Speicherverwaltung

Betriebssysteme
WS 2006/2007

Jörg Kaiser
IVS – EOS

Otto-von-Guericke-Universität Magdeburg

Speicherverwaltung

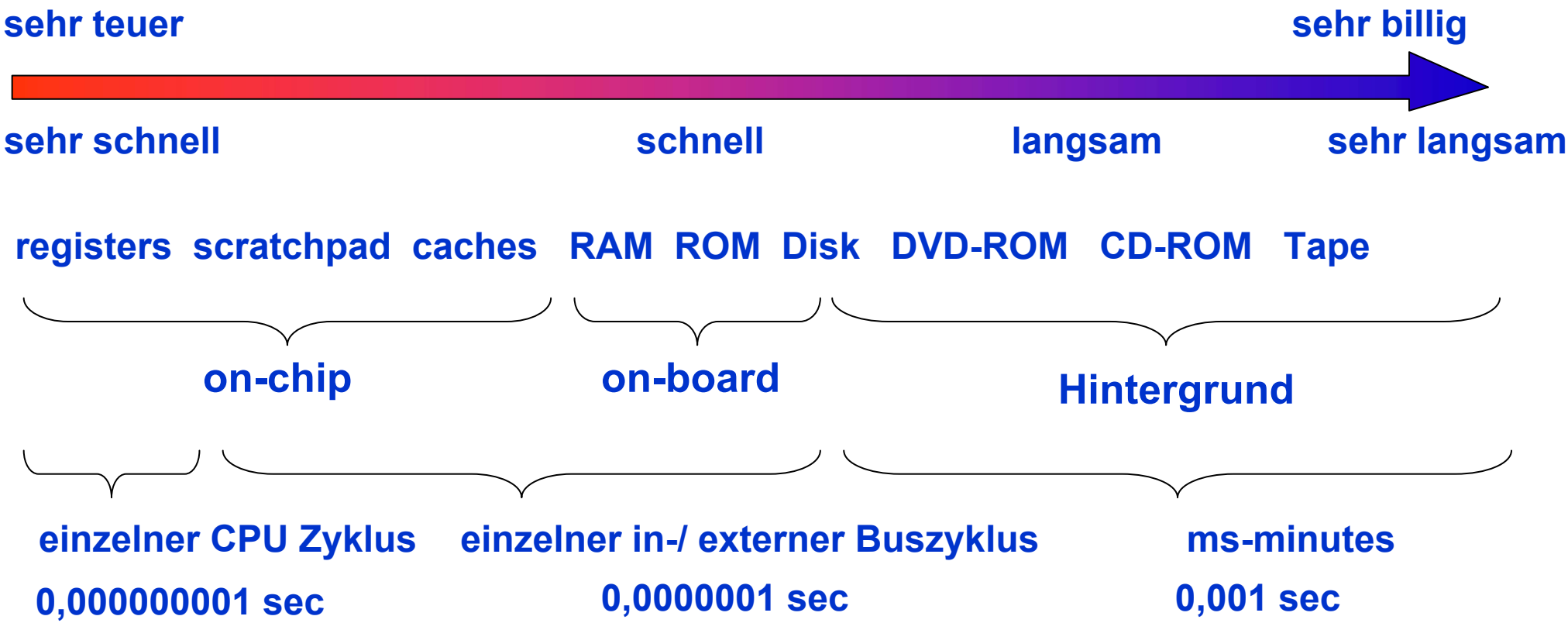
Wozu braucht man eine Speicherverwaltung?
Welche Eigenschaften sollte ein Speicher haben?

