Rechnerarchitektur I (RAI)

Speicherarchitektur

Prof. Dr. Akash Kumar

Chair for Processor Design

cfaed.tu-dresden.de

TECHNISCHE UNIVERSITÄT DRESDEN

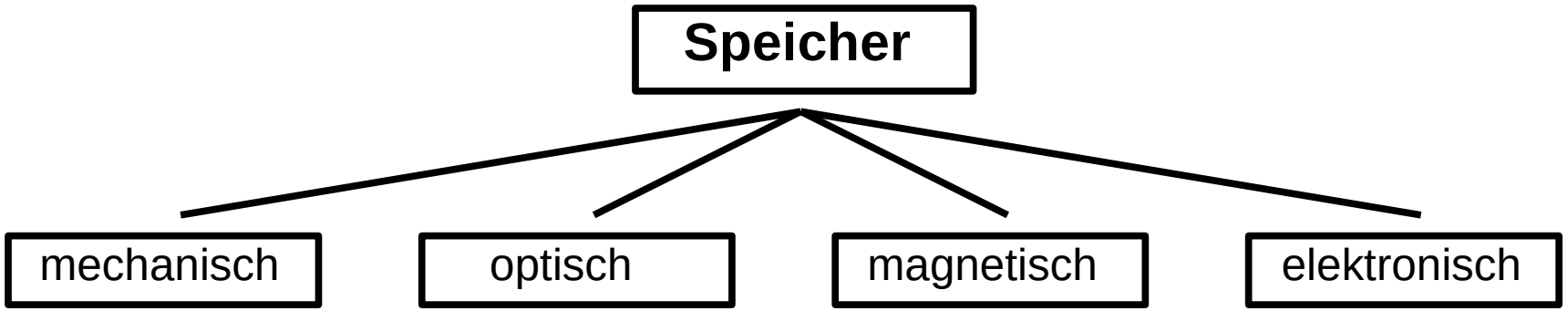
DRESDEN concept

DFG WR WISSENSCHAFTSRAT

Inhalt

- Speicherübersicht
- Speicherhierarchie
- Speicherzugriff
- Virtueller Speicher

Speicherübersicht



mechanisch	optisch	magnetisch	elektronisch
Nockenwalze	Mikrofilm	Ferritkern	Halbleiter
Lochkarte	CD-ROM	Magnetblasen	Speicherprinzip:
Lochstreifen	DVD	Magnettrommel	- elektrostatisch
	MO-Disk	Magnetband	- elektrodynamisch
	Holographie	Magnetplatte	- ferromagnetisch
		Magnetkarte	- ferroelektrisch
			- magnetoresistiv

Unterteilung nach physikalischer Haupteigenschaft (speichern, lesen, schreiben)

Anforderungen an Speicher

Persistente Speicherung:

Die Speicherzustände sind zeitlich stabil und reproduzierbar.

Die Speicherzustände bleiben auch ohne Energieversorgung erhalten.

Veränderbarkeit und Lesbarkeit:

Der Speicherzustand ist zu jedem beliebigen Zeitpunkt lesbar und je nach Anwendung ein- oder mehrmalig oder beliebig oft veränderbar.

Auswählbarkeit:

Die Speicherzustände sind einzeln auswählbar, ansprechbar und eindeutig unterscheidbar.

Bündelung:

Mehrere Speicherzustände sind zu einer Einheit, einem Block zusammengefasst und nur gemeinsam ansprechbar, lesbar oder änderbar.

Eigenschaften von Speichern

Zöiffsart:

sequentiell, strukturiert	SAM (Serial Adressable Memory)
inhaltsbezogen, assoziativ wahlfrei	CAM (Content Adressable Memory)
adressierbar	RAM (Random Access Memory)

Änderbarkeit:

fester Inhalt, nur lesbar	ROM (Read Only Memory)
einmalig programmierbar, nur lesbar	PROM (Programmable ROM)
mehrmalig programmierbar, löschbar, nur lesbar	EPROM (Erasable PROM)
beliebig oft les- und schreibbar	RWM (Read Write Memory)

Persistenz, Zeitverhalten:

statisch, flüchtig	SRAM (Static RAM)
dynamisch, flüchtig	DRAM (Dynamic RAM)
auch bei Spannungswegfall nicht flüchtig	NVRAM (Non-volatile RAM)

Komponenten von Speichern

Speicherelement:
realisiert (physikalisch) die Speicherzustände (meist binär).

Speicherzelle:
Zusammenfassung von Speicherelementen, kleinste ansprechbare Einheit.

Zugriffsbreite:
paralleler Zugriff auf mehrere Speicherzellen gleichzeitig (Bit, Byte, Wort, ...).

Speicherwort:
Feste Anzahl von Speicherzellen, die gleichzeitig als Block aus dem Speicher gelesen bzw. in ihn geschrieben werden kann (typisch 1 Wort).

Speicherkapazität:
Anzahl der effektiv nutzbaren Speicherzellen.

Ansteuerung:
taktsynchrone oder asynchrone Ansteuerung des Speichers.

Zeitverhalten von Speichern

Zugriffszeit (*Access Time*):
minimale Zeit (Latenz) für einen Lese-/Schreibzugriff (Adresse → Datum)

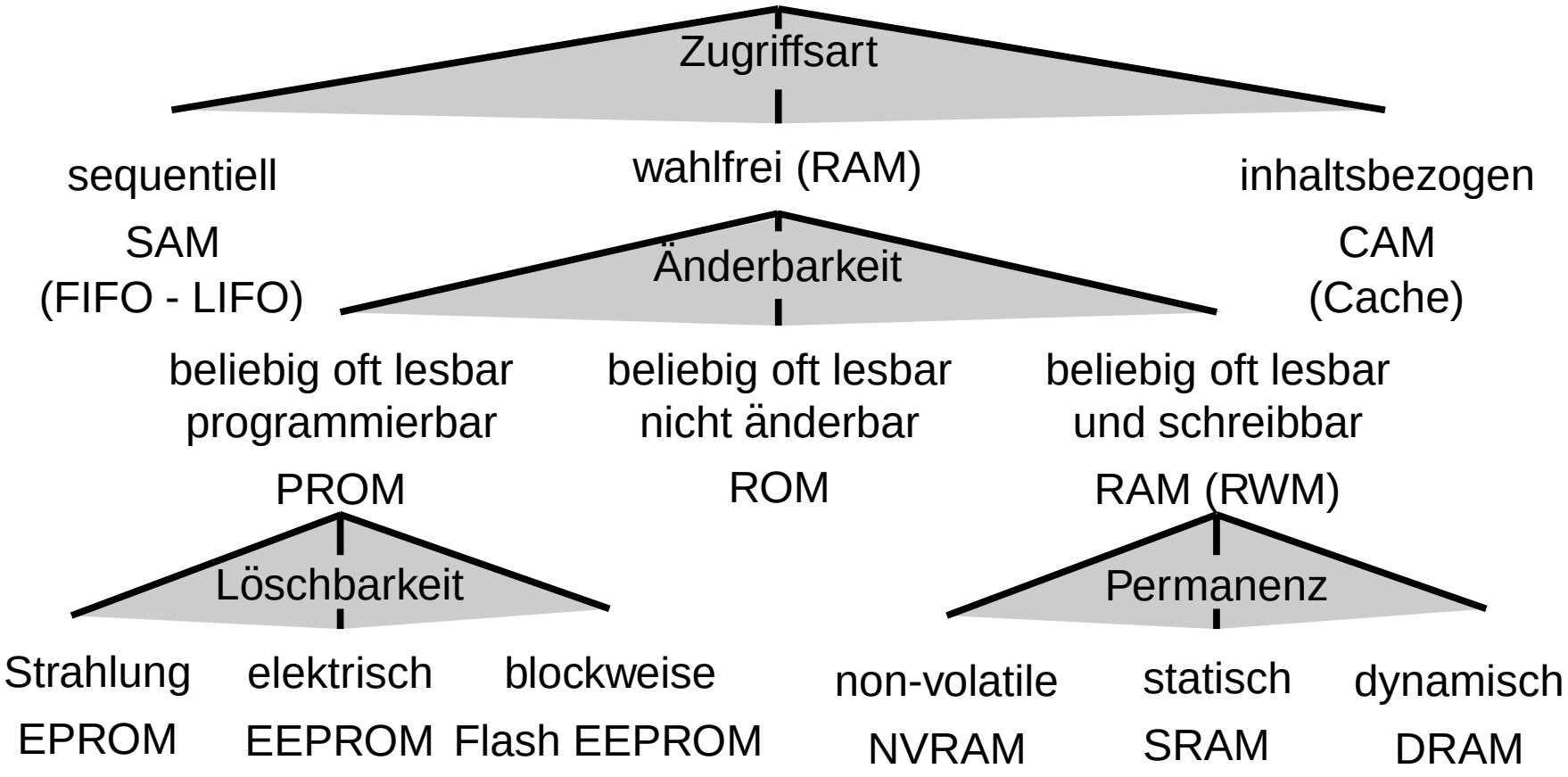
Zykluszeit (*Cycle Time*):
minimale Zeit zwischen zwei aufeinanderfolgenden Lese-/Schreibzugriffen (einschließlich Auffrischen)

Auffrischen (*Refresh*):
Bei Speichern mit destruktivem Lesen bzw. bei dynamischen Speichern muss der Speicherinhalt nach dem Lesen bzw. zyklisch wieder aufgefrischt werden.

Refreshzeit(*Refresh Time*):
Zeit zum Auffrischen des Speichers (nach einem Zugriff bzw. zyklisch)

Speicherbandbreite:
maximale Datenmenge pro Zeiteinheit bzgl. der Lese-/Schreibzugriffe

Halbleiterspeicher



Alle ROM-Typen sind non-volatile

Technisch/Technologisches Problem

Speicherkapazität ⟺ Speicherzugriffszeit

Speicher mit kurzen Zugriffszeiten lassen sich nur mit relativ geringer Kapazität und mit hohem Kostenaufwand realisieren.

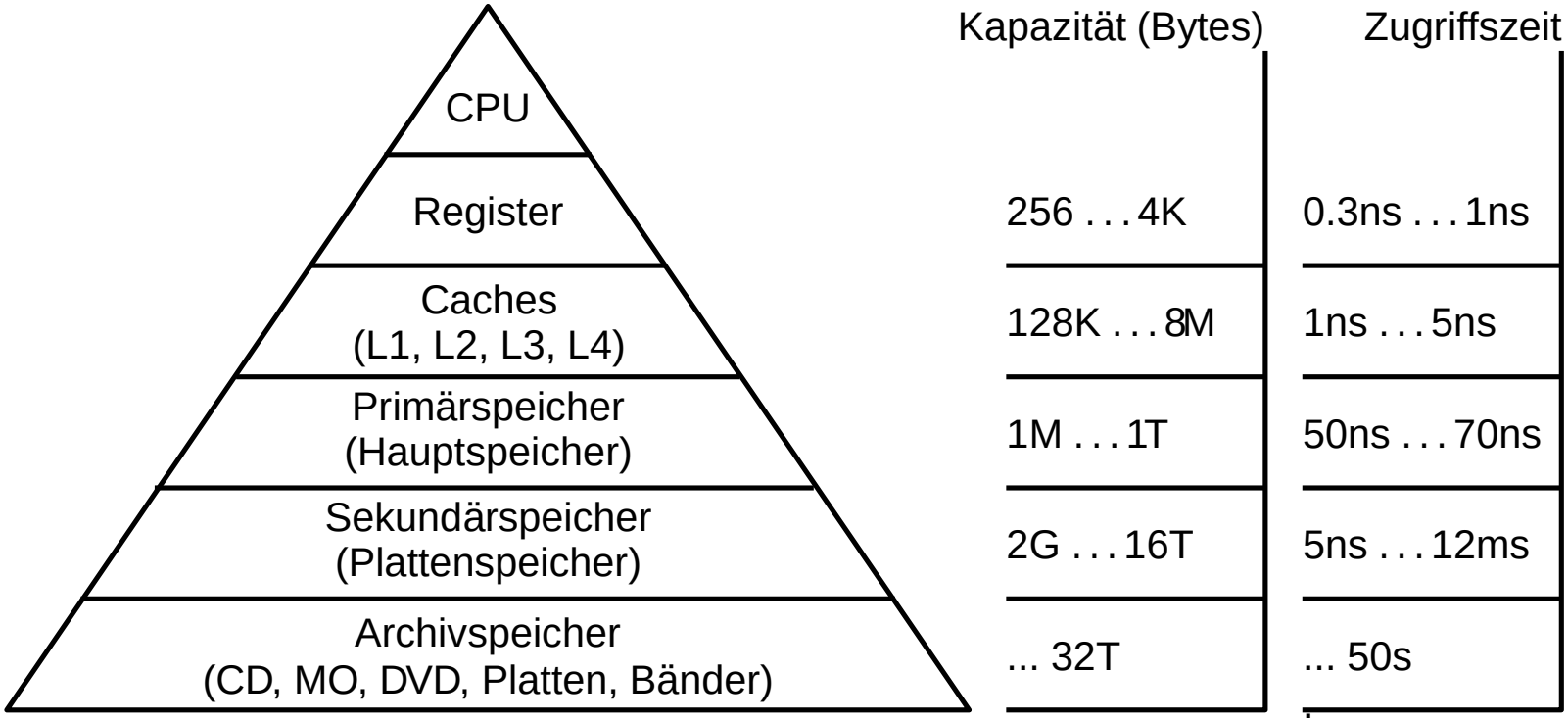
Dagegen lassen sich Speicher mit großer Kapazität nur mit relativ langen Zugriffszeiten kostengünstig realisieren.

Mit der Entfernung von der CPU wachsen die Zugriffszeit und die Kapazität der Speicher gleichermaßen, die Realisierungskosten pro Byte sinken dagegen.

Zielstellung:

- Erhöhung der Speicherbandbreite und Verringerung der Zugriffszeiten bei gleichgroßer oder größerer Speicherkapazität (Realisierungsfrage).
- Minimierung der Speicherkosten/Byte Speicherkapazität (Kostenfrage).

Speicherhierarchie



Lokalitätsprinzip des Speichers

Zeitliche Lokalität:

Nach einem Speicherzugriff ist die Wahrscheinlichkeit hoch, dass in einem der nächsten Befehle ein erneuter Zugriff den selben Speicherplatz erfolgt.

Örtliche Lokalität:

Nach einem Speicherzugriff ist die Wahrscheinlichkeit hoch, dass in einem der nächsten Befehle ein Zugriff auf einen benachbarten Speicherplatz erfolgt.

Ursachen zum Lokalitätsprinzip:

Befehle: Sequentieller Befehlsstrom, Programmschleifen, Unterprogramme, Abspeicherung in Bibliotheken . . .

Daten: Zusammenhängende Datenstrukturen, Speicherzuteilung, Variablenanordnung durch Compiler . . .

90/10 Regel beim Speicher

90% aller Speicherzugriffe erfolgen auf nur 10% der Speicherplätze.

