中山大學
SUN YAT-SEN UNIVERSITY

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

Computer Architecture

计 算 机 体 系 结 构

第15讲：Memory（5）

张献伟

xianweiz.github.io

DCS3013, 11/23/2022

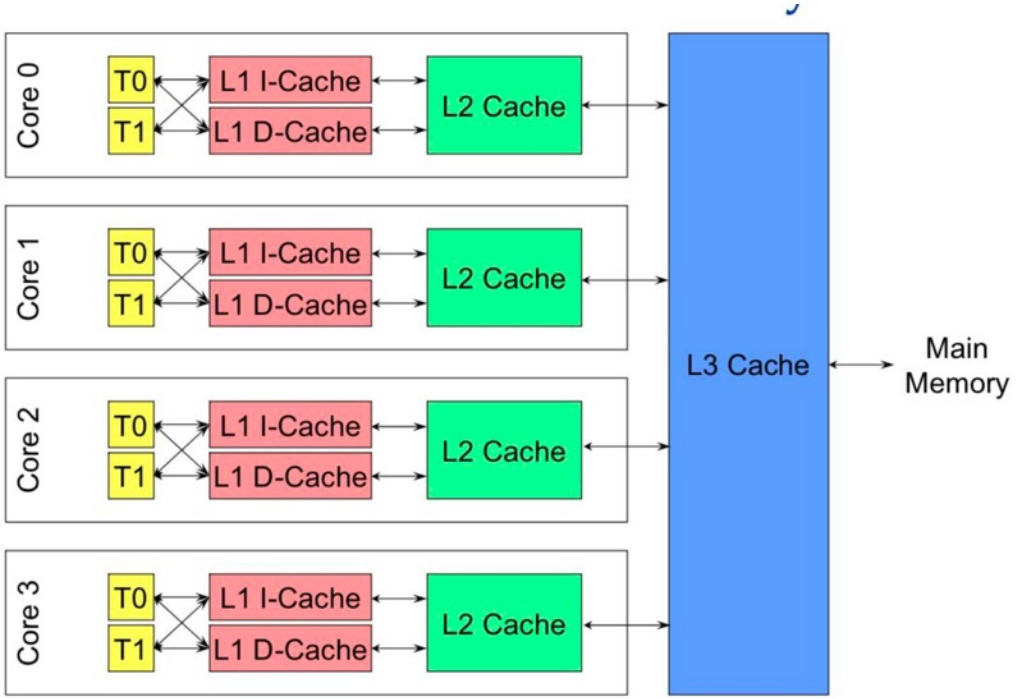
Review Questions

- DRAM page?
  - A DRAM row; a row of sense amplifiers; larger than OS page
- open-page policy?
  - After accessing a row, leave it open (no need to reopen for a hit)
- DDR-1000MHz, 64b interface, what's the bandwidth?
  - 1G x 2 x 64b/8 = 8 GB/s
- sort DDR/HBM/GDDR in bandwidth ascending order?
  - DDR -> GDDR -> HBM (or, HBM -> DDR -> GDDR)
- DRAM scaling issues?
  - Refresh, longer sensing, reliability, power, …
- NVM vs. DRAM?
  - Larger capacity, slower access, lower cost, less power, …

To Further Optimize Cache[优化缓存]

