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14讲：Memory (1)

张献伟

xianweiz.github.io

DCS5637, 11/30/2022

Review Questions

- explain SM or CU.

  The fundamental compute unit to execute GPU tasks, hosting multiple simple cores to run the threads

- relationship of kernel and grid?

  Threads to execute the kernel are organized as a grid.

- steps of kernel launch?

  User-mode queue, command processor, blocks to SM, warps

- stream in GPU?

  software abstraction of queue, path to transmit tasks from CPU

- MPS?

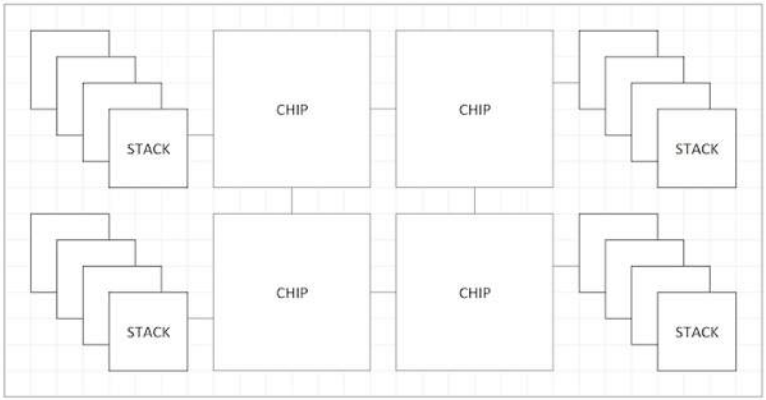
  multi-process service to run multi processes on GPU

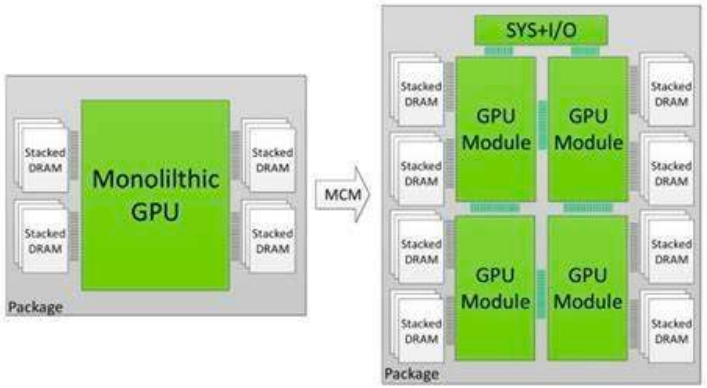
- shared memory in GPU?

  software-controlled L1 cache in SM, fast data share within block

Multi-chip Module

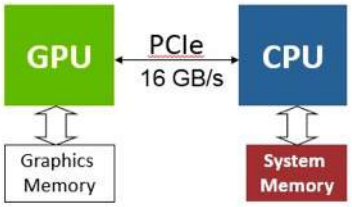
- Aggregating multiple GPU modules within a single package, as opposed to a single monolithic die.
- AMD: Chiplet GPUs
  - MI200: 220 compute units, 14K streaming cores
  - MI100: 120 compute units, 7680 streaming cores
- Nvidia: Multi-Chip-Module (MCM) GPUs
  - Hopper (Ampere -> Lovelace): 300+ SMs, 40K+ CUDA cores
  - A100: 128 SMs, 8192 CUDA cores



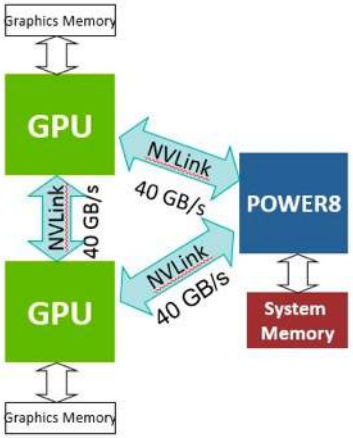


High-speed Links[高速连接]

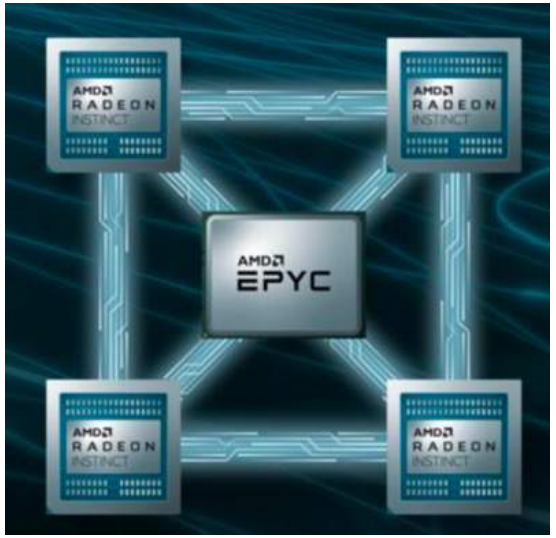
- GPUs are of high compute capability, being bottlenecked on data movement
- High-speed interconnect to achieve significantly higher data movement
  - Nvidia: NVLink
  - AMD: Infinity Fabric
  - Intel: Compute eXpress Link (CXL)



CPU-GPU Systems Connected via PCI-e

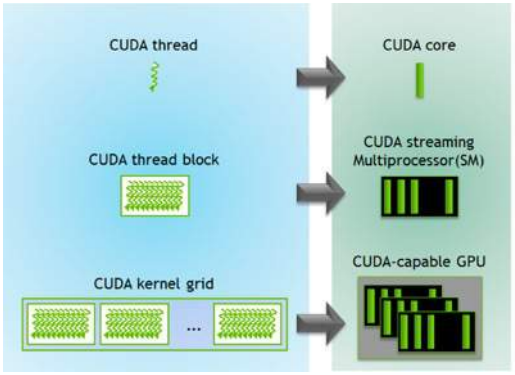


NVLink Enables Fast Unified Memory Access between CPU & GPU Memories



Summary of DLP/GPU[总结]

- Data level parallelism
  - SIMD: operates on multiple data with on single instruction
    - AVX-512 on Intel CPU is the typical example
  - SIMT: consists of multiple scalar threads executing in a SIMD manner
    - GPU is the example with threads executing the same instruction
- GPU hardware and thread organization
  - Device → SM → SIMD/Partition → Core
  - Grid → Block → Warp → Thread

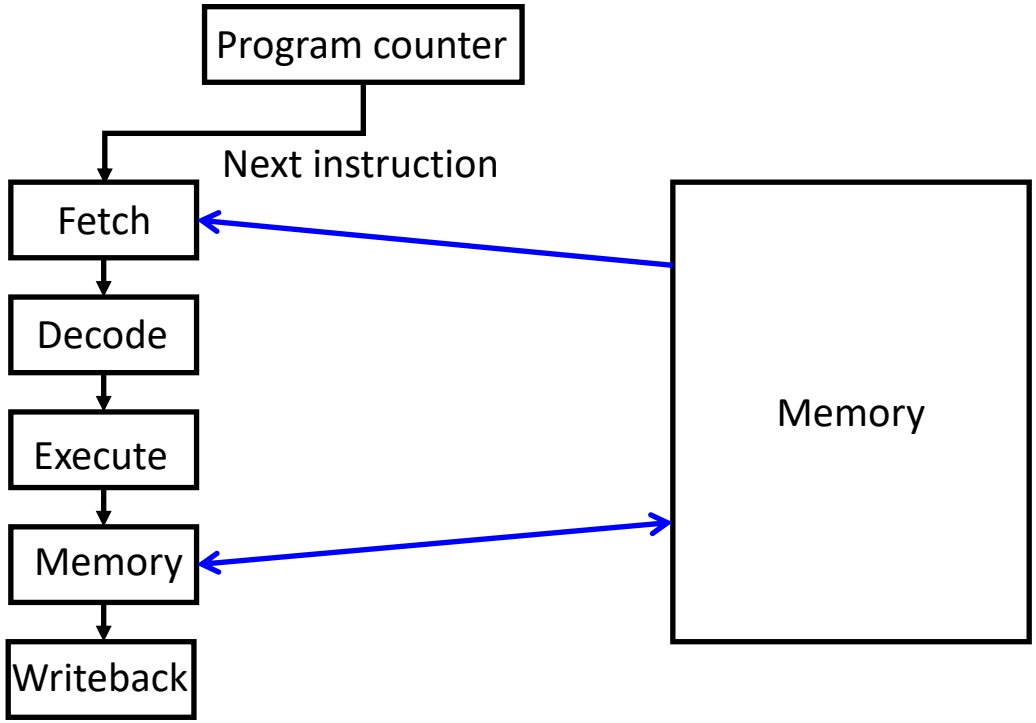


- GPU programming
  - Streams to support concurrency
  - Memory hierarchy and usage (thread, cache/smem, global)
  - Advanced topics: virtualization, profiling/tuning, divergence, etc

Part-I: Memory Hierarchy

Memory Access[存储访问]

