μ-Kernel Construction (4)

IPC Functionality & Interface

IPC Primitives

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

- 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
  (send to, receive from any thread)
- Send to & Receive from
  (send to, receive from specified different thread)

Atomic operation to ensure that server's (callee's) reply cannot arrive before client (caller) is ready to receive.

Atomic operation for optimization reasons. Typically used by servers to reply and wait for the next request (from anyone).

- Are other combinations appropriate?

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

IPC – API

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

Problem

- How to we deal with threads that are
  - Uncooperative
  - Malfunctioning
  - Malicious?
- How to prevent an IPC operation from never completing?

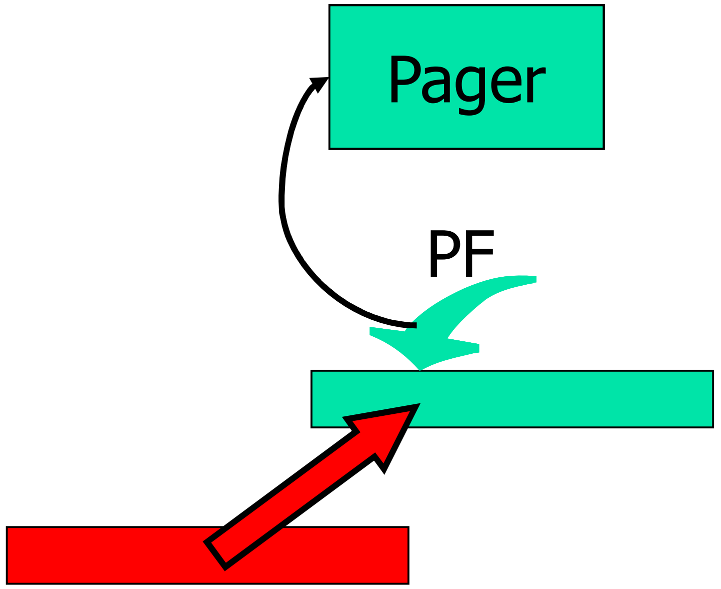
IPC – API

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

IPC – API

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

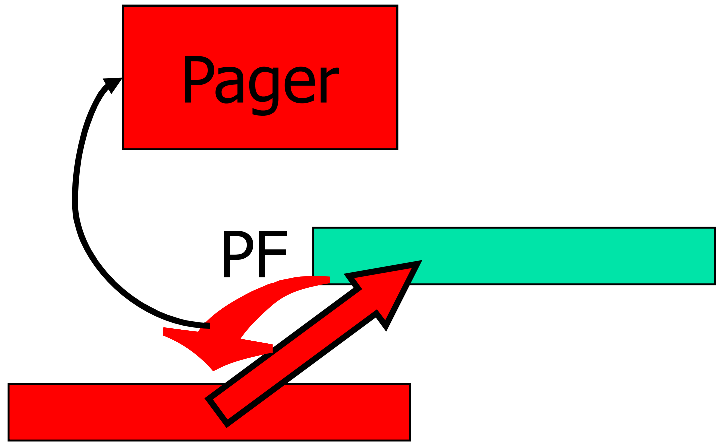
- Attack through receiver's pager



IPC – API

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

- Attack through sender's pager

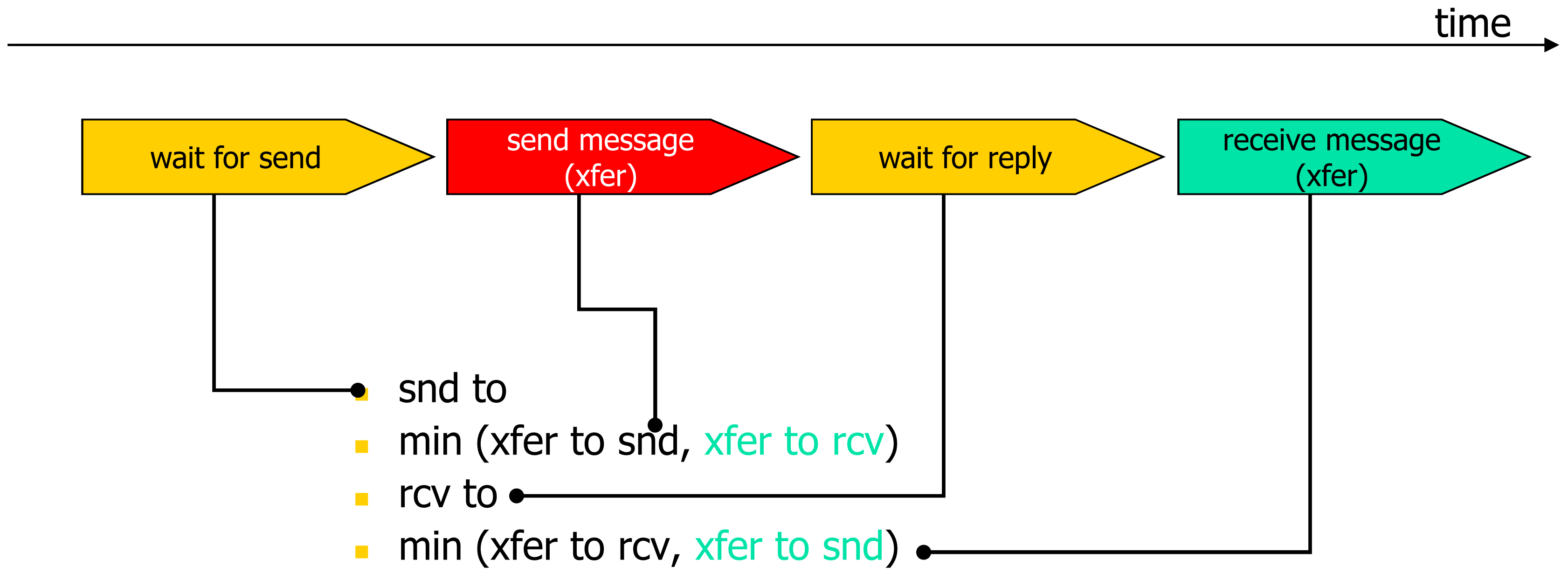


Timeout Problem

- Worst case IPC transfer time is high
  - Potential worst-case is a page fault per memory access
    - IPC time = send timeout + $n \times$ page fault timeout
  - Worst-case for a careless implementation is unbound
    - Pager might respond with null mapping that does not resolve the fault

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)

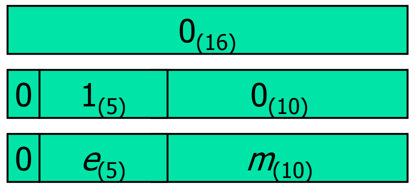
Timeout Issues

- Assume short timeouts need finer granularity than long timeouts
  - Timeouts can always be combined to achieve long fine-grain timeouts

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

IPC – API

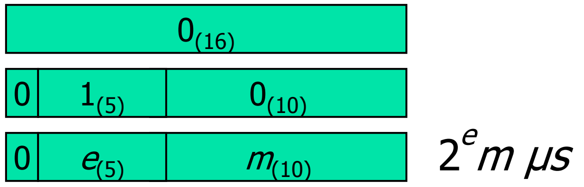
- Timeouts (vX.2, v4)
  - snd timeout, rcv timeout, xfer timeout snd, xfer timeout rcv
    - relative timeout values
      - 0
      - infinite
      - 1 us … 610 h (log)

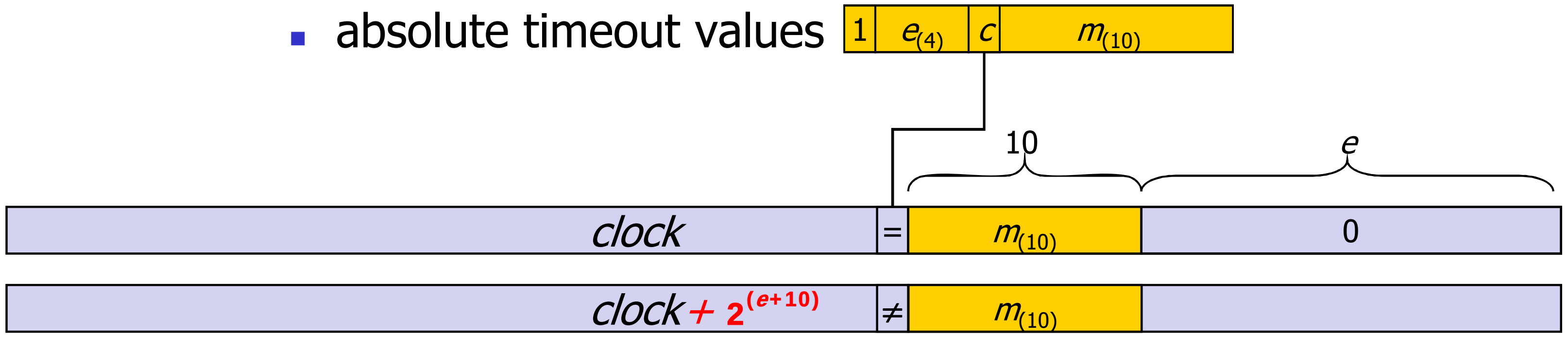


$2^e m\ \mu s$

IPC – API

- Timeouts (vX.2, v4)
  - snd timeout, rcv timeout, xfer timeout snd, xfer timeout rcv
    - relative timeout values
      - 0
      - infinite
      - 1 us … 610 h (log)
    - absolute timeout values





1 | $e_{(4)}$ | c | $m_{(10)}$
10
e
$clock$ | = | $m_{(10)}$ | 0
$clock + 2^{(e+10)}$ | ≠ | $m_{(10)}$

Clarification of the "c" Bit

- User gives absolute timeouts relative to the current epoch (:= all but the least significant 10+e bits of clock).
- Kernel computes absolute timeout via "(clock' & (~0ull << (10+e))) | (m << e)", i.e., "epoch' | (m << e)".
  - The clock readings of the client and the kernel are different!

(a) Timeout 09:50, clock 09:45 => epoch 09:00 => delta := m << e = 50'
  - Kernel reached at clock' 09:48 => epoch' 09:00 => timeout 09:50 (ok)

(b) Timeout 10:12, clock 09:55 => epoch 09:00 => delta 1:12 (must be able to specify "next epoch")
  (1) Kernel reached at clock' 09:59 => epoch 09:00 => timeout 10:12 (ok)
  (2) Kernel reached at clock' 10:01 => epoch 10:00 => timeout 11:12 (wrong)

Instead of specifying "this vs. next epoch" specify least significant bit (LSB) of target epoch:

(a) Timeout 09:50, clock 09:45 => epoch 09:00 => c = LSB(09) == 1, delta 50'
  - Kernel reached at clock' 09:48 => epoch' 09:00 => LSB(09) == 1 == c => epoch" 09:00 => timeout 09:50 (ok)

(b) Timeout 10:12, clock 09:55 => epoch 09:00 => c = LSB(10) == 0, delta 12'
  (1) Kernel reached at clock' 09:59 => epoch' 09:00 => LSB(09) == 1 != c => epoch" 10:00 => timeout 10:12 (ok)
  (2) Kernel reached at clock' 10:01 => epoch' 10:00 => LSB(10) == 0 == c => epoch" 10:00 => timeout 10:12 (ok)

Errors occur only if the epoch changes more than once between the client and the kernel reading the clock, i.e., if more than one complete epoch ((1<<(10+e)) μs ≈ (1<<e) ms) passed in between.

As can be seen in (b1), using c is different from using more bits for the delta (effectively specifying the LSB of the target epoch): epoch' is 09, having delta include the LSB would decrease this to 08 (LSB forced to 0); considering c != LSB(09) increases epoch' to 10.

Do not waste your time understanding this – informational only!

Timeout Range of Values (seconds) [v4, vX.2]

e	*m* =1	*m* =1023
0	0,000001	0,001023
1	0,000002	0,002046
3	0,000008	0,008184
5	0,000032	0,032736
7	0,000128	0,130944
9	0,000512	0,523776
11	0,002048	2,095104
13	0,008192	8,380416
15	0,032768	33,521664
17	0,131072	134,086656
19	0,524288	536,346624
21	2,097152	2145,386496
23	8,388608	8581,545984
25	33,554432	34326,18394
27	134,217728	137304,7357
29	536,870912	549218,943
31	2147,483648	2196875,772

