Übung zu Betriebssystemtechnik

Aufgabe 7: Copy-on-Write

12. Juli 2022

Bernhard Heinloth, Phillip Raffeck & Dustin Nguyen

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

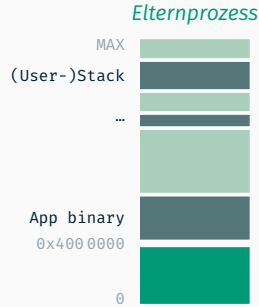
Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

StuBSmI soll durch das Vermeiden von unnötigen Kopieroperationen ressourcenschonender werden

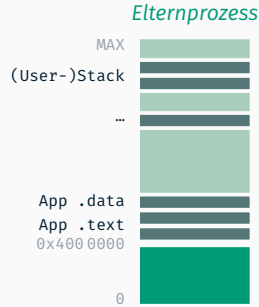
Rekapitulation: Duplizieren eines Prozesses



`fork()`

Beispiel aus Aufgabe 5

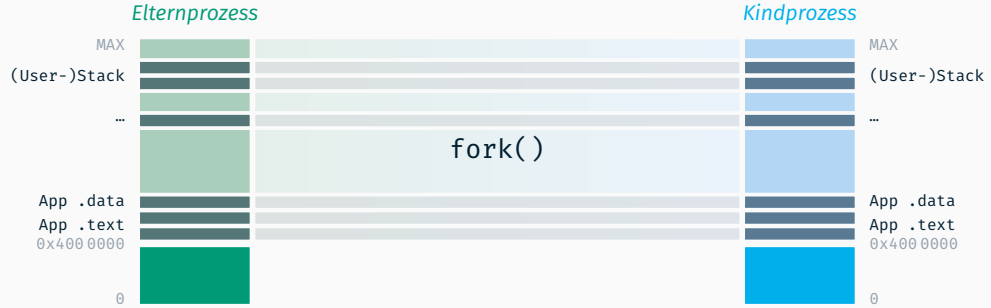
Rekapitulation: Duplizieren eines Prozesses



fork()

Beispiel aus Aufgabe 5 (aber mit Seitengranularität)

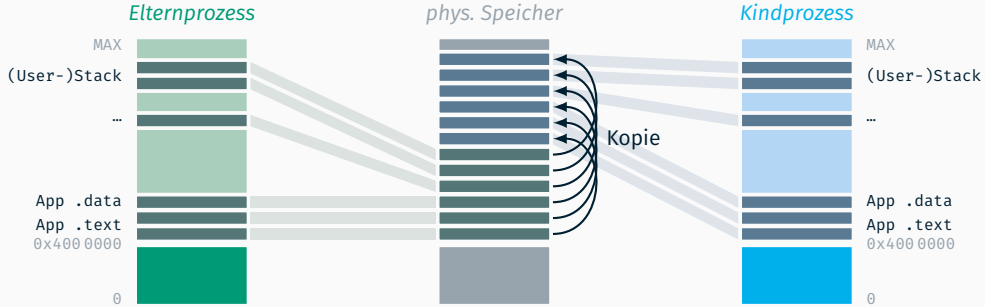
Rekapitulation: Duplizieren eines Prozesses



Beispiel aus Aufgabe 5 (aber mit Seitengranularität)

- `fork()` dupliziert den aktuellen Prozess

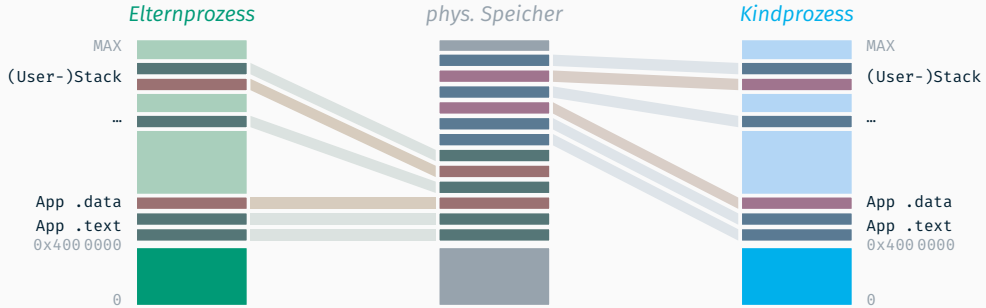
Rekapitulation: Duplizieren eines Prozesses



Beispiel aus Aufgabe 5 (aber mit Seitengranularität)

- `fork()` dupliziert den aktuellen Prozess
- Kindprozess mit (tiefer) Kopie des virt. Speichers

Rekapitulation: Duplizieren eines Prozesses



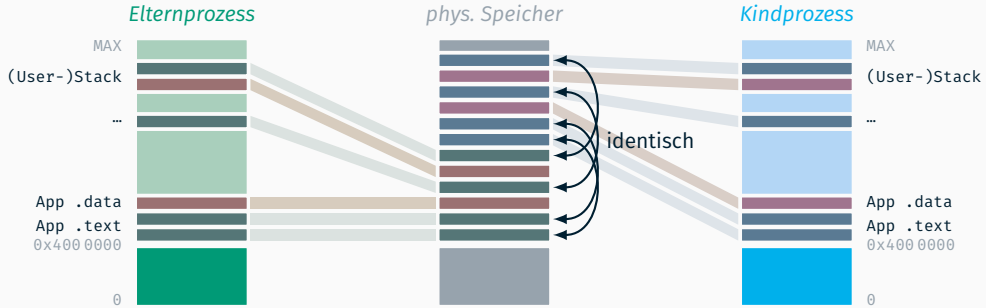
Beispiel aus Aufgabe 5 (aber mit Seitengranularität)

- `fork()` dupliziert den aktuellen Prozess
- Kindprozess mit (tiefer) Kopie des virt. Speichers

Bei der Ausführung wird jedoch nur auf einen Teil schreibend zugegriffen

- *im Beispiel:* Datensegment der App und Stapel

Rekapitulation: Duplizieren eines Prozesses



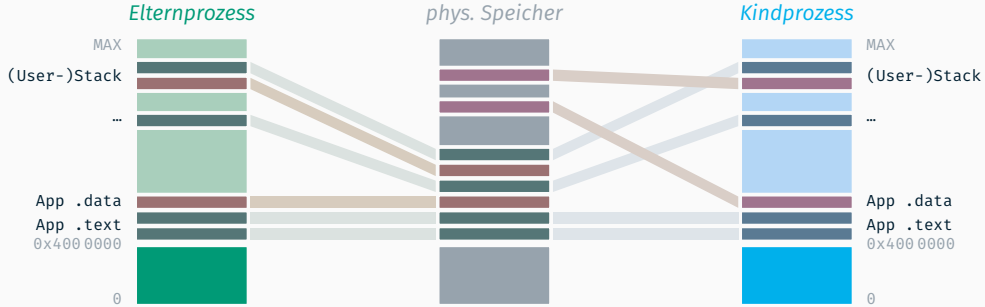
Beispiel aus Aufgabe 5 (aber mit Seitengranularität)

