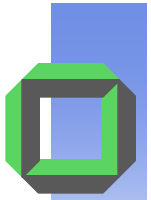
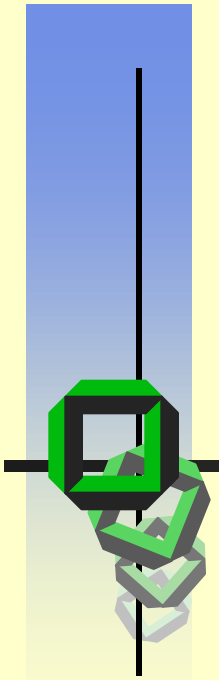


Multiprocessor Solution

```
P(sema S)
begin
 while (TAS(S.flag)==1){};
    { busy waiting }
    S.Count= S.Count-1
    if (S.Count < 0){
       insert_T(S.QWT)
       BLOCK(S)
       {inkl.S.flag=0)!!!}
    }
    else S.flag =0
end
```

```
V(sema S)
begin
 while (TAS(S.flag)==1){};
    { busy waiting }
    S.Count= S.Count+1
    if S.Count ≤ 0 {
       remove_T(S.QWT)
       UNBLOCK(S)
    }
 S.flag =0
end
```

Is this solution already complete?

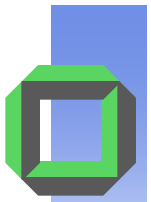


11 Thread Switch & IPC in L4

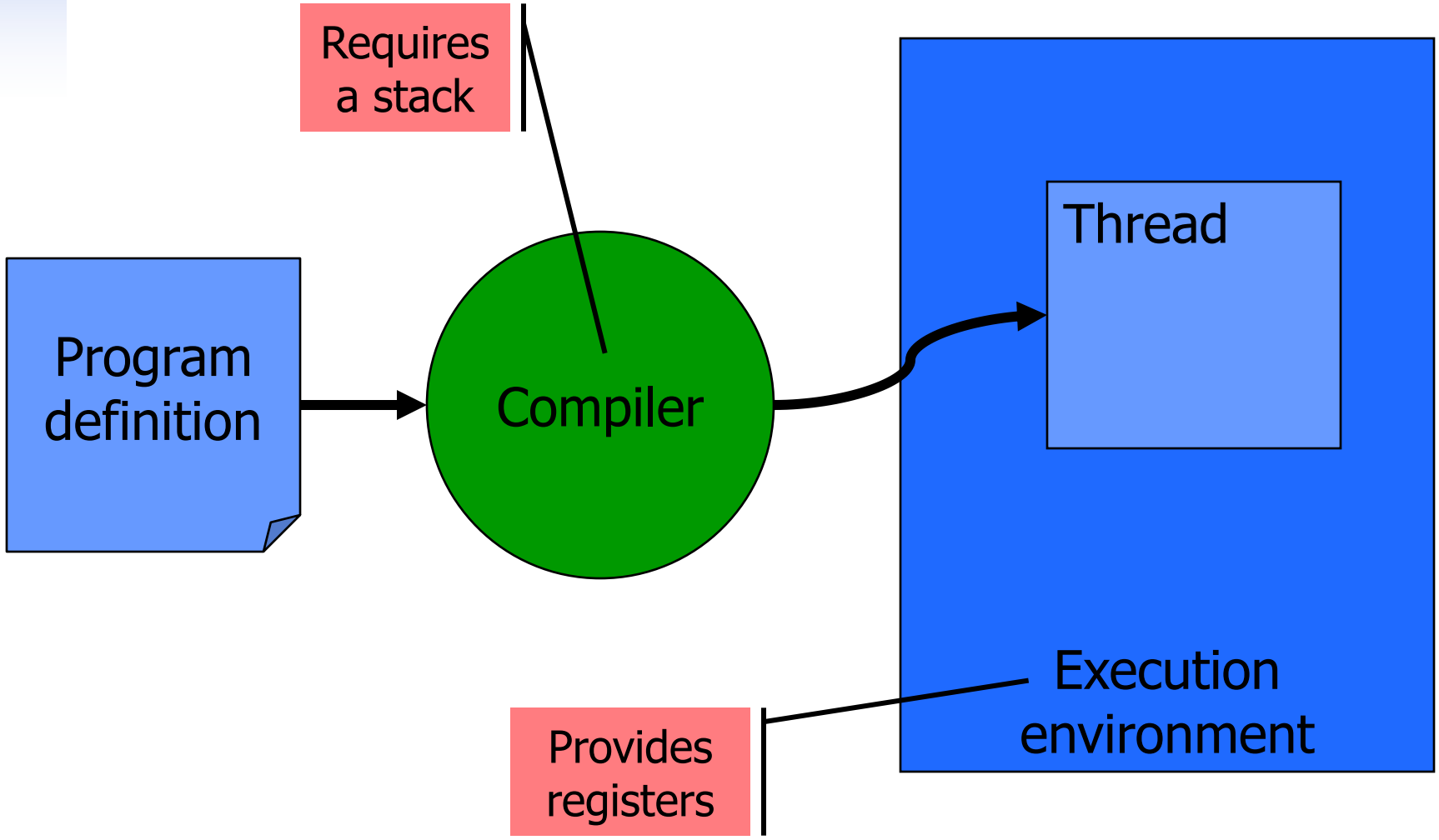
Dezember 3, 2008

Winter Term 2008/2009

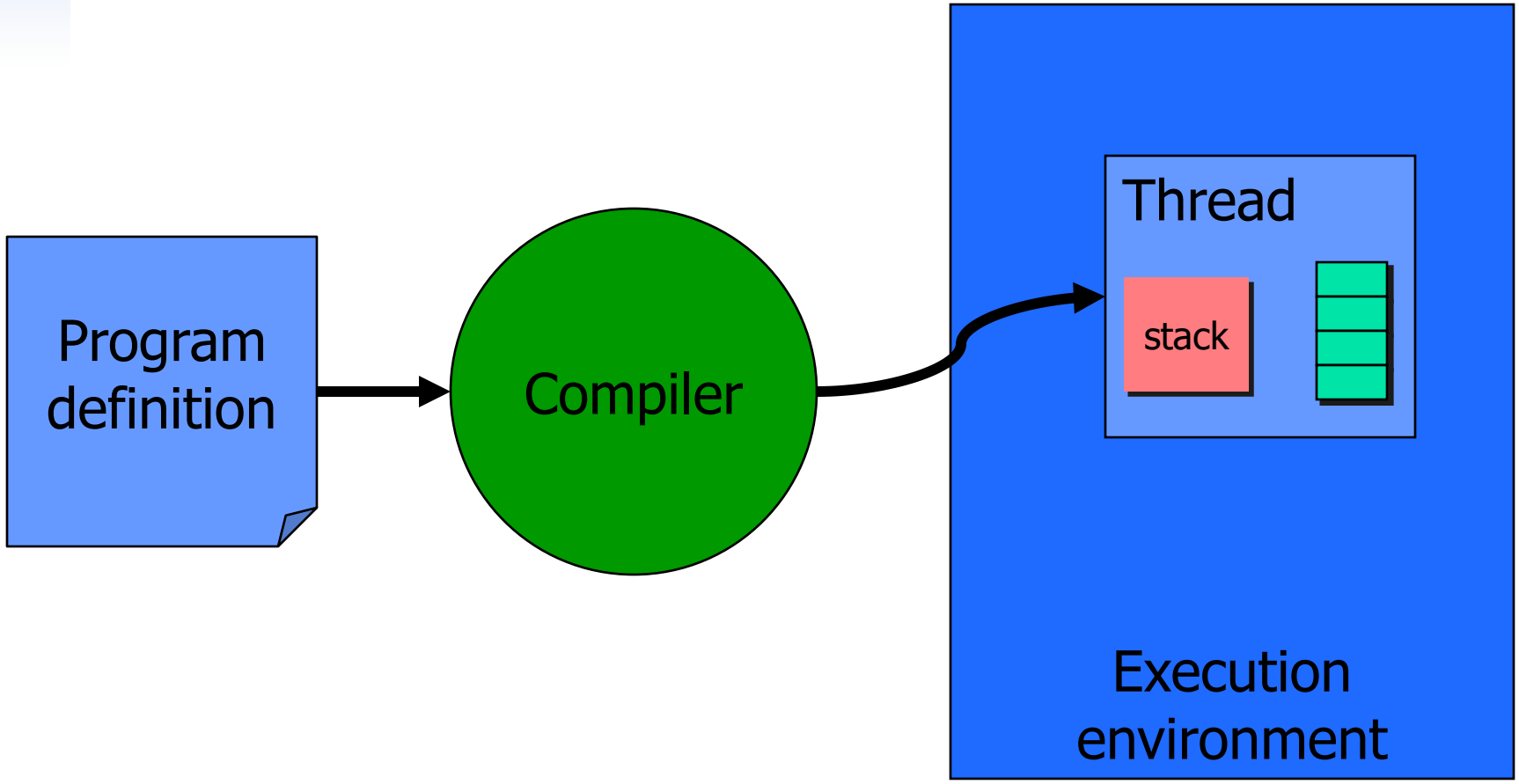
Philipp Kupferschmied

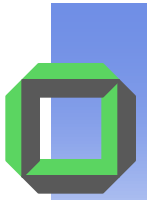


What Defines a Thread?



What Defines a Thread?

