EEL 5764 Computer Architecture

Sandip Ray
Department of Electrical and Computer Engineering
University of Florida

Lecture 11-12:

- Cache Optimizations (Contd.)
- Introduction to Virtual Memory

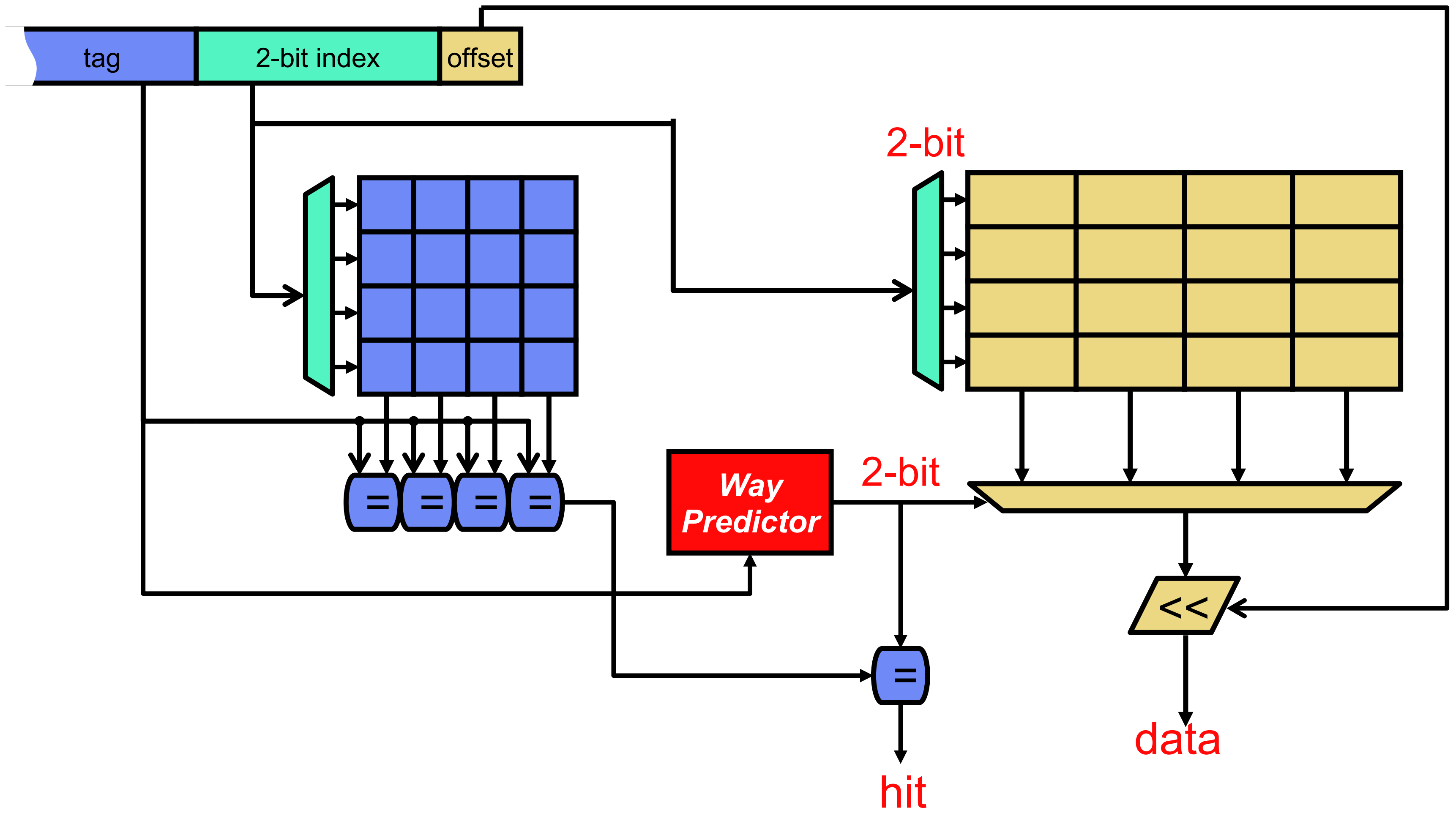
Announcements

- **Mid term 1 is on Oct 22 8:30-9:45pm NEB 202**
  - → Closed book and notes
  - → 1 US-letter sized crib sheet permitted with hand-written notes (both sides)
  - → Electronic calculators permitted
  - → Your smartphone or laptop or any other computing device that can connect to the Internet not permitted
- **For EDGE Students, we will set up Proctor U**
  - → Please watch for additional announcements during lecture or at Canvas
  - → **Exam will be at exactly the same time as regular students**
- **HW1 has been posted. It will be graded.**

Opt 1 – Small and Simple L1 Caches

- Critical timing path in cache hit:
  - → addressing tag memory, then
  - → comparing tags, then
  - → selecting correct set
- Direct-mapped caches can overlap tag comparison and transmission of data
- Lower associativity reduces power because fewer cache lines are accessed. However,
  - → Associativity used to increase size of virtually indexed cache
  - → Higher associativity reduces conflict misses due to multithreading

Opt 2 – Way Prediction



Opt 2 – Way Prediction

- Block associated with prediction (low order tag) bits.
  - → Predicts the next block to be accessed – what locality?
  - → Multiplexer could be set early to select the predicted block, only a single tag comparison
  - → A miss results in checking the other blocks
- Prediction accuracy
  - → > 90% for two-way
  - → > 80% for four-way
  - → I-cache has better accuracy than D-cache
  - → First used on MIPS R10000 in mid-90s
- Extend to predict block as *way selection*
  - → Intends to save power consumption.
  - → Increases mis-prediction penalty