1µs – 1023µs with 1µs granularity

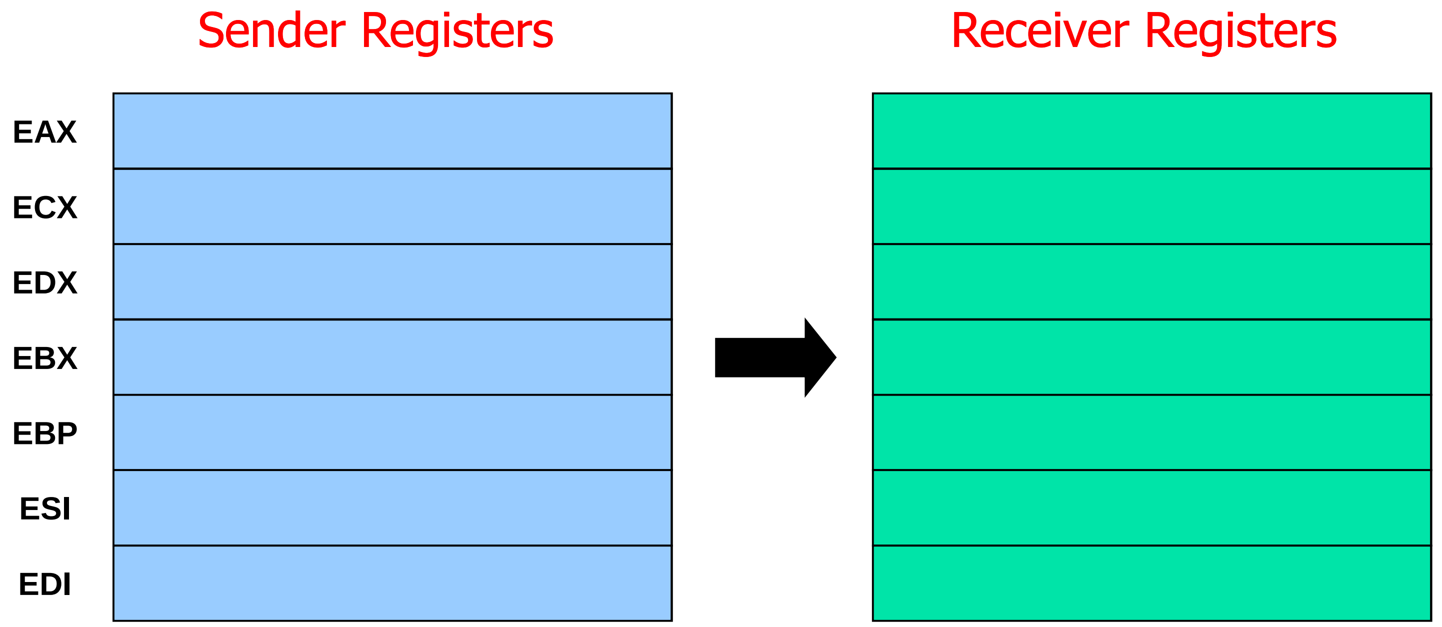
Up to ~610h with ~35min granularity

IPC Parameters

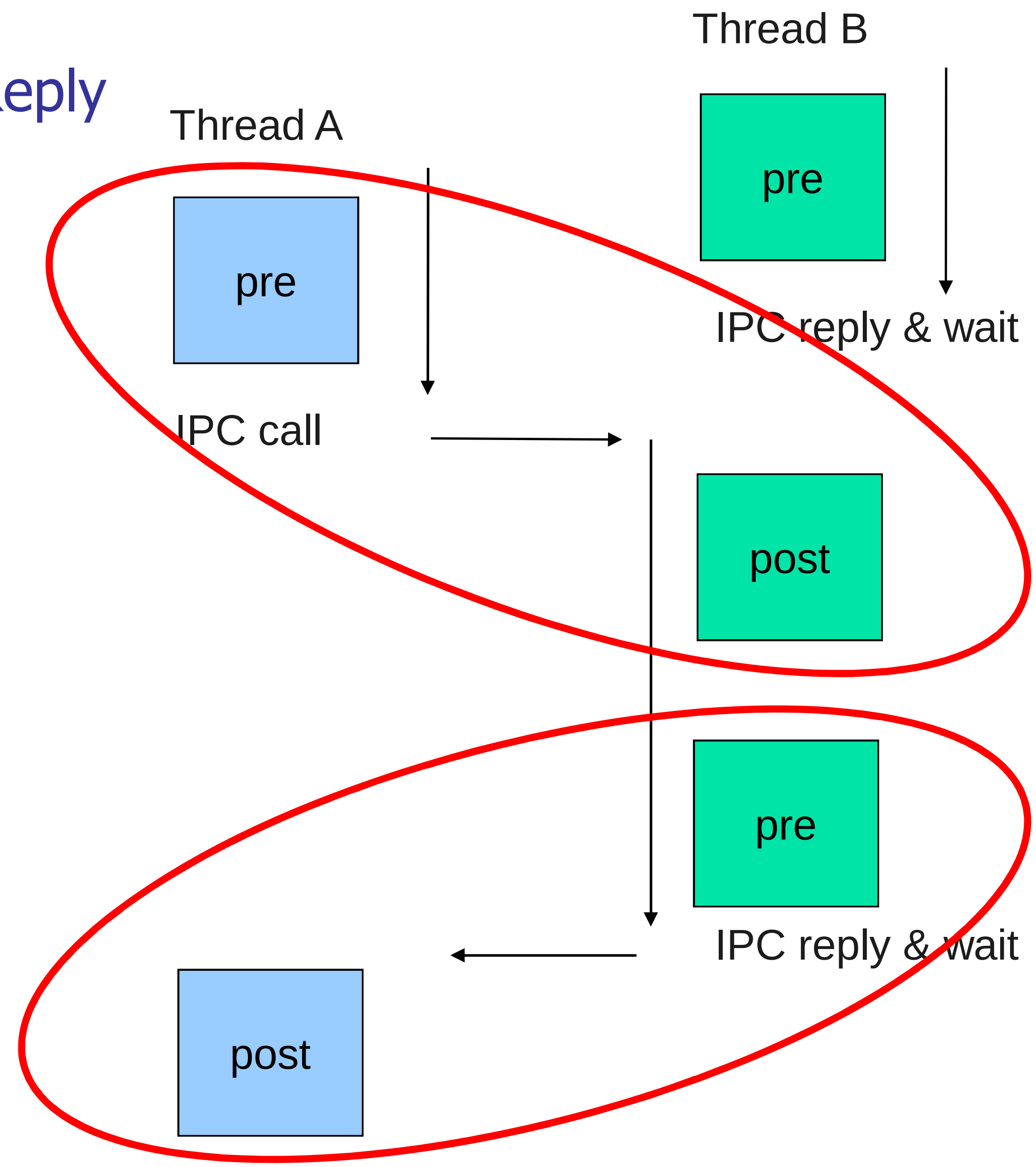
- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for 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
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC

Ideally Encoded in Registers

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

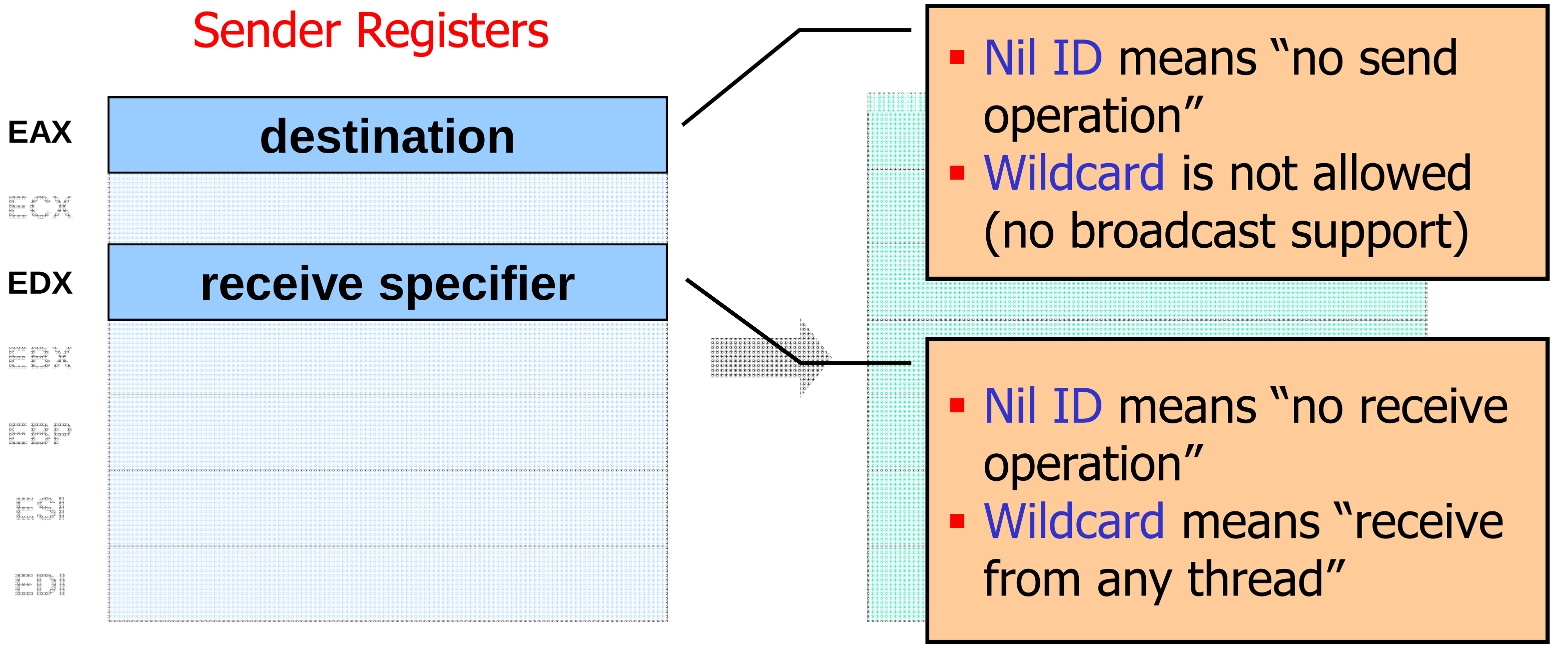


Example: Call-Reply



Send and Receive Encoding

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



- Nil ID means "no send operation"
- Wildcard is not allowed (no broadcast support)

- Nil ID means "no receive operation"
- Wildcard means "receive from any thread"

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 Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from

IPC syscall

- 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
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC

Message Transfer

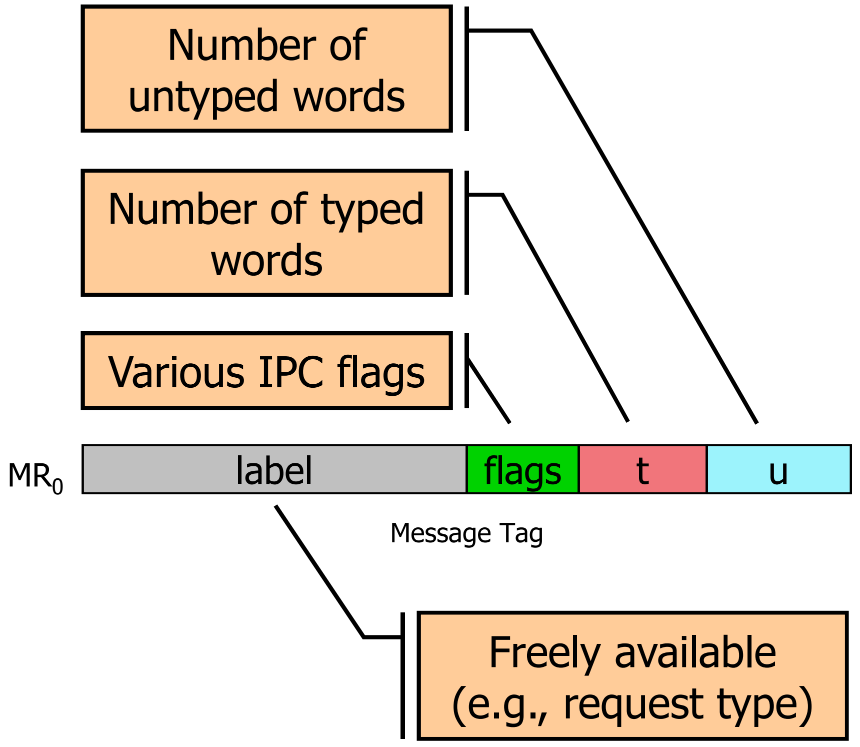
- Assume that 64 extra registers are available
  - Name them MR_0 … MR_{63} (message register 0 … 63)
  - All message registers are transferred during IPC

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
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC

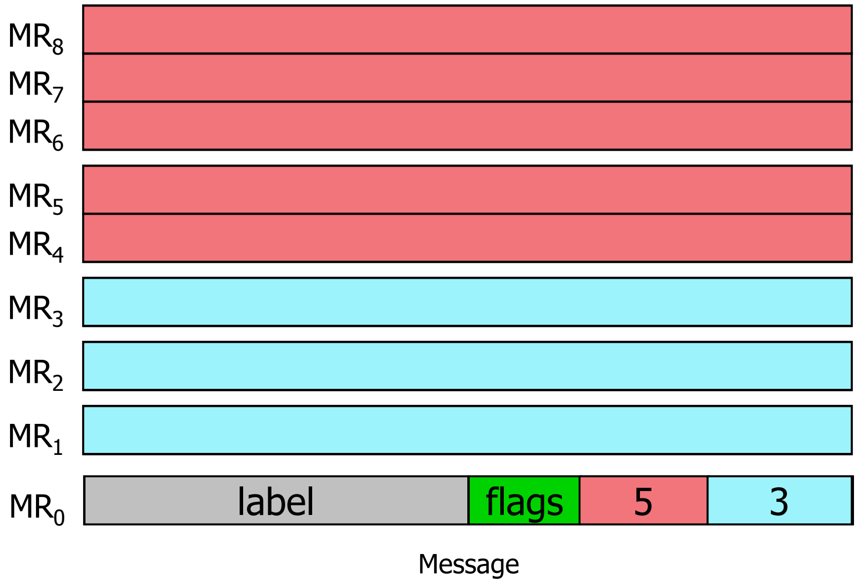
Message Construction

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



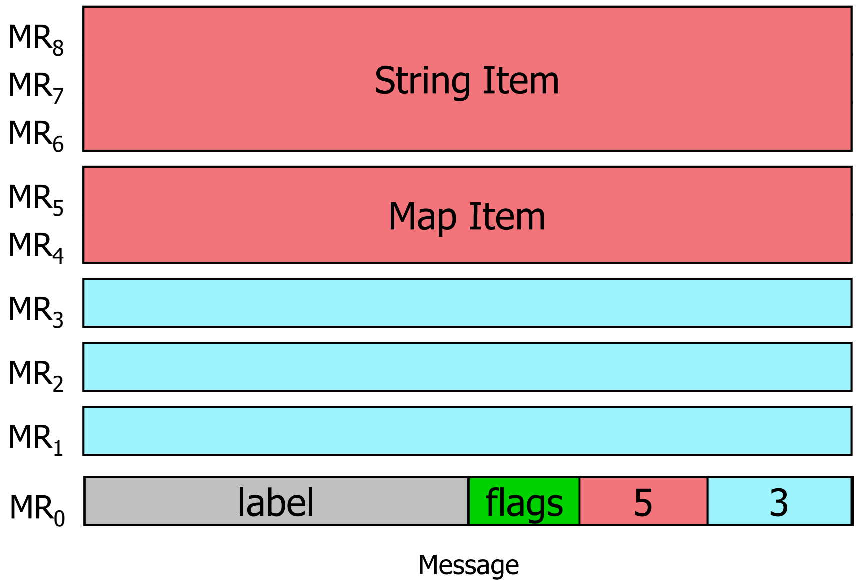
Message Construction

- 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)



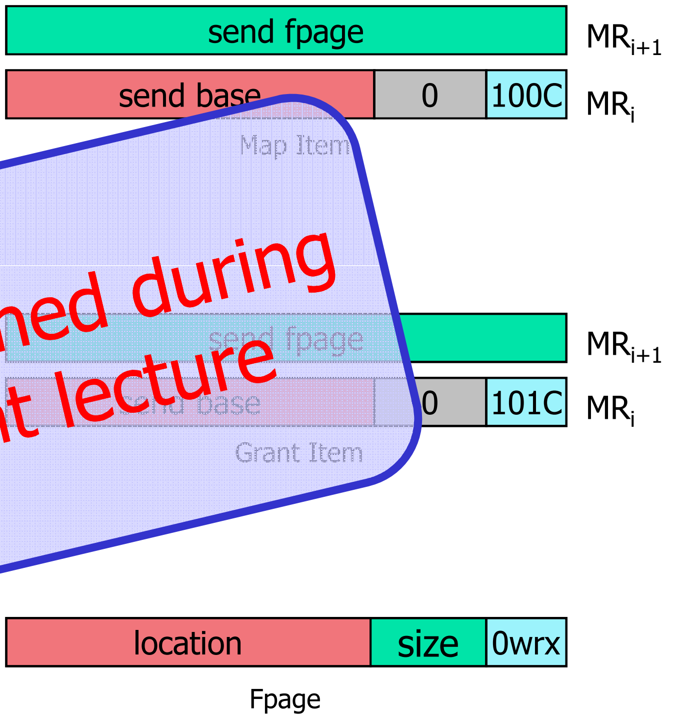
Message Construction

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



Map and Grant Items

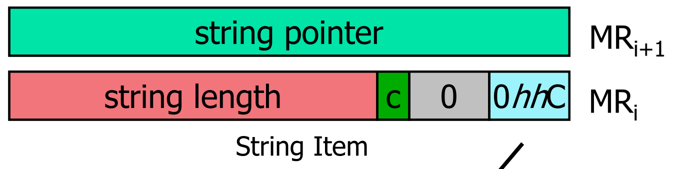
- Two words
  - Send base
  - Fpage
- Lower bits of send base indicates map or grant item



