中山大學 SUN YAT-SEN UNIVERSITY

国家超级计算广州中心 NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

Advanced Computer Architecture
高级计算机体系结构

第12讲：WSC & Interconnect

张献伟

xianweiz.github.io

DCS5367, 12/21/2021

Atomic Pair[原子对]

- Used in the MIPS processor and RISC-V
- RISC-V: load reserved/store conditional
  - Load reserved (lr): loads the contents of memory given by rs1 into rd and creates a reservation on that memory address
    - Also called load linked or load locked
  - Store conditional (sc): stores the value in rs2 into the memory address given by rs1
- Store conditional fails (writes a non-zero) if,
  - The reservation of the load is broken by a write to the same memory location
  - The processor does a context switch between the instructions

Atomic Pair (cont.)

- Instructions lr/sc are used in sequence
  - lr returns the initial value
  - sc returns 0 only if it succeeds
- Example: use lr/sc to implement an atomic exchange on the memory location specified by the contents of x1 with the value in x4
  - Anytime a processor intervenes and modifies the value in memory between the lr and sc, the sc returns a non-zero, causing the code sequence to try again

```
try: mov x3,x4      ;mov exchange value
     lr x2,x1       ;load reserved from
     sc x3,0(x1)    ;store conditional
     bnez x3,try    ;branch store fails
     mov x4,x2      ;put load value in x4?
```

Atomic exchange

```
try: lr x2,x1       ;load reserved 0(x1)
     addi x3,x2,1   ;increment
     sc x3,0(x1)    ;store conditional
     bnez x3,try    ;branch store fails
```

Atomic fetch-and-increment

Implementing Locks[实现锁]

- Once we have an atomic operation, we can use the coherence mechanism of a multiprocessor to implement spin locks
  - Locks that a processor continuously tries to acquire, spinning around a loop until it succeeds
- Simplest implementation of keeping the lock variables in memory (there were no cache coherence)
- Cache the locks using the coherence mechanism to maintain the lock value coherently

```
        addi x2,R0,#1
lockit: EXCH x2,0(x1)     ;atomic exchange
        bnez x2,locket    ;already locked?
```

```
lockit: ld x2,0(x1)       ;load of lock
        bnez x2,locket    ;not available-spin
        addi x2,R0,#1     ;load locked value
        EXCH x2,0(x1)     ;swap
        bnez x2,locket    ;branch if lock wasn't 0
```

Coherence-based Locks[基于一致性]

- Spins by doing reads on a local copy of the lock until it successfully sees that the lock is available
- Then, attempts to acquire the lock by doing a swap operation
  - All processes use a swap inst that reads the old value and stores 1 into the lock variable
  - The single winner will see the 0, and the losers will see a 1 that was placed there by the winner
  - The winning processor executes the code after the lock and, when finished, stores a 0 into the lock variable to release the lock

```
lockit: ld x2,0(x1)        ;load of lock
        bnez x2,locket     ;not available-spin
        addi x2,R0,#1      ;load locked value
        EXCH x2,0(x1)      ;swap
        bnez x2,locket     ;branch if lock wasn't 0
```

Coherence-based Locks (cont.)

- To lock a variable using an atomic swap
  - Once the processor with the lock stores a 0 into the lock (i.e., lock released), all other caches are invalidated and must fetch the new value to update their copy of the lock[锁释放->竞争]
  - One such cache gets the copy of the unlocked value (0) first and performs the swap[赢得竞争]
  - When the cache miss of other processors is satisfied, they find that the variable is already locked, so they must return to testing and spinning[继续]

```
lockit: ld x2,0(x1)       ;load of lock
        bnez x2,locket    ;not available-spin
        addi x2,R0,#1     ;load locked value
        EXCH x2,0(x1)     ;swap
        bnez x2,locket    ;branch if lock wasn't 0
```

Coherence-based Locks (cont.)

Step	P0	P1	P2	Coherence state of lock at end of step	Bus/directory activity
1	Has lock	Begins spin, testing if lock = 0	Begins spin, testing if lock = 0	Shared	Cache misses for P1 and P2 satisfied in either order. Lock state becomes shared.
2	Set lock to 0	(Invalidate received)	(Invalidate received)	Exclusive (P0)	Write invalidate of lock variable from P0.
3		Cache miss	Cache miss	Shared	Bus/directory services P2 cache miss; write-back from P0; state shared.
4		(Waits while bus/ directory busy)	Lock = 0 test succeeds	Shared	Cache miss for P2 satisfied.
5		Lock = 0	Executes swap, gets cache miss	Shared	Cache miss for P1 satisfied.
6		Executes swap, gets cache miss	Completes swap: returns 0 and sets lock = 1	Exclusive (P2)	Bus/directory services P2 cache miss; generates invalidate; lock is exclusive.
7		Swap completes and returns 1, and sets lock = 1	Enter critical section	Exclusive (P1)	Bus/directory services P1 cache miss; sends invalidate and generates write-back from P2.
8		Spins, testing if lock = 0			None

Languages' Memory Models[语言内存模型]

- Besides hardware, compilers can also reorder memory operations
  - Example: the program always prints a string of 100 '1's
  - Possible to optimize the code?
    - Loop-invariant code motion: move the write outside the loop
    - Dead store elimination: remove X = 0
  - These two programs are totally equivalent
    - Produce the same output

```
X = 0
for i in range(100):
  X = 1
  print X
```

```
X = 1
for i in range(100):
  print X
```


https://www.cs.utexas.edu/~bornholt/post/memory-models.html

Languages' Memory Models (cont.)

- Now suppose there's another thread running in parallel with the program, and it performs a single write to X
  - The first program
    - It can print strings like 11101111 …, so long as there's only one single zero (because it will reset X = 1 on the next iteration)
  - The second program
    - It can print strings like 1110000 …, where once it starts printing 0s it never goes back to 1s
  - The first can never print 1110000…; the second cannot print 11011111…

With parallelism, the compiler optimization no longer produces an equivalent program.

```
X = 0
for i in range(100):
  X = 1
  print X
```

```
X = 1
for i in range(100):
  print X
```

X = 0

X = 0

https://www.cs.utexas.edu/~bornholt/post/memory-models.html

Languages' Memory Models (cont.)

- Memory consistency at the program level
- The compiler optimization is effectively reordering
  - It's rearranging (and removing some) memory accesses in ways that may or may not be visible to programmers
- To preserve intuitive behavior, programming languages need memory models of their own,
  - To provide a contract to programmers about how their memory operations will be reordered

std::memory_order

Defined in header <atomic>

```
typedef enum memory_order {
    memory_order_relaxed,
    memory_order_consume,
    memory_order_acquire,                    (since C++11)
    memory_order_release,                    (until C++20)
    memory_order_acq_rel,
    memory_order_seq_cst
} memory_order;
enum class memory_order : /*unspecified*/ {
    relaxed, consume, acquire, release, acq_rel, seq_cst
};
inline constexpr memory_order memory_order_relaxed = memory_order::relaxed;
inline constexpr memory_order memory_order_consume = memory_order::consume;   (since C++20)
inline constexpr memory_order memory_order_acquire = memory_order::acquire;
inline constexpr memory_order memory_order_release = memory_order::release;
inline constexpr memory_order memory_order_acq_rel = memory_order::acq_rel;
inline constexpr memory_order memory_order_seq_cst = memory_order::seq_cst;
```


https://www.cs.utexas.edu/~bornholt/post/memory-models.html

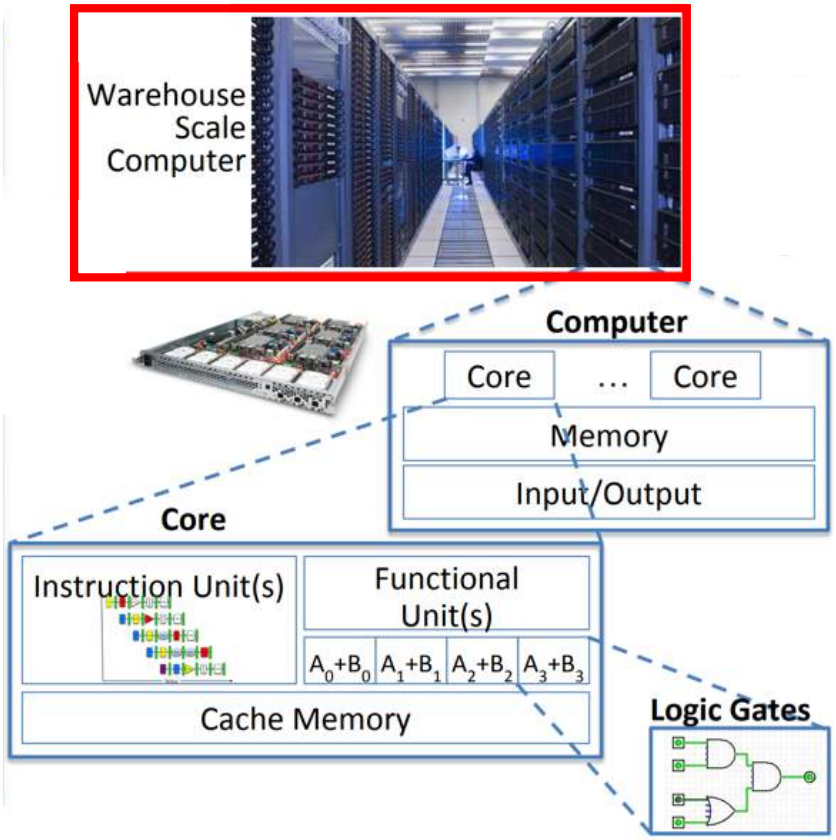
Summary

- Multiprocessors with thread-level parallelism
  - Sharing memory, having private caches
- Cache coherence
  - Snooping: Every cache block is accompanied by the sharing status of that block – all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary
  - Directory-based: A single location (directory) keeps track of the sharing status of a block of memory
- Memory consistency
  - Sequential consistency: maintains all four memory operation orderings (W→R, R→R, R→W, W→W)
  - Relaxed consistency: allows certain orderings to be violated
    - TSO, PSO, RC

Warehouse Scale Computing

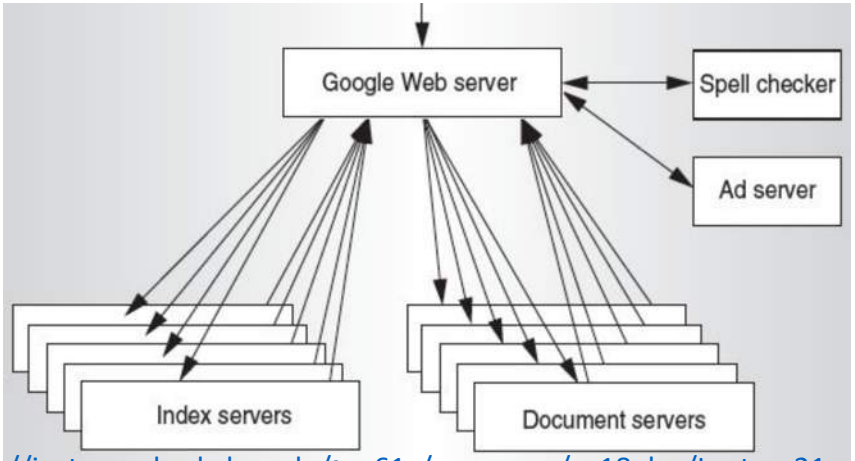
Parallelism[并行]

