

Speicherverwaltung

Betriebssysteme WS 2011/2012



Jörg Kaiser
IVS – EOS

Otto-von-Guericke-Universität Magdeburg

Speicherverwaltung

**Wozu braucht man eine Speicherverwaltung?
Welche Eigenschaften sollte ein Speicher haben?**

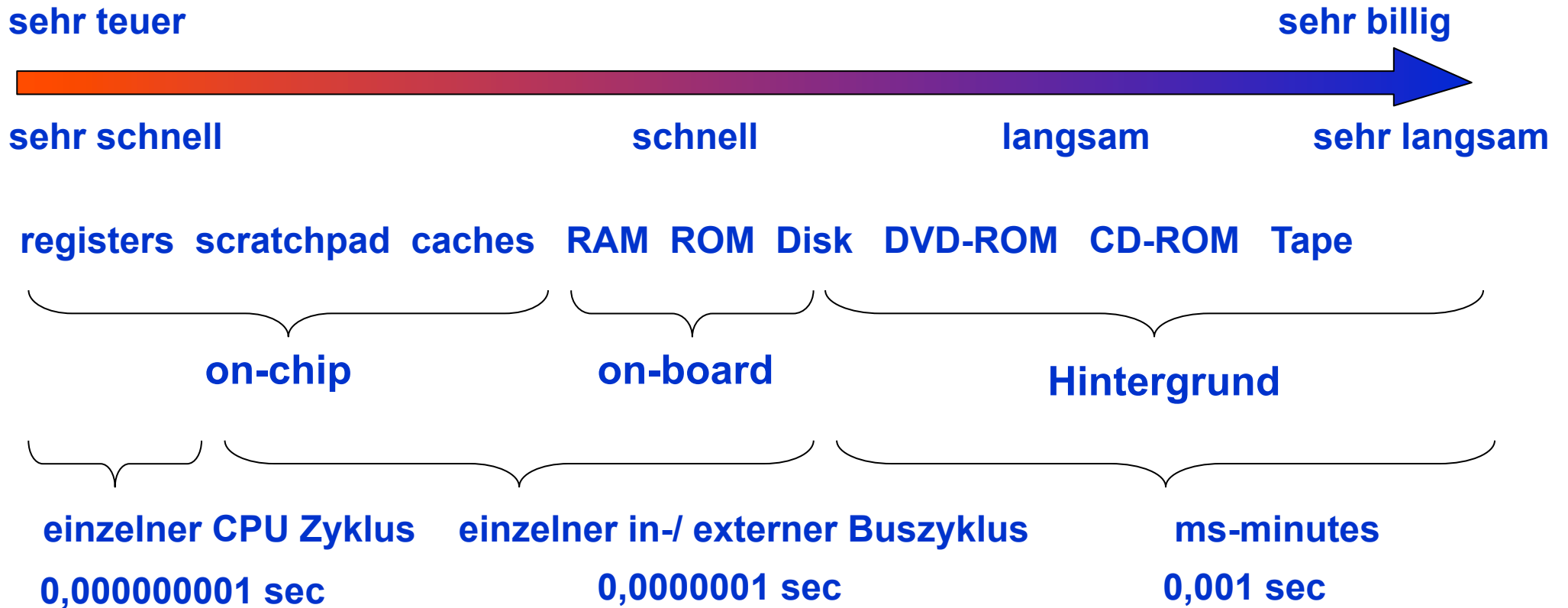
- **unendlich groß,**
- **unendlich schnell,**
- **unendlich billig,**
- **nichtflüchtig,**
- **kein Effekt durch konkurrierende Zugriffe,**
- **Schutz vor fehlerhaften Zugriffen.**

Die Speicherverwaltung hat die Approximierung dieser Eigenschaften zum Ziel !



Speicherverwaltung

Die Kosten-Leistungs-Perspektive



Speicherverwaltung

Register
Scratchpad

wird explizit vom Programmierer/Compiler kontrolliert

Cache

Hardware-Kontrolle

RAM

RAM
Abstraktion

ROM

flüchtig

Disk

DVD-ROM

Dateiabstraktion

CD-ROM

persistent

Tape

unter Kontrolle des Betriebssystems



Speicherverwaltung

Themen für die Speicherverwaltung:

★ **Relokation**

★ **gemeinsame Nutzung**

★ **Zugriffsschutz**

resultiert aus dem Zugriff durch mehrere Prozesse/Programme

★ **Transparenter Zugriff über eine Hierarchie von physischen Speichern hinweg**

logische Organisation

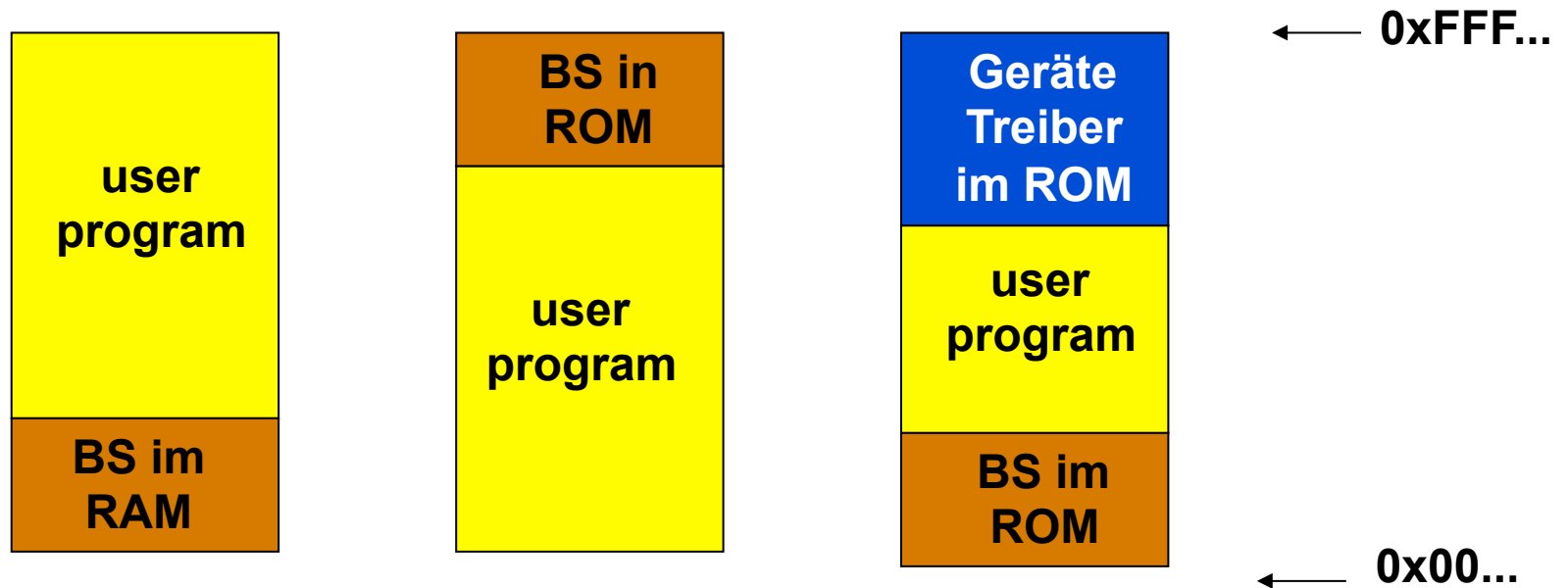
physische Organisation



Multiprogramming mit Speicherpartitionen

Statische Partitionierung:

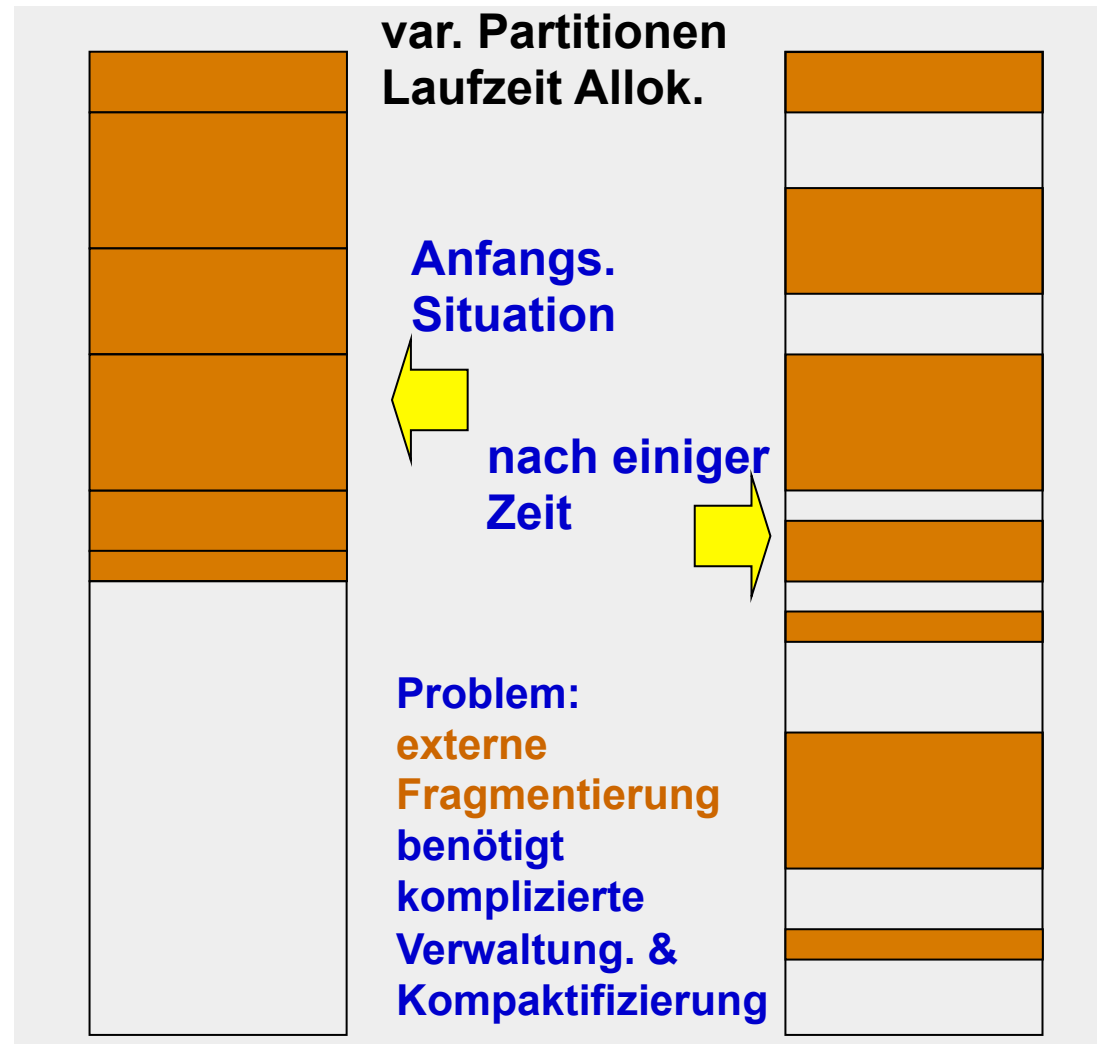
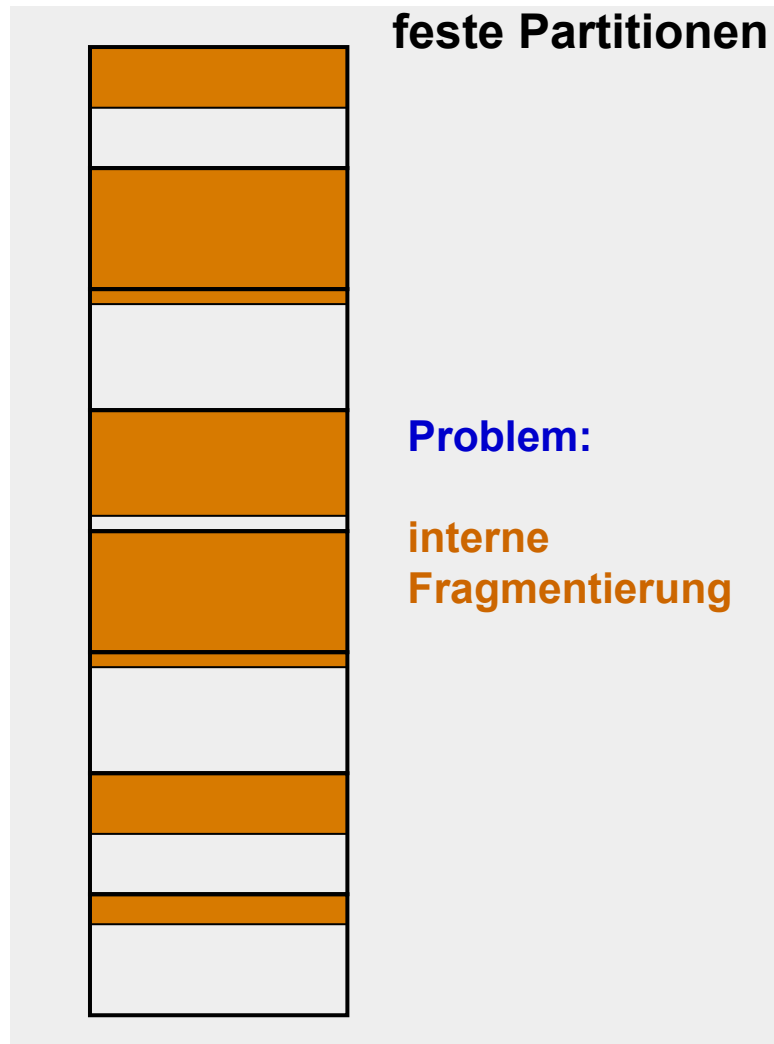
wird heute in einigen eingebetteten Systeme benutzt, z.B. Palmtops, PDAs, etc.



Eigensch.:	Größe der Speicherblocks	Zeitpunkt der Allokation
	feste Größe variable Größe	off-line (statisch) zur Laufzeit (dynamisch)

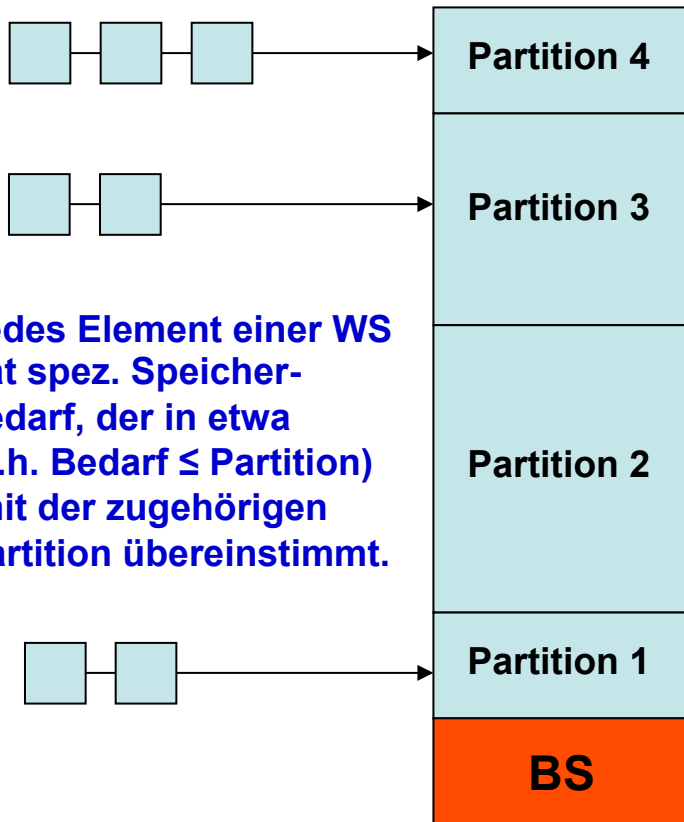


Multiprogramming mit Speicherpartitionen



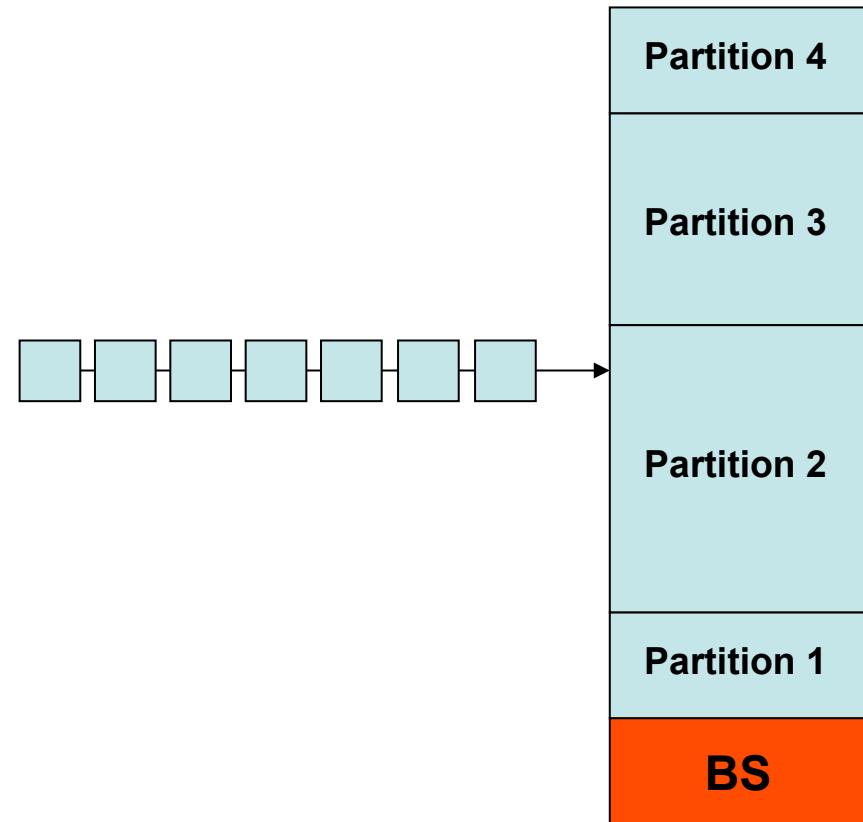
Verwaltung fester Partitionen

mehrere Warteschlangen



Jedes Element einer WS hat spez. Speicherbedarf, der in etwa (d.h. Bedarf \leq Partition) mit der zugehörigen Partition übereinstimmt.

eine Warteschlange




Speicherverwaltung mit Partitionen

Relokation und Zugriffsschutz

Relokation:

Probleme: Programme müssen in beliebigen Speicherbereichen funktionieren.

Mechanismen:

1. Statisches Binden von Speicheradressen zur Compilierungszeit 
2. Relokation beim Einlagern in den Hauptspeicher → loader/linker
3. Relokation zur Laufzeit → benötigt positionsunabhängigen Code
→ setzt Unterstützung durch die Rechnerarchitektur voraus.

Zugriffsschutz:

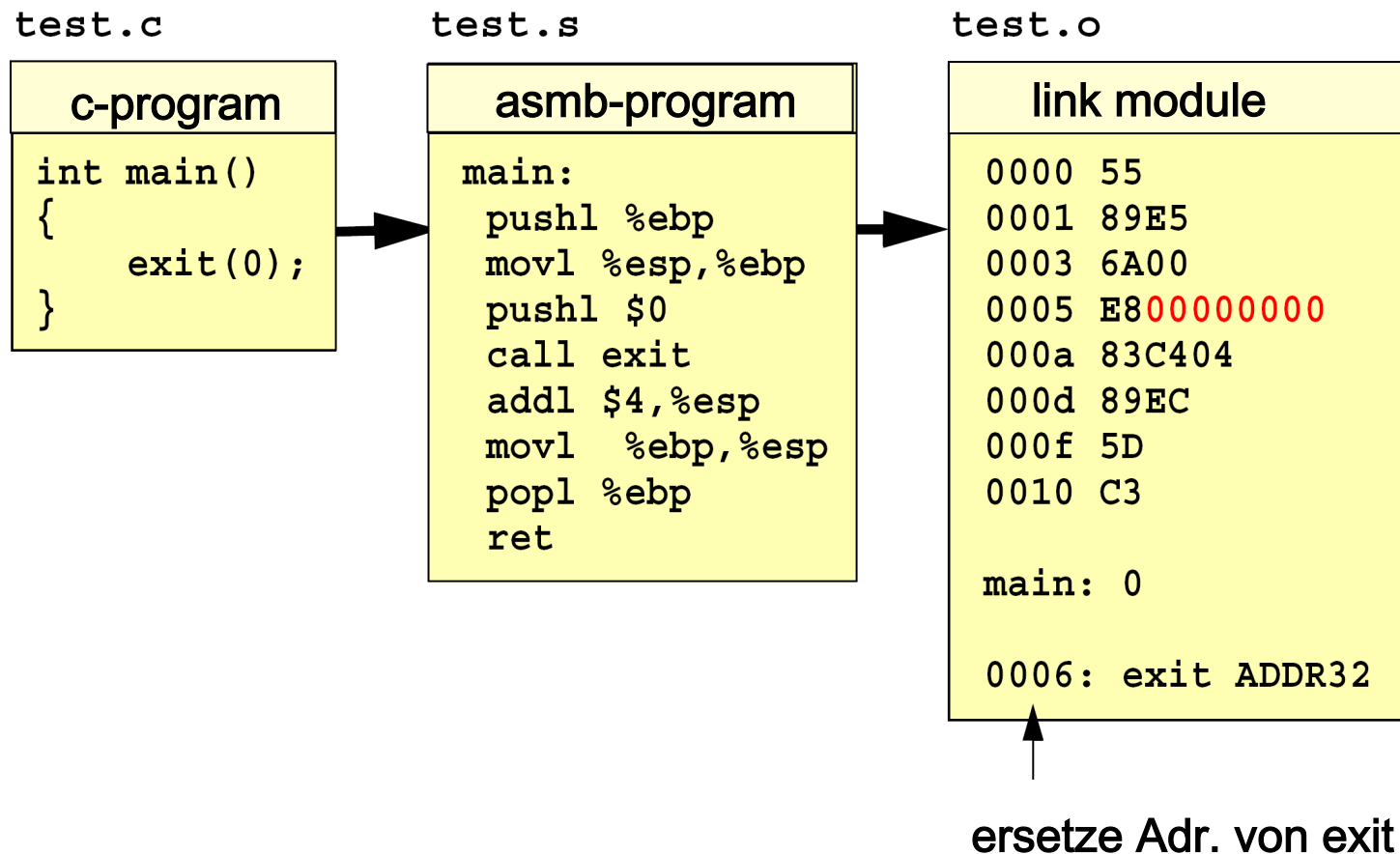
Probleme: Beliebige Referenzen auf Bereiche ausserhalb der Partition.

