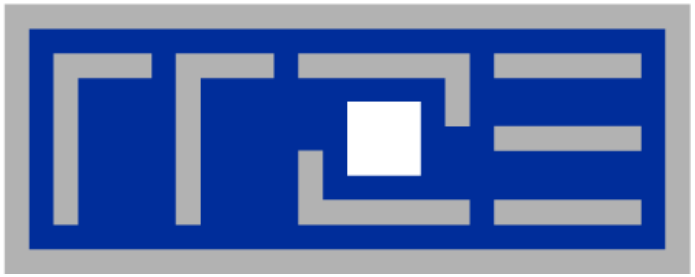


Programming Techniques for Supercomputers:

Parallel Computers

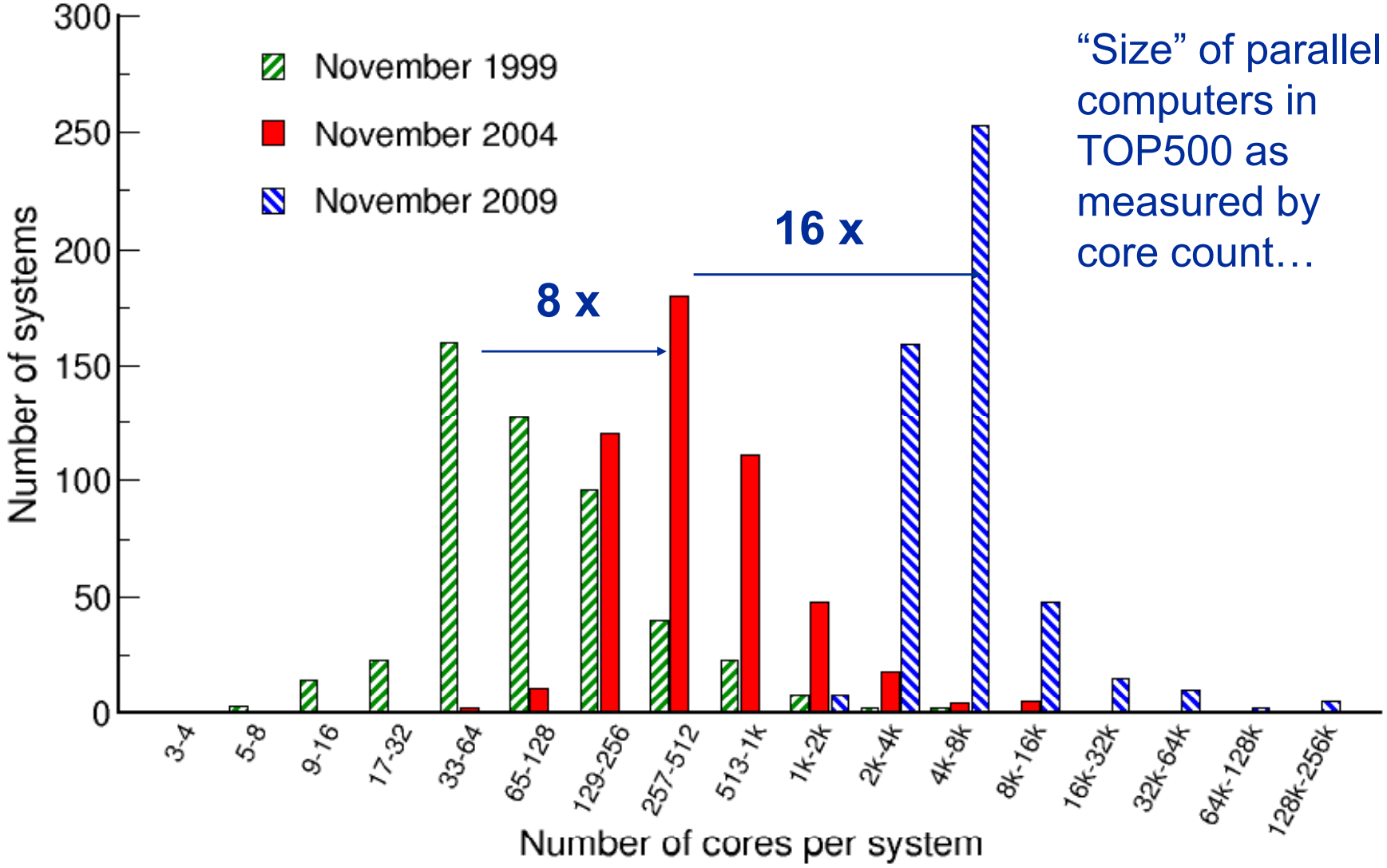
Introduction
Shared-memory computers
Distributed-memory computers / Hybrid systems
Networks - Introduction

Prof. Dr. G. Wellein[a,b] , Dr. G. Hager[a] , Dr. J. Treibig[a] , J. Habich[a]

[a]HPC Services – Regionales Rechenzentrum Erlangen
[b]Department für Informatik

University Erlangen-Nürnberg, Sommersemester 2011

- **Parallel computing is here to stay from Laptop/Desktop to TOP500**



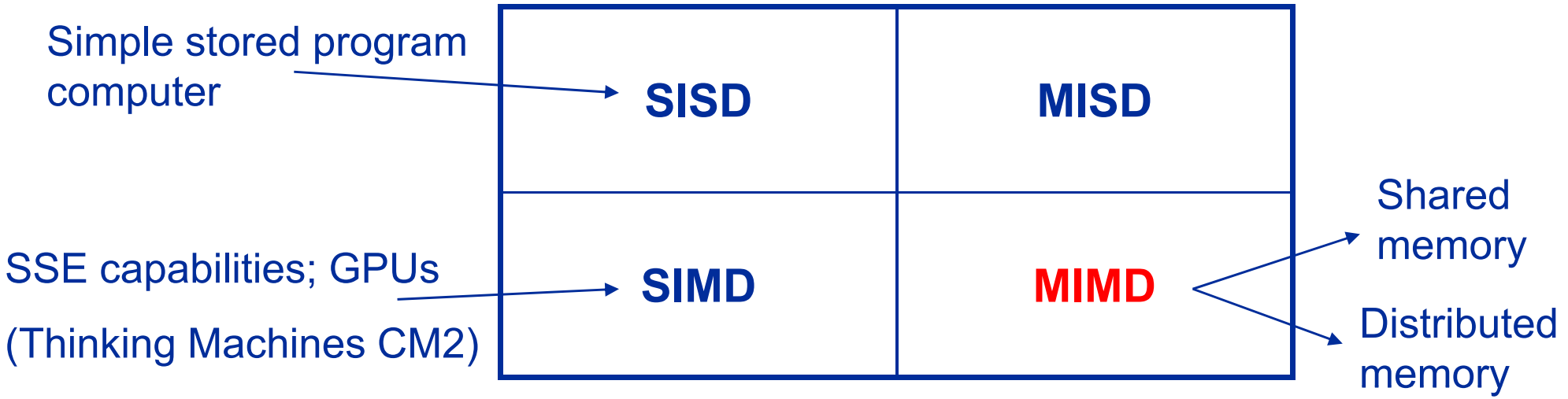
"Size" of parallel computers in TOP500 as measured by core count…

Parallel computers – Introduction

- „Parallel Computing“ : A number of compute elements (cores) solve a problem in a cooperative way

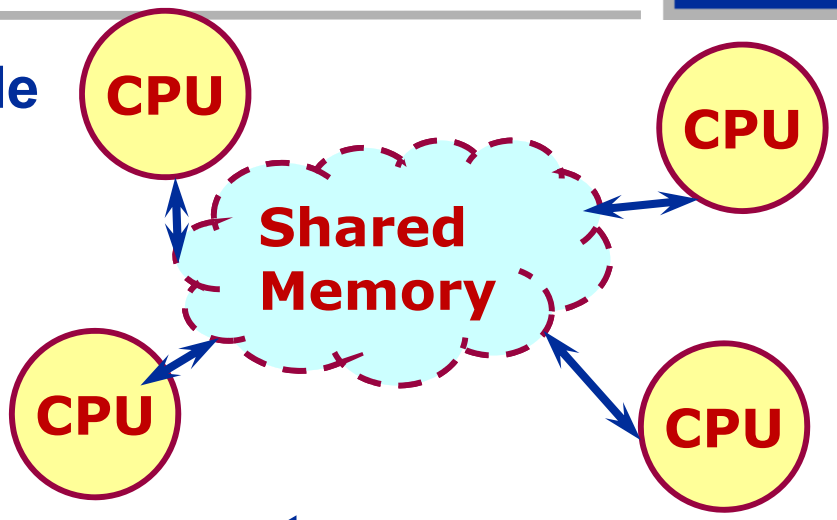
- Parallel Computer: A number of compute elements (cores) connected such way to do parallel computing for a large set of applications

- Classification according to Flynn: Multiple Instruction Multiple Data (MIMD)



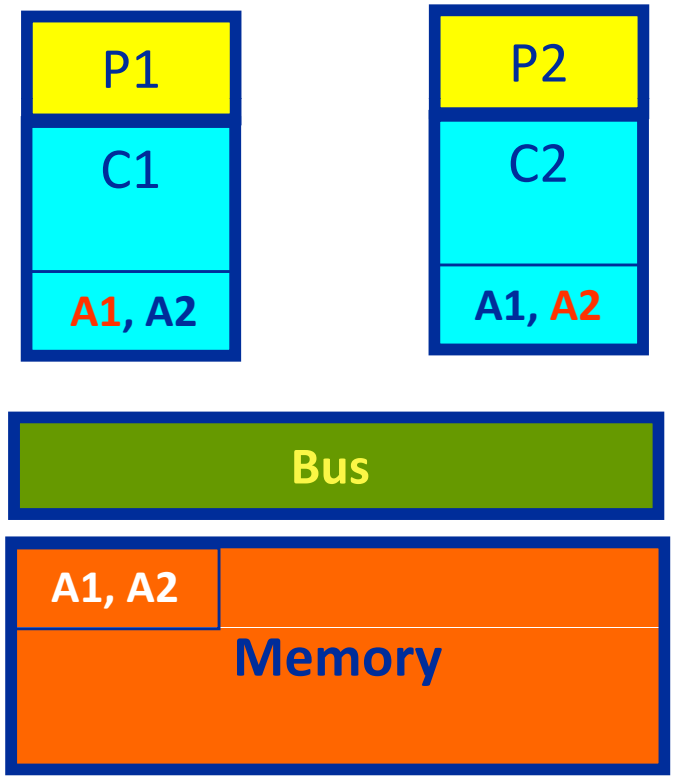
Parallel computers – Shared-Memory Architectures

- **Shared memory computers provide**
  - a single shared address space (memory) for all processors
  - All processors share the same view of the address space!



