CS356: Discussion #10

Cache Lab and Virtual Memory

Marco Paolieri (paolieri@usc.edu)

Illustrations from CS:APP3e textbook

USC University of Southern California

Schedule: Exams and Assignments

- Week 1: Binary Representation **HW0**
- Week 2: Integer Operations
- Week 3: Floating-Point Operations **Data Lab 1**
- Week 4: Assembly (Arithmetic Instruction)
- Week 5: Assembly (Debugging with GDB) **Data Lab 2**
- Week 6: Assembly (Function Calls)
- Week 7: **Bomb Lab** (Oct. 1), **Exam I** (Oct. 4), Security Vulnerabilities
- Week 8: Memory Organization
- Week 9: Caching **Attack Lab**
- **Week 10: Virtual Memory**
- Week 11: Dynamic Memory Allocation and Linking
- Week 12: Processor Organization and **Exam II** (Nov. 8) **Cache Lab**
- Week 13: Processor Organization
- Week 14: Code Optimization and **Thanksgiving**
- Week 15: Cache Coherency and Review **Allocation Lab**
- Week 16: Study Days and **Final** (Dec. 6)

Last Time: Direct-Mapping Cache Simulation

Address breakdown

- C1 has no block offset, 3-bit set address
- C2 has 1-bit block offset, 2-bit set address
- C3 has 2-bit block offset, 1-bit set address

How to run a trace: extract set address (3, 2, 1 bits) from LSB; on miss, load (1, 2, 4) bytes.

Running C3:

- **Get 1: miss.** Put bytes 0-3 in bucket 0.
- **Get 134: miss.** Put 132-135 in bucket 1.
- **Get 212: miss.** Put 212-215 in bucket 1.
- **Get 1: hit.**
- **Get 135: miss.** Put 132-135 in bucket 1.
- **Get 213: miss.** Put 212-215 in bucket 1.
- **Get 162: miss.** Put 160-163 in bucket 0.
- **Get 161: hit.**

Trace

MEM	LSB	C1	C2	C3
1	0000 0001	1m	0m	0m
134	1000 0110	6m	3m	1m
212	1101 0100	4m	2m	1m
1	0000 0001	1h	0h	0h
135	1000 0111	7m	3h	1m
213	1101 0101	5m	2h	1m
162	1010 0010	2m	1m	0m
161	1010 0001	1m	0m	0h
2	0000 0010	2m	1m	0m
44	0010 1100	4m	2m	1m
41	0010 1001	1m	0m	0m
221	1101 1101	5m	2m	1m

```
m_rate: 11/12 9/12 10/12
```

Cache Lab

Goal

- To write a small C simulator of caching strategies.
- Expect about 200-300 lines of code.
- Starting point in your repository.

Traces

- The `traces` directory contains program traces generated by `valgrind`
- The format of each line is: `<operation> <address>,<size>`
  For example: “`I 0400d7d4,8`” “`M 0421c7f0,4`” “`L 04f6b868,8`”
- Operations
  - Instruction load: `I` (ignore these)
  - Data load: `␣L` (hit, miss, miss/eviction)
  - Data store: `␣S` (hit, miss, miss/eviction)
  - Data modify: `␣M` (load+store: hit/hit, miss/hit, miss/eviction/hit)

http://bytes.usc.edu/cs356/assignments/cachelab.pdf

Reference Cache Simulator

```
./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile> (-L|-F)
```

`-h` print usage information

`-v` display trace information

`-s <s>` select the number of set bits (i.e., use $S = 2^s$ sets)

`-E <E>` select the number of lines per set (associativity)

`-b <b>` select the number of block bits (i.e., use $B = 2^b$ bytes / block)

`-t <tracefile>` select a trace

`-L` select the LRU policy

`-F` select the FIFO policy

```
$ ./csim-ref -s 4 -E 1 -b 4 -L -t traces/yi.trace
hits:4 misses:5 evictions:3

$ ./csim-ref -s 4 -E 1 -b 4 -L -t traces/yi.trace -v
 L 10,1 miss
 M 20,1 miss hit
 L 210,1 miss eviction
 M 12,1 miss eviction hit
```

Your Simulator

Fill in the `csim.c` file to:

- Accept the same command-line options.
- Produce identical output.

Rules

- Include name and username in the header.
- Use only C code (must compile with `gcc -std=c99`)
- Use `malloc` to allocate data structures for arbitrary `s`, `E`, `b`
- Implement both LRU and FIFO policies.
- Ignore instruction cache accesses (starting with `I`).
- Assume that memory accesses never cross block boundaries:
  ⇒ Ignore request sizes.
- At the end of your main function, call:
  `printSummary(hit_count, miss_count, eviction_count)`

Evaluation

16 test cases:

- 8 for LRU
- 8 for FIFO

Two case are worth **6 points**, others **3 points**.
⇒ Total of 2 × 6 + 14 × 3 = **54 points**.

You only need to output the **correct number of cache hits, misses, evictions**.

- Each gives you 1/3 of the points for that test.
- You can run **`csim-ref -v`** to check the expected behavior.
- Start from small traces such as `traces/dave.traces`
- Use the **`getopt`** library to parse command-line arguments.
  - `int s = atoi(arg_str); int S = pow(2, s);`

Compile and autograde using **`test-csim`**.