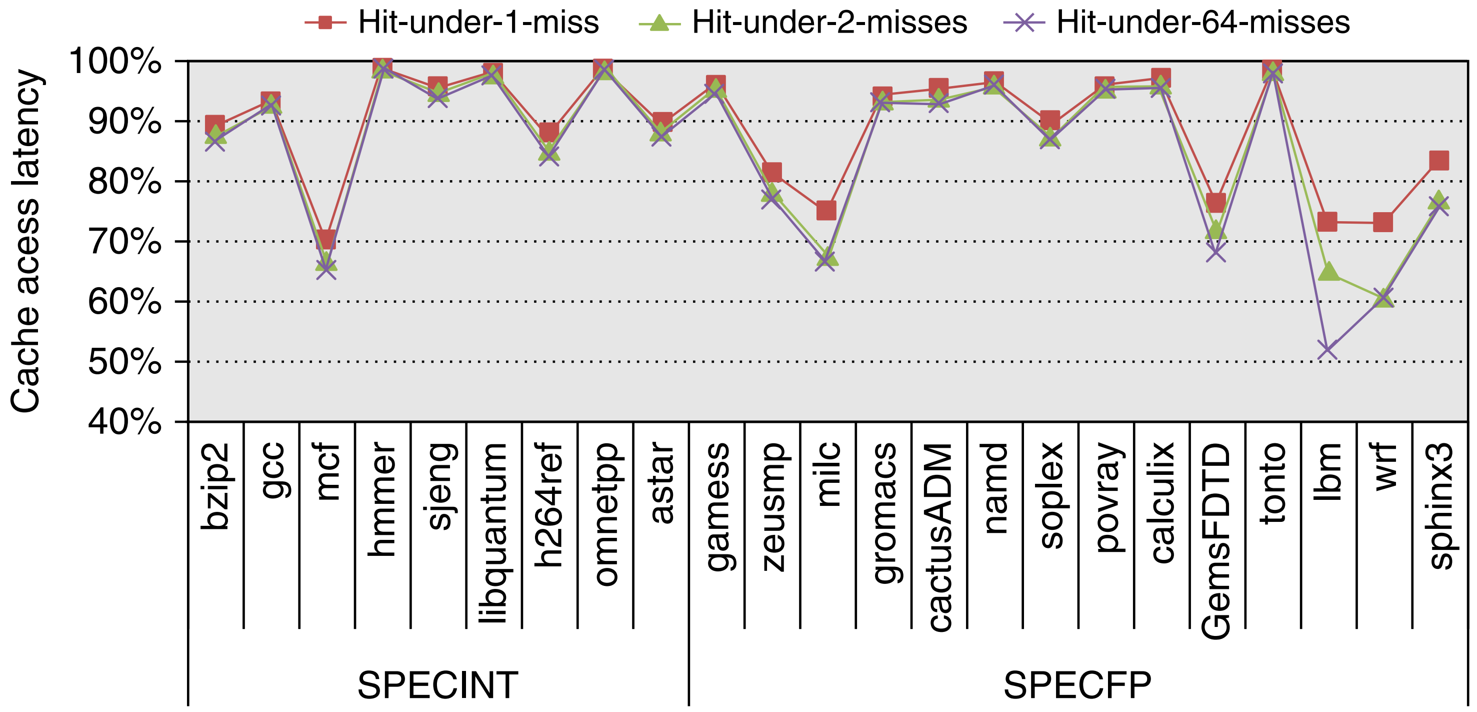
Opt 3 – Pipelining Cache

- Pipeline cache access to improve bandwidth
  - → Faster clock cycle, but slow hit time
  - → Examples:
    - ➤ Pentium: 1 cycle
    - ➤ Pentium Pro – Pentium III: 2 cycles
    - ➤ Pentium 4 – Core i7: 4 cycles

- Increases branch mis-prediction penalty
- Makes it easier to increase associativity
  - → In associative cache, tag compare and data output are serialized

Opt 4 – Nonblocking Caches

- Allow data cache to service hits during a miss
  - → Reduces effective miss penalty
  - → “Hit under miss”
- Extended to “Hit under multiple miss”
  - → L2 must support this

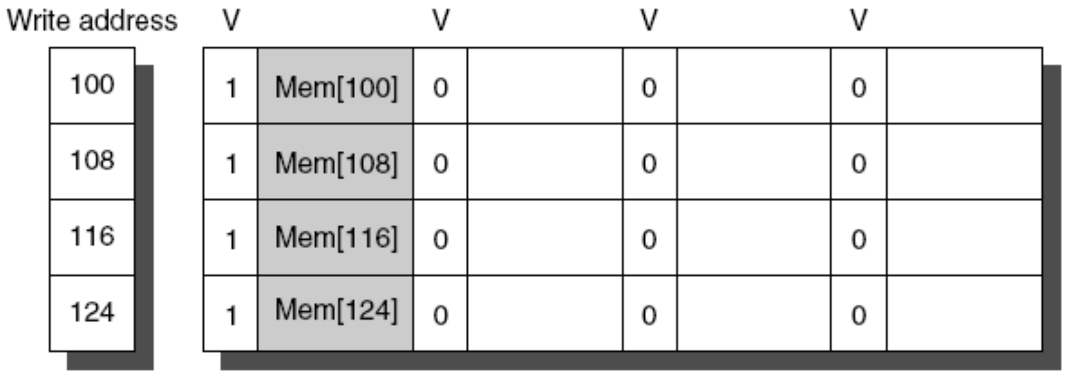


Opt 6 – Critical Word First, Early Restart

- Processor needs to one word in a block
- *Critical word first*
  - → Request missed word from memory first
  - → Send it to the processor as soon as it arrives
- *Early restart*
  - → Request words in normal order
  - → Send missed work to the processor as soon as it arrives
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched
  - → More benefits if block size is larger

