

COM515 Advanced Computer Architecture

Lecture 8. Memory Hierarchy Design II

Prof. Taeweon Suh

Computer Science Education

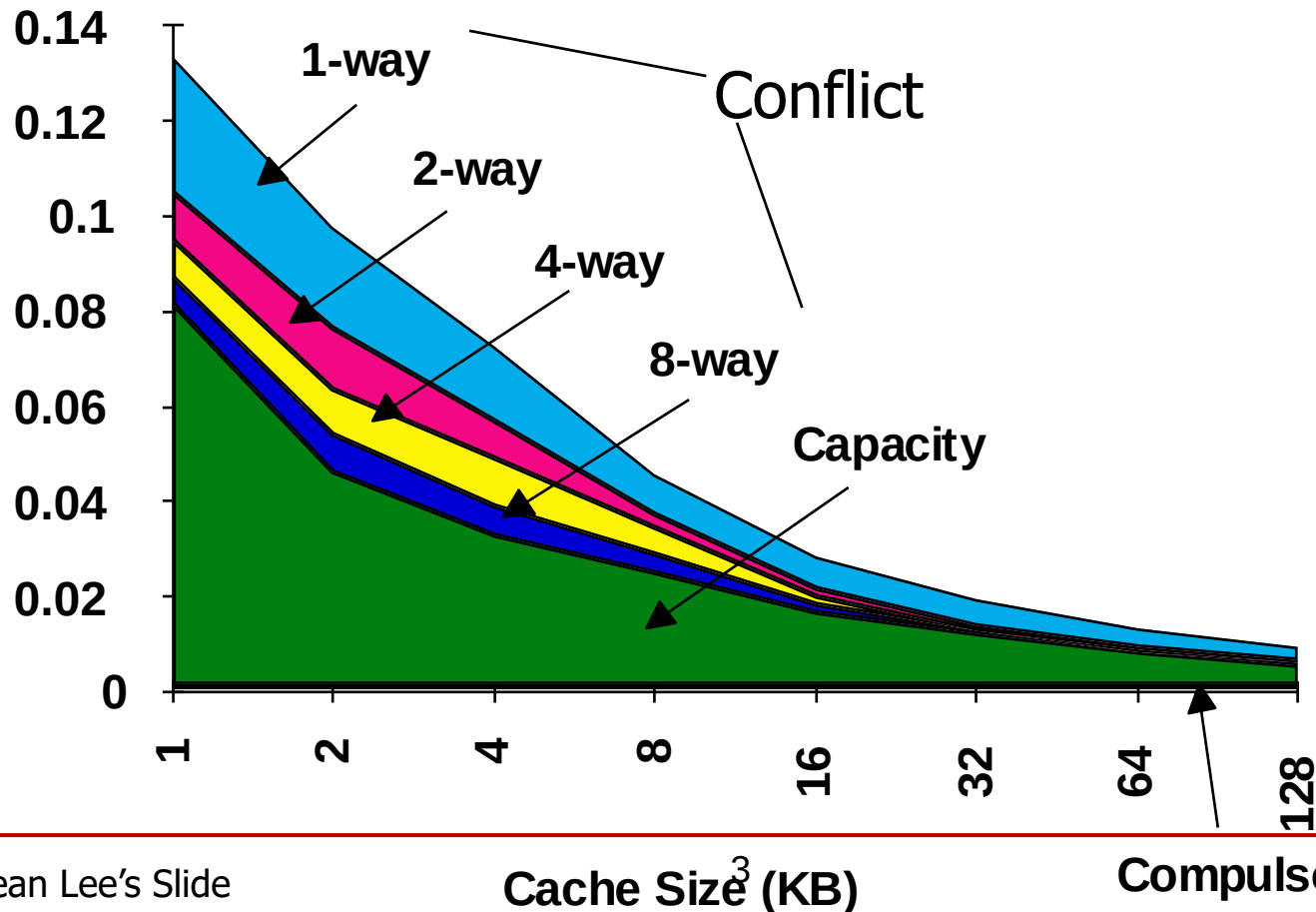
Korea University

Topics to be covered

- Cache Penalty Reduction Techniques
 - Victim cache
 - Assist cache
 - Non-blocking cache
 - Data Prefetch mechanism
- Virtual Memory

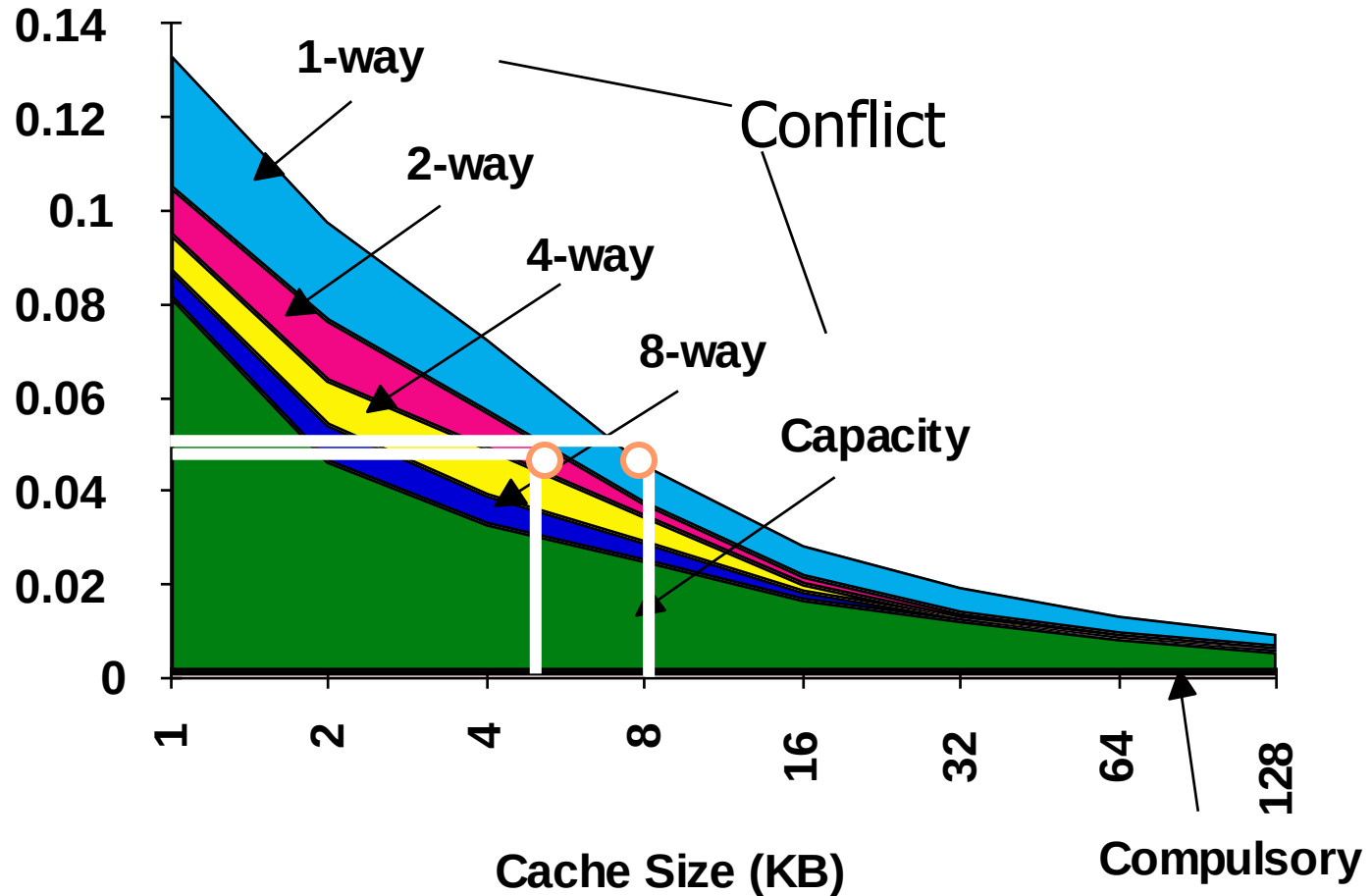
3Cs Absolute Miss Rate (SPEC92)

- Compulsory misses are a tiny fraction of the overall misses
- Capacity misses reduce with increasing sizes
- Conflict misses reduce with increasing associativity

