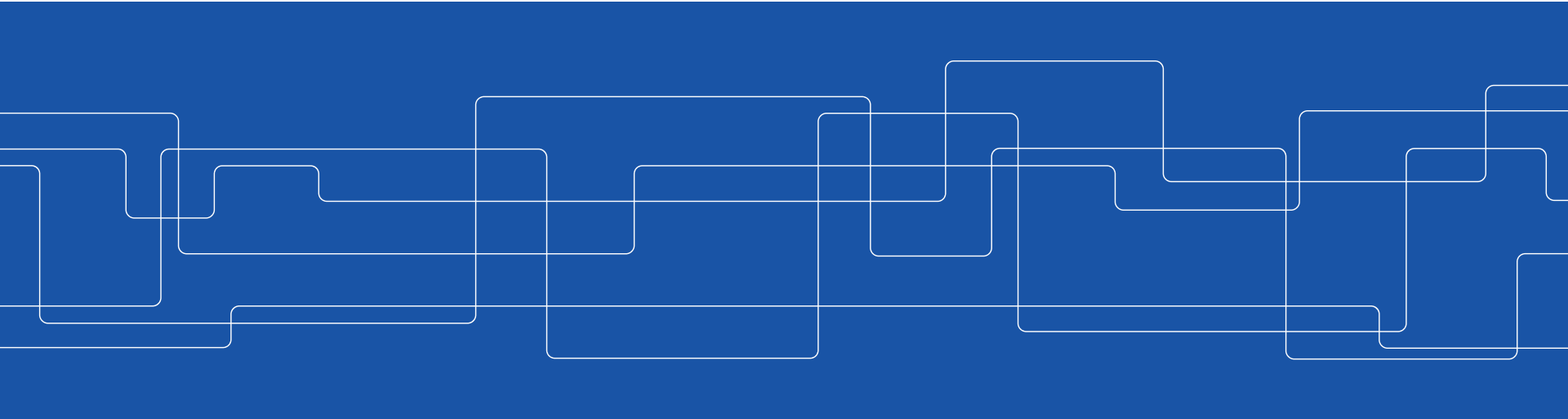
High-Performance Architecture Lectures

1. Basic Computer Organization – *What is a processor and how it works?*
   - Design of PDcLX-1 processor
2. Program Execution – *How does a Code run on a Processor?*
   - Programming PDcLX-1 processor
3. Pipelined Processor – *Increase Performance of our Processor*
   - How much speed-up with pipelined processor? What it is the cost of it?
4. Scalar Processor – *Increase Performance of our Processor*
   - PDcLX-2 and why ISA is important
5. **On the way to Supercomputers – *Caches, Multicore Processor, Networks***
   - **Beskow Supercomputer**

On the way to Supercomputers – *Caches, Multicore Processor, Networks*

Stefano Markidis and Erwin Laure
KTH Royal Institute of Technology



The Memory Wall

Though processor speeds have increased dramatically over the past two decades thanks to pipeline and superscalar architectures, **the speed of main memory has not been able to keep pace**

- It takes such a huge number of processor clock cycles to transfer code and data between main memory and the registers and execution units
  - If no solution to this bottleneck (memory wall), it would kill most of the performance gains brought on by the increase in processor performance

Cache Memories

To solve this, computer system designers fill the speed gap by **placing smaller amounts of faster, more expensive memory, called *cache memory***, in between main memory and the registers.

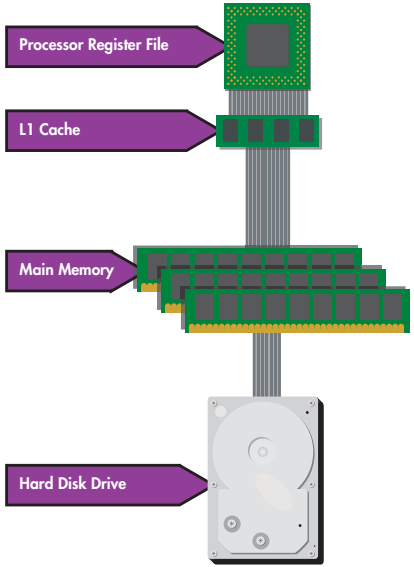
Caches hold chunks of **frequently used code and data**, keeping them within **easy reach of the processor**.

