Outline

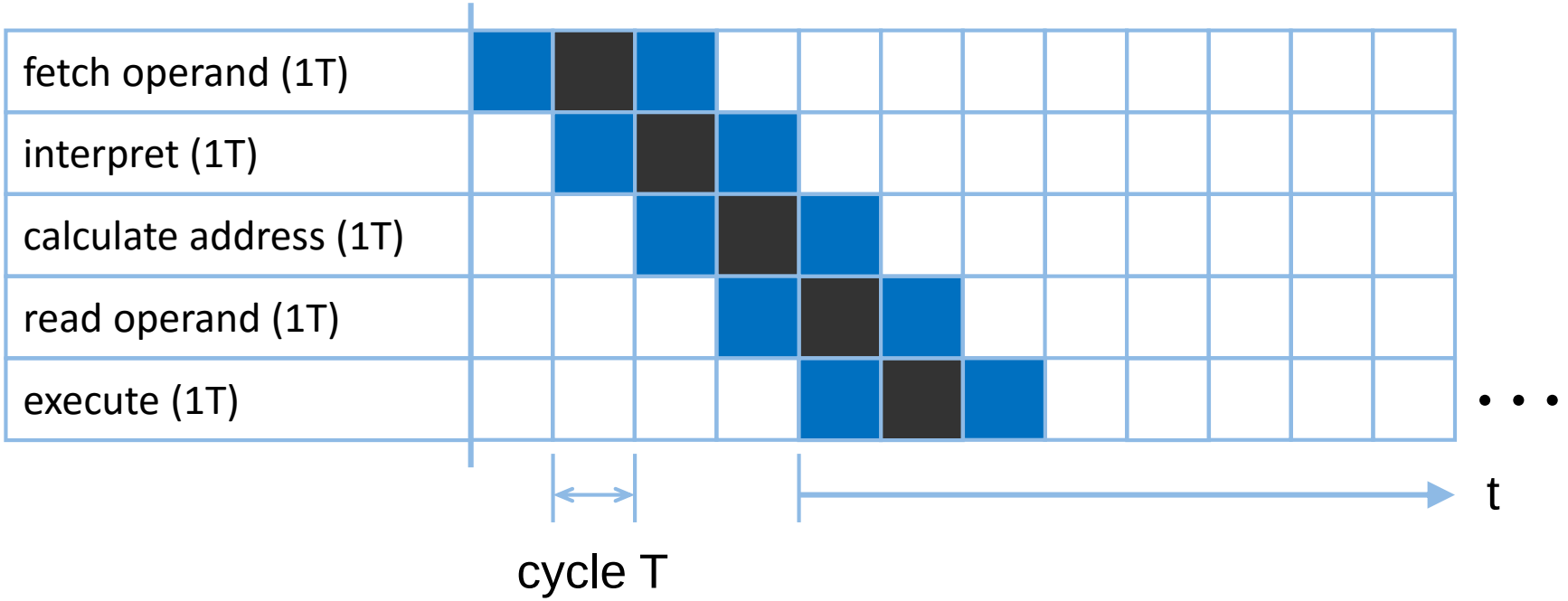
1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → **Multithreaded processors (SMT)**
   - → Multicore processors
   - → Shared-memory computers
     - → Cache coherence
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → Networks
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

Motivation of Hardware support for Multiple Threads

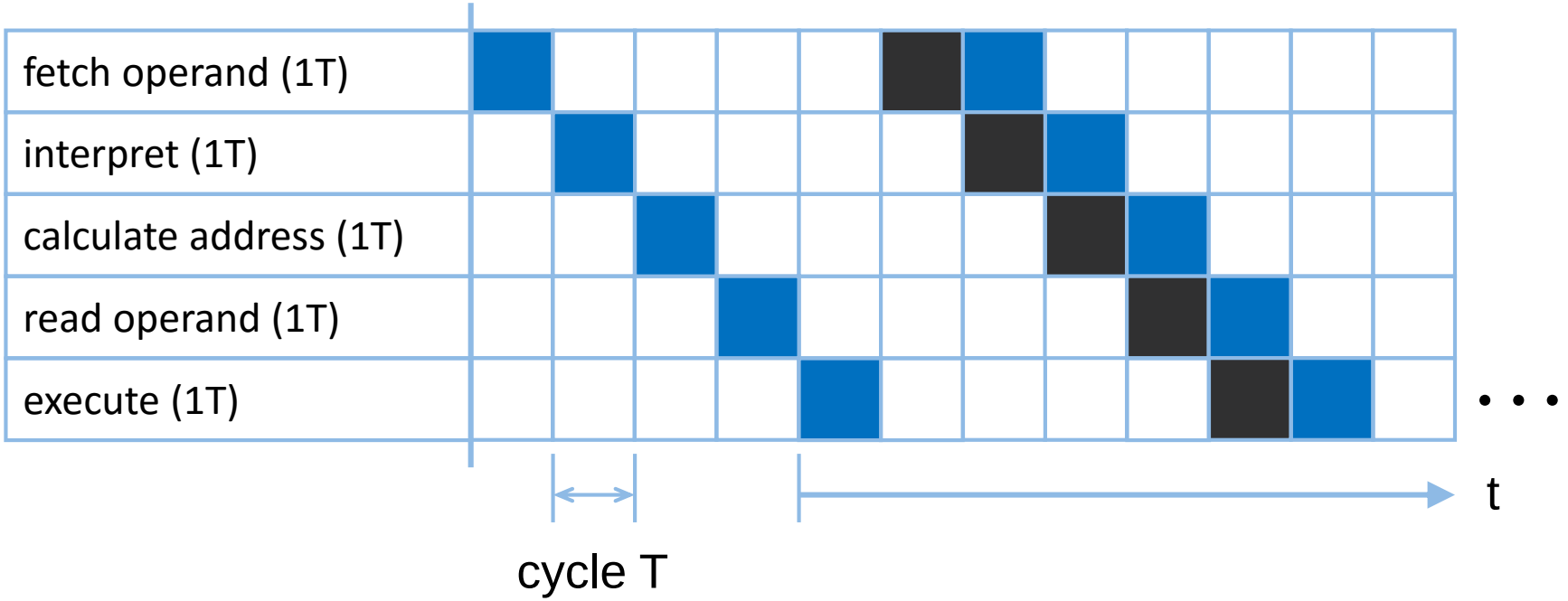
- **Limits of Parallelism on Instruction Level**
  - → Limits of Pipelining
  - → Limits of Superscalarity

Recap: Pipelining

- **Technique to increase throughput**



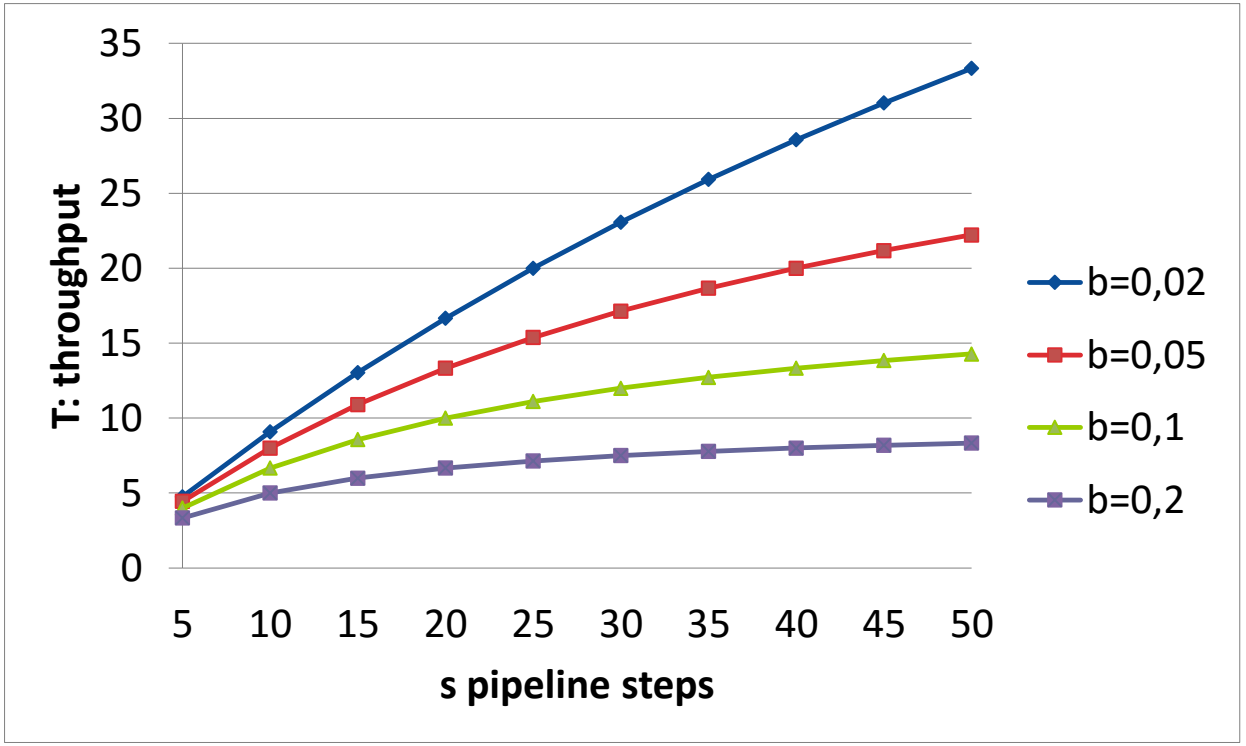
- **Example: misspredicted jump**



Limits of Pipelining

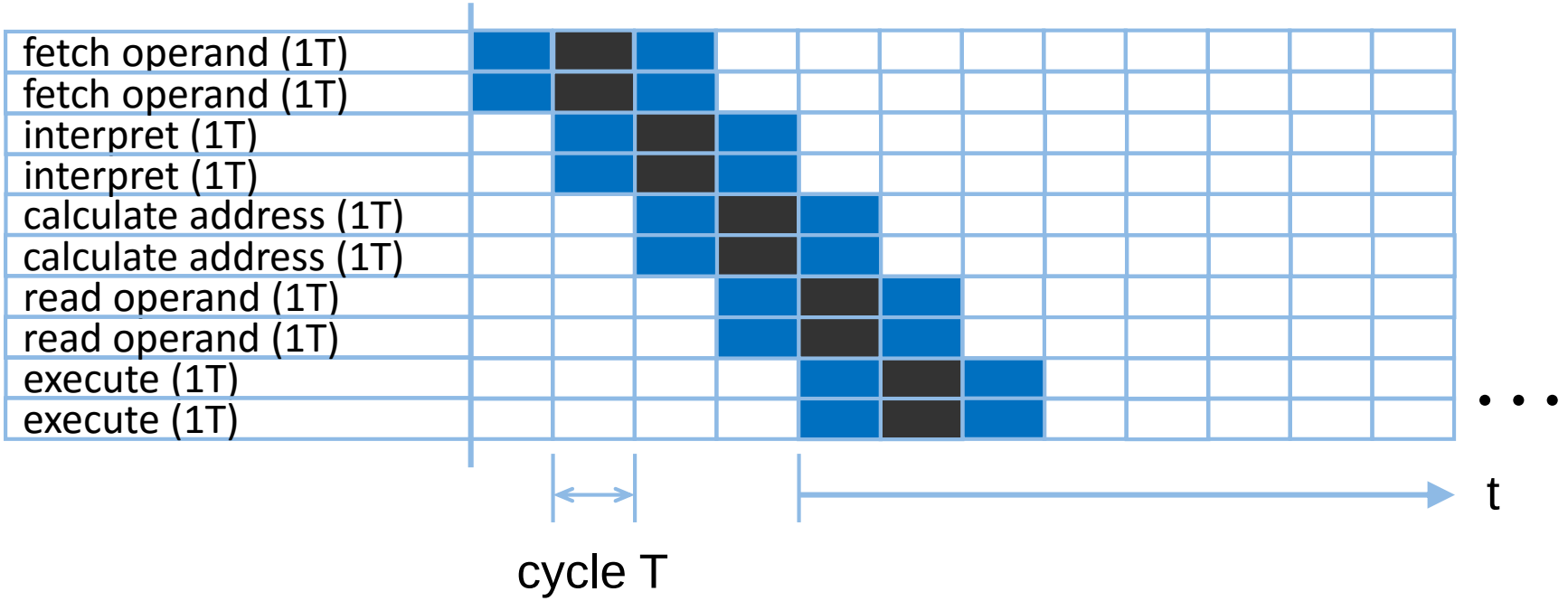
- **Throughput depend on the probability (b) that a pipeline-hazard occurs**
  - → s: number of pipeline steps
  - → c: cycles per instruction
  - → s-k: penalty for a hazard

$$T(S) = \frac{S}{c \cdot (1 + (S - k) \cdot b)}$$



Recap: Superscalarity

- **Process multiple instructions per clock cycle**



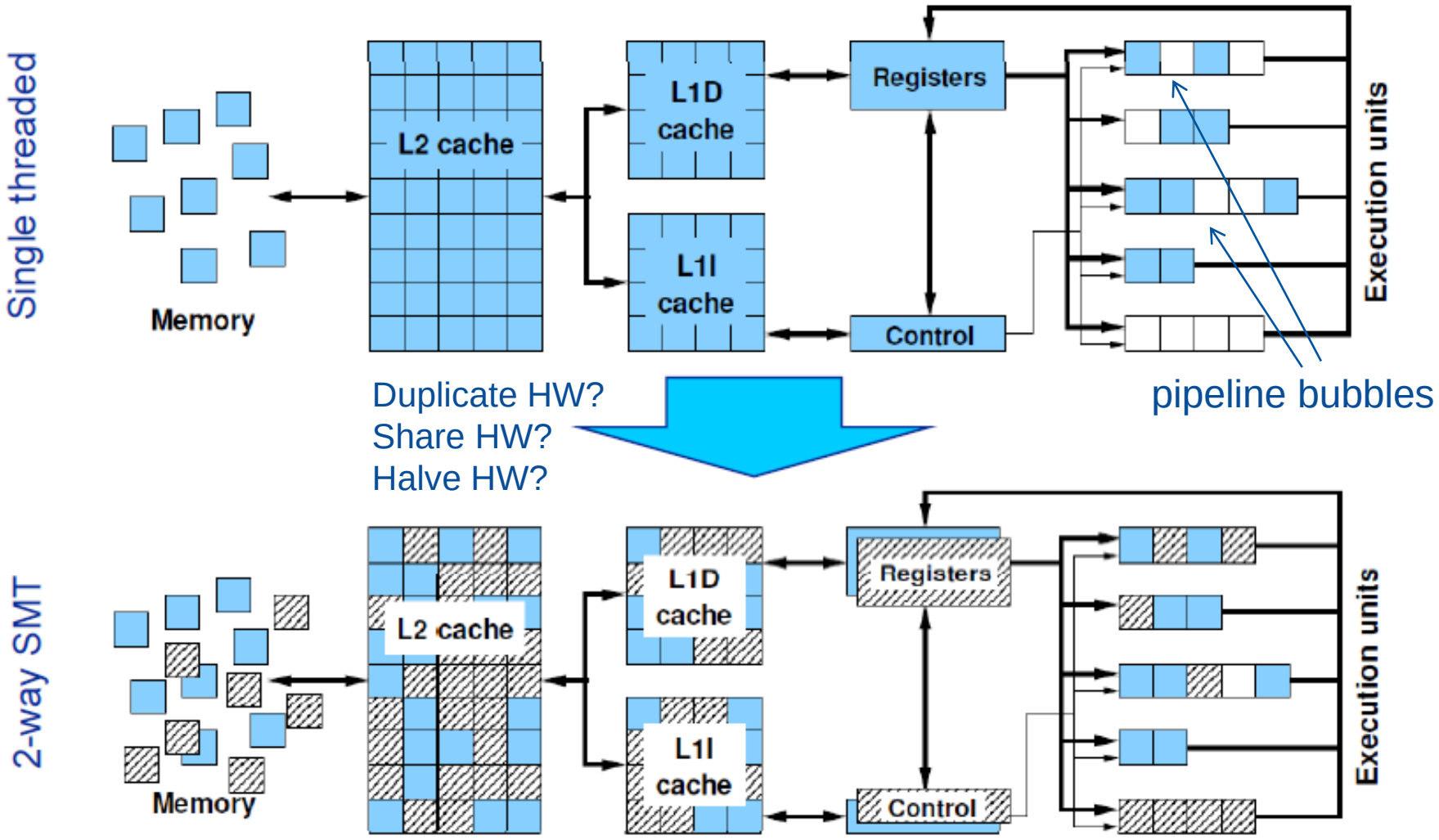
Limits of superscalar Architectures

- Single instruction stream has not enough parallelism
- Dependency checking is costly in time and space (hardware)
- Branch handling
- Register renaming

Multithreaded processors

- **Processors that support more than one path of program execution, e.g. multicore processors execute threads**
- **Often single threaded execution only occupies a part of the available resources**
  - → FP Pipelines
  - → CPU may be idle while waiting for data from memory
  - → Not optimal stream of instructions
- **May lead to frequent pipeline bubbles**
  - → Unfortunately this happens quite often
- **To compensate the concept of SMT (Simultaneous multithreading) is introduced**

Example for 2-way SMT



SMT

- **Usually this concept is referred to as HyperThreading (Intel) or SMT**
  - → Common to all implementations is that the core is “duplicated” on a logical level – e.g. for 2-way SMT one core is represented as 2 logical cores by the OS
  - → Threads that run on logical cores that correspond to the same physical core share all Cache levels and executional resources like Pipelines, Superscalar units, etc…
    - → Correct pinning of threads is mandatory for good utilization of SMT (see later for more information on pinning)
- **SMT is most efficient if the instruction streams have completely different operations pending**

SMT resource utilization

- **Scientific codes tend to utilize hardware resources quite well**
  - → Usually optimized code with loop unrolling, blocking, etc..
  - → Data and Instruction parallelism is high
- **SMT can only increase the throughput if code does not utilize resources in an optimal way**
  - → If the code has a good resource utilization, SMT might not be needed
    - →In this case, pin threads only to physical cores
    - →Or switch off SMT (BIOS, OS config)

Pipeline bubbles can be avoided with SMT

- **Benefit: Better pipeline throughput**
- **Beware: SMT threads share all cache levels**

Thread 0:

```
for(i=0; i < N; ++i)
{
  a[i] = a[i-1]*c;
}
```

Thread 1:

```
for(i=0; i < N; ++i)
{
  b[i] = func(i)*d;
}
```

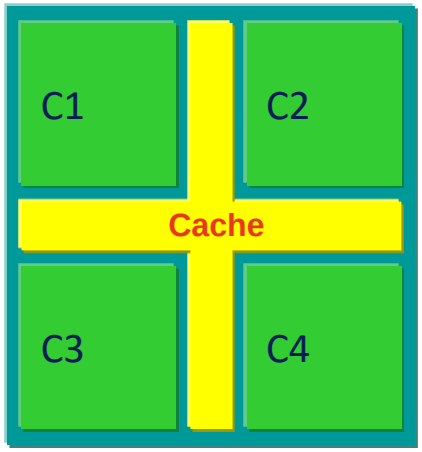
Usually execution stalls,
until a[i-1] is calculated
→ Pipeline bubbling

With SMT, the pipeline
can be filled up with instructions
of an independent second
instruction stream

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multithreaded processors (SMT)
   - → **Multicore processors**
   - → Shared-memory computers
     - → Cache coherence
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → Networks
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

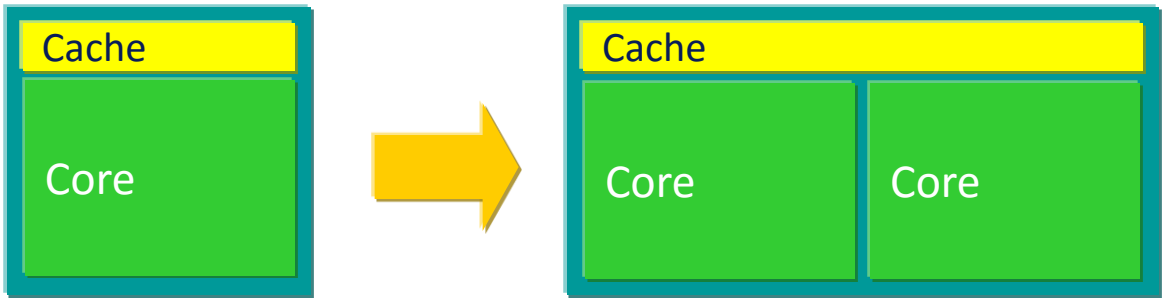
Multicore Processor



- A multi-core processor integrates two or more independent computing cores on a single die
- Commonly the cores share caches
- Cores can operate on different voltage and run different frequencies

Multi-core design

Rule of thumb: Reduction of 1% voltage and 1% frequency reduces the power consumption by 3% and the performance by 0.66%.