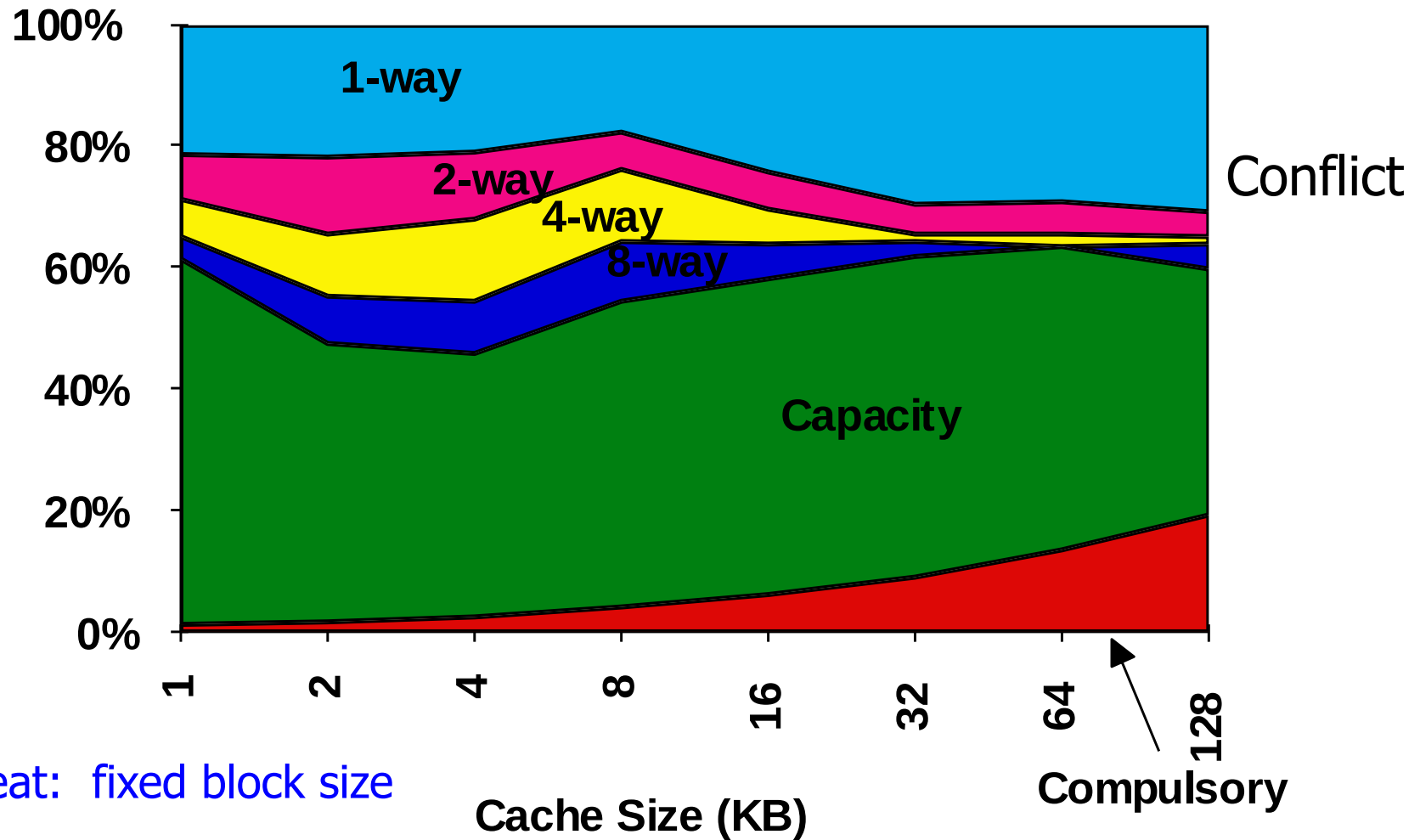


2:1 Cache Rule

Miss rate DM cache size X
 $\approx$ Miss rate 2-way SA cache size $X/2$

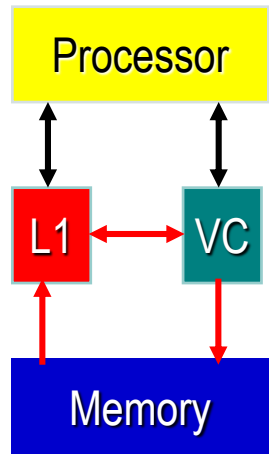


3Cs Relative Miss Rate



Caveat: fixed block size

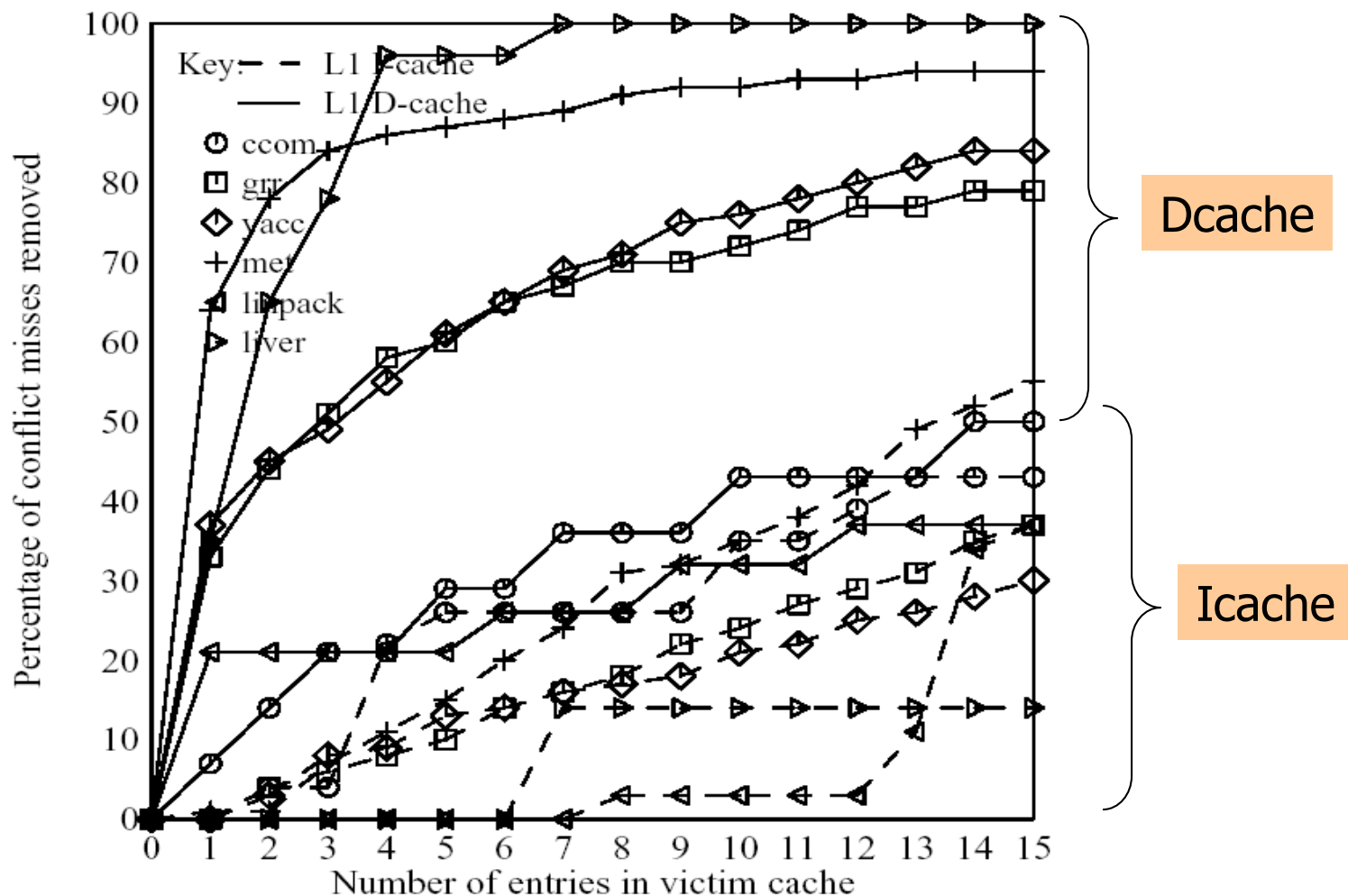
Victim Caching [Jouppi'90]



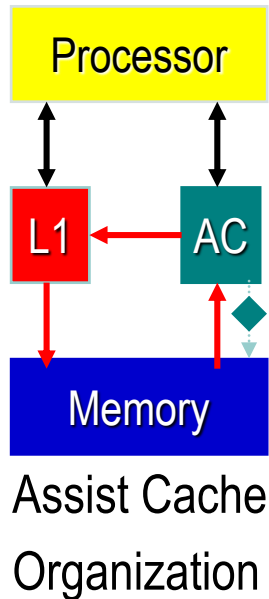
Victim Cache
Organization

- Victim cache (VC)
 - A small, fully associative structure
 - Effective in direct-mapped caches
- Whenever a line is displaced from L1 cache, it is loaded into VC
- Processor checks both L1 and VC simultaneously
- **Swap** data between VC and L1 if → L1 misses and VC hits
- When data has to be evicted from VC, it is written back to memory

% of Conflict Misses Removed



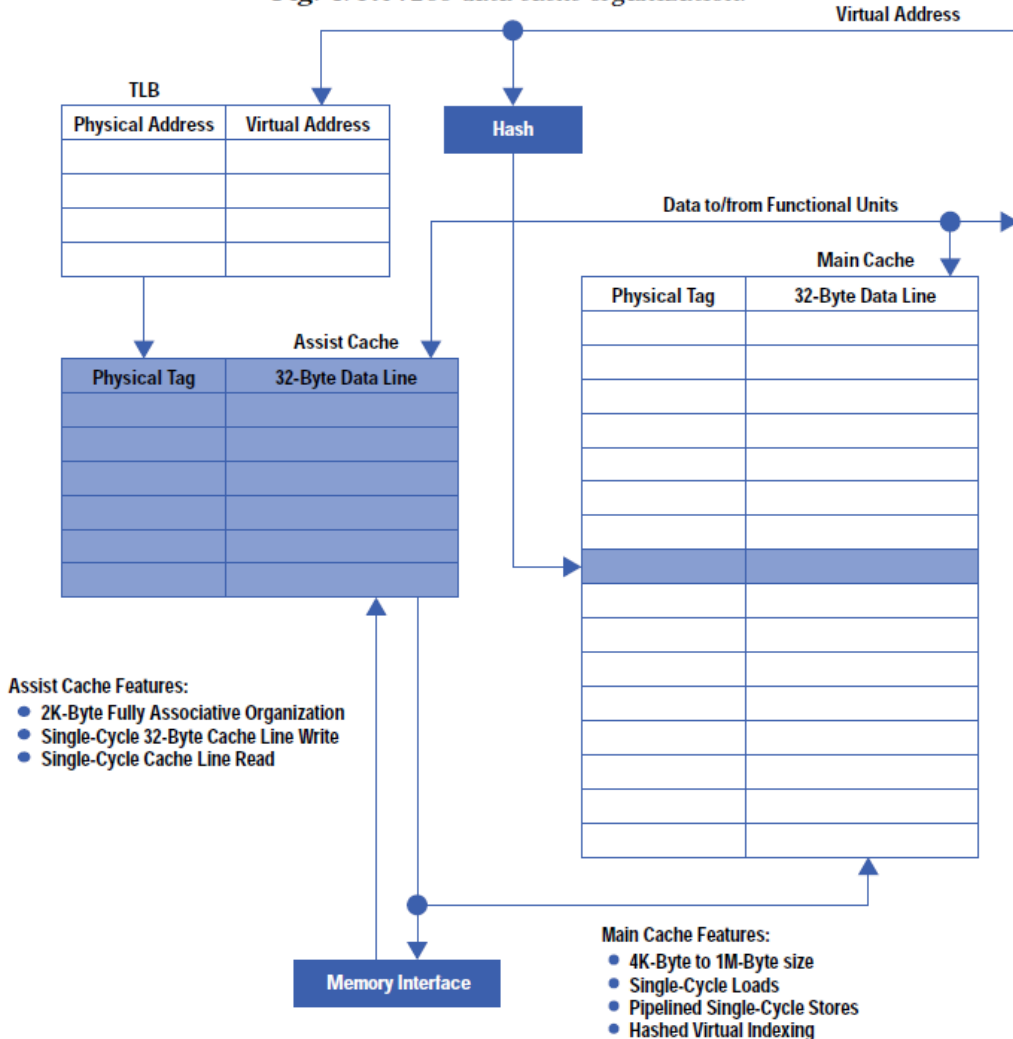
Assist Cache [Chan et al. '96]



- Assist Cache (on-chip) avoids thrashing in main (off-chip) L1 cache (both run at full speed)
 - 64 x 32-byte fully associative CAM
- Data enters Assist Cache when miss (FIFO replacement policy in Assist Cache)
- Data conditionally moved to L1 or back to memory during eviction
 - Flush back to memory when brought in by “Spatial locality hint” instructions
 - Reduce pollution

PA 7200 Data Cache (1996)

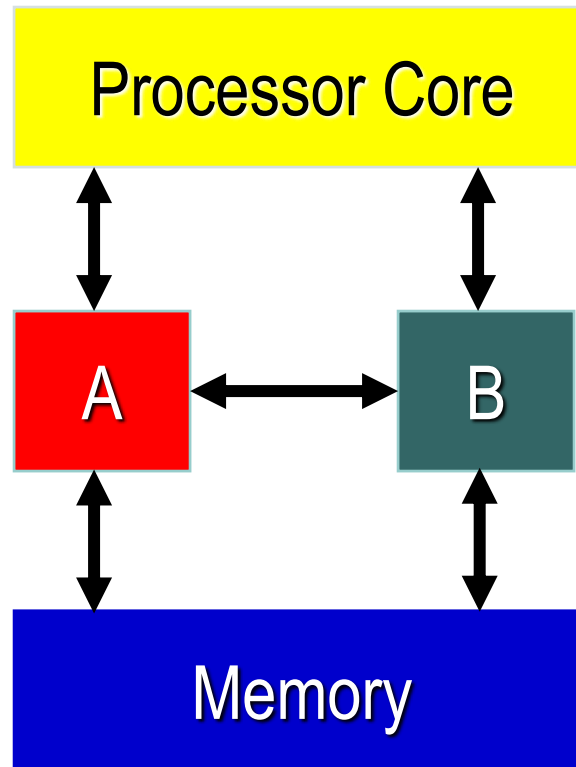
Fig. 4. PA 7200 data cache organization.



```
for i := 0 to N do
  A[i] := B[i] + C[i] + D[i]
```

