CS354: Machine Organization and Programming

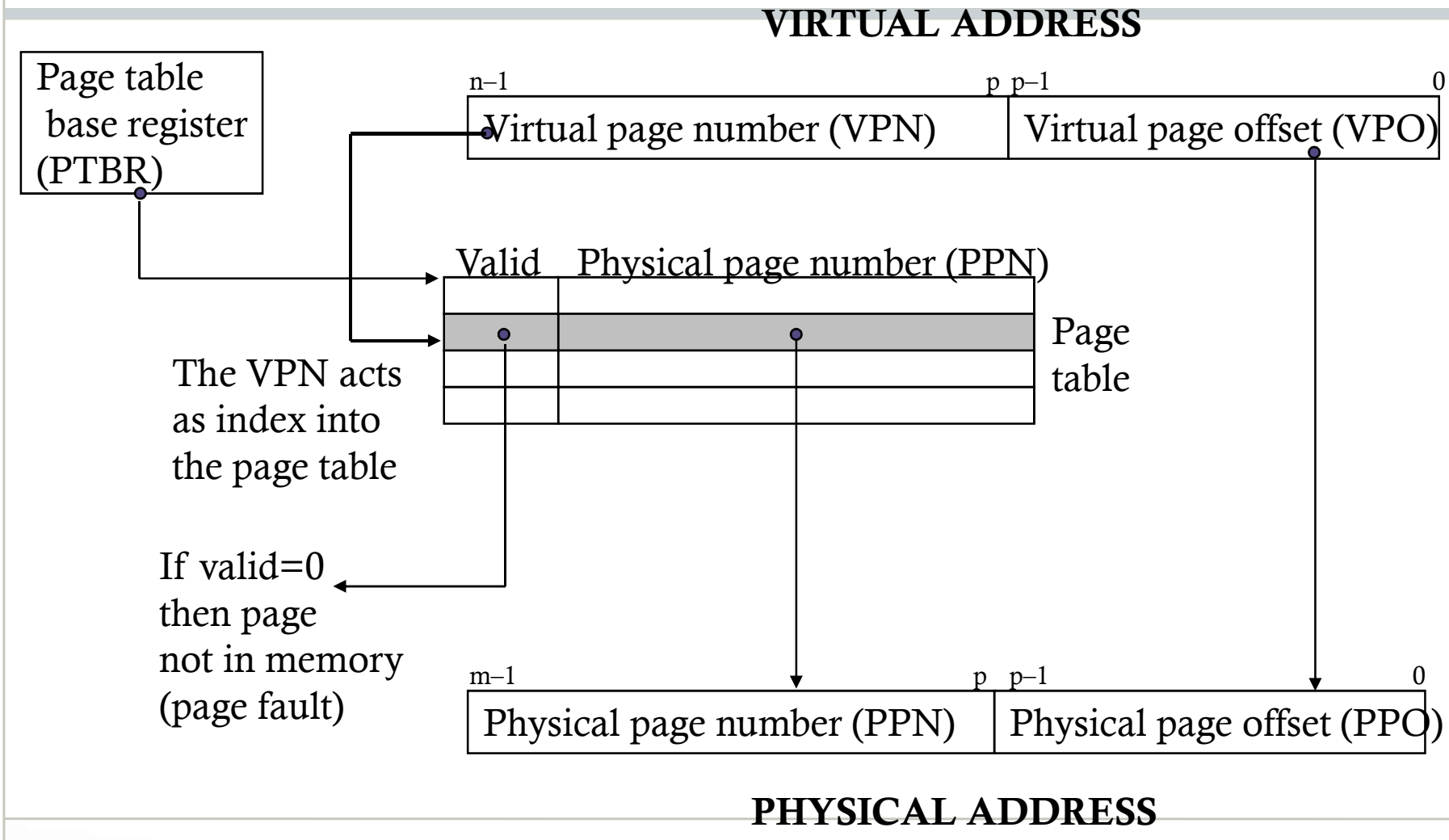
Lecture 31
Friday the November 13th 2015

