

Lecture: Large Caches, Virtual Memory

- Topics: cache innovations, large caches, virtual memory intro

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
- Sidenote: can a fully-associative cache have more misses than a direct-mapped cache of the same size?

More Cache Basics

- L1 caches are split as instruction and data; L2 and L3 are unified
- The L1/L2 hierarchy can be inclusive, exclusive, or non-inclusive
- On a write, you can do write-allocate or write-no-allocate
- On a write, you can do writeback or write-through; write-back reduces traffic, write-through simplifies coherence
- Reads get higher priority; writes are usually buffered
- L1 does parallel tag/data access; L2/L3 does serial tag/data

Tolerating Miss Penalty

- Out of order execution: can do other useful work while waiting for the miss – can have multiple cache misses
 - cache controller has to keep track of multiple outstanding misses (non-blocking cache)
- Hardware and software prefetching into prefetch buffers
 - aggressive prefetching can increase contention for buses

Reducing Miss Rate

- Large block size – reduces compulsory misses, reduces miss penalty in case of spatial locality – increases traffic between different levels, space waste, and conflict misses
- Large cache – reduces capacity/conflict misses – access time penalty
- High associativity – reduces conflict misses – rule of thumb: 2-way cache of capacity $N/2$ has the same miss rate as 1-way cache of capacity N – more energy

Techniques to Reduce Cache Misses

- Victim caches
- Better replacement policies – pseudo-LRU, NRU, DRRIP
-- insertion, promotion, victim selection
- Prefetching, cache compression

Victim Caches

- A direct-mapped cache suffers from misses because multiple pieces of data map to the same location
- The processor often tries to access data that it recently discarded – all discards are placed in a small victim cache (4 or 8 entries) – the victim cache is checked before going to L2
- Can be viewed as additional associativity for a few sets that tend to have the most conflicts

Replacement Policies

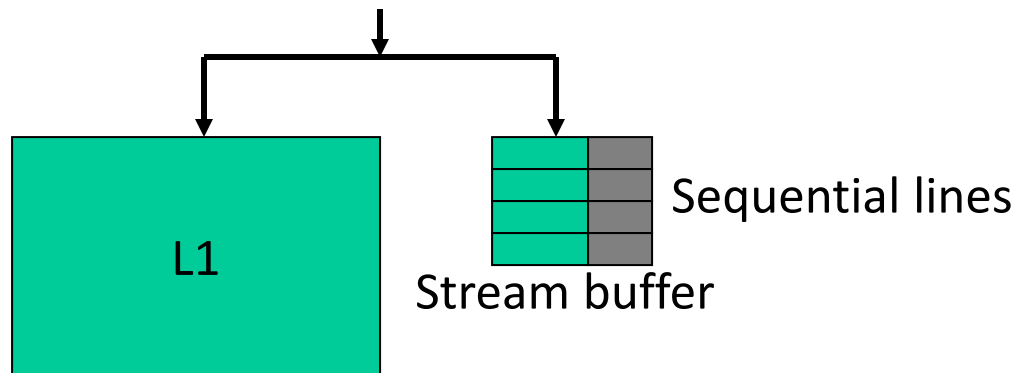
- Pseudo-LRU: maintain a tree and keep track of which side of the tree was touched more recently; simple bit ops
- NRU: every block in a set has a bit; the bit is made zero when the block is touched; if all are zero, make all one; a block with bit set to 1 is evicted

Prefetching

- Hardware prefetching can be employed for any of the cache levels
- It can introduce cache pollution – prefetched data is often placed in a separate prefetch buffer to avoid pollution – this buffer must be looked up in parallel with the cache access
- Aggressive prefetching increases “coverage”, but leads to a reduction in “accuracy” → wasted memory bandwidth
- Prefetches must be timely: they must be issued sufficiently in advance to hide the latency, but not too early (to avoid pollution and eviction before use)

