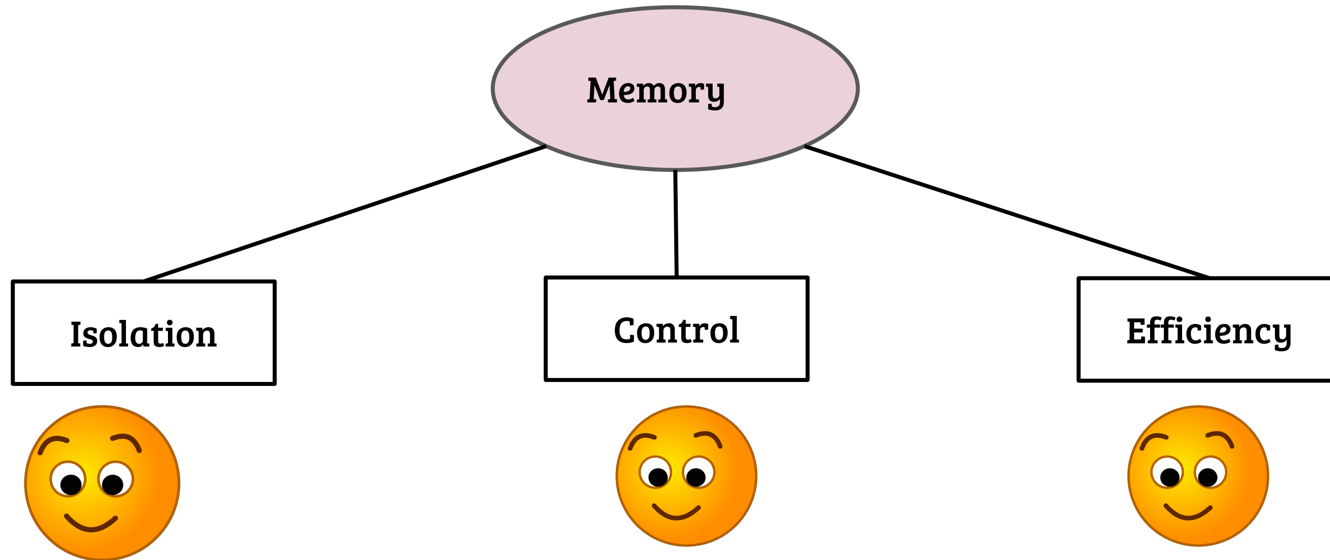
Operating Systems

Resource multiplexing - Memory

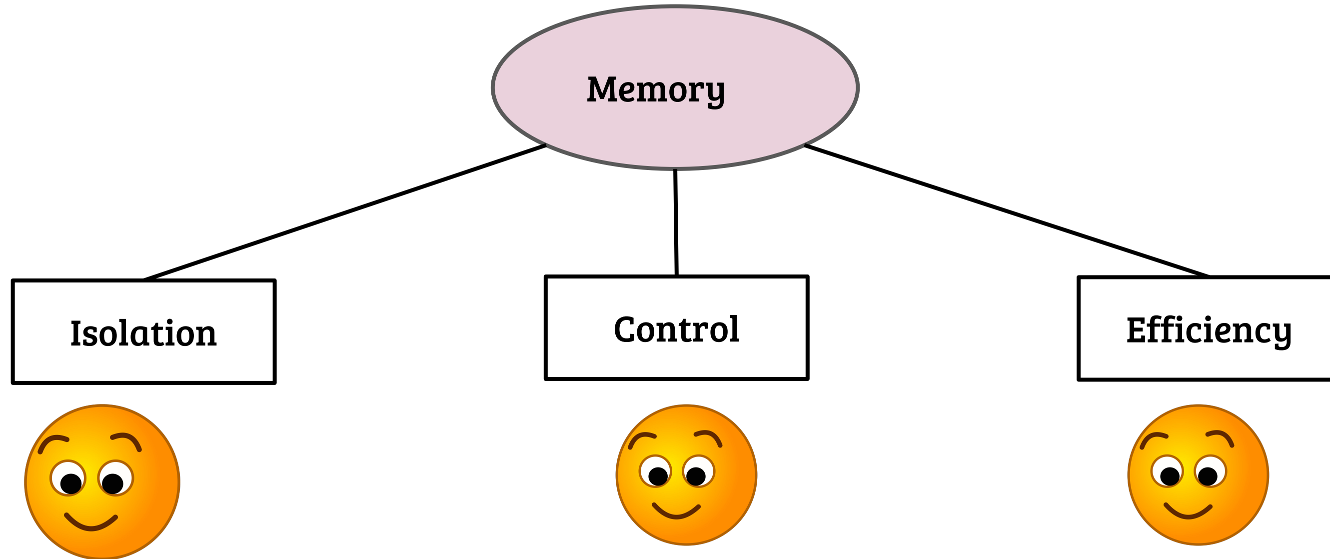
Debadatta Mishra, CSE, IITK

Memory virtualization



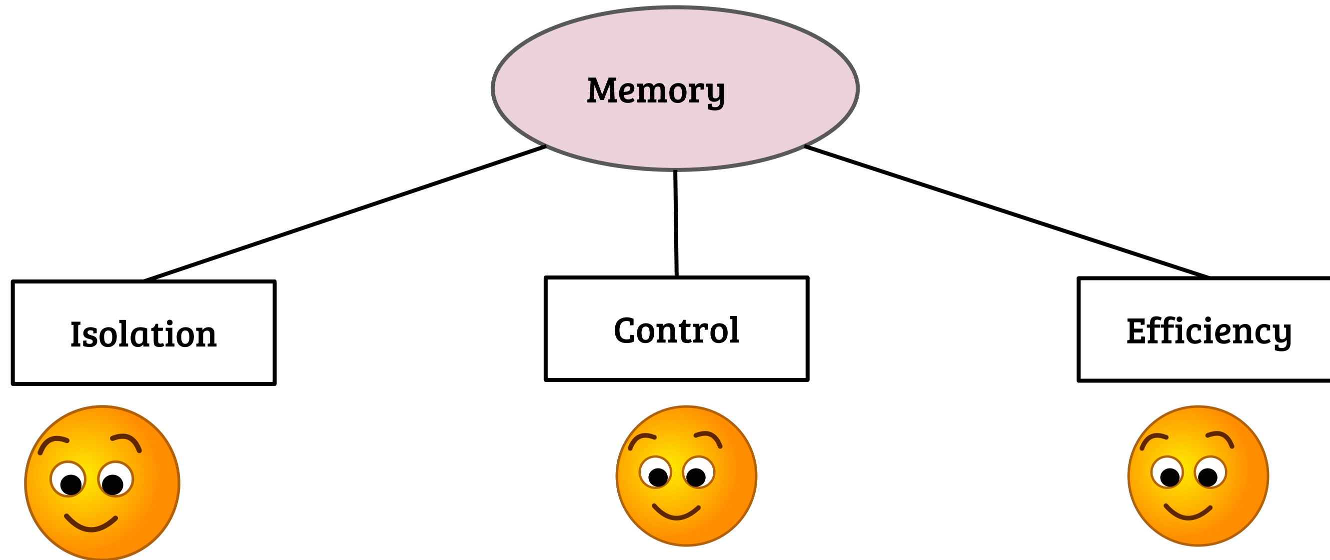
1. Inter and Intra application isolation
2. User and OS isolation with LDA

Memory virtualization



1. Application transparency
2. Data retention guarantees

Memory virtualization



1. Limited OS interventions
2. Flexible resource provisioning

Memory virtualization: Requirements and Usage

1. Inter and Intra application isolation
2. User and OS isolation with LDA

1. Application transparency
2. Data retention guarantees

1. Limited OS interventions
2. Flexible resource provisioning

1. Expect to address code and data in a seamless manner
2. Use dynamic memory allocation
3. Overestimate memory usage
4. Can not be trusted to access memory in a "safe" and "efficient" manner

Let us take a closer look!

