CS162
Operating Systems and Systems Programming
Lecture 20

Caches and TLBs

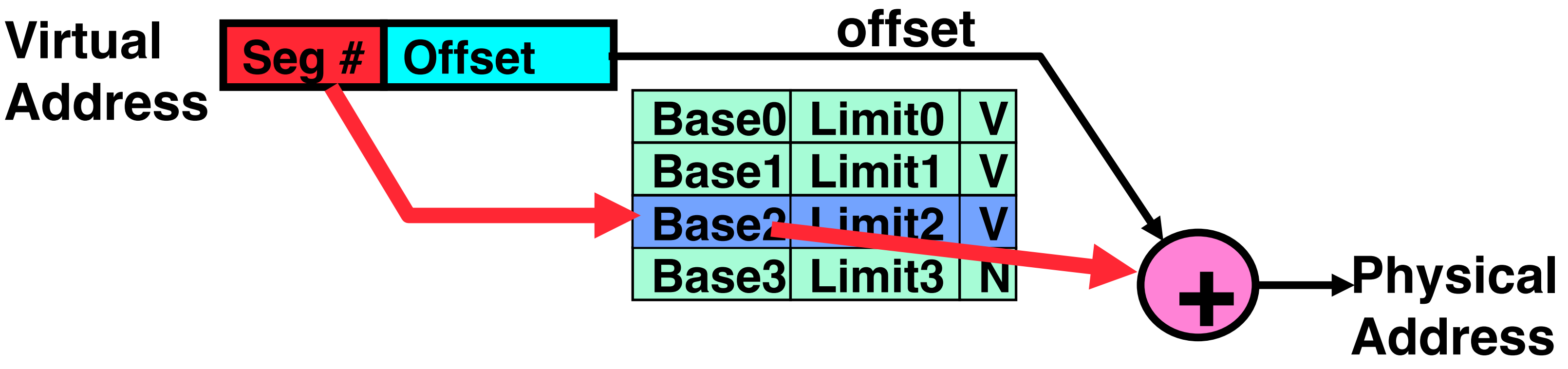
November 7, 2011

Anthony D. Joseph and Ion Stoica

http://inst.eecs.berkeley.edu/~cs162

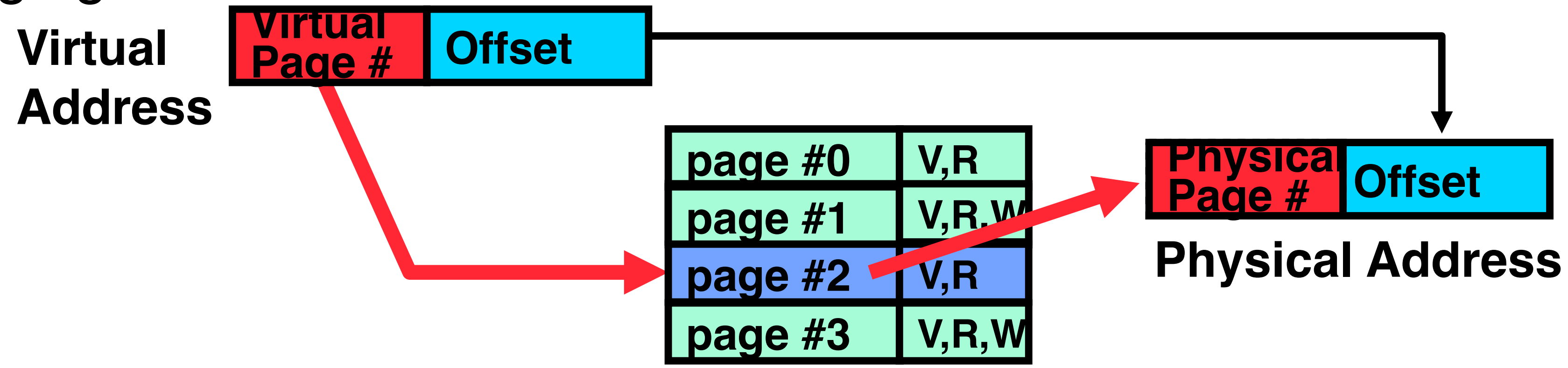
Recap: Segmentation vs. Paging

- Segmentation:



Base0	Limit0	V
Base1	Limit1	V
Base2	Limit2	V
Base3	Limit3	N

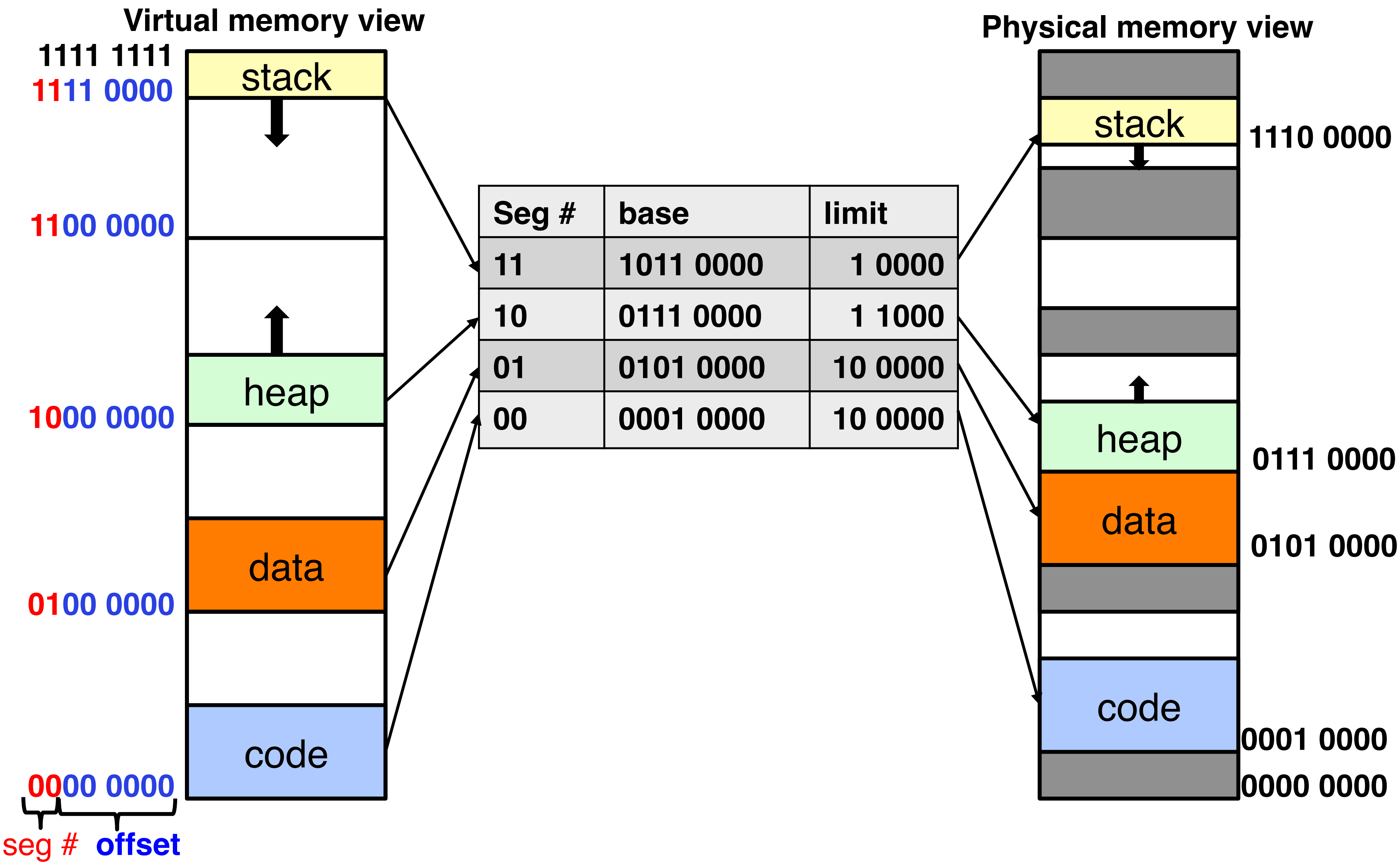
- Paging



page #0	V,R
page #1	V,R,W
page #2	V,R
page #3	V,R,W

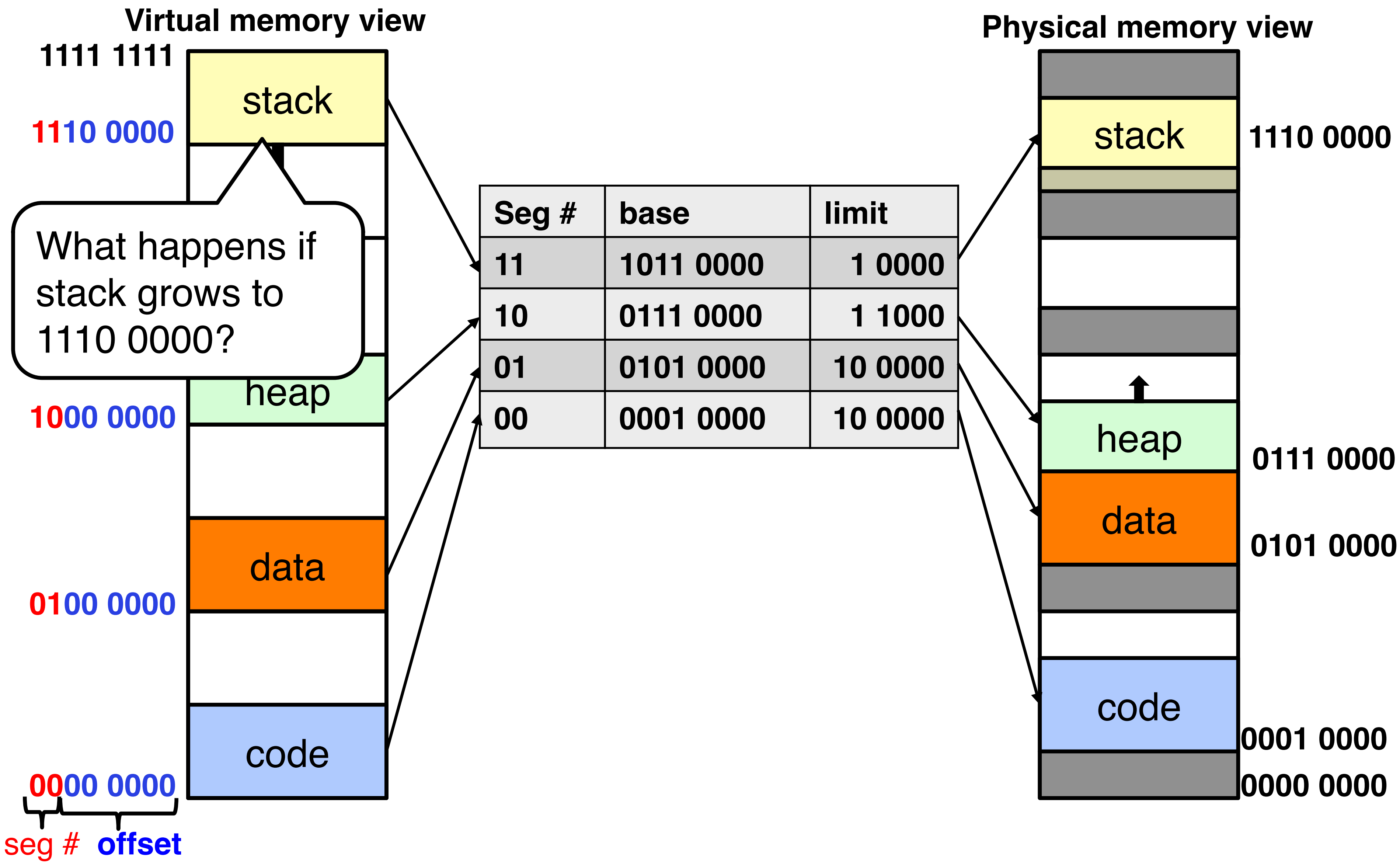
- Note: paging is ***equivalent*** to segmentation when a segment maps onto a page!
  - The offset of the first address in a page is 0

Review: Address Segmentation



Seg #	base	limit
11	1011 0000	1 0000
10	0111 0000	1 1000
01	0101 0000	10 0000
00	0001 0000	10 0000