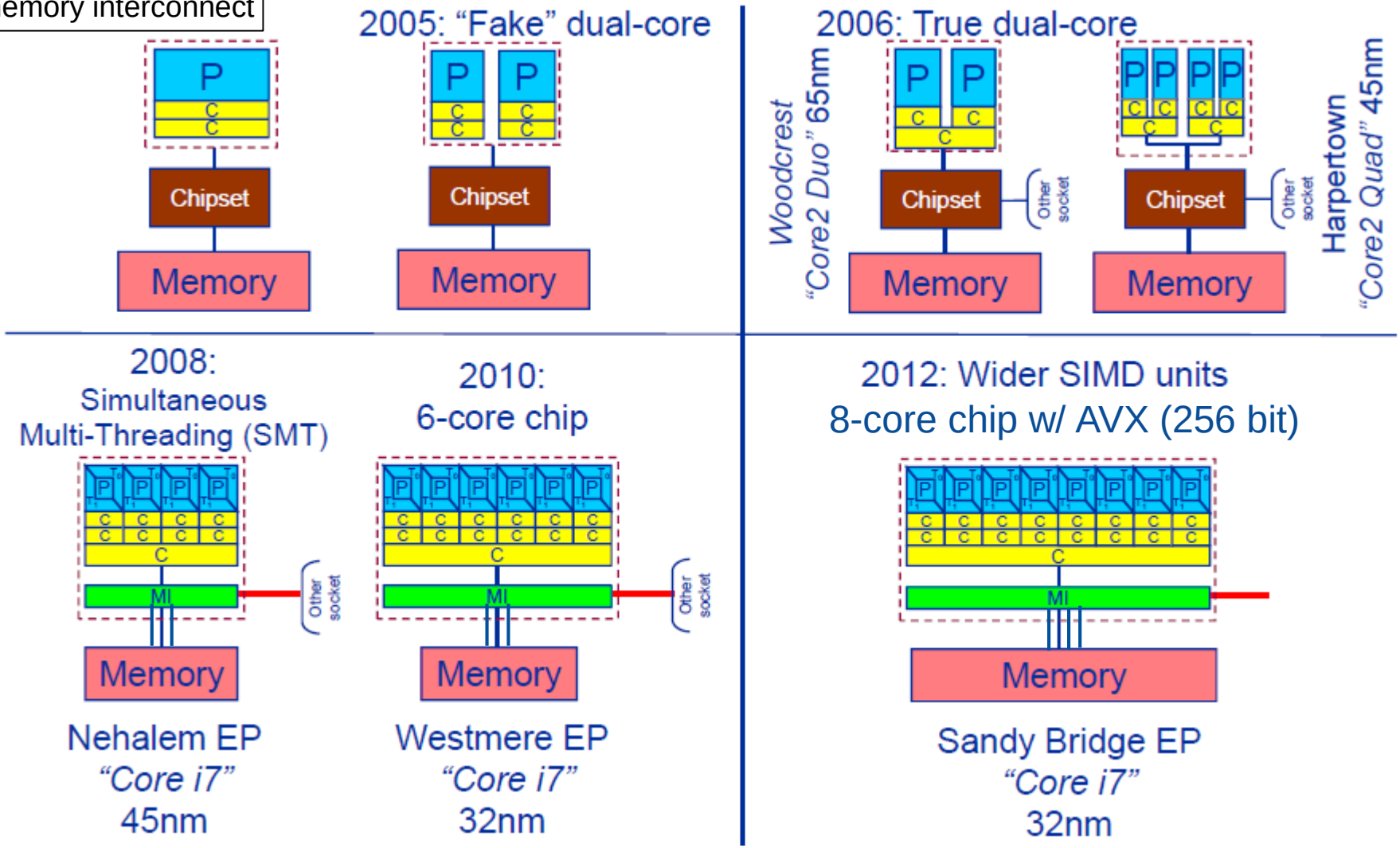


Voltage	= 1	Voltage	= -15%
Freq	= 1	Freq	= -15%
Area	= 1	Area	= 2
Power	= 1	Power	= ~1
Perf	= 1	Perf	= ~1.8

(Based on slides from Shekhar Borkar, Intel Corp.)

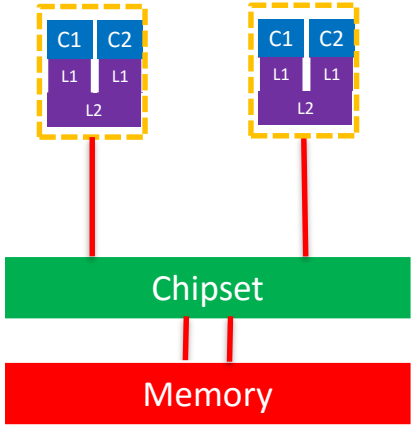
The multicore evolution so far

P: processor
C: cache
MI: memory interconnect

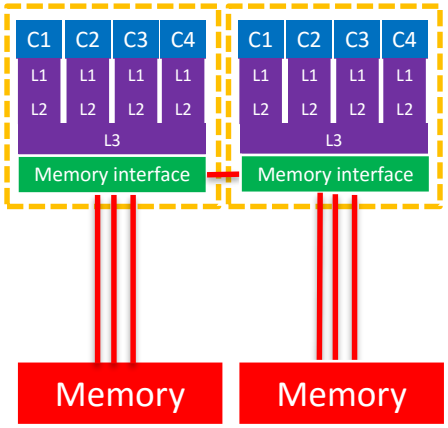


Multicore designs

- **Early multicore design**
  - → Uniform Memory Architecture (UMA)
  - → Flat Memory design



- **Recent multicore design**
  - → ccNUMA (Cache Coherent Non-Uniform Memory Architecture)
  - → Memory Interface + HT/QPI provides inter-socket connectivity



- **More details later**

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → **Shared-memory computers**
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → Networks
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

Shared-memory computers

- **“A shared-memory parallel computer is a system in which a number of running CPUs work on a common, shared physical address space” [1]**
- **Two basic categories**
  - → Uniform Memory Access (UMA):
    - → Flat Memory model: The Memory is equally accessible to all processors with the same performance and bandwidth properties
    - → See SMP systems (Symmetric Multi Processor systems)
  - → Cache-Coherent Non Uniform Memory Access (ccNUMA)
    - → Memory is distributed physically: Usually each processor has it’s own memory module and can access the memory of other processors remotely. Access times differ for local and remote access

[1] Hager, Wellein: Introduction to High Performance Computing for Scientists and Engineers

Cache coherence

- **Cache coherence protocols are implemented in hardware**
- **Most widespread protocol for cache coherence: MESI protocol**
- **Cache lines can be in one of four different stages:**
  - → **M**odified: Cache line has been modified and resides inside the cache. It resides in no other Cache. To ensure consistency it needs to be evicted
  - → **E**xclusive: Cache line has recently been read from memory but not yet modified. It does not reside in any other cache
  - → **S**hared: Cache line has recently been read from memory but not yet modified. There exist other copies of this cacheline in one or more of the other caches
  - → **I**nvalid: This cache line was invalidated recently or never loaded. This state may occur if the cacheline was in Shared state and one CPU requested exclusive ownership

Memory Model

- **From Wikipedia, the free encyclopedia:**
  - → “In computing, a memory model describes the interactions of threads through memory and their shared use of the data.”
  - → “Modern programming languages implement a memory model. The memory model specifies synchronization barriers that are established via special, well-defined synchronization operations such as acquiring a lock by entering a synchronized block or method. The memory model stipulates that changes to the values of shared variables only need to be made visible to other threads when such a synchronization barrier is reached. Moreover, the entire notion of a race condition is entirely defined over the order of operations with respect to these memory barriers”

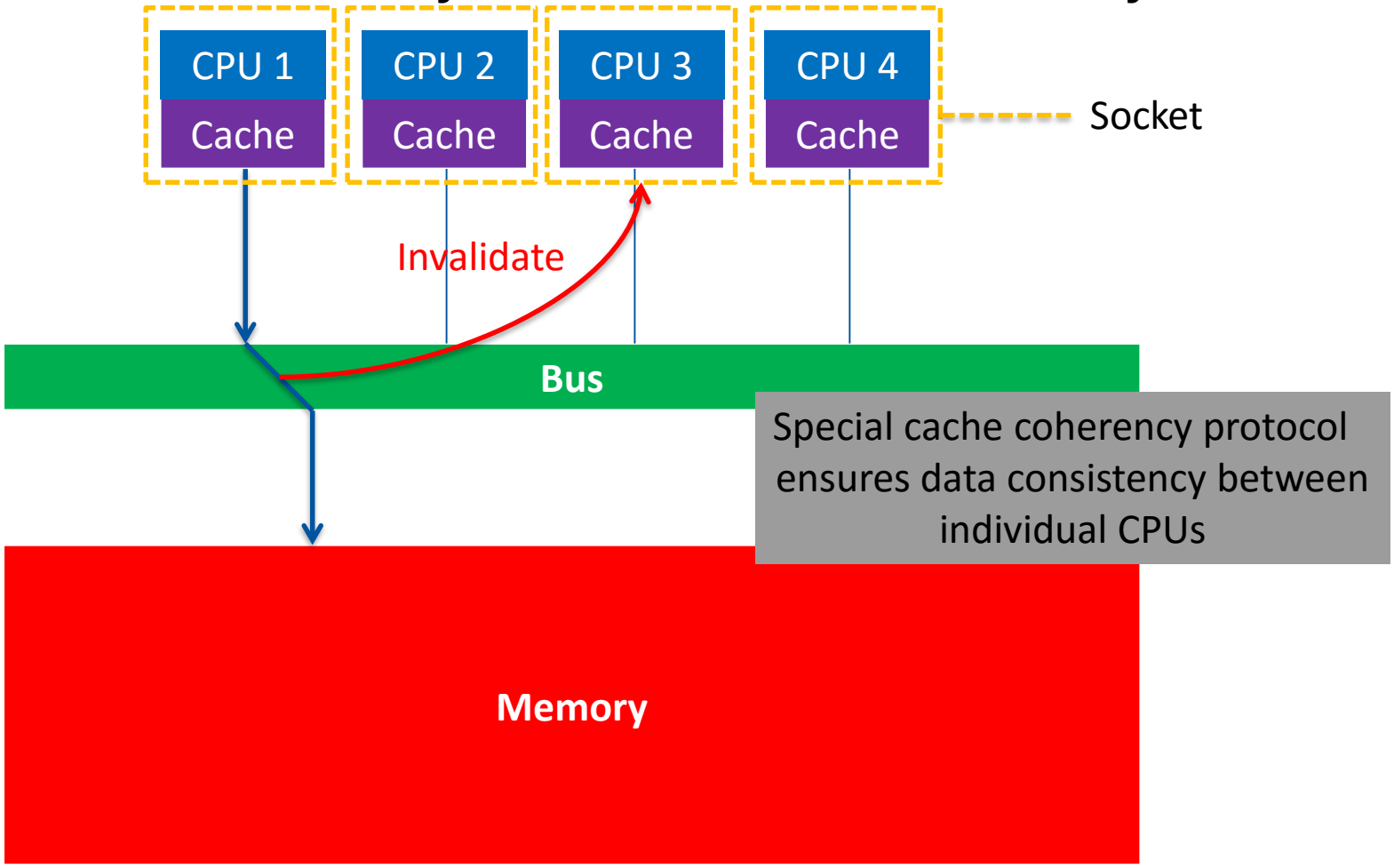
Further reading: Greg Bronevetsky, Bronis R. de Supinski: Complete Formal Specification of the OpenMP Memory Model. International Journal of Parallel Programming 35(4): 335-392 (2007

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → **Shared-memory computers**
     - → **UMA**
     - → ccNUMA
   - → Distributed-memory computers
   - → Networks
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

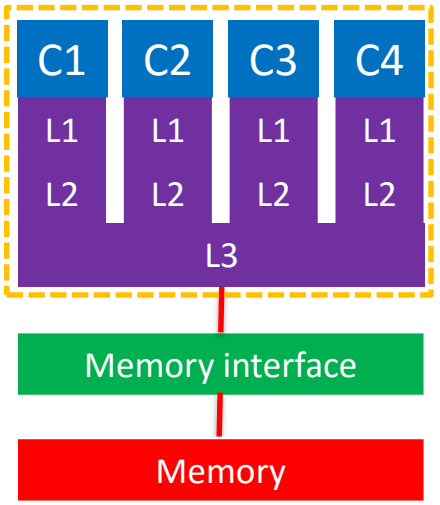
UMA

- **UMA Architecture: memory access over a shared memory bus**



Worst case

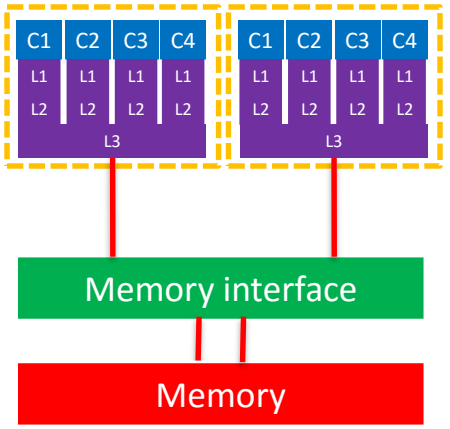
- **As there is no further definition on the memory bus in the UMA concept, in the worst case the bus provides serial access on the main memory.**
  - → Only one CPU can access the memory at a time
    - → Faster memory renders useless in this case
    - → Data access collisions occur frequently
  - → This concept is also present in a multicore chip:
  - → If multicores are arranged in an UMA constellation available bandwidth/peak Balance is reduced further



Best case

- **In the best case a memory crossbar switch provides separate data paths to memory for individual CPUs**
  - → Full memory bandwidth is accessible to every CPU
  - → Accesses only have to be stalled if accesses to the same memory module have to be synchronized

- **In real world configurations:**
  - → Some but not all CPUs at once can access memory concurrently. E.g. 2 of 8 cores →
  - → Programmer may notice that it is advantageous in some situations to spread the used core over more than one socket.



Advantages & disadvantages

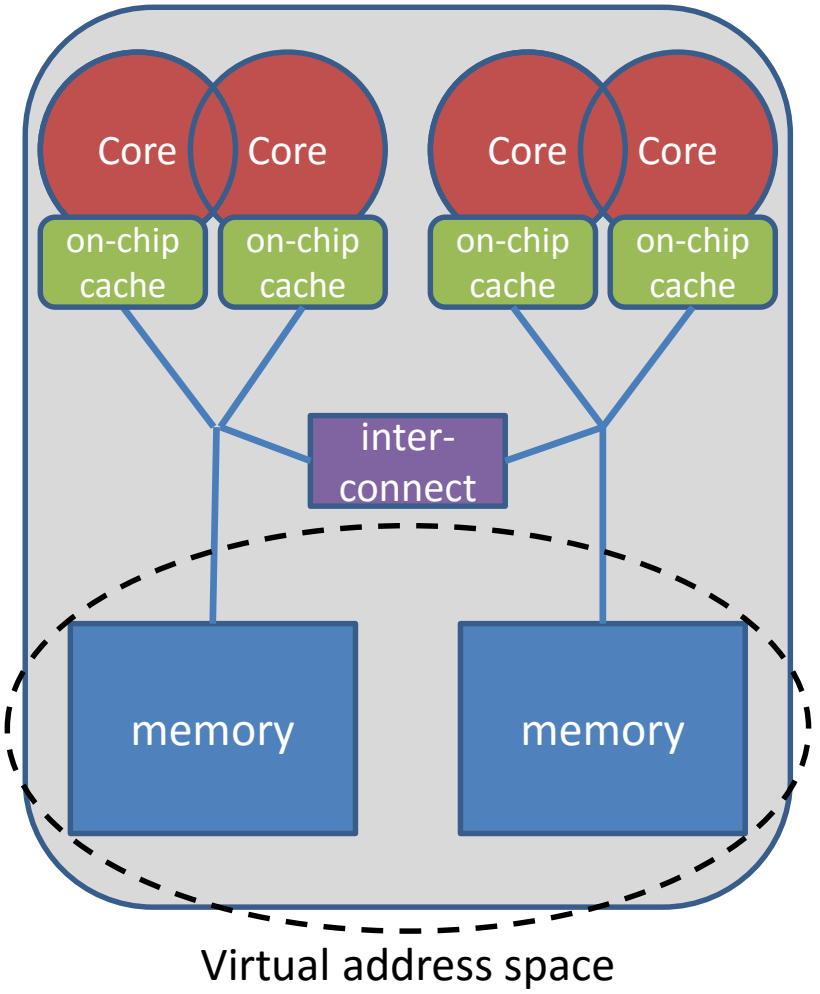
- **Advantages**
  - → Easy to optimize memory accesses as each CPU has the same path to memory
  - → Cache Coherence is easy to implement
- **Disadvantages**
  - → Memory bandwidth limits scalability in many scenarios
- **Examples for UMA architectures**
  - → NEC vector computers
  - → Most desktop computers

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → **Shared-memory computers**
     - → UMA
     - → **ccNUMA**
   - → Distributed-memory computers
   - → Networks
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

ccNUMA

- **Set of processors is organized inside a locality domain with a locally connected memory.**
  - → The memory of all locality domains is accessible over a shared virtual address space.
  - → Other locality domains are access over a interconnect, the local domain can be accessed very efficiently without resorting to a network of any kind
  - → Examples: the RWTH BCS cluster, Westmere Cluster, normally all modern multicore clusters

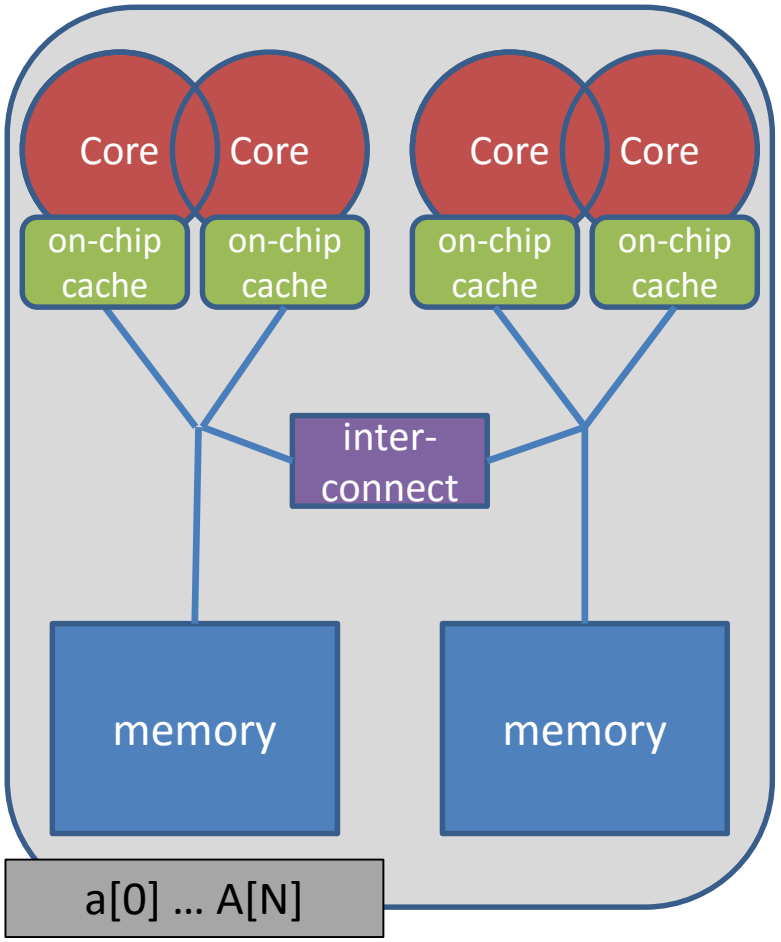


Data Distribution

- Memory is allocated on the NUMA node containing the core executing the initialization
- If not optimal, longer memory access times and hotspots

```
// Executed by thread 0
double* A;
A=(double*)malloc(N*
                  sizeof(double));


for(i=0; i < N; ++i)
{
  A[i] = 0.0;
}
```

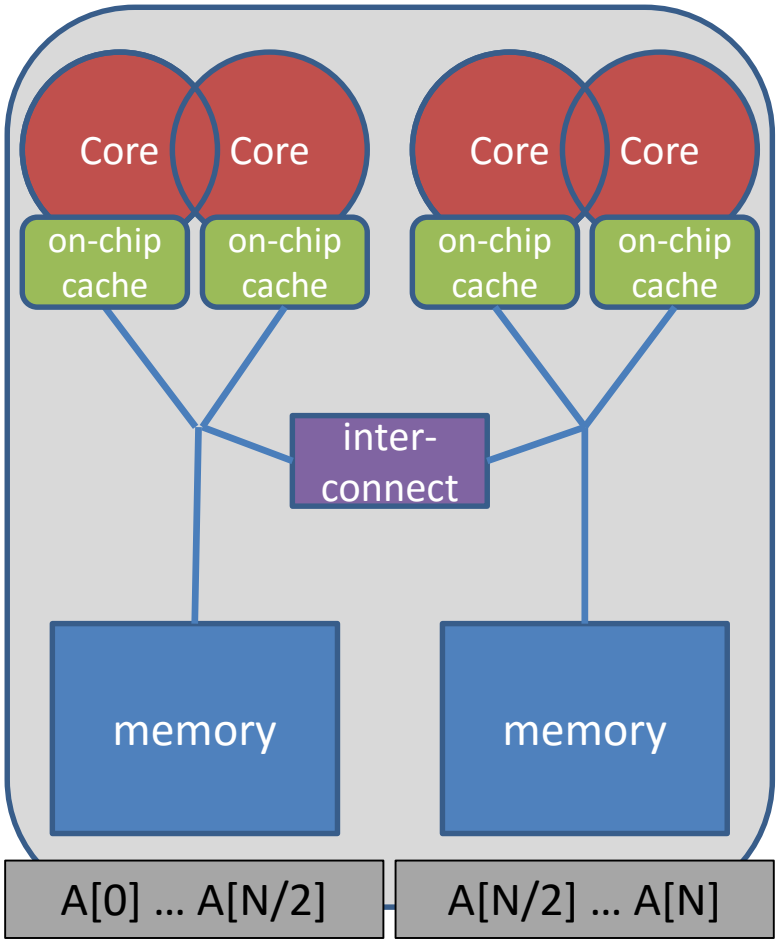


Data Distribution (cont.)

