Compilieren, Linken und Laden eines C-Programms mit zwei Dateien: Beispiel

	Datei1		Datei2	
Quellcode:	eineFunktion(...)		globalVar = 10;	
Assembler:	CALL 34	(BL 34)	MOVE [34], R0	(STR …)
Linker:	CALL 34	(BL 34)	MOVE [134], R0	(STR …)
Loader:	CALL 4034	(BL 4034)	MOVE [4134], R0	(STR …)

Anmerkungen:

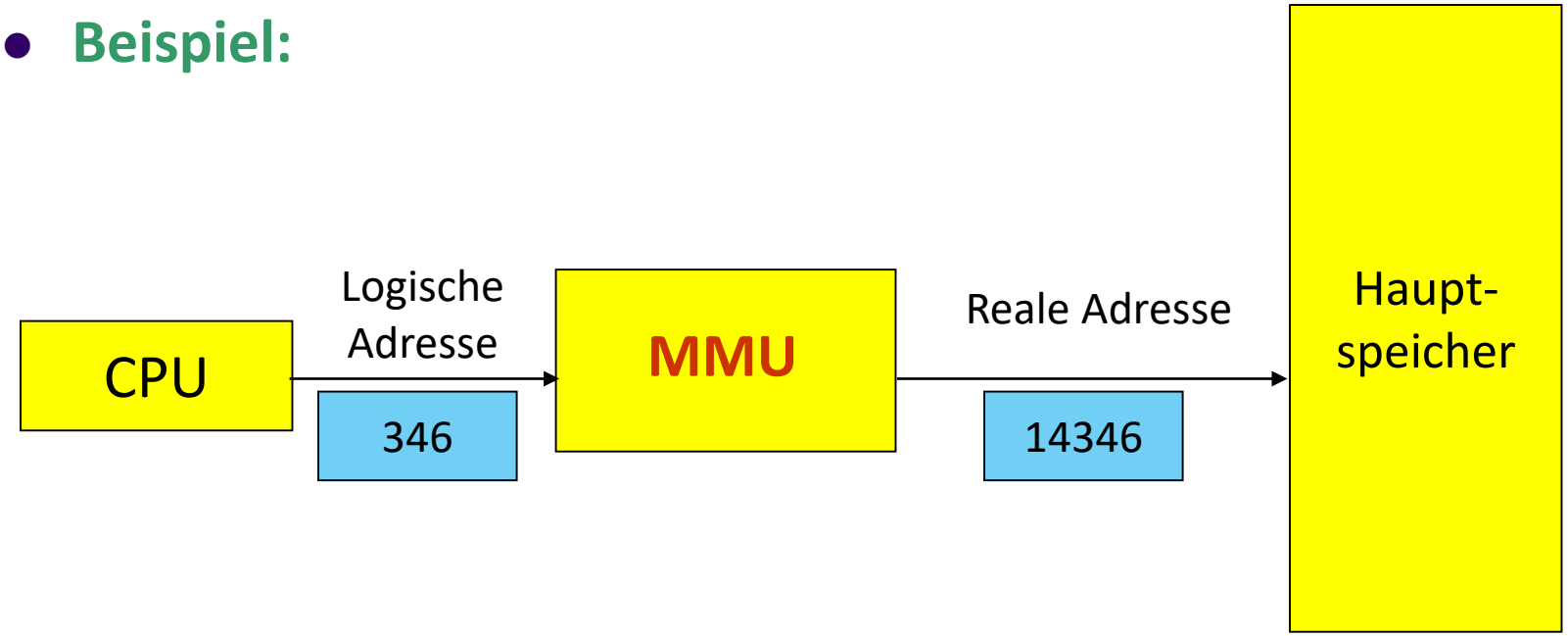
- Jede Datei definiert hier ein Modul.
- **Datei 1:** Modul liegt am Anfang → Offset=0 → „34“ **Datei 2:** 100 Byte reichen dem Linker zum Verschieben der Adressräume, da im Beispiel angenommen wird, dass die max. genutzte Adresse von Datei1 99 ist.
- Statische Adressumsetzung mit Basisregister. Wert: 4000

Adressen

- **Logische Adressen (virtuelle Adressen)**
  - Referenz auf eine Speicheradresse, ohne dass die reale (absolute) Hauptspeicheradresse bekannt ist
  - Eine „Übersetzung“ muss vom System (Betriebssystem oder Hardware) vorgenommen werden
- **Reale Adressen**
  - Die absoluten Adressen im physikalischen Hauptspeicher

Dynamische Adressberechnung zur Relokation von Programmen

- Abbildung einer logischen Adresse auf die reale Adresse durch Hardware:
  Aufgabe der **MMU** (Memory Management Unit)
- **Beispiel:**



Einfache Adressumsetzungsmethode

- Basis-Register („Relocation“-Register)
  - Enthält die Startadresse des Prozesses
  - Berechnungsverfahren:
    Logische Adresse + Wert des Basis-Registers = Reale Adresse
- Limit-Register
  - Endadresse des Prozesses
  - **Wenn Reale Adresse > Limit-Register ➔ Fehler!**
    (Schutzverletzung)

Diese Register werden gesetzt, wenn der Prozess geladen oder verschoben wird!

Aufgaben Abschnitt 4.1.b): Adressberechnung

Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 1

1. In ▶Abbildung 3.3 enthalten das Basis- und das Limitregister den gleichen Wert: 16.384. Ist dies reiner Zufall oder sind die Inhalte immer gleich? Wenn es nur Zufall ist, warum sind es dann in diesem Beispiel dieselben Werte?

