Parallel Programming
- Parallel Architectures -

Prof. Dr. Felix Wolf

Technical University of Darmstadt

Outline

- Classification
- Memory architecture
- Interconnection networks
- Example: Lichtenberg Cluster
- Cache coherence
- Memory consistency
- Synchronization

Taxonomies

- Number of instruction streams vs. number of data streams
- Memory architecture
- Network architecture
- Degree of heterogeneity
- Degree of customization

Flynn’s classification [1966]

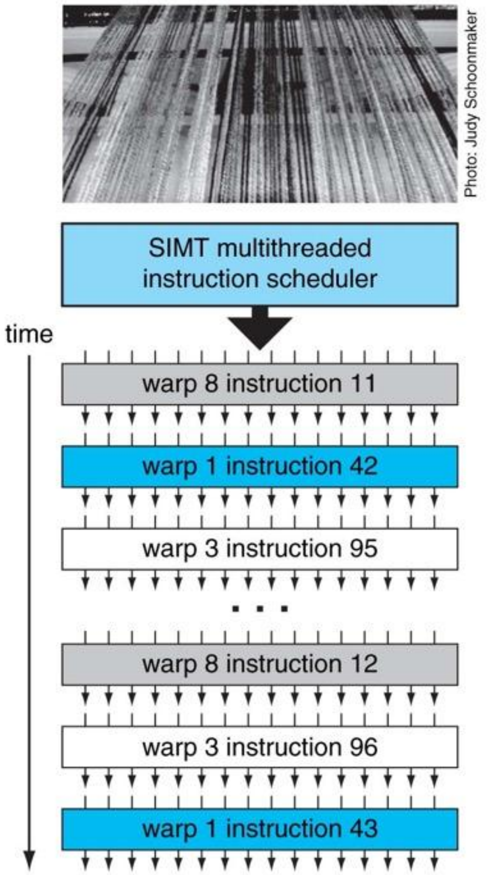
		# Instruction streams	
		Single	Multiple
# Data streams	Single	SISD ▪ Classical uniprocessor	MISD ▪ No commercial multiprocessor of this type ever built
	Multiple	SIMD ▪ Same instruction is executed by multiple processors using different data streams ▪ Data parallelism ▪ Examples: SIMD extensions for multimedia, vector processors	MIMD ▪ Each processor fetches its own instructions and operates on its own data ▪ Thread-level parallelism

MIMD

- Architecture of choice for general-purpose multiprocessors
- Offers high degree of flexibility
  - High performance for one application or multi-programmed multiprocessor
- Can take advantage of off-the-shelf processors
- Popular execution model – Single Program Multiple Data (SPMD)
  - The same program is executed in parallel with each instance having a potentially different control flow

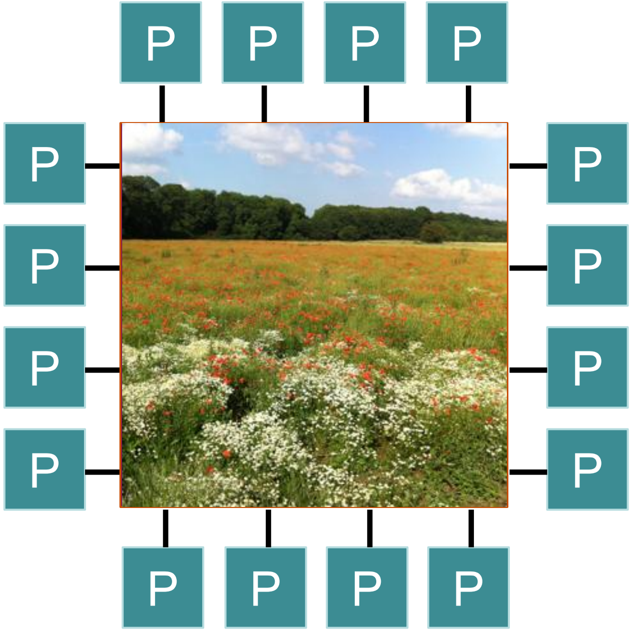
Single-instruction multiple threads (SIMT) - Used on GPUs -

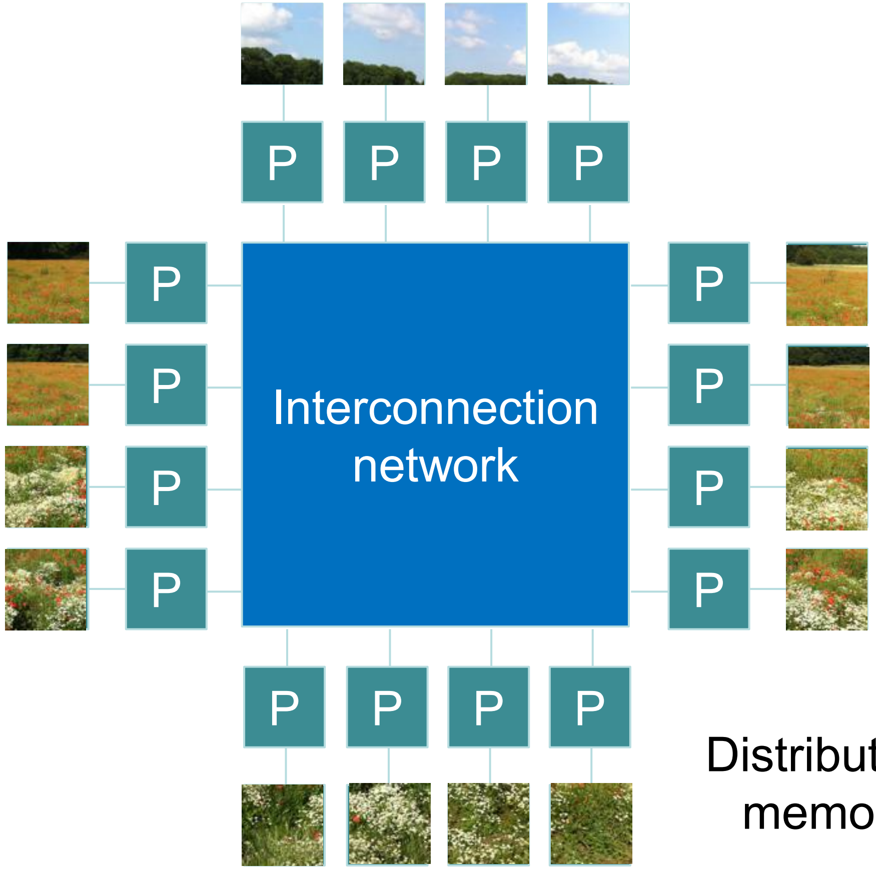
- Creates, manages, schedules, and executes threads in groups of parallel threads called **warps**
- At each instruction issue time, SIMT instruction unit
  - Selects warp that is ready to execute its next instruction
  - Broadcasts instruction to all active threads of that warp
- Individual threads may be inactive to allow for independent branching



Source: Patterson, Hennessy: Computer Organization & Design, 4th edition, Morgan Kaufmann

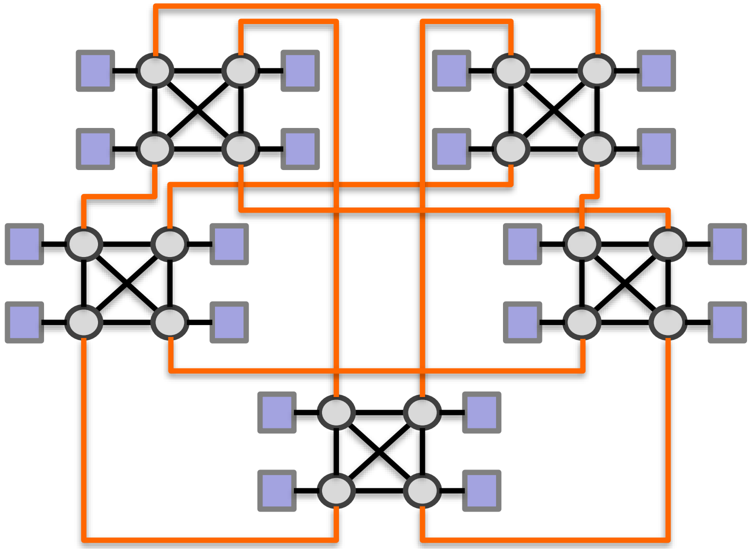
Memory architecture



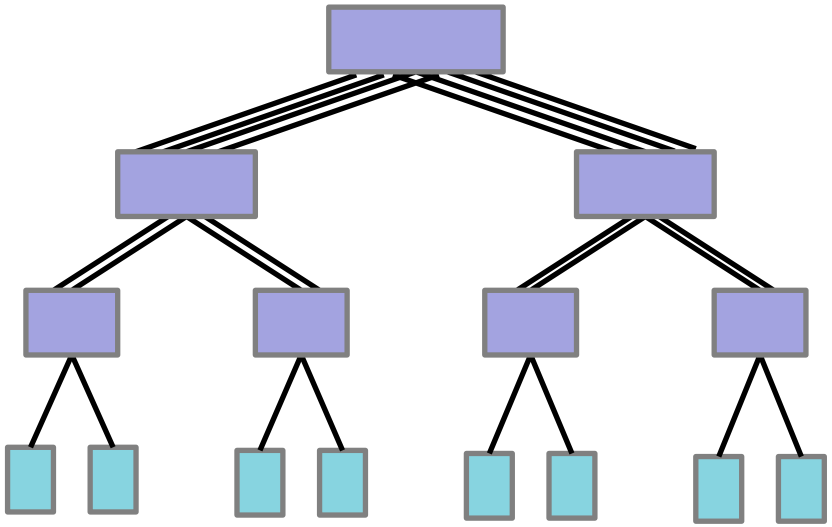


Distributed memory

Popular network architectures for distributed memory systems



Dragonfly
(distributed switched network)

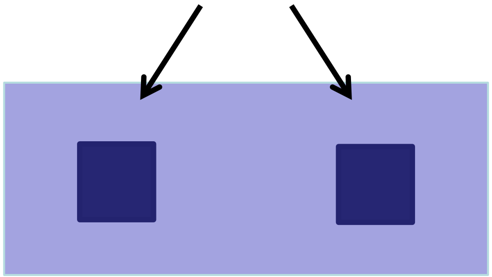


Fat tree
(centralized switched network)

Degree of heterogeneity

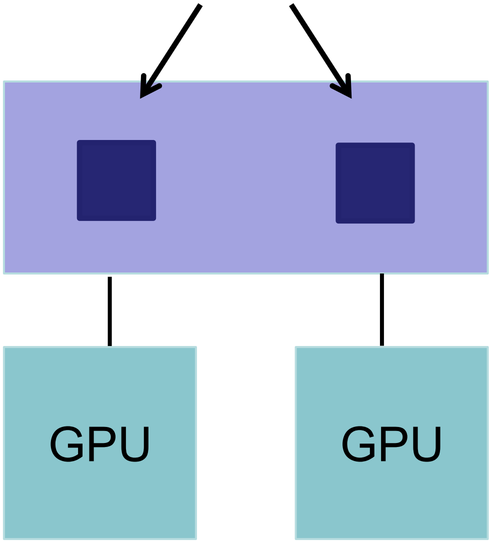
Homogeneous node architecture

Classic server CPUs
(e.g. Intel Xeon)



Heterogeneous node architecture

Classic server CPUs
(e.g. Intel Xeon)



Accelerators

Hardware accelerators

GPU

Data parallelism

Large pool of threads

FPGA

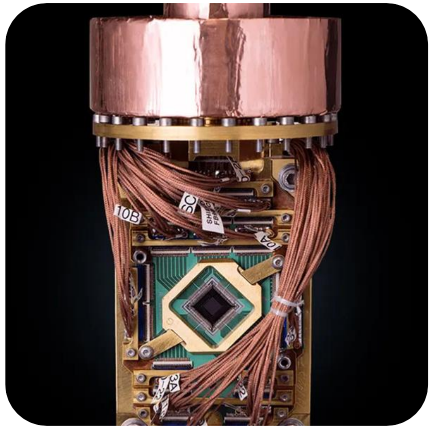
Problem specific

Reconfigurable
datapaths

Neuromorphic

Asynchronous CMOS circuits

Integrates compute
(neurons) and mem./comm.
(synapses)



Quantum

Suitable for exponential-time algorithms

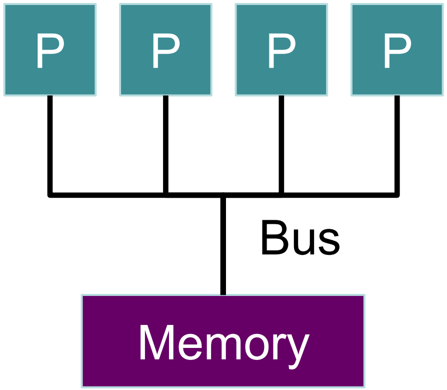
Degree of customization

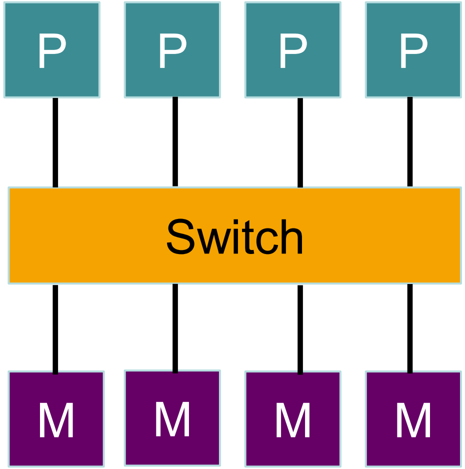
- **Commodity clusters** – standard nodes and standard network
  - Focus on applications with average performance requirements
  - Example: Beowulf cluster
- **Custom clusters** – custom nodes and custom network
  - Focus on applications that exploit large amounts of parallelism on a single problem
  - Example: Fugaku (RIKEN, Japan)
- Above classes are extremes of a broad spectrum

Shared memory

UMA (Uniform memory access)

- Each CPU has same access time to each memory address
- Simple design but limited scalability (typically less than a socket)

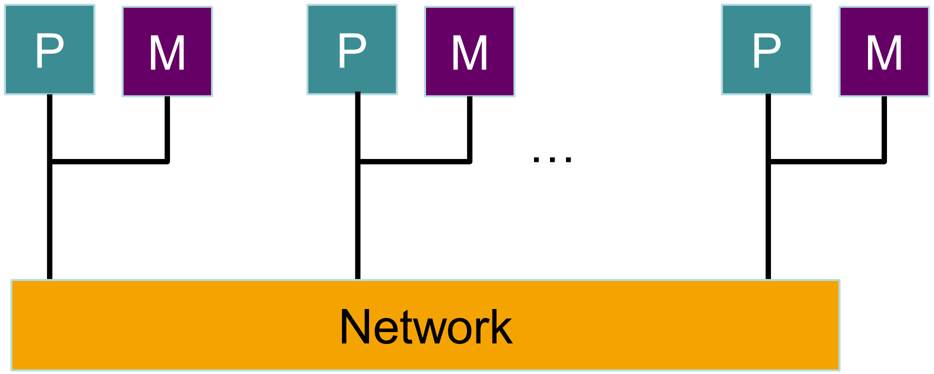




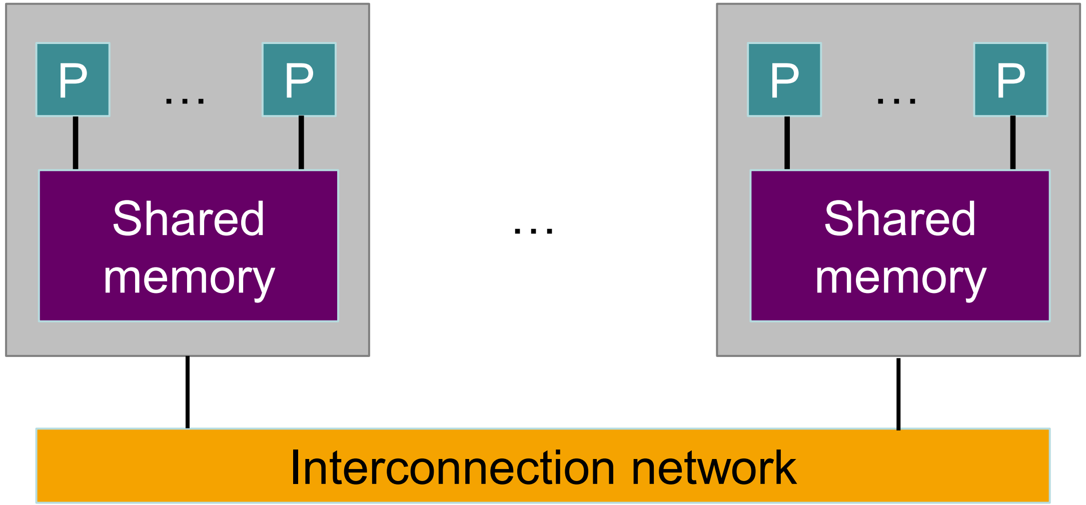
Shared memory (2)

