Karlsruhe Institute of Technology

Microkernel Construction
I.4 – IPC Functionality & Interface

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

IPC Primitives

- Send to
  (a specified thread)
- Receive from
  (a specified thread)
- Receive
  (from any thread)

- Two threads communicate
- No interference from other threads
- Other threads block until it's their turn
- Problem
  - How to communicate with a thread unknown a priori
    (e.g., a server's clients)

IPC Primitives

- Send to
  (a specified thread)
- Receive from
  (a specified thread)
- Receive
  (from any thread)
- Call
  (send to, receive from specified thread)
- Send to & Receive (from)
  (send to, receive from any/specified thread)

- Scenario
  - A client thread sends a message to a server expecting a response
  - The server replies expecting the client thread to be ready to receive
- Problem
  - The client might be preempted between the send to and receive from

IPC Primitives

- Send to
  (a specified thread)
- Receive from
  (a specified thread)
- Receive
  (from any thread)
- Call
  (send to, receive from specified thread)
- Send to & Receive (from)
  (send to, receive from any/specified thread)
- Send async to
  (a specified thread)

- Scenario
  - Thread A wants to notify thread B of an event (e.g., interrupt)
  - Thread B does not need to process the event right away
- Problem
  - With synchronous IPC, thread A has to block until thread B is ready

Message Types

- Registers
  - Short messages, avoid memory during IPC
  - Guaranteed to avoid user-level page faults during IPC
- Strings (optional)
  - In-memory messages copied from sender to receiver
  - May incur user-level page faults during copy operation
- Mappings (optional)
  - Messages that map pages from sender to receiver
  - Can map other resources too (e.g., capabilities)

IPC – API

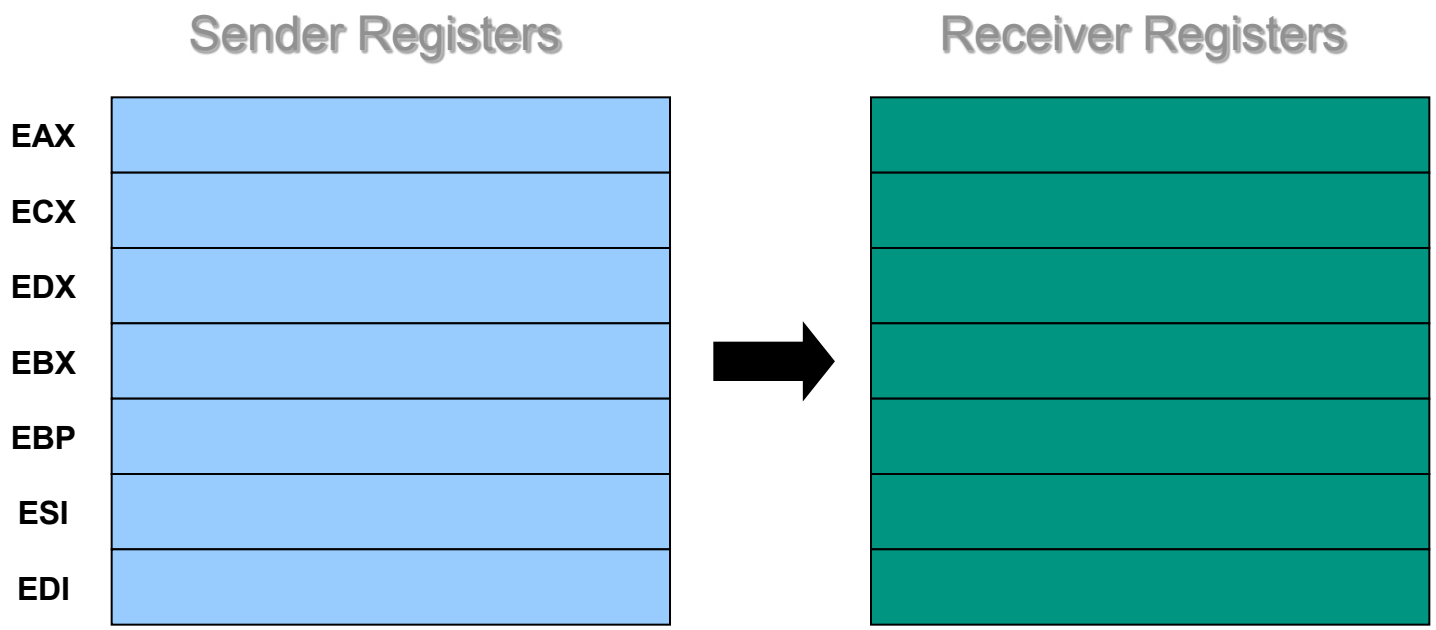
- Operations
  - Send to
  - Receive from
  - Receive
  - Call
  - Send to & Receive
  - Send to & Receive from
  - Send async
- Message Types
  - Registers
  - Strings
  - Mappings

IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination endpoint
- Source endpoint
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender endpoint

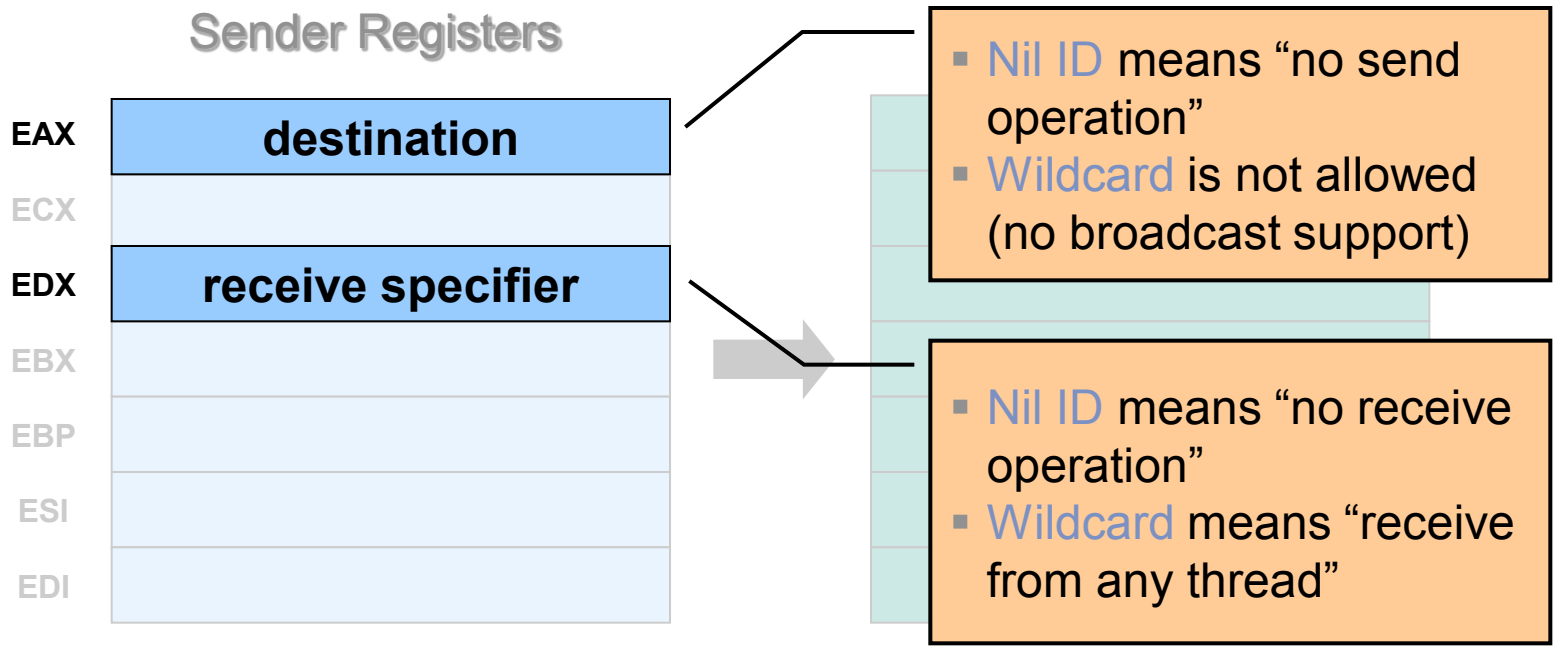
Ideally Encoded in Registers

- Parameters in registers whenever possible
- Make frequent operations simple and fast