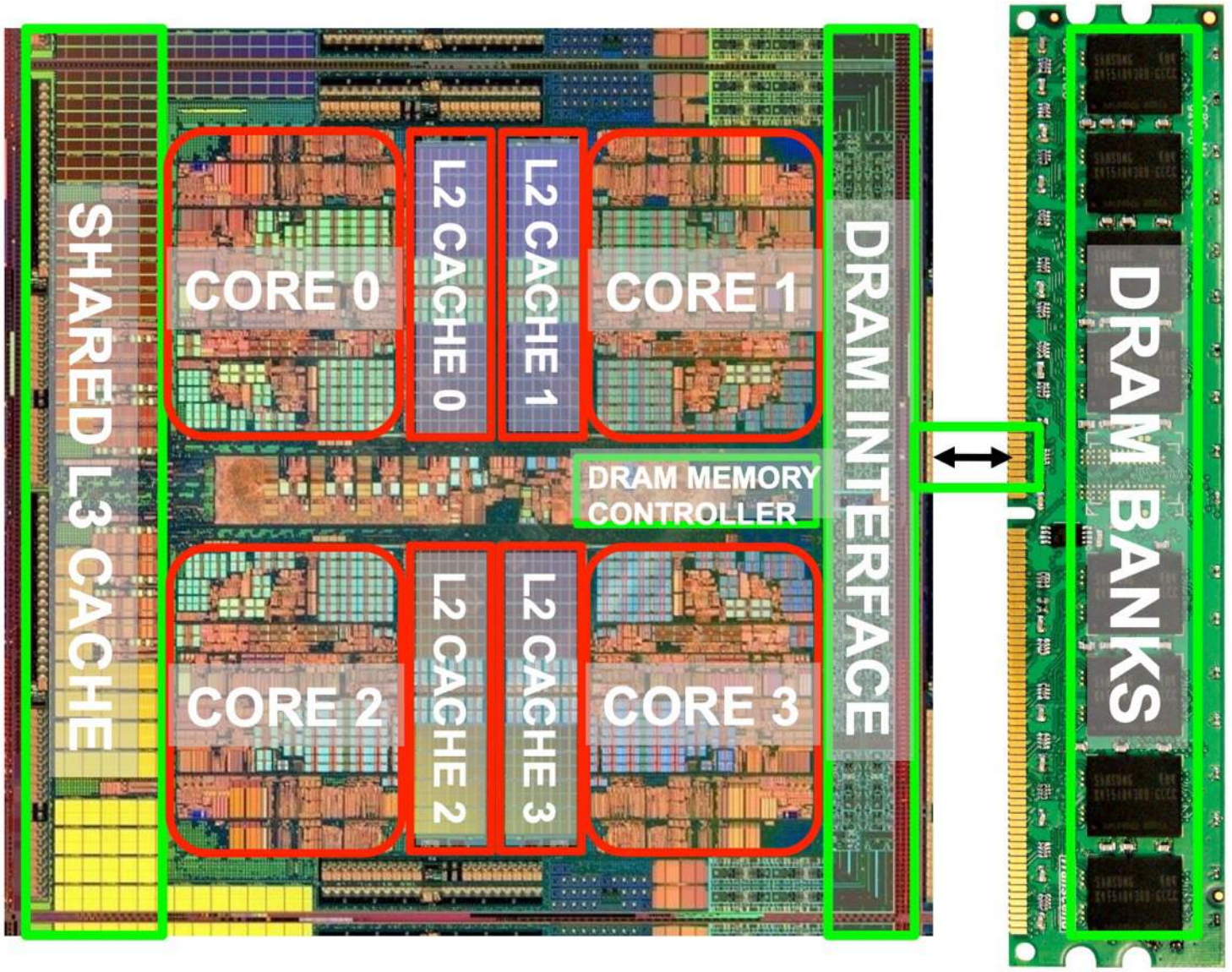
- Programmer's view: read/write (i.e., load/store)
  - Instruction[指令]
  - Data[数据]



Memory[存储]

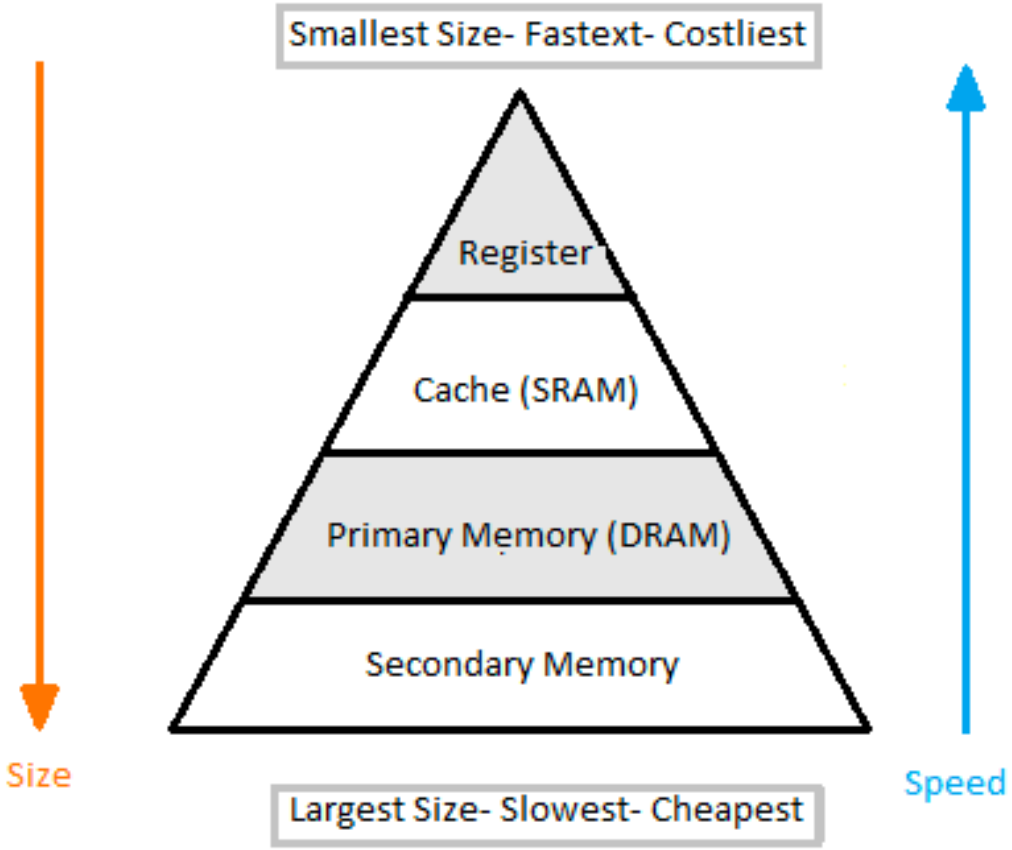
- Ideal memory[理想情况]
  - Zero access time (latency)[零时延]
  - Infinite capacity[无限容量]
  - Zero cost[零成本]
  - Infinite bandwidth (to support parallel accesses)[无限带宽]
- Problem: the requirements are conflicting[问题：需求互斥]
  - Bigger is slower[大容量→长时延]
    - Longer time to determine the location
  - Faster is more expensive[快访问→高成本]
    - More advanced technology
  - Higher bandwidth is more expensive[高带宽→高成本]
    - More access ports, higher frequency, …

Memory in Modern System

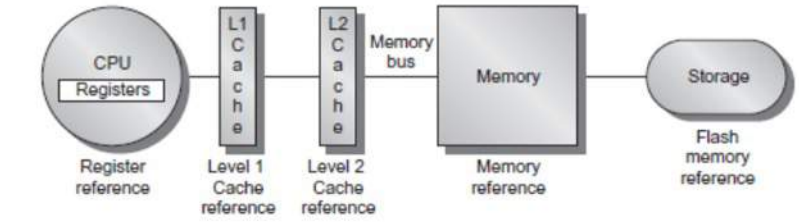


Memory Hierarchy[存储层级]

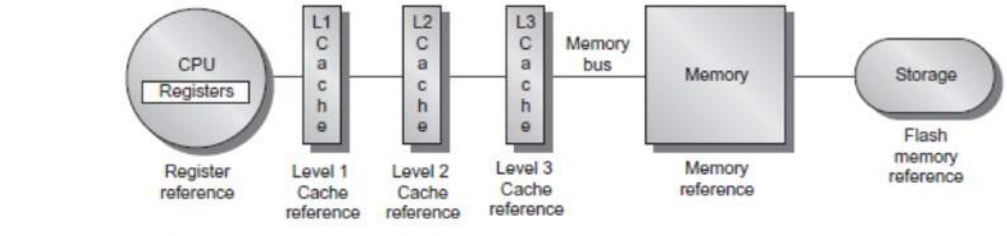
- Goal: provide a memory system with a cost per bit that is almost as low as the cheapest level of memory and a speed almost as fast as the fastest level



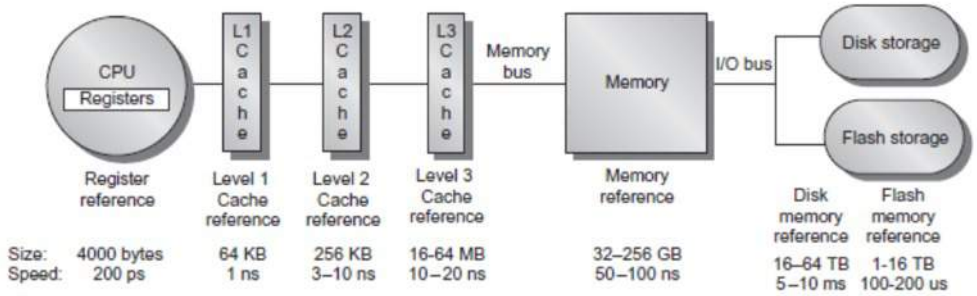
Memory Hierarchy (cont.)



