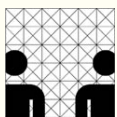


B Betriebssysteme

B3 Speicherverwaltung

- B1 Betriebssysteme: Einführung und Motivation
- B2 Prozesse: Scheduling und Betriebsmittelzuteilung
- B3 Speicherverwaltung**
- B4 Dateisysteme
- B5 Ein-/Ausgabe



B3 Speicherverwaltung (a)

B3.1 Programm-, Prozess- und Maschinenadressen

Def. **Programmadresse**:

Die Programmadresse ist diejenige Adresse, die im Adressteil des Befehls aufgeführt wird.

Def. **Programmadressraum**:

Menge aller möglichen Programmadressen.

Bem.: Programmadressraum ist befehlsformspezifische Größe (nicht programmspezifisch)

Bsp.: Bei s (Binär-) Stellen des Adressteils eines Befehls \rightarrow existieren 2^s Adressen für Programmadressraum

Def. **Prozessadresse**:

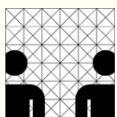
Die Adressen, die Prozess gegenüber Rechenanlage verwendet, heißen Prozessadressen (auch: **effektive Adressen** oder – bei Seiten-/Segment-adressierung, s.u. – **virtuelle Adressen**).

Def. **Prozessadressenraum**:

Menge aller Prozessadressen.

Bem.: – Prozessadressenraum ist programm- bzw. prozessindividuelle Größe

– max. Prozessadressenraum (virtueller Adressenraum) abhängig von Adressierungstechnik; kleiner, gleich oder größer als physikalischer Hauptspeicher



Maschinenadressen und Abbildung virtueller auf Maschinenadressen

Def. **Maschinenadressen**:

Nummern der Hauptspeicherzellen.

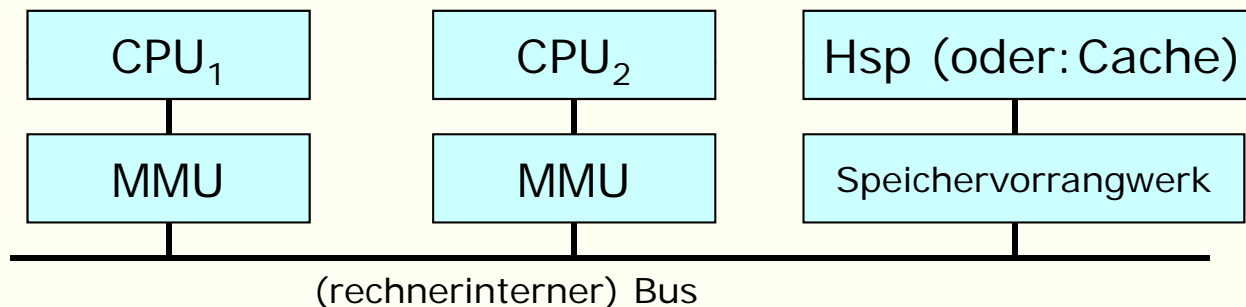
Def. **Maschinenadressraum**:

Menge aller Maschinenadressen.

Bem.: I. Maschinenadressraum: von Hauptspeicher-Ausbaustufe des Rechners abhängig

II. Abbildung von Prozess- auf Maschinenadressen notwendig

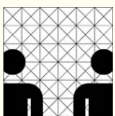
→ hierfür zuständig insbes.: **Memory Management Unit (MMU)** bzw. **Paged MMU (PMMU)**



(für MOTOROLA 68010)

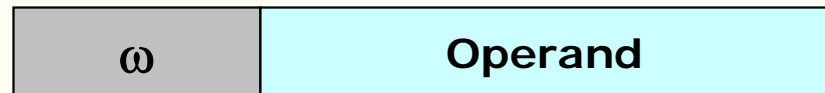
Funktionen der Memory Management Unit (MMU) insbes.:

- Abb. virtueller auf physikal. Adressen (d.h. Verwaltung des virtuellen Speichers), zu Details, vgl. u.a. Abschnitt B3.4
- Kontrolle des Pufferspeichers (Cache)
- Zugriffskontrolle zum Bus



