Lecture 23: Cache, Memory, Virtual Memory

- Today's topics:
  - Cache examples, caching policies
  - Main memory system
  - Virtual memory

Example 1

- 32 KB 4-way set-associative data cache array with 32 byte line sizes
- How many sets?
- How many index bits, offset bits, tag bits?
- How large is the tag array?

Example 1

- 32 KB 4-way set-associative data cache array with 32 byte line sizes

  cache size = #sets x #ways x block size

- How many sets? 256

- How many index bits, offset bits, tag bits?

  8 5 19

- How large is the tag array?

  tag array size = #sets x #ways x tag size

  = 19 Kb = 2.375 KB

Example 2

- A pipeline has CPI 1 if all loads/stores are L1 cache hits
  40% of all instructions are loads/stores
  85% of all loads/stores hit in 1-cycle L1
  50% of all (10-cycle) L2 accesses are misses
  Memory access takes 100 cycles
  What is the CPI?

Example 2

- A pipeline has CPI 1 if all loads/stores are L1 cache hits
  40% of all instructions are loads/stores
  85% of all loads/stores hit in 1-cycle L1
  50% of all (10-cycle) L2 accesses are misses
  Memory access takes 100 cycles
  What is the CPI?

Start with 1000 instructions
1000 cycles (includes all 400 L1 accesses)
+ 400 (l/s) x 15% x 10 cycles (the L2 accesses)
+ 400 x 15% x 50% x 100 cycles (the mem accesses)
= 4,600 cycles
CPI = 4.6

Cache Misses

- On a write miss, you may either choose to bring the block into the cache (write-allocate) or not (write-no-allocate)

- On a read miss, you always bring the block in (spatial and temporal locality) – but which block do you replace?
  - no choice for a direct-mapped cache
  - randomly pick one of the ways to replace
  - replace the way that was least-recently used (LRU)
  - FIFO replacement (round-robin)

Writes

- When you write into a block, do you also update the copy in L2?
  - write-through: every write to L1 → write to L2
  - write-back: mark the block as dirty, when the block gets replaced from L1, write it to L2

- Writeback coalesces multiple writes to an L1 block into one L2 write

- Writethrough simplifies coherency protocols in a multiprocessor system as the L2 always has a current copy of data

Types of Cache Misses

- Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache

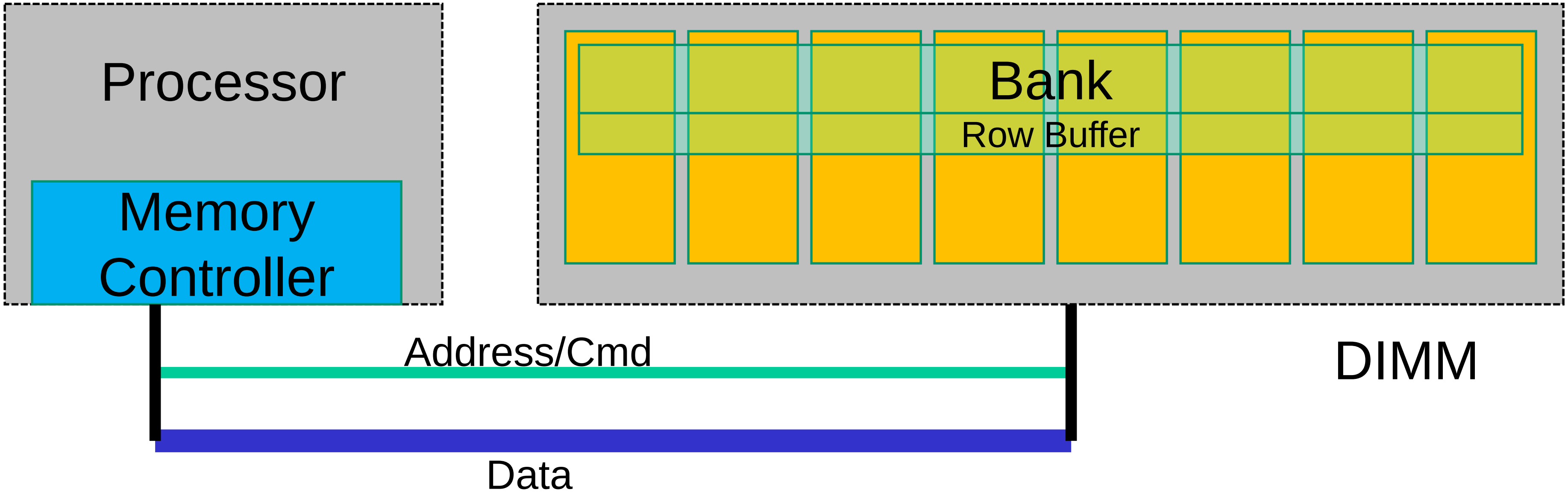
- Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache

- Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache

Off-Chip DRAM Main Memory

- Main memory is stored in DRAM cells that have much higher storage density
- DRAM cells lose their state over time – must be refreshed periodically, hence the name *Dynamic*
- A number of DRAM chips are aggregated on a DIMM to provide high capacity – a DIMM is a module that plugs into a bus on the motherboard
- DRAM access suffers from long access time and high energy overhead

Memory Architecture



- DIMM: a PCB with DRAM chips on the back and front
- The memory system is itself organized into ranks and banks; each bank can process a transaction in parallel
- Each bank has a row buffer that retains the last row touched in a bank (it's like a cache in the memory system that exploits spatial locality) (row buffer hits have a lower latency than a row buffer miss)

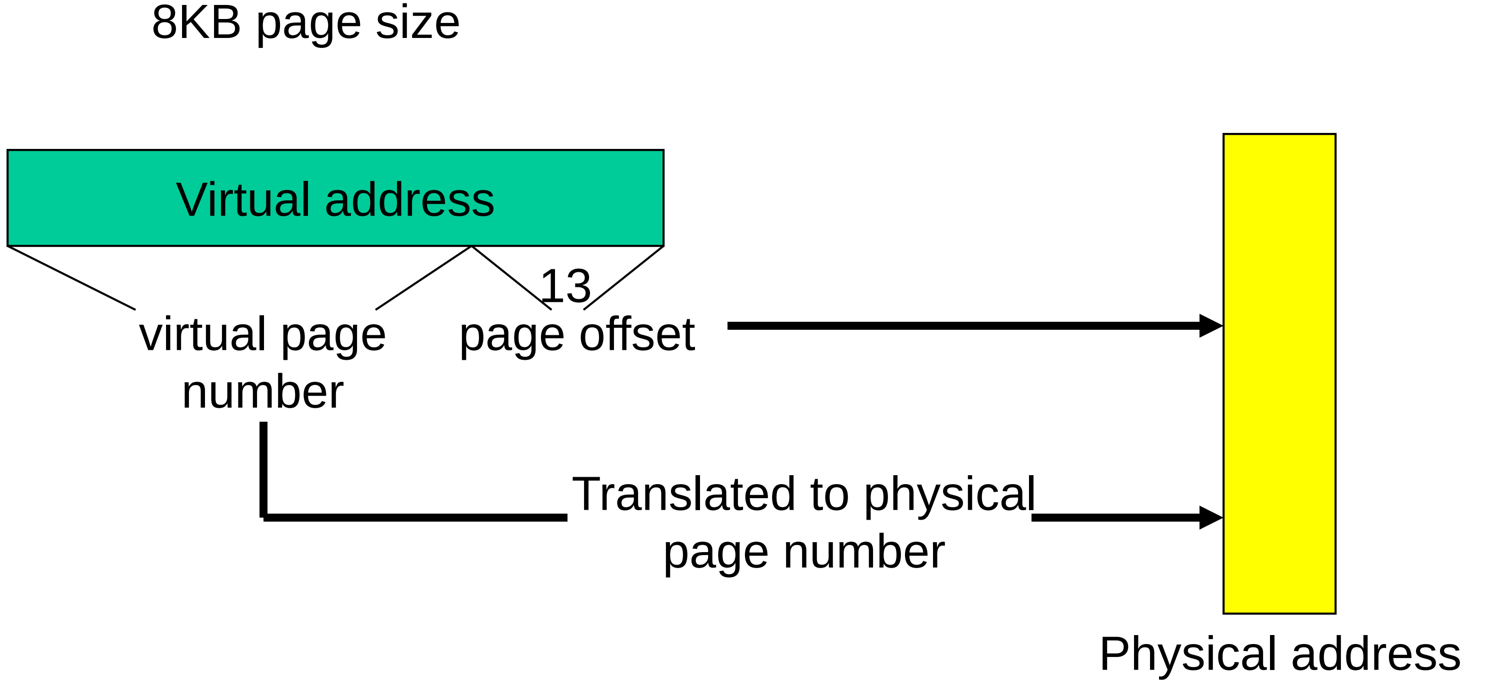
Virtual Memory

- Processes deal with virtual memory – they have the illusion that a very large address space is available to them
- There is only a limited amount of physical memory that is shared by all processes – a process places part of its virtual memory in this physical memory and the rest is stored on disk (called swap space)
- Thanks to locality, disk access is likely to be uncommon
- The hardware ensures that one process cannot access the memory of a different process

Virtual Memory

Address Translation

- The virtual and physical memory are broken up into pages



Memory Hierarchy Properties

- A virtual memory page can be placed anywhere in physical memory (fully-associative)
- Replacement is usually LRU (since the miss penalty is huge, we can invest some effort to minimize misses)
- A page table (indexed by virtual page number) is used for translating virtual to physical page number
- The page table is itself in memory

TLB

- Since the number of pages is very high, the page table capacity is too large to fit on chip

- A translation lookaside buffer (TLB) caches the virtual to physical page number translation for recent accesses

- A TLB miss requires us to access the page table, which may not even be found in the cache – two expensive memory look-ups to access one word of data!

