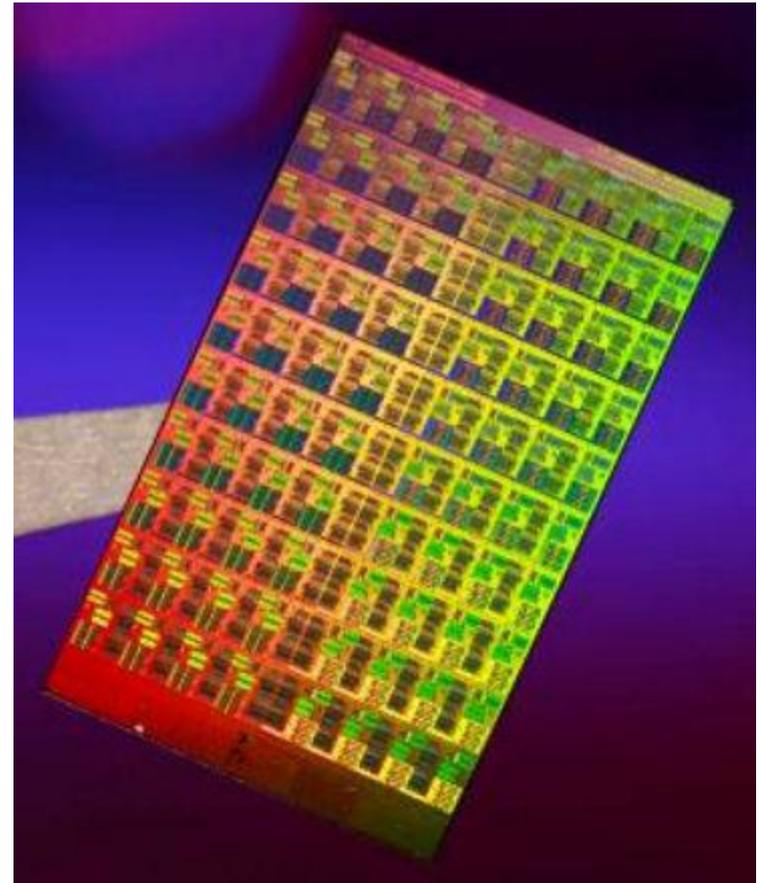


Lecture: Large Caches, Virtual Memory

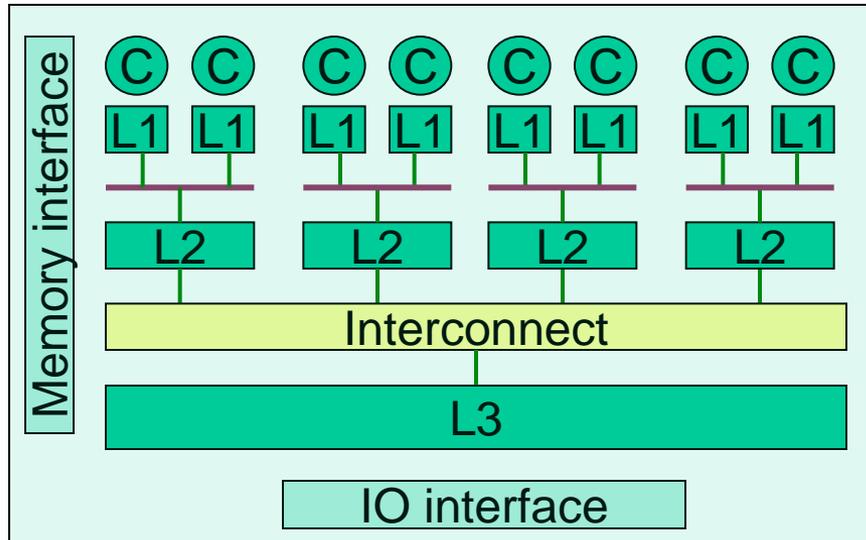
- Topics: shared/private caches, virtual memory, TLBs
(Sections 2.4, B.4, B.5)

