Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Improvement of IPC responsiveness in microkernel-based operating systems

Steffen Reichmann - WAMOS 2015

Agenda

- Introduction
- IPC improvements
  - Adding combined IPC systemcalls
  - Adding asynchronous IPC
  - Zero copy & virtual message registers
  - Abandoning of long IPC
  - Abandoning of IPC timeouts
- Conclusion

Introduction

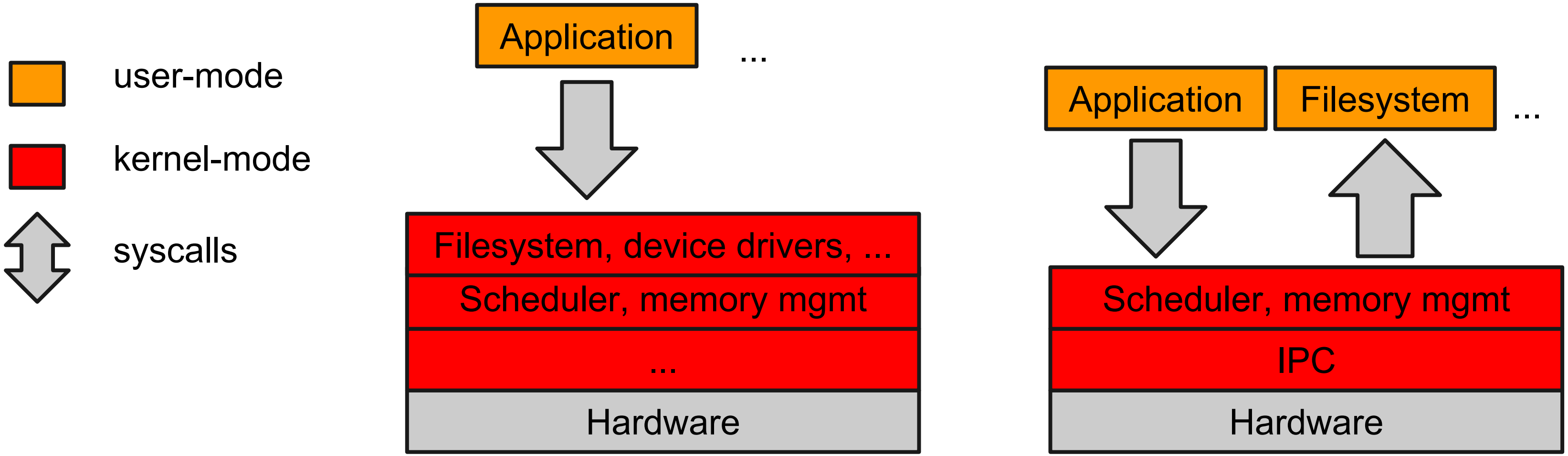
- What is a microkernel?

> *“Traditionally, the word 'kernel' is used to denote the part of the operating system that is mandatory and common to all other software. The basic idea of the -kernel approach is to minimize this part, i.e. to implement outside the kernel whatever possible.”* - J. Liedtke

Introduction

Monolothic kernel

Microkernel



🟧 user-mode

🟥 kernel-mode

⇕ syscalls

Application

...

Filesystem, device drivers, ...

Scheduler, memory mgmt

...

Hardware

Application

Filesystem

...

