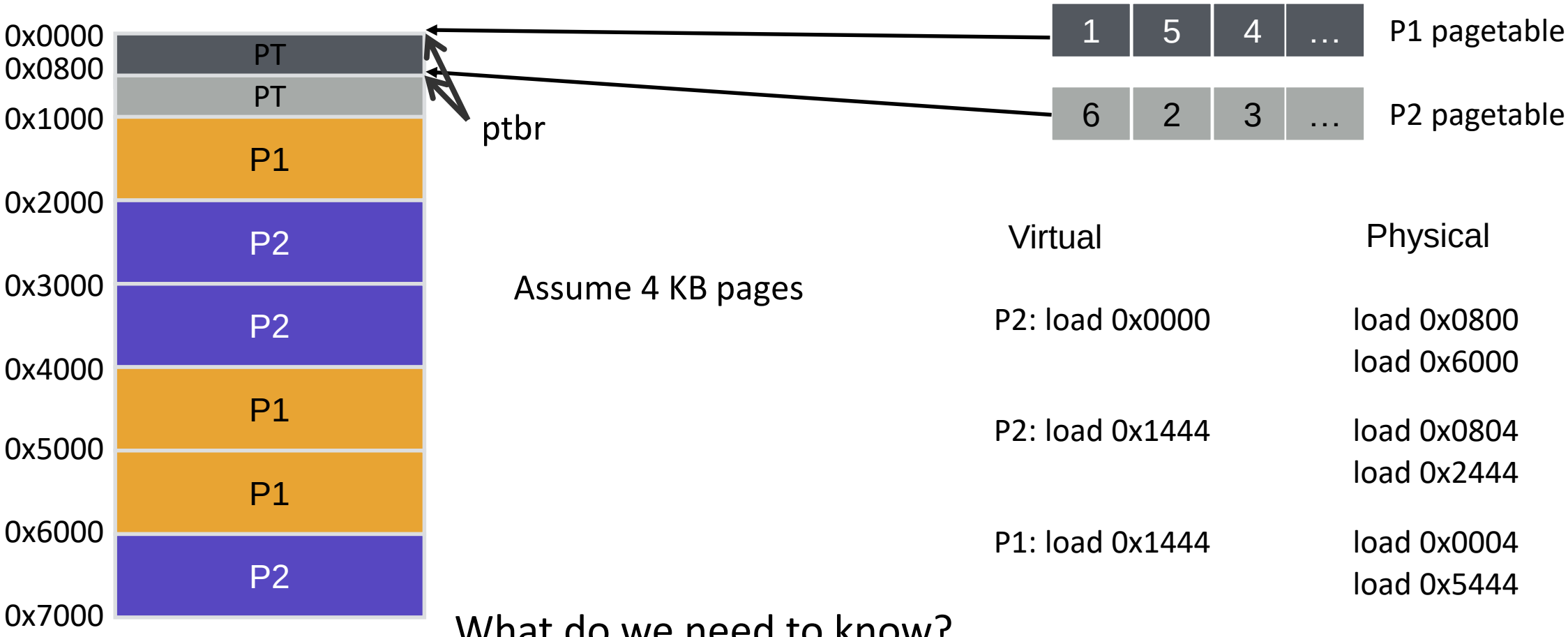
VM: TLBs

Sridhar Alagar

Review: Paging



Assume 4 KB pages

Virtual	Physical
P2: load 0x0000	load 0x0800 load 0x6000
P2: load 0x1444	load 0x0804 load 0x2444
P1: load 0x1444	load 0x0004 load 0x5444

What do we need to know?

