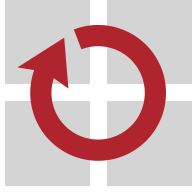
Übung zu Betriebssystemtechnik

Nachrichtenaustausch und Copy-on-Write

27. Juni 2019

Bernhard Heinloth, Andreas Ziegler,
Christian Eichler & Harald Böhm

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

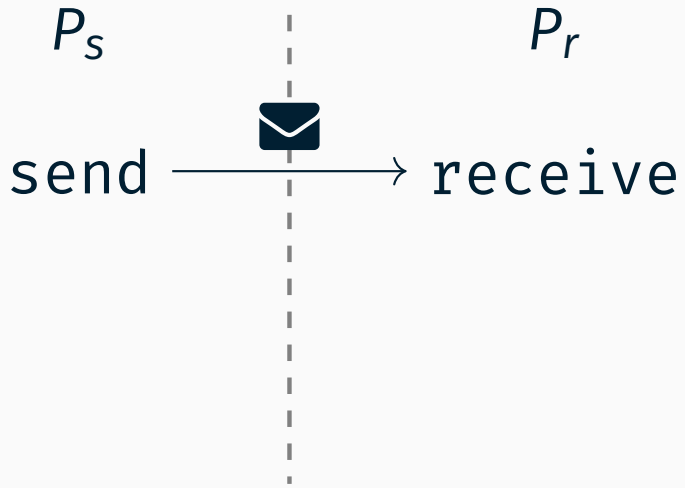
TECHNISCHE FAKULTÄT

Ziel: Getrennte Adressräume überwinden

$\Rightarrow$ Nachrichtenaustausch

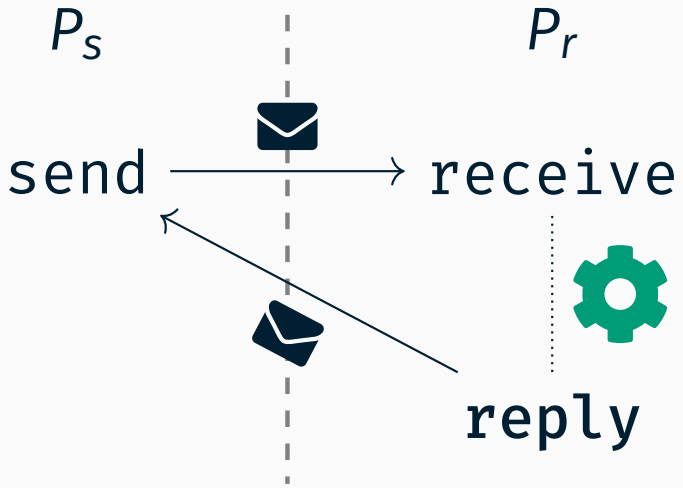
Ziel: Getrennte Adressräume überwinden

$\Rightarrow$ Nachrichtenaustausch:
`send, receive`



Ziel: Getrennte Adressräume überwinden

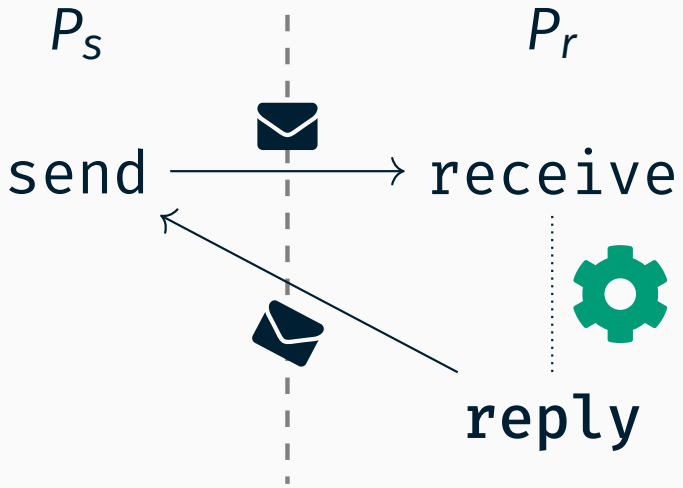
$\Rightarrow$ Nachrichtenaustausch:
`send, receive, reply`



Warum `reply`?

Ziel: Getrennte Adressräume überwinden

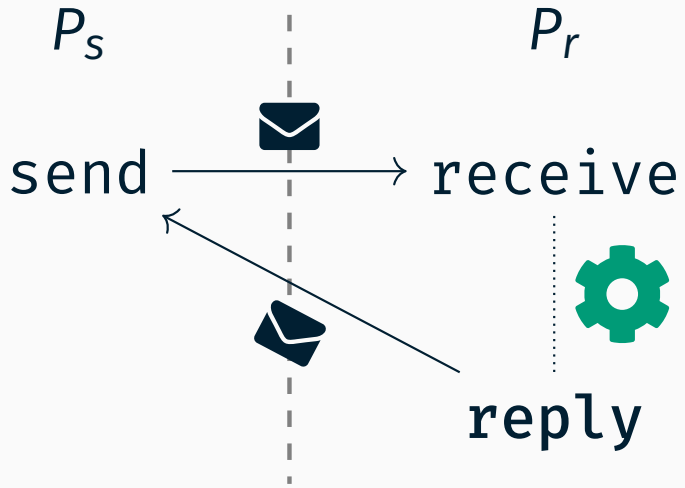
$\Rightarrow$ Nachrichtenaustausch:
`send, receive, reply`



Warum **`reply`**? $\Rightarrow$ Keine Timer nötig um DoS auf Server-Seite zu verhindern!

Ziel: Getrennte Adressräume überwinden

⇒ Nachrichtenaustausch:
`send, receive, reply`



Warum `reply`? ⇒ Keine Timer nötig um DoS auf Server-Seite zu verhindern!

Begriffsklärung: (a-)synchron / (nicht-)blockierend

synchron	Prozess schläft bis Senden fertig
asynchron	Prozess kehrt während Senden zurück
(nicht-)blockierend	Synchronisation der Operationen

Umsetzung

Ablauf `send`

`send(to, data, len, replybuf, replylen)`

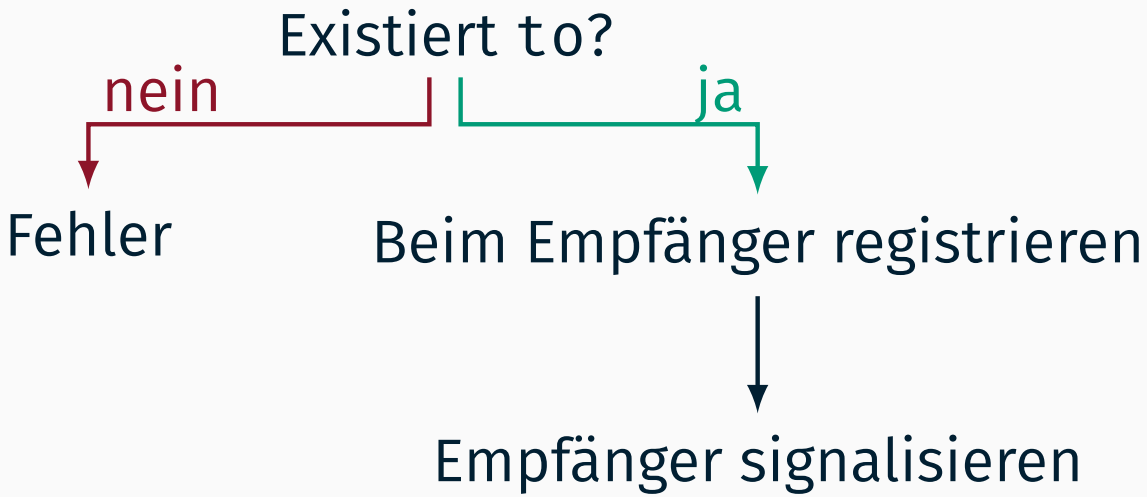
Existiert `to`?

Ablauf send

```
send(to, data, len, replybuf, replylen)
```

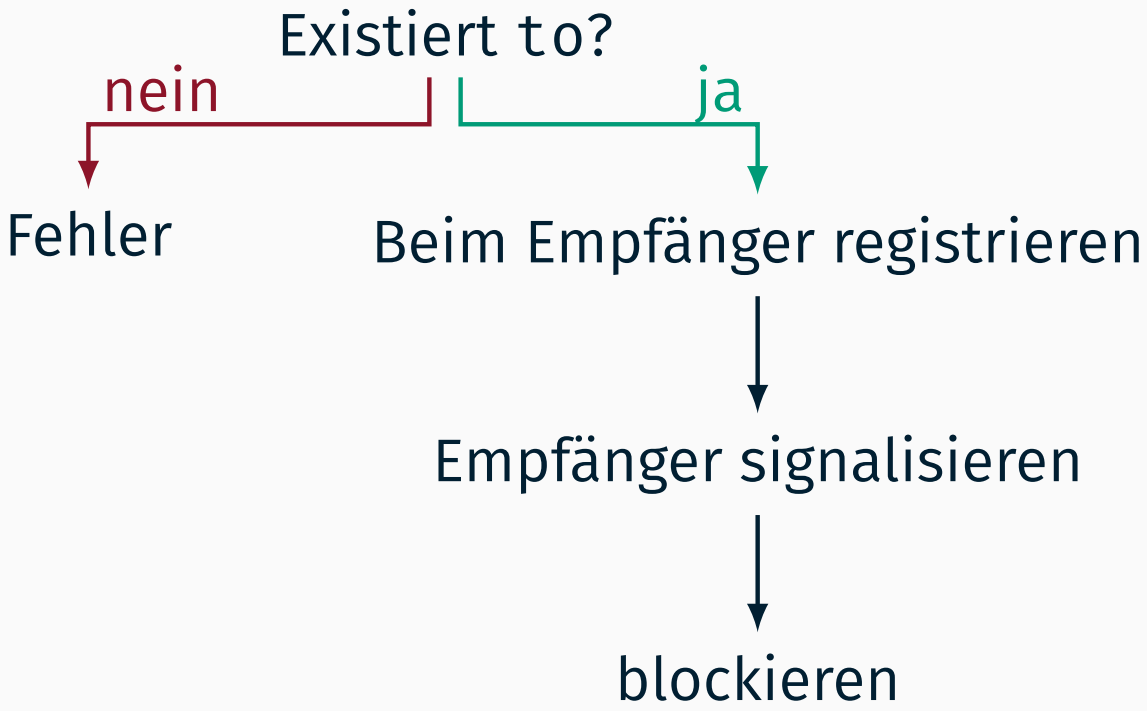
Ablauf send

```
send(to, data, len, replybuf, replylen)
```



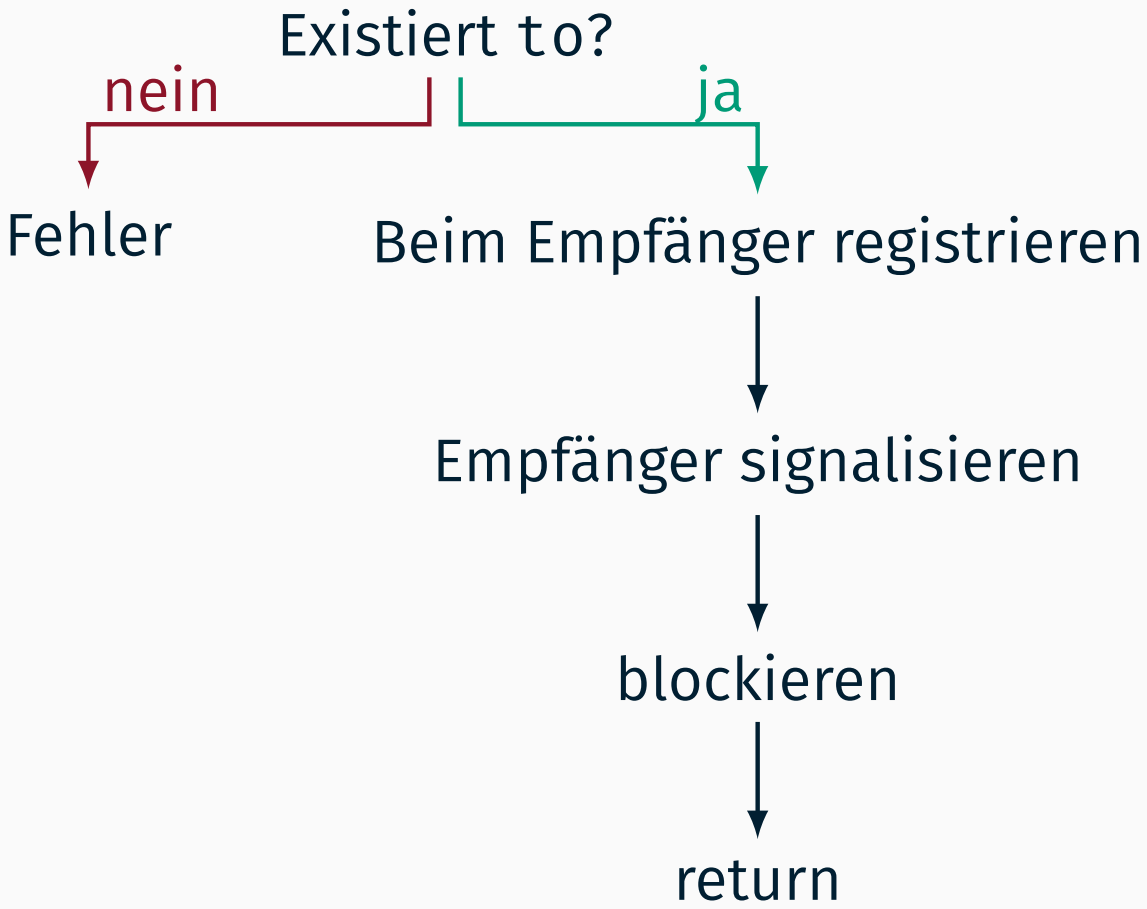
Ablauf send

```
send(to, data, len, replybuf, replylen)
```



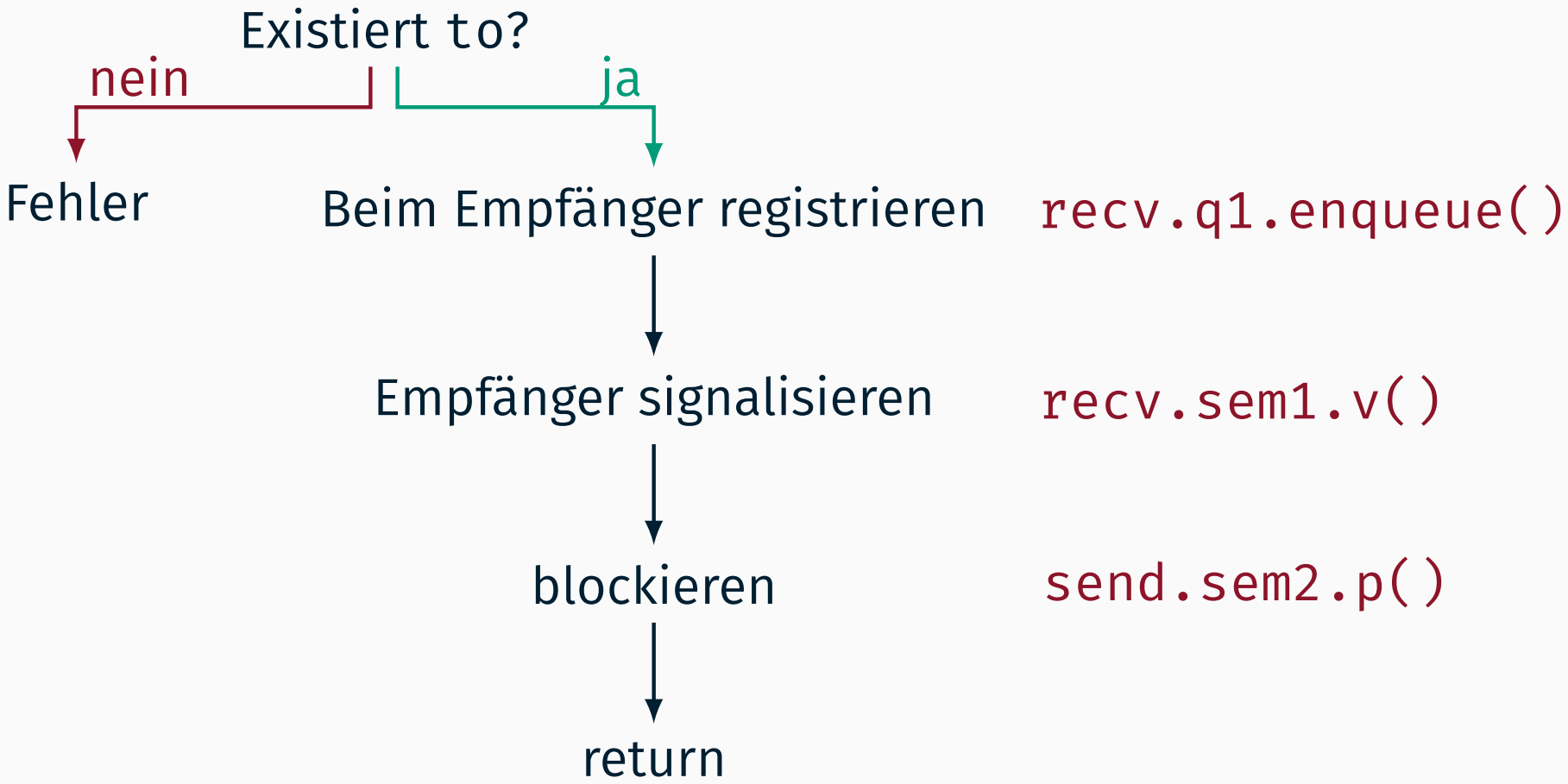
Ablauf send

```
send(to, data, len, replybuf, replylen)
```



