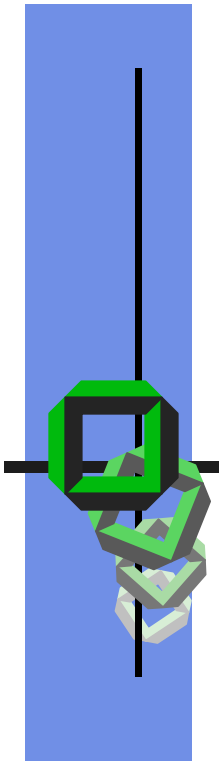
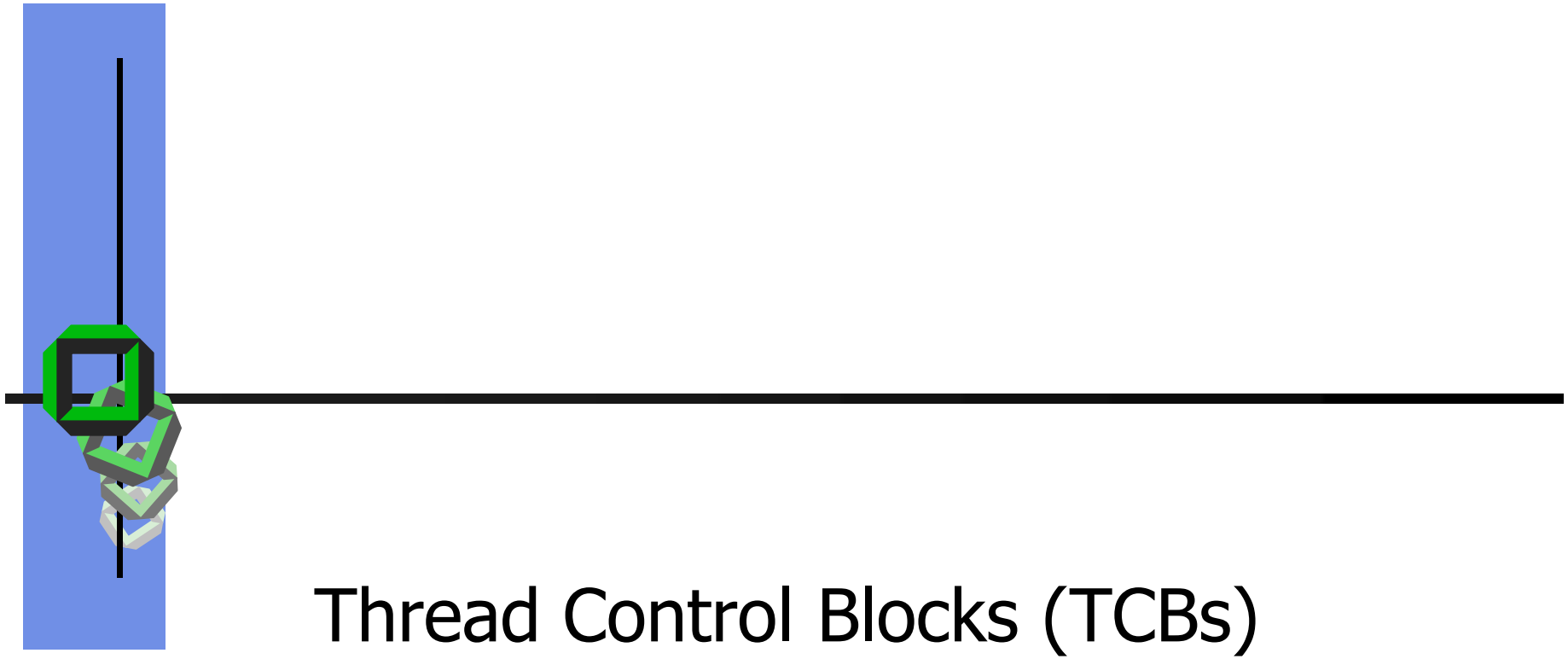
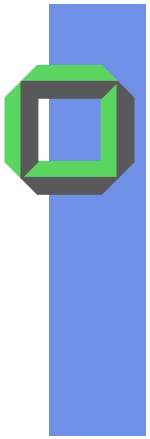


µ-Kernel Construction (3)

TCBs and Address-Space Layouts

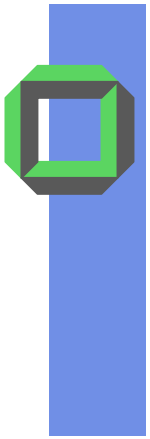


Thread Control Blocks (TCBs)



Fundamental Abstractions

- Thread
- Address space
  - What is a thread?
  - How to implement it?

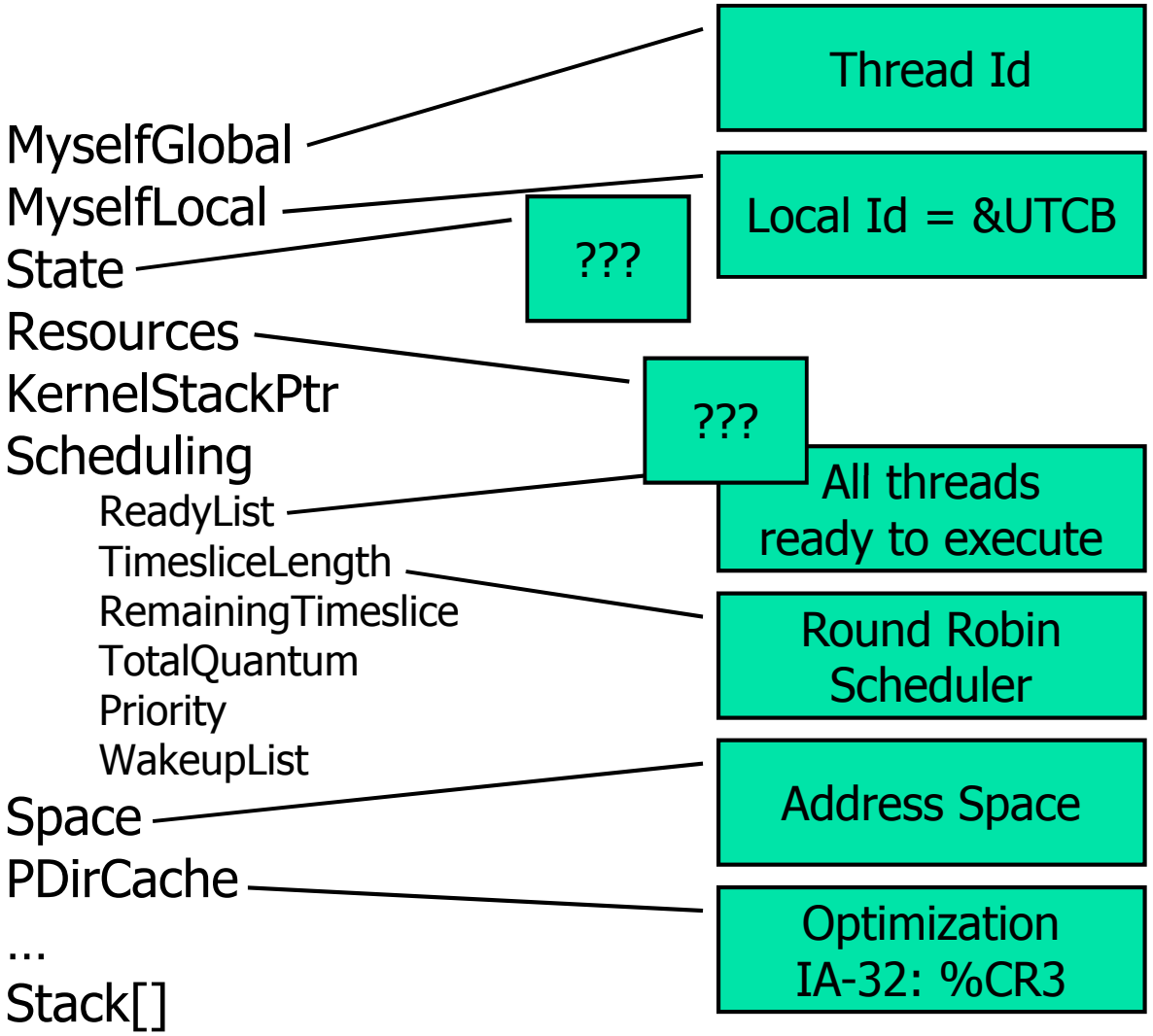


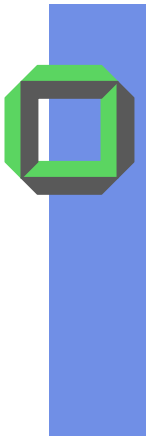
Construction Conclusion

- Thread state must be saved/restored on thread switch
- We need a Thread Control Block (TCB) per thread
- TCBs must be kernel objects
  - TCBs implement threads
- We need to find
  - Any thread's TCB using its global ID
  - The currently executing thread's TCB (per processor)

TCBs implement threads!

