Outline

- Cache basics
- Cache performance
  - Evaluation
  - Improvement
    - Set-associative caches
    - Block replacement policy
    - Multilevel caches
    - Program optimizations
- Virtual memory
- Concluding remarks

Program Optimizations

- Many transformations targeting cache performance performed by compilers
- Many ways to rewrite loops
  - Loop fusion

```
for (i=0; i<n; i++)
    A[i] = C[i] + D[i]
for (i=0; i<n; i++)
    B[i] = C[i] - D[i]
```

```
for (i=0; i<n; i++)
    A[i] = C[i] + D[i]
    B[i] = C[i] - D[i]
```

Program Optimizations

- Many transformations targeting cache performance performed by compilers
- Many ways to rewrite loops
  - Loop interchange
    - Assume row-major storage

```
for (j=1; j<n; j++)
    for (i=0; i<n; i++)
        X[i][j] = 2 * X[i][j-1]
```

Column-by-column accesses

```
for (i=0; i<n; i++)
    for (j=1; j<n; j++)
        X[i][j] = 2 * X[i][j-1]
```

Row-by-row accesses

Outline

- Cache basics
- Cache performance
  - Configurations
  - Evaluation
  - Improvement
- Virtual memory←
  - Four memory hierarchy questions
  - Page table and TLB
- Concluding remarks

Virtual Memory: Concept

- Use main memory as a "cache" for secondary (disk) storage
  - Virtual memory manages these two levels
  - Implemented jointly by hardware and the operating system
- Virtual memory
  - Allows efficient and safe sharing of memory among multiple programs
    - Each gets a private virtual address space
    - Virtual memory maps virtual address spaces to the shared physical memory: controlled access
  - Removes the programming burdens of a small, limited amount of main memory
    - Allow the size of a user program exceed the size of primary memory

Memory vs. Cache

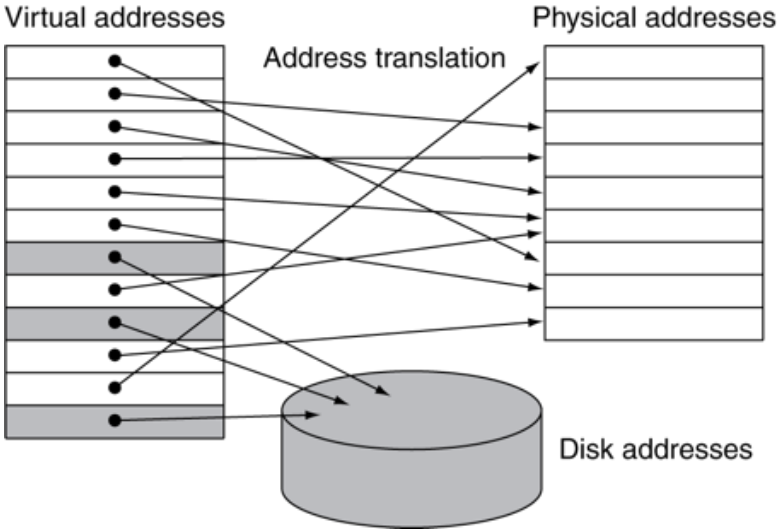
- Analogy to cache
  - Size: cache << memory << address space
  - Both provide big and fast memory - exploit locality
  - Both need a policy - 4 memory hierarchy questions
    - Mapping between cache block number and memory address ⇔ mapping between virtual memory address and physical memory frames
    - Cache blocks ⇔ memory pages
    - Cache misses ⇔ page faults
- Difference from cache
  - Cache primarily focuses on speed
  - VM facilitates transparent memory management
    - Providing large address space
    - Sharing, protection in multi-programming environment

