Betriebssysteme

Vorlesung im Herbstsemester 2008
Universität Mannheim

Kapitel 4: Adressräume

Prof. Dr. Felix C. Freiling
Lehrstuhl für Praktische Informatik 1
Universität Mannheim

Motivation

- Der Adressraum ist die Abstraktion des physikalischen Speichers
  - Einer der unverzichtbaren Basisdienste der Systemsoftware
  - Zusammenhängende Menge von Adressen plus deren Inhalte
  - Maximale Größe definiert durch die Breite des Adressbusses
    - Heute gebräuchlich: 32 Bit Adressen = 4 Gbyte großer Adressraum
- In diesem Kapitel geht es um die Realisierung dieses Basisdienstes
  - Welche Hardwareunterstützung für virtuellen Adressraum gibt es?
  - Wie geschieht die Adressumsetzung?
  - Wie werden Adressräume voneinander isoliert?
  - Welche Ansätze zur Implementierung virtueller Adressräume gibt es?

Positionsbestimmung

- Gliederung der Vorlesung:
  1. Einführung und Formalia
  2. Auf was baut die Systemsoftware auf?
     Hardware-Grundlagen
  3. Was wollen wir eigentlich haben?
     Laufzeitunterstützung aus Anwendersicht
  4. **Verwaltung von Speicher: Virtueller Speicher**
     - **Drei Wochen!**
  5. Verwaltung von Rechenzeit: Virtuelle Prozessoren (Threads)
  6. Synchronisation paralleler Aktivitäten auf dem Rechner
  7. Implementierungsaspekte

Übersicht

- Adressumsetzung (Einführung)
- Organisation von Adressräumen aus Anwendungssicht
- Seitenbasierter virtueller Adressraum
- Dynamische Seitenersetzung
- Segmentbasierter virtueller Adressraum
- Implementierungsaspekte
- Physischer (nicht-virtueller) Adressraum

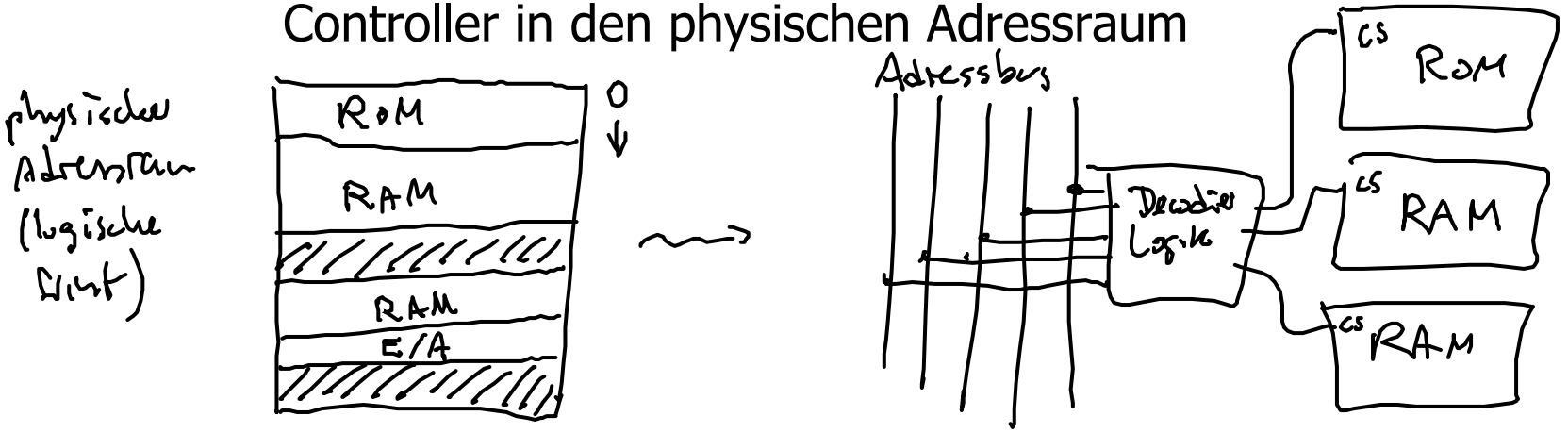
Adressumsetzung (Einführung)

Definition Adressraum

- Adressraum = zusammenhängende Menge von Adressen und deren Inhalte
  - Maximale Größe abhängig von der Prozessorarchitektur
  - 32 Bit ergibt 4 Gbyte (heute manchmal auch 64 Bit)
  - Adressen sind dann 32-Bit-Werte zwischen `0x00000000` und `0xffffffff`
  - Inhalt einer Adresse ist entweder
    - undefiniert
    - n-stelliger Binärwert (meist 1 Byte)
- Manchmal ist nicht der volle Adressraum mit physischem Speicher hinterlegt
  - Nichtreferenzierbare Lücken, Zugriff erzeugt einen Laufzeitfehler
- Zugriffskontext definiert Typ des Adressinhalts
  - In der Instruktionsladephase: binärcodierte Maschinenbefehle
  - In der Operandenladephase: Daten

Adressumsetzung

- Grundlage für jeden Adressraum ist der physische Adressraum des Rechners
  - Zuordnung geschieht durch eine direkte Abbildung der 16, 32 oder 64 Bit breiten Adressen des Prozessors auf die vorhandenen Speicherbausteine und E/A-Controller
- Durchführung in Hardware: Hardwareadressdecoder
  - Umsetzung von Signalen auf dem Adressbus auf die Freigabeleitungen der Hardwarebausteine
  - Entspricht dem "Einblenden" der RAM/ROM-Bausteine sowie der E/A-Controller in den physischen Adressraum

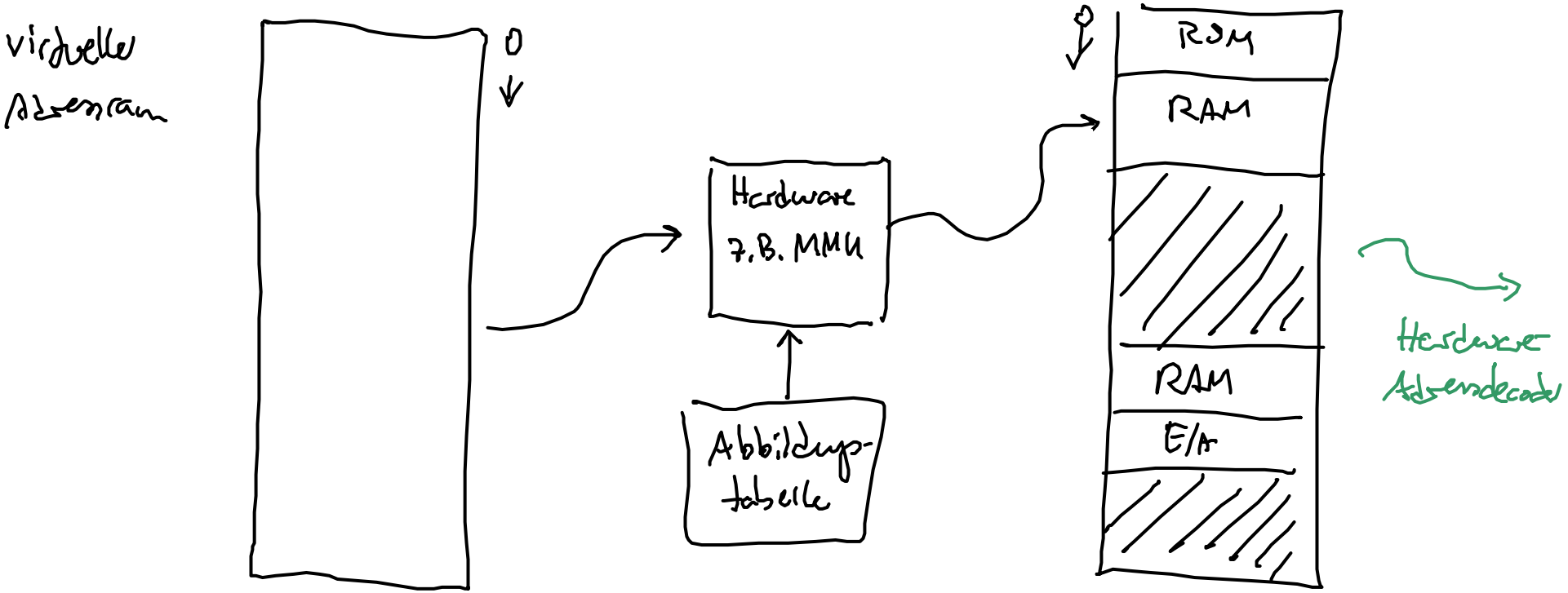


Physischer Adressraum

- Unmittelbare Benutzung des physischen Adressraums erfordert detaillierte Kenntnisse über Struktur und Zusammensetzung
  - Nur empfehlenswert bei kleinen Systemen mit überschaubarer Komplexität
    - Industrielle Steuerungen, eingebettete Systeme
  - Sonderfalls: ältere Betriebssysteme wie MS-DOS, die ausschließlich den physischen Adressraum bereitstellen
- Allgemein sinnvoller: Bereitstellung eines bereinigten virtuellen Adressraums
  - Verbirgt technische Details
  - Hebt vorhandene Beschränkungen auf
- Erreichbar durch zweiten vorgelagerten Abbildungsschritt