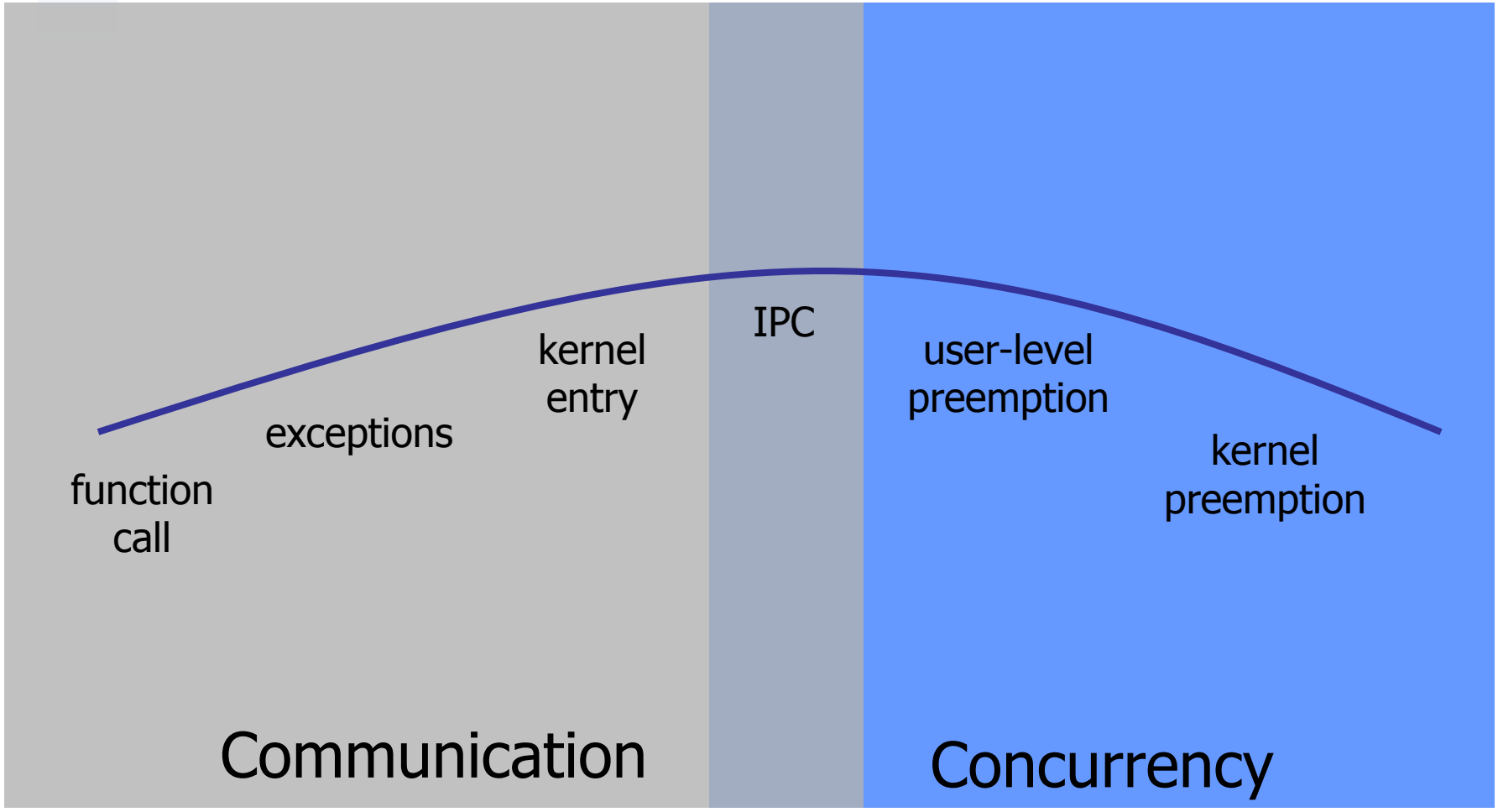




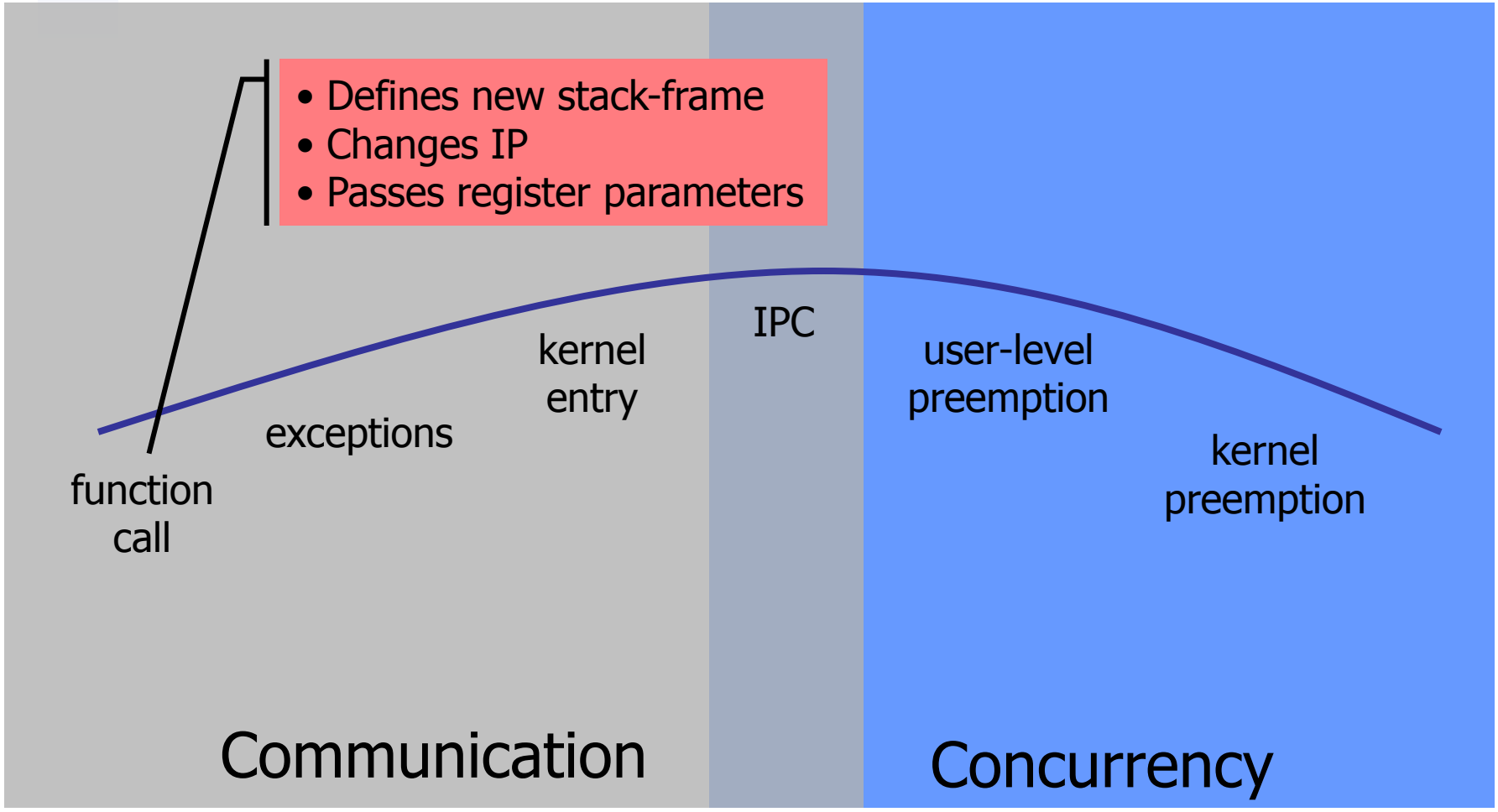
What is a (generalized) Thread Switch?

- Change in the defining parts of a thread
  - Stack pointer
  - Instruction pointer
  - Registers
- Allow later resumption
  - Save all necessary state

