# Kapitel 7: Virtuelle Speicherverwaltung

# Disclaimer

## Disclaimer

Die Inhalte dieses Kapitel basieren auf den Vorlesungsunterlagen für das Fach Computerarchitektur und Betriebssysteme von Andreas Wilkens

# 7: Virtuelle Speicherverwaltung

Gründe für eine virtuelle Speicherverwaltung

1. Ein Prozess sollte auch dann noch ablaufen können, wenn er nur teilweise im Hauptspeicher ist.

   Es reicht aus, wenn die Teile des Prozesses (Daten und Code) im physikalischen Speicher sind, die gerade benötigt werden.

2. Der Speicherbedarf eines Prozesses kann größer als der physikalisch vorhandene Hauptspeicher sein.

3. Beim Linken können für Symbole statische Adressen vergeben werden (Z.B. `main()` wird immer an die Adresse `0x400000` geladen)

4. Programmierer sehen einen kontinuierlichen (linearen) Speicherbereich, der bei Adresse 0 beginnt.

# Physikalischer und Virtueller Speicher

## Definition 8.1 (Physikalischer Speicher)

Dem im Computer verbauten Speicher (RAM) welcher über einen Bus direkt von der CPU oder der MMU angesprochen werden kann.

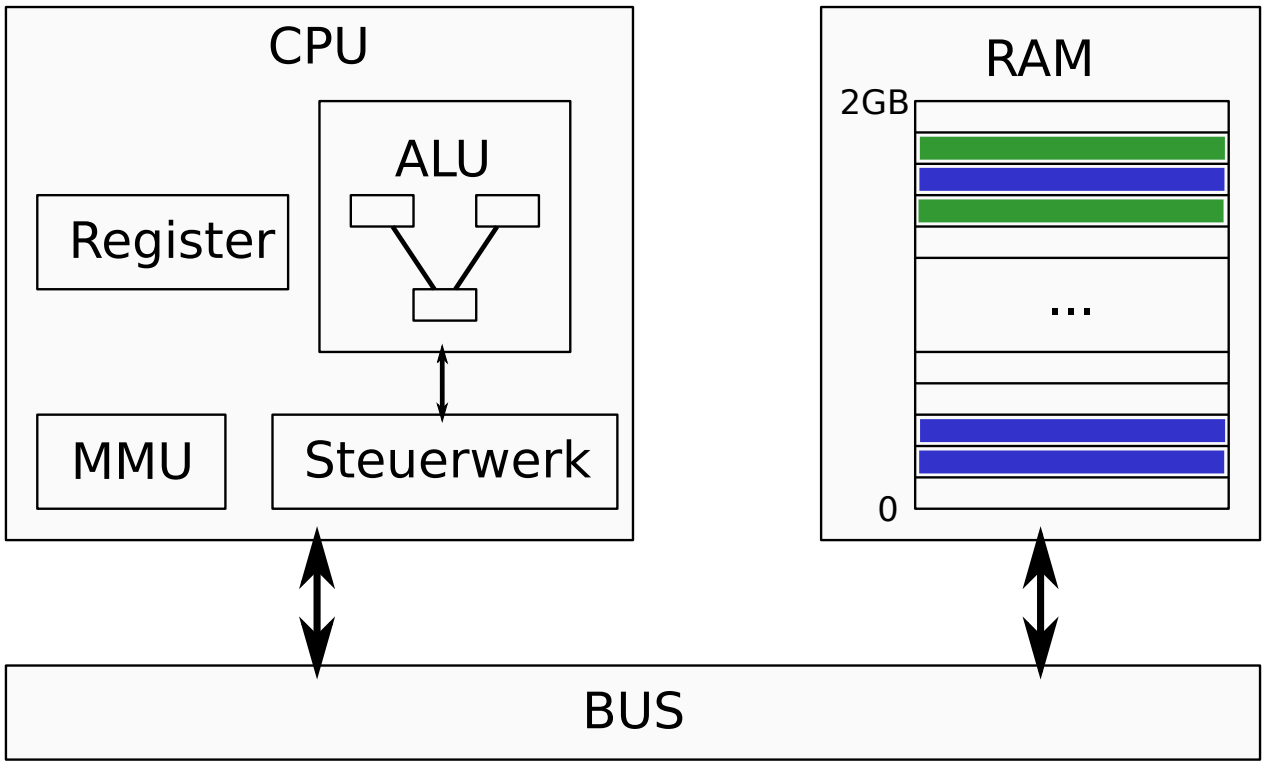
## Definition 8.2 (Virtueller Speicher)

