

263-3855-00: Cloud Computing Architecture

Section 09: Datacenter Hardware

Swiss Federal Institute of Technology Zurich (ETH Zürich)

Last Edit Date: 05/31/2025

Disclaimer and Term of Use:

1. We do not guarantee the accuracy and completeness of the summary content. Some of the course material may not be included, and some of the content in the summary may not be correct. You should use this file properly and legally. We are not responsible for any results from using this file.
2. This personal note is adapted (derived) from the slides and codes from *Professor Ana Klimovic and people from ETH Zurich Systems Group*. Please [contact us](#) to delete this file if you think your rights have been violated.
3. This work is licensed under a [Creative Commons Attribution 4.0 International License](#).

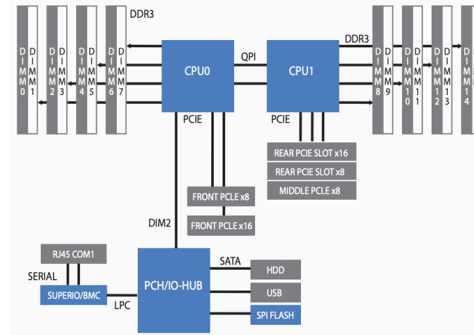
I. CPU server design

1. CPU Design Considerations

- Key differences for CPU server design in a datacenter vs. desktop environment:
 - High request rates (often >1000s per second)
 - Requests often involve processing & moving large volumes of data (big and small data)
 - Need isolation mechanisms for performance and security due to sharing hardware among multiple different workloads
- Goal is to optimize the performance and cost of the datacenter (warehouse-scale computer) as a whole, rather than individual server design
- Specific considerations include:
 - Type of core: "brawny" or "wimpy"
 - Number of cores
 - Number of sockets and NUMA topology
 - Cache hierarchy and size
 - Isolation mechanisms for multi-tenancy (for performance & security)
 - Integration with hardware accelerators, network devices, etc.
 - Power management

2. Brawny vs. Wimpy CPUs

- Low-end (wimpy) cores offer lower performance (e.g., 2x) but at much lower power (e.g., 5x)
- Cloud environments have high request-level parallelism, suggesting the use of many low-end cores
- Reasons for using high-end (brawny) CPU cores:
 - Single-thread performance matters due to Amdahl's Law (serial parts and communication overheads limit speedup)
 - Parallel applications may run less efficiently when computation and data are partitioned into small pieces, especially if threads frequently communicate
 - More threads handling a parallelized request can increase performance variability, where one slow thread limits the whole request's response time
- Amdahl's Law states that speedup is limited by the fraction of the program that cannot be accelerated
- Traditionally, brawny 2-socket servers are common in data centers



- Low-power Arm cores are gaining popularity, such as Amazon Graviton custom-built servers with 64-bit ARM Neoverse cores

3. Amazon Graviton2 ARM-based processor

- Available on Amazon Web Services "m6g" VM instance family
- Architecture based on ARM Neoverse N1 cores:
 - 64 cores connected in a mesh
 - 8 x DDR4-3200 channels with 200 GB/S memory bandwidth
 - No NUMA: every core sees the same path to memory and other cores
 - Large private caches: 64KB L1 caches, 1MB L2 cache per core
 - 32 MB last level cache
 - 64 lanes of PCIe gen4
 - Provides flexibility for different VM configurations (e.g., attaching hardware accelerators, network cards, PCIe-based SSD storage)
 - No simultaneous multithreading ("hyperthreading")
- Comparison to AMD EPYC 7571 (m5a) and Intel Xeon Platinum 8259CL (m5n) servers shows Graviton2 has more cores, a single socket, lower TDP, and a lower price

4. AWS Graviton4

- Announced in Nov. 2023
- Features 96 Neoverse V2 Cores and 2MB L2 cache per core
- Seven chiplet design with 12 DDR5-5600 channels
- Up to 96 lanes of PCIe gen5
- Supports single-socket 24x1 (96 vCPUs) with 50% more cores-per-socket than Graviton3
- Supports coherent multi-socket, enabling single systems with up to 3x more cores and 3x more DRAM than Graviton3

