Karlsruhe Institute of Technology

Microkernel Construction
I.12 – Review

Lecture Summer Term 2017
Wednesday 15:45-17:15 R 131, 50.34 (INFO)

Jens Kehne | Marius Hillenbrand
Operating Systems Group, Department of Computer Science

Threading

- 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

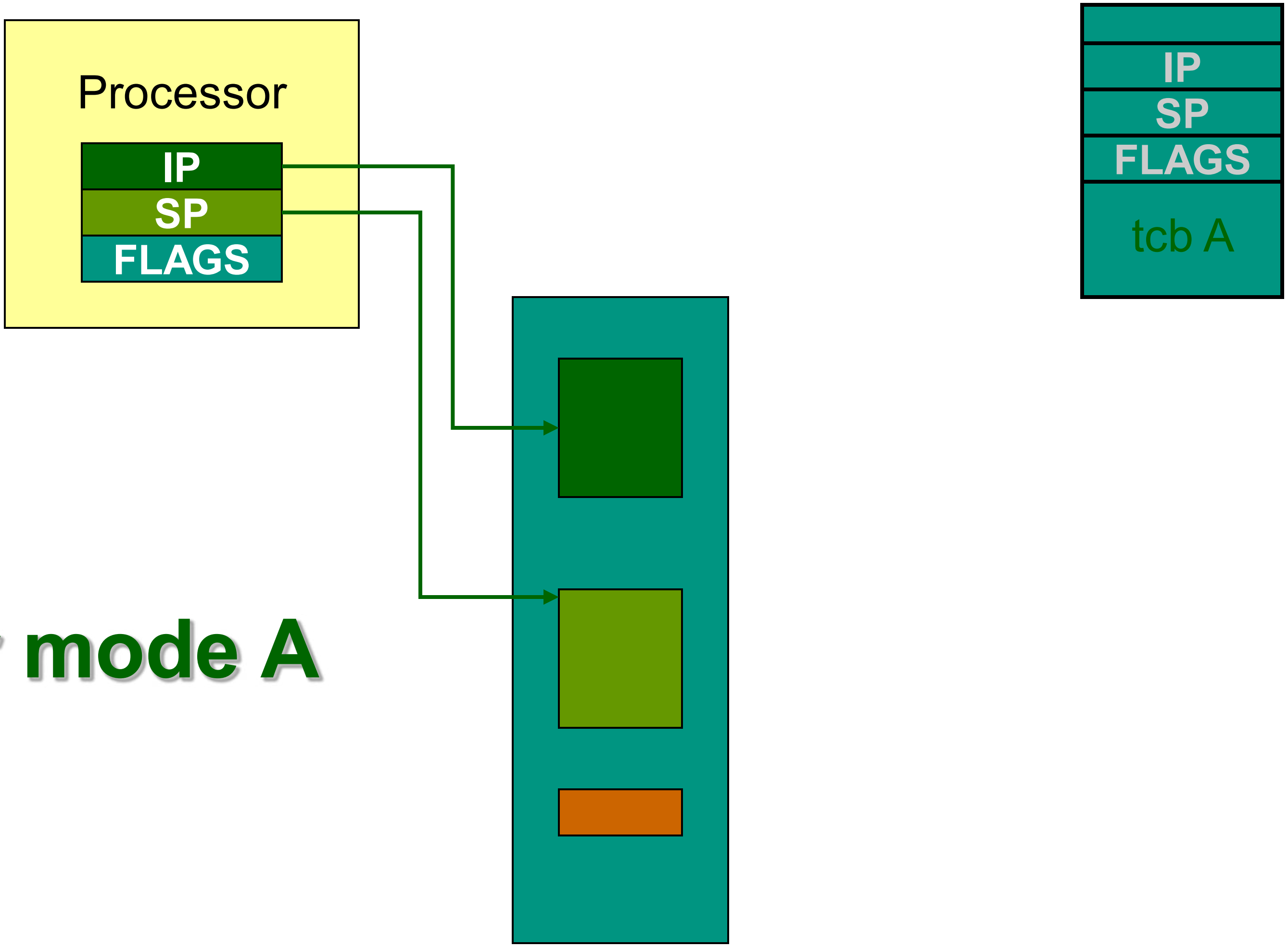
At least partially. We have found some good reasons to implement parts of the TCB in user memory.

- We often need to find
  - Any thread's TCB using its global ID
  - The currently executing thread's TCB (per processor)

Thread Switch A → B

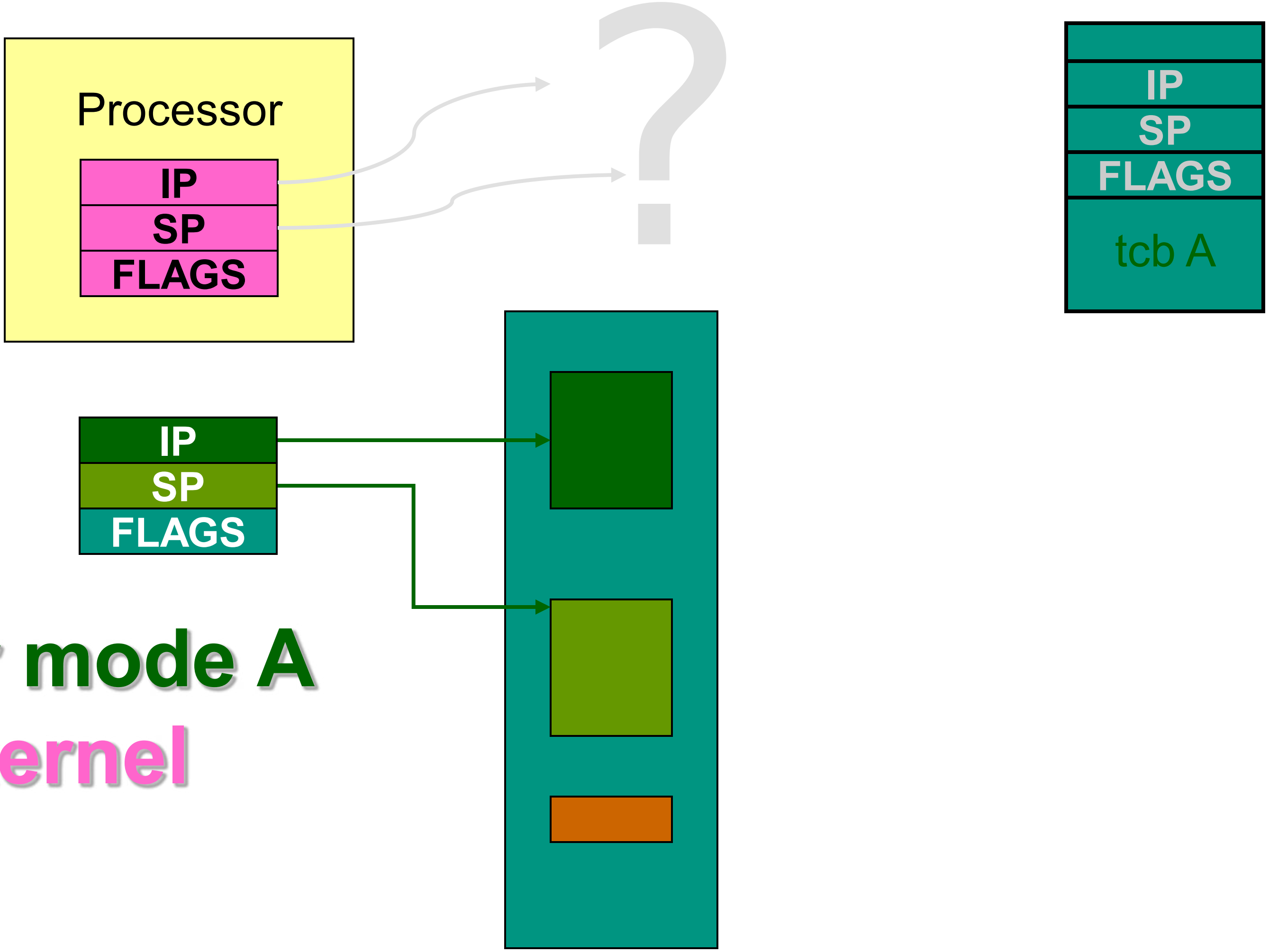
- Thread A is running in user mode
- Thread A experiences an end-of-time-slice or is preempted by a (device) interrupt
- We enter kernel mode
- The microkernel saves the status of thread A on A’s TCB
- The microkernel loads the status of thread B from B’s TCB
- We leave kernel mode
- Thread B is running in user mode