Mechanismen:

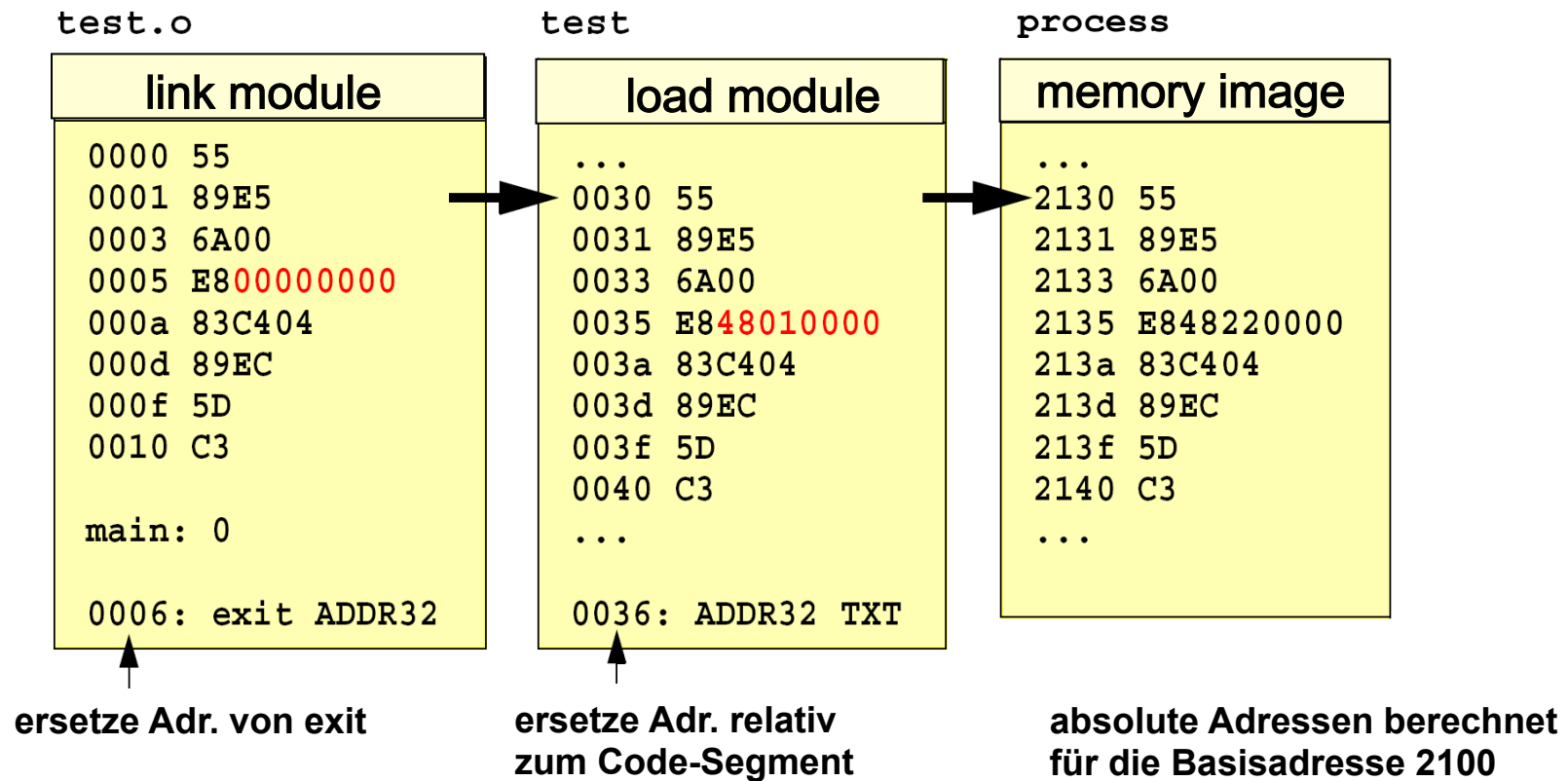
1. Speicherblöcke fester Länge, die mit einem 4-bit Protection-Code gekennzeichnet sind (tagged memory).
Der p-code wird bei jedem Zugriff mit dem entsprechenden Feld des Programm-Status-Worts (PSW) verglichen (IBM 360)
2. Base und Bound Register (CDC 6600)



Relokation

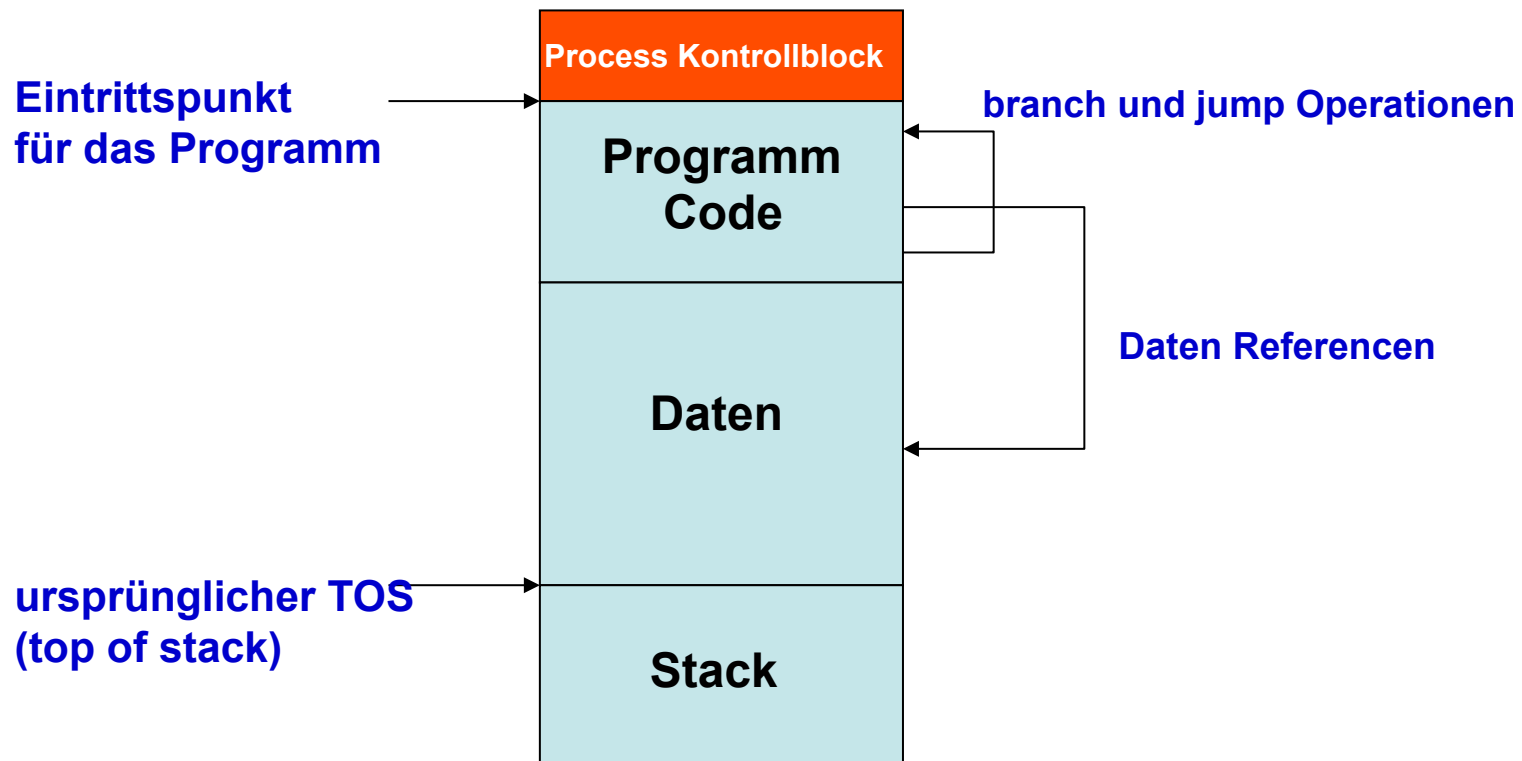


Relokation

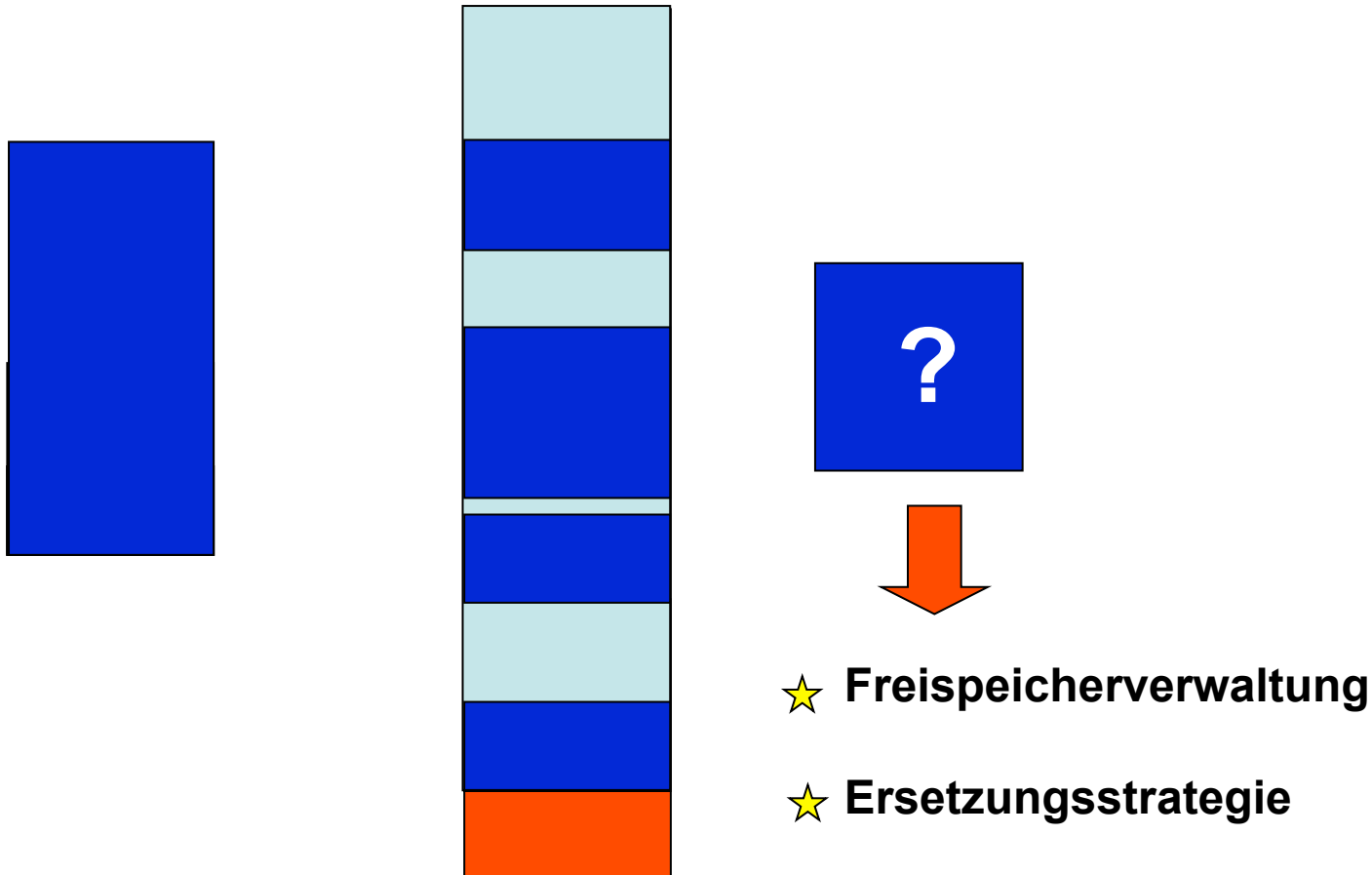


Dynamische Partitionen and Ein-/Auslagern (Swapping)

Ein-/Auslagern: Jedes Prozessabbild wird vollständig vom Speicher auf die Platte geschoben und umgekehrt, mit seinen Code-, Daten- and Stackbereichen.

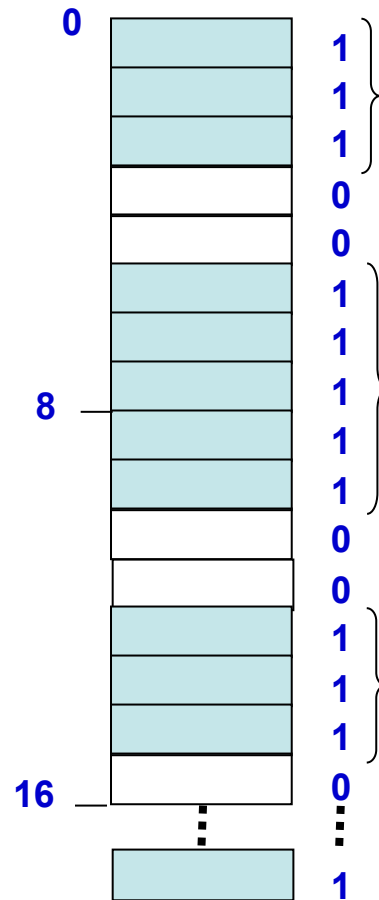


Dynamische Partitionen und Ein-/Auslagern (Swapping)



Dynamische Partitionen und Ein-/Auslagern (Swapping)

Problem: Wo sind freie Blöcke und welche Größe haben sie ?

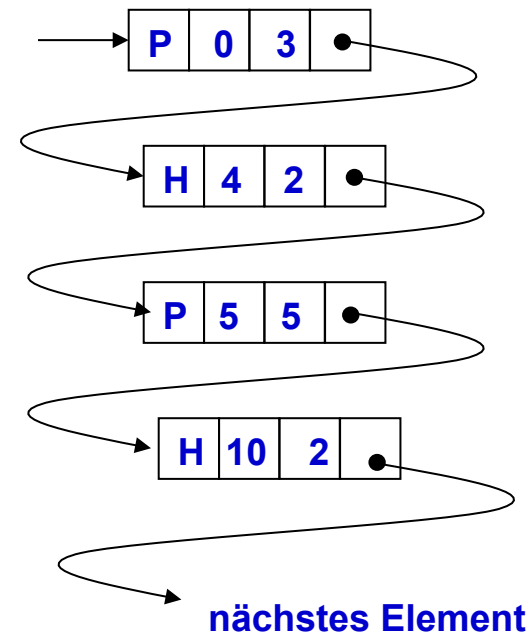


Bitmap-
darstellung:

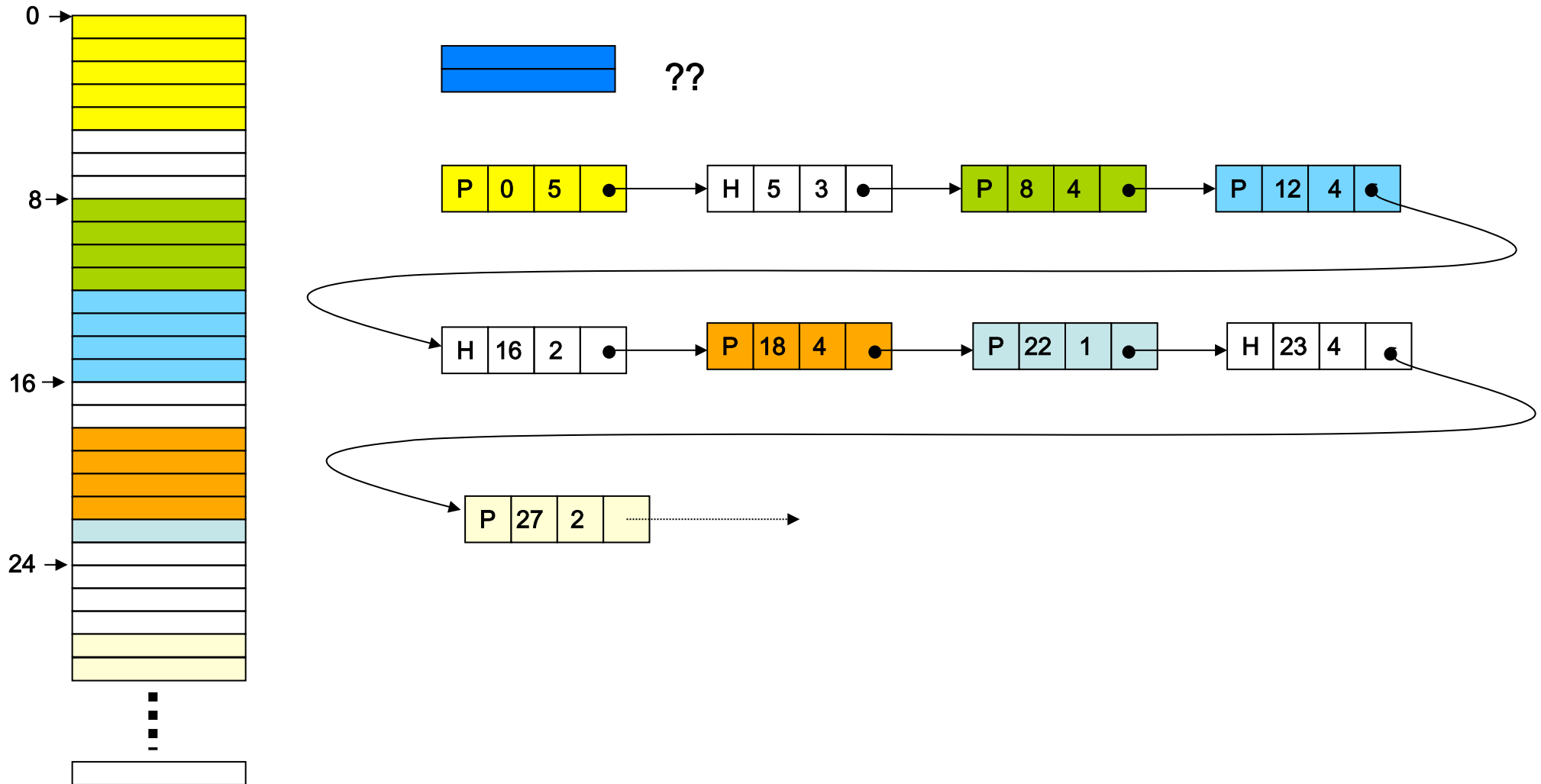
1110011
1110011
10.....

