Virtual Memory

EEL 3713C: Digital Computer Architecture

Quincy Flint

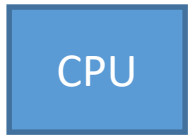
[Ionospheric Radio Lab in NEB]

Outline

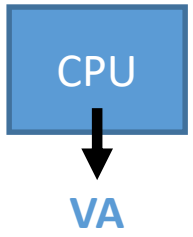
1. Memory Problems
   - Not enough memory
   - Holes in address space
   - Programs overwriting
2. What is Virtual Memory?
   - Layer of indirection
   - How does indirection solve above
   - Page tables and translation
3. How do we implement VM?
   - Create and store page tables
   - Fast address translation
4. Virtual Memory and Caches
   - Prevent cache performance degradation when using VM

TLBs + Caches

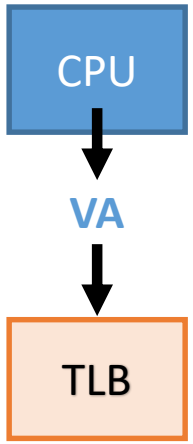
Virtual Cache



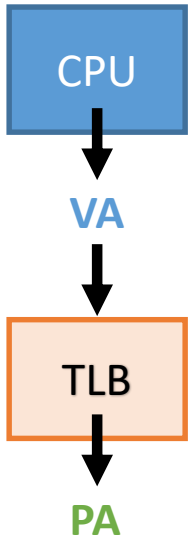
Virtual Cache



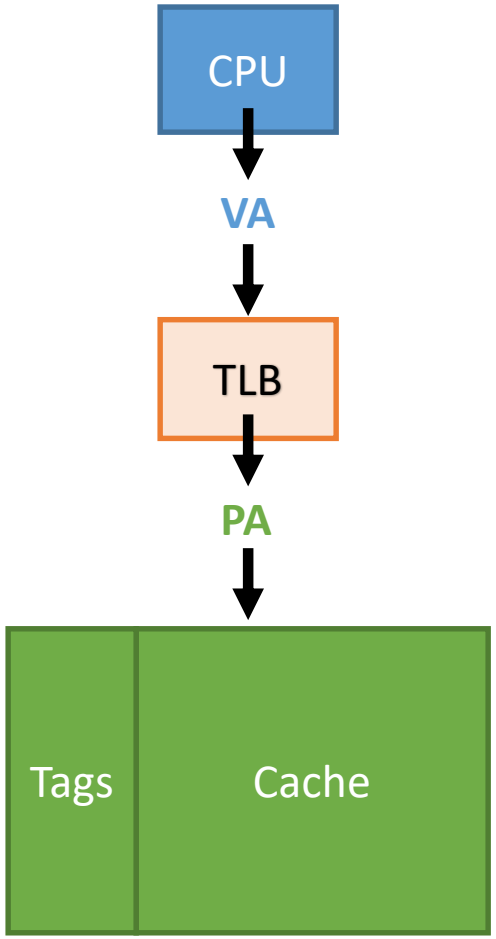
Virtual Cache



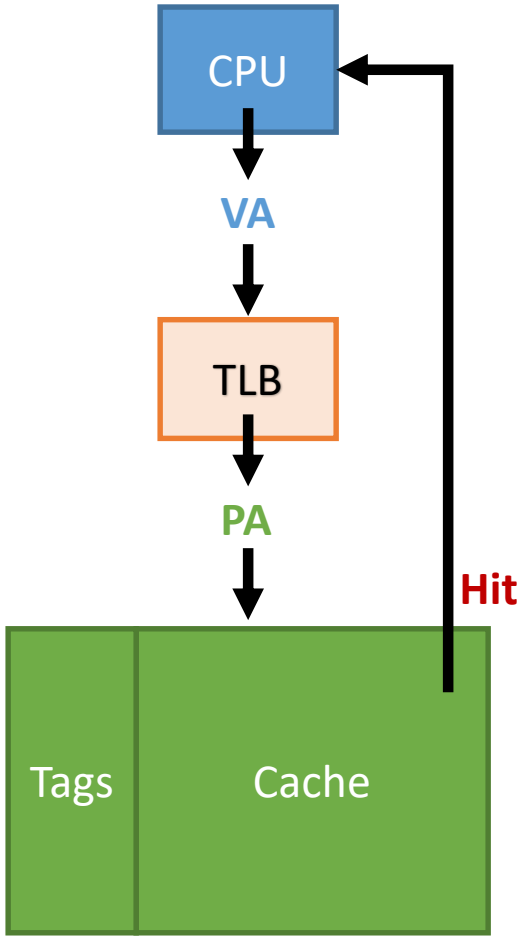
Virtual Cache



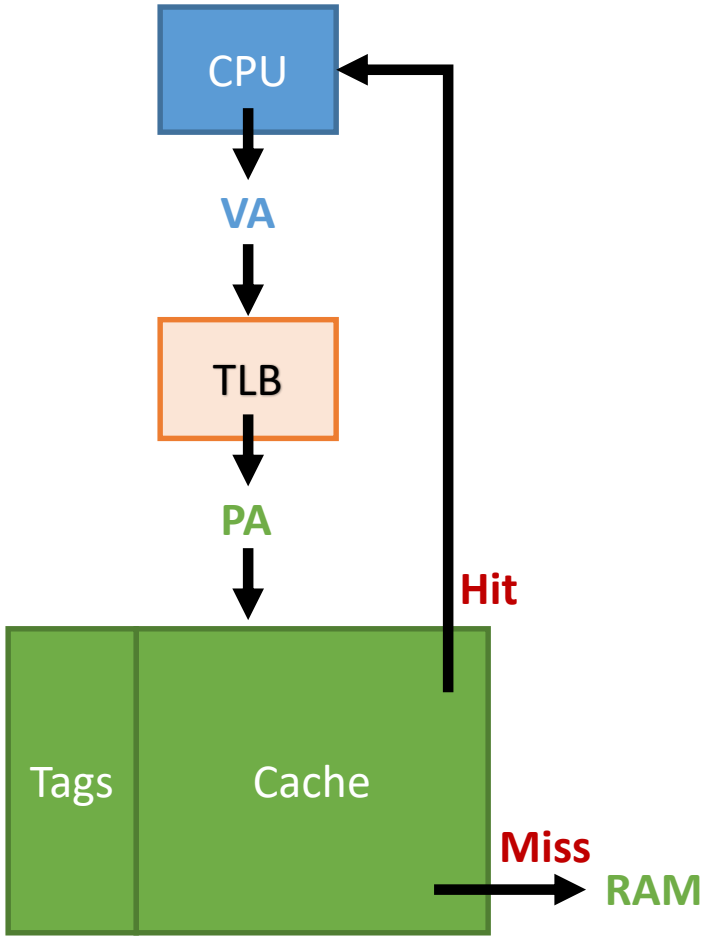
Virtual Cache



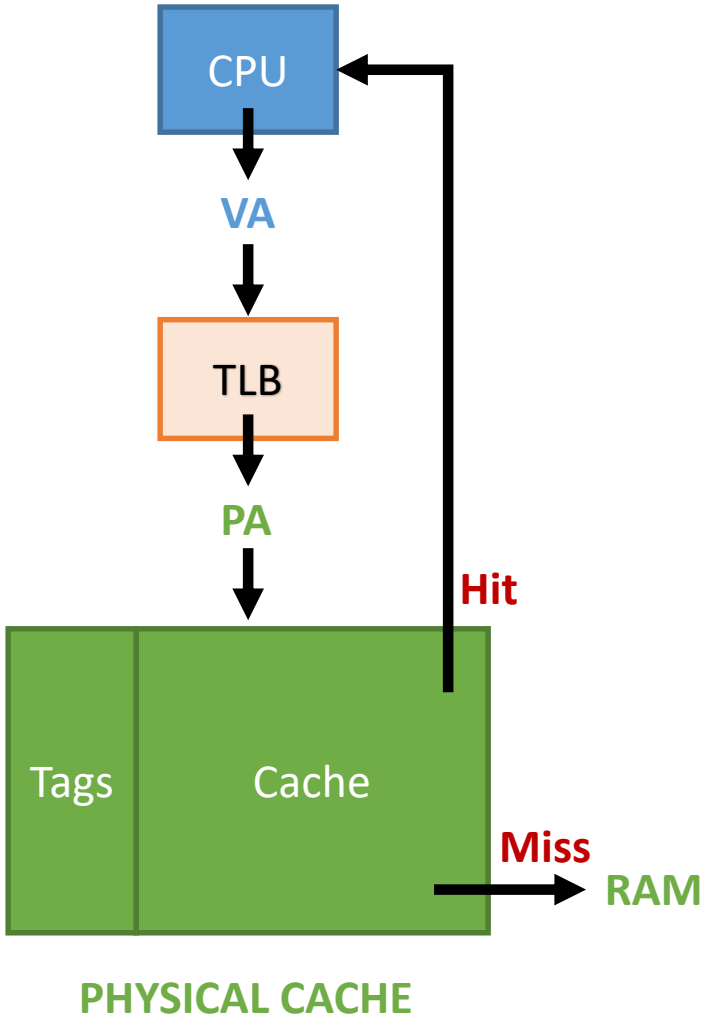
Virtual Cache



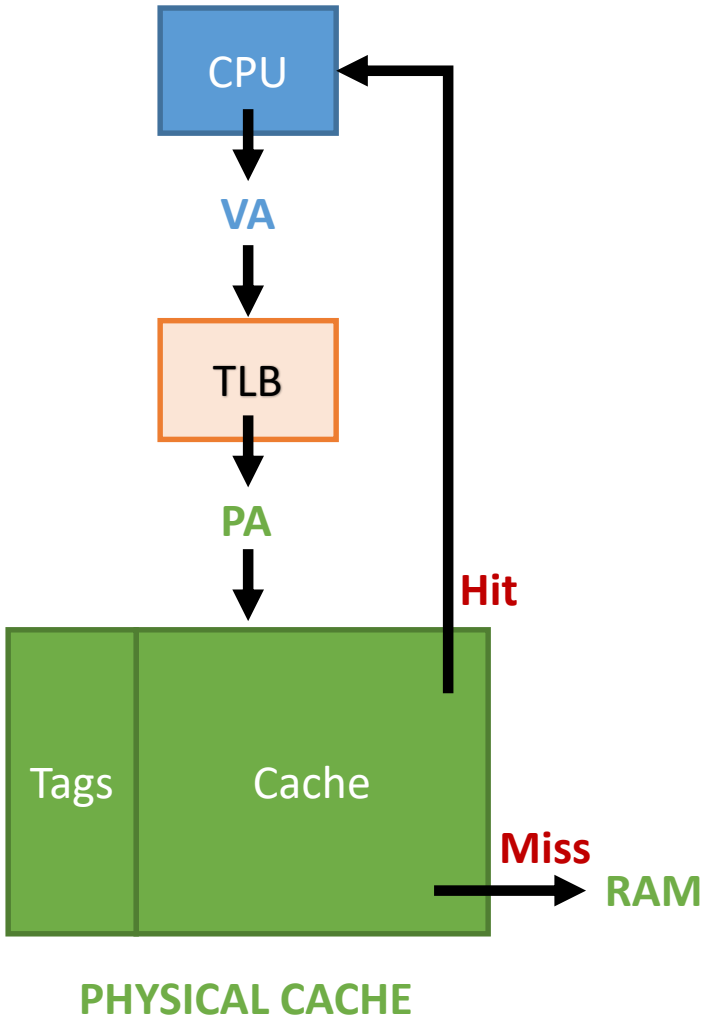
Virtual Cache



Virtual Cache

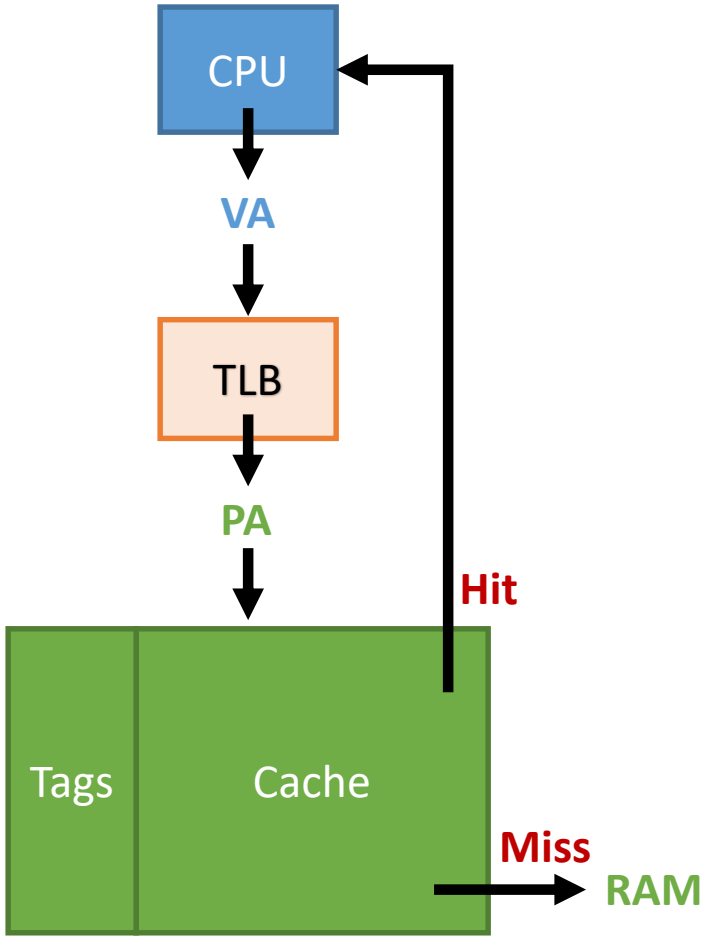


Virtual Cache



- Physical Cache
  - Must access TLB before cache [slow]

Virtual Cache

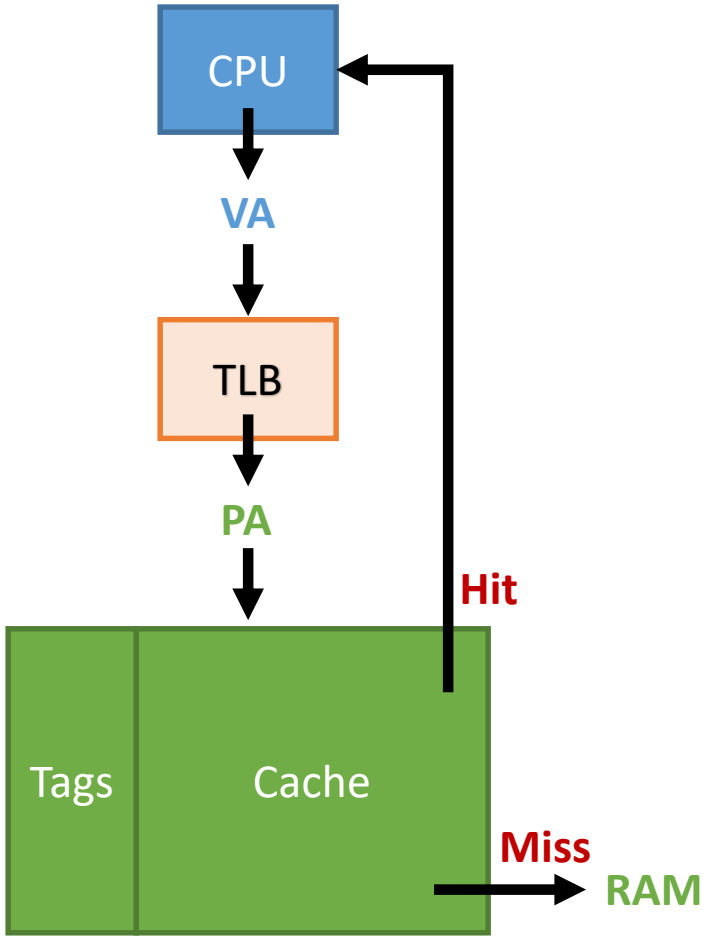


PHYSICAL CACHE

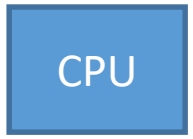
VIRTUAL CACHE

- Physical Cache
  - Must access TLB before cache [slow]

