

# MAJC™

*Microprocessor Architecture for Java Computing*

**Marc Tremblay, Chief Architect**  
**Sun Microsystems Inc.**

*HotChips '99*





# Microprocessor Architecture for Java Computing

*Four-year program to develop a new microprocessor family based on the following four assumptions:*

- 1) Current and future compute tasks are/will be very different than benchmarks used to develop CISC and RISC processors**
  - Algorithms - compute intensive, ratio of compute operations to memory operations is much higher
    - e.g.: IIR filters, VOIP, MPEG2: 3-16 ops/flops per memory op.
  - Data types - digitized analog signals, lots of data
  - I/O - streaming, mostly linearly



# Assumptions (continued)

**2) The dominant platform for networked devices, networked computers, terminals, application servers, etc. will be the Java™ platform.**

- > 1 million Java developers, 72% of Fortune 1000 companies using Java (80% client, 100% server) by 2000, present in Navigator, Explorer, AOL browser, PalmPilots, 2-way pagers, cellular phones, etc.**
- Multi-threaded applications**
  - Observing Java applications with 10's and 100's of threads**
- Run time optimizations is the prime focus**
- Architecture must consider virtual functions, exception model, garbage collection, etc.**

# Assumptions (continued)

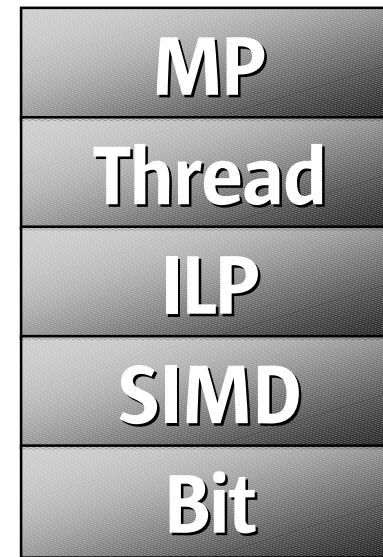
## 3) Parallelism is available at multiple levels

- ILP improves through:

- Data and control speculation
  - non-faulting loads, explicit predication, and traditional methods
- Branch filtering, prefetching, etc.
- Steering loads (indicate where to cache, coherency info, etc.)

- But...

**much more performance available at the thread level**





# Assumptions (continued)

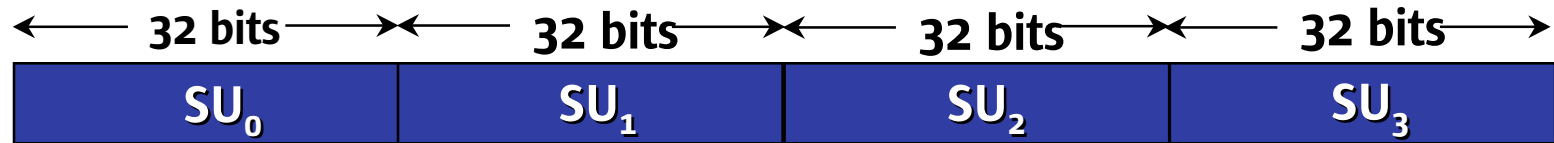
- 4) Companies building a portfolio of IP-blocks.  
Design the architecture for:**
- System-On-a-Chip (SOC)**
  - Multi-Processor System-On-a-Chip (MPSOC)**



# Presentation

- **The ISA**
- **The thread model**
- **Implementation direction**
  - **System-On-a-chip (SOC)**
  - **Multi-Processor System-On-a-Chip (MPSOC)**
- **Modularity and scalability**
- **Conclusion**

# 128-bit VLIW Instruction Packets



↓ header

00 = 1 SU (Super Unit)

01 = 2 SUs

10 = 3 SUs

11 = 4 SUs



- opcodes available for 4-register specifier instructions
- use “rd” for more opcode bits for other instructions