Konfliktlösung: Speicherkapazität Zugriffszeit

Lösung unter Ausnutzung des Lokalitätsprinzips:

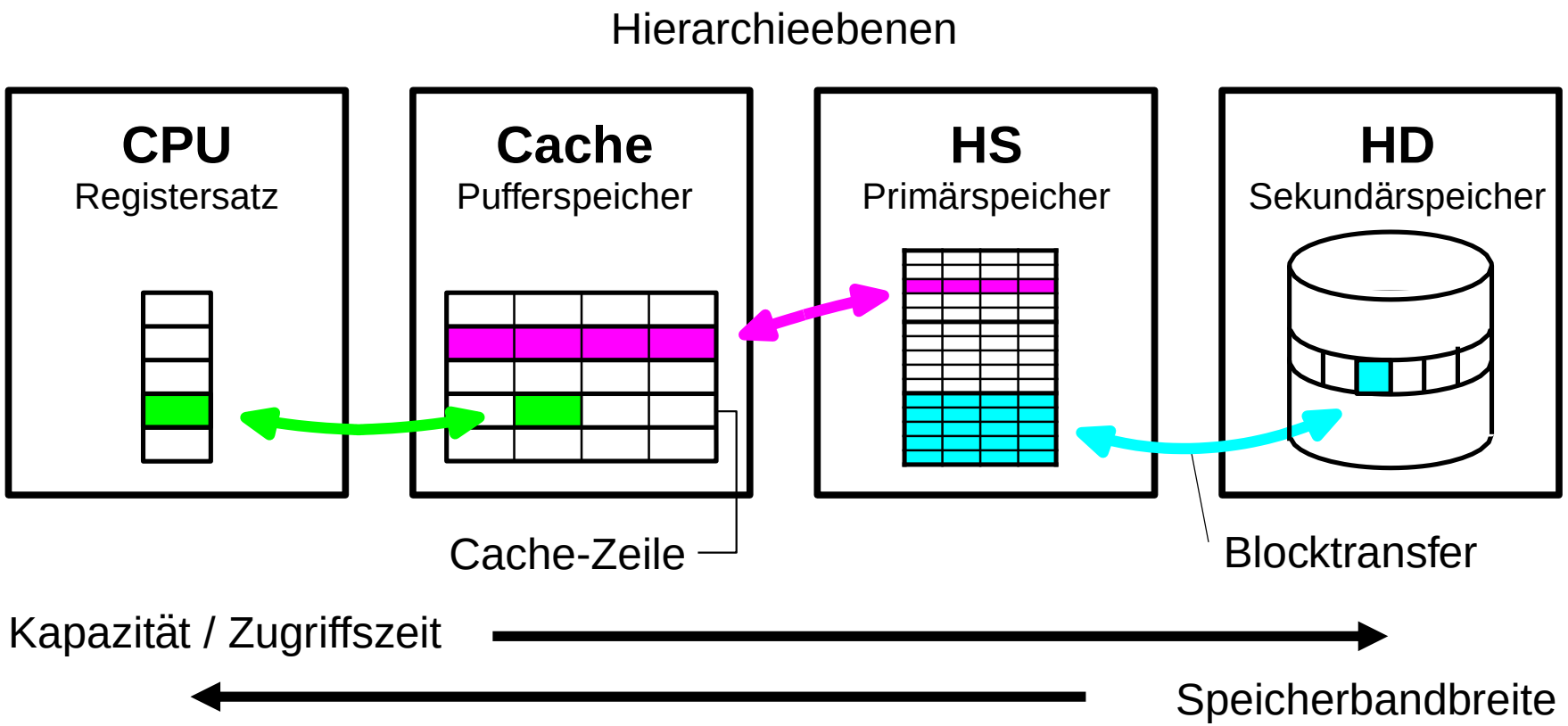
- Häufig benötigte Daten werden im nahen kleinen und schnellen Speicher gehalten (als Kopien der Originaldaten).
- Seltener benötigte Daten werden im fernen großen, jedoch langsamen Speicher gehalten.
- Reicht die Kapazität des kleinen schnellen Speichers nicht mehr aus, so werden nicht mehr benötigte Daten in den großen langsamen Speicher ausgelagert und die neu benötigten Daten aus diesem nachgeladen.

⟹ **Aufbau einer Speicherhierarchie, Bildung von Hierarchieebenen**

Blockprinzip:

Der Austausch der Daten zwischen den einzelnen Hierachieebenen erfolgt nicht einzeln, sondern zusammengefasst zu Blöcken (effektiverer Datentransfer).

Speicherhierarchie - Blockprinzip



Unterschiedliche Blockgrößen zwischen den Hierarchieebenen sind möglich.

Grundbegriffe der Speicherhierarchie

Block: Kleinster zusammenhängender Bereich, der zwischen den Hierarchieebenen transferiert, ausgetauscht wird (nicht notwendig gleich groß).

Frame (Blockrahmen): Bereich der höheren Hierarchieebene in den ein Block eingelagert, transferiert werden kann.

Blocktransfer: Blockaustausch zwischen zwei Hierarchieebenen (Burst, . . .).

Treffer (*Hit*): Die gesuchten Daten befinden sich in einem Frame der höheren Hierarchieebene und sind aktuell.

Trefferrate (*Hit Rate*): Relative Trefferanzahl bezogen auf die Gesamtzugriffe.

Fehlzugriff (*Miss*): Die gesuchten Daten wurden nicht in einem Frame der höheren Hierarchieebene gefunden bzw. sind nicht aktuell, kein Treffer.

Fehlerrate (*Miss Rate*): Relative Fehleranzahl (= 1 - Trefferrate).

Fehlerzuschlag (*Miss Penalty*): Zeit, die bei einem Fehlzugriff zusätzlich bis zur Erlangung der Daten benötigt wird.

Probleme der Speicherhierarchie

Abbildungsproblem:

Wie erfolgt die Abbildung der Blöcke einer Ebene auf die der nächst höheren?

Identifikationsproblem:

Wie werden die gesuchten Daten (Blöcke) lokalisiert und identifiziert?

Ersetzungsproblem:

Welcher Block wird beim Nachladen eines neuen ersetzt und wie?

Aktualisierungsproblem:

Wann und wie erfolgt die Aktualisierung der Blöcke in den einzelnen Ebenen bei der Veränderung von Daten in einem Block?

Konsistenzproblem (Kohärenzproblem):

Die Daten der Blöcke einer Hierarchieebene sind konsistent in den Blöcken aller niederen Ebenen enthalten (Datenkonsistenz über alle Ebenen – mit Ausnahme des Registersatzes) $\Longrightarrow$ Aktualisierungsproblem.

Registersatz

Register:

- SRAM-Speicher (Flipflop-Kette) innerhalb der CPU (mit CPU-Takt getaktet),
- Nutzung als Universalregister oder als Spezialregister (Registertypen, . . .),
- Sonderfunktionen, auch verteilt innerhalb der CPU (Hilfsregister, . . .),
- Realisierung in verschiedenen Datenformaten (Halbwort, Wort, . . .),
- Nutzung zur Rechnersteuerung (Befehlszähler, Statusregister, . . .).

Registersatz (*Register File*):

- fest organisierter Satz von Registern innerhalb der CPU (8, 16, 32, . . .),
- Operandenspeicherung, Registerspeicher der ALU (Daten, Adressen, . . .),
- Register werden nicht adressiert, sondern direkt ausgewählt (Multiplexer),
- werden als Multiport-RAM ausgeführt (z.B. 2 Leseports + 1 Schreibport, . . .),
- aufwendige Hardware-Realisierung (Registerfenster, Schattenregister, . . .).

Übersicht Registersatz

Übersicht Registersatz

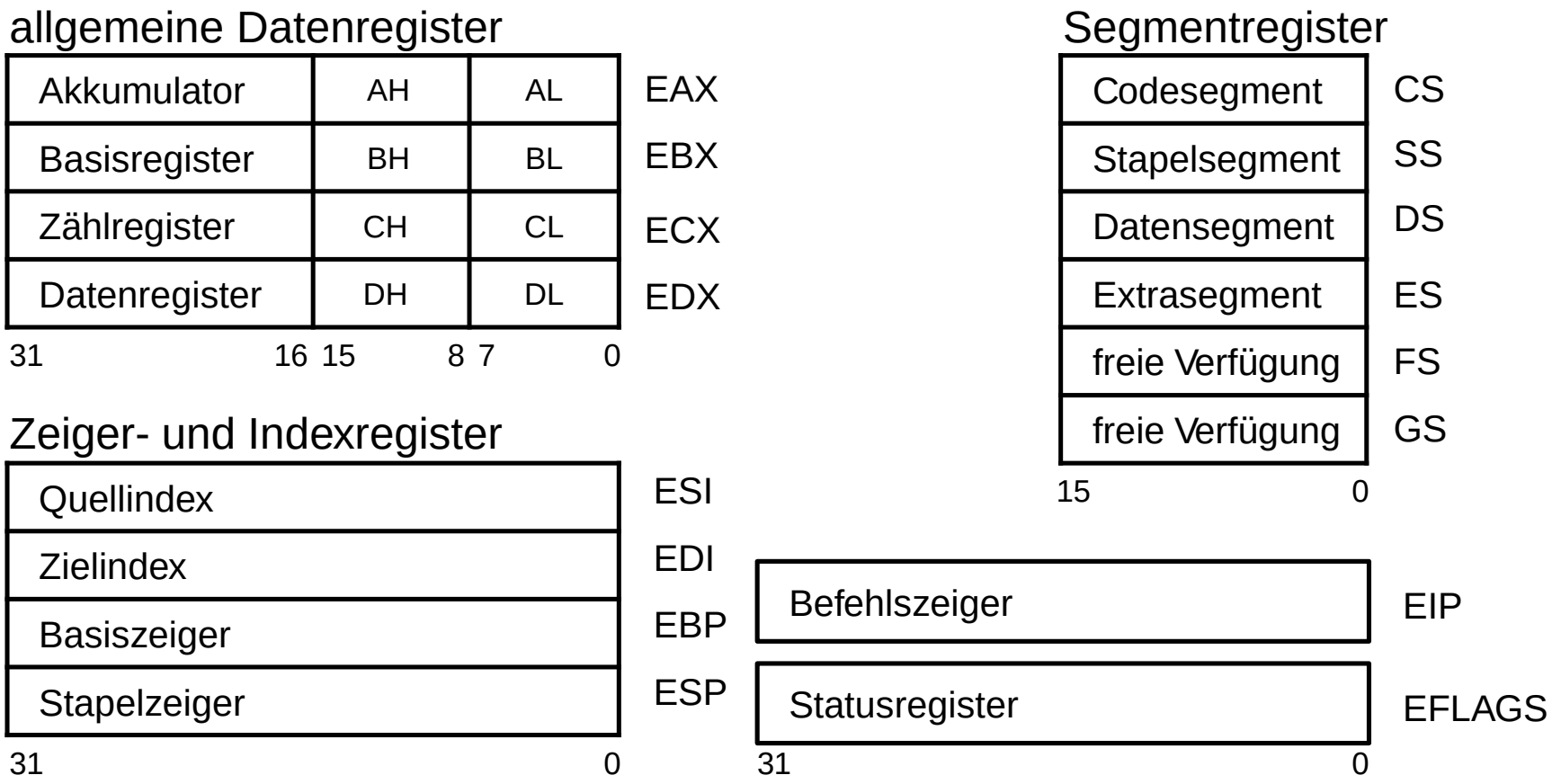
Registersatz

- Datenregister (data register)
  - Ganzzahlenregister (integer register)
  - Gleitkommaregister (floating point register)
- Adressregister (address register)
  - Basisregister (pointer register)
  - Indexregister (displacement register)
  - Segmentregister (segment register)
- Spezialregister (special purpose register)
  - Programmzähler (programm counter, PC)
  - Folgebefehl (instruction pointer, IP)
  - Statusregister (status register)
  - Basisregister (vector base register)
  - Stackregister (stack pointer)
  - Hilfsregister (auxiliary register)

Bei RISC zusammengefasst zum Universal-Registersatz (general purpose register)

GPR-Architektur

Die primären Register des Pentium II



Erweiterungen des Registersatzes

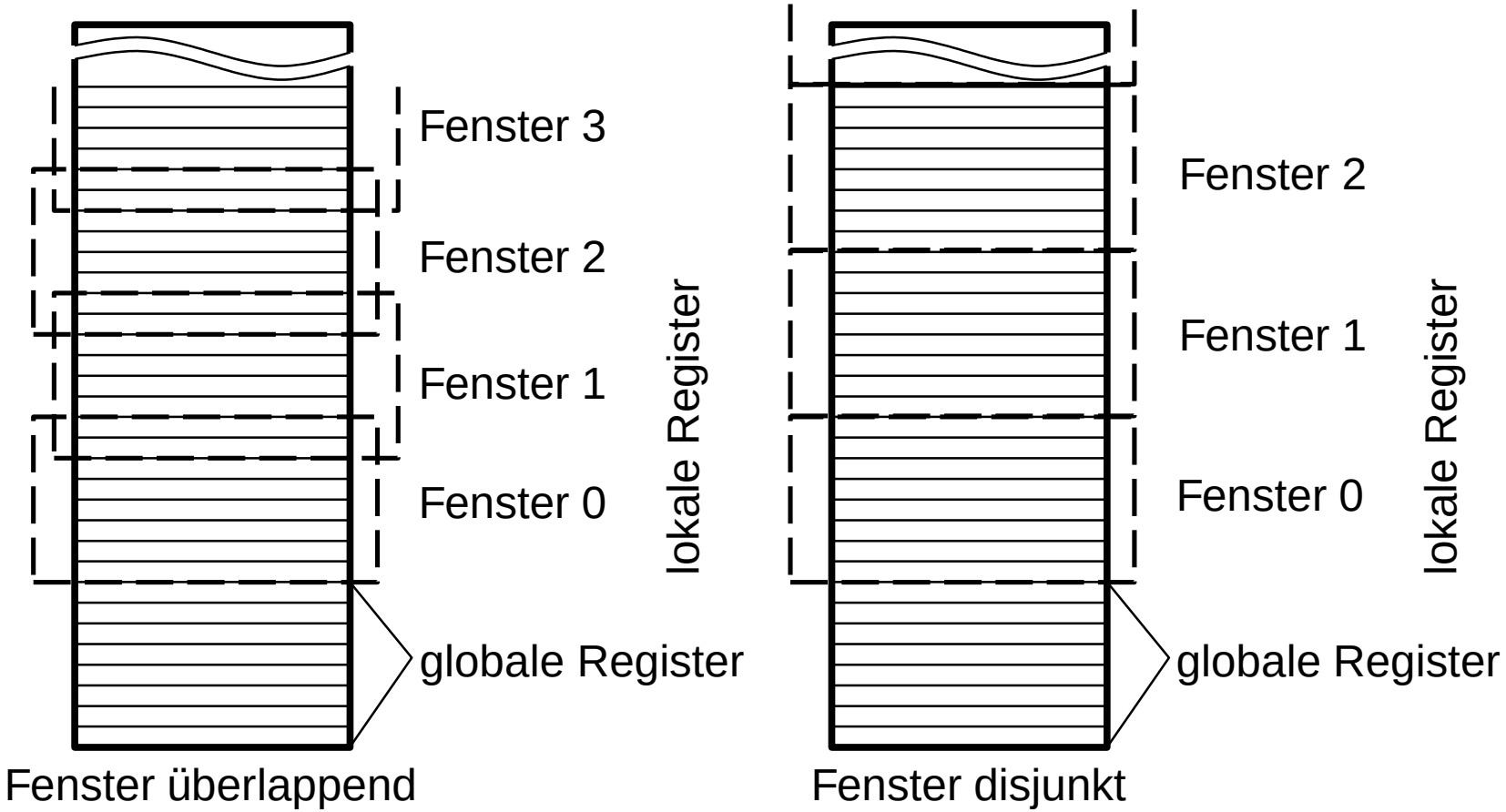
Globale, lokale Register (*global*, *local register*):
Unterteilung des Registersatzes in globale und lokale Register (z.B. Variablenübergabe bei Unterprogrammtechnik). Wurde oft mit Registerfenstern implementiert.

Registerfenster (*register window*):
Unterteilung eines großen Registersatzes durch Registerfenster (kostante, variable Fenstergröße, überlappende, disjunkte Fenster). Lokalisierung des aktuellen Fensters durch Fensterzeiger (Current Window Pointer, CWP), der im Statusregister gespeichert wird (Reduktion des Registeradressraumes).

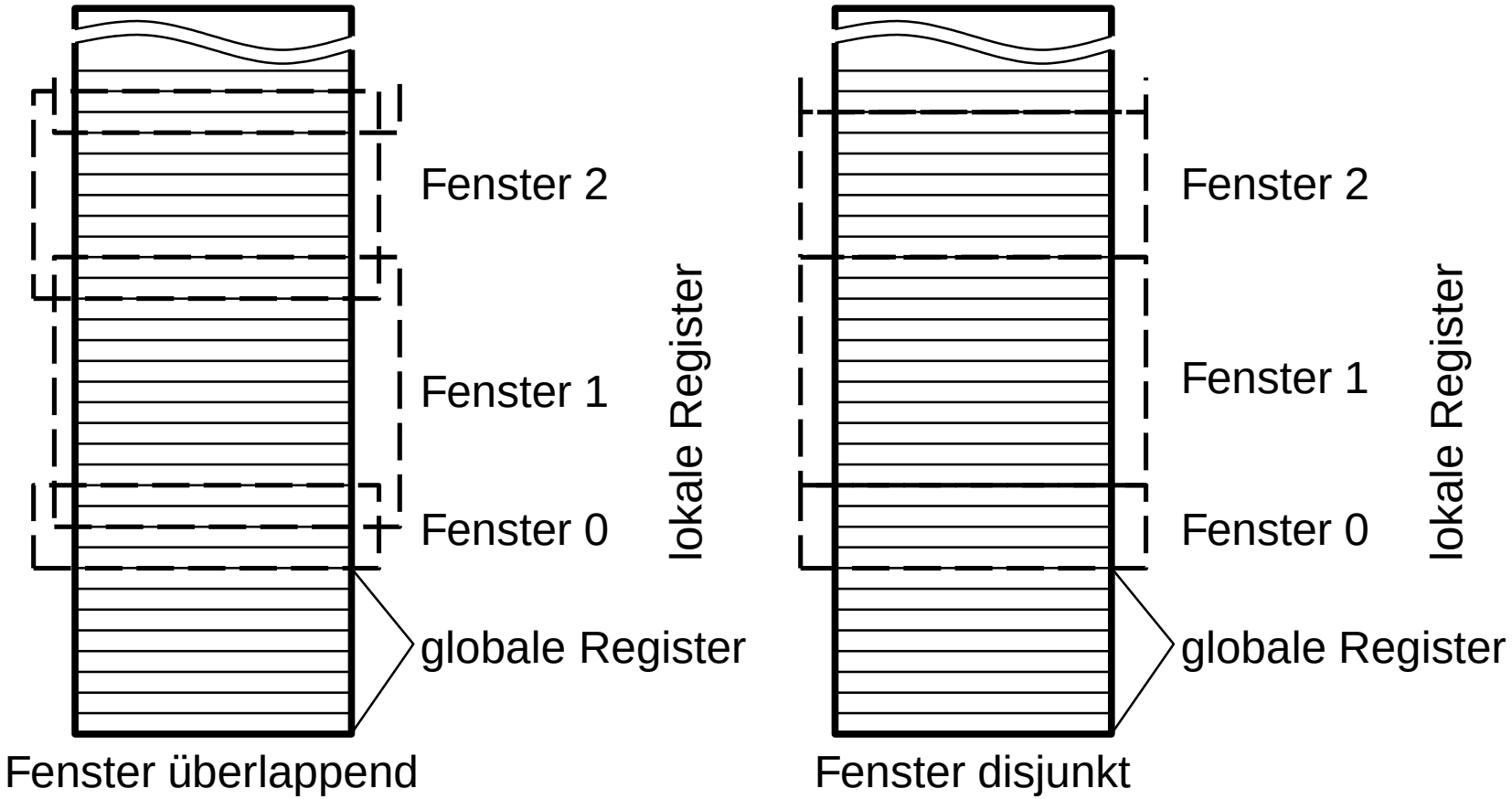
Schattenregister (*shadow register*):
Nutzung eines weiteren Registersatzes im Hintergrund (z.B. für spekulative Befehlsabarbeitung von Programmverzweigungen).