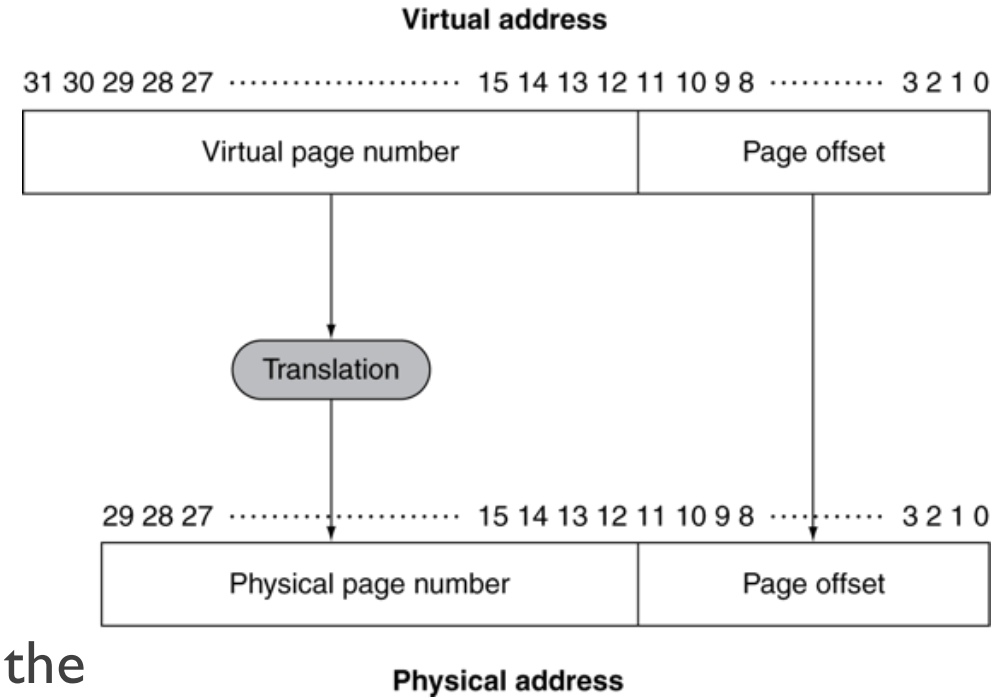
Four Memory Hierarchy Questions

1. Where can a page be placed in main memory?
   - OS allows block to be placed anywhere: fully associative
     - No conflict misses; simpler mapping provides no advantage for software handler
2. Which page should be replaced?
   - An approximation of LRU: true LRU too costly and adds little benefit
     - A reference bit is set to 1 if a page is accessed
     - Periodically cleared to 0 by OS
     - A page with reference bit = 0 has not been used recently
3. What happens on a write?
   - Write back: write through is prohibitively expensive

Four Memory Hierarchy Questions

4. How is a page found in main memory?
   - Translate virtual address into physical address when we load a page in





   - Would like to remember the translation/mapping for future accesses!

Outline

- Cache basics
- Cache performance
  - Configurations
  - Evaluation
  - Improvement
- Virtual memory
  - Four memory hierarchy questions
  - Implementation: Page table and TLB
- Concluding remarks