Section 2
Instructor: Leo Arulraj

Lecture Overview

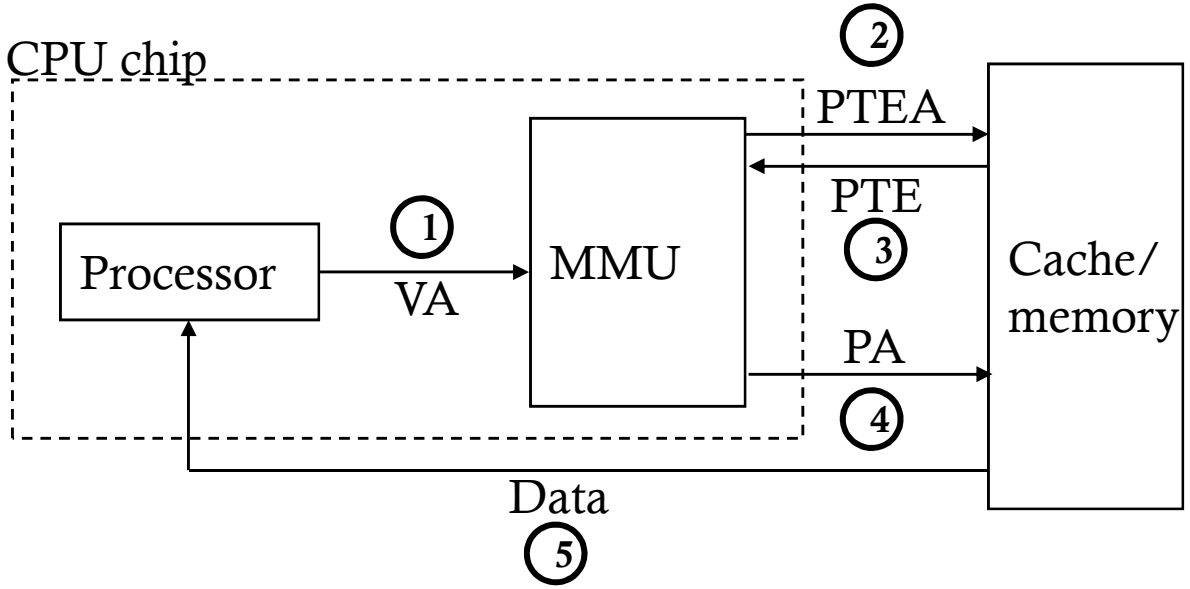
1. Address Translation for Flat Page Tables
2. Paging vs Older Techniques
3. Paging: More details and problems.
4. How to make VM fast – TLB
5. Structure of Page Tables