Running `csim-test`

```
$ make; ./test-csim
EP: LRU                    Your simulator           Reference simulator
Points (s,E,b)      Hits   Misses   Evicts      Hits   Misses   Evicts
     3 (1,1,1)         9        8        6         9        8        6  traces/yi2.trace
     3 (4,2,4)         4        5        2         4        5        2  traces/yi.trace
     3 (2,1,4)         2        3        1         2        3        1  traces/dave.trace
     3 (2,1,3)       167       71       67       167       71       67  traces/trans.trace
     3 (2,2,3)       201       37       29       201       37       29  traces/trans.trace
     3 (2,4,3)       212       26       10       212       26       10  traces/trans.trace
     3 (5,1,5)       231        7        0       231        7        0  traces/trans.trace
     6 (5,1,5)    265189    21775    21743    265189    21775    21743  traces/long.trace
EP: FIFO                   Your simulator           Reference simulator
Points (s,E,b)      Hits   Misses   Evicts      Hits   Misses   Evicts
     3 (4,2,4)         7        5        2         7        5        2  traces/fifo_s1.trace
     3 (4,2,4)        11        7        3        11        7        3  traces/fifo_s2.trace
     3 (4,4,4)         6       11        7         6       11        7  traces/fifo_s3.trace
     3 (5,2,2)        59      354      298        59      354      298  traces/fifo_m1.trace
     3 (3,4,2)        51      362      330        51      362      330  traces/fifo_m1.trace
     3 (5,2,2)       191      188      142       191      188      142  traces/fifo_m2.trace
     3 (3,4,2)       164      215      184       164      215      184  traces/fifo_m2.trace
     6 (4,2,4)    263447    28255    28223    263447    28255    28223  traces/fifo_l.trace
TEST_CSIM_RESULTS=54
```

Problems

- How to parse the input traces?
  - **`fopen`** (open a file), **`fgets`** (read a line), **`sscanf`** (parse a line), **`fclose`**
- How to represent the cache? How to allocate memory for any s, E, b?
  - Cache = **S sets**
  - Each set = **E cache lines**
  - Use **`malloc`** and **`free`**
- What needs to be stored in a cache line?
  - Valid bit, tag, and what else?
  - How to keep track of statistics for **LRU** and **FIFO** policies?
- How to retrieve data at a memory address?
  - How to extract tag / set / block bits from an input address?
  - How to select the correct set? And how to look for a hit?
  - What to update in case of hit (in addition to hit counter)?
  - What to do in case of miss?
- Useful: Print the content of the cache after each request in a trace

Virtual Memory and Address Translation

One more level of indirection

- Compile programs with **virtual addresses**
- Use a Memory Management Unit (MMU) to convert them into **physical addresses** during each memory access... must be **fast**!

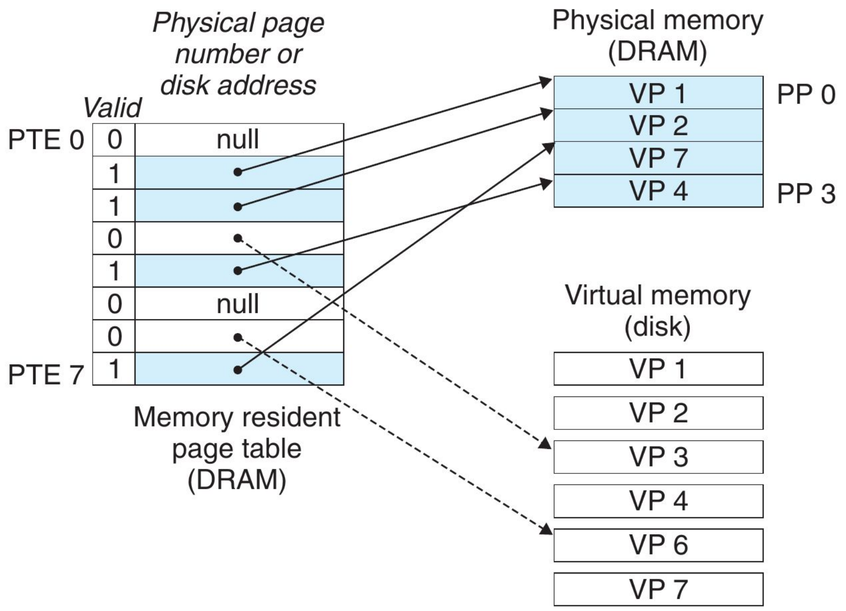
Why?

- To use **main memory as a cache for disk memory**, when main memory is limited and many memory-intensive processes are running.
- To **isolate different processes**, with fine-grained access control and sharing of memory blocks, map I/O devices, and more.

CPU and OS cooperate to make VM very fast

- Latency: 1 ns (L1), 100 ns (RAM), 50 µs (SSD), 20 ms (magnetic disk)
- Page size should be fairly large (4 kB)
- Translation Lookaside Buffer (TLB) to cache address translations

Page Table: Mapping Virtual Addresses



Physical memory acts like a cache for virtual memory pages.