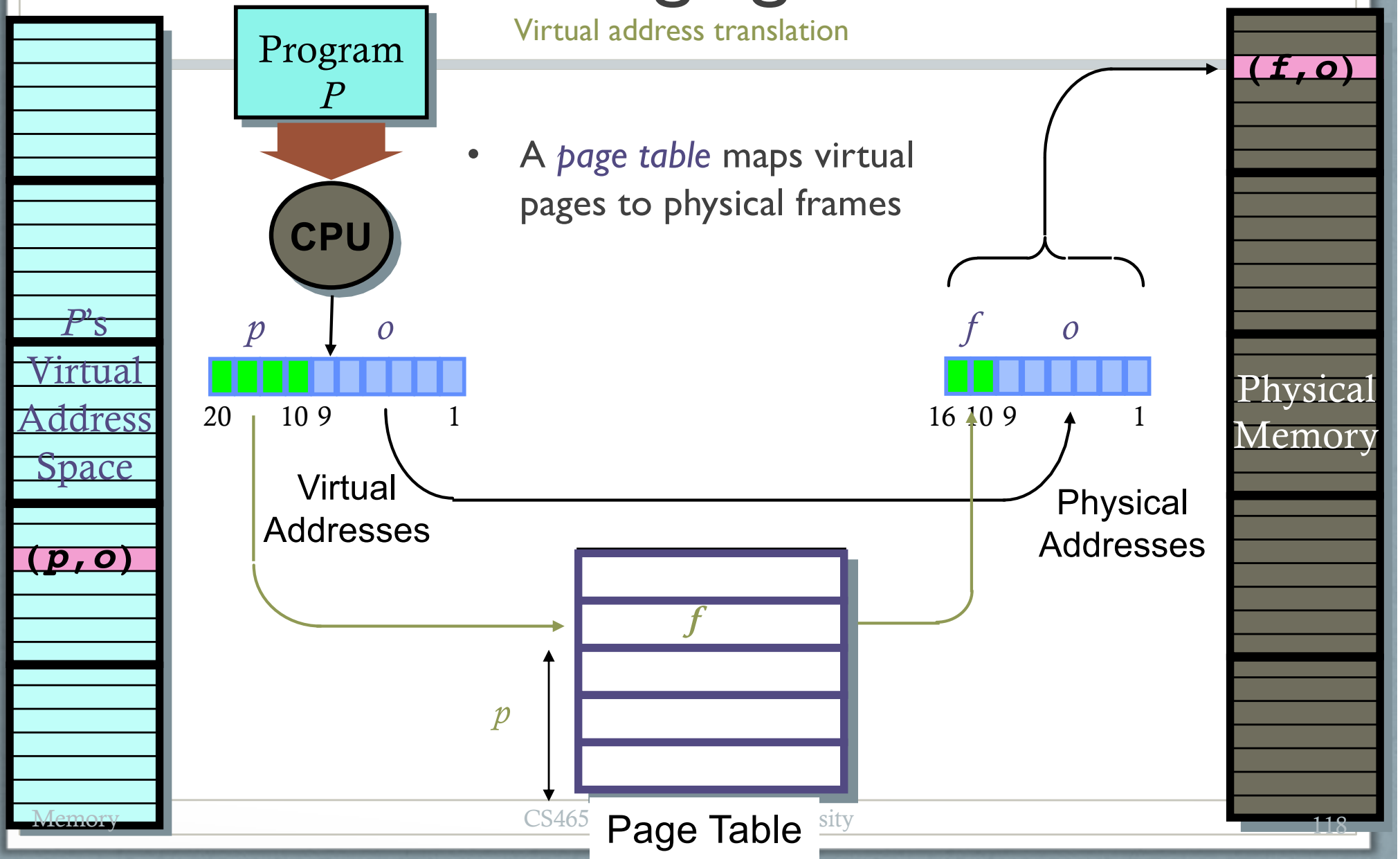
Implementing Virtual Memory

- Problem: given a virtual address request, locate the address in physical memory
- Approach:

  - Fully-associative cache
    - Tag: frame number / page number; Offset: frame / page offset
  - Need to map the virtual address requested by processor needs to a physical memory address
    - (VPN, offset) ➔ (PPN, offset)
      - Virtual Page Number, Physical Page Number
    - Mapping determined when this virtual page is first loaded into memory and remembered in page table for future accesses

