Übung zu Betriebssystemtechnik

Nachrichtenaustausch und Copy-on-Write

11. Juni 2018

Andreas Ziegler
Bernhard Heinloth

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

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Einleitung

Ziel der Übung

Getrennte Adressräume überwinden

$\Rightarrow$ Nachrichtenaustausch

Ziel der Übung

Getrennte Adressräume überwinden

$\Rightarrow$ Nachrichtenaustausch

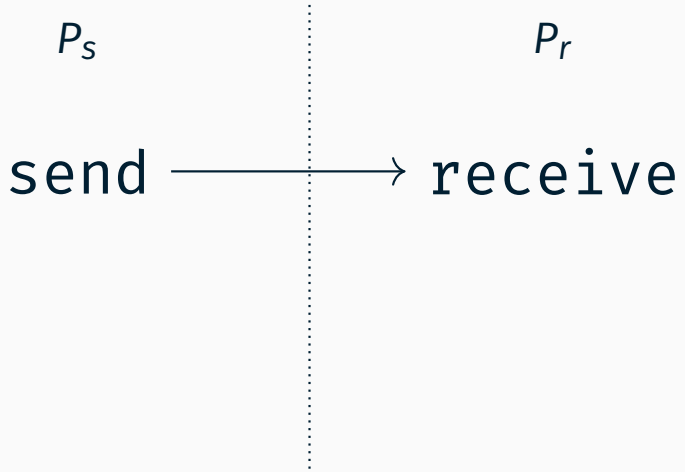
- `send, receive`

Ziel der Übung

Getrennte Adressräume überwinden

$\Rightarrow$ Nachrichtenaustausch

- `send, receive`

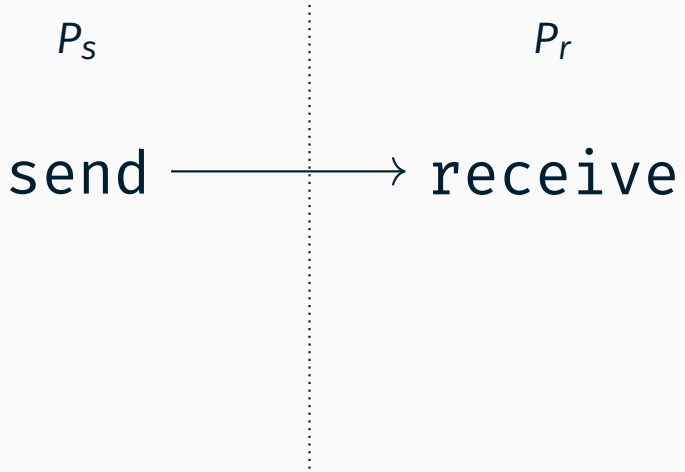


Ziel der Übung

Getrennte Adressräume überwinden

$\Rightarrow$ Nachrichtenaustausch

- `send, receive`
- **`send, receive, reply`**

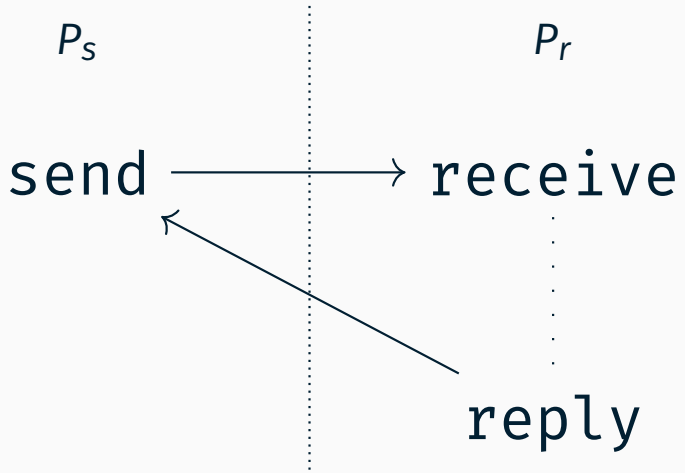


Ziel der Übung

Getrennte Adressräume überwinden

$\Rightarrow$ Nachrichtenaustausch

- `send, receive`
- **`send, receive, reply`**



Nachrichten

Vorteil von `send, receive, reply`:

$\Rightarrow$ Keine Timer nötig, um DoS auf Server-Seite zu verhindern!

Nachrichten

Vorteil von `send, receive, reply`:

$\Rightarrow$ Keine Timer nötig, um DoS auf Server-Seite zu verhindern!

synchron/asynchron vs. blockierend/nichtblockierend?

Nachrichten

Vorteil von `send, receive, reply`:

$\Rightarrow$ Keine Timer nötig, um DoS auf Server-Seite zu verhindern!

synchron/asynchron vs. blockierend/nichtblockierend?

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

Umsetzung

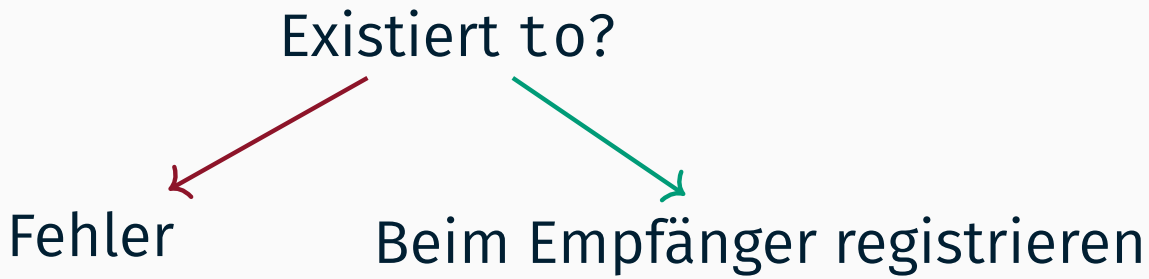
Zustandsdiagramme `send`, `receive`, `reply`

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

Existiert `to`?

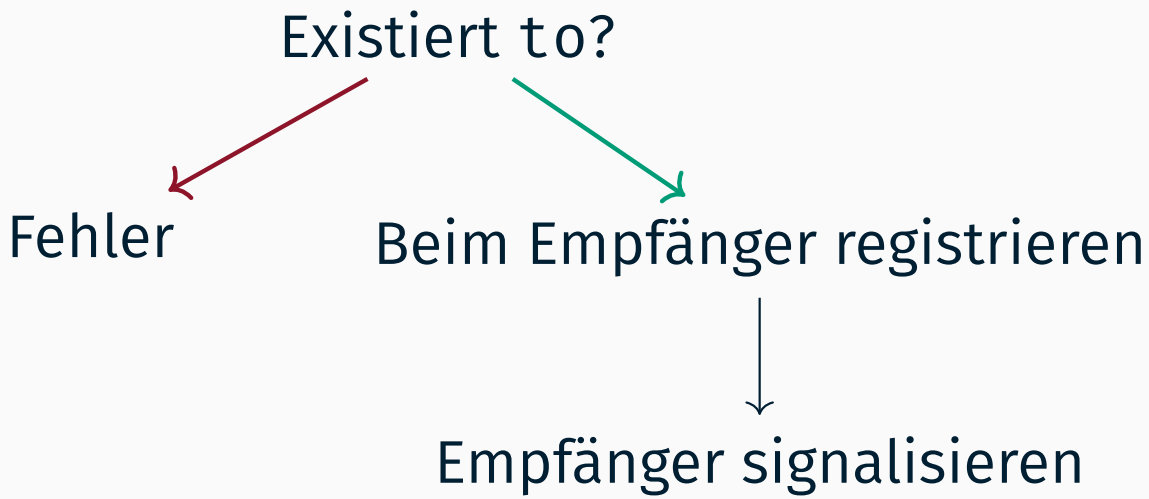
Zustandsdiagramme `send`, `receive`, `reply`

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



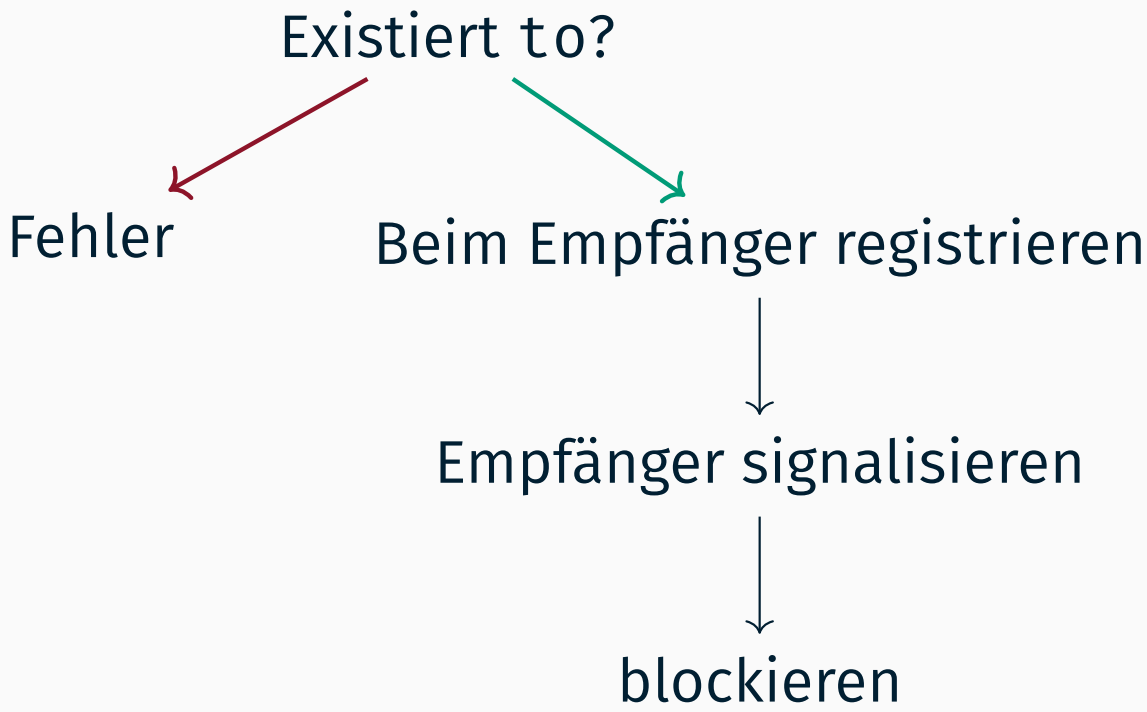
Zustandsdiagramme `send`, `receive`, `reply`

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



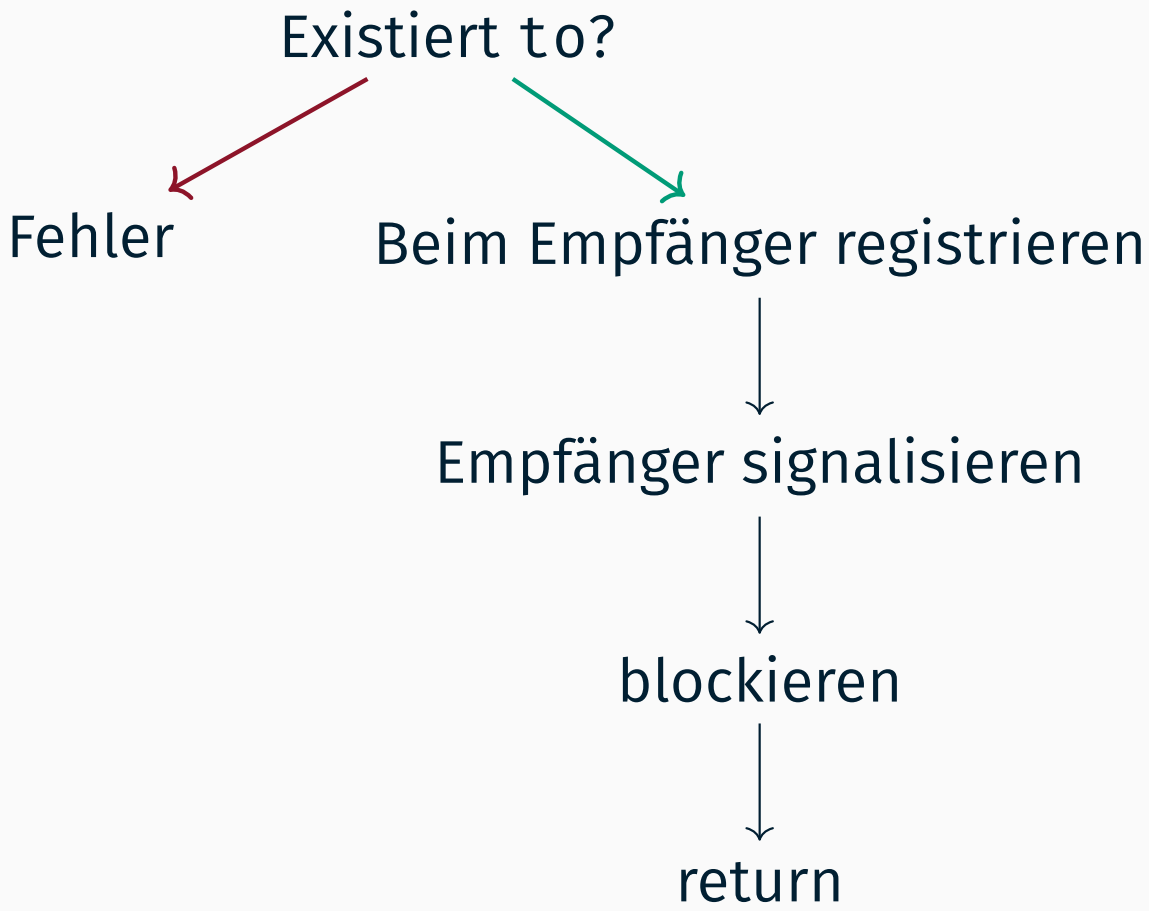
Zustandsdiagramme `send`, `receive`, `reply`

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



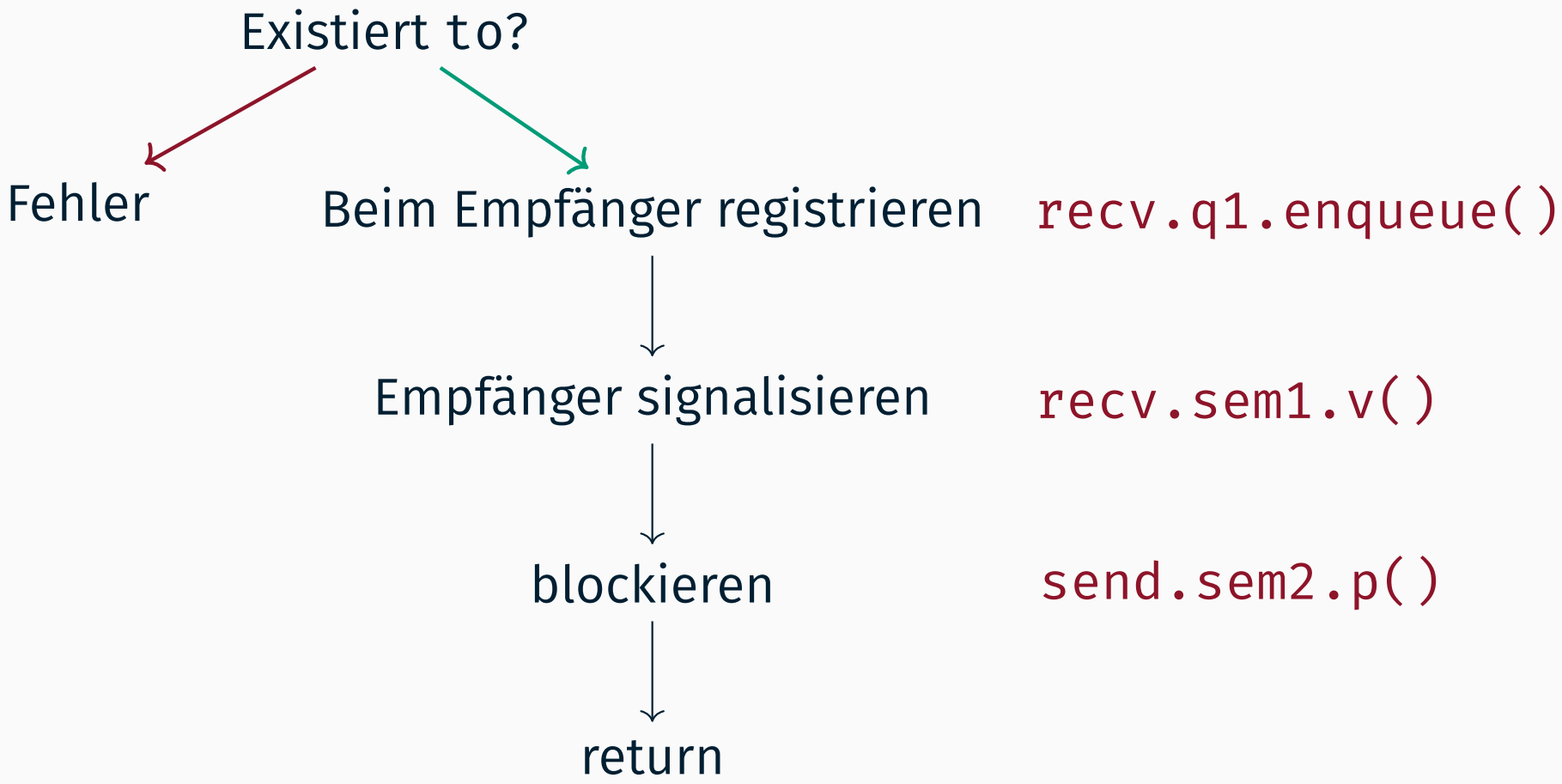
Zustandsdiagramme `send`, `receive`, `reply`

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



Zustandsdiagramme `send`, `receive`, `reply`

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



Zustandsdiagramme `send, receive, reply`

`recv(data, len)`

Auf Signal eines Senders warten

Zustandsdiagramme `send, receive, reply`

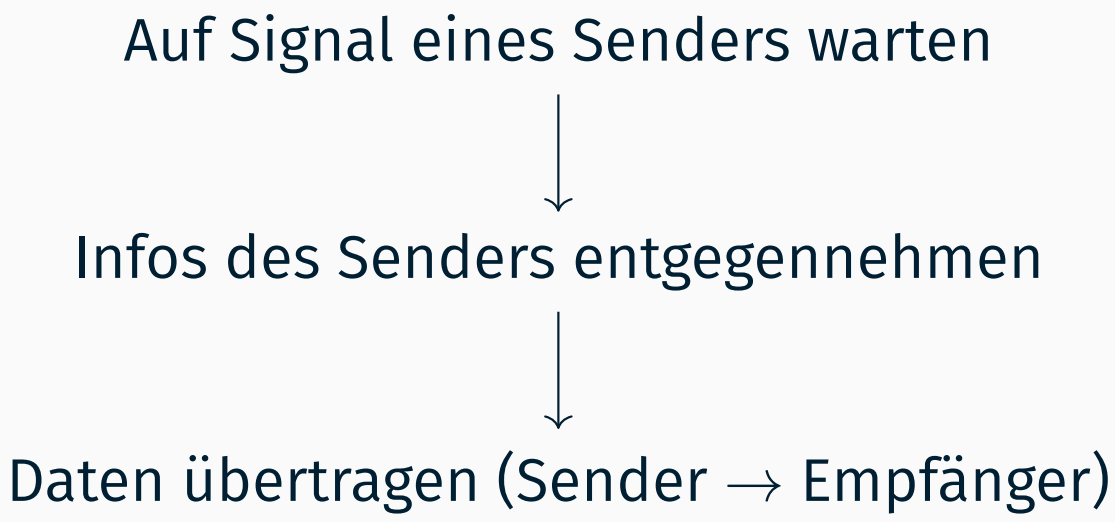
`recv(data, len)`

Auf Signal eines Senders warten

Infos des Senders entgegennehmen

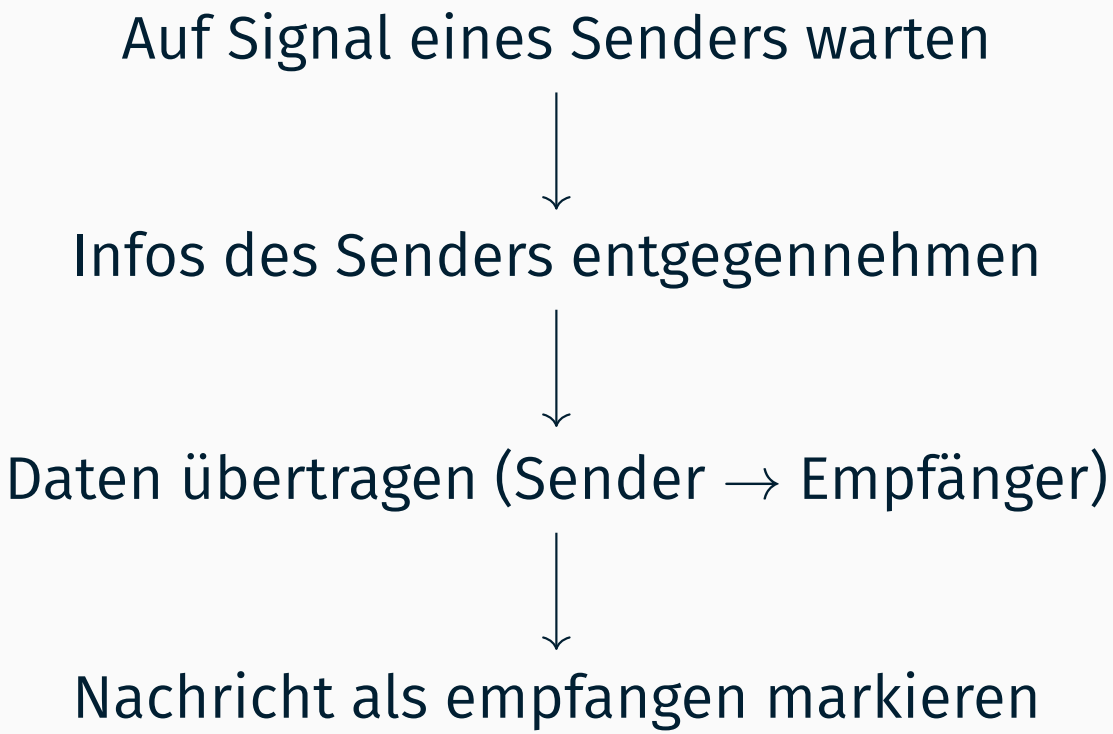
Zustandsdiagramme `send, receive, reply`