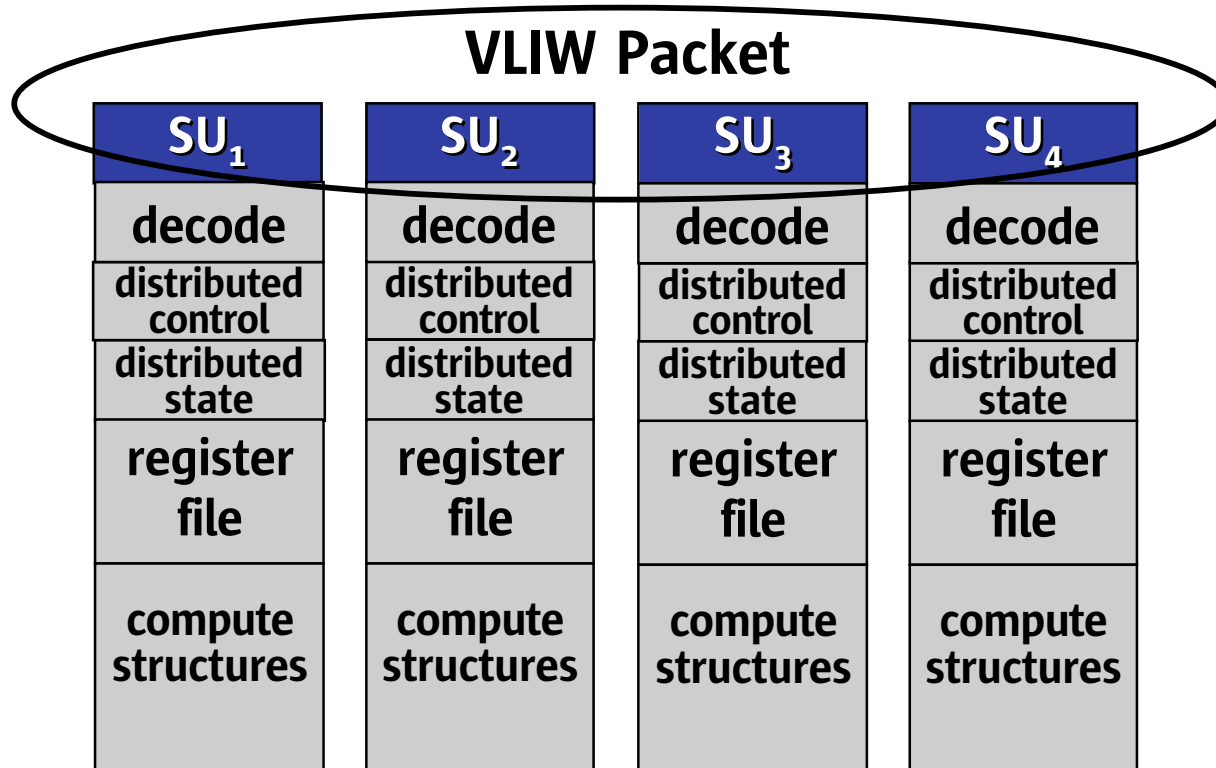
# Important Aspects of the ISA

- **Maximum of four sub-instructions per cycle per CPU**
  - Diminishing ILP improvement for general purpose code and growing impact on cycle time beyond four-issue
  - Exploit parallelism at a higher level
- **No interlock mandated at the architecture level**
  - Applies for both intra- and inter-group dependences
- **Instructions with non-deterministic latencies (e.g. cacheable loads) can be scoreboardd**
- **Opcode space for slot 2, 3, and 4 is identical**
  - No steering needed
  - Slot 1 is mapped onto a quadrant of the same space



# Instruction-Slice Architecture

- **Maximize functionality but minimize the number of functional units**
  - Implementation typically has **1** functional unit for first slot and another one (replicated twice) for the **3** other slots
- **Design a slice with its own register file, local control and state and local wiring**
- **Step-and-repeat design process**
- **Facilitates compiler scheduling, instruction routing (no large template), and reduces wiring drastically**
- **Leads to unified register file and data-type agnostic functional units (high efficiency of compute structures)**



- Implementation can be 1-, 2-, 3-, or 4-issue
- All from mostly the same full custom design

# Large Unified Register File



- Integer, fixed-point, floating-point data co-exist
- Locals: accessible only from the associated functional unit (slice)
- Globals: shared by all slices
- Delimiter: partitions locals and globals
  - e.g.: 64 globals + (4 \* 64 locals) = 320 registers
  - e.g.: 96 globals + (4 \* 32 locals) = 224 registers

# Digital Data Types

- **16-bit**
  - Short integers  
(e.g. pixels, speech data, general purpose)
  - Fixed S.15, S2.13 (e.g. speech data, graphics)
  - With/without saturation, various saturation modes
- **32-bit**
  - Integers, some with saturation
  - Floats  
(e.g. digital communication, graphics, audio data)
- **64-bit**
  - Long integers, double floats (general purpose)
- **Limited 8-bit Support**  
(for auditory, visual distance measures)