The role of compiler

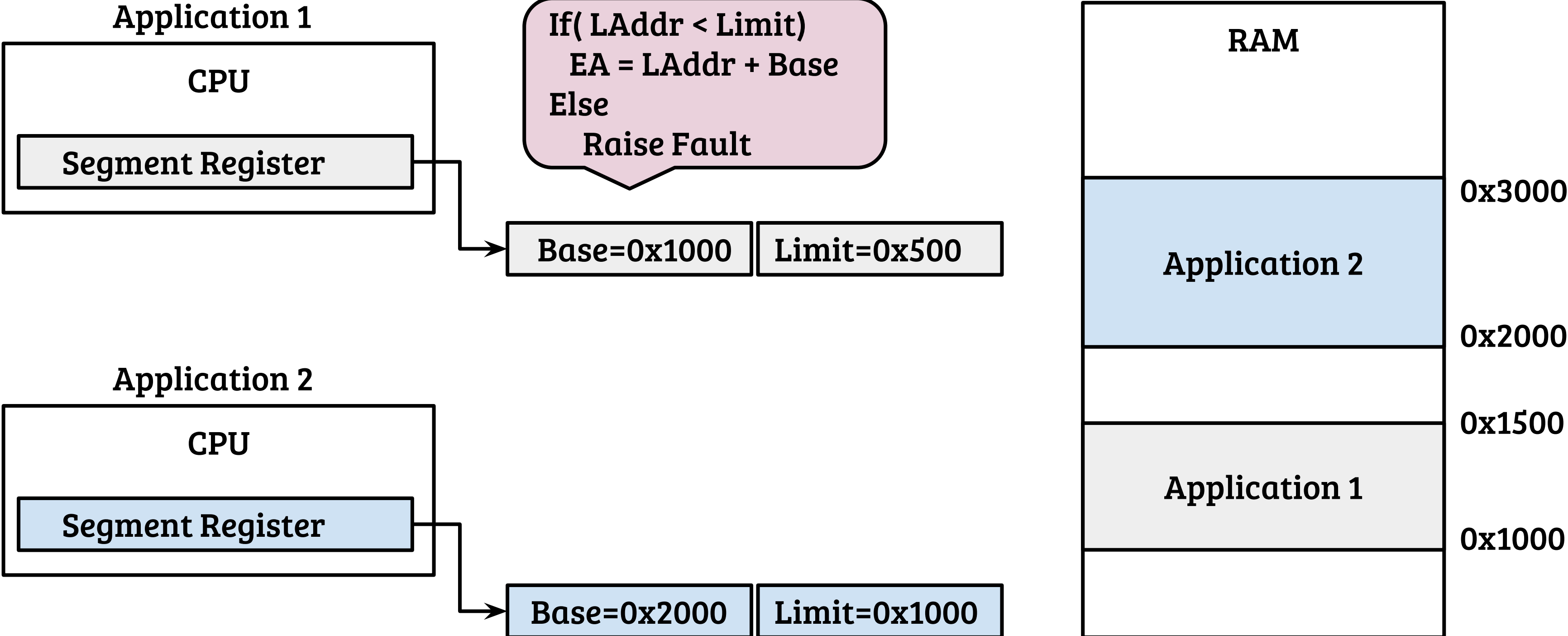
```
code + 0x0:    mov $0, %rcx
code + 0x4:    mov  %stack, %rsp
code + 0x8:    push %rbp
code + 0xa:    mov %rsp, %rbp
code + 0xc:    mov (%data + 0x8, %rcx, 0x8), %rdi
code + 0x10:  add (%data), %rdi
code + 0x14:  mov  %rdi, (%data + 0x8,%rcx, 0x8 )
code + 0x18:   inc %rcx
code + 0x1a:   xor %rcx, $10
code + 0x1c:   jnz (%code + $0x10)

/*Initialization*/
(%data)             .long 0x500
(%data + 0x8)   .long 0    REP (10)
(%stack)           .long 0x0
```

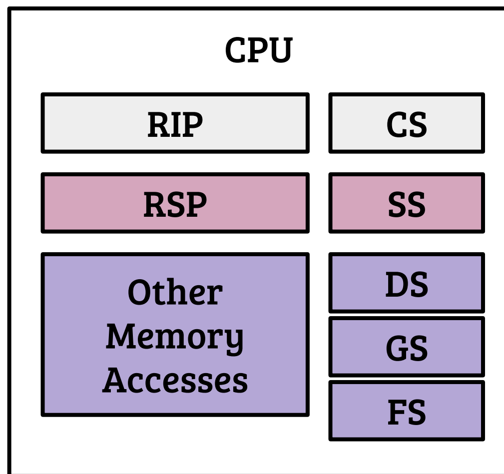
code = 0x1000, data = 0x10000, stack = 0x20000

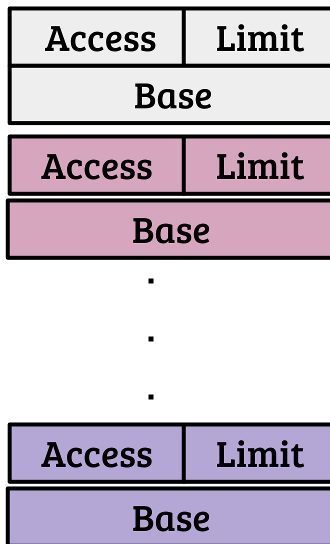
```
u64 value = 0x500;
u64 array[10];
main ( )
{
        u64 ctr = 0;
        for (ctr=0; ctr<10; ctr++)
              array[i] += value;
}
```

Memory segmentation



Memory segmentation (X86)