- Fully associative, write-back
- Same-size pages / frames
- **Page table** keeps track of pages and their location: **look-up table**, not tag+frame#
- **Read by HW, updated by OS**

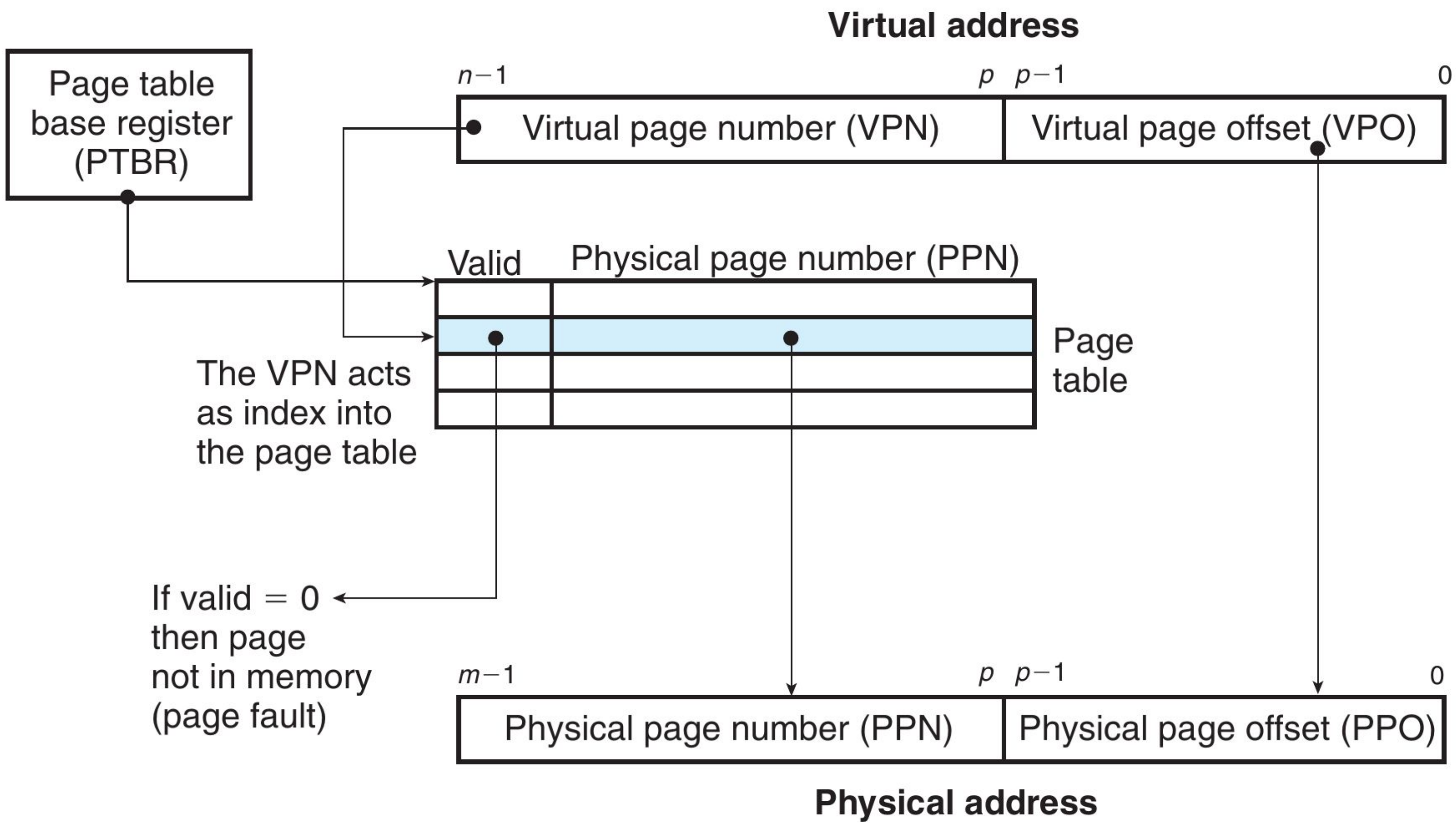
Idea: break virtual and physical address spaces in blocks with same size (4 kB)

- **Pages** of virtual address space
- **Physical block frame** of physical address space

Load virtual pages into physical block frames when needed.

- CPU generates a “page fault exception”
- OS loads the page into a frame (possibly evicting another)

Single-Level Page Table: **PTBR**[VPN] | VPO



Virtual address
n−1
p
p−1
0
Page table base register (PTBR)
Virtual page number (VPN)
Virtual page offset (VPO)
Valid
Physical page number (PPN)
Page table
The VPN acts as index into the page table
If valid = 0 then page not in memory (page fault)
m−1
p
p−1
0
Physical page number (PPN)
Physical page offset (PPO)
Physical address

Example: 32 bit virtual address, 4 kB pages ⇒ 20 bit VPN, **1M page table entries**

- Only 1 GB of physical memory ⇒ 18 bit PPN (translated address is 00...)