The Memory Hierarchy

There are multiple levels of cache between main memory and the registers.

- The level 1 cache (***L1***) is the smallest, most expensive bit of cache, so it's located the closest to the processor's back end.
- Most PC systems have another level of cache, called ***L2 cache***, located between the L1 and main memory
- Some systems even have a third cache level, *L3 cache*, located between the L2 cache and main memory.

- In fact, **main memory itself is really just a cache for the hard disk drive**.



How Caches Work

When the processor needs a particular piece of code or data, it first checks the L1 cache to see if the desired item is present

- If it is - a ***cache hit*** - it moves that item directly to either the fetch stage (in the case of code) or the register file (in the case of data).
- If the item is not present - a ***cache miss*** - the processor checks the slower but larger L2 cache
  - If the item is present in the L2, it's copied into the L1
  - If there's a cache miss in the L2, the processor checks the L3, and so on, until there's either a cache hit, or the cache miss propagates all the way out to main memory

Splitting the Cache - Harvard Architecture

One popular way of laying out the L1 cache is to have **code and data stored in separate halves of the cache**.

- The code half of the cache is often referred to as the ***instruction cache* or *I-cache***, and the data half of the cache is referred to as the ***data cache* or *D-cache*.**

The split L1 cache design is often called the **Harvard architecture**.

Memory Hierarchy Characteristics

Level	Access Time	Typical Size	Technology	Managed By
Registers	1–3 ns	1KB	Custom CMOS	Compiler
Level 1 Cache (on-chip)	2–8 ns	8KB–128KB	SRAM	Hardware
Level 2 Cache (off-chip)	5–12 ns	0.5MB–8MB	SRAM	Hardware
Main Memory	10–60 ns	64MB–1GB	DRAM	Operating system
Hard Disk	3,000,000–10,000,000 ns	20GB–100GB	Magnetic	Operating system/

When Caches Work Well - Locality of Reference

Caching works because of property exhibited by all types of code and data: **locality of reference**.

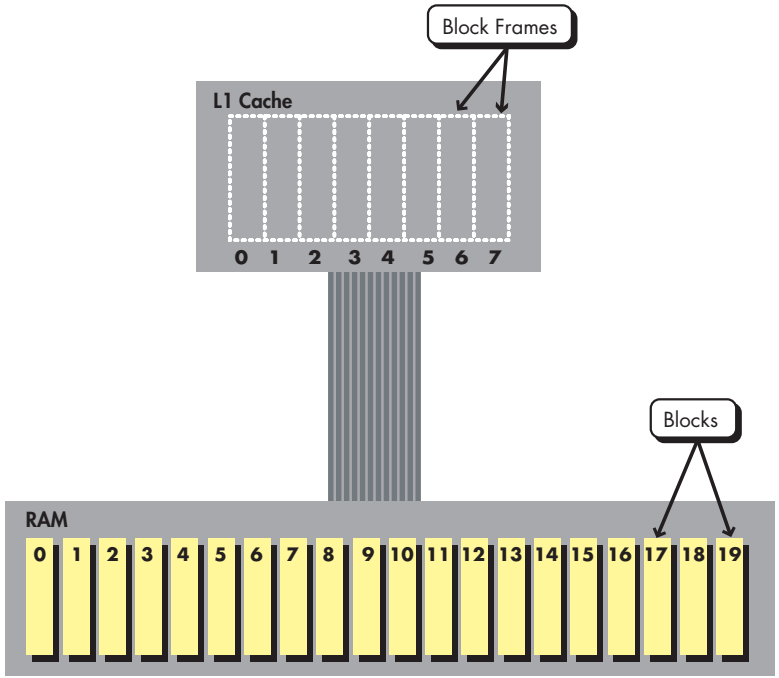
We generally find it useful to talk about two types of locality of reference:

- **Spatial locality** if the CPU needs an item from memory at any given moment, it's likely to need that item's neighbors next
- **Temporal locality** if an item in memory was accessed once, it's likely to be accessed again in the near future

Depending on the type of application, **both code and data streams** can exhibit **spatial and temporal locality** using effectively the caches

Cache Organization: Blocks and Block Frames

- When the CPU requests a particular piece of data from the memory subsystem, that piece gets fetched and loaded into the L1 along with some of its nearest neighbors
  - The data is called the ***critical word***
  - The surrounding group of bytes is a **cache line** or **cache block**.
- Cache blocks form the basic unit of cache organization, and RAM is also organized into blocks of the same size as the cache's blocks.
  - When **a block is moved from RAM to the cache**, it is placed into a special slot in the **cache called a *block frame***.