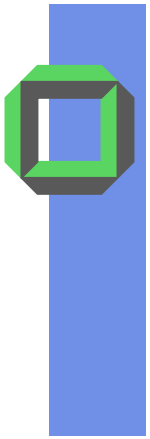
TCB Structure





Construction Conclusion

- Thread state must be saved/restored on thread switch
- We need a Thread Control Block (TCB) per thread
- TCBs must be kernel objects
  - TCBs implement threads
- We need to find
  - Any thread's TCB using its global ID
  - The currently executing thread's TCB (per processor)



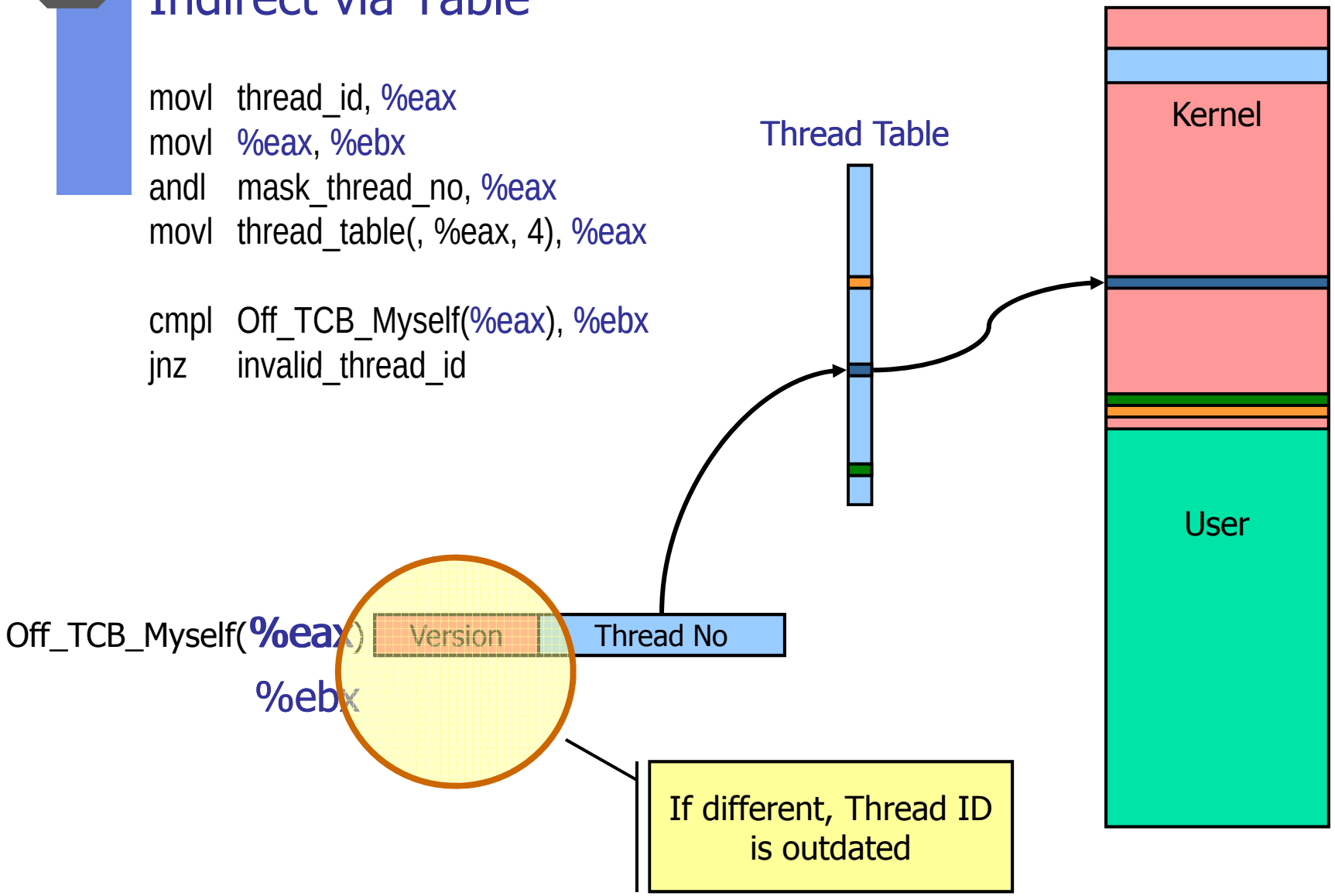
Thread ID

- Thread number
  - To find the TCB
- Thread version number
  - To make thread IDs “unique” in time

Thread ID → TCB

Indirect via Table

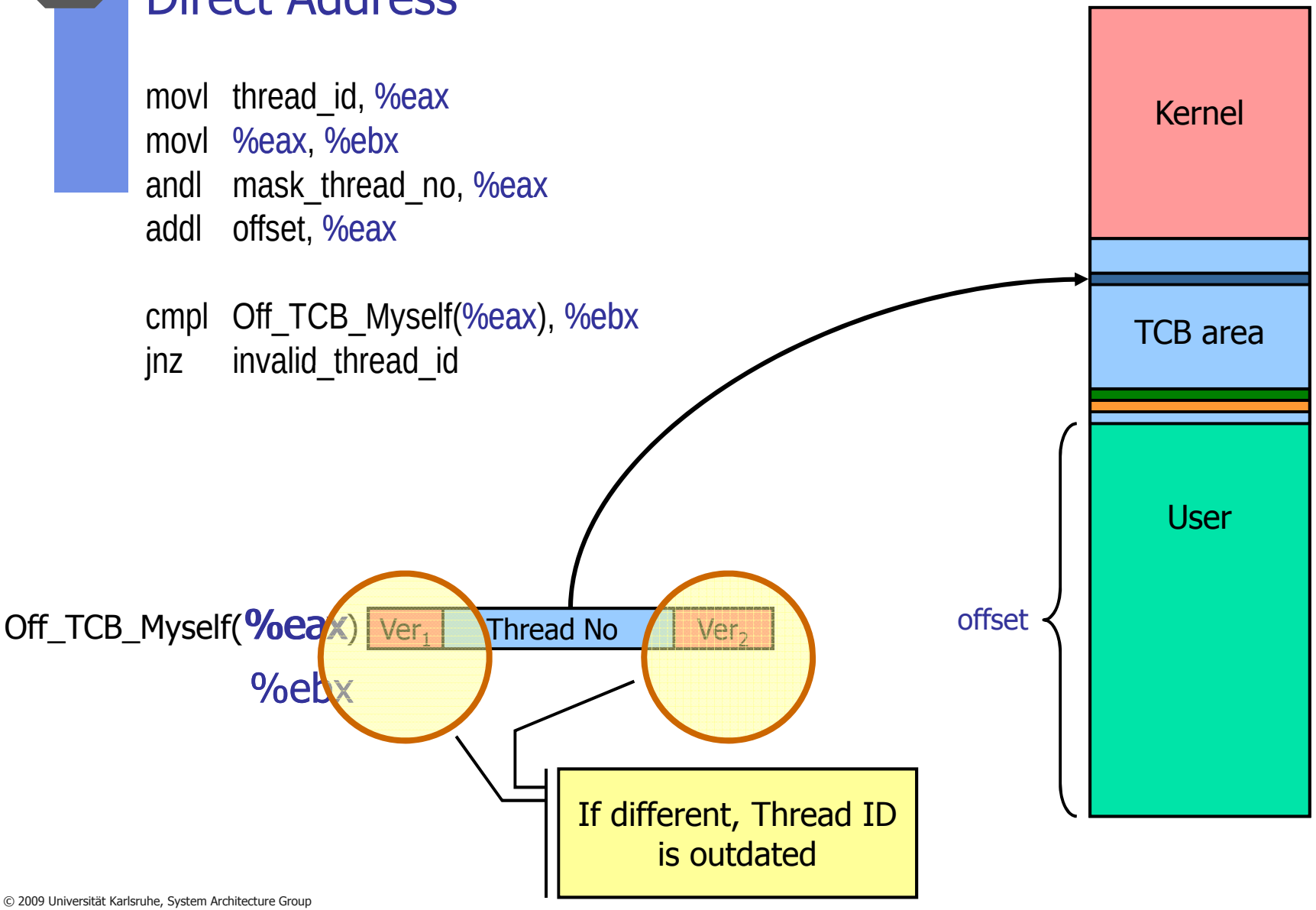
```
movl   thread_id, %eax
movl   %eax, %ebx
andl   mask_thread_no, %eax
movl   thread_table(, %eax, 4), %eax

cmpl   Off_TCB_Myself(%eax), %ebx
jnz    invalid_thread_id
```