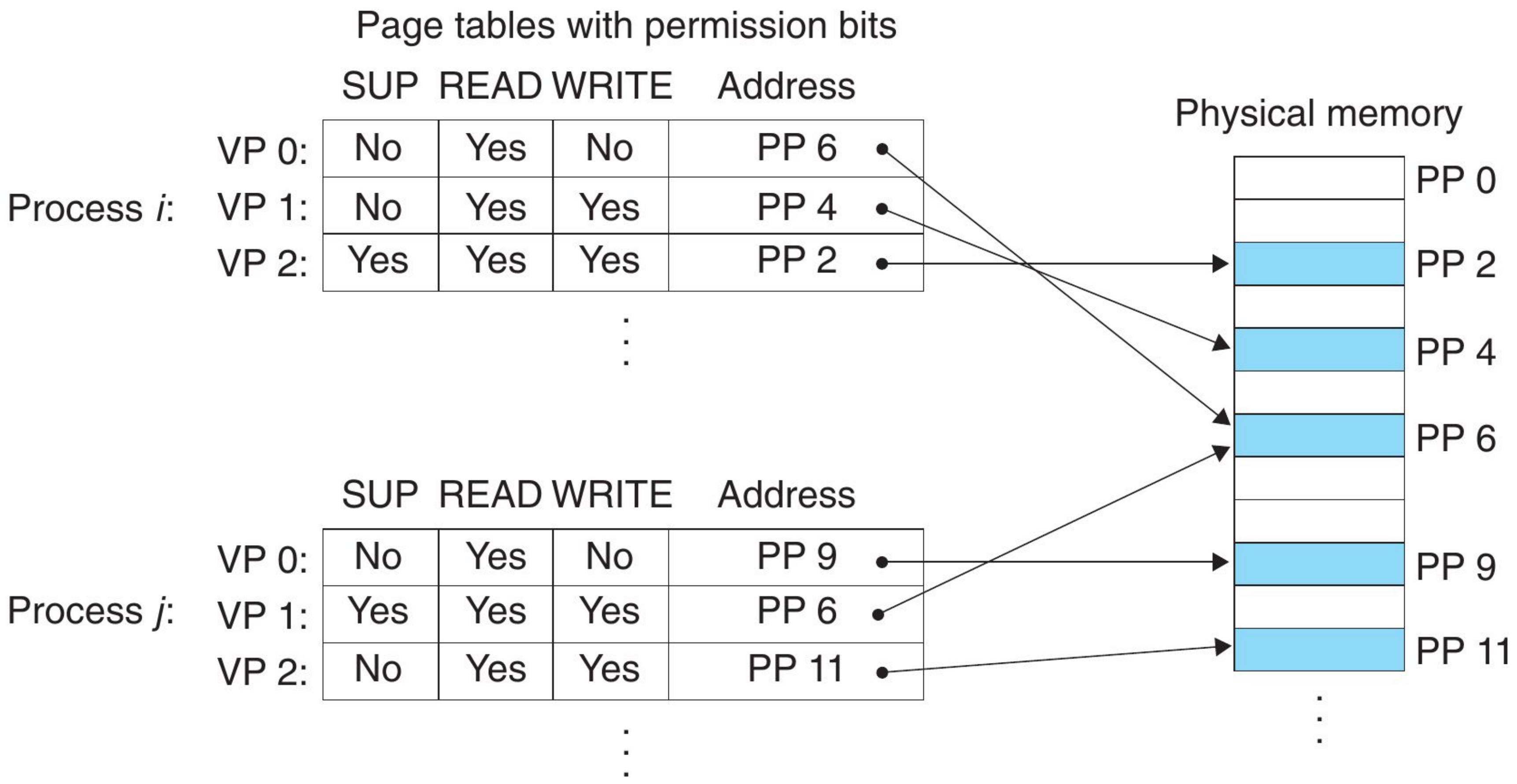
Example: Single-Level Page Table

Index	Valid	PPN
0	0	0x0E
1	1	**0x1E**
2	1	0x16
3	1	**0x06**
4	0	0x0B
5	1	0x1F
6	0	0x15
7	0	0x0A

8-bit virtual addresses, 10-bit physical addresses, 32-byte pages

- Physical address of virtual address **0x2D**? **00101101** => **0 0011 1100 1101**
- Physical address of virtual address **0x7A**? **01111010** => **0 0000 1101 1010**
- Physical address of virtual address **0xEF**? **11101111** =>
- Physical address of virtual address **0xA8**? **10101000** => **0 1000**

A page table for each process



Process *i*:

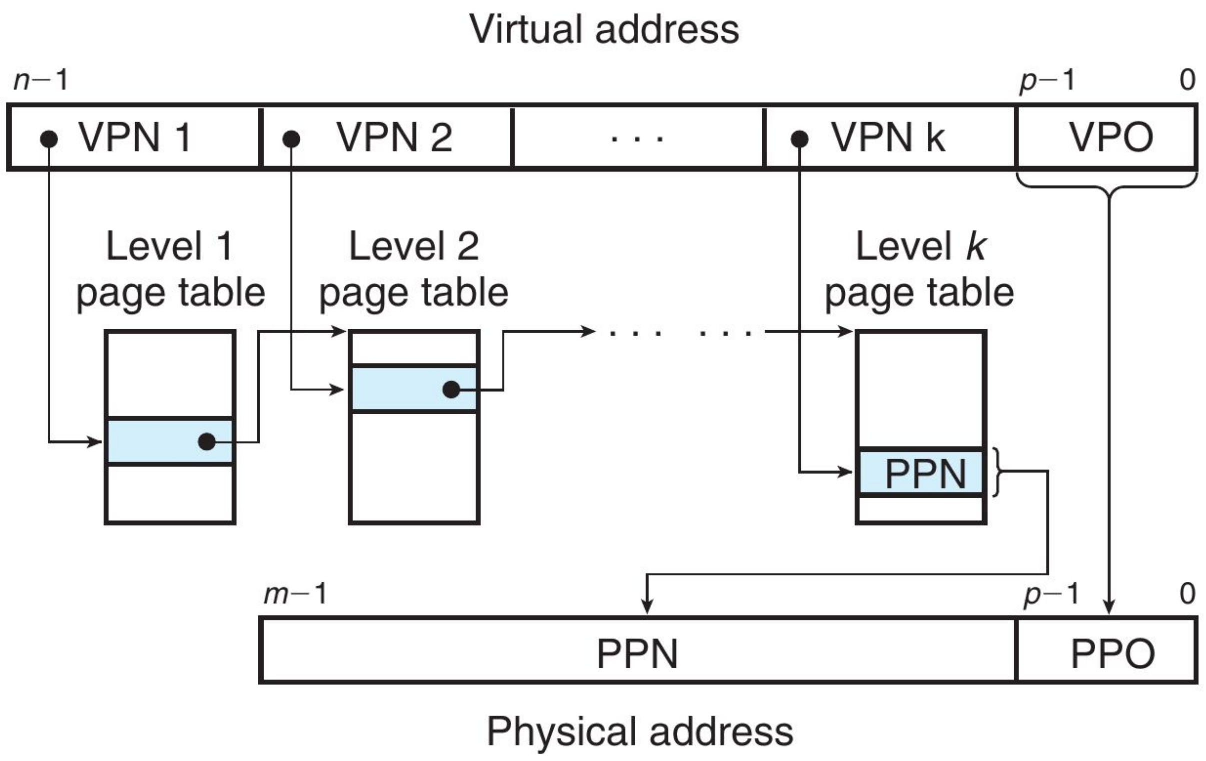
	SUP	READ	WRITE	Address
VP 0:	No	Yes	No	PP 6
VP 1:	No	Yes	Yes	PP 4
VP 2:	Yes	Yes	Yes	PP 2
⋮				