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

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

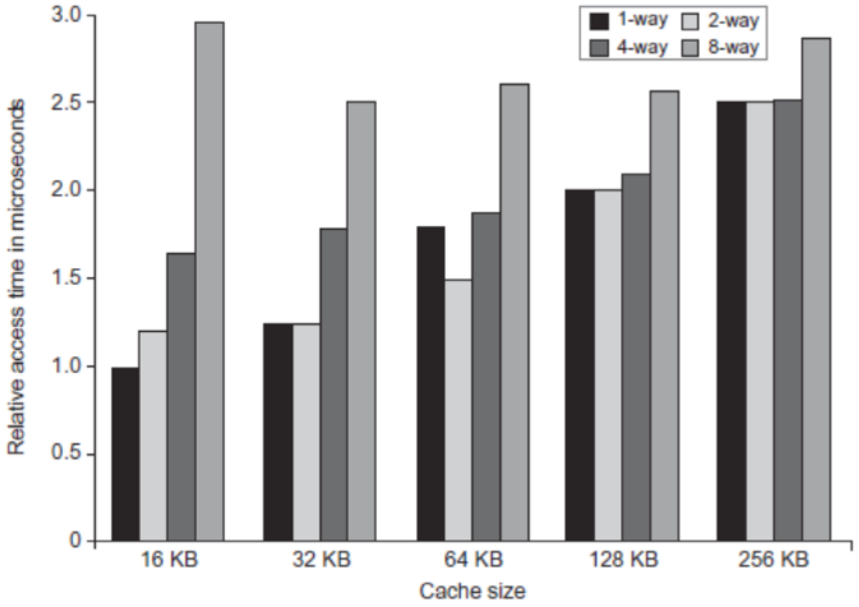


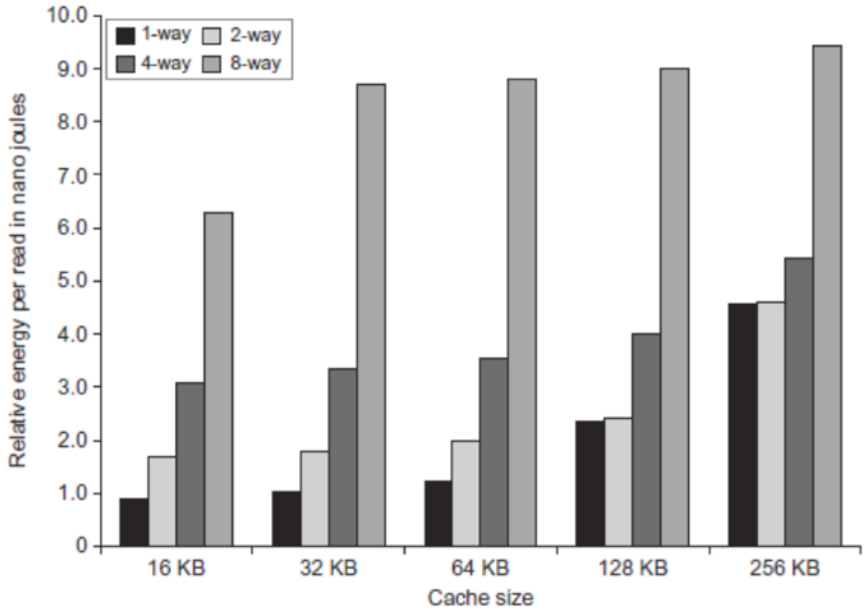
Advanced Cache Optimizations[优化]

- 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]

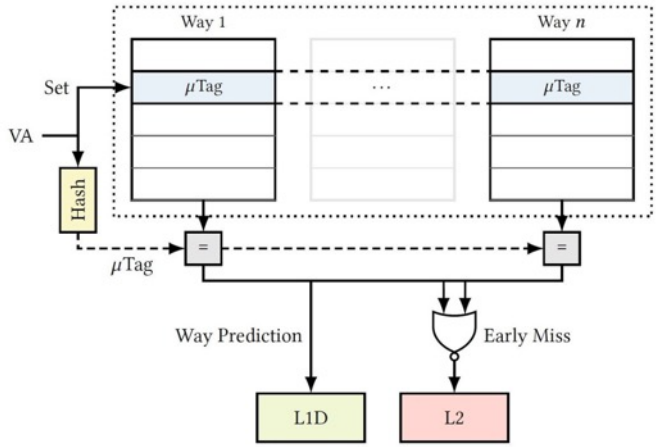
- To reduce hit time and power
- The L1 cache size has recently increased either slightly or not at all
  - Limited size: pressure of both a fast clock cycle and power limitations encourages small sizes
  - Lower level of associativity: reduce both hit time and power





#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

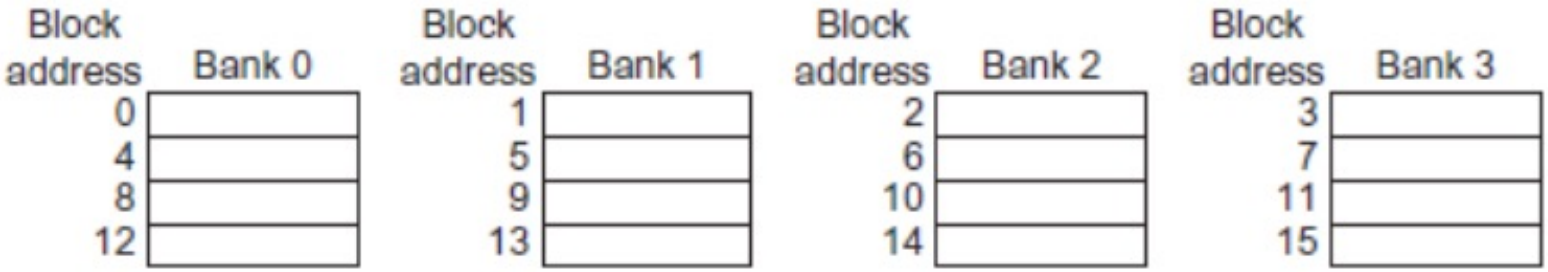


#3: Pipelined[流水线]

- 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

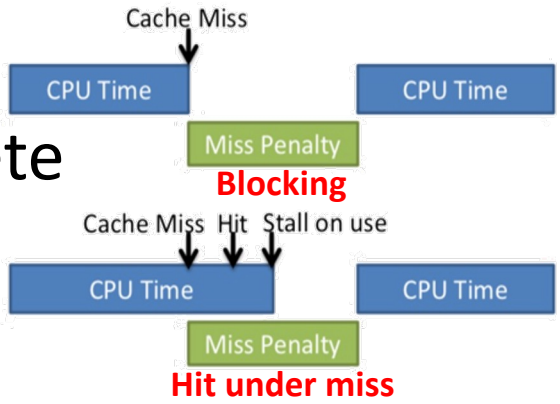
#3: Multibanked[多单元]

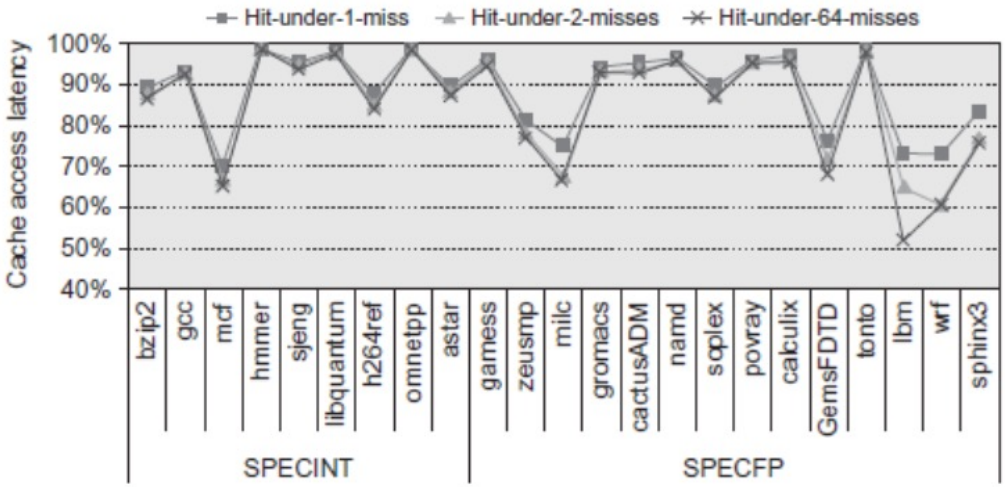
- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Interleave banks according to block address
  - Banking works best when the accesses naturally spread across banks
- Multiple banks 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)