Thread ID → TCB

Direct Address

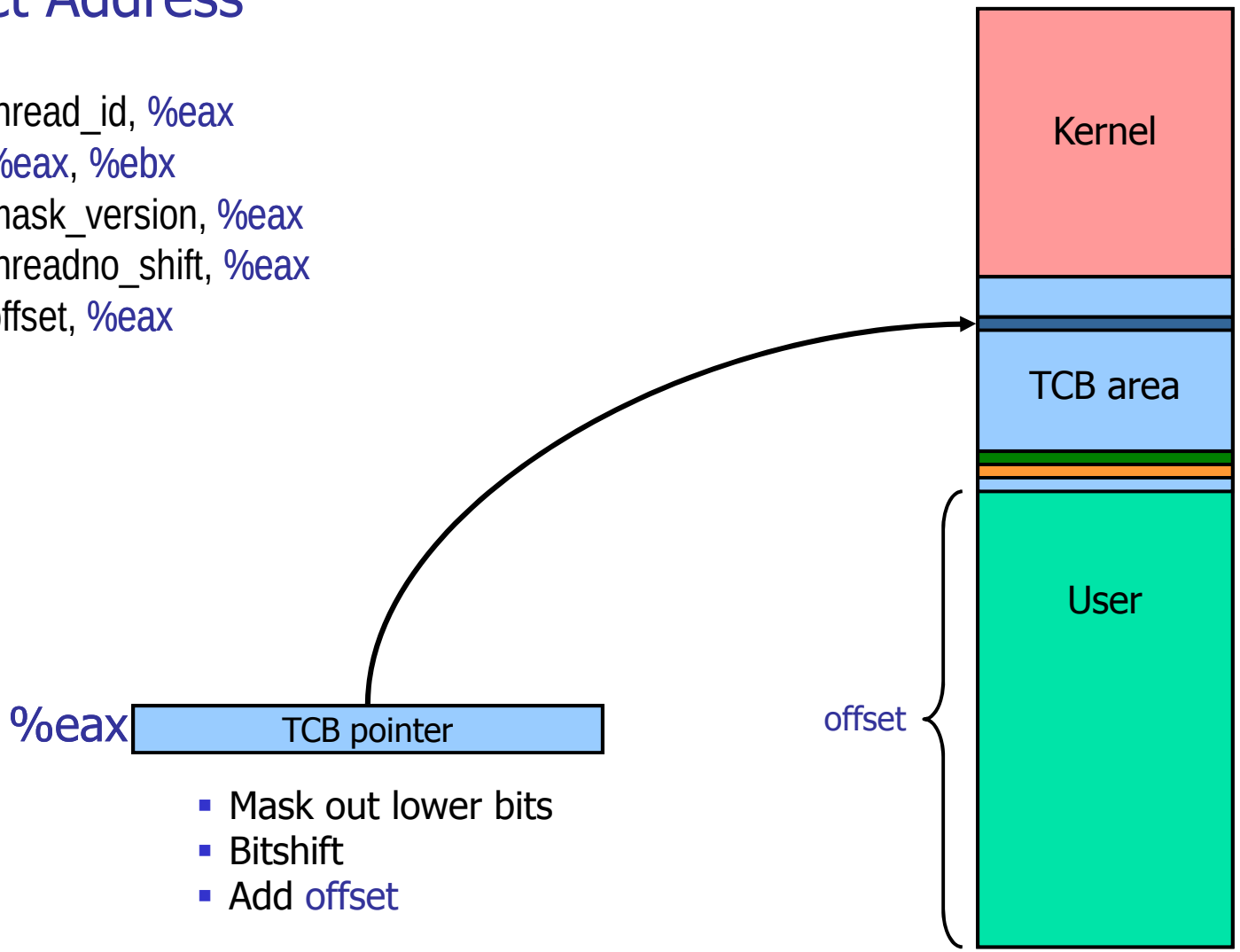
```
movl   thread_id, %eax
movl   %eax, %ebx
andl   mask_thread_no, %eax
addl   offset, %eax

cmpl   Off_TCB_Myself(%eax), %ebx
jnz    invalid_thread_id
```



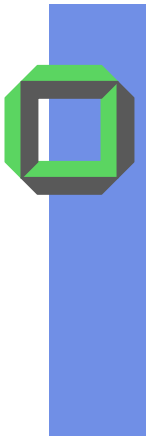
Thread ID → TCB

Direct Address

```
movl   thread_id, %eax
movl   %eax, %ebx
andl   mask_version, %eax
shrl   threadno_shift, %eax
addl   offset, %eax
```



- Mask out lower bits
- Bitshift
- Add offset

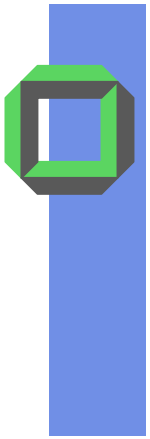


Thread ID Translation

Examples:
- 4 kB pages, 4 kB TCBs
  - ➡ 1 TCB per TLB entry
- 16 kB pages, 2 kB TCBs
  - ➡ 8 TCBs per TLB entry

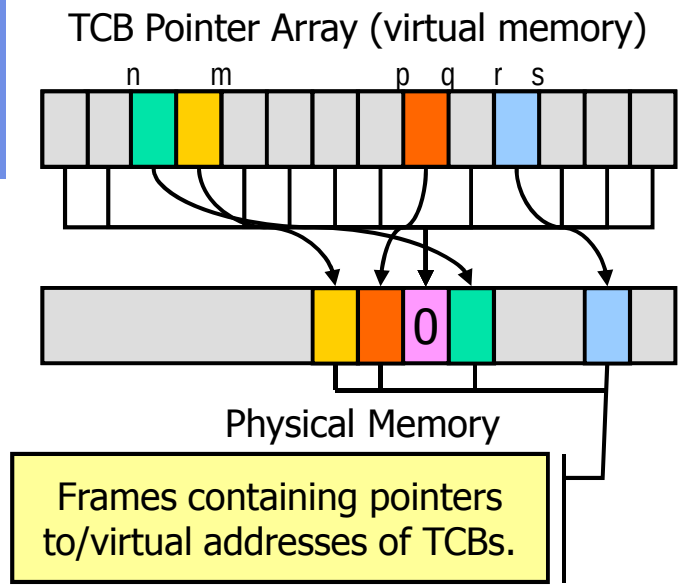
- **Via Table**
  - No MMU required
  - Table access per TCB
  - Many TCBs per TLB entry (TCBs on superpages)
  - TLB entry for table
- **Via Computation**
  - Requires MMU
  - No table access
  - Few TCBs per TLB entry (sparsely populated area)

- **TCB pointer array** requires 1M virtual memory for 256k potential threads
- **Virtual TCB array** may require ≥ 256M virtual memory for 256k potential TCBs



0-Mapping Trick

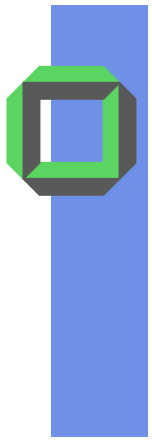
Indirect Addressing



- TCB pointer array requires 1M virtual memory for 256k potential threads

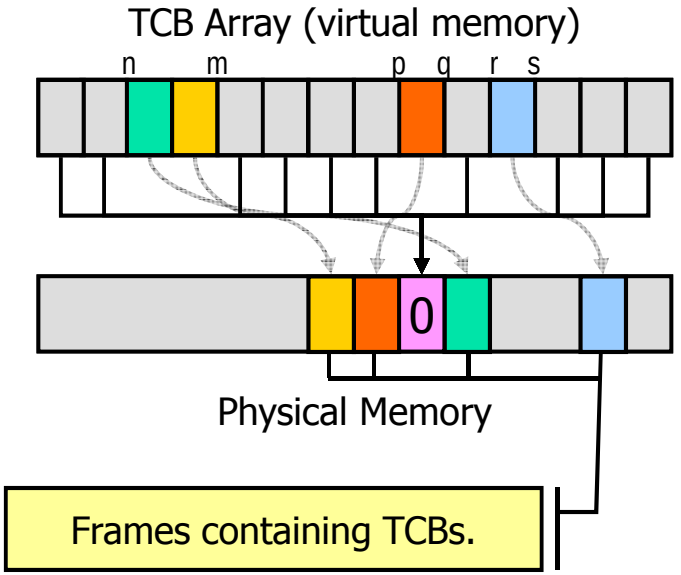
```
cmpl   Off_TCB_Myself(%eax), %ebx
jnz    invalid_thread_id
```

- Allocate physical parts of table on demand
  - Dependent on the max. number of allocated TCBs
- Map unused parts to a 0-filled read-only (r/o) page
  - Any access to unused threads will result in a NULL pointer
  - Requires extra check à la cmpl %eax, 0; jnz invalid
- Or: Map unused parts to a r/o page filled with pointers to a 0-filled r/o page
  - Any access to unused threads will result in an "invalid thread ID" (0)
  - Avoids additional check

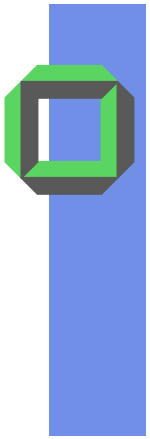


0-Mapping Trick
Direct Addressing

- Allocate physical memory for TCBs on demand
  - Dependent on the max. number of allocated TCBs
- Map all remaining TCBs to a 0-filled read-only page
  - Any access to unused threads will result in "invalid thread ID" (0)
  - Avoids additional check



- Virtual TCB array may require ≥ 256M virtual memory for 256k potential TCBs



Current State (IA-32)

- Virtual TCB array
- 18 bit TIDs, 14 bit version number
  - Max. 256k concurrent threads
- 2 kB per TCB
  - Includes kernel stack
  - Only ~256 B for the TCB proper
- 512 MB virtual memory
  - 50% of the kernel address space