Scheduler, memory mgmt

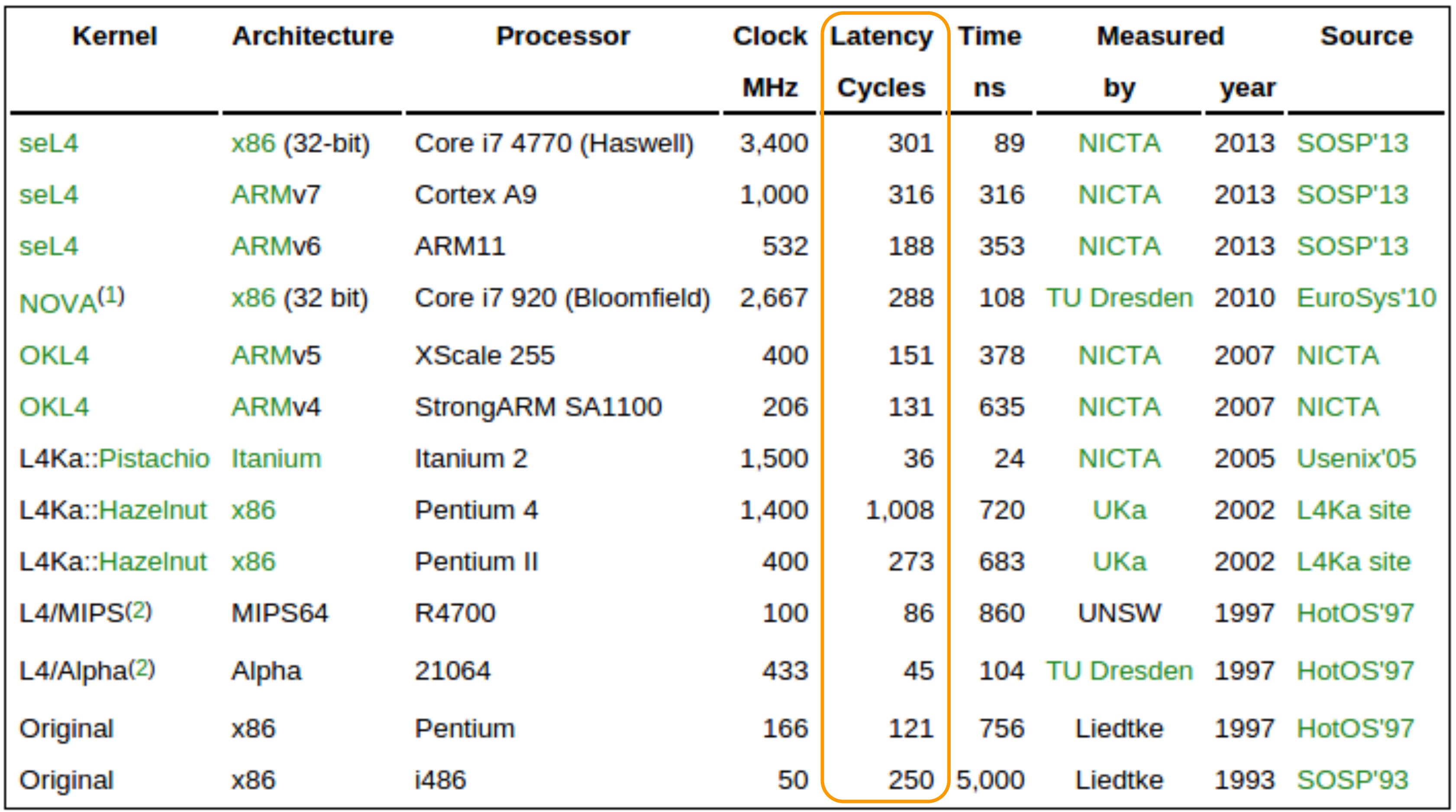
IPC

Hardware

IPC improvements

- Measurement conditions:
  - Hot cache (tight loop)
  - One way (half of round-trip)
  - Inter address space (requiring a full context switch)