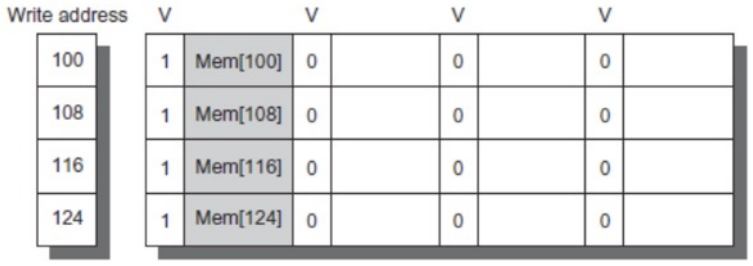


#5: Critical Word First & Early Restart

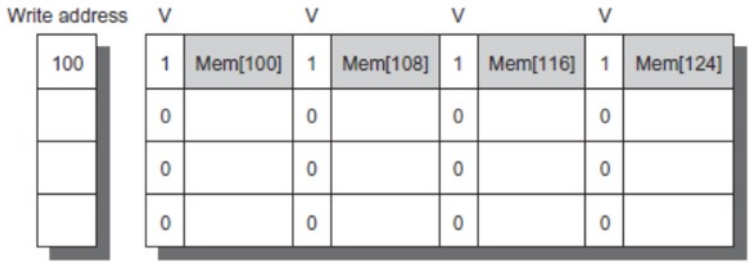
- To reduce miss penalty
- Processor normally needs just one word of the block at a time
  - Don't wait for the full block to be loaded before sending the requested word and restarting the processor
- Critical word first[关键字优先]
  - Request missed word from memory first
  - Send it to the processor as 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

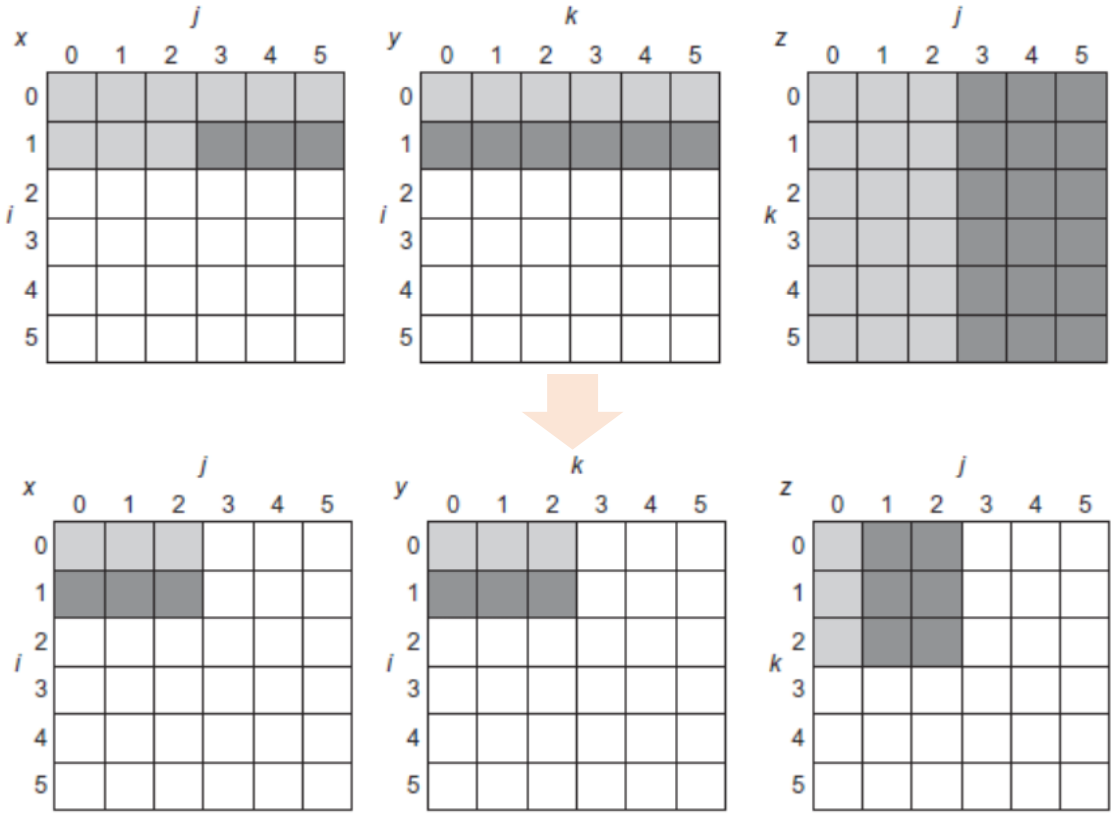
```
/* Before */
for ( j = 0; j < 100; j = j + 1 )
      for ( i = 0; i < 5000; i = i + 1 )
            x[i][j] = 2 * x[i][j];
```