1 = allozierter Block
0 = Loch

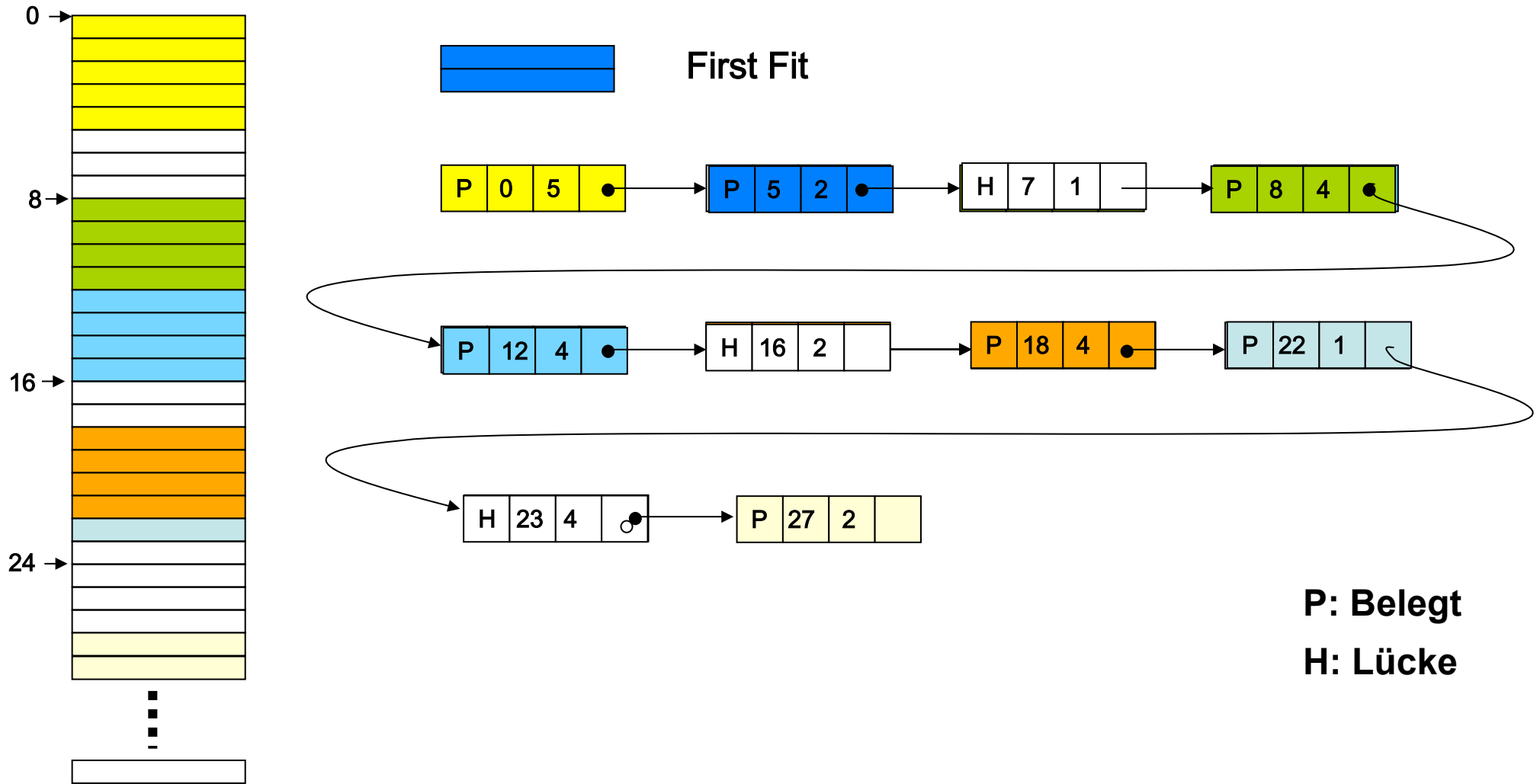
Darstellung durch
verkettete Liste:



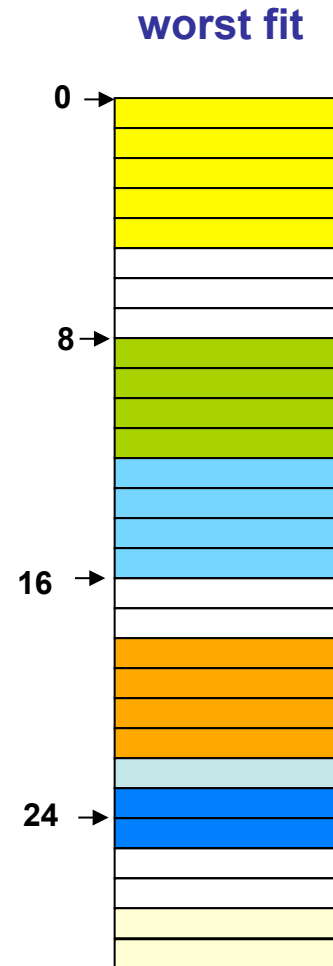
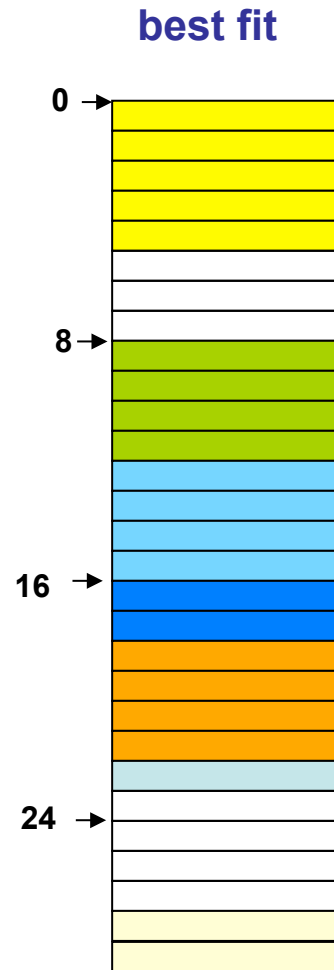
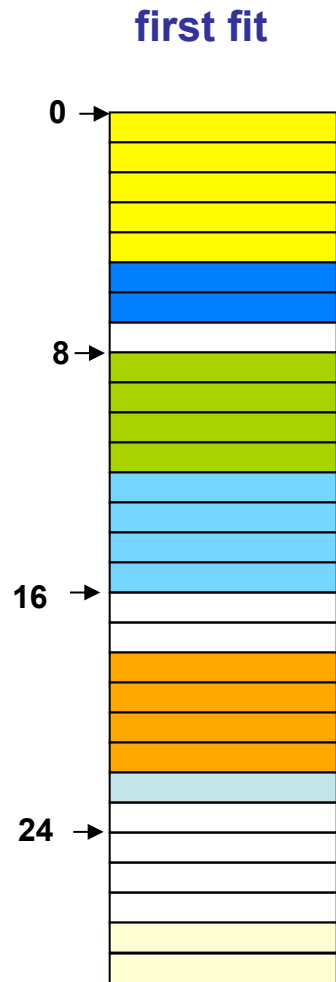
Dynamische Partitionen und Ein-/Auslagern (Swapping)



Dynamische Partitionen und Ein-/Auslagern (Swapping)



Dynamische Partitionen und Ein-/Auslagern (Swapping)

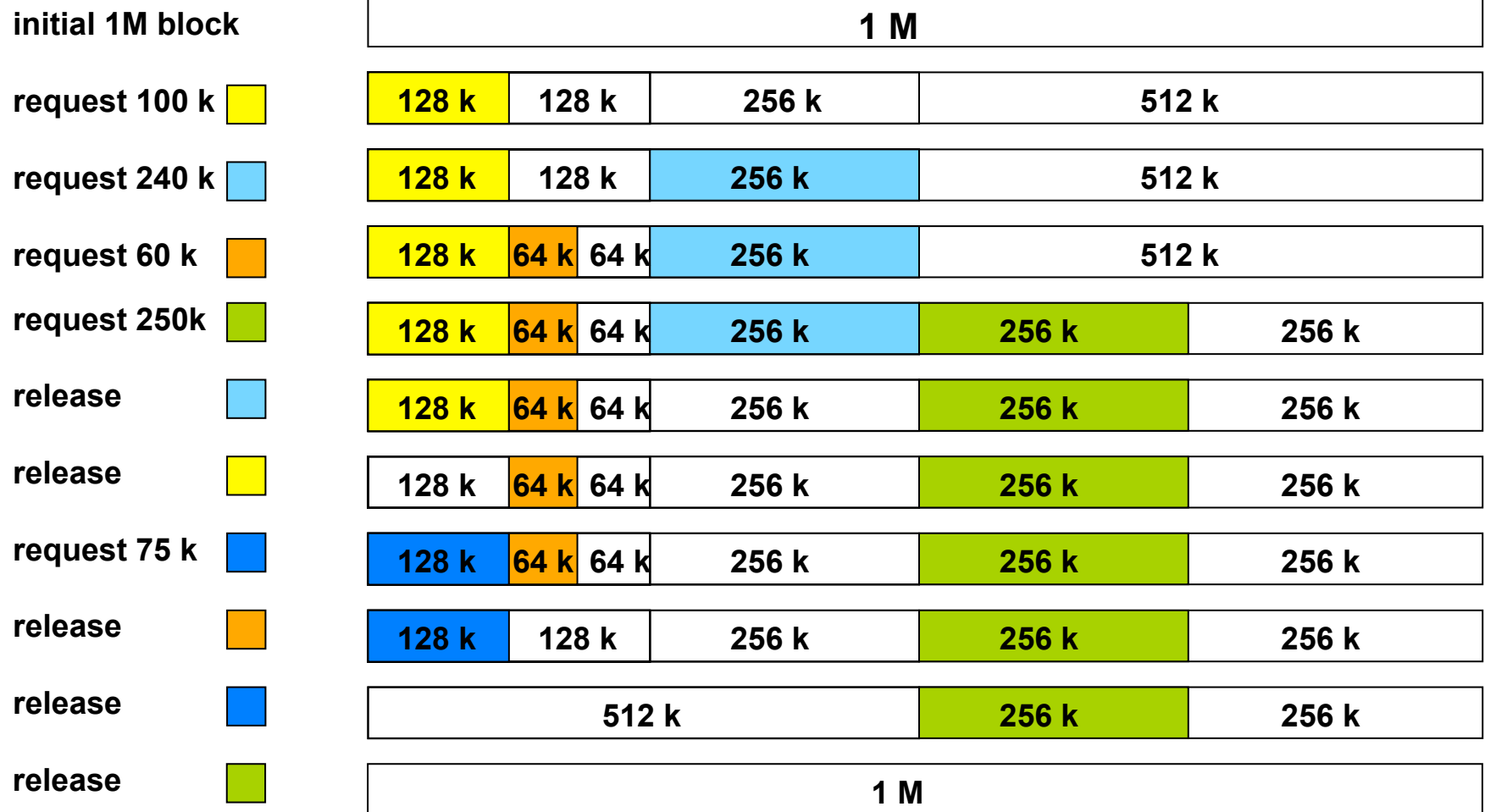


next fit: Startpunkt ist die Position des zuletzt gefundenen Blocks

quick fit: maintains multiple list of free memory blocks according to size of blocks.



Das Buddy-System



Multiprogramming mit Speicherpartitionen

- ➔ **Es wird der physische Speicher verwaltet.**
- ➔ **Prozessen sind Speicherpartitionen zugeteilt**
- ➔ **Speicherpartitionen können fest oder variabel sein**
- ➔ **Verwaltung der Partitionen kann durch eine Warteschlange oder durch mehrere Warteschlangen, die Partitionsgrößen zugeordnet sind, verwaltet werden**



Diskussion:

Bisher wurde betrachtet::

- 1. Verwaltung des **realen, physischen** Speichers.**
- 2. Der Adressraum ist dem Realspeicher angepasst.**
- 3. Ein- und Auslagern von Speicherblocks wird explizit durch das BS vorgenommen.**
- 4. Die Größe der Einheiten wird durch den Programmierer festgelegt.**

Probleme:

- Programme (oder Datensätze) können größer sein als der Realspeicher.**
- Zugriffsschutz, wenn mehrere Prozesse in einem Adreßraum ablaufen.**



Diskussion:

Adressraum eines Prozessors:

32 Bit = 4.294.967.296

64 Bit = 18.446.821.383.201.879.616 ~ 2 x 1019**

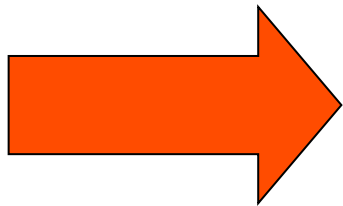


Idee:

**Betrachten des Realspeichers als ein Fenster in einen viel größeren Speicher.
Trennung von logischem und realem Adressraum.**

Wünschenswerte Ziele:

- 1. Transparenter Mechanismus für das Ein-Auslagern von Speicherblocks.**
- 2. Logischer Adreßraum ist sehr viel größer als der Adressraum des Realspeichers.**
- 3. Transparenter Mechanismus zur Relokation.**
- 4. Besserer Zugriffsschutz durch Trennung logischer Adreßräume.**



virtueller Speicher



virtual memory - history

The Atlas team at the University of Manchester created the first prototype version of virtual memory in 1959. They called it One-Level Storage System.

The major achievements were:

- Separation between Logical and Physical address space.
- Introduction of fixed size pages and frames.
- Hardware support for address translation.
- Demand paging
- A replacement algorithm

1960's: all popular commercial operating systems used virtual memory e.g.
(Multics), IBM 360/67, CDC 7600, GE 645, RCA Spectra/70, Burroughs 6500.

1970s: IBM 370, DEC VMS, DEC TENEX, and Unix .

1995: Microsoft included virtual memory in Windows 95.



The Computer Journal

Vol. 4, Issue 3,
October 1961

virtual memory also
described in:

John Fotheringham:
Dynamic Storage Allocation
in the ATLAS Including
Automatic Use of Backing
Store

Communications of the ACM,
Volume 4 , Issue 10
(October 1961)

The Manchester University Atlas Operating System Part I: Internal Organization

By T. Kilburn, D. J. Howarth, R. B. Payne and F. H. Sumner

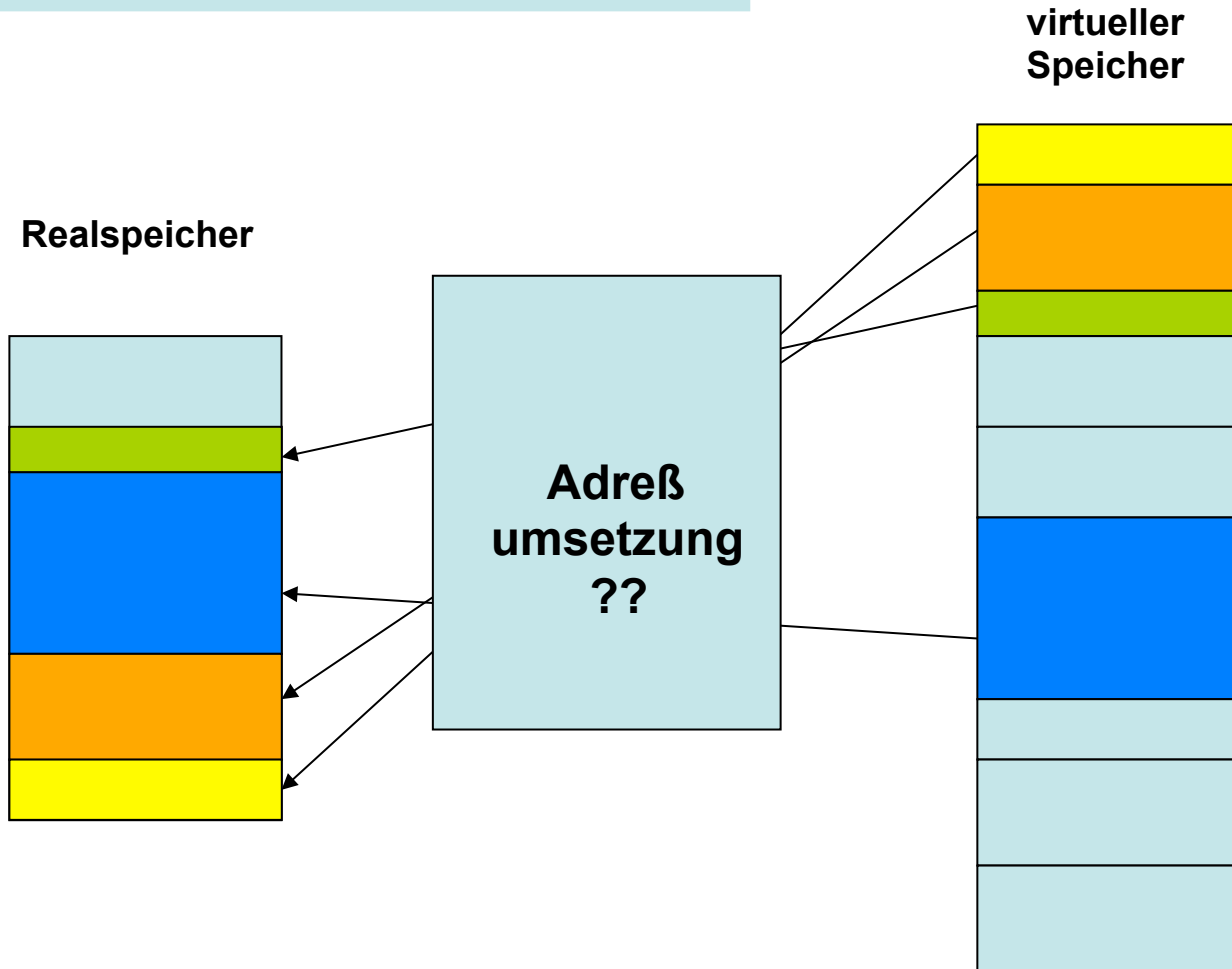
Introduction

Atlas* is the name given to a comprehensive computer system designed by a joint team of Ferranti Ltd. and Manchester University engineers. The computer system comprises the central computer, fixed store, core store, magnetic drum store, magnetic tapes, and a large quantity and variety of peripheral equipments for input and output. The Manchester University Atlas has 32 blocks of core store each of 512 forty-eight bit words. There is also a magnetic drum store, and transfers between core and drum stores are performed automatically, giving an effective one-level store* of over two hundred blocks. The average time for an instruction is between 1 and 2 microseconds. The peripheral equipments available on the Manchester University Atlas include

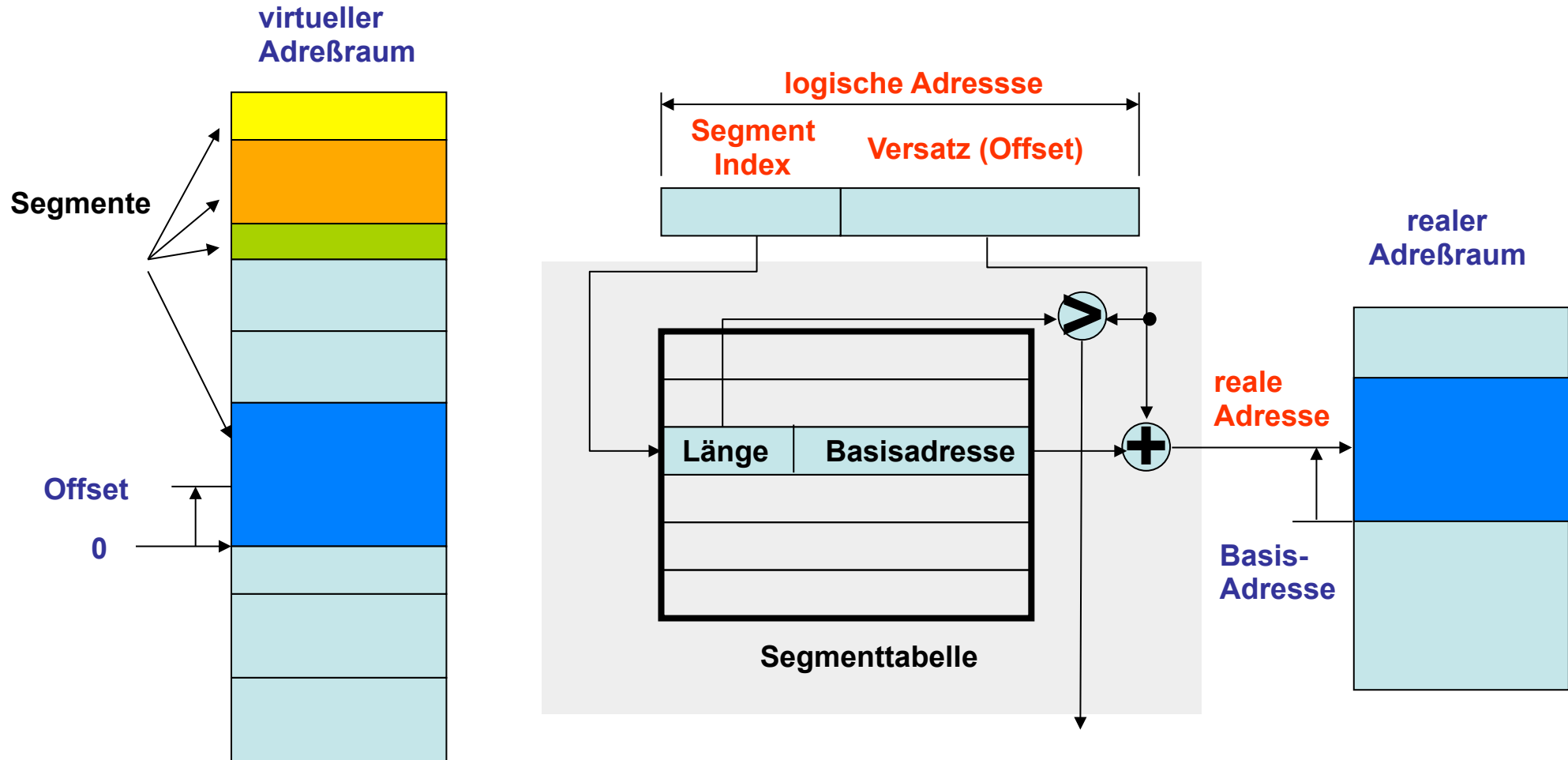
8 magnetic tape decks	90,000 characters per second
4 paper tape readers	300 characters per second
4 paper tape punches	110 characters per second
1 line printer	600 lines per minute
1 card reader	600 cards per minute
1 card punch	100 cards per minute