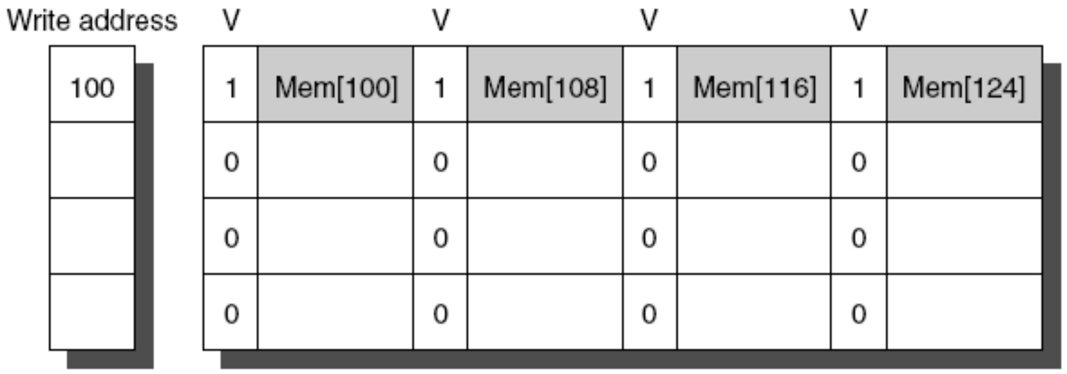
Opt 7 – Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

w/o write merging

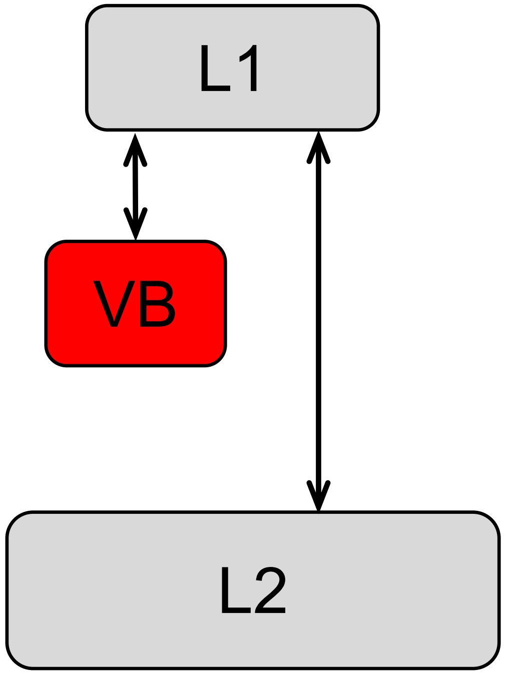


Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

w write merging

Victim Buffer

- A small fully associative cache between L1 and L2
  - →Shared by all sets in L1
- Reduce conflict misses
- On L1 miss, check VB.
  - →Hit -> place block back in L1
- Very effective in practice



Opt 8 – Compiler Optimizations

- Gap between CPU and memory requires SW developer to look at memory hierarchy

```
/* before */
for (j = 0; j < 100; j++)
    for (i = 0; i < 100; i++)
        x[i][j] = 2*x[i][j];
```

Opt 8 – Compiler Optimizations

- Loop Interchange
  → Swap nested loops to access memory in sequential order
  → Expose spatial locality