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

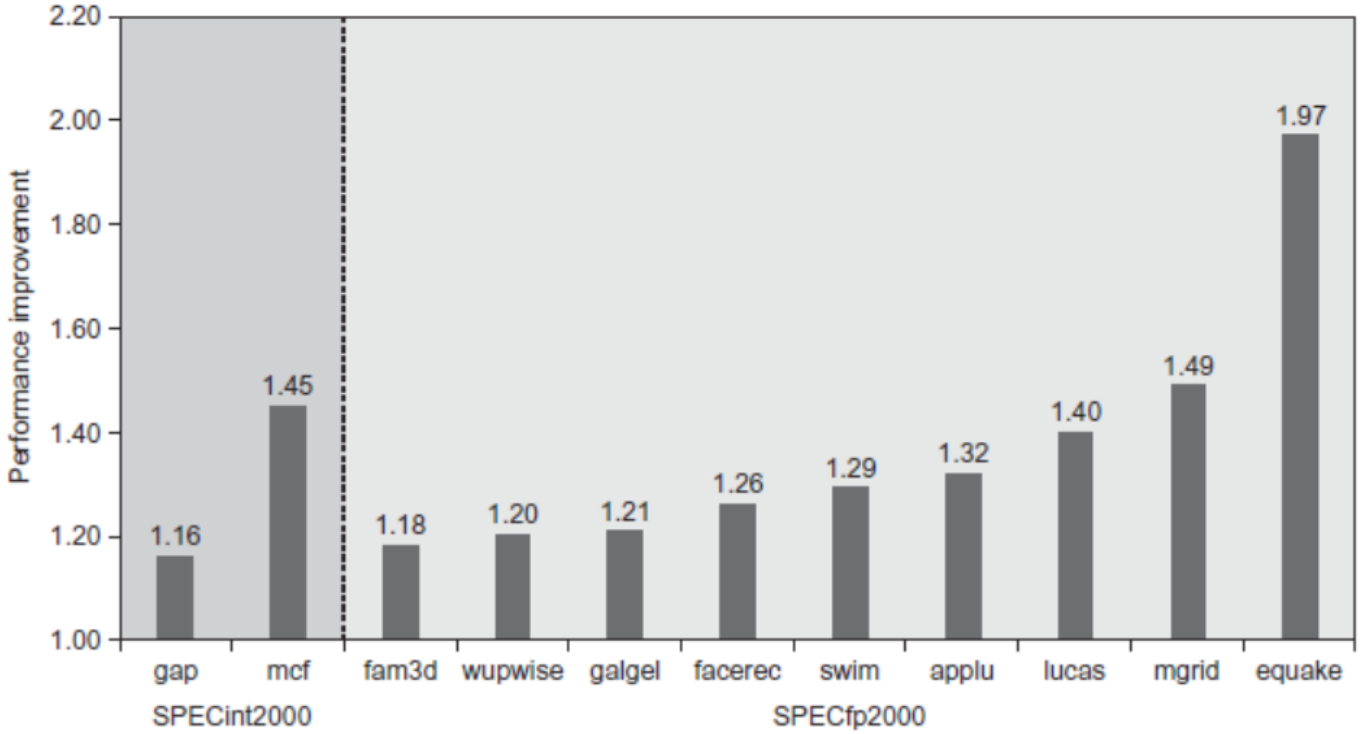
#7: Compiler Optimizations (cont'd)

- 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



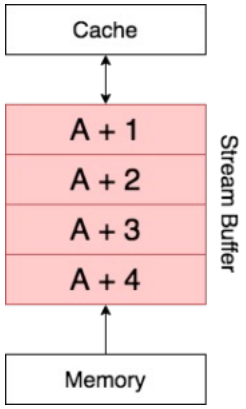
#8: Hardware Prefetching[硬件预取]

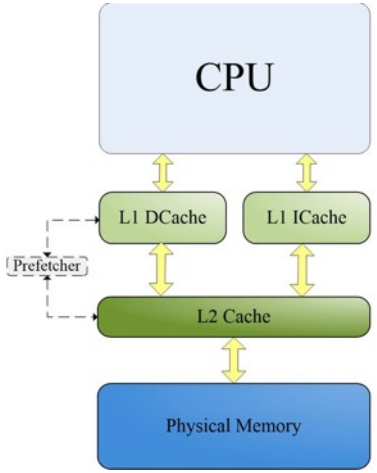
- To reduce miss penalty or miss rate
- Prefetch items before the processor requests them
  - Instruction: fetches two blocks on miss, the requested and the next consecutive
  - Data: prefetch predicted blocks