- “First Touch” placement policy
- Memory is allocated on the NUMA node containing the core executing the thread initializing the respective partition

```
// Executed by thread 0
double* A;
A=(double*)malloc(N*
            sizeof(double));

// Executed by multiple threads
for(i=0; i < N; ++i)
{
  A[i] = 0.0;
}
```



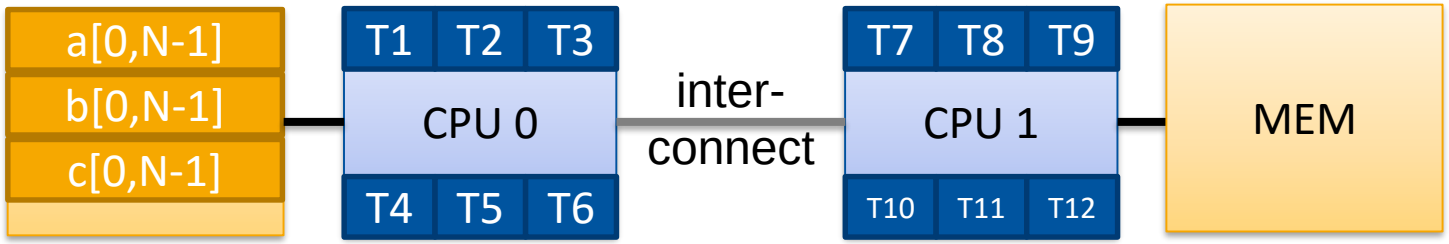
Data Distribution (cont.)

- **Stream example with and without parallel initialization.**

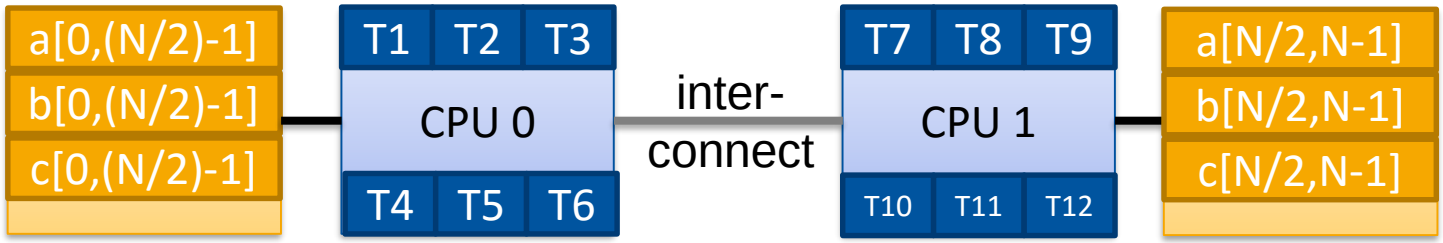
  → 2 socket sytem with Xeon X5675 processors, 12 threads

	copy	scale	add	triad
Serial init.	18.8 GB/s	18.5 GB/s	18.1 GB/s	18.2 GB/s
Parallel init.	41.3 GB/s	39.3 GB/s	40.3 GB/s	40.4 GB/s

Serial initialization:



Parallel initialization:



High-speed interconnect

- **Interconnection between sockets is realized over a special high-speed connection**
  - → AMD → HyperTransport (HT)
    - →Two unidirectional units for transport exist (one for each direction)
    - →Theoretical transfer rate of up to 25.6 GByte/s depending on clockrate, but overhead through additional packet information has to be substracted
  - → Intel → QuickPath (QPI)
    - →Two unidirectional units for transport exist (one for each direction)
    - →Depending on the clockrate of the QPI link, transfer rates of 9,6 – 12,8 GByte/s
  - → Other solutions exist

Advantages & disadvantages

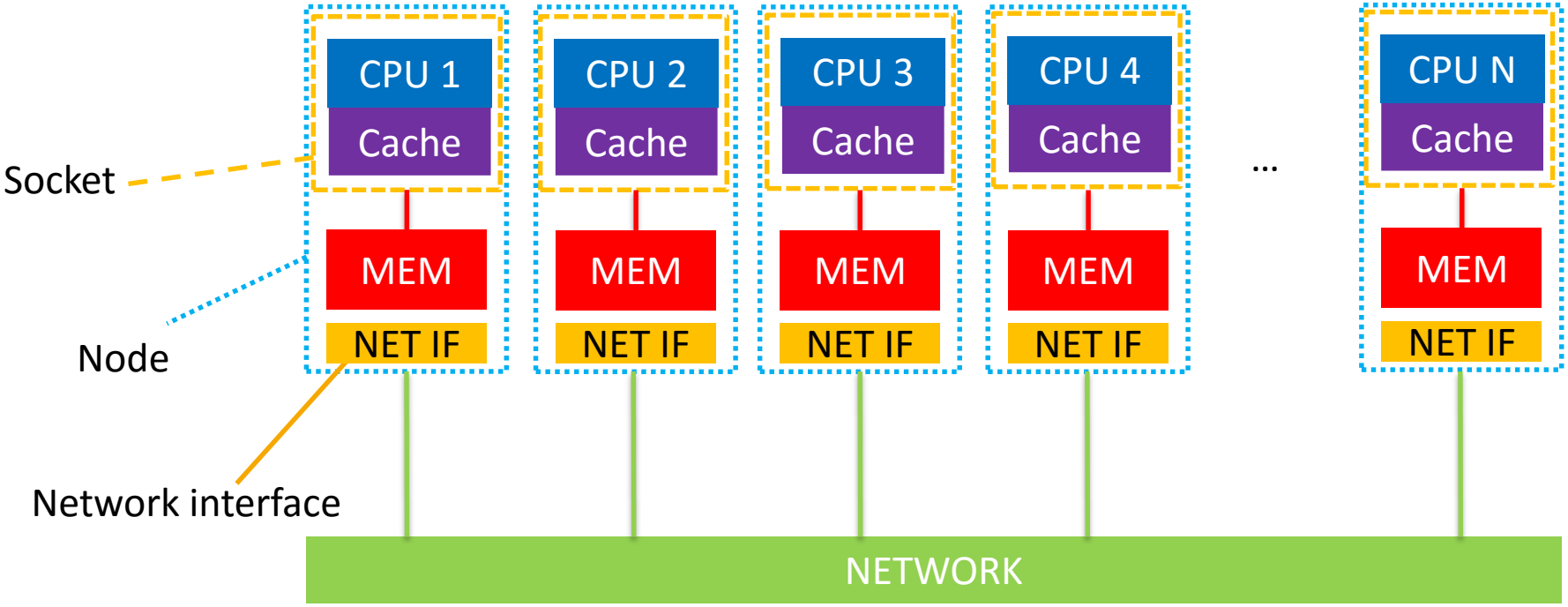
- **Advantages**
  - → Scalable in terms of memory bandwidth
  - → “Arbitrarily” large numbers of processors: There exist systems with over 1024 CPUs
- **Disadvantages**
  - → Efficient programming requires precautions with respect to local and remote memory, although all processors share one address space
    - → Anisotropic design
  - → Cache coherence is hard and expensive in implementation
    - → e.g. recent writes need invalidation and may consume a lot of the available bandwidth

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → **Distributed-memory computers**
     - → Networks
       - → Basic performance characteristics
       - → Network topologies
         - → Buses
         - → Ring & fully connected networks
         - → Switched & fat-tree networks
         - → Mesh networks
       - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

Distributed-memory computers

- **System where memory is distributed among “nodes”**
  - → No other node than the local one has direct access to the local memory

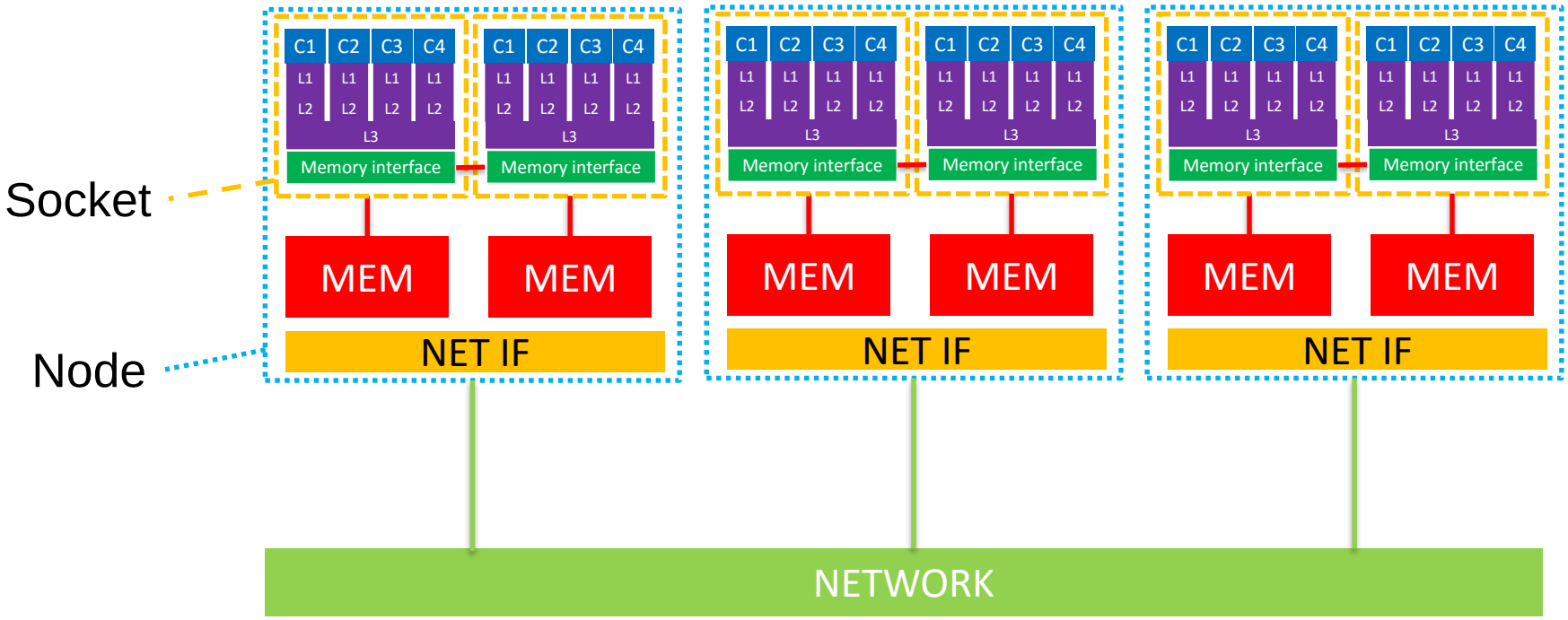


Distributed-memory computers

- **In a purely distributed-memory computers, each processor is connected to an exclusive local memory and a network interface**
  - → The combination of these is called “node”
- **A communication network connects all nodes**
- **The example on the previous slide is also called *No Remote Memory Access* (NORMA)**
- **Data is exchanged between nodes using the network (Message Passing)**
  - → MPI

Hybrid systems

- **All large-scale parallel computing systems are neither purely of the shared nor the distributed memory type. They are a mixture of both.**
  - → Increased anisotropy



Hybrid systems

- **The concept of *hybrid* parallel systems has clear advantages in terms of price**
  - → Much of the infrastructure can be shared
- **But: with more cores per node sharing a single network interface the performance may underlie limitations due to the reduced available network bandwidth per core**
  - → Efficient utilization of hybrid systems is per se unclear
  - → Highly dependent of application and system constellation
- **Hybrid systems can only be utilized to full extent with a mixture of programming paradigms**
  - → e.g. MPI + OpenMP

What you have learnt

- **Which taxonomy for parallel computing does exist?**
- **What can limit scalability?**
- **What does Amdahl's Law (strong scaling) say?**
- **What does Gustafson's Law (weak scaling) say?**
- **What is a multicore processors?**
  - → Why do we have multicore processors?
- **Which advantages do SMT have?**
- **What is a shared-memory computer?**
  - → What is the difference between UMA and ccNUMA?
- **What is a distributed-memory computer?**
  - → How do hybrid systems look like?

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → **Networks**
     - → **Basic performance characteristics**
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

Networks

- **Communication overhead can have significant impact on performance**
- **Network connects e.g.**
  - → Execution units
  - → Processors
  - → Nodes,…
- **Large variety of networking technologies exist with**
  - → Proprietary topology
  - → Open topology
- **Cheapest solution in clusters: Gigabit Ethernet**
- **Most commonly-used solution in clusters: InfiniBand**

Characteristics of point-to-point connections

- **Basis is the same idea as for memory accesses:**
  - → Evaluate network data transfer capabilities

- **A message of N bytes can be transferred in:**

$$T = T_L + \frac{N}{B}$$

T_L: Latency (time a 0-byte message transfer takes
B : Theoretical max. network bandwidth [MB/s]

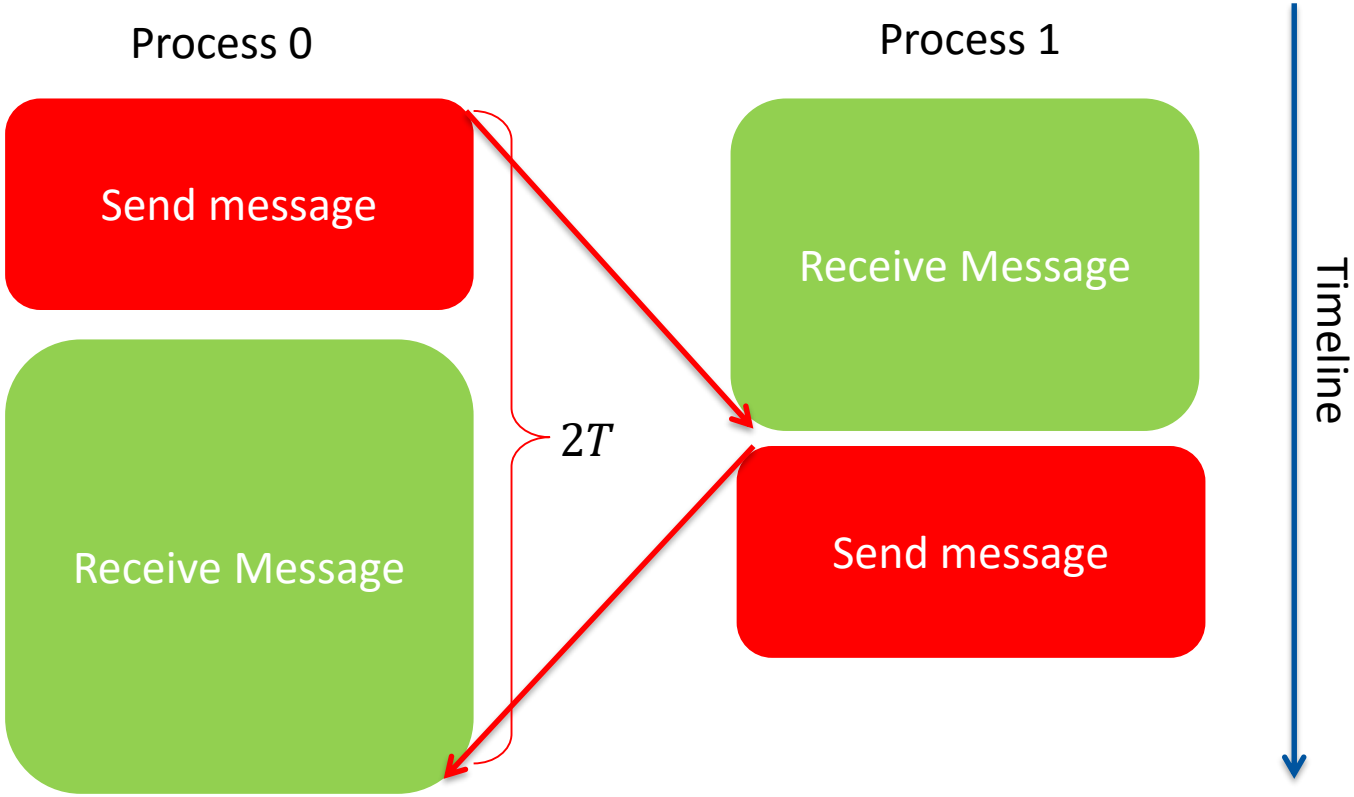
  - → T_L and B usually depend on the message length *N*

- **Effective bandwidth during the transfer of N bytes:**

$$B_{eff} = \frac{N}{T} = \frac{N}{T_L + \frac{N}{B}}$$

Ping Pong benchmark

- **Simplest case for network performance benchmarks: “Ping Pong”**
  - → Sends message of N bytes forth and back between 2 processors



Ping Pong Example

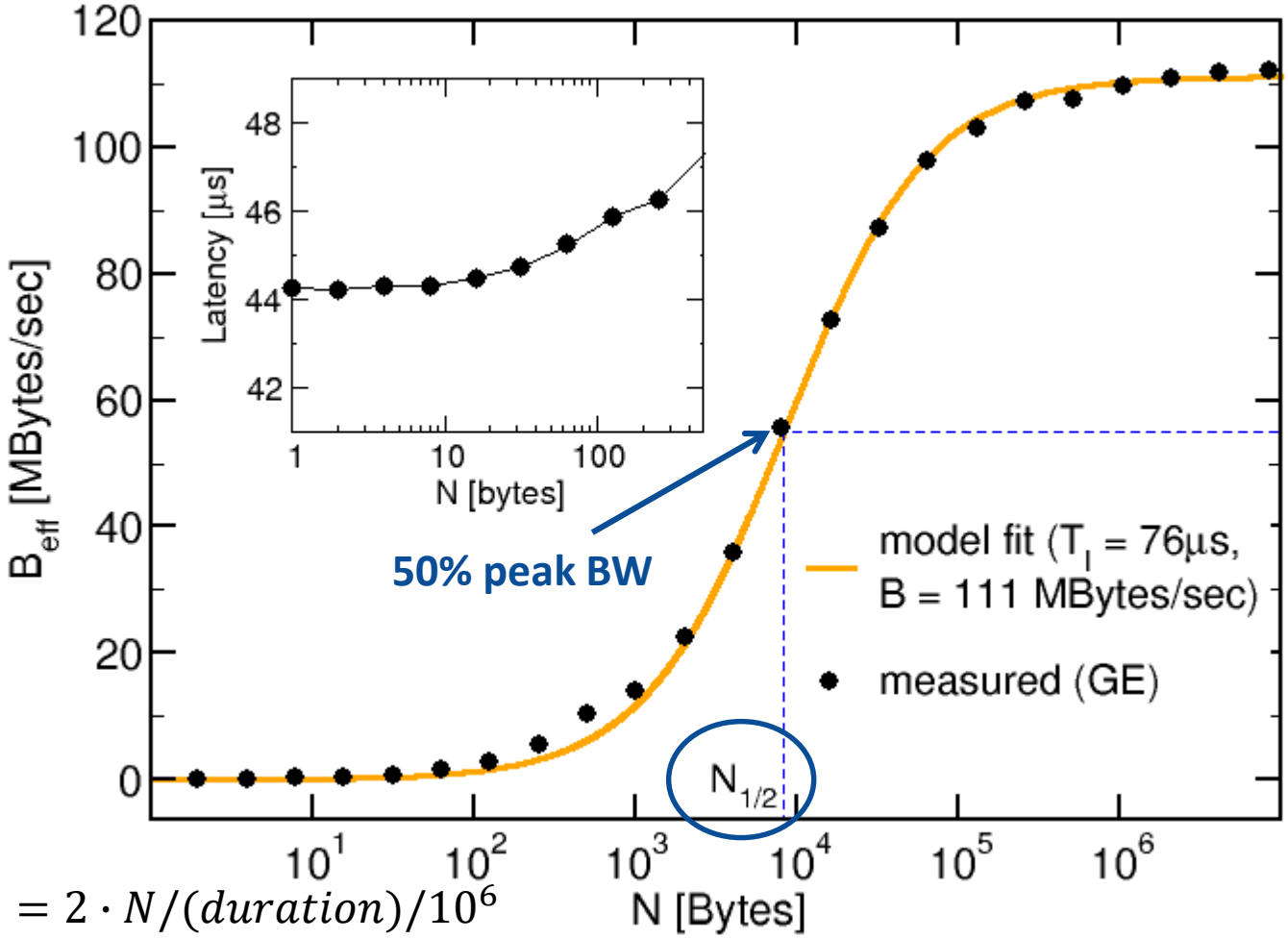
- **Pseudocode: reports bandwidth [MB/s] for different N**

```
myID = get_process_ID();
if( myID == 0 ) // process 0
{
  targetID = 1;
  start=get_walltime();
  send_message(buffer, N, targetID);
  receive_message(buffer, N, targetID);
  duration = get_walltime() – start;
  mbytes_per_sec = 2*N/(duration)/1E6; // MB/s rate
}
else // process 1
{
  targetID = 0;
  receive_message(buffer, N, targetID);
  send_message(buffer, N, targetID);
}
```

Process 0:
→ Send & receive
Process 1:
→ Receive & send

Model fit example (Ethernet)