Paging

Virtual address translation



- A *page table* maps virtual pages to physical frames

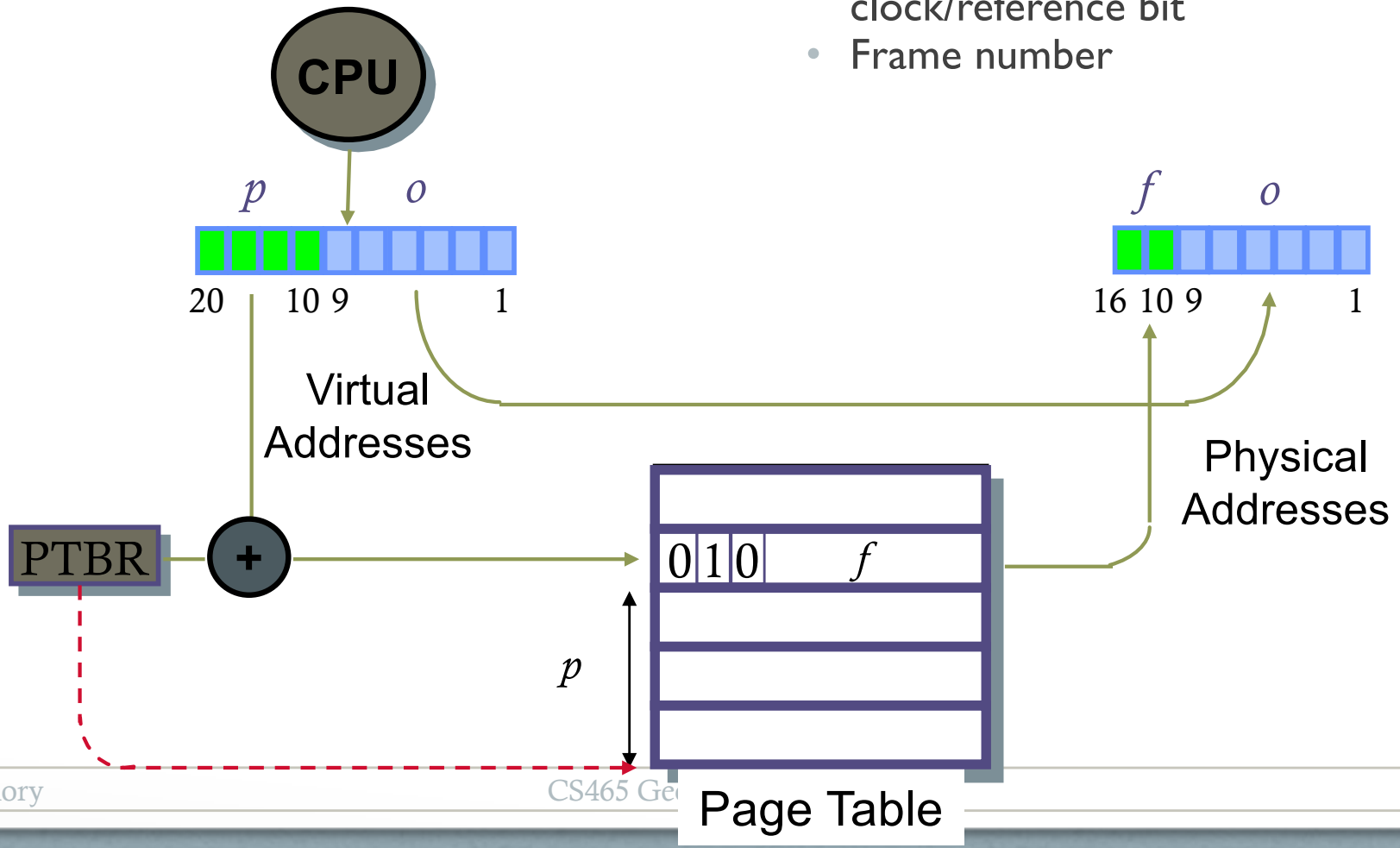
Virtual Address Translation Details

Page table structure

1 table per process

Part of process's state

- Contents:
  - Flags — dirty bit, resident bit, clock/reference bit
  - Frame number



Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table base register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - Page fault
  - PTE can refer to location in swap space on disk

Page Fault

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
    - Hardware must detect the situation, then trap to the operating system so that it can remedy the situation
    - Pick a page to discard (possibly writing it to disk)
    - Start loading the page in from disk
    - Schedule some other process to run
- Later (when page has come back from disk):
  - Update the page table
  - Resume to program so HW will retry and succeed!
- Try to minimize page fault rate
  - Fully associative placement
  - Approximate LRU replacement algorithms