Thread Switch A → kernel → B



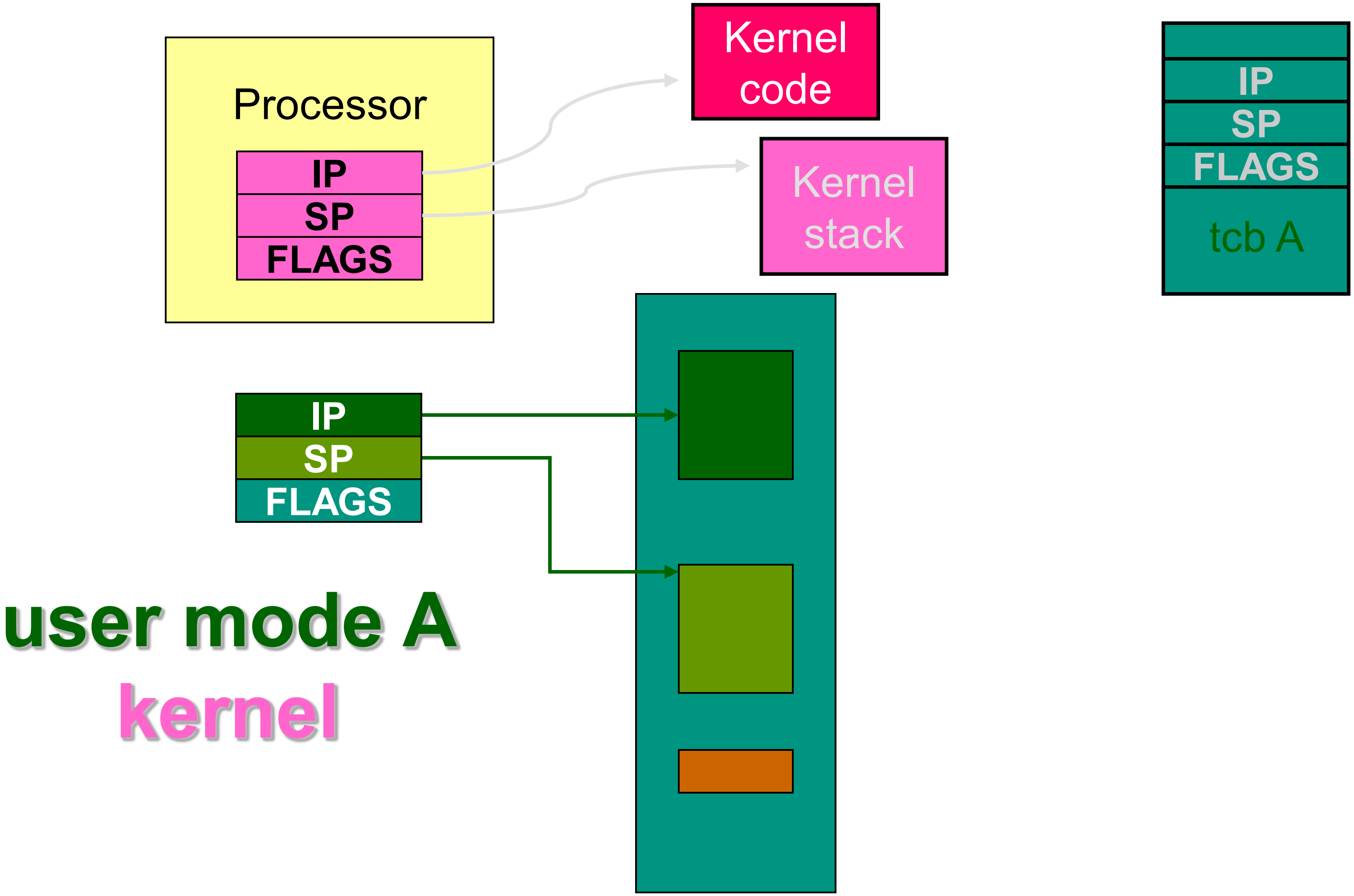
user mode A

Thread Switch A → kernel → B

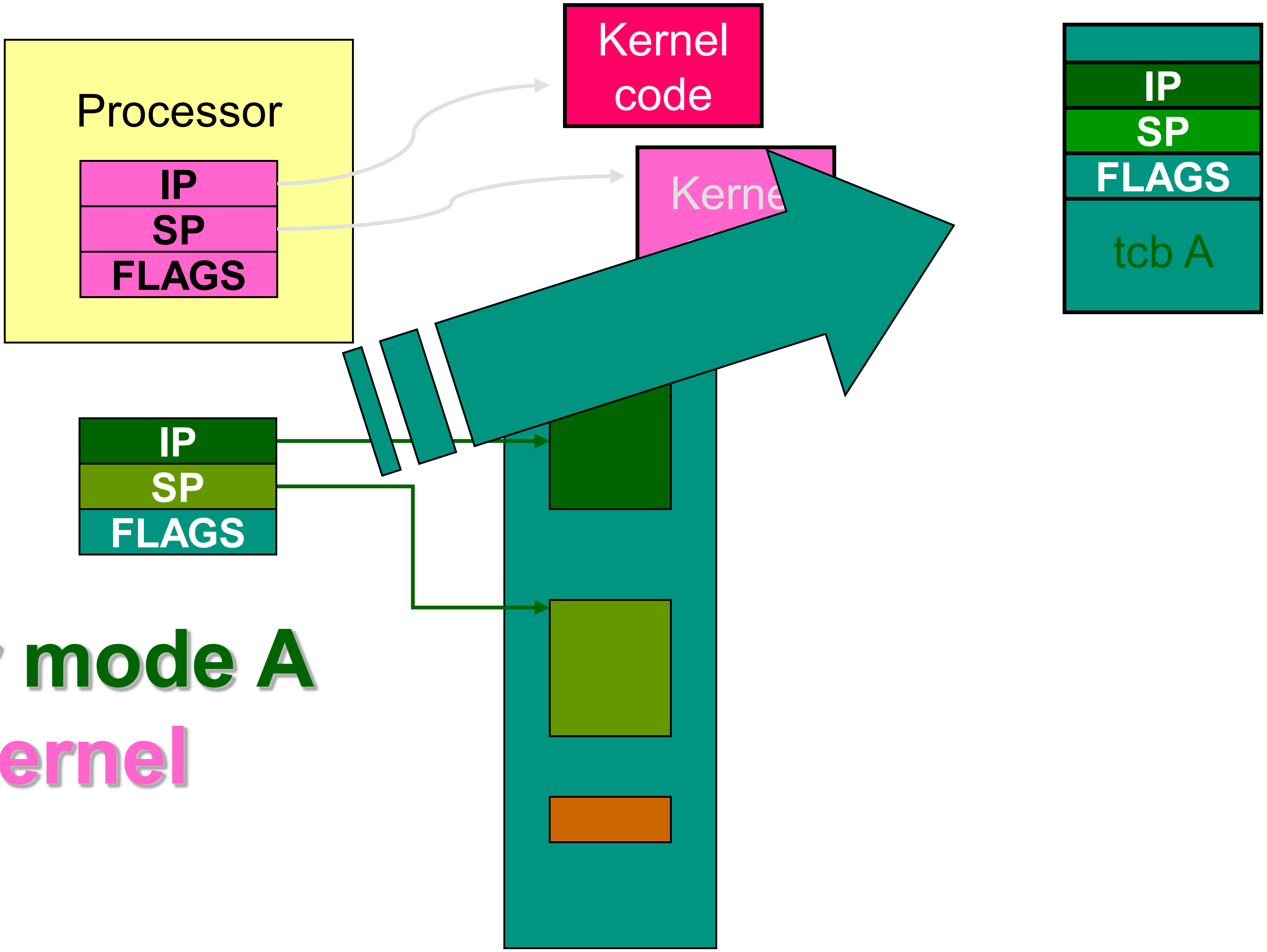


user mode A
kernel

Thread Switch A → kernel → B



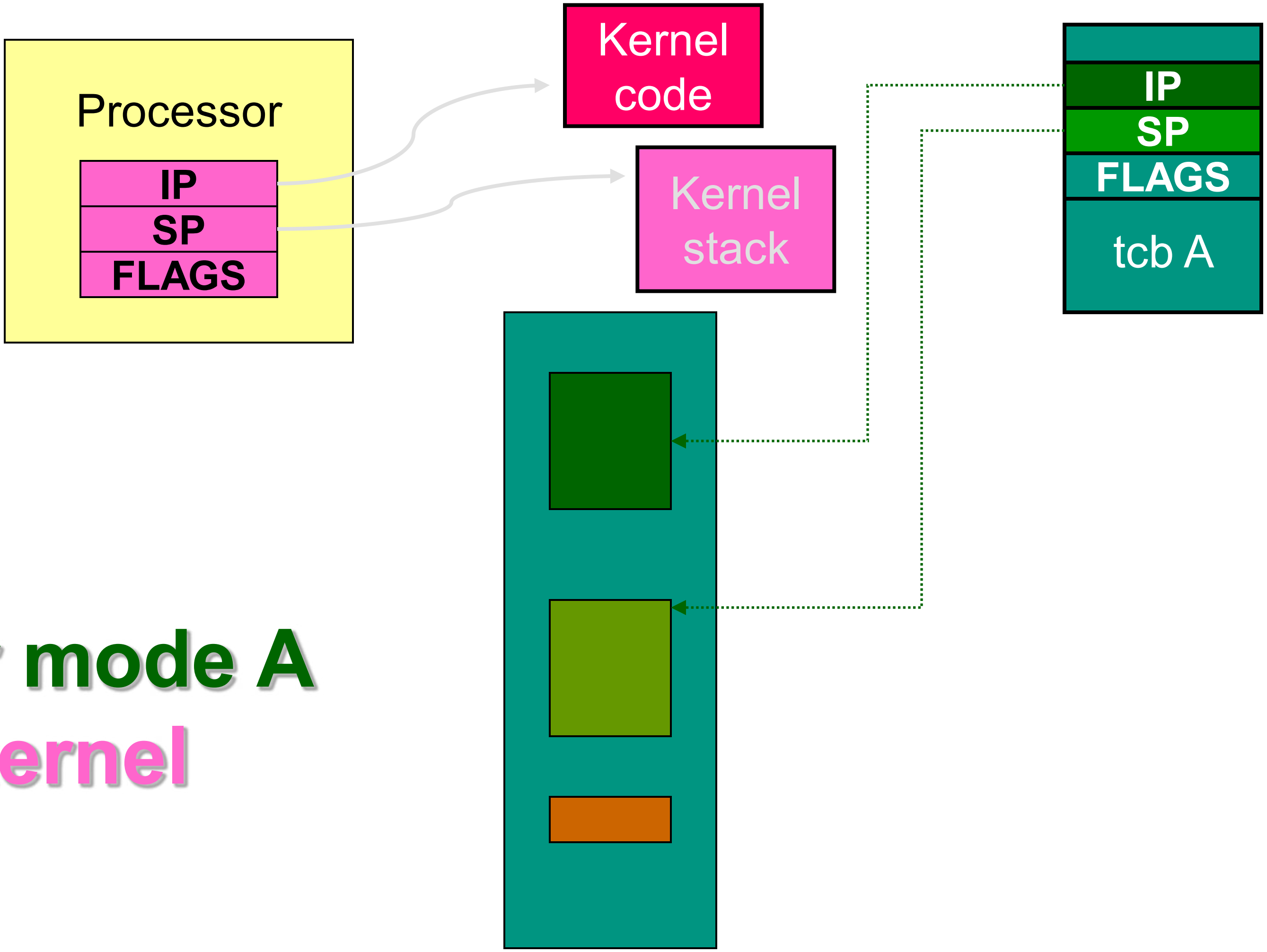
Thread Switch A → kernel → B



user mode A

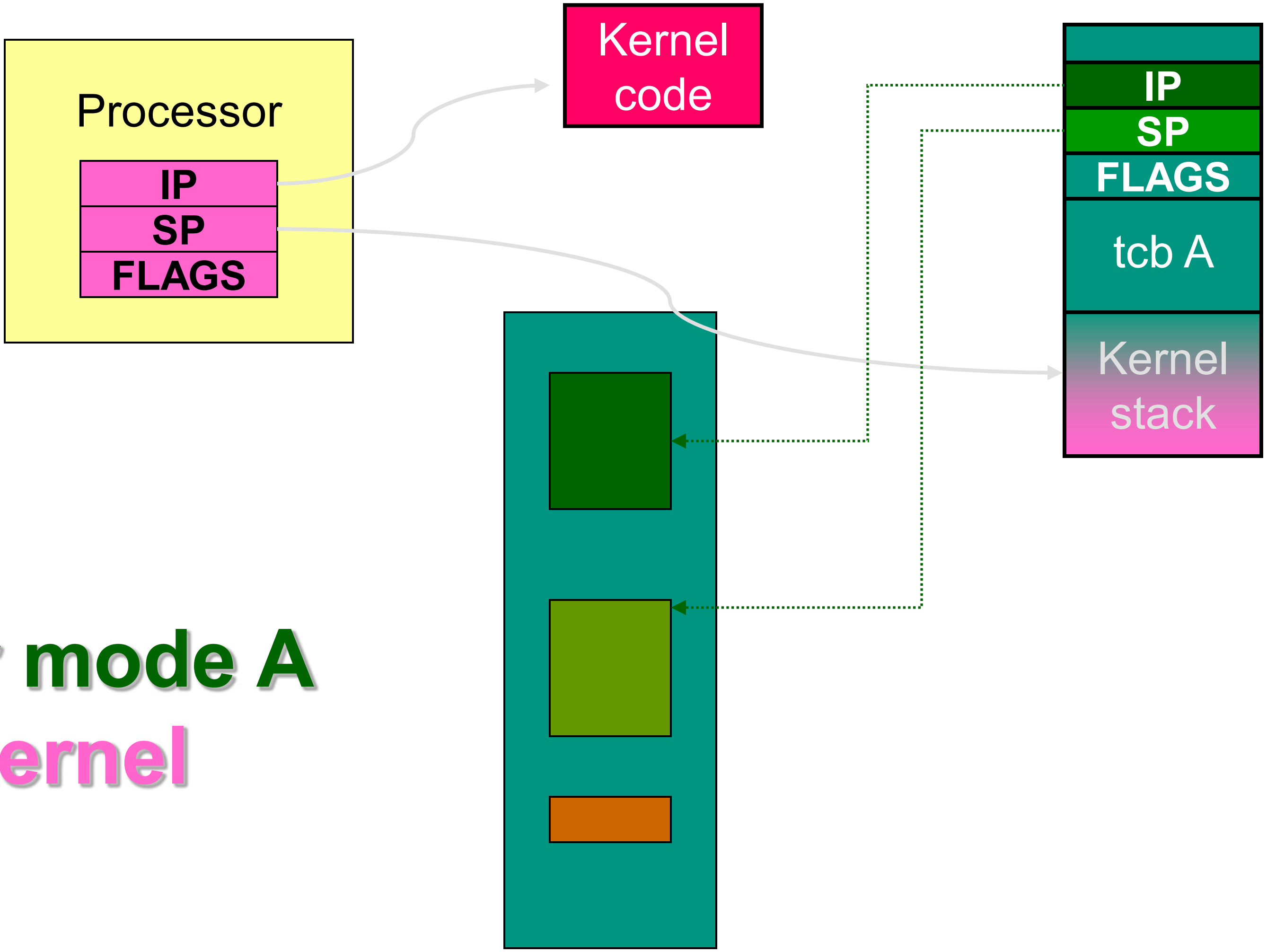
kernel

Thread Switch A → kernel → B



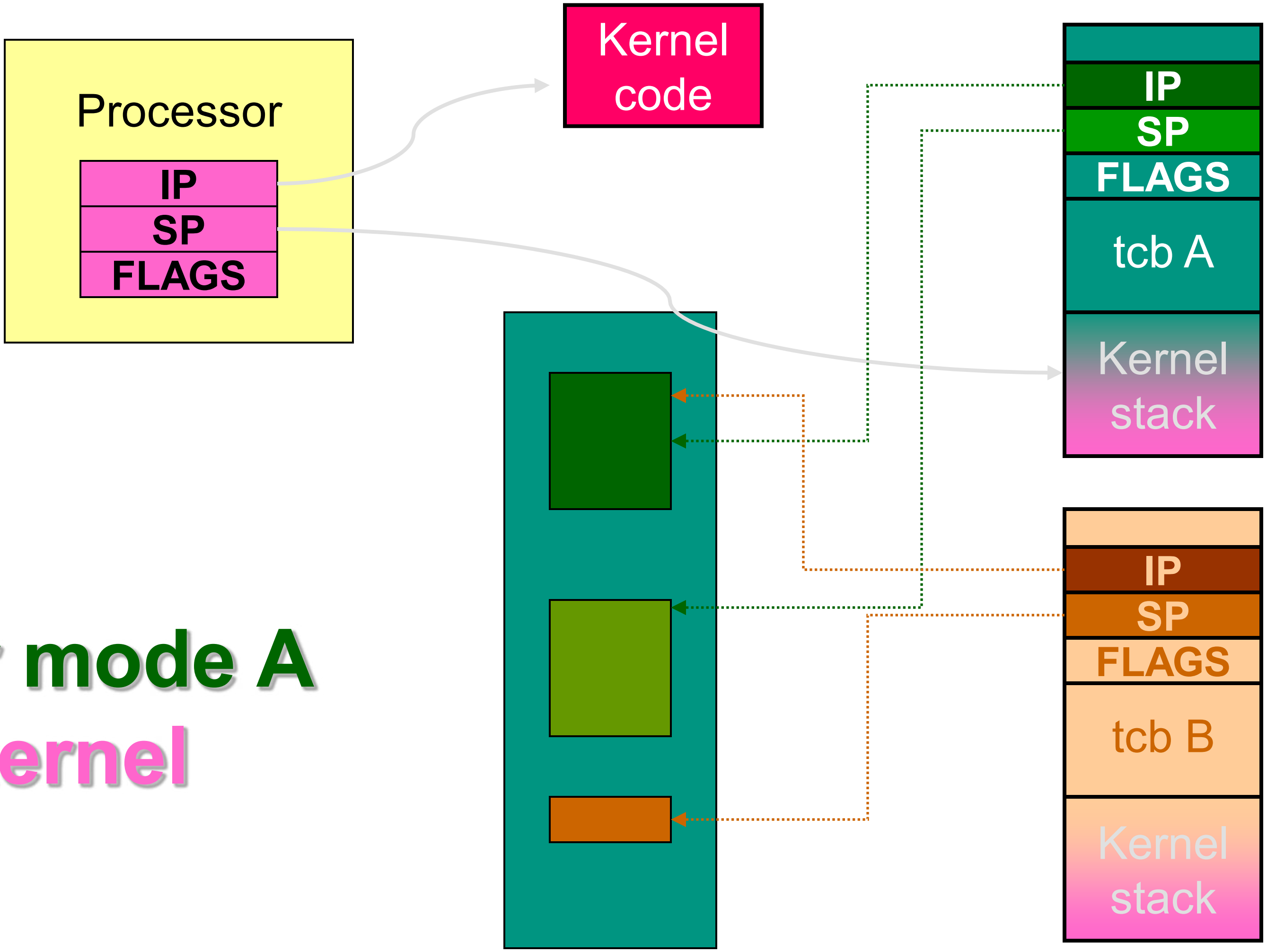
user mode A
kernel

Thread Switch A → kernel → B



user mode A
kernel

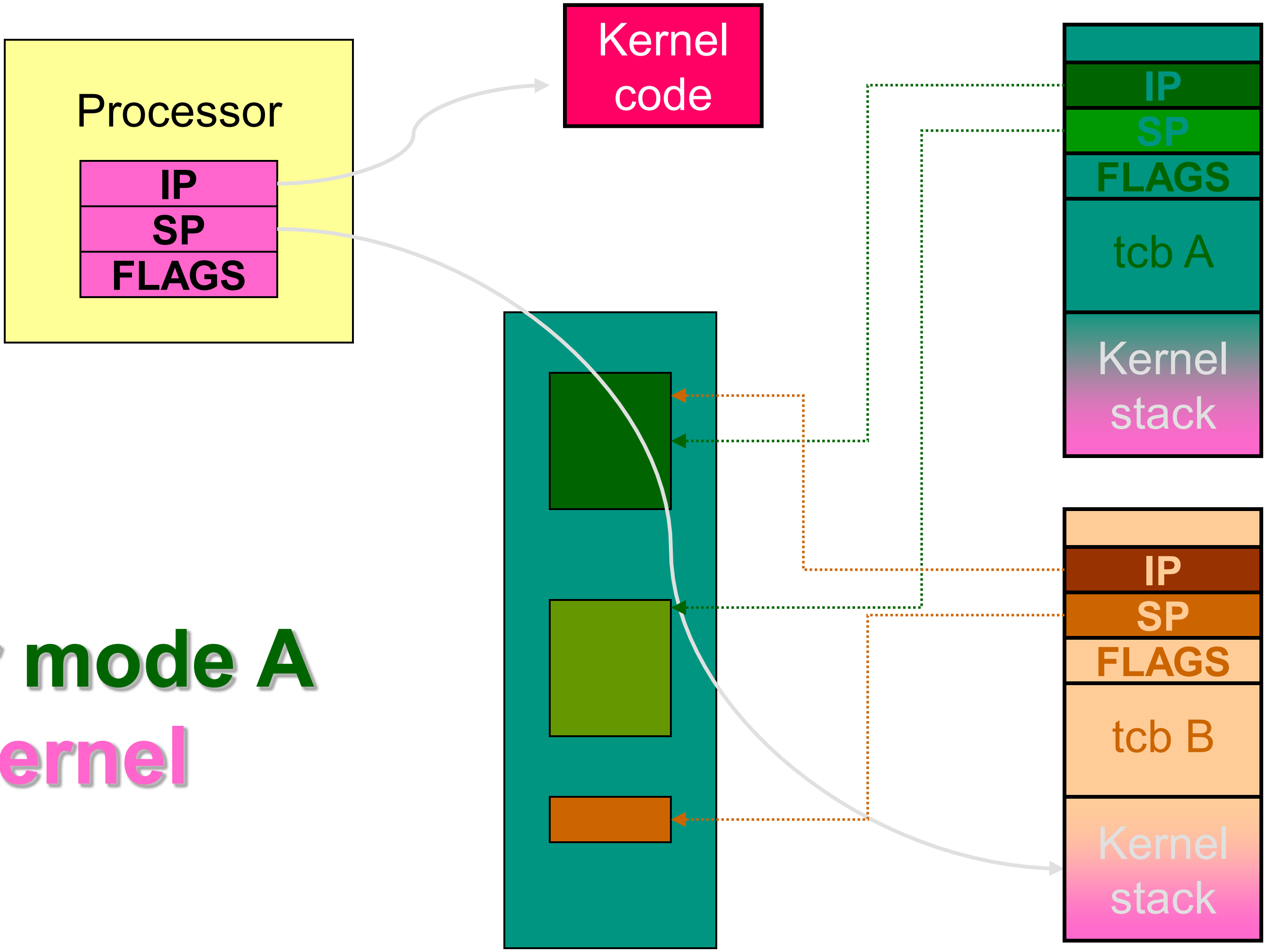
Thread Switch A → kernel → B



user mode A

kernel

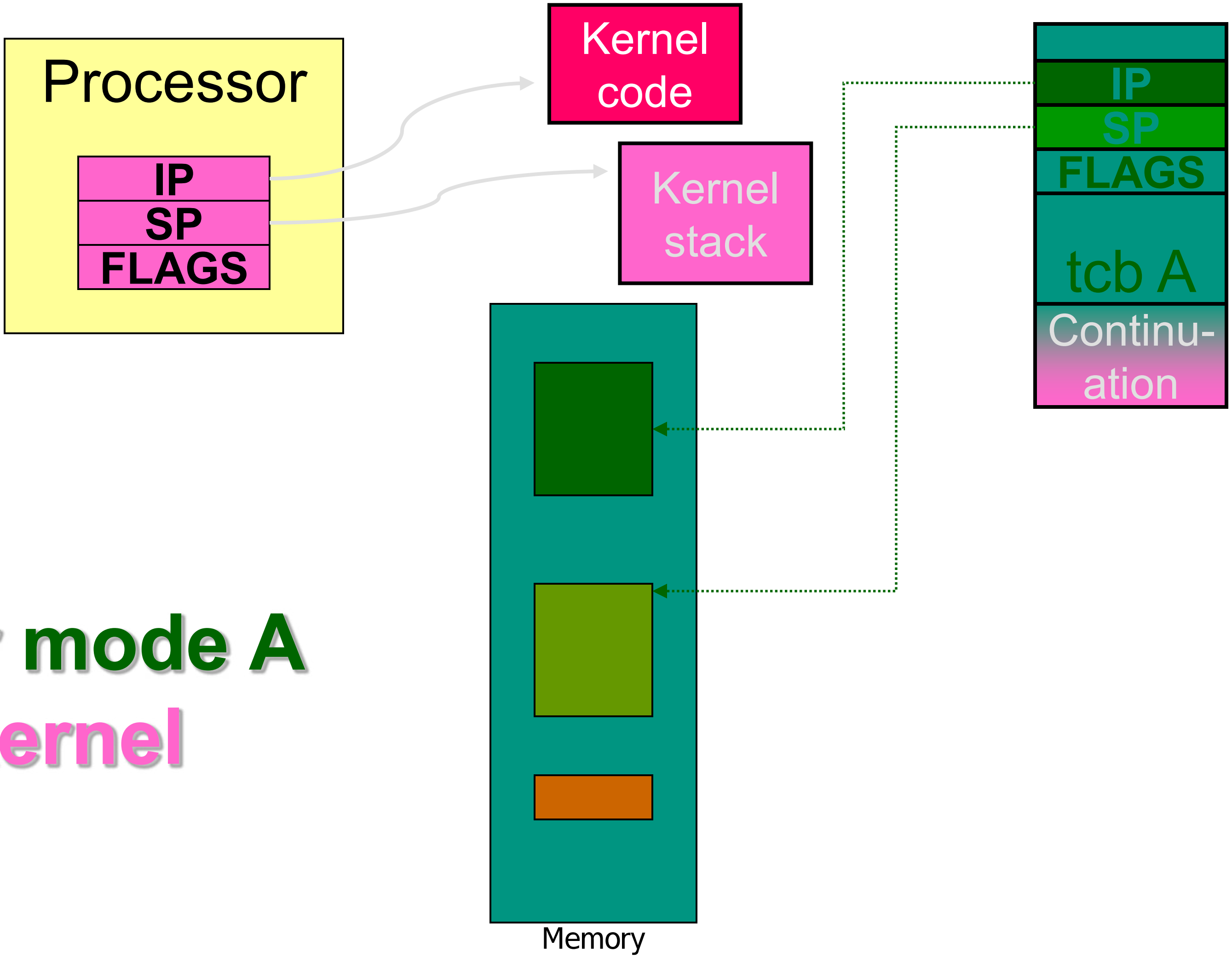
Thread Switch A → kernel → B



user mode A

kernel

Thread Switch with single kernel stack

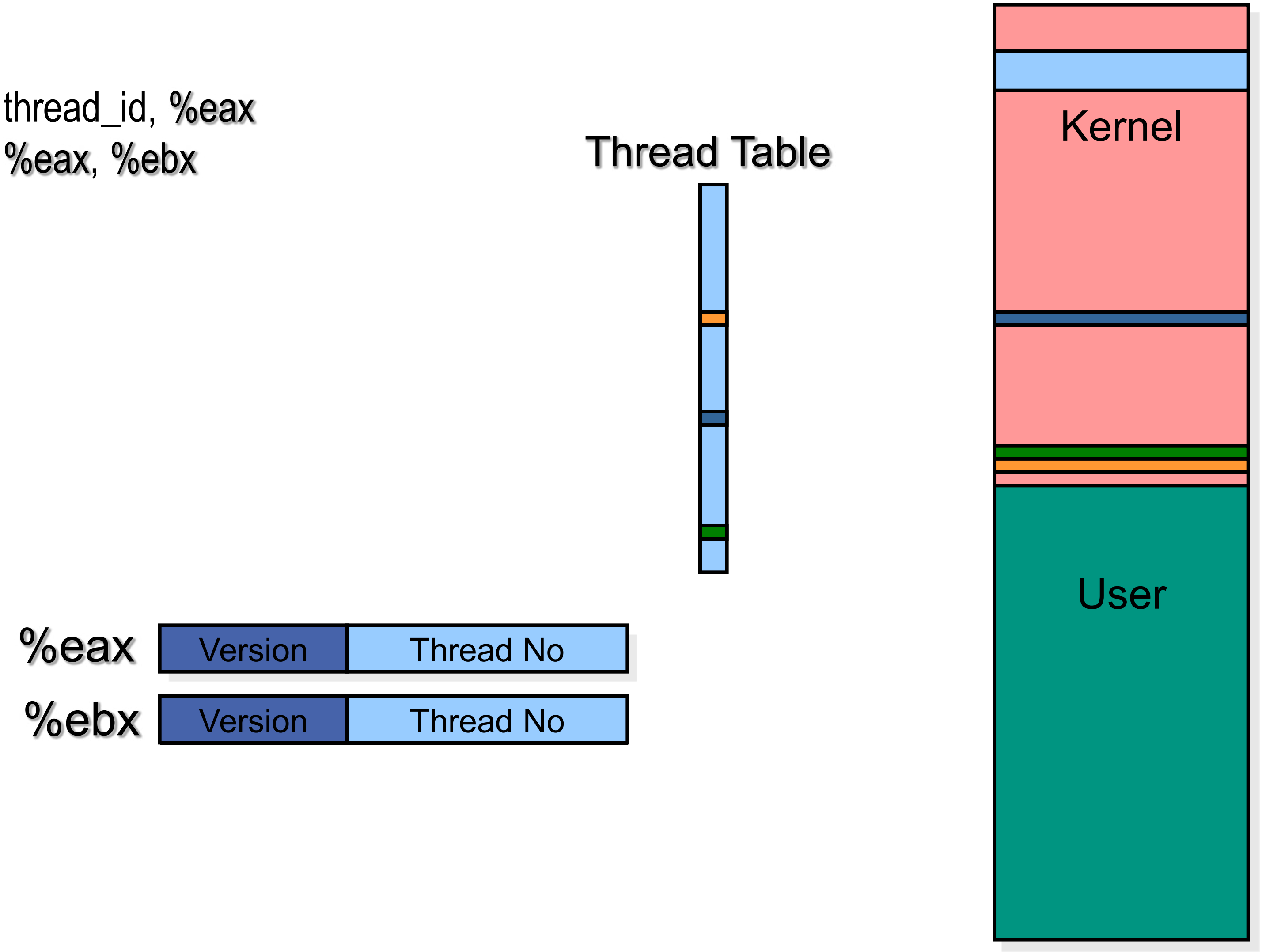


user mode A

kernel

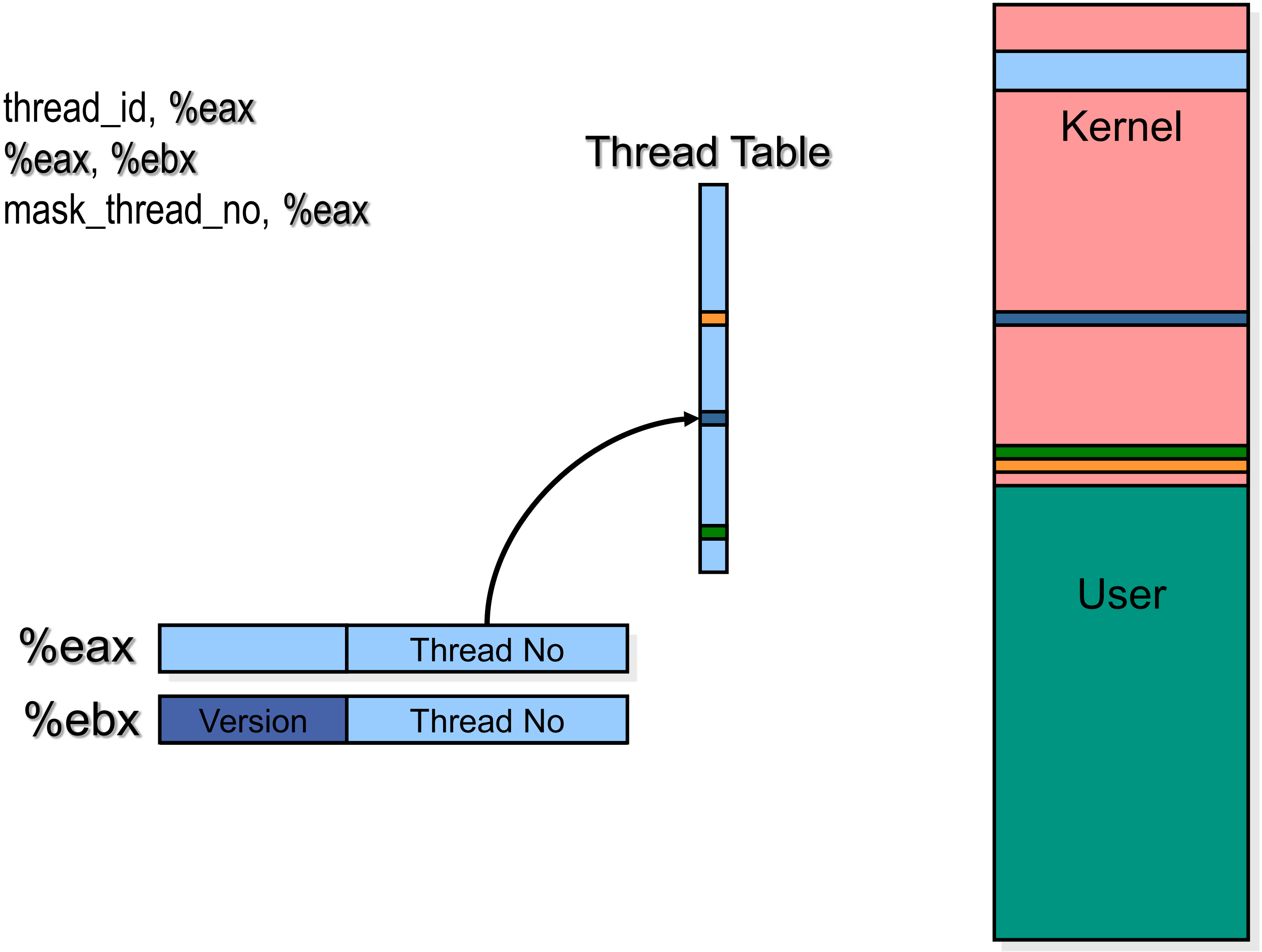
Thread ID → TCB
Indirect via Table

```
movl   thread_id, %eax
movl   %eax, %ebx
```



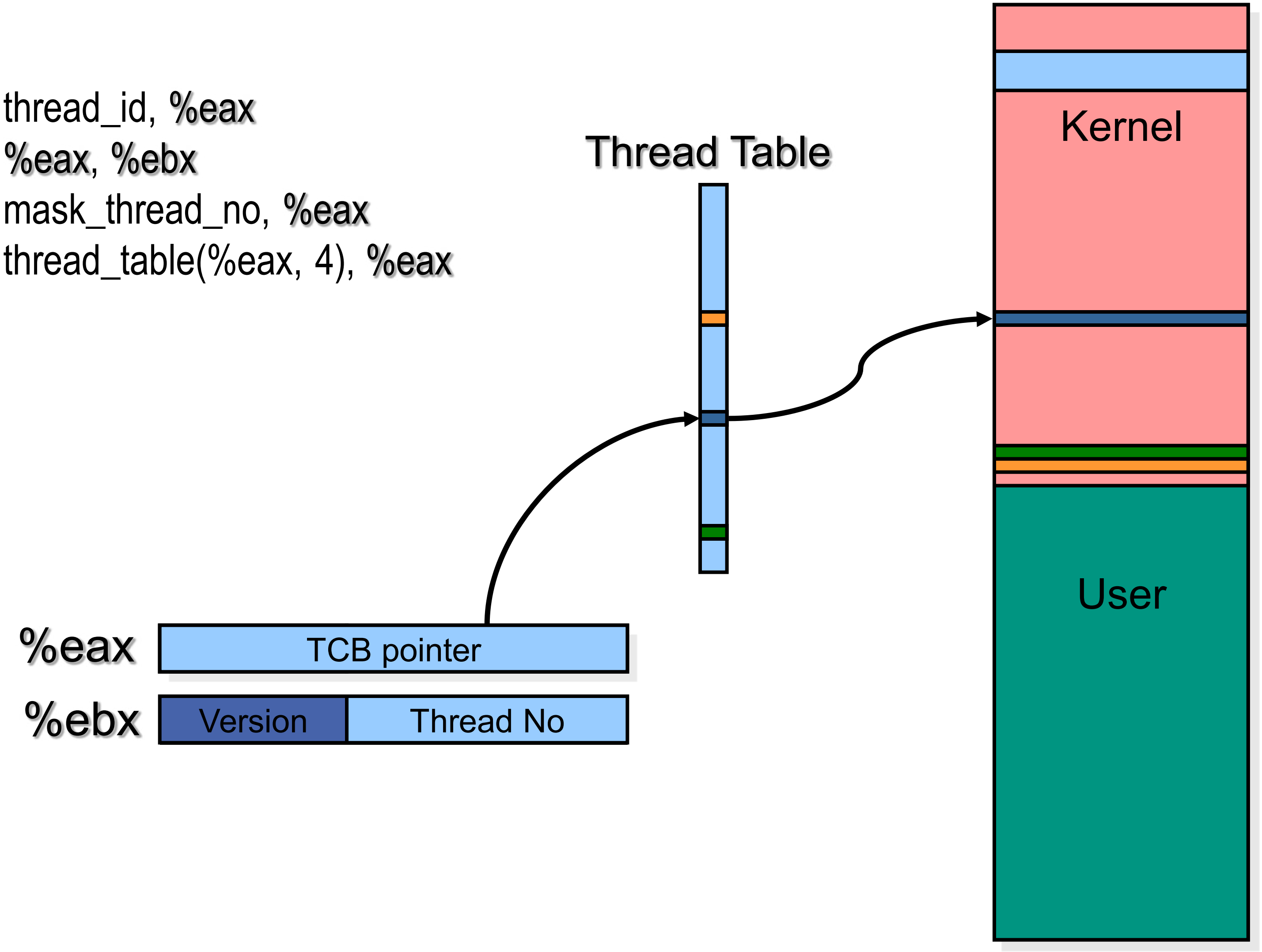
Thread ID → TCB
Indirect via Table

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



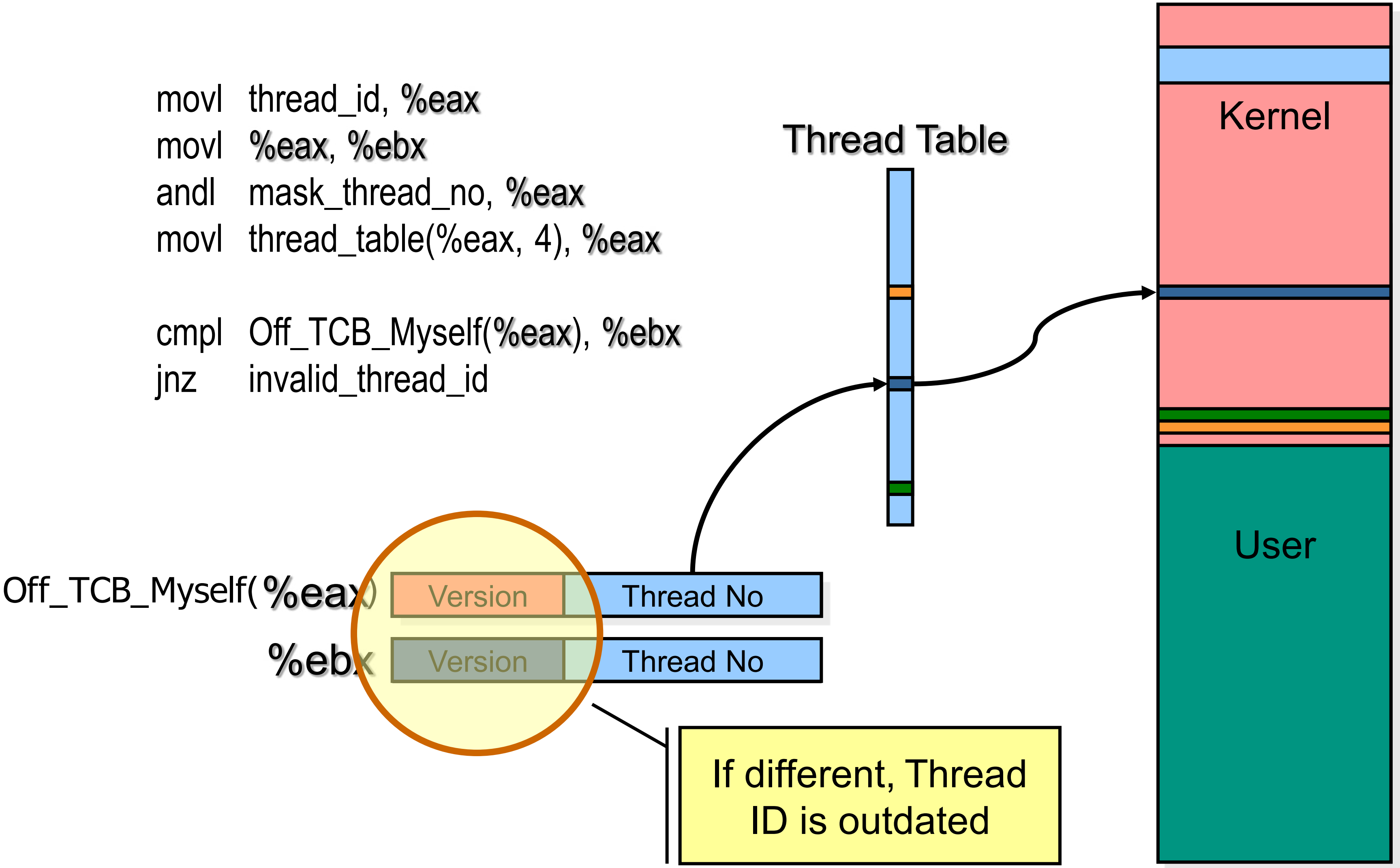
Thread ID → TCB Indirect via Table

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



Thread ID → TCB
Indirect via Table

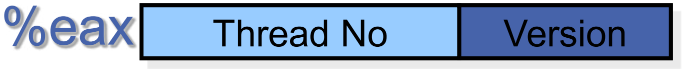
```
movl   thread_id, %eax
movl   %eax, %ebx
andl   mask_thread_no, %eax
movl   thread_table(%eax, 4), %eax

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

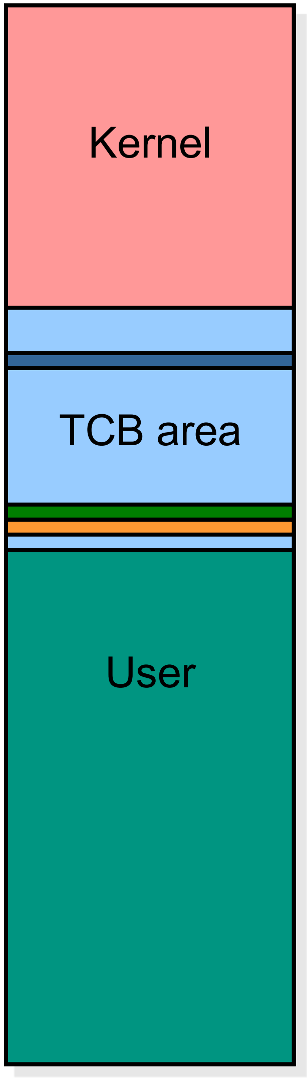


If different, Thread ID is outdated

Thread ID → TCB
Direct Address

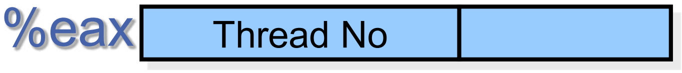
```
movl   thread_id, %eax
movl   %eax, %ebx
```



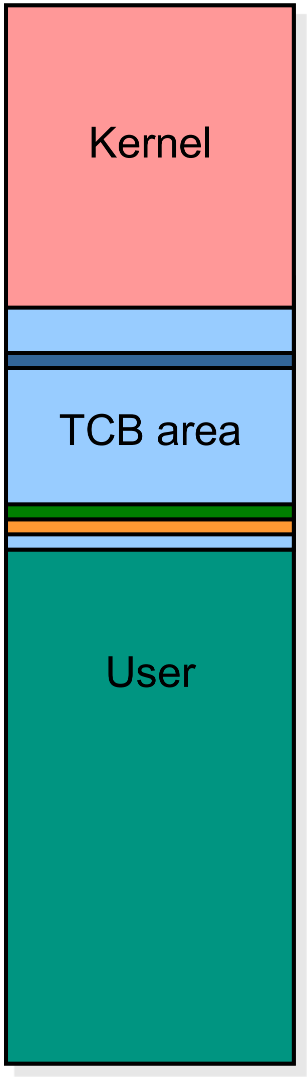


Thread ID → TCB Direct Address

```
movl  thread_id, %eax
movl  %eax, %ebx
andl  mask_version, %eax
```

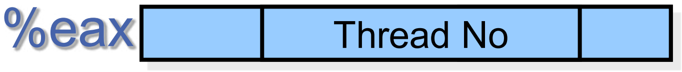


Mask out lower bits



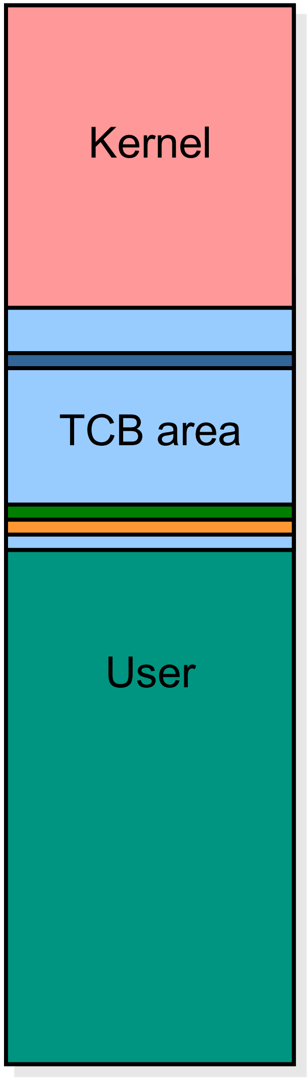
Thread ID → TCB
Direct Address

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



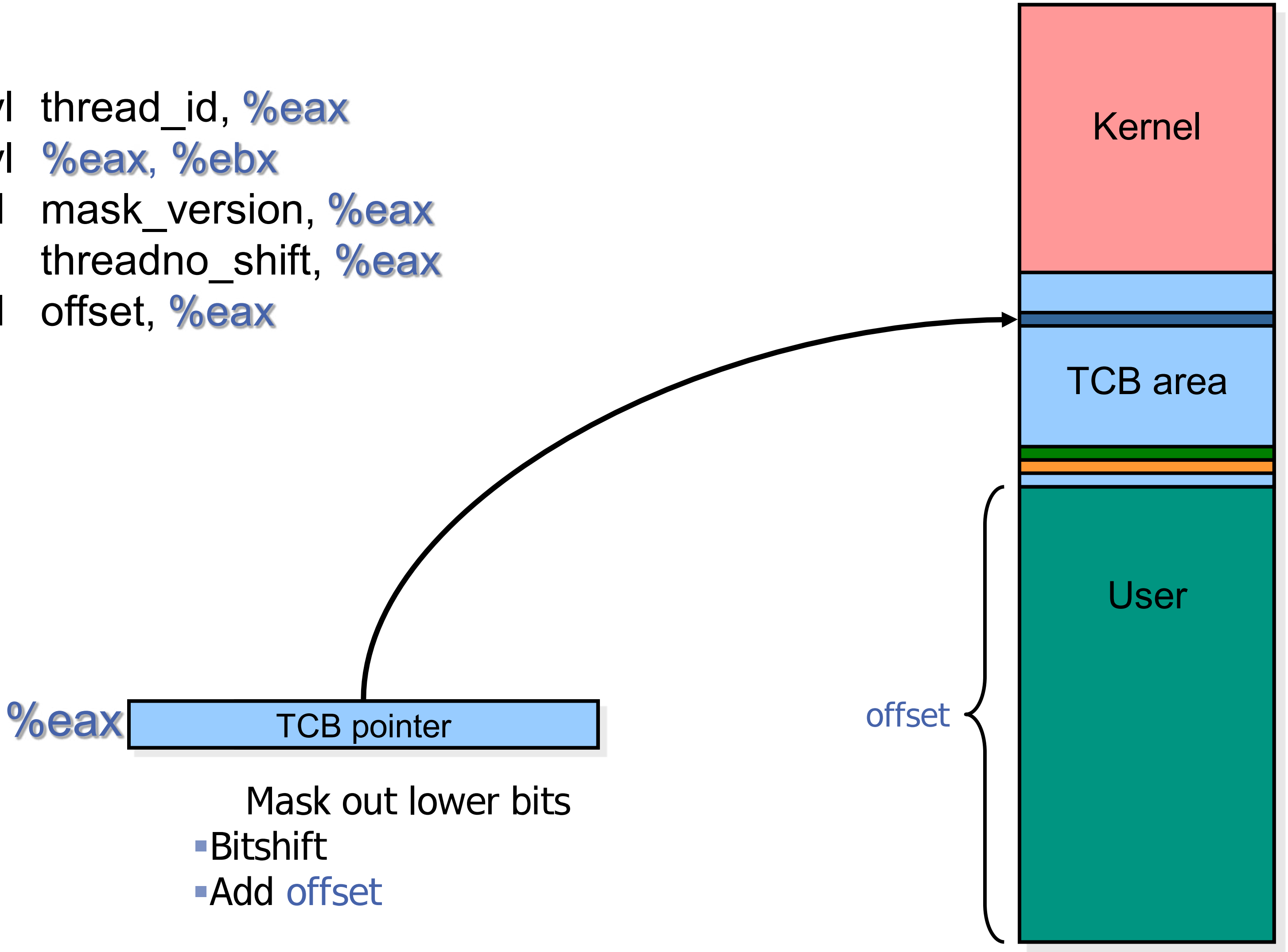
Mask out lower bits

- Bitshift



Thread ID → TCB
Direct Address

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

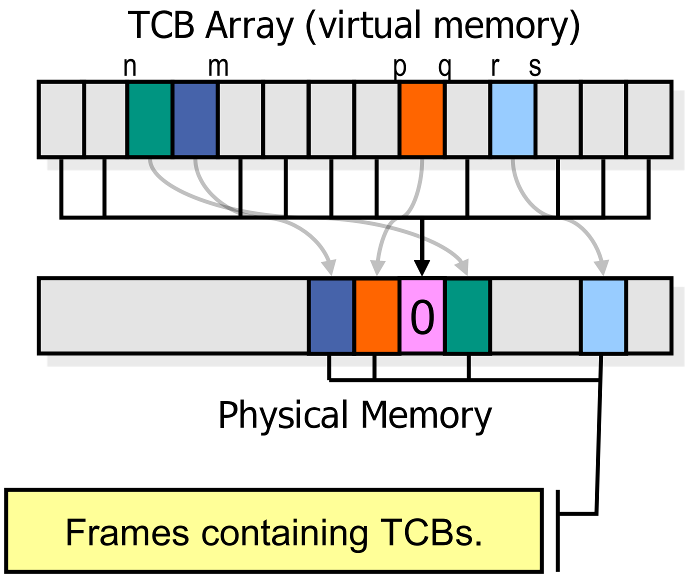


Mask out lower bits

- Bitshift
- Add offset

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** requires **≥ 256 MB** virtual memory for 256k potential TCBs

Thread ID Translation

Via Table

- Table access per TCB
- Many TCBs per TLB entry (TCBs on superpages)
- TLB entry for table (?)

Examples:

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

n

- No table access
- Few TCBs per TLB entry (sparsely populated area)

- TCB pointer array requires 1 MB virtual memory for 256k potential threads
- Virtual TCB array requires ≥ 256 MB virtual memory for 256k potential TCBs

Physical TCB array (seL4)

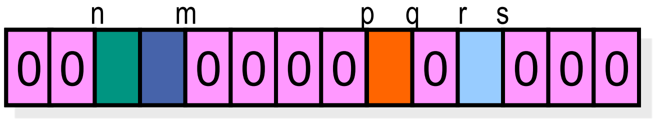
- Problem: Virtual TCB lookups cause TLB misses
  - Virtual TCB lookup is on IPC path!

- Solution: Use physical memory instead

+ No TLB misses
+ Significantly faster overall (Nourai 2005)
+ Easy to verify

– Requires ≥ 256 MB of physical memory!
– MMU may not permit physical addressing
  - Can still emulate physical memory using huge pages + pinning

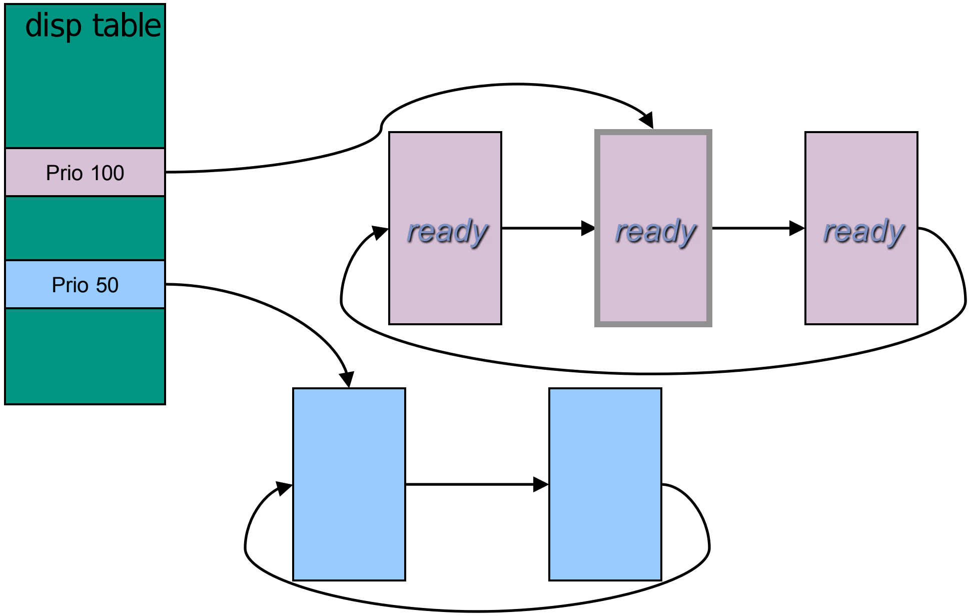
TCB Array (physical memory)



Lazy Dispatching

Thread state toggles frequently (per IPC)

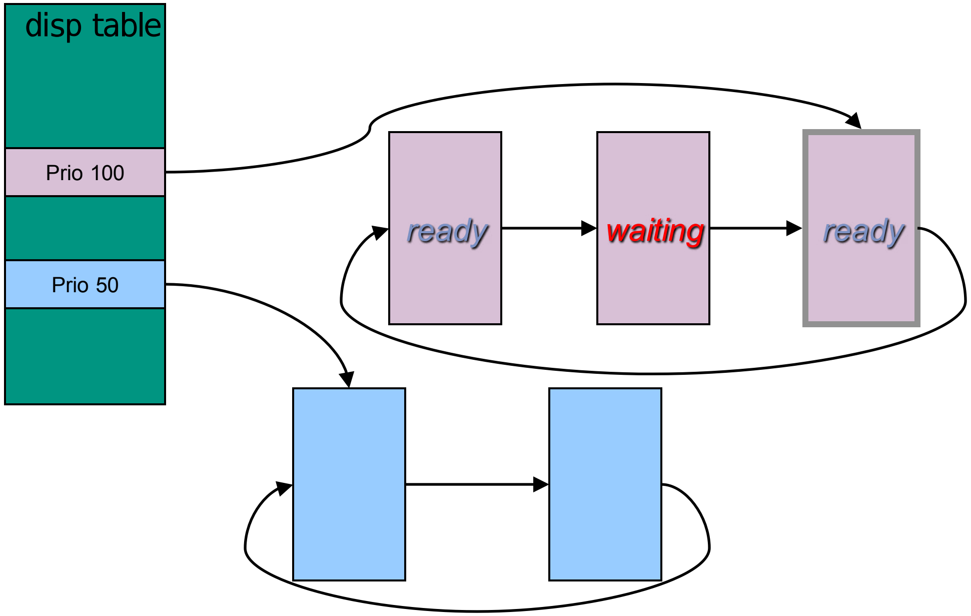
- *ready* ↔ *waiting*
  - Delete/insert ready list is expensive
  - Therefore: delete *lazily* from ready list



Lazy Dispatching

Thread state toggles frequently (per IPC)

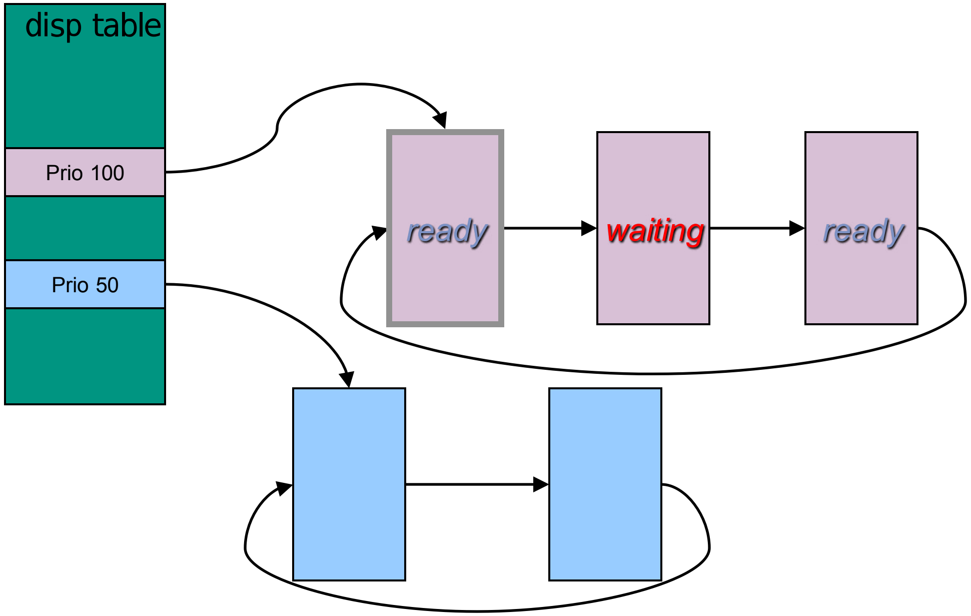
- *ready* ↔ *waiting*
  - Delete/insert ready list is expensive
  - Therefore: delete *lazily* from ready list



Lazy Dispatching

Thread state toggles frequently (per IPC)

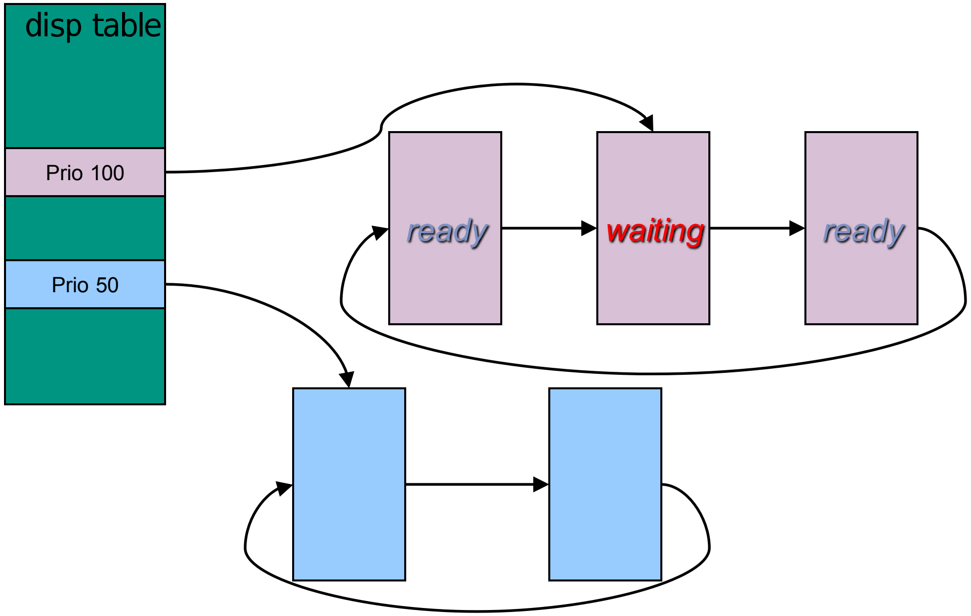
- *ready* ↔ *waiting*
  - Delete/insert ready list is expensive
  - Therefore: delete *lazily* from ready list



Lazy Dispatching

Thread state toggles frequently (per IPC)

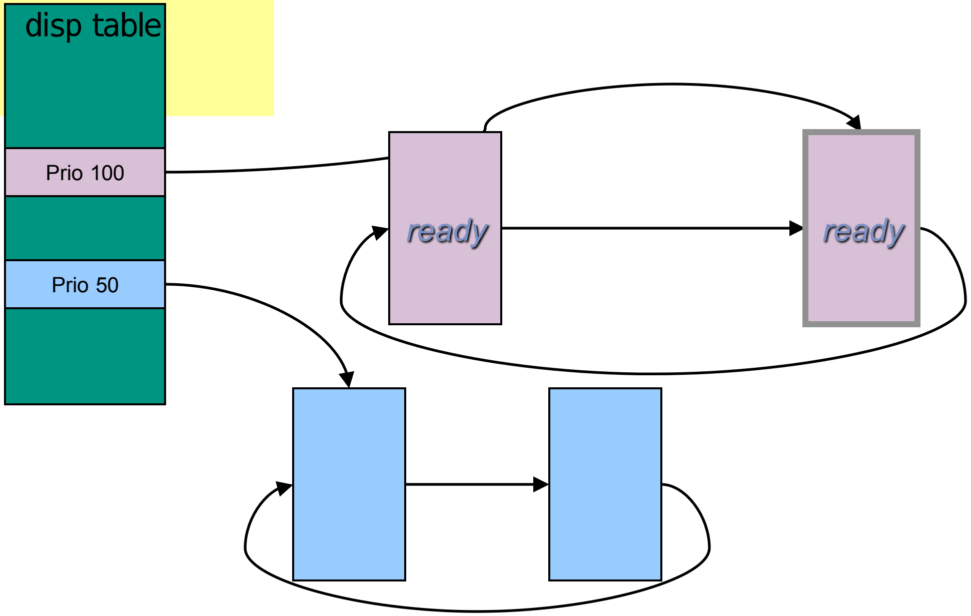
- *ready* ↔ *waiting*
  - Delete/insert ready list is expensive
  - Therefore: delete *lazily* from ready list



Lazy Dispatching

Thread state toggles frequently (per IPC)

- *ready* ↔ *waiting*
  - Delete/insert ready list is expensive
  - Therefore: delete *lazily* from ready list
  - Whenever reaching a non-ready thread
    - Delete it from list
    - Proceed with next



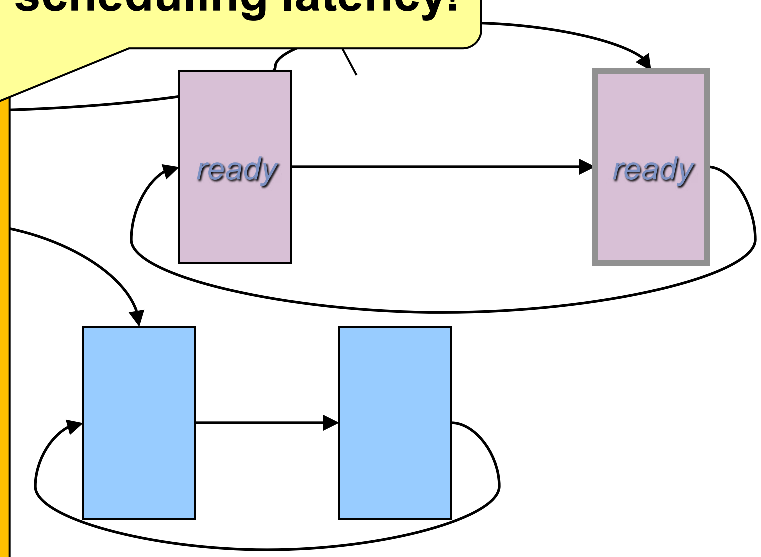
Lazy Dispatching