Vorgelagerte Abbildung

- Abbildung eines logischen/virtuellen Adressraums in den physischen Adressraum
  - Erfordert zusätzlichen Hardware/Software-Aufwand
  - Aufwand rechnet sich durch die Vereinfachung bei der Nutzung des Adressraums



Erläuterungen

- Programmausführung findet im virtuellen Adressraum statt
  - Prozessor referenziert logische Adressen
  - Hardware (MMU = Memory Management Unit) setzt logische Adressen um
  - Basis der Abbildung: veränderbare Abbildungsvorschrift im physischen Speicher!
- Vorteil: Programmgesteuerte Veränderbarkeit der Abbildungsvorschrift
  - Software hat umfassende Kontrolle und Flexibilität bei der Zuteilung des physischen Speichers

Übersicht

- Adressumsetzung (Einführung)
- Organisation von Adressräumen aus Anwendungssicht
  - Wie soll der virtuelle Adressraum organisiert sein?
- Seitenbasierter virtueller Adressraum
- Dynamische Seitenersetzung
- Segmentbasierter virtueller Adressraum
- Implementierungsaspekte
- Physischer (nicht-virtueller) Adressraum

Organisation des Adressraums aus Anwendersicht

Arten von Daten

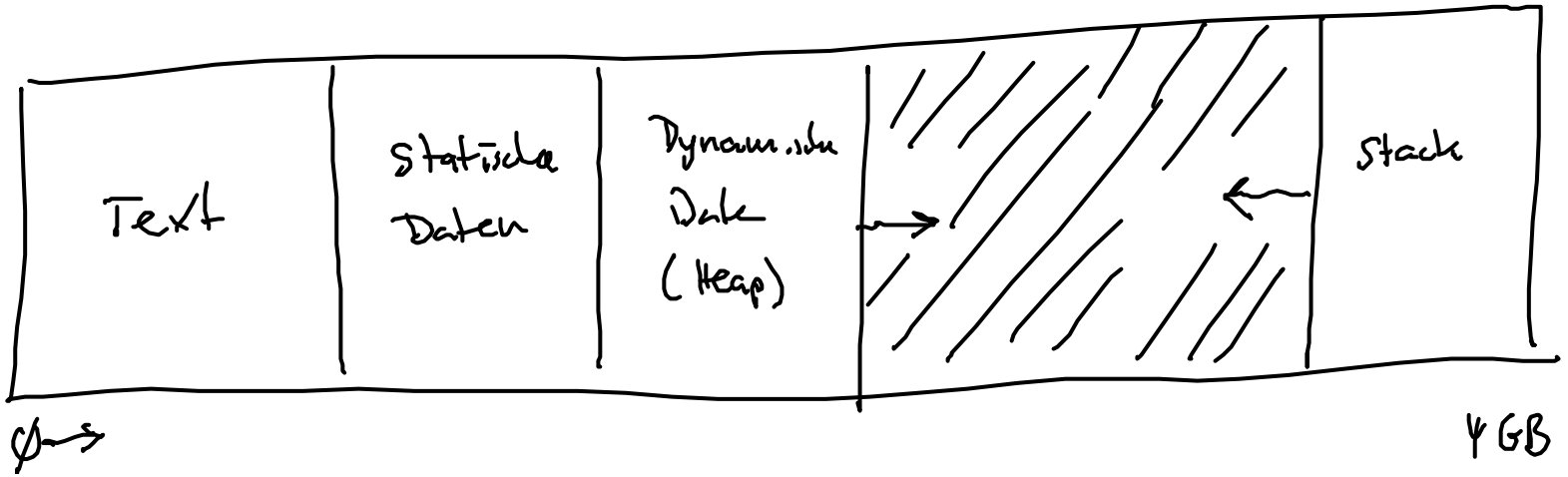
- Im Adressraum müssen alle für die Programmausführung notwendigen Daten zur Verfügung gestellt werden
- Man unterscheidet drei Bereiche:
  - Programmcode (“Text”)
    - Enthält die für die Programmausführung notwendigen Maschineninstruktionen
  - Datenbereich (“Daten”)
    - Speichert die Variablen (der wesentliche Teil des Programmzustandes)
  - Laufzeitkeller (“Stack”)
    - Zur Verwaltung von Unterprogrammaufrufen

Statische und dynamische Daten

- Der Datenbereich wird noch weiter unterteilt:
  - Statischer Datenbereich
    - enthält alle zur Übersetzungszeit bekannten und ggf. vorinitialisierten Datenstrukturen
  - Dynamischer Datenbereich (Heap)
    - enthält alle zur Laufzeit erzeugten Datenstrukturen
    - Verantwortliche Funktionen: malloc (in C), new (in C++ oder Java)
- Meist gibt es auch noch dynamische Daten auf dem Stack
  - Lokale Variablen von Unterprozeduren
    - Gut für rekursive Prozeduren
  - Parameter für Unterprozeduren

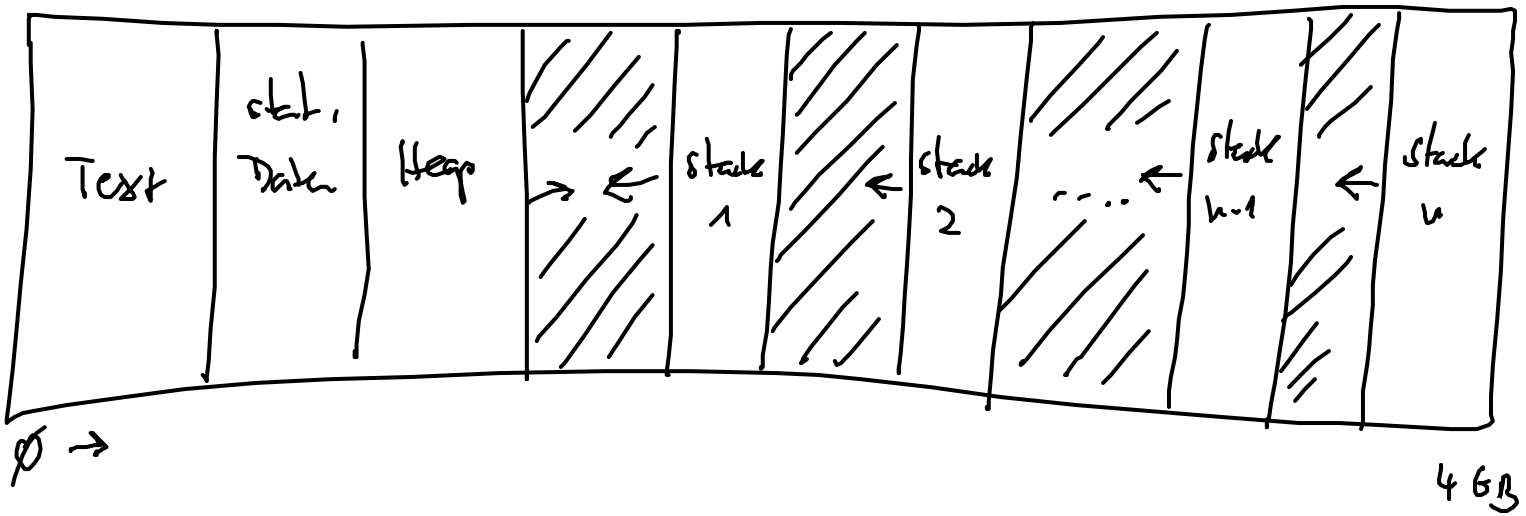
Plazierung im Adressraum

- Die Adressraumverwaltung muß Text, Daten, Stack und Heap im virtuellen Adressraum anordnen
  - Eventuelles Größenwachstum miteinkalkulieren
- Typische Verteilung:
  - Programm, statische Daten, dynamische Daten, großer Abstand, Stack (wächst nach unten)