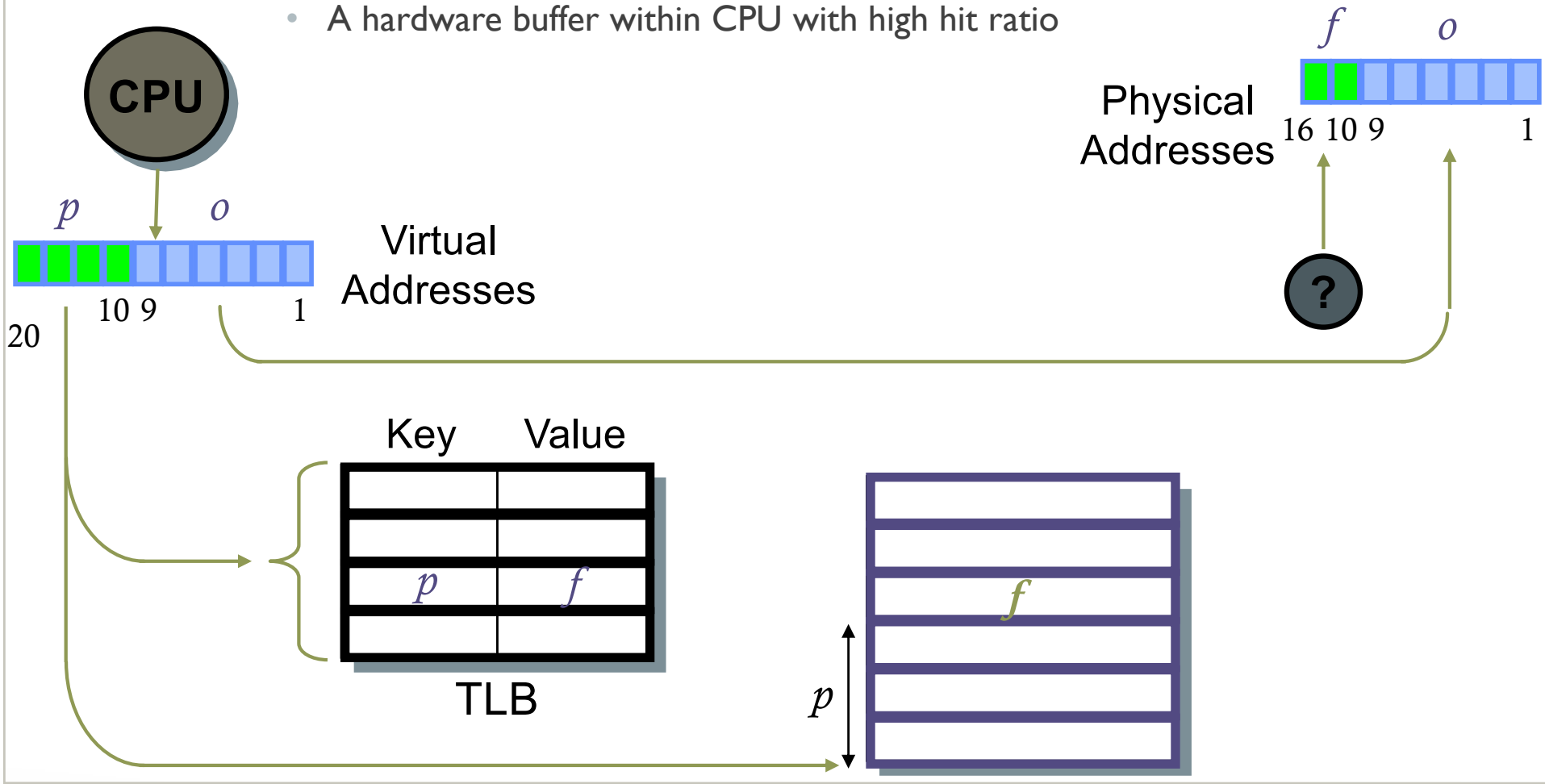
Performance Issue

- We remember the translation/mapping between VPN and PPN in page table
  - Space overhead
  - Time overhead
- How to improve?
  - Multi-level page table:
    - Only need to load partial page table in use (reduced size)
  - TLB: taking advantage of the locality of page table accesses
    - Caching latest page table accesses