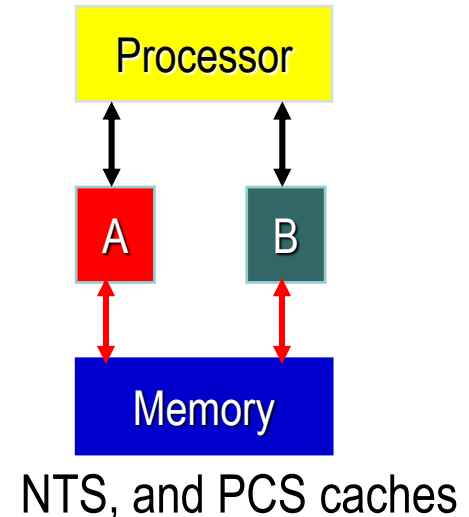
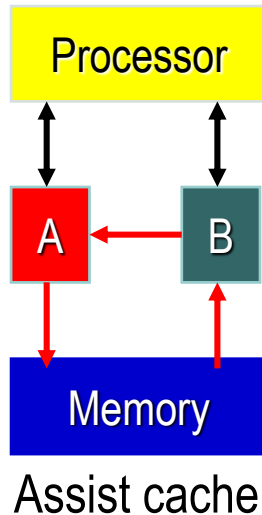
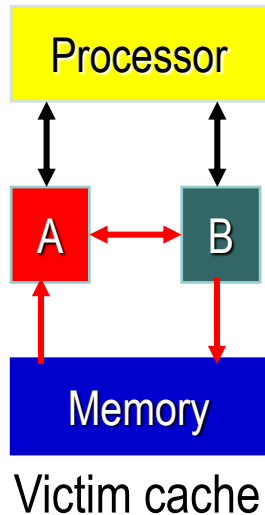
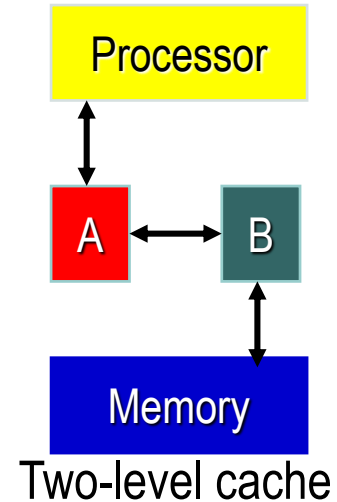
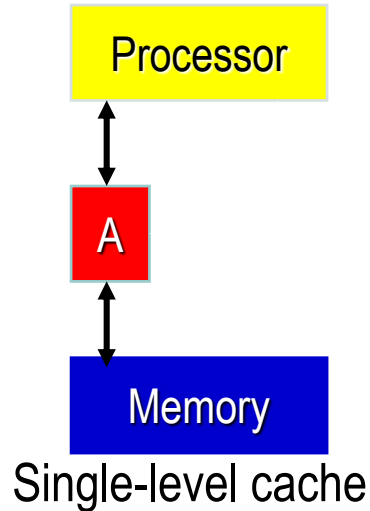
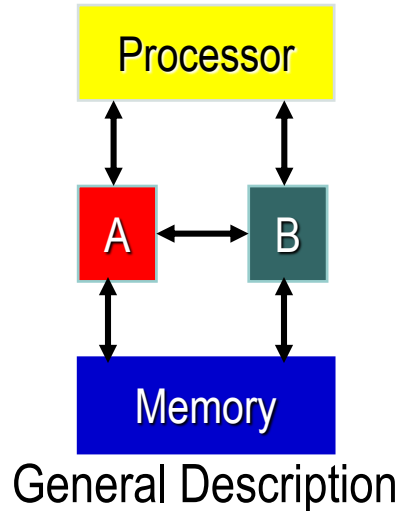
if elements A[i], B[i], C[i], and D[i] map to the same cache index, then a direct mapped cache alone would thrash on each element of the calculation. This would result in 32 cache misses for eight iterations of this loop. With an assist cache, however, each line is moved into the cache system without displacing the others. Assuming sequential 32-bit data elements, eight iterations of the loop causes only the initial four cache misses.

Multi-lateral Cache Architecture



- A Fully Connected Multi-Lateral Cache Architecture
- Most of the cache architectures be generalized into this form

Cache Architecture Taxonomy

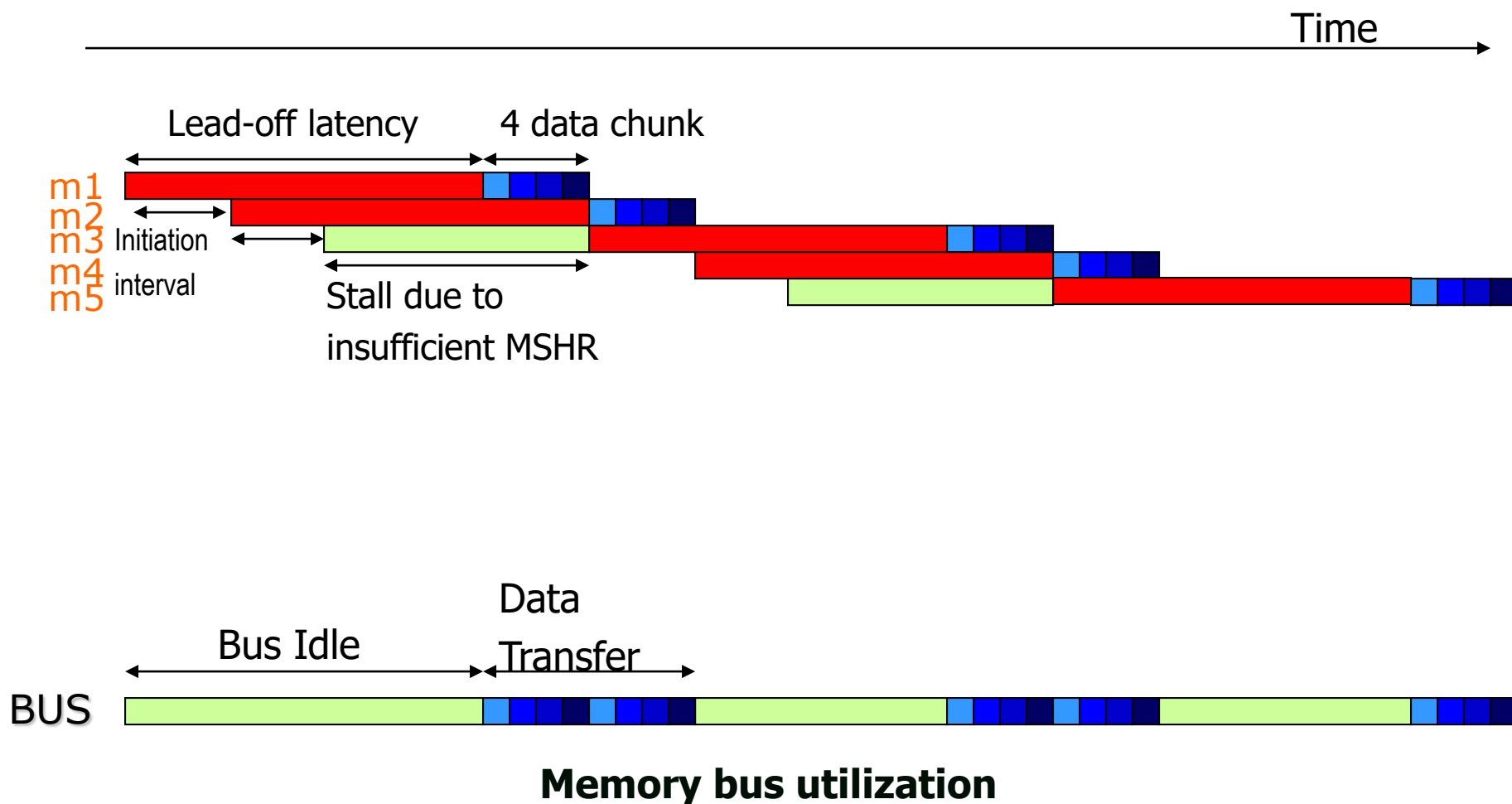


Non-blocking (Lockup-Free) Cache

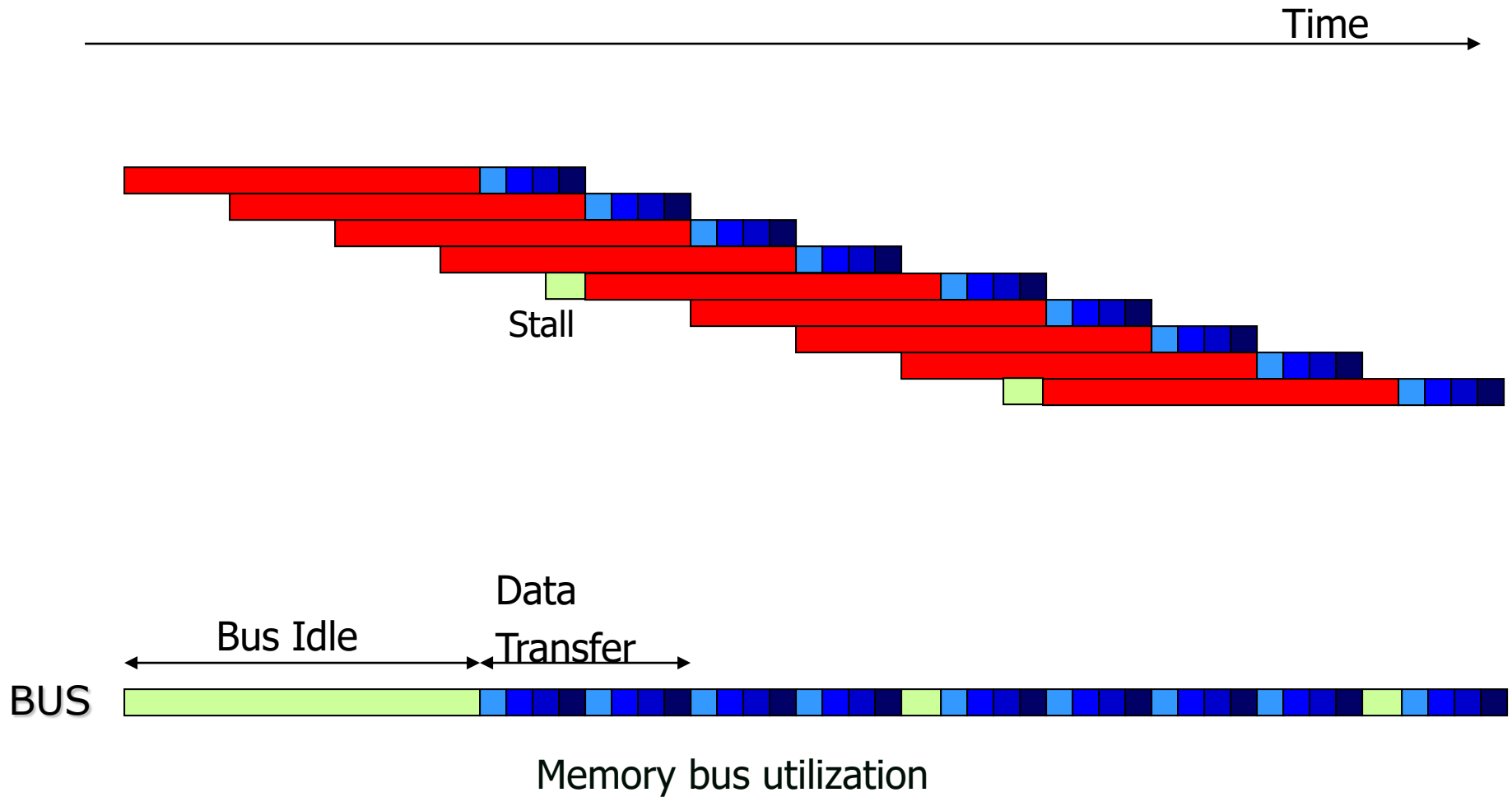
[Kroft '81]