(A) Memory hierarchy for a personal mobile device



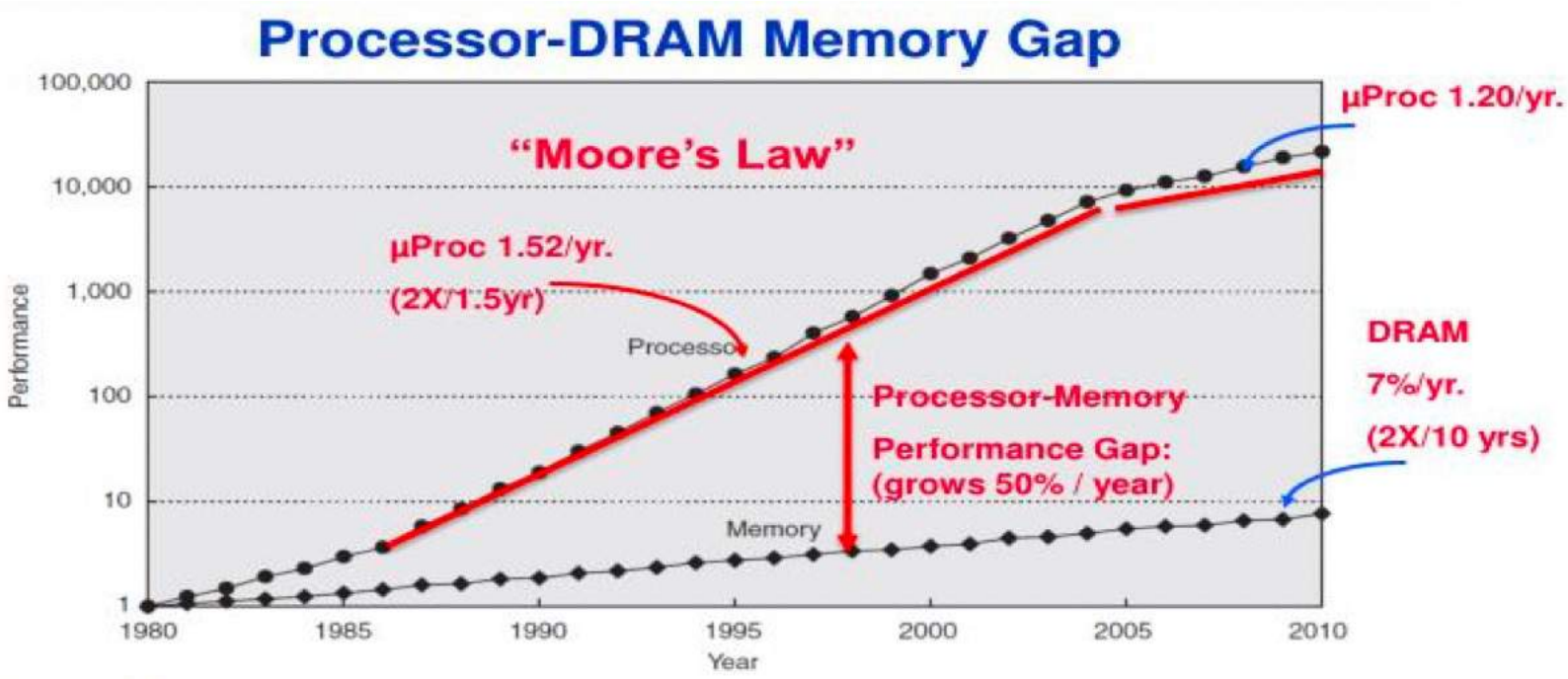
(B) Memory hierarchy for a laptop or a desktop



(C) Memory hierarchy for server

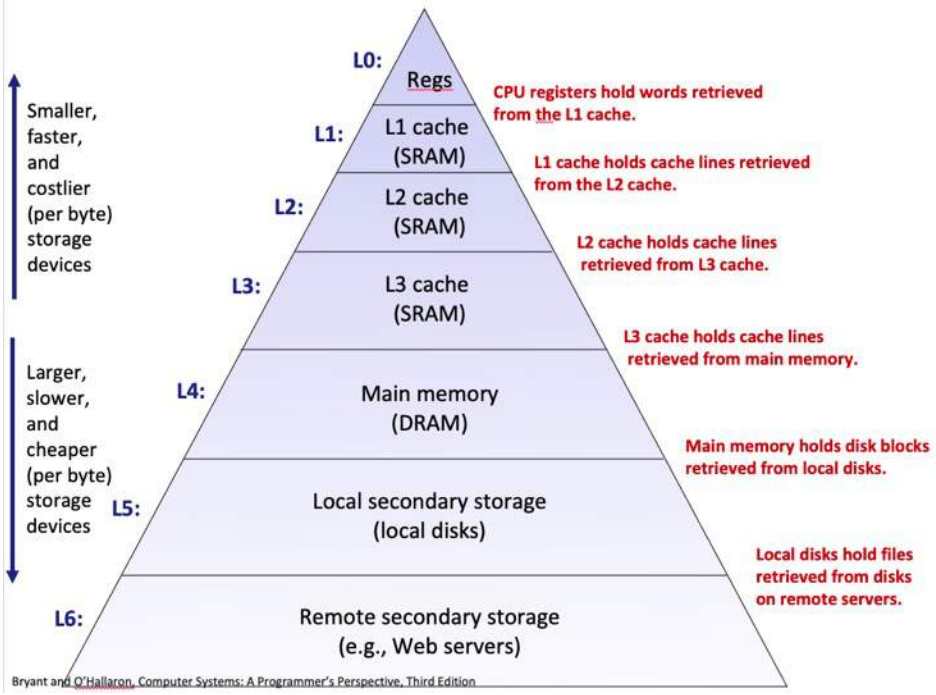
Memory Wall[存储墙]

- On modern machines, most programs that access a lot of data are memory bound
  - Latency of DRAM access is roughly 100-1000 cycles
  - Involves both the limited capacity and the bandwidth of memory transfer



Deeper Hierarchy[更深层级]

- 1980: no cache in micro-processor
- 1989: Intel 486 processor with 8KB on-chip L1 cache
- 1995: Intel Pentium Proc with 256KB on-chip L2 cache
- 2003: Intel Itanium 2 with 6MB on-chip L3 cache
- 2010: 3-level cache on chip, 4th-level cache off chip



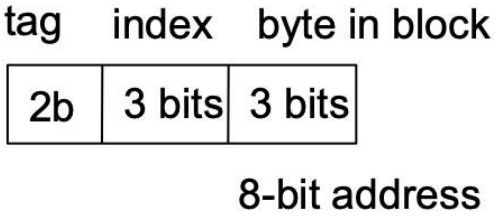
Part-II: Cache

Cache Basics[缓存基础]

- **Block** (line): unit of storage in the cache[缓存单位]
  - Memory is logically divided into cache blocks that map to locations in the cache
- When data referenced[使用]
  - HIT: if in cache, use cached data instead of accessing memory
  - MISS: if not in cache, bring block into cache
    - Maybe have to kick something else out to do it
- Some important cache design decisions
  - Placement[放置]: where and how to place/find a block in cache?
  - Replacement[替换]: what data to remove to make room in cache?
  - Granularity of management[粒度]: large, small, uniform blocks?
  - Write policy[写策略]: what do we do about writes?
  - Instructions/data[指令/数据]: do we treat them separately?

Cache Basics (cont.)

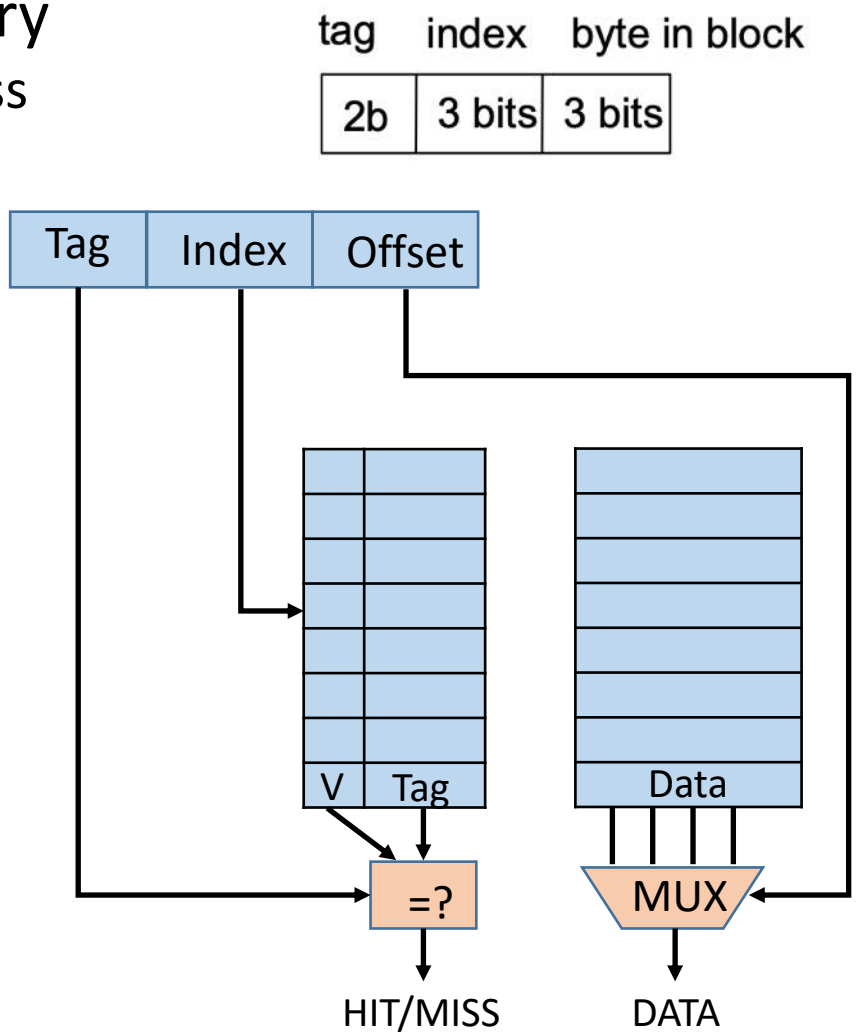
- Memory is logically divided into fixed-size blocks
- Each block maps to a location in the cache, determined by the index bits in the address
  - Used to index into the **tag and data stores**



- Cache access steps
  - 1) index into the tag and data stores with index bits in address
  - 2) check valid bit in tag store
  - 3) compare tag bits in address with the stored tag in tag store
- If a block is in the cache (cache hit), the stored tag should be valid and match the tag of the block

Cache Basics (cont.)

