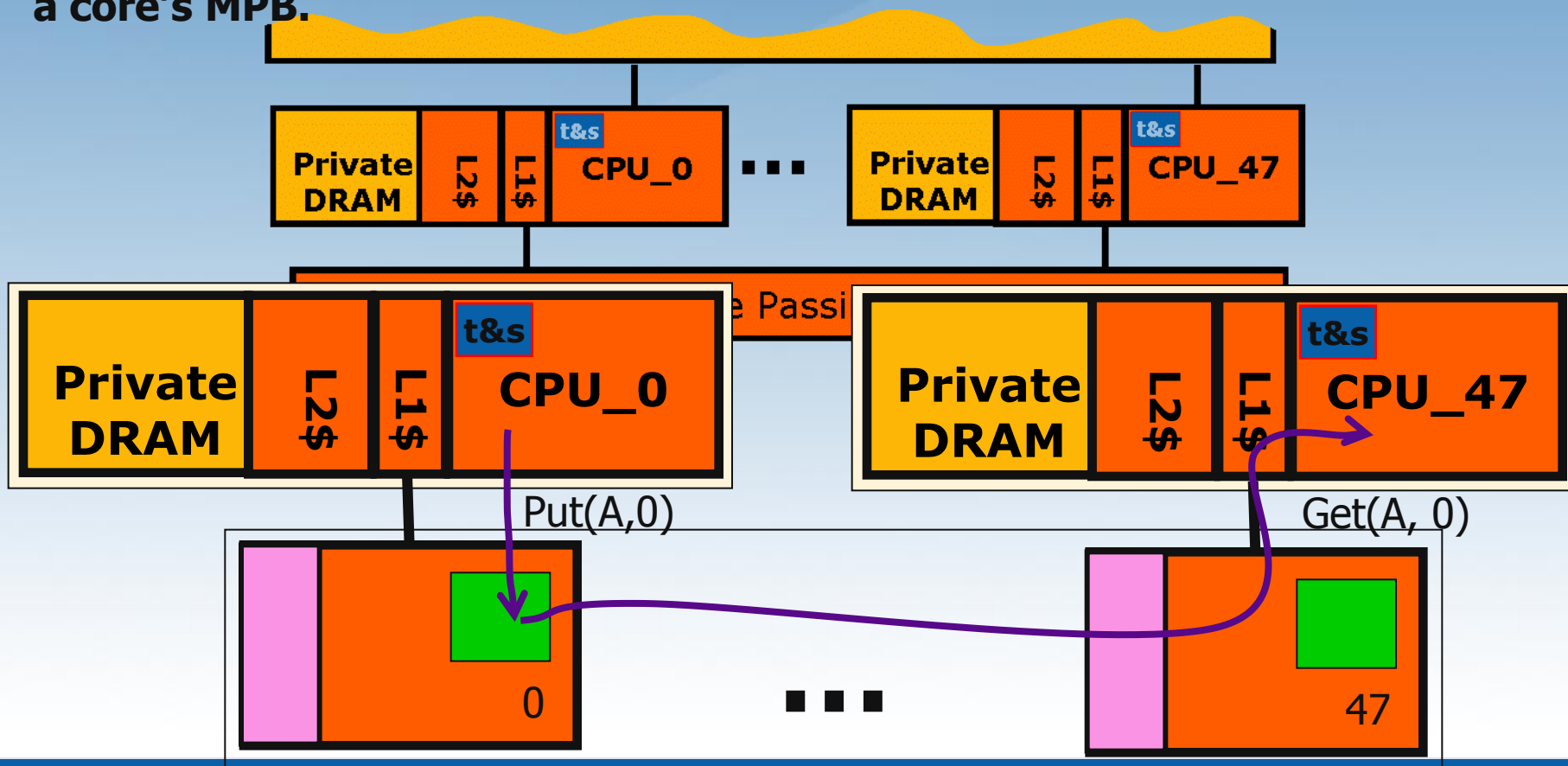
How does RCCE work? Part 3

- The foundation of RCCE is a one-sided put/get interface.
- **Symmetric name space ... Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.**



How does RCCE work? Part 3

- The foundation of RCCE is a one-sided put/get interface.
- **Symmetric name space ... Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.**



... and use flags to make the puts and gets "safe"

The RCCE library

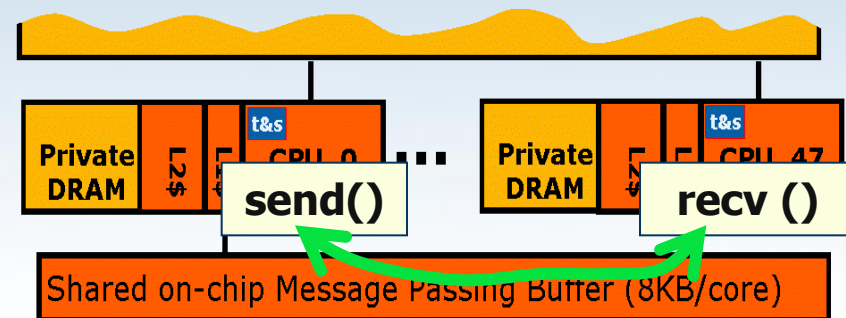
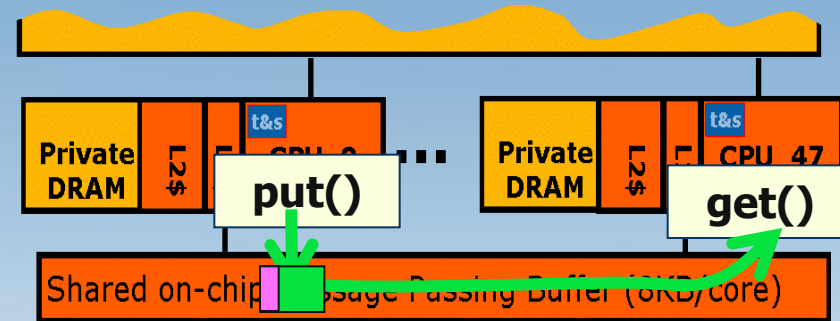
- RCCE API provides the basic message passing functionality expected in a tiny communication library:

- One + two sided interface (`put/get` + `send/recv`) with synchronization flags and MPB management exposed.

- The “gory” interface for programmers who need the most detailed control over SCC

- Two sided interface (`send/recv`) with most detail (flags and MPB management) hidden.

- The “basic” interface for typical application programmers.

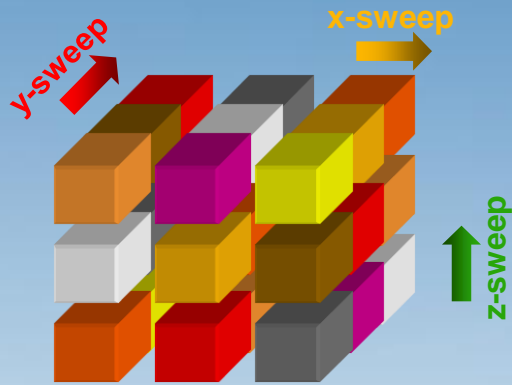


Agenda

- Views of SCC: HW, SW and Platforms
- RCCE: A communication environment for application programmers.
- • Benchmarks and Results
- Power management

Linpack and NAS Parallel benchmarks

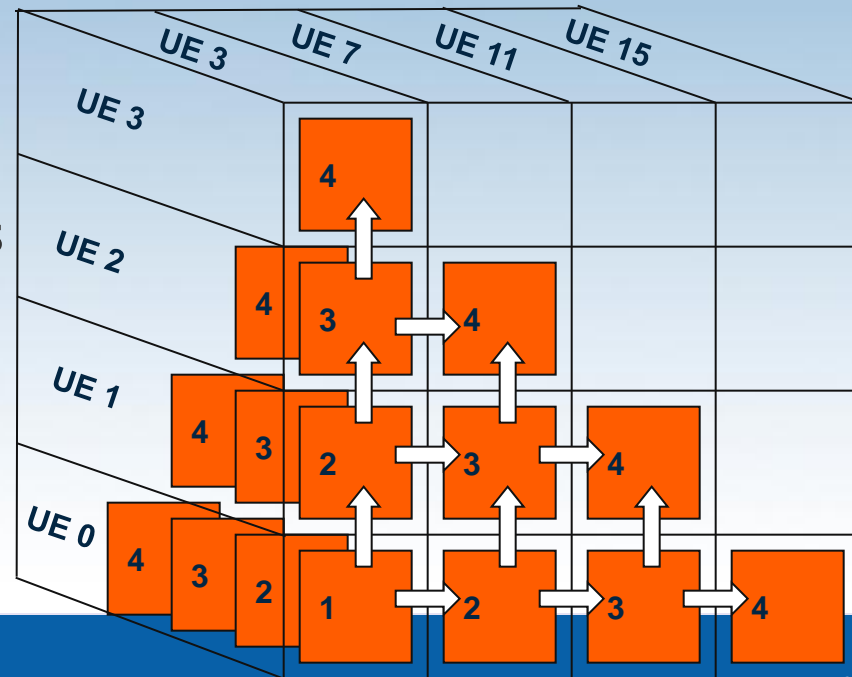
1. **Linpack (HPL):** solve dense system of linear equations
 - Synchronous comm. with "MPI wrappers" to simplify porting



2. **BT: Multipartition decomposition**

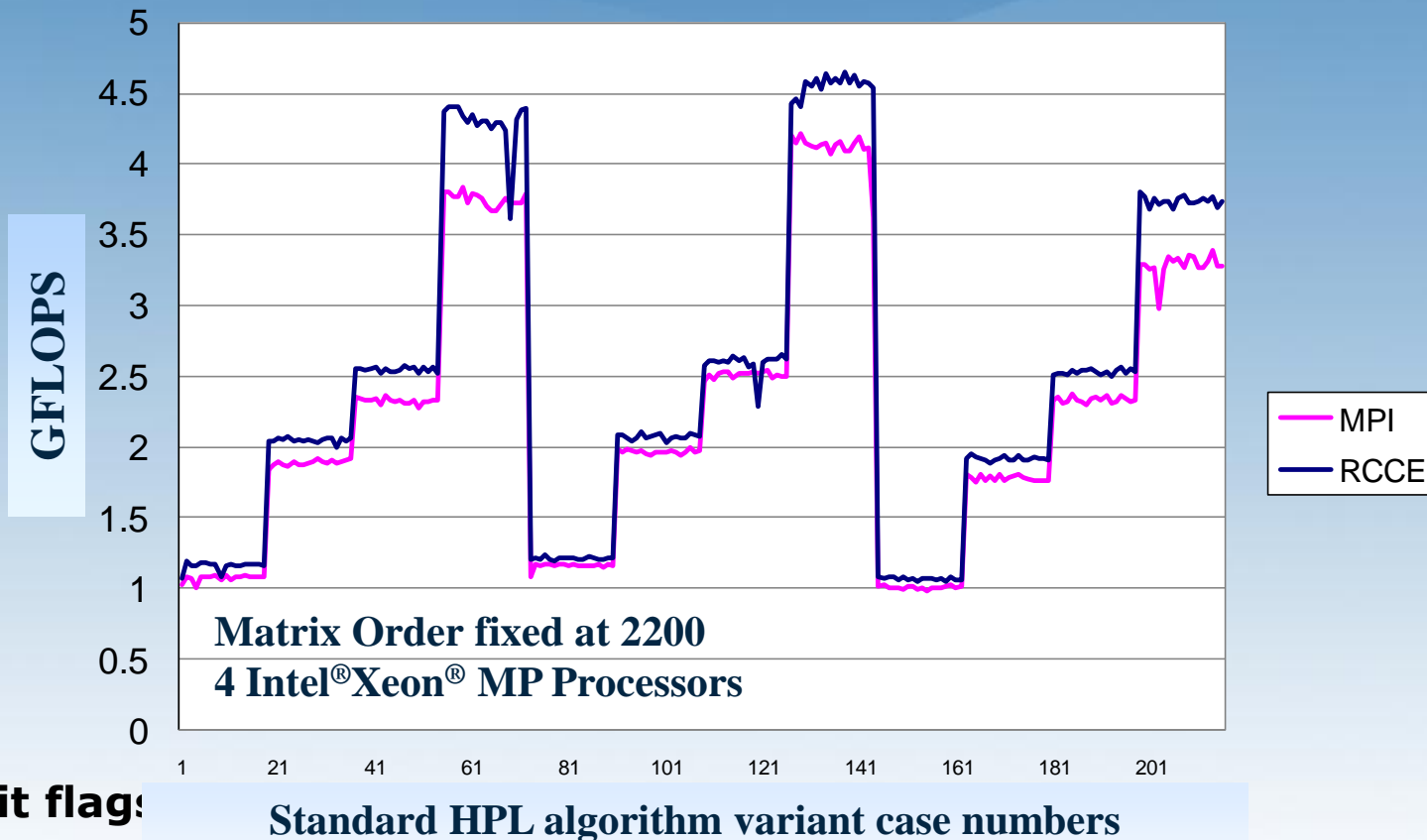
- Each core owns multiple blocks (3 in this case)
- update all blocks in plane of 3x3 blocks
- send data to neighbor blocks in next plane
- update next plane of 3x3 blocks

3. **LU: Pencil decomposition**
Define 2D-pipeline process
 - await data (bottom+left)
 - compute new tile
 - send data (top+right)



RCCE functional emulator vs. MPI

HPL implementation of the LINPACK benchmark



RCCE 1-bit flag:

Standard HPL algorithm variant case numbers

These results provide a comparison of RCCE and MPI on an older 4 processor Intel® Xeon® MP SMP platform* using a tiny 4x4 block size. These are not official MP-LINPACK results.

*3 GHz Intel® Xeon® MP processor in a 4 socket SMP platform (4 cores total), L2=1MB, L3=8MB, Intel® icc 10.1 compiler, Intel® MPI 2.0

Third party names are the property of their owners.

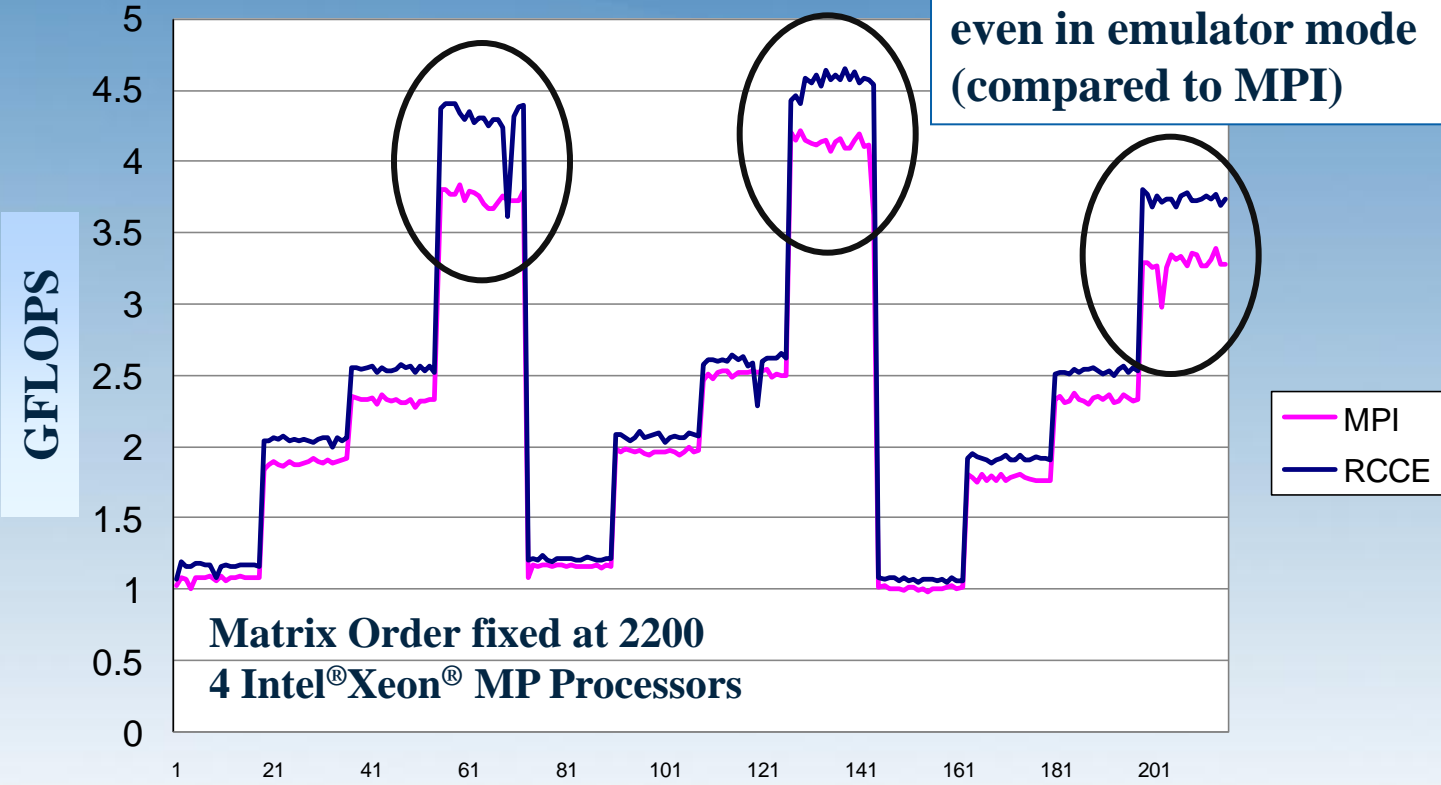
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference <http://www.intel.com/performance> or call (U.S.) 1-800-628-8686 or 1-916-356-3104.