Address Translation



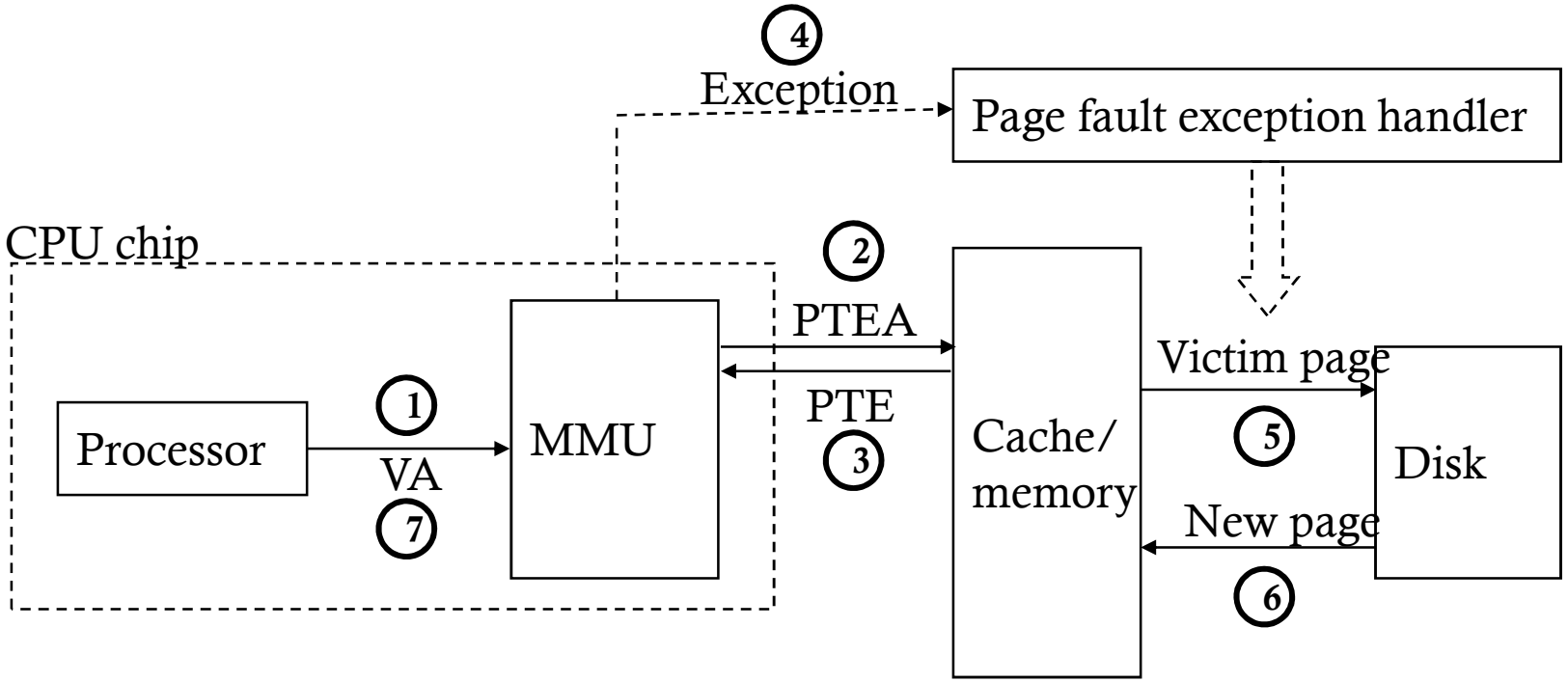
Virtual Memory Hit

Page Hit Handling:



Virtual Memory Miss

Page Fault Handling:



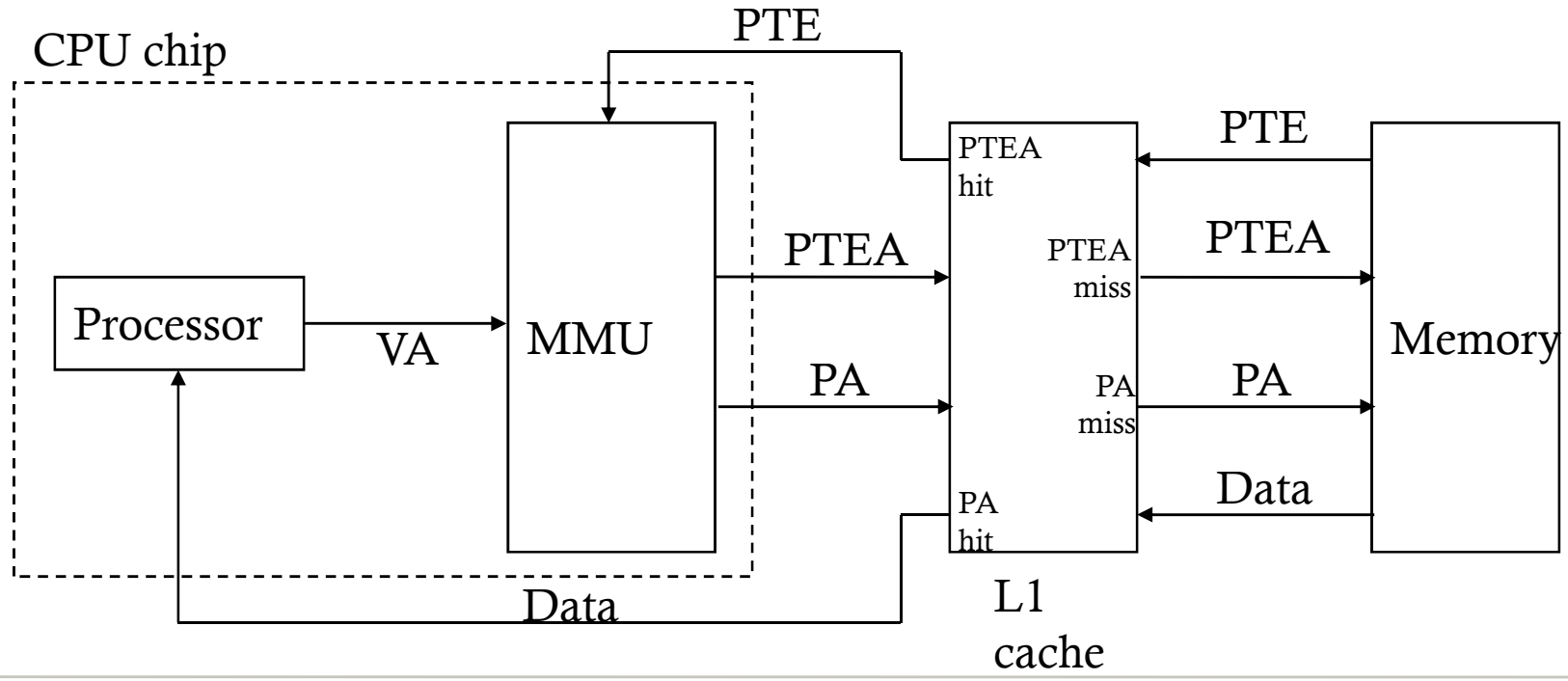
Which Addressing is used for L1 Cache?

CPU Caches can be addressed using either virtual memory address or physical address.

Most systems use physical addressing. Some advantages of this are:

1) Shared pages have just one copy in CPU cache
2) Protection issues are handled by the MMU during address translation before CPU cache lookup.

Using Physical Addresses for CPU Cache Lookup



Alternative of Paging: Segmentation

1. Physical Memory broken into fixed equal size segments/partitions.
2. Hardware Support: A Base register
3. Physical address = virtual address + base register
4. Leads to Internal Fragmentation because all segments have same size.

Alternative of Paging: Base and Bounds

Add a additional bounds/limit register to allow variable sized partitions.

Avoids internal fragmentation.

Leads to External Fragmentation: Just loading and unloading processes produces variable sized empty physical memory regions in DRAM.

Virtual Memory: Paging Recap

1. Each Process has an isolated virtual address space.
2. Virtual Memory implementation via Paging.
3. Each Process has its own page table.
4. Helps with caching disk contents in DRAM.
5. Helps with Memory sharing.
6. Helps with Memory protection using addition protection bits in the PTE.

Virtual Memory: Paging Recap

Address of the Page Table of the currently executing process in a special CPU register (PTBR).

When the operating system reschedules another process, then it updates the PTBR register with the base address of the newly scheduled process.

Virtual Memory: Paging Recap

Advantages of Paging:

- Paging supports flexible address spaces and does not waste physical memory with unused address space. (valid bit)

- Easy to manage the free physical memory space by maintaining a list of free physical frames. (fixed size pages is helpful here)

Virtual Memory: Paging Problem #1

Page tables are too slow.

Each memory reference needs another memory reference to the PTE.

With non-flat page tables, more than one memory references might be needed for looking up the PTE.

Virtual Memory: Paging Problem #2

Page tables are too big.

E.g. with 1KB pages and 4GB virtual address space, each process needs a flat page table of 4 million PTEs.