Registerumbenennung (*register renaming*):
Umbenennen von Registern ohne den Inhalt zu transportieren (z.B. bei Ausführungs- oder Datenkonflikten in der Verarbeitungseinheit).

Registerfenster – Fenster fester Größe



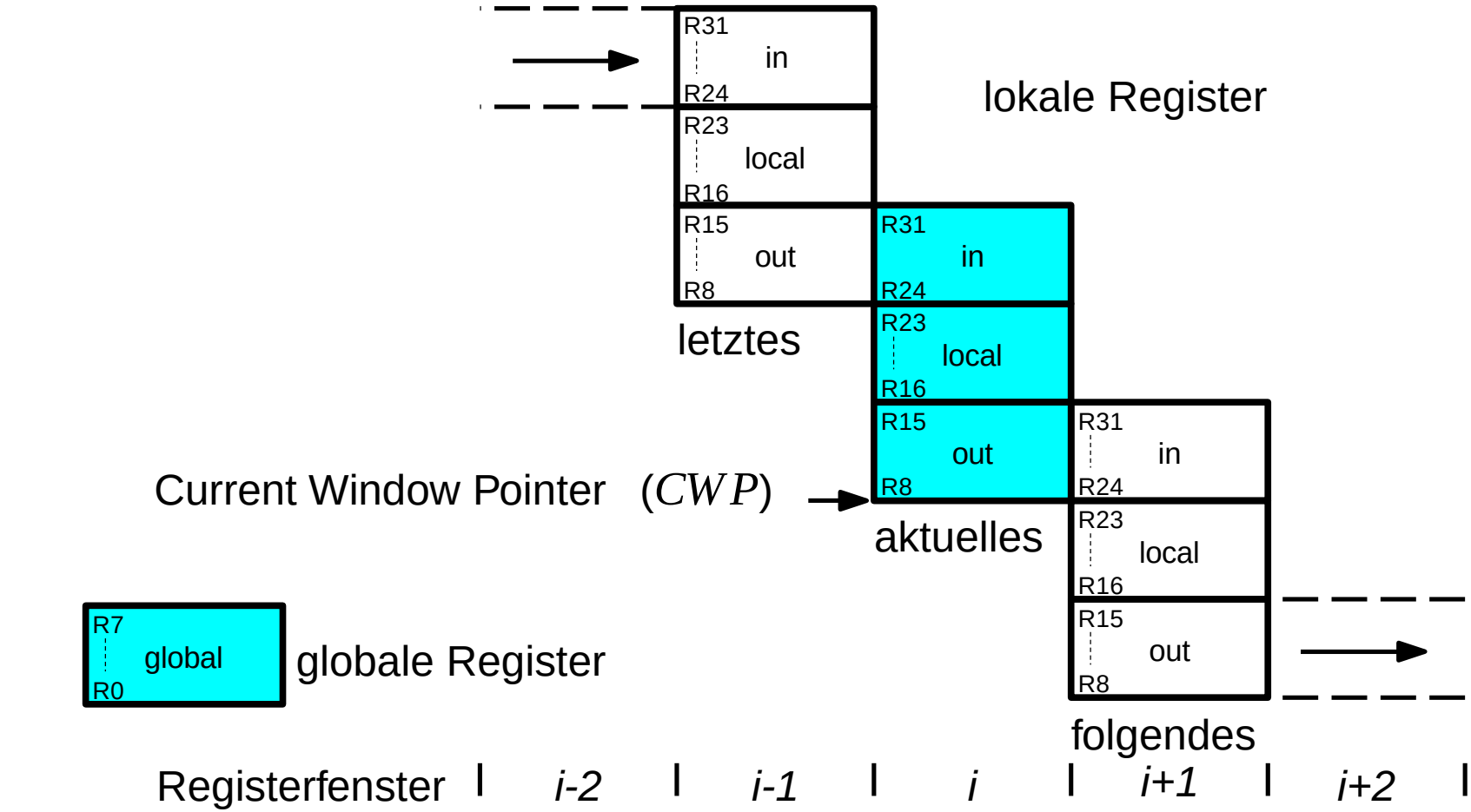
Registerfenster – Fenster variabler Größe



Die allgemeinen Register der UltraSPARC II V9

Register	Andere Bezeichnung	Funktion
R0	G0	Fest auf 0 verdrahtet; Speicher-operationen darauf werden ignoriert.
R1 - R7	G1 - G7	Hält globale Variablen
R8 - R13	O0 - O5	Hält Parameter für die aufzurufende Prozedur
R14	SP	Stapelzeiger
R15	O7	Arbeitsregister
R16 - R23	L0 - L7	Hält lokale Variablen für die laufende Prozedur
R24 - R29	I0 - I5	Hält ankommende Parameter
R30	FP	Zeiger auf die Basis des laufenden Stapelrahmens
R31	I7	Hält die Rückgabeadresse für die laufende Prozedur

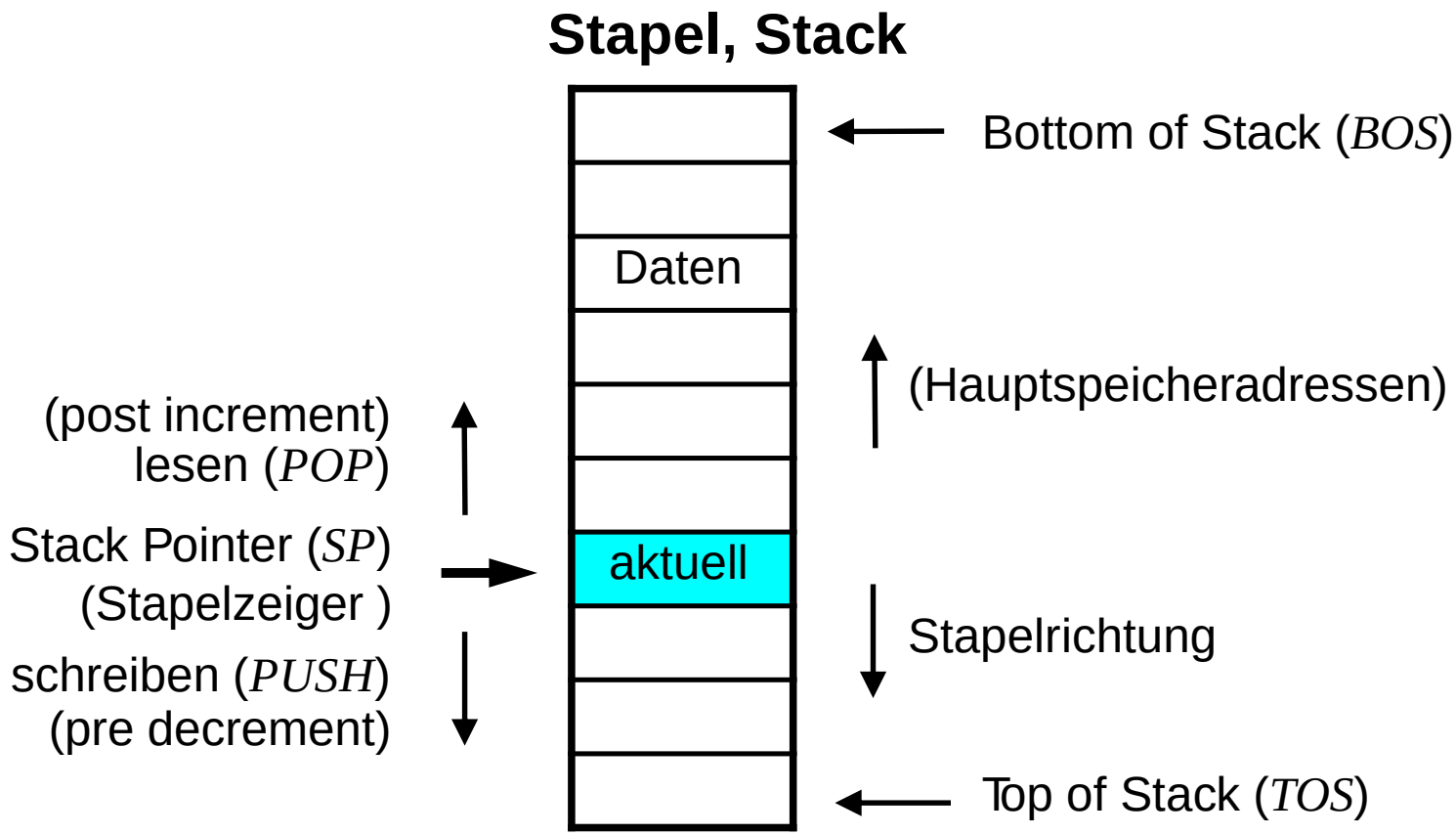
Registerfenster mit konstanter Größe (UltraSPARC II V9)



Stapelspeicher, Stack, Kellerspeicher

- SAM-Speicher (*Serial Addressable Memory*) nach dem LIFO-Prinzip (Last In First Out). Als Hardware-Stack innerhalb der CPU oder als Software-Stack im Hauptspeicher realisierbar. Der Stack, Stapel, wächst meistens nach unten.
- Der Stapelzeiger (*Stack Pointer*, SP) zeigt immer auf das Stapelende, letzte aktuelle Daten. Die Daten im Stack können nicht direkt adressiert werden.
- $PUSH$ – Dekrementieren (erniedrigen) des Stack Pointers und Speichern der Daten auf den Stack (*pre decrement*).
- POP – Lesen der Daten vom Stack und anschließendes Inkrementieren (erhö hen) des Stack Pointers (*post increment*).
- Nutzung des Stack zur effektiven Abspeicherung bzw. Zwischenspeicherung von Daten: Prozessorstatus, Unterprogrammparameter, rekursive Programme, Unterbrechungsroutinen, . . .
- Stack-Typen: system stack, data stack, user stack, programm stack, . . .

Verwaltung des Stapelspeichers



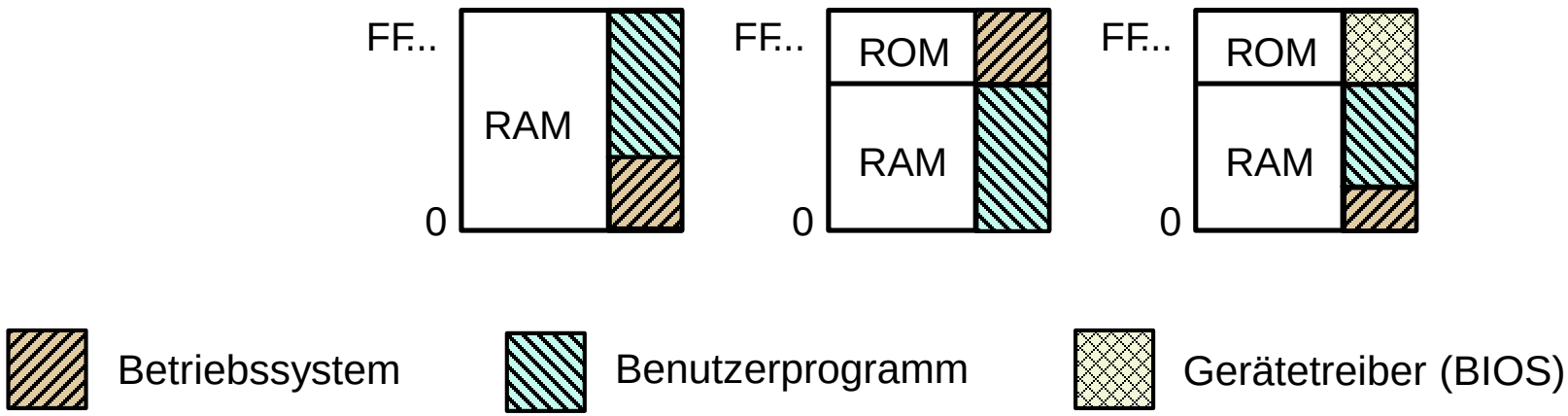
Cache-Speicher

- CAM-Speicher (*Content Addressable Memory*) als Pufferspeicher zur Überbrückung bzw. Anpassung stark unterschiedlicher Zugriffszeiten (z.B. Prozessorregister – Hauptspeicher – Festplatte).
- Der Cache arbeitet inhaltsorientiert, ist nicht direkt adressierbar. Der Vergleich mit dem Inhalt kann auch maskiert erfolgen (Ausblenden einzelner Bits). Ein Cache-Zugriff ist nicht immer eindeutig.
- Im Cache werden nur Kopien der aktuellen Speicherinhalte der darunter liegenden Hierarchieebene abgebildet. Die aktuellen Daten einer Hierarchieebene befinden sich auch immer in allen darunter liegenden.
- Zur Überbrückung sehr großer Differenzen in der Zugriffszeit bzw. Im Datendurchsatz können auch mehrere Caches hintereinander geschaltet werden (*Primary Cache*, *Secondary Cache*, übliche Bezeichnung L1, L2, ... L*n*).
- Der Cache-Speicher kann sich mit auf dem CPU-Chip befinden (*On-Chip Cache*, oft nur L1) oder extern (*Off-Chip Cache*).
- In Multiprozessorsystemen teilen sich u.U. mehrere CPUs einen Cache (z.B. L3, L4)

Speicherzugriff - Einprogrammbetrieb

- nur ein Programm, Nutzer möglich (*single user mode*)
- die maximale Programmgröße wird durch den verfügbaren physischen Speicher begrenzt
- Laden und Ausführen des Programms auf eine bestimmte definierte Adresse

➢ Organisation des Ladens und Abarbeitens durch den Programmierer/Operator



Überlagerungsstrukturen (Overlays)

- Das Programm ist größer als der physische Speicher
- Zerlegung des Programms in Teile, Overlays
- Abspeicherung der Overlays auf dem Sekundärspeicher (Festplatte)
- Laden und Ausführen der ersten Überlagerung, des Hauptprogramms
- Nachladen der weiter benötigten Overlays
- (Überschreiben alter Overlays) entsprechend Overlaytabelle

➢ Der gesammte Overlayprozess wird durch den Programmierer, das Nutzerprogramm gesteuert und verwaltet

Speicherzugriff - Mehrprogrammbetrieb

Realisierung von Mehrprogrammsystemen

- Unterteilung der Abarbeitungszeit in Zeitscheiben (Timesharing-Systeme)
- Unterteilung des Hauptspeichers in Bereiche (Partitionierung)
- ➢ Abarbeitung mehrerer Programme (Prozesse) gleichzeitig, parallel (innerhalb einer Zeitscheibe nur je ein Prozess)
- Einbenutzersysteme (*single user mode*) ein Nutzer mit mehreren Programmen
- Mehrbenutzersysteme (*multi user mode*) mehrere Nutzer mit vielen Programmen
- zu den Nutzerprozessen parallele Abarbeitung von Betriebssystemprozessen, Treibern, E/A ...