Semantics will be explained during memory management lecture

String Items

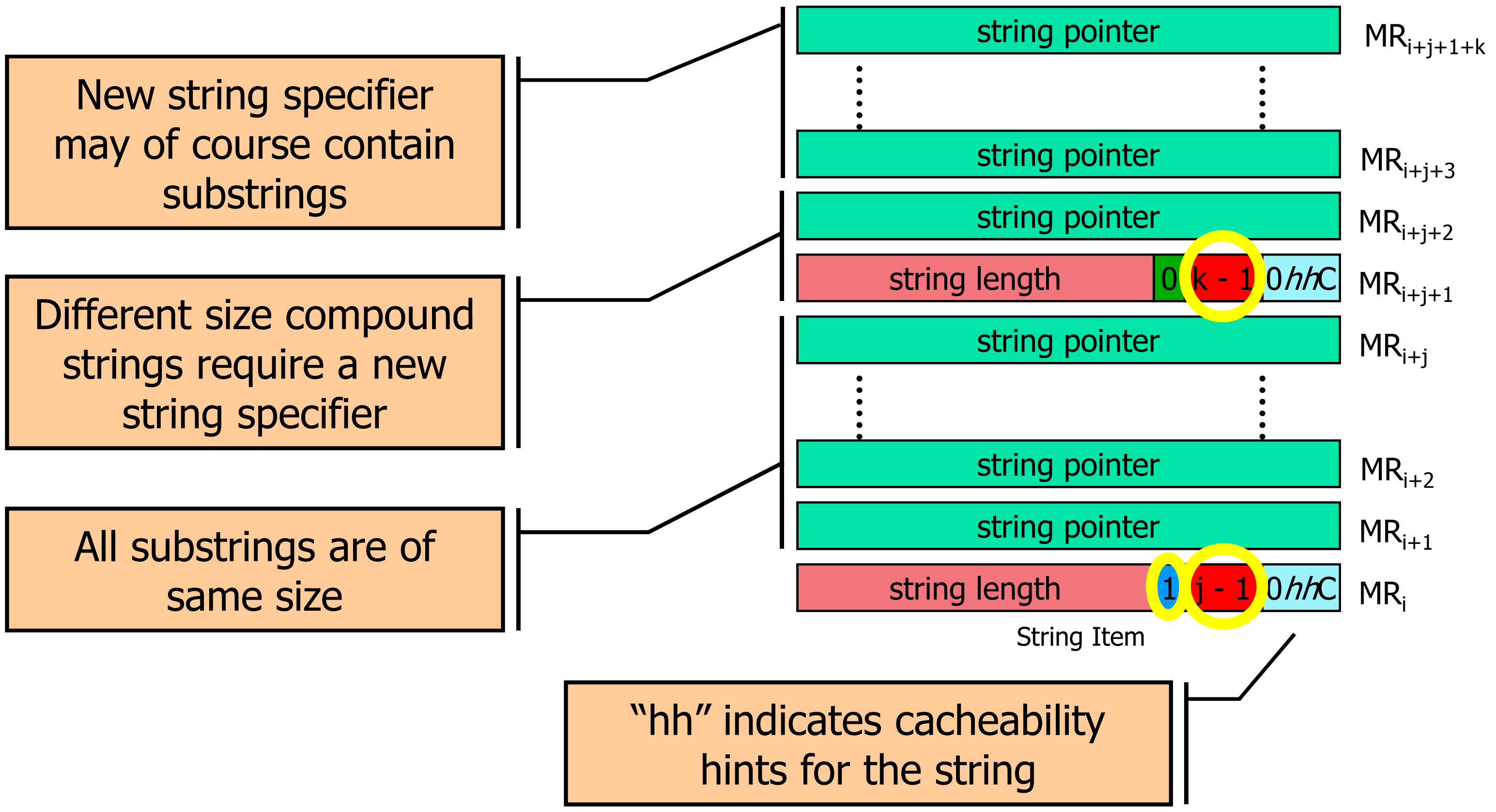
- Up to 4 MB (per string)
- Compound strings supported
  - Allows scatter-gather
- Incorporates cacheability hints
  - Reduce cache pollution for long copy operations



"hh" indicates cacheability hints for the string

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

String Items



IPC Parameters

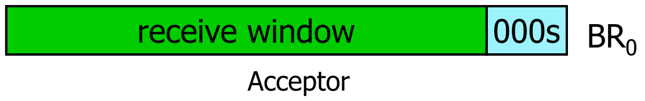
- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for 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
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC

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)



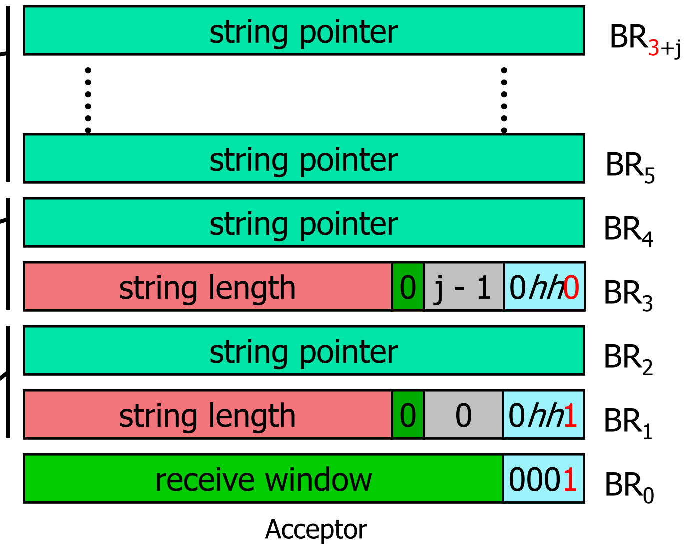
Receiving Messages

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



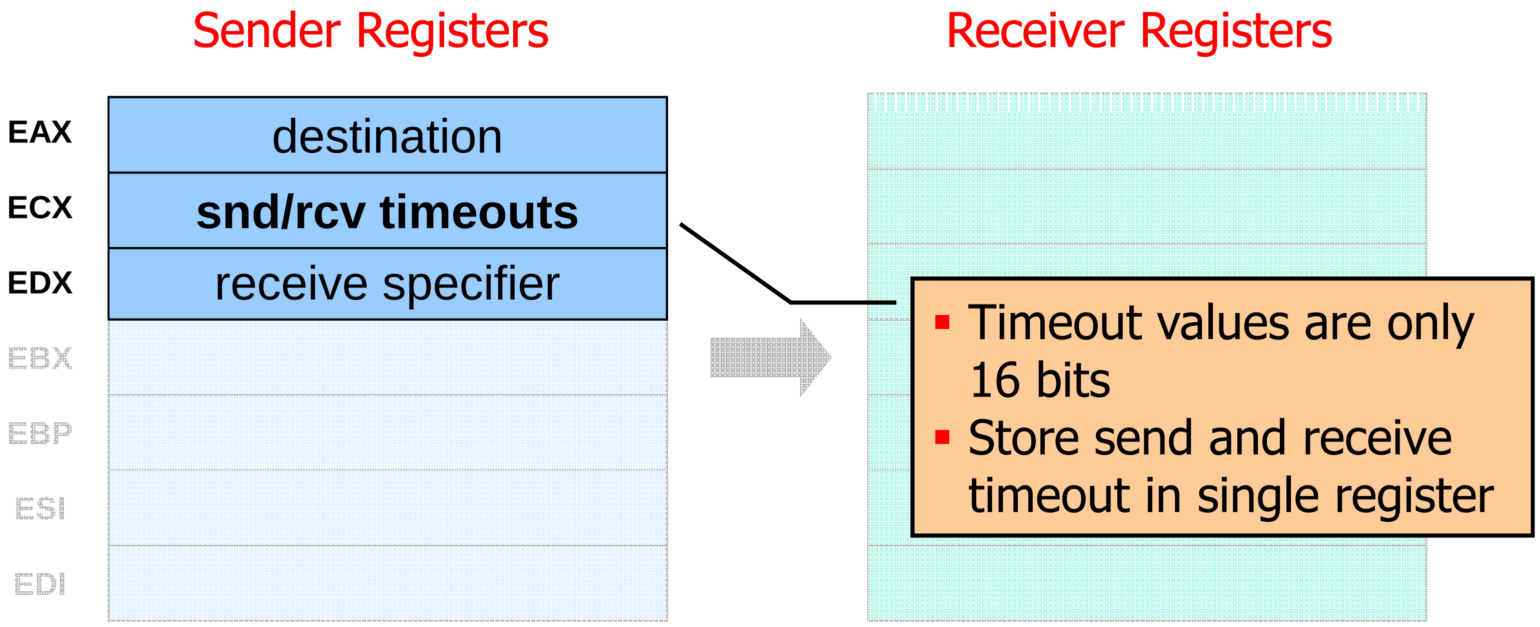
IPC Parameters

- 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
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC

Timeouts

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

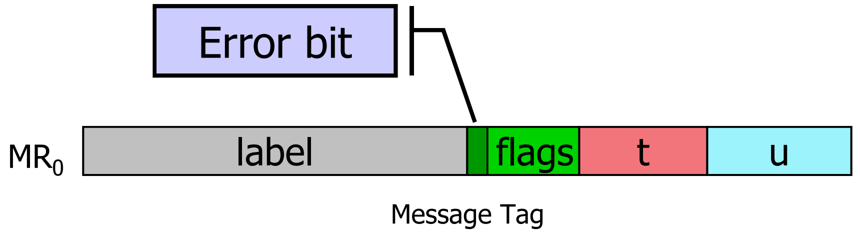


- Timeout values are only 16 bits
- Store send and receive timeout in single register

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
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC

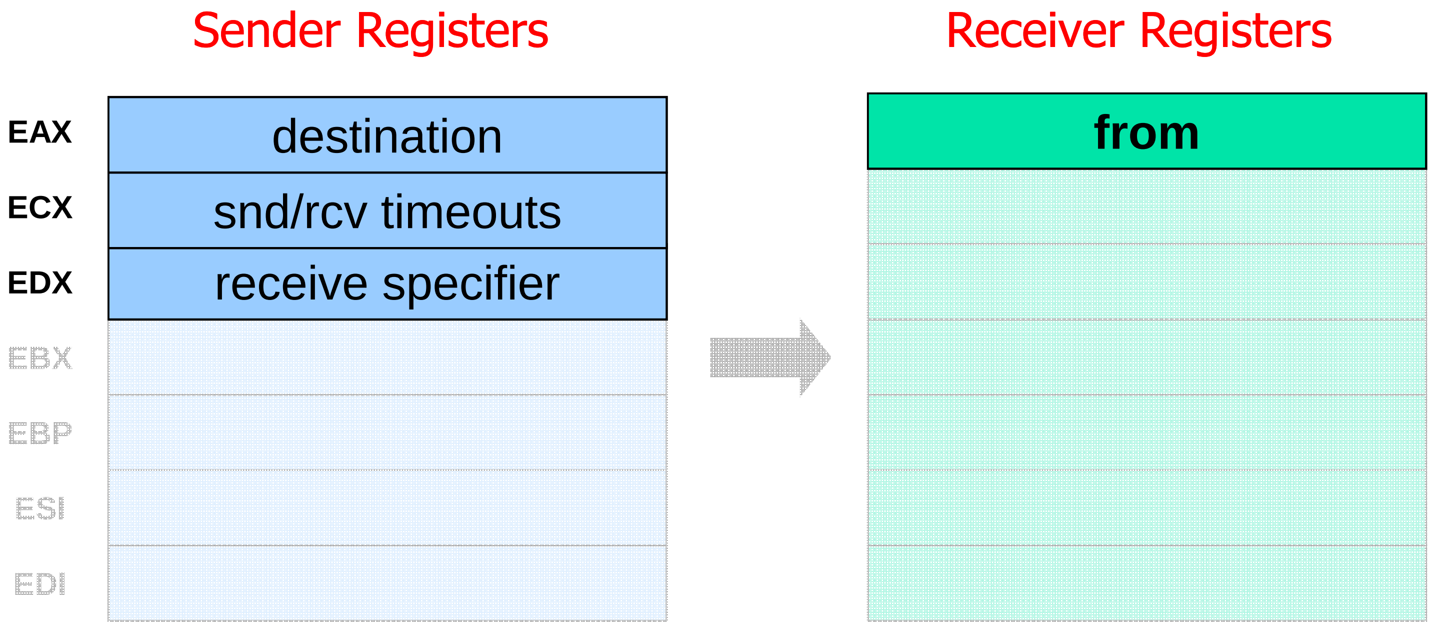
IPC Result



- 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

IPC Result

- IPC errors flagged in MR_0
- Sender’s thread ID stored in register

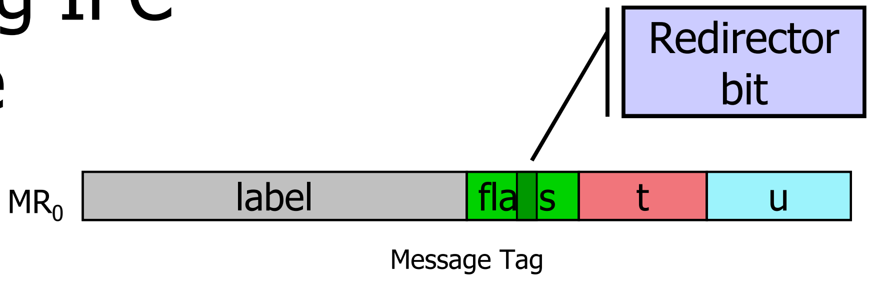


IPC Parameters

- 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
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC

IPC Redirection

- Redirection/deceiting IPC flagged by bit in the message tag
  - Fast check
- When redirection bit set
  - Thread ID to deceit as and intended receiver ID stored in predefined memory locations



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
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC

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+9 spare registers

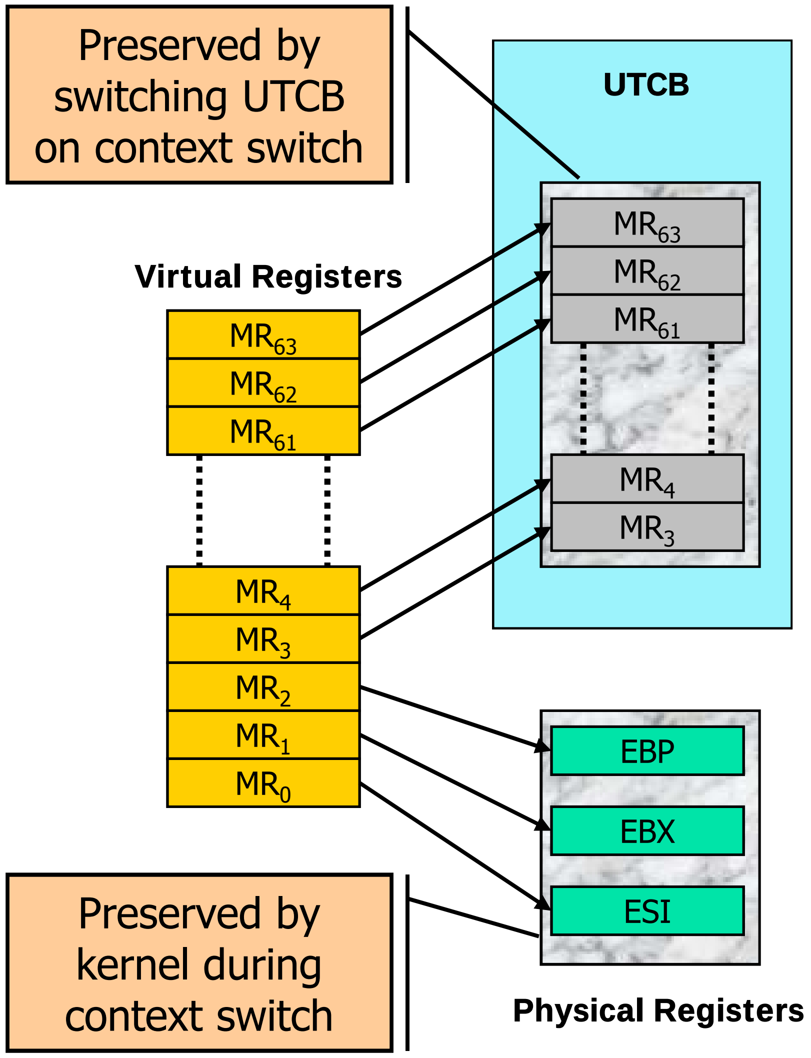
Define as Virtual Registers

- What about predefined memory locations?
  - Must be thread local

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 thread local
  - UTCBs cannot be paged
    - No page faults
    - Registers always accessible



Switching UTCBs (IA-32)

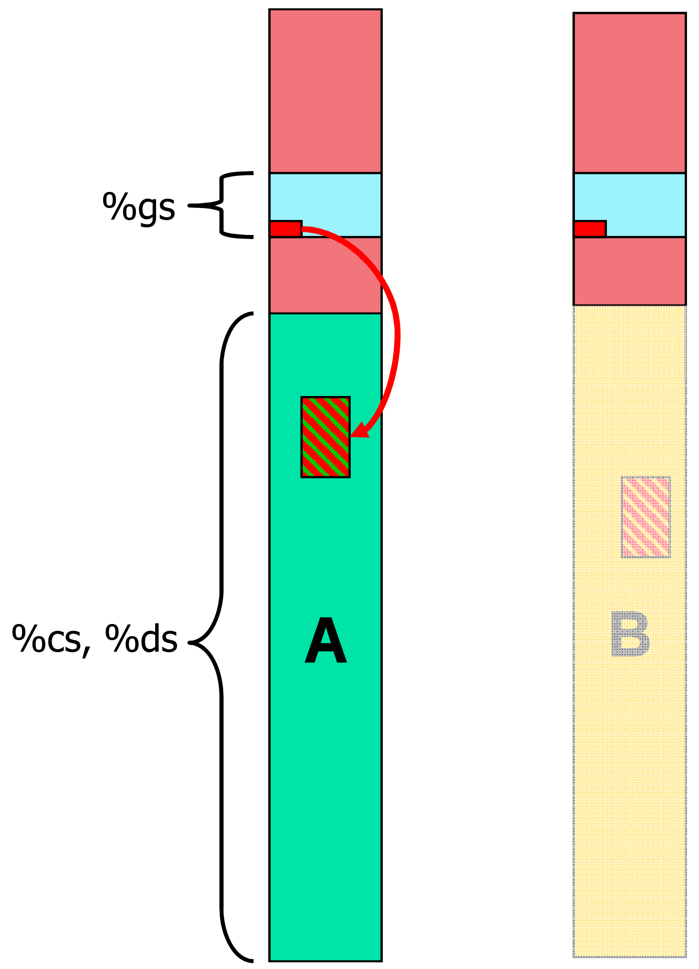
- Locating UTCB must be fast

  (avoid using system call)

- Use separate segment for UTCB pointer

  movl %gs:0, %edi

- Switch pointer on context switches



Switching UTCBs (IA-32)

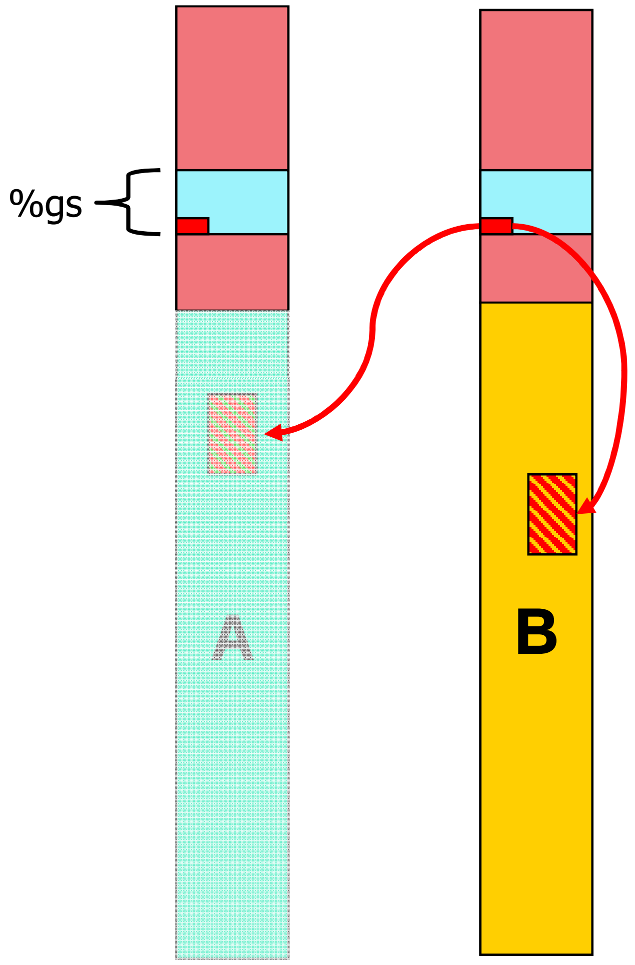
- Locating UTCB must be fast

  (avoid using system call)