Mehrere Stacks

- Bei mehreren Threads im selben Adressraum benötigt man auch mehrere Stacks
  - Stackbereich wird dann typischerweise unterteilt (ein Stack pro Thread)
  - Abstand möglichst groß, erhöht trotzdem die Gefahr von "Kollisionen"



Überschneidungsgefahr

- Überschneidung (Kollision) zwischen einzelnen Stacks und/oder dem Heap muß vermieden werden!
- Normalerweise kein Problem. Beispiel:
  - 20 Mbyte großes Programm (Text und Daten)
  - 4 Gbyte Adressraum = 4096 Mbyte
  - Lücke ist anfangs 4076 Mbyte groß
- Wahrscheinlichkeit der Überschneidung erhöht sich überproportional bei mehreren Laufzeitkellern
  - Überschneidung erzeugen normalerweise schwer zu lokalisierende Laufzeitfehler
  - Schutz vor Überschneidung benötigt Hardwareunterstützung

Größe des Adressraums

- Ohne Zusatztechniken kann ein logischer Adressraum nie größer sein als der physische Adressraum
  - Physischer Adressraum relativ teuer, dadurch nicht immer volle 4 Gbyte bestückt
- Idee: Adressraum um (relativ billigen) Externspeicher erweitern
  - Meist Festplatten, langsam aber billig
  - Umsetzung über eine zusätzliche Tabelle
  - Bei ausgelagertem Speicherinhalt: Unterbrechung der aktuellen Anwendung und Einlagern der Daten durch die Systemsoftware
  - Anschließend: Fortsetzung der unterbrochenen Anwendung

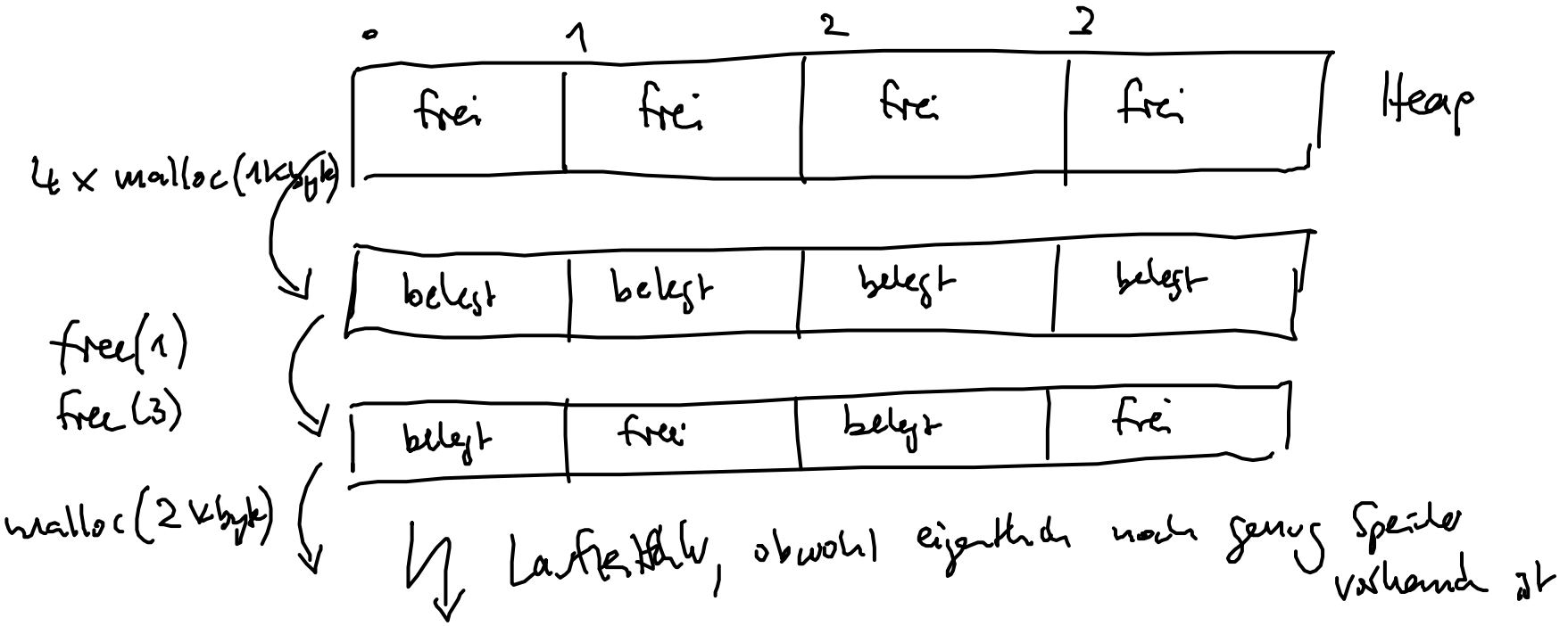
RAM ist Cache für billige aber langsame Externspeicher

Adressraumfragmentierung

- Fragmentierung = Zerstückelung des noch freien, nutzbaren Teiles des Adressraums in kleine Bereiche
  - Problem: Wenn immer einzelne zusammenhängende Speicherbereiche angefordert werden, kann der Fall auftreten, dass eine neue Anfrage nicht befriedigt werden kann, obwohl in der Summe noch genügend freier Speicher bereitsteht
- Unterscheidung zwischen externer und interner Fragmentierung
  - Externe Fragmentierung im dynamischen Datenbereich (Heap)
  - Interne Fragmentierung bei der Anforderung von Bereichen fester Größe (z.B. bei Festplatten)

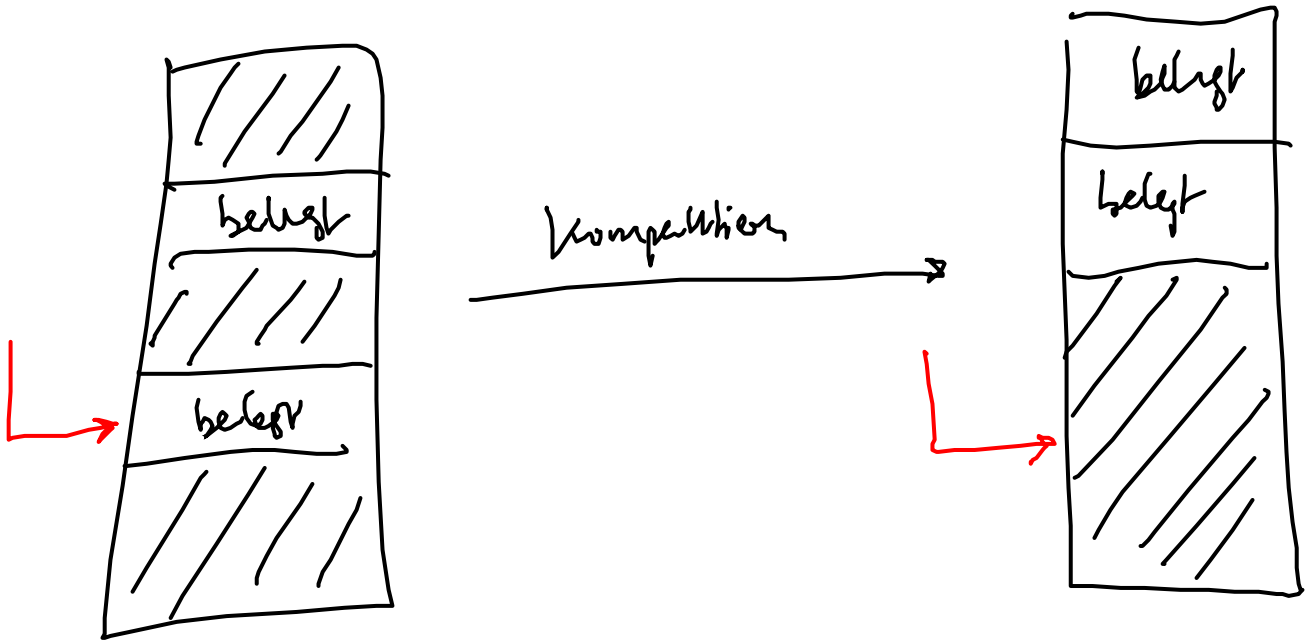
Beispiel: externe Fragmentierung

- Im 4 Kbyte großen Heap werden vier 1-Kbyte Blöcke angefordert, von denen anschließend der erste und dritte wieder freigegeben werden
- Anschließend erfolgt die Anforderung von 2 Kbyte