- **Model fit measured on a Gigabit Ethernet**



$$B_{eff} = 2 \cdot N/(duration)/10^6$$

$N_{1/2}$:
Message length at which half of the saturation bandwidth is reached

Model covers (just) main features:
Small message size
- Low BW (latency dominates)

Large message size
- High BW (latency plays no role)

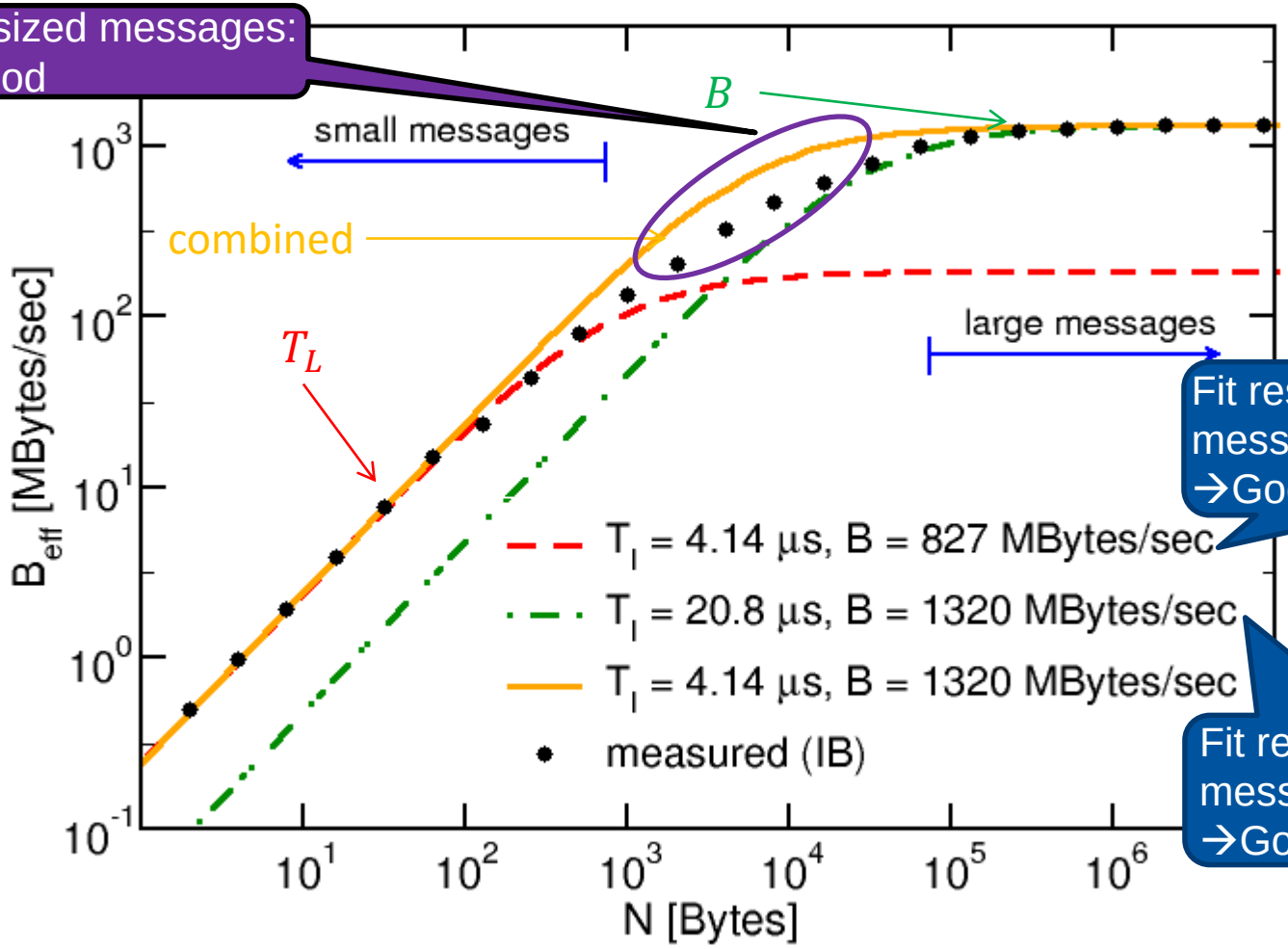
Prof. Dr. G. Wellein, Dr. G. Hager, M. Kreutzer Uni Erlangen-Nürnberg

Quality of the model

- **Main features are covered, but only one qualitative fit of the measurements to the model**
  - → 44 µs measured, 76 µs assumed
  - → Reality shows that model is not too good
- **General causes for high(er) latencies**
  - → Network protocol overhead (message headers, etc.)
  - → TCP defines minimum message size (always sent)
  - → Depending on protocol a transfer or receive is a complex task that passes through multiple software layers (e.g. OSI stack)
  - → Standard PC hardware (ethernet) is not optimized for low-latency I/O
  - → With increasing message size, the software layers may switch to different transmit strategies

Model fit example (InfiniBand)

- **Model fit measurements on DDR InfiniBand (DDR-IB) network**



Latency effects play a role

- **Network interconnects exist with an access speed similar to local memory**
- **But, many applications work with access speeds where latency effects play a role**

  → $N_{\frac{1}{2}}$ is determined to express the extent of this problem

  → $N_{\frac{1}{2}}$: N where $B_{eff} = \frac{B}{2}$

  → Here: $N_{\frac{1}{2}} = B\,T_L$

$$B_{eff} = \frac{N}{T_L + \frac{N}{B}}$$

  → Which effect does an increase (by factor β) in max. network bandwidth have?

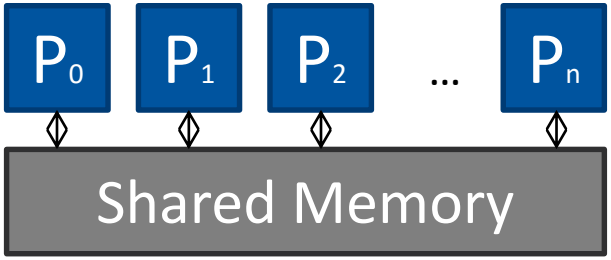
  → Is it beneficial for all messages?

  → Improvement in effective BW at message size N: $\frac{B_{eff}(\beta B, T_L)}{B_{eff}(B, T_L)} = \frac{1 + \frac{N}{N_{\frac{1}{2}}}}{1 + \frac{N}{\beta N_{\frac{1}{2}}}}$

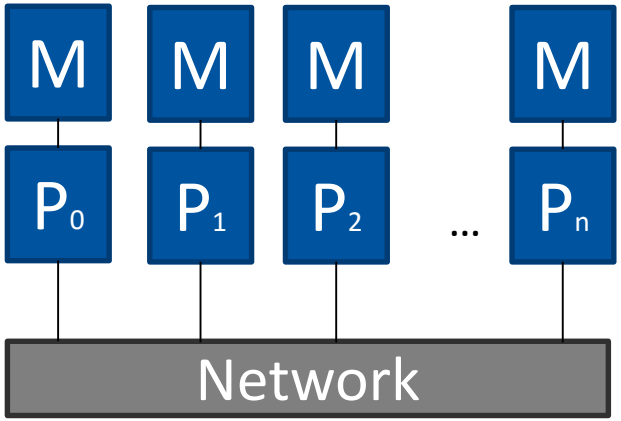
  → E.g. $N = N_{\frac{1}{2}}$, $\beta = 2$: improvement of only 33%

LogP Model

- **PRAM model: abstract shared memory machine**
  - → neglects practical issues as synchronization and communication



- **LogP model: machine connected by a network**
  - → attempts to capture such caracteristics



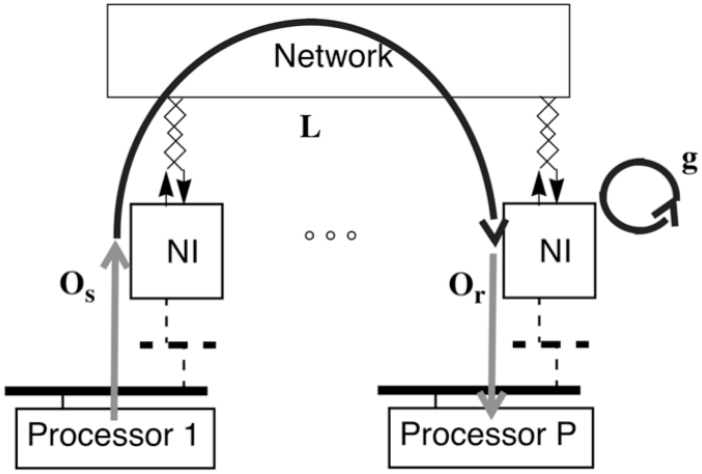
Sources: [1] Culler, Karp, Patterson, Sahay, Schauser, Santos, Subramonian, and von Eicken. 1993. LogP: towards a realistic model of parallel computation. *SIGPLAN Not.* 28, 7 (July 1993)

LogP Model

- **Models for the scaling behavior of applications on parallel computers are of increasing importance**
  - → Usually depend on only a few of the machine‘s individual characteristics
  - → Usually applicable to plenty of different parallel computers

- **LogP model focuses on 4 parameters that parallel algorithms usually adapt to in order to maximize efficiency (abstractly)**
  - → Computing bandwidth
  - → Communication bandwidth
  - → Communication delay (latency)
  - → Efficiency of coupling computation and communication

LogP Model

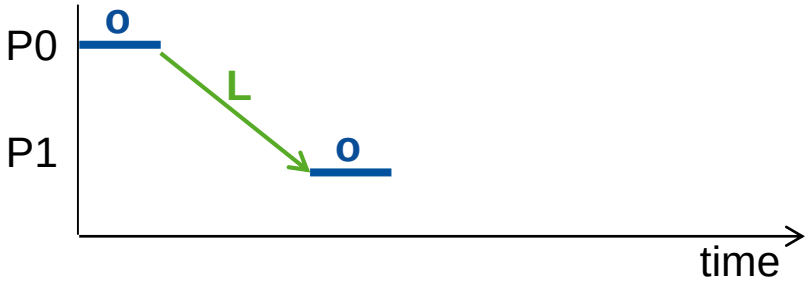


Source: MikeDahlin, University of Texas at Austin

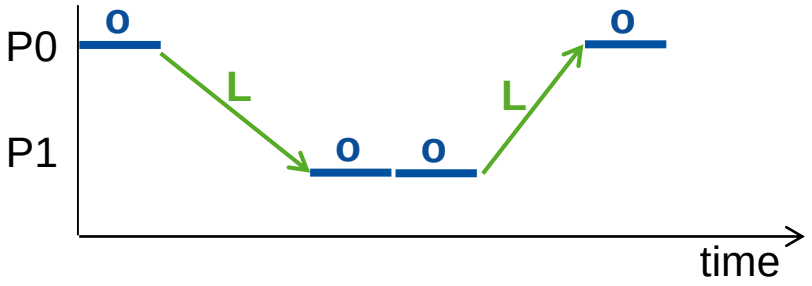
- **L**
  - → Upper boundary on latency which is incurred during a message communication
- **o**
  - → Overhead = length of time that a processor is involved in the transfer/ reception of a message
  - → During this time the processor cannot perform any other operations
- **g**
  - → Gap = minimum time interval between consecutive message transfers or consecutive message receptions at a processor
  - → $\frac{1}{g}$ → available communication bandwidth per core
- **P**
  - → The number of processors/ memory modules

LogP Model example

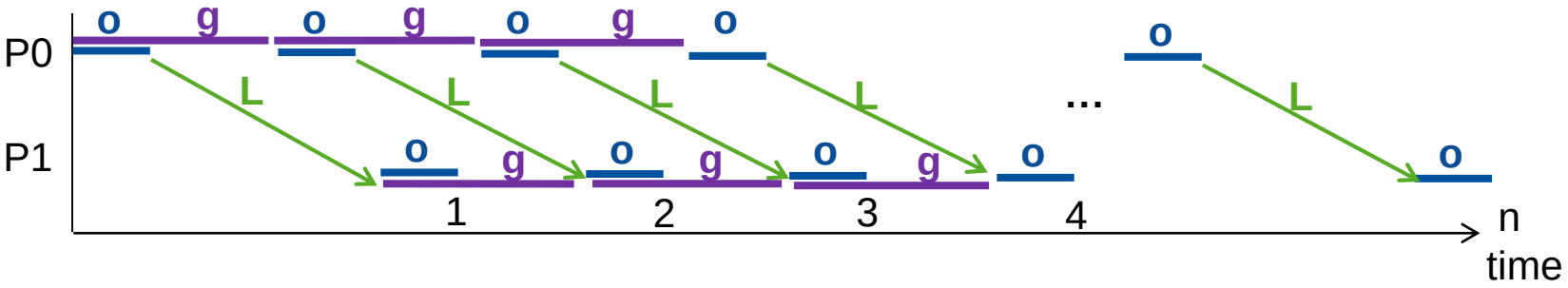
- **Sending a single message**
  - → T = 2o + L



- **Ping Pong Round-Trip**
  - → T = 4o + 2L



- **Transmitting n messages**
  - → T(n) = L + (n-1) * max(g,o) + 2o

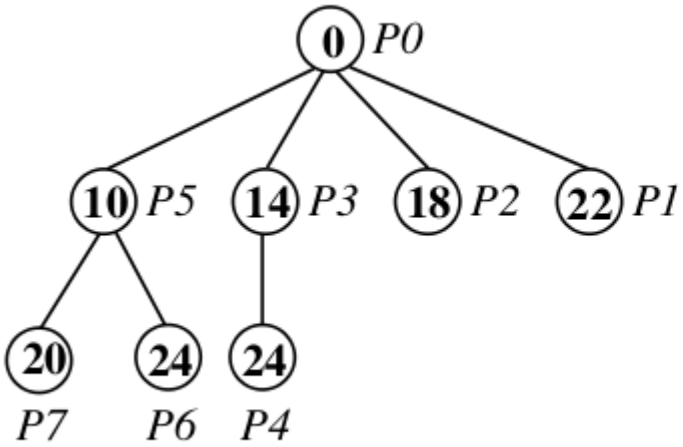


Source: Torsten Hoefler, Networking Basics

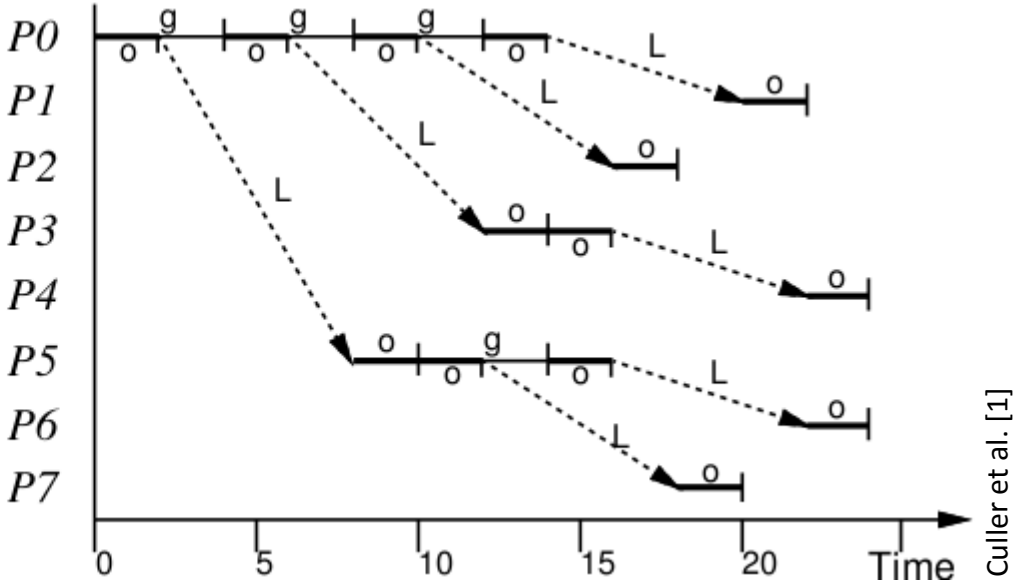
LogP Model example

- **(Optimal) broadcast to P-1 processes**
  - → Each process who received the value sends it on
  - → Each process receives exactly once
- → **Determines max number of procs that can be reached in time t: P(t)**

Just to give an expression that it can be really complicated



P=8, L=6, g=4, o=2



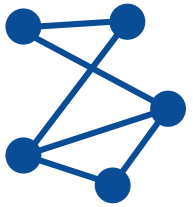
Culler et al. [1]

- → P(t) by generalized Fibonacci recurrence (assuming o>g) $P(t) = \begin{cases} 1 & t < 2o + L \\ P(t-o) + P(t-L-2o) & otherwise \end{cases}$
- → Bounded by [2]: $2^{\left\lfloor \frac{t}{L+2o} \right\rfloor} \leq P(t) \leq 2^{\left\lfloor \frac{t}{o} \right\rfloor}$

Sources: [1] Culler, Karp, Patterson, Sahay, Schauser, Santos, Subramonian, and von Eicken. 1993. LogP: towards a realistic model of parallel computation. *SIGPLAN Not.* 28, 7 (July 1993)
[2] Hoefler et al.: "Scalable Communication Protocols for Dynamic Sparse Data Exchange"
Further reading: Alexandrov, Ionescu, Schauser, and Scheiman. 1995. LogGP: incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation (SPAA '95). ACM, New York, USA, 95-105.

Graph theory (recap)

- **Static interconnection networks: describable as undirected graphs**
  - → Vertices (nodes) = processors/ communication devices
  - → Edges = interconnections



- **Characteristics for cost & performance of interconnection networks**
  - → **Bisection (band)width** b: min number of edges that have to be removed from the network to partition the network into 2 equal halves
  - → **Diameter** dm: max distance between any 2 processors
    - → Distance between 2 processors: min number of edges between them, i.e. shortest path in the graph between the two nodes/ processors
  - → **Edge connectivity** e: min number of edges that must be removed from the network to break it into 2 disconnected networks
  - → **Node connectivity** Δ: max number of edges in the graph

Bisection Bandwidth

- **“Ping Pong” benchmark cannot pinpoint global saturation effects**
  - → Underlying network may not be completely non-blocking: sum over all effective bandwidths for all point-to-point connections < theoretical limit
- **To quantify maximum aggregated communication capacity: *Bisection bandwidth (B_b)***
  - → = Sum of the bandwidths of the minimal number of connections cut when splitting the system into two equal sized parts
  - → Hybrid systems, the more meaningful metric is: available bandwidth per core, i.e. the bisection bandwidth divided by the overall number of cores: B_b/N

→ **How much data can be send through the network?**

- **Intended to give a lower bound for the available bandwidth**
  - → Definition: bandwidth for the worst case partitioning

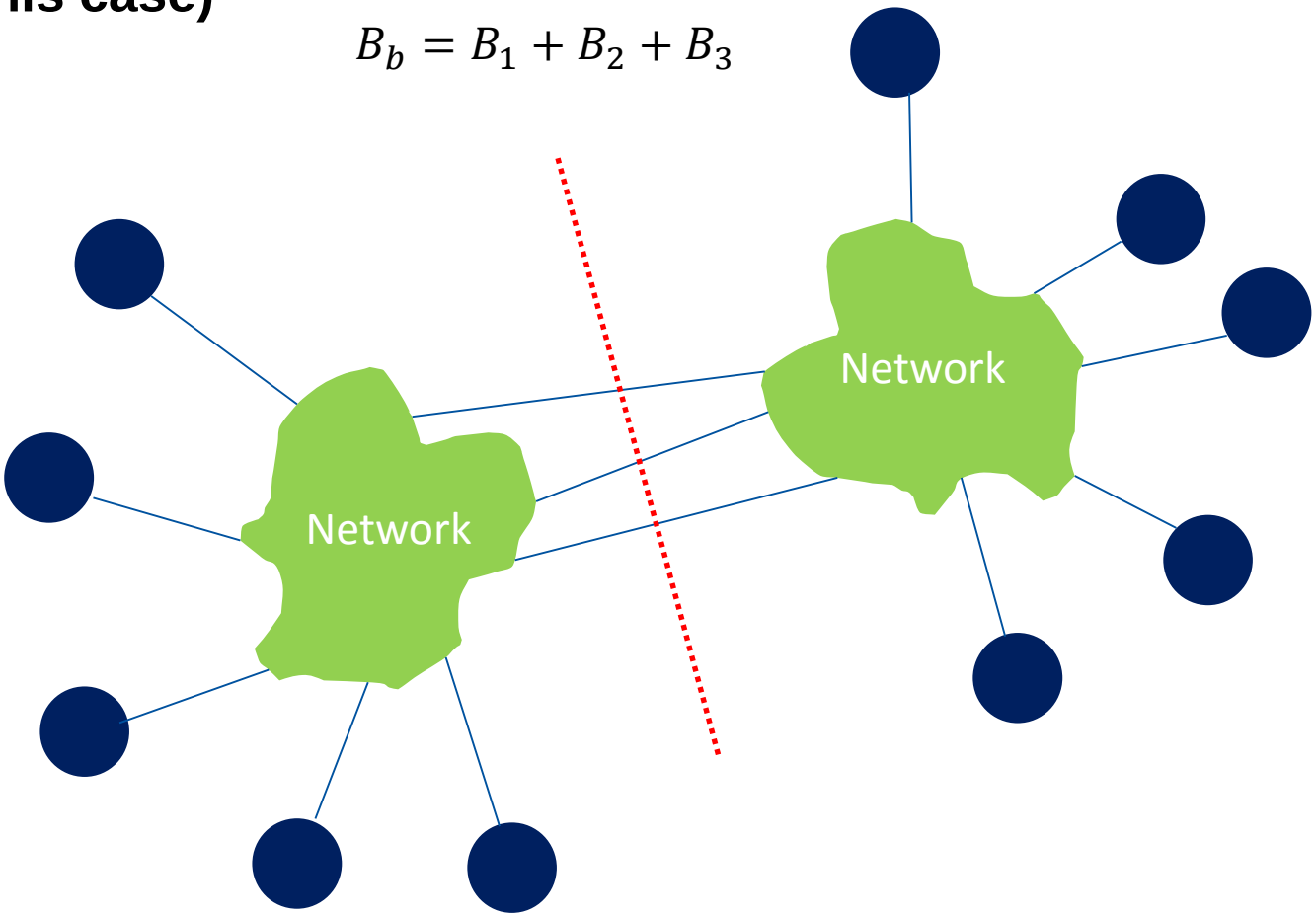
Bisection Bandwidth

- **Bisection Bandwidth depends on:**
  - → Bandwidth per link
  - → Network topology of the system
  - → Uni/Bi directional
- **Full bisection bandwidth (FBB)**
  - → Any two halves can communicate at full speed with each other
    - → Important for global communication