NUMA (Non-uniform memory access)

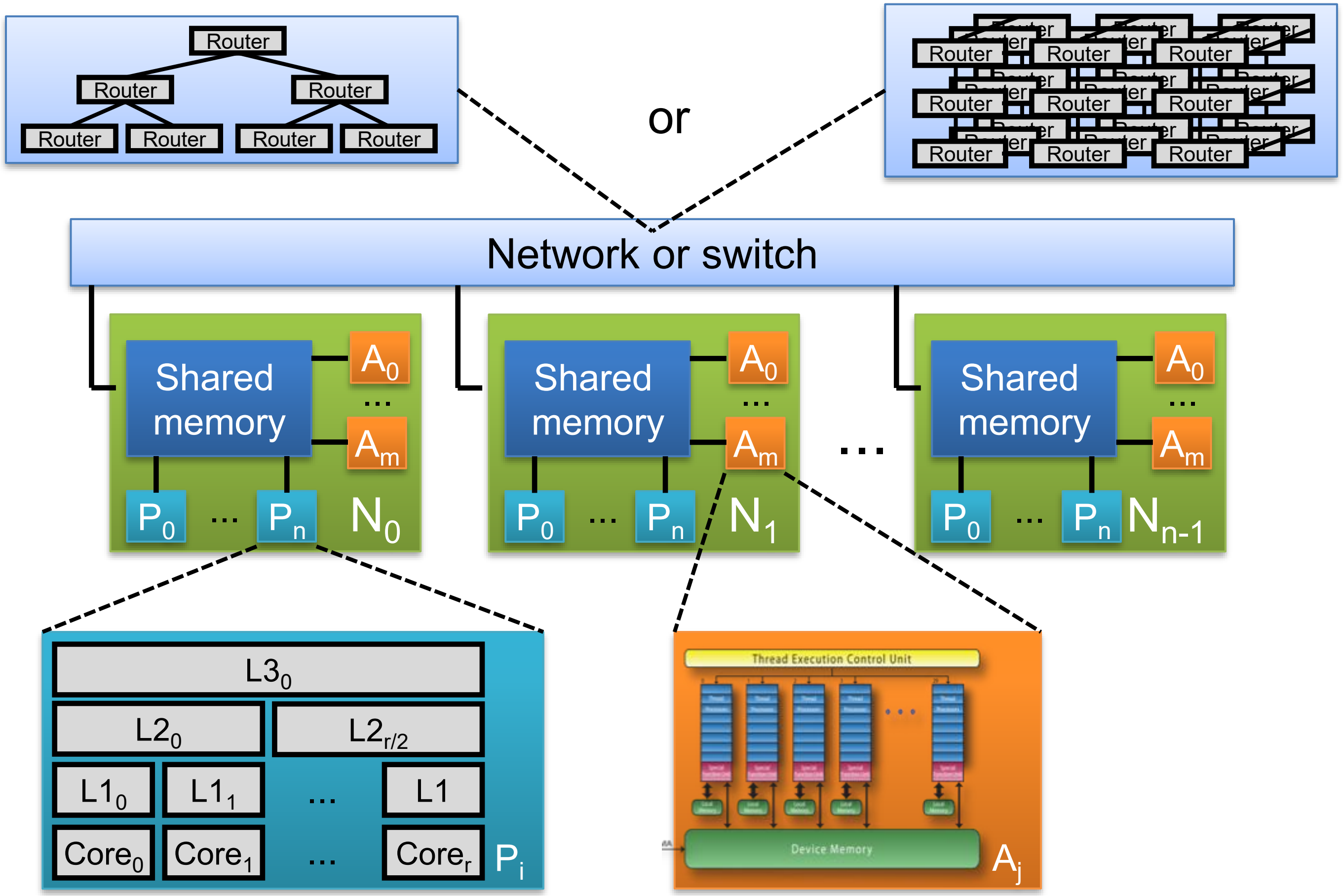
- Memory has affinity to a processor
- Access to local memory faster than to remote memory
- Harder to program but more scalable



Distributed memory (aka multicomputer)



Typical cluster architecture



Interconnection network

- Physical link between components of a parallel system
  - Between processors and memory
  - Between nodes
- Communication via exchange of messages
  - Example: intermediate results, memory requests

Design elements

- **Topology** – determines geometric layout of links and switches
- **Routing technique** – determines paths of messages through network

Network performance

Bandwidth

- Maximum rate at which information can be transferred
- Aggregate bandwidth – total data bandwidth supplied by network
- Effective bandwidth or throughput – fraction of aggregate bandwidth delivered to an application

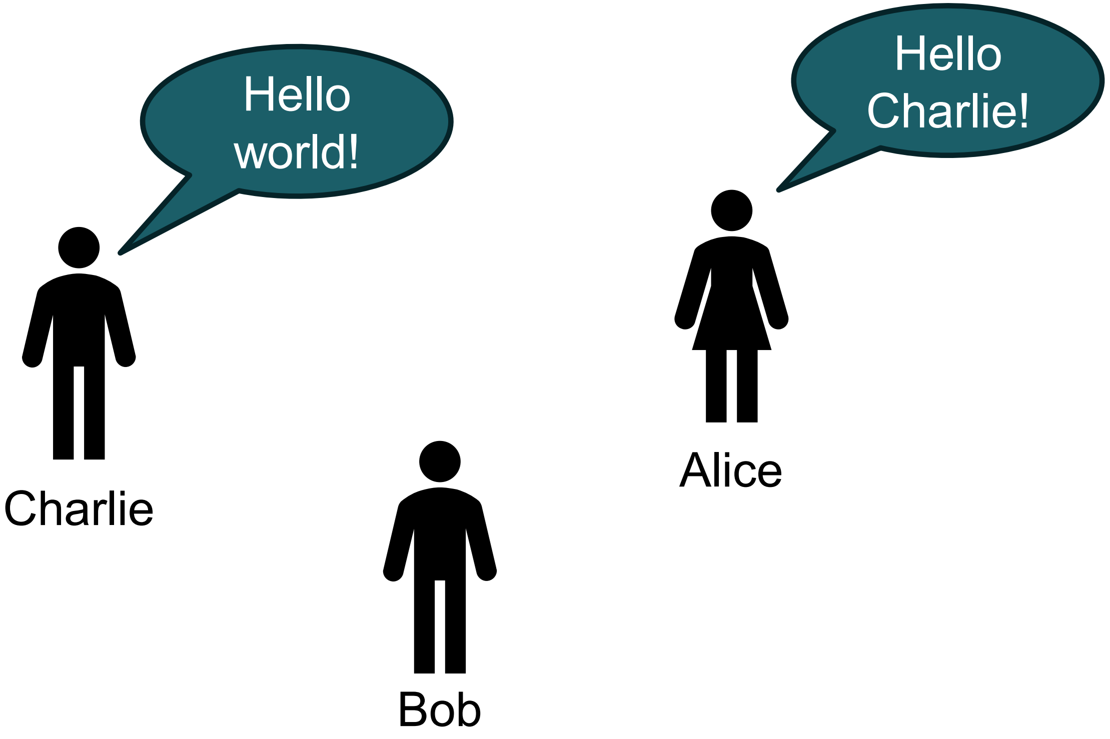
Latency

Time for the first bit of the packet to arrive

- Sending overhead + time of flight + $\dfrac{\text{Packet size}}{\text{Bandwidth}}$ + receiving overhead

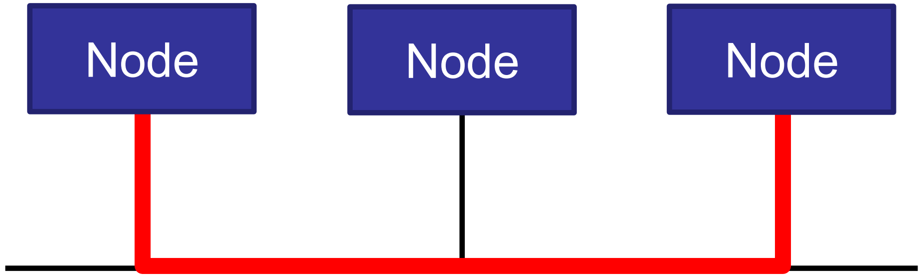
Shared-media networks

- Only one message at a time – processors broadcast their message over medium
- Each processor “listens” to every message and receives the ones for which it is the destination



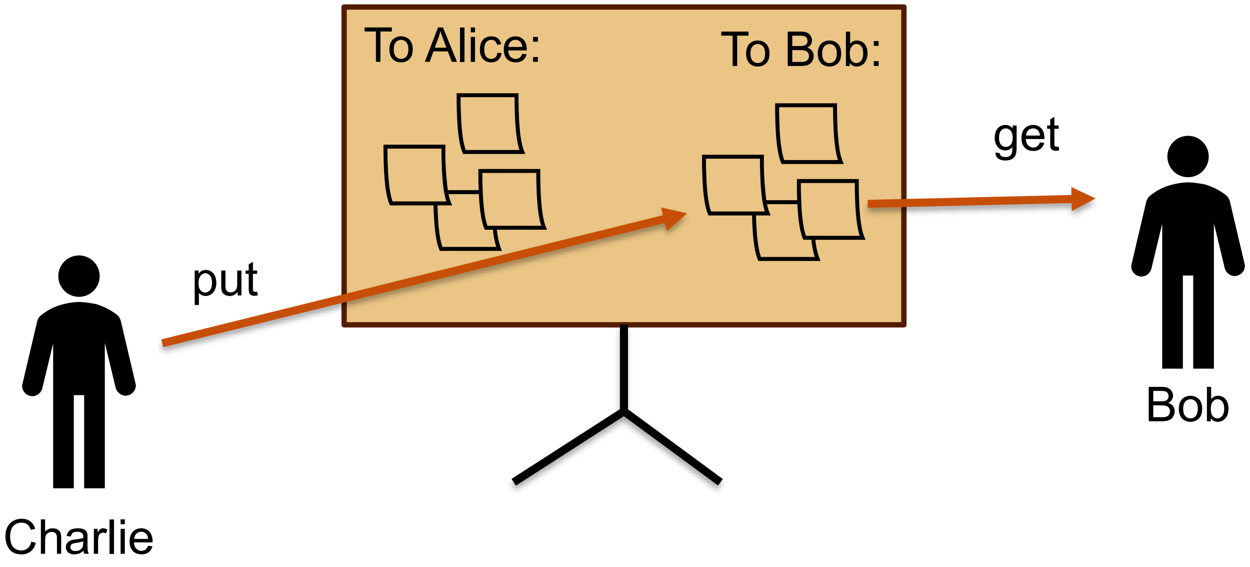
Shared-media networks

- Only one message at a time – processors broadcast their message over medium
- Each processor “listens” to every message and receives the ones for which it is the destination
- Decentralized arbitration
  - Before sending a message, processors listen until medium is free
  - Message collision can degrade performance
- Low cost but not scalable
- Example – bus networks to connect processors to memory



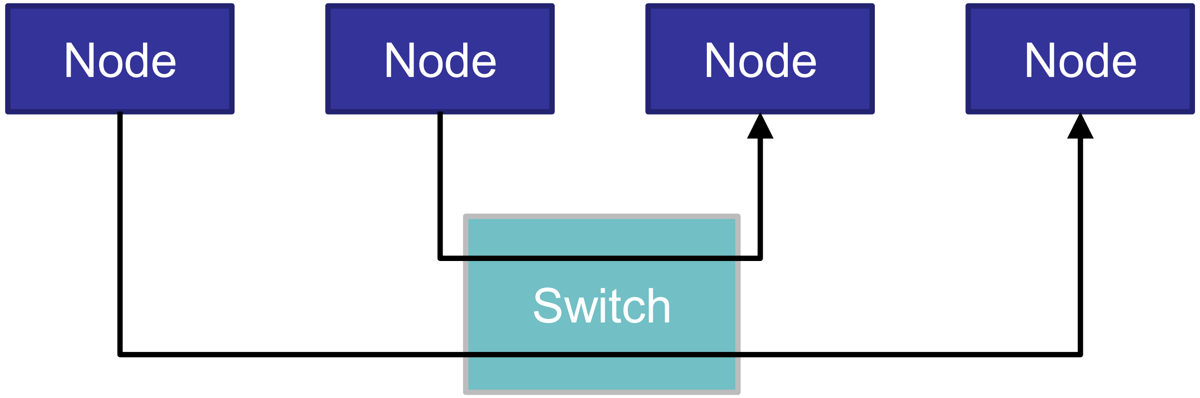
Switched-media networks

- Support point-to-point messages between nodes
- Each node has its own communication path to the switch



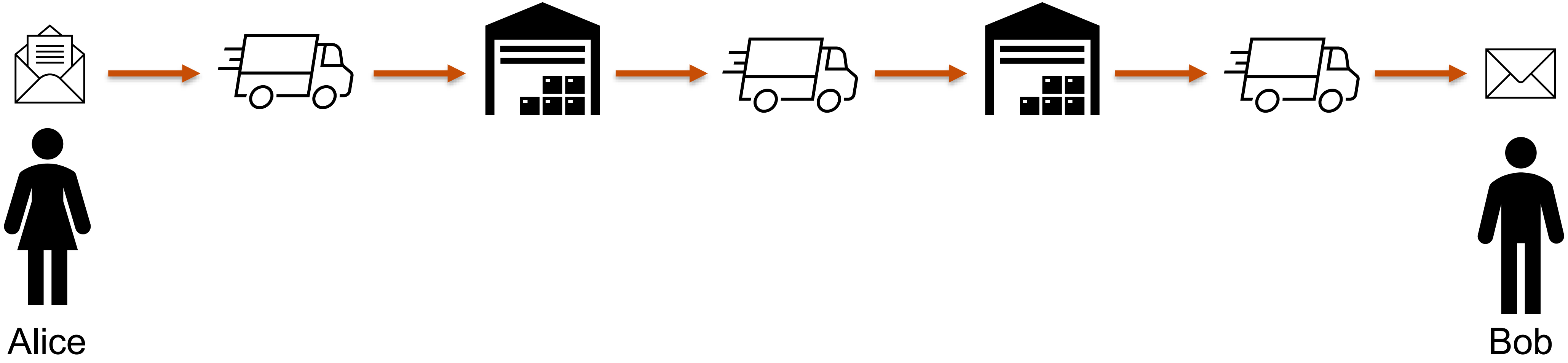
Switched-media networks

- Support point-to-point messages between nodes
- Each node has its own communication path to the switch
- Advantages
  - Support concurrent transmission of multiple messages among different node pairs
  - Scale to very large numbers of nodes



Centralized switched networks

- Also called *indirect* or dynamic interconnection networks
- Connect processors / memory indirectly using several links and intermediate switches



Centralized switched networks

- Also called *indirect* or dynamic interconnection networks
- Connect processors / memory indirectly using several links and intermediate switches
- Examples: switching networks
- Used both for shared- and distributed-memory architectures