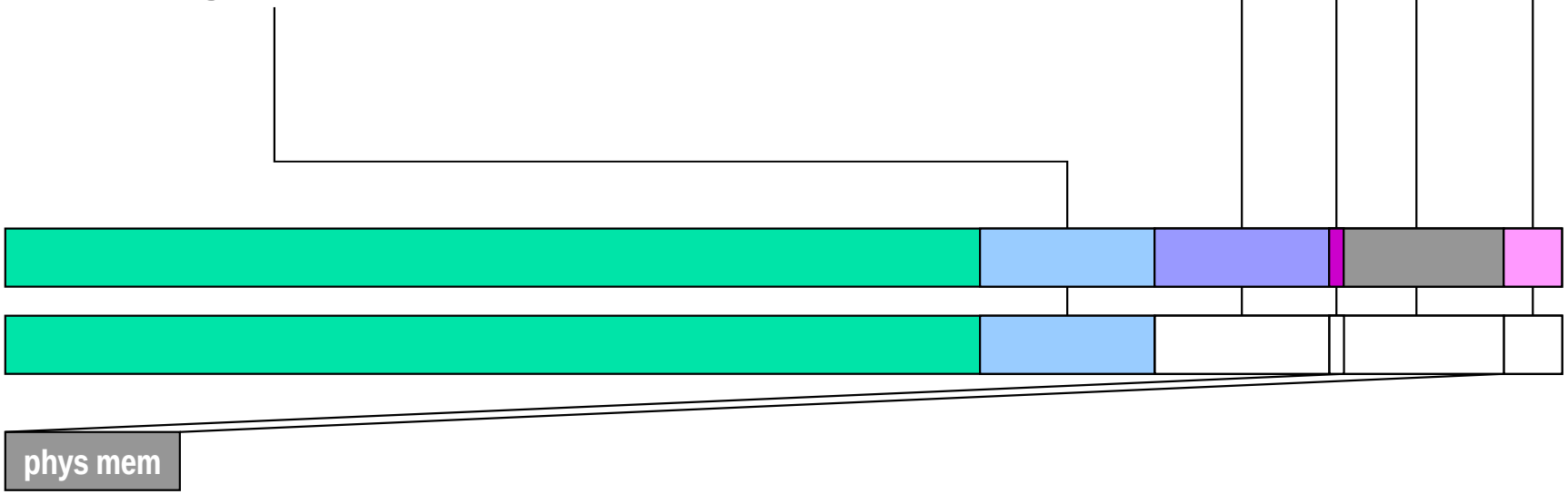
Basic Address-Space Layout

Address-Space Layout

32 bit, Virtual TCB Array

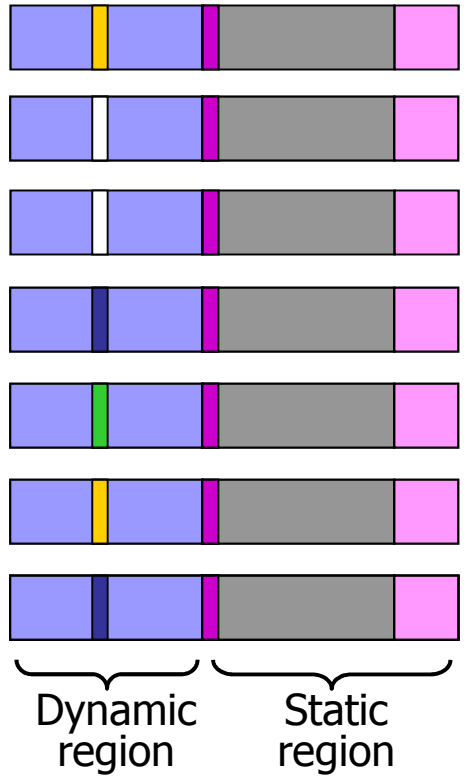
- User regions
- Shared system regions
- Per-space system regions

- Other kernel tables
- Physical memory
- Kernel code
- TCBs



Shared Region Synchronization

- We have
  - Regions shared among all address spaces
  - Separate page table per address space
- Updates occur in dynamic region
  - May lead to inconsistencies
- We need
  - Some form of synchronization within dynamic region
  - Make sure valid virtual memory mappings are synchronized

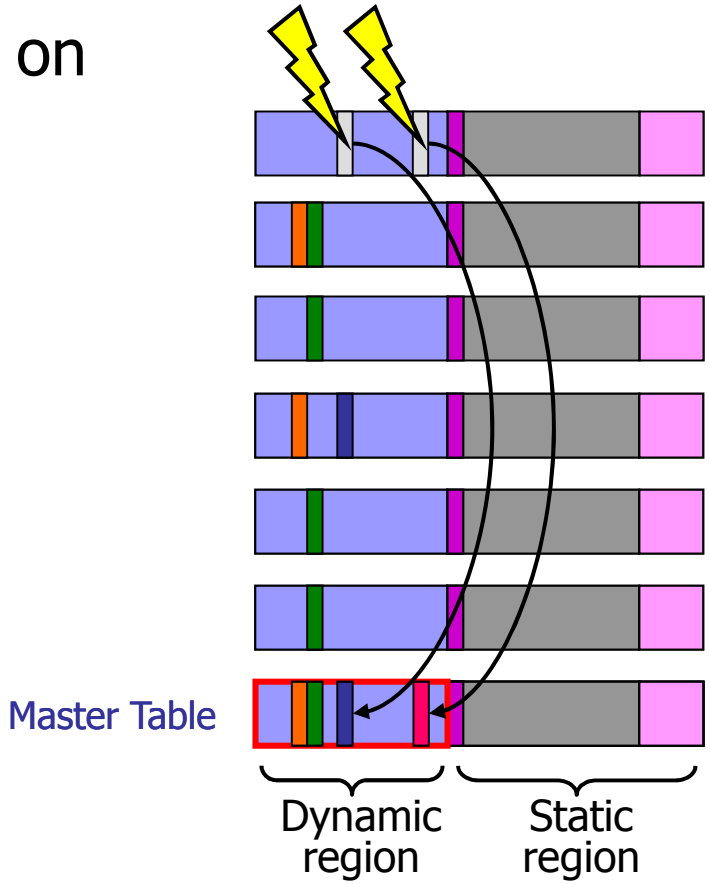


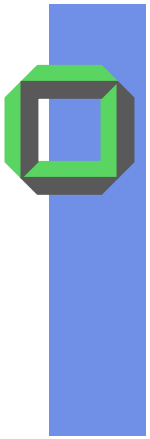
TCB Area Synchronization

Basic Algorithm

- Dedicate one table as master
- Synchronize with master table on page faults
  - Page fault algorithm:

```
if (master entry valid) {
      copy entry from master
} else {
      create new entry in master
      copy entry from master
}
```



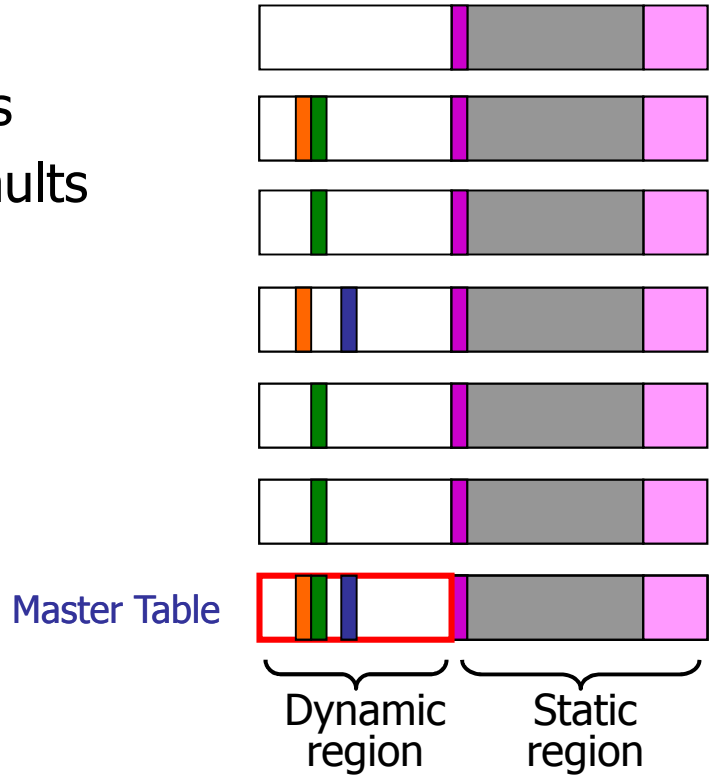


TCB Area Synchronization

Algorithm with 0-Mappings

- Use 0-mappings for invalid TCBs
- Thread creation requires TCB modification
  - Create 0-mappings on read faults
  - Create TCB mappings on write faults