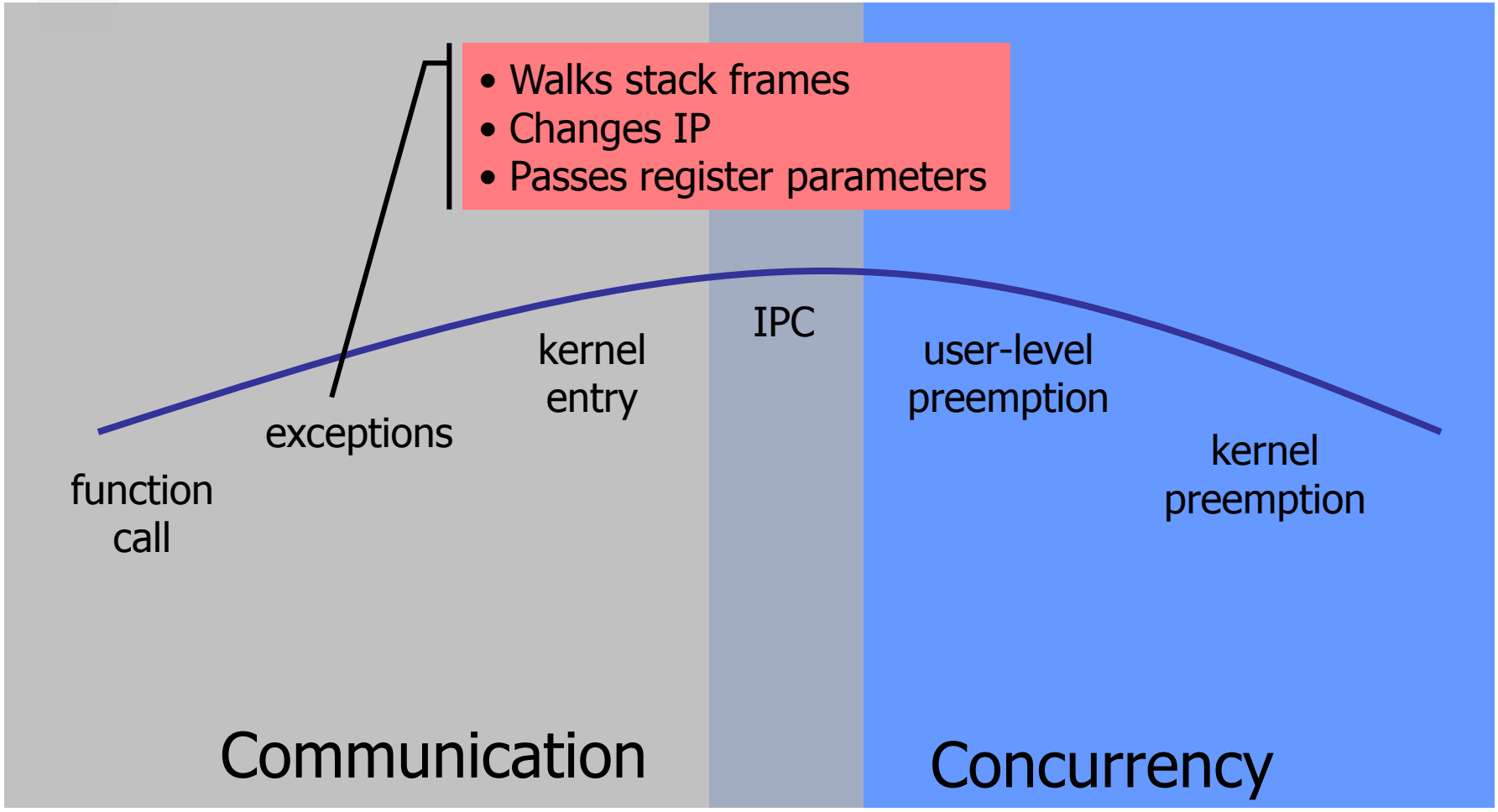
The Stack Switch Spectrum



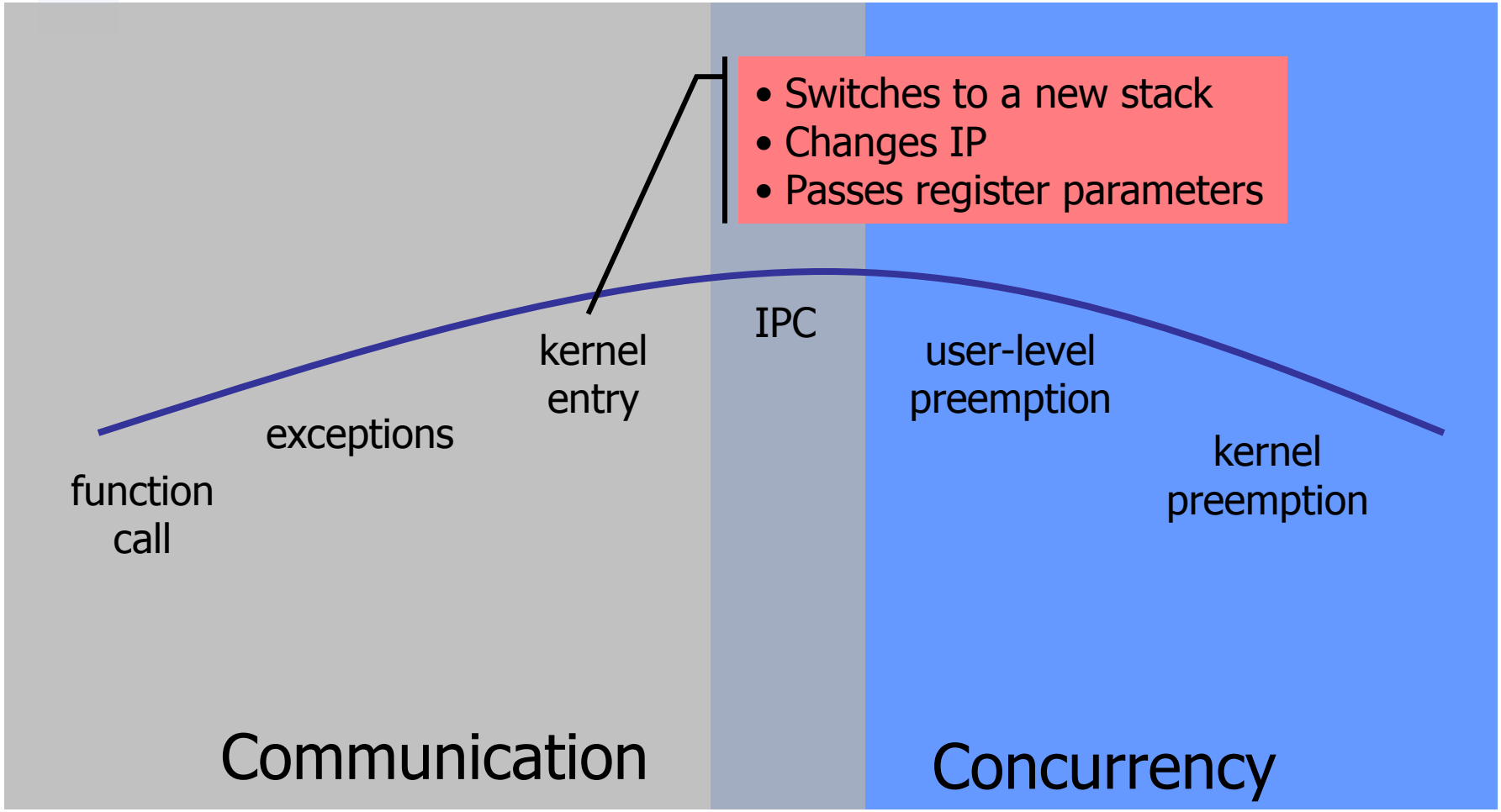
The Stack Switch Spectrum



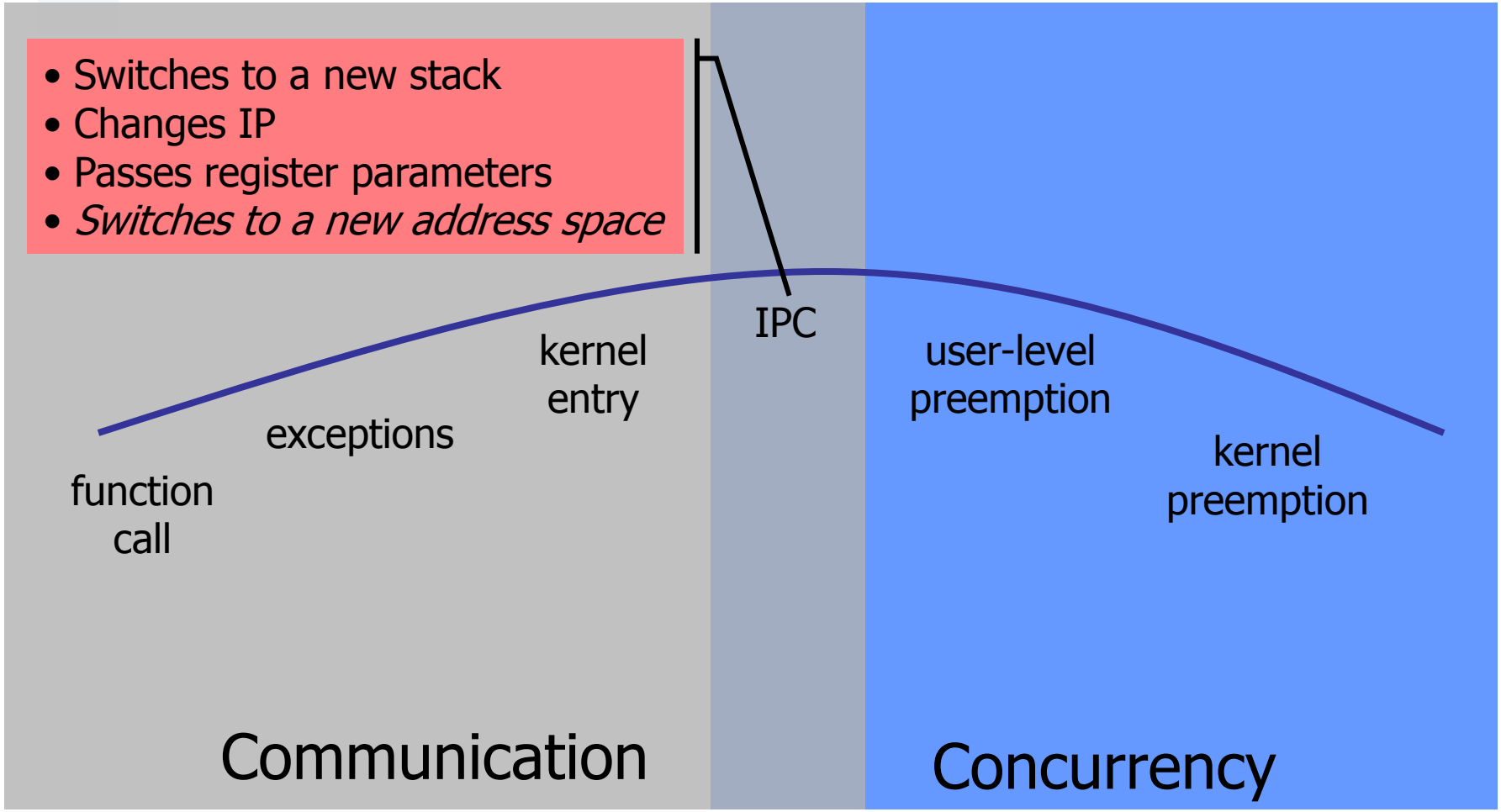
The Stack Switch Spectrum

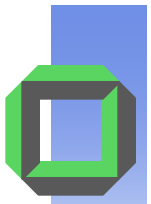


The Stack Switch Spectrum

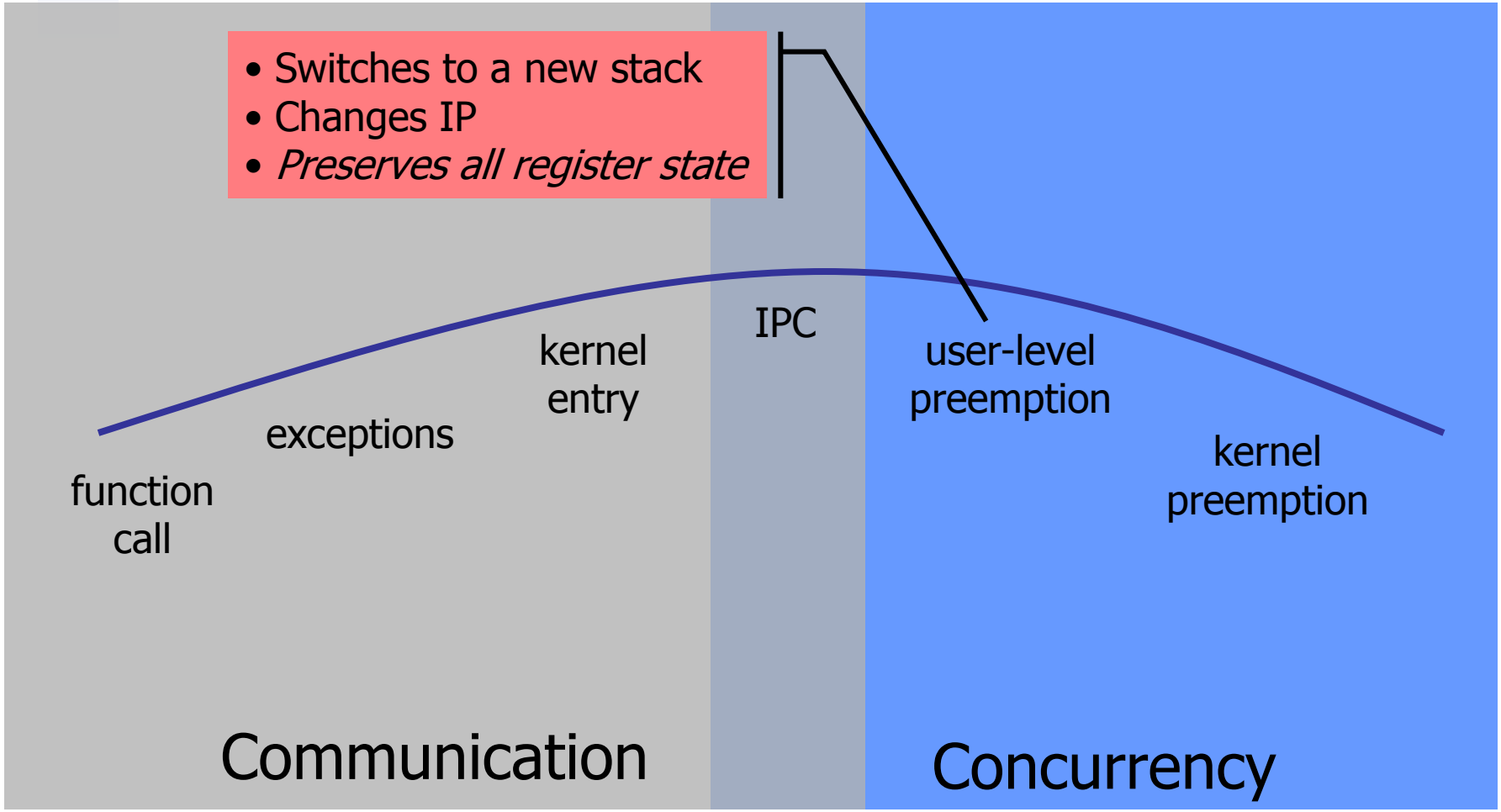


The Stack Switch Spectrum

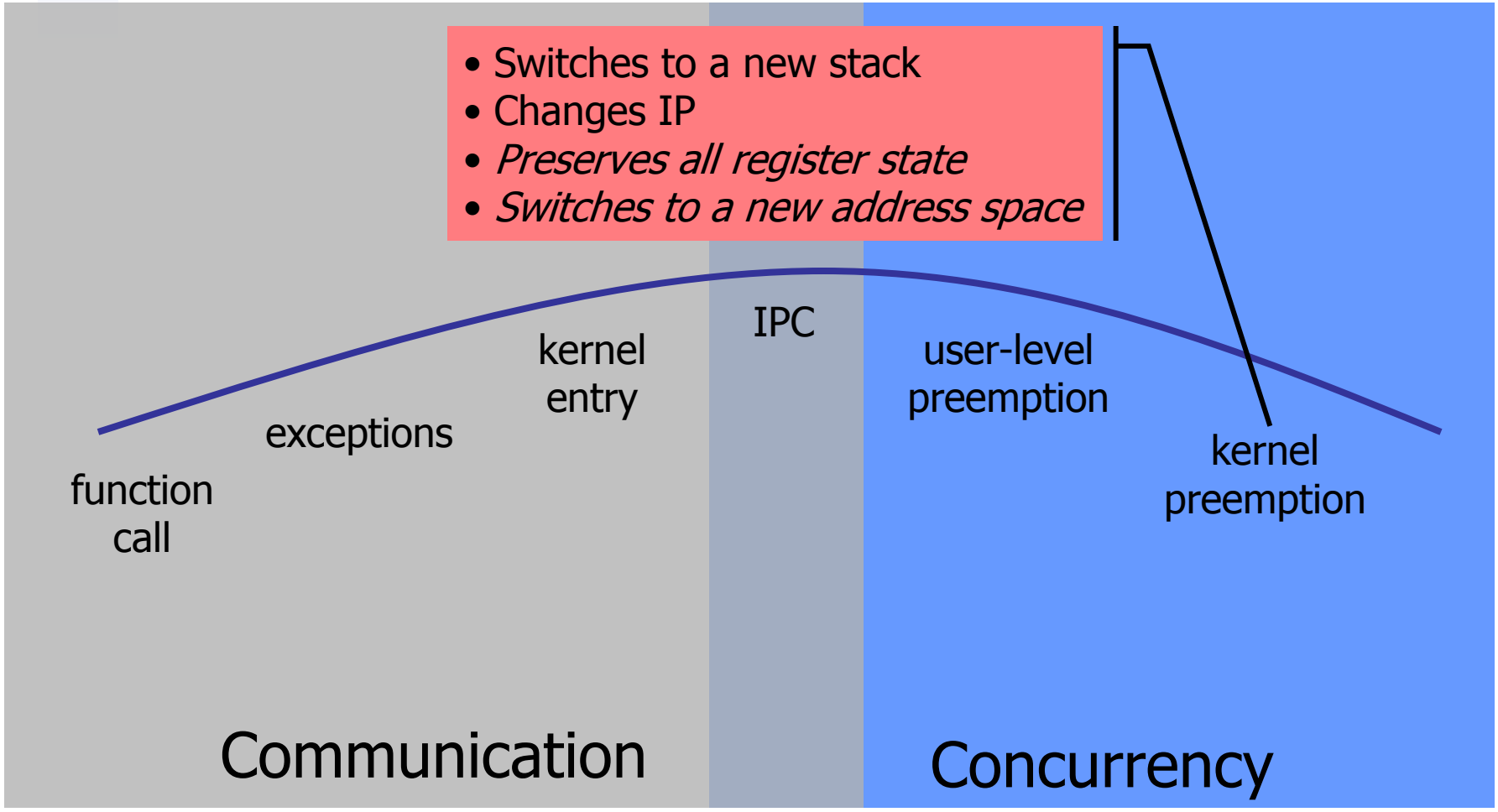




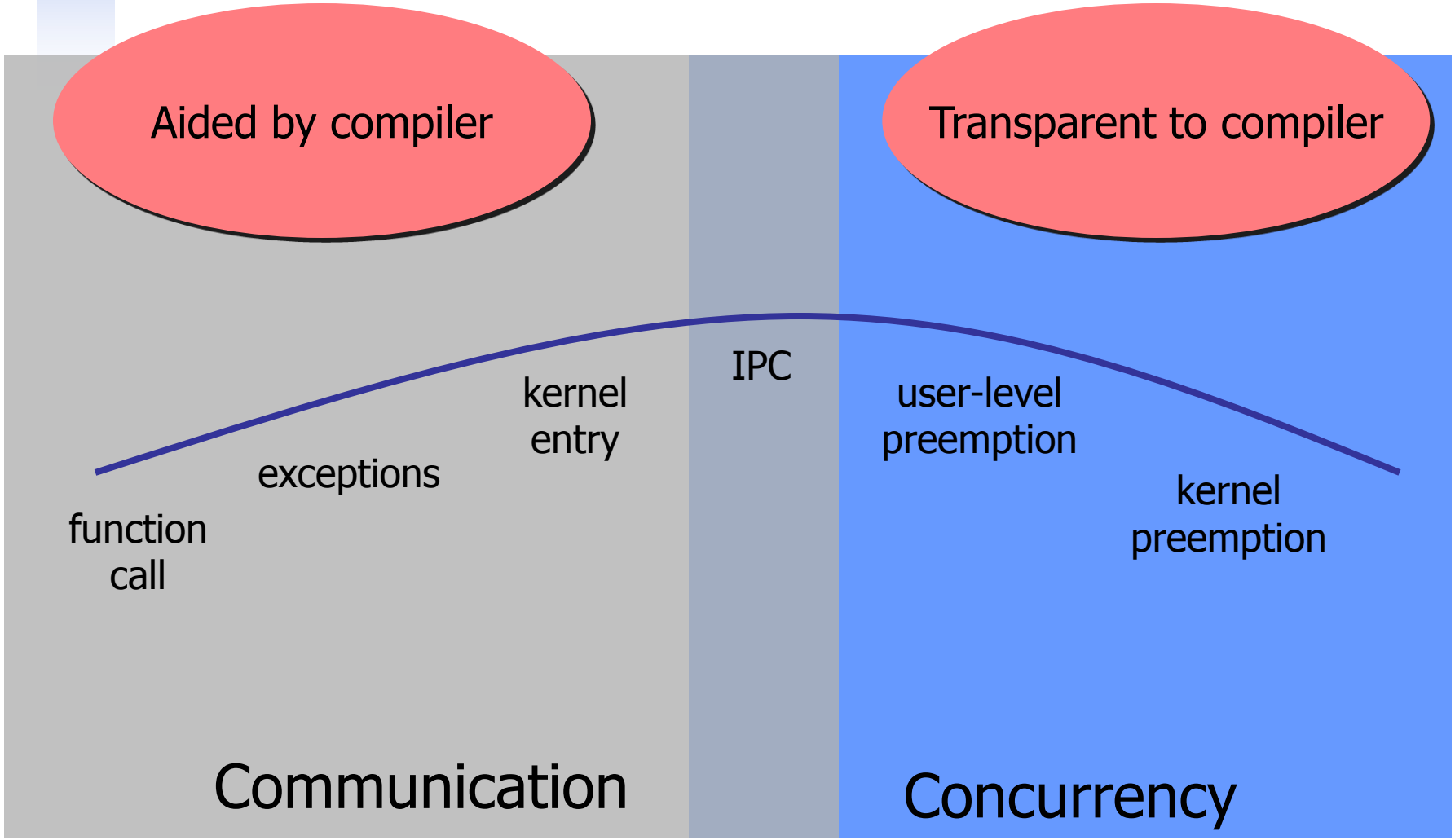
The Stack Switch Spectrum

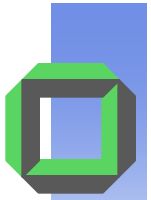


The Stack Switch Spectrum



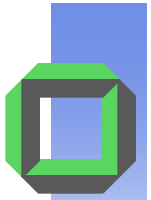
The Stack Switch Spectrum





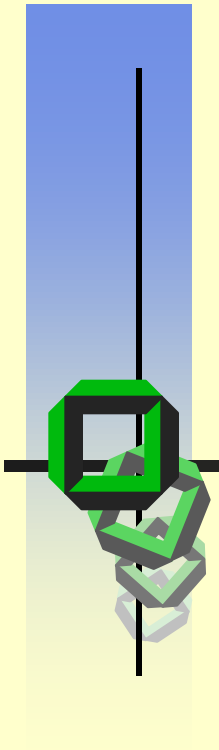
Compiler's Role in Thread Switch

- Cooperative Thread Switch
  - Explicit by program author
  - Supported by compiler
- Preemptive Thread Switch
  - Destructive to compiler's state machine
  - Must be transparent to compiler