B3.2 Programmbedingte Adressierungstechniken

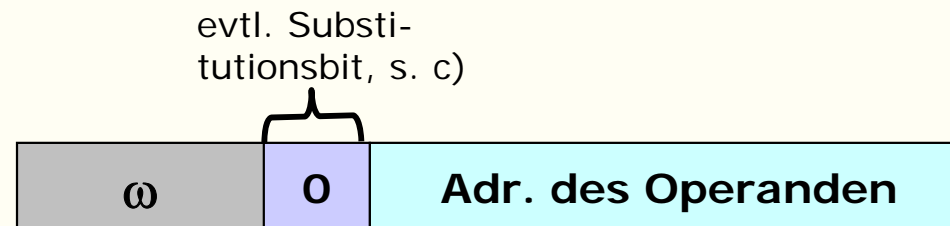
a) **Unmittelbare Adressierung:** *Adressierung 0. Referenzstufe*



z.B. (in VAX11-Assemblersprache) : **MOVB #8, R0**
→ Wert 8 (1 Byte) in Register R0 abgespeichert

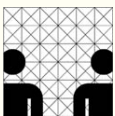
unmittelbare Adressierung: **"immediate mode"**

b) **Direktadressierung:** *Adressierung 1. Referenzstufe*



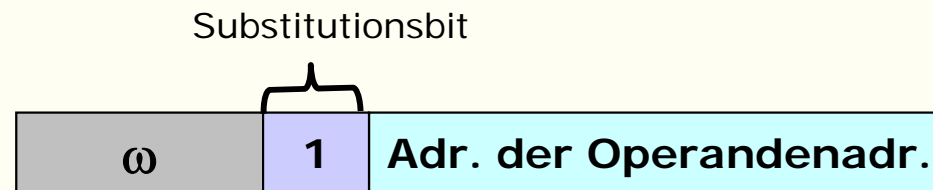
z.B. (in VAX11-Assemblersprache) : **MOVB @#800, R0**
→ Inhalt der Hsp-Zelle mit Adr. 800 (1 Byte) in Register R0

Direktadressierung: **"absolute mode"**



Programmbedingte Adressierungstechniken (Forts.)

c) **Indirekte Adressierung** (Substitution): *Adressierung 2.Referenzstufe*



z.B. (in VAX11-Assemblersprache): **MOVB @800, R0**

→ falls Inhalt der Hsp-Zelle mit Adr. 800 = a_0 :

Inhalt der Zelle mit Adr. a_0 (1 Byte) in Register R0

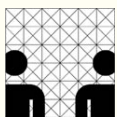
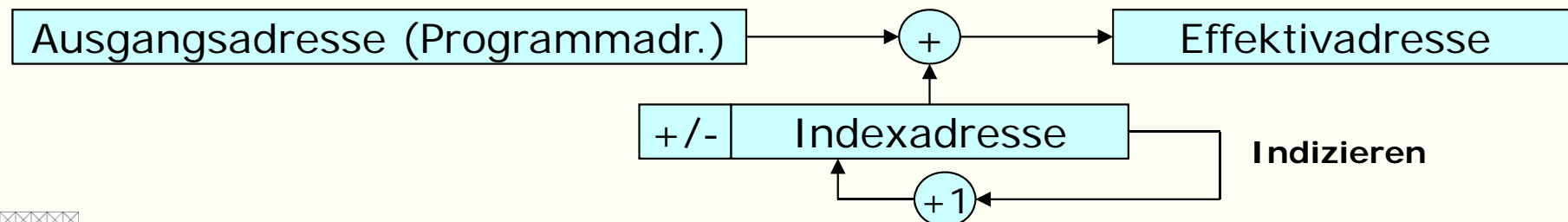
Indirekte Adressierung auf VAX11 approximierbar durch:

“relative deferred mode”

d) **Indizierte Adressierung** (Indizierung, Index-Adressierung)

zur Ausgangsadresse werden Inhalt(e) von ≥ 1 Indexregister(n) addiert;

Unterstützung der Bearbeitung von Vektoren; evtl. automatisches Inkrementieren/Dekrementieren