- Prevent pipeline from stalling due to cache misses (continue to provide hits to other lines while servicing a miss on one/more lines)
- Uses Miss Status Handler Register (MSHR)
 - Tracks cache misses, allocate one entry per cache miss (called *fill buffer* in Intel P6 proliferation)
 - New cache miss checks against MSHR
 - Pipeline stalls at a cache miss only when MSHR is full
 - Carefully choose number of MSHR entries to match the sustainable bus bandwidth

Bus Utilization (MSHR = 2)



Bus Utilization (MSHR = 4)



Prefetch (Data/Instruction)

- Predict what data will be needed in future
- Pollution vs. Latency reduction
 - If you correctly predict the data that will be required in the future, you **reduce latency**. If you mispredict, you bring in unwanted data and **pollute the cache**
- To determine the effectiveness
 - When to initiate prefetch? (Timeliness)
 - Which lines to prefetch?
 - How big a line to prefetch? (note that cache mechanism already performs prefetching.)
 - What to replace?
- Software (data) prefetching vs. hardware prefetching

Software-controlled Prefetching

- Use instructions
 - Existing instruction
 - Alpha's load to r31 (hardwired to 0)
 - The Alpha architecture supports data prefetch via load instructions with a destination of register R31 or F31, which prefetch the cache line containing the addressed data. Instruction `LDS` with a destination of register F31 prefetches for a store.
 - Specialized instructions and hints
 - Intel's SSE: `prefetchnta`, `prefetcht0/t1/t2`
 - MIPS32: `PREF`
 - PowerPC: `dcbt` (data cache block touch), `dcbtst` (data cache block touch for store)
- Compiler or hand inserted prefetch instructions

Software-controlled Prefetching

- No Prefetching

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

- Assuming that each cache block holds 4 elements,
 - Result in 2 misses per 4 iterations



- Simple Prefetching

```
for (i=0; i < N; i++) {  
    prefetch (&a[i+1]);  
    prefetch (&b[i+1]);  
  
    sum += a[i]*b[i];  
}
```

- Problem: Unnecessary prefetching operations

Software-controlled Prefetching

- Prefetching + Loop unrolling

```
/* unroll loop 4 times */  
  
for (i=0; i < N; i+=4) {  
    prefetch (&a[i+4]);  
    prefetch (&b[i+4]);  
  
    sum += a[i]*b[i];  
    sum += a[i+1]*b[i+1];  
    sum += a[i+2]*b[i+2];  
    sum += a[i+3]*b[i+3];  
}
```



```
prefetch (&sum);  
prefetch (&a[0]);  
prefetch (&b[0]);  
  
/* unroll loop 4 times */  
  
for (i=0; i < N-4; i+=4) {  
    prefetch (&a[i+4]);  
    prefetch (&b[i+4]);  
  
    sum += a[i]*b[i];  
    sum += a[i+1]*b[i+1];  
    sum += a[i+2]*b[i+2];  
    sum += a[i+3]*b[i+3];  
}  
  
sum += a[N-4]*b[N-4];  
sum += a[N-3]*b[N-3];  
sum += a[N-2]*b[N-2];  
sum += a[N-1]*b[N-1];
```

- Problem

- 1st and last iterations

Hardware-based Prefetching

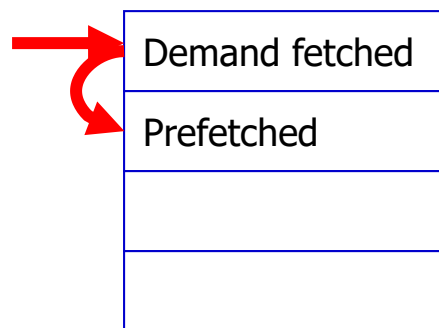
- Sequential prefetching
 - Prefetch on miss
 - Tagged prefetch
 - Both techniques are based on “One Block Lookahead (OBL)” prefetch: Prefetch line $(L+1)$ when line L is accessed based on some criteria

Sequential Prefetching

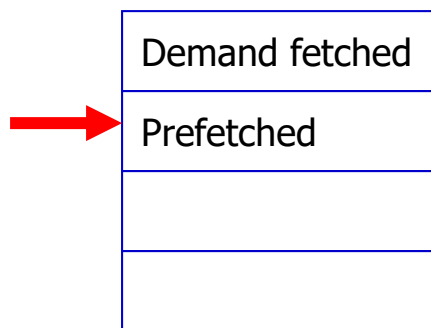
- Prefetch on miss
 - Initiate prefetch ($L+1$) whenever an access to L results in a miss
 - Alpha 21064 does this for instructions (prefetched instructions are stored in a separate structure called stream buffer)
- Tagged prefetch
 - Idea: Whenever there is a “first use” of a line (demand fetched or previously prefetched line), prefetch the next one
 - One additional “Tag bit” for each cache line
 - Tag the prefetched, not-yet-used line ($\text{Tag} = 1$)
 - Tag bit = 0 : the line is demand fetched, or a prefetched line is referenced for the first time
 - Prefetch ($L+1$) only if Tag bit = 1 on L

Sequential Prefetching

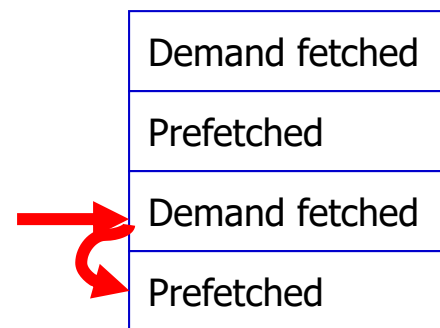
Prefetch-on-miss when accessing contiguous lines



miss

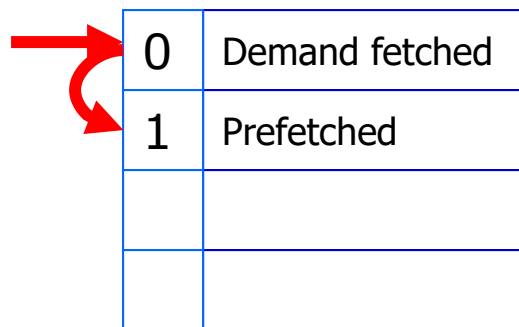


hit

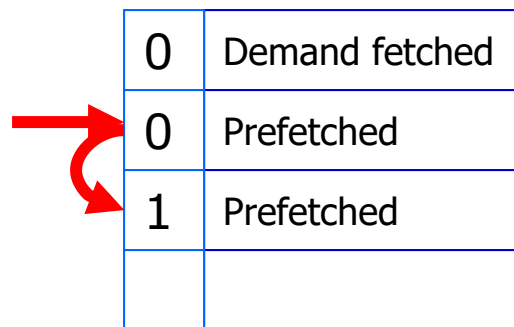


miss

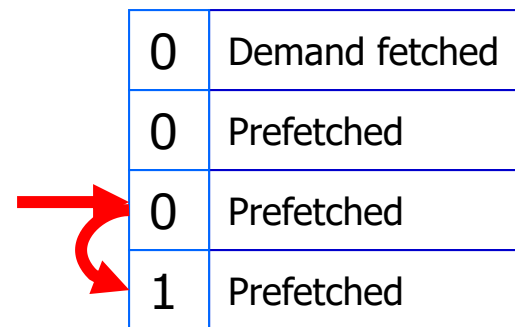
Tagged Prefetch when accessing contiguous lines



miss



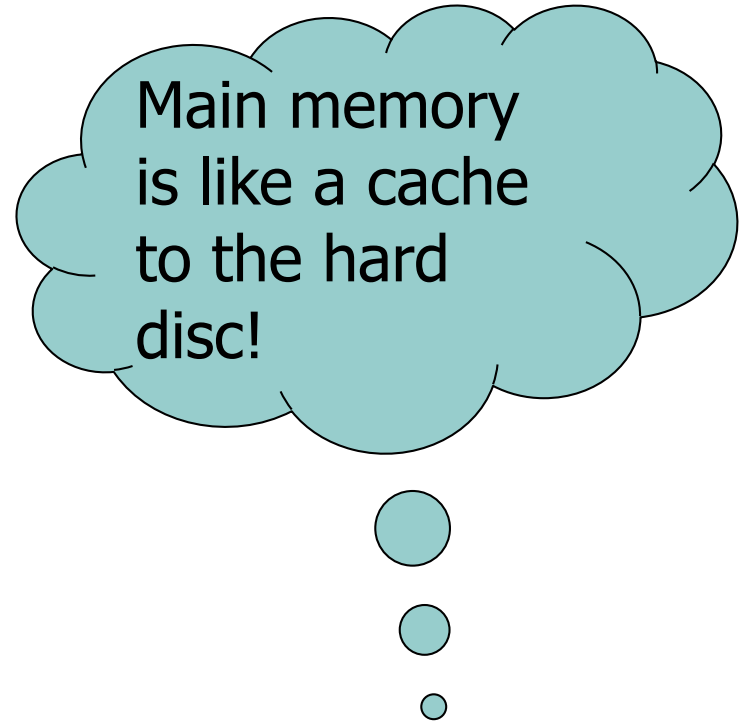
hit



hit

Virtual Memory

- **Virtual memory** – separation of logical memory from physical memory.
 - Only a part of the program needs to be in memory for execution. Hence, logical address space can be much larger than physical address space.
 - Allows address spaces to be shared by several processes (or threads).
 - Allows for more efficient process creation.
- Virtual memory can be implemented via:
 - Demand paging
 - Demand segmentation



Virtual Address

- The concept of a virtual (or logical) *address space* that is bound to a separate *physical address space* is central to memory management
 - **Virtual address** – generated by the CPU
 - **Physical address** – seen by the memory
- Virtual and physical addresses are the same in compile-time and load-time address-binding schemes; virtual and physical addresses differ in execution-time address-binding schemes

Advantages of Virtual Memory

- Translation:

- Program can be given consistent view of memory, even though physical memory is scrambled
- Only the most important part of program ("Working Set") must be in physical memory.
- Contiguous structures (like stacks) use only as much physical memory as necessary yet grow later.

