Cray X1 Architecture Overview and Optimization CUG Workshop May, 2004

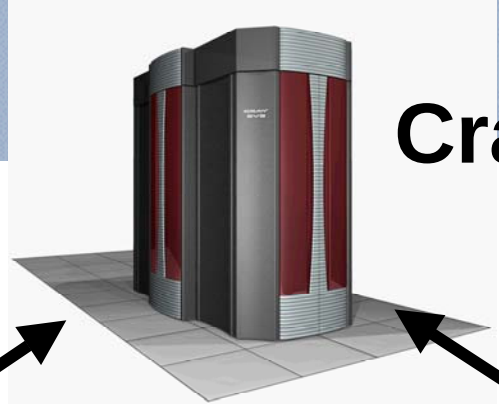
James L. Schwarzmeier
Cray Inc.
jads@cray.com
715-726-4756

Cray X1 Performance after One Year

- many improvements in PE, OS, optimization knowledge base
- X1 vector processor:
  - typical % peak (12.8 GF) on decent vector code 15-40%
  - typical speedup over (5.2 GF) Power 4 ~ 7-30x
- X1 network:
  - plenty of bandwidth (except GUPS)
  - MPI latency improving
  - CAF/UPC exceptional for collective operations
- customers doing problems on X1 they could never do before (some discussed at CUG)
  - vectorization can be work, but a *demonstrable path to HPC*
- X1 often limited more by CPU and memory *latencies* than memory *bandwidth* ➔ good for Cray X1E (lower bandwidth and lower latency than X1)

CUG 2004

Cray X1



Cray PVP

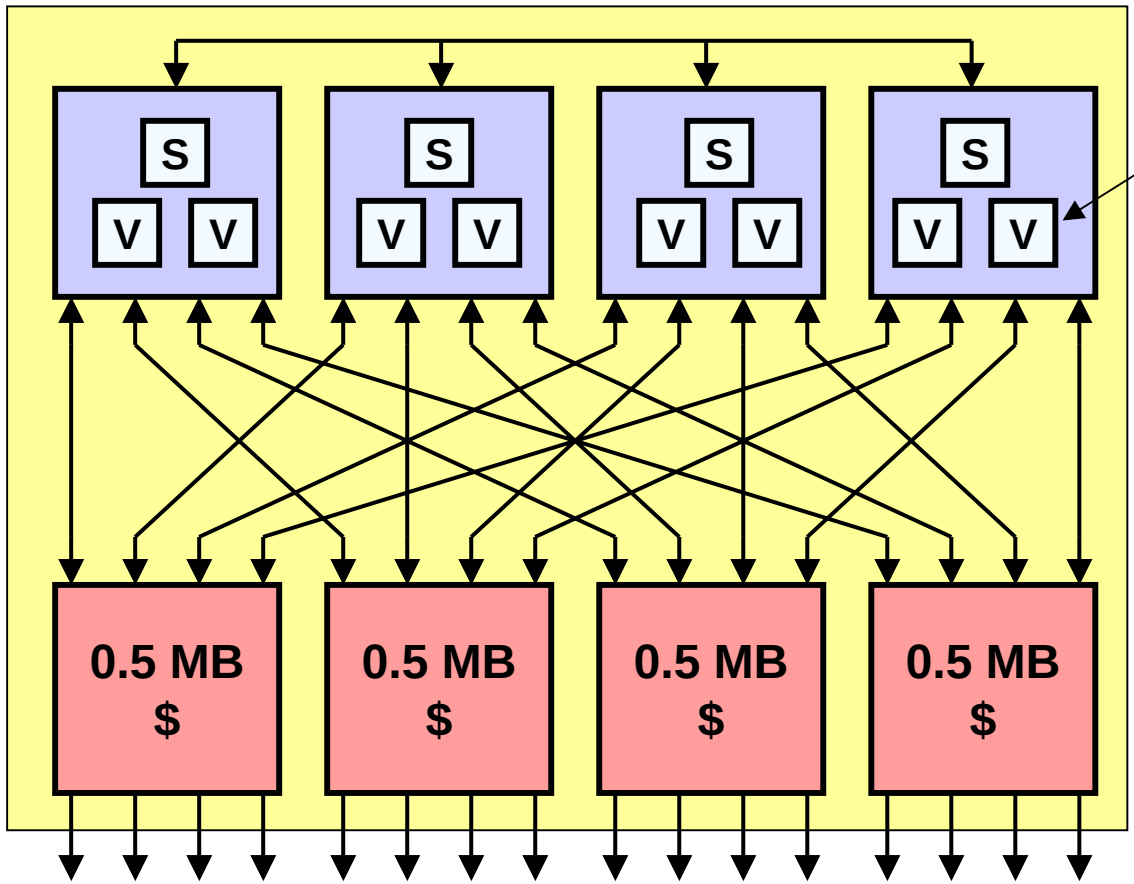
- Powerful single processors
- Very high memory bandwidth
- Non-unit stride computation
- Special ISA features
- Modernized the ISA

Cray T3E

- Distributed shared memory
- High bw scalable network
- Optimized communication and synchronization features
- Improved via custom processors