```
do
    round robin if necessary ;
    if current[highest active p] ≠ nil then
        B := current[highest active p] ; return
    elif highest active p > 0 then
        highest active p -= 1
    else
        idle
    fi
od .
round robin if necessary:
    while current[highest active p] ≠ nil do
        next := current[highest active p].next ;
        if current[highest active p].state ≠ ready then
            delete from list (current[highest active p])
        elif current[highest active p].rem ts = 0 then
            next.rem ts := new ts
        else
            return
        fi ;
        current[highest active p] := next
    od .
```

(PC)

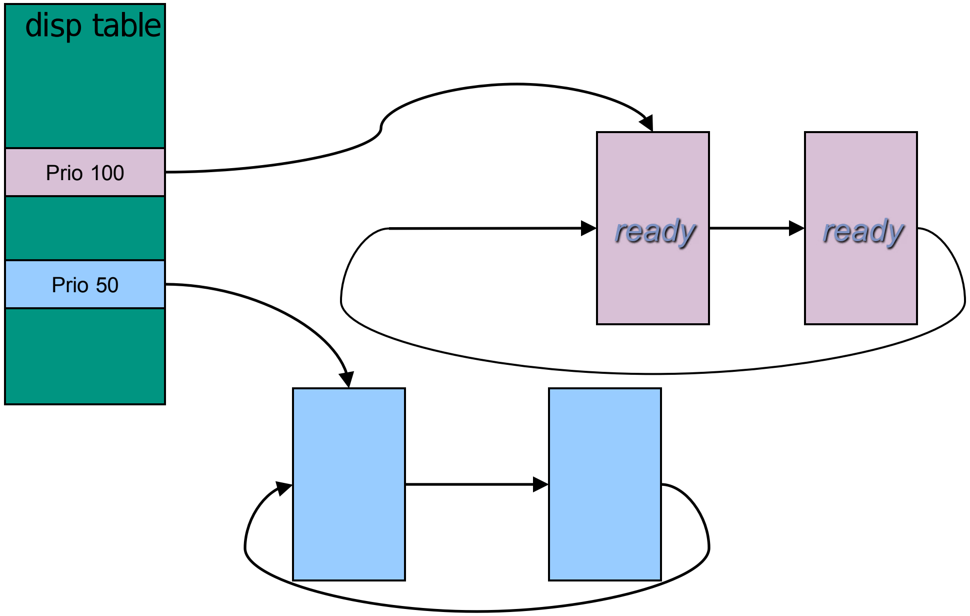
d

O(n) scheduling latency!



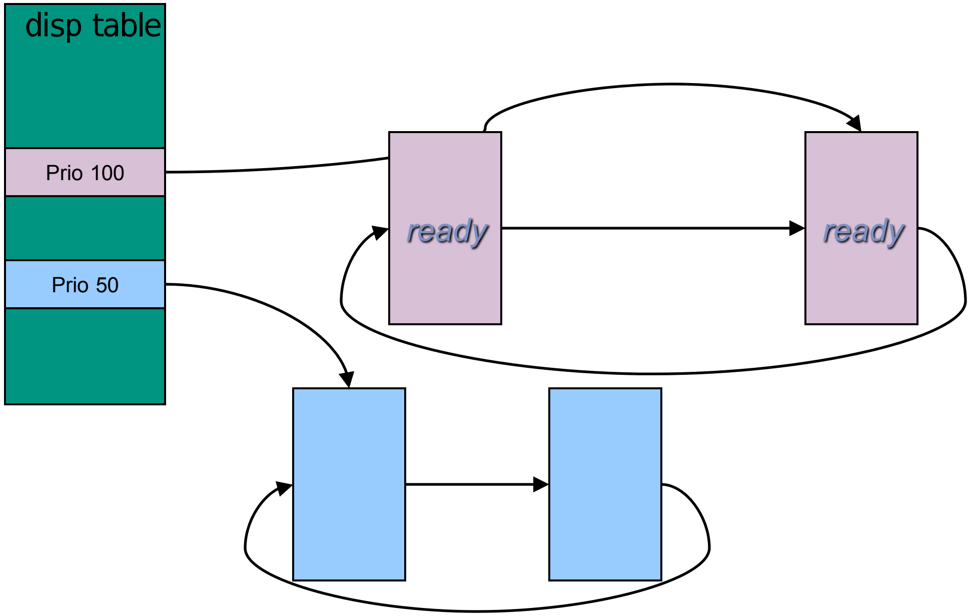
Benno Scheduling (seL4)

Ready list contains all threads
except the currently running thread



Benno Scheduling (seL4)

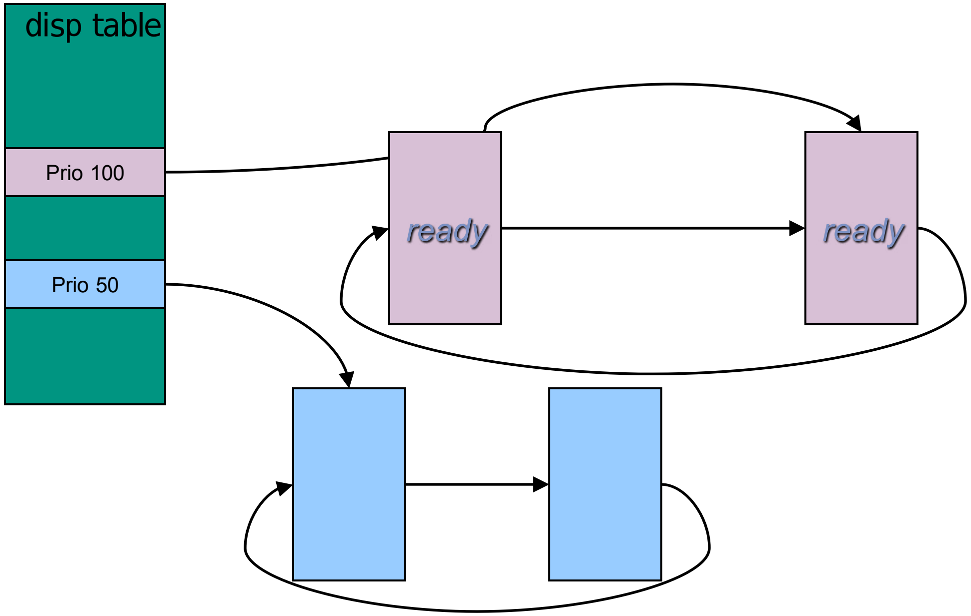
Ready list contains all threads
  except the currently running thread



Benno Scheduling (seL4)

Ready list contains all threads
except the currently running thread

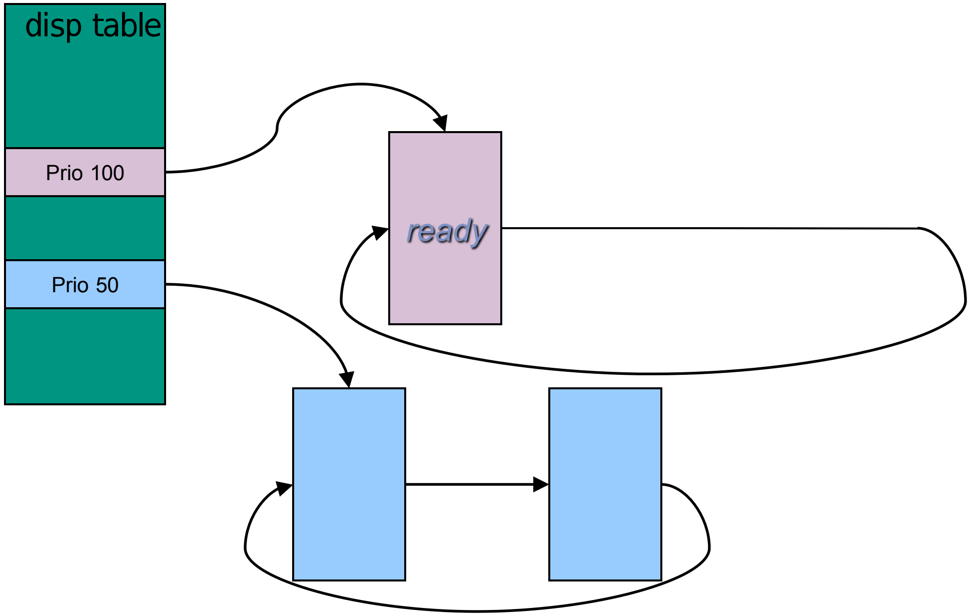
- *ready* ↔ *waiting*
  - Don't re-insert blocked thread



Benno Scheduling (seL4)

Ready list contains all threads
except the currently running thread

- *ready* ↔ *waiting*
  - Don’t re-insert blocked thread

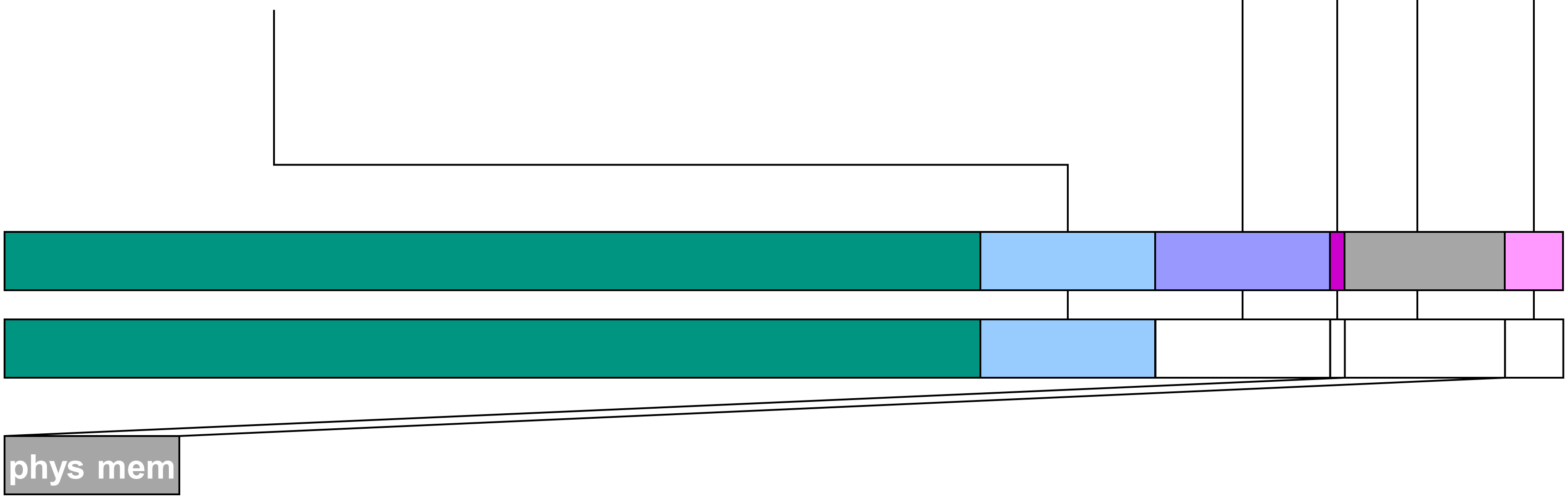


ADDRESS-SPACE LAYOUT

Address-Space Layout

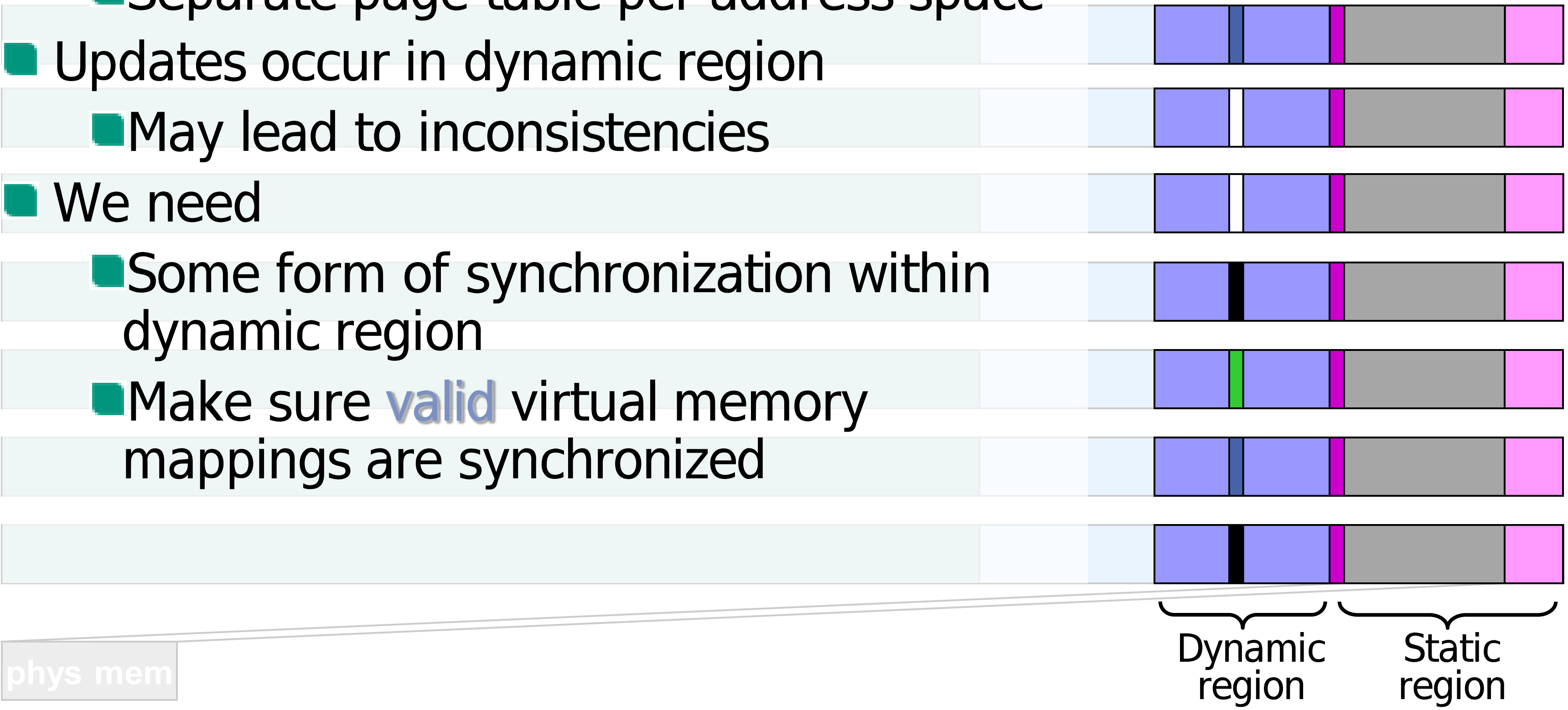
32bits, Virtual TCBs

- User regions
- Shared system regions
  - Other kernel tables
  - Physical memory
  - Kernel code
  - TCBs
- Per-space system regions



Shared Region Synchronization

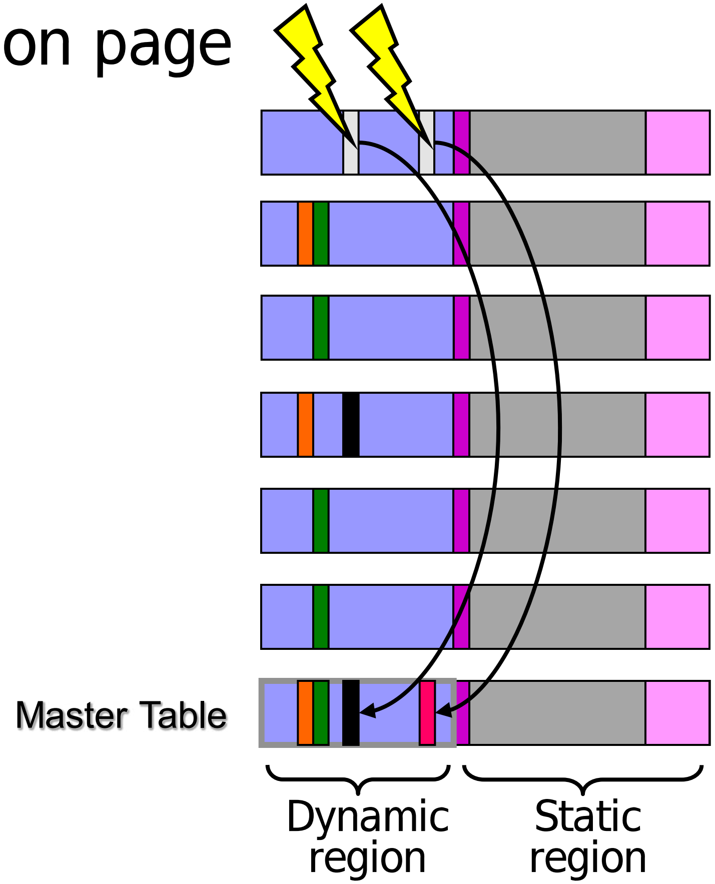
- We have
  - Region shared among all address spaces
  - Separate page table per address space
- Updates occur in dynamic region
  - May lead to inconsistencies
- We need
  - Some form of synchronization within dynamic region
  - Make sure valid virtual memory mappings are synchronized



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
}
```



IPC

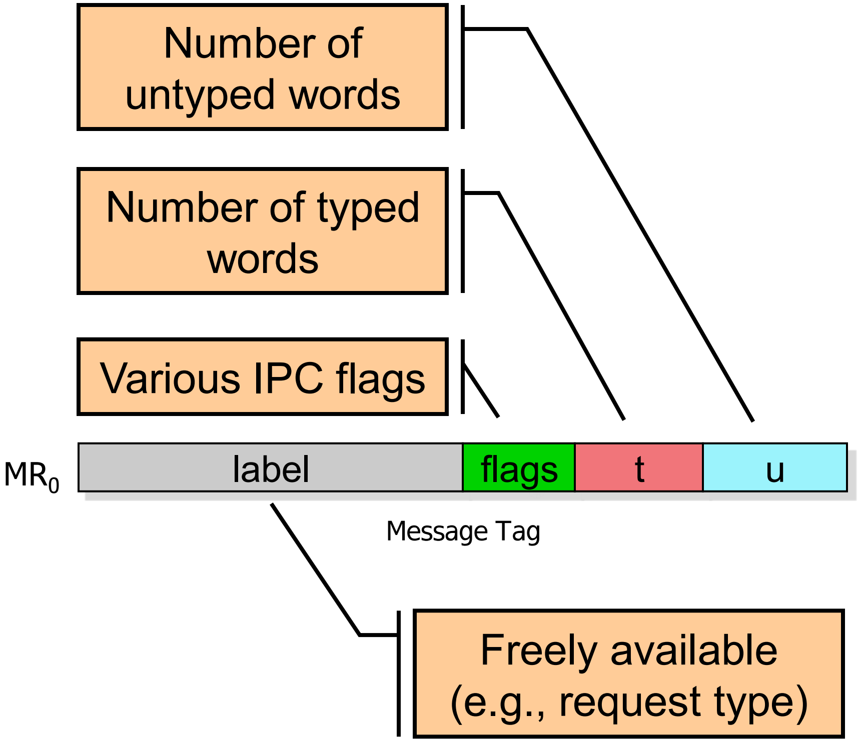
IPC – API

- Operations
  - Send to
  - Receive from
  - Receive
  - Call
  - Send to & Receive any
  - Send to & Receive from
  - Send async

- Message Types
  - Registers
  - Strings
  - Map pages

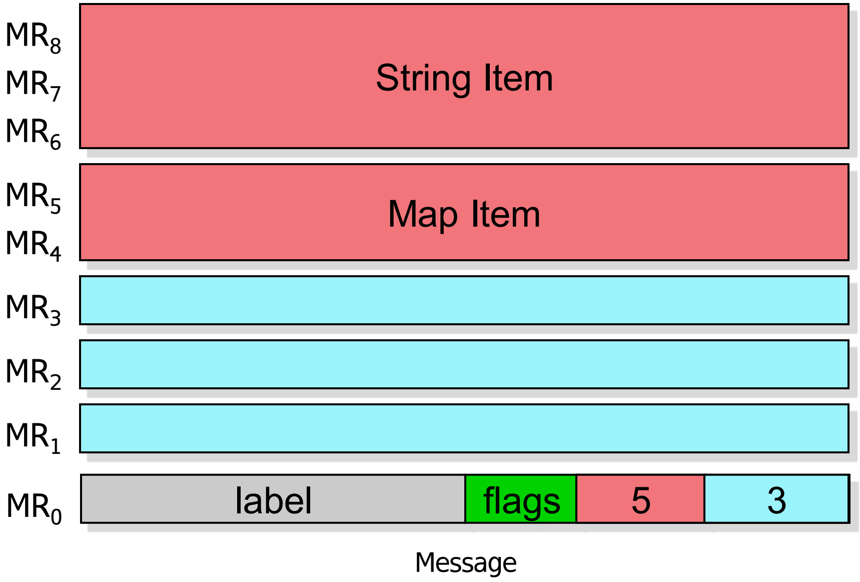
Message Construction

- Messages are stored in registers (MR_0 … MR_{63})
- First register (MR_0) acts as message tag
- Subsequent registers contain
  - Untyped words (u), and
  - Typed words (t)
    (e.g., map item, string item)



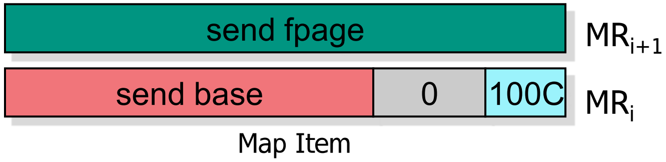
Message Construction

- Typed items occupy one or more words
- Four currently defined items
  - Map item (2 words)
  - Grant item (2 words)
  - String item (2+ words)
  - Capability (2 words)
- Typed items can have arbitrary order

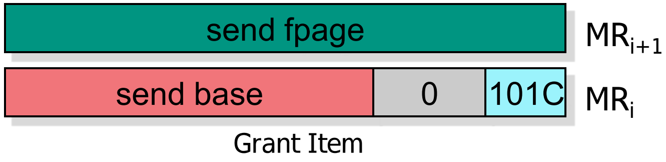


Map and Grant Items

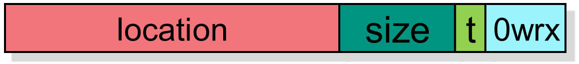
- Two words
  - Send base
  - Fpage
- Lower bits of send base indicates map or grant item



Map Item



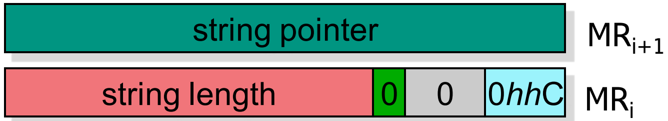
Grant Item



Fpage

String Items

- Up to 4 MB (per string)
- Compound strings supported
  - Allows scatter-gather
- Incorporates cacheability hints
  - Reduce cache pollution for long copy operations



String Item

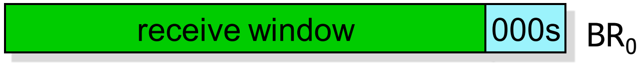
“hh” indicates cacheability hints for the string

E.g., only use L2 cache, or do not use cache at all

Receiving Messages

- Receiver buffers are specified in registers (BR_0 … BR_{33})

- First BR (BR_0) contains "Acceptor"
  - May specify receive window (if not nil-fpage)
  - May indicate presence of receive strings/buffers (if s-bit set)