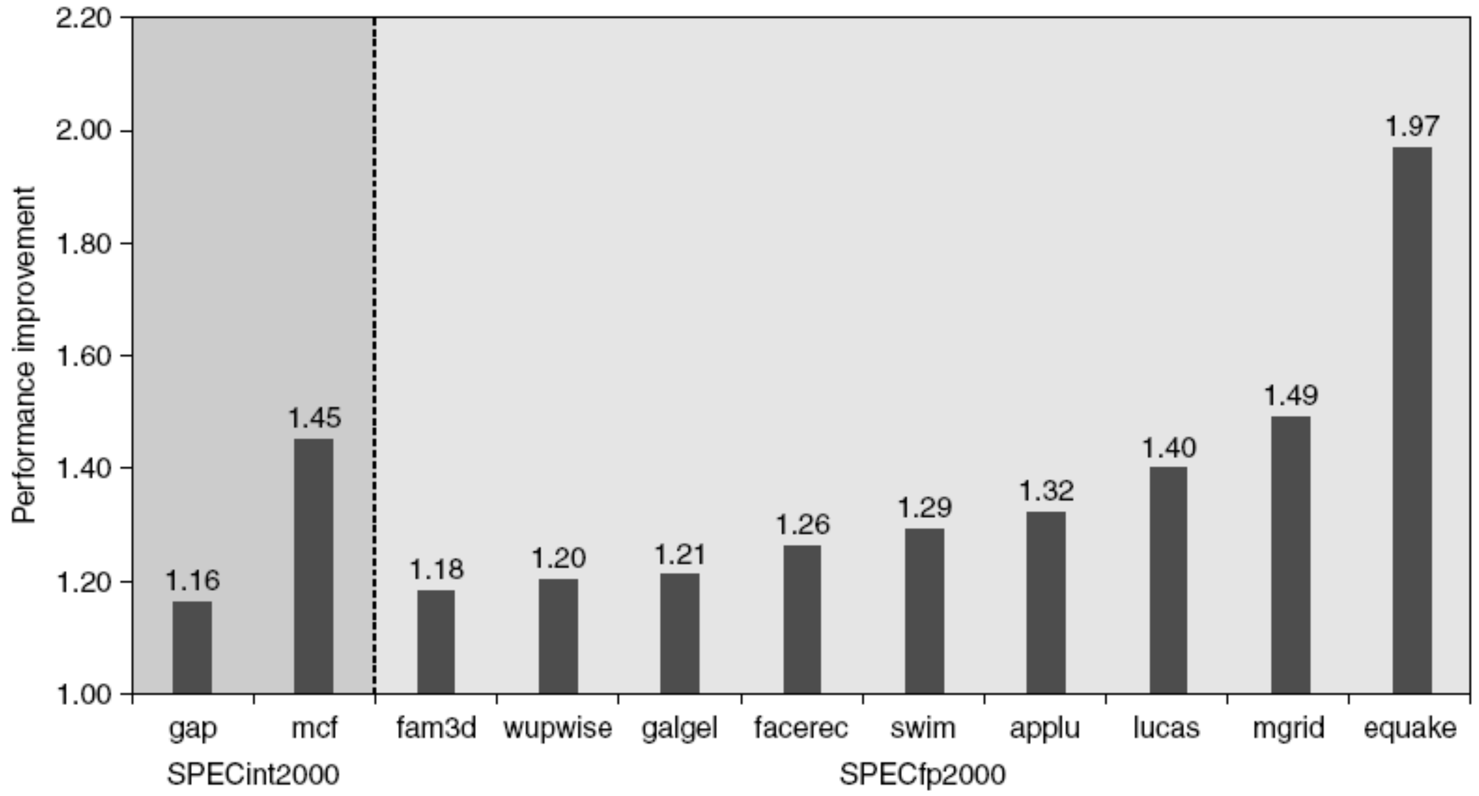
```
/* after*/
for (i = 0; i < 100; i++)
    for (j = 0; j < 100; j++)
        x[i][j] = 2*x[i][j];
```

Opt 8 – Compiler Optimizations

- Blocking
  - → Instead of accessing entire rows or columns, subdivide matrices into blocks
  - → Requires more memory accesses but improves temporal locality of accesses

Opt 9 – Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)
- Can hurt power if prefetched data are not used.



Some results obtained on Pentium 4 w. Pre-fetching

Opt 10 – Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn’t cause exceptions

- Register prefetch
  - → Loads data into register
- Cache prefetch
  - → Loads data into cache

- Combine with loop unrolling and software pipelining

Cache - Summary

- A small and fast buffer between CPU and main memory
- Direct mapped cache

  → shorter hit time, higher miss rate, lower power consumption
- Associative cache

  → longer hit time, lower miss rate, higher power consumption
- Performance evaluation

  → AMAT – average memory access time: maybe misleading

  → CPI with stall cycles due to cache misses: more accurate

Virtual Memories

Why Virtual Memory

- Limited physical memory leads to complications

- Size of a program is larger than main memory size

- Memory demand of multiple programs is larger than main memory size

- *Observation: a program often needs to small part of memory during its execution*

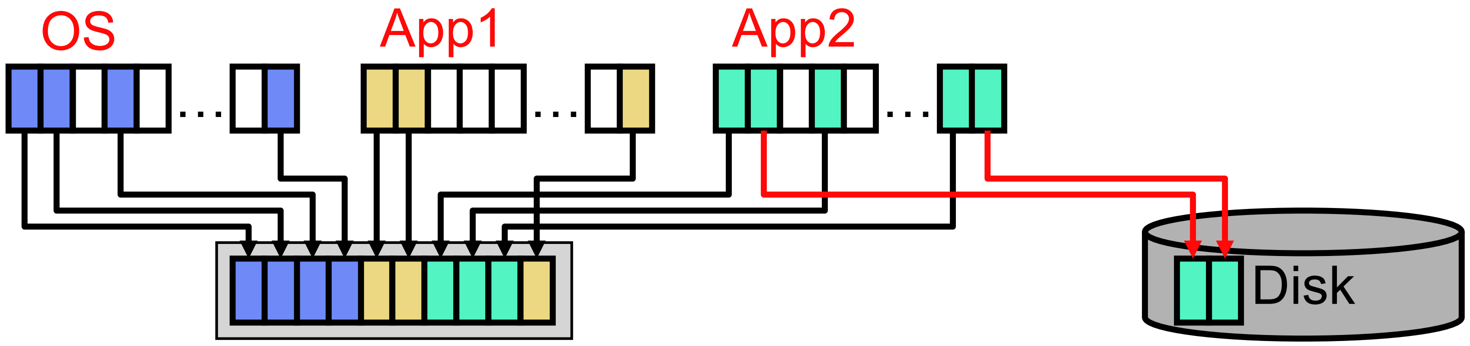
  →Load what is needed into main memory!