Extreme scalability with high bandwidth custom processors

Cray X1 Multi-Streaming Processor



12.8 Gflops (64 bit)

25.6 Gflops (32 bit)

51 GB/s ↑

25-41 GB/s ↓

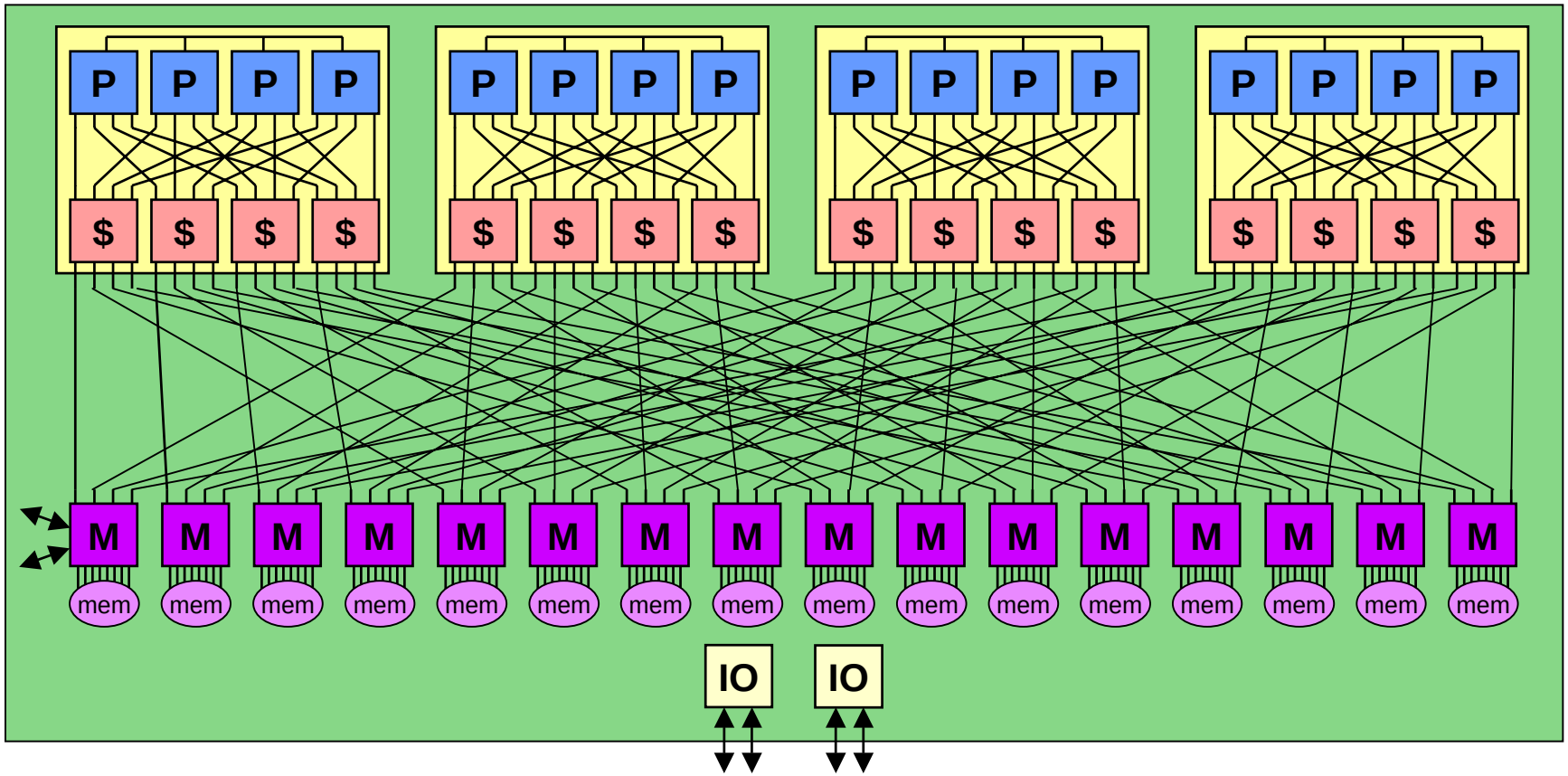
2 MB Ecache

At frequency of 400/800 MHz

To local memory and network: 25.6 GB/s ↑
12.8 - 20.5 GB/s ↓

CUG 2004

Cray X1 Node (a T932 on a board)



Inter node network:

- 2 ports per M chip
- 1.6 GB/s full duplex per link

I/O connections:

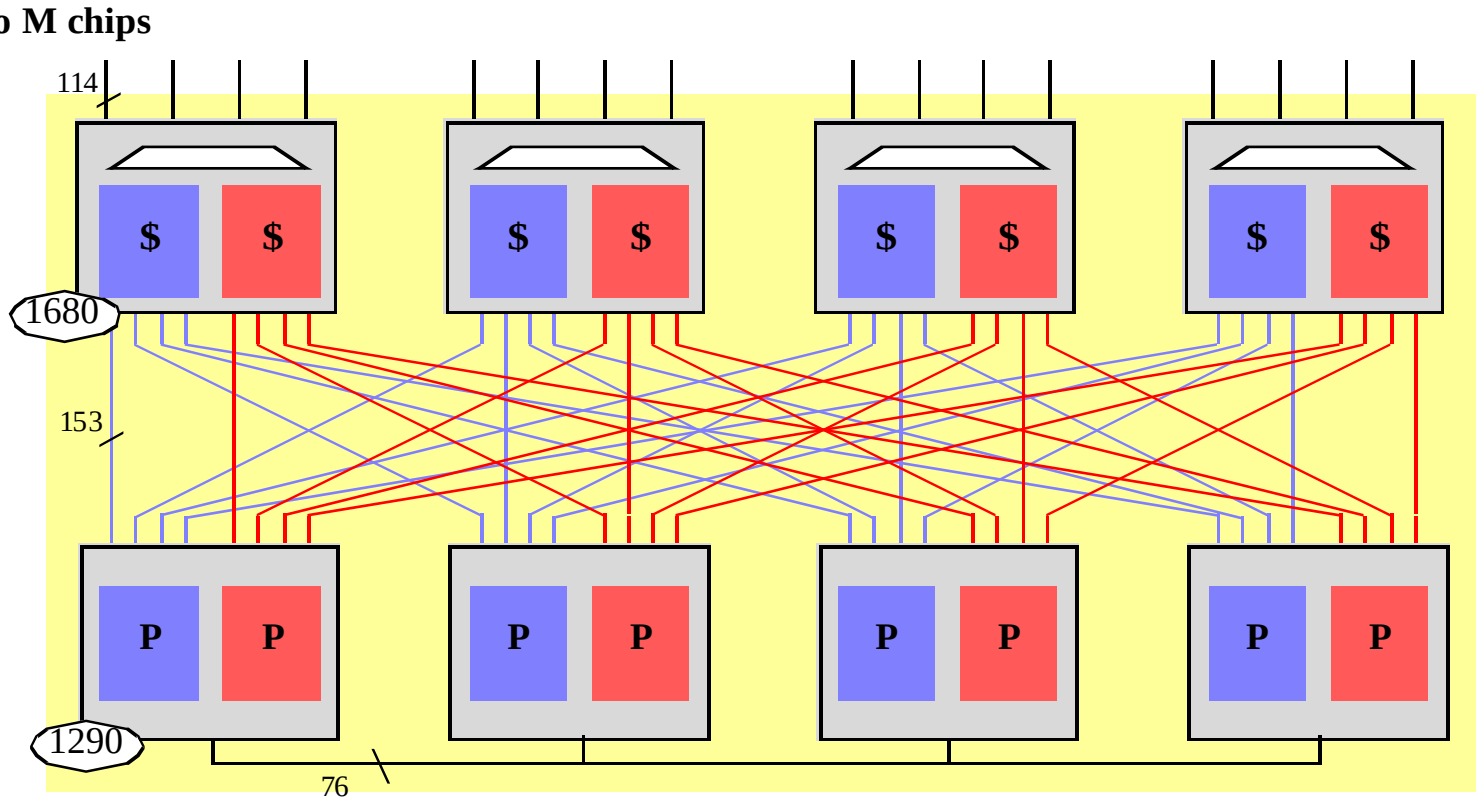
- 4 ports per node
- 1.2 GB/s full duplex per link

