Asynchronous **I**nter **P**rocess **C**ommunication

Design und Implementierung basierend auf dem L4 Mikrokern

sowie Kompatibilität bezüglich SOS

(**S**imple **O**perating **S**ystem)

vorgetragen von Steffen Knapp

Ablauf der Präsentation:

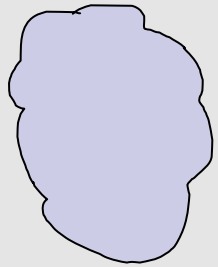
- Einführung in die Thematik
- Kommunikationsmodell
- Warum asynchrones IPC ?
- Abstraktes Design des IPC
- Implementierung in L4
- Kompatibilität bzgl. SOS

Ablauf der Präsentation:

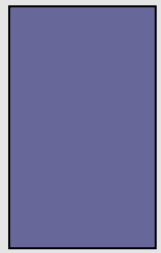
- **Einführung in die Thematik**
- Kommunikationsmodell
- Warum asynchrones IPC ?
- Abstraktes Design des IPC
- Implementierung in L4
- Kompatibilität bzgl. SOS

Grundbegriffe

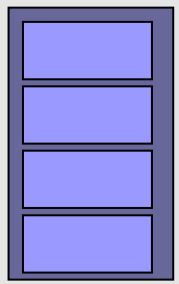
- Adressräume:
  - Menge von (virtuellen) Adressen



- Prozess / Task:
  - „Programm in Ausführung“
  - arbeitet in jeweils eigenem Adressraum

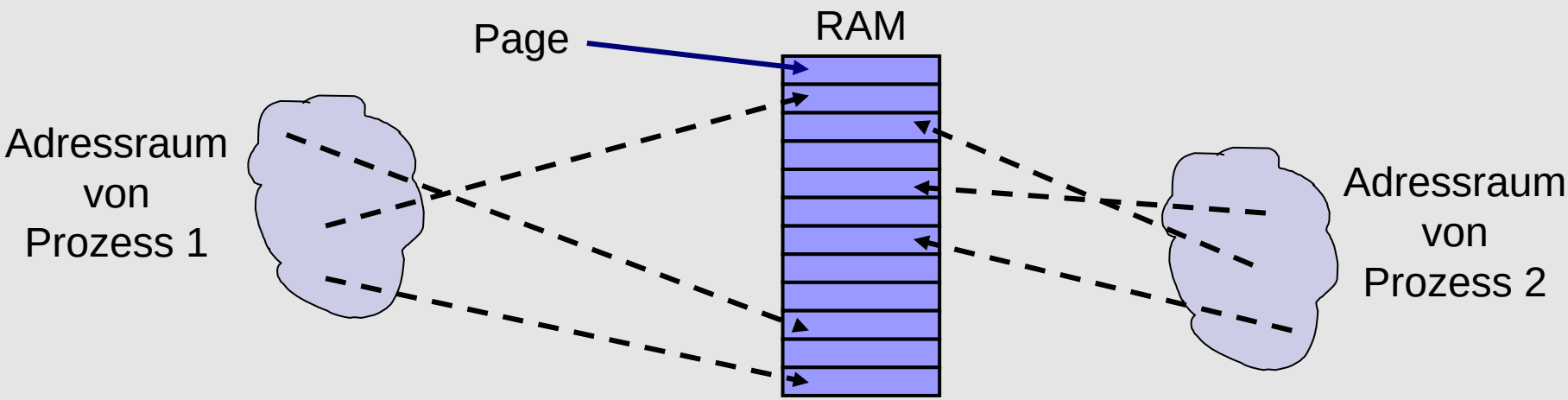


- Thread
  - Unterteilung innerhalb eines Prozesses (Multithread)



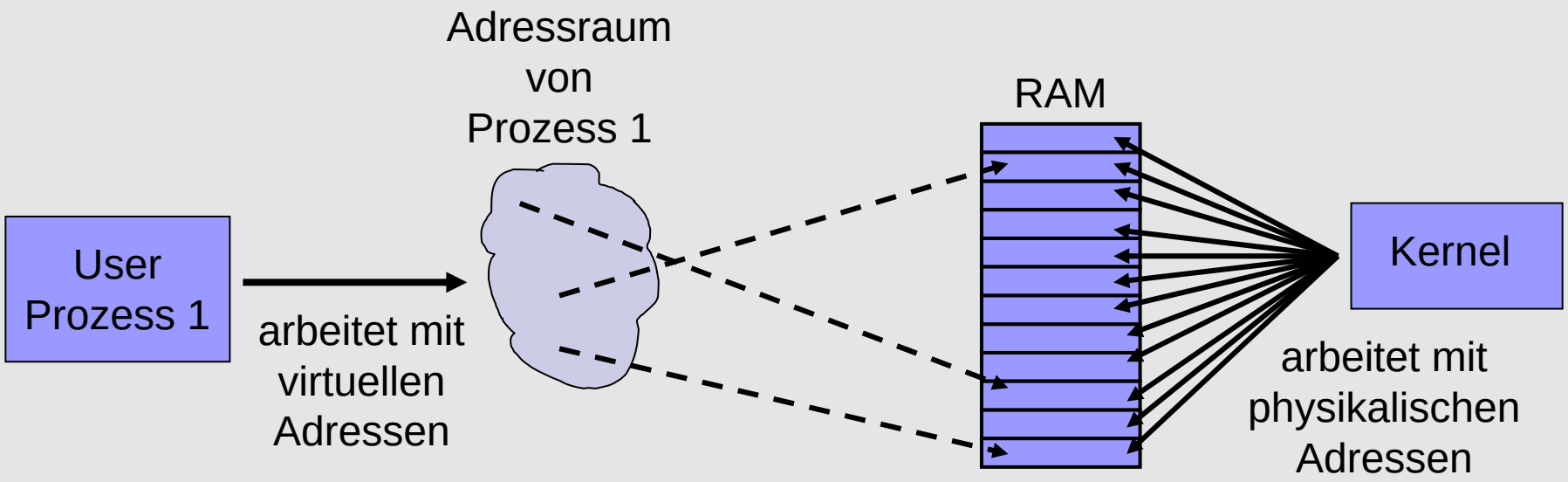
Verwaltung des Speichers

- Die Adressen eines Adressraums werden auf physikalischen Speicher abgebildet (mapping)
- Prozesse können nur auf den in ihrem Adressraum abgebildeten Speicher zugreifen
- Page: Unterteilung des Speichers



Weitere Grundbegriffe

- Mode
  - man unterscheidet zwischen
    - User Mode: Adressübersetzung
    - Kernel Mode: direkter Zugriff auf RAM



Techniken der Datenübertragung

- zero copy
  - Bezeichnung für Übertragungsvorgang, bei denen Daten nicht explizit kopiert, sondern mit Hilfe von Speicher-Abbildungen übergeben werden.
- copy on write
  - Daten werden *vorläufig* durch Speicher-Abbildungen übergeben. Der Kopiervorgang wird so lange hinausgezögert, bis die zu kopierenden Daten verändert werden. Vor der Änderung wird dann eine Kopie der Daten angelegt.