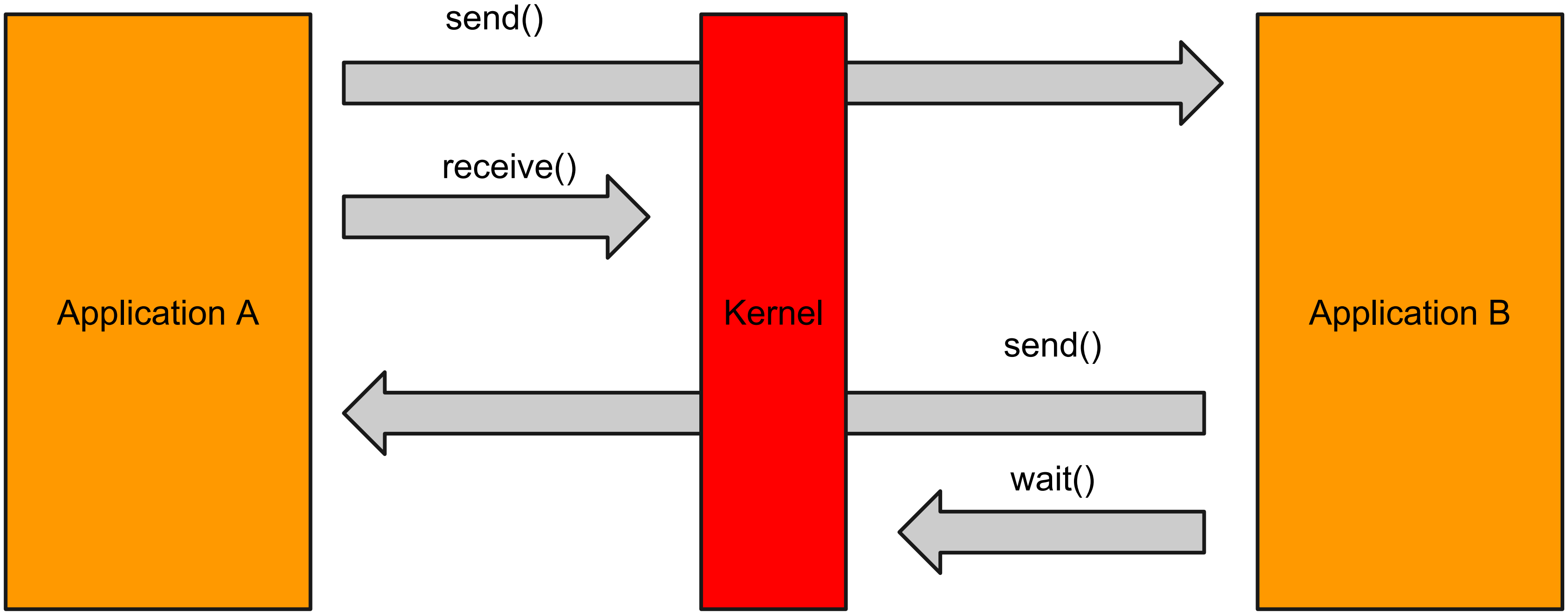
- Notes:
  - MIPS and Alpha were incomplete, add ~100 cycles



