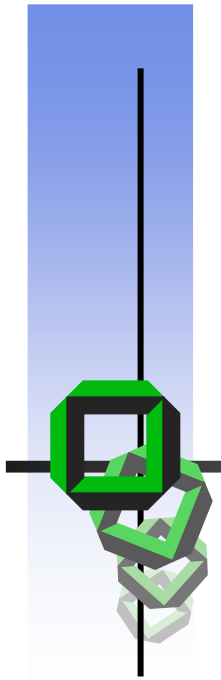


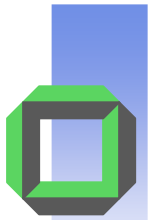
µ-Kernel Construction (10)

Interrupts, Exceptions, and CPU virtualization

and some IA-32 hacks

(updated on 2009-07-20)

© 2009 System Architecture Group, Universität Karlsruhe

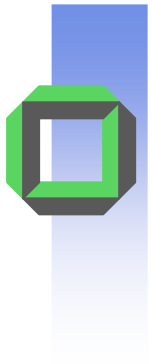


L4 Kernel Paradigm

Everything the kernel needs to handle in a secure manner will either become invisible or be hidden behind an abstraction.

- Exceptions not handled by the kernel itself will be posted to user land
  - Page faults
  - Hardware interrupts
  - Other exceptions

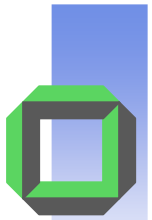
© 2009 System Architecture Group, Universität Karlsruhe



Exception Sources

- From current instruction stream
  - Page fault
  - Numeric
  - Unaligned data access
  - Debug
  - Speculation
- External
  - Device interrupts
  - Timer interrupt
  - Cross-processor interrupt

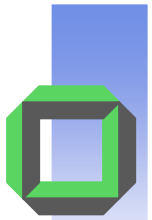
© 2009 System Architecture Group, Universität Karlsruhe



Exception Classes

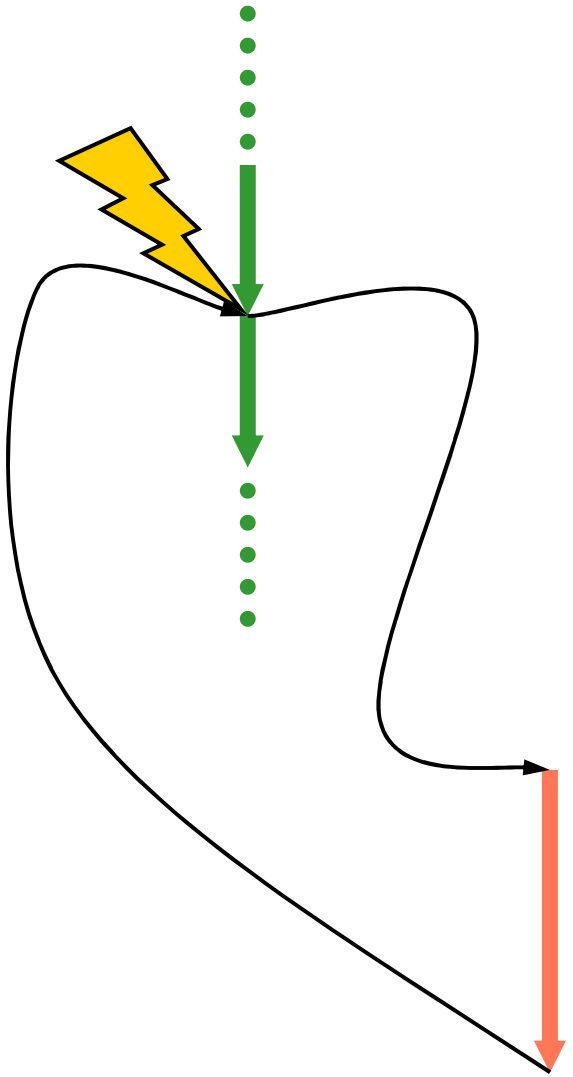
- Traps
  - Sensed after an instruction
  - Deal with the cause, then continue
- Faults
  - Signaled during execution of current instruction
  - Fix the problem, then retry (or skip)