Swapping

Unter Swapping (Ein-/Auslagerung) wird bei Timesharing-Systemen das Verschieben ganzer Prozesse (Programme) zwischen primären und sekundären Speicher (Hauptspeicher-Festplatte) verstanden.

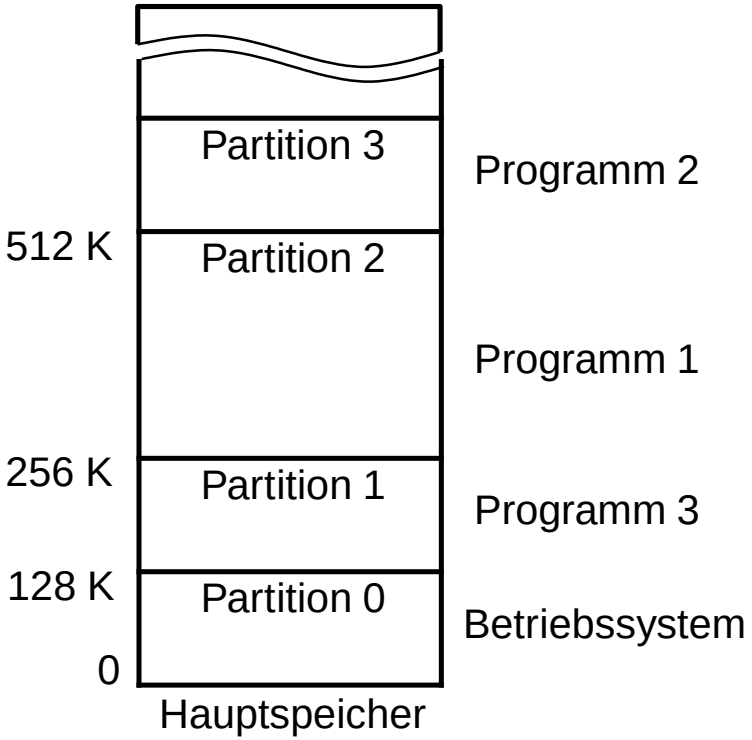
- Definition eines Swap-Bereiches auf der Festplatte
- Allokation von Swap-Partitionen im Swap-Bereich (analog Hauptspeicherpartitionierung)
- Ein- und Auslagerung von Prozessen als Transfer zwischen Hauptspeicher- und Swap-Partitionen

Ursachen, Notwendigkeit für Swapping:

- es gibt mehr Prozesse als mögliche Speicherpartitionen (zu kleiner Hauptspeicher)
- Prozesse mit langen Wartezeiten (z.B. auf E/A-Geräte)
- inaktive Prozesse

Mehrprogrammbetrieb mit fester/variabler Speicherpartitionierung

(IBM-360-OS/MFT Multiprogramming with a Fixed number of Tasks, 1966)
(IBM-OS/MVT Multiprogramming with a Variable number of Tasks, 1971)



Relokation (fest):

- Adresstransformation (Verschiebung entsprechend Partitionsanfang) beim Laden der Programme oder in Hardware (Basisregister)

Relokation (variabel):

- Nutzung variabler Partitionen (feste Partitionen sind uneffektiv bzgl. Speicherplatzausnutzung)
- dynamische Speicherallokation, durch Prozesse gesteuert
- Speicherallokation der Prozesse ändert sich ständig

Fragmentation

Interne Fragmentation:
Programmgröße und Partitionsgröße stimmen nicht überein
→ Partitionen fester Größe enthalten Freiräume, Löcher

Externe Fragmentation:
Prozessgröße ändert sich während der Abarbeitung (wachsen, schrumpfen),
Prozesse werden ein- und ausgelagert
→ zwischen Partitionen variabler Größe entstehen Freiräume, Löcher

Defragmentierung (Speicherverdichtung):
Verschiebung aller Prozesse im Speicher zur Vermeidung
von externer Fragmentation (zeitaufwendig)

Virtueller Speicher

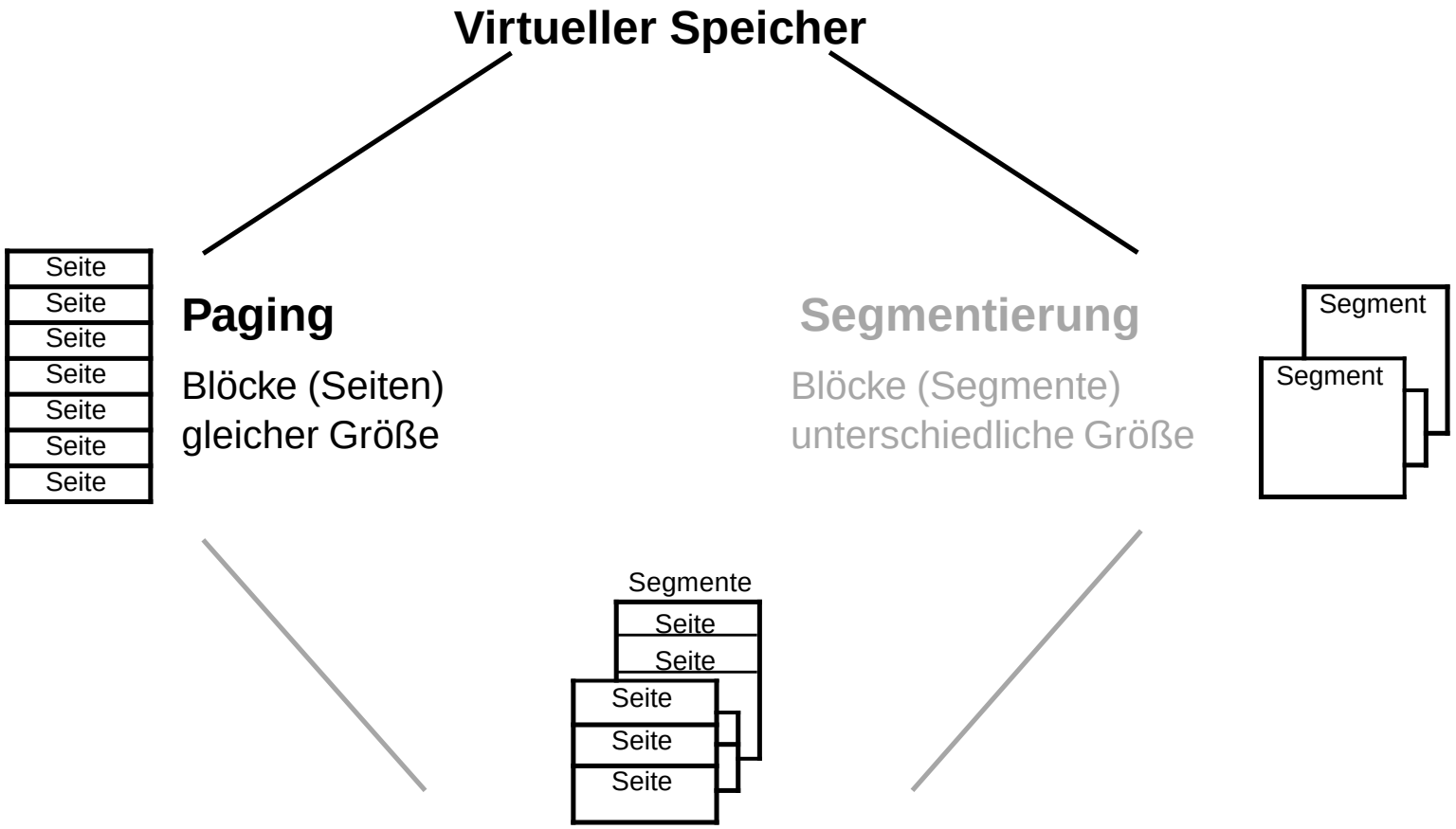
Zielstellung:

- Vergrößerung des nutzbaren Adressraumes weit über den des technisch verfügbaren Hauptspeichers hinaus
- Unterstützung mehrerer getrennter Adressräume
- Realisierung ”beliebig“ teilbarer und nutzbarer Adressräume
- Trennung von einzelnen Prozessen bzgl. Zugriffsrechte, Schutzmechanismen
- gemeinsame Nutzung von Prozessen bzw. Speicherbereichen
- Effektive Einbeziehung des Sekundärspeichers (Festplatte), kurze Lade- und Speicherzeiten
- Entlastung des Programmierers von aufwendiger Adressrechnung, Overlayprozess, Partitionierung, Verwaltungsaufgaben

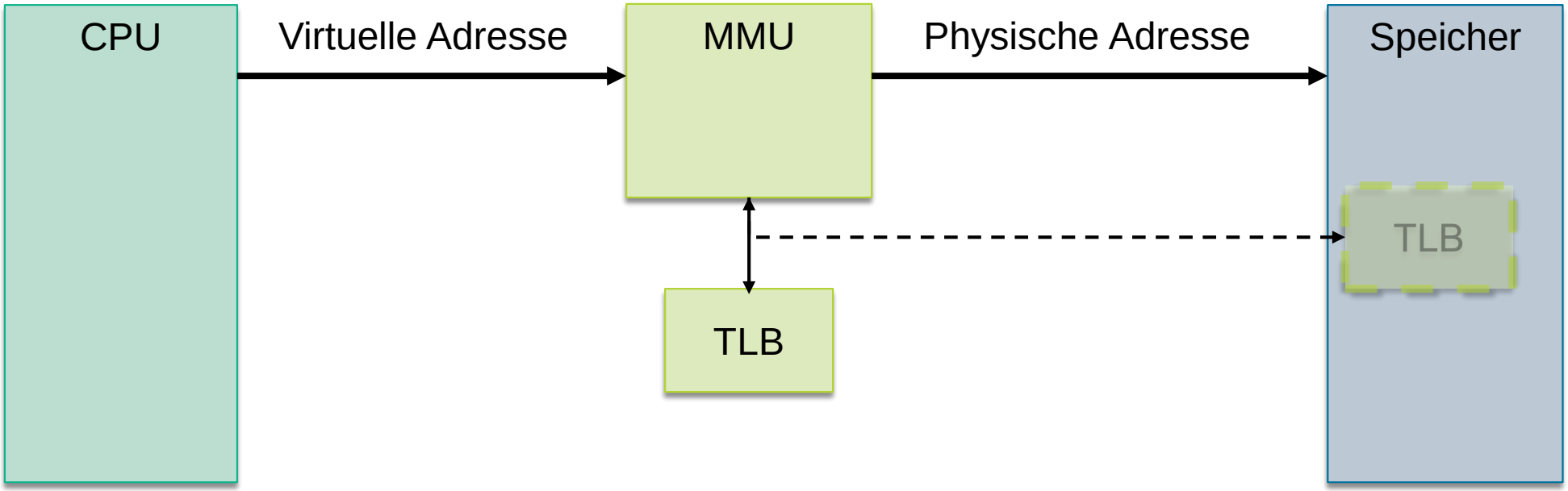
Grundkonzept des Virtuellen Speicher

- Trennung von Adressraum (*address space*) und Speicheradressen (physischer Speicher)
- Einführung eines oder mehrerer beliebig großer virtueller Adressräume
- Unterteilung des virtuellen und des physischen Adressraumes in Blöcke gleicher oder variabler Größe
- Adressumsetzung (*address translation*) zwischen virtuellen und physischen Adressen durch Tabellen
- Nutzung des Sekundärspeichers für Ein- und Auslagerungen
- Automatische Realisierung der gesamten Prozessaufteilung, Speicherzuordnung, Adresstransformation, Umlagerungen, durch die Hardware und das Betriebssystem, für den Nutzer völlig transparent

Virtueller Speicher



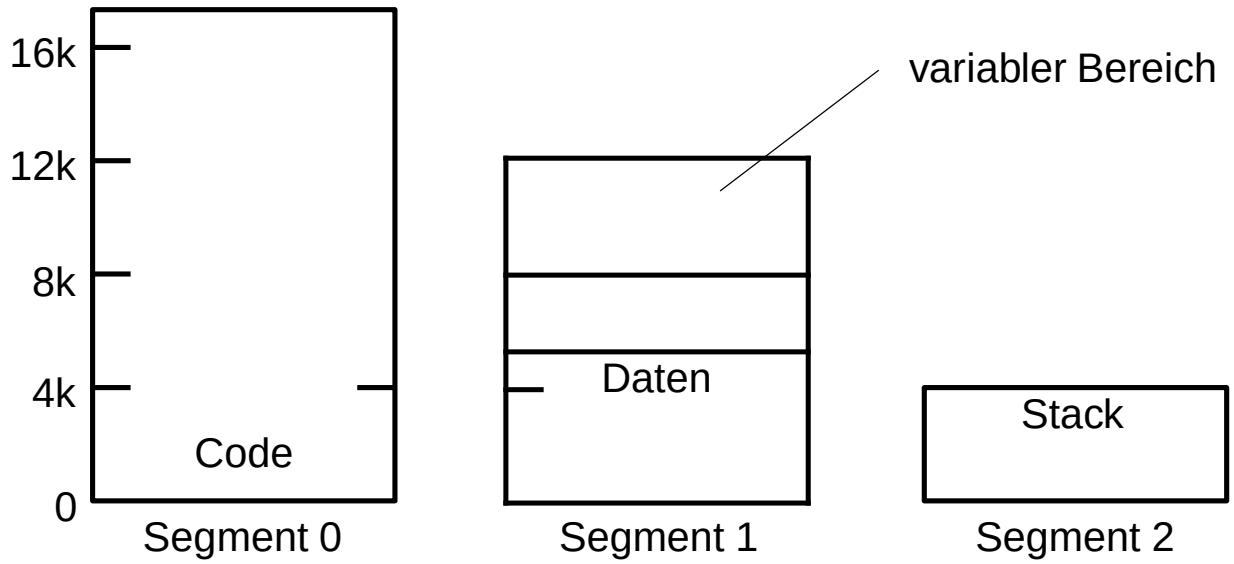
Zusammenwirken von CPU und MMU zur Berechnung physischer Adressen



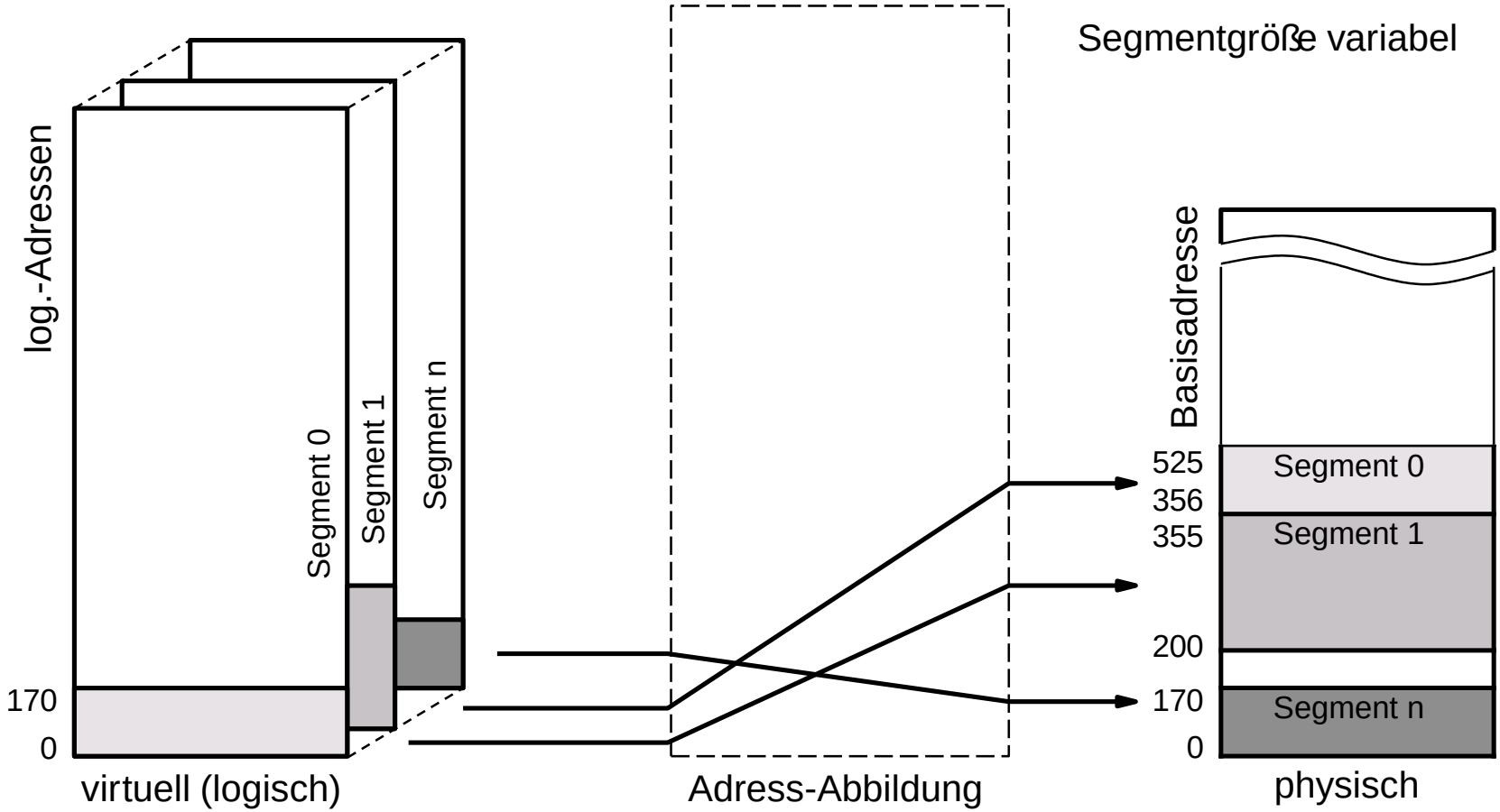
Translation Lookaside-Buffer (TLB): Cache für Adressumsetzung, kann auch selbst im Hauptspeicher liegen