RCCE functional emulator vs. MPI

HPL implementation of the LINPACK benchmark

Low overhead synchronous message passing pays off even in emulator mode (compared to MPI)



RCCE 1-bit flag:

Standard HPL algorithm variant case numbers

These results provide a comparison of RCCE and MPI on an older 4 processor Intel® Xeon® MP SMP platform* using a tiny 4x4 block size. These are not official MP-LINPACK results.

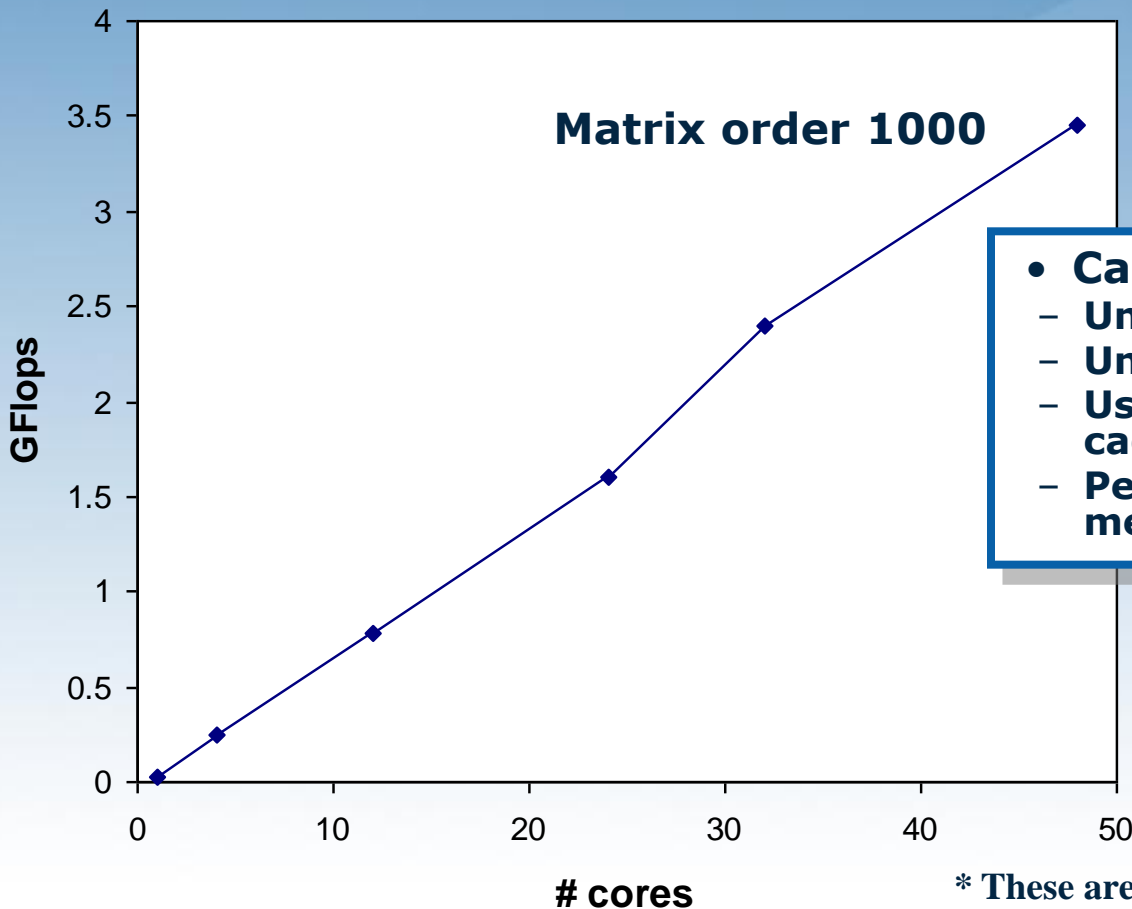
*3 GHz Intel® Xeon® MP processor in a 4 socket SMP platform (4 cores total), L2=1MB, L3=8MB, Intel® icc 10.1 compiler, Intel® MPI 2.0
Third party names are the property of their owners.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference <http://www.intel.com/performance> or call (U.S.) 1-800-628-8686 or 1-916-356-3104.



Linpack, on the Linux SCC platform

- **Linpack (HPL)* strong scaling results:**
 - **GFLOPS vs. # of cores for a fixed size problem (1000).**
 - **This is a tough test ... scaling is easier for large problems.**



- **Calculation Details:**

- **Un-optimized C-BLAS**
- **Un-optimized block size (4x4)**
- **Used latency-optimized whole cache line flags**
- **Performance dropped ~10% with memory optimized 1-bit flags**

* These are not official LINPACK benchmark results.

SCC processor 500MHz core, 1GHz routers, 25MHz system interface, and DDR3 memory at 800 MHz.

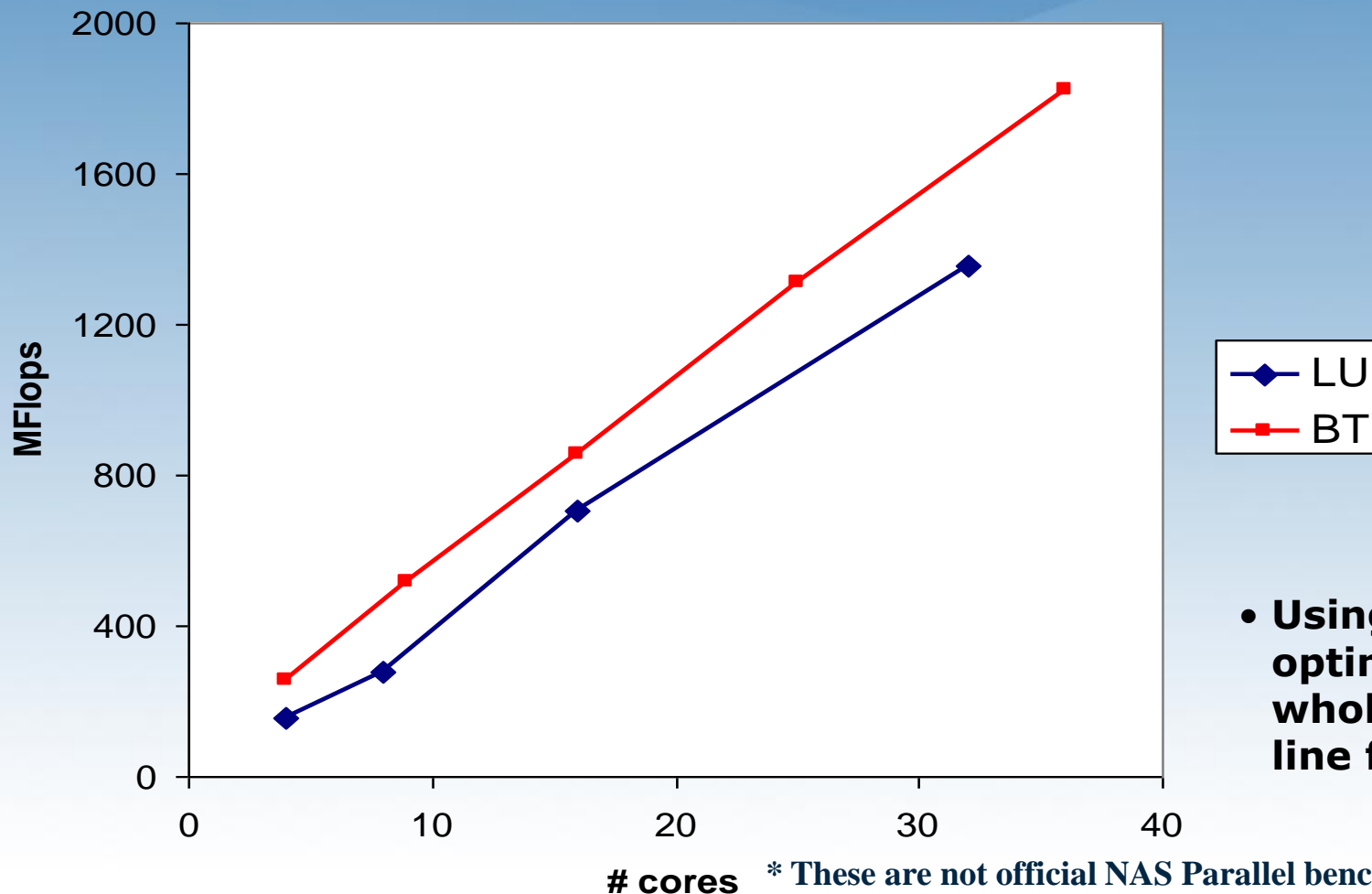
Third party names are the property of their owners.



Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference <http://www.intel.com/performance> or call (U.S.) 1-800-628-8686 or 1-916-356-3104.

LU/BT NAS Parallel Benchmarks, SCC

Problem size: Class A, 64 x 64 x 64 grid*



- Using latency optimized, whole cache line flags

* These are not official NAS Parallel benchmark results.

SCC processor 500MHz core, 1GHz routers, 25MHz system interface, and DDR3 memory at 800 MHz.

Third party names are the property of their owners.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference <http://www.intel.com/performance> or call (U.S.) 1-800-628-8686 or 1-916-356-3104.



Agenda

- Views of SCC: HW, SW and Platforms
- RCCE: A communication environment for application programmers.
- Benchmarks and Results
- • Power management

RCCE Power Management API

- RCCE power management emphasizes safe control: V/GHz changed together within each 4-tile (8-core) power domain.
 - A Master core sets V + GHz for all cores in domain.
 - > RCCE_istep_power():
 - steps up or down V + GHz, where GHz is max for selected voltage.
 - > RCCE_wait_power():
 - returns when power change is done
 - > RCCE_step_frequency():
 - steps up or down only GHz- Power management latencies
 - V changes: Very high latency, $O(\text{Million})$ cycles.
 - GHz changes: Low latency, $O(\text{few})$ cycles.

Conclusions

- RCCE software works
 - RCCE's restrictions (Symmetric MPB memory model and blocking communications) have not been a fundamental obstacle
 - Functional emulator is a useful development/debug device
- SCC architecture
 - The on-chip MPB was effective for scalable message passing applications
 - Software controlled power management works ... but it's challenging to use because (1) granularity of 8 cores and (2) high latencies for voltage changes
 - The Test&set registers (only one per core) will be a bottleneck.
 - > Sure wish we had asked for more!
- Future work
 - Add shmalloc() to expose shared off-chip DRAMM (in progress).
 - Move resource management into OS/drivers so multiple apps can work together safely.
 - We have only just begun to explore power management capabilities ... we need to explore additional usage models.

SW Acknowledgements

- **SCC System software:**

Management Console software

Michael Riepen

BareMetalC workflow

Michael Riepen

Linux for SCC

Thomas Lehnig

Paul Brett

System Interface FPGA development

Matthias Steidl

TCP/IP network driver

Werner Haas

- **SCC Application software:**

RCCE library and apps

Rob Van der Wijngaart

Tim Mattson

Developer tools

Patrick Kennedy

(Intel compilers and math libraries)

Mandelbrot app + viz

Michael Riepen

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

Software Managed Coherency on SCC

Sai Luo, Xiaocheng Zhou, Tiger Hu Chen,
Shoumeng Yan, Ying Gao
Intel Labs China

Wei Liu, Brian Lewis, Bratin Saha
Programming Systems Lab



Revive an old topic: cache coherence ?

- Software-managed coherence was a popular topic
 - Been around at least a couple of decades
 - Mostly targeting multiprocessors or clusters of workstations

Revive an old topic: cache coherence ?

- Software-managed coherence was a popular topic
 - Been around at least a couple of decades
 - Mostly targeting multiprocessors or clusters of workstations
- World is changing
 - Many cores on a single die
 - Much higher bandwidth and lower latency
 - Running out of power budget

Revive an old topic: cache coherence ?

- Software-managed coherence was a popular topic
 - Been around at least a couple of decades
 - Mostly targeting multiprocessors or clusters of workstations
- World is changing
 - Many cores on a single die
 - Much higher bandwidth and lower latency
 - Running out of power budget
- World is *not* changing
 - Legacy code written in shared memory programming model
 - Coherent memory requirement from ISVs

Revive an old topic: cache coherence ?

- Software-managed coherence was a popular topic
 - Been around at least a couple of decades
 - Mostly targeting multiprocessors or clusters of workstations
- World is changing
 - Many cores on a single die
 - Much higher bandwidth and lower latency
 - Running out of power budget
- World is *not* changing
 - Legacy code written in shared memory programming model
 - Coherent memory requirement from ISVs

What is the right trade-off: HW vs. SW?

Why Software-Managed Coherency?

(Why not hardware)

- No or minimal hardware!
 - Limited power budget on many-core
 - High complexity and validation effort to support hardware cache coherence protocol

Why Software-Managed Coherency?

- No or minimal hardware! **(Why not hardware)**
 - Limited power budget on many-core
 - High complexity and validation effort to support hardware cache coherence protocol
- Flexibility: Dynamic reconfigurable coherency domains
 - Multiple applications running in separate coherency domains
 - Good match to SCC
 - Enable more optimizations: load balancing etc.

Why Software-Managed Coherency?

- No or minimal hardware! **(Why not hardware)**
 - Limited power budget on many-core
 - High complexity and validation effort to support hardware cache coherence protocol
- Flexibility: Dynamic reconfigurable coherency domains
 - Multiple applications running in separate coherency domains
 - Good match to SCC
 - Enable more optimizations: load balancing etc.
- Emerging applications
 - Most data are RO-shared, few are RW-shared
 - Coarse-grained synchronization: Map-Reduce, BSP, etc

Why Software-Managed Coherency?

(Why not hardware)

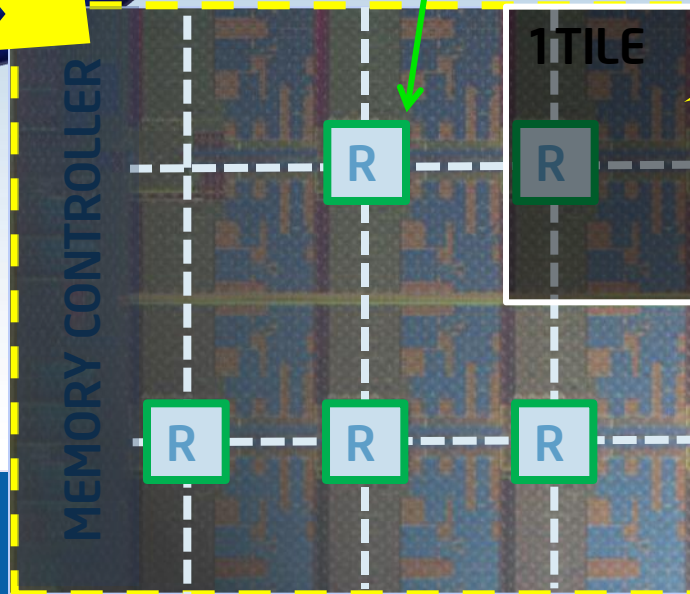
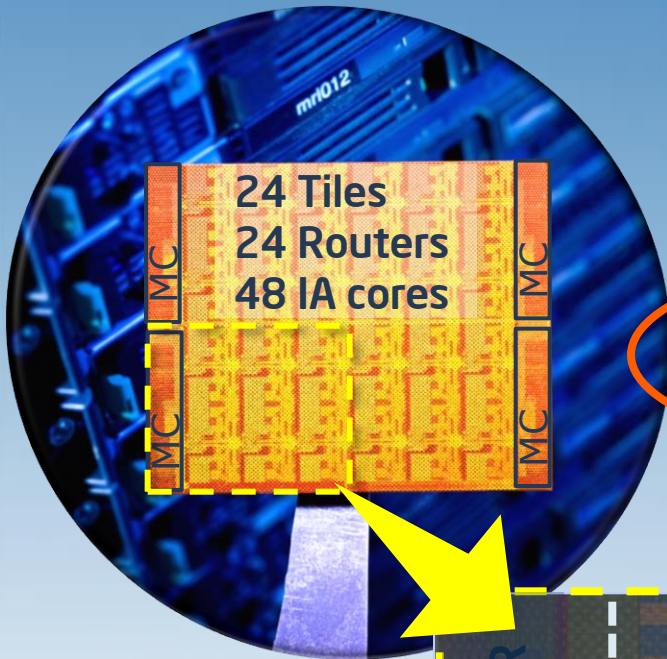
- No or minimal hardware!
 - Limited power budget on many-core
 - High complexity and validation effort to support hardware cache coherence protocol
- Flexibility: Dynamic reconfigurable coherency domains
 - Multiple applications running in separate coherency domains
 - Good match to SCC
 - Enable more optimizations: load balancing etc.
- Emerging applications
 - Most data are RO-shared, few are RW-shared
 - Coarse-grained synchronization: Map-Reduce, BSP, etc