Local memory:

- 200 GB/s peak bw
- 8-32 GB per node

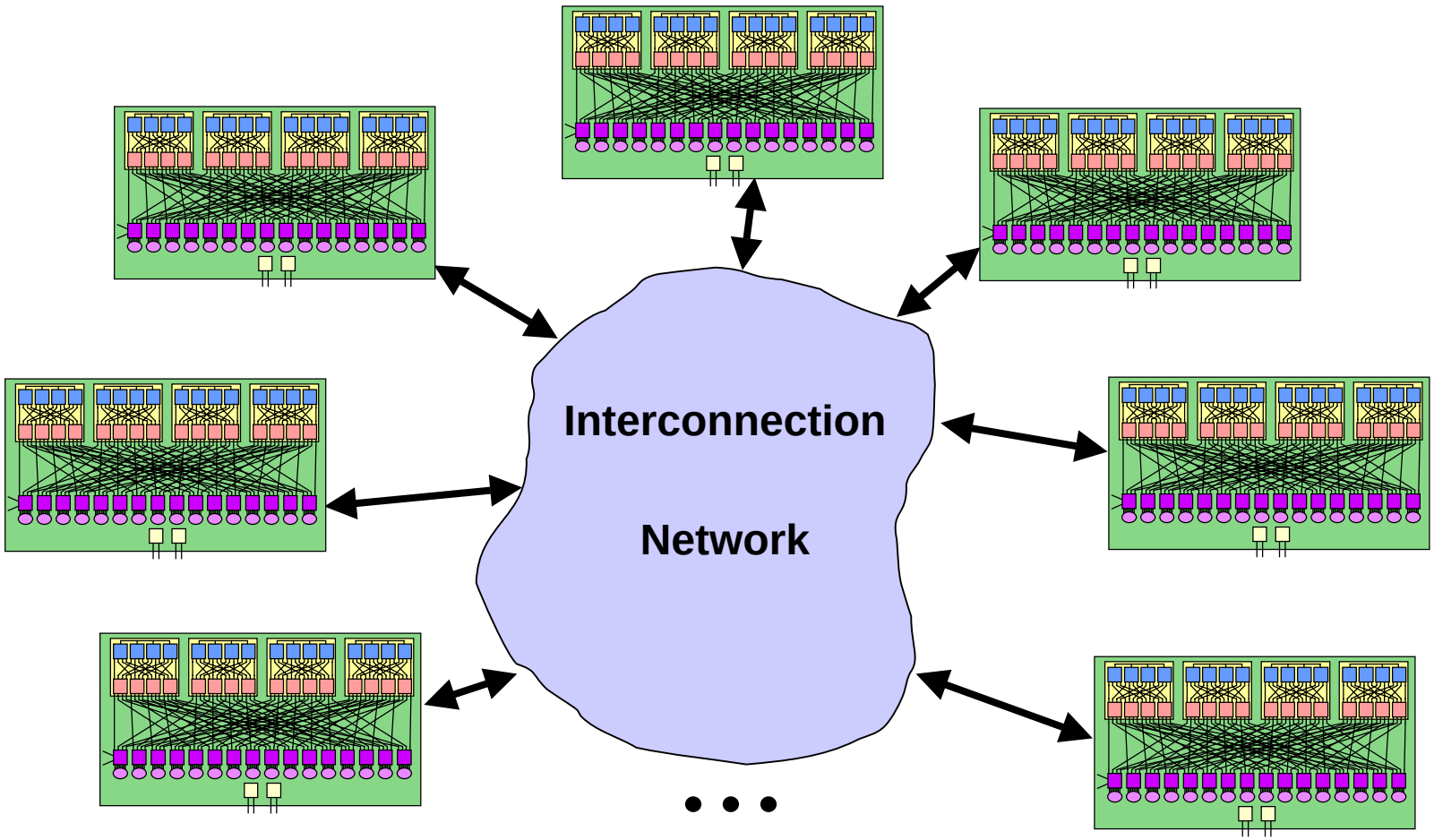
CUG 2004

X1E MCM



- **Re-implement P and E chips in 0.08mm IC technology**
- **Place two MSPs on each MCM**
  - ⇒ **Double the processor density (8 MSPs/module)**
- **Significant frequency increase (~50%)**