Ablauf send

```
send(to, data, len, replybuf, replylen)
```



Ablauf `receive`

`recv(data, len)`

Auf Signal eines Senders warten

Ablauf `receive`

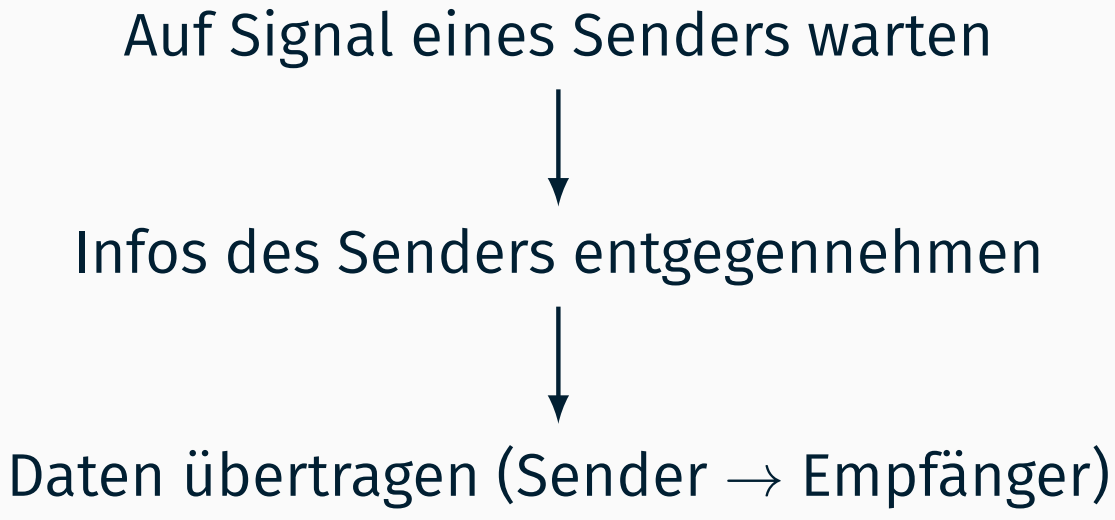
`recv(data, len)`

Auf Signal eines Senders warten

Infos des Senders entgegennehmen

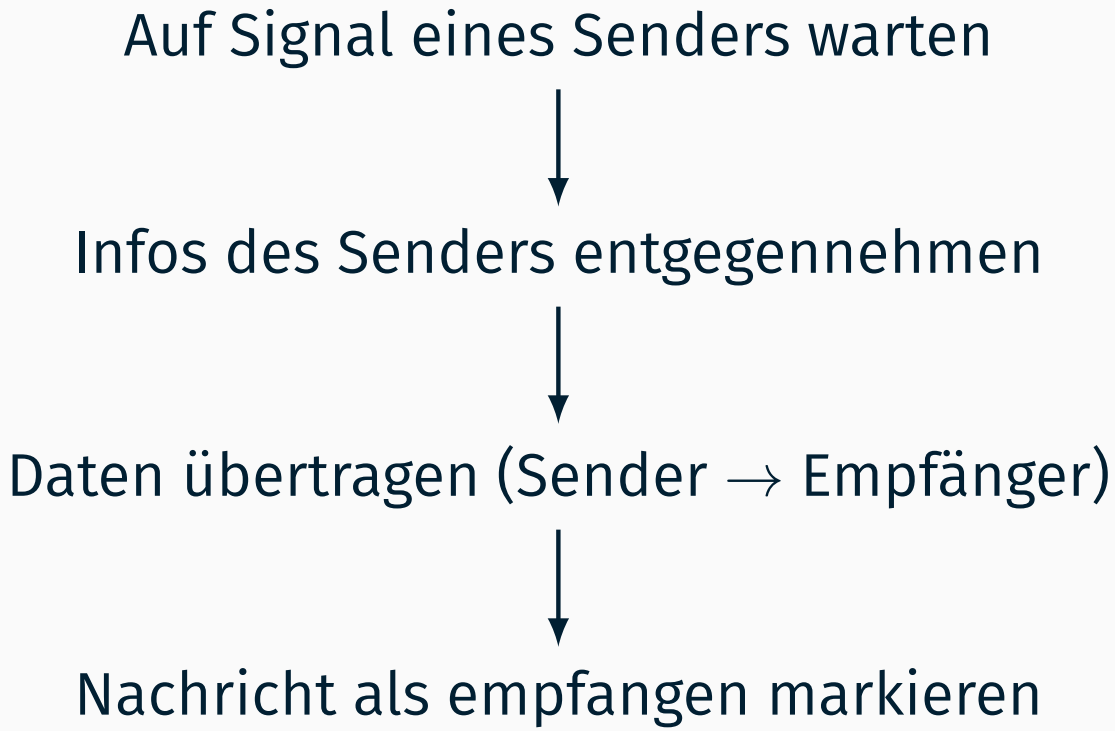
Ablauf `receive`

```
recv(data, len)
```



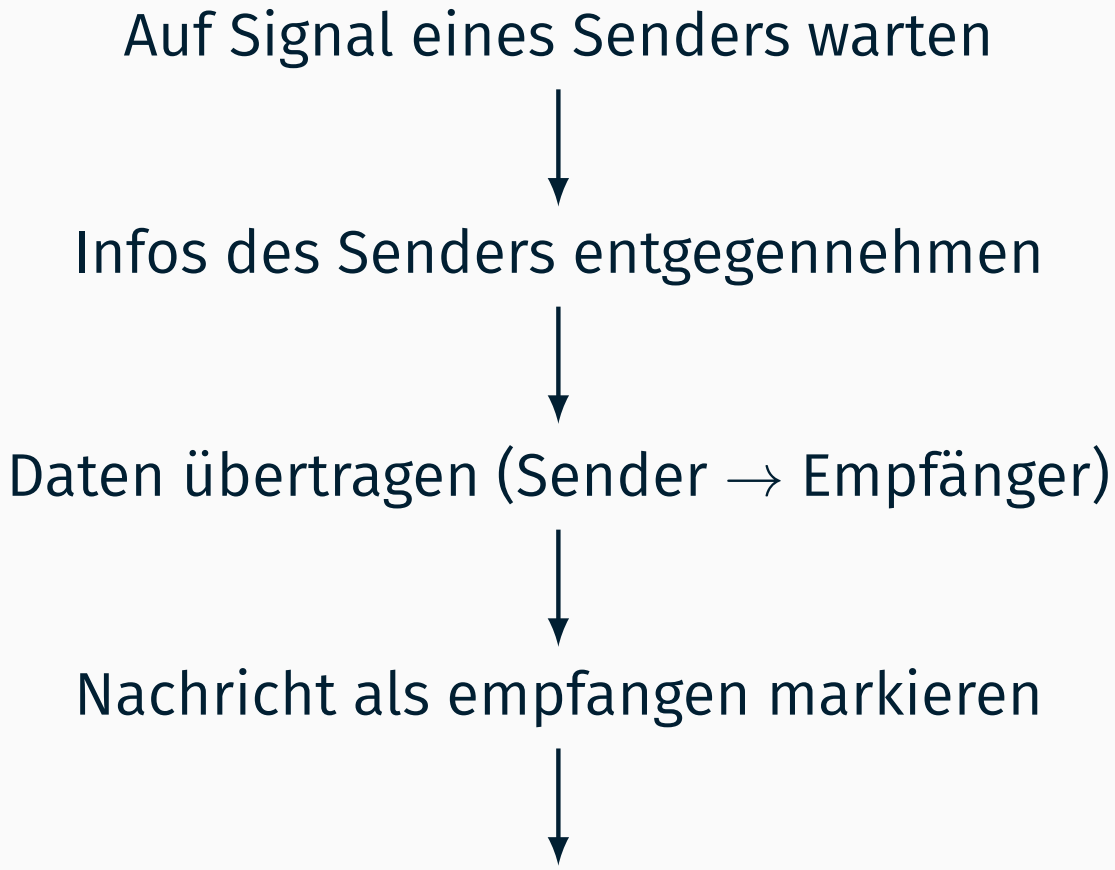
Ablauf `receive`

`recv(data, len)`



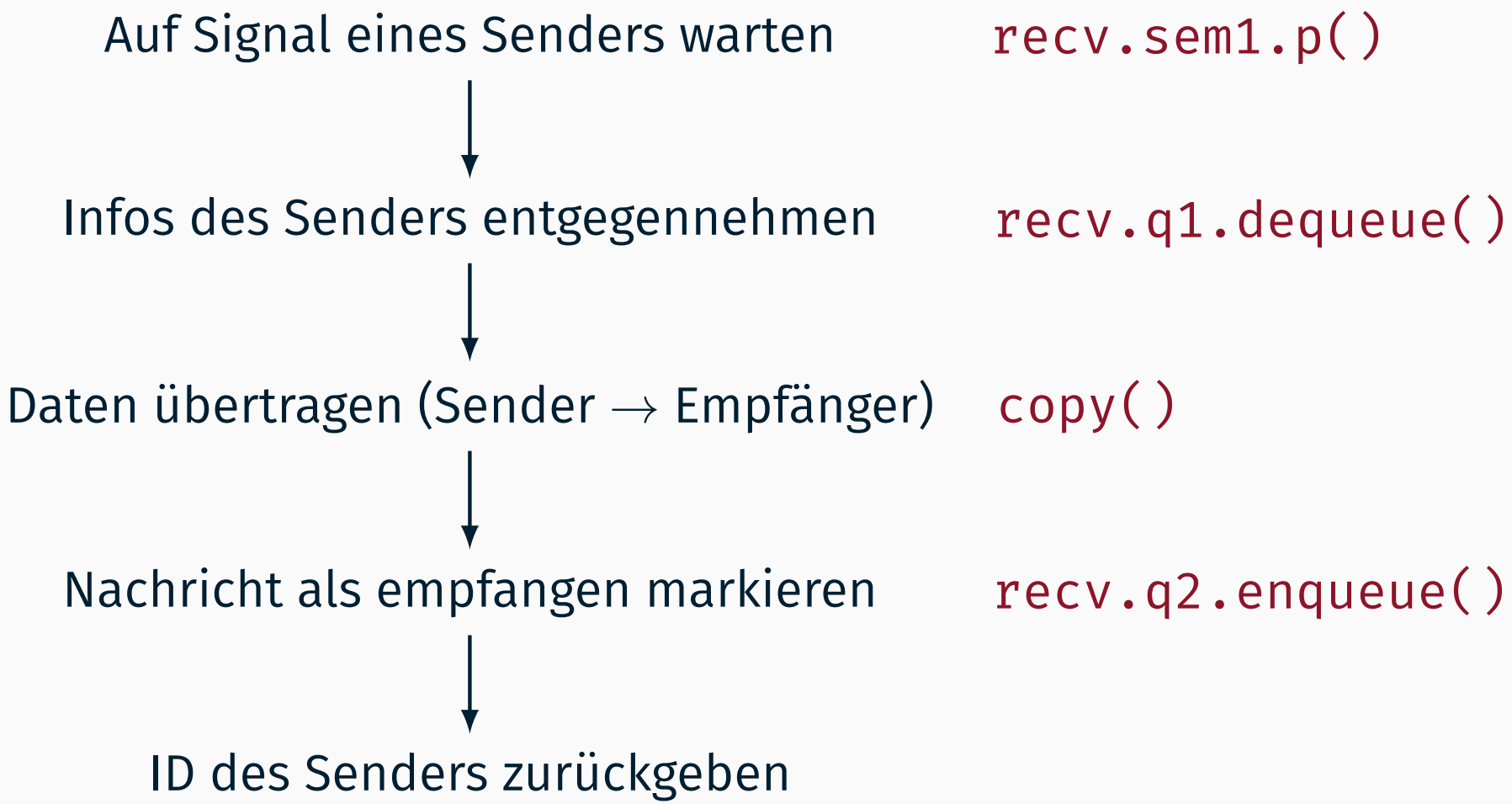
Ablauf `receive`

`recv(data, len)`



Ablauf `receive`

`recv(data, len)`



Ablauf `reply`

`reply(to, data, len)`

- Gab es ein `send` von `to`?

Ablauf `reply`

```
reply(to, data, len)
```

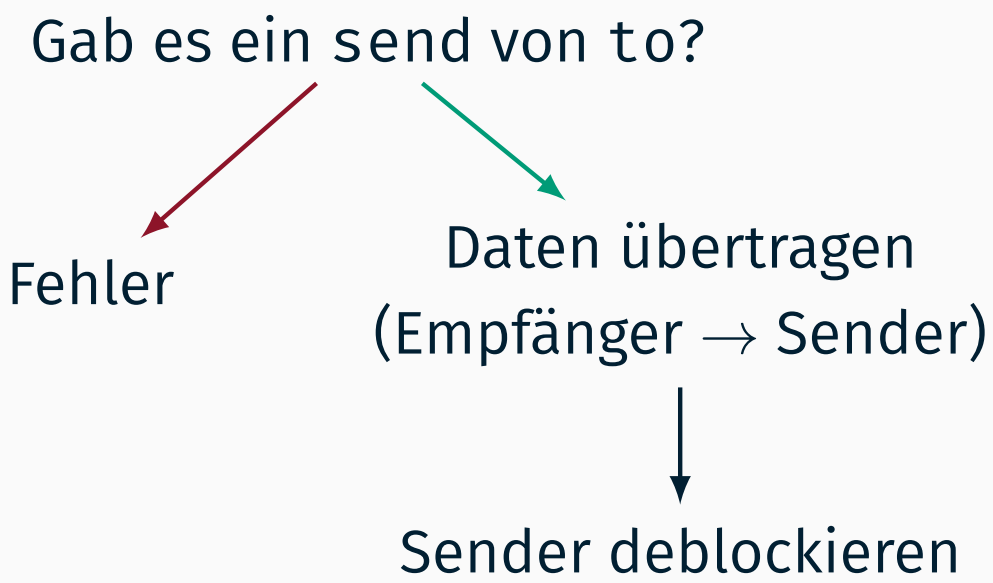
Gab es ein `send` von `to`?

Fehler

Daten übertragen
(Empfänger $\rightarrow$ Sender)

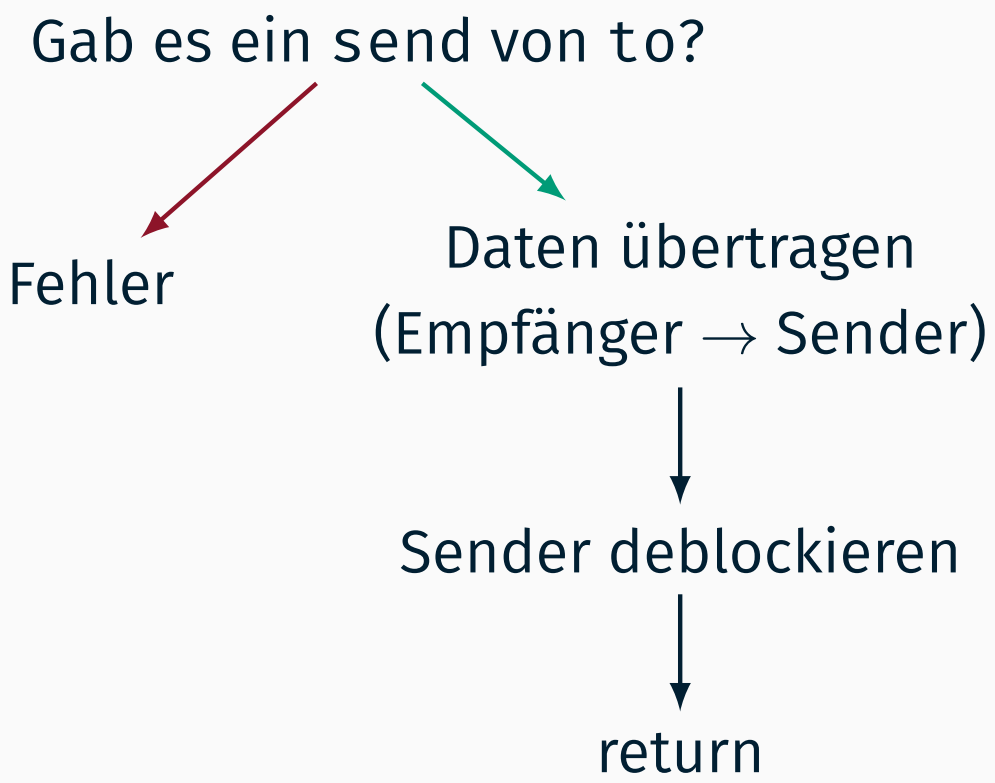
Ablauf `reply`

```
reply(to, data, len)
```



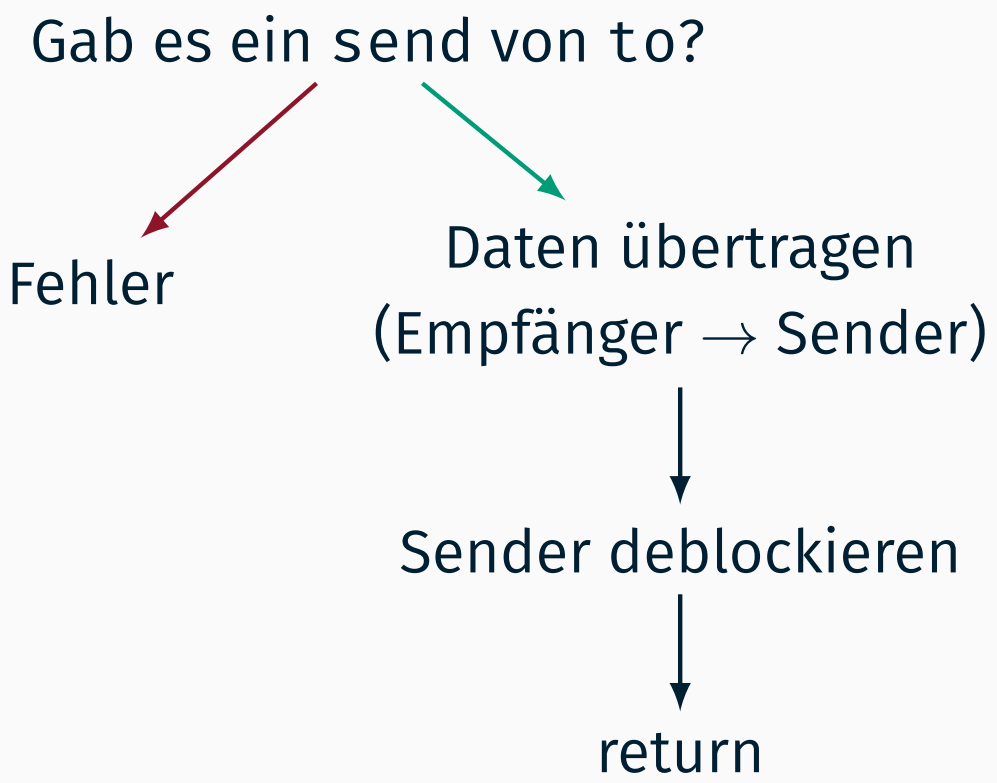
Ablauf `reply`

```
reply(to, data, len)
```



Ablauf `reply`

`reply(to, data, len)`



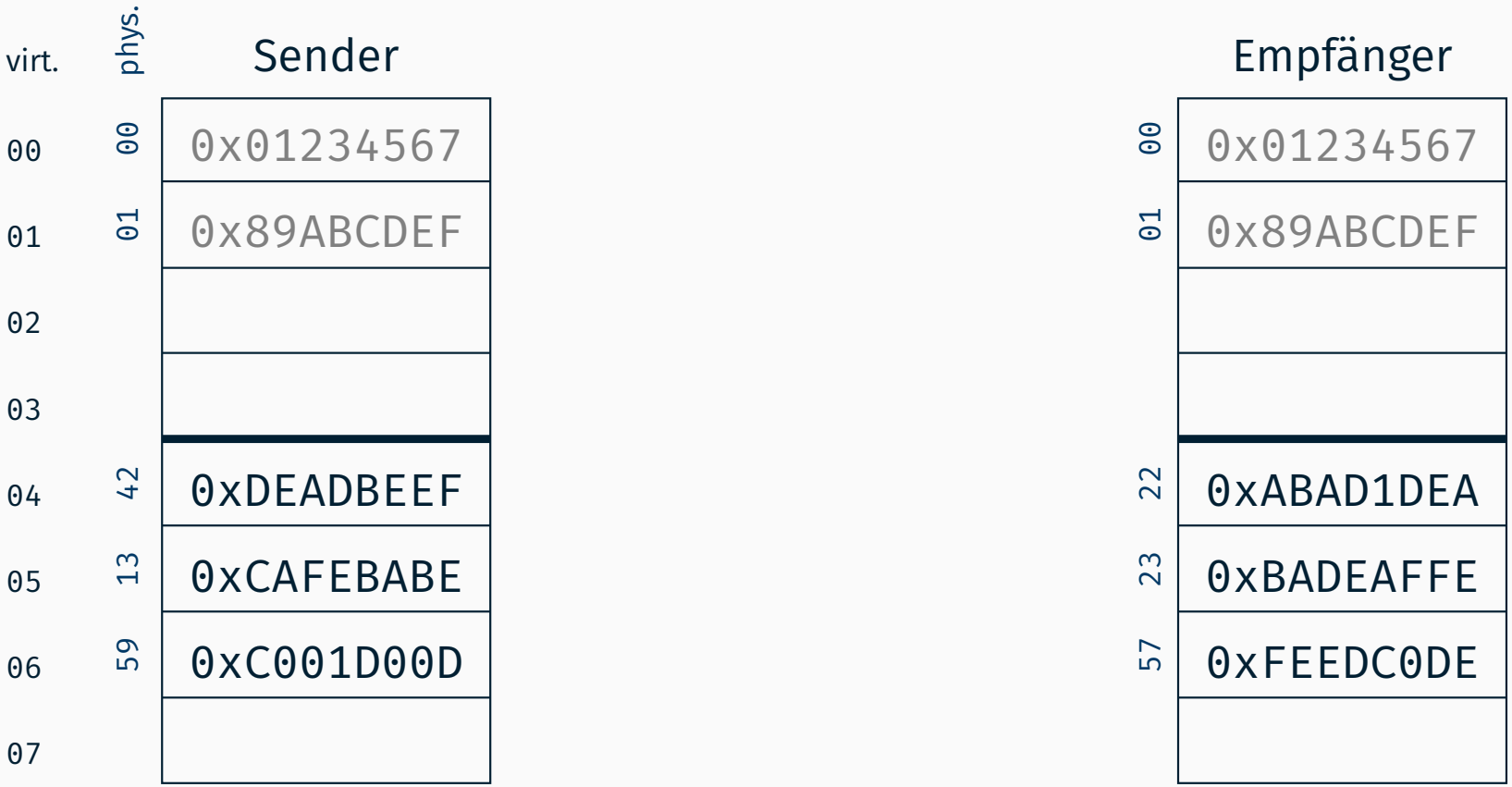
Nachricht mit ID in Queue 2?