Kostenfrage

- Kontext - Wechsel:
  - Wechsel zwischen Prozessen
  - Beinhaltet großen administrativen Aufwand:
    Sichern der alten + Laden der neuen Umgebung
- Mode - Wechsel:
  - Wechsel zwischen User und Kernel Mode
    Sichern und umsetzen von Registern

Wer spricht mit wem?

- Kommunikation von Prozessen / Threads über Adressräume

- Kommunikation innerhalb der Adressräume wird nicht näher betrachtet
  - direkter Zugriff auf Daten möglich, da gleicher Adressraum

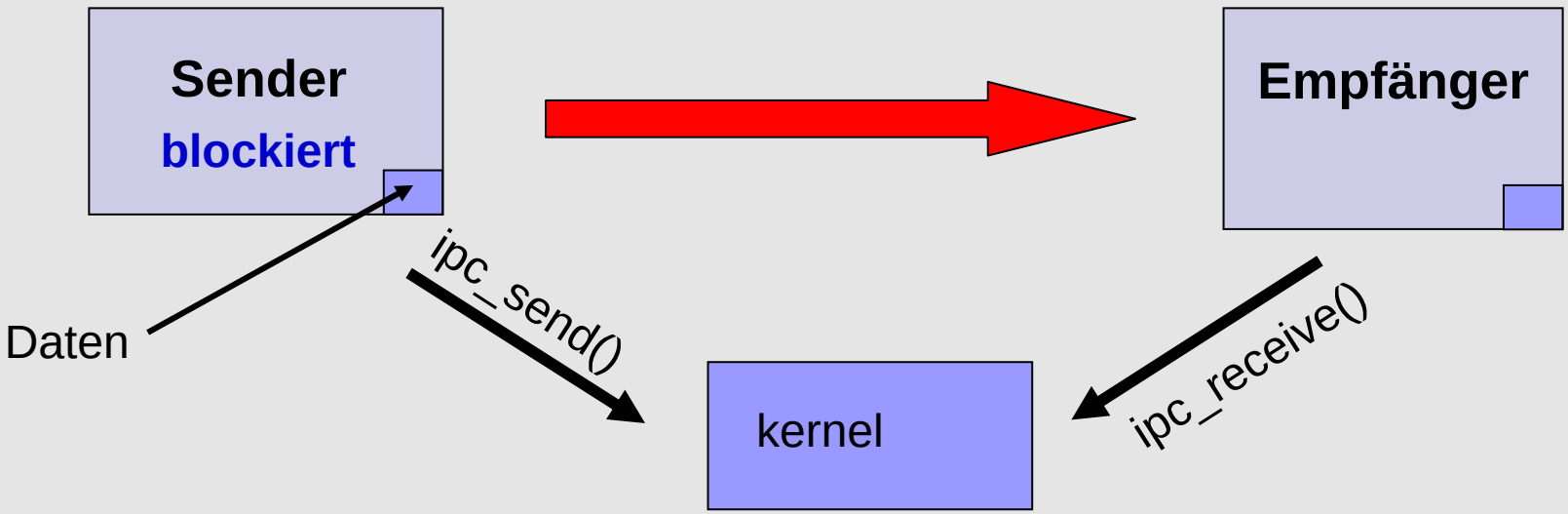
Ablauf der Präsentation:

- Einführung in die Thematik
- **Kommunikationsmodell**
- Warum asynchrones IPC ?
- Abstraktes Design des IPC
- Implementierung in L4
- Kompatibilität bzgl. SOS

„Inter Process Communication“

synchron:

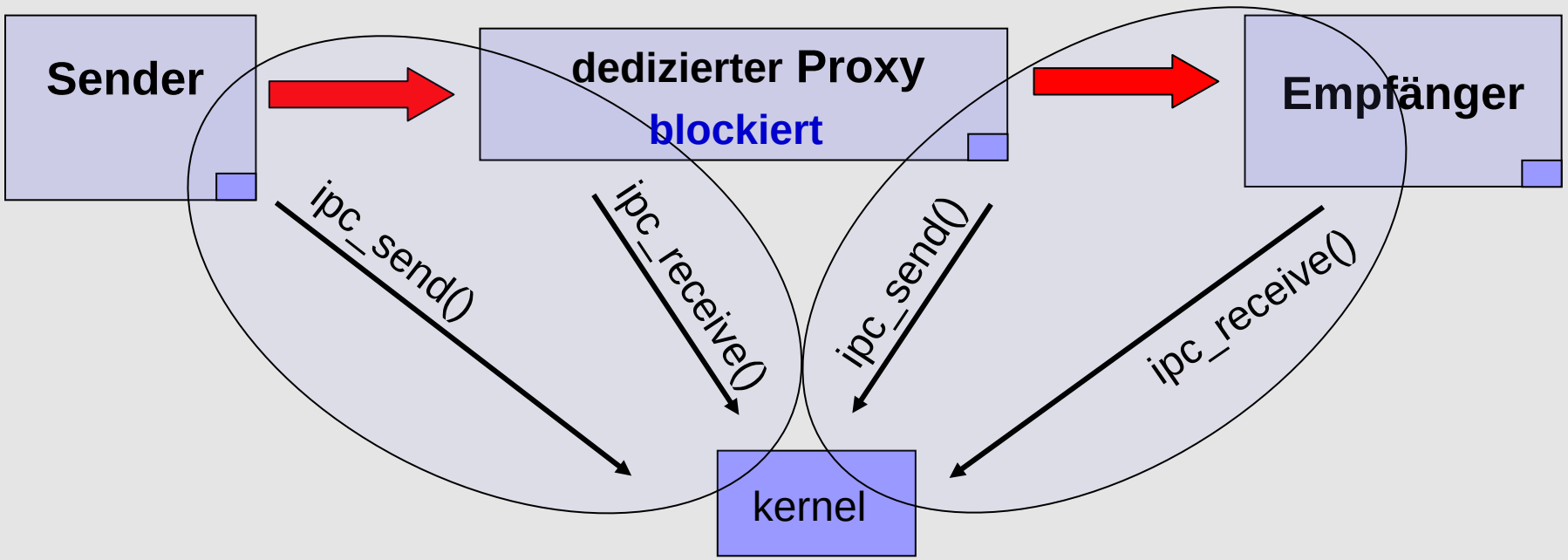
- Nachricht wird erst übertragen, wenn Empfänger dies initiiert
- Dieser Zeitpunkt (und somit die Dauer der Blockierung) ist vom Empfänger beliebig wählbar



„Inter Process Communication“

asynchron via synchron:

- dedizierter Proxy wartet auf Nachricht vom Sender
- der Proxy ist für die weitere Zustellung der Nachricht zuständig
- transparent für synchrone Kommunikations-Protokolle



Timeout

- Blockierung eines Prozesses kann zeitlich beschränkt werden
- Insbesondere timeout von 0 möglich
- Bei Auftreten eines timeout:
  - Blockierung wird mit einer Fehlermeldung abgebrochen
  - IPC Nachricht ist nicht übermittelt
  - ggf. Wiederholung zum späteren Zeitpunkt

Empfangs-Arten

- open receive
  - Empfänger nimmt Nachrichten von jedem Sender an
- closed receive
  - Empfänger nimmt Nachrichten nur von einem vorher spezifizierten Sender an