Send and Receive Encoding

- 0 (Nil ID) is a reserved thread ID
- Define -1 as a wildcard thread ID



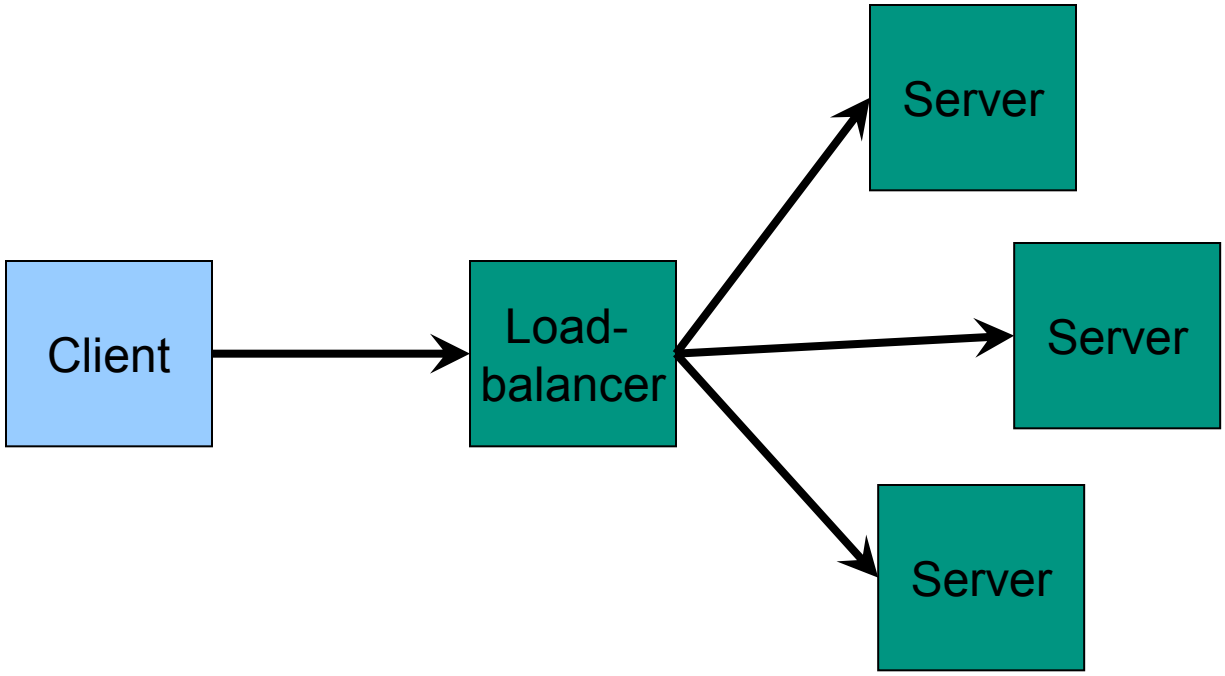
Why Use a Single Call Instead of Many?

- The implementation of the individual send and receive is very similar to the combined send and receive
  - We can use the same code
    - We reduce cache footprint of the code
    - We make applications more likely to be in cache
- L4 only implements combined "send to A and receive from B" syscall
  - A may but need not be equal to B
  - A or B may be 0 to avoid a send or receive phase
    - A == B == 0 is just a costly no-operation

IPC endpoints

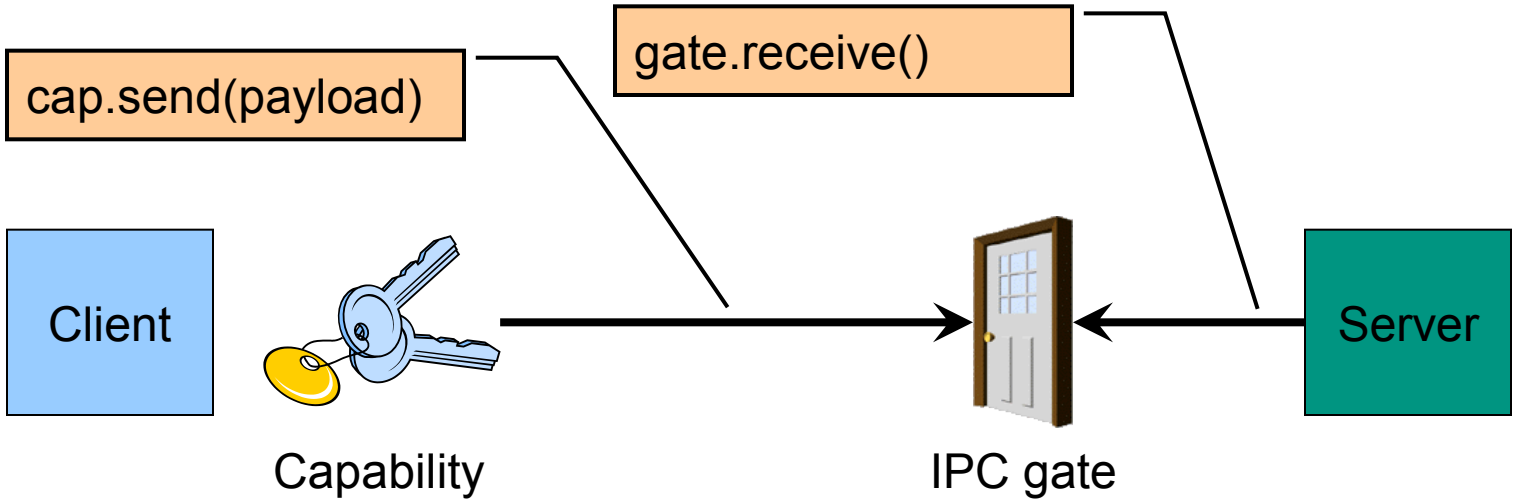
- How do we specify the destination of an IPC message?
- Idea 1: Use thread ID

