ADVANCED CACHE OPTIMIZATIONS

Prof Debdeep Mukhopadhyay

Computer Sc &. Engg,

Indian Institute of Technology Kharagpur

CACHE PERFORMANCE

- Average Memory Access Time (AMAT)=HIT TIME + MISS RATE x MISS PENALTY
  - Reduce Hit Time
  - Reduce Miss Rate
  - Reduce Miss Penalty

REDUCE HIT TIME

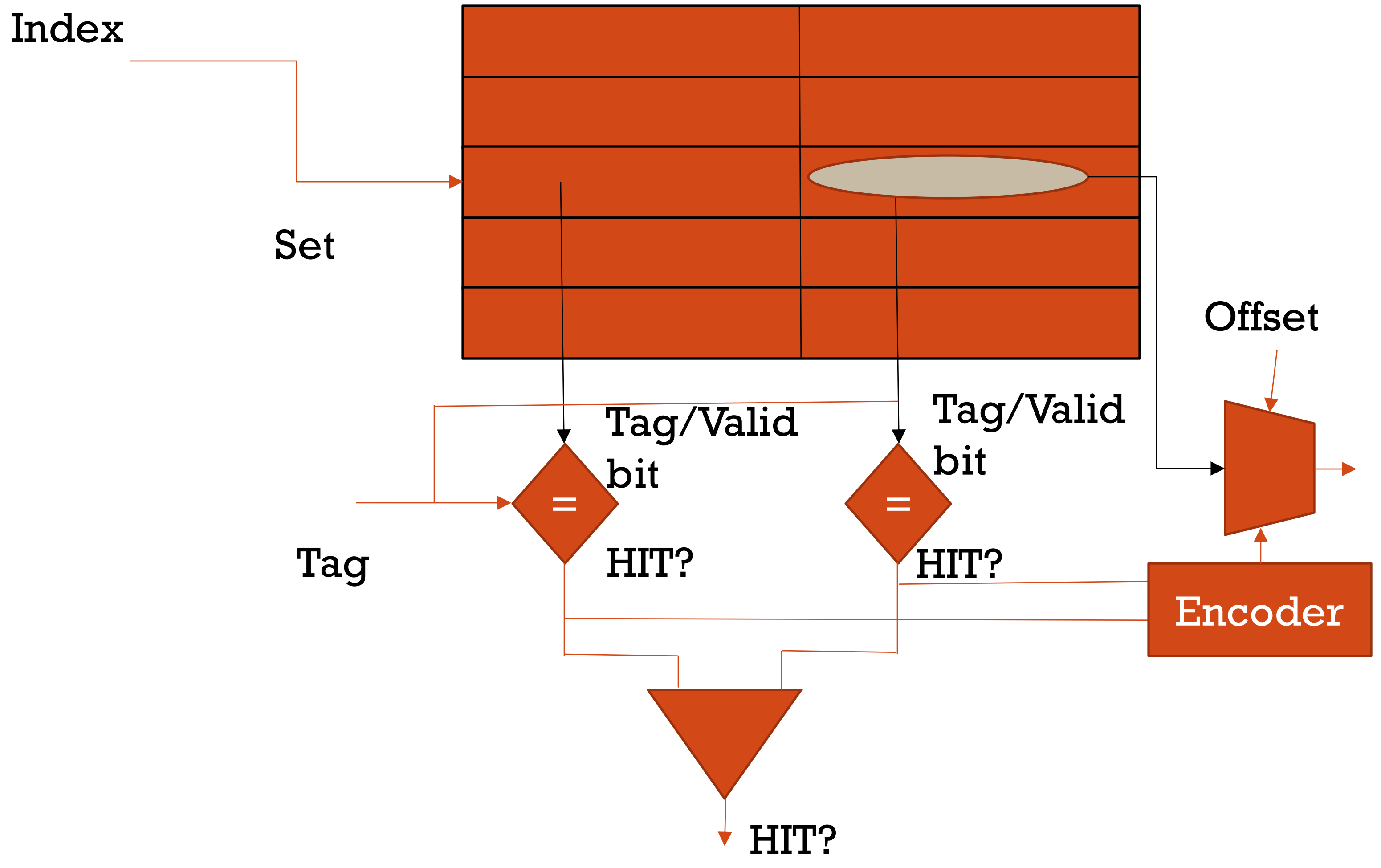
- Reduce Cache Size (But bad for Miss Rate)
- Reduce Cache Associativity (Bad for Miss Rate)
- Overlap Cache Hit with Another Hit
- Overlap Cache Hit with TLB Hit
- Optimize Lookup for Common Case
- Maintain Replacement State more efficient