Segmentierung

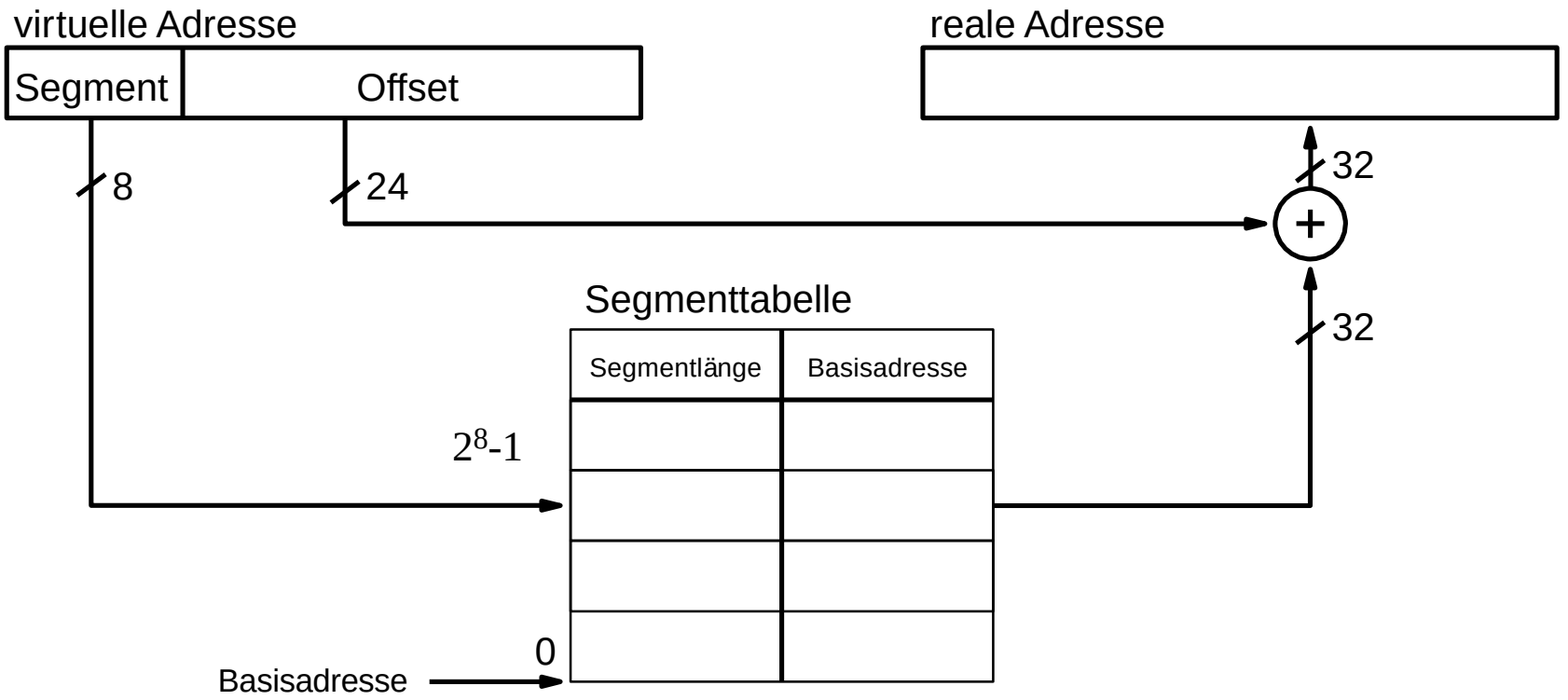
1. unterschiedliche Inhalte
2. dynamische Segmentlängenänderung
3. verschiedene Segmentlängen



Segmentierung

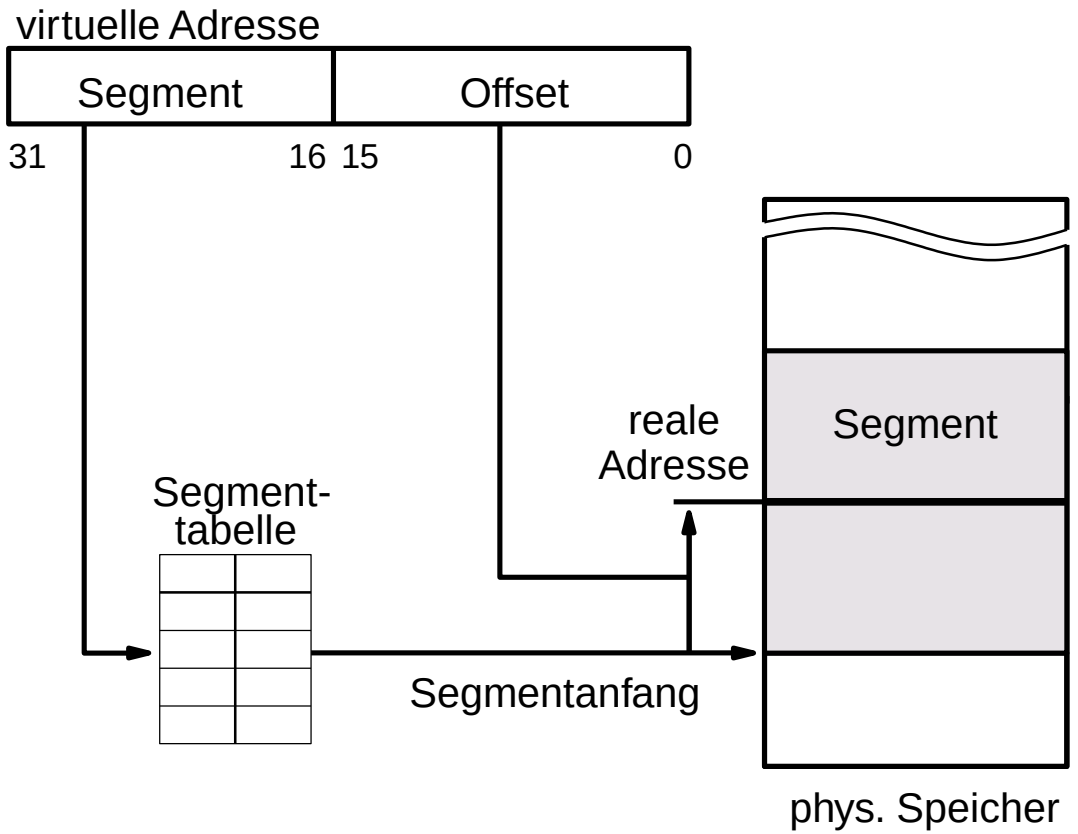


Adressabbildung bei Segmentierung

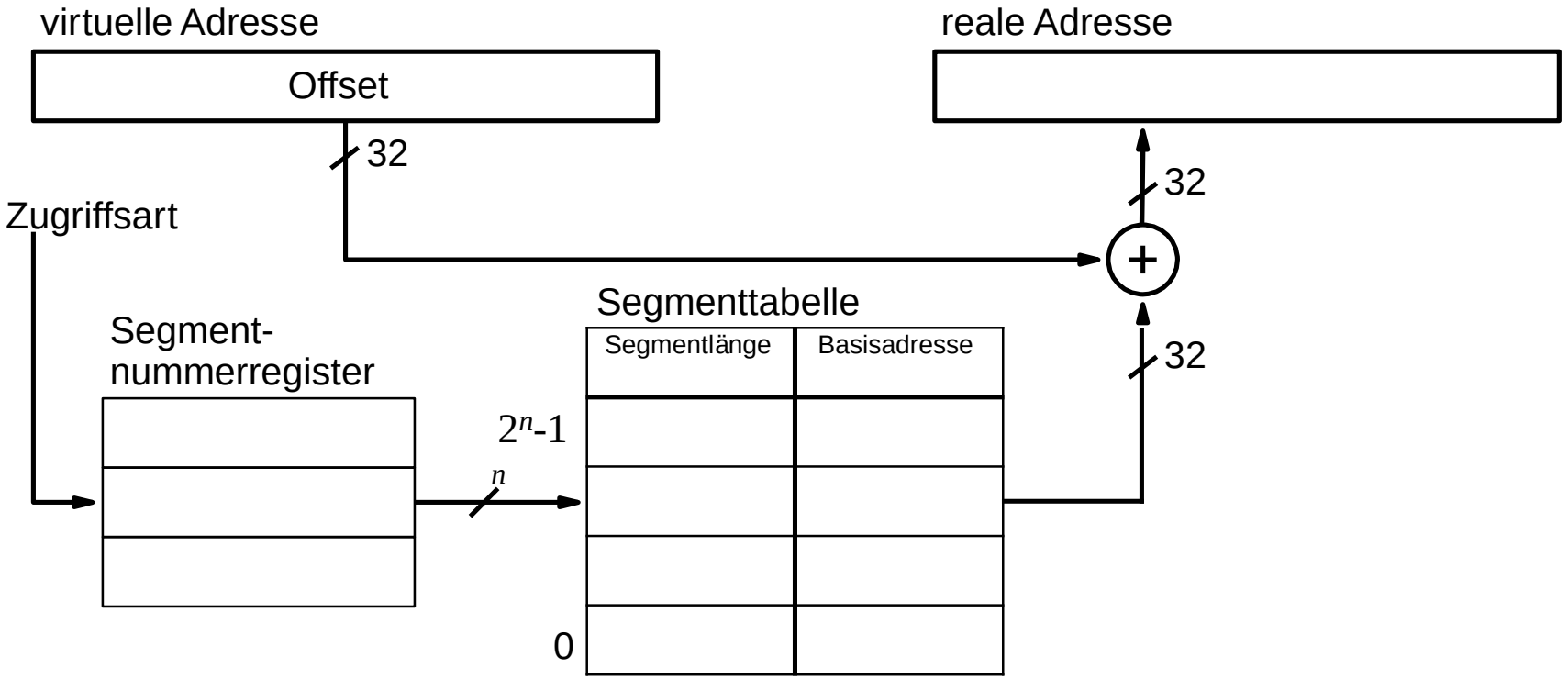


Segmenttabelle enthält Segmentlänge und Basisadresse der Segmente die sich im phys. Speicher befinden (Segment Basisadresse).

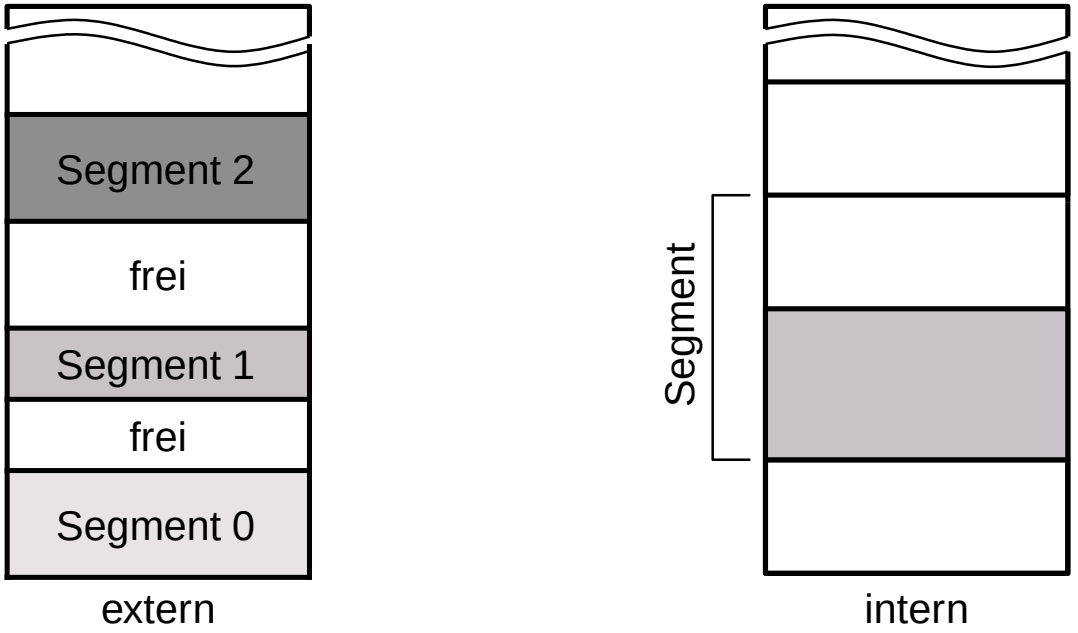
Darstellung der Segmentadressierung



Adressabbildung bei Segmentierung mit Segmentnummerregister



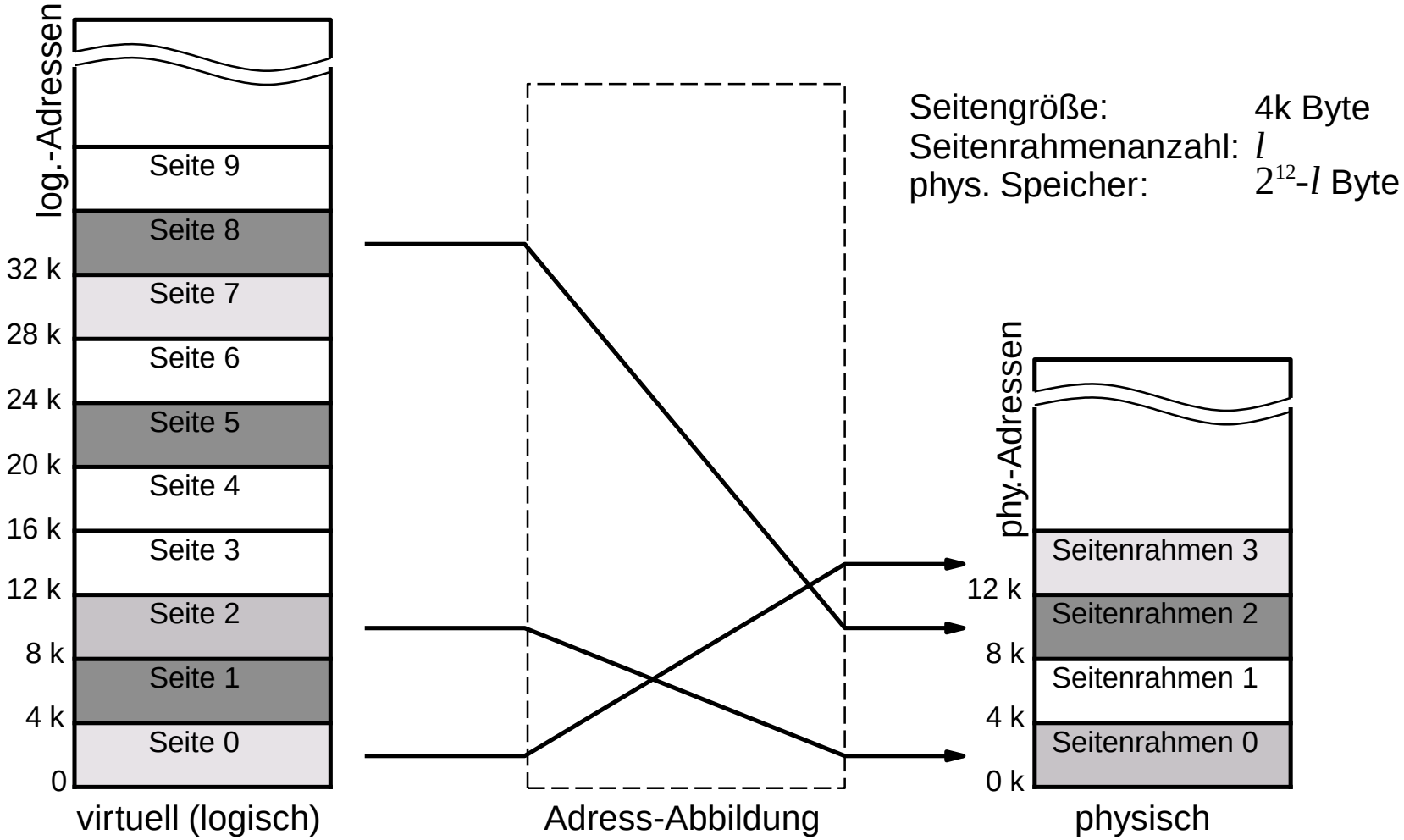
Fragmentierung bei Segmentierung



Seitenadressierung, Paging (1)