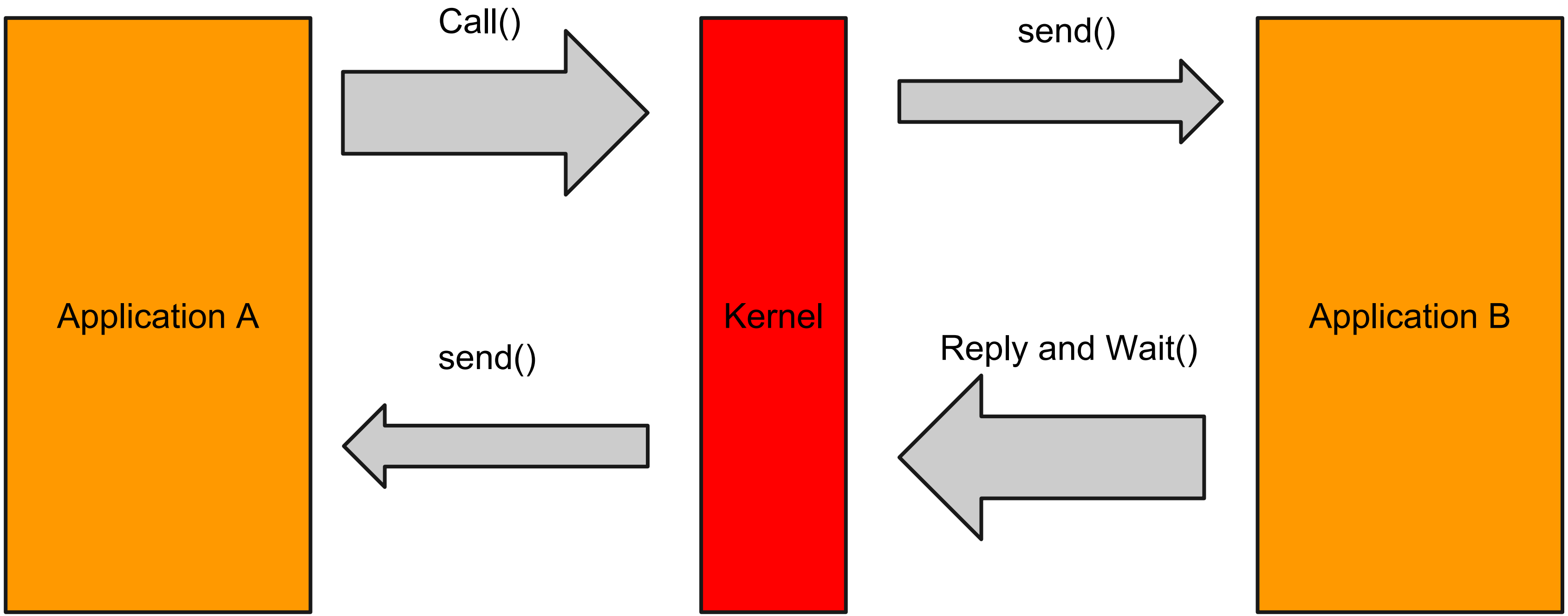
Kernel	Architecture	Processor	Clock	Latency	Time	Measured		Source
			MHz	Cycles	ns	by	year	
seL4	x86 (32-bit)	Core i7 4770 (Haswell)	3,400	301	89	NICTA	2013	SOSP'13
seL4	ARMv7	Cortex A9	1,000	316	316	NICTA	2013	SOSP'13
seL4	ARMv6	ARM11	532	188	353	NICTA	2013	SOSP'13
NOVA[1]	x86 (32 bit)	Core i7 920 (Bloomfield)	2,667	288	108	TU Dresden	2010	EuroSys'10
OKL4	ARMv5	XScale 255	400	151	378	NICTA	2007	NICTA
OKL4	ARMv4	StrongARM SA1100	206	131	635	NICTA	2007	NICTA
L4Ka::Pistachio	Itanium	Itanium 2	1,500	36	24	NICTA	2005	Usenix'05
L4Ka::Hazelnut	x86	Pentium 4	1,400	1,008	720	UKa	2002	L4Ka site
L4Ka::Hazelnut	x86	Pentium II	400	273	683	UKa	2002	L4Ka site
L4/MIPS[2]	MIPS64	R4700	100	86	860	UNSW	1997	HotOS'97
L4/Alpha[2]	Alpha	21064	433	45	104	TU Dresden	1997	HotOS'97
Original	x86	Pentium	166	121	756	Liedtke	1997	HotOS'97
Original	x86	i486	50	250	5,000	Liedtke	1993	SOSP'93

