# Betriebssystemtechnik

*Adressräume: Trennung, Zugriff, Schutz*

## X. Spezialfälle: Adaptiver Speicherschutz

Wolfgang Schröder-Preikschat / Volkmar Sieh

21. Juni 2023

# Gliederung



# Bedarfssynchrone Prozessisolation

*„just in time“*

# Bedarfssynchrone Prozessisolation

*„just in time“*

- **gemeinsamer Hauptspeicher** (*shared memory*) bildet die typische Basis für mehr-/vielkernige Prozessoren
  - dabei verfügt jeder einzelne Rechenkern über eine eigene MMU
  - wichtiger Bestandteil jeder MMU ist der TLB
    - *translation lookaside buffer*, auch bekannt als **Übersetzungspuffer** ☺

# Bedarfssynchrone Prozessisolation

*„just in time“*

- **gemeinsamer Hauptspeicher** (*shared memory*) bildet die typische Basis für mehr-/vielkernige Prozessoren

- für derartige Prozessoren sollten Anwendungen **mehr-/vielfädige Prozesse** ermöglichen, um in ihrer Leistung skalieren zu können [9]
  - verteilt über die Rechenkerne finden diese Prozesse gleichzeitig statt
    - sie bilden sogenannte **gleichzeitige Prozesse** ☺
  - dabei teilen sich die Prozessfäden einer Anwendung denselben Adressraum

# Bedarfssynchrone Prozessisolation

*„just in time“*

- **gemeinsamer Hauptspeicher** (*shared memory*) bildet die typische Basis für mehr-/vielkernige Prozessoren
- für derartige Prozessoren sollten Anwendungen **mehr-/vielfädige Prozesse** ermöglichen, um in ihrer Leistung skalieren zu können [9]
- bewirkt eine Fadenaktion eine Adressraumänderung, ist diese auf den Rechenkernen der anderen Fäden der Anwendung nachzuziehen
  - erzwungenes Unterbrechen von Prozessoren für TLB-Konsistenzaktionen
  - ein wahres „Niederschießen“ von Rechenkernen: „***TLB shootdown***“ [2]

# Bedarfssynchrone Prozessisolation

*„just in time“*

- **gemeinsamer Hauptspeicher** (*shared memory*) bildet die typische Basis für mehr-/vielkernige Prozessoren
- für derartige Prozessoren sollten Anwendungen **mehr-/vielfädige Prozesse** ermöglichen, um in ihrer Leistung skalieren zu können [9]
- bewirkt eine Fadenaktion eine Adressraumänderung, ist diese auf den Rechenkernen der anderen Fäden der Anwendung nachzuziehen
- Adressraumisolation von Prozessen ist ein **Anwendungskriterium**, das nur im Bedarfsfall vom System durchgesetzt wird [5, 4]

# Gliederung



# Speicherschutz

- **Sicherheit** (*security*)
  - Schutz einer Entität vor ihrer Umwelt

# Speicherschutz

- **Sicherheit** (*security*)
  - Schutz einer Entität vor ihrer Umwelt
- differenzierte Zugriffskontrolle
  - Text, Daten, Stapel, ...
  - lesen, schreiben, ausführen, ...

# Speicherschutz

- **Sicherheit** (*security*)
  - Schutz einer Entität vor ihrer Umwelt

- erreichbar durch:
  - **Überwachung** der Adressen
    - typisch für Mehradressraumsysteme
  - **Randomisierung** von Adressbereichen
    - möglich für Einadressraumsysteme
  - **Typsicherheit** der Programme
    - abhängig von Sprache/Kompilierer

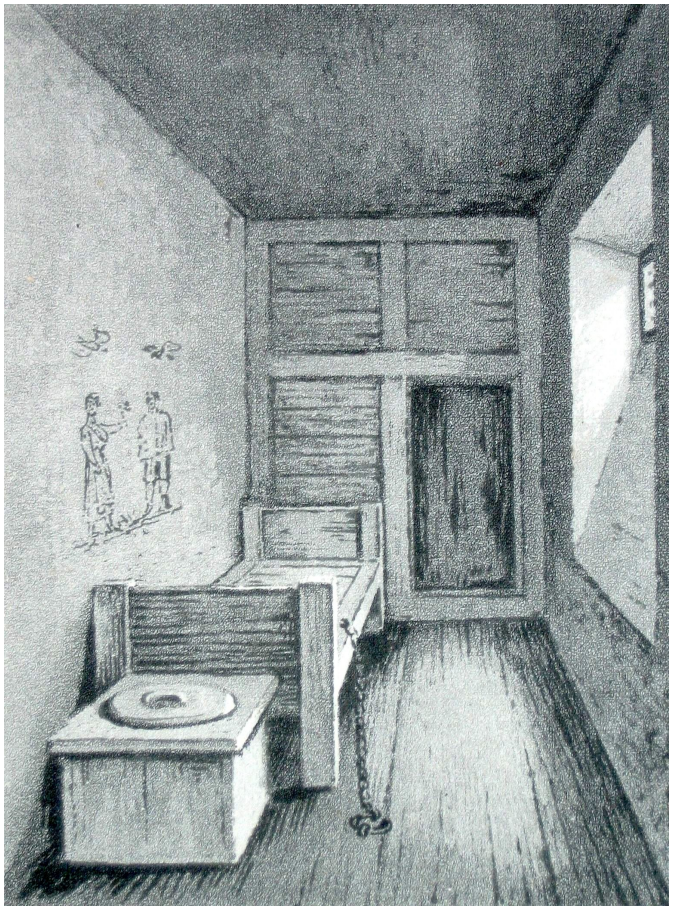
# Speicherschutz

- **Sicherheit** (*security*)
  - Schutz einer Entität vor ihrer Umwelt

- „benutzt“ [8] Befehlssatzebene und. . .
  - Betriebssystemebene oder Kompilierer
  - je nachdem ob Typsicherheit aller Programme gewährleistet ist

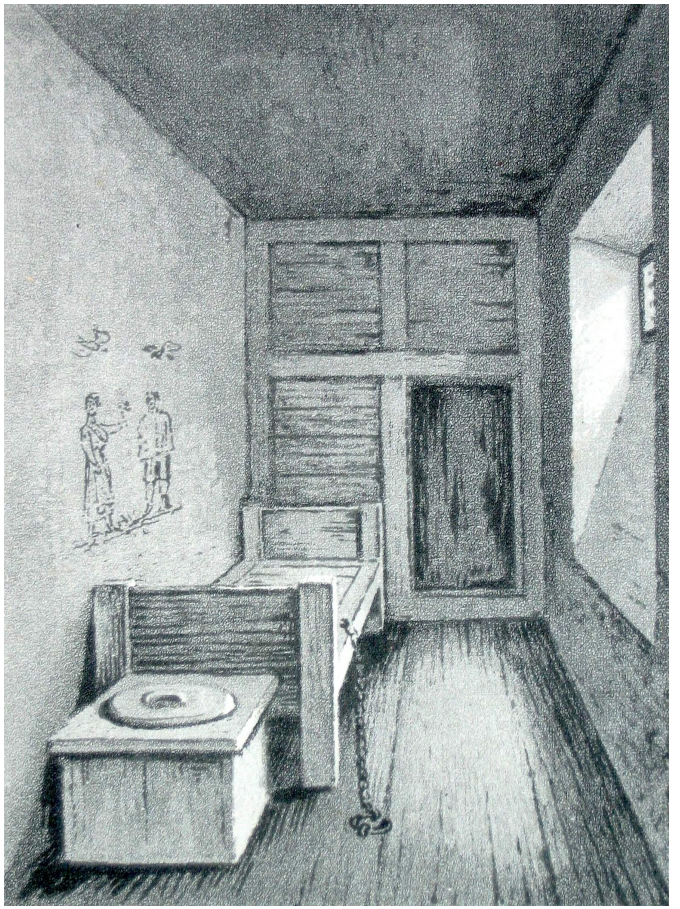
# Adressraumisolation

- [Betriebs-]**Sicherheit** (*safety*)
  - Schutz der Umwelt vor einer Entität



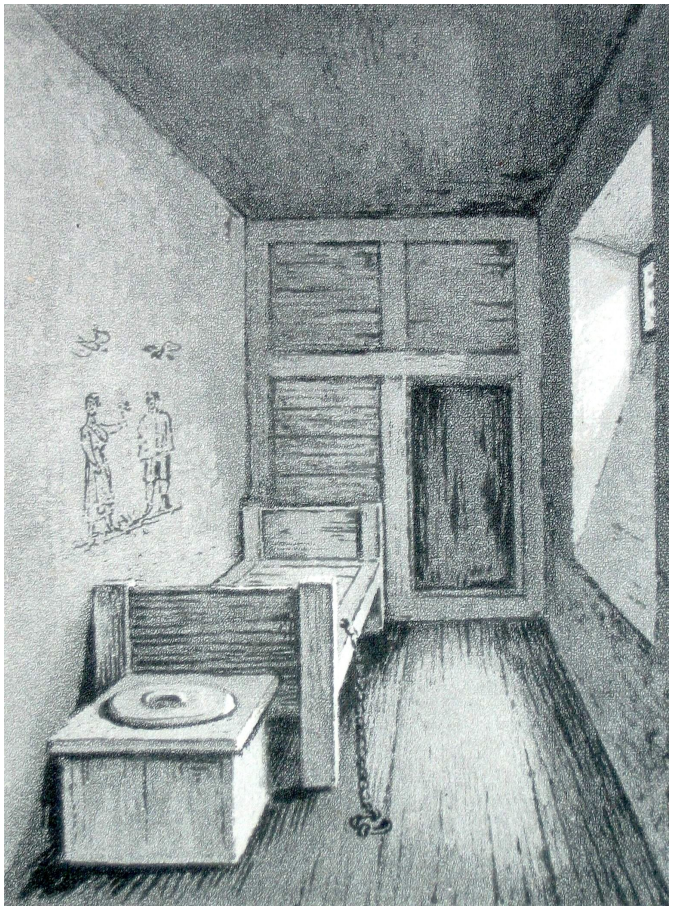
# Adressraumisolation

- [Betriebs-]**Sicherheit** (*safety*)
  - Schutz der Umwelt vor einer Entität
- softwarebasiert
  - vollständige Interpretation der Befehle
    - CSIM virtuelle Maschine [6, S. 21]
  - **Kompilierung**
    - typsichere Programmiersprache



# Adressraumisolation

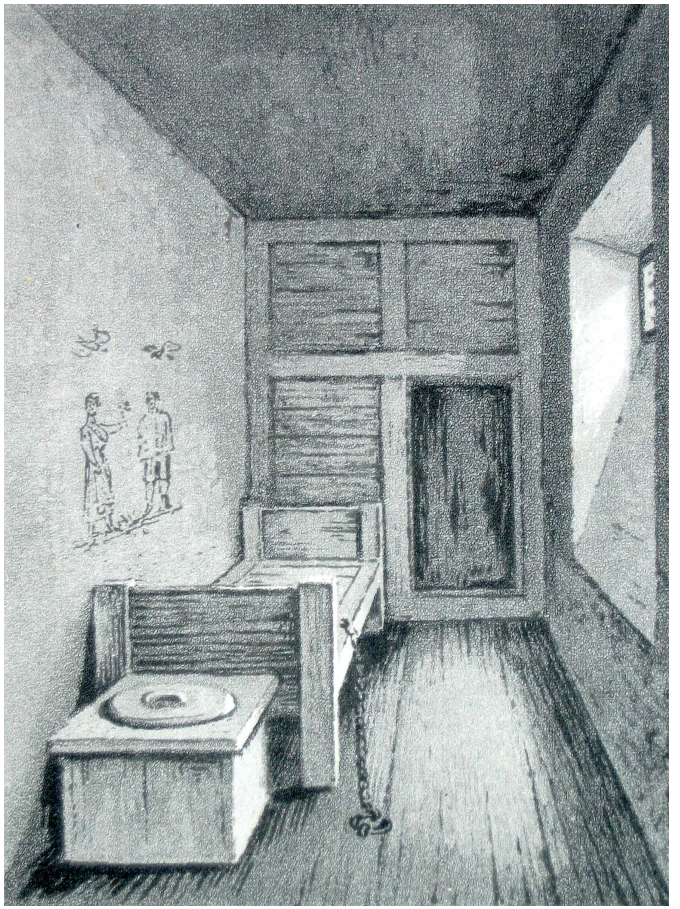
- [Betriebs-]**Sicherheit** (*safety*)
  - Schutz der Umwelt vor einer Entität
- softwarebasiert



- hardwarebasiert
  - MMU- oder MPU-basiert
  - **partielle Interpretation** der Zugriffe
    - im Ausnahmefall (*trap*)
    - durch das Betriebssystem

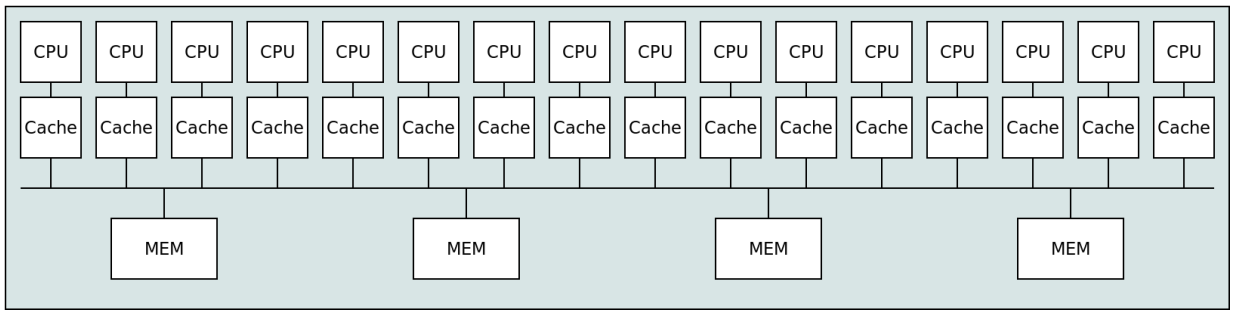
# Adressraumisolation

- [Betriebs-]**Sicherheit** (*safety*)
  - Schutz der Umwelt vor einer Entität
- softwarebasiert
- hardwarebasiert
- „benutzt“ [8] Befehlssatzebene und. . .
  - Betriebssystemebene oder Kompilierer
  - je nachdem ob Typsicherheit aller Programme gewährleistet ist