SW-managed coherency can achieve comparable performance

SCC architecture, a brief overview

- 45nm Hi-K metal-gate silicon
- 48 IA cores
- 6x4 2D mesh network
- 4 DDR3 memory controllers
- On-die message buffers
- **No hardware cache coherency**

Dual-core Tile



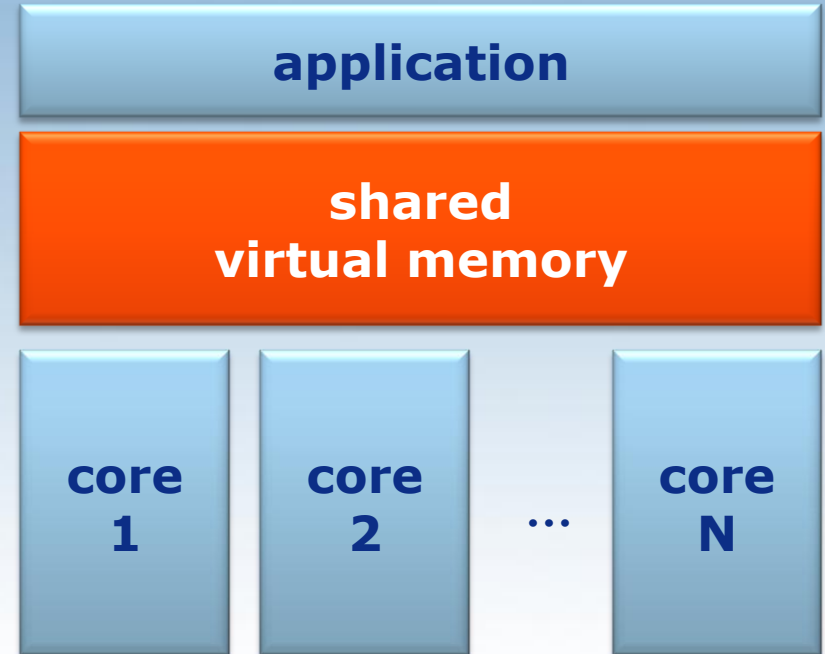
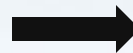
Outline

- Motivation
- Overview of SW managed coherence
- Implementation and Optimizations
- Our results
- Challenges for future research

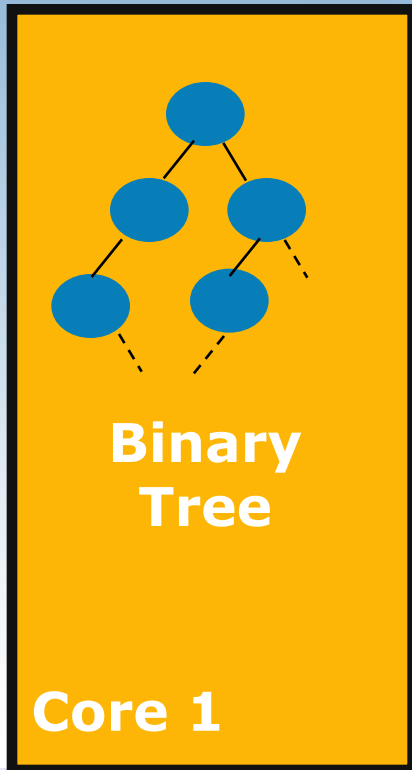
Overview

- Shared virtual memory can be used to support coherency
 - Similar to DSM
 - A single shared memory view to applications
 - Seamlessly sharing data structure and pointers among multiple cores
- No special HW support is needed.

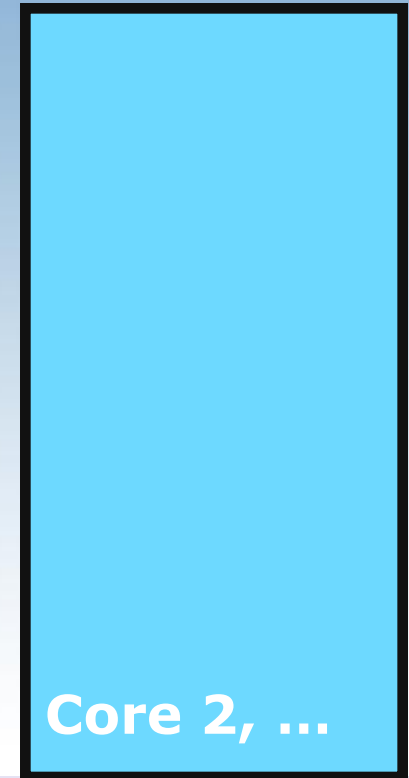
Cores in SCC have separate address spaces



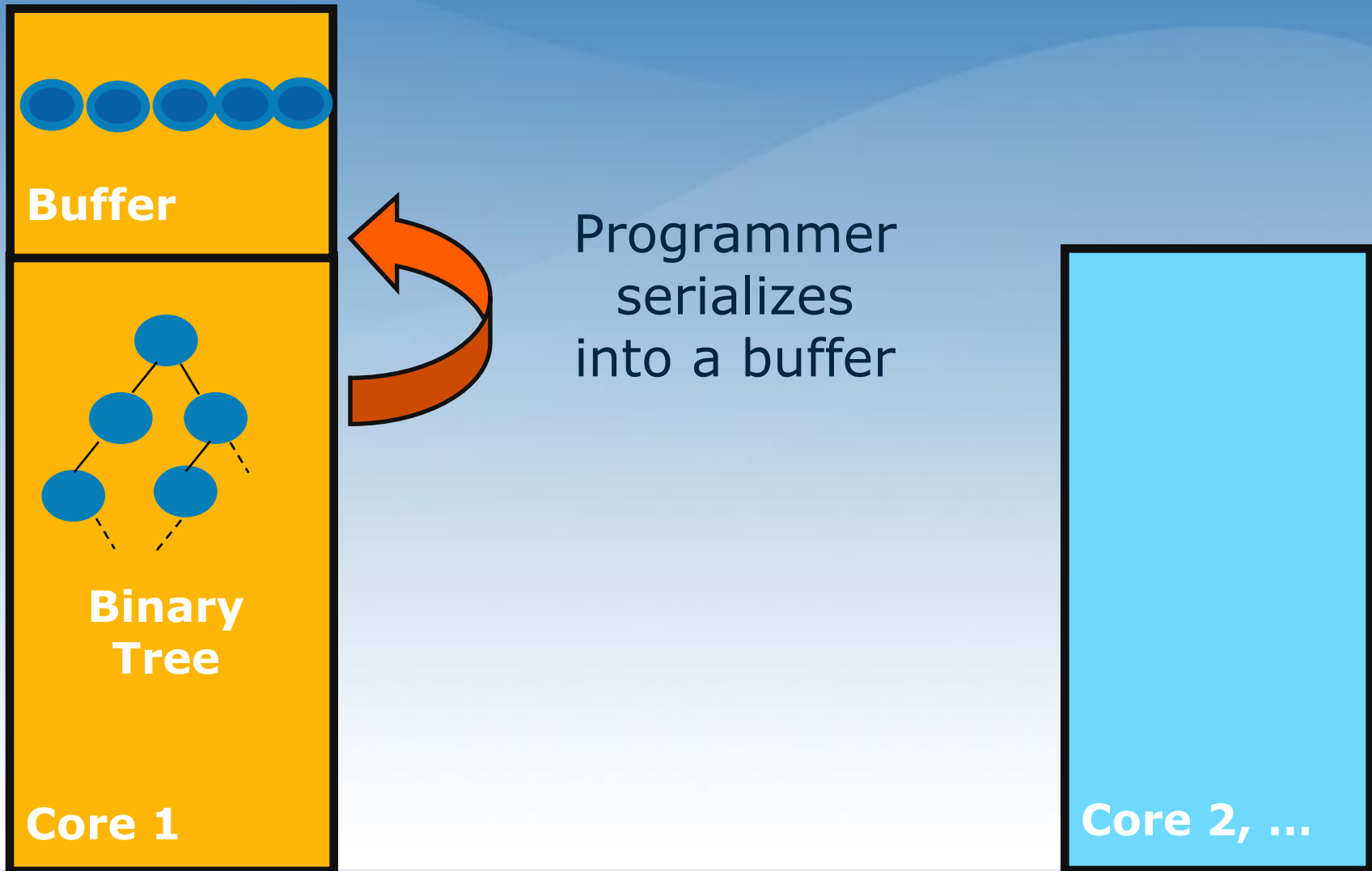
Why Shared Virtual Memory?



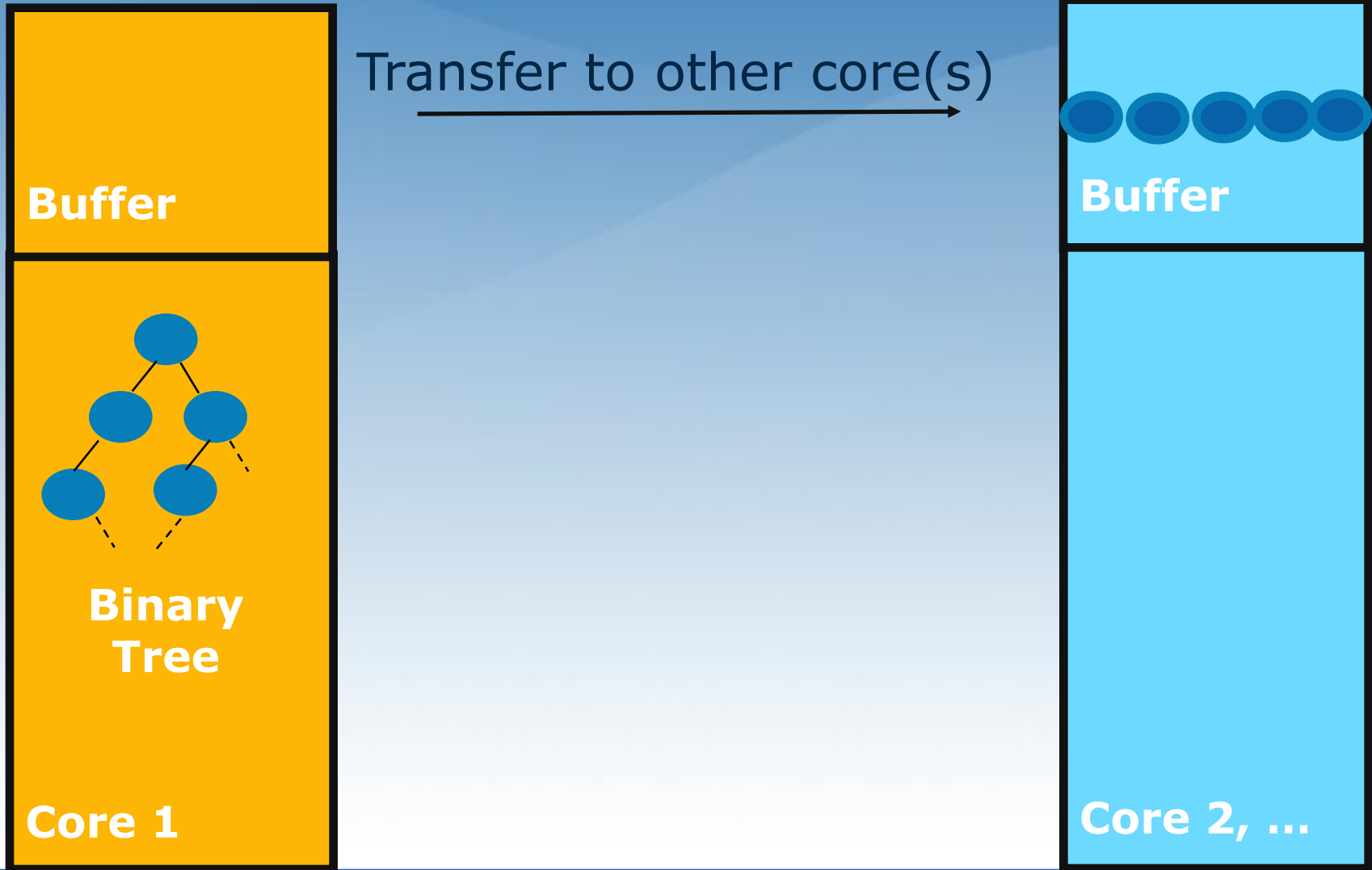
**Separate
Memory Spaces
without Coherency**



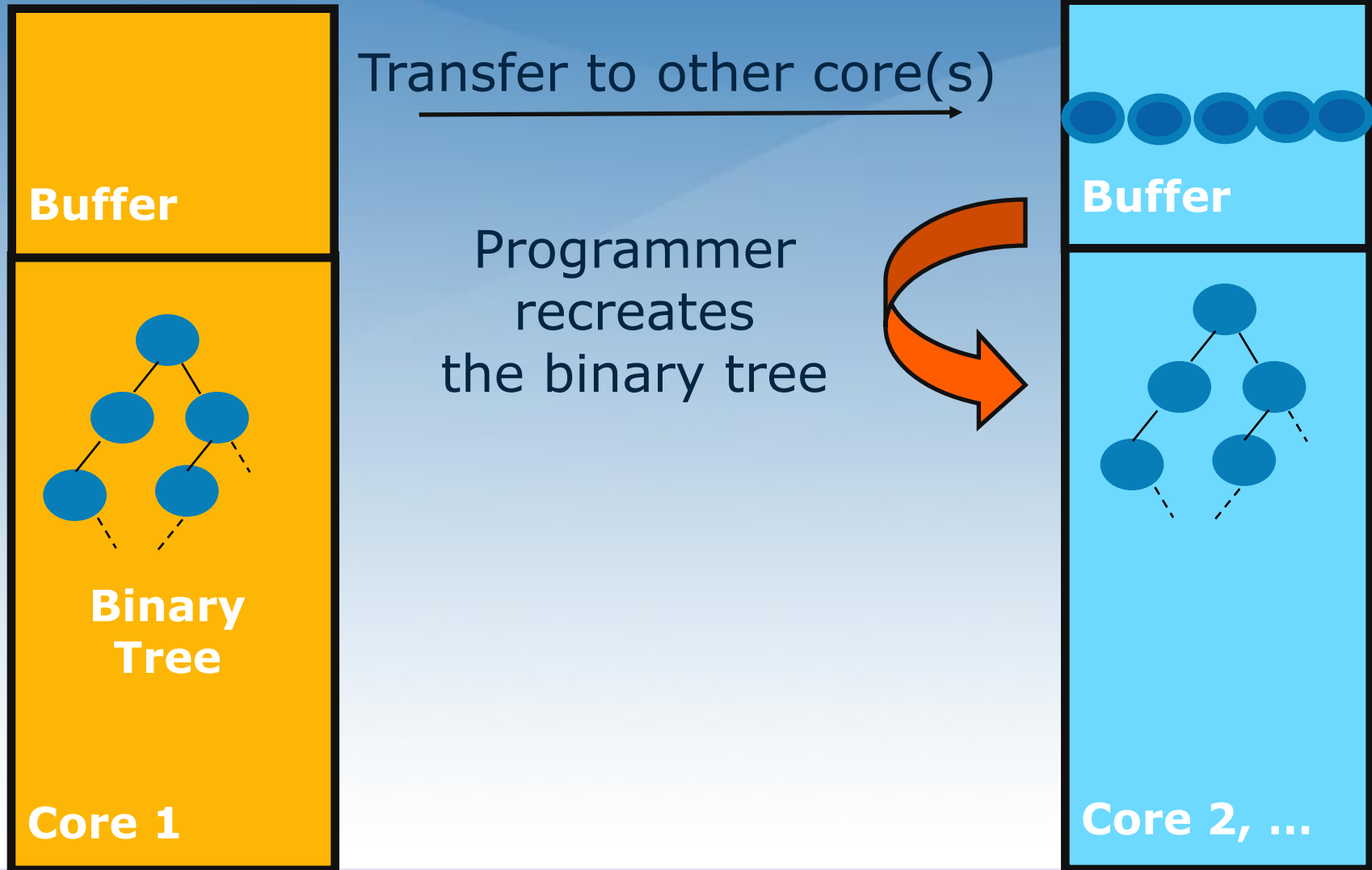
Why Shared Virtual Memory?