- `fork()` dupliziert den aktuellen Prozess
- Kindprozess mit (tiefer) Kopie des virt. Speichers

Bei der Ausführung wird jedoch nur auf einen Teil schreibend zugegriffen

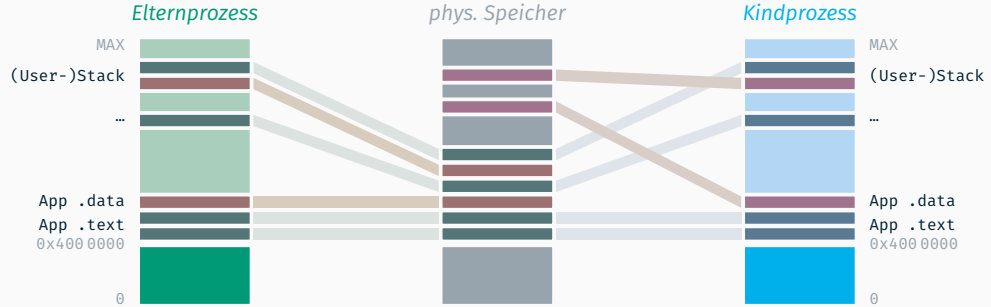
- *im Beispiel:* Datensegment der App und Stapel
- viele kopierte Seiten bleiben weiterhin identisch

Idee: Mitbenutzung identischer Seiten



Wieso nicht gleich identische Seiten mitbenutzen?

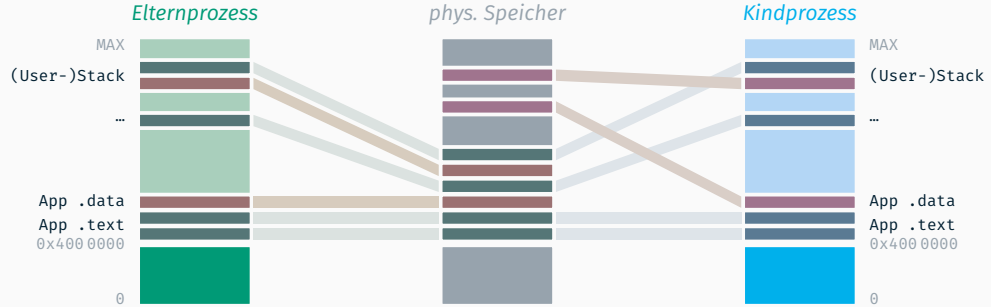
Idee: Mitbenutzung identischer Seiten



Wieso nicht gleich identische Seiten mitbenutzen?

- Vermeidung von (zeitintensiven) Kopieroperationen bei `fork()`
- sparsamere Belegung des (begrenzten) physikalischen Speichers

Idee: Mitbenutzung identischer Seiten

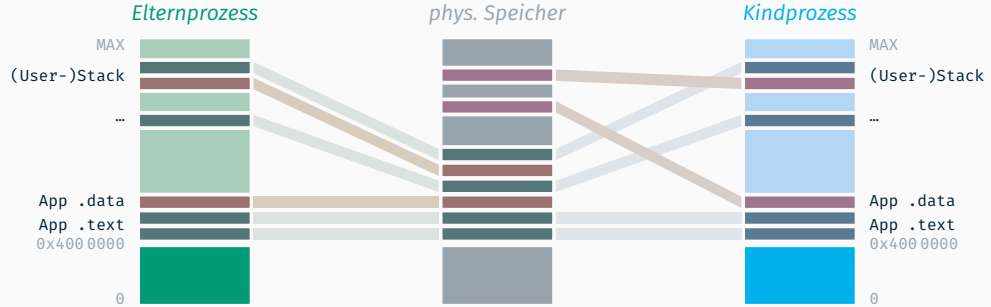


Wieso nicht gleich identische Seiten mitbenutzen?

- Vermeidung von (zeitintensiven) Kopieroperationen bei `fork()`
- sparsamere Belegung des (begrenzten) physikalischen Speichers

Problem: Erkennung von identischen Seiten

Idee: Mitbenutzung identischer Seiten

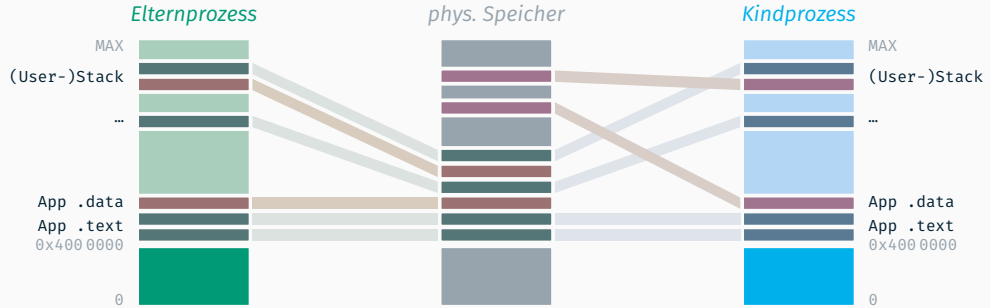


Wieso nicht gleich identische Seiten mitbenutzen?

- Vermeidung von (zeitintensiven) Kopieroperationen bei `fork()`
- sparsamere Belegung des (begrenzten) physikalischen Speichers

Problem: Erkennung von identischen Seiten bzw. geänderten Seiten

Idee: Mitbenutzung identischer Seiten



Wieso nicht gleich identische Seiten mitbenutzen?

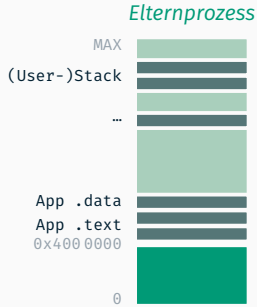
- Vermeidung von (zeitintensiven) Kopieroperationen bei `fork()`
- sparsamere Belegung des (begrenzten) physikalischen Speichers

Problem: Erkennung von identischen Seiten bzw. geänderten Seiten

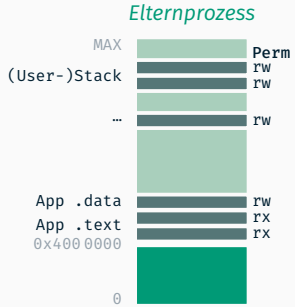
Lösungsansatz: Schreibzugriffe können bei eingeschränkten Berechtigungen (*read-only*) über Seitenfehler erkannt werden