3 Wege der Datenübertragung

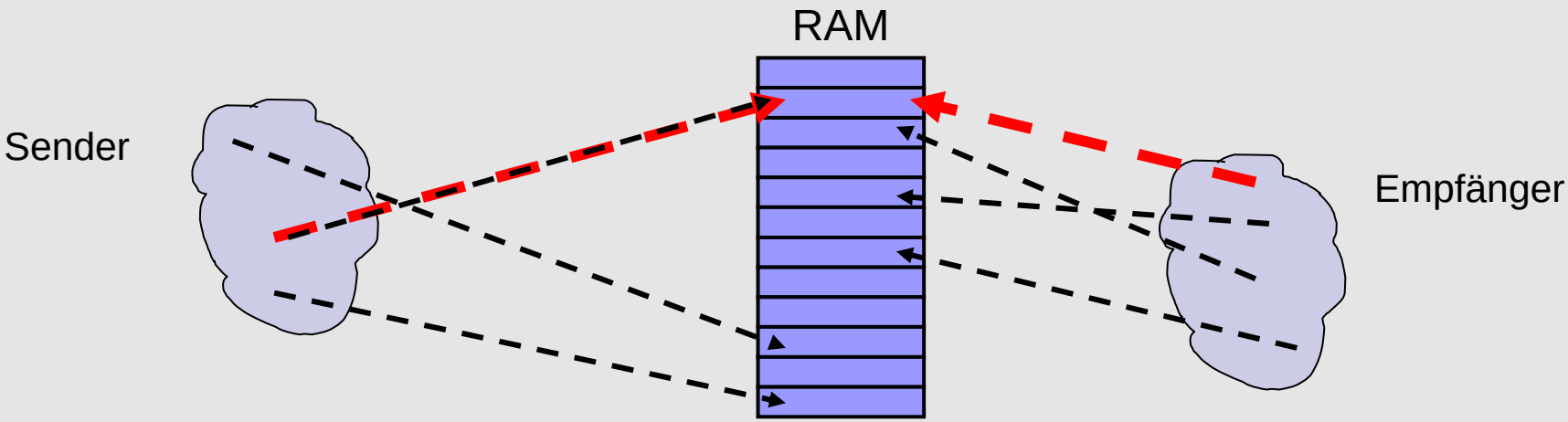
1) Flexpages

dynamischer Verweis auf Memory-Pages

a) Pages können an andere Prozesse übergeben werden (Donate)

b) Prozesse können sich die Pages ‚teilen' (Share)

→ schnell, da lediglich Zuordnung umgesetzt wird (zero copy)



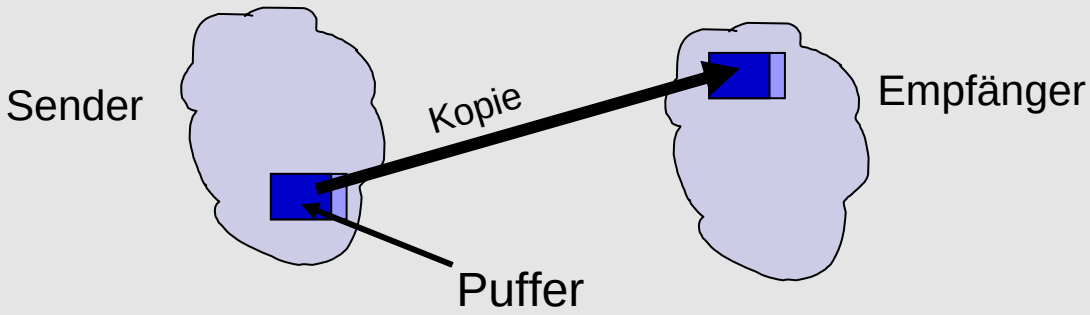
3 Wege der Datenübertragung

2) String Dopes

Multi-Tupel bestehend aus:

- Pointer auf Puffer
- Nutzdatenlänge innerhalb des Puffers
  - Daten werden mit Hilfe von Kopieroperationen übertragen
  - Bevor Daten übertragen werden muss der Empfänger Speicher allozieren

    → langsam



3 Wege der Datenübertragung

3) Register

kleinere Datenmengen können innerhalb der IPC Nachricht als Registerinhalte übertragen werden

- Der Registerinhalt wird dabei einfach umkopiert

  → schnell, jedoch nur einsetzbar bei kleinen Datenmengen

Ablauf der Präsentation:

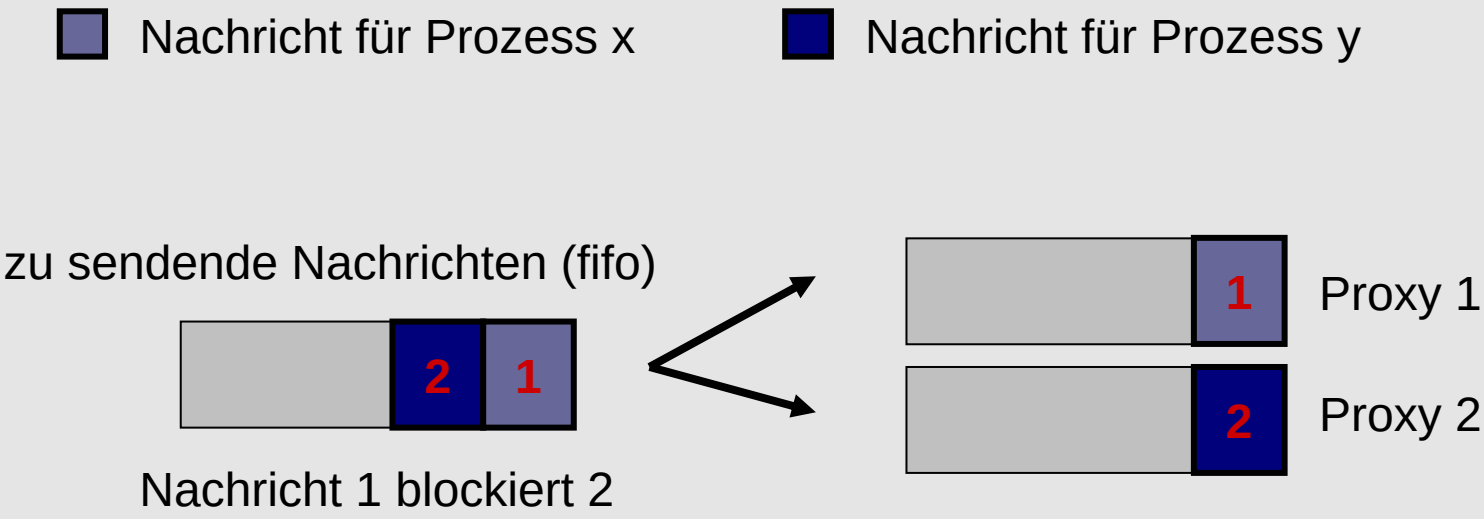
- Einführung in die Thematik
- Kommunikationsmodell
- **Warum asynchrones IPC ?**
- Abstraktes Design des IPC
- Implementierung in L4
- Kompatibilität bzgl. SOS