`recv(data, len)`



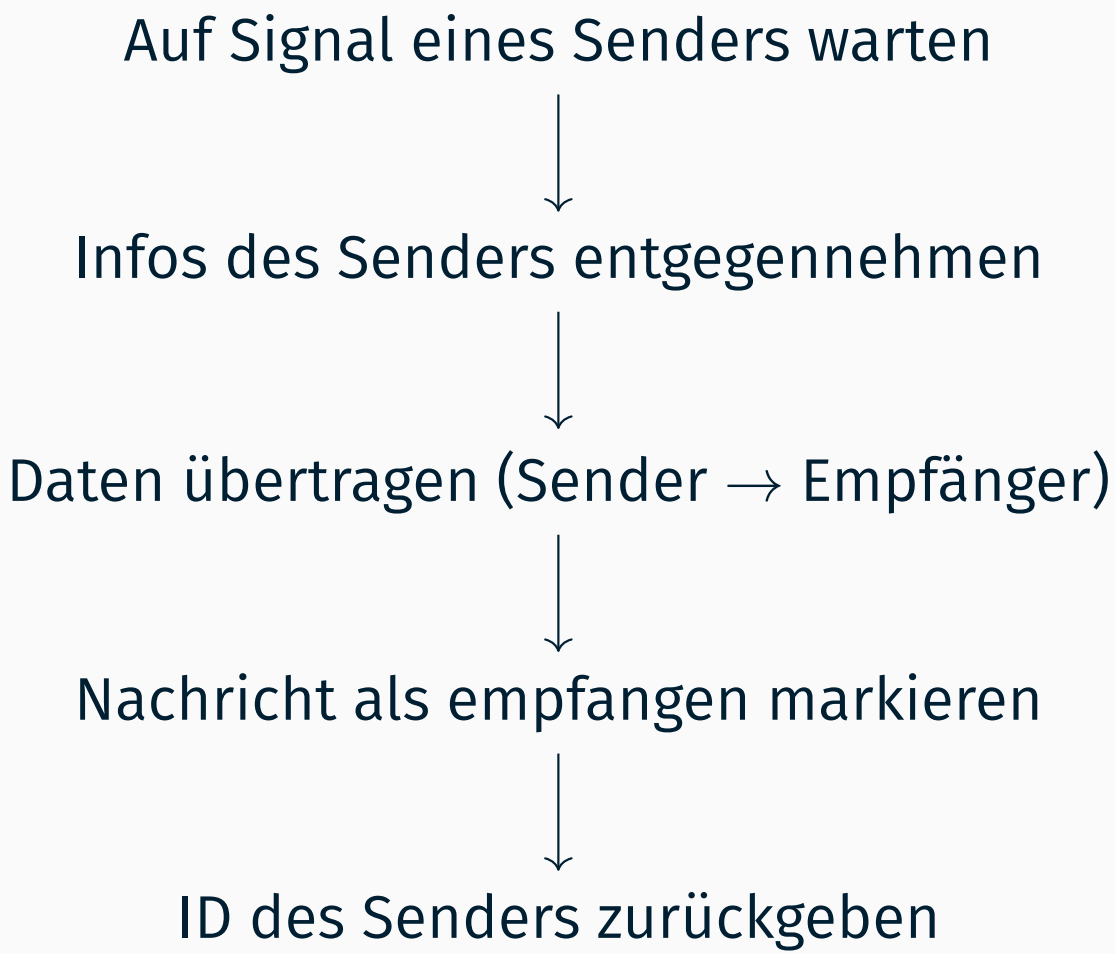
Zustandsdiagramme `send, receive, reply`

`recv(data, len)`



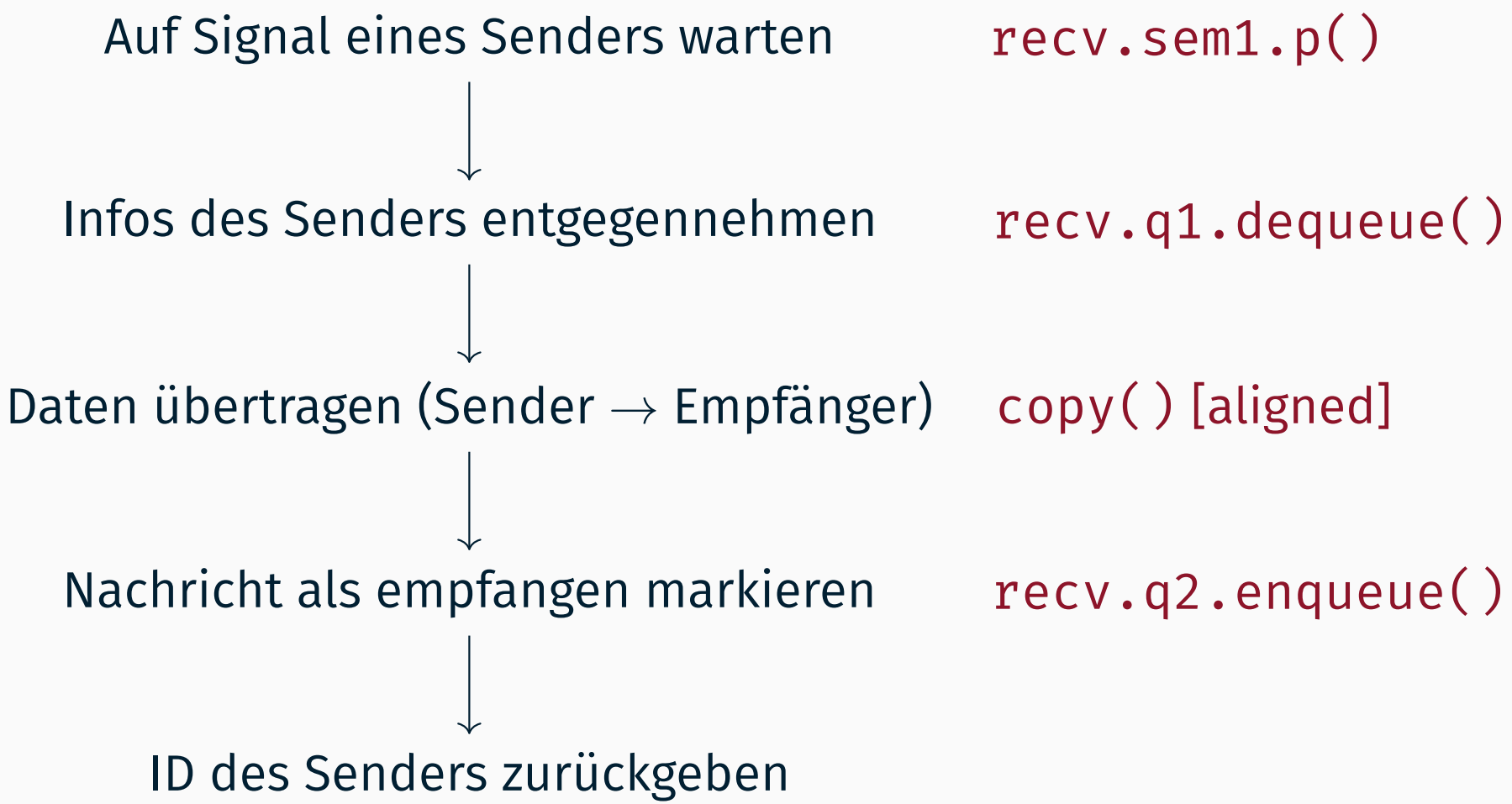
Zustandsdiagramme `send, receive, reply`

`recv(data, len)`



Zustandsdiagramme `send`, `receive`, `reply`

`recv(data, len)`



Zustandsdiagramme `send`, `receive`, `reply`

`reply(to, data, len)`

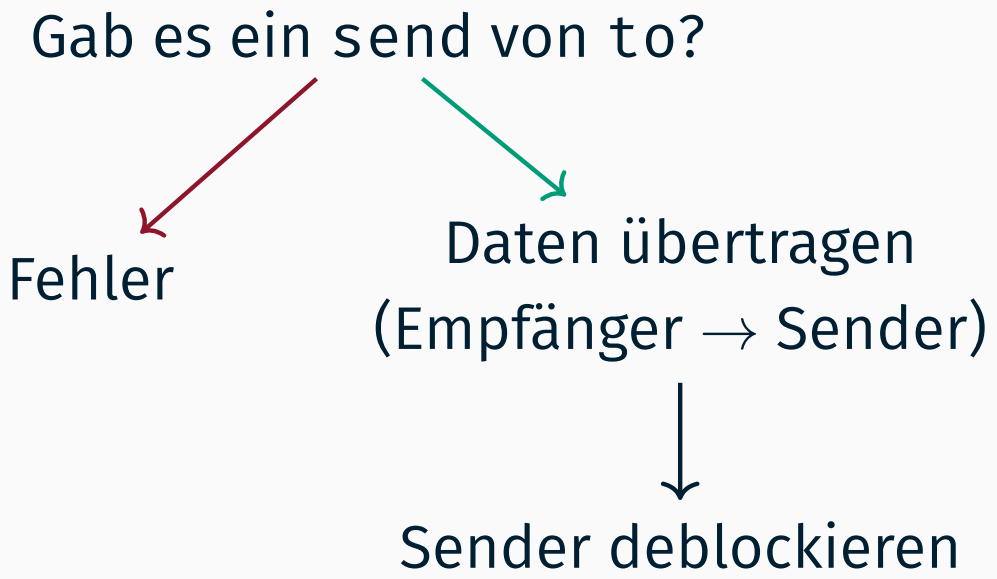
- Gab es ein `send` von `to`?

Zustandsdiagramme `send`, `receive`, `reply`

`reply(to, data, len)`

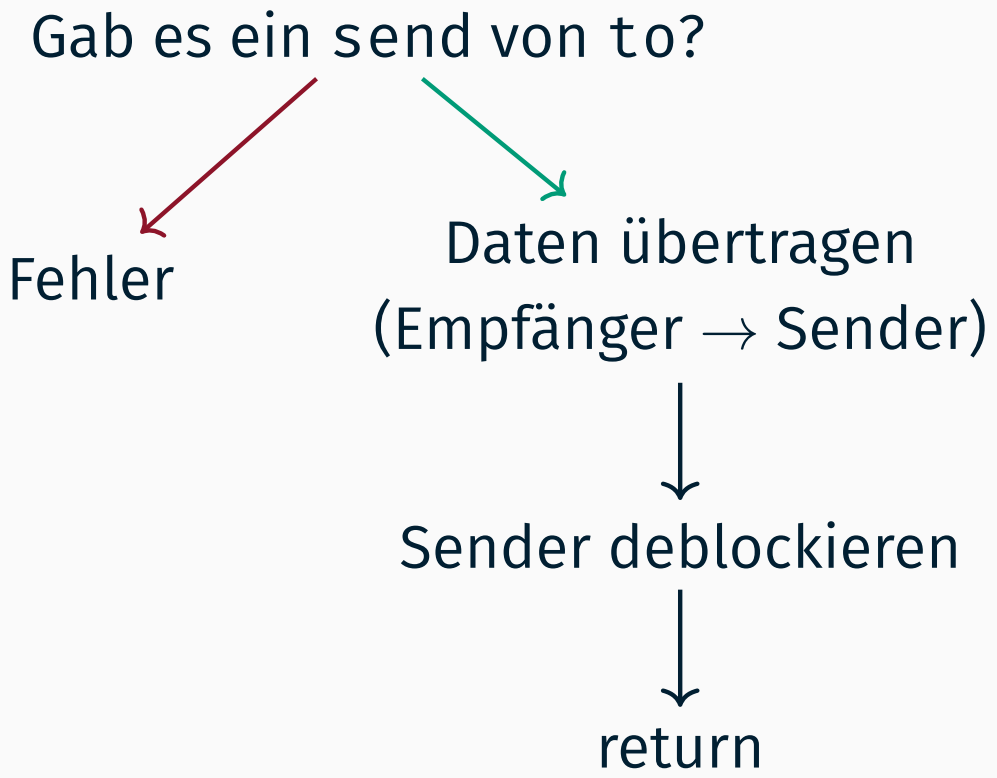
Zustandsdiagramme `send`, `receive`, `reply`

`reply(to, data, len)`



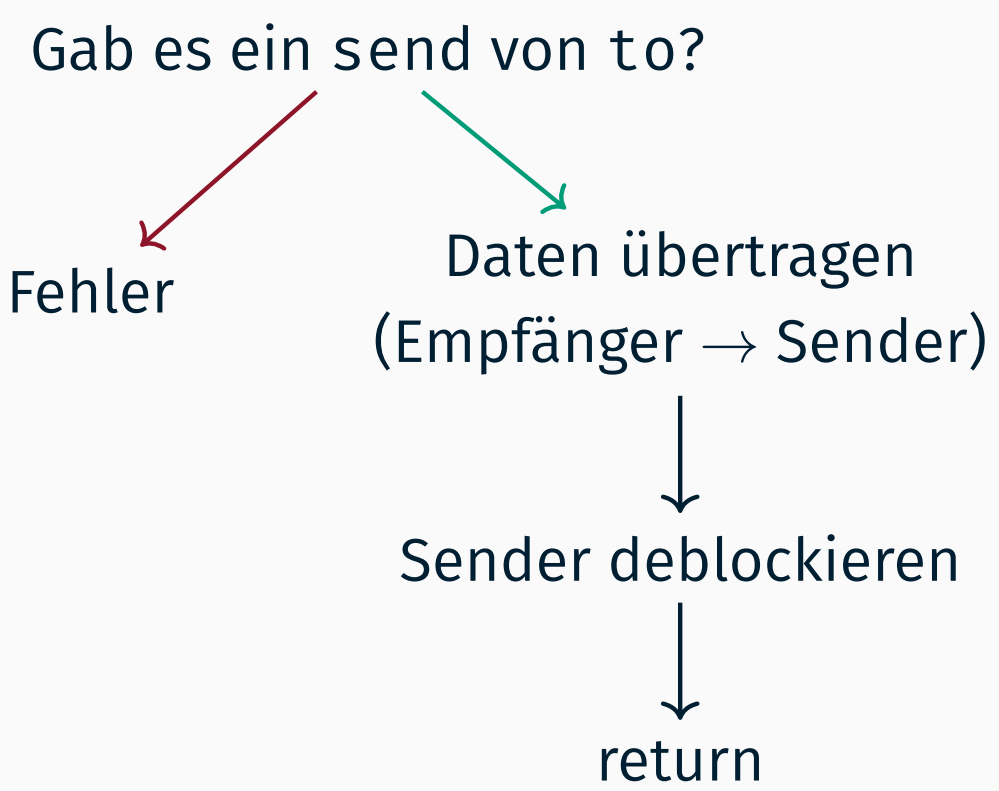
Zustandsdiagramme `send`, `receive`, `reply`

`reply(to, data, len)`



Zustandsdiagramme `send`, `receive`, `reply`

`reply(to, data, len)`



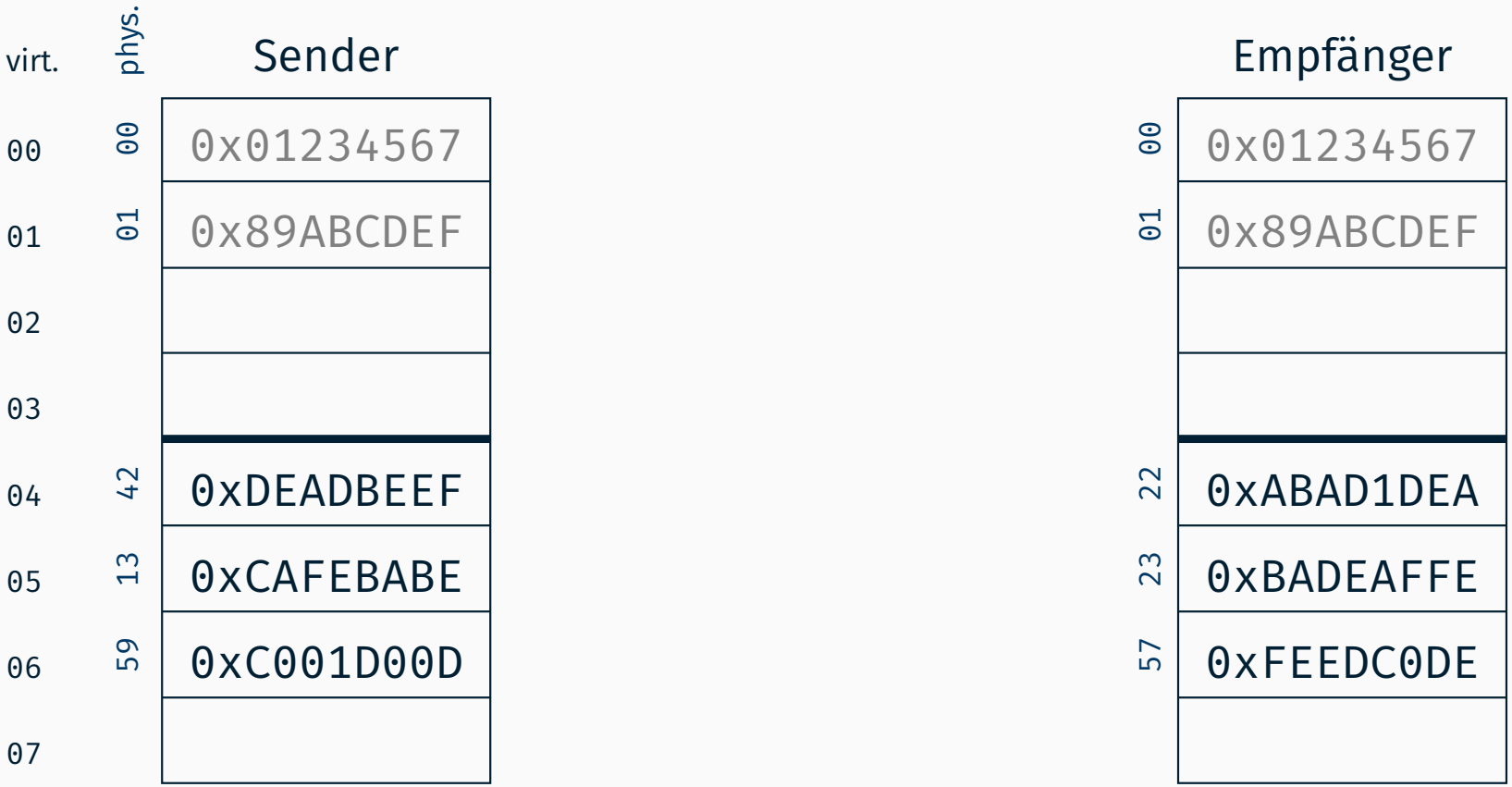
Nachricht mit ID in Queue 2?