Why Virtual Memory

- Programs were divided into pieces and identified pieces that are mutually exclusive
- These pieces were loaded and unloaded under user program control during execution
- Calls between procedures in different modules was leading to overlaying of one module with the other
- Used to be done by hand
  →Significant burden on programmers

Basics of Virtual Memory

- Programs use virtual addresses (VA)
- Memory uses physical addresses (PA)
- VA -> PA at the page granularity
  - →pages can be anywhere in memory
  - →or in disk



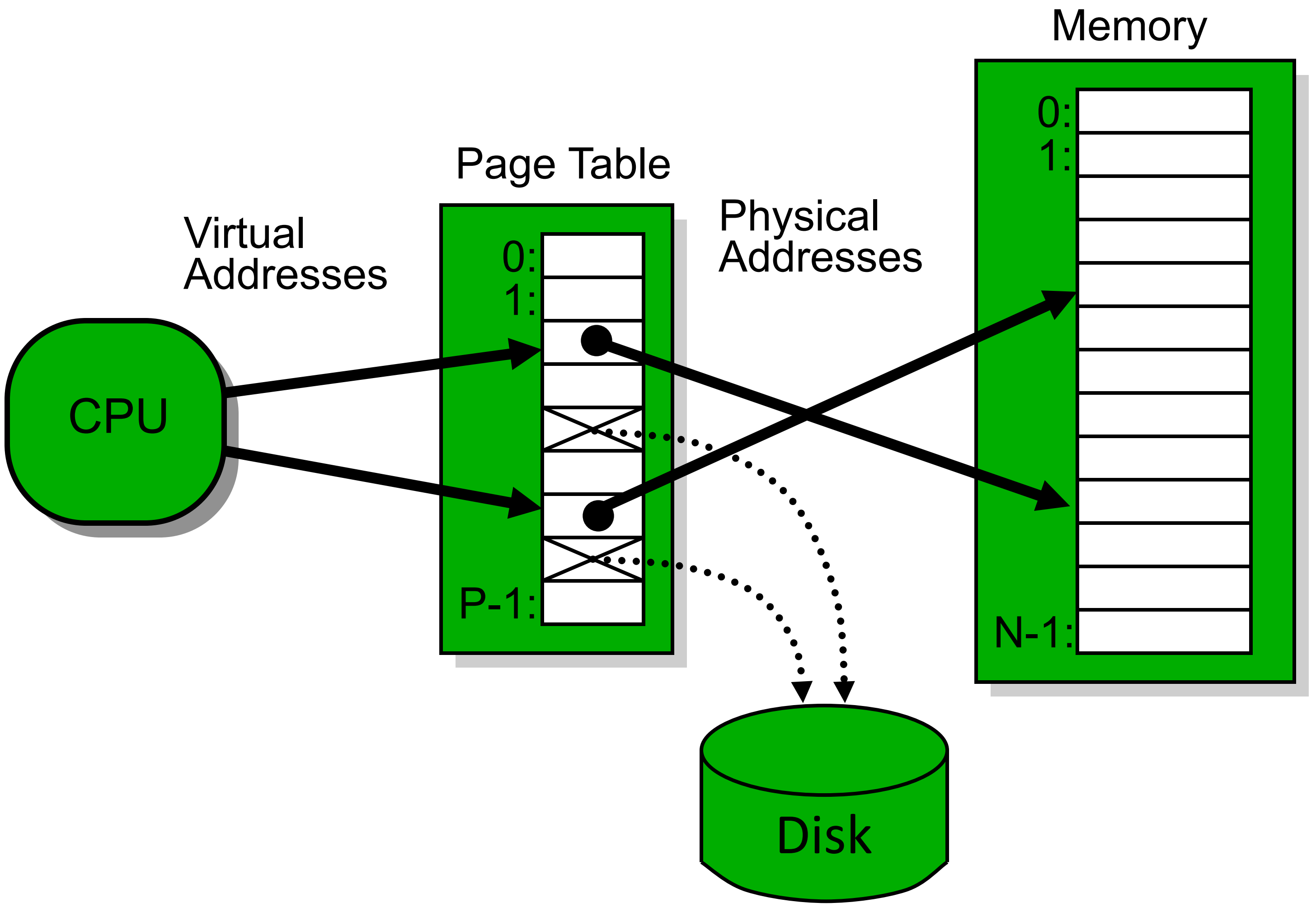
Basics of Virtual Memory

- Use physical DRAM as cache for disk
  - → Address space of a process can exceed physical memory size
  - → Sum of address spaces of multiple processes can exceed physical memory
- Simplify memory management
  - → Multiple processes resident in main memory
    - ➤ Each process with its own address space
  - → Only “active” code and data is actually in memory
    - ➤ Allocate more memory to process as needed
- Provide protection
  - → One process can’t interfere with another
    - ➤ Because they operate in different address spaces
  - → User process cannot access privileged information
    - ➤ Different sections of address space have different permissions

Basic Issues

- Shares same basic concepts of cache memory but different terminology
- Issues
  - → Mapping: translation of virtual to physical address
  - → Management: controlled sharing and protection. Protection in multi-programming environment
- Mapping techniques
  - → Paging (demand paging)
  - → Segmentation

A Simplified View of Virtual Memory



Cache vs Virtual Memory

Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–3 clock cycles	100–200 clock cycles
Miss penalty	8–200 clock cycles	1,000,000–10,000,000 clock cycles
(access time)	(6–160 clock cycles)	(800,000–8,000,000 clock cycles)
(transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.1–10%	0.00001–0.001%
Address mapping	25–45-bit physical address to 14–20-bit cache address	32–64-bit virtual address to 25–45-bit physical address