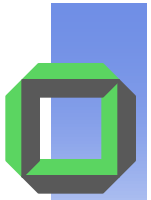


Cooperative vs. Preemptive

- Cooperative (IPC)
  1. Enter the kernel (syscall/trap)
  2. Switch to kernel stack
  3. Cooperative switch to target thread
  4. Restore user thread
  5. Exit kernel
- Preemptive
  1. Enter the kernel (interrupt/exception)
  2. Switch to kernel stack
  3. Cooperative switch to target thread
  4. Restore user thread
  5. Exit kernel

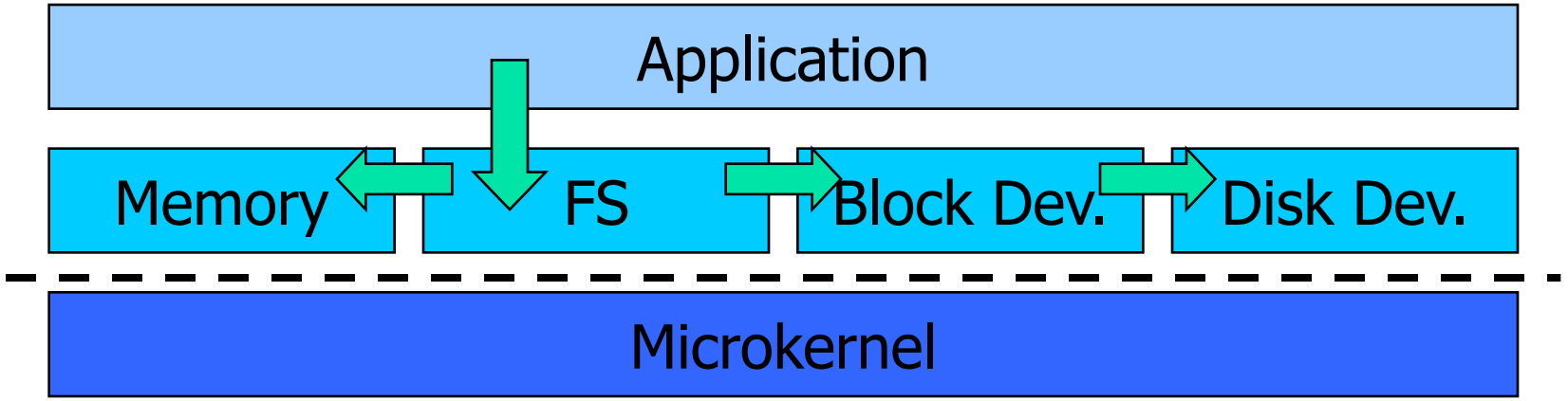


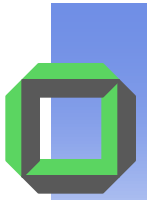
Fast Thread Switch



Scenario

- Multi-threaded computations
  - Weather forecast
  - Air flow simulation
- Multi-server operating systems

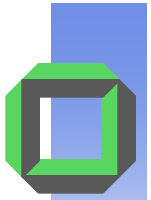




The Goods and the Bads

+ OS designers have power/influence
  - Everybody depends on us
  - We can make applications fast

– OS designers have power/influence
  - Everybody depends on us