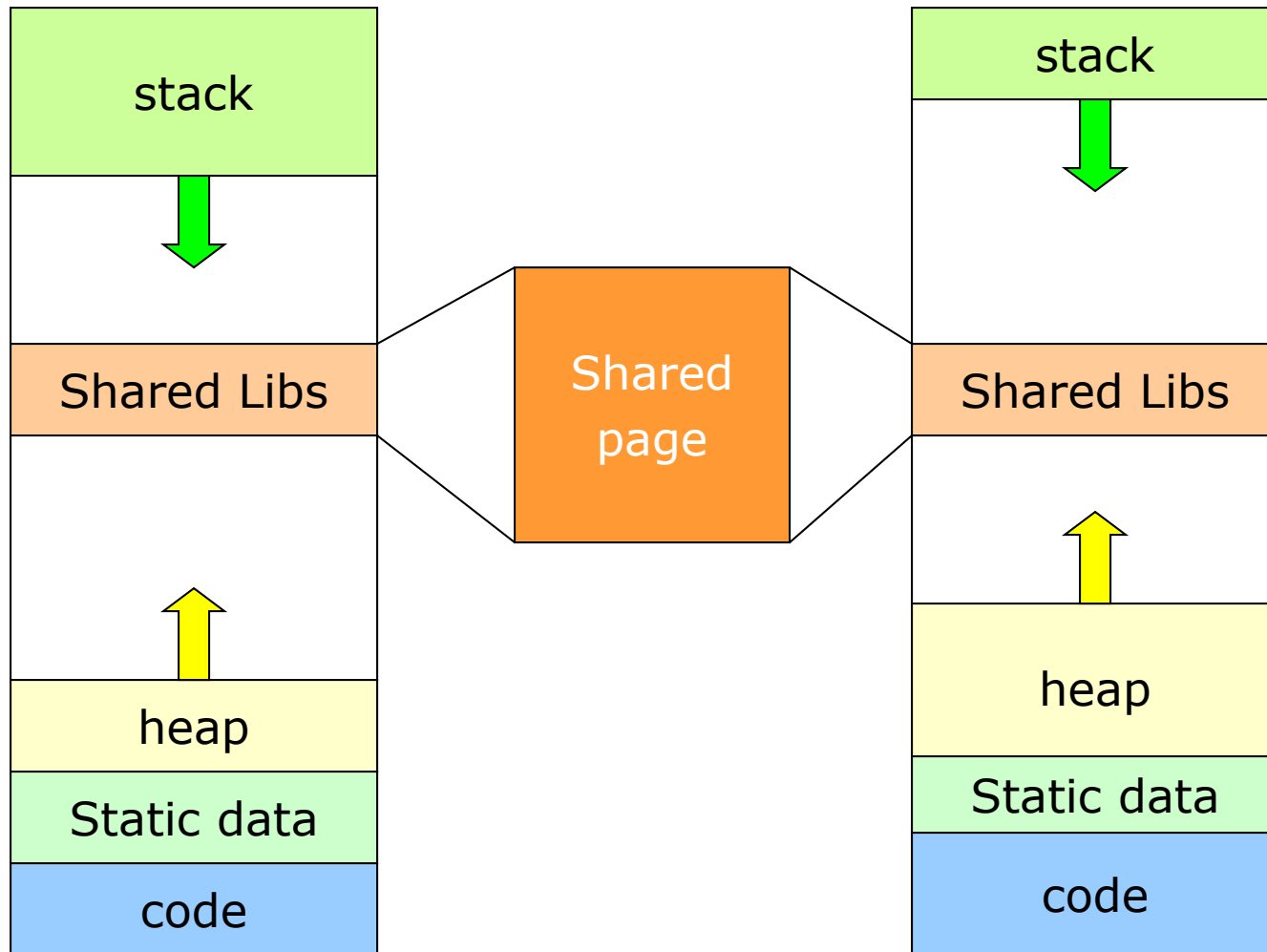
- Protection:

- Different threads (or processes) protected from each other.
- Different pages can be given special behavior
 - (Read Only, Invisible to user programs, etc).
- Kernel data protected from User programs
- Very important for protection from malicious programs
=> Far more "viruses" under Microsoft Windows

- Sharing:

- Can map same physical page to multiple users ("Shared memory")

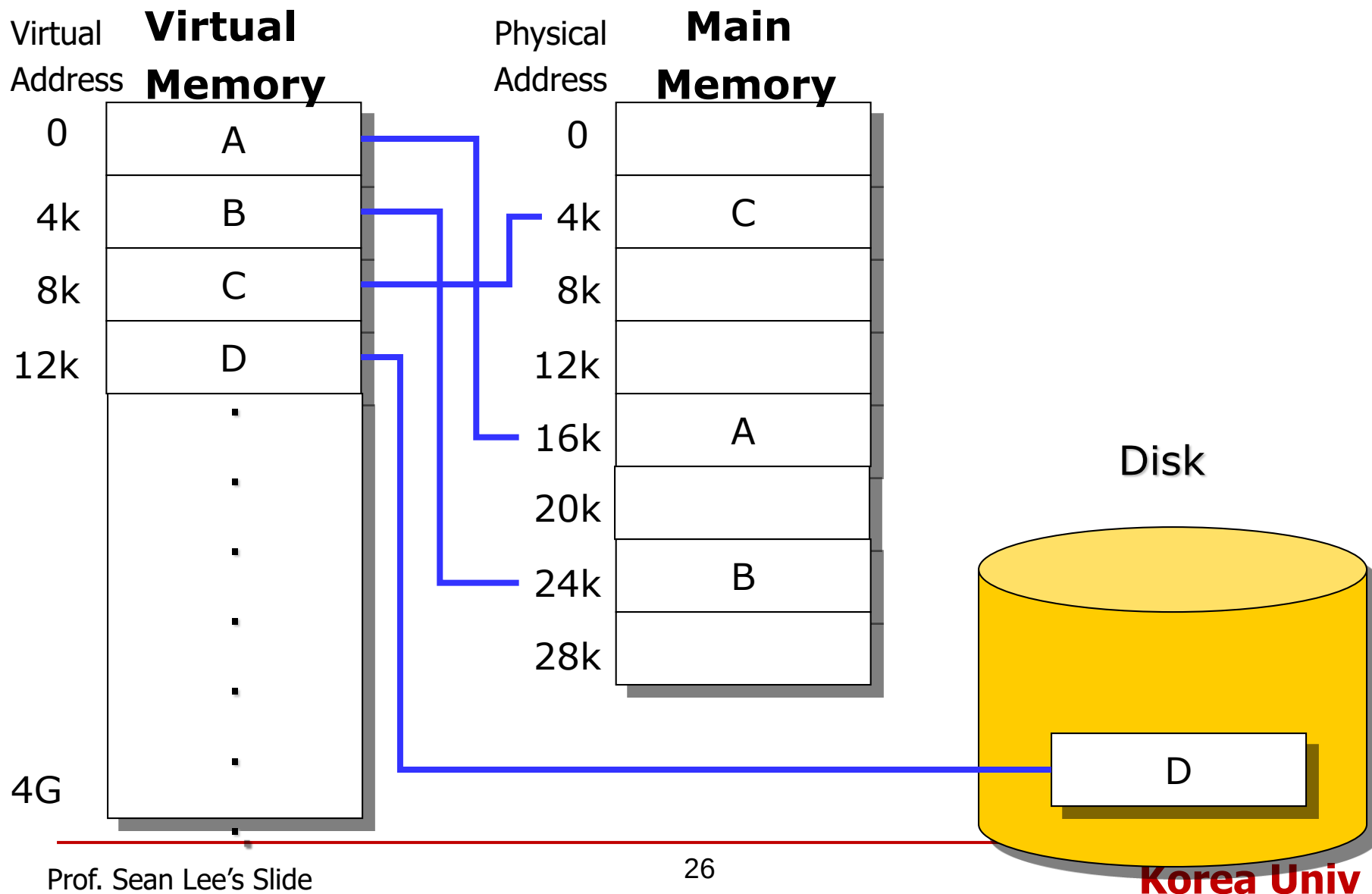
Use of Virtual Memory



Process A

Process B

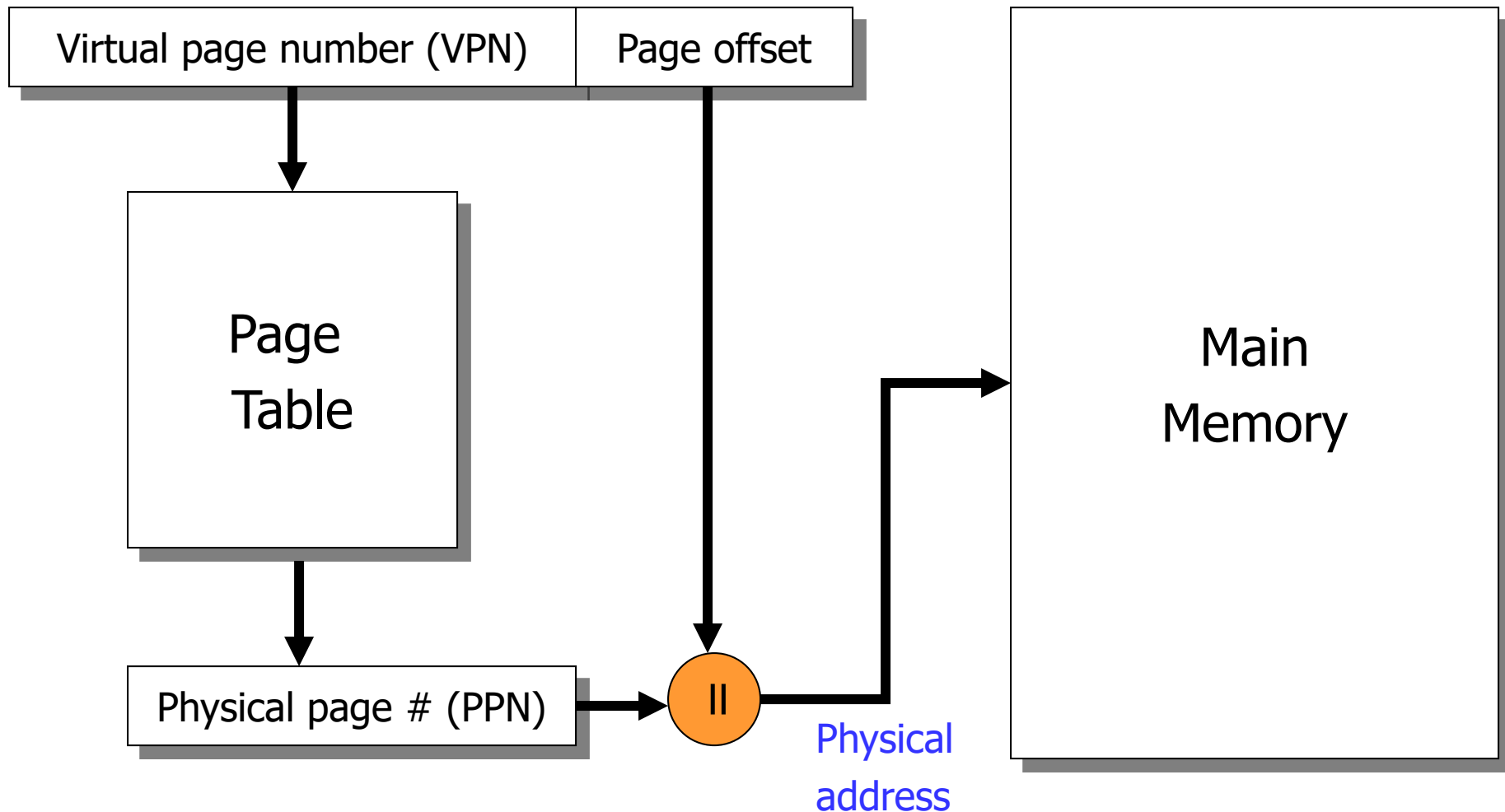
Virtual vs. Physical Address Space



Paging

- Divide physical memory into fixed-size blocks (e.g., 4KB) called **frames**
- Divide logical memory into blocks of same size (4KB) called **pages**
- To run a program of size n pages, need to find n free frames and load program
- Set up a **page table** to map page addresses to frame addresses (operating system sets up the page table)