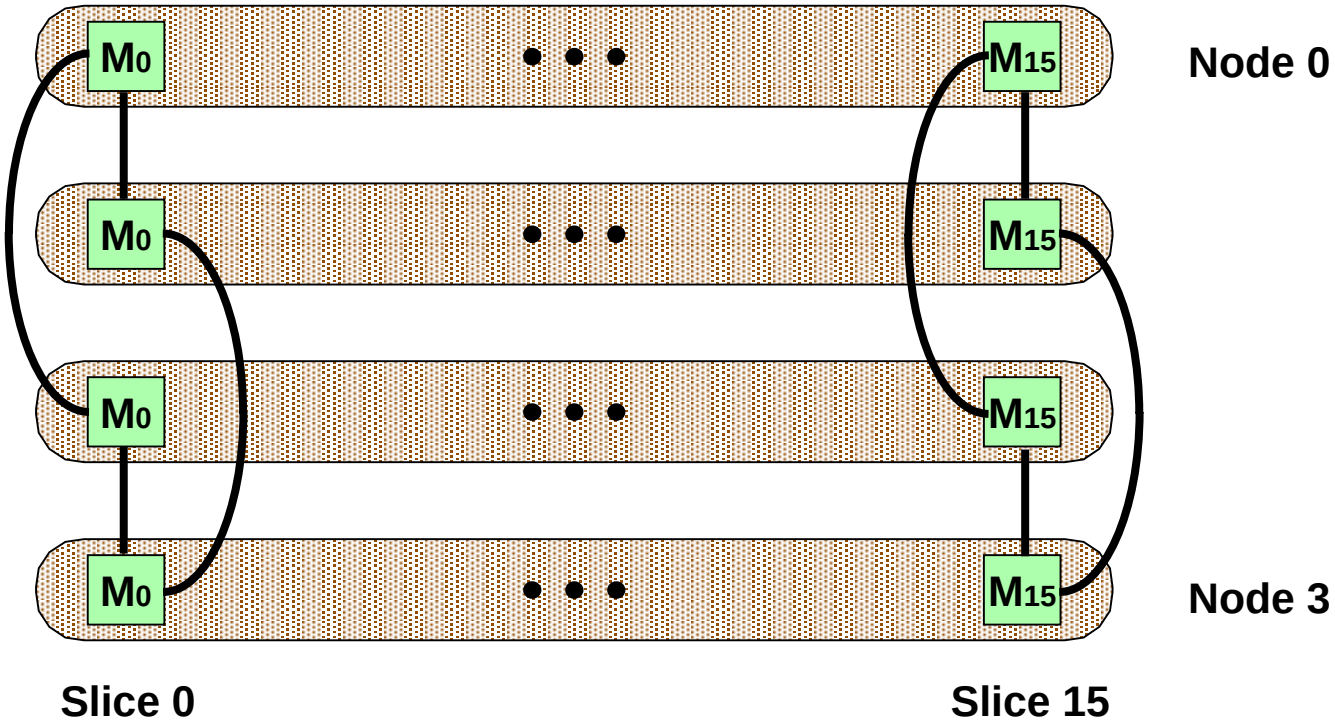
NUMA Scalable up to 1024 Nodes



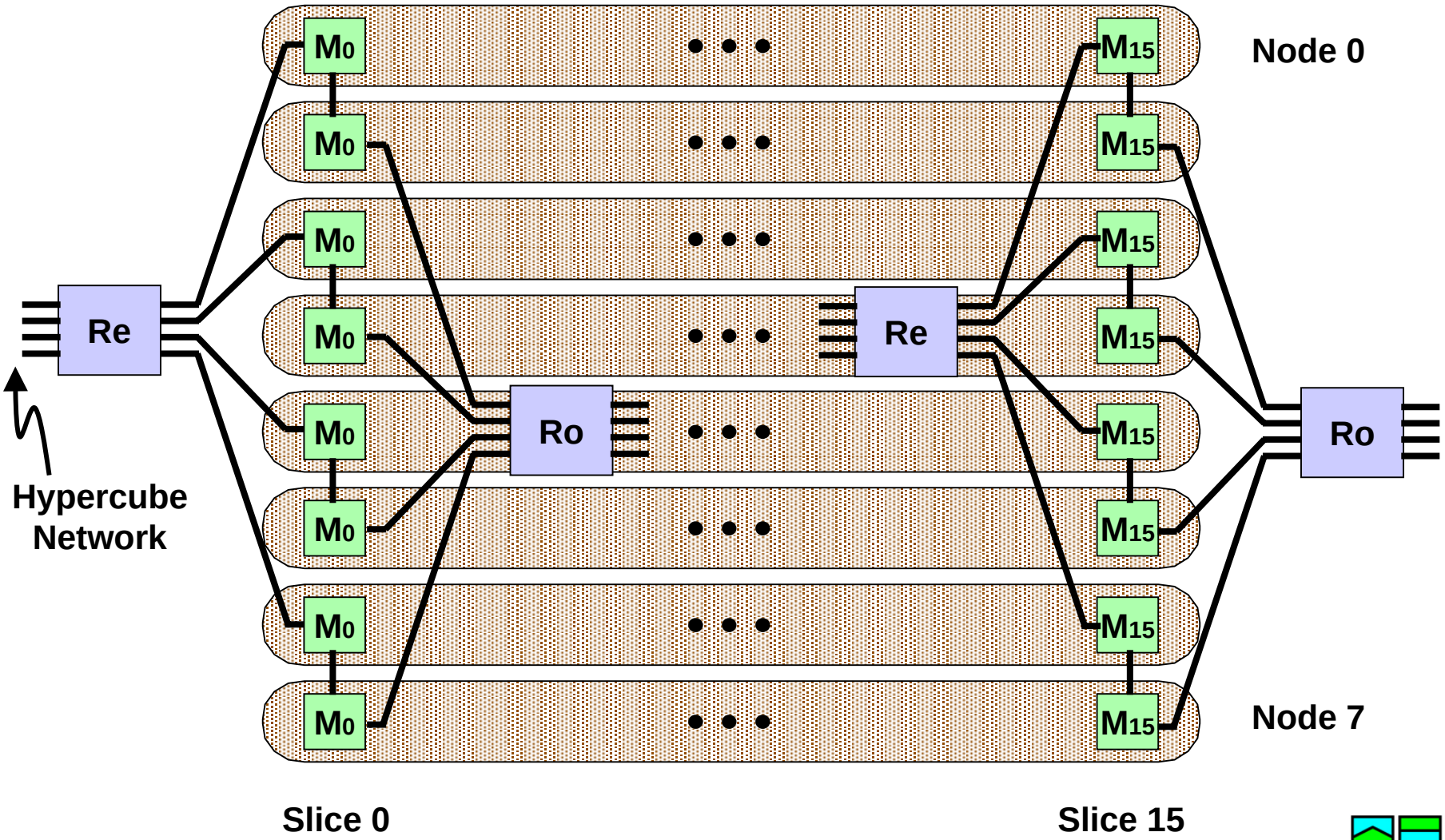
- 32 parallel networks for bandwidth
- Quad-bristled hypercubes to 512 CPUs