- **Two basic categories of shared memory systems**
  - Uniform Memory Access (UMA):
    Memory is equally accessible to all processors with the same performance (Bandwidth & Latency)
  - cache-coherent Non Uniform Memory Access (ccNUMA):
    Memory is physically distributed but appears as a single address space:
    Performance (Bandwidth & Latency) is different for local and remote memory access
  - Copies of the same cache line may reside in different caches → Cache coherence protocols guarantees consistency all time (for UMA & ccNUMA)
  - Cache coherence protocols do not alleviate parallel programming for shared-memory architectures!

- **Data in cache is only a copy of data in memory**
  - Multiple copies of same data on multiprocessor systems
  - Cache coherence protocol/hardware ensure consistent data view
  - Without cache coherence, shared cache lines can become clobbered: (Cache line size = 2 WORD; A1+A2 are in a single CL)



P1	P2
Load A1	Load A2
Write A1=0	Write A2=0

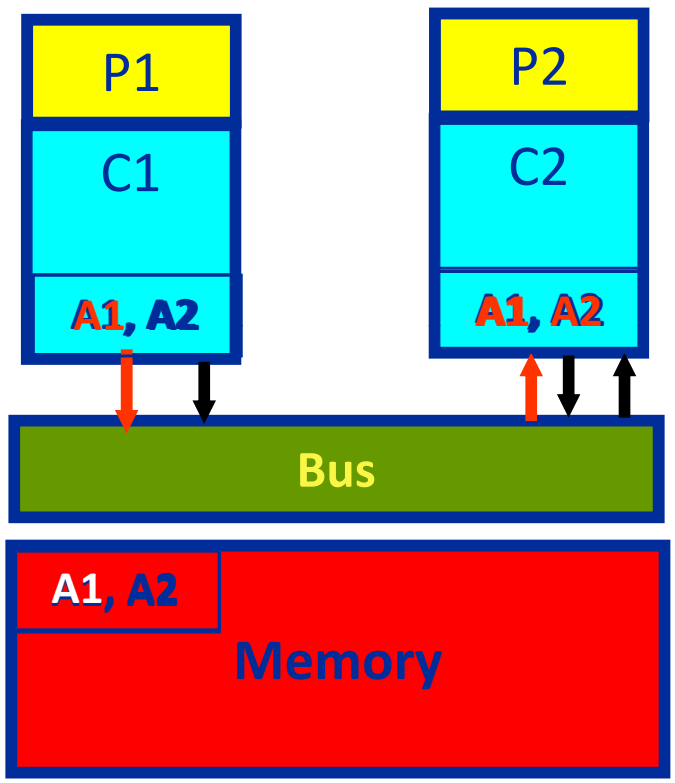
Write-back to memory leads to incoherent data

A1, A2 | A1, A2 | A1, A2

C1 & C2 entry can not be merged to:

A1, A2

- Cache coherence protocol must keep track of cache line status



P1	P2
Load A1	Load A2
Write A1=0:	
1. Request exclusive access to CL	
2. Invalidate CL in C2	
3. Modify A1 in C1	
	Write A2=0:
	1. Request exclusive CL access
	2. CL write back+ Invalidate
	3. Load CL to C2
	4. Modify A2 in C2

t

C2 is exclusive owner of CL

Parallel computers – Cache coherence

- **Cache coherence can cause substantial overhead**
  - may reduce available bandwidth
- **Different implementations**
  - Snoop: On modifying a CL, a CPU must broadcast its address to the whole system
  - Directory, “snoop filter”: Chipset (“network”) keeps track of which CLs are where and filters coherence traffic
- **Directory-based ccNUMA can reduce pain of additional coherence traffic**
- **But always take care:**

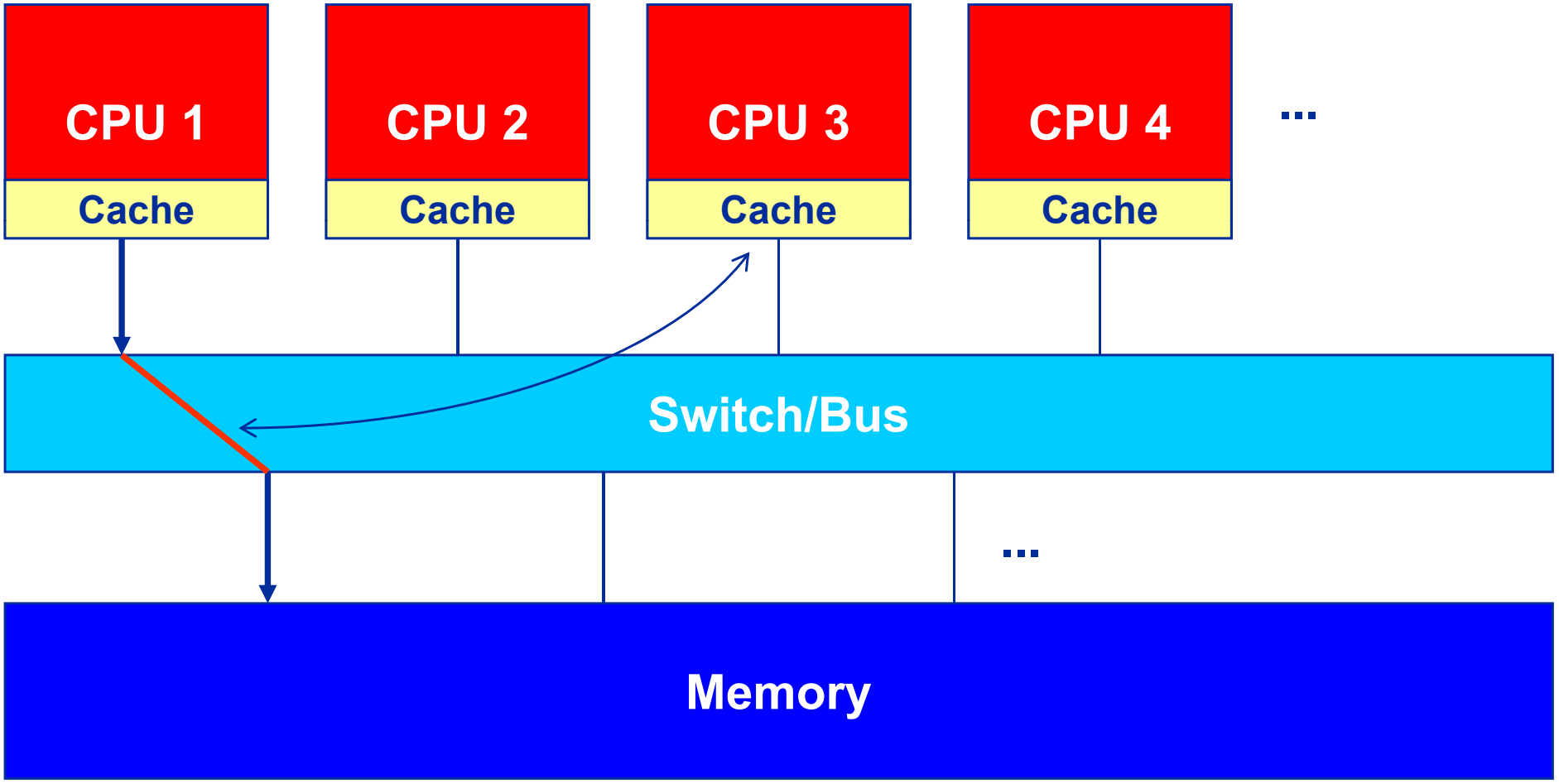
Multiple processors should never write frequently to the same cache line (“false sharing”)!

- **Widespread cache coherence protocol: MESI protocol**

- **A cache line can have four different states:**
  - **M**odified: Cache line has been modified in this cache, and it resides in no other cache. Cache line needs to be evicted to ensure memory consistency
  - **E**xclusive: Cache line has been read from main memory but not (yet) modified. There are no (valid) copies in other caches
  - **S**hared: Cache line has been read from memory but no modified. There may be valid copies in other caches
  - **I**nvalid: This cache line does not reflect any sensible data. Usually this happens if the cache line was in **S** state and another processor requested exclusive ownership

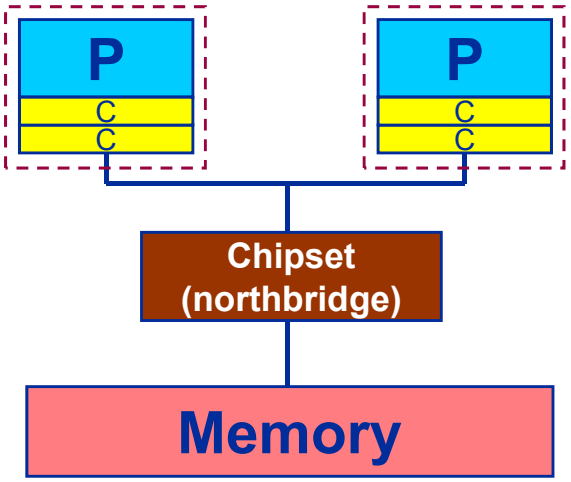
Parallel computers: Shared-memory: UMA

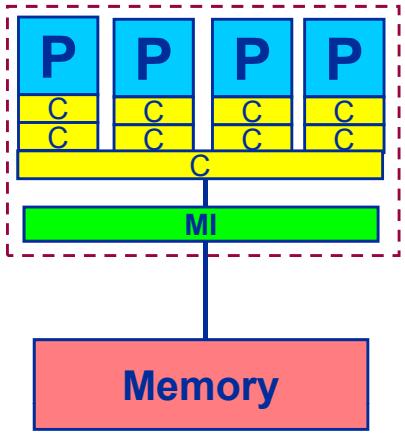
- **UMA Architecture: switch/bus arbitrates memory access**
  - Special protocol ensures cross-CPU cache data consistency
  - Flat memory – also known as „Symmetric Multi-Processor“ (SMP)



Parallel computers: Shared-memory: UMA / Bus based

- **Worst case: bus system provides single bandwidth to multiple processors**
  - Only one CPU at a time can use the bus and access memory at any one time – No need to provide for faster memory
  - Collisions occur frequently, causing one or more CPUs to wait for “bus ready” (contention)
  - Even worse: Shared memory bus in current multi-core chips further reduce available bandwidth/peak FLOP balance!
  - Price/performance ratio was good for certain applications for a long time
  - Similar concept in a single multi-core chip