`copy()`

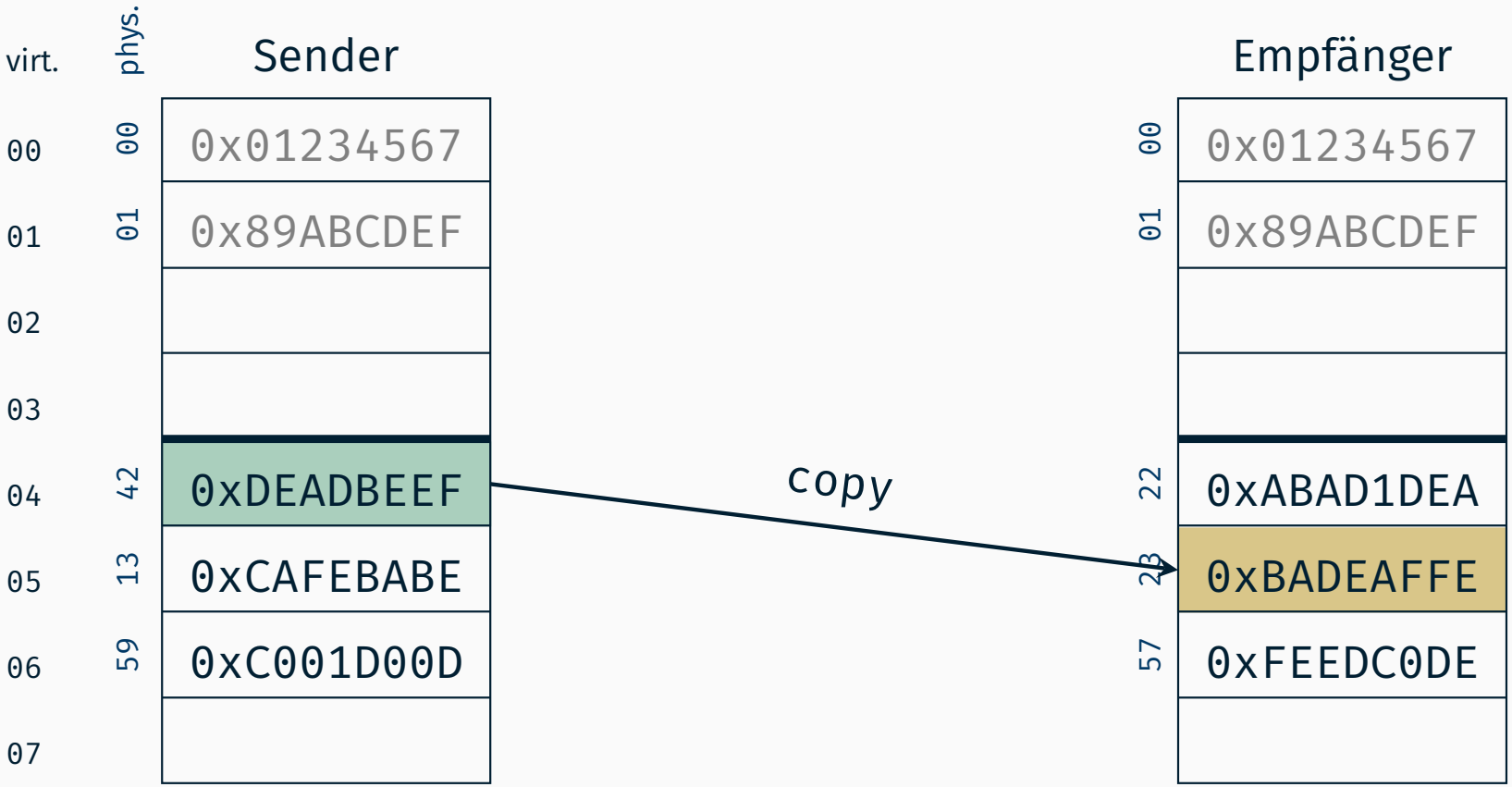
`send.sem2.v()`

Kopieren zwischen Adressräumen

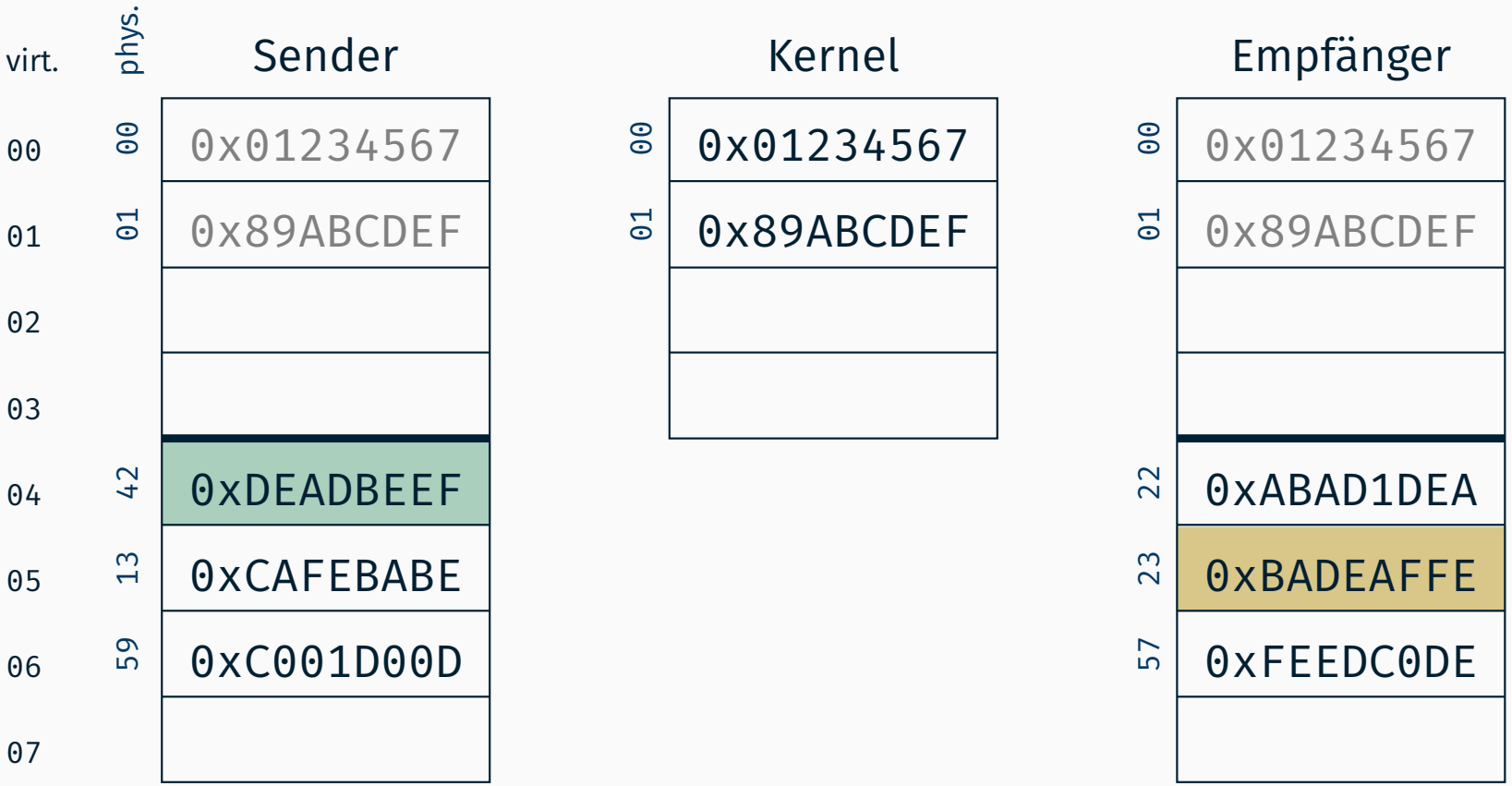
Kopieren zwischen Adressräumen



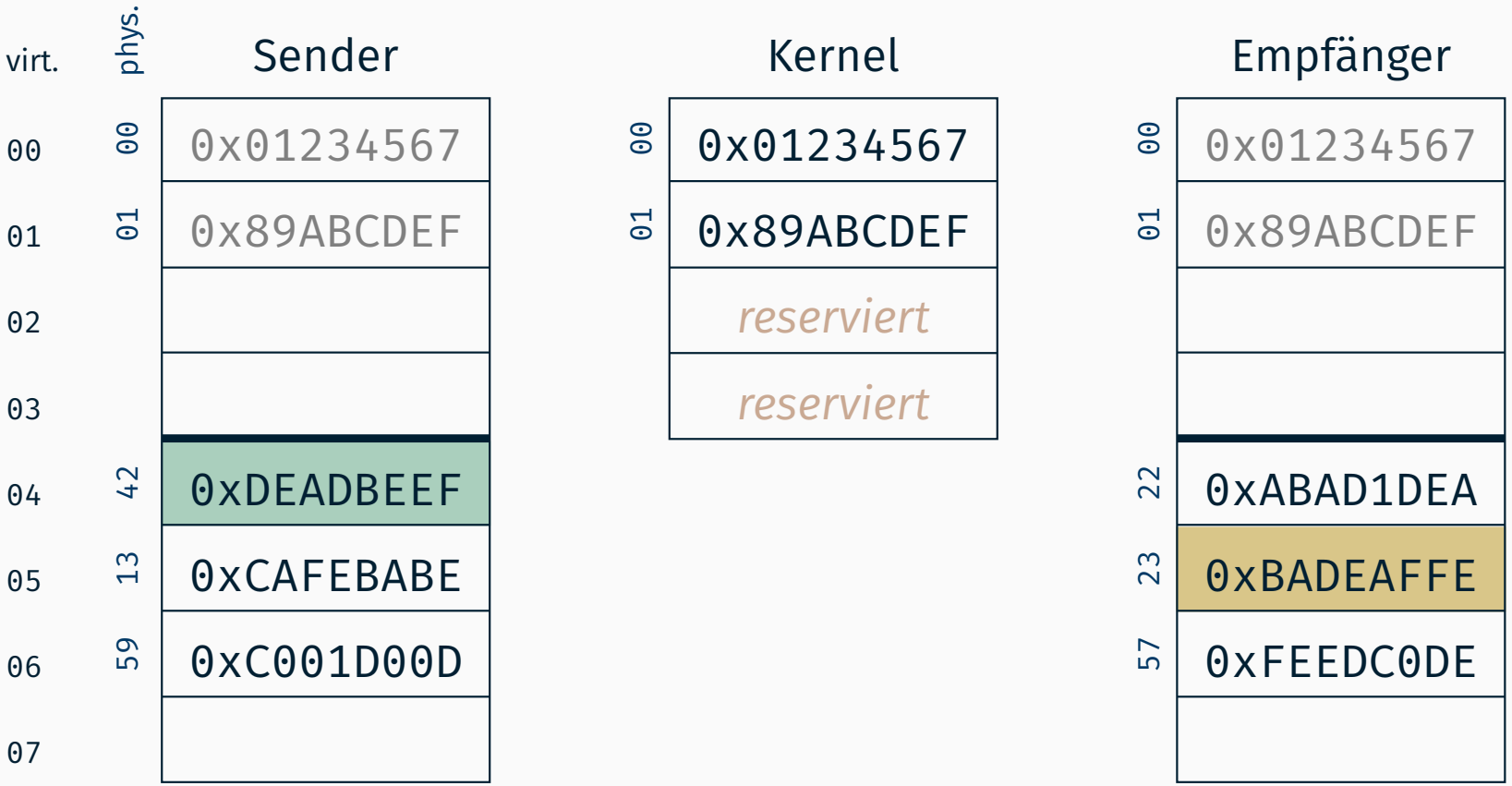
Kopieren zwischen Adressräumen



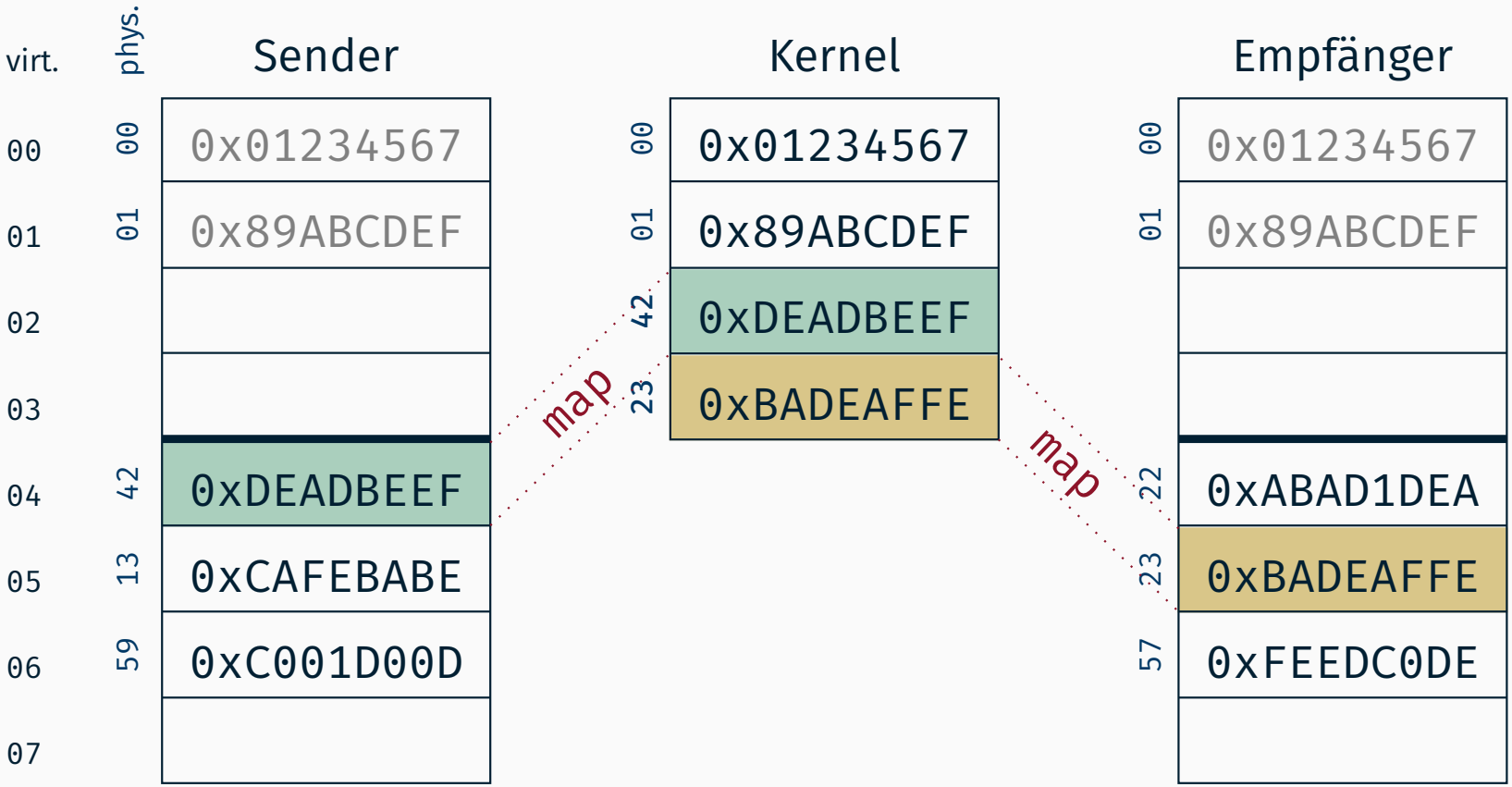
Kopieren zwischen Adressräumen



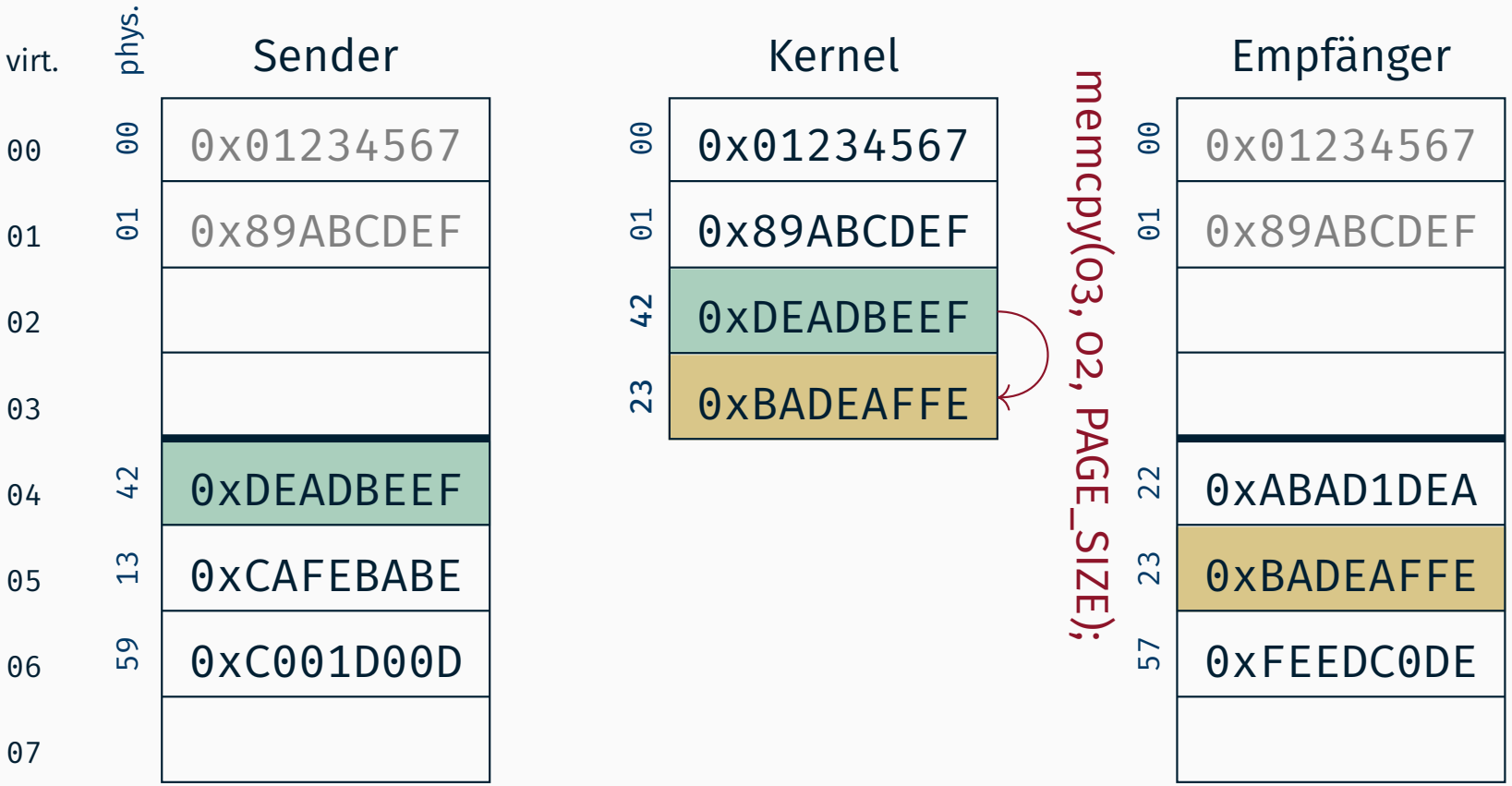
Kopieren zwischen Adressräumen