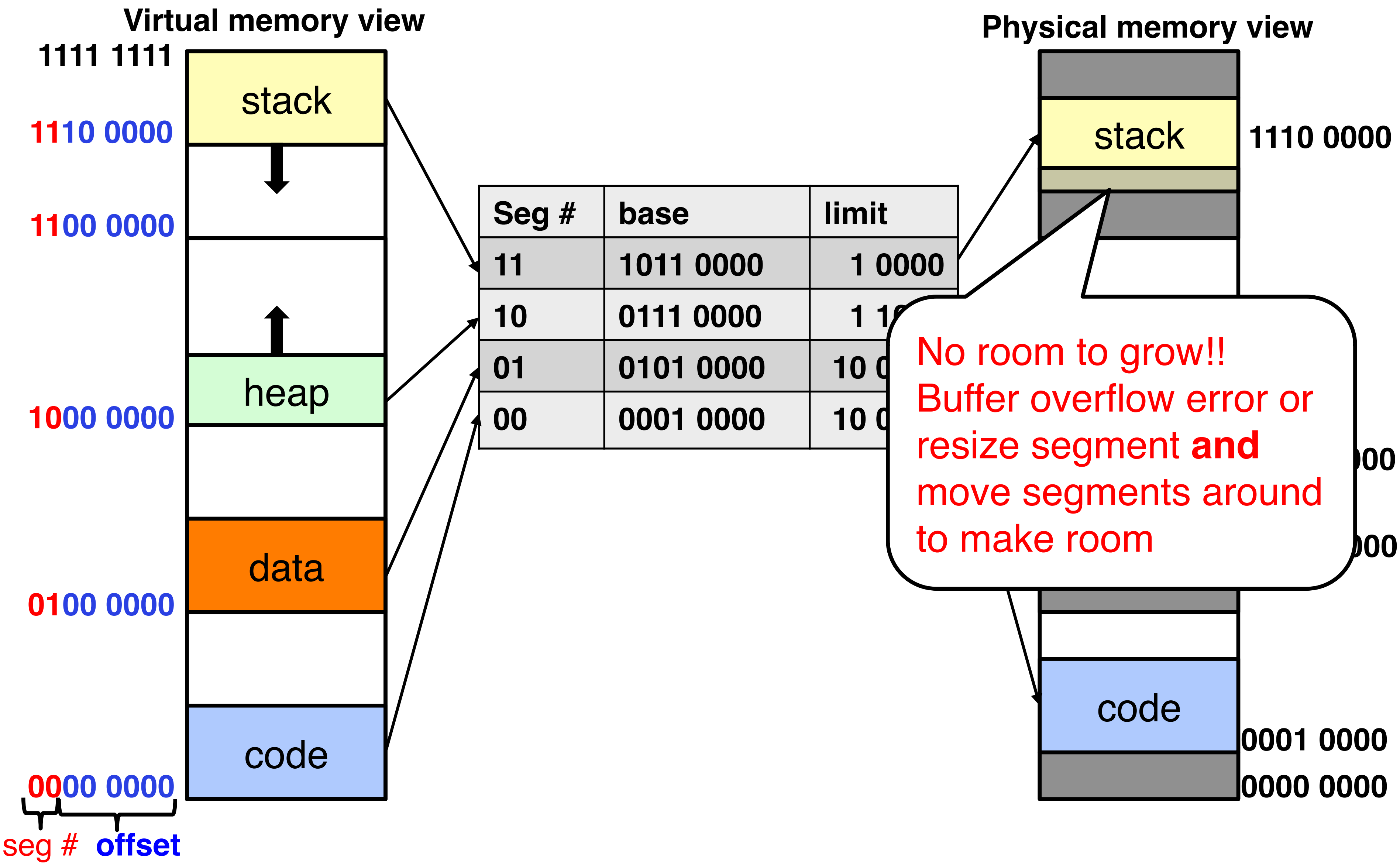
Review: Address Segmentation



Seg #	base	limit
11	1011 0000	1 0000
10	0111 0000	1 1000
01	0101 0000	10 0000
00	0001 0000	10 0000

What happens if stack grows to 1110 0000?

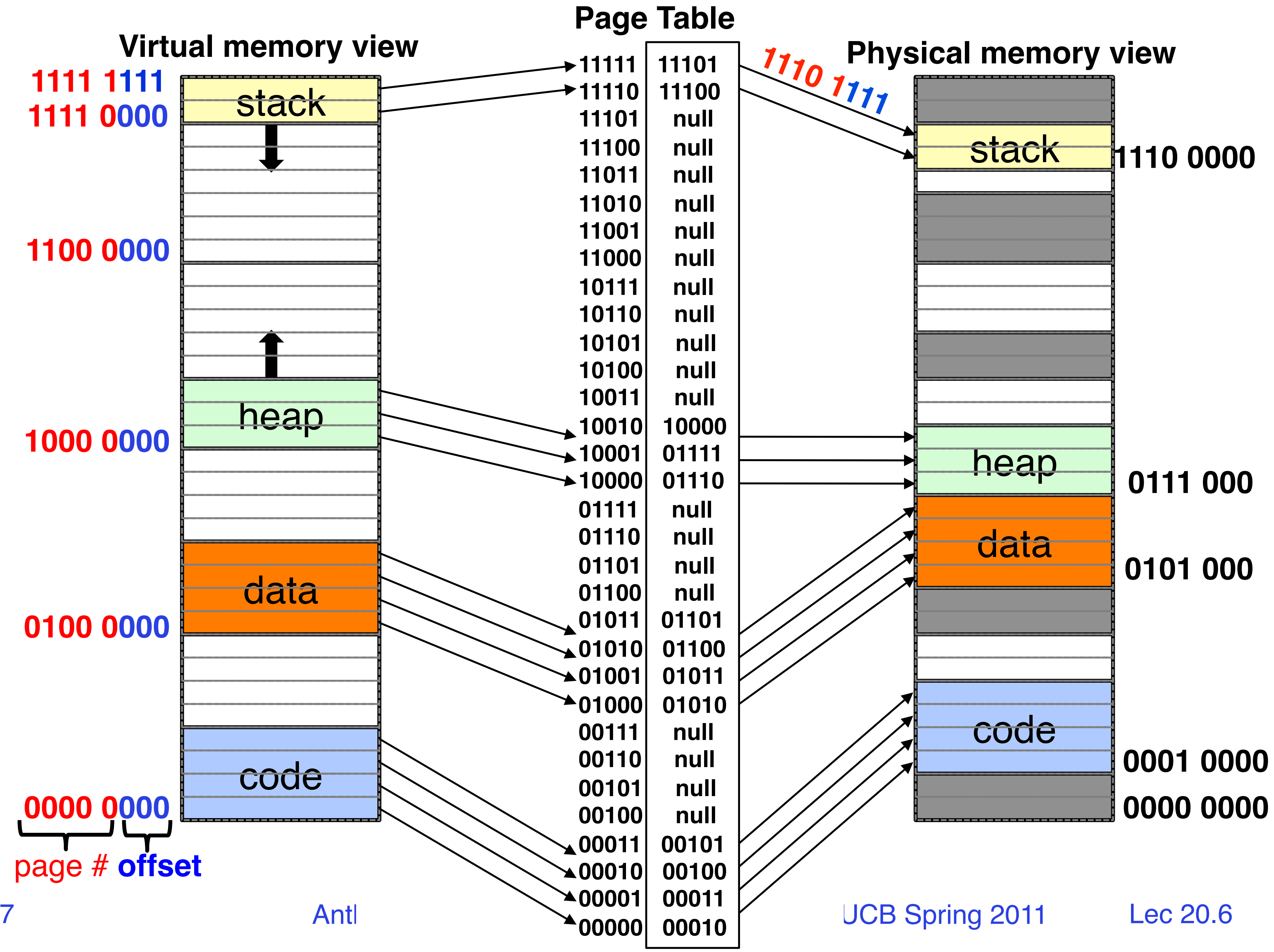
Review: Address Segmentation



Seg #	base	limit
11	1011 0000	1 0000
10	0111 0000	1 1
01	0101 0000	10 0
00	0001 0000	10 0

No room to grow!! Buffer overflow error or resize segment **and** move segments around to make room

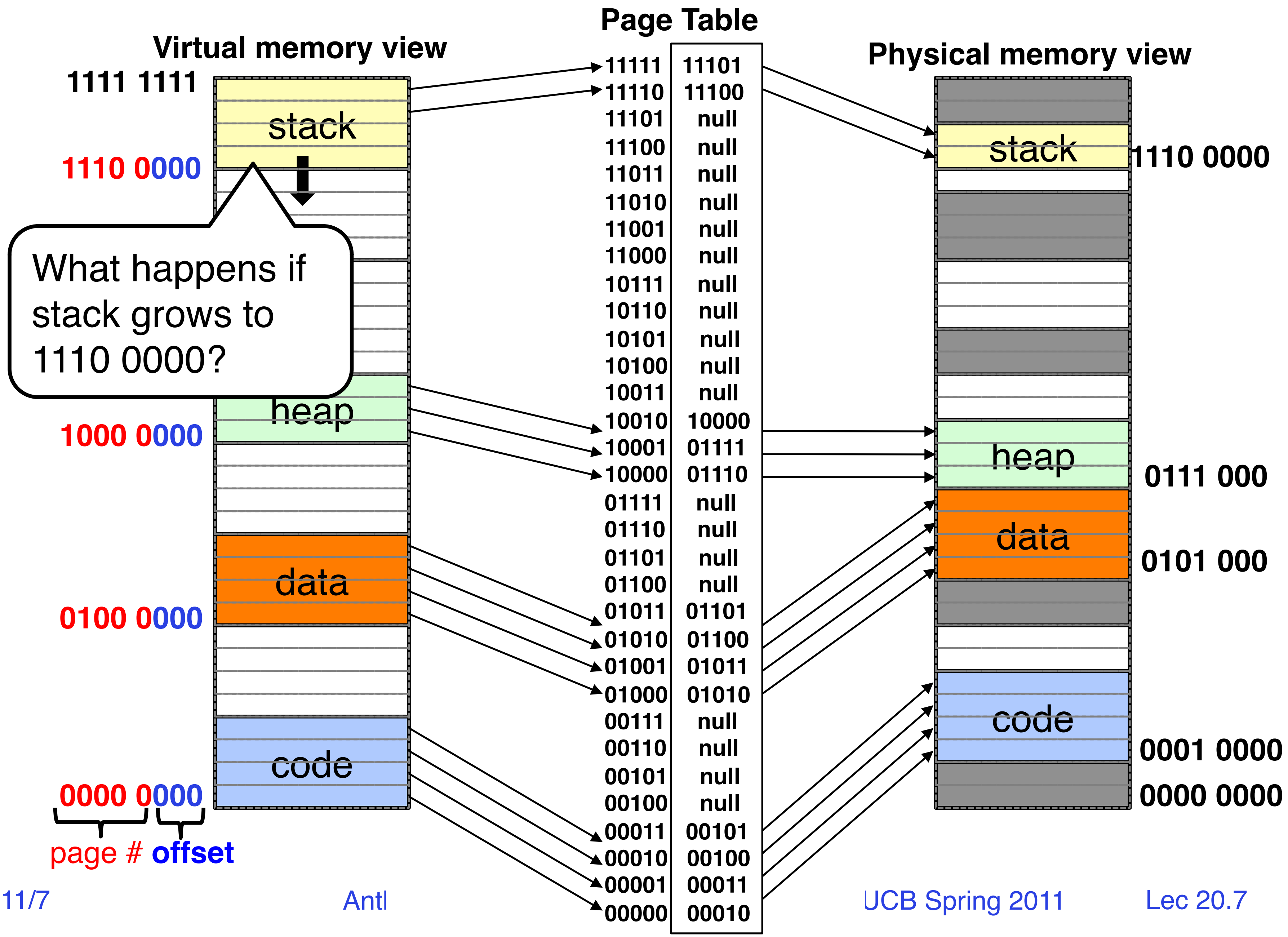
Review: Paging



Page Table

11111	11101
11110	11100
11101	null
11100	null
11011	null
11010	null
11001	null
11000	null
10111	null
10110	null
10101	null
10100	null
10011	null
10010	10000
10001	01111
10000	01110
01111	null
01110	null
01101	null
01100	null
01011	01101
01010	01100
01001	01011
01000	01010
00111	null
00110	null
00101	null
00100	null
00011	00101
00010	00100
00001	00011
00000	00010