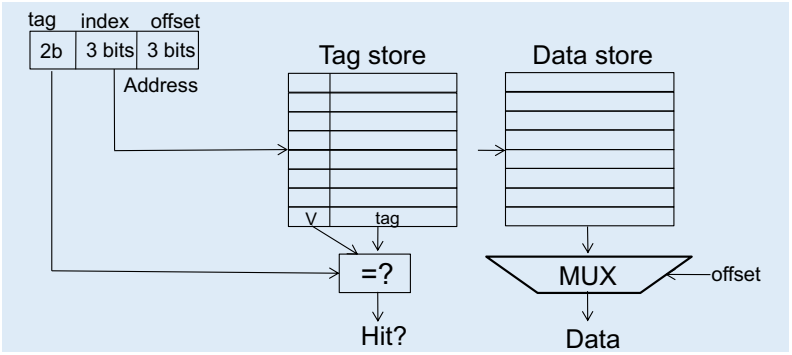
- Assume byte-addressable memory
  - Capacity: 256 bytes → 8-bit address
  - Block: 8 bytes → 3-bit offset
  - #blocks: 32 (256/8)
- Assume cache
  - Capacity: 64 bytes → 3-bit index
    - Holding 8 blocks (64/8)
- What is a tag store?
  - Tag
  - Metadata
    - Valid bit
    - Replacement policy bits
    - Dirty bit
    - ECC

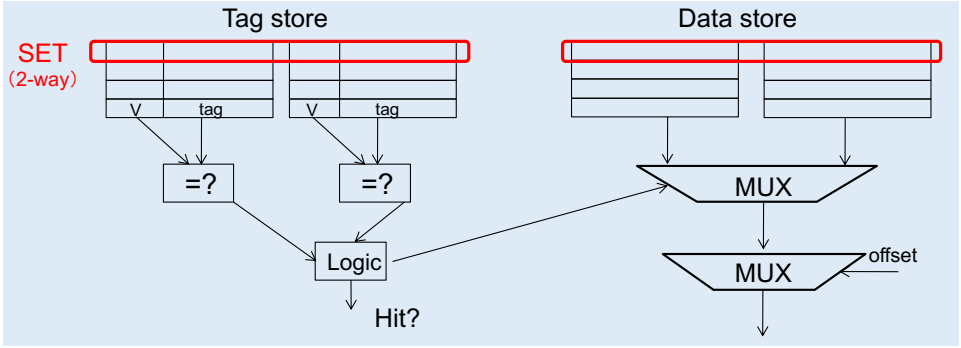


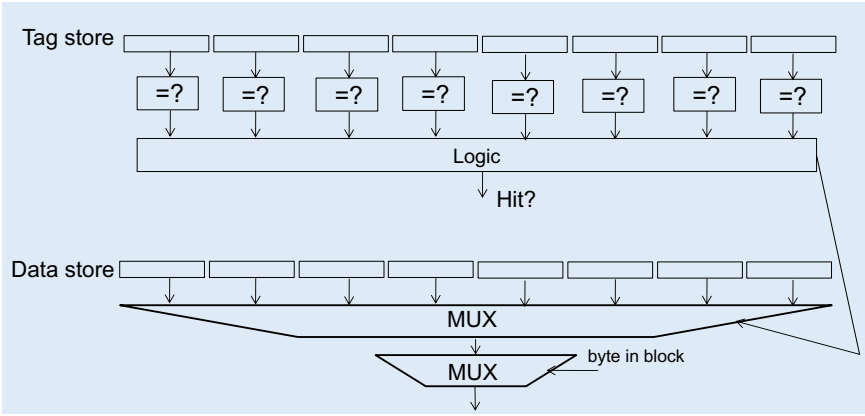
MUX: multiplexer (数据选择器)

Cache Organization

- Direct mapped
  - For each item (block) of data in memory, there is **exactly one** location in the cache where it might be
- Set associative
  - associative memory within the set
- Fully associative







Evaluation Metrics[评价指标]

- Cache hit ratio
  - (# hits) / (# hits + # misses) = (# hits) / (# accesses)
- Average memory access time (AMAT)
  - (hit-ratio * hit-latency) + (miss-ratio * miss-latency)
- Cache hit rate: # of misses per kilo instructions (MPKI)

Example: Assume that
Processor speed = 1 GHz (1 n.sec. clock cycle)
Cache access time = 1 clock cycle
Miss penalty = 100 n.sec (100 clock cycles)
I-cache miss ratio = 1%, and D-cache miss ratio = 3%
74% of memory references are for instructions and 26% for data

Effective cache miss ratio = 0.01 * 0.74 + 0.03 * 0.26 = 0.0152
Av. (effective) memory access time = 1 + 0.0152 * 100 = 2.52 cycles = 2.52 n.sec

Cache Misses: 3 Cs

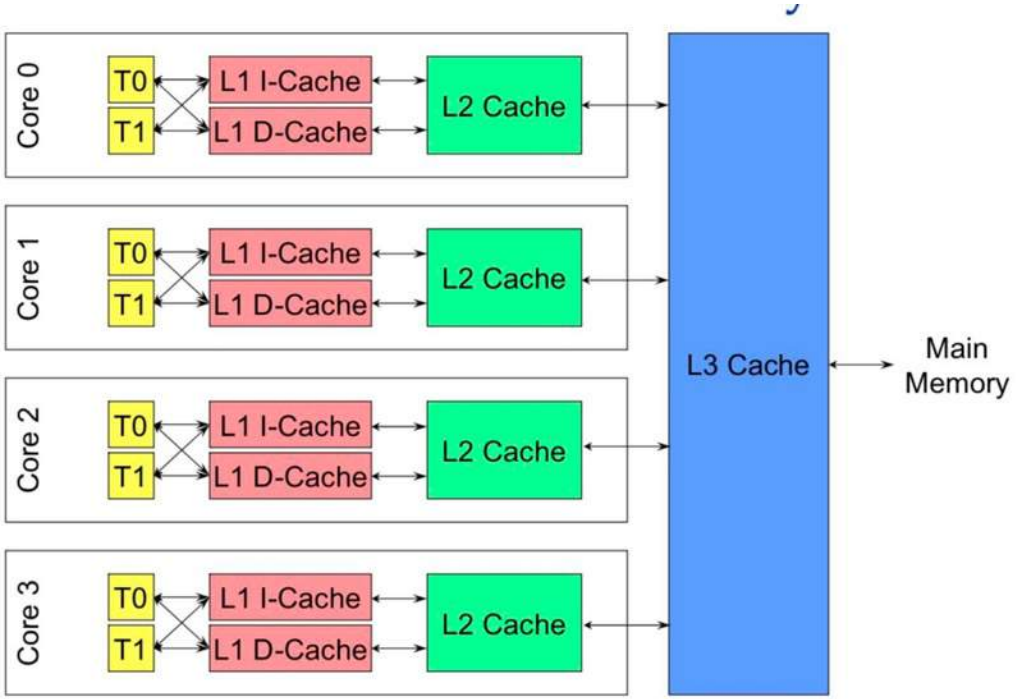
- **Compulsory**/Cold[强制性未命中]
  - First access to a block which is not in the cache
    - The block must be brought into the cache
  - Cache size does not matter
  - Solution: prefetching
- **Capacity**[容量性未命中]
  - Cache cannot contain all bocks needed during program execution
    - Blocks are evicted and later retrieved
  - Solution: increase cache size, stream buffers
- **Conflict**[冲突性未命中]
  - Occurs in set associative or direct mapped caches when too many blocks are mapped to the same set
  - Solution: increase associativity, victim cache
    - No conflict misses in fully associative cache

Coherence (invalidation) misses: other process updates memory

To Optimize Cache[优化缓存]

- Average memory access time (AMAT) = (hit-rate * hit-latency) + (miss-rate * miss-latency)

- Basic requirements
  - Hit latency
  - Miss rate
  - Miss penalty
- Two more requirements
  - Cache bandwidth
  - Power consumption