Vorteile von asynchronem IPC

- nicht oder nur minimal blockiert:
  - Effizientere Prozessausführung
  - Hohe Verfügbarkeit, z.B. ist der Datenbankserver (bei Anfragen) nicht blockiert durch Festplattenzugriffe
- Trennung von Senden und Empfangen

Vorteile von asynchronem IPC

- Optimierungen möglich, z.B. können durch ‚per-receiver‘ Proxy Blockierung innerhalb der Nachrichten aufgelöst werden



Nachteile von asynchronem IPC

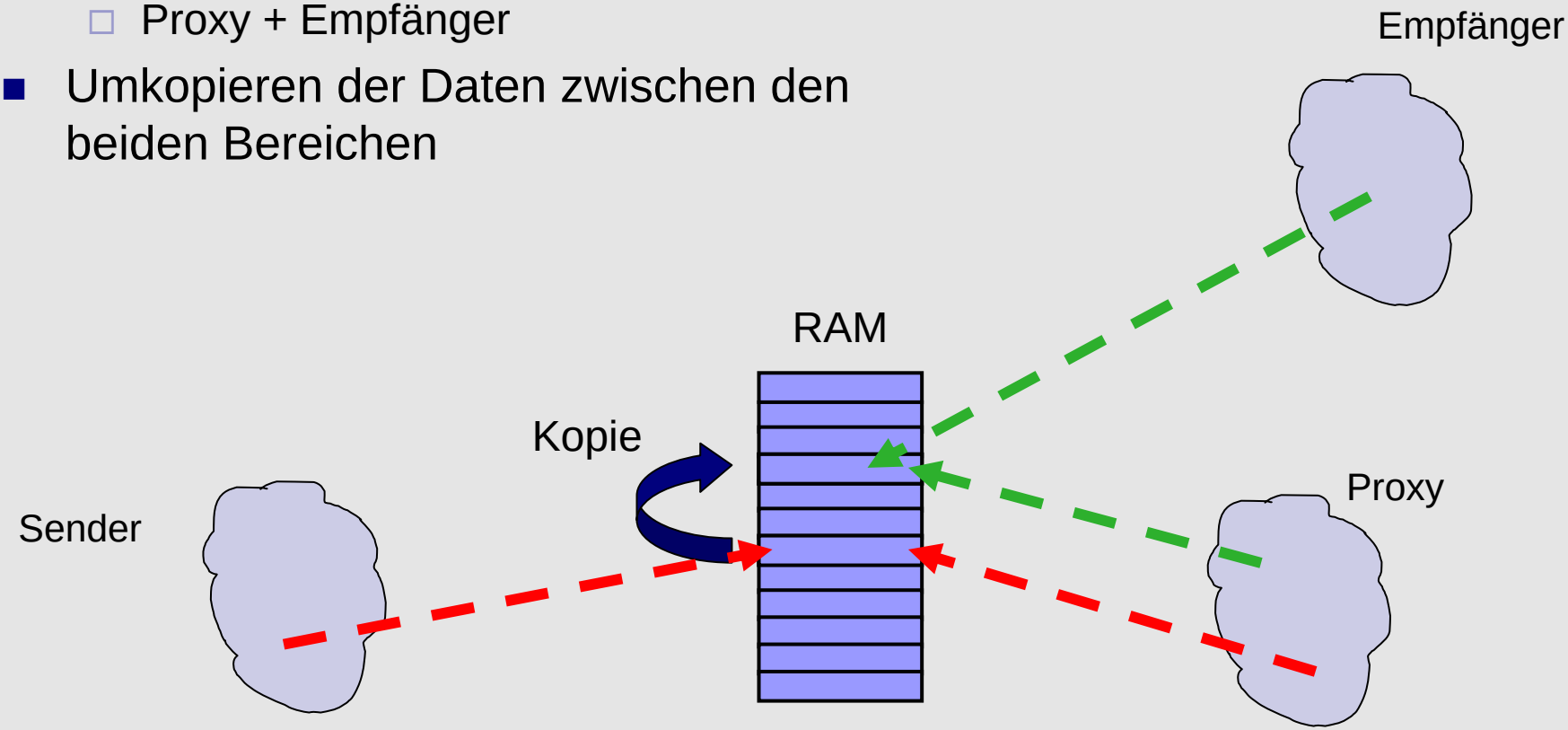
- Anzahl der Nachrichten verdoppelt sich:
  - doppelt so viele Daten pro Nachricht müssen bei Kopieroperationen übertragen werden (größerer Overhead)
  - Nachrichtenwege und somit die Antwortzeiten sind länger (höhere Latenzzeit)
- Proxy muss verwaltet werden

Ablauf der Präsentation:

- Einführung in die Thematik
- Kommunikationsmodell
- Warum asynchrones IPC ?
- **Abstraktes Design des IPC**
- Implementierung in L4
- Kompatibilität bzgl. SOS

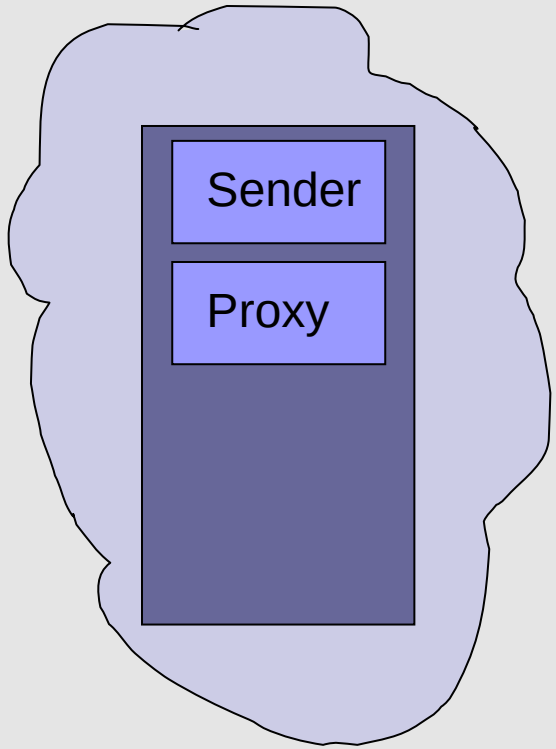
Optimierung: Shared Memory

- „Shared Memory“ zwischen
  - Sender + Proxy (copy on write)
  - Proxy + Empfänger
- Umkopieren der Daten zwischen den beiden Bereichen



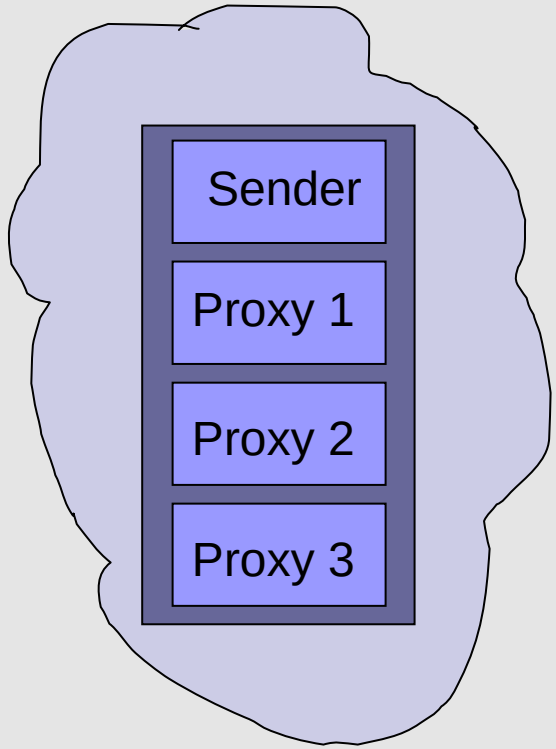
Optimierung: Co-Location

- Integration des Proxy in den Adressraum des Prozesses
  - spart zusätzlichen Kontext-Wechsel
  - Sender nicht mehr von externen Prozessen (Proxy) abhängig