Copy-on-Write

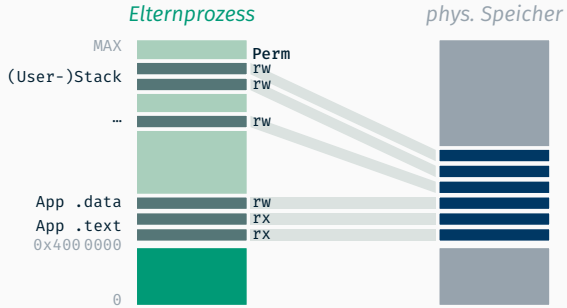
Copy-on-Write am Beispiel `fork()`



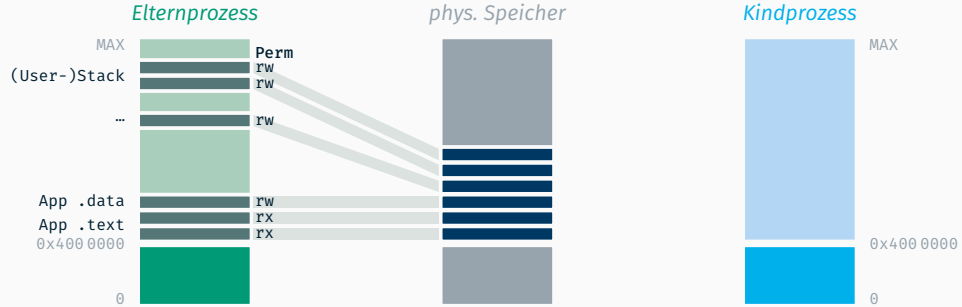
Copy-on-Write am Beispiel `fork()`



Copy-on-Write am Beispiel `fork()`

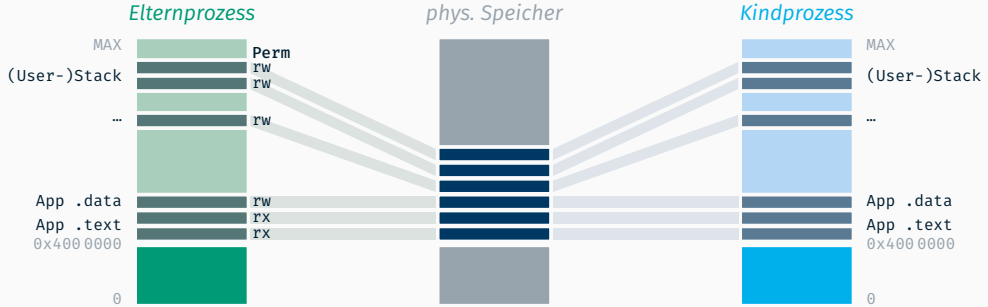


Copy-on-Write am Beispiel `fork()`



`fork()`

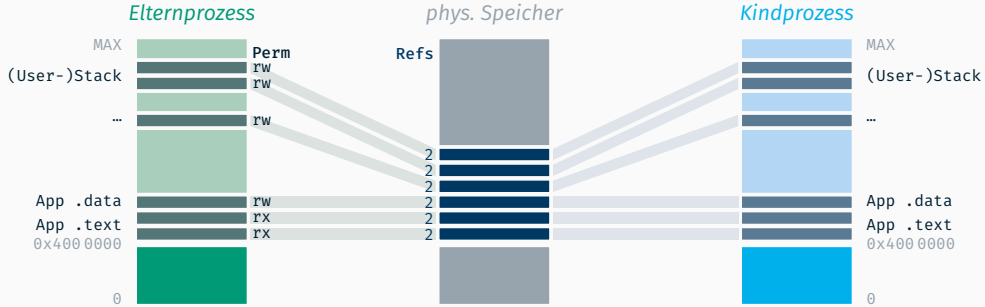
Copy-on-Write am Beispiel `fork()`



`fork()` mit (flacher) Kopie

- Referenz auf die gleiche phys. Seite im Kindprozess

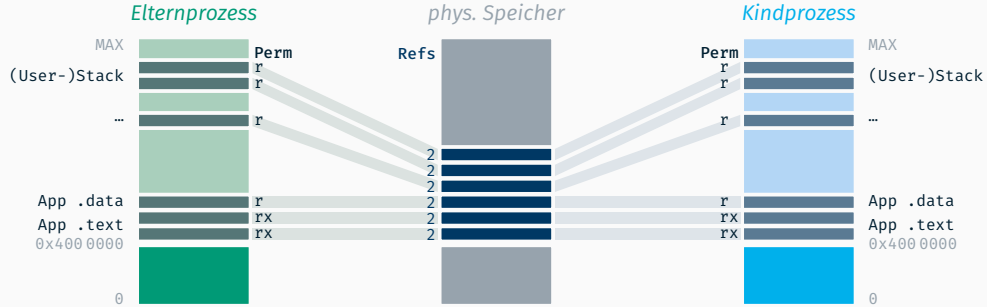
Copy-on-Write am Beispiel `fork()`



`fork()` mit (flacher) Kopie

- Referenz auf die gleiche phys. Seite im Kindprozess
- für alle beteiligte phys. Seiten werden die Referenzen gezählt

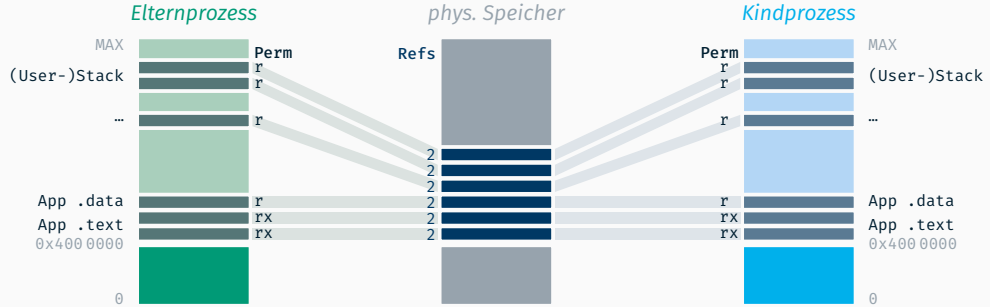
Copy-on-Write am Beispiel `fork()`



`fork()` mit (flacher) Kopie

- Referenz auf die gleiche phys. Seite im Kindprozess
- für alle beteiligte phys. Seiten werden die Referenzen gezählt
- den Prozessen wird Schreibzugriff auf die Seiten entzogen