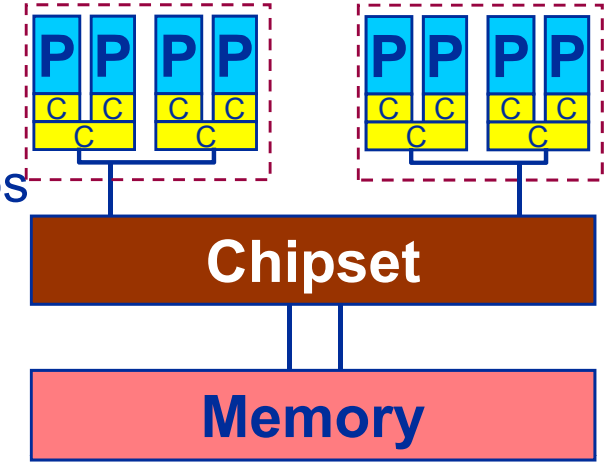




Parallel computers: Shared Memory: UMA / Crossbar based

- **Best case: memory crossbar switch provides separate data path to memory for each CPU**
  - Can saturate full memory bandwidth of every CPU concurrently
  - Bus contention occurs only if same memory module is accessed by more than one CPU
  - Memory interleaving is a must
  - Example:
    - NEC SX-9 node: 16 CPUs, 256 GB/sec each, 4096 GB/sec per node bandwidth

- **Reality: Compromises are made**
  - E.g. 2 of 8 cores can use full bandwidth concurrently
  - May be advantageous not to use all cores on a node
  - Example: Intel Clovertown based on 2 quad-core chips
    2 sockets (8 cores), 10.6 GB/sec each,
    21.2 GB/sec node bandwidth; i.e. 2.66 GB/s per core
    - However, not all of this bandwidth might be available in practice



Parallel computers: Shared Memory: UMA Nodes

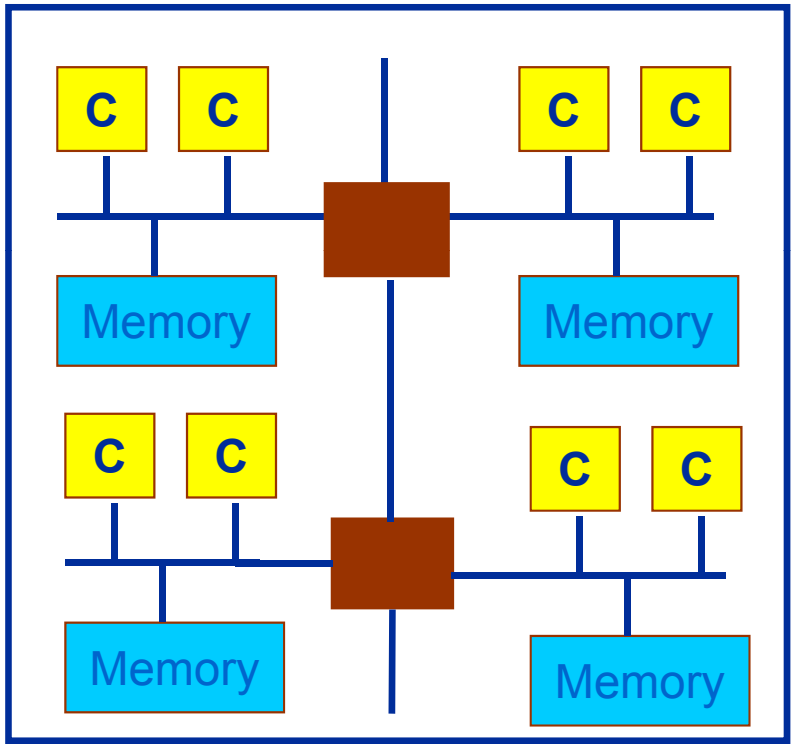
- **Examples:**
  - Dual Xeon nodes, e.g. old RRZE (Intel) Transtec Cluster
  - NEC vector systems
  - Your dual core laptop computer

- **Advantages**
  - Cache Coherence (see below) is "easy" to implement
  - Easy to optimize memory access
  - Incremental parallelization
  - Large memory configuration in a single address space

- **Disadvantages**
  - Memory bandwidth and price (!) often limit scalability
    (2 – 8 cores per UMA node)

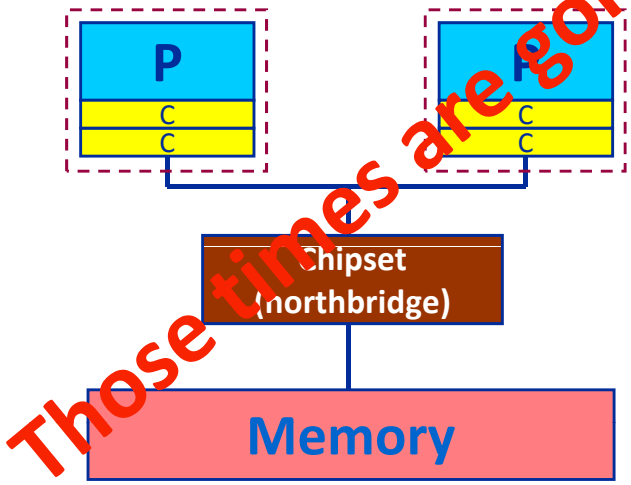
Parallel shared memory computers: ccNUMA/Node Layout

- **ccNUMA:**
  - Single address space although physically distributed memory through proprietary hardware concepts (e.g. NUMALink in SGI systems)
- **Advantages:**
  - Memory bandwidth: scalable
  - Systems with up to 1024 CPUs+ are available
- **Disadvantages:**
  - Cache Coherence hard to implement / expensive
  - Performance depends on access to local or remote memory
- **Examples: AMD Opteron nodes, SGI Altix, Intel Nehalem nodes – all modern multi-socket nodes**