- Instruction-level parallelism (ILP)
  - Pipelining, speculation, OoO, …
- Data-level parallelism (DLP)
  - Vectors, GPU, AVX, …
- Thread-level parallelism (TLP)
  - Multithreading, multi-cores
- Request-level parallelism (RLP)
  - Parallelism among multi decoupled tasks



Request-level Parallelism[请求级并行]

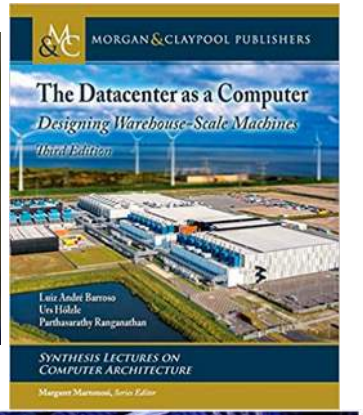
- Hundreds or thousands of requests per second
  - Not your laptop or cell-phone, but popular Internet services like web search, social networking, …
  - Such requests are largely independent
    - Often involve read-mostly databases
    - Rarely involve strict read–write data sharing or synchronization across requests
- Computation easily partitioned within a request and across different requests



https://inst.eecs.berkeley.edu/~cs61c/resources/su18_lec/Lecture21.pdf

Luiz André Barroso

- Google Fellow & former VP of Engineering
- ACM-IEEE CS Eckert-Mauchly Award
  - For pioneering the design of **warehouse-scale computing** and driving it from concept to industry
- Award lecture:
  - Q: What to focus on while educating the next generation of computer engineers and scientists?
  - A: … **being a good programmer** is really important, doesn't matter what you are where you are in the field of computer science; make sure you are **graduating good programmers** whatever that means for you …



Warehouse-scale Computer[仓储规模]

- Massive scale datacenters: 10,000 to 100,000 servers + networks to connect them together
  - Emphasize cost-efficiency
  - Attention to power: distribution and cooling
  - (relatively) homogeneous hardware/software
- Single gigantic machine
- Offer very large applications (Internet services): search, voice search (Siri), social networks, video sharing
- Very highly available: < 1 hour down/year
  - Must cope with failures common at scale
- "…WSCs are no less worthy of the expertise of computer systems architects than any other class of machines" (Barroso and Hoelzle, 2009)

Warehouse-scale Computer (cont.)

- Differences with **HPC** "clusters":
  - Clusters have higher performance processors and network
    - HPC apps are more interdependent and communicate more frequently
  - Clusters emphasize TLP and DLP, WSCs emphasize request-level parallelism
    - HPC emphasizes latency to complete a single task vs. bandwidth to complete many independent tasks
    - HPC clusters tend to have long-running jobs that keep the servers fully utilized
- Differences with **datacenters**:
  - Datacenters consolidate different machines and software into one location
  - Datacenters emphasize virtual machines and hardware heterogeneity in order to serve varied customers

Design Goals of WSC[设计目标]

- WSCs share many goals and requirements with servers
  - Cost-performance
    - Work done per $
  - Energy efficiency
    - Work done per J
  - Dependability via redundancy
    - 99.99% of availability, i.e., less 1h down per year
  - Network I/O
    - Good interface to external world
  - Both interactive and batch processing workloads
    - Interactive: e.g., search and social networking with Billions of users
    - Batch: calculate metadata useful to such services, e.g., MapReduce jobs to convert crawled pages into search indices

Design Goals of WSC (cont.)

- Unique to WSCs
  - Ample parallelism:
    - Batch apps: many independent data sets with independent processing (Data-Level and Request-Level Parallelism)
  - Scale and its Opportunities/Problems
    - Relatively small number of WSC make design cost expensive and difficult to amortize
    - But price breaks are possible from purchases of very large numbers of commodity servers
    - Must also prepare for high component failures
  - Operational Costs Count:
    - Cost of equipment purchases << cost of ownership
  - Location counts
  - Computing efficiently at low utilization
    - WSC servers are rarely fully utilized

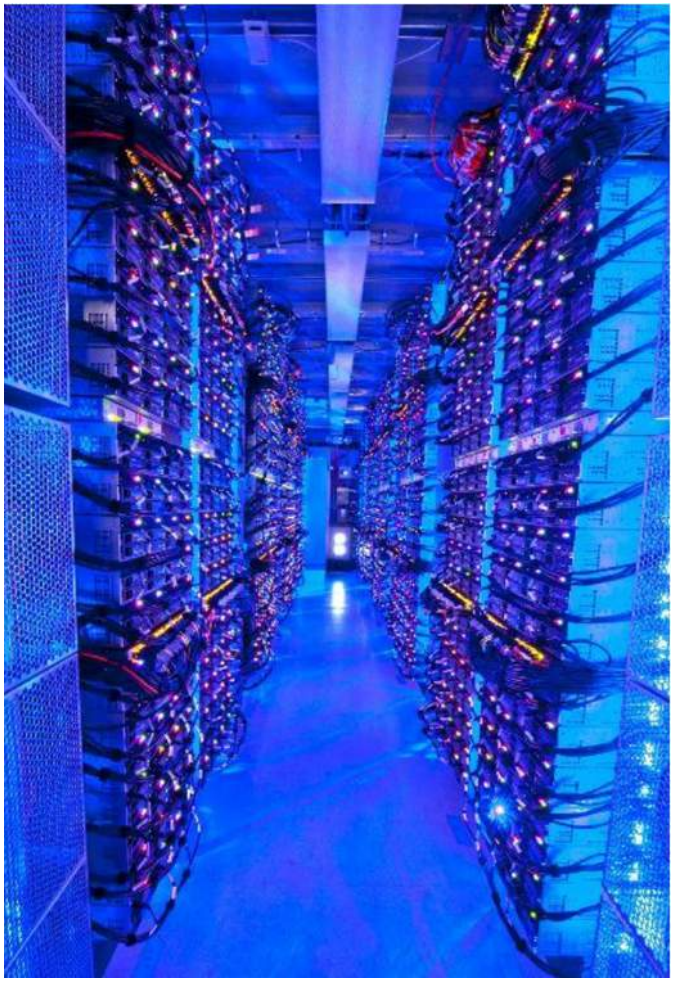
Google's Oregon WSC

https://inst.eecs.berkeley.edu/~cs61c/resources/su18_lec/Lecture21.pdf

Containers in WSCs[集装箱]

Inside WSC

Inside Container



Programming Models for WSCs[编程模型]

- Batch processing framework: MapReduce
  - The MapReduce runtime environment schedules map tasks and reduce tasks to the nodes of a WSC
  - MapReduce can be thought of as a generalization of the SIMD operation
    - Except that a function to be applied is passed to the data
- Map: (in_key, in_value) → list(interm_key, interm_val)
  - Slice data into "shards" or "splits" and distribute to workers
  - Compute set of intermediate key/value pairs
- Reduce: (interm_key, list(interm_value)) → list(out_value)
  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)

MapReduce Example

- Map phase: (doc name, doc contents) → list(word, count)

// "I do I learn"" → [("I",1),("do",1),("I",1),("learn",1)]

```
map(key, value):
    for each word w in value:
      emit(w, 1)
```

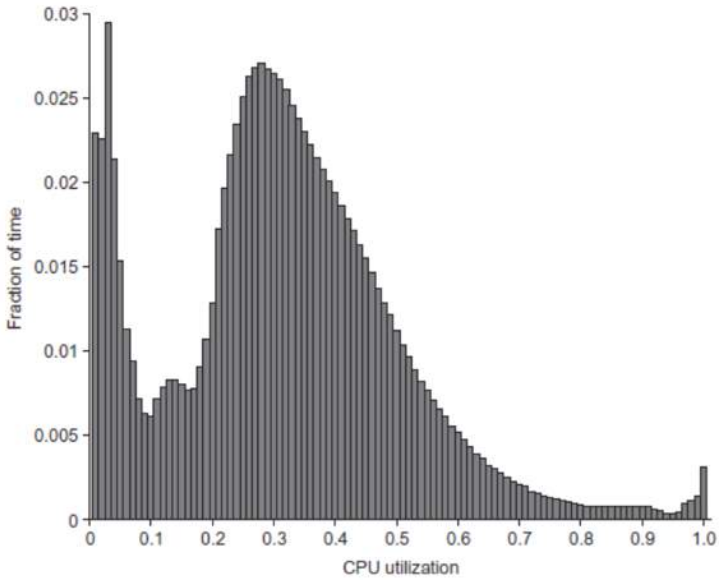
- Reduce phase: (word, list(count)) → (word, count_sum)

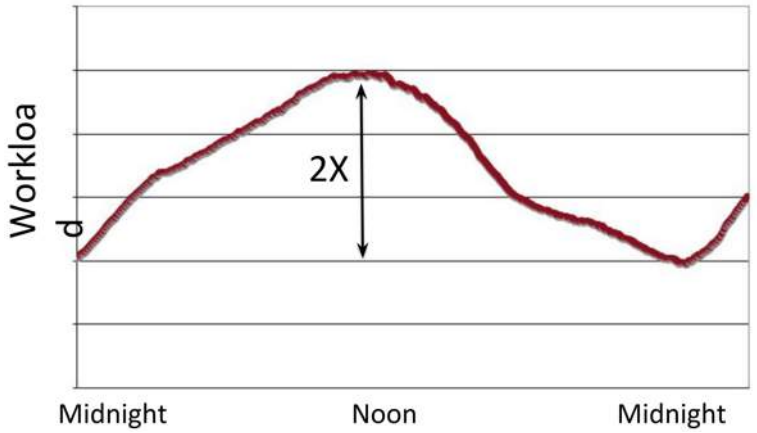
// ("I", [1,1]) → ("I",2)

```
reduce(key, values):
   result = 0
   for each v in values:
      result += v
   emit(key, result)
```

WSC Software[软件]