Acceptor

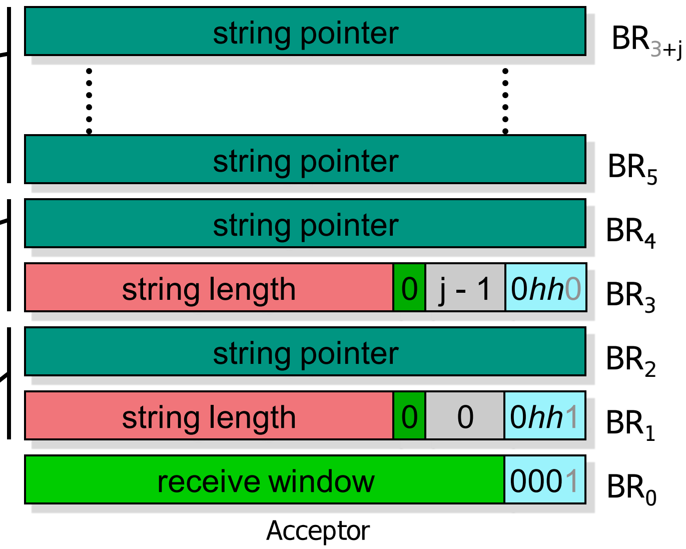
Receiving Messages

If C-bit in string item is cleared, it indicates that no more receive buffers are present

A receive buffer can of course be a compound string

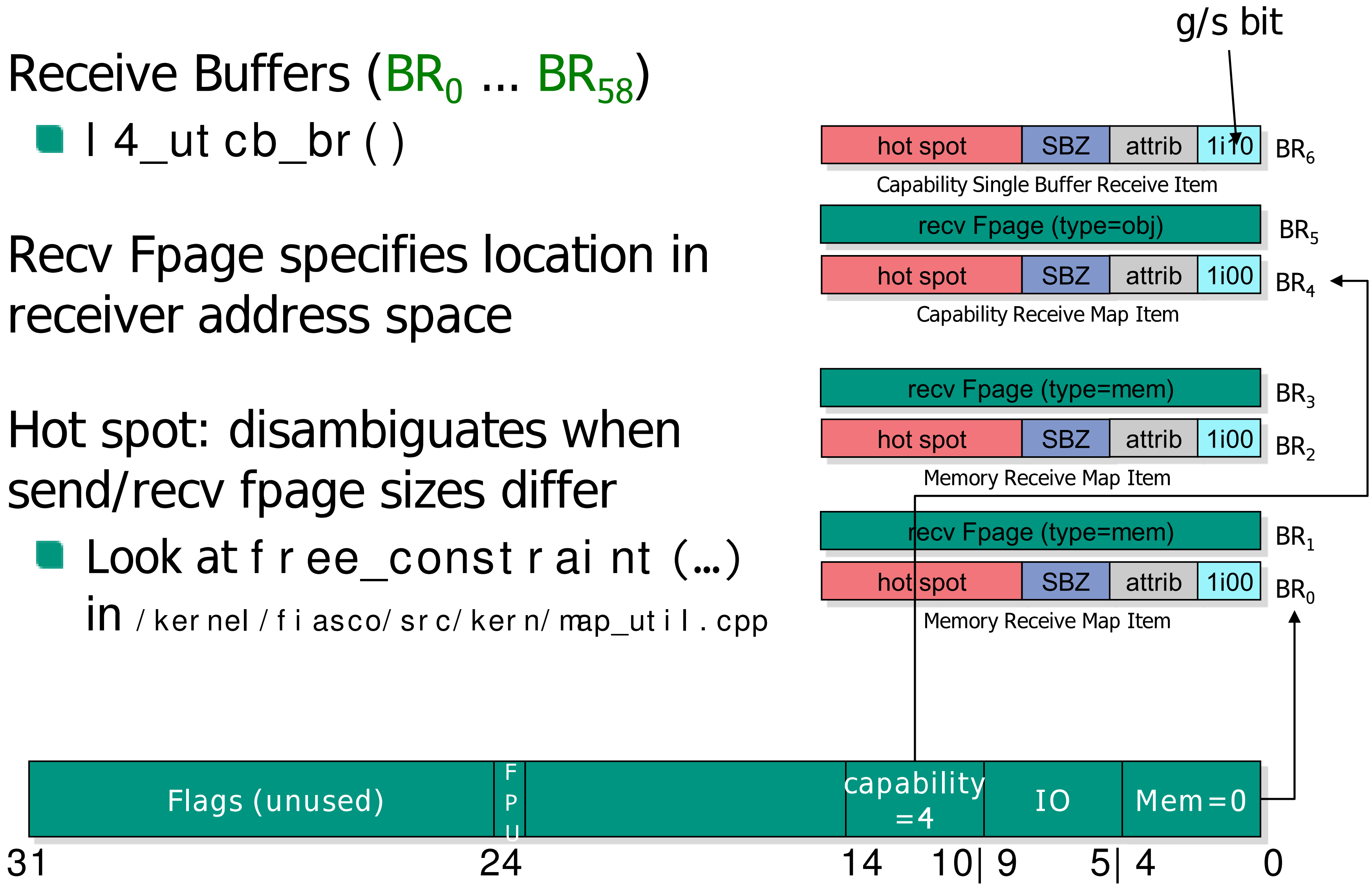
If C-bit in string item is set, it indicates presence of more receive buffers

The s-bit set indicates presence of string items acting as receive buffers



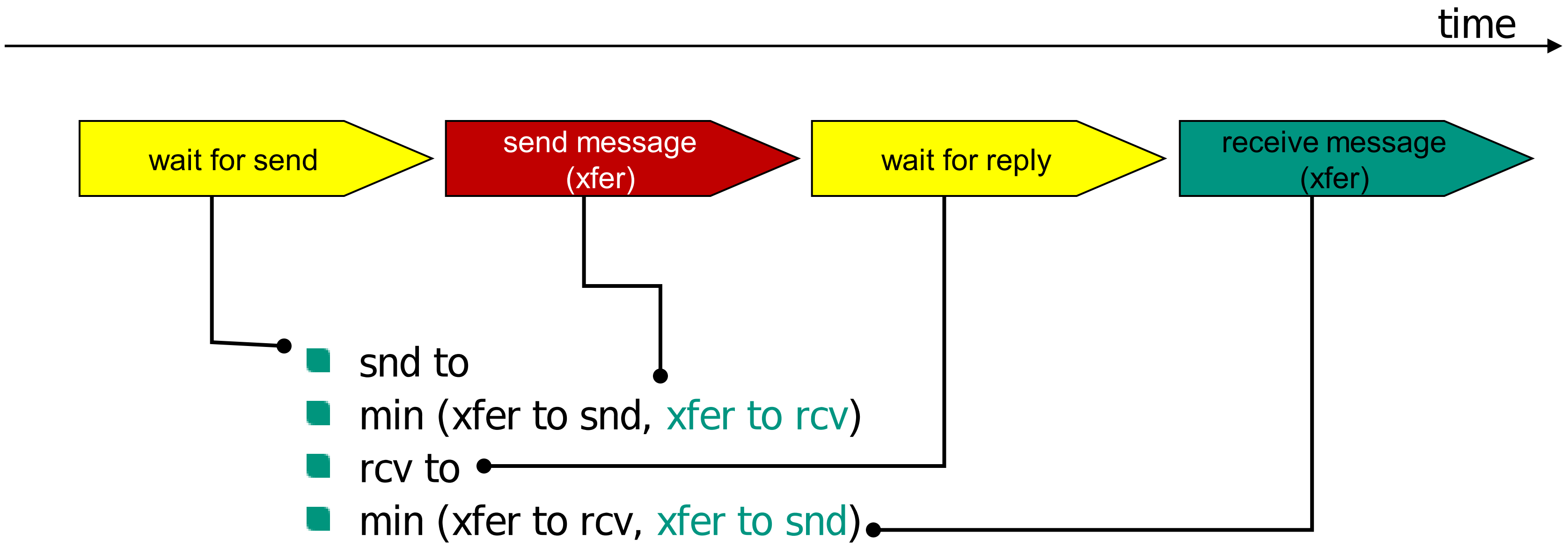
Receiving mappings in Fiasco.OC

- Receive Buffers (BR_0 … BR_{58})
  - l4_utcb_br()
- Recv Fpage specifies location in receiver address space
- Hot spot: disambiguates when send/recv fpage sizes differ
  - Look at free_constraint(…) in /kernel/fiasco/src/kern/map_util.cpp



Timeouts

- snd timeout, rcv timeout, xfer timeout snd, xfer timeout rcv



- snd to
- min (xfer to snd, xfer to rcv)
- rcv to
- min (xfer to rcv, xfer to snd)

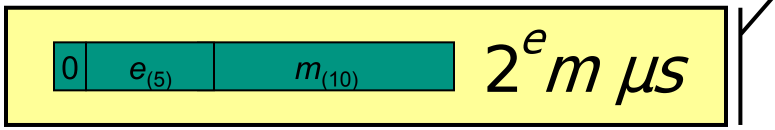
(specified by the partner thread)

Timeout Issues

- What timeout values are typical or necessary?

- How do we encode timeouts to minimize space needed to specify all four values?

- Timeout values
  - ∞ (infinite)
    - Client waiting for a (trusted) server
  - 0 (zero)
    - Server responding to a client
    - Polling
  - Specific time
    - 1 us – 610 h (log)



Timeout Issues

- Timeout values
  - ∞ (infinite)
    - Client waiting for a (trusted) server
  - 0 (zero)
    - Server responding to a client
    - Polling
  - ~~Specific time~~
    - ~~1 µs – 610 h (log)~~

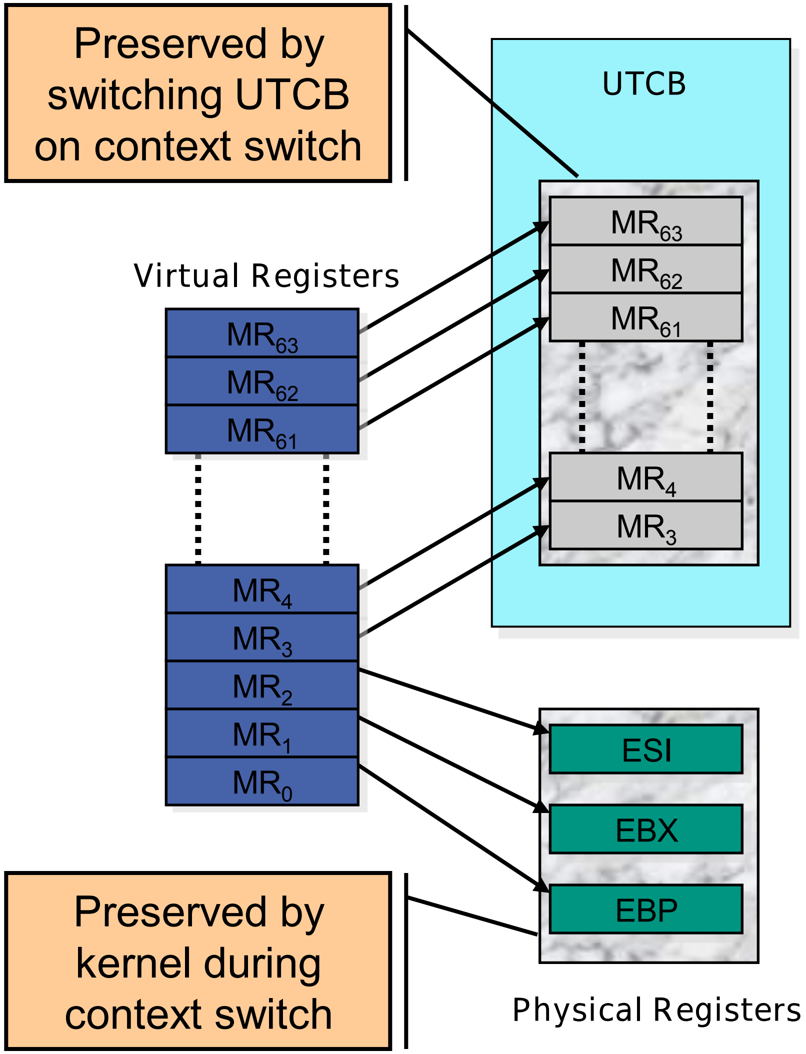
- Does not happen in practice
  - Cannot predict how long a given transfer will take

- SeL4: 1 bit timeout (zero or infinite)

What are Virtual Registers?

- Virtual registers are backed by either
  - Physical registers, or
  - Non-pageable memory

- UTCBs hold the memory backed registers
  - UTCBs are thread local
  - UTCB can not be paged
    - No page faults
    - Registers always accessible



Implementation Goal

- Most frequent kernel op: short IPC
  - Thousands of invocations per second
- Performance is critical
  - Structure IPC for speed
  - Structure entire kernel to support fast IPC
- What affects performance?
  - Cache line misses
  - TLB misses
  - Memory references
  - Pipe stalls and flushes
  - Instruction scheduling

Fast Path

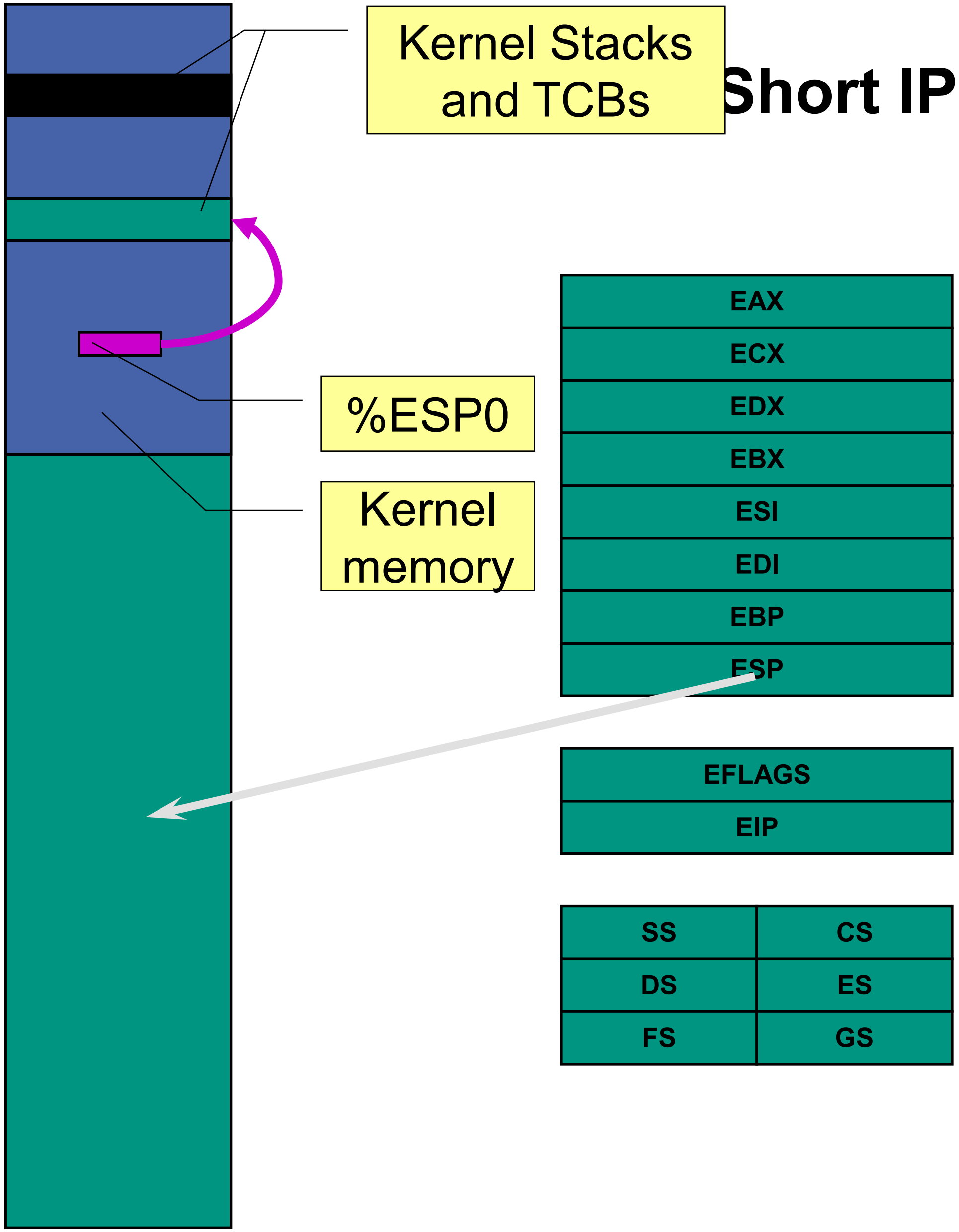
- Optimize for common cases
  - Write in assembler
  - Non-critical paths written in C++
    - But still fast as possible
- Avoid high-level language overhead
  - Function call state preservation
  - Incompatible code optimizations
- We want every cycle possible!
  - At least sometimes …

Avoid memory accesses

- Memory references are slow
  - Avoid in IPC
    - E.g., use lazy scheduling
  - Avoid in common case
    - E.g., (xfer) timeouts

- Microkernel should minimize artifacts
  - Cache pollution
  - TLB pollution
  - Memory bus