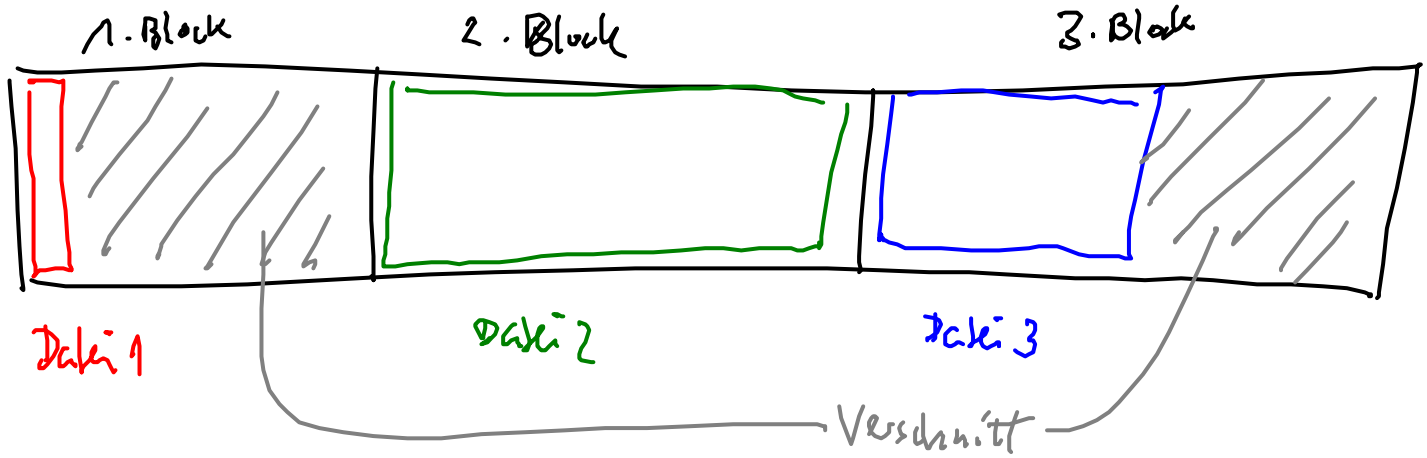
Mögliche Abhilfe und Probleme

- Zusammenschieben der Speicherbereiche
  - relativ zeitäufwändig
- Problem: Verweise "von außen" auf diese Speicherbereiche



Interne Fragmentierung

- Interne Fragmentierung tritt immer dann auf, wenn Speicherbereiche nur in fester Größe zugeteilt werden
- Beispiel: Festplatten teilen in der Regel mindestens einen Block zu
  - Blockgröße zwischen 1 und 8 Kbyte
  - Die Anforderung von Speicher für ein einzelnes Byte verbraucht immer einen ganzen Block; entstehender Verschnitt bleibt unbenutzt



Schutz

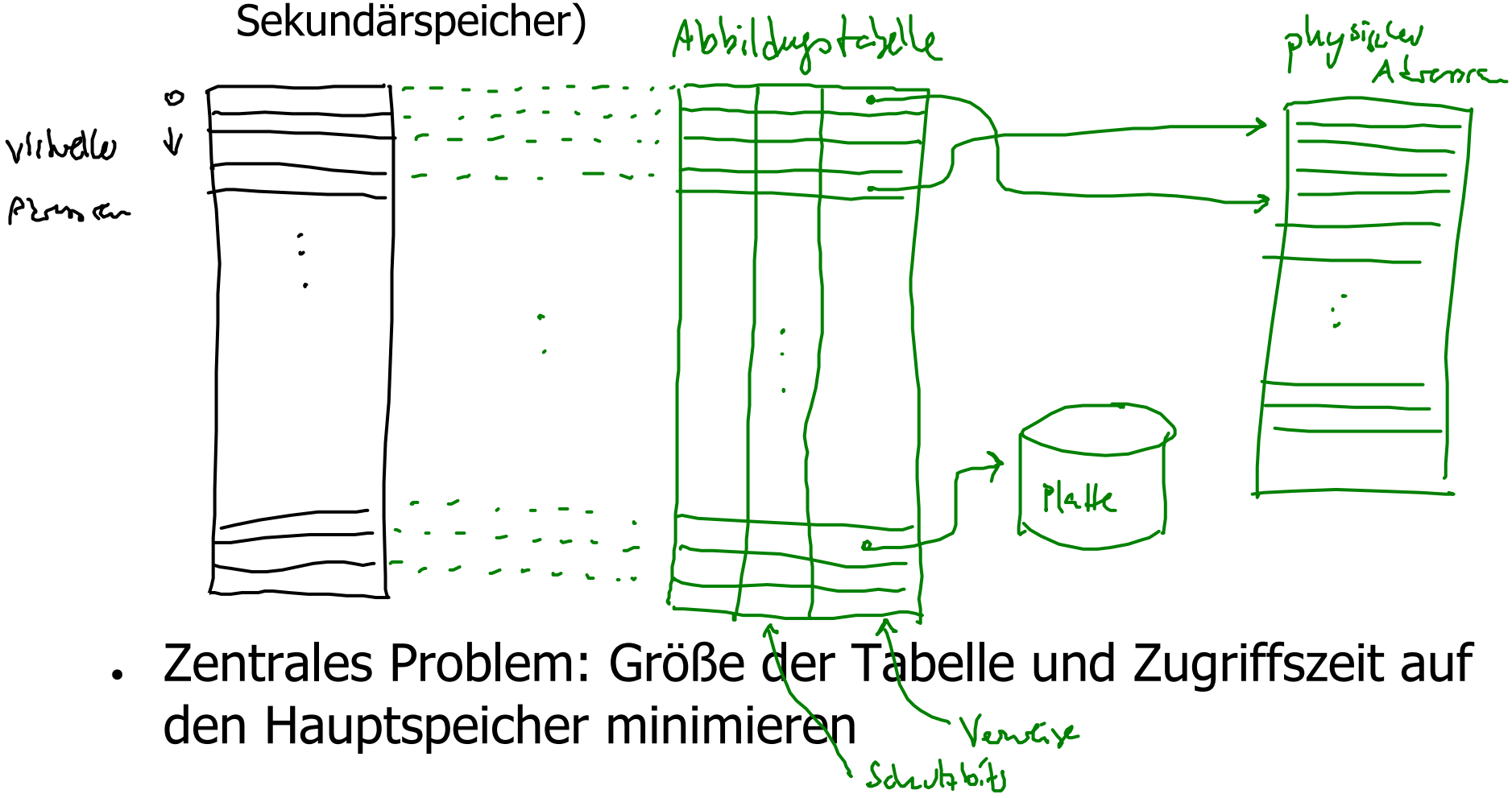
- Adressraumbereiche unterscheiden sich in den Arten des Zugriffs:
  - Auf den Text wird normalerweise nur lesend zugegriffen
    - Schutz des Text-Bereichs vor schreibendem Zugriff ist erstrebenswert
  - Daten werden normalerweise nicht ausgeführt
    - Schutz vor verschiedenen Arten von Malware
- Technisch benötigt man zusätzliche Informationen pro Speicherzelle, um verschiedene Zugriffsmodi festzulegen
  - Modi etwa: Kein Zugriff, lesend, schreibend, ausführend
  - Zusatzbits in Umsetzungstabelle dürfen nur kontrolliert verändert werden können
  - Versuch der unerlaubten Veränderung führt zu einer Unterbrechnung
  - Im Vordergrund steht Schutz vor fehlerhaften Anwendungen

Anwendungsanforderungen

- Die wichtigsten Forderungen aus Sicht der Anwendungsprogrammierung zusammengefasst:
  - Adressraum sollte homogen und zusammenhängend sein
  - Größe des genutzten Adressraums sollte unabhängig vom physischen Speicherausbau des Rechners sein
  - Es sollte verschiedene Zugriffsarten geben und ein entsprechenden Zugriffsschutz
  - Überschneidungen von Heap und Stack sollten erkannt und möglichst vermieden werden
- Bei Verwaltung mehrere Adressräume zusätzlich:
  - Schutz der Adressräume vor anderen Anwendungen
  - Faire Aufteilung der Speicherressourcen auf die Adressräume
  - Speicherökonomie, insbesondere minimale Fragmentierung