PIPELINED CACHE

- Basic Idea is to overlap one hit with another.
  - Introduce pipelining in cache
- Consider an access to the cache in cycle N, results in Hit.
- If another access comes in cycle N+1, and also should result in Hit.
- But if the access to the cache takes multiple cycles, then the second access has to wait.
- Thus,

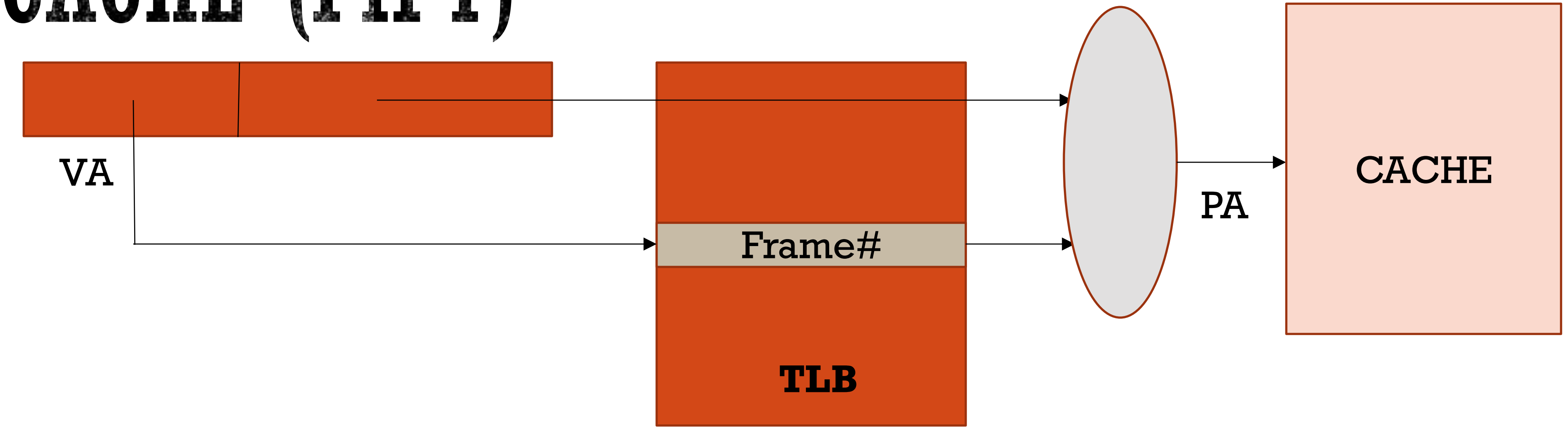
Hit Time (for second access)=Actual Hit Time + Wait Time

PIPELINING THE CACHE



- Index→Set
- Reading out the tags and valid bits to know if the access is a hit
- Combining the respective hits to determine whether we have an overall hit
- Using the offset to select the corresponding byte.
- Usually, the access is divided into stages.
- The stages can be forming the different cycles in the pipelined cache.
- An L1 cache is usually pipelined into 3 stages.