  - We can make applications slow



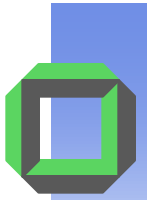
The Goods and the Bads

+ OS designers have power/influence
  - Everybody depends on us
  - We
- OS d
  - Eve
  - We

Linus Torvalds
Date: Fri, 30 Nov 2001 21:15:50 -0800 (PST)

"… If you want to see a system that was more thoroughly designed, you should probably point not to Dennis (*Ritchie*) and Ken (*Thompson*), but to systems like L4 and Plan-9, and people like Jochen Liedtke and Rob Pike. …"

http://kerneltrap.org/article.php?sid=398

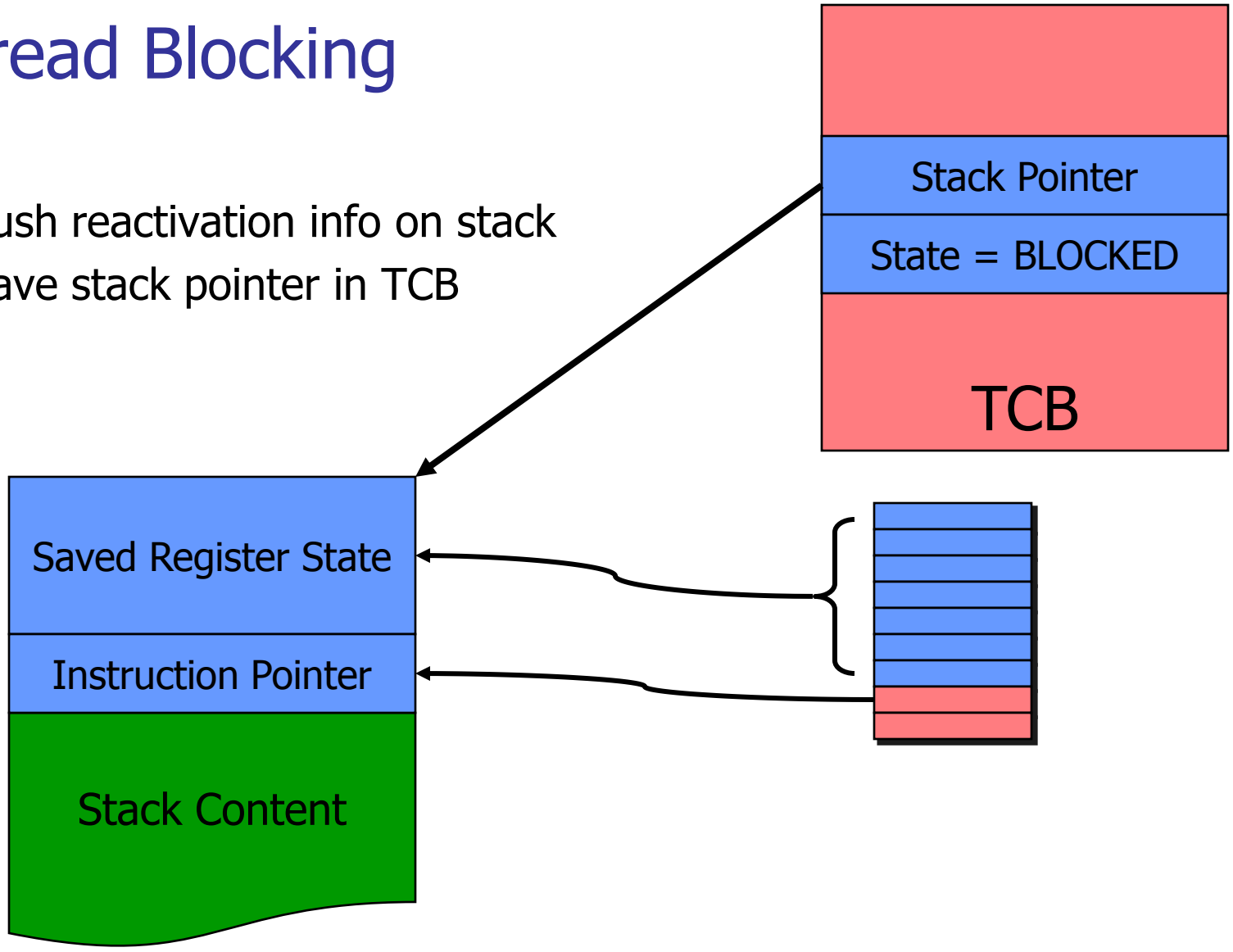


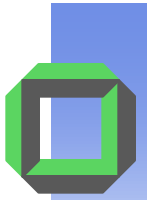
Thread Switch Activation Records

- Where to store thread switch records?
  - On the thread's stack
  - Remember the thread's stack location
- Advantages of using stack
  - Stack proposed anyway
  - Some architectures are stack efficient
    - E.g. IA32