Process *j*:

	SUP	READ	WRITE	Address
VP 0:	No	Yes	No	PP 9
VP 1:	Yes	Yes	Yes	PP 6
VP 2:	No	Yes	Yes	PP 11
⋮				

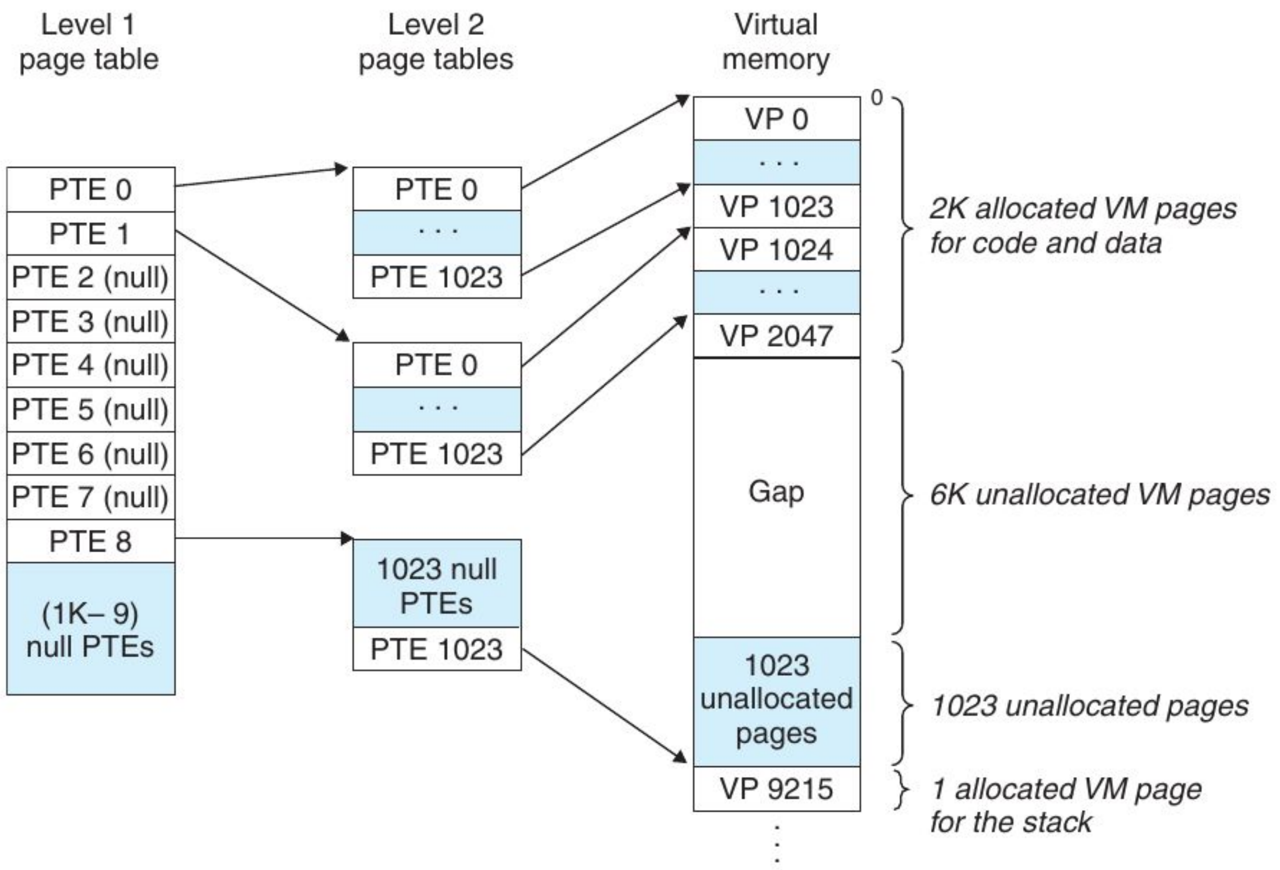
Page-level memory protection and sharing (page tables in kernel memory).
Context switch: load PTBR from Global Descriptor Table (GDT) to CR3 register.

Multi-Level Page Table: More indirections



The virtual address space can be very large for a single process.
⇒ Most of the page table entries are not used
⇒ **Idea:** use a **page directory** where entries point to next-level tables (if present)
⇒ Each level contains base of next table (if present), **last level contains PPN**

Multi-Level Page Table: Space savings



Drawback: more memory accesses, more latency...