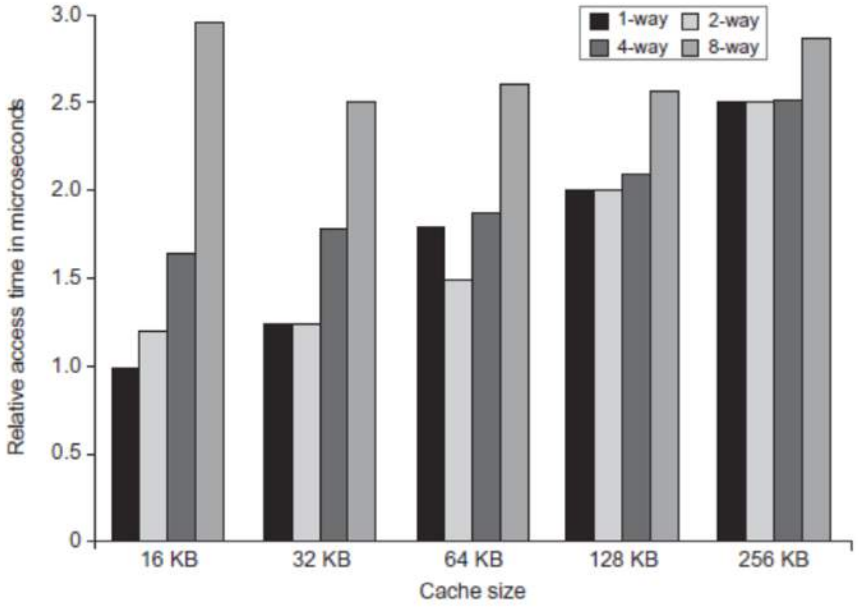


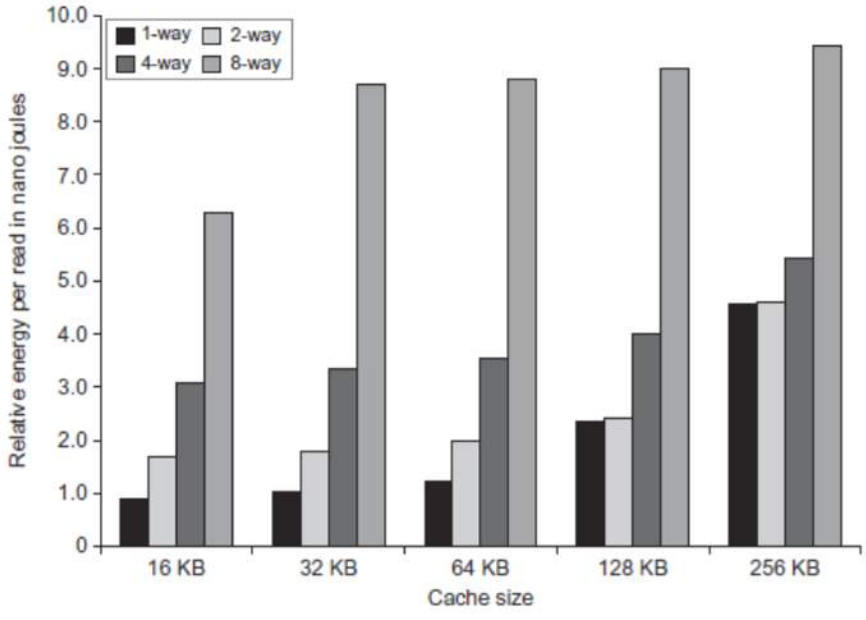
Advanced Cache Optimizations[优化]

- Reducing the hit time[缩短命中时延]
  - Small and simple first-level caches
  - Way prediction
- Increasing cache bandwidth[提高缓存带宽]
  - Pipelined caches
  - Multibanked caches
  - Non-blocking caches
- Reducing miss penalty[降低不命中开销]
  - Critical word first
  - Merging write buffers
- Reducing miss rate[降低不命中率]
  - Compiler optimizations

#1: Small & Simple 1st-level Cache[小]

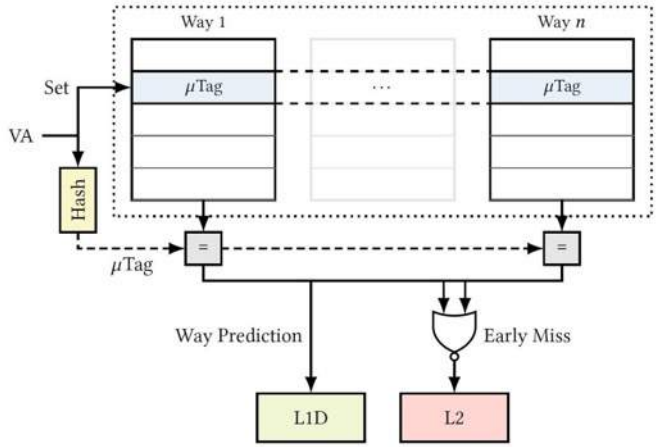
- To reduce hit time and power
- The L1 cache size has recently increased either slightly or not at all
  - Limited size: pressure of both a fast clock cycle and power limitations encourages small sizes
  - Lower level of associativity: reduce both hit time and power





#2: Way Prediction[预测]

- To reduce hit time
  - Add extra bits in the cache to predict the way of the next cache access
    - Block predictor bits
  - Multiplexor is set early to select the desired block
    - And in that clock cycle, only a single tag comparison is performed in parallel with reading the cache data
  - A miss results in checking the other blocks for matches in the next clock cycle
- Miss-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s, now used on ARM Cortex-A8

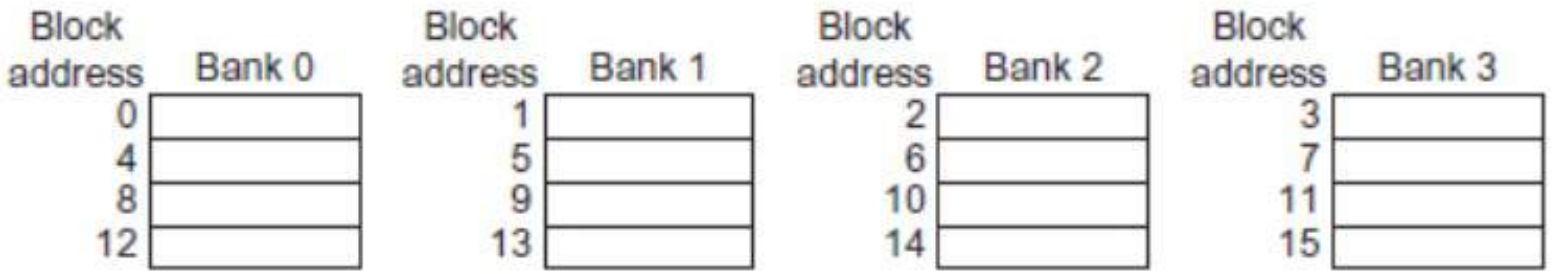


#3: Pipelined[流水线]

- To increase bandwidth
  - Primarily target at L1, where access bandwidth constrains instruction throughput
  - Multibanks are also used in L2/L3, but mainly for power
- Pipelining L1
  - Stages
    - address calculation
    - disambiguation (decoder)
    - cache access (parallel tag and data)
    - result drive (aligner)
  - Allows a higher clock cycle, at the cost of increased latency
  - Examples
    - Pentium: 1 cycle, Pentium Pro – III: 2, Pentium 4 – Core i7: 4 cycles

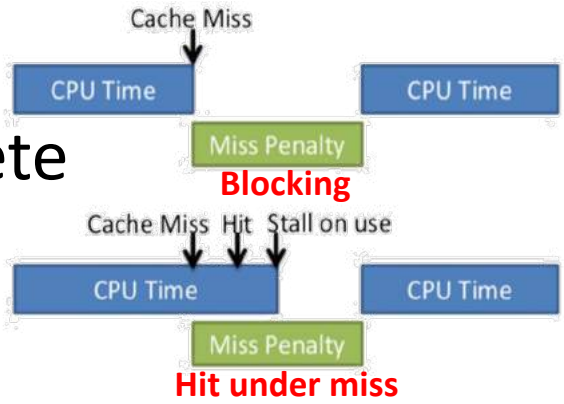
#3: Multibanked[多单元]

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Interleave banks according to block address
  - Banking works best when the accesses naturally spread across banks
- Multiple banks also are a way to reduce power consumption in both caches and DRAM

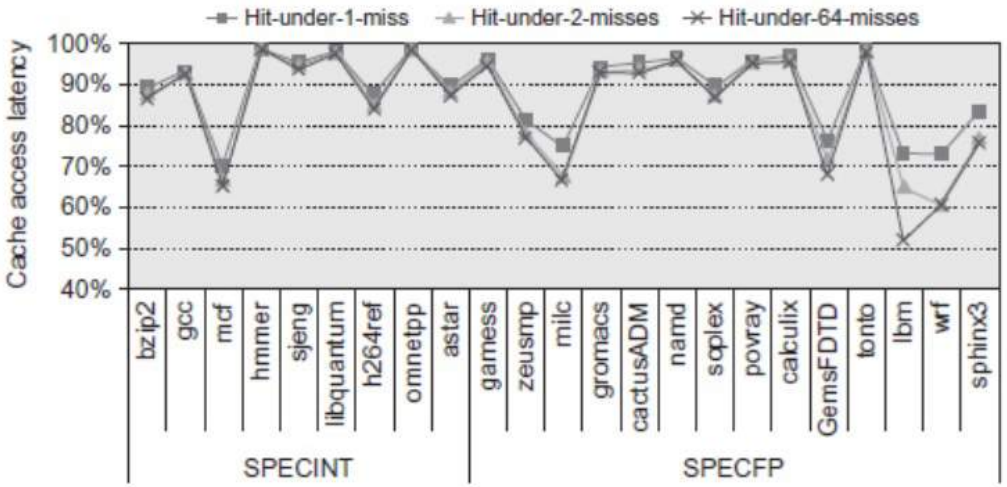


#4: Nonblocking Caches[非阻塞]

- To increase cache bandwidth
- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"



- Nontrivial to implement the nonblocking
  - Arbitrating contention between hits and misses; tracking outstanding misses
  - Miss Status Handling Registers (MSHRs)

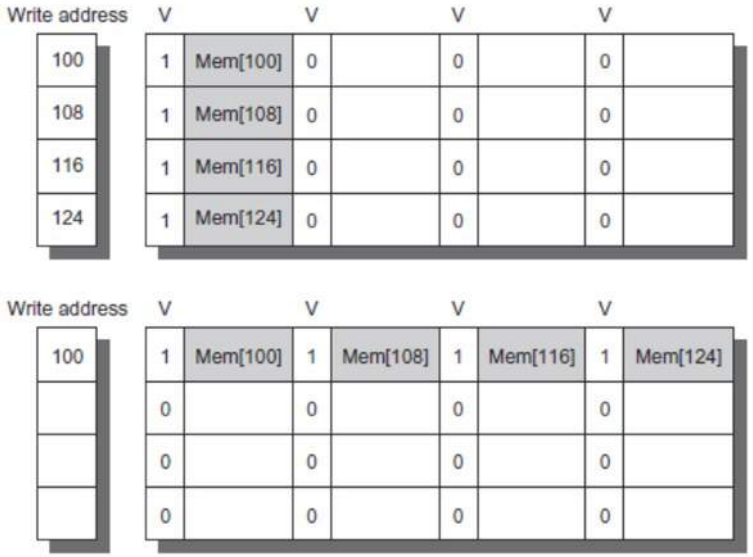


#5: Critical Word First & Early Restart

- To reduce miss penalty
- Processor normally needs just one word of the block at a time
  - Don't wait for the full block to be loaded before sending the requested word and restarting the processor
- Critical word first[关键字优先]
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart[提早重启]
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives
- Effectiveness depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