Assuming each PTE is 32 bits in size, each page table needs 16MB space.

What about pages of size 4KB?

Why not just use much larger pages?

How to make virtual memory fast?
(Performance Overheads during Address Translation)

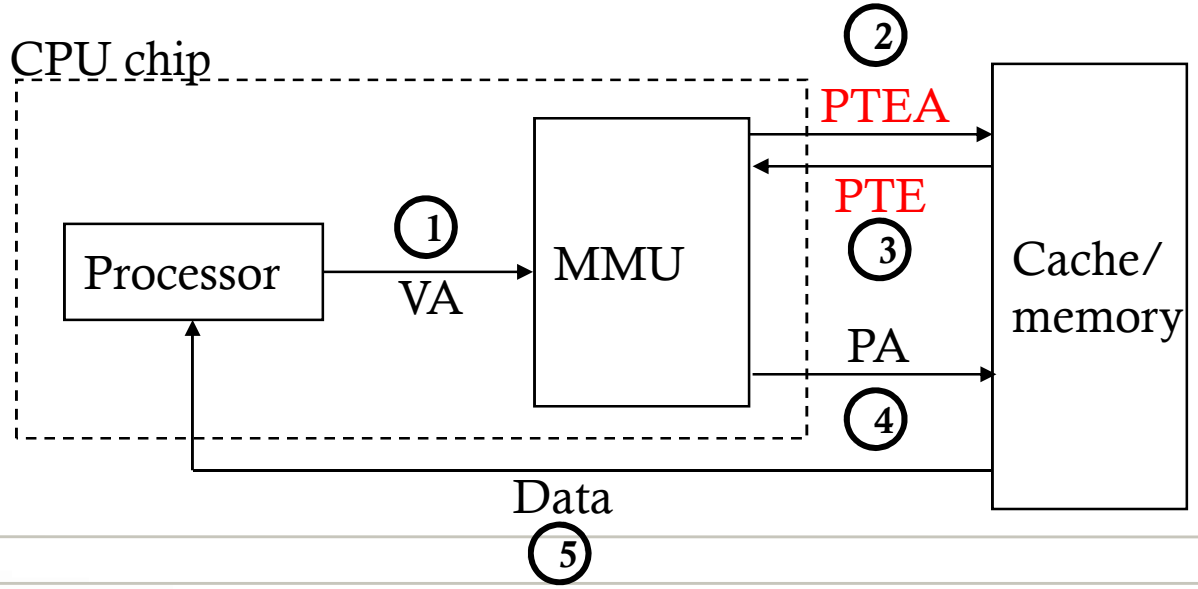
Worst case overhead: Involves an additional fetch the PTE from memory at a cost of tens to hundreds of cycles. (for flat page tables)

If the PTE is cached in L1 Cache, then the penalty is lesser.

How to avoid this overhead? Caching to the rescue again!

How to make virtual memory fast?

We want to avoid the expensive additional references to memory for fetching PTE in steps 2,3.