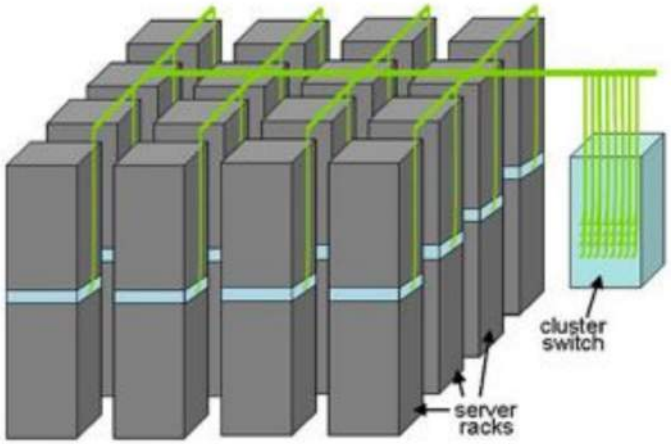
- Must scale up and down gracefully in response to varying demand
  - Varying workloads impact Availability
- Must cope with failures gracefully
  - High failure rate impact Reliability Availability
- More elaborate hierarchy of memories, failure tolerance, workload accommodation makes WSC software development more challenging than software for single computer



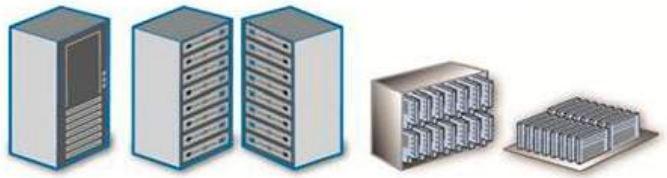


Equipment Inside a WSC

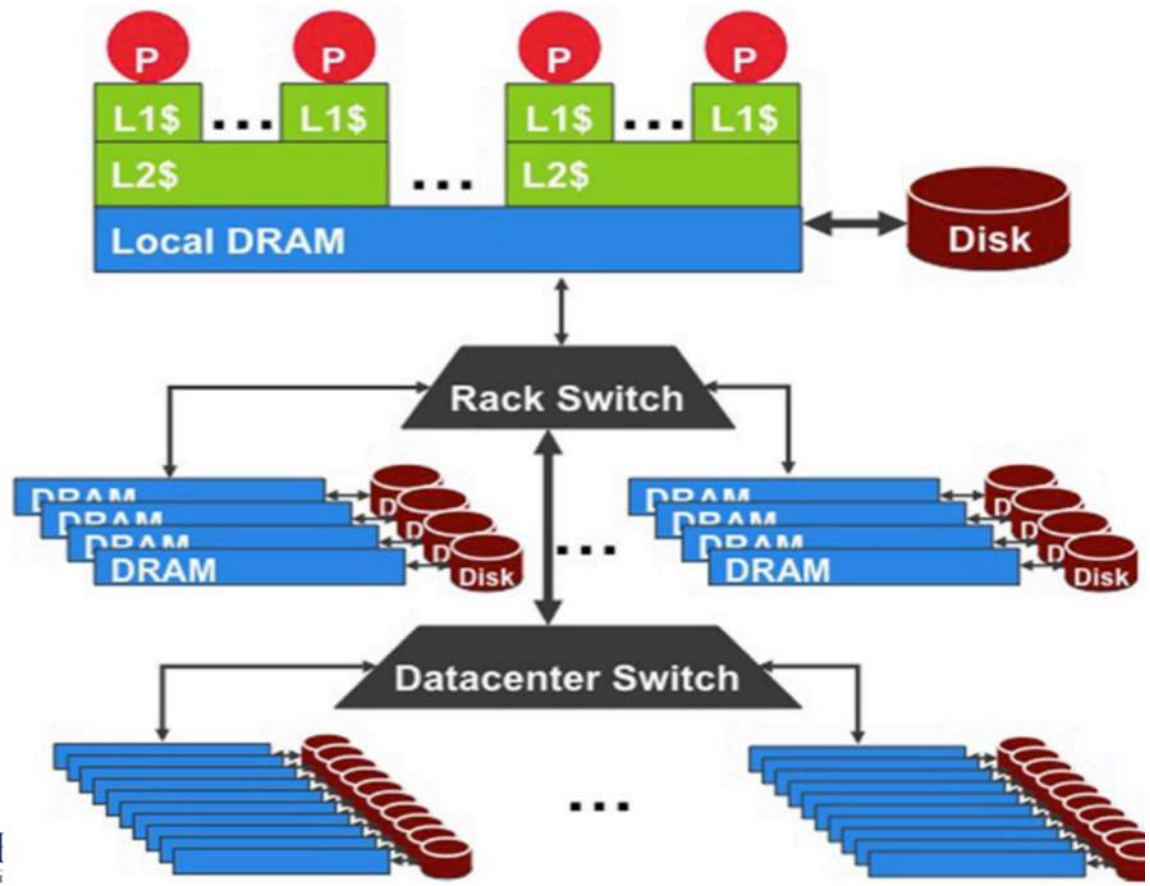
- **Server**[服务器]
  - 1 ¾ inches high “1U”
  - 8 cores, 16 GB DRAM, 4x1 TB disk
- **Rack[机架]**
  - 7 foot
  - 40-80 servers + Ethernet local area network (1-10 Gbps) switch in middle (“rack switch”)
- **Array** (a.k.a., cluster)[集群]
  - 16-32 server racks + larger local area network switch (“array switch”)
    - Expensive switch (10X bandwidth, 100x cost)



Server, Rack, Array



WSC Architecture[架构]

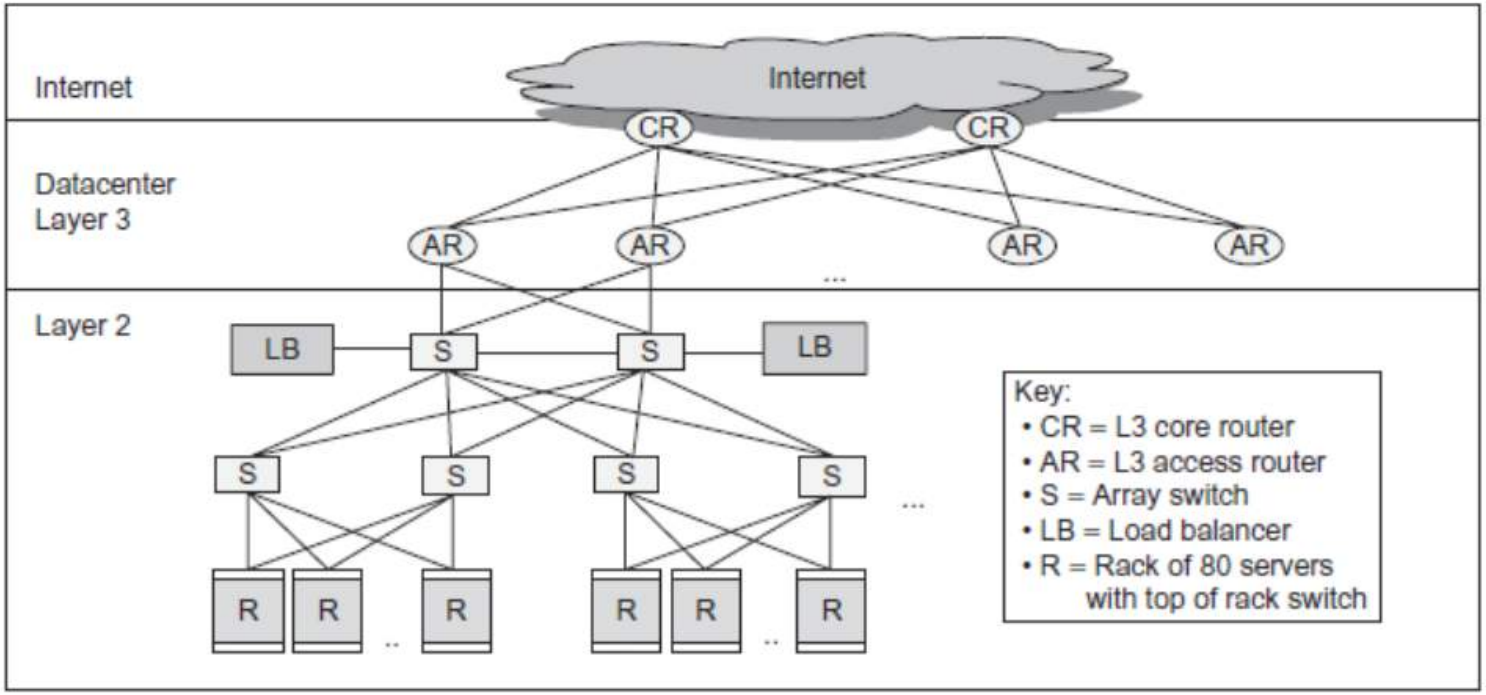


- 1U Server:
  - DRAM: 64GB, 100ns
  - Disk: 10TB, 10ms
- Rack (80 severs):
  - DRAM: 5TB, 300µs
  - Disk: 800TB, 11ms
- Array (30 racks):
  - DRAM: 150TB, 500µs
  - Disk: 24PB, 12ms

Lower latency to DRAM in another server than local disk
Higher bandwidth to local disk than to DRAM in another server

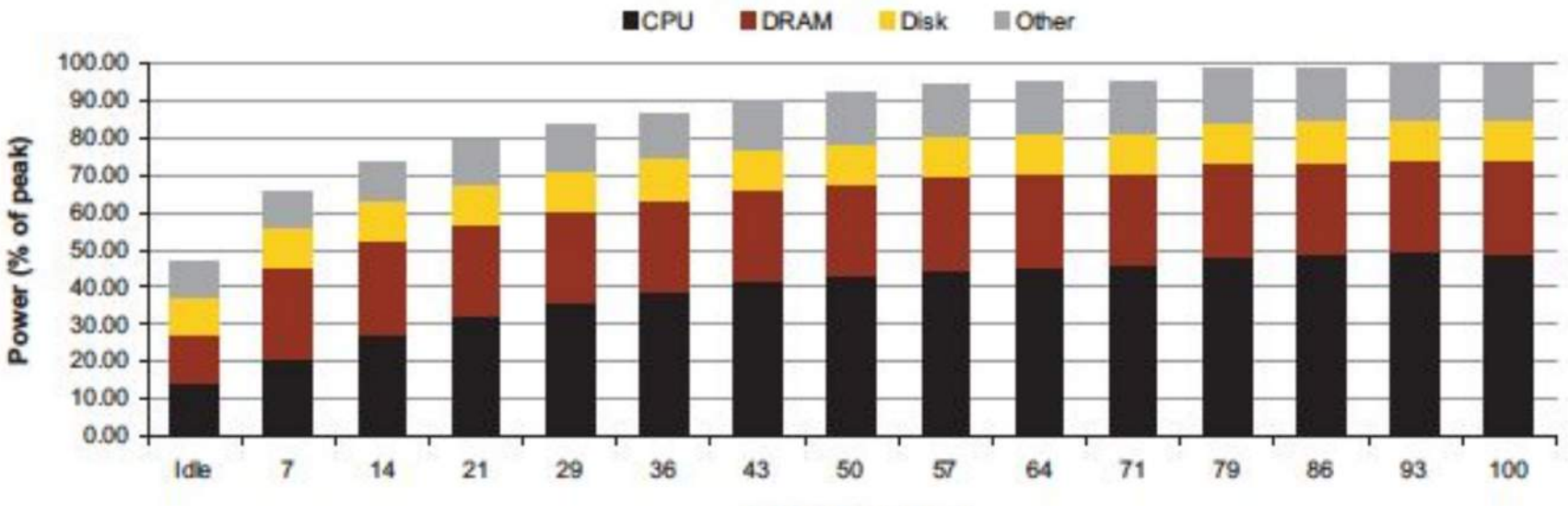
Network[网络]