- Problem: Must specify exactly one receiver

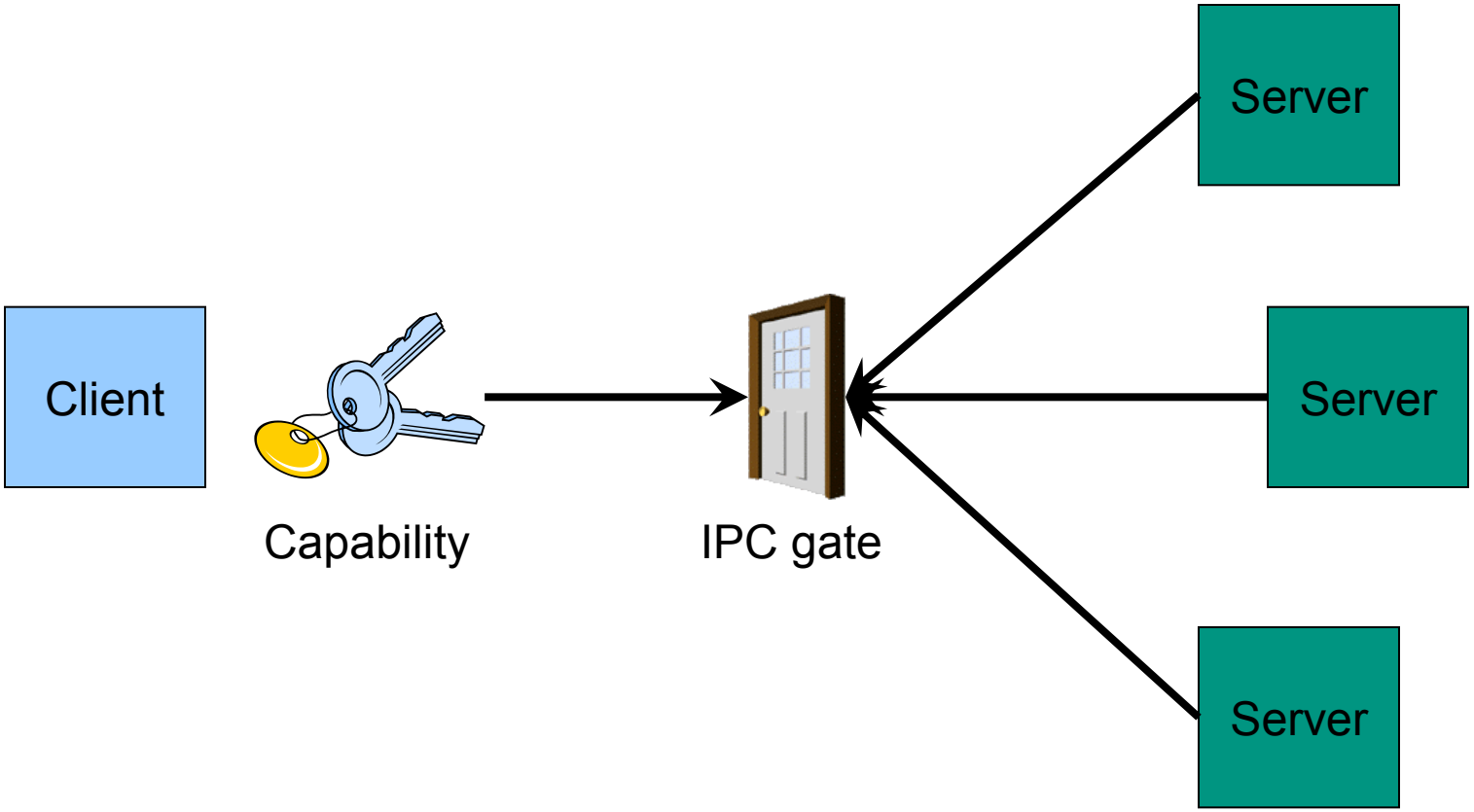


IPC gates

- Idea 2: Decouple IPC endpoints from threads



IPC gates



IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination endpoint
- Source endpoint

IPC syscall

- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender endpoint

- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string

Message Transfer

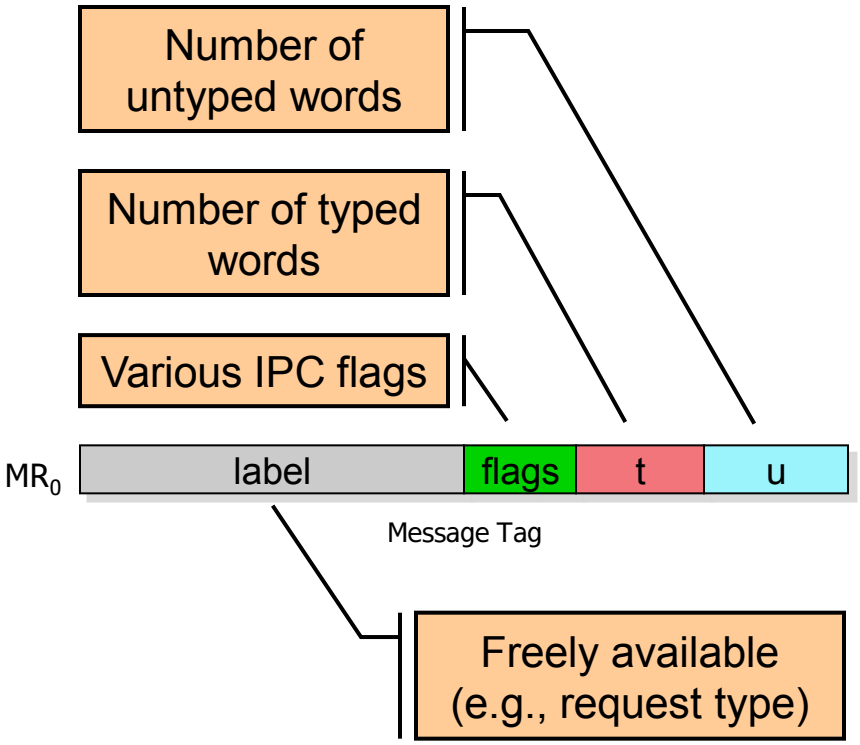
- Assume that 64 extra registers are available
  - Name them MR_0 ... MR_{63} (message register 0 … 63)
  - Only used message registers are transferred during IPC (see MR_0)

IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination endpoint
- Source endpoint
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender endpoint

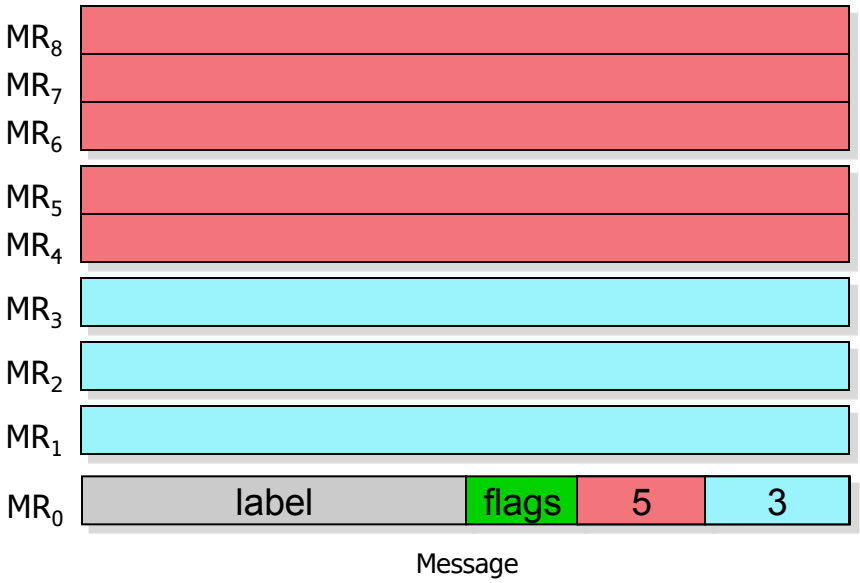
Message Construction

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



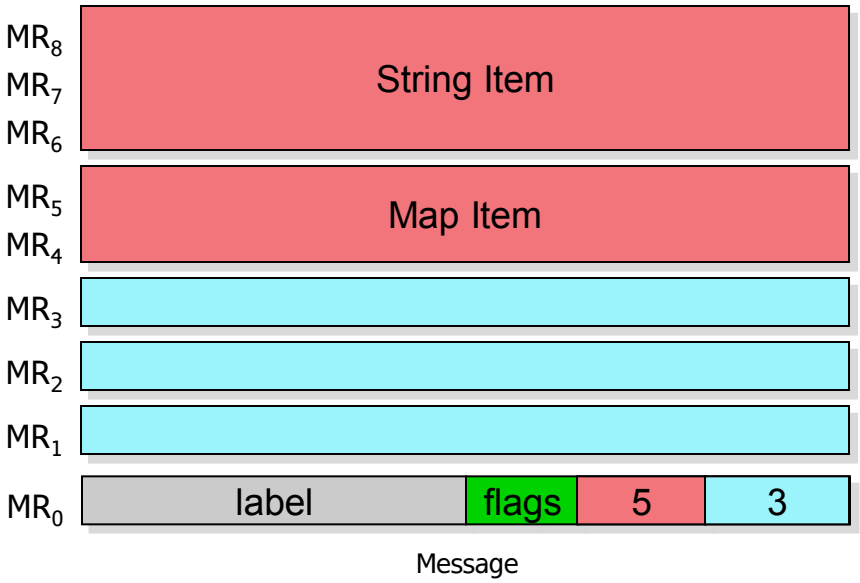
Message Construction

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



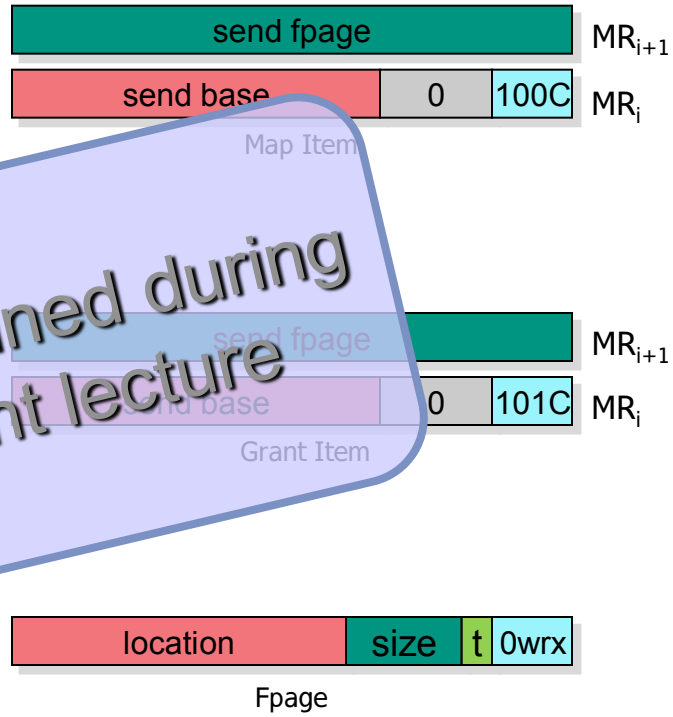
Message Construction

- Typed items occupy one or more words
- Four common item types
  - 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