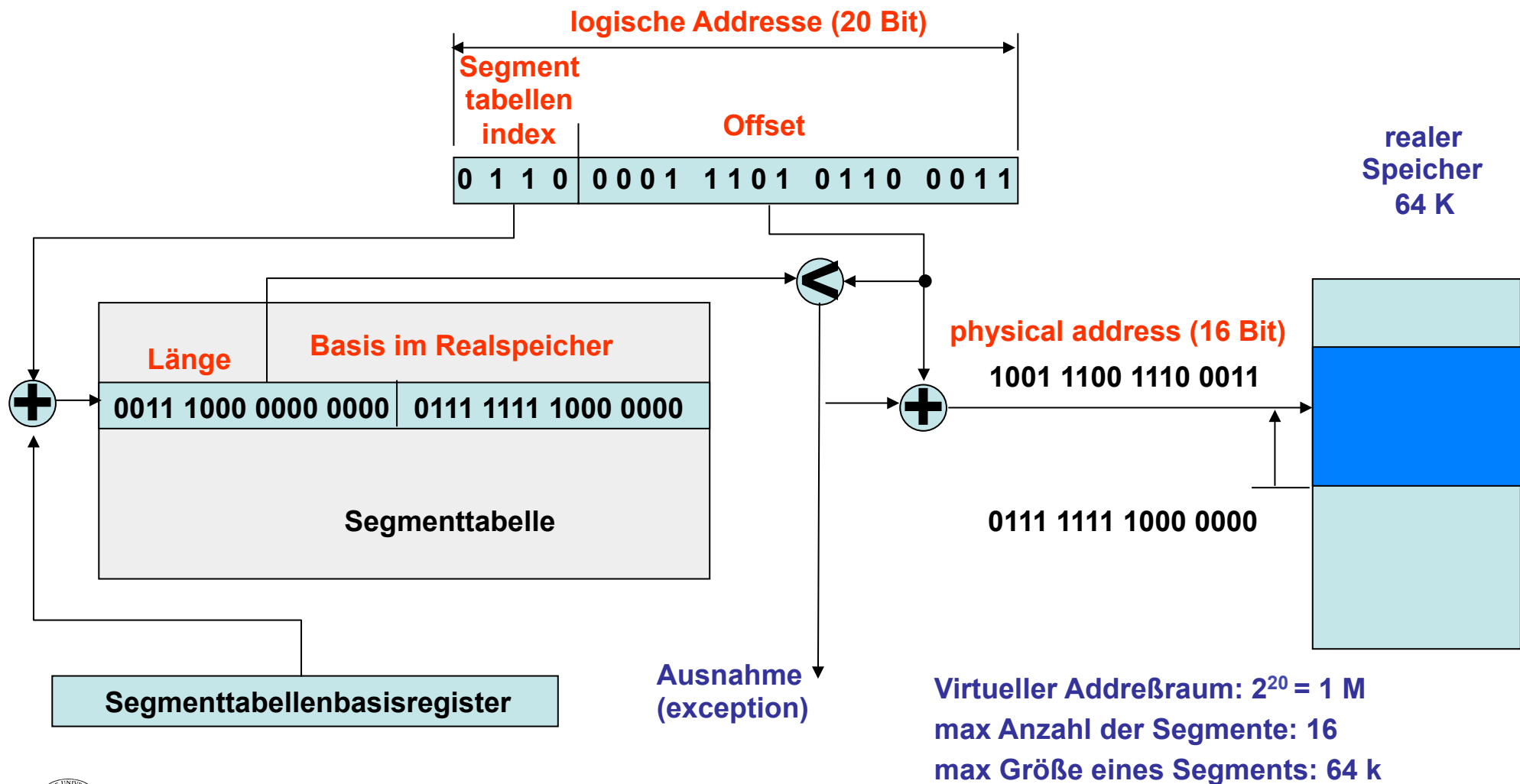
virtual memory



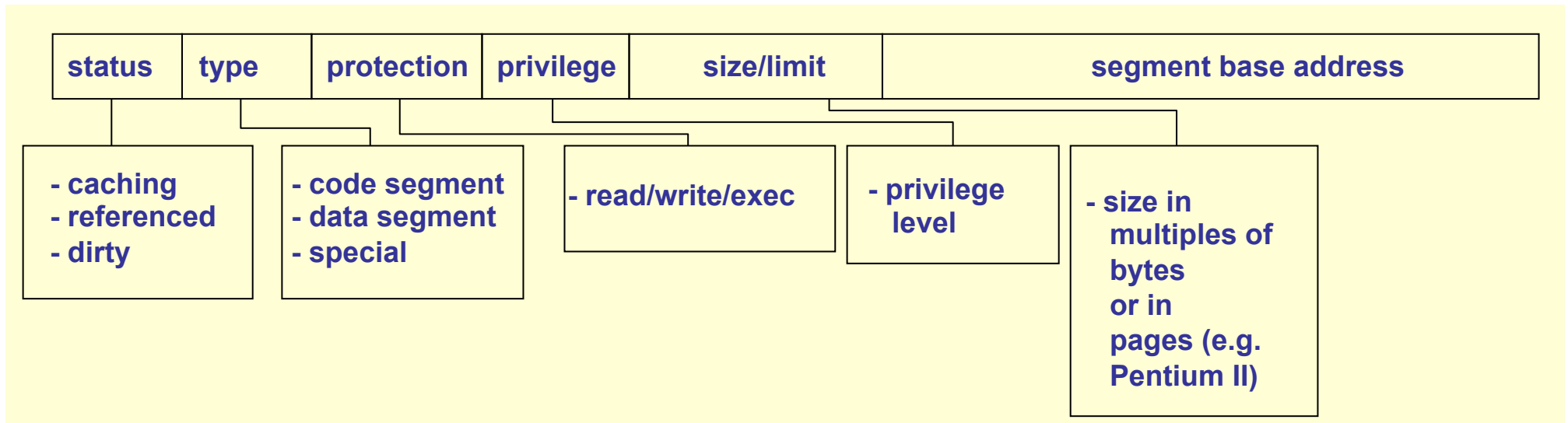
"segmentierter" virtueller Speicher



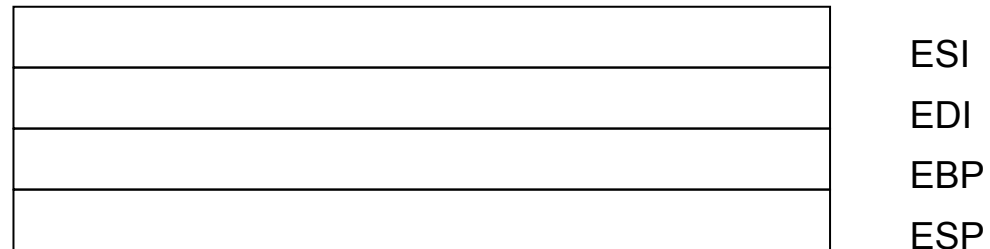
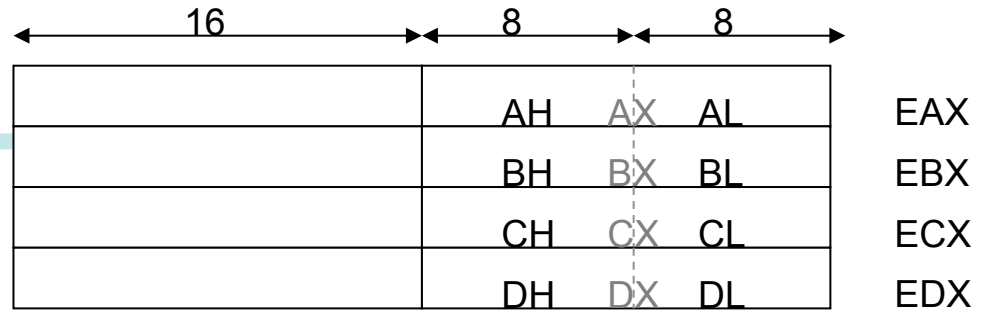
Segmentierter Speicher - Adreßumsetzung



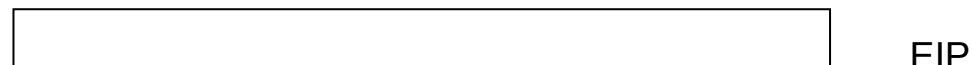
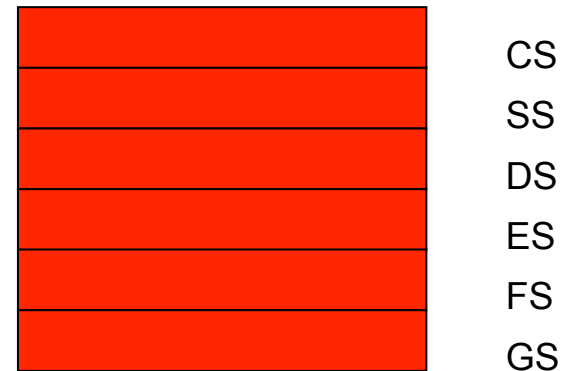
Segmentdeskriptor des Pentium



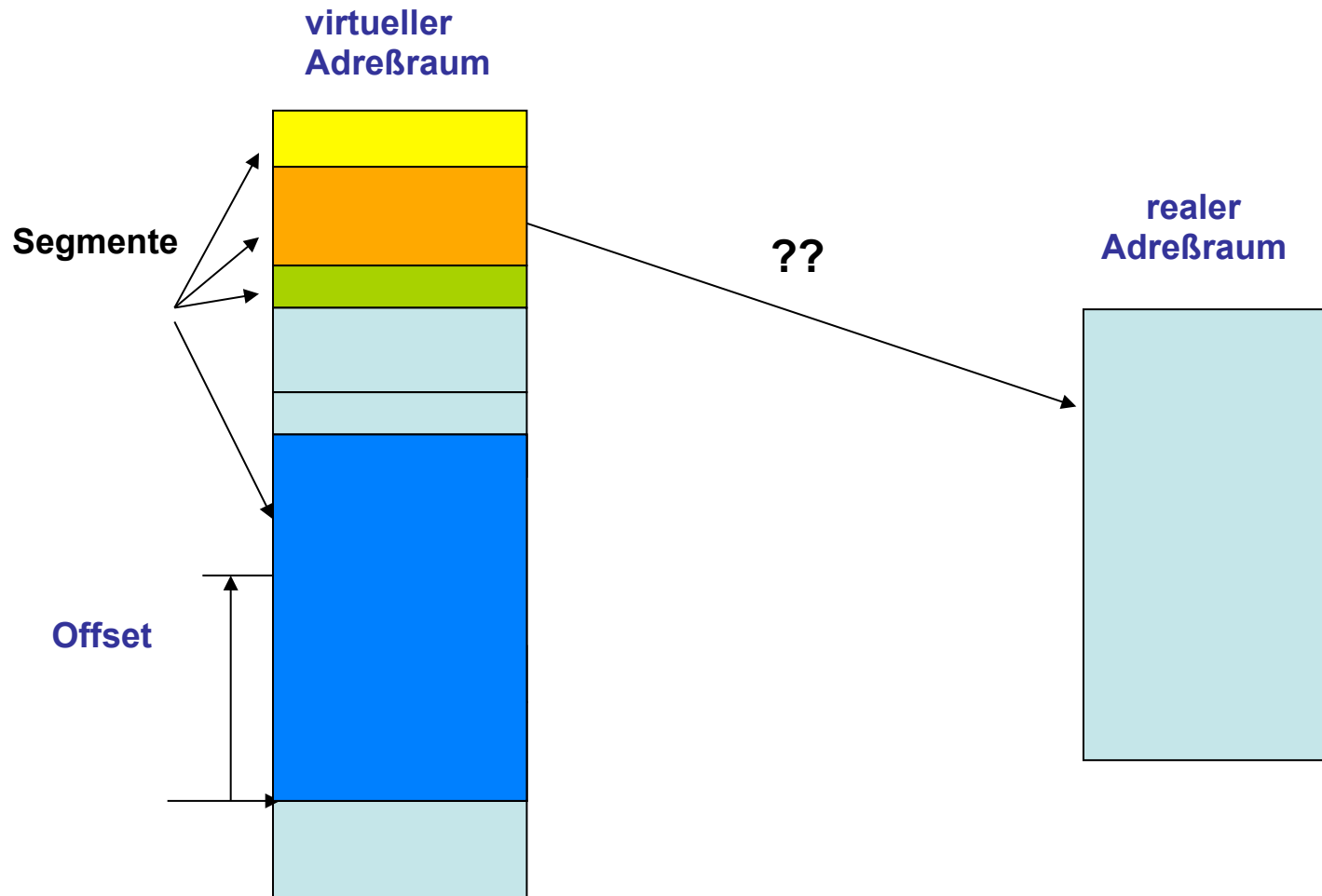
(Haupt) Registersatz des Pentium 4



- CS: enthält Basis des Code Segments**
- SS: enthält Basis des Stack-Segments**
- DS: enthält Basis des Daten-Segments**
- ES: enthält Basis des Extra-Segments**
- FS: zusätzliches Extra-Segment (ab 386)**
- GS: zusätzliches Extra-Segment (ab 386)**



Probleme des segmentierten virtuellen Speichers



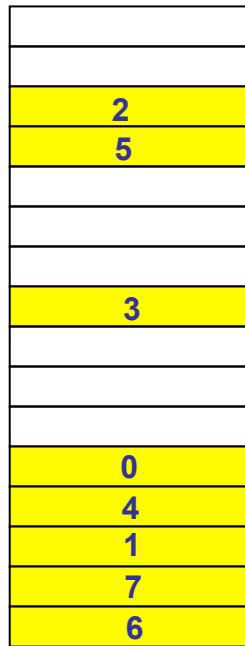
Seitenorientierter virtueller Speicher



Idee: Seiten fester Größe werden auf Kacheln (Seitenrahmen: frames) fester Größe im Hauptspeicher abgebildet.

virtueller
Adreßraum

60k - 64k
56k - 60k
52k - 56k
48k - 52k
44k - 48k
40k - 44k
36k - 40k
32k - 36k
28k - 32k
24k - 28k
20k - 24k
16k - 20k
12k - 16k
8k - 12k
4k - 8k
0k - 4k



(virtuelle)
Seiten

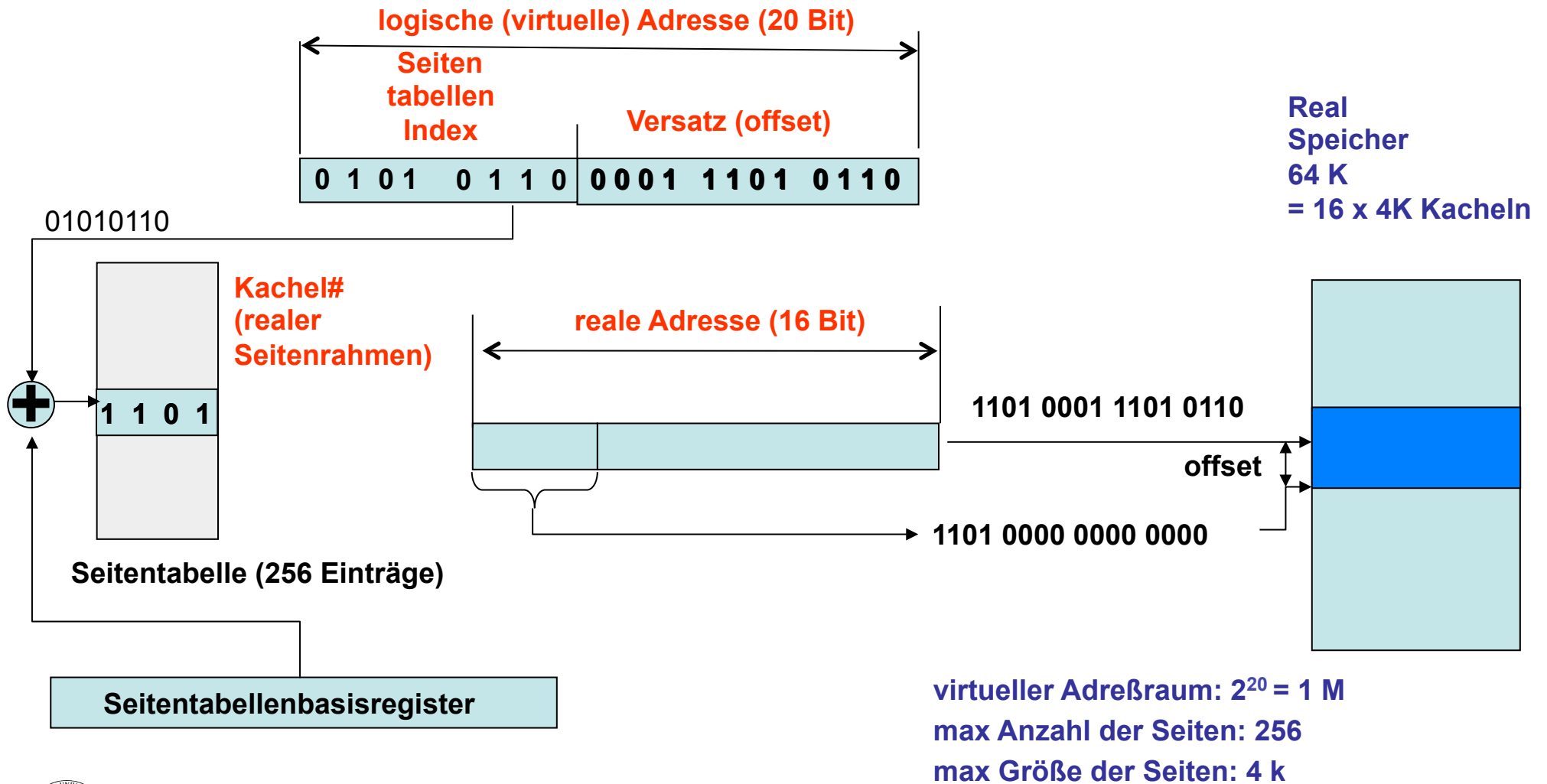
Realspeicher-
Adresse

28k - 32k
24k - 28k
20k - 24k
16k - 20k
12k - 16k
8k - 12k
4k - 8k
0k - 4k

(reale)
Seitenrahmen
bzw.
Kacheln



Adreßumsetzung mit Seiten



Größe der Seitentabellen

32-Bit Adreßraum: 4G Adressen

Seitentabellengröße @ 4k: 1M Einträge

64-Bit Adreßraum: 4G·4G Adressen

Seitentabellengröße @ 4k: 1M · 4G Einträge

1. Seitengröße erhöhen: z.B. UltraSPARC II unterstützt 8k, 64k, 512k and 4M Seiten

Grund: weniger Seiten

Problem: interne Fragmentierung

2. Seitentabellen ein-und auslagern: mehrstufige Seitentabelle

Grund: a.) Virtueller Teil Speicher ist billig, b.) es wird aktuell immer nur ein kleiner der Seiten benötigt (→ working set)

Problem: eine zusätzliche Indirektionsstufe

3. Flexible Zuordnung von Tabelleneinträgen: Invertierte Seitentabellen

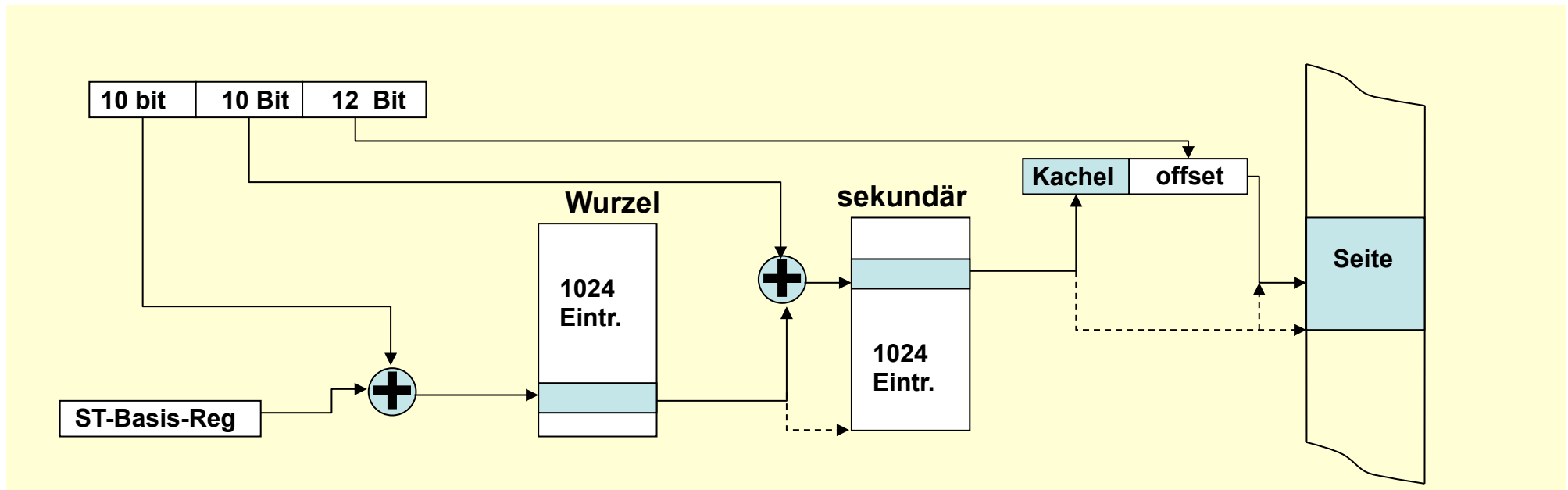
Verfahren: Tabelleneinträge werden Seiterahmen fest zugeordnet

Grund: Der reale Adreßraum ist klein im Vergleich zum virtuellen.