- Zuordnung eines für jeden Prozess gesonderten virtuellen Adressraumes (*virtual address space*) mit der Adresse 0 beginnend
- Unterteilung des virtuellen Adressraumes in Blöcke gleicher Größe (Seiten, *Pages*)
- Die Seitengröße entspricht immer einer Potenz von 2 (z.B. 4 Kbyte, 4 Mbyte)
- Unterteilung des physischen Adressraumes in Blöcke der gleichen Größe wie beim virtuellen Adressraum (Seitenrahmen, *Page Frames*)
- Die Seitenrahmen des physischen Adressraumes können jeweils genau eine Seite des virtuellen aufnehmen

Paging



Seitengröße: 4k Byte
Seitenrahmenanzahl: l
phys. Speicher: 2^{12}-l Byte

Adressabbildung mit Seitentabelle

1. Suchen der virtuellen Seite in der Seitentabelle
   → Adressierung der Seitentabelle mit der Seitennummer
2. Herauslesen der Seitenrahmennummer aus der Seitentabelle
3. Zusammenfügen von Seitenrahmennummer und Seitenoffset ergibt die physische Adresse im Speicher

Befindet sich die gesuchte Seite nicht in einem Seitenrahmen des physischen Speichers (Present/Absent-Bit ist 0), so ist diese Seite durch das Betriebssystem nachzuladen und die Seitentabelle zu aktualisieren.

virtuell	Seitentabelle	physisch
Seitennummer + Offset		Seiterahmennummer + Offset

Ersetzen der Seitenummer duch die Seitenrahmennummer

Seitenadressierung, Paging (2)

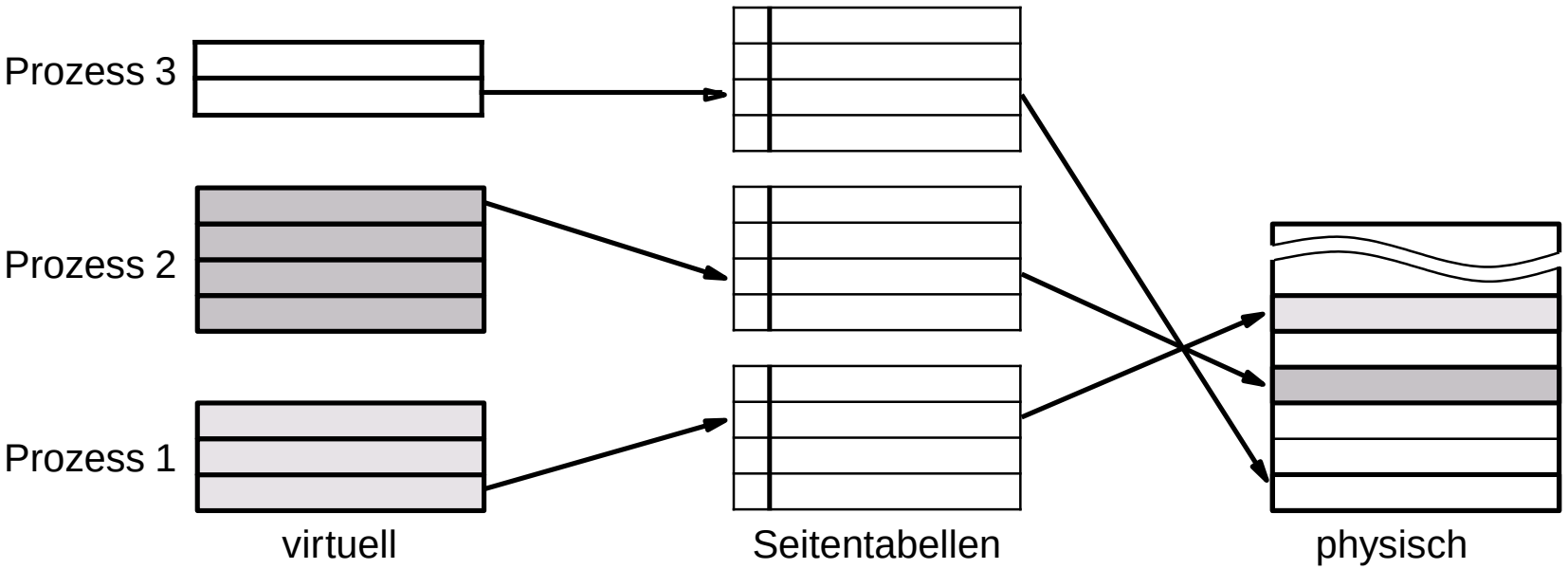
- Unterteilung der virtuellen und physischen Adressen in Seitennummer (*page number*) bzw. Seitenrahmennummer und Seitenoffset (*page offset*)
- Realisierung der Abbildung virtuelle → physische Adresse (Address Translation) durch eine Seitentabelle (*page table*)
- Die Adressabbildung virtuell physisch erfolgt hardwaremässig durch die Memory Managemant Unit (MMU) und wird durch das Betriebssystem unterstützt
- Die Seitenadressierung erfolgt automatisch, für den Nutzer transparent. Programmierung ohne Berücksichtigung von Paging mit ”virtuellen“ Adressen

Typische Seitentabelleneinträge

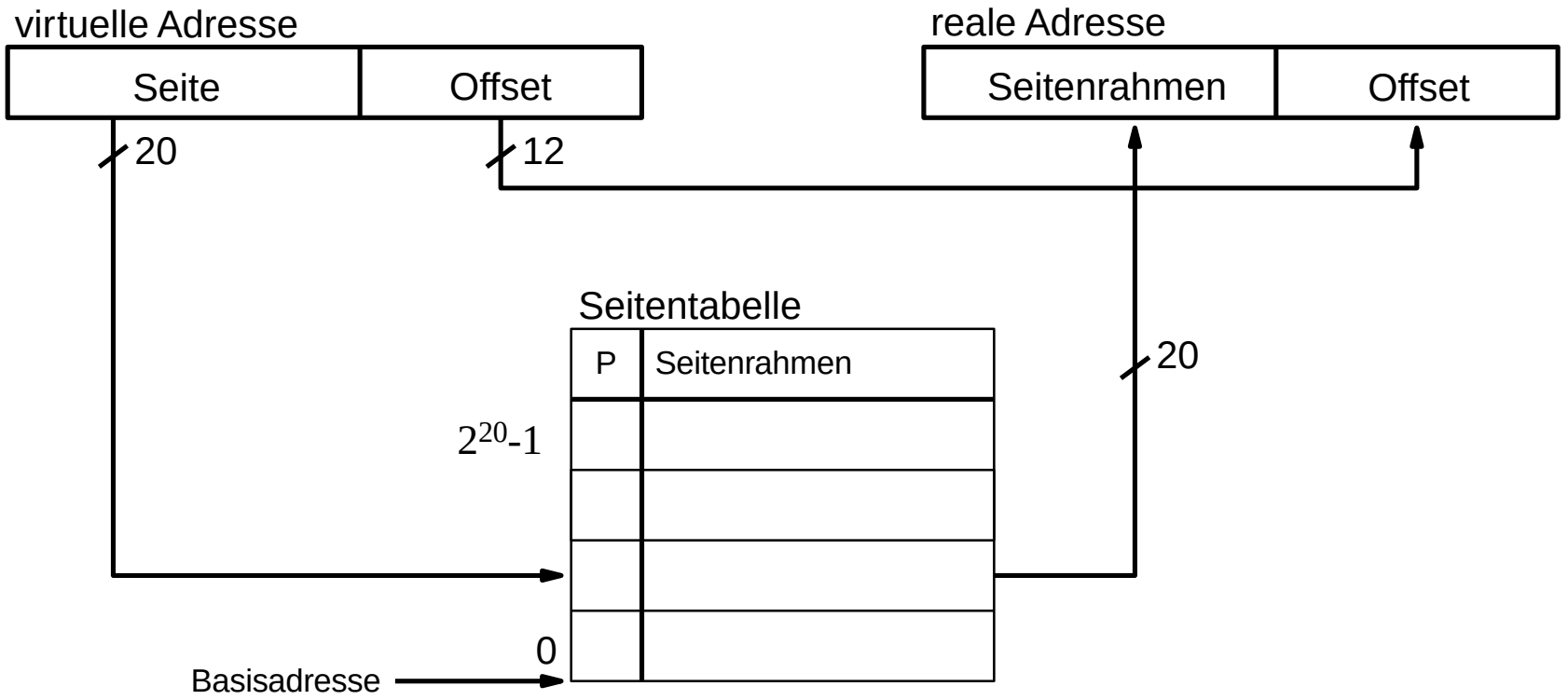
- Seitenrahmennummer (physischer Speicher)
- Verfügbarkeit (Present/Absent-Bit)
- Schutz (Privileg)
- Verändert (Dirty-Bit)
- Referenziert (Accessed)
- Caching

→ Enthält keine Information über die Lage der Seite im Sekundärspeicher (Festplatte), wird durch das Betriebssystem organisiert

Privater Adressraum pro Prozess (Nutzer)



Adressabbildung bei Paging

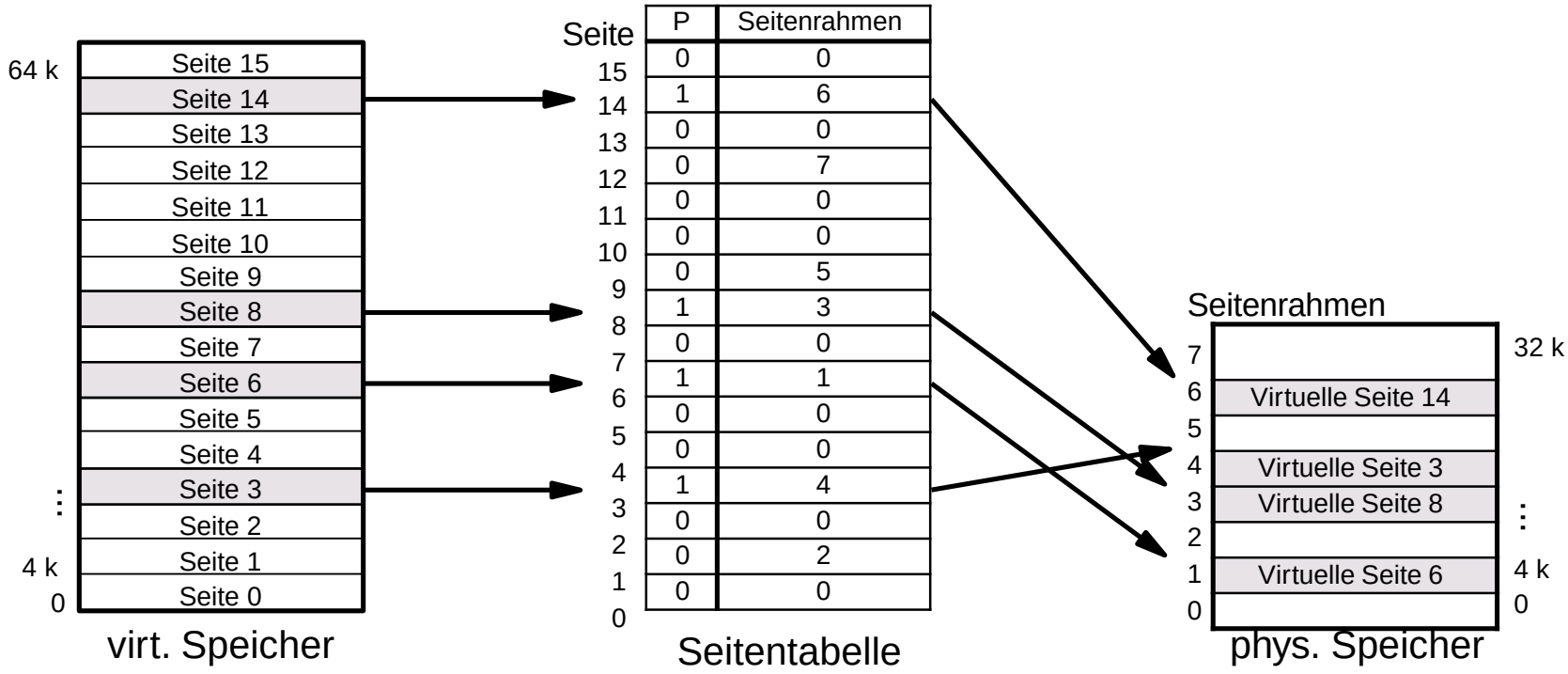


Seitentabelle enthält die Seitenrähmen der Seiten die sich im phys. Speicher befinden (Seite → Seitenrahmen).

P: Present/Absent-Bit (1 = Vorhanden im Hauptspeicher / 0 = Fehlt im Hauptspeicher)

Beispiel für Seitentabelle

Mögliche Abbildung von einen Virtuellen Speicher mit 16 Seiten auf einen Arbeitsspeicher mit 8 Seitenrahmen.



Forderungen und Größe linearer Seitentabellen

Forderungen

- Realisierung großer umfangreicher Seitentabellen für jeden Prozess getrennt, mit Schutzfunktionen, ...
- sehr schneller Adressabbildung (kleine Latenzzeiten)
- Realisierung von gemeinsamen Speicher (*Shared Memory*)

Größe

m-bit virt. Adressraum, s-bit Seitengröße, k-byte Seitentabelleneintrag
$\rightarrow (2^{m-s} \cdot k)$-byte Größe einer Seitentabelle

Beispiel

- 32-bit virt. Adressraum, 12-bit Seitengröße, 4-byte Seitentabelleneintrag
  $\rightarrow$ 4-Mbyte Seitentabellengröße für jeden Prozess
- 64-bit virt. Adressraum, 20-bit Seitengröße, 8-byte Seitentabelleneintrag

Optimierung der Seitengröße

größere Seiten

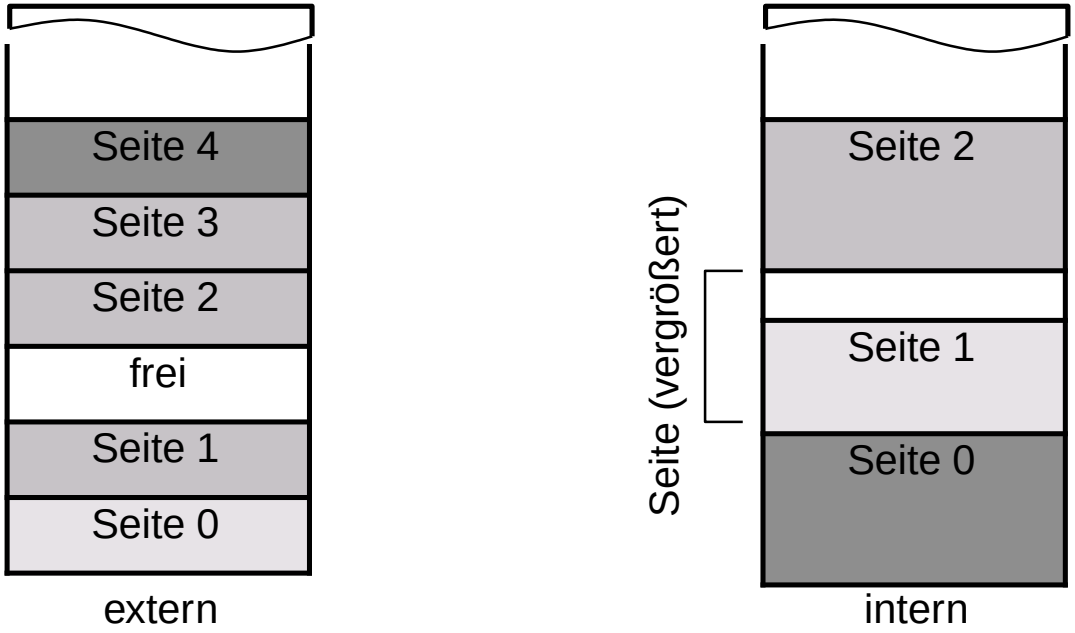
- höhere interne Fragmentation (ungenutzte Speicherbereiche innerhalb der Seiten)
- höherer Aufwand beim Nachladen von Seiten (Seitenfehler)

kleinere Seiten

- größere Seitentabellen (mehr Seitentabelleneinträge)
- größerer Adressabbildungsaufwand (aufwendigere MMU)