© 2009 System Architecture Group, Universität Karlsruhe

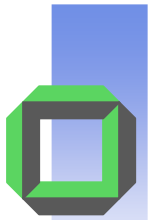


Exception Handling

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



© 2009 System Architecture Group, Universität Karlsruhe



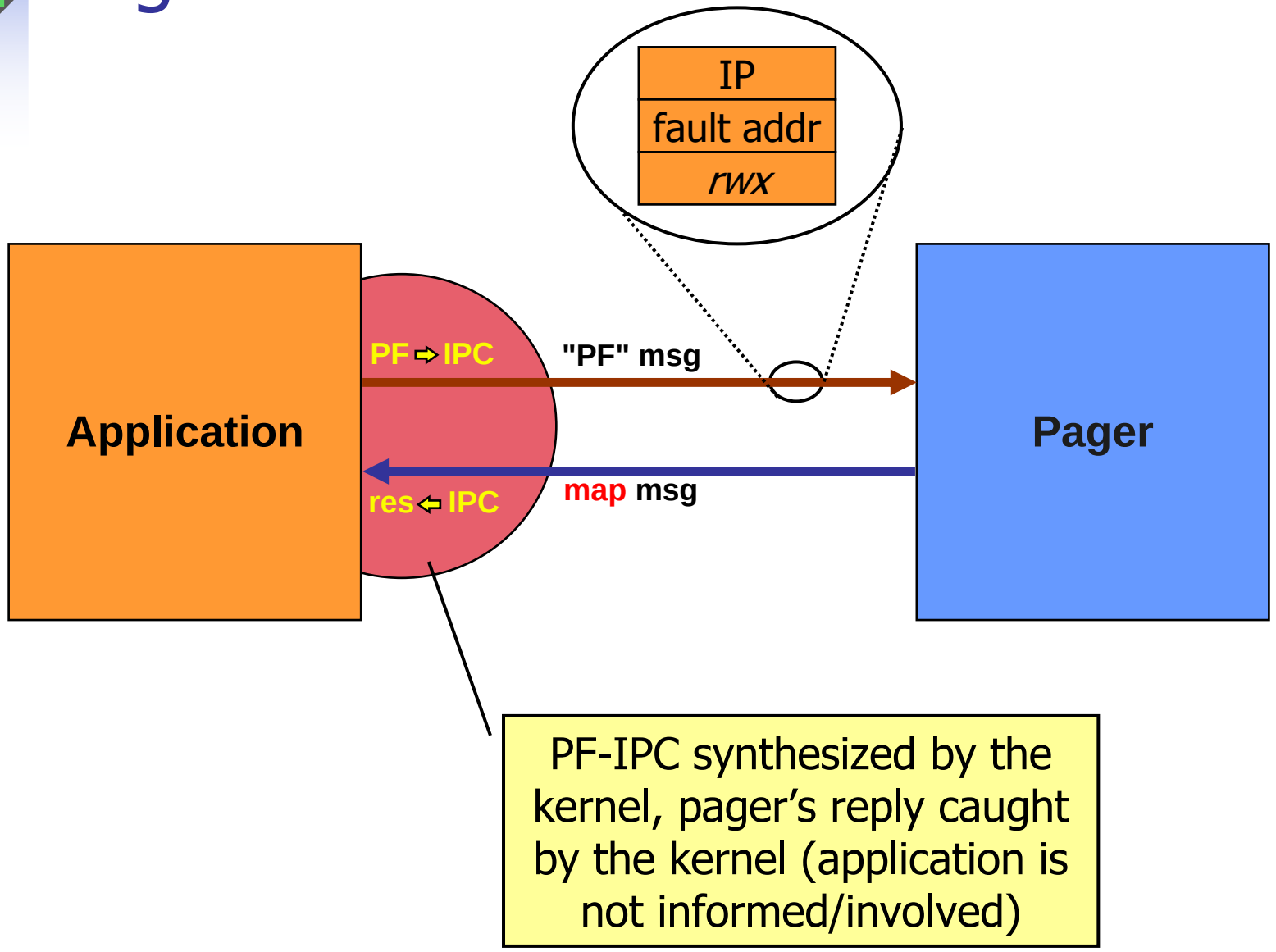
L4 Kernel Paradigm

Everything the kernel needs to handle in a secure manner will either become invisible or be hidden behind an abstraction.

- Exceptions not handled by the kernel itself will be posted to user land
  - **Page faults**
  - Hardware interrupts
  - Other exceptions

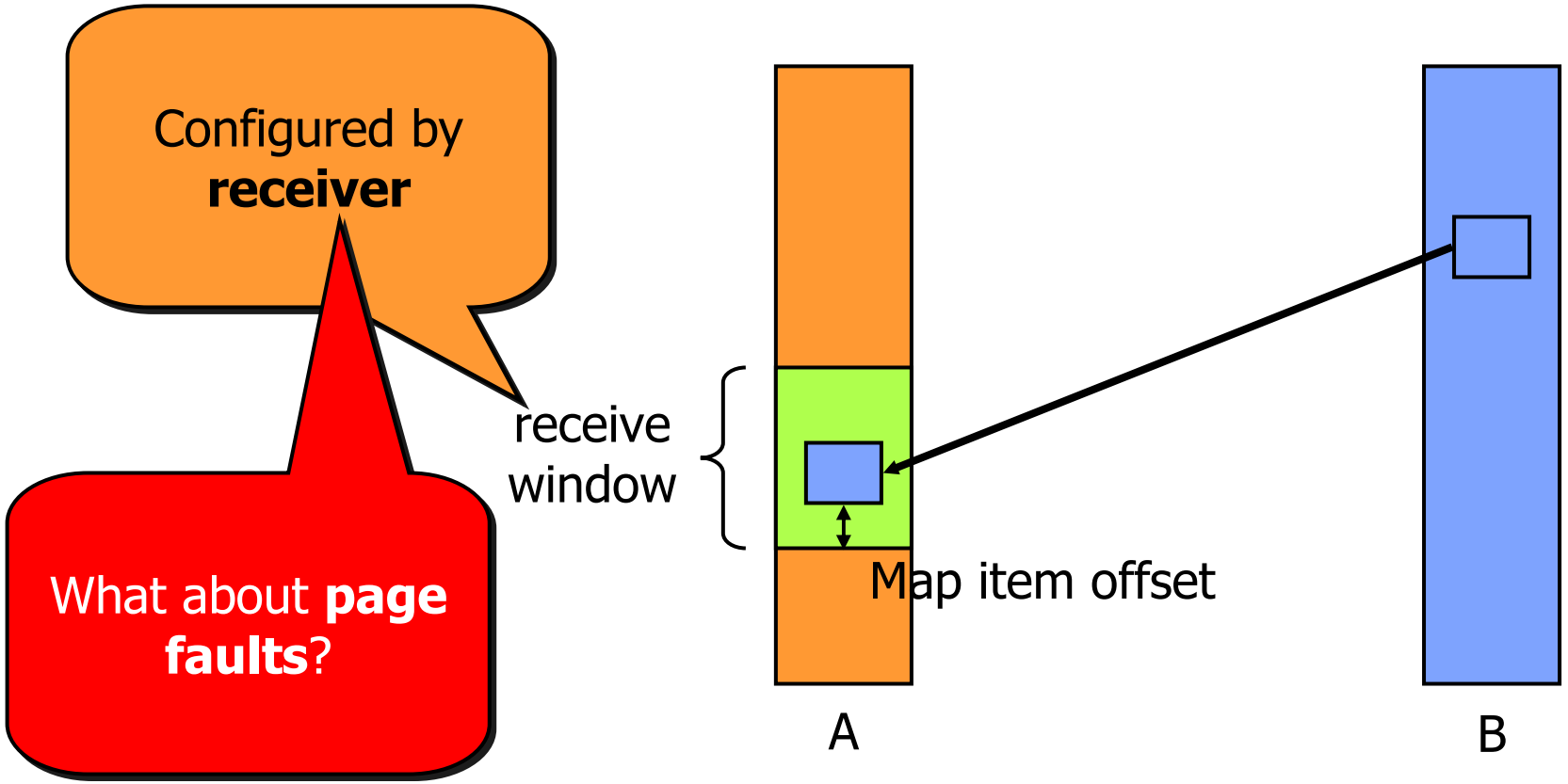
© 2009 System Architecture Group, Universität Karlsruhe