Einfache Adressumsetzungsmethode

- Basis-Register („Relocation“-Register)
  - Enthält die Startadresse des Prozesses
  - Berechnungsverfahren:
    Logische Adresse + Wert des Basis-Registers = Reale Adresse
- Limit-Register
  - Endadresse des Prozesses
  - **Wenn Reale Adresse > Limit-Register ➔ Fehler!**
    (Schutzverletzung)

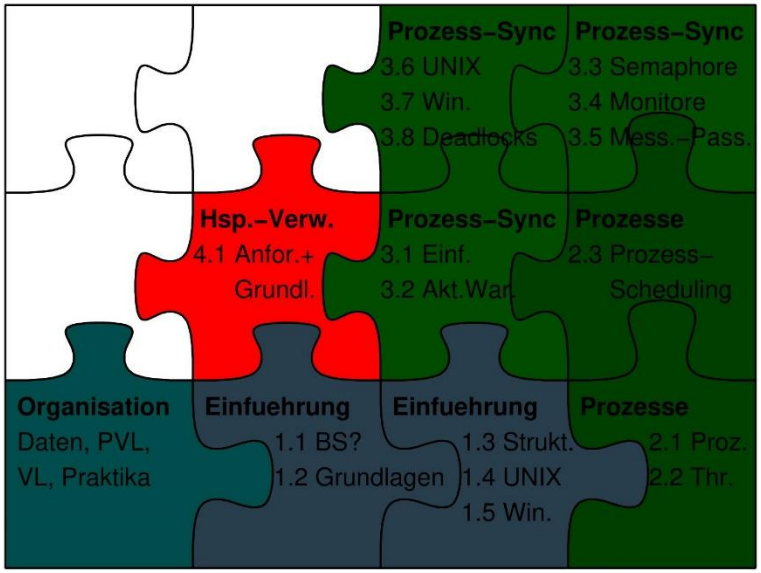
Diese Register werden gesetzt, wenn der Prozess geladen oder verschoben wird!

Frage aber jetzt: wie sind Basis-/Limit-Register zu wählen??

➔ Zuweisungs-/Freigabeverfahren

Kapitel 4
Hauptspeicher-Verwaltung

1. Grundlagen
   a) Anforderungen
   b) Adressberechnung
   c) Einfache Zuweisungs- und Freigabeverfahren
   d) Implementierungsaspekte
2. Virtueller Speicher
   a) Einführung und Prinzipien
   b) Paging
   c) Pagingstrategien
   d) Unix / Windows
   e) Meltdown / Spectre