B3.3 Nutzung von Basisadressen

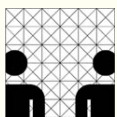
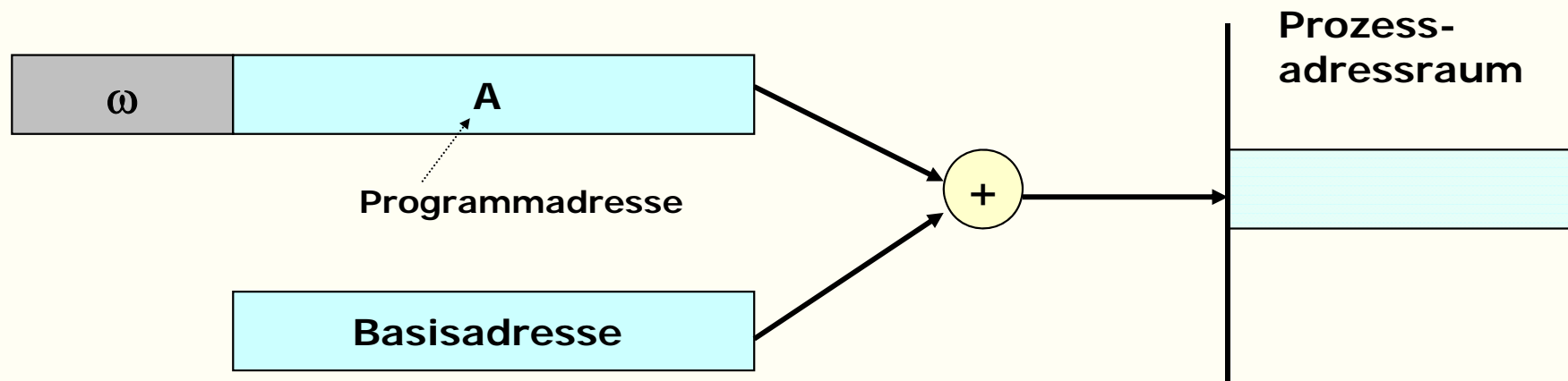
Offene Basisadressen

Häufige Eigenschaft von Prozessen:

Nur schmale Adressumgebungen benötigt für längere Intervalle (z.B. bei Durchlaufen von Schleifen)

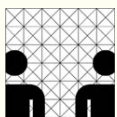
→ **Wunsch**: Beschreibung von Prozessadressen mittels kurzer Programmadressen (und Basisadresse), d.h. Befehlswortverkürzung

Lösung: **Adressbildung mit offenen Basisadressen**
(ohne Substitution und Indizierung):



Bemerkungen zur Verwendung offener Basisadressen

- **Basisadresse** = Grundadresse des aktuellen Ausschnitts
- Basisadresse in speziellem Register des Leitwerks gespeichert (vgl. **Basisadressregister** in CPU)
- 1:1- Zuordnung zwischen Adressausschnitt und Basisadresse
→ Ersetzung der Basisadresse bei Übergang zu neuem Ausschnitt, evtl. nur neues Basisadressregister wählen
- Anzahl der Prozessadressenräume = Anzahl der benutzten (verschiedenen) Basisadressen
- Adressrechnung bei jeder Adressierung mit offener Basisadresse
- Basisadressen heißen **offen**, da durch Prozess modifizierbar
- offene Basisadressierung ist kein Ersatz für Indizierung bzw. Substitution



