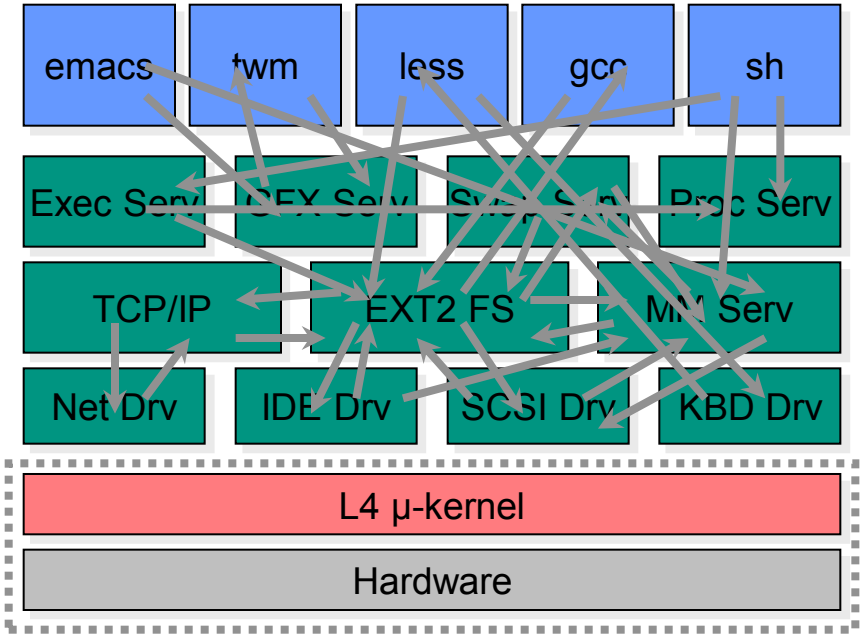
Karlsruhe Institute of Technology

Microkernel Construction
I.5 – IPC Implementation

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

Microkernel Based Systems: The Challenge



General IPC Algorithm

- Validate parameters
- Locate target thread
  - Return error if unavailable
- Transfer message
  - Untyped items (short IPC)
  - Typed items (long IPC)
- Schedule target thread
  - Switch address space as necessary
- Wait for IPC (reply/next request)

IPC IMPLEMENTATION

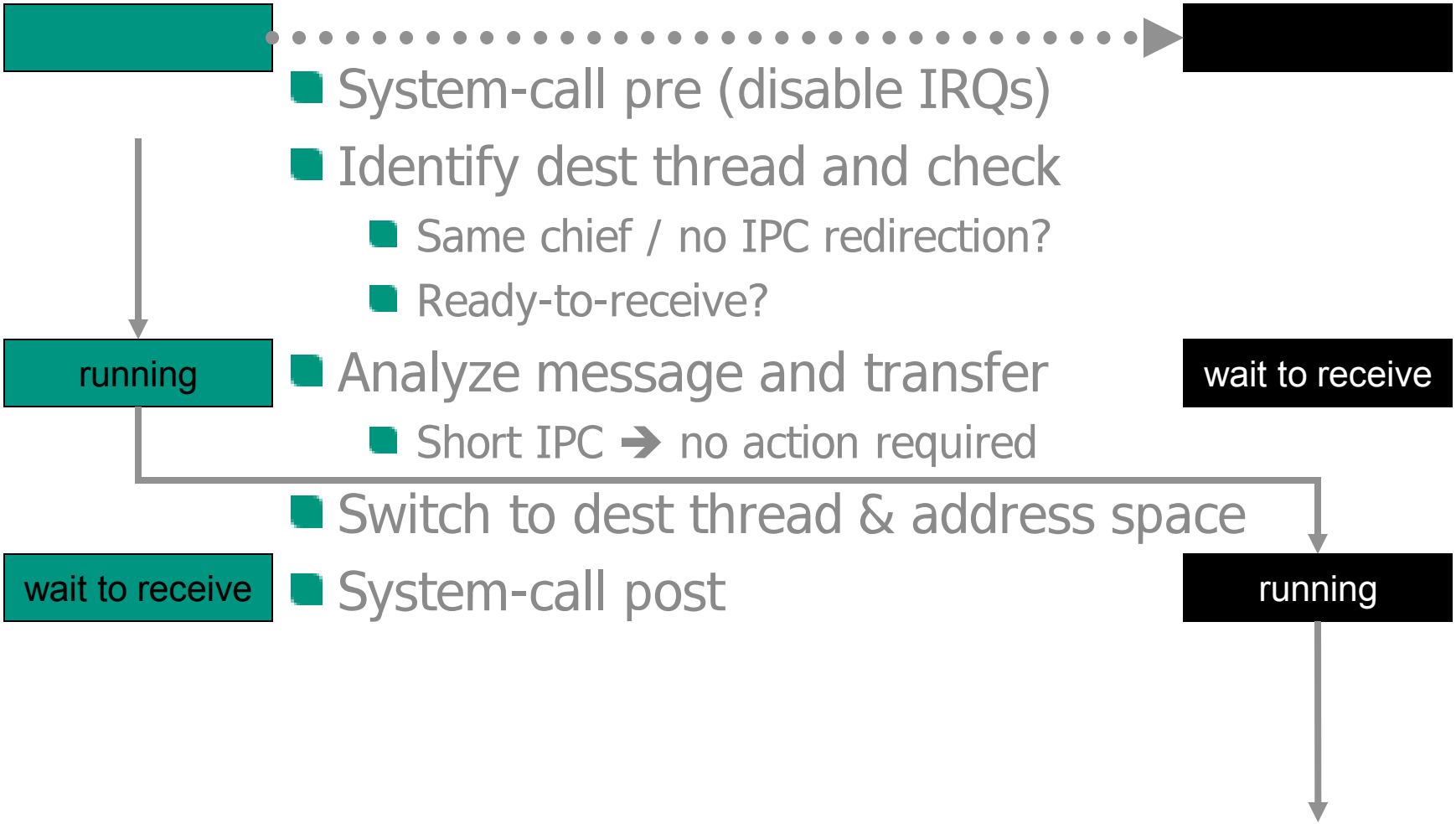
Short IPC

Short IPC (uniprocessor)

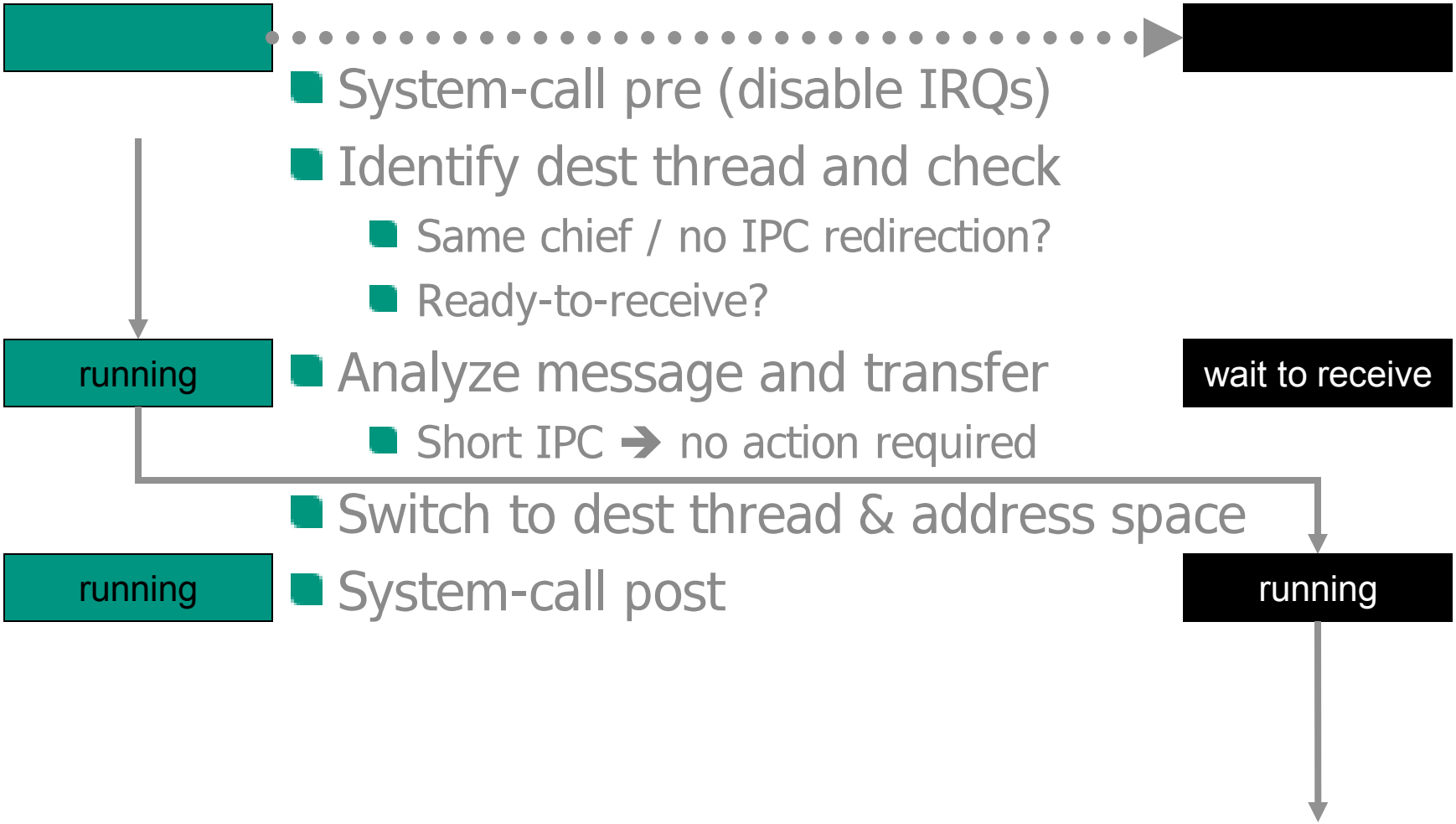
- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Short IPC ➔ no action required
- Switch to dest thread & address space
- System-call post

The critical path

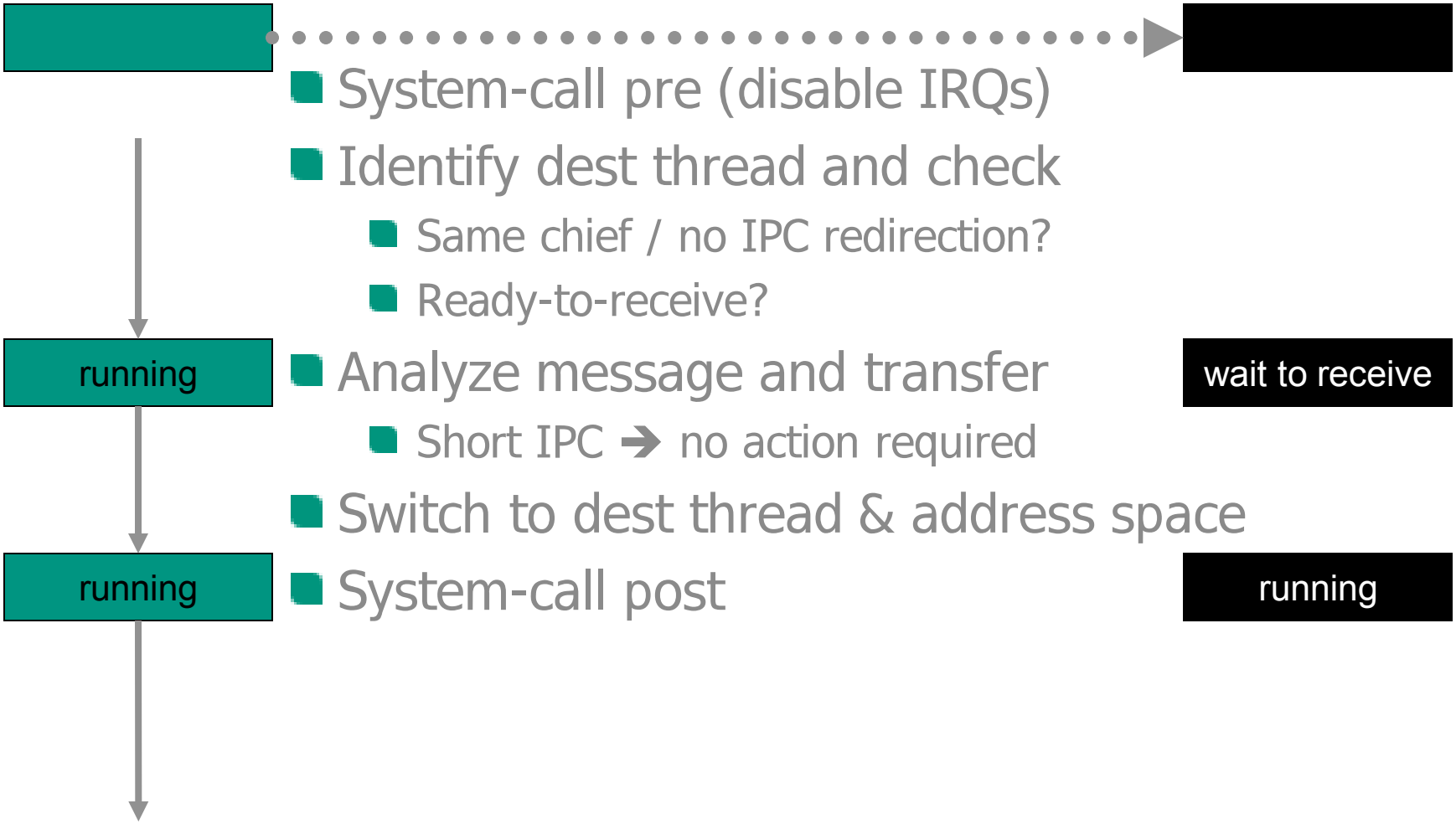
Short IPC (uniprocessor)
“call”



Short IPC (uniprocessor)
“send” (eagerly)

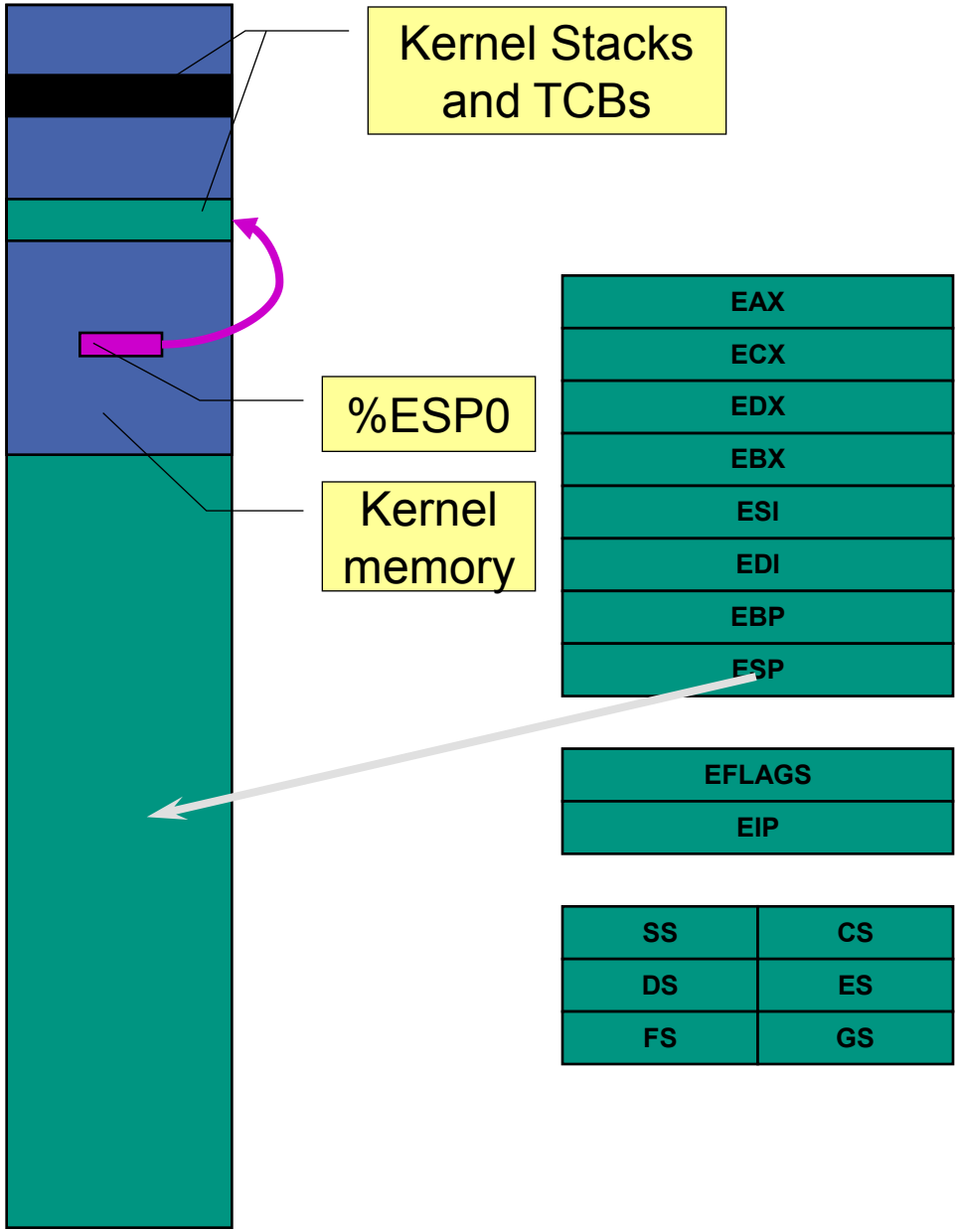


Short IPC (uniprocessor)
“send” (lazily)