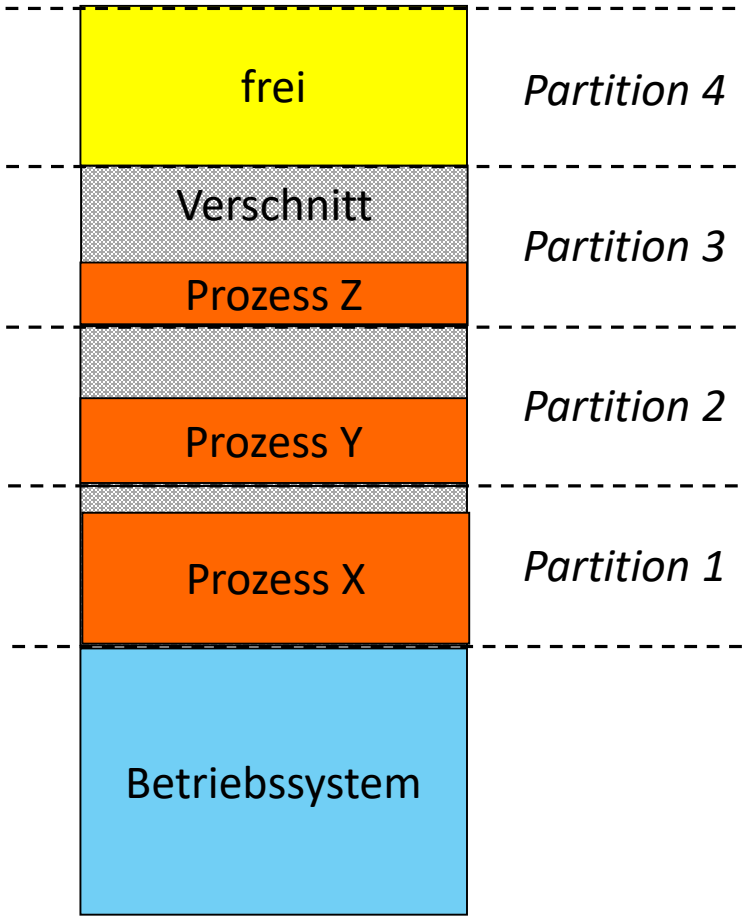


Hauptspeicheraufteilung bei Multiprogramming: Feste Partitionierung

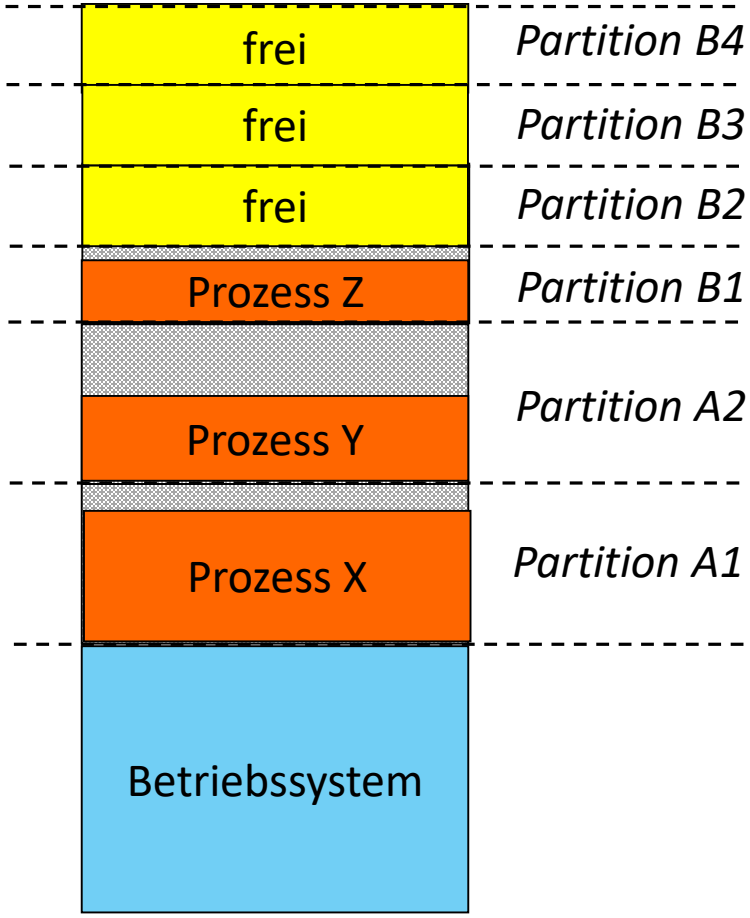
- Aufteilung in **feste** Anzahl Partitionen (Teile)
  - Jeder Prozess belegt genau eine Partition
  - Jeder Prozess, dessen Platzbedarf $\leq$ der Größe einer freien Partition ist, kann geladen werden
  - Wenn alle Partitionen voll sind, kann das Betriebssystem einzelne Prozesse leicht aus-/ einlagern
  - Varianten bzgl. der **Partitionsgröße**:
    - Alle Partitionen haben eine einheitliche Größe
    - Es gibt unterschiedliche Partitionsgrößen
      → Verringerung des nicht-nutzbaren Hauptspeichers („Verschnitt“)

Beispiele: Feste Partitionierung

Feste Hauptspeicher-Partitionen **einheitlicher** Größe:



Feste Hauptspeicher-Partitionen **unterschiedlicher** Größe:



Feste Partitionierung: Probleme

- Ein Programm kann zu groß sein für die Partition
  - der Programmierer muss dann sein Programm aufteilen: Randnotiz **„Overlay“-Technik (*)**
- Der Hauptspeicher wird nicht effizient genutzt
  - jedes Programm belegt eine komplette Partition → „Verschnitt“ → ungenutzer freier Speicher **(„Interne Fragmentierung“)**

Feste Partitionierung: Probleme

Beispiel Fortran:
Hauptprogramm
...
 CALL OVERLAY (filename1, ...) *1
 CALL OVERLAY (filename2, ...) *2
 ...
 CALL OVERLAY (filename3, ...) *3
...
Ende Hauptprogramm

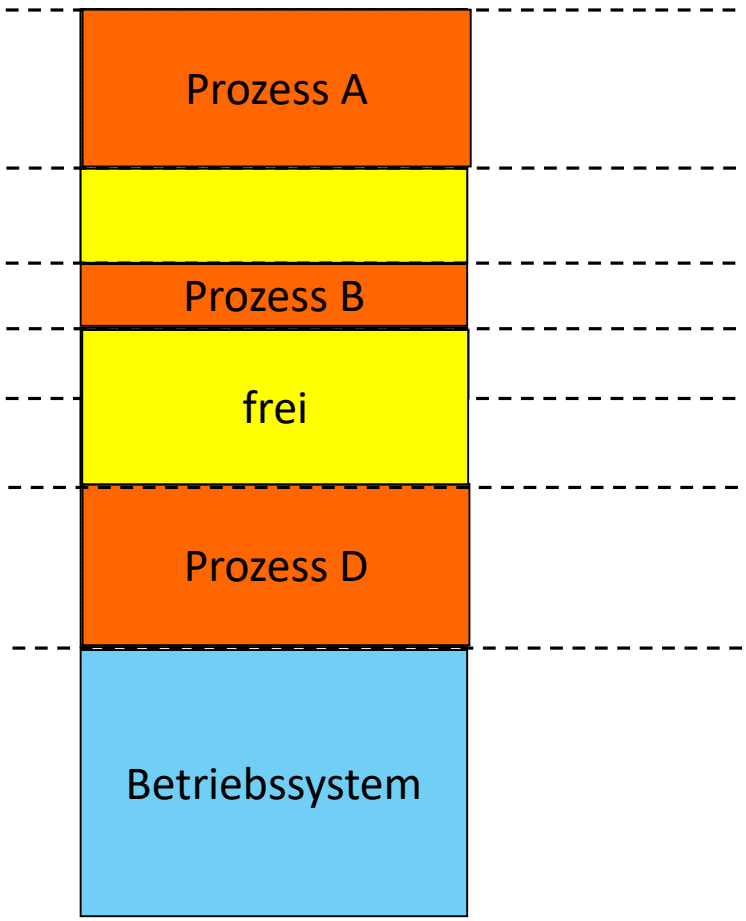
OVERLAY (filename1, …)
 Fortran-Code für das gesamte OVERLAY Pa
 END
OVERLAY (filename2, …)
 Fortran-Code für das gesamte OVERLAY Paket 1.1
 END
…

Beispiel Turbo Pascal:

```
program meinprojekt;
overlay procedure kundenverwaltung;
 ...
overlay procedure warenverwaltung;
 ...
 { Hauptprogramm }
begin
 while input <> "ende" do begin
   case input of
     "kunden": kundenverwaltung;
     "waren": warenverwaltung;
   end;
 end;
end.
```