Page Fault IPC



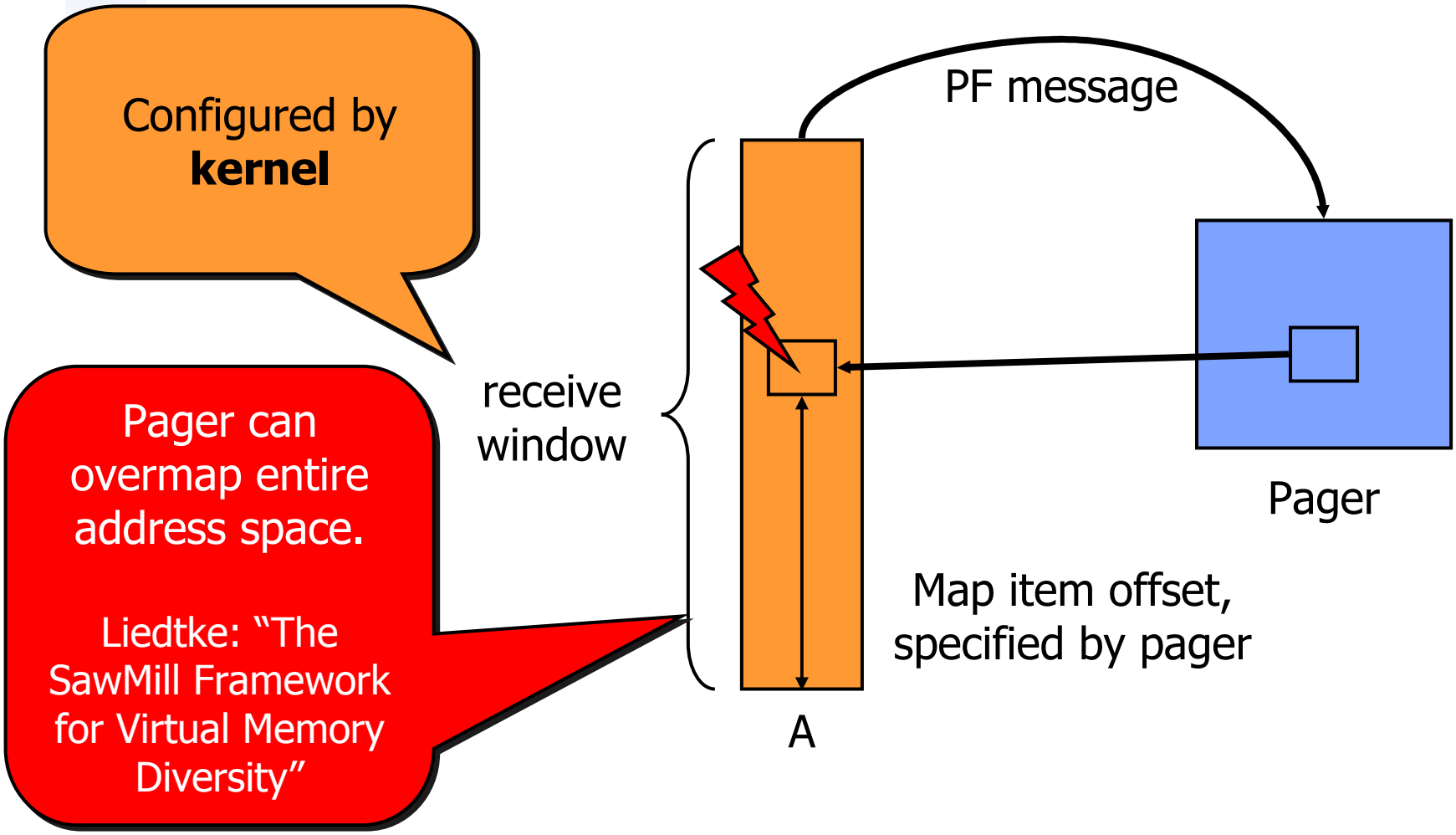
© 2009 System Architecture Group, Universität Karlsruhe

IPC Map



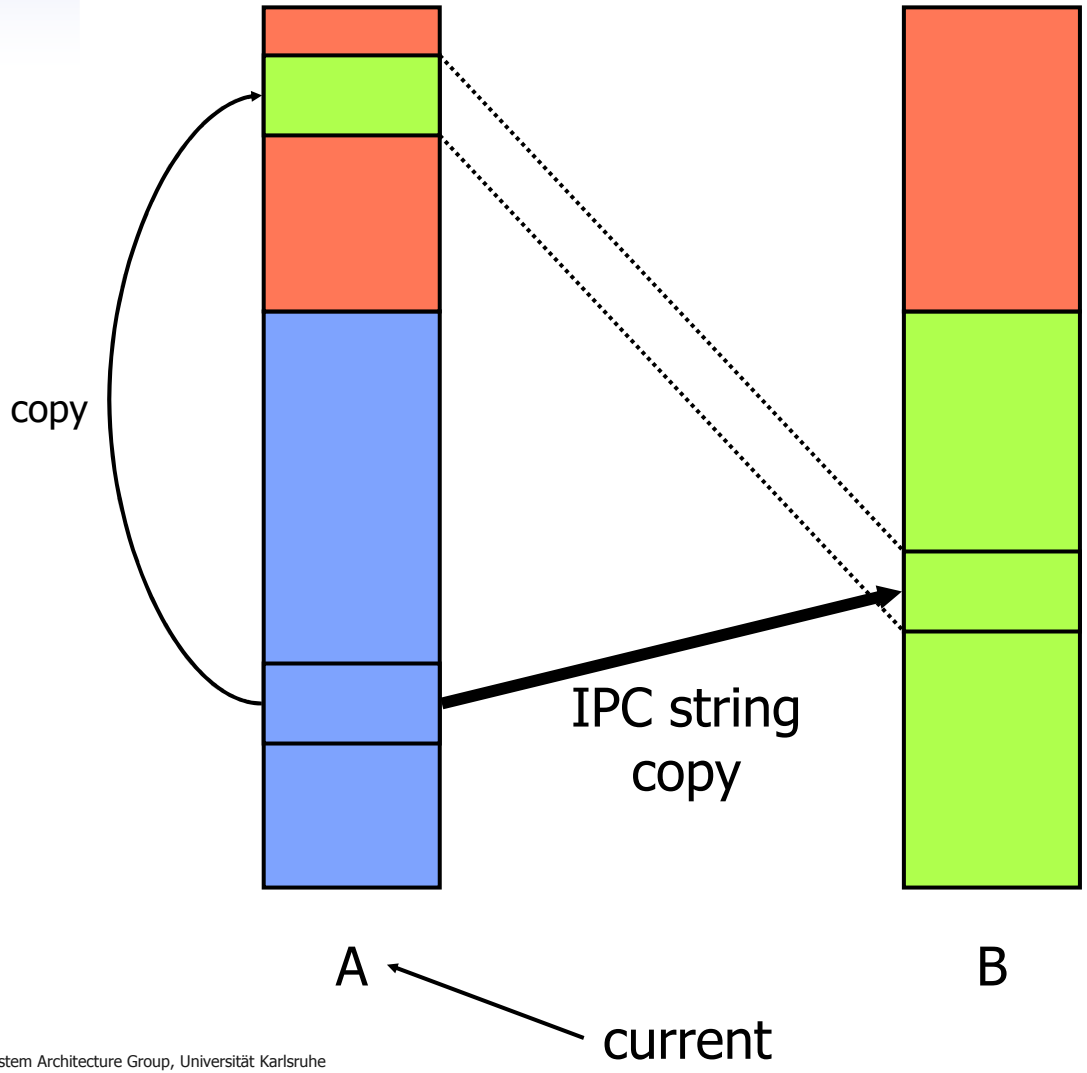
© 2009 System Architecture Group, Universität Karlsruhe