Virtual Cache



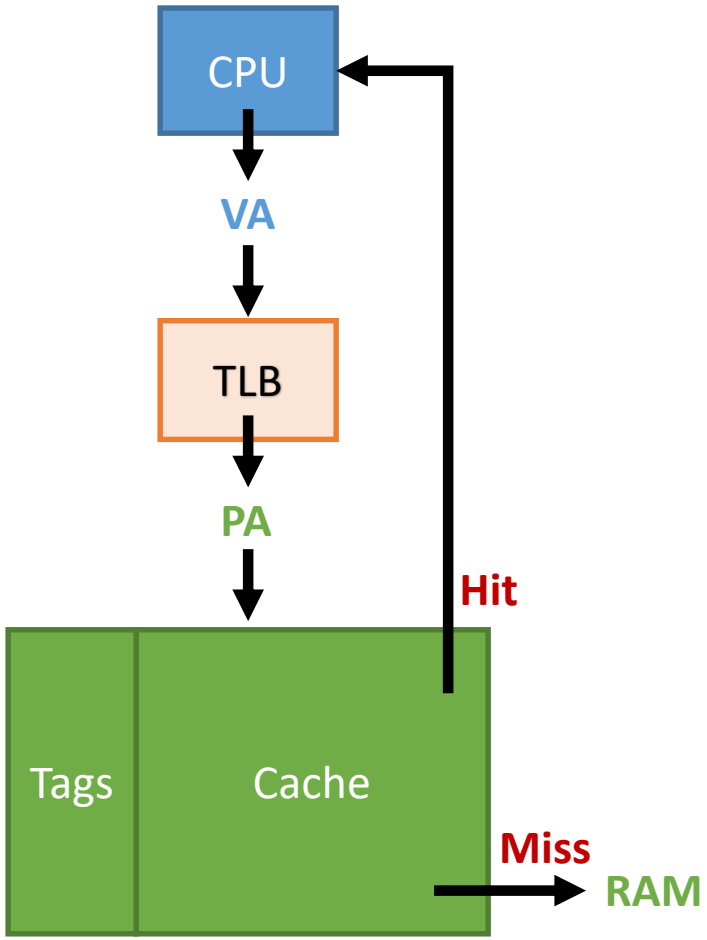
PHYSICAL CACHE



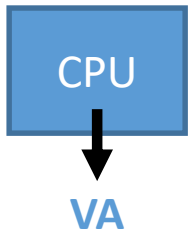
VIRTUAL CACHE

- Physical Cache
  - Must access TLB before cache [slow]

Virtual Cache



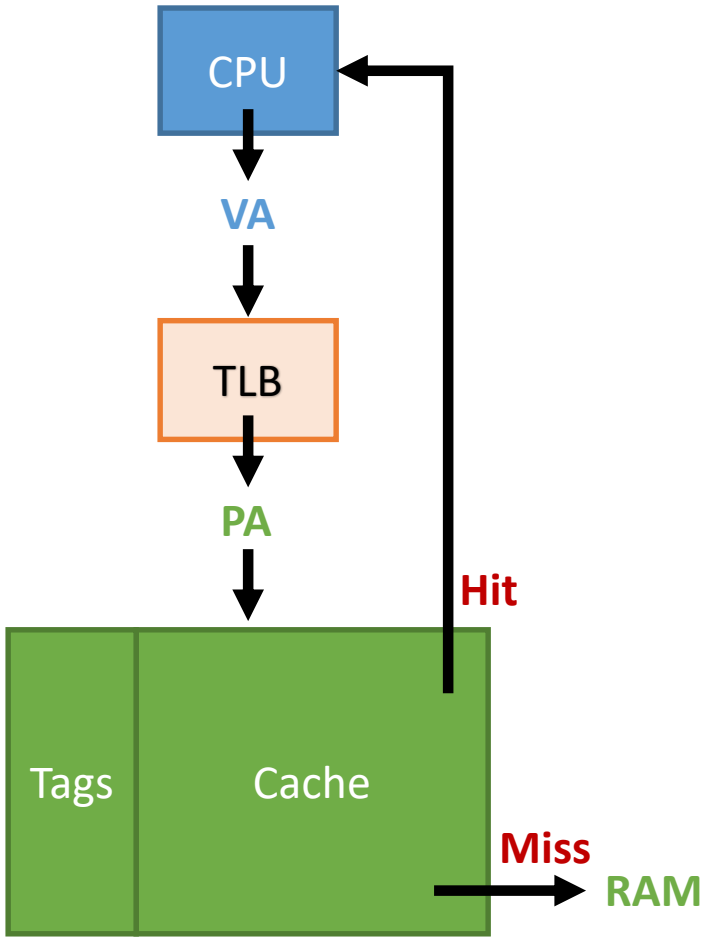
PHYSICAL CACHE



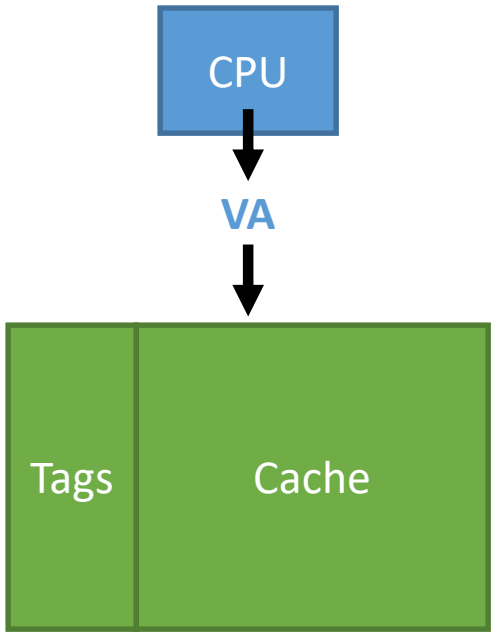
VIRTUAL CACHE

- Physical Cache
  - Must access TLB before cache [slow]

Virtual Cache



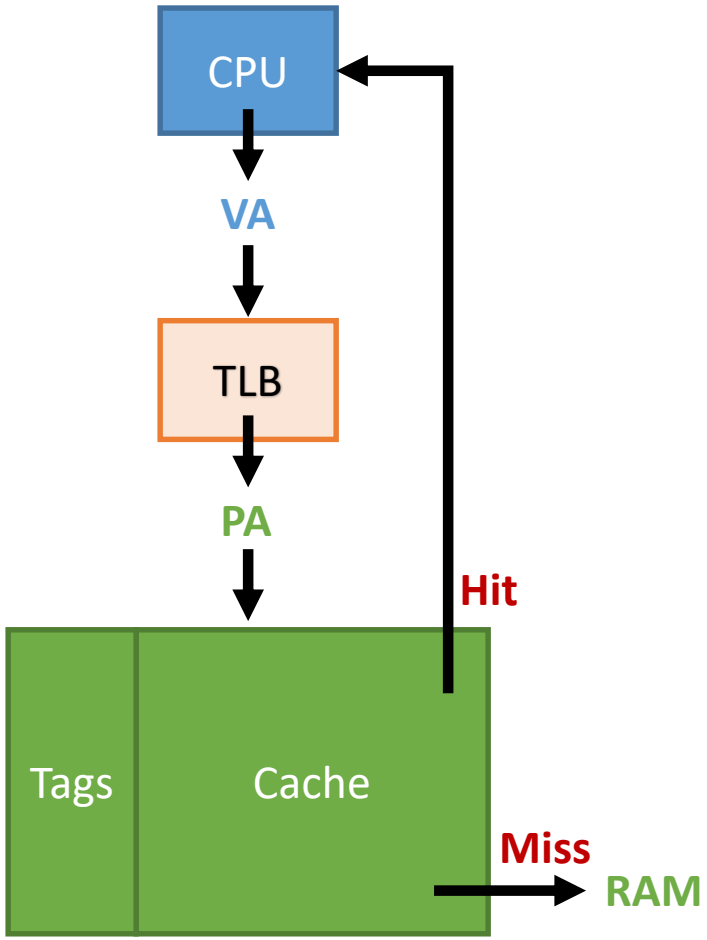
PHYSICAL CACHE



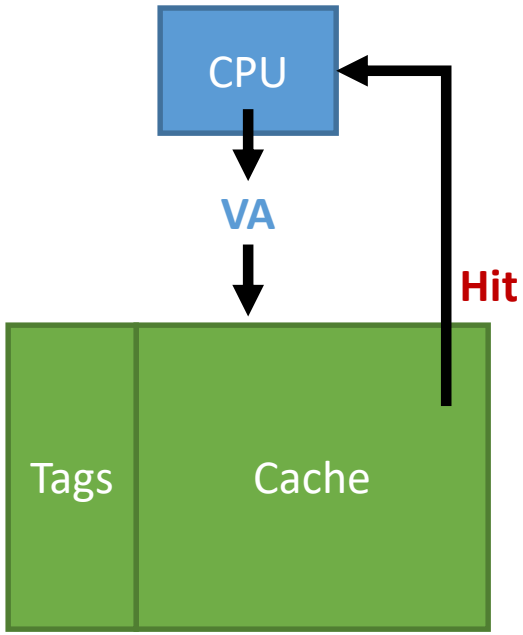
VIRTUAL CACHE

- Physical Cache
  - Must access TLB before cache [slow]

Virtual Cache



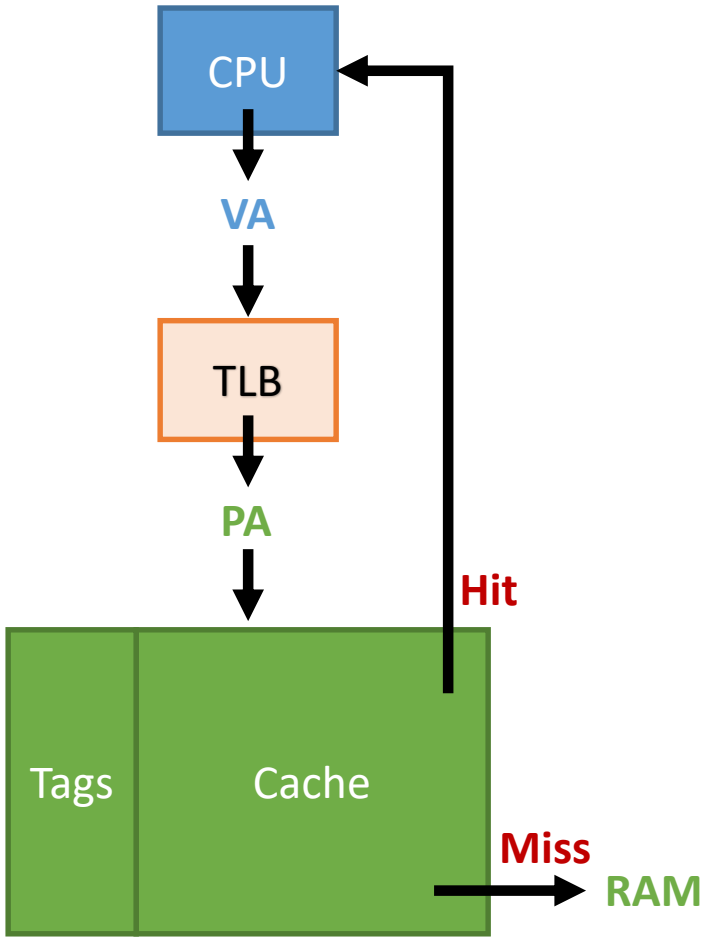
PHYSICAL CACHE



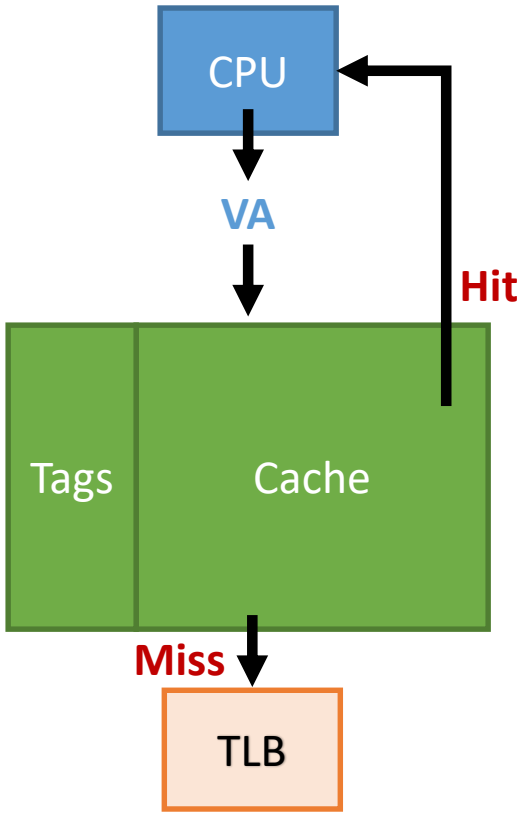
VIRTUAL CACHE

- Physical Cache
  - Must access TLB before cache [slow]

Virtual Cache



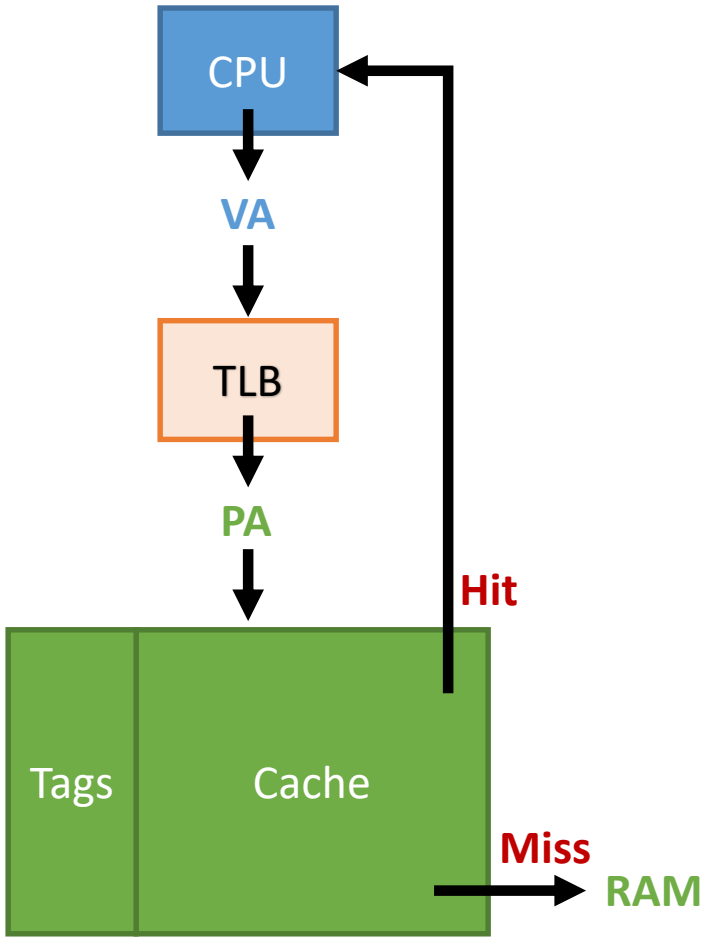
PHYSICAL CACHE

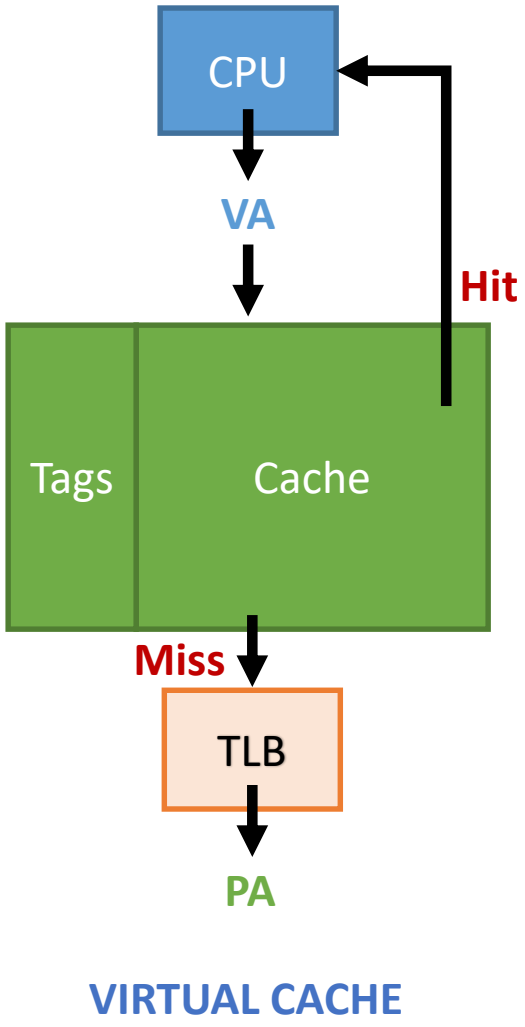


VIRTUAL CACHE

- Physical Cache
  - Must access TLB before cache [slow]