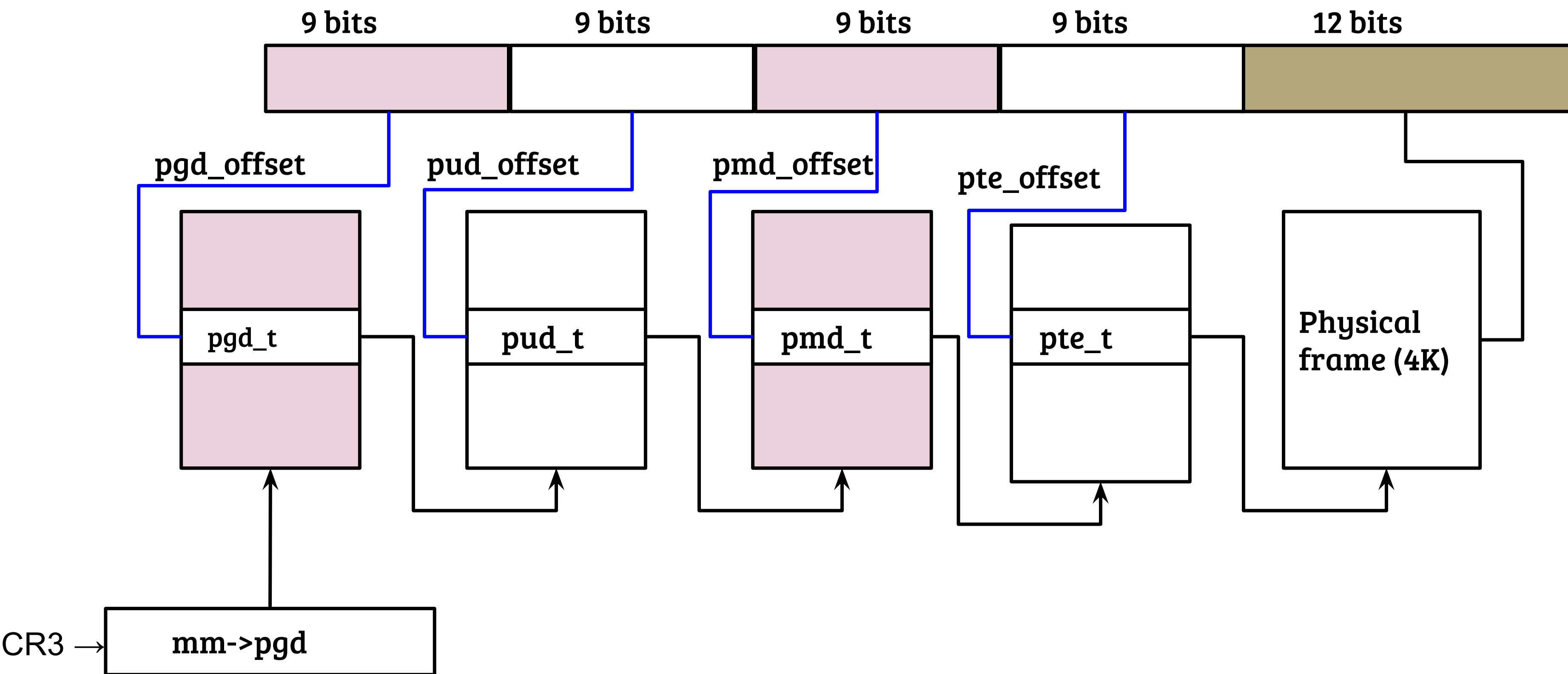


- Segment descriptor table
- Accessible from ring-0
- Global descriptor table and local descriptor table (LGDT, SGDT)
- In 64-bit, segmentation is minimally used
  - Flat segmentation model
  - Used to implement privileges, entry gates (for interrupt, exception etc.)