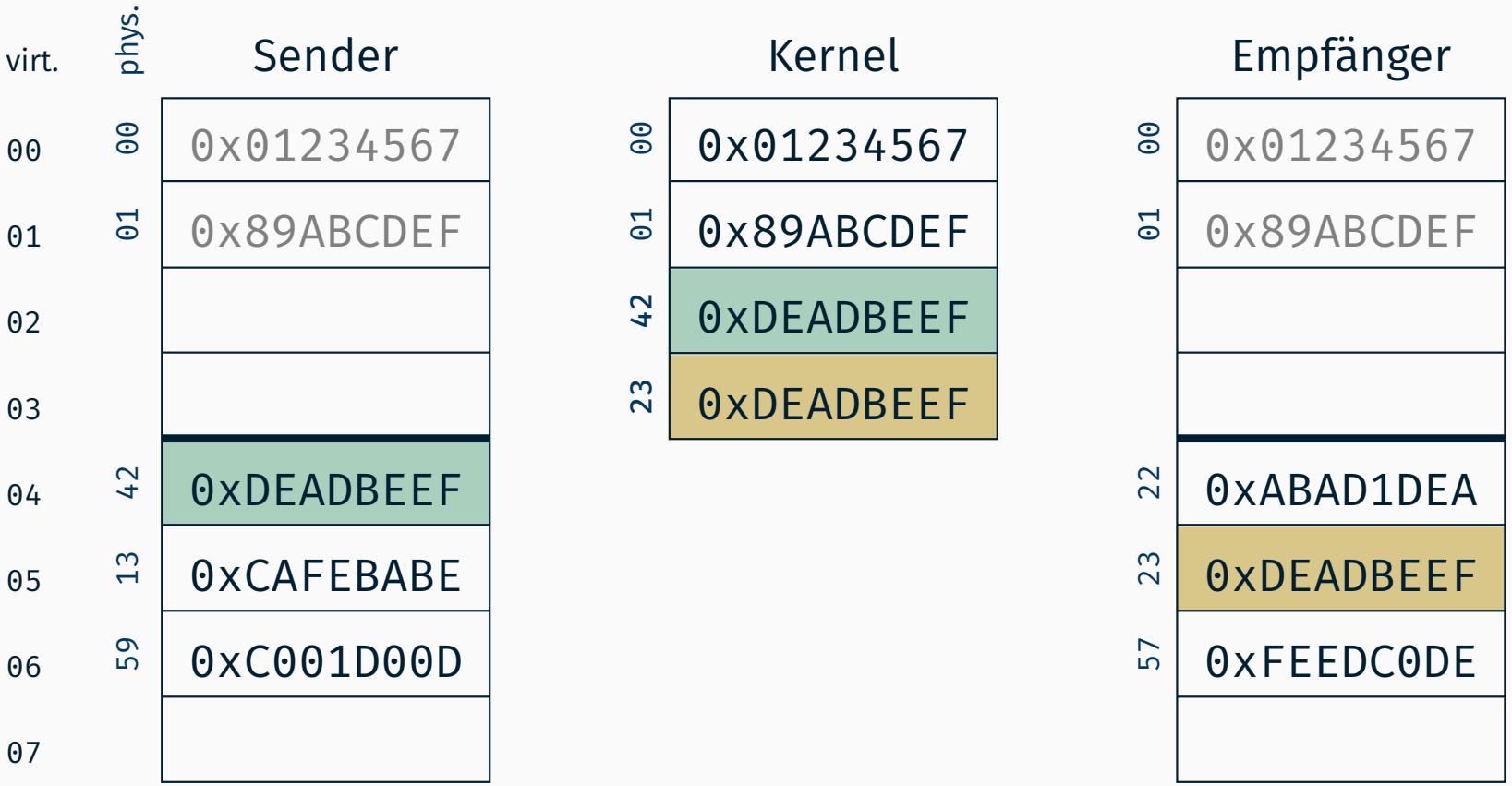
Kopieren zwischen Adressräumen



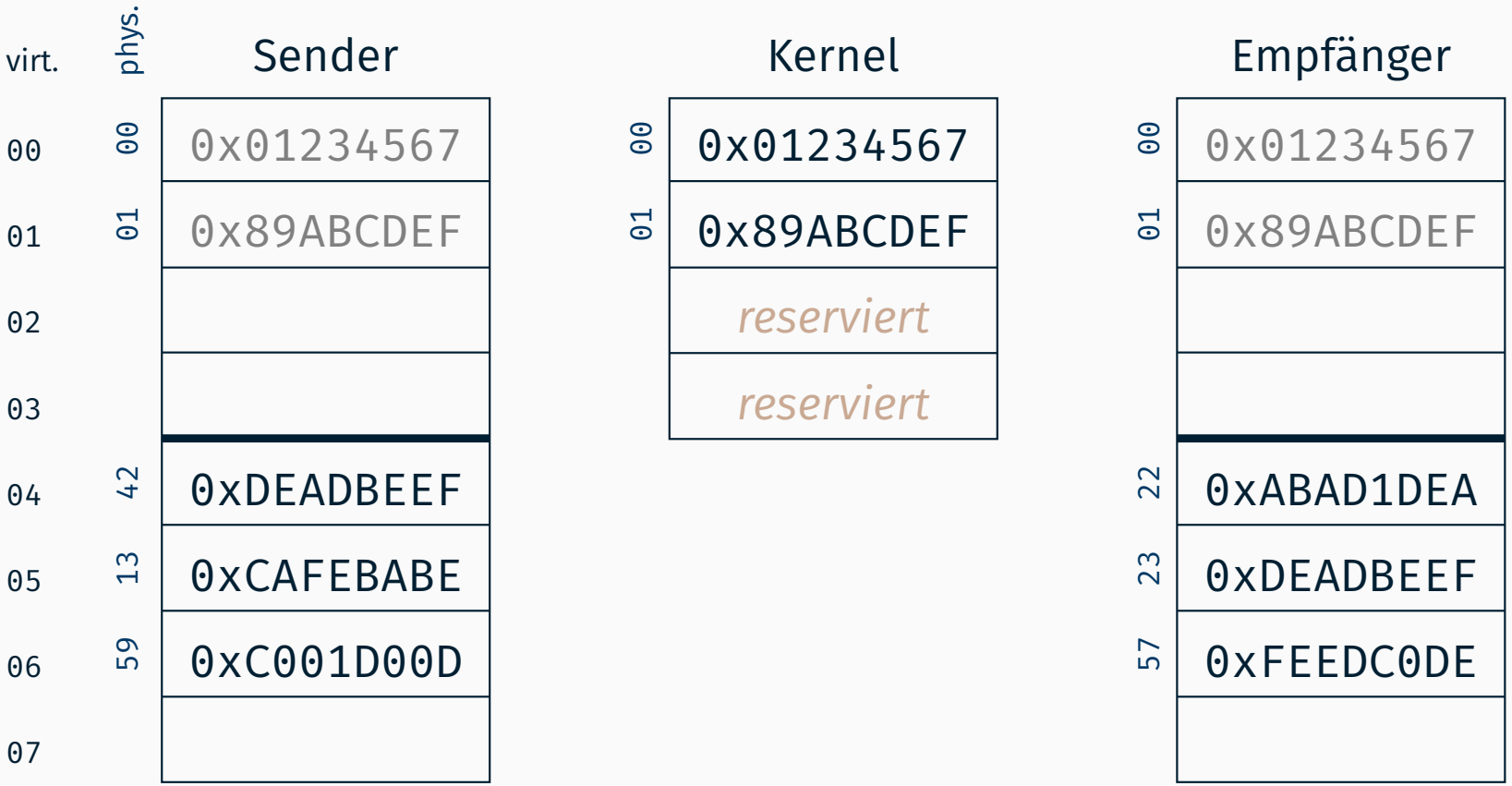
Kopieren zwischen Adressräumen



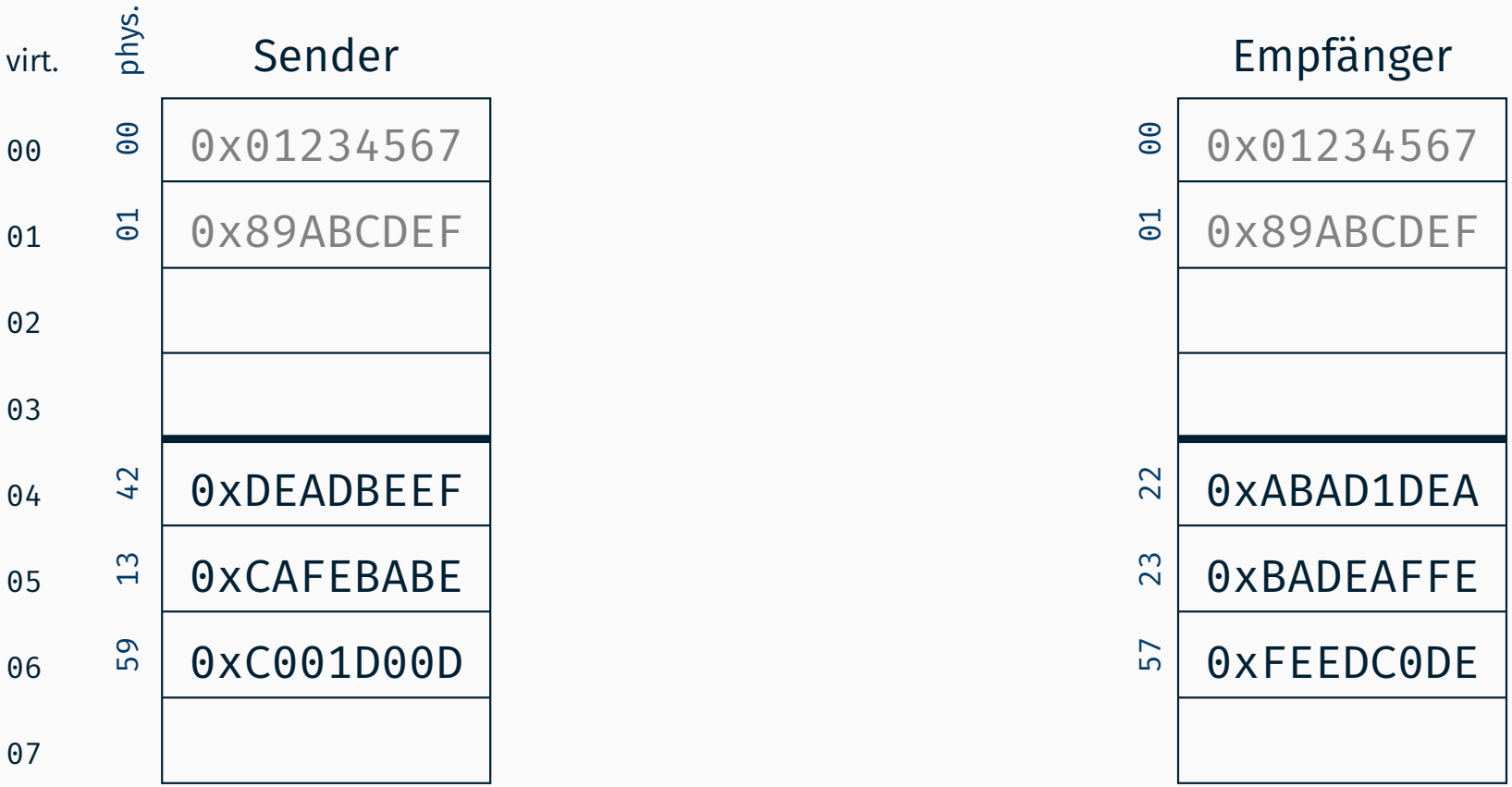
Kopieren zwischen Adressräumen



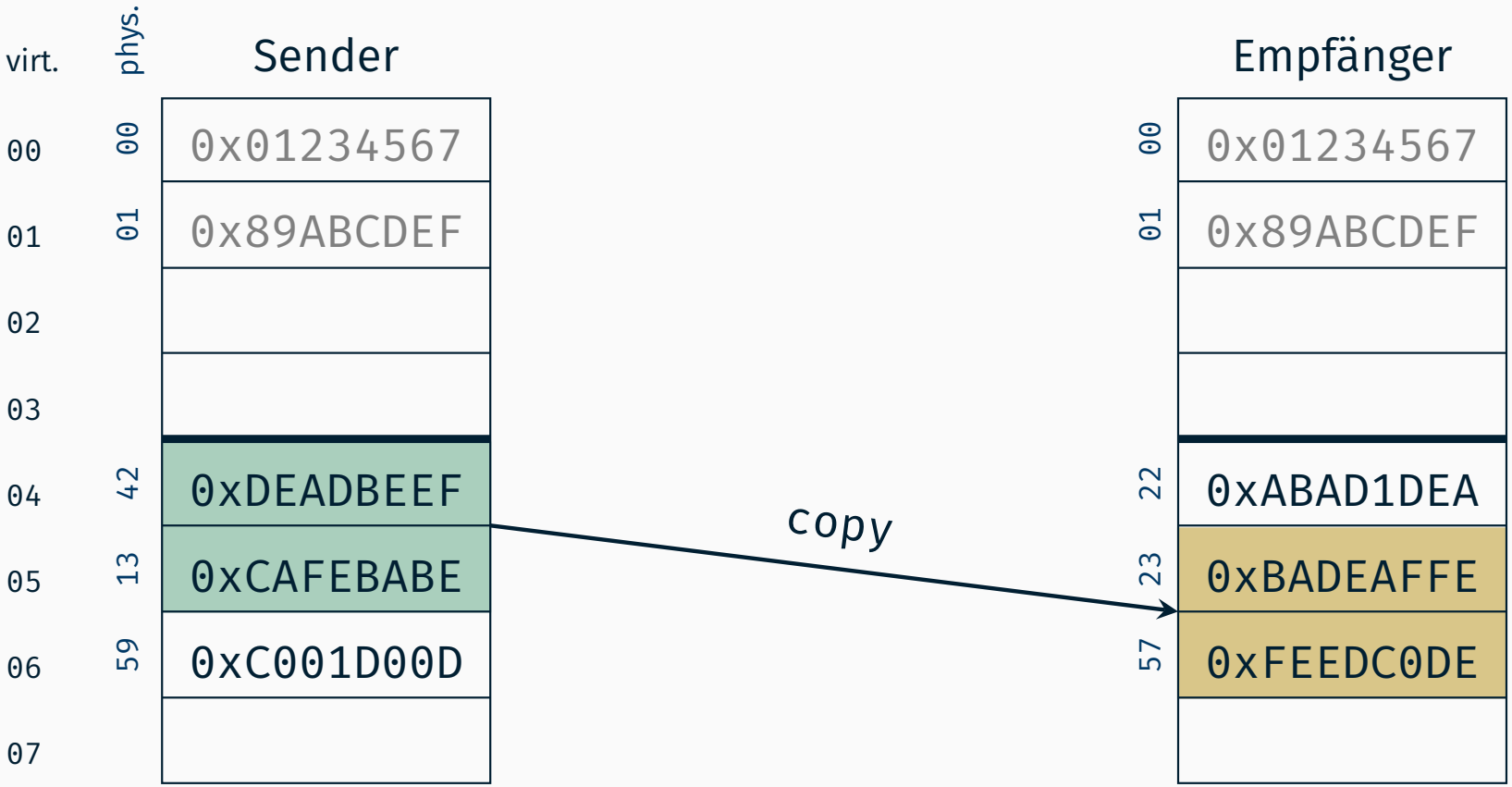
Kopieren zwischen Adressräumen



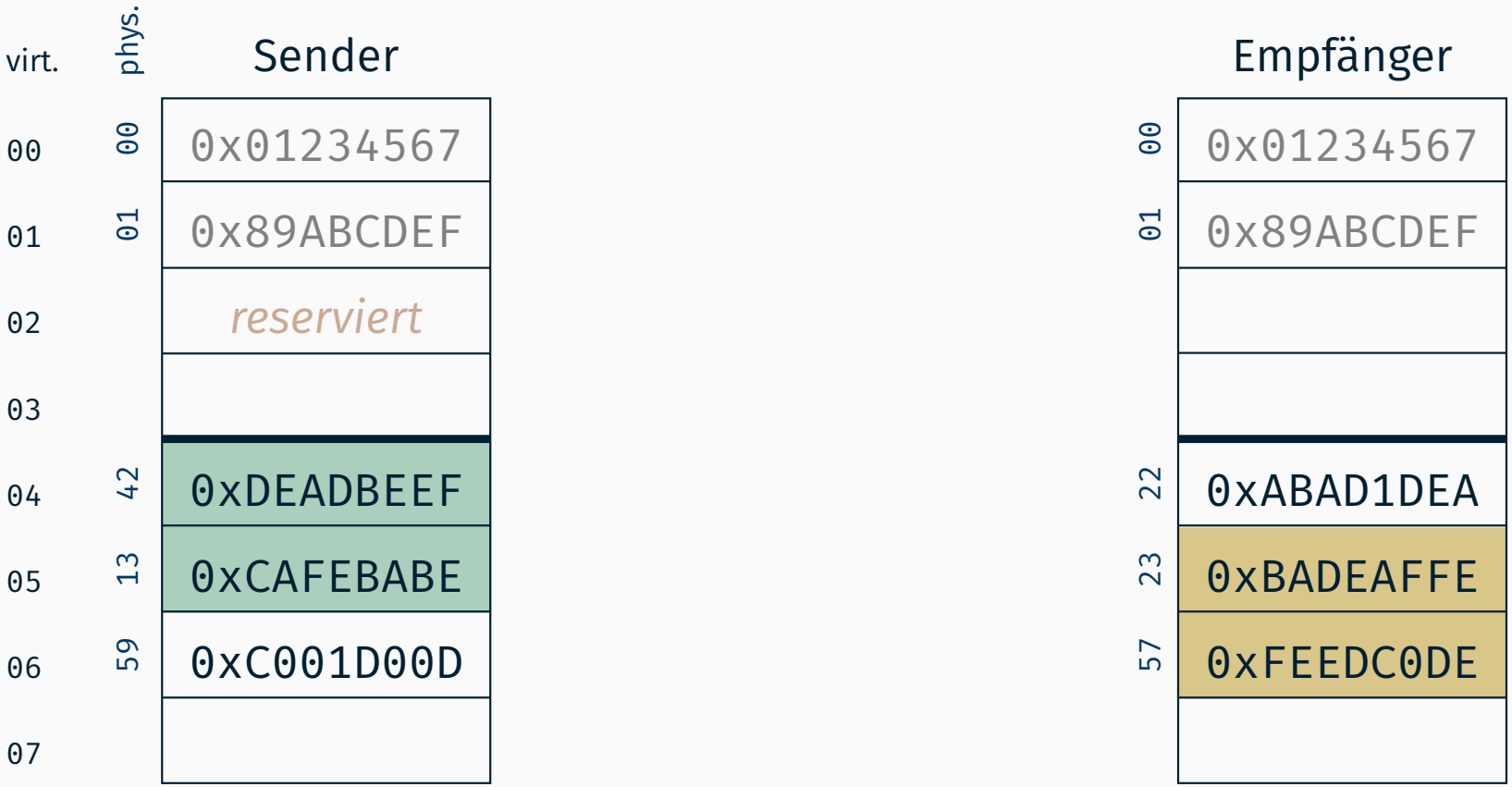
Kopieren zwischen Adressräumen (optimiert)



