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15讲：Memory (2)

张献伟

xianweiz.github.io

DCS5637, 12/7/2022

Review Questions

- memory wall?

  Enlarged processor-memory gap, leaving apps memory-bound

- what is ‘tag’ in cache access?

  Part of address to be used to decide the access is hit/miss.

- cache associativity?

  Cache is organized as sets, each of which contains multi blocks.

- types of cache misses?

  Compulsory, capacity, conflict.

- reduce miss ratio?

  Larger capacity, higher associativity, more levels, larger blocks.

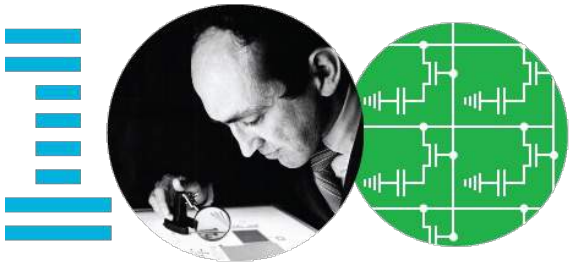
- critical word first?

  Fetch back requested words first to reduce miss penalty.

DRAM

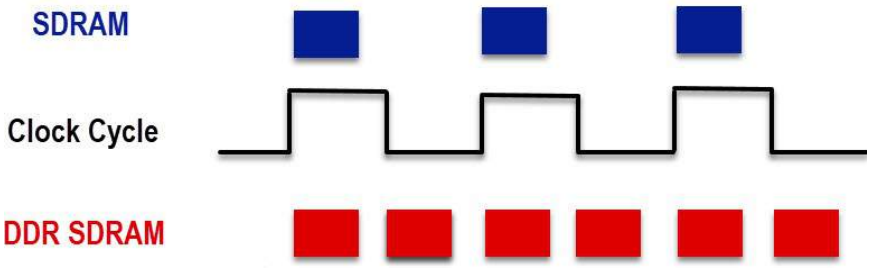
- History
  - 1966: Invented by Robert Dennard of IBM
  - 1967: DRAM patent was filed (issued 1968)
  - 1970: Intel built 1Kb DRAM chip (3T cell)
  - ~1975: 4Kb DRAM chip (1T cell)

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."

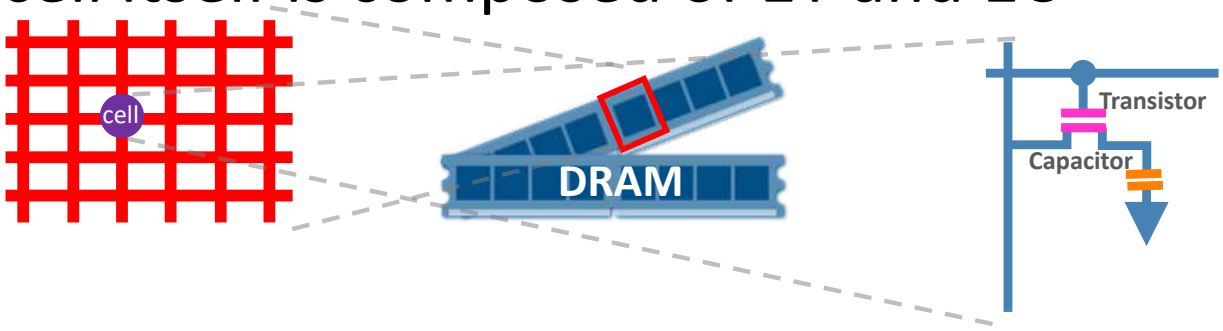
- SDRAM = DRAM with a clocked interface
- DDR SDRAM = double data rate, transfer data at both clock edges
  - DDR1 (2.5 V, 200-400 MHz)
  - DDR2 (1.8 V, 400-1066 MHz)
  - DDR3 (1.5 V, 800-2133 MHz)
  - DDR4 (1.2 V, 1600-5333 MHz)
  - DDR5 (1.1 V, 3200-6400 MHz)



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

DRAM Structure[结构]

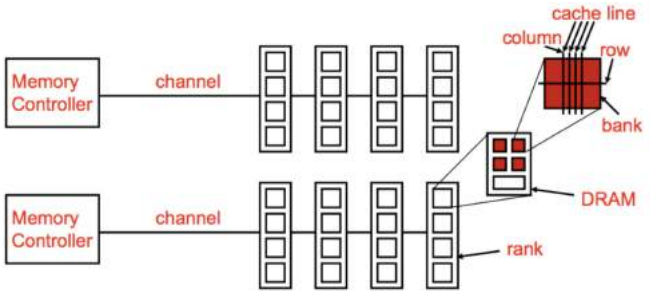
- DRAM is provided as DIMMs, which contain a bunch of chips on each side
- DRAM chip can be thought of as 2D array
- Each intersection in the array is one cell
- The cell itself is composed of 1T and 1C

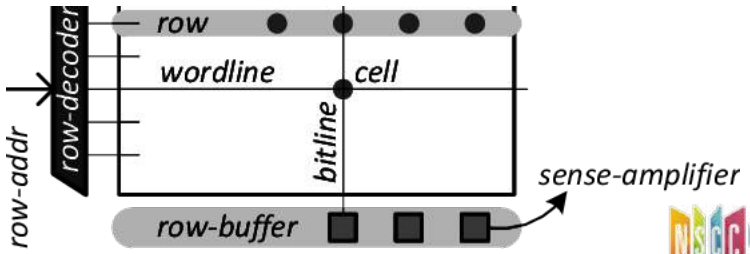


2D Array

DIMM/Chip

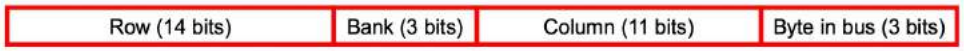
DRAM Cell

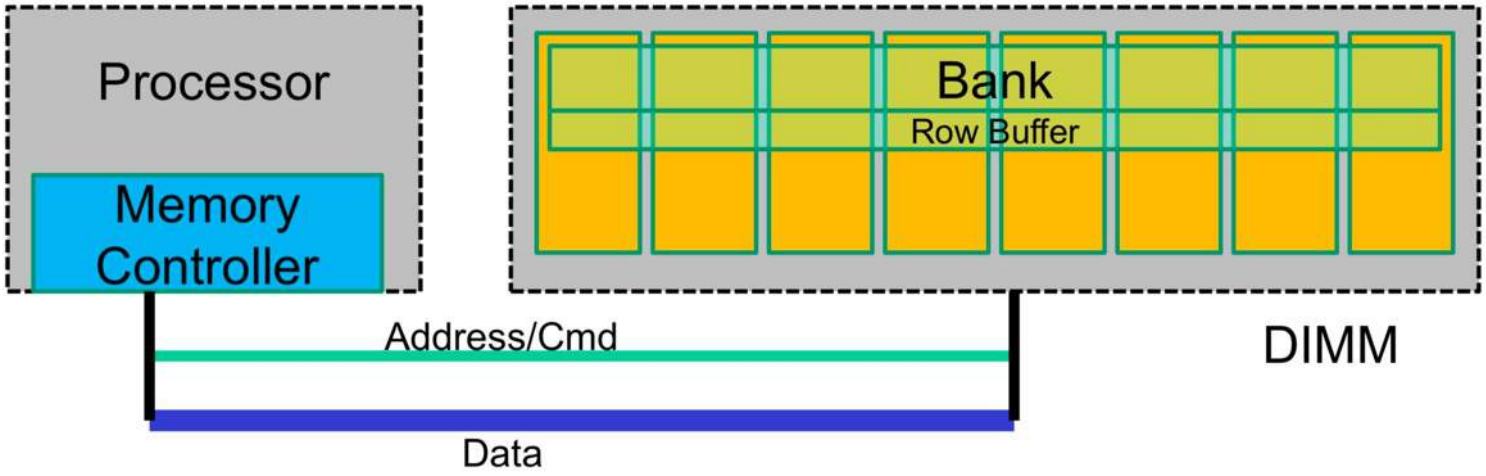




DRAM Structure (cont.)

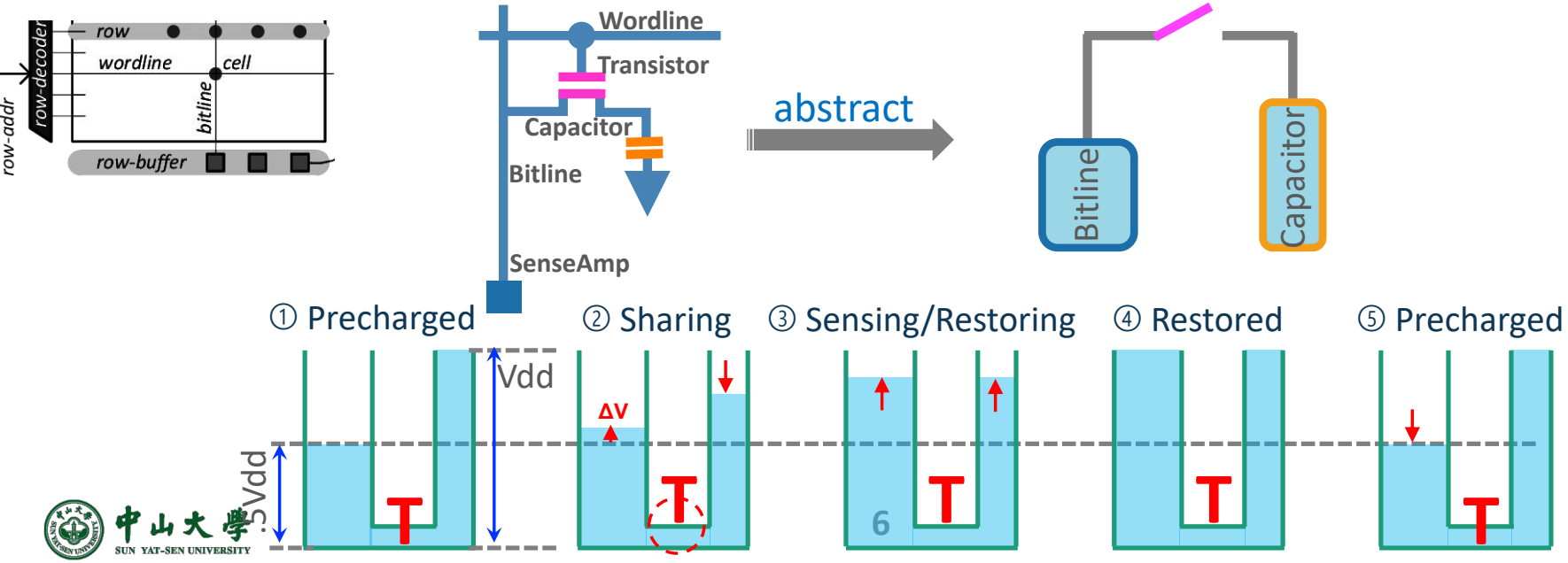
- A rank consists of multiple (parallel) chips contributing to the same transaction
- A memory chip is organized internally as a number of banks (1-8 usually)
  - Physical bank: chip level, a portion of memory arrays
  - Logical bank: rank level, one physical bank from each chip
- Each memory bank has a "row buffer", which is non-volatile (SRAM registers)