Page Table and Address Translation



Page Table Structure Examples

- One-to-one mapping, space?
 - Large pages → Internal fragmentation (similar to having large line sizes in caches)
 - Small pages → Page table size issues
- Multi-level Paging
- Inverted Page Table

Example:

64 bit address space, 4 KB pages (12 bits), 512 MB (29 bits) RAM

Number of pages = $2^{64}/2^{12} = 2^{52}$

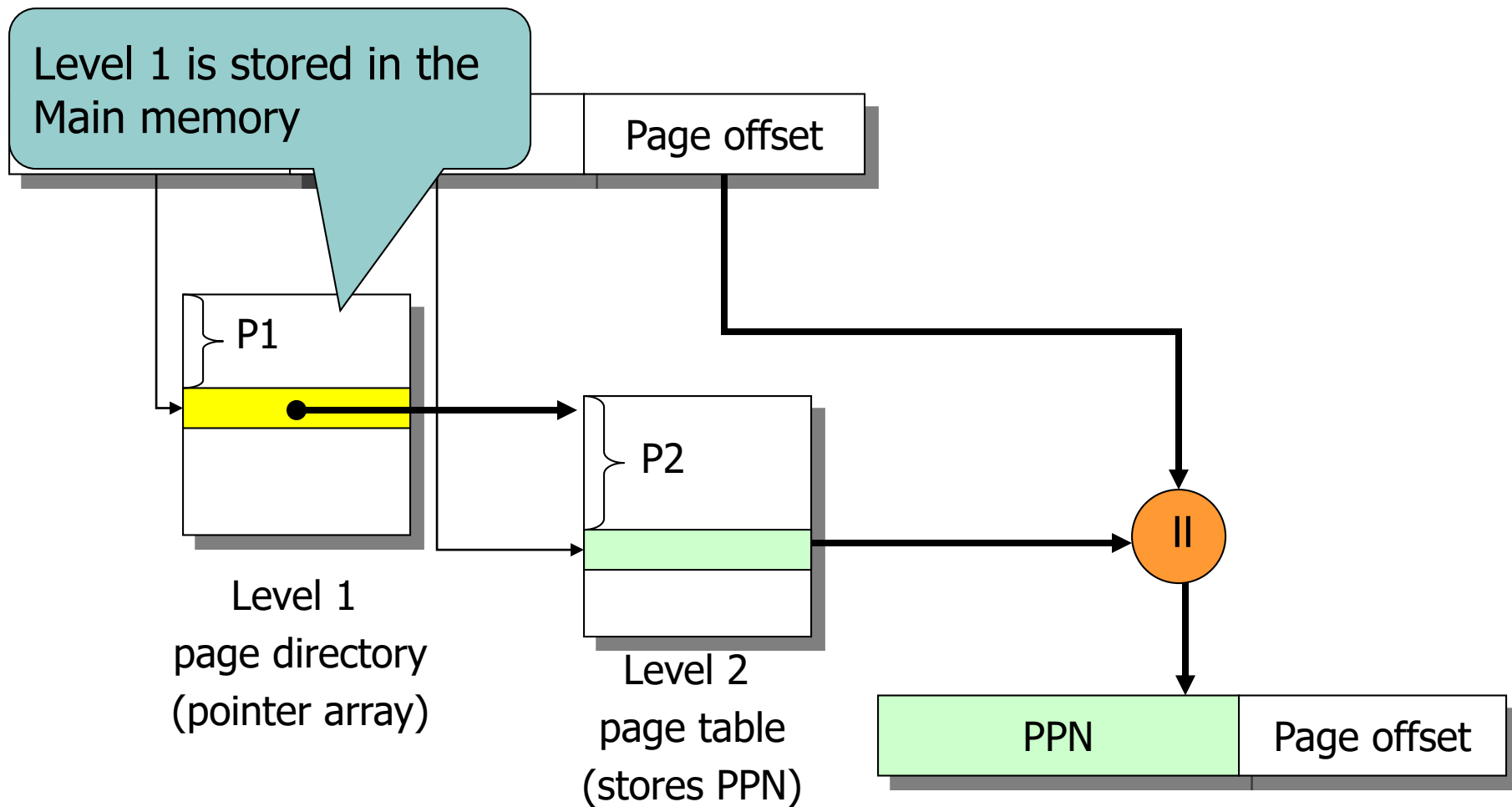
(The page table has as many entrees)

Each entry is ~4 bytes, the size of the Page table is 2^{54} Bytes = 16 Petabytes!

Can't fit the page table in the 512 MB RAM!

Multi-level (Hierarchical) Page Table

- Divide virtual address into multiple levels



Inverted Page Table

- One entry for each real page of memory
- Shared by all active processes
- Entry consists of the virtual address of the page stored in that real memory location, with Process ID information
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs

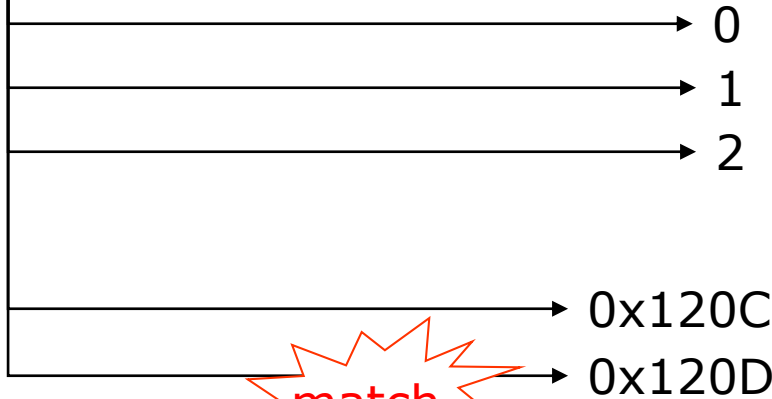
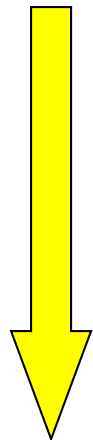
Linear Inverted Page Table

- Contain entries (size of physical memory) in a linear array
- Need to traverse the array sequentially to find a match
- Can be time consuming

PID = 8 Virtual Address

VPN = 0x2AA70	Offset
---------------	--------

PPN
Index



match

PPN = 0x120D	Offset
--------------	--------

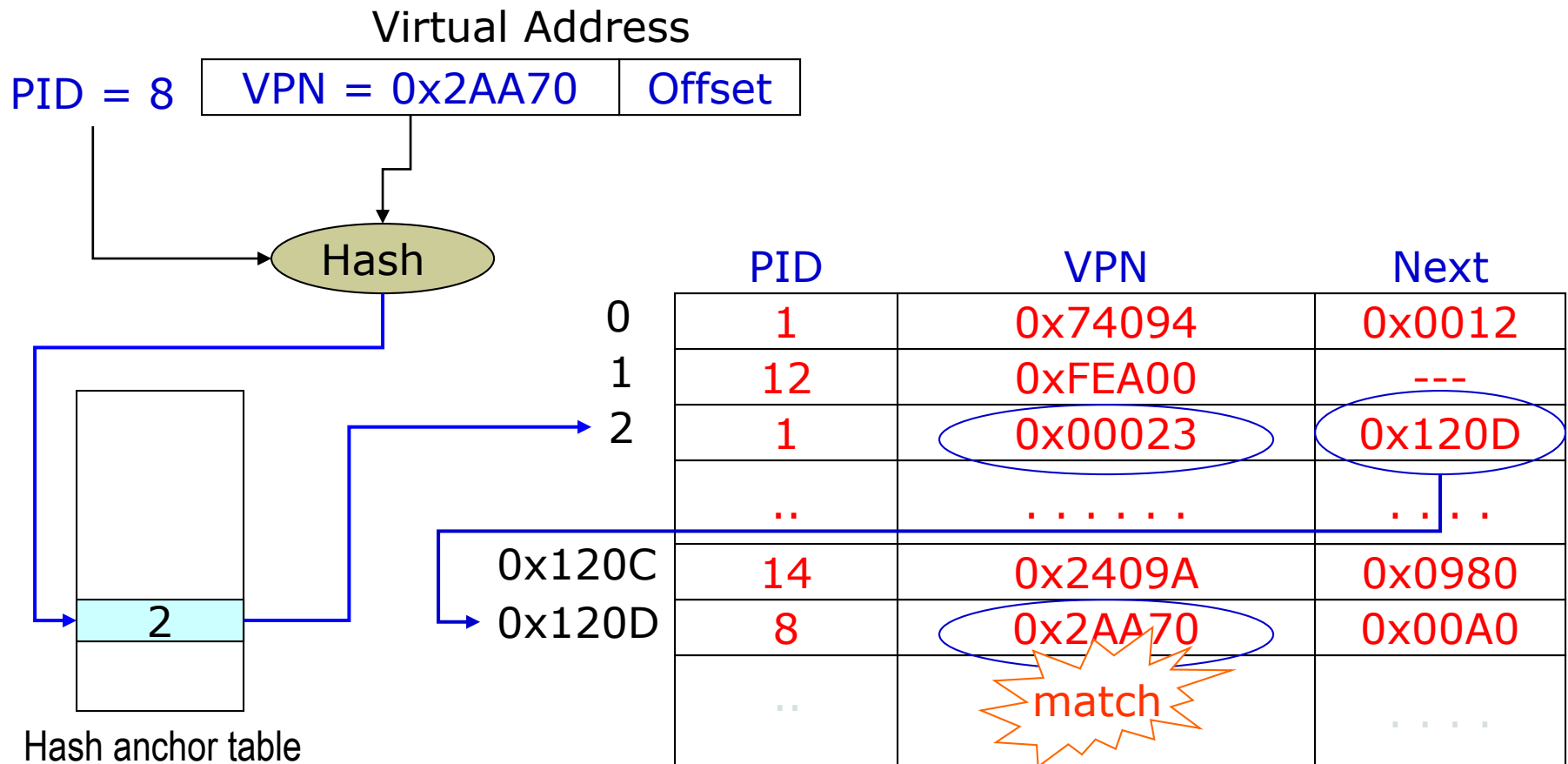
Physical Address

PID	VPN
1	0x74094
12	0xFEAA00
1	0x00023
..	
14	0x2409A
8	0x2AA70
..	