Hauptspeicheraufteilung bei Multiprogramming:
Dynamische Partitionierung

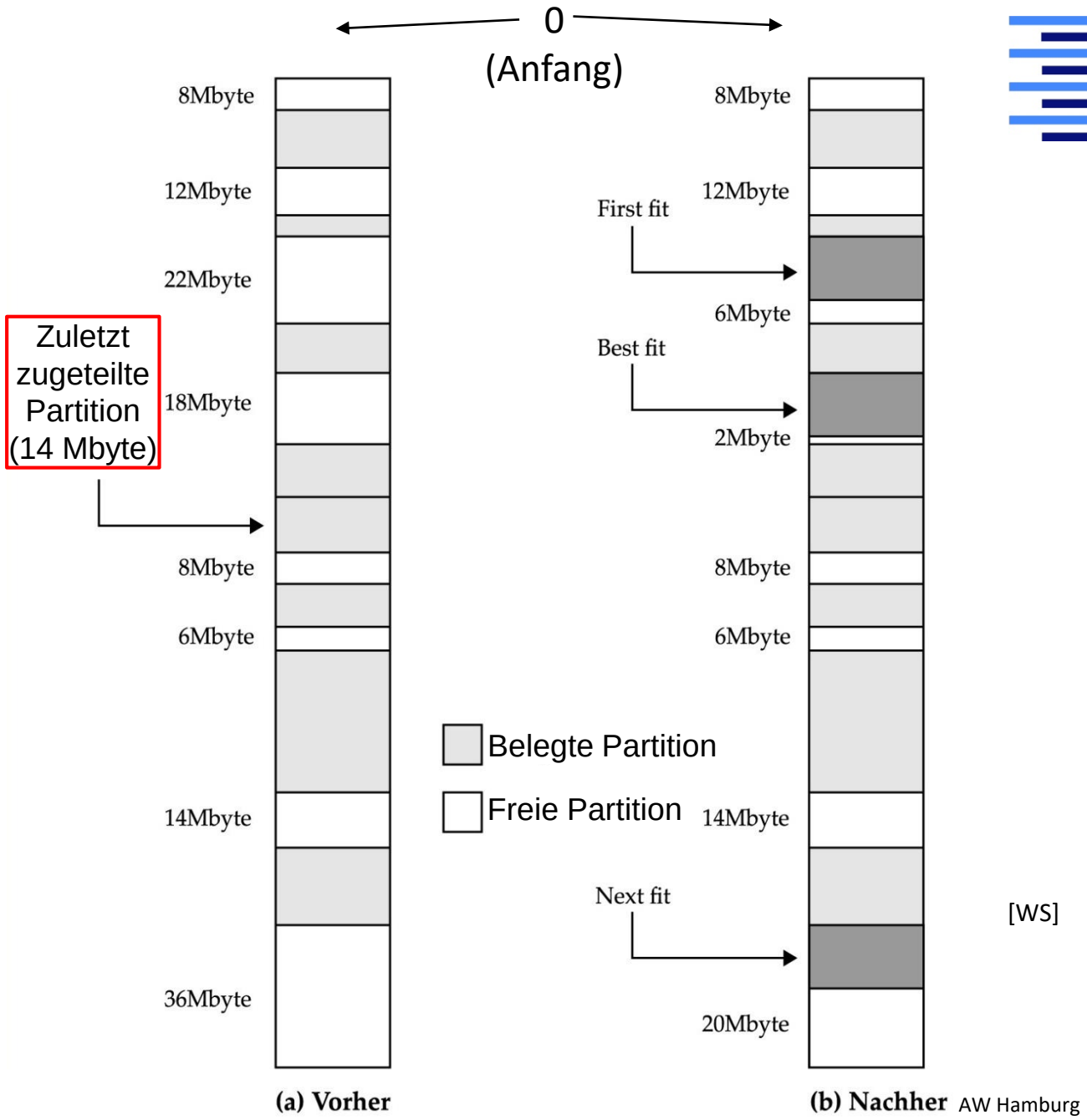
- Es gibt eine **variable** Anzahl von Partitionen
- Partitionen haben **unterschiedliche Größen**
- Die Partitionen werden an die Prozessgröße angepasst
  - Nach Zuweisung einer Partition zu einem Prozess wird der restliche freie Platz eine **neue Partition**
  - **Zusammenfassen** von freien Partitionen ist möglich



Dynamische Partitionierung: Platzierungsstrategien

- Das Betriebssystem muss entscheiden, welche freie Partition welchem Prozess zugewiesen wird
- Algorithmen:
  - **First-Fit**
    - Sucht von vorne die nächste freie Partition, die passt
  - **Next-Fit**
    - Sucht ab der zuletzt belegten Partition die nächste freie Partition, die passt
  - **Best-Fit**
    - Auswahl der freien Partition, bei der am wenigsten Platz verschwendet wird

Beispiel für Platzierungs-strategien: Platzierung eines 16 MByte großen Prozesses



[WS]

Dynamische Partitionierung: Bewertung der Platzierungsalgorithmen

- **Best-Fit:**
  - Jeweils bestes Ergebnis, aber Fragmentierung:
  - Weil immer kleine Speicherreste bleiben, muss das Betriebssystem am häufigsten umsortieren -> schlechtes Ergebnis!
  - Aufwendigste Suche!
- **First Fit:**
  - Schnelles Verfahren!
  - Viele Prozesse im vorderen Speicherbereich
  - Meist hinten noch Platz für große Prozesse
- **Next-Fit:**
  - Belegt Speicher gleichmäßiger als First Fit, nachteilig für große Prozesse
  - Die größte freie Partition wird eher verwendet (liegt meist hinten)
  - Umsortieren, um wieder Platz für große Prozesse zu erhalten, ist oft nötig
  - Leichter Nachteil gegenüber First Fit!