Problem: viel mehr Seiten als Seiterahmen → keine eindeutige Abbildung

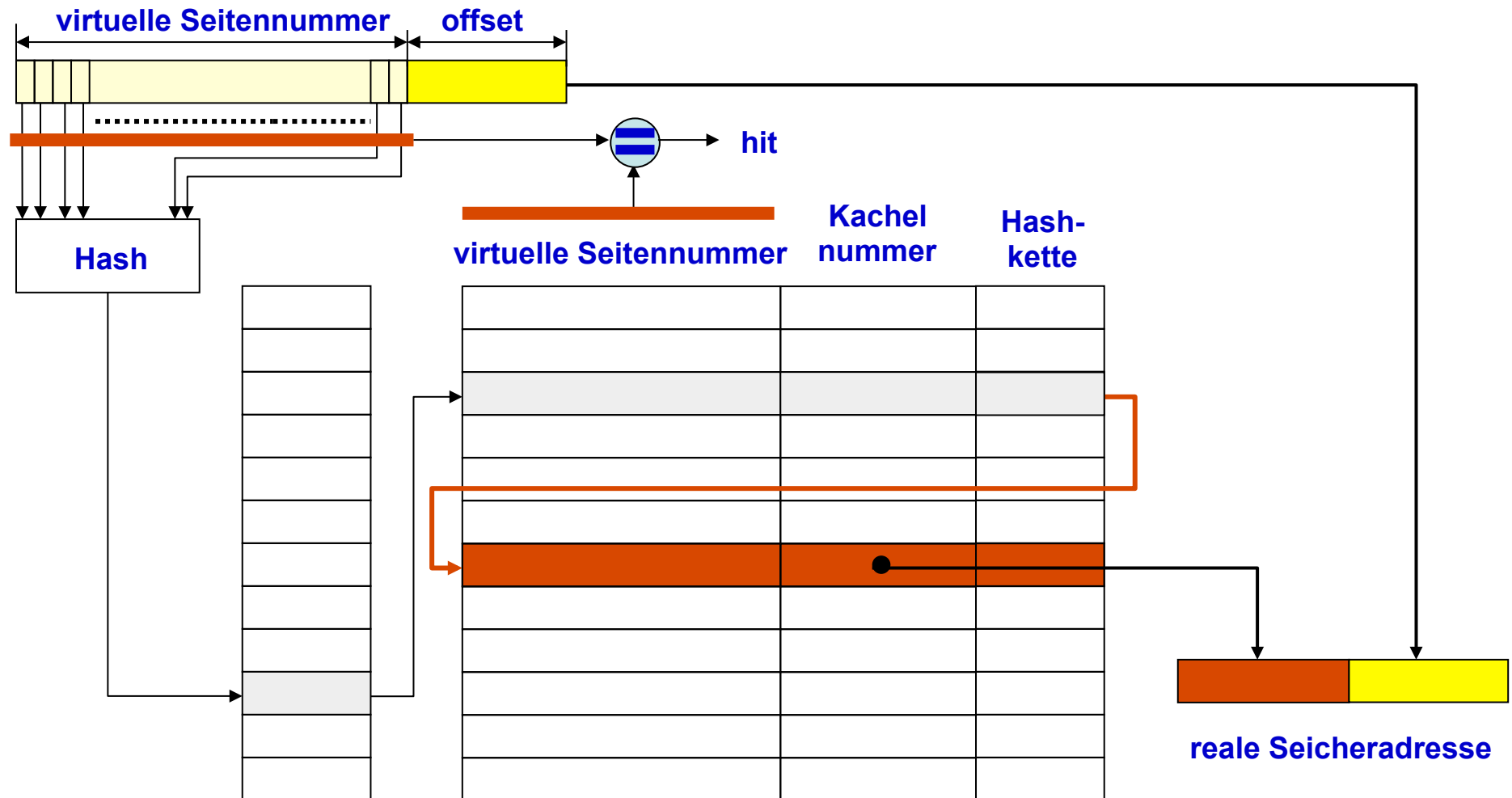


mehrstufige hierarchische Seitentabellen



invertierte Seitentabelle







PPC, AS/400



Seitengröße

für n Segmente mit p bytes ist die interne Fragmentierung: $n \cdot p/2$

Zielkonflikte:

Seitengröße:	large	small
interne Fragmentierung ($n \cdot p/2$):		
Seitentabellengröße+Management Ovh.		
Ladezeit von der Platte		

Übliche Seitengrößen liegen zwischen 512 Bytes und 64KB.

Heute werden Seitengrößen von 4KB oder 8KB benutzt.



invertiert oder nicht invertiert?

Pro invertiert:

Seitentabelle ist proportional zur Größe des Realspeichers:

z.B. für 1 G @ 8k Seiten: 128k Einträge unabhängig von Größe d. virt. A-Raums

Con invertiert:

- Hashing muss bei jedem Speicherzugriff durchgeführt werden.
- Verkettung kann zu mehrfachen Zugriff führen.
- benötigt komplexe Ersetzungs-und Verwaltungsstrategien.

ABER:









- Hierarchische Seitentabellen benötigen ebenfalls mehrfachen Zugriff.
- wenn sekundäre Seitentabellen ausgelagert sind: mehrfacher Plattenzugriff.



Hardwareunterstützung wird in allen Fällen benötigt !



Diskussion: Segmente gegen Seiten

	Transparenz für Progr.	Anzahl d. Adreßräume	virt. Adr. raum > realer Speicher	variable Objektgr.	Fragment.	Verw. overhead	Hauptgrund für Einführung
Seitenor.		ein			intern		unendlich viel Speicher
Segmentor.		viele			extern		mehrere Adr. räume



Das Lokalitätsprinzip

Lokalitätsprinzip:

1. In einem beschränkten zeitlichen Fenster zeigen Programme eine geringe räumliche Streuung der Referenzen, sowohl für Instruktionen als auch für Datenzugriffe.
2. Aus dem Referenzierungsverhalten in der Vergangenheit kann man auf zukünftige Referenzierungen schließen.

Das Lokalitätsprinzip ist die Basis für jeglichen Caching-Mechanismus!



Verwaltung des virtuellen Speichers

Woher weiss man, wie viele Seiten benötigt werden?

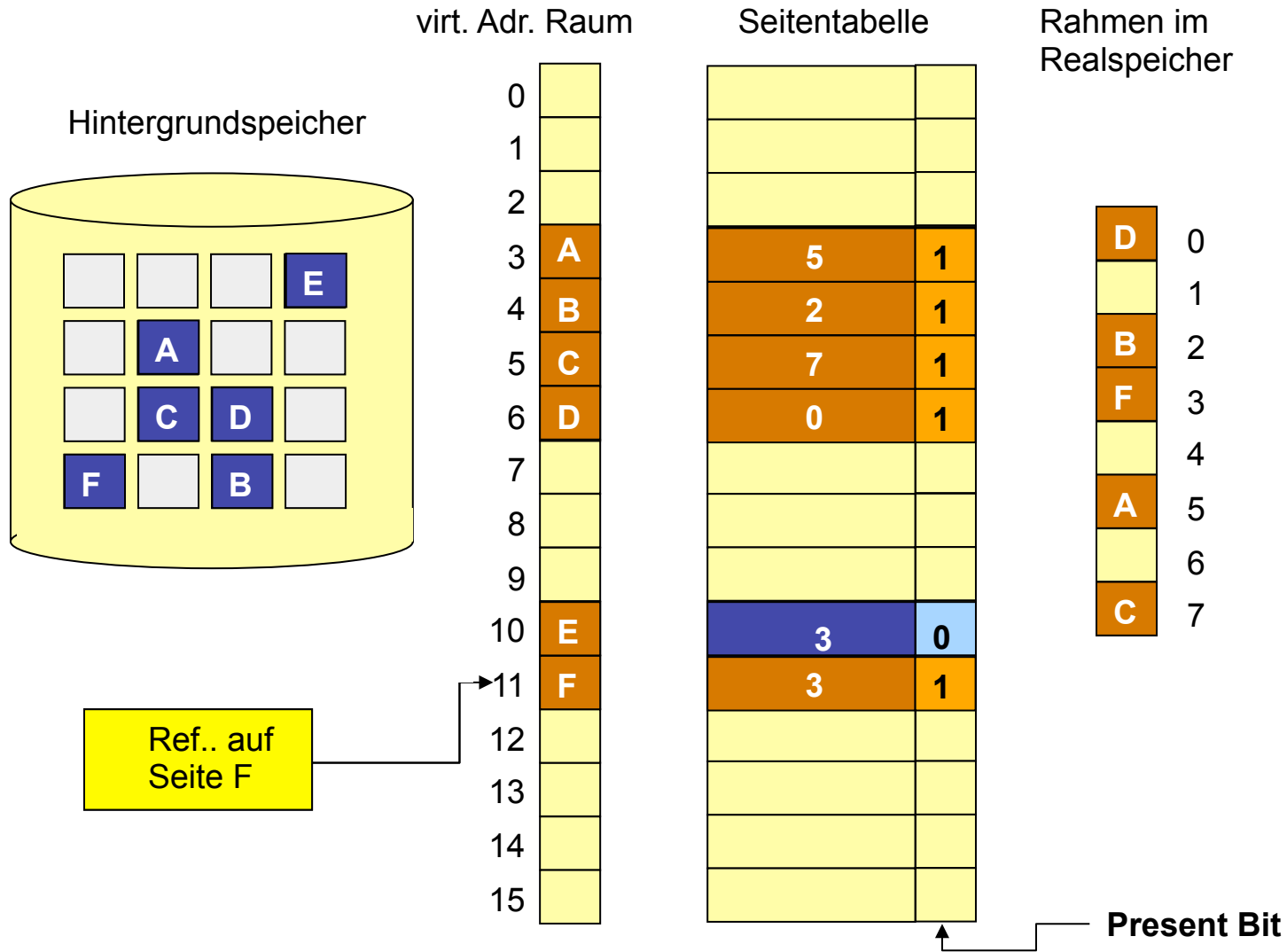
➔ einlagern von Seiten auf Anforderung

Was tun, wenn mehr Seiten benötigt werden als Seitenrahmen zur Verfügung stehen?

➔ Seitenersetzungsmechanismen



Demand Paging



Demand Paging: Diskussion

Normaler Zugriff, kein Seitenfehler: Zugriffszeit ~ 5 - 200 ns

Wie hoch ist die Zugriffszeit bei einem Seitenfehler?

Annahmen {
p: Wahrscheinlichkeit für einen Seitenfehler
Normale effektive Zugriffszeit: 100 ns
Laden einer Seite von der Platte: ~ 20 ms

Effektive Zugriffszeit für $p=0,01$:

$$(1-p) \cdot 100 + p \cdot 20.000.000 = 0,99 \cdot 100 + 0,01 \cdot 20.000.000 = 99 + 200.000 \text{ ns} = 198 \mu\text{s}$$

Um im Bereich der normalen Zugriffszeit zum Realspeicher zu bleiben muss die Wahrscheinlichkeit eines Seitenfehlers in der Größenordnung von 0,000005 liegen !

➡ 1 Seite pro 200000 Zugriffe darf zu einem Seitenfehler führen!

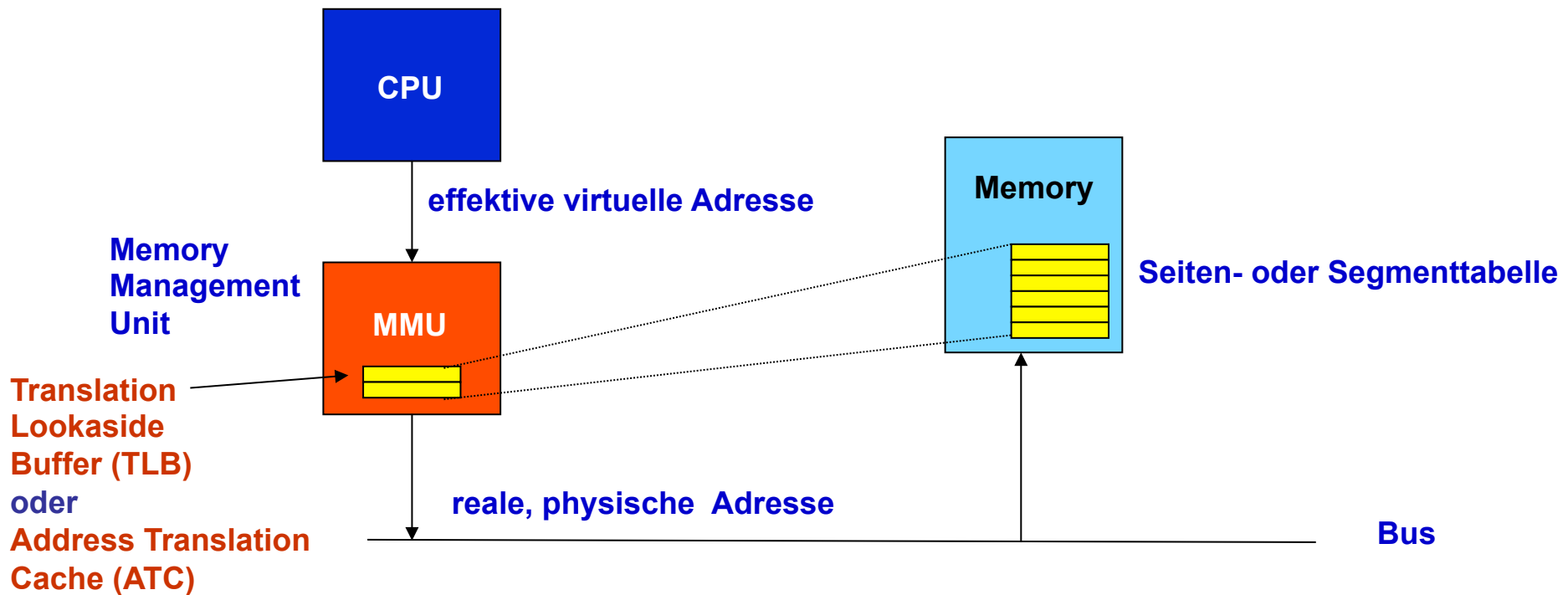


Die MMU: Speicherverwaltungseinheit

Adreßübersetzung benötigt mehrere Ebenen der Indirektion

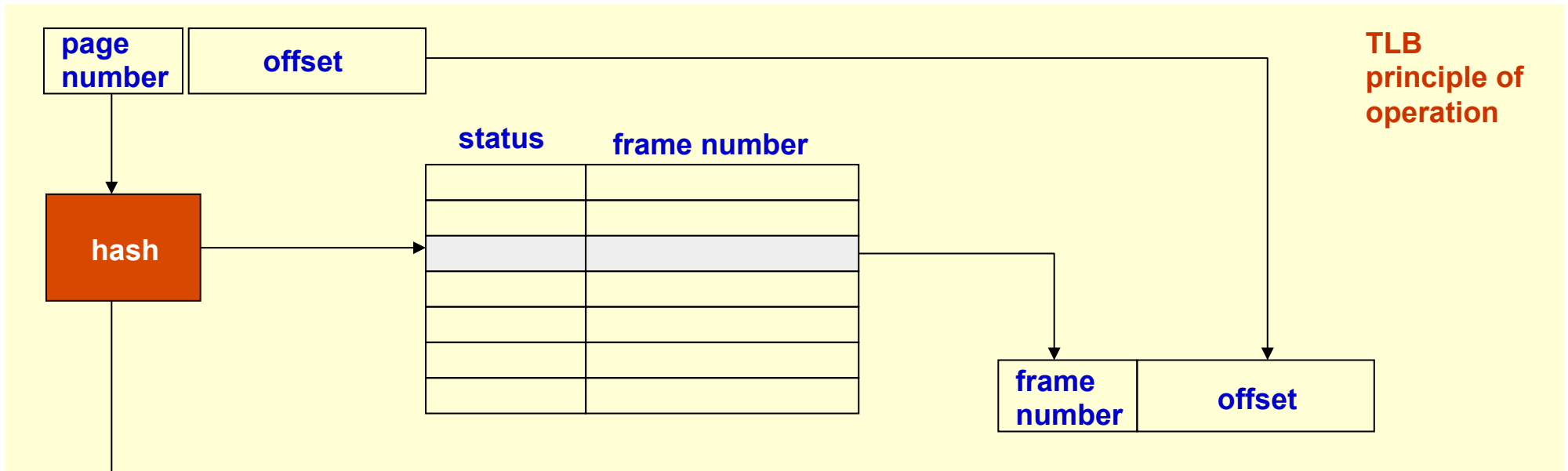
➔ negative Auswirkung auf Leistung !

➔ benötigt Hardwareunterstützung zur Beschleunigung.



Translation Lookaside Buffer

The Translation Lookaside Buffer TLB (sometimes also termed Address Translation Cache: ATC) is a cache for page or segment descriptors. It is in the critical address path of a CPU to memory including, in most cases, cache memory and is accessed with every memory instruction.



miss: exception handled by OS

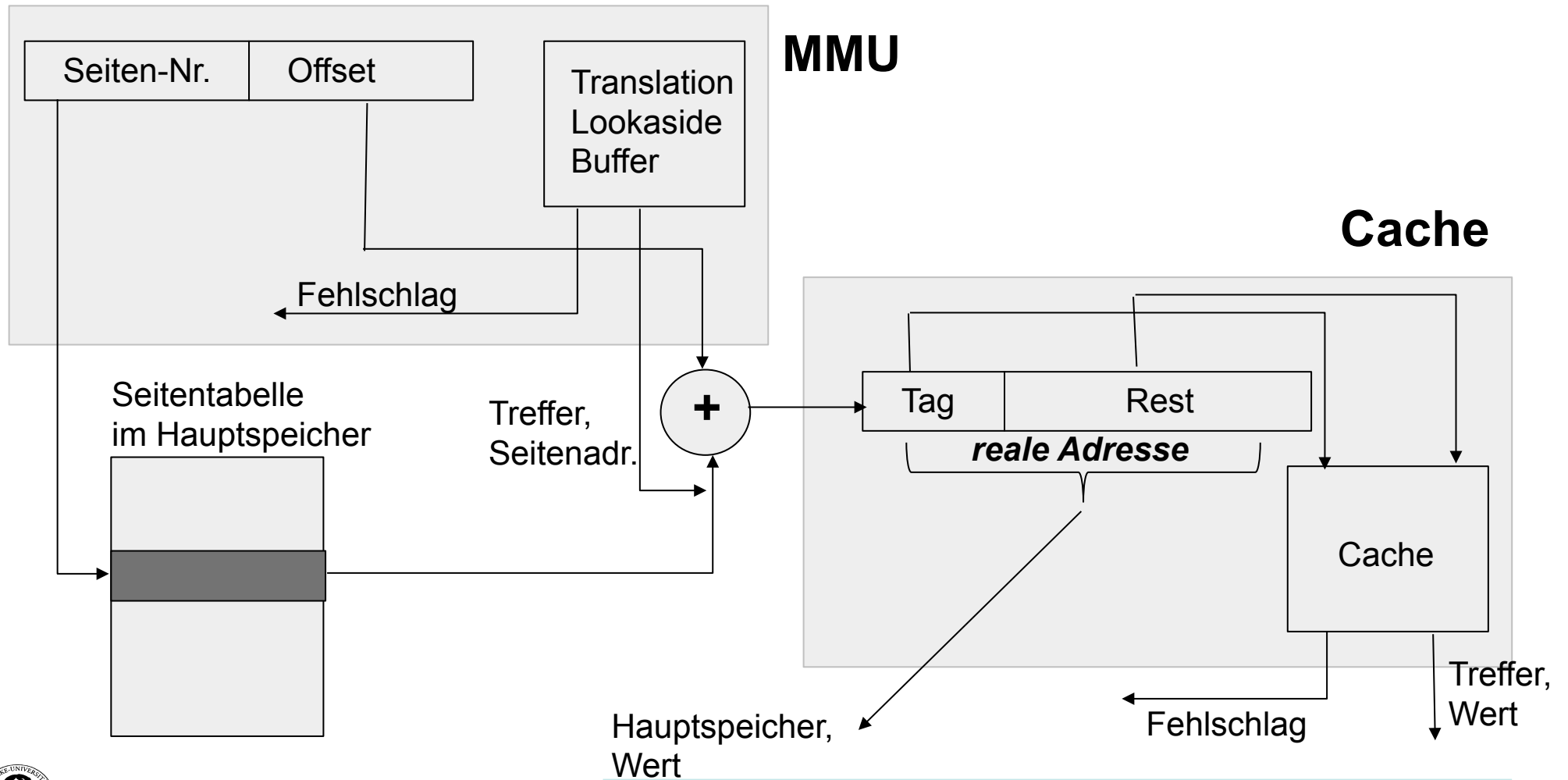
key characteristics:

size of TLB: 32 -4096 entries
hash: full associative, set associative
hit rate (typical): 99% - 99,99 %
hit penalty: ~1 cycle
miss penalty: 10 - 30 cycles

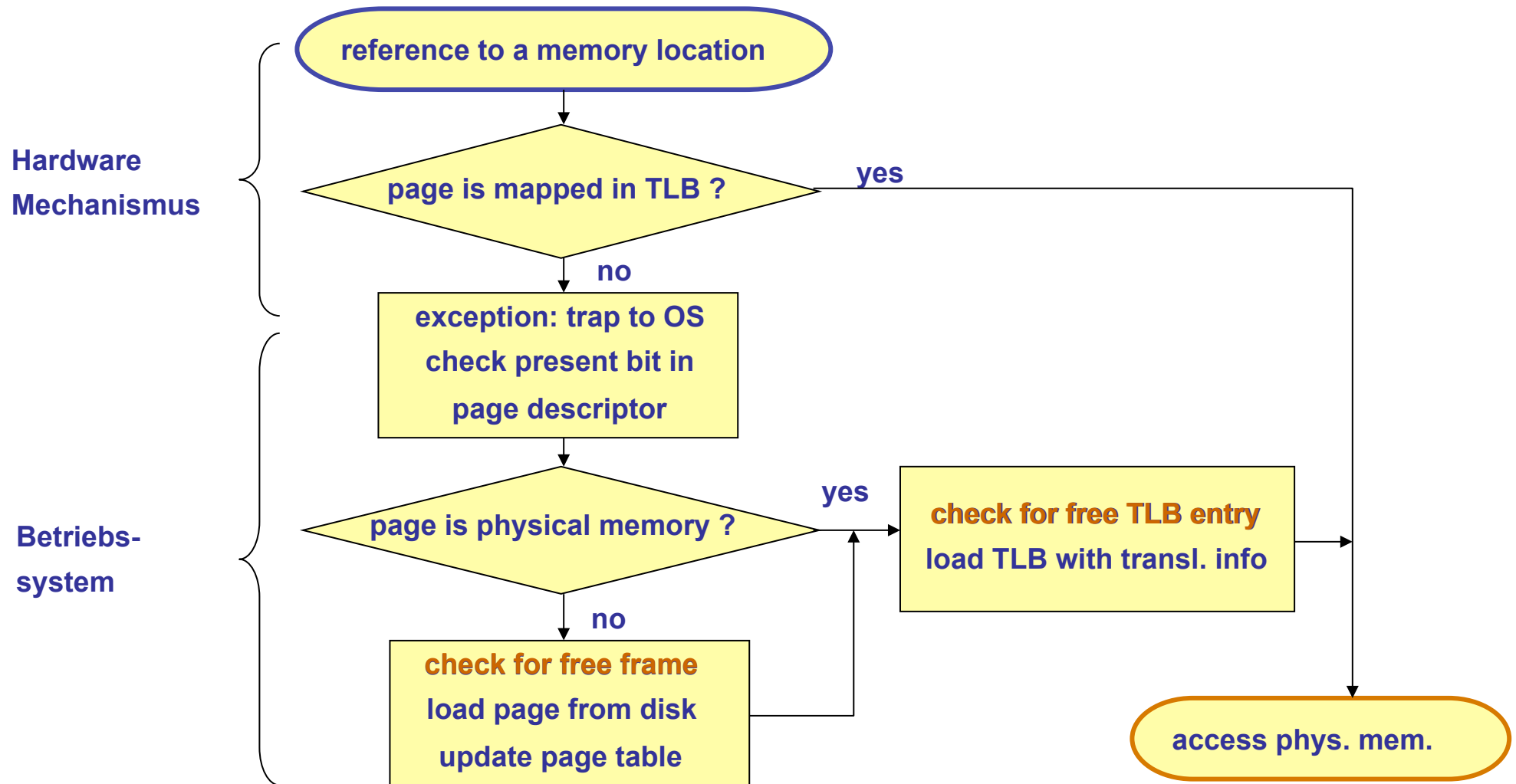


MMU und Cache

virtuelle Adresse von CPU



Demand Paging



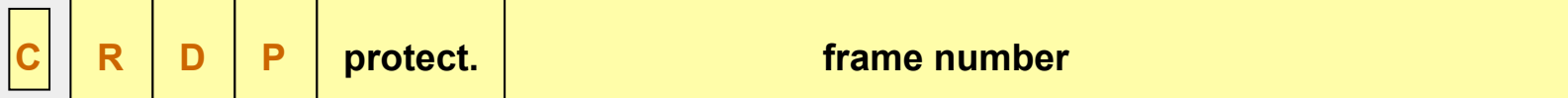
Seitenersetzungsverfahren

Das Betriebssystem muss zukünftige Anforderungen vorhersagen. Dazu kann es nur die Analyse der Gegenwart und Vergangenheit ausnutzen.