- Use separate segment for UTCB pointer

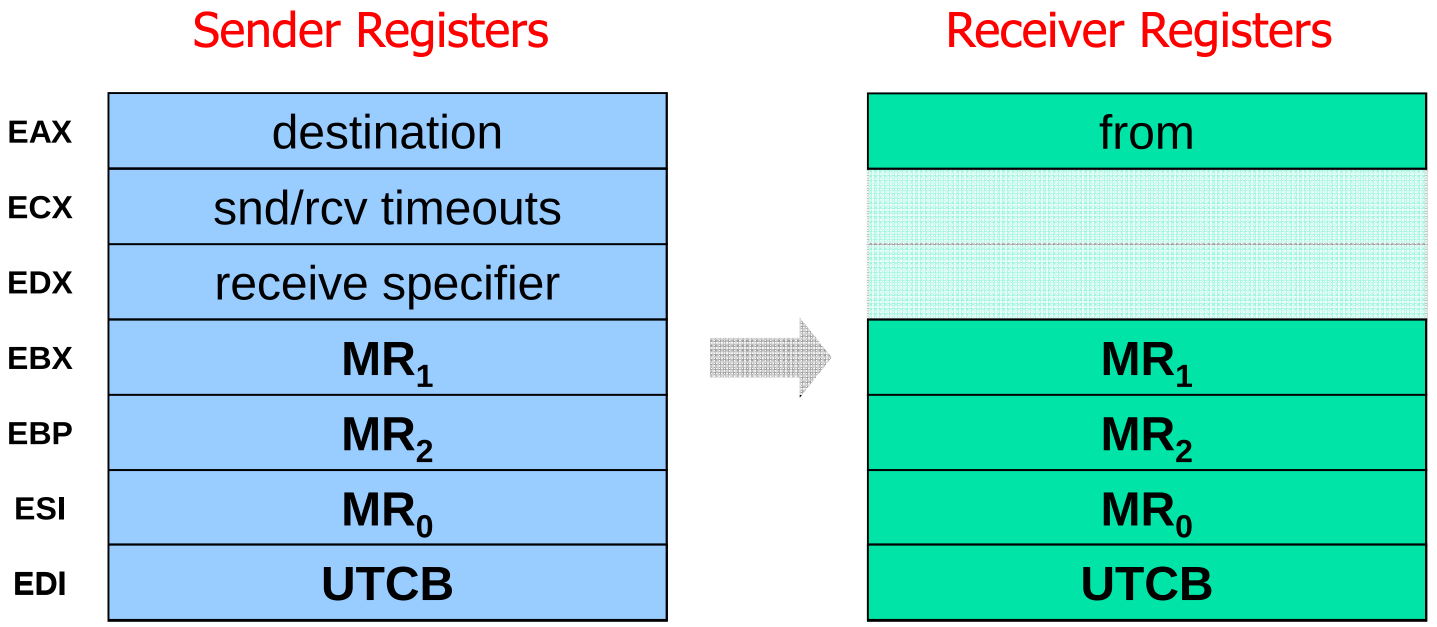
  movl %gs:0, %edi

- Switch pointer on context switches



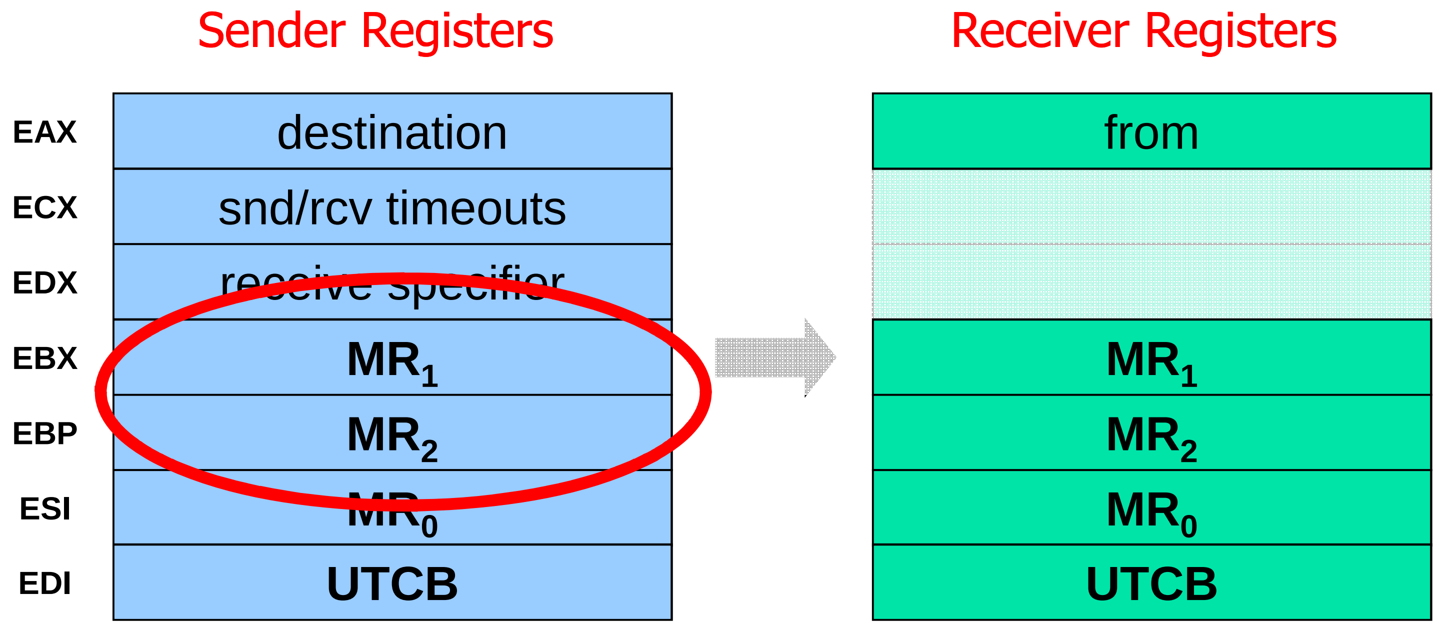
Message Registers and UTCB

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



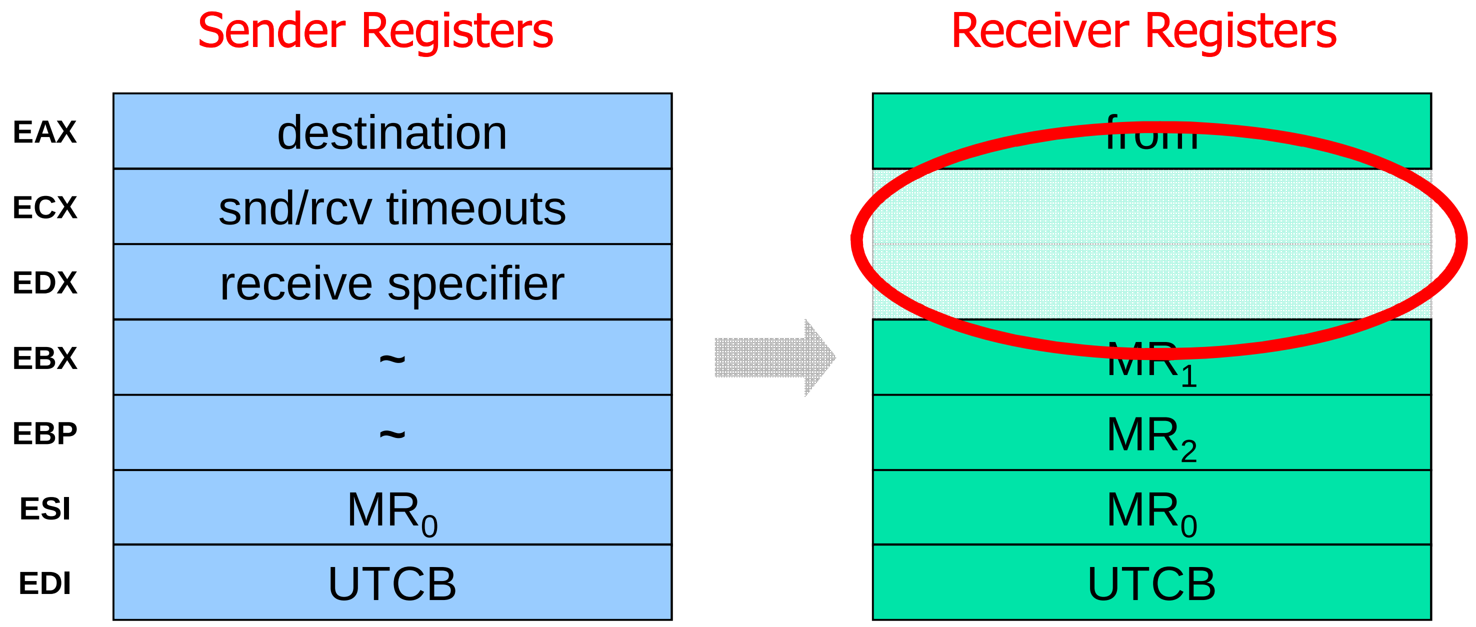
Free Up Registers for Temporary Values

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



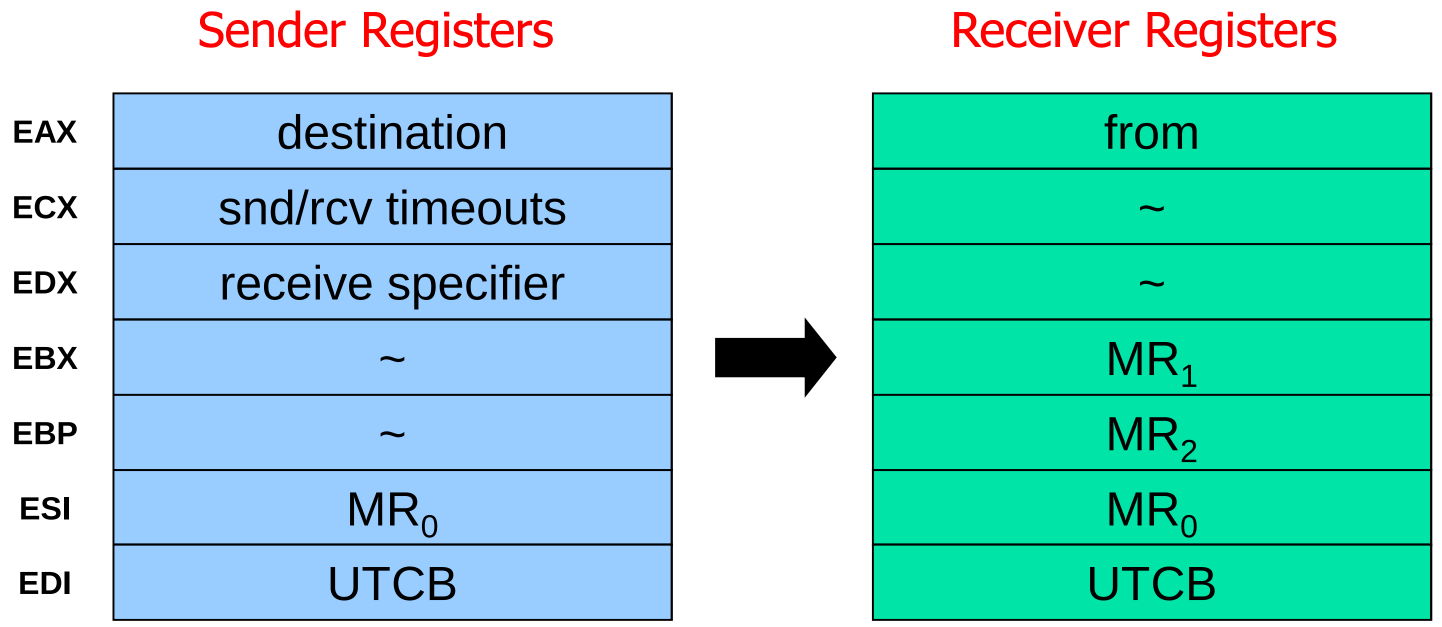
Free Up Registers for Temporary Values

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



IPC Register Encoding

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

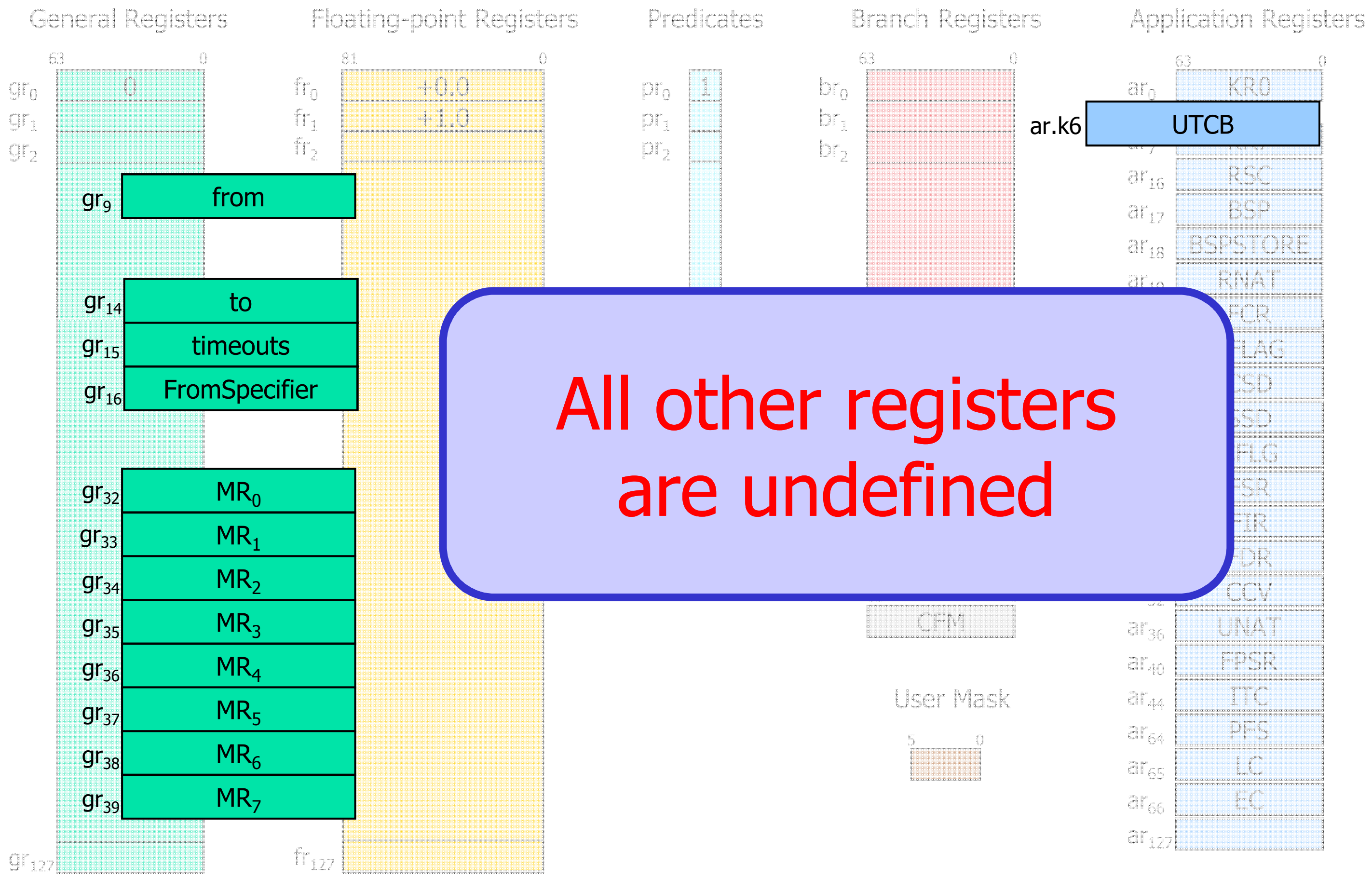


	Sender Registers	Receiver Registers
EAX	destination	from
ECX	snd/rcv timeouts	~
EDX	receive specifier	~
EBX	~	MR_1
EBP	~	MR_2
ESI	MR_0	MR_0
EDI	UTCB	UTCB