- The WSC needs 40 arrays to reach 100K servers
  - One more level in the networking hierarchy
- Conventionally, Layer 3 routers to connect the arrays together and to the Internet



Power vs. Server Utilization[能耗]

- Server power usage as load varies idle to 100%
- Uses ½ peak power when idle!
- Uses ⅔ peak power when 10% utilized! 90%@ 50%!
- Most servers in WSC utilized 10% to 50%
- Goal should be Energy-Proportionality: % peak load = % peak energy



https://inst.eecs.berkeley.edu/~cs61c/resources/su18_lec/Lecture21.pdf

Power Usage Effectiveness[电源使用效率]

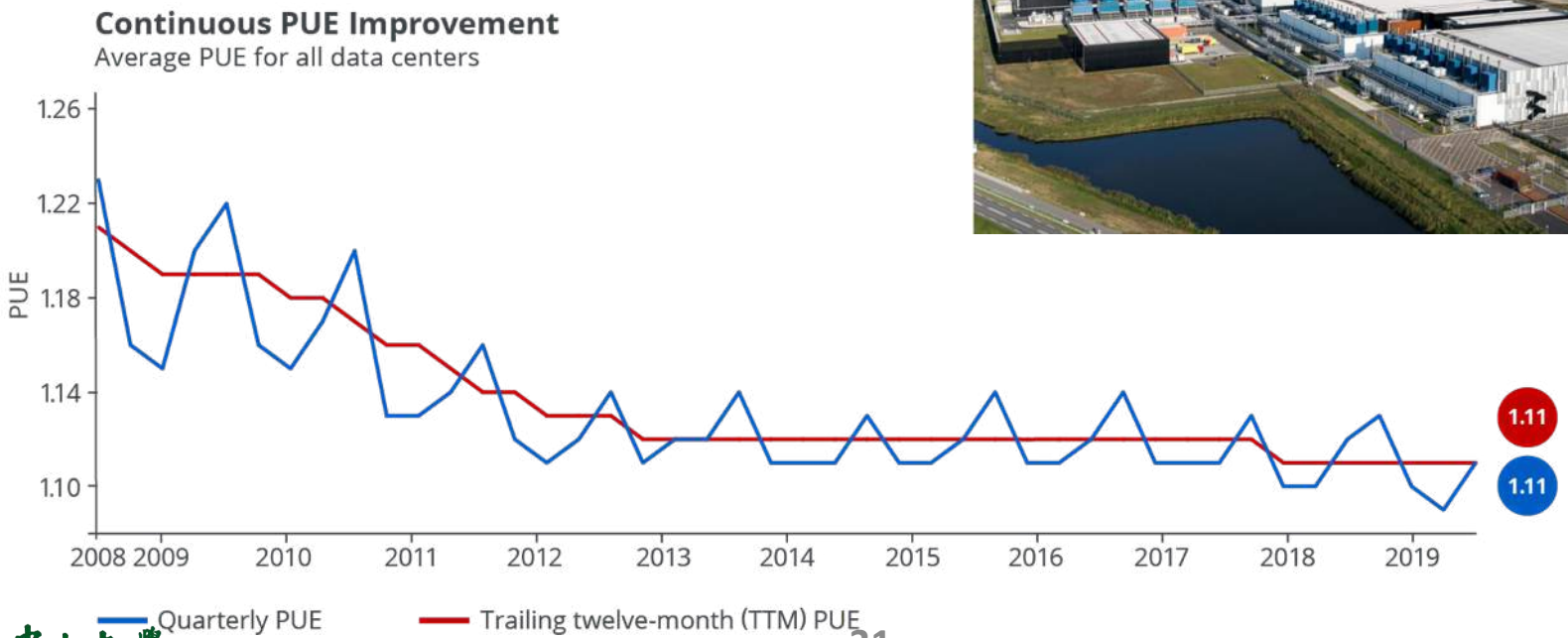
- Overall WSC Energy Efficiency: amount of computational work performed divided by the total energy used in the process
- Power Usage Effectiveness (PUE):

$$\frac{\text{Total Building Power}}{\text{IT equipment Power}}$$

  - Power efficiency measure for WSC, not including efficiency of servers, networking gear
  - Power usage for non-IT equipment increases PUE
  - 1.0 is perfection, higher numbers are worse
  - Google WSC's PUE: 1.2

Power Usage Effectiveness (cont.)

- Average PUE of the 15 google WSCs 2008 – 2017
- Google's Belgium WSC PUE: 1.09
  - Careful air flow handling
  - Elevated cold aisle temperatures
  - Use of free cooling
  - Per-server 12-V DC UPS

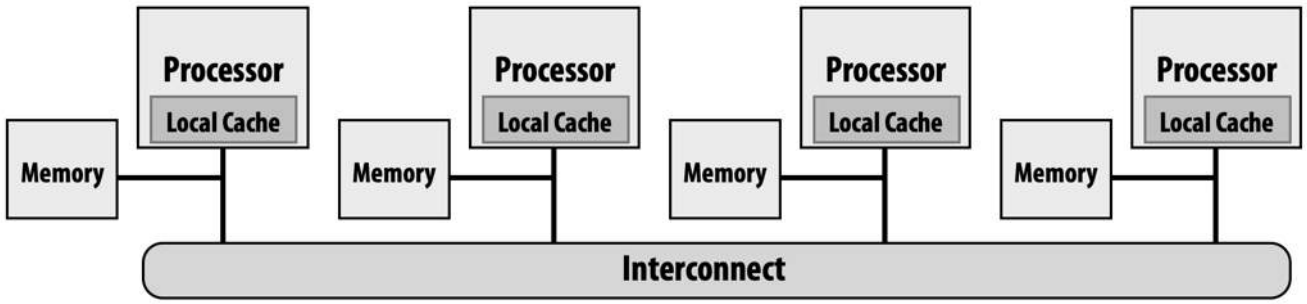


https://inst.eecs.berkeley.edu/~cs61c/resources/su18_lec/Lecture21.pdf

Interconnection Network

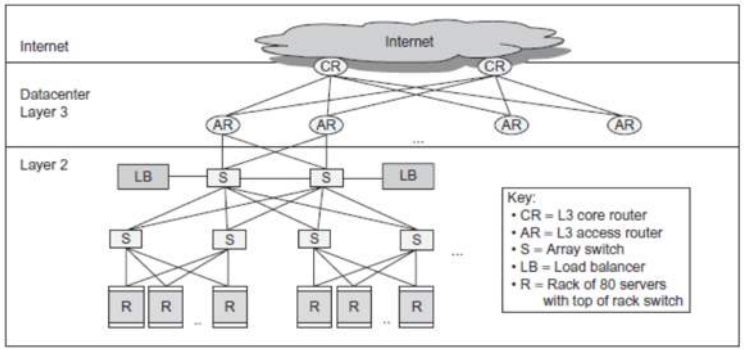
Interconnection Networks[互联网络]

- An **Interconnection Network** (ICN) is a programmable system that transports data between terminals
  - To hold our parallel machines together, at the core of parallel computer architecture
  - Share basic concept with LAN/WAN, but very different trade-offs due to very different time scale/requirements
- Interconnection networks can be grouped into four domains[分类]
  - Depending on number and proximity of devices to be connected



Different Scales of Networks

- **Local-Area Networks**[局域网络]
  - Interconnect autonomous computer systems
  - Machine room or throughout a building or campus
  - Hundreds of devices interconnected (1,000s with bridging)
  - Maximum interconnect distance
    - Few meters to tens of kilometers
    - Example (most popular): Ethernet, with 10 Gbps over 40Km
- **Wide-Area Networks**[广域网络]
  - Interconnect systems distributed across the globe
  - Internetworking support is required
  - Millions of devices interconnected
  - Maximum interconnect distance
    - many thousands of kilometers



Different Scales of Networks (cont.)

- **System-Area Networks**[系统区域网络]
  - Interconnects within one “machine”
    - Interconnect in a multi-processor system
    - Interconnect in a supercomputer
- Hundreds to thousands of devices interconnected
  - Tianhe-2 supercomputer (16K nodes, each with 2 12-core processors)
- Maximum interconnect distance
  - Fraction to tens of meters (typical)
  - A few hundred meters (some)
    - InfiniBand: 120 Gbps over a distance of 300m

https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa15/cse610/slides/07-icn.pdf

Different Scales of Networks (cont.)

- **On-Chip Networks**[片上网络]
  - Interconnect within a single chip
- Devices are micro-architectural elements
  - Caches, directories, processor cores
- Currently, designs with 10s of devices are common
  - Ex: IBM Cell, Intel multicores, Tile processors
- Projected systems with 100s of devices on the horizon
- Proximity: millimeters

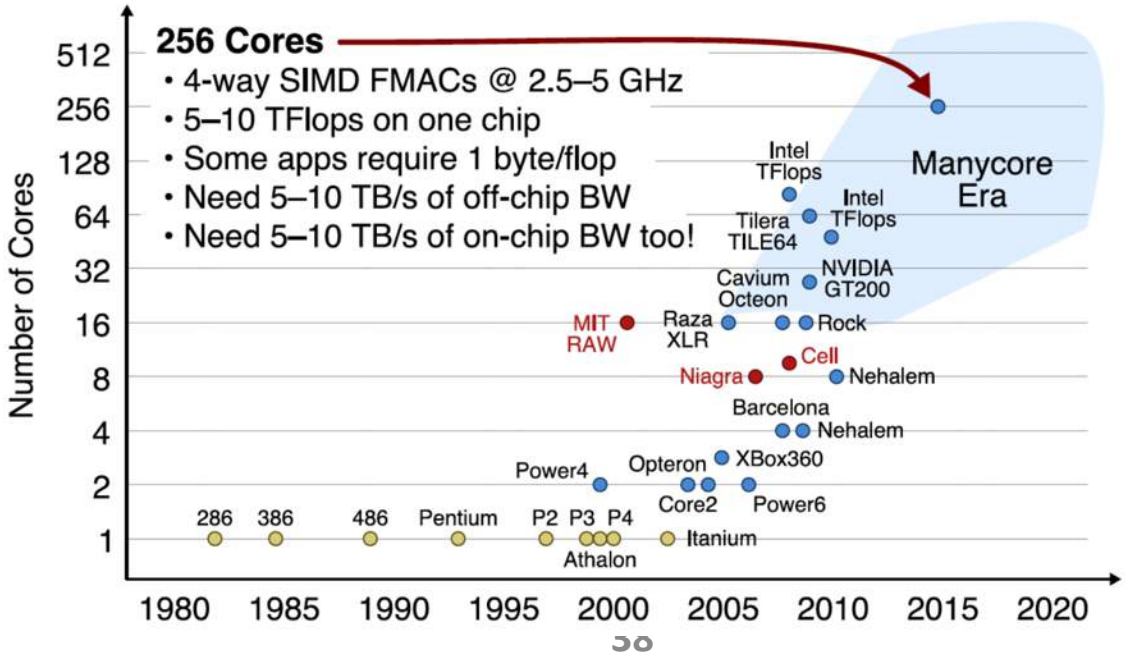
We are concerned with On-Chip and System-Area Networks

Why Study Interconnects?