Tags for Tagging Block Frames

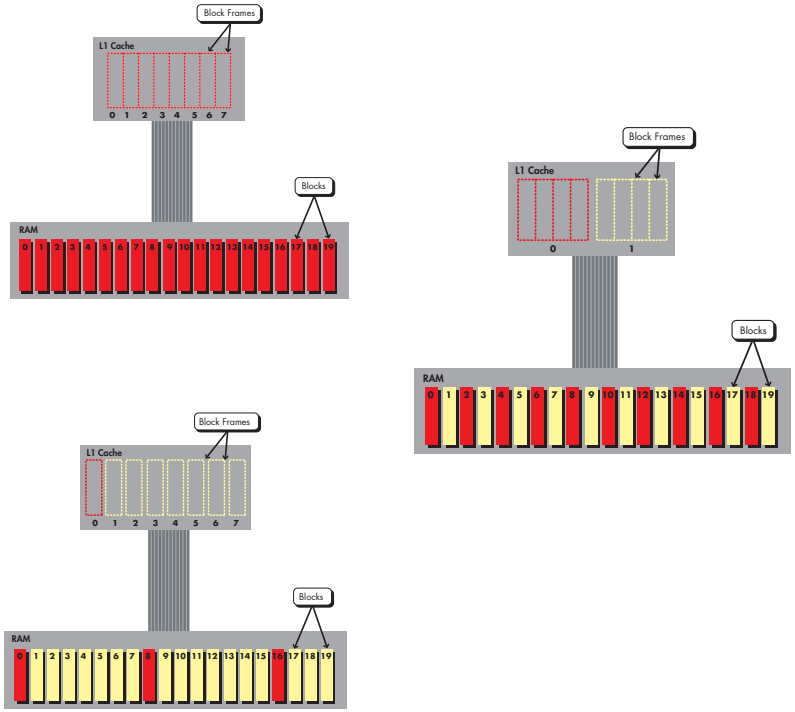
When the CPU requests a byte from memory, it needs to know

1. whether or not the needed block is actually in the cache
2. the location of the block within the cache (cache hit)
3. the location of critical word within the block (cache hit)
   - A cache accommodates all three needs by associating a **special piece of memory- a tag -** with each block frame in the cache. **The *tag* holds information about the blocks**

- The larger the cache, the greater the number of blocks, and the greater the number of blocks, **the more tag RAM you need to search** and the longer it can take to locate the correct block.

Fully Associative/Direct/N-Way Associative Mapping

- There are three scheme for mapping RAM blocks to cache block frames
  - Associative Mapping
  - Direct Mapping
  - Set N-way Associative Mapping



Eviction Policies

- Caches can increase the amount of benefit they derive from **temporal locality** by implementing **an intelligent *replacement policy* (*eviction policy***).
  - A replacement policy dictates which of the blocks currently in the cache will be replaced by any new block that gets fetched in.

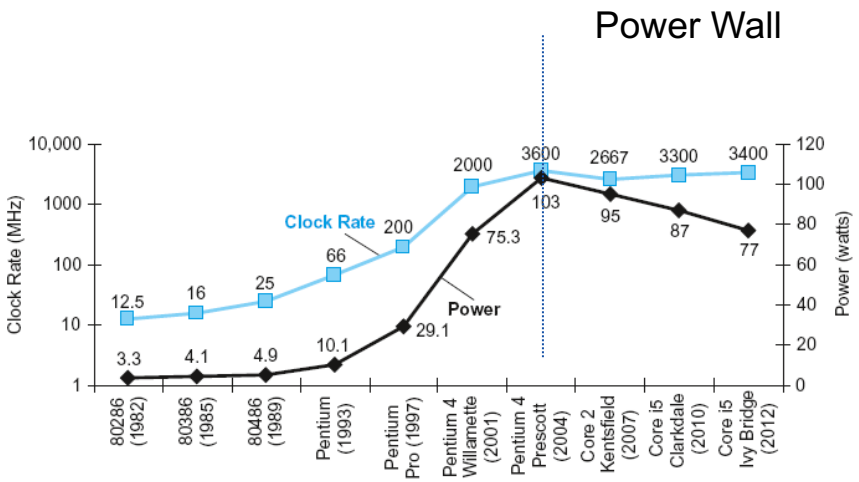
Least Recently Used (LRU) Eviction Policy