Case study: IA-64

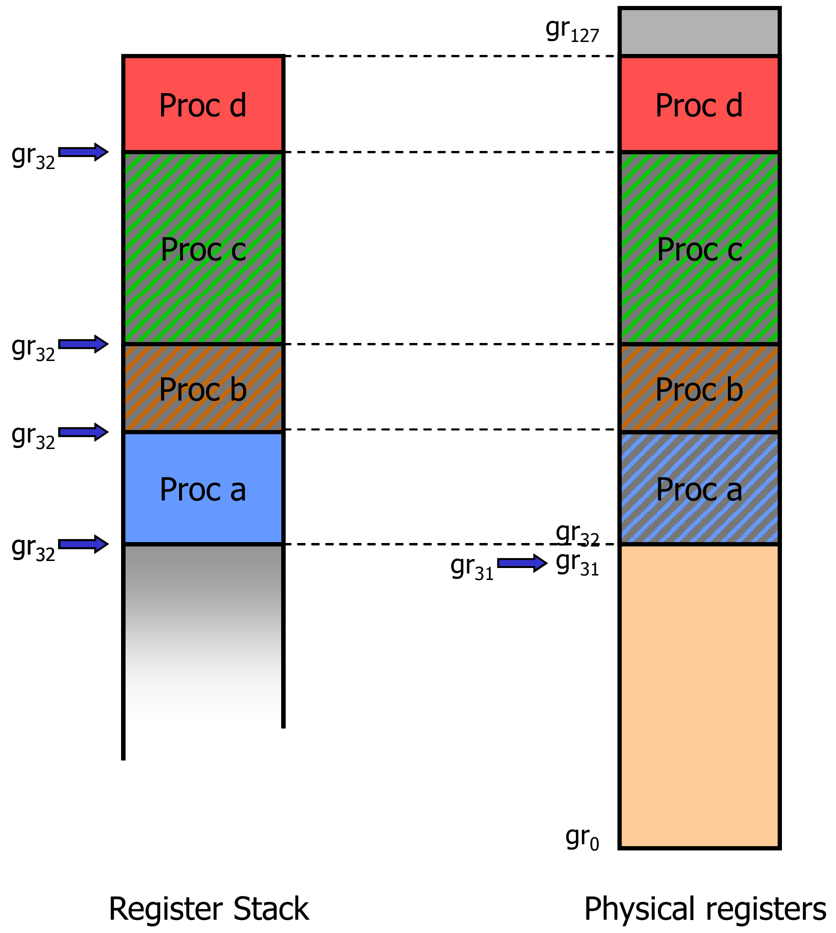
IPC Register Usage

Register Encoding on IA-64

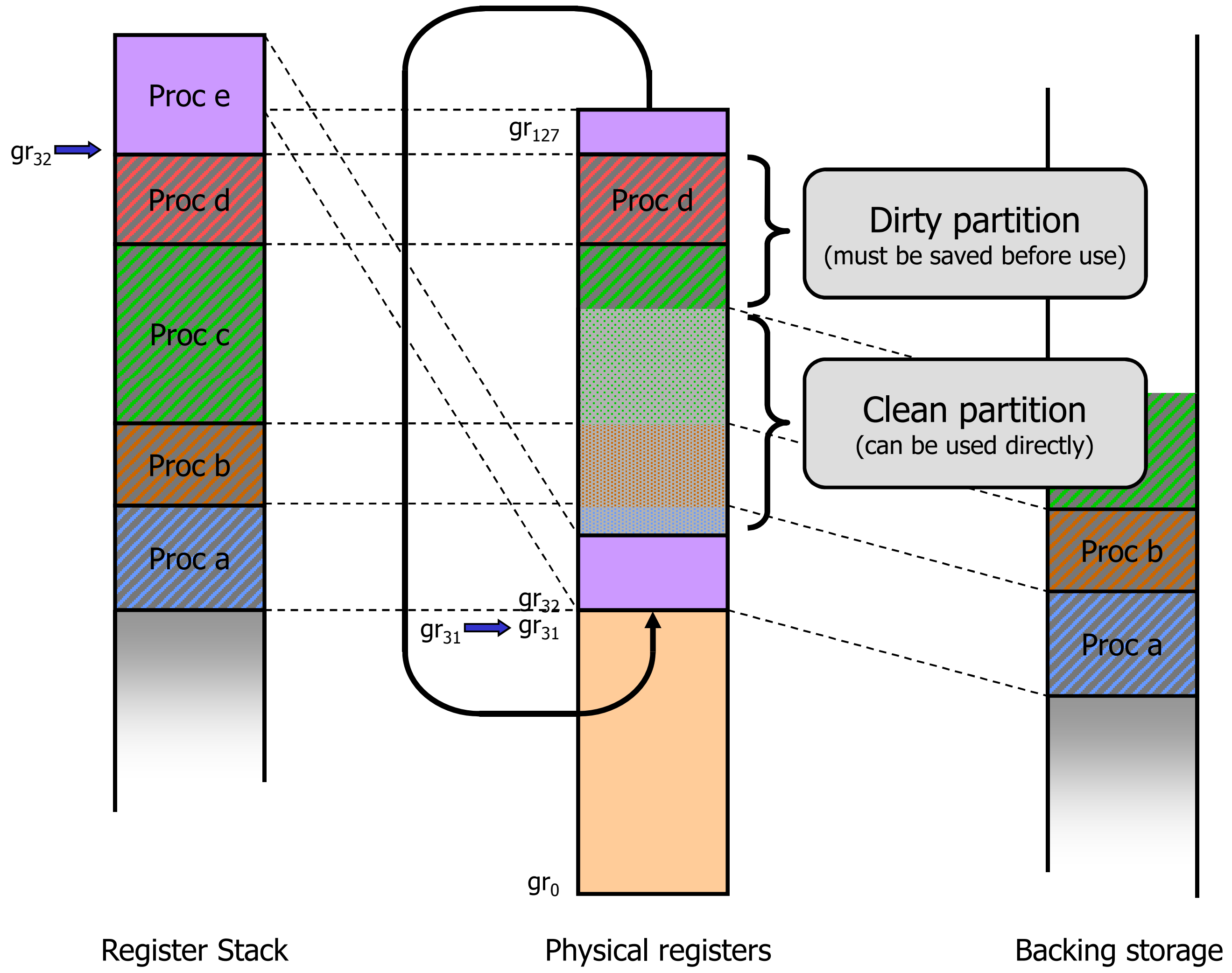


All other registers are undefined

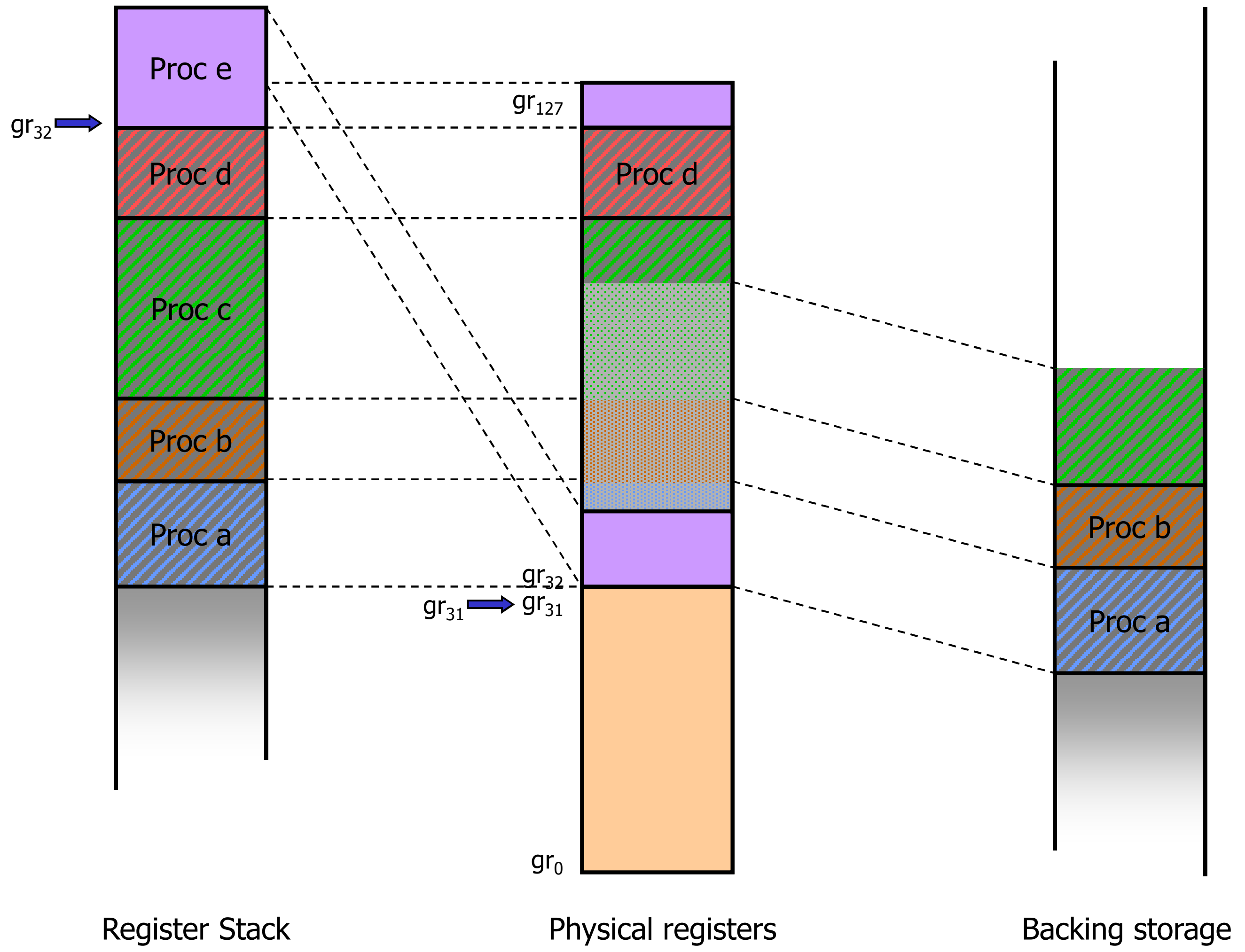
Register Stack Engine



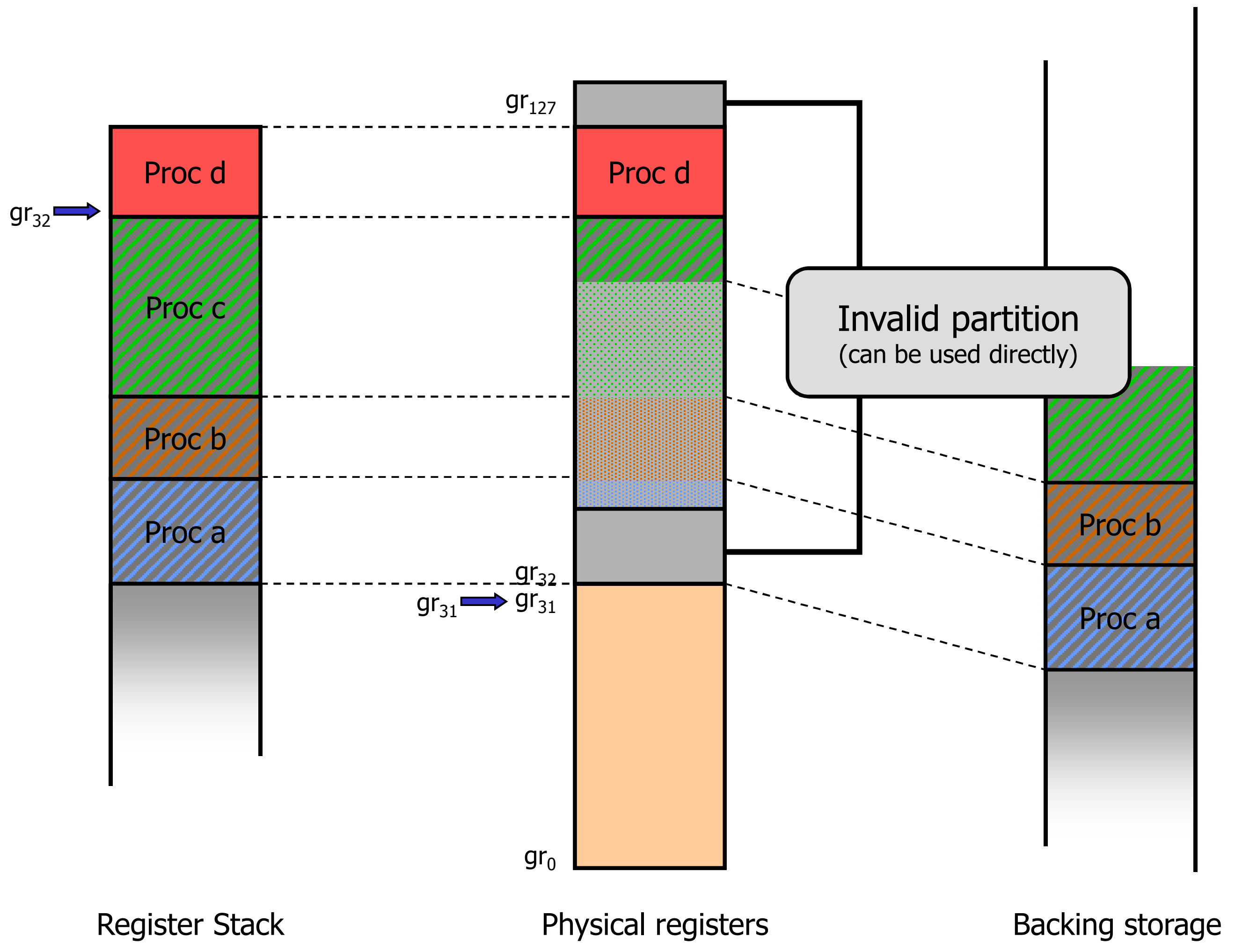
Register Stack Engine



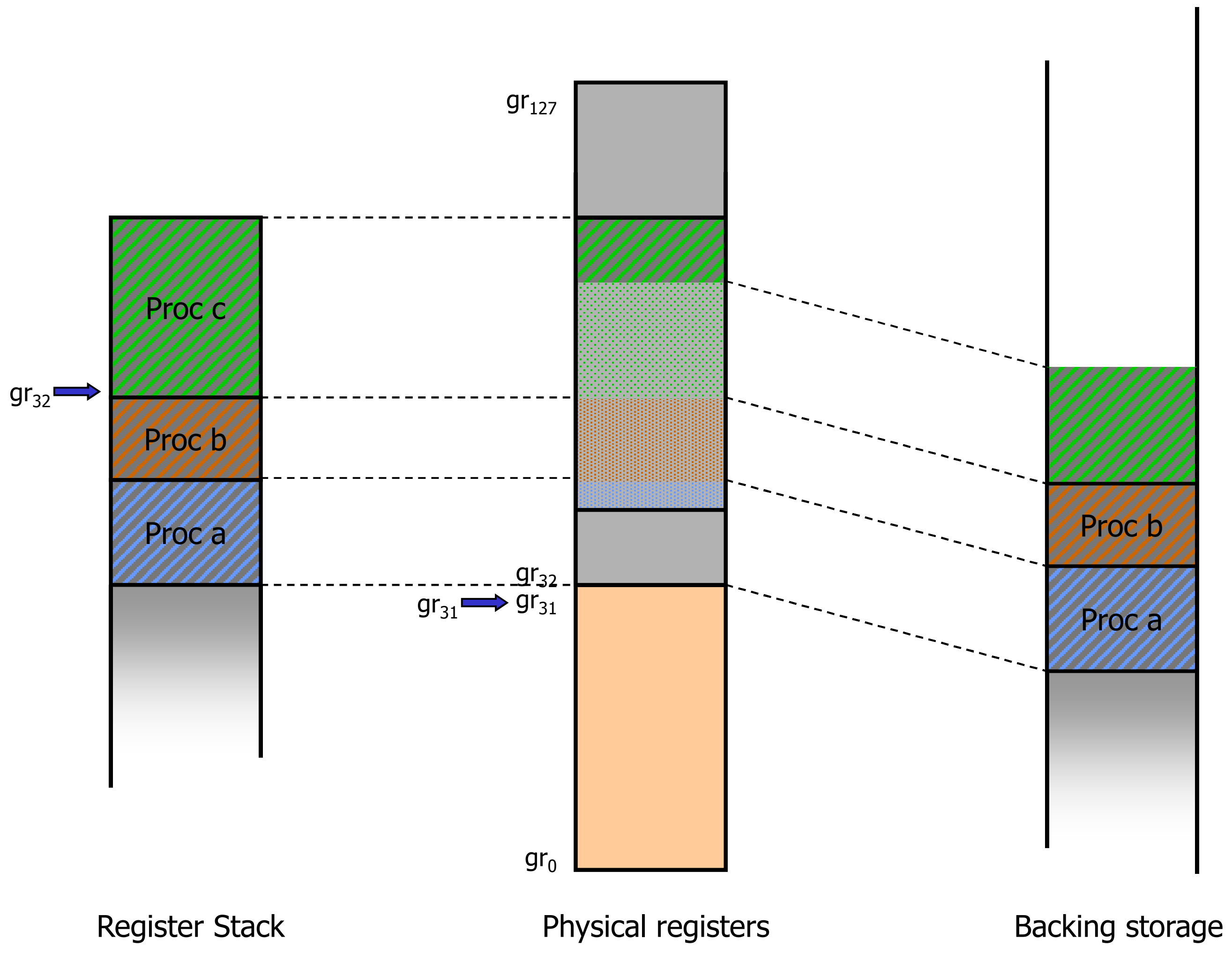
Register Stack Engine