# DSP and Floating-Point

- True multiply-add, dot-add, dot-sub, etc.
- bit-extract, byte-shuffle, leading-one, etc.
- Interval arithmetic
  - directed rounding
  - min and max
  - pick\_conditional, move\_conditional
- Proper data types and saturation modes
- All of the above available and controllable for each functional unit
  - e.g. 3 multiply-add can be done per cycle per CPU



# Thread Model

- **Architecture supports:**
  - **Multiple independent processes per chip**
    - Servers, multiple JVMs
  - **Independent threads in the same address space**
    - Multi-threaded applications, libraries, system software, GC, JIT
  - **Threads sharing data**
    - Focus is on very tightly-coupled CPUs
  - **Space-Time Computing (STC) for the JavaTM platform**
  - **Vertical multithreading**



# Key Java Properties for Thread-Level Parallelism (TLP)

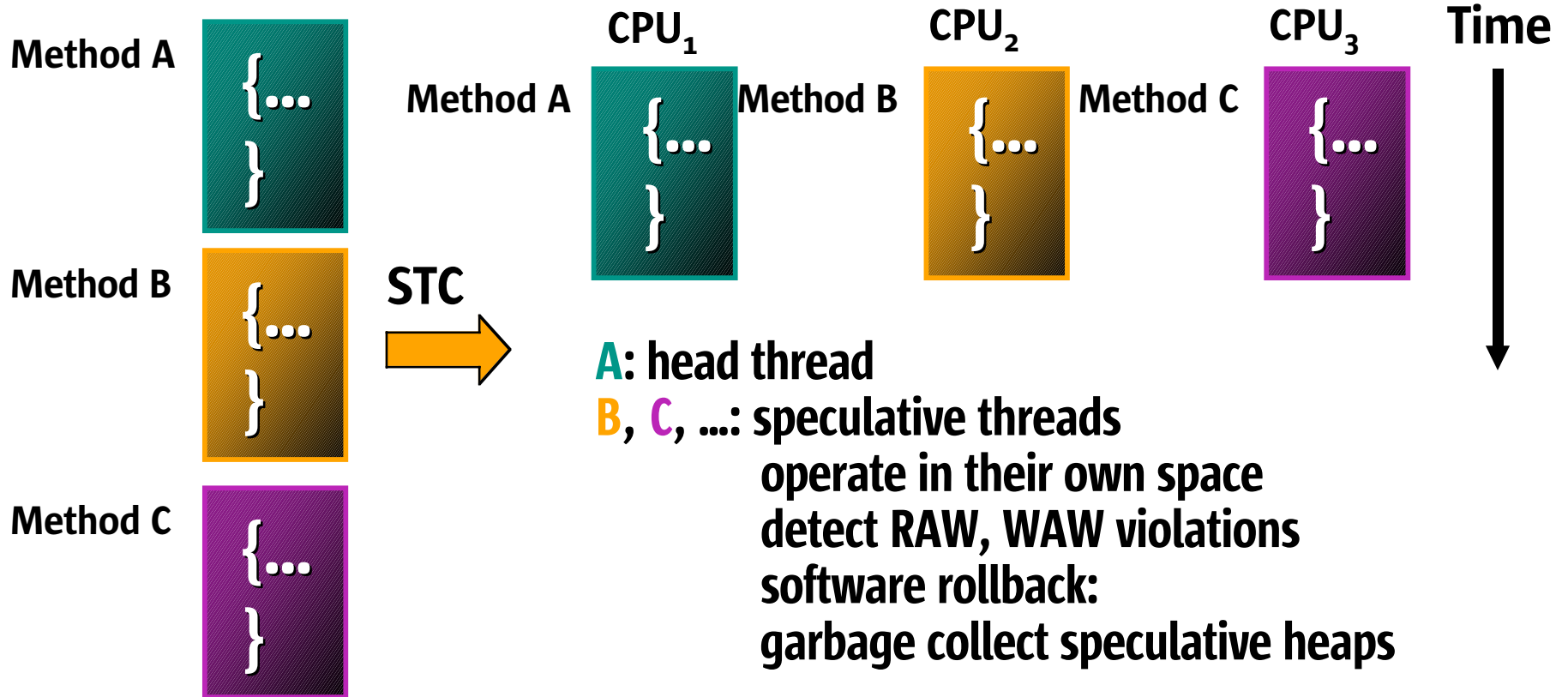
- Object-oriented nature of the language
- Gosling property of bytecodes
- Pass by value semantic of Java for primitive types
- Method communication limited to the heap and return values
- ➔ Stack values after execution of a void method is known at the start of the next method
- Specific Java bytecodes access objects
- ➔ Software filtering of instructions to watch