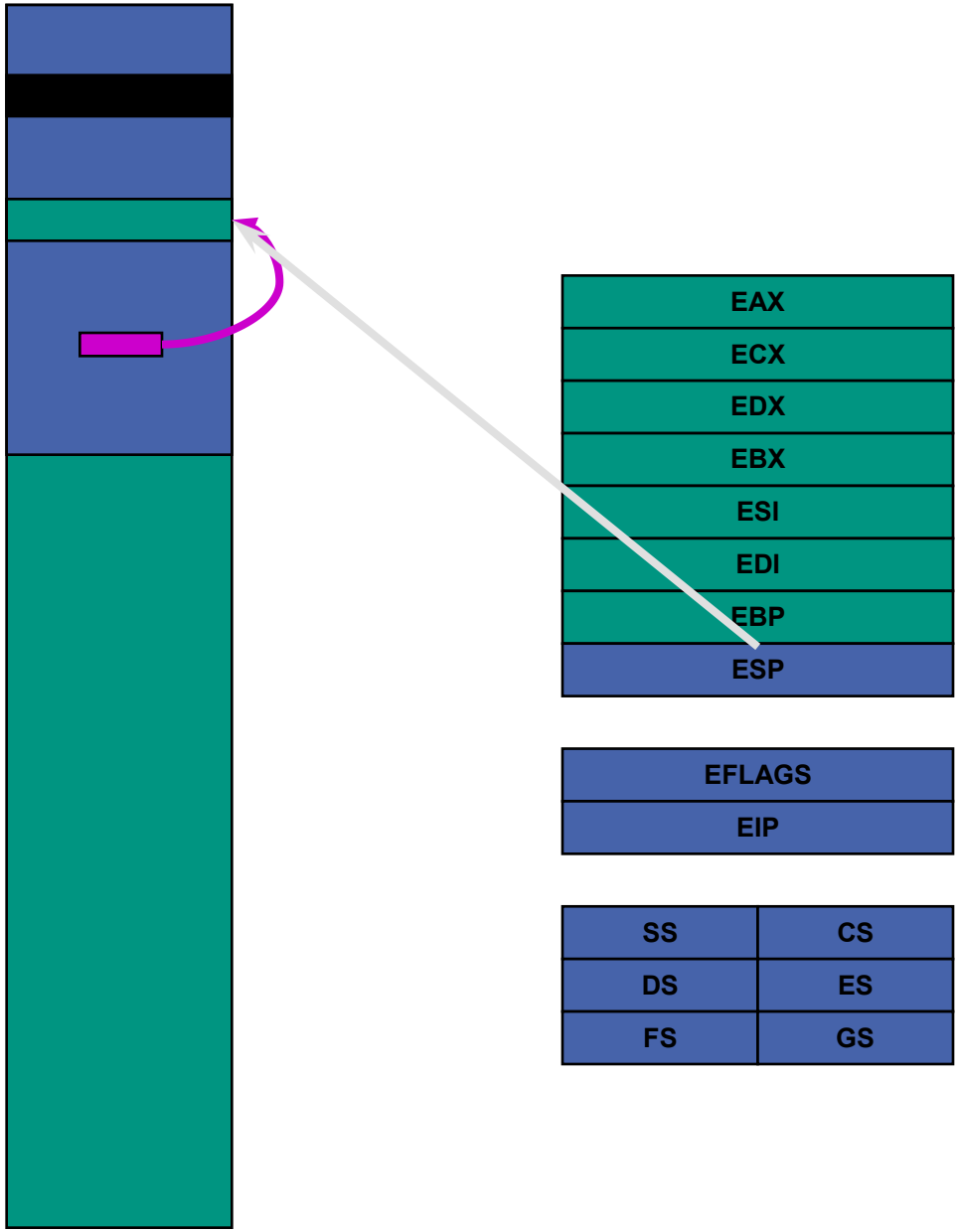


- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Short IPC ➔ no action required
- Switch to dest thread & address space
- System-call post

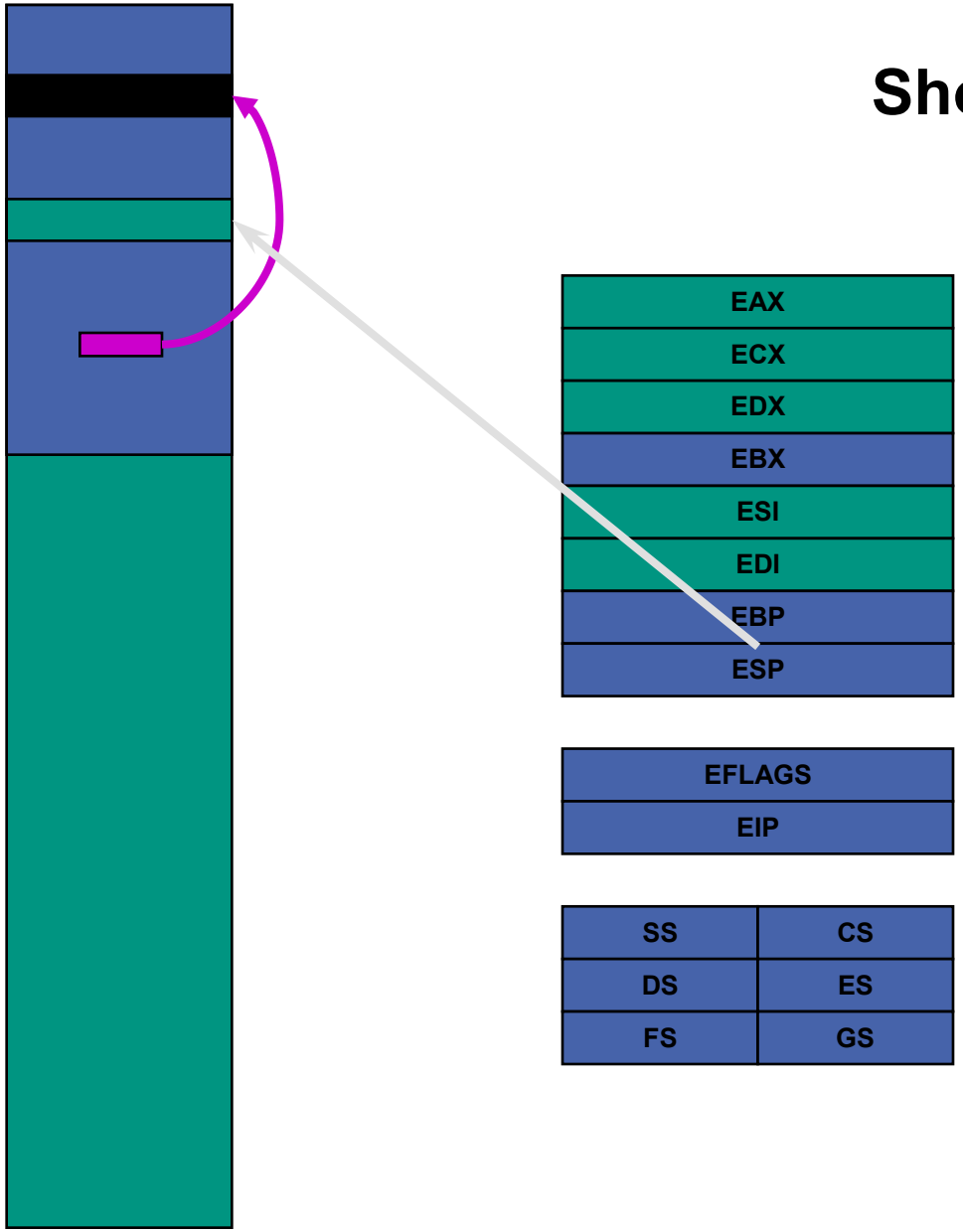
Short IPC


Kernel Stacks and TCBs
%ESP0
Kernel memory
EAX
ECX
EDX
EBX
ESI
EDI
EBP
ESP
EFLAGS
EIP
SS
CS
DS
ES
FS
GS

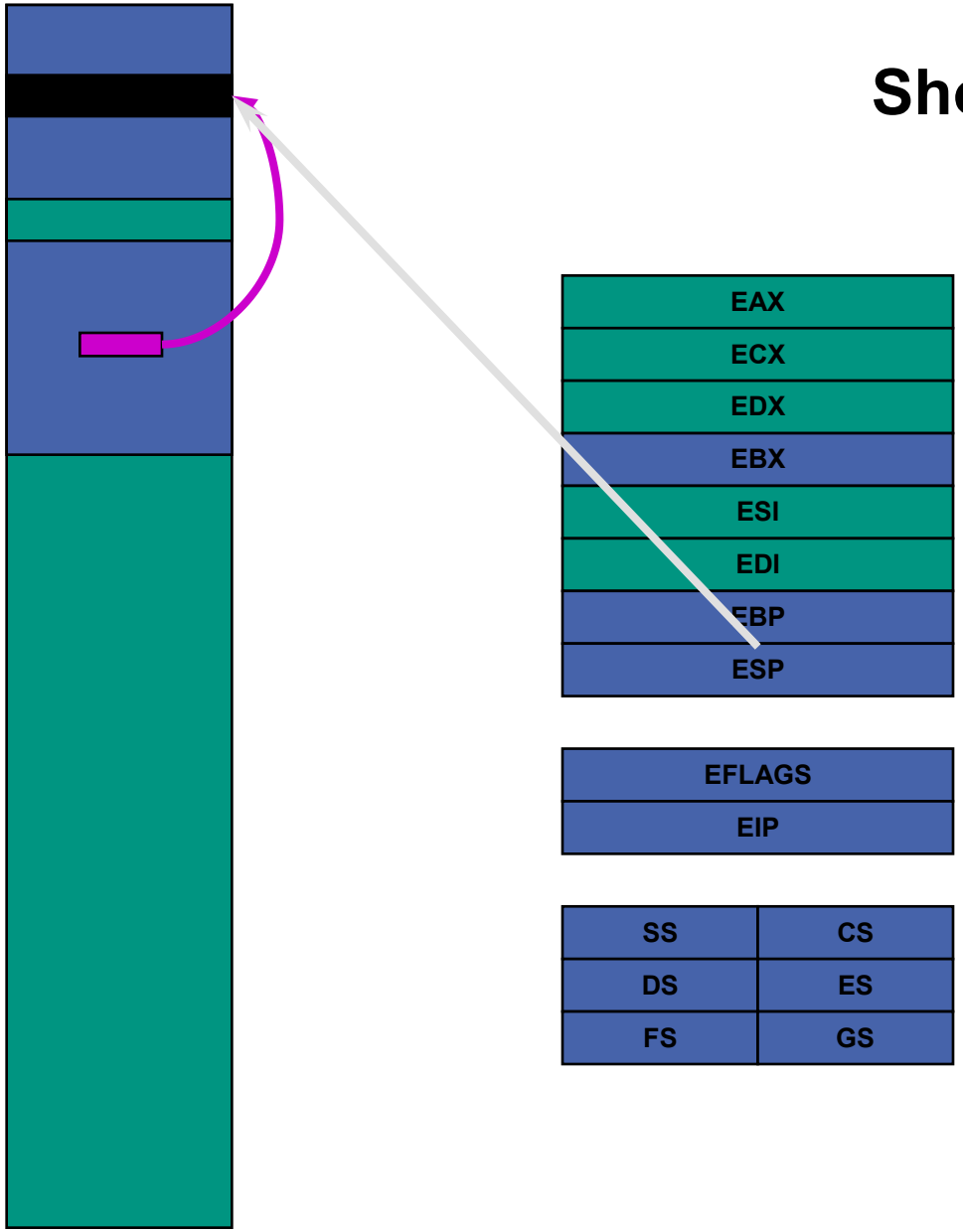
Short IPC



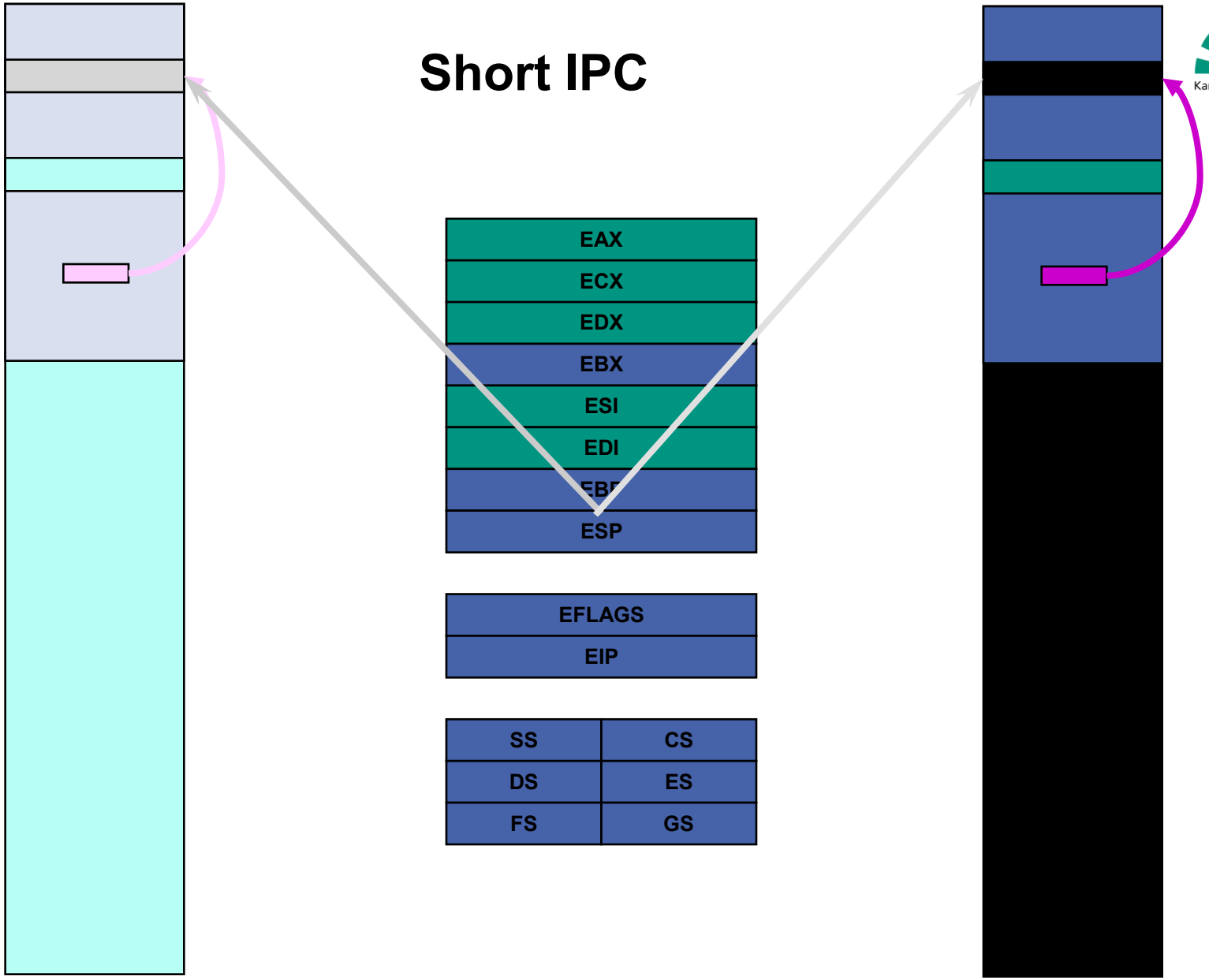
Short IPC



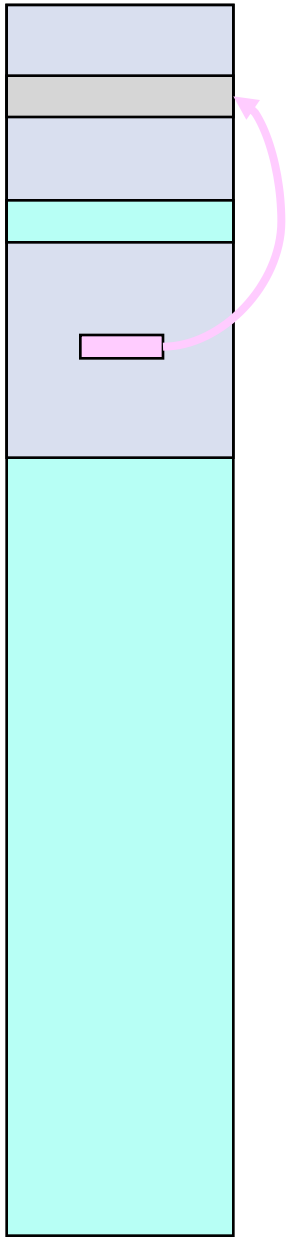
Short IPC

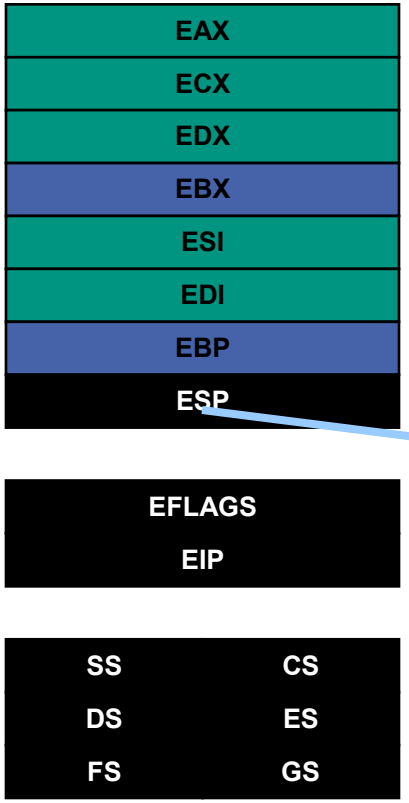


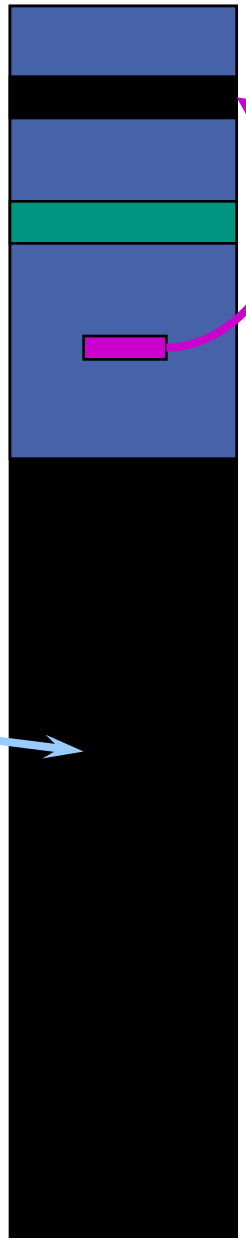
Short IPC



Short IPC







IPC via sysenter/sysexit

- Real register use
  - EAX: dest. TID → sender TID
  - ECX: timeouts → user IP (sysexit)
  - EDX: receive TID → user SP (sysexit)
  - EBX: (scratch) → MR_1
  - EBP: (scratch) → MR_2
  - ESI: MR_0 [only unchanged register]
  - EDI: UTCB(sender) → UTCB(receiver)