Short IPC



Kernel Stacks and TCBs

%ESP0

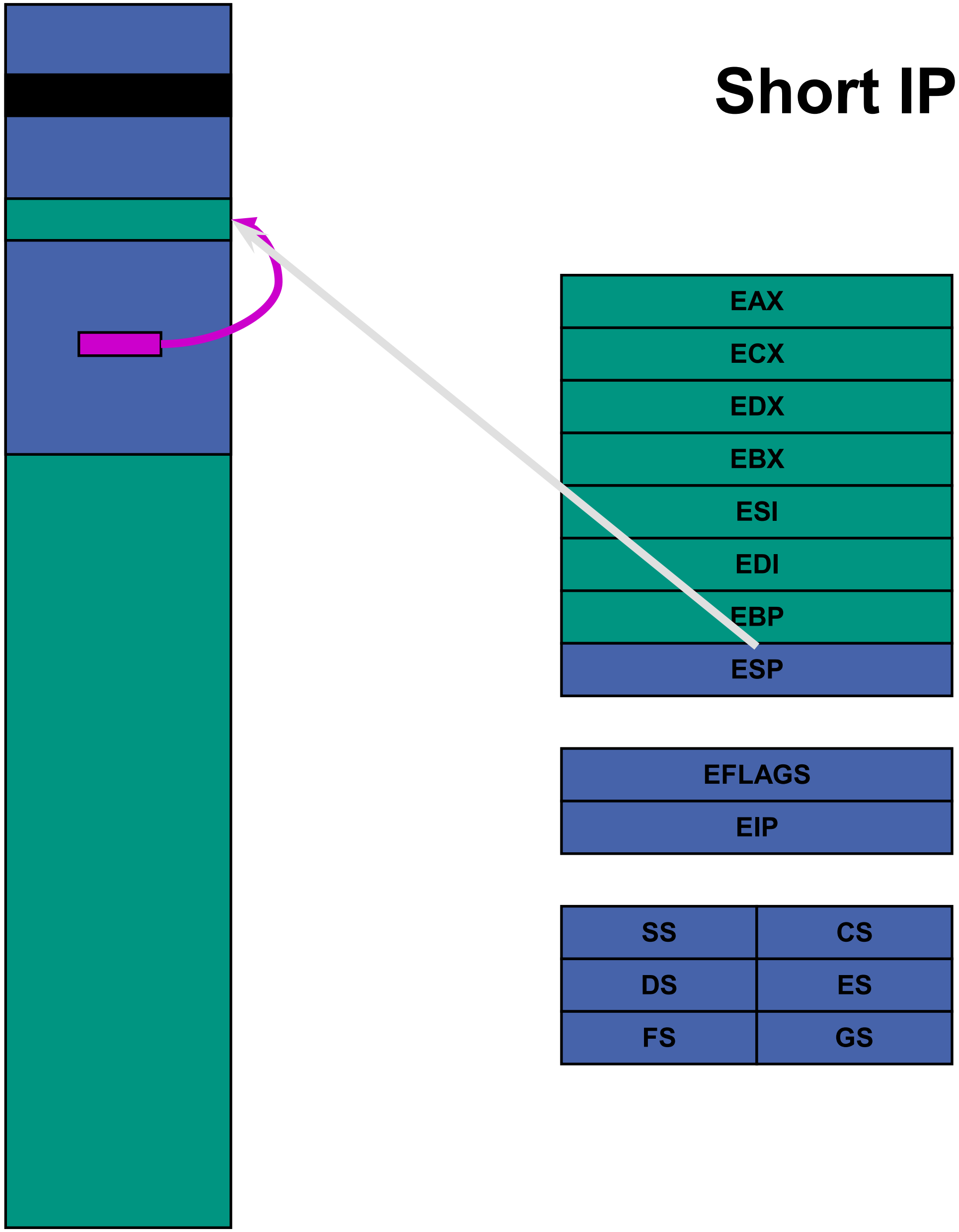
Kernel memory

EAX
ECX
EDX
EBX
ESI
EDI
EBP
ESP

EFLAGS
EIP

SS	CS
DS	ES
FS	GS

Short IPC

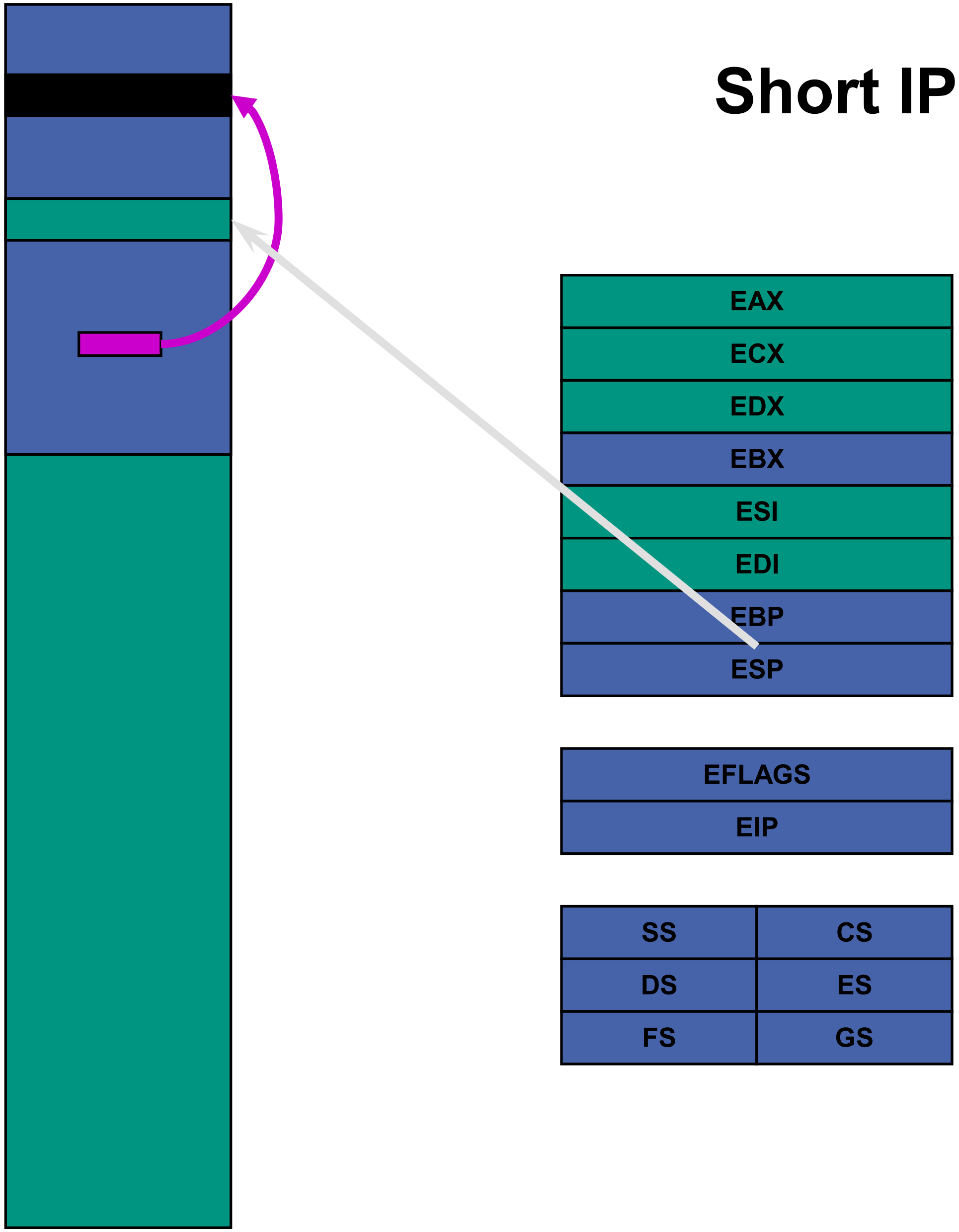


EAX
ECX
EDX
EBX
ESI
EDI
EBP
ESP

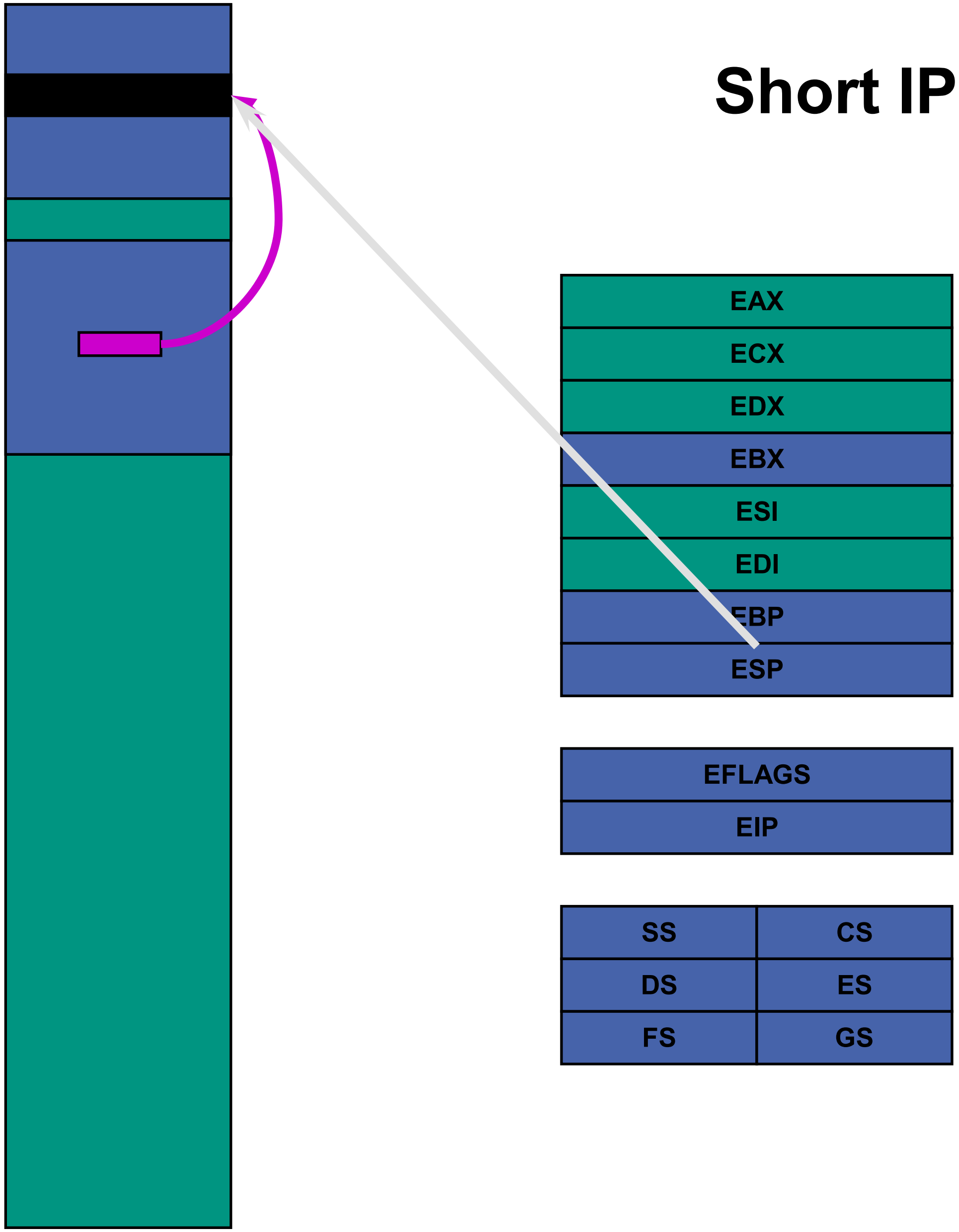
EFLAGS
EIP

SS	CS
DS	ES
FS	GS

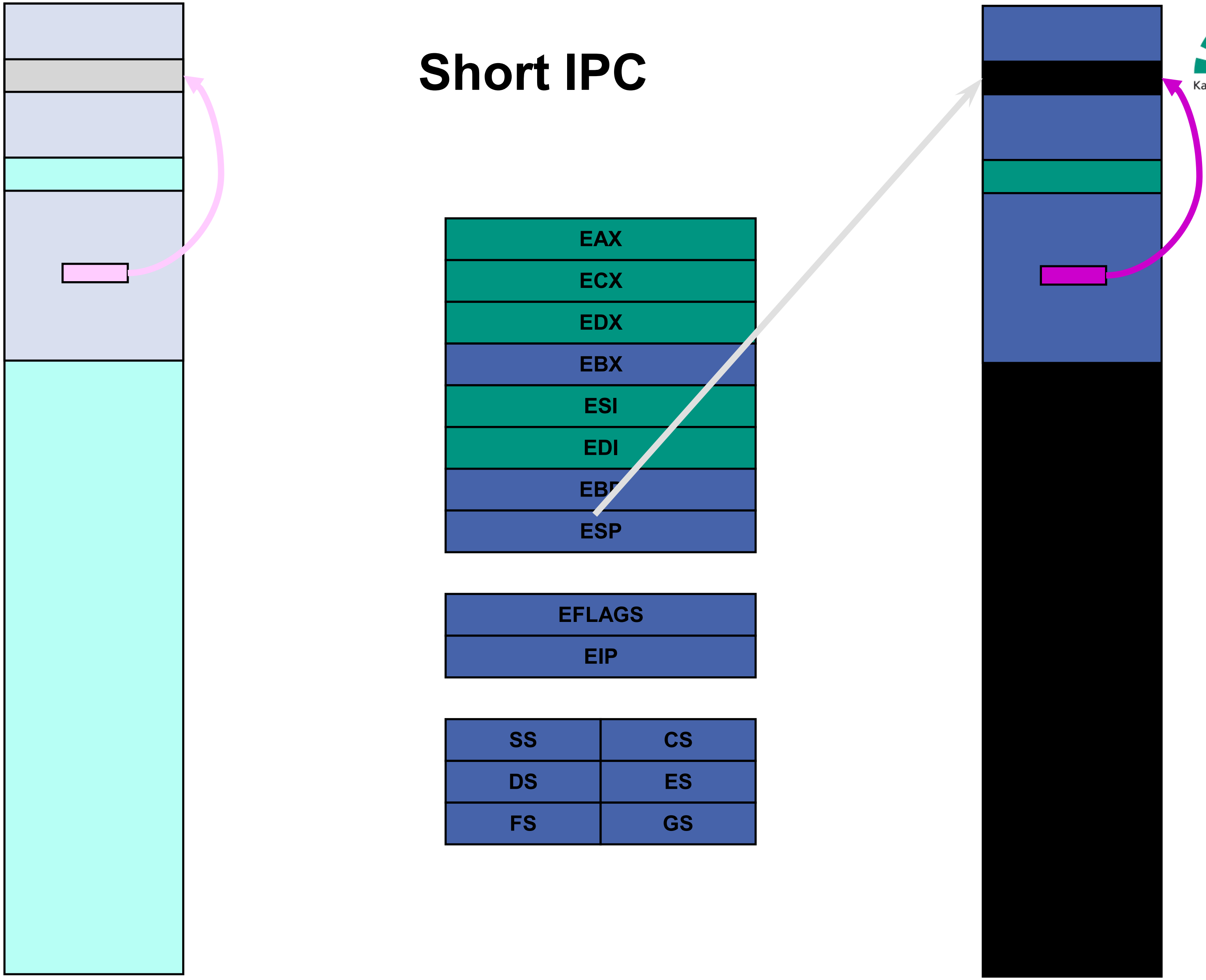
Short IPC



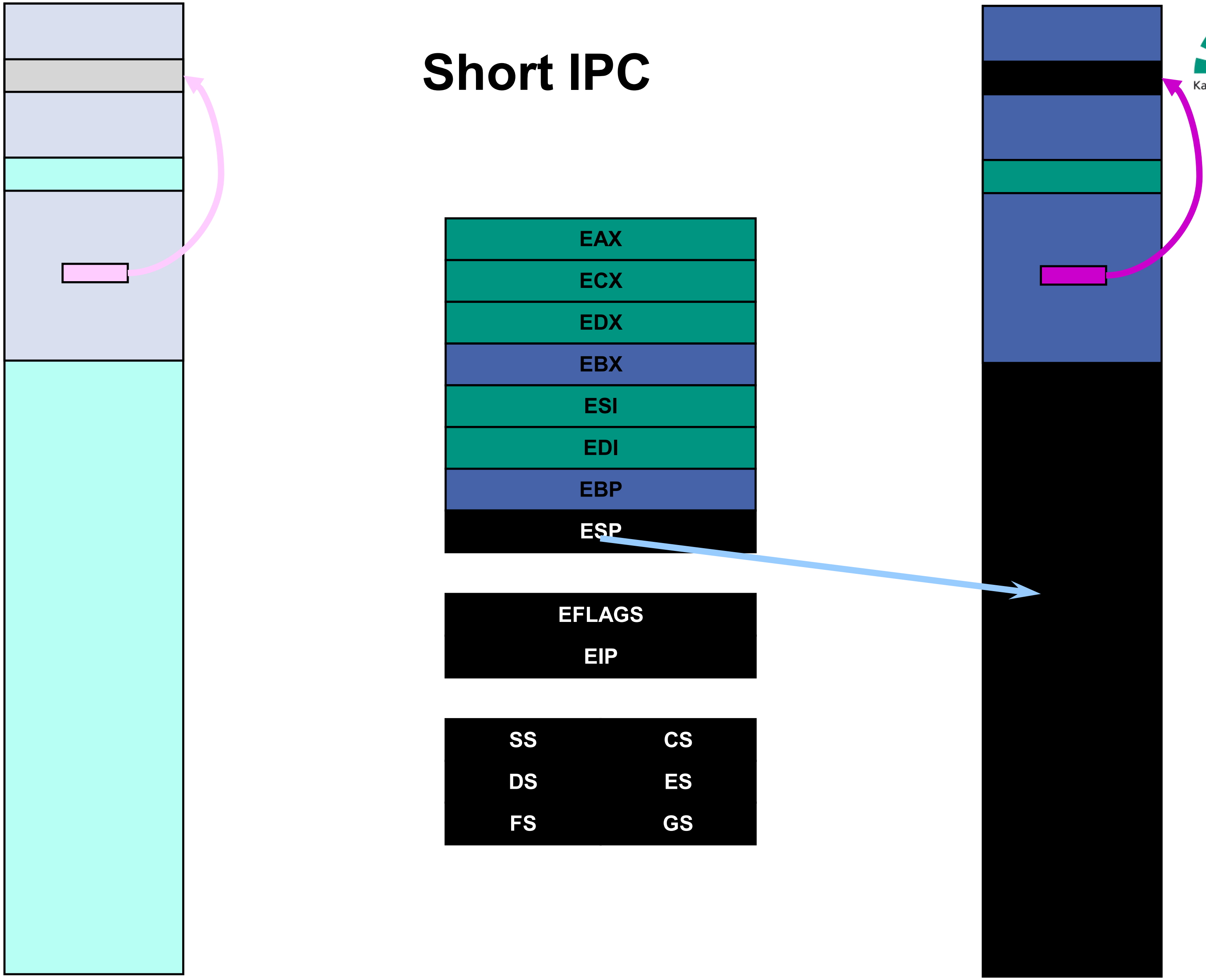
Short IPC



Short IPC

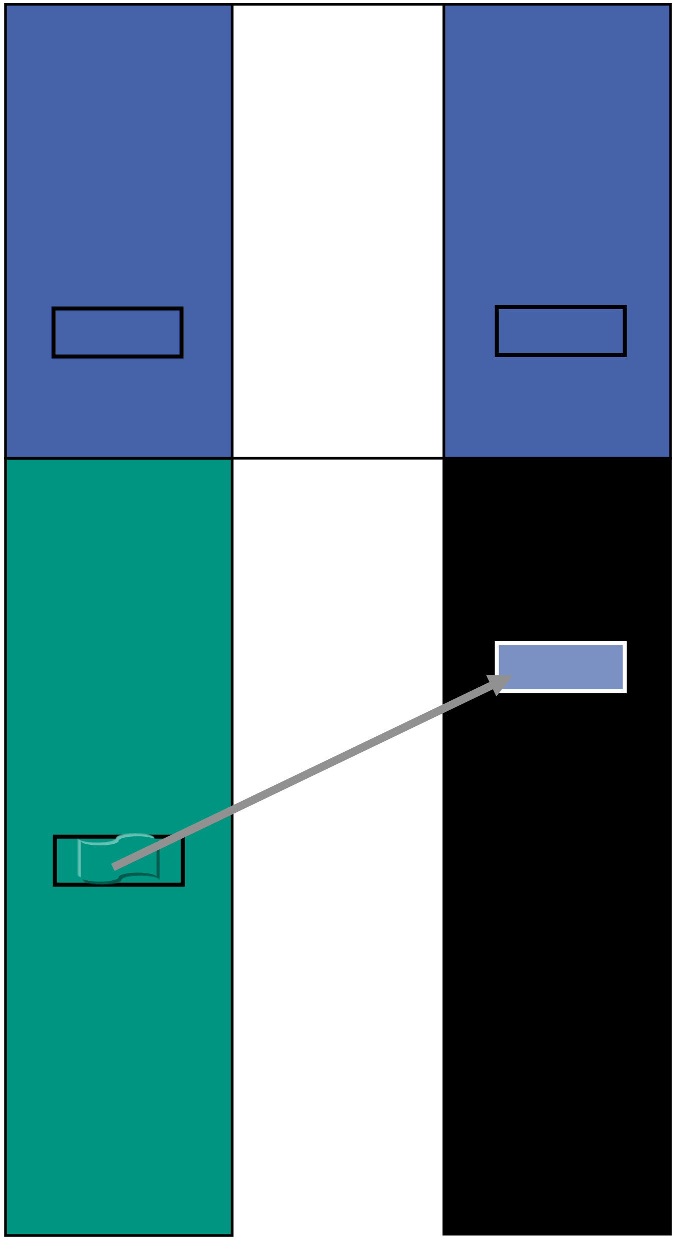


Short IPC



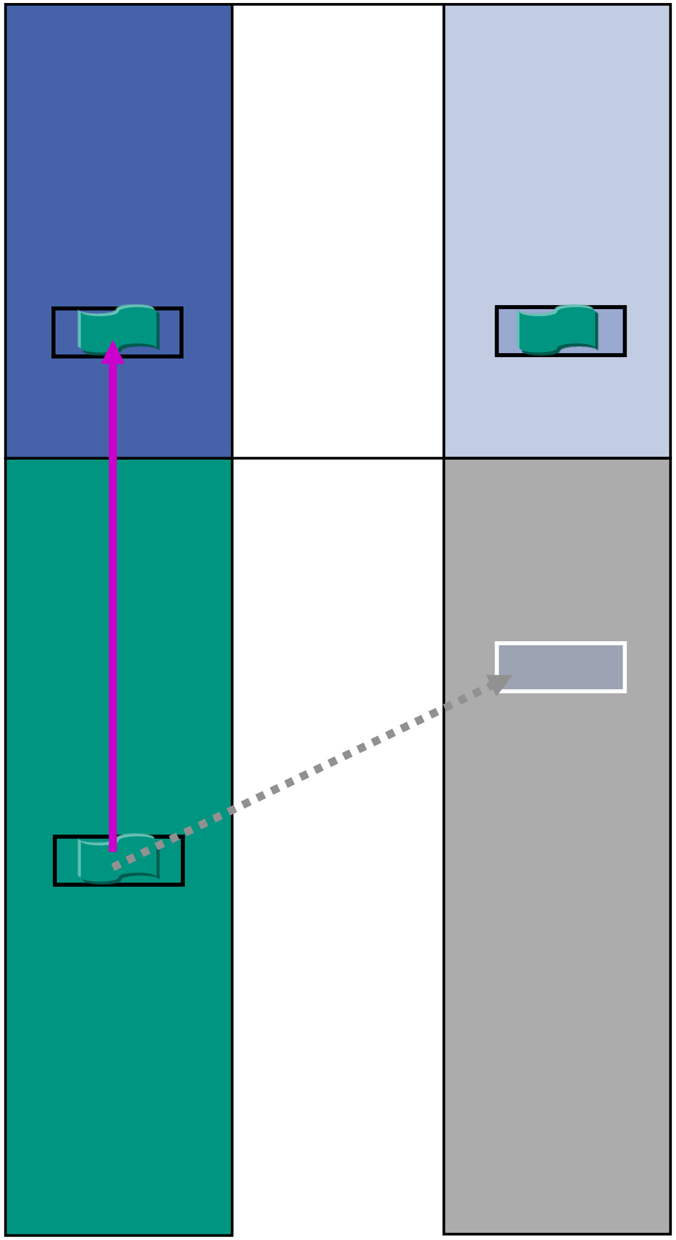
String IPC / memcpy

- Why?
  - Trust
  - Granularity
  - Synchronous ("atomic") transfer



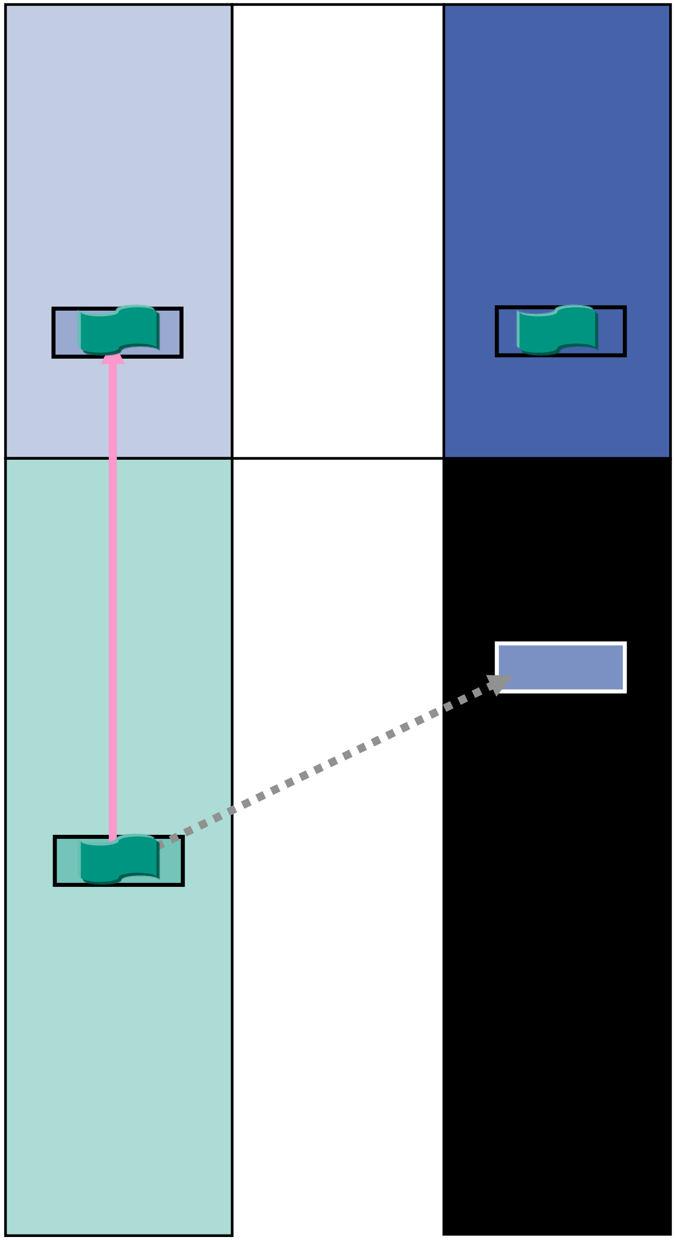
Copy In – Copy Out

- Copy into kernel buffer



Copy In – Copy Out

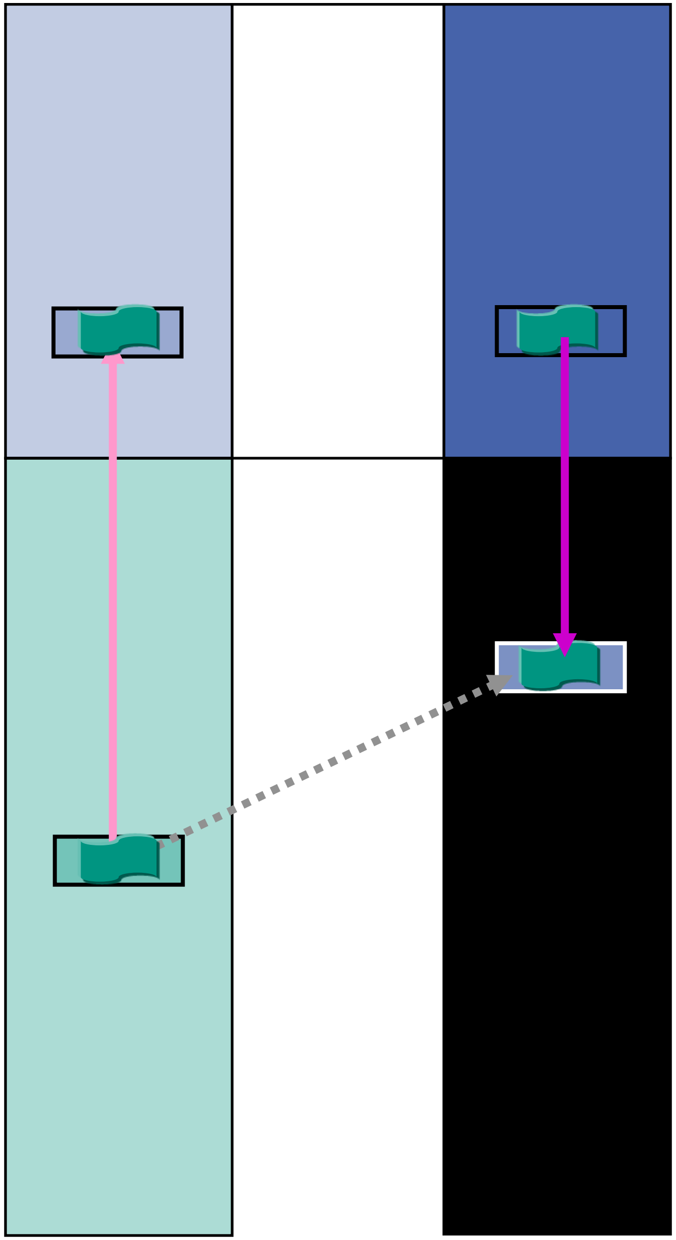
- Copy into kernel buffer
- Switch spaces



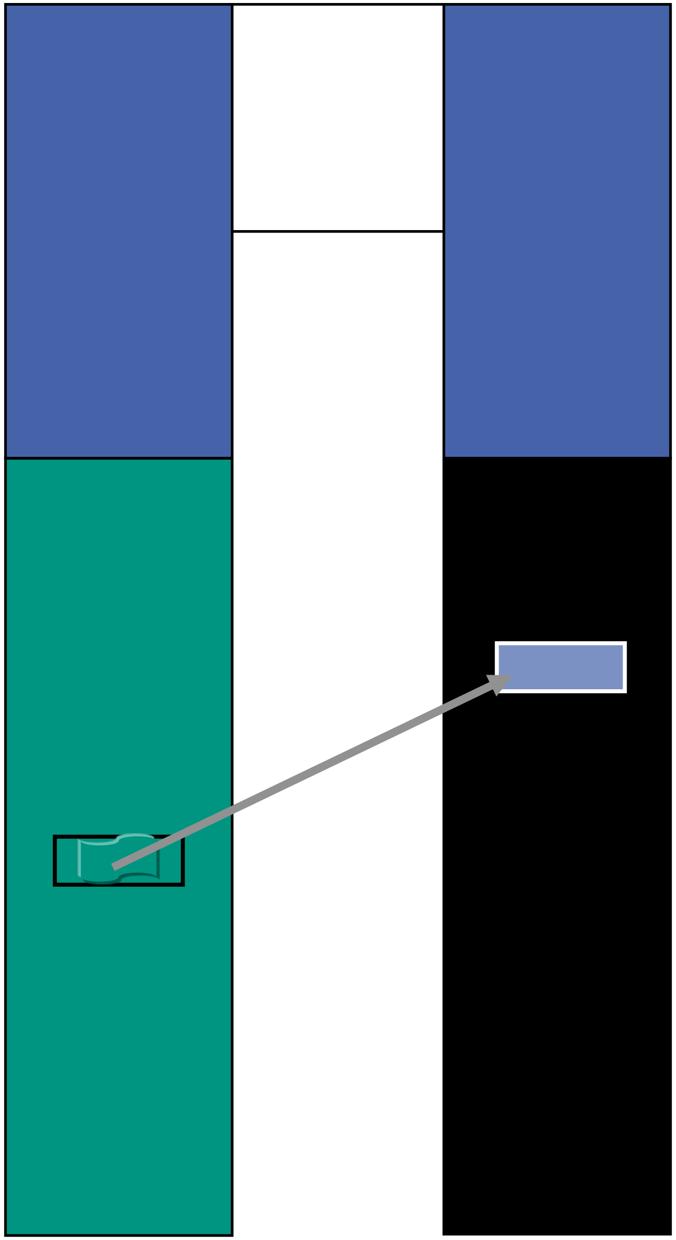
Copy In – Copy Out

- Copy into kernel buffer
- Switch spaces
- Copy out of kernel buffer

- Costs for *n* words
  - 2×2*n* r/w operations
  - Example: 8 words / cache
    - 3×*n*/8 cache lines
    - 1×*n*/8 cache misses (small *n*)
    - 4×*n*/8 cache misses (large *n*)

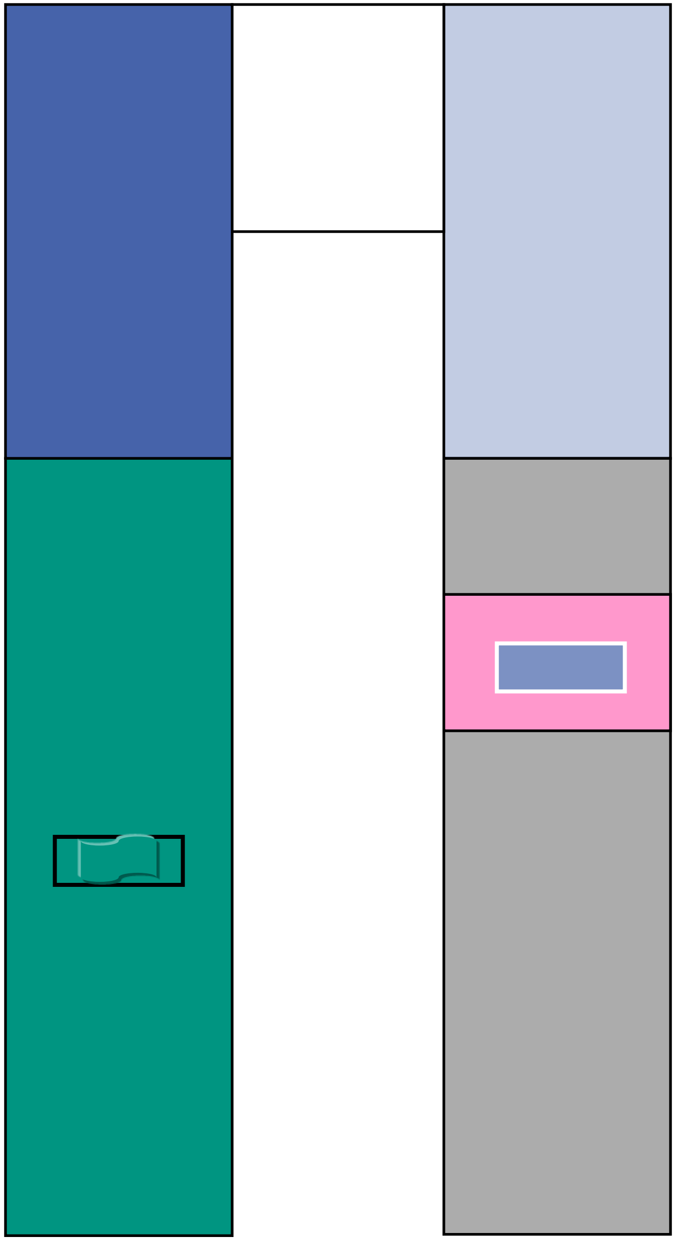


Temporary Mapping



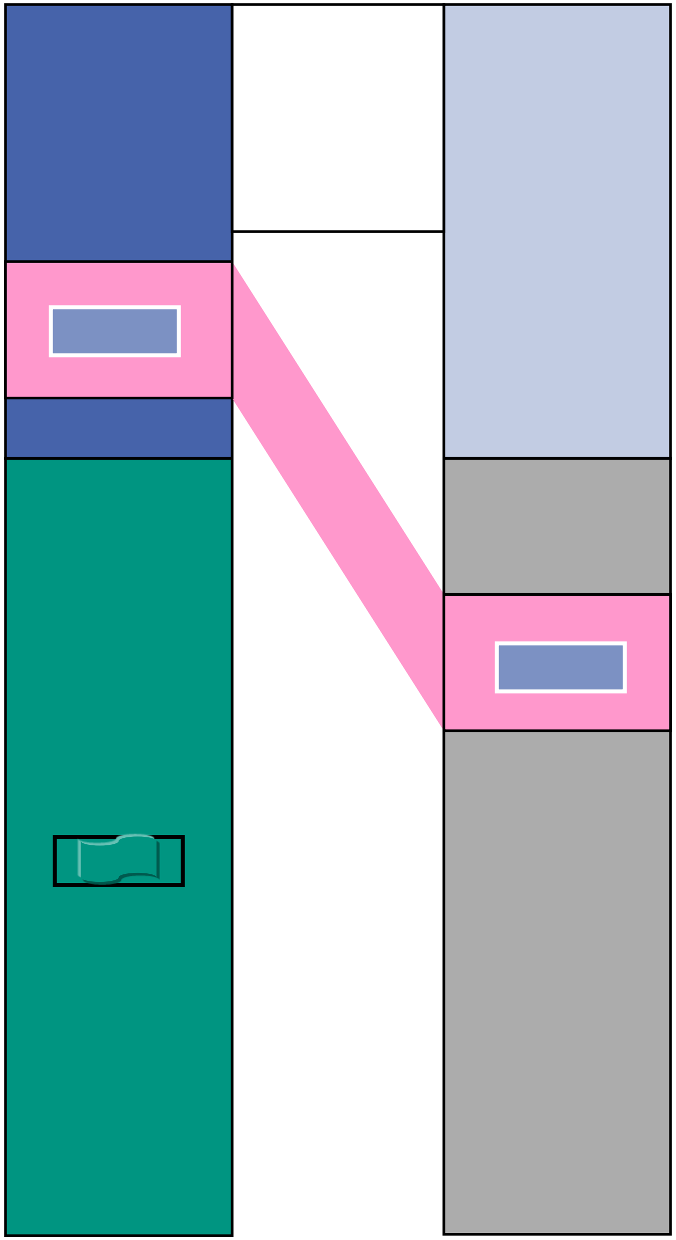
Temporary Mapping

- Select dest area (2x4 MB)



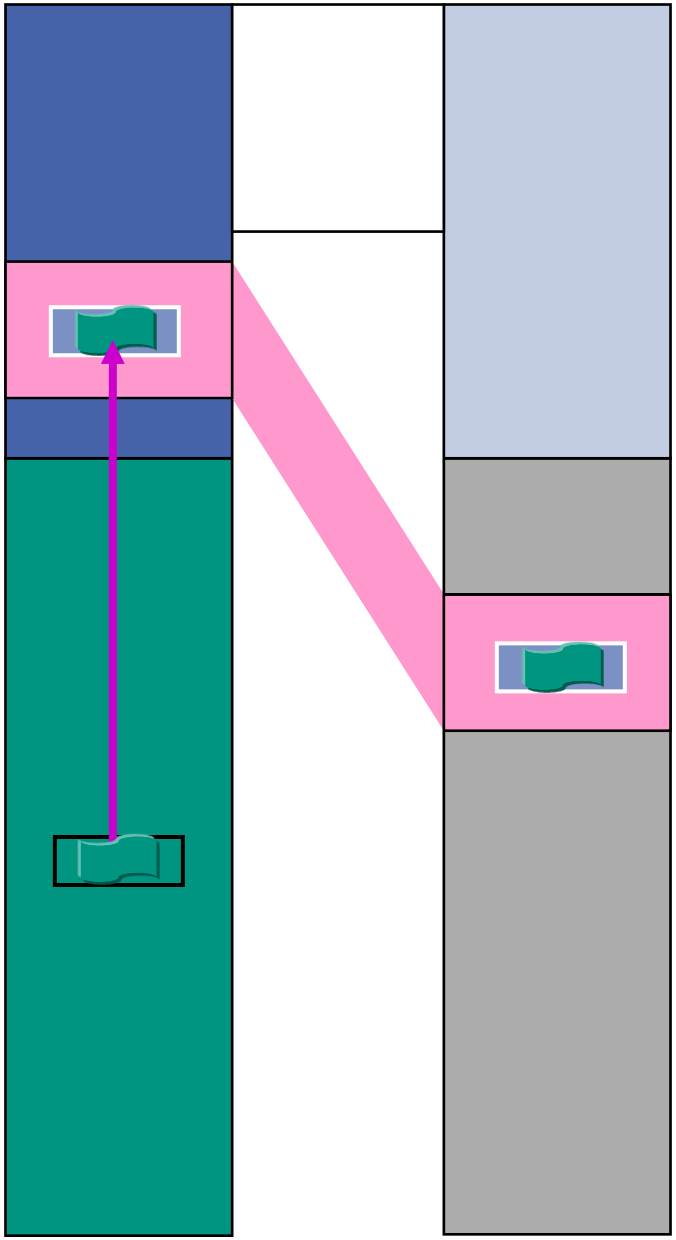
Temporary Mapping

- Select dest area (2x4 MB)
- Map into source AS (kernel)



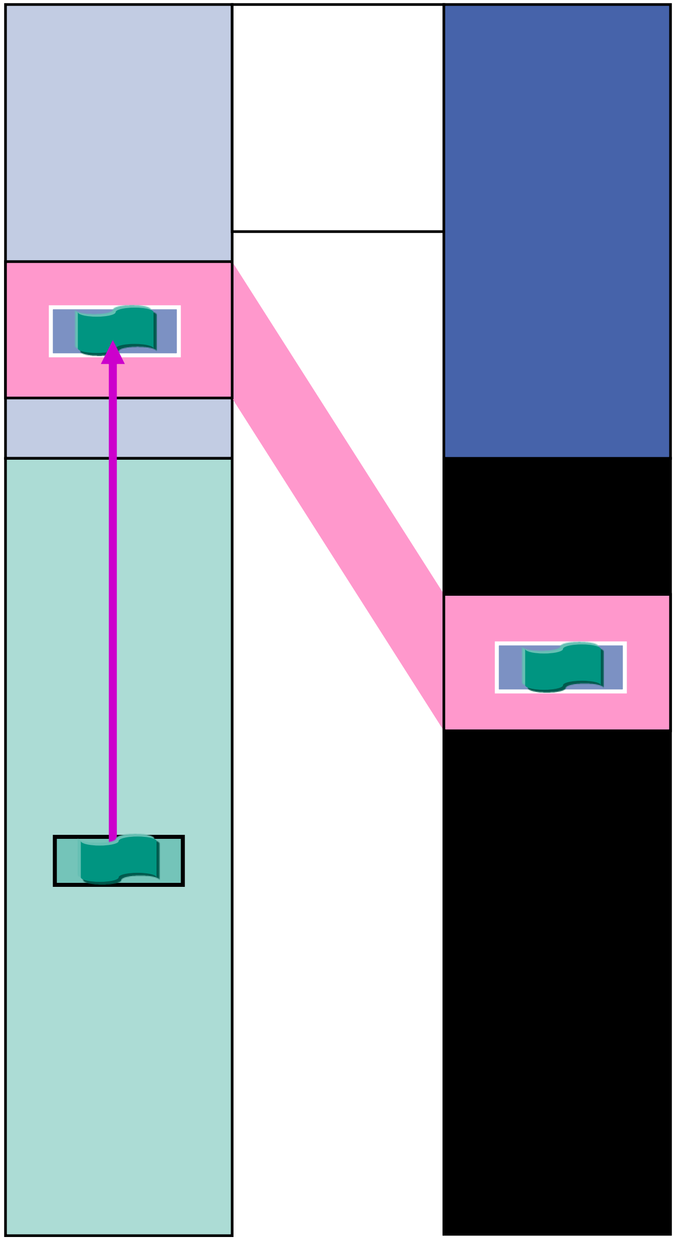
Temporary Mapping

- Select dest area (2x4 MB)
- Map into source AS (kernel)
- Copy data



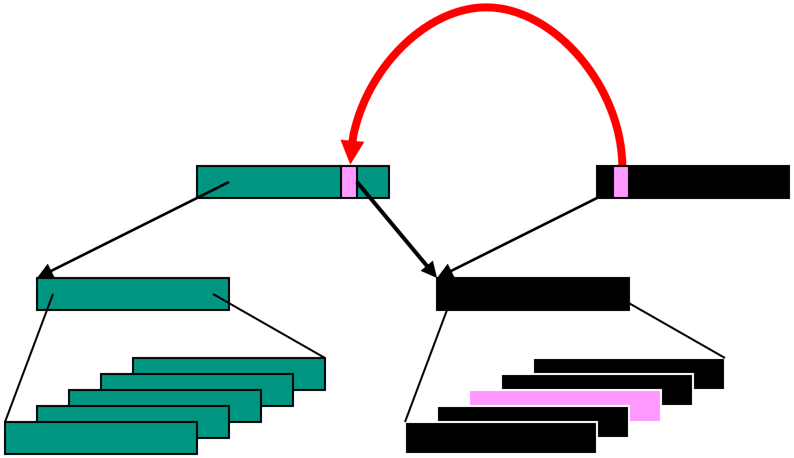
Temporary Mapping

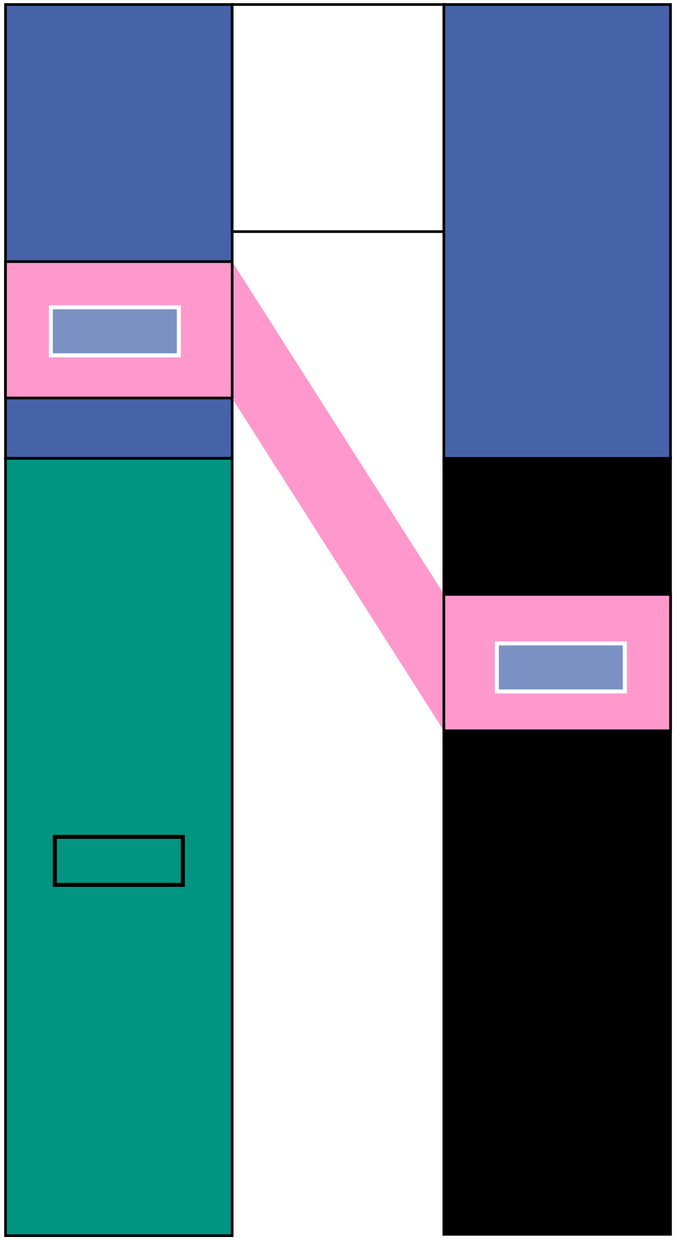
- Select dest area (2x4 MB)
- Map into source AS (kernel)
- Copy data
- Switch to dest space



Temporary Mapping

- Copy 2 page directory entries (PDEs) from dest
  - Addresses in temporary mapping area are resolved using dest's page *tables*



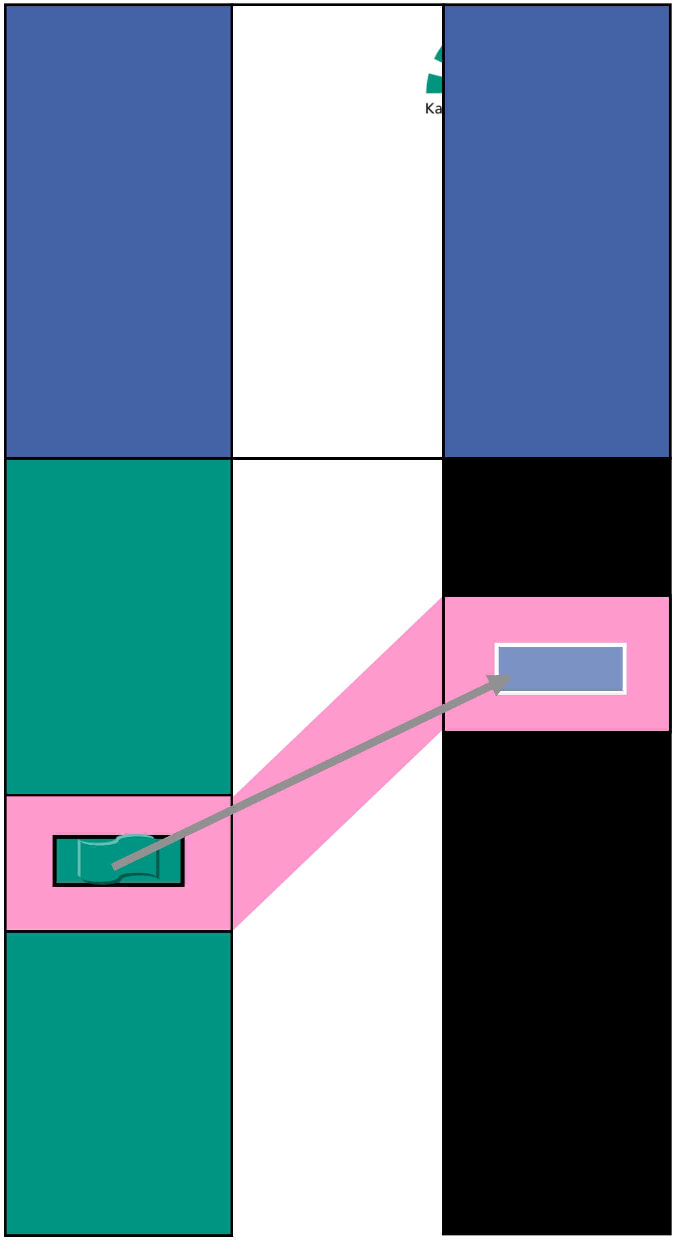


String IPC: Better than shared memory?