Hauptspeicheraufteilung bei Multiprogramming: Dynamische Partitionierung

Probleme

- Der Hauptspeicher wird immer noch nicht effizient genutzt
  - es entstehen „Löcher“ im Speicher durch kleine Partitionen („externe Fragmentierung“)
    - Abhilfe: Das Betriebssystem könnte die Partitionen regelmäßig umkopieren (ist aber sehr aufwändig)
- Ein Programm kann zu groß sein für den gesamten verfügbaren Hauptspeicher → **„Overlay“-Technik** nötig

Aufgaben Abschnitt 4.1.c): Einfache Zuweisungs- und Freigabeverfahren

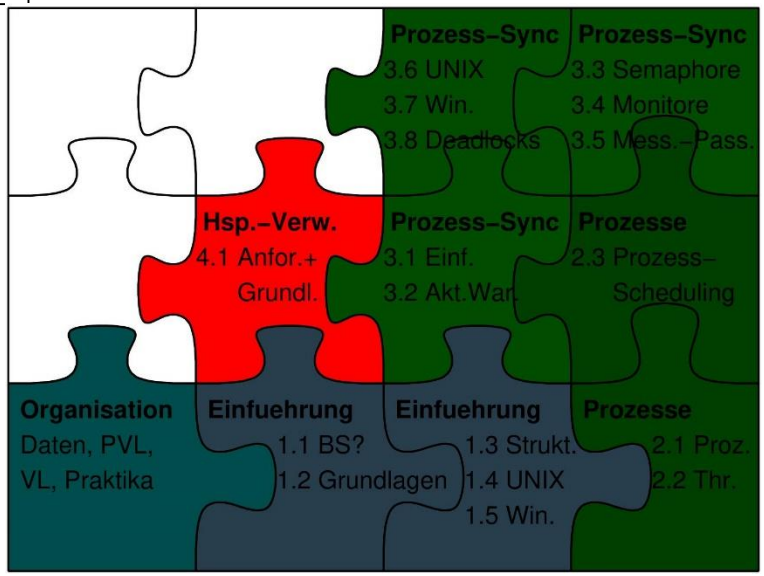
Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 2

2. Ein Swapping-System entfernt Lücken aus dem Speicher durch Verdichtung. Wenn viele Lücken und Segmente zufällig über den Speicher verteilt sind und das Lesen oder Schreiben eines 32-Bit-Wortes 10 ns dauert, wie lange dauert es dann ungefähr, 128 MB Speicher zu verdichten? Der Einfachheit halber nehmen wir an, dass das Wort 0 in einer Lücke liegt und dass das letzte Wort im Speicher Daten enthält.

Anmerkung: Gehen Sie davon aus, dass die Programme stumpf zusammengeschoben werden. Gehen Sie weiterhin davon aus, 70% des Speicher belegt sind.

Kapitel 4
Hauptspeicher-Verwaltung

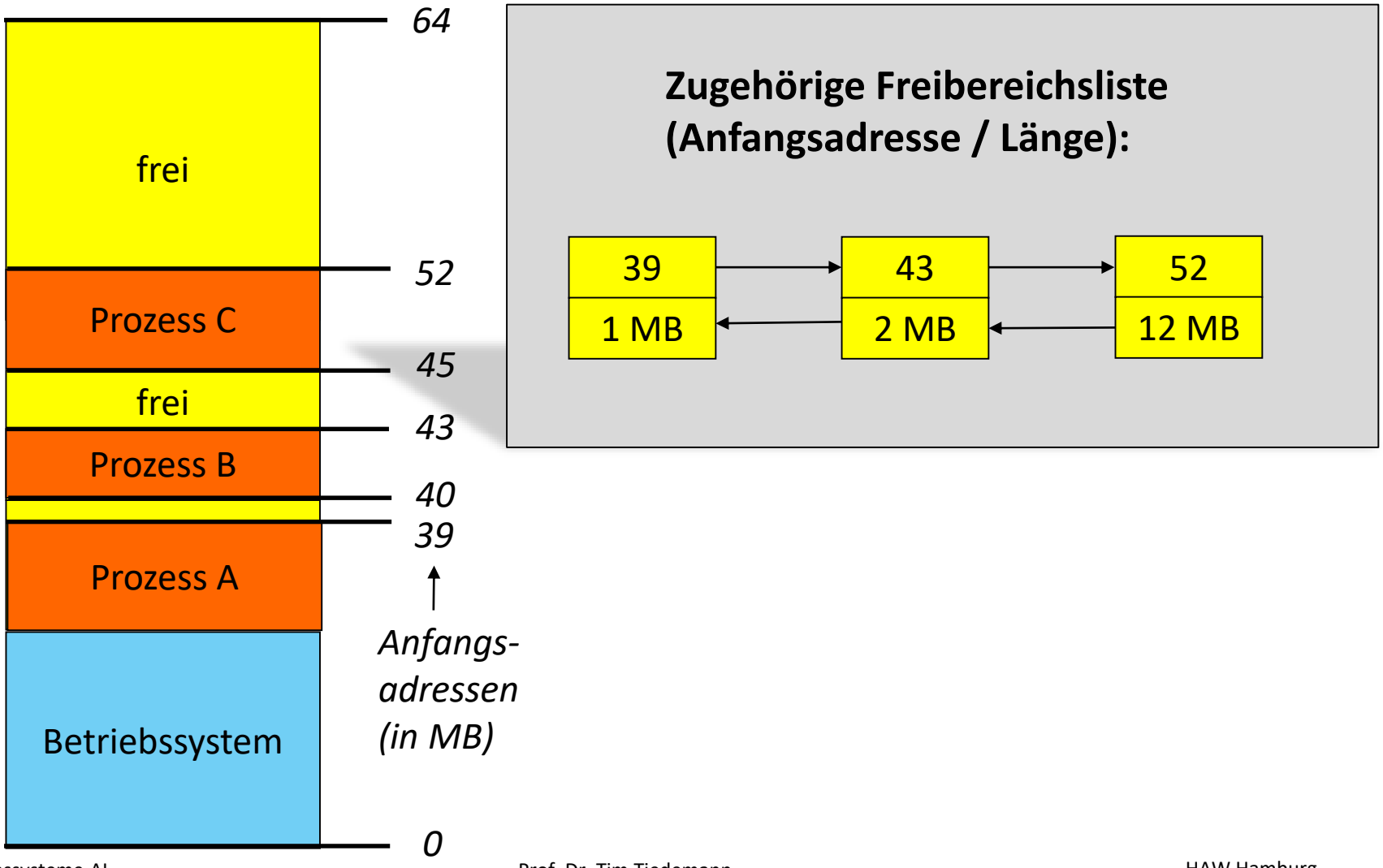
1. Grundlagen
   a) Anforderungen
   b) Adressberechnung
   c) Einfache Zuweisungs- und Freigabeverfahren
   d) Implementierungsaspekte
2. Virtueller Speicher
   a) Einführung und Prinzipien
   b) Paging
   c) Pagingstrategien
   d) Unix / Windows
   e) Meltdown / Spectre