- The optimal replacement policy is to evict the block that has gone the longest period of time without being used, or the *Least Recently Used (LRU) block*.
  - if a block hasn't been used in a while, it's less likely to be part of the current working set
- Most caches wind up implementing some type of *pseudo-LRU algorithm* that approximates true LRU by marking blocks as more and more *dirty* the longer they sit unused in the cache.

Hitting the Power Wall

Since 2003 Intel and other processor manufacturers started paying attention to the power dissipation of their chips.

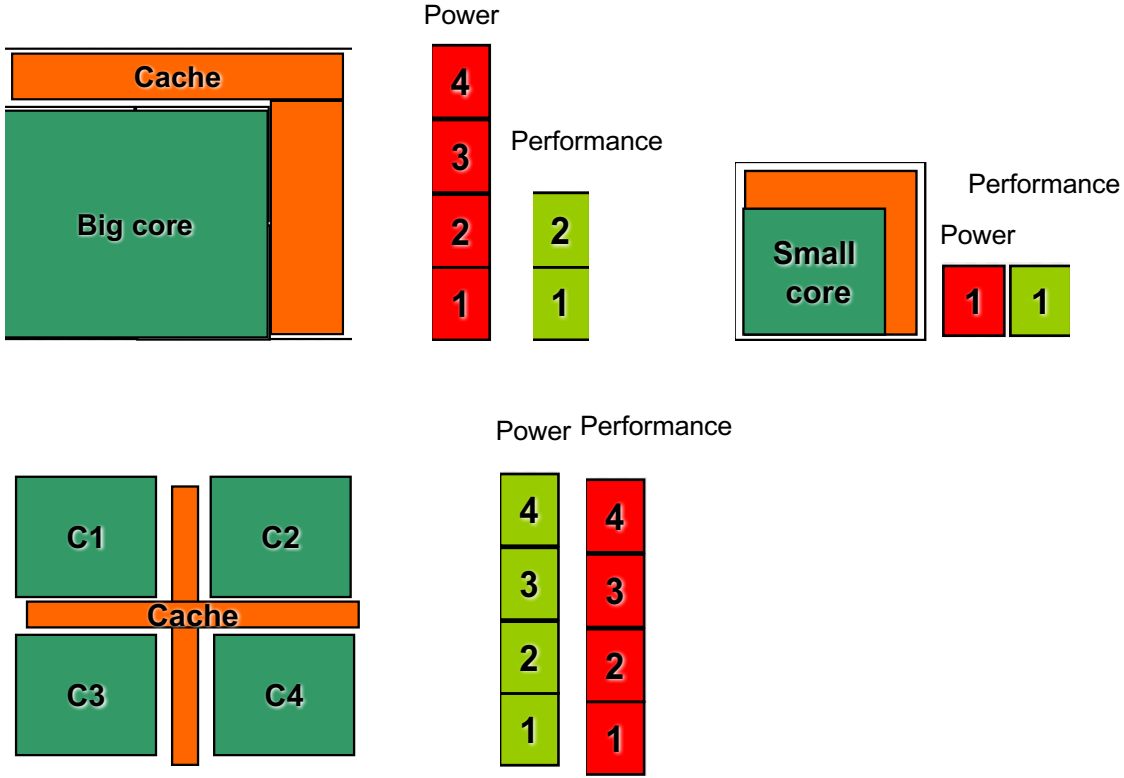
- *Power wall* is a term used by Intel to describe the point at which its **chips' *power density* began to limit further integration and clock speed scaling**.