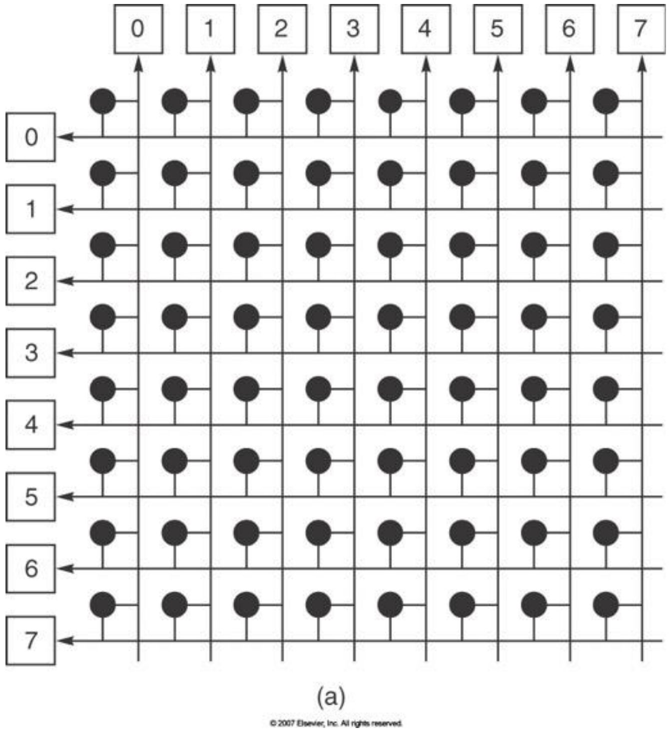
Crossbar switch

Non-blocking

- Links are not shared among paths to unique destinations

Requires N^2 crosspoint switches

- Limited scalability

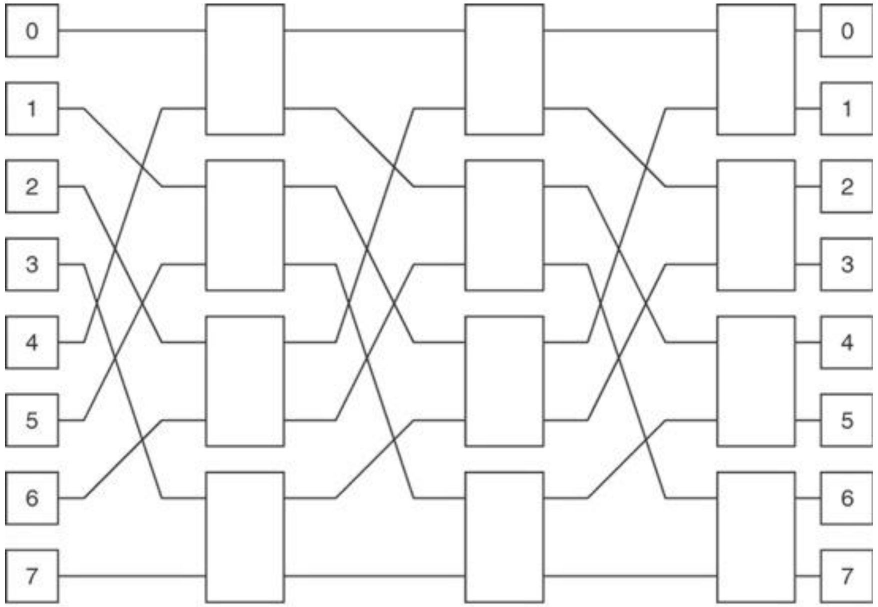


Source: Hennessy, Patterson: Computer Architecture, 4th edition, Morgan Kaufmann

Multistage interconnection network (MIN)

Example: Omega network

- Complexity O(N log N)
- Perfect shuffle permutation at each stage
- Blocking due to paths between different sources and destinations simultaneously sharing network links
- Omega with k x k switches
  - $\log_k N$ stages ; $N/k \log_k N$ switches

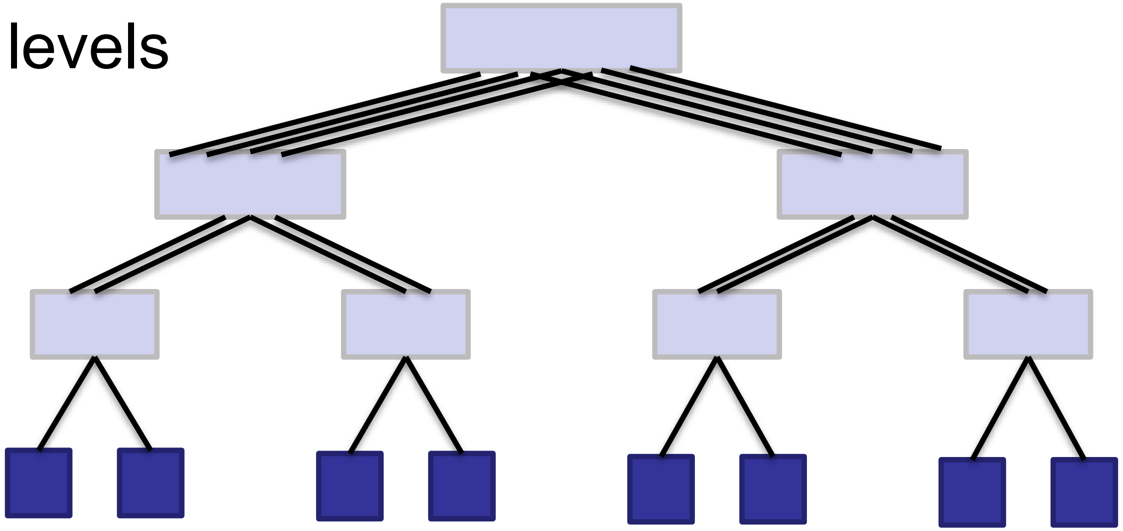


Source: Hennessy, Patterson: Computer Architecture, 4th edition, Morgan Kaufmann

MINs can be extended to **rearrangeably** non-blocking topologies

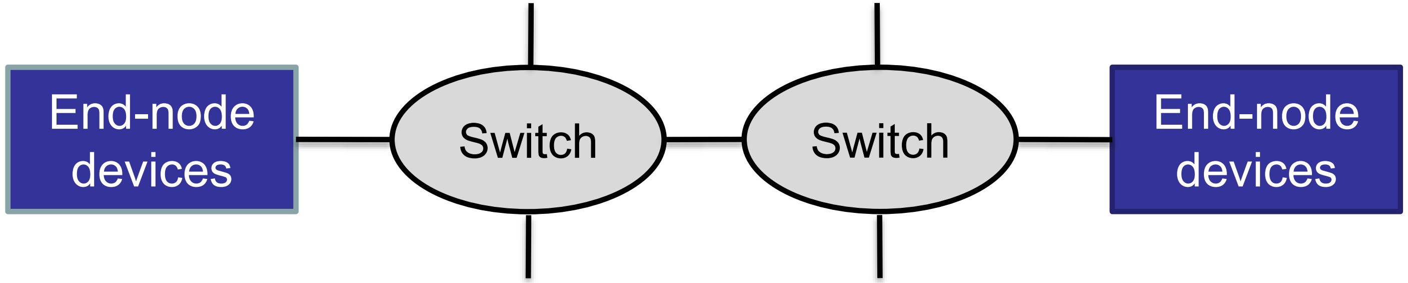
Fat tree

- Balanced tree where
  - Leaves = end node devices
  - Vertices = switches
- Total link bandwidth constant across all levels
- Switches often composed of multiple smaller switches
- Popular topology for cluster interconnects

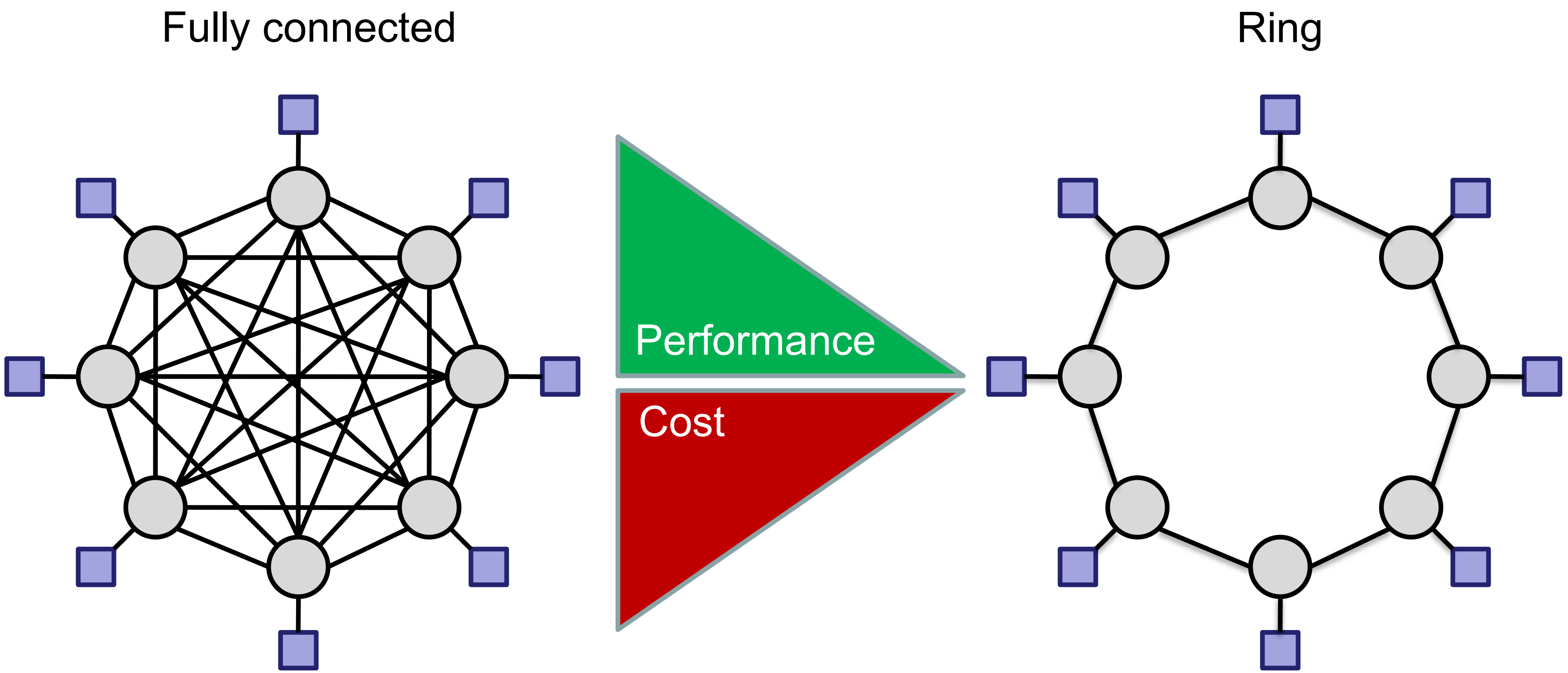


Distributed switched networks

- Each network switch has one or more end node devices directly attached to it
- End node devices = processor(s) + memory
  - Directly connected to other nodes without going through external switches
  - Mostly used for distributed-memory architectures
- Also called *direct* or static interconnection networks
- Ratio of switches to nodes = 1:1

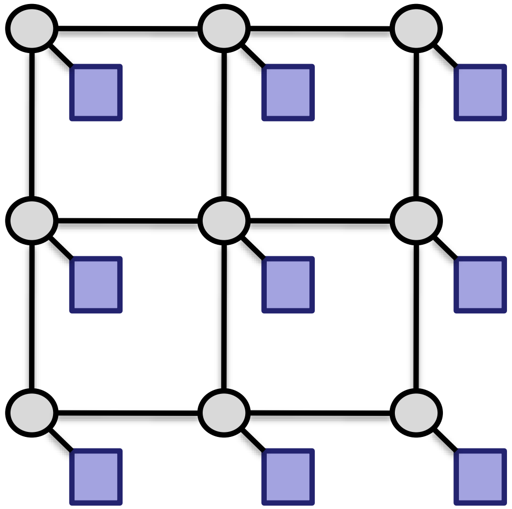


Network topologies



N-dimensional meshes

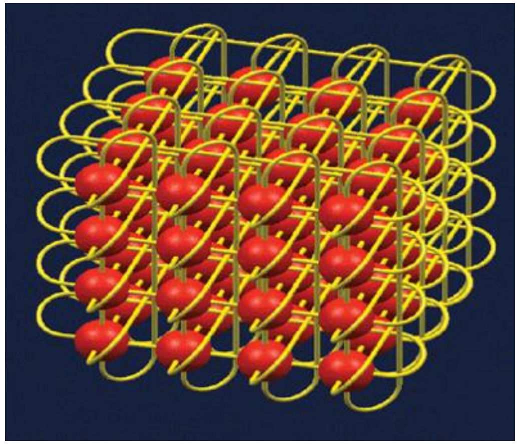
- Direct link to neighbors
- Each node has 1 or 2 neighbors per dimension
  - 2 in the center
  - Less for border or corner nodes
- Efficient nearest neighbor communication
- Suitable for large numbers of nodes



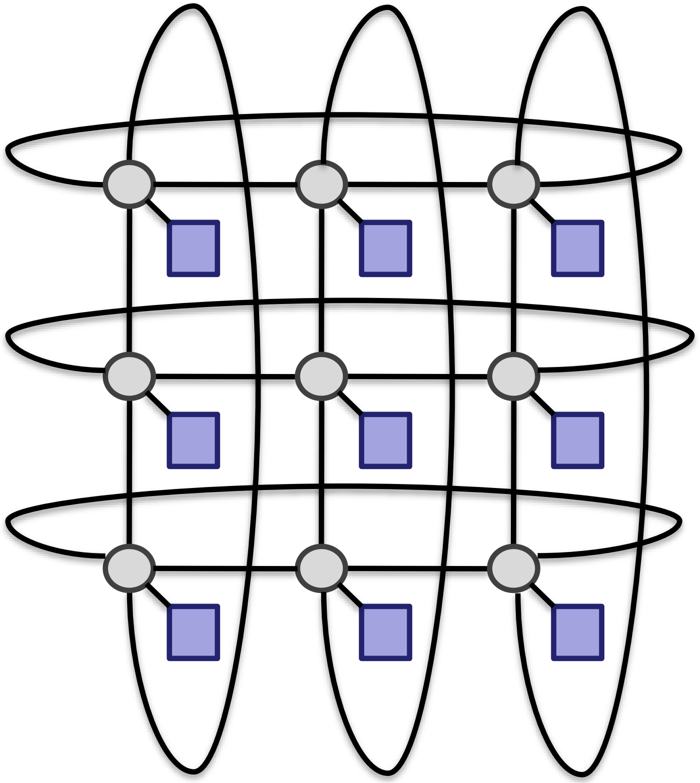
2D mesh

Torus

- Mesh with wrap-around connections
- Each node has exactly 2 neighbors per dimension
- Typically 3-6 dimensions