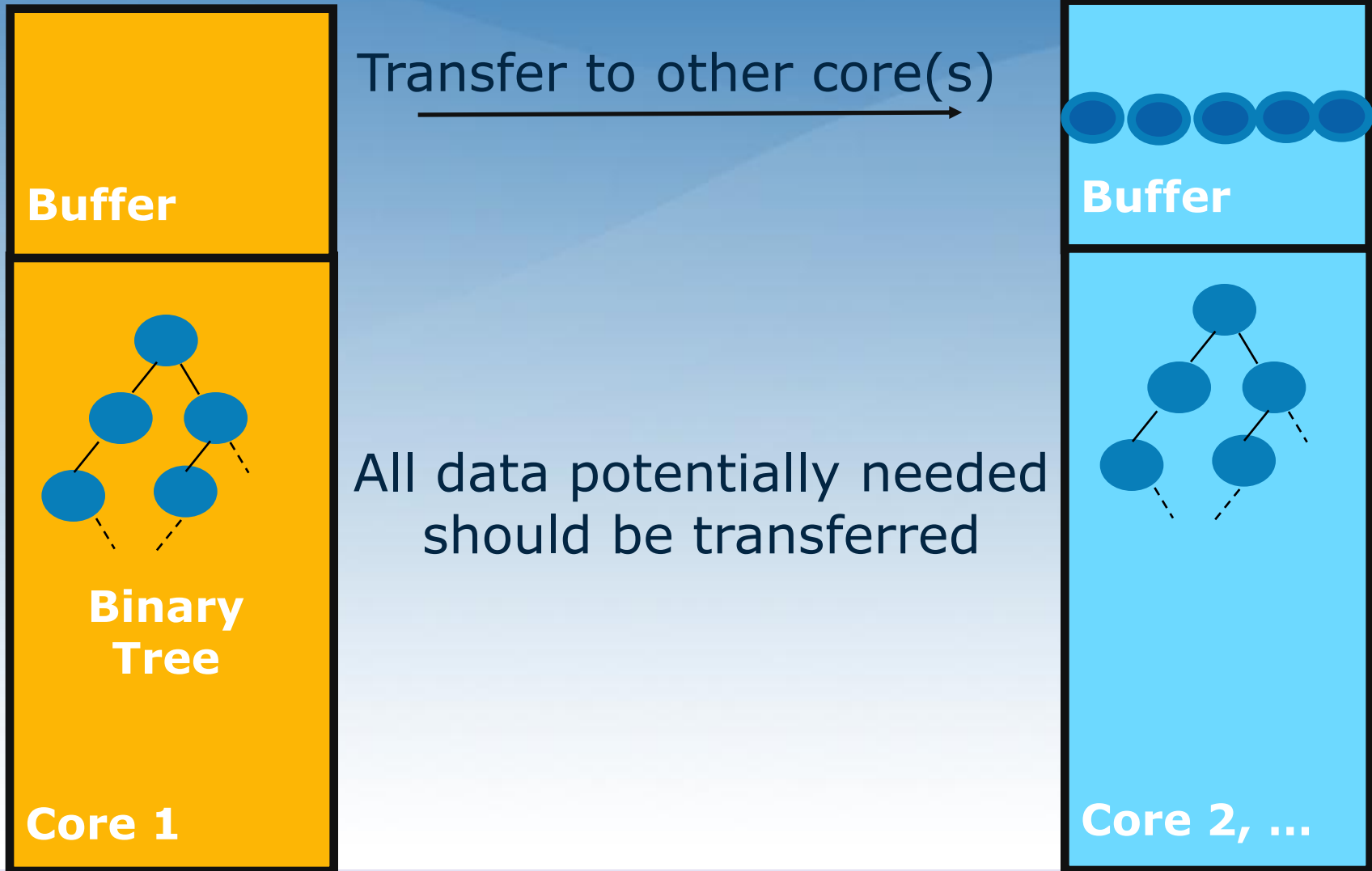
Why Shared Virtual Memory?



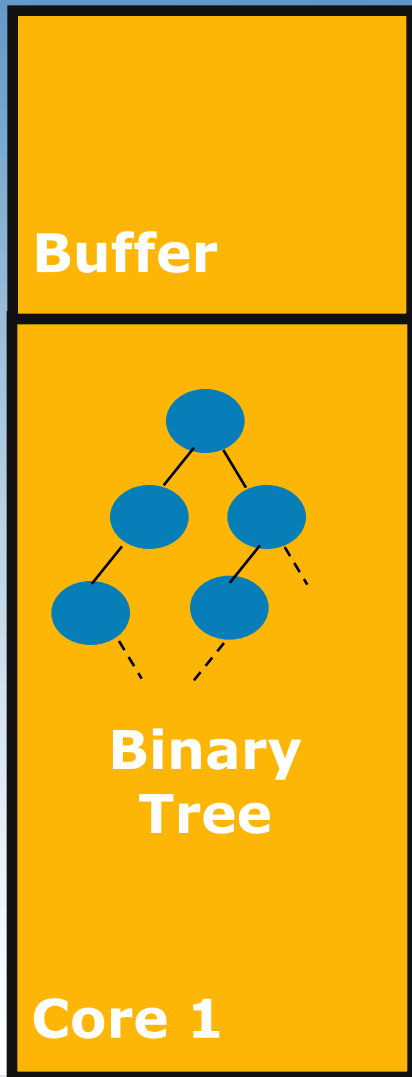
Why Shared Virtual Memory?



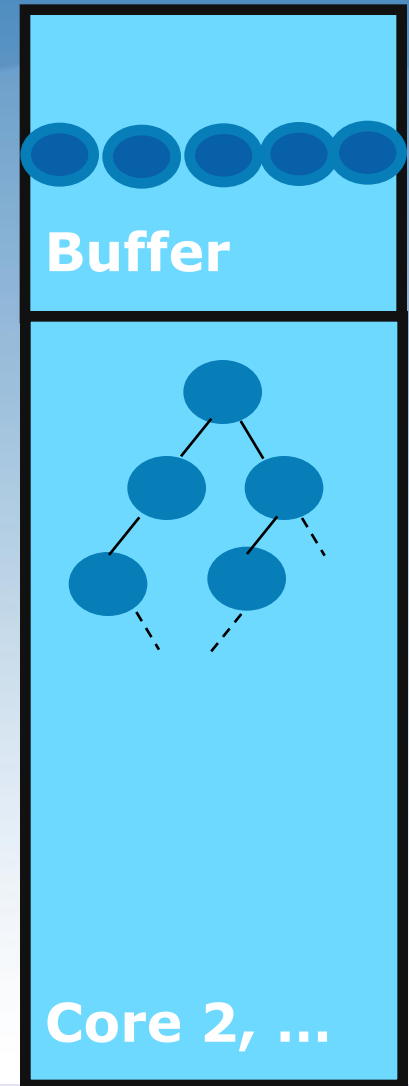
Why Shared Virtual Memory?



Why Shared Virtual Memory?



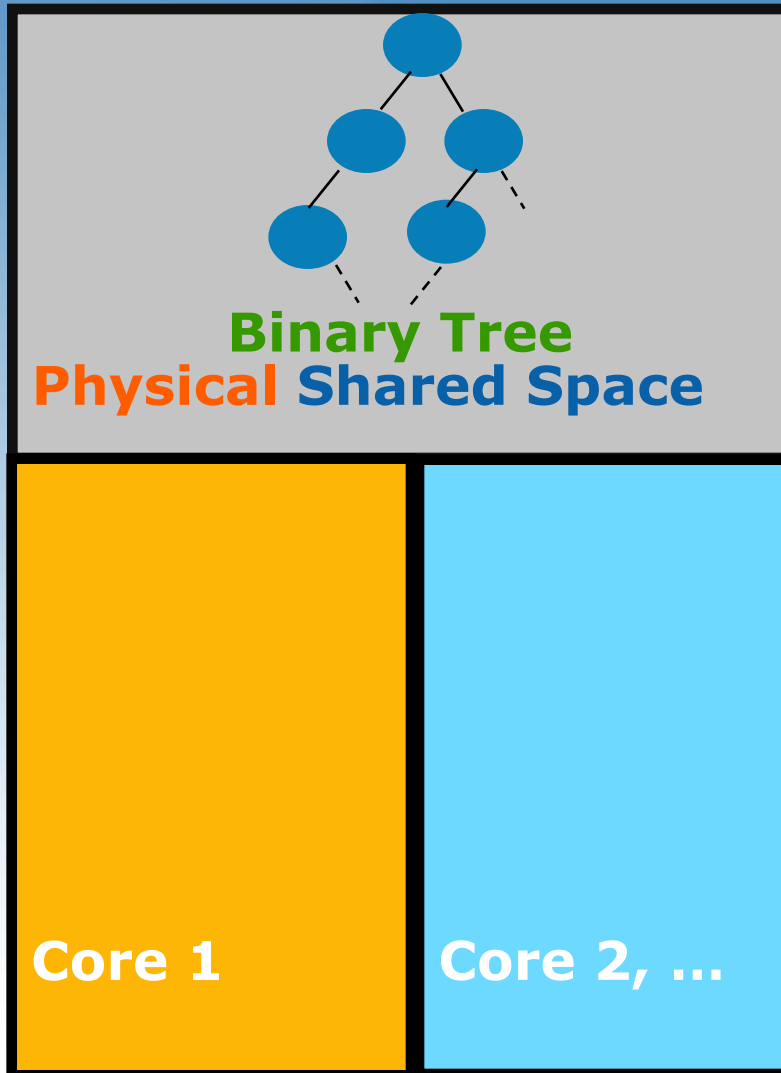
Transfer to other core(s)



Even worse, what to do if one node is modified at one core?

Why Shared Virtual Memory? (Cont.)

Or



Explicit data management goes away

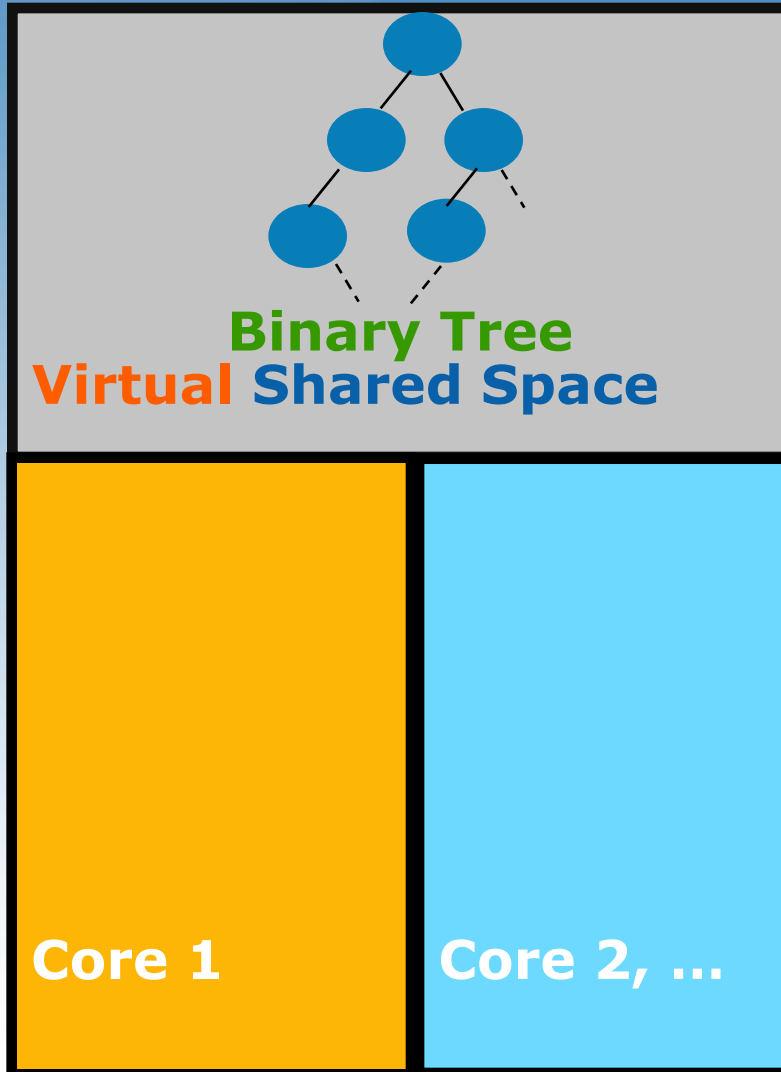
Only data really needed are accessed

But:
SCC has no hardware cache coherency,
So the shared space must not be cached

It is a performance hit

Why Shared Virtual Memory? (Cont.)

How
about



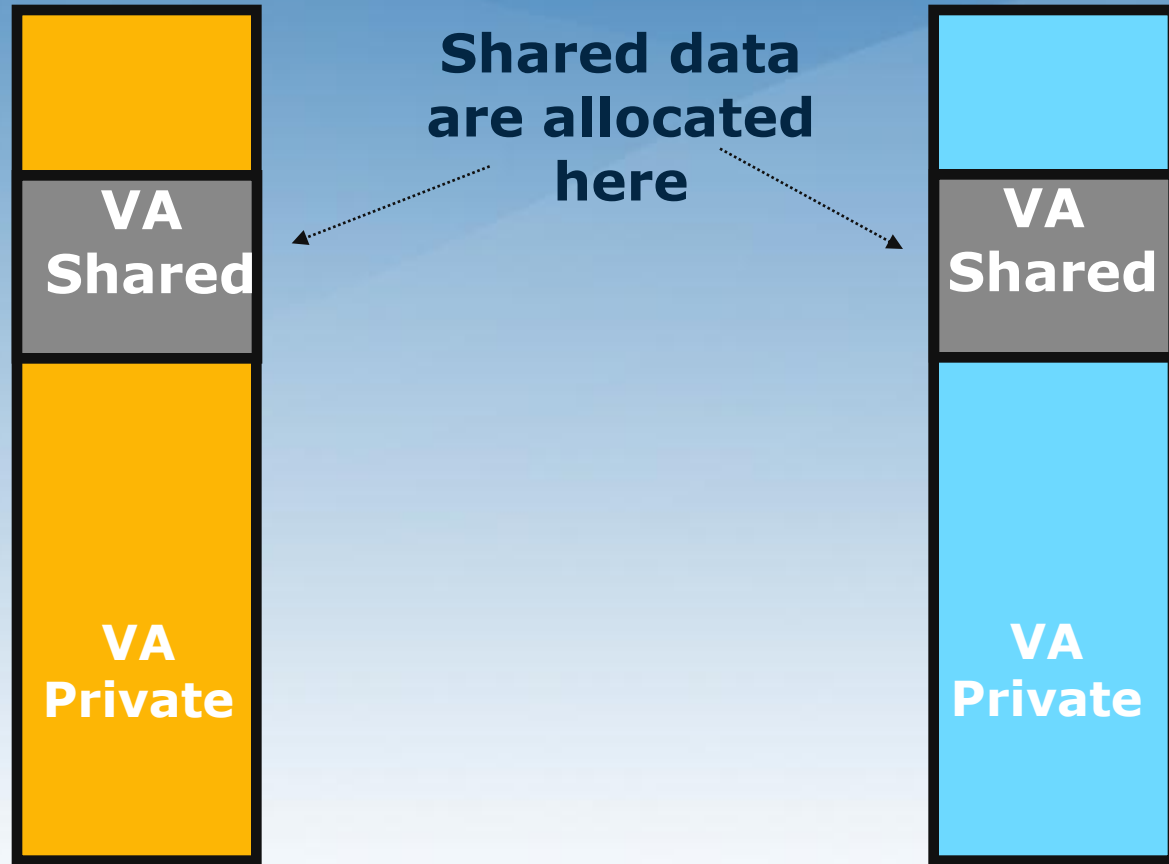
Shared data are allocated in the shared virtual address space

They are cacheable (higher performance)