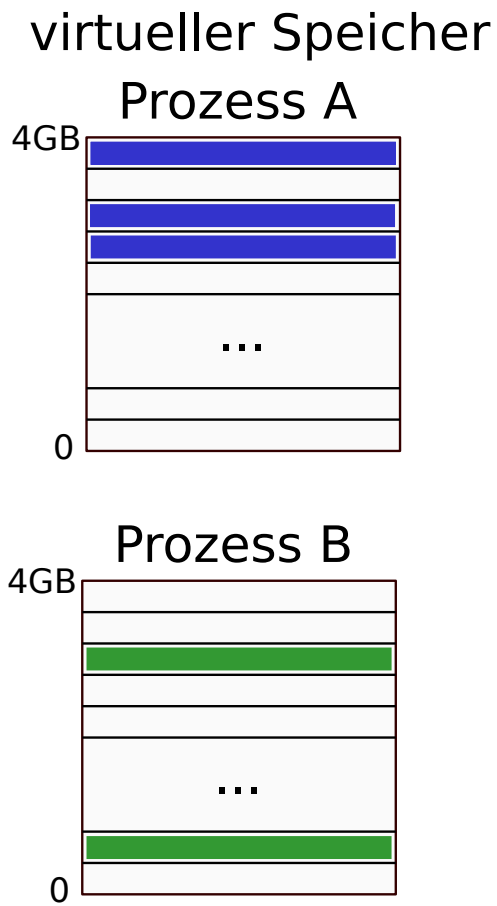
Speicherbereich der einem Prozess durch das Betriebssystem zur Verfügung gestellt wird.

## Anmerkung

Der physikalische Speicher bezieht sich auf den Computer, während der virtuelle Speicher auf einen Prozess bezogen wird!

# Illustration: Virtueller Speicher



# Speicheradressen

## Definition 8.3 (Physikalischer Speicheradresse)

Eine Adresse innerhalb des physikalischen Speichers eines Rechners.

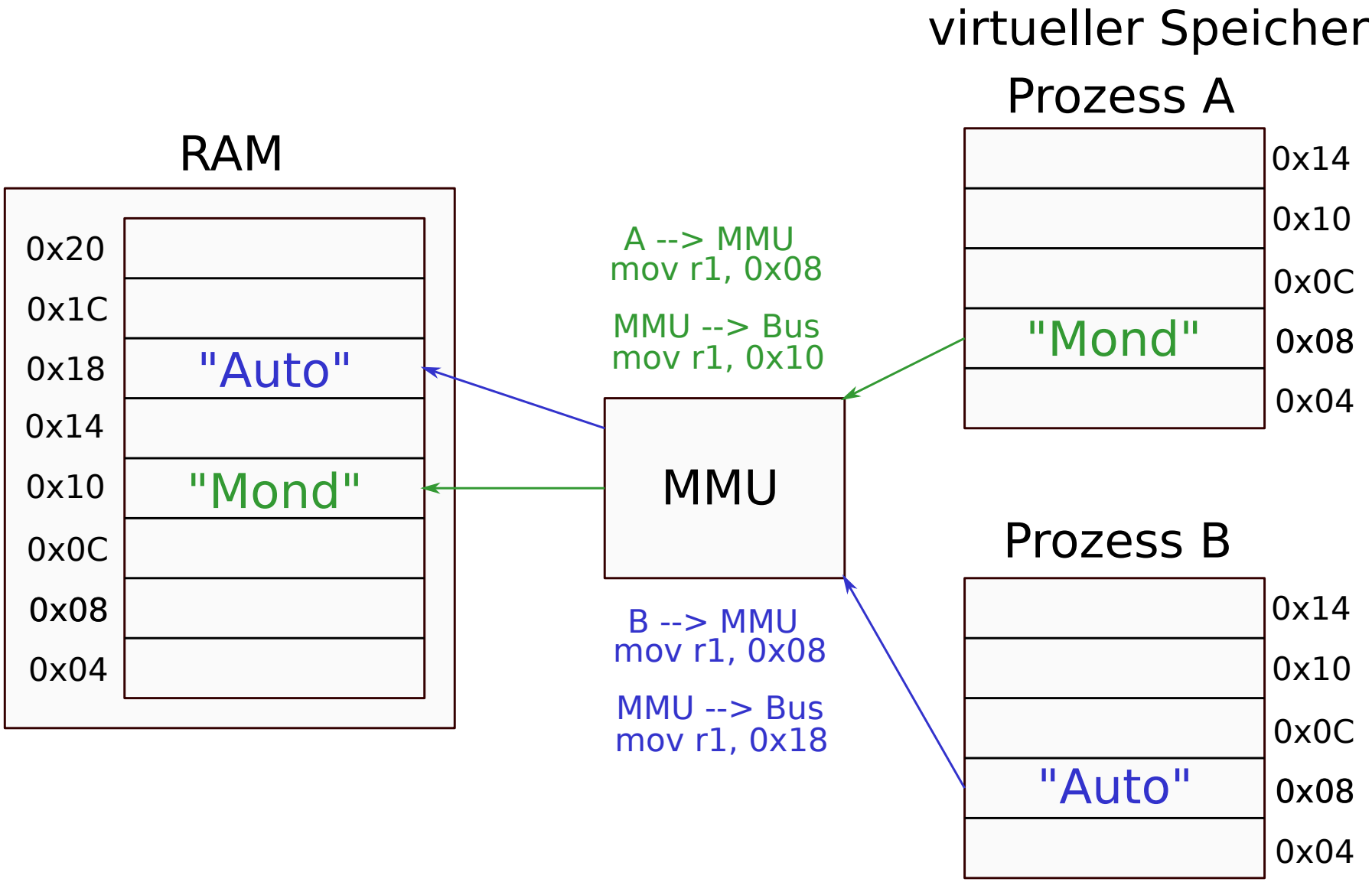
## Definition 8.4 (Virtuelle Speicheradresse)