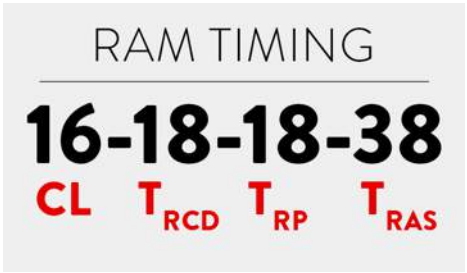
DRAM Operations[操作]

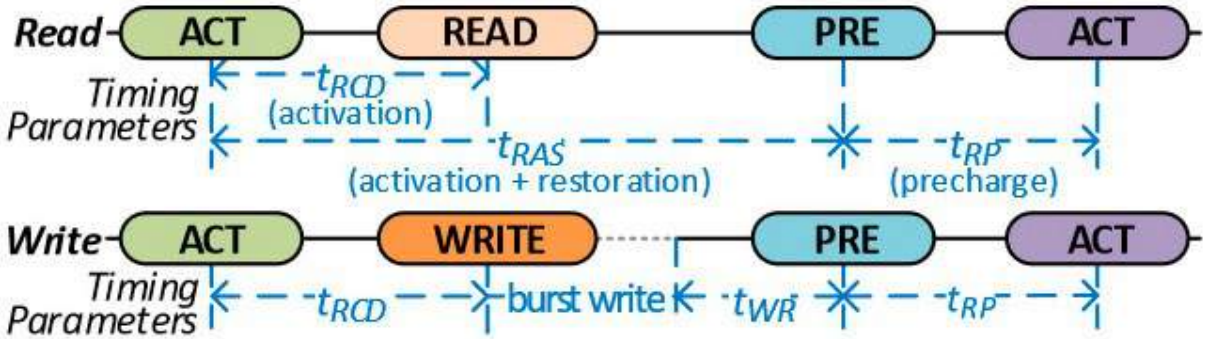
- To read a byte (a similar process applies for writing):
  - The MC sends the row address of the byte
  - The entire row is read into the row buffer (the row is opened)
  - The MC sends the column address of the byte
  - The memory returns the byte to the controller (from the row buffer)
  - The MC sends a Pre-charge signal (close the open row)



Timing Constraints[时序参数]

- Key timings
  - *tRCD*: the minimum number of clock cycles required to open a row and access a column
  - *tCAS*: number of cycles between sending a column address to the memory and the beginning of the data in response
  - *tRAS*: the minimum number of clock cycles required between a row active command and issuing the precharge command
  - *tRP*: number of clock cycles taken between the issuing of the precharge command and the active command
  - *tWR*: write recovery time



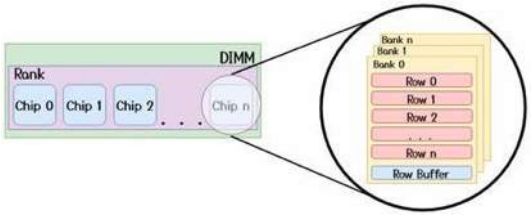


Page Mode[页模式]

- A “DRAM row” is also called a “DRAM page”
  - Usually larger than the OS page, e.g., 8KB vs. 4KB
- Row buffers act as a cache within DRAM
- Open page
  - Row buffer hit: ~20 ns access time (must only move data from row buffer to pins)
  - Row buffer conflict: ~60 ns (must first precharge the bitlines, then read new row, then move data to pins)
- Closed page
  - Empty row buffer access: ~40 ns (must first read arrays, then move data from row buffer to pins)
  - Steps
    - Activate command opens row (placed into row buffer)
    - Read/write command reads/writes column in the row buffer
    - Precharge command closes the row and prepares the bank for next access

Page Mode[页模式]

- A “DRAM row” is also called a “DRAM page”
  - Usually larger than the OS page, e.g., 8KB vs. 4KB
- Row buffers act as a cache within DRAM



- Open page
  - Row buffer hit: ~20 ns access time (must only move data from row buffer to pins)
  - Row buffer conflict: ~60 ns (must first precharge the bitlines, then read new row, then move data to pins)
- Closed page
  - Empty row buffer access: ~40 ns (must first read arrays, then move data from row buffer to pins)
  - Steps
    - Activate command opens row (placed into row buffer)
    - Read/write command reads/writes column in the row buffer
    - Precharge command closes the row and prepares the bank for next access

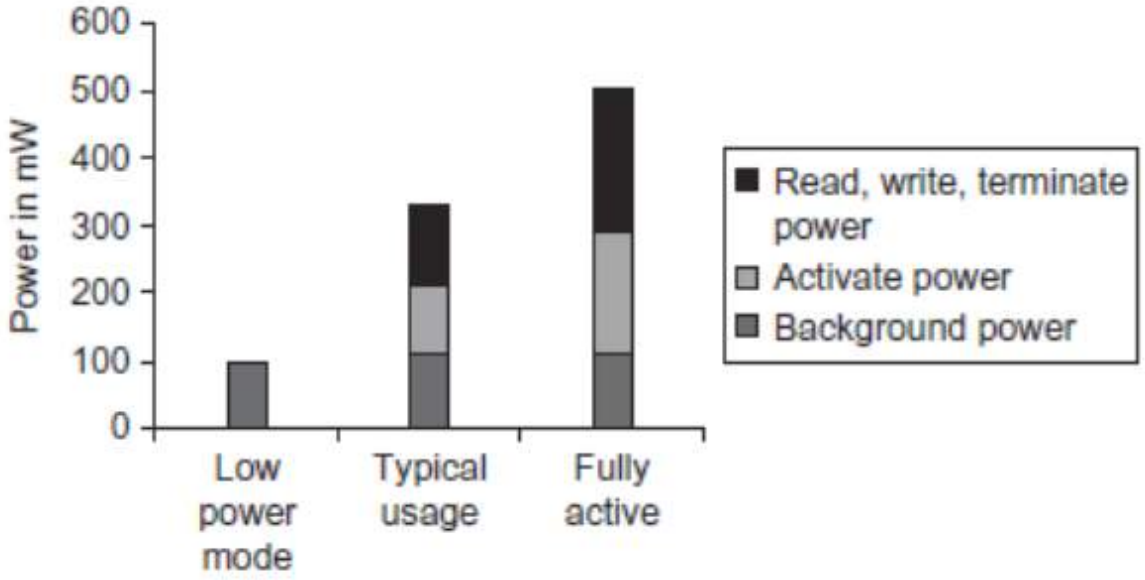
DRAM Bandwidth[带宽]

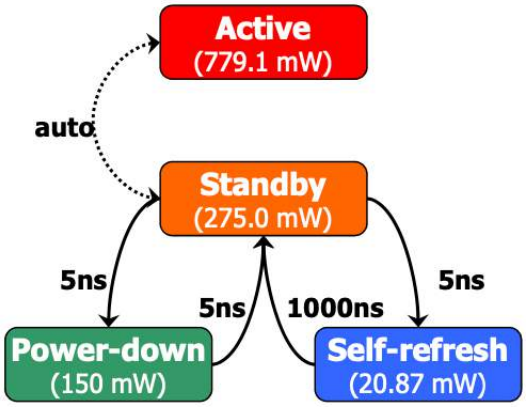
- Reading from a cell in the core array is a very slow process
  - DDR: Core speed = ½ interface speed
  - DDR2/GDDR3: Core speed = ¼ interface speed
  - DDR3/GDDR4: Core speed = ⅛ interface speed
  - … likely to be worse in the future
- Calculation: *transfer_rate * interface_width*
  - Example: 266 MT/s * 64b = 2128 MB/s

Standard	I/O clock rate	M transfers/s	DRAM name	MiB/s/DIMM	DIMM name
DDR1	133	266	DDR266	2128	PC2100
DDR1	150	300	DDR300	2400	PC2400
DDR1	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1333	2666	DDR4-2666	21,300	PC21300

Memory Power[功耗]

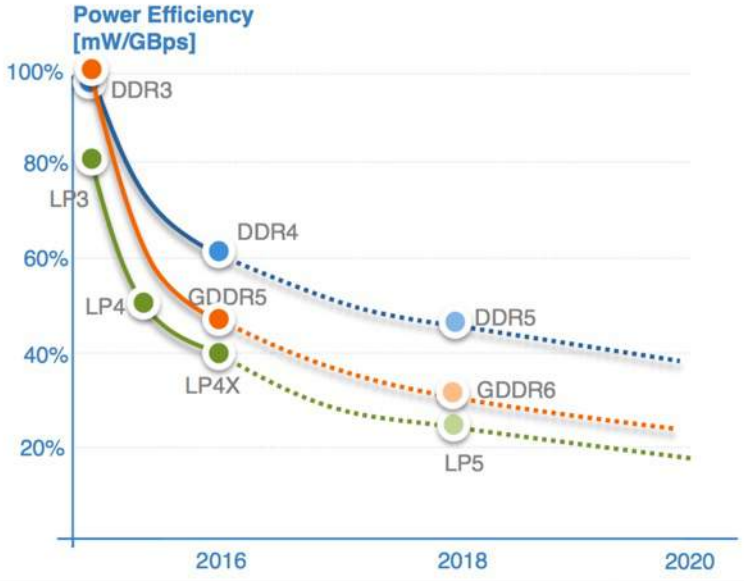
- Dynamic + static[动态和静态]
  - read/write + standby
- Reduce power[降低功耗]
  - Drop operating voltage
  - Power-down mode: disable the memory, except internal automatic refresh