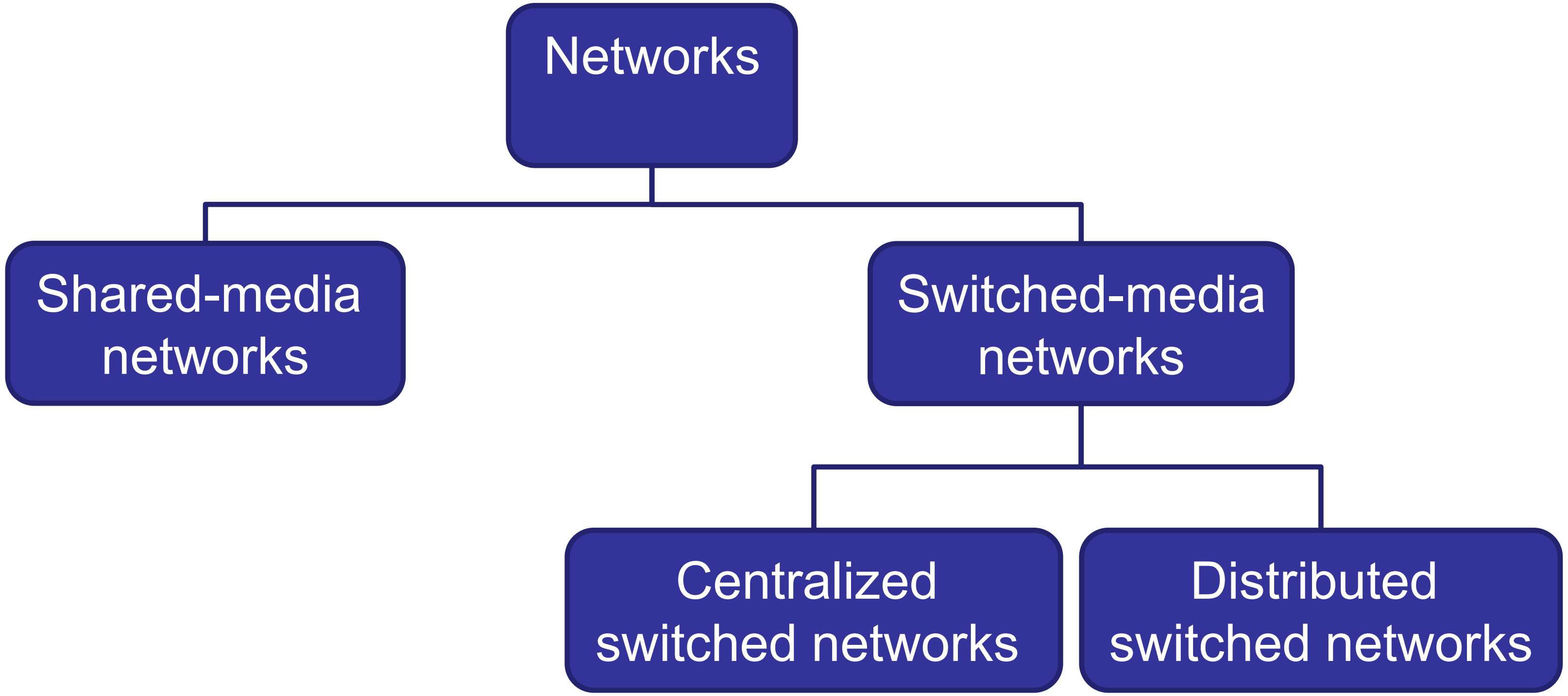


3D torus



2D torus

Network classes



Networks

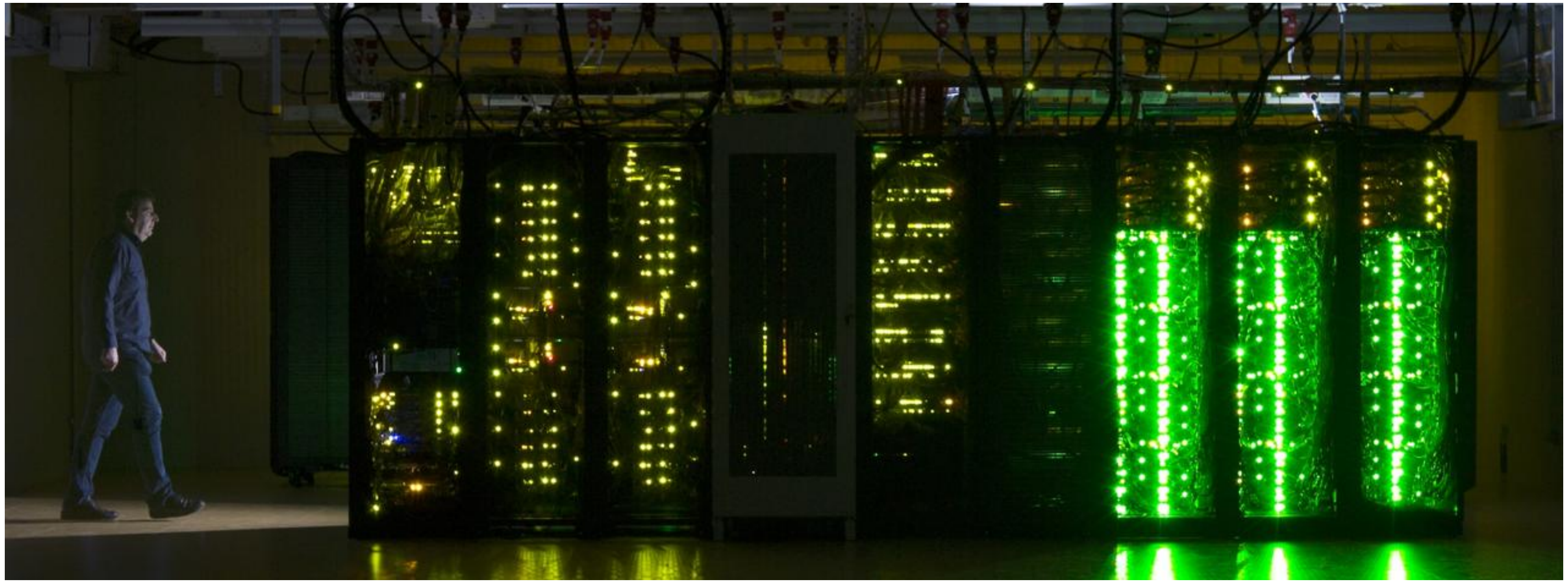
Shared-media networks

Switched-media networks

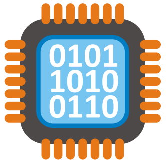
Centralized switched networks

Distributed switched networks

LICHTENBERG Cluster @ TU Darmstadt

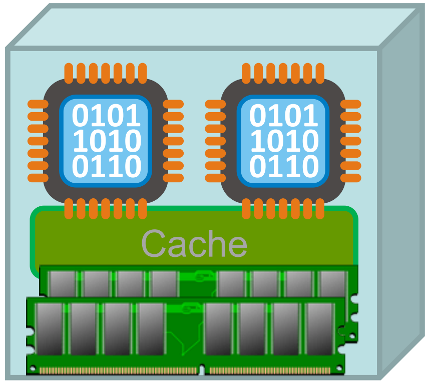


Lichtenberg – A Parallel, Yet *Hierarchical* System



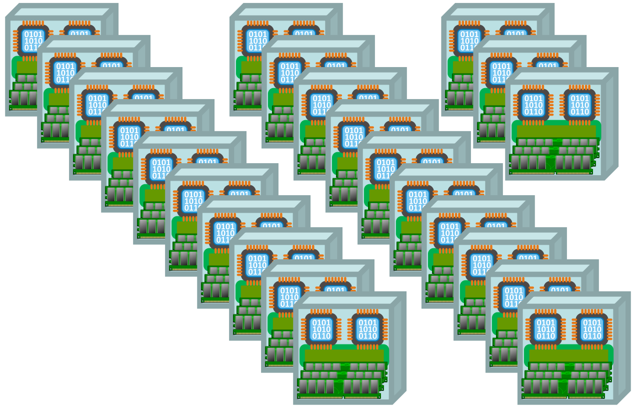
Processor = CPU (known from your desktop PC)

- Multiple cores per CPU (in our nodes typically: 48 to 52)
- Shared memory (all cores access the very *same* memory chips)



Compute node (similar to a better workstation or server)

- Server with multiple processors (typical node: 2)
- Very fast connection between processors and memory
- Shared memory between processors



Cluster

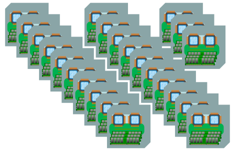
- All compute nodes as a battalion, commanded by a sophisticated scheduler
- Very fast interconnect between the nodes for high-throughput and low-latency communication
- *No* shared memory between nodes

LICHTENBERG II

- Approx. ***tripling*** the computing capabilities of Lichtenberg
  - Peak real performance: ≥ **2x3 PFLOP/s**
- Nodes based on Intel® **D**ata **C**enter **B**locks, using Cascade Lake-AP and Sapphire Rapids processors (96-104 cores @ 2.3-2.1 GHz)
- **MPI section**: 630+576 nodes with 384-512 GBytes RAM each
- **ACC section**: 19 nodes with Nvidia Tesla - V100, - A100, - H100, Intel PVC and AMD MI300X
- **MEM section**: 5 nodes with 1.5-6,0 TByte RAM each
- **Interconnect**: Infiniband HDR100
- **Cooling**: hot water (≥ 40°C), reusing most of the waste heat for heating other buildings

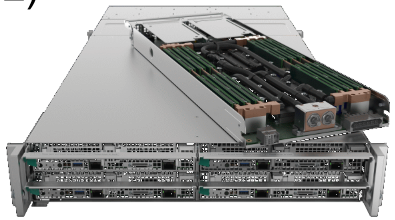
Compute nodes

MPI section



630 nodes à

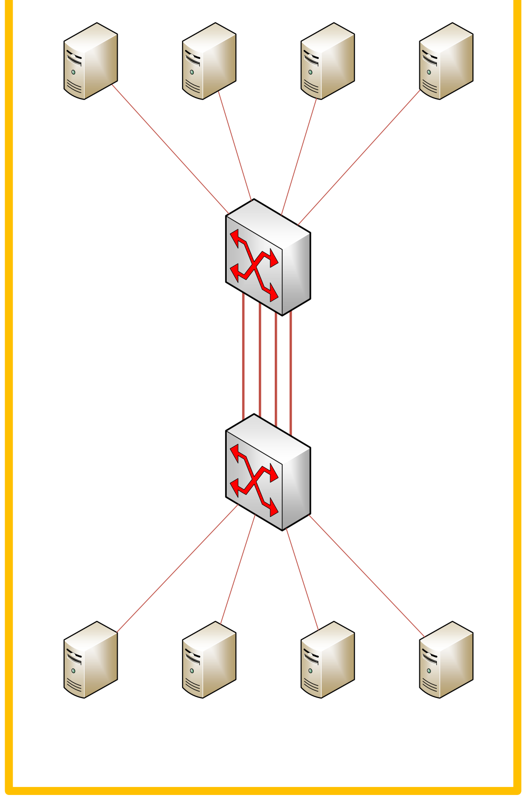
- **96 cores** (2x Intel Cascade-Lake AP @2.3 GHz)
- Intel® Virtualization Technology (VT-x)
- Intel® TSX-NI
- 2x Intel® AVX-512
- VNNI (for DL/ML Inference)
- ≥ 4 NUMA domains
- **384 GBytes RAM**



576 nodes à

- **104 cores** (2x Intel Sapphire Rapids @2.1 GHz)
- Intel® Virtualization Technology (VT-x)
- Intel® TSX-NI
- 2x Intel® AVX-512
- VNNI (for DL/ML Inference)
- ≥ 4 NUMA domains
- **512 GBytes RAM**

Lichtenberg 2



- Details on https://www.hrz.tu-darmstadt.de/hlr → Operations → Hardware

Big-Mem nodes

MEM section

2 nodes à

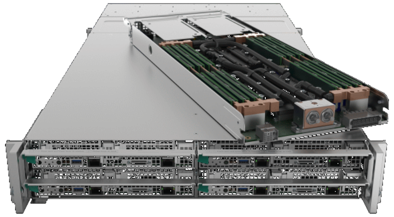
- 96 cores (2x Intel Cascade-Lake AP @2.3 GHz)
- ≥ 4 NUMA domains
- **1536 GBytes RAM**

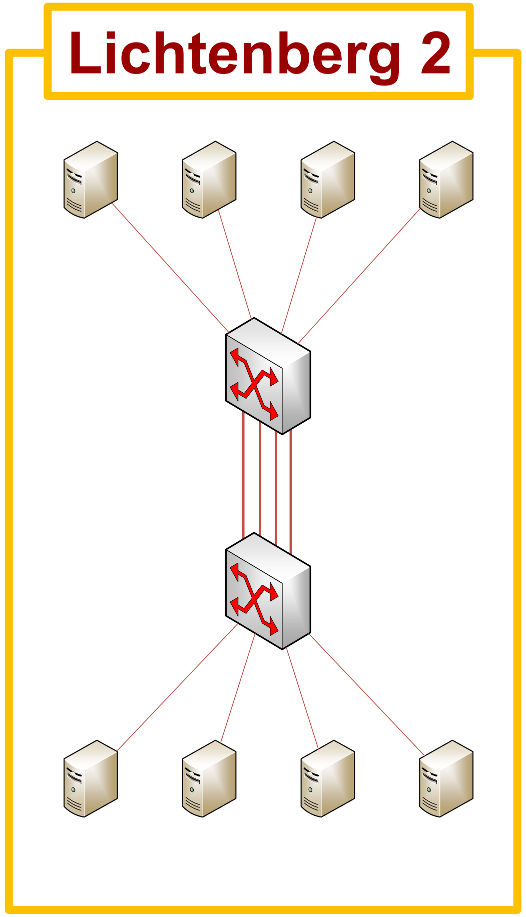
2 nodes à

- 104 cores (2x Intel Sapphire Rapids @2.1 GHz)
- ≥ 4 NUMA domains
- **2048 GBytes RAM**

1 node à

- 192 cores (4x Intel Sapphire Rapids @2.1 GHz)
- ≥ 8 NUMA domains
- **6144 GBytes RAM**



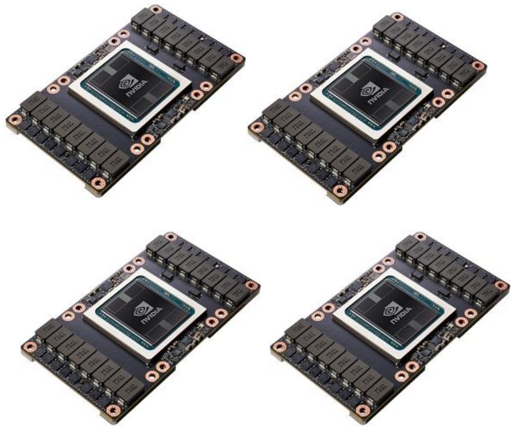


Accelerator nodes I

4 nodes à • **96 cores (Intel Cascade-Lake)** • **384 GBytes RAM, 2.933 GHz memory clock**	**7 nodes à:** **96/128 cores (Intel/AMD)** **1536 GBytes RAM, 4.800 GHz memory clock**	**2 nodes à:** • **128 cores (AMD Epyc)** • **1536 GBytes RAM, 4.800 GHz memory clock**
4 Nvidia Tesla V100 (GV100 chip) • 32 GB CoWoS-HBM2 ECC RAM, 900 GB/s memory bandwidth • 5,120 CUDA cores • 640 Tensor cores • Single Precision Performance: 15.7 TFLOP/s • Double Precision Performance: 7.8 TFLOP/s • Tensor Performance: 125 TFLOP/s	**4-8 Nvidia Tesla A100** (GA100 chip) • 40 GB CoWoS-HBM2 ECC RAM, 1550 GB/s memory bandwidth • 8,192 CUDA cores • 432 TensorFlow32 cores • Single Precision Performance: 19.5 TFLOP/s • Double Precision Performance: 9.7 TFLOP/s • Tensor Performance: 312 TFLOP/s	**4 Nvidia Tesla H100** • 80 GB CoWoS-HBM3 ECC RAM, 16,896 CUDA cores • 528 TensorFlow cores • Single Precision Performance: 51 TFLOP/s • Double Precision Performance: 9.7 TFLOP/s • Tensor Performance: >=756 TFLOP/s