Eine Adresse innerhalb des virtuellen Speichers eines Prozesses.

## Anmerkung

- Eine physikalische Adresse ist eindeutig, d.h. es gibt sie nur einmal pro Rechner.
- Eine virtuelle Adresse ist nur innerhalb eines Prozesses eindeutig.
- Die Memory Management Unit (MMU) übersetzt eine physikalische in eine virtuelle Adresse.

# Illustration: Grobe Funktionsweise einer MMU



# Speicherverwaltung

## Definition 8.5 (Speicherverwaltung)

Die (Haupt-)Speicherverwaltung ist Teil des Betriebssystems und erledigt alle erforderlichen Aufgaben zur Verwaltung des physikalischen und des virtuellen Speichers.

## Anmerkung

Virtueller Speicher existiert nur in der Vorstellung! Alle Daten müssen entweder im physikalischen Speicher abgelegt werden, oder auf einen Hintergrundspeicher (wie z.B. die Festplatte) ausgelagert sein.

# Seiten und Seitenrahmen

## Definition 8.6 (Seitenrahmen (*engl. Pageframe*))

Ein Seitenrahmen (kurz: Frame oder Rahmen) ist ein Zusammenhängenden Block von Speicherzellen des physikalischen Speichers.

## Definition 8.7 (Seite (*engl. Page*))

Zusammenhängender Block von Speicherzellen des virtuellen Speichers. Die Blockgröße einer Seite entspricht immer exakt der Größe eines Seitenrahmens.

## Anmerkung

Die übliche Größe eines Seitenrahmens ist 4096 Byte.