# Space-Time Computing (STC)

- **Speculative threads operate in:**
  - a different space - the speculative heap
  - a different time - future time, in which objects are still unborn with respect to the head thread
- **Speculative threads interact often given that they share common objects that do not need to be replicated**
- **Versioning of objects is done in a separate space which is cached locally**




# Space-Time Computing in MAJC



## *Joins:* collapsing of dimensions

- heap merging
- new head thread

# MAJC and STC

- **MPSOC allows fast object sharing**
- **Very tightly-coupled architecture**
  - Very fast coherency bandwidth
  - Fast thread synchronization
-  **A few cycles (vs. 100 for traditional MPs)**
- **Fast thread control mechanism**
  - Independent of memory access instructions
- **Special instructions for STC**
- **Relaxed Memory Ordering (only!)**
- **Results given during presentation**

# Vertical Multi-Threading

- Throughput computing (vs. only latency computing)
- Cache misses typically cost 75-100 cycles (to DRAM)
- Example: 5% miss, for 100 instructions
  - hits run in 1 cycle: 95 cycles
  - misses run in 75 cycles:  $5 * 75 = 375$  cycles
  - utilization around 20%



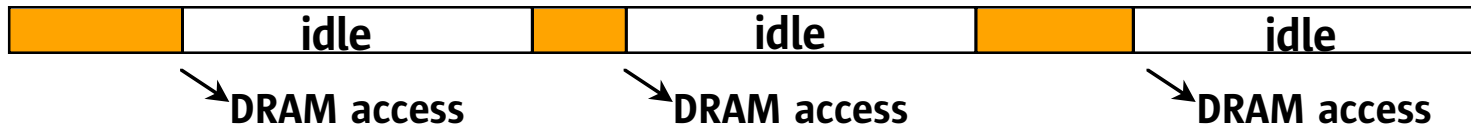
 = processor computing

 = processor idle

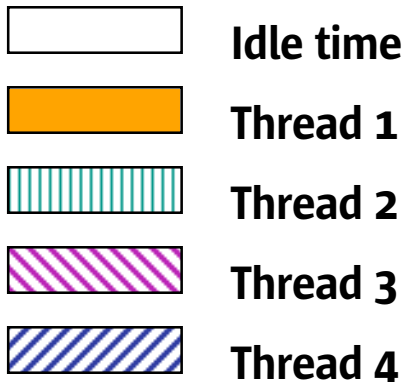
# Vertical Multithreading

Time  


**Current processors:**



**MAJC implementations:**



- Switch thread as soon as a cache miss occurs
- Maintain state for 4 running threads
- Thread priority allows for running one thread without penalty