Problem: Three-Level Page Table

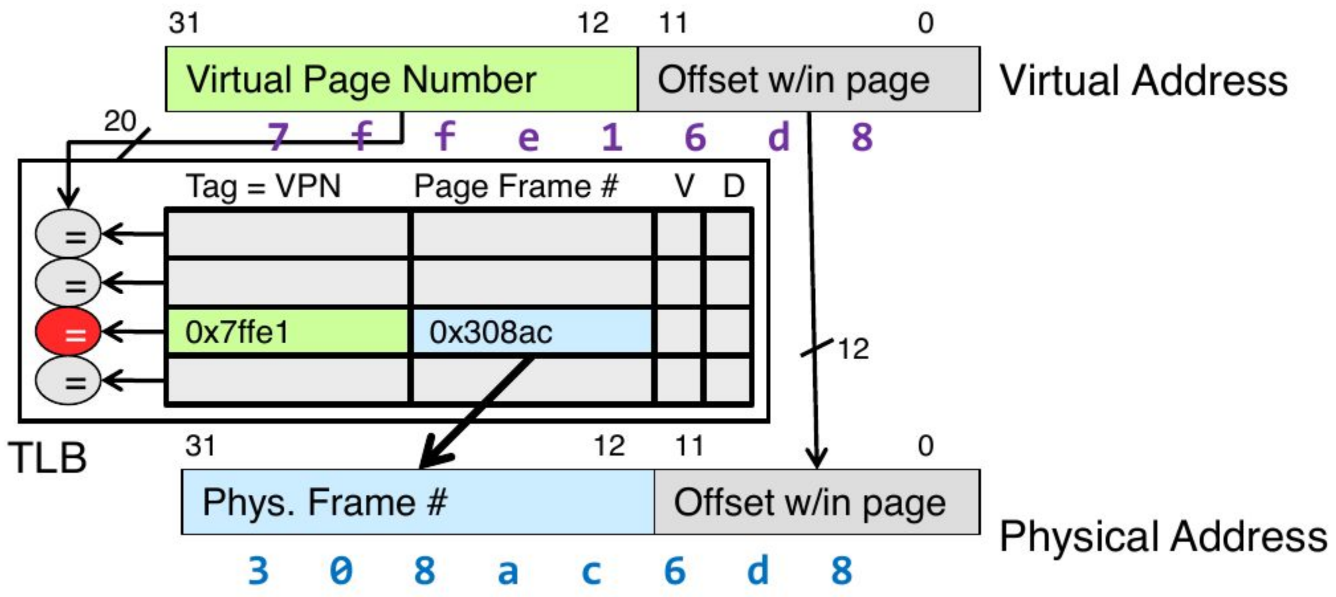
Consider a 3-level VM system with:

- 36-bit physical address space
- 32-bit virtual address space
- 4 kB pages
- Page tables implemented as look-up tables
- 256 entries for page directory
- 64 entries in second-level page table

Find out:

- The layout of virtual addresses (1st / 2nd / 3rd table offset, page offset)
- The number of entries in third-level page table
- The size of each page table (assume 4 bytes for each entry)
- Minimum size of entries of third page table?