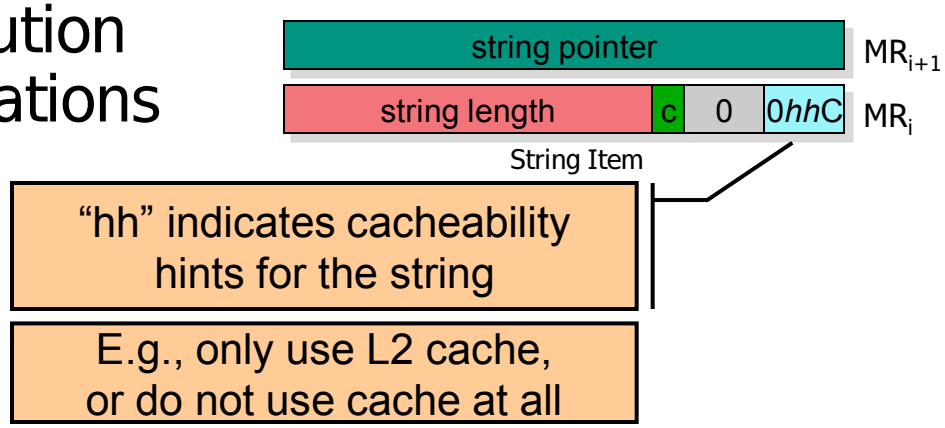
- Two words
  - Send base
  - Fpage
- Lower bits of send base indicates map or grant item



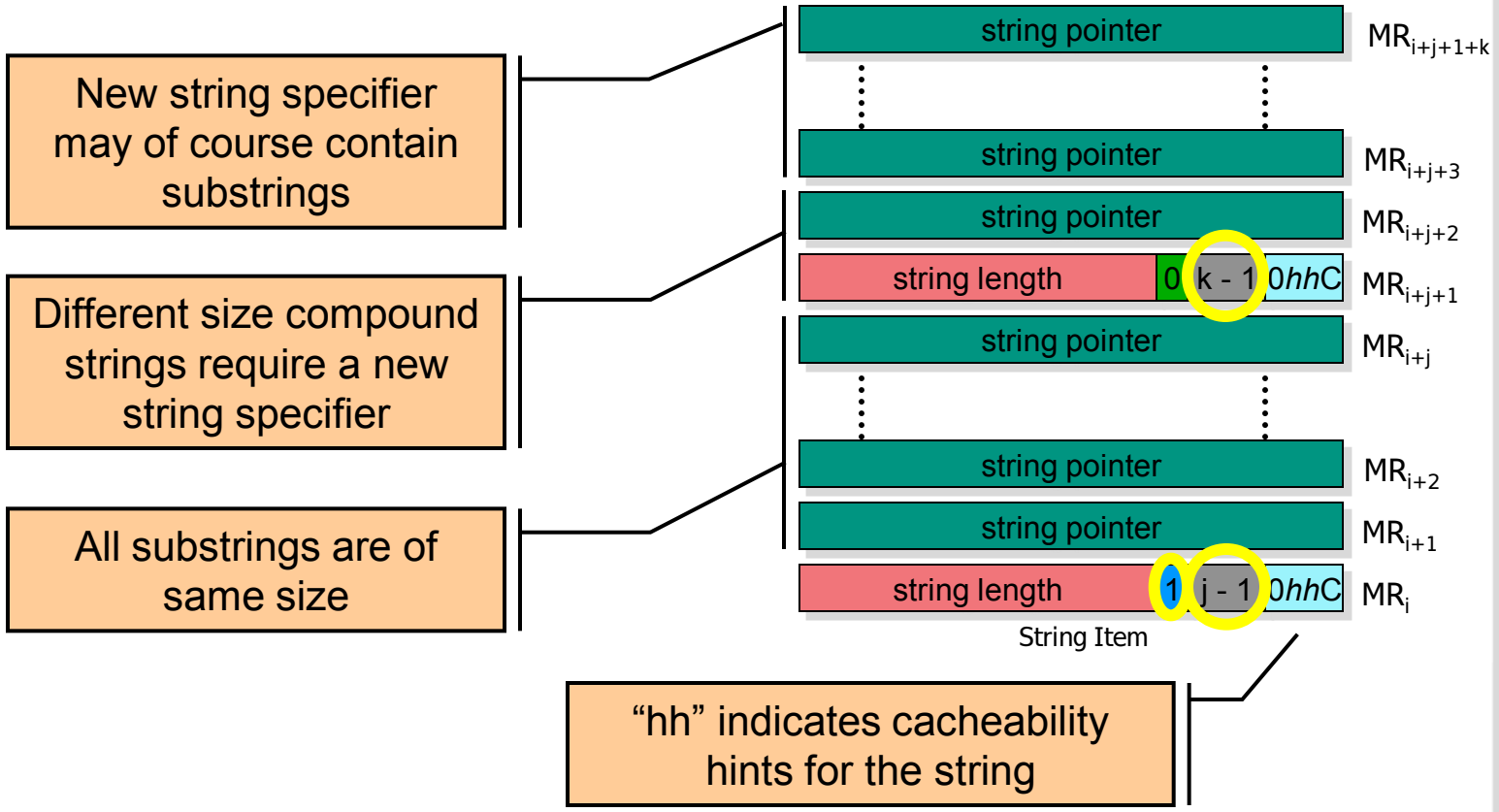
Semantics will be explained during memory management lecture

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 Items



IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination endpoint
- Source endpoint
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender endpoint

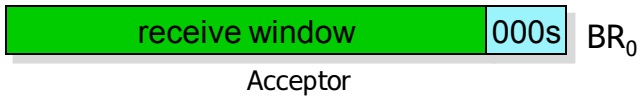
String Reception

- Assume that 34 extra registers are available
  - Name them BR_0 ... BR_{33} (buffer register 0 ... 33)
  - Buffer registers specify
    - Receive strings
    - Receive window for mappings

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)



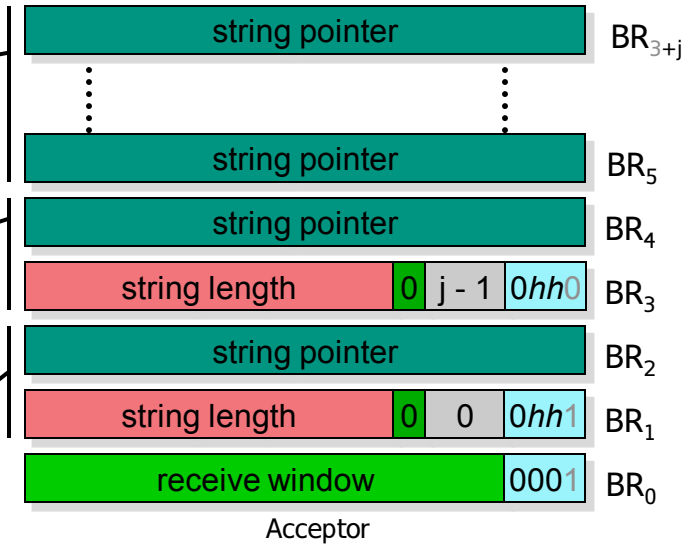
Receiving Messages

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 asynchronous messages

- Kernel must buffer asynchronous messages
- Assume that 1 extra register is available
- Limit message payload to 1 register (MR_1)
  - ORed to receive register
- Two ways to receive:
  - Synchronously (block on specific bit mask)
  - Asynchronously (read register)

IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination endpoint
- Source endpoint
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender endpoint

Problem

- How to we deal with threads that are
  - Uncooperative
  - Malfunctioning
  - Malicious?

- How to prevent an IPC operation from never completing?