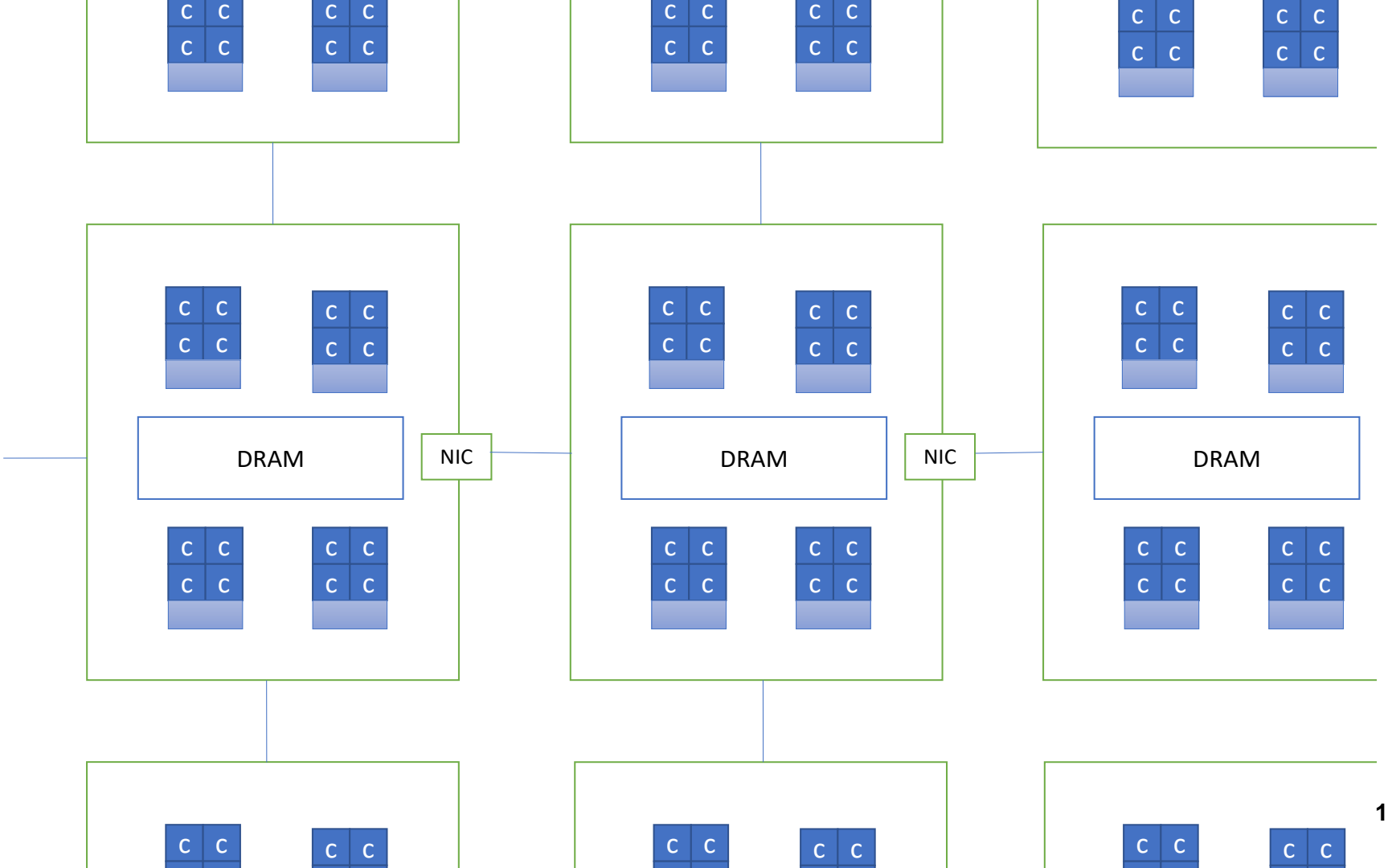
Welcome to the Multicore Era

Many **simple small processor (cores) at lower clock** frequency are **more power efficient**

- What does simple means?
- How many cores today on a CPU?
- How many a GPU?



Supercomputers



Most of modern supercomputer hardware are built following two principles:

- Use of **commodity hardware**: Intel CPUs, AMD CPUs,…
- Using **parallelism** to achieve very **high performance**

What Makes a Supercomputer a Supercomputer?

The use of **high-speed dedicated** interconnection (between nodes) network.