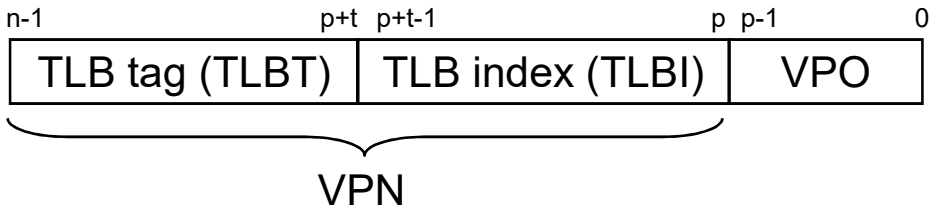


TLB – Translation Lookaside Buffer

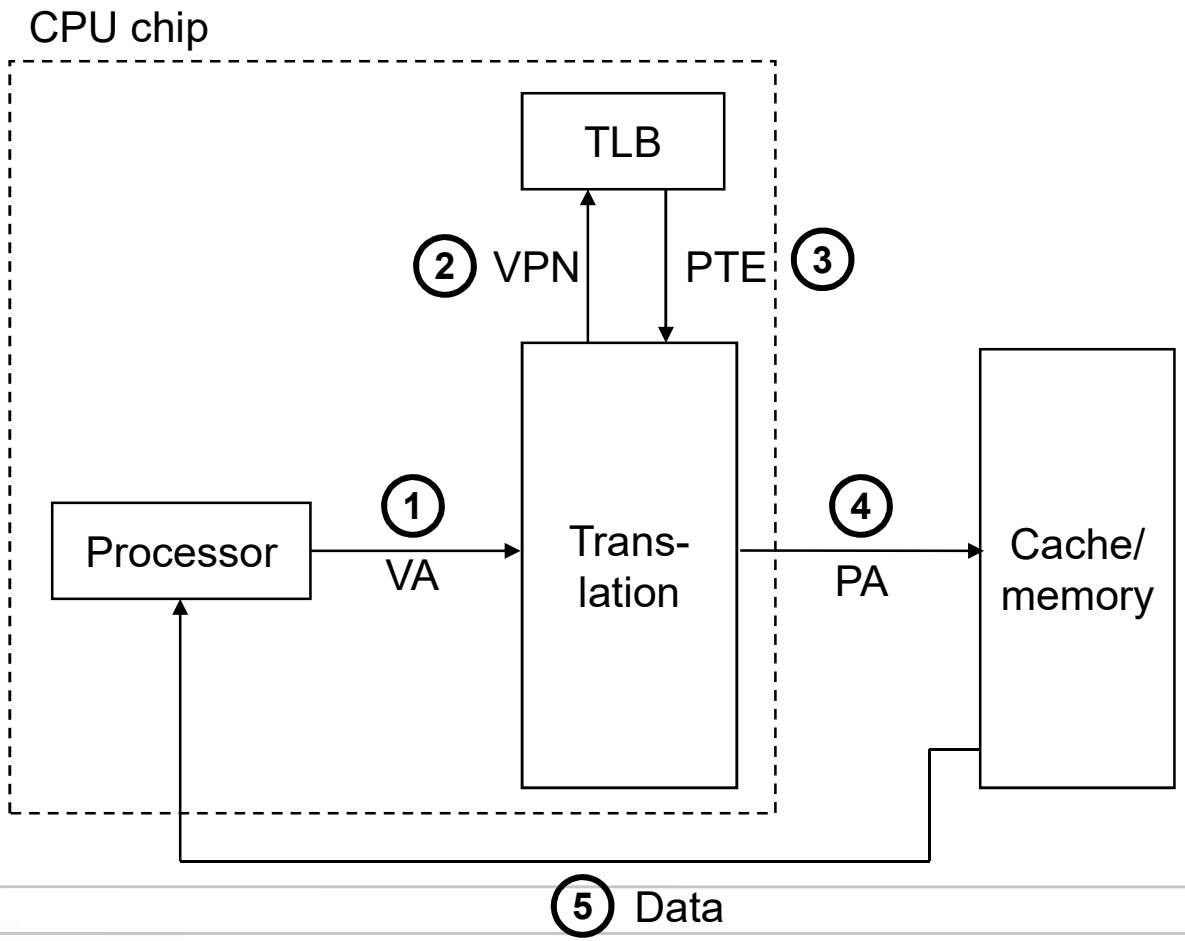
TLB is a small, virtually addressed cache.

Each line holds a single PTE.