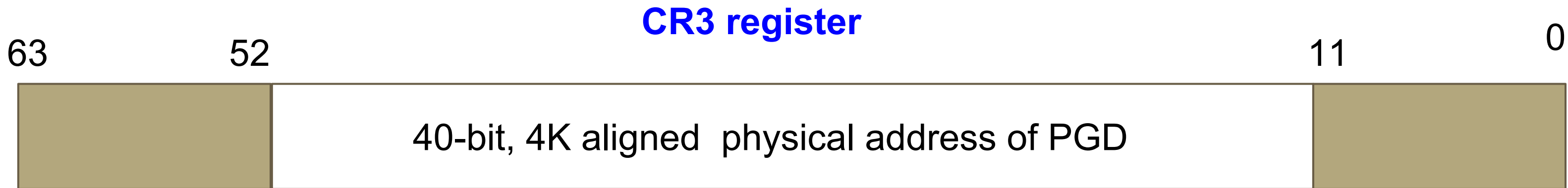
Segmentation: granularity issue

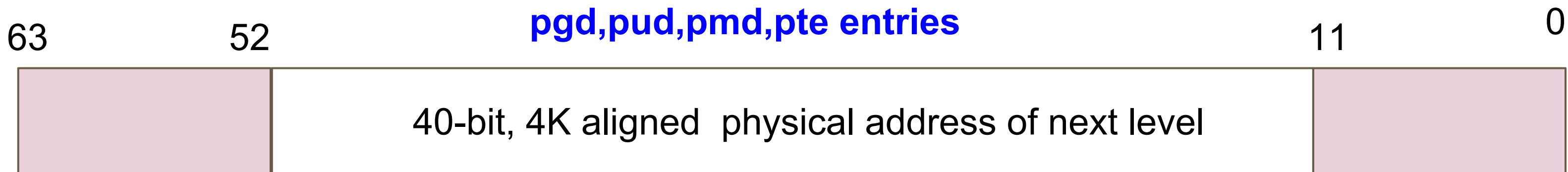
- Theoretically, segmentations is not a problem
  - We can have “a lot of” segments for an active application
  - One caveat though, which is?
- How can we address the hardware limitation?
  - Segmentation is like a guided (through pointers) one-step translation mechanism
  - Can we expand it to multi-step lookup?
  - Data structures?
- Design attributes
  - Minimize translation overheads $\longrightarrow$ lookup latency, memory usage
  - Support for sparse and dynamic mappings $\longrightarrow$ lazy allocation, memory sharing

4-level page tables (48-bit virtual address)



X86_64 page table entries (48-bit)





Some important flags

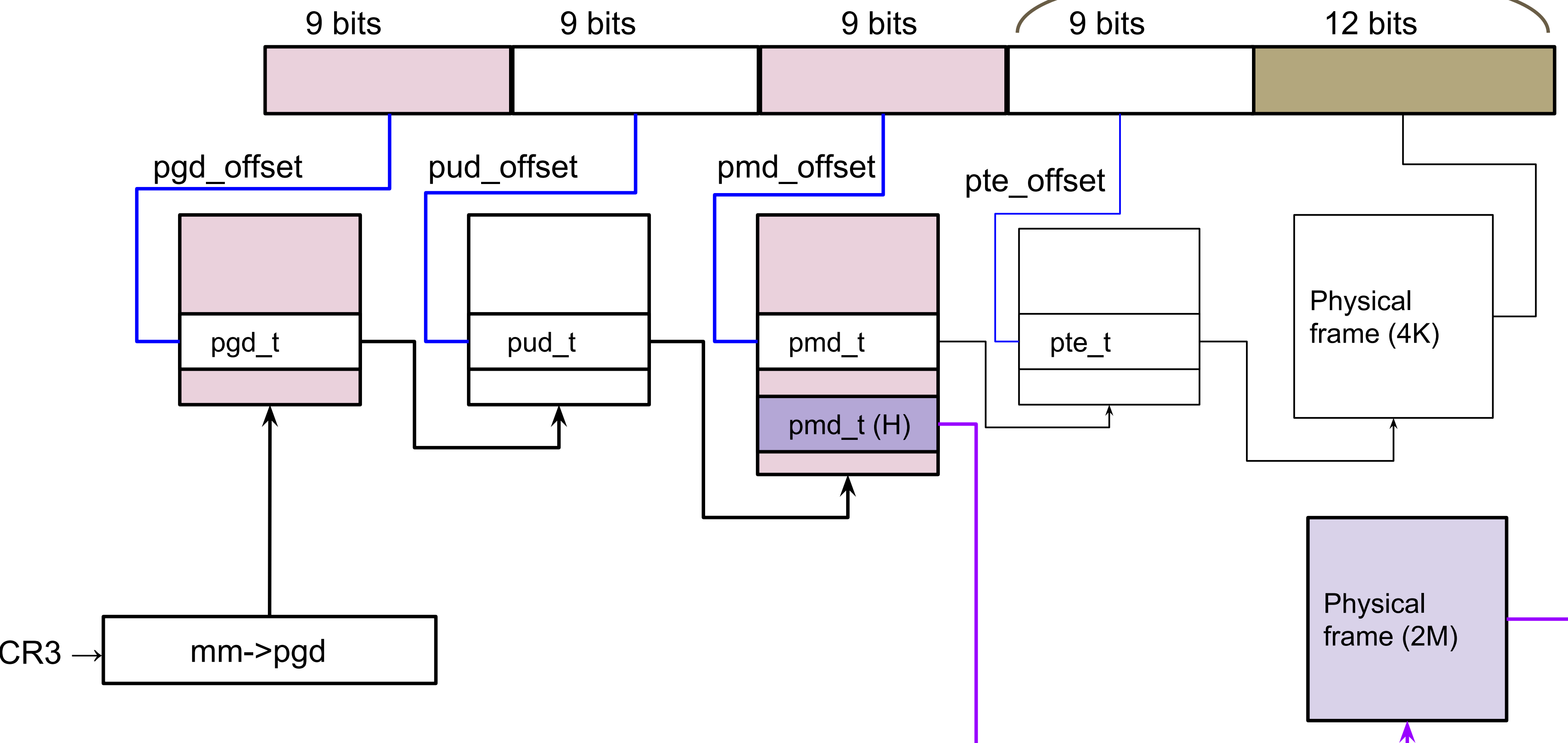
0 (present/absent) 1 (read/write) 2 (user/supervisor), 5(accessed) 7(huge page) 63(execute permissions)