Optimierung: ‚Per Receiver' Proxy

- Ein Proxy pro Empfänger
  - Nachrichten für verschiedene Empfänger können sich nicht mehr blockieren
  - mehr Parallelität



Ablauf der Präsentation:

- Einführung in die Thematik
- Kommunikationsmodell
- Warum asynchrones IPC ?
- Abstraktes Design des IPC
- **Implementierung in L4**
- Kompatibilität bzgl. SOS

Identifizierung

- Jeder Thread ist durch einen systemweit eindeutigen UID gekennzeichnet

  (**U**nique **ID**entifier)

- IPC Nachrichten werden mit Hilfe des UID adressiert

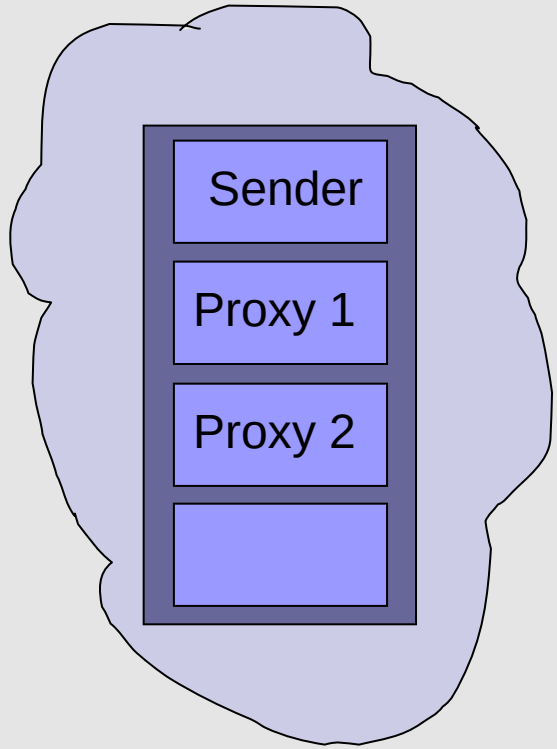
Proxy

- Unterbringung in einem Adressraum

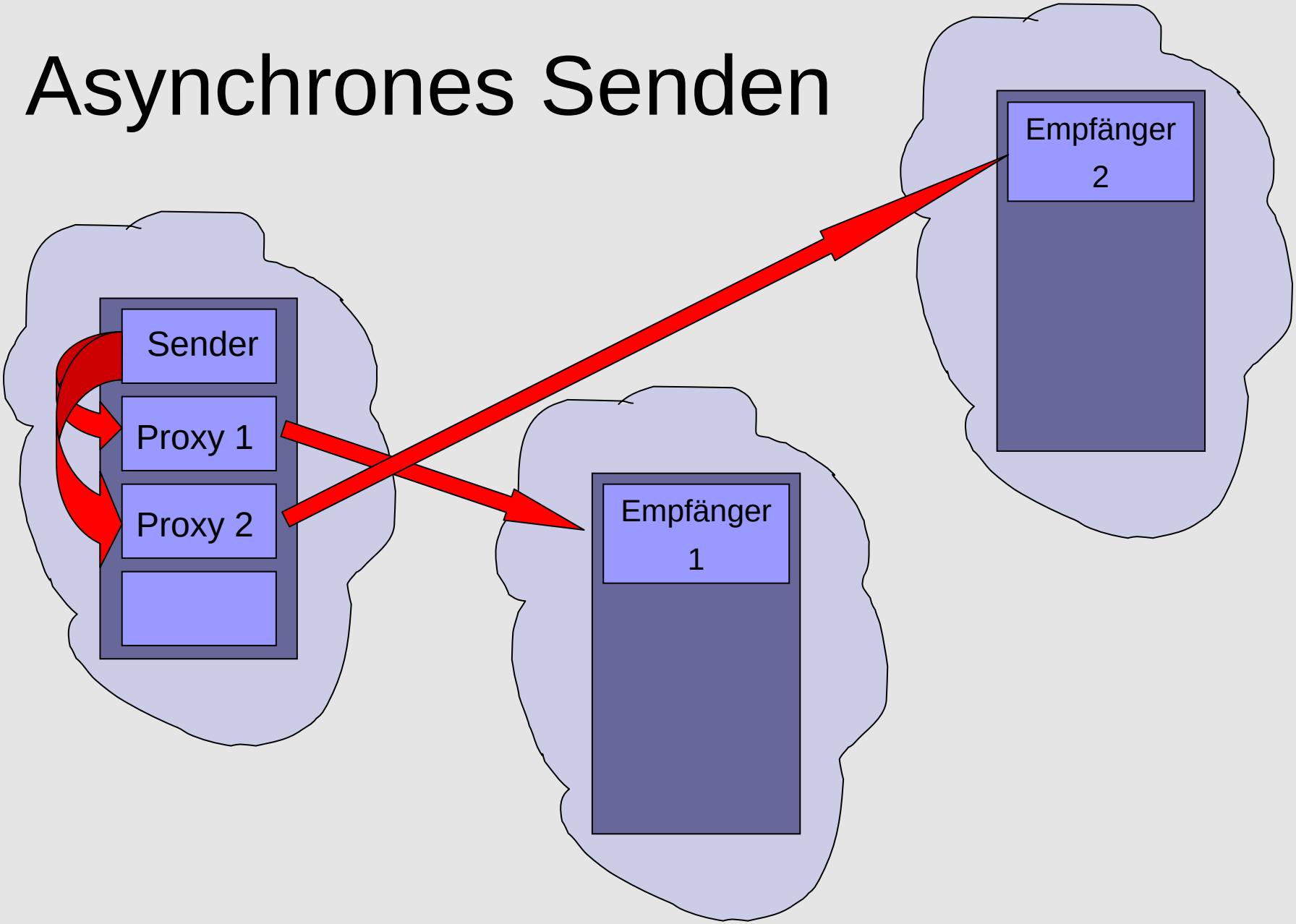
  „*co-located*“

- ein Proxy pro Empfänger

  „*per receiver*“

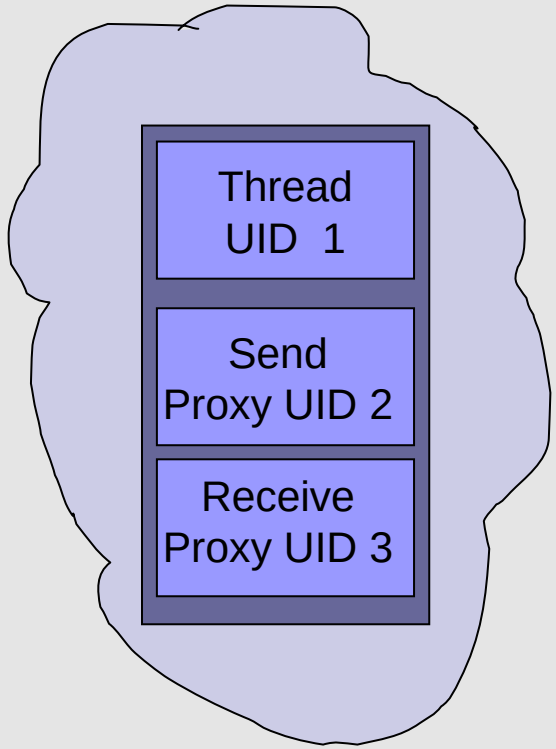


Asynchrones Senden

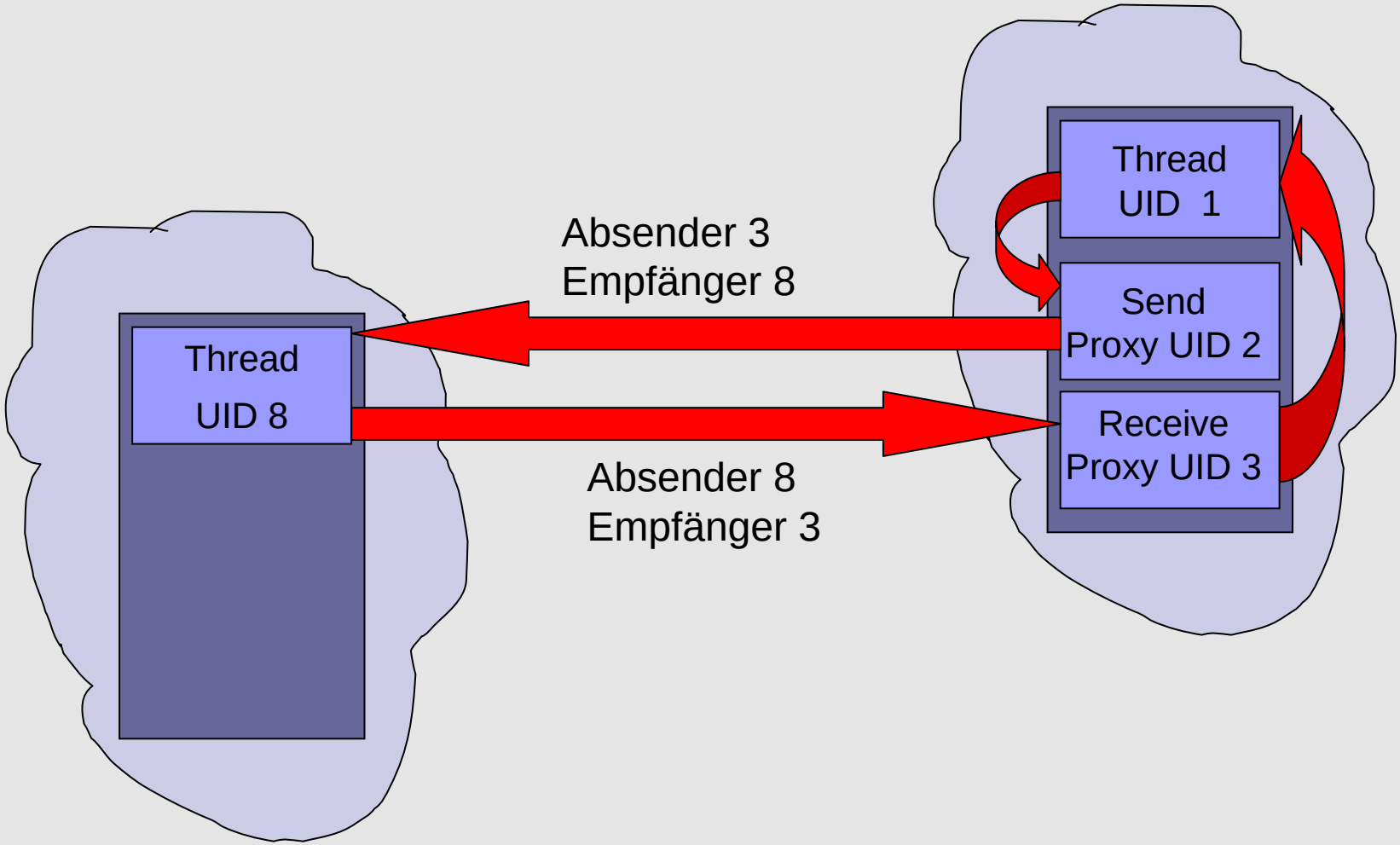


Asynchrones Empfangen bei Antworten

- Dedizierter Empfangs-Proxy
- Vorgehensweise
  a) Ändern der Absender-UID innerhalb der Nachricht
  b) Senden der Nachricht
  c) Receive Proxy empfängt Antwort



Beispiel asynchrones Senden + Empfangen



Prinzipien der Optimierungen

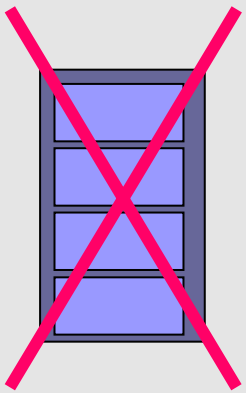
- Address-Space-Sharing
- Address-Space-Sharing
- Address-Space-Sharing

Ablauf der Präsentation:

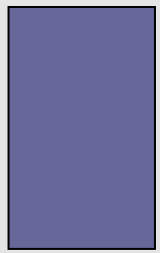