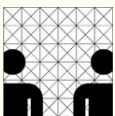
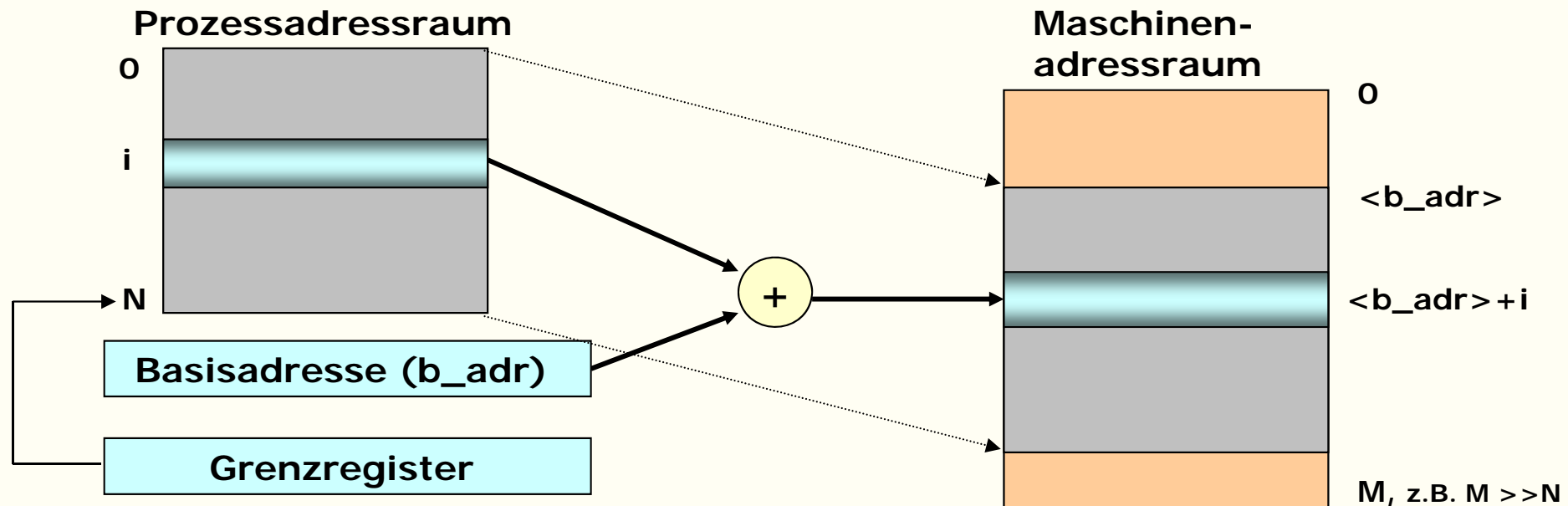
Nutzung von Basisadressen (Forts.)

Verdeckte Basisadressen

Adressierungstechniken in B3.3 und B3.4 (verdeckte Basisadressen, Seitenadressierung, Segmentierung) dienen zur Erleichterung des Mehrprogrammbetriebs.

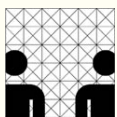
→ **Wunsch**: leichte Verschiebungsmöglichkeit der Daten des Prozesses im Hsp.; einfache Abbildung von Prozess- auf Maschinenadressen (u.a. Aufgabe des Laders).

Lösung: **Adressbildung mit verdeckten Basisadressen**



Bemerkungen zur Verwendung verdeckter Basisadressen

- Prinzip der verdeckten Basisadressierung analog zu offener Basisadressierung
→ *jedoch*: keine Zugriffsmöglichkeit auf Basisadressen durch Benutzer
- verdeckte Basisadressen ersparen Lader Adressumsetzungen
→ Programm ist lageinvariant
- ***Grenzregister*** (*limit register*) speichert höchste zugelassene Prozessadresse
→ Kontrolle bei jedem Hauptspeicher-Zugriff durch Leitwerk (prozessorientierter Speicherzugriffsschutz)



B3.4 Segmentierung und Seitenadressierung

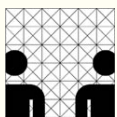
Segmentierung mit verdeckten Basisadressen

Def. **Segment**:

Zusammenhängender Bereich im Prozessadressraum, der bzgl. der Zugriffsrechte homogen ist und höchstens *einen* umfangsvariablen Inhalt besitzt.

Bemerkung:

- Prozess arbeitet auf zahlreichen Segmenten (funktionelle Einheiten, z.B. Blöcke, Prozedurrümpfe); evtl. Segmente auf Peripheriespeicher
- einem Segment ist zugeordnet:
Nummer, verdeckte Basisadresse (zur Abbildung auf Maschinenadressraum), Grenzregister, Zugriffsrecht
- Segmentliste = Menge aller Segmentbasisadressen