TLB(Translation Look-aside Buffer)

Using TLBs to Speedup Address Translation

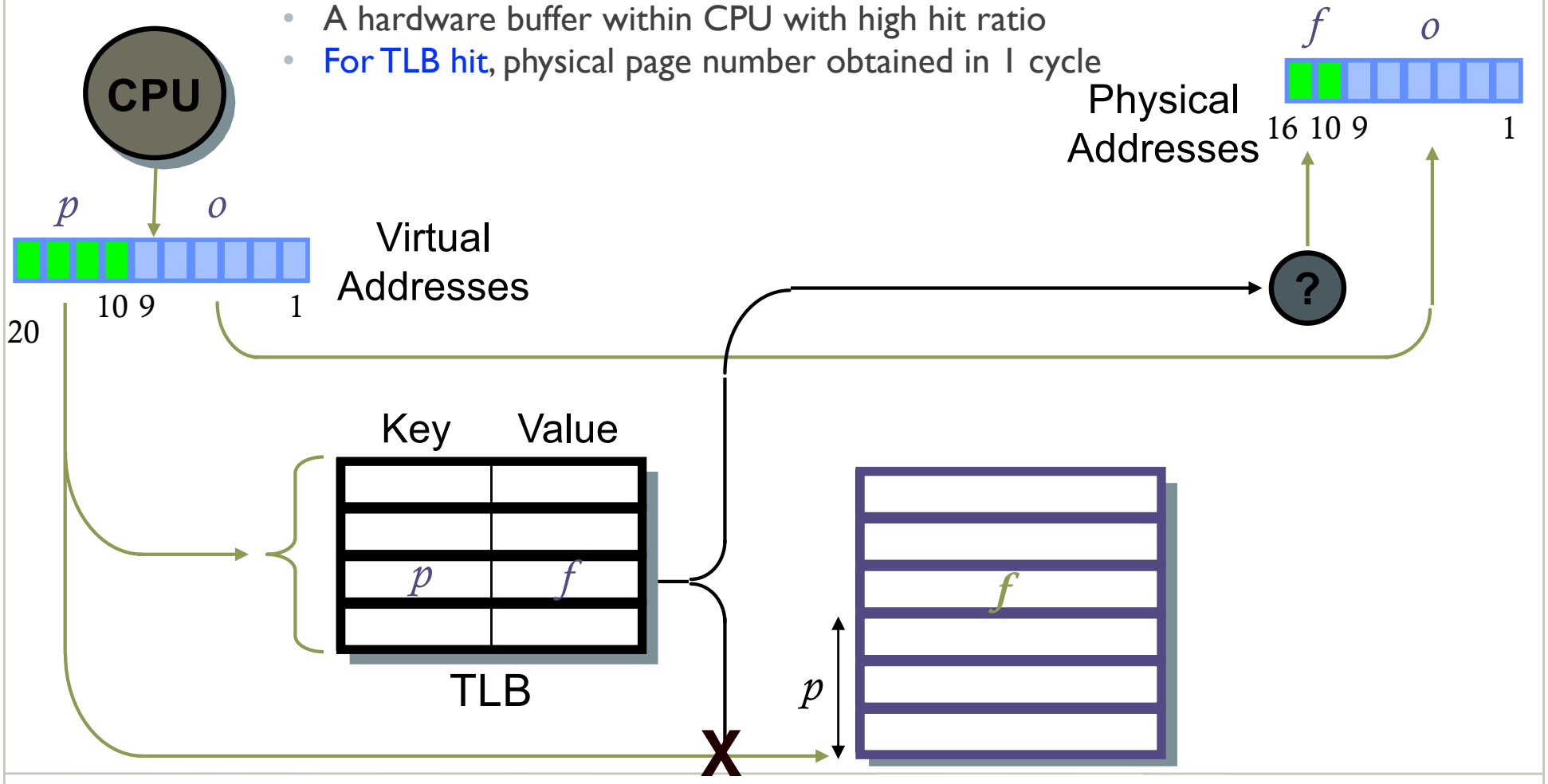
- Cache recently accessed page-to-frame translations in a TLB
  - A hardware buffer within CPU with high hit ratio