- They provide external connectivity from system to outside world
  - Also, connectivity within a single computer system at many levels
    - I/O units, boards, chips, modules and blocks inside chips
- Interconnection networks should be well designed
  - To transfer the maximum amount of information
  - Within the least amount of time (and cost, power constraints)
  - So as not to bottleneck the system
- Application: managing communication can be critical to performance

Why Study Interconnects? (cont.)

- Trends: high demand on communication bandwidth
  - increased computing power and storage capacity
  - switched networks are replacing buses
- Computer architects/engineers must <u>understand interconnect problems and solutions</u> in order to more effectively design and evaluate systems



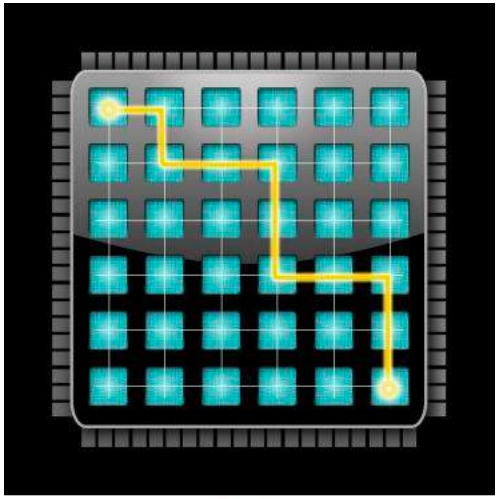
https://ece757.ece.wisc.edu/lect09-interconnects-1-intro.pdf

Basic Definitions[基本定义]

- An interconnection network is a graph of ***nodes*** interconnected using ***channels***
- ***Node****[节点]*: a vertex in the network graph
  - ***Terminal*** nodes: where messages originate and terminate
  - ***Switch*** (***router***) nodes: forward messages from in ports to out ports
  - ***Switch degree***: number of in/out ports per switch
- ***Channel****[信道]*: an edge in the graph
  - i.e., an ordered pair (x,y) where x and y are nodes
  - Channel = link (transmission medium) + transmitter + receiver
  - ***Channel width***: w = number of bits transferred per cycle
  - ***Phit*** (physical unit or digit): data transferred per cycle
  - ***Signaling rate***: f = number of transfer cycles per second
  - ***Channel bandwidth***: b = w × f

Basic Definitions (cont.)

- ***Path*** (or ***route***): a sequence of channels, connecting a source node to a destination node
- ***Minimal Path***: a path with the minimum number of channels between a source and a destination
  - Rxy = set of all minimal paths from x to y
- ***Network Diameter***: longest minimal path over all (source, destination) pairs



Source: MIT

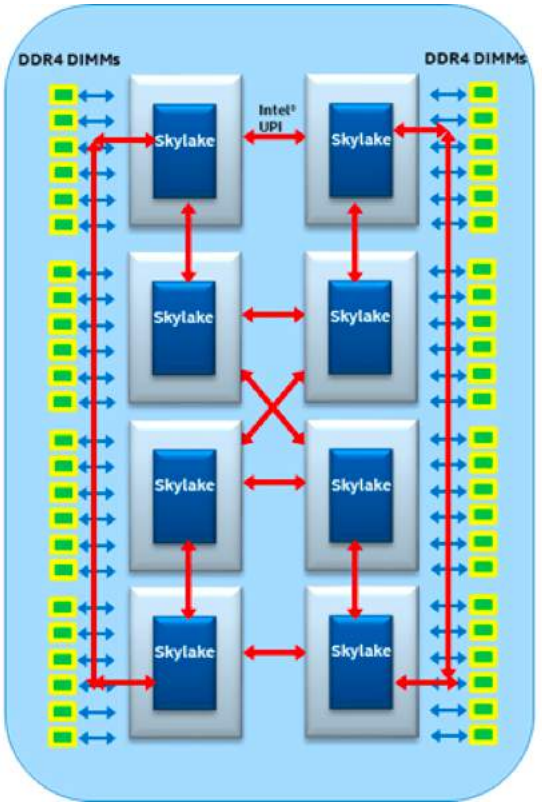
https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa15/cse610/slides/07-icn.pdf

ICN Design Considerations[设计考虑]

- Application requirements
  - Number of terminals or ports to support
  - Peak bandwidth of each terminal
  - Average bandwidth of each terminal
  - Latency requirements
  - Message size distribution
  - Expected traffic patterns
  - Required quality of service
  - Required reliability and availability
- Job of an interconnection network is to transfer information from source node to dest. node in support of network transactions that realize the application
  - Latency as small as possible
  - As many concurrent transfers as possible
  - Cost as low as possible

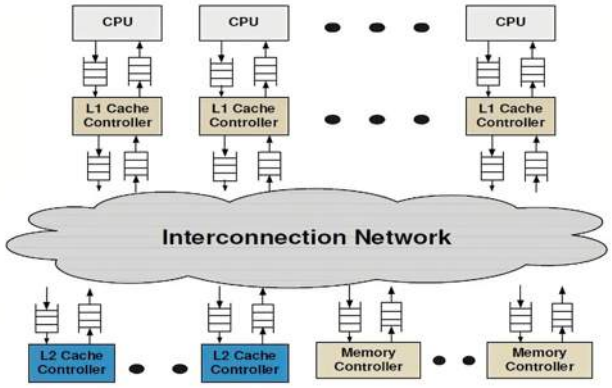
ICN Design Considerations (cont.)

- Example requirements for a coherent processor-memory interconnect
  - Processor ports 1-2048
  - Memory ports 1-4096
  - Peak BW 8 GB/s
  - Average BW 400 MB/s
  - Message Latency 100 ns
  - Message size 64 or 576 bits
  - Traffic pattern arbitrary
  - Quality of service none
  - Reliability no message loss
  - Availability 0.999 to 0.99999



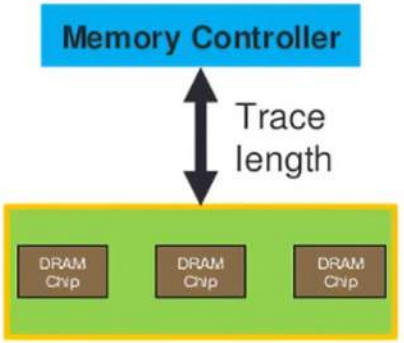
ICN Design Considerations (cont.)

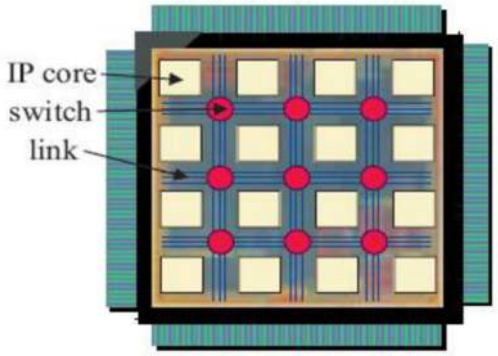
- Technology constraints
  - Signaling rate
  - Chip pin count (if off-chip networking)
  - Area constraints (typically for on-chip networking)
  - Chip cost
  - Circuit board cost (if backplane boards needed)
  - Signals per circuit board
  - Signals per cable
  - Cable cost
  - Cable length
  - Channel and switch power constraints
  - …



Off-chip vs. On-chip ICNs[片外 vs. 片上]

- Off-chip: I/O bottlenecks
  - Pin-limited bandwidth
  - Inherent overheads of off-chip I/O transmission
- On-chip
  - Wiring constraints
    - Metal layer limitations
    - Horizontal and vertical layout
    - Short, fixed length
    - Repeater insertion limits routing of wires
      - Avoid routing over dense logic
      - Impact wiring density
  - Power
    - Consume 10-15% or more of die power budget
  - Latency
    - Different order of magnitude
    - Routers consume significant fraction of latency



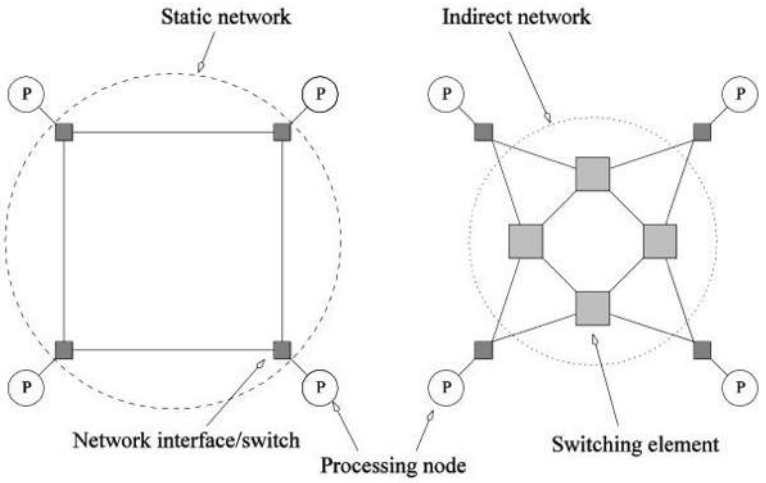


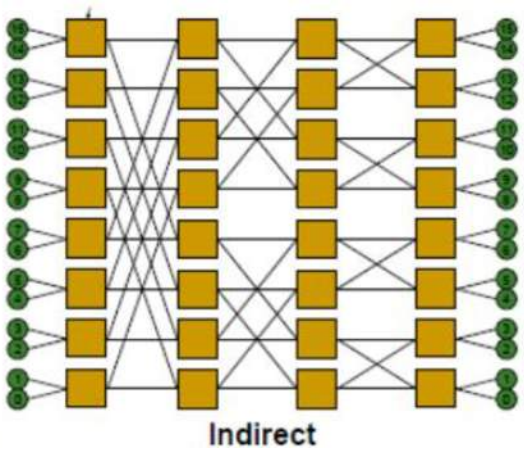
Main Aspects of an ICN[主要因素]