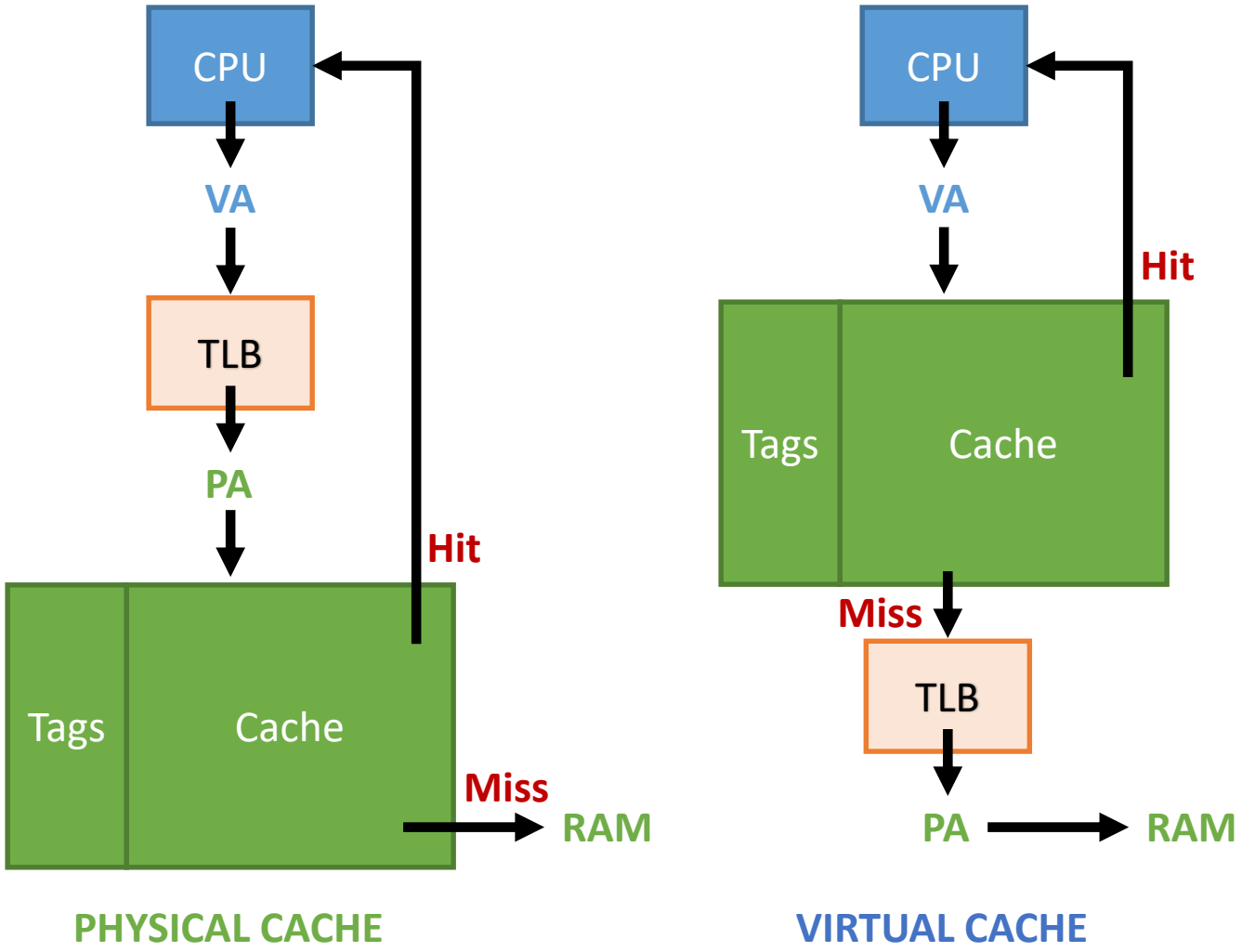
Virtual Cache





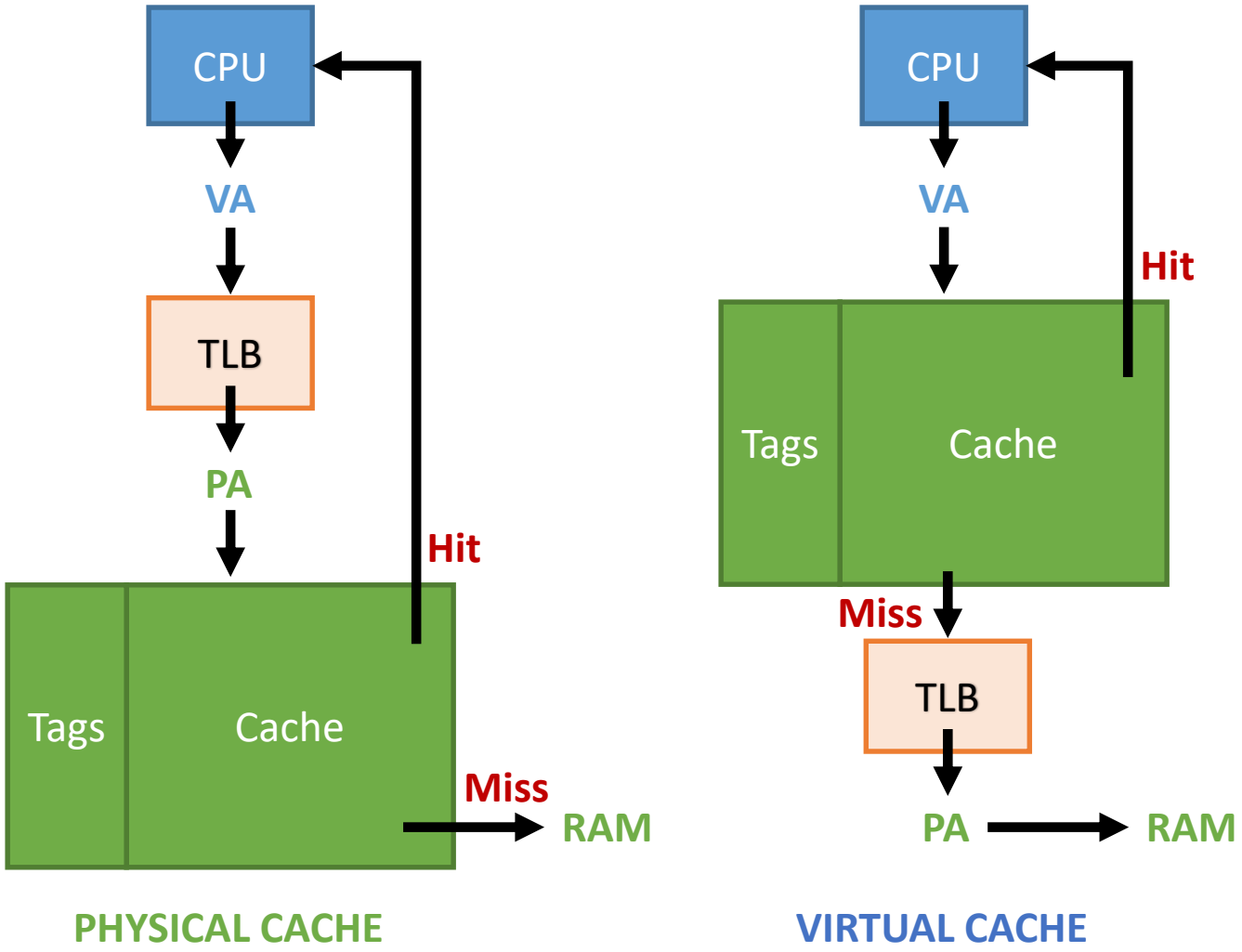
- Physical Cache
  - Must access TLB before cache [slow]

Virtual Cache



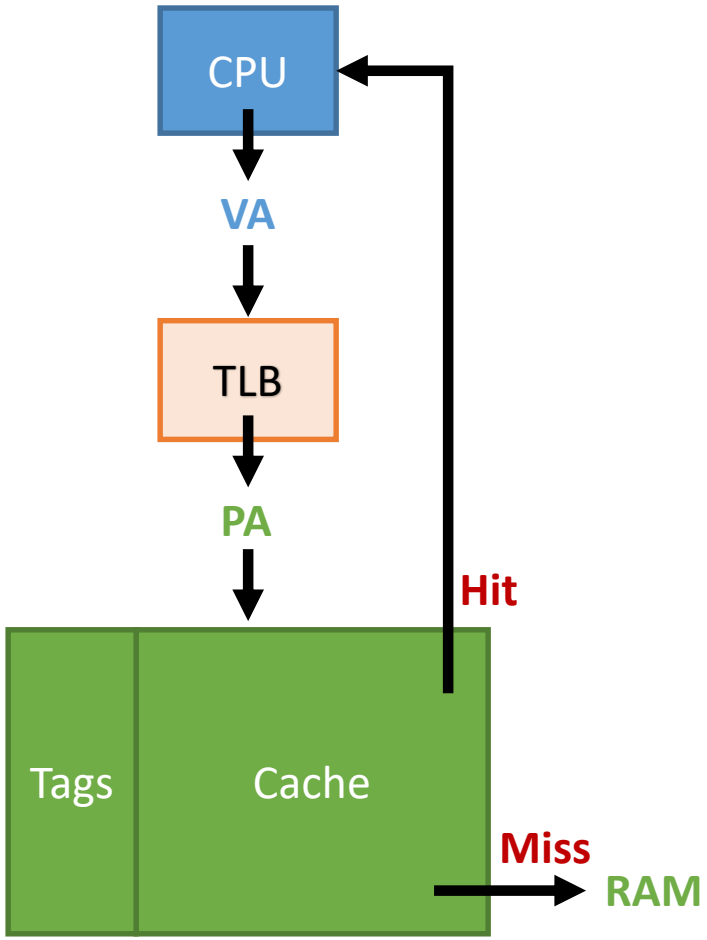
- Physical Cache
  - Must access TLB before cache [slow]

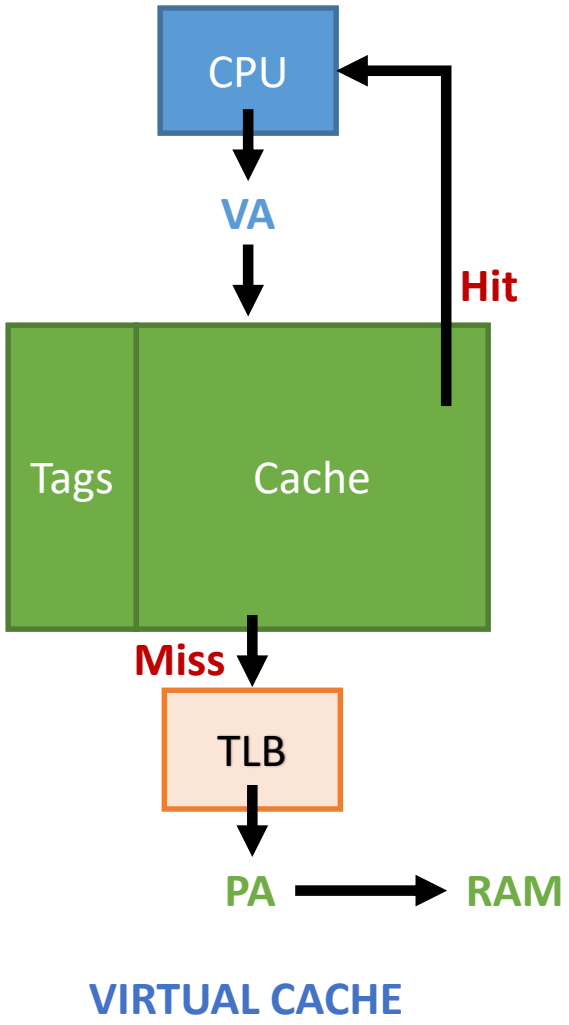
Virtual Cache



- Physical Cache
  - Must access TLB before cache [slow]

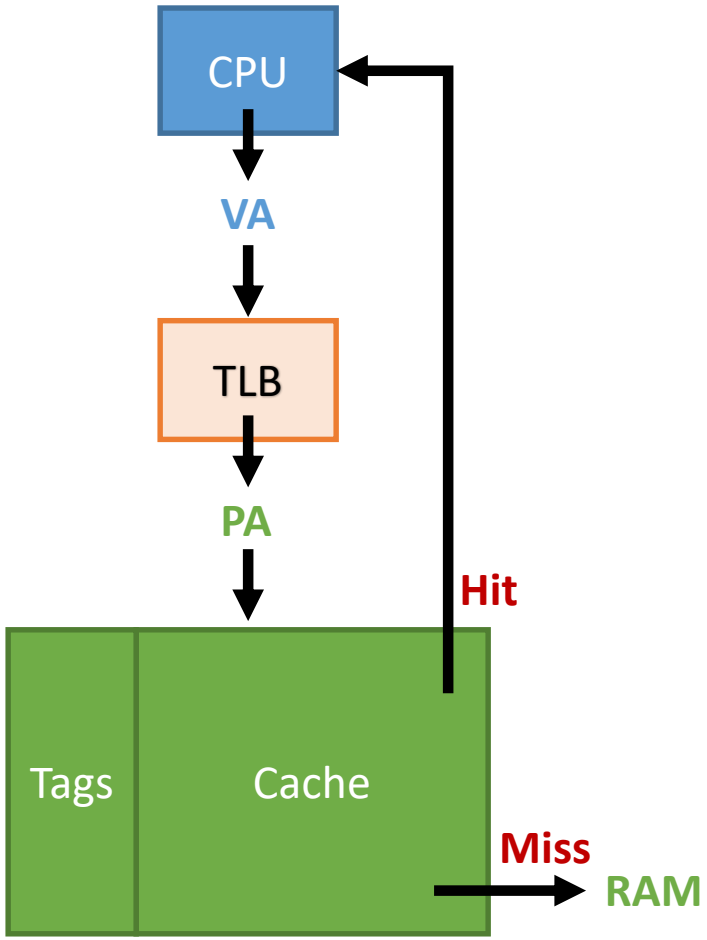
Virtual Cache



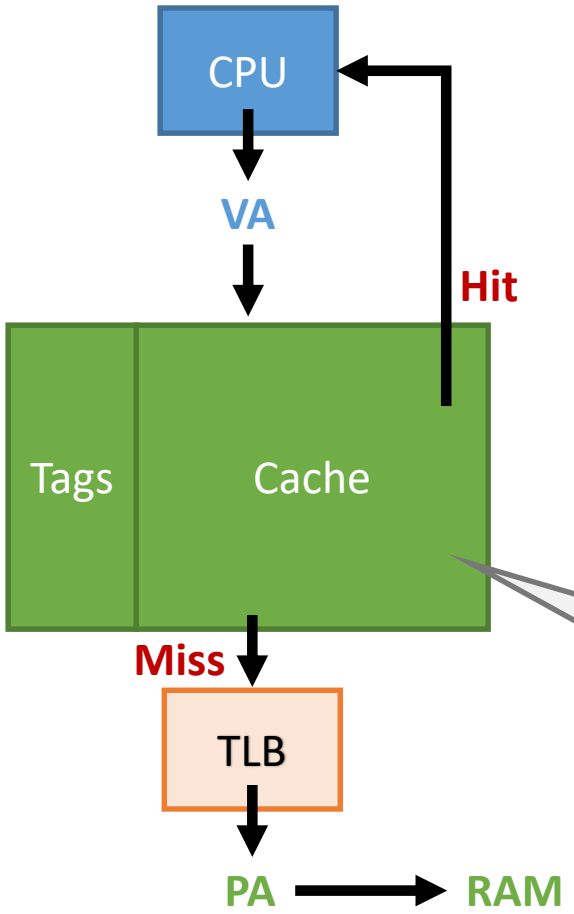


- Physical Cache
  - Must access TLB before cache [slow]
- Virtual Cache
  - Only use TLB on cache miss [faster]