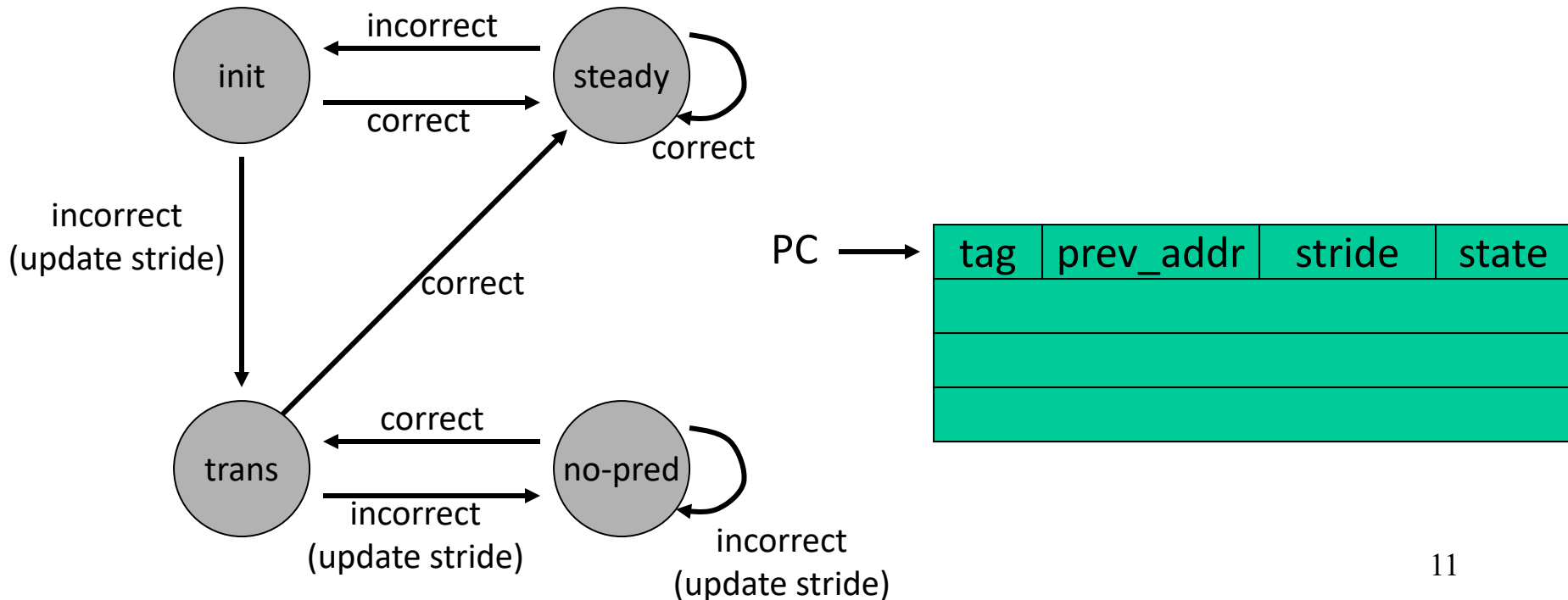
Stream Buffers

- Simplest form of prefetch: on every miss, bring in multiple cache lines
- When you read the top of the queue, bring in the next line



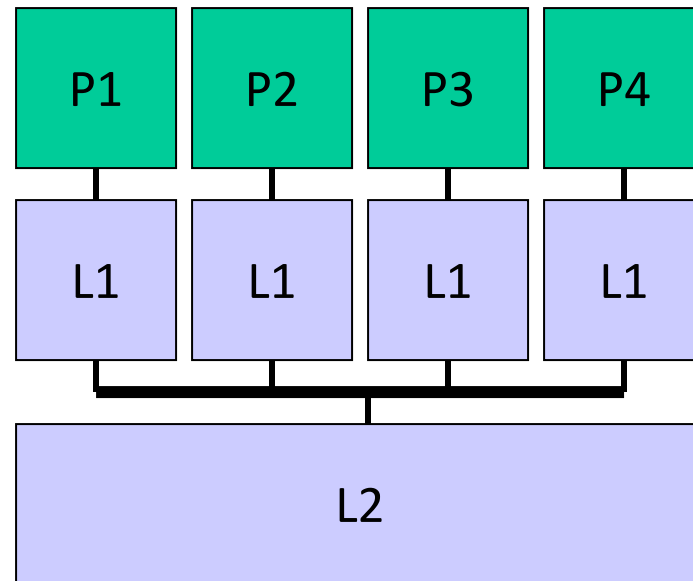
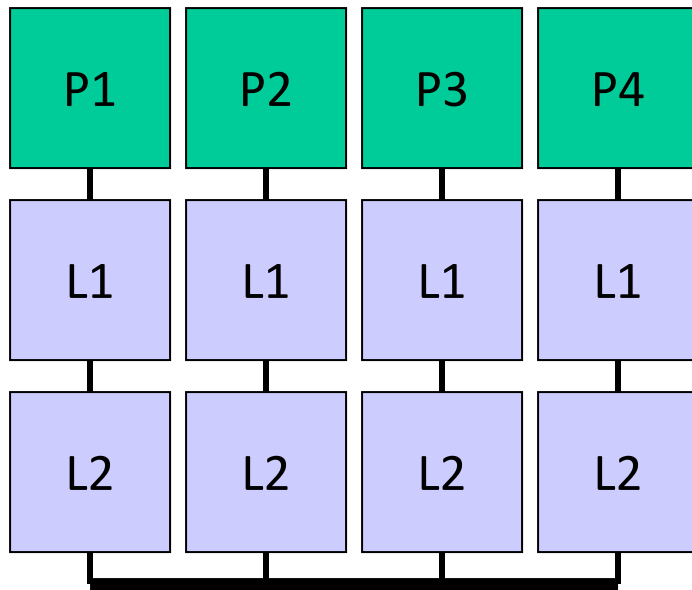
Stride-Based Prefetching