# Ermittlung der aktuellen Seitengröße

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main() {
5    printf("page␣size:␣%ld␣bytes\n", sysconf(_SC_PAGESIZE));
6  }
```

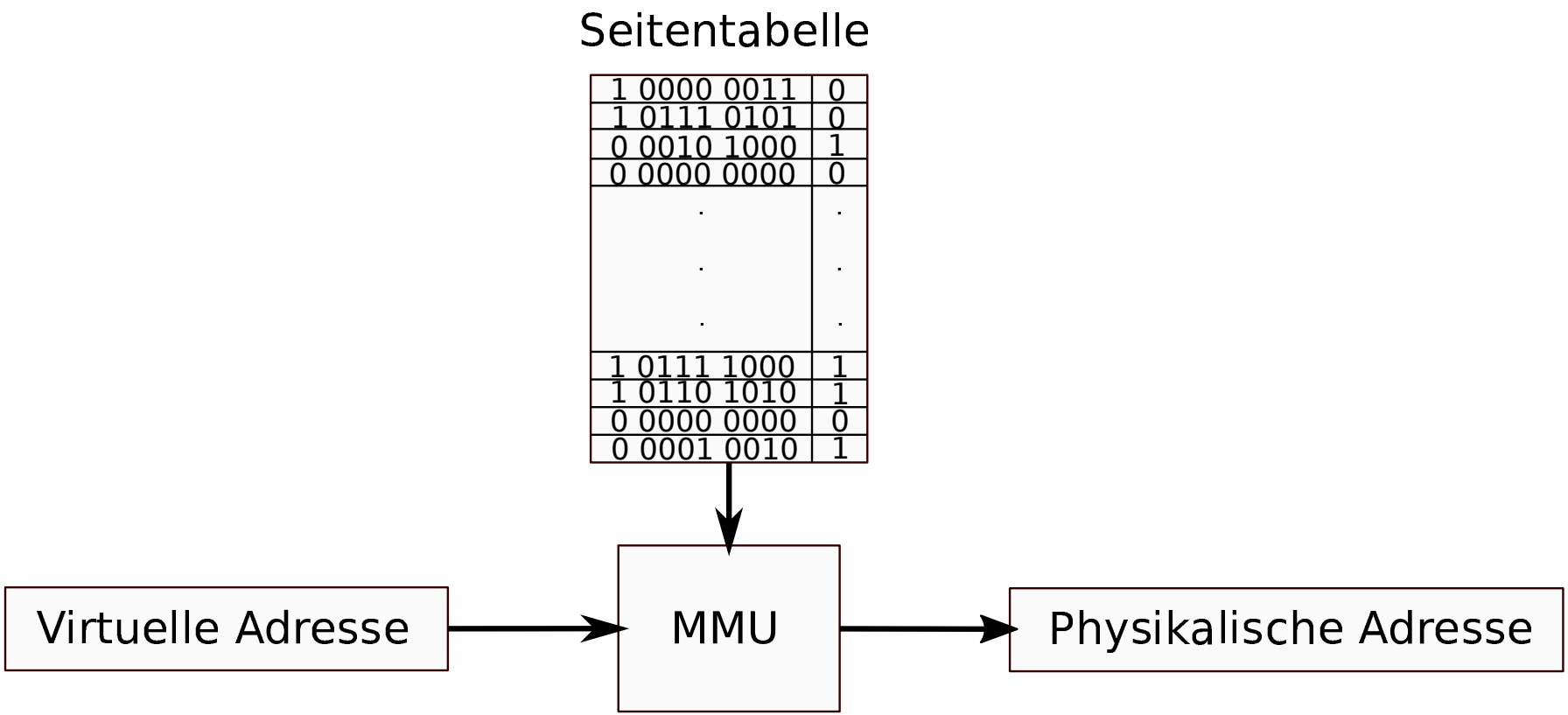
# 7.1: Seitentabellen

- Die MMU *rechnet* virtuelle Adressen mit Hilfe von Seitentabellen in physikalische Adressen um.
- Für jeden Prozess existiert eine eigene Seitentabelle.
- In der Seitentabelle wird Buch geführt, wo sich eine Seite momentan befindet.
- Je nach Betriebssystem kommen entweder einstufige Seitentabellen oder mehrstufige Seitentabellen zum Einsatz.
- In der Vorlesung behandeln wir nur einstufige Seitentabellen.
- Mehrstufige Seitentabellen dienen der Optimierung.

# Einstufige Seitentabellen

- Die einstufige Seitentabelle hat für jeden Seite einen Eintrag.
- Ein Eintrag (Zeile) enthält die folgenden Informationen
  - Seitenrahmen-Nr.
  - Present-/Absent-Bit (1=Gültig, 0= Ungültig)
- Die Seitentabellen sind im RAM abgelegt.
- Die MMU hat ein Adressregister welches die Startadresse der Seitentabelle des momentan laufenden Prozesses enthält.

# Illustration: Arbeitsweise der MMU



# Beispiel: Virtuelle Speicherverwaltung (1/2)

- Arbeitsspeicher: 512 MB
- Virtueller Speicher: 4096 MB (4 GB)
- Seitengröße: 64 KB
- Anzahl Seitenrahmen: $\frac{512MB}{64KB} = \frac{2^{29}}{2^{16}} = 2^{13}$
- Seiten pro Prozess: $\frac{4096MB}{64KB} = \frac{2^{32}}{2^{16}} = 2^{16}$
- Länge einer virtuellen Adresse: 32 Bit
  - Die ersten 16 Bit: Seite.
  - Die letzten 16 Bit: Byte-Offset.
- Länge einer physikalischen Adresse: 29 Bit
  - Die ersten 13 Bit: Seitenrahmen.
  - Die letzten 16 Bit: Byte-Offset.

# Beispiel: Virtuelle Speicherverwaltung (2/2)

| Seitenrahmen-Nr. (Frame-No.) | Present-/ Absent-Bit |
|---|---|
| 0 1111 0000 0011 | 1 |
| 0 1100 0111 0101 | 0 |
| 1 0101 0011 1110 | 0 |
| 0 1001 1111 1001 | 1 |
| . | . |
| . | . |
| . | . |
| 0 0000 0000 0000 | 0 |
| 1 1111 1111 1111 | 0 |
| 0 0000 1111 1111 | 0 |
| 1 0000 0111 1000 | 1 |
| 1 1001 1001 0000 | 1 |
| 0 0000 0000 0000 | 0 |
| 0 0000 1001 1111 | 1 |


Umrechnung: Virtuelle Adresse $\rightarrow$ physikalischen Adresse

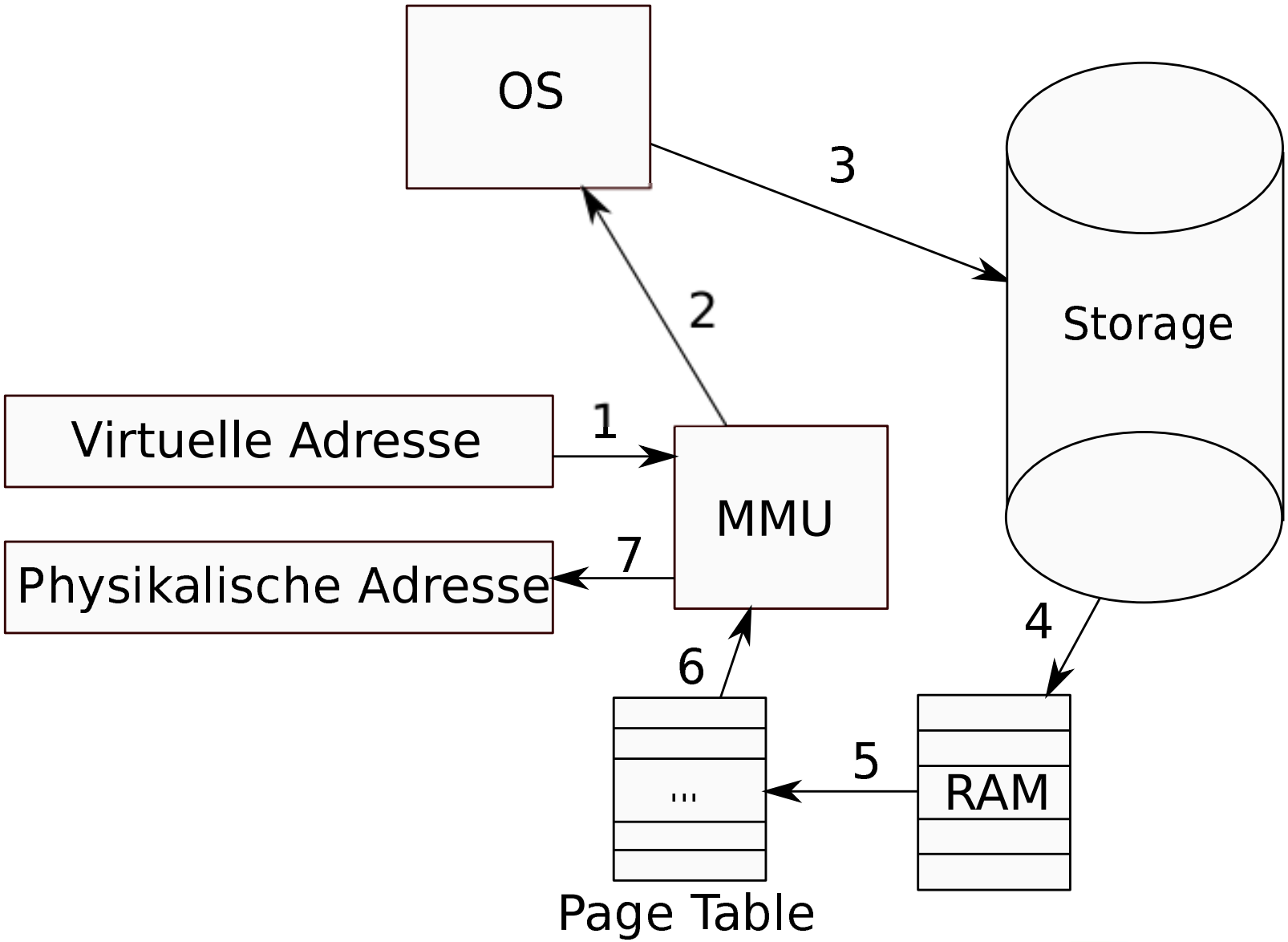
Virtuelle Adresse: 0000 0000 0000 0011 0011 1100 0000 1111

Physikalischen Adresse: 1 0000 0111 1000 0011 1100 0000 1111

# Pagefault

- Wenn das Present-/Absent Bit 0 ist, löst die MMU die Exception Pagefault aus.
- Gründe für Page Fault