Location of page table in memory (ptbr)

Size of each page table entry (assume 4 bytes)

Review: Paging Pros and Cons

- Advantages
  - No external fragmentation
  - All free pages are equivalent
- Disadvantages
  - Page tables are too big
    - Must have one entry for every page of address space
  - Accessing page tables is too slow
  - Doubles number of memory references per instruction

Example: Array

```
int sum = 0;
for (i=0; i<N; i++){
      sum += a[i];
}
```

Assume ‘a’ starts at 0x3000

Ignore instruction fetches

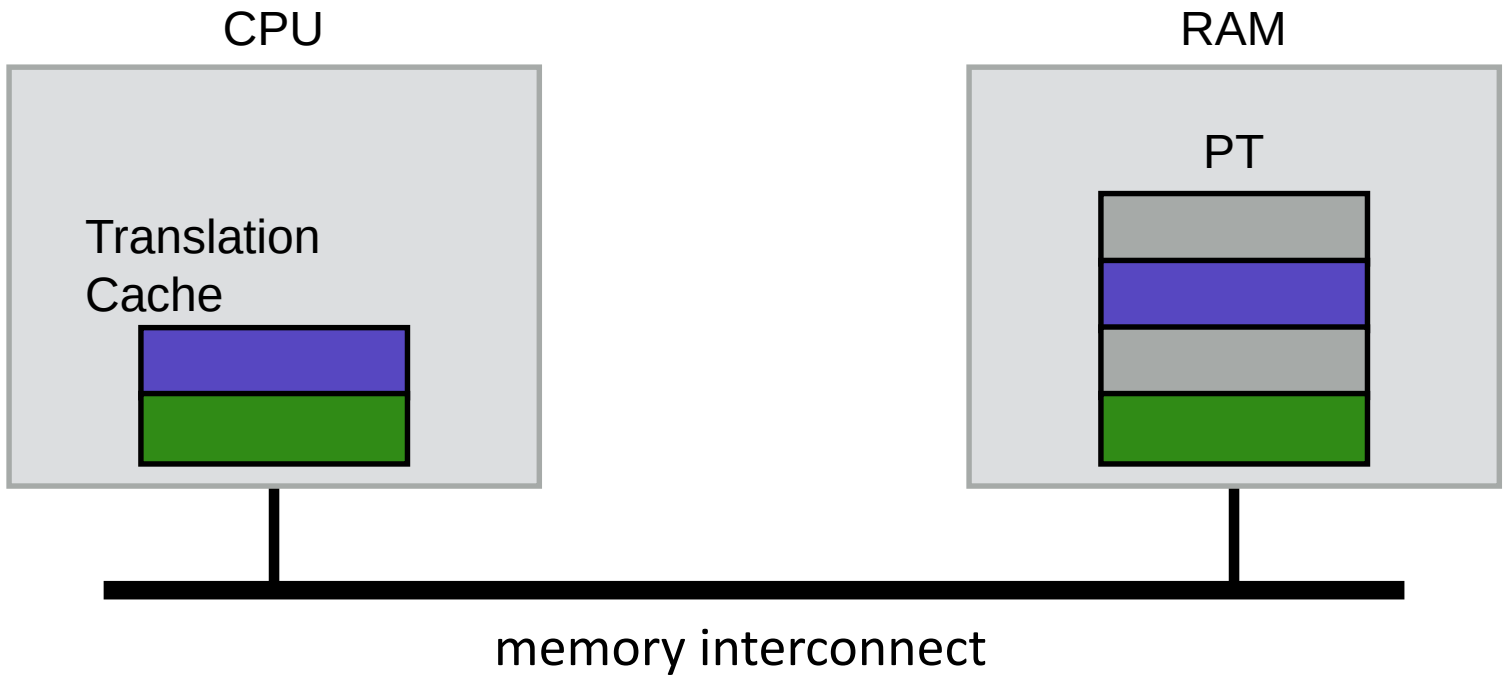
virtual addresses	physical addresses?
load 0x3000	load 0x100C
	load 0x7000
load 0x3004	load 0x100C
	load 0x7004
load 0x3008	load 0x100C
	load 0x7008
load 0x300C	load 0x100C
	load 0x700C
…	

Observation:
Repeatedly access same PTE because program repeatedly accesses same virtual page

What can you infer?