Translation Lookaside Buffer

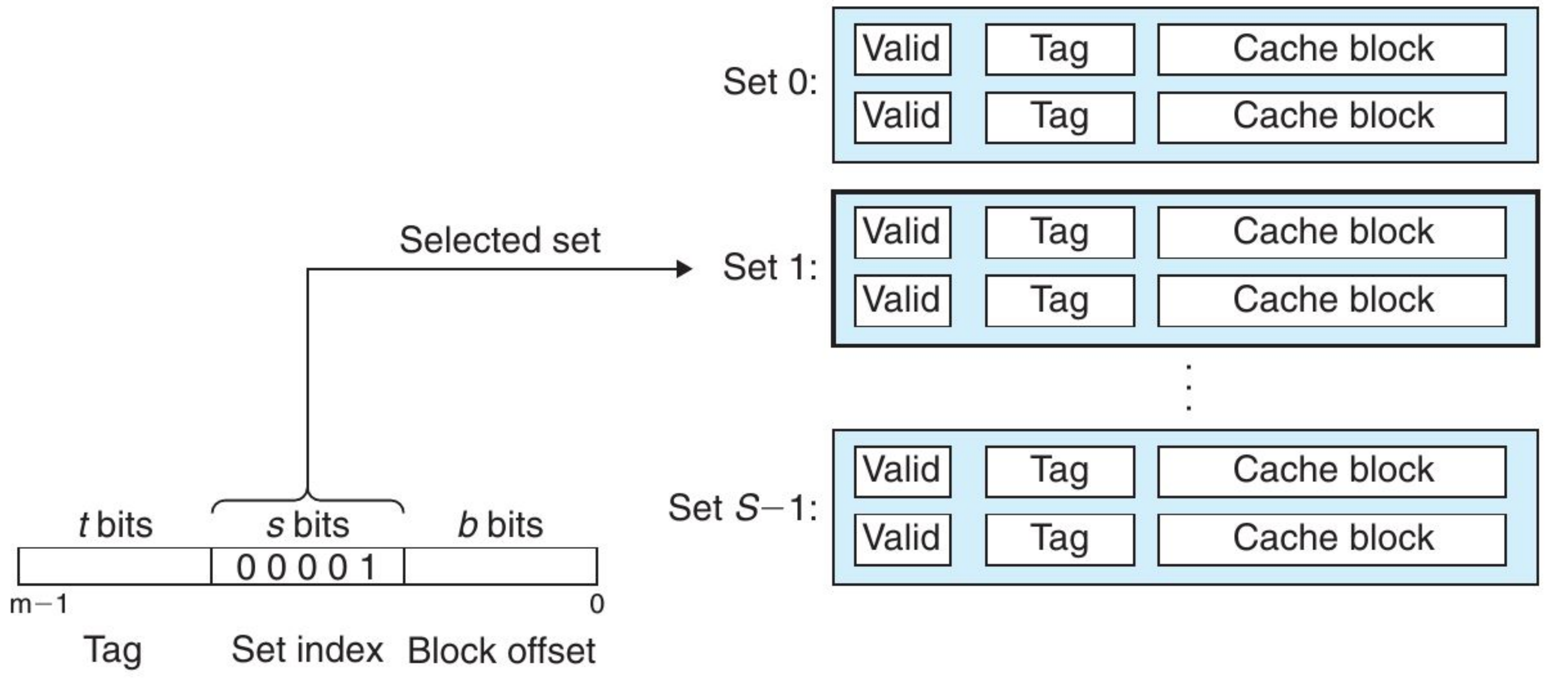


A *k*-level page table requires *k* memory accesses in the worse case.
Idea: cache address mappings inside the CPU (10 ns hit time).

- **VPN is the cache tag, PPN is the entire cache block**
- High degree of associativity (4-way or fully-associative: low miss rate)
- Usually smaller than data cache (fast lookup, low hit time)

Average Access Time = (**Hit Time**) + (**Miss Rate**) × (**Miss Penalty**)

K-way Set Associative Caches (1 < *K* < *C*/*B*)



Differences of TLB

- No block offset: cache block is the entire PPN
- TLB is usually smaller/faster than caches

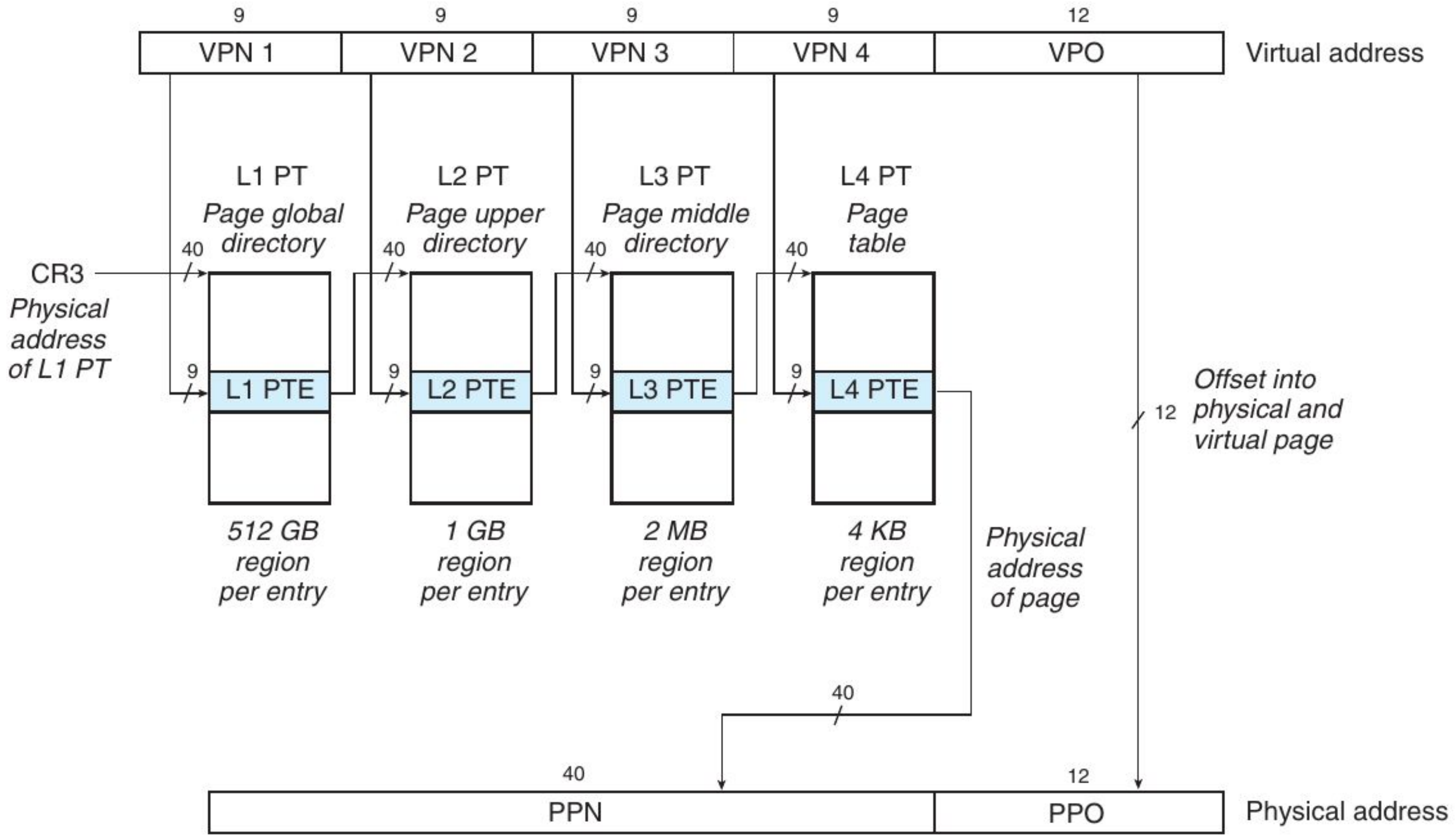
Example: 2-way set associative TLB

Index	Valid	Tag	PPN
0	1	0x13	**0x30**
	0	0x34	0x58
1	0	0x1F	0x80
	1	0x2A	0x72
2	1	**0x1F**	0x95
	0	0x20	0xAA
3	1	0x3F	0x20
	0	0x3E	0xFF