#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

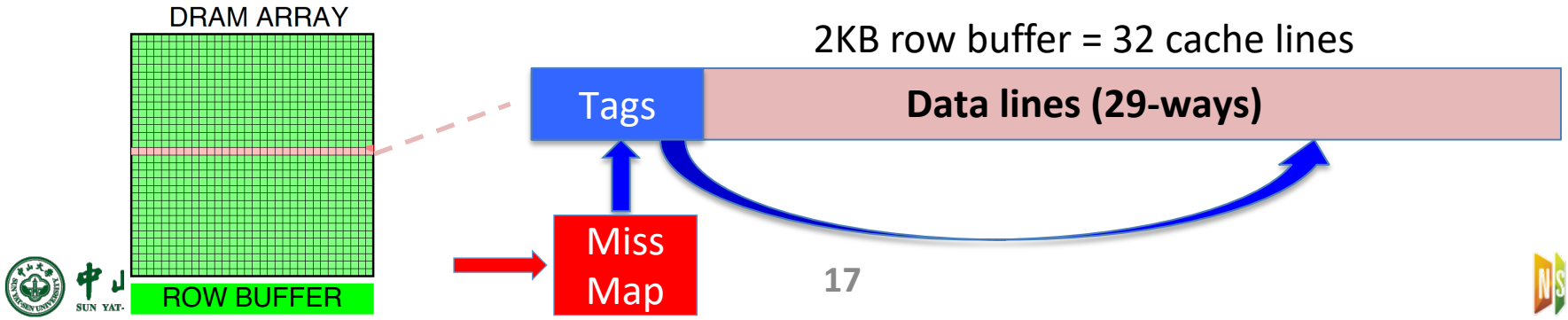
#9: Compiler-controlled Prefetching

- 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

中山大學
SUN YAT-SEN UNIVERSITY

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

Computer Architecture

计 算 机 体 系 结 构

第15讲：TLP（1）

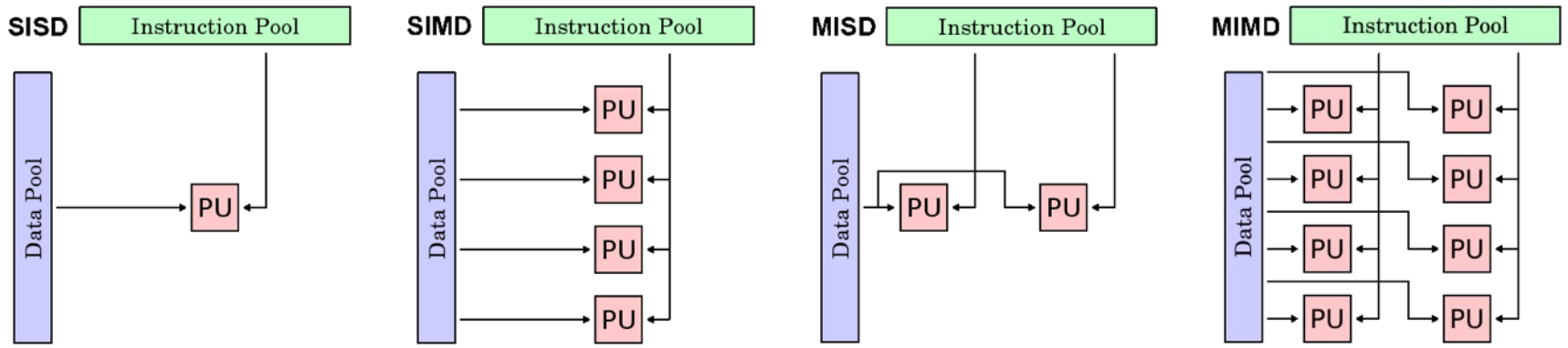
张献伟

xianweiz.github.io

DCS3013, 11/23/2022

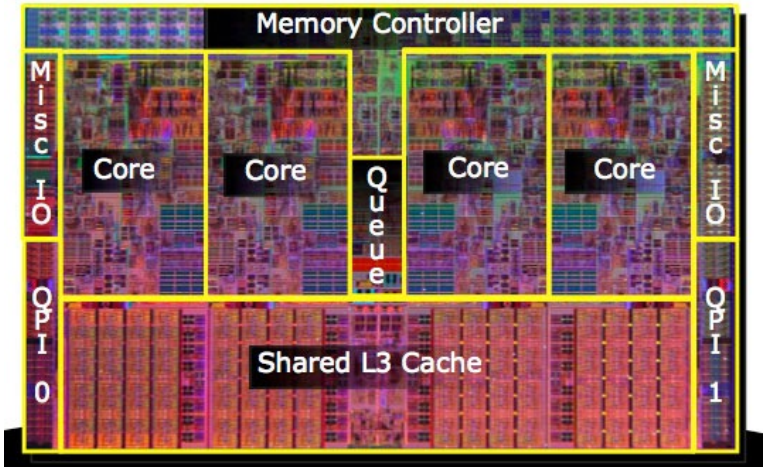
Flynn's Taxonomy[分类]

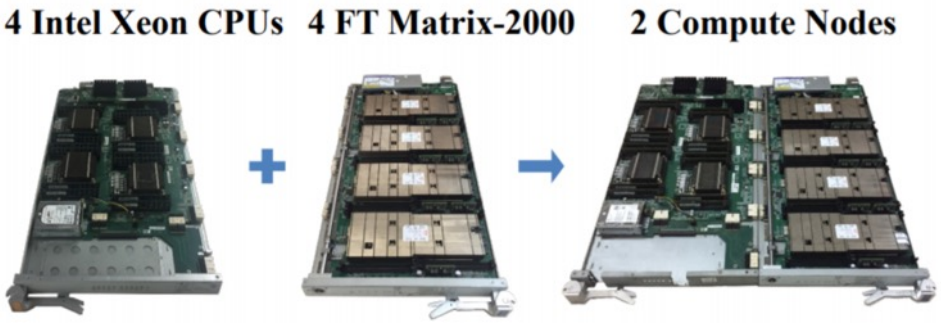
- SISD: single instruction, single data
  - A serial (non-parallel) computer
- **SIMD**: single instruction, multiple data
  - Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing
- MISD: multiple instruction, single data
  - Few (if any) 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