Intel 80-Core Prototype – Polaris

Prototype chip with an entire die of SRAM cache stacked upon the cores



Example Intel Studies

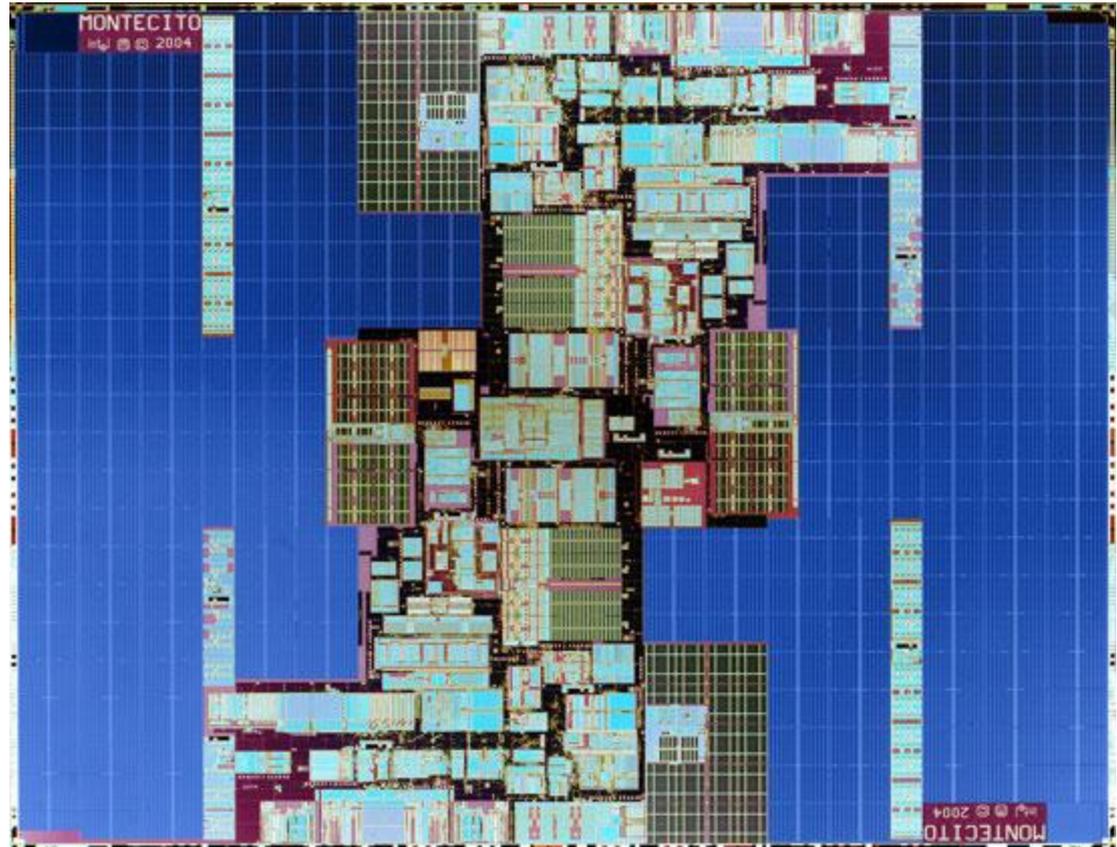


From Zhao et al.,
CMP-MSI Workshop 2007

L3 Cache sizes up to 32 MB

Intel Montecito Cache

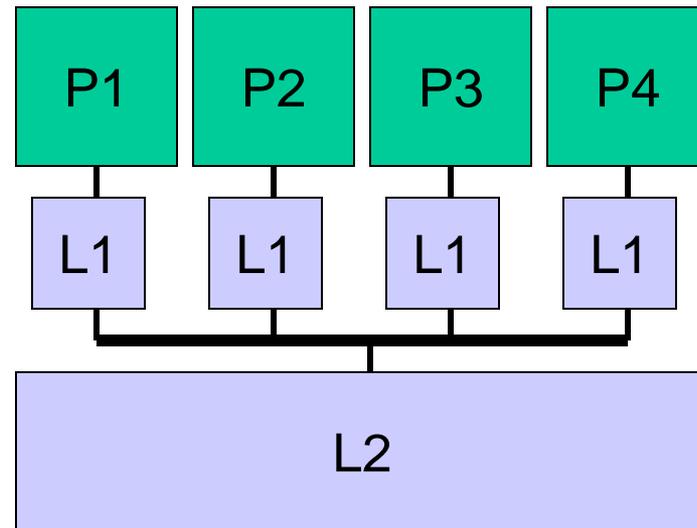
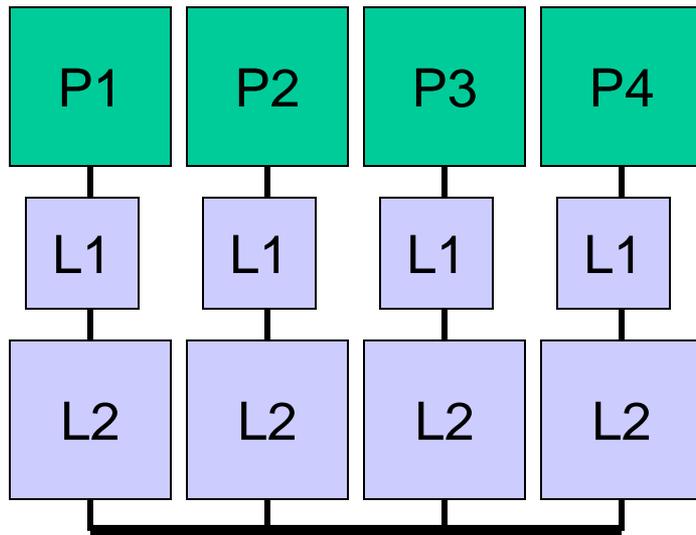