  - Programmierfehler: Zugriff auf nicht allozierten Speicher.
  - Die Seite wurde ausgelagert, d.h. wurde durch die Seite eines anderen Prozesses verdrängt. (**Warum?**)
- Wurde die Seite ausgelagert, wird Sie durch den Pagefault wieder eingelagert und die MMU darf erneut die Adresse auflösen.

# Illustration Pagefault



# Anmerkung und Fragen

## Anmerkung

In der Praxis hat ein Seitentabelleneintrag noch weiter Informationen.

**Fragen?**

- Welche Vor- und Nachteile ergeben sich bei sehr großen Seiten?
- Welche Vor- und Nachteile ergeben sich bei sehr kleinen Seiten?

# Swapping und Paging

## Definition 8.8 (Swapping)

Das Aus- bzw. Einlagern eines kompletten Prozesses.

## Definition 8.9 (Paging)

Ein- bzw. Auslagern von Teilen (Seiten) eines Prozesses.

Eine Seite kann . . .

- in einem Seitenrahmen (Arbeitsspeicher) eingelagert sein
- auf einen Hintergrundspeicher (z.B. HDD oder SDD) ausgelagert sein.

- **Frage:** Warum finden Ein- und Auslagerungen statt?
- **Frage:** Welche Änderung muss der Kernel bei einer Auslagerung an der Seitentabelle vornehmen?

# 7.2: Seitenersetzungsstrategien

- Bei einem Pagefault muss die benötigte Seite eingelagert werden.
- Falls kein Seitenrahmen mehr frei ist, wird durch eine Seitenersetzungsstrategie ermittelt welche Seite ausgelagert werden muss, damit die benötigte Seite eingelagert werden kann.
- Verdrängungsstrategie ist ein Synonym für Seitenersetzungsstrategie.
- Im folgenden werden klassische Verdrängungsstrategien betrachtet.

# Genereller Ablauf einer Seitenersetzung

1. Die MMU stellt fest, dass die benötigte Seite **A** nicht in eingelagert ist und löst deshalb einen Pagefault aus.
2. Es ist kein freier Seitenrahmen mehr verfügbar.
3. Die zu ersetzende Seite **B** im Seitenrahmen *X* wird bestimmt.
4. **B** wird ausgelagert um Platz für **A** zu schaffen.
5. **A** wird in den Seitenrahmen *X* eingelagert.

## Anmerkung

- Das Auslagern von Seiten kostet viel Zeit!

  $\Longrightarrow$ Beeinträchtigung der Performance des Gesamtsystems.
- Seiten sollen nur Ausgelagert werden, falls dies wirklich notwendig ist.

# Einsparen von Auslagerung

1. Angenommen Seite **A** wurde bereits im Hintergrundspeicher abgelegt.
2. **A** wurde seit der letzten Einlagerung nicht modifiziert.
3. Der Seitenrahmen von **A** kann nun, ohne einer erneute Auslagerung, frei gegeben werden.
4. Modifizierte Seiten können nicht einfach ersetzt werden, sondern müssen zuvor im Hintergrundspeicher gesichert wurden.