Page Fault Receive Window



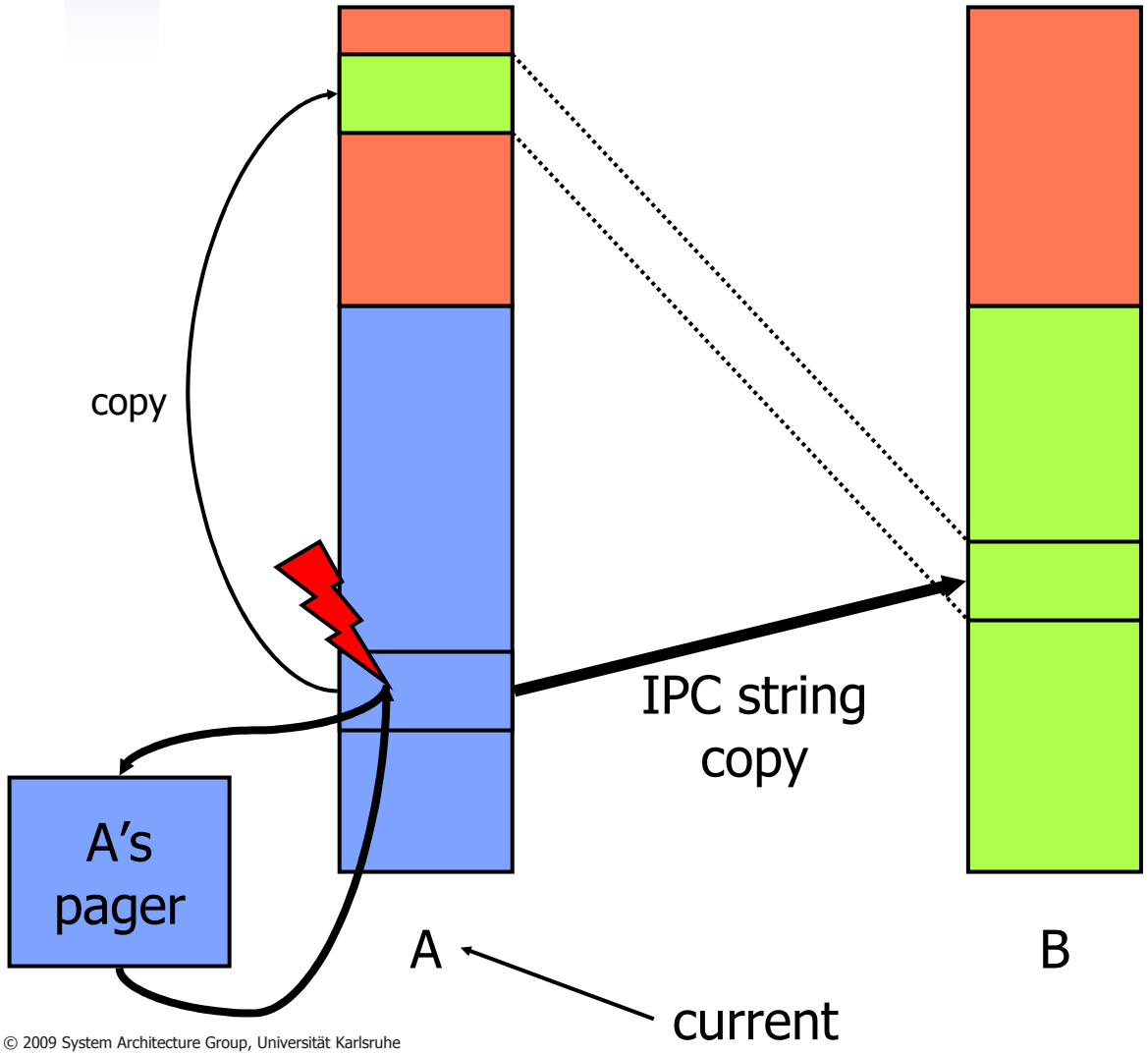
© 2009 System Architecture Group, Universität Karlsruhe

String Copy



© 2009 System Architecture Group, Universität Karlsruhe

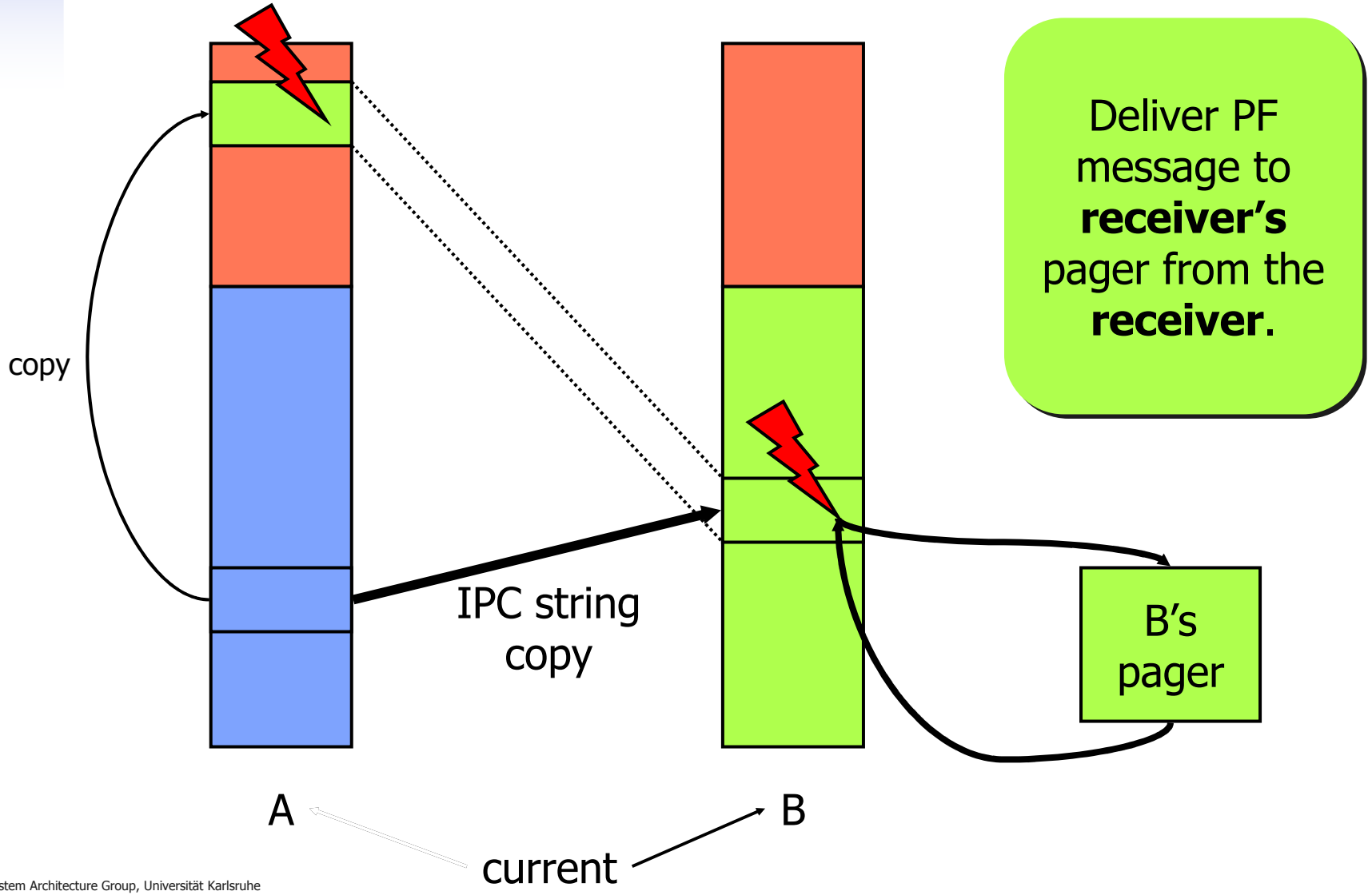
String Copy: Sender Page Fault



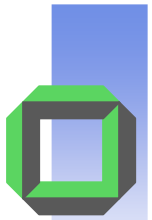
Deliver PF message to **sender's** pager from the **sender.**