Thread Blocking

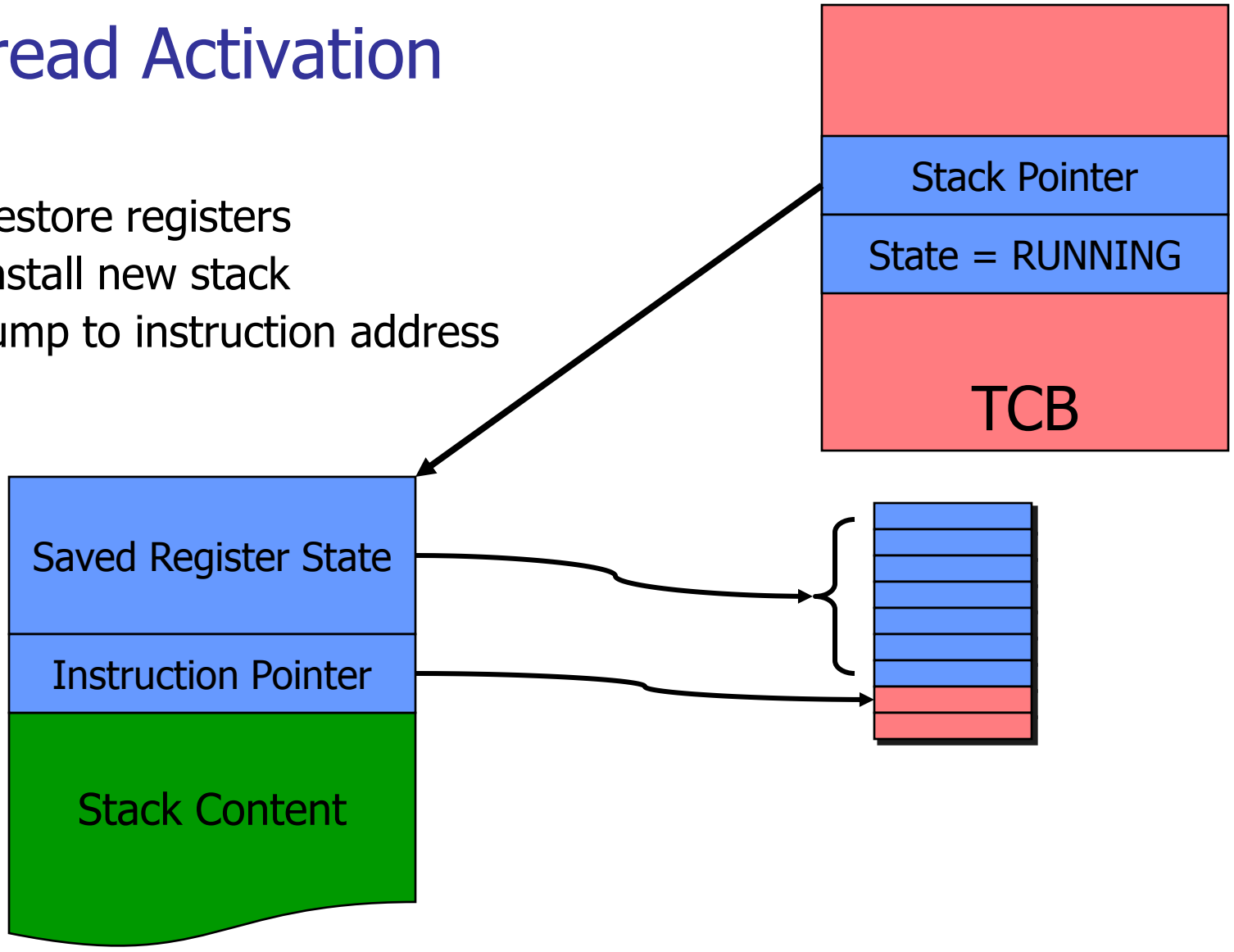
- Push reactivation info on stack
- Save stack pointer in TCB

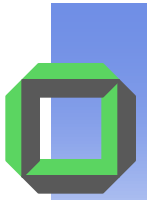




Thread Activation

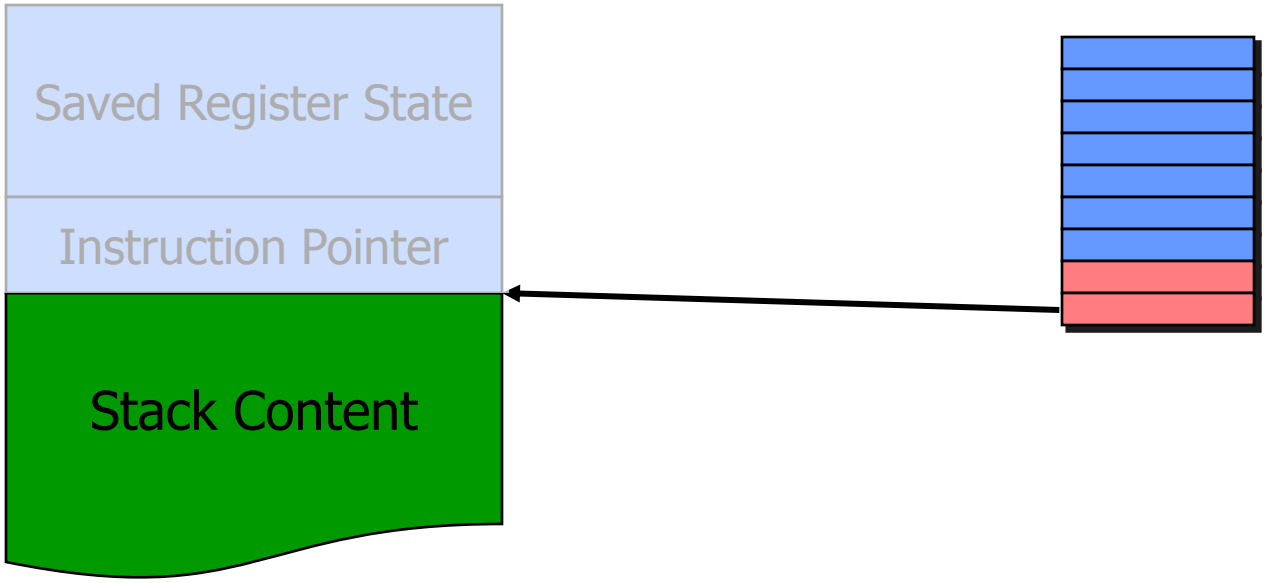
- Restore registers
- Install new stack
- Jump to instruction address