5. Als nächsten beschäftigen wird uns damit wie das Betriebssystem feststellt, ob eine eingelagerte Seite modifiziert wurde.

# Das Modifiziert-Bit (M-Bit)

| Seitenrahmen-Nr. | Present-/Absent-Bit | M-Bit |
|---|---|---|
| 0 1111 0000 0011 | 1 | 0 |
| 0 1100 0010 1010 | 0 | 1 |
| 1 0101 1010 0010 | 0 | 0 |
| 1 0101 1010 0010 | 1 | 1 |
| ... | ... | ... |

Die Seitentabellen hat eine Spalte M-Bit (Modifiziert-Bit oder Dirty-Bit).

- M-Bit = 1: Der Inhalt der zugehörigen Seite wurde modifiziert.
- M-Bit = 0: Der Inhalt der zugehörigen Seite wurde nicht modifiziert.

# Setzen des M-Bits

- Falls die MMU eine Adresse für einen Schreibinstruktion umrechnet setzt sie das M-Bit für den Seitentabelleneintrag.

- Ist das M-Bit einer zu ersetzenden Seite gesetzt, so muss diese in den Hintergrundspeicher geschrieben werden.

# Optimaler Seitenersetzungsalgorithmus (1/2)

## Beobachtung

Es ist eine schlechte Idee eine Seite zu ersetzen welche bei der Ausführung der nächsten Assemblerinstruktion benötigt wird.
Eine solche Strategie würde die Performance des Rechners minimieren.

## Definition 8.10 (Bélády Algorithmus)

Ersetze die Seite, welche in Zukunft am längsten nicht mehr benötigt wird.

## Beobachtung

Bei der optimalen Seitenersetzungsstrategie werden in Zukunft am wenigsten weitere Seitenfehler ausgelöst.

# Beispiel

Seitenzugriffe ab dem Zeitpunkt t=0: 1,0, 2, 3, 1, 1, 2, 4

Seitentabelle zum Zeitpunkt t=0.

| Seitenrahmen-Nr. | Present-/Absent-Bit | M-Bit |
|---|---|---|
| 10 00 | 1 | 0 |
| 00 00 | 0 | 0 |
| 00 00 | 0 | 0 |
| 01 00 | 1 | 1 |
| 00 00 | 1 | 1 |
| 11 00 | 1 | 0 |
| 00 00 | 0 | 0 |
| 00 00 | 0 | 0 |

# Optimaler Seitenersetzungsalgorithmus (2/2)

- **Frage:** Wie lässt sich die optimale Seitenersetzungsstrategie implementieren?
- **Frage:** Wird bei der optimale Seitenersetzungsstrategie das M-Bit benötigt?

# First In First Out (FIFO)

- Bei der FIFO-Seitenersetzungsstrategie wird die Seite ersetzt, die bereits am längsten eingelagert ist.
- Einfach zu implementieren (Verkettete Liste)
- Lange Auslagerung ist kein Indiz dafür, dass eine Seite nicht in Kürze wieder benötigt wird.
- In der Praxis hat dieses Verfahren keine große Bedeutung.

# Beispiel FIFO

## Zustand zum Zeitpunkt $t = 0$

- Kommende Seitenzugriffe: 1,0, 2, 3, 1, 1, 2, 4
- FIFO t=0: 3 $\rightarrow$ 1 $\rightarrow$ 5 $\rightarrow$ 7

| Seitenrahmen-Nr. | Present-/Absent-Bit | M-Bit |
