Microkernel Construction

Threads and Address Spaces

SS2011

Udo Steinberg

TU Dresden
Operating
Systems Group

L4 Microkernel Abstractions

- Threads
  - What is a thread?
  - How are threads implemented?
- Address Spaces
- Inter-Process Communication

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread: Definition

- A thread ...
  - is an independent flow of control inside an address space
  - is identified by a unique thread identifier
  - communicates with other threads using IPC
  - is characterized by a set of registers and thread state information
  - is dispatched by the kernel according to a defined schedule

Udo Steinberg

TU Dresden
Operating
Systems Group

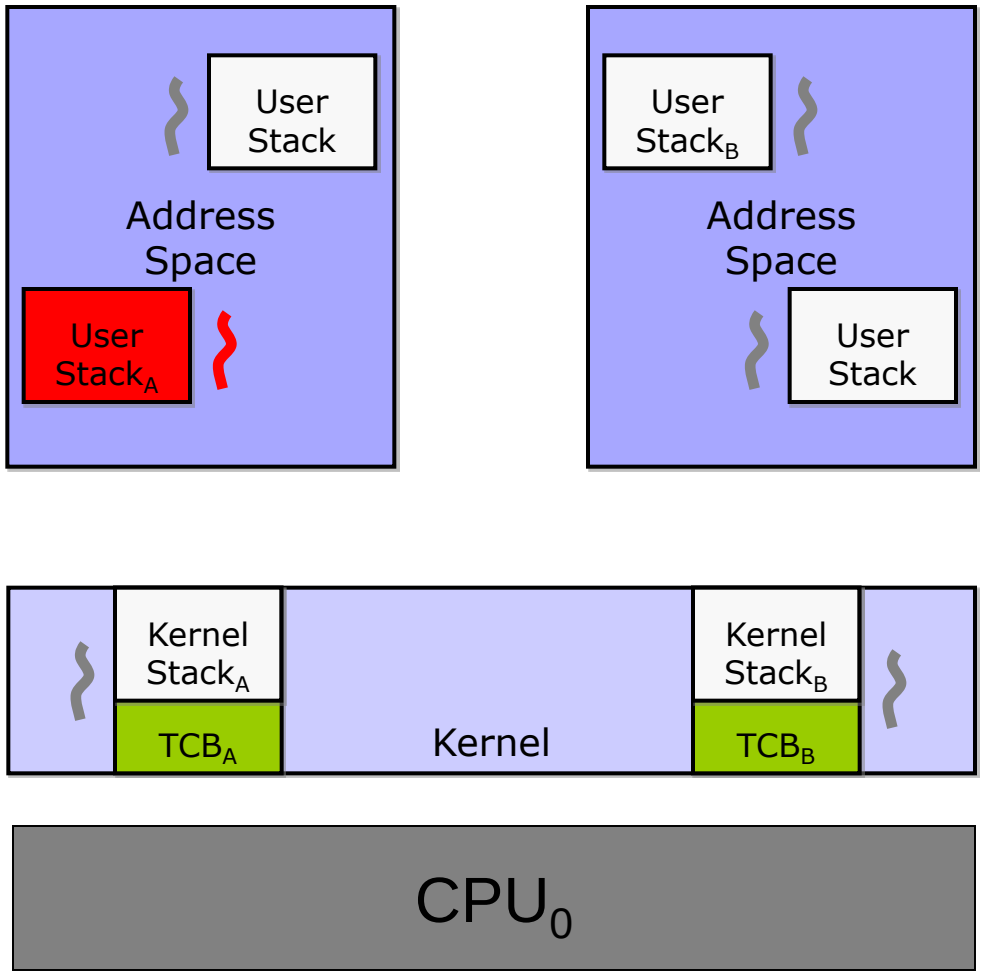
Thread Properties

- Thread ID
- State
  - running (thread is currently executing on the CPU)
  - ready (thread is waiting to execute on the CPU)
  - blocked (thread is waiting for an IPC rendezvous/timeout)
- Register Set
  - Instruction Pointer (IP)
  - Stack Pointer (SP)
  - General-Purpose Registers (GPRs)
- Stack
- Address Space
- Scheduling Parameters

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Control Flow



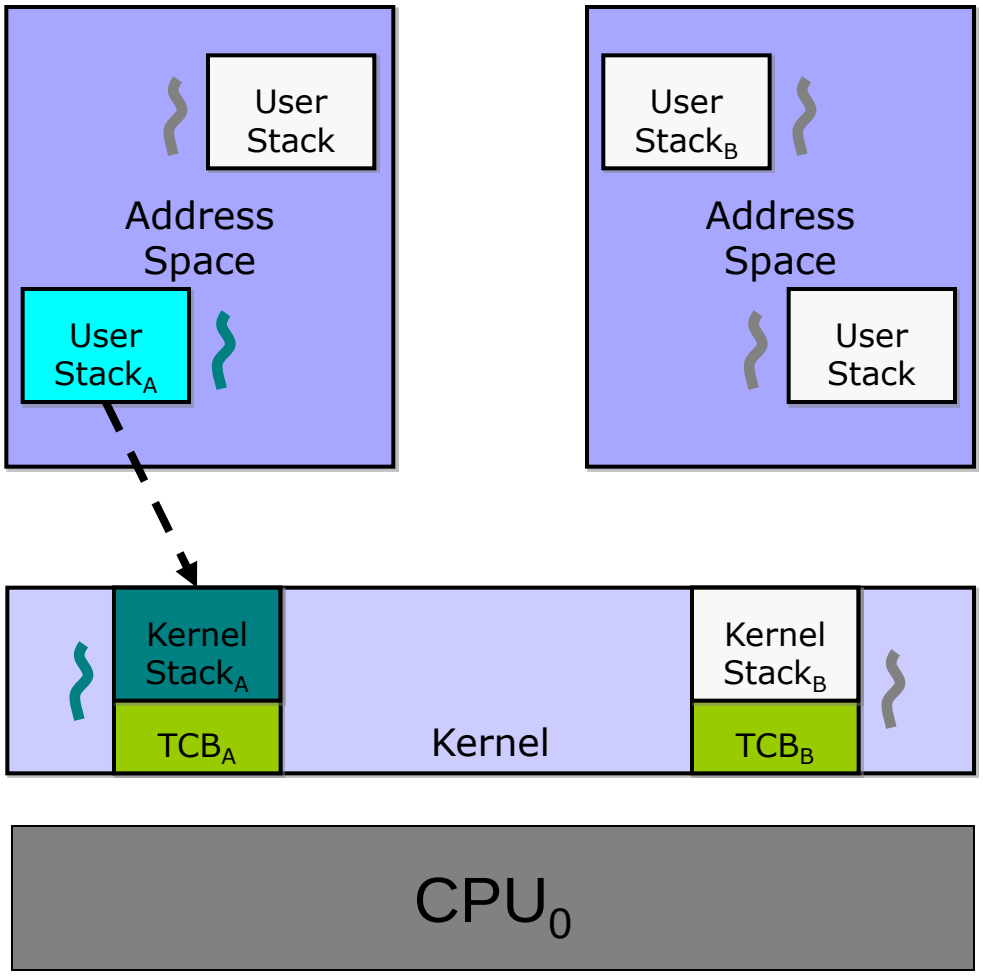
- User Part
  - Program code
  - Program data
  - User Stack

- Kernel Part
  - Kernel code
  - Kernel stack
  - TCB

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Control Flow



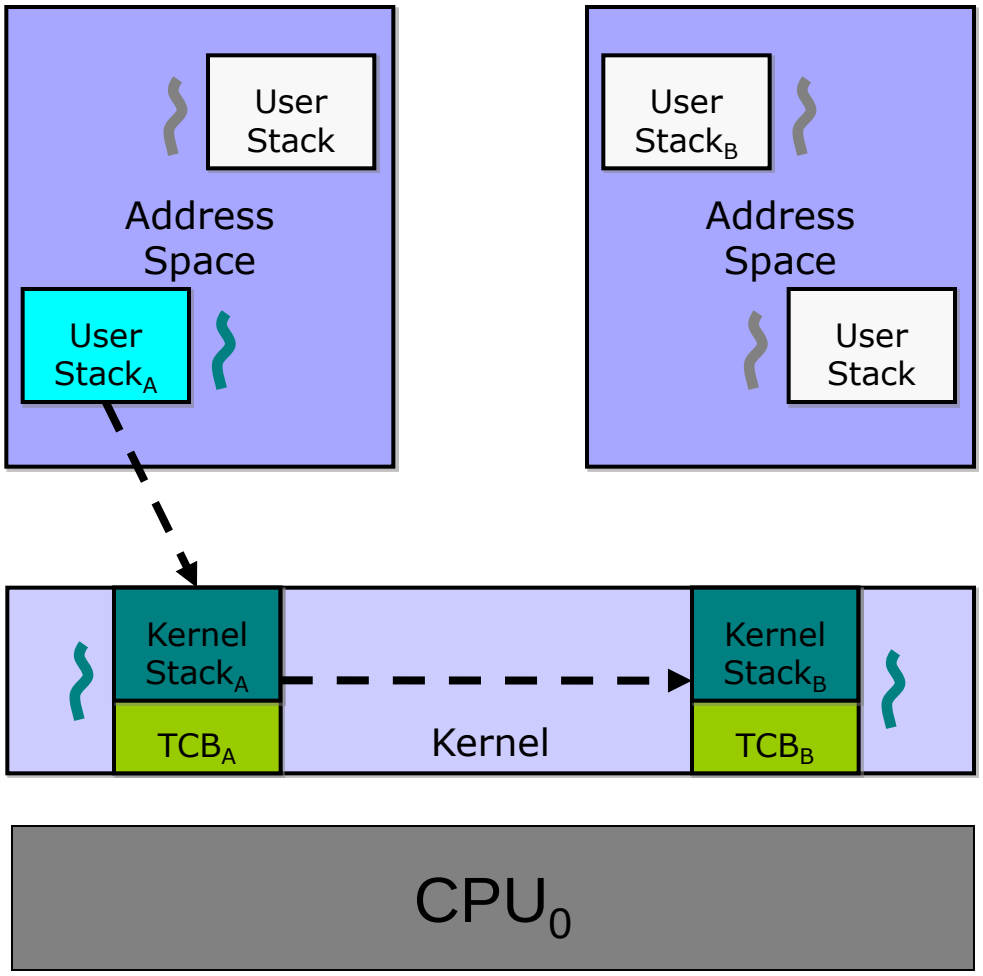
- User Part
  - Program code
  - Program data
  - User Stack