- Trust?
  - Grant items prevent unmapping
- Granularity?
  - Sender decides memory layout
- Synchronous (“atomic”) transfer?
  - Additional short IPC for signaling

- Tunneled page faults, copy area multiplexing
- Violates minimality

No string IPC in 3rd gen L4!

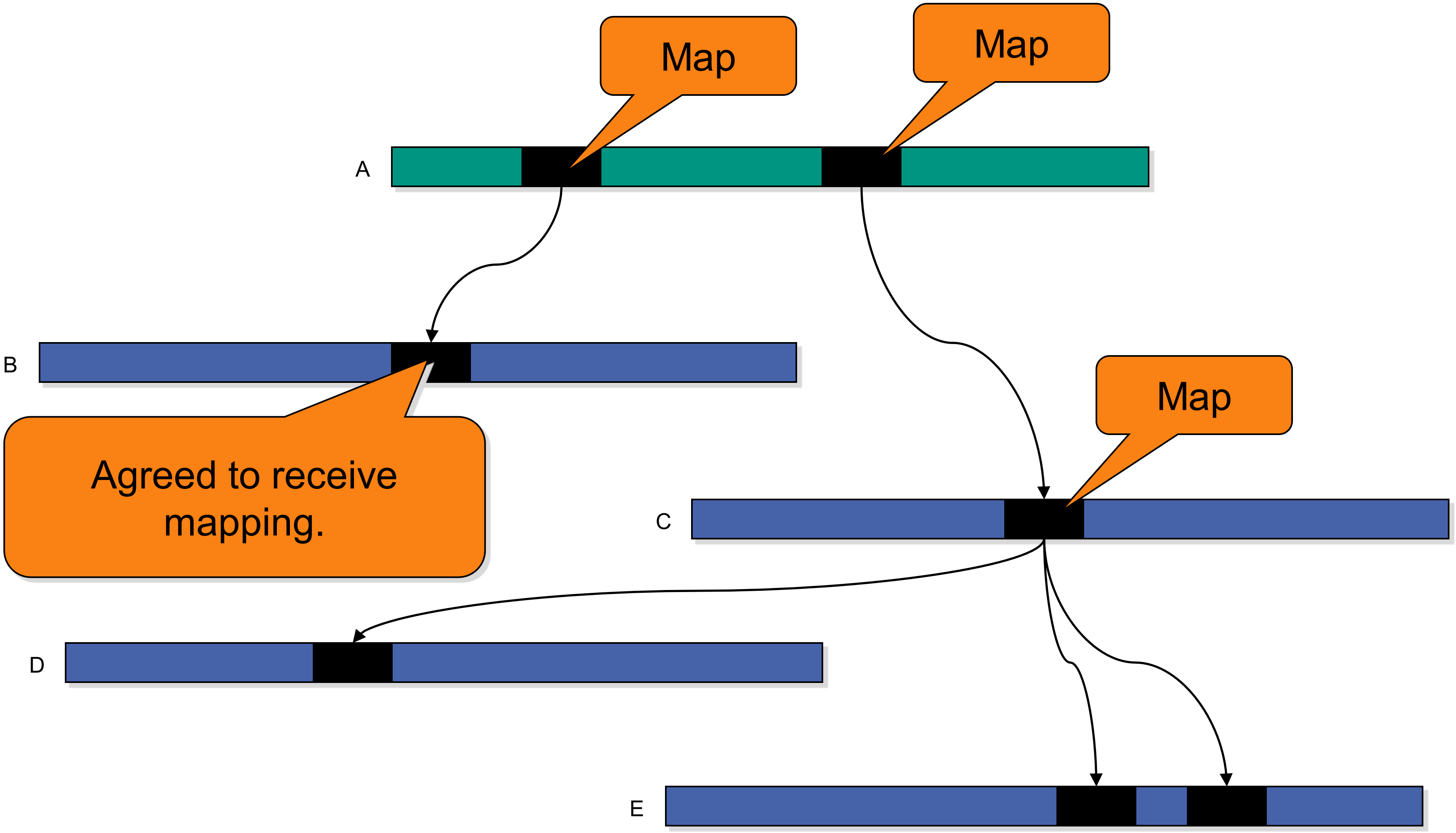


MAPPING

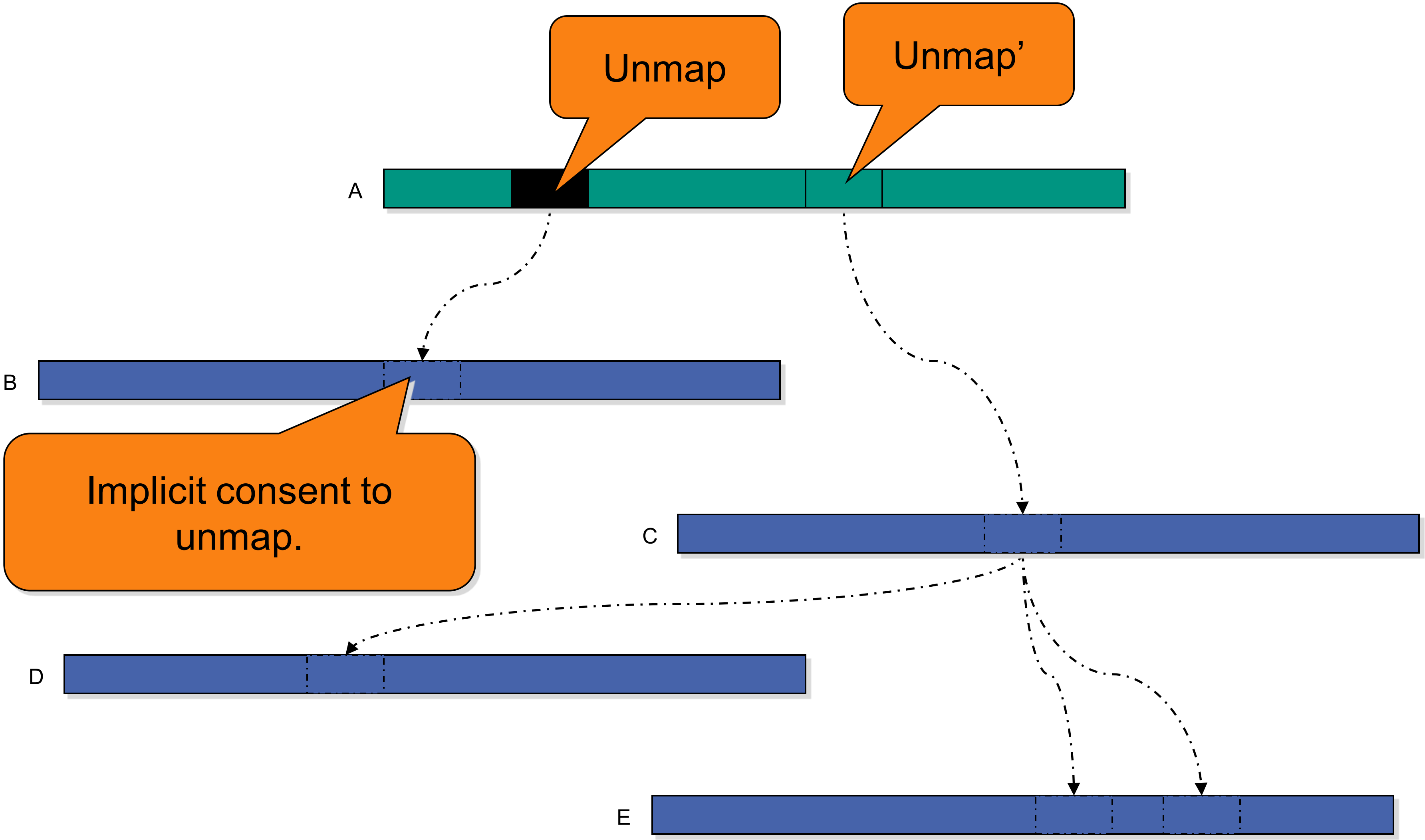
Mechanisms

- We need tools to build address spaces
  - Map
  - Unmap
- We need security
  - Access permissions [rwx]
- We need resource control
  - Page fault messages [detect page use]

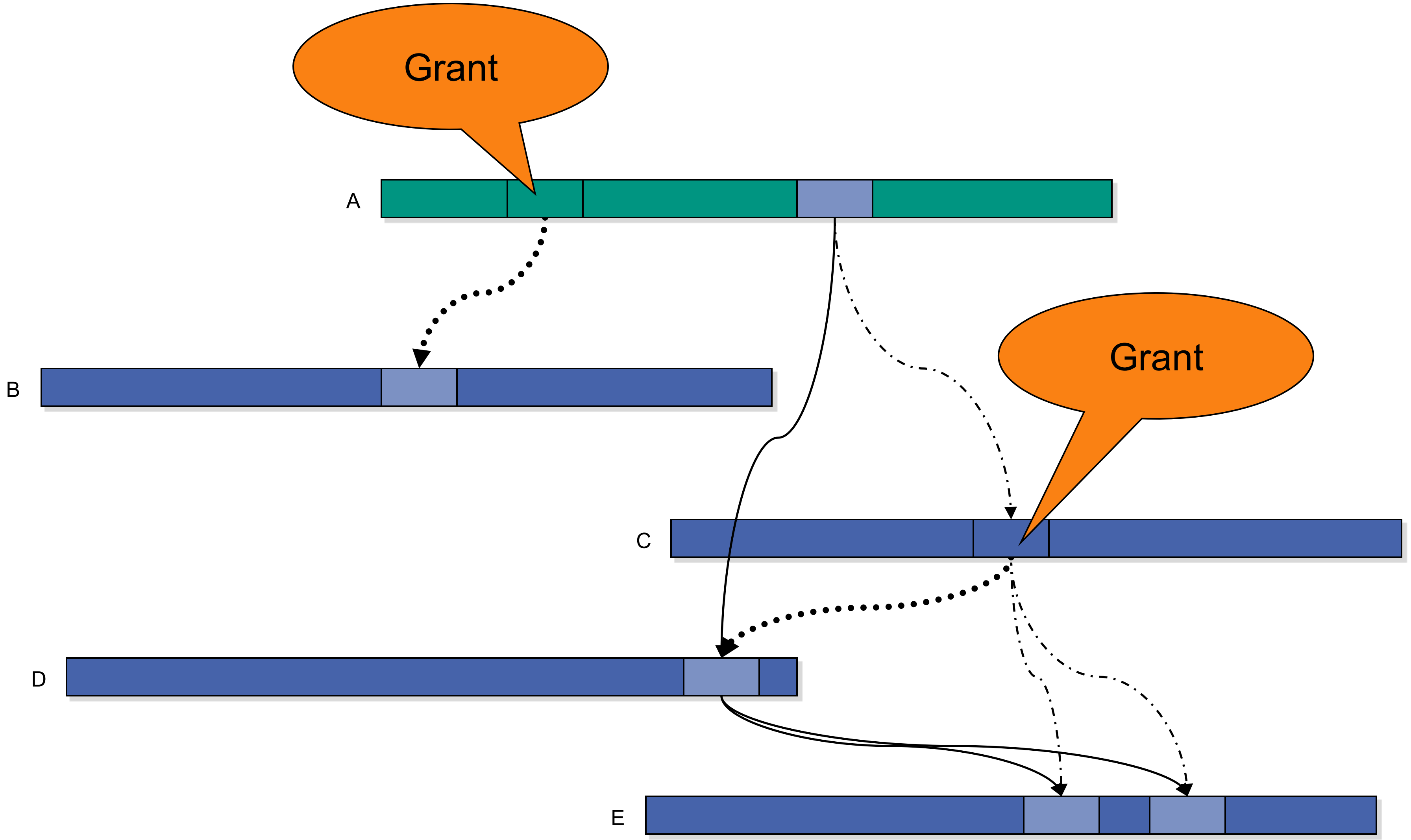
Map



Unmap

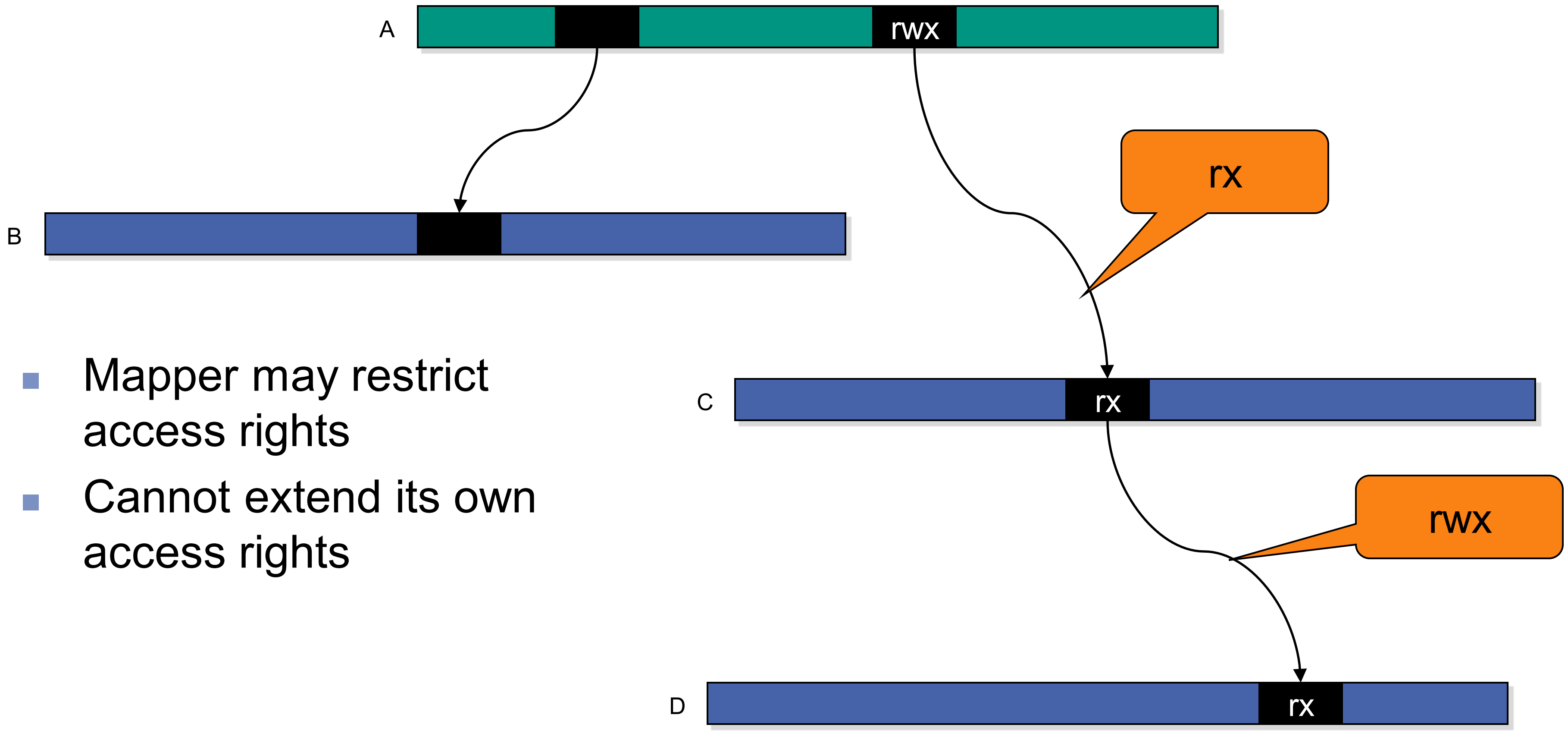


Grant



Access Rights – Map

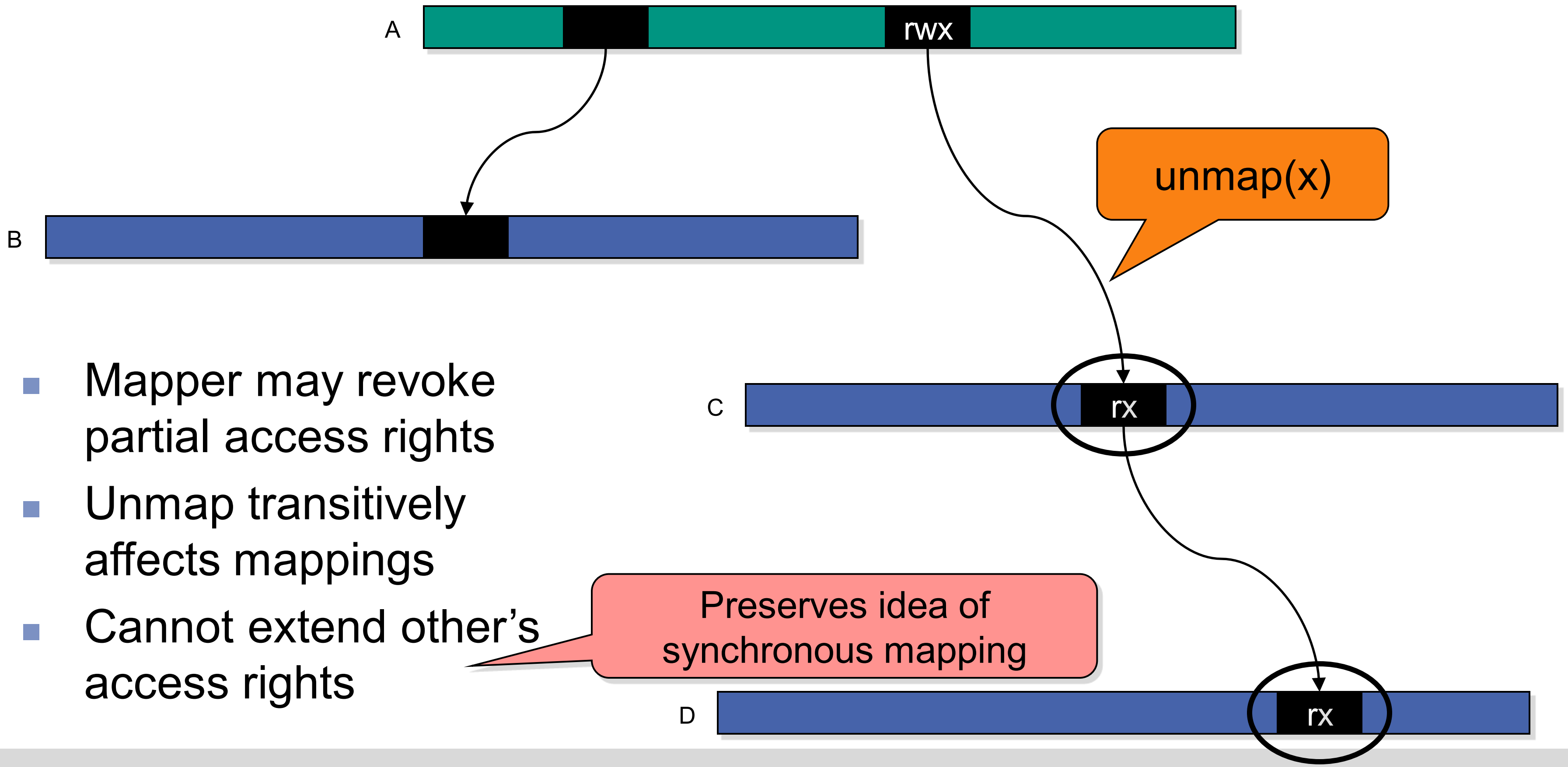
r = Read
w = Write
x = eXecute



- Mapper may restrict access rights
- Cannot extend its own access rights

Access Rights – Unmap

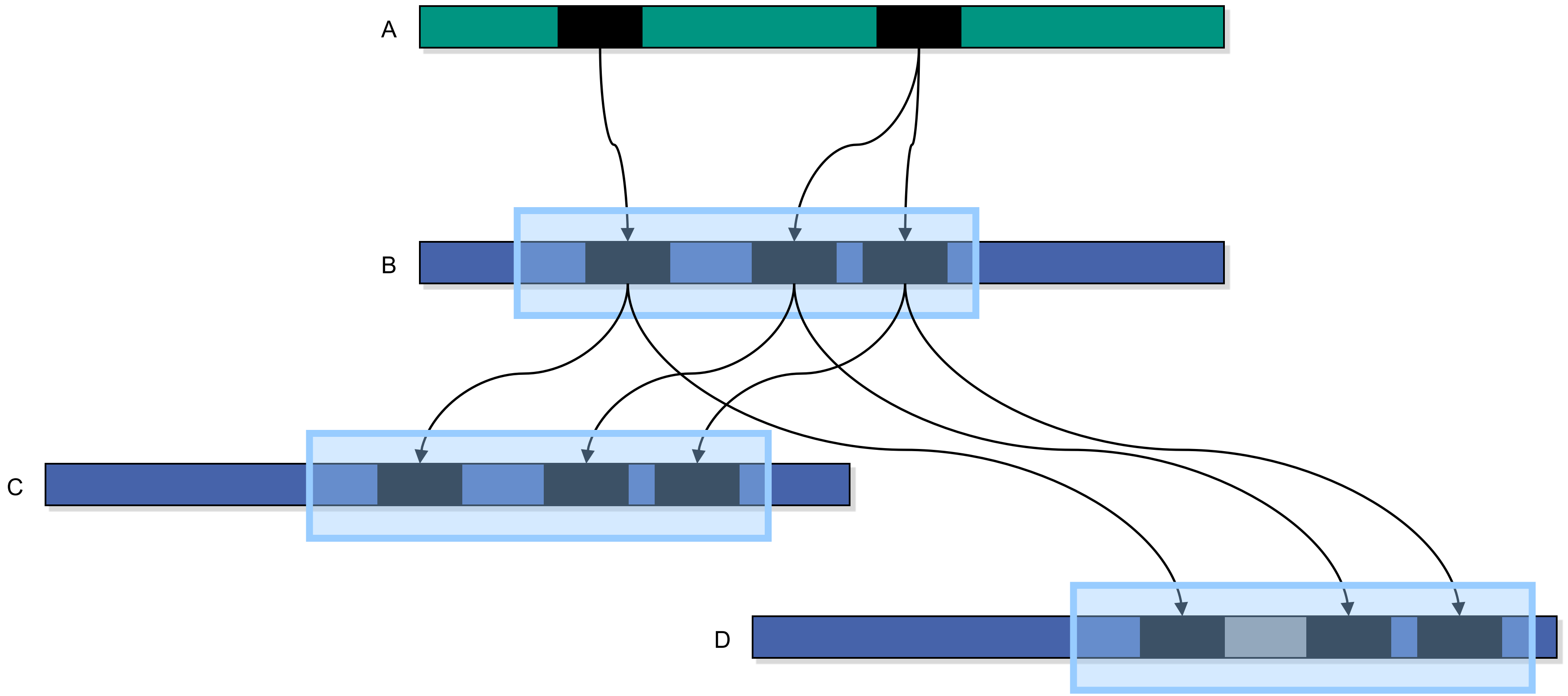
r = Read
w = Write
x = eXecute



- Mapper may revoke partial access rights
- Unmap transitively affects mappings
- Cannot extend other's access rights

Preserves idea of synchronous mapping

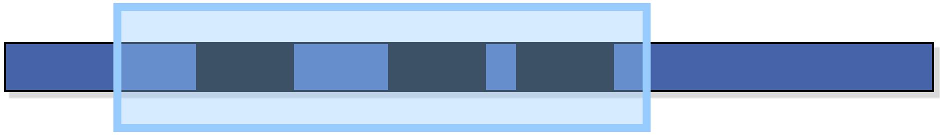
Mapping Regions



Mapping Regions: Flex Pages

- Abstraction: flex page
  - Contiguous regions of virtual address space
    - Sparse physical mappings possible
  - Called `fpage`
  - Abstracts architecture's page sizes

- Fpage semantics
  - Inseparable object
  - Aligned to its size
  - Size is power of 2, min. $4096=2^{12}$ byte



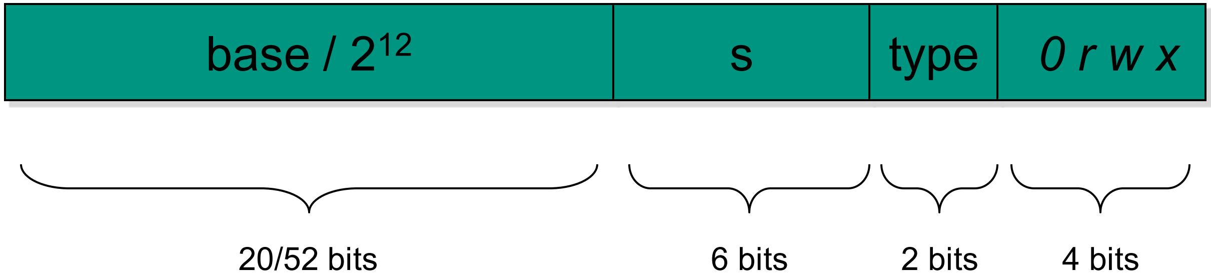
Fpage Encoding

fpage(base, size=2^s)
$s \geq 12$
base mod $2^s = 0$

Type

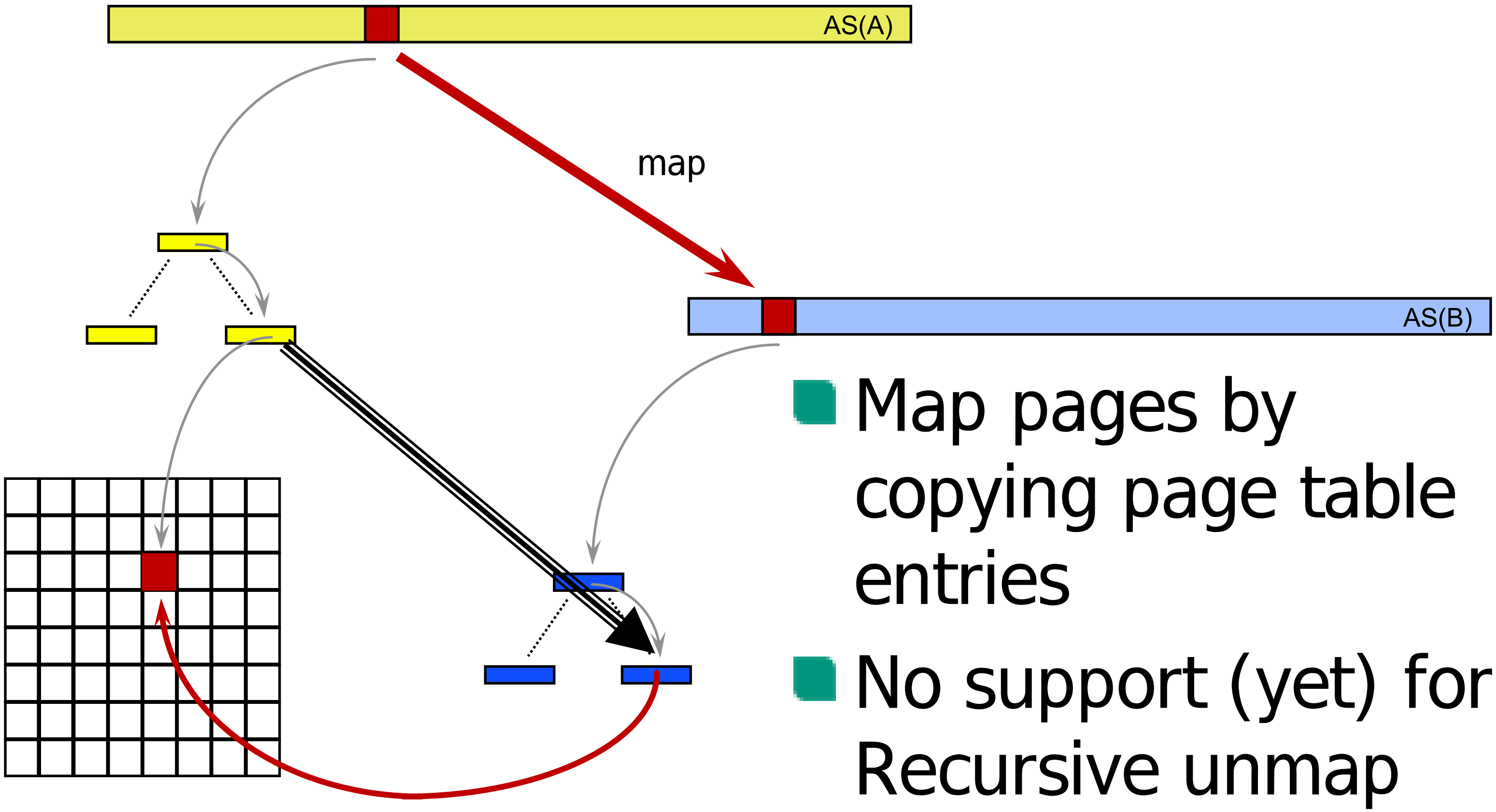
```
L4_FPAGE_SPECIAL = 0,
L4_FPAGE_MEMORY  = 1,
L4_FPAGE_IO      = 2,
L4_FPAGE_OBJ     = 3, //capability
```

- Special cases
  - Complete address space (base=0, s=1)
  - Nothing: nilpage (0)