`copy()`

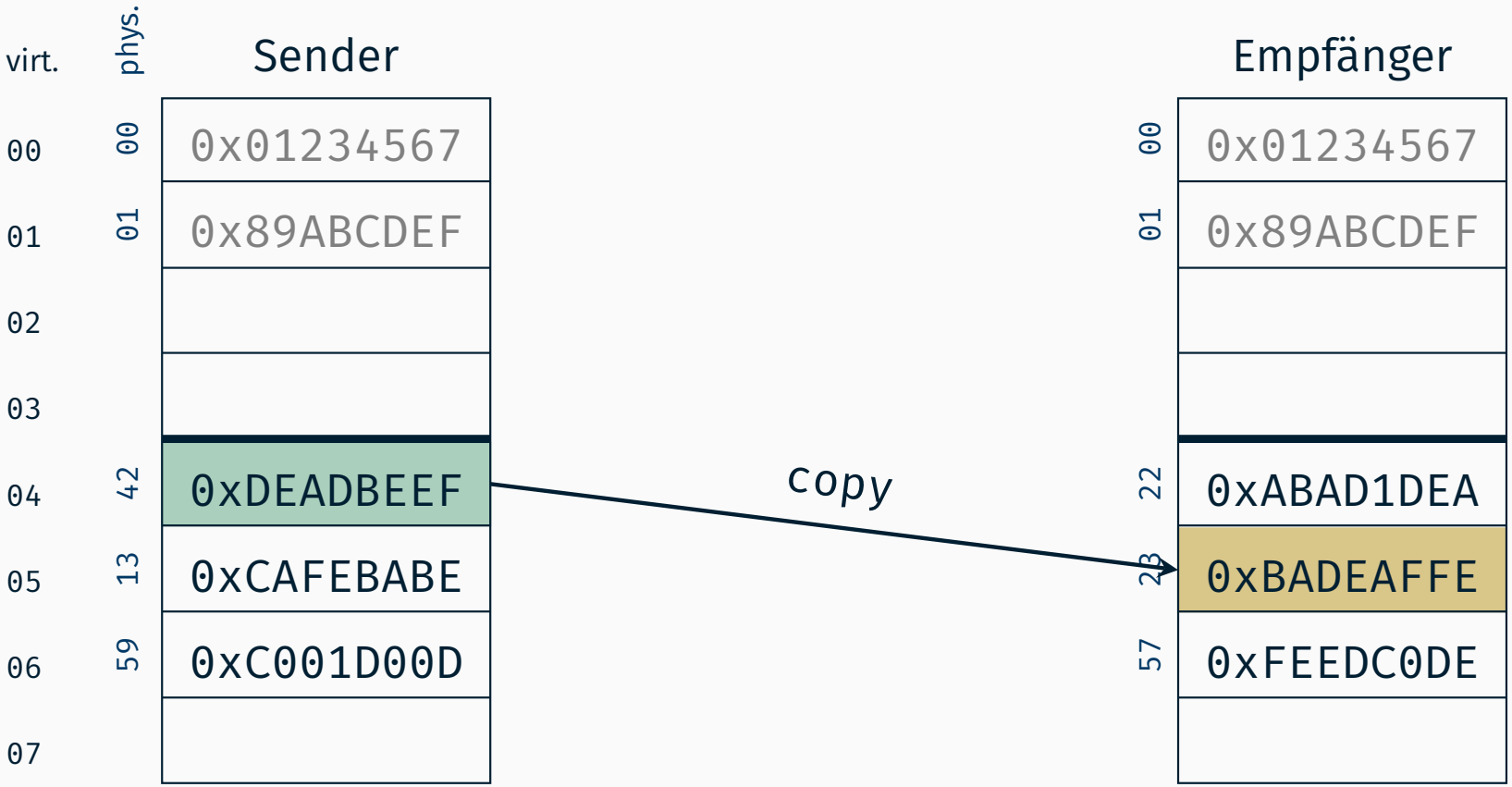
`send.sem2.v()`

Kopieren zwischen Adressräumen

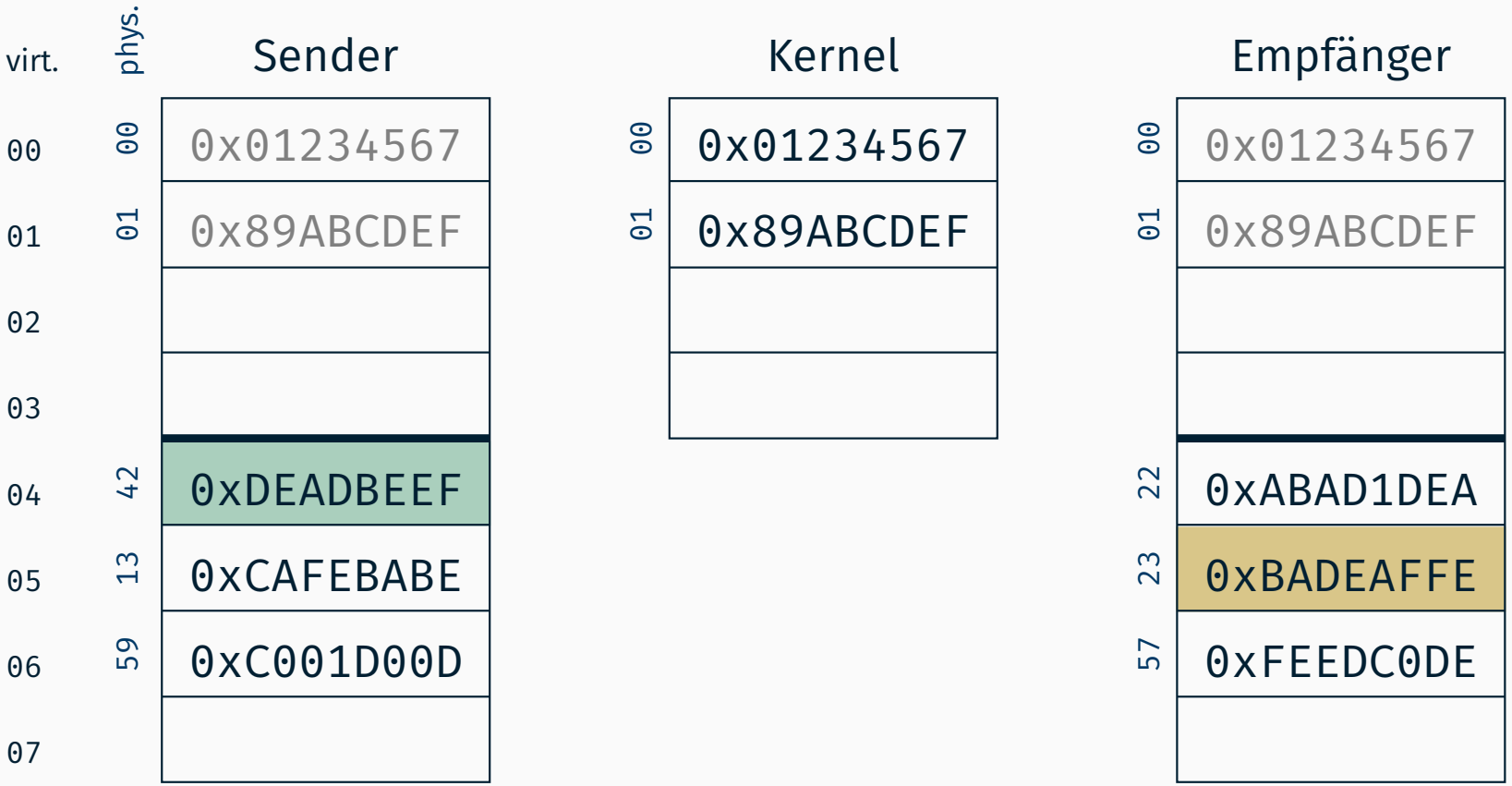
Kopieren zwischen Adressräumen



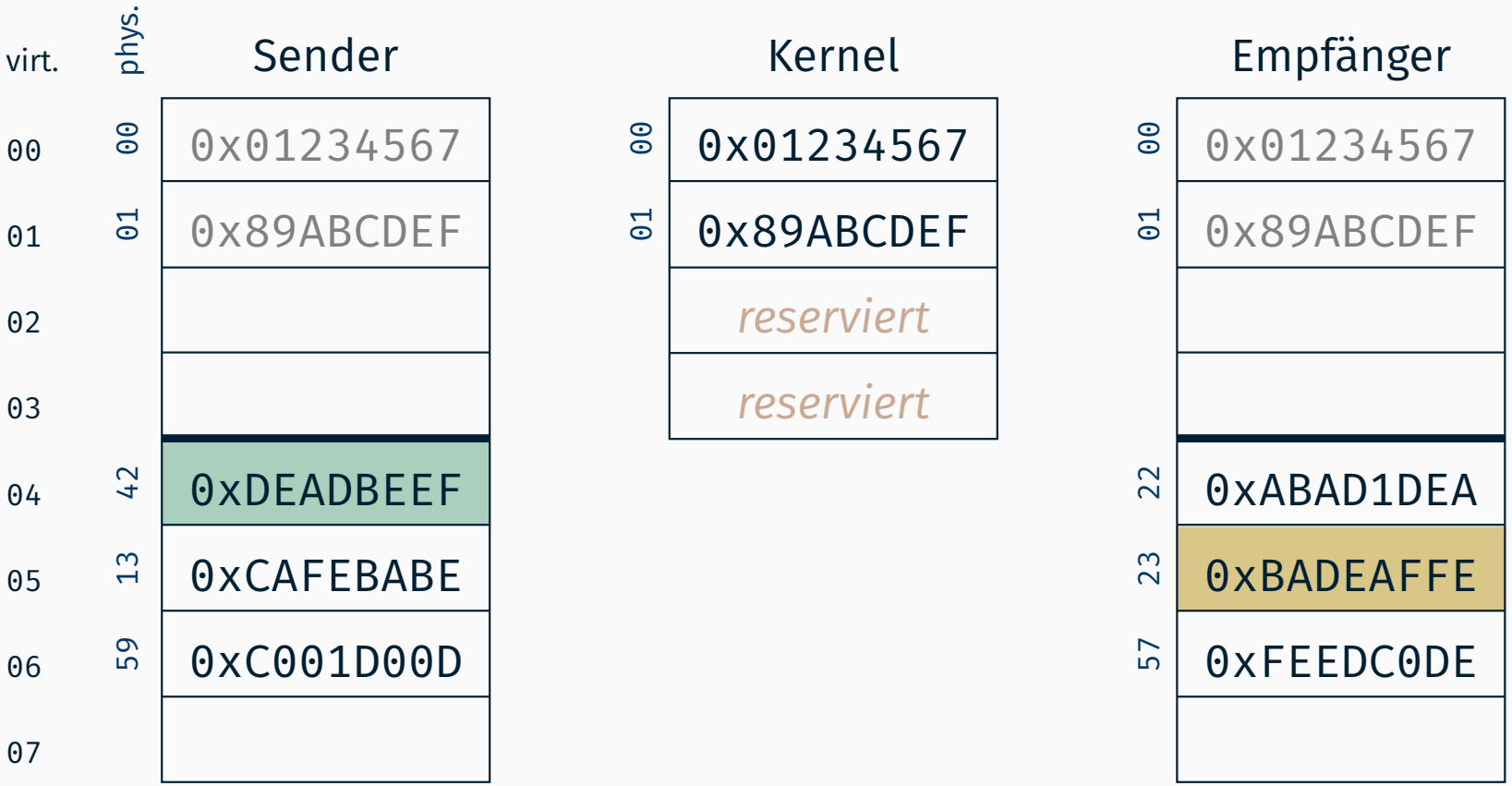
Kopieren zwischen Adressräumen



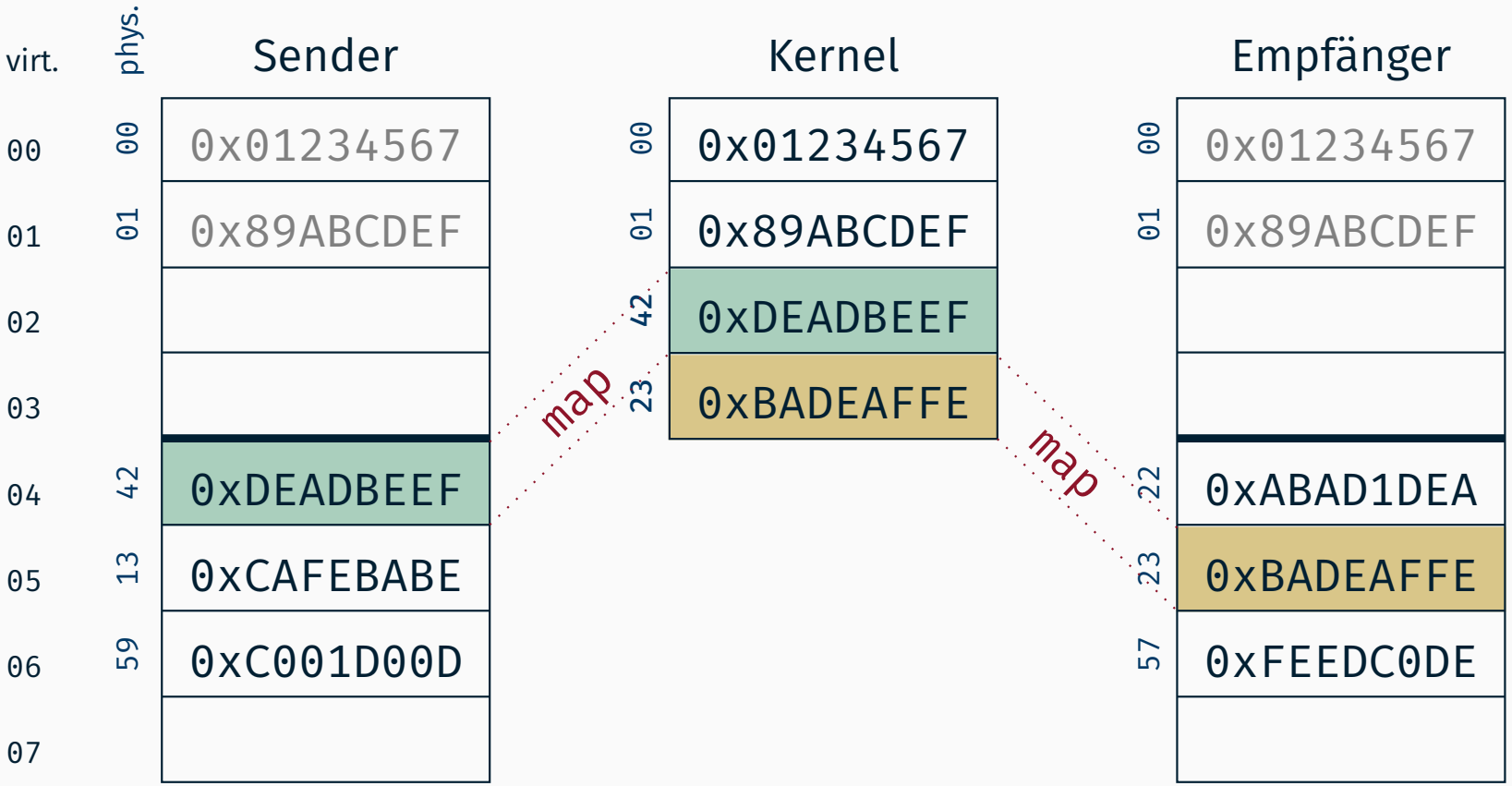
Kopieren zwischen Adressräumen



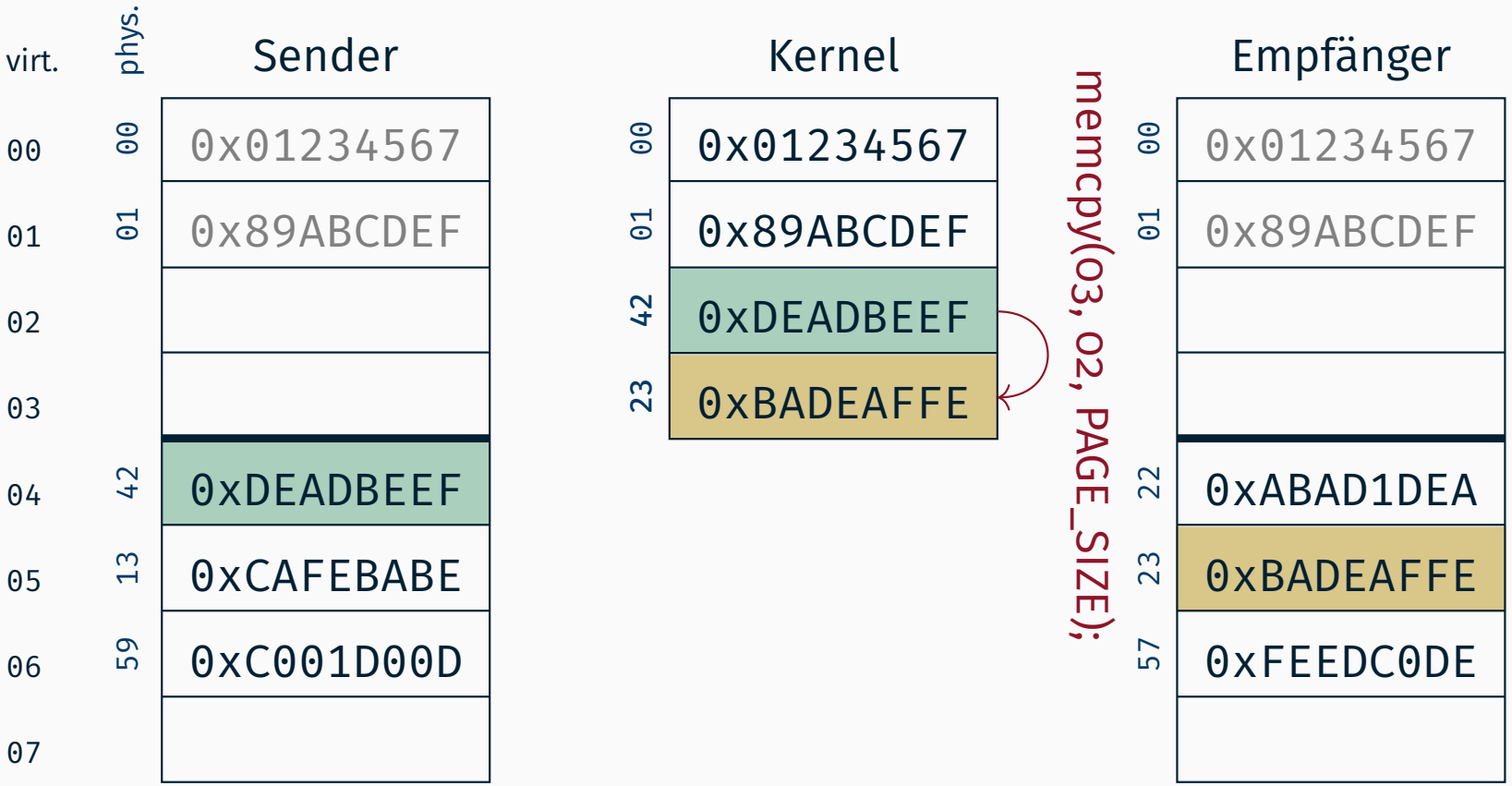
Kopieren zwischen Adressräumen



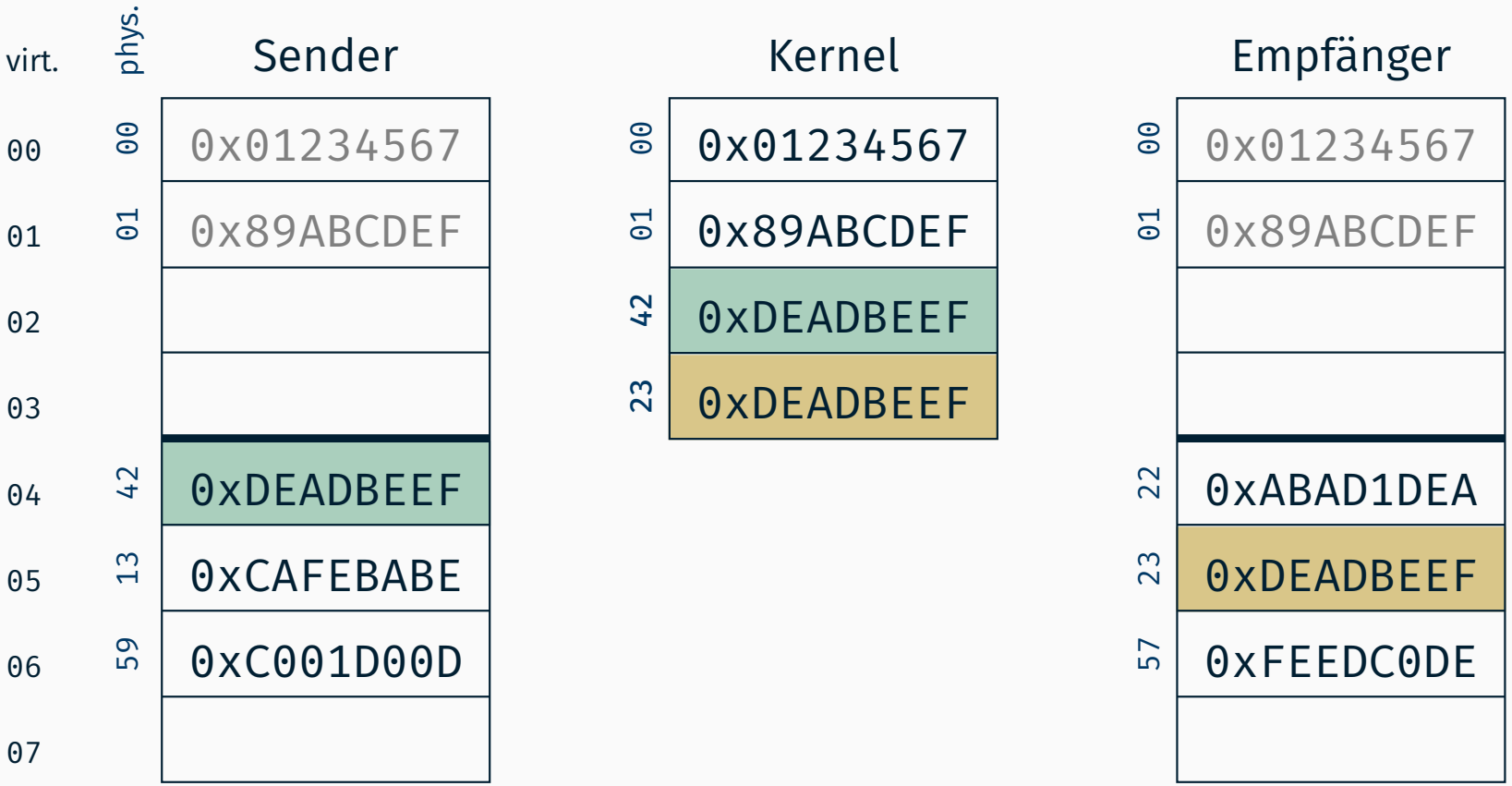