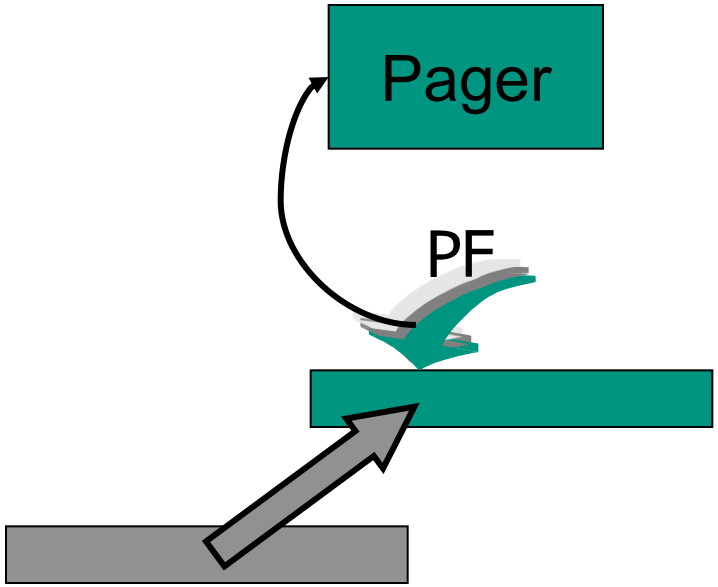
IPC – API

- Timeouts (v2, vX.0)
  - snd timeout, rcv timeout

IPC – API

- Timeouts (v2, vX.0)
  - snd timeout, rcv timeout
  - snd-pf
    - specified by sender

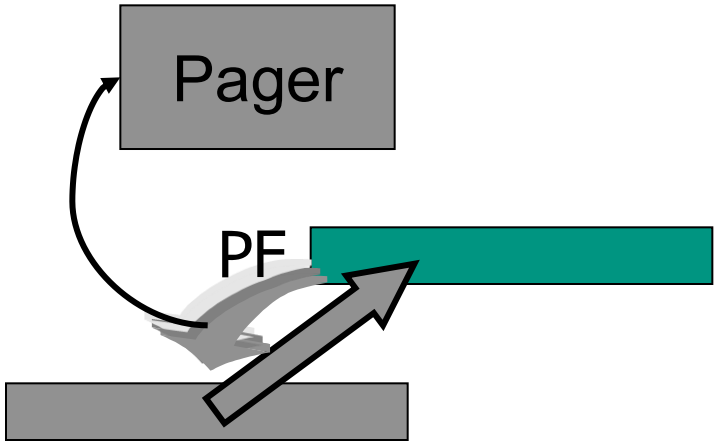
- Attack through receiver's pager



IPC – API

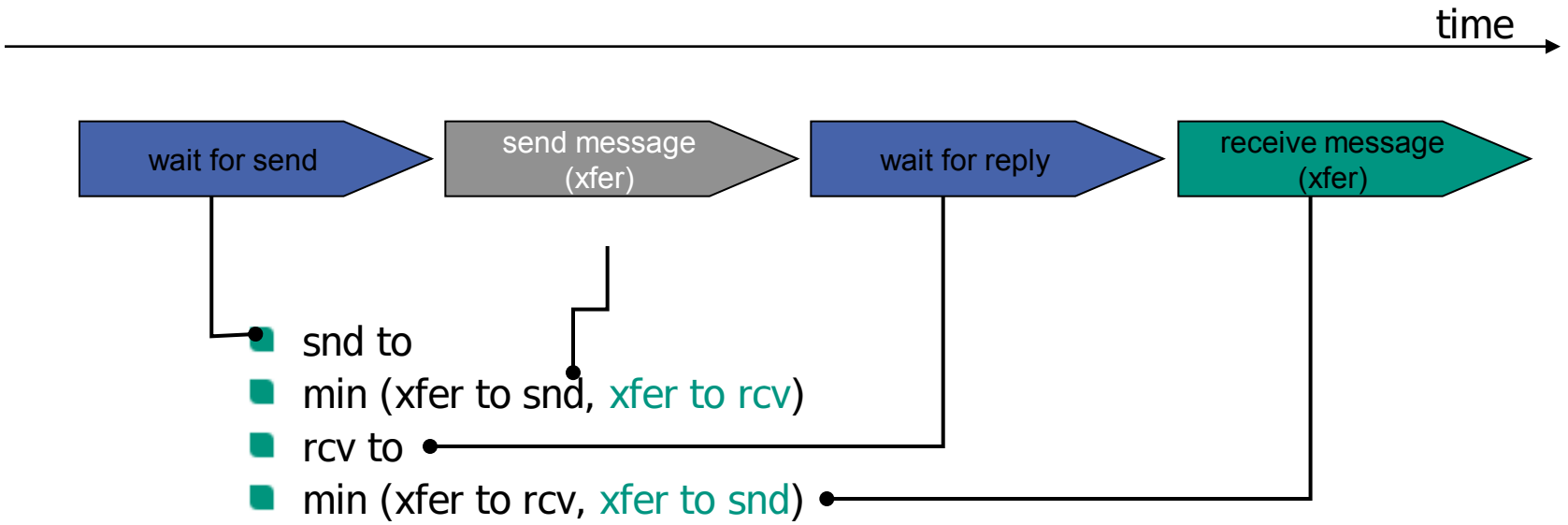
- Timeouts (v2, vX.0)
  - snd timeout, rcv timeout
  - snd-pf / rcv-pf
    - specified by receiver

- Attack through sender's pager



IPC – API

- Timeouts (vX.2, v4)
  - snd timeout, rcv timeout, xfer timeout snd, xfer timeout rcv

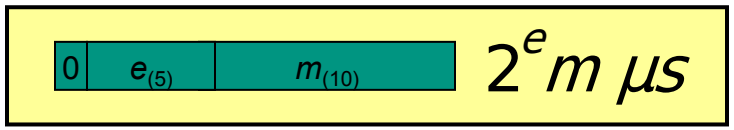


(specified by the partner thread)

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

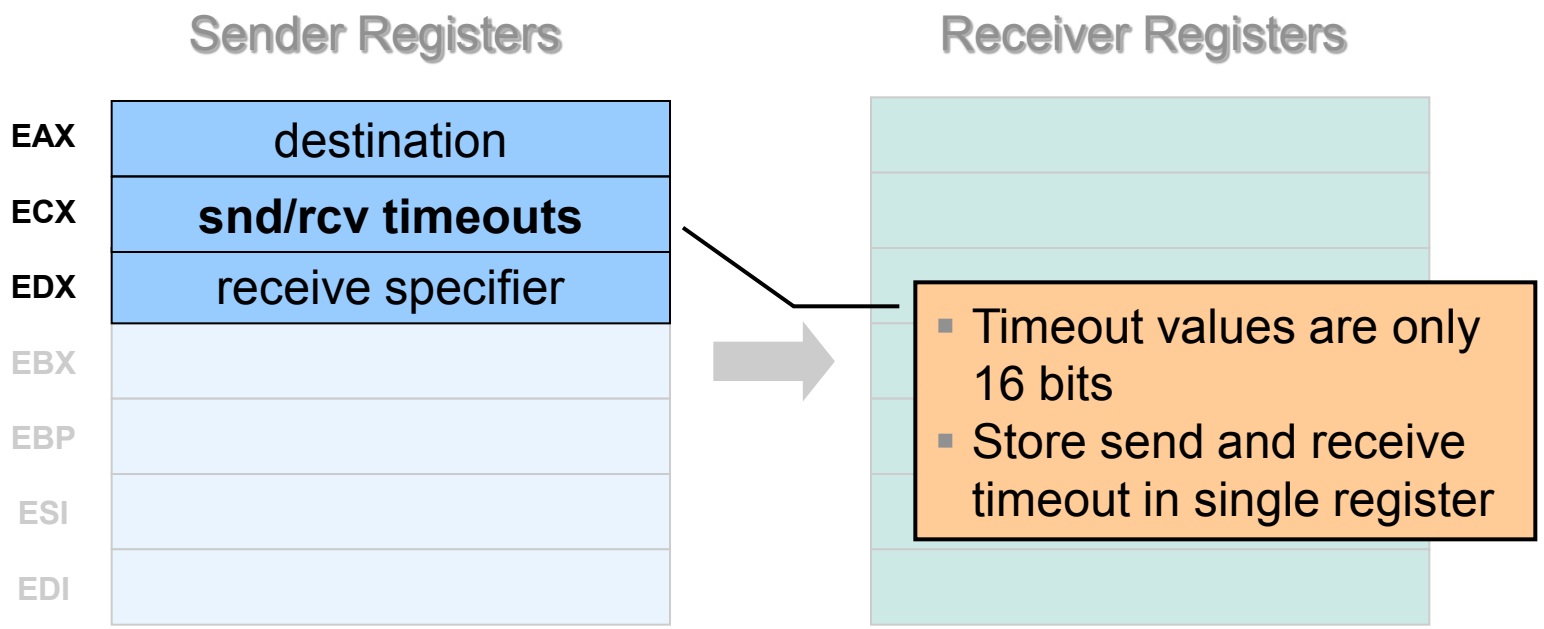
- Timeout values
  - ∞ (infinite)
    - Client waiting for a (trusted) server
  - 0 (zero)
    - Server responding to a client
    - Polling
  - Specific time
    - 1 µs – 610 h (log)