- Einführung in die Thematik
- Kommunikationsmodell
- Warum asynchrones IPC ?
- Abstraktes Design des IPC
- Implementierung in L4
- **Kompatibilität bzgl. SOS**

Prozessmodell

- separate Adressräume für jeden Prozess
- keine Threads
  - „Co-location“ ist **nicht** möglich:
    →Jeder Proxy muss eigenständiger Prozess sein
  - „Per Receiver“ Proxy sehr aufwendig:
    →Bei n Kommunikationspartnern müssten n Proxies bereitgestellt und verwaltet werden



keine Threads
nur eigenständige
Prozesse



Datenübertragung bei Synchronem IPC

- Flex-Pages:
  - C0 arbeitet nicht direkt auf dem Speicher, sondern auf Datenstrukturen
  - Aus Sicht der Prozesse kann keine Aussage darüber getroffen werden, wo im Speicher Platz für diese Datenstrukturen alloziert ist
  - Aus diesem Grunde kann das Konzept von Flex-Pages in SOS nicht eingesetzt werden

Datenübertragung bei Synchronem IPC

- String-Dopes
  - ist die gewählte Form der Datenübertragung
- Register
  - möglich, aber (noch) nicht implementiert

Einschränkung bei IPC

- Aufgrund des restriktiven Typsystems von C0 besitzen insbesondere auch Pointer einen festen Typ
- Für diesen Typ muss zur Kompilierung der Speicherverbrauch bekannt sein
- Aus diesem Grunde haben Puffer in C0 eine fixe Größe