Network for Small Systems

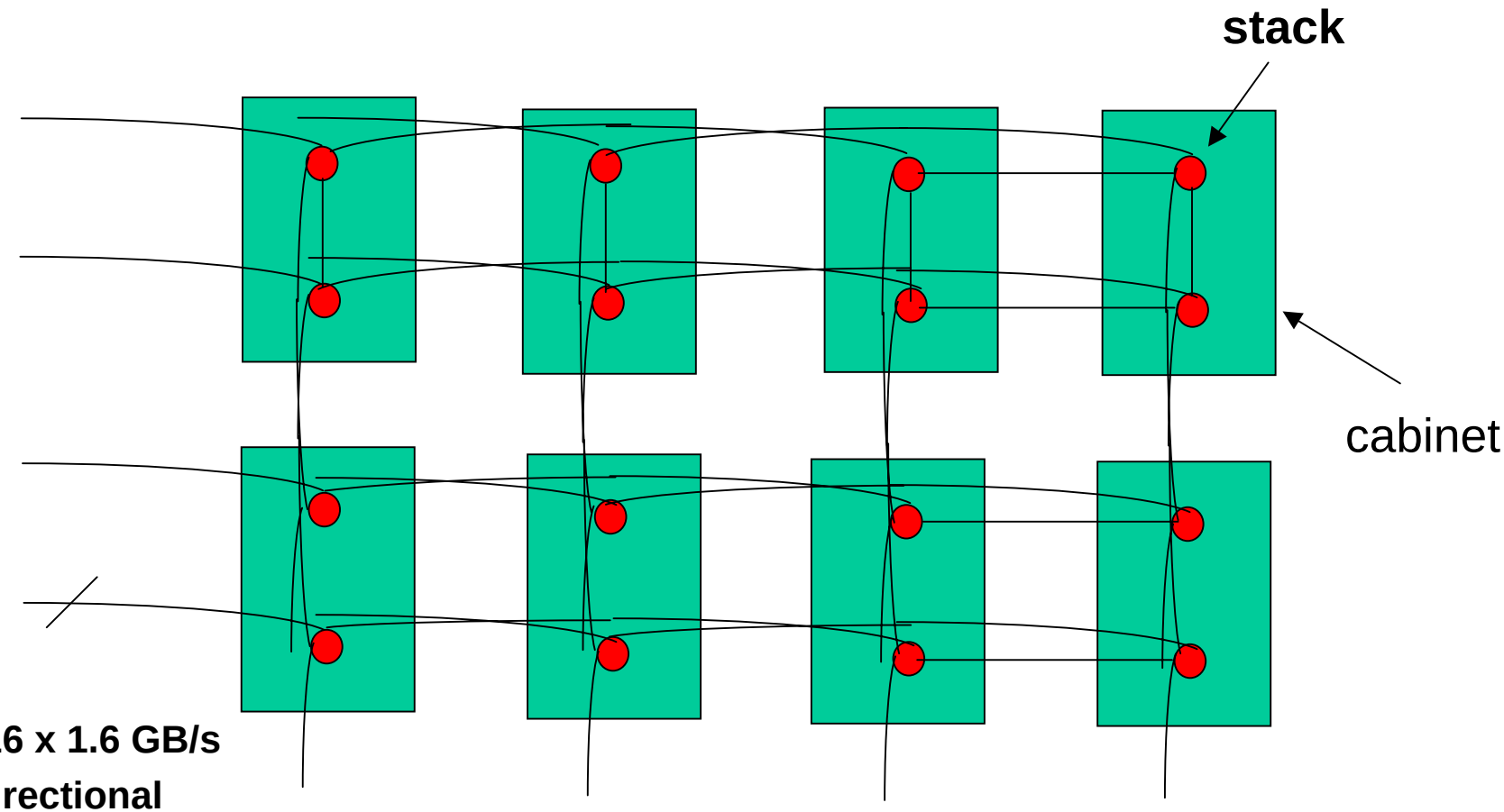
- **Up to 16 CPUs, can connect directly via M chip ports**



‘Stack’: Scalable Network Building Block



Configuring a Large X1 System



2 x 16 x 1.6 GB/s
Bi-directional

Programmer's View

- X1 as a collection of *shared memory* SMP nodes, each running OpenMP or single processor vector jobs, with Single System Image (SSI) over whole machine
  - **MSP-mode** automatically by compiler, *or* **SSP-mode**
  - OpenMP, pthreads (using either 4 MSPs or 16 SSPs)
  - 51 GFLOPS nodes, high UMA memory bandwidth, 16-32 GB/node

OR

- X1 as a large MPP with vector processors running *distributed memory* jobs, with high bandwidth interconnect between processors
  - MPI, shmem(), UPC, Co-Array Fortran (CAF)
  - same kinds of optimizations as on microprocessor-based machines
    - work and data decomposition
    - cache blocking (higher BW in cache, MSP improves short VL)
  - but *less* concern about communication/computation ratio, memory stride and bandwidth

Update of the Cray ISA

- Many more registers: 32 vector, 128 scalar (64 *A*, 64 *S*)
  - ⇒ fewer spills, greater scheduling flexibility
- All operations performed under mask (set of 8 64-bit mask registers)
  - ⇒ can vectorize loops with conditionals without scatter/gathers
- 32-bit integer and floating point memory refs and ops, IEEE
  - ⇒ double peak speed execution
- Allocating and non-allocating vector memory references
  - ⇒ better application cache behavior, efficient explicit communication
- Relaxed, architecture-defined memory ordering model with explicit synchronization instructions
  - ⇒ hardware is less constrained so common case can run faster

P Chip Summary

Scalar

- 2-way, o-o-o, 2-deep branch prediction, 64 active instructions
- Partitioned into address unit (A registers) and data unit (S registers)
- 8-way deep register shadowing ⇒ 1024 physical scalar registers

Vector

- 2 Vector Execution Pipes running at 800 MHz ⇒ 3.2 GFLOPS
- Double peak speed for packed 32-bit operations ⇒ 6.4 GFLOPS
- Load buffers for load renaming

Custom CMOS block

- Used once for each vector pipe, and once for scalar core
- Contains load buffers, registers, functional units and muxing

Memory

- 16 KByte Icache, 16 KByte Dcache (scalar only)
- Separate TLBs for Instruction, scalar, and vector
- 12.8 GB/s load bandwidth to Ecache (non-unit-stride)
- Up to 512 outstanding loads per SSP