- Kernel Part
  - Kernel code
  - Kernel stack
  - TCB

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Control Flow



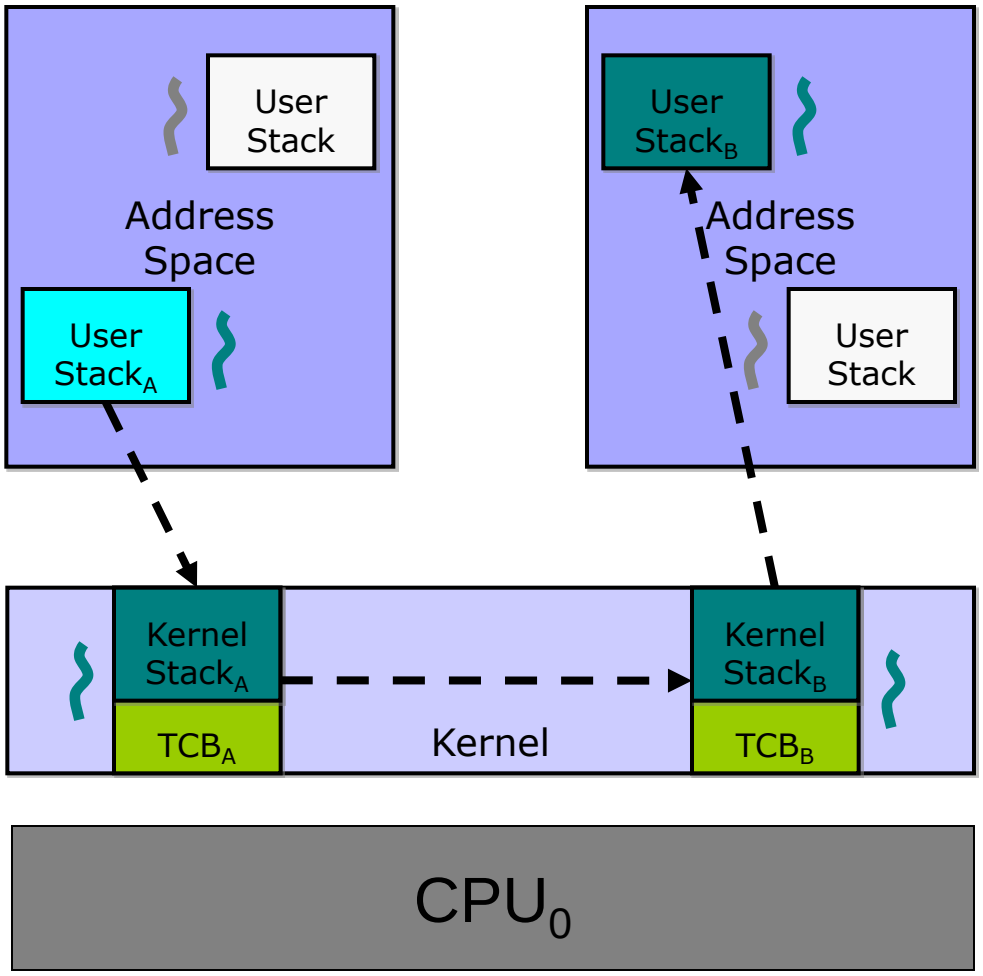
- User Part
  - Program code
  - Program data
  - User Stack

- Kernel Part
  - Kernel code
  - Kernel stack
  - TCB

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Control Flow



- User Part
  - Program code
  - Program data
  - User Stack

- Kernel Part
  - Kernel code
  - Kernel stack
  - TCB

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Observations

- Each thread is bound to one particular CPU at one point in time
- Only one thread per CPU is running at one point in time
- On an n-way SMP system n threads can thus run at once
- All other threads bound to a particular CPU are inactive (ready or blocked) inside the kernel meanwhile

Udo Steinberg

TU Dresden
Operating
Systems Group

Threads: Implementation Conclusions

- Thread state must be saved/restored on thread switch
- We need a thread control block (TCB) per thread
- TCBs must be kernel objects
  - actually some parts of the TCB can be safely exported to user space (UTCBs)
- TCBs implement threads

Udo Steinberg

TU Dresden
Operating
Systems Group

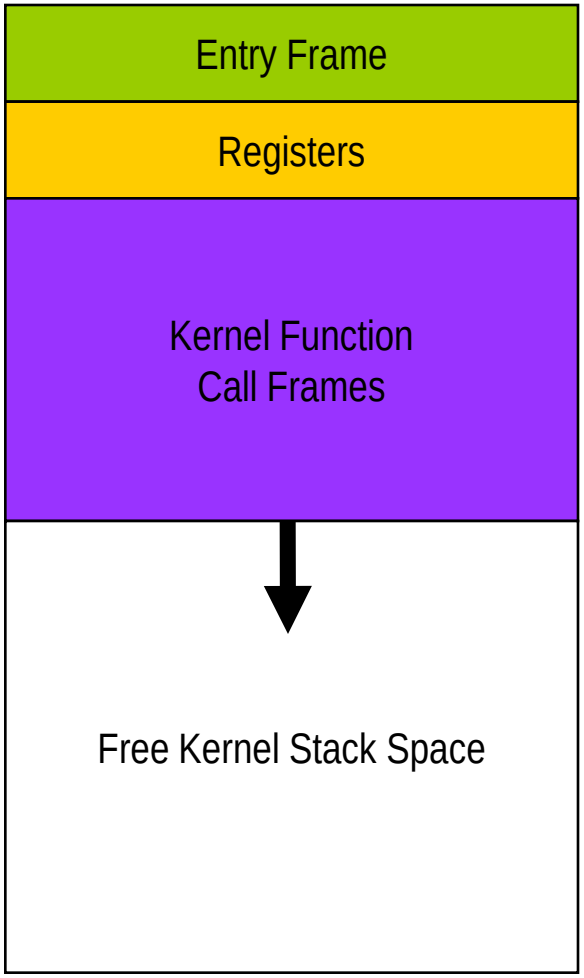
Thread Control Block (TCB)

- Thread State
- Kernel Stack Pointer
- Address Space Pointer
- Lock Count
- Thread Lock
- Scheduling Information
  - Scheduling Mode
  - Scheduling Context
  - Period Length
- FPU State
- Prev/Next List Pointers
- IPC Partner
- Sender List
- Pager
- Preempter
- Timeouts
- IPC Windows

Udo Steinberg

TU Dresden
Operating
Systems Group

Kernel Stack Layout: Timer Interrupt (x86)



- SS, ESP, EFLAGS, CS, EIP
- EAX, EDX, ECX, ESI, EDI
- function parameters
- function return address
- callee-saved registers
  - EBP, EDI, ESI, EBX
- local variables

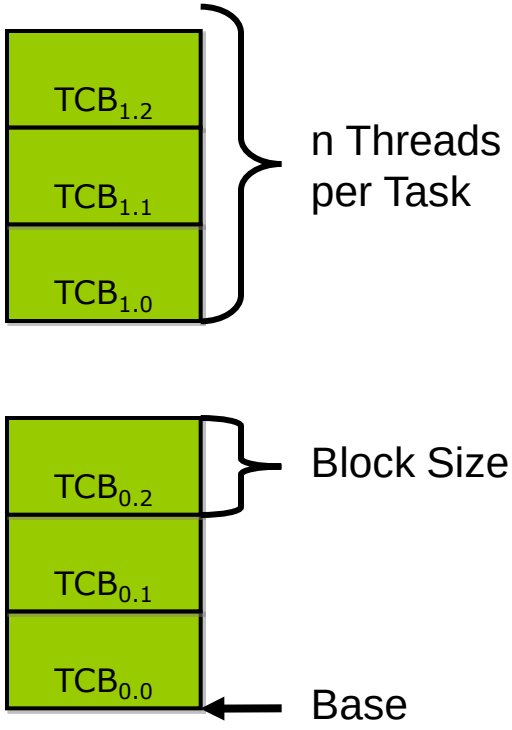
1. thread_timer_interrupt
2. Thread::handle_timer_interrupt
3. Context::schedule
4. Context::switch_to
5. Context::switch_exec
6. Context::switch_cpu