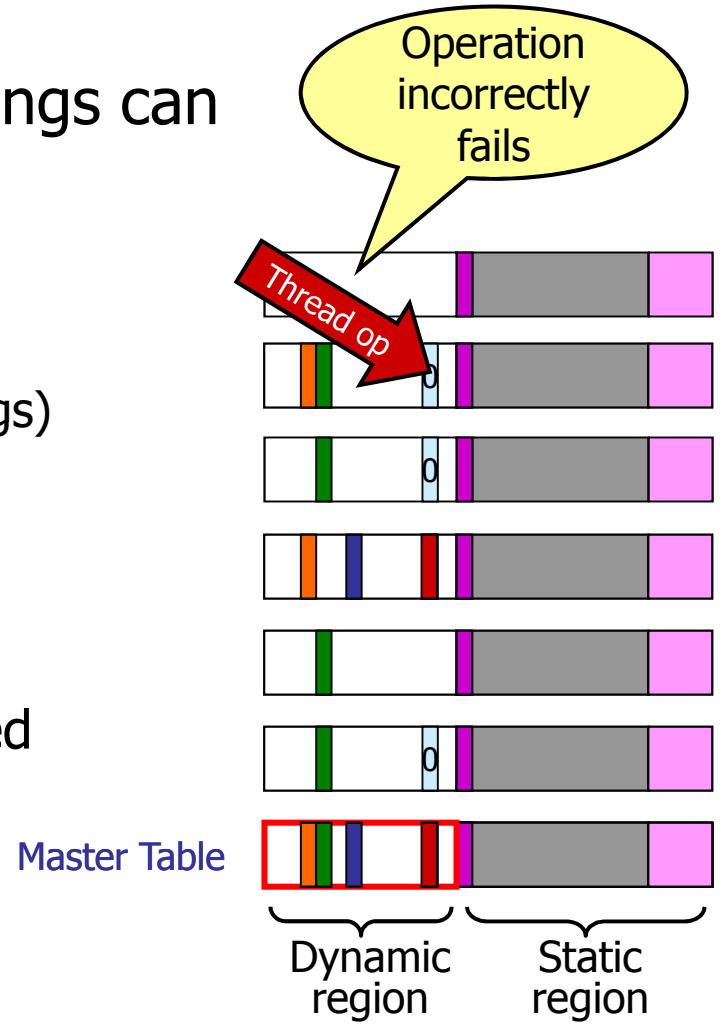
```
if (master entry not valid) {
      if (read fault) {
            create 0-mapping in master
      } else {
            create TCB entry in master
      }
}
copy entry from master
```



TCB Area Synchronization

Modifying Mappings

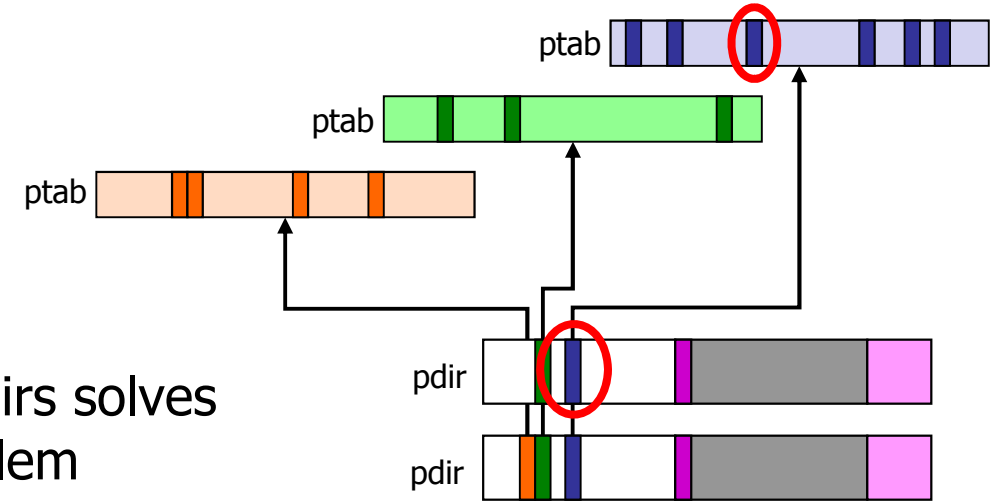
- Removing or modifying mappings can not be handled lazily
  - Must be handled brute force
  - Avoid removing mappings
    (i.e., do not remove TCB mappings)
- Potential problem
  - Create 0-Mappings (invalid TCBs)
  - Create a real TCB mapping
  - 0-Mappings must now be updated



TCB Area Synchronization

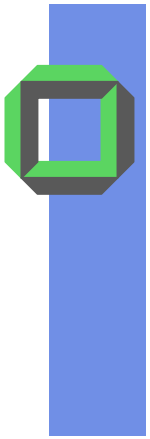
Modifying Mappings

- Page tables have multiple levels
  - IA-32: page directories and page tables
- We only synchronize top level (page directory)
- Modifications in lower levels visible in all spaces



Conclusion:

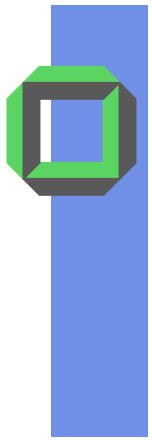
- Synchronization of pdirs solves the modification problem



Processor-Specific Memory

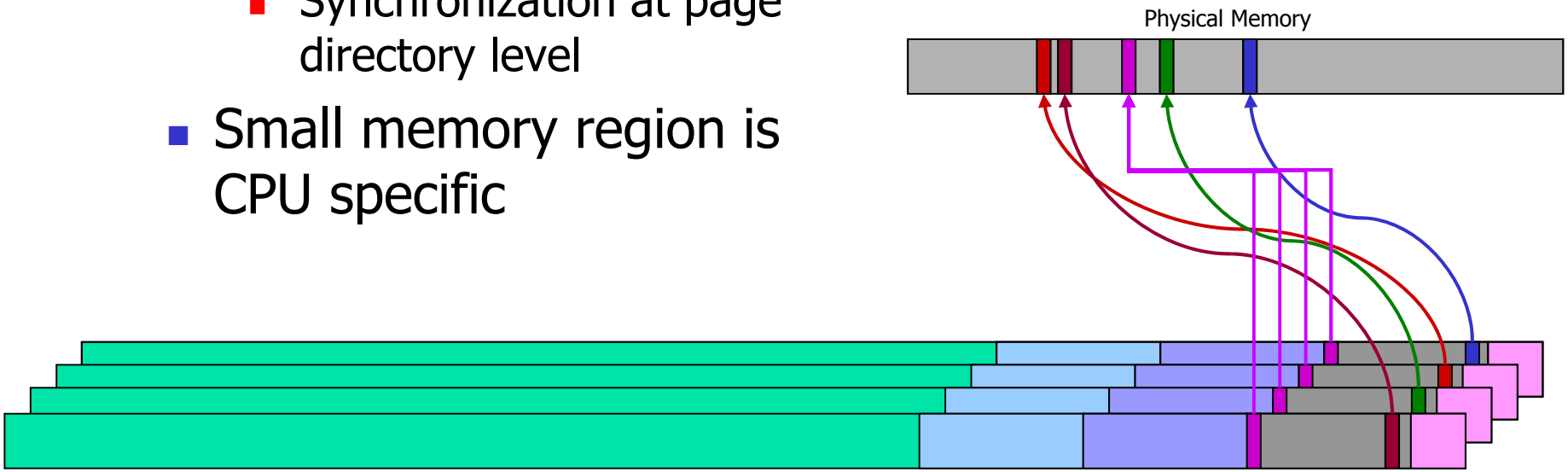
- Certain objects and variables should be processor local
  - Ready queues, CPU ID, etc.
  - Prevents cache conflicts
- Will require frequent access

- Solution: per-CPU memory regions
  - Same virtual address
  - Different backing store
  - Avoids indirection table
    (i.e., no extra memory access)



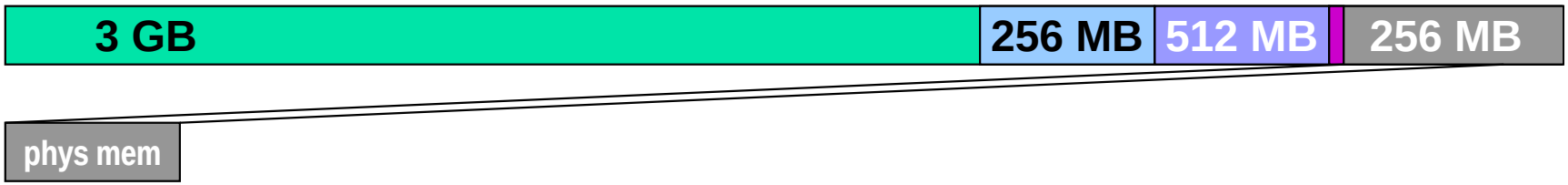
Processor-Specific Memory

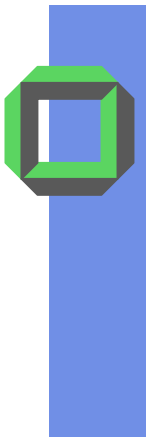
- One page table per CPU
- Most content identical
  - Requires synchronization
    (eagerly or lazily)
  - Synchronization at page directory level
- Small memory region is CPU specific



Limitations
32 bit, Virtual TCBs

- L4Ka::Pistachio/IA-32
  - TCB area size ➔ 256k threads
  - 256 MB physical memory window
    - Accessible at virtual address = physical address + offset
  - Remaining physical memory not directly kernel-accessible
    - Available for users and paged kernel data (e.g., TCBs)