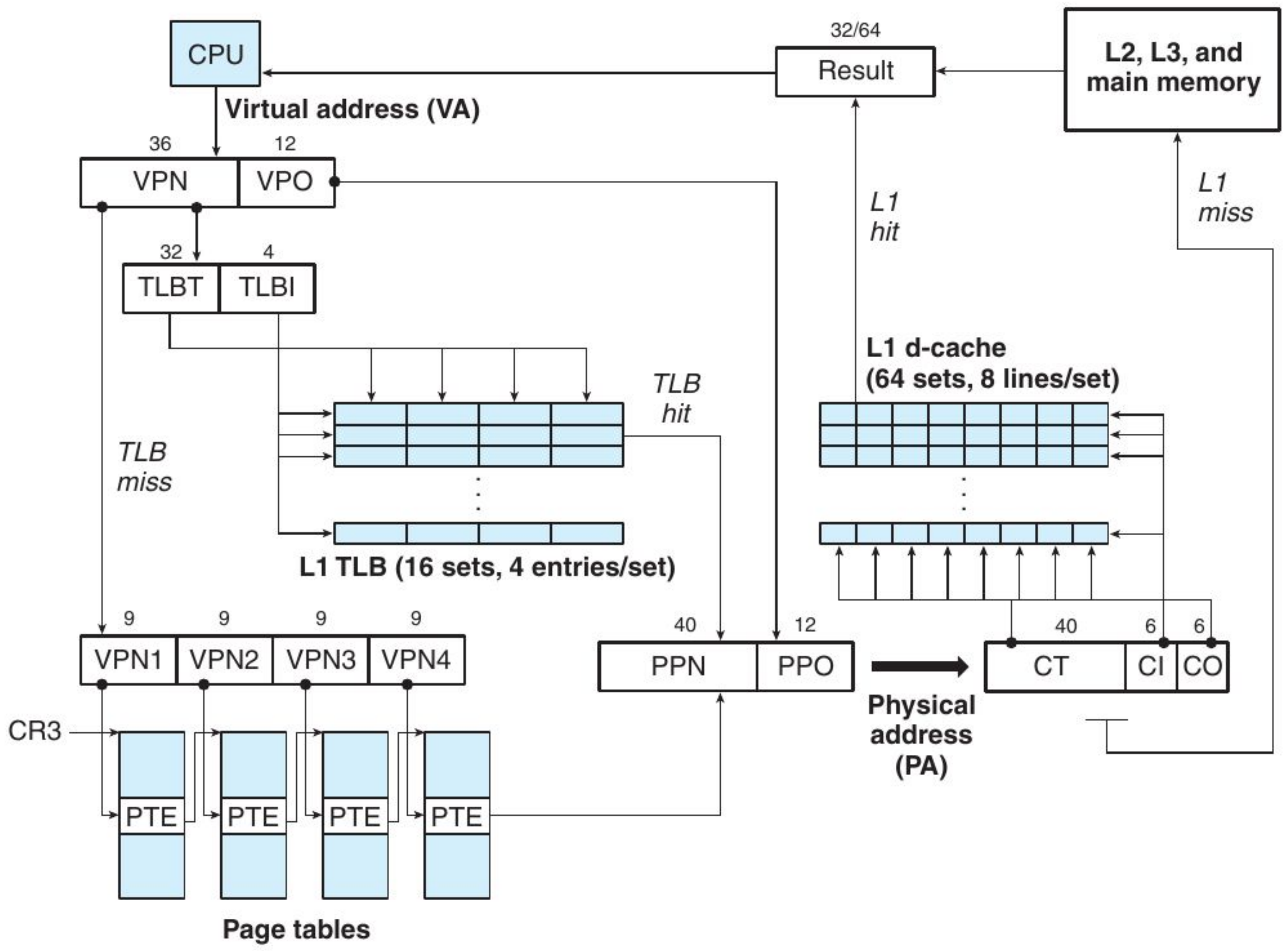
16-bit virtual and physical addresses, 256-byte pages

- Physical address of virtual address **`0x7E85`** `== 0111 1110 1000 0101`
- Virtual address of physical address **`0x3020`** `== 0011 0000 0010 0000`

Intel Core i7: Address Translation



Intel Core i7: TLB and translation before L1



More Exercises

http://bytes.usc.edu/cs356/docs/cs356_vm_sol.pdf

Virtual Memory

32-bit virtual addresses, 36-bit physical addresses, 16 kB pages

- Bits of page offset? VPN bits? PPN bits?
- Number of pages in virtual and physical memory?
- Page table size with 4 byte entries?
- VPN bits breakdown for 3-level (32 / 64 / unknown)-entries?
  - Worst-case size with 4 byte entries and 10 pages in use?
- 4-way set associative TLB with 128 total entries
  - VPN bits mapping to tag / set / page offset?