DRAM Variants[变种]

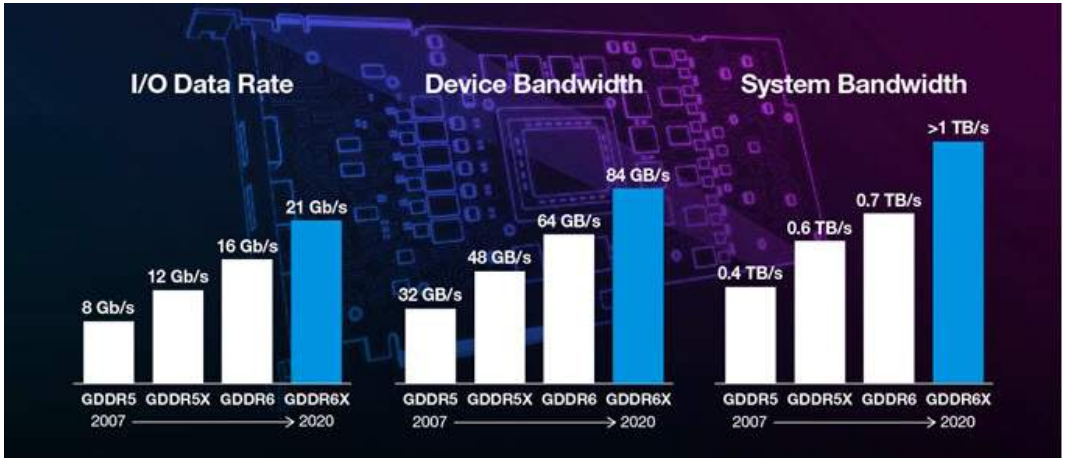
- DDR
  - DDR3: 1.5V, 800MHz, 64b → 1.6G*64b = 12.8GB/s
- GDDR: graphics memory for GPUs
  - GDDR5: based on DDR3, 8Gb/s, 32b → 8G*32b = 32GB/s
- LPDDR: low power DRAM, a.k.a., mobile memory
  - Lower voltage, narrower channel, optimized refresh



Source: ISCA2016, Samsung

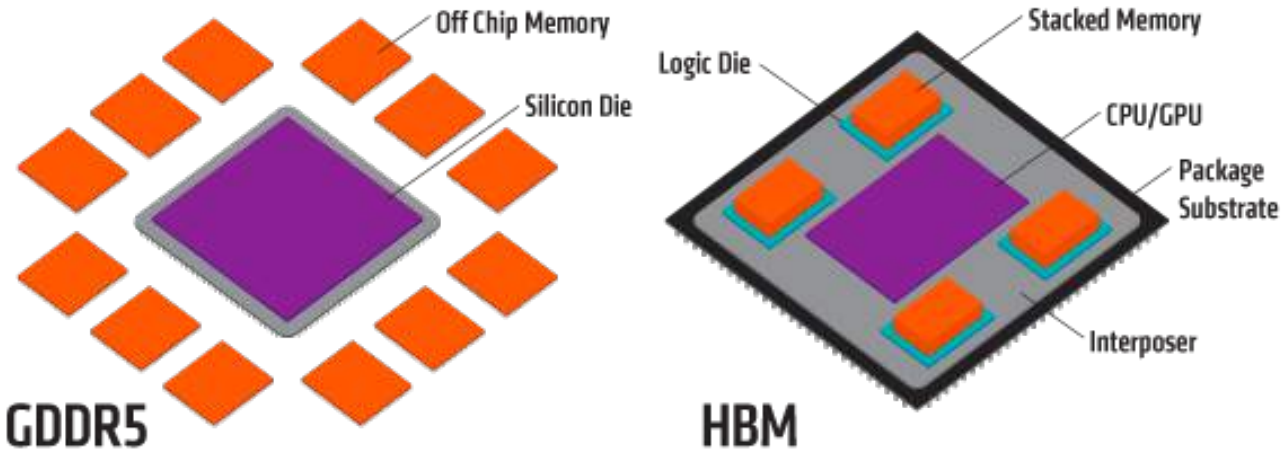
DDR5 & GDDR6

- DDR
  - DDR4: 1-1.2V, 1333MHz, 64b → 21.3GB/s x 4 = 85.2GB/s
  - DDR5: 1.1V, 6.4Gbps, 64b → 51.2GB/s x 4 = 204.8GB/s
- GDDR
  - GDDR5: 8Gb/s(~7), 256b, 224GB/s, 12GB, GTX 980
  - GDDR5X: 12Gb/s(~10), 256b, 320GB/s, 8GB, GTX 1080
  - GDDR6: 16Gb/s(~14), 256b, 448GB/s, 10GB, RTX 2080
  - GDDR6X: 21Gb/s (~19), 320b, 760GB/s, 10GB, RTX 3080
  - GDDR6X: 21Gb/s (~21), 384b, 1008GB/s, 24GB, RTX 4090



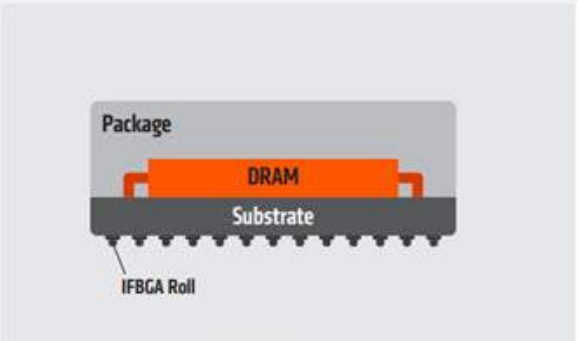
Stacked DRAMs[堆叠]

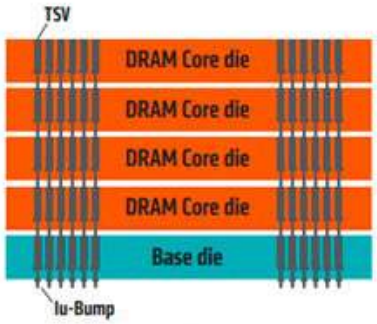
- Stacked DRAMs in same package as processor
  - High Bandwidth Memory (HBM)
- HBM consumes less power and still maintains significantly higher bandwidth in a small form factor
  - To keep the TDP target low, HBM's clock speed is limited to 1GBPs but, it makes up for it with its 4096 bits of the memory bus



HBM[高带宽内存]

- A normal stack consist of four 4 DRAM dies on a base die and has two 128-bit channels per DRAM die
  - Making 8 channels in total which results in a 1024-bit interface
  - 4 HBM stacks gives a width of 4 * 1024 = 4096b, 1Gb/s
  - Bandwidth: 4096b * 1Gb/s = 512GB/s
- Nvidia Tesla P100: HBM2, 4096b, 16GB, 732.2GB/s
- Nvidia Tesla A100: HBM2e, 5120b, 40GB, 1555GB/s

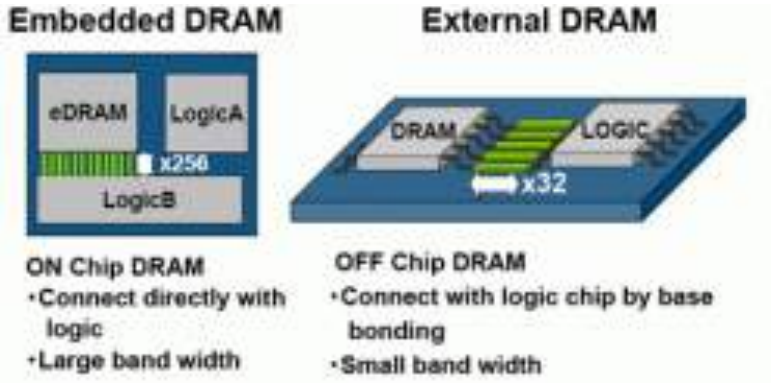




GDDR5	Per Package	HBM
32-bit	Bus Width	1024-bit
Up to 1750MHz (7GBps)	Clock Speed	Up to 500MHz (1GBps)
Up to 28GB/s per chip	Bandwidth	>100GB/s per stack
1.5V	Voltage	1.3V

eDRAM[嵌入式]

- eDRAM: embedded DRAM
  - DRAM integrated on the same die with ASIC/logic
- No pin limitations
  - Can access using a wide on-chip buses
- System power savings
  - Avoids off-chip I/O transfers

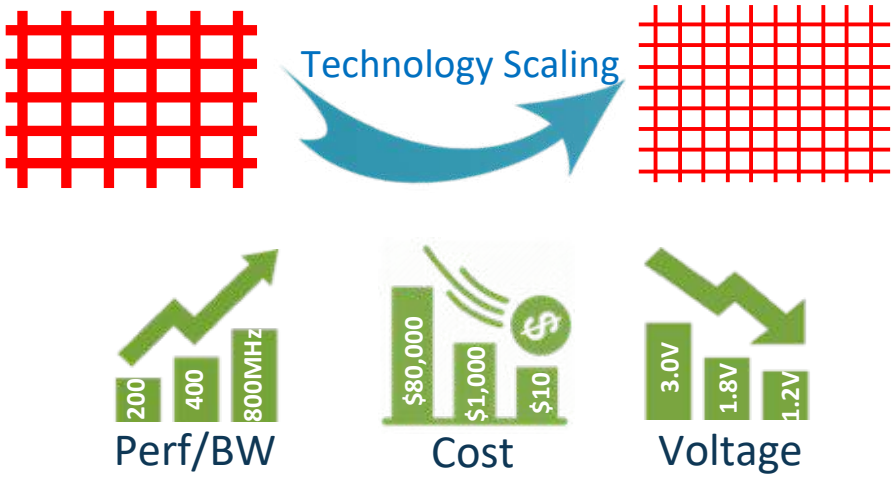


Use of eDRAM in various products

Product name	Amount of eDRAM
IBM z15	256+ MB
IBM's System Controller (SC) SCM, with L4 cache for the z15	960 MB
Intel Haswell, Iris Pro Graphics 5200 (GT3e)	128 MB
Intel Broadwell, Iris Pro Graphics 6200 (GT3e)	128 MB
Intel Skylake, Iris Graphics 540 and 550 (GT3e)	64 MB
Intel Skylake, Iris Pro Graphics 580 (GT4e)	64 or 128 MB
Intel Coffee Lake, Iris Plus Graphics 655 (GT3e)	128 MB
PlayStation 2	4 MB
PlayStation Portable	4 MB
Xbox 360	10 MB
Wii U	32 MB

https://wikimili.com/en/EDRAM

DRAM Scaling[缩放]