Kopieren zwischen Adressräumen



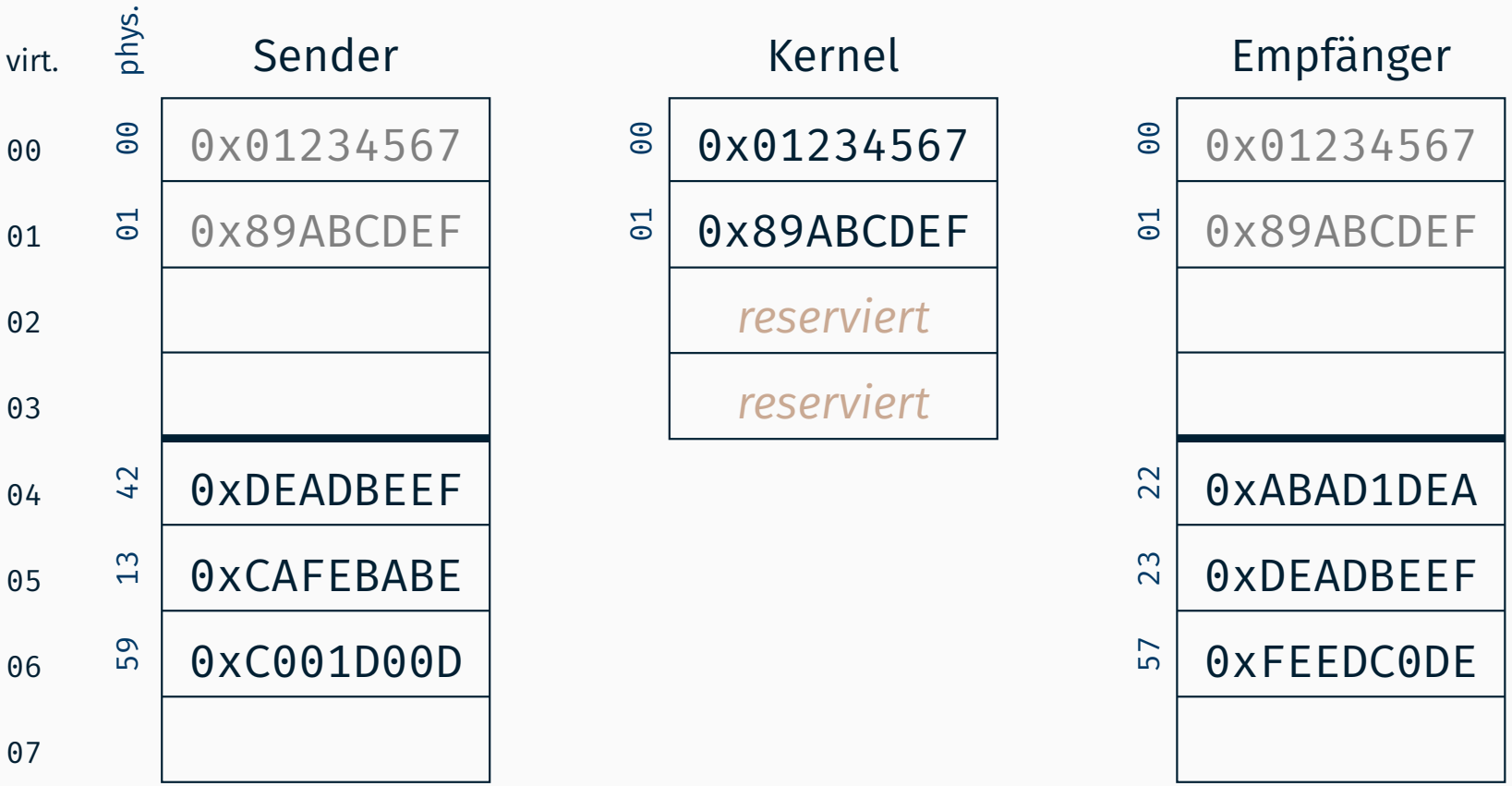
Kopieren zwischen Adressräumen



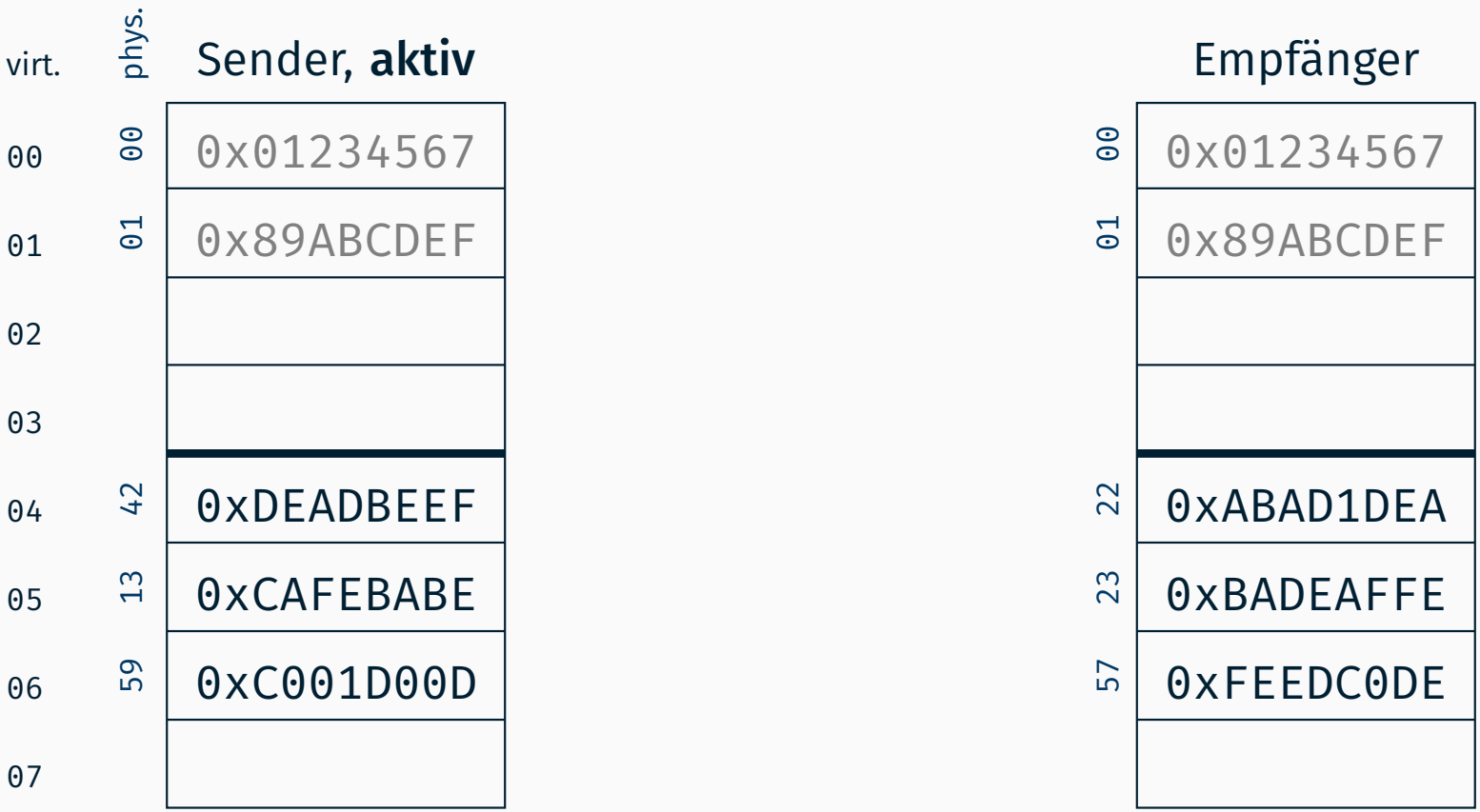
Kopieren zwischen Adressräumen



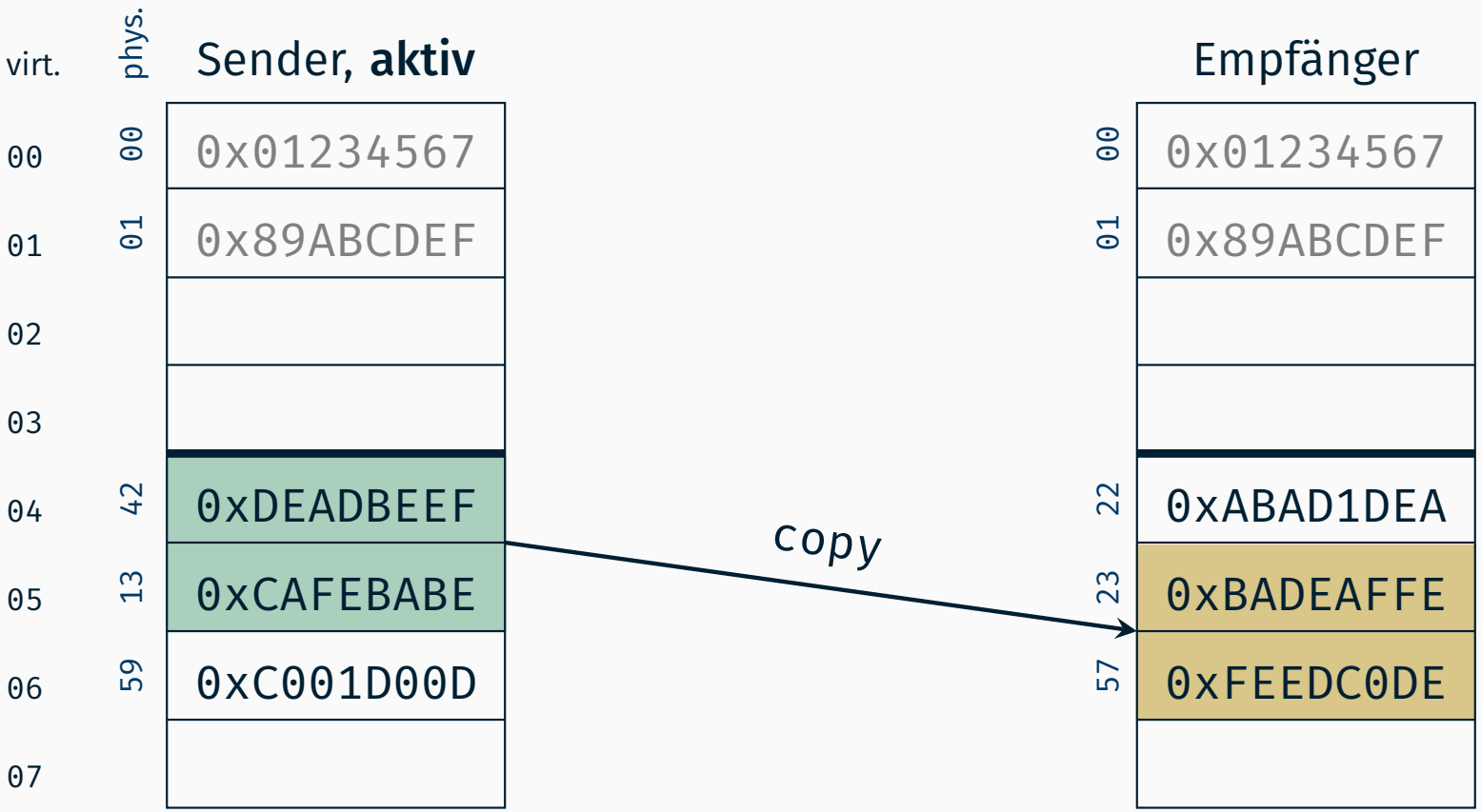
Kopieren zwischen Adressräumen



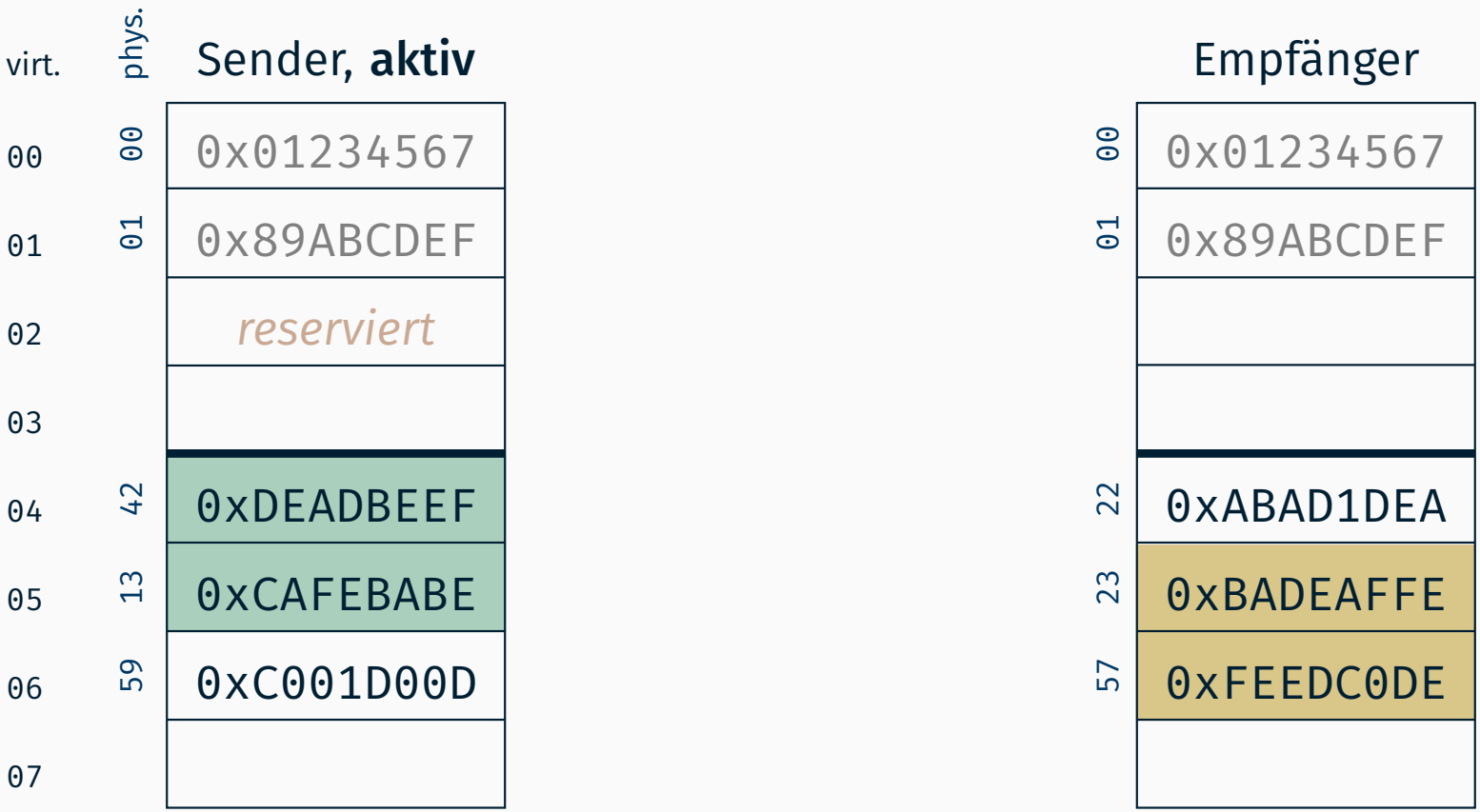
Kopieren zwischen Adressräumen (optimiert)



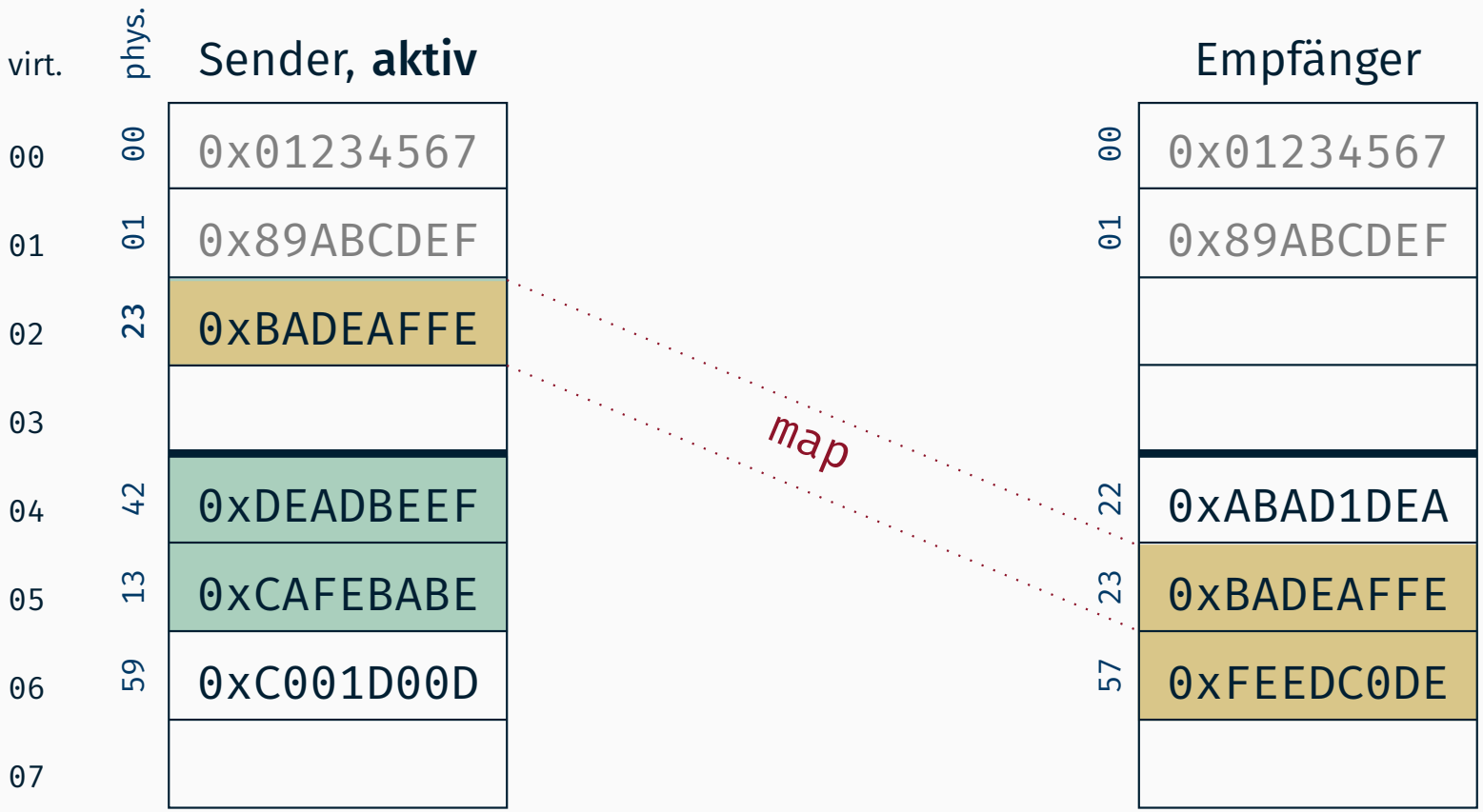
Kopieren zwischen Adressräumen (optimiert)