Virtual Cache



PHYSICAL CACHE

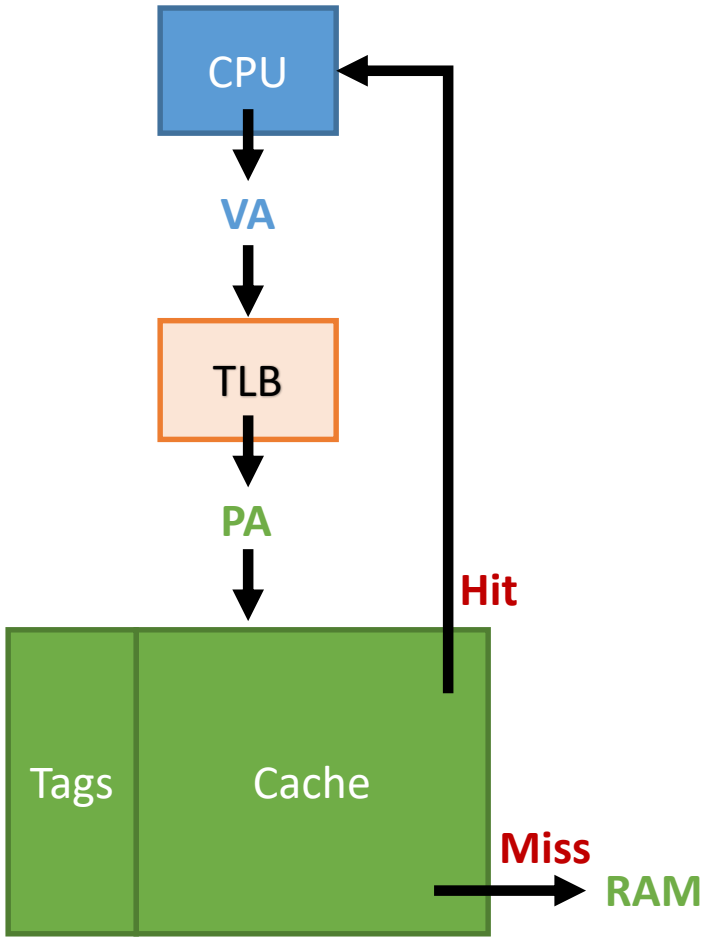


VIRTUAL CACHE

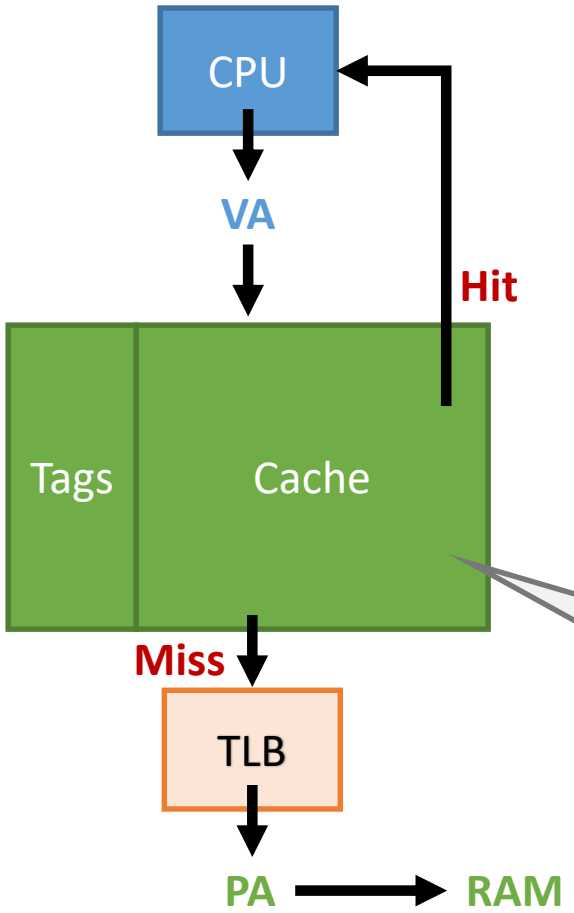
- Physical Cache
  - Must access TLB before cache [slow]
- Virtual Cache
  - Only use TLB on cache miss [faster]

Q: Can 2 programs share a virtual cache?

Virtual Cache



PHYSICAL CACHE



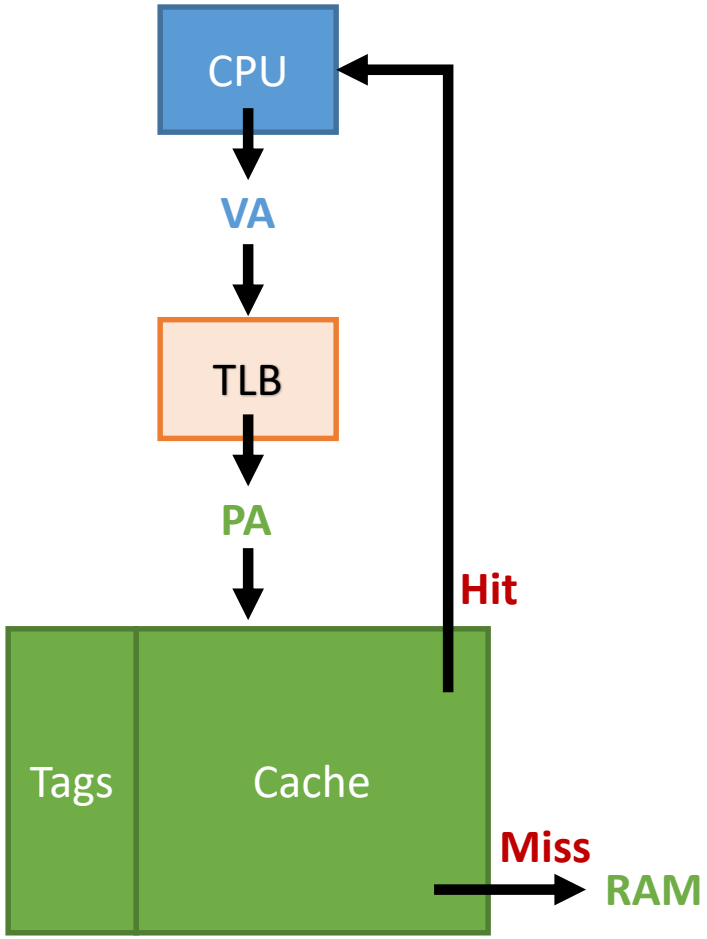
VIRTUAL CACHE

- Physical Cache
  - Must access TLB before cache [slow]
- Virtual Cache
  - Only use TLB on cache miss [faster]

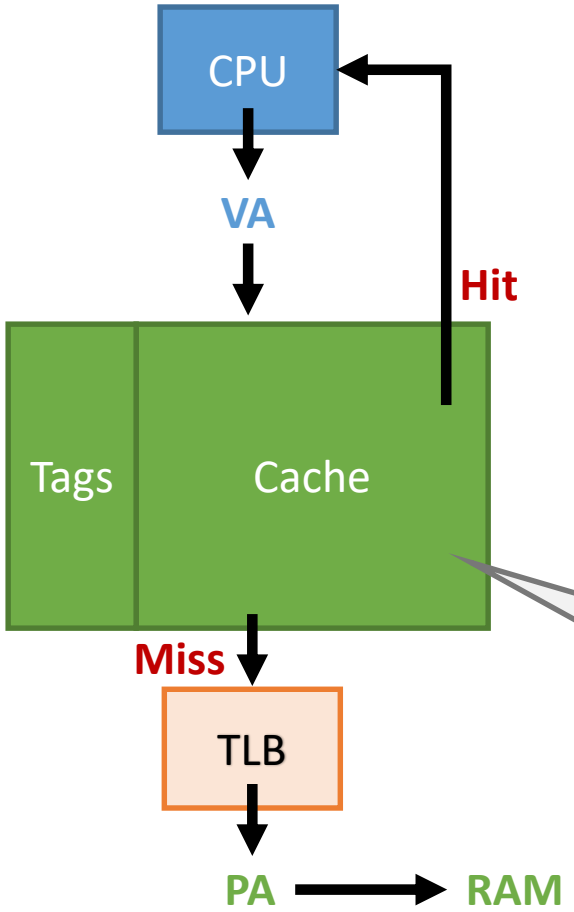
Q: Can 2 programs share a virtual cache?

A: No. The virtual cache stores data in a virtual address, there is now way to provide isolation..

Virtual Cache



PHYSICAL CACHE



VIRTUAL CACHE

- Physical Cache
  - Must access TLB before cache [slow]
- Virtual Cache
  - Only use TLB on cache miss [faster]

Q: Can 2 programs share a virtual cache?

A: No. The virtual cache stores data in a virtual address, there is now way to provide isolation..

… Unless we tag the entry or flush the TLB!

VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache

VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently

VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

- Basic Concept:
  - Lookup data in cache with virtual address

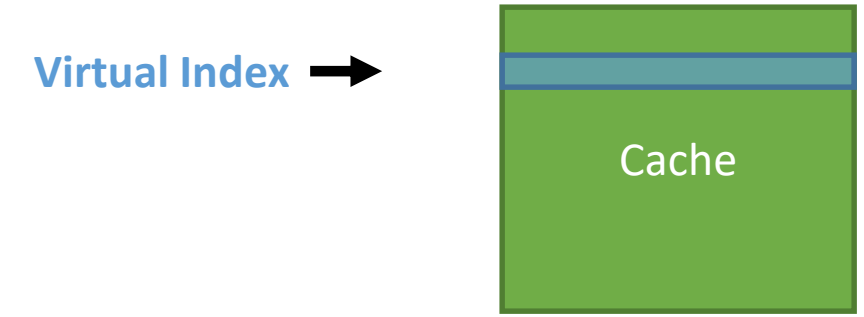
VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag

VIPT Cache

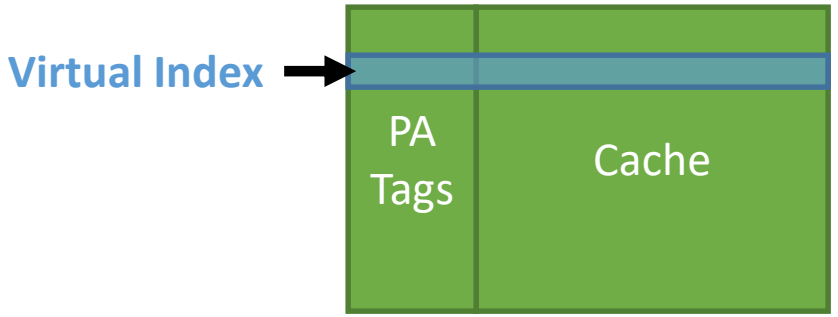
- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM
- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag



VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

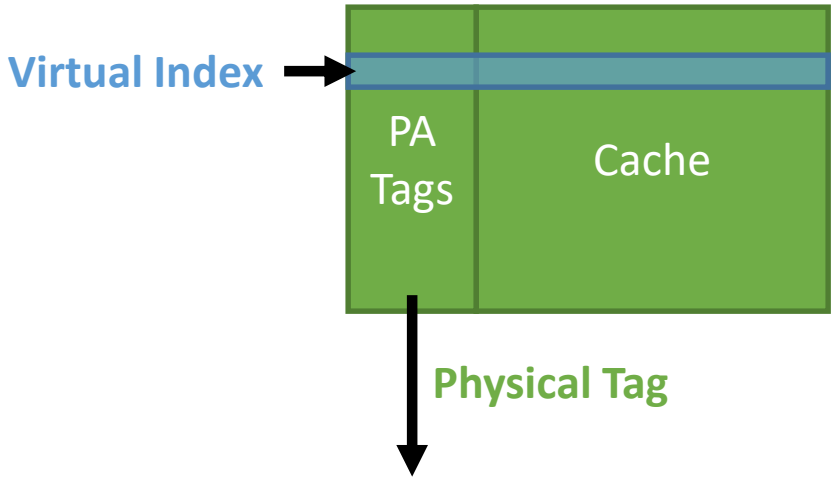
- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag



VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

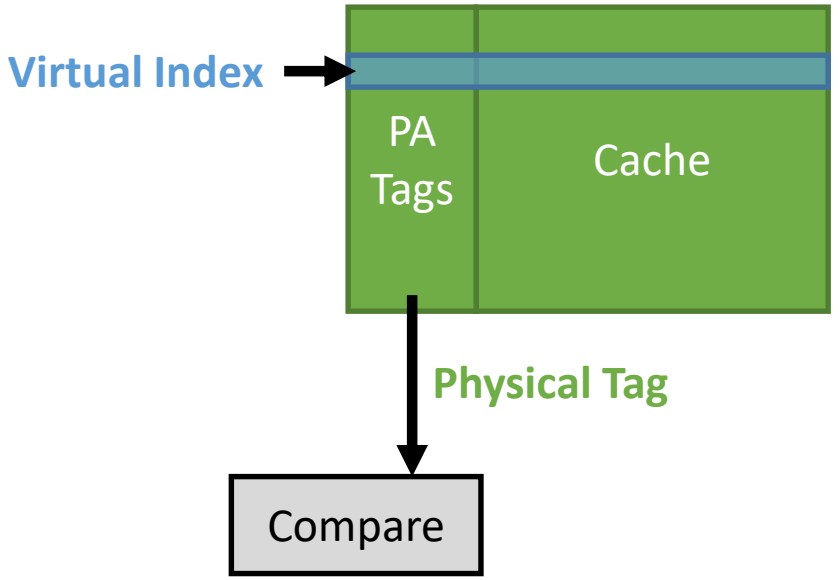
- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag



VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

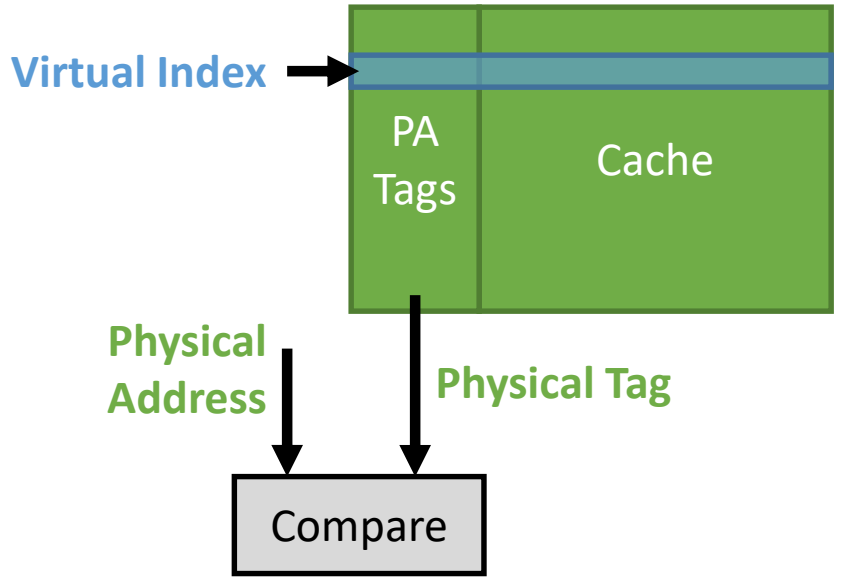
- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag



VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

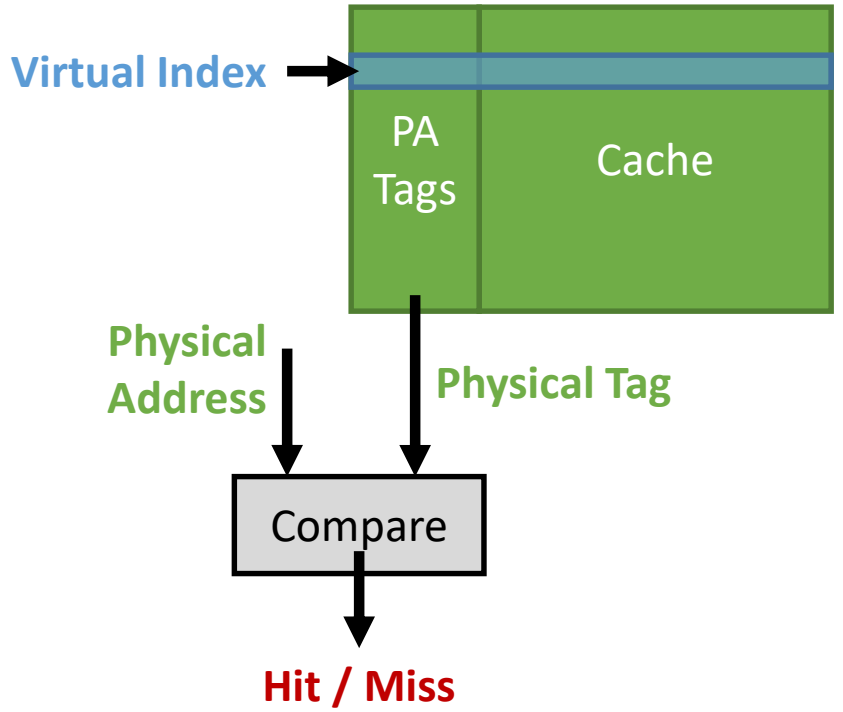
- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag



VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

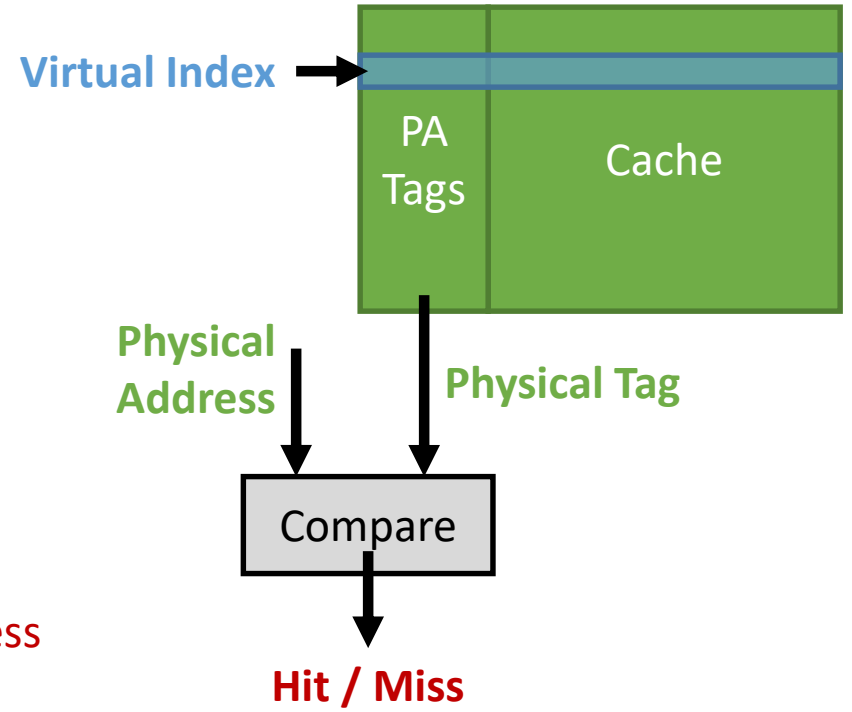
- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag



VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM

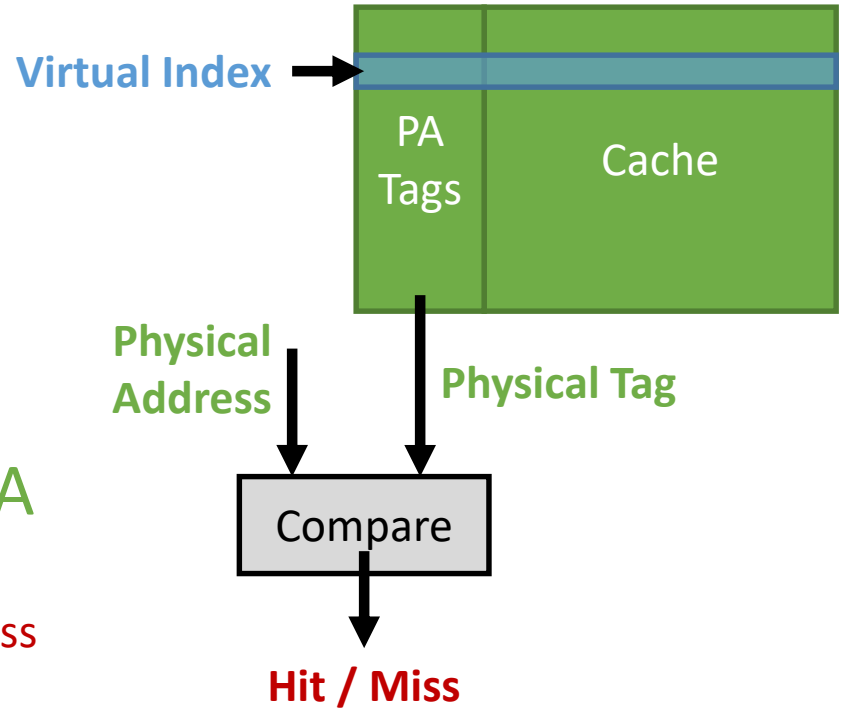
- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag



Hit if the tag matches the Physical Address

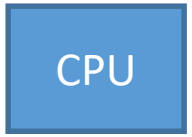
VIPT Cache

- Virtually Indexed, Physically Tagged [**VIPT**] Cache
  - Allows us to access the TLB and the cache concurrently
  - Provides memory protection, like VM
- Basic Concept:
  - Lookup data in cache with virtual address
  - Verify data is correct using physical tag
- Data in cache is *indexed* by VA, *tagged* by PA