|---|---|---|
| 00 00 | 1 | 0 |
| 00 00 | 0 | 0 |
| 00 00 | 0 | 0 |
| 01 00 | 1 | |
| 10 00 | 1 | 1 |
| 11 00 | 1 | 0 |
| 00 00 | 0 | 0 |
| 00 00 | 0 | 0 |

# Bélády Anomalie

- Bei der FIFO-Strategie können sich die Anzahl der Pagefaults durch die Anzahl der Seitenrahmen erhöhen.

  Anomalie: Mehr Pageframes führt zu mehr Pagefaults

- Beispiel
  - Leere Seitentabelle / Seitenrahmen
  - Zugriffsreihenfolge: 1,2,3,4,1,2,5,1,2,3,4,5
  - 3 Seitenrahmen: 9 Pagefaults
  - 4 Seitenrahmen: 10 Pagefaults

# Second Chance

- Optimierung der FIFO-Seitenersetzungsstrategie
- Seitentabelle bekommt noch ein Referenziert-Bit (R-Bit).
- Das R-Bit wird gesetzt, wenn auf eine Seite zugegriffen wird.
- Bei Second Chance wird zunächst die längste eingelagerte Seite betrachtet (Erstes Element der FIFO-Datenstruktur).
  - R-Bit=0: Seite wird ausgelagert
  - R-Bit=1: R-Bit wird auf 0 gesetzt und nächstes Element der FIFO-Datenstruktur wird betrachtet.
- Bei manchen Varianten werden in regelmäßigen Abständen alle R-Bits auf 0 gesetzt.

# Beispiel Second-Chance

## Zustand zum Zeitpunkt $t = 0$

- Kommende Seitenzugriffe: 1,0, 2, 3, 1, 1, 2, 4
- FIFO t=0: $3 \rightarrow 1 \rightarrow 5 \rightarrow 7$

| Seitenrahmen-Nr. | Present-/Absent-Bit | M-Bit | R-Bit |
|---|---|---|---|
| 00 00 | 1 | 0 | 1 |
| 00 00 | 0 | 0 | 0 |
| 00 00 | 0 | 0 | 0 |
| 01 00 | 1 | 0 | 0 |
| 10 00 | 1 | 1 | 1 |
| 11 00 | 1 | 1 | 0 |
| 00 00 | 0 | 0 | 0 |
| 00 00 | 0 | 0 | 0 |

# Überlegungen zu dem R-Bit und M-Bit

## Überlegungen

- Ist das M-Bit gesetzt so muss die Seite auf den Hintergrundspeicher geschrieben werden.
- Ist das R-Bit gesetzt, wird davon ausgegangen das die Seite bald wieder benötigt wird.

# Vier Klassen von eingelagerten Seiten

Aus unseren Überlegungen ergeben sich vier Güteklassen.

- Klasse 0: Eingelagerte Seiten mit R-Bit=0 und M-Bit=0.
  Sehr gute Kandidaten zum ersetzen.
- Klasse 1: Eingelagerte Seiten mit R-Bit=0 und M-Bit=1.
  Ersetzen solcher Kandidaten ist OK.
- Klasse 2: Eingelagerte Seiten mit R-Bit=1 und M-Bit=0.
  Kein guten Kandidaten zum ersetzen.
- Klasse 3: Eingelagerte Seiten mit R-Bit=1 und M-Bit=1.
  Wenn möglich sollte diese Kandidaten nicht ersetze werden.

# Not Recently Used (NRU)

## Vorgehensweise der NRU-Strategie

1. Erstelle nach Klassen sortierte Liste aller eingelagerten Seiten
   - Klasse 0: R-Bit=0 und M-Bit=0
   - Klasse 1: R-Bit=0 und M-Bit=1
   - Klasse 2: R-Bit=1 und M-Bit=0