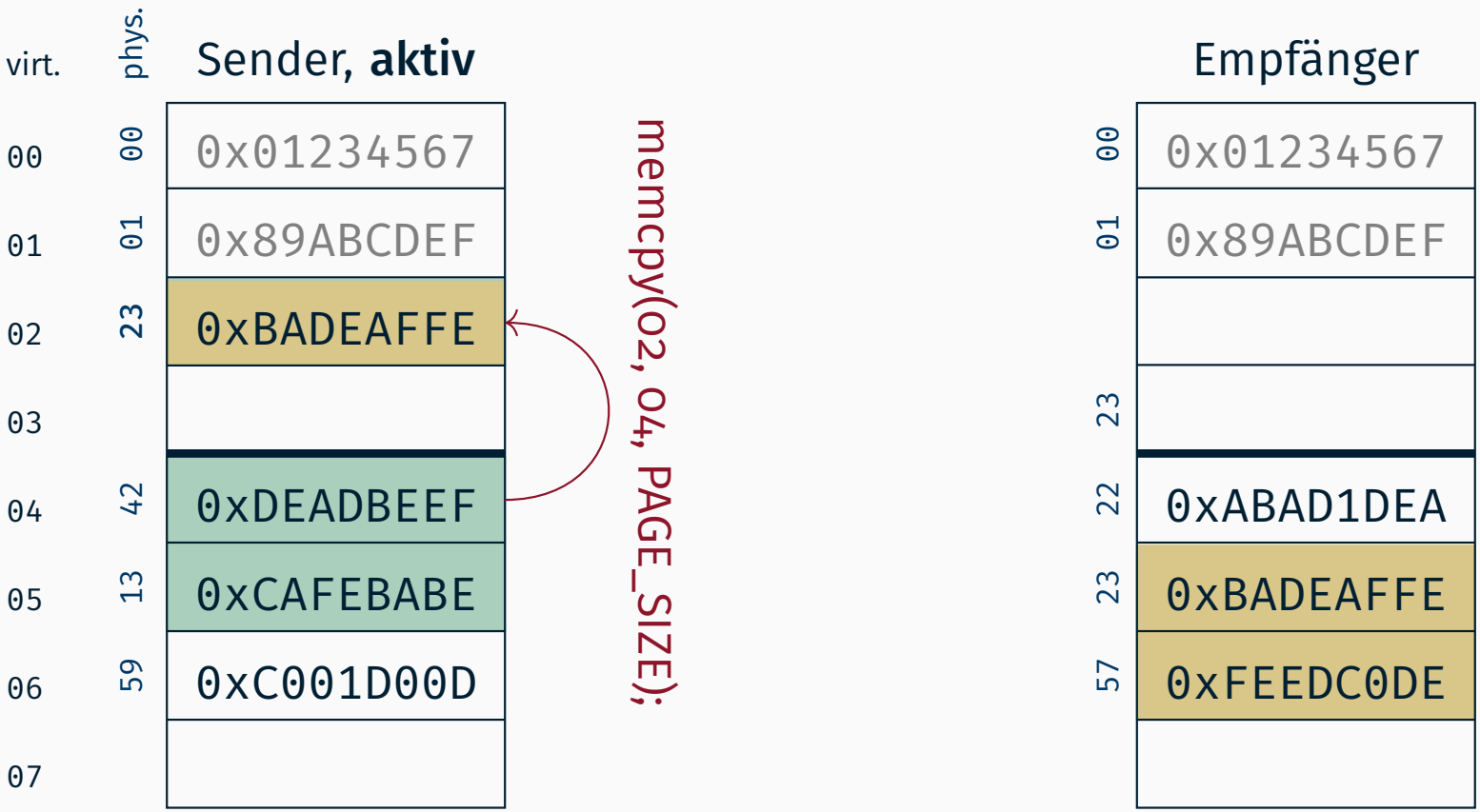
Kopieren zwischen Adressräumen (optimiert)



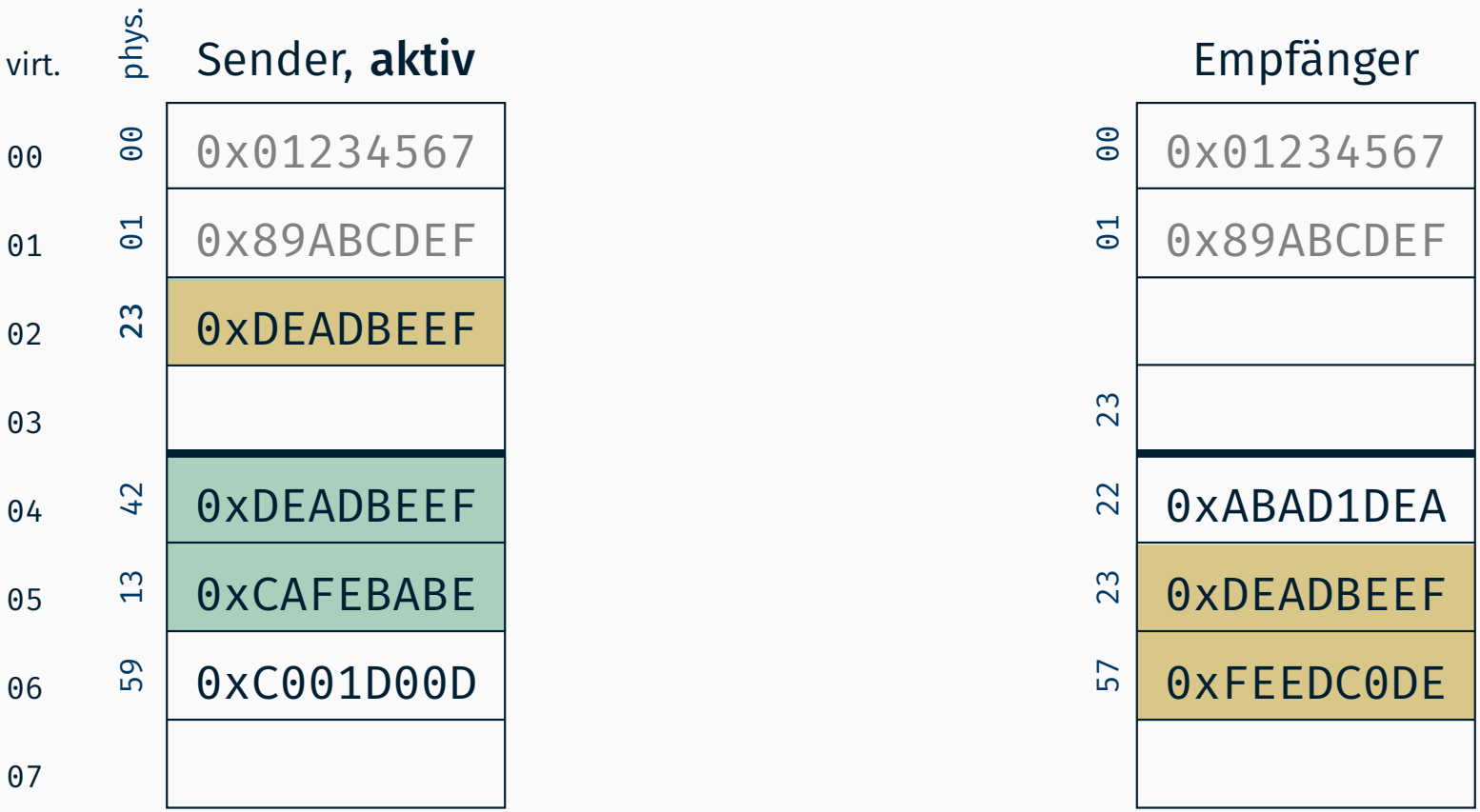
Kopieren zwischen Adressräumen (optimiert)



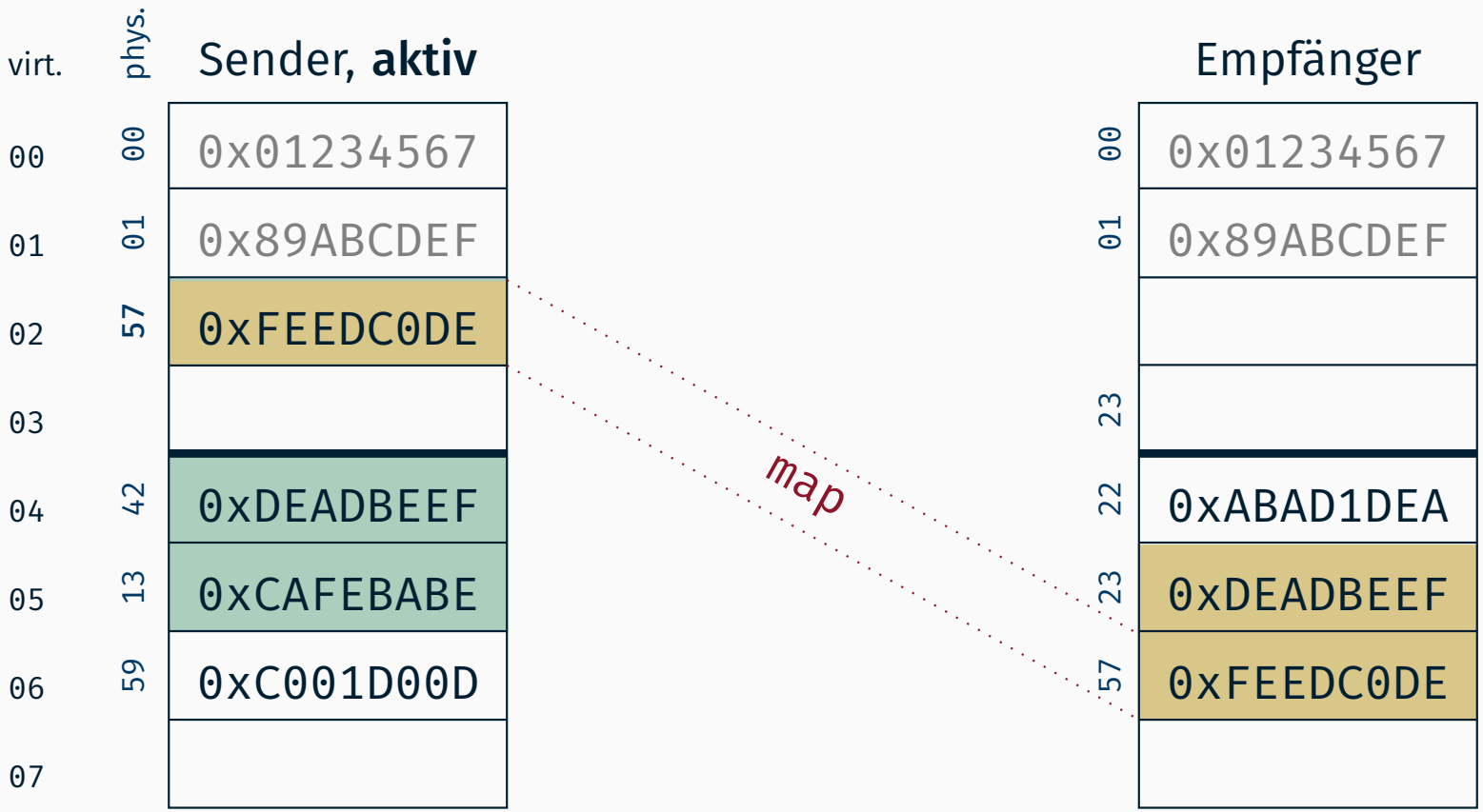
Kopieren zwischen Adressräumen (optimiert)



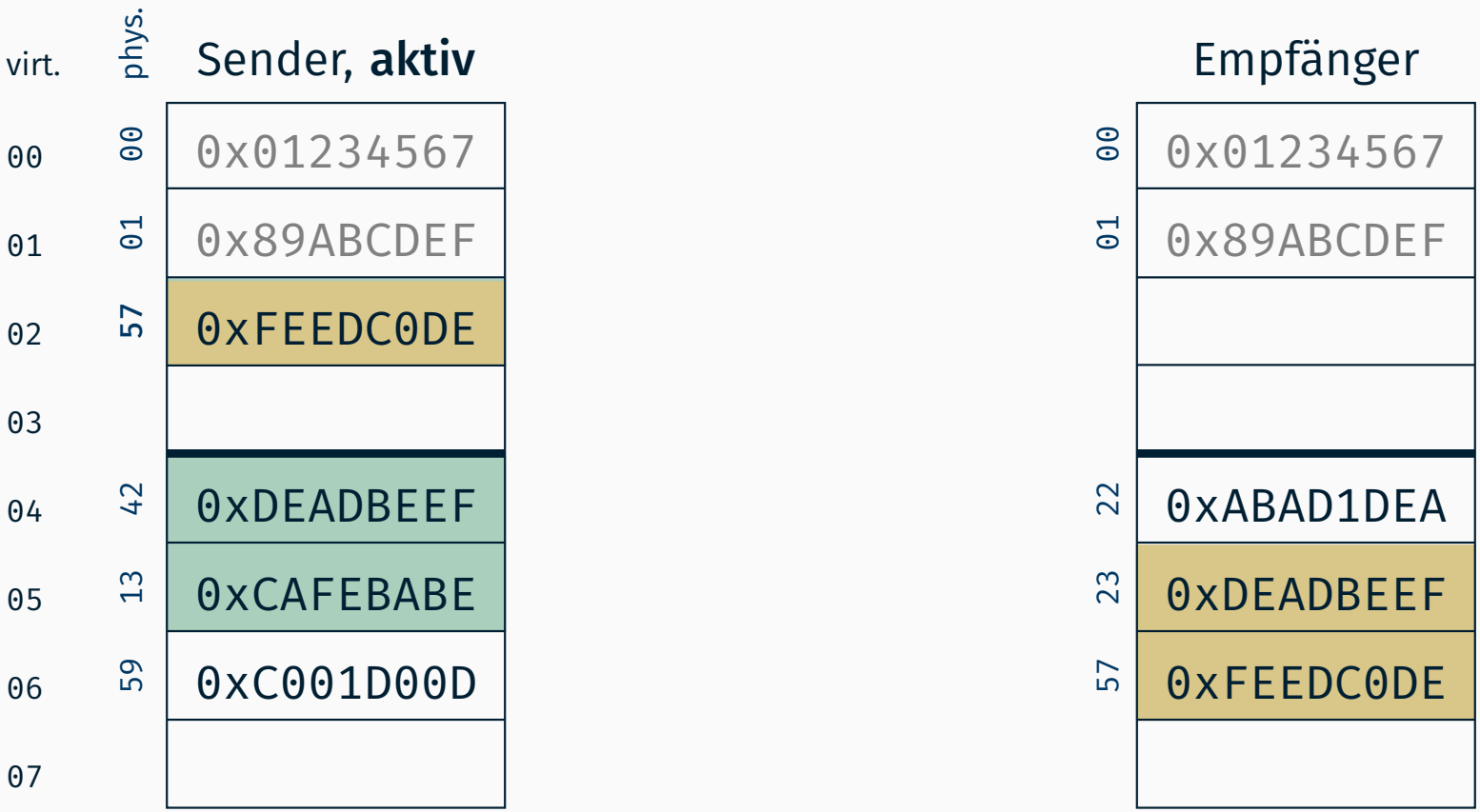
Kopieren zwischen Adressräumen (optimiert)



Kopieren zwischen Adressräumen (optimiert)



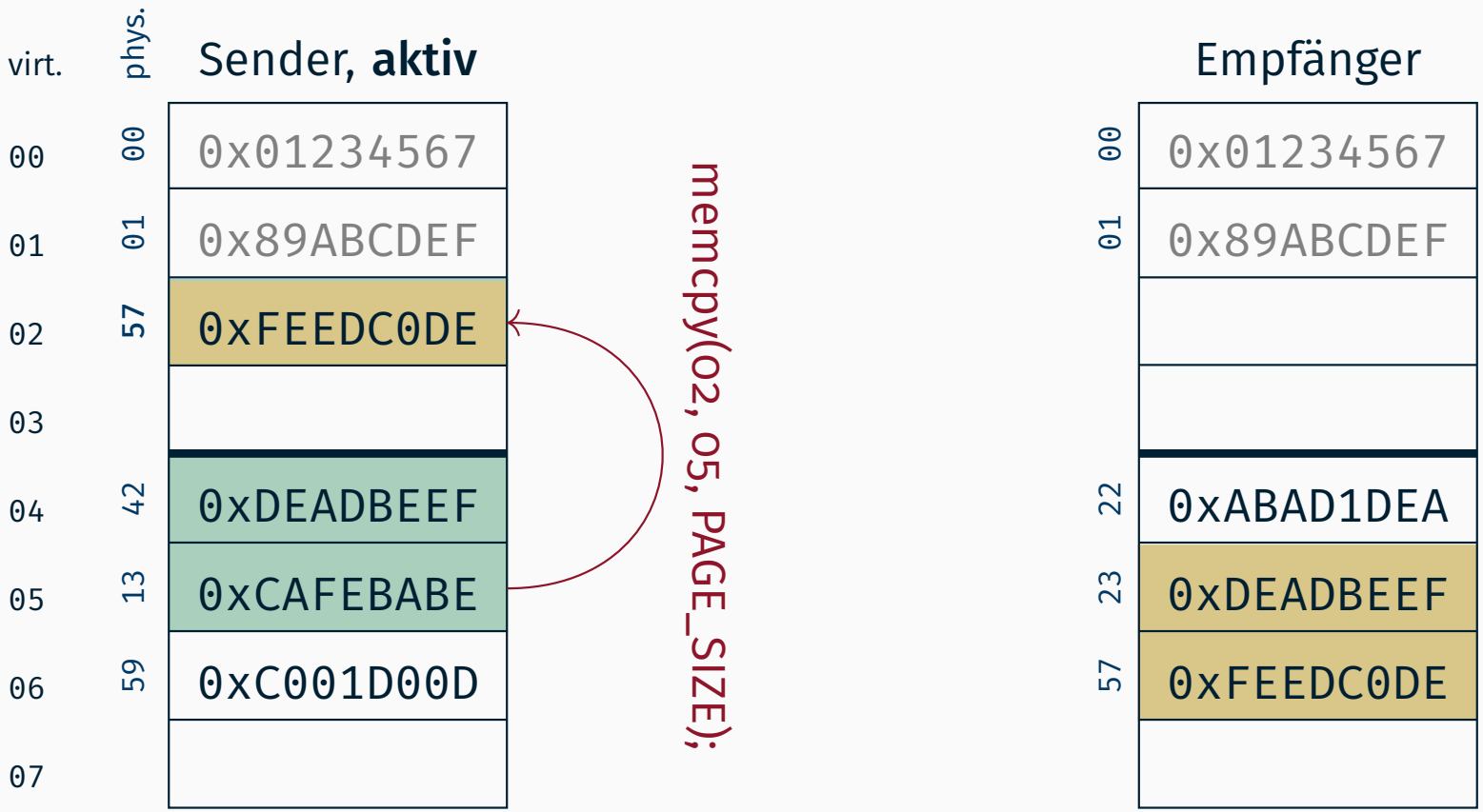
Kopieren zwischen Adressräumen (optimiert)



Achtung: Nach Änderung des Mappings TLB invalidieren!