Custom Block Design

- Custom CMOS Design
  - 800 MHz core - 400 MHz interface
  - Used for both vector pipes and scalar A/S block
  - Register file and mux to FUGs
  - 32 64-bit vector registers, 32 Elements per pipe
- 3 Functional Unit Groups
  - FUG1 - Int +, FP +, Int Comp, FP Comp, Logical, Insert Imm, Byte
  - FUG2 - Int *, FP *, and Shift
  - FUG3 - FP /, SQRT, Convert, POP, LZ, CPYS, ABS, Logical, Merge
- Load Buffers corresponding to 8 vector loads
  - 256 64-bit dwords outstanding (8x32 vector elements per pipe)
  - Used to *pre-load* data from memory

Design for Scalability

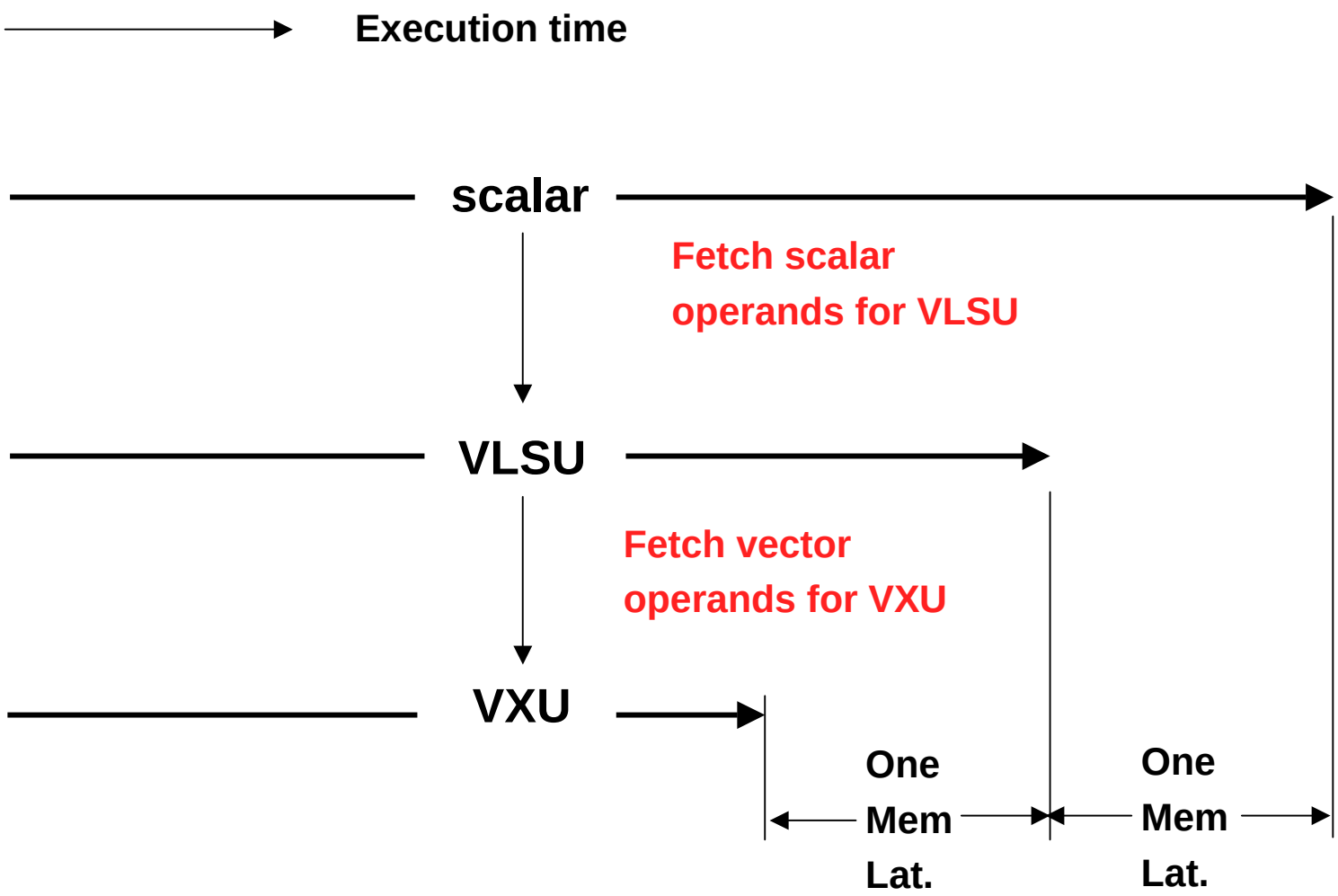
- Distributed shared memory (DSM) architecture
  - Low latency, load/store access to *entire* machine (tens of TBs)
  - Absolute minimum message latency via native vector instructions
- Decoupled vector memory architecture for latency tolerance
  - Thousands of outstanding references, flexible addressing
- Very high performance network
  - High bandwidth fine-grained transfers
- Architectural features for scalability
  - Remote address translation (no misses – RTT can hold all required translations on a remote node)
  - Global coherence protocol optimized for distributed memory
  - Fast synchronization (LSYNC, MSYNC, GSYNC, Fetch&Op)
- Parallel I/O scales with system size

CUG 2004

Cache Coherence

- Global coherence, but only cache memory from *local* node (8-32 GB)
  - Supports SMP-style codes up to 4 MSP (4-way sharing)
  - References outside this domain converted to non-allocate
  - Keeps directory entry and protocol simple
- Explicit cache allocation control
  - Per instruction hint for vector references
  - Per page hint for scalar references
  - Use non-allocating references for explicit communication or to avoid cache pollution (PUT example)
- Coherence directory stored on the M chips (rather than in DRAM)
  - Low latency and *really* high bandwidth to support vectors
    - Typical CC system: 1 directory update/proc/(100-200 ns)
    - Cray X1: 3.2 directory updates/MSP/ns