Bisection Bandwidth example

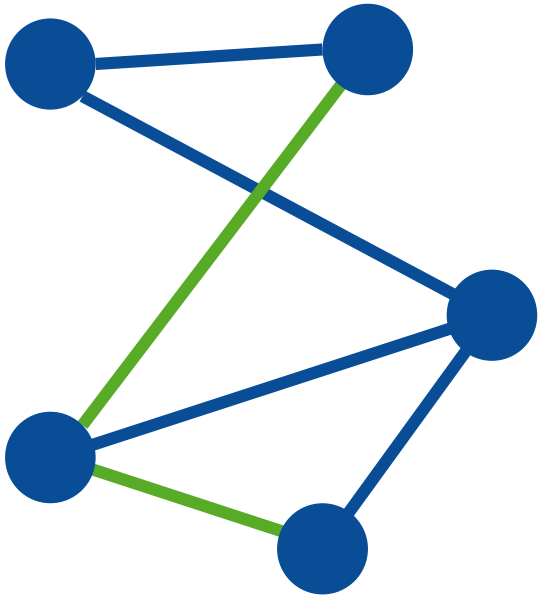
- **Bisection bandwidth: Sum of cut connections bandwidths (3 in this case)**

$$B_b = B_1 + B_2 + B_3$$



Diameter

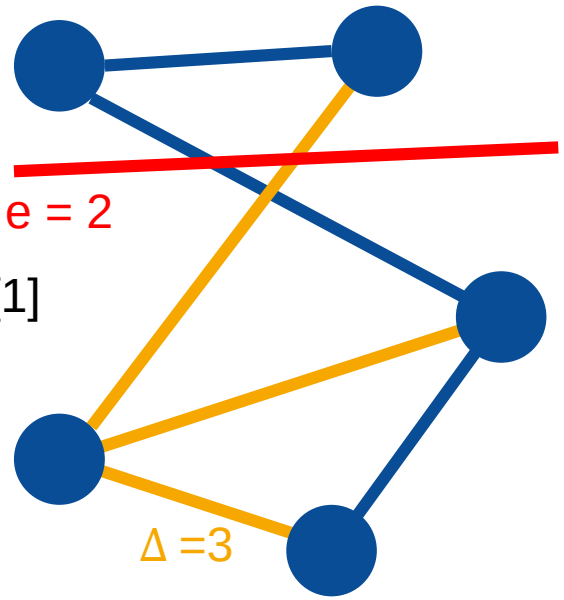
- **= Max distance between any 2 processors**
- **Indicates**
  - → Maximum transmission delay
  - → Maximum power consumption to transmit a packet
  - → Rough cost of the interconnection network
- **Average distance matters too**
  - → Average path length for all node pairs
  - → Average delay and power consumption
- **Here, focus on diameter for simplicity**



dm = 2

Edge/ node connectivity

- **Edge connectivity e**
  - → Min number of edges that must be removed from the network to break it into 2 disconnected networks
  - → Metric for reliability of network
    - → Larger edge connectivity → locally more reliable [1]



- **Node connectivity**
  - → Actually the max degree of the graph (network) $\Delta(G)$
    - → Max over the degrees of all vertices
  - → Gives the number of interconnect ports of a node (to buy)

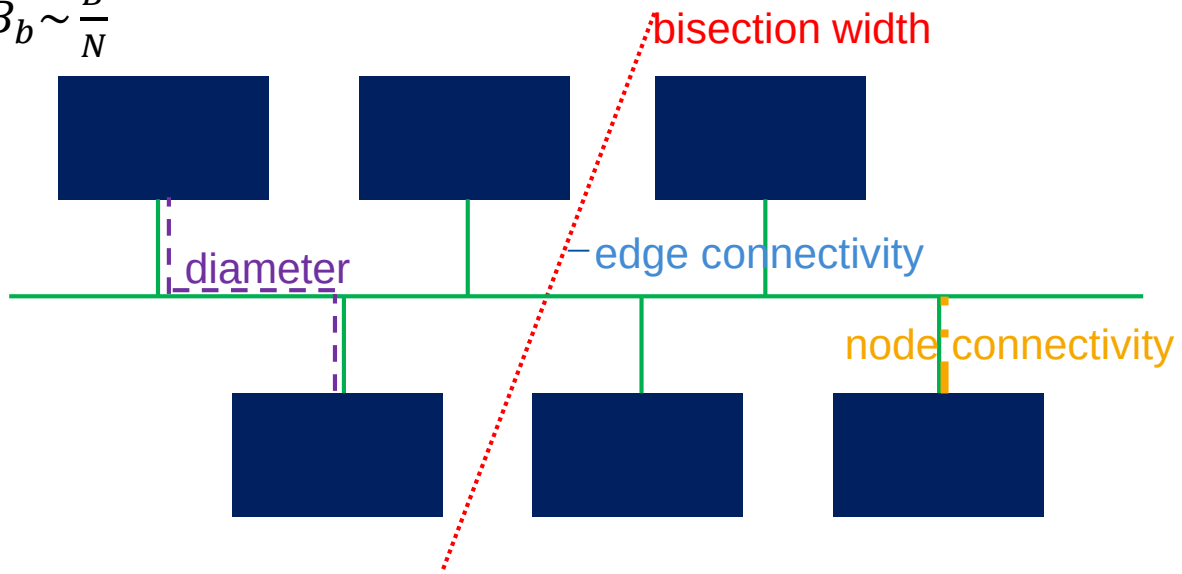
Source: [1] Bauer et al: Combinatorial optimization problems in the analysis and design of probabilistic networks, 2006

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → **Networks**
     - → Basic performance characteristics
     - → **Network topologies**
       - → **Buses**
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

Buses

- **Bus = shared medium (by multiple communication devices are)**
  - → Usable by exactly one communicating device at a time
- **Bisection bandwidth B_b: B (independent of #devices)**
  - → If all devices use bus: $B_b \sim \frac{B}{N}$
- **Diameter: 1**
- **Edge connectivity: 1**
- **Node connectivity: 1**



- **Examples: PCI (Peripheral Component Interconnect), some multicore designs use buses as a interconnect for separate CPU chips to memory**

N: #processing devices
B: connection bandwidth

Advantages & disadvantages

- **Easy to implement**
- **Lowest latency at small utilization**
- **Ready-made hardware components exist that fulfill the necessary protocols i.e. collision detection**

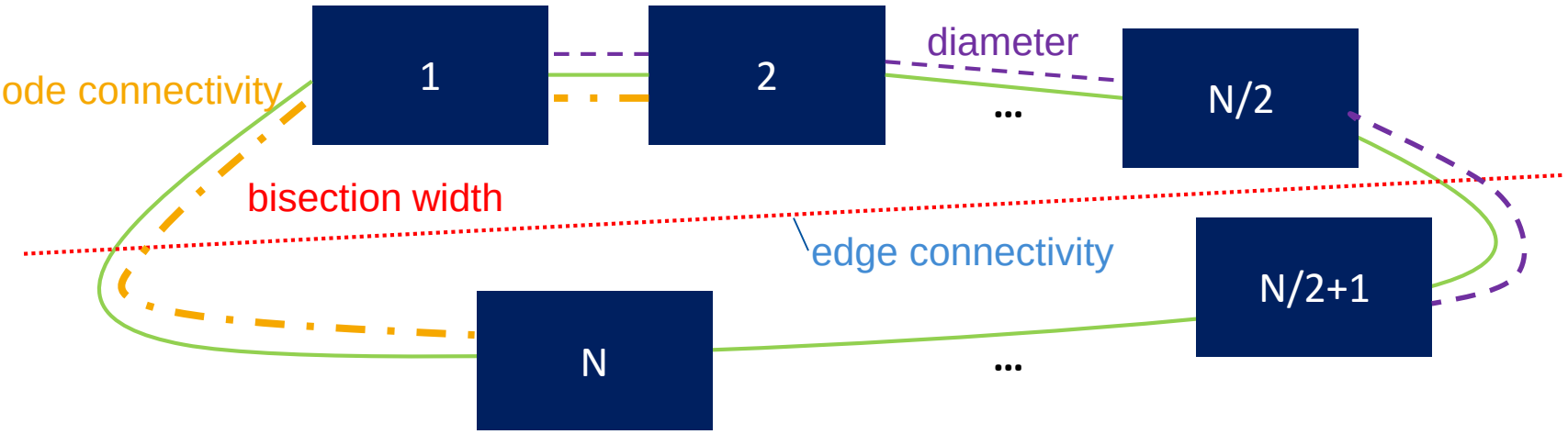
- **Most important drawback: buses are blocking**
- **All devices share the available bandwidth**
  - → The more devices are connected, the lower the average bandwidth
  - → Buses are prone to failure
    - → Local technical problems can easily affect the communication between all communication devices

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → **Networks**
     - → Basic performance characteristics
     - → **Network topologies**
       - → Buses
       - → **Ring & fully connected networks**
       - → Switched & fat-tree networks
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

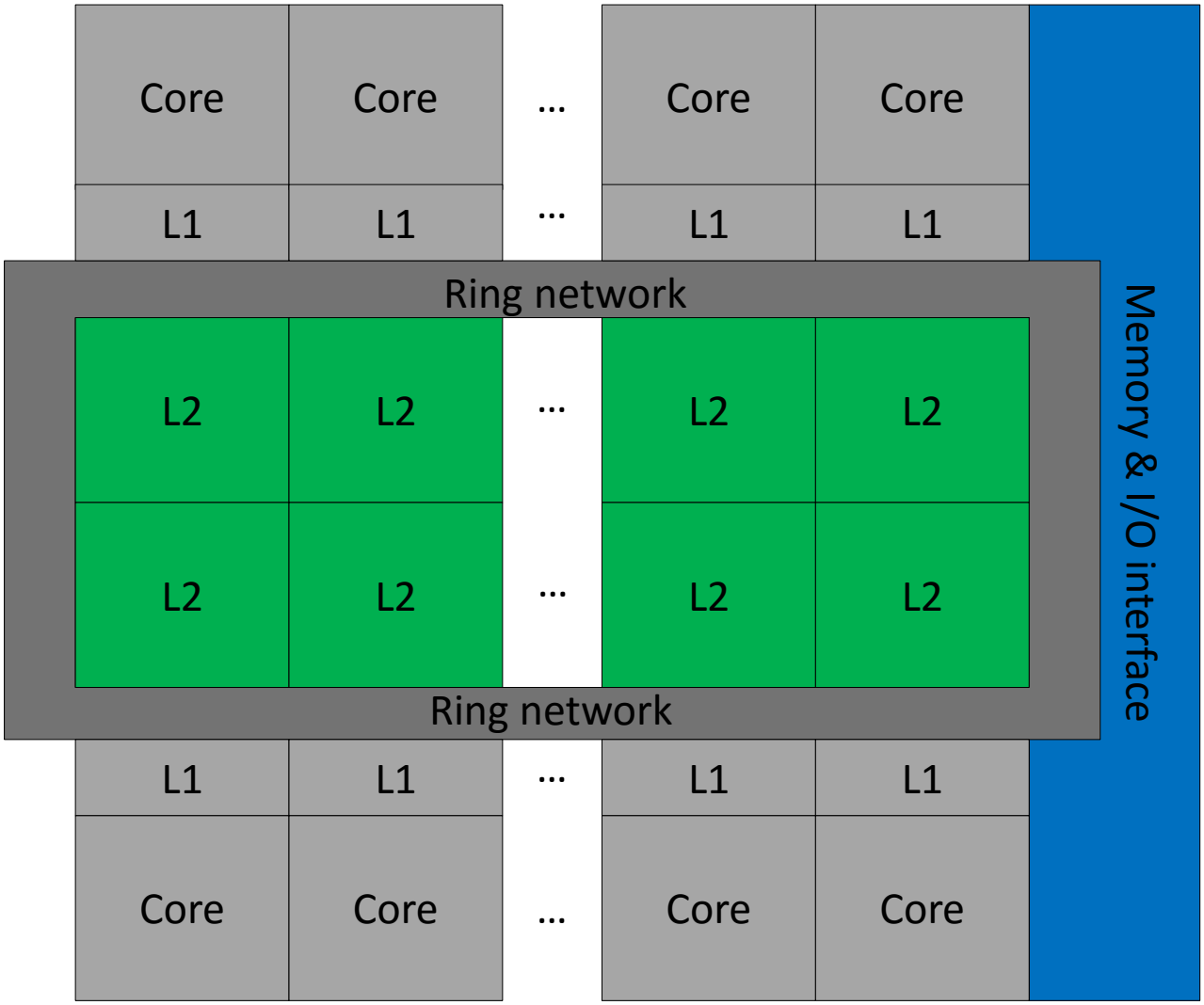
Ring network

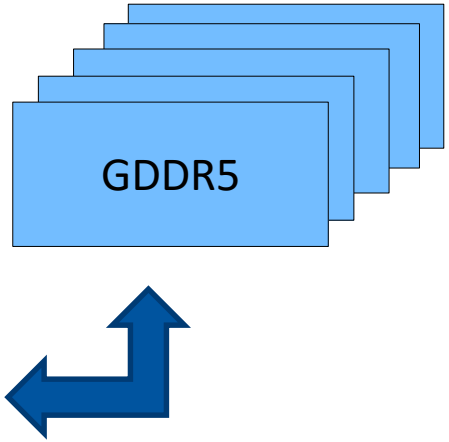
- **Diameter:** $\left\lfloor \frac{N}{2} \right\rfloor$
- **Bisection bandwidth: 2B**
- **Edge connectivity: 2**
- **Node connectivity: 2**



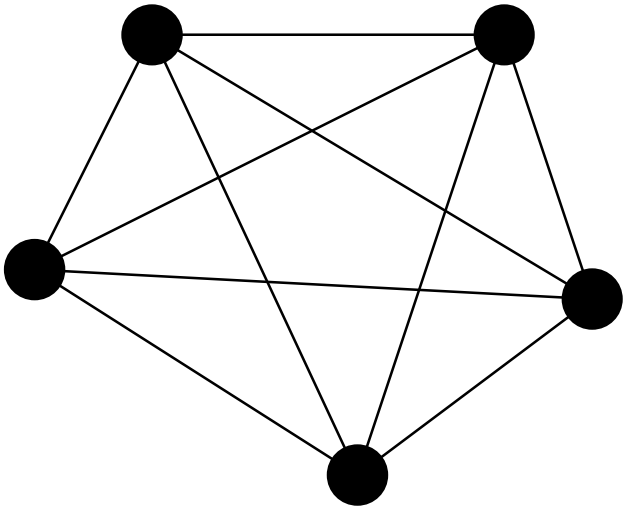
N: #processing devices
B: connection bandwidth

Example for a Ring network: Intel Phi Cache connection





Fully connected network



- **Values for diameter, bisection bandwidth & edge connectivity are left for the exercise**

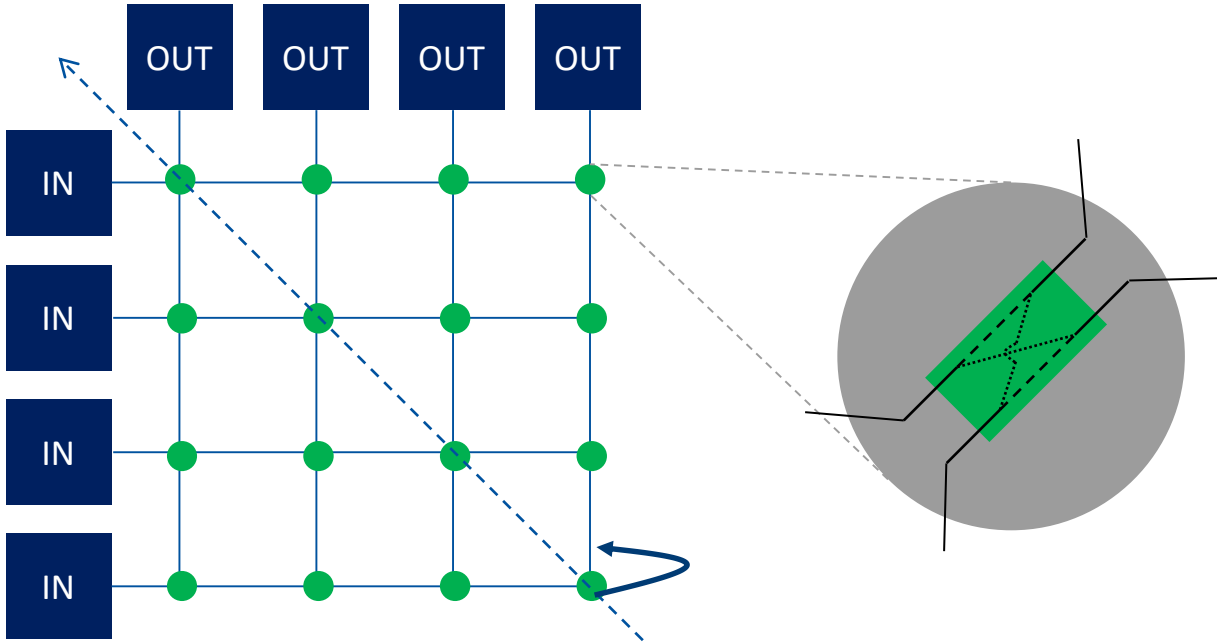
Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → **Networks**
     - → Basic performance characteristics
     - → **Network topologies**
       - → Buses
       - → Ring & fully connected networks
       - → **Switched & fat-tree networks**
       - → Mesh networks
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

Switched & fat-tree networks

- **Commonly the networks of todays standard clusters are of the switched type**
  - → Compute nodes are assigned into groups
  - → Groups are connected to a single switch (“leaf switches”)
    - →Switch is usually a non-blocking crossbar-switch
- **Switches are connected with each other using**
  - → a switch hierarchy (“spine switches”) or
  - → directly
- **“Distance” between any two devices: heterogeneous**
- **Example: Diameter of a bus is always 1**

Non blocking crossbar



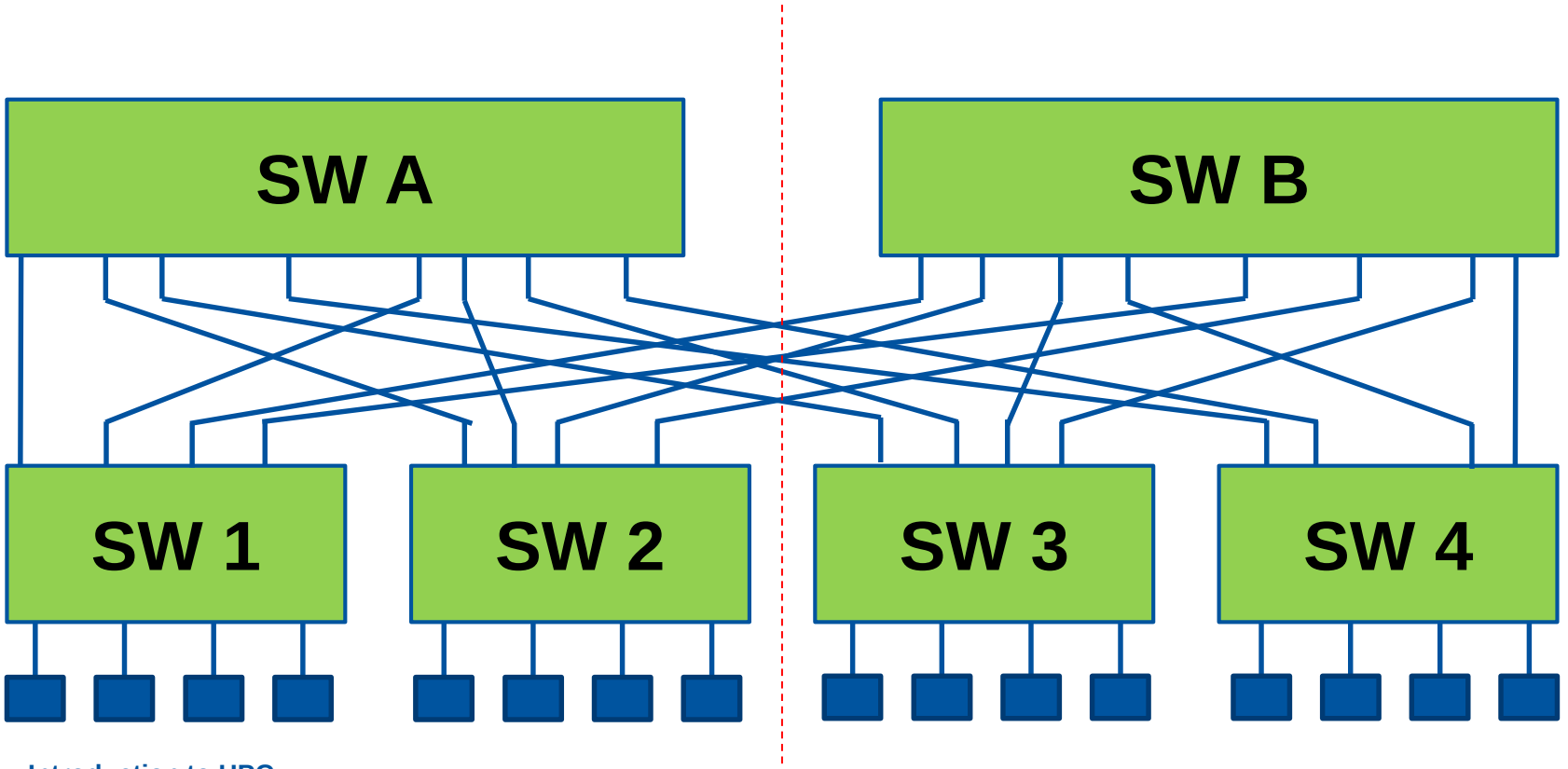
- Forms non-blocking connections between a number of input and a number of output elements
- Can be used e.g. as a 4-port non-blocking switch if folded diagonal