- Wann wurde die Seite eingelagert?
- Wurde auf die Seite zugegriffen?
- Wurde die Seite modifiziert?
- Welche Prozesse sind zur Zeit aktiv?



Seiten-
Deskriptor



C: Caching, R: Referenced, D: Dirty (modified), P: Present



Optimale Seitenersetzungsstrategien

ref. sequence		1	2	3	4	1	2	5	1	2	3	4	5
frame assignment in phys. memory	frame 1	1	1	1	1	1	1	1	1	1	3	4	4
	frame 2		2	2	2	2	2	2	2	2	2	2	2
	frame 3			3	4	4	4	5	5	5	5	5	5
control state: distance to next reference	frame 1	4	3	2	1	3	2	1	∞	∞	∞	∞	∞
	frame 2	∞	4	3	2	1	3	2	1	∞	∞	∞	∞
	frame 3	∞	∞	7	7	6	5	5	4	3	2	2	∞
		P	P	P	P			P			P	P	

3 Kacheln
7 Seitenfehler

ref. sequence		1	2	3	4	1	2	5	1	2	3	4	5
frame assignment in phys. memory	frame 1	1	1	1	1	1	1	1	1	1	1	4	4
	frame 2		2	2	2	2	2	2	2	2	2	2	2
	frame 3			3	3	3	3	3	3	3	3	3	3
	frame 4				4	4	4	5	5	5	5	5	5
control state: distance to next reference	frame 1	4	3	2	1	3	2	1	∞	∞	∞	∞	∞
	frame 2	∞	4	3	2	1	3	2	1	∞	∞	∞	∞
	frame 3	∞	∞	7	6	5	4	3	2	1	∞	∞	∞
	frame 4	∞	∞	∞	7	6	5	5	4	3	2	1	∞
		P	P	P	P	-	-	P	-	-	-	P	-

4 Kacheln
6 Seitenfehler

Seitenersetzungsstrategien

Not-recently-used → unterscheidet 4 Seitenklassen:

Klasse 0: R=0, D=0

Klasse 1: R=0, D=1

Klasse 2: R=1, D=0

Klasse 3: R=1, D=1

NRU ersetzt eine beliebige Seite aus der niedrigsten nichtleeren Klasse

Problem

Ref. Folge		1	2	3	4	1	2	5	1	2	3	4	5
Kachelzuordnung im Realspeicher	frame 1	1	1	1	1	1	1	5	1	1	3	3	5
	frame 2		2	2	2	2	2	2	2	2	2	2	5
	frame 3			3	4	4	4	4	4	4	4	4	4
Kontrollstatus: Seitenklasse	frame 1	2	3	3	3	3	2	2	2	0	2	2	2
	frame 2	-	2	2	2	0	2	3	3	1	1	1	3
	frame 3	-	-	2	2	0	2	3	3	1	3	3	3
		P	P	P	P			P	P		P		P

8 page faults



Seitenersetzungsstrategien

FIFO: Ersetzt die Seite, die am längsten im Speicher ist.

Ref. Folge		1	2	3	4	1	2	5	1	2	3	4	5
Kachelzuordnung im Realspeicher	frame 1	1	1	1	4	4	4	5	5	5	5	5	5
	frame 2		2	2	2	1	1	1	1	1	3	3	3
	frame 3			3	3	3	2	2	2	2	2	4	4
Kontrollstatus: Alter der Seite	frame 1	0	1	2	0	1	2	0	1	2	3	4	5
	frame 2	-	0	1	2	0	1	2	3	4	0	1	2
	frame 3	-	-	0	1	2	0	1	2	3	4	0	1
		P	P	P	P	P	P	P			P	P	

9 Seitenfehler



Seitenersetzungsstrategien

FIFO: Ersetzt die Seite, die am längsten im Speicher ist.

Ref. Folge		1	2	3	4	1	2	5	1	2	3	4	5
Kachelzuordnung im Realspeicher	frame 1	1	1	1	4	4	4	5	5	5	5	5	5
	frame 2		2	2	2	1	1	1	1	1	3	3	3
	frame 3			3	3	3	2	2	2	2	2	4	4
Kontrollstatus: Alter der Seite	frame 1	0	1	2	0	1	2	0	1	2	3	4	5
	frame 2	-	0	1	2	0	1	2	3	4	0	1	2
	frame 3	-	-	0	1	2	0	1	2	3	4	0	1
		P	P	P	P	P	P	P			P	P	

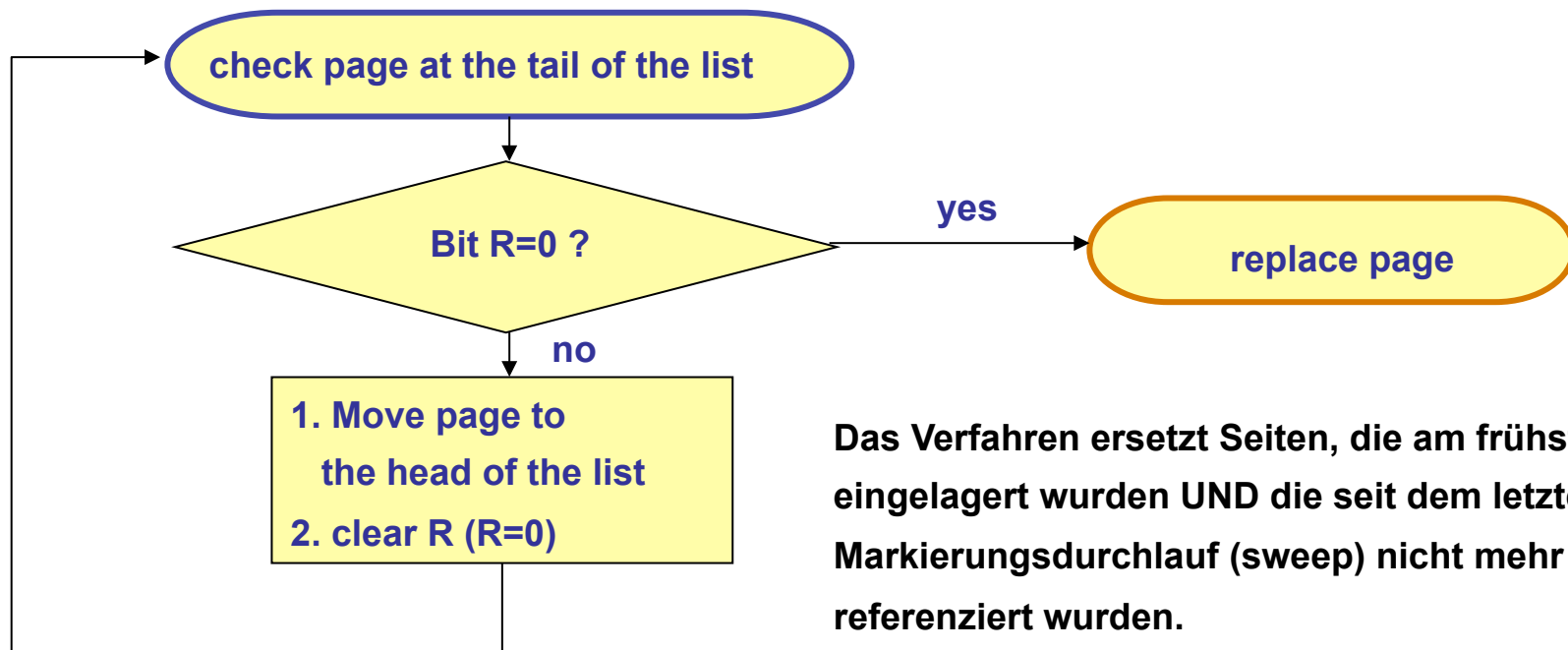
9 Seitenfehler



Seitenersetzungsstrategien

Variation von FIFO: Der "Second Chance" Algorithmus

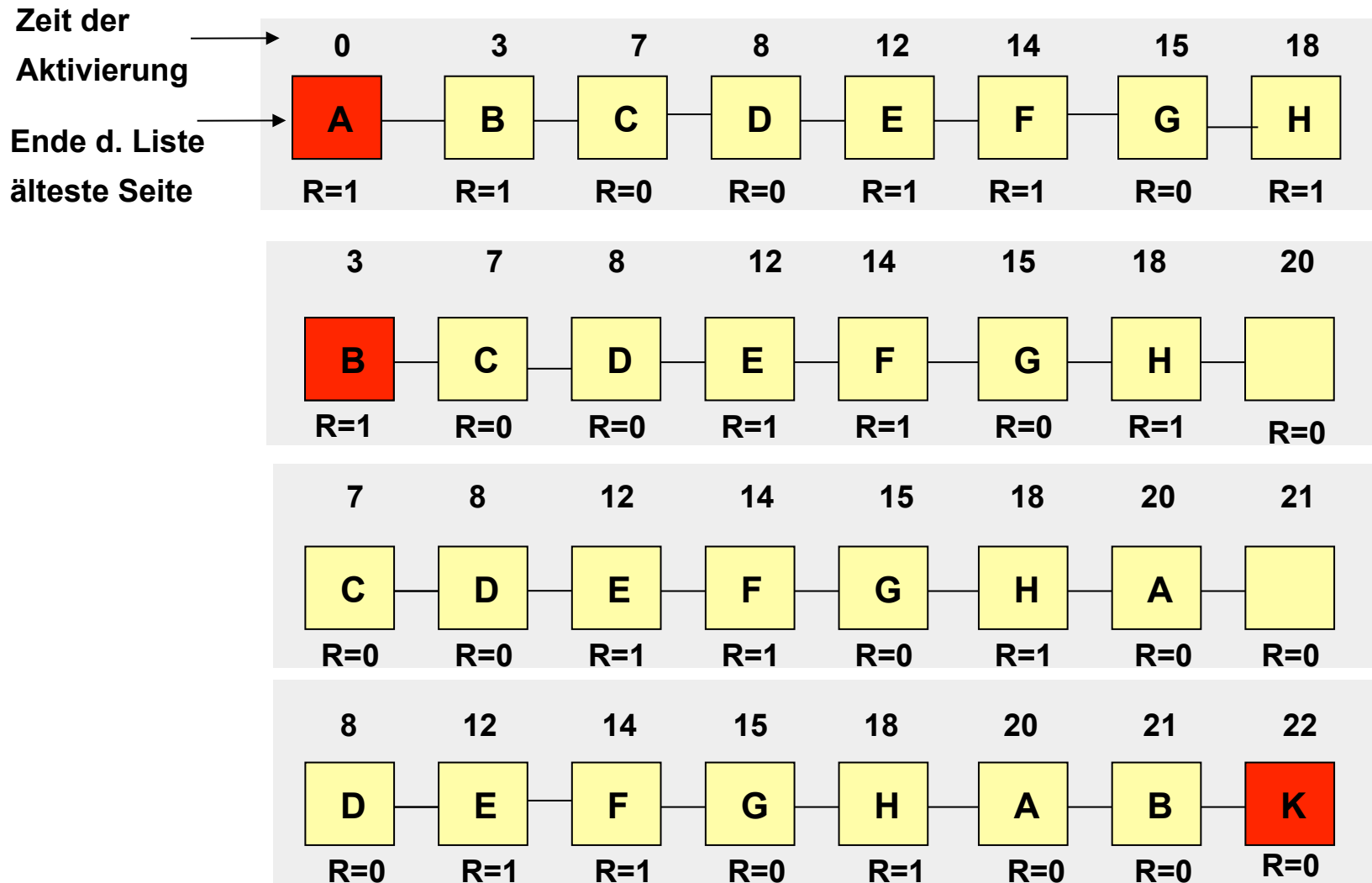
Die Seiten sind in einer Liste nach FIFO geordnet



Das Verfahren ersetzt Seiten, die am frühesten eingelagert wurden UND die seit dem letzten Markierungsdurchlauf (sweep) nicht mehr referenziert wurden.



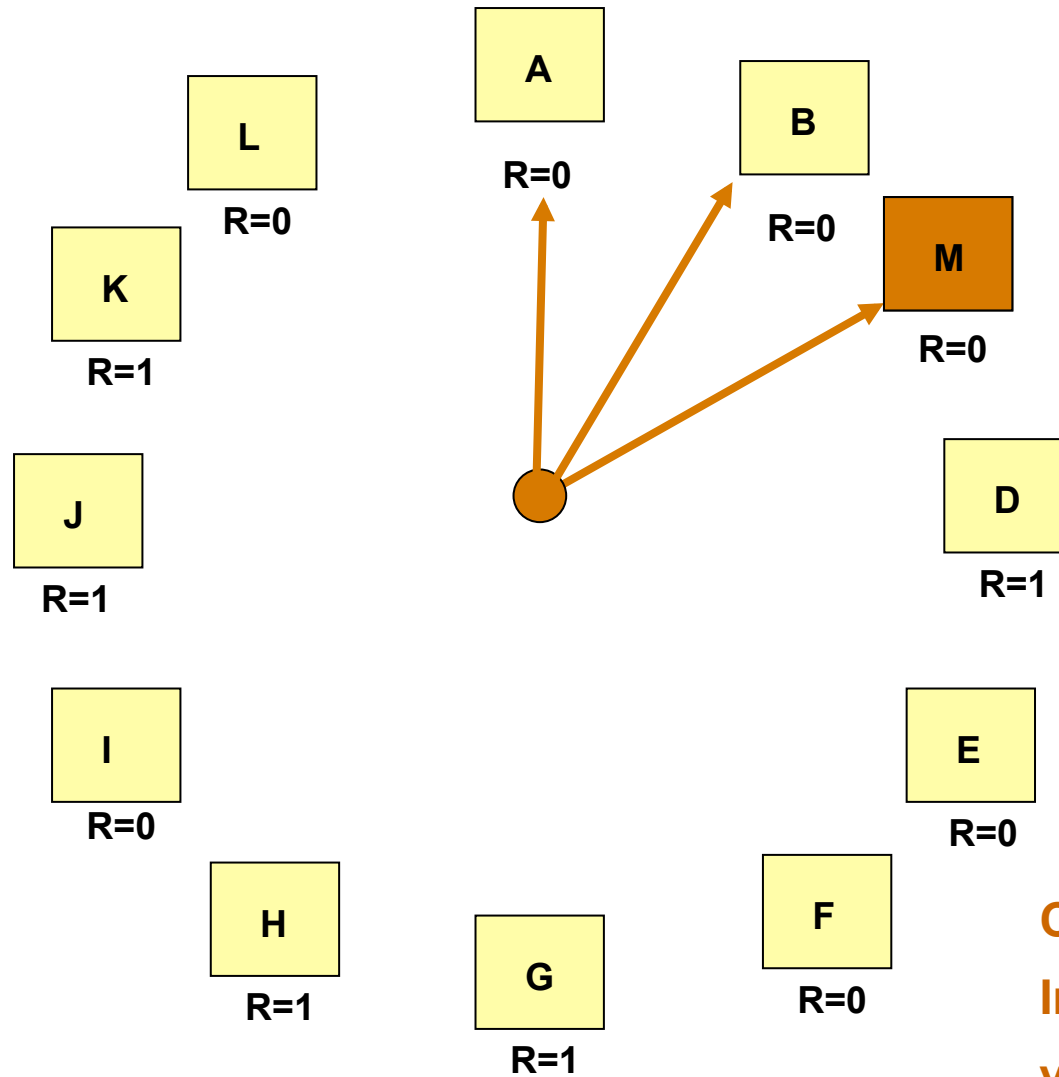
Der "Second Chance" Algorithmus



Seiten sind nach FIFO geordnet



Implementierung von "second chance": Der "Clock" Algorithmus

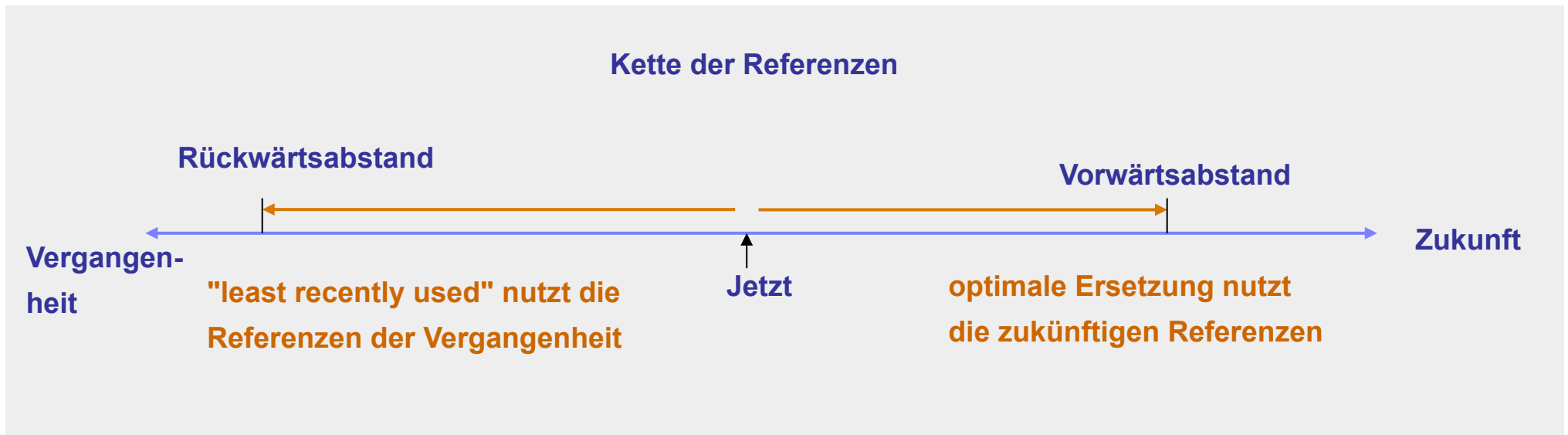


Clock ist eine intelligente Implementierung von "Second Chance"



Seitenersetzungsstrategien

Least-Recently-Used: Verdränge die Seite, die am längsten nicht referenziert wurde.



Problem: Least Recently Used ist schwer zu implementieren !