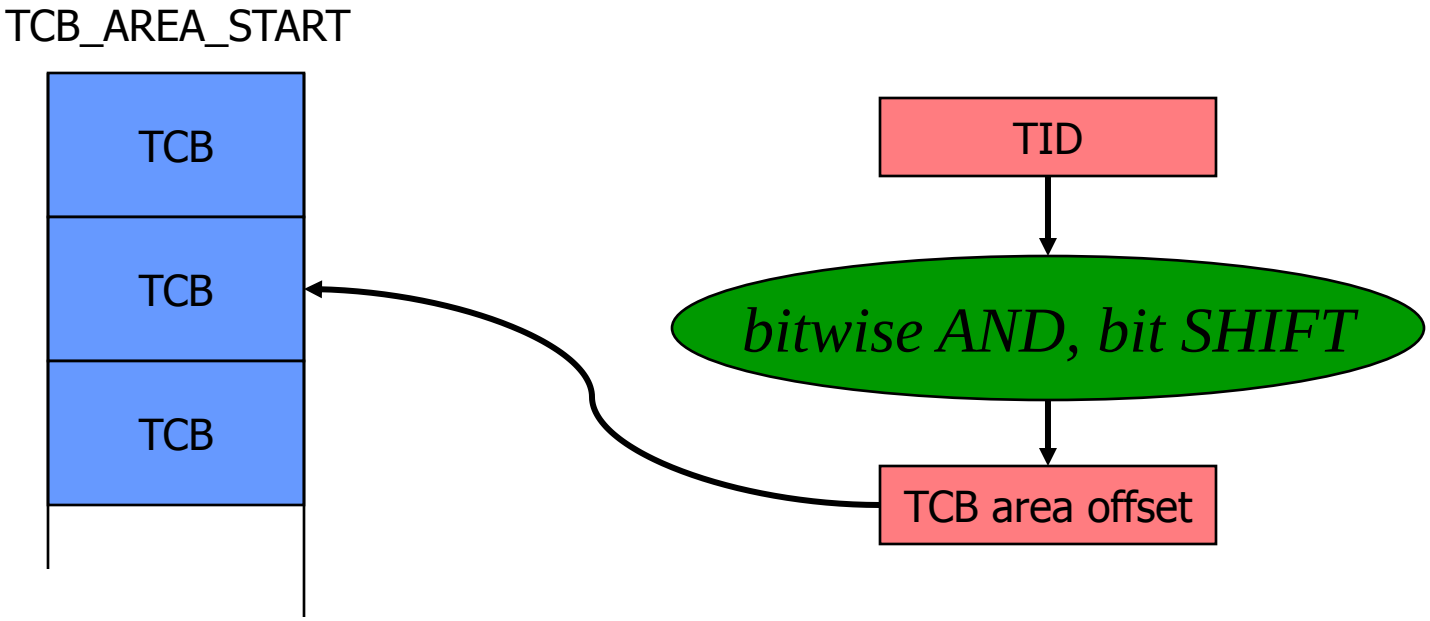
Thread Activation

- Restore registers
- Install new stack
- Jump to instruction address



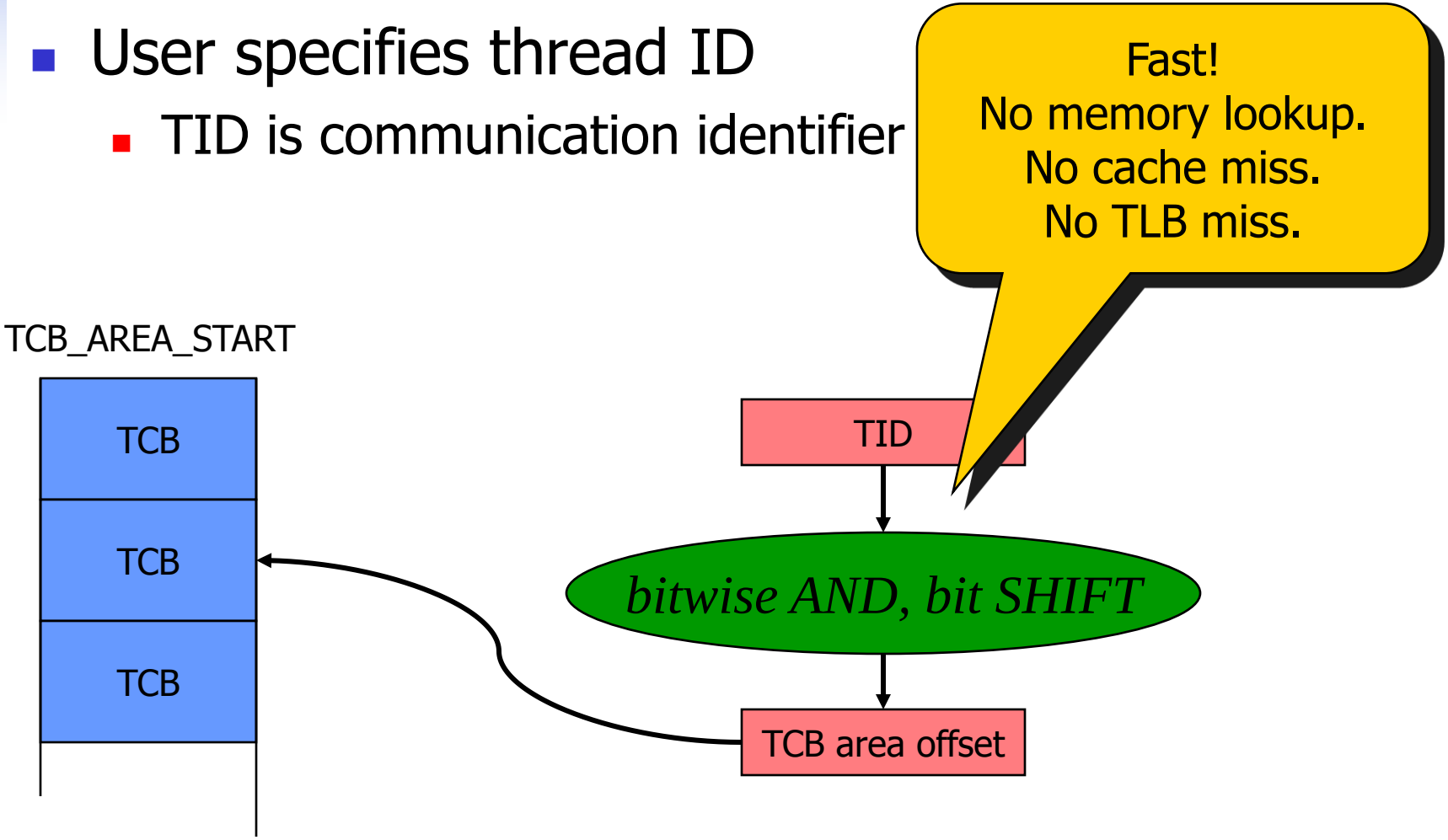
How to Locate the TCB?

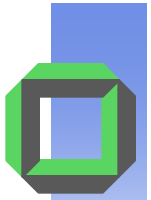
- User specifies thread ID
  - TID is communication identifier



How to Locate the TCB?

- User specifies thread ID
  - TID is communication identifier





Cooperative Switch Example

```
mov ToTcb, %eax
mov CurrentTcb, %ebx
mov Ptab(%eax), %ecx
mov %cr3, %edx
```

- Save registers content (to stack)

```
push %esi
push %edi
push %ebp
push _restart /* IP */
```

- Switch stack

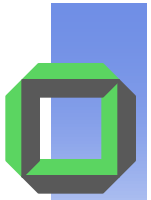
```
mov %esp, Stack(%ebx)
mov Stack(%eax), %esp
```

- Switch address space (if necessary)

```
cmp %ecx, %edx
je _same_as
mov %ecx, %cr3
```

- Restore registers

```
_same_as: ret
_restart: pop %ebp
pop %edi
pop %esi
```



Cooperative Switch in C (using gcc)

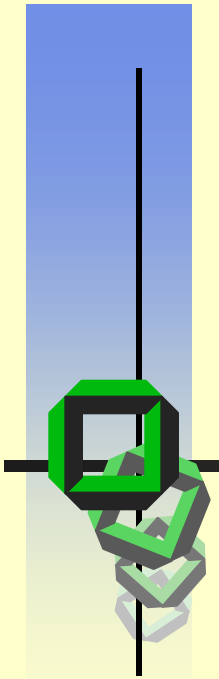
```
inline void SwitchTo(Tcb_t * thread)
{
    inline asm ("
          push    %%ebp              /* save registers */
          push    2f                 /* save IP */
          mov     %%esp, %2          /* switch stack */
          mov     %0, %%esp
          mov     %%cr3, %%edx       /* switch address space */
          cmp     %1, %%edx
          je      1f
          mov     %1, %%cr3
    1:    ret                        /* continue as "thread" */
    2:    pop     %%ebp"             /* restore registers */
          :
          : "m"(thread->Stack),
            "c"(thread->Ptab),       /* ecx contains ptab */
            "m"(current->Stack),
    /*clobber*/ : "eax", "ebx", "ecx", "edx", "esi", "edi", "memory");
}
```

Cooperative Switch in C (using gcc)

```
inline void SwitchTo(Tcb_t * thread)
{
    inline asm ("
            push    %%ebp
            push    2f
            mov     %%esp, %2
            mov     %0, %%esp
            mov     %%cr3, %%edx
            cmp     %1, %%edx
            je      1f
            mov     %1, %%cr3
    1:      ret
    2:      pop     %%ebp"                          res
            :
            : "m"(thr                    ck),
              "c"(                >Ptab),       /* ecx contains ptab */
                                urrent->Stack),
    /*clobber*/: "eax", "ebx", "ecx", "edx", "esi", "edi", "memory");
}
```

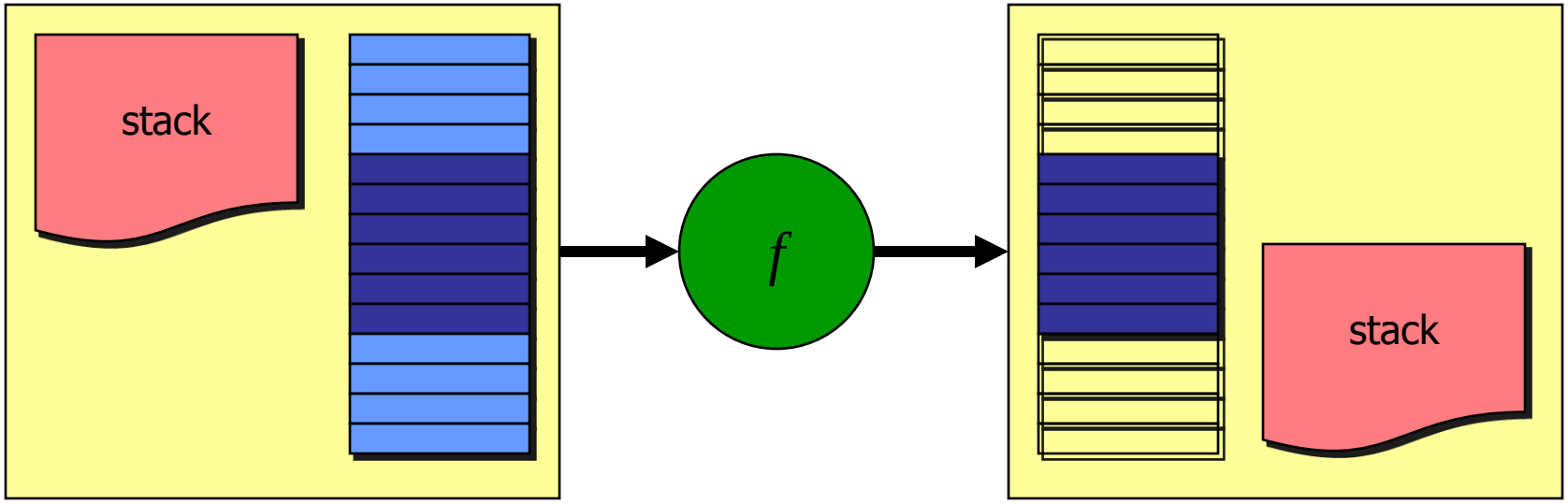
Trick:

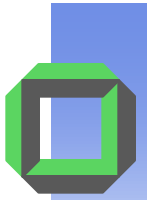
Let **gcc** save and restore the registers... it knows best which are really in use!