- How come is network is so important in supercomputer?

MARE NOSTRUM SC

The use of **high-speed dedicated** interconnection (between nodes) network.

- How come is network is so important in supercomputer?

Communication across network is slowest process in supercomputer!

MARE NOSTRUM SC

Time Scales in a Supercomputer

Time for one computation → 1 ns

Time to move data from cache to process → ~2-4 ns

Time to move data from RAM to process → ~ 10-100 ns

Time to communicate data → it depends on how much data and how many message but > **1 us** (1,000x processing time!)

Network Performance Measures

- The overall performance of supercomputers depends critically on the performance of the network used to connect the individual nodes
    - How fast can messages be transmitted and how much data can be exchanged?
- **Latency**: time between *start* of packet *transmission* to the *start* of packet *reception (s)*
- **Bandwidth**: how much data can be transmitted over the network (bit/s)

Different Technologies

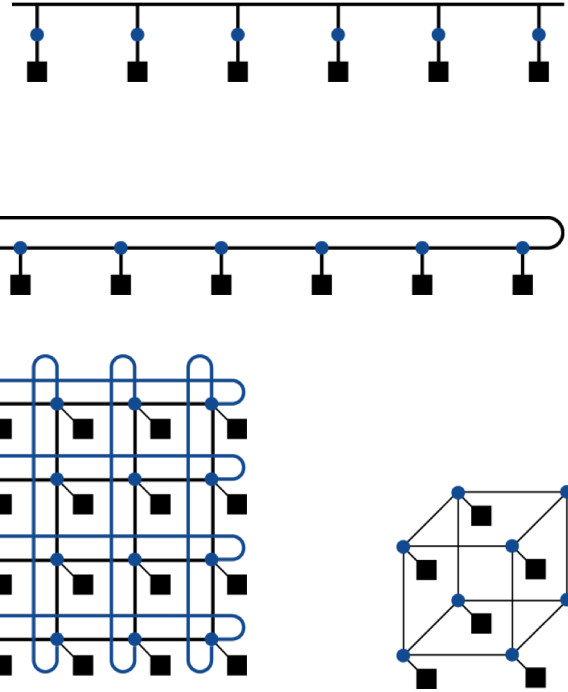
- Ethernet
- Myrinet
- Infiniband

- Proprietary networks
  - Cray Aries
  - IBM BlueGene

- **Differ in bandwidth and latency**

Different Topologies

- Networks can be **arranged in a number of ways**
  - Typical design goals is to balance **performance** and **cost**
  - Some common topologies
    - bus, ring, mesh, tree, fat-tree



Beskow – Our Supercomputer

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
68	Commissariat a l'Energie Atomique (CEA) France	**Tera-1000-1** - bullx DLC 720, Xeon E5-2698v3 16C 2.3GHz, Infiniband FDR Bull, Atos Group	70,272	1,871.0	2,586.0	1,042
69	KTH - Royal Institute of Technology Sweden	**Beskow** - Cray XC40, Xeon E5-2695v4/E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	67,456	1,802.5	2,438.1	842
70	Institute for Molecular Science Japan	**Molecular Simulator** - NEC LX Cluster, Xeon Gold 6148/6154, Intel Omni-Path NEC	38,552	1,785.6	3,072.8	544
71	BASF Germany	**QURIOSITY** - Apollo XL230k, Xeon Gold 6148 20C 2.4GHz, Intel Omni-Path HPE	35,280	1,750.2	2,709.5	
72	CINECA Italy	**Marconi Intel Xeon** - Lenovo NeXtScale nx360M5, Xeon E5-2697v4 18C 2.3GHz, Omni-Path Lenovo	54,432	1,723.9	2,003.1	1,361

2 x Intel CPUs per node:

- 9 of the cabinets have Xeon E5-2698v3 Haswell 2.3 GHz CPUs (16 cores per CPU)
- 2 of the cabinets have Xeon E5-2695v4 Broadwell 2.1 GHz CPUs (18 cores per CPU)
- Aries network

67,456 cores in total

References

- *Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture* by Jon Stokes
- *Computer Architecture: A Quantitative Approach* by Hennessy and Patterson
- *Computer Organization and Design* by Patterson and Hennessy