- ptbr: 0x1000; PTE 4 bytes each
- VPN 3 -> PPN 7

Cache Page Translations



TLB: Translation Lookaside Buffer

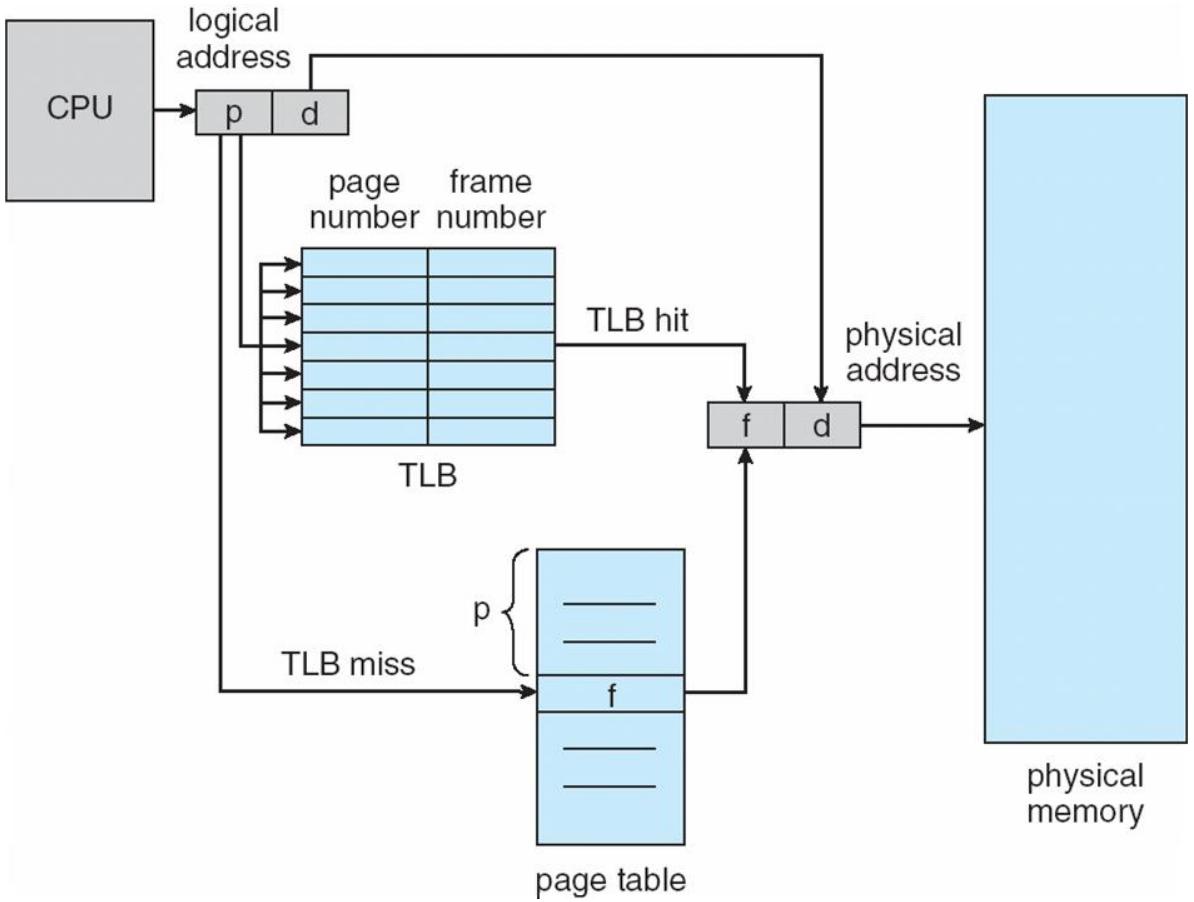
TLB is Fully Associative

- Associative memory

Page #	Frame #
p1	f1

- Address translation (p, f)
  - If p1 is in associative register, get frame # f1 out
  - Do parallel search

Paging with TLB Hardware



TLB Example - Iterate Array

```
int sum = 0;
for(i = 0;i < 2048;i++){
      sum += a[i];
}
```

What will TLB behavior look like?