© 2009 System Architecture Group, Universität Karlsruhe

String Copy: Receiver Page Fault

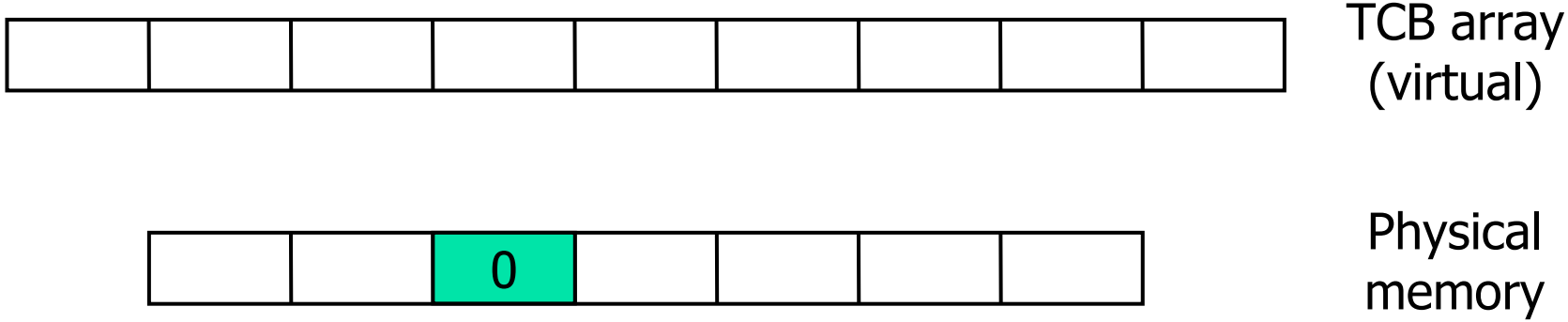


© 2009 System Architecture Group, Universität Karlsruhe



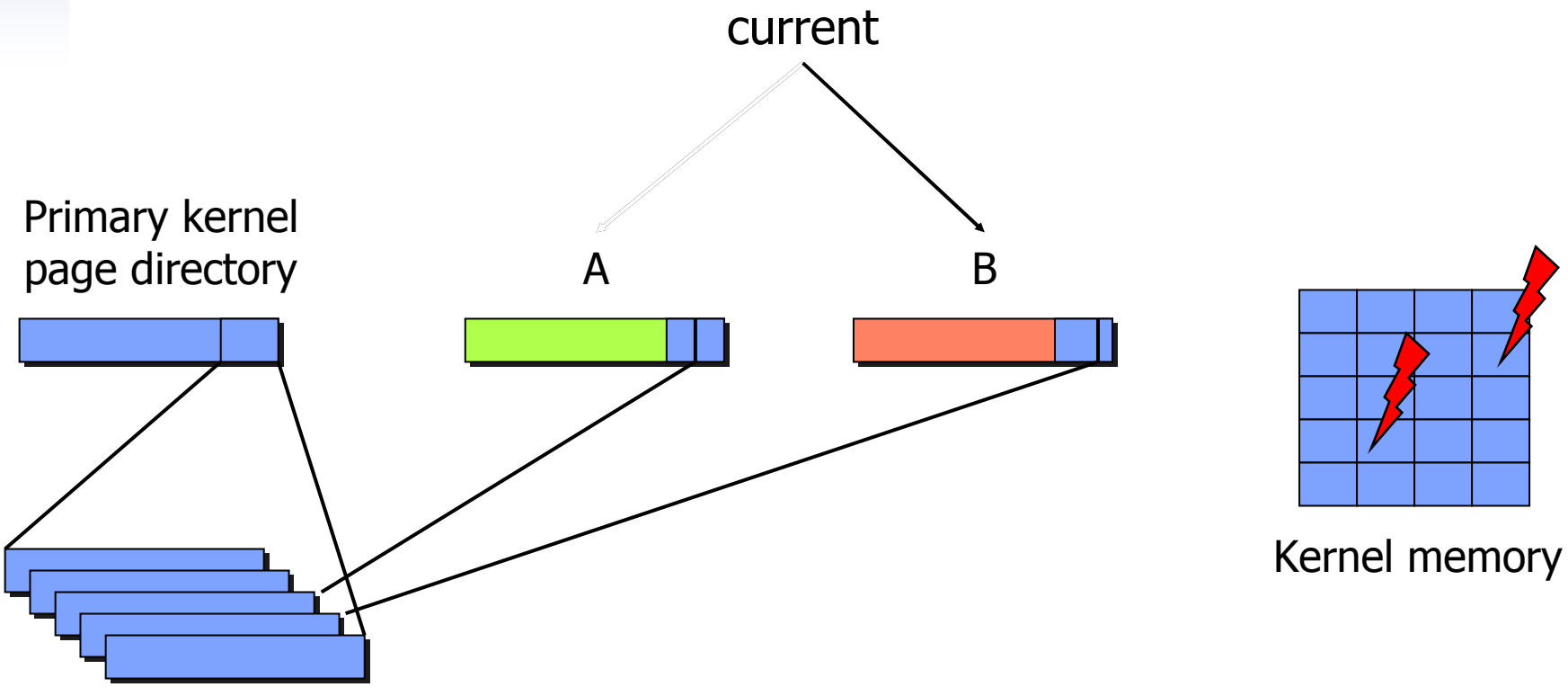
Invisible Page Faults

- Kernel handles some page faults internally
  - Virtual TCB array – map on demand
    - Exclusive empty page on write access
    - Shared 0-filled page read-only on read access
      - Avoid DoS attack on memory used for TCBs
      - Map exclusive empty page on later write



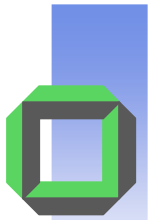
© 2009 System Architecture Group, Universität Karlsruhe

Lazy Kernel Space Building



Note: The kernel shares page **tables**, not page **directories**;
implemented by copying page directory entries.

© 2009 System Architecture Group, Universität Karlsruhe

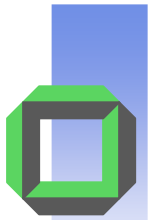


L4 Kernel Paradigm

Everything the kernel needs to handle in a secure manner will either become invisible or be hidden behind an abstraction.