Accelerator nodes II

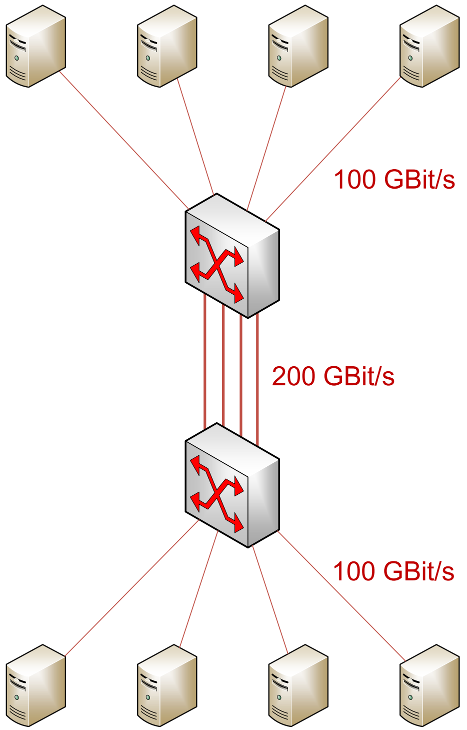
<table>
<tr><td colspan="2"><b><i>5+1</i> nodes à</b>
• <b>104/128 cores (Intel/AMD)</b>
• <b>1024/2304 GBytes RAM, 4.800 GHz memory clock</b></td></tr>
<tr><td>5x <i>PVC</i> nodes à</td><td>1x <i>MI300X</i> nodes à</td></tr>
<tr><td><b>4 Intel Ponte-Vecchio</b>
• 900 – 1600 MHz Taktfrequenz
• 128 Xe-Cores
• 1024 Xe Vector Engines
• 128 Ray Tracing Units
• 1024 Intel® Xe Matrix Extensions (Intel® XMX) Engines
• 128 GByte HBM2e (High Bandwidth Memory): 3276.8 GB/s</td><td><b>8 AMD MI300X</b>
• 2100 MHz GPU clock
• 1216 Matrix cores
• 19456 Stream Prozessoren
• 81.7 TFlop/s | 163.4 TFlop/s Performance (Double Precision, FP64 | Matrix FP64)
• 163.4 TFlop/s | 653.7TFlop/s Performance (Single Precision, FP32 | Matrix FP32)
• 192 GB HBM3 RAM</td></tr>
</table>



The new interconnect of LICHTENBERG II

Infiniband for multi-node calculations and for storage access

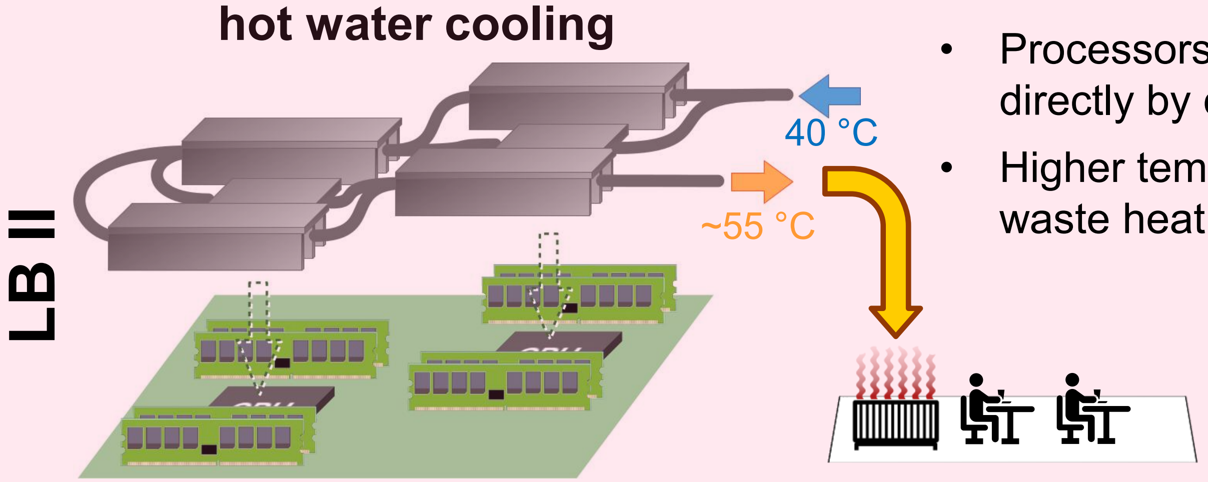
- **HDR** at 200 GBit/s switch ↔ switch
- **HDR100** at 100 GBit/s switch ↔ node due to PCIe Gen3/4/5
- Fully **non-blocking** "fat tree" topology (1:1) for HDR100
  ≙ every node can do MPI at full bandwidth with any other, unaffected by all other nodes' communication load



Ethernet

- 2x 1 GBit/s per node "base" network (management / booting / monitoring)

Infrastructure
Hot-water cooling



- Processors and memory modules cooled directly by cooling fluid
- Higher temperatures allow for reuse of waste heat

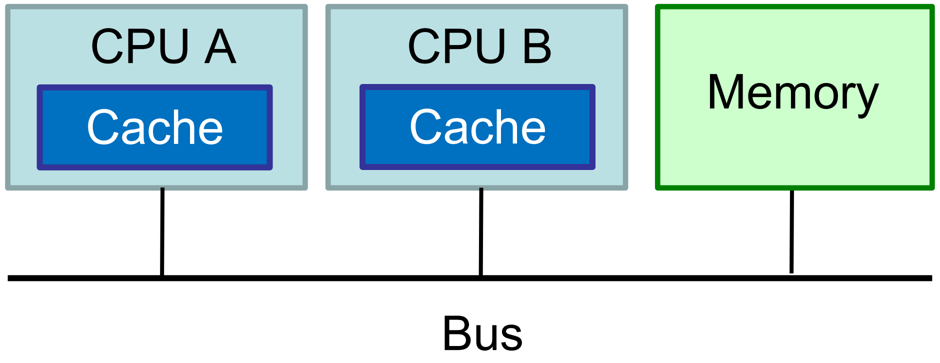
Temperatures	
inlet	outlet
40 °C	52–55 °C

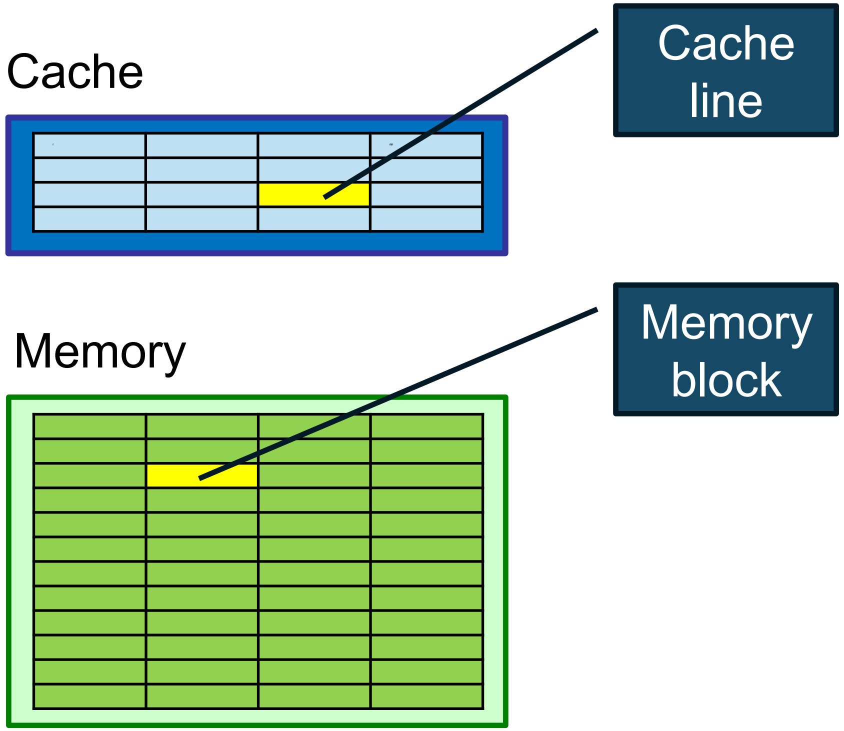
File systems

Mount point	/home	/work/scratch
Size	2+4 PB	
Access time	Fast	
Accessibility	Global (cluster)	Global (cluster)
Data availability	permanent	8 weeks
Quota*	50 GB or 4 Mio. files	20 TB or 20 Mio. files
Backup	Daily + snapshots	none

Parallelism and memory hierarchy

- A processor’s view of the memory is through its cache





Cache coherence

- Problem – **different processors may see different values**
  - Without further precautions (!)

Time	Event	Cache CPU A	Cache CPU B	Memory
0				1
1	CPU A reads X	1		1
2	CPU B reads X	1	1	1
3	CPU A stores 0 in X	0	1	0

- Cache coherence – which value will be returned by a read?
  - Cache coherence protocols prevent different versions of the same cache line from appearing simultaneously in two or more caches

Coherence of a memory system

1. A read by a processor P to a location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P
2. A read by a processor to location X that follows a write by another processor returns the written value if the read and the write are sufficiently separated in time and no other writes to X occur between the two accesses
3. Writes to the same location are serialized, that is, two writes to the same location by any two processors are seen in the same order by all processors.

Cache coherence protocols

Snooping

- Every cache that has a copy of a block of physical memory also has a copy of the sharing status of the block
- No centralized state is kept
- All caches are accessible via some broadcast medium (i.e., bus)
- All caches snoop on the medium to see whether they have a copy of a block that is being requested

Directory-based

- The sharing status of a block of physical memory is kept in one location (i.e., the directory)
- Two variants
  - Centralized directory (UMA)
  - Distributed directory (NUMA)
- Distributed directory needed if scalability is a concern

Snooping protocols

Write invalidate

- Remaining copies are invalidated on a write
- Most common for both snooping and directory-based protocols
- Guarantees exclusive access

Event	Bus activity	Cache CPU A	Cache CPU B	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

Write update or write broadcast

- All copies are updated on a write
- Consumes more bandwidth – less common

Implementation of a write invalidate

- Invalidate is performed by broadcasting address to be invalidated on the bus
- All processors snoop on the bus and watch the addresses: if address is in their cache, data are invalidated
- Writes to a shared data item are serialized – only one processor can have access to the bus at a time

Finding data items on a miss

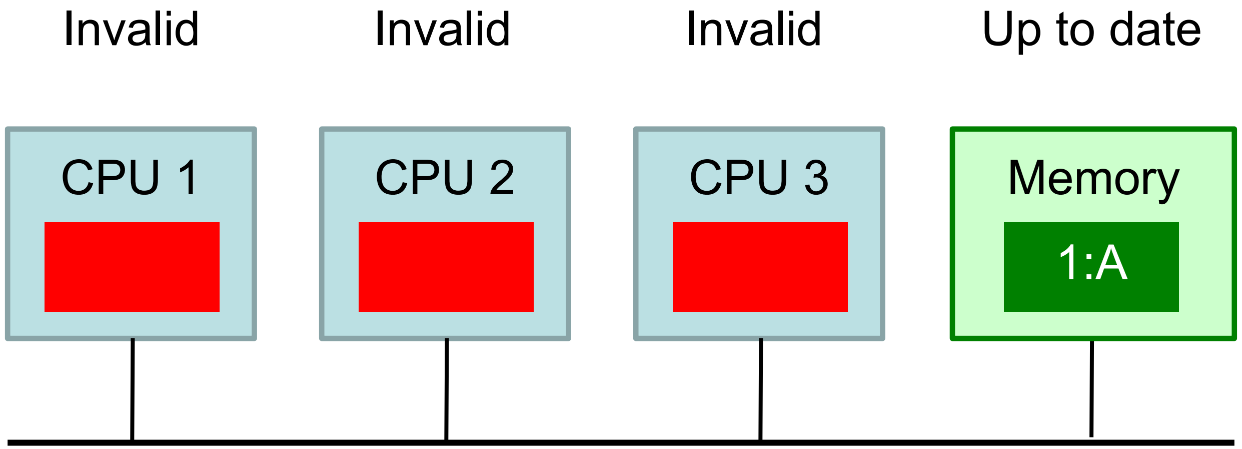
- **Write-through caches**
  - Data can be retrieved from main memory
- **Write-back caches**
  - Cache misses handled via snooping – if processor has dirty copy of a cache line, it provides the block in response to a request and causes memory access to be aborted
  - Complexity arises from transferring block to requesting processor – can take longer than getting it from main memory - especially if processors are on separate chips
  - Lower memory bandwidth demands & hence more scalable – often used at outermost cache levels
  - Example: MESI protocol

MESI protocol

- Popular write-back cache coherence protocol
  - Used e.g. in Intel Core i7
- Each cache line can be in one of the following four states

1. Modified = cache line is valid; memory is invalid; no other copies exist
2. Exclusive = no other cache holds the line; memory up to date;
3. Shared = multiple caches may hold the line; memory up to date
4. Invalid = no valid data

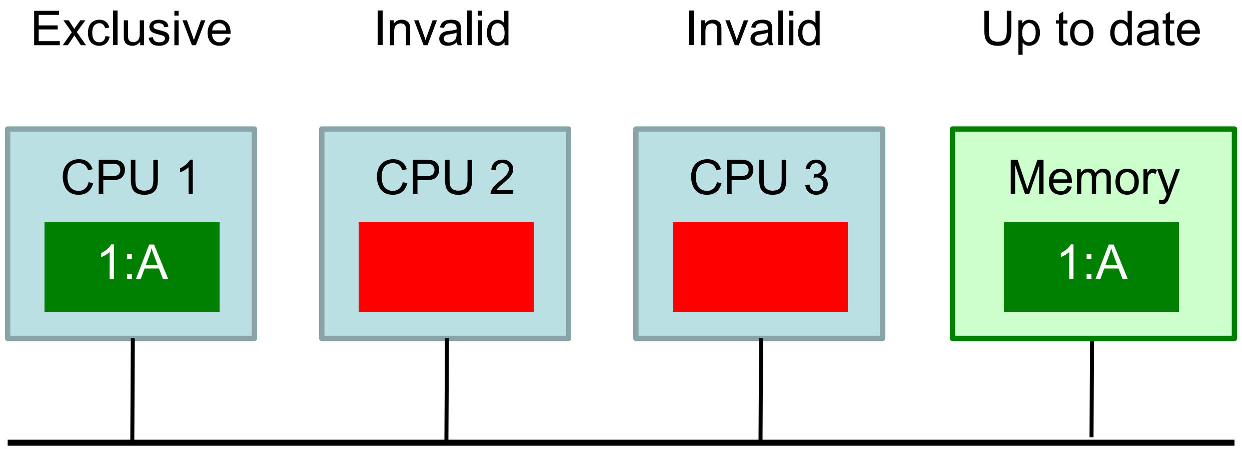
MESI protocol – example



Next: CPU1 read 1

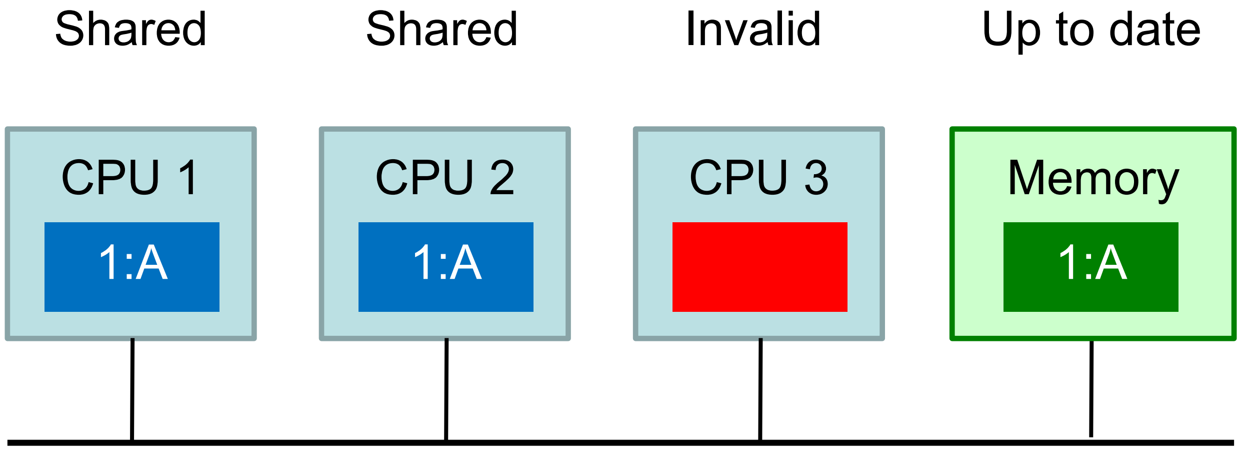
N:X = cache line address : contents version