5. Hyperscalers Building Own Hardware

- Examples of custom CPU servers: AWS Graviton, Microsoft Cobalt, Google Axion (all Arm CPUs)
- Examples of specialized accelerators: Microsoft Maia AI accelerator, Google TPU AI accelerator, Google VPU accelerator, AWS Trainium and Inferentia AI accelerators
- Reasons for building own hardware:
 - Enough volume and demand to invest in developing and taping out their own hardware
 - Avoid reliance on a few hardware vendors and their pricing
 - Ability to customize hardware for their specific workloads
- Co-designing hardware & software enables greater optimization by owning the full stack

6. NVIDIA Grace ARM-based processor

- Announced in April 2021 as NVIDIA's first datacenter CPU
- Aims to deliver 10x the performance of today's fastest servers on complex AI and HPC workloads
- Combines energy-efficient ARM CPU cores with a new low-power memory subsystem for high performance and efficiency
- NVIDIA Grace Hopper Superchip combines Grace CPU and Hopper GPU
 - Grace CPU features up to 72 Cores, 512 GB LPDDR5X Memory @ 546 GB/s Bandwidth, and 64x PCIe Gen5 Lanes
 - Hopper GPU features up to 96 GB HBM3 GPU Memory @ 3 TB/s Bandwidth
 - NVLink C2C provides up to 900 GB/s Bandwidth between CPU and GPU, which is 7x higher bandwidth than PCIe Gen5
 - Includes hardware coherency

7. Clusters of Large vs. Small Servers

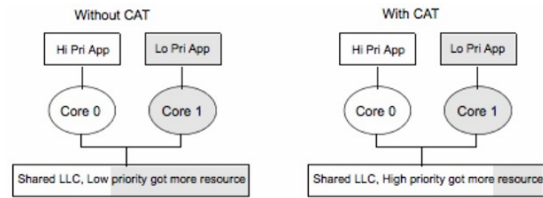
- Primary issue: datacenter applications are often too large even for the largest server
- If going outside a big server is necessary, the performance penalty is similar to having many small servers
- Clusters of smaller servers are often more cost-effective
- The 2-socket server is currently considered a "sweet spot"
- Performance model shows that under light communication, many nodes have relatively small performance overhead
- Under medium to high communication, there are significant performance penalties for using many nodes

8. CPU Hardware Isolation Mechanisms

- Needed because applications on separate cores can contend for resources in shared caches and memory hierarchy, leading to the "noisy neighbor" problem

- Intel Cache Allocation Technology (CAT):

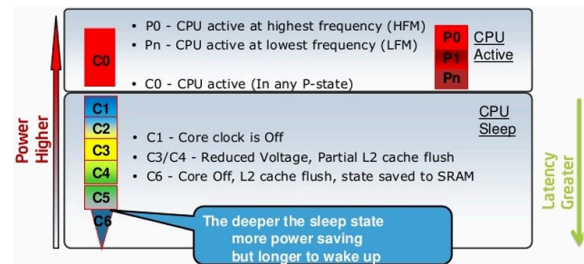
- Partitions ways of a highly-associative last-level cache (LLC) into smaller subsets
- Cores assigned to a subset only allocate cache lines within that subset on refills, but can hit in any part of the LLC



- Currently, no commercially available support for DRAM bandwidth isolation

- Dynamic Voltage Frequency Scaling (DVFS):

- Enables scaling CPU core frequency independently
- P-states manage power consumption while active (P0 highest, Pn lowest frequency)
- C-states manage power consumption while idle (C1 core clock off, C6 core off)
- Deeper sleep states save more power but take longer to wake up



II. Hardware accelerators

1. General Purpose vs. Specialization

- The decline of general-purpose computing due to slow improvements leads to specialization
- CPUs are designed for a range of tasks in a time-shared manner, involving context switching, isolation, memory virtualization
- CPUs are low latency but low throughput
- GPUs are high latency but high throughput
- Comparison of CPU (Intel Xeon E5-2690v4) and GPU (NVIDIA P100):
 - GPU has significantly more parallel calculations (3,584 vs 28), higher memory bandwidth (732 GB/s vs 76.8 GB/s), but higher L1 cache access latency (80 vs 5-12 clock cycles)