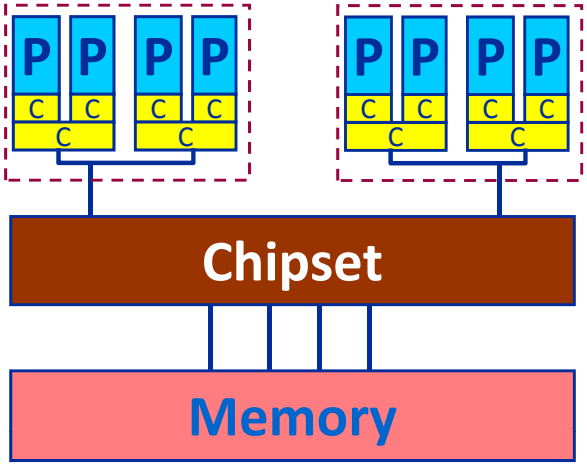


Parallel shared memory computers: Some examples

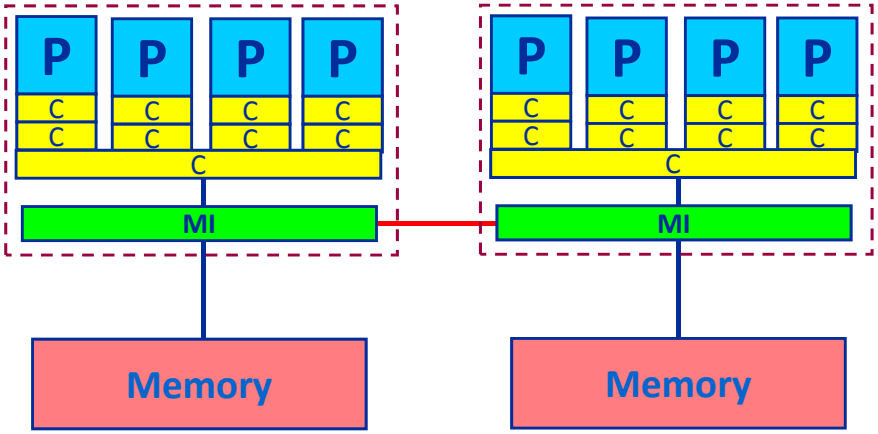
- Dual CPU Intel Xeon node



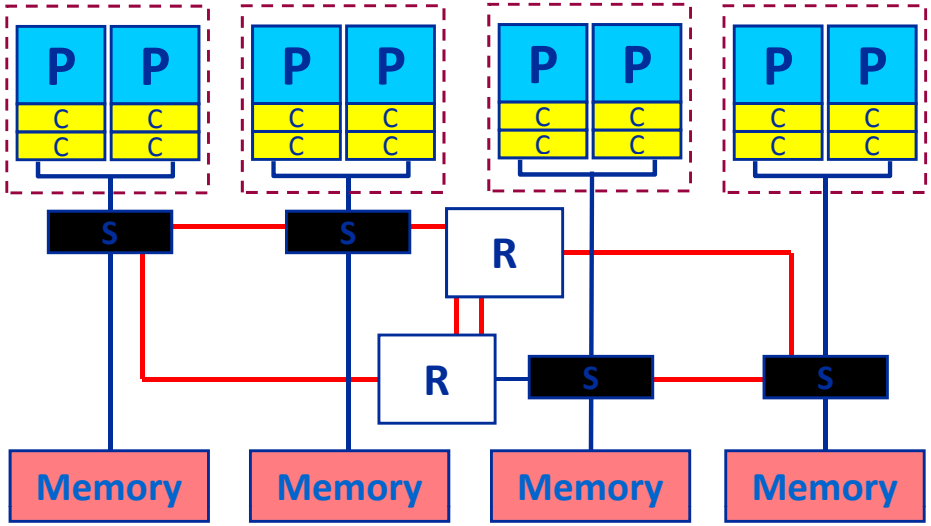
- Dual Intel "Core2" QC node



- Dual AMD Opteron or Intel Nehalem (QC) node

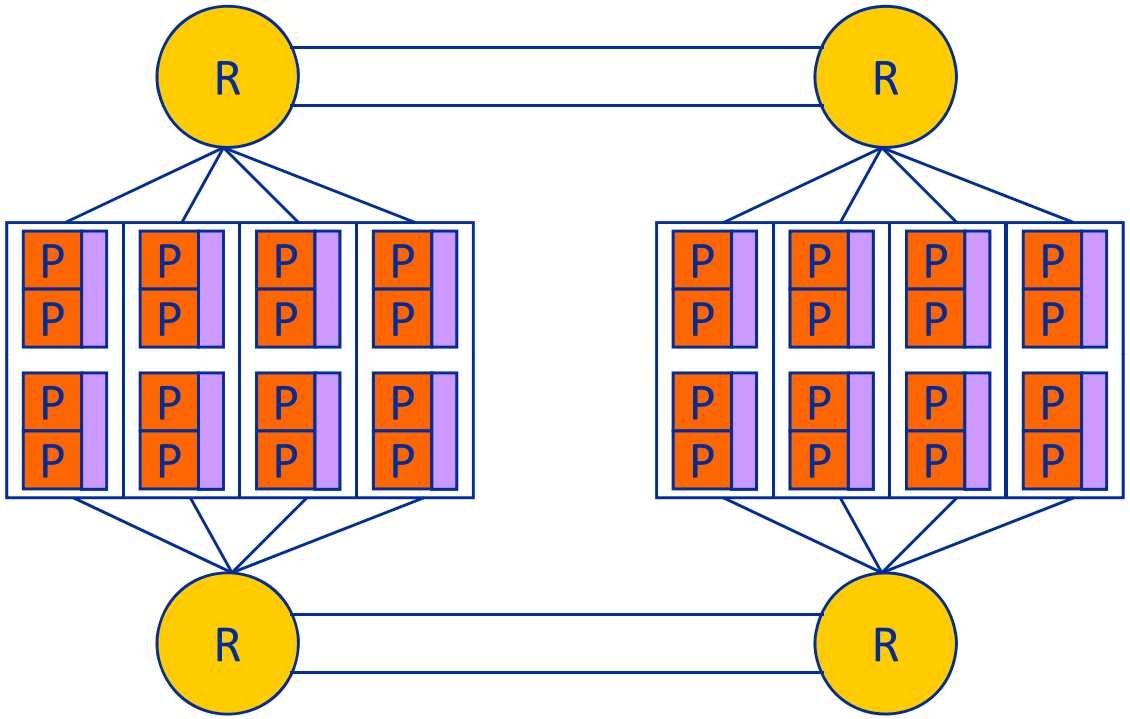


- SGI Altix

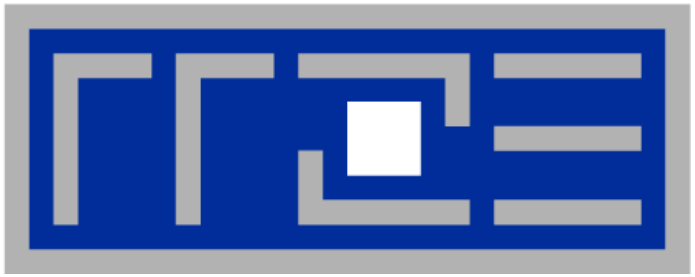


Parallel shared memory computers: larger ccNUMA nodes

- **System Architecture cont'd**
  - Layout for 32-socket SGI Altix system →
  - … and you know this can get really large!



- **Who keeps track of the contents of all those caches?**
  - **Cache coherence mechanisms**
- **How do we make sure that memory pages are as close as possible to the CPUs they are needed on?**
  - **ccNUMA optimization**

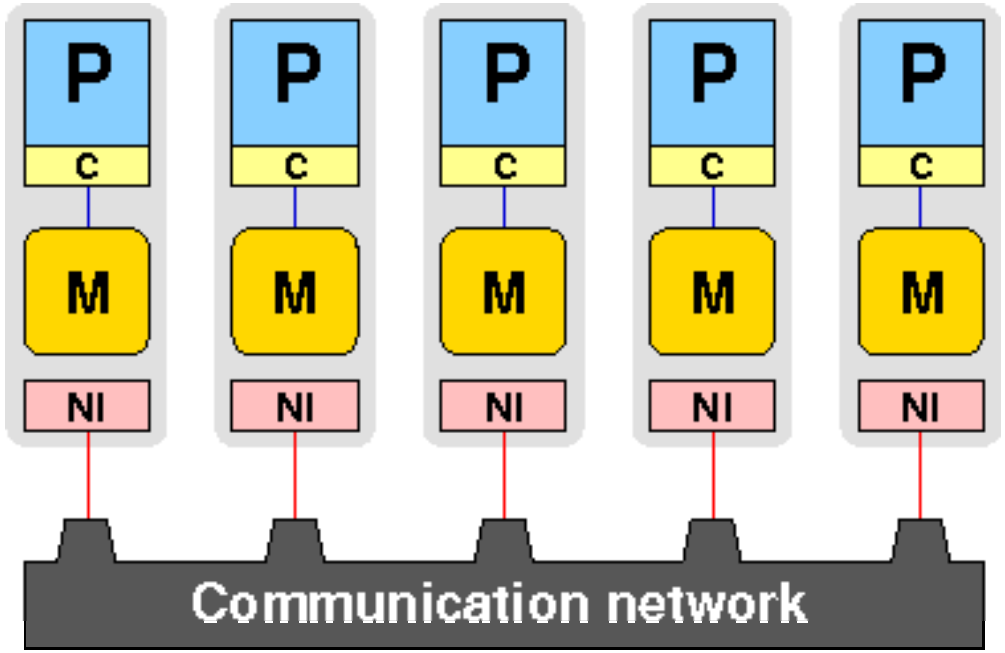


Distributed-memory computers & hybrid systems

Parallel distributed-memory computers: Basics

Pure distributed-memory parallel computer:

- Each processor P is connected to exclusive local memory (MM) and a network interface (NI) → "Node"
- A (dedicated) communication network connects alls nodes
- No global cache-coherent shared address space → **N**o **R**emote **M**emory **A**ccess (**NORMA**)
- Data exchange between nodes: Passing messages via network ("Message Passing")
- Some architectures provide limited remote memory access for speeding up message passing, e.g. through a global NON-COHENRENT address space (NUMA)



Prototype of first PC clusters:

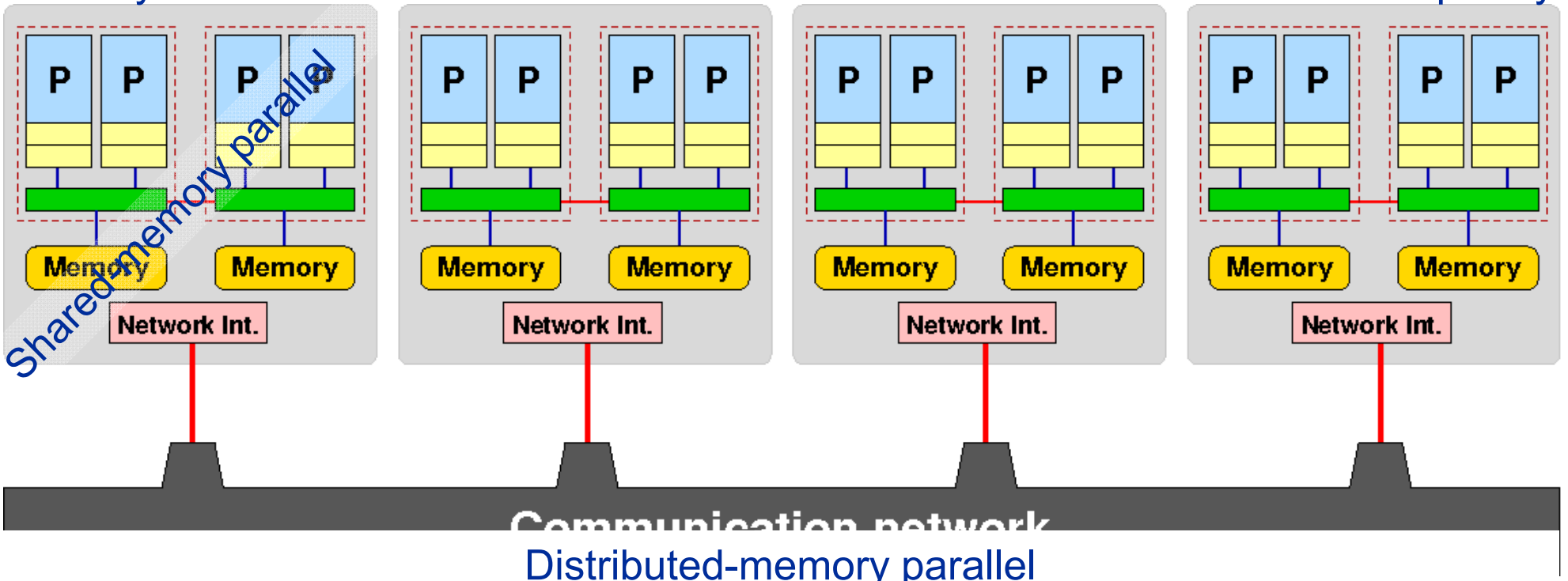
- Node: Single core/CPU PC
  Network: Ethernet

First Massively Parallel Processing architectures: CRAY T3D/E, Intel Paragon

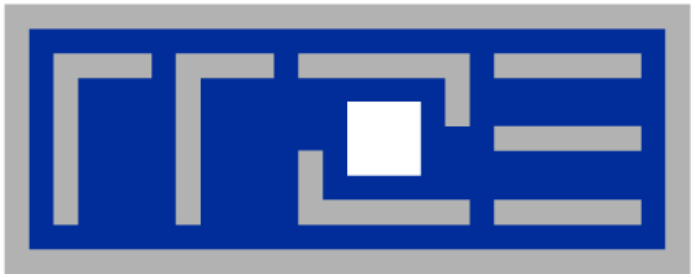
Parallel distributed-memory computers: Hybrid system

Standard concept of most modern large parallel computers: **Hybrid**/hierarchical

- Compute node is a 2- or 4-socket shared memory compute nodes with a NI.
- Communication network (GBit, Infiniband) connects the nodes
- Price / (Peak) Performance is optimal; Network capabilities / (Peak) Perf. gets worse
- Parallel Programming? Pure Message Passing is standard. Hybrid programming?
- Today: GPUs / Accelerators are added to the nodes to further increase complexity



Distributed-memory parallel



Networks

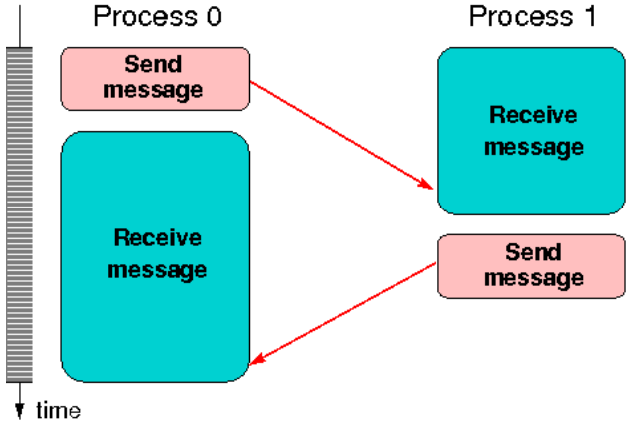
What are the basic ideas and performance characteristics of modern networks…