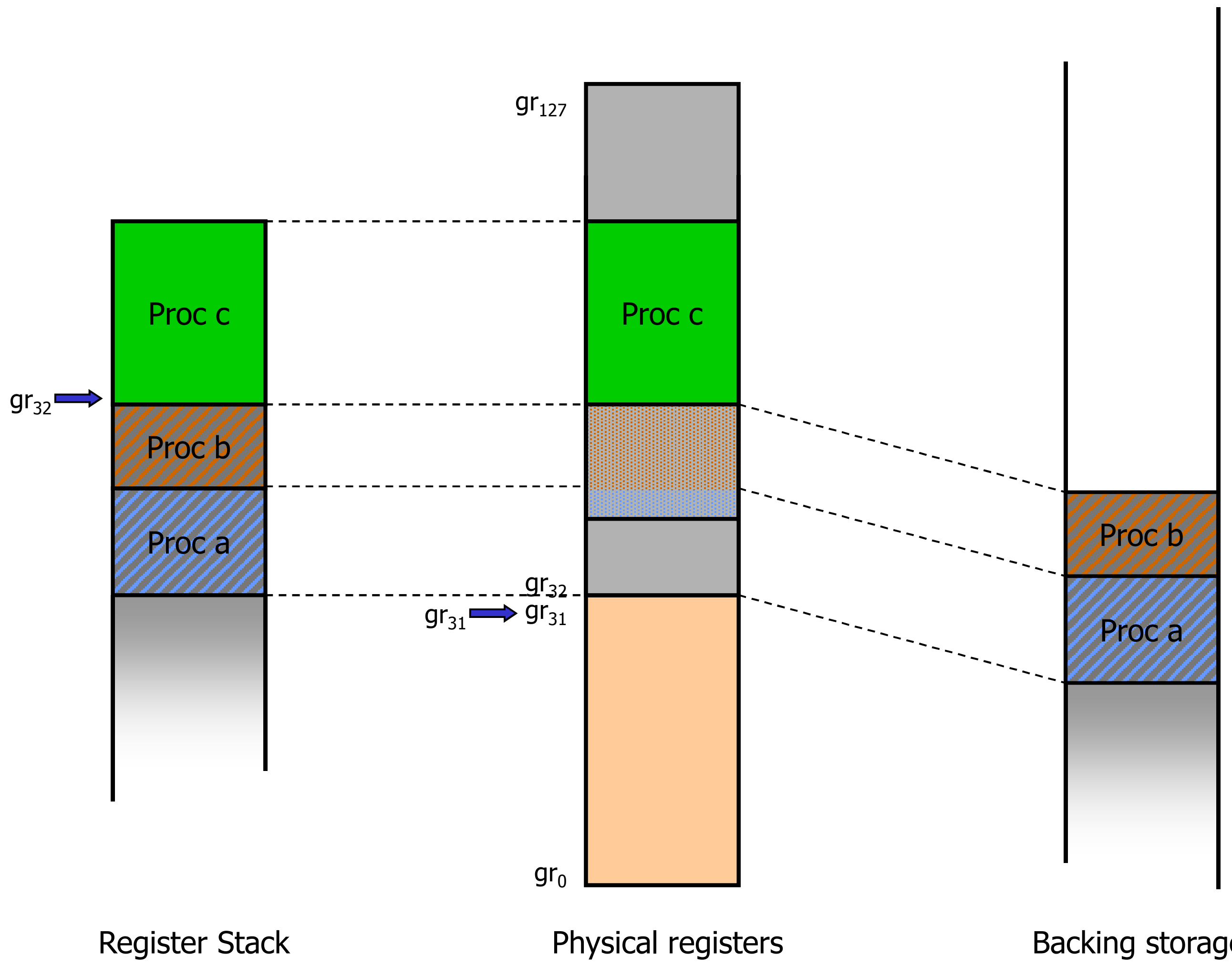
Register Stack Engine



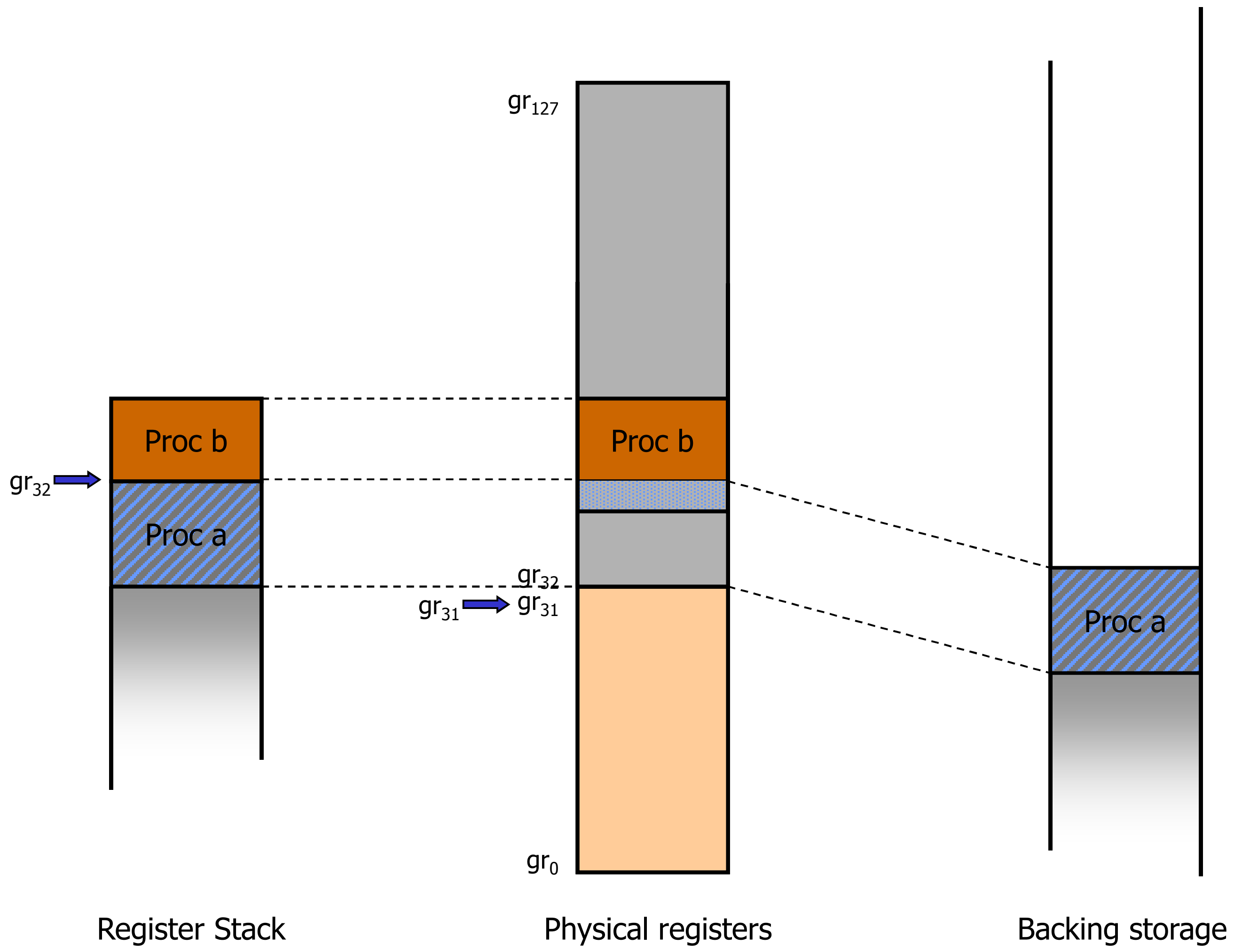
Register Stack Engine



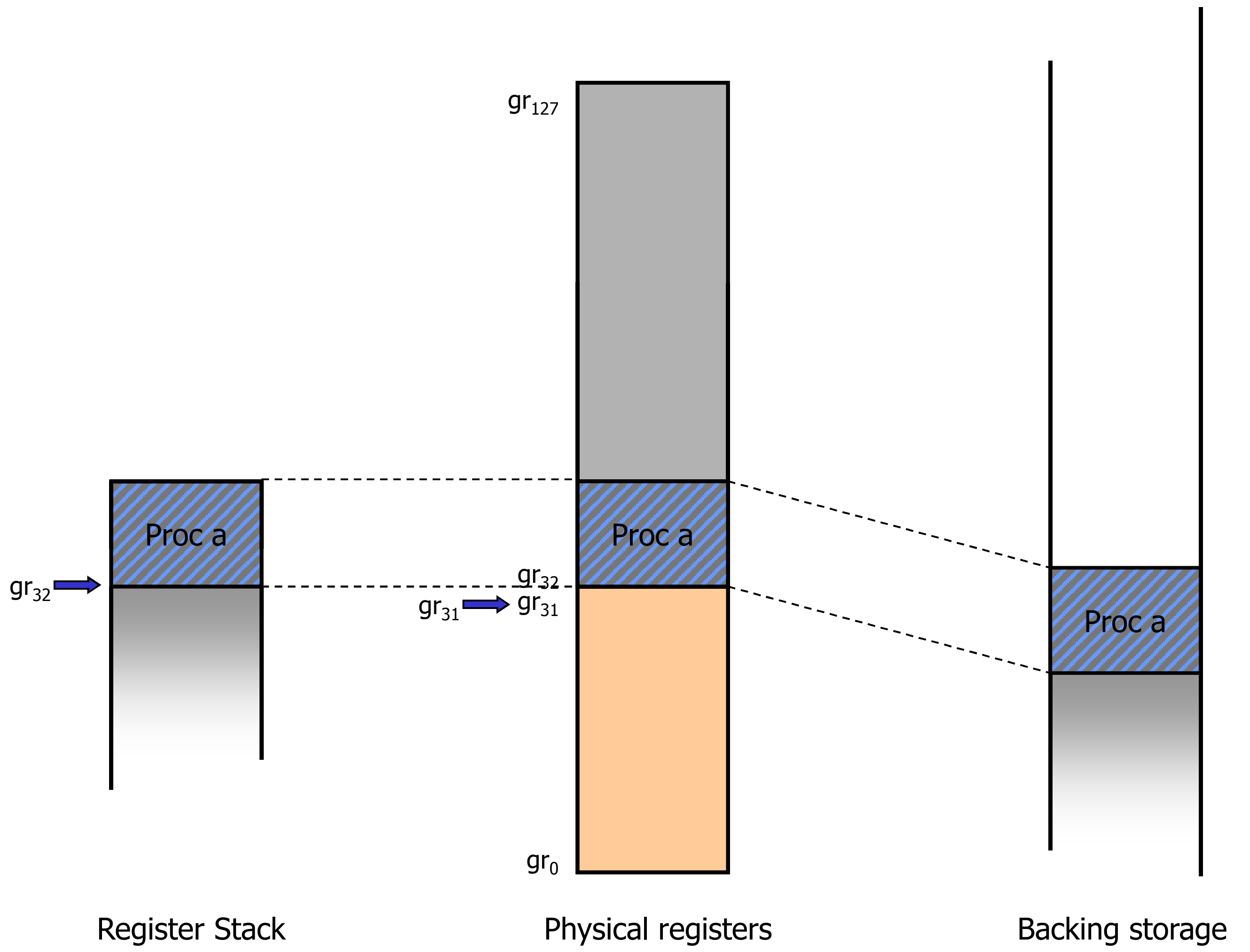
Register Stack Engine



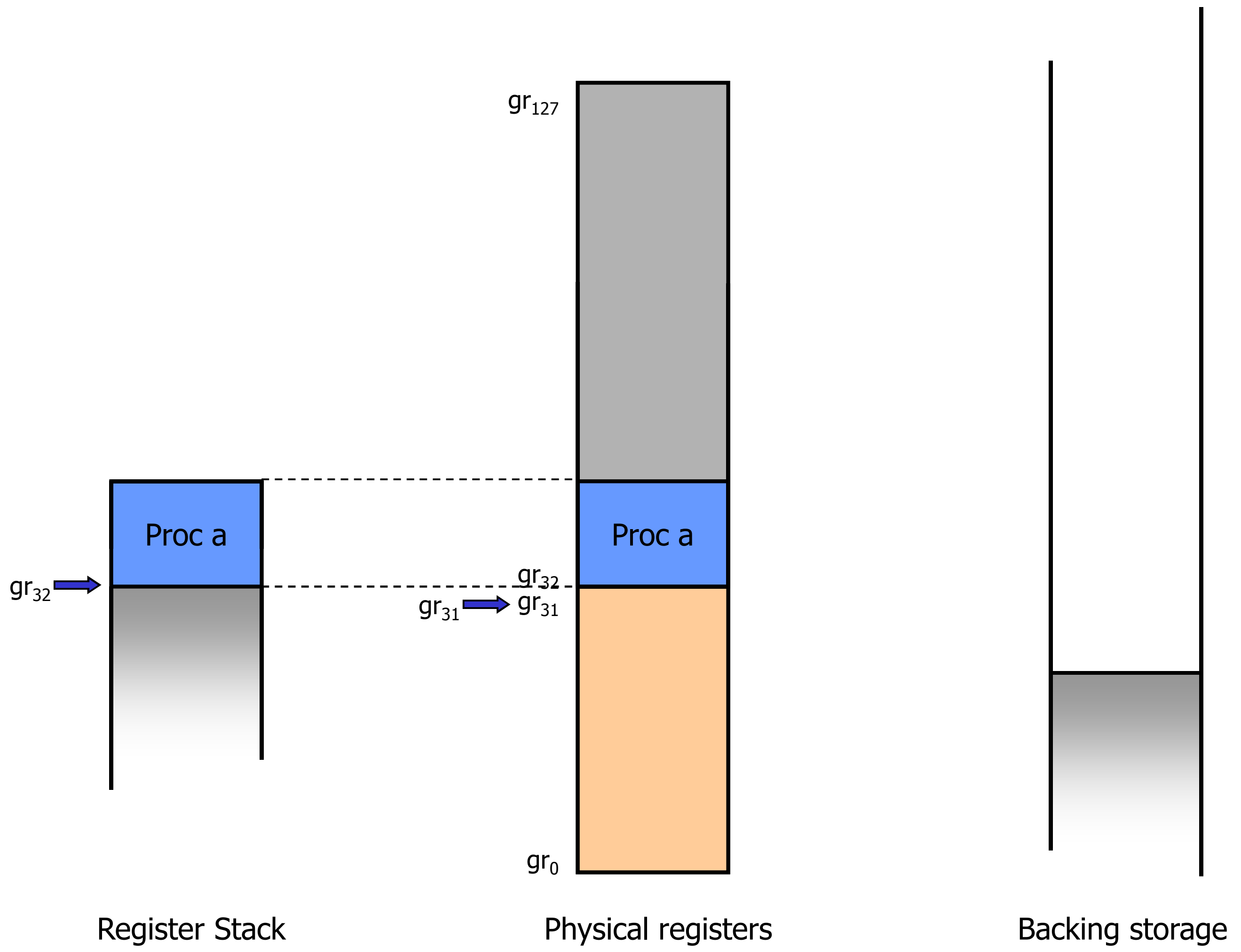
Register Stack Engine



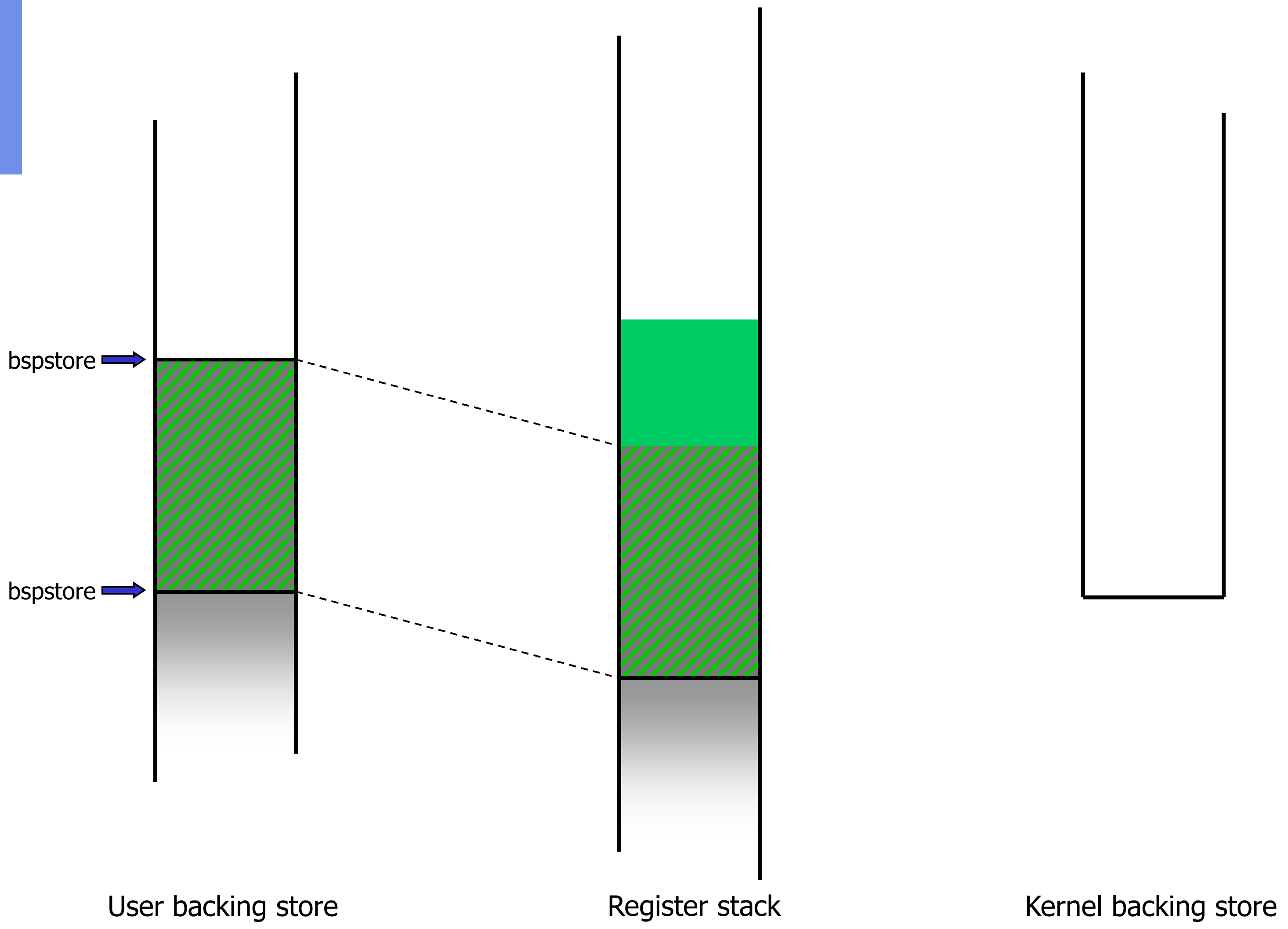
Register Stack Engine



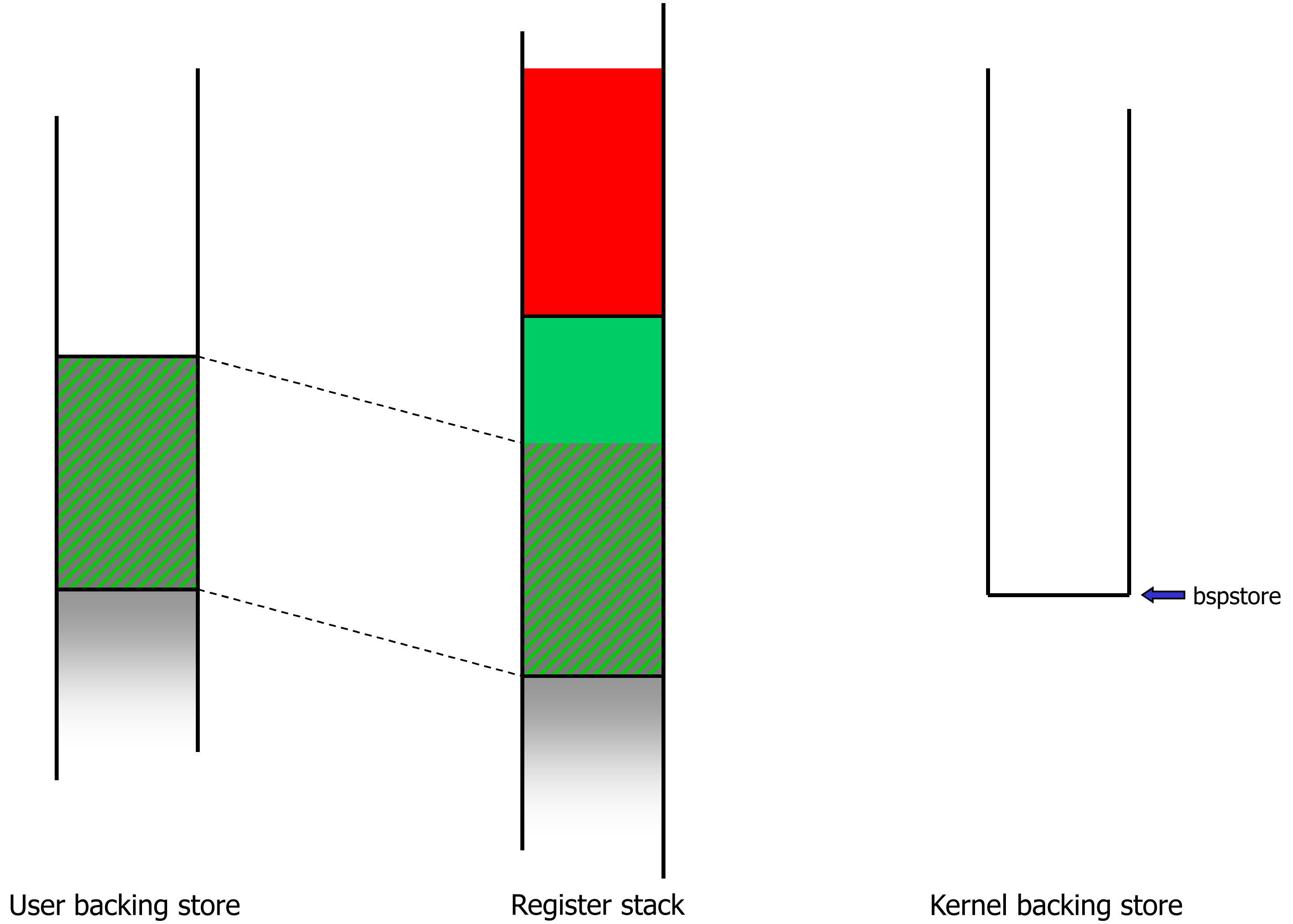
Register Stack Engine