Quelle: http://l4hq.org/docs/performance.php

Adding combined IPC calls



Adding combined IPC calls



Adding combined IPC calls

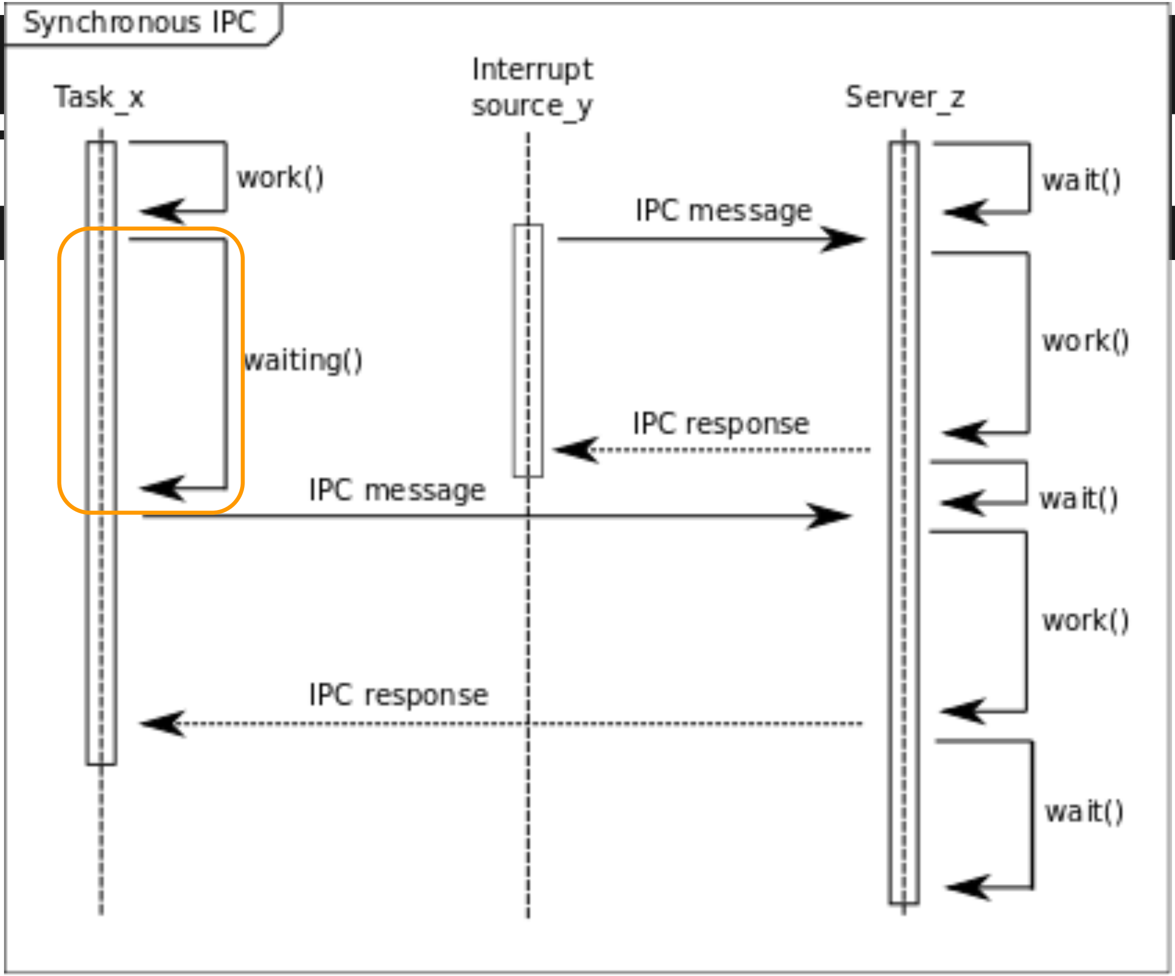
- Combining IPC saves two systemcalls
  - Send() & receive() ⇒ call()
  - Send() & wait() ⇒ reply and wait()
- Call() initiates instant context switch

Adding asynchronous IPC

Synchronous IPC only:

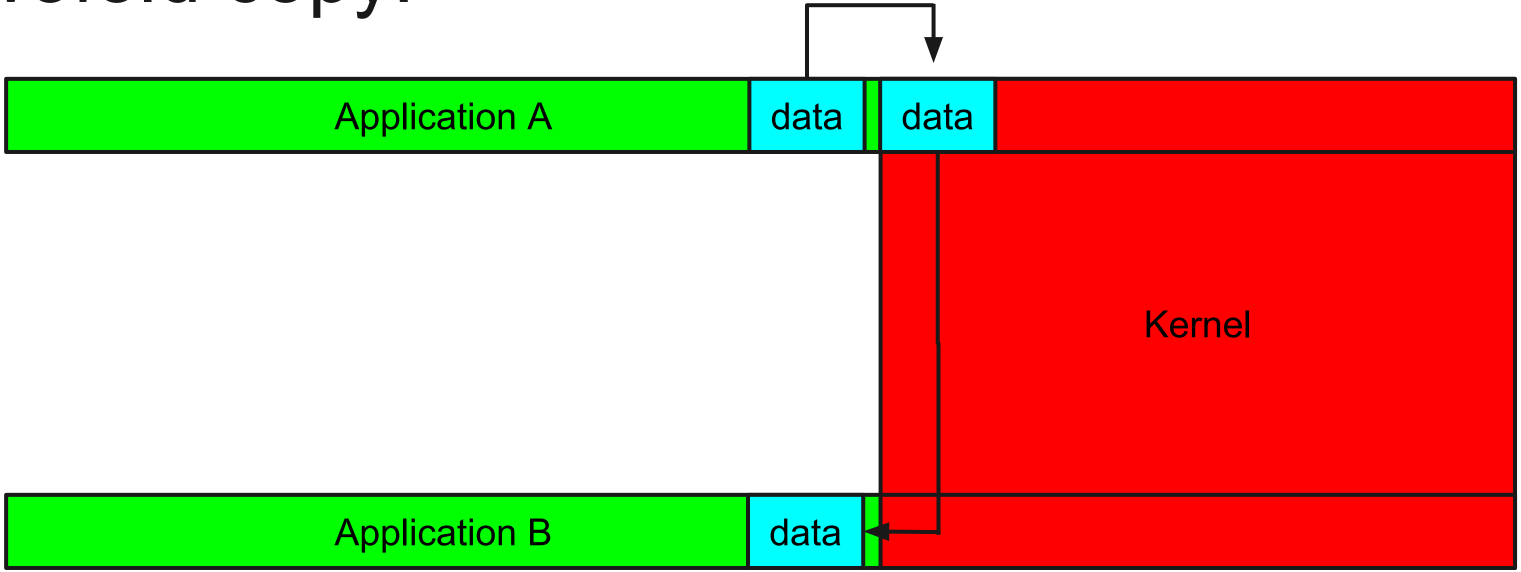
- Upside
  - No message buffering needed in the kernel
  - Only one way for IPC ⇒ “more pure”
- Downside
  - Multithreading needed ⇒ more complexity