Assume ‘a’ starts at VA 0x1000:

load 0x1000

load 0x1004

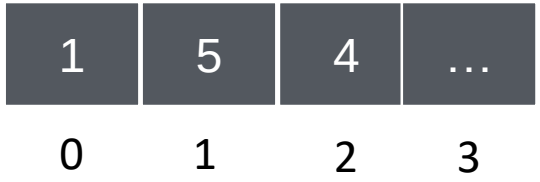
load 0x1008

load 0x100C

…

TLB Example

Pagetable @ 0x0000



CPU's TLB

VPN	PPN
1	5
2	4

Virt	Phys
load 0x1000	load 0x0004
load 0x1004	load 0x5000 (TLB hit)
	load 0x5004 (TLB hit)
load 0x1008	load 0x5008 (TLB hit)
load 0x100c	load 0x500C
…	…
load 0x2000	load 0x0008
	load 0x4000 (TLB hit)
load 0x2004	load 0x4004

TLB Performance

- miss rate = #TLB miss / # TLB Lookup
- In the sequential array access example:
  - # miss = 2
  - # lookup = 2048
  - miss rate = 2/2048 = 0.1%
  - hit rate = 99.9%
- What access pattern will be bad for hit rate?
- TLB Performance vs Page size?

Locality of reference

- Spatial Locality: future reference to nearby address
- Temporal Locality: future reference to the same data
- Both helps TLB hit rate

Who handles LB miss?

- H/W
  - Should know where page table is stored (CR3 in x86)
  - PT structure fixed and agreed between H/W and OS
  - Walk the page table and fetch the appropriate entry

- OS. How will it know the TLB miss?
  - Interrupt
  - Software managed TLB

Replacement Policies

- If TLB is full, which entry should be replaced?
- LRU: replace the least recently used
- Random: replace randomly chosen entry
- Which is better?

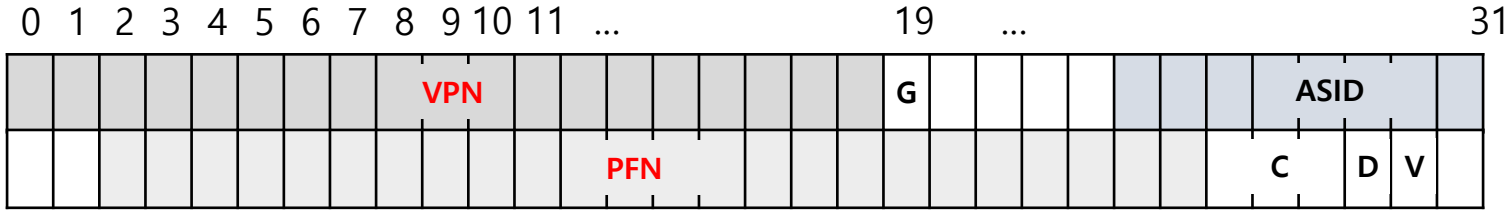
Context Switching

- Any problem for TLB?
  - TLB contain previous process entries
- Flush the old entries during context switch
  - will loose all cached translation
  -
- Share TLB across context switch
  - Address space identifier (ASID)
  - Tag each entry with ASID

Other bits in TLB entry

- valid/invalid

All 64 bits of this TLB entry(example of MIPS R4000)



Flag	Content
19-bit VPN	The rest reserved for the kernel.
24-bit PFN	Systems can support with up to 64GB of main memory($2^{24} * 4KB$ pages).
Global bit(G)	Used for pages that are globally-shared among processes.
ASID	OS can use to distinguish between address spaces.
Coherence bit(C)	determine how a page is cached by the hardware.
Dirty bit(D)	marking when the page has been written.
Valid bit(V)	tells the hardware if there is a valid translation present in the entry.

Disclaimer

- Some of the materials in this lecture slides are from the lecture slides by Prof. Arpaci, Prof. Youjip, and other educators. Thanks to all of them.