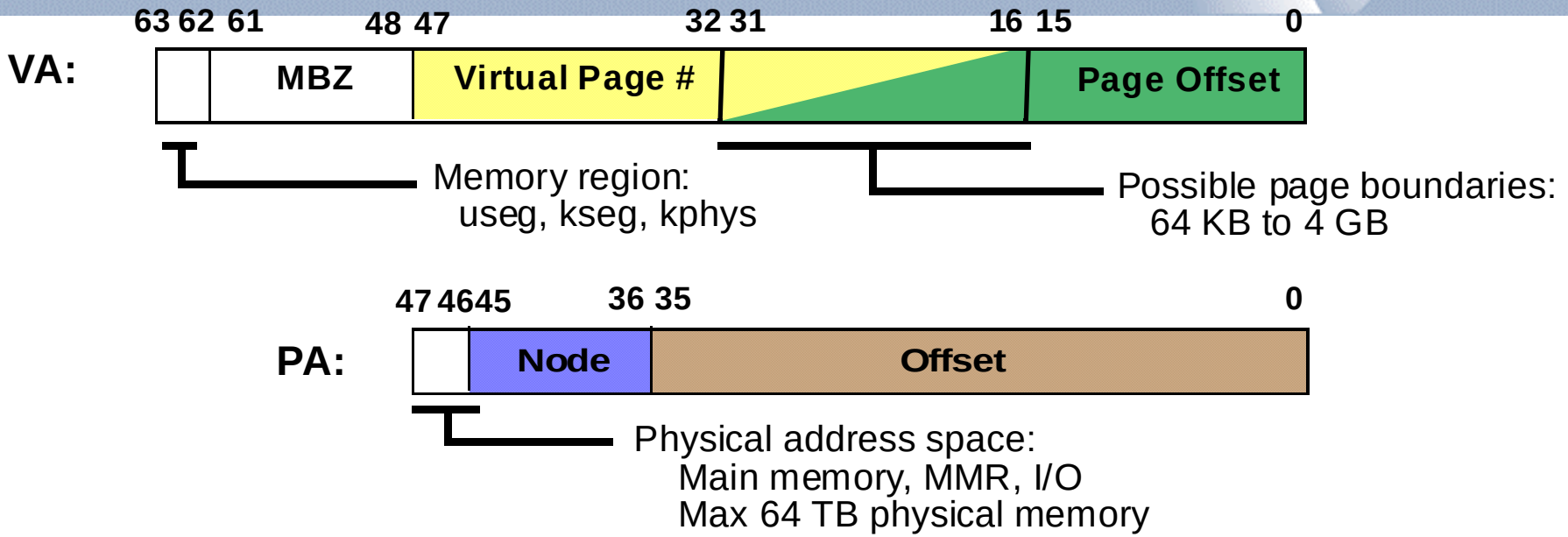
Decoupling on X1



Addressing on X1

- X1 is *globally* addressible, meaning any processor has HW support to read/write any memory location in the machine
- X1 HW has *two* address translation mechanisms, the ordinary processor TLB ('flexible' mode, aprun .. -F .. a.out), or *remote* translation ('accelerated' mode, aprun ..-A .. a.out)
  - *same* user code runs in either case
  - *shared* memory jobs use processors and memory of *one* node
  - *DM* jobs (SPMD model) have virtual processor ID as part of address
    - at execution time Cray startup code sets upper bits of all virtual address segments for all processors to contain virtual node bits (starting at zero) and virtual processor bits, etc.
    - access to 'processor' part of address done with libraries (MPI, SHMEM) or through language extensions (CAF, UPC). For MPI, bcopy(x_from, x_to, n) moves data via vector load and store: x_to(1:n) = x_from(1:n)
    - OS-set processor TLBs and RTTs map virtual nodes to physical nodes and produce complete physical addresses needed for HW to route requests

Address Translation



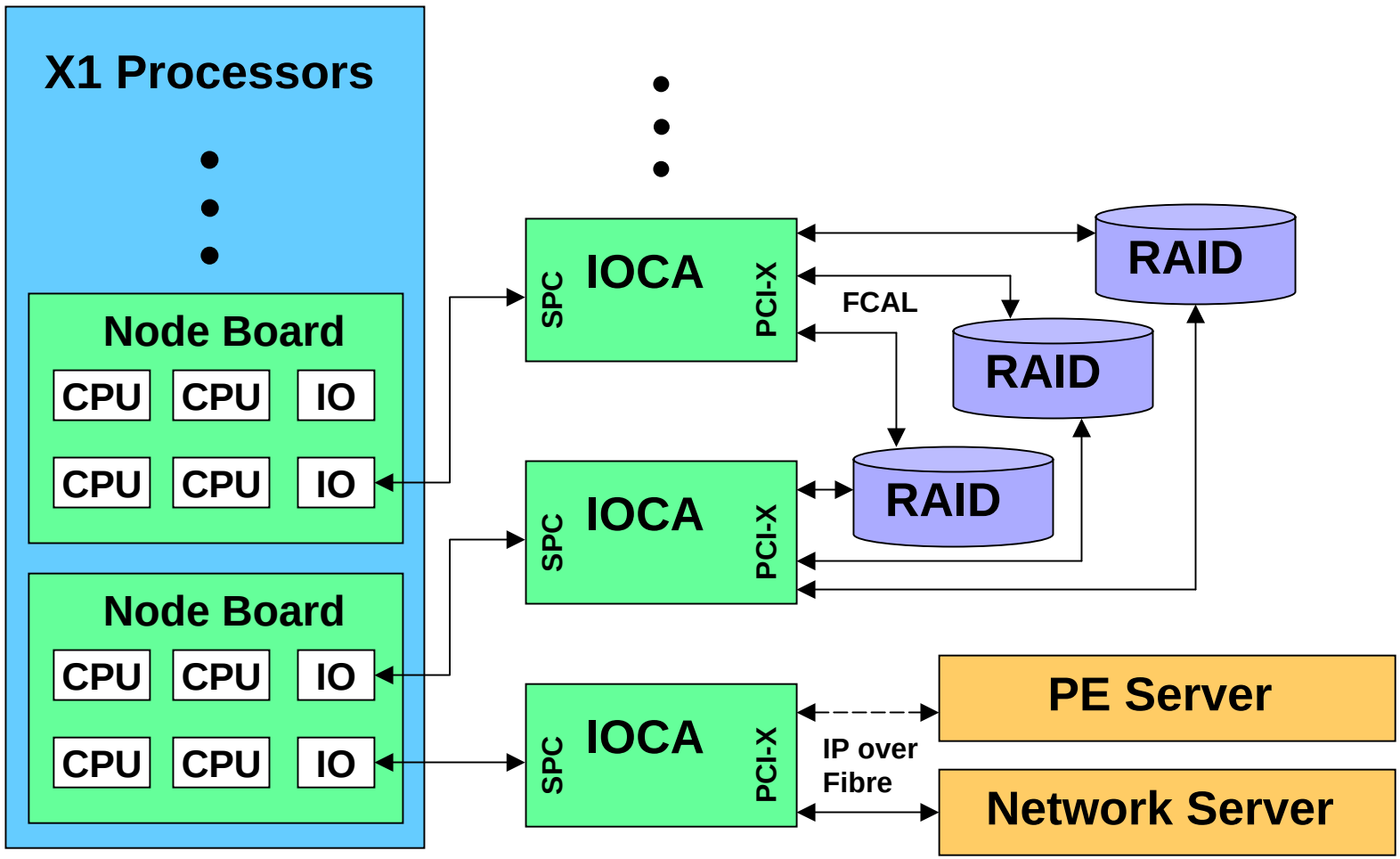
- *Source* translation uses 256 entry TLBs with multiple page sizes: virtual page # bits translated locally
  - 48 bit VA gets translated to 46 bit PA = 10 b node + 36 b offset
  - allows *non-contiguous* multi-node jobs to improve system utilization
  - aprun … **-F** … a.out ← 'F' instructs HW to disable remote translation