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread ID to TCB Translation

- Problem:
  - We have to find a thread's TCB from its thread ID efficiently
- Solution:
  - Align TCBs in kernel virtual memory



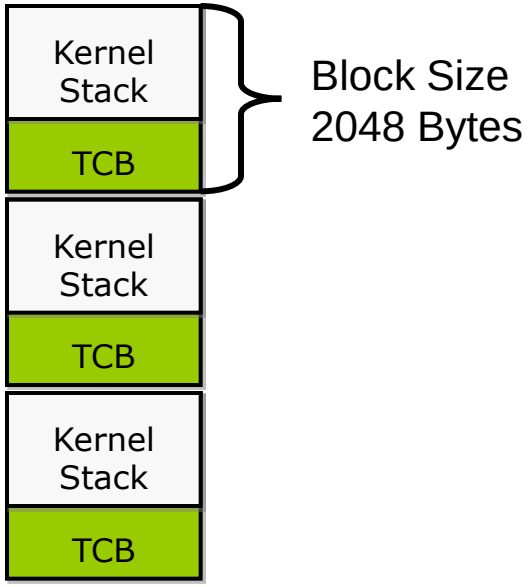

```
Context *t = base + (Task * n  + Thread) * Block Size;
```

Udo Steinberg

TU Dresden
Operating
Systems Group

Stack Pointer to TCB Translation

- Problem:
  - We have to find the currently executing thread's TCB efficiently
- Solution:
  - Align kernel stacks together with the TCBs and round down current stack pointer

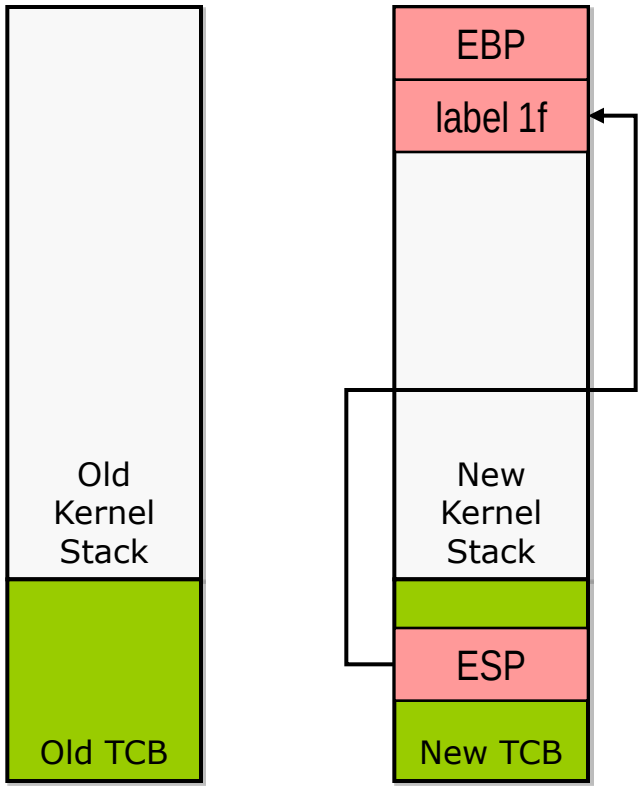



```
Context *current = sp & ~(Block Size – 1);
```

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Details (x86)

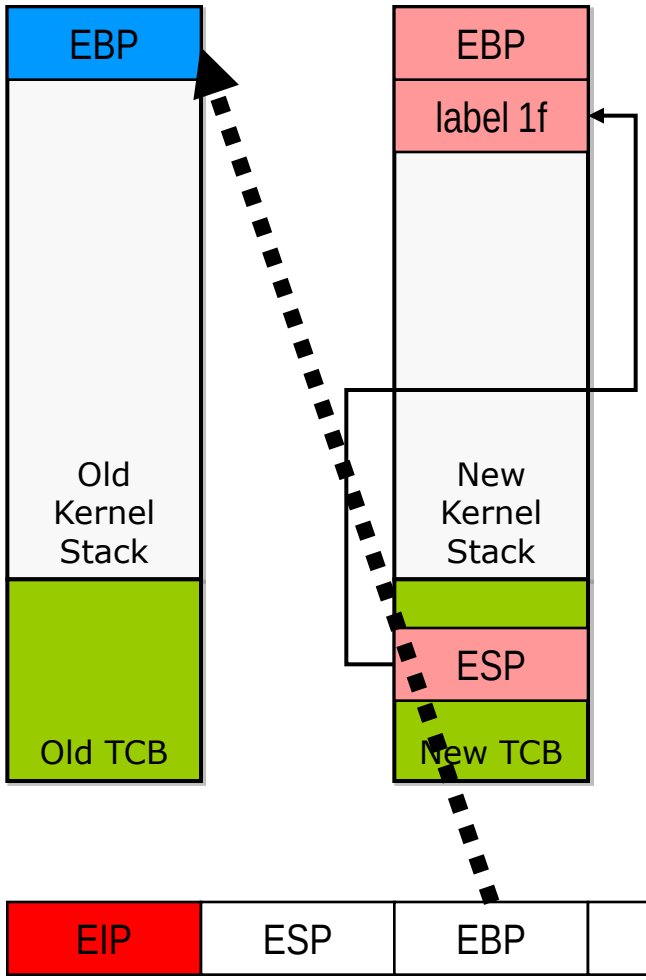



```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Details (x86)

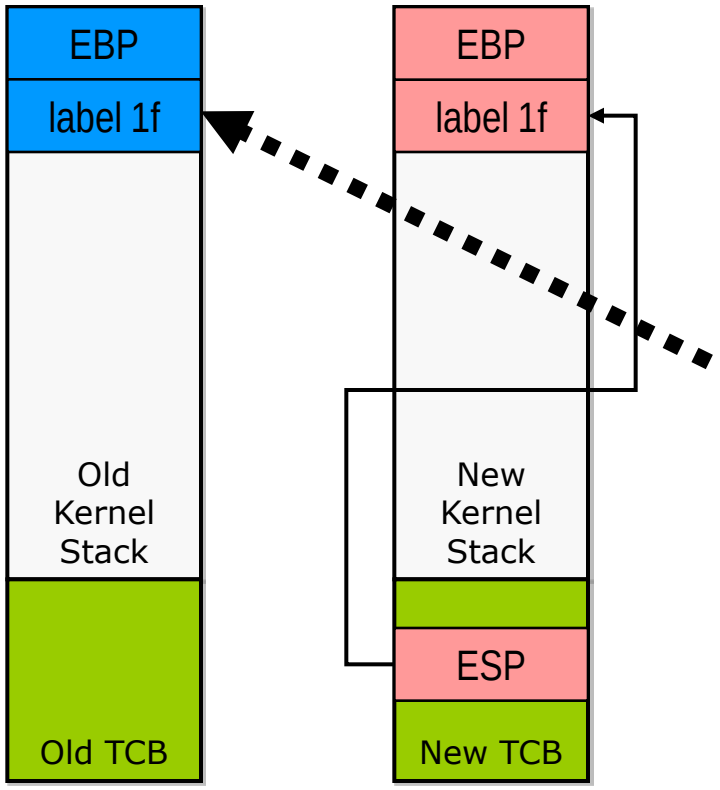



```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Details (x86)

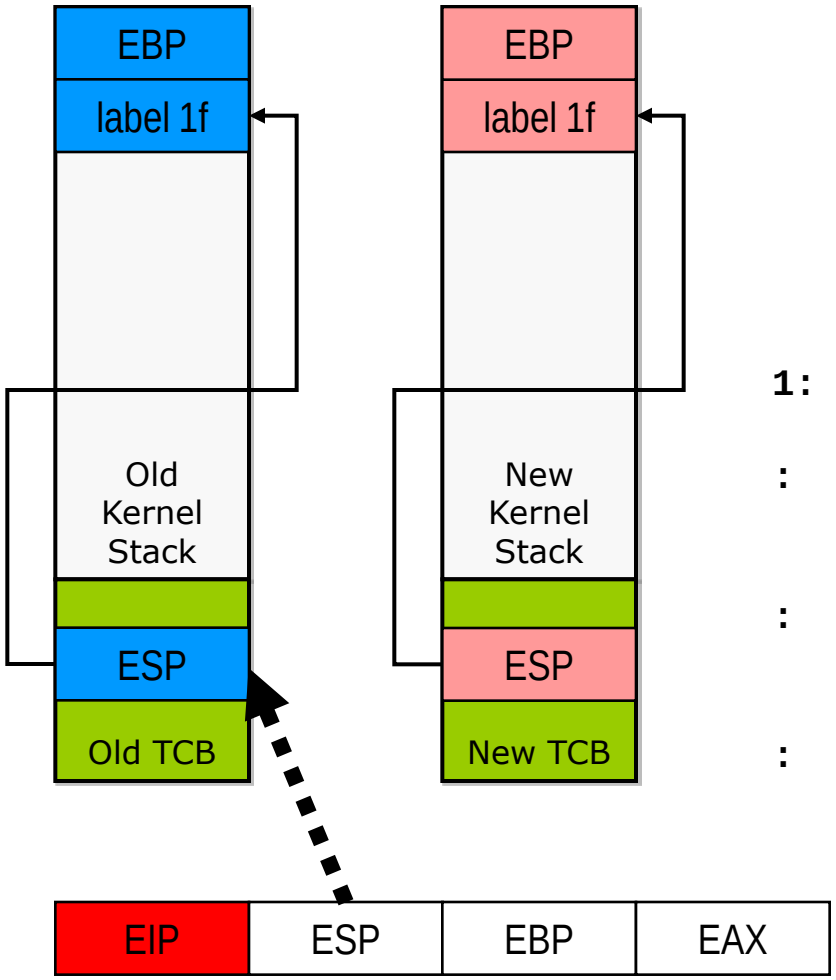



```
        pushl %%ebp
        pushl $1f
        movl  %%esp, (%0)
        movl  (%1), %%esp
        movl  %2, %%eax
        call  switchin_context_label
        popl  %%eax
        jmp   *%%eax
1:      popl %%ebp

:       "=c“ (dummy1),
        "=S" (dummy2),
        "=D" (dummy3)
:       "c" (&_kernel_sp),
        "S" (&t->_kernel_sp),
        "D" (t)
:       "eax",
        "ebx",
        "edx",
        "memory"
```