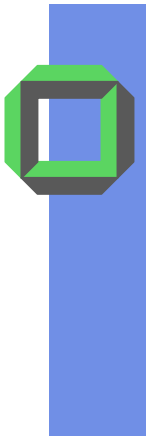
Physical Memory Window

- Used by the kernel for
  - Page tables
  - Kernel memory
  - Kernel debugger
- Only when kernel accesses physical addresses
  - Limit valid physical range to remap size (256 MB)
  - Or …

- Map and unmap
- Copy IPC

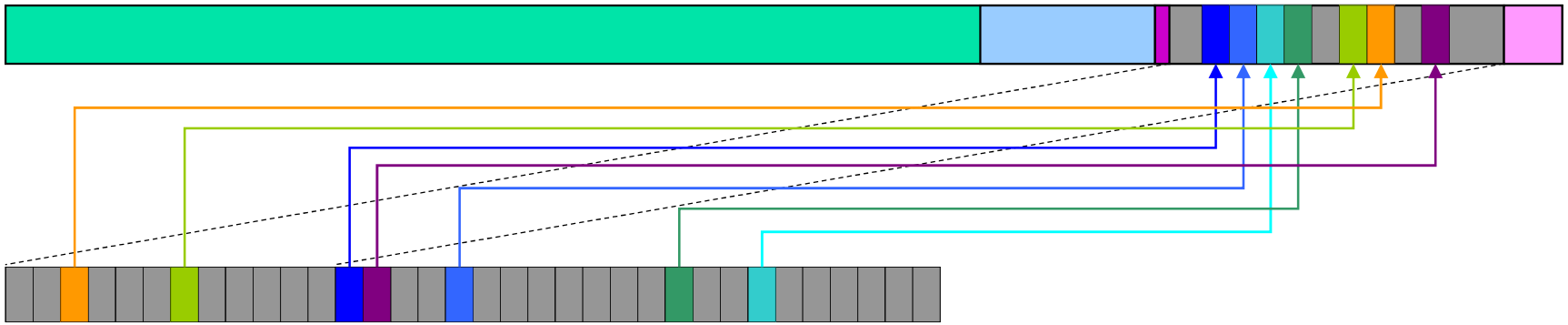
- Address spaces
- (TCBs)

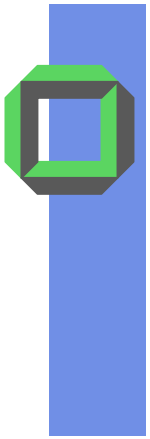
- KDB output
- Mem Dump



Physical-to-Virtual Pagetable

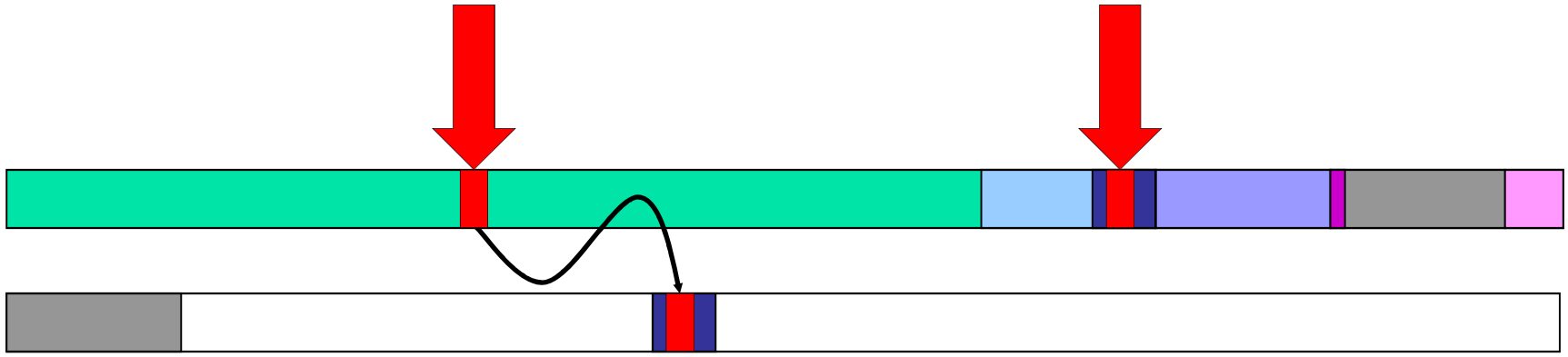
- Remap kernel-used pages
- Obtain virtual from physical address
  - Walk physical-to-virtual ptab in software
- Access physical memory via virtual address
- Costs?
  - Cache, TLB, runtime

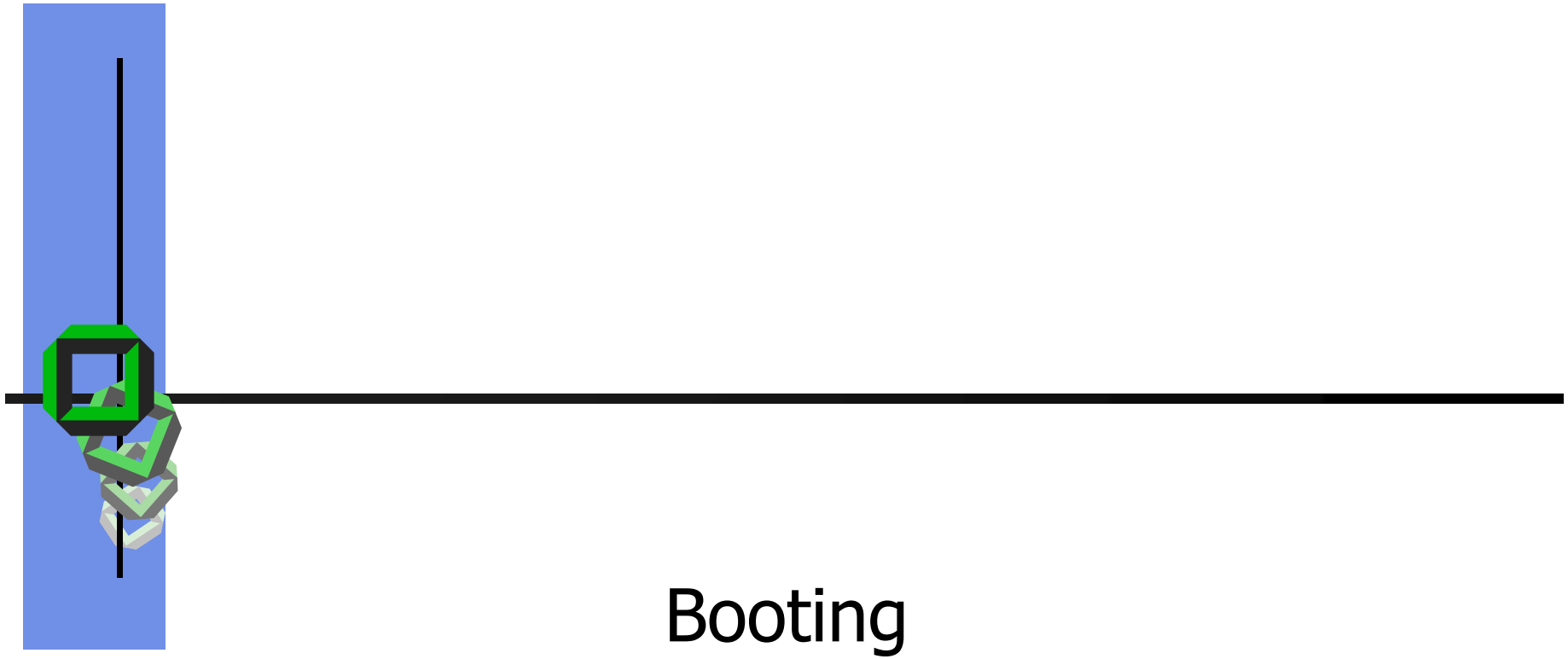




Kernel Debugger (not performance critical)

- Might want/need to access memory (maybe in different address space)
- Walk page table in software
- Remap on demand (4 MB)
- Optimization: check if already mapped

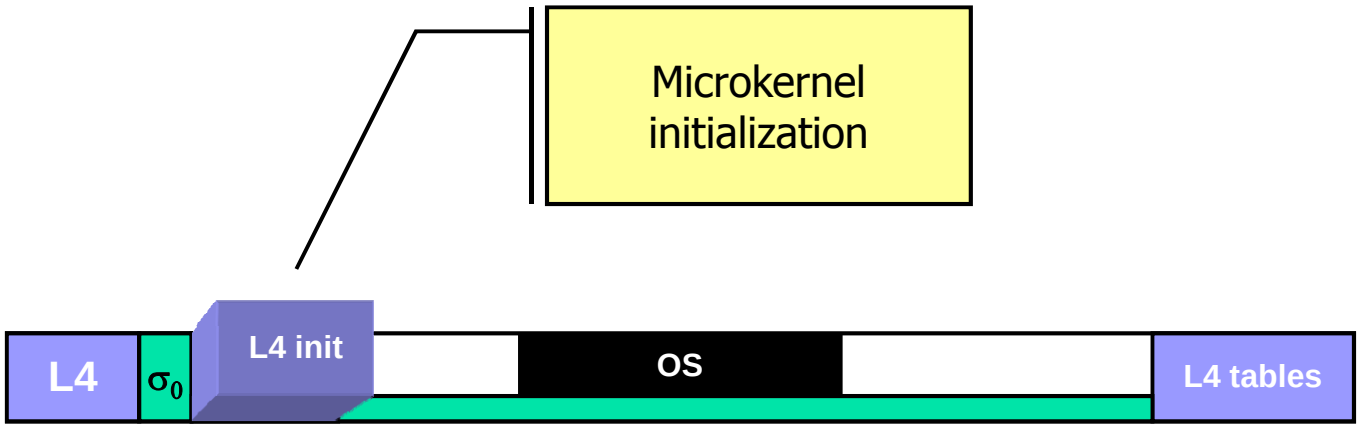


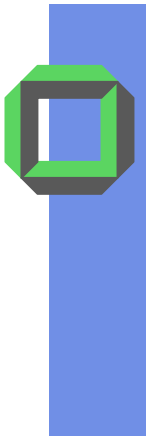


Booting

OS Booting

Microkernel Booting





Microkernel Booting

- Boot loader (e.g., Grub)
- Microkernel initialization
  - Basic memory
  - Basic VM
  - Exception handling
  - Processor (+ coprocessor)
  - Hardware interrupts
  - TCBs
  - Dispatching
- Create σ_0 & root task
- Release memory of init code
- Start σ_0 & root