Data coherency are managed by software

Users don't care about where the data locate and how many copies exist

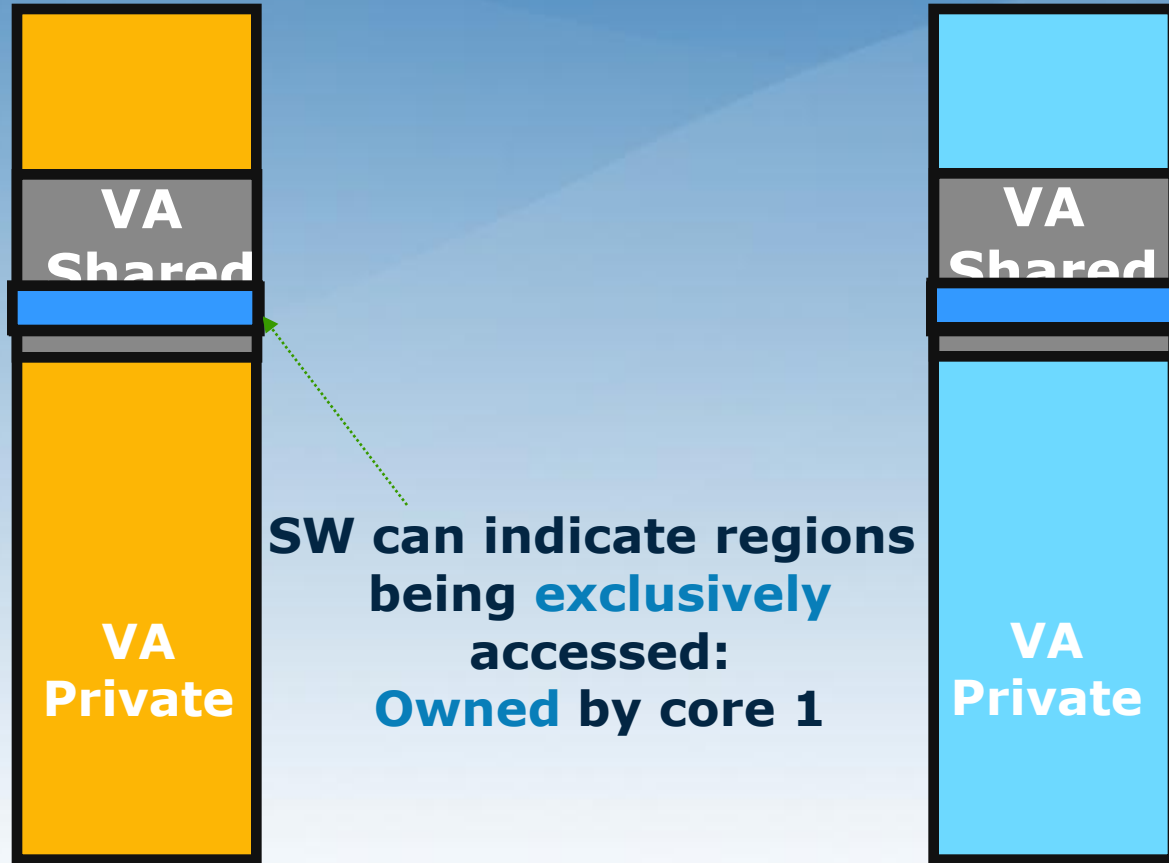
Shared Virtual Memory Model



Core 1 virt addr space

Core 2 virt addr space

Shared Virtual Memory Model

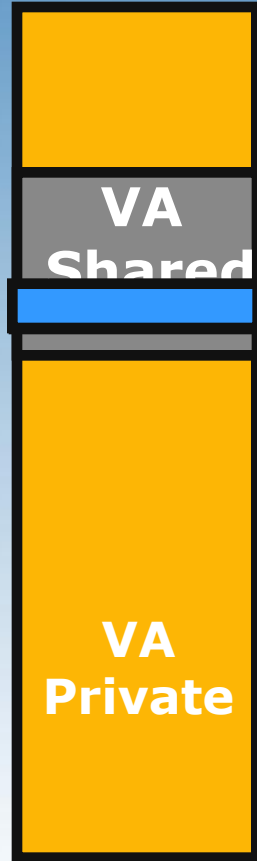


Core 1 virt addr space

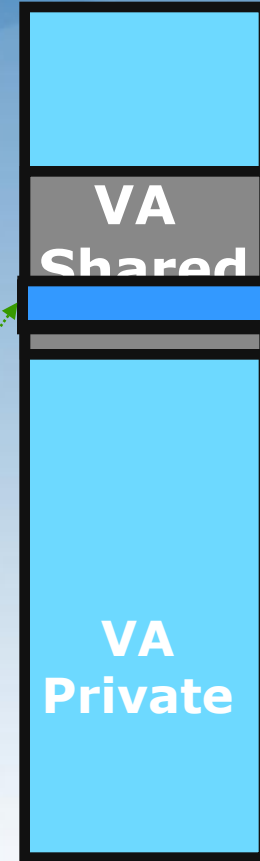
Core 2 virt addr space

SW can indicate regions being **exclusively** accessed:
Owned by core 1

Shared Virtual Memory Model

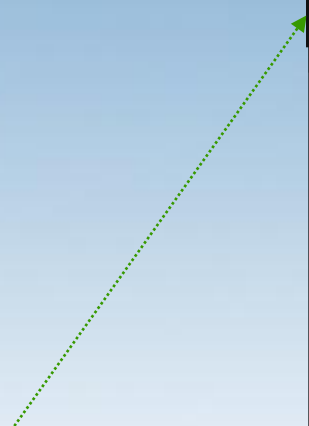


Core 1 virt addr space

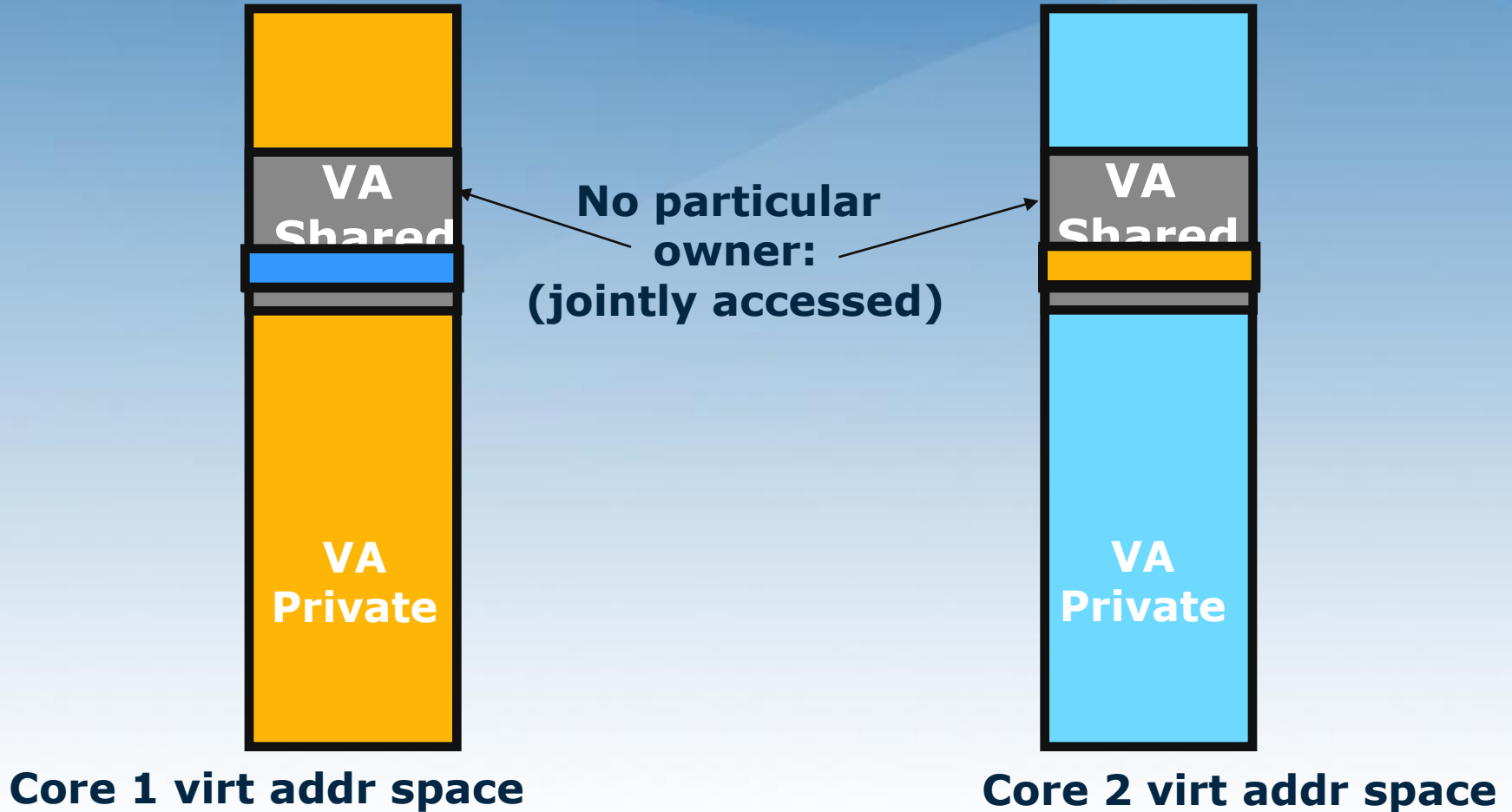


Core 2 virt addr space

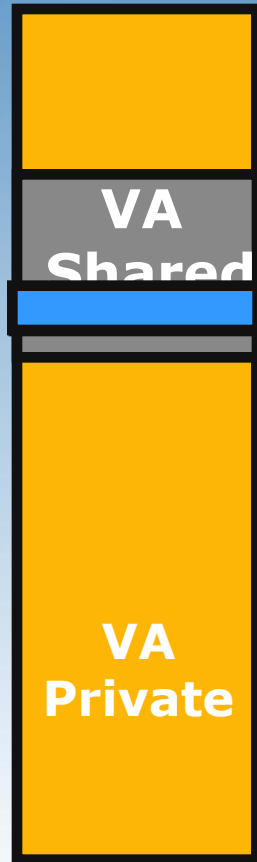
These regions can be handed over:
Owned by core 2



Shared Virtual Memory Model



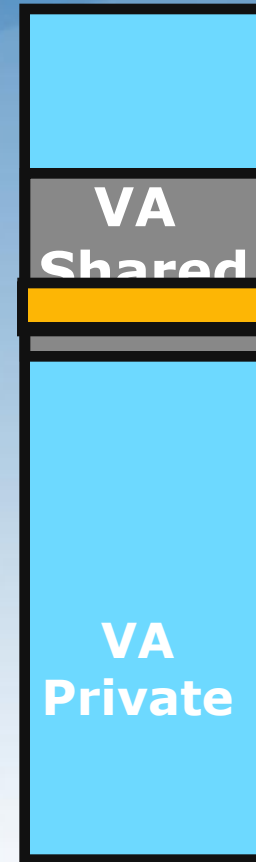
Shared Virtual Memory Model



Core 1 virt addr space

Partially shared
Release consistency
Ownership

Cut down
coherent overhead



Core 2 virt addr space

Outline

- Motivation
- Overview of SW managed coherence
- Implementation and Optimizations
- Our results
- Challenges for future research

Language and Compiler Support

- New “shared” type qualifier

- > *shared* int a; //a shared variable
 - > *shared* int* pa; //a pointer to a shared int
 - > *shared* int* *shared* pb; //a shared pointer to a shared int

- Static checking rules enforced by the compiler

- > No sharing between stack variables



- foo() {*shared* int c;}

- > Shared pointer can't point to private data



- int* *shared* pc;

- > And more on pointer assignment and casting etc.

Runtime Support

- Partial sharing on page-level
 - Only those actually shared are subjected to consistency maintenance

Runtime Support

- Partial sharing on page-level
 - Only those actually shared are subjected to consistency maintenance
- Release consistency model
 - Consistency only guaranteed at the sync points (release, acquire)
 - > Significantly reduce coherence traffic
 - Many applications already follow RC model
 - > E.g. sync points: pthread_create, mutex, barrier, ...
 - > Release/Acquire can be inserted automatically at these points

Runtime Support

- Partial sharing on page-level
 - Only those actually shared are subjected to consistency maintenance
- Release consistency model
 - Consistency only guaranteed at the sync points (release, acquire)
 - > Significantly reduce coherence traffic
 - Many applications already follow RC model
 - > E.g. sync points: pthread_create, mutex, barrier, ...
 - > Release/Acquire can be inserted automatically at these points
- Ownership rights
 - No coherence traffic until ownership changed
 - They are treated as hints (i.e. optimization opportunities)
 - > Fault on touch: fault if touch something owned by others
 - > Promote on touch: promote to “jointly accessible”

Object Collision Detection Example: Share Memory Approach

```
struct Ball {  
    Vector position, velocity;  
    int area_id;  
    shared struct Ball* next; // balls in the area  
};
```

```
shared struct Ball* areas[N];
```

```
void collision(shared struct Ball* all) {  
    // do collision detection  
    // and compute the new position/velocity  
    .....  
}
```

```
void simulate()  
{  
    for(i=0; i<N; i++)  
        thd[i] = spawn(collision, areas[i]);  
    for(i=0; i<N; i++)  
        join(thd[i]);  
    update_area_array();  
}
```

Object Collision Detection Example: Share Memory Approach

```
struct Ball {  
    Vector position, velocity;  
    int area_id;  
    shared struct Ball* next; // balls in the area  
};
```

```
shared struct Ball* areas[N];
```

```
void collision(shared struct Ball* all) {  
    // do collision detection  
    // and compute the new position/velocity  
    .....  
}
```

```
void simulate()  
{  
    for(i=0; i<N; i++)  
        thd[i] = spawn(collision, areas[i]);  
    for(i=0; i<N; i++)  
        join(thd[i]);  
    update_area_array();  
}
```

- It's just like writing a pthread program
- Implicit sync points at spawn, join, the beginning and ending of collision()

Example: Message Passing Approach

```
typedef struct Ball Ball;
struct Ball {
    Vector position, velocity;
    int area_id;
    Ball* next; // balls in the same area
};
Ball* areas[N];

void collision(int id)
{
    // receive the data objects
    // and recreate the structure
    for(i=0; i<N; i++) {
        areas[i] = NULL;
        while(recv(id, buf)) {
            b = malloc(sizeof(Ball));
            *b = *buf;
            b->next = areas[i];
            areas[i] = b;
        }
    }

    // do collision detection
    // and compute the new pos/vel
    .....

    // send back new data
    // and free the local objects
    for(b=all; b; b=next) {
        new_id = get_area_id(b);
        send(new_id, b);
        next = b->next; free(b);
    }
}

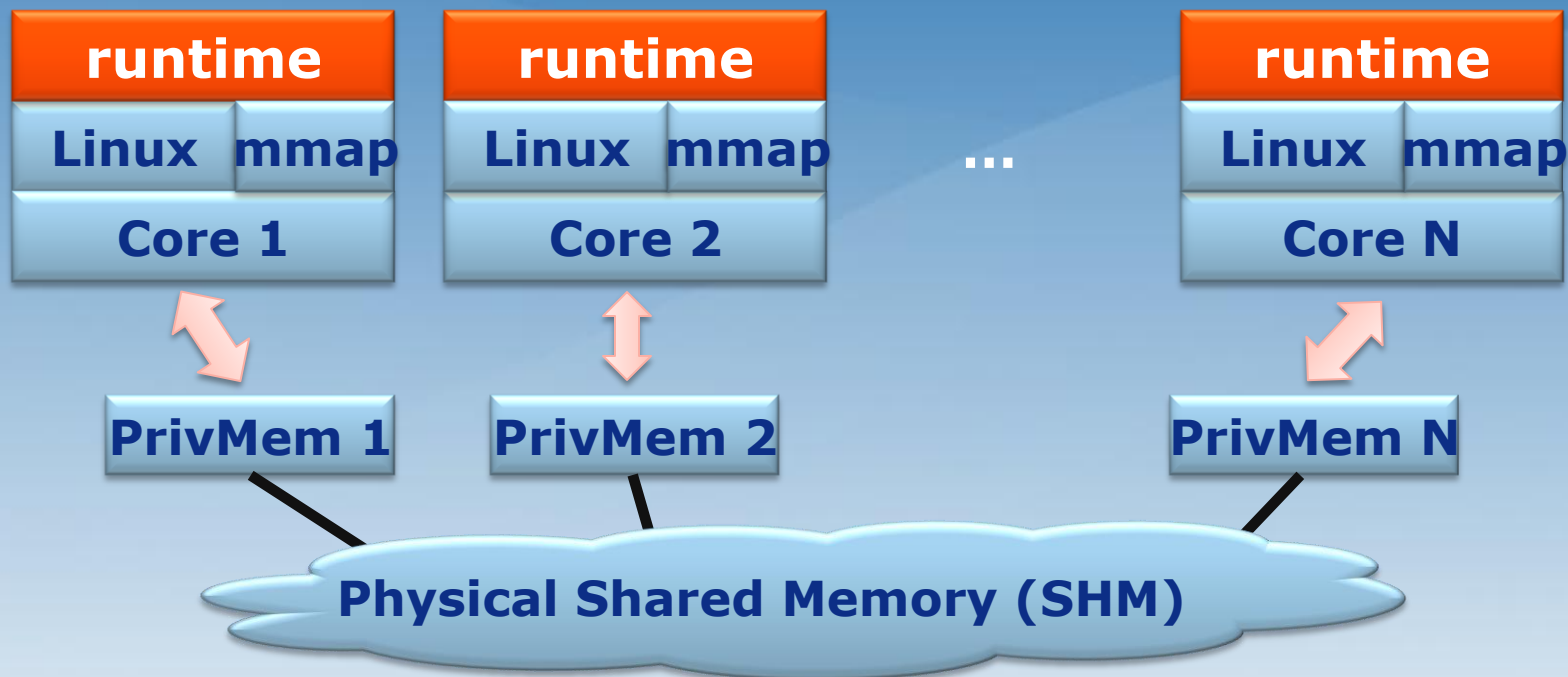
void simulate()
{
    // spawn
    for(i=0; i<N; i++)
        thd[i] = spawn(collision, i);
    // send data to the individual threads
    // and destroy the objects

    for(i=0; i<N; i++) {
        for(b=areas[i]; b; b=next) {
            for(j=0; j<N; j++) send(j, b);
            next = b->next; free(b);
        }
    }

    // gather data back
    // and recreate the link list
    for(i=0; i<N; i++) {
        areas[i] = NULL;
        while(recv(id, buf)) {
            b = malloc(sizeof(Ball));
            *b = *buf;
            b->next = areas[i];
            areas[i] = b;
        }
    }
    // join
    for(i=0; i<N; i++)
        join(thd[i]);
}
}
```

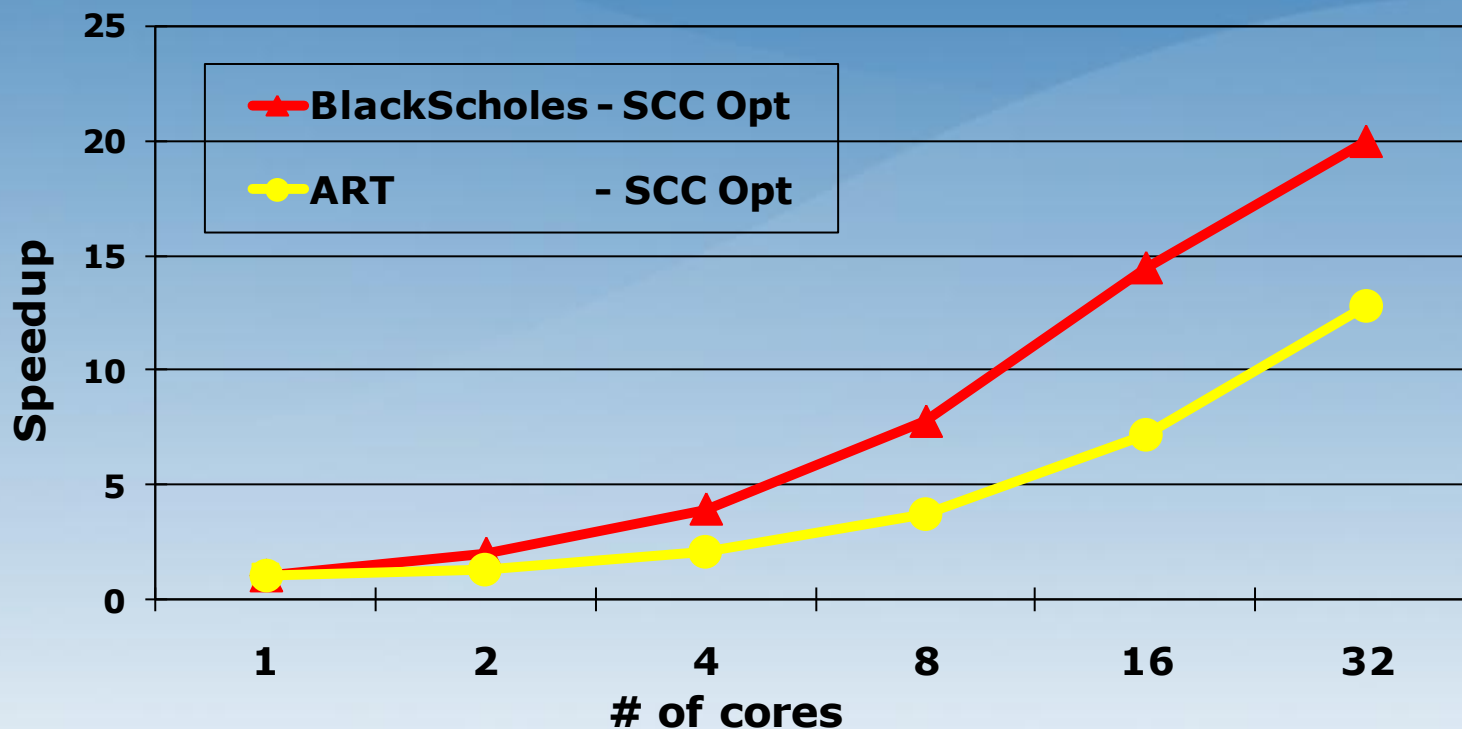
- Lots of code are spent in data serialization and reconstruction.
- Is error-prone and might dead-lock.
- All data are sent even not used.