Realisierung der Forderungen

- Zentrales Konstrukt: Speicherabbildungstabelle
  - Enthält Zugriffsmodi und Ort der Daten (undefiniert, Hauptspeicher, Sekundärspeicher)



- Zentrales Problem: Größe der Tabelle und Zugriffszeit auf den Hauptspeicher minimieren

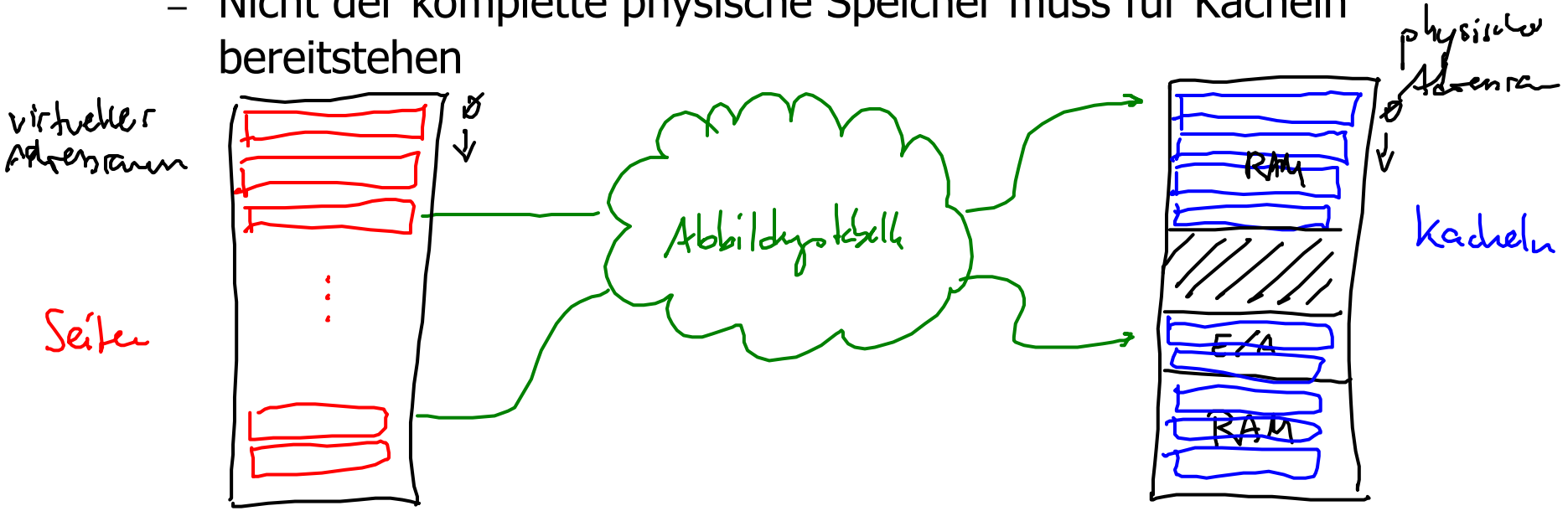
Übersicht

- Adressumsetzung (Einführung)
- Organisation von Adressräumen aus Anwendungssicht
- Seitenbasierter virtueller Adressraum
  - Minimierung externer Fragmentierung durch Verwendung von Speicherseiten gleicher Größe
- Dynamische Seitenersetzung
- Segmentbasierter virtueller Adressraum
- Implementierungsaspekte
- Physischer (nicht-virtueller) Adressraum

Seitenbasierter Virtueller Speicher

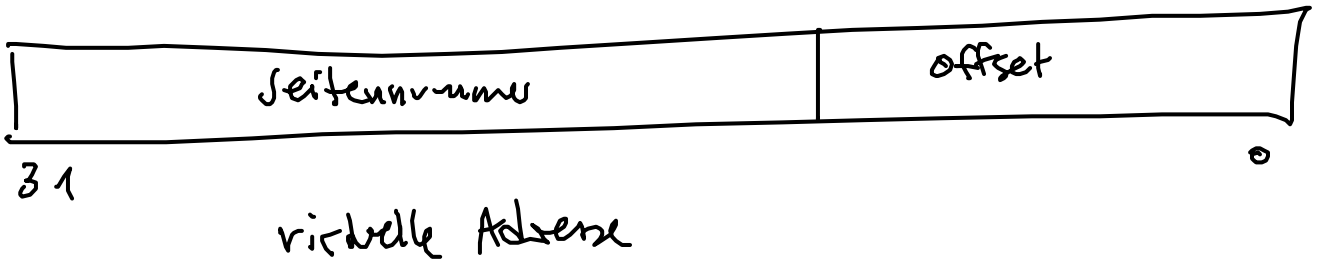
Seiten und Kacheln

- Seitenbasierte Verfahren unterteilen den virtuellen Adressraum in aufeinanderfolgende Seiten gleicher Größe
- Der physische Speicher wird in Kacheln der Größe einer Seite eingeteilt
  - Die Zuordnung von Seiten und Kacheln ist dynamisch und wird durch eine Tabelle realisiert
  - Nicht der komplette physische Speicher muss für Kacheln bereitstehen



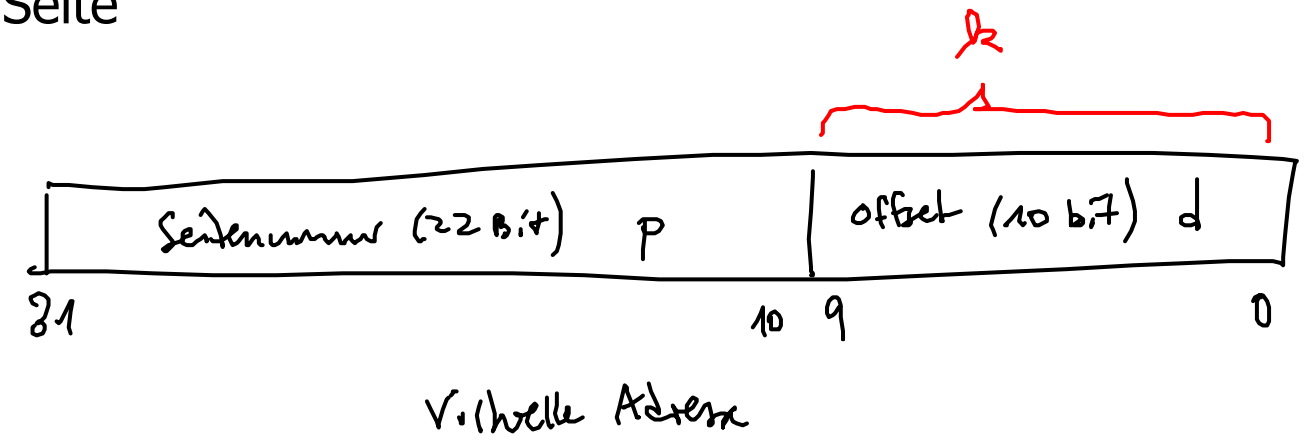
Hardwareunterstützung

- Seitenbasierte Verfahren kommen ohne spezielle Register aus
  - Im Gegensatz zu segmentbasierten Verfahren (siehe später) gibt es
    - keine besonderen Adressierungsarten
    - keine besonderen Register im Prozessor
  - Man benötigt lediglich eine nachgeschaltete MMU, die die Adressumsetzung vornimmt
- Eine Adresse wird immer als 2-Tupel aufgefaßt
  - Seitenanteil (welche Seite im virtuellen Adressraum ist gemeint?)
  - Offset (welche Adresse in der Seite ist gemeint?)