- For each load, keep track of the last address accessed by the load and a possibly consistent stride
- FSM detects consistent stride and issues prefetches



Shared Vs. Private Caches in Multi-Core

- What are the pros/cons of a shared L2 cache?



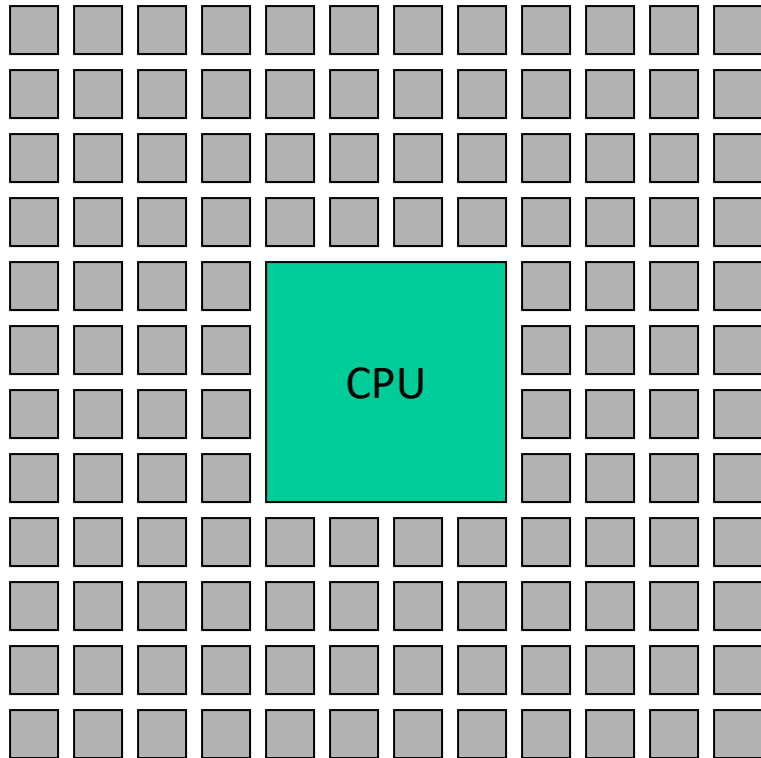
Shared Vs. Private Caches in Multi-Core

- Advantages of a shared cache:
 - Space is dynamically allocated among cores
 - No waste of space because of replication
 - Potentially faster cache coherence (and easier to locate data on a miss)
- Advantages of a private cache:
 - small L2 → faster access time
 - private bus to L2 → less contention

UCA and NUCA

- The small-sized caches so far have all been uniform cache access: the latency for any access is a constant, no matter where data is found
- For a large multi-megabyte cache, it is expensive to limit access time by the worst case delay: hence, non-uniform cache architecture