Two cores, each
with a private
12 MB L3 cache
and 1 MB L2



Naffziger et al., Journal of Solid-State Circuits, 2006

Shared Vs. Private Caches in Multi-Core

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



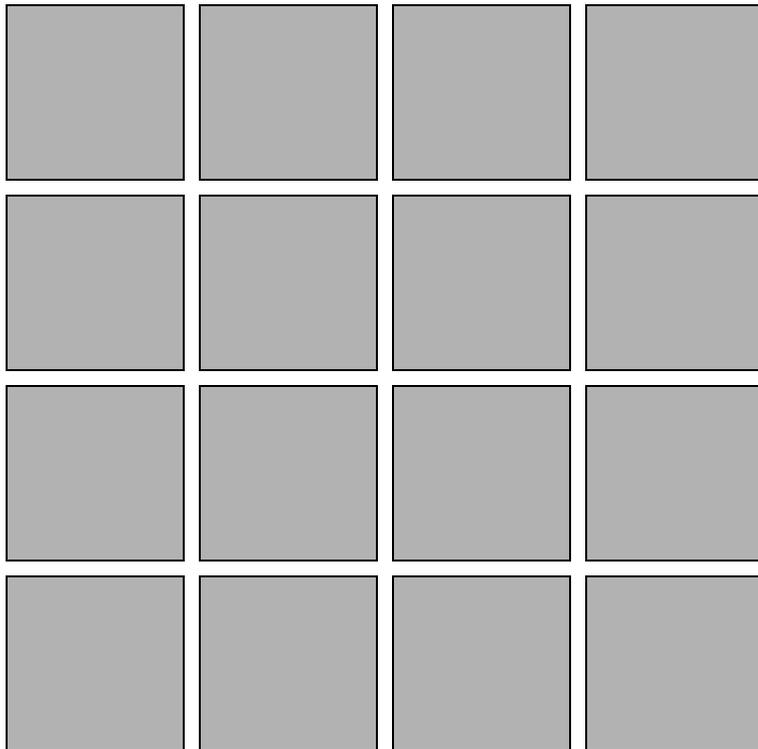
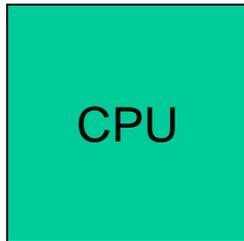
Shared Vs. Private Caches in Multi-Core