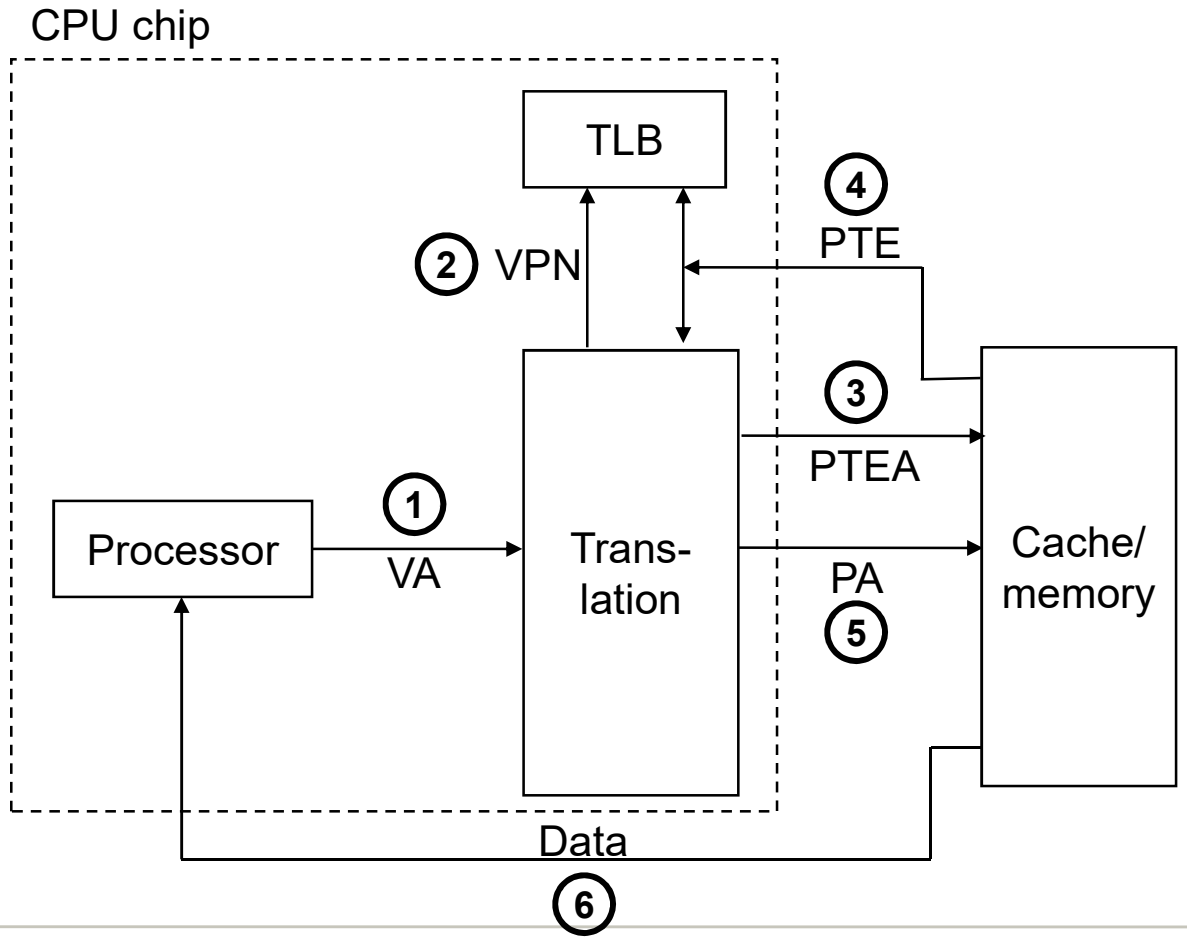
TLB similar in organization to L1 but has a high set associativity.



TLB Hit


CPU chip
TLB
2
VPN
PTE
3
Processor
1
VA
Trans-
lation
4
PA
Cache/
memory
5
Data

TLB Miss


CPU chip
TLB
4
PTE
2
VPN
3
PTEA
1
Processor
VA
Trans-
lation
PA
5
Cache/
memory
Data
6

TLB and context switches

What should happen to the TLB contexts when the OS schedules a new process?

Two solutions:

1) Flush: Clear the TLB cache entirely

2) ASID: Address Space Identifier with each PTE in order to isolate between address spaces of multiple processes.