MESI protocol – example



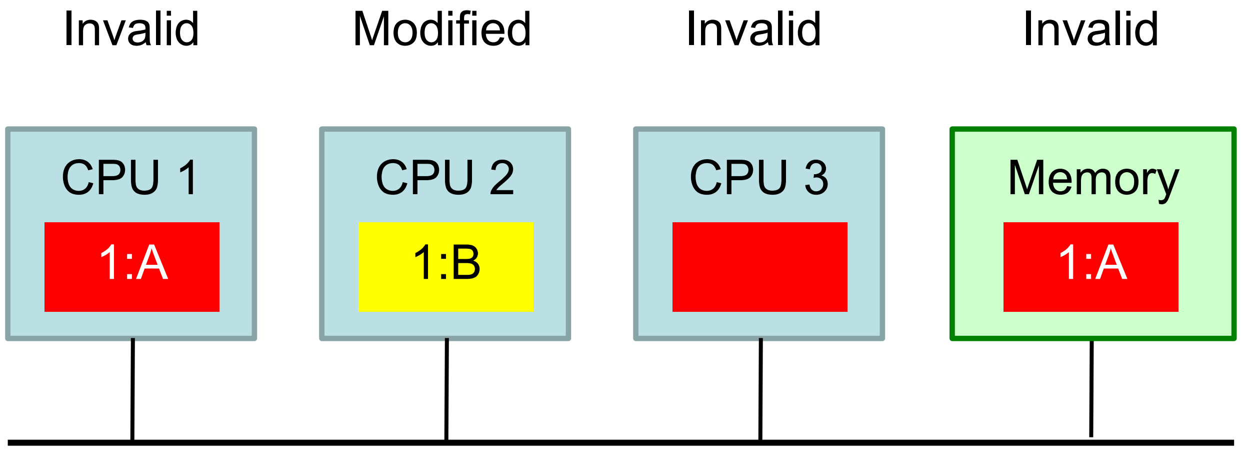
Next: CPU2 reads 1

MESI protocol – example



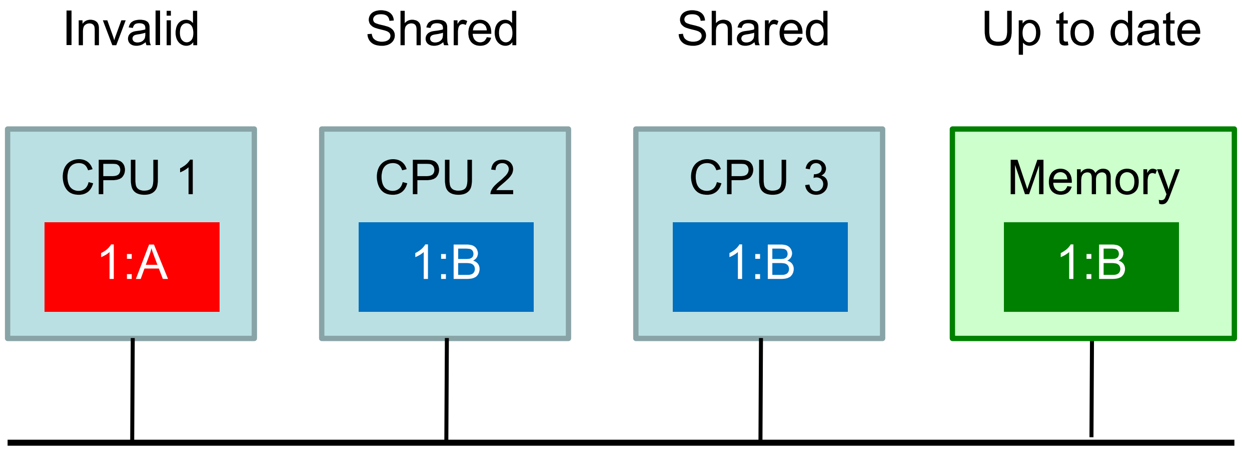
Next: CPU2 writes B to 1

MESI protocol – example



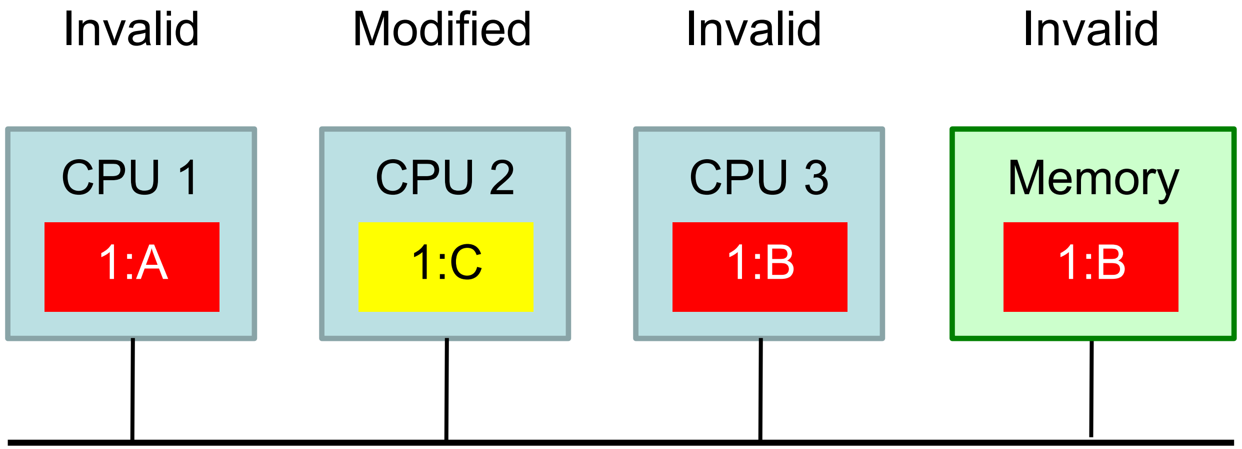
Next: CPU3 reads 1

MESI protocol – example



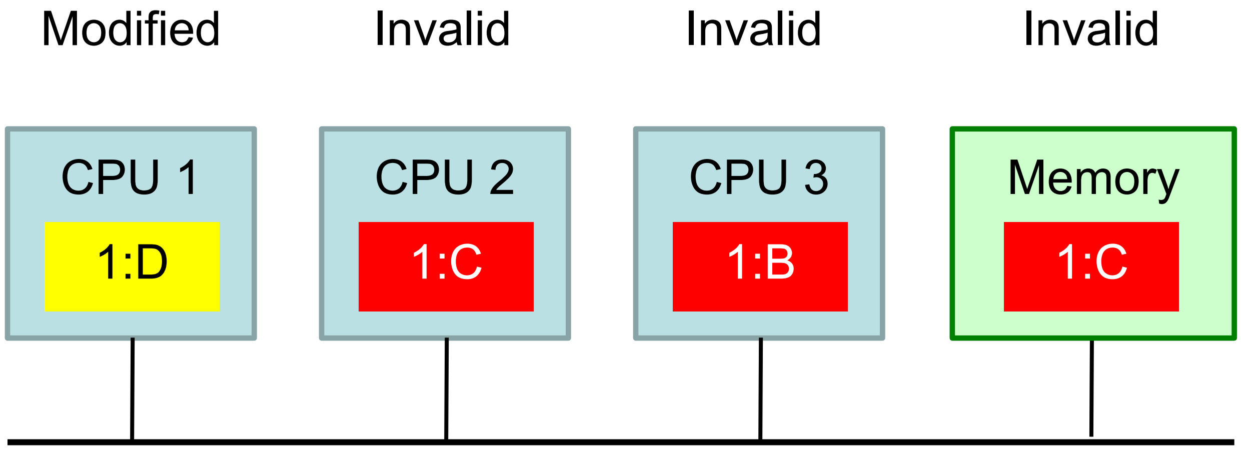
Next: CPU2 writes C to 1

MESI protocol – example



Next: CPU1 writes D to 1

MESI protocol – example



Coherence misses

- Uniprocessor misses
  - Compulsory, capacity, conflict
- **True sharing miss** – cache miss occurring because a block was invalidated by another processor writing the same word
  - Arises from communication of data through coherence mechanism
  - Independent of cache-line size
- **False sharing miss** – cache miss occurring because a block was invalidated by another processor writing a different word
  - Miss that would not occur if block size were one word

Coherence misses (2)

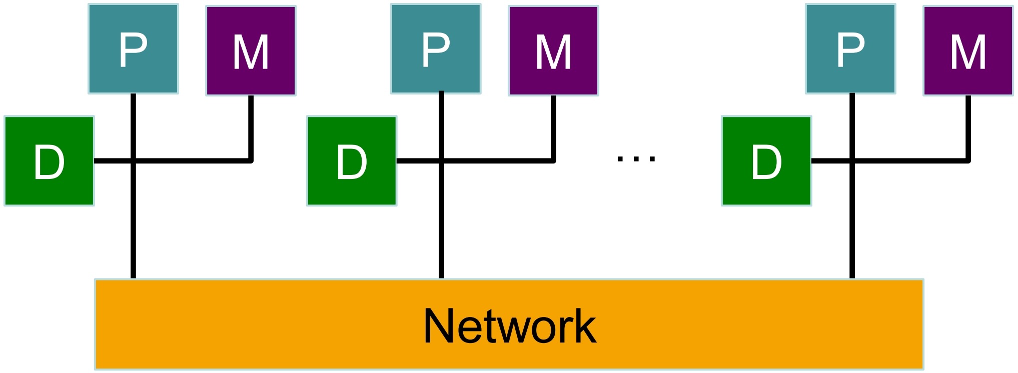
- Assume X1 and X2 are in the same cache line, and initially in the caches of P1 and P2 in shared state

Time	P1	P2	Miss
1	Write X1		Invalidate X1 & X2 on P2
2		Read X2	False miss – X1 not used by P2
3	Write X1		Invalidate X1 & X2 on P2
4		Write X2	False miss – X1 not used by P2; Invalidate X1 & X2 on P1
5	Read X2		True miss – P2 has written X2 before

- Coherence misses most important for tightly-coupled applications that share significant amounts of user data

Cache-coherent NUMA systems

- Coherence of caches established via directory
  - Distributed database storing location and status cache of lines
  - Requires fast hardware because it must be queried on every memory reference



Example

- 256 nodes
- 1 CPU + 16 MB RAM per node
- Total memory 2^{32} bytes
- 2^{26} cache lines, 64 bytes each
- Memory statically allocated among nodes
  - 0-16M in node 0, 16-32M in node 1, etc.
- Directory of each node holds entries for the 2^{18} cache lines comprising its 2^{24} bytes of memory
- Assumption: a line can be held in at most one cache

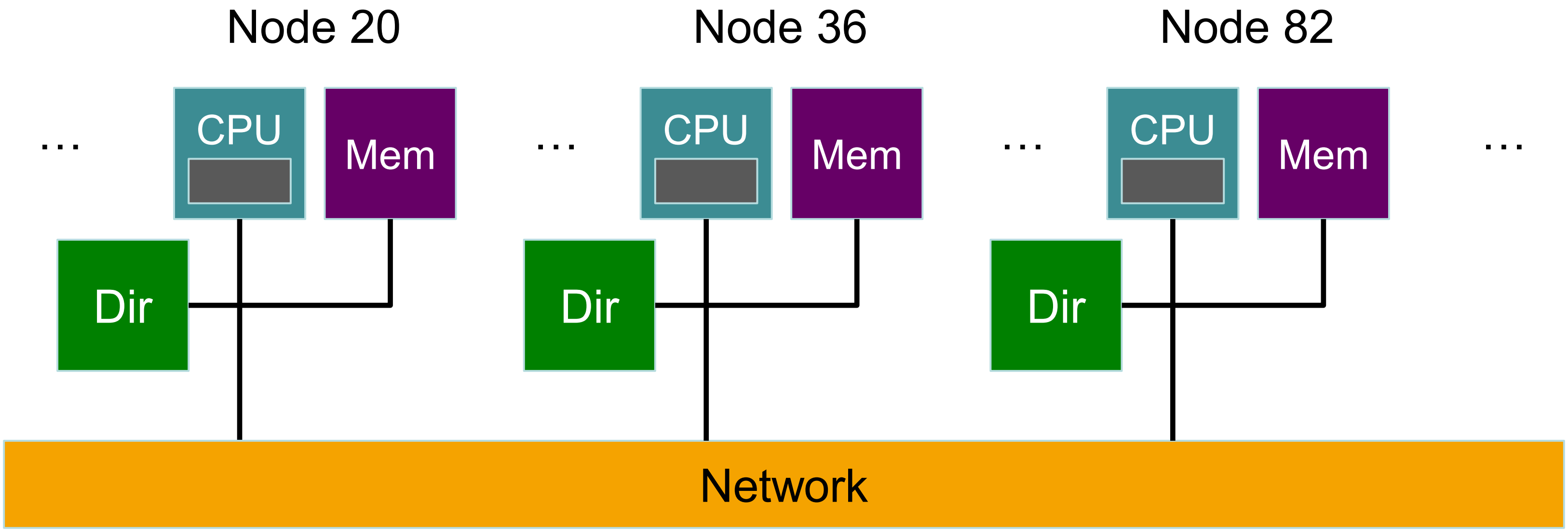
Example (2)

- Consider LOAD instruction from CPU 20 that references a cached line at address 0x24000108
- MMU splits address into three parts

Bits	8	18	6
	Node	Line	Offset

- Address 0x24000108 = node 36 ; line 4 ; offset 8

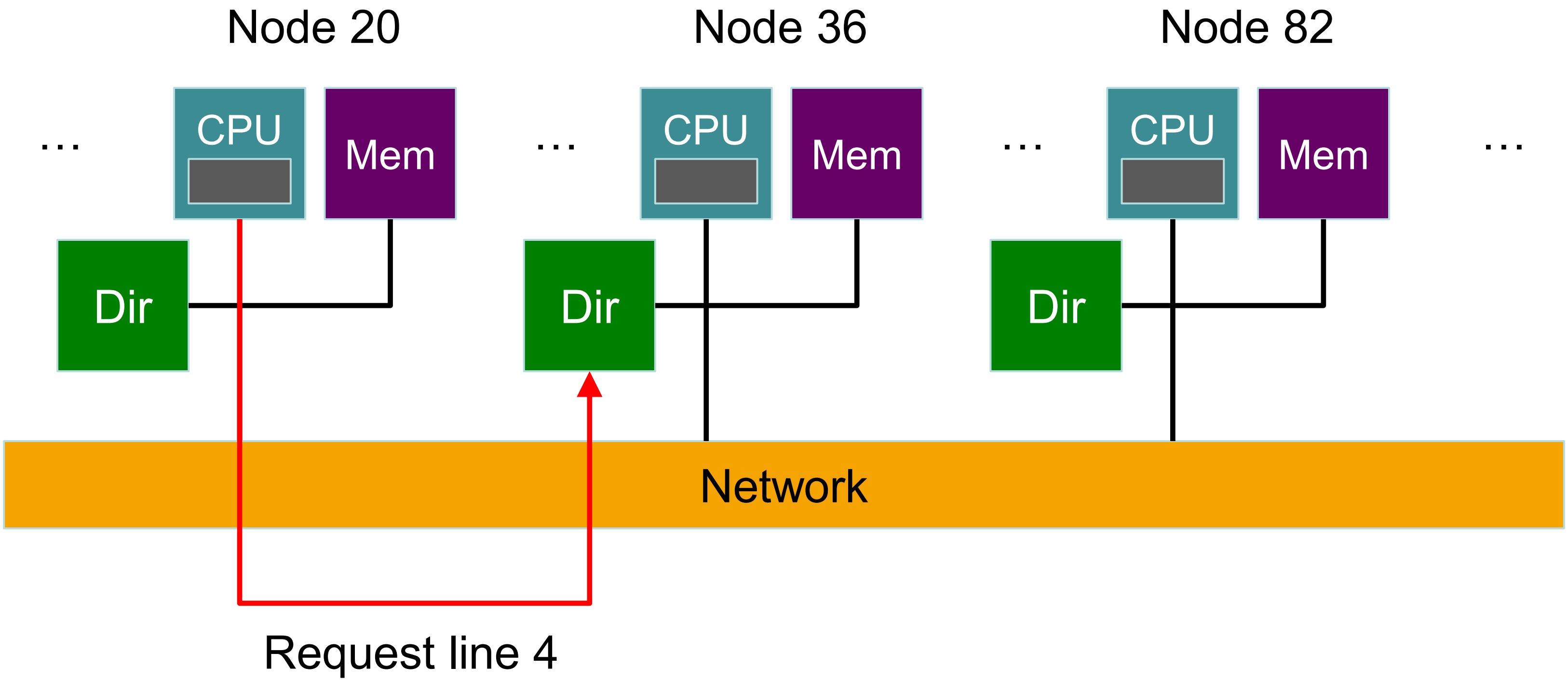
Example (3)