→ Implementierung von verschiedenen Seitengrößen

Fragmentierung bei Paging



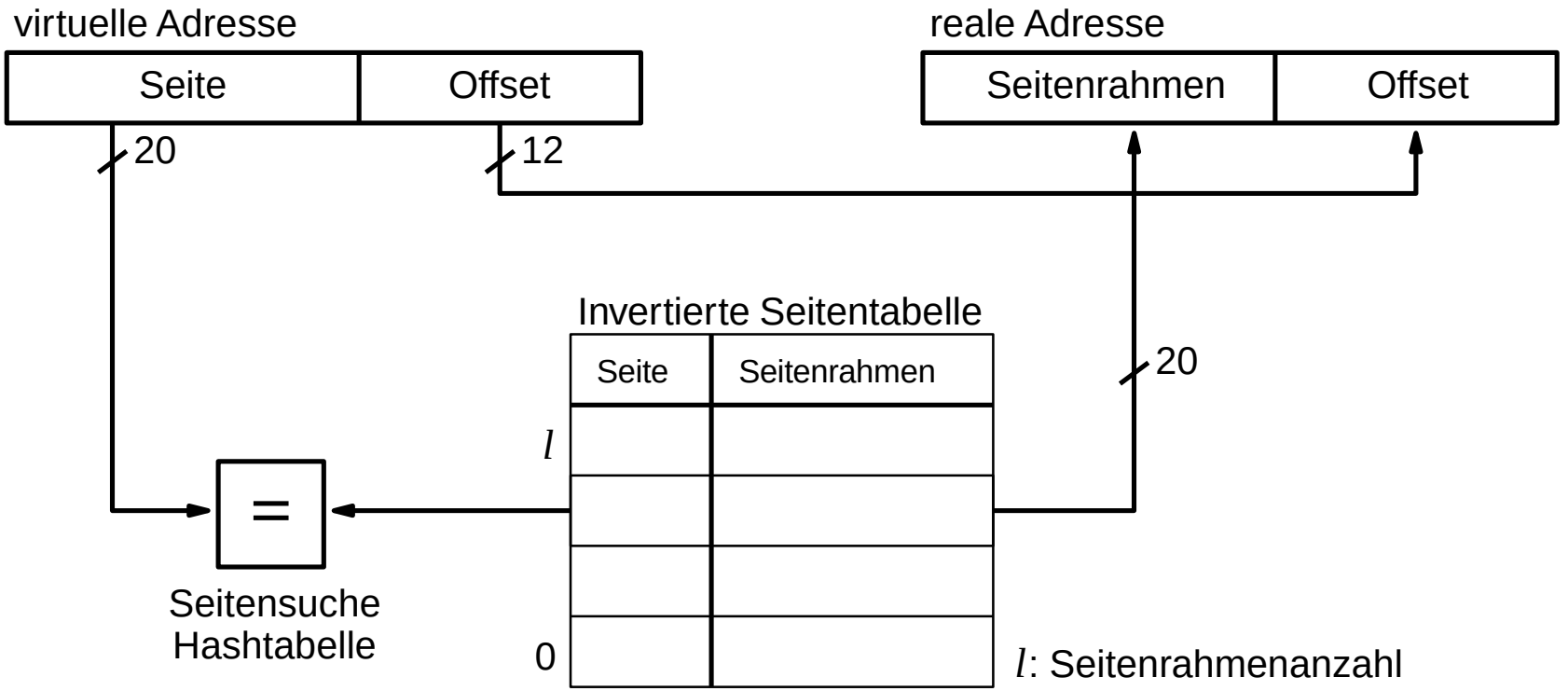
Effektivere Adressabbildung

viele und große virtuelle Adressräume → viele und große Seitentabellen (hoher Speicherbedarf und große Latenzzeiten).
Aber, nur eine kleine Menge an virtuellen Seiten werden wiklich genutzt.

Auswege

- effiziente Abspeicherung der Seitentabellen, nur genutzter Bereich
- Vergrößerung der Seiten (Verringerung der Seitenanzahl)
- mehrstufige Seitentabellen
- invertierte Seitentabellen (Mashing Methode)
- caching der Tabelleneinträge (Translation Lookaside Buffers, TLB)

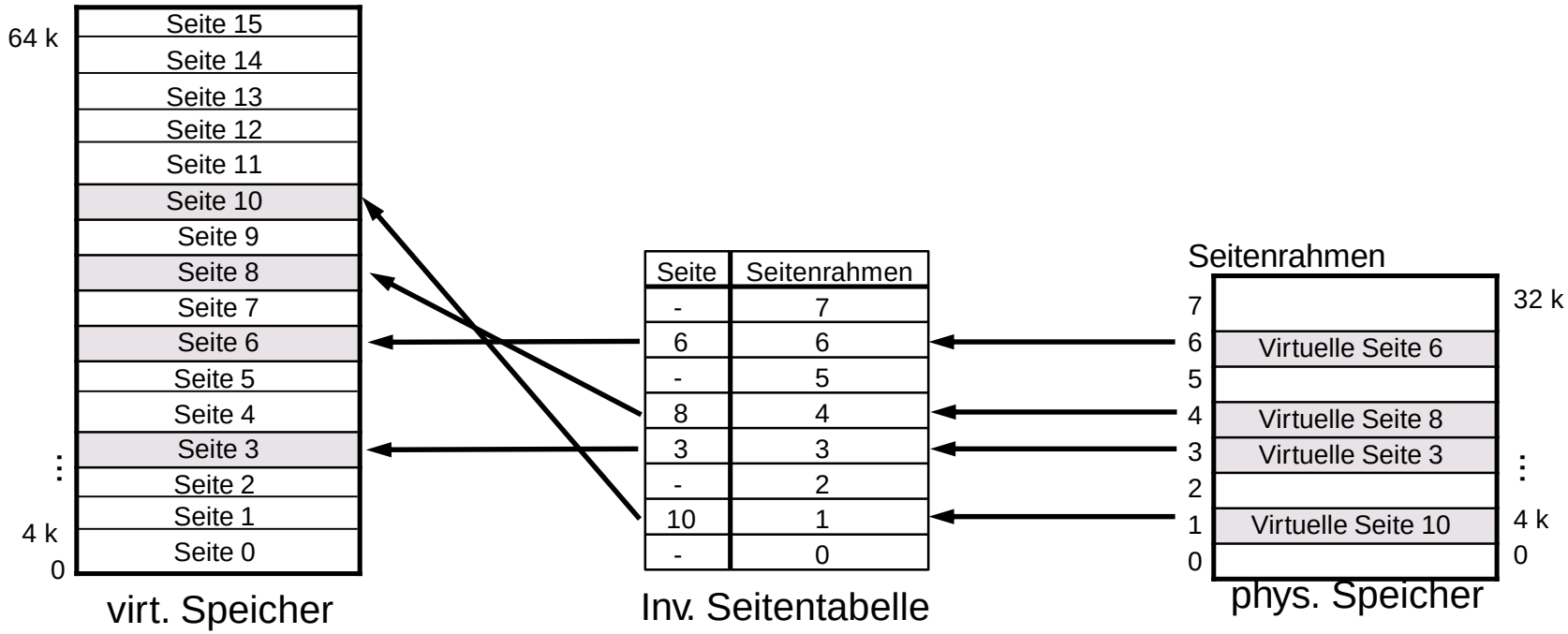
Adressabbildung mit Invertierter Seitentabelle



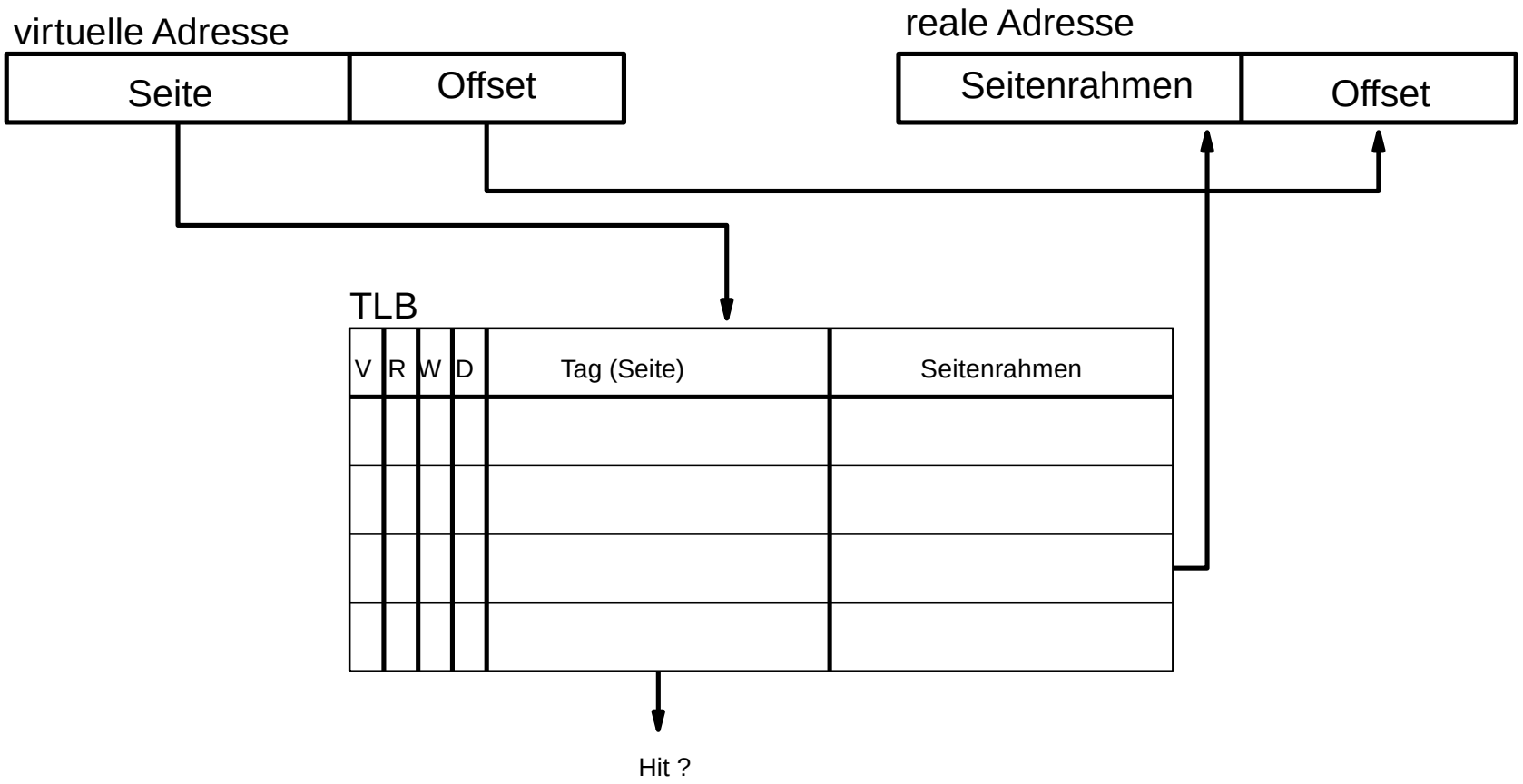
Invertierte Seitentabelle enthält nur Seiten die sich in Seitenrähmen des phys. Speicher befinden (Seitenrahmen → Seite).

Beispiel für Invertierte Seitentabelle

Mögliche Abbildung von einen Arbeitsspeicher mit 8 Seitenrahmen auf einen Virtuellen Speicher mit 16 Seiten.

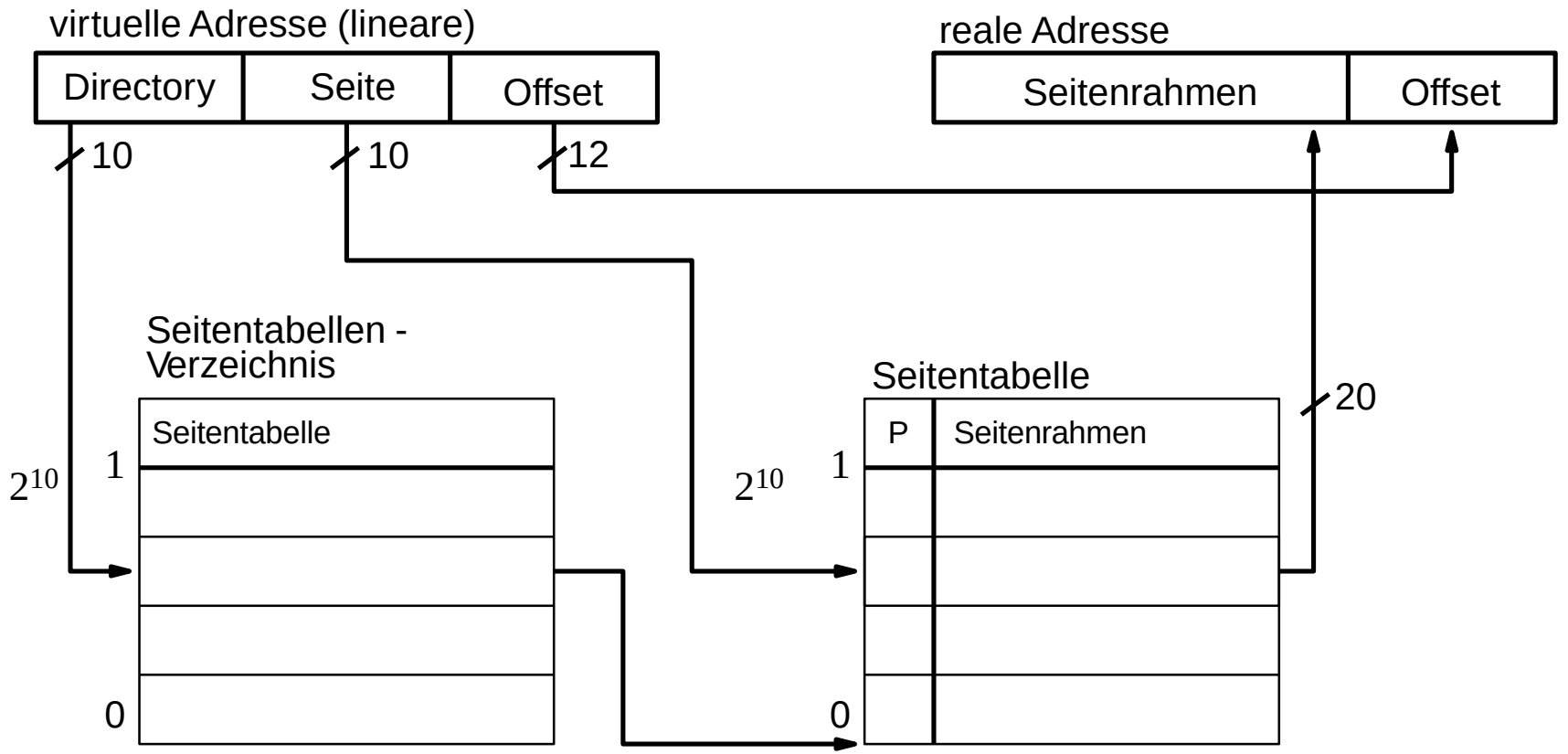


Translation Lookaside Buffers (TLB)