Non blocking crossbar

- **Crossbars can be used directly as interconnects in computer systems**
  - → Example: Scalable UMA memory access
  - → Hitachi SR8000
- **These switches can be used to cascade tree hierarchies (next slide)**

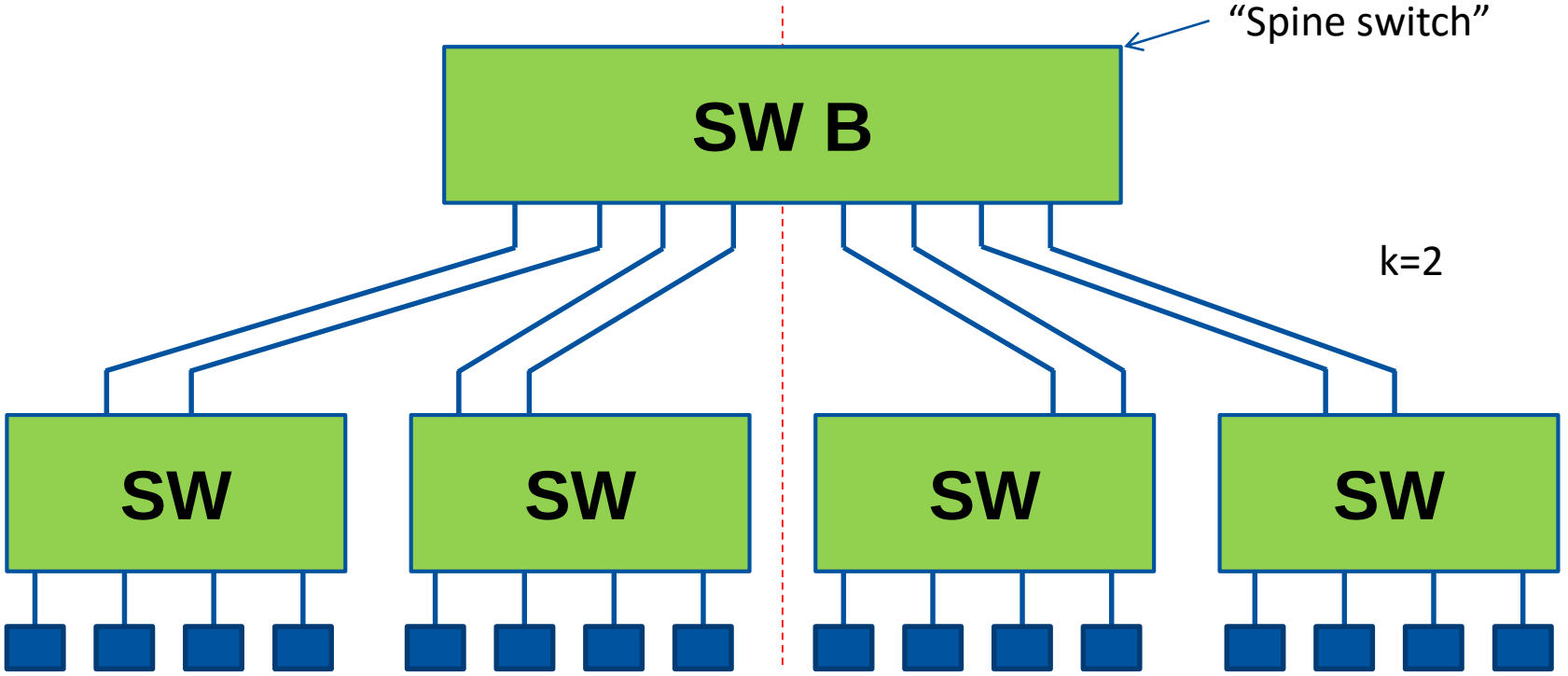
“Fully non blocking”

- N/2 end to end connections, full bandwidth: $B_b = B\frac{N}{2}$
- $\frac{B_b}{N} = const = \frac{B}{2}$



“Oversubscribed”

- **Single “spine” switch**
  - → Does not support N/2 full end-to-end connections
  - → $\frac{B_b}{N} = const = \frac{B}{2k}$, where k is the over subscription factor

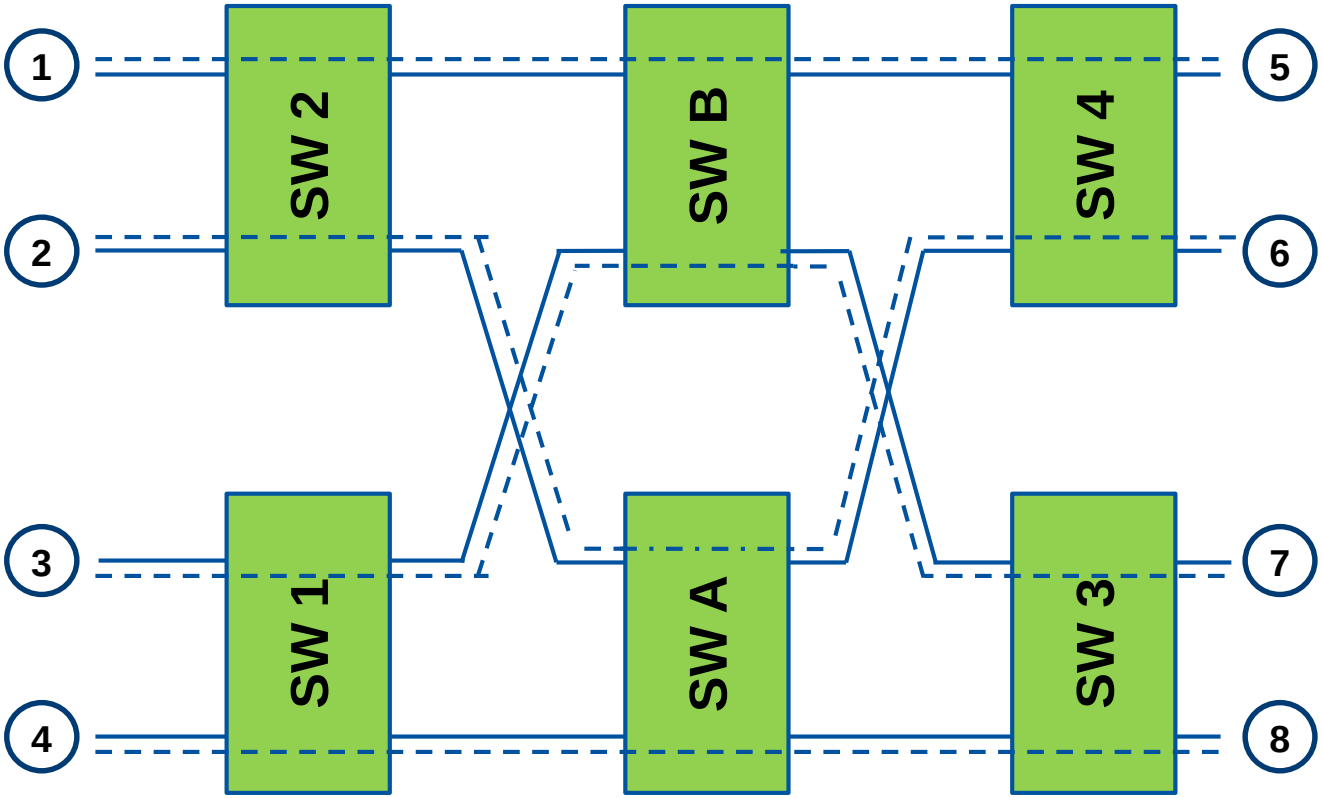


“Oversubscribed”

- **Network infrastructure in “oversubscribed” type networks must be capable of “intelligent” routing**
  - → If that is not possible, some node-to-node connections may be faster than others
- **Maximum latency depends only on: #layers in switch hierarchy**
- **But, bottlenecks may still exist**
  - → If connections between compute elements are “hardwired” (see next slide)

Collisions with static routing

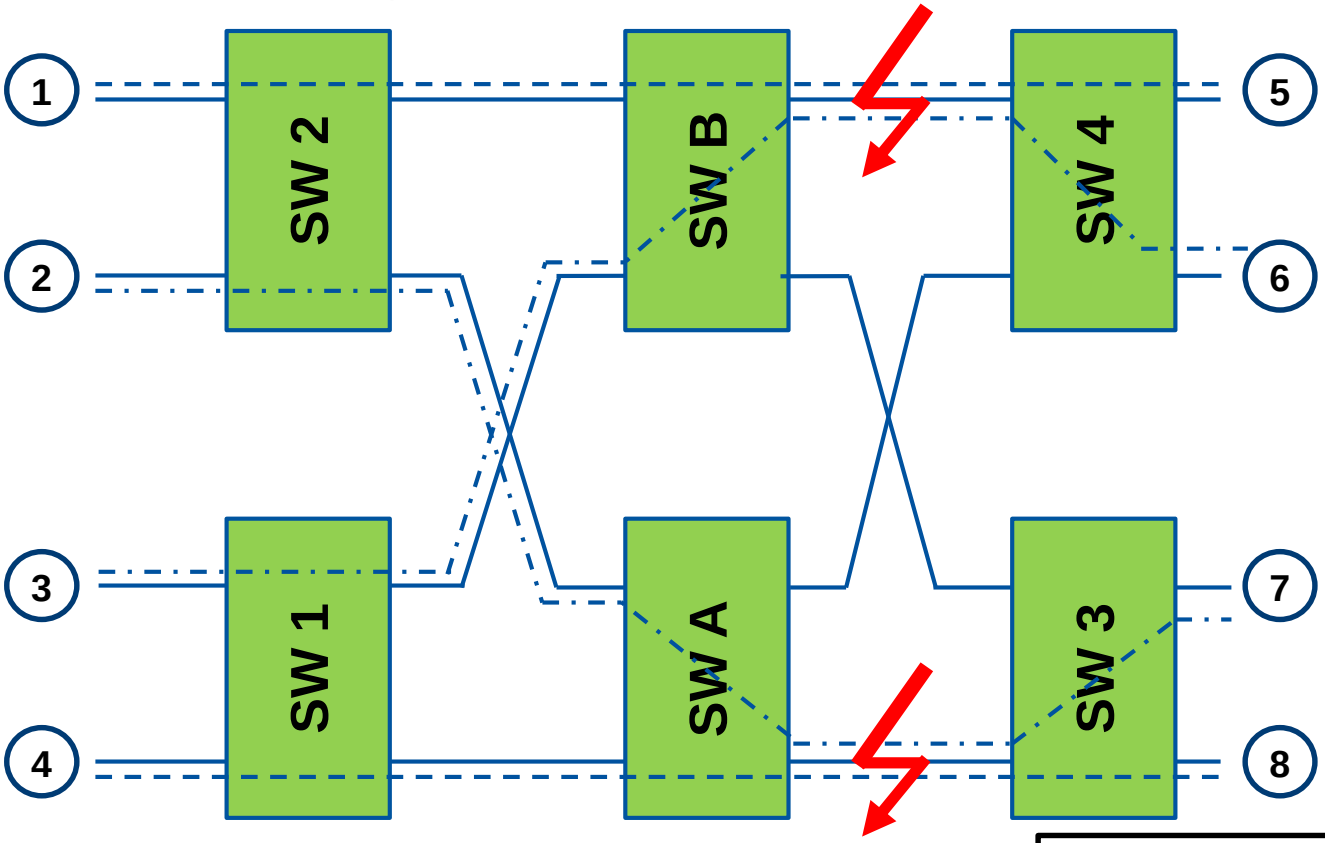
1. **Communication pattern:** **1 → 5, 2 → 6, 3 → 7, 4 → 8**



Collisions with static routing

1. Communication pattern: 1 → 5, 2 → 6, 3 → 7, 4 → 8
2. **Switch routing 2 → 6, 3 → 7: 1 → 5, 2 → 7, 3 → 6, 4 → 8**

→ collisions occur if 1→5, 4→8 are not rerouted at the same time



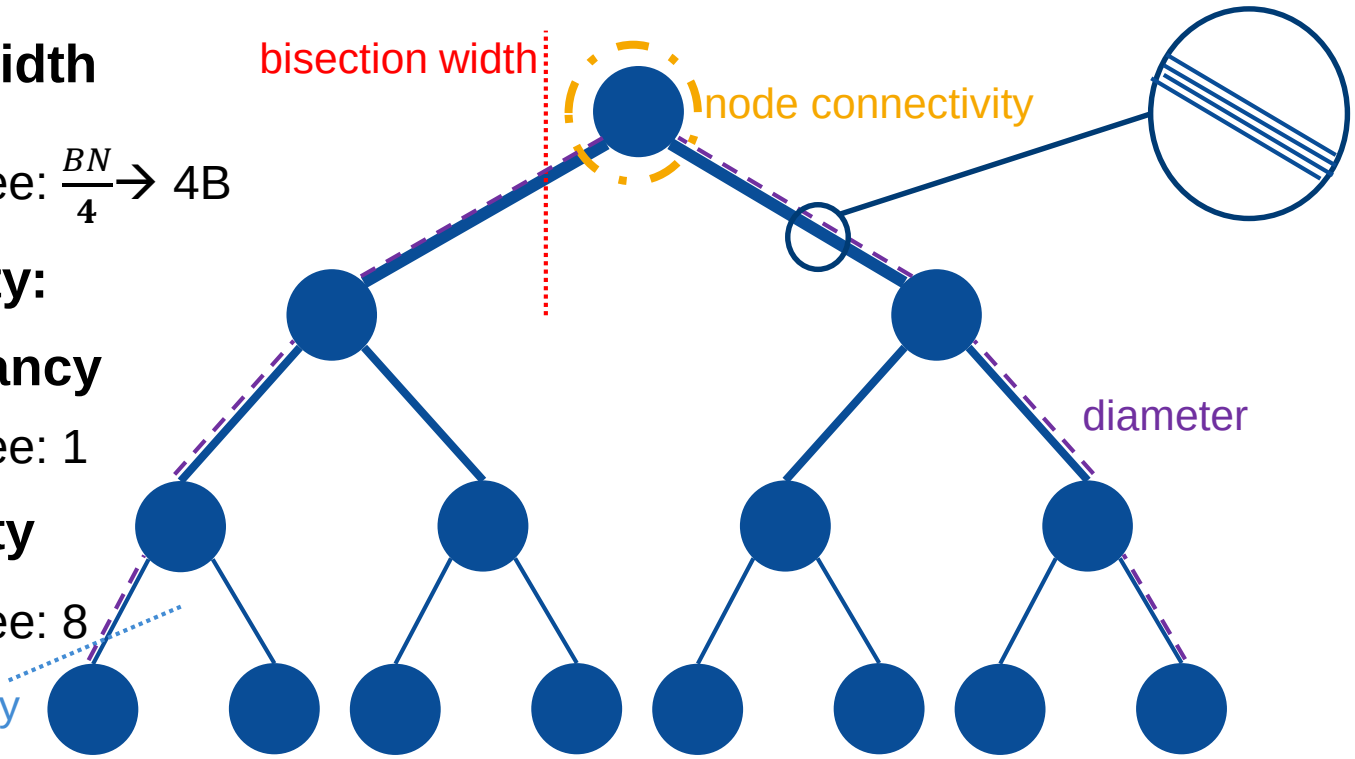
Static vs. adaptive routing

- **Static routing: still the “quasi” standard in commonly used interconnects**
- **Adaptive routing**
  - → Contrary to static routing
  - → Selects data paths through the network topology depending on the network load
  - → Avoids collisions
  - → Bears the potential to make full use of the underlying network hierarchies theoretical maximum bandwidth
- **Things are improving in commodity switch products**
- **Disadvantage of fat-tree type networks in large configurations**
  - → Limited scalability
  - → To overcome this drawback: some modern supercomputers e.g. the Blue Gene feature a mesh network in the form of a multidimensional torus.

Example (binary) full fat-tree

Here: k = 2
N=15

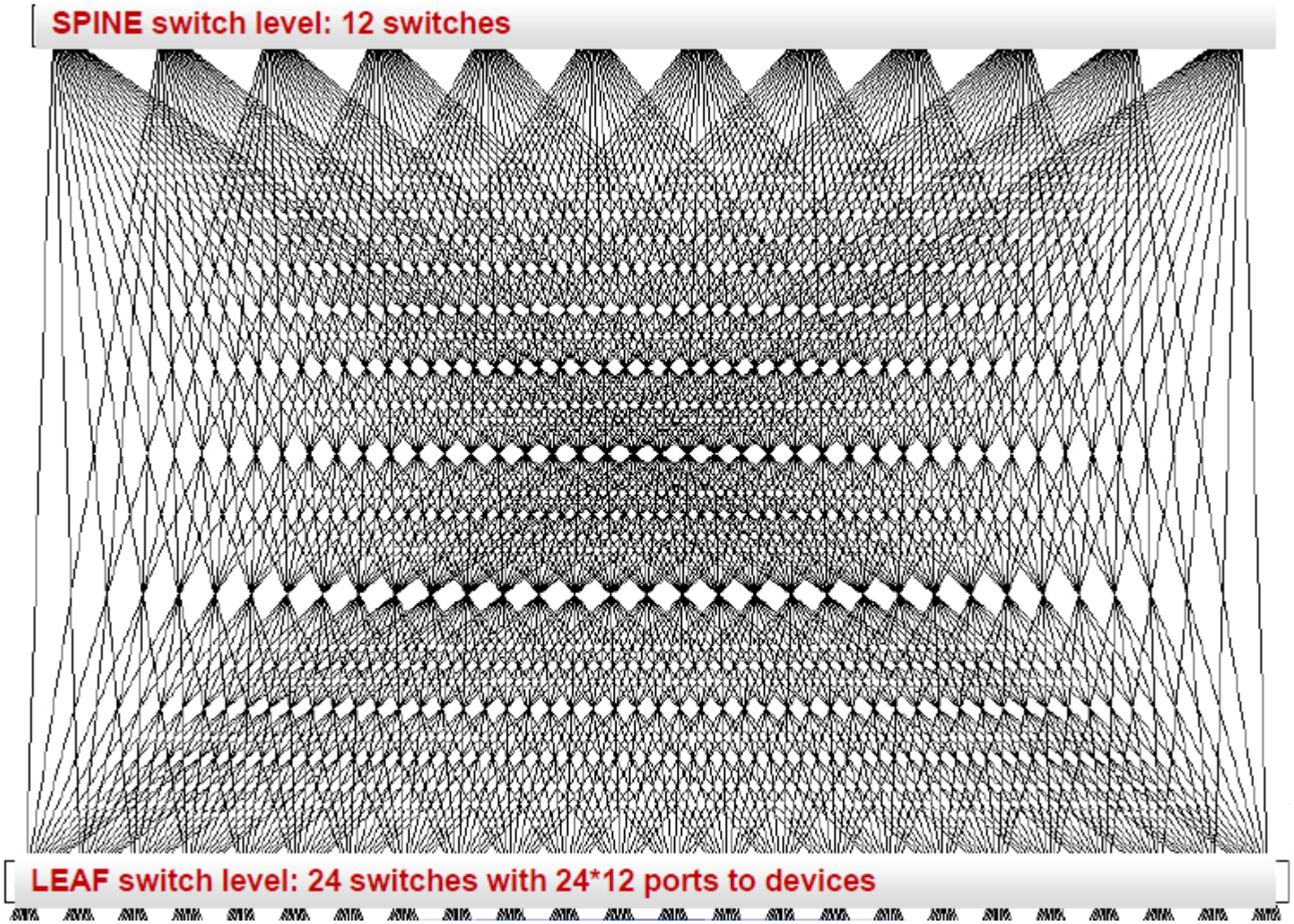
- **Diameter: 2*levelOfHierarchy**
  - → Binary full fat-tree: 6
- **Bisection bandwidth**
  - → Binary full fat-tree: $\frac{BN}{4}$ → 4B
- **Edge connectivity:**
  1 w/o redundancy
  - → Binary full fat-tree: 1
- **Node connectivity**
  - → Binary full fat-tree: 8