Networks – Basic performance characteristics

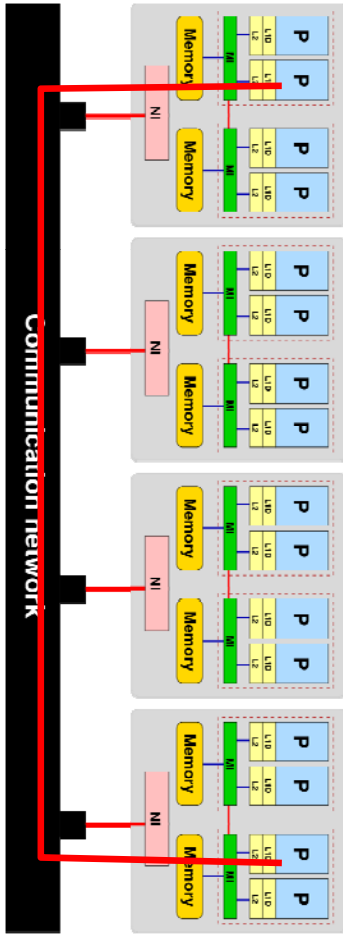
- **Evaluate the network capabilities to transfer data**
- **Use the same idea as for main memory access:**
  - Total transfer time for a message of N Bytes is:

$$T = T_L + N/B$$

  - T_L is the latency (transfer setup time [sec]) and B is asymptotic (N → oo) network bandwidth [MBytes/sec]
- **Consider simplest case ("Ping Pong")**
  - Two processors in different nodes communicate via network ("Point-to-point")
  - A single message of N Bytes is sent forward and backward



Overall data transfer is 2 N Bytes!

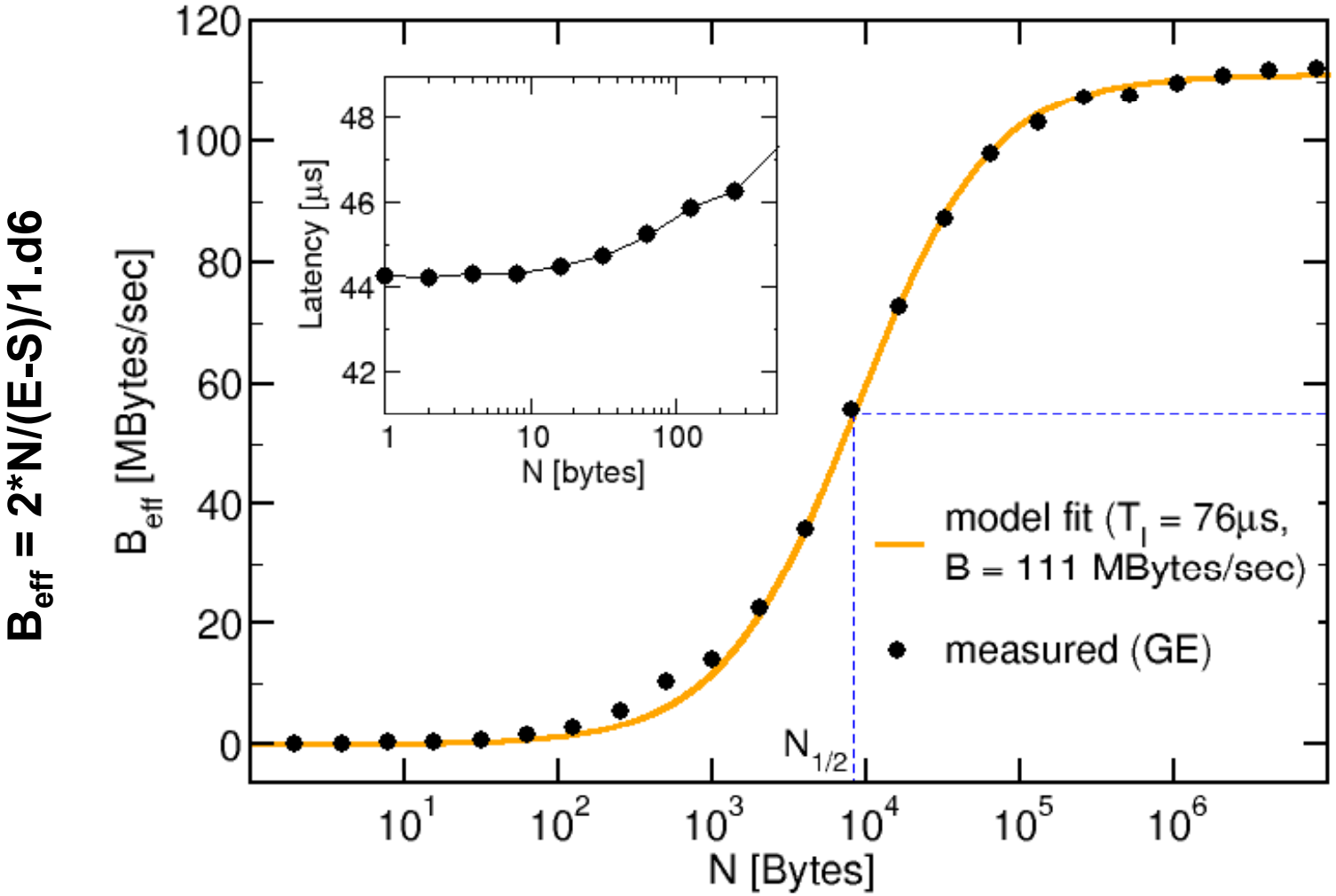


Ping-Pong benchmark (pseudocode)

```
1 myID = get_process_ID()
2 if(myID.eq.0) then
3    targetID = 1
4    S = get_walltime()
5    call Send_message(buffer,N,targetID)
6    call Receive_message(buffer,N,targetID)
7    E = get_walltime()
8    MBYTES = 2*N/(E-S)/1.d6 ! MBytes/sec rate
9    TIME = (E-S)/2*1.d6 ! transfer time in microsecs
10 ! for single message
11 else
12   targetID = 0
13   call Receive_message(buffer,N,targetID)
14   call Send_message(buffer,N,targetID)
15 endif
```

Ping-Pong benchmark for GBit-Ethernet (GigE) network

$N_{1/2}$: Message size where 50% of peak bandwidth is achieved



Asymptotic bandwidth B=111 Mbytes/sec ←→ 0.888 GBit/s

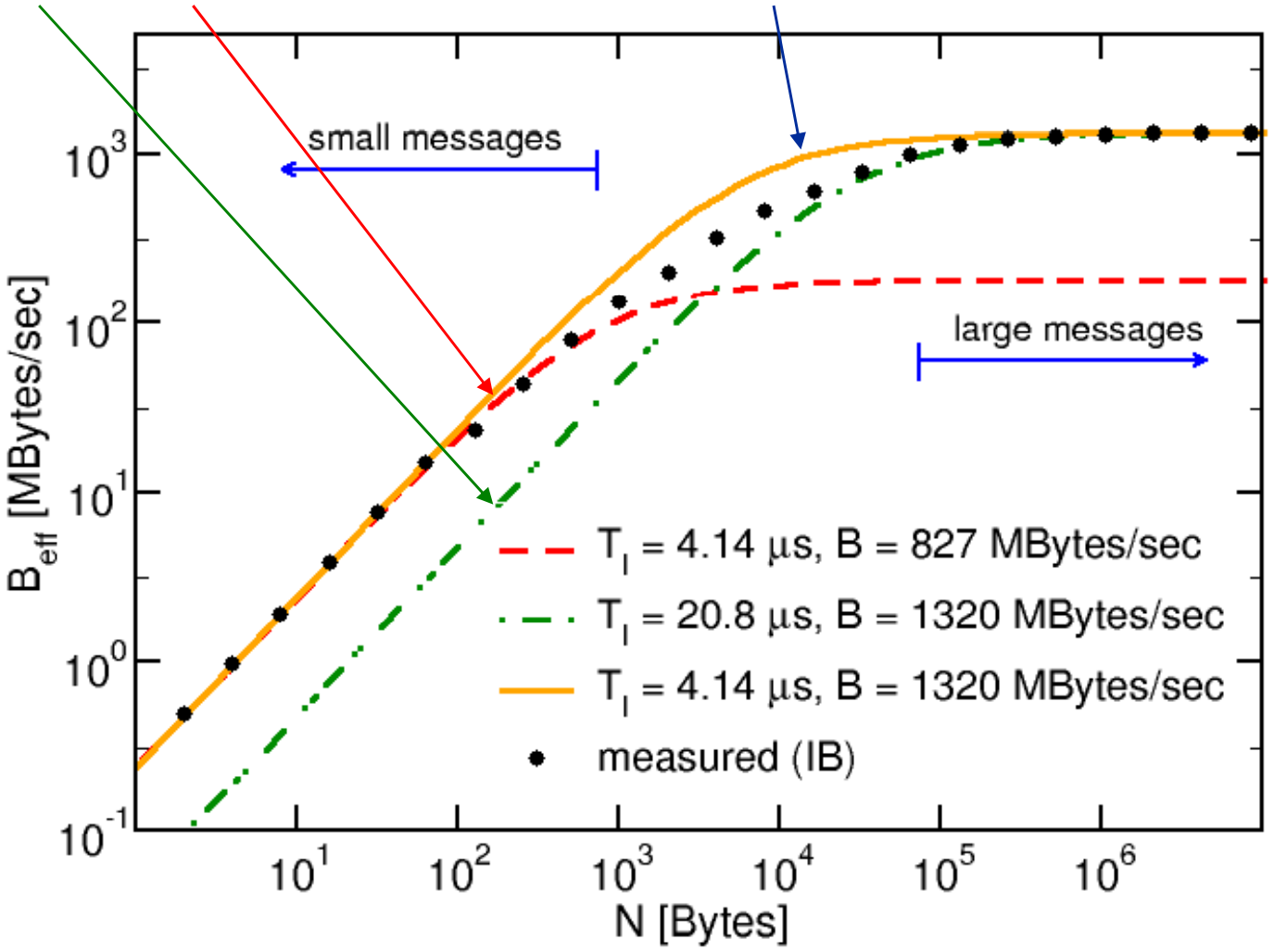
Latency (N → 0):

Only qualitative agreement:

44 µs vs. 76 µs

Ping-Pong benchmark for DDR Infiniband (DDR-IB) network

Determine B and T_L independently and combine them

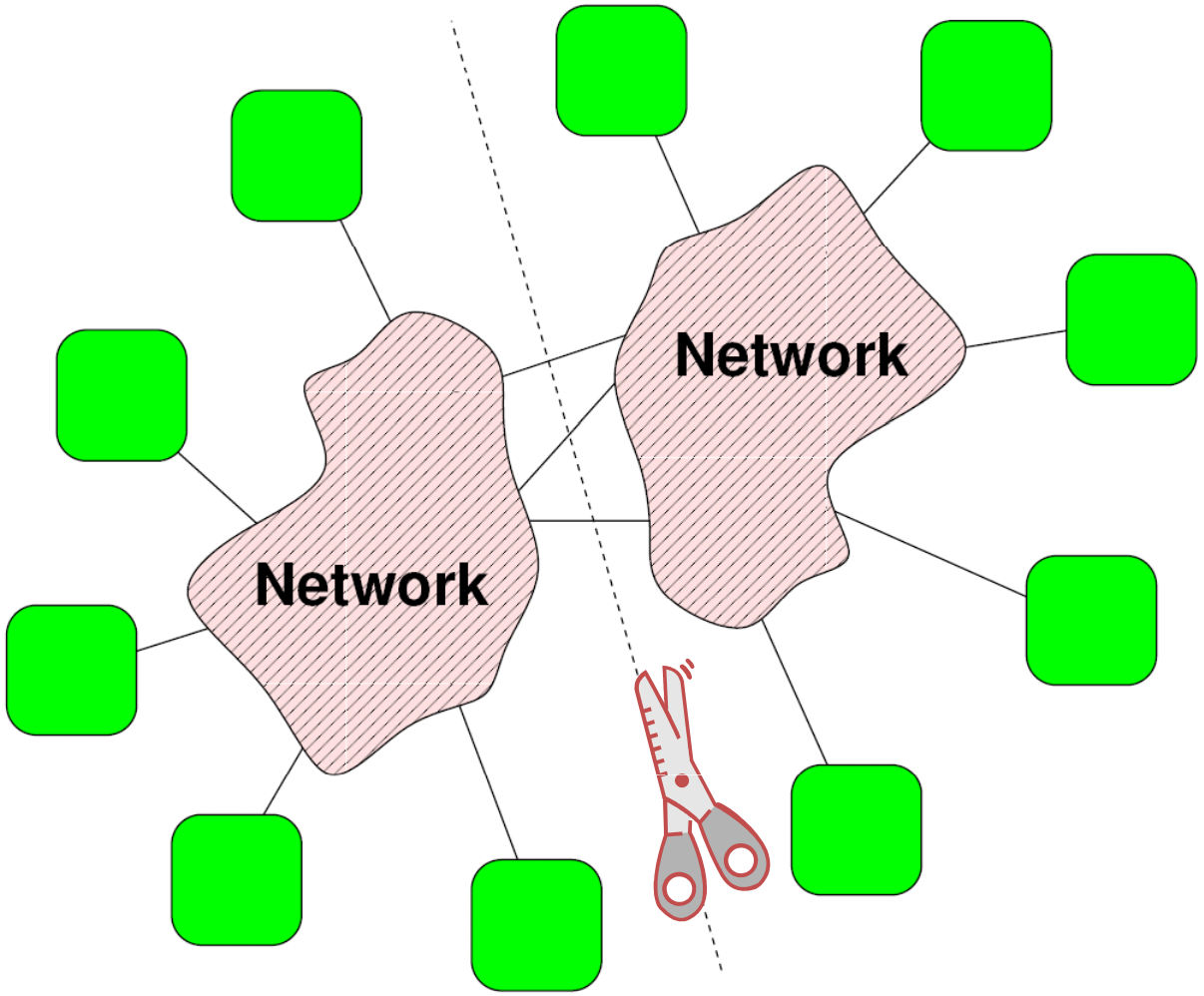


Networks – Basic performance characteristics

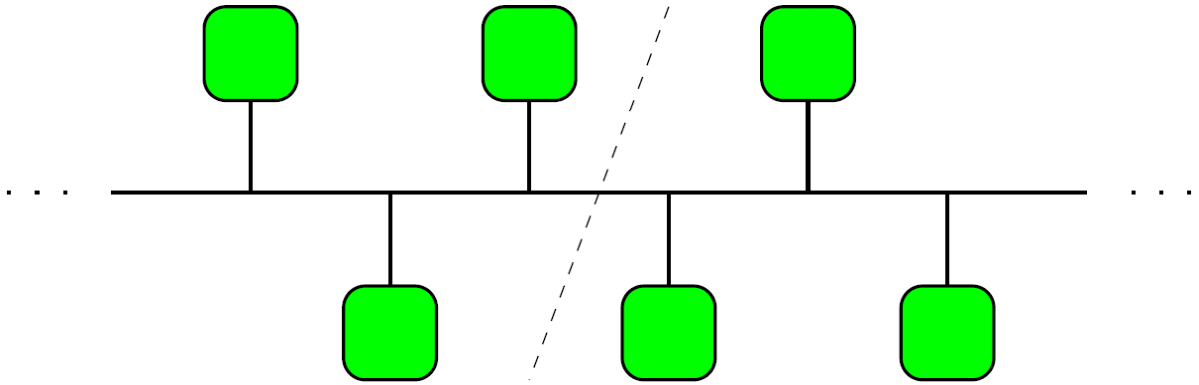
- **"First Principles" modeling of $B_{eff}(N)$ provides good qualitative results but quantitative description in particular of latency dominated region (N small) may fail because**
  - Overhead for transmission protocols, e.g. message headers
  - Minimum frame size for message transmission, e.g. TCP/IP over Ethernet does always transfer frames with N>1
  - Message setup/initialization involves multiple software layers and protocols; each software layer adds to latency; hardware only latency is often small
  - As the message size increases the software may switch to different protocol, e.g. from "eager" to "rendezvous"
- **Typical message sizes in applications are neither small nor large → $N_{1/2}$ value is also important: $N_{1/2} = B * T_L$**
- **Network balance: Relate network bandwidth (B or $B_{eff}(N_{1/2})$) to computer power (or main memory bandwidth) of the nodes**

Networks: Topologies & Bisection bandwidth

- Network bisection bandwidth B_b is a general metric for the data transfer "capability" of a system
- Minimum sum of the bandwidths of all connections cut when splitting the system into two equal parts
- More meaningful metric when comparing systems: Bisection BW per core or per node, B_b/N
- Bisection BW depends on
  - Bandwidth per link
  - Network topology
- Uni- or Bi-directional band



Network topologies: Bus



- Bus can be used by one connection at a time
- Bandwidth is shared among all devices
- Bisection BW is constant:
  → $B_b/N \sim 1/N$
- Collision detection, bus arbitration protocols must be in place
- Examples: PCI bus, memory bus of multi-core chips, diagnostic buses, internal ring bus of the Cell processor,…
- Advantages
  - Low latency
  - Easy to implement
- Disadvantages
  - Shared bandwidth, not scalable
  - Problems with failure resiliency (one defective agent may block bus)
  - Fast buses for large N require large signal power

Network topologies: Switches and Fat-Trees

- Standard clusters are built with switched networks
- Compute nodes (“devices”) are split up in groups – each group is connected to a single (small) (non-blocking crossbar-)switch (“leaf switches”)
- Leaf switches are connected with each other using an additional switch hierarchy (“spine switches”) or directly (for small configs)

- In switched networks the “distance” between any two devices is heterogeneous (number of “hops” in switch hierarchy)

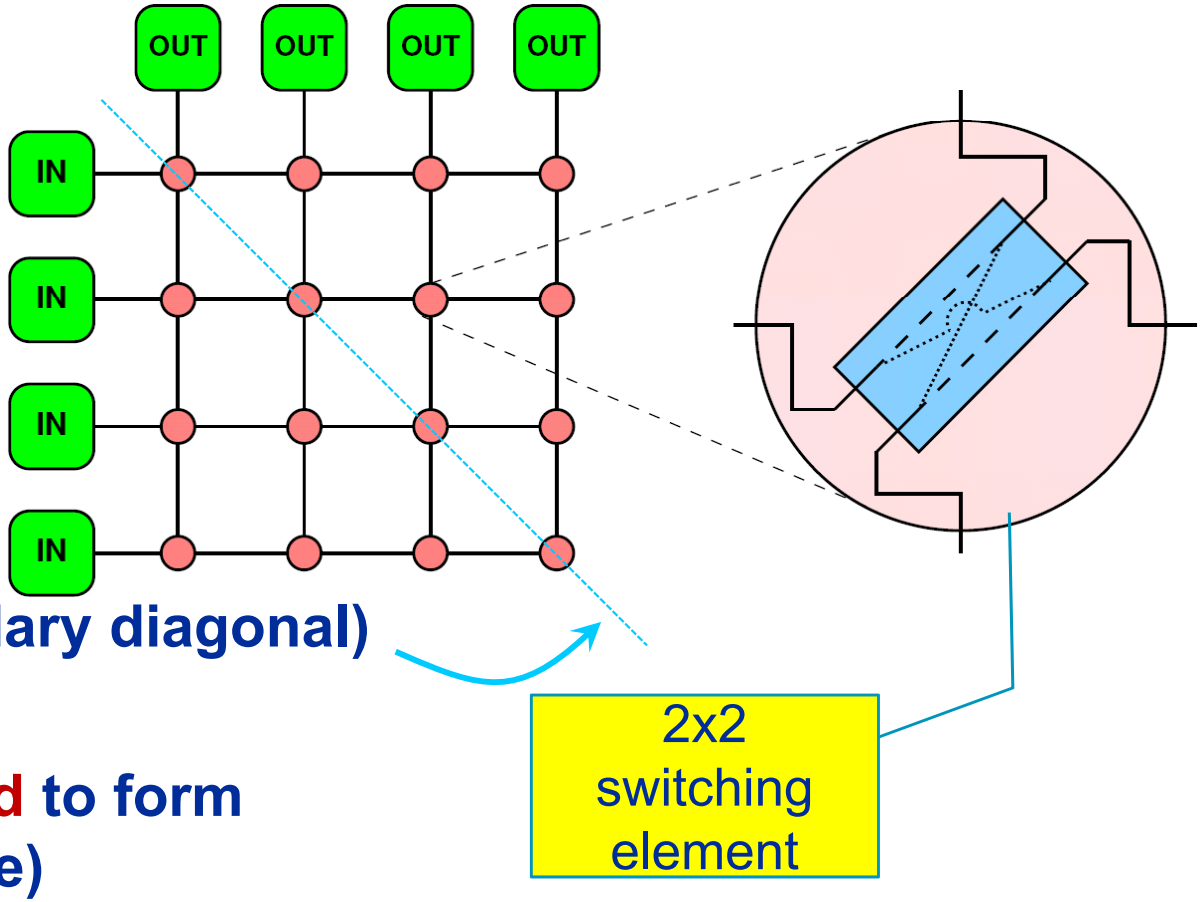
→ Diameter of a network: The maximum number of hops required to connect two arbitrary devices