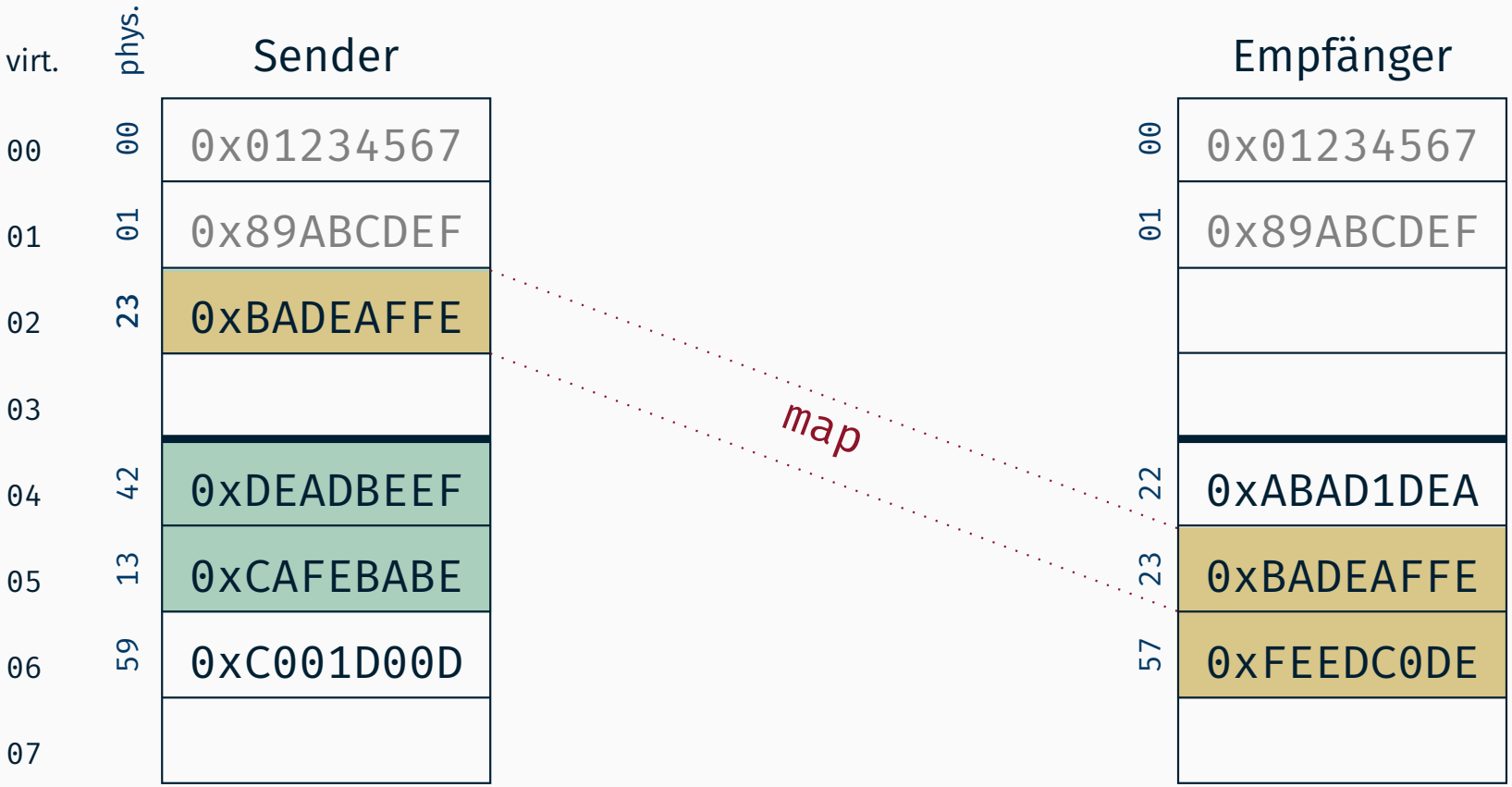
Kopieren zwischen Adressräumen (optimiert)



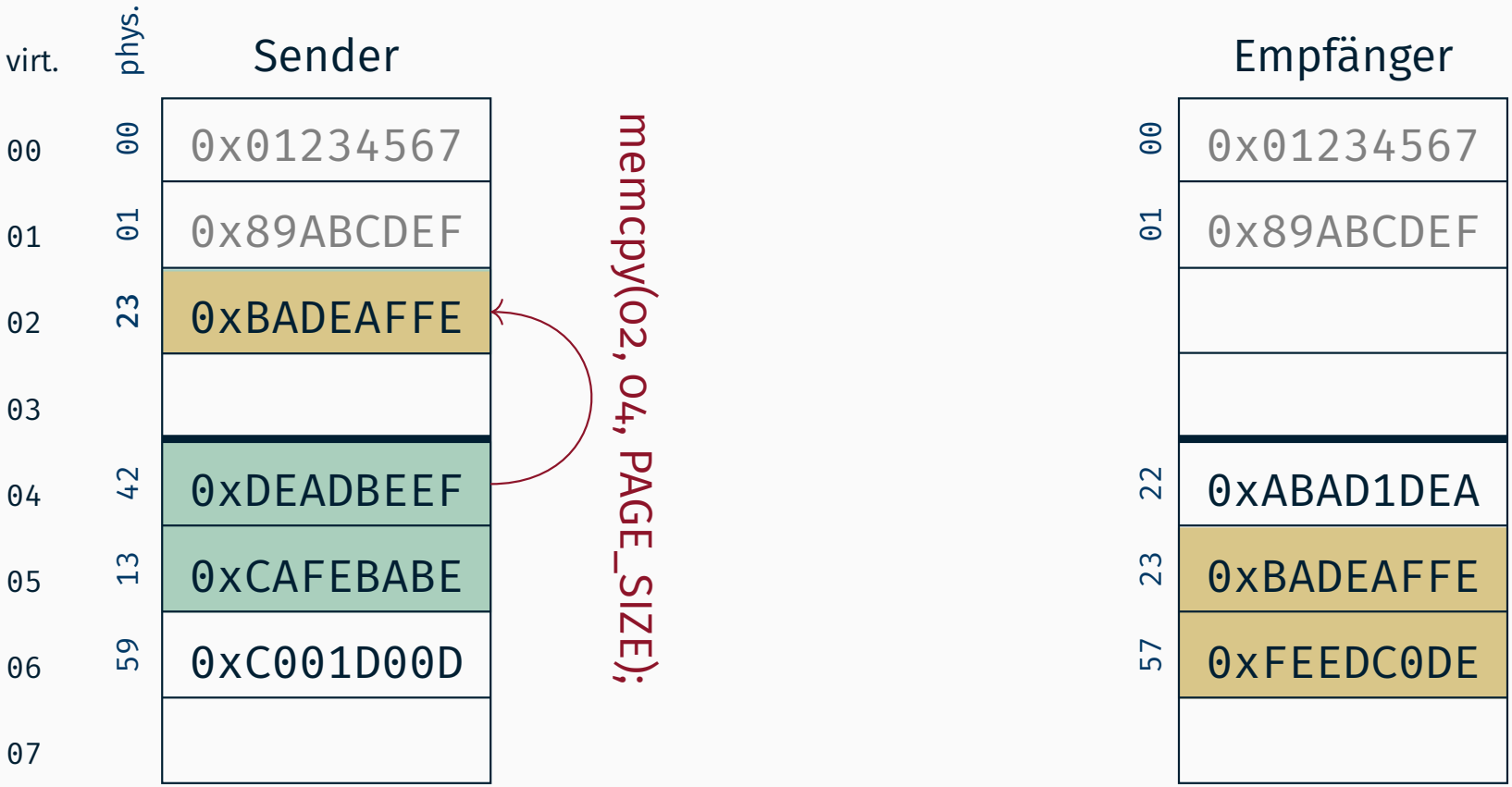
Kopieren zwischen Adressräumen (optimiert)



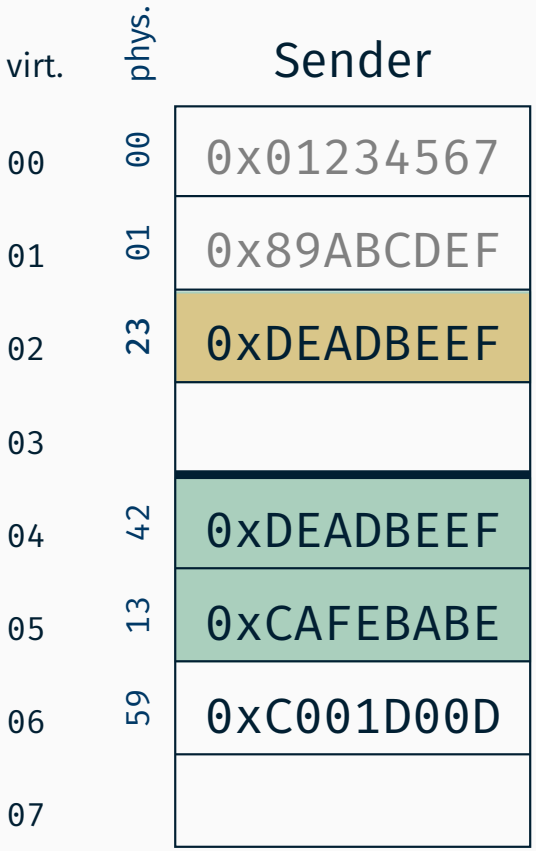
Kopieren zwischen Adressräumen (optimiert)



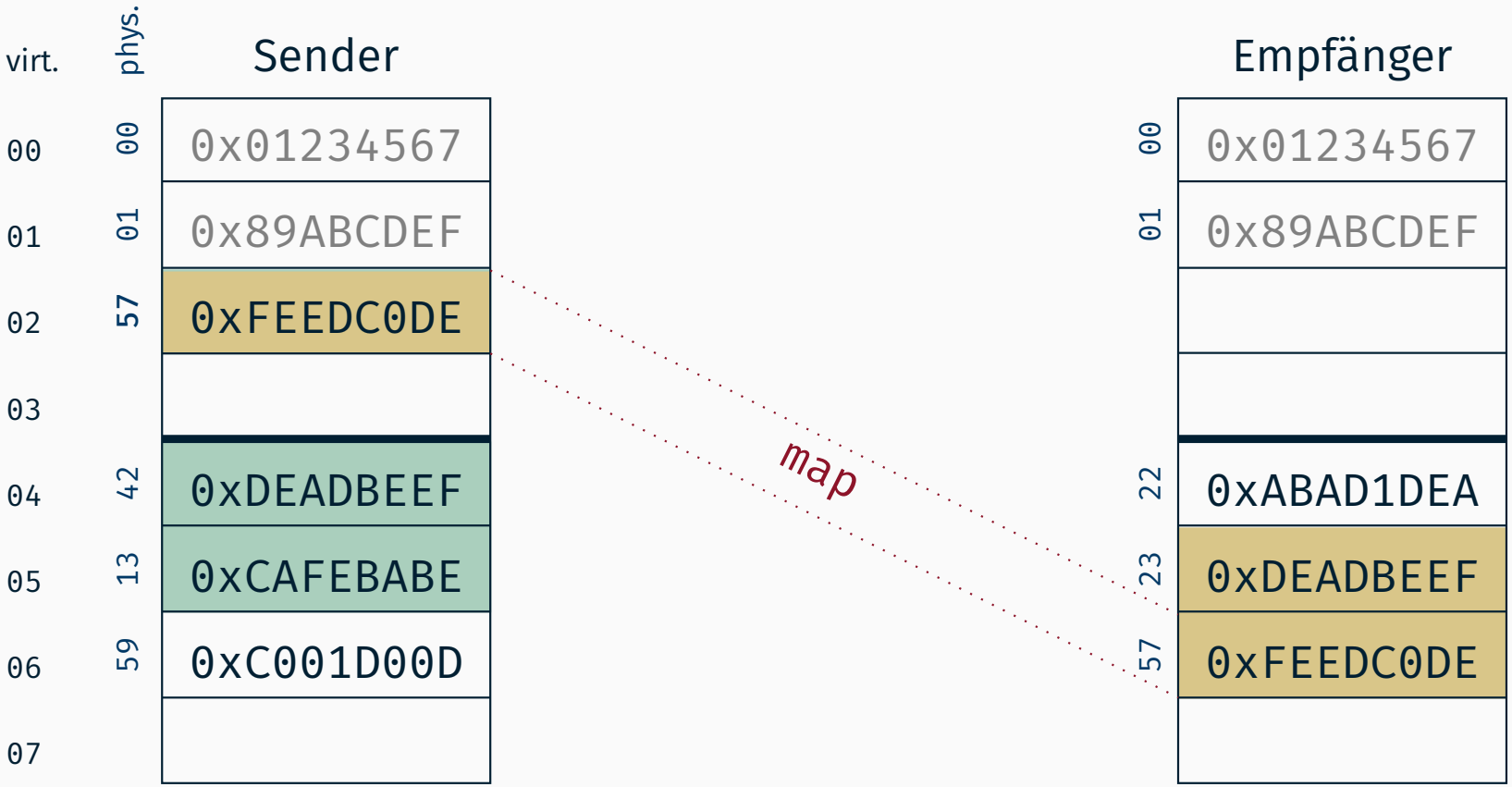
Kopieren zwischen Adressräumen (optimiert)



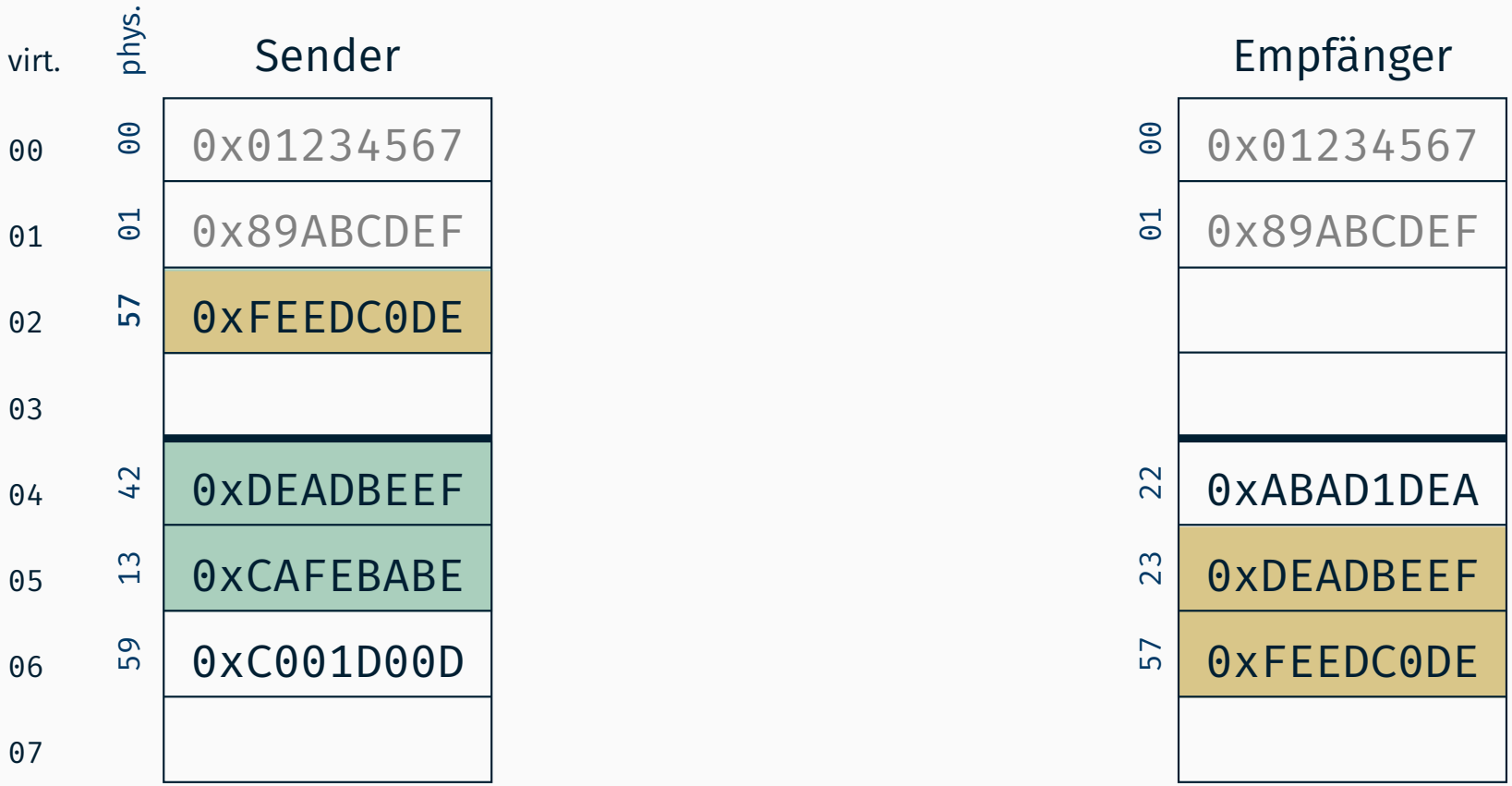
Kopieren zwischen Adressräumen (optimiert)



Kopieren zwischen Adressräumen (optimiert)



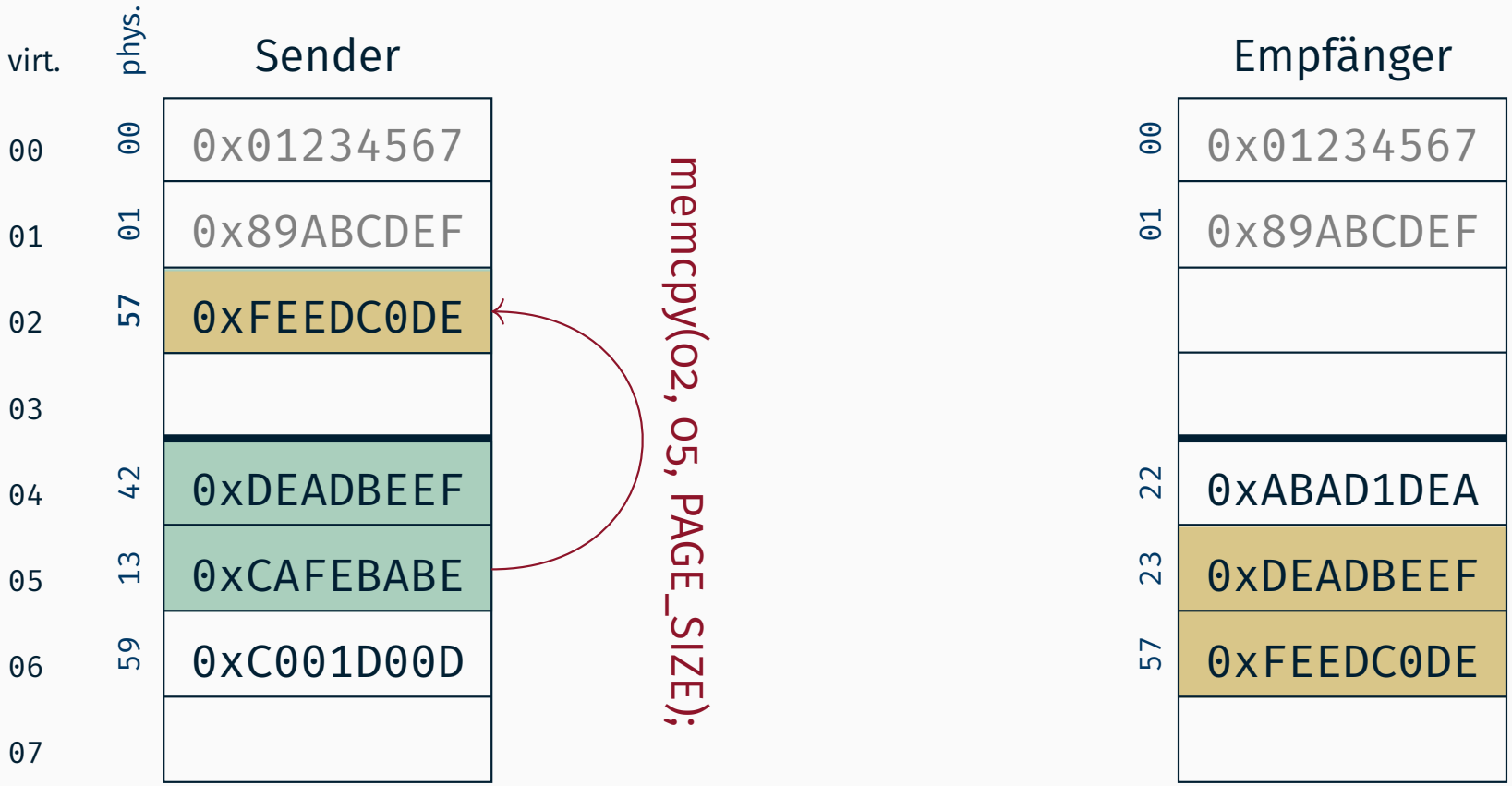
Kopieren zwischen Adressräumen (optimiert)