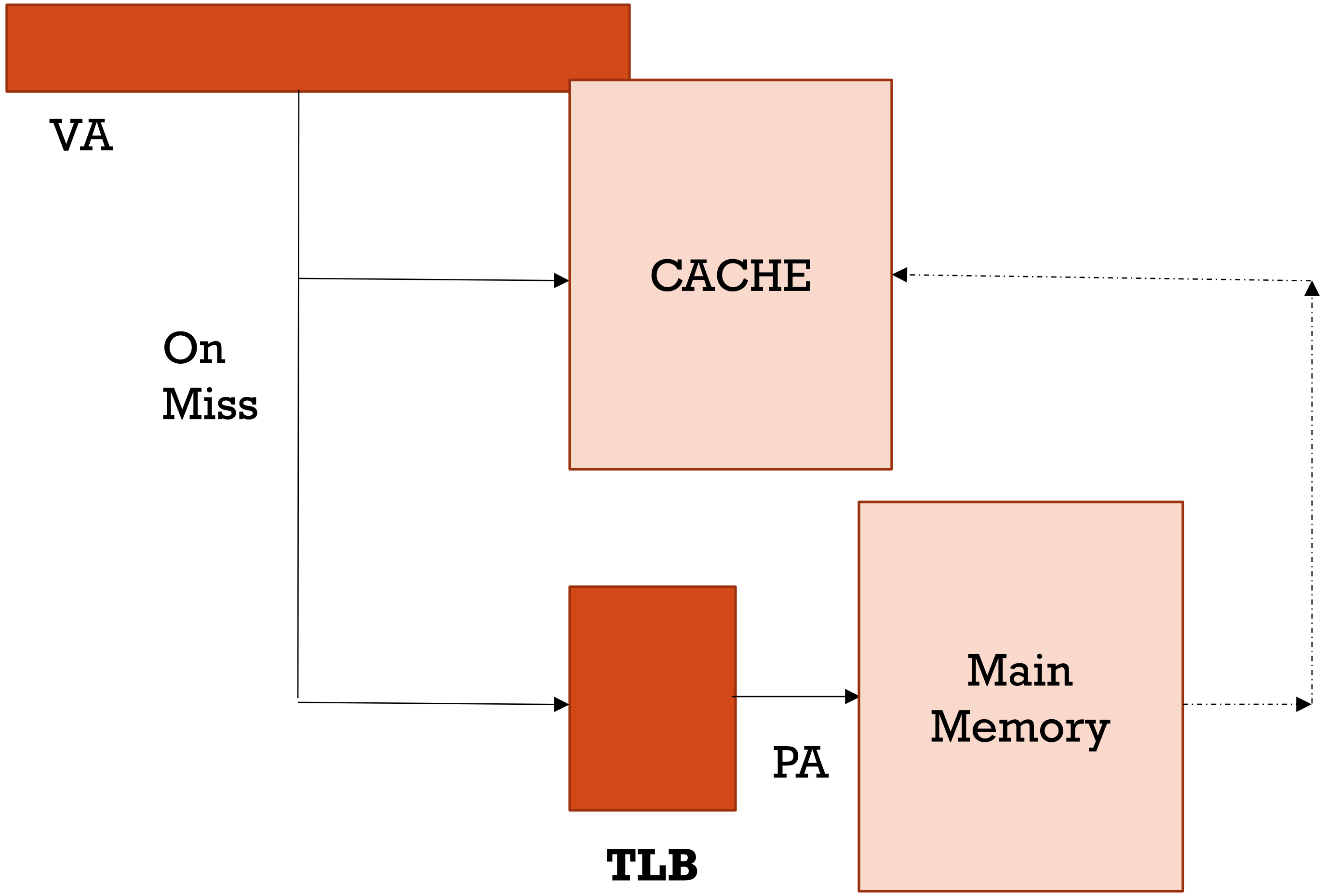
PHYSICALLY INDEXED PHYSICALLY TAGGED CACHE (PIPT)



- Hit time is affected because we have to access the TLB before the cache.
- Assuming the TLB and cache each takes 1 clock cycle, we need 2 cycles to get the data.

A cache that is accessed with the Physical Address (PA) is called Physically Accessed Cache or Physically Indexed Physically Tagged (PIPT) cache.

VIRTUALLY ACCESSED CACHE



On Miss a VA→PA mapping is obtained to access the main memory and update the cache.

Hit Time is same as that of Cache Hit Time

On Cache Hit, it is expected there is no VA to PA translation.