2. GPU Architecture

- GPUs manage parallelism hierarchically, with GPU Processing Clusters, Texture Processing Clusters, and Streaming Multiprocessors (SMs)

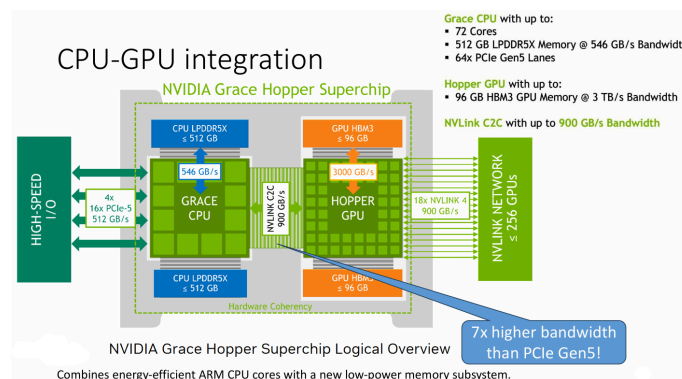
- SMs include:
 - L1 Instruction Cache
 - Warp Scheduler (32 thread/clock)
 - Dispatch Unit (32 thread/clock)
 - Register File (16,384 x 32-bit)
 - INT32 ALUs, FP32 ALUs, FP64 ALUs
 - Tensor cores (for AI workloads)
 - Memory controllers and Texture units
- GPUs became dominant for neural networks due to their ALUs, SIMT parallelism, and data parallelism
- Still specialized and not suitable for all tasks



3. When to Use a GPU

- Data Parallel problems
- Large scale multi-threading
- Little to no synchronization between threads needed
- Smaller data sizes and processing can be batched
- High throughput is desired
- Less useful when:
 - Irregular, non-sequential memory access patterns (e.g., tree traversals)
 - Large amounts of data (moving data in/out of GPU has large overhead)

4. New Interconnects: Compute Express Link (CXL)

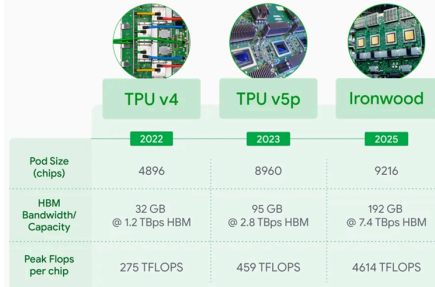


- Open standard cache-coherent interconnect for processors, memory expansion, and accelerators
- Maintains coherency between CPU memory and memory on attached devices
- Enables resource sharing/pooling for higher performance, reduces software stack complexity, and lowers overall system cost

- Needed because:
 - Modern workloads have high memory capacity requirements, and spilling to SSD is too slow
 - Remote memory access needs to be fast
 - Accelerators are increasingly common and need high-bandwidth access between CPU and devices

5. From GPUs to TPUs

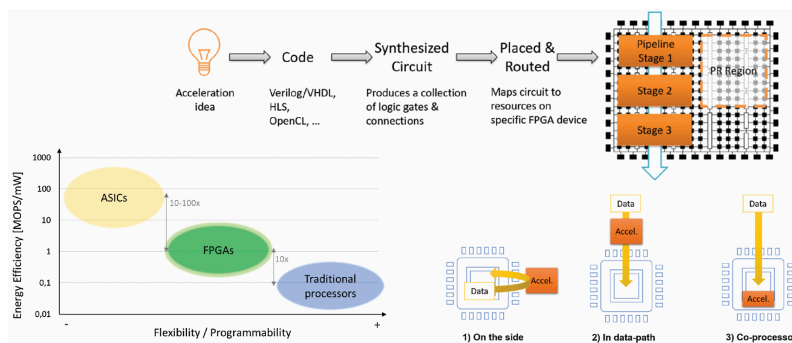
- GPUs are good for ML model training but are power-hungry, have limited memory capacity, and are expensive
- Google makes extensive use of NNs, requiring huge computing capacity for training
- TPUs (Tensor Processing Units) are Google's specialized AI accelerators



	2022	2023	2025
Pod Size (chips)	4896	8960	9216
HBM Bandwidth/Capacity	32 GB @ 1.2 TBps HBM	95 GB @ 2.8 TBps HBM	192 GB @ 7.4 TBps HBM
Peak Flops per chip	275 TFLOPS	459 TFLOPS	4614 TFLOPS