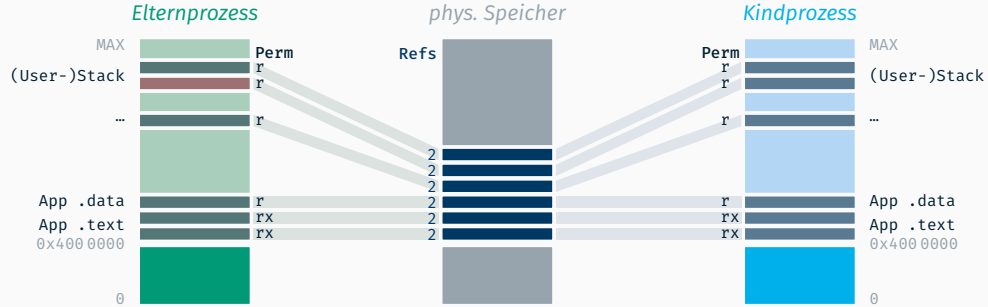
Copy-on-Write am Beispiel `fork()`



`fork()` mit (flacher) Kopie

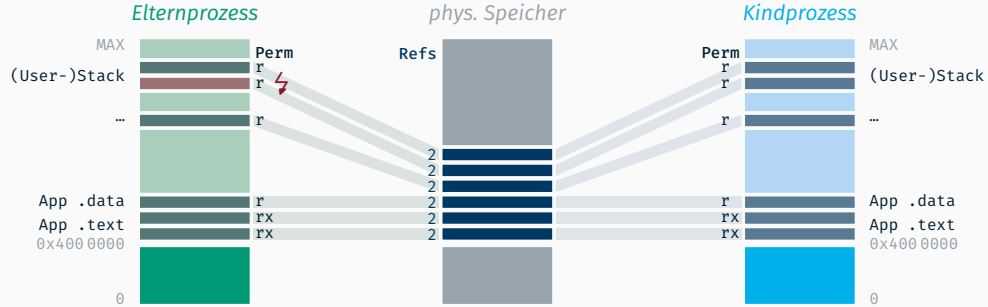
- Referenz auf die gleiche phys. Seite im Kindprozess
- für alle beteiligte phys. Seiten werden die Referenzen gezählt
- den Prozessen wird Schreibzugriff auf die Seiten entzogen (ursprüngliche Zugriffsberechtigung muss aber gemerkt werden!)

Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

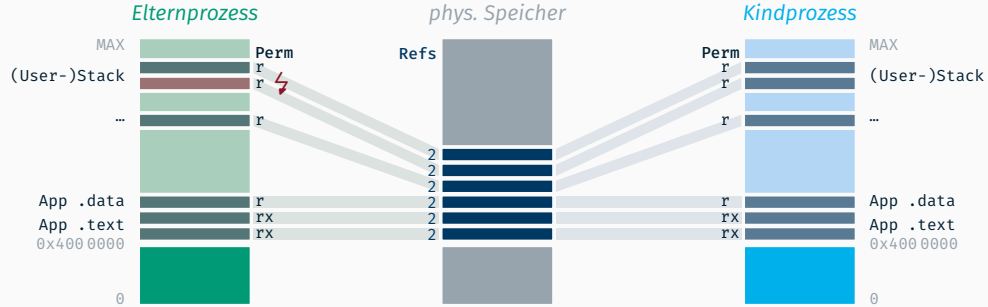
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler

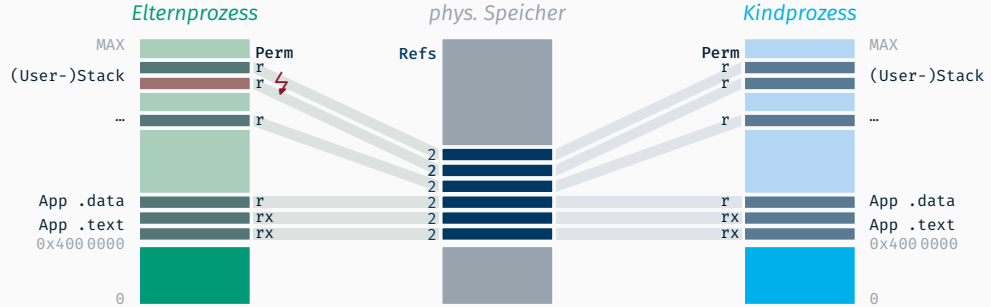
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`

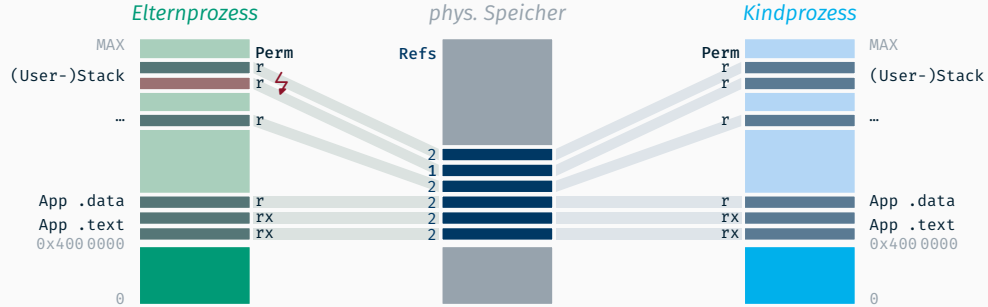
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt

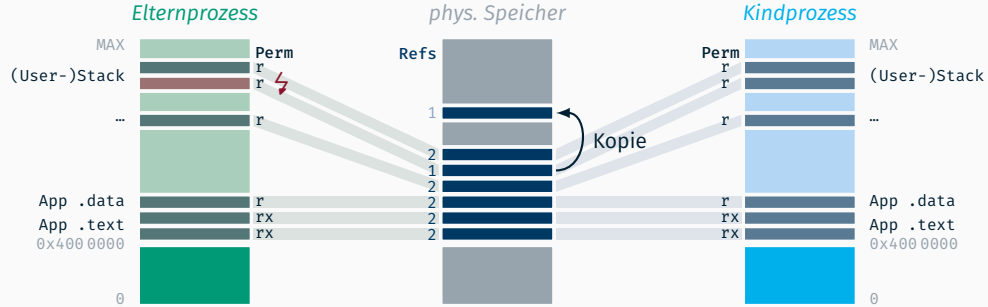
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert

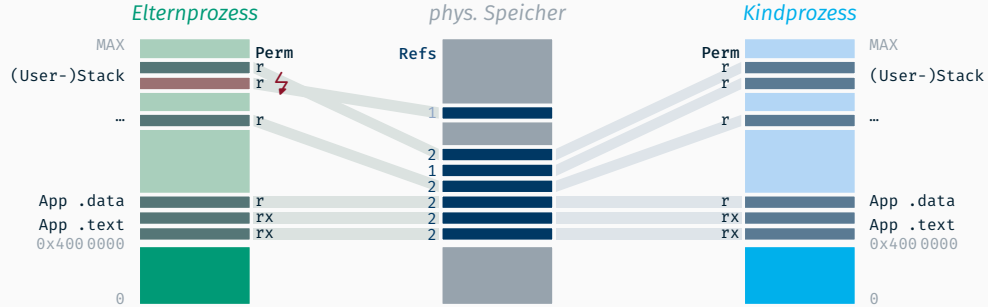
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt

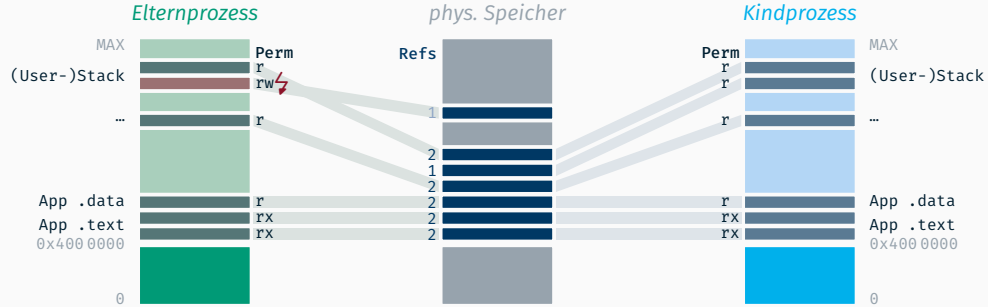
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt und das Mapping angepasst

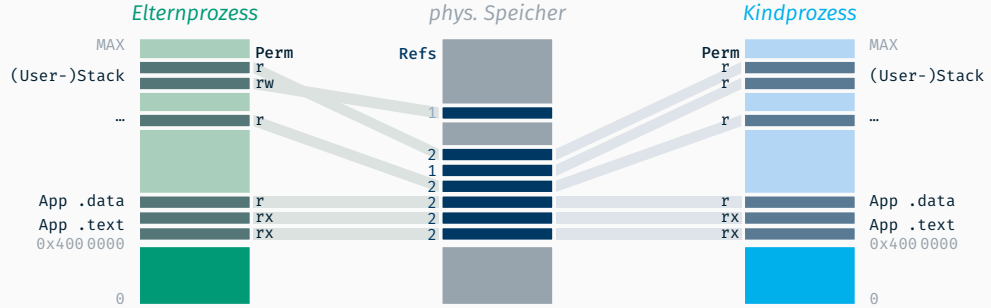
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt und das Mapping angepasst
3. ursprüngliche Zugriffsberechtigung der (Fehler-)Seite gesetzt

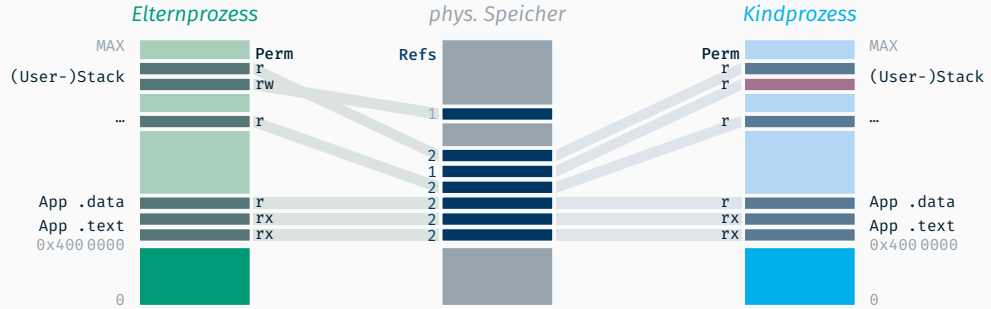
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt und das Mapping angepasst
3. ursprüngliche Zugriffsberechtigung der (Fehler-)Seite gesetzt
4. Ausführung fortgesetzt

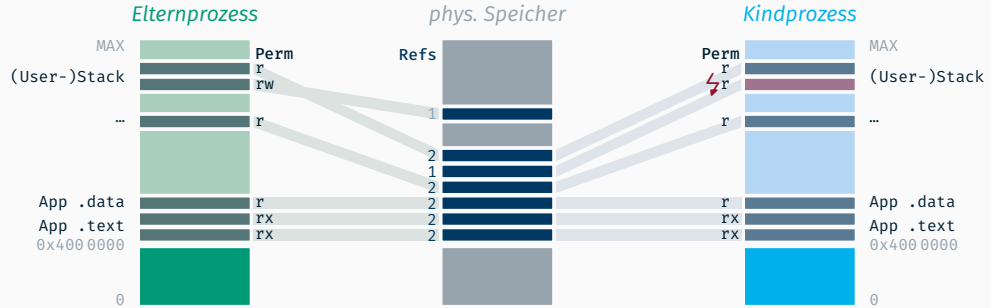
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt und das Mapping angepasst
3. ursprüngliche Zugriffsberechtigung der (Fehler-)Seite gesetzt
4. Ausführung fortgesetzt

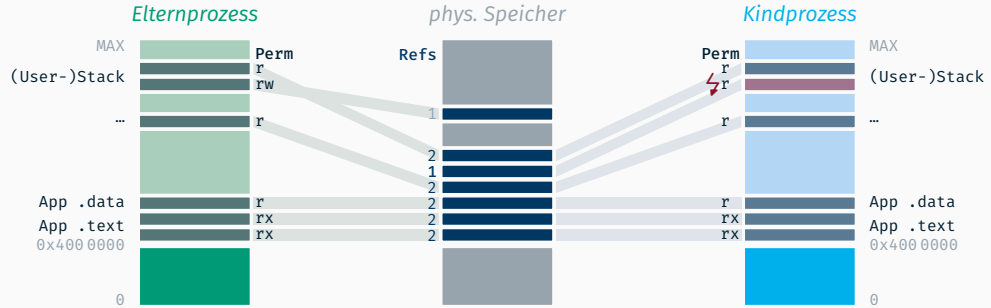
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt und das Mapping angepasst
3. ursprüngliche Zugriffsberechtigung der (Fehler-)Seite gesetzt
4. Ausführung fortgesetzt

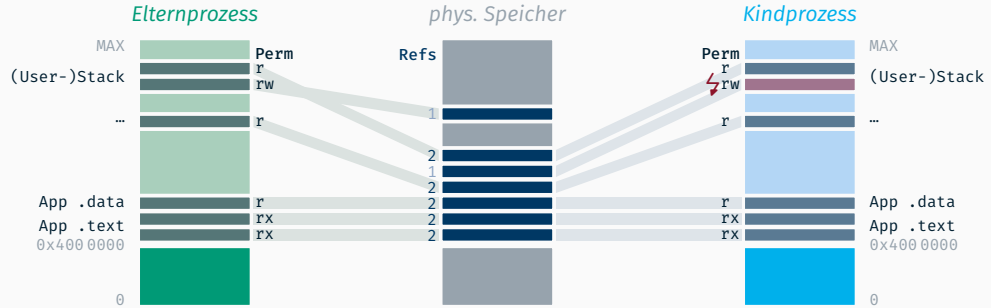
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt und das Mapping angepasst
3. ursprüngliche Zugriffsberechtigung der (Fehler-)Seite gesetzt
4. Ausführung fortgesetzt