Achtung: Nach Änderung des Mappings TLB invalidieren!

```
asm volatile("invlpg␣(%0)": : "r"(address) : "memory");
```

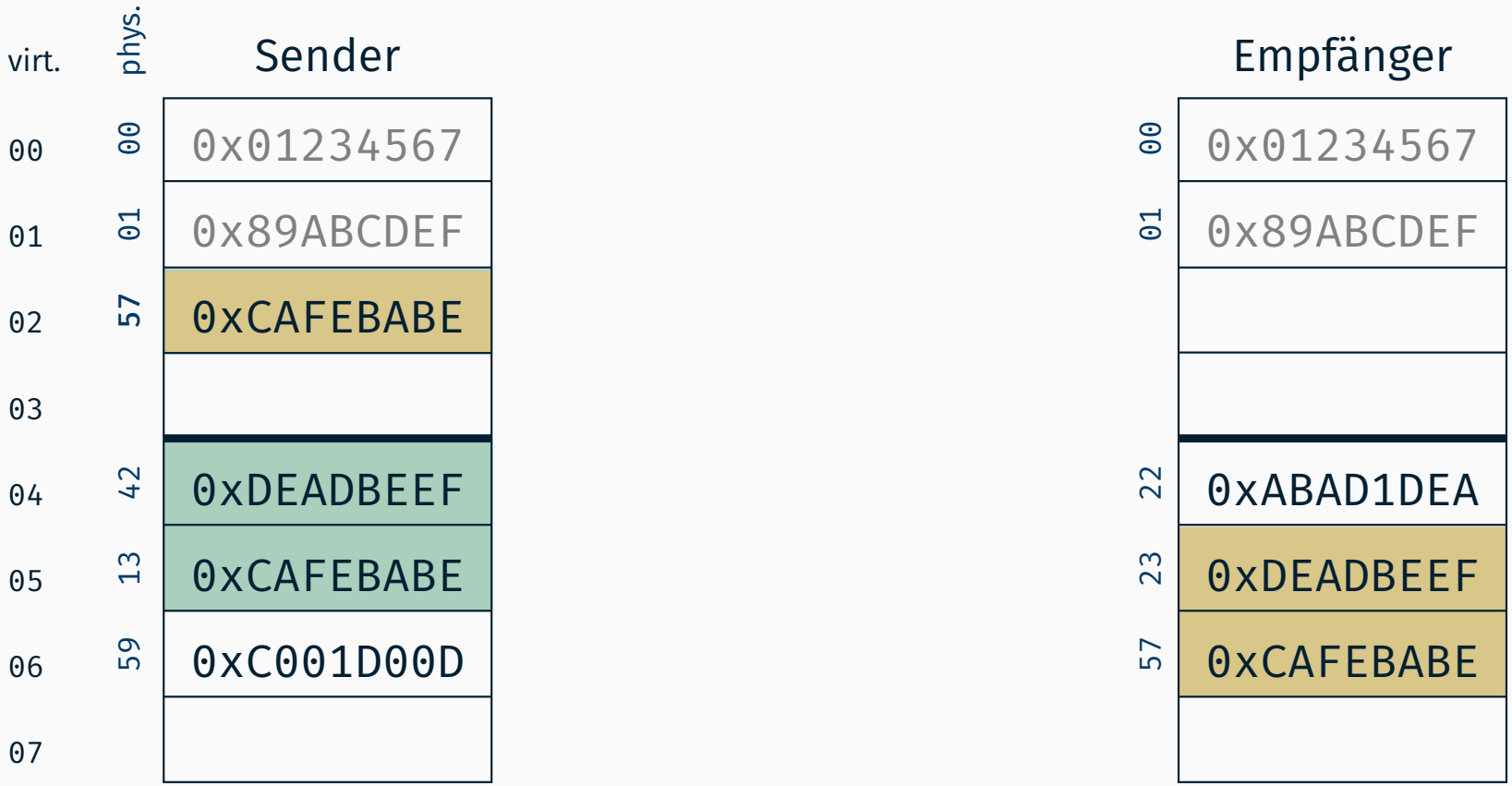
Kopieren zwischen Adressräumen (optimiert)



Achtung: Nach Änderung des Mappings TLB invalidieren!

```
asm volatile("invlpg␣(%0)": : "r"(address) : "memory");
```

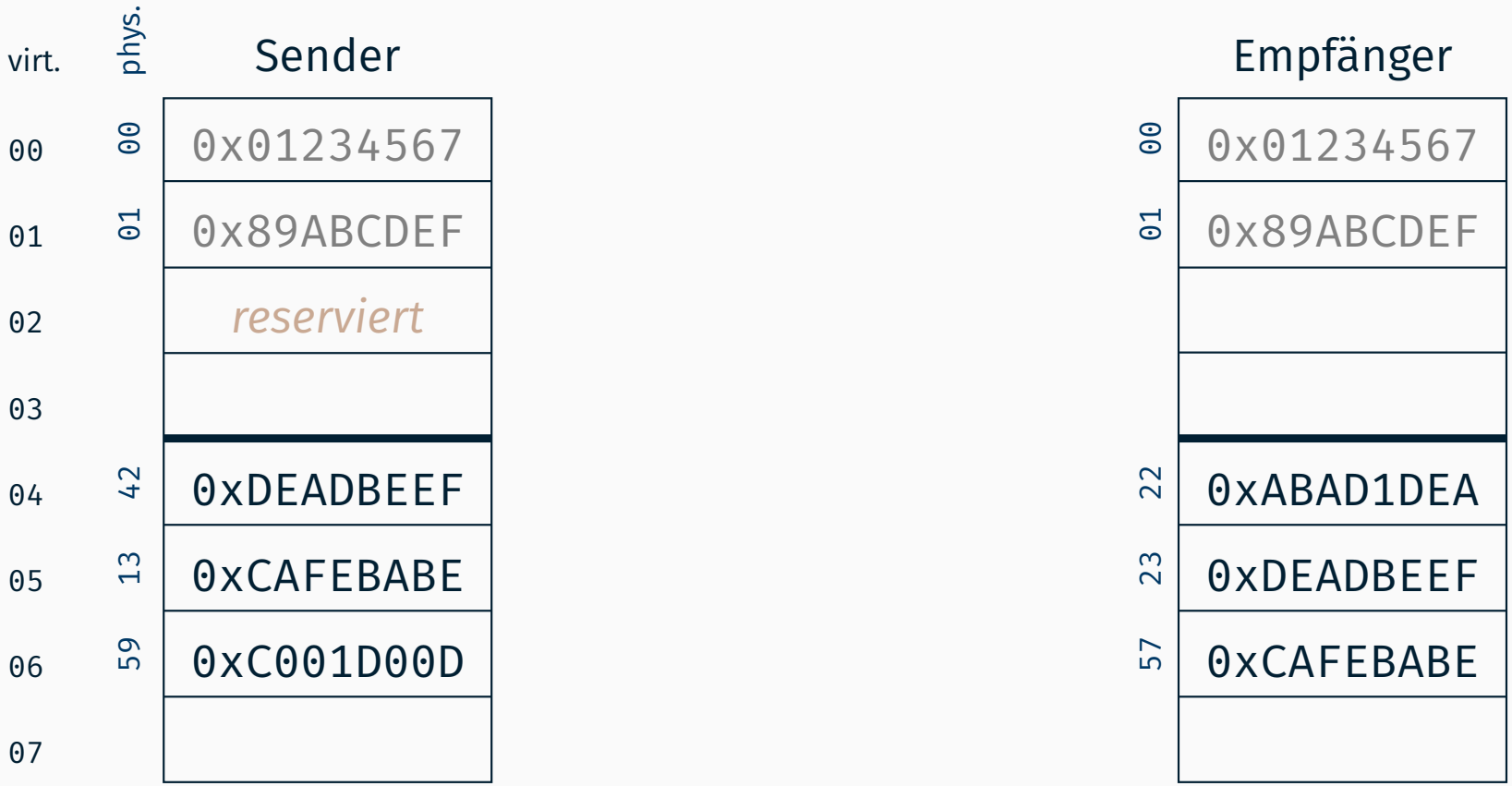
Kopieren zwischen Adressräumen (optimiert)



Achtung: Nach Änderung des Mappings TLB invalidieren!

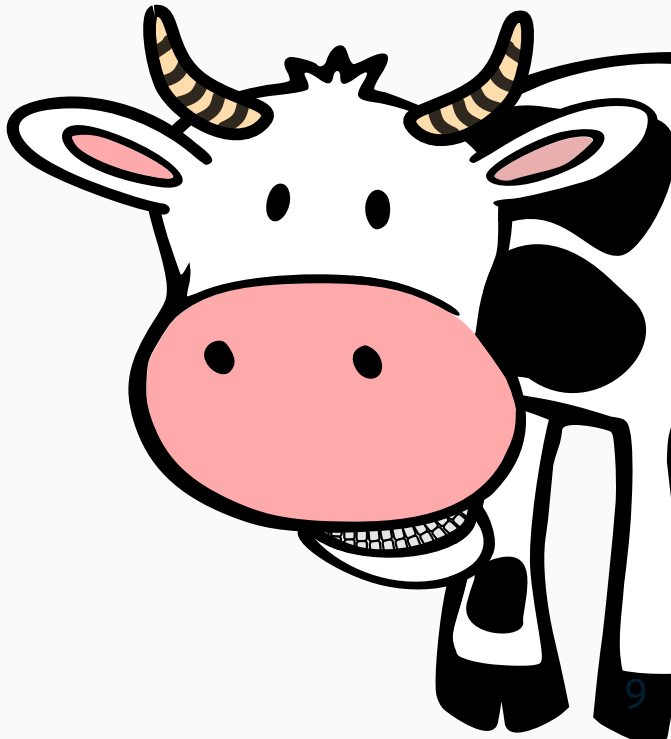
```
asm volatile("invlpg␣(%0)": : "r"(address) : "memory");
```

Kopieren zwischen Adressräumen (optimiert)

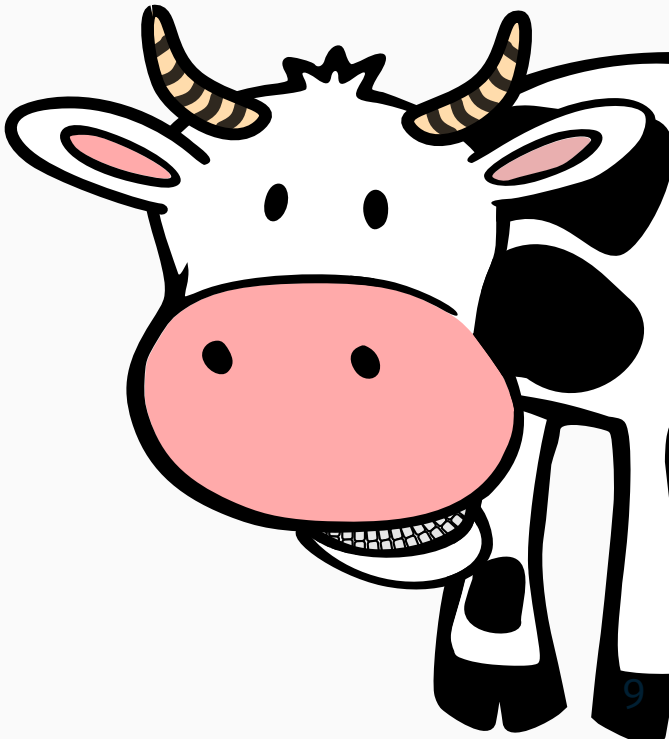


Achtung: Nach Änderung des Mappings TLB invalidieren!

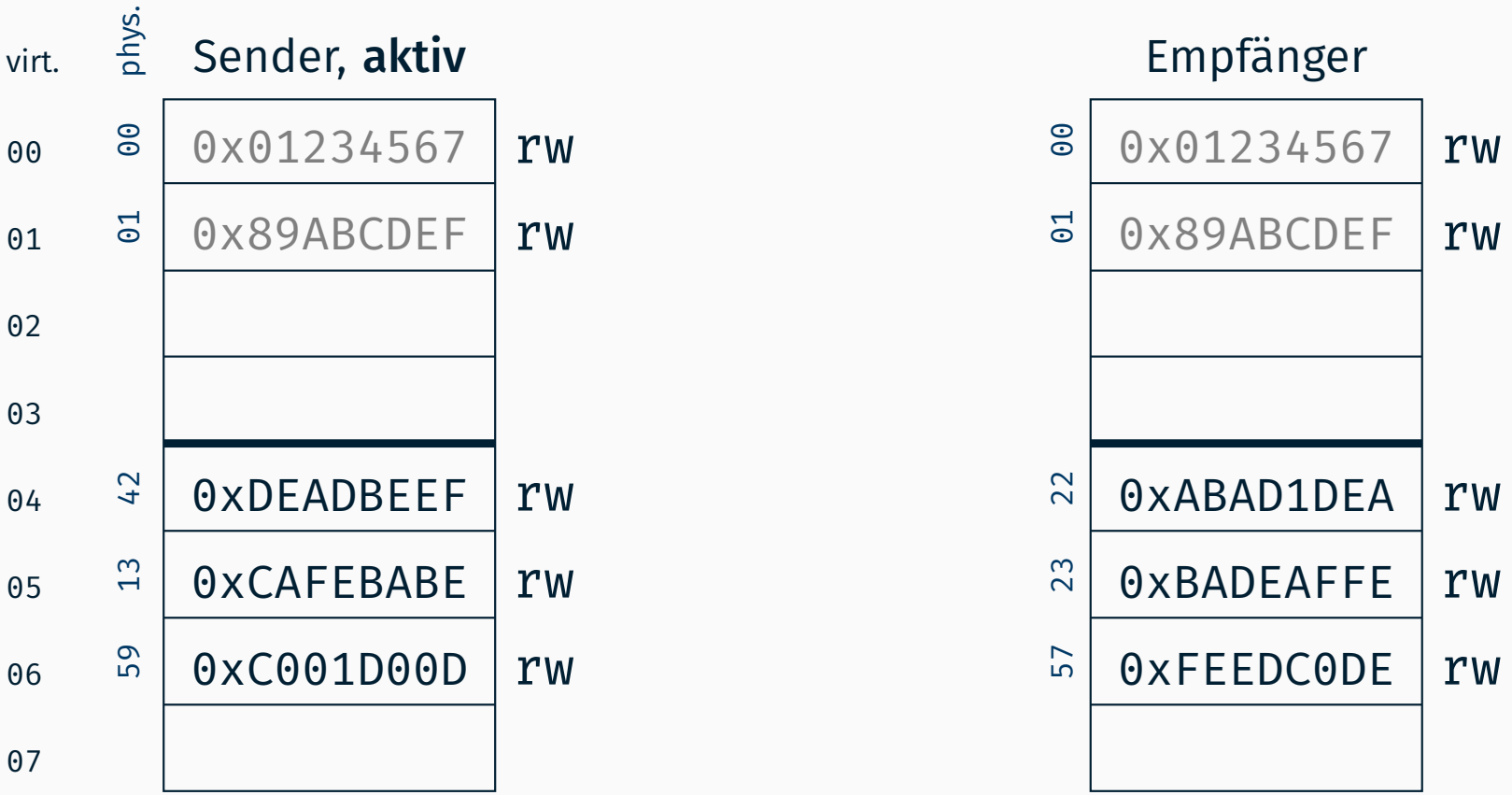
```
asm volatile("invlpg␣(%0)": : "r"(address) : "memory");
```



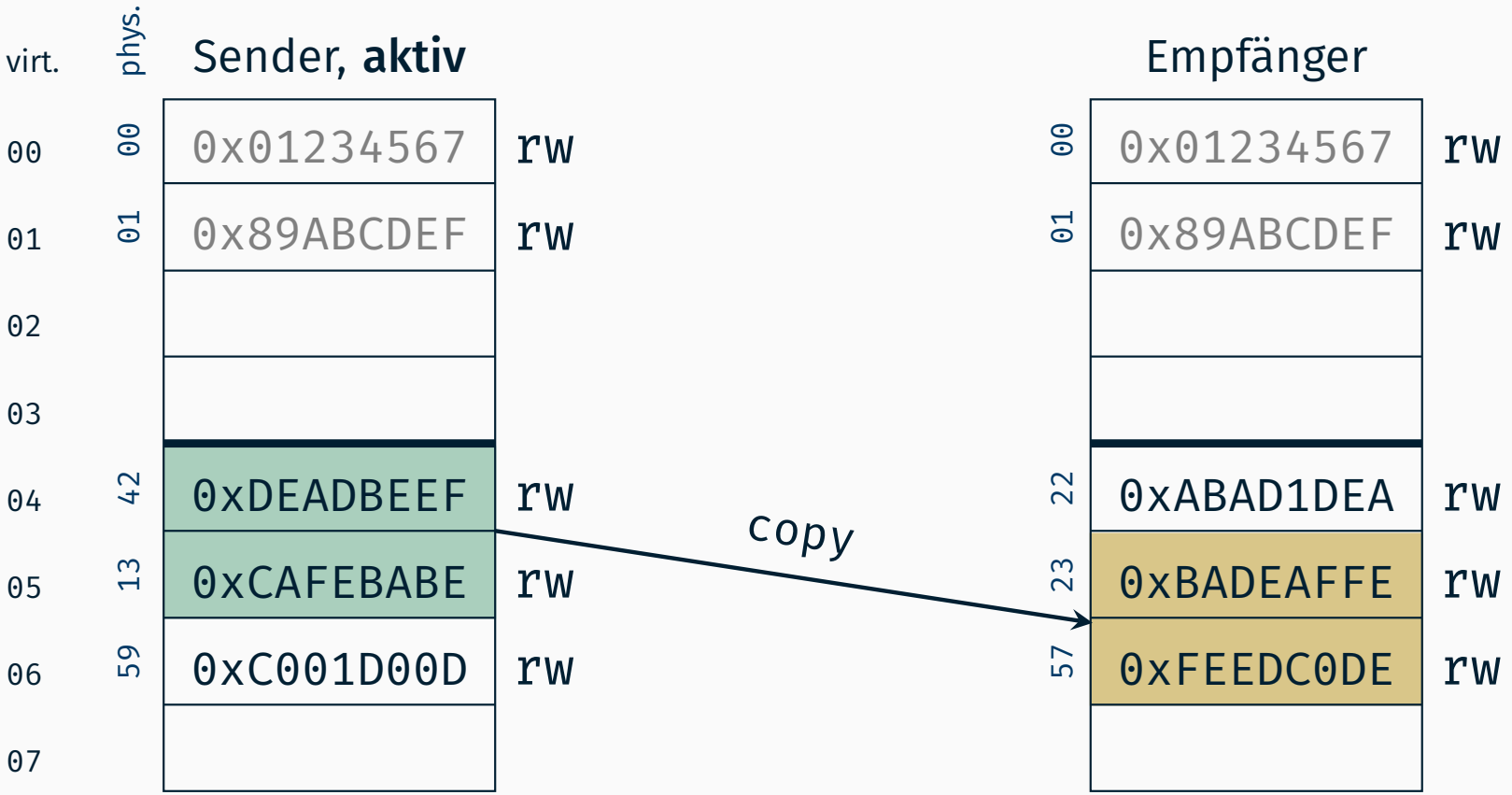
Copy-on-Write (CoW)