6. FPGAs (Field-Programmable Gate Arrays)

- Useful for tasks where CPUs struggle, but GPUs/TPUs don't help (e.g., networking, sequences of non-floating point/integer operations, data parallelism with irregular memory accesses)
- Can be used as a "bump in the wire" to offload tasks like data encryption and network virtualization
- Offer a trade-off between energy efficiency (higher than traditional processors, lower than ASICs) and flexibility/programmability



7. Cloud Caches (Amazon Aqua)

- Advanced Query Accelerator for Amazon Redshift
- Designed to deliver up to 10x performance on queries with large scans, aggregates, and filtering
- Features AWS-designed analytics processors, SSDs, and parallel execution

8. Hardware Specialization Trends

- Beyond accelerators, specialization is appearing at other levels:
 - Smart storage (e.g., AQUA)
 - Smart memory (processing in memory)
 - Smart NICs (processing data near the network)
 - Software Defined Networking (programmable switches)
- This leads to a growing number of new processors and computer configurations
- **Example:** Video Coding Unit (VCU)
 - Google's specialized accelerator for warehouse-scale video transcoding
 - Achieves 20-33x better compute-efficiency vs. optimized CPU baseline for transcoding video to different formats/resolutions
 -
 - Addresses high demand for LLM inference clusters which are challenging for thermal and power management

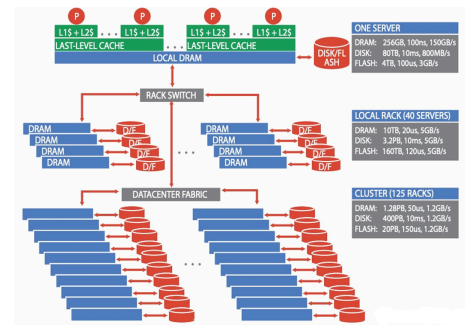
III. Datacenter Storage Hierarchy

Hierarchical storage from fast, small caches to slow, large disks

Key metrics: size, latency, bandwidth

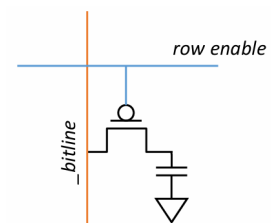
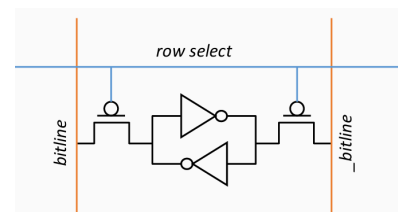
Example capacities and latencies:

- One Server: DRAM (256GB, 100ns, 150GB/s), Flash (4TB, 100us, 3GB/s), Disk (80TB, 10ms, 800MB/s)
- Rack: DRAM (10TB, 20us, 5GB/s), Flash (160TB, 120us, 5GB/s), Disk (3.2PB, 10ms, 5GB/s)
- Cluster: DRAM (1.28PB, 50us, 1.2GB/s), Flash (20PB, 150us, 1.2GB/s), Disk (400PB, 10ms, 1.2GB/s)