			Best case access time (no precharge)			Precharge needed
Production year	Chip size	DRAM type	RAS time (ns)	CAS time (ns)	Total (ns)	Total (ns)
2000	256M bit	DDR1	21	21	42	63
2002	512M bit	DDR1	15	15	30	45
2004	1G bit	DDR2	15	15	30	45
2006	2G bit	DDR2	10	10	20	30
2010	4G bit	DDR3	13	13	26	39
2016	8G bit	DDR4	13	13	26	39

Scaling Issues[问题]

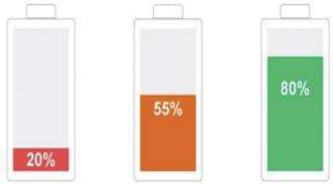
- DRAM cells are more leaky[数据流失]
  - More frequent refreshes
- Slower access[访问时延]
  - Longer sensing and restoring time
- Decreased reliability[可靠性]
  - Cross-talking noise, enlarged process variations

Less charge
higher leakage current

More Leaky

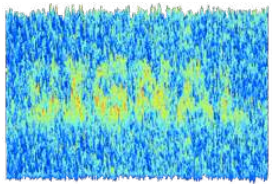
Larger resistance
Weaker signal

Longer Sensing



Larger resistance
Lower voltage

Prolonged Restore

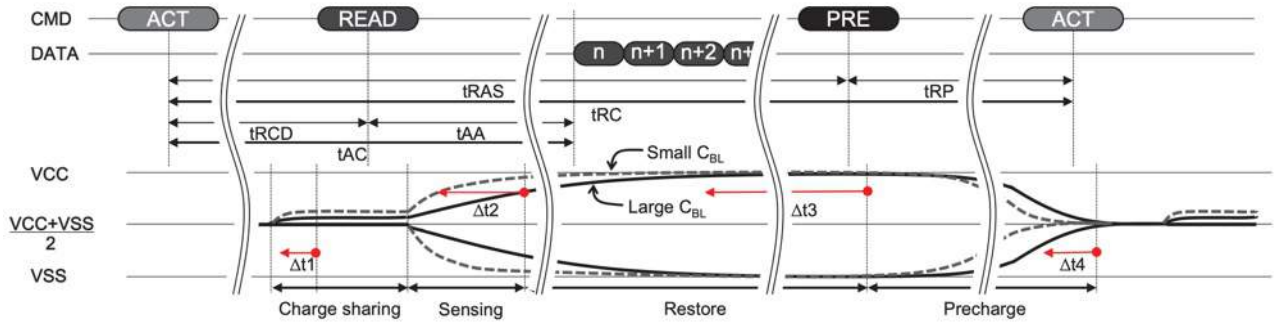


Nearer cells
Process variations

Severer Noise

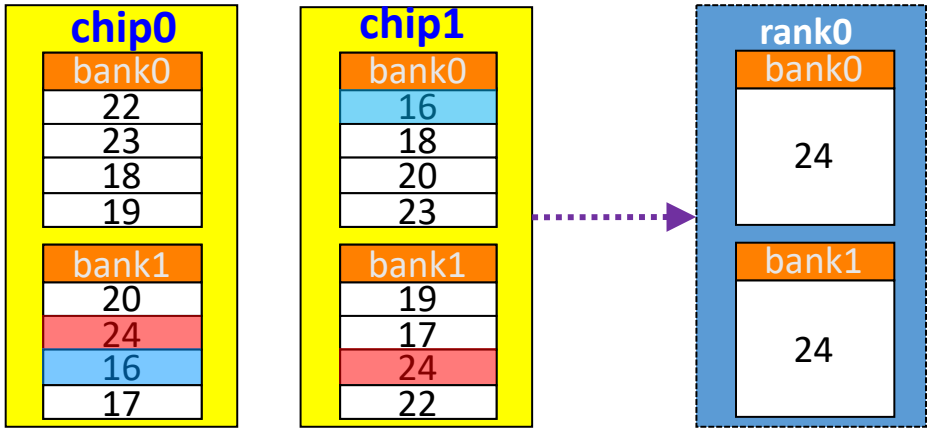
DRAM Researches[前沿研究]

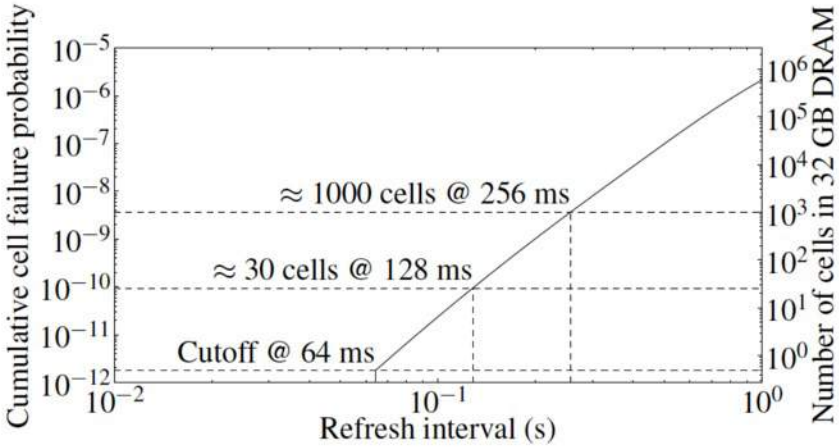
- Sharing/sensing timing reduction[读取时延]
  - Optimize DRAM internal structures [CHARM'ISCA13, TL-DRAM'HPCA13, etc]
  - Utilize existing timing margins [NUAT'HPCA14, AL-DRAM'HPCA15, etc]
- DRAM restore studies[恢复时延]
  - Identify the restore scaling issue [Co-arch'MEM14, tWR'Patent15, etc]
  - Reduce restore timings [AL-DRAM'HPCA15, MCR'ISCA15, RT'HPCA16]
- Memory-based approximate computing[近似计算]
  - Skip DRAM refresh [Flikker'ASPLOS11, Alloc'CASES15, etc]
  - Restore [DrMP'PACT17]



DRAM Researches (cont'd)

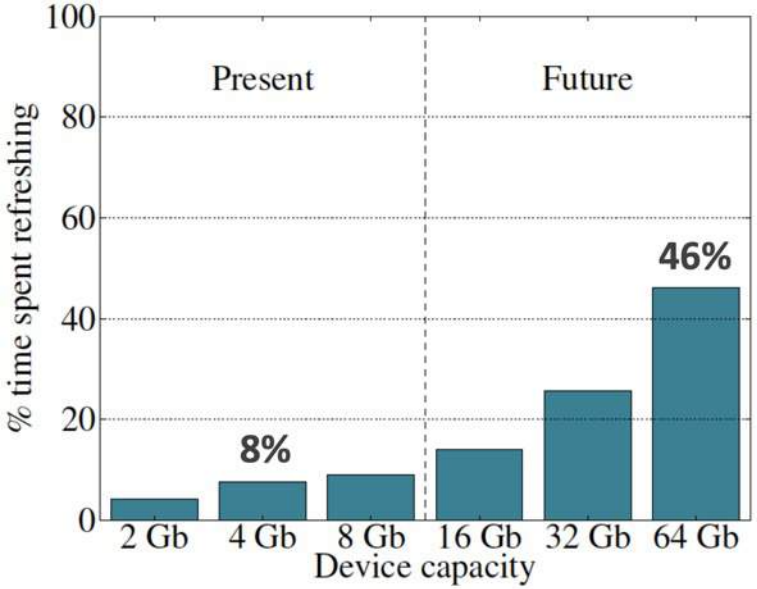
- Nowadays DRAMs are worst-case determined
- Examples:
  - Refresh: only very few rows need to be refreshed at the worst-case rate
  - Timings: overall timing constraints are determined by the worst one
- Idea: use common-case instead

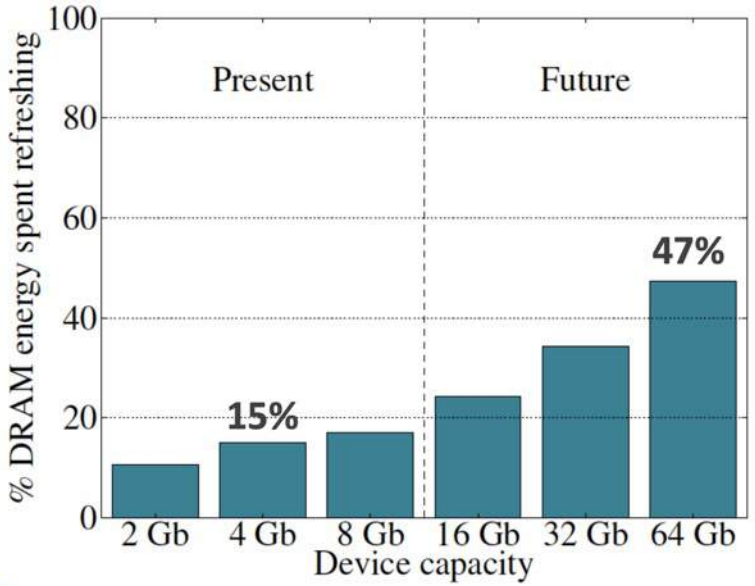




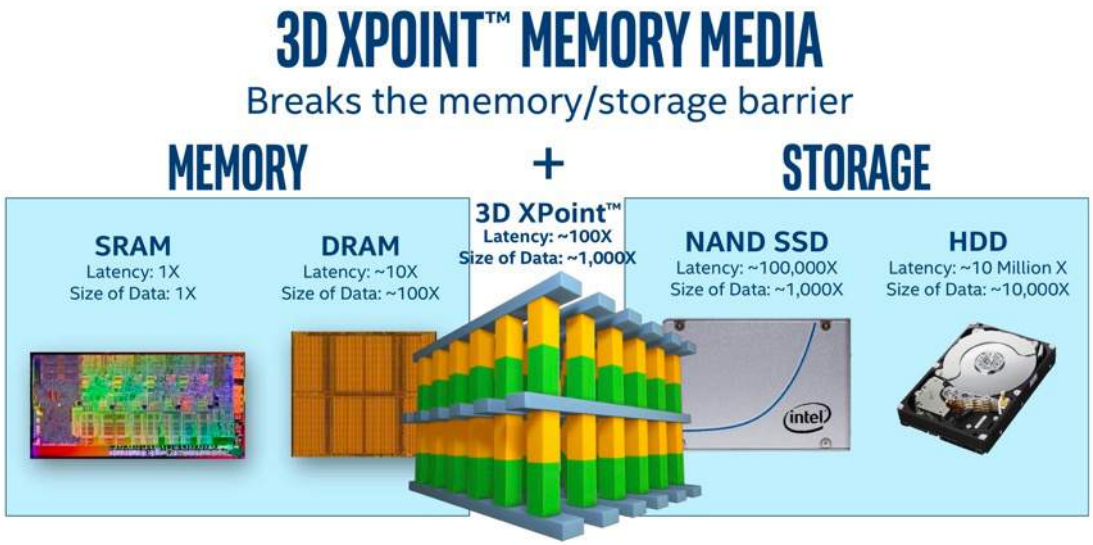
Refresh Issues[刷新问题]