- 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

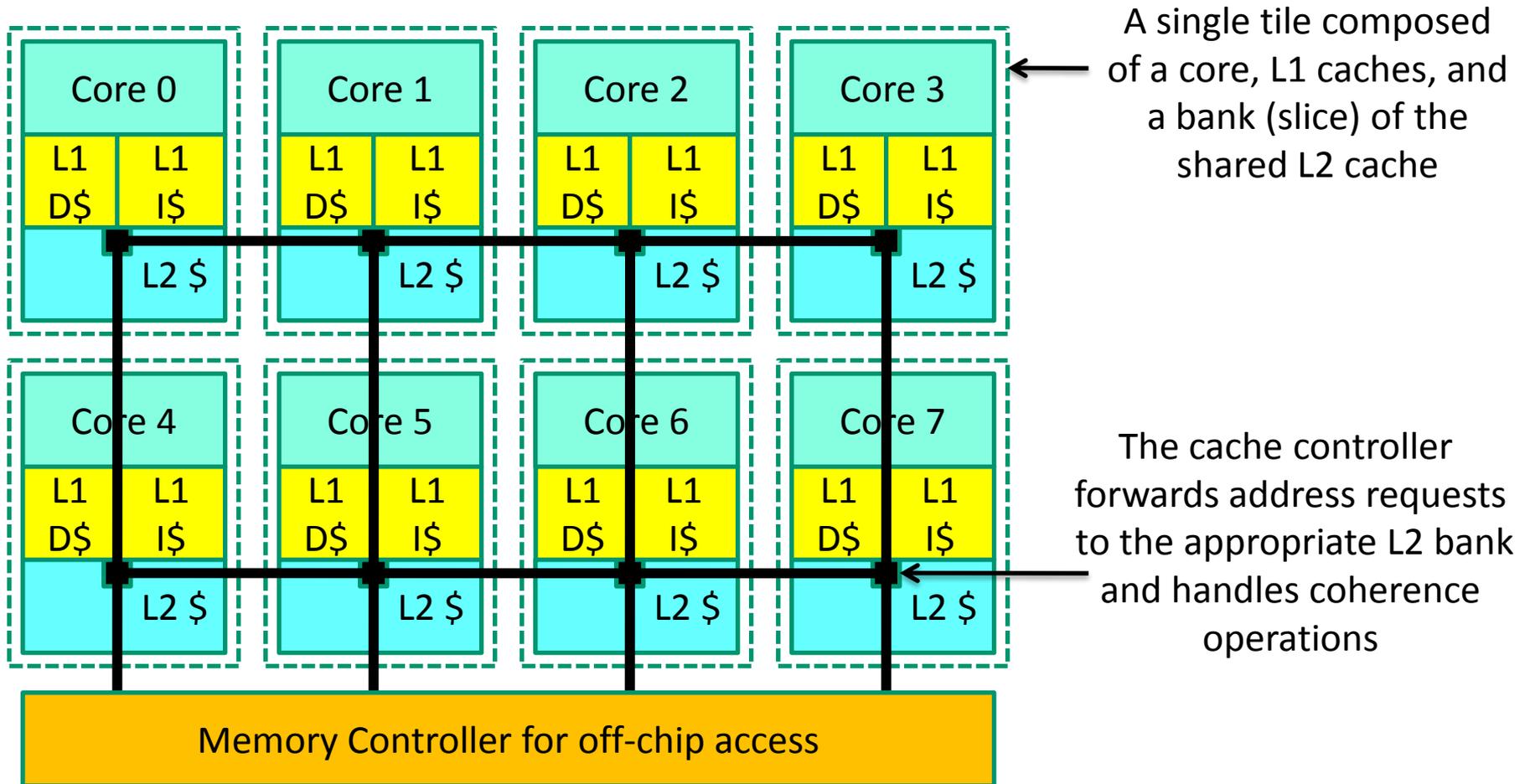
Large NUCA



Issues to be addressed for Non-Uniform Cache Access:

- Mapping
- Migration
- Search
- Replication

Shared NUCA Cache



Virtual Memory

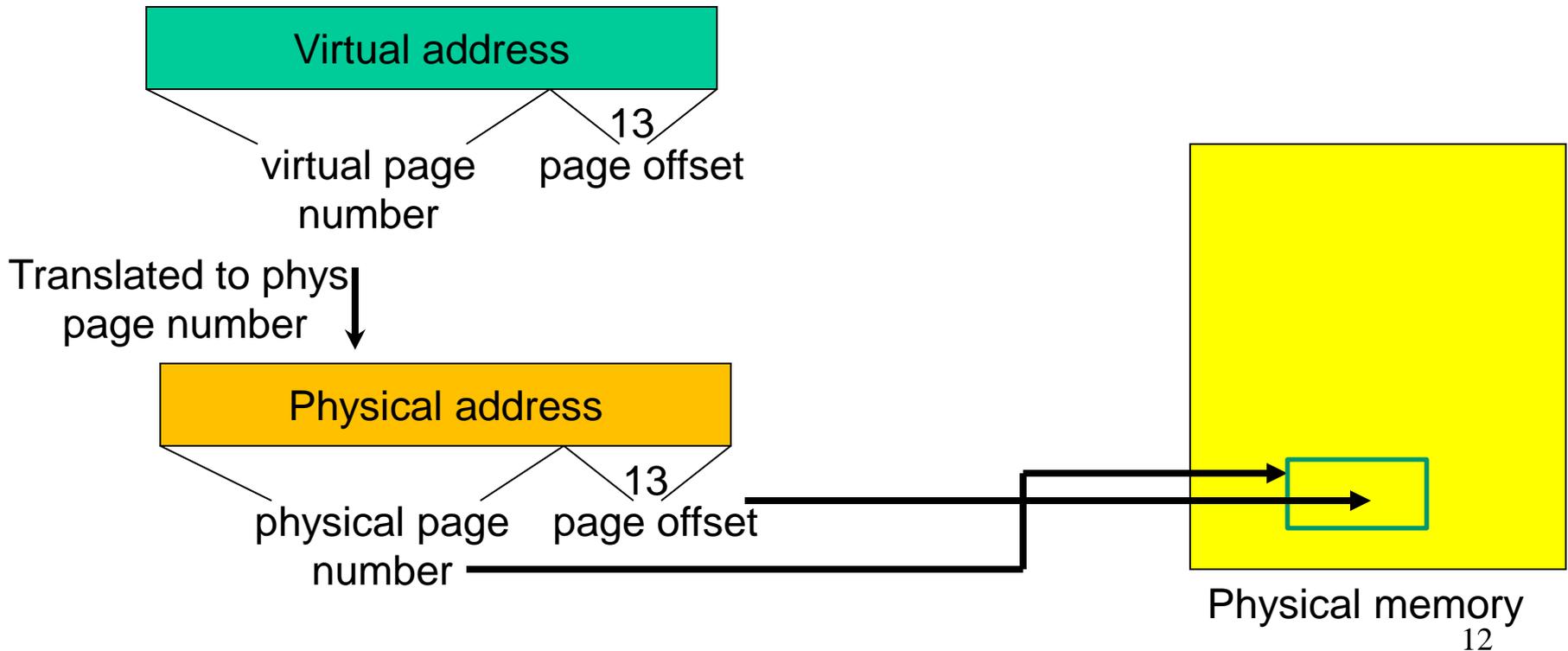
- Processes deal with virtual memory – they have the illusion that a very large address space is available to them
- There is only a limited amount of physical memory that is shared by all processes – a process places part of its virtual memory in this physical memory and the rest is stored on disk
-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