Fast IPC in L4

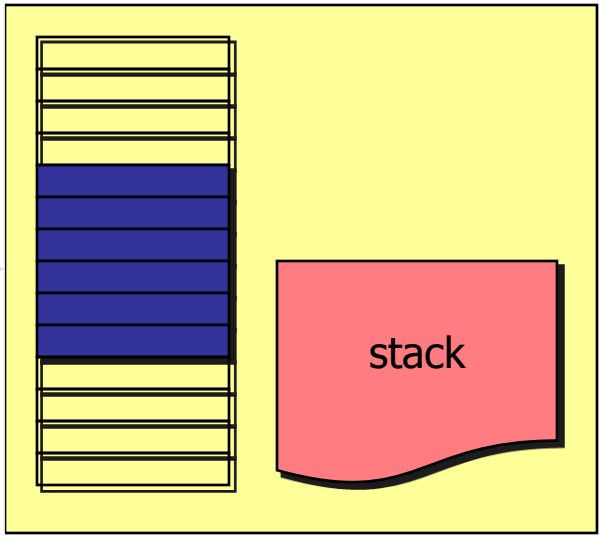
Communication

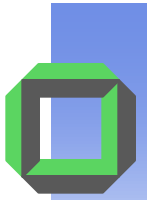




Communication

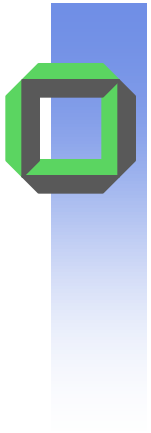
- Parameters in register file
- Avoids memory
  - No cache footprint
  - No TLB footprint
- Compiler optimizations
  - Register allocation
  - Register use



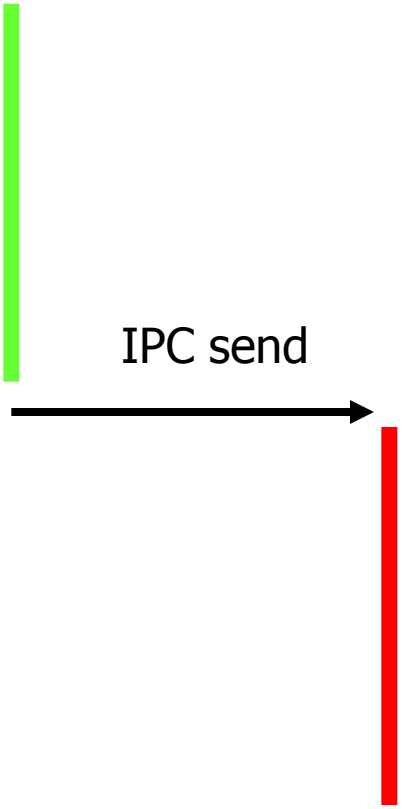


Reasons for High IPC Performance

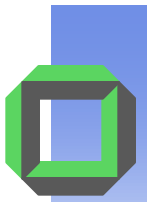
- Synchronous IPC
- Register based
  - Cache footprint!!!
- Hand-optimized



IPC (in Multiple Steps)



1. Start IPC send to red
   1. Enter kernel
   2. Reset finite kernel stack
2. SwitchTo(red)
   1. Kernel thread switch
3. Continue …
   1. Exit to user
   2. Restore user stack



IPC (in Multiple Steps)

Combine into single step.

Skip switch to kernel stack of receiver. Switch directly to target user thread.

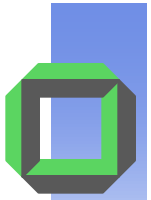
IPC send

Start IPC send to red

1. Enter kernel
2. Reset finite kernel stack

2. SwitchTo(red)
   1. Kernel thread switch

3. Continue …
   1. Exit to user
   2. Restore user stack

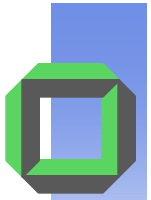


IPC Message Transfer

- Describe message in 4 registers
  - Receiver, IPC type, send+recv structure
- 3 remaining registers (ebx, edx, edi)

  → Message

- Other things to do
  - Enter and exit kernel
  - Do some checks



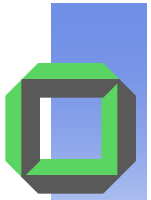
IA32 IPC Implementation

%eax: to_tcb, %ebp: current, (%ebx, %edx, %edi): message

```
TRAP_HANDLER_SEND:
    cmp Partner(%eax), %ebp
    jne block
    cmp State(%eax), WAITING
    jne block
    mov RUNNING, State(%eax)
    push ipc_fastpath_ret
    mov %esp, Stack(%ebp)
    mov Ptab(%eax), %eax
    mov %cr3, %ebp
    cmp  %eax, %ebp
    je   exit_to_user
    mov  %eax, %cr3
exit_to_user:
```

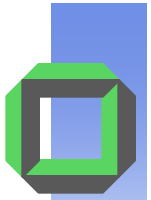
Where is the message transfer?

```
/* eax: snd desc, ecx: USP, ebp: rcv desc, esp: KSP (snd TID), esi: rcv TID */
/* ebx, edx (->ebp), edi: message, 53 instructions in kernel mode */

ipc_sysenter:
      mov       (%esp), %esp
      pushl     $X86_UDS        /* User Data Segment Selector */
      pushl     %ecx            /* save user ESP */
      push      $0              /* fake pushf */
      pushl     $0x1b
      pushl     (%ecx)          /* save user EIP on kernel stack */
      movl      12(%ecx), %ecx  /* load sender's timeouts */
      pushl     %ebp            /* rcv desc */
      pushl     %eax            /* snd desc */
#define to_tcb %eax
#define current %ebp

      orl       %ebp, %eax      /* eax = snd_desc | rcv_desc */
      orl       %ecx, %eax      /* eax = snd_desc | rcv_desc | timeout */

      leal      TCB_AREA(%esi), to_tcb
      jne       ipc_slowpath_to               /* not short call(NEVER) */
      andl      $L4_TIDTCB_MASK, to_tcb
      mov       %esp, current
      andl      $L4_TCB_MASK, current

      cmpl      OFS_TCB_MYSELF(to_tcb), %esi  /* valid TCB for receiver? */
      jne       ipc_slowpath                  /* to_tcb->myself != dest */
      movl      OFS_TCB_THREAD_STATE(to_tcb), %ecx
      add       $1, %ecx
      orl       OFS_TCB_UNWIND_IPC_SP(to_tcb), %ecx
      orl       OFS_TCB_RESOURCES(to_tcb), %ecx
      jne       ipc_slowpath                  /* not waiting or on slow IPC path */
      testb     $1, OFS_TCB_QUEUE_STATE(to_tcb)
      je        ipc_slowpath                  /* not in ready queue */

      /* COND:  ecx == 0 */
      orl       OFS_TCB_PARTNER(to_tcb), %ecx /* to_tcb->partner -> ecx */
      je        1f                            /* open wait */
      cmpl      OFS_TCB_MYSELF(current), %ecx
      jne       ipc_slowpath                  /* to_tcb->partner != myself */
      /* if we reach that point we perform the fast ipc */
1:
      movl      %esi, OFS_TCB_PARTNER(current)      /* current->partner = dest */
      xorl      %esi, %esi
      movl      %esi, OFS_TCB_MSG_DESC(current)     /* current->msg_desc = 0 */
      subl      $1, %esi
      movl      %esi, OFS_TCB_THREAD_STATE(current) /* current->state = WAITING */
      movl      $6, OFS_TCB_THREAD_STATE(to_tcb)    /* to_tcb->state = RUNNING */

      /* create "nice" stack frame for slow path switch */
      pushl     $ipc_fastpath_ret
      movl      %esp, OFS_TCB_STACK(current)  /* save stack */

      /* perform task switch if required */
      movl      %cr3, %ebp                    /* get curr. ptab */
      movl      OFS_TCB_PAGE_DIR(to_tcb), %ecx/* get dst ptab */
      movl      OFS_TCB_MYSELF(current), %esi /* set sender */
      cmpl      %ebp, %ecx                    /* same ptab? */
      je        2f                            /* do not reload */
      movl      %ecx, %cr3                    /* load new ptab */

2:
      /* we don't care about the other guy's stack :) */
      addl      $L4_TOTAL_TCB_SIZE - 4, %eax
      movl      %eax, __tss + 4/* prepare dst's kernel stack */
      movl      %edx, %ebp     /* save msg0 around sysexit */
      movl      -8(%eax), %ecx /* user ESP (stored there above) */
      movl      -20(%eax), %edx/* user EIP (stored there above) */
      addl      $16, %ecx
      subl      $2, %edx
      subl      %eax, %eax
      sti
      sysexit

ipc_slowpath:
      /* call a C function to handle more complex cases */
      ...
```



What you've learned

- We can communicate across a thread switch
- How to perform some generic thread switching
- How to build the worlds fastest IPC path



More details

- http://i30www.ira.uka.de/
  - Microkernel Construction (Lecture)
- http://l4ka.org/
  - L4Ka microkernel project