Review: Paging

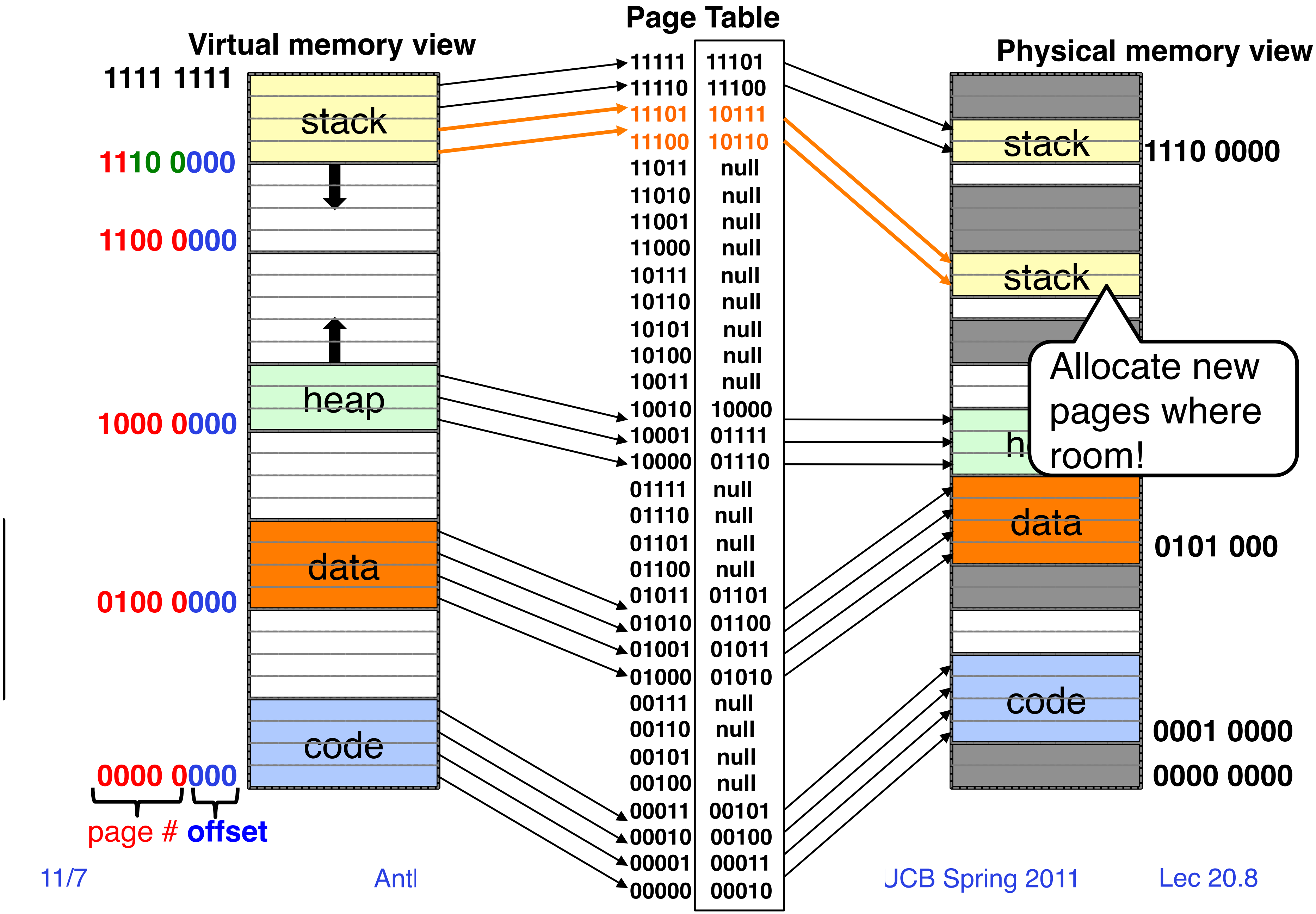


What happens if stack grows to 1110 0000?

Page Table

11111	11101
11110	11100
11101	null
11100	null
11011	null
11010	null
11001	null
11000	null
10111	null
10110	null
10101	null
10100	null
10011	null
10010	10000
10001	01111
10000	01110
01111	null
01110	null
01101	null
01100	null
01011	01101
01010	01100
01001	01011
01000	01010
00111	null
00110	null
00101	null
00100	null
00011	00101
00010	00100
00001	00011
00000	00010

Review: Paging

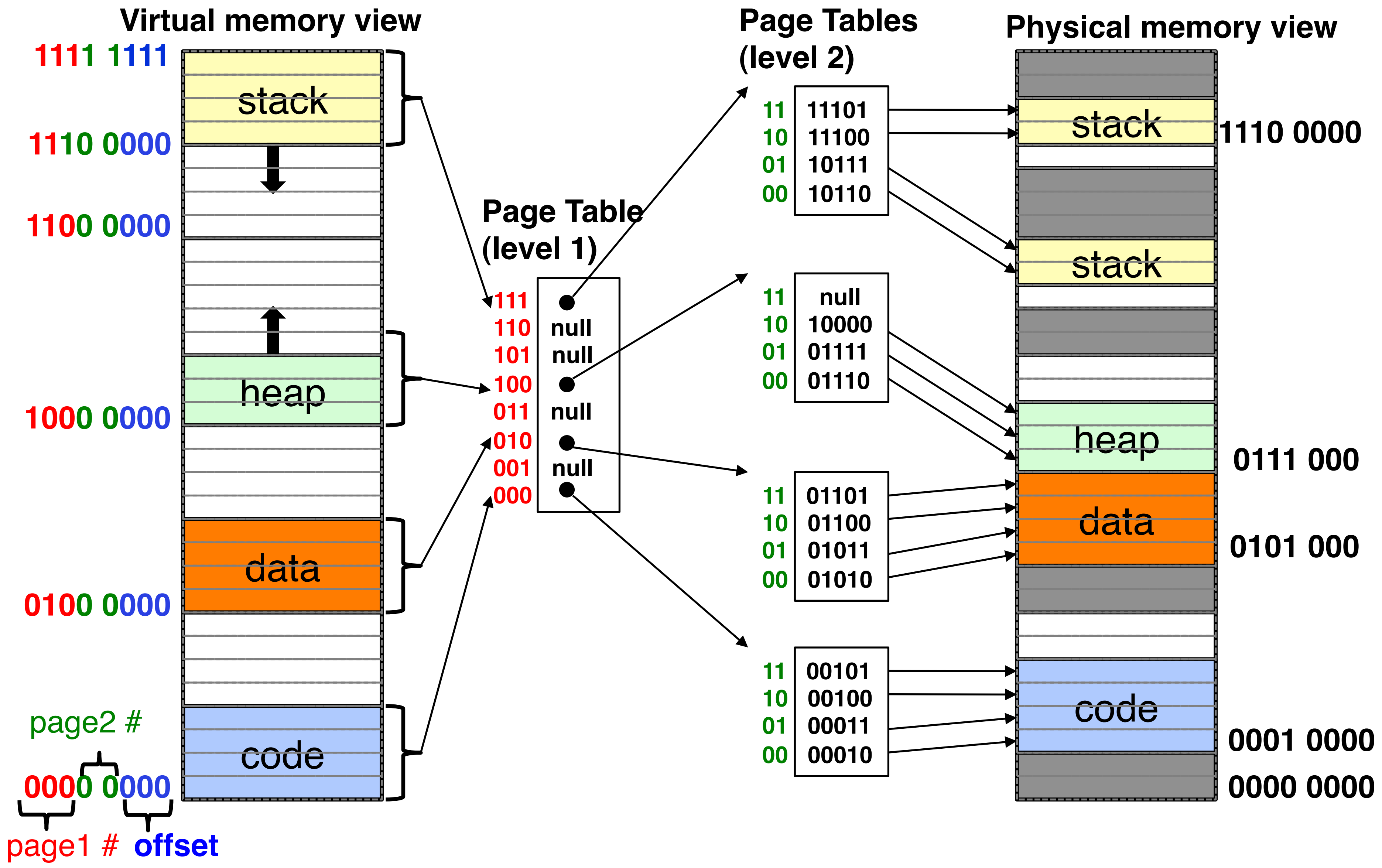


Page Table

11111	11101
11110	11100
11101	10111
11100	10110
11011	null
11010	null
11001	null
11000	null
10111	null
10110	null
10101	null
10100	null
10011	null
10010	10000
10001	01111
10000	01110
01111	null
01110	null
01101	null
01100	null
01011	01101
01010	01100
01001	01011
01000	01010
00111	null
00110	null
00101	null
00100	null
00011	00101
00010	00100
00001	00011
00000	00010

Allocate new pages where room!

Review: Two-Level Paging



111	●
110	null
101	null
100	●
011	null
010	●
001	null
000	●

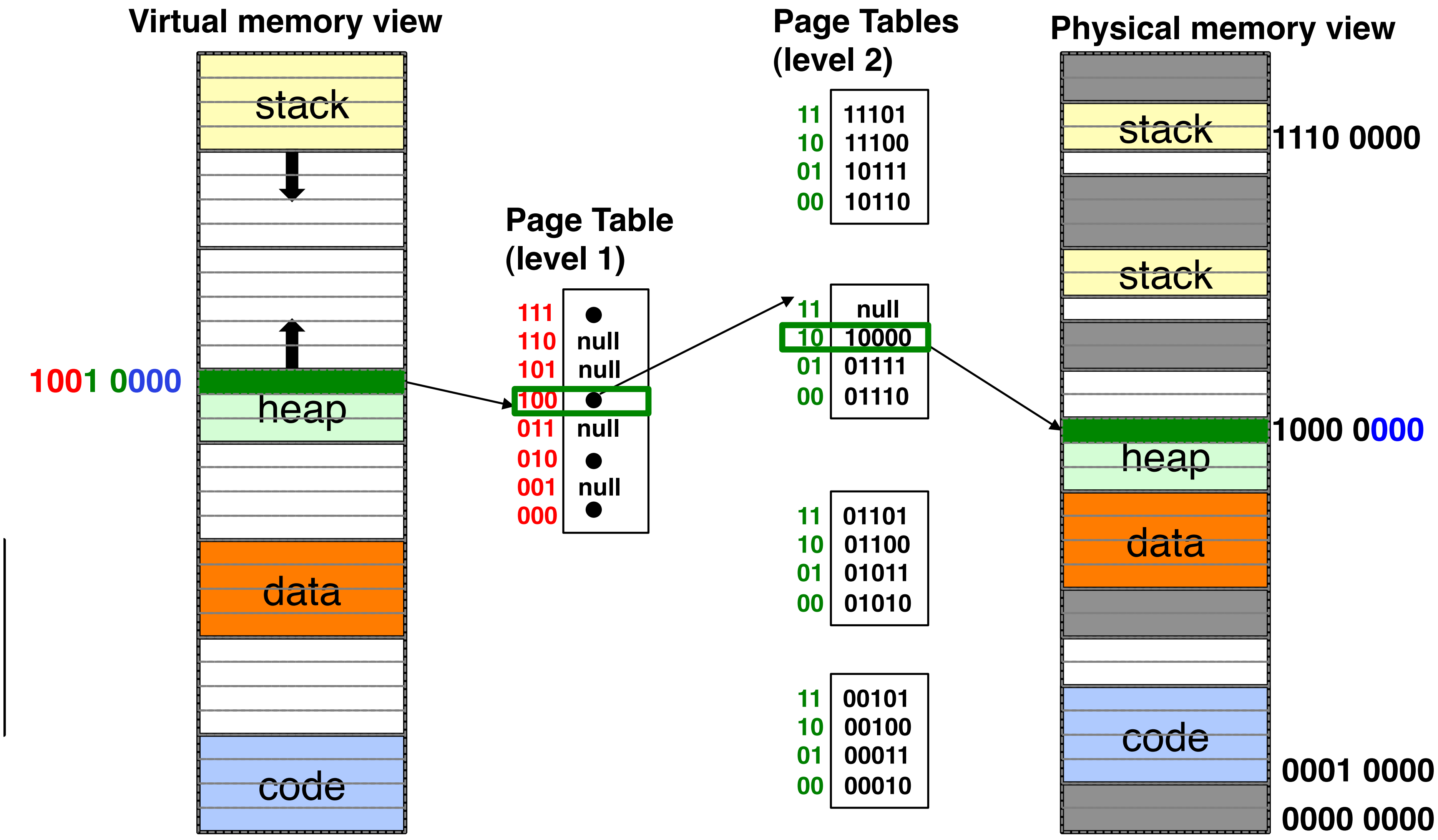
11	11101
10	11100
01	10111
00	10110

11	null
10	10000
01	01111
00	01110

11	01101
10	01100
01	01011
00	01010

11	00101
10	00100
01	00011
00	00010

Review: Two-Level Paging



Page Table (level 1)