PROBLEMS WITH VIRTUALLY ACCESSED CACHE: PROBLEM 1

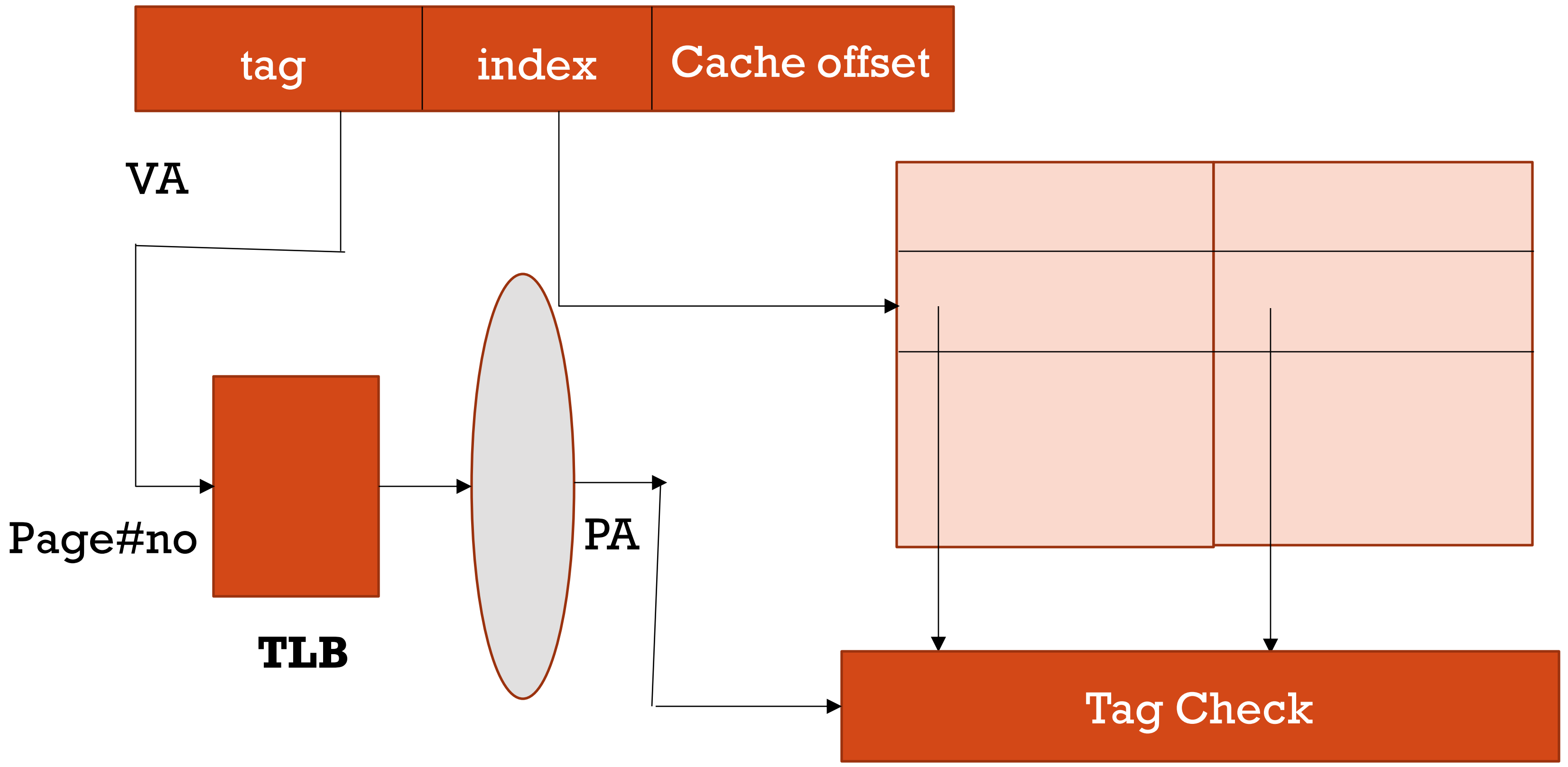
- TLB in addition to the frame nos. also has certain bits to store permissions, like valid bits etc.

- So, even though we do not need to get the VA→PA mapping through the TLB, the processor needs to access the TLB to get the permissions that tell us whether we are allowed to read, write or execute that location.