Examples for fat-tree networks

- **Infiniband (dominant interconnect in HPC) (here 4 lane examples)**
  - → SDR: 10 GBit/s
  - → DDR: 20 GBit/s
  - → QDR: 40 GBit/s
  - → FDR: 56.25 GBit/s
  - → EDR: 103.125 GBit/s
- **Myrinet**
  - → Last version: 10 GBit/s
  - → Decreasing importance for the HPC field
- **“Commodity” Ethernet**
  - → More than 50% of all TOP500 clusters use normal ethernet (either 1GBit/s or 10 GBit/s)

Examples for fat-tree networks



288 port IB fat-tree

Spine + Leaf level: 12+24= 36 switches

Fat tree can become expensive and difficult to scale

Outline

1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → **Networks**
     - → Basic performance characteristics
     - → **Network topologies**
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → **Mesh networks**
     - → Networks in Top500
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

Mesh networks

- **Forms a multidimensional Cartesian arrangement of compute elements**
  - → Compute elements are interconnected with their immediate neighbors
  - → No direct connections between elements which are not immediate neighbors
  - → Usually the connections are wrapped around the boundaries of the *n-dimensional mesh* to form a torus topology
- **Specialized units inside the compute elements take care of all network traffic that does not concern the local node**
  - → Bypassing the CPU whenever possible
- **Network diameter: sum of the system's size in all direction of the hypercube**

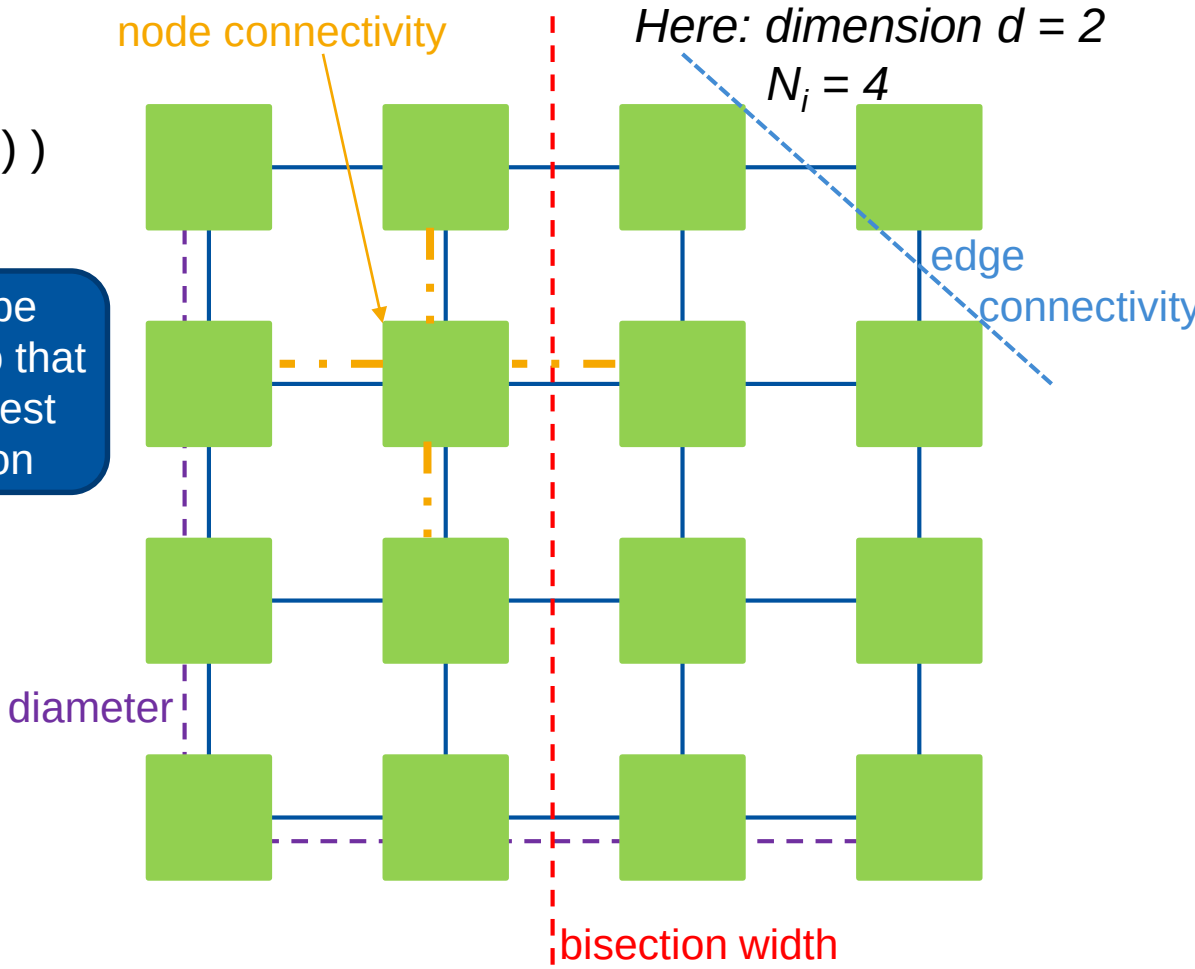
Example (2D) mesh

- **Diameter:** $\sum_{i=1}^{d}(N_i - 1)$
  - → 2D-case: 6 (=(4-1)+(4-1))
- **Bisection bandwidth:**

$$B\left(\prod_{i=1}^{d-1} N_i\right)$$

N_i must be ordered, so that N_d is biggest dimension

  - → 2D-case: 4B
- **Edge connectivity: d**
  - → 2D-case: 2
- **Node connectivity: 2d**
  - → 2D-case: 4



N: #processing devices
N_i: #processing devices in direction *i*
B: connection bandwidth
d: dimension

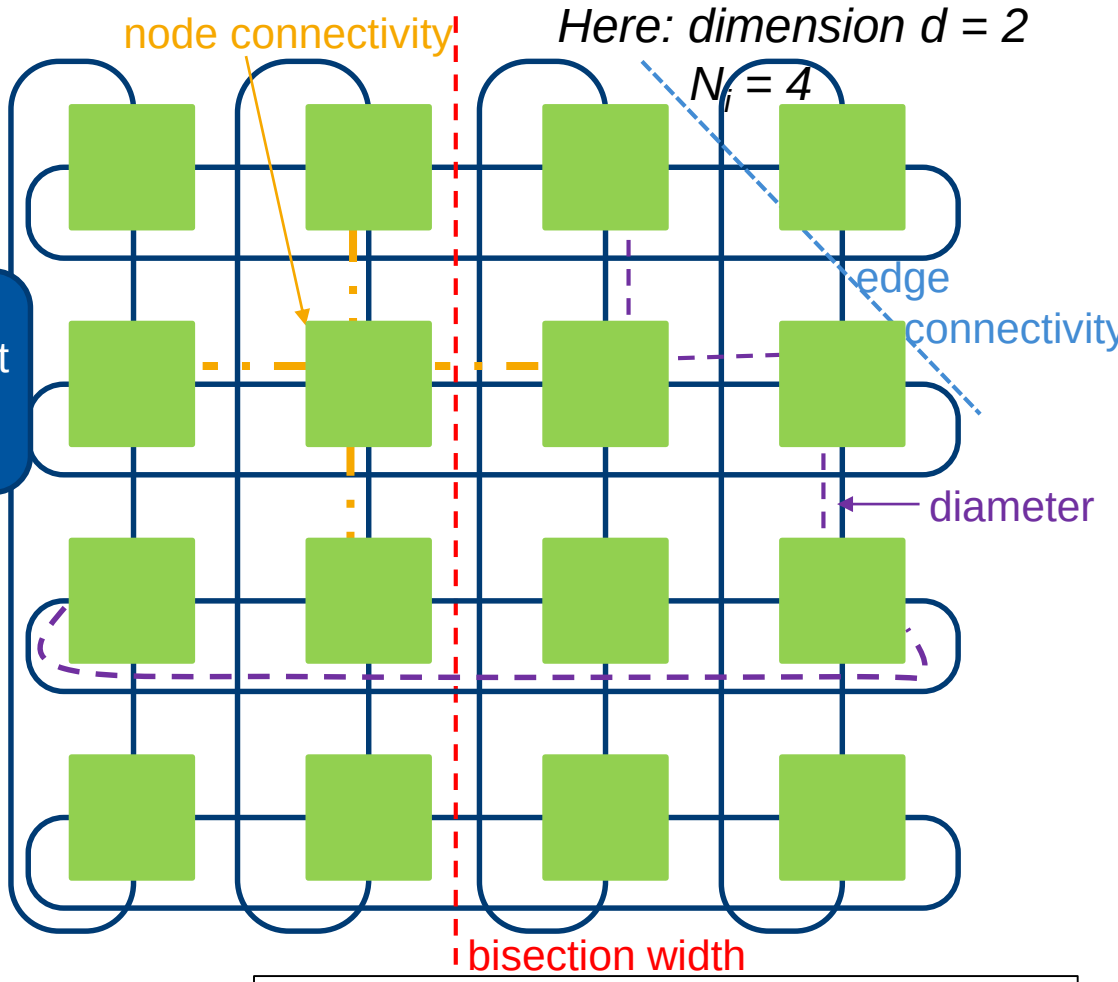
Example (2D) torus

- **Diameter:** $\sum_{i=1}^{d} \left\lfloor \frac{N_i}{2} \right\rfloor$
  - → 2D-case: 4 (=2+2)
- **Bisection bandwidth:**

$$B2\left(\prod_{i=1}^{d-1} N_i\right)$$

N_i must be ordered, so that N_d is biggest dimension

  - → 2D-case: 8B
- **Edge connectivity: 2d**
  - → 2D-case: 4
- **Node connectivity: 2d**
  - → 2D-case: 4



N: #processing devices
N_i: #processing devices in direction *i*
B: connection bandwidth
d: dimension

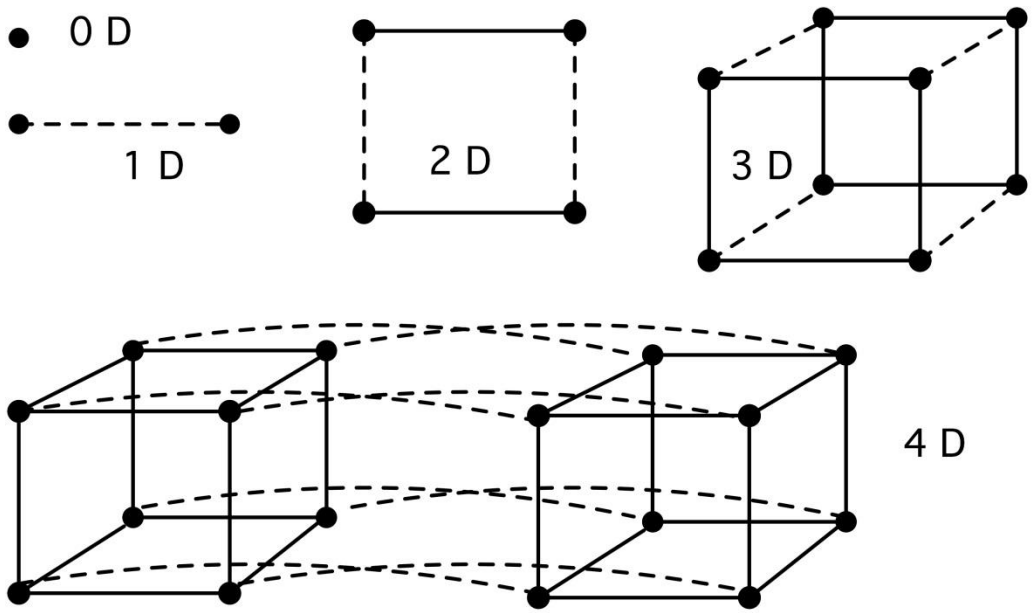
Torus

- **Inside a torus, each node acts as a „router“**
- **Bisection bandwidth does not scale linearly:**
  - → $B_b \sim 2(N_1 N_2 \ldots N_{d-2} N_{d-1})$ for $N_1 \leq N_2 \leq \cdots \leq N_d$
  - → For a regular d-dimensional torus with N nodes: $2N^{\frac{d-1}{d}}$
  - → $\frac{B_b}{N} \rightarrow 0$ for large N
- **Machines utilizing a torus network scale good in practice**
  - → Contrary to what these facts look like
- **Examples for torus networks: IBM Blue Gene, Cray XT**

Hypercube

- **= Multidimensional mesh of processors with exactly two processors in each dimension**
- **With network diameter d**

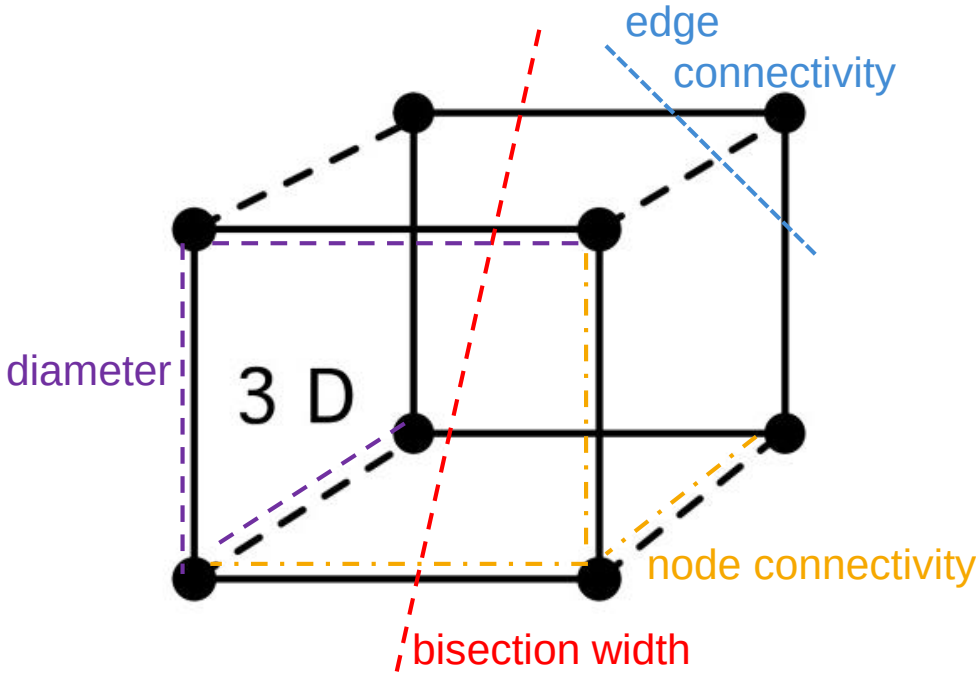
  → d dimensional hypercube thus consists of $\mathbf{2}^{\boldsymbol{d}}$ processors



Example (3D) hypercube

- **Diameter: d**
  - → 3D-case: 3
- **Bisection bandwidth:** $B2^{d-1}$
  - → 3D-case: 4B
- **Edge connectivity: d**
  - → 3D-case: 3
- **Node connectivity: d**
  - → 3D-case: 3

Here: dimension d = 3

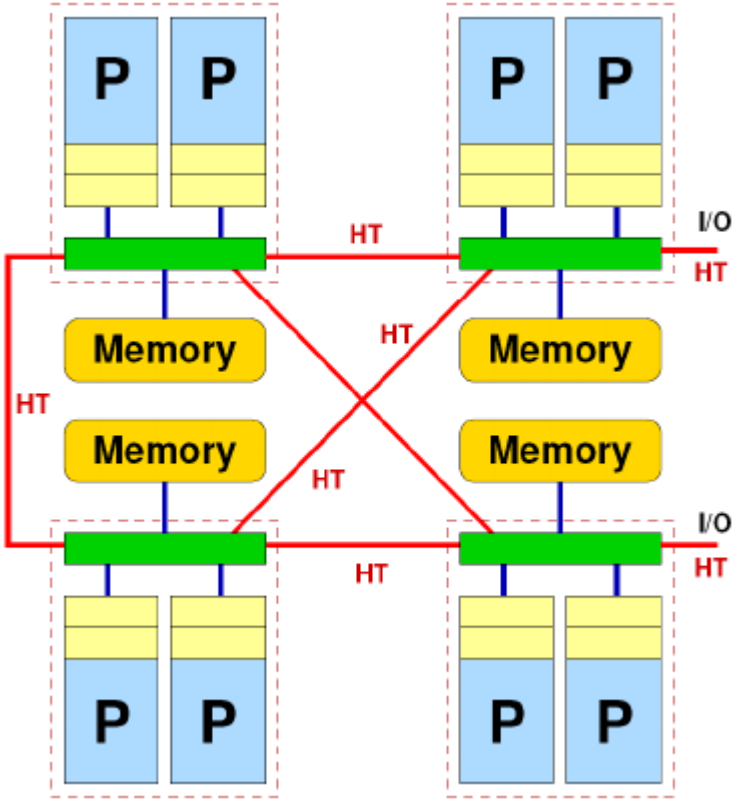


$N = 2^d$: #processing devices
B: connection bandwidth
d: dimension

Application of mesh networks

- **Shared-memory systems for ccNUMA capable connections between locality domains (smaller scale)**

- **Example AMD 4-socket dual-core node with HyperTransport fabric**
  - → 4-socket ccNUMA system with a HyperTransport-based mesh network
  - → Heterogeneous construct: rightmost HT-links are used for IO
    - → Communication between the 2 locality domains on the right incurs an additional hop via one of the other domains



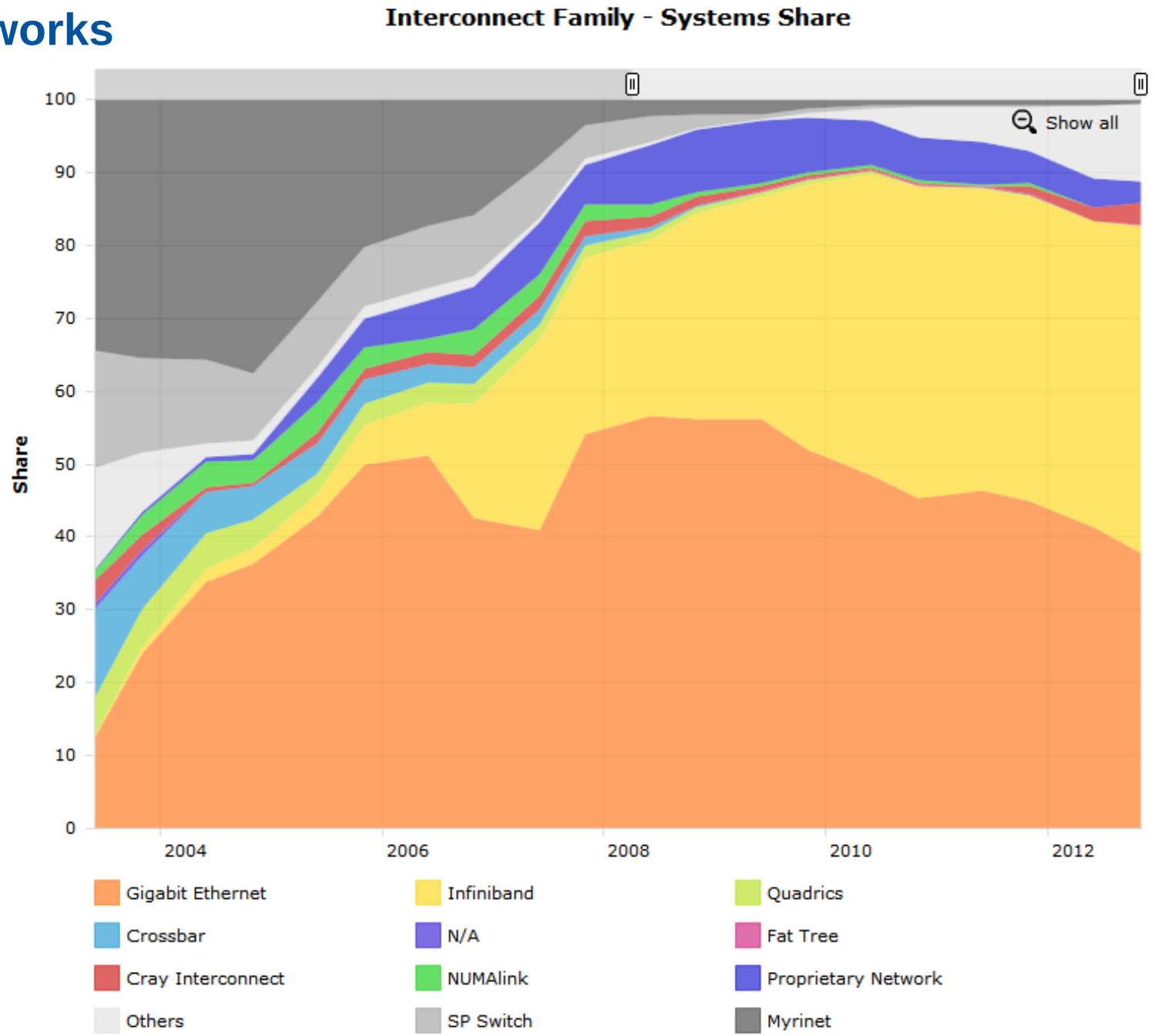
Overview of Topologies

Topology	Max degree of a network	Edge connectivity	Diameter of network	Bisection bandwidth
Bus	1	1	1	B
Ring	2	2	$\left\lfloor \frac{N}{2} \right\rfloor$	$2B$
Fully connected	see exercise			
Sw./Fat Tree	1 w/o redundancy	depends on design	2*Levels_of_ Hierarchy	depends on design
Mesh	$2d$	d	$\sum_{i=1}^{d} (N_i - 1)$	$B\left(\prod_{i=1}^{d-1} N_i\right)$
Torus	$2d$	$2d$	$\sum_{i=1}^{d} \left\lfloor \frac{N_i}{2} \right\rfloor$	$B2\left(\prod_{i=1}^{d-1} N_i\right)$
Hypercube	d	d	d	$B2^{d-1}$