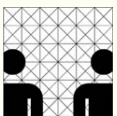
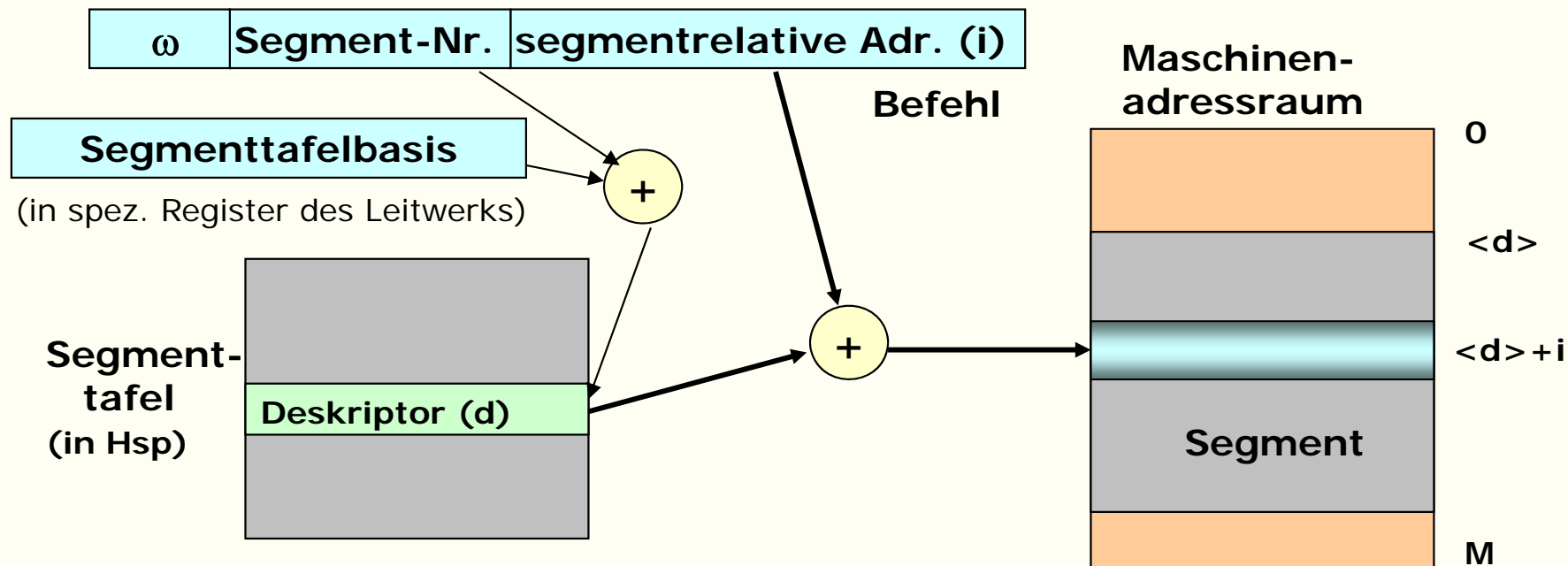


Segmentierung (Forts.)

Vorteile der Segmentierung:

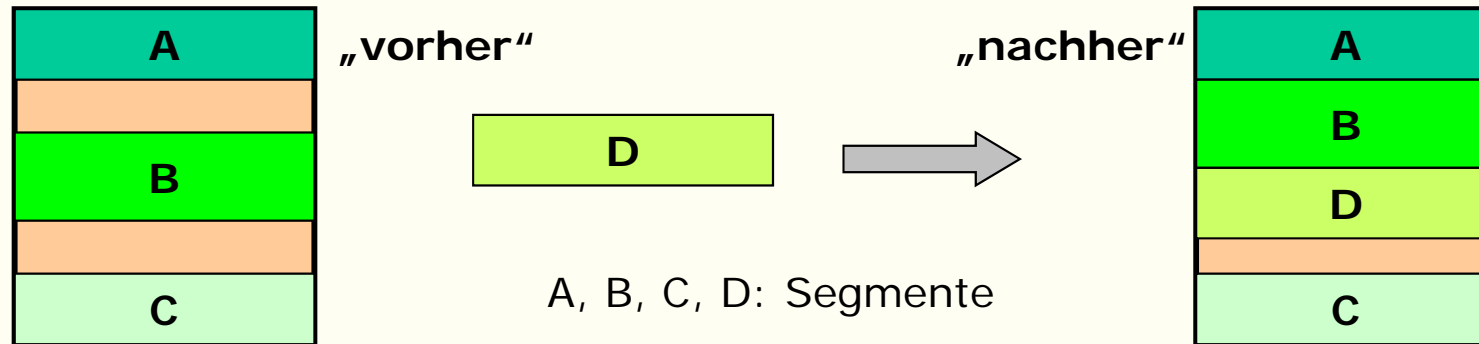
- keine Adressumsetzungen durch Montierer notwendig
- Segmente in Maschinenadressraum beliebig abspeicherbar (u.a. dynamisches Vergrößern)
- gemeinsame Nutzung von Segmenten durch verschiedene Prozesse (prozessspezifische Segmentlisten)