PROBLEMS WITH VIRTUALLY ACCESSED CACHE: PROBLEM 2

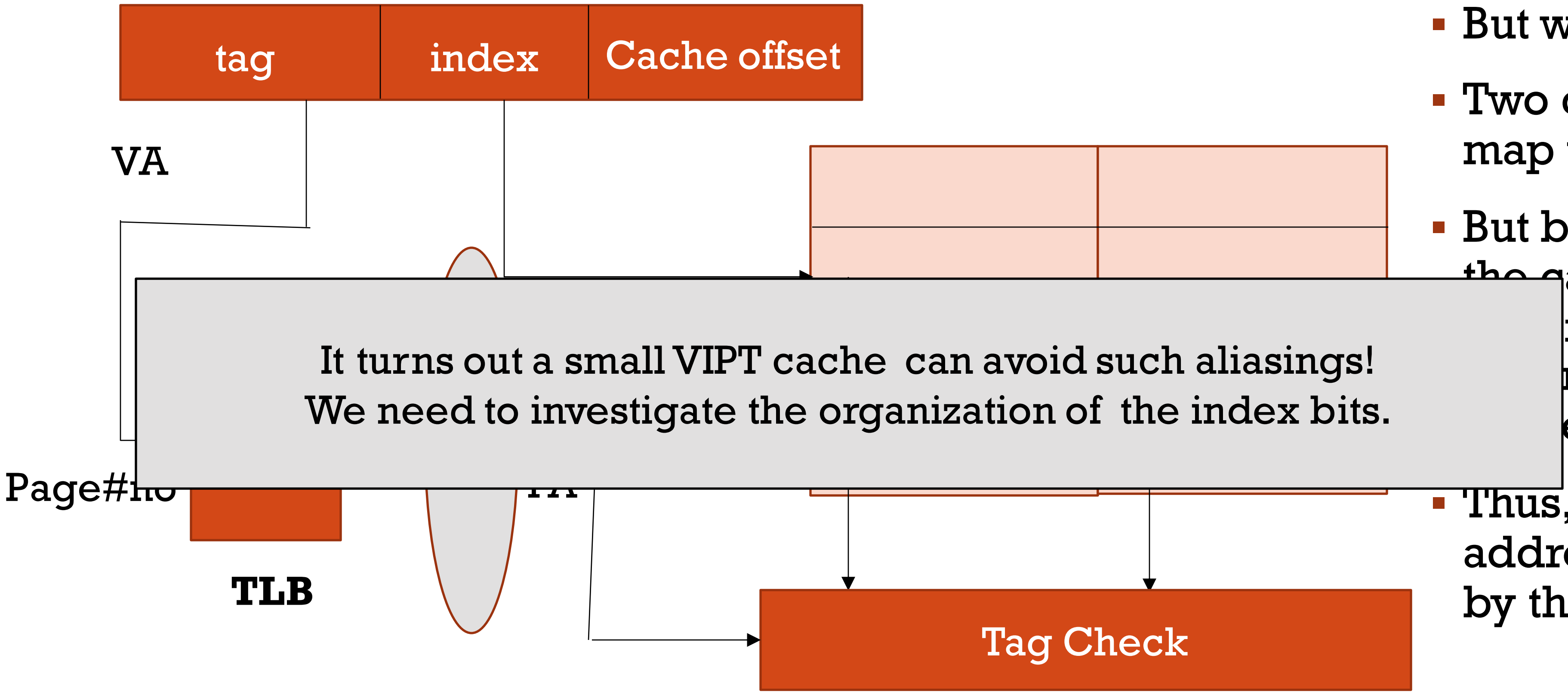
- A bigger problem is in the case of context switch.
  - VA is specific to a process.
  - One process runs and we fill the cache with some data
  - Another process executes, the virtual addresses that may overlap with the addresses from the previous process, should be targeted to different data.
  - It should be translated to different physical address in the TLB etc.
  - But the cache only sees the VA!
- Thus, we need to flush the cache on context switch.
- Every time there is a context switch there is a burst of cache misses.

Virtually Indexed Physically Tagged (VIPT) Cache



- Hit Time=Cache Hit Time (as the TLB access is typically faster)
- Context Switch?
  - The tag check is done with the physical tag.
  - So, the VA for a different process may map to the same cache set.
  - But, the actual tag will not match, and we will have a miss.
  - Data would be brought from the main memory to the cache.
  - Thus, it will operate like the physical cache.