   - Klasse 3: R-Bit=1 und M-Bit=1
2. Lagere die erste Seite der Liste aus.
3. Setze die R-Bits aller Seiten auf 0.

# Beispiel: NRU

Seitenzugriffe ab dem Zeitpunkt t=0: 1,0, 2, 3, 1, 1, 2, 4

Seitentabelle zum Zeitpunkt t=0.

| Seitenrahmen-Nr. | Present-/Absent-Bit | M-Bit | R-Bit |
|---|---|---|---|
| 10 00 | 1 | 0 | 1 |
| 00 00 | 0 | 0 | 0 |
| 00 00 | 0 | 0 | 0 |
| 01 00 | 1 | 1 | 0 |
| 00 00 | 1 | 1 | 1 |
| 11 00 | 1 | 0 | 0 |
| 00 00 | 0 | 0 | 0 |
| 00 00 | 0 | 0 | 0 |

# Least Recently Used (LRU)

- Ersetze Seite welche am längsten nicht mehr benutzt wurde.
- Implementierungsstrategien
  - Jeder Seitentabelleneintrag bekommt einen Referenced-Timestamp.
  - Der Referenced-Timestamp muss bei jedem Seitenzugriff aktualisiert werden.
  - Die Seitentabelleneinträge werden in einer sortieren verketten Liste gespeichert.
  - Bei einem Seitenzugriff wird der entsprechende Seitentabelleneinträge an das Ende der Liste verschoben.
  - Seite am Listenanfang wird ersetzt.

# Beispiel: LRU

- Anzahl Seitenrahmen: 4
- Seitentabelle ist leer.
- Seitenzugriffe: 3, 6, 1, 0, 4, 1, 6, 1, 6, 2, 1, 1, 7, 0, 0, 3

# The Not Frequently Used (NFU)

- Die am wenigsten genutzt Seite wird ausgelagert.
- Jeder Seitentabelleneintrag bekommt einen Zähler (`ctr=0`) der ermittelt wie oft eine Seite genutzt wurde.
  - Einlagern der Seite: `ctr=0`
  - Periodisch: `ctr+=R-Bit; R-Bit=0`

# Working-Set

## Die 80/20-Regel

80 Prozent der von einer (komplexen) Software bereitgestellten Funktionen werden höchstens von 20 Prozent der Nutzer tatsächlich eingesetzt.

## Beobachtung

Viele nacheinander auf der CPU ausgeführten Befehlen nur relativ wenig verschiedene virtuelle Seiten angesprochen werden. Genau diese Seiten bilden nun das **Working Set** des betrachteten Prozesses.

Der Working Set Algorithmus versucht nun, alle zum **Working Set** eines Prozesses gehörenden Seiten ständig im Hauptspeicher zu halten.

# Working-Set Implementation

- Lösche periodisch alle R-Bits.
- R-Bit=1: Seite wurde im aktuellen Intervall genutzt und gehört daher zum Working Set des Prozesses.
- R-Bit=0: Seite wurde im aktuellen Intervall NICHT genutzt und gehört daher NICHT zum Working Set des Prozesses.
- Ersetze eine Seite bei dem das R-Bit nicht gesetzt ist.

# Weiterführende Literatur

Der Kollege John Bell (University of Illinois, Chicago) hat ergänzendes Lehrmaterial zu diesem Thema ins Netz gestellt.

- Hauptspeicher
- Virtueller Speicher