TLB(Translation Look-aside Buffer)

Using TLBs to Speedup Address Translation

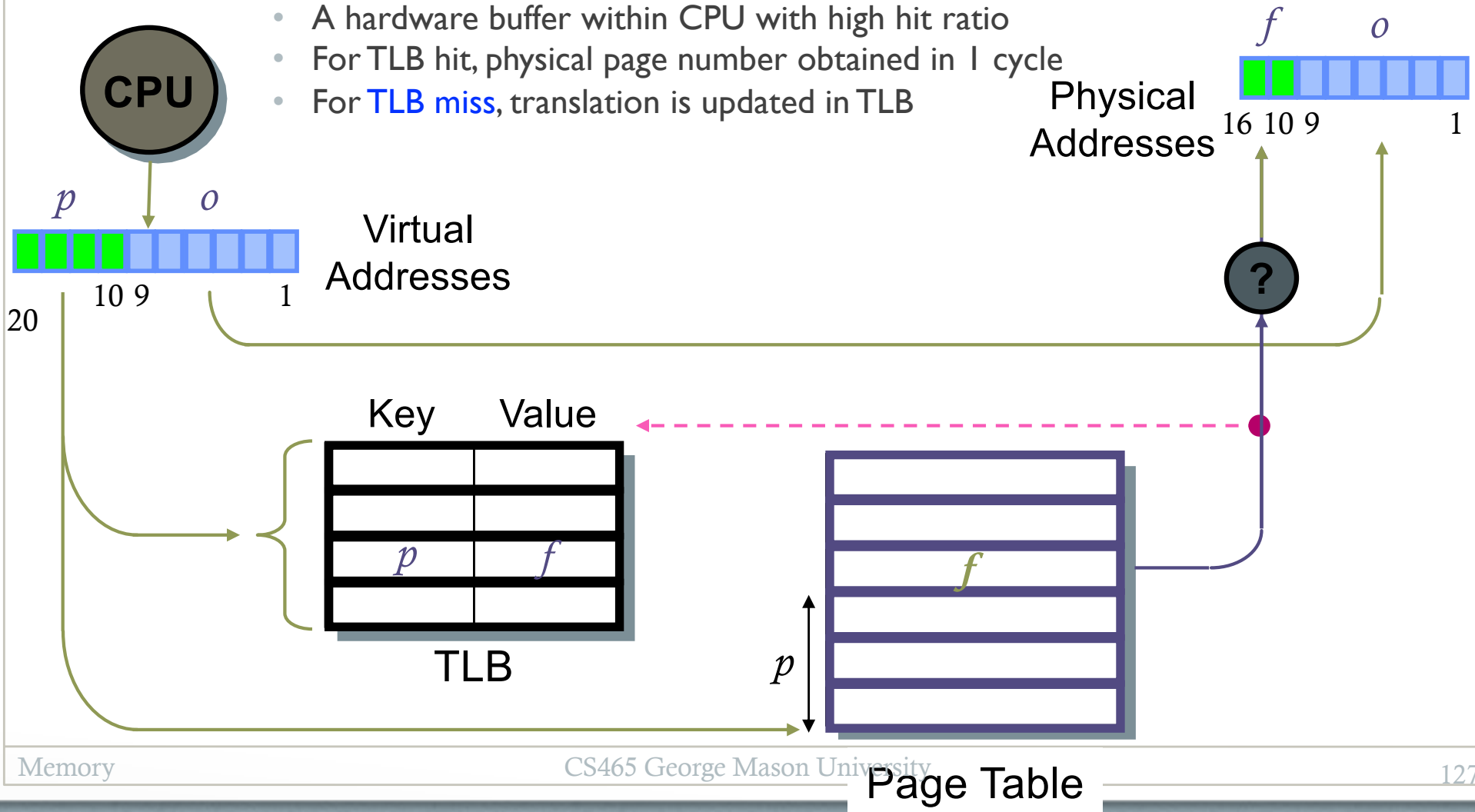
- Cache recently accessed page-to-frame translations in a TLB
  - A hardware buffer within CPU with high hit ratio
  - For TLB hit, physical page number obtained in 1 cycle