Addi PC


Synchronous IPC
Task_x
Interrupt source_y
Server_z
work()
wait()
IPC message
waiting()
work()
IPC response
wait()
IPC message
work()
IPC response
wait()

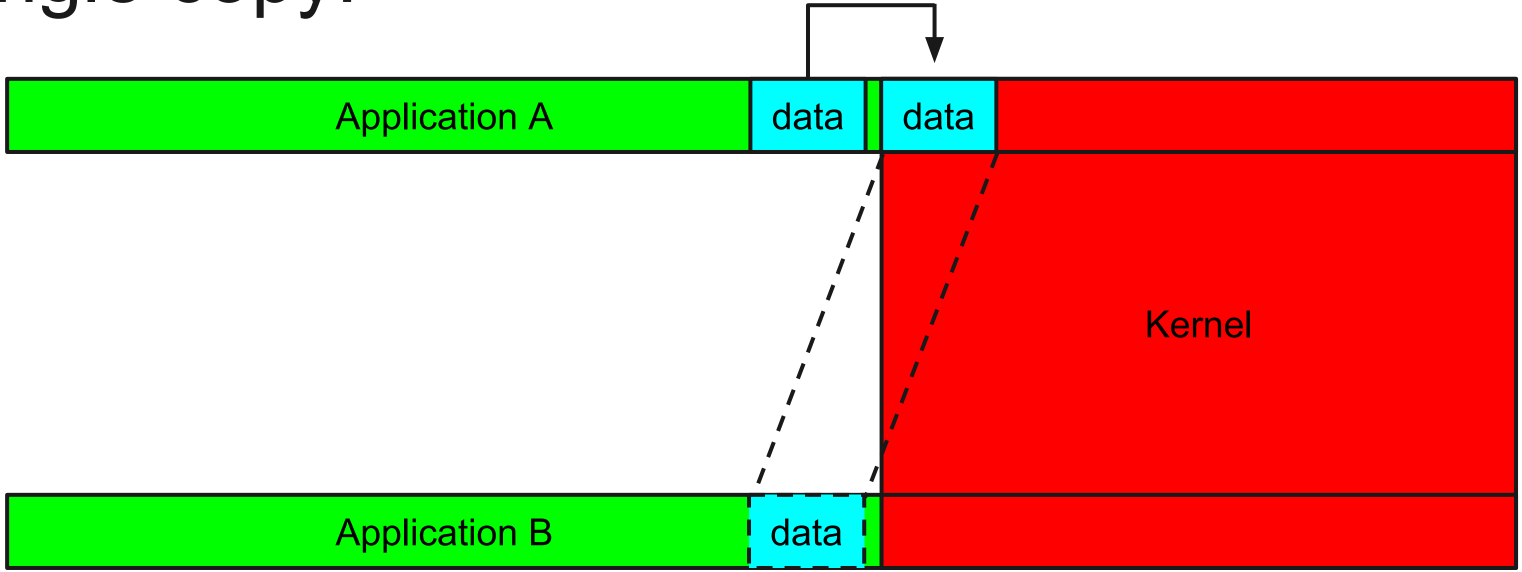
Zero copy

Twofold copy:



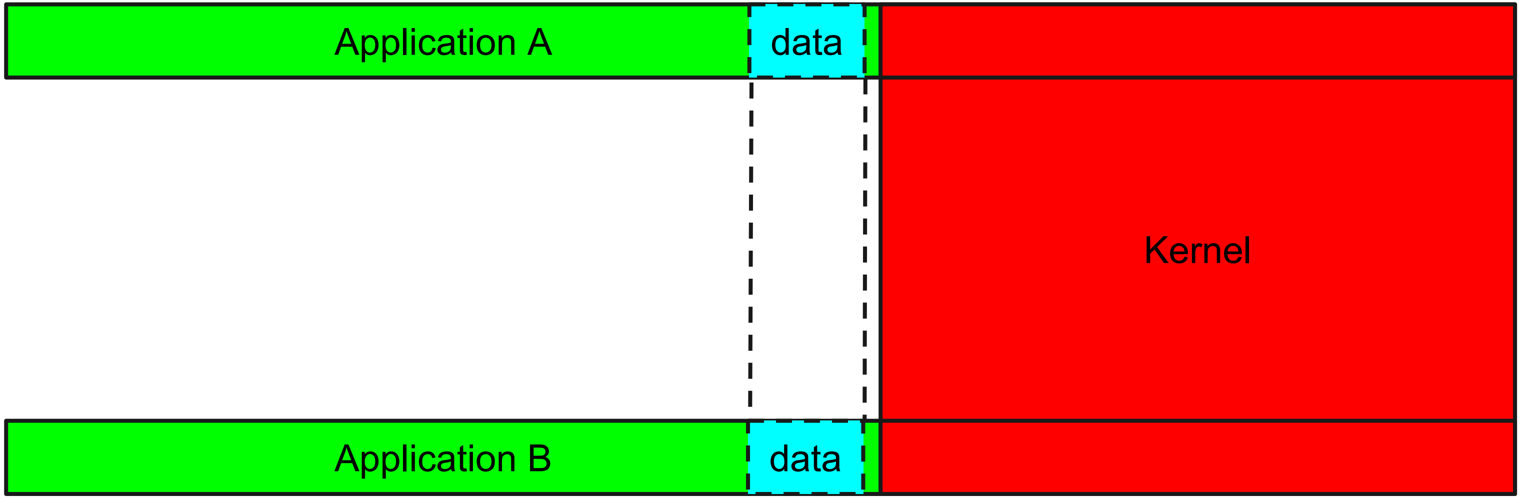
Zero copy

Single copy:



Zero copy

Zero copy:



Zero copy

Procedure:

1. Kernel initiates IPC from sender’s context
2. Message registers are not altered
3. On receiver’s context, the data is still there

⇒ Number and size of message registers vary

Virtual message registers

- Set of configurable virtual message registers
  - Some pinned to physical registers
  - Rest pinned to extra per-thread space

⇒ Better platform independency

Abandoning long IPC

Idea: multiple registers with one copy

- Problem: Can lead to pagefaults

- Solution:
  - Pagefault handling in user-space?
  - Better: Transfer data via shared buffer

⇒ Abandoning of long IPC

Abandoning IPC timeouts

- Timeouts prevent tasks from blocking forever
- Theoretically: Timeout configurable for every IPC
- Practically: Guessing of good timings is very hard
  ⇒ Mostly set to ∞
- Watchdogs showed to be more commonly used

Conclusion

- IPC improvements
  - Not one huge, but many small improvements
  - Better hardware
  - “From 5.000 ns to 89 ns”

- Future outlooks
  - Responsiveness gets faster and faster
  - Architectural advantage: Security

Sources

- From l3 to sel4 what have we learnt in 20 years of l4 microkernels?
    - *Elphinstone and G. Heiser*

- Improving ipc by kernel design
    - *J. Liedtke*

- On micro-kernel construction
    - *J. Liedtke*