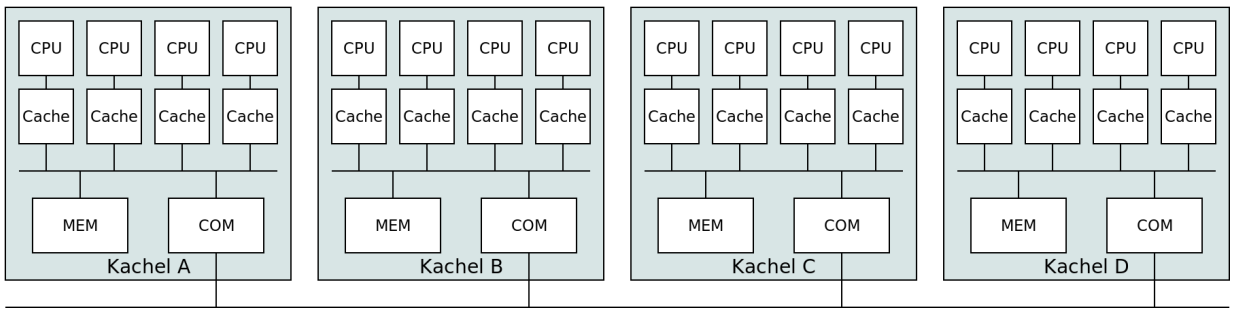


# Vielkernprozessoren

## UMA: Uniform Memory Access

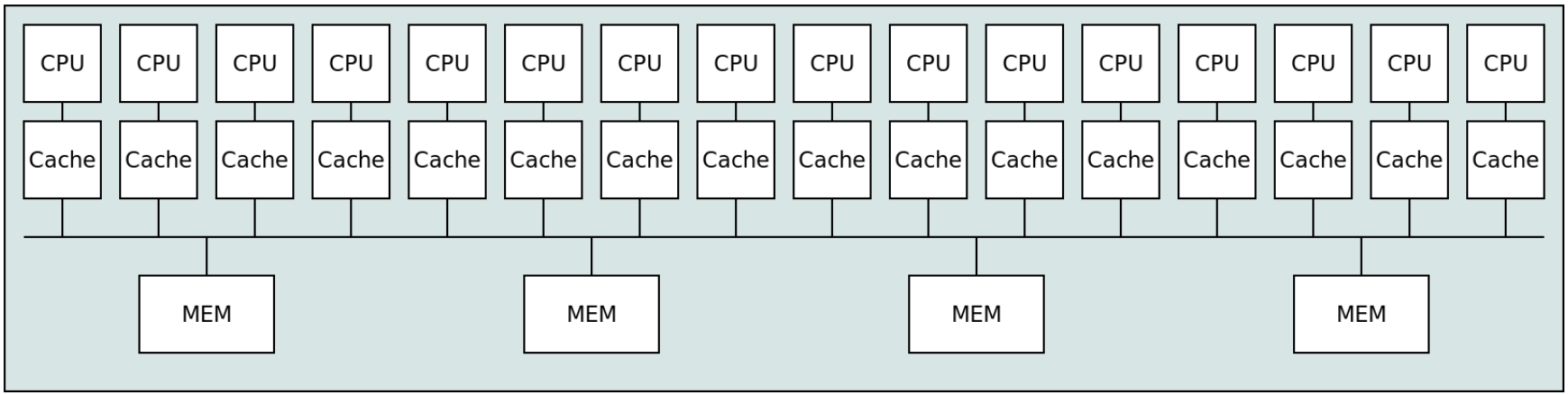



## NUMA: Non-Uniform Memory Access




COM: Einheit zum Weiterleiten von Speicherzugriffen

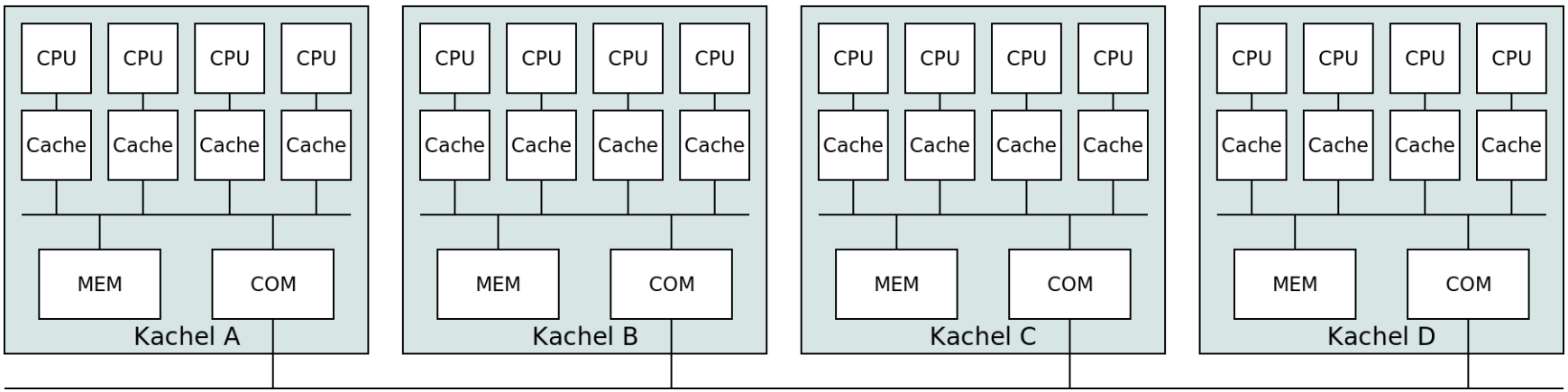
# Vielkernprozessoren - Uniform Memory Access




\+ Cache-Kohärenz möglich („Snooping”, „MESI”-Protokoll)

\- (zu) hohe Bus-Auslastung

# Vielkernprozessoren - Non-Uniform Memory Access

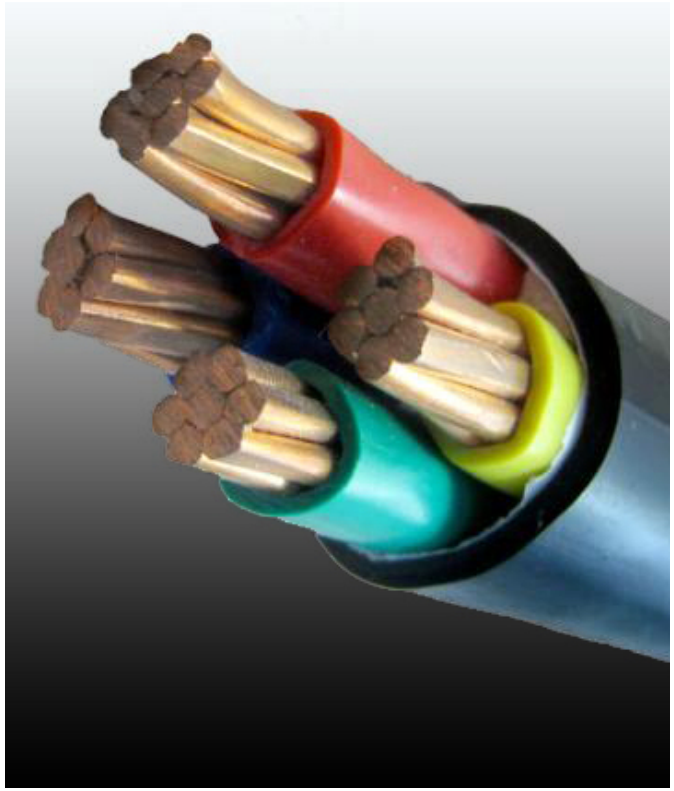



\+ geringe Bus-Auslastung

\- Cache-Kohärenz unmöglich

# Vielkernprozessoren

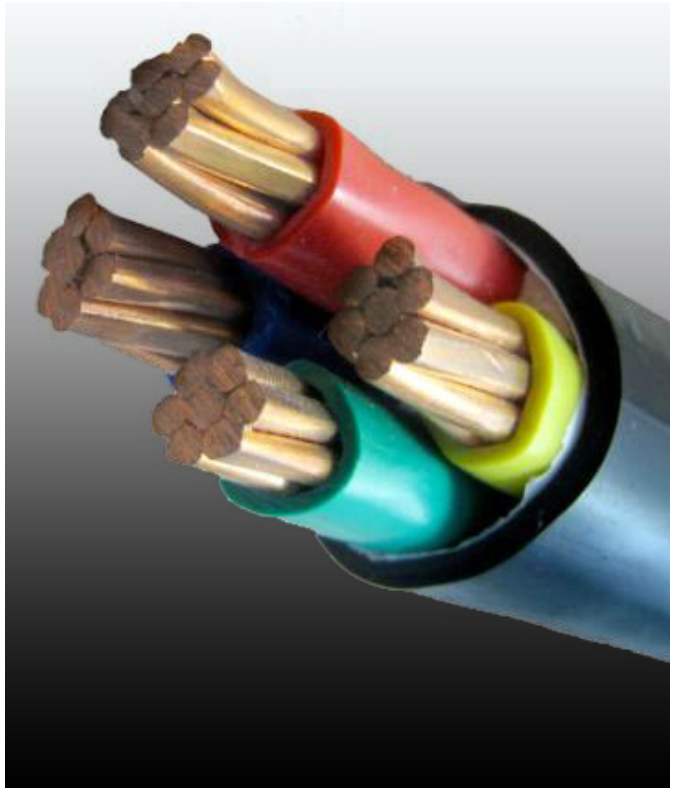
- Symbol[1] eines hochmodernen gekachelten 28-Kern-Prozessors: ☺
  - 7 Kerne pro Kachel
    - cache-kohärent
    - homogen auf Kernebene
  - 4 Kacheln pro CPU
    - cache-inkohärent
    - heterogen auf Kachelebene



[1] Suchergebnis, als `wosch` nach dem Lesen des Artikels von Herb Sutter [9] im Internet erstmalig nach dem Begriff „Multi-Core“ stöberte.

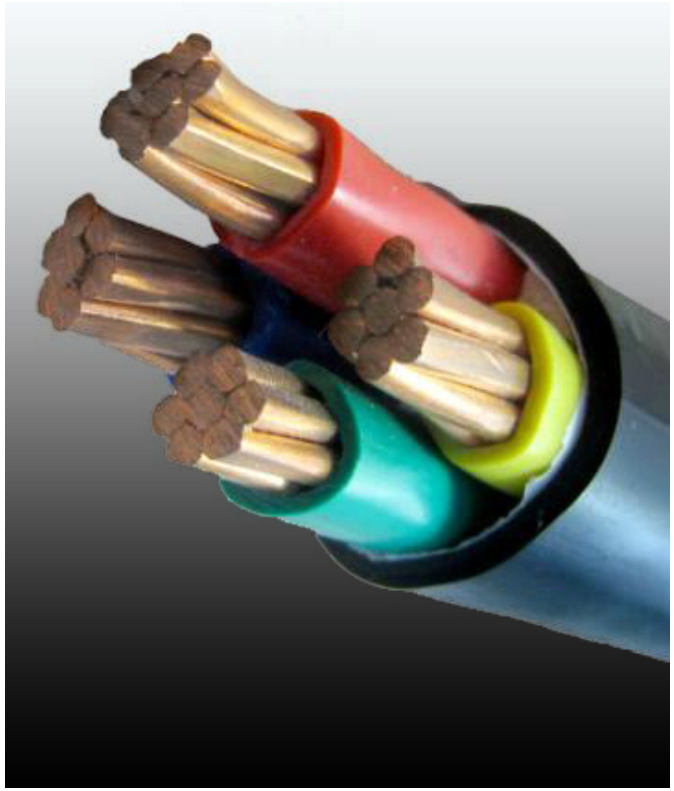
# Vielkernprozessoren

- „viel“ geht weit über „mehr“ hinaus
  - mehrere zehn Kerne als untere Grenze
  - Hunderte oder Tausende sind keine Utopie



[1] Suchergebnis, als `wosch` nach dem Lesen des Artikels von Herb Sutter [9] im Internet erstmalig nach dem Begriff „Multi-Core“ stöberte.

# Vielkernprozessoren



- **globaler gemeinsamer Hauptspeicher**
  - verteilt über alle Kacheln
  - woraus aber nicht zwingend global einheitliche **Speicherkonsistenz** folgt

[1] Suchergebnis, als `wosch` nach dem Lesen des Artikels von Herb Sutter [9] im Internet erstmalig nach dem Begriff „Multi-Core“ stöberte.

# Gemeinsamer Arbeitsspeicher

- die Skalierbarkeit des *Shared-Memory*-Paradigmas steht und fällt mit der Art der Parallelität und der Programmierung [7]
  - grobkörnige Mitbenutzung und Botschaftenaustausch (*message passing*) zeigen etwa den gleichen Grad an Skalierung
  - feinkörnige Ansätze sind dagegen sehr stark durch die Beschränkungen des Gesetzes von Amdahl [1] begrenzt

# Gemeinsamer Arbeitsspeicher

- ein Problemfall, nicht zuletzt wegen des **TLB** (*translation lookaside buffer*)[2]
  - einer pro Rechenkern
  - die CPU fährt aber kein Kohärenzprotokoll

[2] Hieronymus im Gehäus, Albrecht Dürer, 1514: Hieronymus gilt im übertragenen Sinne als Schutzpatron der Übersetzer.

# Gemeinsamer Arbeitsspeicher

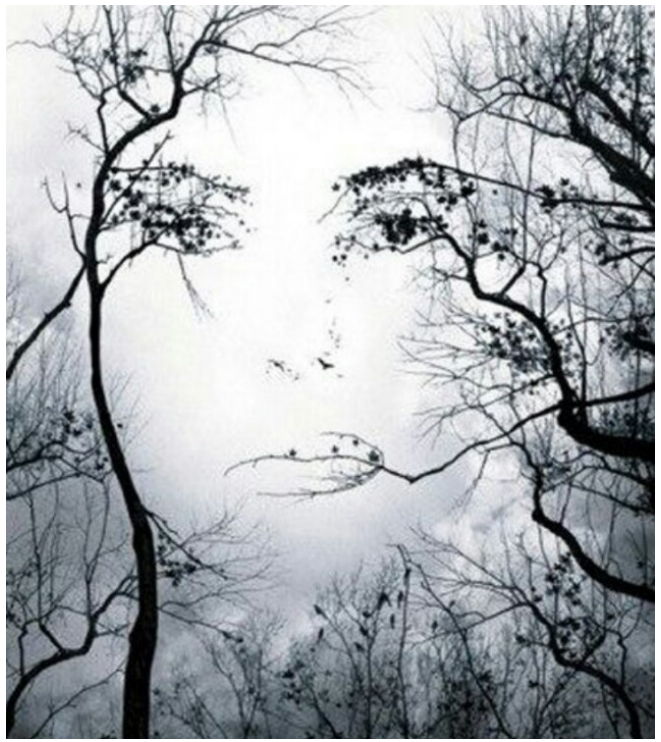
- ein Problemfall, nicht zuletzt wegen des **TLB** (*translation lookaside buffer*)[2]

- erzeugt **Interferenz** innerhalb desselben gemeinsam genutzten Adressraums
  - gleichzeitige Prozesse
  - verschiedenen Rechenkernen zugeordnet

[2] Hieronymus im Gehäus, Albrecht Dürer, 1514: Hieronymus gilt im übertragenen Sinne als Schutzpatron der Übersetzer.

# Interferenz

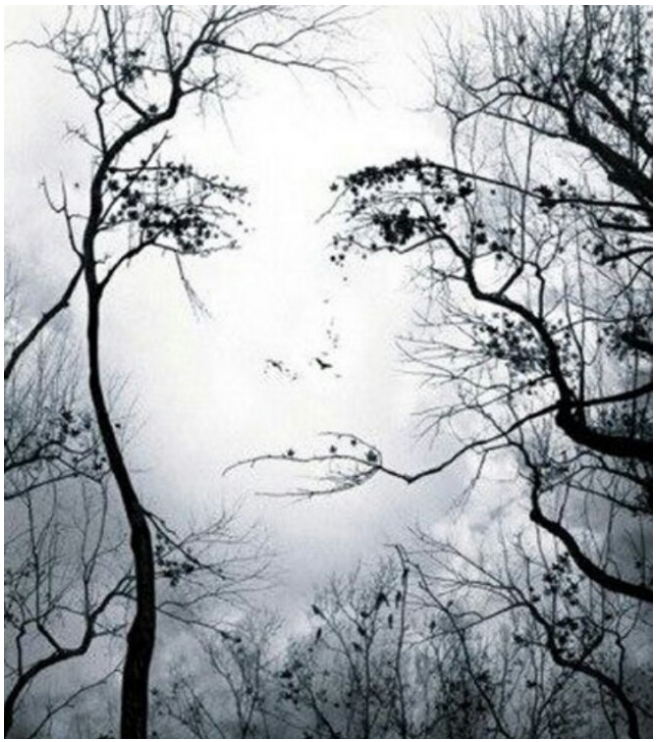
- **Überlagerung** von Aktionen[3] beim Zusammentreffen zweier oder mehrerer gleichzeitiger Prozesse
  - als Folge des Zugriffs auf eine gemeinsam genutzte Ressource
  - auch ausgelöst durch widersprüchliche Planungs-/Auswahlentscheidungen



[3] Hier bereits die parallele Ausführung eines Maschinenbefehls durch die CPU.