# MAJC Support for Vertical Multi-threading

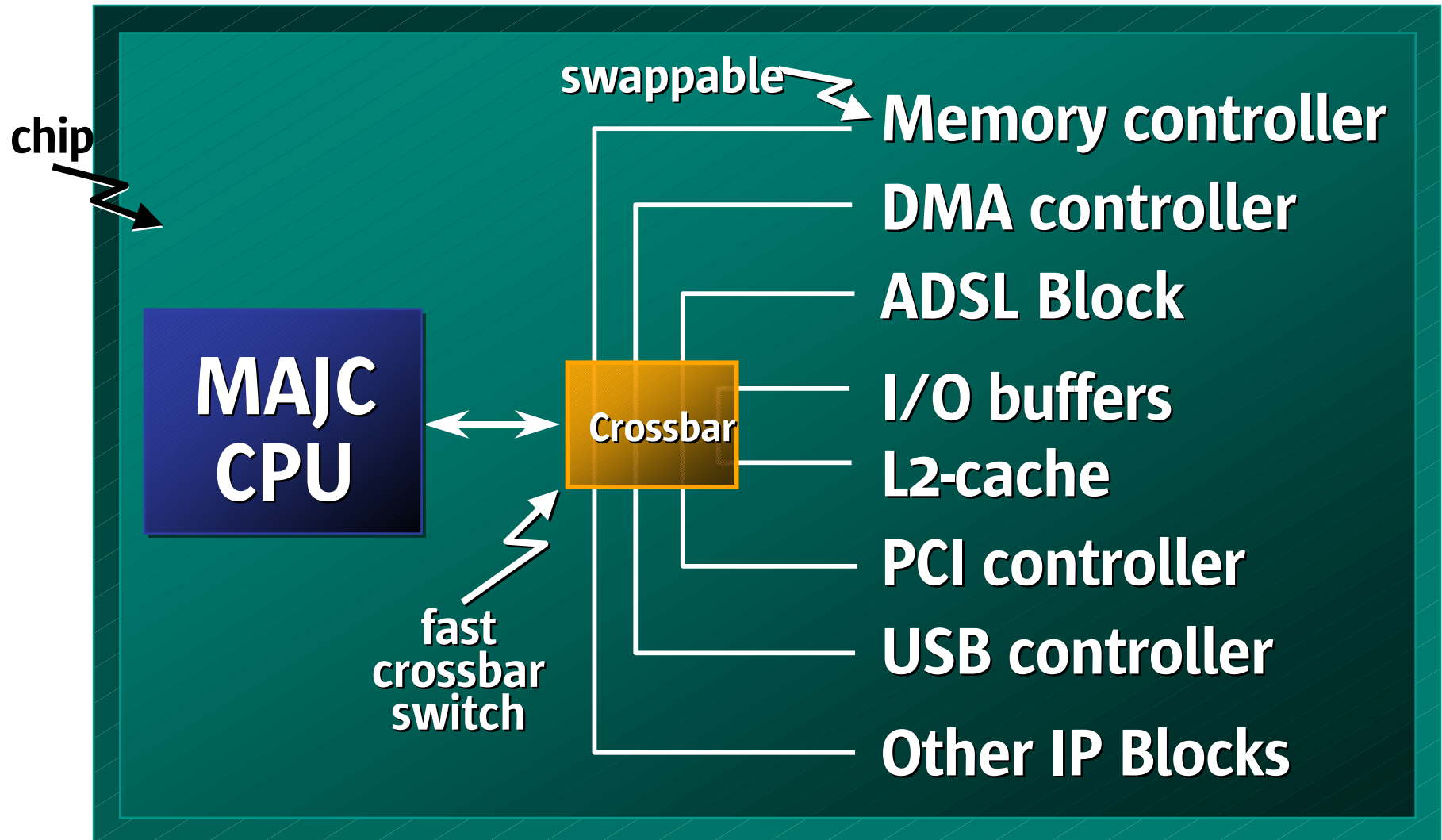
- **Special instructions**
- **Large unified register file**
  - Can be partitioned into groups of registers
  - Each group can be “enabled” for each thread
  - Reads and writes from/to groups are monitored
- **Vertical state**
  - Few centralized control registers to keep track of



# Implementation Direction: System-On-a-Chip (SOC)

- **Architecture allows the integration of IP blocks:**
  - e.g. I/O controllers (PCI, 1394, USB, etc.)
  - Memory controllers (DRDRAMs, SDRAMs)
  - Compute blocks (Graphics pre- and post-processors, mixed-logic blocks for QAM, QPSK, etc.)
  - Multiple levels of caches
- **How?**
  - Fast crossbar switch, running at processor speed, connecting CPU's with blocks
  - Handshake protocol allows interface flexibility
  - Asynchronous interfaces for different clock speed I/O's