- Exceptions not handled by the kernel itself will be posted to user land
  - Page faults
  - **Hardware interrupts**
  - Other exceptions

© 2009 System Architecture Group, Universität Karlsruhe

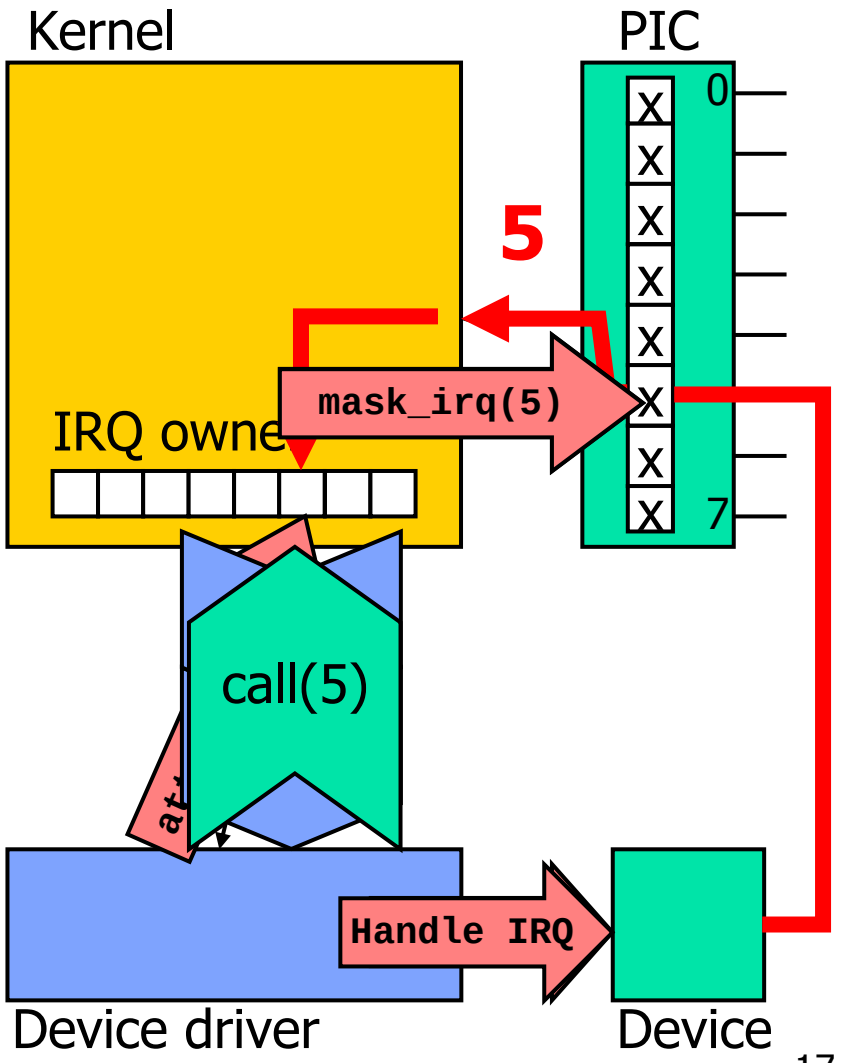


Hardware Interrupts

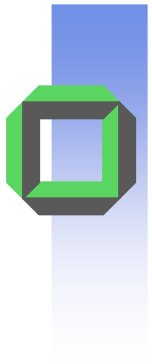
- Kernel hides first-level interrupt logic
  - No user messing with interrupt hardware
  - Deliver interrupts via IPC
  - More portable software
- Kernel interrupt handler
  - Translates interrupt into IPC
  - Sender: interrupt thread ID
    - Represents interrupt request line
  - Receiver: attached thread (user interrupt handler)
  - Message contents: Protocol ID
- Message destination
  - A thread needs to “attach to an interrupt”

© 2009 System Architecture Group, Universität Karlsruhe

Interrupt Handling



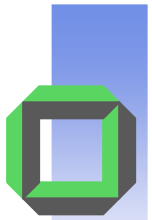
© 2009 System Architecture Group, Universität Karlsruhe



Hardware Interrupts

- Kernel uses some interrupts for itself
  - Timer tick – triggers scheduler
    - Timer device and interrupt line not available to user
  - Inter-processor interrupts (SMP)
    - Kernel hides IPI hardware
    - Cross-CPU user communication via IPC
- Kernel debugger may use interrupts
  - Performance counters – profiling
  - NMI – last resort debug aid

© 2009 System Architecture Group, Universität Karlsruhe

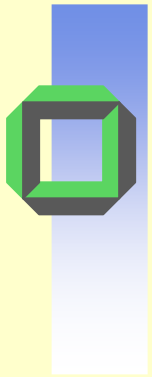


L4 Kernel Paradigm

Everything the kernel needs to handle in a secure manner will either become invisible or be hidden behind an abstraction.

- Exceptions not handled by the kernel itself will be posted to user land
  - Page faults
  - Hardware interrupts
  - **Other exceptions** } CPU L4