# Interferenz

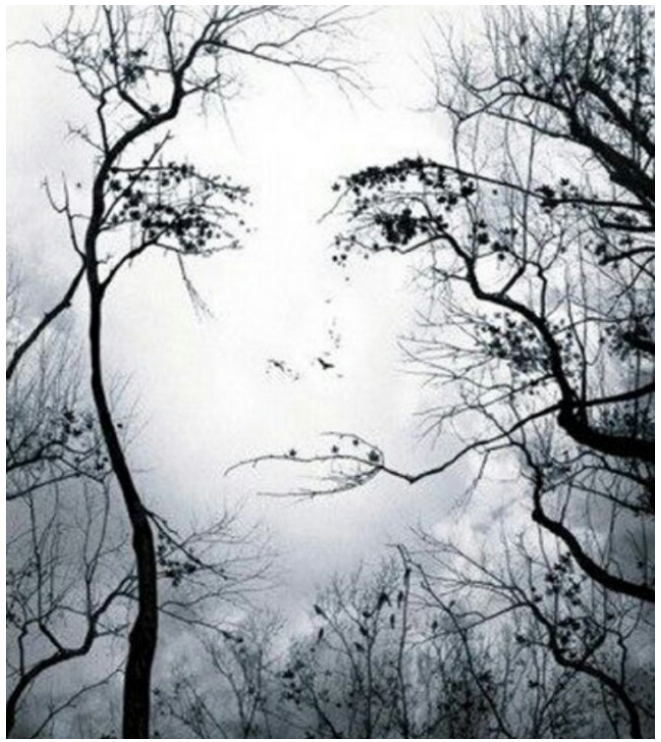
- **Rauschen** im Hintergrund der Ausführung von Programmen (*background noise*)
  - nichtfunktionales Merkmal
  - das jede Ausführungsumgebung besitzt



↪ **Betriebssystemrauschen** (*operating-system noise*)

# Interferenz

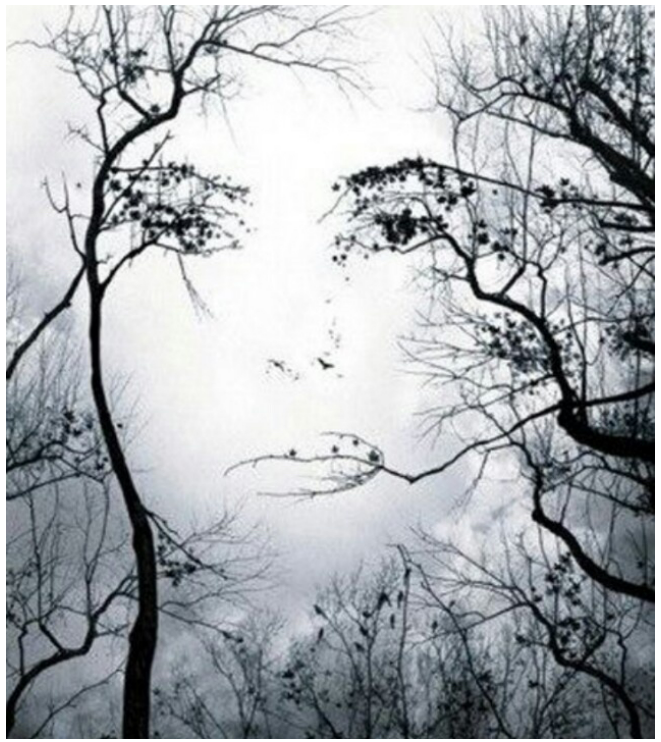
- **Rauschen** im Hintergrund der Ausführung von Programmen (*background noise*)
- damit **Behinderung** eines Prozesses durch gleichzeitige äußere Einwirkungen
  - eines anderen Prozesses
  - auf derselben oder einer anderen CPU



↪ **Betriebssystemrauschen** (*operating-system noise*)

# Interferenz

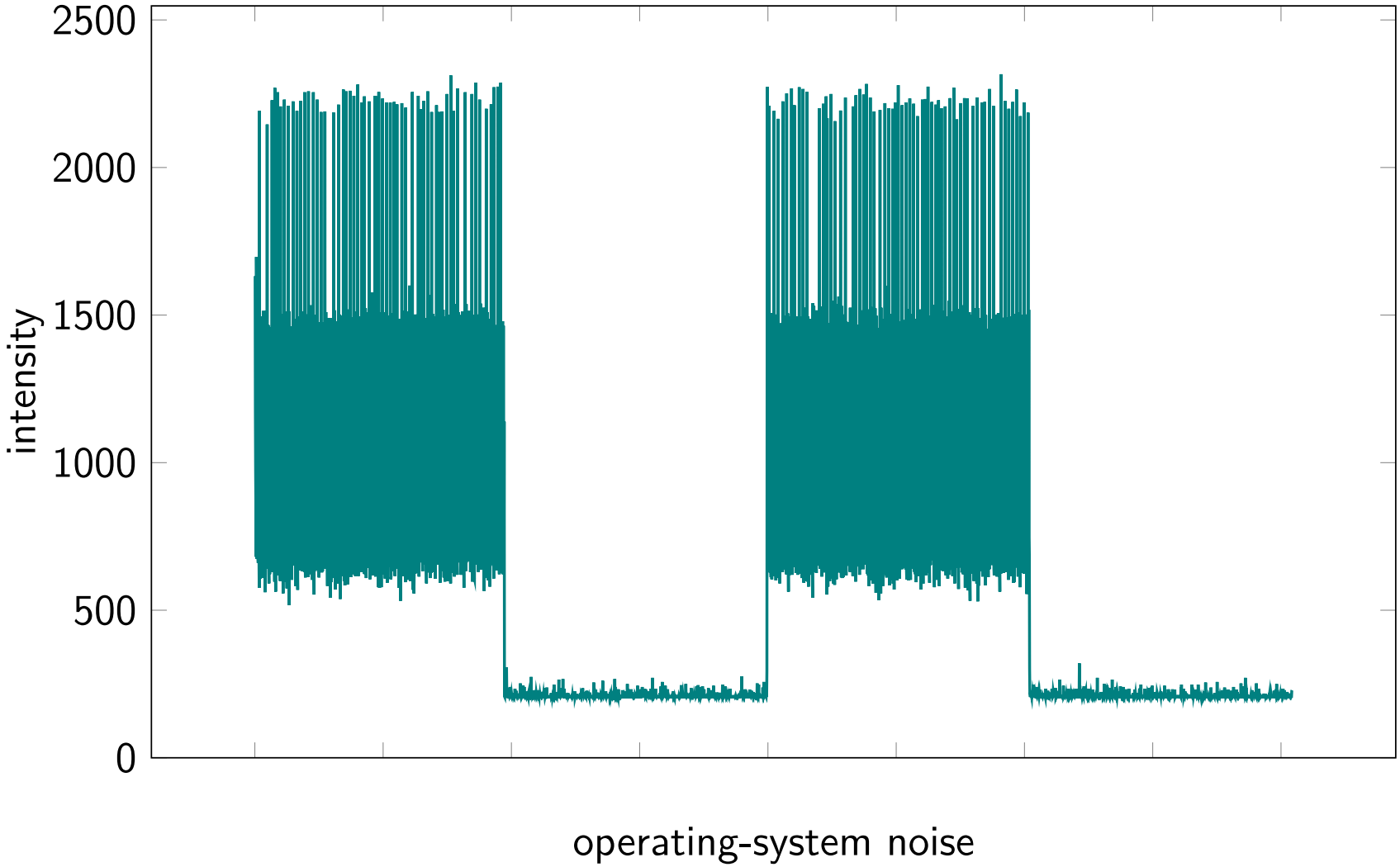
- **Rauschen** im Hintergrund der Ausführung von Programmen (*background noise*)
- damit **Behinderung** eines Prozesses durch gleichzeitige äußere Einwirkungen
- eigentlich Normalität, sofern die Störung in Grenzen bleibt



↪ **Betriebssystemrauschen** (*operating-system noise*)

# „TLB-Lärm“ beim Teilen

Adressraumisolation „an“ vs. „aus“ [5]




# Gliederung



# PGAS

*partitioned global address space*

# PGAS

*partitioned global address space*

- ein **Programmiermodell** mit folgender Prämisse [3]:
  - ein global gemeinsam genutzter Adressraum verbessert die Produktivität
  - erfordert aber eine Unterscheidung zwischen „lokal“ und „entfernt“
    - um Leistungsoptimierungen zu ermöglichen
    - um Skalierbarkeit auf groß angelegten parallelen Architekturen zu erreichen

- ein **Programmiermodell**

- aus **Betriebssystemsicht** handelt es sich dabei um:
  - mehrere Hauptspeicherpartitionen, die sich kooperierende Prozesse teilen
    - in logischer Hinsicht ein gemeinsamer verteilter Hauptspeicher
  - ein einzelner Adressraum, in dem die kooperierenden Prozesse residieren
    - genauer: ein und derselbe logische/virtuelle Adressraum für diese Prozesse

# PGAS

*partitioned global address space*

- ein **Programmiermodell**

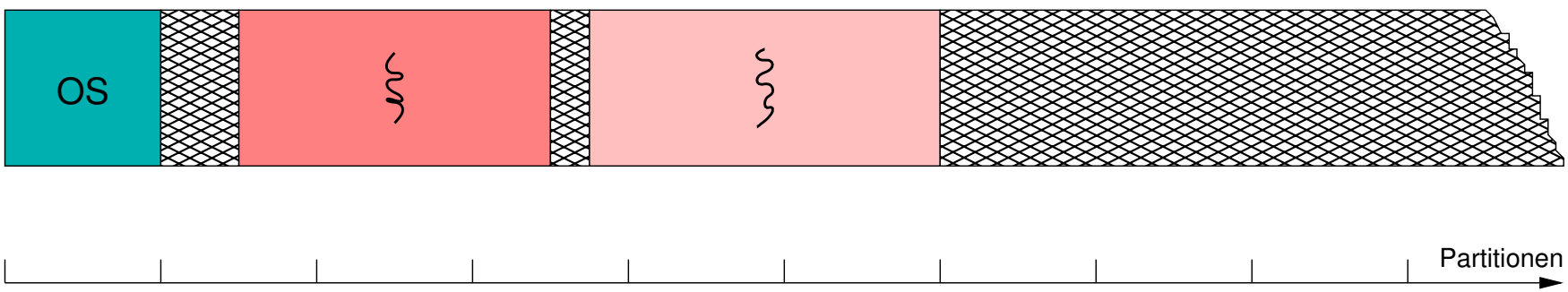
- aus **Betriebssystemsicht** handelt es sich dabei um:
  - mehrere Hauptspeicherpartitionen, die sich kooperierende Prozesse teilen
    - in logischer Hinsicht ein gemeinsamer verteilter Hauptspeicher
  - ein einzelner Adressraum, in dem die kooperierenden Prozesse residieren
    - genauer: ein und derselbe logische/virtuelle Adressraum für diese Prozesse

↪ Anforderung/Freigabe von Speicher bedingt Adressraumänderungen

- ein **Programmiermodell**

- aus **Betriebssystemsicht** handelt es sich dabei um:
  - mehrere Hauptspeicherpartitionen, die sich kooperierende Prozesse teilen
    - in logischer Hinsicht ein gemeinsamer verteilter Hauptspeicher
  - ein einzelner Adressraum, in dem die kooperierenden Prozesse residieren
    - genauer: ein und derselbe logische/virtuelle Adressraum für diese Prozesse

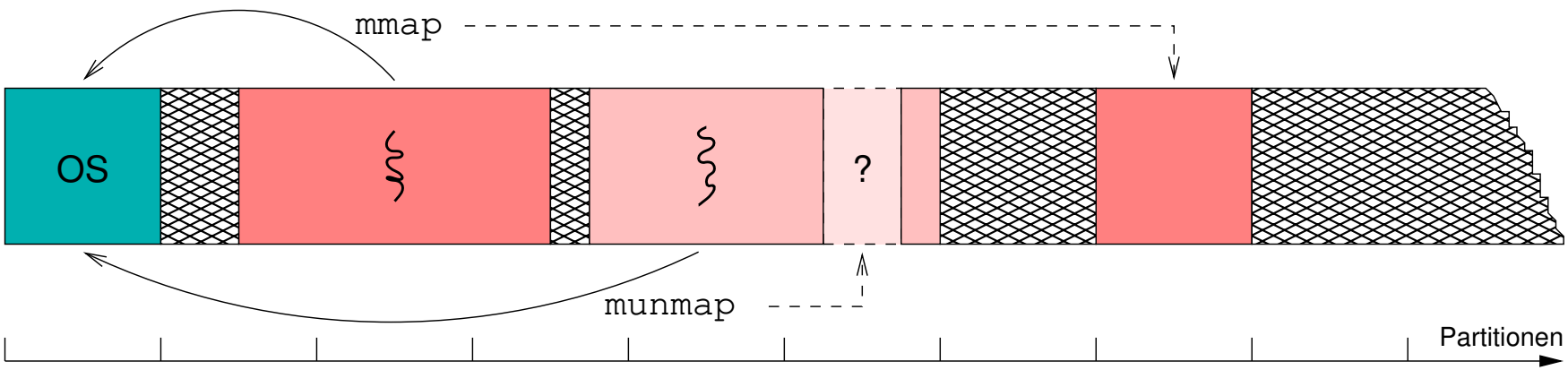
↪ Anforderung/Freigabe von Speicher bedingt Adressraumänderungen




- ein **Programmiermodell**

- aus **Betriebssystemsicht** handelt es sich dabei um:
  - mehrere Hauptspeicherpartitionen, die sich kooperierende Prozesse teilen
    - in logischer Hinsicht ein gemeinsamer verteilter Hauptspeicher
  - ein einzelner Adressraum, in dem die kooperierenden Prozesse residieren
    - genauer: ein und derselbe logische/virtuelle Adressraum für diese Prozesse

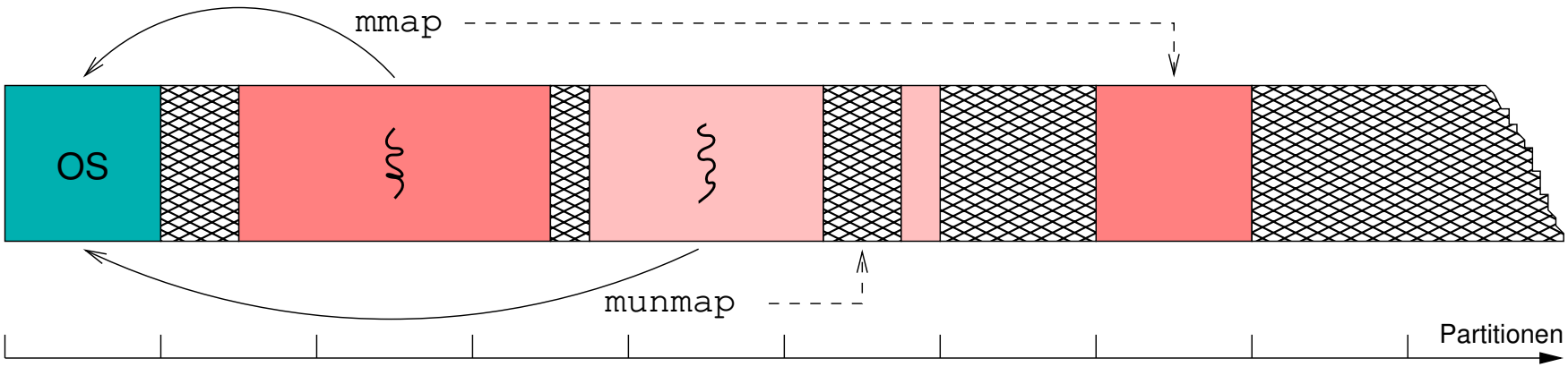
↪ Anforderung/Freigabe von Speicher bedingt Adressraumänderungen




- ein **Programmiermodell**

- aus **Betriebssystemsicht** handelt es sich dabei um:
  - mehrere Hauptspeicherpartitionen, die sich kooperierende Prozesse teilen
    - in logischer Hinsicht ein gemeinsamer verteilter Hauptspeicher
  - ein einzelner Adressraum, in dem die kooperierenden Prozesse residieren
    - genauer: ein und derselbe logische/virtuelle Adressraum für diese Prozesse

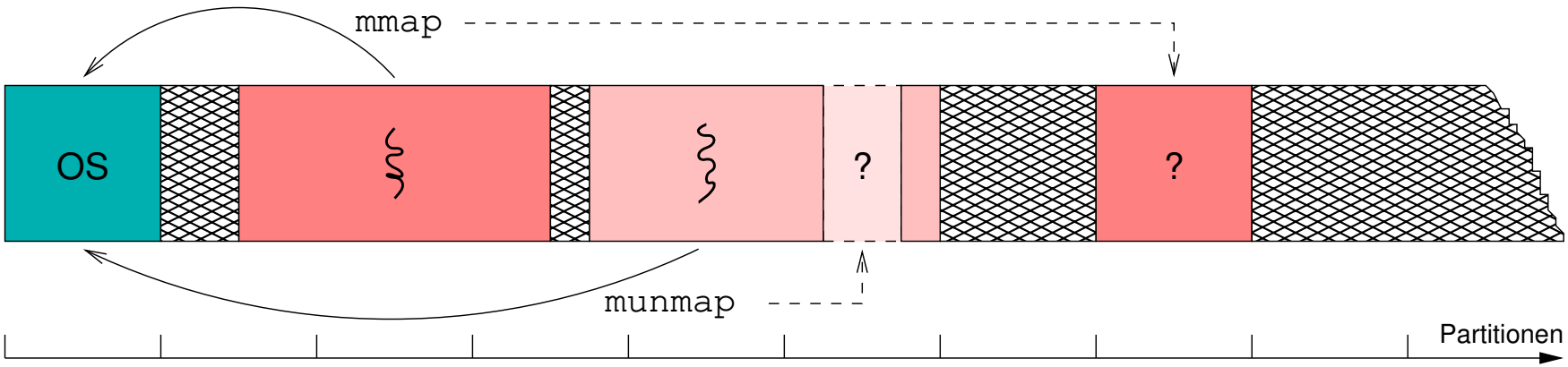
↪ Anforderung/Freigabe von Speicher bedingt Adressraumänderungen




- ein **Programmiermodell**

- aus **Betriebssystemsicht** handelt es sich dabei um:
  - mehrere Hauptspeicherpartitionen, die sich kooperierende Prozesse teilen
    - in logischer Hinsicht ein gemeinsamer verteilter Hauptspeicher
  - ein einzelner Adressraum, in dem die kooperierenden Prozesse residieren
    - genauer: ein und derselbe logische/virtuelle Adressraum für diese Prozesse