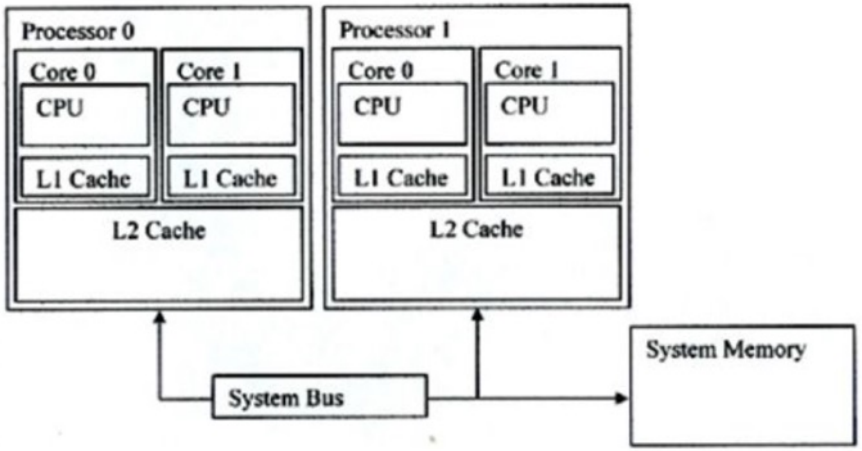




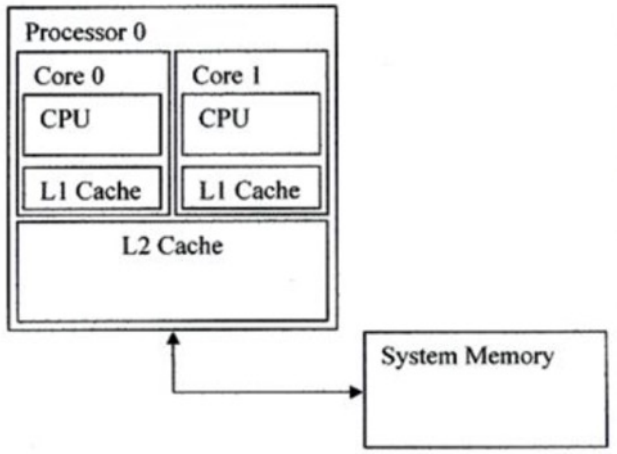
https://www.slideshare.net/abshinde/multiprocessor-74969041

Multiprocessor[多处理器]

- **Multi-processor**
  - **Multiple CPUs** tightly coupled to cooperate on a problem
  - Each CPU may be a multicore design
- **Multicore processor**
  - Multiprocessor where the CPU cores co-exist on a single processor chip (i.e., **single CPU** w/ multi cores)



Multi-Processor System with Cores that share L2 Cache



Multi-Core Processor with Shared L2 Cache

Why Multiprocessor?[使用的几个原因]

- Not that long ago, multiprocessors were expensive, exotic machines
- Reason **#1**: running out of ILP that we can exploit[ILP有限]
  - Can't get much better performance out of a single core that's running a single program at a time
- Reason **#2**: power/thermal constraints[能耗/散热限制]
  - Even if we wanted to just build fancier single cores at higher clock speeds, we'd run into power and thermal obstacles
- Reason **#3**: Moore's Law[摩尔定律]
  - Lots of transistors → what else are we going to do with them?
  - Historically: use transistors to make more complicated cores with bigger and bigger caches
  - But we just saw that this strategy has run into problems

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

Thread-Level Parallelism[线程级并行]

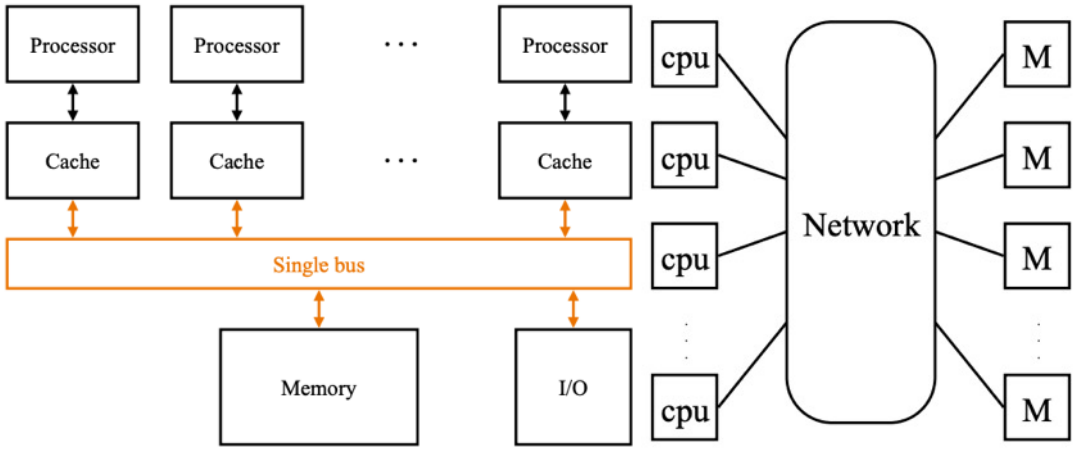
- Thread-Level parallelism[并行]
  - Have multiple program counters
  - Uses MIMD model
  - Targeted for tightly-coupled shared-memory multiprocessors
- Why TLP?[原因]
  - Hard to further increase core performance (e.g., clock speed)
  - Hard to find and exploit more ILP
- Implementation[实现]
  - **Multiprocessor**[多处理器]
    - Multicore processor[多核处理器]
    - Multi-processor[多个处理器]
  - **Multithreaded processor**[多线程处理器]

Multithreading[多线程]

- **Basic idea:** processor resources are expensive and should not be left idle
- On **uniprocessor**, multithreading occurs by *time-division multiplexing*[时分复用]
  - Processor switches between different threads
  - *Context switching* happens frequently enough user perceives threads as running at the same time
- **Multithreaded processor**: single CPU core that can execute multiple threads simultaneously
  - Switching
  - Simultaneous multithreading (SMT) → "hyperthreading" (Intel)

Classifying Multiprocessors[分类]