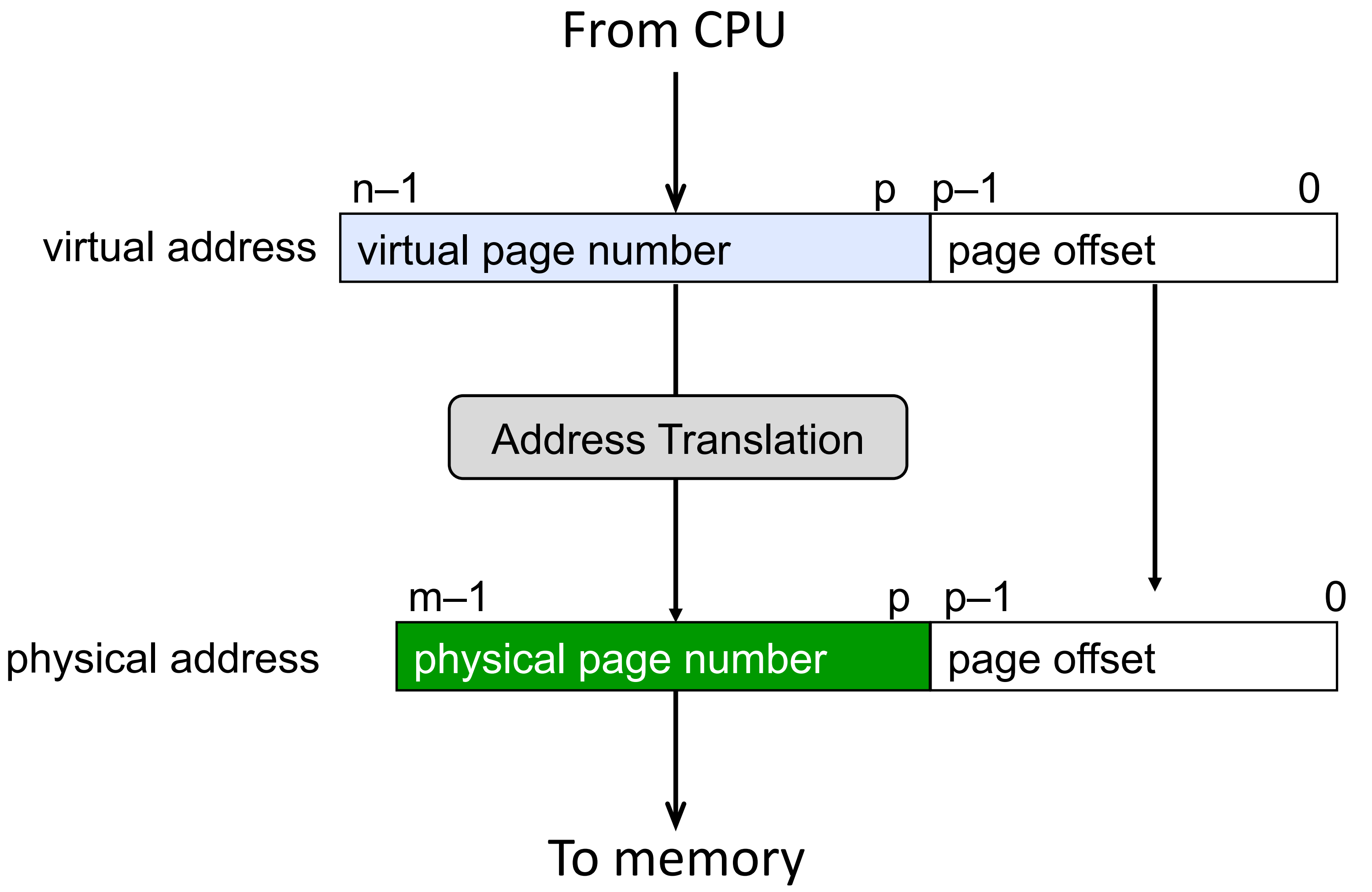
Cache vs Virtual Memory

- Terminology
  - →Block → *page*
  - →Cache miss → *page fault*
- Replacement on cache memory misses by hardware whereas virtual memory replacement is by OS
- VM can have fixed or variable size blocks
  - →page vs segmentation: they both have pros and cons