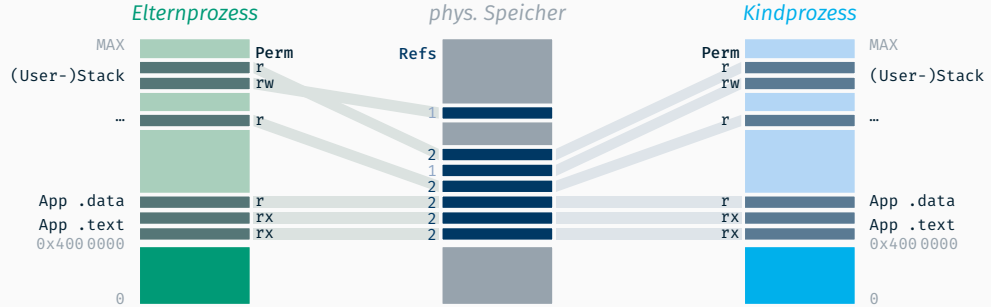
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt und das Mapping angepasst
3. ursprüngliche Zugriffsberechtigung der (Fehler-)Seite gesetzt
4. Ausführung fortgesetzt

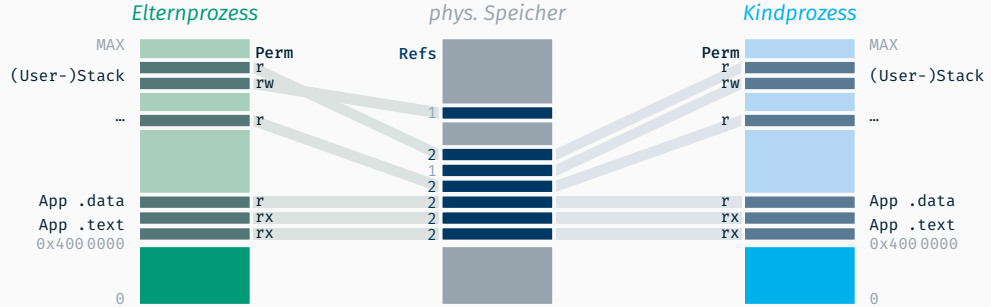
Copy-on-Write am Beispiel `fork()`



Bei einem Schreibzugriff…

1. gibt es einen Seitenfehler ➔ Sprung in `pagefault_handler`
2. sofern es mehr Referenzen der phys. Seite gibt, wird der Zähler dekrementiert, eine Kopie erstellt und das Mapping angepasst
3. ursprüngliche Zugriffsberechtigung der (Fehler-)Seite gesetzt
4. Ausführung fortgesetzt

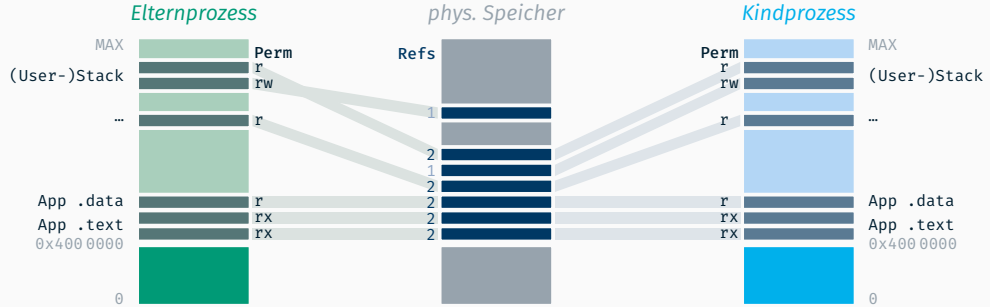
Copy-on-Write am Beispiel `fork()`



Hinweise

- es ist ausreichend, einer phys. Seite erst im Zusammenhang mit einer flachen Kopie einen Referenzzähler zuzuweisen

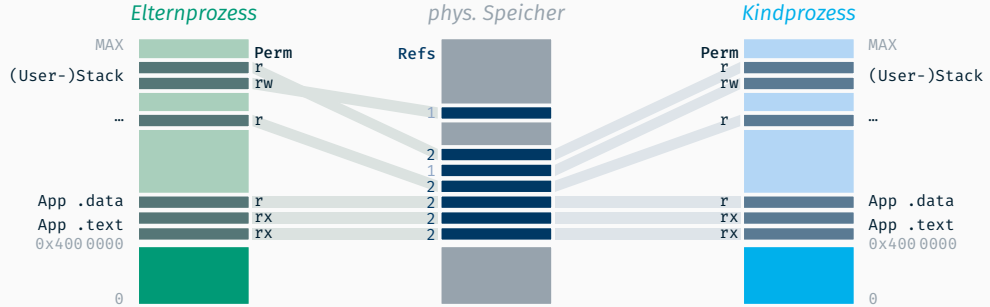
Copy-on-Write am Beispiel `fork()`



Hinweise

- es ist ausreichend, einer phys. Seite erst im Zusammenhang mit einer flachen Kopie einen Referenzzähler zuzuweisen
  ➔ sofern kein Zähler vorhanden ist, gibt es implizit eine Referenz

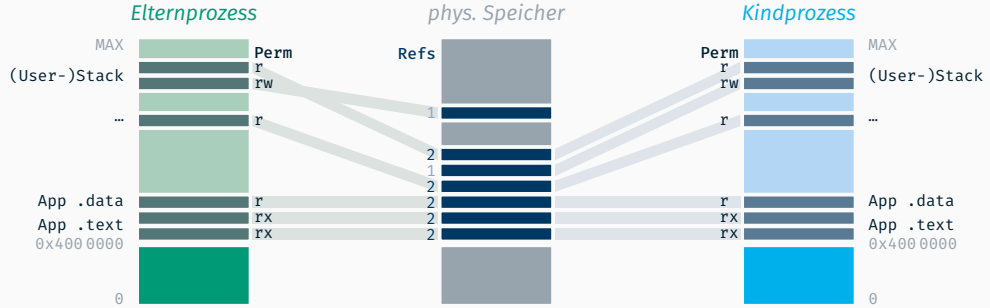
Copy-on-Write am Beispiel `fork()`



Hinweise

- es ist ausreichend, einer phys. Seite erst im Zusammenhang mit einer flachen Kopie einen Referenzzähler zuzuweisen
  ➔ sofern kein Zähler vorhanden ist, gibt es implizit eine Referenz
- bei Anpassungen des Mappings muss TLB gespült werden

Copy-on-Write am Beispiel `fork()`



Hinweise

- es ist ausreichend, einer phys. Seite erst im Zusammenhang mit einer flachen Kopie einen Referenzzähler zuzuweisen
  ➔ sofern kein Zähler vorhanden ist, gibt es implizit eine Referenz
- bei Anpassungen des Mappings muss TLB gespült werden
- Behandlung von [initialen] *read-only*-Seiten genau überlegen!

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmI** für alle Kopieroperationen verwendet werden

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben
- sie eine (oder mehrere) volle Seite(n) umfasst

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben
- sie eine (oder mehrere) volle Seite(n) umfasst

→ **auch für IPC (`send-recv-reply`) nutzbar**

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben
- sie eine (oder mehrere) volle Seite(n) umfasst

→ **auch für IPC (`send-recv-reply`) nutzbar**

- ggf. passende Ausrichtung erzwingen (→ GCC `aligned`-Attribut)

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben
- sie eine (oder mehrere) volle Seite(n) umfasst

→ **auch für IPC (`send-recv-reply`) nutzbar**

- ggf. passende Ausrichtung erzwingen (→ GCC `aligned`-Attribut)
- Kombination aus Copy-on-Write und klassischem `memcpy`

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben
- sie eine (oder mehrere) volle Seite(n) umfasst

→ **auch für IPC (`send-recv-reply`) nutzbar**

- ggf. passende Ausrichtung erzwingen (→ GCC `aligned`-Attribut)
- Kombination aus Copy-on-Write und klassischem `memcpy`
  → Inhalte jenseits der Puffergrenze dürfen nicht kopiert werden!