Implementierung der Speicherverwaltung

- **Speicher(belegungs)verwaltung mit Bitmaps**
  - Jeder Speichereinheit (z.B. Partition) entspricht ein Bit in einem Speicherwort („Bitmap“). Dessen Wert signalisiert, ob die Einheit frei ist.
  - Problem: Die Suche nach mehreren zusammenhängenden Einheiten ist relativ aufwändig (Wortgrenzen!)
- **Speicherverwaltung mit verketteten Listen („Freibereichslisten“)**
  - In einer doppelt verketteten Liste spezifiziert jedes Element einen freien Bereich im Hauptspeicher durch Angabe von realer Anfangsadresse und Länge

Beispiel für eine Freibereichsliste



Auslagerung von Prozessen

- Falls nicht **alle** Prozesse im Hauptspeicher gehalten werden können: Auslagerung von Prozessen auf Platte (ggf. temporär)!
- Ansätze:
  - Ein- und Auslagern **kompletter** Prozesse: **Swapping**
  - Dynamische Ein- und Auslagerung von Programm**teilen**:
    - Laden von Bibliotheksprozeduren auf Anforderung: **„dynamic loading“**
    - Ein- und Auslagerung einzelner vom Programmierer manuell aufgeteilter Programmstücke: **„Overlays“**
    - Automatische Aufteilung von Prozessen und dynamisches Ein- und Auslagern einzelner Prozessteile auf Anforderung: **„Virtueller Speicher“**

Aufgaben Abschnitt 4.1.d): Implementierungsaspekte

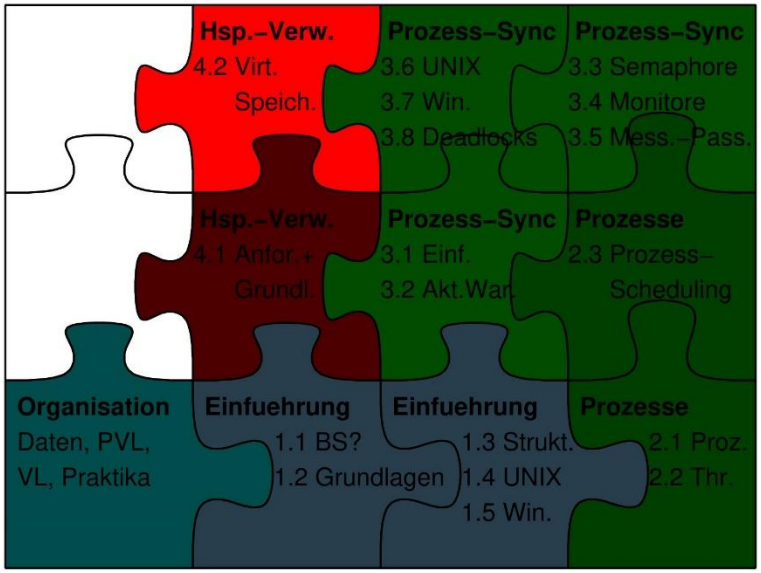
Quelle: Tanenbaum, Moderne Betriebssysteme, 3. Auflage, Kapitel 3, Aufgabe 3

3. In dieser Aufgabe vergleichen wir die Speicherkosten für die Verwaltung von freiem Speicher mit einer Bitmap und mit verketteten Listen. Der Speicher ist 128 MB groß und wird in Einheiten von n Byte zugeteilt. Für die verkettete Liste setzen wir voraus, dass der Speicher abwechselnd aus 64 MB großen Lücken und Segmenten besteht. Außerdem nehmen wir an, dass jeder Knoten in der Liste 32 Bit für die Adresse, 16 Bit für die Länge und 16 Bit für den Zeiger auf den nächsten Knoten benötigt. Wie viel Speicher ist für jede der beiden Methoden nötig? Welche ist besser?

Anmerkung: Fehler im Text: Gehen Sie davon aus, dass die Lücken und Segmente 64 KB (nicht 64 MB) groß sind.

Kapitel 4
Hauptspeicher-Verwaltung

1. Anforderungen und Grundlagen
2. Virtueller Speicher
   a) Einführung und Prinzipien
   b) Paging
   c) Pagingstrategien
   d) Unix / Windows
   e) Meltdown / Spectre