Optimized implementation for SCC



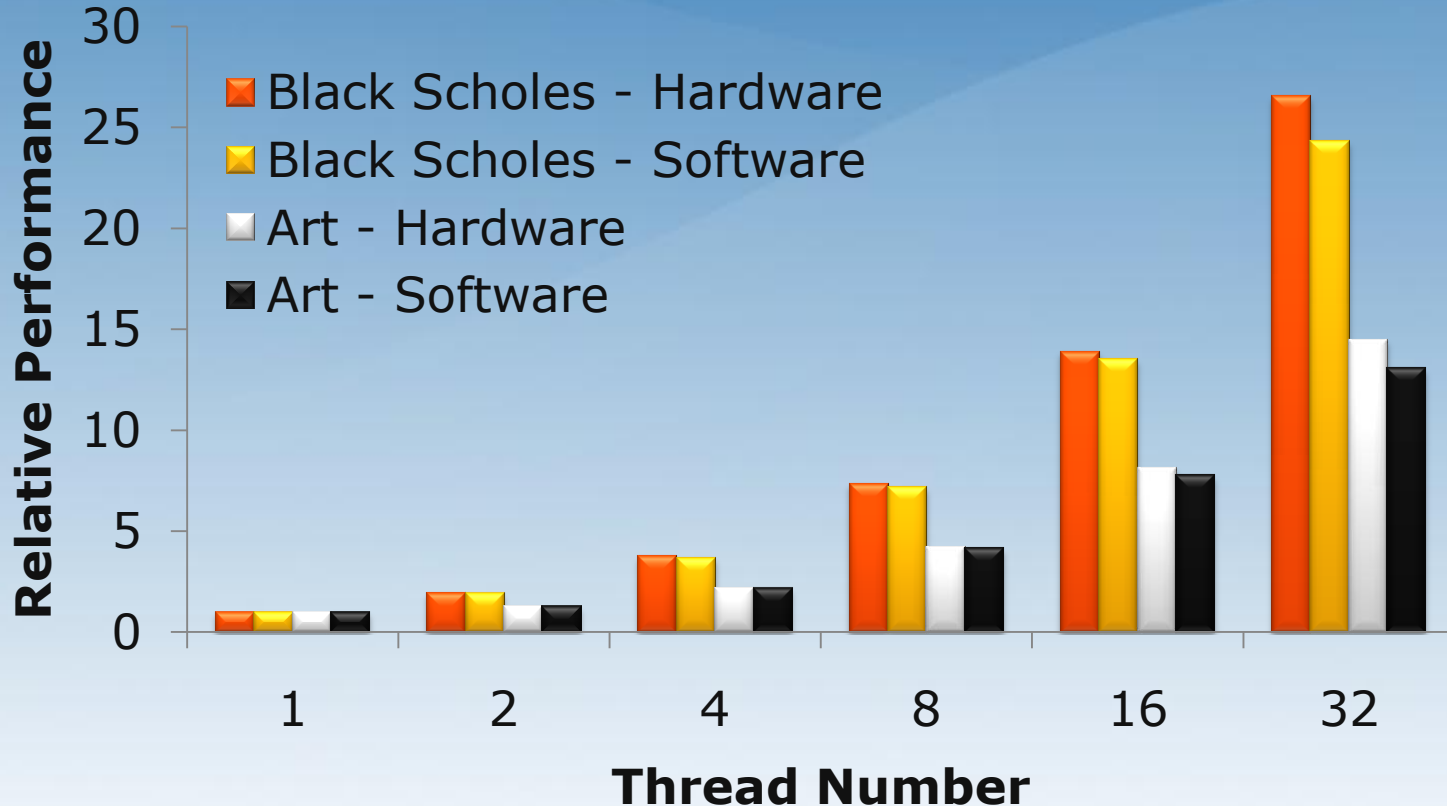
- Leverage shared memory (SHM) support in SCC
- Golden copy is saved at SHM, needn't communicate with any other nodes
- Do memcpy between cacheable private memory & uncacheable SHM

Scalability on SCC



- Significantly improved scalability, up to 20X on 32 cores.
- More optimizations (WIP)

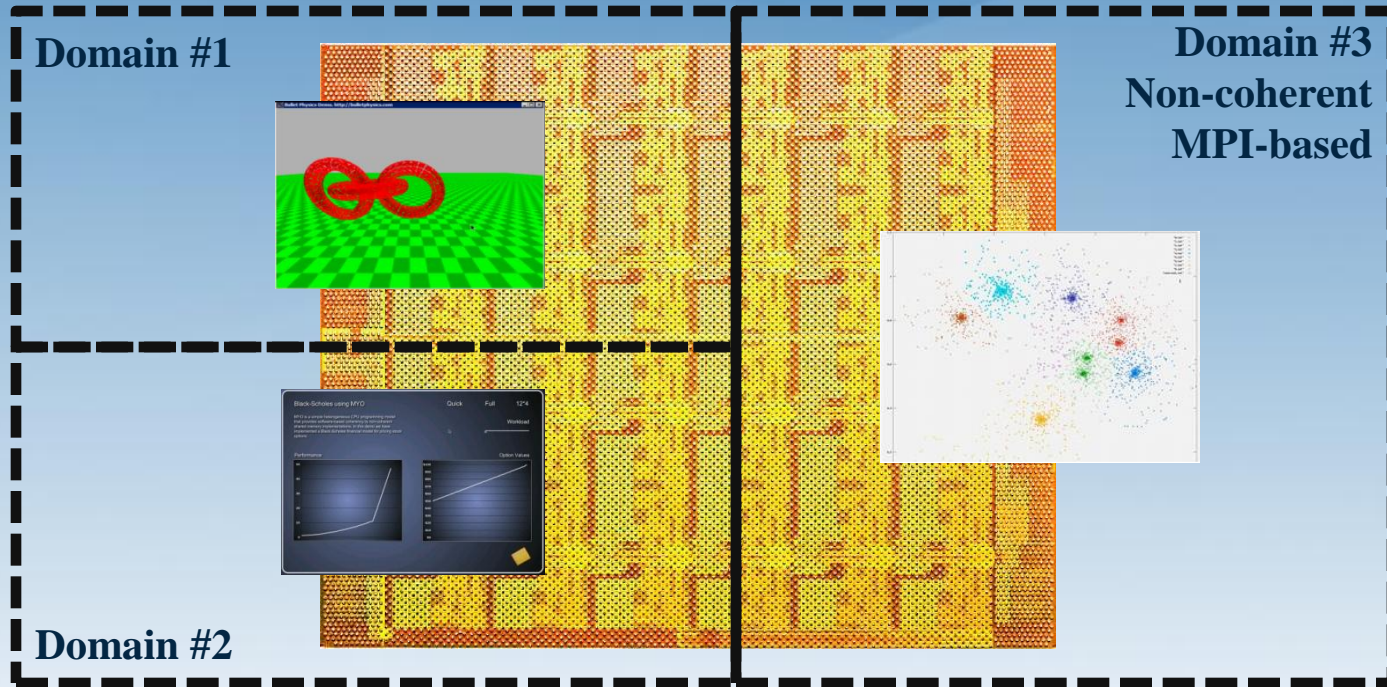
SW managed coherence vs. HW coherence on 32way SMP server (process per core)



- Software managed coherency is as efficient as hardware cache coherency

Emerging usage models

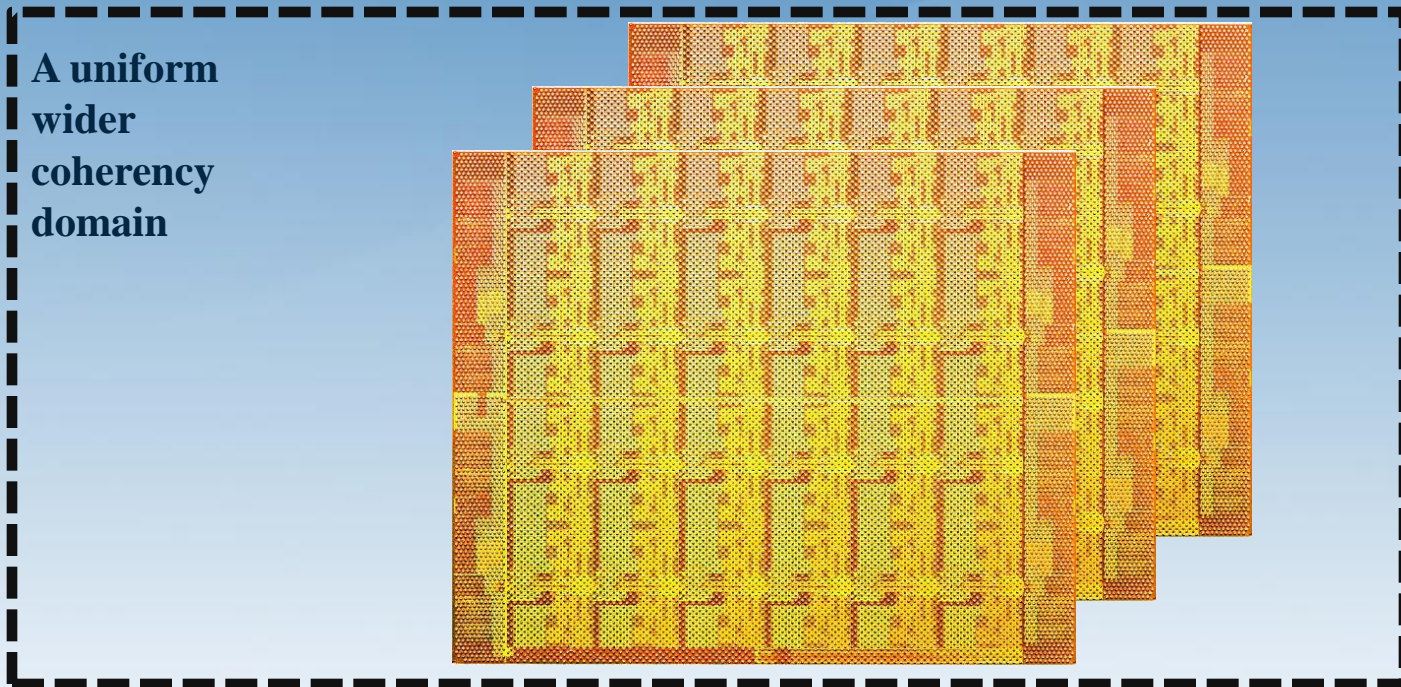
- Separated coherency domains



- Whole system partitioned into multiple coherency domains
- Dynamic reconfigurable
- Mixed mode: share memory in one domain with MPI in others

Another usage models

- Multiple SCC chips



- When an application is massively parallel, more SCC chips can be connected together to form a uniform wider coherency domain

Summary

- We believe software managed coherency on non-coherent many-core is the future trend
- A prototyped partially shared virtual memory system demonstrates it can be:
 - Easy to program
 - Comparable performance vs. hardware coherence
 - Adaptive to future advanced usage models
- Also opens new research opportunities

Challenges for future research

- This revived “software managed coherency” topic opens many “cold cases”
- What are the right software optimizations?
 - Prefetching, locality, affinity, consistency model
 - And more...
- What is the right hardware support?
- How do emerging workloads adapt to this?

Please contact us if you are interested in this topic.

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

JavaScript Farm on SCC

Adam Welc, Richard L. Hudson,
Tatiana Shpeisman, Ali-Reza Adl-Tabatabai

Programming Systems Lab



ParWeb Project

- Today's web applications
 - Becoming more and more sophisticated and complex (productivity suites, photo editing, games, etc.)
 - Still largely sequential, despite availability of multi-core and many-core, due to limitations of programming tools and support infrastructure
- Project goals
 - Flexible and expressive parallel programming model
 - Utilize all available resources to maximum

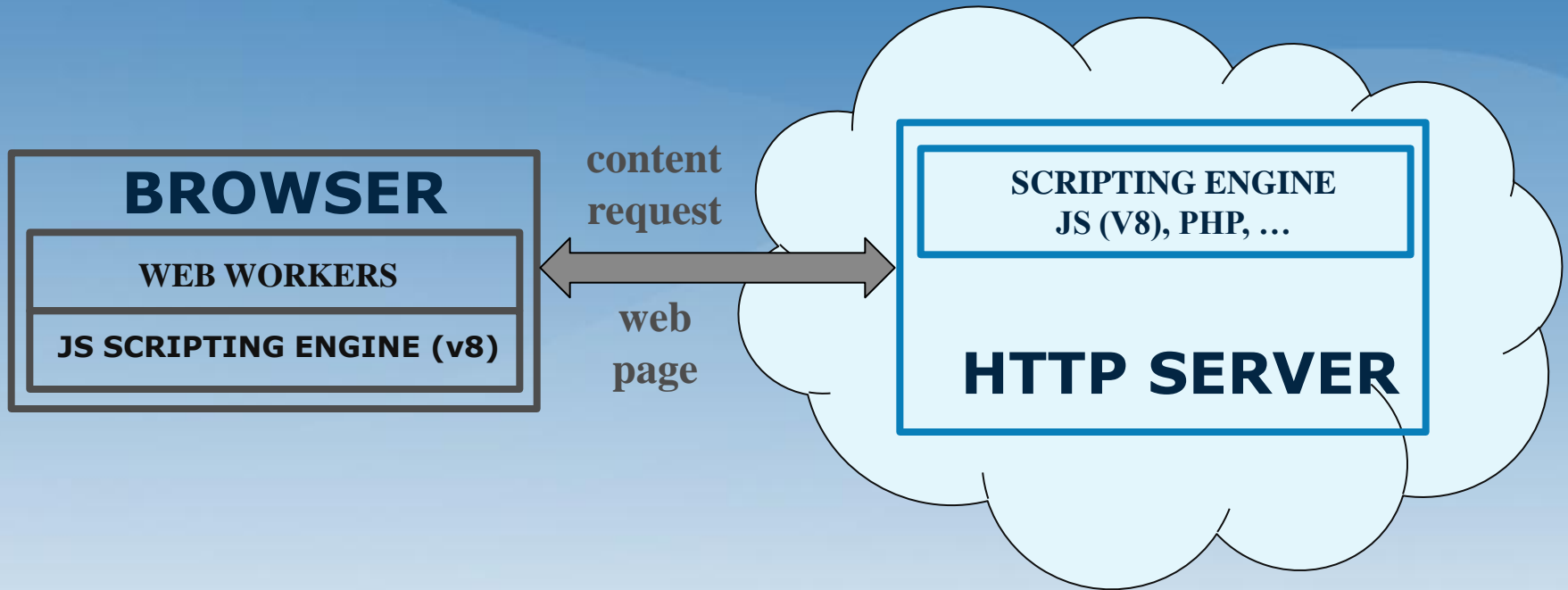
JavaScript

- Object-oriented dynamically typed scripting language
- Designed for productivity
 - One of the most popular web technologies
 - One of the most popular scripting languages
- Limited support for parallelism
 - Web workers (in HTML 5) designed to increase GUI responsiveness
 - Web workers can communicate with HTTP servers via message passing