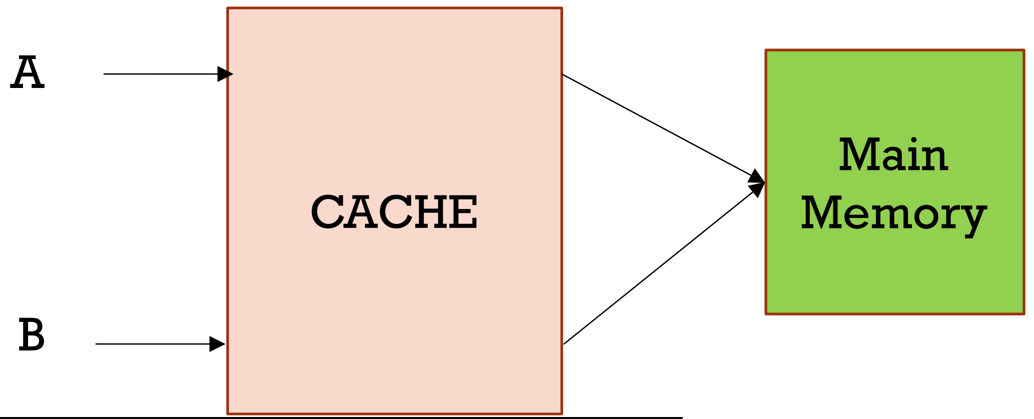
VIRTUALLY INDEXED PHYSICALLY TAGGED (VIPT) CACHE – ALIASING?



It turns out a small VIPT cache can avoid such aliasings!
We need to investigate the organization of the index bits.

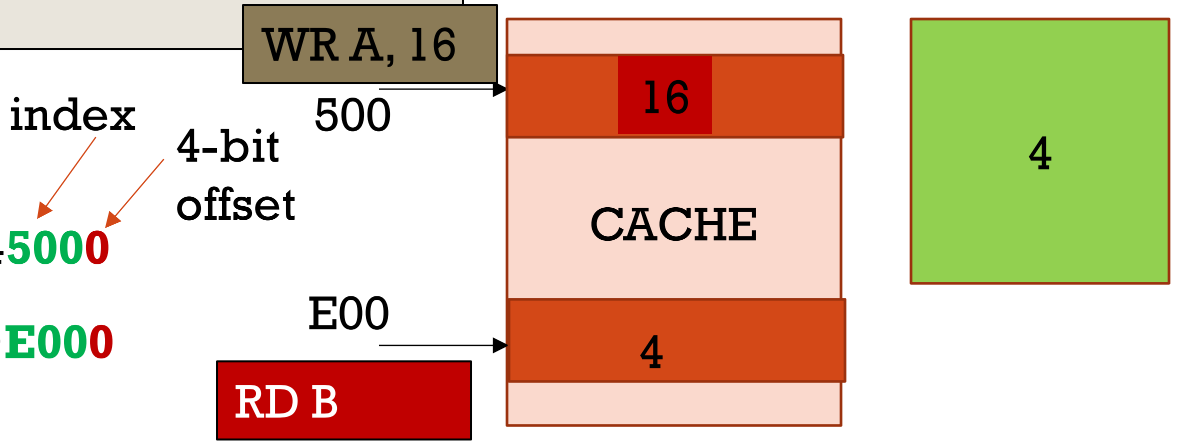
- But what about aliasing?
- Two different VAs can map to the same PA.
- But because we access the cache with the VAs, might end up in rent sets of the e!
- Thus, write to one address will not be seen by the other.