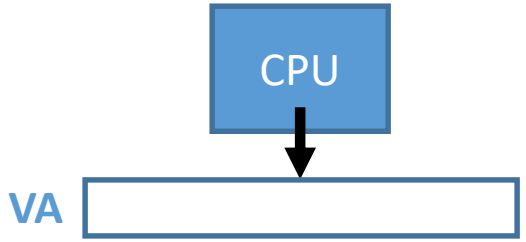


Hit if the tag matches the Physical Address

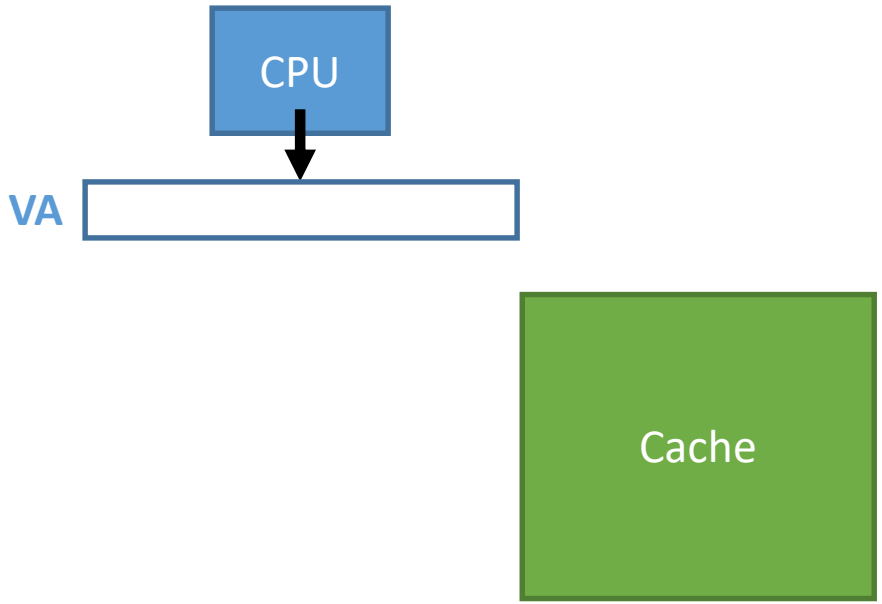
VIPT Cache



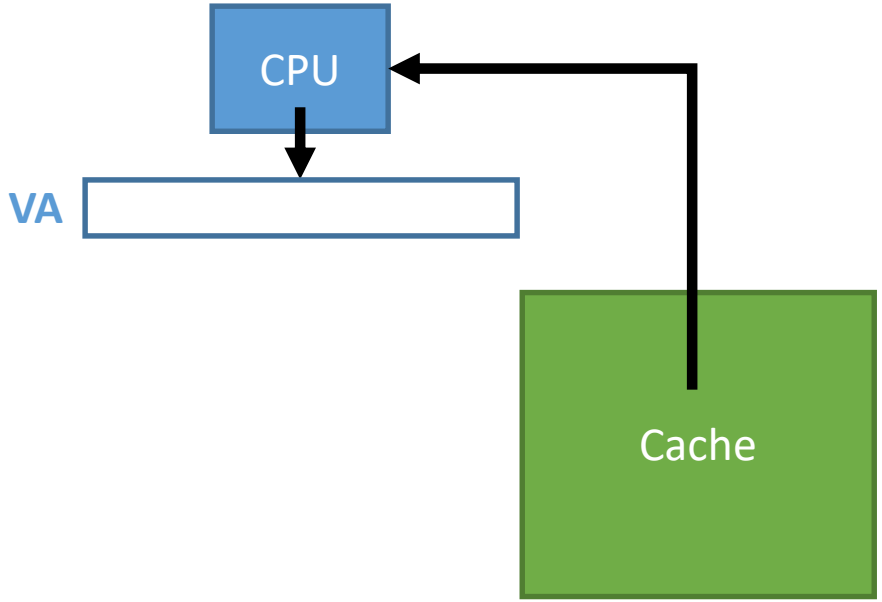
VIPT Cache



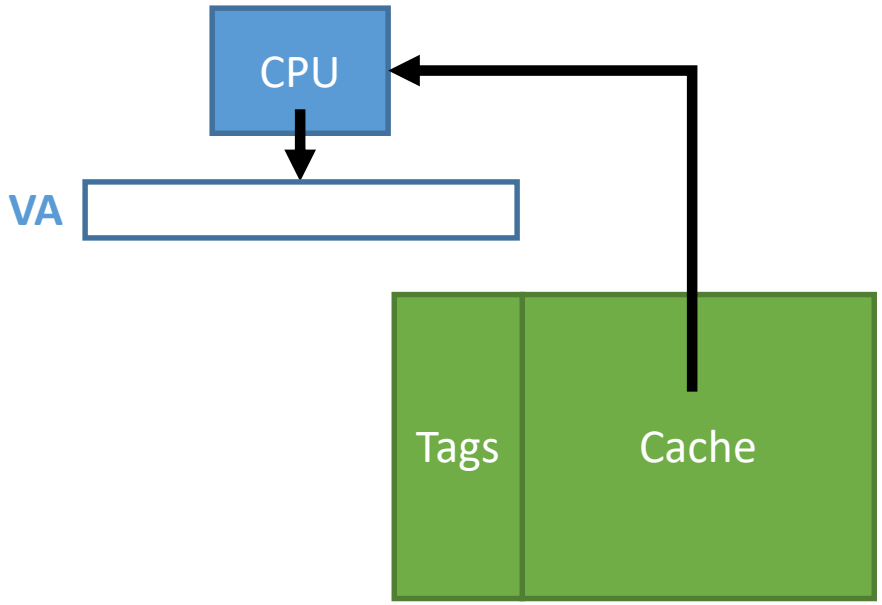
VIPT Cache



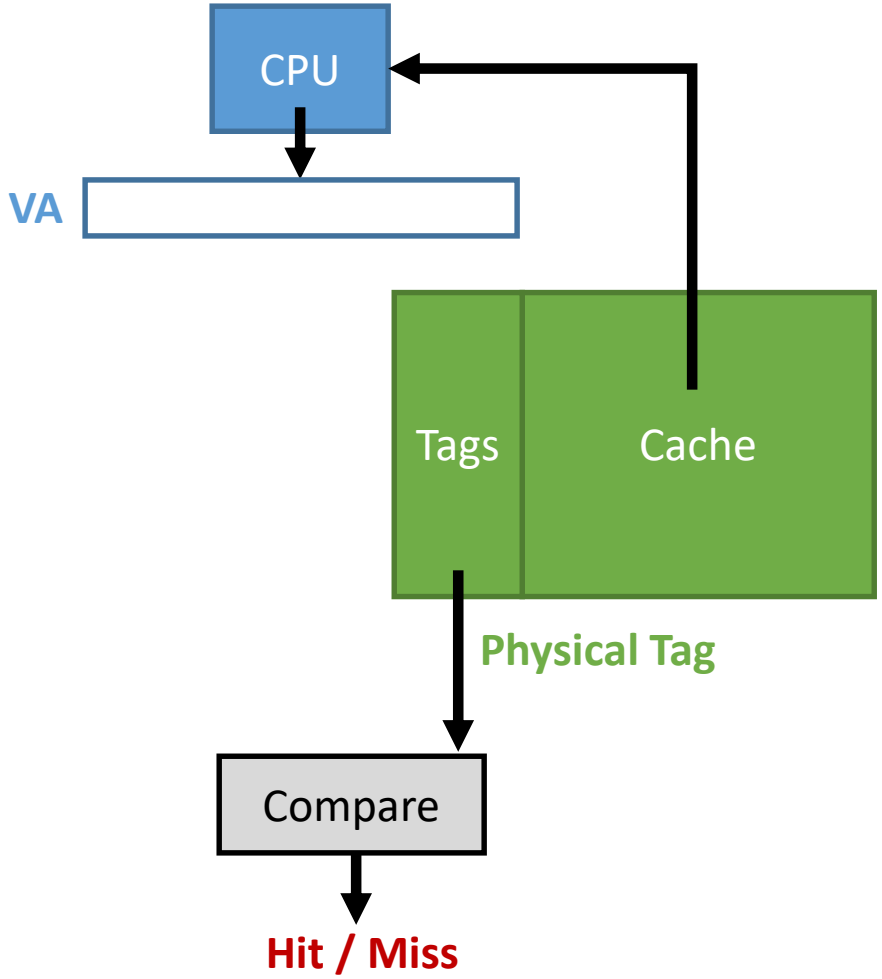
VIPT Cache



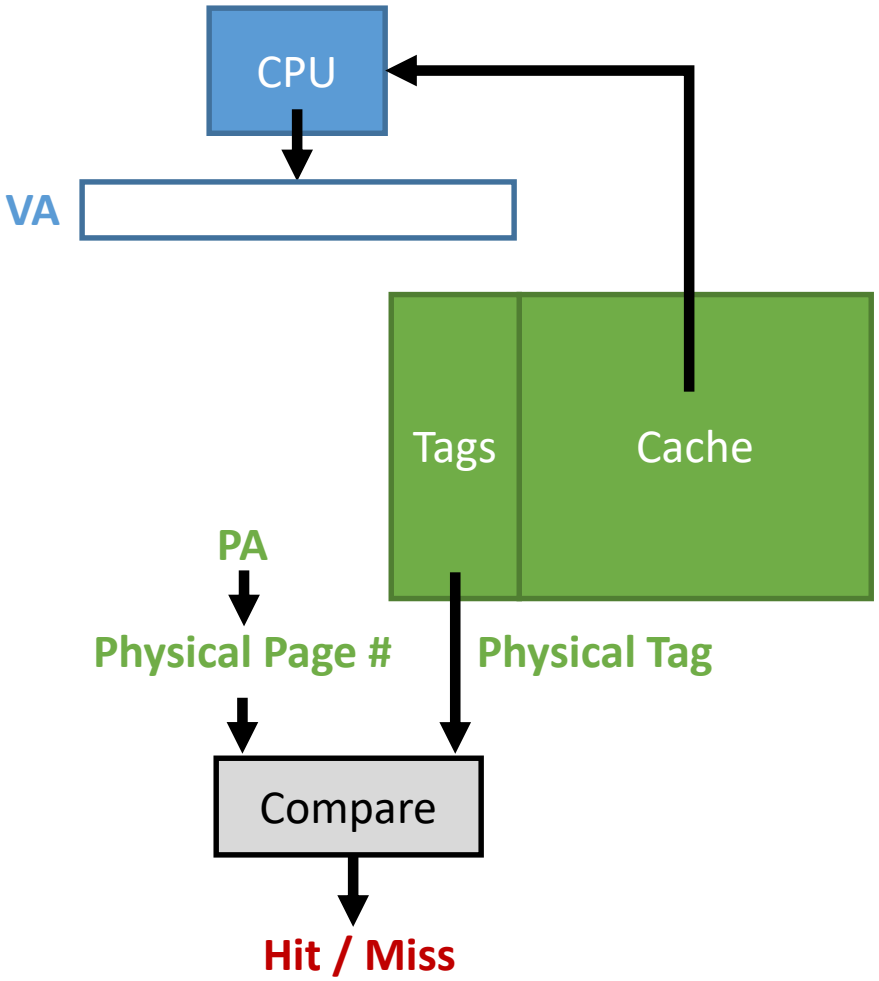
VIPT Cache



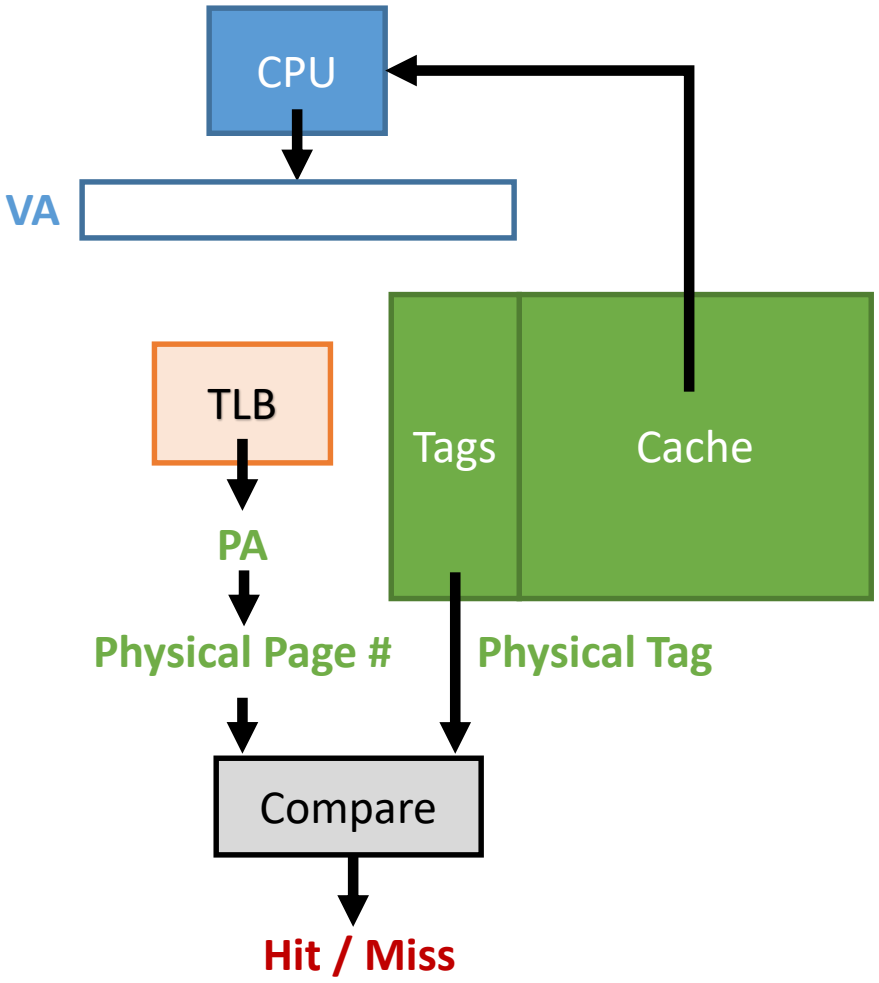
VIPT Cache



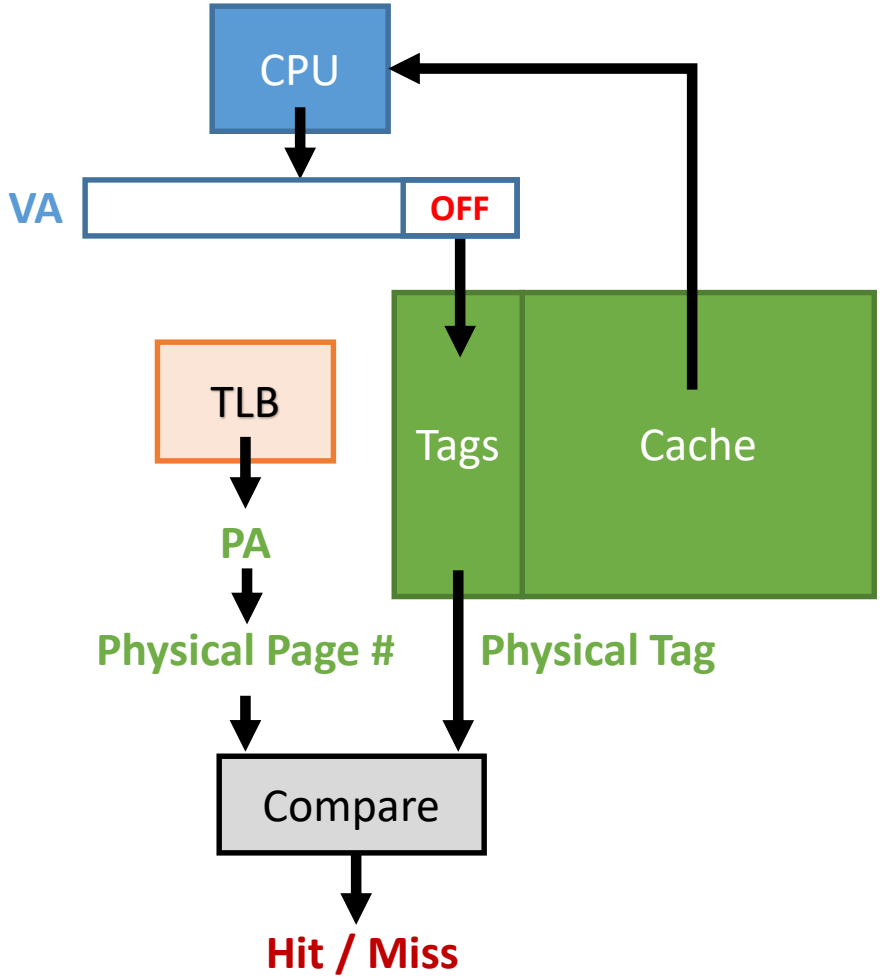
VIPT Cache


CPU
VA
Tags
Cache
PA
Physical Page #
Physical Tag
Compare
Hit / Miss

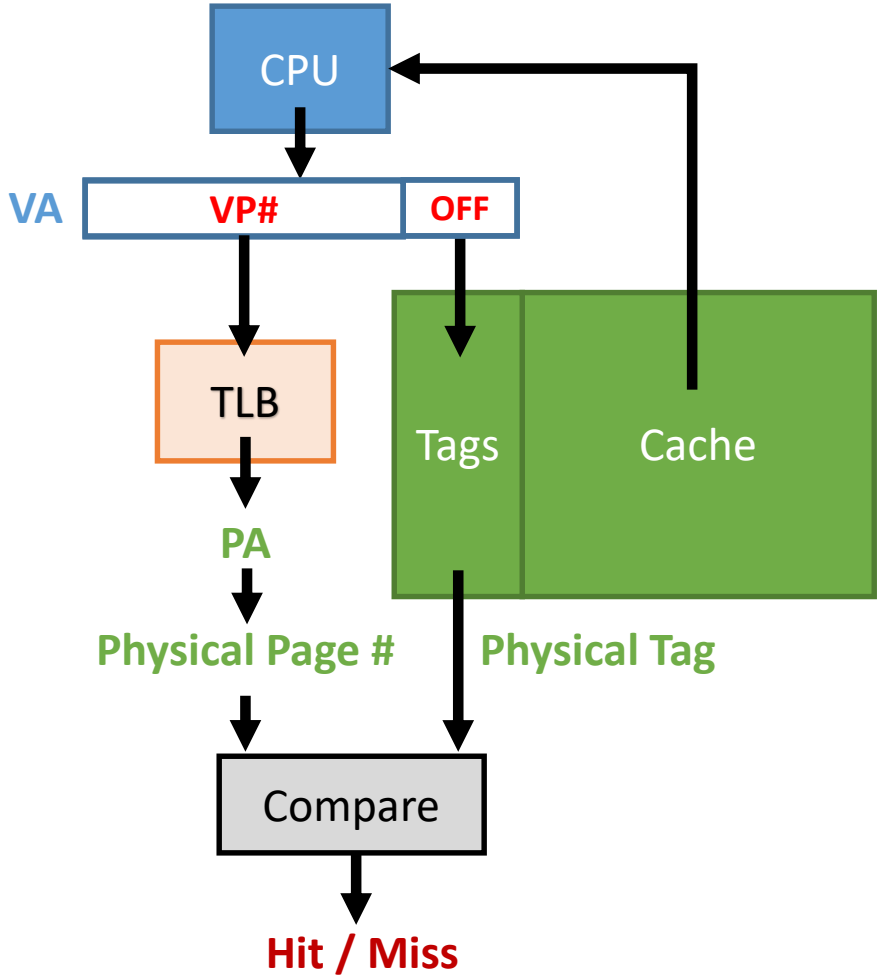
VIPT Cache


CPU
VA
TLB
PA
Physical Page #
Tags
Cache
Physical Tag
Compare
Hit / Miss

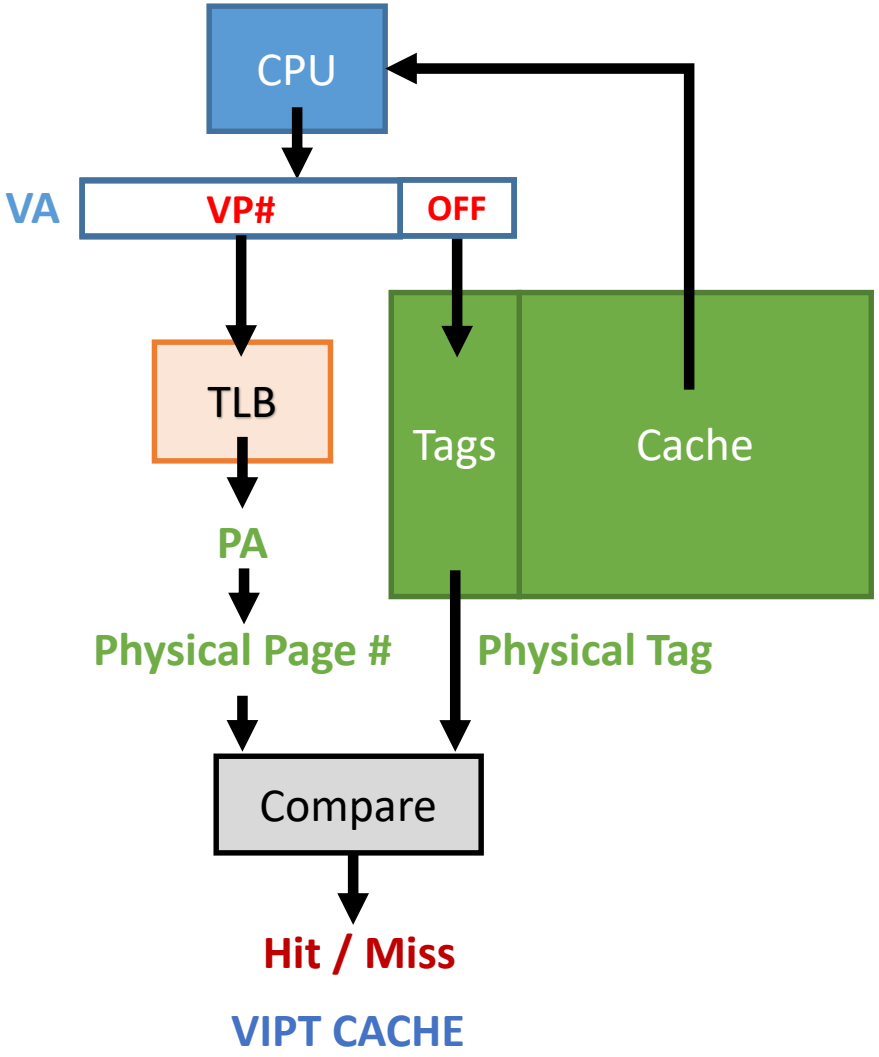
VIPT Cache


CPU
VA
OFF
TLB
Tags
Cache
PA
Physical Page #
Physical Tag
Compare
Hit / Miss

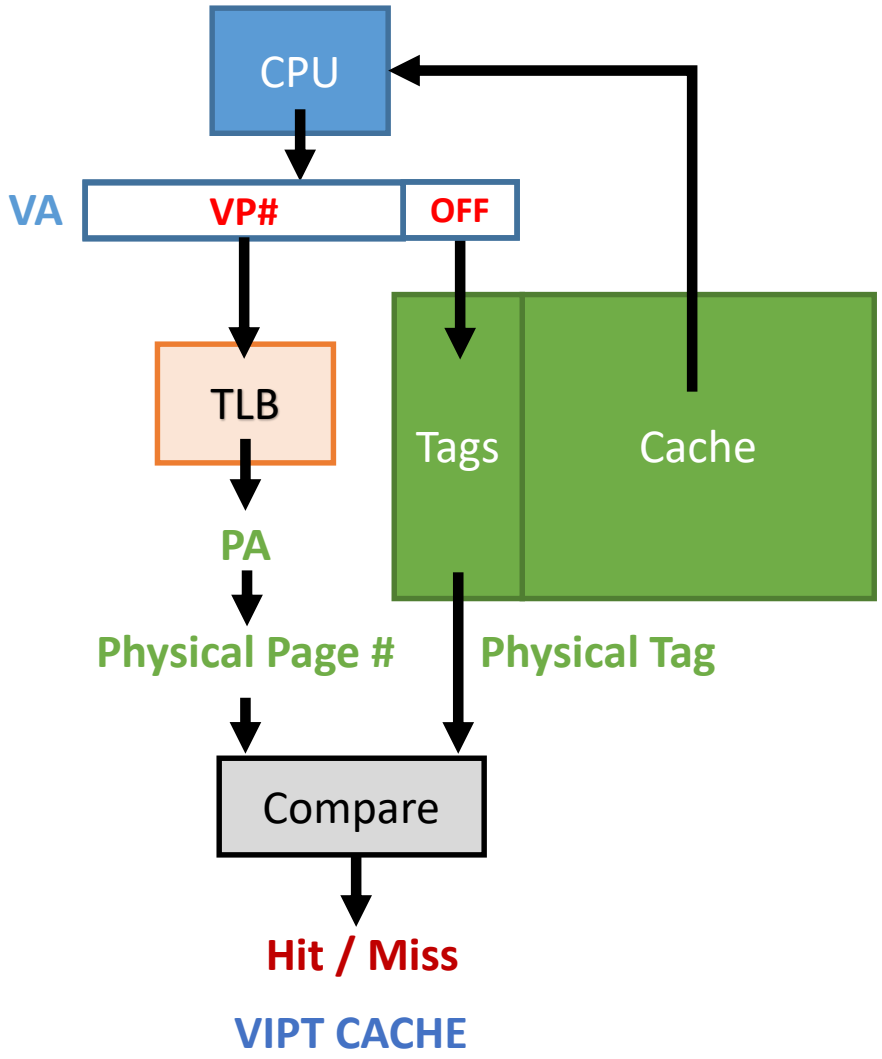
VIPT Cache


CPU
VA
VP#
OFF
TLB
Tags
Cache
PA
Physical Page #
Physical Tag
Compare
Hit / Miss

VIPT Cache