Seitenersetzungsstrategien

Least-Recently-Used

Seite 1 wird referenziert

ref. sequence		1	2	3	4	1	2	5	1	2	3	4	5
frame assignment in phys. memory	frame 1	1	1	1	1	1	1	1	1	1	1	1	5
	frame 2		2	2	2	2	2	2	2	2	2	2	2
	frame 3			3	3	3	3	5	5	5	5	4	4
	frame 4				4	4	4	4	4	4	3	3	3
control state: backward distance	frame 1	0	1	2	3	0	1	2	0	1	2	3	0
	frame 2	-	0	1	2	3	0	1	2	0	1	2	3
	frame 3	-	-	0	1	2	3	0	1	2	3	0	1
	frame 4	-	-	-	0	1	2	3	4	5	0	1	2

P P P P

P

P

P

7 page faults



Optimale Seitenersetzungsstrategie

zum Vergleich:

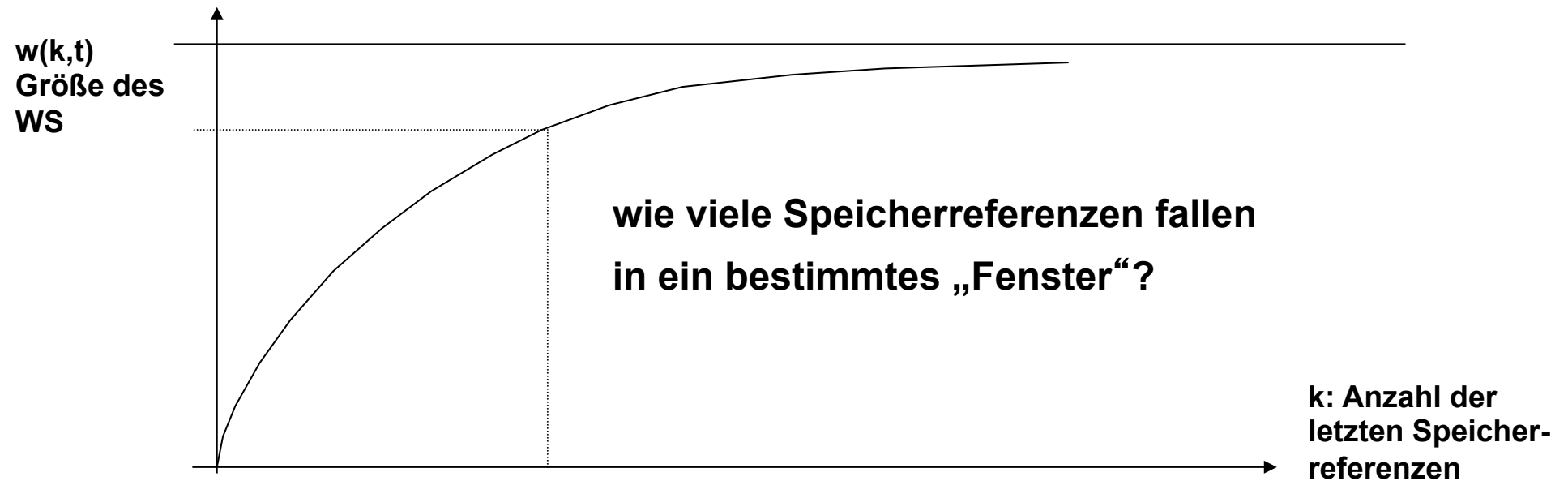
ref. sequence		1	2	3	4	1	2	5	1	2	3	4	5
frame assignment in phys. memory	frame 1	1	1	1	1	1	1	1	1	1	1	4	4
	frame 2		2	2	2	2	2	2	2	2	2	2	2
	frame 3			3	3	3	3	2	3	3	3	3	3
	frame 4				4	4	4	5	5	5	5	5	5
control state: distance to next reference	frame 1	4	3	2	1	3	2	1	∞	∞	∞	∞	∞
	frame 2	∞	4	3	2	1	3	2	1	∞	∞	∞	∞
	frame 3	∞	∞	7	6	5	4	3	2	1	∞	∞	∞
	frame 4	∞	∞	∞	7	6	5	5	4	3	2	1	∞
		P	P	P	P	-	-	P	-	-	-	P	-

4 Kacheln
6 Seitenfehler



Seitenersetzungsstrategien

Der "Working Set" Algorithmus



Die Menge der Seiten, die von einem Prozess in einem bestimmten Zeitfenster benutzt werden, wird als Working Set (WS) bezeichnet.

Peter Denning: The Working Set Model for Program Behaviour, CACM, May 1968



Der WS Algorithmus

Verdrängungsalgorithmus:

- durchlaufe alle S.-Deskriptoren
- if $R=1$: set vt to cvt and set $R=0$;
- if $R=0 \wedge (cvt - vt) > t$:
verdränge die Seite;
- if $R=0 \wedge (cvt - vt) < t$:
keine Änderung;
- wenn keine Seite gefunden wird mit:
 $R=0 \wedge (cvt - vt) < t$ then
verdränge älteste Seite;
- wenn alle Seiten referenziert wurden
verdränge beliebige Seite.

Seitendeskriptor

Seitentabelle	
2083	1
2003	1
1981	1
1620	0
2014	1
2020	1
2032	1
1160	0
.....	

R-Bit

virtual time: vt

Das Feld enthält die Zeit des letzten Zugriffs auf die Seite.

"virtual time" ist eine Prozess-lokale Repräsentation der Zeit, die mit Prozessbeginn startet.

current virtual time: cvt

2204

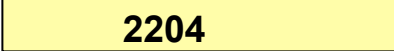
$t = 800$

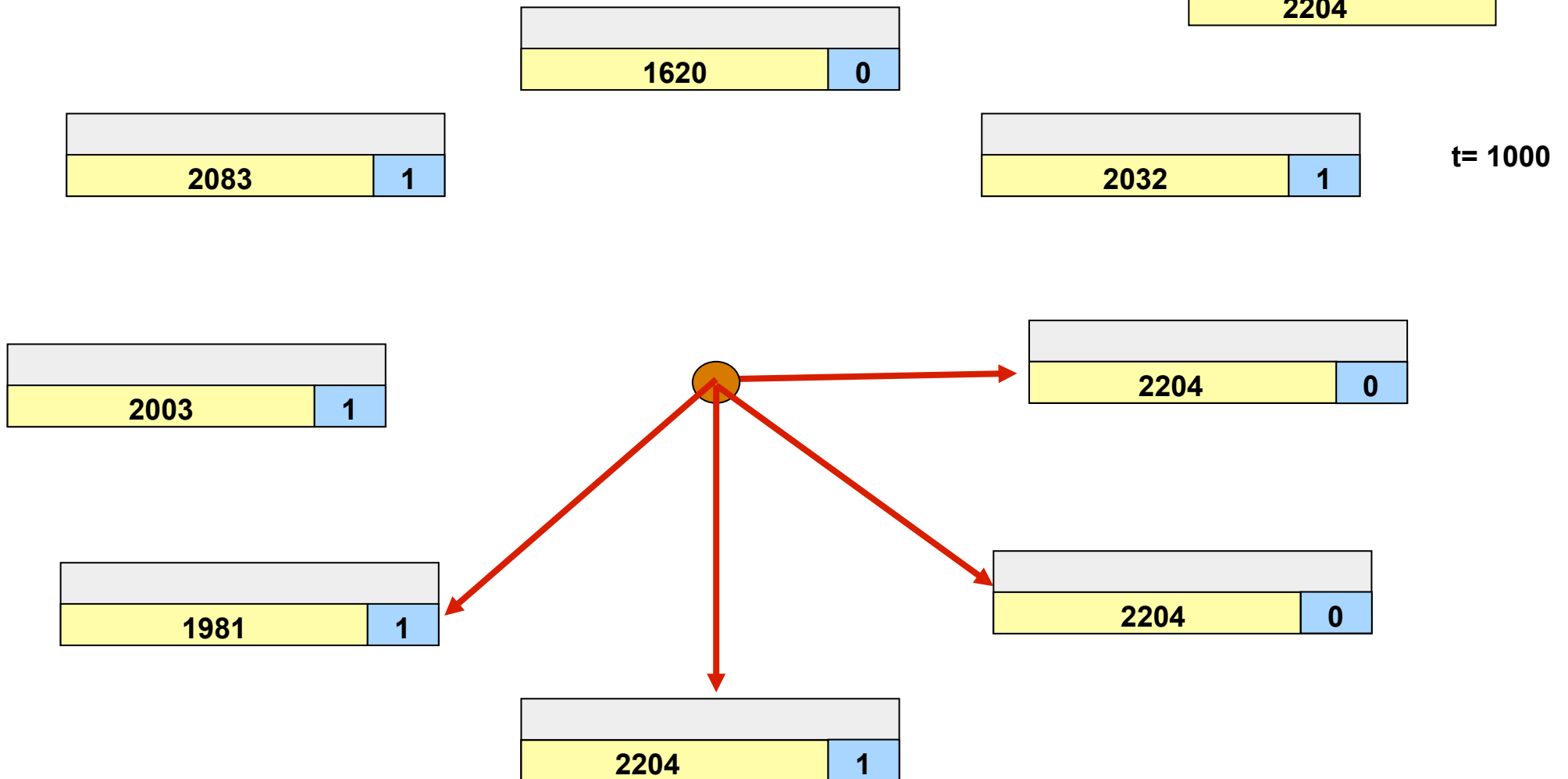


Der WS-Clock Algorithmus

 R-Bit

current virtual time: cvt

 2204



page replacement policies summary

Algorithmus	Eigenschft.	impl.	Kommentar
optimal	😬 😬 😬	😬 😬 😬	nur zum Vergleich, kann nicht realisiert werden
NRU:	😬	😬 😬	einfach und einfach zu realisieren
FIFO	😬	😬 😬	einfach; Problem: wichtige (alte) Seiten werden verdrängt
2nd chance:	😬 😬	😬	substantielle Verbesserung von FIFO
Clock:	😬 😬	😬 😬	intelligente Implementierung von 2nd Chance
LRU:	😬 😬 😬	😬 😬	exzellent, aber schwierig in der Implementierung
WS:	😬 😬 😬	😬	gut, Implementierungsprobleme, prozesslokale Strategie
WSClock:	😬 😬 😬	😬 😬	gut + effizient, prozesslokale Strategie



Seitenersetzungsstrategien

FIFO: Ersetzt die Seite, die am längsten im Speicher ist.

Ref. Folge		1	2	3	4	1	2	5	1	2	3	4	5
Kachelzuordnung im Realspeicher	frame 1	1	1	1	4	4	4	5	5	5	5	5	5
	frame 2		2	2	2	1	1	1	1	1	3	3	3
	frame 3			3	3	3	2	2	2	2	2	4	4
Kontrollstatus: Alter der Seite	frame 1	0	1	2	0	1	2	0	1	2	3	4	5
	frame 2	-	0	1	2	0	1	2	3	4	0	1	2
	frame 3	-	-	0	1	2	0	1	2	3	4	0	1
		P	P	P	P	P	P	P			P	P	

9 Seitenfehler



Seitenersetzungsstrategien

FIFO: Belady's Anomalie

Ref. Folge		1	2	3	4	1	2	5	1	2	3	4	5
Kachelzuordnung im Realspeicher	frame 1	1	1	1	1	1	1	5	5	5	5	4	4
	frame 2		2	2	2	2	2	2	1	1	1	1	5
	frame 3			3	3	3	3	3	3	2	2	2	2
	frame 4				4	4	4	4	4	4	3	3	3
Kontrollstatus: Alter der Seite	frame 1	0	1	2	3	4	5	0	1	2	3	0	1
	frame 2	-	0	1	2	3	4	5	0	1	2	3	0
	frame 3	-	-	0	1	2	3	4	5	0	1	2	3
	frame 4	-	-	-	0	1	2	3	4	5	0	1	2

P P P P - - P P P P P P 10 Seitenfehler

Obwohl mehr Kacheln vorhanden sind, werden mehr Seitenfehler erzeugt!
 Grundsätzliches Problem: FIFO berücksichtigt nicht die Nutzung einer Seite.



the class of stack algorithms

ref. string

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

pages in
phys. memory

m

pages out of
phys. memory

n-m

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1	
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4	
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3	
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7	
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

p p p p p p p p



p



distance string

∞	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6	1	1	4	2	3	5	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

distance:= distance to TOS

important properties of stack algorithms: $M(m,r) \subseteq M(m+1,r)$

m: # of frames, r: distance index



predicting page fault rate

ref. string

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

distance string

	∞	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6	1	1	4	2	3	5	3
--	----------	----------	----------	----------	----------	----------	----------	---	----------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$C_1 \dots C_\infty$

distance values:	1	2	3	4	5	6	7	8	∞
occurrence:	4	3	3	3	2	1	0	0	8

$C_1 = 4$
$C_2 = 3$
$C_3 = 3$
$C_4 = 3$
$C_5 = 2$
$C_6 = 1$
$C_7 = 0$
$C_8 = 0$
$C_\infty = 8$

page fault rate F for n pages and m frames

$$F_m = \sum_{k=m+1}^n C_k + C_\infty$$



$F_1 = 20$
$F_2 = 17$
$F_3 = 14$
$F_4 = 11$
$F_5 = 9$
$F_6 = 8$
$F_7 = 8$
$F_8 = 8$
$F_\infty = 8$

$C_2 + \dots + C_\infty$
 $C_3 + \dots + C_\infty$
 $C_4 + \dots + C_\infty$
 $C_5 + \dots + C_\infty$
 $C_6 + \dots + C_\infty$
 $C_7 + \dots + C_\infty$
 $C_8 + \dots + C_\infty$
 C_∞



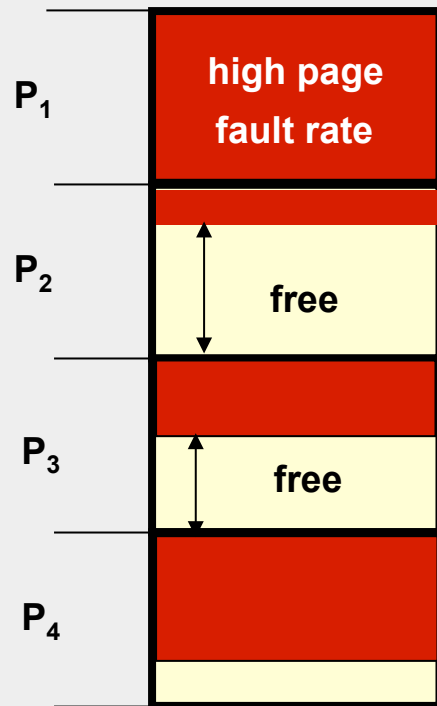
design issues for paging

- ➔ **local vs. global paging policies**
- ➔ **page size**
- ➔ **separating program and data pages**
- ➔ **sharing of pages**
- ➔ **release policies**
- ➔ **transparency issues and interface to the virtual memory**



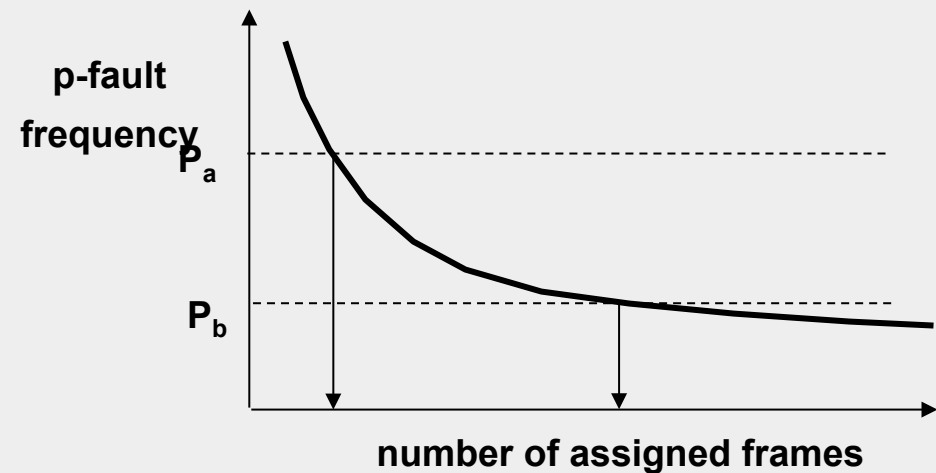
local paging policies

local policy: a fixed budget of frames is assigned to every process.



local policy: a variable budget of frames is assigned to every process.

- ➔ assignment proportional to process size.
- ➔ use distance chain analysis to determine number of frames needed (static).
- ➔ use page fault frequency measurements to determine number of frames (dynamic). (PFF-algorithm)



global paging policies







- ➔ **global policies are more flexible and have more potential to balance memory requests.**
- ➔ **global policies do not work with all page replacement strategies, e.g. a global working set does not make sense.**
- ➔ **OS must prevent monopolization of memory by one or a few processes.**
- ➔ **Swapping processes to disk.**



page size

for n segments with p bytes, the internal fragmentation is: $np/2$

trade-offs:

page size:	large	small
internal fragmentation ($n \cdot p/2$):		
page table size+management overhead		
load time from disk		

Common page sizes are in the range between 512 Bytes and 64KB.

Today, page sizes of 4KB or 8KB are most common.



Examples

Unix "et al."



Fragen:

Wie ist der virtuelle Speicher generell aufgeteilt?

Welche Datenstrukturen gibt es zur virt. Sp. Verwaltung?

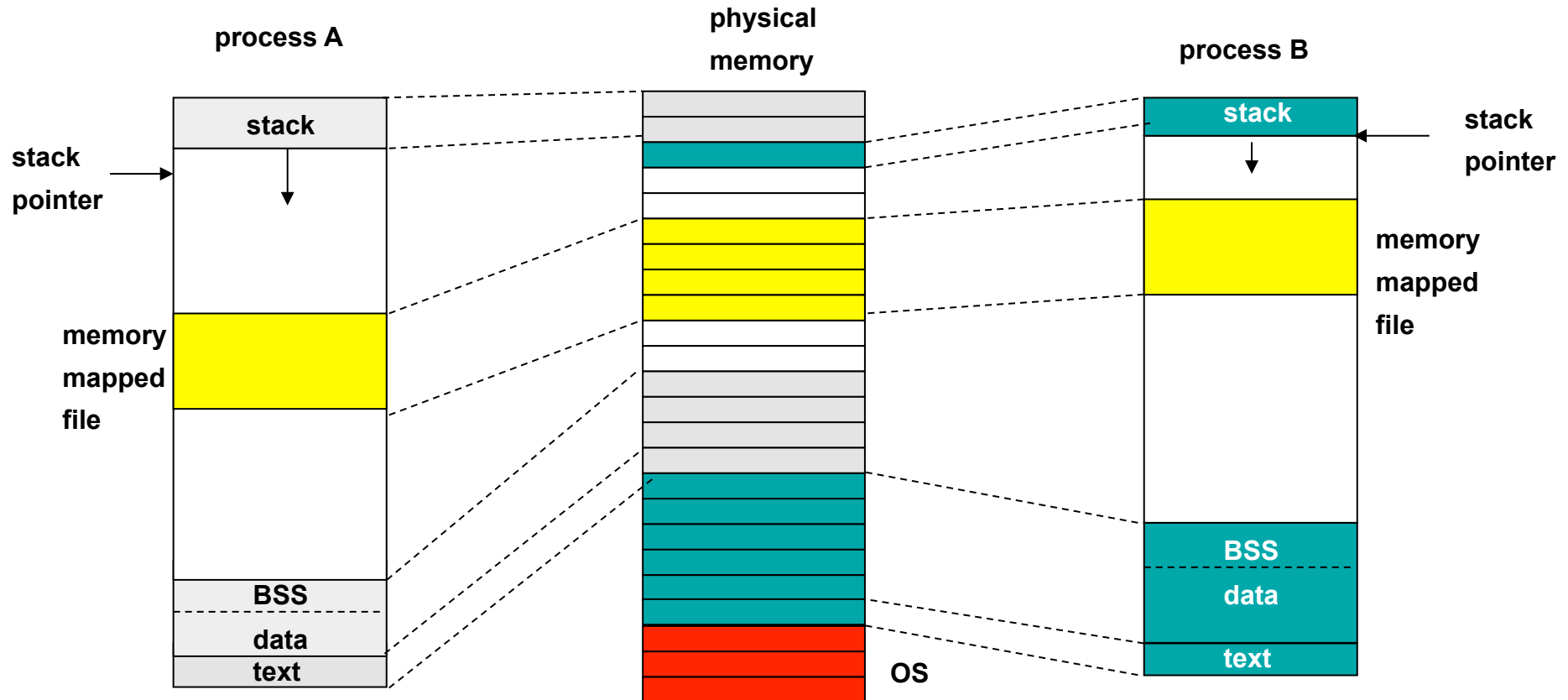
Wie findet man freie Seitenrahmen?

Welche Strukturen werden zur Verwaltung ausgelagerter Seiten bereitgestellt?

Wie wird die Seitenersetzung durchgeführt?



Example: Memory management in Unix et al.



- supports processor architectures which separate program (read-only text) and data.
- stack contains context variables which define the execution environment for the process.



Example: Memory management in Unix et al.