↪ Anforderung/Freigabe von Speicher bedingt Adressraumänderungen




# Adressraumkonsistenz

- Änderungen von Attributen global gemeinsam genutzter Adressräume müssen in konsistenter Art und Weise geschehen

# Adressraumkonsistenz

- Änderungen von Attributen global gemeinsam genutzter Adressräume müssen in konsistenter Art und Weise geschehen
  - Attribute, erfasst über die **Seitendeskriptoren** der Adressräume
    - Anzahl – Größe des gesamten Adressraums
    - Anzahl – Maß für die Gesamtzahl der benötigten Seitentabellen
    - Zustand – Ort einer Seite und die mit ihr verbundenen Rechte

# Adressraumkonsistenz

- Änderungen von Attributen global gemeinsam genutzter Adressräume müssen in konsistenter Art und Weise geschehen
  - Attribute, erfasst über die **Seitendeskriptoren** der Adressräume
  - wobei ein Seitendeskriptor eine logische und physische Erscheinung hat

    logisch – der Repräsentant im globalen virtuellen Adressraum
    – identifiziert/lokalisiert die Seite
    ↪ scheinbar nur einmal vorhanden

# Adressraumkonsistenz

- Änderungen von Attributen global gemeinsam genutzter Adressräume müssen in konsistenter Art und Weise geschehen
  - Attribute, erfasst über die **Seitendeskriptoren** der Adressräume
  - wobei ein Seitendeskriptor eine logische und physische Erscheinung hat

    physisch – der Repräsentant im jeweils lokalen Speicher eines Prozessors
    – identifiziert/lokalisiert den Seitenrahmen
    ↪ wirklich mehrfach vorhanden, mit verschiedenen Werten

# Adressraumkonsistenz

- Änderungen von Attributen global gemeinsam genutzter Adressräume müssen in konsistenter Art und Weise geschehen
  - Attribute, erfasst über die **Seitendeskriptoren** der Adressräume

- wobei ein Seitendeskriptor eine logische und physische Erscheinung hat

  logisch
  - der Repräsentant im globalen virtuellen Adressraum
  - identifiziert/lokalisiert die Seite $\rightsquigarrow$ **Referenz**

  $\hookrightarrow$ scheinbar nur einmal vorhanden

  physisch
  - der Repräsentant im jeweils lokalen Speicher eines Prozessors
  - identifiziert/lokalisiert den Seitenrahmen $\rightsquigarrow$ **Replikat**

  $\hookrightarrow$ wirklich mehrfach vorhanden, mit verschiedenen Werten

# Adressraumkonsistenz

- Änderungen von Attributen global gemeinsam genutzter Adressräume müssen in konsistenter Art und Weise geschehen
  - Attribute, erfasst über die **Seitendeskriptoren** der Adressräume
  - wobei ein Seitendeskriptor eine logische und physische Erscheinung hat
- jede **lokale Domäne** führt Buch über die in ihrem Gültigkeitsbereich vorliegenden Verwaltungsstrukturen
  - Bezugspunkt dabei ist die CPU, genauer: der **einzelne Rechenkern**
    - deren MMU zu programmieren und TLB zu verwalten ist

# Verwaltungsstrukturen eines Prozessadressraums

- Prozessisolation „benutzt“ bestimmte **funktionale Merkmale**, die typischerweise einem Betriebssystem zugerechnet werden

# Verwaltungsstrukturen eines Prozessadressraums

- Prozessisolation „benutzt“ bestimmte **funktionale Merkmale**, die typischerweise einem Betriebssystem zugerechnet werden
  (a) eine für gewöhnlich mehrstufige Seitentabelle pro Prozessexemplar
    - erfasst die Seitendeskriptoren des zugewiesenen Arbeitsspeichers

# Verwaltungsstrukturen eines Prozessadressraums

- Prozessisolation „benutzt“ bestimmte **funktionale Merkmale**, die typischerweise einem Betriebssystem zugerechnet werden

(b) Liste der Gebrauchsstücke an Hauptspeicher pro Prozessexemplar
  - erfasst zugewiesene statische/dynamische (seitennummerierte) Segmente

# Verwaltungsstrukturen eines Prozessadressraums

- Prozessisolation „benutzt“ bestimmte **funktionale Merkmale**, die typischerweise einem Betriebssystem zugerechnet werden

(c) der Übersetzungspuffer (TLB) pro CPU/Rechenkern
  - erfasst kürzlich erfolgte Übersetzungen von logischen/virtuellen Adressen

# Verwaltungsstrukturen eines Prozessadressraums

- Prozessisolation „benutzt“ bestimmte **funktionale Merkmale**, die typischerweise einem Betriebssystem zugerechnet werden

(d) kernglobale Speicherverwaltung pro Kachel (*tile*)
  - erfasst verfügbare statische/dynamische (seitennummerierte) Segmente

# Verwaltungsstrukturen eines Prozessadressraums

- Prozessisolation „benutzt“ bestimmte **funktionale Merkmale**, die typischerweise einem Betriebssystem zugerechnet werden

(e) eine globale Seitentabelle pro System
  - erfasst alle Anwendungsprogramme plus Betriebssystem

# Verwaltungsstrukturen eines Prozessadressraums

- Prozessisolation „benutzt“ bestimmte **funktionale Merkmale**, die typischerweise einem Betriebssystem zugerechnet werden

- bei Adressraumänderungen ist dabei ein **verteilter Zustand** logisch zusammenhängender TLB/MMU Einheiten konsistent zu halten
  - relevant sind Aktionen, die den global gemeinsamen Adressraum betreffen

# Verwaltungsstrukturen eines Prozessadressraums

- Prozessisolation „benutzt“ bestimmte **funktionale Merkmale**, die typischerweise einem Betriebssystem zugerechnet werden

(c) der Übersetzungspuffer (TLB) pro CPU/Rechenkern
  - erfasst kürzlich erfolgte Übersetzungen von logischen/virtuellen Adressen

- bei Adressraumänderungen ist dabei ein **verteilter Zustand** logisch zusammenhängender TLB/MMU Einheiten konsistent zu halten
  - relevant sind Aktionen, die den global gemeinsamen Adressraum betreffen
  - allerdings sind TLB/MMU nur lokal „von ihrem Kern aus“ zugängig

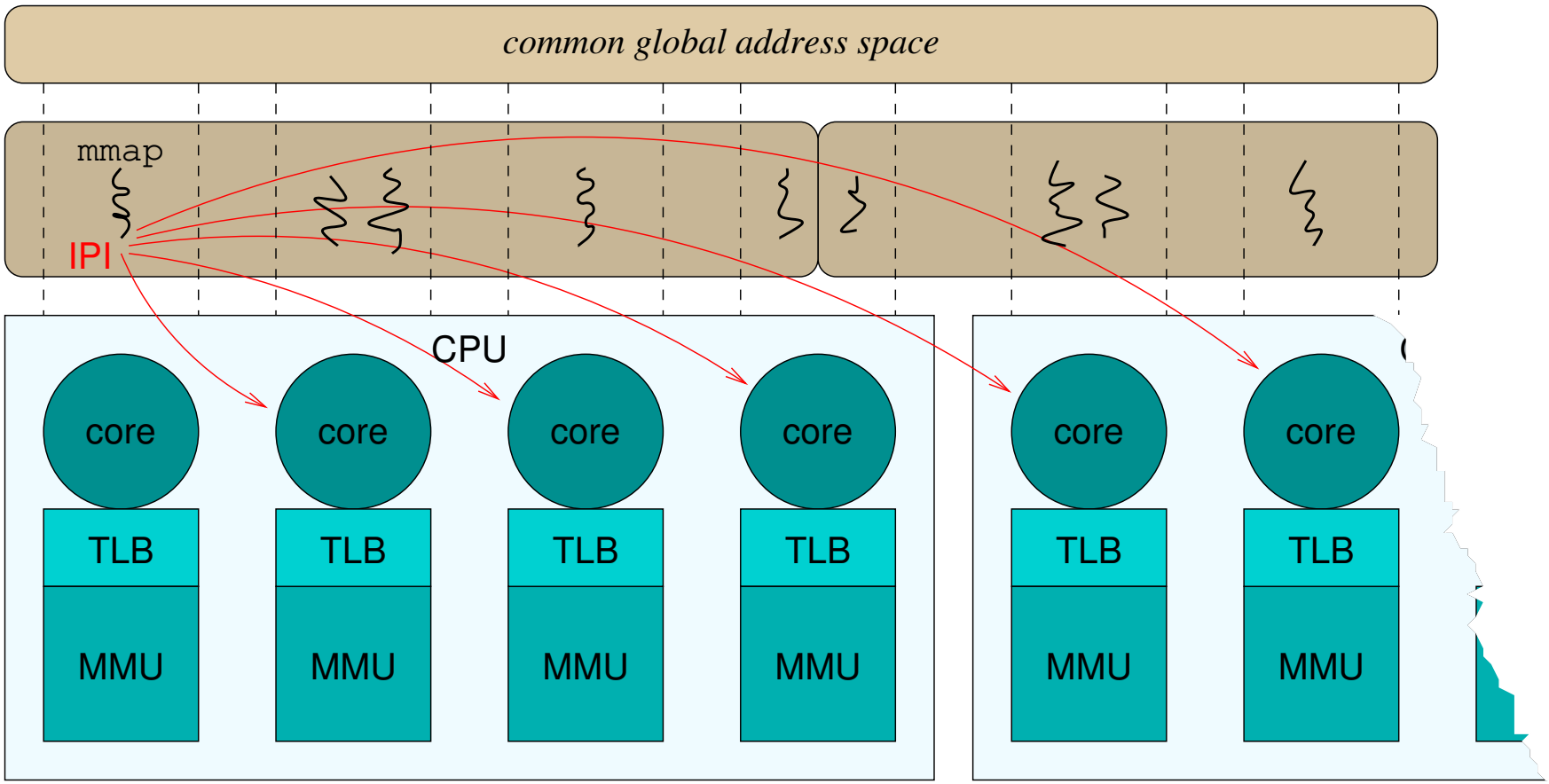
    ↪ per **IPI** (*interprocessor interrupt*) sind Kerne zur „Wartung“ anzustoßen

# Global ≡ Kernübergreifend



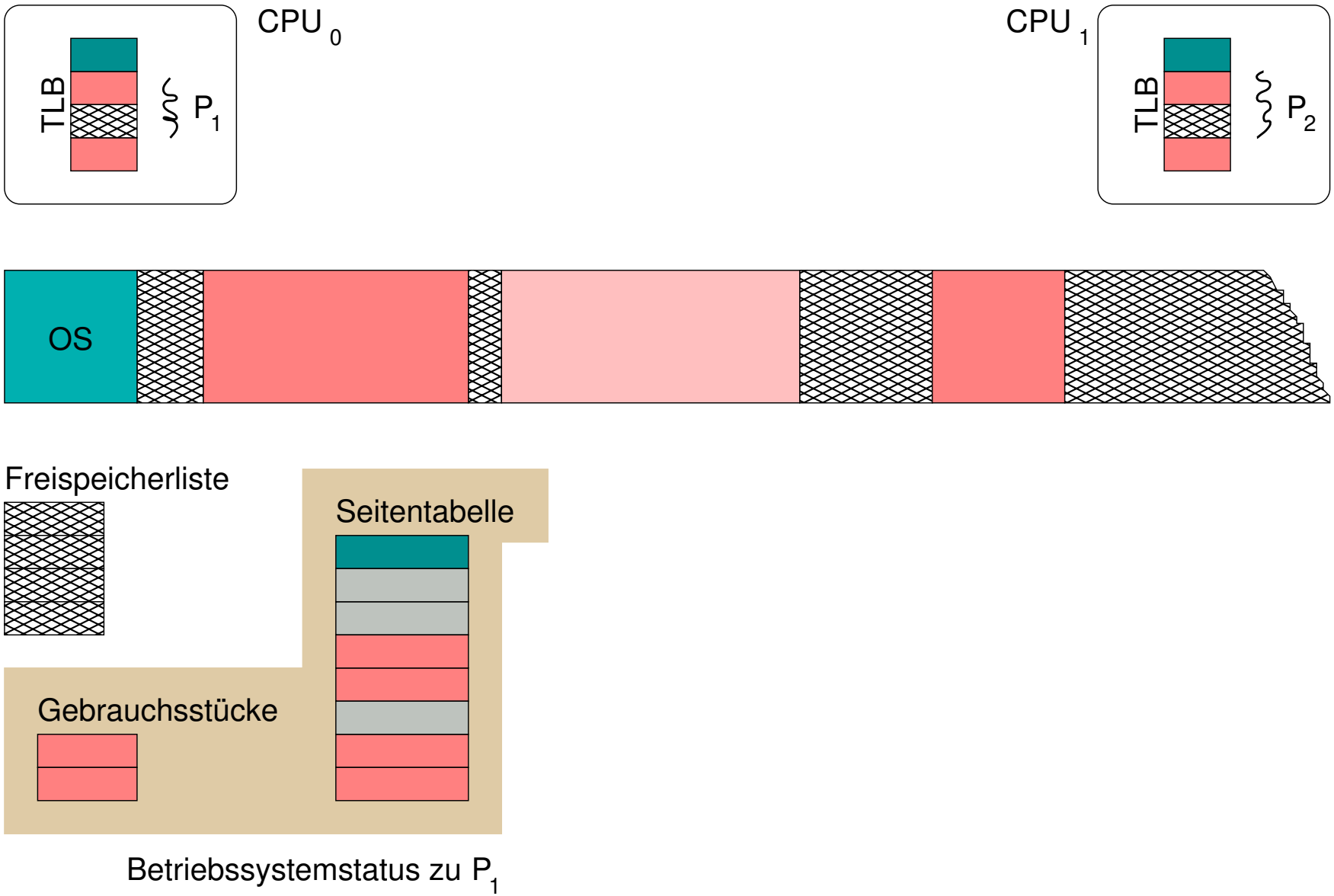
- auf Basis kernlokaler Ressourcen zur globalen Adressraumverwaltung
  - insbesondere **MMU** und **TLB**