Example: Diameter of bus=1

- “Perfect” world: “Fully non-blocking”, i.e. any choice of N/2 disjoint device pairs can communicate at full speed

Non-blocking crossbar

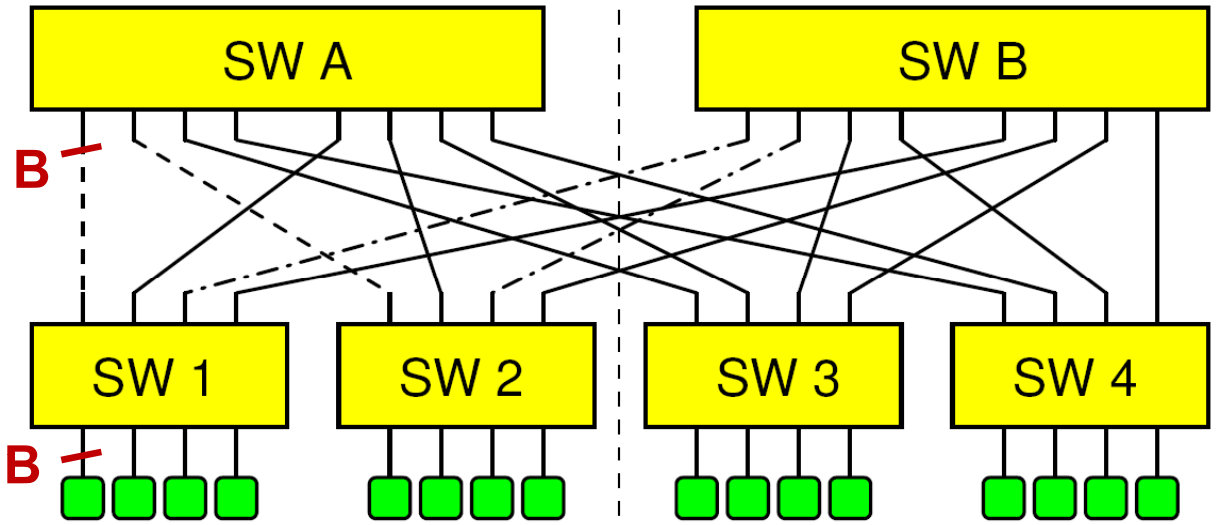
- A non-blocking crossbar can mediate a number of connections between a group of input and a group of output elements
- This can be used as a 4-port non-blocking switch (fold at the secondary diagonal)



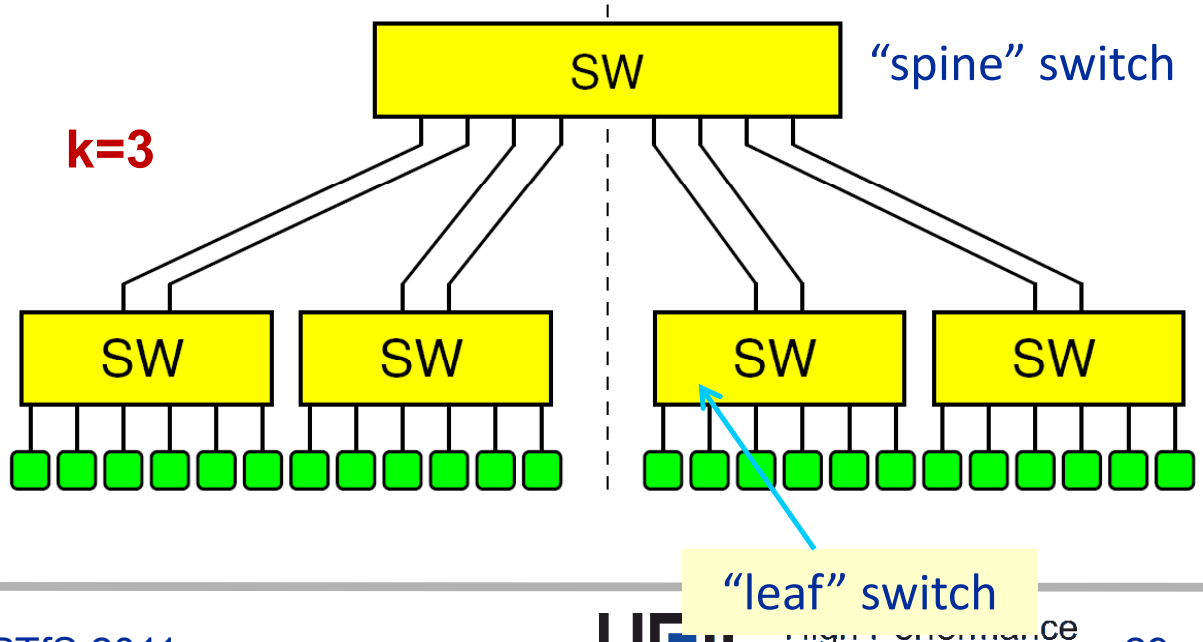
- Switches can be cascaded to form hierarchies (common case)
- Crossbars can also be used directly as interconnects in computer systems
  - Example: Scalable UMA memory access – see later
  - (Historic) example: Hitachi SR8000

"Fat tree" switch hierarchies

- **"Fully non-blocking"**
  - N/2 end-to-end connections with full bandwidth → $B_b = B * N/2$
  - **B_b/N = const. = B/2**
  - Sounds good, but see next slide

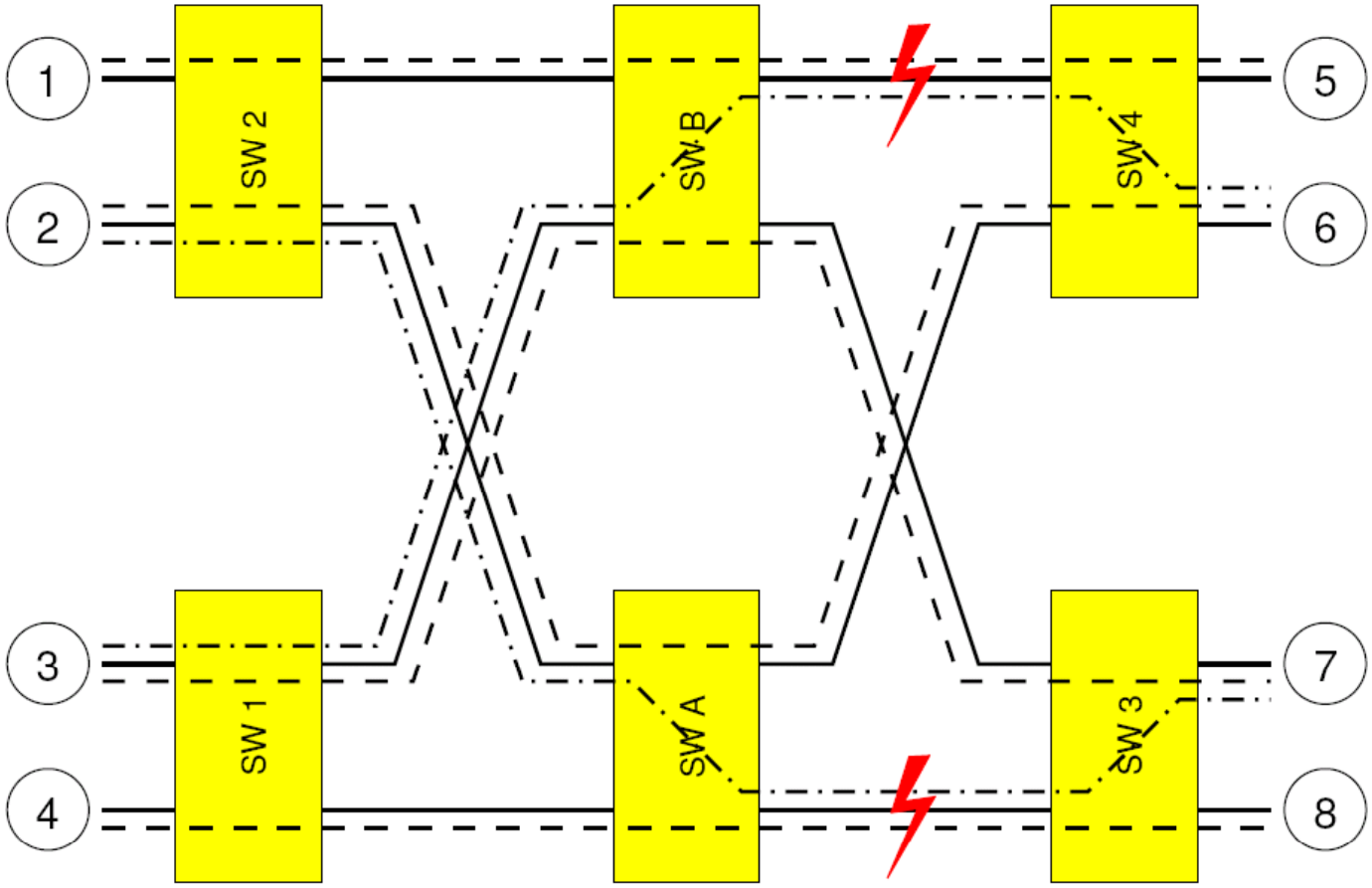


- **"Oversubscribed"**
  - Spine does not support N/2 full BW end-to-end connections
  - **B_b/N = const. = B/2k**, where k is the oversubscription factor
  - Intelligent resource management is crucial