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

- SeL4: 1 bit timeout (zero or infinite)

Timeouts

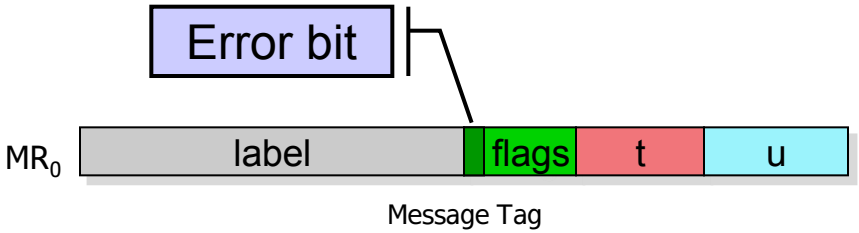
- Send and receive timeouts are the important ones
  - Xfer timeouts only needed during string transfer
  - Store xfer timeouts in predefined memory location



IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination endpoint
- Source endpoint
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender endpoint

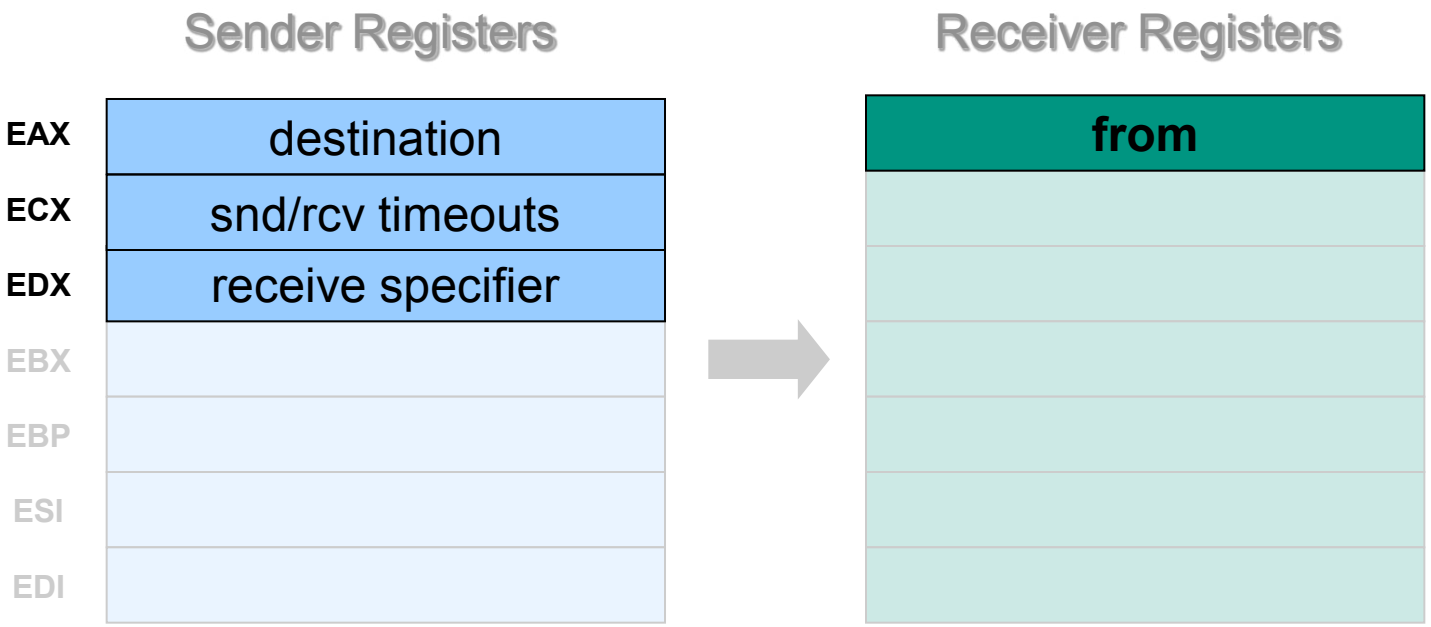
IPC Result



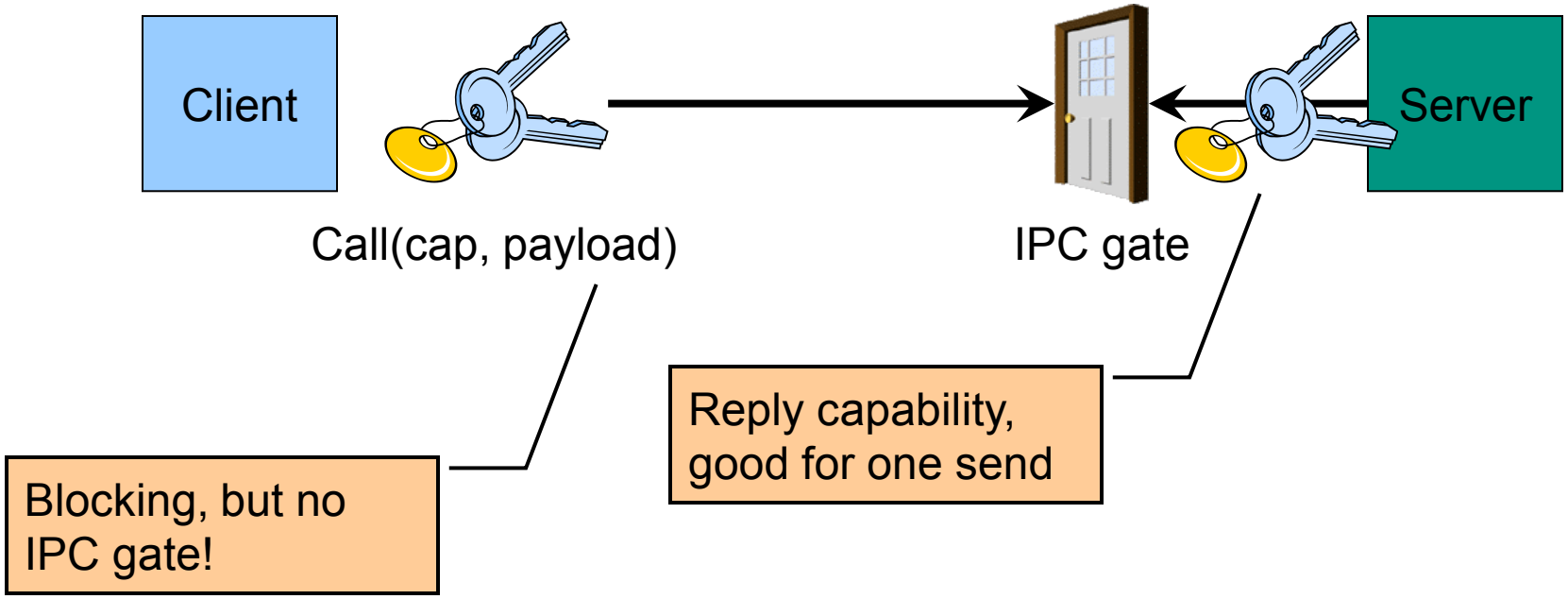
- Error conditions are exceptional
  - Not common case
  - No need to optimize for error handling
- Bit in received message tag indicates error
  - Fast check
- Exact error code store in predefined memory location