*Source: Intel manual Vol: 3A 4.5

Paging: design parameters

- Design issue: Translation overheads
  - Virtual address size → number of translation levels → translation overhead
  - Large page size → reduced number of translation levels, memory fragmentation issues
- Hybrid page size support
- Collapse page tables starting from lowest level (pte)
  - pte level is removed and pmd addresses 21-bits = 2MB
  - pte and pmd are removed, pud addresses 30-bits = 1GB

Paging: mixed page size support



Paging: translation efficiency

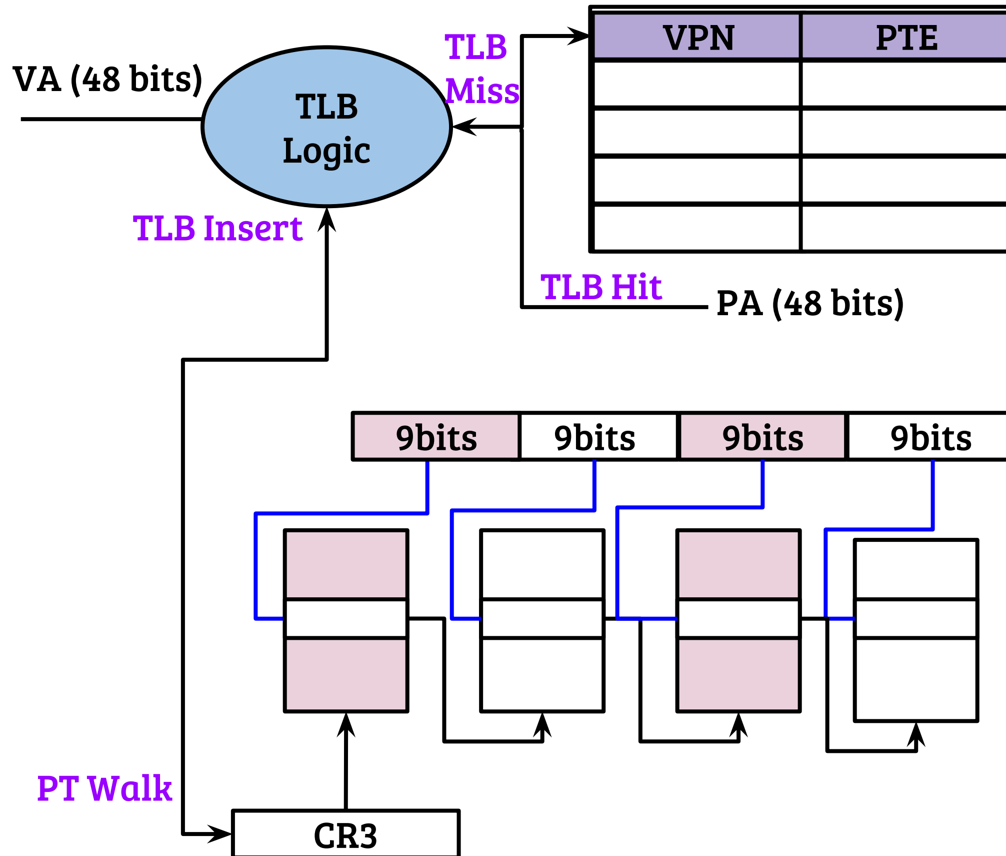
```
unsigned  long *V1, *V2, *V3;
int size = 32 * 1024;

for ( ctr=0; ctr < size; ctr++){
      V3[ctr] = V1[ctr] + V2[ctr];
}

/* RSP = 0x8000000 - 0x7FFF000,
   RIP = 0x10000 - 0x10080
   V1 = 0x4000000,
   V2 = 0x4200000,
   V3 = 0x4400000*/
```

- Consider 4-levels of translation, 48-bit virtual address
- Memory accesses required for translation, considering
  - Code execution, Data access and Stack operations
- Caches (L1, L2 etc.) can help, How?
- L1, L2 Caches are not sufficient, Why?
- A specialized cache to store recent translations required

Translation Lookaside buffer (TLB)



- TLB stores VPN to PTE mapping
- Lookup {VPN = VA >> 12}
- Hit: Physical address = PA(PTE, VA)
- Miss: PTE = PTWalk (VPN), InsertTLB (VPN, PTE) , PhysicalAddress = (PTE, VA)