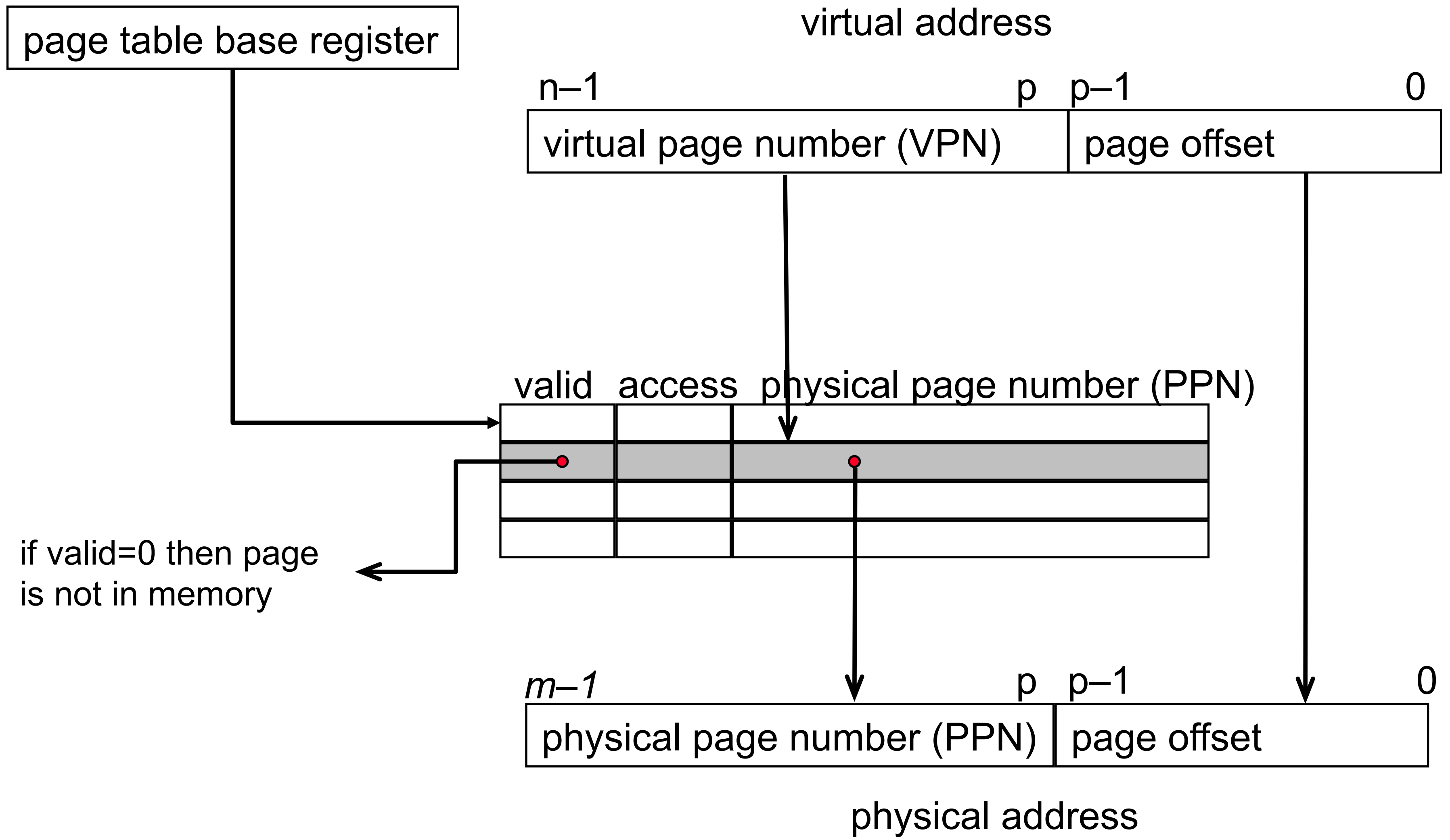
Virtual Memory Design Issues

- Page size should be large enough to try to amortize high access time
  - →Typical size: 4KB – 16KB
- Reducing page fault rate is important
  - →Fully associative placement of pages in memory
- Page faults not handled by hardware
  - →OS can afford to use clever algorithm for page replacement to reduce page fault rate
- Write through approach is too expensive
  - →Write back approach is always used