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 is affecting 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 …

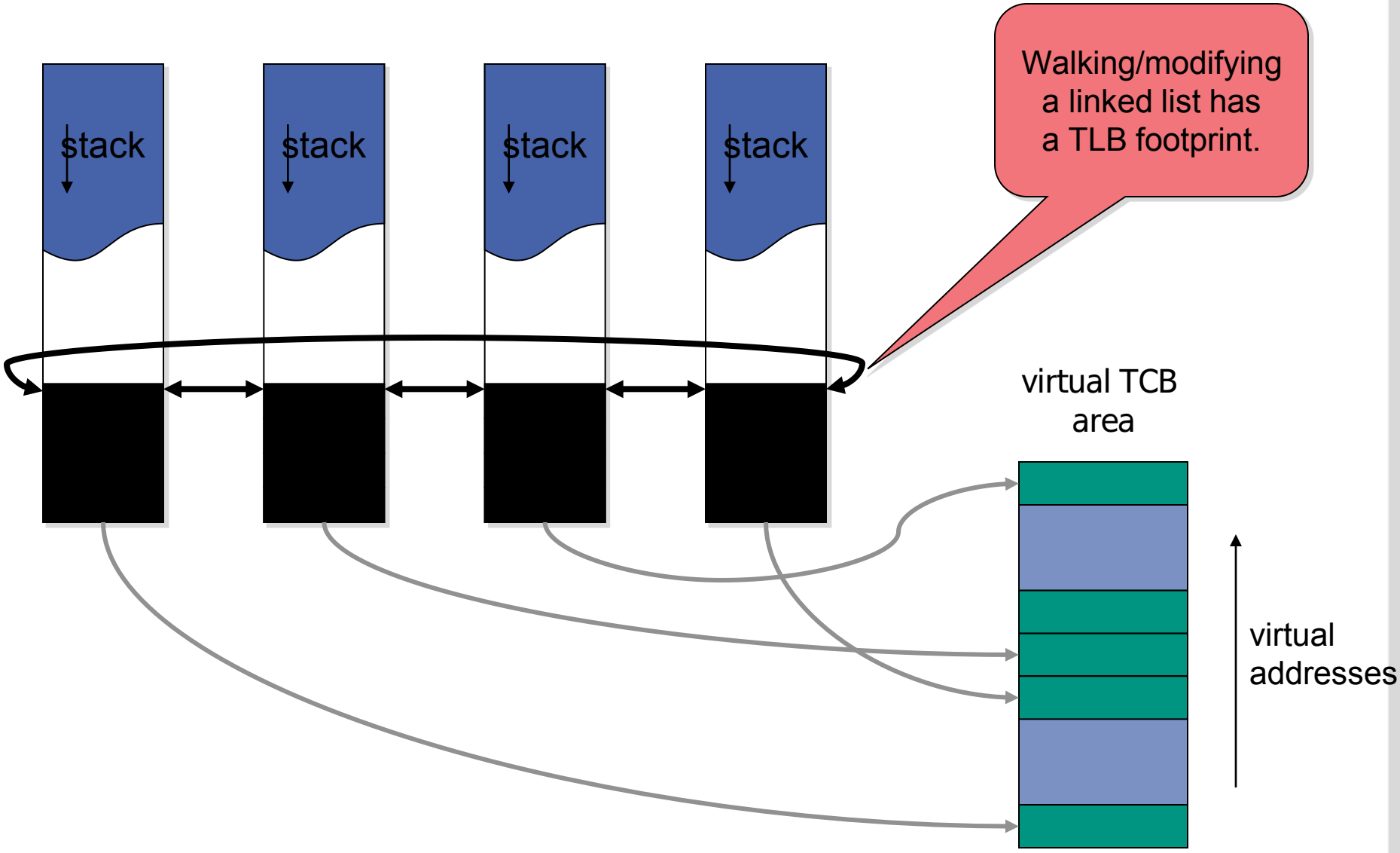
IPC Requirements for Fast Path

- Untyped message
- Single runnable thread after IPC
  - Must be valid call-like IPC
    - Send phase
      - Target is already waiting
    - Receive phase
      - Sender is not ready to couple, causing us to block
  - Switch threads, originator blocks
- Infinite receive timeout
  - Send timeout can be ignored: receiver is waiting
  - Xfer timeouts do not apply for untyped messages

Memory is “Forbidden”

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

TLB Problem with Eager Scheduling



Lazy Scheduling

- Do not update the scheduling lists
  - Blocked sender remains in ready list
    - Check real thread state when dispatching
    - May be unblocked before being scheduled
      ➔ avoids list manipulations
  - Unblocked receiver not added to ready list
    - Append to ready list at end of timeslice
    - May block before
      ➔ avoids list manipulations

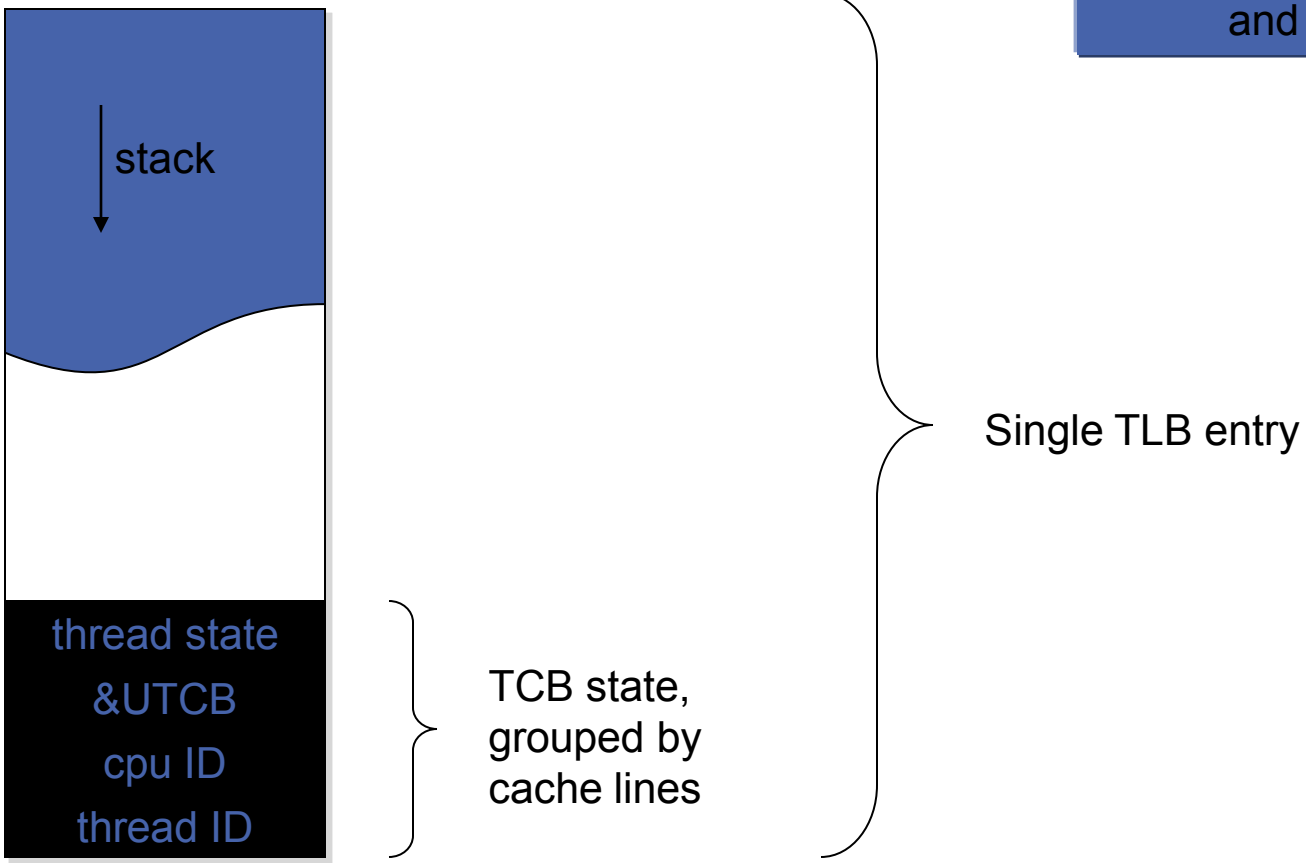
Discussed in more detail next week

Memory is “Forbidden”

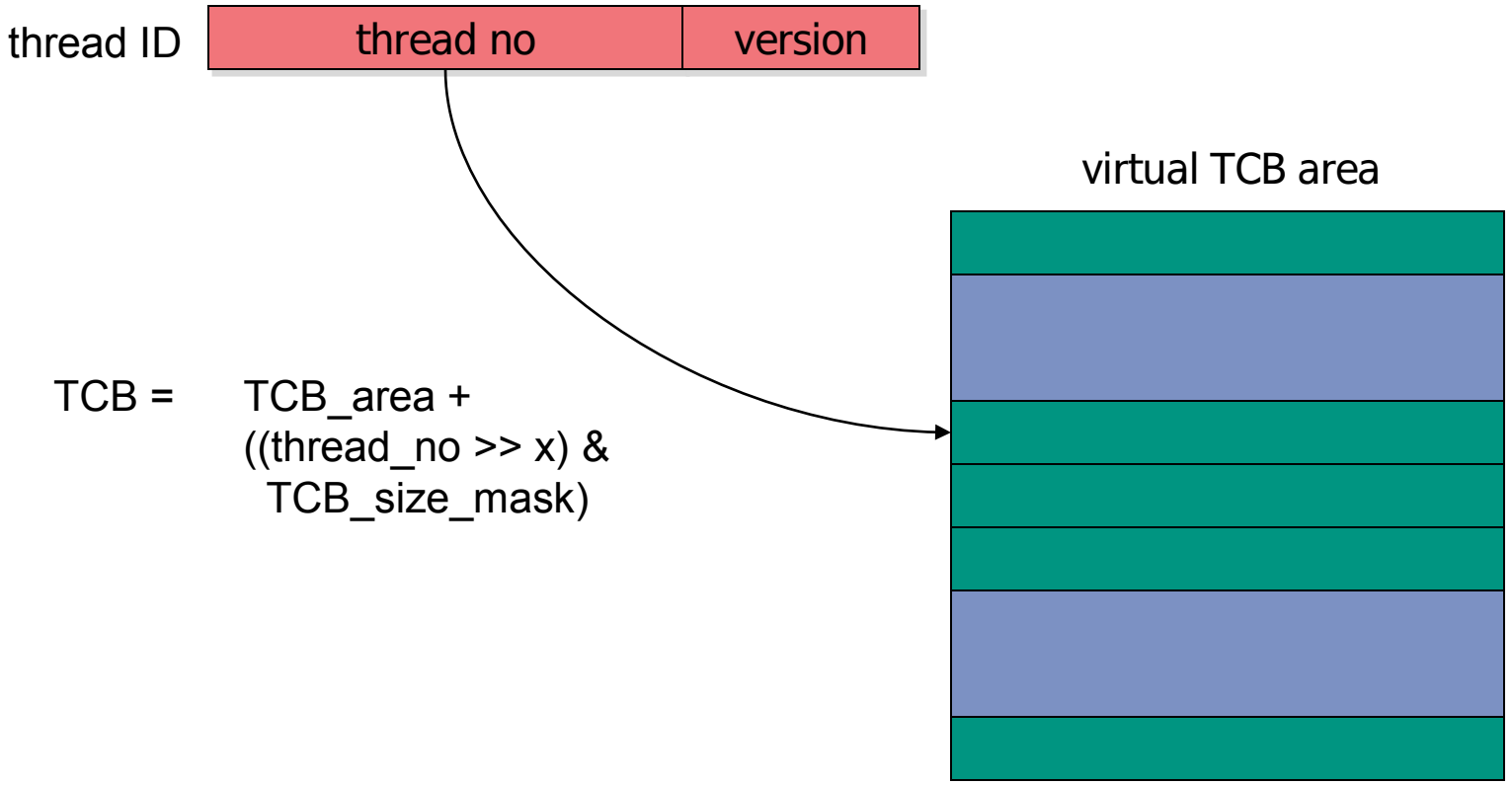
- Memory references are slow
  - Avoid in common case
    - E.g., (xfer) timeouts
  - Avoid in IPC
    - E.g., use lazy scheduling
- Microkernel should minimize artifacts
  - Cache pollution
  - TLB pollution
  - Memory bus

Optimized Memory

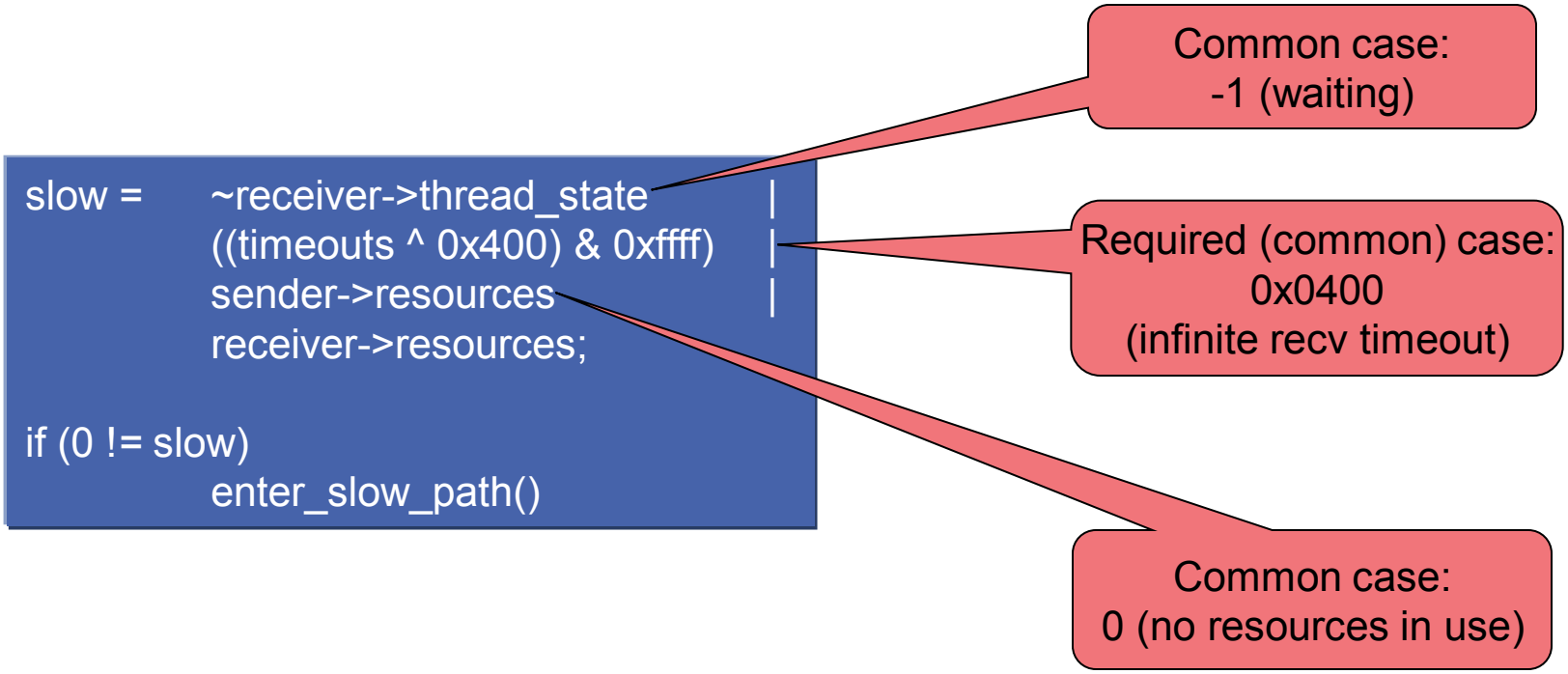
Also, hard-wire TLB entries for kernel code and data.