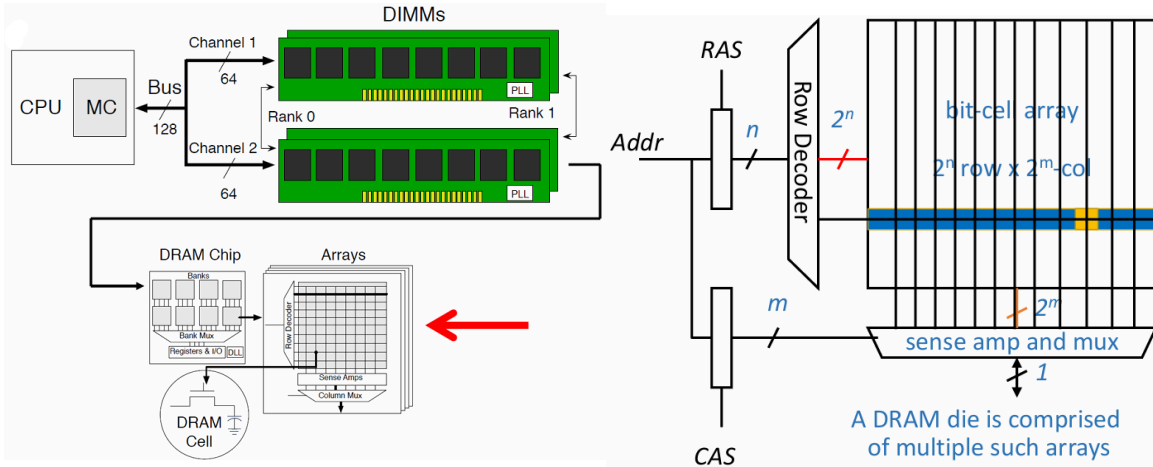


1. SRAM vs. DRAM

- SRAM (Static Random Access Memory):
 - Uses transistors to drive bitlines
 - Fast access, no refreshes, simpler manufacturing
 - Lower density (6 transistors per cell), higher cost
 - Preferable for register files and L1-L3 caches
- DRAM (Dynamic Random Access Memory):
 - Bits stored as charges on node capacitance (non-restorative)
 - Destructive read (loses charge when read)
 - Must periodically refresh (every 10s of ms)



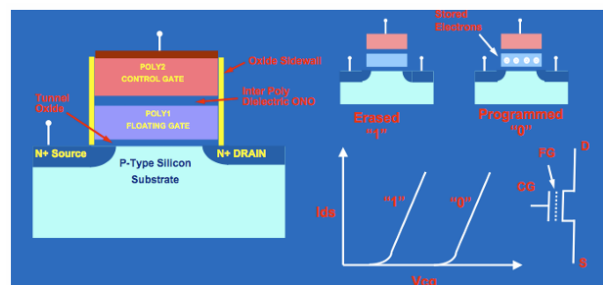
- Much higher capacity and density, lower cost
- Preferable for stand-alone memory chips
- Embedded DRAM sometimes used for L3/L4 caches
- DRAM memory organization uses multiple channels, ranks, chips, and banks for parallelism



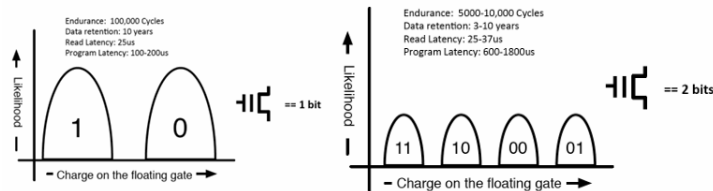
- DRAM access sequence: decode row address, select bits drive bit-lines, entire row read, amplify row data, decode column address, send to output, pre-charge bit-lines
- DRAM controller periodically refreshes rows
- Memory-Access Protocol uses commands like ACTIVATE, READ, WRITE, PRECHARGE, REFRESH
- Row Buffer Management Policies:
 - Open row policy: keeps row open after access (pro: next access might need same row, con: next access might need different row causing conflict)
 - Closed row policy: closes row immediately after access (pro: avoids conflict if next access needs different row, con: extra latency if same row is needed, wasted energy)
 - Adaptive policies attempt to predict next access to the same row

2. Flash Storage Technology

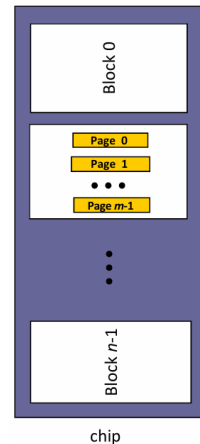
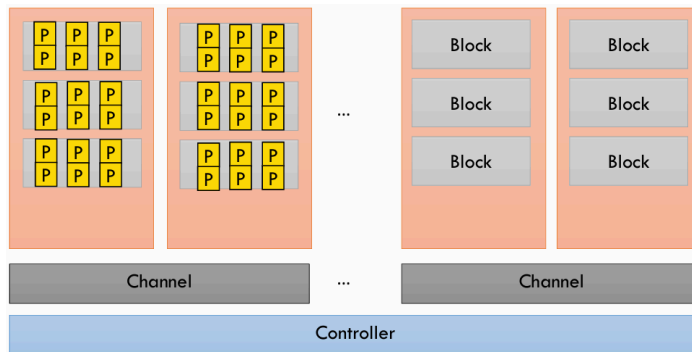
- Semiconductor, non-volatile memory
- Compared to mechanical hard disk: lower latency, higher bandwidth, lower power, lighter weight, smaller size, shock resistance
- Rough cost per bit comparison (DRAM: Flash: Disk): 100:10:1