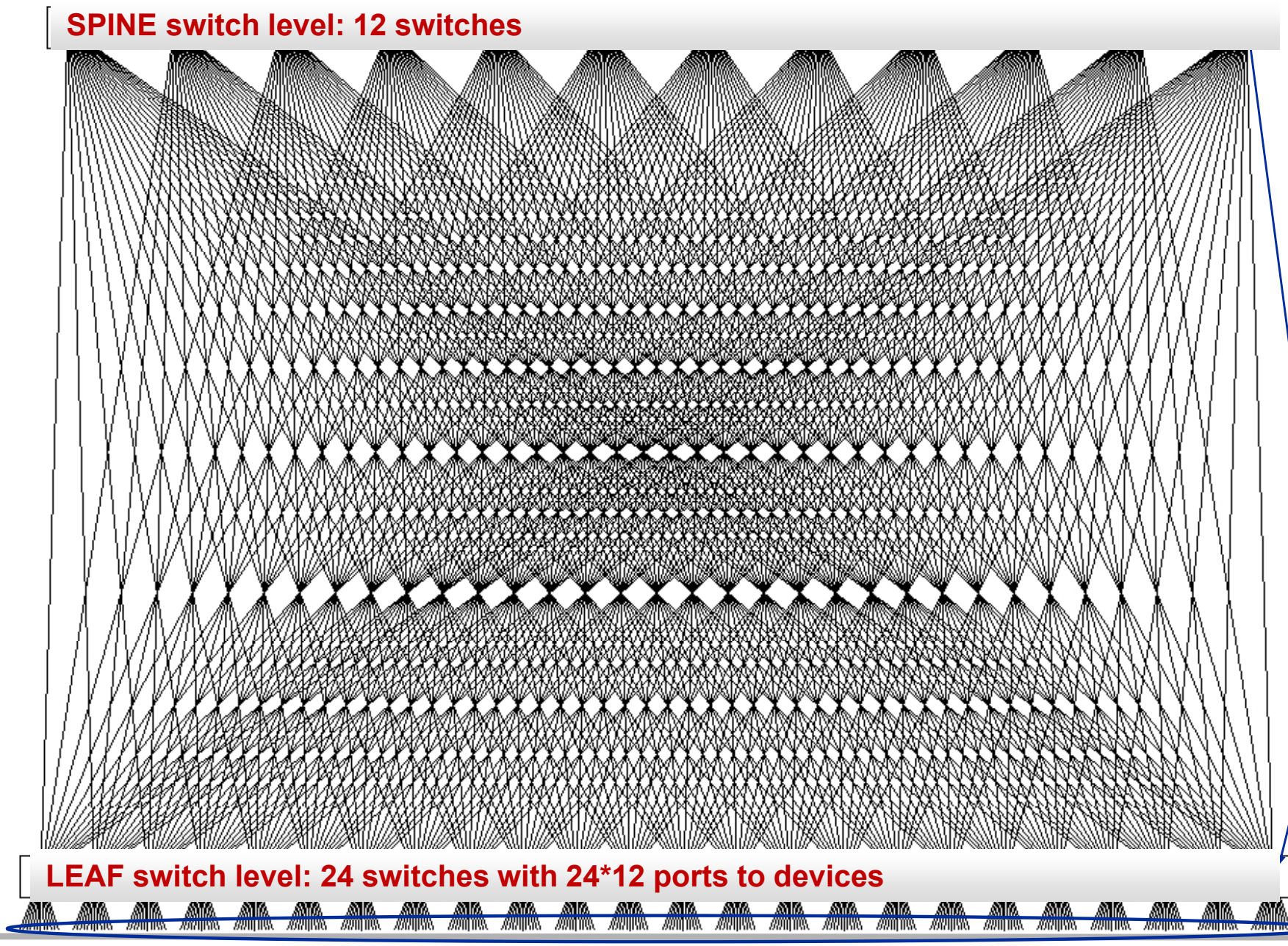


Fat trees and static routing

- **If all end-to-end data paths are preconfigured ("static routing"), not all possible combinations of N agents will get full bandwidth**
  - Example: – – – – is a collision-free pattern here
  - Change 2→6, 3→7 to 2→7, 3→6: – · – · – has collisions if no other connections are re-routed at the same time
- Static routing is still a quasi-standard in "commodity" inter-connects
- However, things are starting to improve slowly



Full fat-tree: „Single“ 288-port IB DDR-Switch



Basic building blocks:

24-port switches

Σ = 12+24 switches

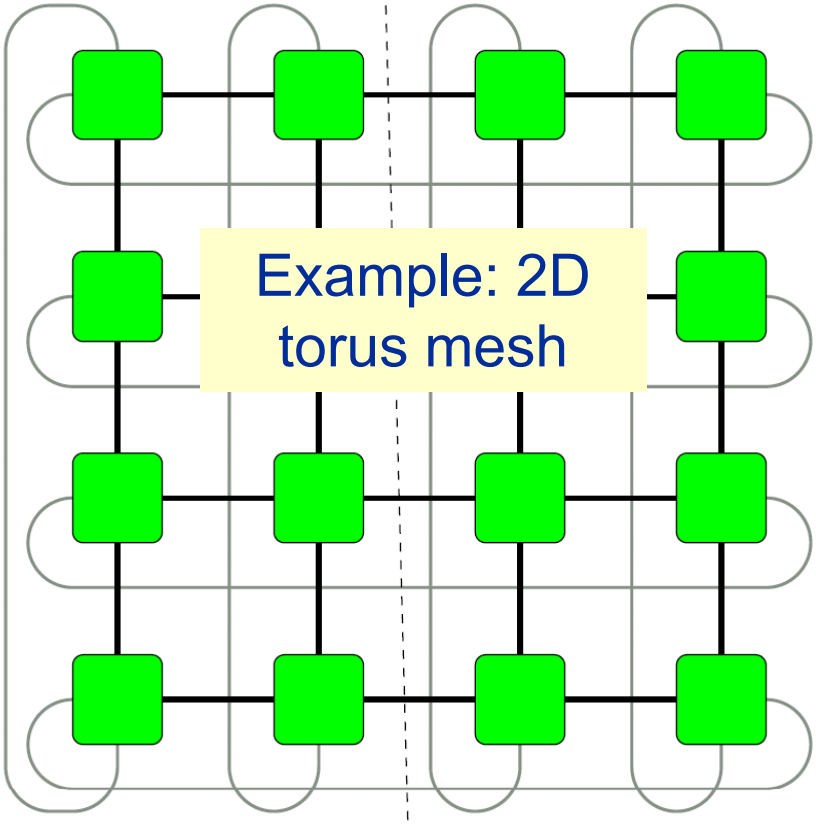
Fat tree networks – Examples

- **Ethernet**
  - 1 Gbit/s &10 Gbit/s variants; > 50% of all Top500 entries (November 2009)
- **InfiniBand**
  - Dominant high-performance “commodity” interconnect
    - SDR: 10 Gbit/s per link and direction (10 bits/byte)
    - DDR: 20 Gbit/s per link and direction
      (Building blocks: 24-port switches)
    - QDR: … you figure that out by yourself
      - QDR IB is used in the RRZE’s TinyBlue cluster
      - Building blocks: 36 port switches → “Large” 36*18=648-port switches
- **Myrinet**
  - Current version: 10 Gbit/s per link and direction
  - Interoperable with 10 Gbit/s Ethernet
  - Waning importance for HPC

- **Expensive and complex to scale continuously to very high node counts**

Meshes

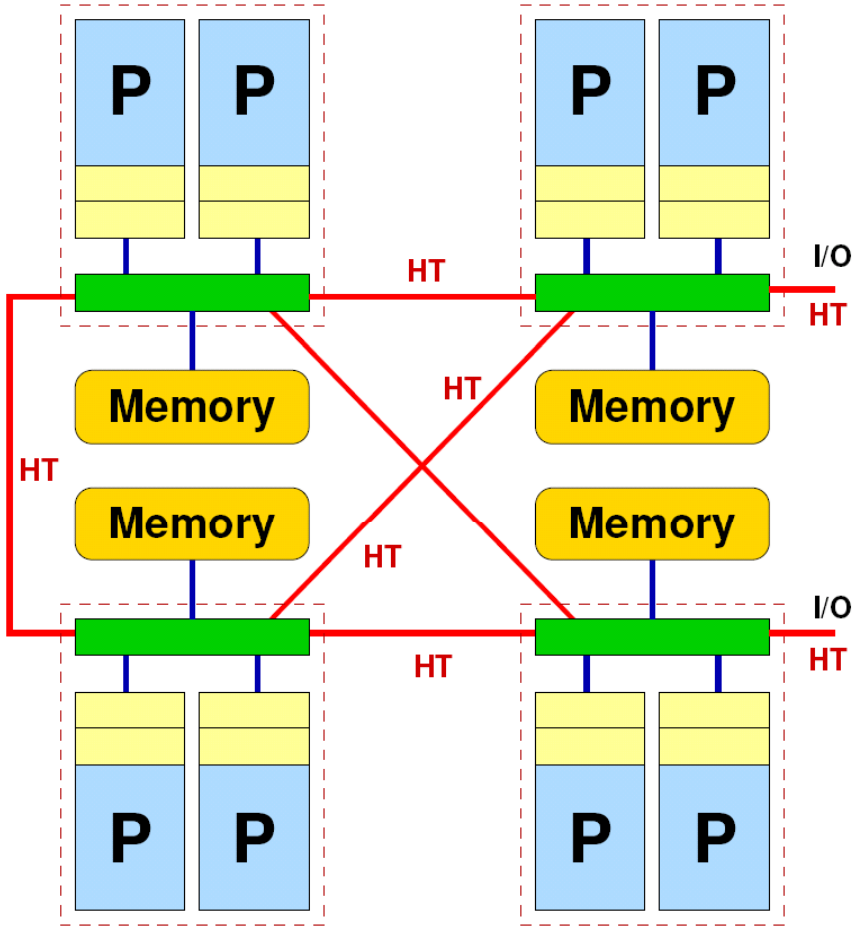
- **Fat trees can become prohibitively expensive in large systems**
- **Compromise: Meshes**
  - n-dimensional Hypercubes
  - Toruses (2D / 3D)
  - Many others (including hybrids)
- **Each node is a "router"**
- **Direct connections only between direct neighbors**
- **This is not a non-blocking corossbar!**
  - Intelligent resource management and routing algorithms are essential



- **Toruses are used in very large systems: Cray XT, IBM Blue Gene**
  - $B_b \sim N^{(d-1)/d}$ → $B_b/N \to 0$ for large N
  - Sounds bad, but those machines show good scaling for many codes
  - Well-defined and predictable bandwidth behavior!

Meshes

- **Advantages of toroidal/cubic meshes**
  - Limited cabling required
  - Cables can be kept short
- **Meshes can come in all shapes and sizes**
  - Example: 4-socket dual-core AMD Opteron node with HyperTransport fabric
  - This mesh is “asymmetric” since two sockets use one HT link each for I/O



4-socket 2xhexa-core AMD Magny-Cours:

3D torus