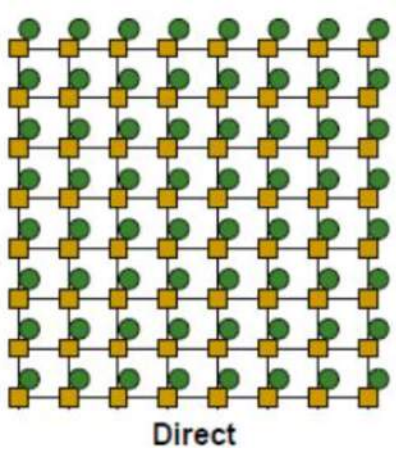
- **Topology**[拓扑]
  - Static arrangement of channels and nodes in a network
- **Routing**[寻路]
  - Determines the set of paths a message/packet can follow
- **Flow control**[流控]
  - Allocating network resources (channels, buffers, etc.) to packets and managing contention
- **Switch microarchitecture**[交换机微架构]
  - Internal architecture of a network switch
- **Network interface**[网络接口]
  - How to interface a terminal with a switch
- **Link architecture**[链接架构]
  - Signaling technology and data representation on the channel

Types of Topologies[拓扑类型]

- Direct[直接拓扑结构]
  - Each router is associated with a terminal node
  - All routers are sources and destinations of traffic
- Indirect[非直接拓扑结构]
  - Routers are distinct from terminal nodes
  - Terminal nodes can source/sink traffic
  - Intermediate nodes switch traffic between terminal nodes

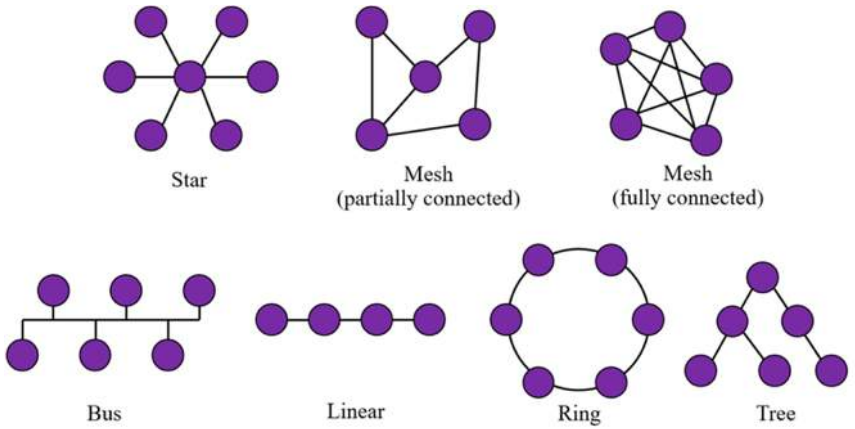






Network Topologies[拓扑]

- Blocking vs. Non-Blocking
  - If connecting any permutation of sources & destinations is possible, network is non-blocking; otherwise network is blocking
- A variety of network topologies have been proposed and implemented
  - These topologies tradeoff performance for cost
  - Commercial machines often implement hybrids of multiple topologies for reasons of packaging, cost, and available components



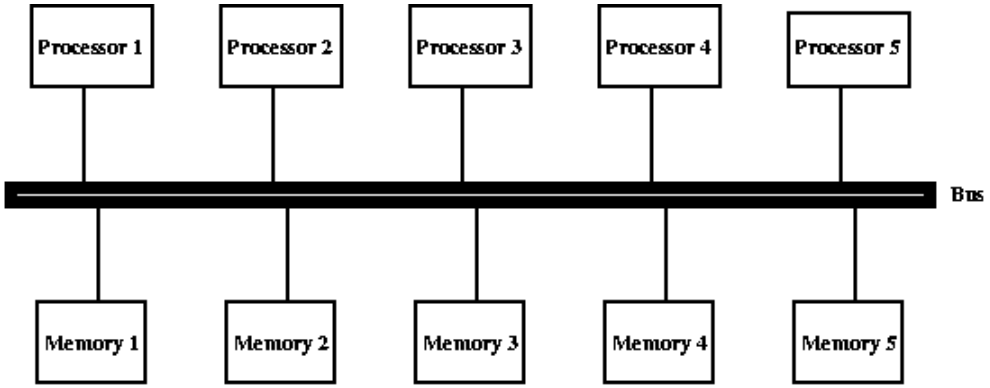
http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Interconnection%20Network.pdf

Metrics for Comparing Topologies[指标]

- **Switch degree**[交换度]
  - Proxy for switch complexity
- **Hop count**[跳数] (average and worst case)
  - Proxy for network latency
- **Maximum channel load**[最大通道负载]
  - A proxy for hotspot load
- **Bisection bandwidth**[对半带宽]
  - Proxy for maximum traffic a network can support under a uniform traffic pattern
- **Path diversity**[路径多样性]
  - Provides routing flexibility for load balancing and fault tolerance
  - Enables better congestion avoidance

Topologies: Buses[总线]

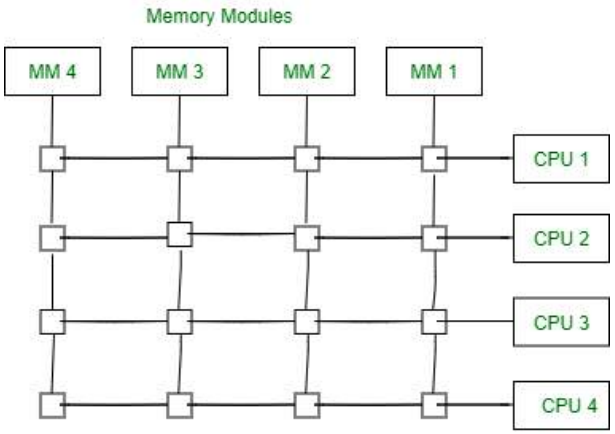
- All processors access a common bus for exchanging data
- The distance between any two nodes is O(1) in a bus. The bus also provides a convenient broadcast media
- However, the bandwidth of the shared bus is a major bottleneck
- Typical bus based machines are limited to dozens of nodes
  - Sun Enterprise servers and Intel Pentium based shared-bus multiprocessors are examples of such architectures

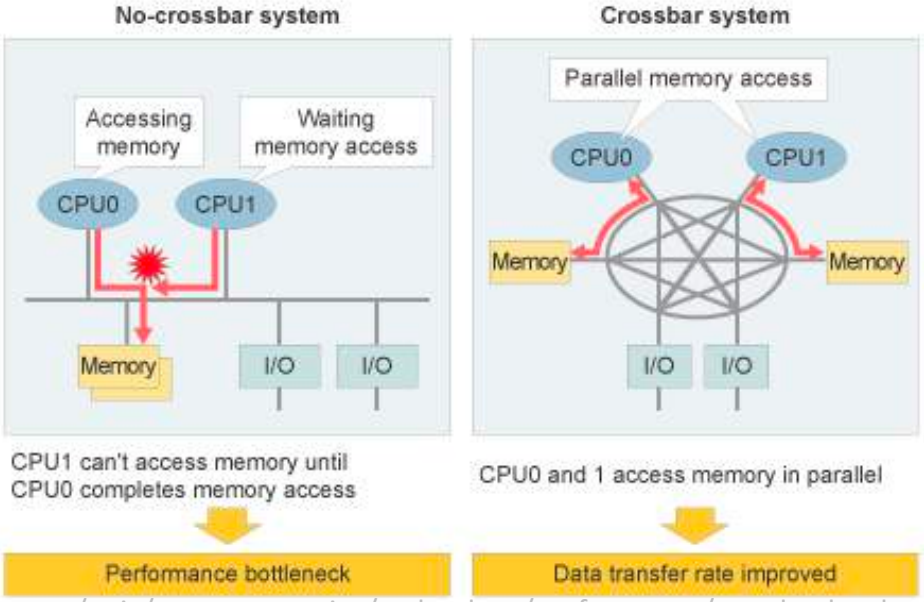


http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Interconnection%20Network.pdf

Topologies: Crossbars[交叉]

- A crossbar network uses an *p*×*m* grid of switches to connect *p* inputs to *m* outputs in a non-blocking manner
- The cost of a crossbar of *p* processors grows as O(p2)
  - This is generally difficult to scale for large values of *p*
  - Examples of machines that employ crossbars include the Sun Ultra HPC 10000 and the Fujitsu VPP500



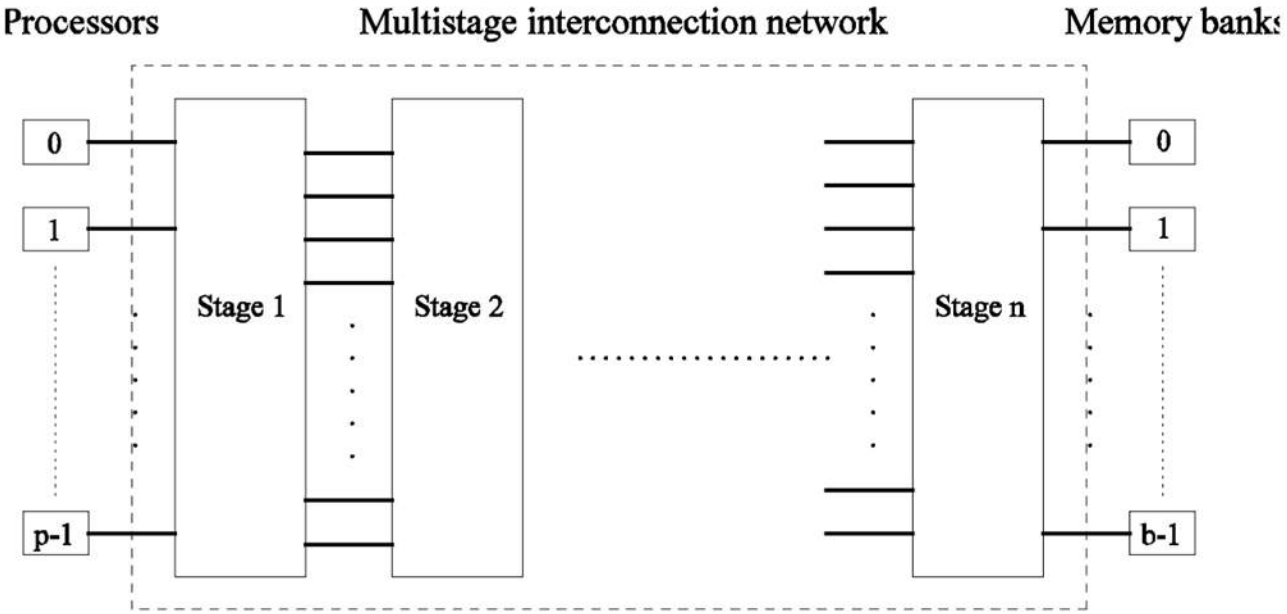


https://www.fujitsu.com/global/products/computing/servers/unix/sparc-enterprise/technology/performance/crossbar.html

http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Interconnection%20Network.pdf