- Rough access latency comparison (DRAM: Flash: Disk): 1:5,000:1,000,000
- Flash Cell: Stores bits as charge trapped in a floating gate, modulating the threshold voltage of the underlying transistor
- Multi-level Cells (MLC) store more bits per cell (e.g., 2 bits) but have lower endurance and higher program latency than Single-Level Cells (SLC)



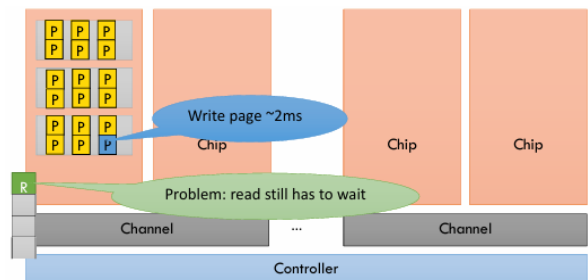
- SLC is faster, less dense, more reliable (100K-1M erase cycles), ~\$1.5/GB, used in enterprise drives
- MLC is slower, denser, less reliable (1K-10K erase cycles), ~\$0.08/GB, used in consumer drives
- Internal Architecture: multiple levels of parallelism (controllers, channels, chips, blocks, pages)
- Page: minimum unit of read/write (512 bytes - 8KB + spare area)
- Block: minimum unit of erasing (64-256 pages per block)



- Flash Operations:
 - Read page: 20-80us
 - Write (program) data to a page: 1-2ms (only 1→0 transitions, must be ordered within a block)
 - Erase block: 3-7ms (pages must be erased before writing, no update-in-place)

- Read/Write Interference: each chip processes one operation at a time, reads wait for slow write/erase

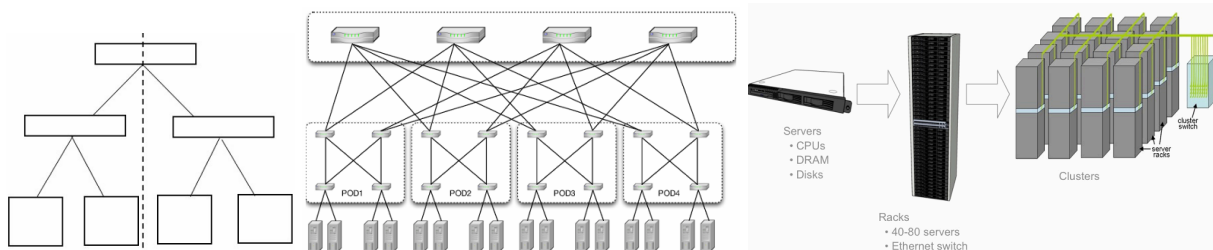
- Example: With 99% read workload, throughput is halved compared to 100% read workload



- Reliability:
 - Wear out: physical damage from programming/erasing
 - Writing disturb: programming pages corrupts other pages in block
 - Read disturb: reading data corrupts data in block (after many reads)
 - Flash Translation Layer (FTL):
 - Makes raw flash look like a disk by exposing a block-based interface
 - Manages program/erase granularity mismatch, equalizes wear, handles power cycle recovery
 - Maintains logical-to-physical block address map and block info table
- User
 - Logical Block Address (LBA)
 - FTL
 - Logical → physical map
 - Wear leveling
 - Power cycle recovery
 - Flash
 - Write pages in order
 - Erase/Write granularity
 - Wears out
- Write Amplification: number of physical pages written to flash for every logical page written by user
 - Can be reduced by reserving extra capacity, writing sequentially, or storing frequently written data in memory
 - Wear Leveling: maximizes device lifetime by uniformly distributing block wear
 - Keeps statistics on write/erase cycles
 - Erases blocks with low erase counts and moves data when current block wears out