#6: Merging Write Buffers[写缓冲合并]

- To reduce miss penalty
- When storing to a block that is already pending in the write buffer, update write buffer
- Advantages
  - Multiword writes are usually faster than writes one word a time
  - Reduces stalls due to full write buffer
- Do not apply to I/O addresses[I/O设备]



No write buffering

Write buffering

#7: Compiler Optimizations[编译]

- To reduce miss rate, without any hardware changes
- Loop interchange
  - Swap nested loops to access memory in sequential order
  - Improving spatial locality
    - Maximizes use of data in a cache block before they are discarded

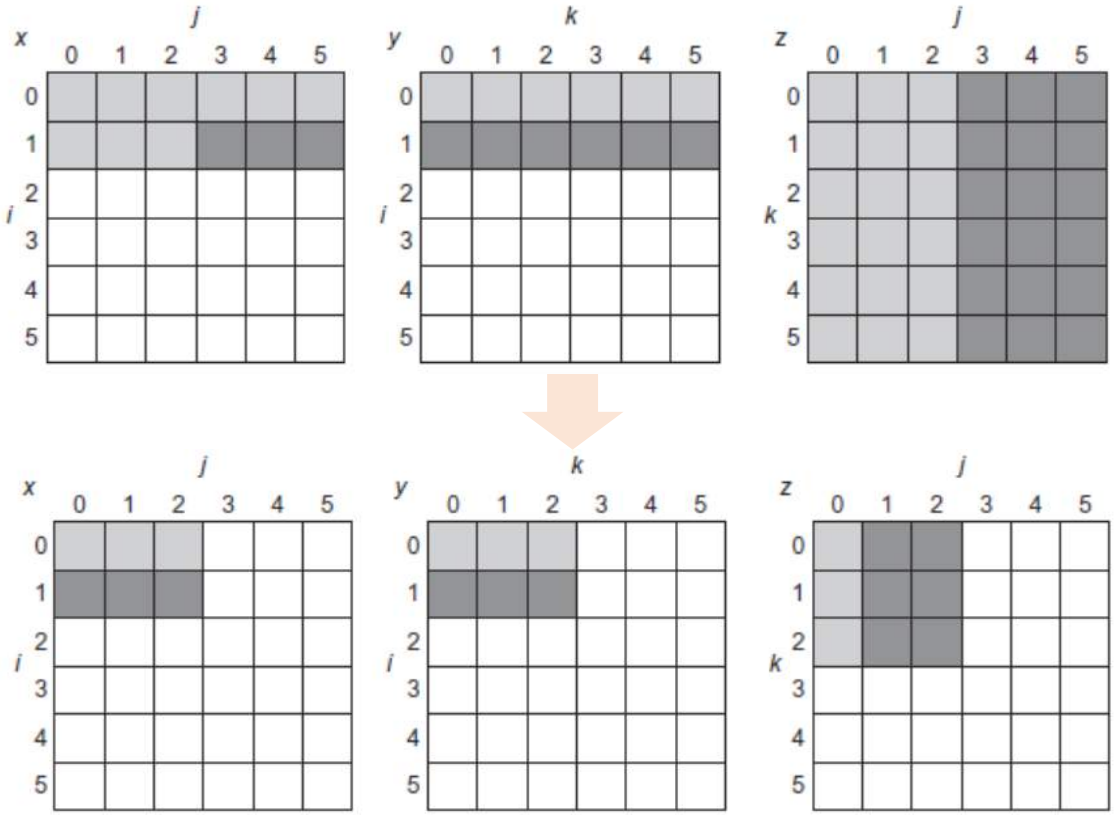
```
/* Before */
for ( j = 0; j < 100; j = j + 1 )
      for ( i = 0; i < 5000; i = i + 1 )
            x[i][j] = 2 * x[i][j];
```



```
/* After */
for ( i = 0; j < 5000; i = i + 1 )
      for ( j = 0; j < 100; j = j + 1 )
            x[i][j] = 2 * x[i][j];
```

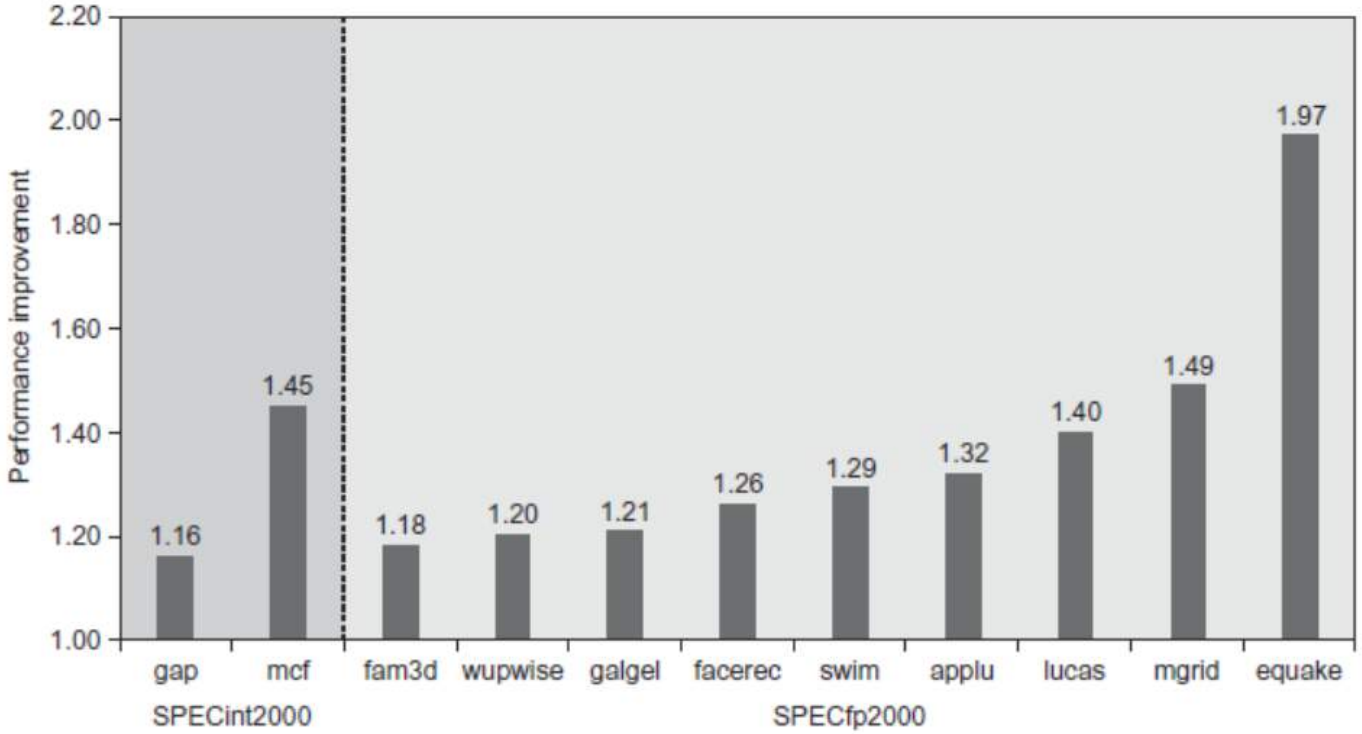
#7: Compiler Optimizations (cont'd)

- Blocking to reduce cache misses
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Exploits a combination of spatial and temporal locality, and can even help register allocation



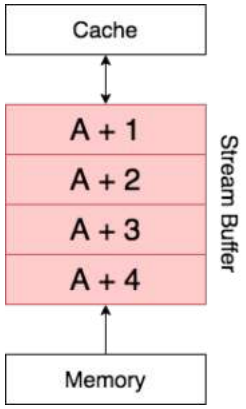
#8: Hardware Prefetching[硬件预取]

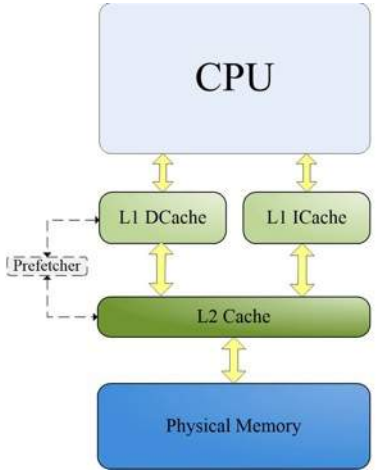
- To reduce miss penalty or miss rate
- Prefetch items before the processor requests them
  - Instruction: fetches two blocks on miss, the requested and the next consecutive
  - Data: prefetch predicted blocks



#8: Hardware Prefetching (cont'd)

- What to prefetch? (prefetch useful data)
  - Next sequential
  - Stride
  - General pattern
- Where to place?
  - Directly into caches
  - External buffers





- When to prefetch?
  - Prefetched data should be timely provided
- Prefetching relies on extra memory bandwidth
  - Should not interfere much with demand accesses
  - Otherwise it hurts performance