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

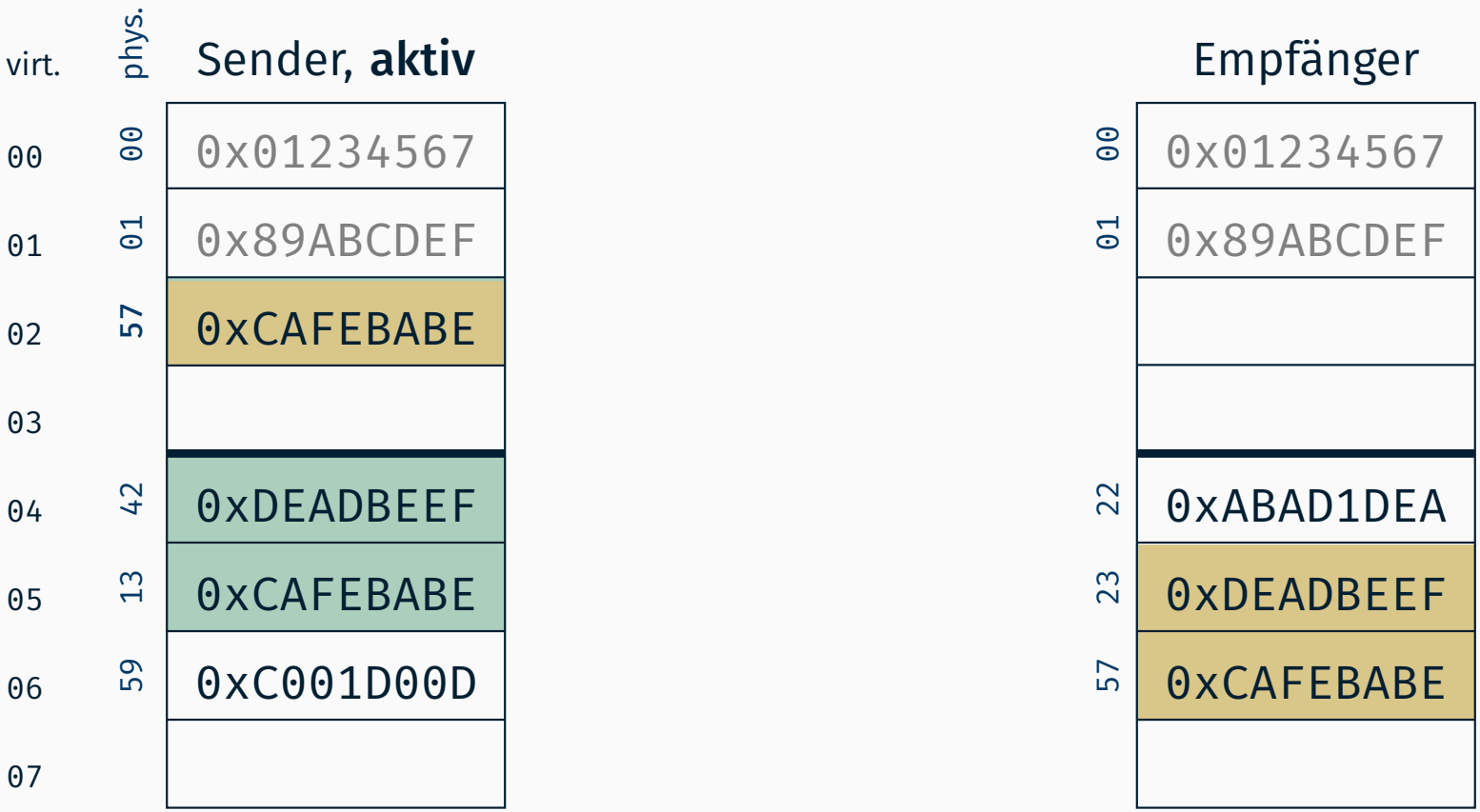
Kopieren zwischen Adressräumen (optimiert)



Achtung: Nach Änderung des Mappings TLB invalidieren!

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

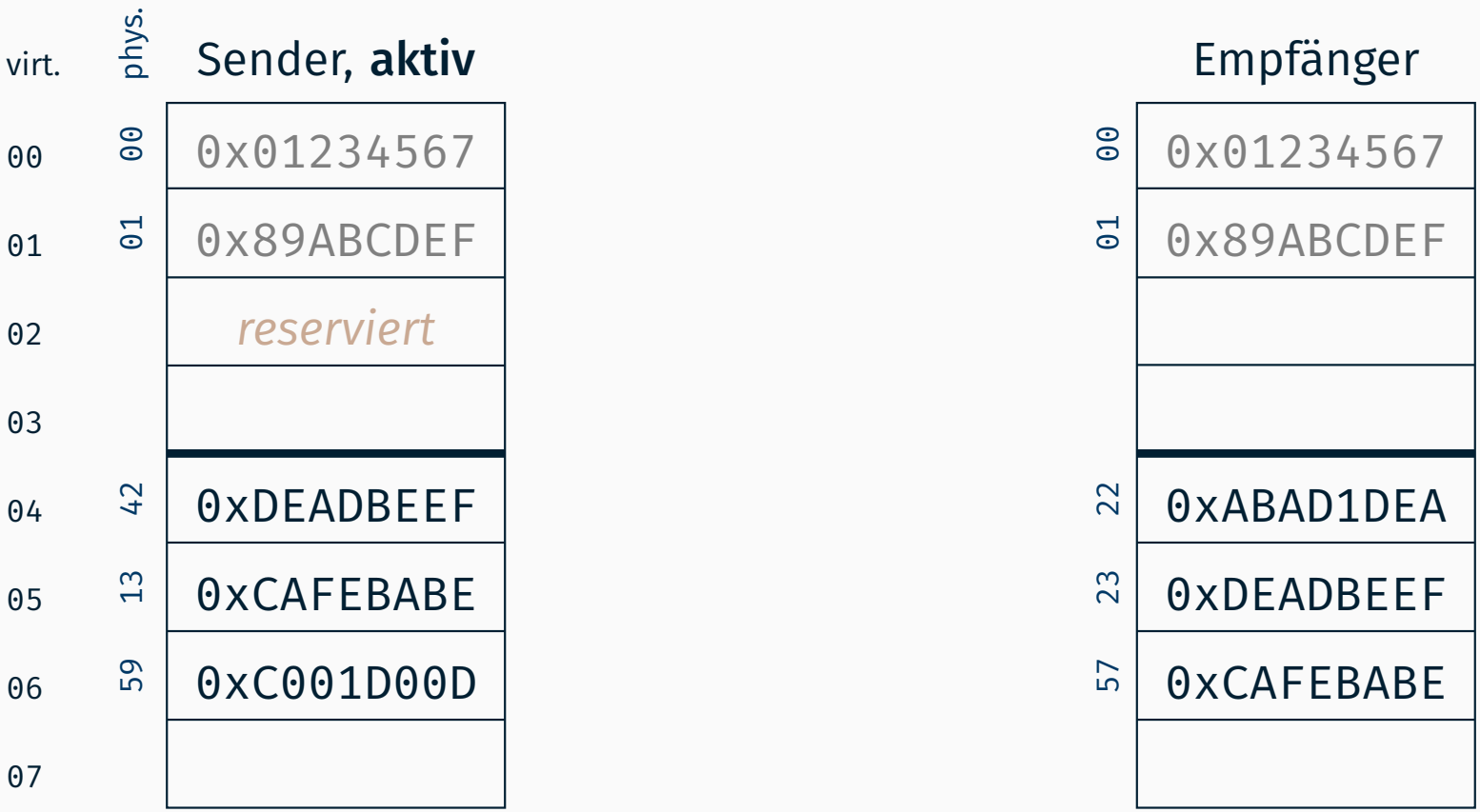
Kopieren zwischen Adressräumen (optimiert)



Achtung: Nach Änderung des Mappings TLB invalidieren!

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

Kopieren zwischen Adressräumen (optimiert)

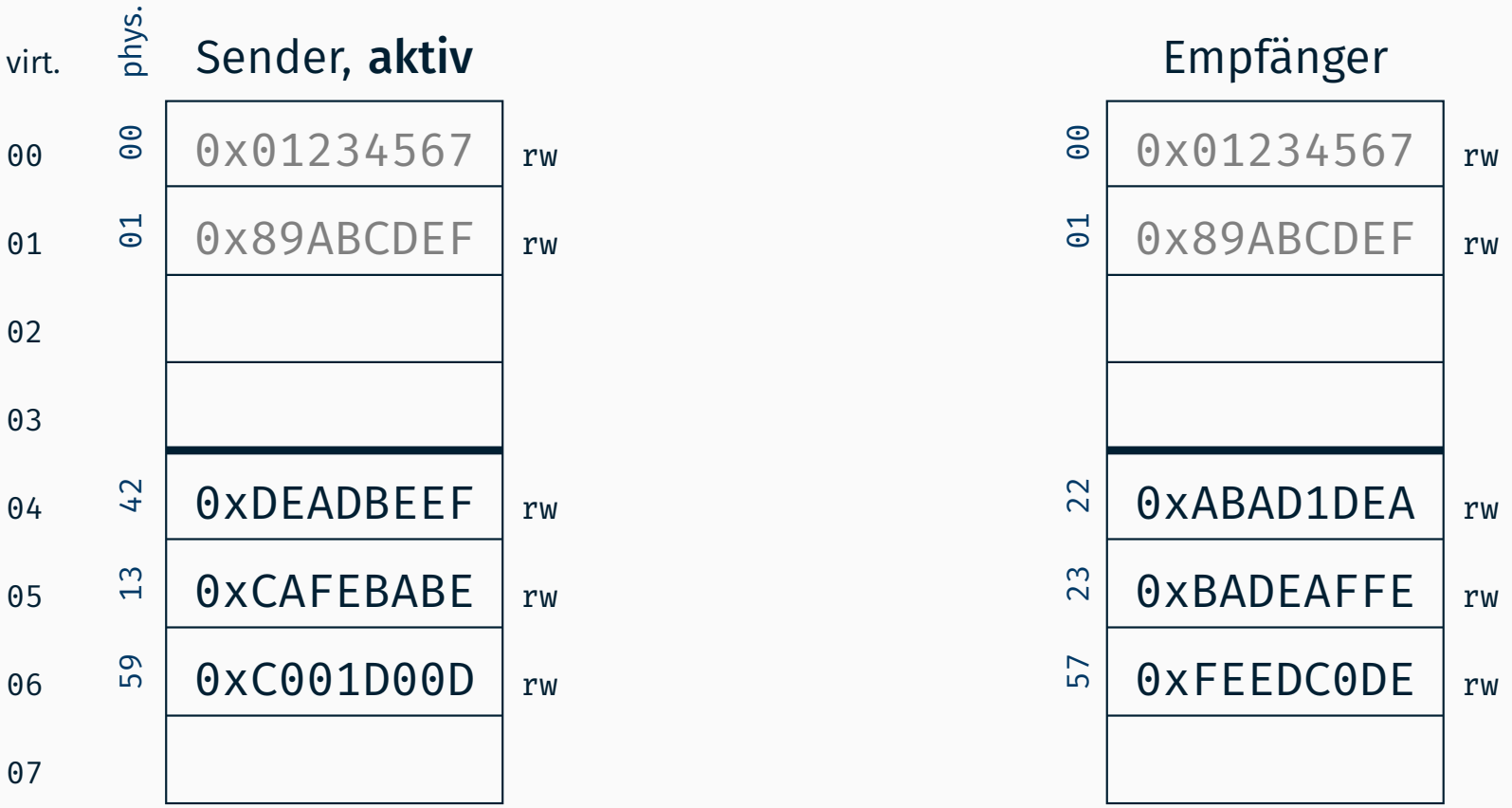


Achtung: Nach Änderung des Mappings TLB invalidieren!

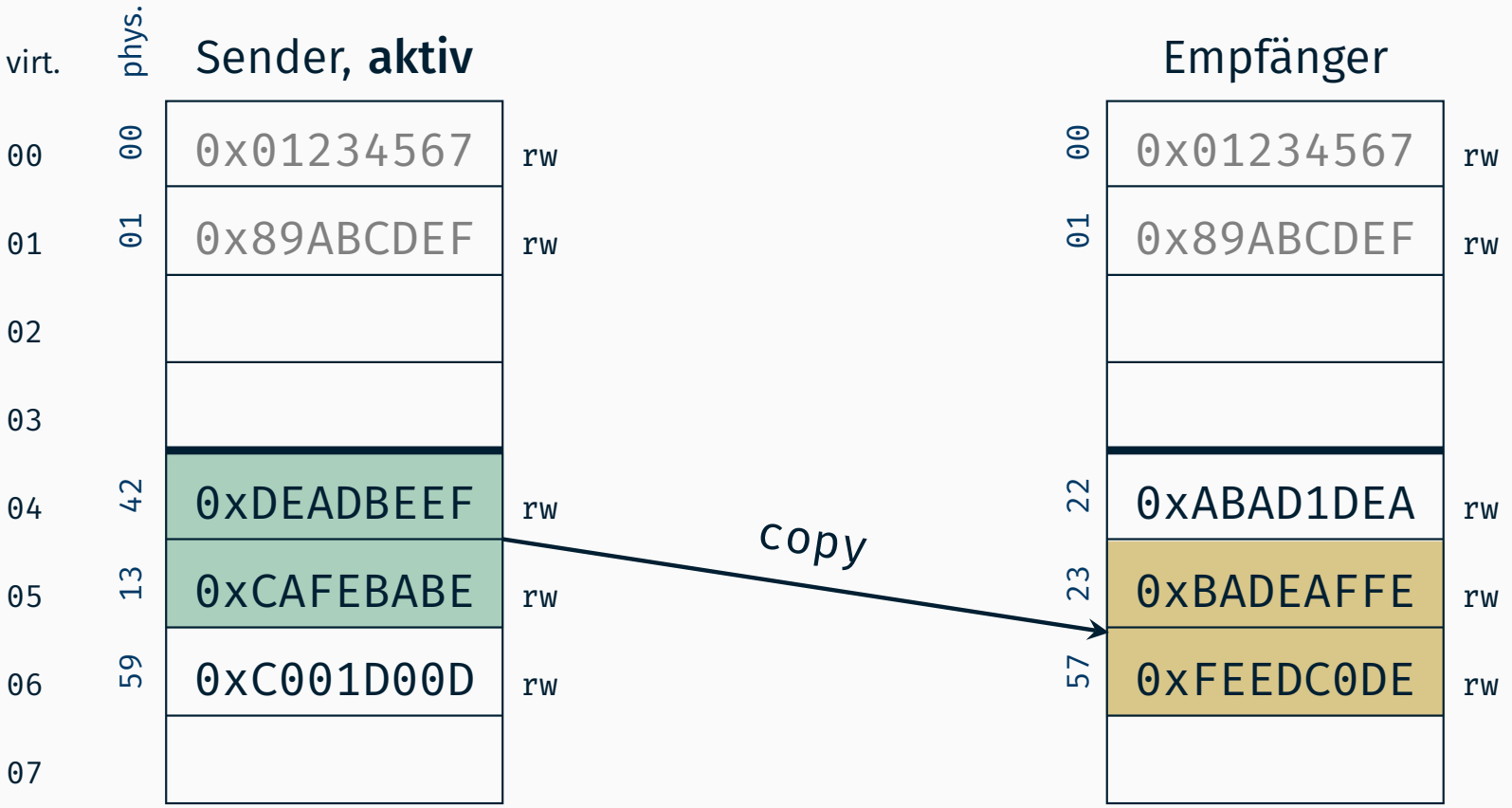
```
asm volatile("invlpg (%0)": : "r"(address) : "memory");
```