Paging system

Data structures in System V Release 4 (SVR4):

Page table

Disk block-descriptor table

Frame data table

Swap-use-table

Page table entry:

frame number	age	copy on write	D	R	V	protect.
--------------	-----	---------------	---	---	---	----------

Disk block-descriptor:

swap device number	dev. block number	storage type
--------------------	-------------------	--------------

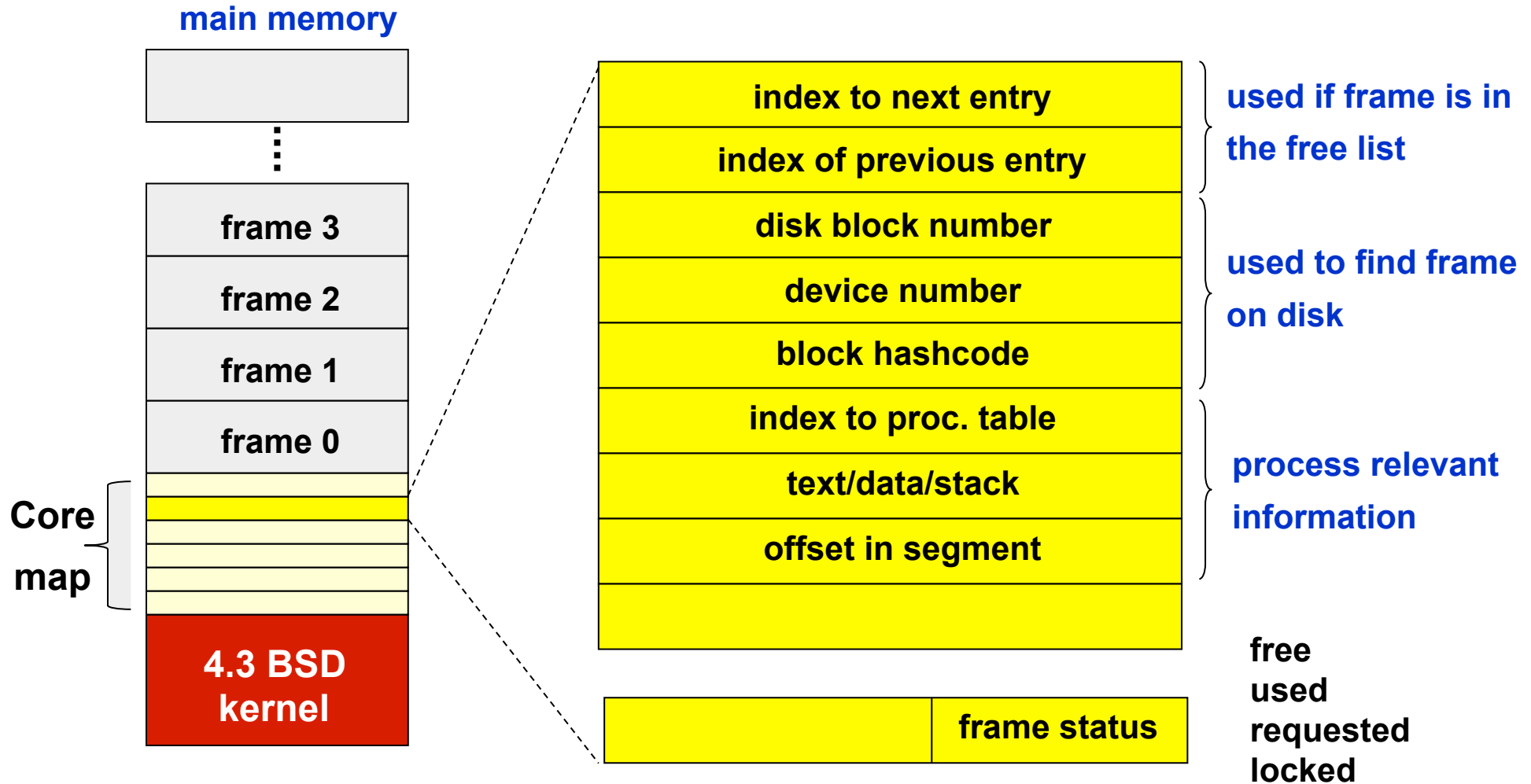
frame data table entry:

fr. status	ref.cnt	log. device	block no.	pointer
------------	---------	-------------	-----------	---------

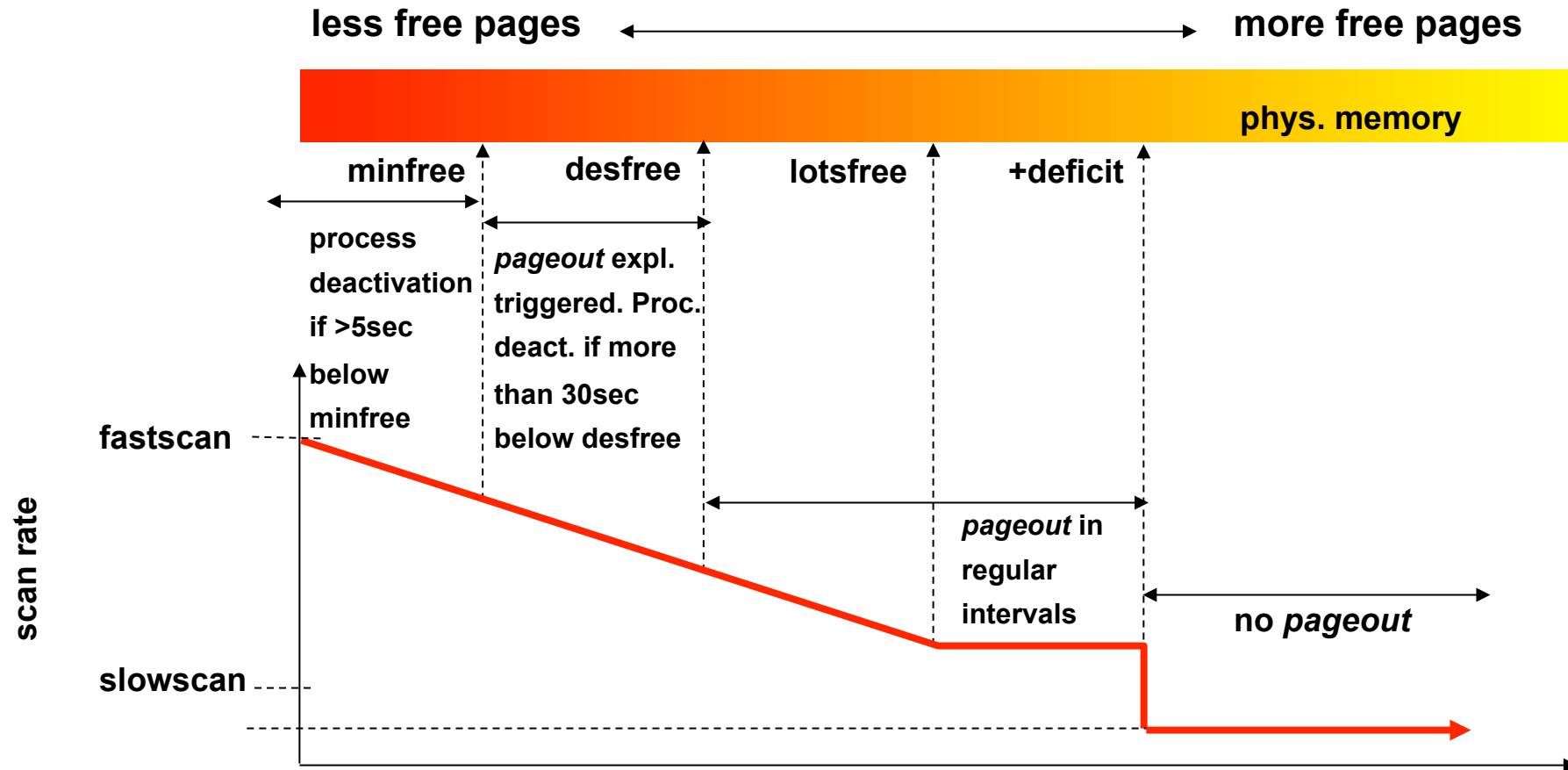
- other frame table pointers
- list of free frames
- hash queue



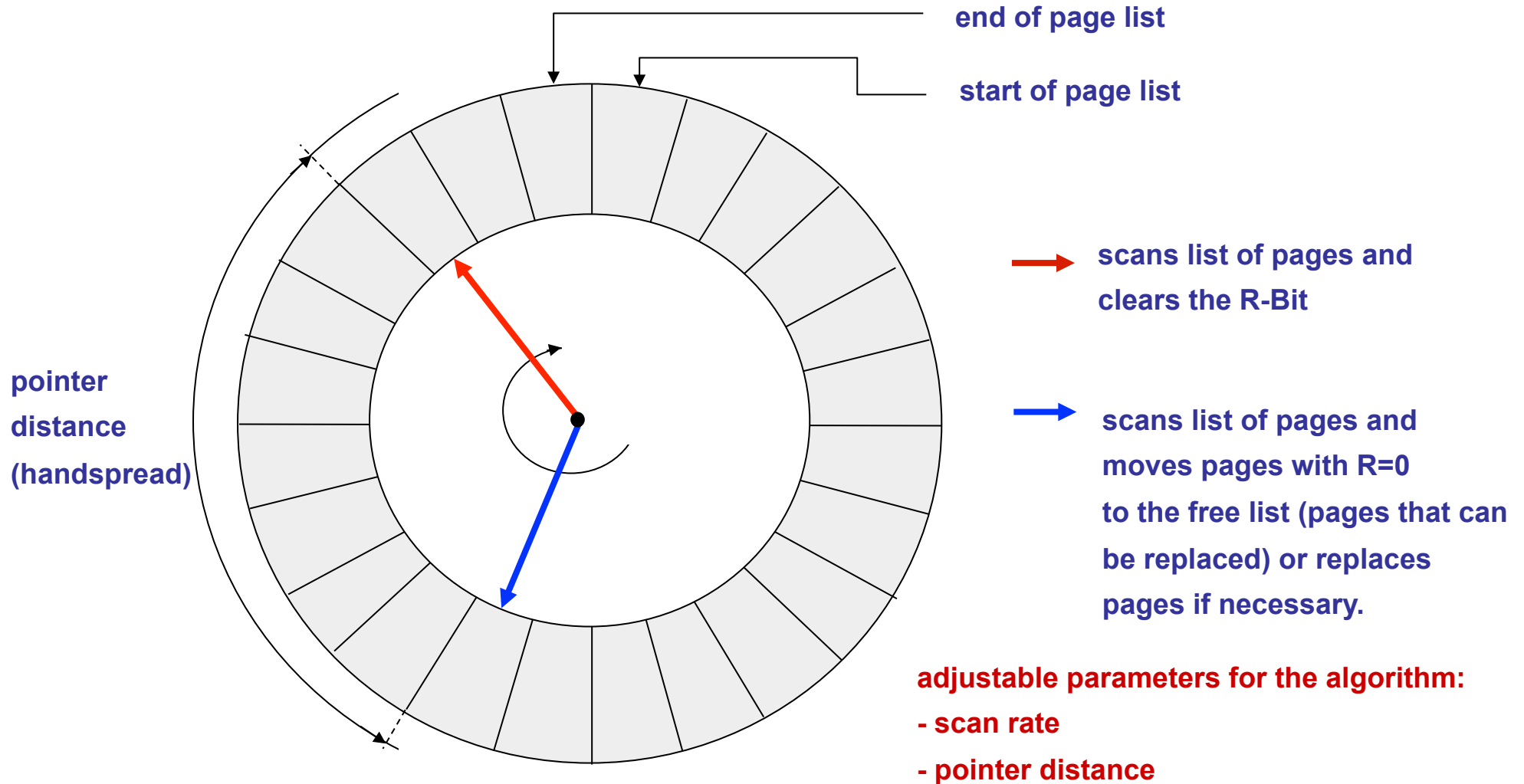
Beispiel: Organisation des HS in 4BSD



Memory management in Unix et al.



releasing pages: clock with two pointers



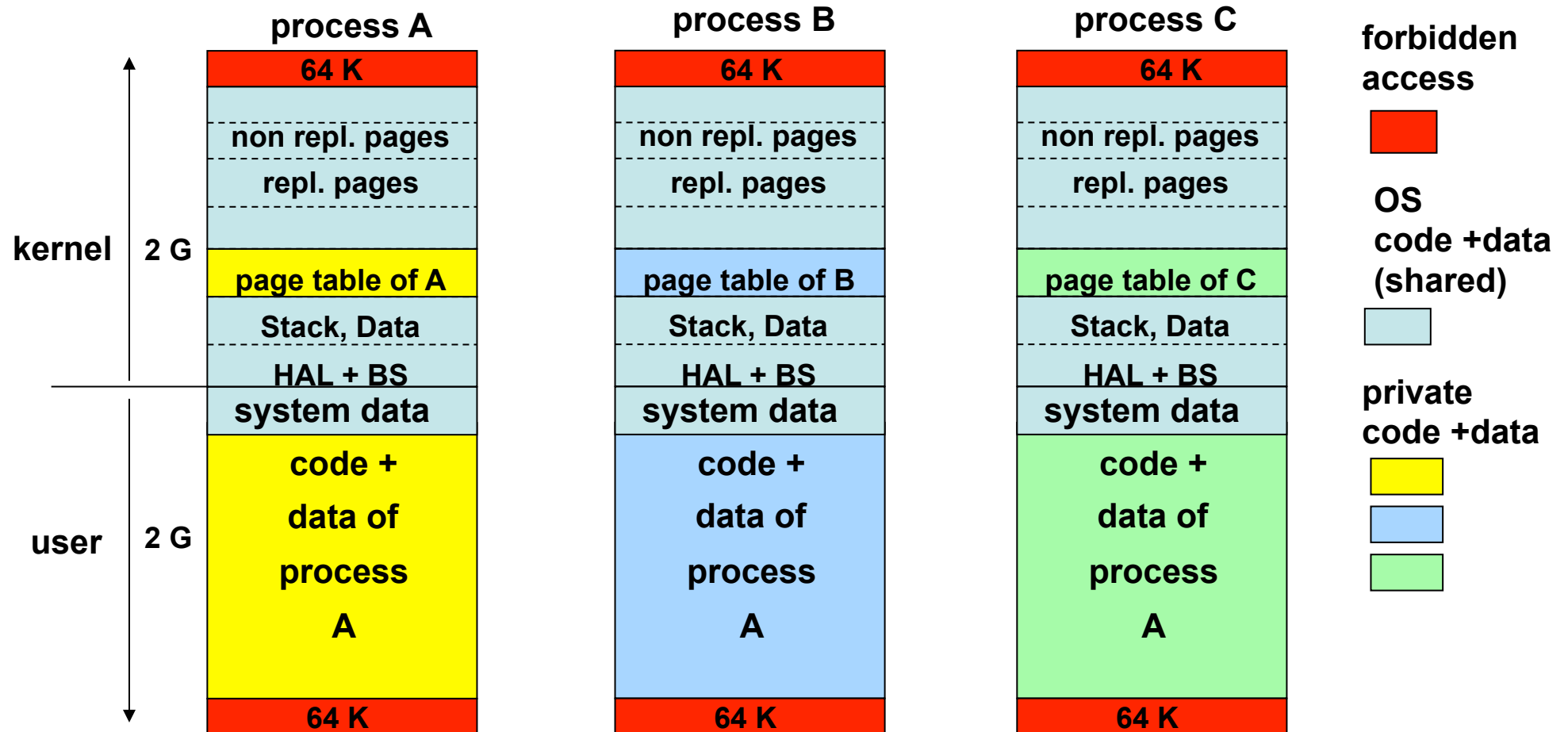
Examples

Windows 2000

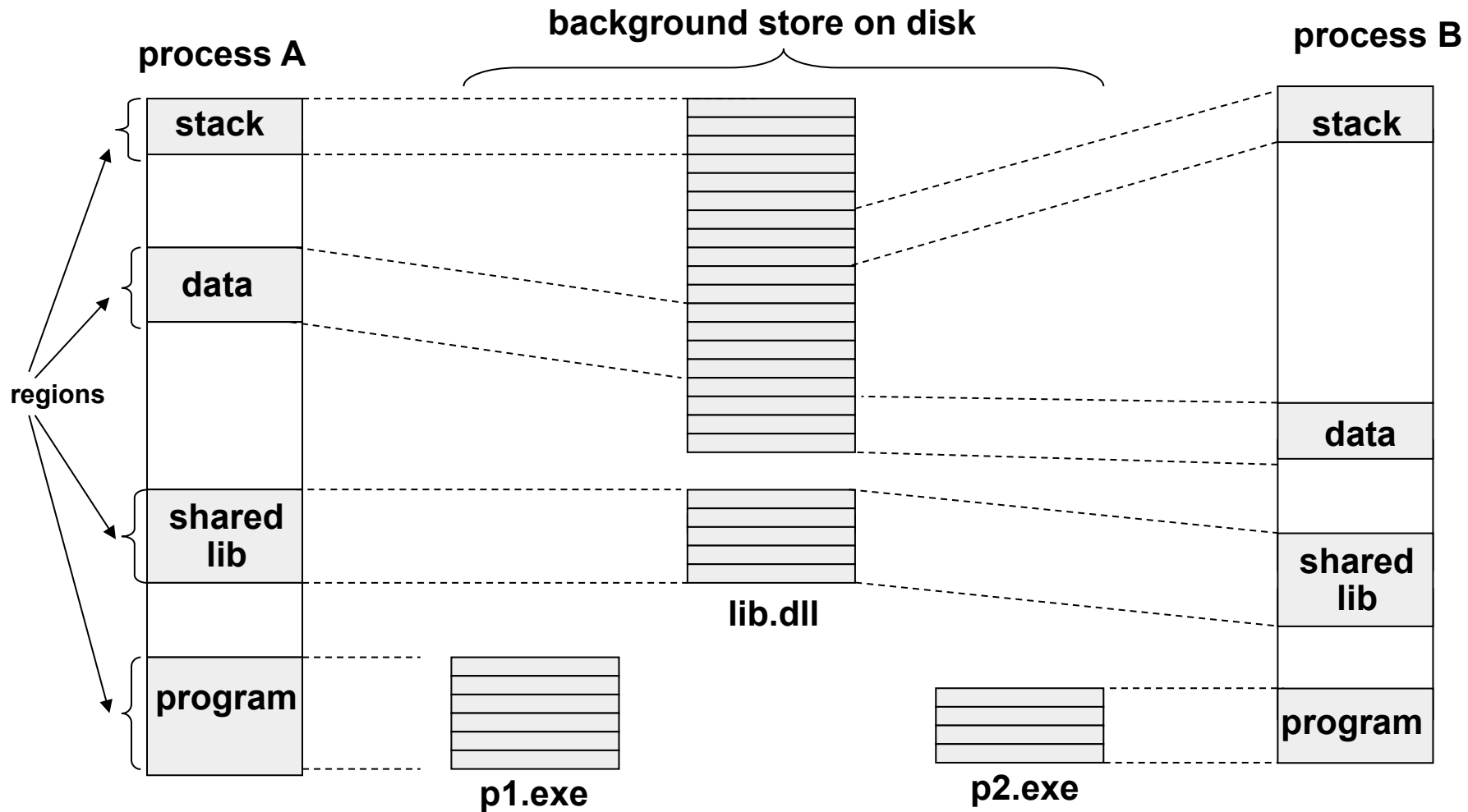


Example: Memory management in Windows 2k

Every process has its own virtual address space of 4 GB



Example: Memory management in Windows 2k



Example: Memory management in Windows 2k

Page replacement:

Basic algorithm: working set (local replacement policy!)

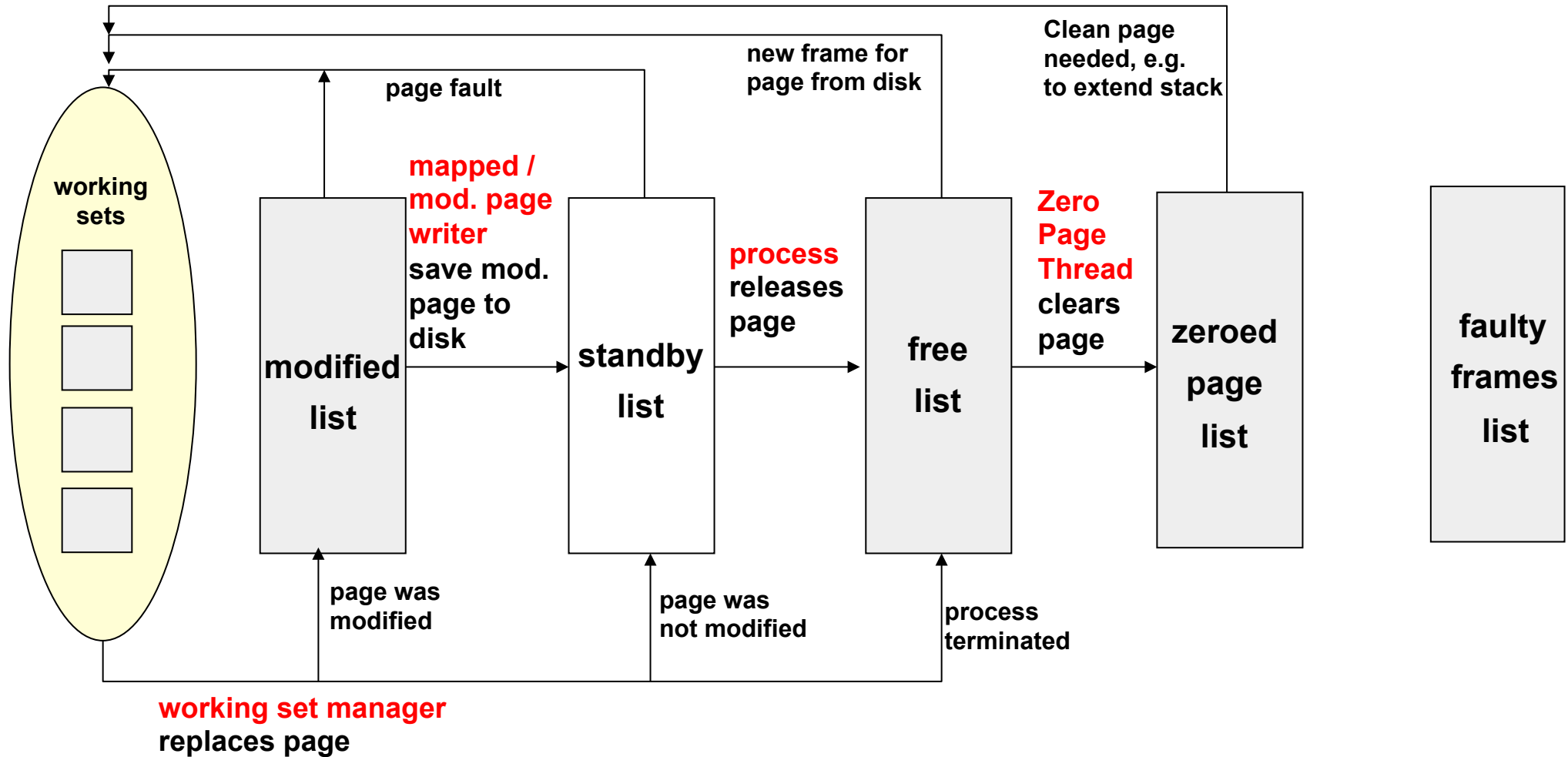
parameter: min (20..50), max (45..345)

Balance set manager: checks for enough free pages.

Working set manager: checks the WS for replacable pgs.



Example: Memory management in Windows 2k



Zusammenfassung

Ziele: unendlich {groß, schnell, billig}, -->Zielkonflikte

Verwaltung des physischen Speicher:

Interne und externe Fragmentierung

Lokale und globale Listenverwaltung

Buddy System

Virtueller Speicher:

Segmentierung: + private Adreßräume, Schutz
- Segment muss in physischen Speicher passen

Seitenorientierung: + riesiger linearer Adreßraum
- Schutz, Problem der Seitentabellen

Speicherverwaltung:

Problem der sehr hohen Trefferrate, Lokalitätsprinzip

Seitenersetzungsstrategien und Vorhersage der phys. Speichergröße

Beispiele zur Speicherverwaltung in Unix und Windows