Line	Cached?	Where?
0	0	
1	0	
2	1	82
3	0	
4	0	

Directory of CPU 36

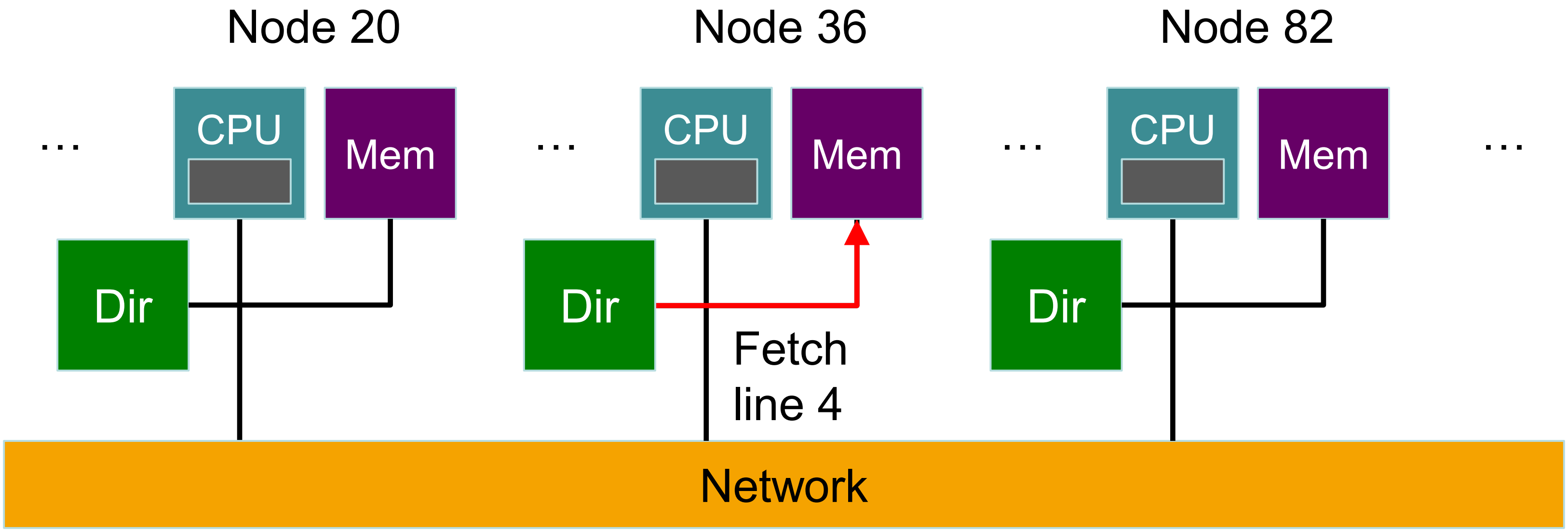
Example (3)



Line	Cached?	Where?
0	0	
1	0	
2	1	82
3	0	
4	0	

Directory of CPU 36

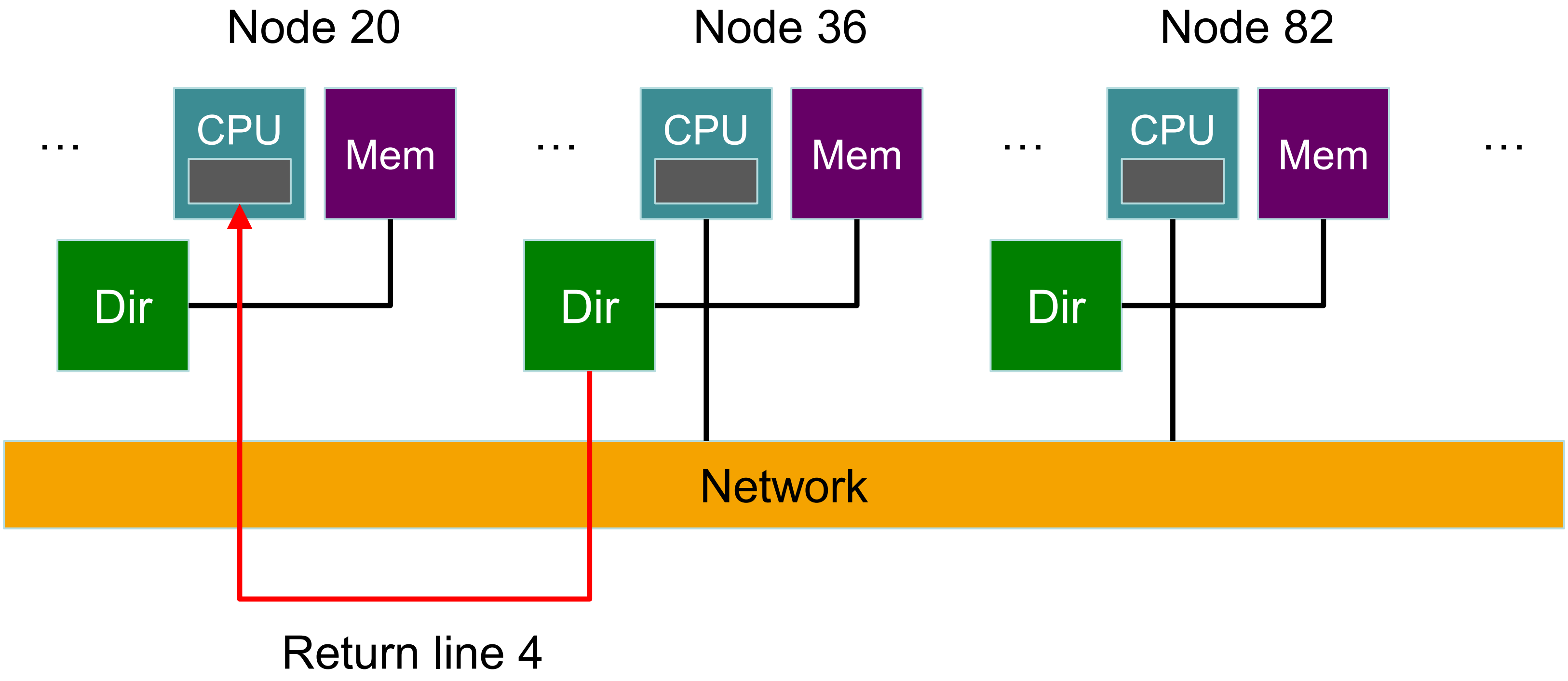
Example (3)



Line	Cached?	Where?
0	0	
1	0	
2	1	82
3	0	
4	0	

Directory of CPU 36

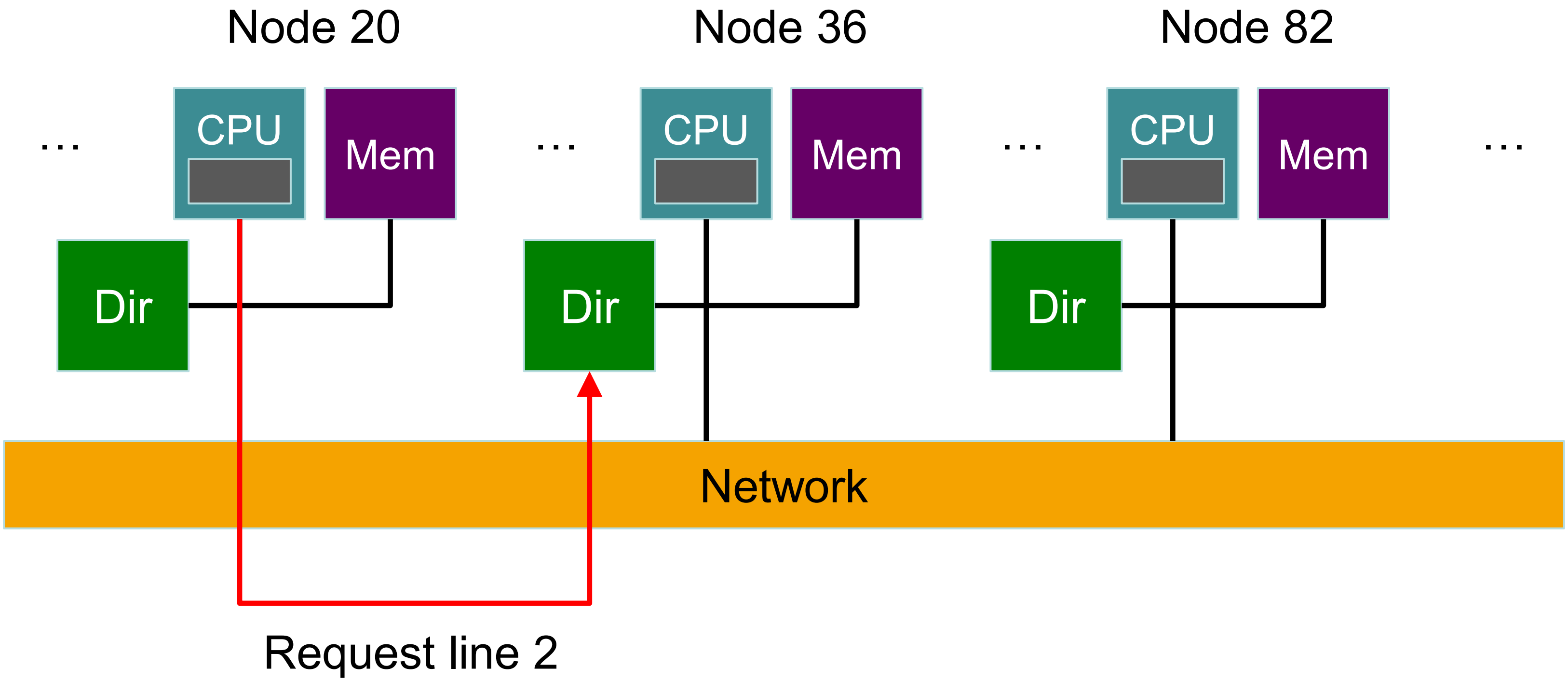
Example (3)



Line	Cached?	Where?
0	0	
1	0	
2	1	82
3	0	
4	1	20

Directory of CPU 36

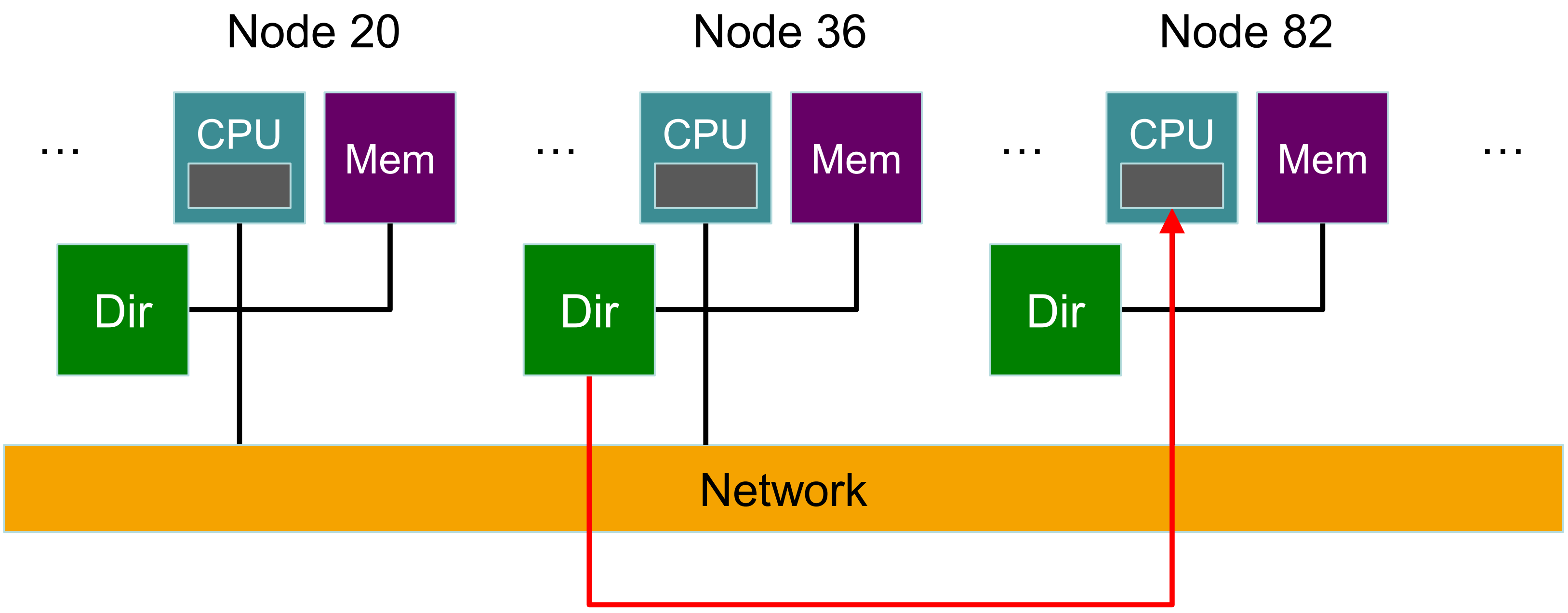
Example (3)



Line	Cached?	Where?
0	0	
1	0	
2	1	82
3	0	
4	1	20

Directory of CPU 36

Example (3)

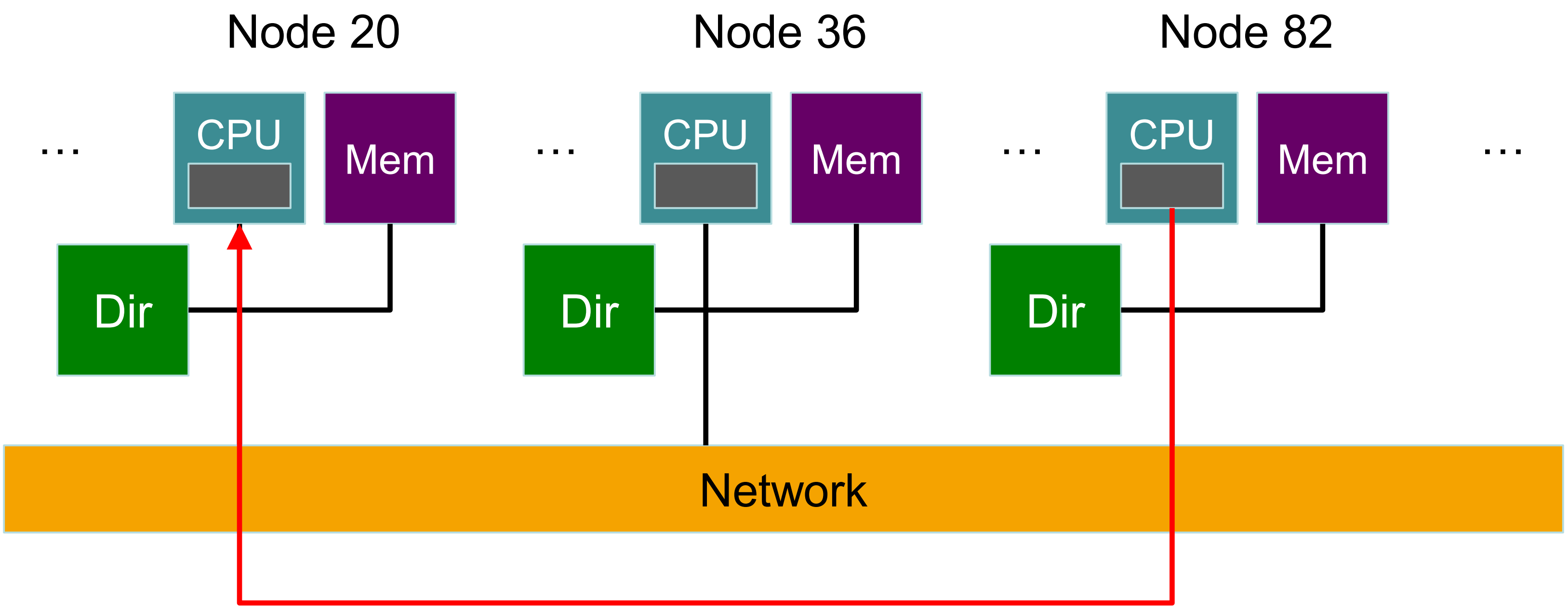


Please send line 2 to node 20

Line	Cached?	Where?
0	0	
1	0	
2	1	82
3	0	
4	1	20

Directory of CPU 36

Example (3)