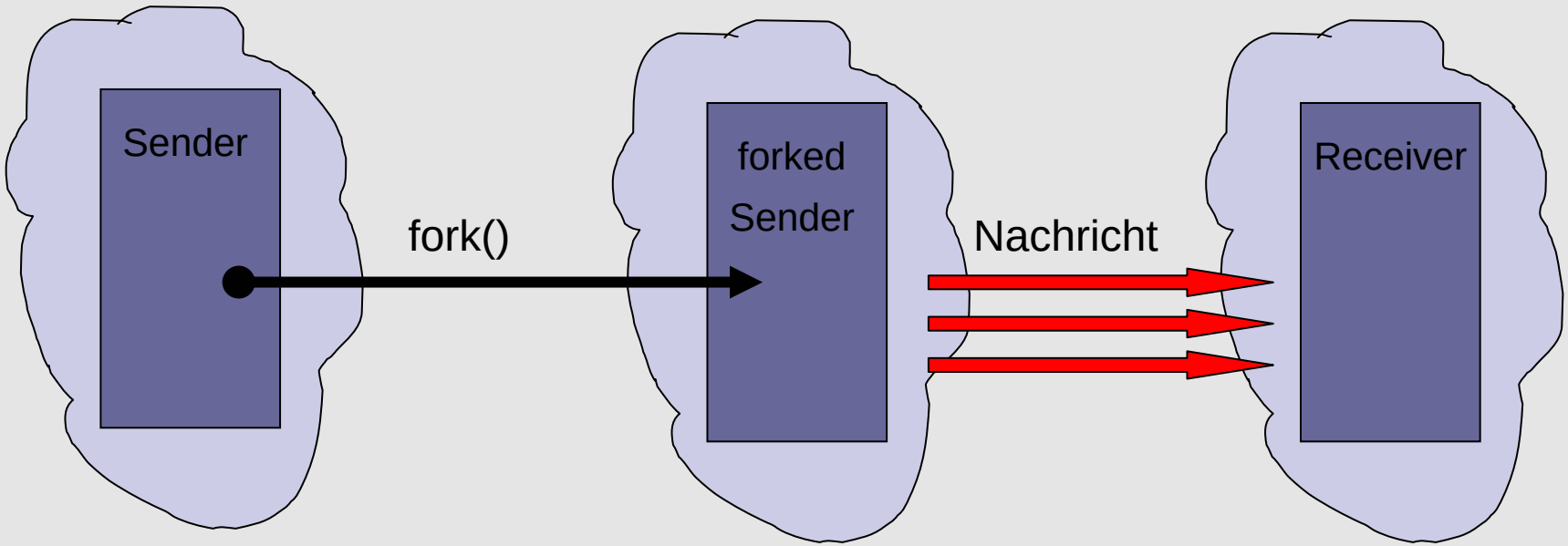
→ Große Daten müssen auf mehrere IPC-Nachrichten aufgeteilt werden

Knackpunkt

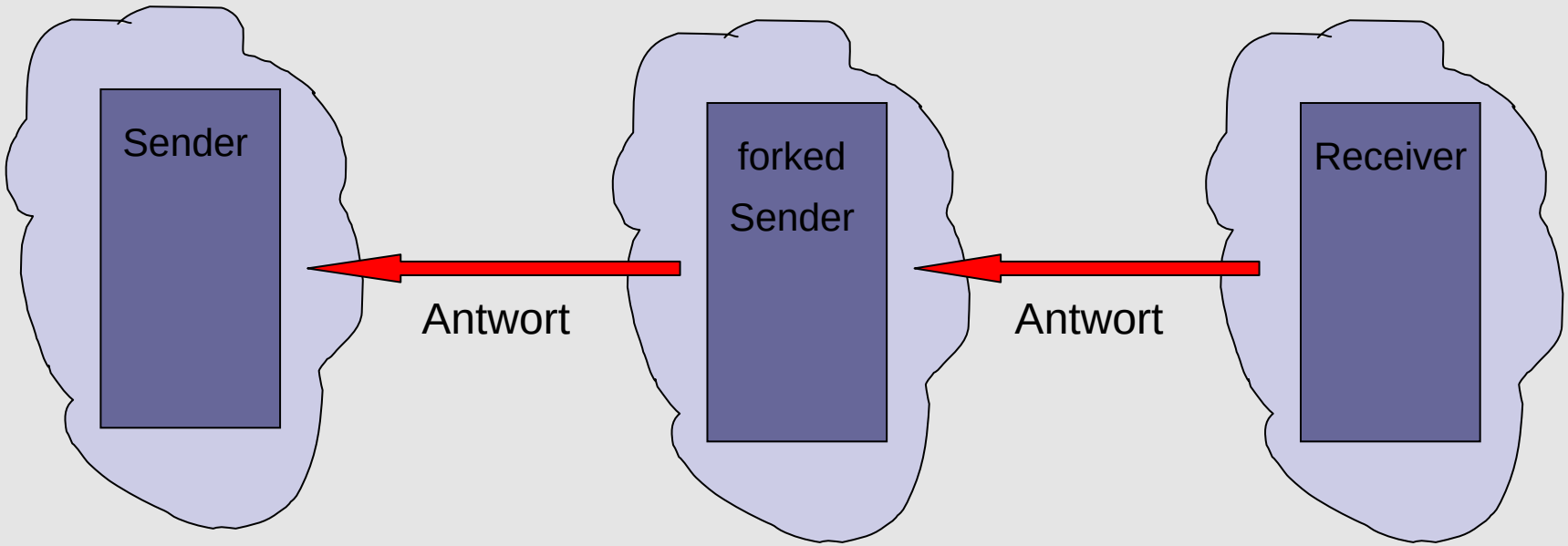
- Datenübertragung zum Proxy
  a) z.Zt. Kopier-Operationen
     → ineffizient, da Daten zweimal kopiert werden müssen

  b) evtl. copy on write (vom Sender zum Proxy)
     → nur Kopie zu Empfänger nötig
     (solange Daten nicht geändert werden)