Avoid Table Lookups

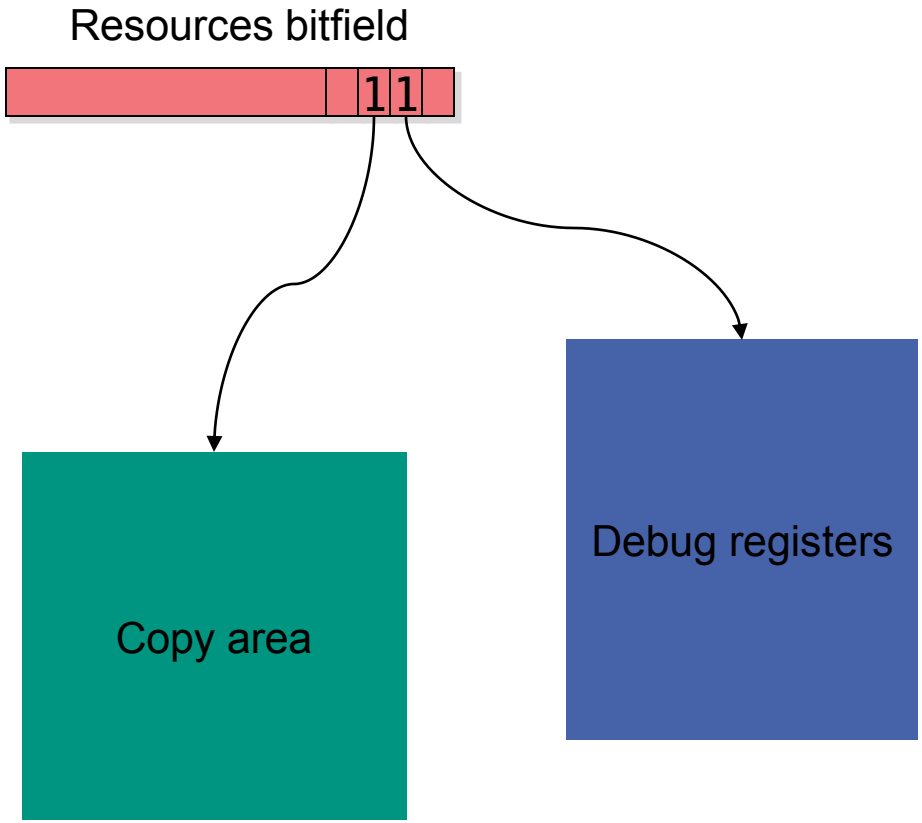


Branch Elimination



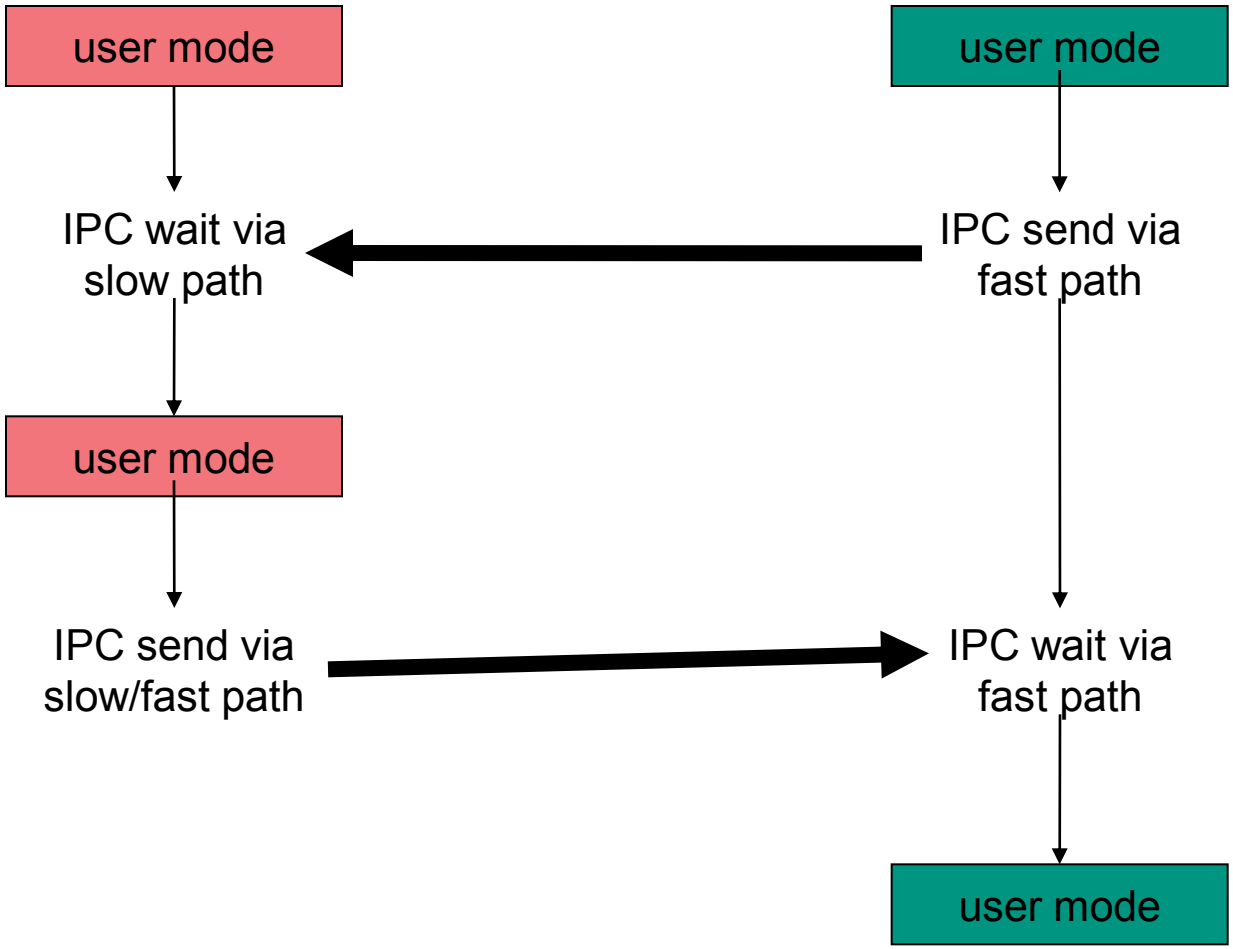
- Reduces branch prediction foot print
- Avoids mispredictions, stalls, and flushes
- Slightly increases latency for slow path

TCB Resources



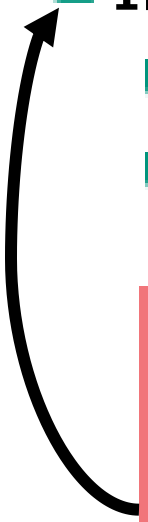
- One bit per resource
- Fast path checks entire word
  - If not 0, jump to resource handlers

Slow and Fast



Consistent State

- Cooperative thread scheduling in kernel
- TCB in consistent state for IPC wait
  - IPC restores user mode context
    - Avoids cycles for restoring kernel context
    - Fast path can activate slow path TCB



Problem?

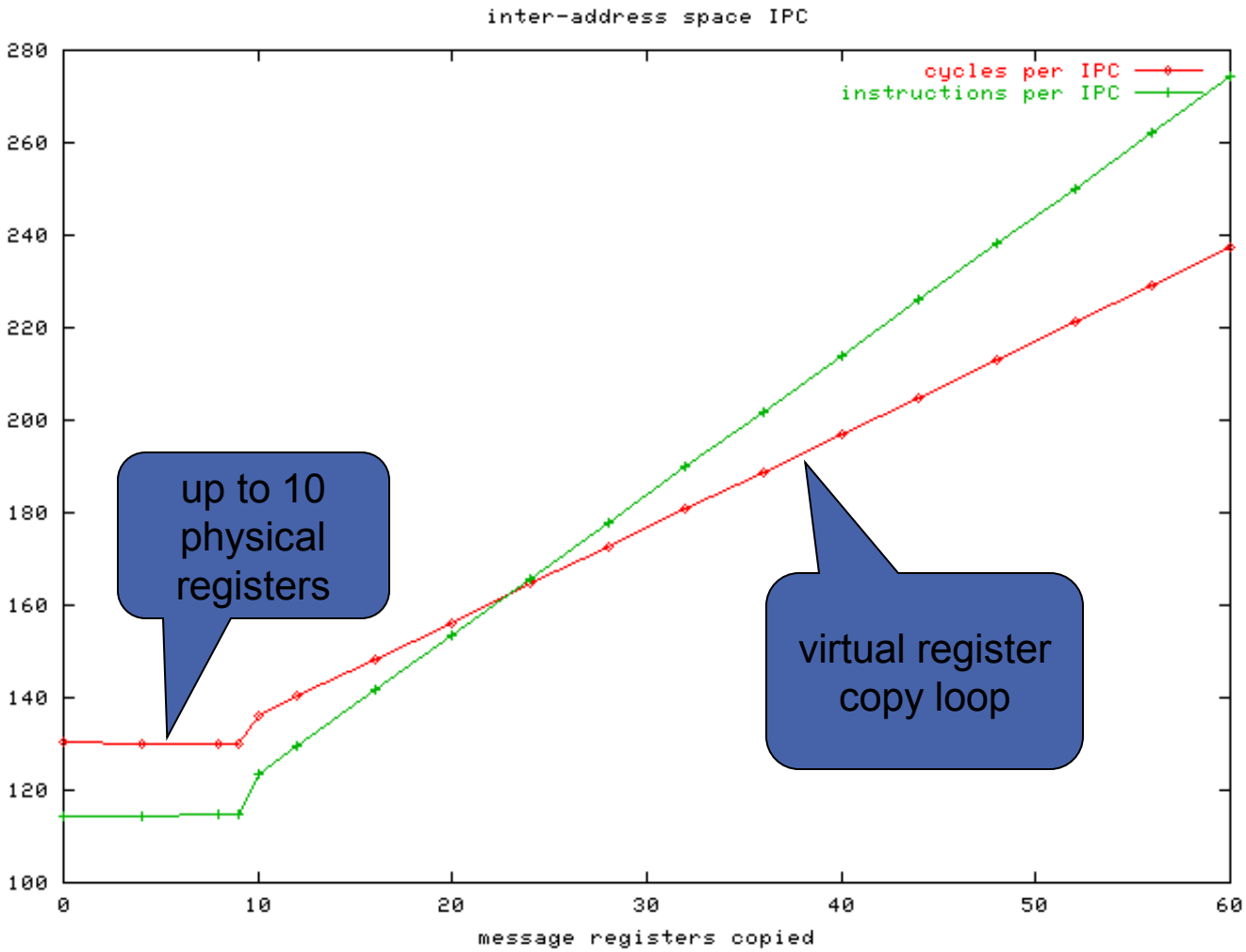
Can't use fast path for kernel threads.

How often do kernel threads use IPC?

How to efficiently detect kernel threads?

➔ Use resource bit!

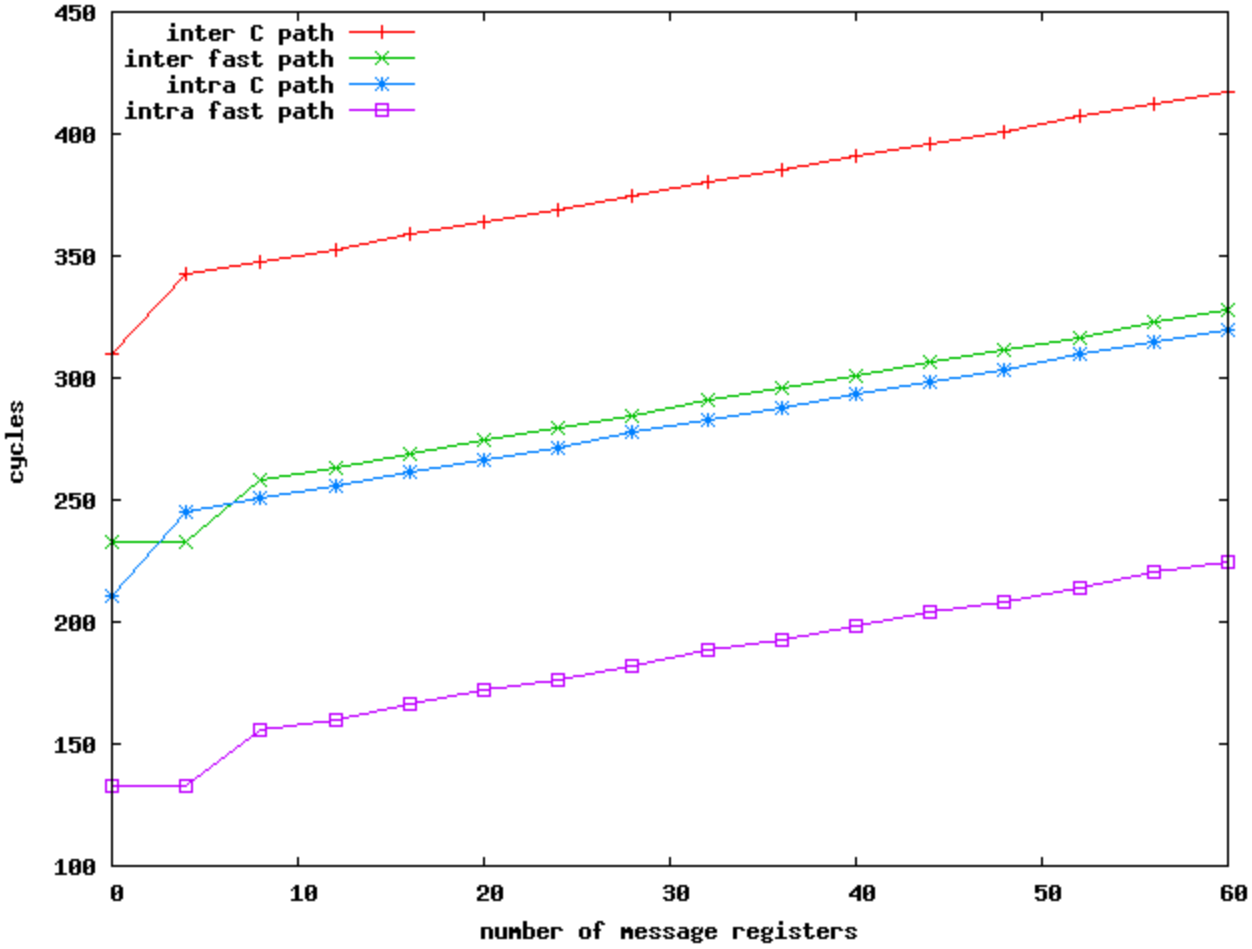
Short IPC Performance (1)



IBM PowerPC 750,
500 MHz,
32 registers

Many cycles wasted on pipeline flushes for privileged instructions.

Short IPC Performance (2)



AMD Opteron 242,

1.6 GHz, 64 bit

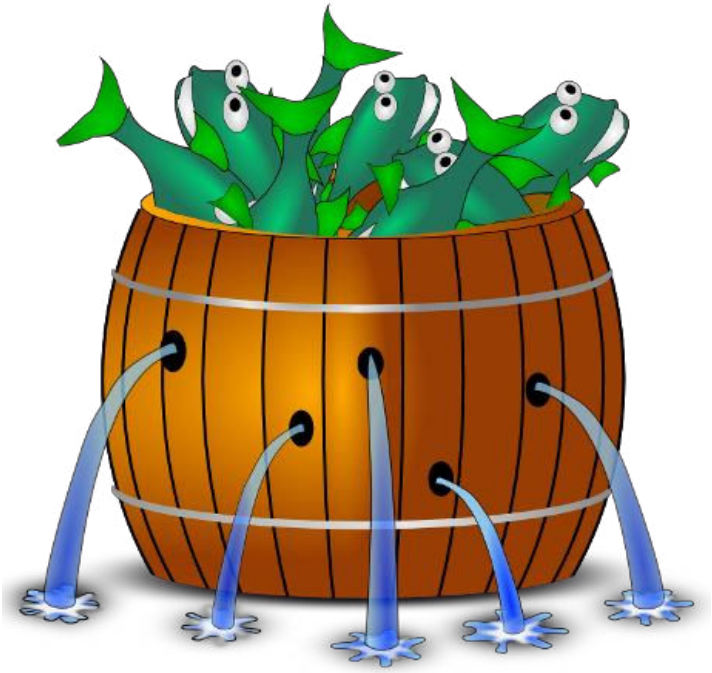
Short IPC: One fundamental problem…

Only works if sender and receiver are on the same core!