↪ die nur lokal zugängig sind, aber von entfernt den Zustand ändern müssen

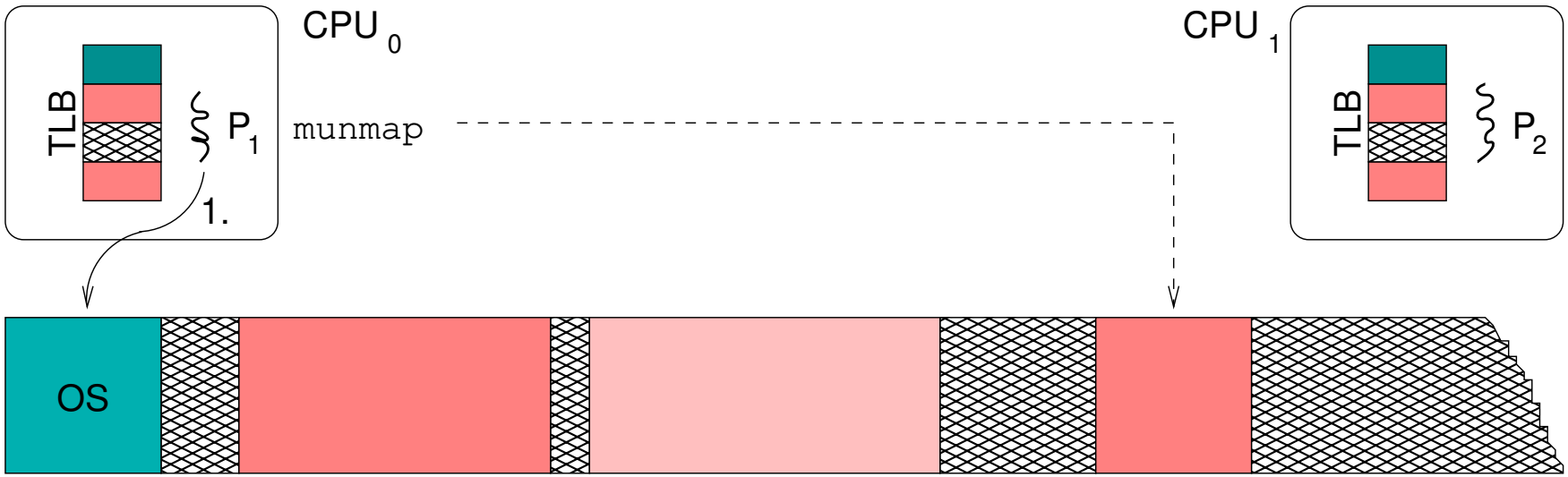


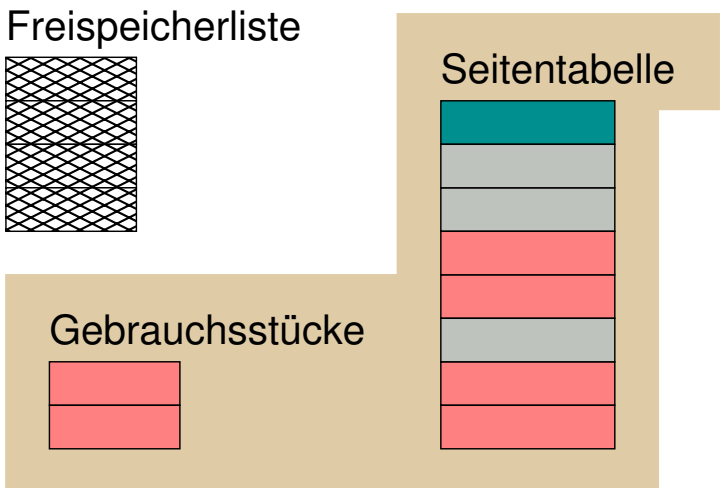

# Isolationsbetrieb

# Isolationsbetrieb



# Isolationsbetrieb

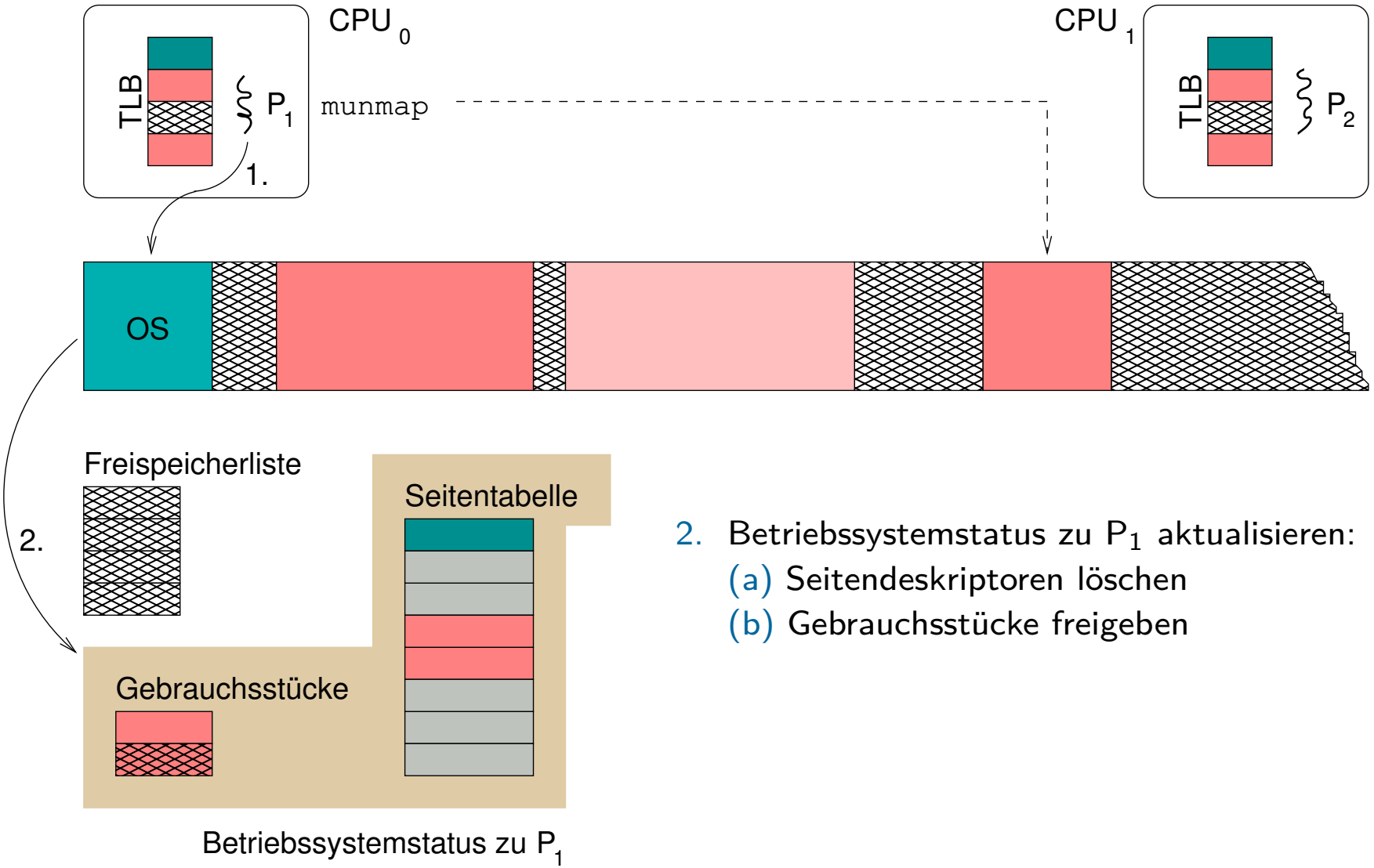



Betriebssystemstatus zu $P_1$

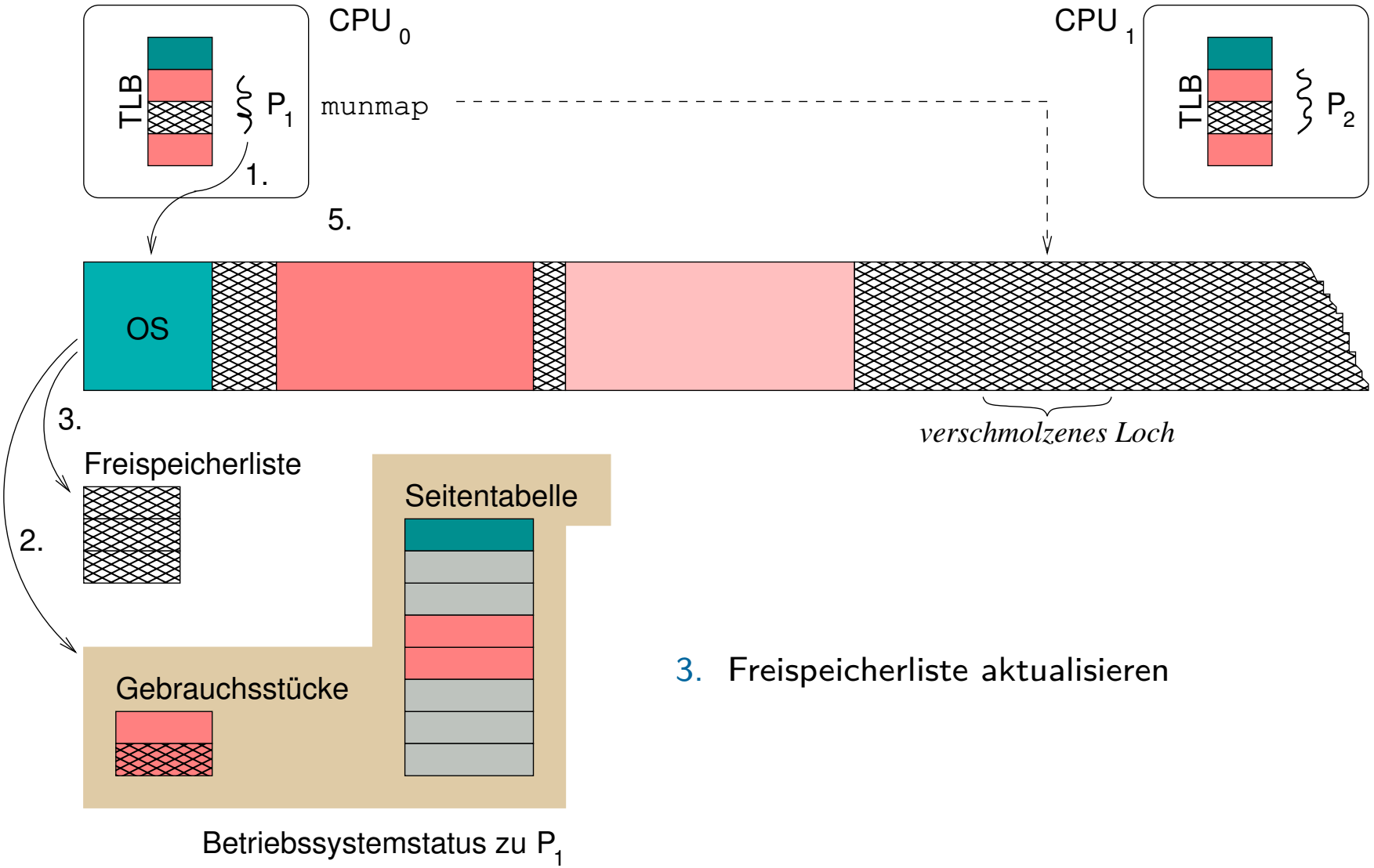
1. $P_1$ ruft `munmap()`

# Isolationsbetrieb



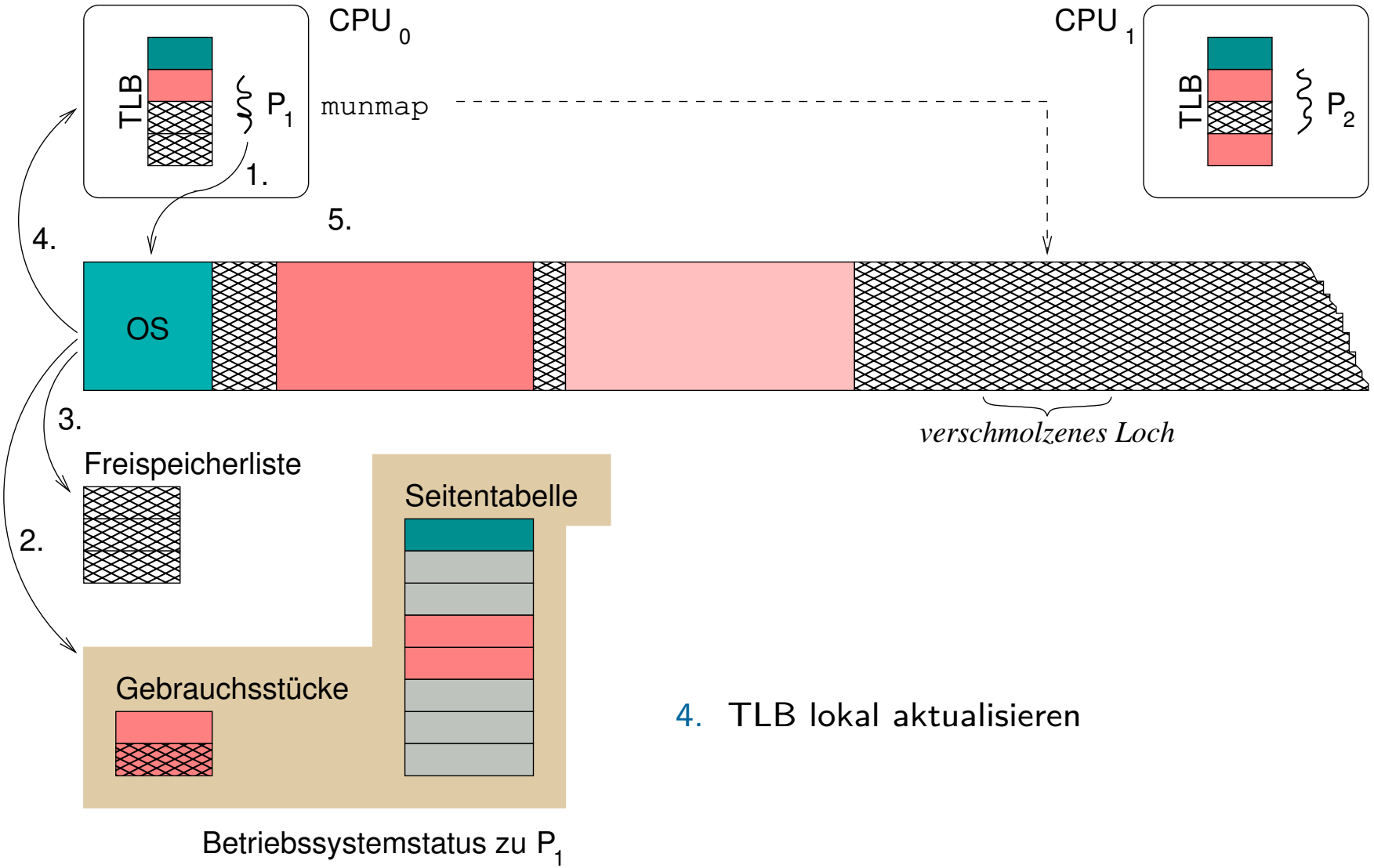

2. Betriebssystemstatus zu $P_1$ aktualisieren:
   (a) Seitendeskriptoren löschen
   (b) Gebrauchsstücke freigeben

# Isolationsbetrieb



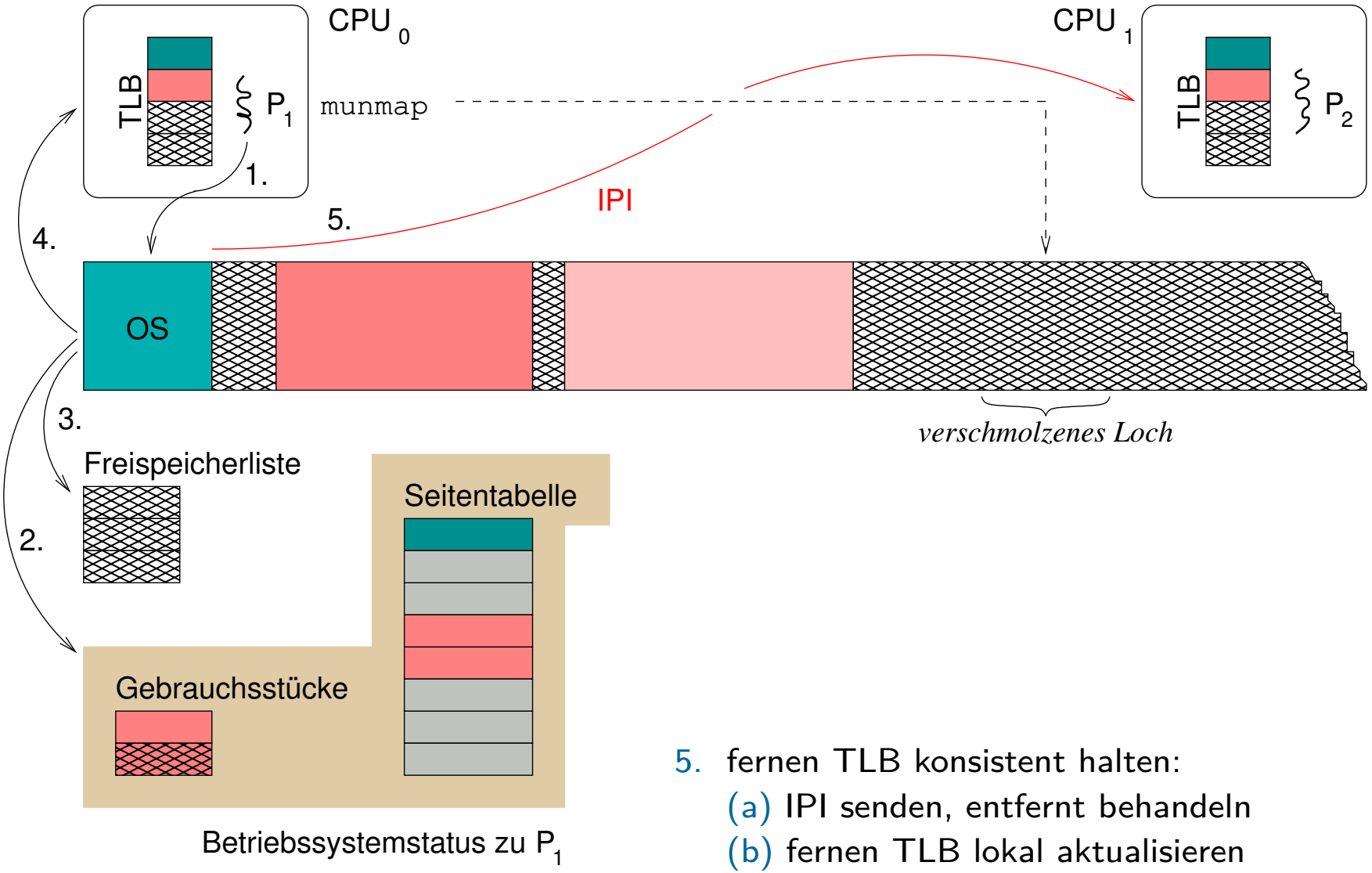

3. Freispeicherliste aktualisieren

# Isolationsbetrieb




4. TLB lokal aktualisieren

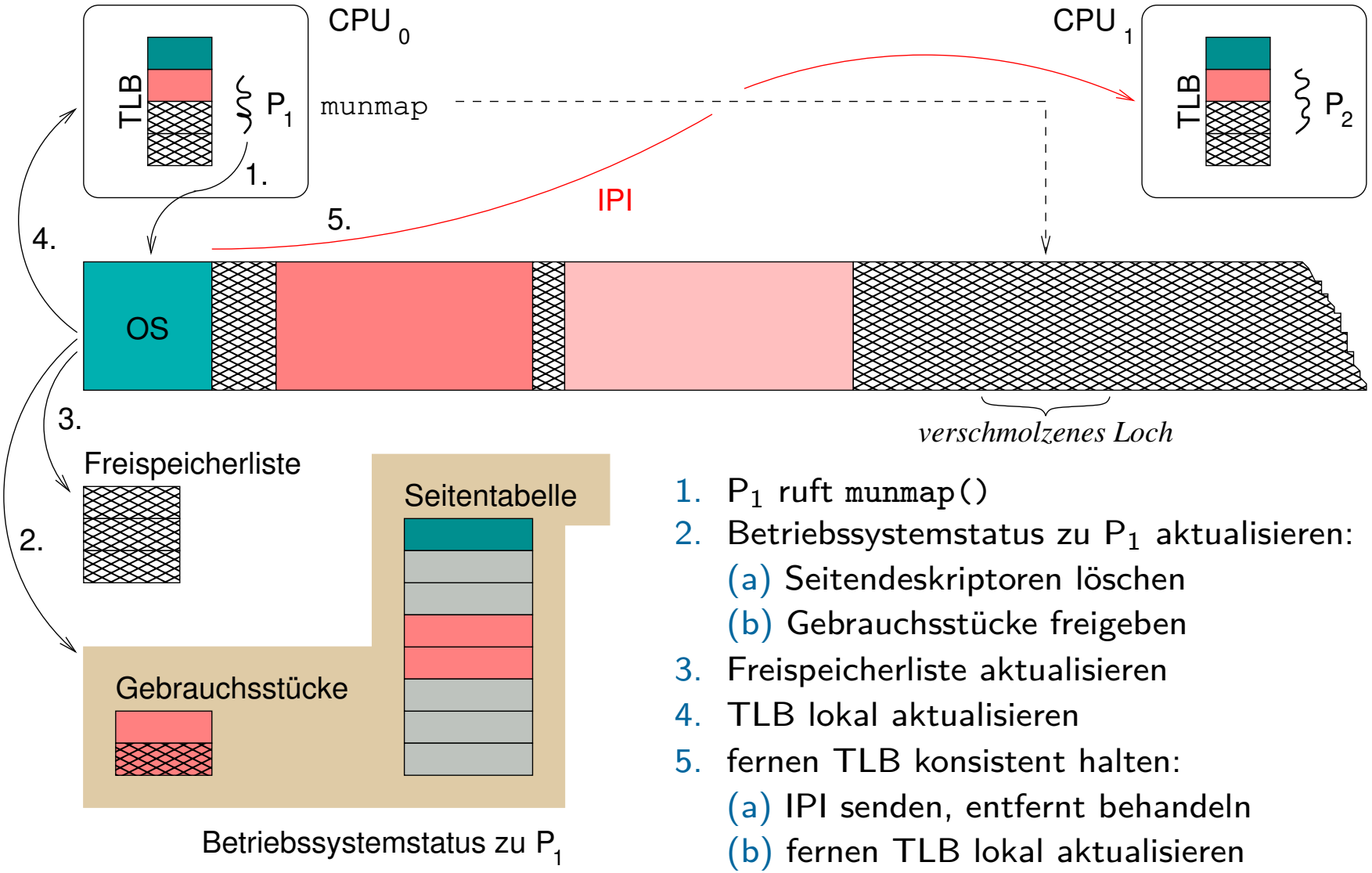
# Isolationsbetrieb




5. fernen TLB konsistent halten:
   (a) IPI senden, entfernt behandeln
   (b) fernen TLB lokal aktualisieren

# Isolationsbetrieb






1. $P_1$ ruft `munmap()`
2. Betriebssystemstatus zu $P_1$ aktualisieren:
   (a) Seitendeskriptoren löschen
   (b) Gebrauchsstücke freigeben
3. Freispeicherliste aktualisieren
4. TLB lokal aktualisieren
5. fernen TLB konsistent halten:
   (a) IPI senden, entfernt behandeln
   (b) fernen TLB lokal aktualisieren

. . . schließlich $P_1$ wieder aufnehmen

# Isolationsbetrieb „*considered harmful*“ . . .

- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**

# Isolationsbetrieb „*considered harmful*“ . . .

- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**
  - der eine Adressraumänderung auslösende Faden erzeugt eine **Laufgefahr** (*race condition*) durch zeitweilig inkonsistente TLB-Inhalte

# Isolationsbetrieb „*considered harmful*“ . . .

- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**
  - der eine Adressraumänderung auslösende Faden erzeugt eine **Laufgefahr** (*race condition*) durch zeitweilig inkonsistente TLB-Inhalte
    - ist der betreffende Systemaufruf (z.B. `munmap`) der Abstraktionsebene des Fadens zuzurechnen, darf ein kritischer Abschnitt angenommen werden
      ↪ Bestandteil des nichtsequentiellen Programms, das den Fadens definiert

# Isolationsbetrieb „*considered harmful*“ . . .

- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**
  - der eine Adressraumänderung auslösende Faden erzeugt eine **Laufgefahr** (*race condition*) durch zeitweilig inkonsistente TLB-Inhalte

    – „versteckt“ sich der Systemaufruf jedoch in einer tieferliegenden Ebene, etwa innerhalb einer Bibliotheksfunktion, fehlt Wissen zur Prozesssynchronisation
    $\rightsquigarrow$ Annahme des schlimmsten Falls