Thread ID as sender endpoint

- Sender's thread ID stored in register



Capability as sender endpoint



IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender thread ID

Virtual Registers

- What about message and buffer registers?
  - Most architectures do not have 64+34 spare registers
- What about predefined memory locations?
  - Must be thread local

Virtual Registers

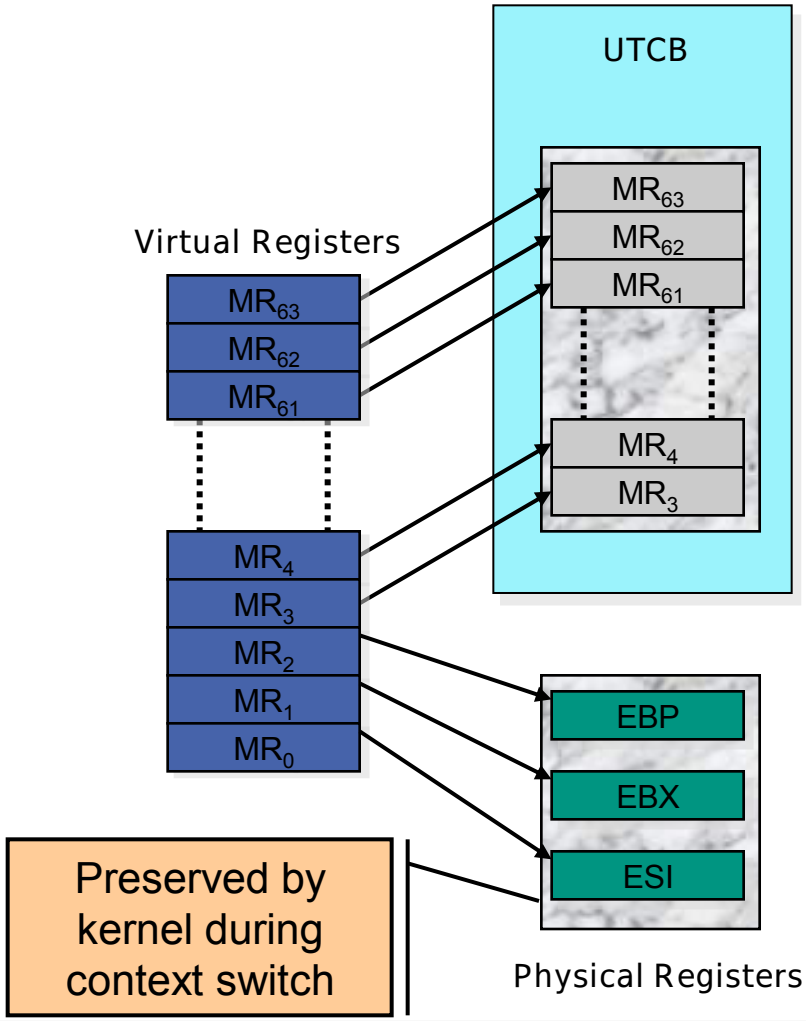
- What about message and buffer registers?
  - Most architectures do not have 64+34 spare registers
- What about predefined memory locations?
  - Must be thread local

Define as Virtual Registers

Define as Virtual Registers

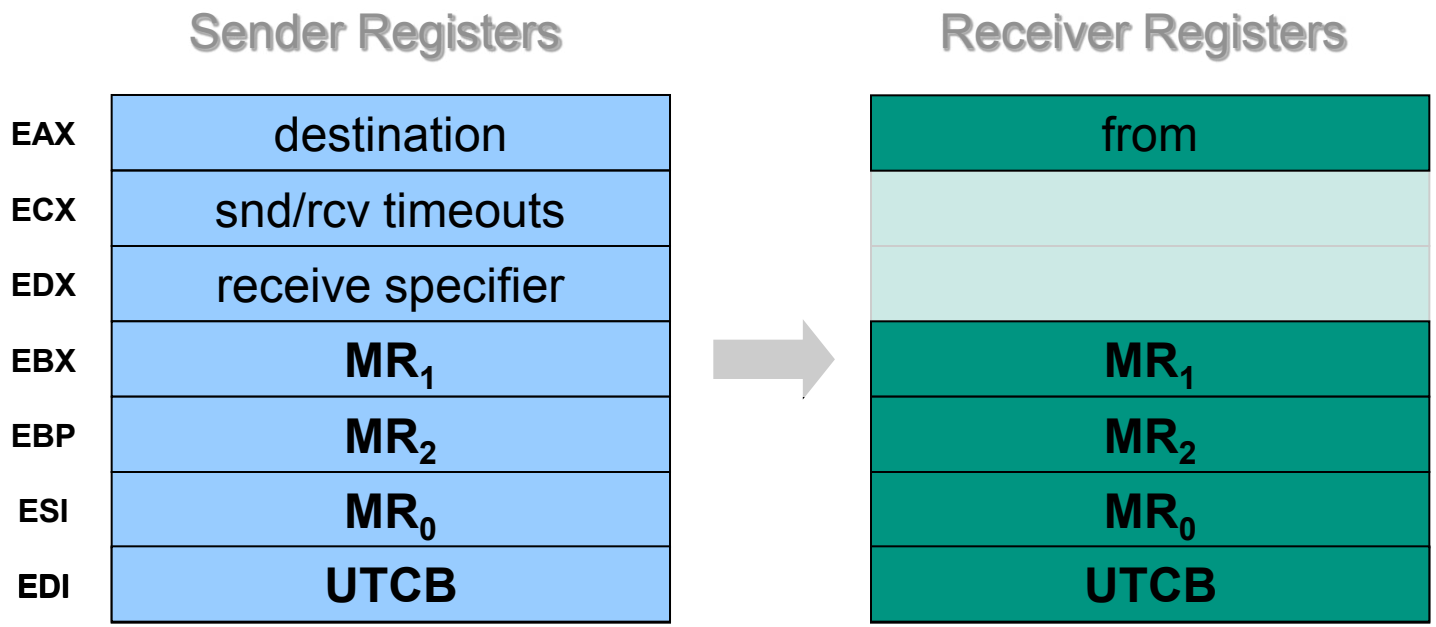
What are Virtual Registers?

- Virtual registers are backed by either
  - Physical registers, or
  - Non-pageable memory
- UTCBs hold the memory backed registers
  - UTCBs are known in user and kernel space
  - UTCBs are thread local
  - UTCBs cannot be paged
    - No page faults
    - Registers always accessible