…but nowadays, we have SMP

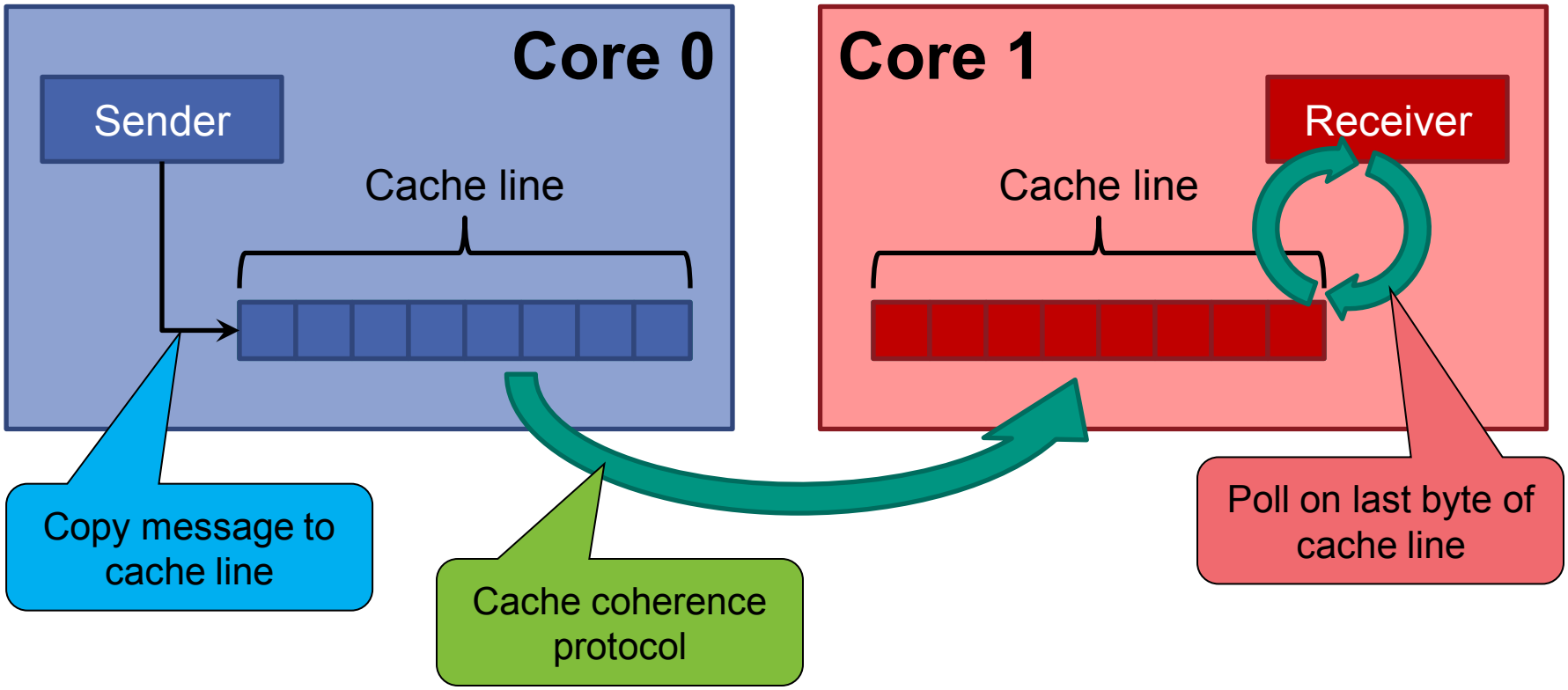
Barrelfish

- Baumann et al (ETH Zürich/Microsoft Research)
- Assume many cores (100s!)
- One kernel per core
  - „Multikernel OS“
- Shared-nothing architecture
  - All communication explicit



IPC in Barrelfish

- Use cache-coherent memory



IPC in Barrelfish

- Payload transferred via fastest possible channel
  - Ideally, message never touches memory

- Performance (dual Opteron 2220, 2.8 GHz):

	Latency (cycles)	Throughput (msgs/kcycle)	Icache lines	Dcache lines
URPC	450	3.42	9	8
L4 IPC	424	2.36	25	13

- But: Polling wastes cycles
  - Barrelfish assumes dedicated core
- False sharing can affect IPC performance

IPC IMPLEMENTATION

Long IPC

Long IPC (uniprocessor)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Long/map:
    - *– transfer message –*
- Switch to dest thread & address space
- System-call post

Preemptions possible!
(end of timeslice, device interrupt, …)

Pagefaults possible!
(in source and dest address space)

Long IPC (uniprocessor)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Long/map:
    - Lock both partners
    - *– transfer message –*
    - Unlock both partners
- Switch to dest thread & address space
- System-call post

Preemptions possible!
(end of timeslice, device interrupt, …)

Pagefaults possible!
(in source and dest address space)

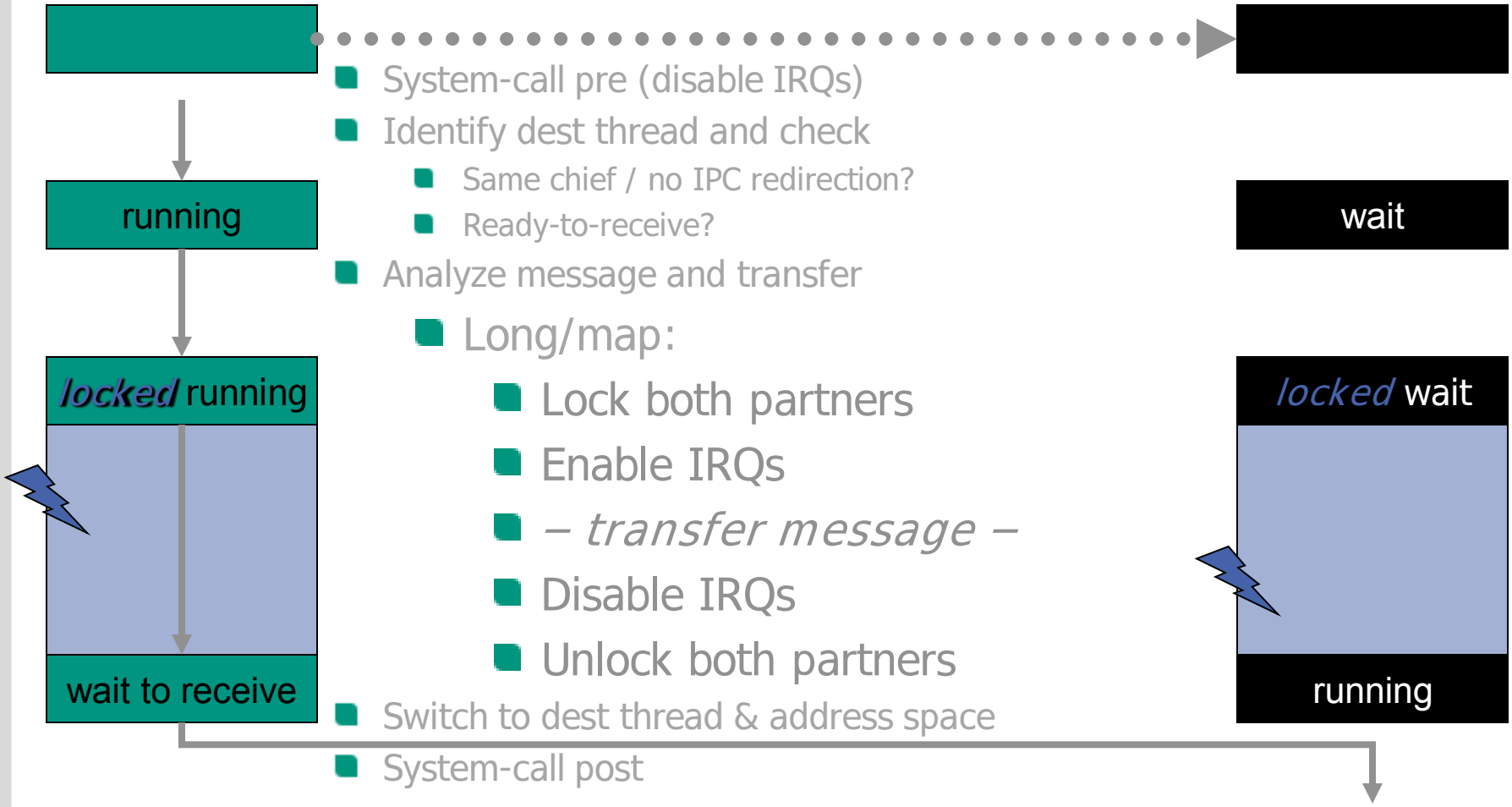
Long IPC (uniprocessor)

- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Long/map:
    - Lock both partners
    - Enable IRQs
    - *– transfer message –*
    - Disable IRQs
    - Unlock both partners
- Switch to dest thread & address space
- System-call post

Preemptions possible!
(end of timeslice, device interrupt, …)

Pagefaults possible!
(in source and dest address space)

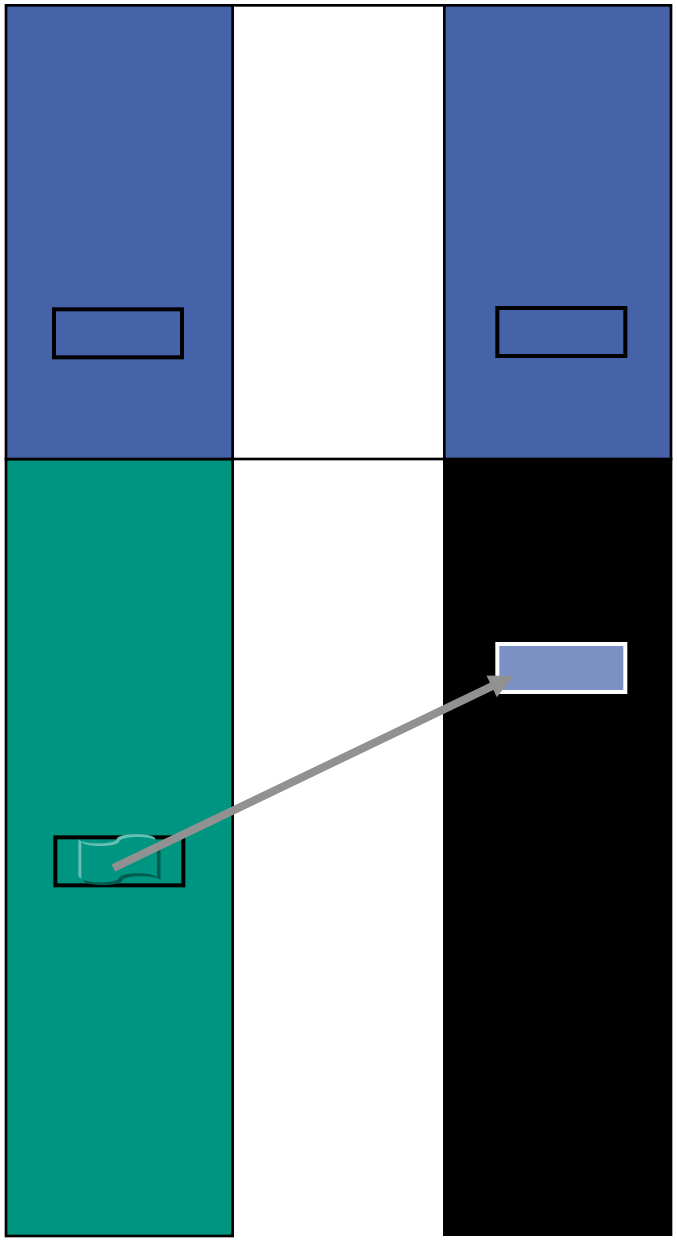
Long IPC (uniprocessor)



- System-call pre (disable IRQs)
- Identify dest thread and check
  - Same chief / no IPC redirection?
  - Ready-to-receive?
- Analyze message and transfer
  - Long/map:
    - Lock both partners
    - Enable IRQs
    - *– transfer message –*
    - Disable IRQs
    - Unlock both partners
- Switch to dest thread & address space
- System-call post

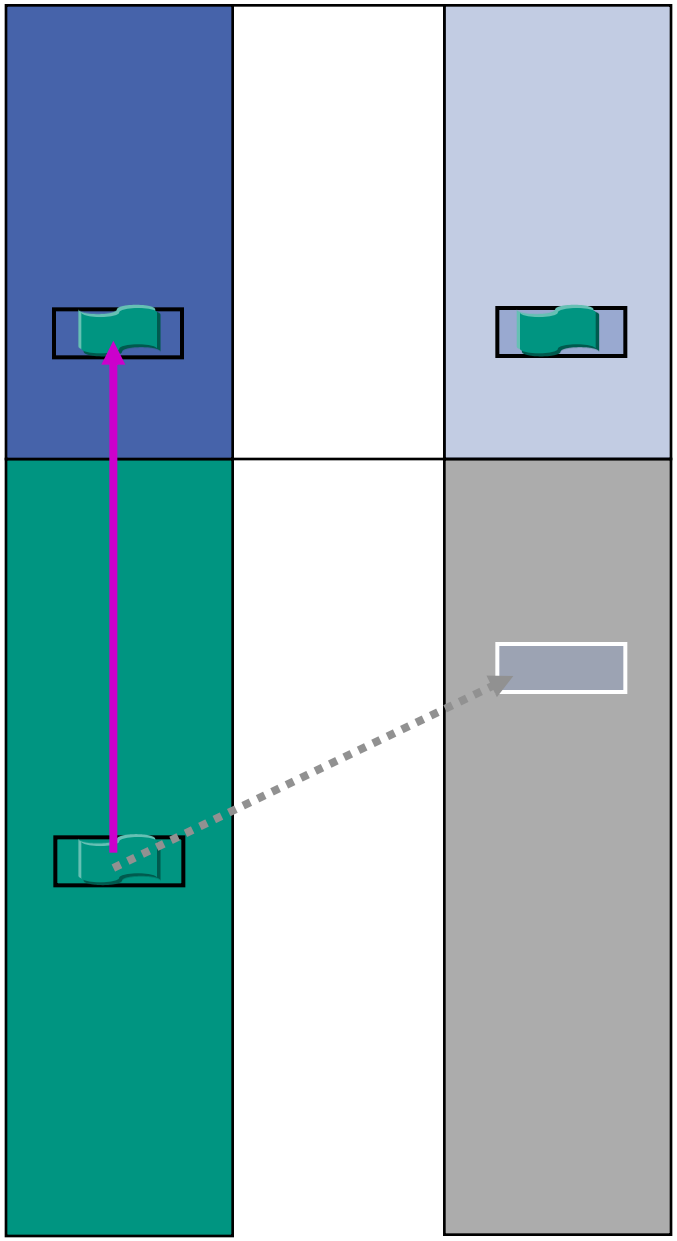
String IPC / memcpy

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



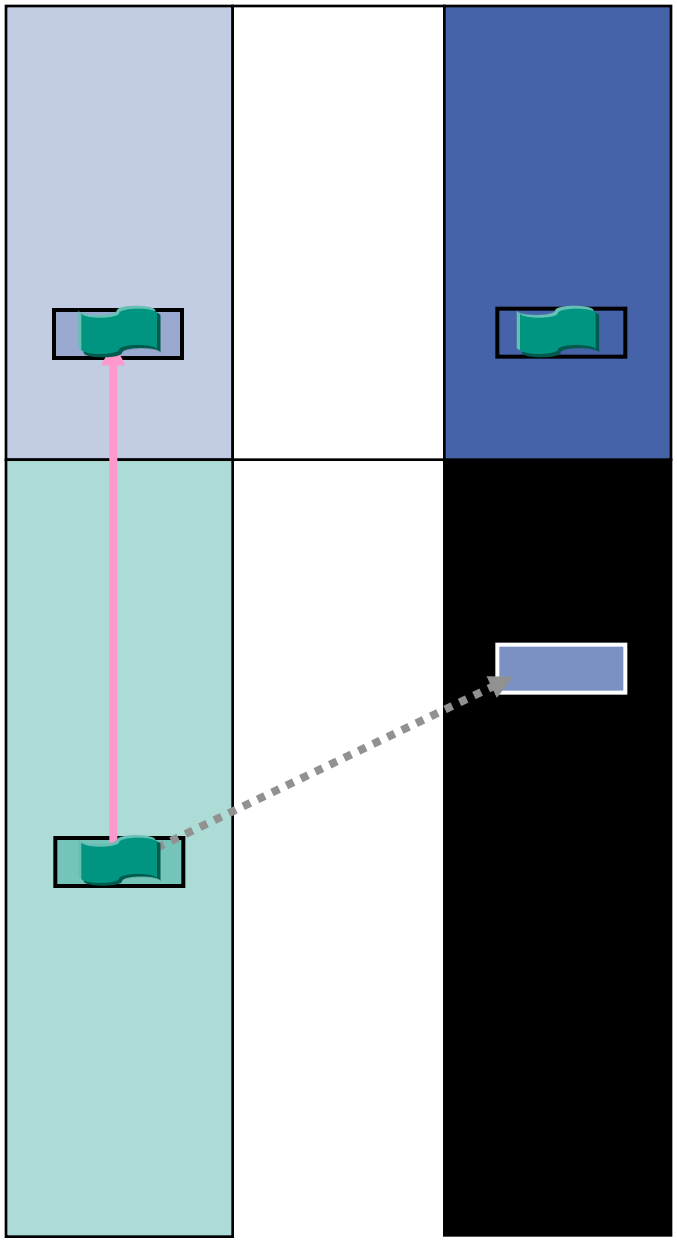
Copy In – Copy Out

- Copy into kernel buffer



Copy In – Copy Out

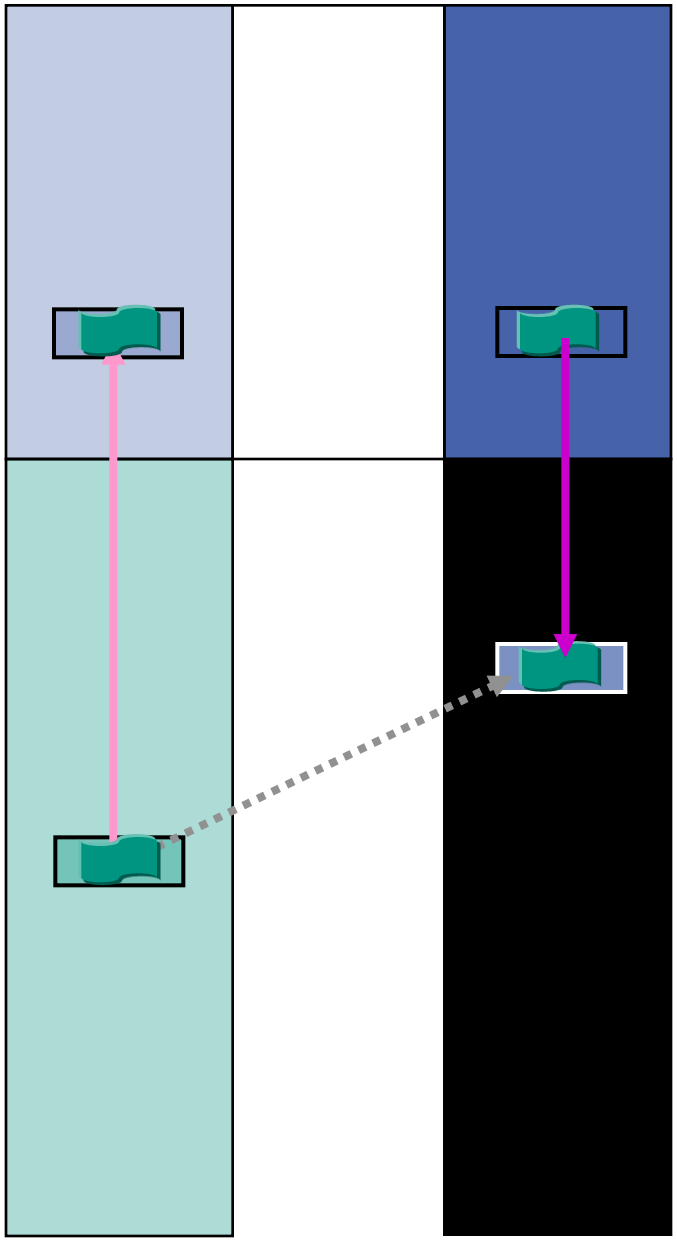
- Copy into kernel buffer
- Switch spaces