Outline

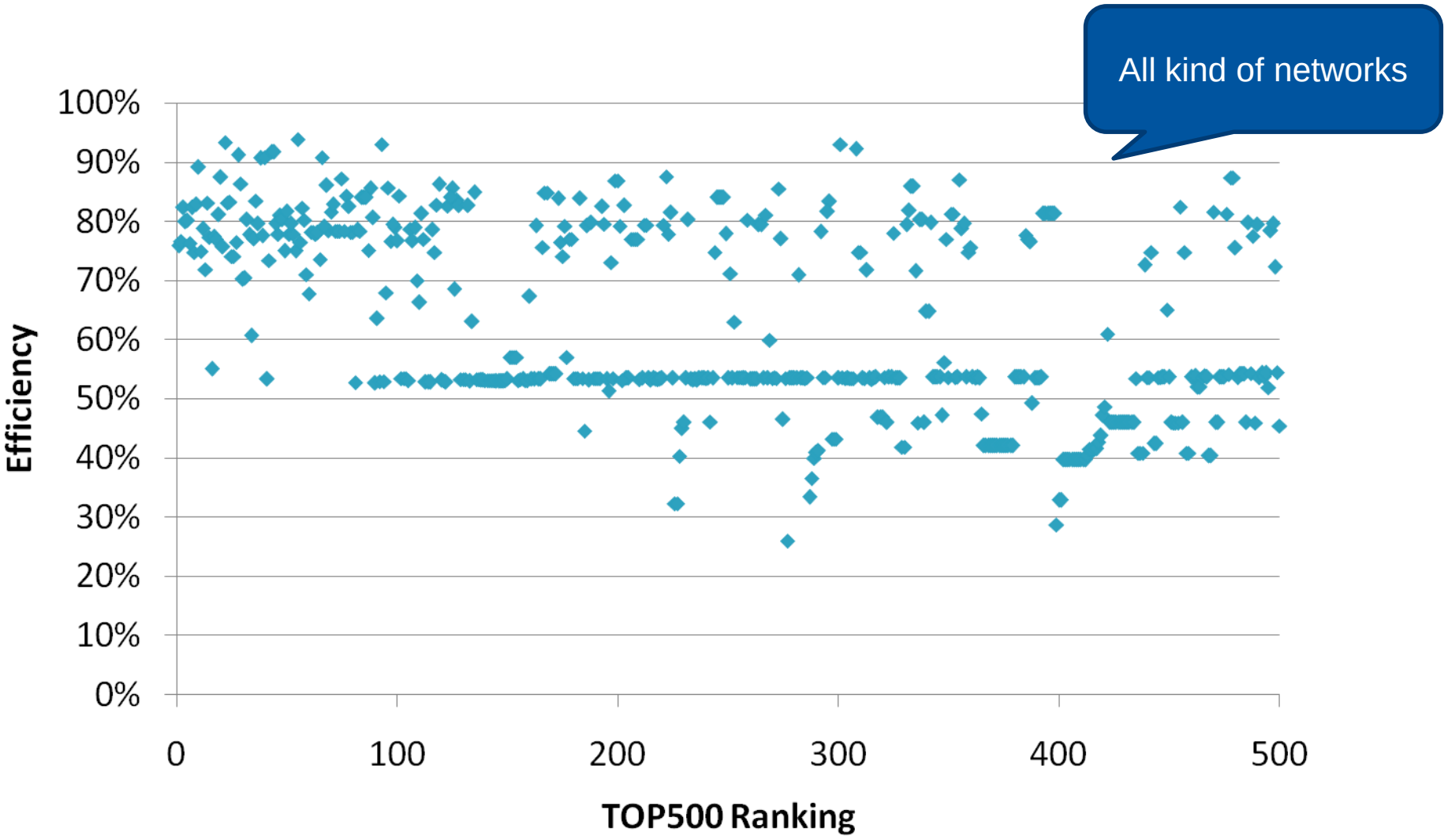
1. Why supercomputers?
2. Modern processors
3. Basic optimization techniques for serial code
4. Data access optimization
5. **Parallel computers**
   - → Flynn's taxonomy
   - → Basic limitations of par. computing
     - → Amdahl's law
     - → Gustafson's law
   - → Multicore processors
   - → Multithreaded processors (SMT)
   - → Shared-memory computers
     - → UMA
     - → ccNUMA
   - → Distributed-memory computers
   - → **Networks**
     - → Basic performance characteristics
     - → Network topologies
       - → Buses
       - → Ring & fully connected networks
       - → Switched & fat-tree networks
       - → Mesh networks
     - → **Networks in Top500**
6. Parallelization and optimization strategies
7. Parallel algorithms
8. Distributed-memory programming with MPI
9. Shared-memory programming with OpenMP
10. Hybrid programming (MPI + OpenMP)
11. Heterogeneous architectures (GPUs, Xeon Phis)
12. Energy efficiency

Networks



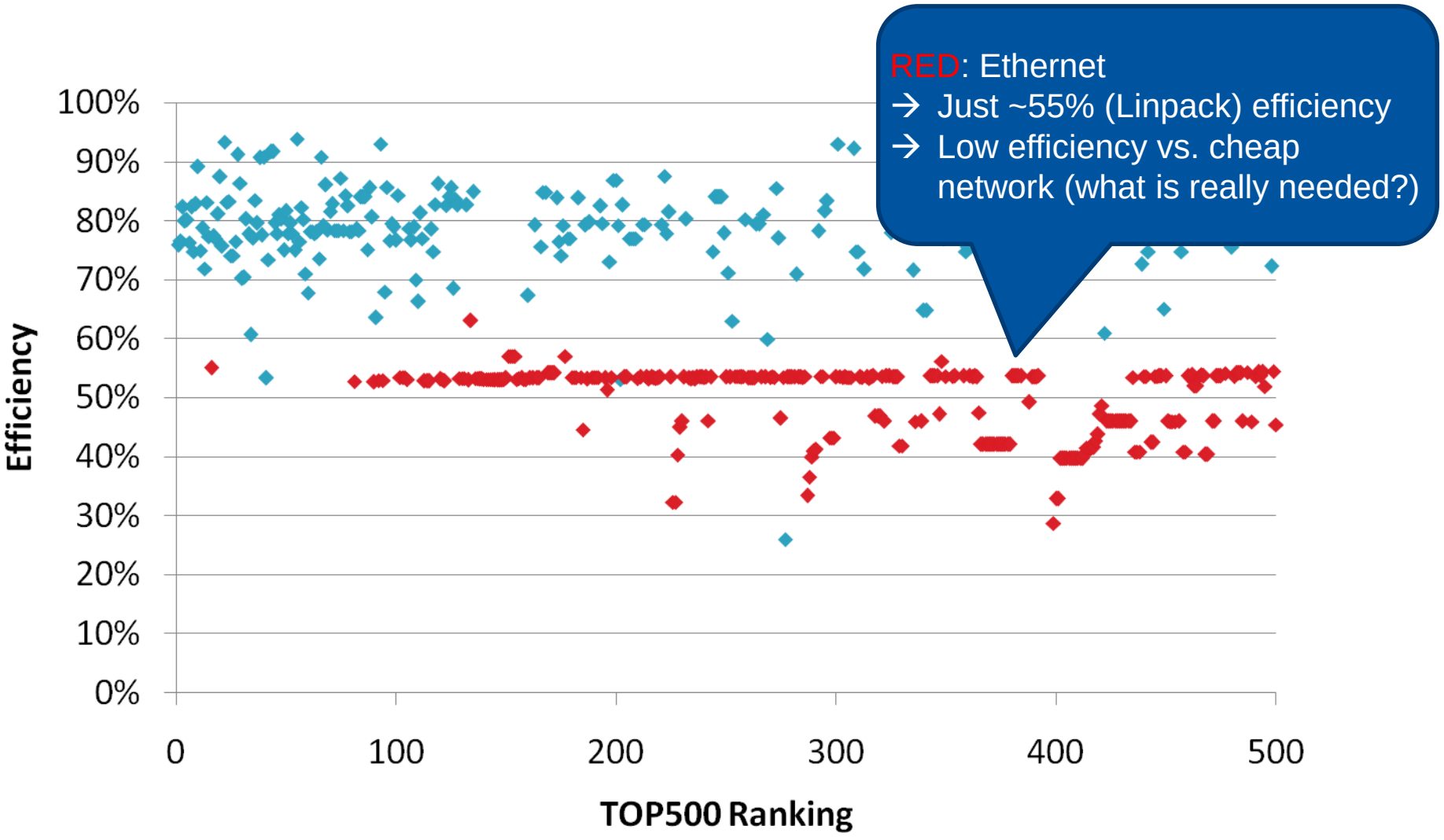
TOP500

Linpack Efficiency



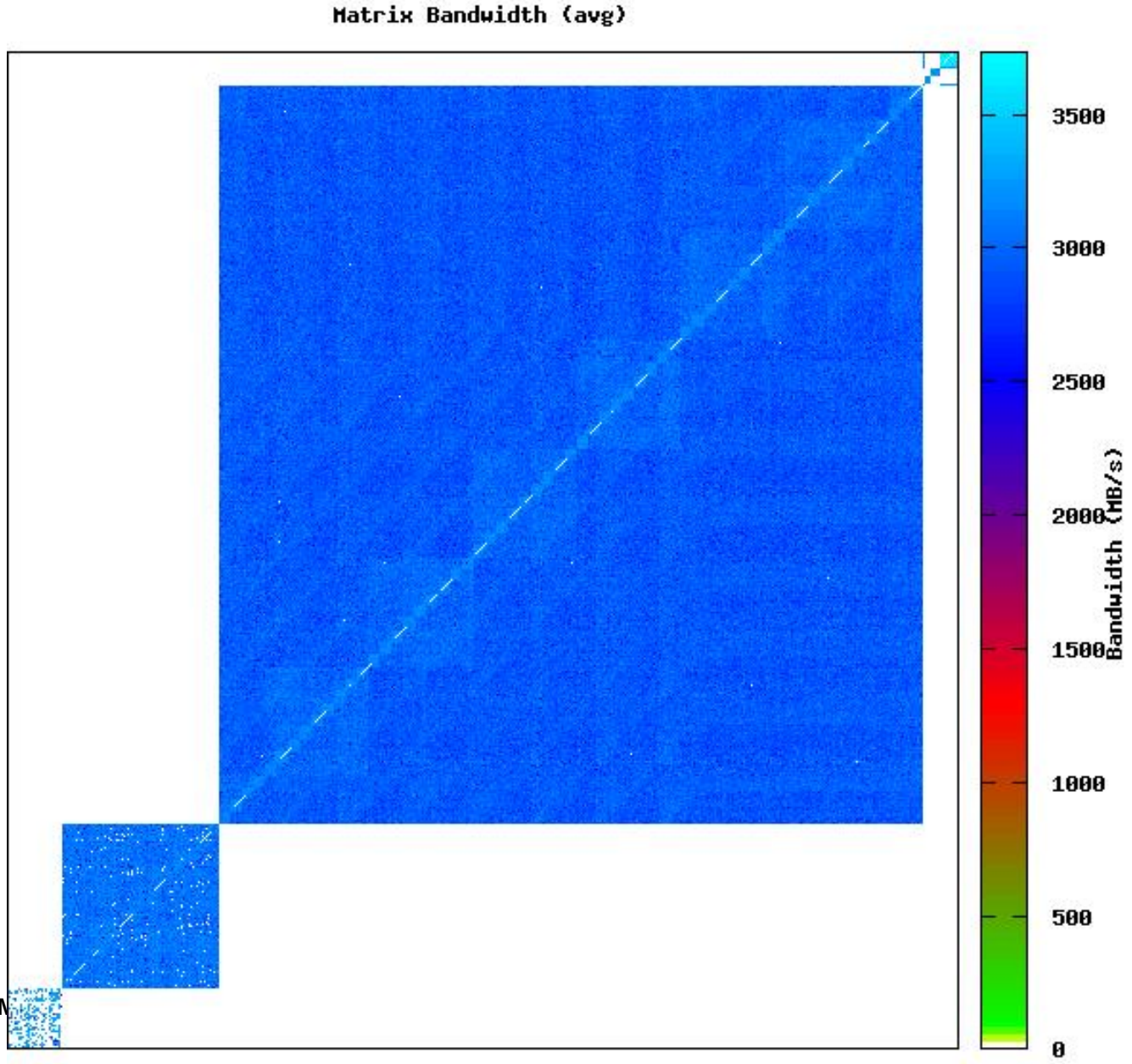
(E. Strohmeier, ISC'09)

Linpack Efficiency vs. Network

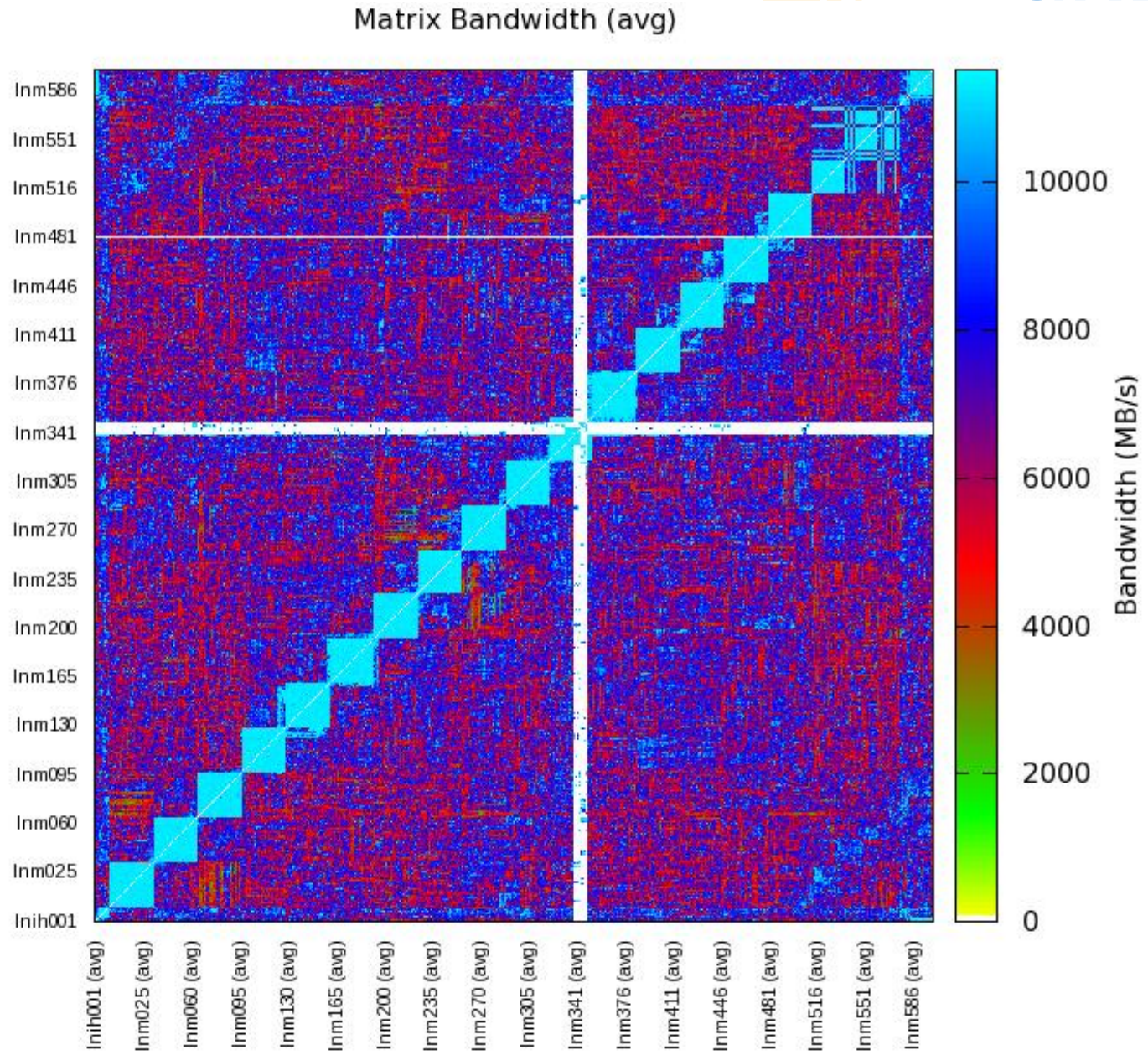


(E. Strohmeier, ISC'09)

Matrix Bandwidth Full Fat-tree (RWTH Bull Cluster)



Matrix Bandwidth: 1:2 Blocking (RWTH CLAIX)



Quiz: What you have learnt

Question 1: A loop with one load, one store and one DP floating point operation per iteration is executed on an architecture with machine balance $B_M = 0.01 \frac{\text{Words}}{\text{Flop}}$ and a peak performance $P_{max} = 200\ \text{Gflop/s}$. Determine the lighspeed *P* for absolute performance in Gflop/s.

a) $P = 1\ \text{Gflop/s}$

b) $P = 2\ \text{Gflop/s}$

c) $P = 4\ \text{Gflop/s}$

d) $P = 200\ \text{Gflop/s}$

Quiz: What you have learnt

Question 1: A loop with one load, one store and one DP floating point operation per iteration is executed on an architecture with machine balance $B_M = 0.01\ \frac{\text{Words}}{\text{Flop}}$ and a peak performance $P_{max} = 200\ \text{Gflop/s}$. Determine the lighspeed *P* for absolute performance in Gflop/s.

$$a)\ P = l \cdot P_{max} = min\left(1, \frac{B_M}{B_c}\right) \cdot P_{max}$$

$$= min\left(1, \frac{0.01\ \frac{\text{Words}}{\text{Flop}}}{\frac{2\text{Words}}{1\ \text{Flop}}}\right) \cdot\ 200\ \text{Gflop/s}$$

$$= 1\ \text{Gflop/s}$$

Quiz: What you have learnt

Question 2: Which of the following statements about shared memory systems is true?

a) Memory accesses can be easily optimized on UMA systems due to its anisotropic (directionally dependent) memory design
b) Most of the world‘s fastest computers with >100.000 CPU cores are UMA systems
c) Cache coherence in ccNUMA systems is easily achievable compared to cache coherence in UMA systems
d) ccNUMA systems are scalable in terms of their memory bandwidth

Quiz: What you have learnt

Question 2: Which of the following statements about shared memory systems is true?

d) ccNUMA systems are scalable in terms of their memory bandwidth

Quiz: What you have learnt

Question 3: Given is a network with bandwidth $B = 1000\ \frac{MB}{s}$ and latency $T_l = 50\ \text{ms}$. Determine $N_{1/2}$ for this network.

a) $N_{1/2} = 5\ \text{MB}$

b) $N_{1/2} = 50\ \text{MB}$

c) $N_{1/2} = 100\ \text{MB}$

d) $N_{1/2} = 500\ \text{MB}$

Quiz: What you have learnt

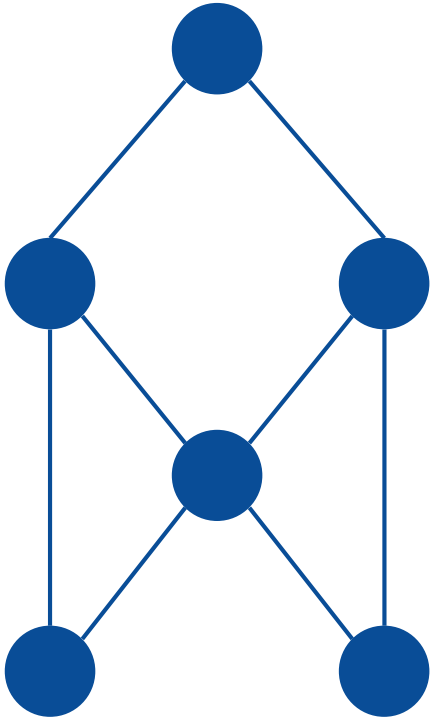
Question 3: Given is a network with bandwidth $B = 1000\ \frac{MB}{s}$ and latency $T_l = 50\ \text{ms}$. Determine $N_{1/2}$ for this network.

b) $N_{1/2} = T_l \cdot B = 0.05\ s \cdot 1000\ \frac{MB}{s} = 50\ \text{MB}$

Quiz: What you have learnt

Question 4: Determine the diameter and edge connectivity of the following network.

a) Diameter: 1, edge connectivity: 2
b) Diameter: 2, edge connectivity: 2
c) Diameter: 2, edge connectivity: 3
d) Diameter: 3, edge connectivity: 3



Quiz: What you have learnt

Question 4: Determine the diameter and edge connectivity of the following network.

b) diameter: 2,
edge connectivity: 2