Udo Steinberg

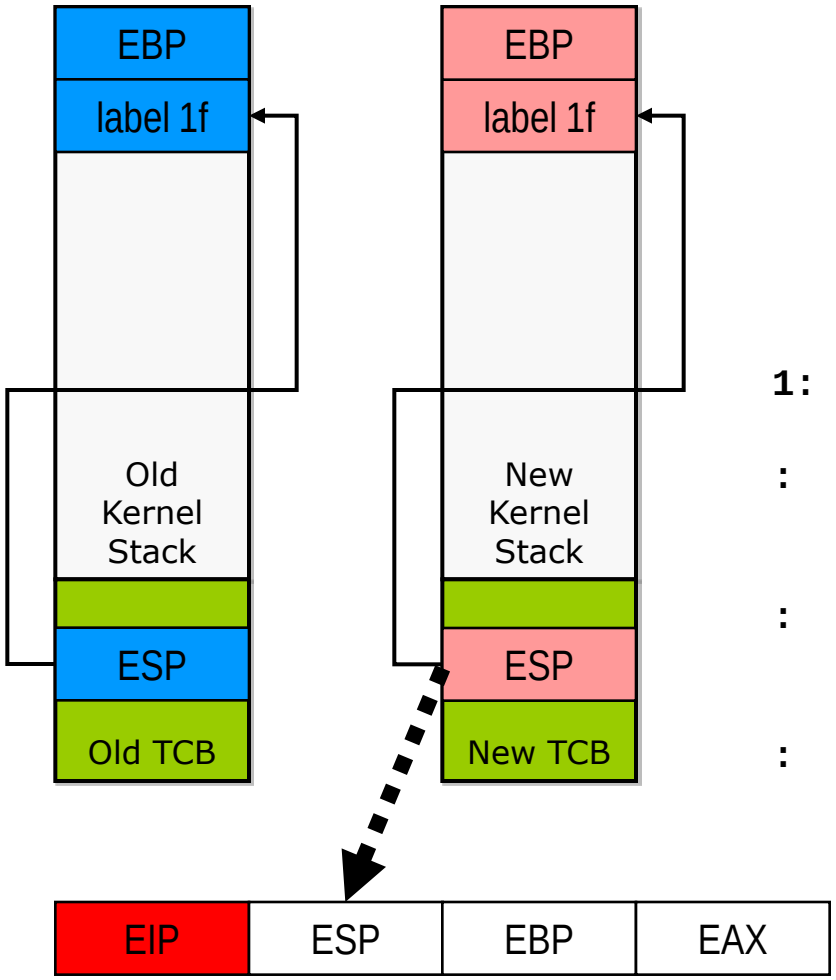
TU Dresden
Operating
Systems Group

Thread Switch: Details (x86)




```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```

Thread Switch: Details (x86)

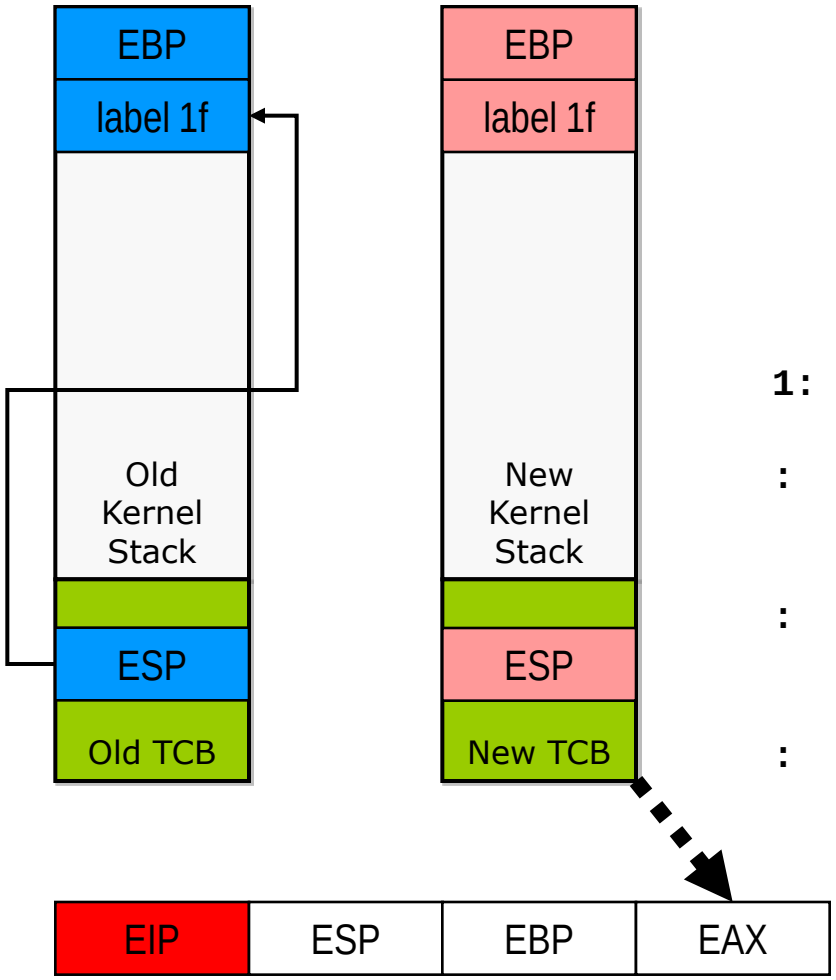



```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Details (x86)