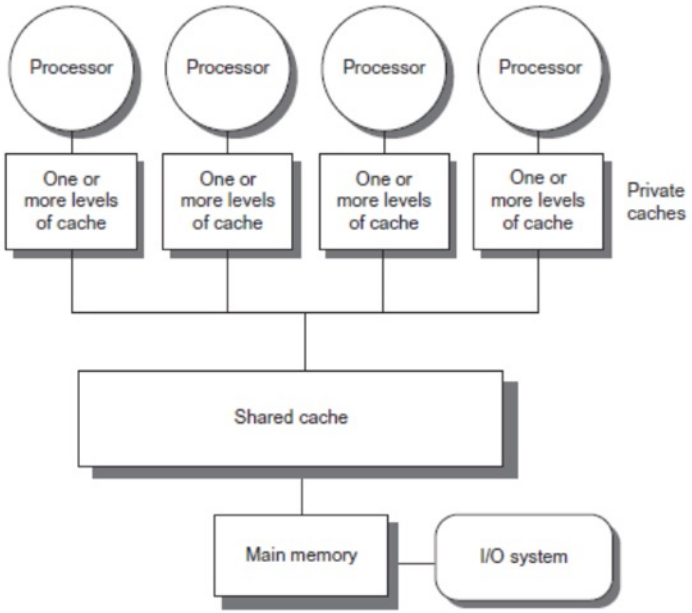
- Interconnection network[互联网络]
  - Bus
  - Network
- Memory topology[内存]
  - UMA
  - NUMA



- 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

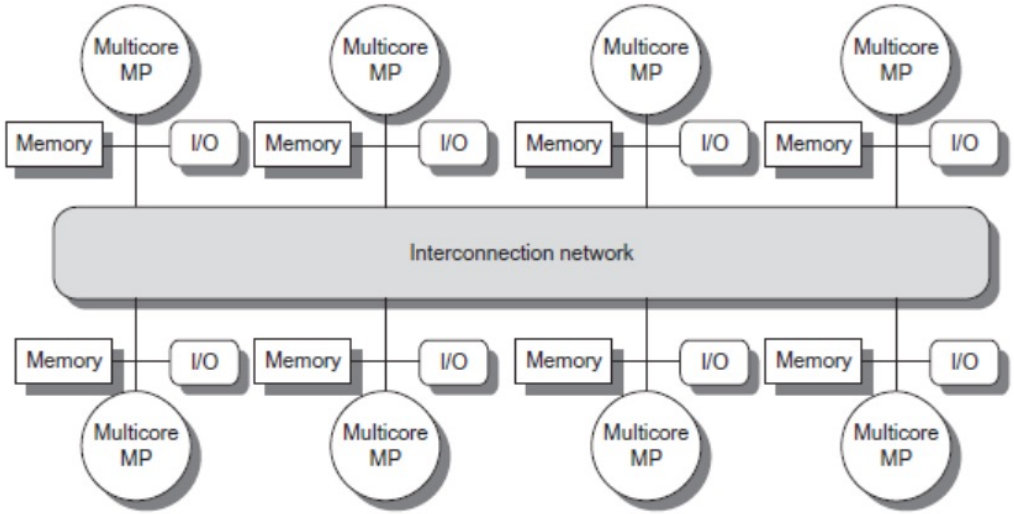
SMP[对称型]

- 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



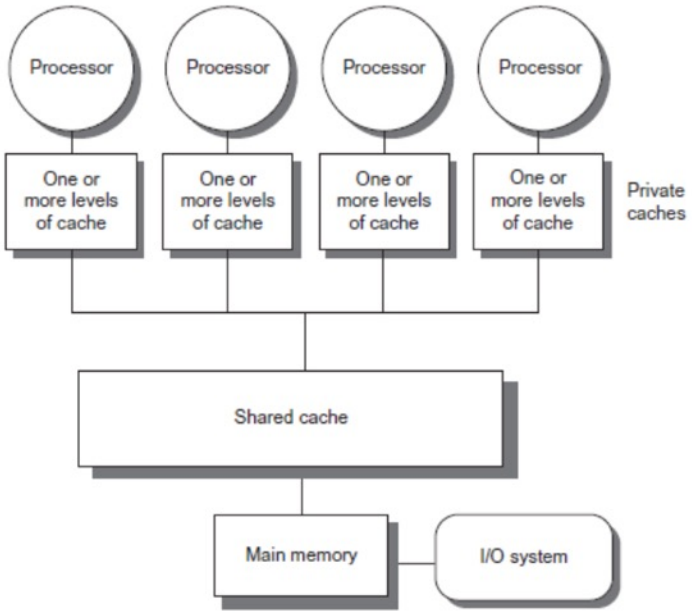
DSM[分布式共享内存]

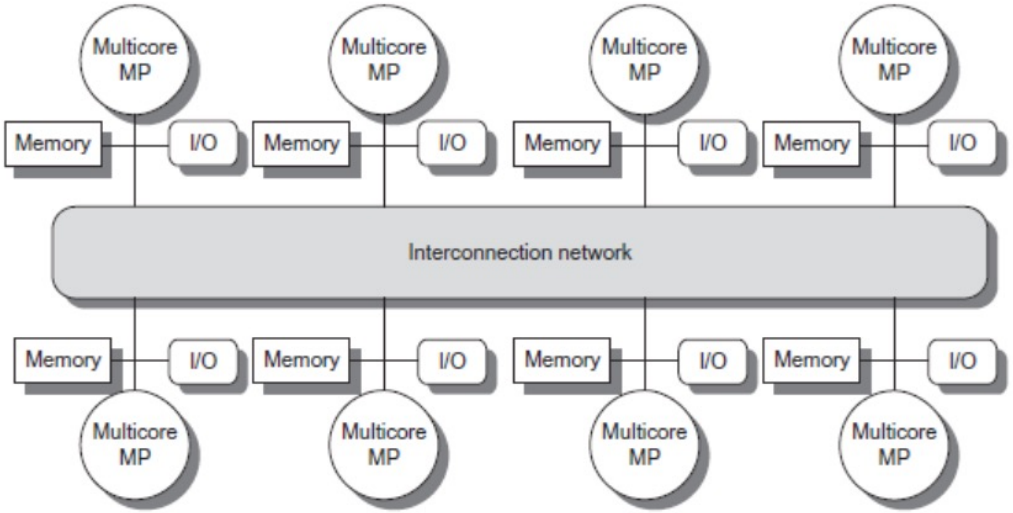
- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Non-uniform memory access/latency (NUMA)
    - The access time depends on the location of a data word in memory
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks



Shared Memory[共享内存]

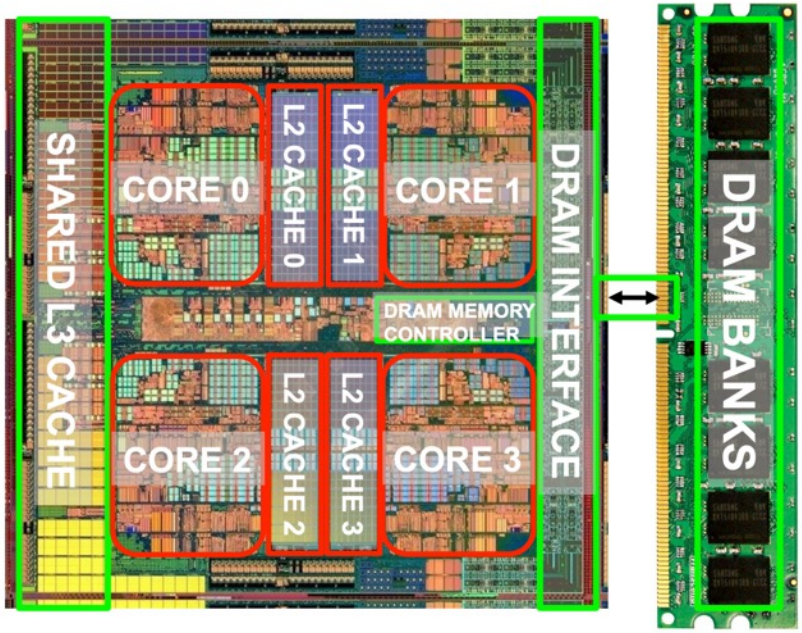
- The term "**shared memory**" associated with both SMP and DSM refers to the fact that the address space is shared
  - Communication among threads occurs through the shared address space
  - Thus, a memory reference can be made by any processor to any memory location

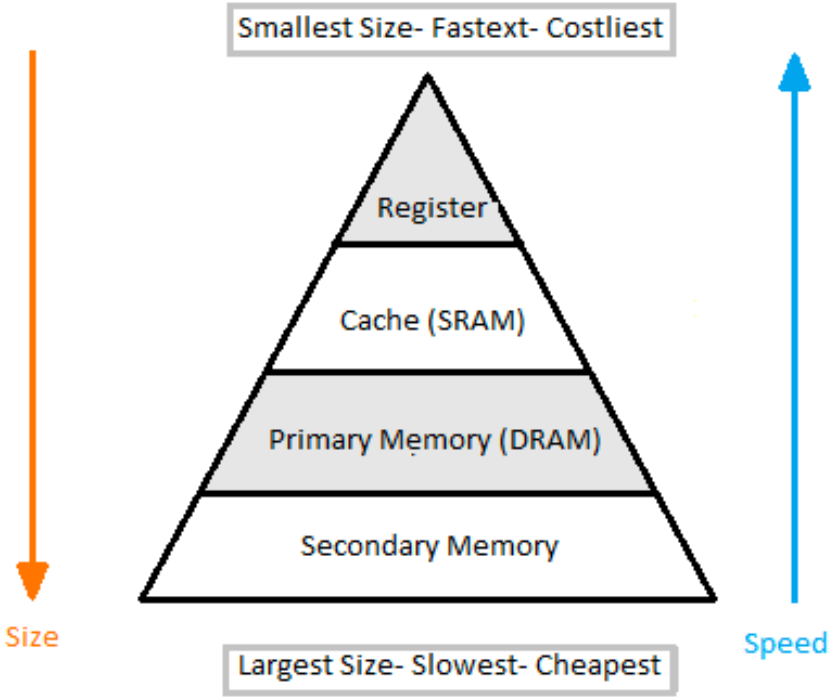




There Exist Caches

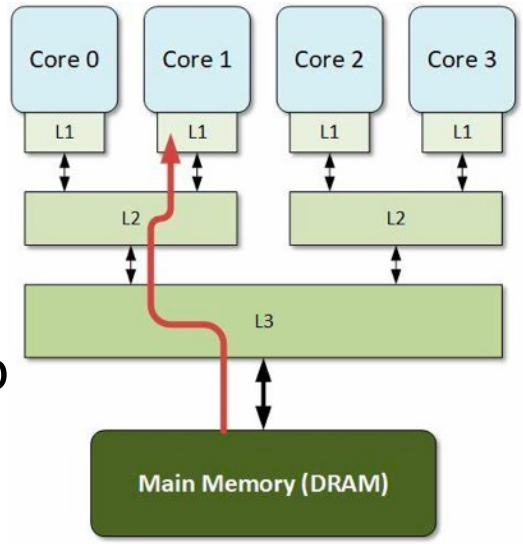
- Recall memory hierarchy, with cache being provided to shorten access latency
  - Each core of multiprocessors has a cache (or multiple caches)
- Caching complicates the data sharing





Data Caching[数据缓存]

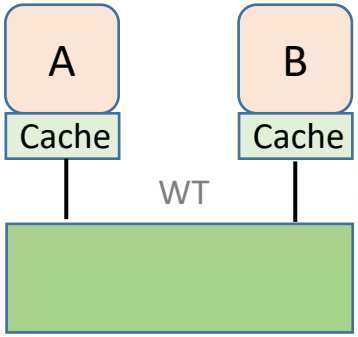
- **Private** data: used by a single processor
- **Shared** data: used by multiple processors
  - Essentially providing communication among the processors through reads and writes of the shared data



- 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