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben
- sie eine (oder mehrere) volle Seite(n) umfasst

→ **auch für IPC (`send-recv-reply`) nutzbar**

- ggf. passende Ausrichtung erzwingen (→ GCC `aligned`-Attribut)
- Kombination aus Copy-on-Write und klassischem `memcpy`
  → Inhalte jenseits der Puffergrenze dürfen nicht kopiert werden!
- transparente Implementierung bei gutem Software-Engineering

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben
- sie eine (oder mehrere) volle Seite(n) umfasst

→ auch für IPC (`send-recv-reply`) nutzbar

- ggf. passende Ausrichtung erzwingen (→ GCC `aligned`-Attribut)
- Kombination aus Copy-on-Write und klassischem `memcpy`
  → Inhalte jenseits der Puffergrenze dürfen nicht kopiert werden!
- transparente Implementierung bei gutem Software-Engineering
  - generische `copy`-Funktion, welche anhand obiger Kritierien bei jeder Seite zwischen flacher und tiefer Kopie wählt

Weitere Einsatzmöglichkeiten

Copy-on-Write kann in **StuBSmi** für alle Kopieroperationen verwendet werden, solange

- diese im Userspace ist
- Quell- & Zielseite die gleiche Ausrichtung haben
- sie eine (oder mehrere) volle Seite(n) umfasst

→ **auch für IPC (`send-recv-reply`) nutzbar**

- ggf. passende Ausrichtung erzwingen (→ GCC `aligned`-Attribut)
- Kombination aus Copy-on-Write und klassischem `memcpy`
  → Inhalte jenseits der Puffergrenze dürfen nicht kopiert werden!
- transparente Implementierung bei gutem Software-Engineering
  - generische `copy`-Funktion, welche anhand obiger Kritierien bei jeder Seite zwischen flacher und tiefer Kopie wählt
  - keine Anpassung an IPC (oder `fork`) Code notwendig

Umsetzung Referenzzähler

Referenzzähler

Für jede Seite im physikalischen Speicher (ab 64 MiB) wird potenziell ein Referenzzähler gebraucht.

Referenzzähler

Für jede Seite im physikalischen Speicher (ab 64 MiB) wird potenziell ein Referenzzähler gebraucht.

In **StuBSmi** gelten dabei zur Vereinfachung folgende Einschränkungen:

- Kompatibilität bis 100 TiB physikalischer Speicher
- Unterstützung für bis zu einer Billion Referenzen pro Seite

Referenzzähler

Für jede Seite im physikalischen Speicher (ab 64 MiB) wird potenziell ein Referenzzähler gebraucht.

In **StuBSmi** gelten dabei zur Vereinfachung folgende Einschränkungen:

- Kompatibilität bis 100 TiB physikalischer Speicher (47 Bit)
- Unterstützung für bis zu einer Billion Referenzen pro Seite (40 Bit)

Referenzzähler

Für jede Seite im physikalischen Speicher (ab 64 MiB) wird potenziell ein Referenzzähler gebraucht.

In **StuBSmi** gelten dabei zur Vereinfachung folgende Einschränkungen:

- Kompatibilität bis 100 TiB physikalischer Speicher (47 Bit)
- Unterstützung für bis zu einer Billion Referenzen pro Seite (40 Bit)

→ **Verwendung von Schattenseitentabellen (shadow page tables)**

Referenzzähler

Für jede Seite im physikalischen Speicher (ab 64 MiB) wird potenziell ein Referenzzähler gebraucht.

In **StuBSmi** gelten dabei zur Vereinfachung folgende Einschränkungen:

- Kompatibilität bis 100 TiB physikalischer Speicher (47 Bit)
- Unterstützung für bis zu einer Billion Referenzen pro Seite (40 Bit)

→ **Verwendung von Schattenseitentabellen (shadow page tables)**

- bestehende Implementierung (aus Aufgabe 3) wiederverwenden

Referenzzähler

Für jede Seite im physikalischen Speicher (ab 64 MiB) wird potenziell ein Referenzzähler gebraucht.

In **StuBSmi** gelten dabei zur Vereinfachung folgende Einschränkungen:

- Kompatibilität bis 100 TiB physikalischer Speicher (47 Bit)
- Unterstützung für bis zu einer Billion Referenzen pro Seite (40 Bit)

→ **Verwendung von Schattenseitentabellen (shadow page tables)**

- bestehende Implementierung (aus Aufgabe 3) wiederverwenden
- nun aber für physikalische (statt virtuelle) Seitenadressen

Referenzzähler

Für jede Seite im physikalischen Speicher (ab 64 MiB) wird potenziell ein Referenzzähler gebraucht.

In **StuBSmi** gelten dabei zur Vereinfachung folgende Einschränkungen:

- Kompatibilität bis 100 TiB physikalischer Speicher (47 Bit)
- Unterstützung für bis zu einer Billion Referenzen pro Seite (40 Bit)

→ **Verwendung von Schattenseitentabellen (shadow page tables)**

- bestehende Implementierung (aus Aufgabe 3) wiederverwenden
- nun aber für physikalische (statt virtuelle) Seitenadressen
- in der *Page Table* (unterste Ebene) Referenzzähler statt der Zielseitenadresse speichern

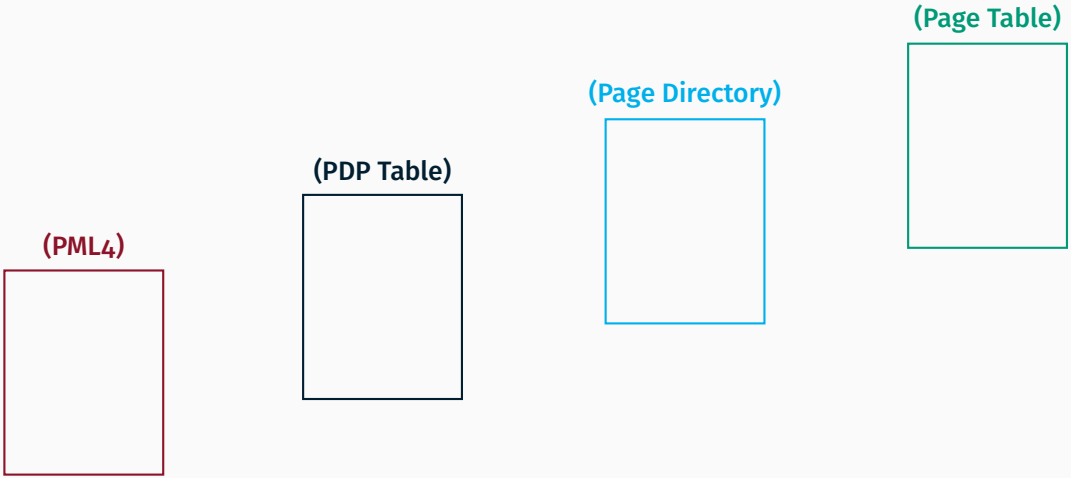
Schattenseitentabellen für Referenzzähler am Beispiel