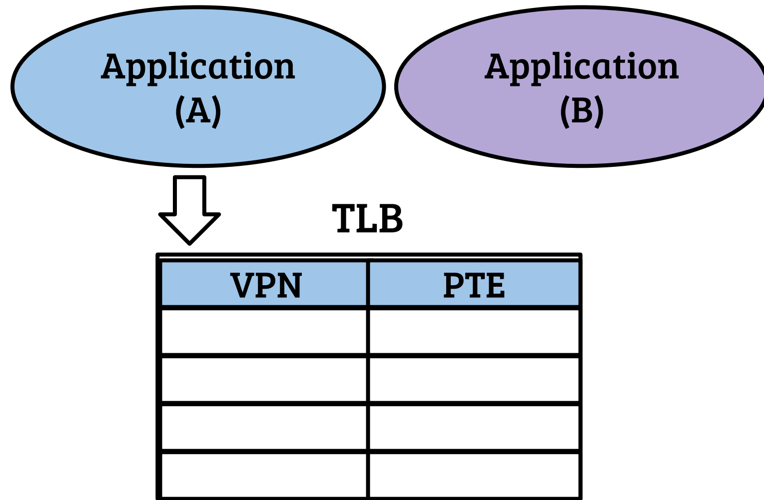
Paging with TLB: translation efficiency

TLB

VPN	PTE
0x10	
0x7FFF	
0x403E	
0x403F	
0x423E	
0x423F	
0x443E	
0x443F	

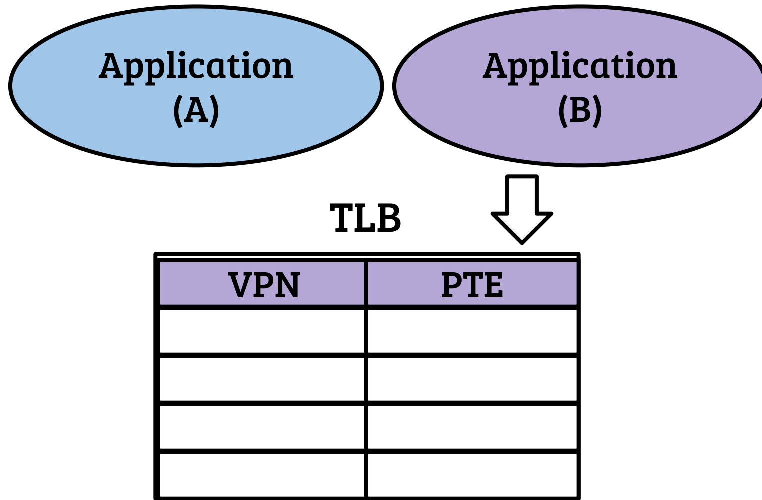
- TLB caches most recently used V to P translations
- How does TLB help addressing page table walk overheads?
- TLB + (L1, L2)
- When TLB fails to help?

TLB: Sharing across applications



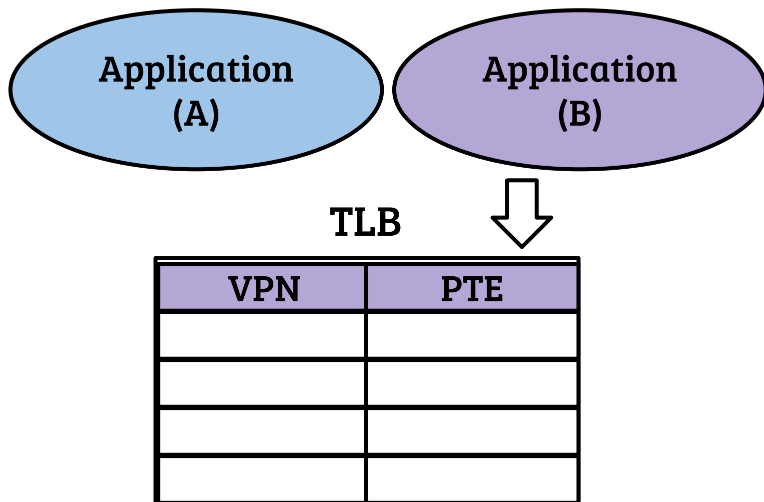
- What happens when OS schedules application B switching out A?

TLB: Sharing across applications



- What happens when OS schedules application B switching out A?
- Solution 1 : All entries of A are purged
  - Disadvantages?