Copy-on-Write

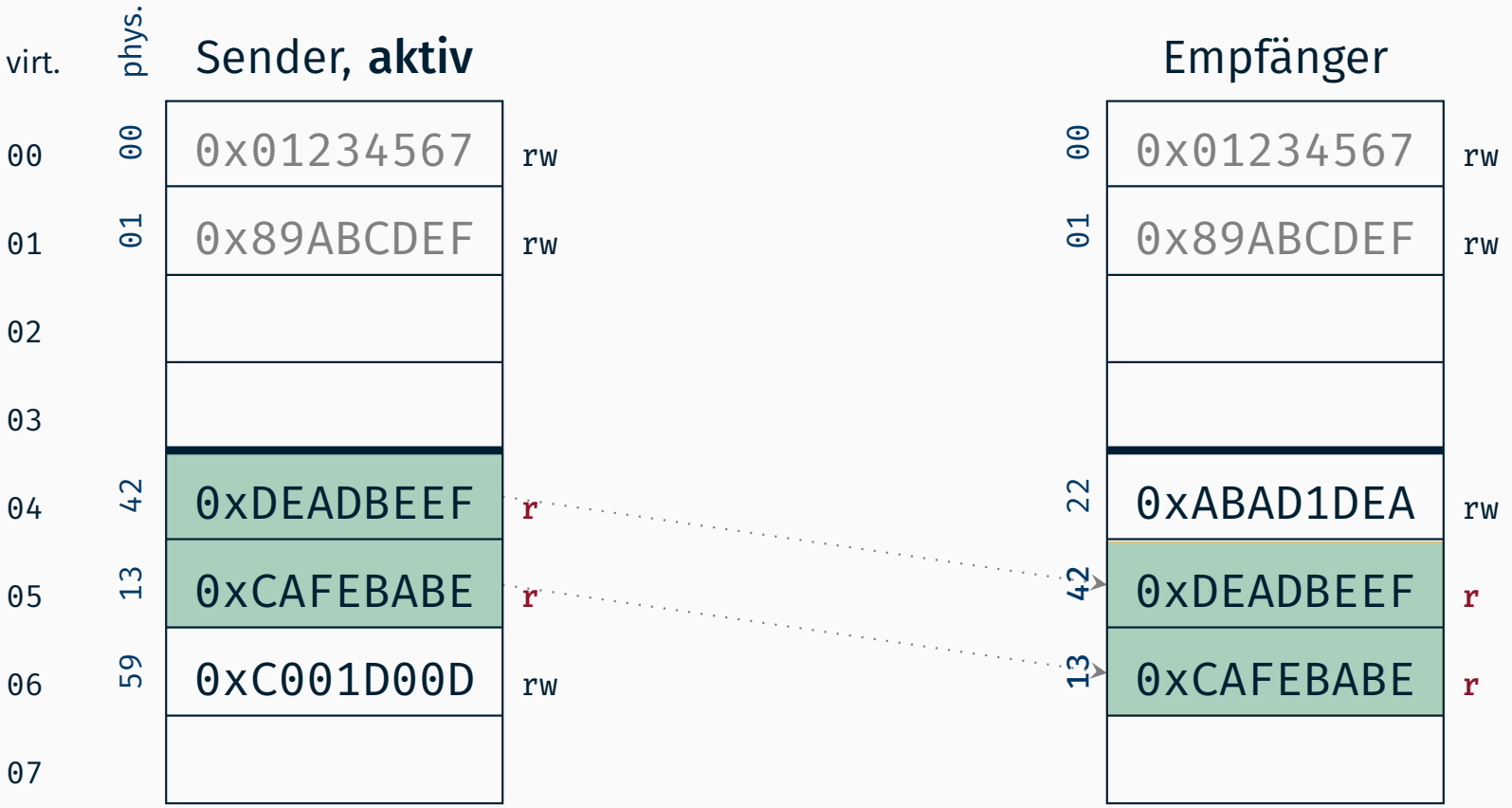
Copy-on-Write



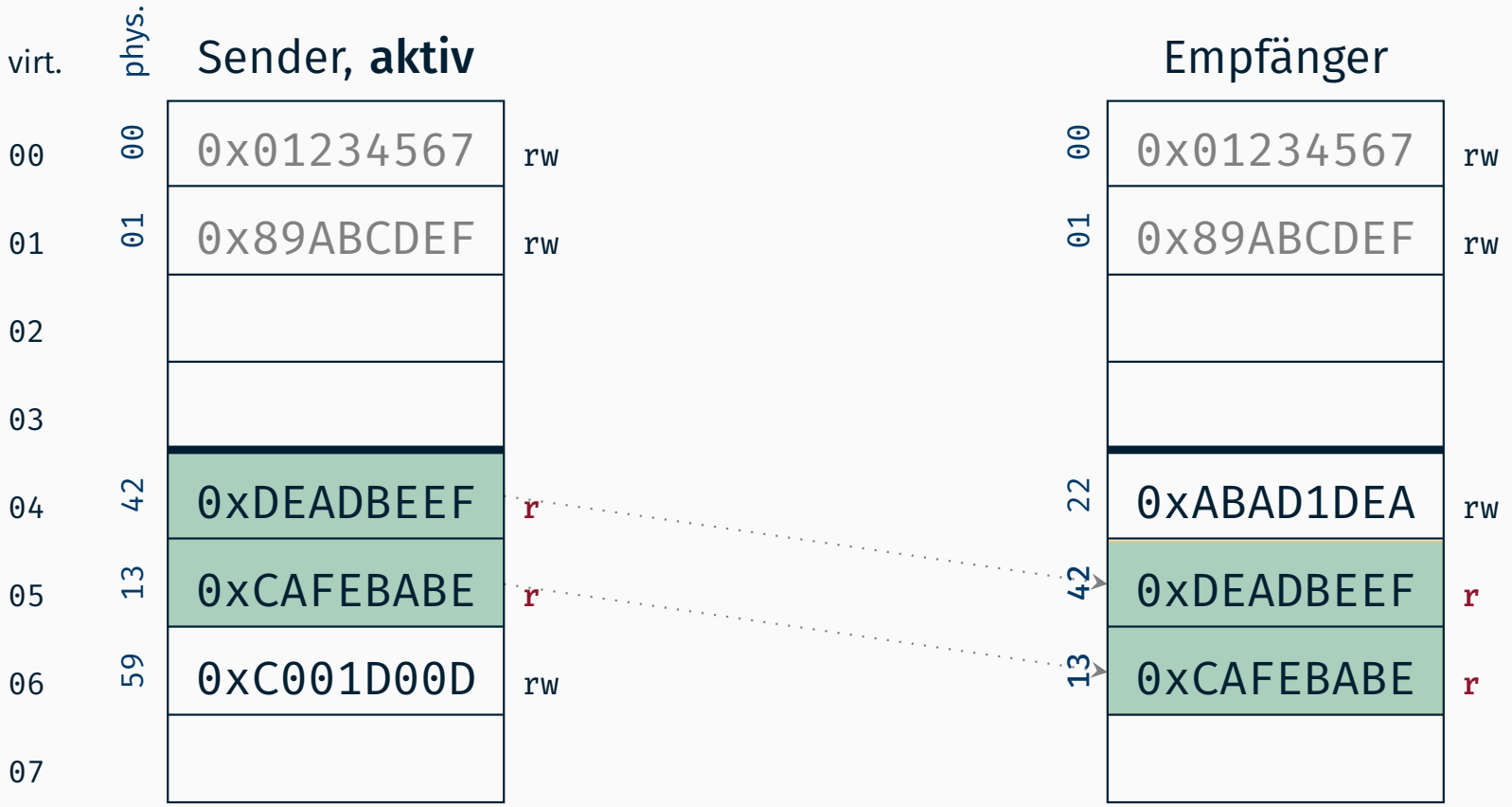
Copy-on-Write



Copy-on-Write

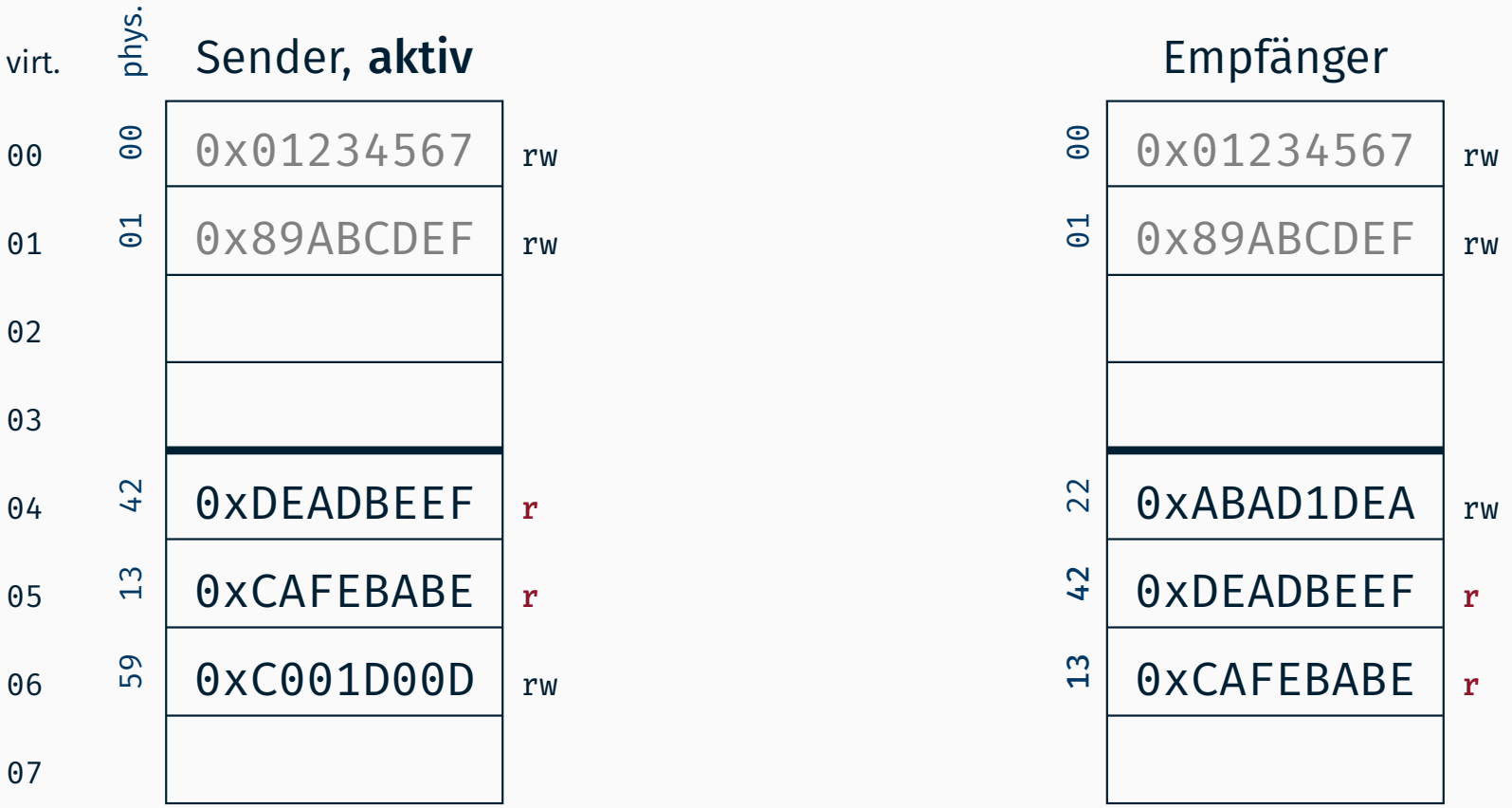


Copy-on-Write



Achtung: Nach Änderung des Mappings TLB invalidieren!

Copy-on-Write



Achtung: Nach Änderung des Mappings TLB invalidieren!

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?

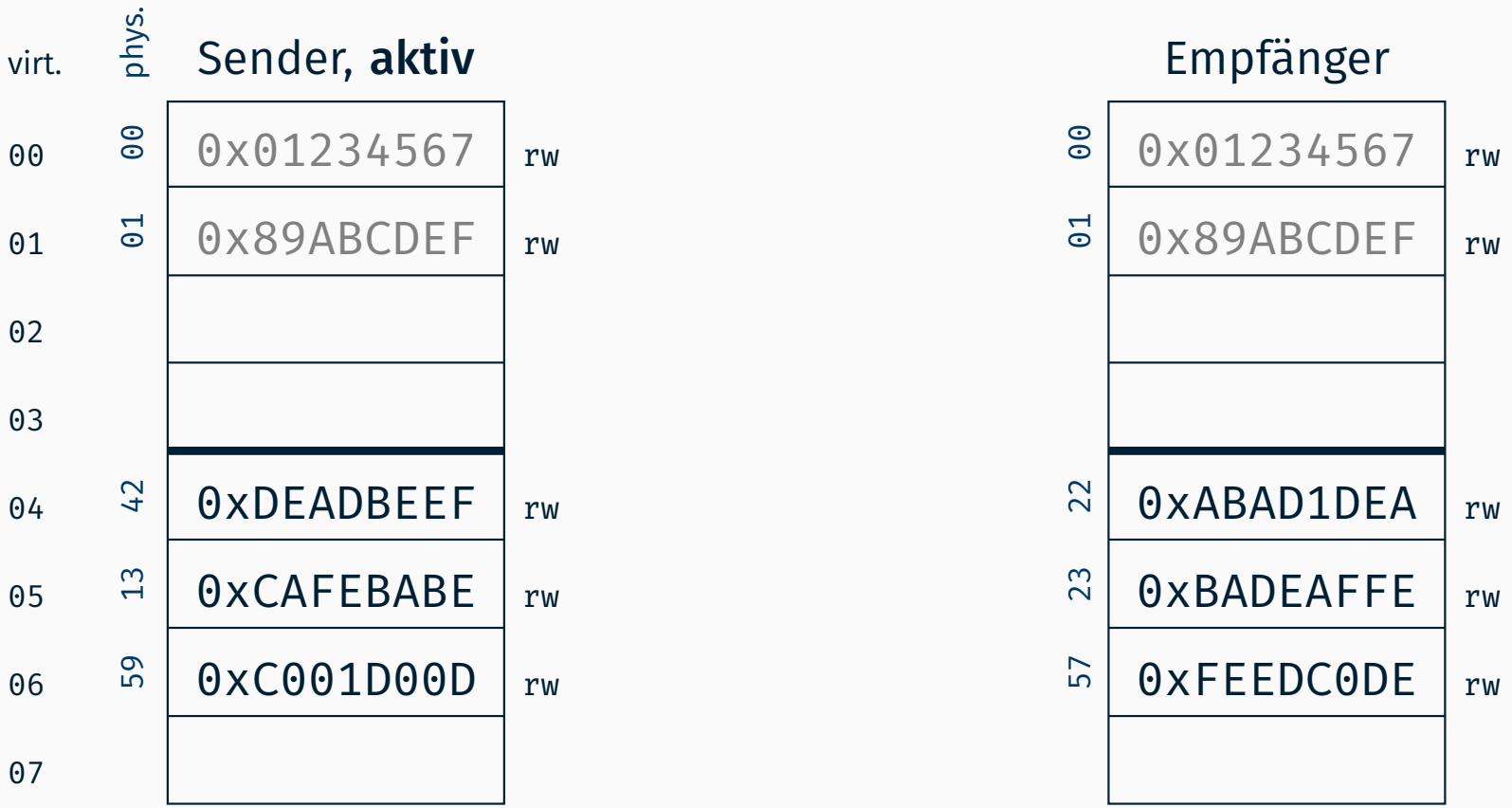
- das Gleiche!
- Referenzen zählen (auch bei `fork()`)

Implementierung des Referenzzählers?

- (z.B.) Shadow-Pagetable

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

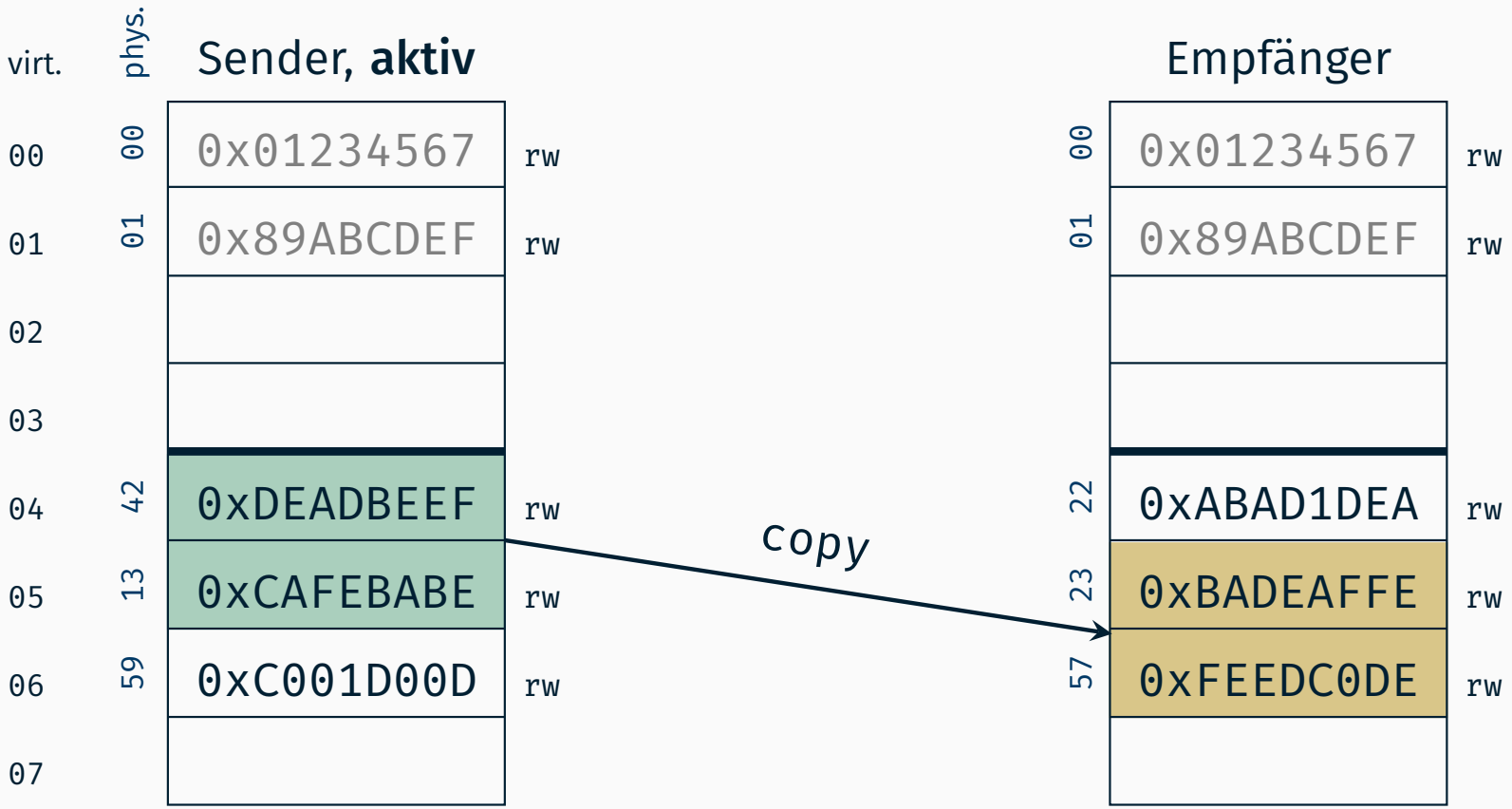
Copy-on-Write



Referenzzähler:

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

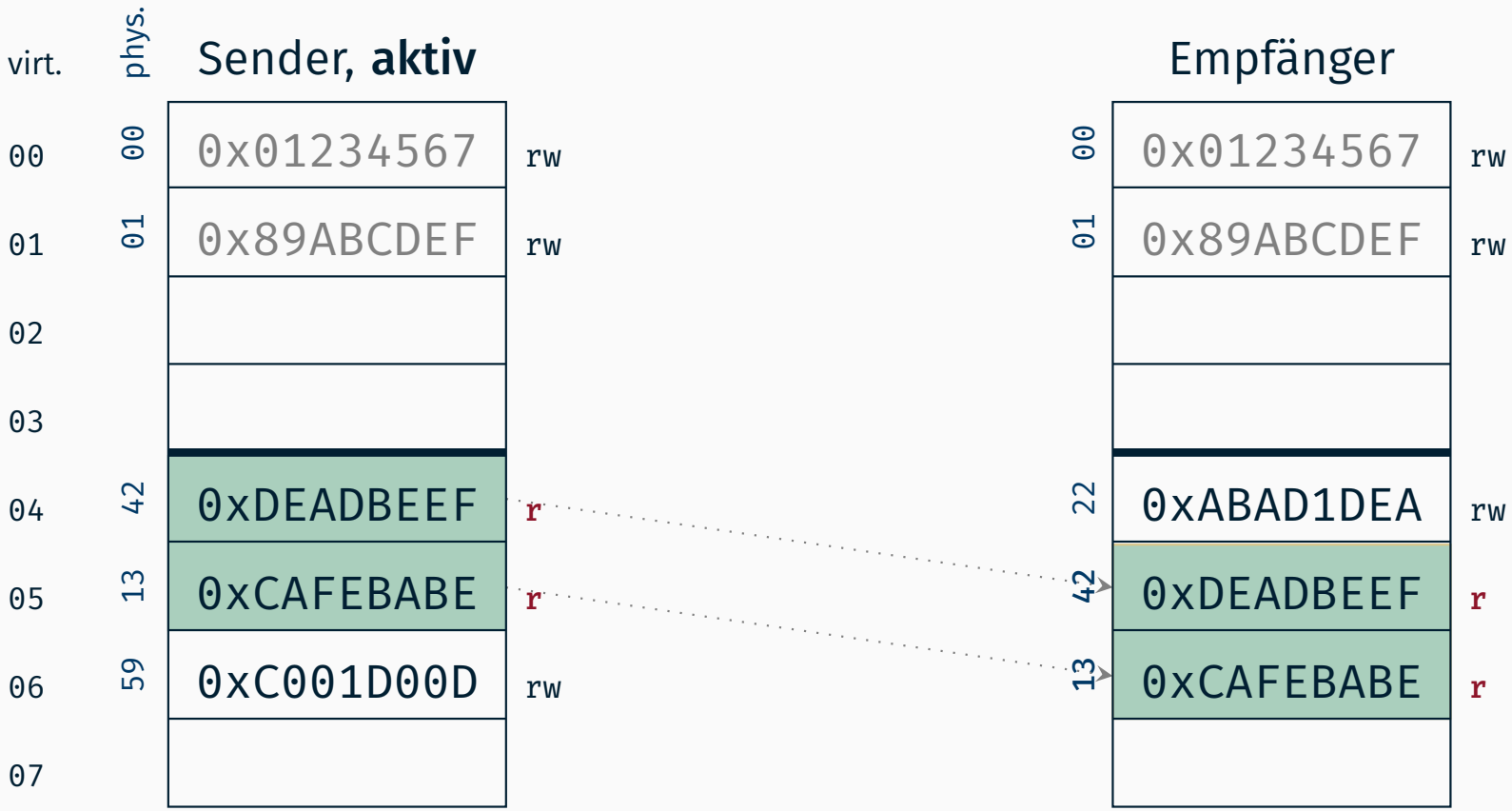
Copy-on-Write



Referenzzähler:

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

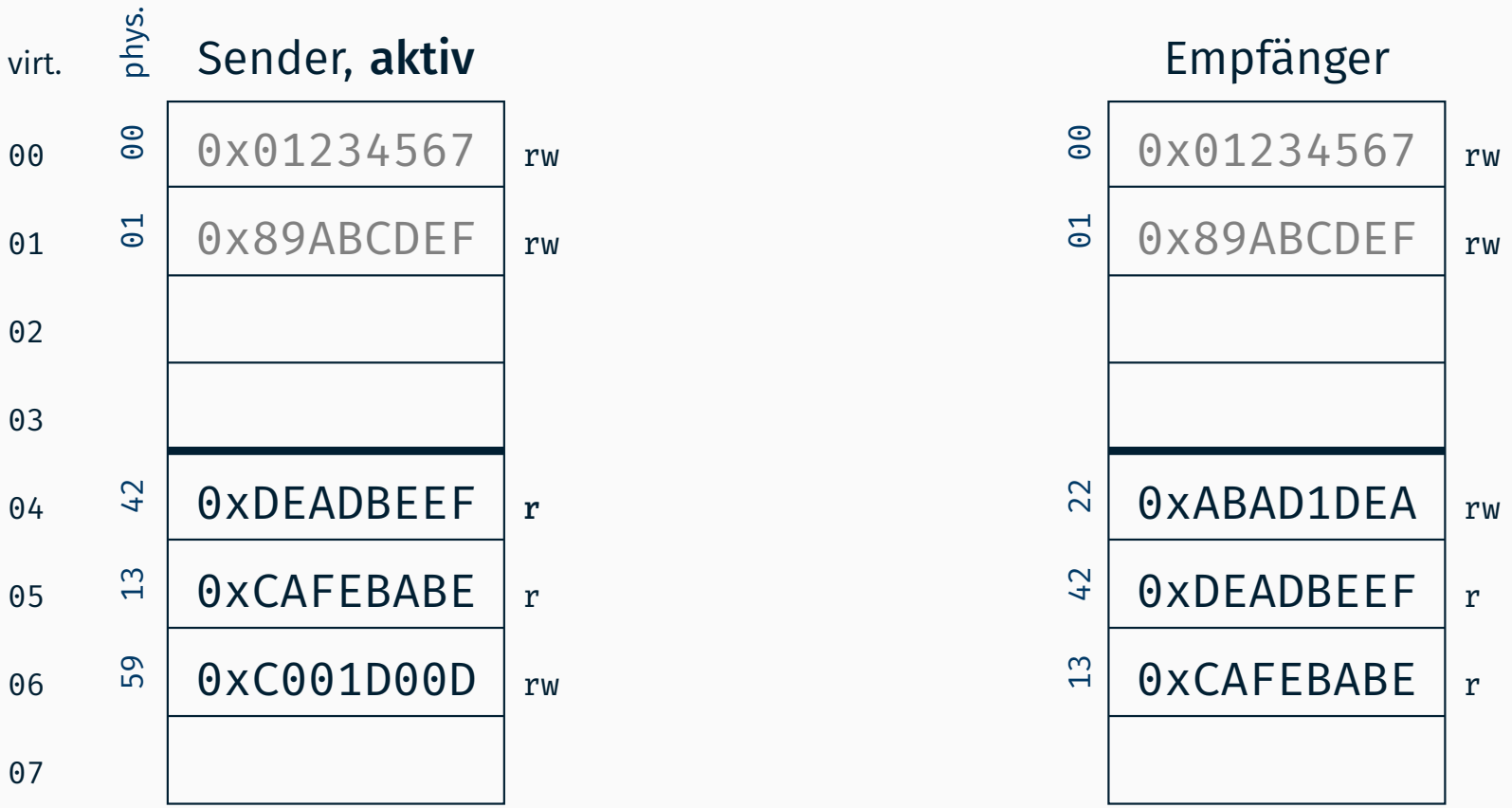
Copy-on-Write



Referenzzähler:

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

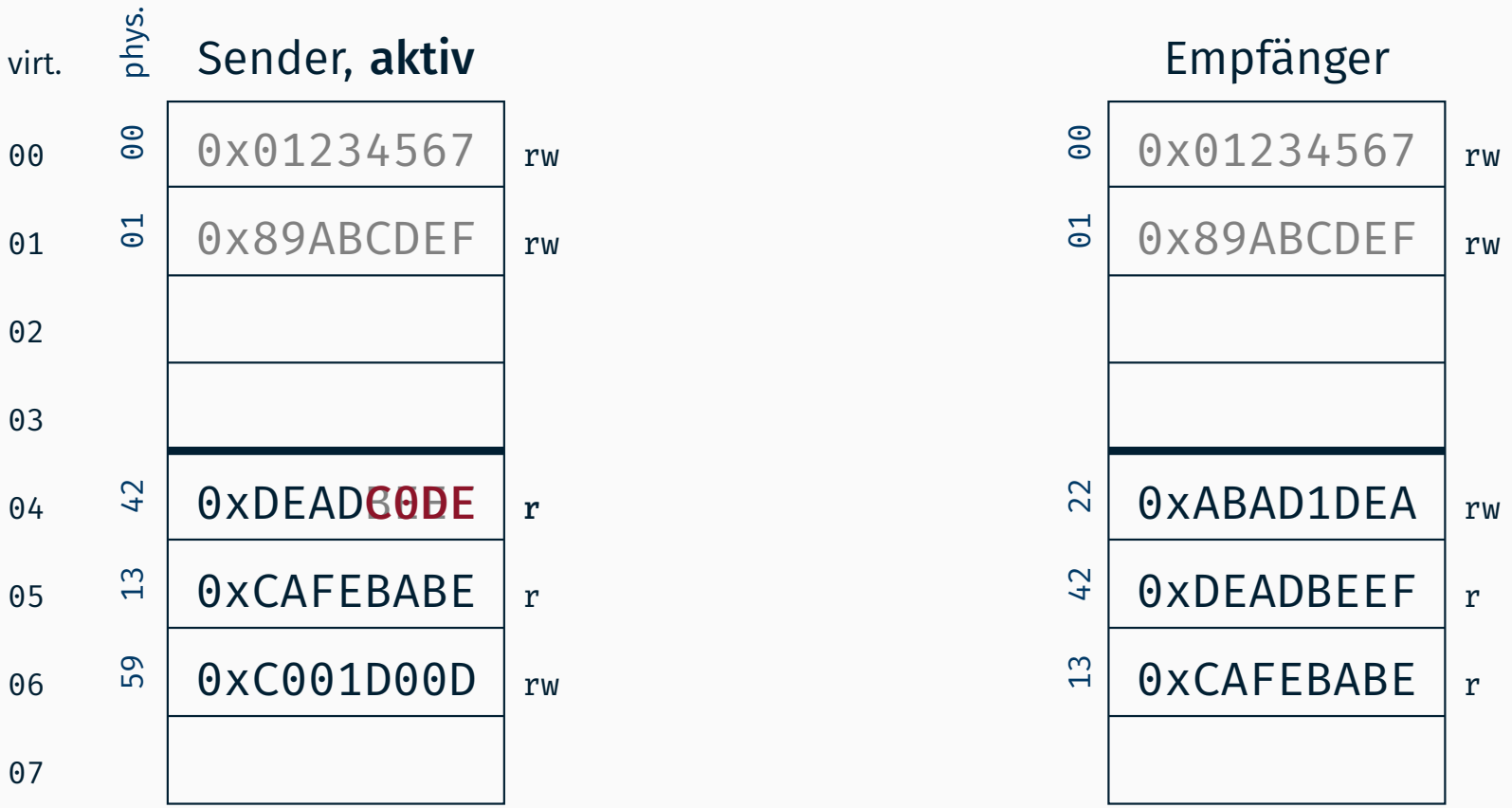
Copy-on-Write



Referenzzähler:

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

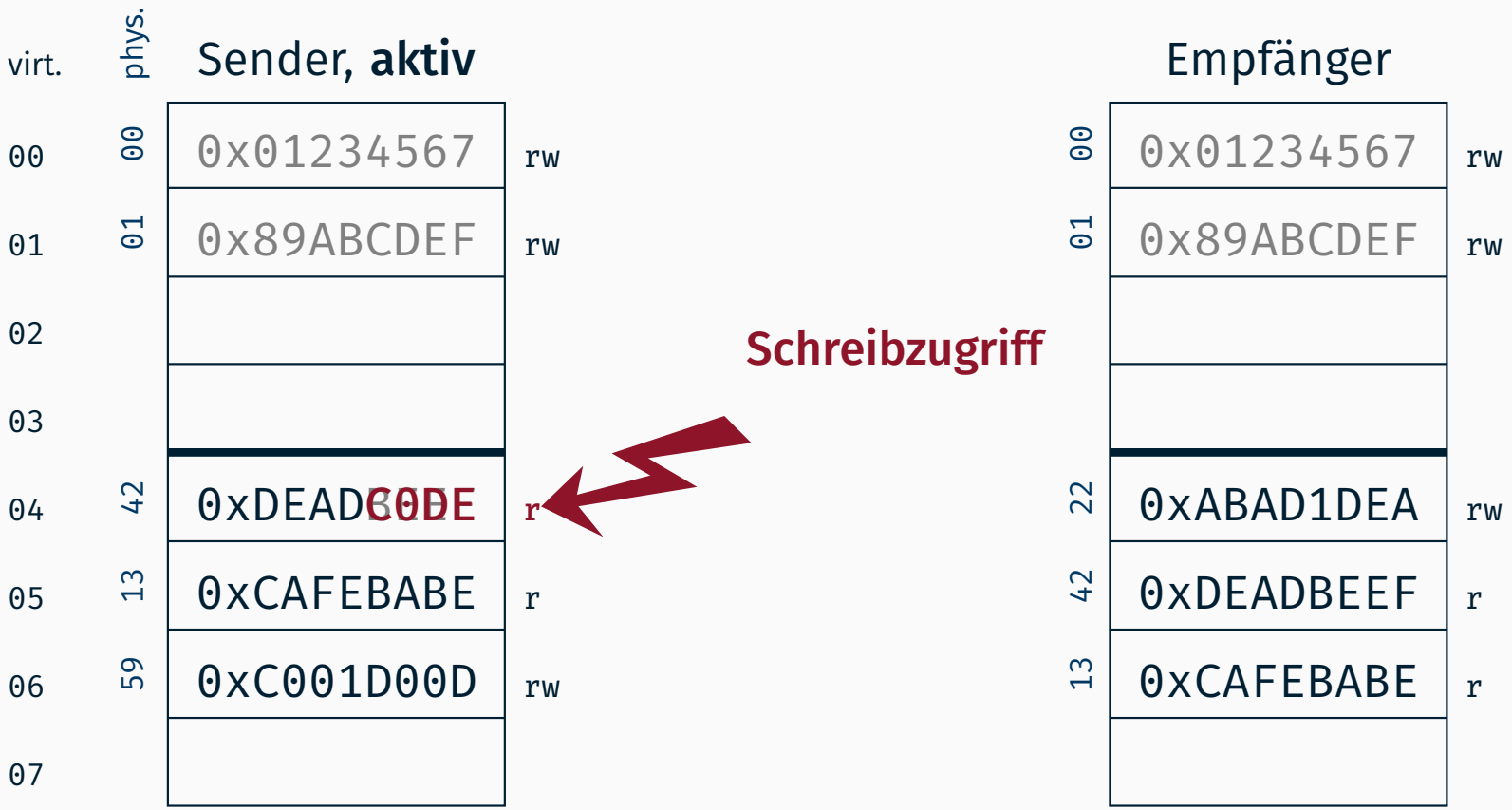
Copy-on-Write



Referenzzähler:

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

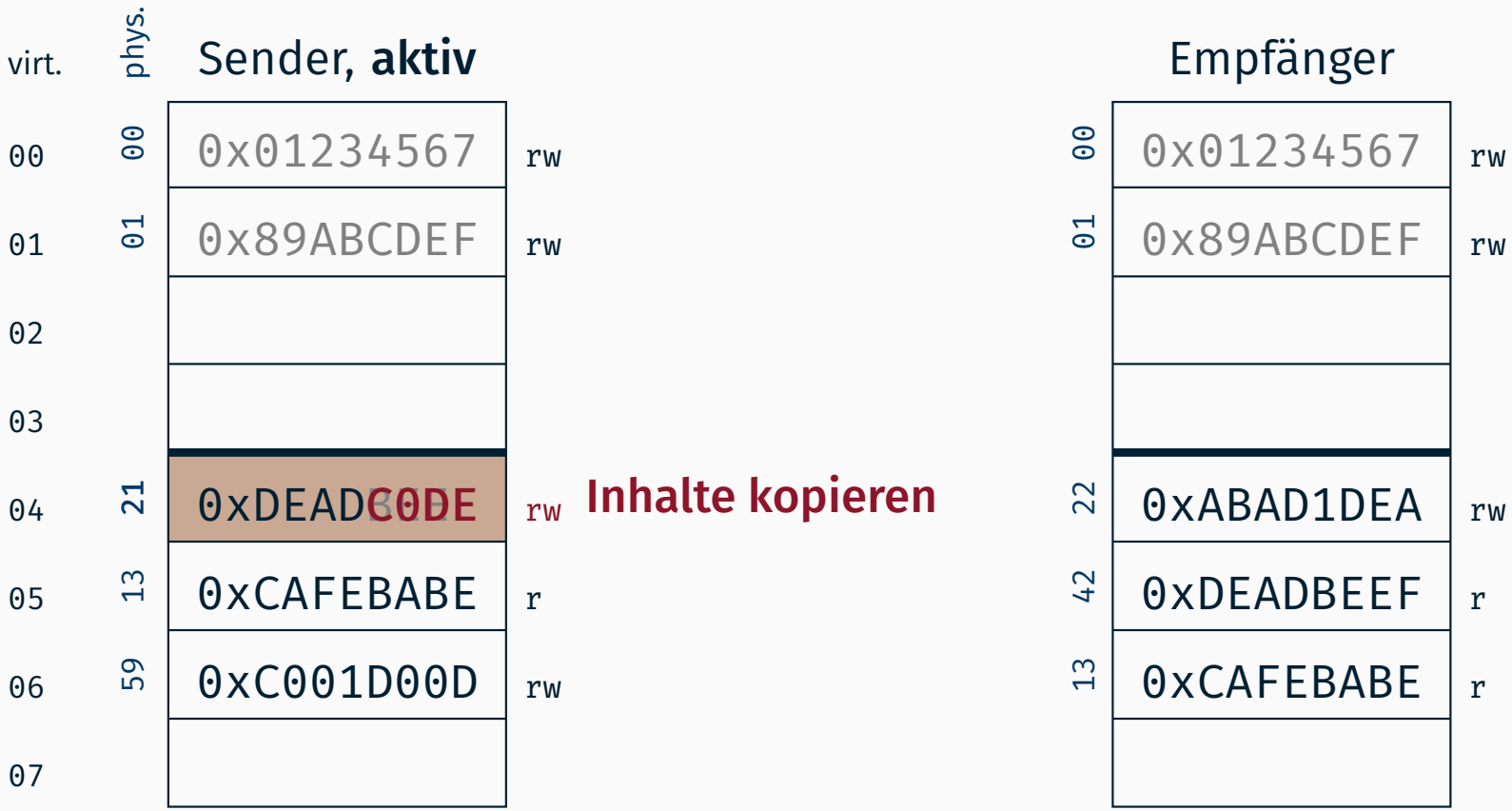
Copy-on-Write



Referenzzähler:

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

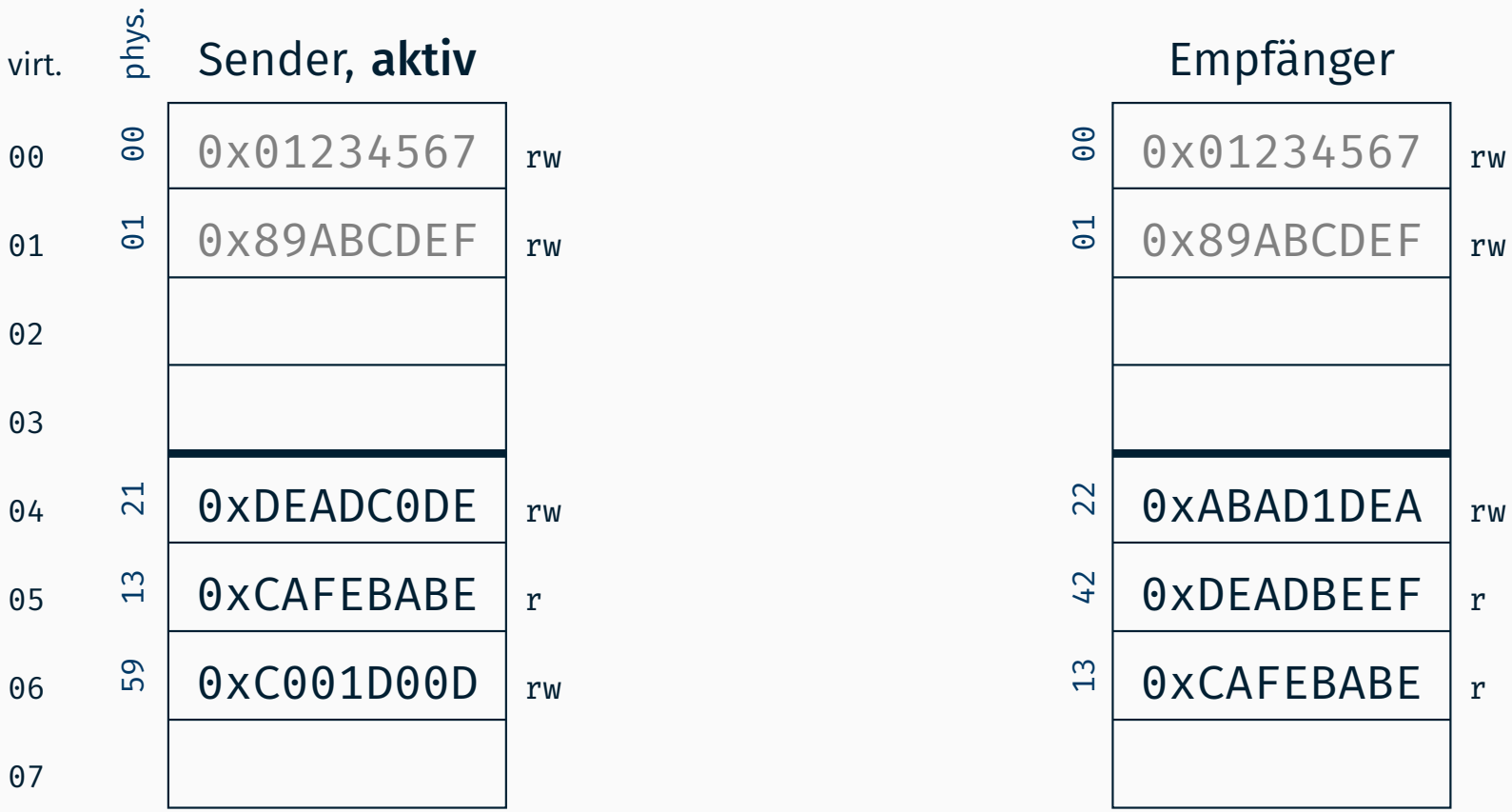
Copy-on-Write



Referenzzähler:

phys.	...	12	13	14	...	21	22	23	...	42	...	57	58	59	...
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 (`int 14/0xe`)

- eigener Handler, parallel zu Systemaufruf und Interrupts

Seitenfehler-Behandlung

Umsetzung von Copy-on-Write mittels Pagefaults (`int 14/0xe`)

- eigener Handler, parallel zu Systemaufruf und Interrupts

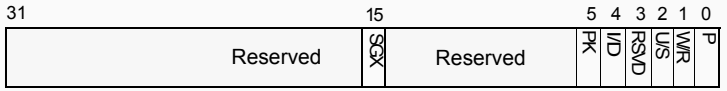
Informationen über Seitenfehler

- 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!)

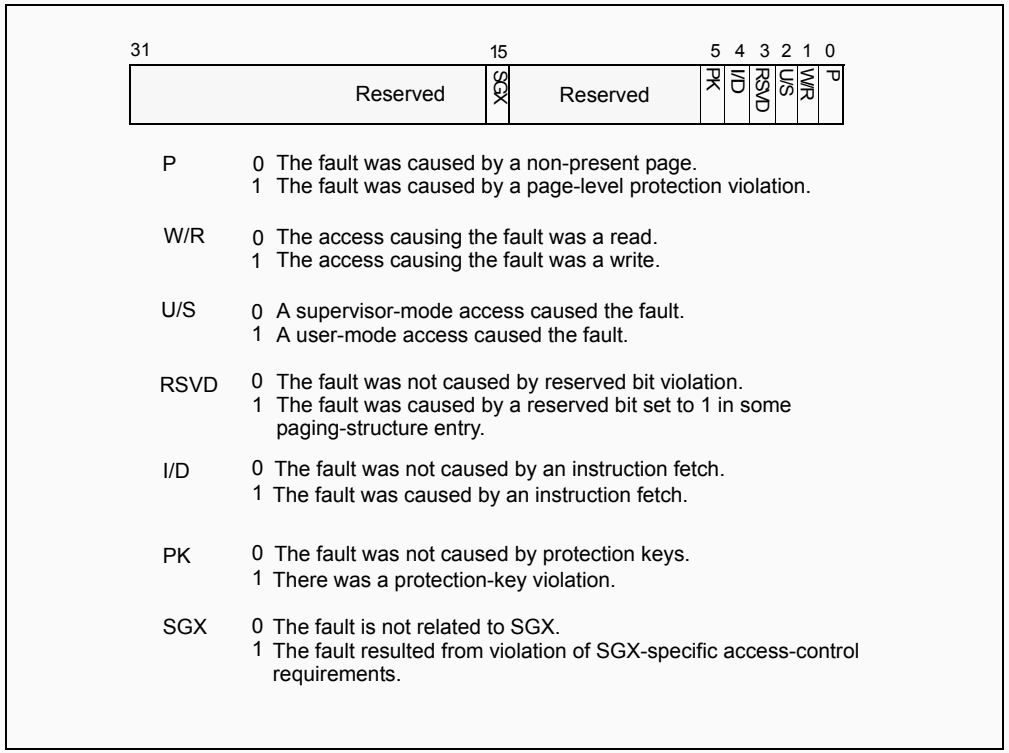


P	0	The fault was caused by a non-present page.
	1	The fault was caused by a page-level protection violation.
W/R	0	The access causing the fault was a read.
	1	The access causing the fault was a write.
U/S	0	A supervisor-mode access caused the fault.
	1	A user-mode access caused the fault.
RSVD	0	The fault was not caused by reserved bit violation.
	1	The fault was caused by a reserved bit set to 1 in some paging-structure entry.
I/D	0	The fault was not caused by an instruction fetch.
	1	The fault was caused by an instruction fetch.
PK	0	The fault was not caused by protection keys.
	1	There was a protection-key violation.
SGX	0	The fault is not related to SGX.
	1	The fault resulted from violation of SGX-specific access-control requirements.

Seitenfehler-Behandlung (cont.)

Fehlercode auf dem Kernel-Stack

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



P	0	The fault was caused by a non-present page.
	1	The fault was caused by a page-level protection violation.
W/R	0	The access causing the fault was a read.
	1	The access causing the fault was a write.
U/S	0	A supervisor-mode access caused the fault.
	1	A user-mode access caused the fault.
RSVD	0	The fault was not caused by reserved bit violation.
	1	The fault was caused by a reserved bit set to 1 in some paging-structure entry.
I/D	0	The fault was not caused by an instruction fetch.
	1	The fault was caused by an instruction fetch.
PK	0	The fault was not caused by protection keys.
	1	There was a protection-key violation.
SGX	0	The fault is not related to SGX.
	1	The fault resulted from violation of SGX-specific access-control requirements.

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?