Adressierungsschema bei Segmentierung mit Basisadressierung



Seitenadressierung I

Problem: notwendige Verschiebung von Segmenten im Hauptspeicher bei Segmentadressierung

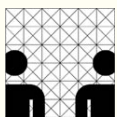


→ Vermeidung von Verschiebungen durch Seitenadressierung

Def. **Kacheln** (*physical pages, page frames*):
durch Unterteilung des Maschinenadressraums gewonnene
Einheiten konstanter Größe.

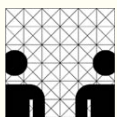
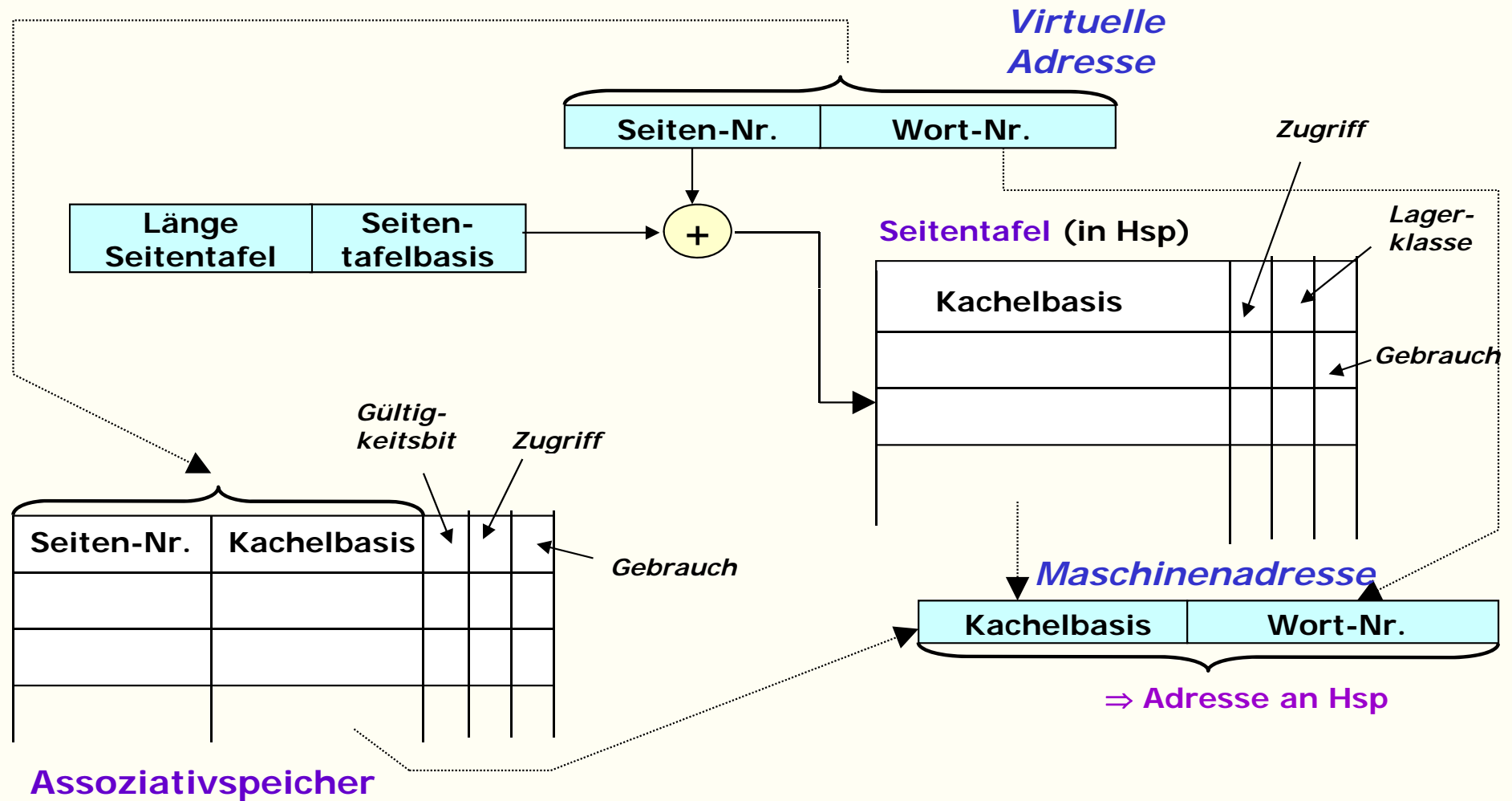
Def. **Seite** (*page*):
Teilmenge des Prozessadressraums in der Größe von einer
Kachel.