- With higher DRAM capacity, more time will be spent on refresh operations, greatly blocking normal reads/writes
- With further scaled DRAMs, more cells need to be refreshed at likely higher rates than today
- Overheads on both performance and energy





Emerging Memory[新型存储]



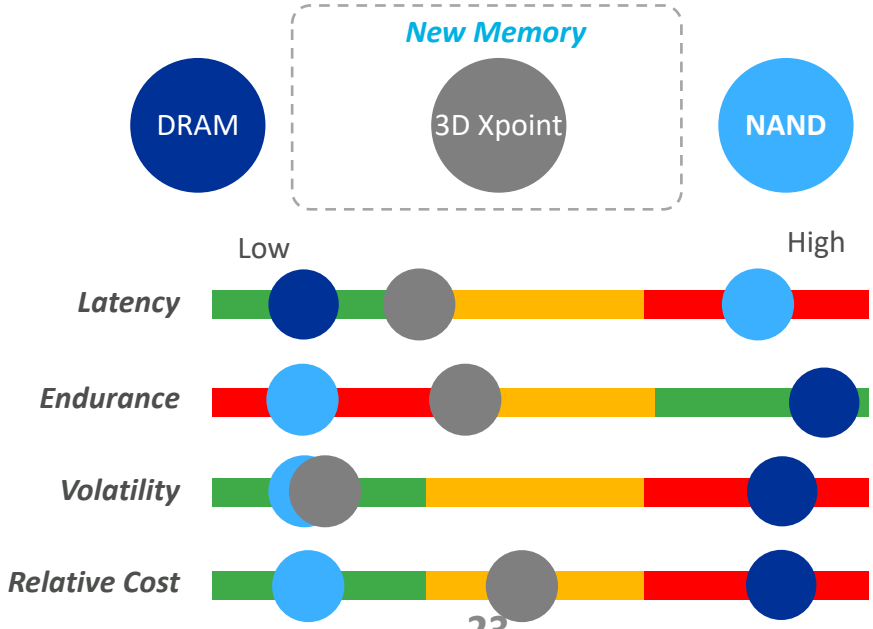
	Hard disk drive (HDD)	Dynamic RAM (DRAM)	NAND single-level cell (SLC) flash	Phase change RAM (PCRAM) SLC	Spin-torque transfer RAM (STT-RAM)	Resistive RAM (ReRAM)
Data retention	Y	N	Y	Y	Y	Y
Cell size (F = feature size)	N/A	6 to $10F^2$	4 to $6F^2$	4 to $12F^2$	6 to $50F^2$	4 to $10F^2$
Access granularity (Bytes)	512	64	4,192	64	64	64
Endurance (writes)	$>10^{15}$	$>10^{15}$	10^4 to 10^5	10^8 to 10^9	$>10^{15}$	10^{11}
Read latency	5 ms	50 ns	25 us	50 ns	10 ns	10 ns
Write latency	5 ms	50 ns	500 us	500 ns	50 ns	50 ns
Standby power	Disk access mechanisms	Refresh	N	N	N	N

NVM[非易失性存储]

- Numerous emerging memory candidates
  - Many fall between NAND and DRAM
- Pros and cons
  - Non-volatility with fraction of DRAM cost/bit
  - Ideal for large memory systems
  - Slower access and limited lifetime

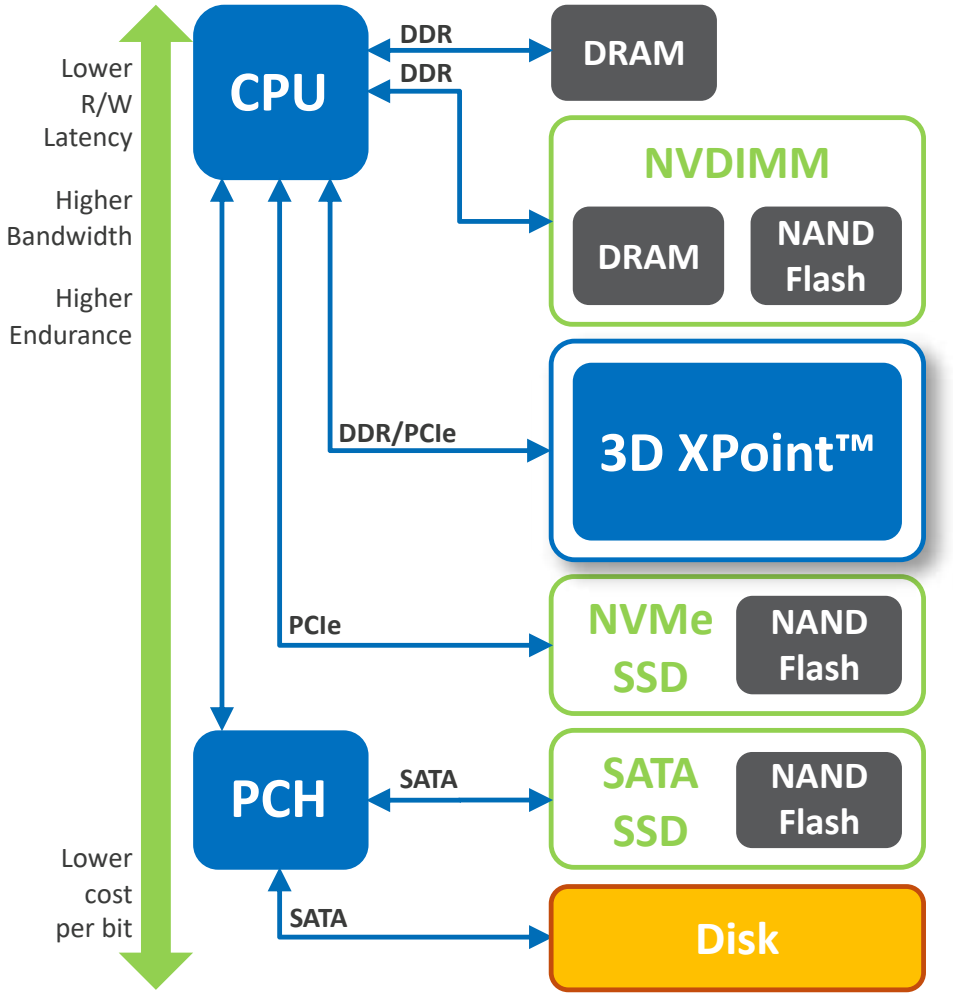
https://www.tomshardware.com/news/intel-kills-optane-memory-business-for-good

https://investors.micron.com/news-releases/news-release-details/micron-updates-data-center-portfolio-strategy-address-growing



Future Memory System[未来存储系统]

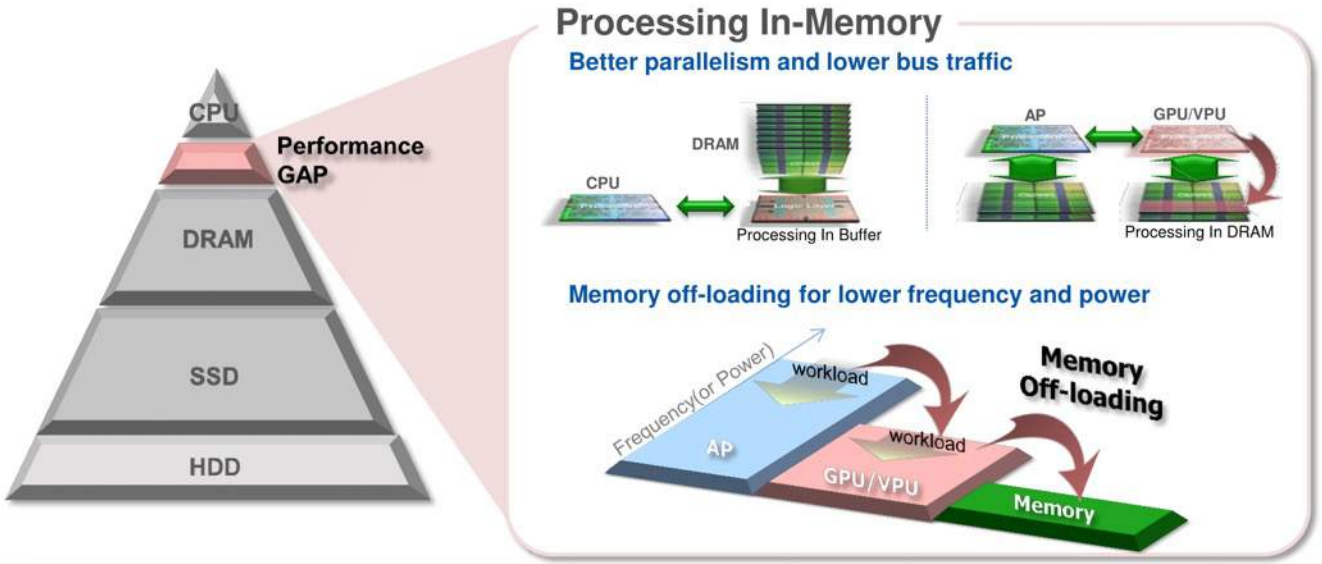
- Demands[需求]
  - Low latency
  - Large size
  - High bandwidth
  - Low power/energy
- Hybrid memory[混合]
  - DRAM + emerging
- Abstracted interface[抽象]
  - Hide device characteristics
- Changing processor-memory relationship[存算]
  - Processor-centric to memory-centric