111	●
110	null
101	null
100	●
011	null
010	●
001	null
000	●

Page Tables (level 2)

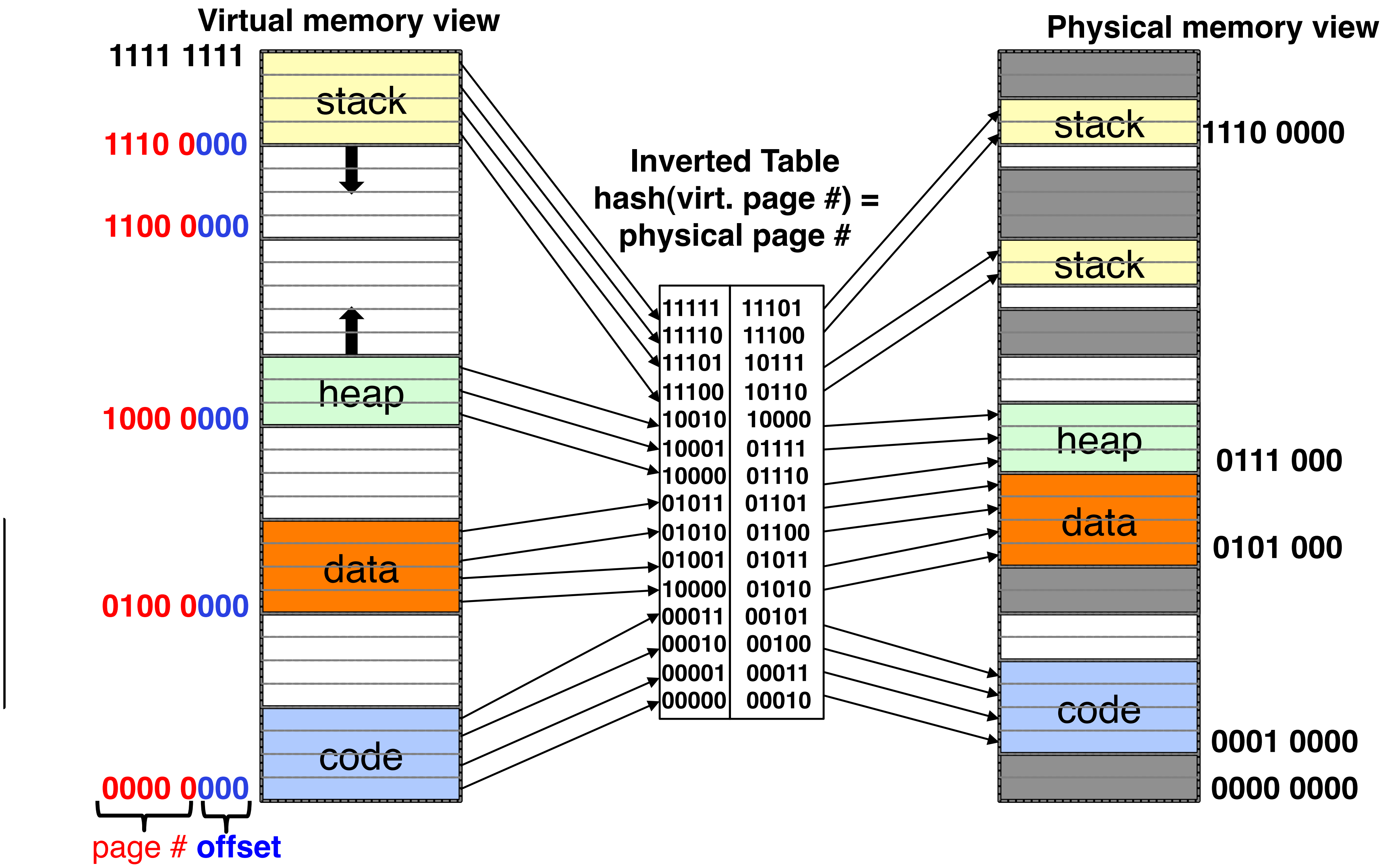
11	11101
10	11100
01	10111
00	10110

11	null
10	10000
01	01111
00	01110

11	01101
10	01100
01	01011
00	01010

11	00101
10	00100
01	00011
00	00010

Review: Inverted Table



Inverted Table
hash(virt. page #) =
physical page #

11111	11101
11110	11100
11101	10111
11100	10110
10010	10000
10001	01111
10000	01110
01011	01101
01010	01100
01001	01011
10000	01010
00011	00101
00010	00100
00001	00011
00000	00010

Address Translation Comparison

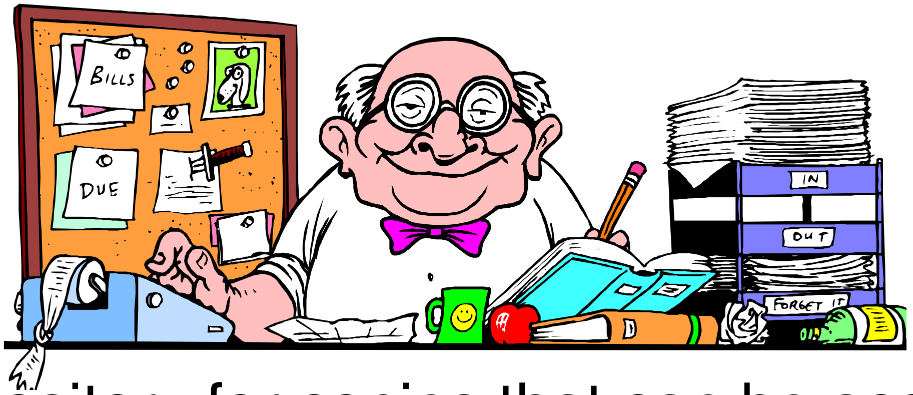
	Advantages	**Disadvantages**
Segmentation	Fast context switching: Segment mapping maintained by CPU	External fragmentation
Paging (single-level page)	No external fragmentation	Large size: Table size ~ virtual memory
Paged segmentation	Table size ~ # of pages in virtual memory	Multiple memory references per page access
Two-level pages		
Inverted Table	Table size ~ # of pages in physical memory	Hash function more complex

Goals for Today

- Caching
  - Misses
  - Organization
- Translation Look aside Buffers (TLBs)

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated from lecture notes by Kubiatowicz.

Caching Concept

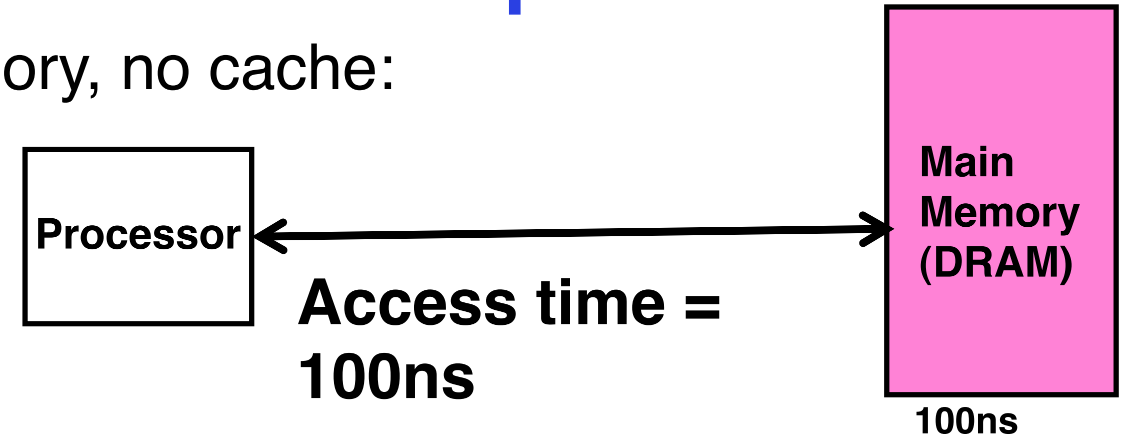


- Cache: a repository for copies that can be accessed more quickly than the original
  - Make frequent case fast and infrequent case less dominant
- Caching underlies many of the techniques that are used today to make computers fast
  - Can cache: memory locations, address translations, pages, file blocks, file names, network routes, etc…
- Only good if:
  - Frequent case frequent enough and
  - Infrequent case not too expensive
- Important measure: Average Access time =

  (Hit Rate x Hit Time) + (Miss Rate x Miss Time)

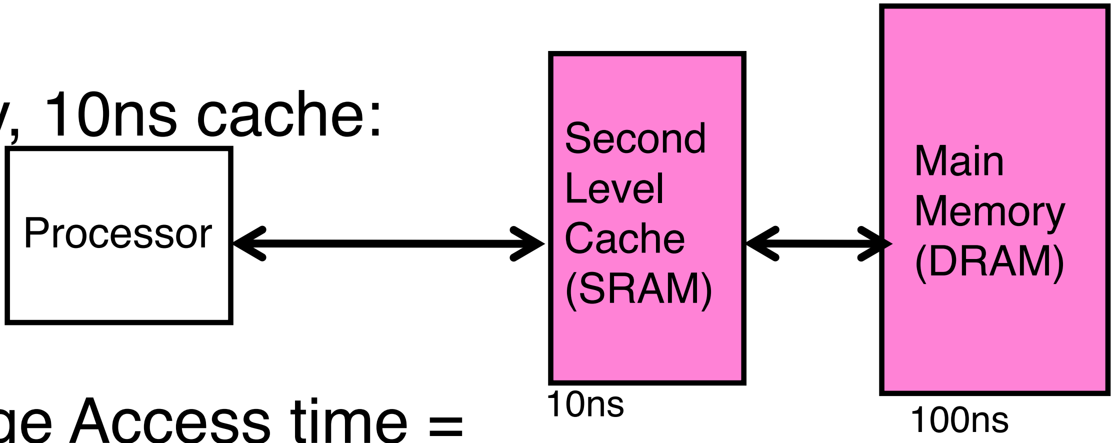
Example

- Data in memory, no cache:



Access time = 100ns

- Data in memory, 10ns cache:

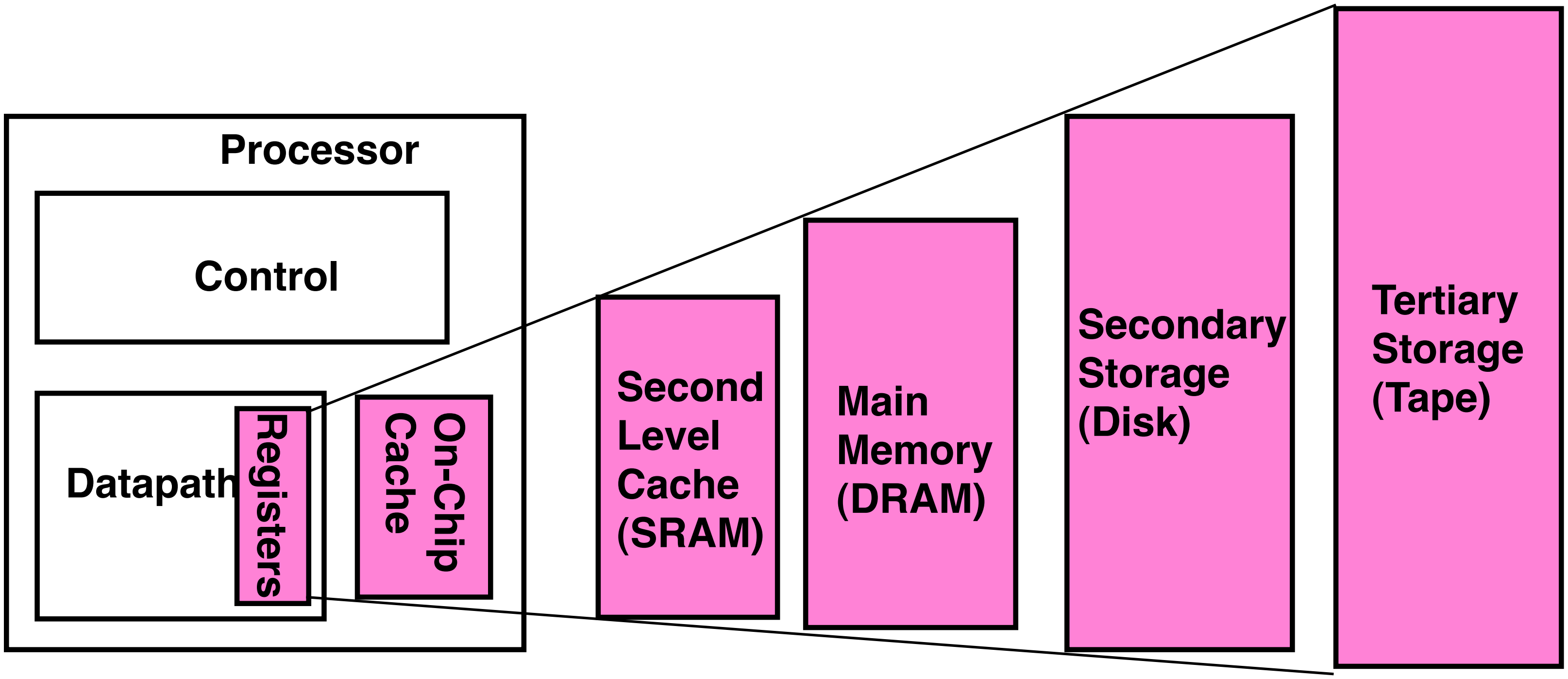


Average Access time =
(Hit Rate x HitTime) + (Miss Rate x MissTime)

- HitRate + MissRate = 1
- HitRate = 90% → Average Access Time = 19ns
- HitRate = 99% → Average Access Time = 10.9ns

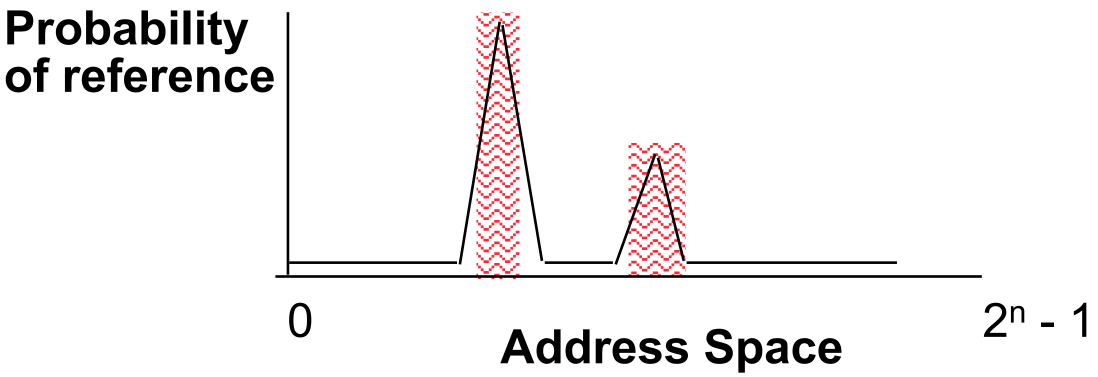
Review: Memory Hierarchy

- Take advantage of the principle of locality to:
  - – Present as much memory as in the cheapest technology
  - – Provide access at speed offered by the fastest technology

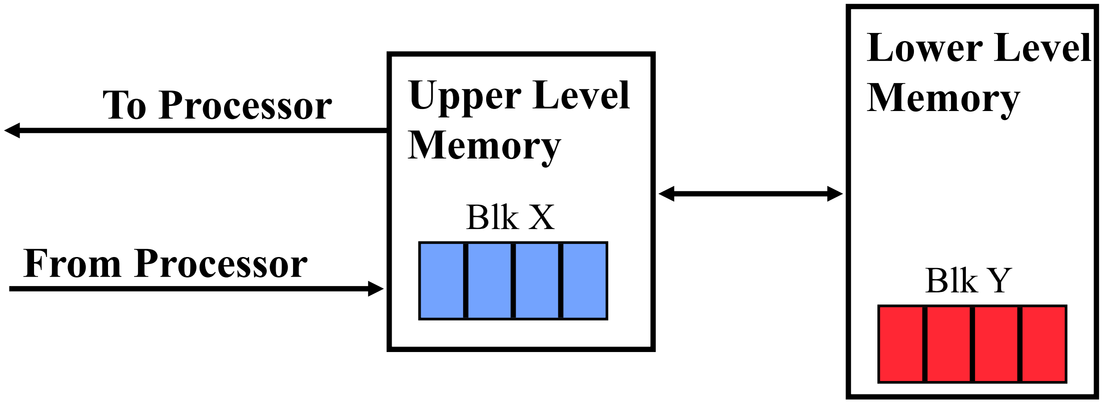


Speed (ns):	1s	10s-100s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks-Ms	Ms	Gs	Ts

Why Does Caching Help? Locality!



- Temporal Locality (Locality in Time):
  - Keep recently accessed data items closer to processor
- Spatial Locality (Locality in Space):
  - Move contiguous blocks to the upper levels

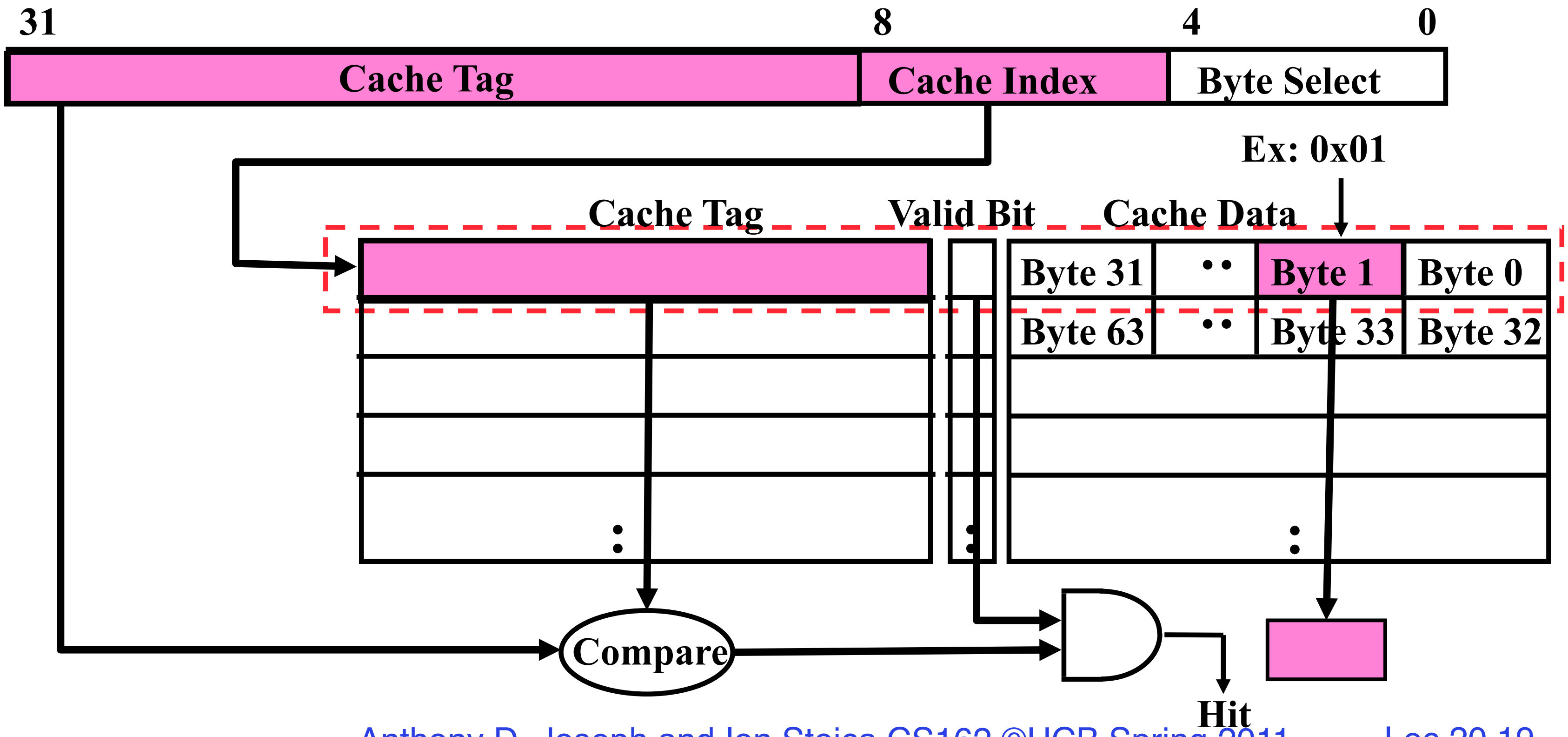


Review: Sources of Cache Misses

- Compulsory (cold start): first reference to a block
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: When running "billions" of instruction, Compulsory Misses are insignificant
- Capacity:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- Conflict (collision):
  - Multiple memory locations mapped to same cache location
  - Solutions: increase cache size, or increase associativity
- Two others:
  - Coherence (Invalidation): other process (e.g., I/O) updates memory
  - Policy: Due to non-optimal replacement policy

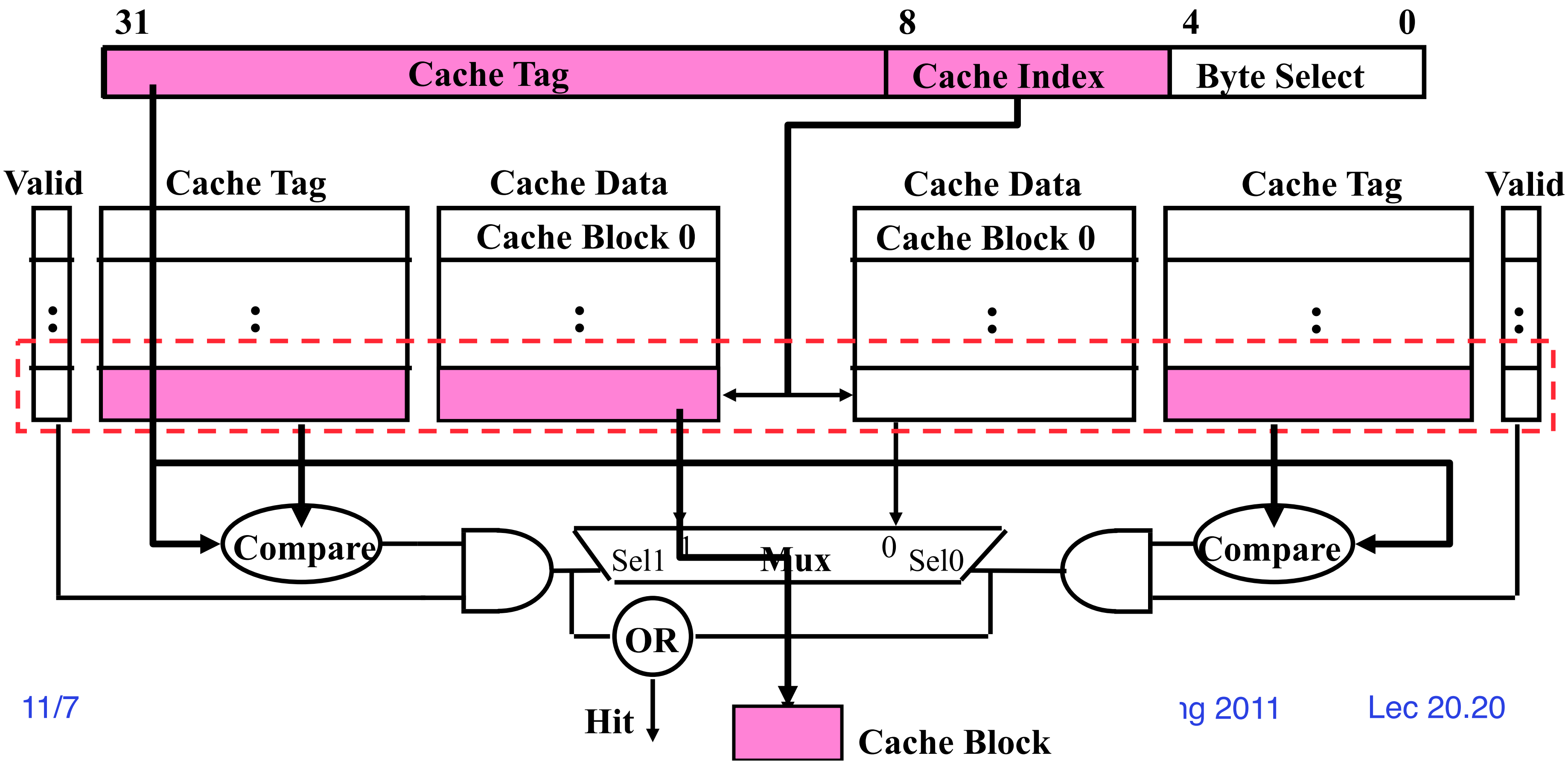
Direct Mapped Cache

- Cache index selects a cache block
- "Byte select" selects byte within cache block
  - – Example: Block Size=32B blocks
- Cache tag fully identifies the cached data
- Data with same "cache index" shares the same cache entry
  - – Conflict misses