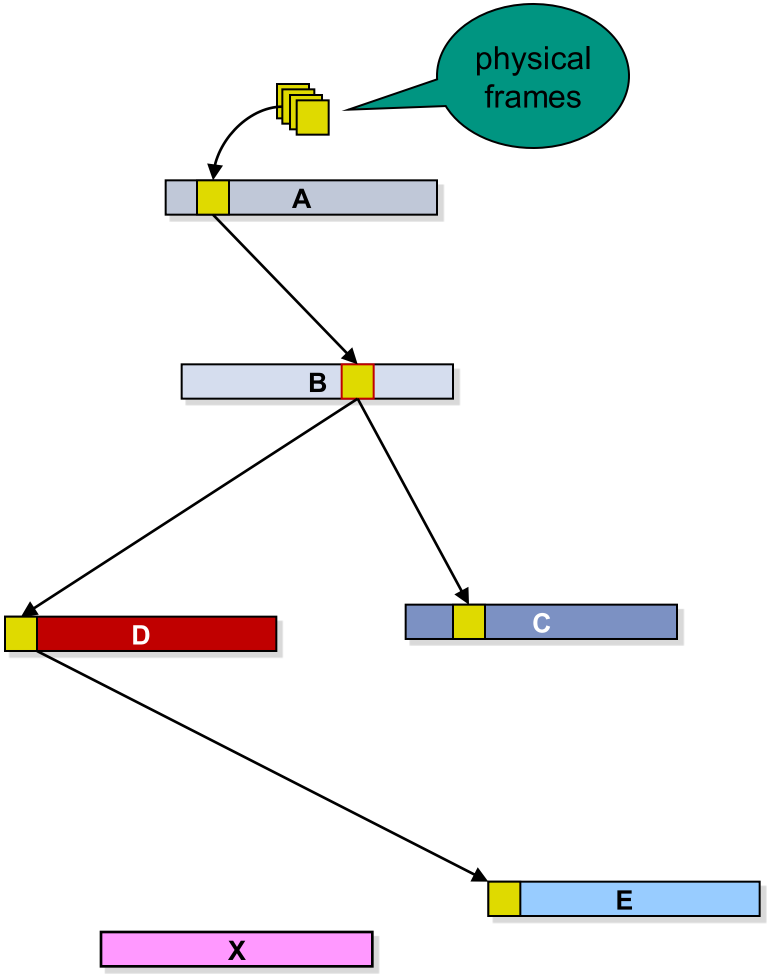
base / 2^{12}	s	type	*0 r w x*
20/52 bits	6 bits	2 bits	4 bits

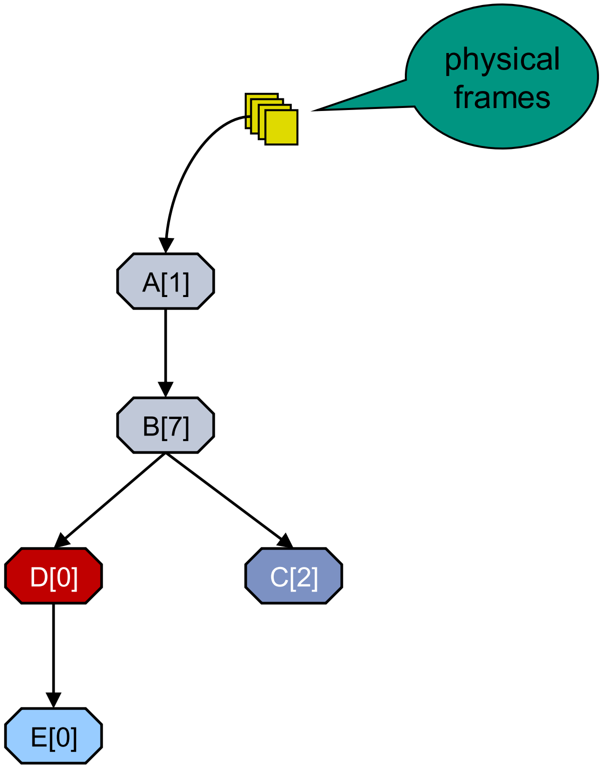
Mapping Pages



- Map pages by copying page table entries
- No support (yet) for Recursive unmap

Mapping Database

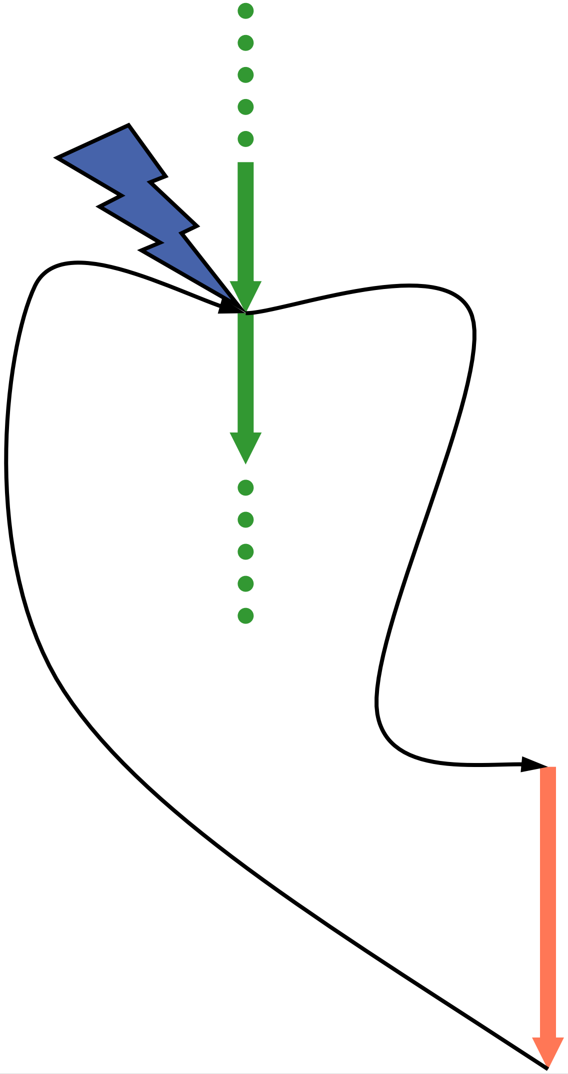




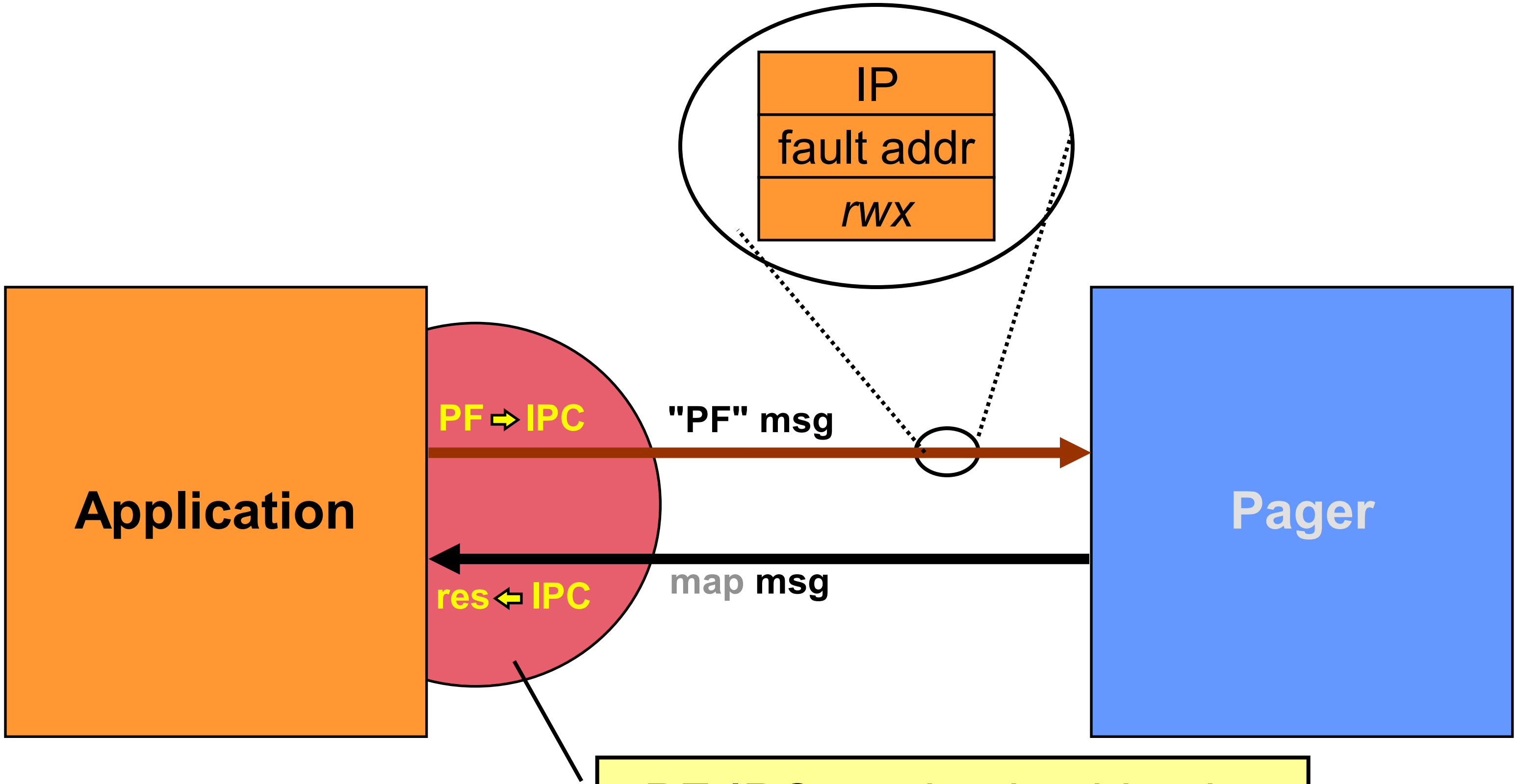
INTERRUPTS + EXCEPTIONS

Event Handling

1. Program executes happily
2. Event occurs
3. Activate event handler
   - Save current state
   - Switch to privileged mode
   - Execute event handler
4. Fix the problem / handle event
5. End of event handling
   - Restore state
   - Switch to previous mode
   - Continue interrupted program
6. Program executes happily again



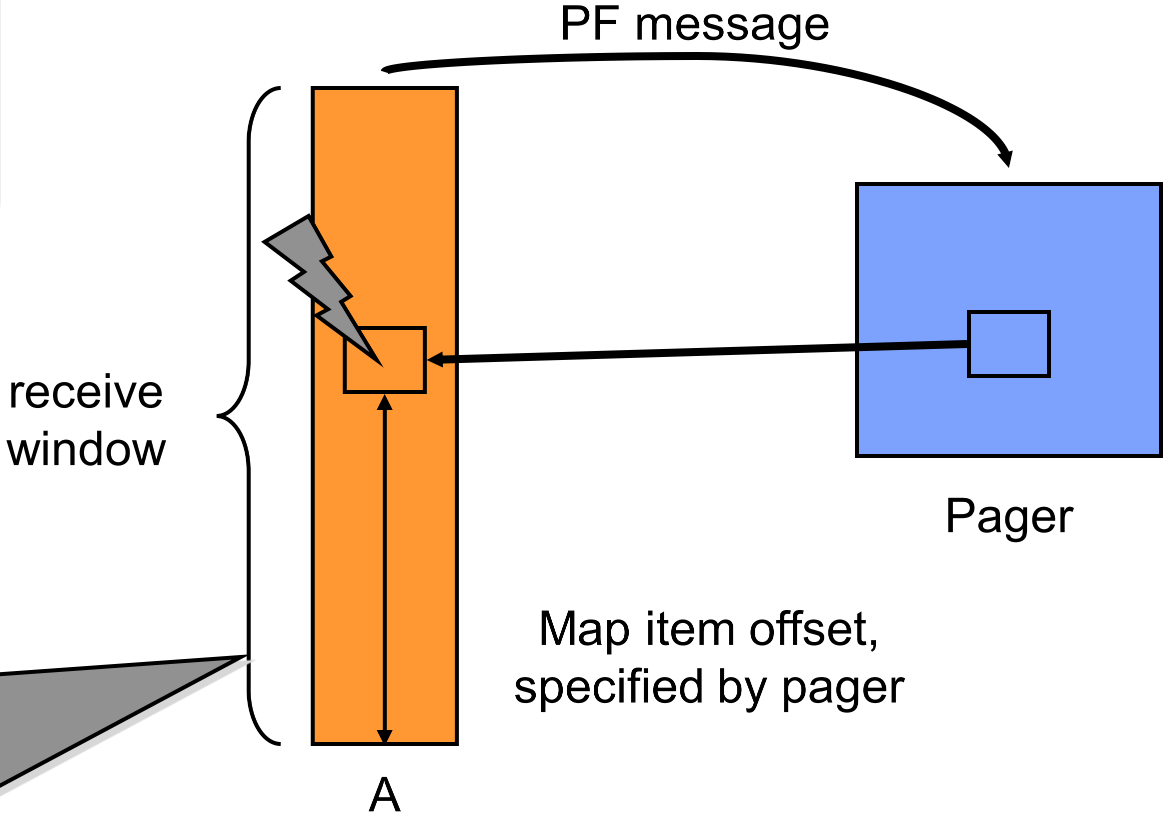
Page Fault IPC



PF-IPC synthesized by the kernel, pager's reply caught by the kernel (application is not informed/involved)

Page Fault Receive Window

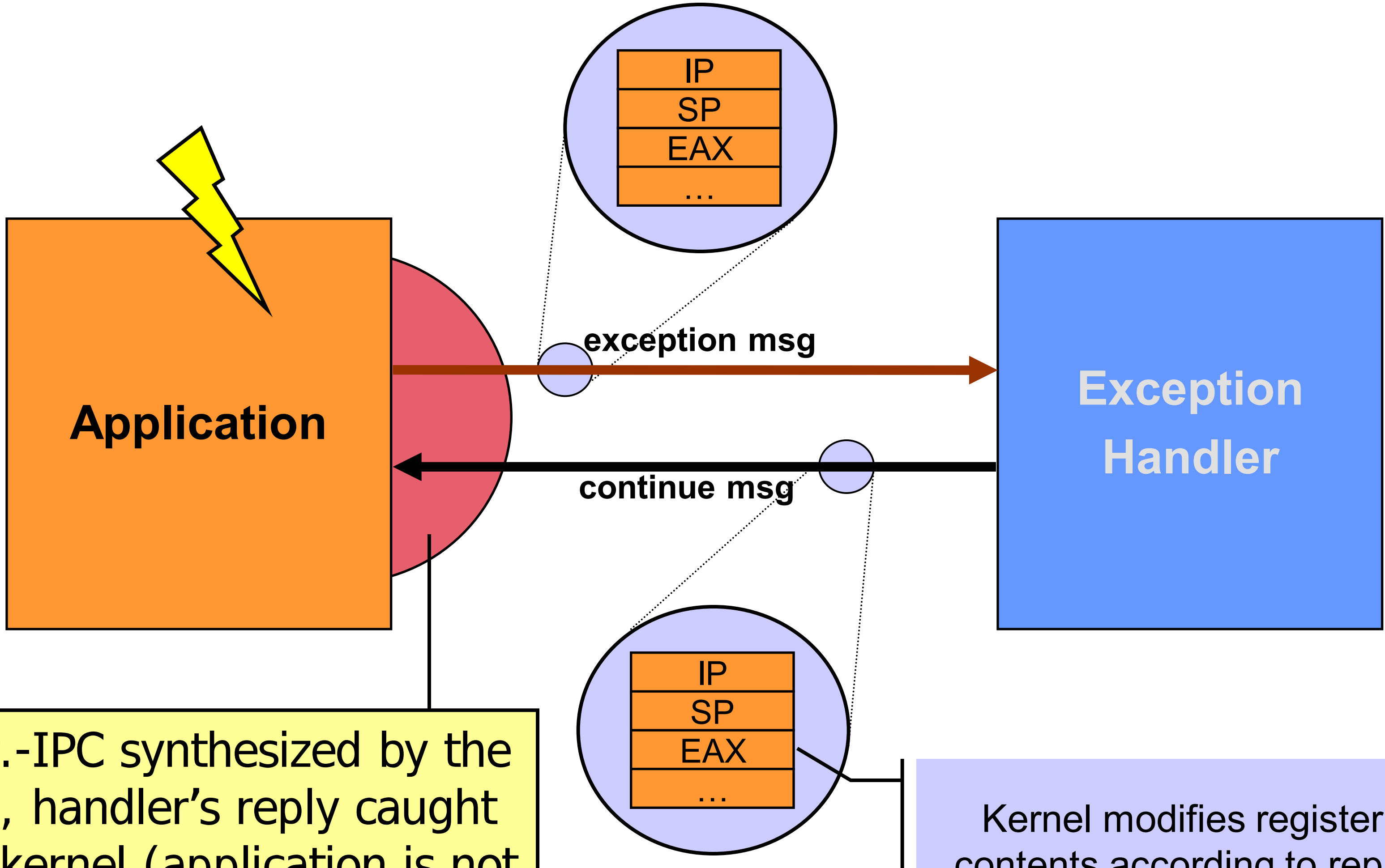
Configured by **kernel**



Pager can overmap entire address space.

Liedtke: “The SawMill Framework for Virtual Memory Diversity”

New Exception Handling Model



Except.-IPC synthesized by the kernel, handler’s reply caught by the kernel (application is not informed/involved).

Kernel modifies register contents according to reply message

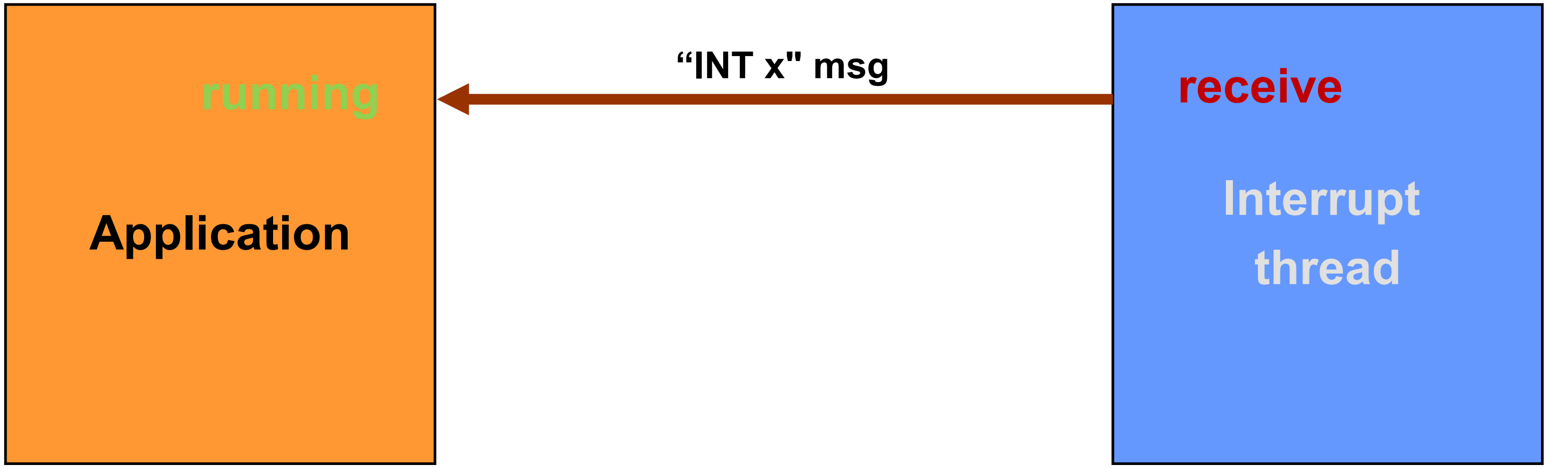
Synchronous vs. asynchronous interrupt IPC

receive

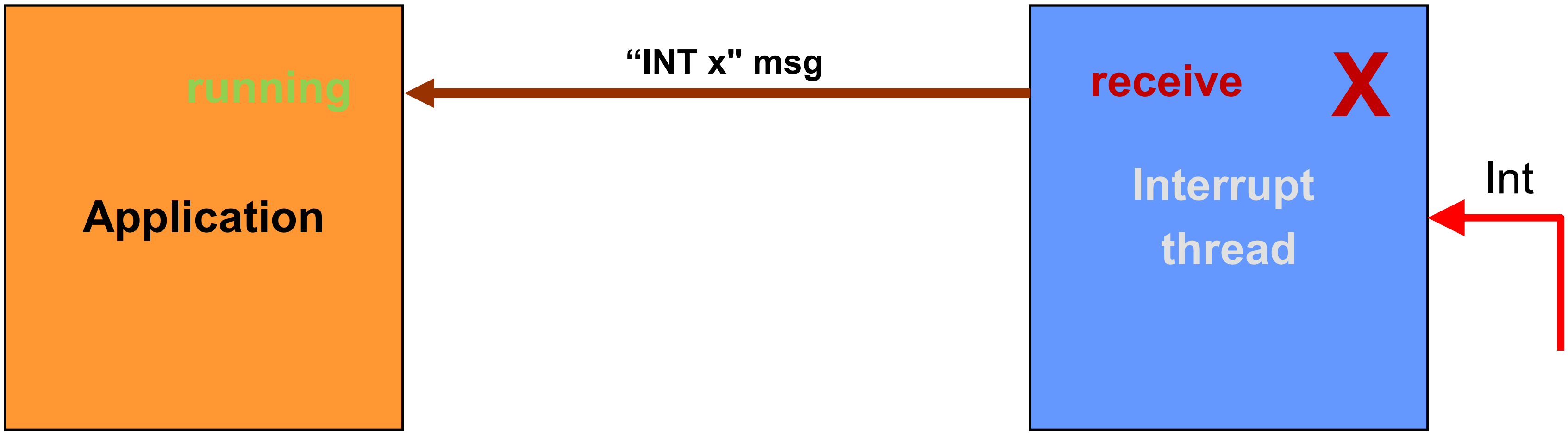
Application

Interrupt thread

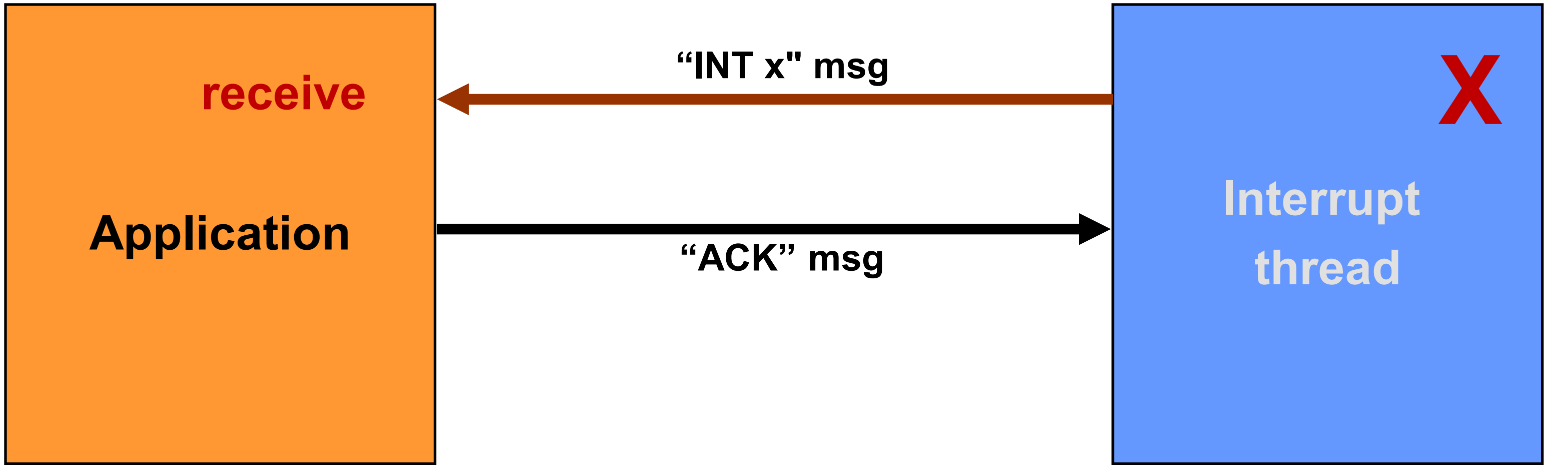
Synchronous vs. asynchronous interrupt IPC



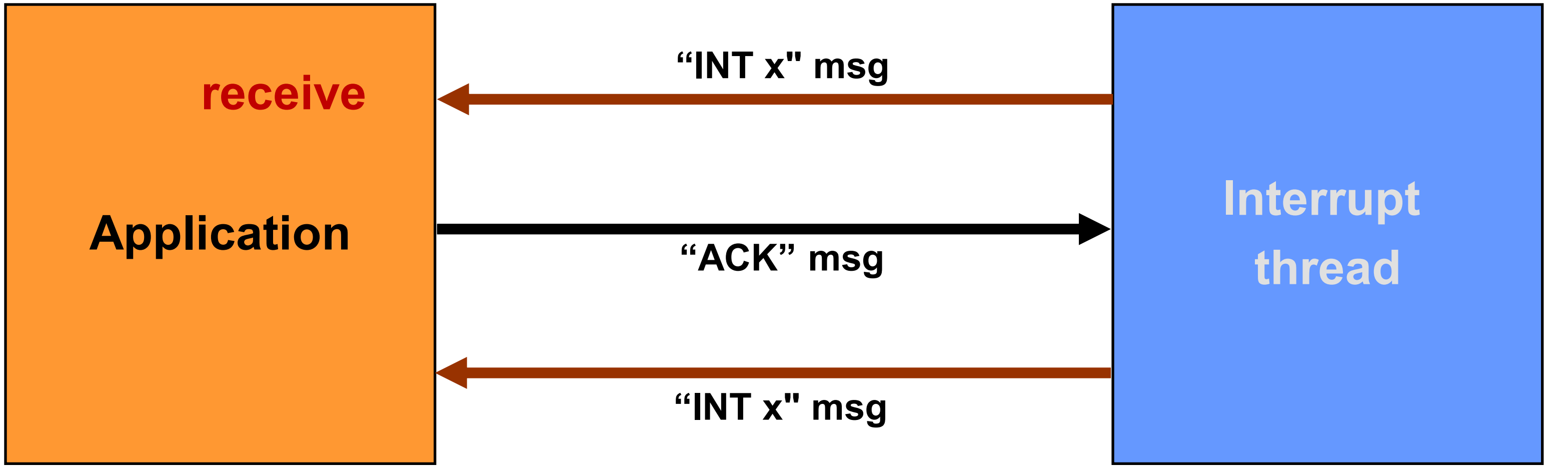
Synchronous vs. asynchronous interrupt IPC



Synchronous vs. asynchronous interrupt IPC



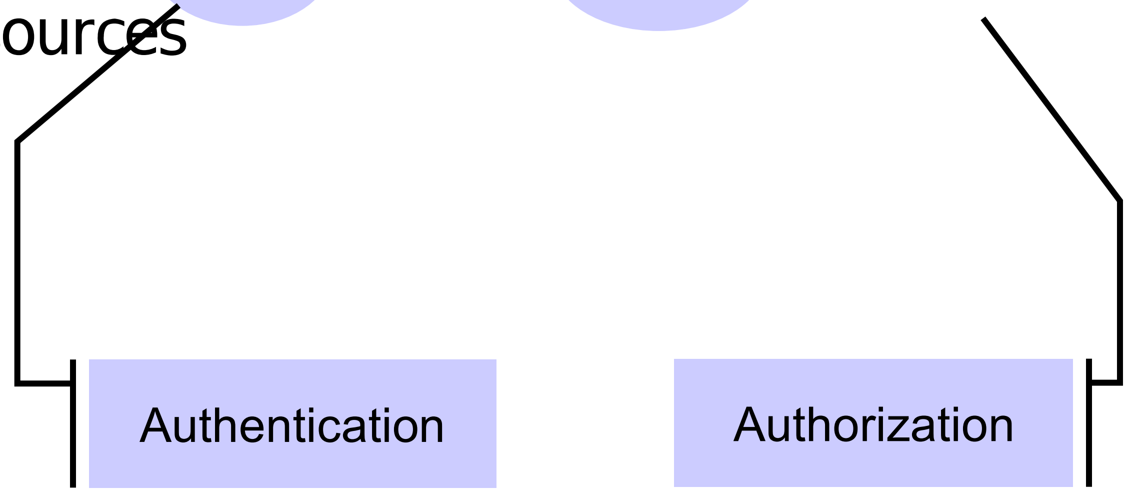
Synchronous vs. asynchronous interrupt IPC



SECURITY

Security Policy

- Specifies who has what type of access to which resources



Authentication

Authorization

Authentication

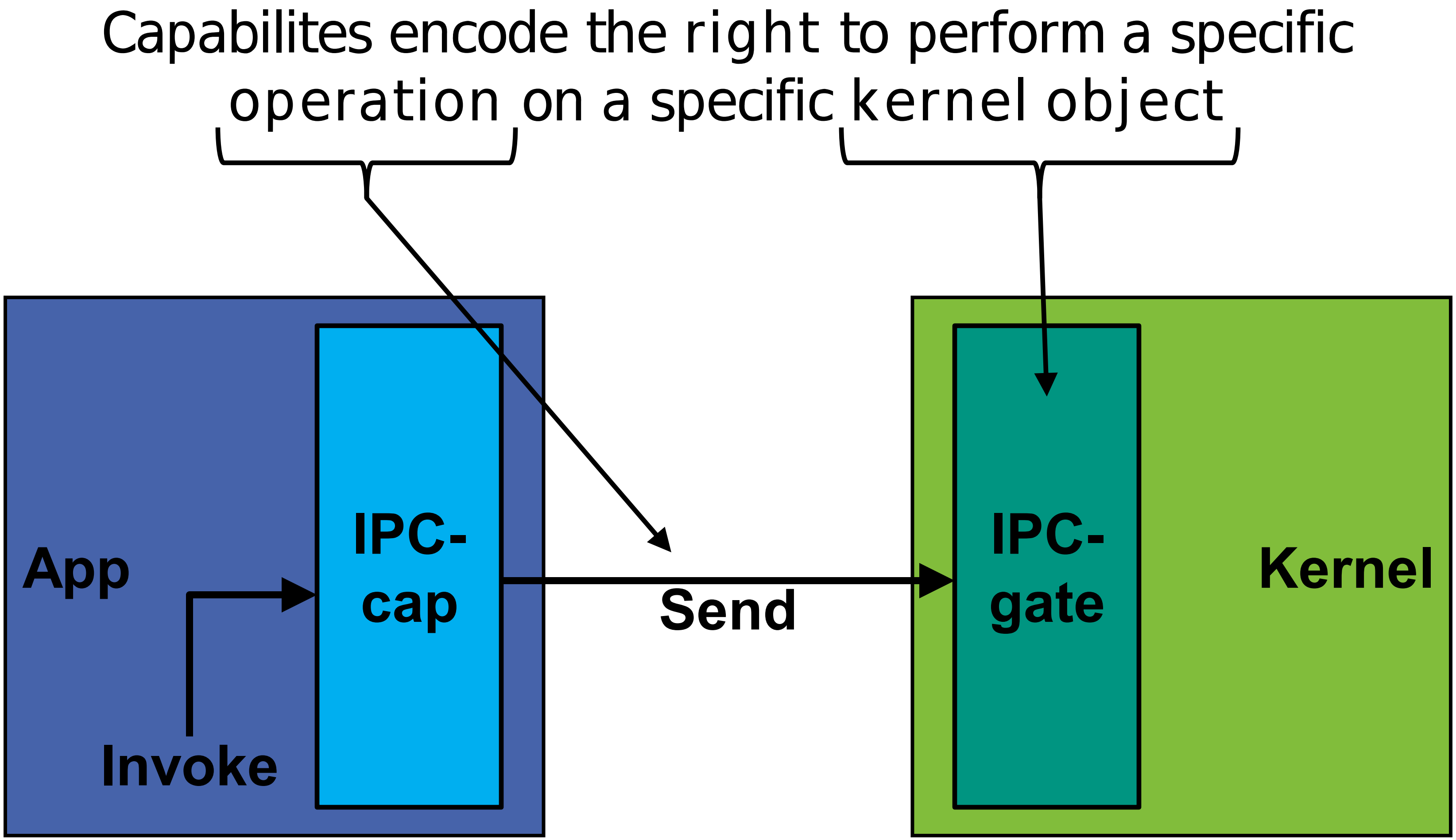
- Unforgeable endpoint identifiers
  - Thread ID of sender returned by kernel
  - Capabilities generated by kernel
  - Thread identifiers can be mapped to
    - Tasks
    - Users
    - Groups
    - Machines
    - Domains
  - Authentication is outside the microkernel – any policy can be implemented

Authorization

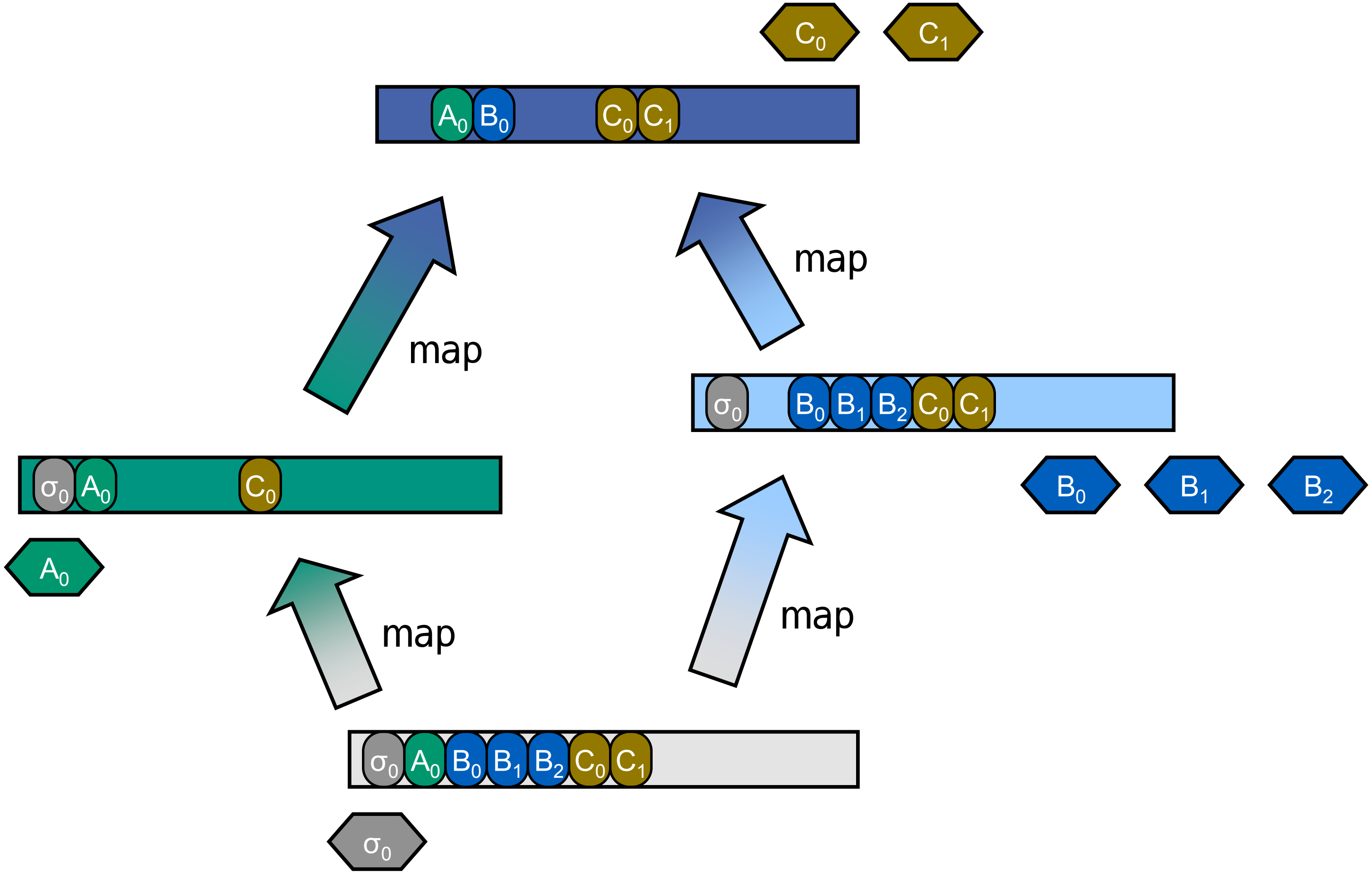
- Servers implement objects; clients access objects via IPC
- Servers receive unforgeable client identities from the IPC mechanism
  - Servers can implement arbitrary access control policy
- No special mechanisms needed in the microkernel

Is this really true???