© 2009 System Architecture Group, Universität Karlsruhe

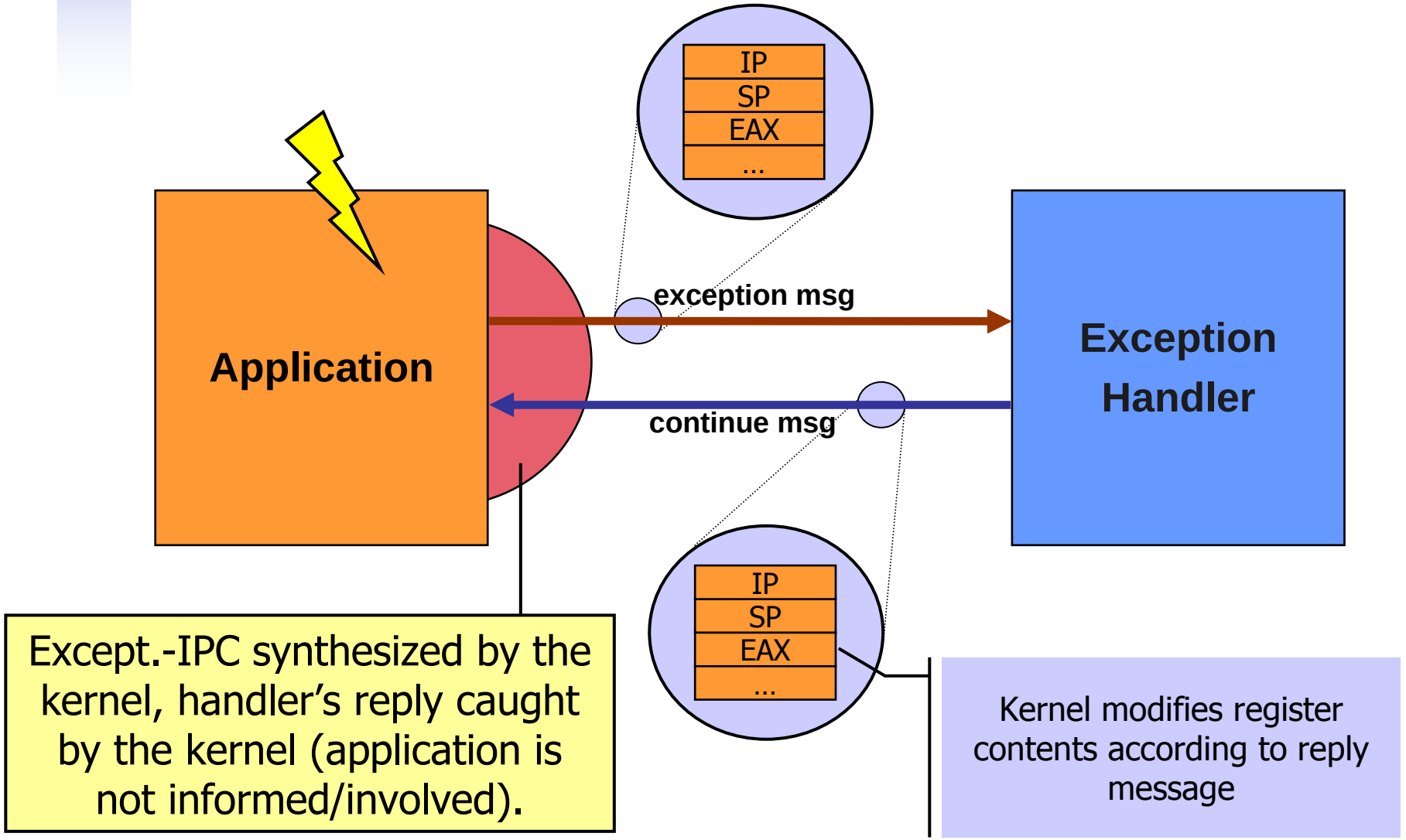


Old Exception Handling Model

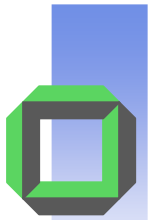
- Model
  - Create exception frame on user stack
  - Restart thread at a predefined exception handler
  - Return from exception handler using special instruction
- Problems
  - Very x86-ish, inconsistent
  - Requires a valid user stack
  - Poor performance for virtualization (too many kernel entries)
  - Recursive exception handling?
  - Safety?

© 2009 System Architecture Group, Universität Karlsruhe

New Exception Handling Model



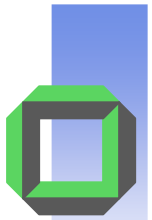
© 2009 System Architecture Group, Universität Karlsruhe



Invisible Exceptions

- Kernel handles some exceptions internally
  - Coprocessor/FPU virtualization
  - Transparent small space extension
  - TLB misses in software-loaded TLBs
- Kernel debugger handles some exceptions
  - Breakpoints
  - Single-stepping
  - Branch tracing

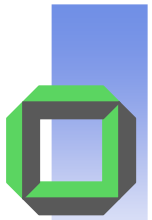
© 2009 System Architecture Group, Universität Karlsruhe



Review: Processor Multiplexing

- Hardware model
  - One thread
  - One address space
  - Exclusive access to resources (such as FPU)
- Microkernel exports
  - Multiple threads
  - Multiple address spaces
  - Maintain threads' view of the world
    - Threads have exclusive access to resources
- Multiplex abstractions onto existing hardware
  - Switch register file contents at thread switch
  - Potentially switch MMU state at thread switch
  - Switch FPU content etc. at thread switch

© 2009 System Architecture Group, Universität Karlsruhe



FPU Virtualization

- Strict switching

```
Thread switch:
    Store current thread's FPU state
    Load new thread's FPU state
```

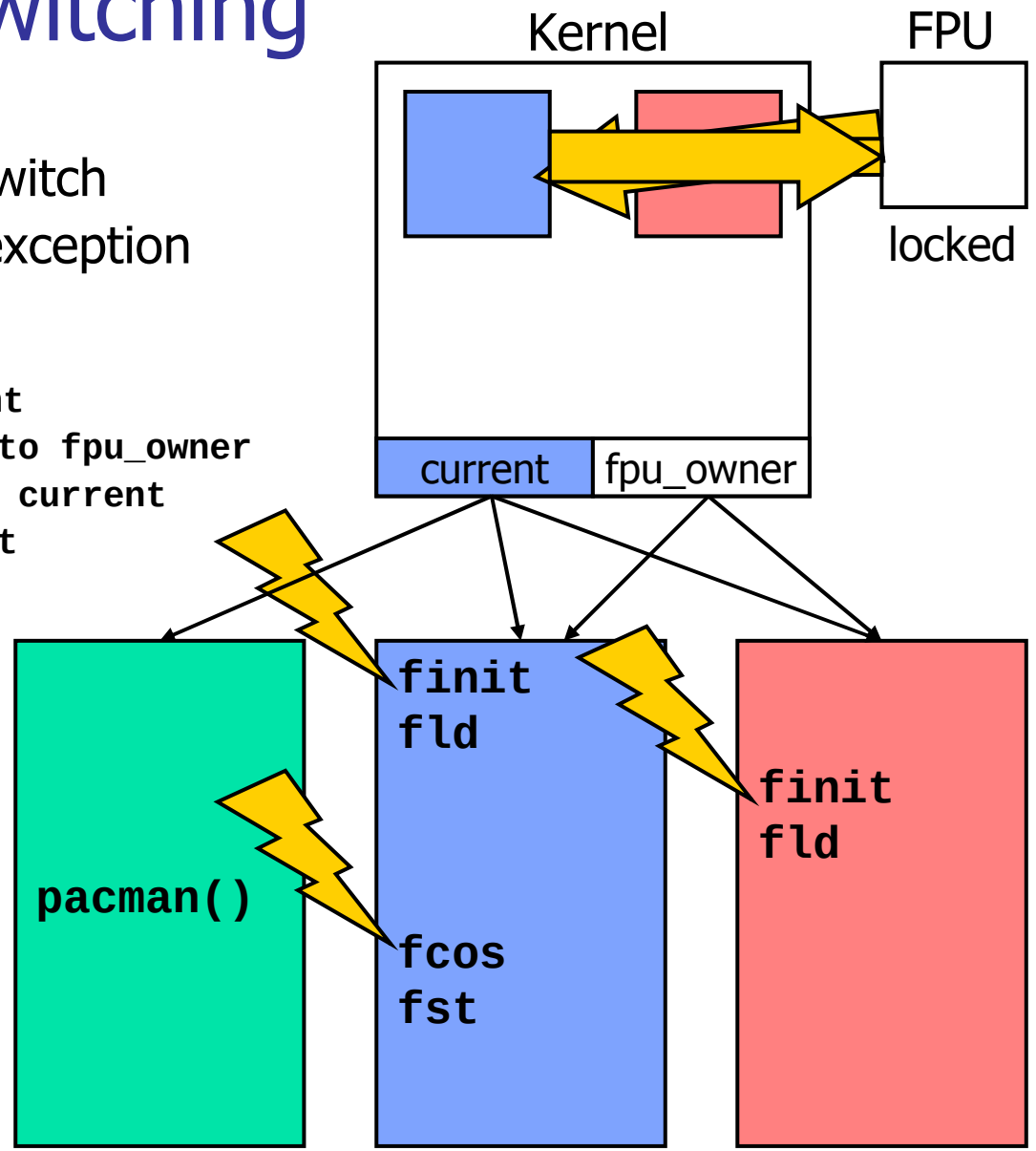
  - Extremely expensive
    - IA-32's full SSE2 state is 512 Bytes
    - IA-64's floating point state is ~1.5KB
  - May not even be required
    - Threads do not always use FPU