Anforderungen an einen idealen Speicher aus Programmierersicht

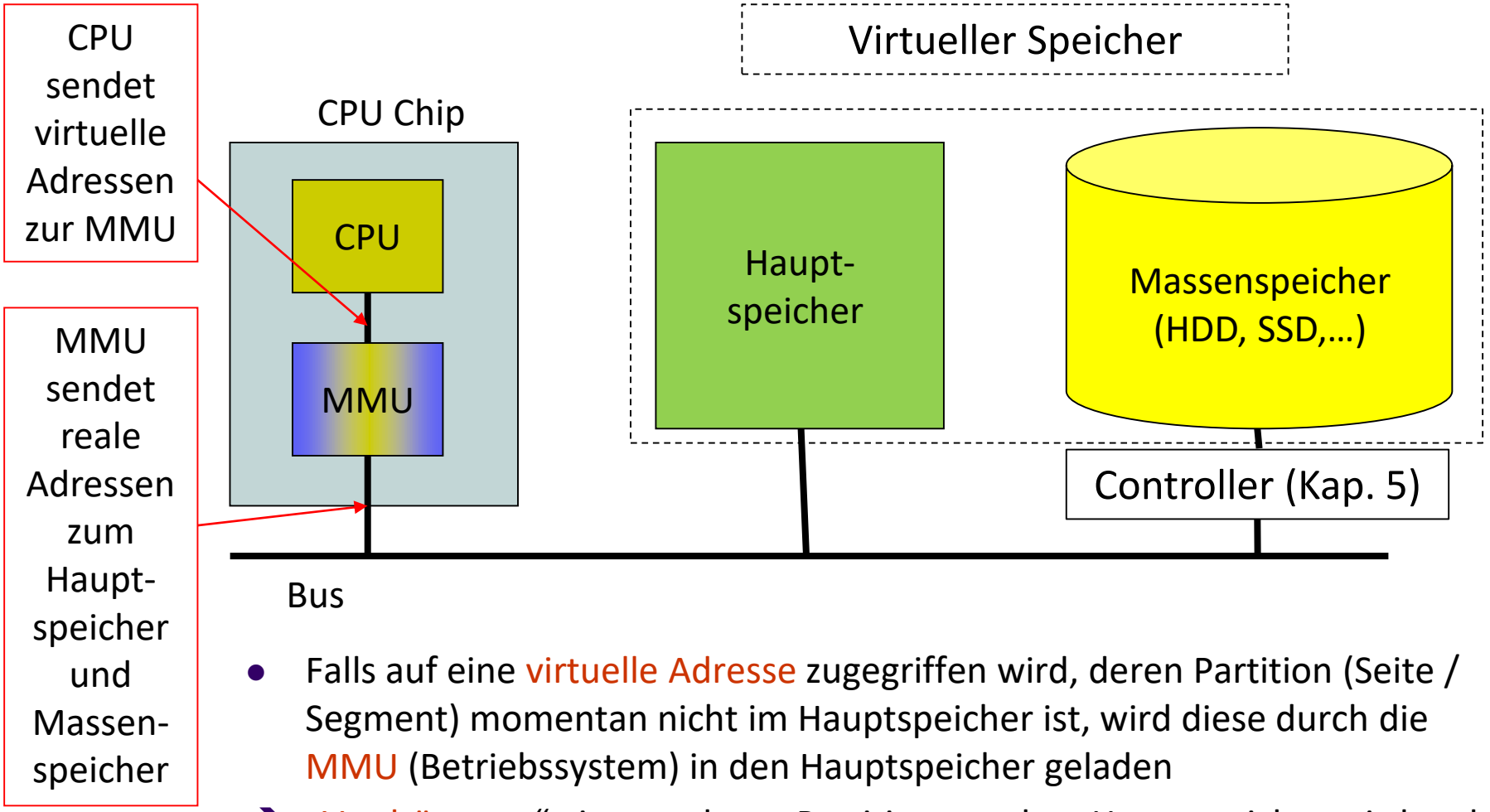
- **Unbeschränkte Größe**, so dass jedes beliebig große Programm ohne zusätzlichen Aufwand geladen und verarbeitet werden kann
- **Einheitliches Adressierungsschema** für **alle** Speicherzugriffe (keine Unterscheidung von Speichermedien)
- **Direkter Zugriff** auf den Speicher (ohne Zwischentransporte)
- **Schutz vor fremden Zugriffen** in den eigenen Speicherbereich

➔ **Ziel: „Virtueller Speicher“ mit idealen Eigenschaften!**

Realisierung eines virtuellen Speichers

- Aufteilung des Hauptspeichers in viele (kleine) Partitionen
  - feste Größe: **Seiten** (→ Paging)
  - unterschiedliche Größe: **Segmente (*)**
- **Jeder Prozess wird auf mehrere Partitionen verteilt!**
- Partitionen können einzeln auf die Platte ausgelagert werden
- Jeder Prozess benutzt eigene logische („virtuelle“) Adressen
- Jeder virtuelle Adressraum kann größer als der physikalische Hauptspeicher sein (nur durch Plattenkapazität beschränkt)

Abbildung der virtuellen Adressen auf reale Adressen durch die MMU

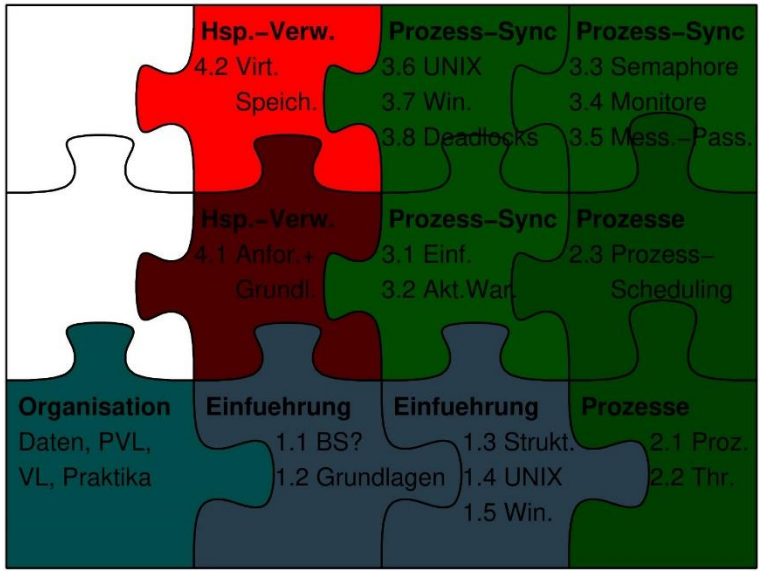


- Falls auf eine virtuelle Adresse zugegriffen wird, deren Partition (Seite / Segment) momentan nicht im Hauptspeicher ist, wird diese durch die MMU (Betriebssystem) in den Hauptspeicher geladen