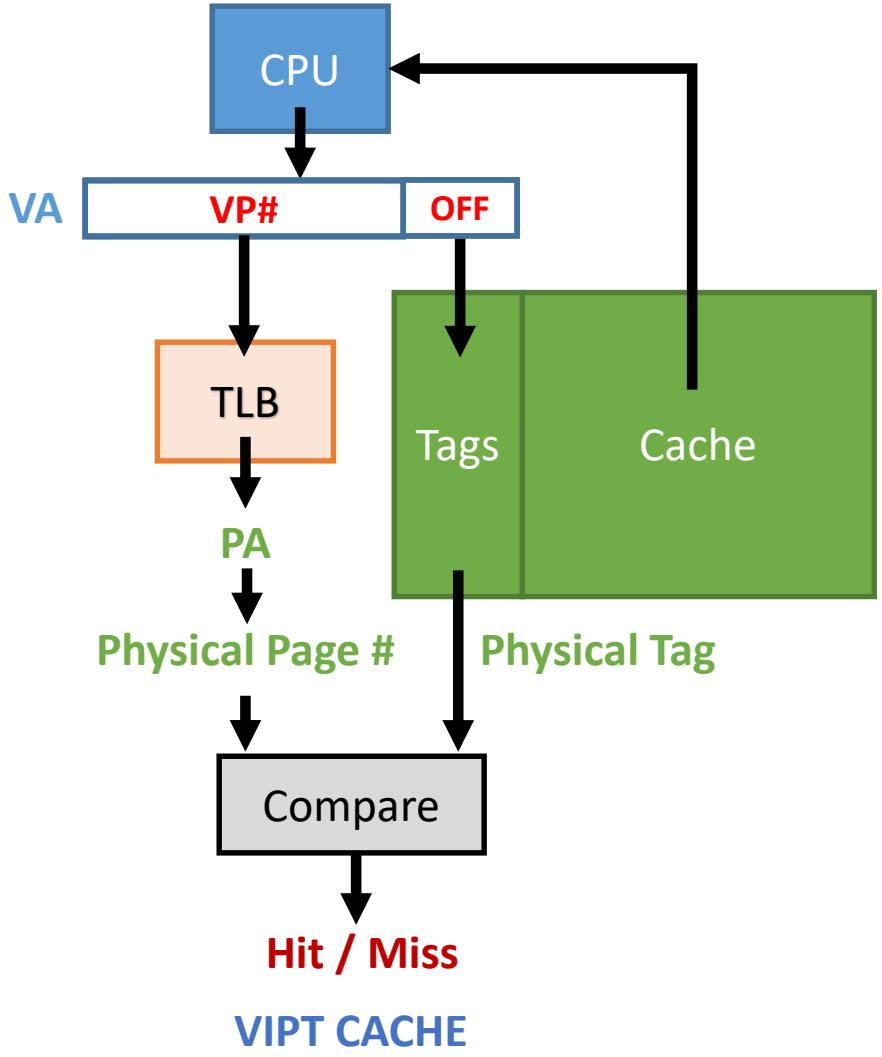

CPU
VA
VP#
OFF
TLB
Tags
Cache
PA
Physical Page #
Physical Tag
Compare
Hit / Miss
VIPT CACHE

VIPT Cache



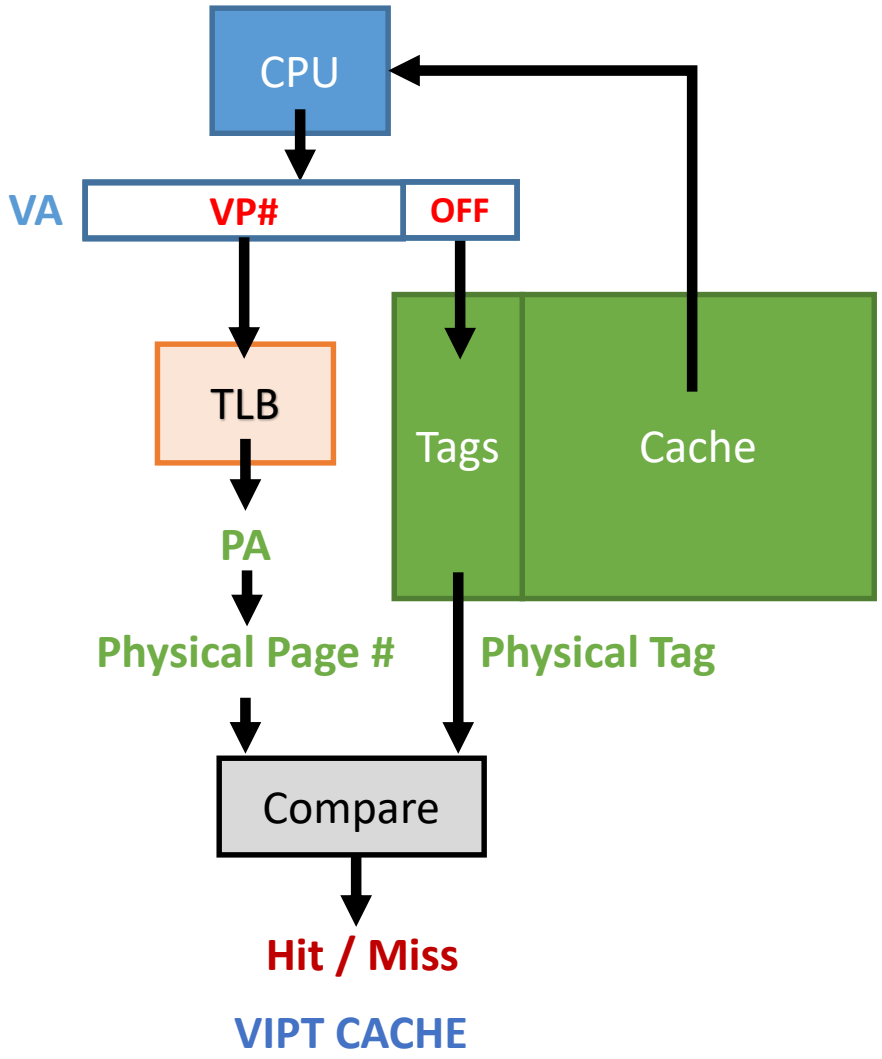
- **VIPT Cache**: Simultaneously translate address [VA→PA] in TLB and look up data in cache

VIPT Cache



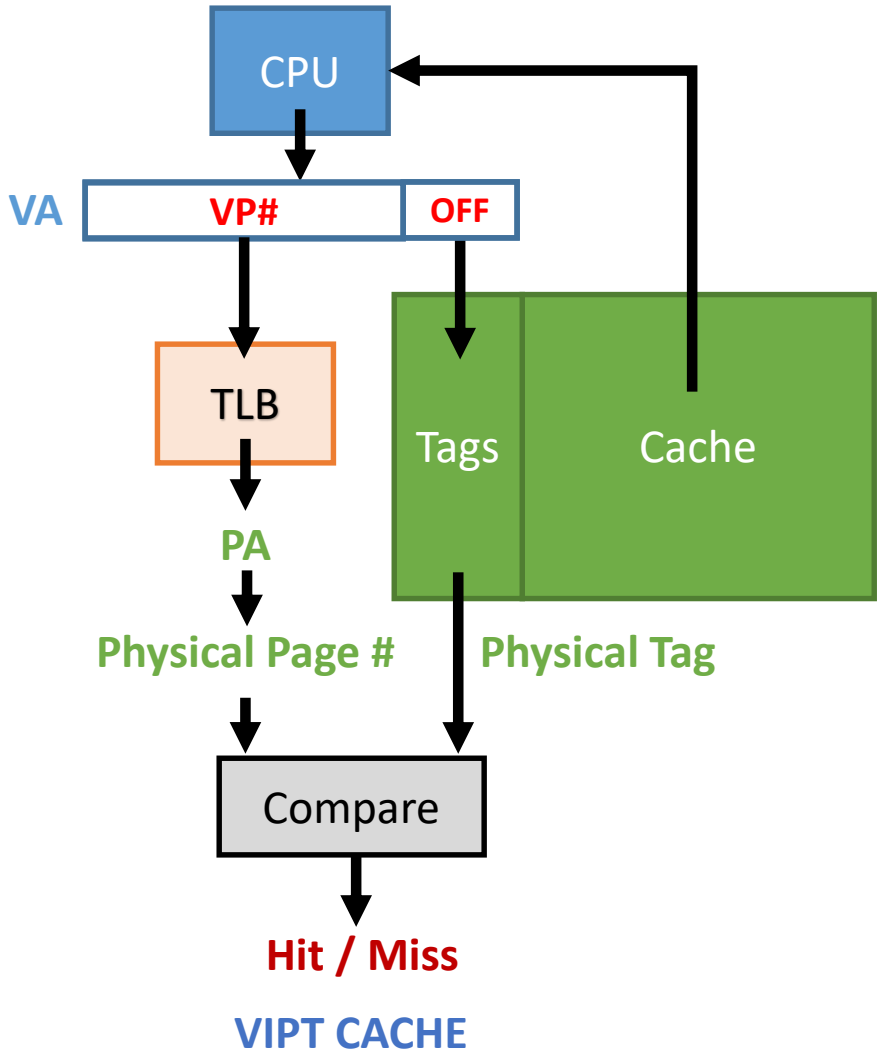
- **VIPT Cache**: Simultaneously translate address [VA→PA] in TLB and look up data in cache
  - Index **TLB** using Virtual Page Number

VIPT Cache



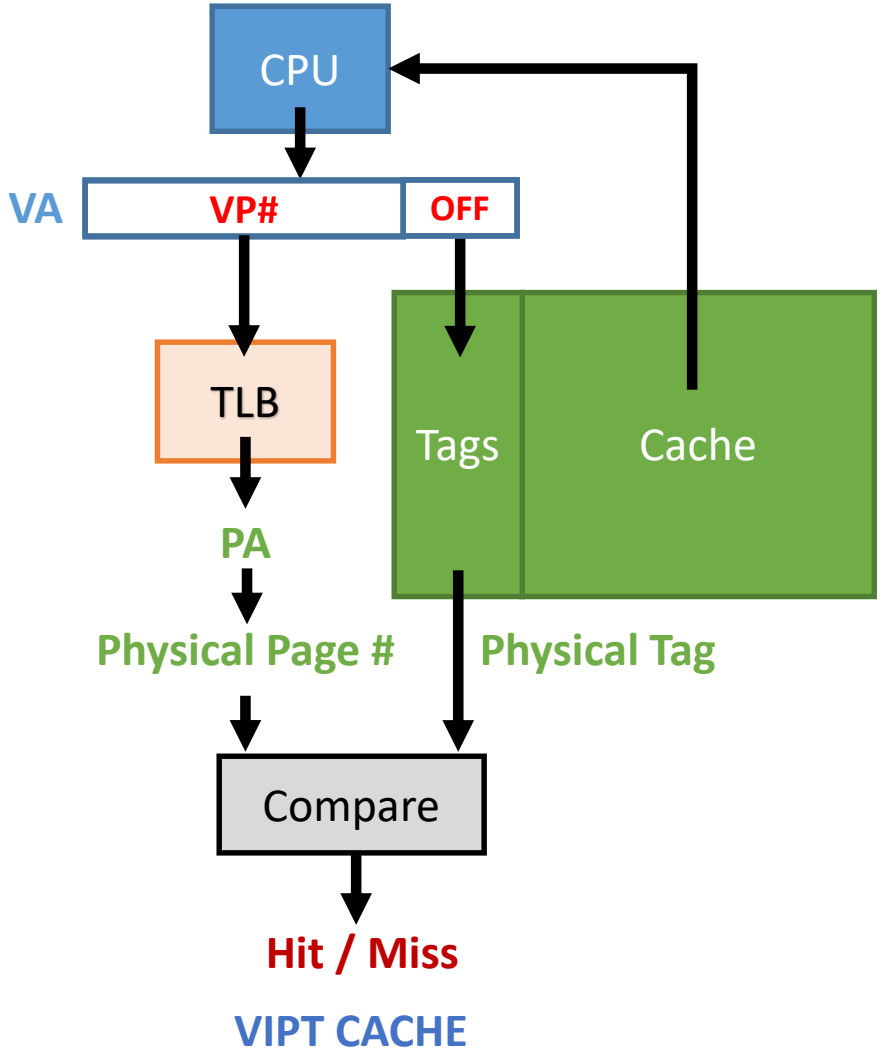
- **VIPT Cache**: Simultaneously translate address [VA→PA] in TLB and look up data in cache
  - Index **TLB** using Virtual Page Number
  - Index **cache** using Page Offset

VIPT Cache



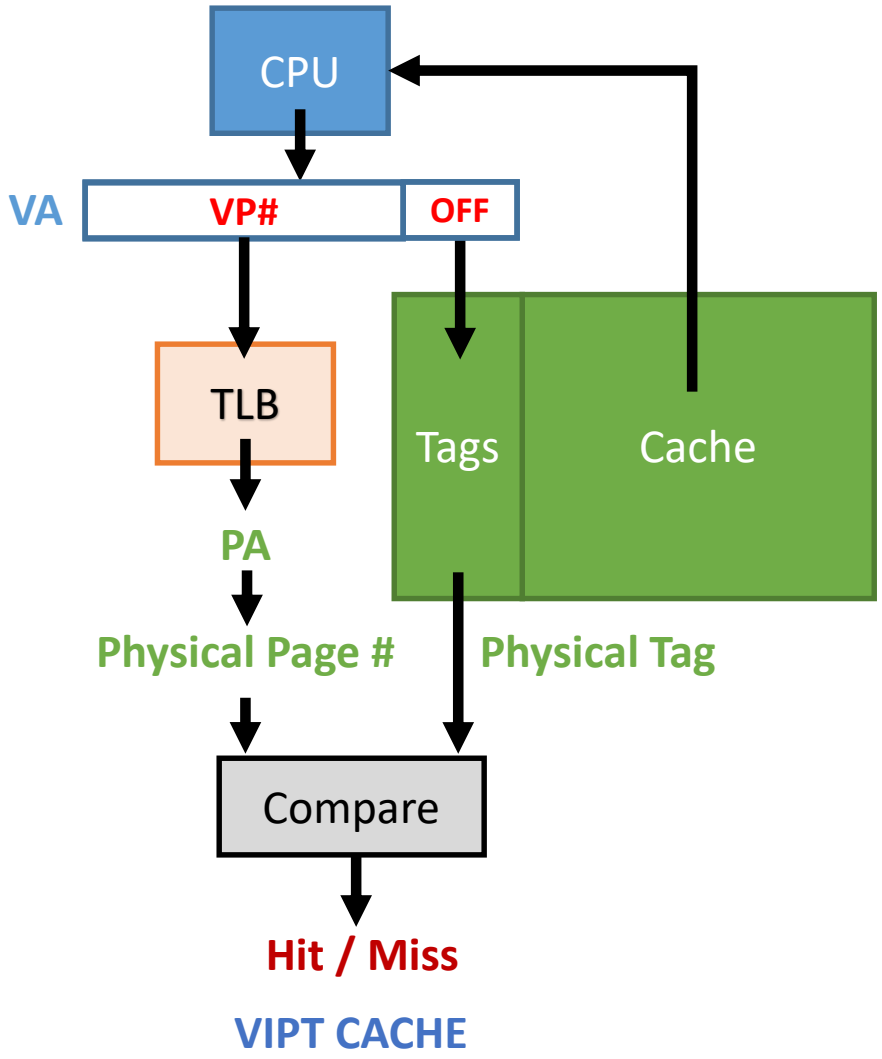
- **VIPT Cache**: Simultaneously translate address [VA→PA] in TLB and look up data in cache
  - Index **TLB** using Virtual Page Number
    - TLB outputs Physical Page
  - Index **cache** using Page Offset