Linear Inverted Page Table

Hashed Inverted Page Table

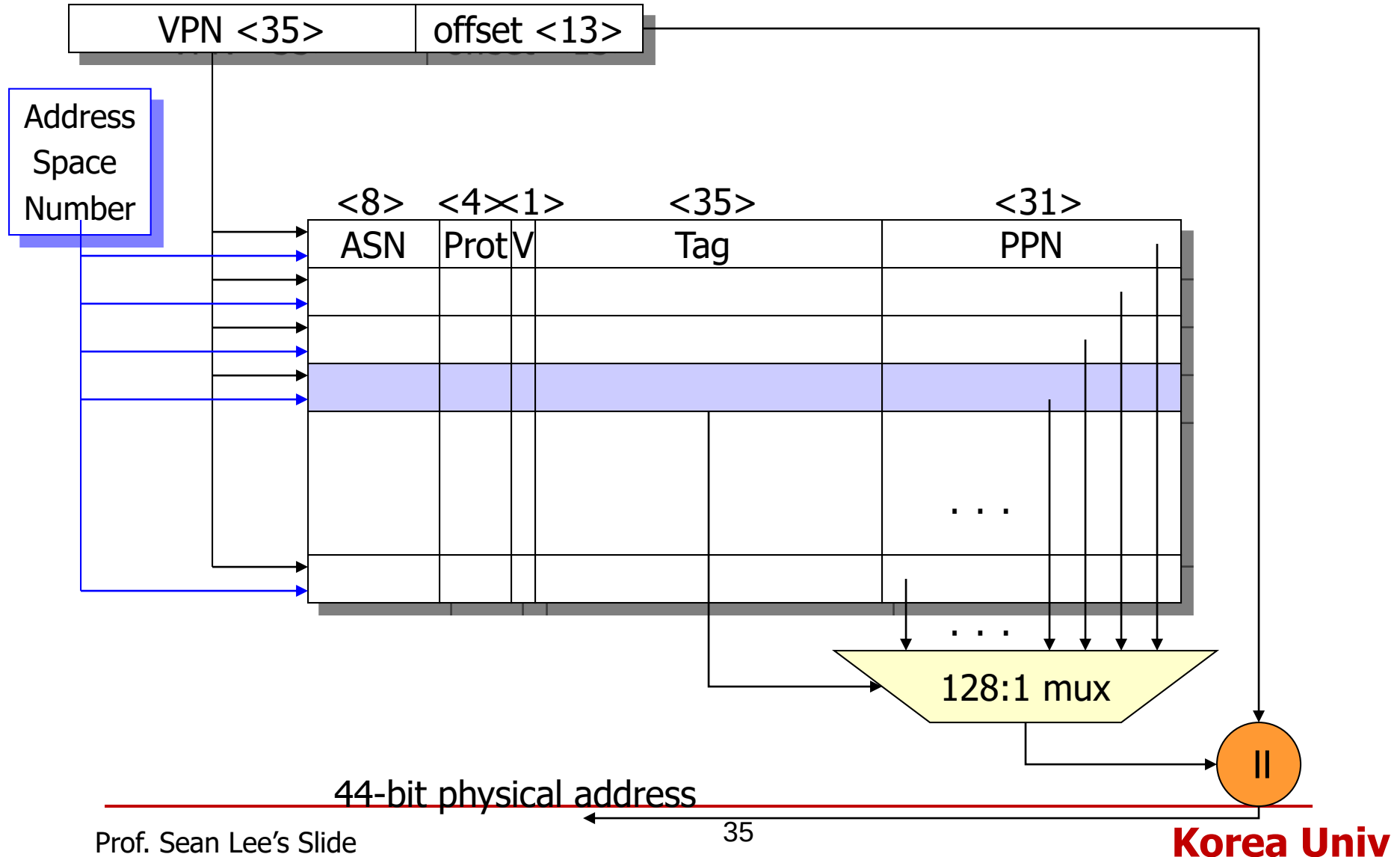
- Use hash table to limit the search to smaller number of page-table entries



Fast Address Translation

- How often address translation occurs?
- Where the page table is kept?
- Keep translation in the hardware
- Use Translation Lookaside Buffer (TLB)
 - Instruction-TLB & Data-TLB
 - Essentially a cache (tag array = VPN, data array=PPN)
 - Small (32 to 256 entries are typical)
 - Typically fully associative (implemented as a content addressable memory, CAM) or highly associative to minimize conflicts

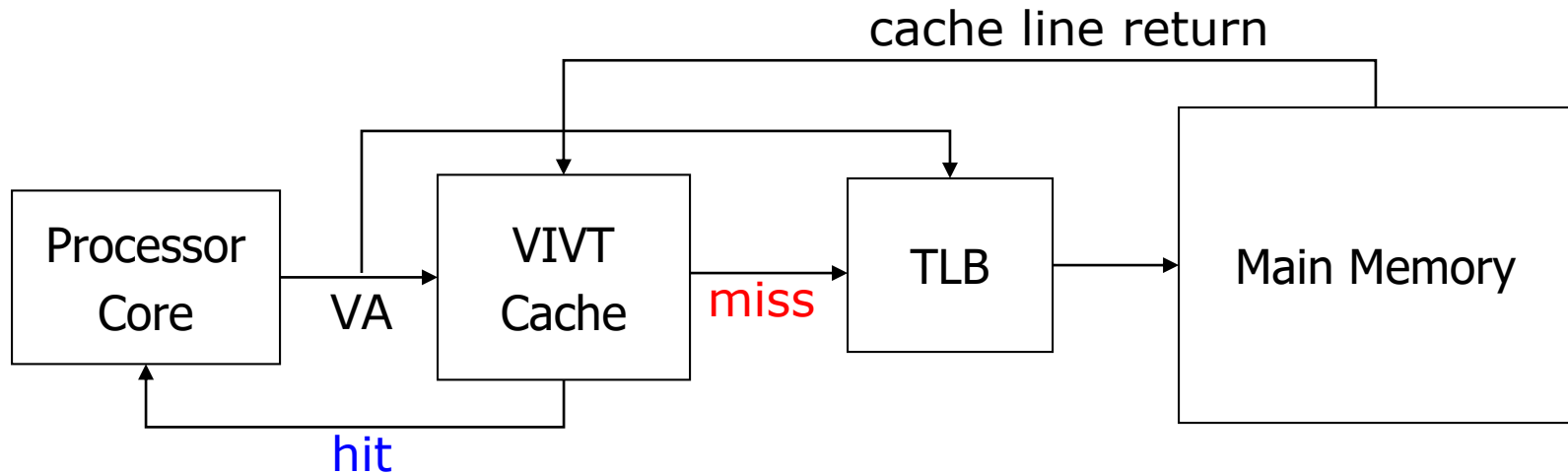
Example: Alpha 21264 data TLB



TLB and Caches

- Several Design Alternatives
 - VIVT: Virtually-indexed Virtually-tagged Cache
 - VIPT: Virtually-indexed Physically-tagged Cache
 - PIVT: Physically-indexed Virtually-tagged Cache
 - Not outright useful, R6000 is the only used this.
 - PIPT: Physically-indexed Physically-tagged Cache

Virtually-Indexed Virtually-Tagged (VIVT)

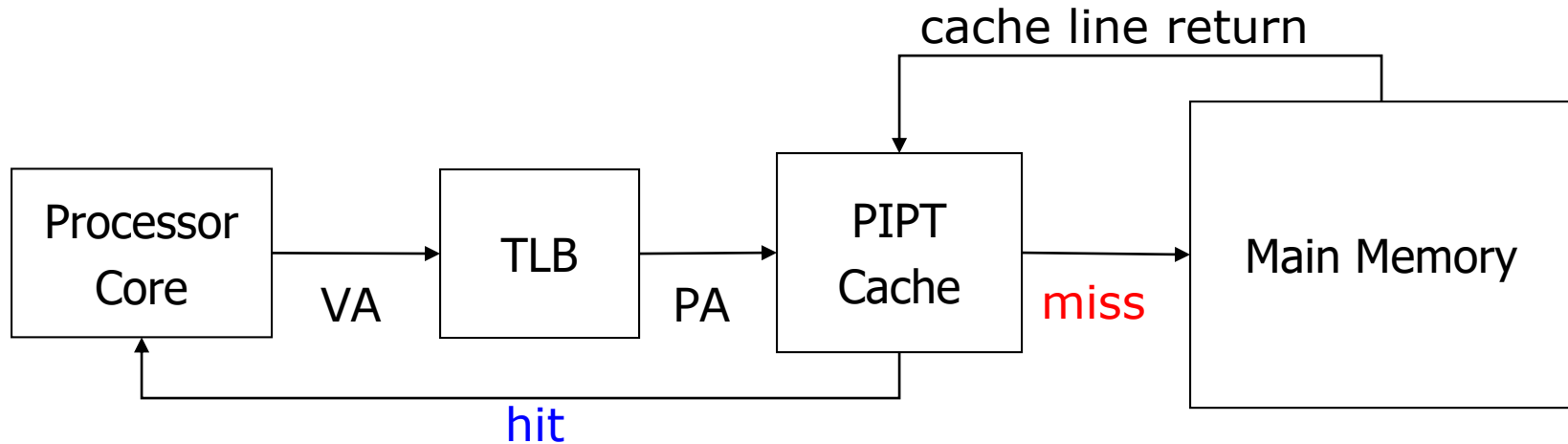


- Fast cache access
- Only require address translation when going to memory (miss)
- Issues?

VIVT Cache Issues - Aliasing

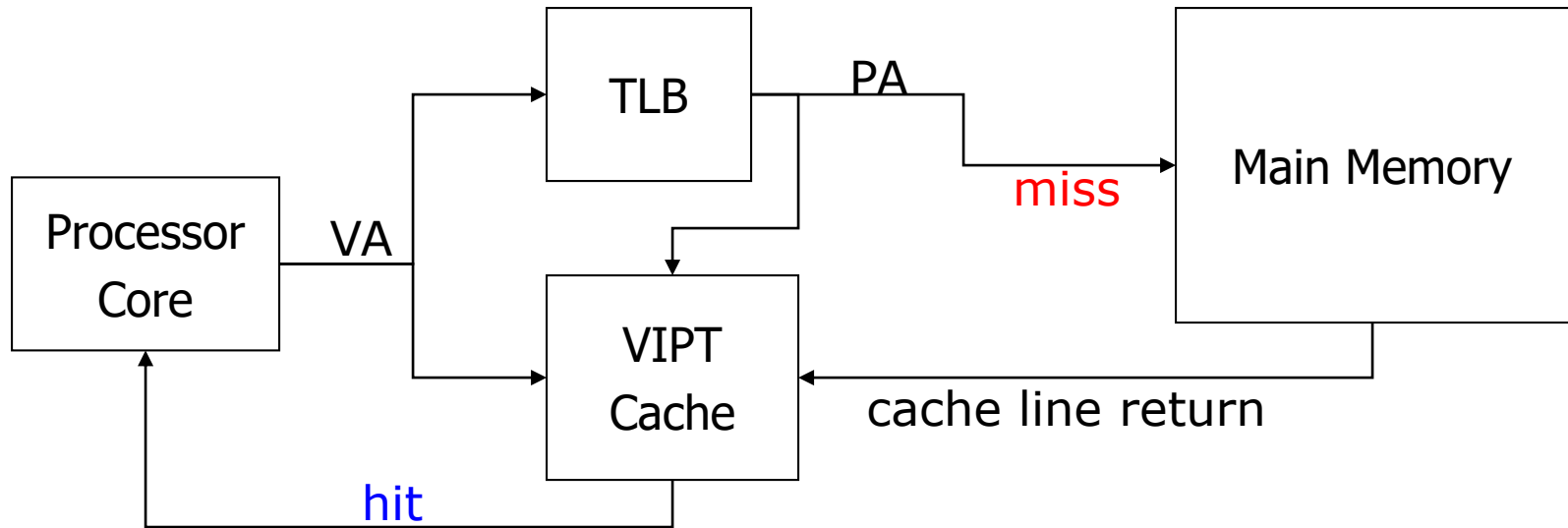
- Homonym
 - Same VA maps to different PAs
 - Occurs when there is a context switch
 - Solutions
 - Include process id (PID) in cache or
 - Flush cache upon context switches
- Synonym (also a problem in VIPT)
 - Different VAs map to the same PA
 - Occurs when data is shared by multiple processes
 - Duplicated cache line in VIPT cache and VIVT\$ w/ PID
 - Data is inconsistent due to duplicated locations
 - Solution
 - Can Write-through solve the problem?
 - Flush cache upon context switch
 - If $(\text{index} + \text{offset}) < \text{page offset}$, can the problem be solved? (discussed later in VIPT)

Physically-Indexed Physically-Tagged (PIPT)