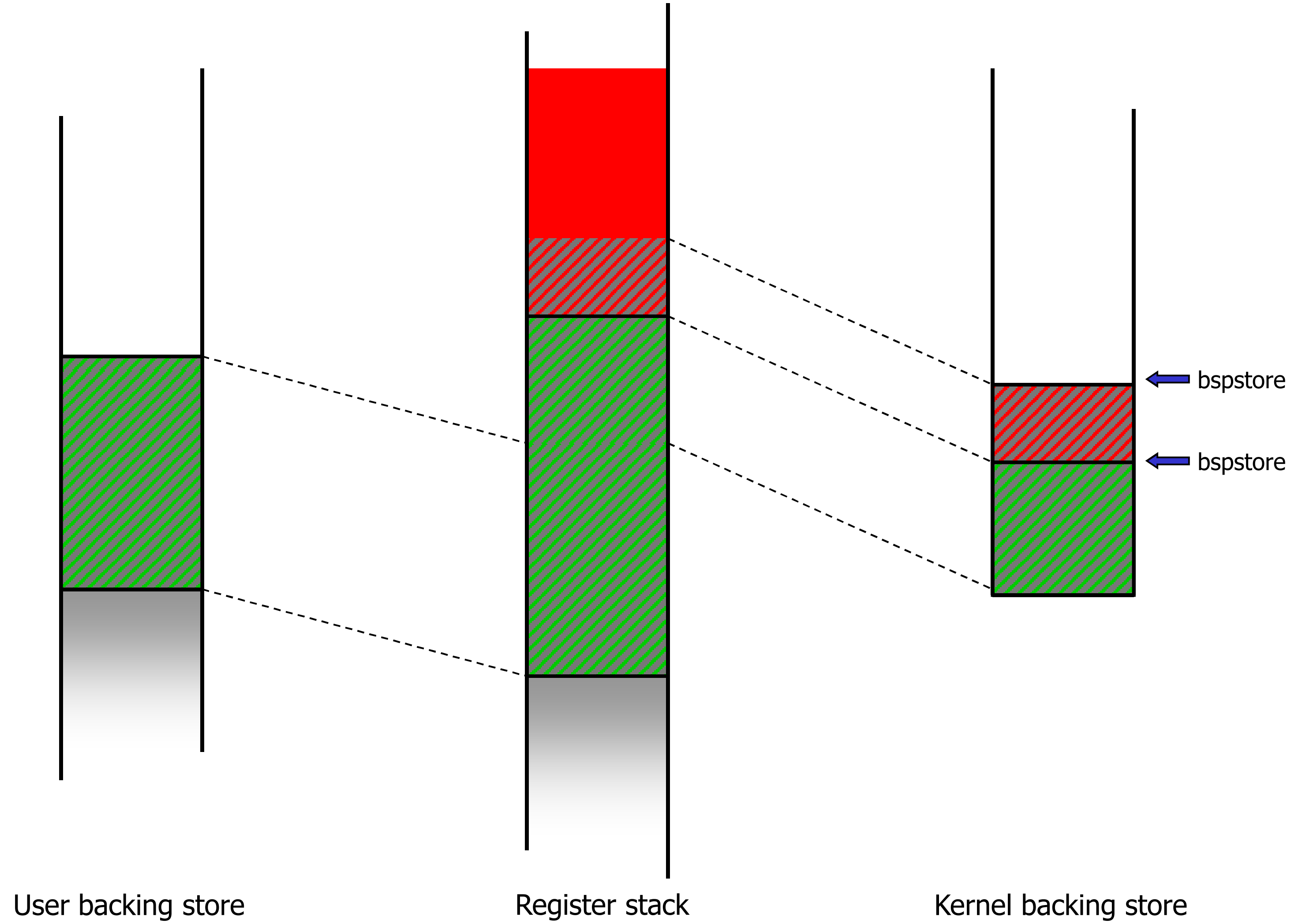
Backing Store Switch for System Calls



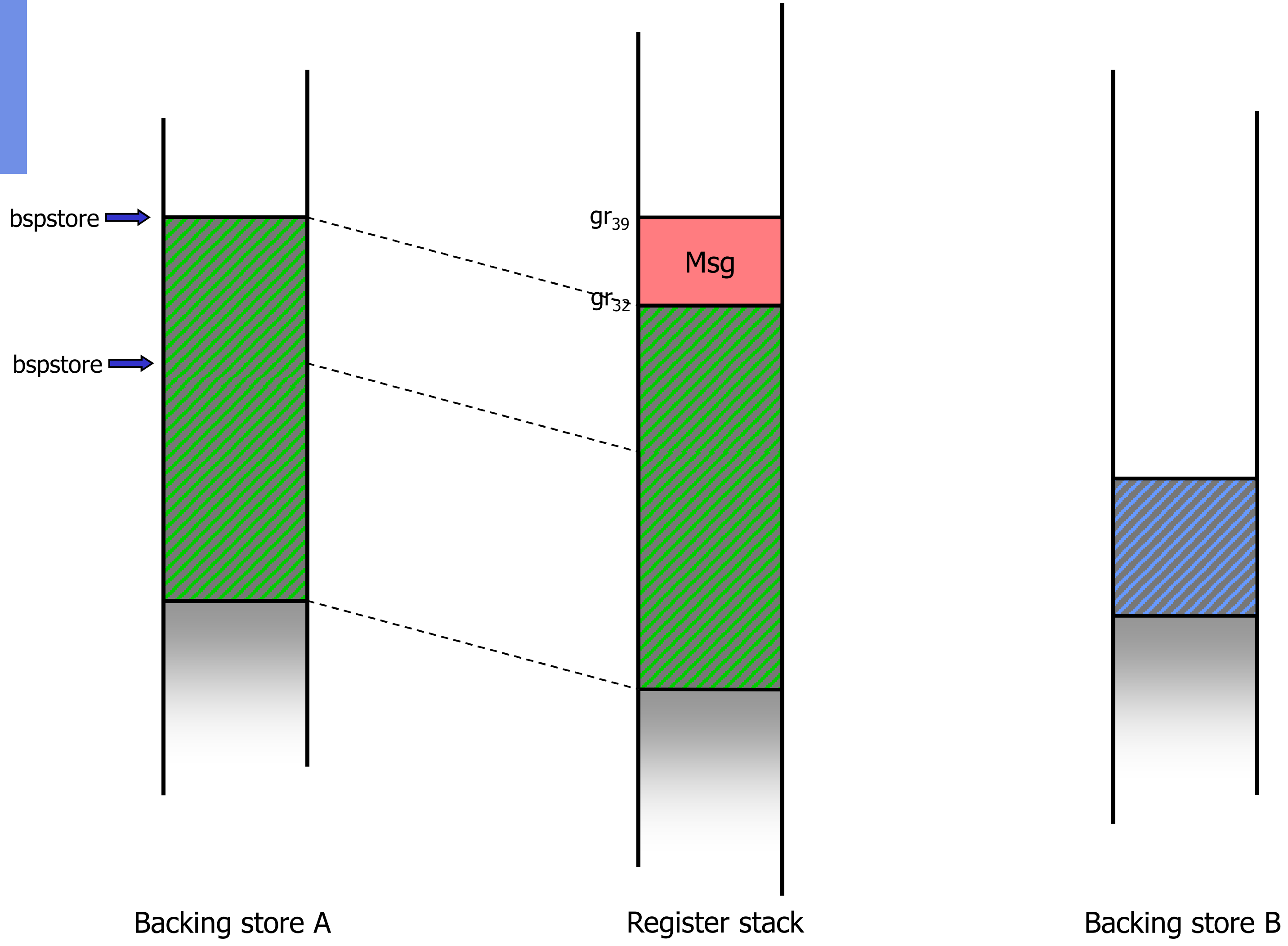
Backing Store Switch for System Calls



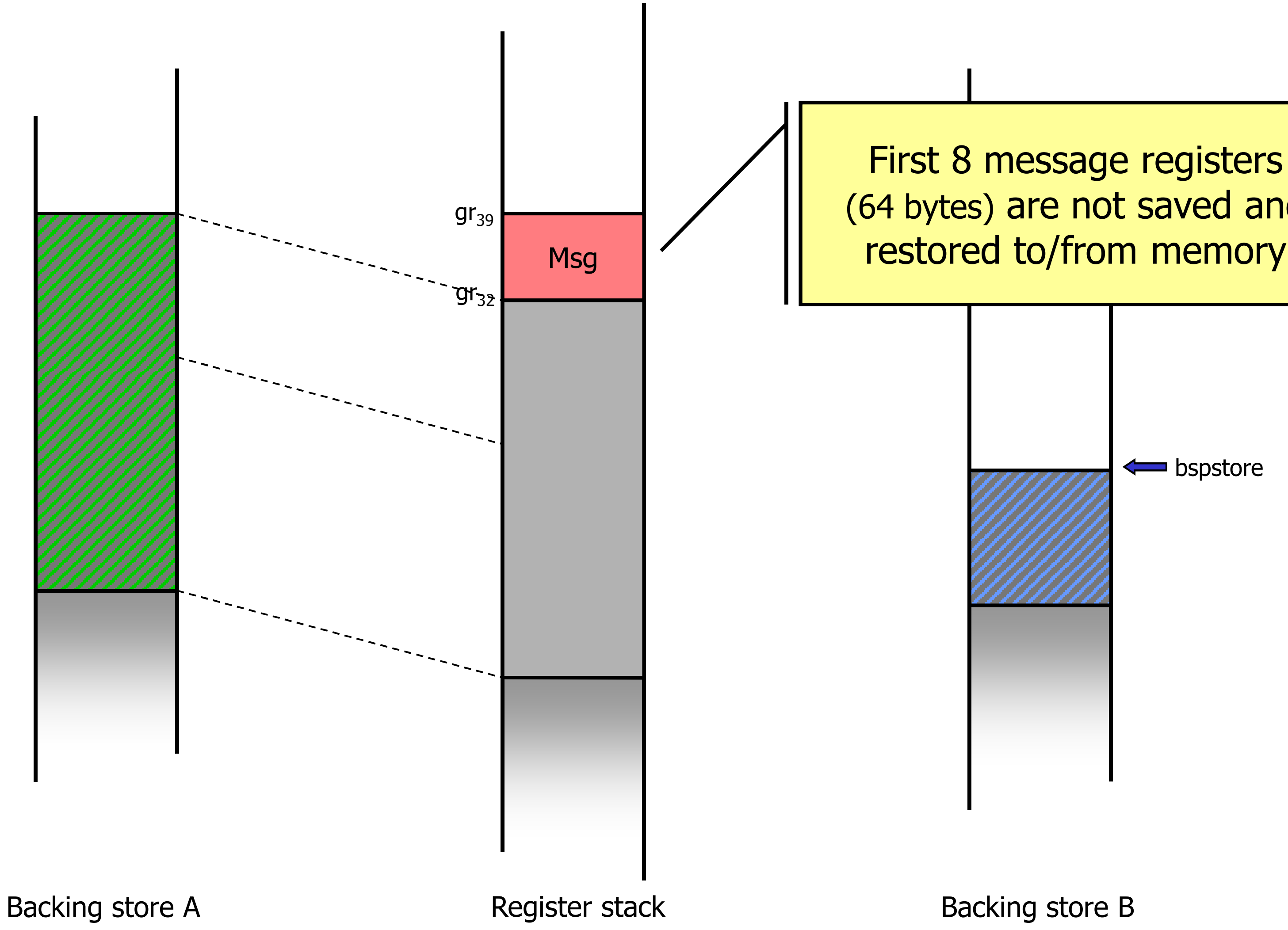
Backing Store Switch for System Calls



Register Stack During IPC



Register Stack During IPC



First 8 message registers (64 bytes) are not saved and restored to/from memory