Physikalische Adresse: `0x5a22306e8f42`

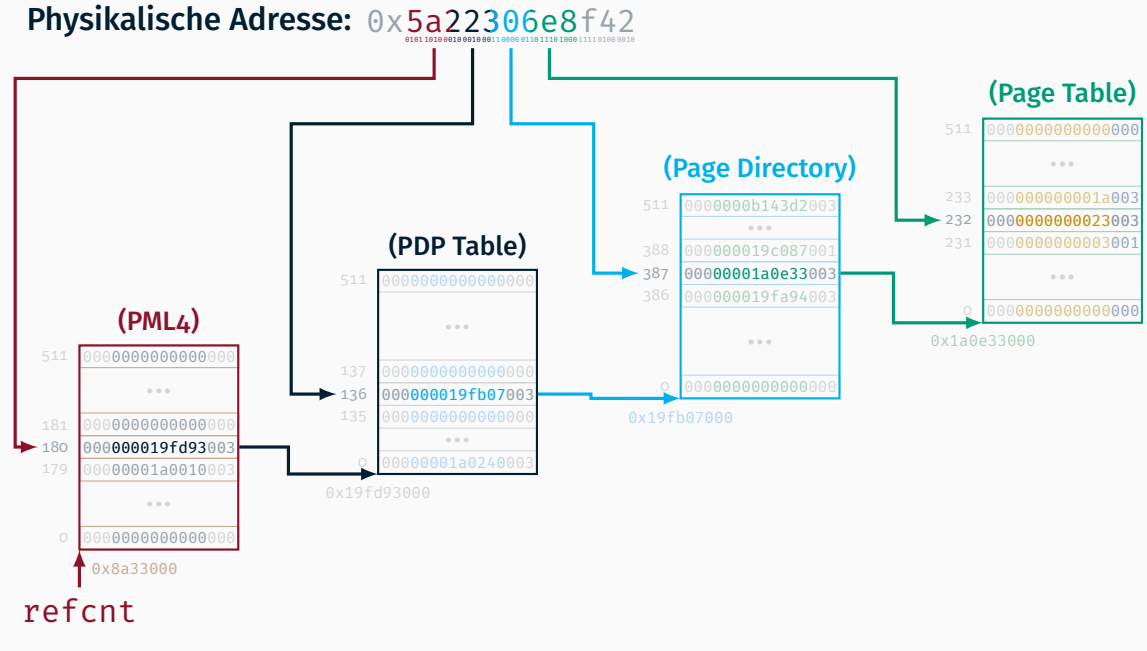
Schattenseitentabellen für Referenzzähler am Beispiel

Physikalische Adresse: `0x5a22306e8f42`

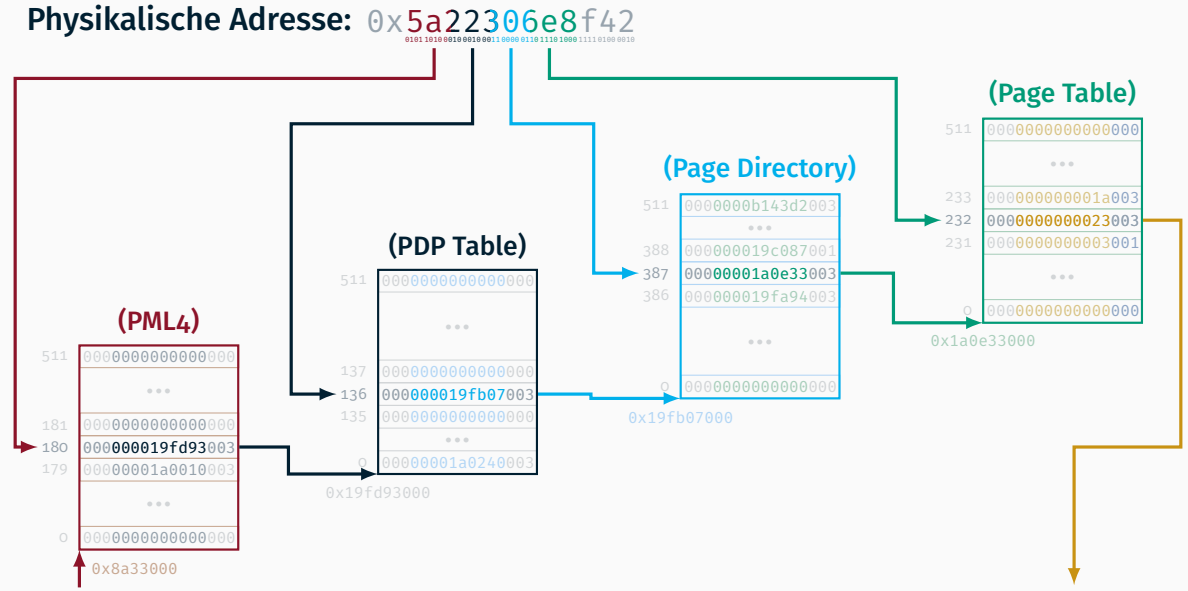
`0101 1010 0010 0010 0011 0000 0110 1110 1000 1111 0100 0010`



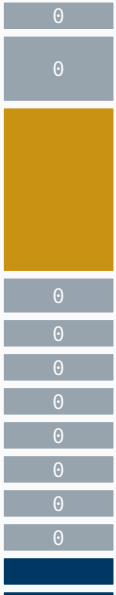
Schattenseitentabellen für Referenzzähler am Beispiel



Schattenseitentabellen für Referenzzähler am Beispiel



Eintrag in der Schattenseitentabelle (Shadow Page-Table)



Execute Disable

ignoriert

~~**Physikalische Adresse** der 4 KiB Zielseite~~

Referenzzähler für die korrespondierende physikalische Seite

ignoriert

Global

Page Size

Dirty

Accessed

Page-Level Cache Disable

Page-Level Write Through

User Mode

Writeable: nur lesender (0) oder auch schreibender (1) Zugriff

Present: Eintrag aktiv (1) oder inaktiv (0)

Fragen?

Fast geschafft – letzte Aufgabe!

Fragen?

Fast geschafft – letzte Aufgabe!

- Freitag, 15. Juli: keine Rechnerübung
- Dienstag, 19. Juli: ISER Vorführung
- Dienstag, 26. Juli: Zusammenfassung, Besprechung der Evaluation