© 2009 System Architecture Group, Universität Karlsruhe

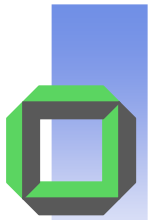
Lazy FPU Switching

- Lock FPU on thread switch
- Unlock at first use – exception handled by kernel

```
Unlock FPU
If fpu_owner != current
   Save current state to fpu_owner
   Load new state from current
   fpu_owner := current
```

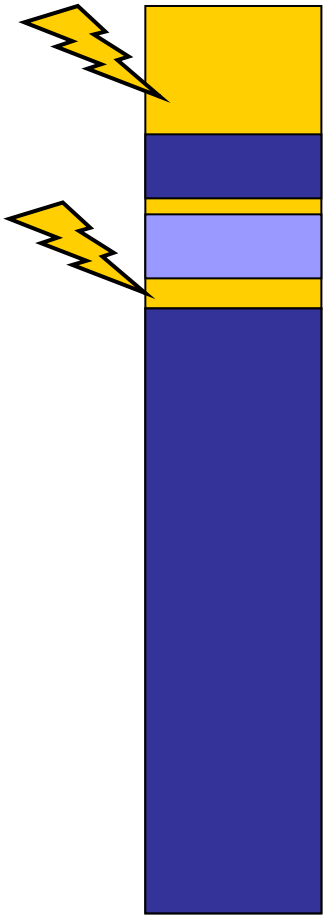


© 2009 System Architecture Group, Universität Karlsruhe



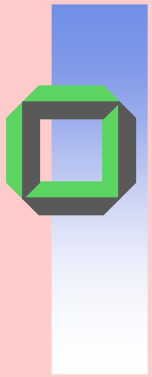
Small Spaces

- Access beyond small segment limit
  - Exception #GP or #SF
  - Kernel enlarges small space
  - Thread continues
- Next segment violation will be fatal



© 2009 System Architecture Group, Universität Karlsruhe

– Skipped in 2009 –

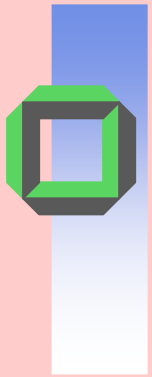


Privileged IA-32 Instructions

- Privileged instructions
  - **`lidt`** – Load interrupt descriptor table
  - **`rdmsr`**, **`wrmsr`** – Access model-specific registers
  - **`wbinvd`** – Write back and invalidate caches
  - **`lgdt`**, **`lldt`**, **`ltr`**, …
- IOPL-sensitive
  - **`cli`**/**`sti`** – Disable/enable interrupts
  - **`in`**, **`out`**, **`ins`**, **`outs`** – Access I/O address space

© 2009 System Architecture Group, Universität Karlsruhe

– Skipped in 2009 –

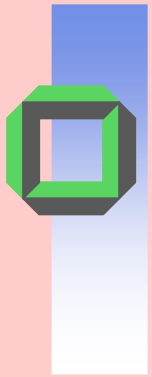


LIDT Emulation (Old Exc. Model)

- Threads use **`lidt`** instruction to install their own exception handlers
  - Privileged instruction – exception #GP
  - Analyze faulting instruction
  - Kernel emulates **`lidt`** – stores IDT pointer
  - Thread continues after **`lidt`** instruction
- Per-thread IDT pointer
- Unhandled exceptions are routed via IDT
  - Create exception frame on user stack
  - Continue thread on address in IDT
  - Thread returns using **`iret`** instruction

© 2009 System Architecture Group, Universität Karlsruhe

– Skipped in 2009 –

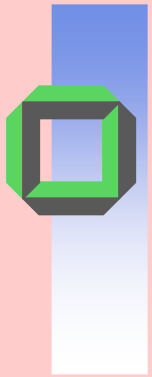


I/O Privilege Levels

- IOPL lives in EFLAGS register
  - But user mode cannot modify IOPL
- IOPL sensitive instructions
  - CPL=0
    - Kernel mode, no restrictions
  - CPL=3, IOPL = 3
    - User mode, almost no restrictions
  - CPL=3, IOPL < 3
    - **`cli`**/**`sti`** cause #GP
    - **`in`**/**`out`** controlled by I/O permission bitmap

© 2009 System Architecture Group, Universität Karlsruhe

– Skipped in 2009 –

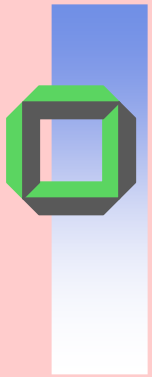


I/O Port Access Control

- I/O permission bitmap (IOPBM)
  - One bit per I/O port (64 kbit)
    - 0 – Access allowed
    - 1 – Exception #GP
  - Part of IA-32 task state segment
  - Task-local
    - Switch IOPBM with page tables
- Kernel translates #GP into IPC
  - Analyze faulting instruction
  - I/O port fault message sent to pager
    - Port address, access size
  - Pager maps I/O fpage
  - Kernel resets bits in IOPBM
- Use existing mechanisms – map and unmap

© 2009 System Architecture Group, Universität Karlsruhe

– Skipped in 2009 –

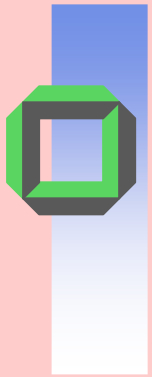


Virtualizing the Interrupt Flag

- Interrupt enable flag in EFLAGS
  - User cannot modify IF
- **`cli`**/**`sti`** cause exception #GP
  - Analyze faulting instruction
  - Flip user's IF
  - Per-thread IF
- But … expensive
  - Unusable for implementing critical sections

© 2009 System Architecture Group, Universität Karlsruhe

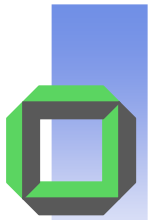
– Skipped in 2009 –



Protected Mode Virtual Interrupts

- Hardware support
  - Allows enforcing maximum interrupt latency
  - Two new flags in EFLAGS register (VIF, VIP)
- **`cli`**/**`sti`** in user mode updates Virtual IF
  - Less costly – no exception
- Hardware interrupts still subject to real IF
  - Deliver interrupts immediately or
  - Postpone delivery
- Kernel can set VIP flag
  - Indicates pending interrupt
  - Next **`sti`** will cause #GP
  - Kernel can deliver pending interrupts

© 2009 System Architecture Group, Universität Karlsruhe



Delayed Preemptions

- Thread can ask for extension of time slice
- Kernel *can* delay preemption
  - Unless thread with higher prio wakes up
  - Up to a maximum delay
    - Set by thread's scheduler
  - Preemption-pending bit in TCR

- If kernel preempts thread …
  - Notification IPC to exception handler

© 2009 System Architecture Group, Universität Karlsruhe