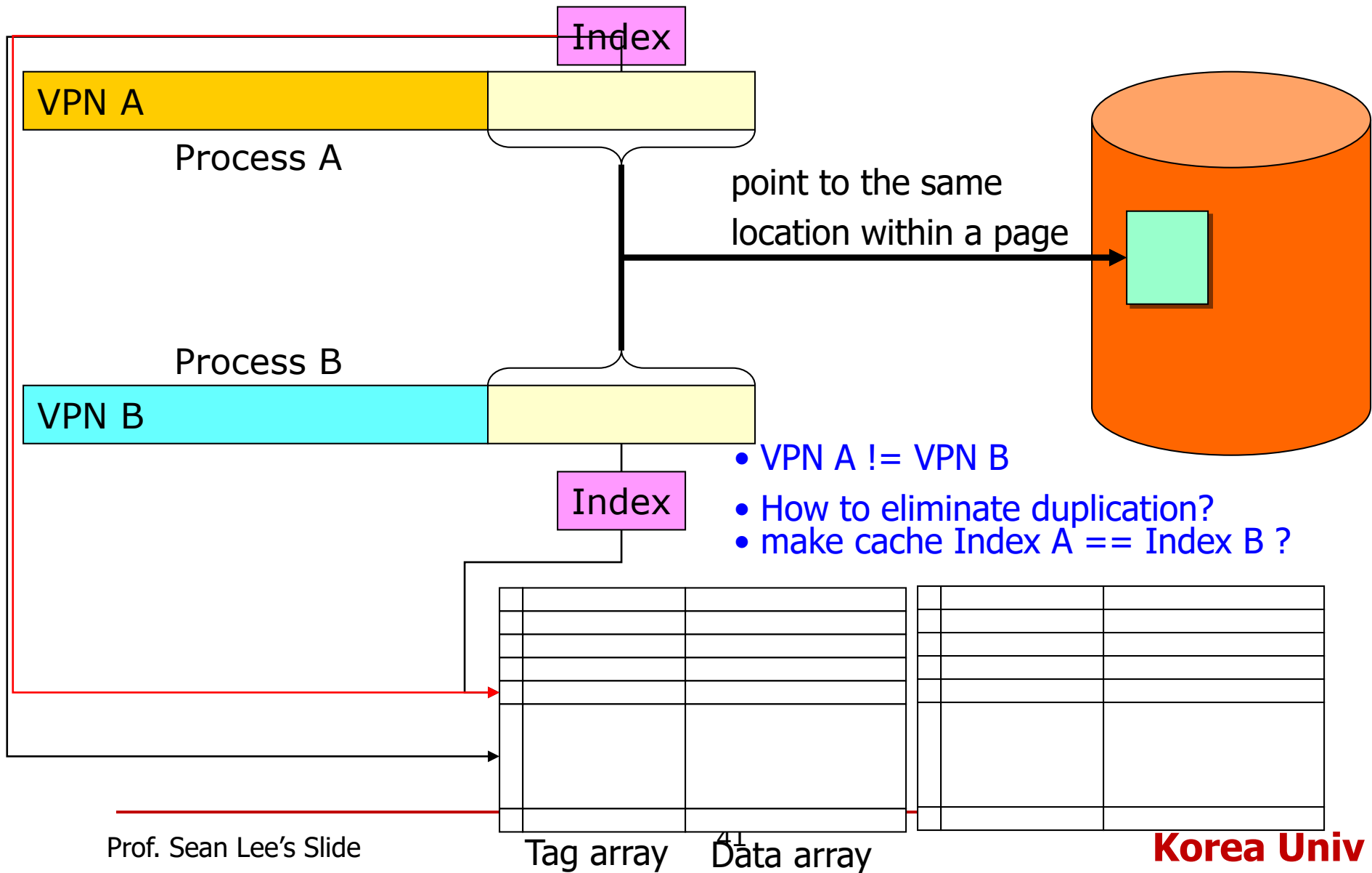
- Slower, always translate address before accessing memory
- Simpler for data coherence

Virtually-Indexed Physically-Tagged (VIPT)

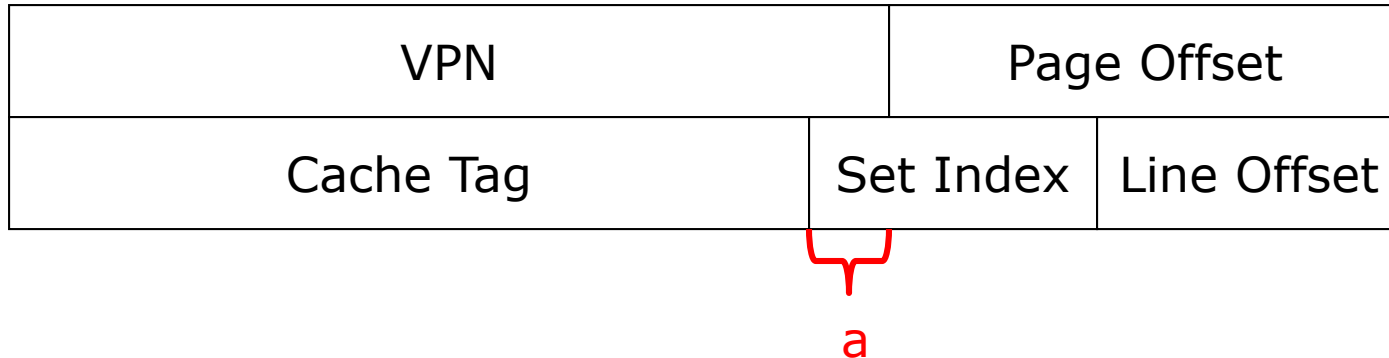


- Gain benefit of a VIVT and PIPT
- Parallel Access to TLB and VIPT cache
- No Homonym
- How about Synonym?

Deal w/ Synonym in VIPT Cache



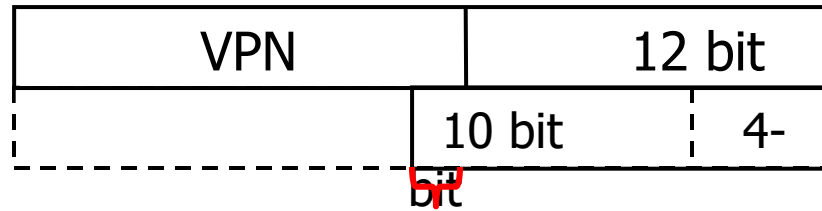
Synonym in VIPT Cache



- If two VPNs do not differ in **a** then there is no synonym problem, since they will be indexed to the same set of a VIPT cache
- Imply # of sets cannot be too big
- Max number of sets = page size / cache line size
 - Ex: 4KB page, 32B line, max set = 128
- A complicated solution in MIPS R10000

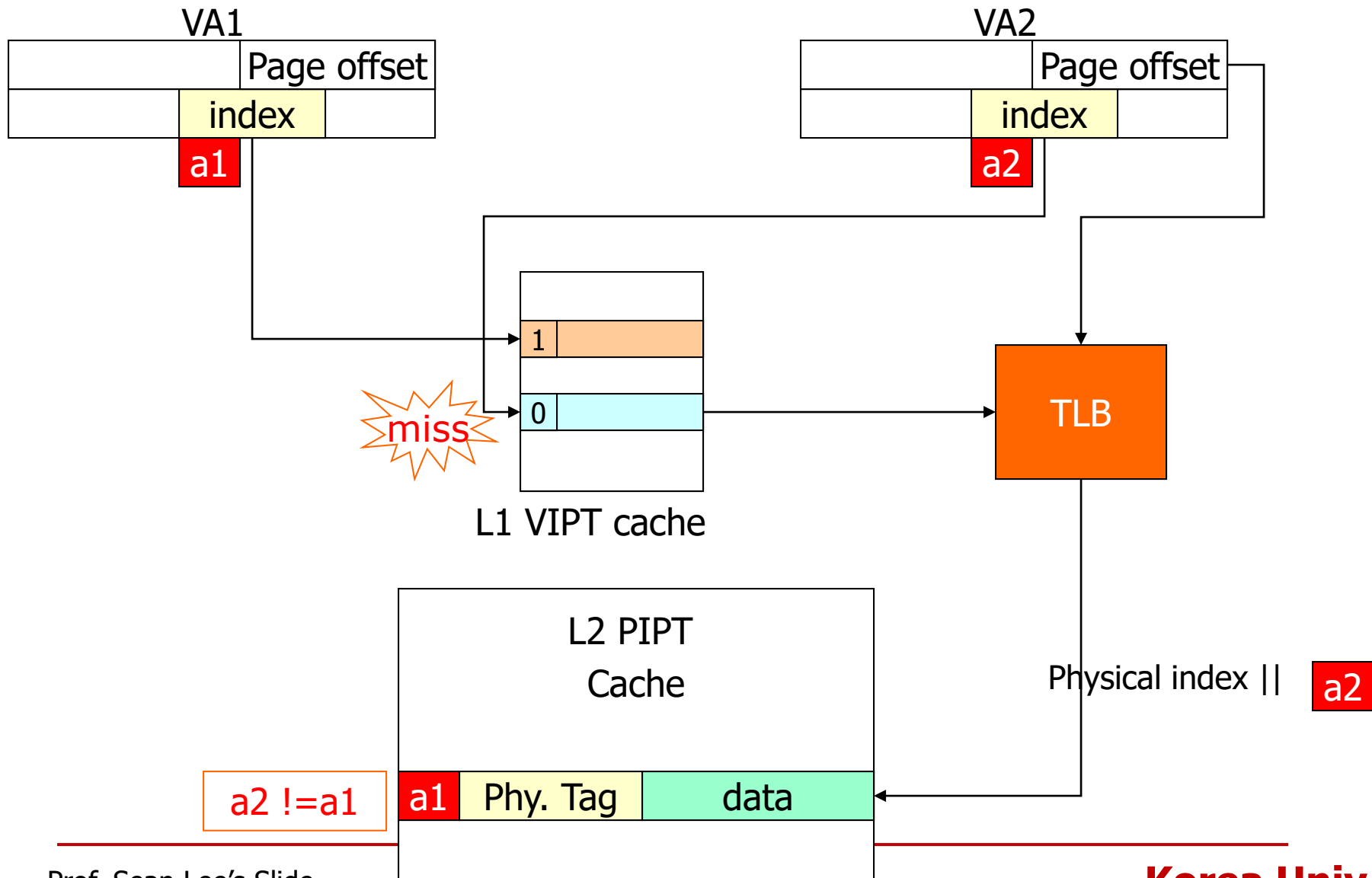
R10000's Solution to Synonym

- 32KB 2-Way Virtually-Indexed L1



- Direct-Mapped Physical L2
 - L2 is *Inclusive* of L1
 - VPN[1:0] is *appended* to the “tag” of L2
- Given two virtual addresses *VA1* and *VA2* that differs in *VPN[1:0]* and both map to the same physical address *PA*
 - Suppose *VA1* is accessed first so blocks are allocated in L1&L2
 - What happens when *VA2* is referenced?
 - VA2* indexes to a different block in L1 and misses
 - VA2* translates to *PA* and goes to the same block as *VA1* in L2
 - Tag comparison fails (since *VA1*[1:0]≠*VA2*[1:0])
 - Treated just like as a L2 conflict miss ⇒ *VA1*'s entry in L1 is ejected (or dirty-written back if needed) due to *inclusion policy*

Deal w/ Synonym in MIPS R10000



Deal w/ Synonym in MIPS R10000