# Isolationsbetrieb „*considered harmful*“ . . .

- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**
  - der eine Adressraumänderung auslösende Faden erzeugt eine **Laufgefahr** (*race condition*) durch zeitweilig inkonsistente TLB-Inhalte
  - für einen über die Rechenkerne verteilter mehr-/vielfädiger Prozess ist der IPI ein **Gruppenruf** (*multicast*) oder **Rundruf** (*broadcast*)

# Isolationsbetrieb „*considered harmful*“ . . .

- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**
  - der eine Adressraumänderung auslösende Faden erzeugt eine **Laufgefahr** (*race condition*) durch zeitweilig inkonsistente TLB-Inhalte

- für einen über die Rechenkerne verteilter mehr-/vielfädiger Prozess ist der IPI ein **Gruppenruf** (*multicast*) oder **Rundruf** (*broadcast*)
  - dieser sollte mit der Anzahl der dem gleichzeitigen Prozess jeweils dynamisch zugewiesenen Rechenkerne skalieren

# Isolationsbetrieb „*considered harmful*“ . . .

- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**
  - der eine Adressraumänderung auslösende Faden erzeugt eine **Laufgefahr** (*race condition*) durch zeitweilig inkonsistente TLB-Inhalte

- für einen über die Rechenkerne verteilter mehr-/vielfädiger Prozess ist der IPI ein **Gruppenruf** (*multicast*) oder **Rundruf** (*broadcast*)
  - dieser sollte mit der Anzahl der dem gleichzeitigen Prozess jeweils dynamisch zugewiesenen Rechenkerne skalieren
  - zudem muss der rufende Betriebssystemkern Gewissheit darüber haben, dass sämtliche gerufenen „*IPI handler*“ ihre Aufgaben haben erledigen können

# Isolationsbetrieb „*considered harmful*“ . . .

- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**
  - der eine Adressraumänderung auslösende Faden erzeugt eine **Laufgefahr** (*race condition*) durch zeitweilig inkonsistente TLB-Inhalte
  - für einen über die Rechenkerne verteilter mehr-/vielfädiger Prozess ist der IPI ein **Gruppenruf** (*multicast*) oder **Rundruf** (*broadcast*)

    ↪ skalierbarer verlässlicher Gruppenruf (*scalable reliable multicast*)

# Isolationsbetrieb „*considered harmful*“ . . .

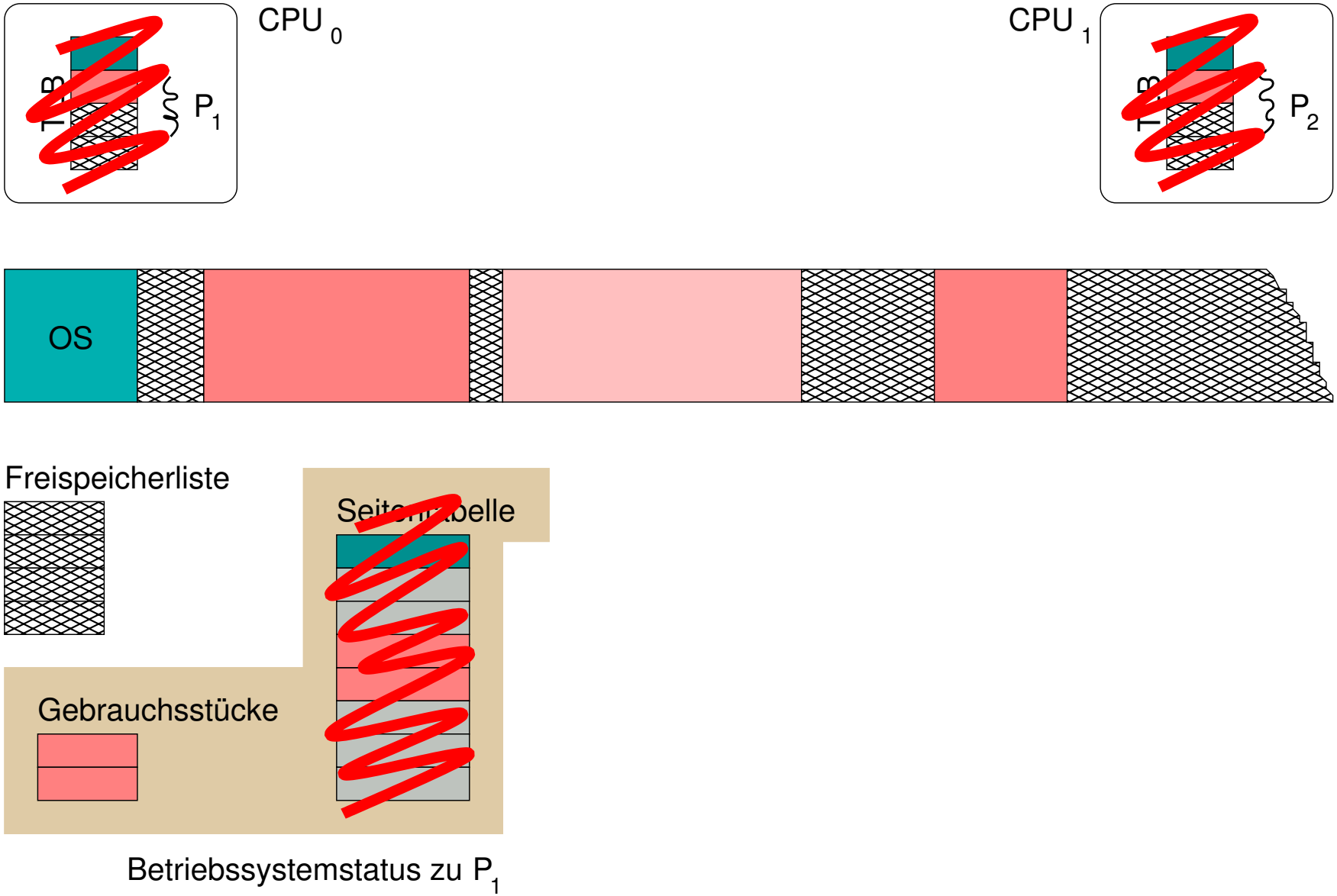
- Simultanverarbeitung mit einem Mehrprozessorsystem hat **Untiefen**
  - der eine Adressraumänderung auslösende Faden erzeugt eine **Laufgefahr** (*race condition*) durch zeitweilig inkonsistente TLB-Inhalte
  - für einen über die Rechenkerne verteilter mehr-/vielfädiger Prozess ist der IPI ein **Gruppenruf** (*multicast*) oder **Rundruf** (*broadcast*)
  - nicht zuletzt die durch den IPI verursachte **Störung** ferner Rechenkerne, die diese zu unbekannten Zeitpunkten mit unbekannter Frequenz trifft