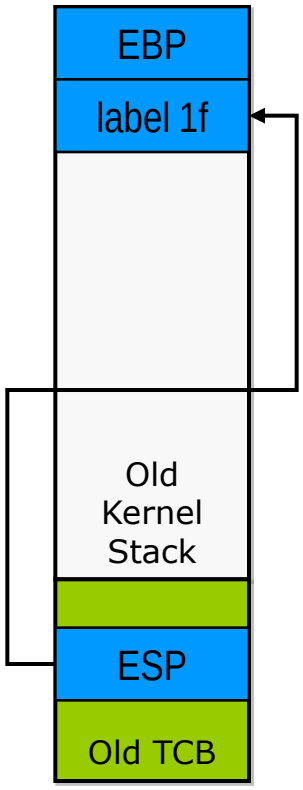


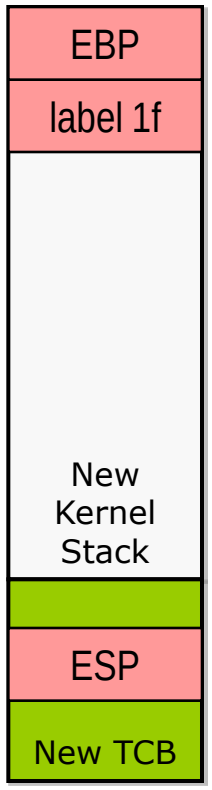

```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Details (x86)



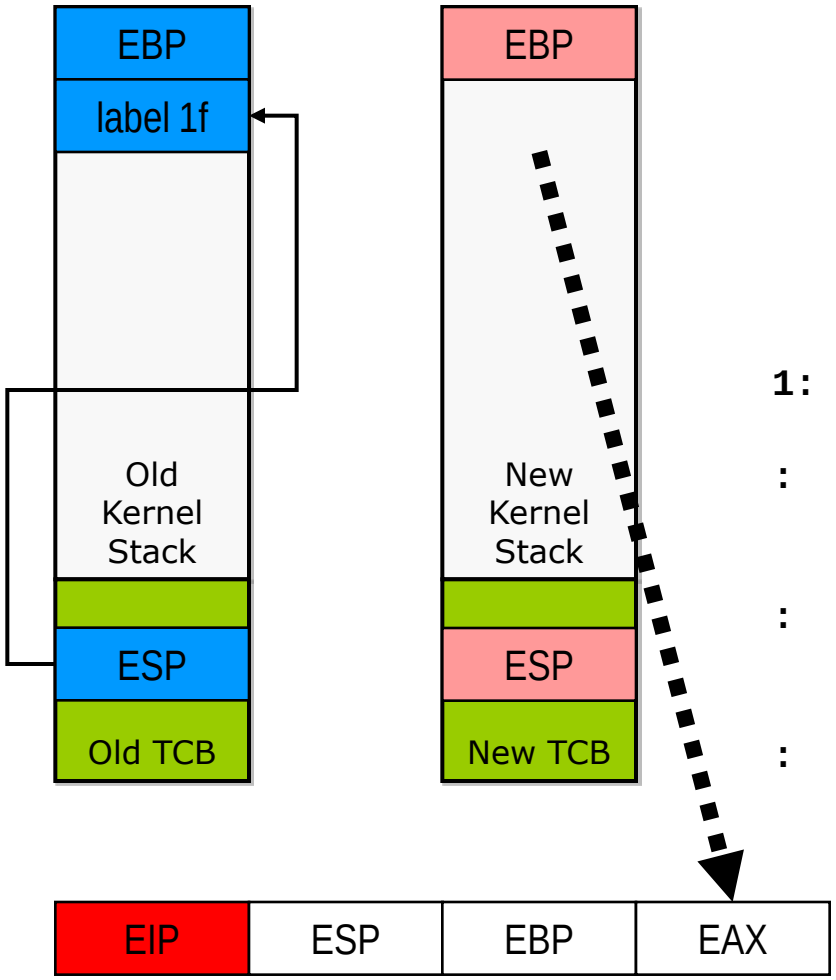



```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```

Udo Steinberg

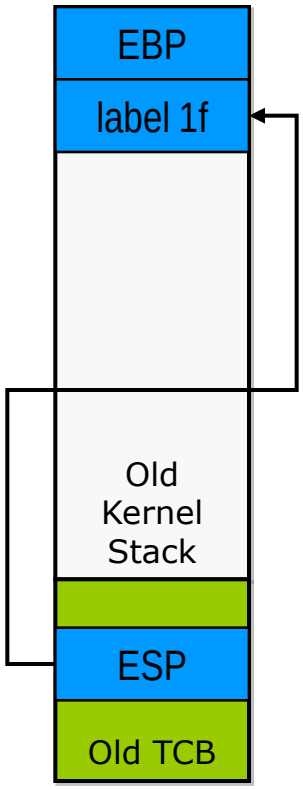
TU Dresden
Operating
Systems Group

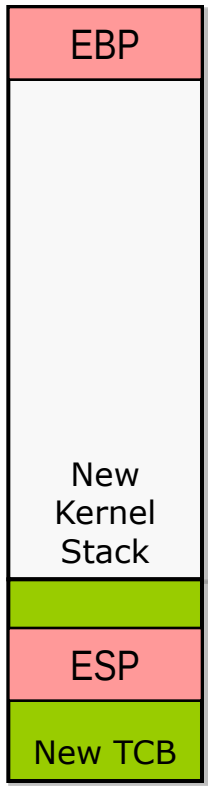
Thread Switch: Details (x86)



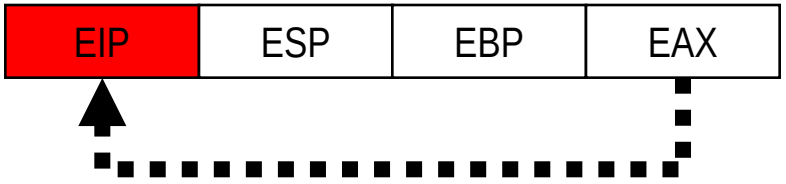

```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```

Thread Switch: Details (x86)





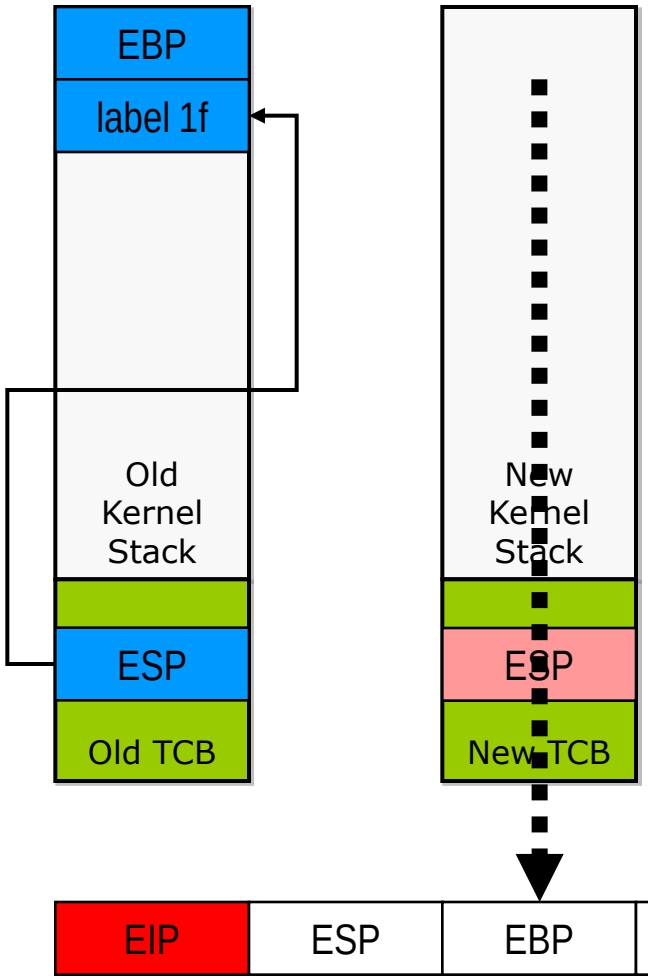

```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```



Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Details (x86)




```
    pushl %%ebp
    pushl $1f
    movl  %%esp, (%0)
    movl  (%1), %%esp
    movl  %2, %%eax
    call  switchin_context_label
    popl  %%eax
    jmp   *%%eax
1:  popl %%ebp

:   "=c“ (dummy1),
    "=S" (dummy2),
    "=D" (dummy3)
:   "c" (&_kernel_sp),
    "S" (&t->_kernel_sp),
    "D" (t)
:   "eax",
    "ebx",
    "edx",
    "memory"
```

Udo Steinberg

TU Dresden
Operating
Systems Group

Thread Switch: Register Save/Restore (x86)

- Code explicitly saved register EBP on kernel stack
- What about the other registers?
  - EAX, EBX, EDX explicitly listed in clobber list
  - ECX, ESI, EDI implicitly clobbered by dummy output parameters
  - Clobbered registers:
    - saved by compiler in function prologue
    - restored by compiler in function epilogue
- Memory clobber acts as compiler barrier
  - Prohibits reordering of instructions (optimizations)
  - Prohibits caching of memory variables in registers

Udo Steinberg

TU Dresden
Operating
Systems Group

Kernel Stack for Kernel Entry

- How does the processor know which kernel stack to use when a thread enters the kernel?
- Information stored in Task State Segment (TSS)
  - contains one stack pointer for each privilege level (0,1,2)
  - tss->esp0 (stack pointer for ring 0)
- Kernel Stack Pointer in TSS updated on each thread switch by function Context::switchin_context