- Build boot Pagetable
- Switch to virtual mode

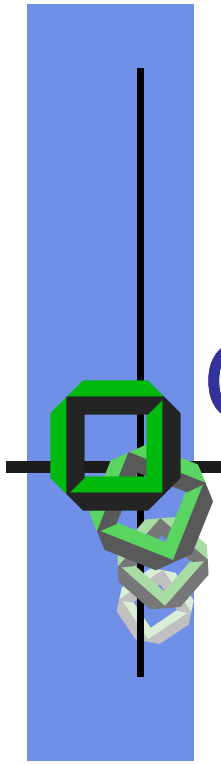
- IDT
- KDebug

- Check features
- GDT
- MSRs (sysenter/sysexit)
- FPU
- Local timer interrupt
- CPU-local pagetable

- IO-APIC/PIC

- Initialize first TCB
- switch_to

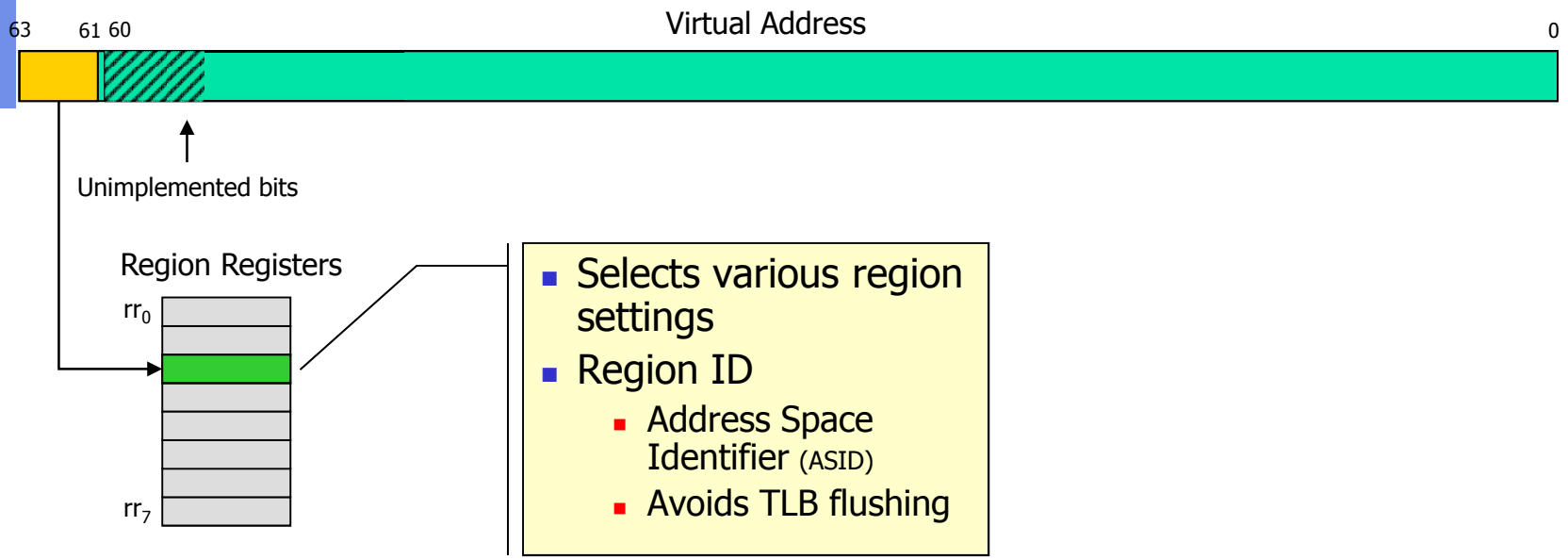
- Init ready list
- Init wakeup list



Case study: IA-64

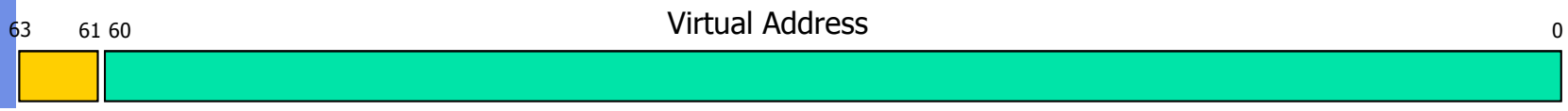
Address Space Layout and Memory Management