# Offenbetrieb



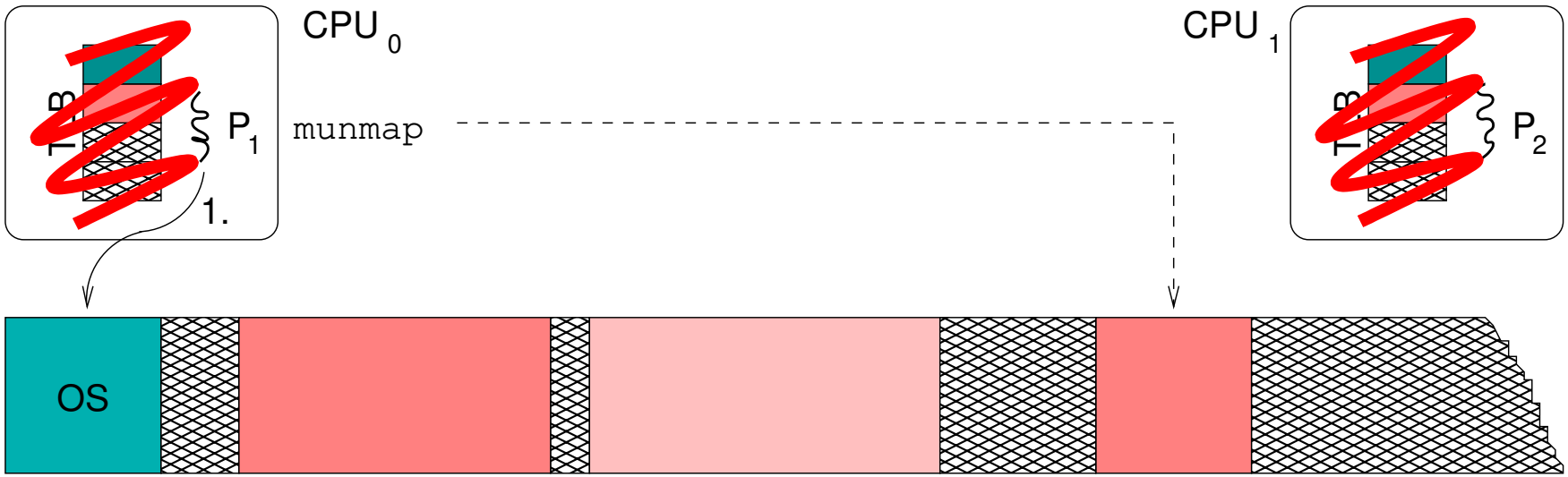
# Offenbetrieb

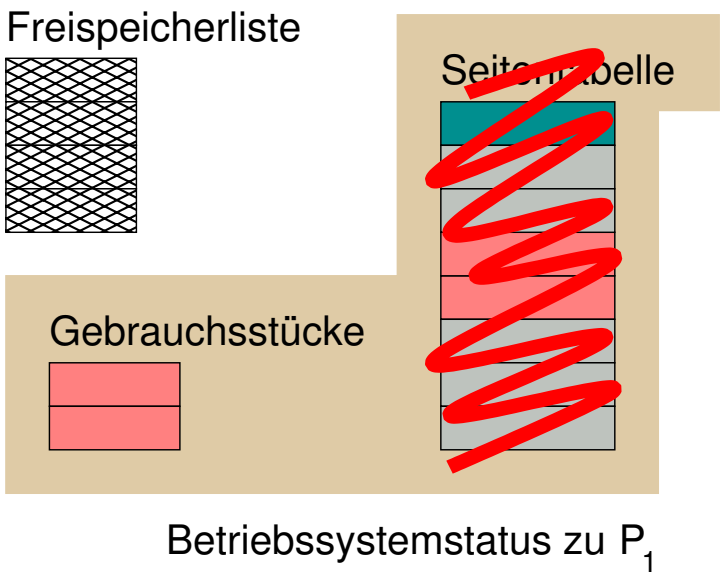




# Offenbetrieb





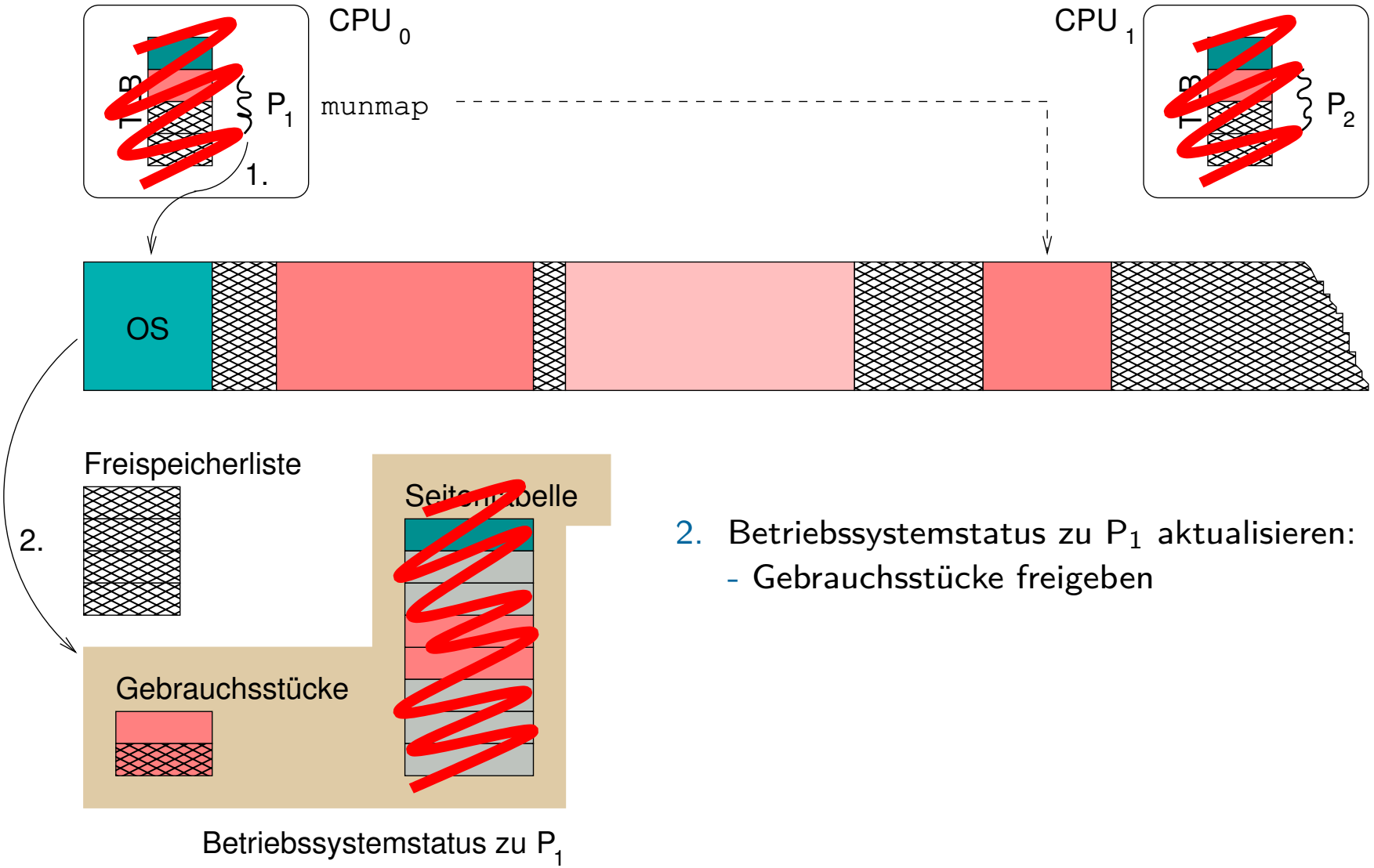




Betriebssystemstatus zu $P_1$

1. $P_1$ ruft `munmap()`

# Offenbetrieb

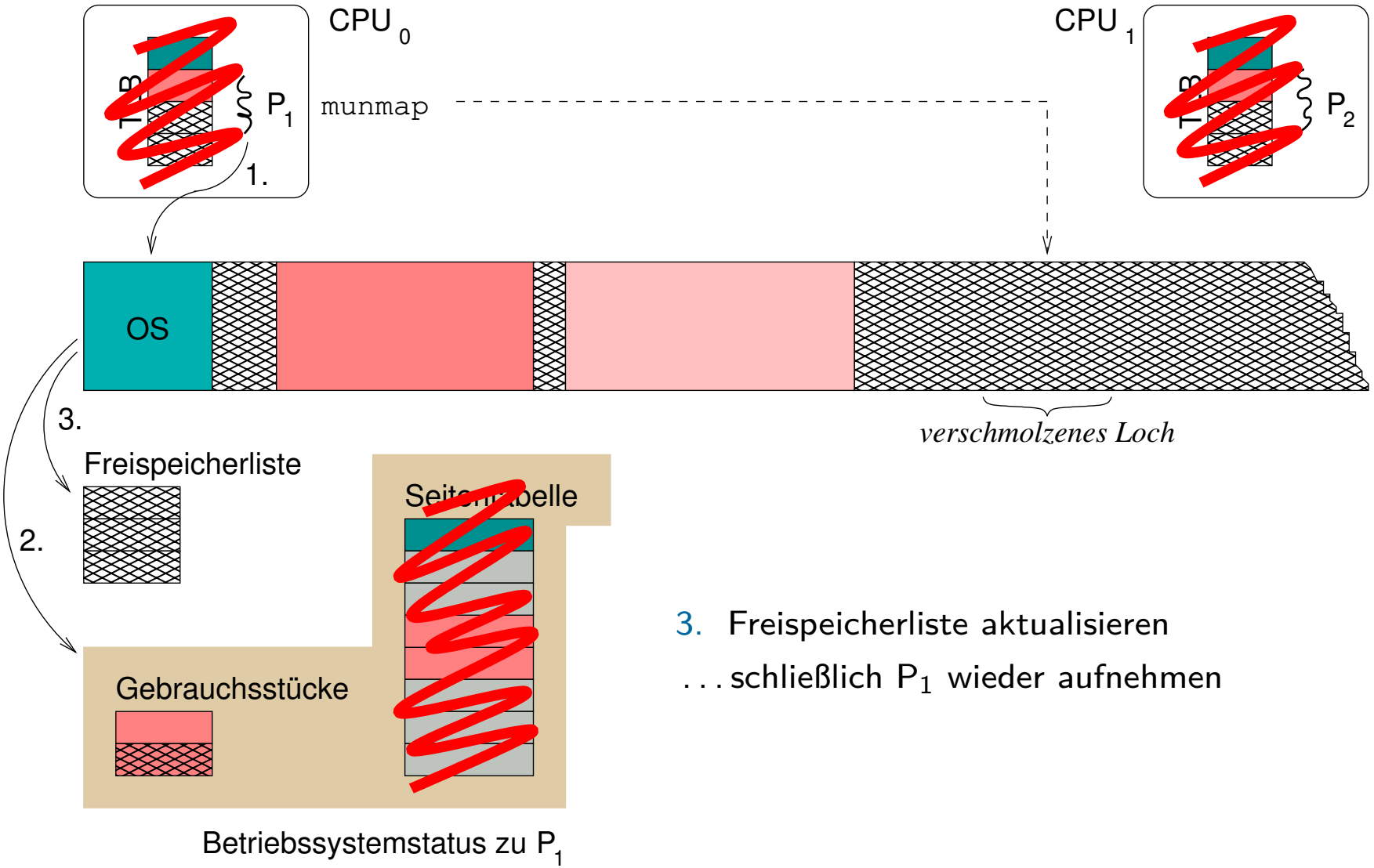





2. Betriebssystemstatus zu $P_1$ aktualisieren:
   - Gebrauchsstücke freigeben

# Offenbetrieb



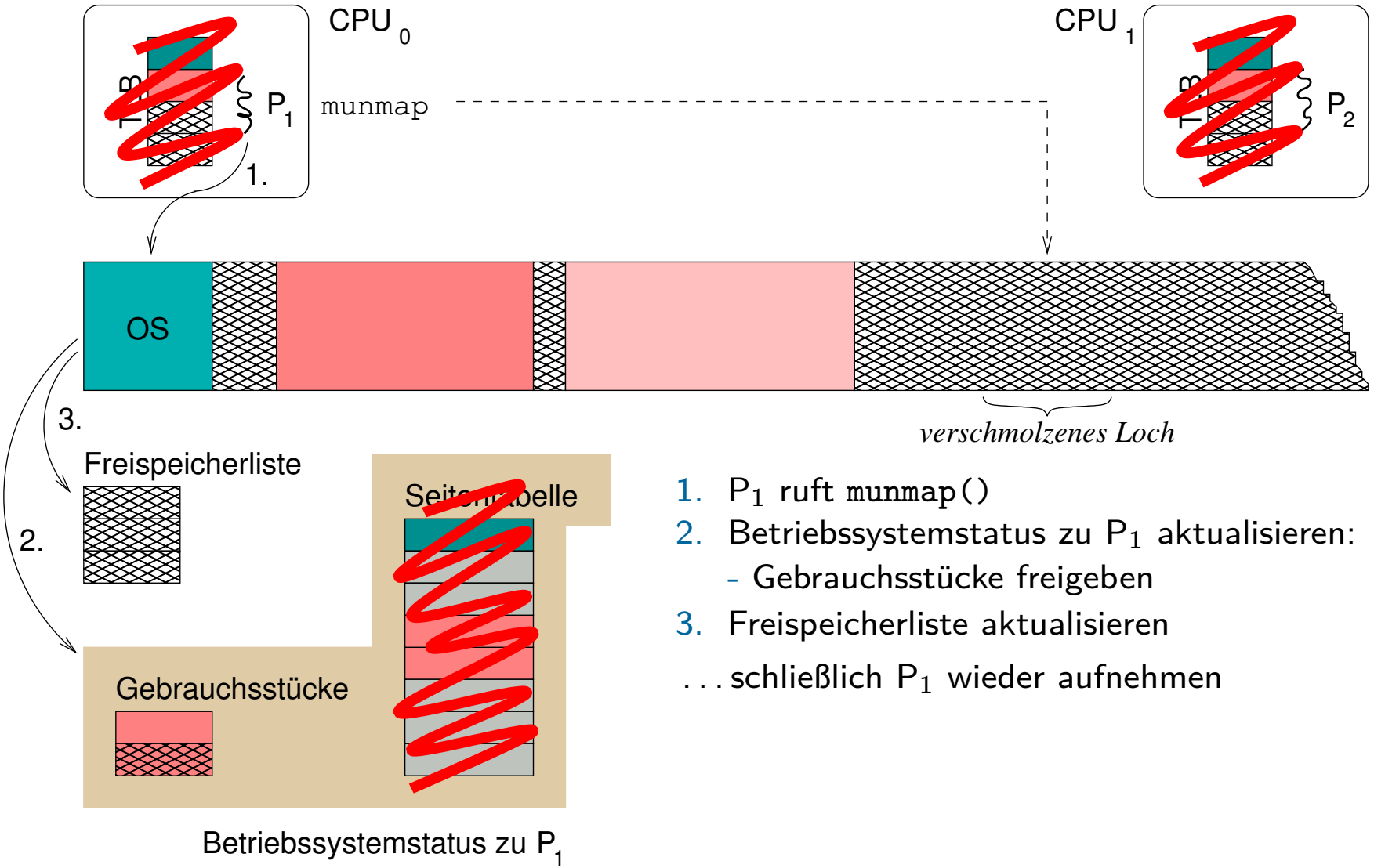



3. Freispeicherliste aktualisieren

. . . schließlich $P_1$ wieder aufnehmen

# Offenbetrieb



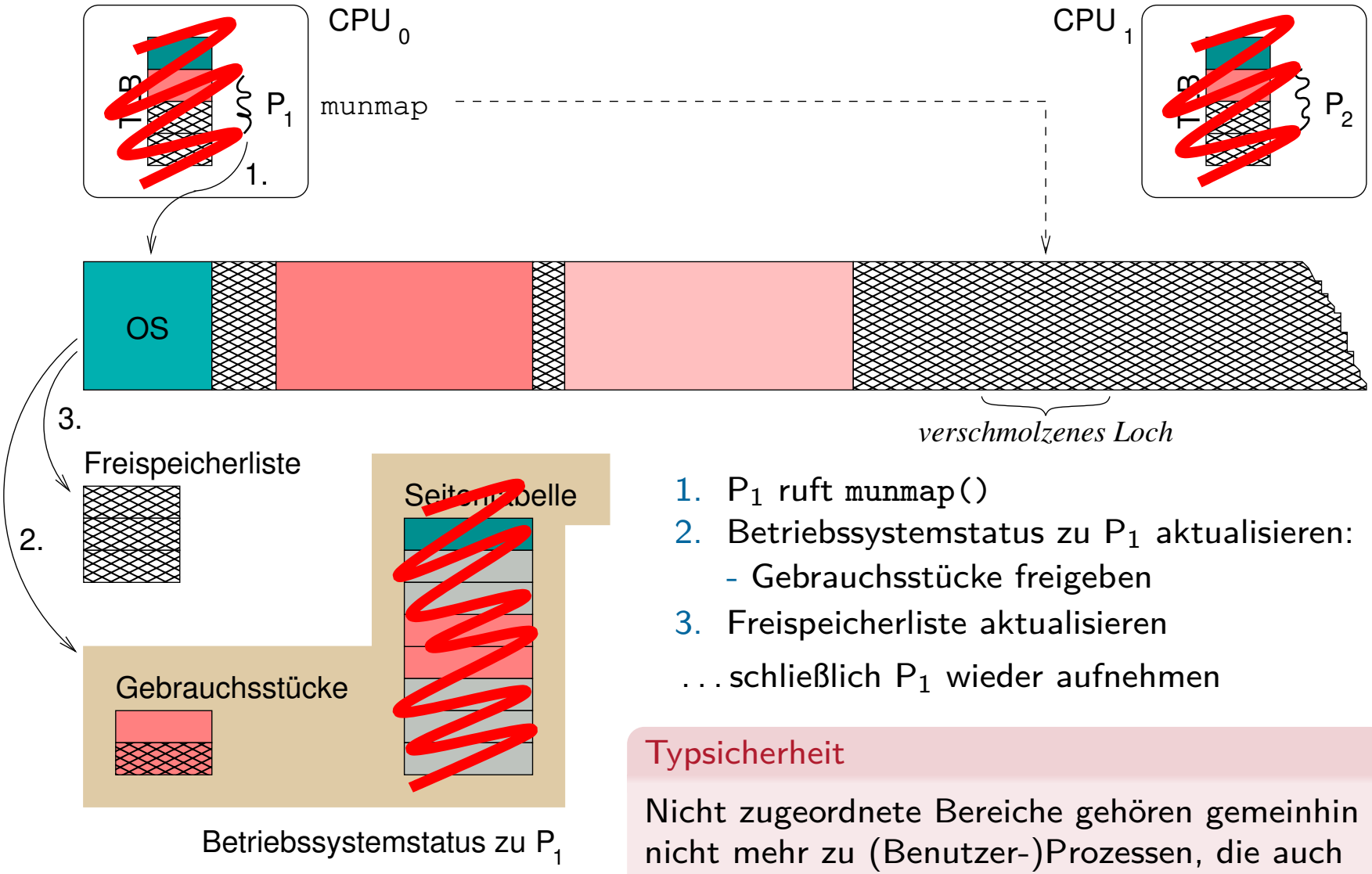



1. $P_1$ ruft `munmap()`
2. Betriebssystemstatus zu $P_1$ aktualisieren:
   - Gebrauchsstücke freigeben
3. Freispeicherliste aktualisieren

. . . schließlich $P_1$ wieder aufnehmen

# Offenbetrieb






1. $P_1$ ruft `munmap()`
2. Betriebssystemstatus zu $P_1$ aktualisieren:
   - Gebrauchsstücke freigeben
3. Freispeicherliste aktualisieren

. . . schließlich $P_1$ wieder aufnehmen

**Typsicherheit**

Nicht zugeordnete Bereiche gehören gemeinhin nicht mehr zu (Benutzer-)Prozessen, die auch niemals auf diese Regionen zugreifen sollten.