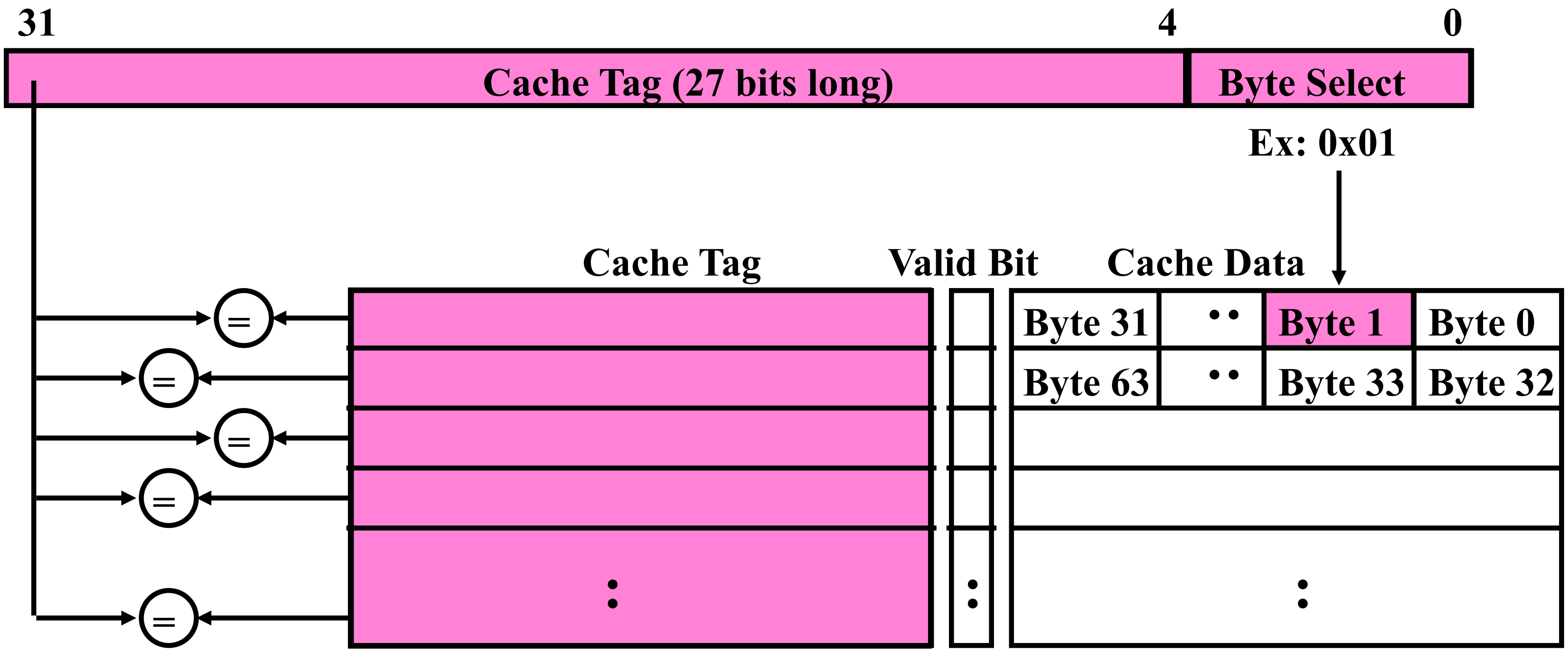
Set Associative Cache

- N-way set associative: N entries per Cache Index
  - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - Two tags in the set are compared to input in parallel
  - Data is selected based on the tag result



Fully Associative Cache

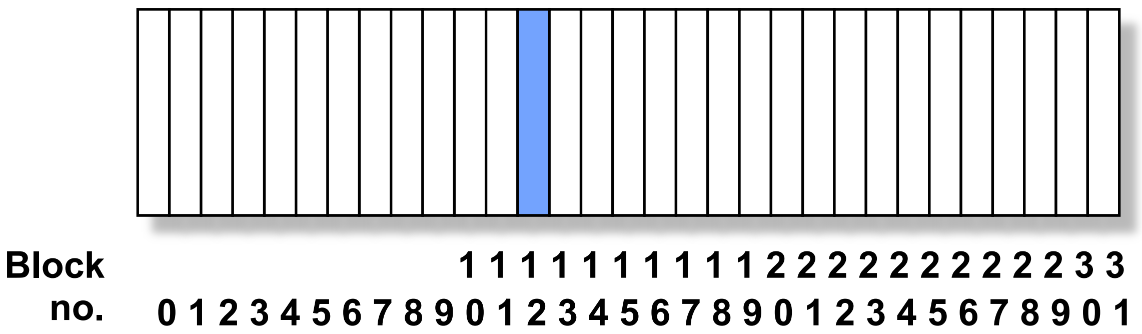
- Fully Associative: Every block can hold any line
  - – Address does not include a cache index
  - – Compare Cache Tags of all Cache Entries in Parallel
- Example: Block Size=32B blocks
  - – We need N 27-bit comparators
  - – Still have byte select to choose from within block



Where does a Block Get Placed in a Cache?

- Example: Block 12 placed in 8 block cache

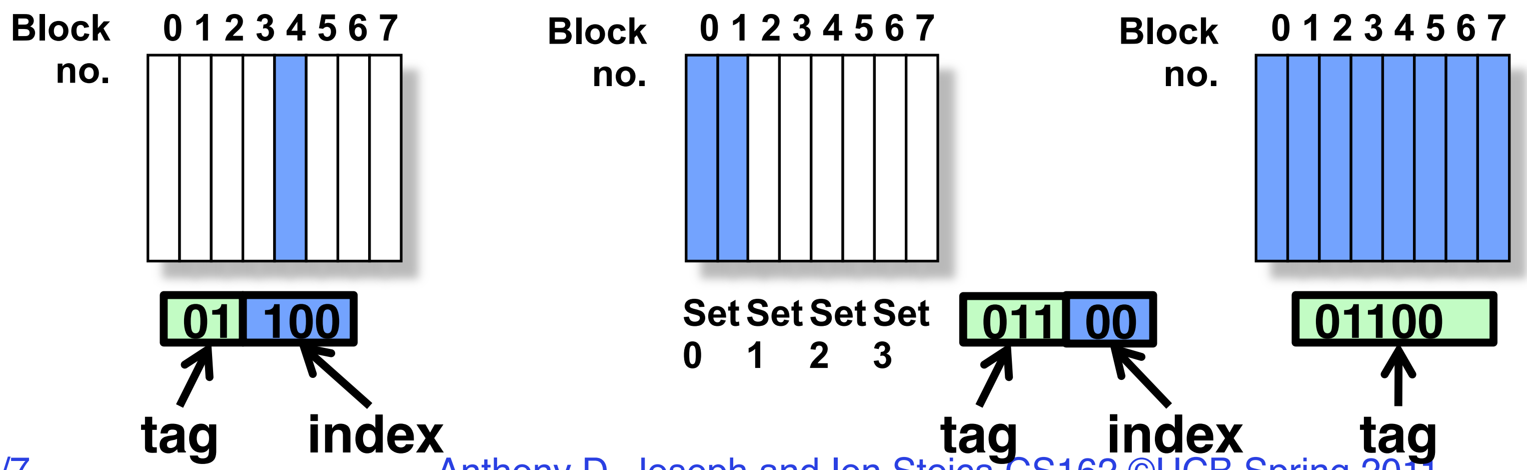
32-Block Address Space:



Direct mapped:
block 12 (01100) can go only into block 4 (12 mod 8)

Set associative:
block 12 can go anywhere in set 0 (12 mod 4)

Fully associative:
block 12 can go anywhere



Which block should be replaced on a miss?

- Easy for Direct Mapped: Only one possibility
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

What happens on a write?

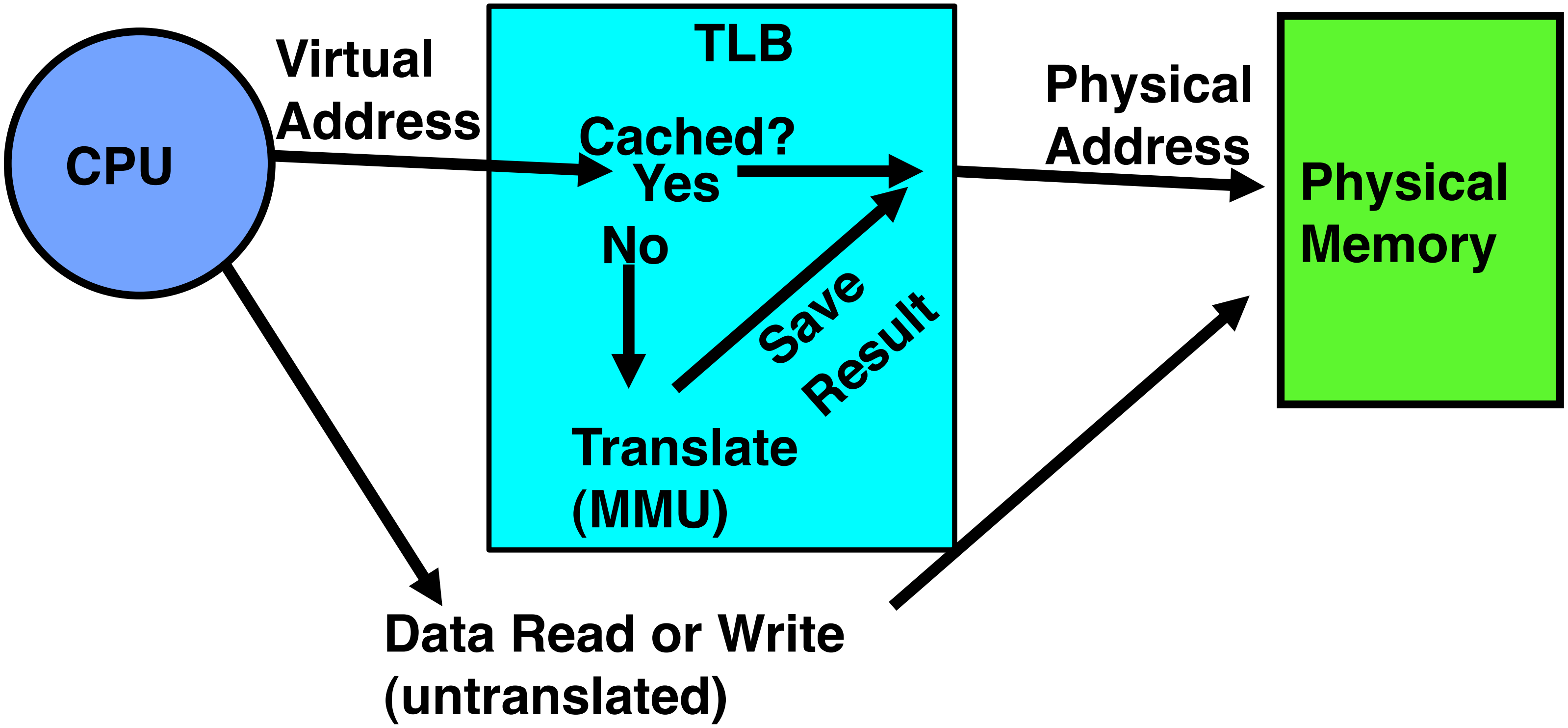
- Write through: The information is written both to the block in the cache and to the block in the lower-level memory
- Write back: The information is written only to the block in the cache.
  - Modified cache block is written to main memory only when it is replaced
  - Question is block clean or dirty?
- Pros and Cons of each?
  - WT:
    - PRO: read misses cannot result in writes
    - CON: processor held up on writes unless writes buffered
  - WB:
    - PRO: repeated writes not sent to DRAM
processor not held up on writes
    - CON: More complex
Read miss may require writeback of dirty data