Address Translation (cont)

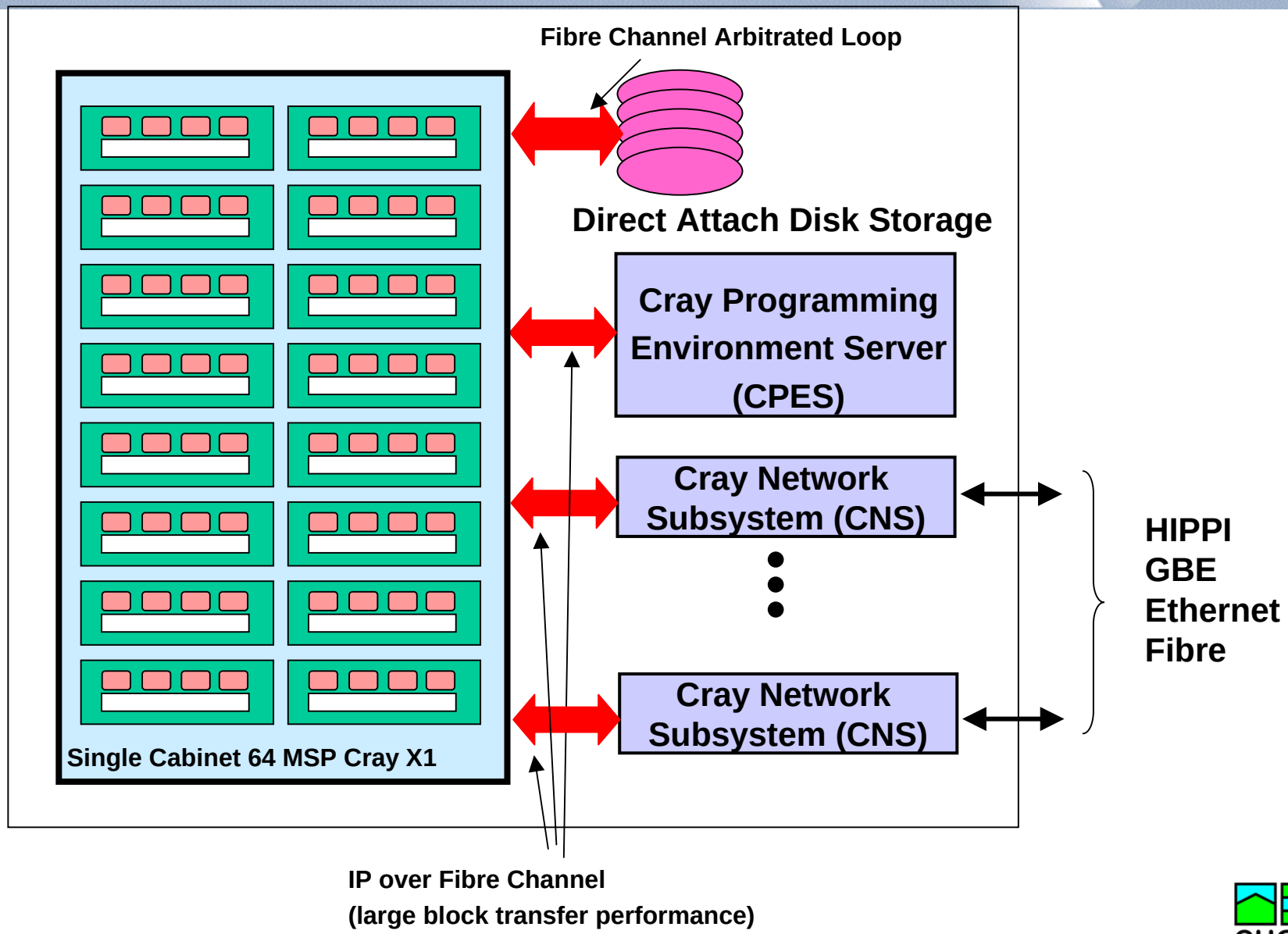
- *Remote* translation bypasses local TLB & uses table of 64K entries (spread across 16 M chips on each node) with 16 MB pages to translate incoming virtual offset → physical offset on local node
  - VA → physical node (used to route RVA) + RVA (translated remotely)
    - *virtual* node bits + BasePhysNode → *physical* node
    - checks to see if Vnode = MyNode and Vnode < NodeLimit
    - requires physically contiguous nodes
  - sends *virtual* offset part of VA to remote node for translation into *physical* offset on remote node
  - aprun … **-A** … a.out ← 'A' instructs HW to enable RTT
  - TLB only need hold translations for *one* node ⇒ *X1 can reference remote memory with **no** TLB misses*

CUG 2004

Cray X1 I/O Architecture



Cray X1 System IO Architecture



256 Processor Cray X1 System

3.2 Tflops