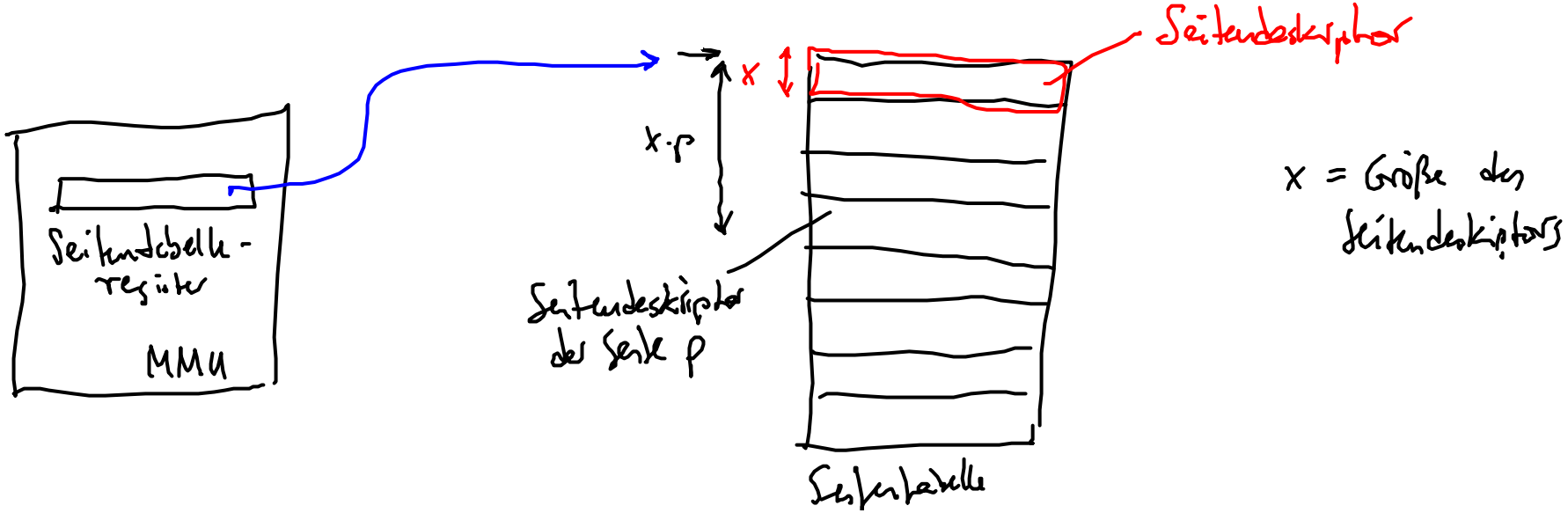
Größe einer Seite

- Die Größe einer Seite ist immer eine Zweierpotenz
  - Bei Seitengröße 2^k benötigt man k Bits zur Adressierung jeder Speicherzelle in der Seite (Offset)
  - Beispiel: k = 10, Seitengröße 1 Kbyte
- Die MMU interpretiert die niederwertigsten k Bits auf dem Adressbus als Offset, die höherwertigen Bits legen die Seite fest
  - Bei 1 Kbyte Seitengröße und 32 Bit Adressbus bleiben 22 Bit für die Seite



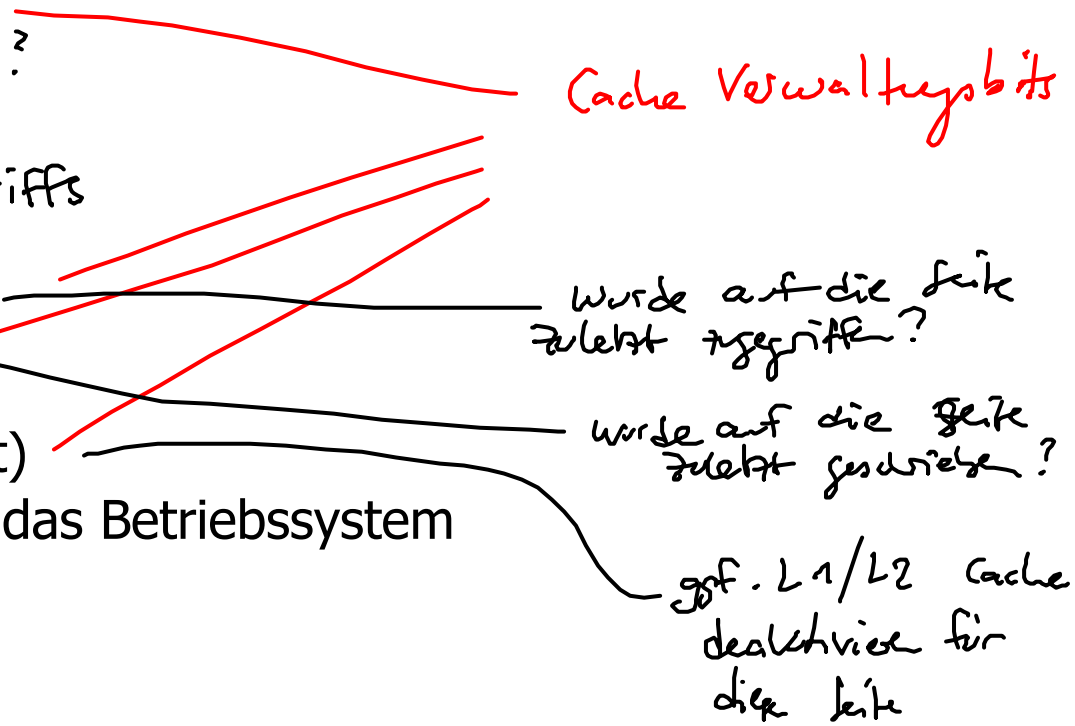
Seitentabelle

- Pro Seite des virtuellen Adressraums gibt es einen Seitendeskriptor
  - Seitendeskriptoren werden in einer großen Tabelle gespeichert (Seitentabelle)
  - Anfang der gerade gültigen Seitentabelle wird in einem speziellen Register der MMU gehalten
  - Die Adresse des Seitendeskriptors der Seite *p* kann man dann errechnen:



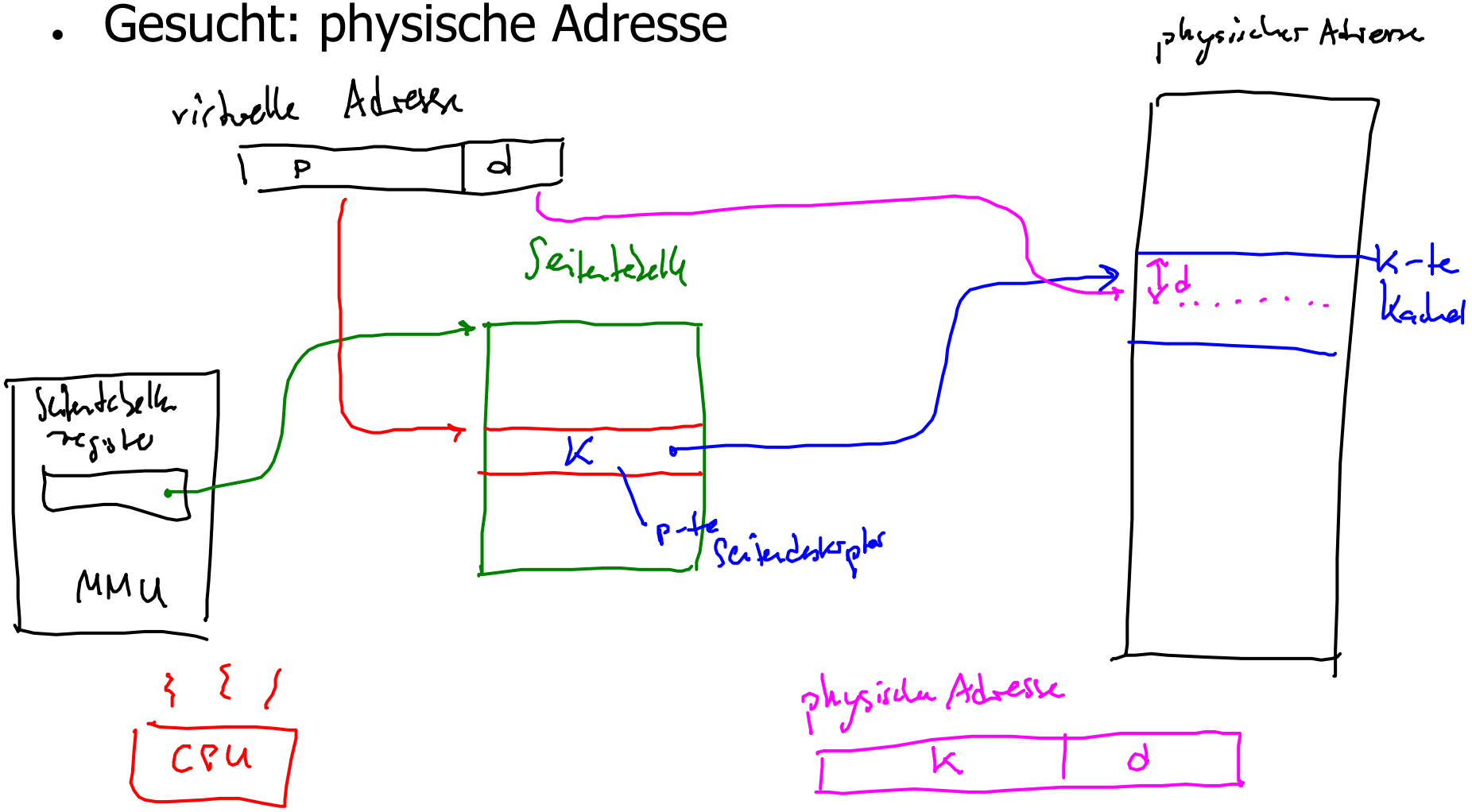
Seitendeskriptor

- Ein Seitendeskriptor enthält in der Regel folgende Informationen:
  - ~~Seiten~~Kachel adresse (physische Adresse der Seite im Hauptspeicher)
    Verweis auf die dazugehörige Hauptspeicherkachel
  - Presence-Bit (P-Bit)
    Ist Seite im Hauptspeicher?
  - Schutzbits
    Regeln Art des Zugriffs
  - Referenced-Bit (R-Bit)
  - Dirty-Bit (D-Bit)
  - Cache-Disable-Bit (C-Bit)
  - frei benutzbare Bits für das Betriebssystem



(Erfolgreiche) Seitenumsetzung

- Gegeben: Virtuelle Adresse, Zeiger auf Seitentabelle
- Gesucht: physische Adresse



Laufzeitfehler

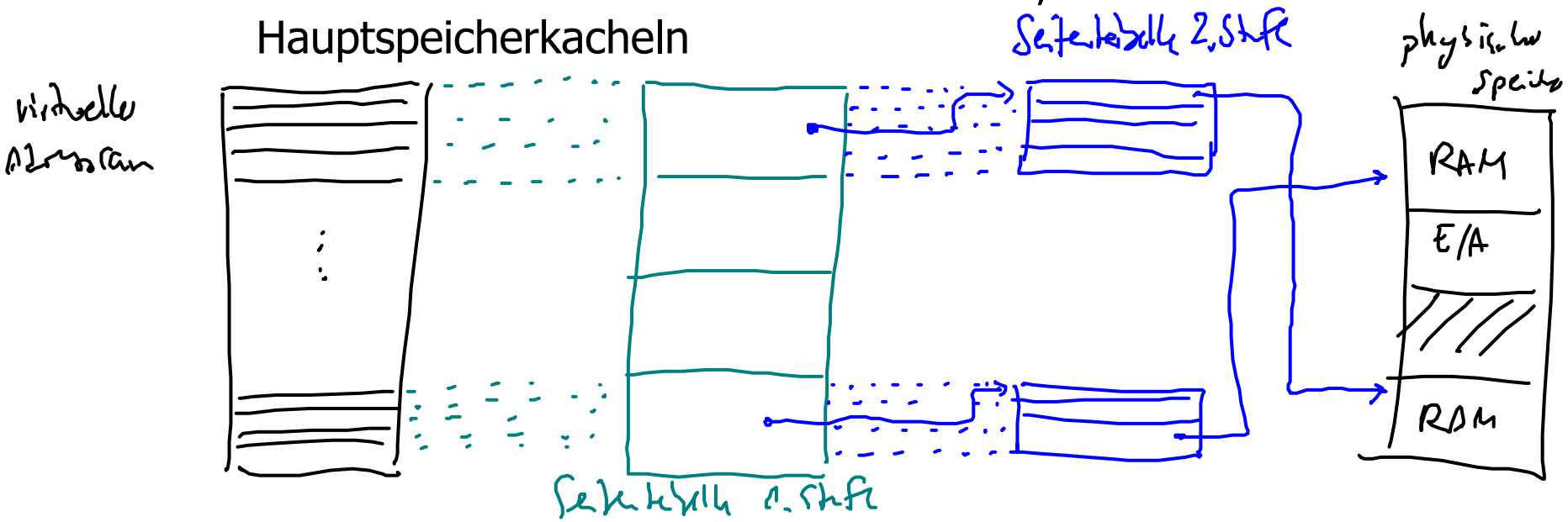
- Folgende Laufzeitfehler können bei der Seitenumsetzung auftreten:
  - Seite existiert nicht
    - In der Seitentabelle durch einen besonderen Nulldeskriptor gekennzeichet
    - Adressfehler: Adresse wird gegenwärtig von der Anwendung nicht benutzt, Unterbrechung
  - Zugriffsrechte stimmen nicht
    - Schutzbits der Seite entsprechen nicht dem Aufrufkontext
    - Schutzverletzung, Unterbrechung
  - Seite ist momentan ausgelagert
    - Erkennen mittels P-Bit
    - Unterbrechung, Einlagerung der Seite in den Hauptspeicher veranlassen (wird später genauer behandelt)

Diskussion

- Seitentabellen können sehr groß sein
  - Problem: Plazierung von Heap und Stack an unterschiedlichen Enden des virtuellen Speichers
  - Seitentabelle muss immer voll aufgebläht sein
  - Beispiel: 8 Byte pro Seitendeskriptor, 4 Kbyte Seitengröße (12 Bit Offset), 32 Bit Adressbus ergibt 8 Mbyte große Seitentabelle
    - Unverhältnismäßig für Systeme mit nur 16 bis 64 Mbyte Speicherausbau
  - Verschwendung vieler Seitendeskriptoren zwischen Heap und Stack
    - bei 20 Mbyte Programm werden 99,5% der Einträge in der Seitentabelle nicht benötigt
- Lösung: Mehrstufige Seitentabellen

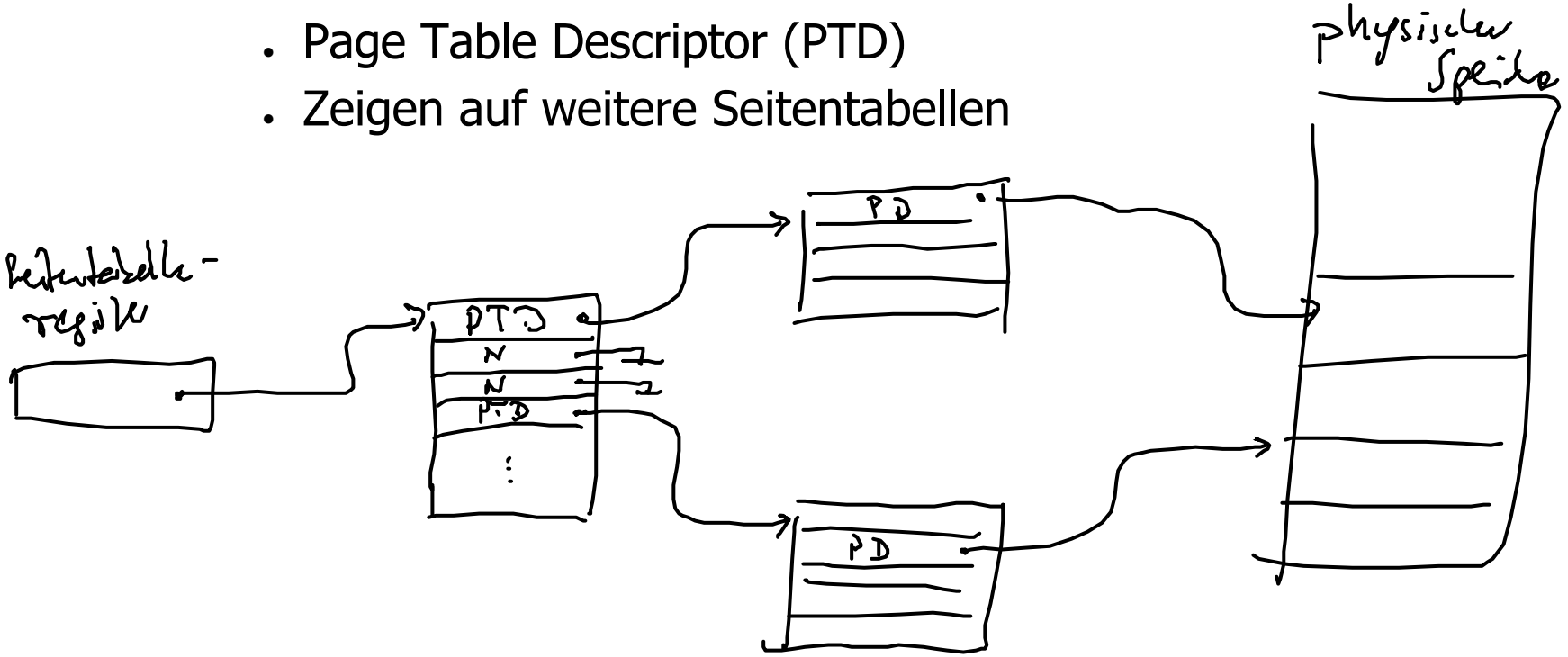
Mehrstufige Seitentabelle

- Verfeinerung der einstufigen Tabelle: mehrere Abbildungsschritte hintereinander
  - Gängig: zwei, maximal drei Abbildungsstufen
  - Jede Abbildungsstufe hat eigene Seitentabelle
  - Größe der beschriebenen Seiten wird mit jedem Schritt kleiner
- Beispiel: Zweistufige Abbildung
  - Erste Stufe: Unterteilung in wenige, dafür aber große Seiten
  - Zweite Stufe: Seitentabellen wie vorher, d.h. direkter Verweis auf Hauptspeicherkacheln