Announcements

- Project 2 code due tomorrow: Tuesday, November 8, 11:59
- Project 3
  - EC2
  - Authentication
  - DB backend used for authentication, recording moves
  - Recovery from game server failure
- Exam regrades have been entered
- I'll be away November 8-17
  - No office hours next week
  - Samsung Forum (Seoul, Korea)
  - HotNets (MIT)

5min Break

Caching Applied to Address Translation



- Question is one of page locality: does it exist?
  - Instruction accesses spend a lot of time on the same page (since accesses sequential)
  - Stack accesses have definite locality of reference
  - Data accesses have less page locality, but still some…
- Can we have a TLB hierarchy?
  - Sure: multiple levels at different sizes/speeds

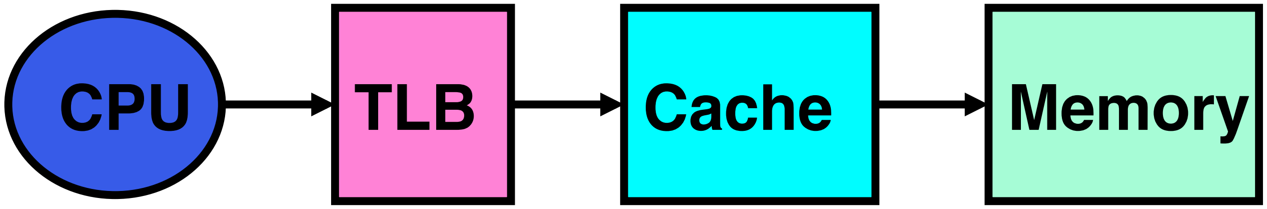
What Actually Happens on a TLB Miss?

- Hardware traversed page tables:
  - On TLB miss, hardware in MMU looks at current page table to fill TLB (may walk multiple levels)
    - » If PTE valid, hardware fills TLB and processor never knows
    - » If PTE marked as invalid, causes Page Fault, after which kernel decides what to do afterwards
- Software traversed Page tables
  - On TLB miss, processor receives TLB fault
  - Kernel traverses page table to find PTE
    - » If PTE valid, fills TLB and returns from fault
    - » If PTE marked as invalid, internally calls Page Fault handler
- Most chip sets provide hardware traversal
  - Modern operating systems tend to have more TLB faults since they use translation for many things

What happens on a Context Switch?

- Need to do something, since TLBs map virtual addresses to physical addresses
  - Address Space just changed, so TLB entries no longer valid!
- Options?
  - Invalidate TLB: simple but might be expensive
    - What if switching frequently between processes?
  - Include ProcessID in TLB
    - This is an architectural solution: needs hardware
- What if translation tables change?
  - For example, to move page from memory to disk or vice versa…
  - Must invalidate TLB entry!
    - Otherwise, might think that page is still in memory!

What TLB organization makes sense?



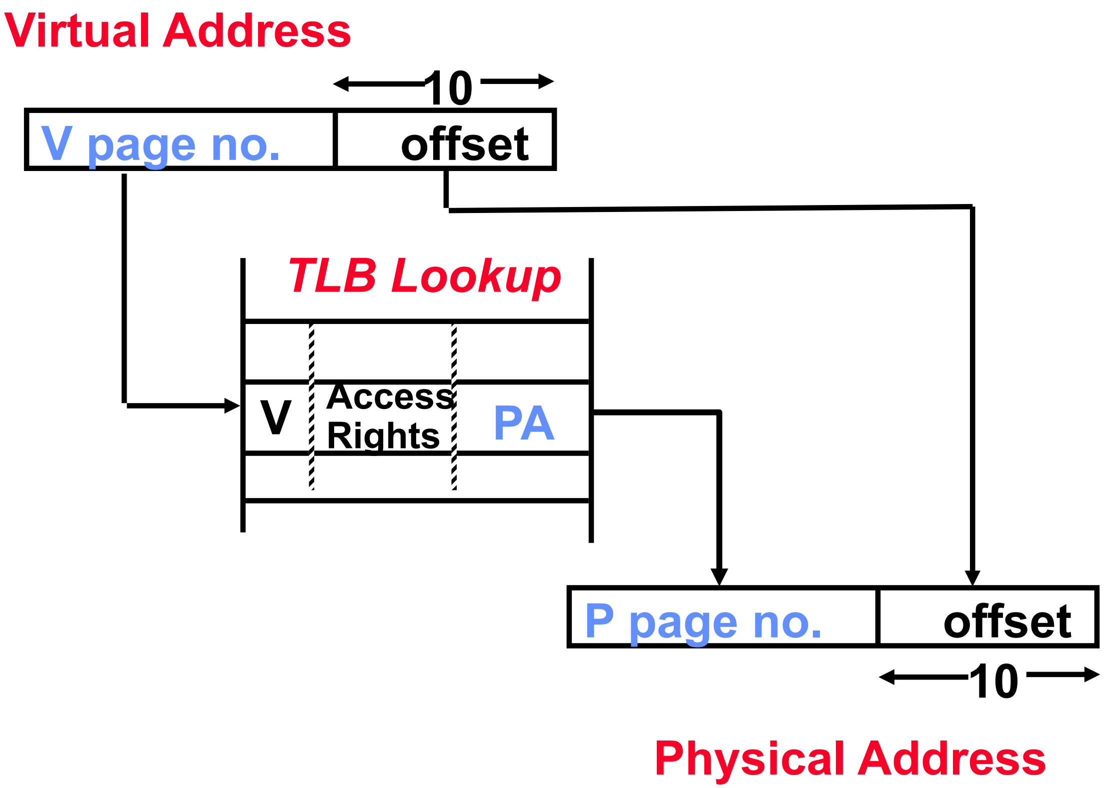
- Needs to be really fast
  - Critical path of memory access
  - Seems to argue for Direct Mapped or Low Associativity
- However, needs to have very few conflicts!
  - With TLB, the Miss Time extremely high!
  - This argues that cost of Conflict (Miss Time) is much higher than slightly increased cost of access (Hit Time)
- Thrashing: continuous conflicts between accesses
  - What if use low order bits of page as index into TLB?
    - » First page of code, data, stack may map to same entry
    - » Need 3-way associativity at least?
  - What if use high order bits as index?
    - » TLB mostly unused for small programs

TLB organization: include protection

- How big does TLB actually have to be?
  - Usually small: 128-512 entries
  - Not very big, can support higher associativity
- TLB usually organized as fully-associative cache
  - Lookup is by Virtual Address
  - Returns Physical Address + other info
- What happens when fully-associative is too slow?
  - Put a small (4-16 entry) direct-mapped cache in front
  - Called a "TLB Slice"
- When does TLB lookup occur?
  - Before cache lookup?
  - In parallel with cache lookup?

Reducing translation time further

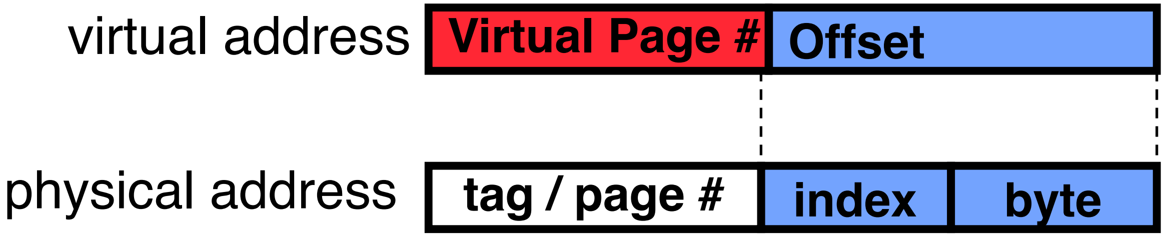
- As described, TLB lookup is in serial with cache lookup:



- Machines with TLBs go one step further: they overlap TLB lookup with cache access.
  – Works because offset available early

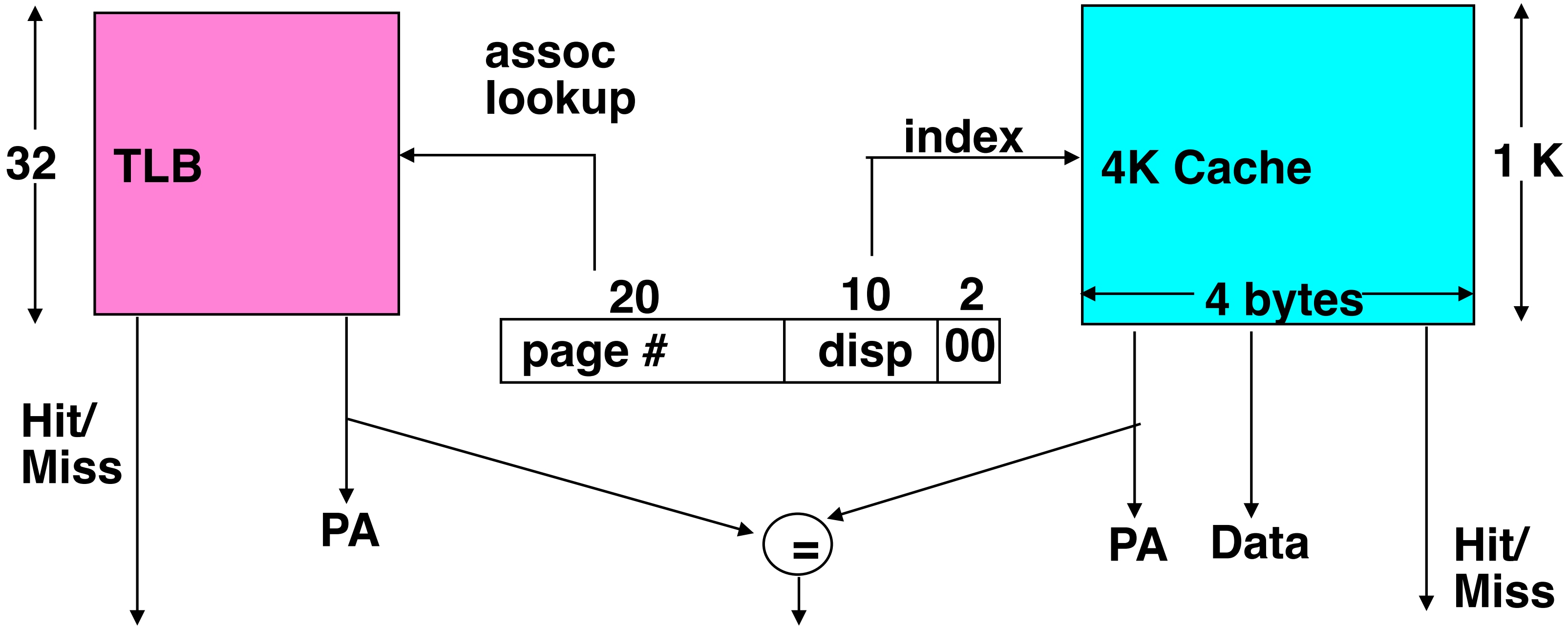
Overlapping TLB & Cache Access (1/2)

- Main idea:
  - Offset in virtual address exactly covers the “cache index” and “byte select”
  - Thus can select the cached byte(s) in parallel to perform address translation