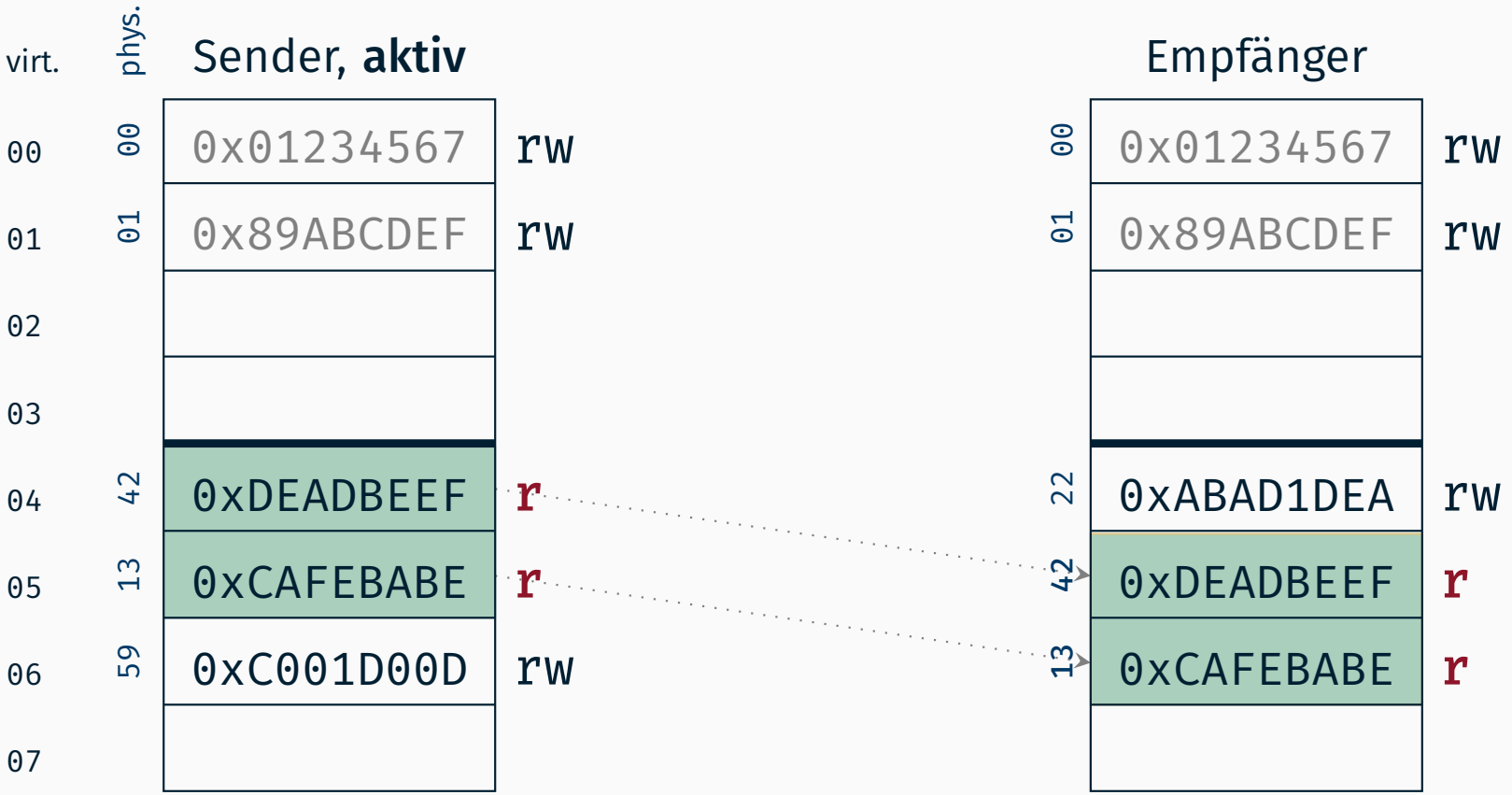
Copy-on-Write



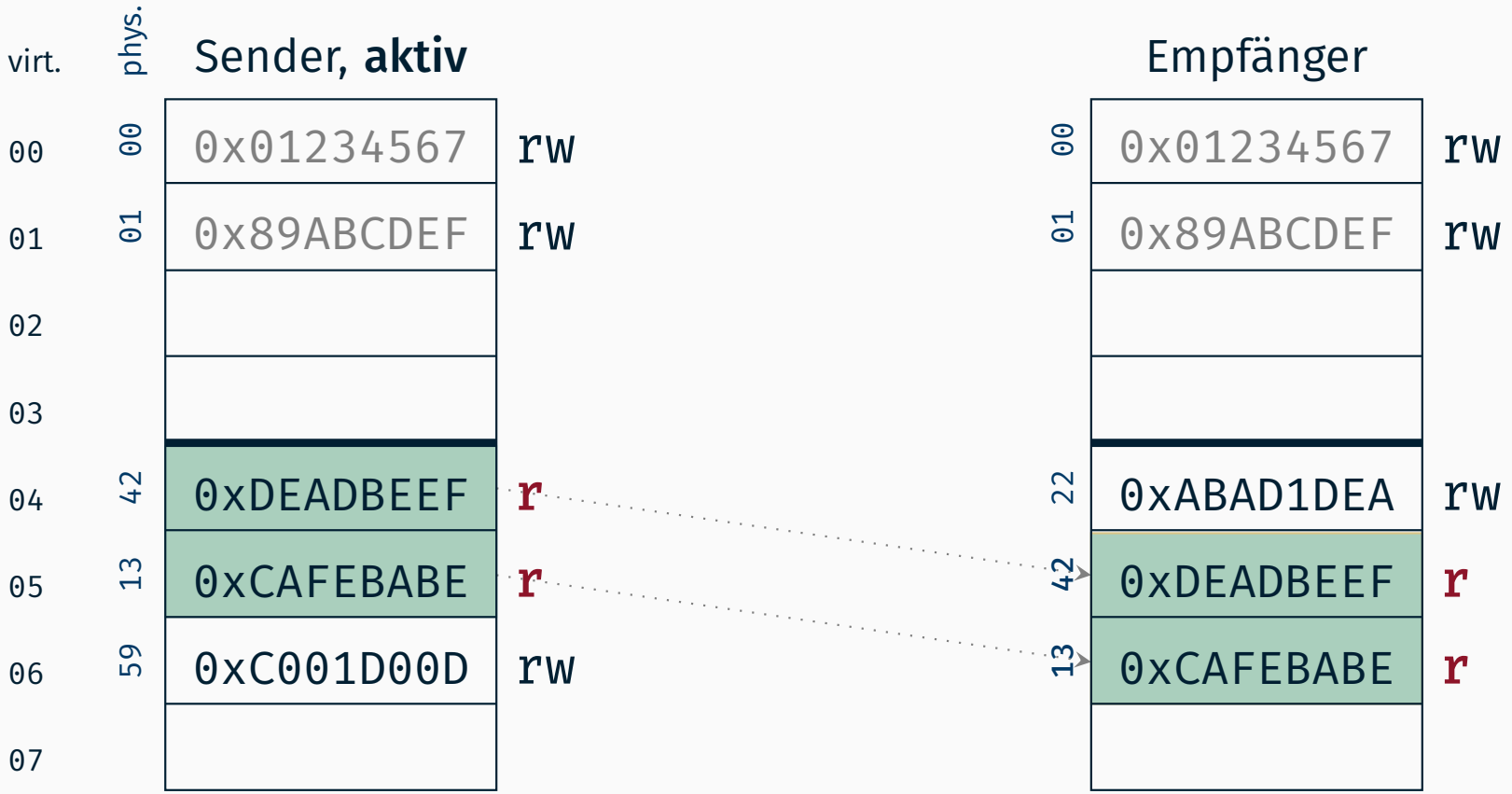
Copy-on-Write



Copy-on-Write



Copy-on-Write



Achtung: Nach Änderung des Mappings TLB invalidieren!

```
asm volatile("invlpg␣(%0)": : "r"(address) : "memory");
```

Copy-on-Write: Details

Wann/wo werden Seiten tatsächlich kopiert?

Copy-on-Write: Details

Wann/wo werden Seiten tatsächlich kopiert?

- Im Pagefault-Handler (Trap bei Schreibzugriffen)

Copy-on-Write: Details

Wann/wo werden Seiten tatsächlich kopiert?

- Im Pagefault-Handler (Trap bei Schreibzugriffen)

Was passiert, wenn Puffer weitergeschickt wird?

Copy-on-Write: Details

Wann/wo werden Seiten tatsächlich kopiert?

- Im Pagefault-Handler (Trap bei Schreibzugriffen)

Was passiert, wenn Puffer weitergeschickt wird?

- das Gleiche!

Copy-on-Write: Details

Wann/wo werden Seiten tatsächlich kopiert?

- Im Pagefault-Handler (Trap bei Schreibzugriffen)

Was passiert, wenn Puffer weitergeschickt wird?

- das Gleiche!
- Referenzen zählen (auch bei `fork()`)

Implementierung des Referenzzählers?

Copy-on-Write: Details

Wann/wo werden Seiten tatsächlich kopiert?

- Im Pagefault-Handler (Trap bei Schreibzugriffen)

Was passiert, wenn Puffer weitergeschickt wird?

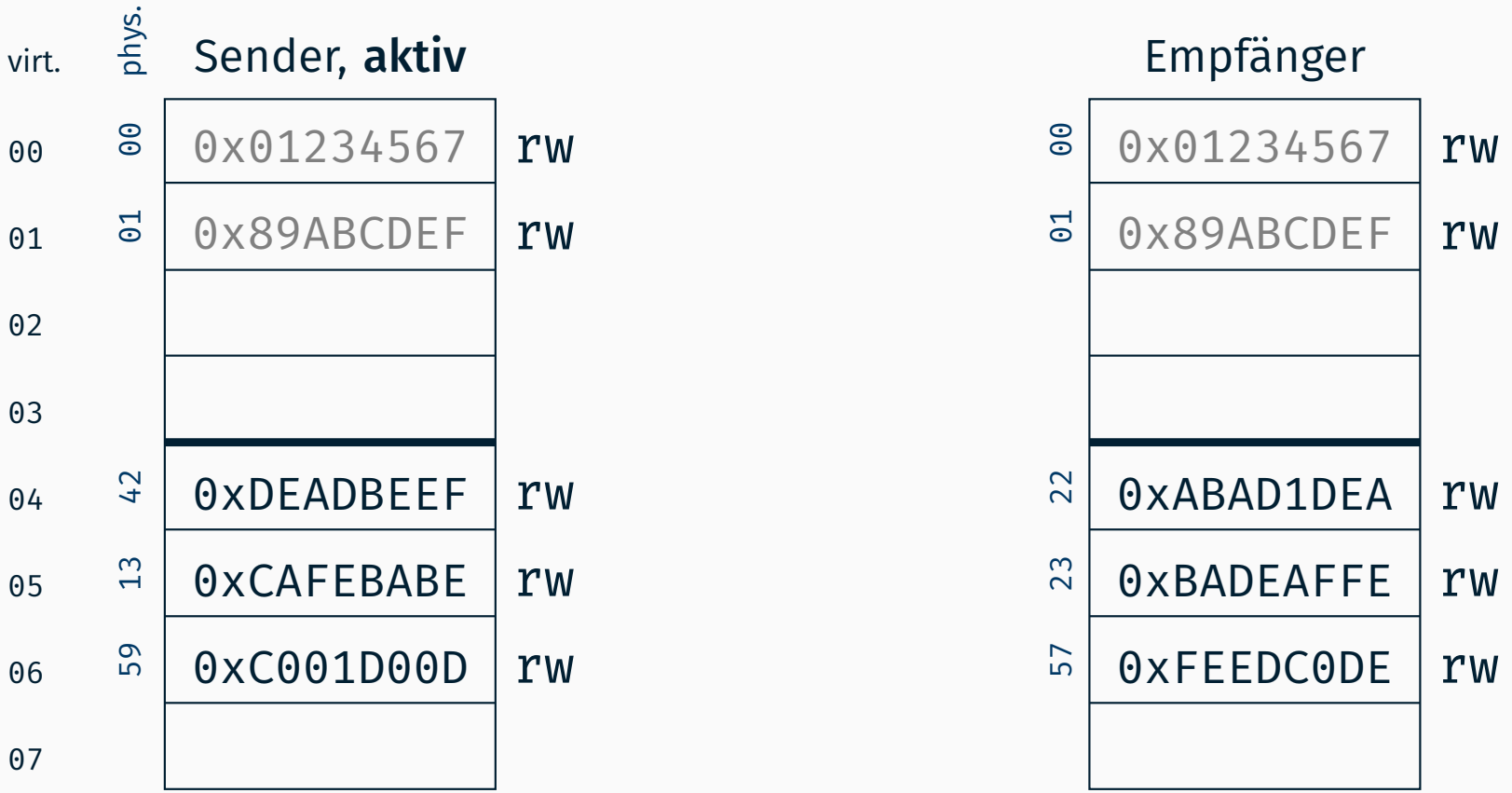
- das Gleiche!
- Referenzen zählen (auch bei `fork()`)

Implementierung des Referenzzählers?

- (z.B.) Shadow-Pagetable

```
struct PTE_Counter {
    uint32_t present:1,
             counter:31;
};
```

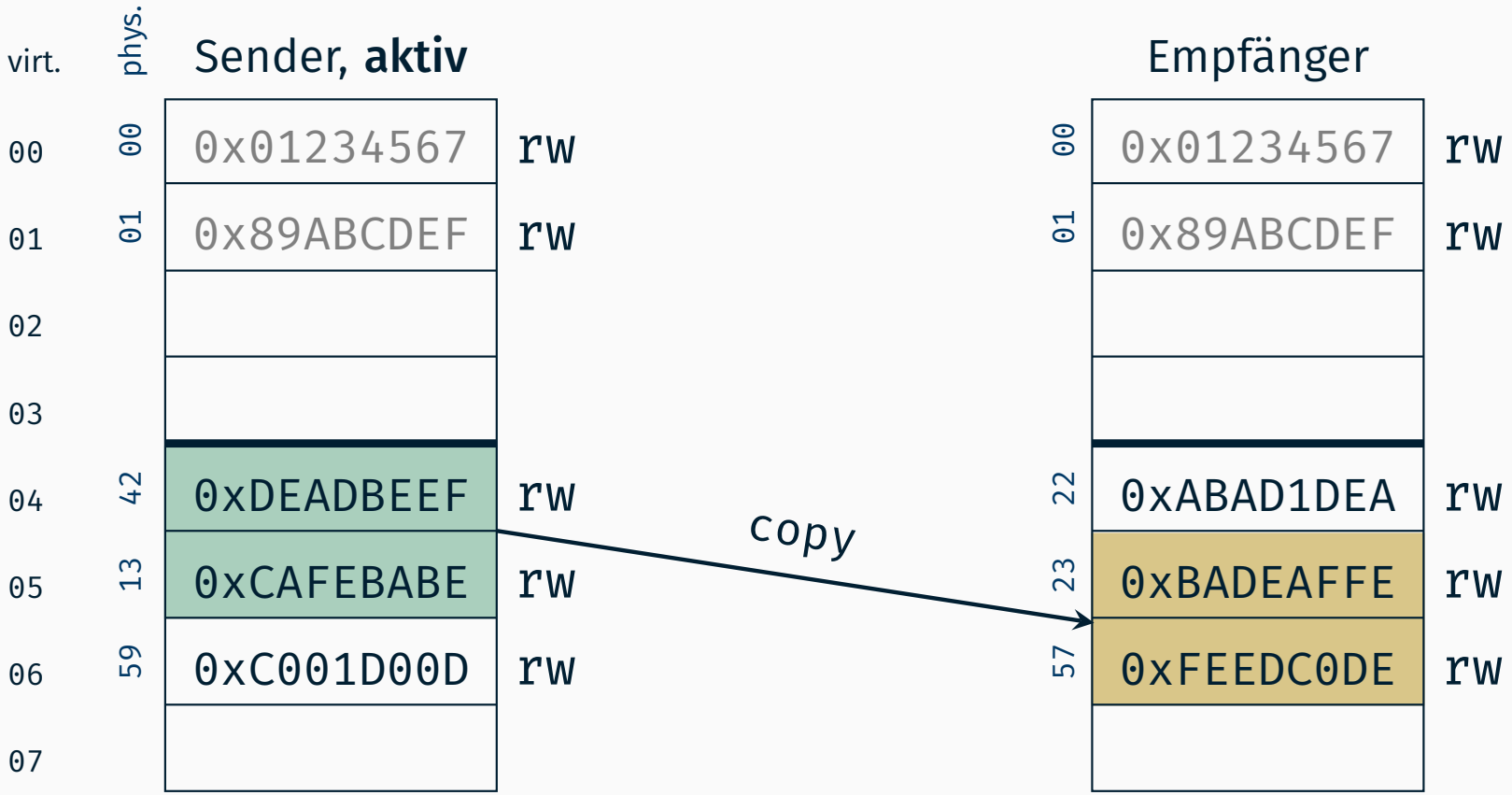
Copy-on-Write



Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58	59	...
Ref.	...	0	1	0	...	0	1	1	...	1	...	1	0	1	...

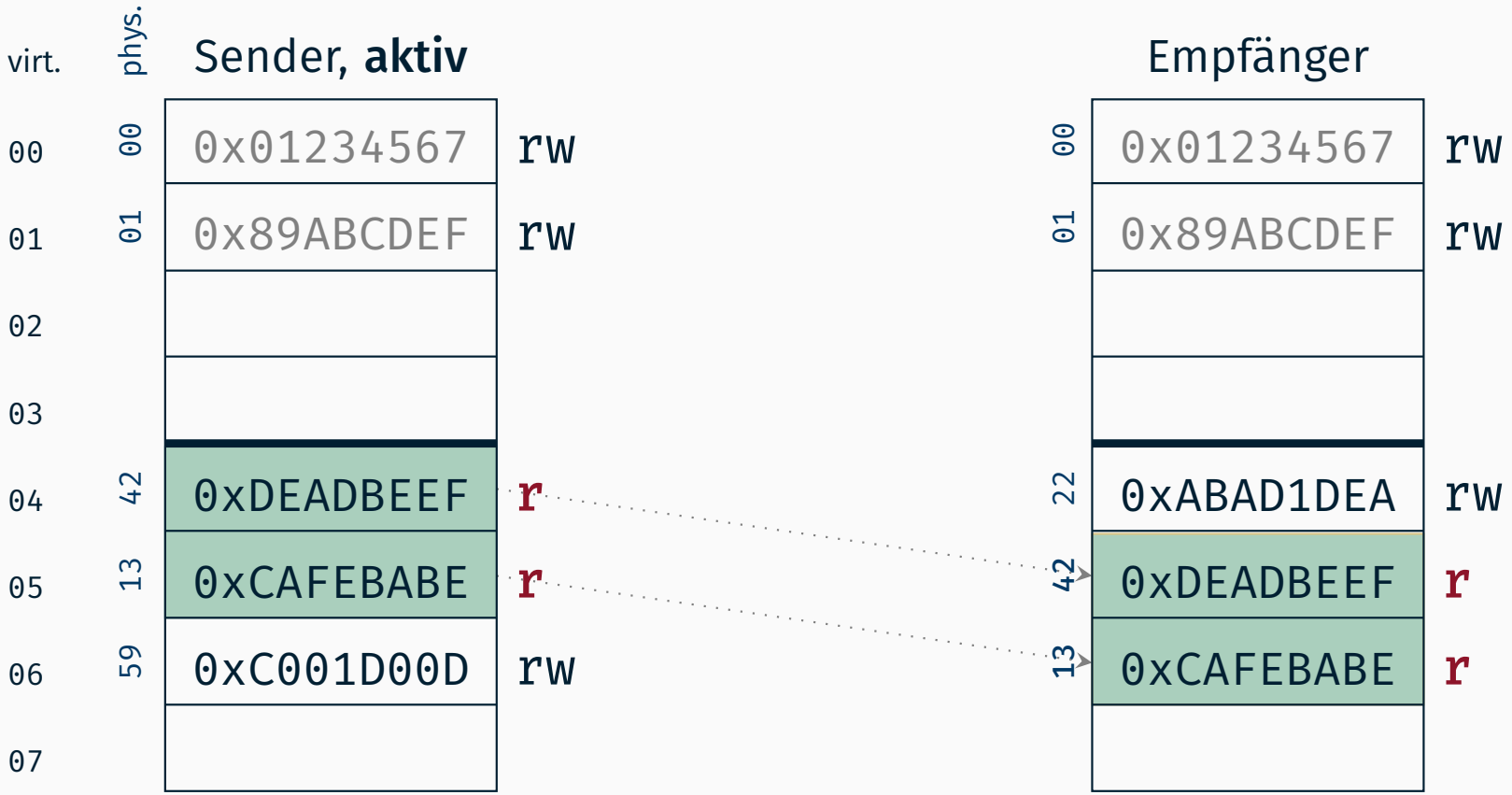
Copy-on-Write



Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58	59	...
Ref.	...	0	1	0	...	0	1	1	...	1	...	1	0	1	...