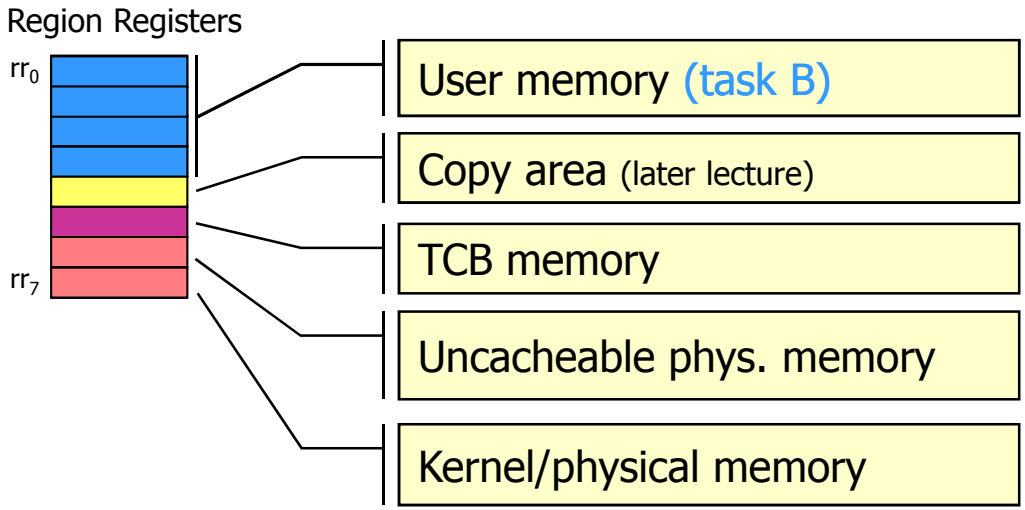
IA-64 Address Space Layout



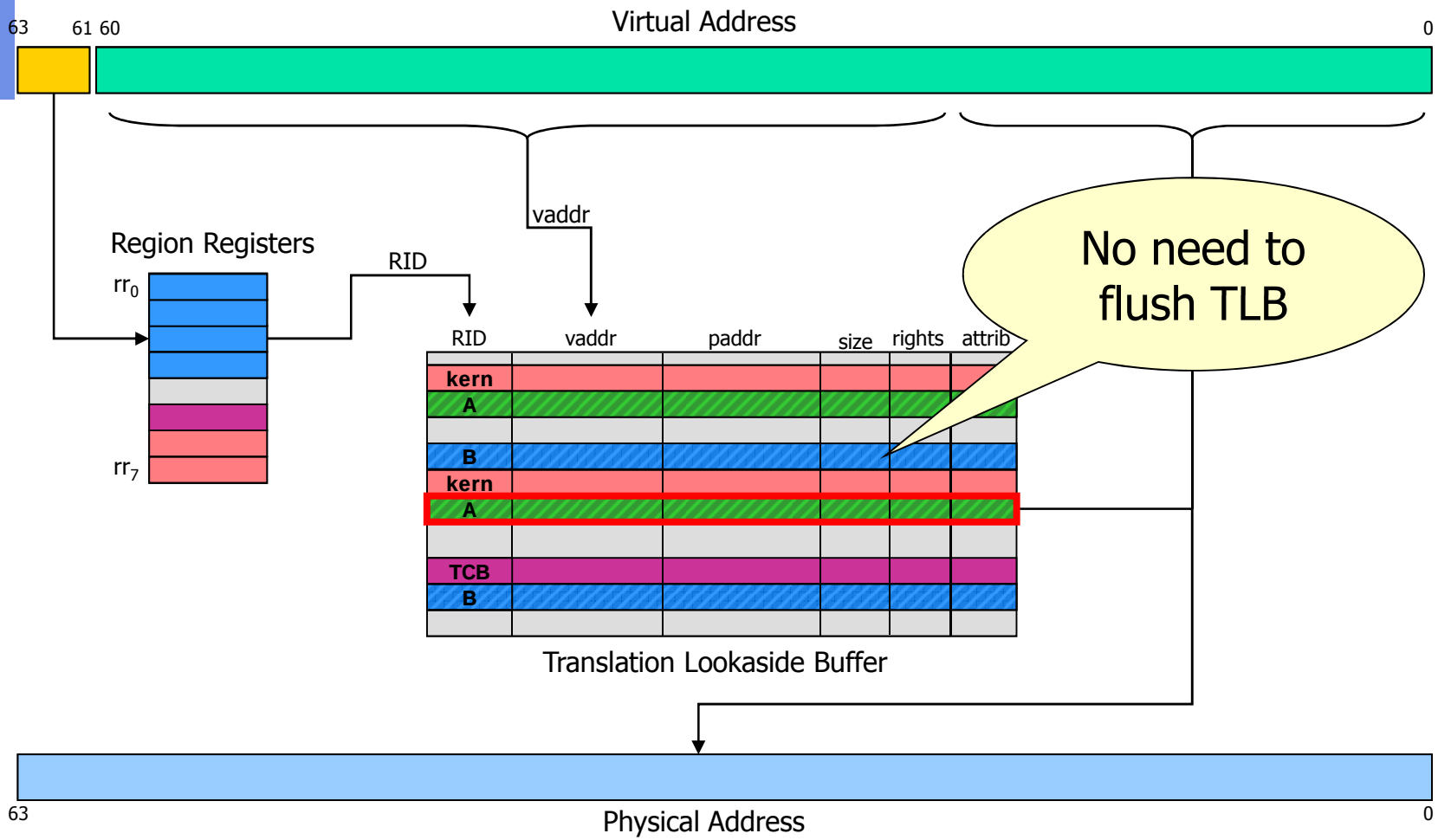
- Selects various region settings
- Region ID
  - Address Space Identifier (ASID)
  - Avoids TLB flushing

IA-64 Address Space Layout

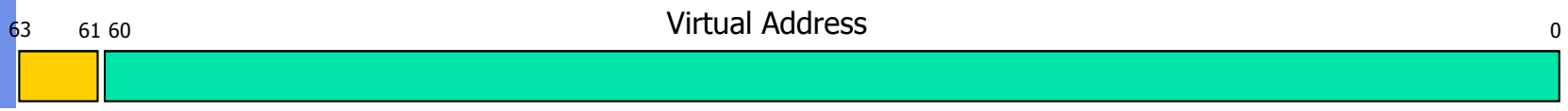




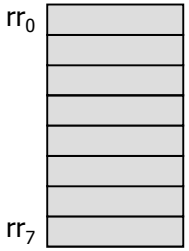
IA-64 Address Translation



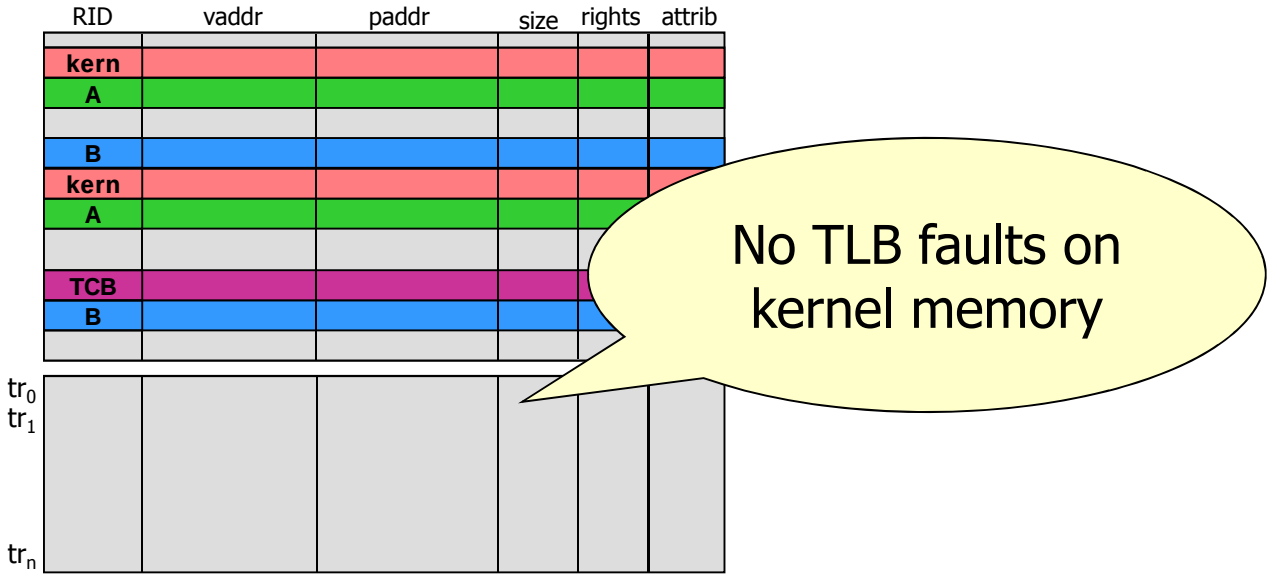
IA-64 TLB Pinning



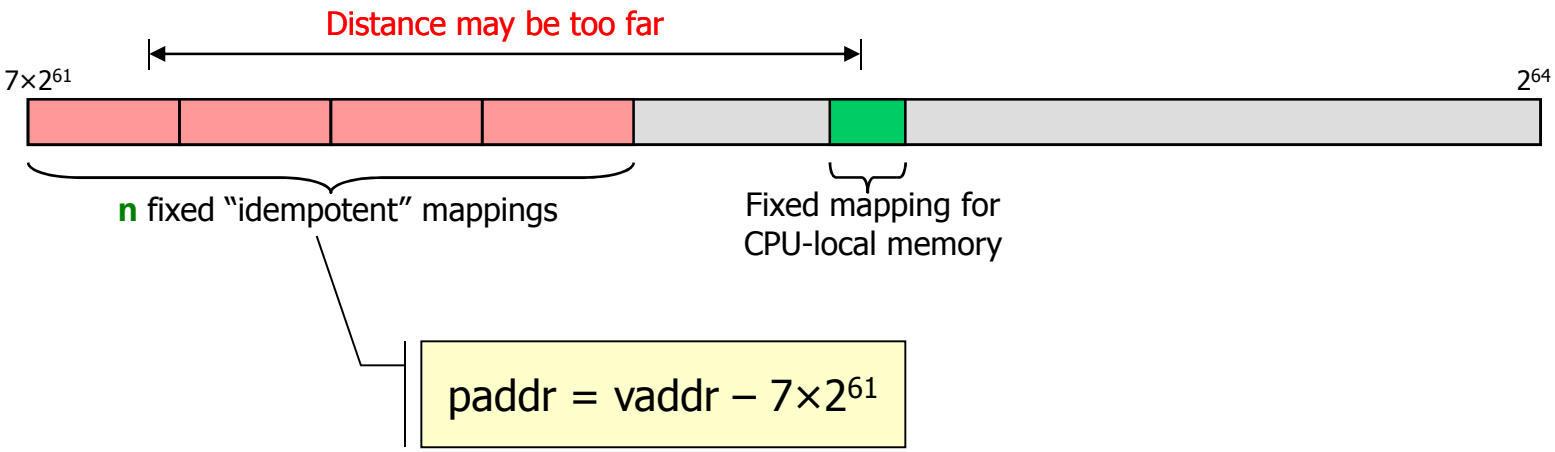
Region Registers



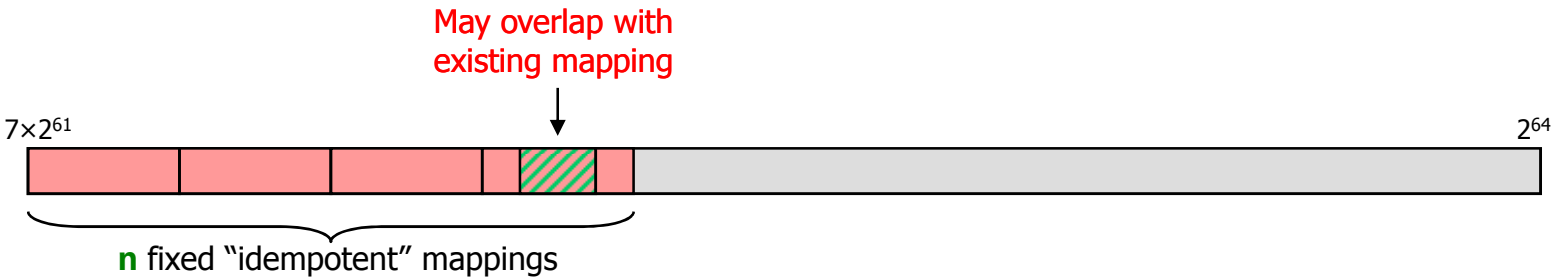
Translation Lookaside Buffer



IA-64 Kernel Region



IA-64 Kernel Region



IA-64 Kernel Region