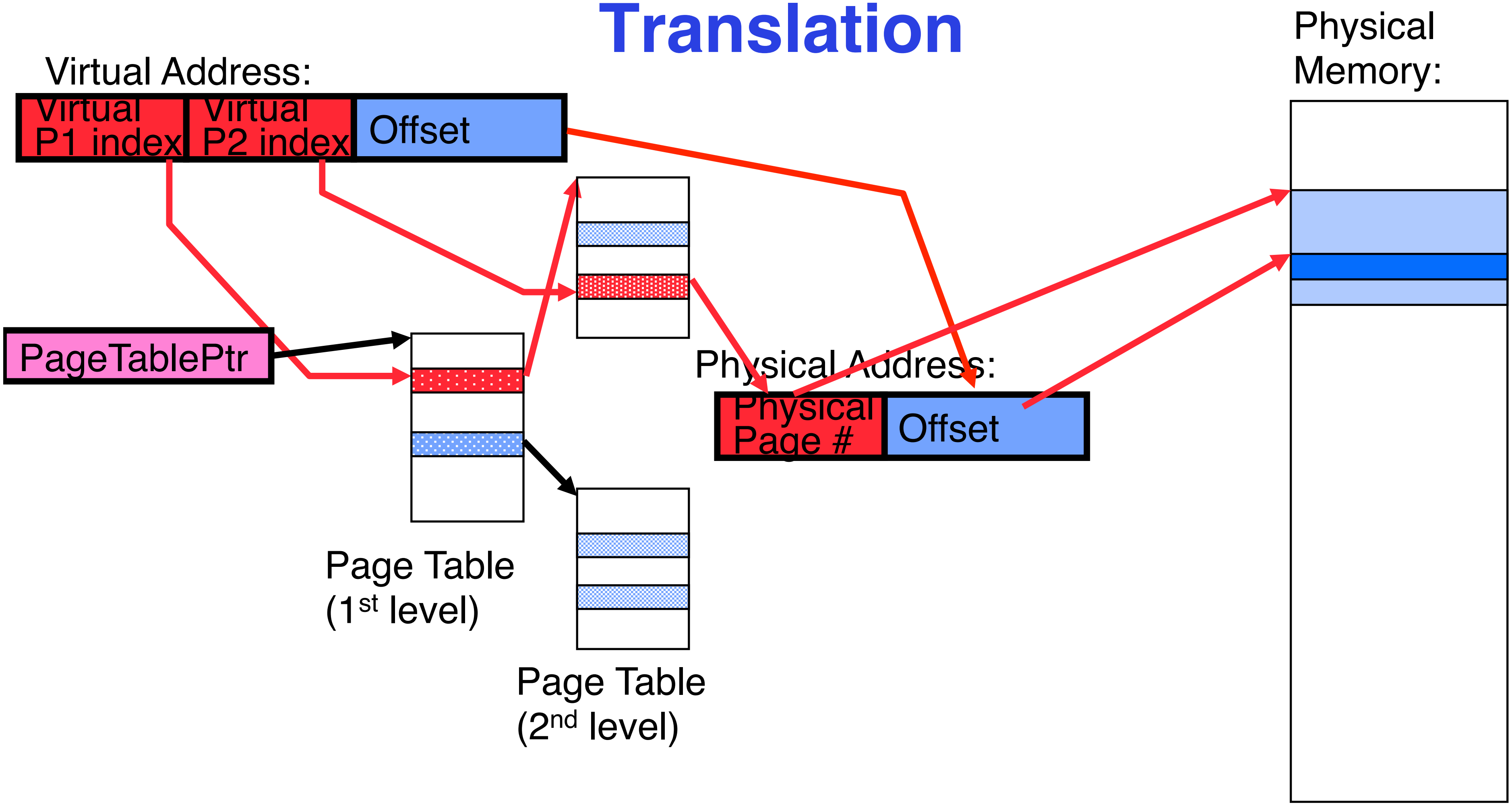
Overlapping TLB & Cache Access (1/2)

- Here is how this might work with a 4K cache:

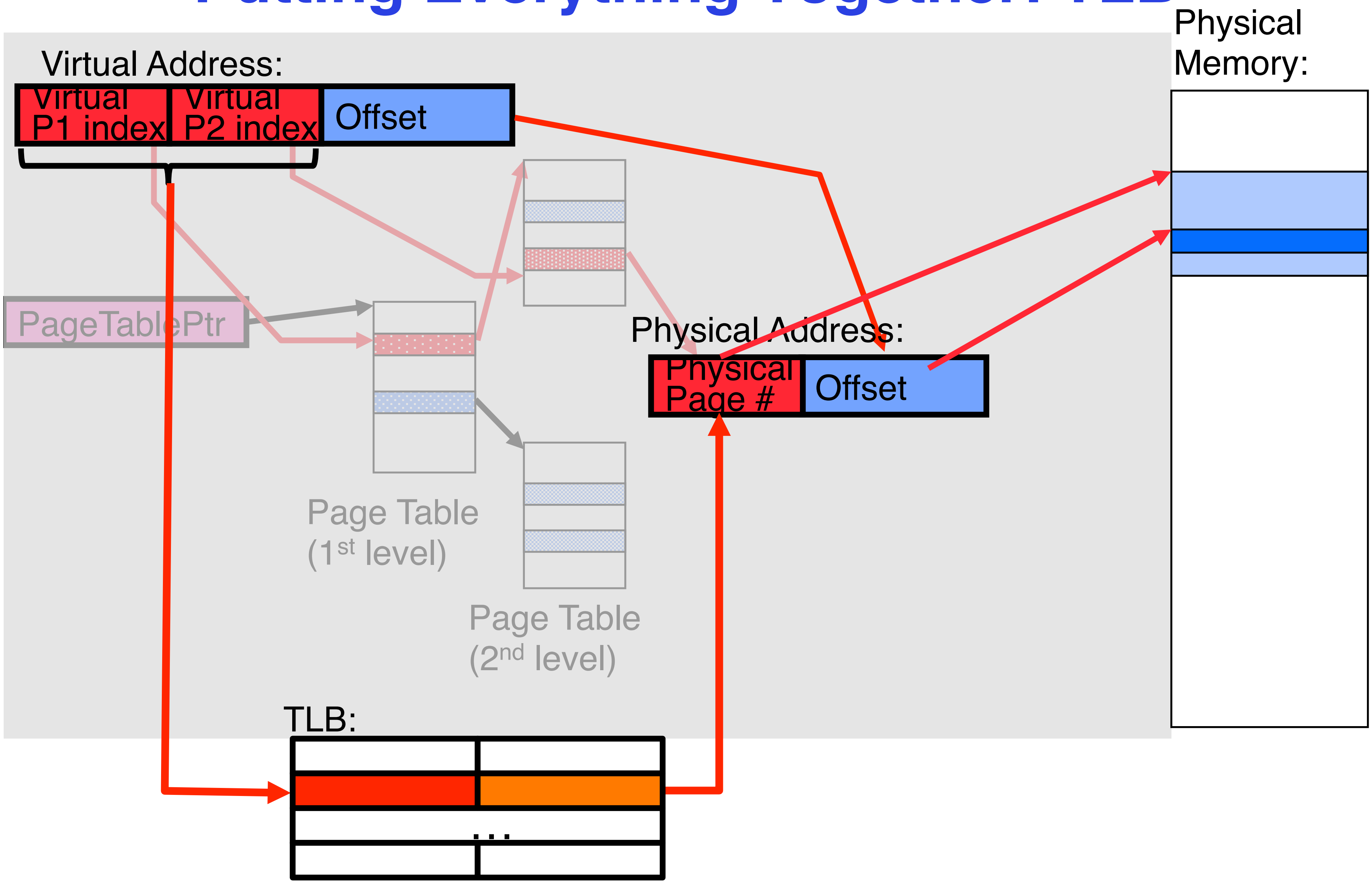


- What if cache size is increased to 8KB?
  - Overlap not complete
  - Need to do something else. See CS152/252

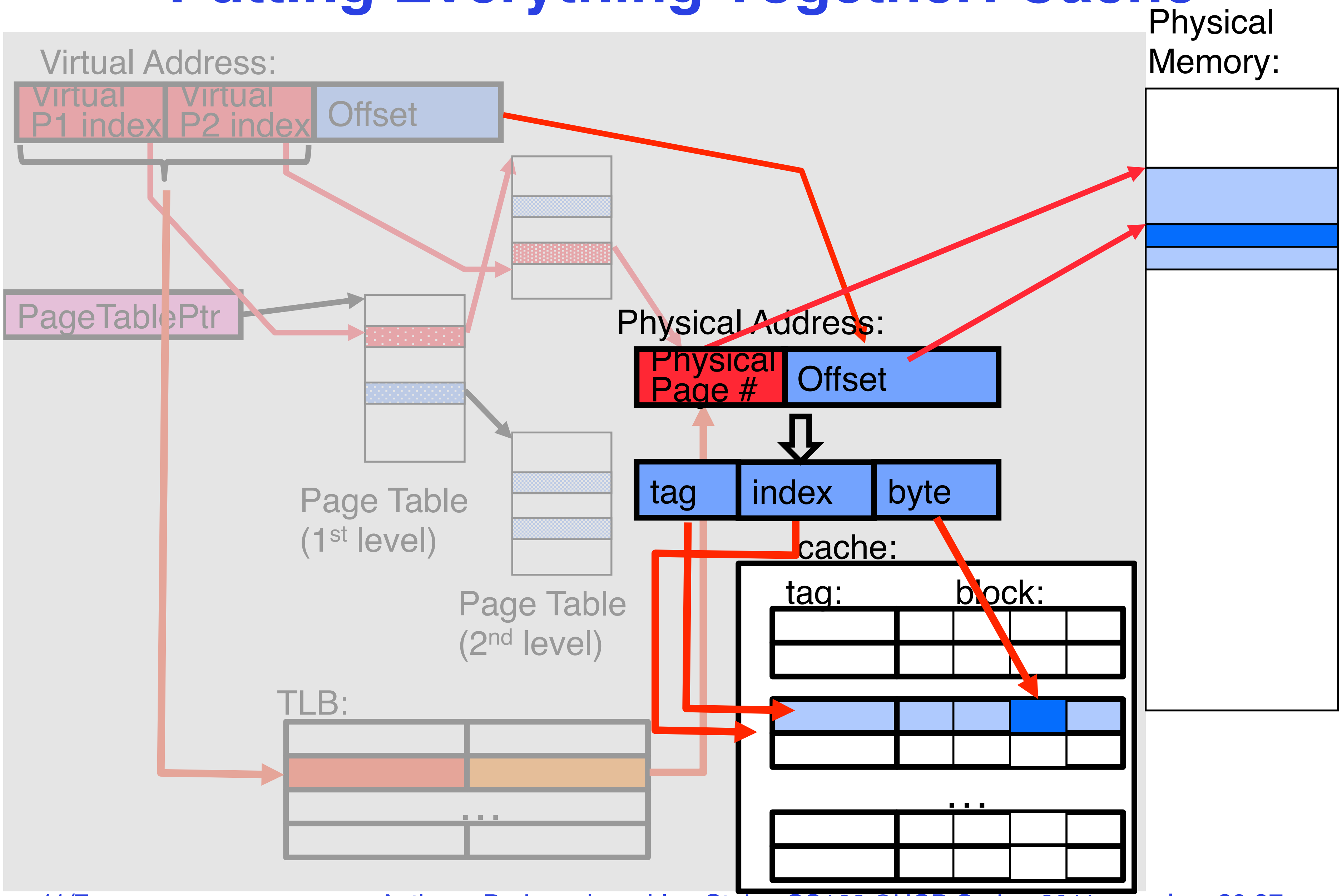
Putting Everything Together: Address Translation



Putting Everything Together: TLB



Putting Everything Together: Cache



Summary (1/2)

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - » Temporal Locality: Locality in Time
    - » Spatial Locality: Locality in Space

- Three (+1) Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity
  - Capacity Misses: increase cache size
  - Coherence Misses: Caused by external processors or I/O devices

Summary (2/2)

- Cache Organizations:
  - Direct Mapped: single block per set
  - Set associative: more than one block per set
  - Fully associative: all entries equivalent
- TLB is cache on address translations
  - Fully associative to reduce conflicts
  - Can be overlapped with cache access