Virtual Memory Address Translation

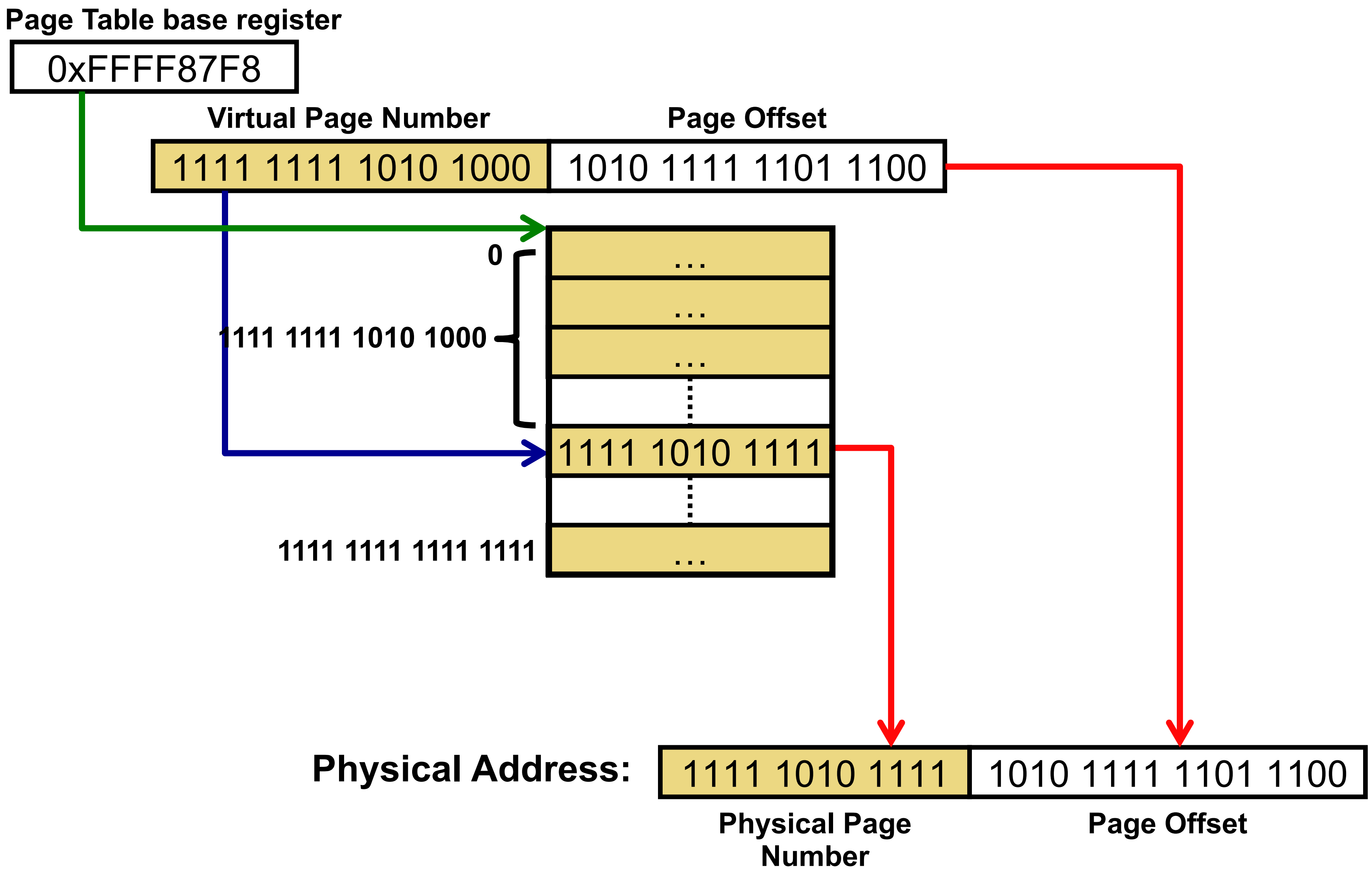


Address Translation


page table base register
virtual address
n–1
p
p–1
0
virtual page number (VPN)
page offset
valid
access
physical page number (PPN)
if valid=0 then page is not in memory
m–1
p
p–1
0
physical page number (PPN)
page offset
physical address

- Each process has its own page table.

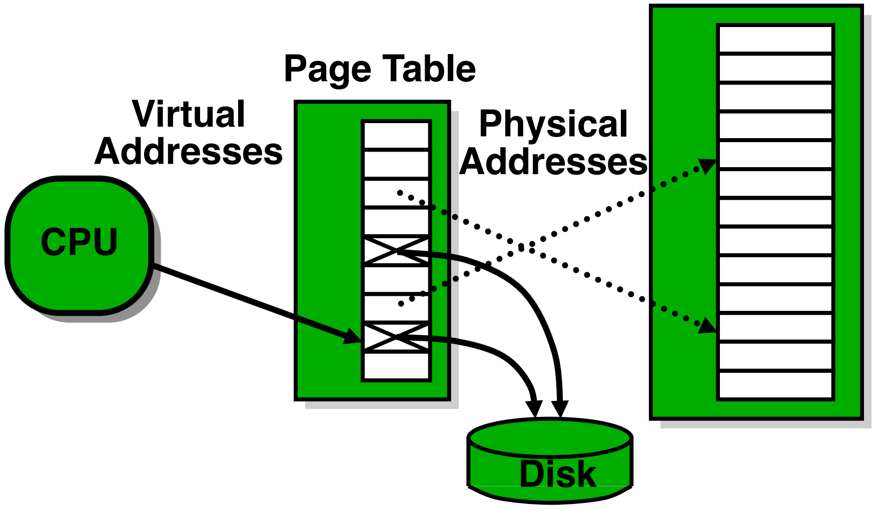
Address Translation - Example



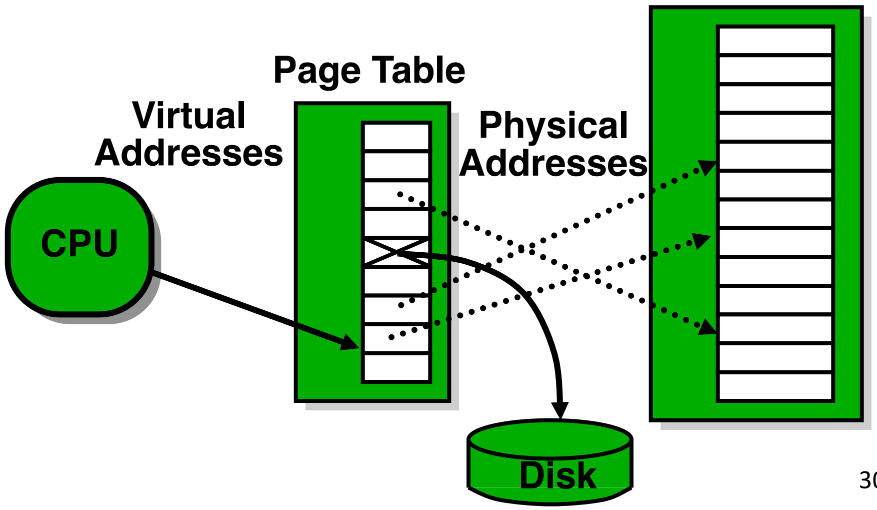
Page Faults

- What if object is on disk rather than in memory?
  - → Page table entry indicates virtual address not in memory
  - → OS exception handler invoked to move data from disk into memory – VM and Multiprogramming are symbiotic
    - ➤ Current process suspends, others can resume
    - ➤ OS has full control over placement, etc.

Before fault



After fault


Memory
Page Table
Virtual Addresses
Physical Addresses
CPU
Disk