➔ „Verdrängung“ einer anderen Partition aus dem Hauptspeicher wird evtl. notwendig

Kapitel 4
Hauptspeicher-Verwaltung

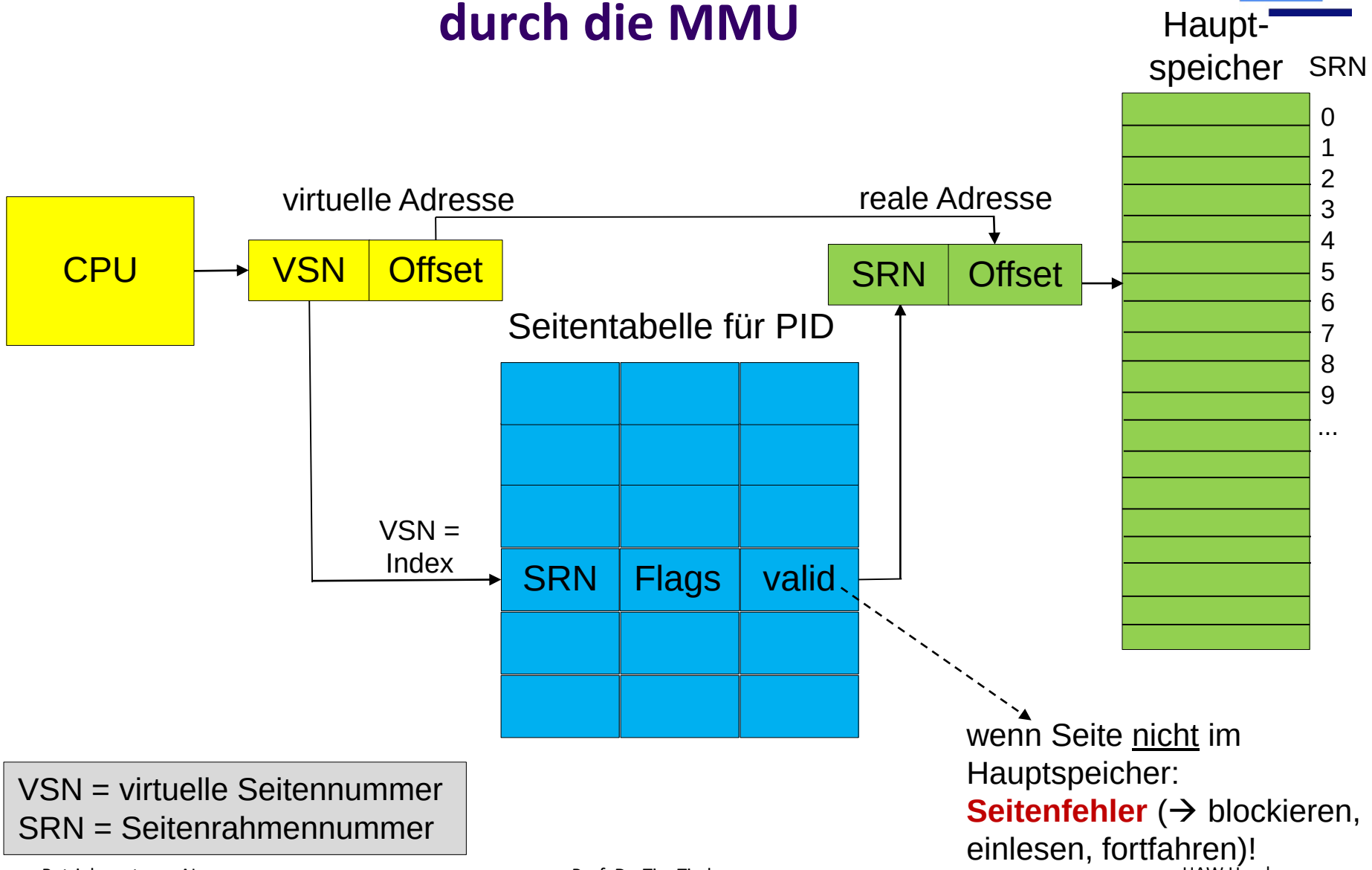
1. Anforderungen und Grundlagen
2. Virtueller Speicher
   a) Einführung und Prinzipien
   b) Paging
   c) Pagingstrategien
   d) Unix / Windows
   e) Meltdown / Spectre



Grundprinzip des Paging

- Linearer Adressraum mit virtuellen Adressen
  - Virtuelle Adressen …
    - … beginnen für *jeden* Prozess bei 0
    - … werden fortlaufend durchnummeriert
    - … täuschen einen virtuellen Speicher vor
- Aufteilung des virtuellen Adressraums in Seiten
  - Seite = Partition fester Größe *(„Page“)*
  - Jeder Seite wird ein zusammenhängender realer Speicherbereich zugeordnet, auch *Seitenrahmen („Page Frame“)* genannt
  - Ein Seitenrahmen kann im Hauptspeicher oder auf der Platte liegen, muss bei Zugriff aber in den Hauptspeicher geladen werden
- Abbildung von virtuellen auf reale Adressen (virtuelle Seiten → Seitenrahmen) durch eine Seitentabelle

Ermittlung einer realen Adresse mittels Seitentabelle durch die MMU



VSN = virtuelle Seitennummer
SRN = Seitenrahmennummer