NDP/PIM[近内存/存内计算]

https://cseweb.ucsd.edu//~swanson/papers/IEEEMicro2014WONDP.pdf

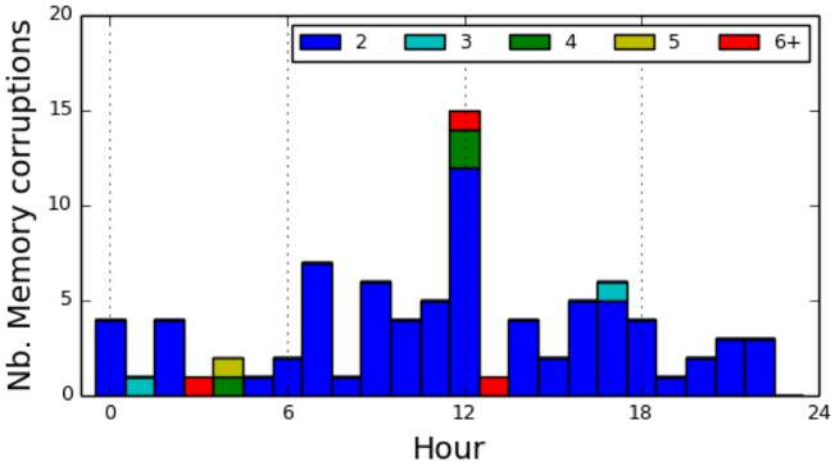
- Near data processing
  - Minimize data movement by computing at the most appropriate location in the hierarchy
  - In NDP, computation can be performed right at the data's home, either in caches, main memory, or persistent storage
- Processing-in-memory
  - Do computation inside the memory



Source: Samsung

Memory Dependability[可靠性]

- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic/transient errors
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use sparse rows to replace defective rows
- Chip-level errors
  - Chipkill: a RAID-like error recovery technique
- *Stuck-at errors*
  - May use data-dependent sparing
- Endurance problems
- Cross-talk (bit-line & word-line)
- Read/write disturbance

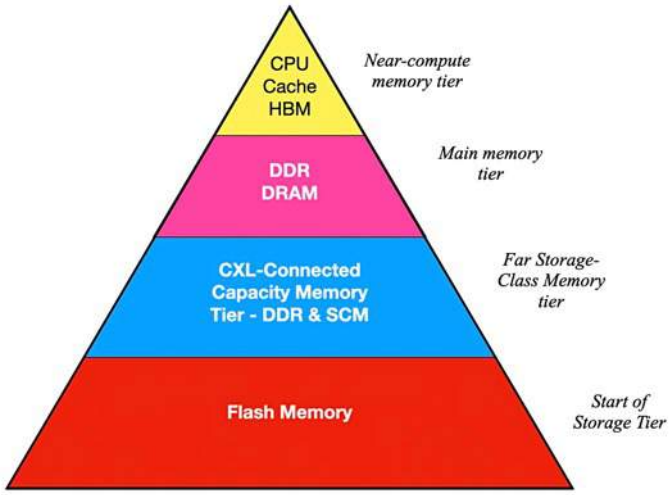


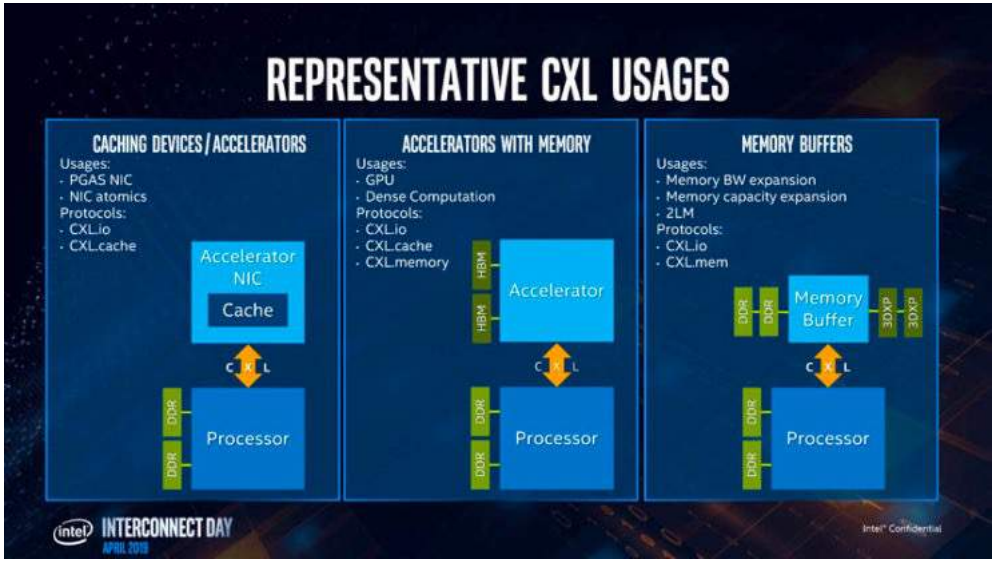
Number of memory errors per hour for multi-bit corruptions

https://upcommons.upc.edu/bitstream/handle/2117/96529/Unprotected%20Computing.pdf

Storage Class Memory (SCM)

- An era of very big, PB-level memory pools
- The big memory pooling is made possible by the compute express link (CXL)
- CXL is a standard for linking memory bus devices together: CPUs, GPUs, and memory (and a few other more exotic things like TPUs and DPUs).





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15讲：TLP (1)

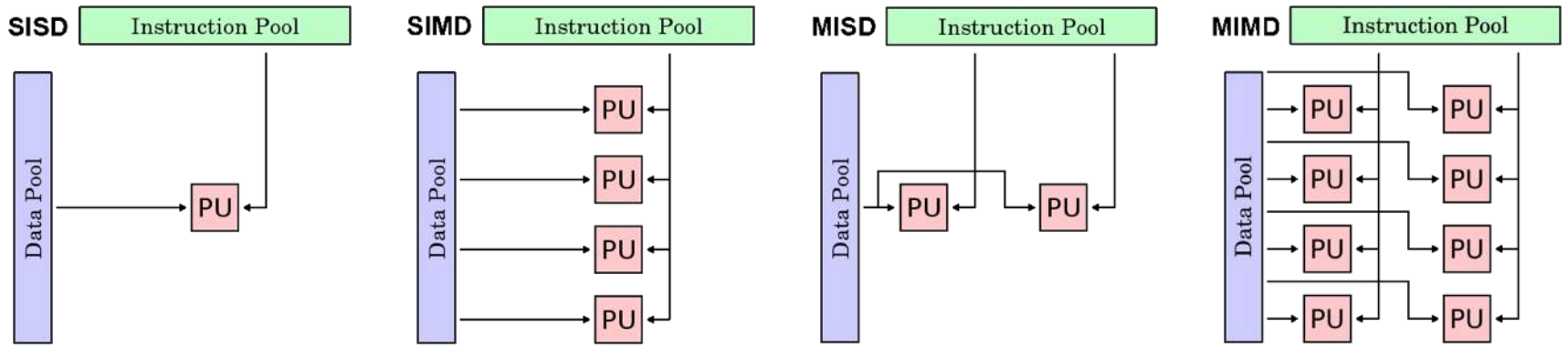
张献伟

xianweiz.github.io

DCS5637, 12/7/2022

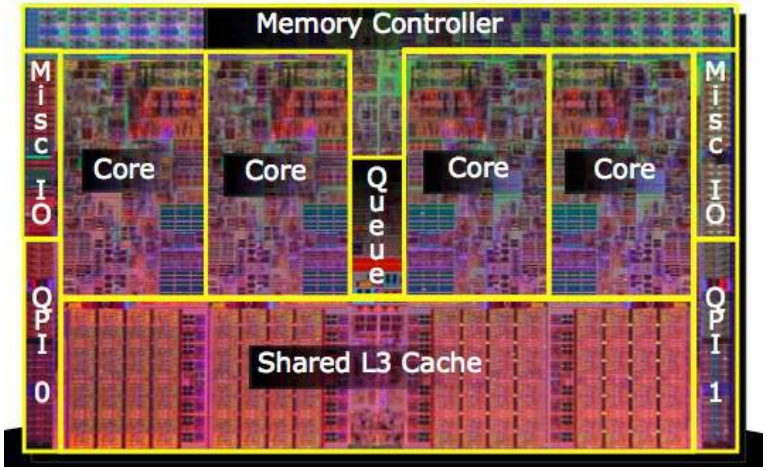
Flynn's Taxonomy[分类]

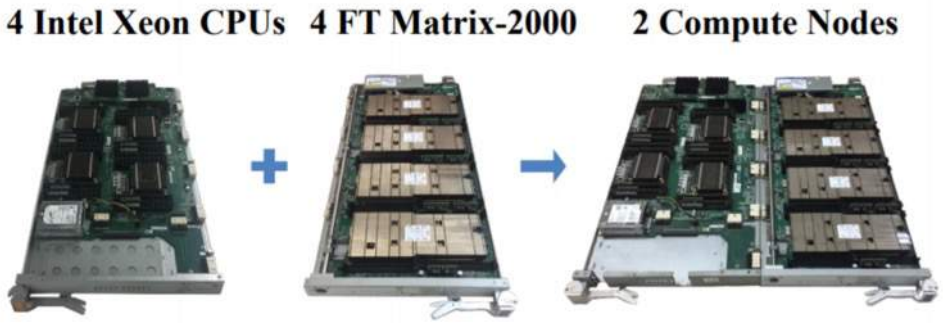
- SISD: single instruction, single data
  - A serial (non-parallel) computer
- **SIMD**: single instruction, multiple data
  - Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing
- MISD: multiple instruction, single data
  - Few (if any) actual examples of this class have ever existed
- MIMD: multiple instruction, multiple data
  - Examples: supercomputers, multi-core PCs, VLIW



MIMD[多指令多数据]

- Machines using MIMD have a number of processors that function asynchronously and independently
- Each processor fetches its own instructions and operates on its own data
- At any time, different processors may be executing different instructions on different pieces of data