TLB: Sharing across applications



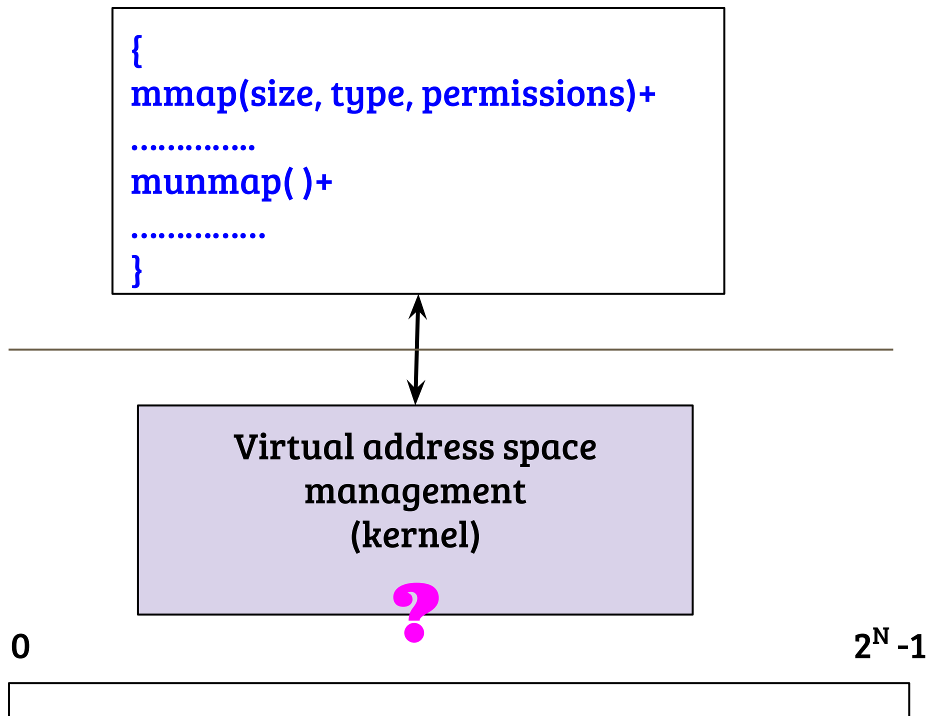
- What happens when OS schedules application B switching out A?
- Solution 1 : All entries of A are purged
  - Disadvantages?
- Solution 2: A and B share the TLB
  - How?

Virtual memory management

- Applications require memory with different properties
  - access permissions, sharing, file backed vs. anonymous
- /proc/{pid}/maps and mmap( ) system call

- Why OS should worry how user-space virtual addresses are managed?
  - let a user-space library handle it
  - only virtual to physical translation is managed by OS
  - Issues?

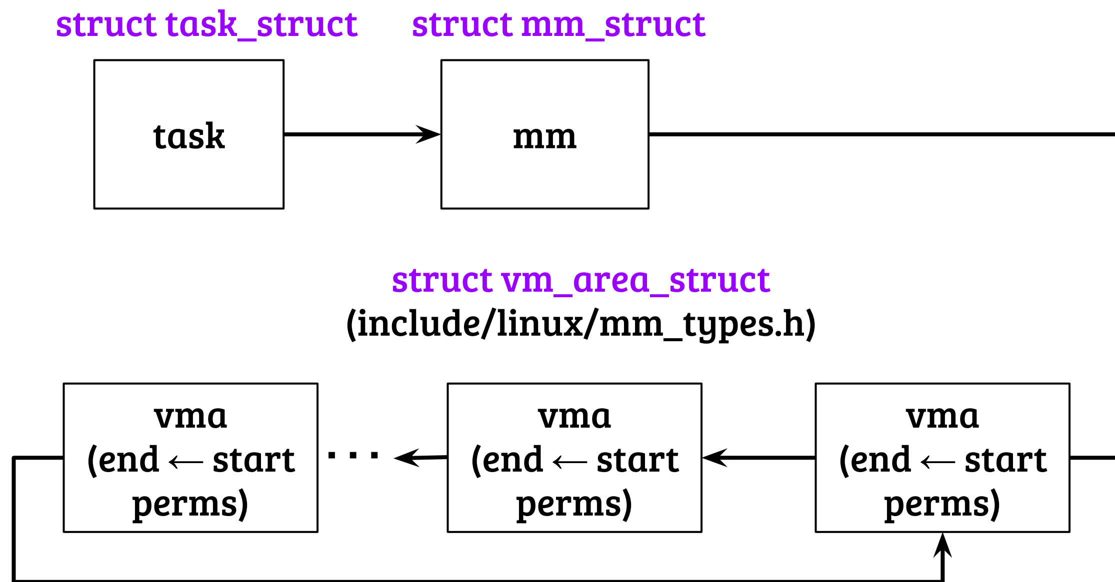
Virtual memory management



Virtual address space management alternatives

- contiguous allocation based on memory region type
  - Inflexible
- sparse allocation
  - sorted list of used ranges
  - scalability issues
  - Can be solved using balanced search trees

Virtual memory management



- start and end never overlaps between two vm areas
- can merge/extend vmas if permissions match
- linux maintains both rb_tree and a sorted list (see mm/filemap.c)