Topologies: Multistage[多级]

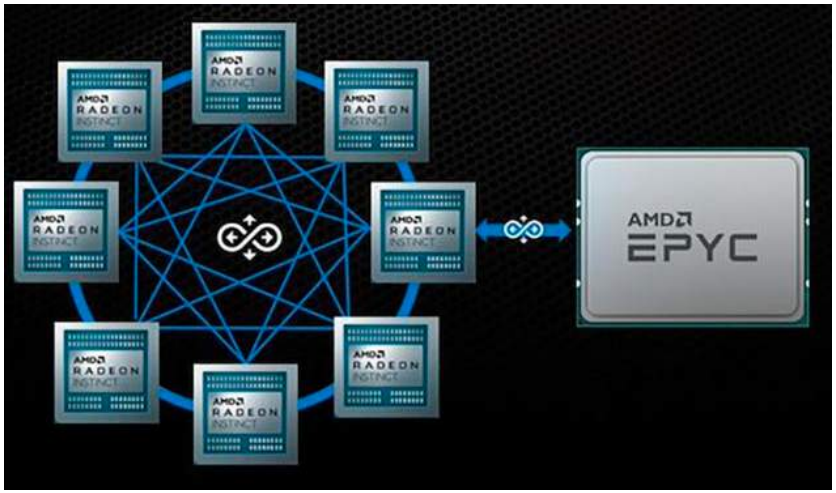
- Multistage interconnects strike a compromise between Buses and Crossbars
  - Crossbars have excellent performance scalability but poor cost scalability
  - Buses have excellent cost scalability, but poor performance scalability

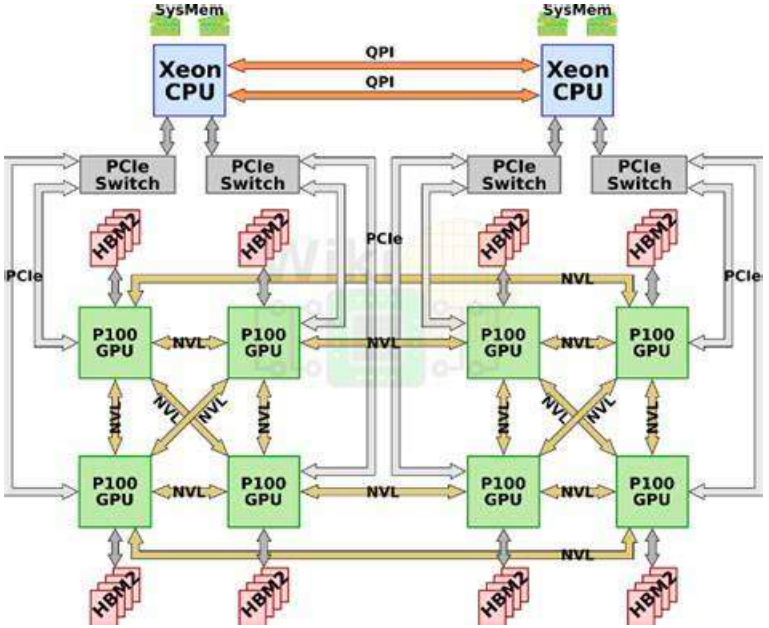


http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Interconnection%20Network.pdf

Topologies: Fully Connected[全连]

- Each processor is connected to every other processor
- The number of links in the network scales as O(p2)
- While the performance scales very well, the hardware complexity is not realizable for large values of p
- In this sense, these networks are static counterparts of crossbars

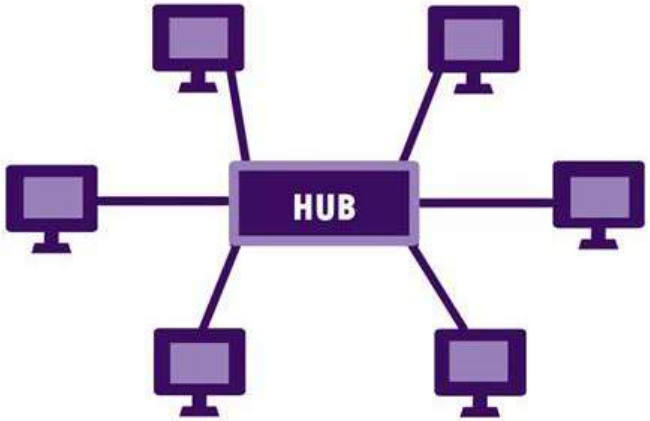


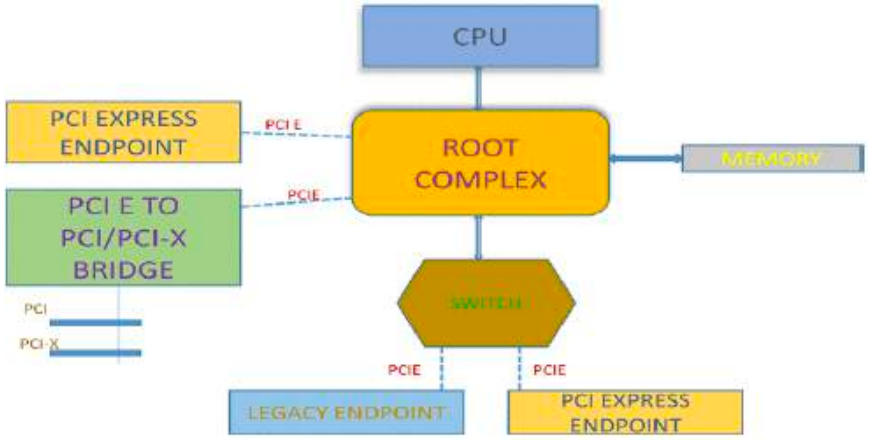


http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Interconnection%20Network.pdf

Topologies: Star Connected[星型]

- Every node is connected only to a common node at the center
- Distance between any pair of nodes is O(1)
- However, the central node becomes a bottleneck
- In this sense, star connected networks are static counterparts of buses

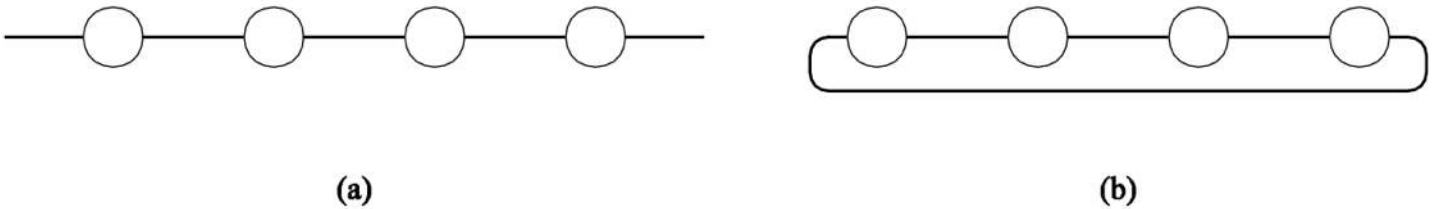




http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Interconnection%20Network.pdf

Topologies: Linear Arrays, Meshes, …

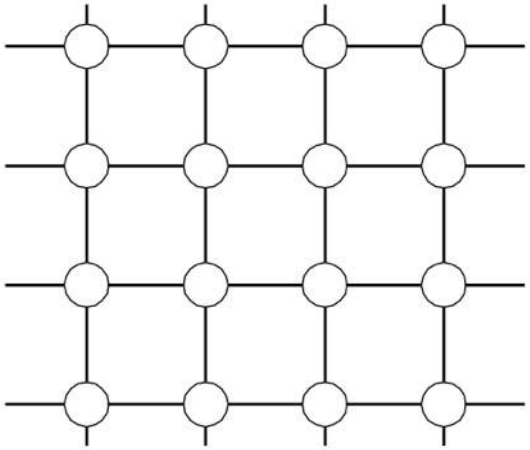
- In a **linear array**, each node has two neighbors, one to its left and one to its right
  - If the nodes at either end are connected, we refer to it as a 1-D torus or a ring
- A generalization to 2 dimensions has nodes with 4 neighbors, to the north, south, east, and west
- A further generalization to *d* dimensions has nodes with *2d* neighbors
  - A special case of a d-dimensional mesh is a hypercube. Here, d = log p, where p is the total number of nodes



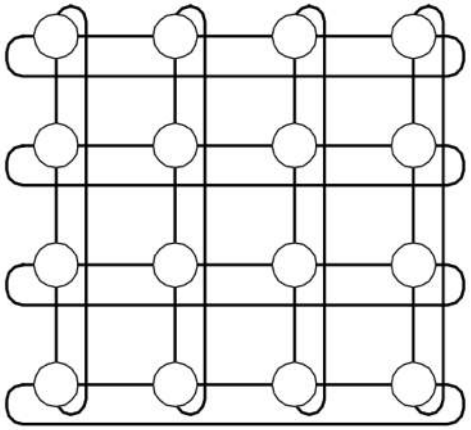
(a) with no wraparound links; (b) with wraparound link.