VIPT Cache



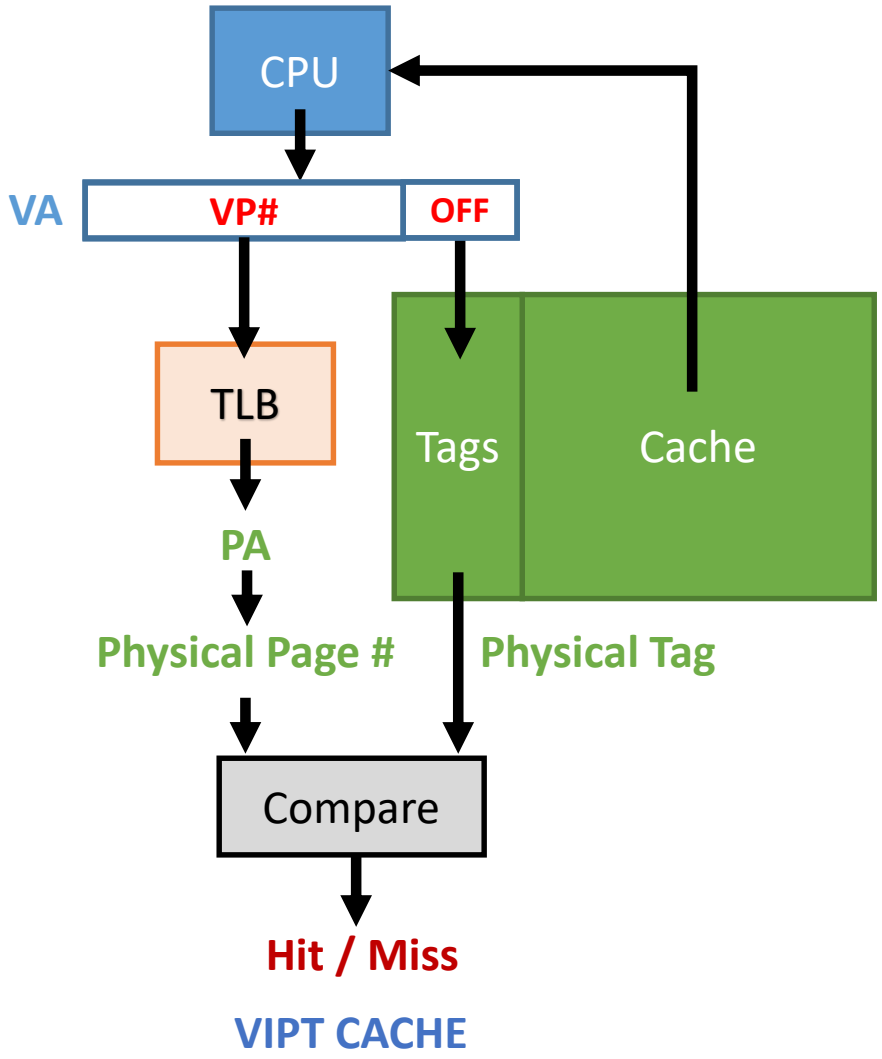
- **VIPT Cache**: Simultaneously translate address [VA→PA] in TLB and look up data in cache
  - Index **TLB** using Virtual Page Number
    - TLB outputs Physical Page
  - Index **cache** using Page Offset
    - Cache outputs Physical Tag

VIPT Cache



- **VIPT Cache**: Simultaneously translate address [VA→PA] in TLB and look up data in cache
  - Index **TLB** using Virtual Page Number
    - TLB outputs Physical Page
  - Index **cache** using Page Offset
    - Cache outputs Physical Tag
  - Hit if Physical Page = Physical Tag

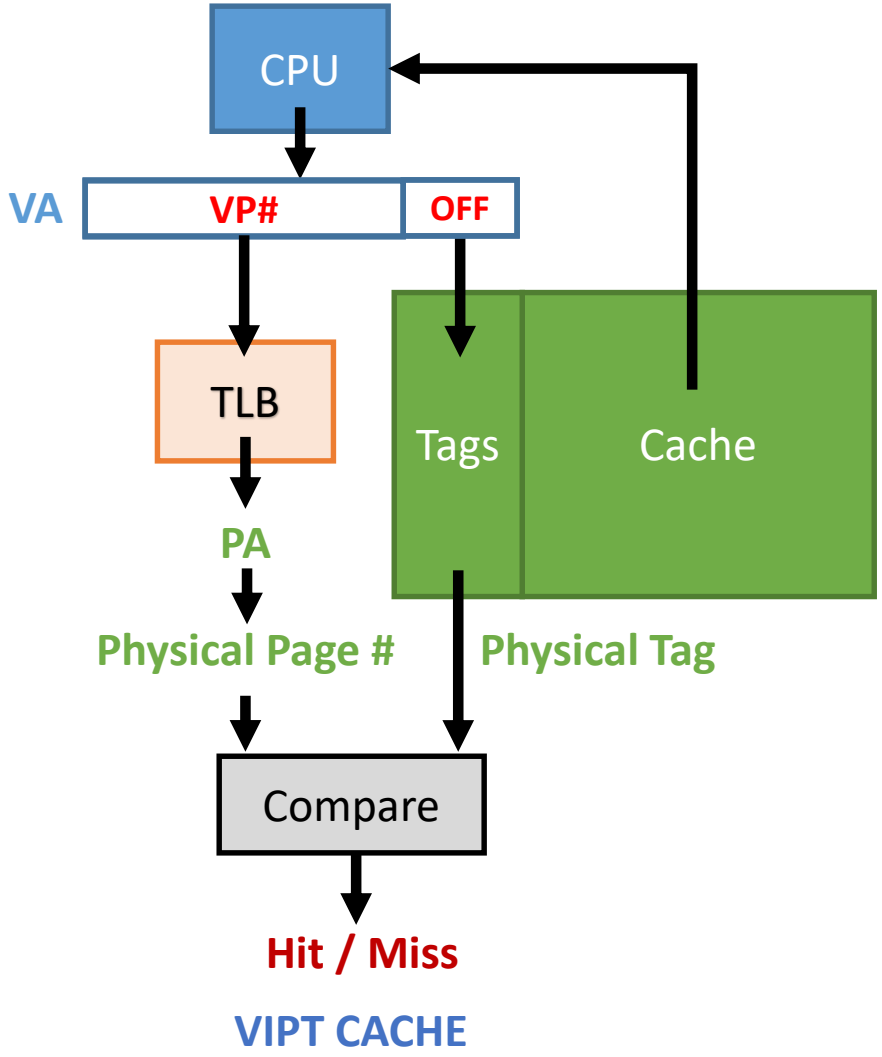
VIPT Cache



- **VIPT Cache**: Simultaneously translate address [VA→PA] in TLB and look up data in cache
  - Index **TLB** using Virtual Page Number
    - TLB outputs Physical Page
  - Index **cache** using Page Offset
    - Cache outputs Physical Tag
  - Hit if Physical Page = Physical Tag

- VIPT cache is fast *and* safe!

VIPT Cache



- **VIPT Cache**: Simultaneously translate address [VA→PA] in TLB and look up data in cache
  - Index **TLB** using Virtual Page Number
    - TLB outputs Physical Page
  - Index **cache** using Page Offset
    - Cache outputs Physical Tag
  - Hit if Physical Page = Physical Tag

- VIPT cache is fast *and* safe!
  - BUT… cache size is limited by page offset

Illustration from the textbook

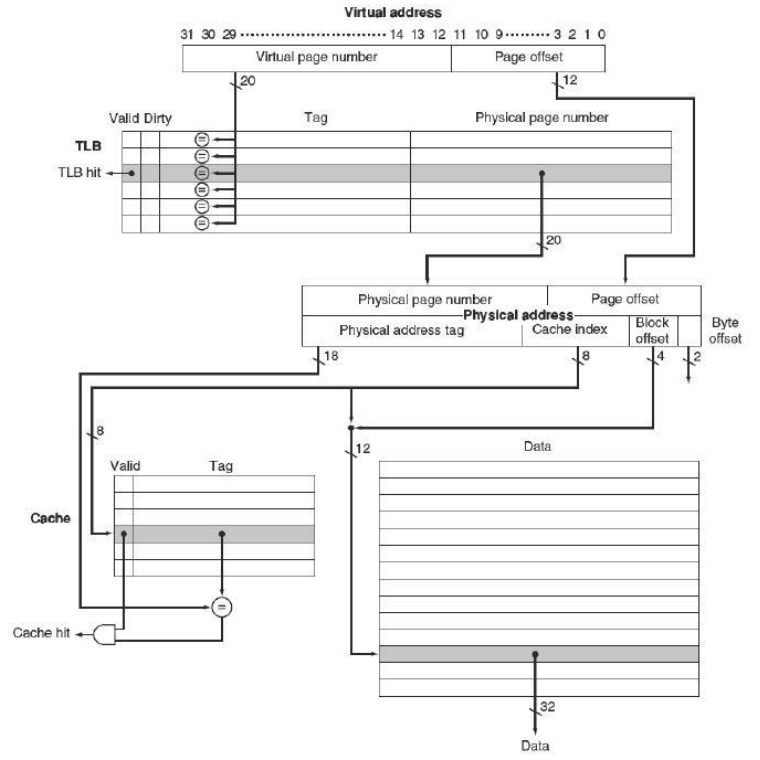


FIGURE 5.30 The TLB and cache implement the process of going from a virtual address to a data item in the Intrinsity FastMATH. This figure shows the organization of the TLB and the data cache, assuming a 4 KiB page size. This diagram focuses on a read; Figure 5.31 describes how to handle writes. Note that unlike Figure 5.12, the tag and data RAMs are split. By addressing the long but narrow data RAM with the cache index concatenated with the block offset, we select the desired word in the block without a 16:1 multiplexor. While the cache is direct mapped, the TLB is fully associative. Implementing a fully associative TLB requires that every TLB tag be compared against the virtual page number, since the entry of interest can be anywhere in the TLB. (See content addressable memories in the *Elaboration* on page 408.) If the valid bit of the matching entry is on, the access is a **TLB hit**, and bits from the physical page number together with bits from the page offset form the index that is used to access the cache.

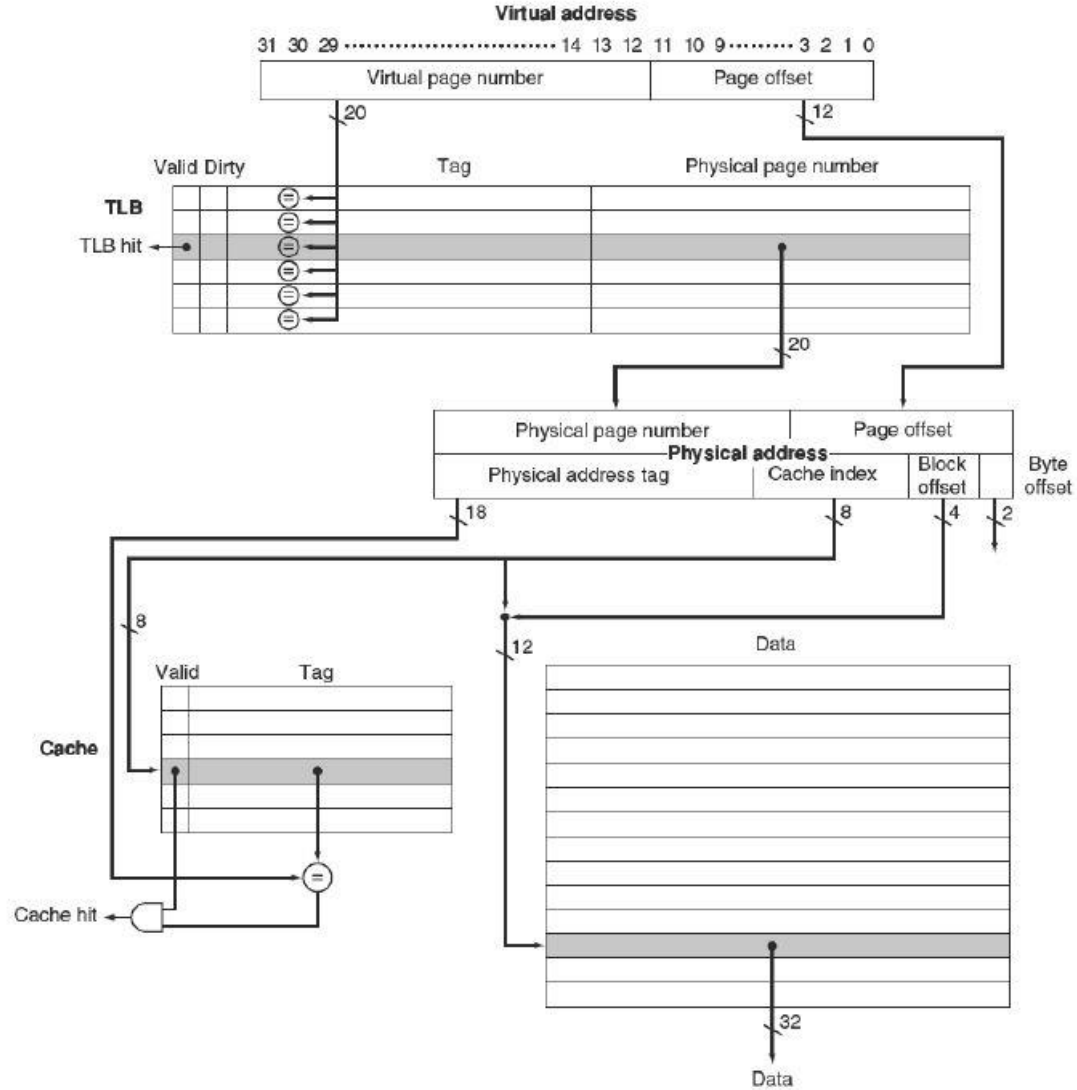


FIGURE 5.30 The TLB and cache implement the process of going from a virtual address to a data item in the Intrinsity FastMATH. This figure shows the organization of the TLB and the data cache, assuming a 4 KiB page size. This diagram focuses on a read; Figure 5.31 describes how to handle writes. Note that unlike Figure 5.12, the tag and data RAMs are split. By addressing the long but narrow data RAM with the cache index concatenated with the block offset, we select the desired word in the block without a 16:1 multiplexor. While the cache is direct mapped, the TLB is fully associative. Implementing a fully associative TLB requires that every TLB tag be compared against the virtual page number, since the entry of interest can be anywhere in the TLB. (See content addressable memories in the *Elaboration* on page 408.) If the valid bit of the matching entry is on, the access is a **TLB hit**, and bits from the physical page number together with bits from the page offset form the index that is used to access the cache.

Quiz: VIPT Caches

Q: With 4kB pages, how many Bytes can a direct-mapped VIPT cache store?

I. 4 kB
II. 8 kB
III. 8 MB
IV. 800 kB
V. 400 MB

Quiz: VIPT Caches

Q: With 4kB pages, how many Bytes can a direct-mapped VIPT cache store?

I. 4 kB
II. 8 kB
III. 8 MB
IV. 800 kB
V. 400 MB

A: 4 kB

We can only use the page offset bits to index the virtual cache. With 4 kB pages we have 12 bits of page offset. This explains why level 1 caches are so small.

If we increase set-associativity we can make this seem larger!

Cache Size, Page Size, Associativity

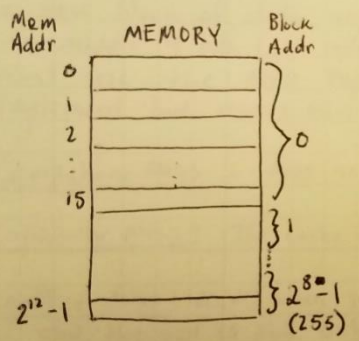
- So far we have seen direct mapped caches and TLBs
- What if we increase associativity?

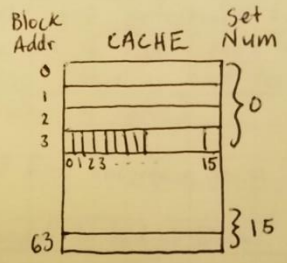
$$\frac{Cache\ Size}{Page\ Size} = Cache\ Associativity$$

COA Exam – My Solution

Assume cache of size 1,024 Bytes, block size of 16 Bytes, and 4-way set associative. The memory address length is 12 bits and is byte-addressable.

Step 1: draw out memory + cache





$\#\text{cache lines} = \frac{\text{cache size}}{\text{block size}} = 2^{10-4} = 64$

$\text{Block Addr} = \left\lfloor \frac{\text{Mem Addr}}{\text{Block Size}} \right\rfloor$

$\#\text{cache sets} = \frac{\#\text{cache lines}}{\text{associativity}} = \frac{64}{4} = 16$

$\#\text{Block Addr's} = \frac{\text{Memory size}}{\text{Block size}} = 2^{12-4} = 2^8 = 256$

References

- David Black-Schaffer: Lecture Series on Virtual Memory
- Patterson, Hennessy: Computer Organization and Design: the Hardware/Software Interface
- Intel Hardware Data-Sheets
- Linux: Anatomy of a Program in Memory