Virtual Memory and Page Tables

Address Translation

- The virtual and physical memory are broken up into pages

8KB page size



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

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

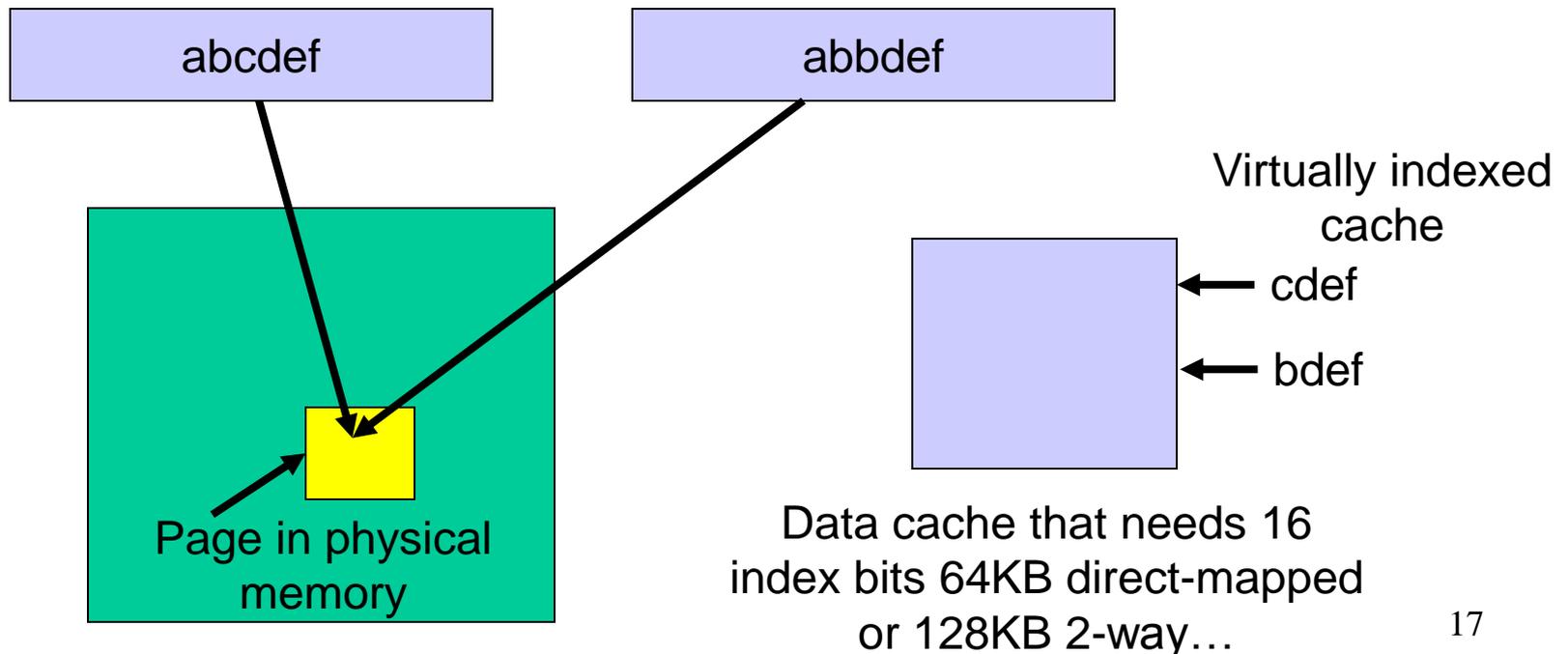
TLB and Cache

- 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 – can we 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