Udo Steinberg

TU Dresden
Operating
Systems Group

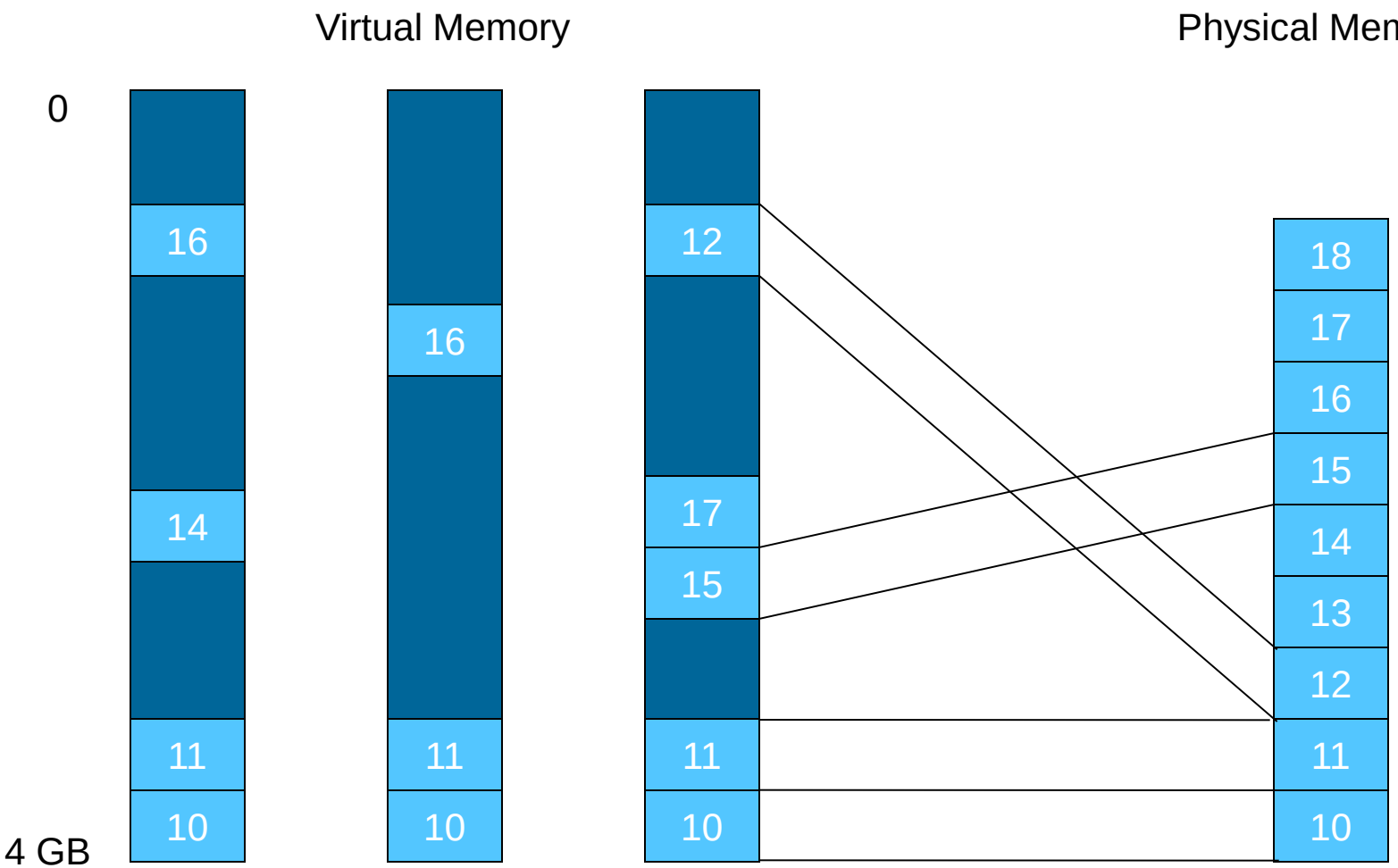
L4 Microkernel Abstractions

- Threads
- Address Spaces
  - What is an address space?
  - How does an address space provide protection and isolation?
- Inter-Process Communication

Udo Steinberg

TU Dresden
Operating
Systems Group

Virtual Memory



Udo Steinberg

TU Dresden
Operating
Systems Group

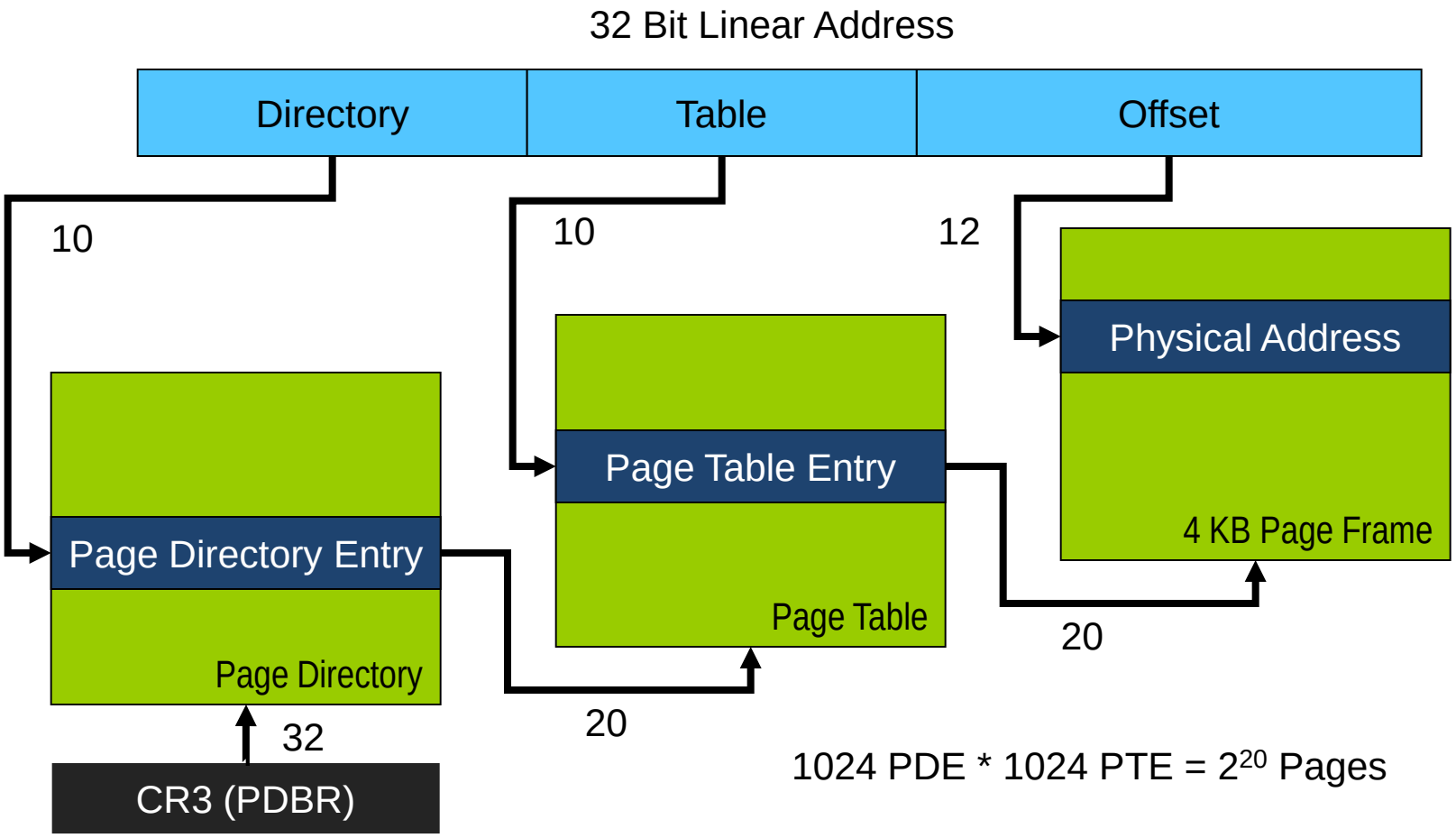
Paging

- Translation of linear to physical addresses by the Memory Management Unit (MMU)
- Implemented by processor data structures:
  - Page Directory Base Register (CR3)
  - Page Directory (PDIR)
    - an array of up to 1024 page-directory entries (PDEs), each 32bit wide, contained in a 4KB page
  - Page Table (PTAB)
    - an array of up to 1024 page-table entries (PTEs), each 32bit wide, contained in a 4KB page
- Paging data structures use physical addresses

Udo Steinberg

TU Dresden
Operating
Systems Group

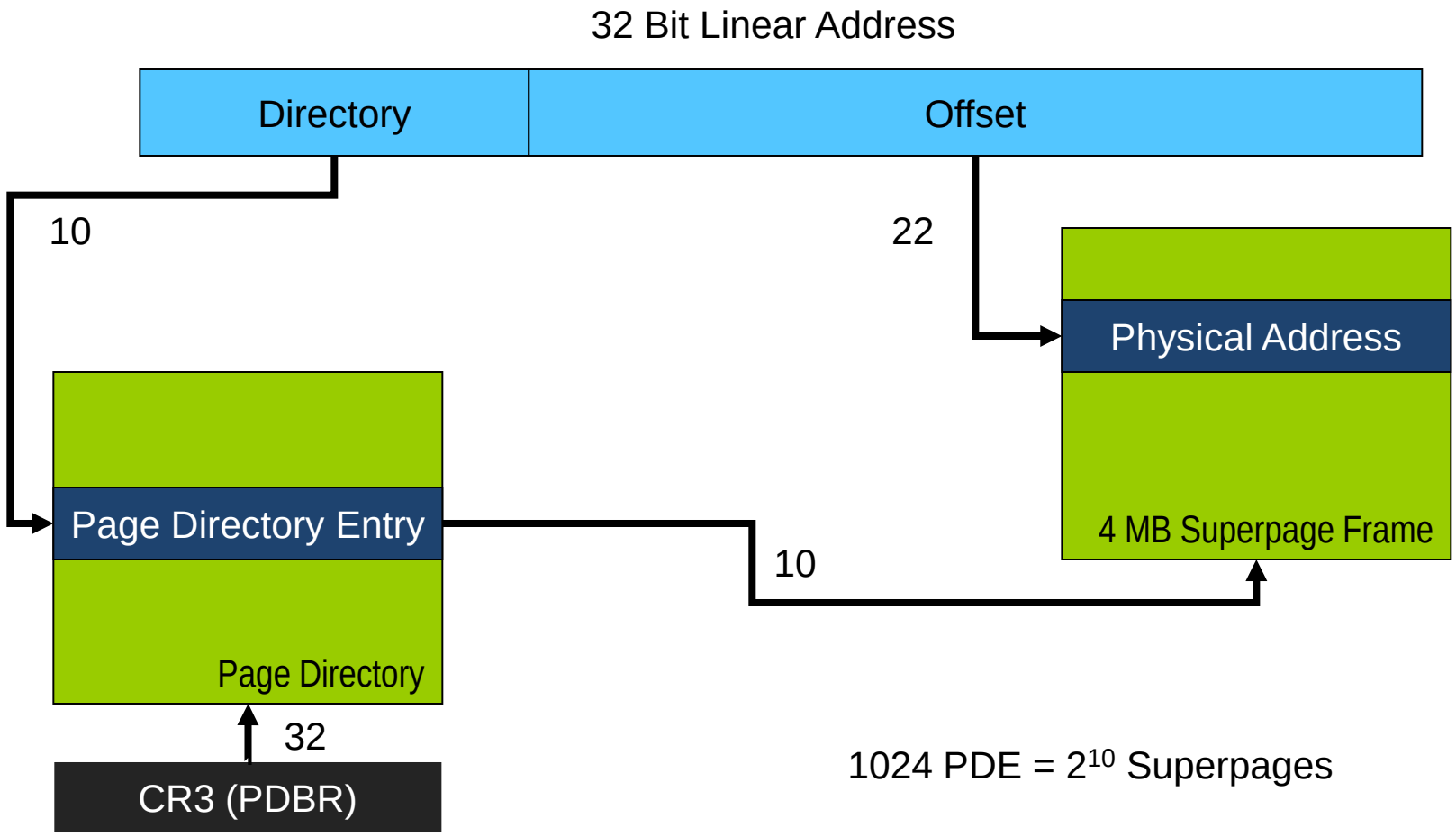
Address Translation: 4KB-Pages (x86)