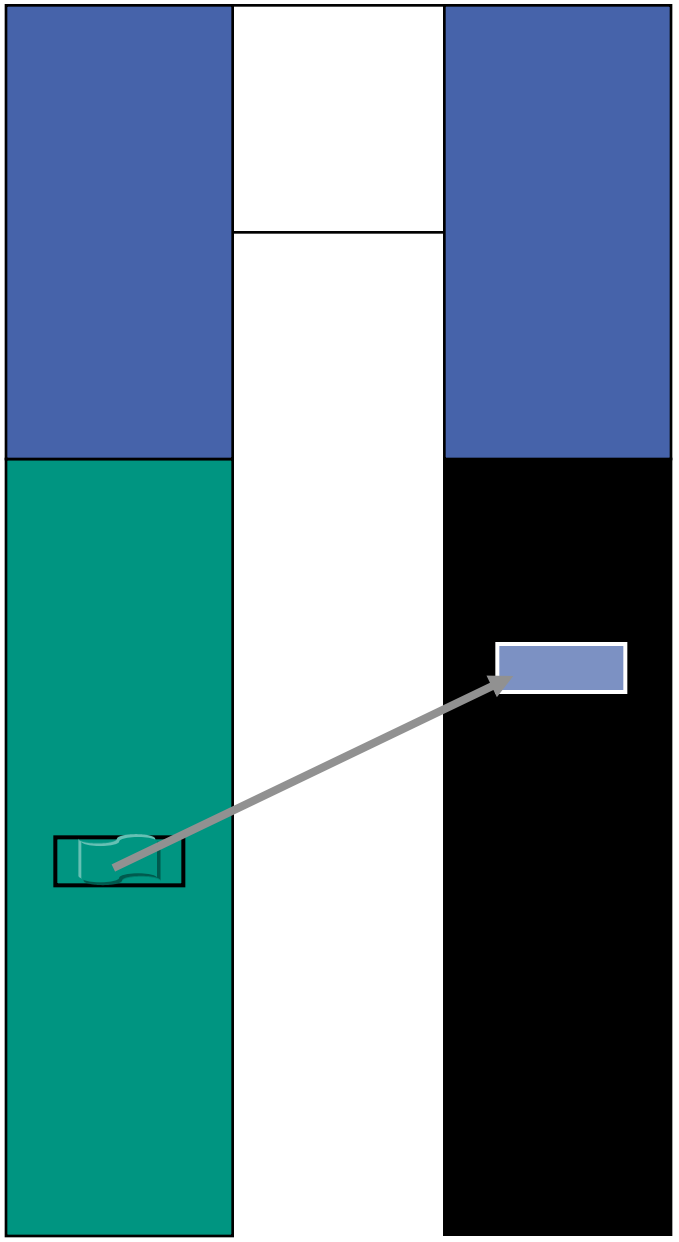
Copy In – Copy Out

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

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

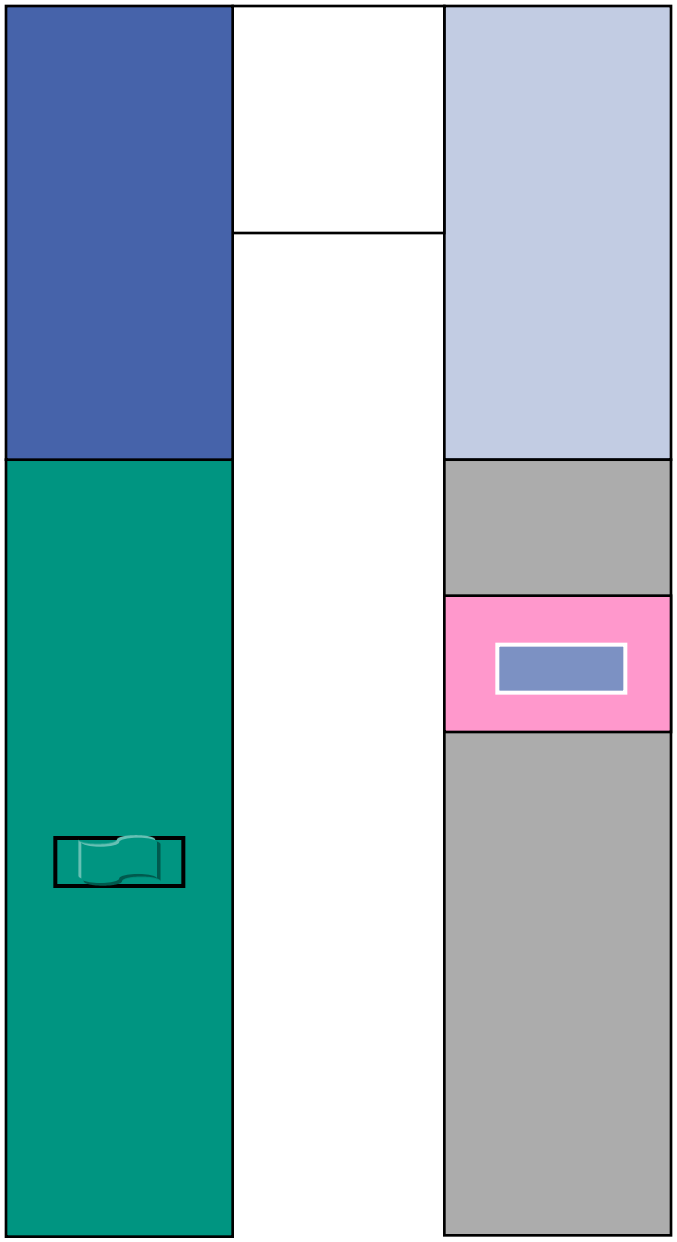


Temporary Mapping



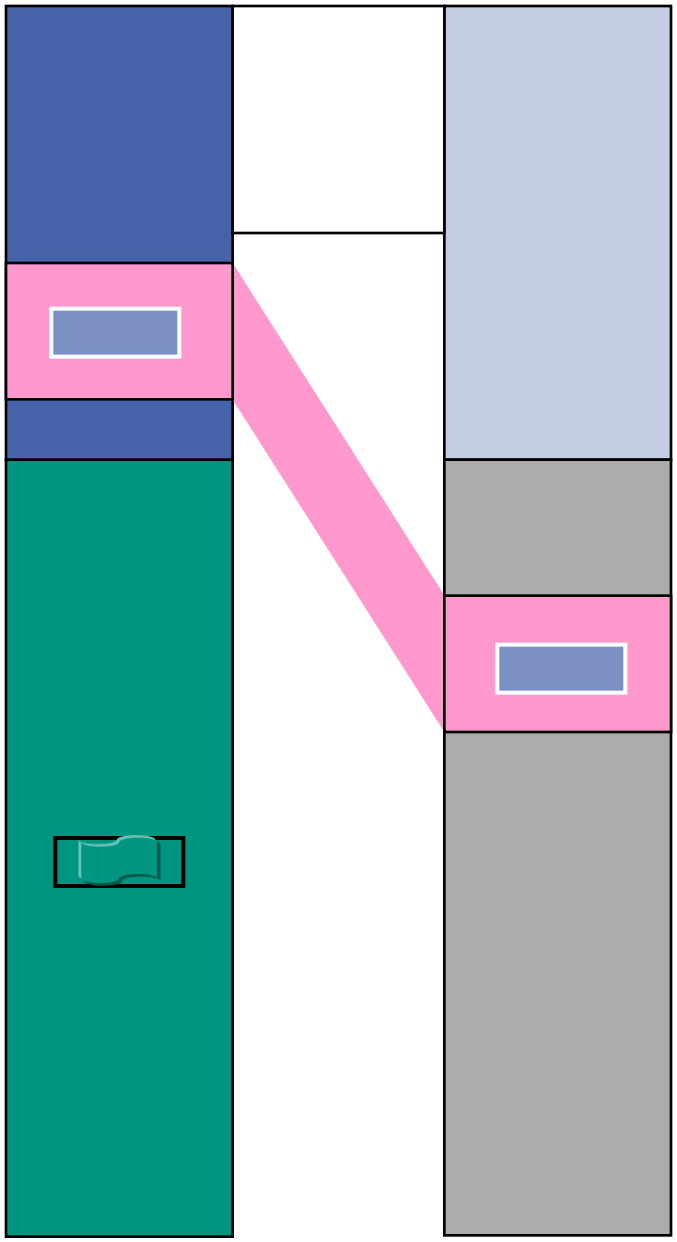
Temporary Mapping

- Select dest area (2x4 MB)



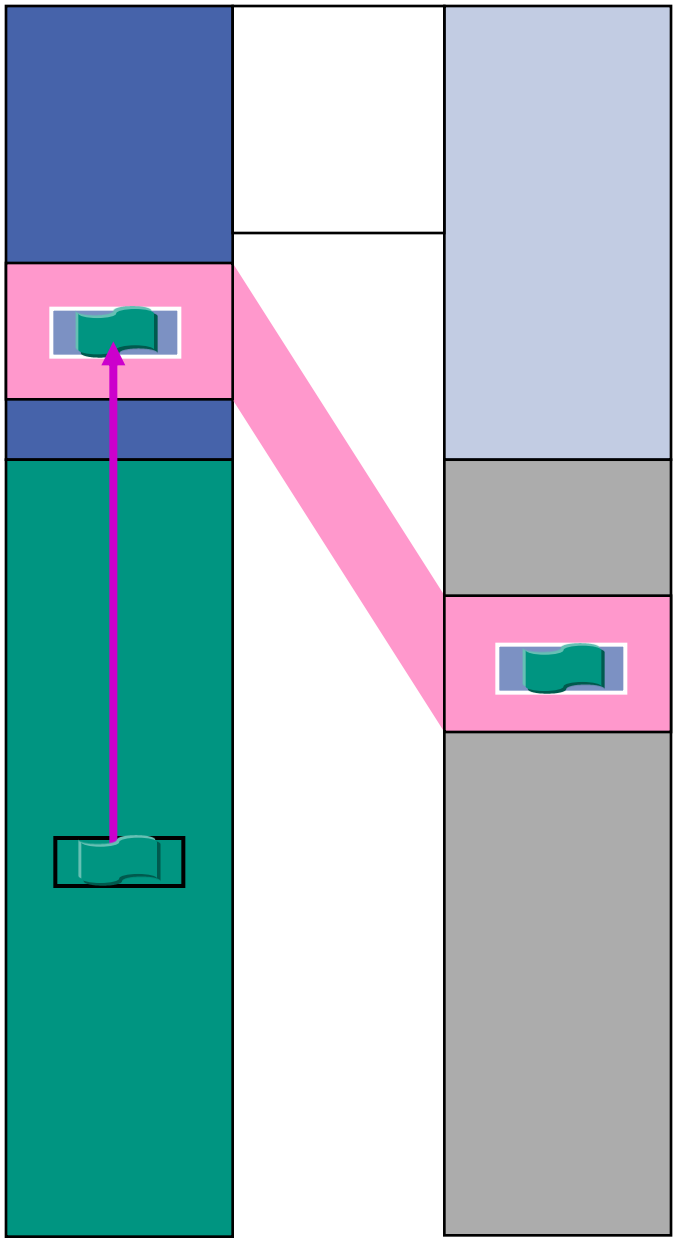
Temporary Mapping

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



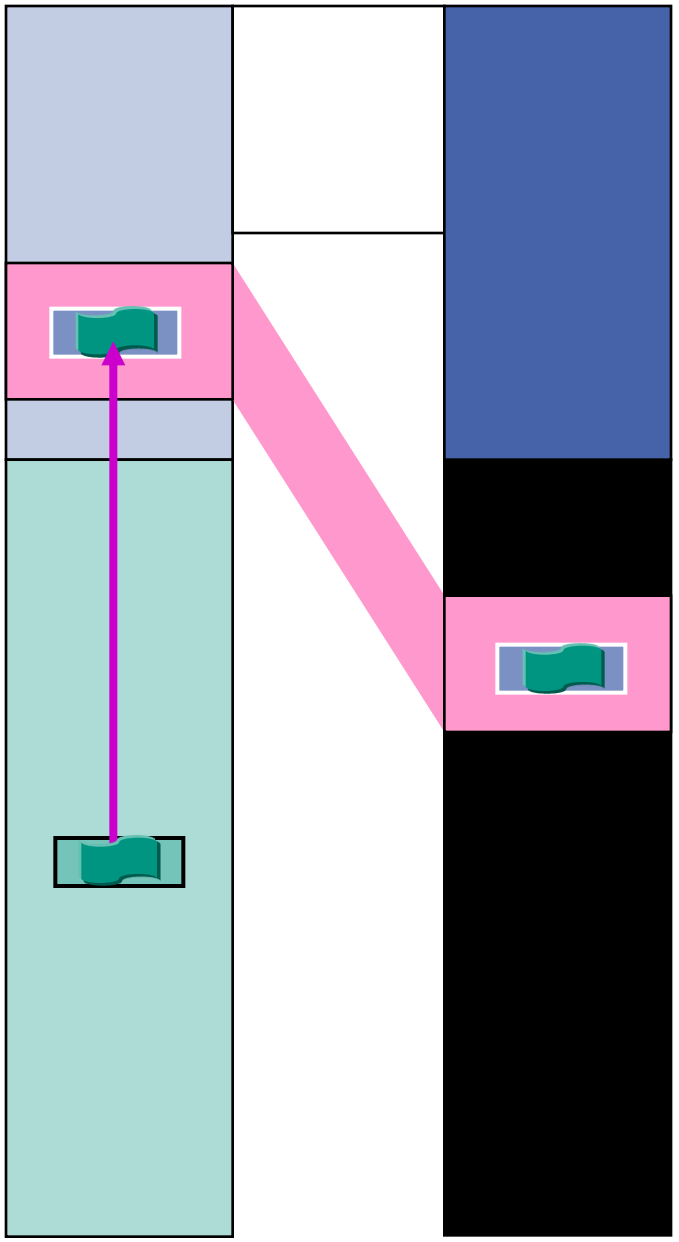
Temporary Mapping

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



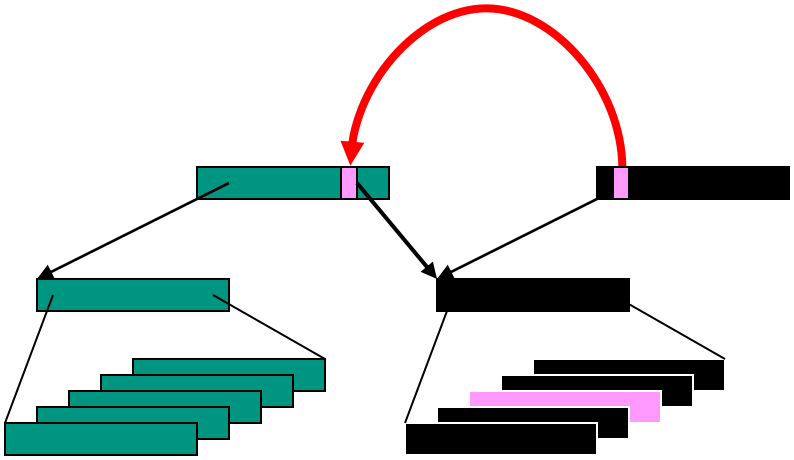
Temporary Mapping

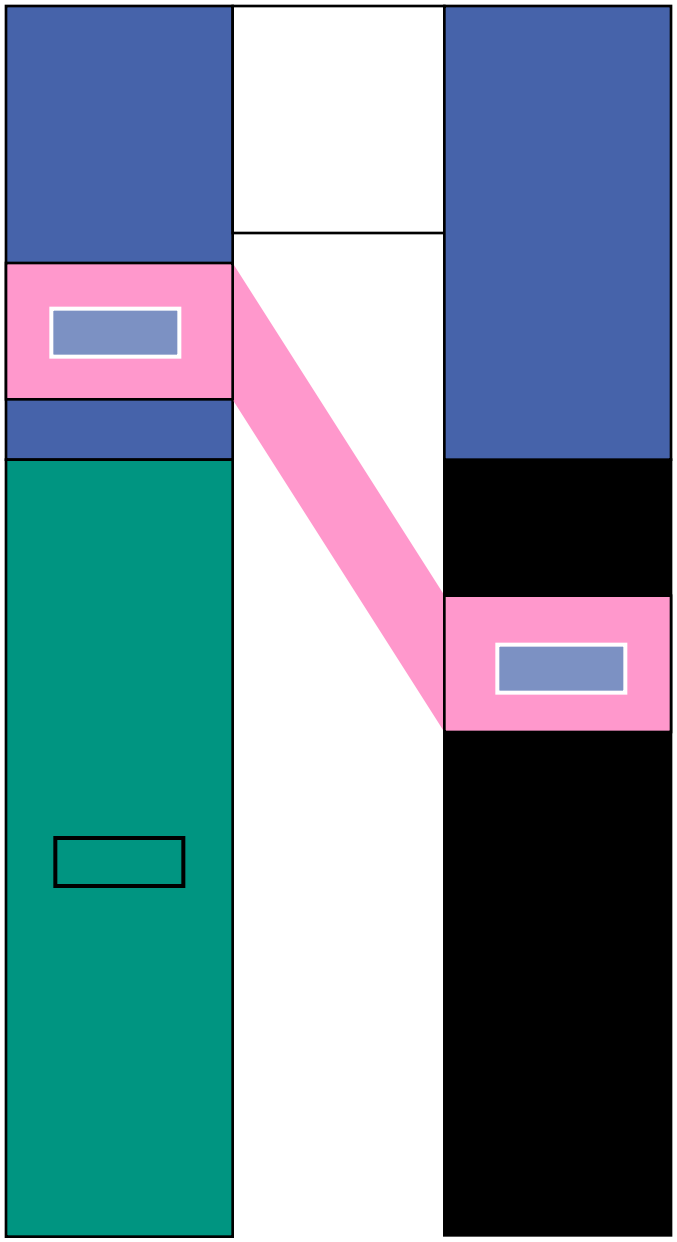
- Select dest area (2x4 MB)
- Map into source AS (kernel)
- Copy data
- Switch to dest space