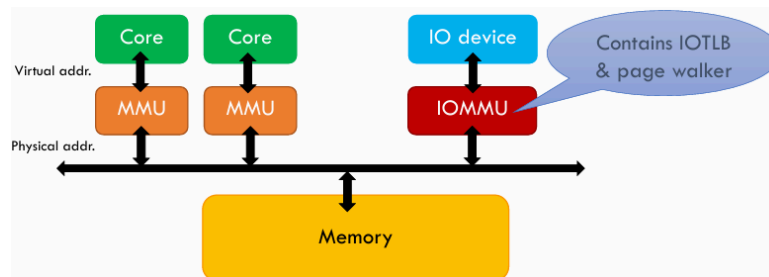
IV. Data center networking

- Servers need to be connected to enable communication between all servers
- Challenge: scaling bisection bandwidth (available bandwidth between any two equal network halves) as the number of servers scales
- Large switches are limited by pins and power, so switch chips are cascaded in hierarchical topologies
- Links higher in topology are over-subscribed (cost-performance trade-off)
- Servers use high bandwidth networks (10, 100, 400 Gb/s Ethernet) with NICs
- Connection path: Server NIC -> Top of Rack Switch -> Cluster Switch



1. I/O Virtualization

- Goals: isolation (safety), scalability, performance
- Two aspects:
 - Virtual device interface (e.g., paravirtualization, Single Root I/O Virtualization (SR-IOV))
 - I/O Memory Management Unit (IOMMU)
- SR-IOV: Hardware support for sharing a PCIe device between multiple guests, appearing as separate physical devices
 - Hypervisor not involved in data transfers; independent memory space, interrupts, and DMA streams for guests
 - Requires hardware support from the I/O device
- IOMMU: Hardware that intercepts DMA transactions and interrupts
 - Key capabilities: virtual address translation for DMA, interrupt remapping and virtualization, memory protection for DMA (security)



- DMA (Direct Memory Access): Separate engine for data transfer between I/O device and memory without CPU involvement
 - CPU describes transfer to DMA engine, which performs read/writes and sends interrupt on completion
 - Advantage: decouples data transfer from processing, CPU does not copy data, higher performance

2. How Software Knows When Data is Available

- Interrupts:
 - Context switch and process interrupt on request arrival, return to program after handling
 - Advantage: I/O work only when requests available
 - Disadvantage: Higher latency due to interrupt handling overhead
- Polling:
 - Periodically check register status for available requests

- Advantage: Lower latency for request processing
- Disadvantage: Processor always busy, even without requests

3. Solutions for Slow OS Networking

- Bypass the kernel: Move TCP processing to user-space and optimize software (e.g., DPDK library)
- Offload TCP processing to NIC hardware
- NIC Acceleration Techniques:
 - Checksum offload, large segment offloading, automatic header generation
 - Large receive offload (combining incoming packets)
 - Header splitting (placing header/payload in separate buffers to improve locality and enable prefetching/zero copy)
- Interrupt Coalescing (interrupt moderation): NIC processes many packets before raising an interrupt
 - Trade-off: power savings vs. latency
 - Software approximation: poll for a while after an interrupt
- Interrupt Steering:
 - Send all interrupts to single core (+locality, -imbalance, -scalability)
 - Spread interrupts across all cores (+balance, +scalability, -locality, -interference)
 - Send to core running app waiting for data (e.g., Intel's flow director) (+data locality, +little interference, +/-balance, +/-scalability)
- Replace classic Ethernet: Use a lossless fabric (e.g., Infiniband) and offload Remote Direct Memory Access (RDMA)
 - RDMA allows accessing remote memory on a node without involving its CPU

V. Related readings

1. [Profiling a warehouse-scale computer](#)
2. (Optional) [Memory Hierarchy for Web Search](#)
3. (Optional) [TAPAS: Thermal- and Power-Aware Scheduling for LLM Interface in Cloud Platforms](#)
4. (Optional) [Warehouse-Scale Video Acceleration: Co-design and Deployment in the Wild](#)
5. (Optional) [A Hardware Accelerator for Protocol Buffers](#)
6. (Optional) [AsmDB: Understanding and Mitigating Front-End Stalls in Warehouse-Scale Computers](#)