Udo Steinberg

TU Dresden
Operating
Systems Group

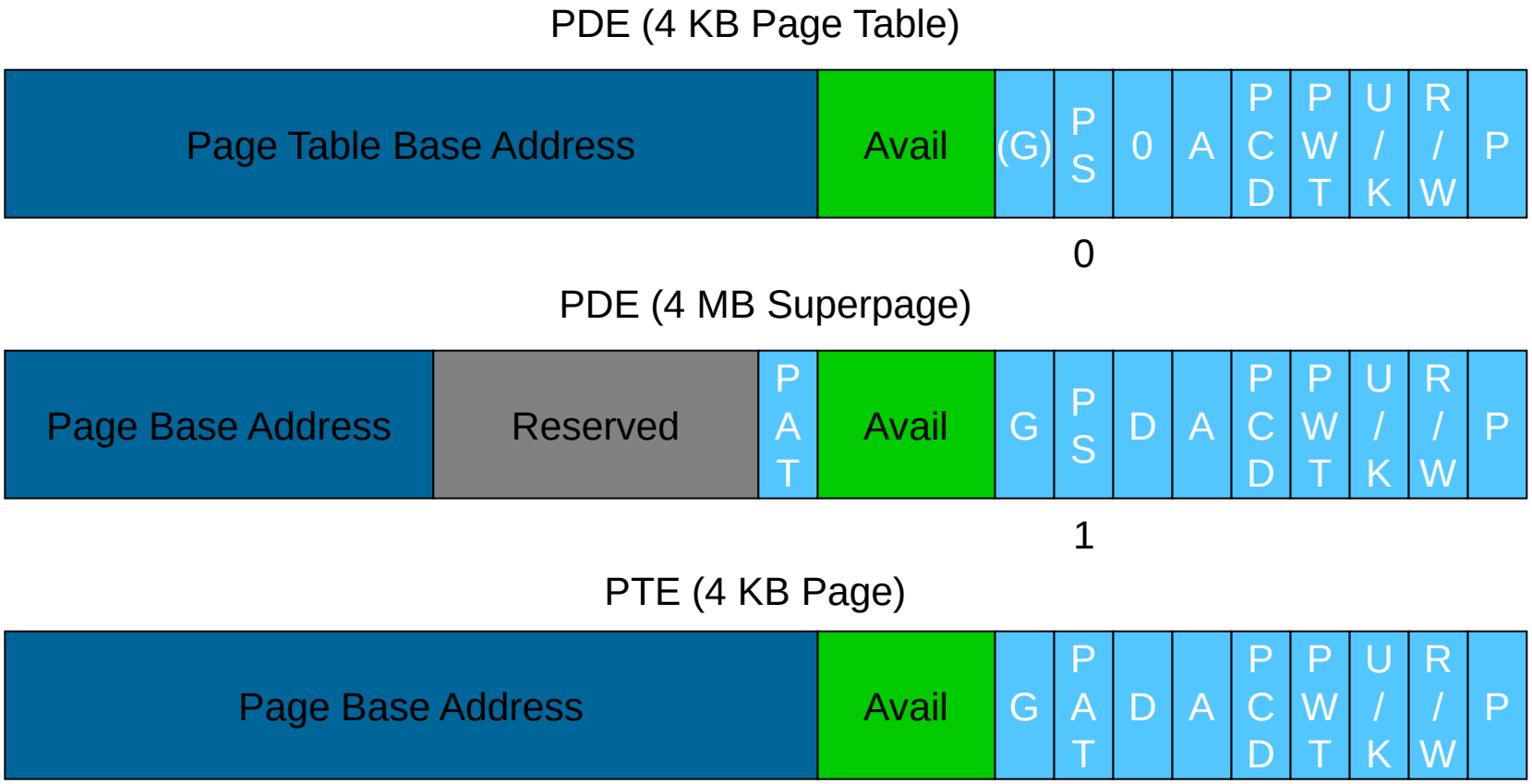
Address Translation: 4MB-Superpages (x86)



Udo Steinberg

TU Dresden
Operating
Systems Group

Page-Directory and Page-Table Entries (x86)



Udo Steinberg

TU Dresden
Operating
Systems Group

Page-Table Management

- Implemented in C++ class „Mem_space“
- v_insert
  - Insert a memory mapping from page tables
  - Upgrade attributes on a memory mapping
- v_delete
  - Remove a memory mapping from page tables
  - Downgrade attributes on a memory mapping
- v_lookup
  - Look up a mapping by linear address
  - Returns physical address, page size, page attributes

Udo Steinberg

TU Dresden
Operating
Systems Group

Address-Space Switch

- Switching to a thread in a different address space requires an address-space switch
- Address-Space Switch happens by loading a new page-directory address into the PDBR (CR3)
- CR3 reload implicitly causes a TLB flush

Udo Steinberg

TU Dresden
Operating
Systems Group

Translation Lookaside Buffer (TLB)

- Caches recent linear-to-physical address translations and page protection bits for rapid access
- Avoids expensive page-table walk
- Must be kept consistent with the page-table hierarchy by the OS (no TLB coherency protocol)
- When modifying a page table, OS must flush relevant TLB entries