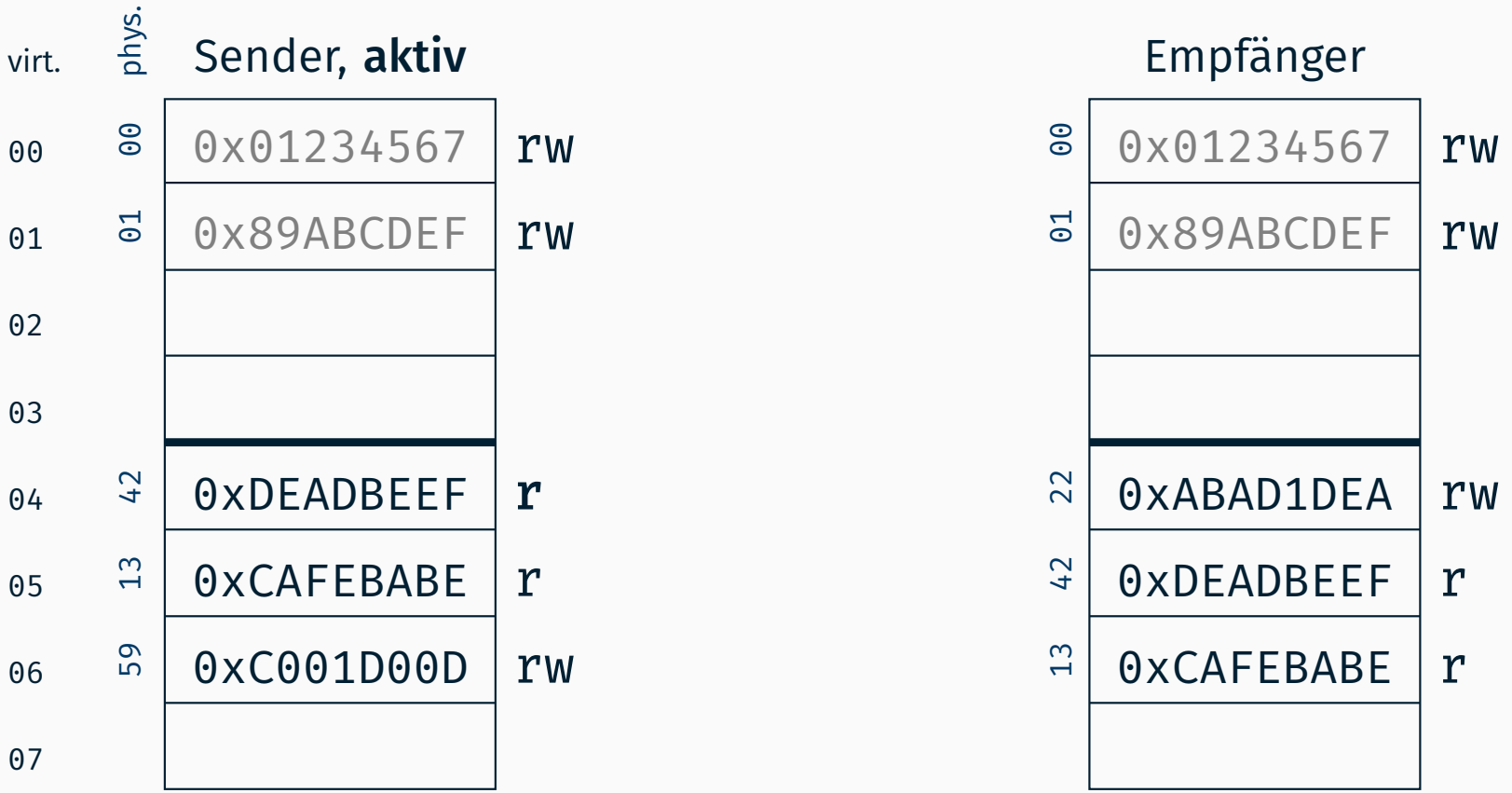
Copy-on-Write



Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58	59	...
Ref.	...	0	2	0	...	0	1	0	...	2	...	0	0	1	...

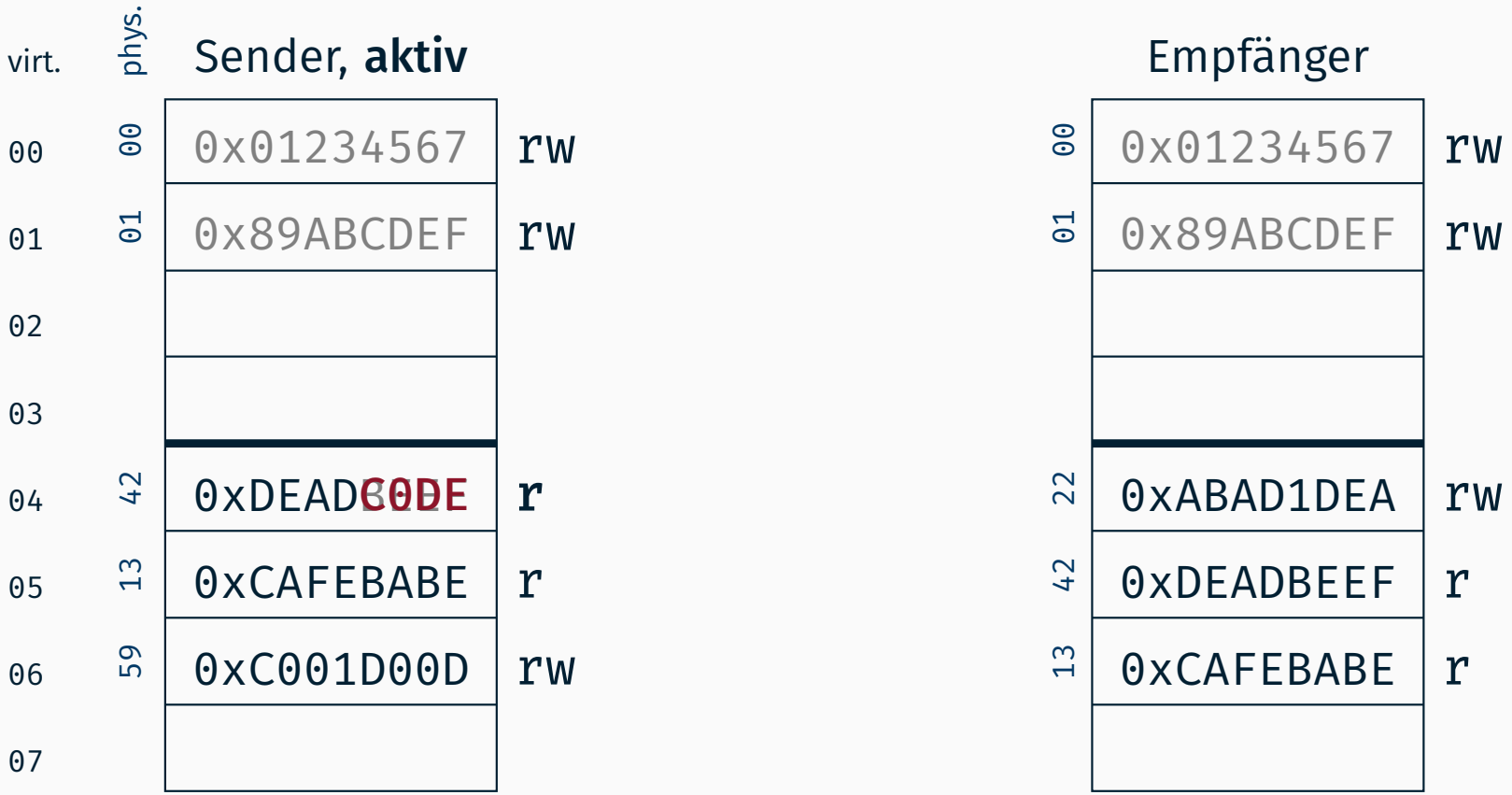
Copy-on-Write



Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58	59	...
Ref.	...	0	2	0	...	0	1	0	...	2	...	0	0	1	...

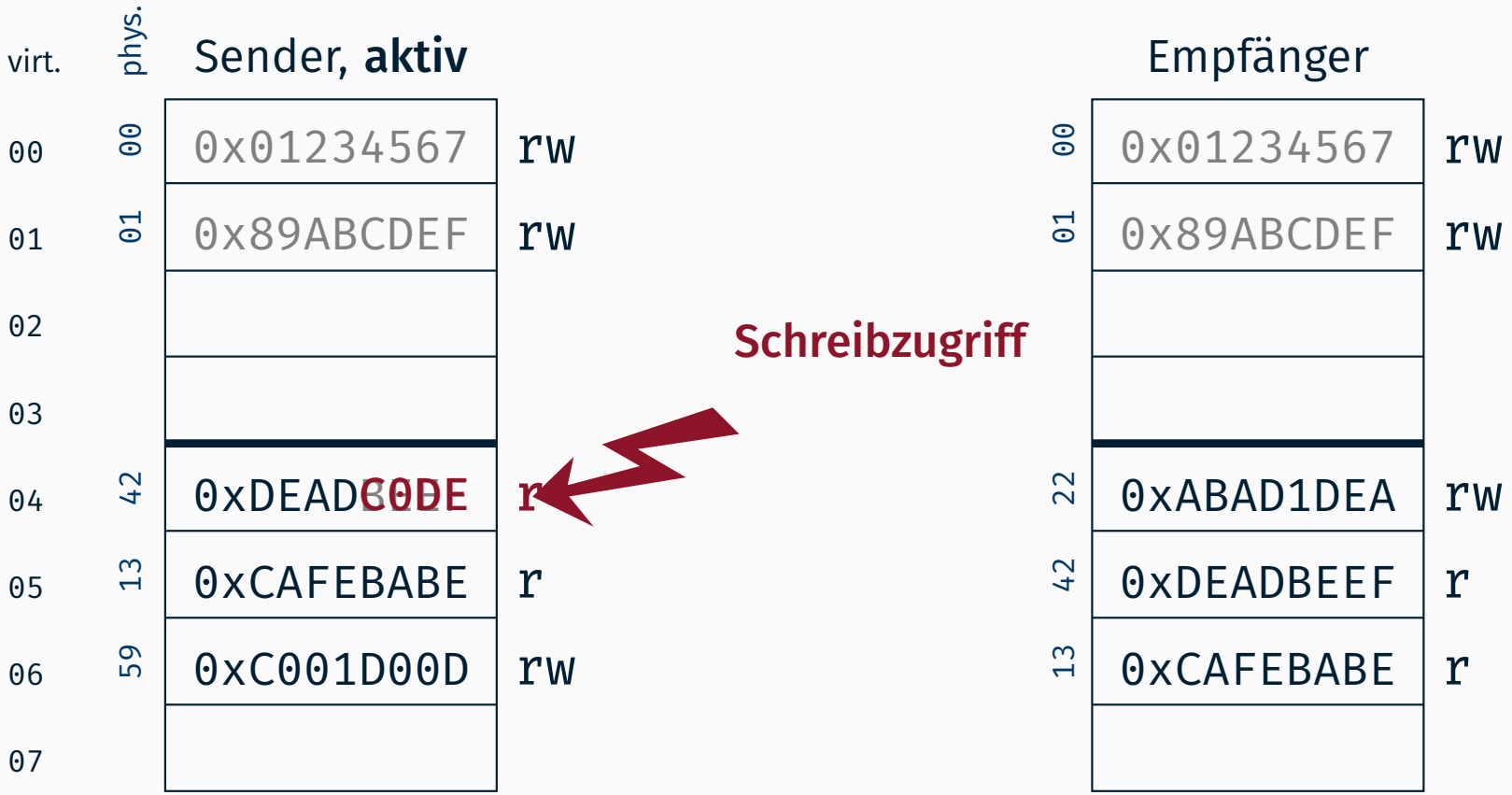
Copy-on-Write



Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58	59	...
Ref.	...	0	2	0	...	0	1	0	...	2	...	0	0	1	...

Copy-on-Write



Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58	59	...
Ref.	...	0	2	0	...	0	1	0	...	2	...	0	0	1	...

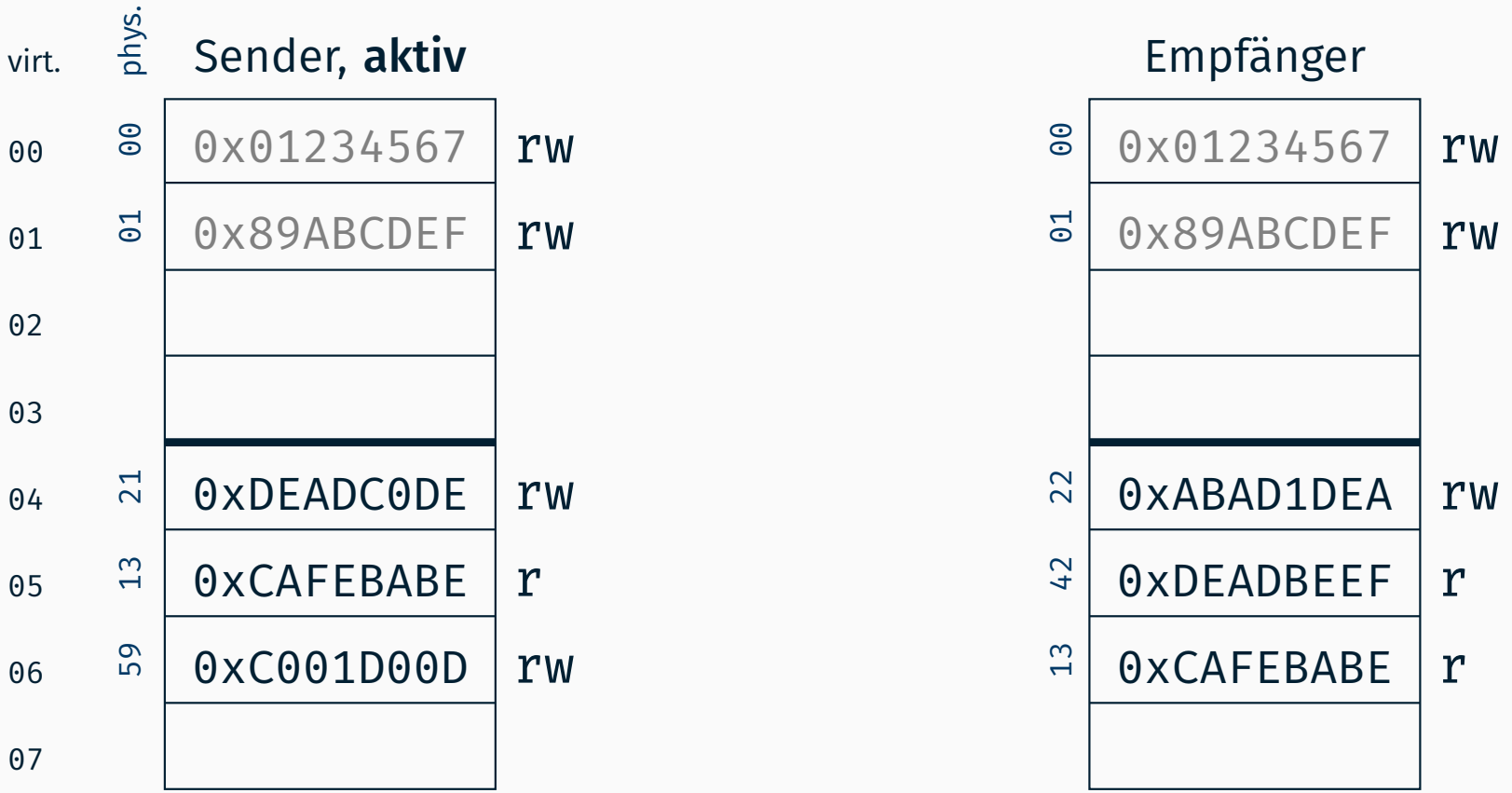
Copy-on-Write

virt.	phys.	Sender, **aktiv**		Empfänger	phys.	
00	00	0x01234567	rw	0x01234567	00	rw
01	01	0x89ABCDEF	rw	0x89ABCDEF	01	rw
02						
03						
04	21	0xDEADC0DE	rw **Inhalte kopieren**	0xABAD1DEA	22	rw
05	13	0xCAFEBABE	r	0xDEADBEEF	42	r
06	59	0xC001D00D	rw	0xCAFEBABE	13	r
07						

Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58		
Ref.	...	0	2	0	...	1	1	0	...	1	...	0	0	1	.

Copy-on-Write



Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58	59	...
Ref.	...	0	2	0	...	1	1	0	...	1	...	0	0	1	...

Seitenfehler-Behandlung

Umsetzung von Copy-on-Write mittels Pagefaults

- eigener Handler für `int 14/0xe`
- parallel zu Systemaufruf und Interrupts

Seitenfehler-Behandlung

Umsetzung von Copy-on-Write mittels Pagefaults

- eigener Handler für `int 14/0xe`
- parallel zu Systemaufruf und Interrupts

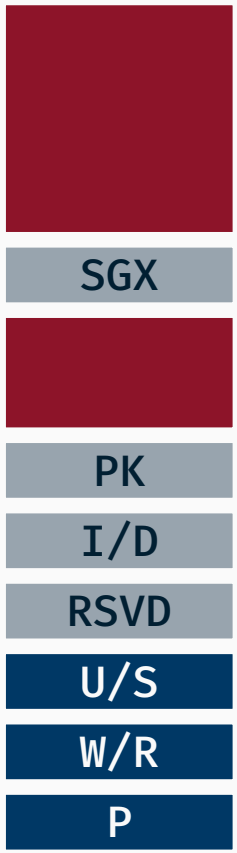
Informationen über Seitenfehler

- Adresse, die Fehler verursacht hat, liegt in `cr2`
- Fehlercode auf dem Kernel-Stack

Seitenfehler-Behandlung (cont.)

Fehlercode auf dem Kernel-Stack

Von Hardware nach dem Kontext abgelegt ($\rightarrow$ wegräumen!)



Reserviert

1 falls Software Guard Extensions

Reserviert

1 falls Memory Protection Key

Trap während Data Fetch (0) / Instruction Fetch (1)

1 falls reserviertes Bit in Pagingstruktur auf 1 gesetzt

Trap im Supervisormodus (0) / Usermodus (1)

Trap durch lesen (0) / schreiben (1)

Seite abwesend (0) / Schutzfehler (1)

Seitenfehler-Behandlung (cont.)

Fehlercode auf dem Kernel-Stack

Von Hardware nach dem Kontext abgelegt ($\rightarrow$ wegräumen!)

31 … 16		**Reserviert**
	SGX	1 falls Software Guard Extensions
14 … 6		**Reserviert**
5	PK	1 falls Memory Protection Key
4	I/D	Trap während Data Fetch (0) / Instruction Fetch (1)
3	RSVD	1 falls reserviertes Bit in Pagingstruktur auf 1 gesetzt
2	U/S	Trap im Supervisormodus (0) / **Usermodus (1)**
1	W/R	Trap durch lesen (0) / **schreiben (1)**
0	P	Seite abwesend (0) / **Schutzfehler (1)**

Copy-on-Write: P=1, W/R=1, U/S=1, PT-Eintrag read only

Neue Prozesse erzeugen

Wir implementieren `fork`

- Soll **Copy-on-Write** für gesamten User-Bereich nutzen (auch Stack!)
- Rückgabewert: ID des jeweils anderen

- Neuer Prozess $\rightarrow$ Wechsel von Ring 0 nach Ring 3
  $\Rightarrow$ Rückgabewert, Instruktions-/Stapel-Zeiger

Fragen?