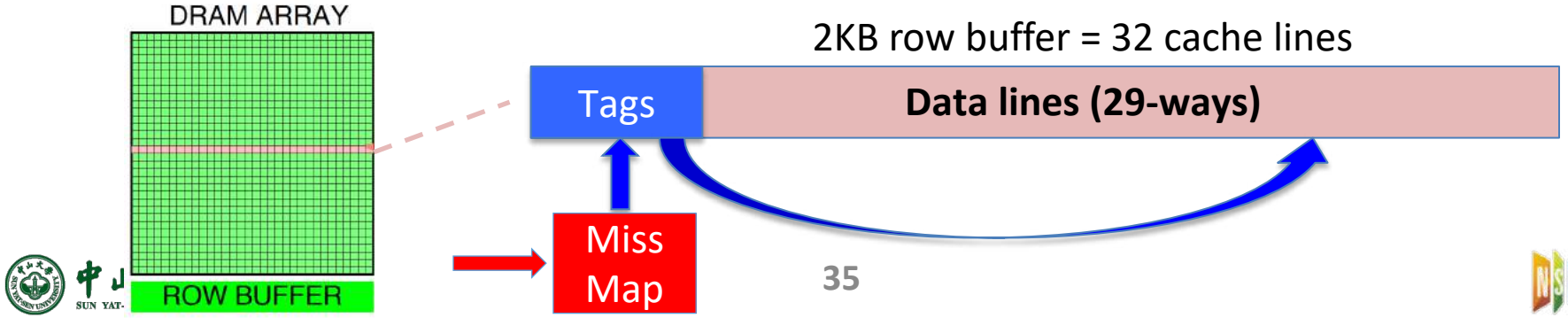
#9: Compiler-controlled Prefetching

- To reduce miss penalty or miss rate
- Compiler inserts prefetch instructions to request data before the processor needs it
- Two flavors
  - Register prefetch: loads the value into a register
  - Cache prefetch: loads data into the cache
- Typically *nonfaulting* prefetches
  - Simply turns into no-ops if they would normally result in an exception
- Compilers must take care to gain performance
  - Issuing prefetch instructions incurs an instruction overhead

#10: Use HBM[高带宽内存]

- Use HBM to build massive L4 caches, size of 128MB - 1GB
- Tags of HBM cache
  - 64B block: 1GB L4 requires 94MB of tags
    - Issue: cannot place in on-chip caches
  - 4KB block: 1GB L4 requires <1MB tag
    - Issues: inefficient use of huge blocks, and high transfer overhead

- One approach (L-H, MICRO'2011):
  - Each SDRAM row is a block index
  - Each row contains set of tags and 29 data segments
  - 29-set associative



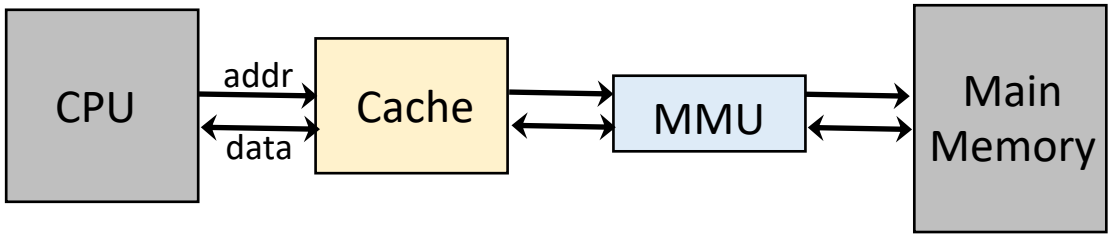
Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs
HBM as additional level of cache		+/–	–	+	+	3	Depends on new packaging technology. Effects depend heavily on hit rate improvements

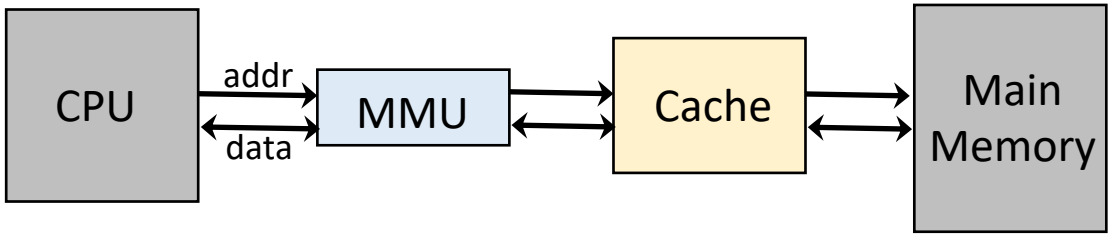
Part-III: Main Memory

Virtual vs. Physical Caches

- Cache works on virtual addresses

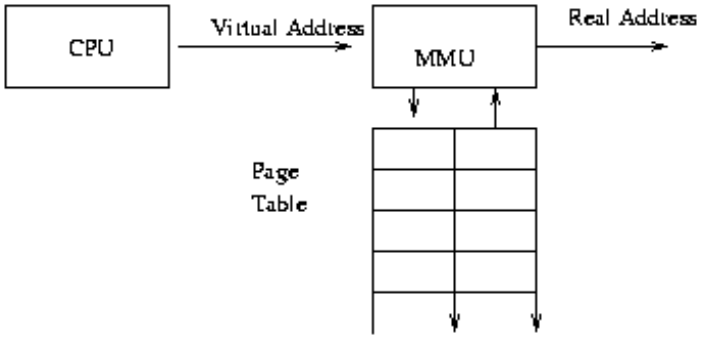


- Cache works on physical addresses



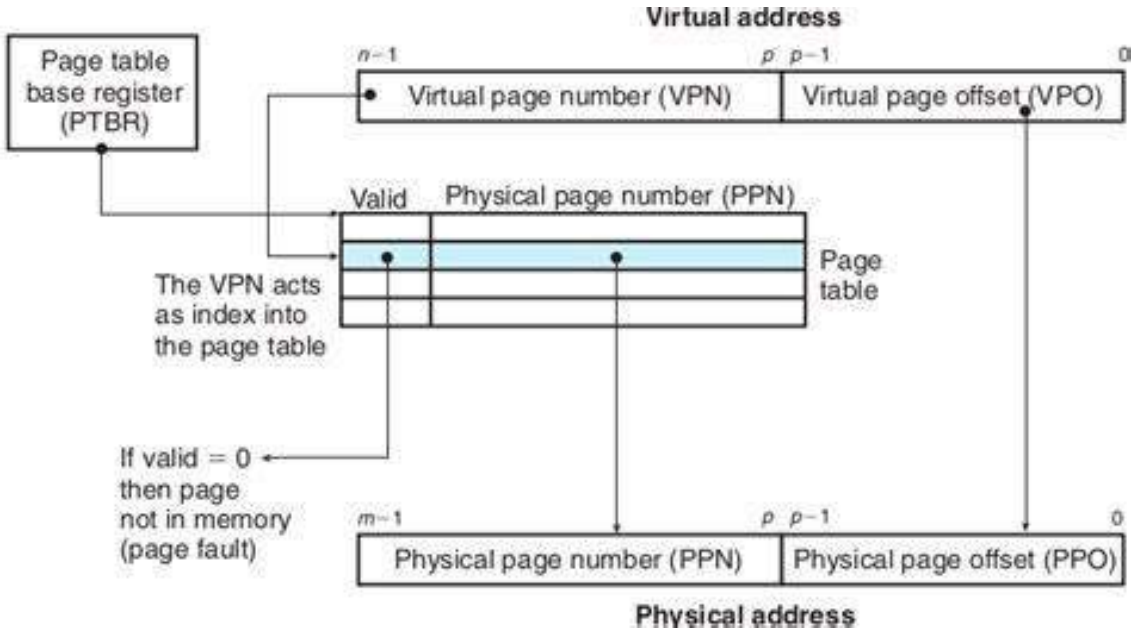
Address Translation[地址转换]

- Address Translation: the hardware converts virtual addresses into physical addresses via an OS-managed lookup table (page table)
- HW and SW cooperatively manage the translation
  - OS software
  - Address translation hardware in MMU
  - Page tables stored in physical memory or disk
- Memory management unit[内存管理单元]
  - Includes Page Table Base Register(s), TLBs, page walkers

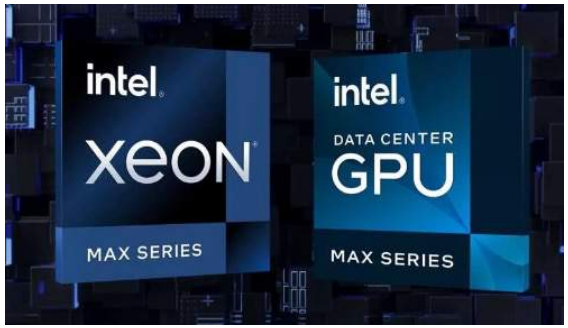


Address Translation (cont.)