Temporary Mapping

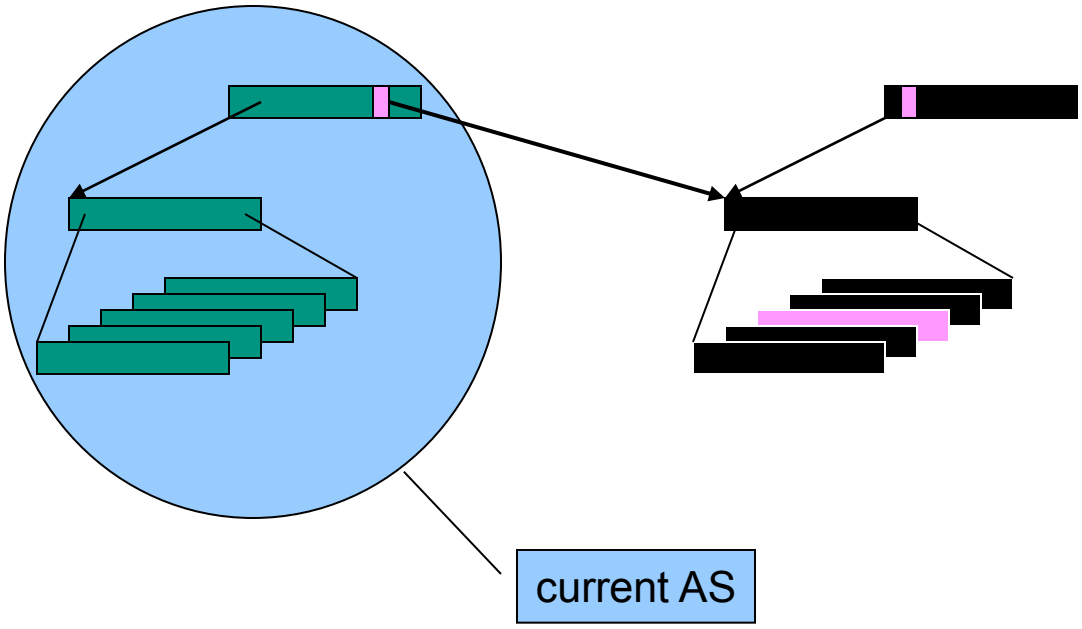
- Copy 2 page directory entries (PDEs) from dest
  - Addresses in temporary mapping area are resolved using dest's page *tables*

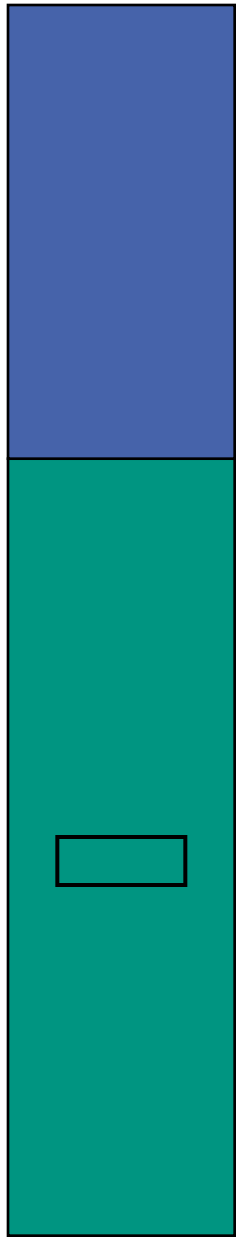




Temporary Mapping

- Problems
  - Multiple threads per AS
  - Mappings might change while message is copied

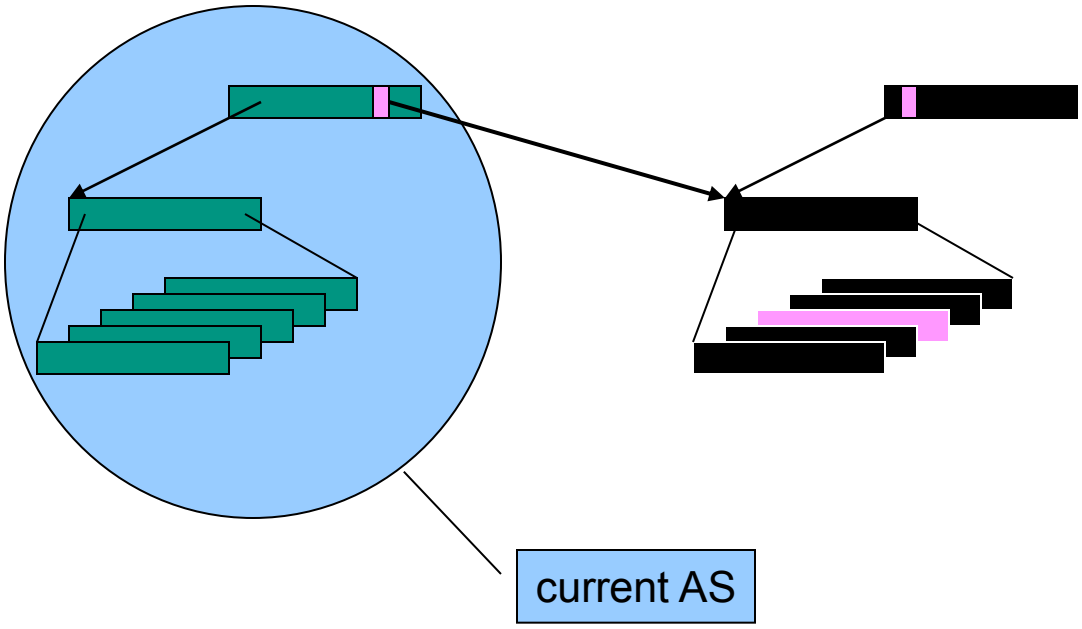


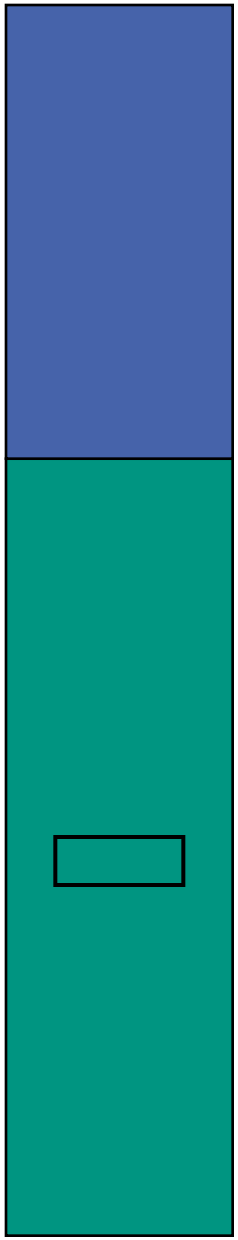


- How long to keep PTE?
- What about TLB?

Temporary Mapping

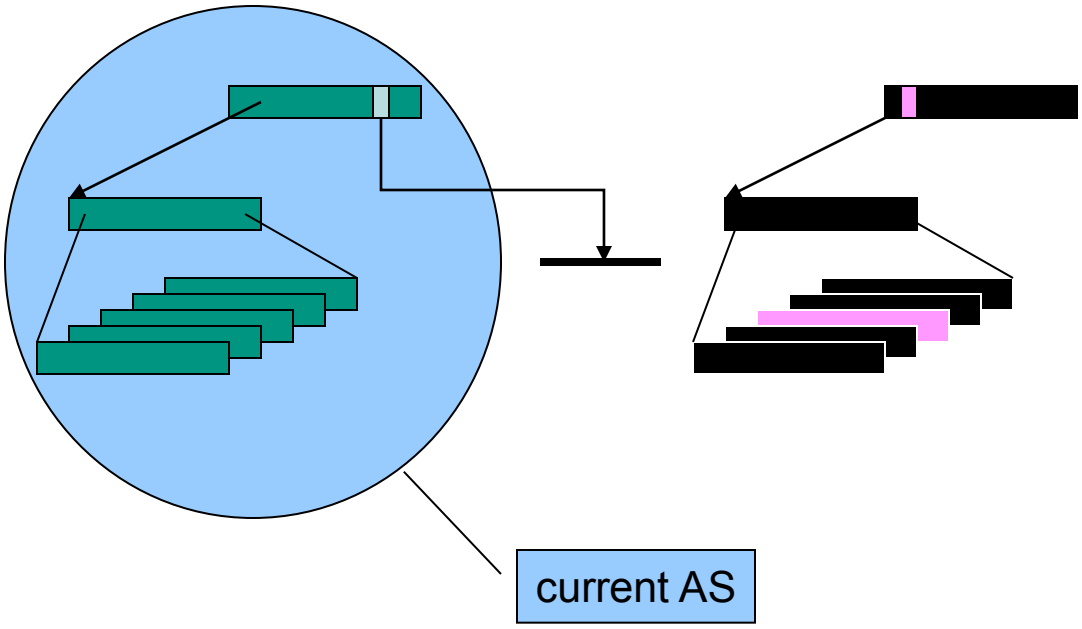
- When switching threads during IPC

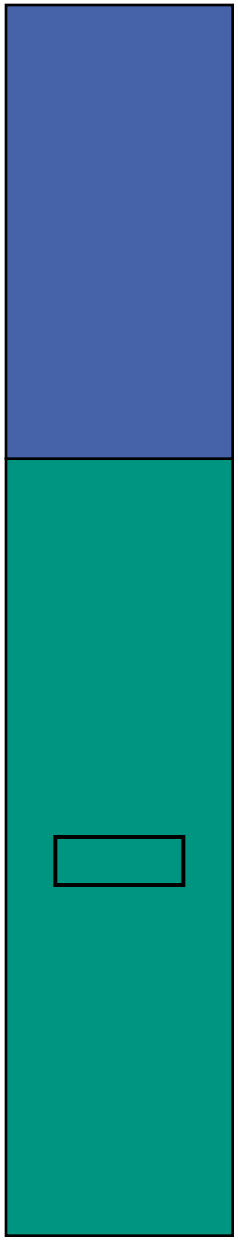




Temporary Mapping

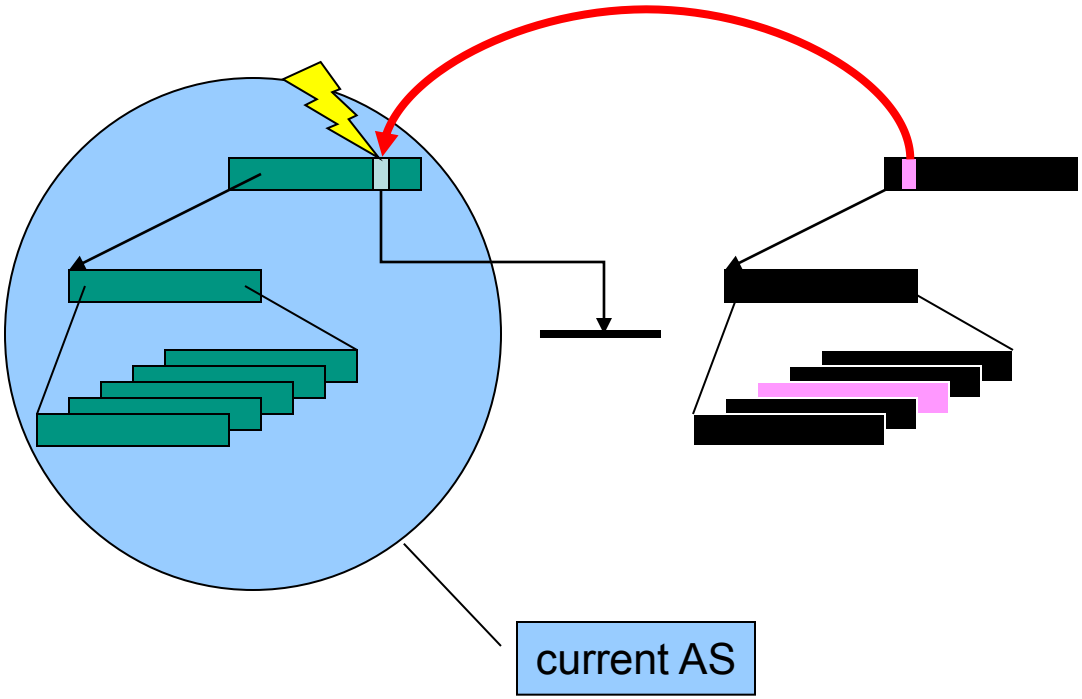
- When switching threads during IPC
  - Invalidate PDE
  - Flush TLB

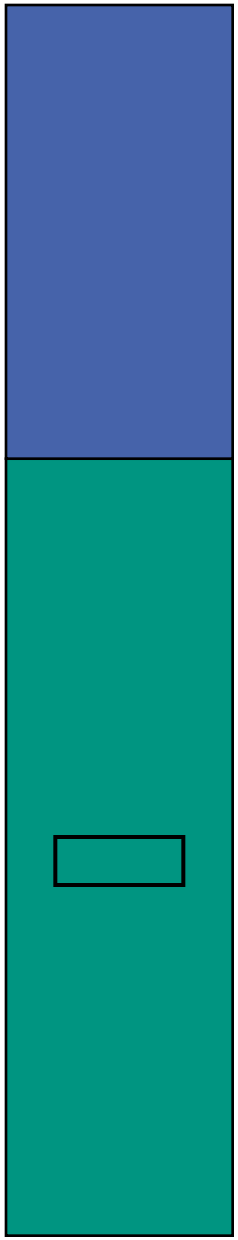




Temporary Mapping

- When returning to a thread
  - Page Fault in Copy Area
  - (Re)copy PDE from receiver