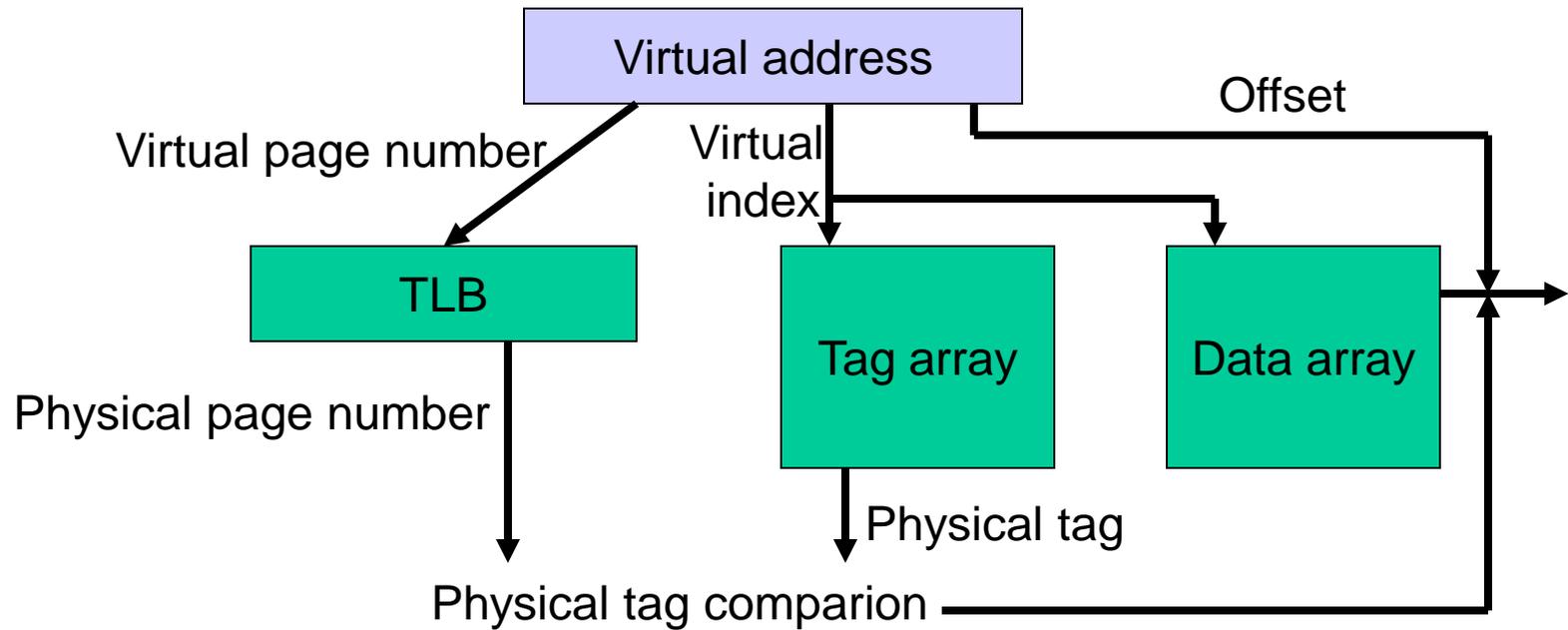
TLB and Cache

Virtually Indexed Caches

- 24-bit virtual address, 4KB page size → 12 bits offset and 12 bits virtual page number
- To handle the example below, the cache must be designed to use only 12 index bits – for example, make the 64KB cache 16-way
- Page coloring can ensure that some bits of virtual and physical address match



Cache and TLB Pipeline



Virtually Indexed; Physically Tagged Cache

Protection

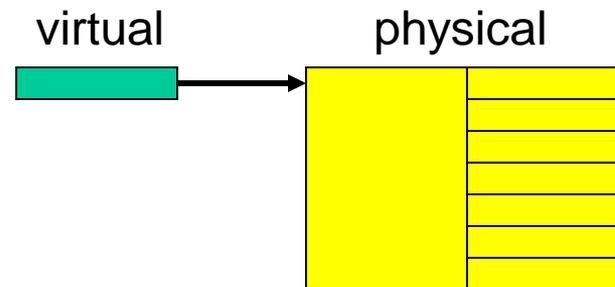
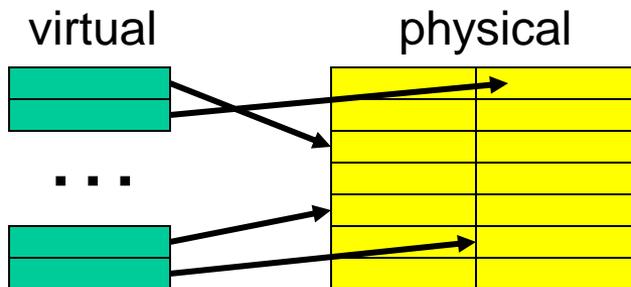
- The hardware and operating system must co-operate to ensure that different processes do not modify each other's memory
- The hardware provides special registers that can be read in user mode, but only modified by instrs in supervisor mode
- A simple solution: the physical memory is divided between processes in contiguous chunks by the OS and the bounds are stored in special registers – the hardware checks every program access to ensure it is within bounds
- Protection bits are tracked in the TLB on a per-page basis

Superpages

- If a program's working set size is 16 MB and page size is 8KB, there are 2K frequently accessed pages – a 128-entry TLB will not suffice
- By increasing page size to 128KB, TLB misses will be eliminated – disadvantage: memory waste, increase in page fault penalty
- Can we change page size at run-time?
- Note that a single page has to be contiguous in physical memory

Superpages Implementation

- At run-time, build superpages if you find that contiguous virtual pages are being accessed at the same time
- For example, virtual pages 64-79 may be frequently accessed – coalesce these pages into a single superpage of size 128KB that has a single entry in the TLB
- The physical superpage has to be in contiguous physical memory – the 16 physical pages have to be moved so they are contiguous



Ski Rental Problem

- Promoting a series of contiguous virtual pages into a superpage reduces TLB misses, but has a cost: copying physical memory into contiguous locations
- Page usage statistics can determine if pages are good candidates for superpage promotion, but if cost of a TLB miss is x and cost of copying pages is Nx , when do you decide to form a superpage?
- If ski rentals cost \$50 and new skis cost \$500, when do I decide to buy new skis?
 - If I rent 10 times and then buy skis, I'm guaranteed to not spend more than twice the optimal amount

Title

- Bullet