- A virtual page is mapped to
  - A physical frame, if the page is in physical memory
  - A location in disk, otherwise
- If an accessed virtual page is not in memory, but on disk
  - Virtual memory system brings the page into a physical frame and adjusts the mapping → this is called *demand paging*



Intel Xeon Max



https://www.servethehome.com/intel-xeon-max-cpu-is-the-sapphire-rapids-hbm-line/

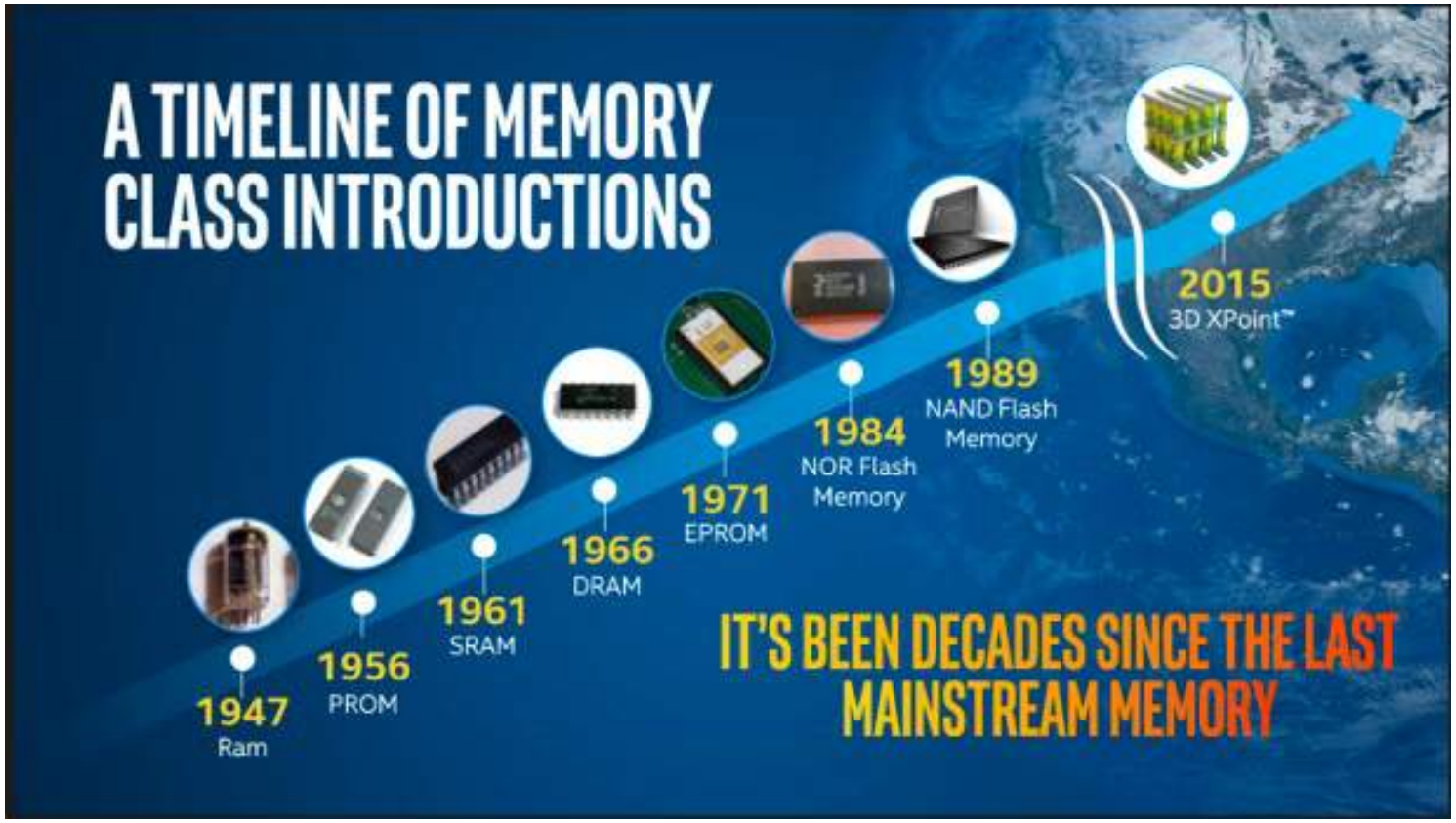
https://www.tomshardware.com/news/intel-fires-up-xeon-max-cpus-gpus-to-rival-amd-nvidia

https://www.intel.com/content/www/us/en/newsroom/news/introducing-intel-max-series-product-family.html#gs.iub4p3

中山大學 SUN YAT-SEN UNIVERSITY

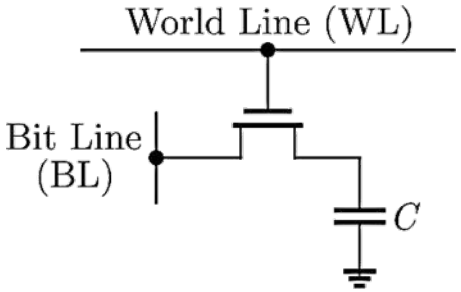
Memory Technology[存储技术]

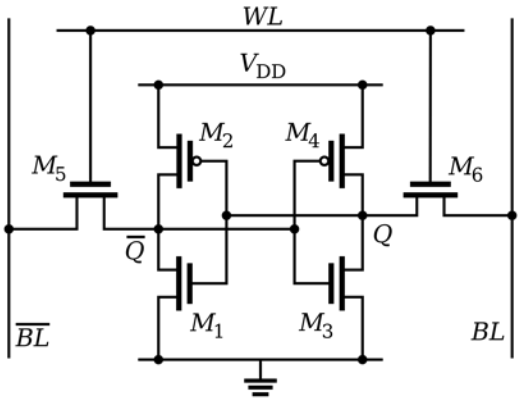
- Performance of main memory
  - Latency: affects Cache Miss Penalty
  - Bandwidth: affects I/O & Large Block Miss Penalty

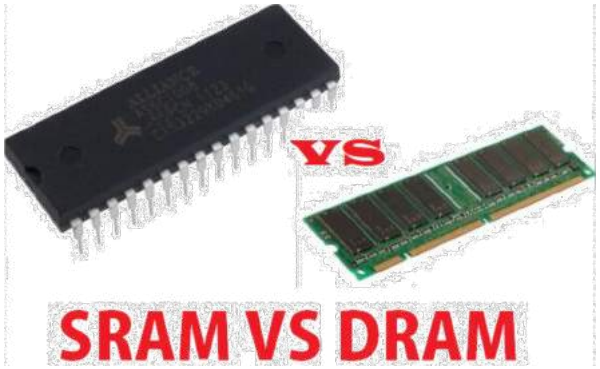


DRAM vs. SRAM

- Main Memory uses *DRAM*: Dynamic Random Access Memory
  - Needs to be refreshed periodically (one row at a time)
  - Addresses divided into 2 halves (memory as a 2D matrix):
    - *RAS* or *Row Access Strobe*
    - *CAS* or *Column Access Strobe*
- Cache uses *SRAM*: Static RAM
  - No refresh (6 transistors/bit vs. 1)
    - *Size*: DRAM/SRAM *4-8*
    - *Cost/Cycle time*: SRAM/DRAM *8-16*



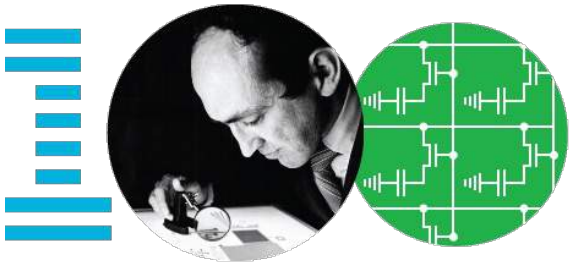




DRAM

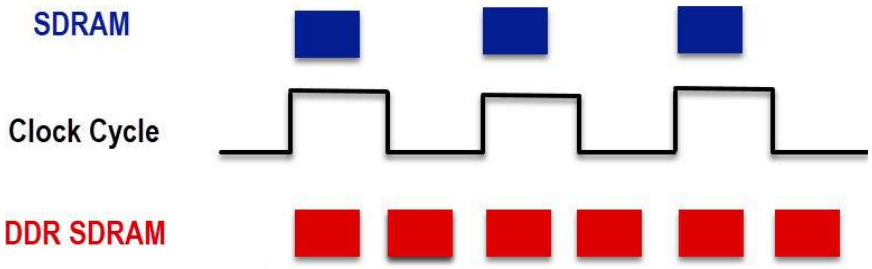
- History
  - 1966: Invented by Robert Dennard of IBM
  - 1967: DRAM patent was filed (issued 1968)
  - 1970: Intel built 1Kb DRAM chip (3T cell)
  - ~1975: 4Kb DRAM chip (1T cell)

Dennard Scaling Law: as transistors shrank, so did necessary voltage and current; power is proportional to the area of the transistor



"I knew it was going to be a big thing, but I didn't know it would grow to have the wide impact it has today."

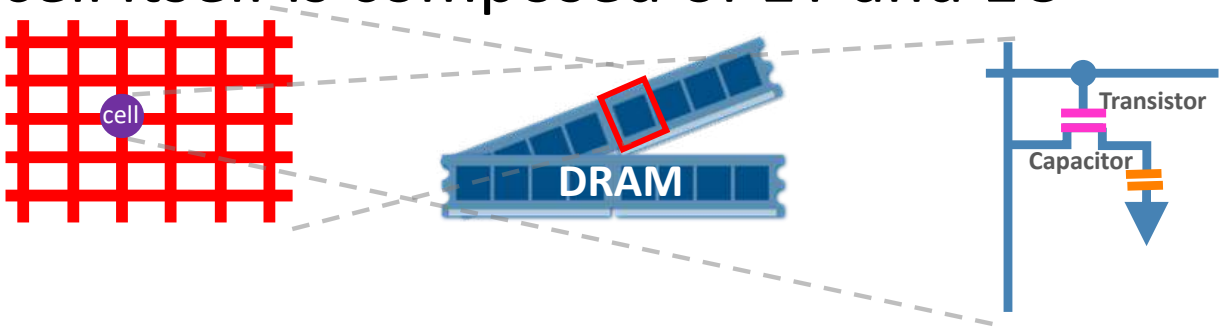
- SDRAM = DRAM with a clocked interface
- DDR SDRAM = double data rate, transfer data at both clock edges
  - DDR1 (2.5 V, 200-400 MHz)
  - DDR2 (1.8 V, 400-1066 MHz)
  - DDR3 (1.5 V, 800-2133 MHz)
  - DDR4 (1.2 V, 1600-5333 MHz)
  - DDR5 (1.1 V, 3200-6400 MHz)



https://www.ibm.com/ibm/history/ibm100/us/en/icons/dram/

DRAM Structure[结构]

- DRAM is provided as DIMMs, which contain a bunch of chips on each side
- DRAM chip can be thought of as 2D array
- Each intersection in the array is one cell
- The cell itself is composed of 1T and 1C



2D Array DIMM/Chip DRAM Cell