TLB(Translation Look-aside Buffer)

Using TLBs to Speedup Address Translation

- Cache recently accessed page-to-frame translations in a TLB
  - A hardware buffer within CPU with high hit ratio
  - For TLB hit, physical page number obtained in 1 cycle
  - For TLB miss, translation is updated in TLB



TLB Organization

Virtual Address	Physical Address	Dirty	Ref	Valid	Access	ASID
0xFA00	0x0003	Y	N	Y	R/W	34
0x0040	0x0010	N	Y	Y	R	0
0x0041	0x0011	N	Y	Y	R	0

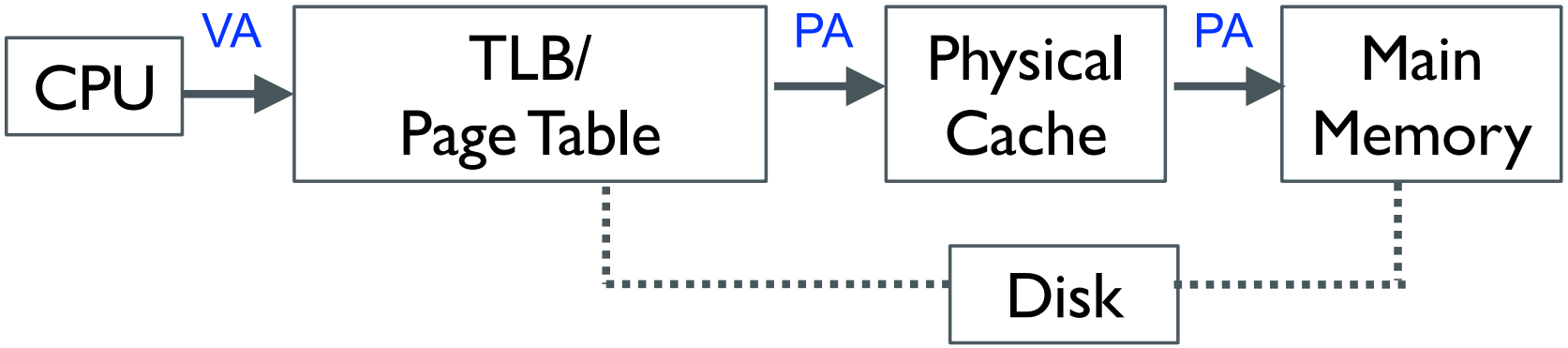
- TLB can be organized in different ways
  - Small fully-associative cache
  - Larger set-associative cache w/ a small associativity
- Include protection
  - Dirty => Page modified (Y/N)?
  - Ref => Page touched (Y/N)?
  - Valid => TLB entry valid (Y/N)?
  - Access => Read? Write?
  - ASID => Which user?

TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

Memory/TLB/Cache Interaction

- Assume physical addresses for cache



  - Translation from virtual address to physical address needed before cache access
  - Best case: hit in TLB + hit in cache
  - Worst case: three misses

- Other variations: virtually addressed cache; virtually indexed physically tagged cache [details in CS471]

Outline

- Cache basics
- Cache performance
- Virtual memory
  - Four memory hierarchy questions
  - Implementation: Page table and TLB
- Concluding remarks
  - A general framework

The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Block Placement

- Determined by associativity
  - Direct mapped (one-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location
- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

- Hardware caches
  - Reduce comparisons to reduce cost
- Virtual memory
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate

Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - More practical variations
    - Performance similar to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep state information
- Virtual memory
  - Typically write-back, given disk write latency

Sources of Misses

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk
- Memory system design is critical for multiprocessors
  - More to discuss next lecture (if time permits)