Baumstruktur der Verweise

- Mehrstufige Seitentabelle ist ein Baum aus Verweisen:
  - Blätter: Seitentabellen im ursprünglichen Sinn
    - Page Descriptor (PD)
    - Zeigen auf Hauptspeicherkacheln
  - Innere Knoten: Seitentabellendeskriptoren
    - Page Table Descriptor (PTD)
    - Zeigen auf weitere Seitentabellen



Seitentabellendeskriptor

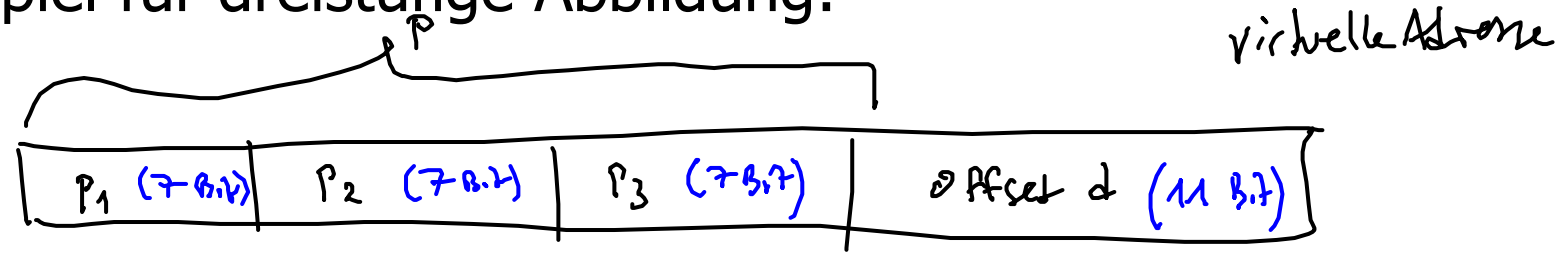
- Ob der momentane Eintrag ein Blatt oder ein innerer Knoten ist, wird anhand eines Typfeldes im Deskriptor festgelegt
  - Seitendeskriptoren enthalten auch ein Typfeld
- Ein Seitentabellendeskriptor enthält:
  - Typfeld (wie Seitendeskriptor)
  - Adresse der Seitentabelle
  - ~~Ggf. Größe der Seitentabelle~~ (analog zur ~~Segmentlänge~~)
  - Presence-Bit (P-Bit)
  - frei benutzbare Bits
- P-Bit: Auch Seitentabelle kann ausgelagert werden
  - siehe spätere Übung

Virtuelle Adresse

- Bei mehrstufigen Verfahren ist die virtuelle Adresse zusätzlich untergliedert:
  - Seitenanteil von p Bits wird in L Stufen unterteilt:

$$p = p_1 + p_2 + \ldots + p_L$$

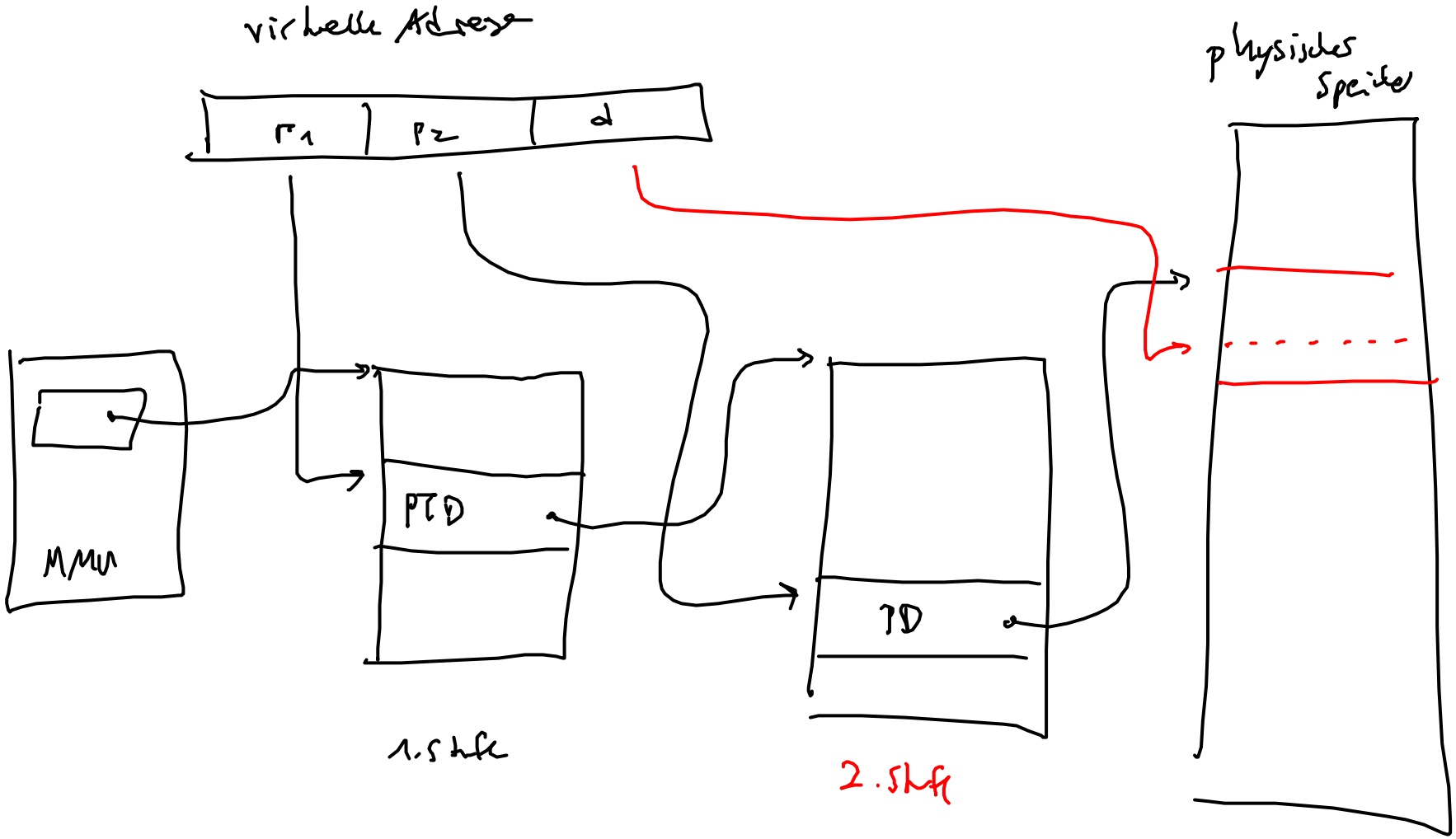
  - Adressabbildung beginnt beim höchstwertigen Bit
- Beispiel für dreistufige Abbildung:



  - Erste Stufe: 2^7 = 128 Seiten zu je 32 Mbyte
  - Zweite Stufe: 2^7 = 128 Seiten zu je 256 Kbyte
  - Dritte Stufe: 2^7 = 128 Seiten zu je 2 Kbyte

Mehrstufige Adressumsetzung

- Beispiel für zweistufige Abbildung



Mehrstufige Adressumsetzung

- Beispiel für zweistufige Abbildung mit Page Descriptor in der ersten Stufe
  - Hauptspeicherseite muss besonders ausgerichtet sein (deswegen diese Variante unüblich)

[zum selbermalen]

Dreistufige Abbildung