Parallelizing JavaScript on SCC

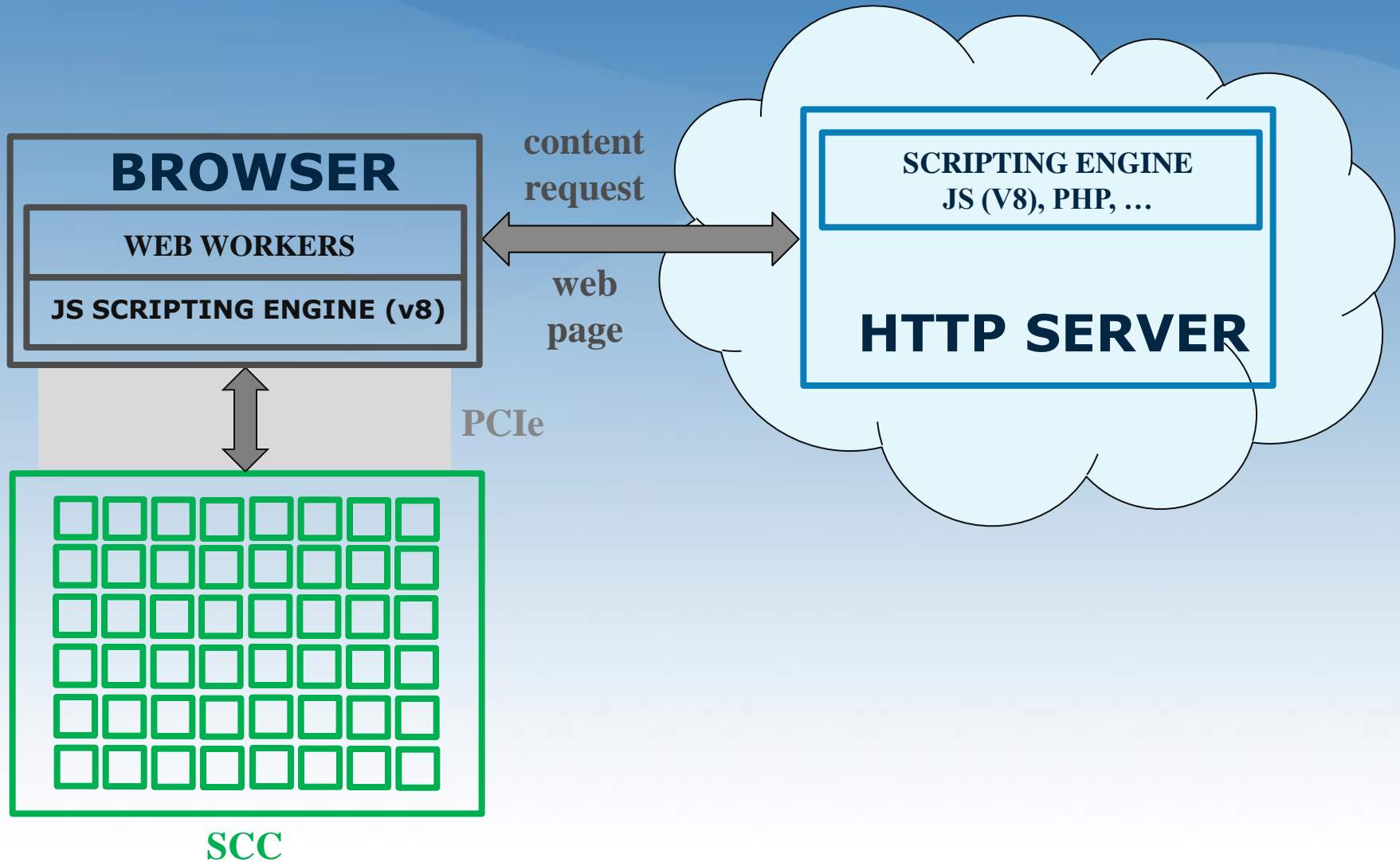
- Offload computation from the client (browser) to the server farm on SCC
- Utilize as many off-the-shelf components as possible for high productivity
 - Client and server code written in pure JavaScript
 - Unmodified client (browser) running on host
 - Largely unmodified execution engine (based on Google's v8) for servers running on SCC
 - Standard libraries and tool-chain used on SCC

Web App Architecture

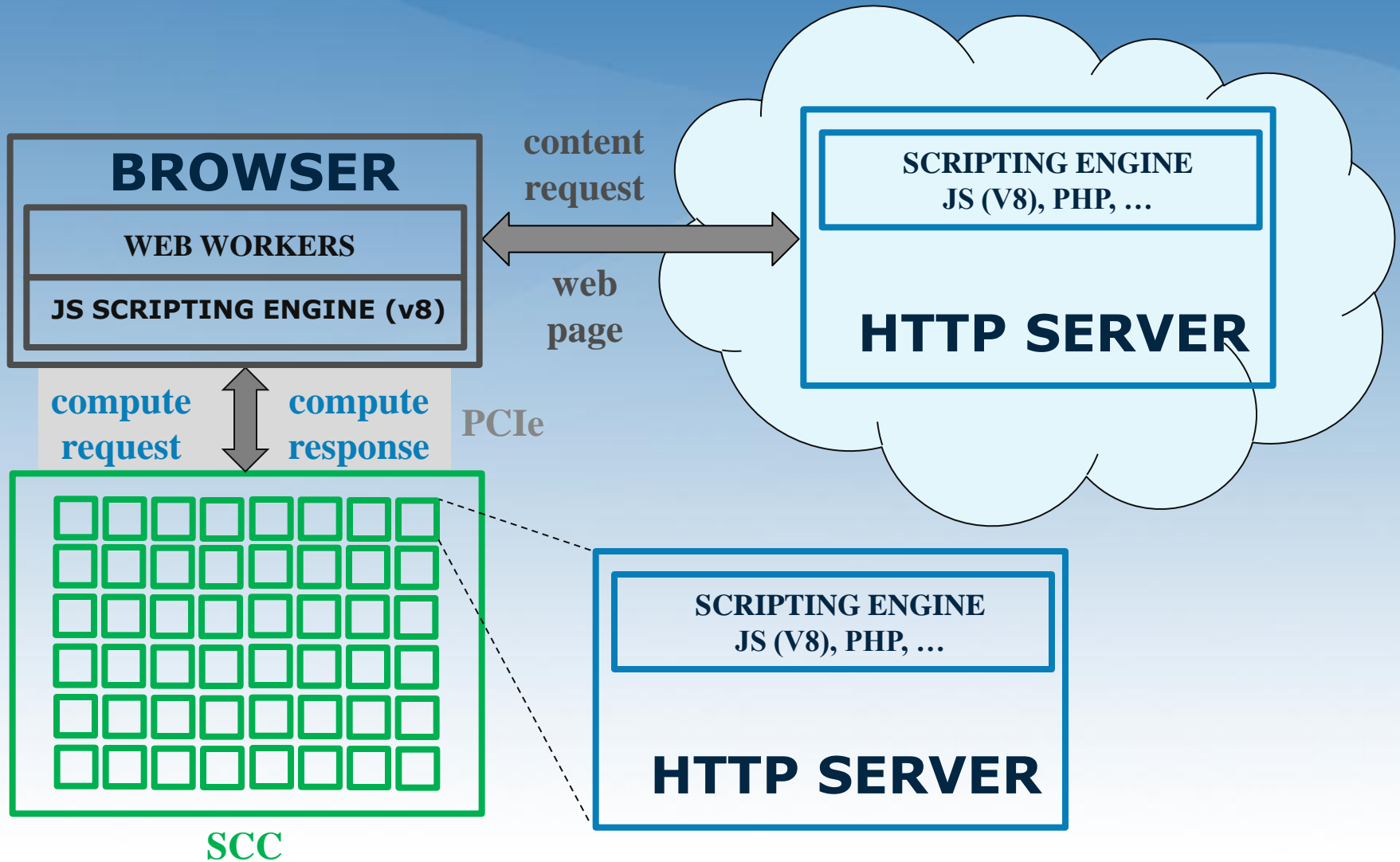


- HTTP server's scripting engine typically used for dynamic web page generation
- Can be used for general-purpose computation as well

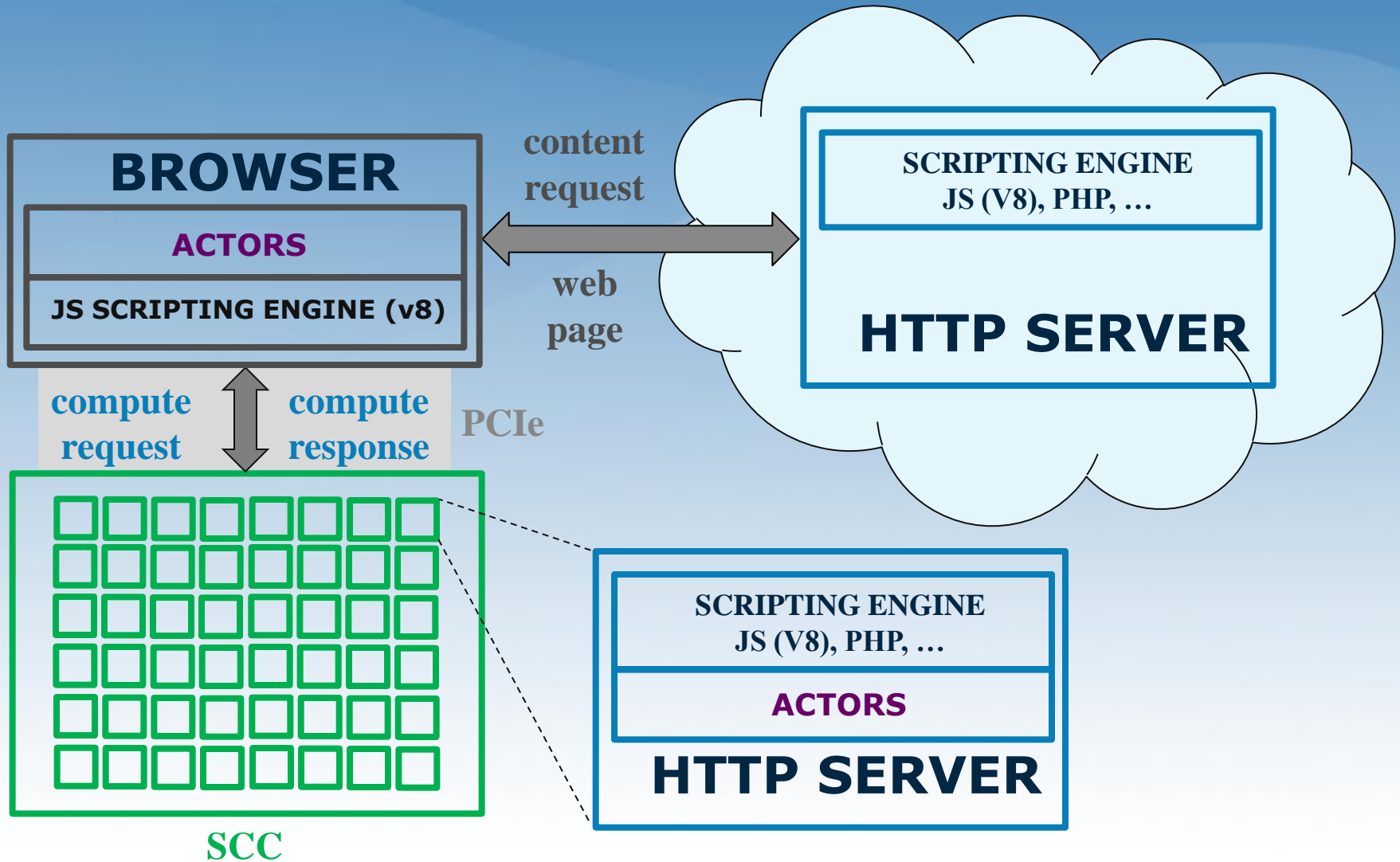
Single-chip Cloud Computer



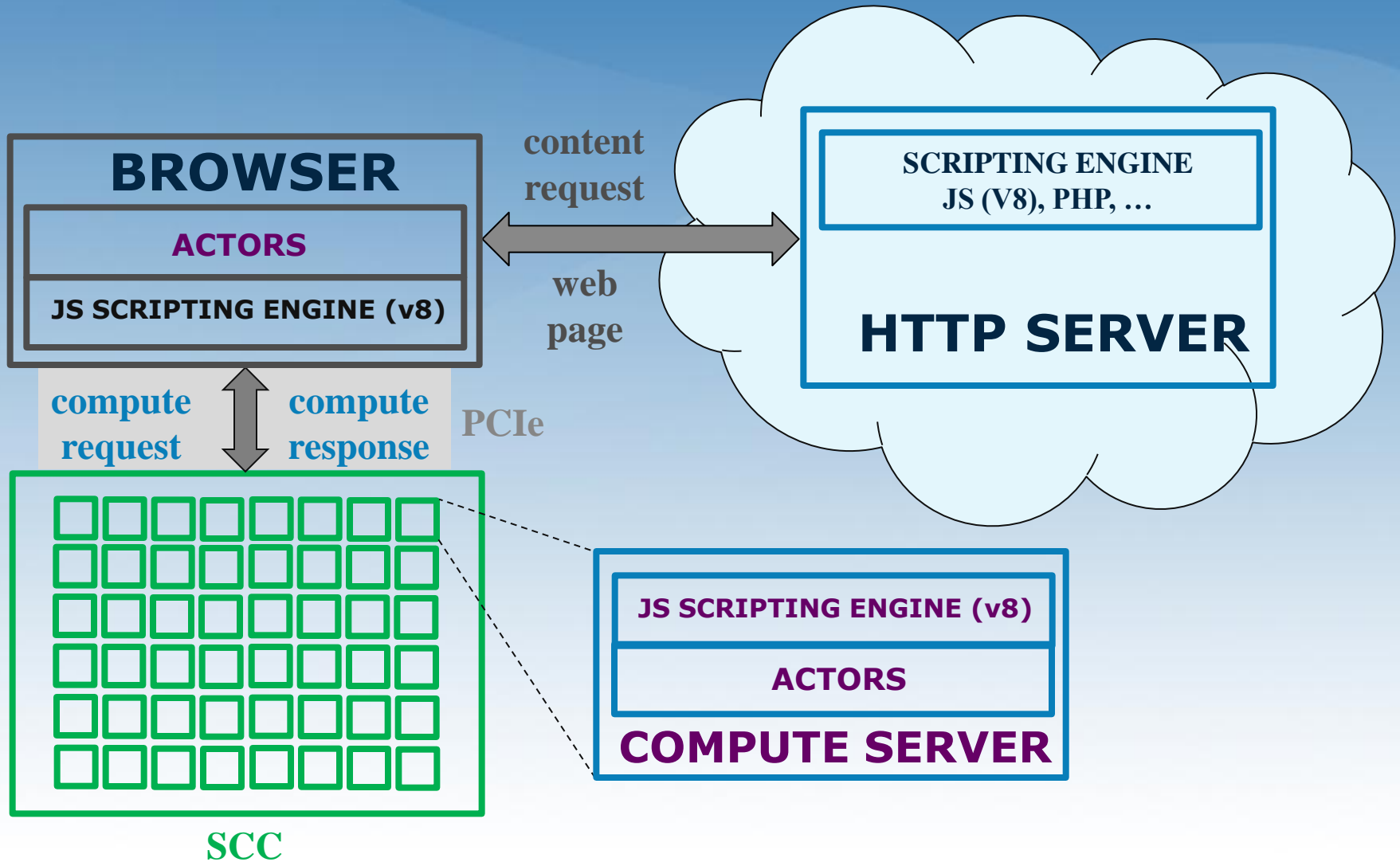
Enabling SCC



Web Workers → Actors



Compute Servers



Infrastructure

- Actors on both sides programmed in JavaScript
- Support for compute servers
 - Google's v8 JavaScript execution engine at the core
 - v8cgi wrapper around v8 for standalone execution environment (minor modifications)
- Unmodified browser can only "talk" HTTP
 - Custom communication layer for compute servers written in JavaScript
 - Problem with "single-origin policy"