Bem.: typische Seiten-/Kachelgrößen: 1024, 2048 Wörter (klein gegenüber
mittlerer Segmentgröße)



Seitenadressierung II

Abb. von Prozessadressen auf Maschinenadressen bei Seitenadressierung

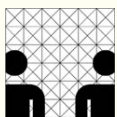
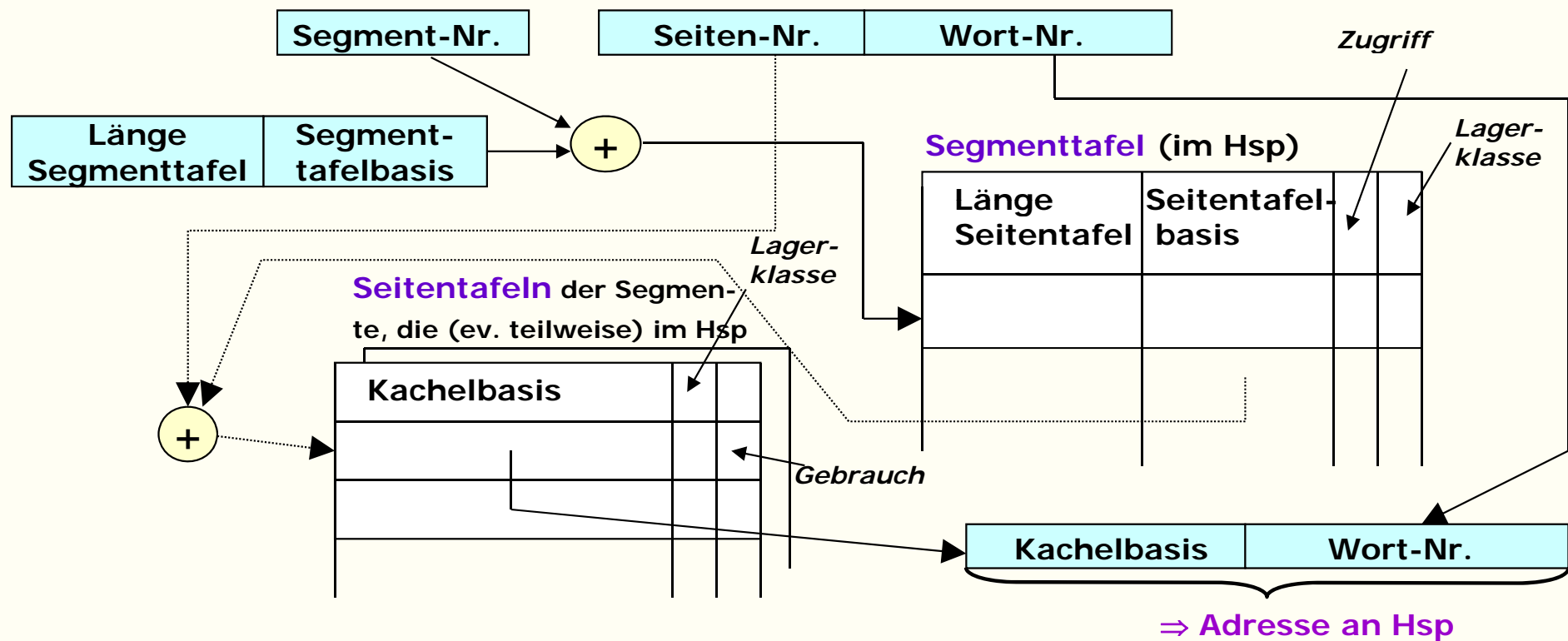