# Falsch positiver Zugriff bei Seitenadressierung

- die CPU greift **wortweise** auf den Hauptspeicher zu, der allerdings vom Betriebssystem **seitenweise** zur Verfügung gestellt wird

# Falsch positiver Zugriff bei Seitenadressierung

- die CPU greift **wortweise** auf den Hauptspeicher zu, der allerdings vom Betriebssystem **seitenweise** zur Verfügung gestellt wird
  - angenommen ein Prozess fordert 42 Bytes an dynamischen Speicher an, den das Betriebssystem nur in Einheiten von 4096 Bytes vergibt

# Falsch positiver Zugriff bei Seitenadressierung

- die CPU greift **wortweise** auf den Hauptspeicher zu, der allerdings vom Betriebssystem **seitenweise** zur Verfügung gestellt wird
  - angenommen ein Prozess fordert 42 Bytes an dynamischen Speicher an, den das Betriebssystem nur in Einheiten von 4096 Bytes vergibt
  - weiter angenommen, der angeforderte dynamische Speicher wird an einer Seitengrenze im Prozessadressraum verfügbar gemacht

# Falsch positiver Zugriff bei Seitenadressierung

- die CPU greift **wortweise** auf den Hauptspeicher zu, der allerdings vom Betriebssystem **seitenweise** zur Verfügung gestellt wird
  - angenommen ein Prozess fordert 42 Bytes an dynamischen Speicher an, den das Betriebssystem nur in Einheiten von 4096 Bytes vergibt
  - weiter angenommen, der angeforderte dynamische Speicher wird an einer Seitengrenze im Prozessadressraum verfügbar gemacht
  - sei `char*` *daaaf* Zeiger auf einen verfügbar gemachten Speicherbereich und `int(`*daaaf*`)`$/4096 = 0$, also ein seitenausgerichteter Wert

# Falsch positiver Zugriff bei Seitenadressierung

- die CPU greift **wortweise** auf den Hauptspeicher zu, der allerdings vom Betriebssystem **seitenweise** zur Verfügung gestellt wird

- so ist die Ausführung der Anweisungsfolge

```
1  daaaf = malloc(42);
2  if (daaaf)
3      for (i = 0; i < 1234; i++)
4          daaaf[i] = '*';
```

  für alle Werte von $i$ physisch gültig, obwohl nur $i \in [0, 41]$ logisch gilt

# Falsch positiver Zugriff bei Seitenadressierung

- die CPU greift **wortweise** auf den Hauptspeicher zu, der allerdings vom Betriebssystem **seitenweise** zur Verfügung gestellt wird

- so ist die Ausführung der Anweisungsfolge

```
1  daaaf = malloc(42);
2  if (daaaf)
3      for (i = 0; i < 1234; i++)
4          daaaf[i] = '*';
```

  für alle Werte von $i$ physisch gültig, obwohl nur $i \in [0, 41]$ logisch gilt
- das heißt, die ungültigen Zugriffe auf `daaaf[j]` mit $j \in [42, 4095]$ werden keinen Fehler liefern, da dieser Bereich noch zur gültigen Seite zählt

# Gliederung



# Bedarfsgerechter Speicherschutz

- Übergänge zwischen verschiedenen **Betriebsmodi** ermöglichen

# Bedarfsgerechter Speicherschutz

- Übergänge zwischen verschiedenen **Betriebsmodi** ermöglichen
  - MMU-basierten Schutz des Prozessadressraums ausschalten
    - Seitentabelle (für alle Fäden) des betreffenden Prozesses deaktivieren
    - sofern nötig, nur die globale (systemspezifische) Seitentabelle aktiv lassen

# Bedarfsgerechter Speicherschutz

- Übergänge zwischen verschiedenen **Betriebsmodi** ermöglichen

  - MMU-basierten Schutz des Prozessadressraums einschalten
    - Seitendeskriptoren auf Basis der Gebrauchsstücke wiederherstellen
    - Seitentabelle (für alle Fäden) des betreffenden Prozesses aktivieren
    - lokale (prozessspezifische) Seitentabelle anwenden

# Bedarfsgerechter Speicherschutz

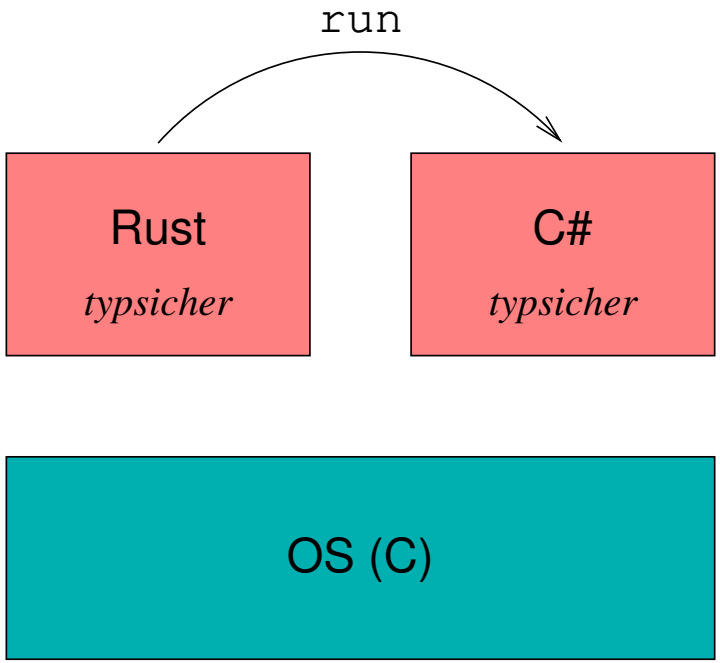
- Übergänge zwischen verschiedenen **Betriebsmodi** ermöglichen
  - MMU-basierten Schutz des Prozessadressraums ausschalten
  - MMU-basierten Schutz des Prozessadressraums einschalten
- Ein- und Ausschalten ist eine „lärmende Aktion“
  - IPI als Gruppenruf (*multicast*) an die relevanten Prozessoren senden
  - den TLB der jeweiligen MMU spülen (*flush*)

# Bedarfsgerechter Speicherschutz

- Übergänge zwischen verschiedenen **Betriebsmodi** ermöglichen
  - MMU-basierten Schutz des Prozessadressraums ausschalten
  - MMU-basierten Schutz des Prozessadressraums einschalten
- Ein- und Ausschalten ist eine „lärmende Aktion“ — jedoch „selten“
- die MMU isoliert nur Prozesse typunsicherer Programme
  - Prozesse typsicherer Programme sind durch den Kompilierer isoliert
  - so ergeben sich zwei Momente für den Wechsel des Betriebsmodus’
    - Ladezeitpunkt: typunsicheres Programm per MMU im Adressraum isolieren
    - Entsorgungszeitpunkt: Isolation des typunsicheren Programms aufheben

# Betriebsmodi

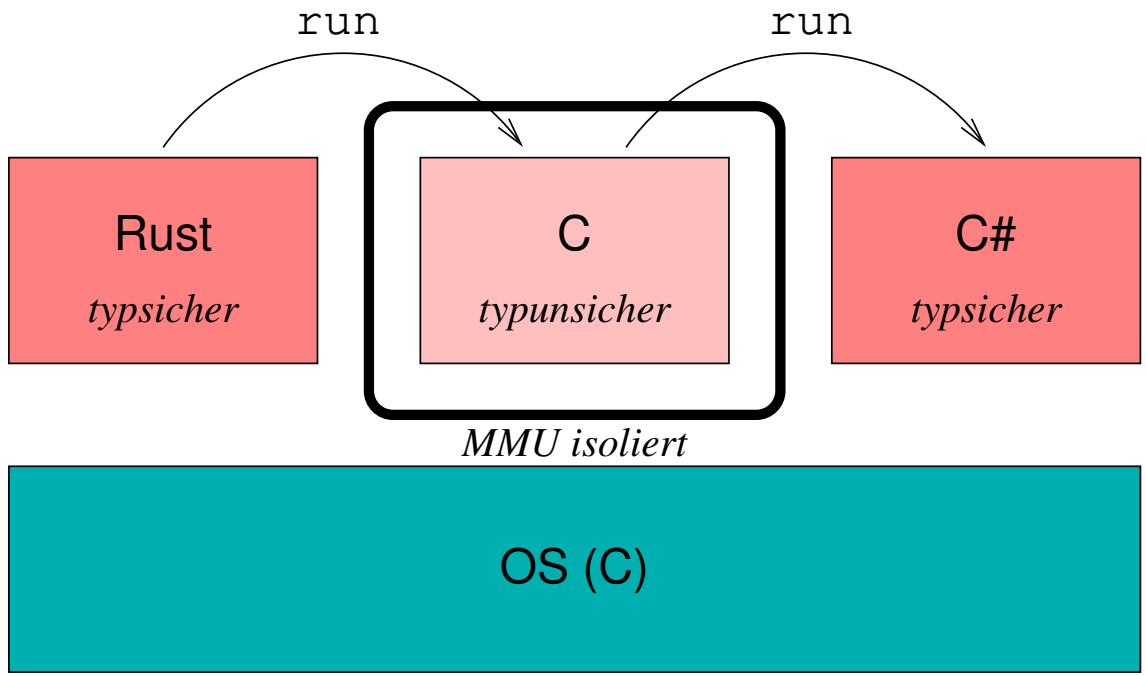





- Prozesse typischerer Anwendungsprogramme im Offenbetrieb belassen

# Betriebsmodi

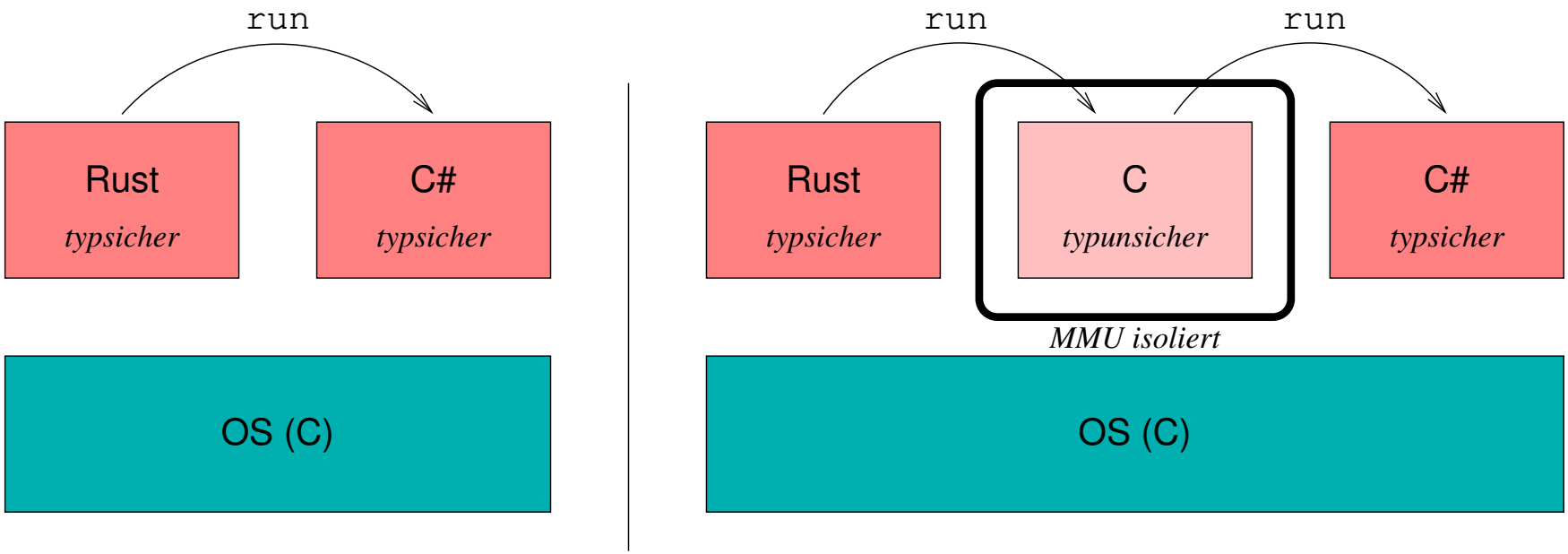





- nur die Prozesse typunsicherer Anwendungsprogramme beim Laden im eigenen Adressraum einschließen $\rightsquigarrow$ Isolationsbetrieb
  - Schutz der Umgebung vor solchen Prozessen

# Betriebsmodi






- Prozesse typischerer Anwendungsprogramme im Offenbetrieb belassen
- nur die Prozesse typunsicherer Anwendungsprogramme beim Laden im eigenen Adressraum einschließen $\rightsquigarrow$ Isolationsbetrieb

- Unabhängig von der Auslegung als MAS oder SAS, jedoch:

MAS
- **physische Isolation** der unsicheren Teile mittels MMU (hier)

SAS
- ggf. nur **logische Isolation** durch Adressraumrandomisierung

# Gliederung



# Resümee

# Resümee

- erzwungenes Unterbrechen von Prozessoren in ihren Aktivitäten, um dort TLB-Konsistenzaktionen durchzuführen, ist massiv störend
  - systemweiter IPI als Folge einer Adressraumänderung eines Fadens
  - der IPI unterbricht die laufende Aktion des empfangenden Rechenkerns
    - ***TLB shootdown*** [2]

# Resümee

- erzwungenes Unterbrechen von Prozessoren in ihren Aktivitäten, um dort TLB-Konsistenzaktionen durchzuführen, ist massiv störend

- diese Maßnahme forciert Überlagerungen beim Zusammentreffen der Unterbrechung mit einem gleichzeitigen Prozess
  - **Interferenz**, deren Zeitpunkt und Frequenz unbestimmt ist ☹
    - verursacht durch einen „externen“ Prozessfaden eines fernen Rechenkerns

# Resümee

- erzwungenes Unterbrechen von Prozessoren in ihren Aktivitäten, um dort TLB-Konsistenzaktionen durchzuführen, ist massiv störend

- diese Maßnahme forciert Überlagerungen beim Zusammentreffen der Unterbrechung mit einem gleichzeitigen Prozess
  - **Interferenz**, deren Zeitpunkt und Frequenz unbestimmt ist ☹
    - verursacht durch einen „externen“ Prozessfaden eines fernen Rechenkerns
  - die Störungen sind problematisch für zeitabhängige gleichzeitige Prozesse
    - beeinträchtigen aber auch allgemein die Performanz paralleler Anwendungen
    - insb. Anwendungen, die „Gleichschritt“ (*lock-step*) von Fäden erwarten

# Resümee

- erzwungenes Unterbrechen von Prozessoren in ihren Aktivitäten, um dort TLB-Konsistenzaktionen durchzuführen, ist massiv störend
- diese Maßnahme forciert Überlagerungen beim Zusammentreffen der Unterbrechung mit einem gleichzeitigen Prozess
- **speichersichere Prozesse** vor zu erwartenden Störungen bewahren, die ein **MMU-basierter Schutz** zur Folge haben kann
  - nur speicherunsichere Prozesse durch die MMU überwachen lassen
  - Adressraumisolation von Prozessen adaptiv gestalten [4]

# Resümee



- erzwungenes Unterbrechen von Prozessoren in ihren Aktivitäten, um dort TLB-Konsistenzaktionen durchzuführen, ist massiv störend
- diese Maßnahme forciert Überlagerungen beim Zusammentreffen der Unterbrechung mit einem gleichzeitigen Prozess
- **speichersichere Prozesse** vor zu erwartenden Störungen bewahren, die ein **MMU-basierter Schutz** zur Folge haben kann

# Literaturverzeichnis I


[1] Amdahl, G. M.:
Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities.
In: *Proceedings of the AFIPS Spring Joint Computer Conference (AFIPS 1967 (Spring))*, AFIPS Press, 1967, S. 483–485

[2] Black, D. L. ; Rashid, R. F. ; Golub, D. B. ; Hill, C. R.:
Translation Lookaside Buffer Consistency: A Software Approach.
In: *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*.
New York, NY, USA : Association for Computing Machinery, 1989 (ASPLOS III). – ISBN 0897913000, S. 113–122

[3] De Wael, M. ; Marr, S. ; De Fraine, B. ; Van Cutsem, T. ; De Meuter, W. :
Partitioned Global Address Space Languages.
In: *ACM Comput. Surv.* 47 (2015), Mai, Nr. 4, S. 62:1–27.
http://dx.doi.org/10.1145/2716320. –
DOI 10.1145/2716320


# Literaturverzeichnis II


[4] Drescher, G. :
*Adaptive Address-Space Management for Resource-Aware Applications*, Lehrstuhl für Verteilte Systeme und Betriebssysteme, Department Informatik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Dissertation, 2021.
https://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/16511

[5] Drescher, G. ; Schröder-Preikschat, W. :
Adaptive Memory Protection for Many-Core Systems.
Version: 2016.
http://dx.doi.org/10.4230/DagRep.6.10.120.
In: *Adaptive Isolation for Predictability and Security* Bd. 6.
Dagstuhl Publishing, 2016. –
DOI 10.4230/DagRep.6.10.120, S. 120–153:140

[6] Goldberg, R. P.:
Architectural Principles for Virtual Computer Systems / Harvard University, Electronic Systems Division.
Cambridge, MA, USA, Febr. 1973 (ESD-TR-73-105). –
PhD Thesis


# Literaturverzeichnis III


[7] Kremenetsky, M. ; Raefsky, A. ; Reinhardt, S. :
Poor Scalability of Parallel Shared Memory Model: Myth or Reality?
In: *Proceedings of the 2003 International Conference on Computational Science (ICCS 2003)* Bd. LNCS 2660, Springer-Verlag Berlin Heidelberg, 2003 (Lecture Notes in Computer Science), S. 657–666

[8] Schröder-Preikschat, W. :
Hierarchien.
In: Lehrstuhl Informatik 4 (Hrsg.): *Betriebssystemtechnik — Adressräume: Trennung, Zugriff, Schutz.*
FAU Erlangen-Nürnberg, 2013 (Vorlesungsfolien), Kapitel 4

[9] Sutter, H. :
Turn Toward Concurrency in Software—Your free lunch will soon be over. What can you do about it?
In: *Dr. Dobb's Journal* 30 (2005), März, Nr. 3, S. 16–22