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

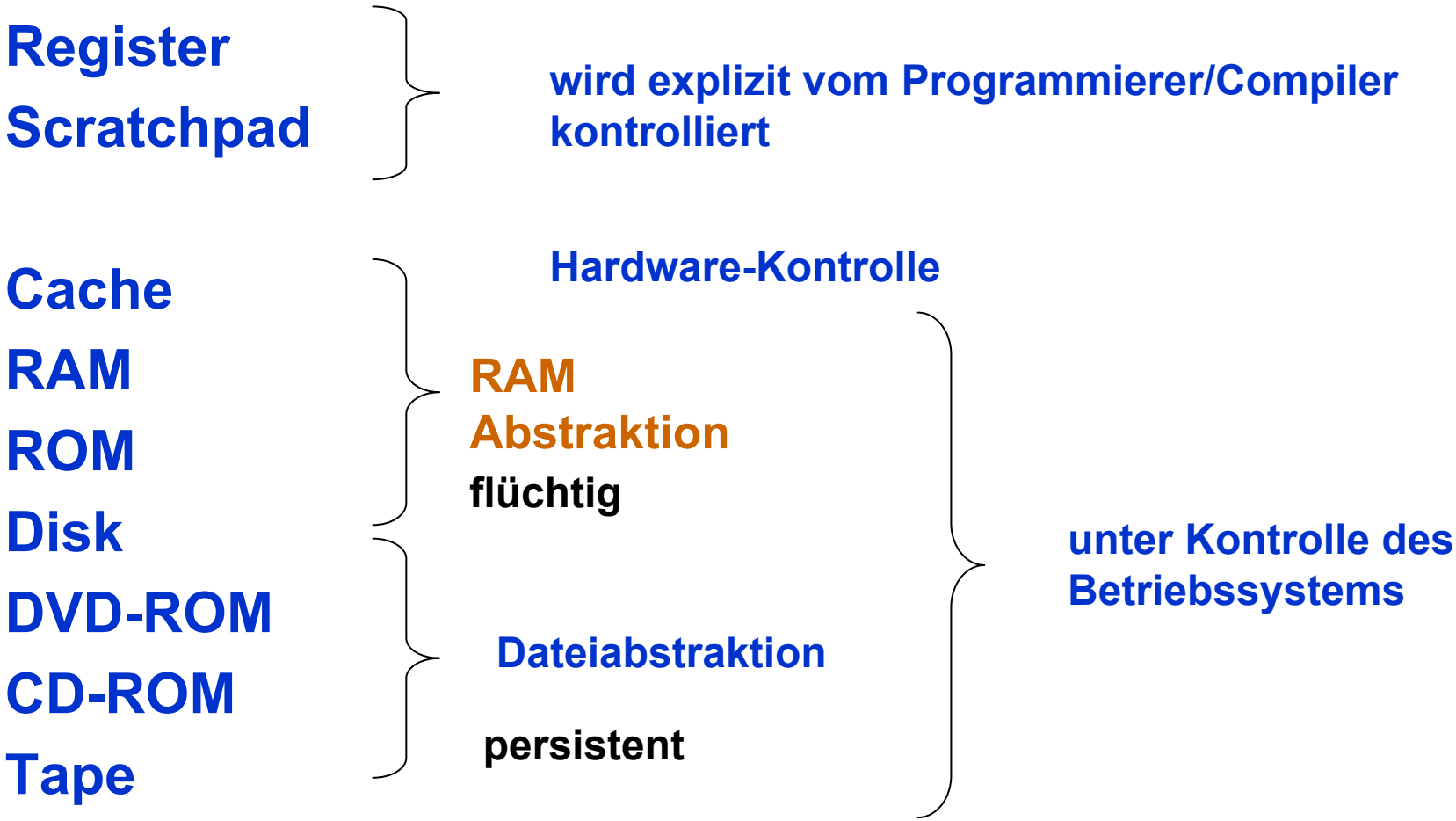
Die Speicherverwaltung hat die Approximierung dieser Eigenschaften zum Ziel !