Single-origin Policy

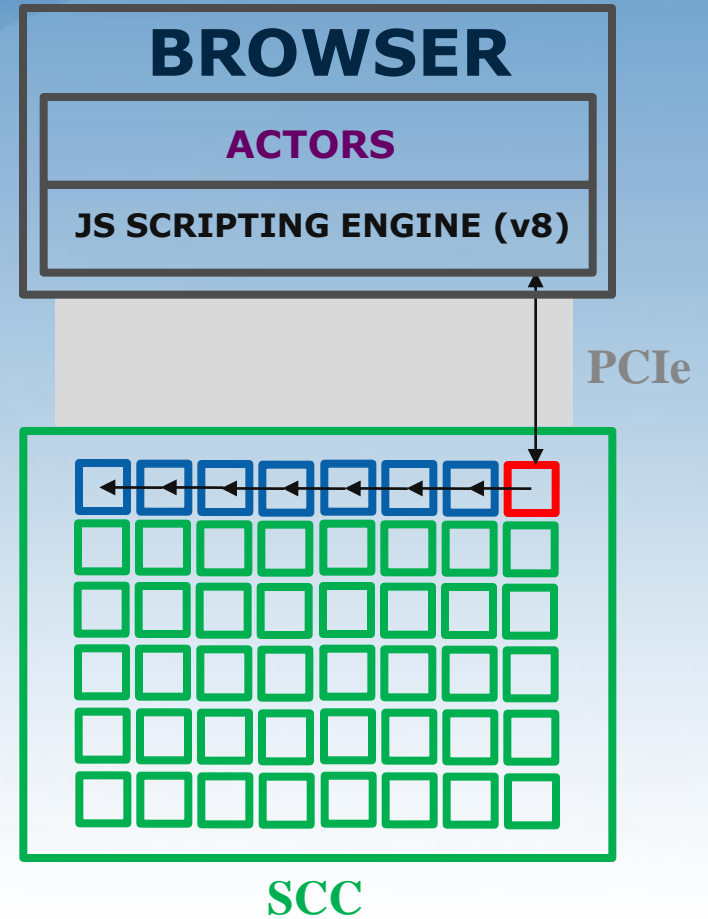
- A script downloaded from a server can only exchange messages with that single server
- Except...
 - A script can run in global scope or in “iframe”
 - Multiple iframes created by the same program can be downloaded from different servers
 - Variables and function definitions can be shared between global scope and iframes
- Solution – generic iframes downloaded from compute servers and used to bootstrap computation defined in global scope

Case Studies

- Parallel ray-tracer (presented here) and parallel physics engine
- Identical infrastructure
- Parallel ray-tracer based on sequential JavaScript app from Google's JavaScript v8 benchmark suite
- Two different approaches for work distribution
 - Dispatcher
 - Direct

Dispatcher Approach

- Reduce host/device network traffic
- Utilize 8 cores (for simplicity):
 - Dispatcher actor on core 0
 - Worker actors on the remaining 7 cores
 - Dispatcher talks to browser and deals work
 - Workers perform actual computation



Browser Front End


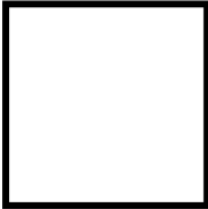
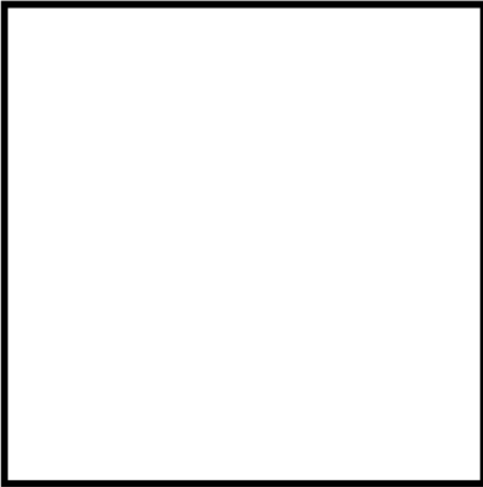
Concurrent Raytrace - Dispatch - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Concurrent Raytrace - Dispatch

Concurrent Raytrace - Dispatch

Parallelized at PSL based on sequential raytrace benchmark from Google JavaScript V8 benchmark suite, v.5

		
2500 pixels	19600 pixels	115600 pixels
<input type="button" value="Run"/> 1 worker	<input type="button" value="Run"/> 8 workers	<input type="button" value="Run"/> 47 workers

Dispatcher Results


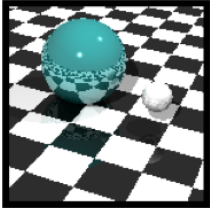
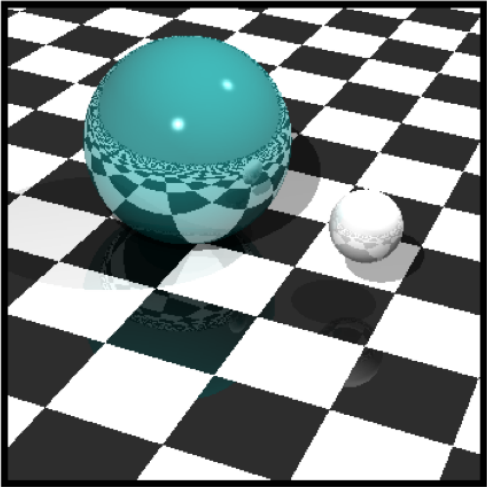
Concurrent Raytrace - Dispatch - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Concurrent Raytrace - Dispatch

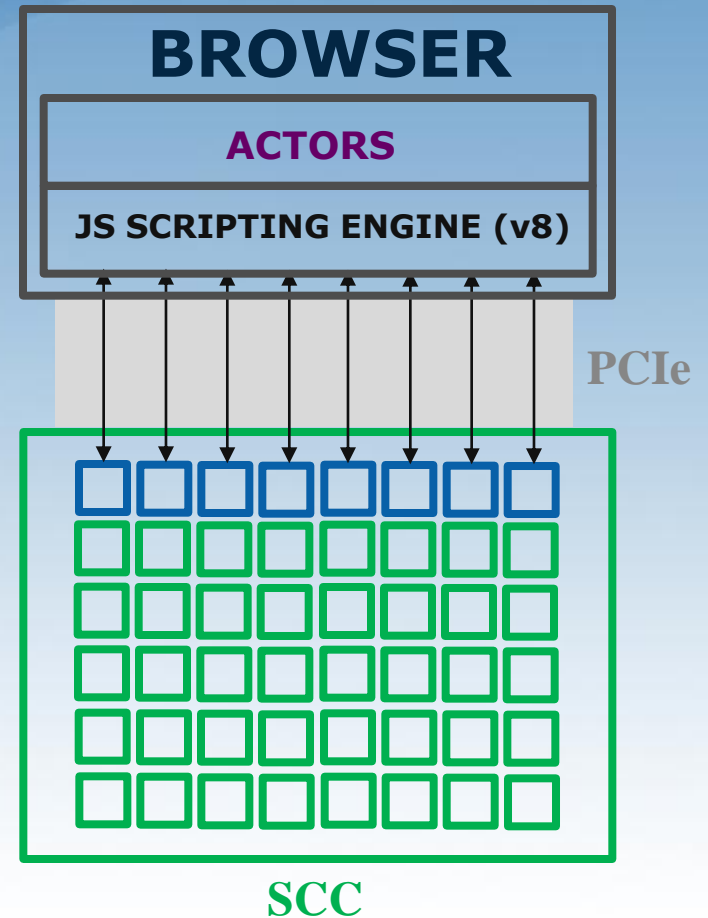
Concurrent Raytrace - Dispatch

Parallelized at PSL based on sequential raytrace benchmark from Google JavaScript V8 benchmark suite, v.5

		
2500 pixels	19600 pixels	115600 pixels
<input type="button" value="Run"/> 1 worker	<input type="button" value="Run"/> 8 workers	<input type="button" value="Run"/> 47 workers
x1	x4.6	x23.2
(Total: 4.24 s)	(Total: 8.68 s)	(Total: 17.78 s)

Direct Approach

- Assume host/device traffic is a non-issue
- Again, utilize 8 cores (for simplicity):
 - Worker actors on all 8 cores
 - Workers talk to browser and perform actual computation



Direct Results


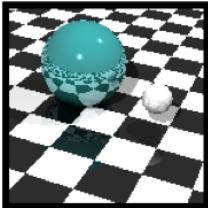
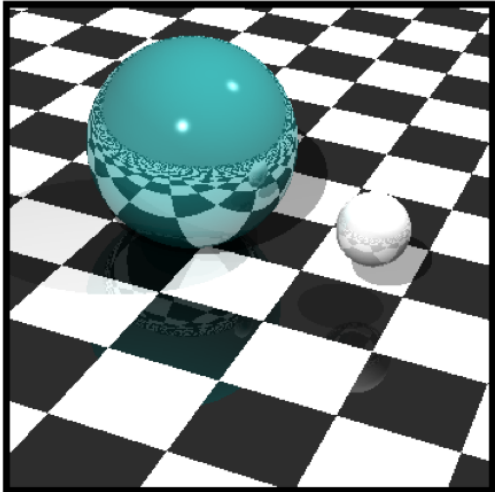
Concurrent Raytrace - Direct - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Concurrent Raytrace - Direct

Concurrent Raytrace - Direct

Parallelized at PSL based on sequential raytrace benchmark from Google JavaScript V8 benchmark suite, v.5

		
2500 pixels	19600 pixels	119025 pixels
<input type="button" value="Run"/> 1 worker	<input type="button" value="Run"/> 8 workers	<input type="button" value="Run"/> 48 workers
x1	x4.6	x21.2
(Total: 3.42 s)	(Total: 5.87 s)	(Total: 7.69 s)

(DISPATCH)

Direct Results


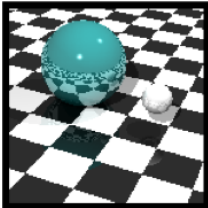
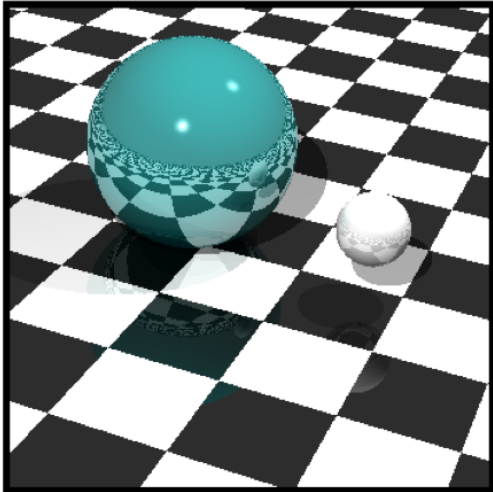
Concurrent Raytrace - Direct - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Concurrent Raytrace - Direct

Concurrent Raytrace - Direct

Parallelized at PSL based on sequential raytrace benchmark from Google JavaScript V8 benchmark suite, v.5

		
2500 pixels	19600 pixels	119025 pixels
<input type="button" value="Run"/> 1 worker	<input type="button" value="Run"/> 8 workers	<input type="button" value="Run"/> 48 workers
x1	x4.6	x21.2
(Total: 3.42 s)	(Total: 5.87 s)	(Total: 7.69 s)
(Total: 4.24 s)	(Total: 8.68 s)	(Total: 17.78 s)

(DISPATCH)

Results Summary

- Actor distribution does make a difference
- Both approaches yield similar scalability for large pictures – $\sim 50\%$
- Host/device network effortlessly handles 48 independent connections
 - Direct approach is $\sim 2x$ faster than dispatcher approach
 - Multiple worker sending rendered image piece-wise help hiding communication latency

Conclusions

- Prototyped a server farm on SCC
- Infrastructure utilizes mostly off-the-shelf software components and tools
- Demonstrated good scalability
- Future work includes
 - Better implementation of the communication layer in a plug-in (no iframes!)
 - Experimentation with other applications that exercise the actor model (e.g. inter-actor communication and collaboration)

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 Break**
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

SCC Co-Travelers Program

Bob Noradki

Microprocessor & Programming Research Lab



SCC Co-Travelers Program

Starter Package for Participants

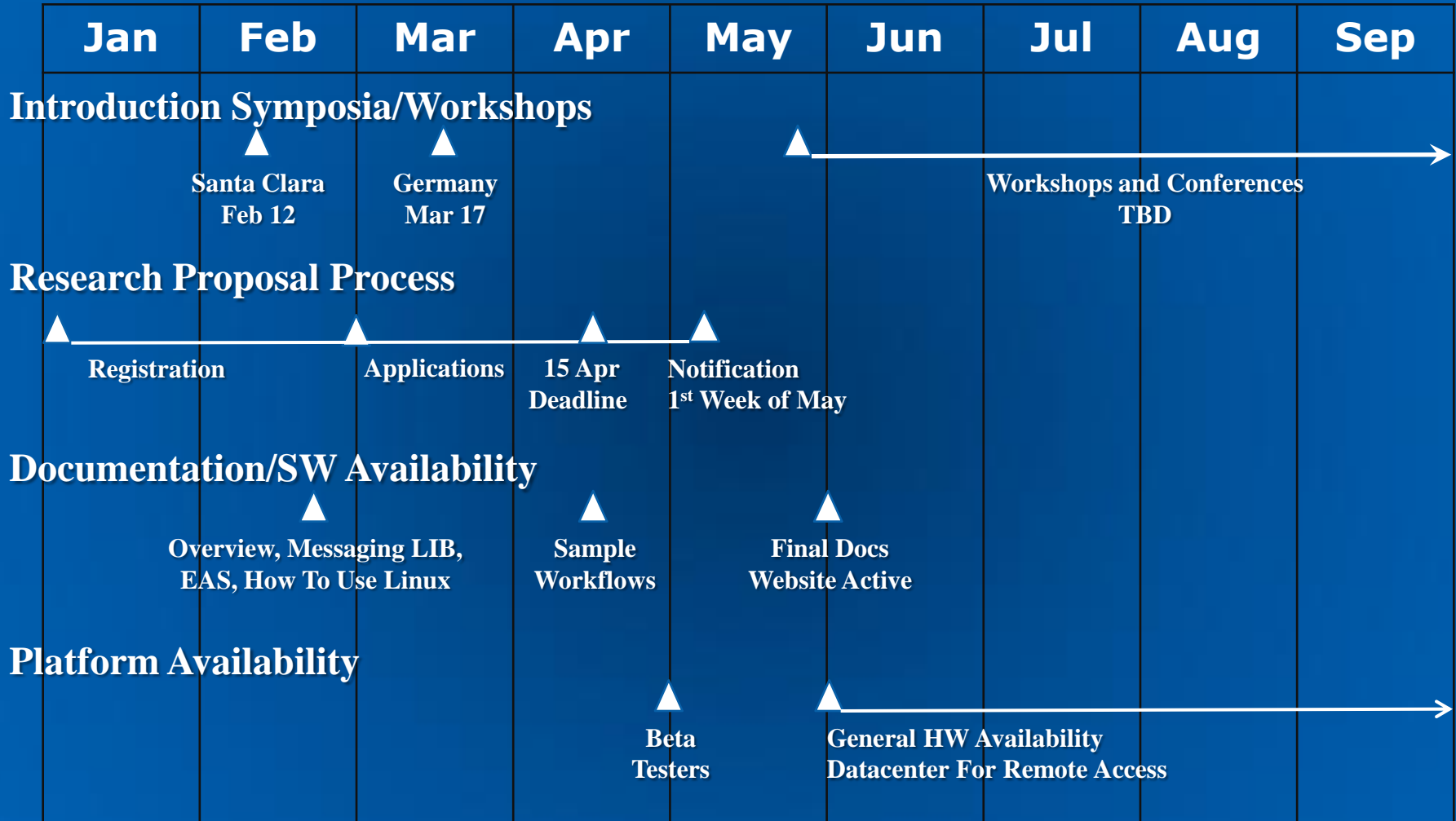
- Access to an SCC System: Remote access or at your site – depending on your research needs
- Documentation: SCC Architecture, How To Guides, RCCE Messaging Library Spec, SCC APIs, more
- Open Source SW: Sample Linux image for SCC, several workflows ported to SCC environments
- Tools: Intel Compilers, Trace Analyzer and Collector. Math Libraries (MKL) and Integrated Performance Primitives (IPP) may be included
- Support: SW: Self-Supporting w/in the community, HW: through Intel Premier Support

SCC Co-Travelers Program

Community Development

- Website to facilitate a strong community
 - User-contributed software will be a key enabler
 - Shared ideas to help overcome research problems
 - Peer assistance on “How To” issues, design research plans
 - Intel Looking for Forum/Topic Sponsors
- Regular Intel Sponsored Conferences
 - Chance to present Data, discuss Results
- Frequent SCC Special Topic Workshops
 - Memory, Power Mgmt, Languages, etc
 - Face to Face and/or Webinars
 - Community driven, not just Intel taking lead

SCC Timeline



Research Grant Application Process

- Key Dates

- Applications Available End of Feb 2010
- Applications Close 15 Apr 2010
- Final Selection Notification 1st Week of May 2010

- Working through existing Research Grant Processes

- Academics: Research Grant Proposal thru Intel ERO
- Industry: Work with Intel Labs and Intel Field Sales
- Government: Use your Existing Intel Labs Contact

- Selection Criteria

- Will it help Intel build better Hardware?
- Will it advance the development of parallel programs for cloud and client?
- Details to be provided on the application forms

Questions?

- General Questions or to request an Application
SCC_Symposia@intel.com
- Technical Questions re: SCC Platform, Research
SCC_Technical_Questions@intel.com

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 *Adjourn*

Agenda

- 10:00 **Welcome and Opening Remarks**
- 10:15 **SCC Hardware Architecture Overview**
- 11:15 **Today's SCC Software Environment**
- 12:15 *Buffet Lunch – Informal discussions*
- 13:15 **Message Passing on the SCC**
- 13:45 **Software-Managed Coherency**
- 14:15 **Application "Deep Dive": Javascript Farm on SCC**
- 14:45 *Break*
- 15:00 **Plans for future SCC access**
- 15:30 **Q&A**
- 16:30 Adjourn**