# Pluggable IP Modules





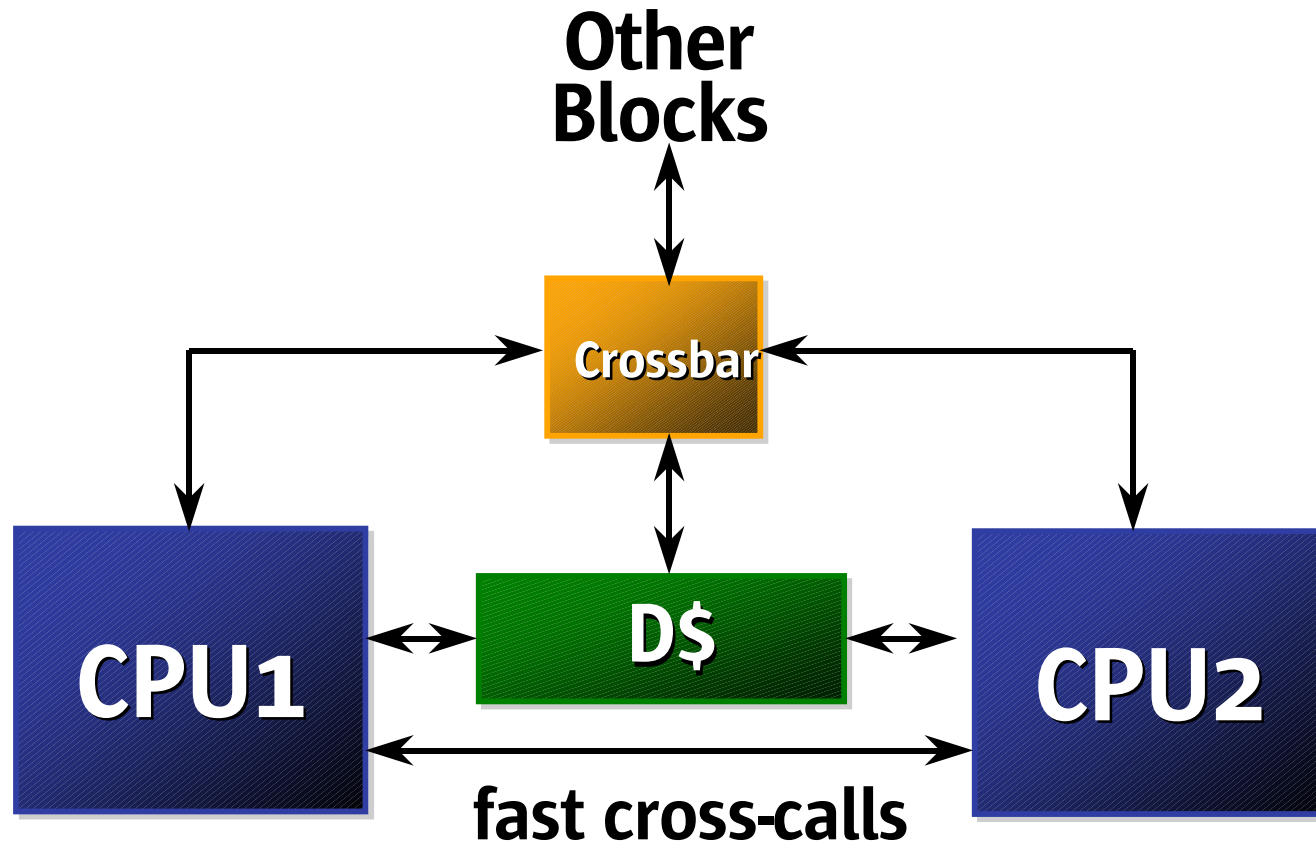
# Multi-Processor System-On-a-Chip (MPSOC)

- **Very tightly-coupled CPUs residing on same die**
  - CPUs share first-level cache or
  - CPUs have a fast coherency protocol (a few cycles)

➔ enables new levels of performance
- **Limit complexity of each CPU (e.g. issue-width)**
- **Limit hardware coherency domain to one die**
- **Scaleable communication protocol between CPUs**



# MPSOC - Implementation Example





# Modularity and Scalability

- **Modularity**
  - Instruction-slice architecture
  - Step-and-repeat full custom design
  - Handshake protocols
  - Asynchronous interfaces
- **Scalability**
  - Variable issue-width: 1, 2, 3, 4
  - Variable register file size: 32, 64, ..., 512
  - Datapath and address space: 32, 64
  - Number of processing units per die: 1, 2, ..., 1024



# Conclusion

- **Not having binary compatibility (legacy) requirements opens the door for innovation**
- **A new architecture should:**
  - **Address parallelism at multiple levels**
  - **Allow fast clock implementations without thousand-people-year efforts**
  - **Reach millions of programmers through the Java™ platform and API's**
- **An architecture is only as good as its implementations, stay tuned for chip disclosures**



*Sun*<sup>®</sup>

microsystems

**We're the dot in .com<sup>™</sup>**