Topologies: Mesh[网格]

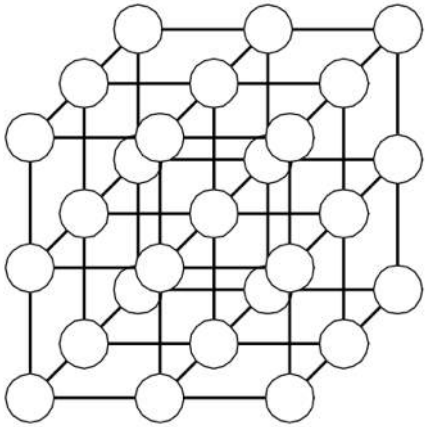
- Two and three dimensional meshes
  - (a) 2-D mesh with no wraparound
  - (b) 2-D mesh with wraparound link (2-D torus)
    - Mesh is not symmetric on edges: performance very sensitive to placement of task on edge vs. middle
    - Torus avoids this problem
  - (c) a 3-D mesh with no wraparound



(a)

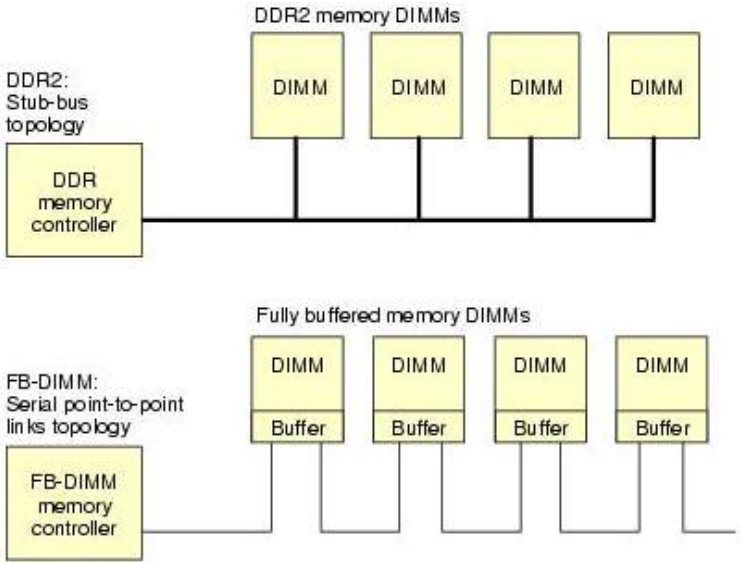


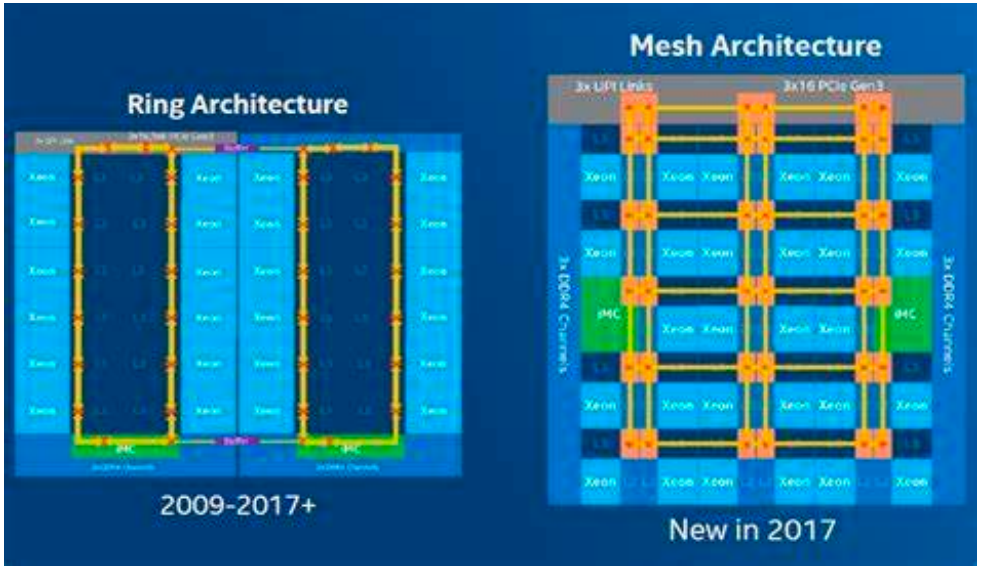
(b)



(c)

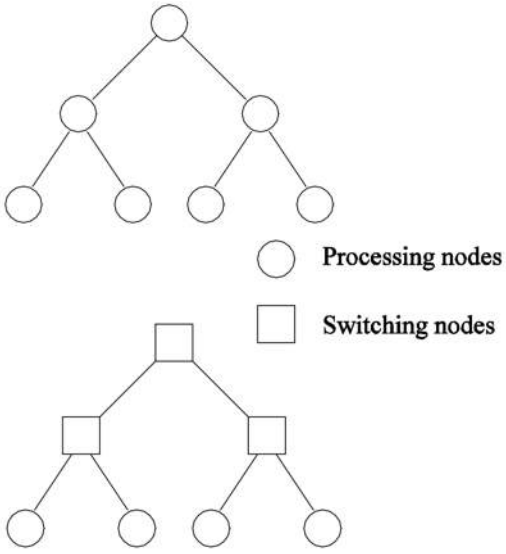
Examples

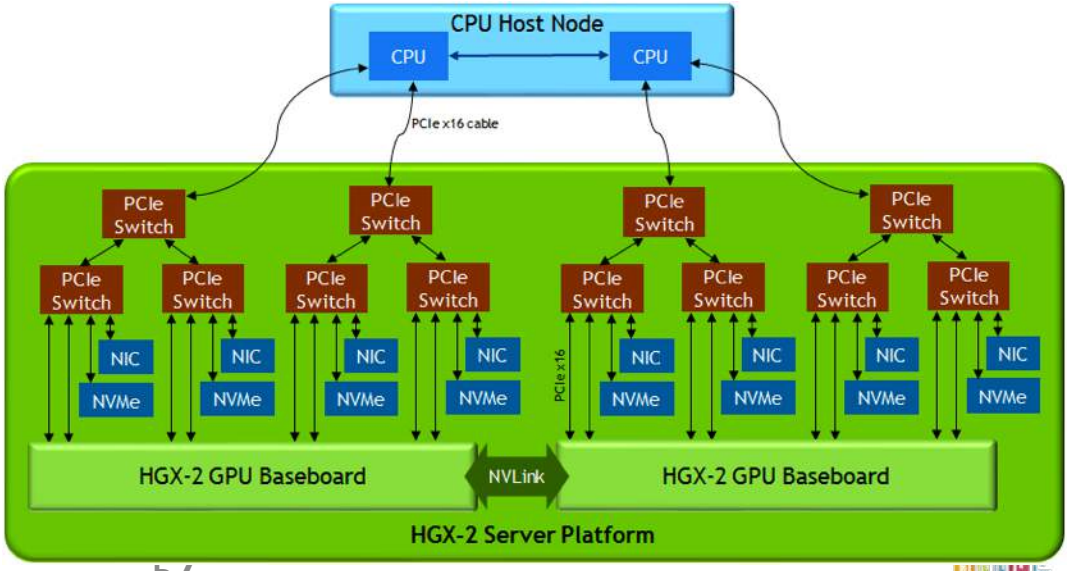

DDR2 memory DIMMs
DDR2:
Stub-bus
topology
DIMM
DIMM
DIMM
DIMM
DDR
memory
controller
Fully buffered memory DIMMs
FB-DIMM:
Serial point-to-point
links topology
DIMM
DIMM
DIMM
DIMM
Buffer
Buffer
Buffer
Buffer
FB-DIMM
memory
controller


Ring Architecture
Mesh Architecture
2009-2017+
New in 2017

Topologies: Tree[树型]

- Diameter and average distance logarithmic
  - *k*-ary tree, height = log*k* N
  - Address specified *d*-vector of radix *k* coordinates describing path down from root
  - Route up to common ancestor and down
- Trees can be laid out in 2D with no wire crossings
  - This is an attractive property of trees

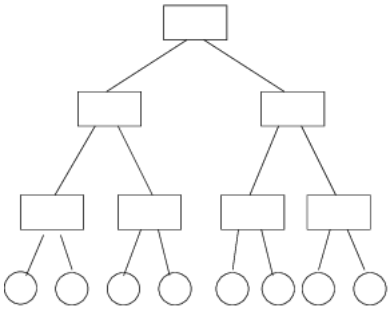




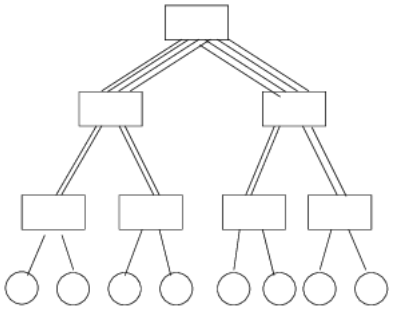
http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Interconnection%20Network.pdf

Topologies: Fat-Tree[胖树]

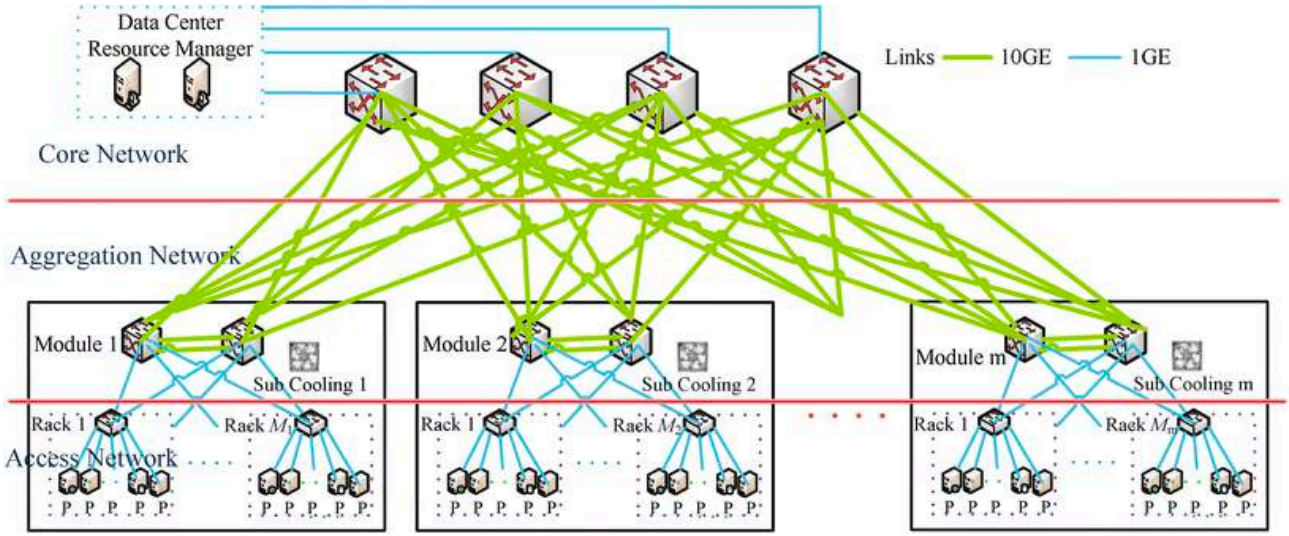
- Links higher up the tree potentially carry more traffic than those at the lower levels
- For this reason, a variant called a **fat-tree**, fattens the links as we go up the tree



(a) Binary tree



(b) Binary fat–tree



http://www.cs.csi.cuny.edu/~gu/teaching/courses/csc76010/slides/Interconnection%20Network.pdf

And Other Topologies …

- Many other topologies with different properties discussed in the literature
  - Clos Networks
  - Omega networks
  - Benes networks
  - Bitonic networks
  - Flattened Butterfly
  - Dragonfly
  - Cube-connected cycles
  - HyperX
  - …
- However, these are typically special purpose and not used in general purpose hardware