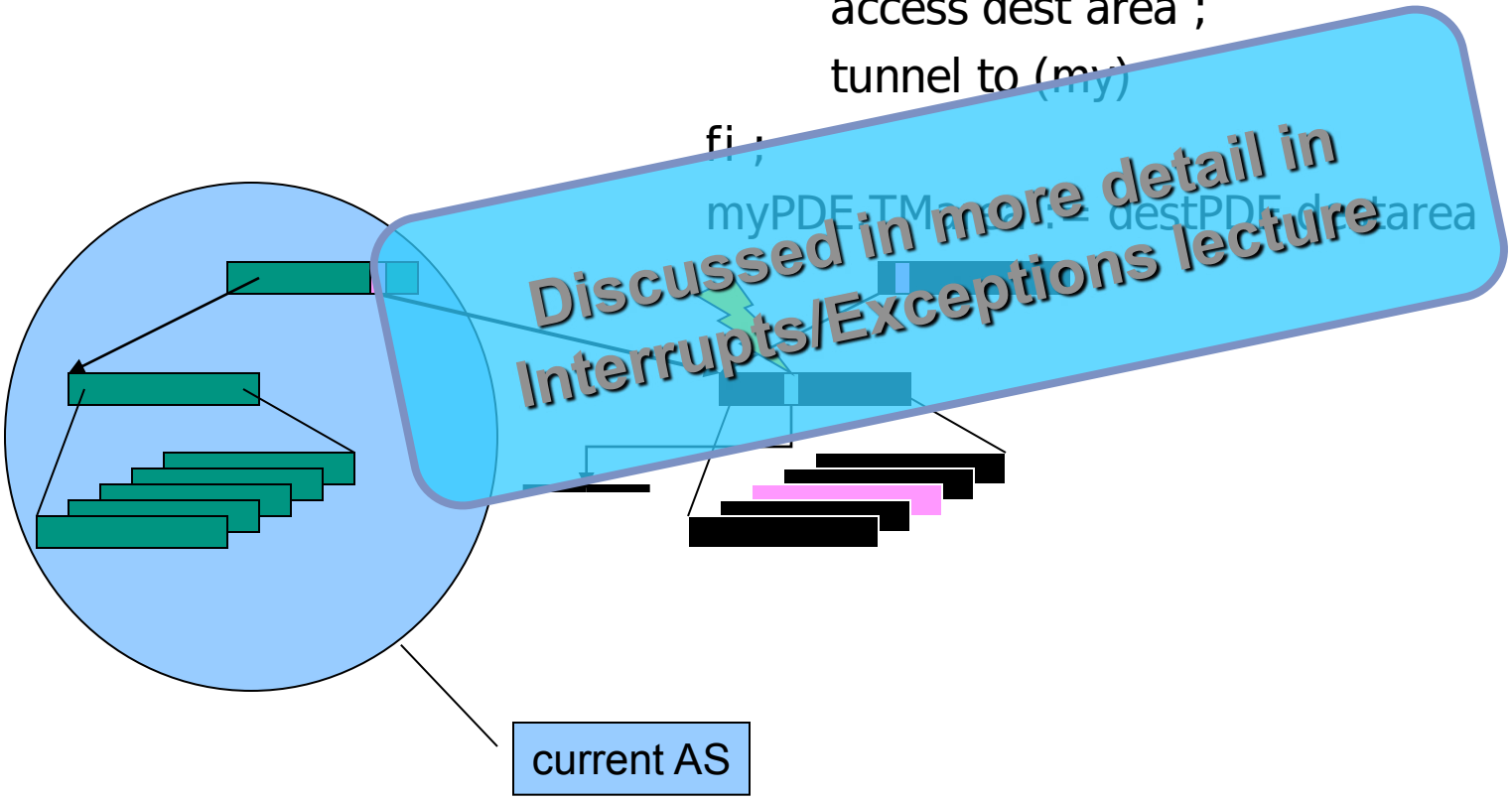




Temporary Mapping

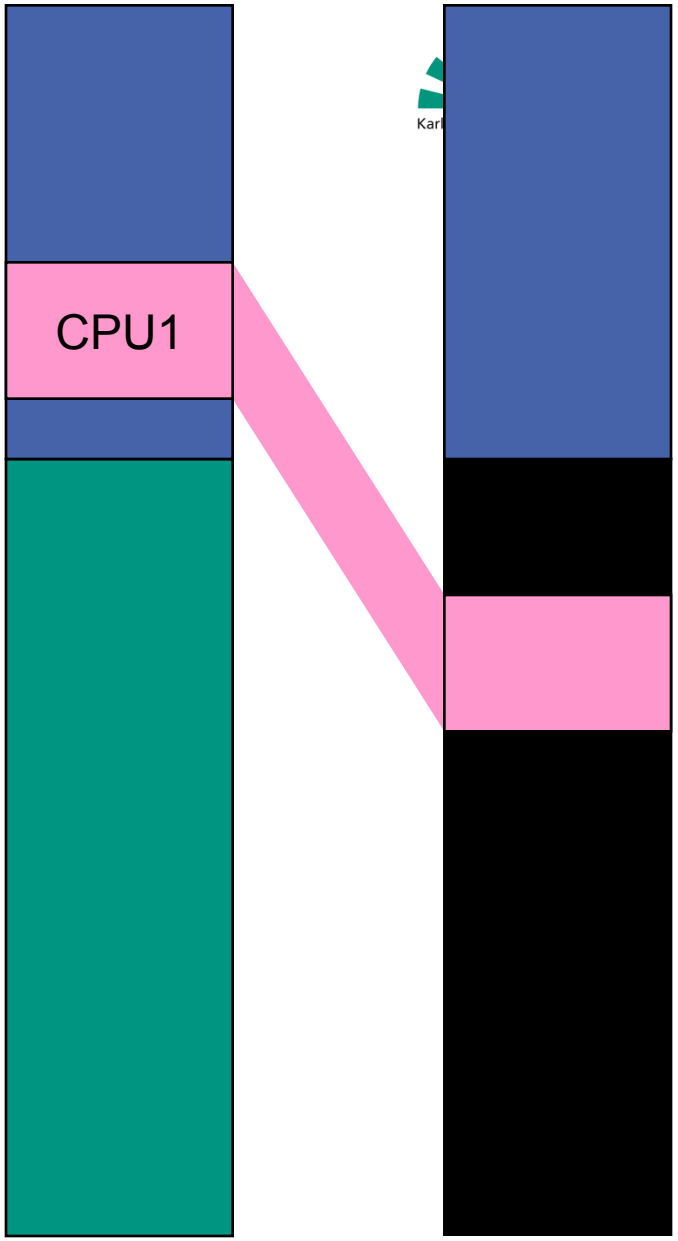
- **Page Fault Resolution:**

```
TM area PF:
    if myPDE.TMarea = destPDE.destarea then
            tunnel to (partner) ;
            access dest area ;
            tunnel to (my)
    fi ;
    myPDE.TMarea := destPDE.destarea
```



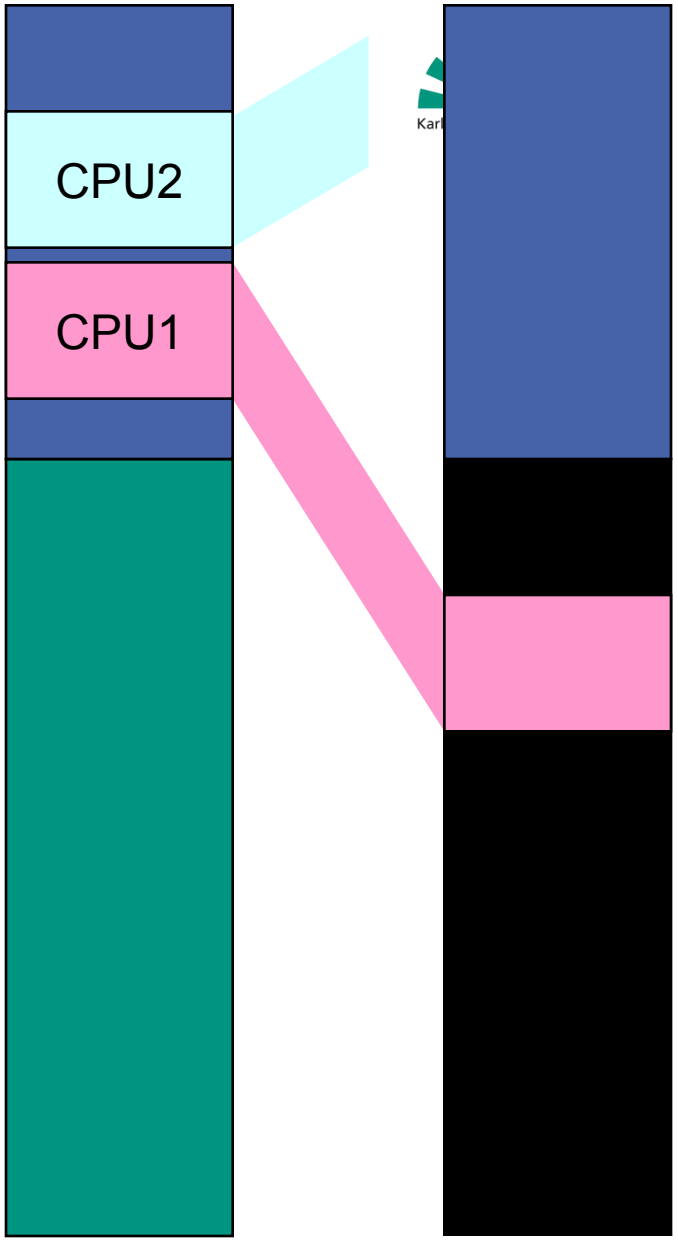
Temporary Mapping

- SMP



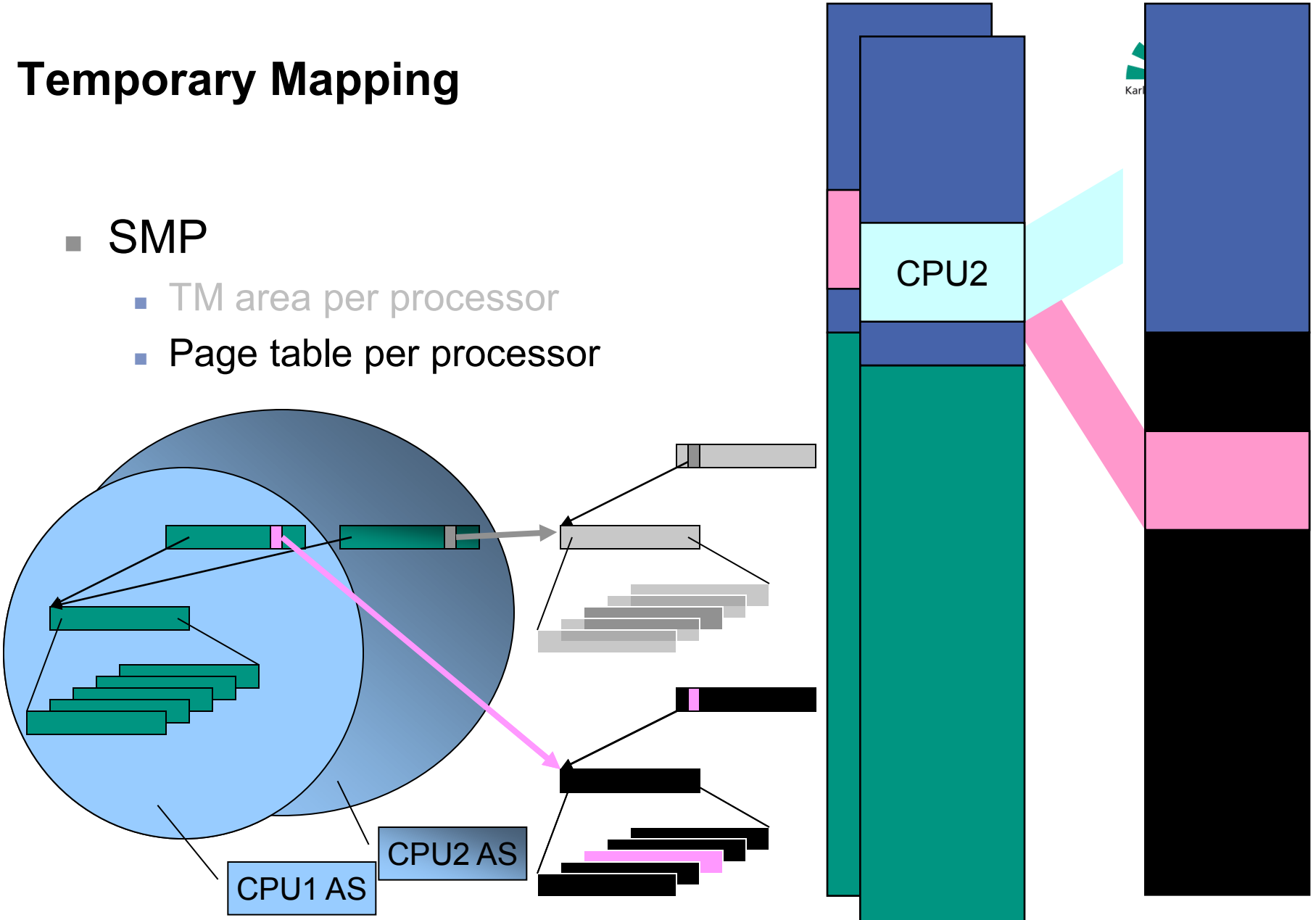
Temporary Mapping

- SMP
  - TM area per processor



Temporary Mapping

- SMP
  - TM area per processor
  - Page table per processor



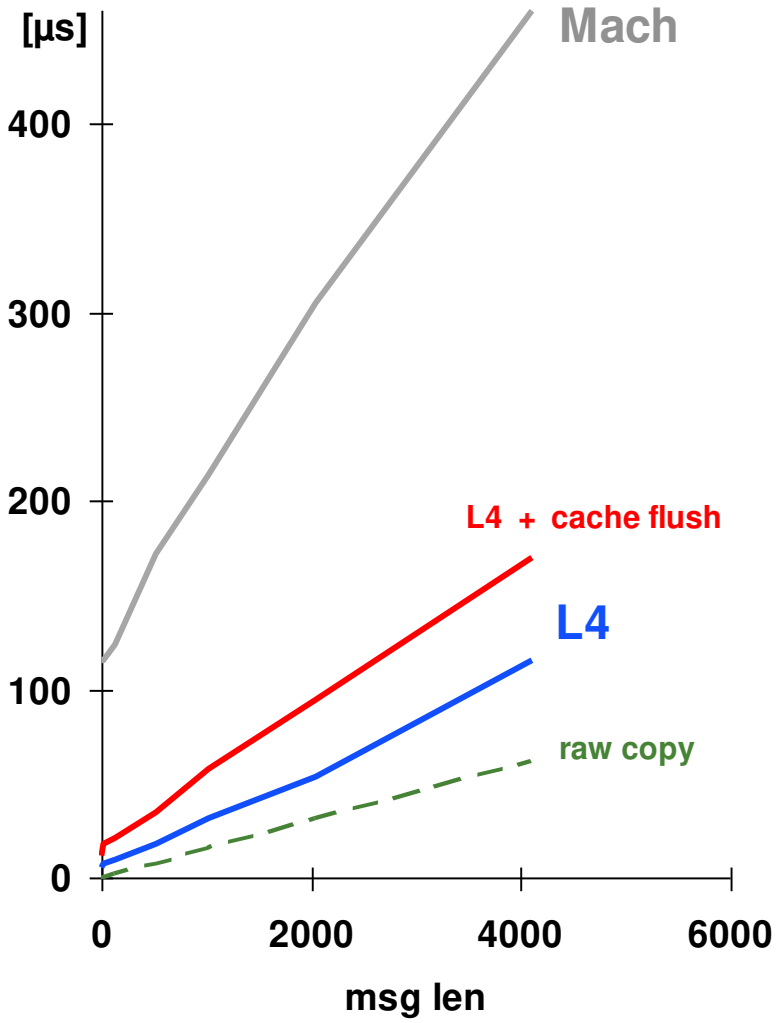
Cost Estimates for Copying *n* Words

	Copy in - copy out	Temporary mapping
R/W operations	$2 \times 2n$	$2n$
Cache lines	$3 \times n/8$	$2 \times n/8$
Small n overhead cache misses	$n/8$	0
Large n cache misses	$4 \times n/8$	$2 \times n/8$
Overhead TLB misses	2	n / (words per page)
Startup instructions	0	~ 50

(assuming 8 words/cache line)

486 IPC Cost

- Mach: Copy in/out
- L4: Temp. mapping

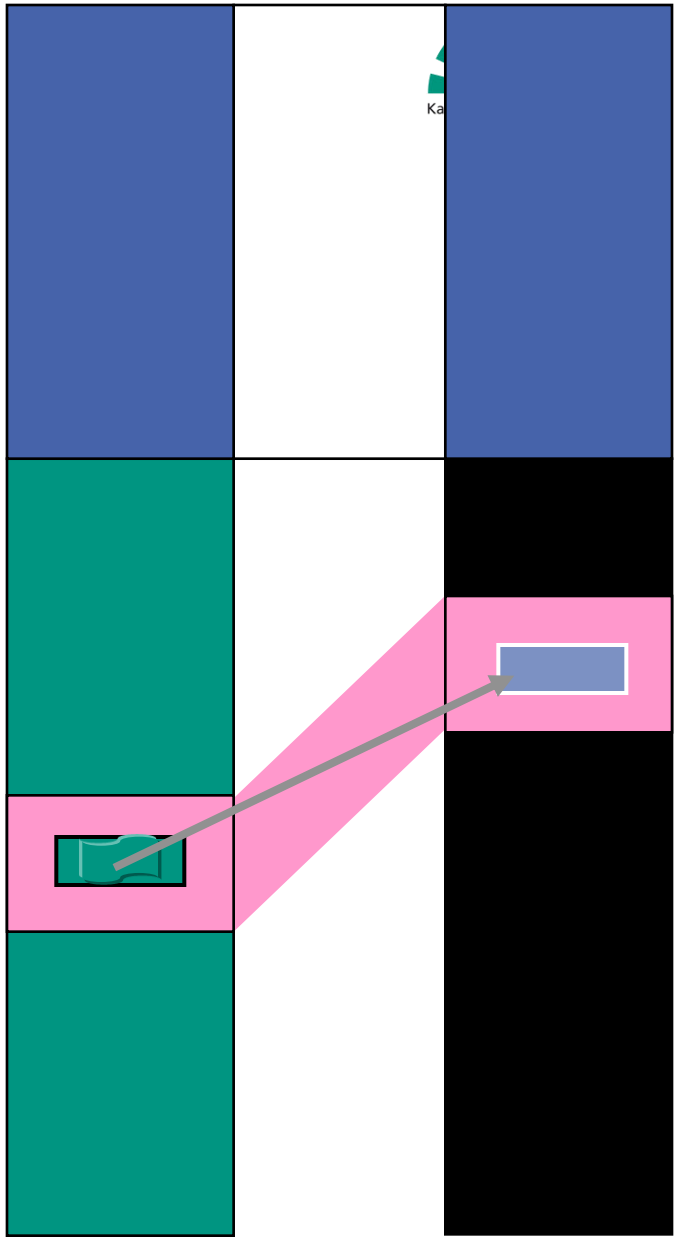


String IPC: Better than shared memory?

- 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!



Summary

- IPC: Single most important operation in µ-Kernels
  - ➔Structure entire kernel for fast IPC!
- Short IPC
  - Payload in registers
  - Context switch, just leave payload alone
  - Avoid memory references
  - SMP: Use L1 cache
- String IPC
  - Temp mapping ➔ only one copy
  - But: Pagefault tunneling, copy area multiplexing
  - Only implemented for backward compatibility