Udo Steinberg

TU Dresden
Operating
Systems Group

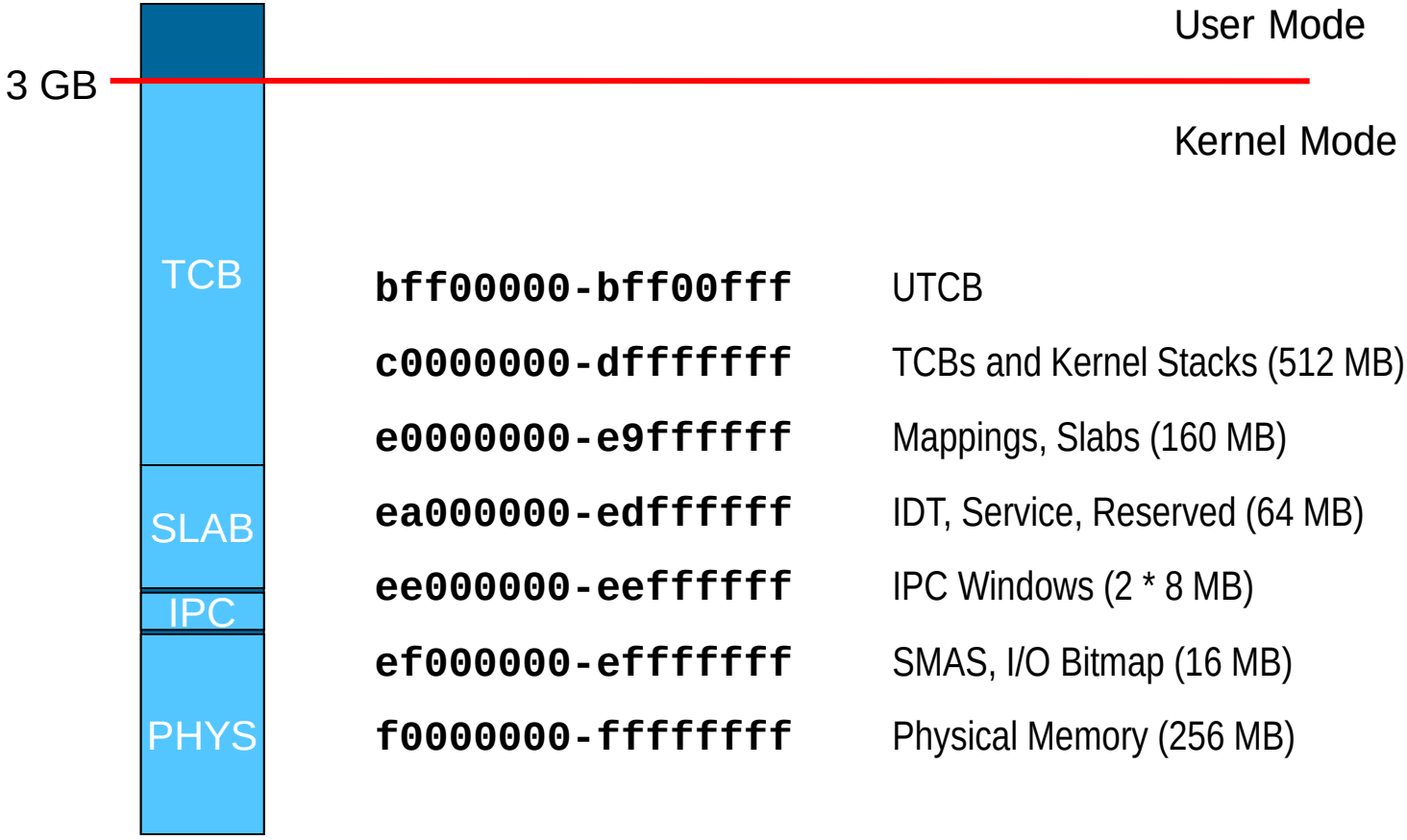
TLB Flush / TLB Shootdown

- TLB flush triggered by CR3 reload or INVLPG
- No TLB flush required when upgrading page attributes
- CR3 reload does not flush pages with global bit set
- Changes to page-tables active on other CPUs must be explicitly invalidated via TLB shootdown
  - Expensive signaling and synchronization
  - Inter-Processor-Interrupt (IPI)

Udo Steinberg

TU Dresden
Operating
Systems Group

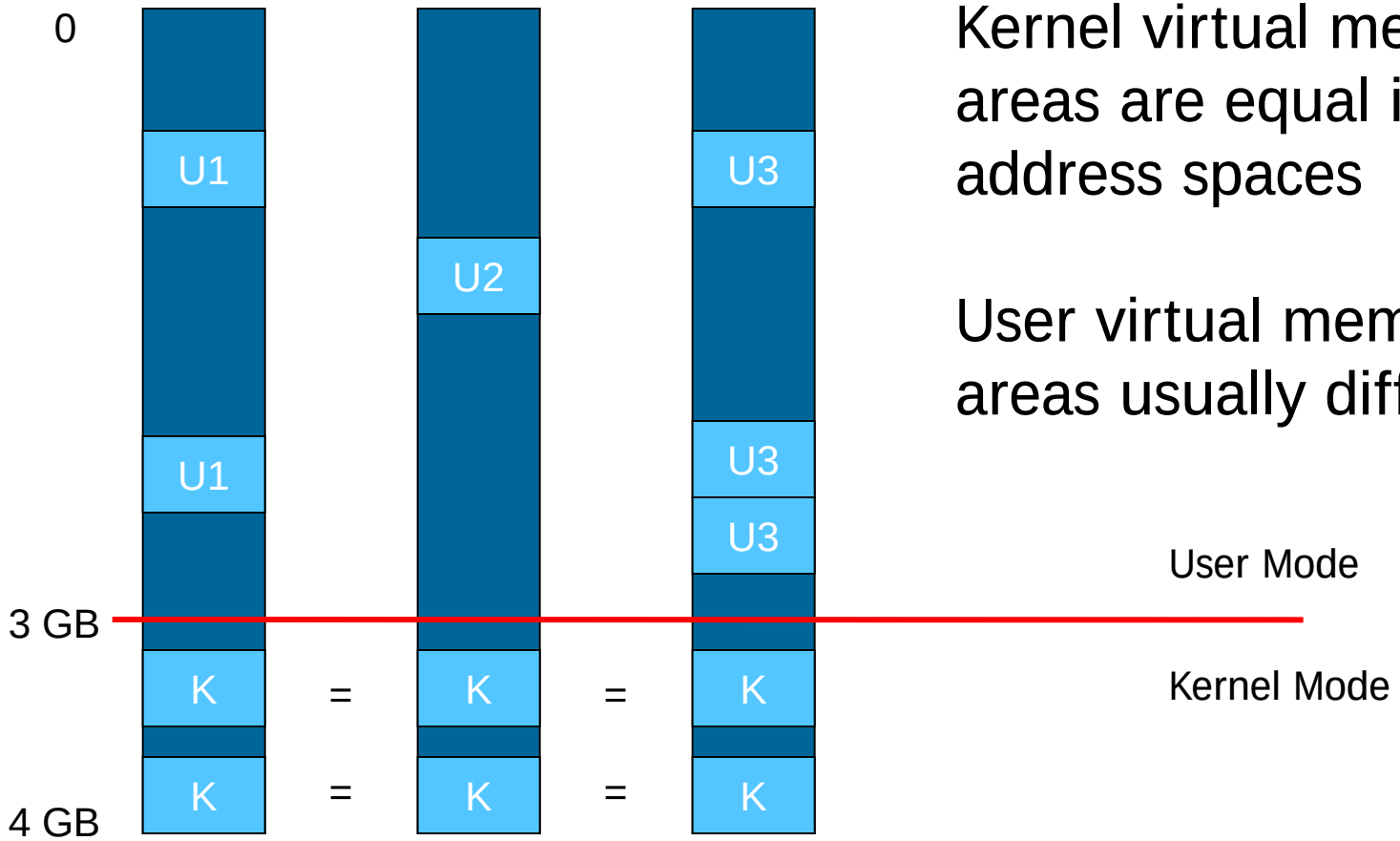
Kernel Virtual-Memory Layout



Udo Steinberg

TU Dresden
Operating
Systems Group

Kernel-Memory Synchronization



Kernel virtual memory areas are equal in all address spaces

User virtual memory areas usually differ

Udo Steinberg

TU Dresden
Operating
Systems Group

Kernel-Memory Synchronisation

- Kernel Memory Region must be kept the same throughout all address spaces
- We can share all 256 page tables or superpages covering the kernel region between all address spaces
- Kernel makes additions in master page directory and copies PDEs on demand to other address spaces
  - i.e., whenever a page-fault in a kernel region occurs

Udo Steinberg

TU Dresden
Operating
Systems Group

FPU Switch

- Floating-Point Unit (FPU) executes in parallel to CPU
- OS wants to give each thread the notion that it can use the FPU exclusively (similar to CPU)
- Naive approach:
  - „Eager“ FPU context switching
  - Save/restore FPU context on each thread switch
  - But: FPU context is up to 512 bytes large (SSE)
  - Performance killer!

Udo Steinberg

TU Dresden
Operating
Systems Group

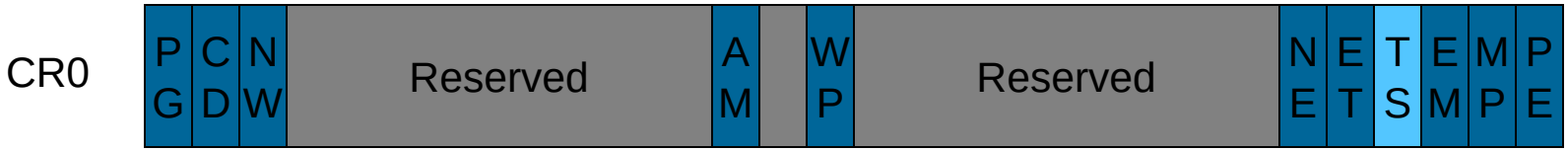
Lazy FPU Switch: General Idea

- We want to know if/when a thread uses the FPU
- We only want to save the FPU state when it has been modified
- We don‘t want to save the FPU state when switching from a thread that used the FPU to a thread that is not going to use the FPU and then later restore the old (unmodified) FPU state

Udo Steinberg

TU Dresden
Operating
Systems Group

Lazy FPU Switch: Detecting FPU Access



- If CR0.TS (Task Switched) flag is set, execution of any FPU instruction will generate a #NM exception prior to the execution of the FPU instruction
- If CR0.TS is clear, no #NM exception will be generated on FPU access

Udo Steinberg

TU Dresden
Operating
Systems Group

Lazy FPU Switch: Requirements

- We need:
  - A pointer to the thread which most recently used the FPU, i.e., whose state is currently in the FPU
    - current FPU owner
  - An FPU context save area for each thread using the FPU
    - allocated from an in-kernel slab allocator
  - A fast way to detect if a thread is currently the FPU owner
    - implemented by Thread_fpu_owner state flag

Udo Steinberg

TU Dresden
Operating
Systems Group

Lazy FPU Switch: Implementation

- Bootup: Set CR0.TS (FPU marked busy)
- Thread accessing the FPU will generate #NM
  - #NM handler saves FPU state of previous FPU owner
  - #NM handler restores/inits FPU state of current thread
  - Current thread becomes new FPU owner
  - Clear CR0.TS (FPU accesses no longer cause #NM)
- CR0.TS update during thread switch:
  - Switching away from FPU owner – set CR0.TS
  - Switching to FPU owner – clear CR0.TS

Udo Steinberg

TU Dresden
Operating
Systems Group