ALIASING EXAMPLE IN A VIPT



mmap(A,4096,fd,0) – creates a new mapping in the VA space of the calling process. It puts physical memory at a given location in program virtual memory, and optionally copies a file's contents there.
See:
https://www.cs.uaf.edu/2016/fall/cs301/lecture/11_02_mmap.html
Likewise, mmap(B,4096,fd,0)

Consider, a 64kB, Direct Mapped Cache with 16-byte Block Size.
Hence, no of index bits=12, as $(64 \times 2^{10})/16=2^{12}$

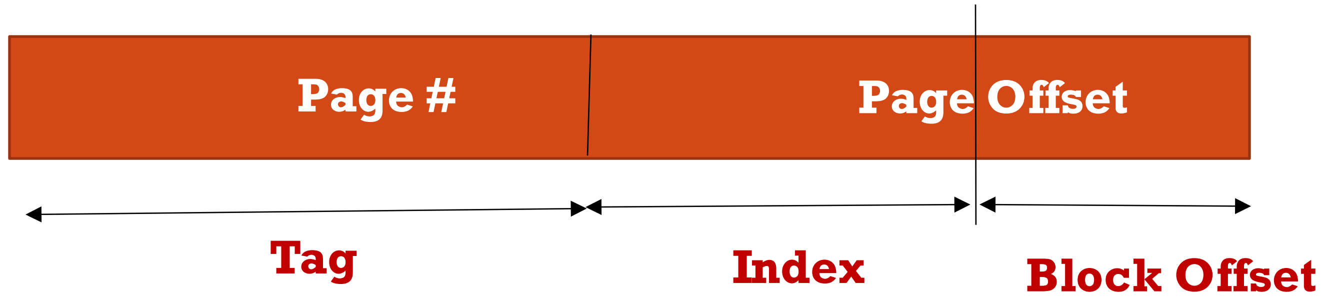


Say, A=0x12345000

Say, B=0ABCDE000

Write Back Cache

AVOIDING THE ALIASING



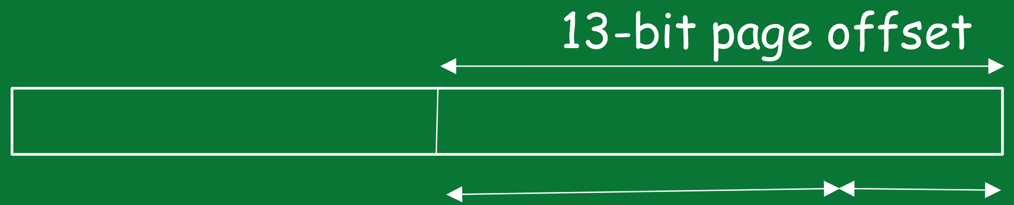
- Consider, 4 kB page size => 12-bit page offset
- 32 Byte Cache Block => 5-bit block offset
- Thus, if the cache has less than (12-5)=7 bits, ie. 128 sets then the index bits all come from the "Page Offset".
- For aliasing we are considering two VAs whose PAs are same => Page Offsets are also same.
- Thus, here the index is also same, and both the VAs map to the same set in the cache, hence there is no aliasing!

VIPT QUIZ

- 4-way Set Associative Cache
- 16 Byte Block Size
- 8 kB Page Size, ie. $2^3 x 2^{10}=2^{13}$. So, 13 bit is for Page Offset.
- What is the maximum size of the cache for no aliasing?

VIPT QUIZ

- 4-way Set Associative Cache
- 16 Byte Block Size
- 8 kB Page Size, ie. $2^3 \times 2^{10} = 2^{13}$. So, 13 bit is for Page Offset.
- What is the maximum size of the cache for no aliasing?



Hence, for no aliasing maximum index bits=13-4=9.
Thus, the maximum size of cache is $2^9 \times 2^4 \times 4 = 2^{15}$=32kB.

4-bit Block offset

Interestingly, maximum cache size is 2^{13}x4Bytes.

Page Size

Associativity

REAL VIPT CACHES

CACHE SIZE≤ASSOCIATIVITY X PAGE SIZE
PENTIUM 4: 4 WAY SET ASSOCIATIVE, WITH PAGE SIZE=4KB. THUS, THE. L1 CACHE IS OF SIZE 16KB.
CORE-2, NEHHALEM, SANDY BRIDGE, HASWELL: 8 WAY SET ASSOCIATIVE, PAGE SIZE=4KB. THUS, L1 CACHE IS OF SIZE 32KB.
SKY LAKE: 16 WAY SET ASSOCIATIVE. THUS CACHE SIZE IS 64KB.