Segmentierung mit Seitenadressierung I

Kombination der Adressierungsverfahren:

- Segmentierung zur Abb. von Programm- in Prozessadressenräume
- Seitenadressierung zur Abb. von Segmenten in Maschinenadressraum

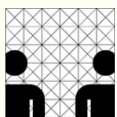
Adressierungsschema bei Segmentierung mit Seitenadressierung



Segmentierung mit Seitenadressierung II

Vorteile:

- Vereinfachung des Lade-/Montiervorgang (durch Segmentierung)
- Verschiebung von Segmenten überflüssig (durch Seitenadressierung), allerdings: Hsp-Verschnitt
- leichte Vergabemöglichkeit von Zugriffsrechten
- Folge: jeweils 3 Hsp-Zugriffe notwendig:
 - auf Segmenttafel
 - auf Seitentafel
 - auf Operanden

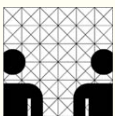


Seitenverdrängungsstrategien

- Sofern benötigte Seite im Hsp nicht vorhanden, entsteht
SEITENFEHLER (page fault)
→ entsprechende Kachel von Plattenspeicher einzulesen
(... und *betroffener Prozess* wartet nunmehr auf E/A, d.h. er *ist blockiert*)
- Nach Einlesen der Seite von der Platte:
irgendeine Kachel im Hsp. muss überschrieben („verdrängt“) werden ⇒ welche ???
- Ergo: **Seitenverdrängungsstrategien** benötigt, wie z.B.
 - S1: **Random (zufällig)**: Zufällig ausgewählte Seite wird verdrängt.
 - S2: **FIFO** : Älteste Seite im Kachelspeicher wird verdrängt.
 - S3: **LFU** (*"Least Frequently Used"*): Eine Seite mit den wenigsten Zugriffen wird verdrängt; Näherungsimplementation erfolgt über "Benutzt"-Zähler.
 - S4: **LRU** (*"Least Recently Used"*): Eine Seite mit max. Rückwärtsdistanz (am längsten unbenutzt) wird verdrängt; Implementation z.B. mittels eines Kellers.
 - S5: **RNU** (*"Recently Not Used"*): Eine Seite, die im letzten Beobachtungsintervall nicht referiert wurde, wird verdrängt.

Problem: evtl. Vielzahl von Seitenfehlern bei (zu) hohem Grad von Multiprogramming, d.h. zu viele Prozesse konkurrieren um Hsp

⇒ **"Seitenflattern" (Thrashing)**



Optimale “Füllung” des Rechners mit Prozessen bei Paging

Füllung $f(t)$ = Anzahl der zu einem Zeitpunkt t im Rechner existierenden Prozesse (z.B. bezogen auf Prozesse, die sich in einem der Prozesszustände „*blockiert*“, „*bereit*“ und „*CPU-belegend*“ befinden, vgl. das Prozessmodell aus Kap. B2).

Nachteile bei

zu großer Füllung:

- weniger Hauptspeicher (bzw. Cache) pro Prozess und deshalb größere Wahrscheinlichkeit für Seitenfehler \Rightarrow mehr E/A, häufigeres „Context Switching“;
- Erhöhung der E/A-Zeiten bei Seitenfehler.

zu kleiner Füllung:

- i.d.R. größere Wahrscheinlichkeit, dass alle grundsätzlich rechenwilligen Prozesse E/A-aktiv sind (ergo: temporär kein Prozess im Zustand „bereit“ und somit CPU-Vergabe/-Nutzung unmöglich)

Gewählter Grad von Multiprogramming legt Füllung weitestgehend fest.
Optimale Wahl u.a. abhängig von Hsp-Größe, Lokalitätsverhalten der Prozesse, E/A-Zeiten und Rechenintensität der Prozesse (im Vgl. zu ihrer E/A-Intensität)