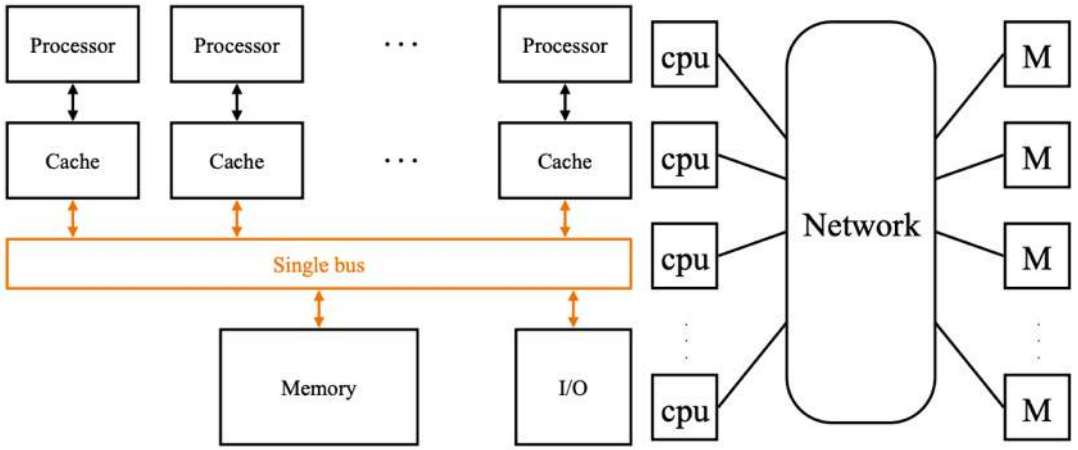




https://www.slideshare.net/abshinde/multiprocessor-74969041

Classifying Multiprocessors[分类]

- Interconnection network[互联网络]
  - Bus
  - Network
- Memory topology[内存]
  - UMA
  - NUMA



- Programming model[编程模型]
  - Shared memory[共享内存]: every processor can name every address location
  - Message passing[消息传递]: each processor can name only it's local memory. Communication is through explicit messages

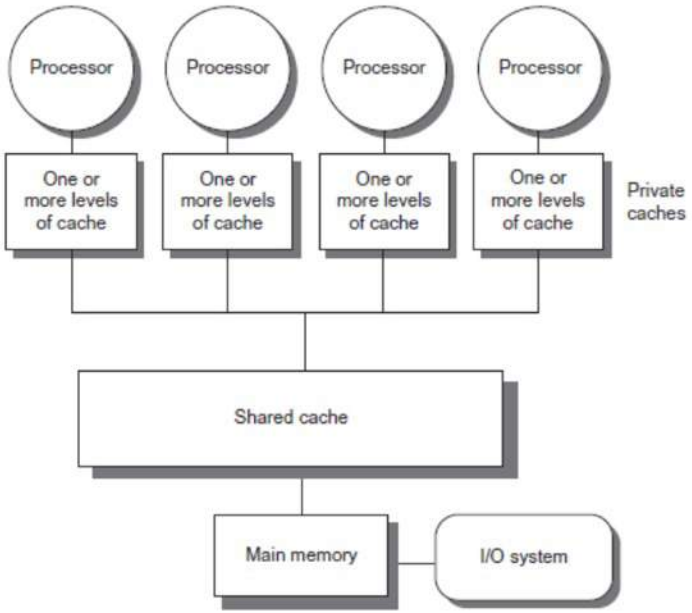
How to Keep Multiprocessor Busy?

- Single core processors exploit ILP
  - Multiprocessors exploit TLP: **thread-level parallelism**
- What's a thread?
  - A program can have one or more threads of control
  - Each thread has its own PC and own arch registers
  - All threads in a given program share resources (e.g., memory)
- OK, so where do we find more than one thread?
  - Option #1: Multi-programmed workloads
    - Run multiple single-threaded programs at same time
  - Option #2: Explicitly multithreaded programs
    - Create a single program that has multiple threads that work together to solve a problem

A Fundamental Turn Toward Concurrency in Software
Your free lunch will soon be over. What can you do about it?
https://www.cs.helsinki.fi/u/kerola/rio/papers/sutter_2005.pdf

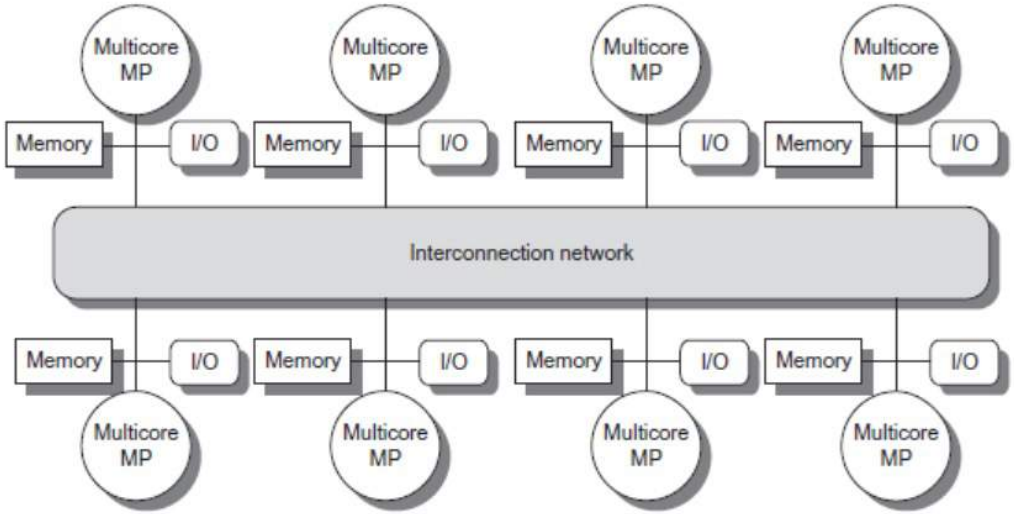
SMP[对称型]

- Symmetric (shared-memory) multiprocessors (SMPs)
  - A.k.a., centralized shared-memory multiprocessors
  - A.k.a., uniform memory access (UMA) multiprocessors
  - Small number of cores (typically <= 8)
  - Share a single centralized memory that all processors have equal access to, hence "symmetric"
    - Uniform access latency



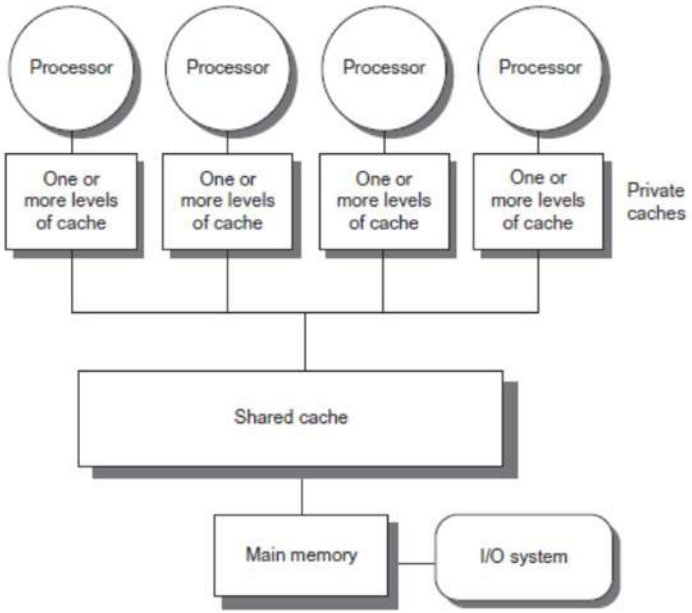
DSM[分布式共享内存]

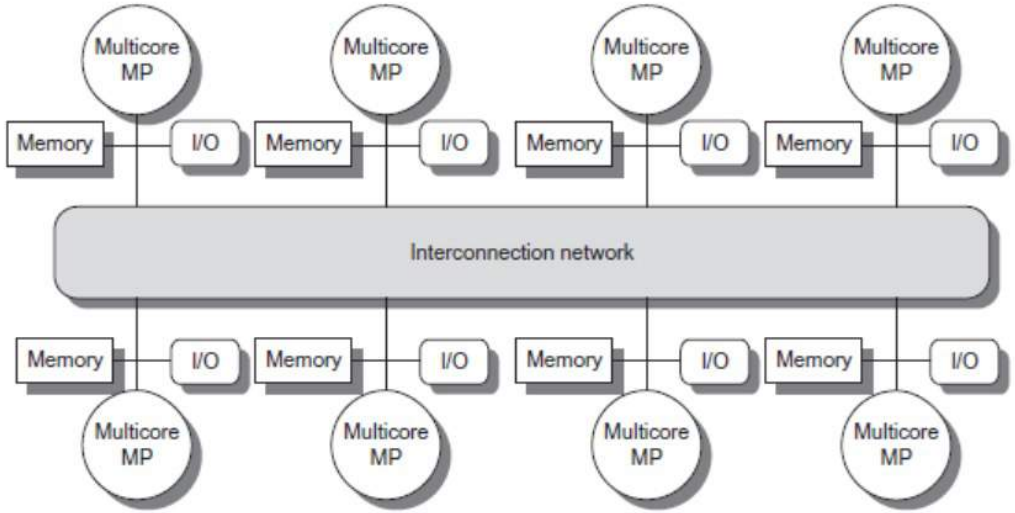
- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Non-uniform memory access/latency (NUMA)
    - The access time depends on the location of a data word in memory
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks



Shared Memory[共享内存]

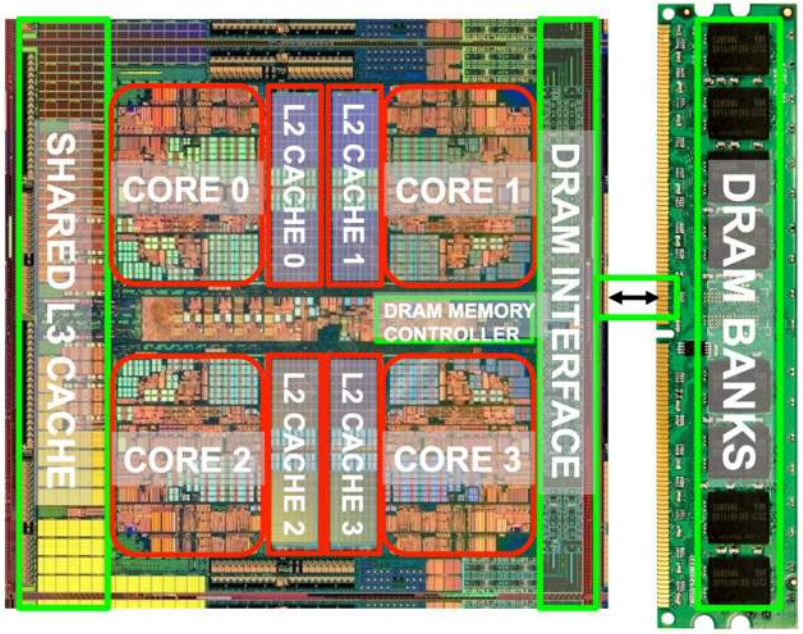
- The term "**shared memory**" associated with both SMP and DSM refers to the fact that the address space is shared
  - Communication among threads occurs through the shared address space
  - Thus, a memory reference can be made by any processor to any memory location