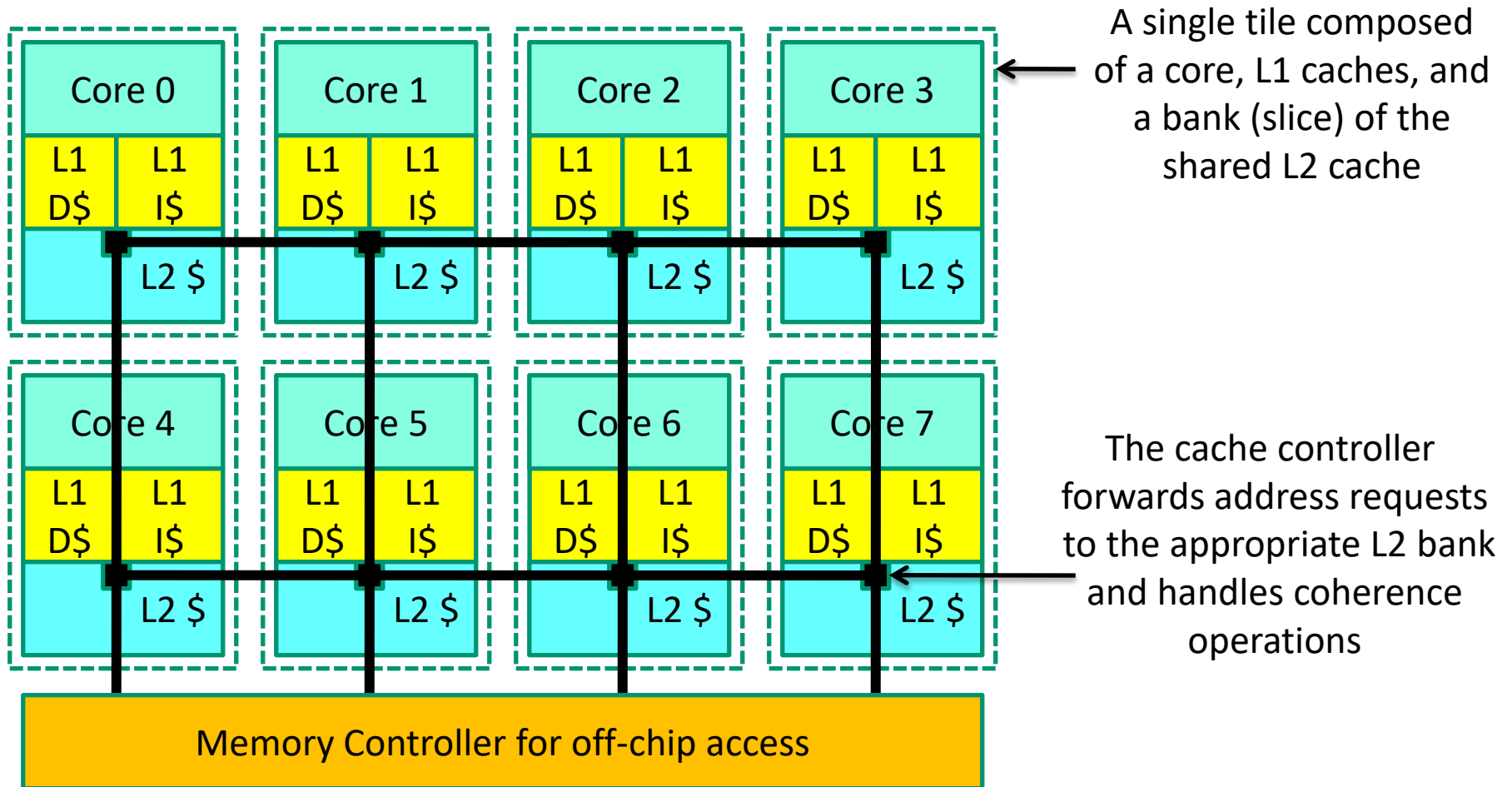
Large NUCA



Issues to be addressed for
Non-Uniform Cache Access:

- Mapping
- Migration
- Search
- Replication

Shared NUCA Cache

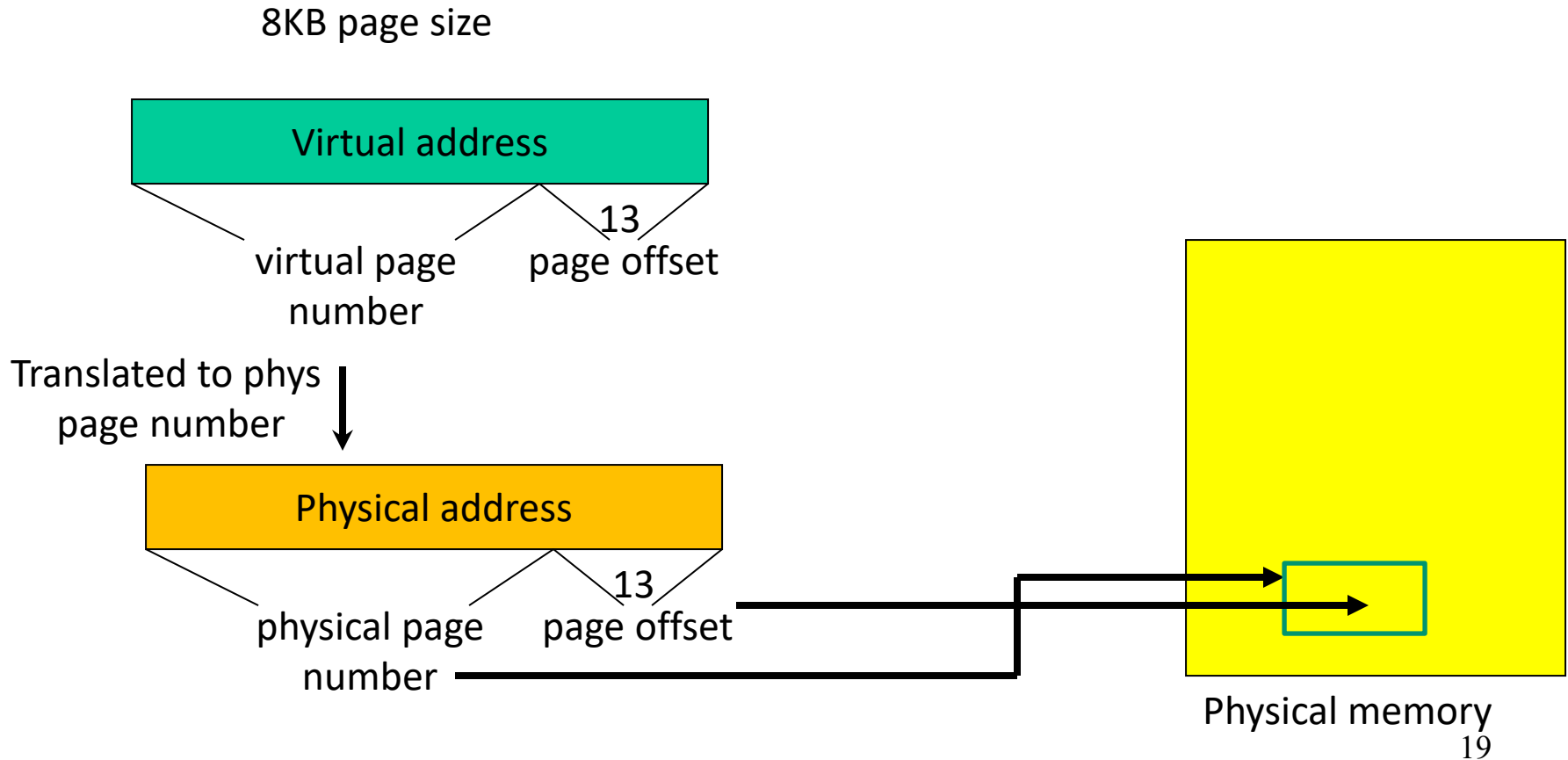


Virtual Memory

- 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
- Thanks to locality, disk access is likely to be uncommon
- The hardware ensures that one process cannot access the memory of a different process

Address Translation

- The virtual and physical memory are broken up into pages

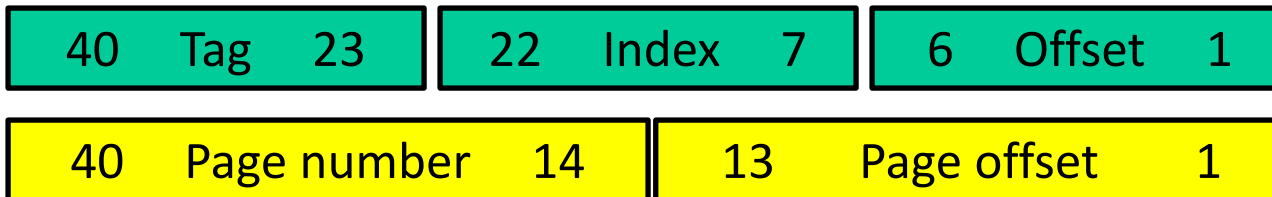


Problem 1

- Assume a large shared LLC that is tiled and distributed on the chip. Assume 16 tiles. Assume an OS page size of 8KB. The entire LLC has a size of 32 MB, uses 64-byte blocks, and is 8-way set-associative. Which of the 40 physical address bits are used to specify the tile number? Provide an example page number that is assigned to tile 0.

Problem 1

- Assume a large shared LLC that is tiled and distributed on the chip. Assume 16 tiles. Assume an OS page size of 8KB. The entire LLC has a size of 32 MB, uses 64-byte blocks, and is 8-way set-associative. Which of the 40 physical address bits are used to specify the tile number? Provide an example page number that is assigned to tile 0.



The cache has 64K sets, i.e., 6 block offset bits, 16 index bits, and 18 tag bits. The address also has a 13-bit page offset, and 27 page number bits. Nine bits (bits 14-22) are used for the page number and the index bits. Any four of those bits can be used to designate the tile number, say, bits 19-22. An example page number assigned to tile 0 is xxx...xxx0000xxx...xxx

bit 22 19

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 memory-disk hierarchy can be either inclusive or exclusive and the write policy is writeback