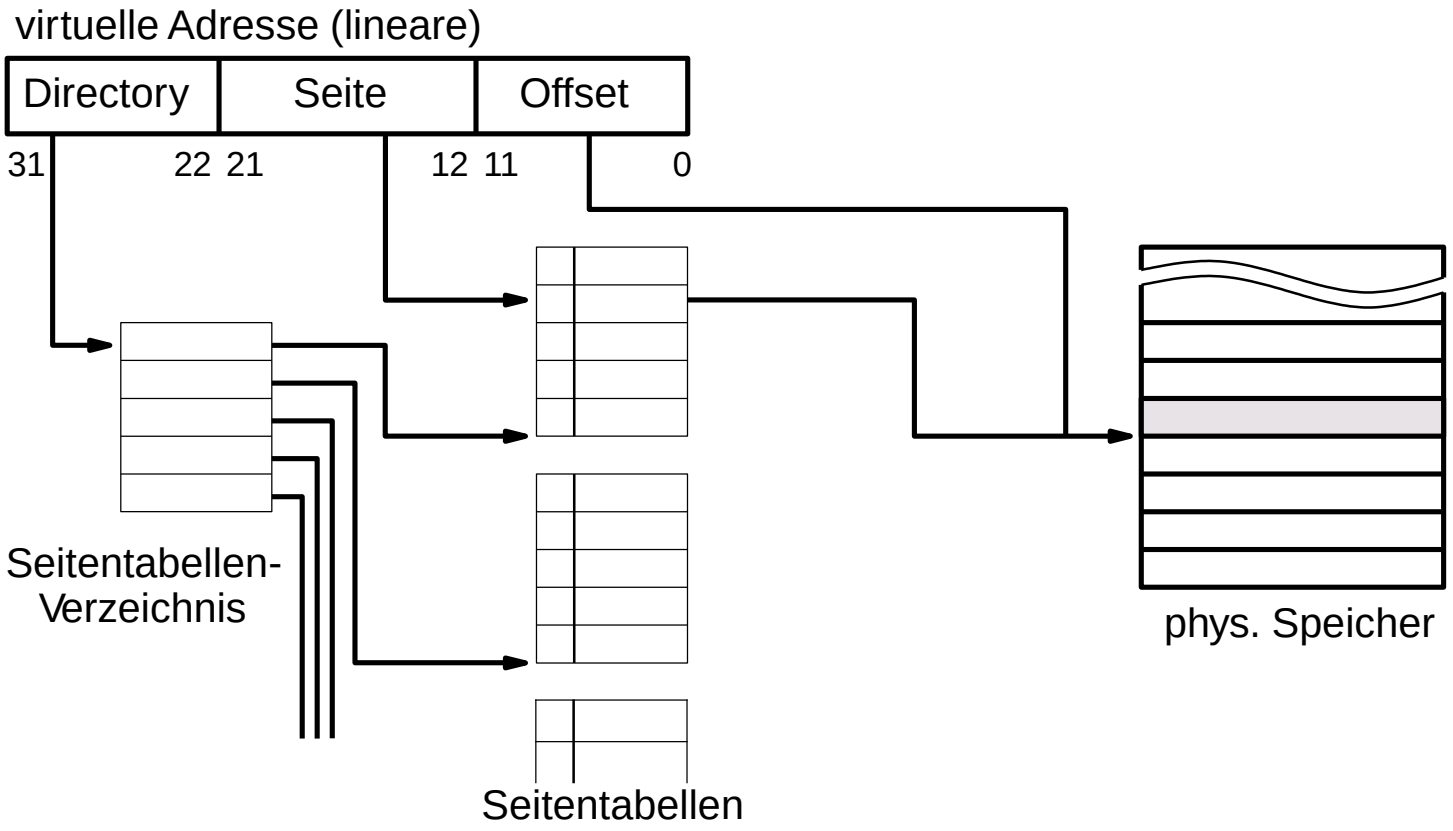


V = Valid R = Read W = Write D = Dirty

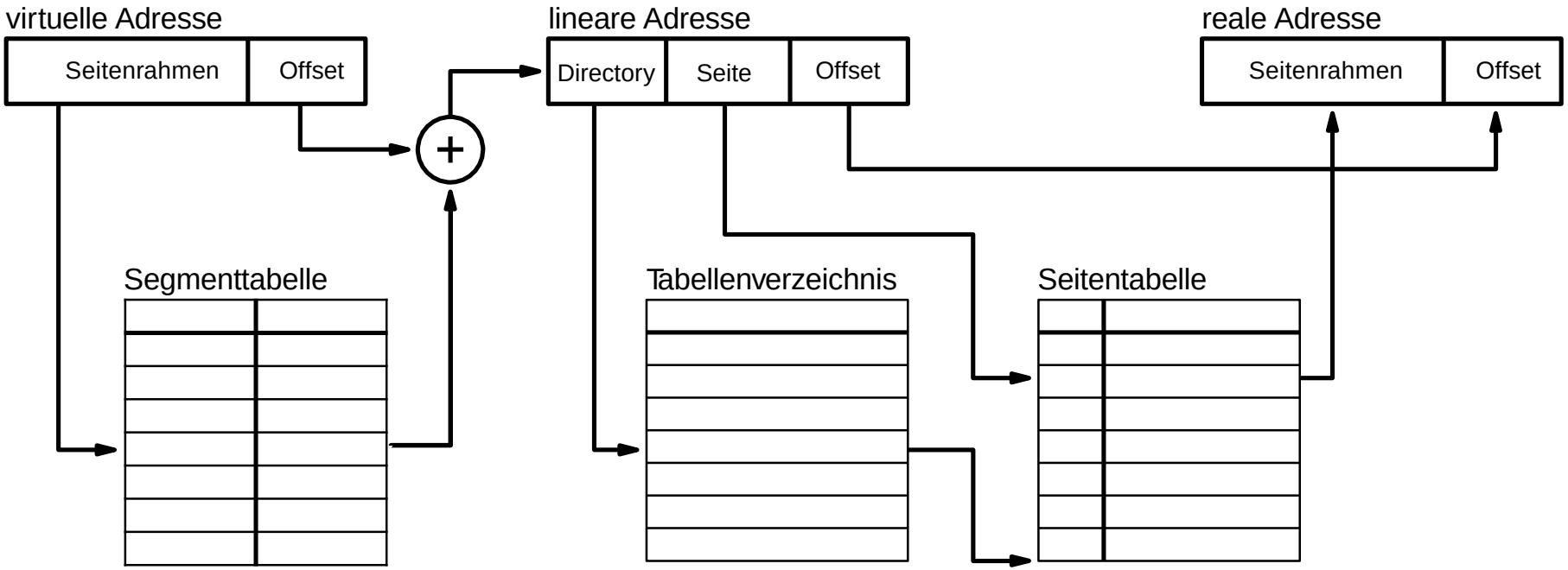
Zweistufige Seitenadressierung



Darstellung bei der Zweistufigen Seitenadressierung



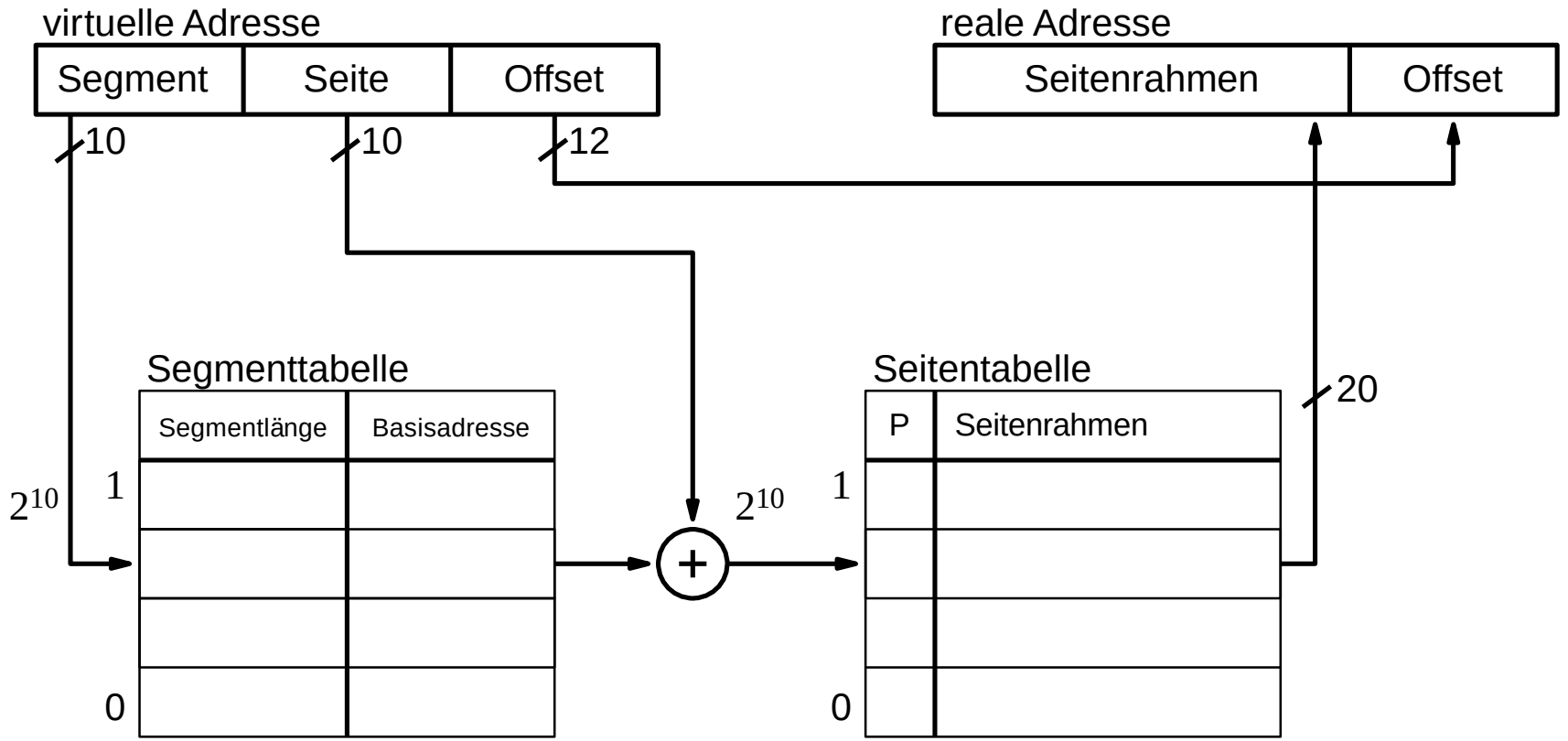
Virtuelle Speicheradressierung beim Pentium II



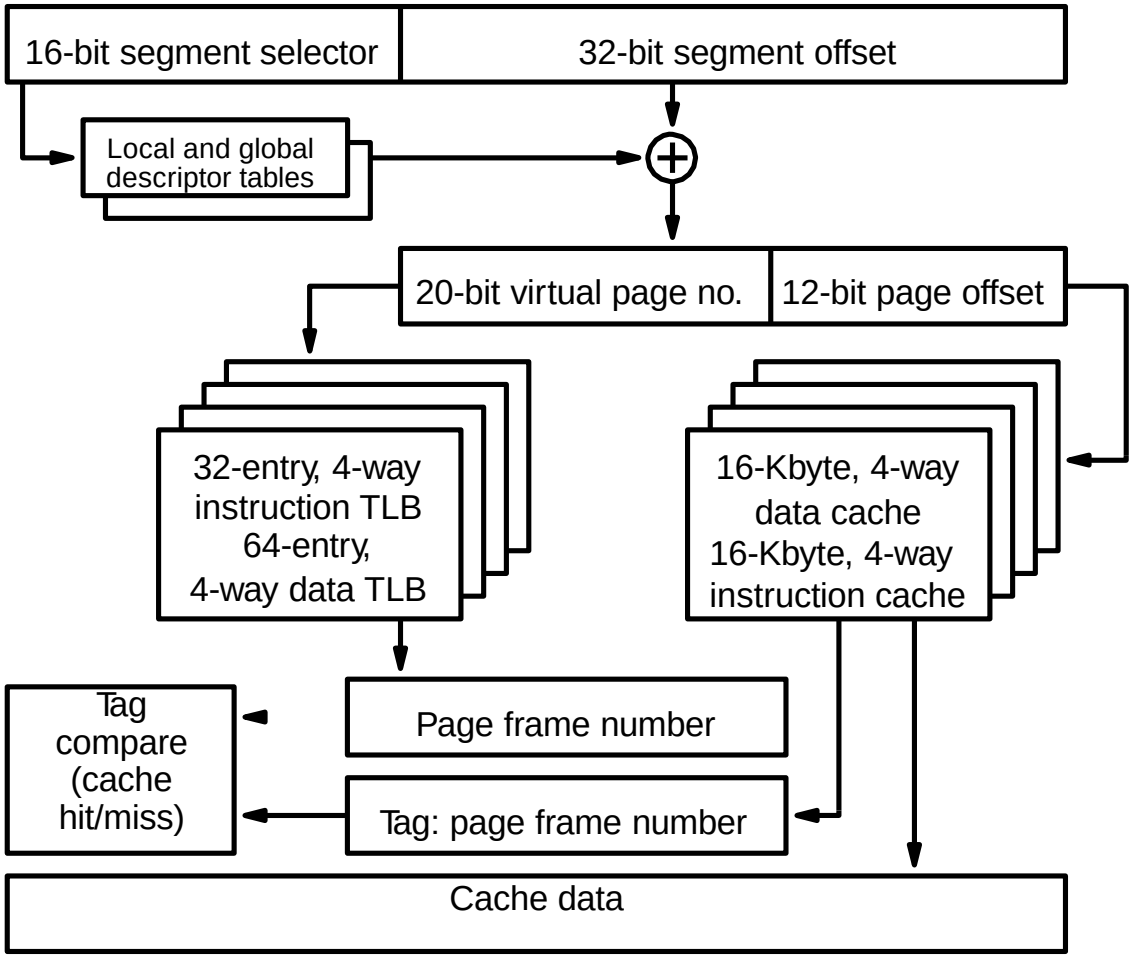
Segementierung

Zweistufige Seitenadressierung

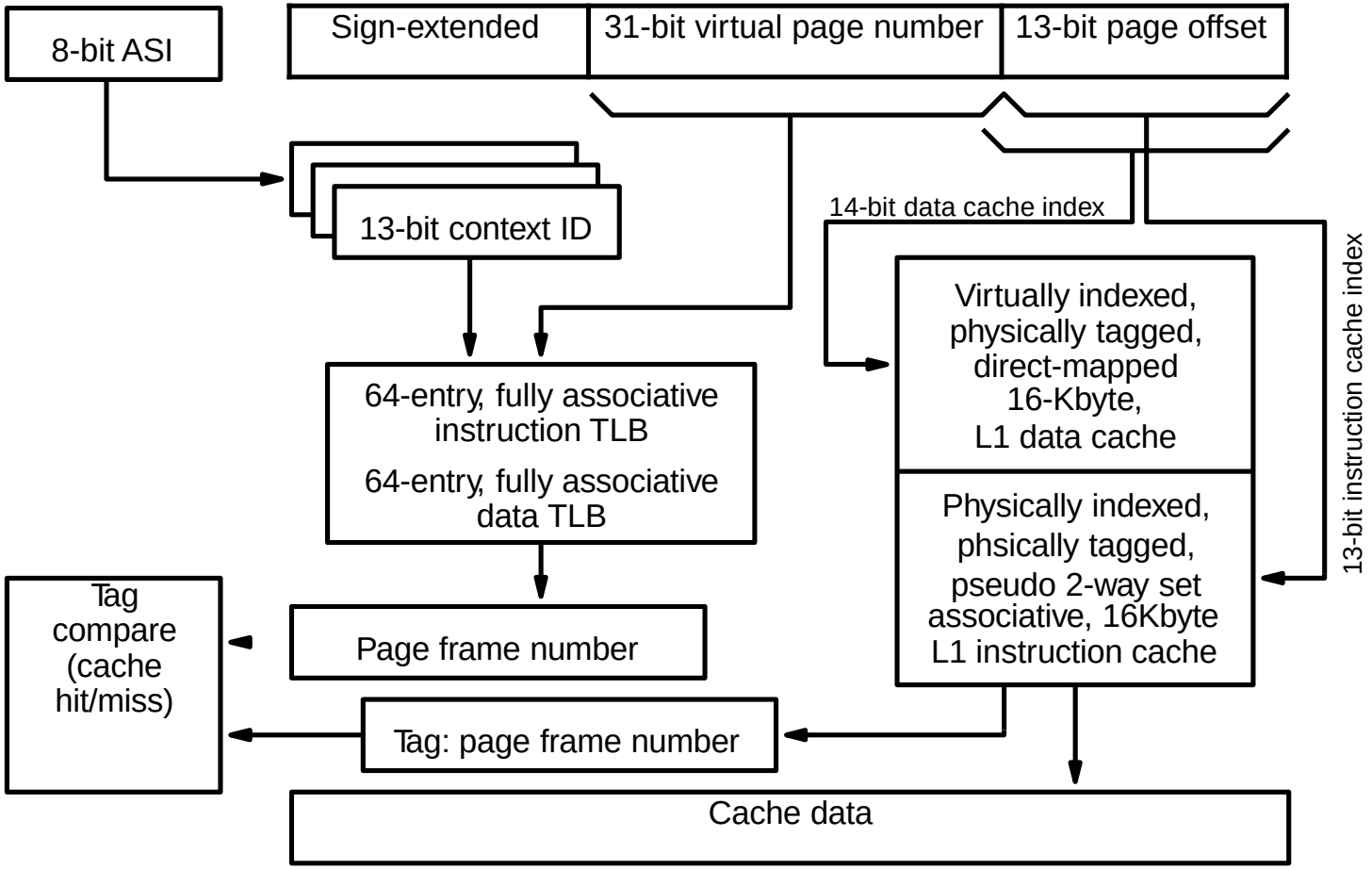
Zweistufige Adressabbildung durch Kombination von Segmentierung und Paging



Pentium II Adressierungsmechanismus



UltraSPARC-I Adressierungsmechanismus



Virtuelle Speicheradressierung beim SPARC V8