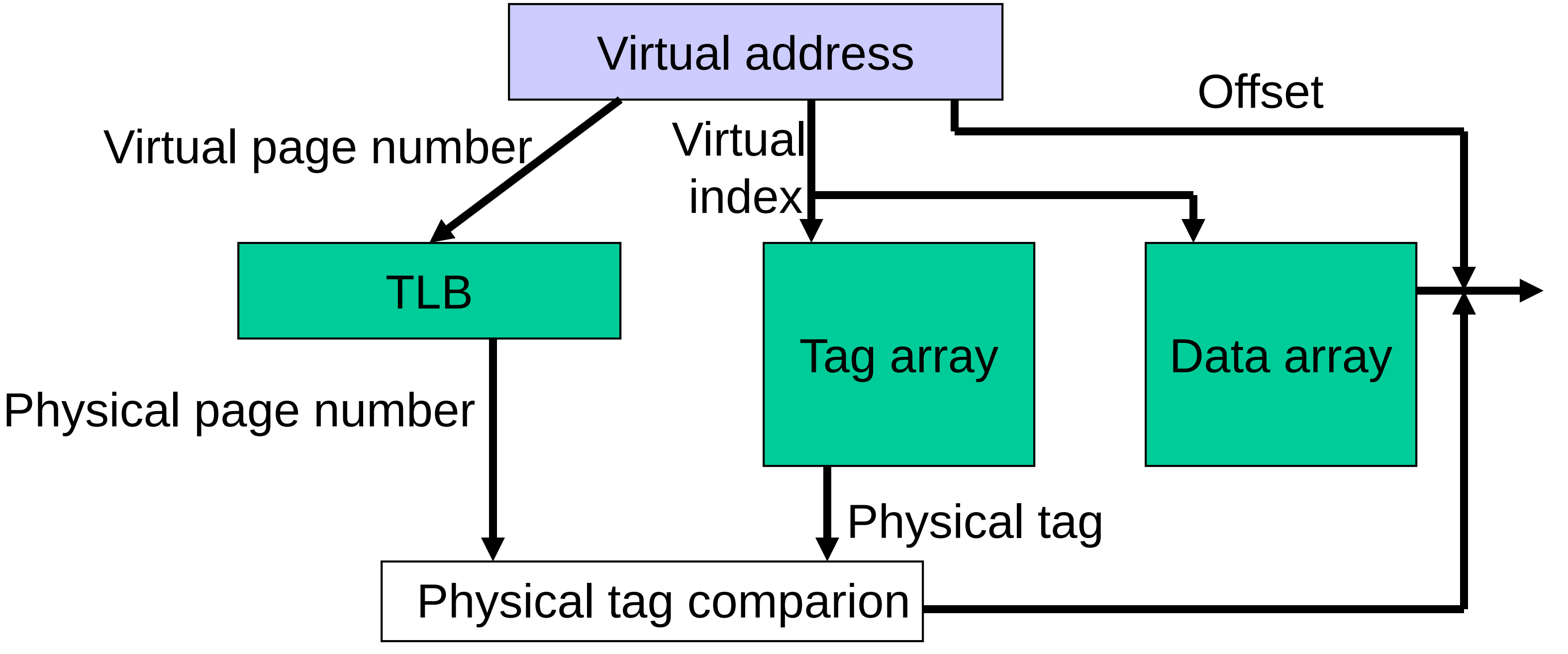
- A large page size can increase the coverage of the TLB and reduce the capacity of the page table, but also increases memory waste

TLB and Cache

- Is the cache indexed with virtual or physical address?
  - To index with a physical address, we will have to first look up the TLB, then the cache → longer access time
  - Multiple virtual addresses can map to the same physical address – must ensure that these different virtual addresses will map to the same location in cache – else, there will be two different copies of the same physical memory word

- Does the tag array store virtual or physical addresses?
  - Since multiple virtual addresses can map to the same physical address, a virtual tag comparison can flag a miss even if the correct physical memory word is present

Cache and TLB Pipeline



Virtually Indexed; Physically Tagged Cache

Bad Events

- Consider the longest latency possible for a load instruction:
  - TLB miss: must look up page table to find translation for v.page P
  - Calculate the virtual memory address for the page table entry that has the translation for page P – let's say, this is v.page Q
  - TLB miss for v.page Q: will require navigation of a hierarchical page table (let's ignore this case for now and assume we have succeeded in finding the physical memory location (R) for page Q)
  - Access memory location R (find this either in L1, L2, or memory)
  - We now have the translation for v.page P – put this into the TLB
  - We now have a TLB hit and know the physical page number – this allows us to do tag comparison and check the L1 cache for a hit
  - If there's a miss in L1, check L2 – if that misses, check in memory
  - At any point, if the page table entry claims that the page is on disk, flag a page fault – the OS then copies the page from disk to memory and the hardware resumes what it was doing before the page fault … phew!

Title

- Bullet