Capabilites


Capabilites encode the right to perform a specific operation on a specific kernel object
App
IPC-cap
Invoke
Send
IPC-gate
Kernel

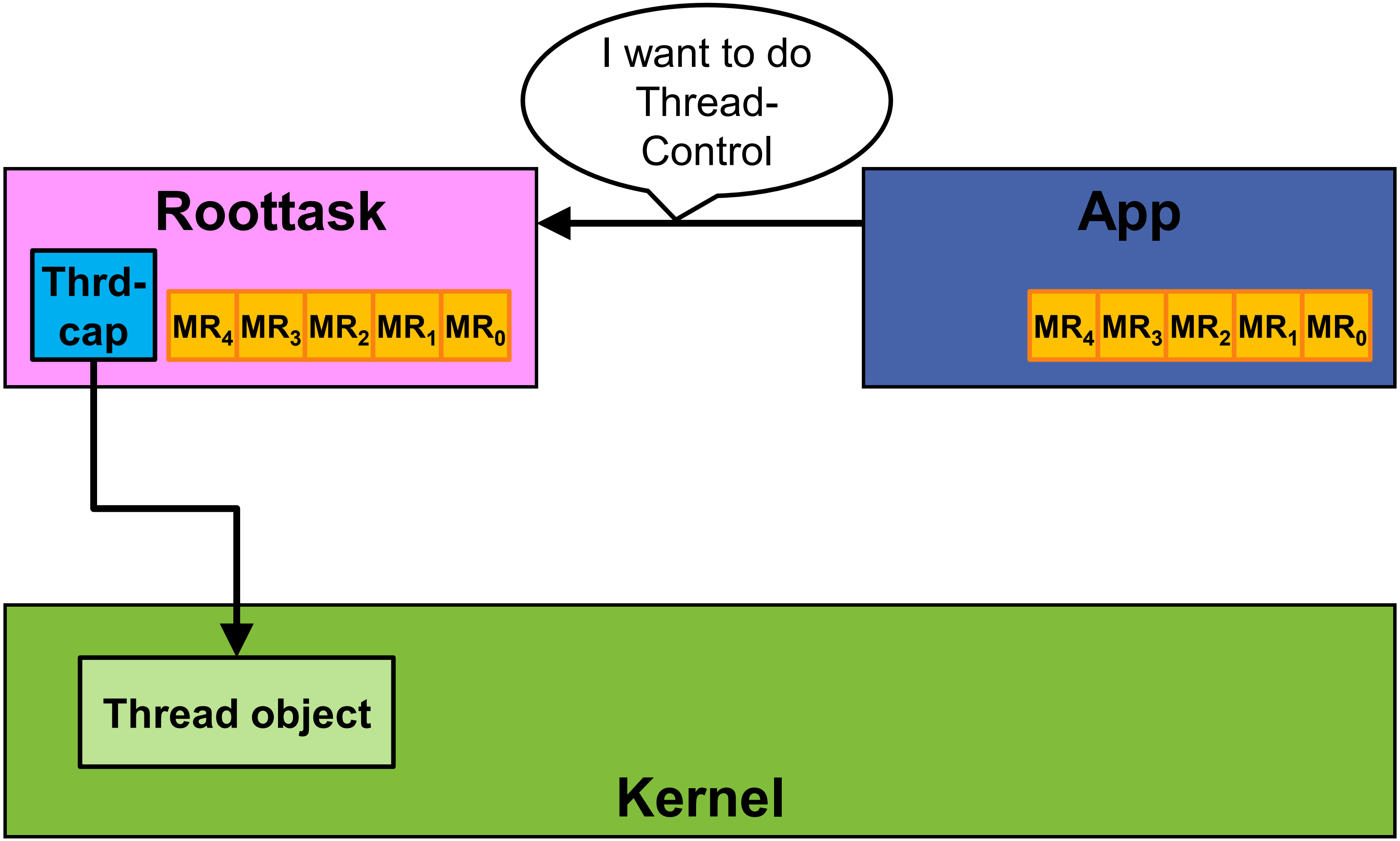
Communication Spaces with capabilites



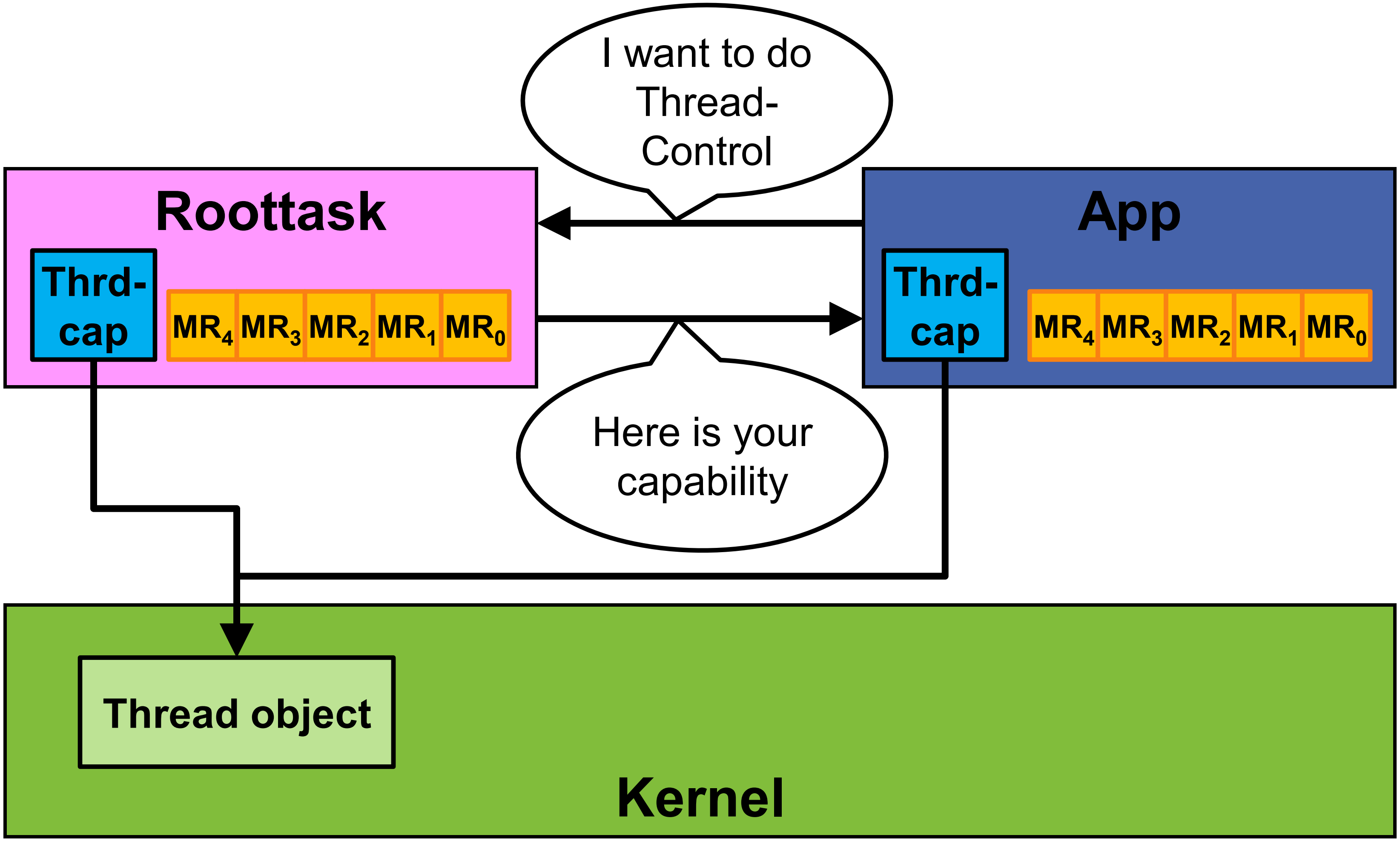
Capability properties

- Capabilities contain
  - Pointer to a kernel object
  - Access rights
- Capabilities live in kernel space
  - Not directly accessible to user
  - Referenced by index in per-AS capability array
- Capabilities provide:
  - Fine-grained access control
  - Local naming (name = idx in capability array)
    - Index has no meaning in other ASes!

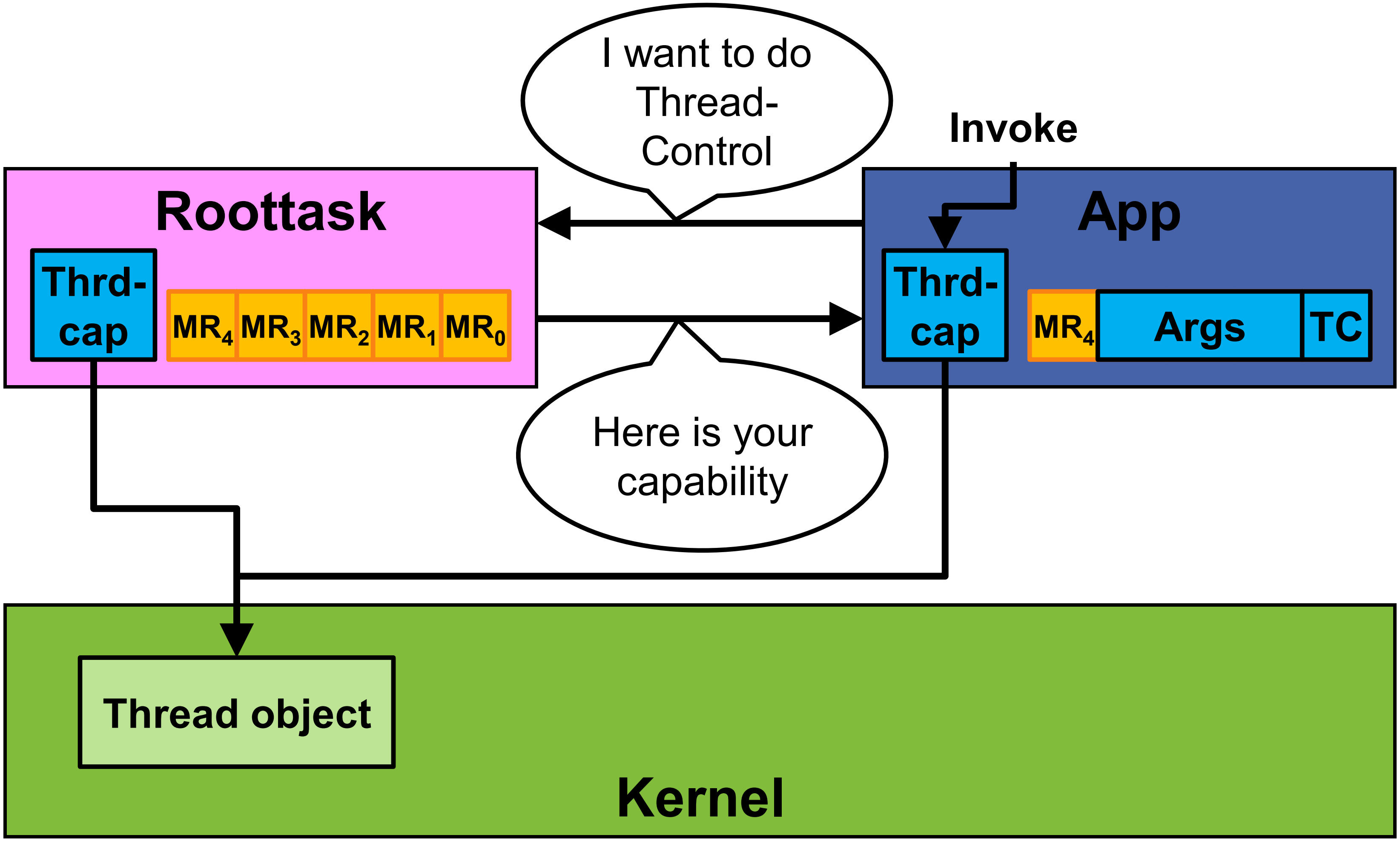
System calls with capabilities


I want to do Thread-Control
Roottask
Thrd-cap
MR_4 MR_3 MR_2 MR_1 MR_0
App
MR_4 MR_3 MR_2 MR_1 MR_0
Thread object
Kernel

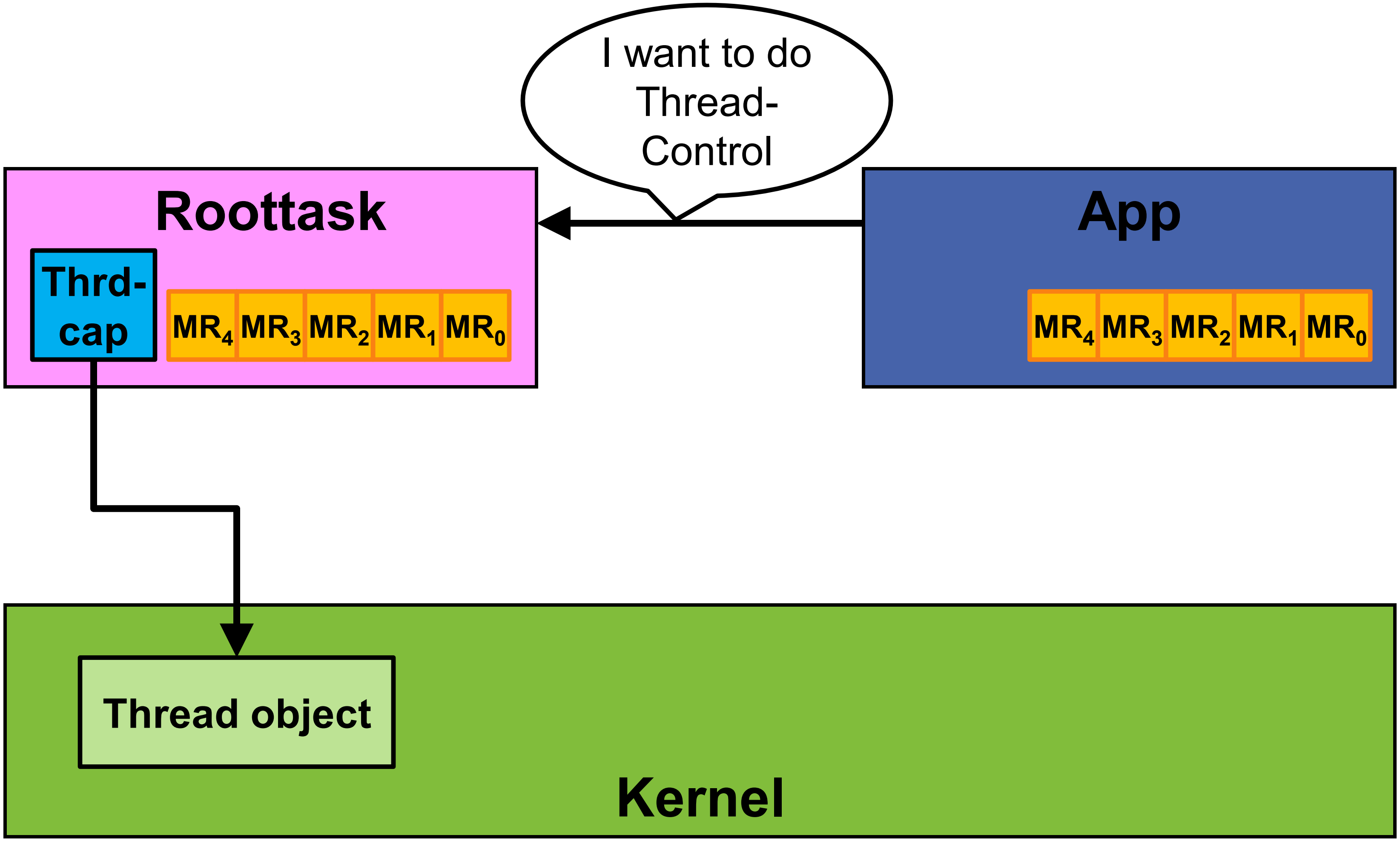
System calls with capabilities



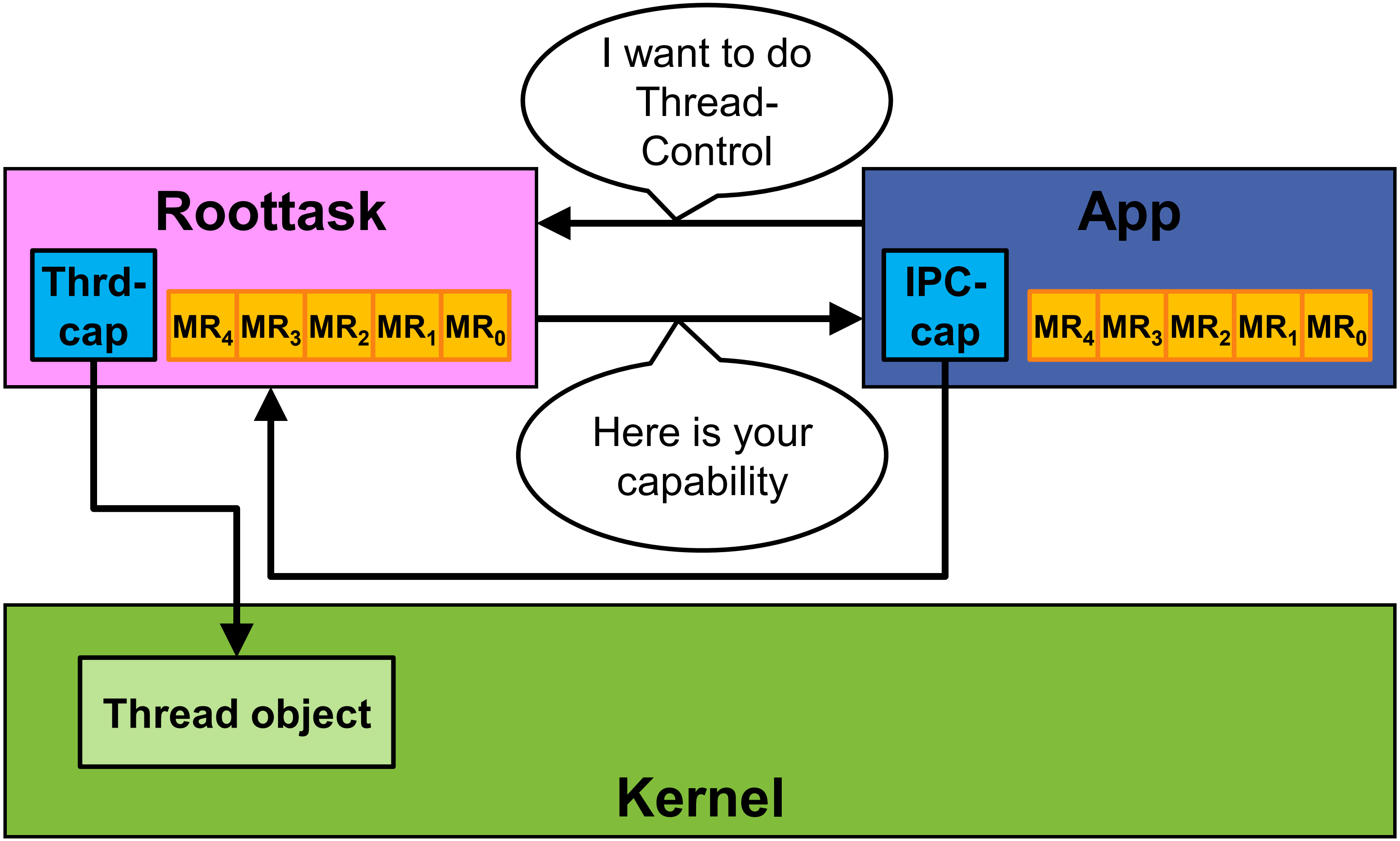
System calls with capabilities



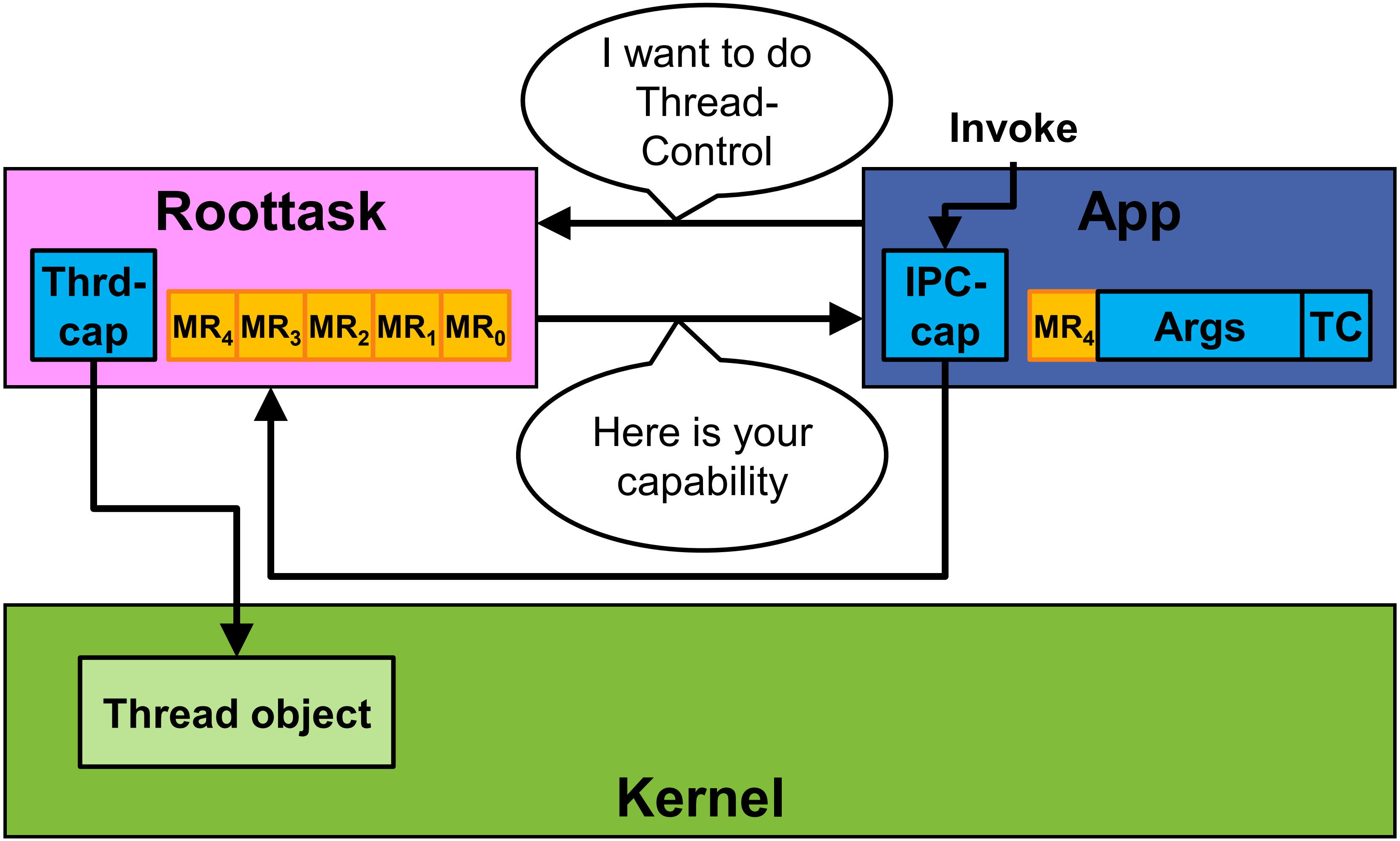
System call indirection with capabilities


I want to do Thread-Control
Roottask
Thrd-cap
MR_4 MR_3 MR_2 MR_1 MR_0
App
MR_4 MR_3 MR_2 MR_1 MR_0
Thread object
Kernel

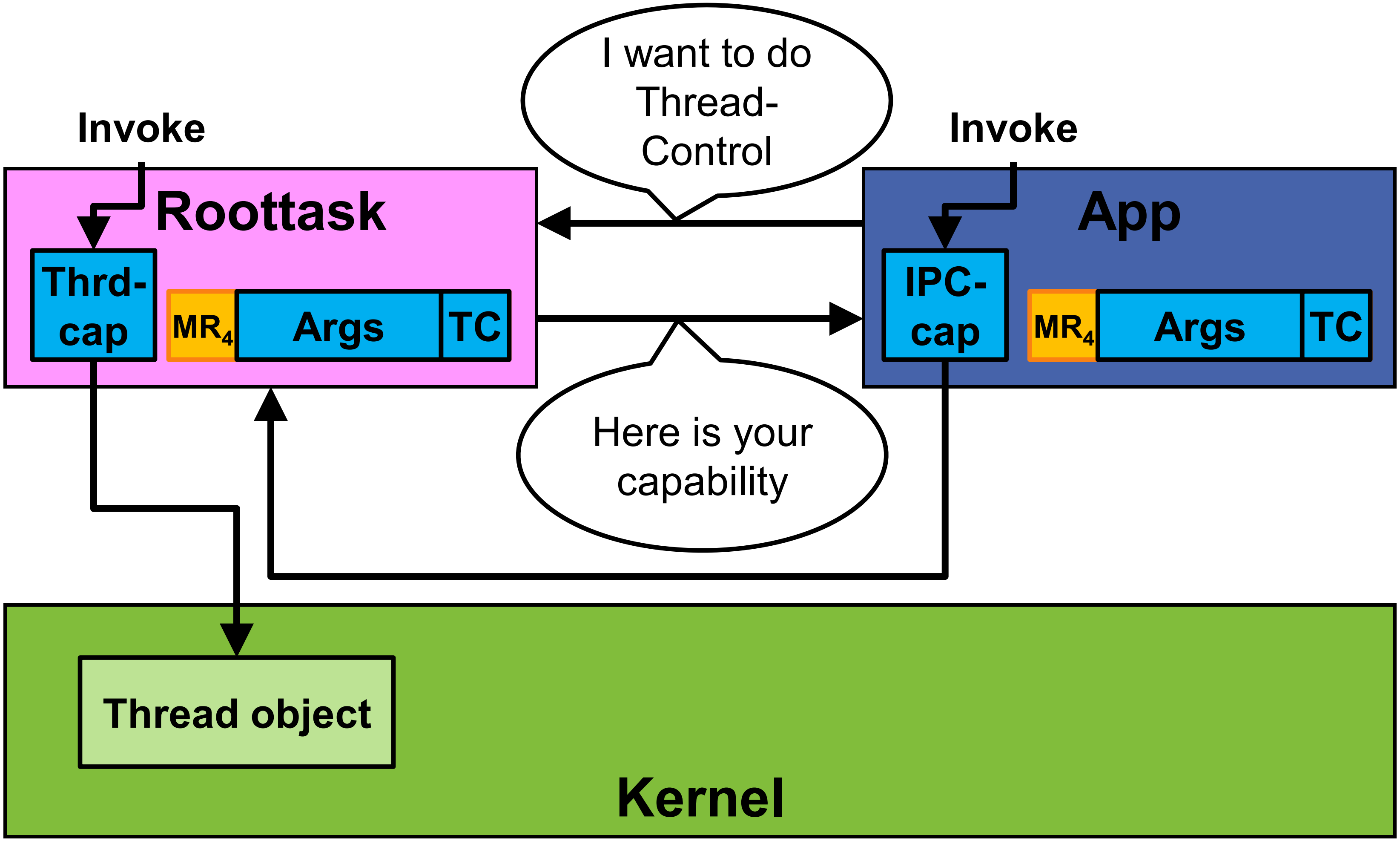
System call indirection with capabilities



System call indirection with capabilities



System call indirection with capabilities



Other Key Ideas

- Avoid memory
  - No indirection (TCB area)
  - Lazy scheduling
- Make clever use of HW features
  - Sysenter/sysexit ➔ IPC
- Serialize recursive algorithms
  - “Recursive” unmap

General Hints

- Study concepts, not details
- Give short but detailed answers
- If you don’t know the answer, think aloud
- Local IPC and Small spaces will NOT be on the exam!
- And most importantly

Don’t panic!