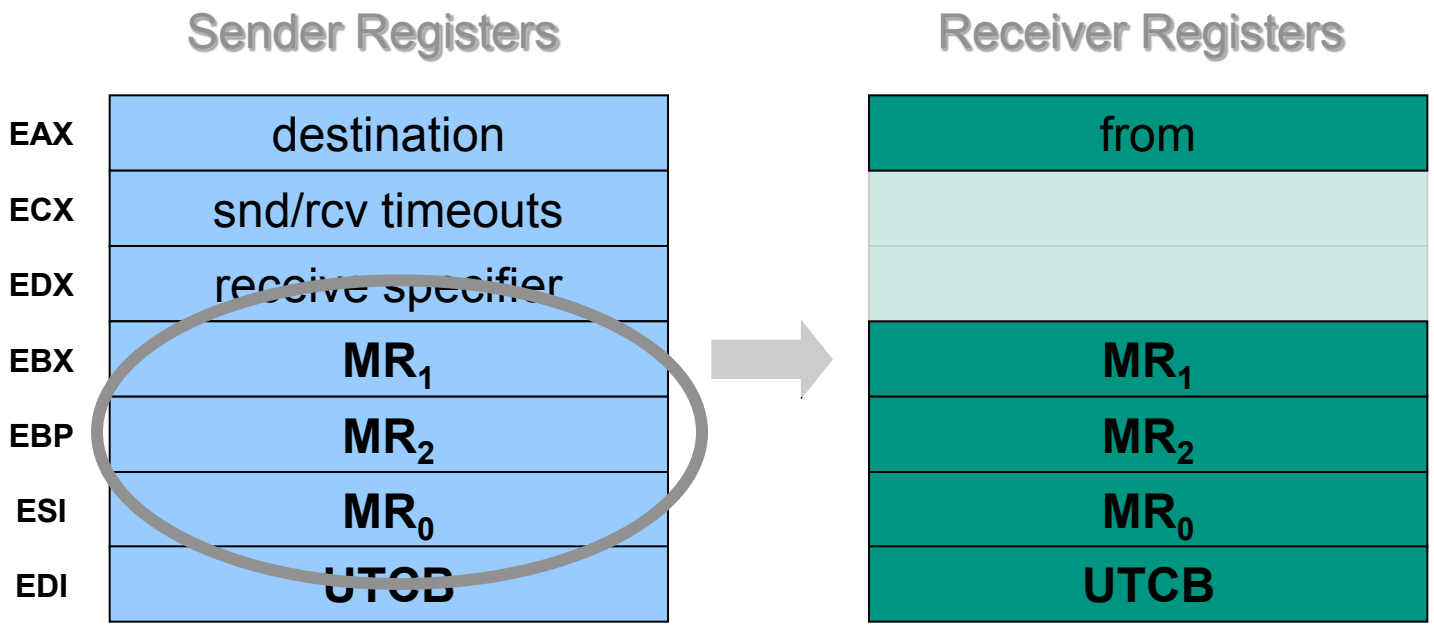
Message Registers and UTCB

- Some MRs are mapped to physical registers
- Kernel will need UTCB pointer anyway – pass it



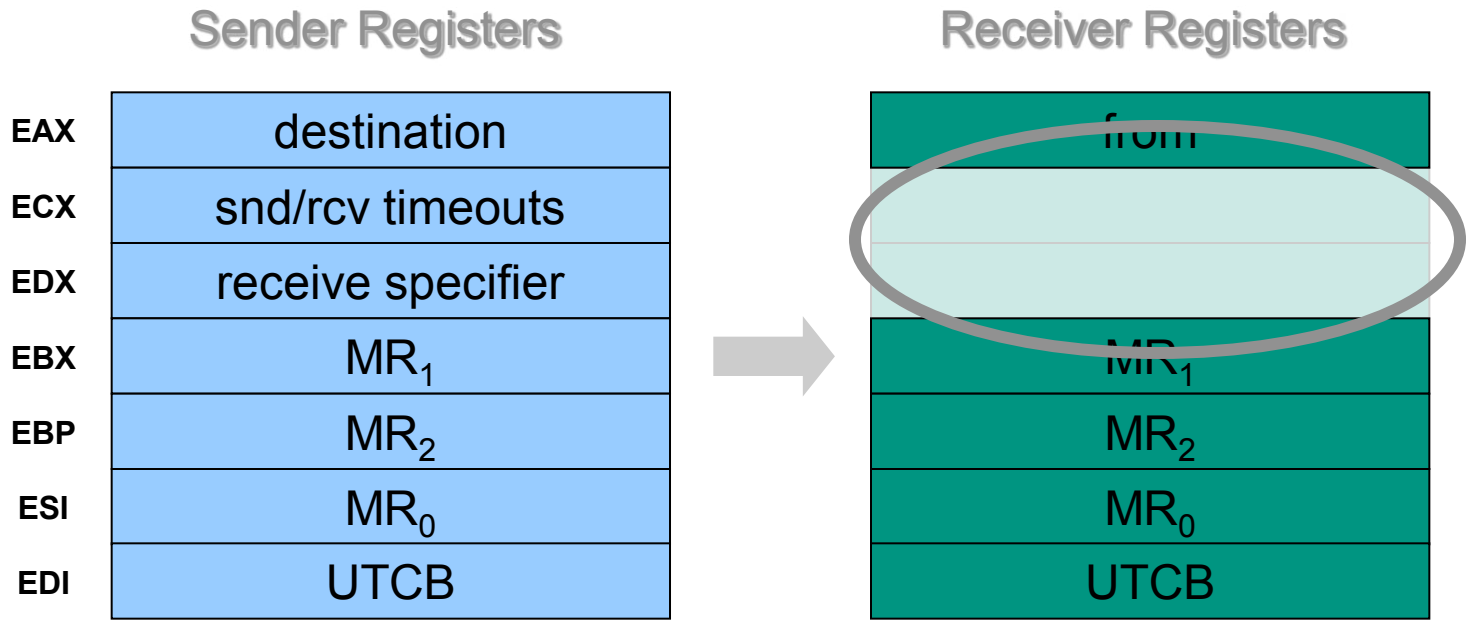
Free Up Registers for Temporary Values

- Kernel needs registers for temporary values
- MR_0, MR_1 and MR_2 are the only values that the kernel may not need



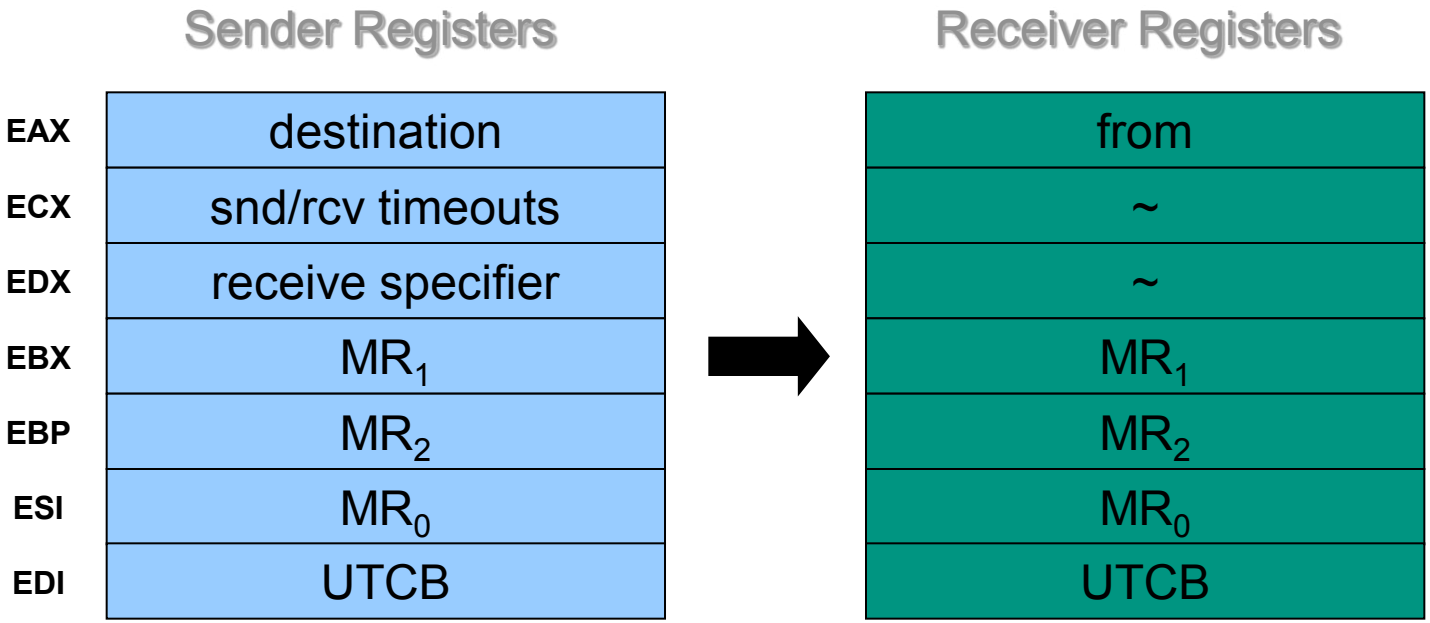
Free Up Registers for Temporary Values

- Sysexit instruction requires
  - ECX = user IP
  - EDX = user SP



IPC Register Encoding

- Parameters in registers whenever possible
- Make frequent operations simple and fast



Summary

- IPC has many parameters
  - Operation (send, receive, combinations)
  - Communication partners
  - Actual message to transfer
- IPC Timeouts required
  - Handle malfunctioning / uncooperative threads
  - Encode as short as possible
- Compact encoding of IPC operation
  - Send and receive specifier
  - Nilthread == no operation

Summary (2)

- Message construction in virtual Message Registers
  - Untyped words
  - String items / compound Strings
  - Map/Grant items
- Encoding virtual registers
  - Backed by memory in UTCB
  - In physical registers whenever possible
- Receive Buffers in virtual Buffer Registers
  - Acceptor
  - Receive Strings