Return line 2 to node 20

Line	Cached?	Where?
0	0	
1	0	
2	1	20
3	0	
4	1	20

Directory of CPU 36

Number of copies

- Directory has 2^{18} 9-bit entries = 1.76% of total memory
- Limitation: a line can be cached at only one node
- Alternatives
  - k entries per line, allowing copies at up to k nodes
  - Bitmap with one bit per node (substantial increase in memory overhead)
  - 8-bit field as head of a linked list (requires extra storage for list pointers and time overheard for searching the list)

Status of a line

Another optimization is to keep track of a line’s status (dirty or clean)

- The home node can satisfy a read request for a clean line from its local memory without having to forward it to another node’s cache
- A read request for a dirty line must still be forwarded to the node holding the copy
- No advantage if only one copy is allowed because any request requires invalidation of the previous copy
- Modification of a cached line requires home node to be informed and invalidation of all other copies
- Potentially significant coherence traffic

Memory semantics

- Shared memory = image of a **single shared address space**
  - Promises intuitive programming
- Implementation quite complex in reality
  - Many memory modules, each holding some portion of the physical memory
  - CPUs and memories often connected by complex interconnection network
  - Memory hierarchy with multiple levels (registers down to main memory)

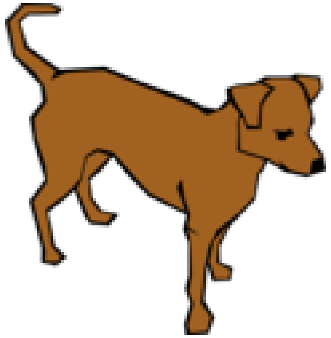
Order of updates

Can be influenced by two factors

- Order in which memory request “messages” arrive not necessarily the same as the one in which they were issued
  - A single thread may observe writes in an order different from the order another thread wrote them
  - Order may even differ among multiple readers
- Compiler may re-order instructions (even possible on uni-processor systems)

Piecewise update of reality

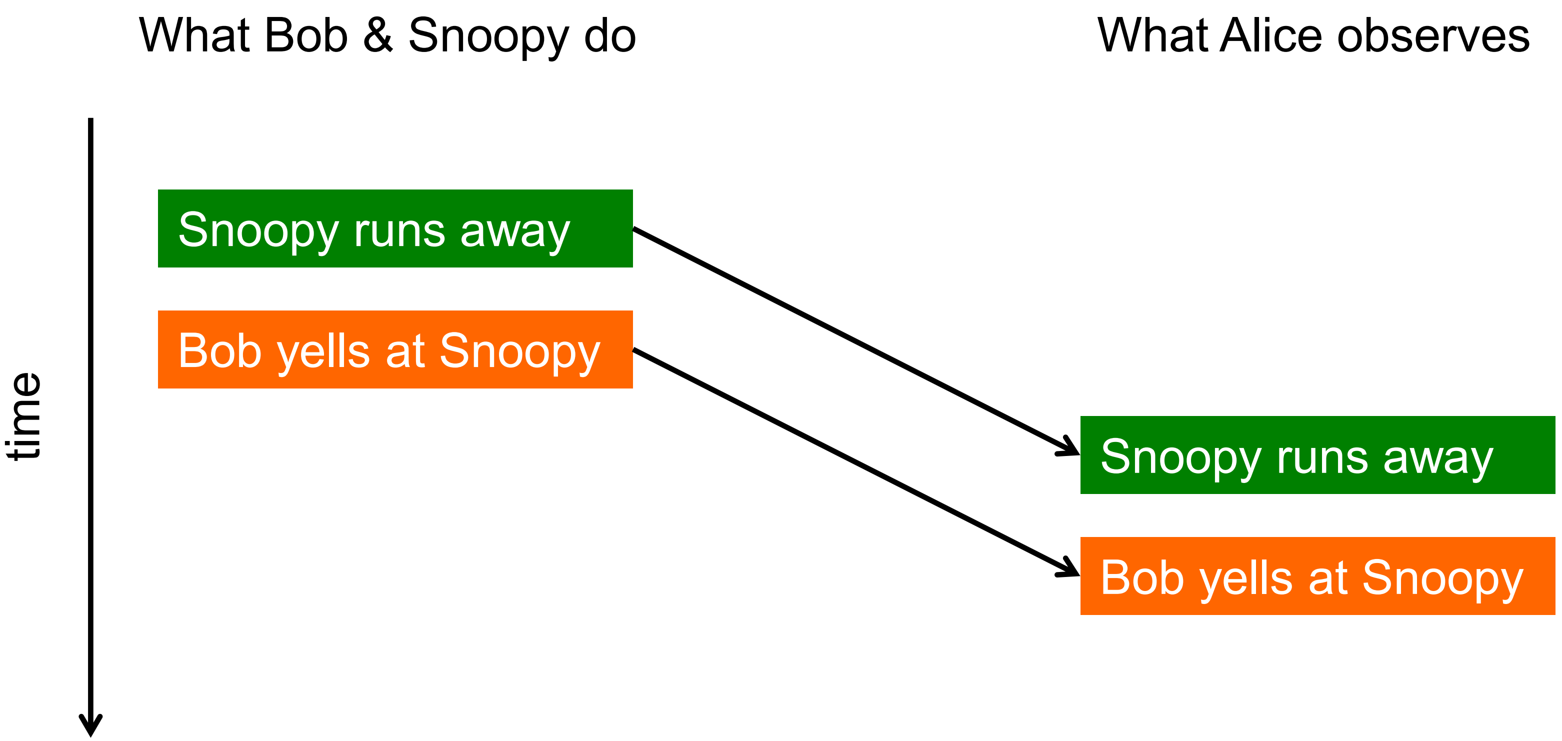
Bob



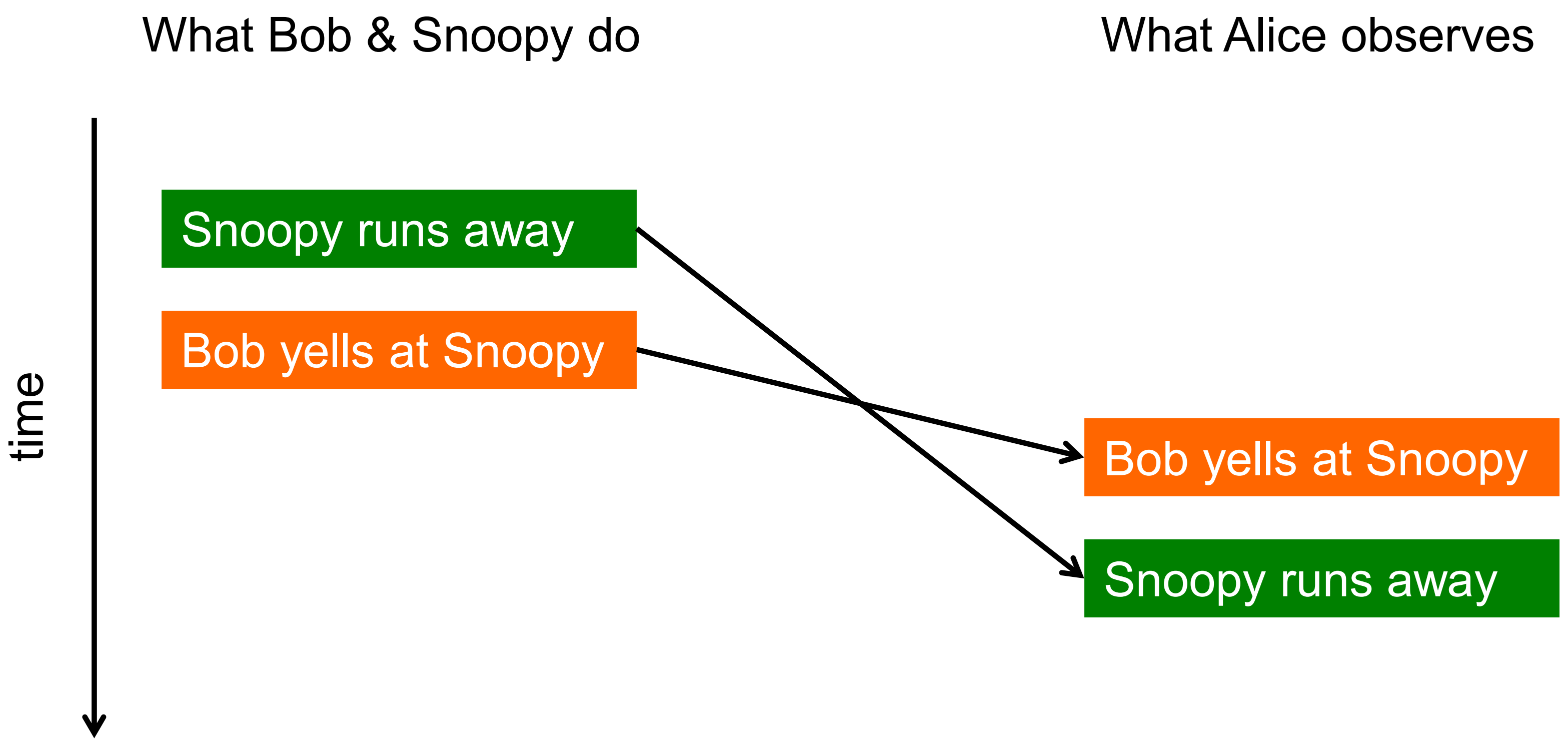
Snoopy
(Bob’s dog)

Alice
(observer)

Piecewise update of reality (2)



Perceived update order reversed



Instruction reordering

Thread 1

```
int x;
bool x_init = false;

void init()
{
  x = initialize();
  x_init = true;
  // …
}
```

Thread 2

```
extern int x;
extern bool x_init;

void f2()
{
  int y;
  while (!x_init)
    usleep(10000);
  y = x;
  // …
}
```

Instruction reordering (2)

- Compiler (or hardware instruction scheduler) may decide to execute `x_init = true` first
  - No thread-local dependence
  - Thread 2 may assign an uninitialized x to y

- Since thread 2 makes no assignments to `x_init`, an optimizer may decide to lift evaluation of `x_init` out of the loop
  - Thread 2 may sleep forever or not at all

Memory coherence vs consistency

Coherence – behavior of the memory system when a **single** memory location is accessed by multiple threads

Consistency – ordering of accesses to **different** memory locations, observable from various threads in the system

- When must a processor see a value that has been updated by another processor?
- In what order does a processor observe the data writes of another processor?
- Aka **memory ordering**

Sequential consistency

The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program

[Lamport, 1979]

- Advantage – simple programming paradigm
- Disadvantage – potential performance degradation

Relaxed consistency models

- Sequential consistency is convenient, but impractical because of potential performance degradation
- Idea of relaxed consistency models
  - Let reads and writes complete out of order
  - Use synchronization to enforce ordering where important
- Relaxed consistency models can be distinguished by the orderings they guarantee / relax

Desirable in practice: **sequential consistency for data-race-free programs**

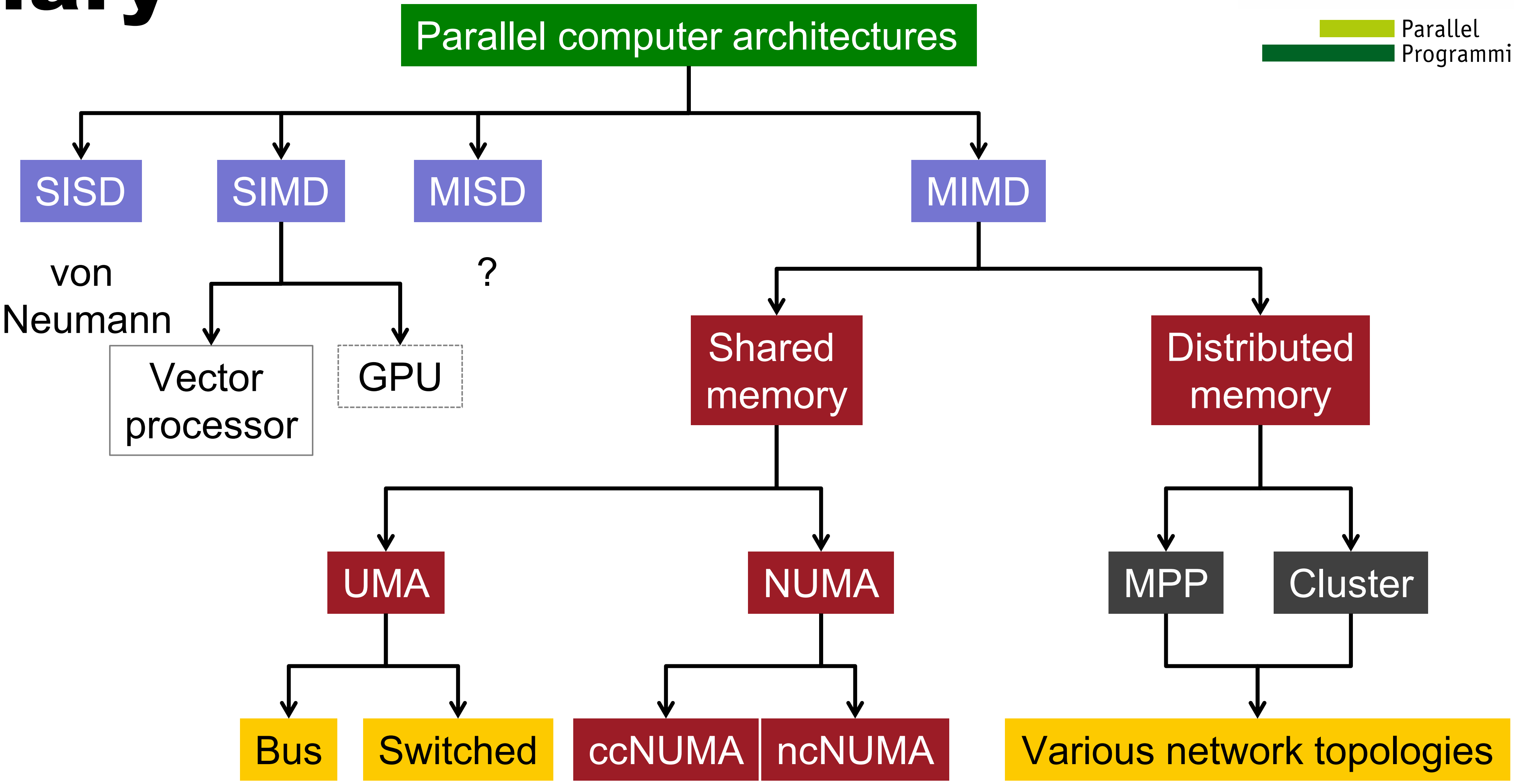
Synchronization

- A program is synchronized if all accesses to shared data are ordered by synchronization operations
- Updates of a single location not ordered by synchronization are called **data races**
- Synchronization allows programs to behave as under sequential consistency even if the architecture implements a more relaxed consistency model
- Building synchronization mechanisms is hard

Synchronization (2)

- Synchronization mechanisms implemented using hardware supplied synchronization instructions
- Uninterruptible instruction or instruction sequence capable of **atomically reading and changing a value** together with the ability to tell whether read and write were performed atomically
- Synchronization can become a bottleneck in
  - Large-scale multiprocessors
  - High-contention situations

Summary



cc = cache-coherent, nc = not cc