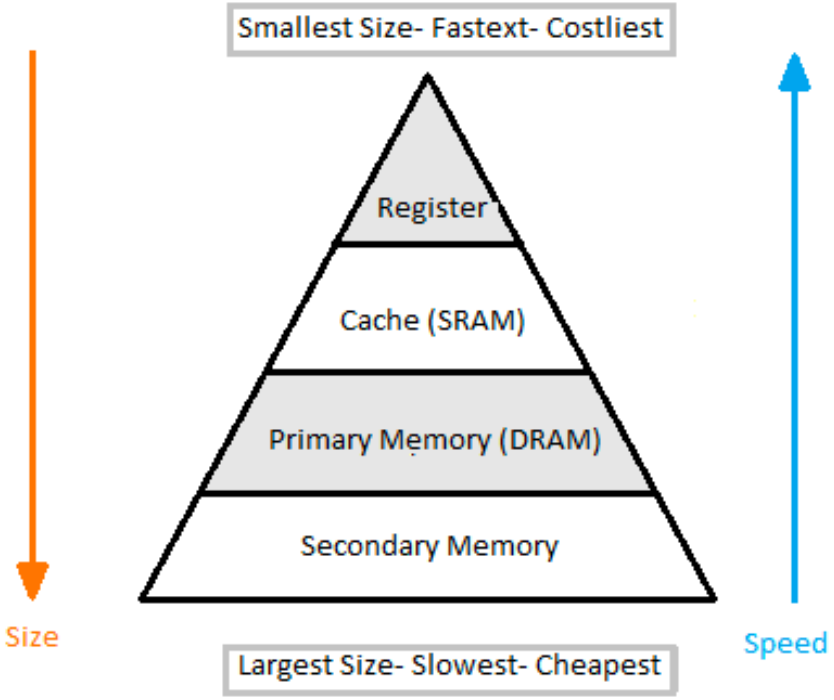




There Exist Caches

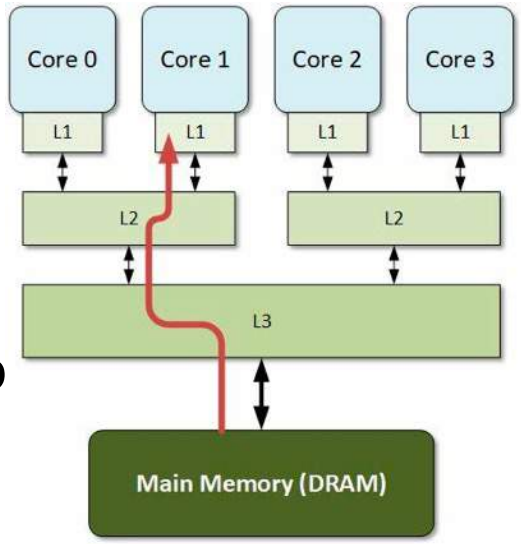
- Recall memory hierarchy, with cache being provided to shorten access latency
  - Each core of multiprocessors has a cache (or multiple caches)
- Caching complicates the data sharing





Data Caching[数据缓存]

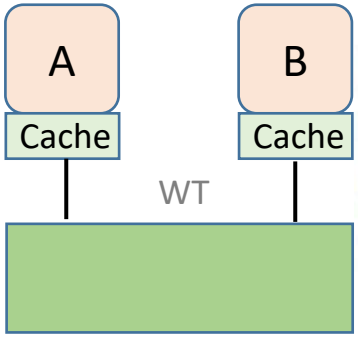
- **Private** data: used by a single processor
- **Shared** data: used by multiple processors
  - Essentially providing communication among the processors through reads and writes of the shared data



- Caching private data
  - Migrated to cache, reducing access time
  - No other processor uses the data (identical to uniprocessor)
- Caching shared data
  - Replicated in multiple caches
    - Reduced access latency, reduced contention
  - Introduces a new problem: **cache coherence**

Cache Coherence[缓存一致性]

- Processors may see different values of the same data
  - The view of memory held by two different processors is through their individual caches, which, without any additional precautions, could end up seeing two different values
- Cache coherence problem[缓存一致性问题]
  - Conflicts between global state (main memory) and local state (private cache)
  - At time 4, what if processor B reads X?



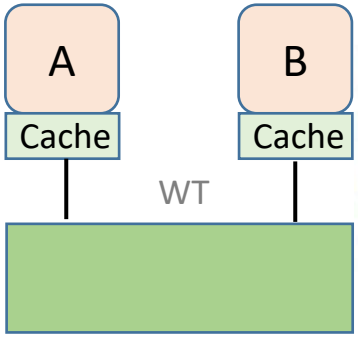
Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Enforcing Coherence[保证一致性]

- Coherent caches provide
  - **Migration**: movement of data[搬运]
    - A data item can be moved to a local cache and used there in a transparent fashion
  - **Replication**: multiple copies of data[备份]
    - Make a copy of the data item in the local cache, so that shared data can be simultaneously read
- Whose responsibility? Software?
  - Can programmer ensure coherence if caches invisible to sw?
  - What if the ISA provided a cache flush instruction?
    - FLUSH-LOCAL A: flushes/invalidates the cache block containing address A from a processor's local cache
    - FLUSH-GLOBAL A: flushes/invalidates the cache block containing address A from all other processors' caches
    - FLUSH-CACHE X: flushes/invalidates all blocks in cache X

Enforcing Coherence (cont.)

- Software solutions are of high overheads
  - And, programming burden
- Multiprocessors adopt a hardware solution to maintain coherent caches[硬件方案]
  - Supporting the migration and replication is critical to performance in accessing shared data
- For the example,
  - Invalidate all other copies of X when A writes to it

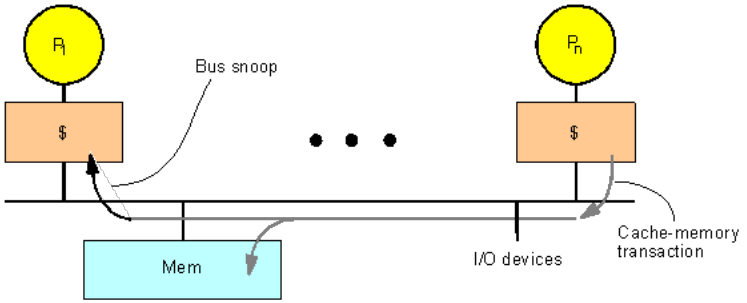


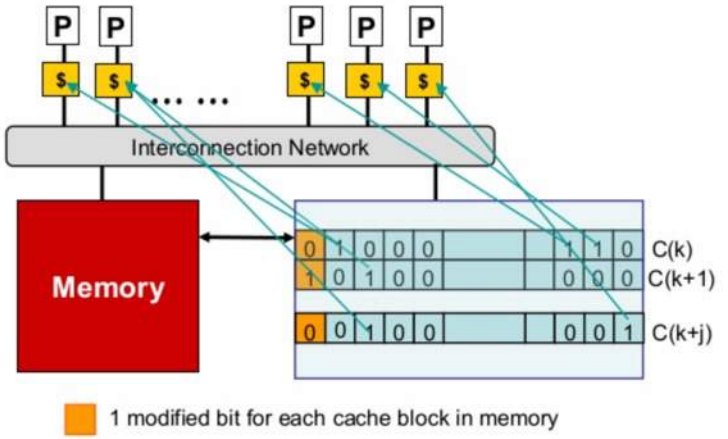
Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1 ✗	0

How do you know which copies to invalidate?

Coherence Protocols[缓存一致性协议]

- Cache **coherence protocols**: the rules to maintain coherence for multiple processors
  - Key is to track the state of any sharing of a data block
- Two classes of protocols
  - Snooping[窥探]
    - Each core tracks sharing status of each block
  - Directory based[基于目录]
    - Sharing status of each block kept in one location





Snooping Coherence Protocols[窥探]

- Write invalidation protocol[写无效]
  - Ensure that a processor has exclusive access to a data item before it writes that item
  - Exclusive access ensures that no other readable or writable copies of an item exist when the write occurs
    - All other cached copies of the item are **invalidated** (👉 that's the name)
- Write update/broadcast protocol[写更新]
  - Update all the cached copies of data item when that item is written
  - Must broadcast all writes to shared cache lines, and thus consumes considerably more bandwidth
- Write invalidation protocol is by far the most common
  - We'll focus on it

Write Invalidation Protocol[写无效]

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
- Example
  - Invalidation protocol working on a snooping bus for a single block (X) with write-back caches

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
		Neither cache initially holds X and the value of X in memory is 0		0
Processor A reads X	Cache miss for X	0		0
		Processor A reads X, migrating from memory to the local cache		
Processor B reads X	Cache miss for X	0	0	0
		Processor B reads X, migrating from memory to the local cache		
Processor A writes a 1 to X	Invalidation for X	1		0
		Processor A writes X, invalidating the copy on B		
Processor B reads X	Cache miss for X	1	1	1
		Processor B reads X, A responds with the value canceling the mem response and updates both B's cache and memory		

MSI Protocol

- Invalidation protocol for write-back caches
- Each data block can be[数据块状态]
  - Uncached: not in any cache
  - Clean in one or more caches and up-to-date in memory, or
  - Dirty in exactly one cache

Dirty in more caches???

- Correspondingly, we record the coherence state of each block in a cache as[一致性状态]
  - Invalid: block contains no valid data
  - Shared: a clean block (can be shared by other caches), or
  - Modified/Exclusive: a dirty block (cannot be in any other cache)

MSI protocol = Modified/Shared/Invalid

Makes sure that if a block is dirty in one cache, it is not valid in any other cache and that a read request gets the most updated data