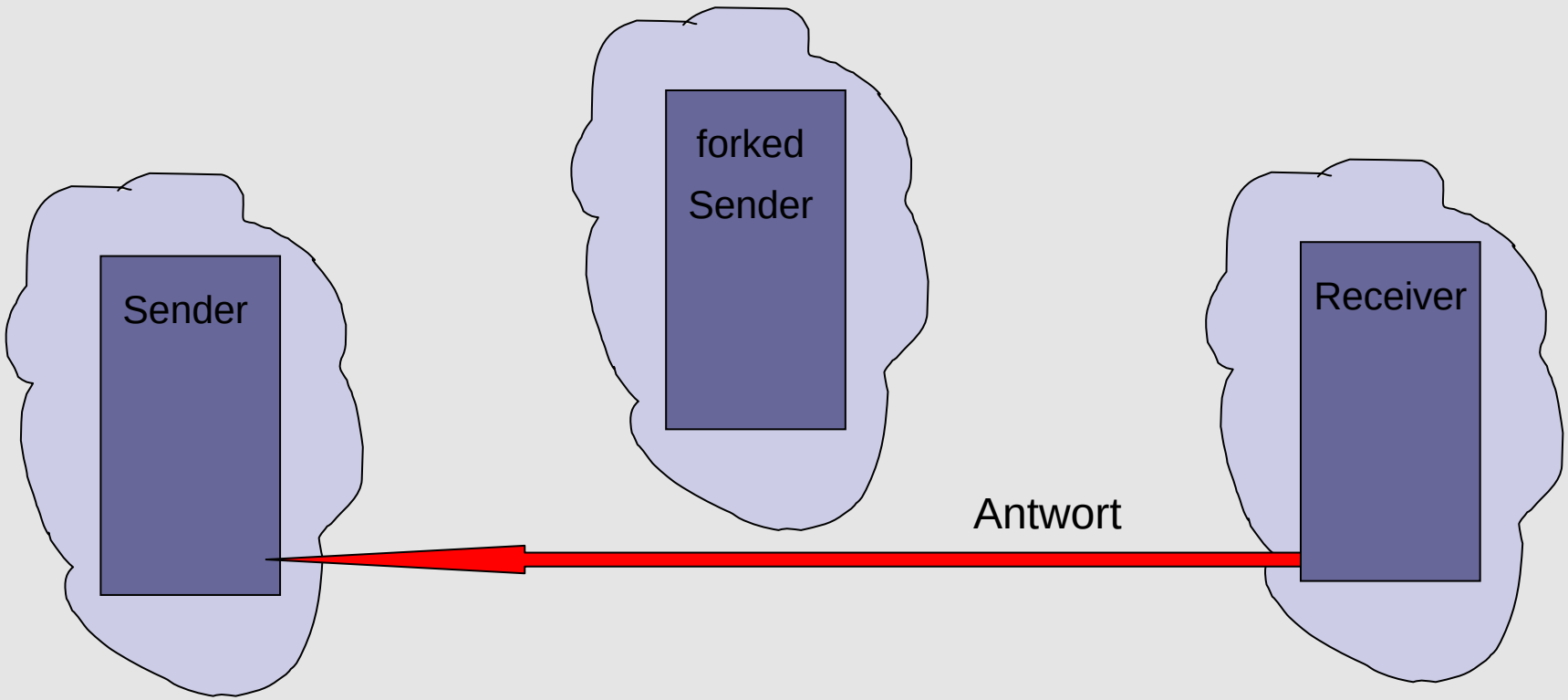
Mögliche Implementierung von *ipc_send_async()*



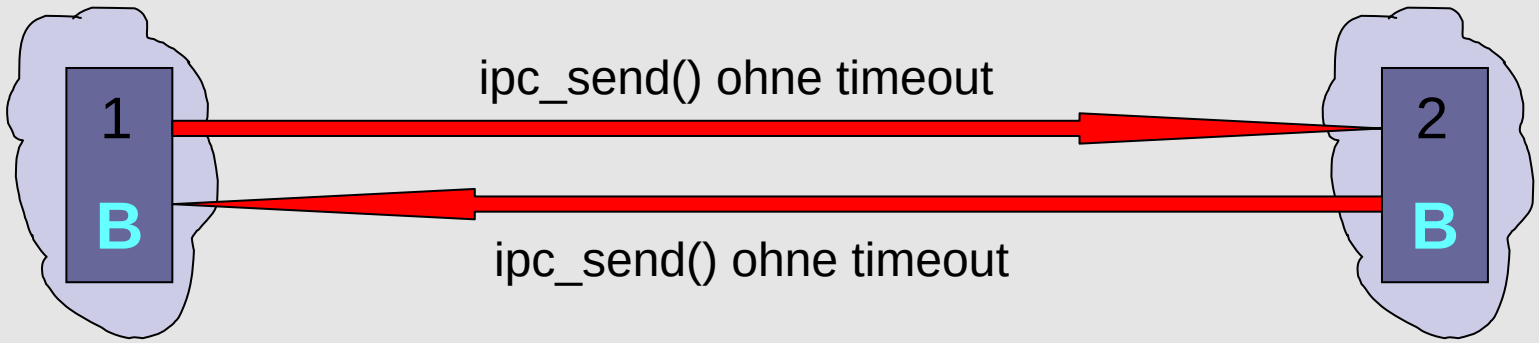
Mögliche Implementierung von *ipc_receive()*



2. Mögliche Implementierung von *ipc_receive()*



Vorteile der 2. *ipc_receive()* Implementierung

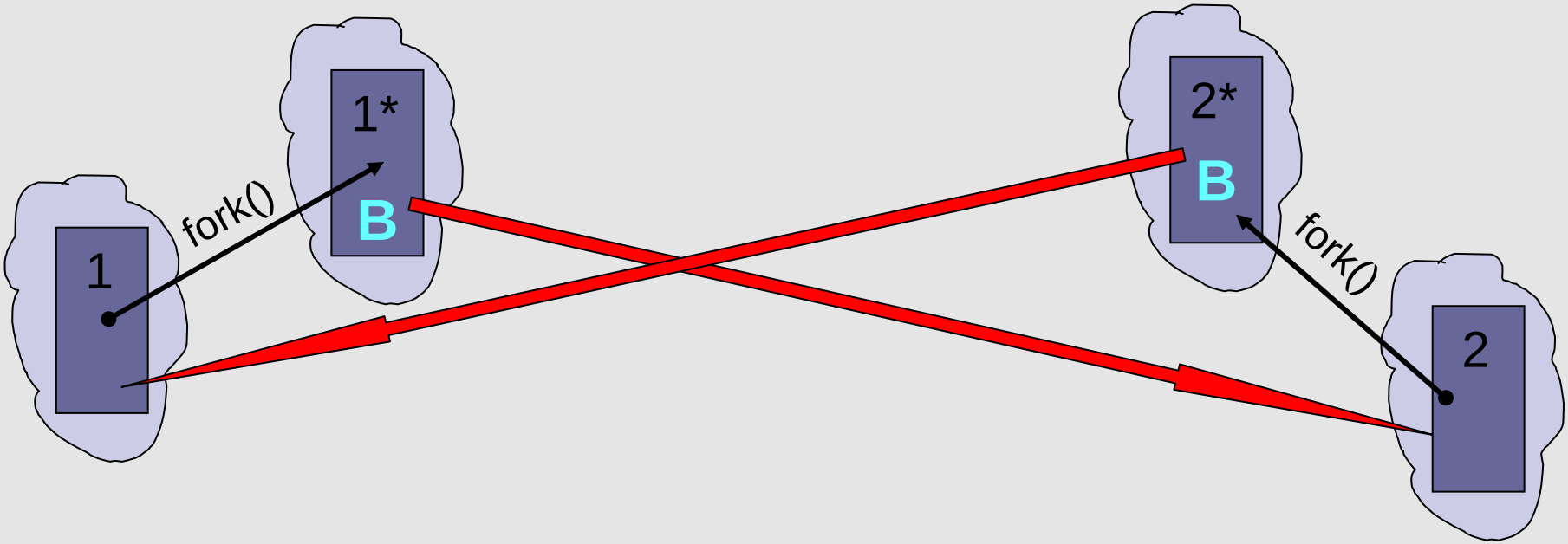


zum gleichem Zeitpunkt ipc_send

→ deadlock

Vorteile der 2. *ipc_receive()* Implementierung

- Trennung von Senden und Empfangen



Vielen Dank
für die Aufmerksamkeit