Speicherverwaltung

Die Kosten-Leistungs-Perspektive



Speicherverwaltung



Speicherverwaltung

Themen für die Speicherverwaltung:

- Relokation
- gemeinsame Nutzung
- Zugriffsschutz

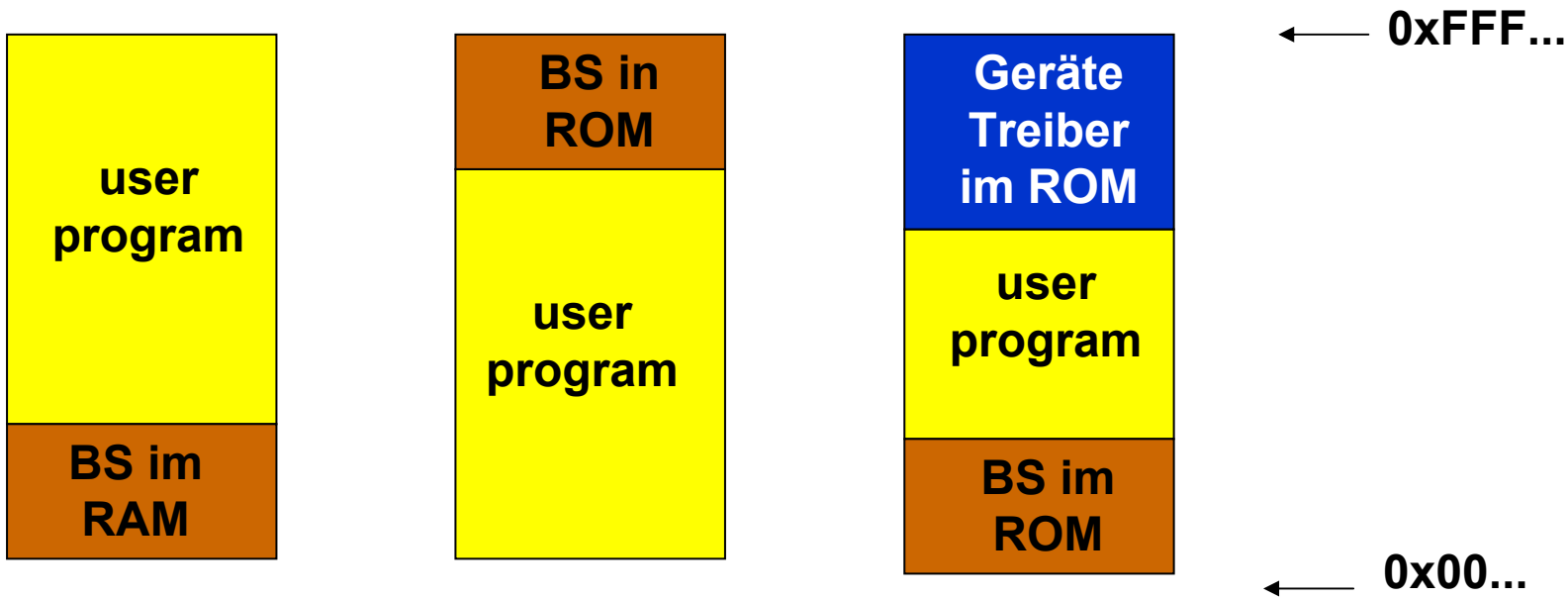
} resultiert aus dem Zugriff durch mehrere Prozesse/Programme

- Transparenter Zugriff über eine Hierarchie von physischen Speichern hinweg
  - logische Organisation
  - physische Organisation

Multiprogramming mit Speicherpartitionen

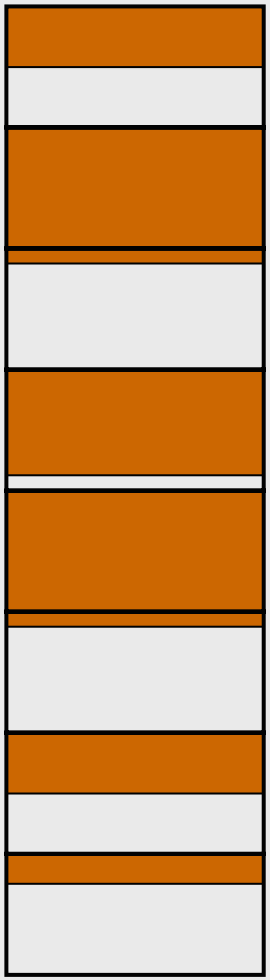
Statische Partitionierung:

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

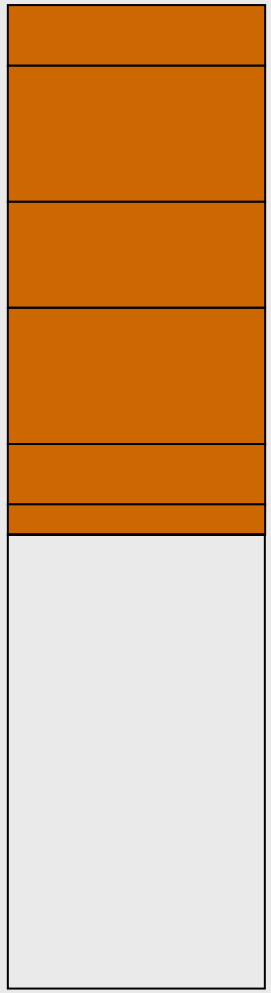


feste Partitionen

Problem:

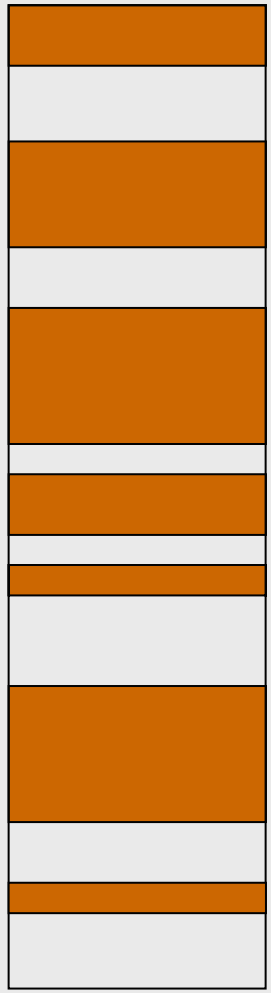
interne Fragmentierung

var. Partitionen Laufzeit Allok.



Anfangs. Situation

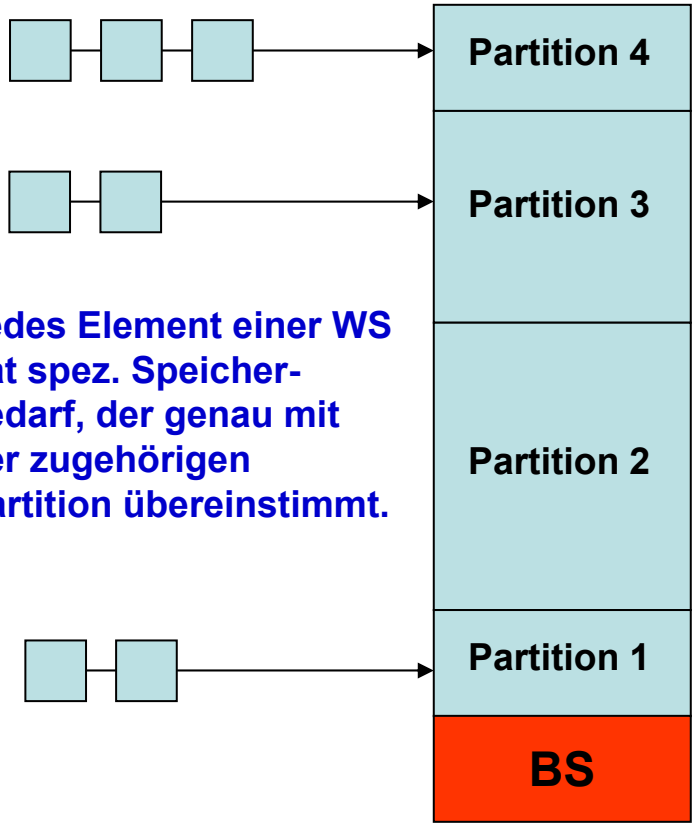
nach einiger Zeit



Problem: externe Fragmentierung benötigt komplizierte Verwaltung. & Kompaktifizierung

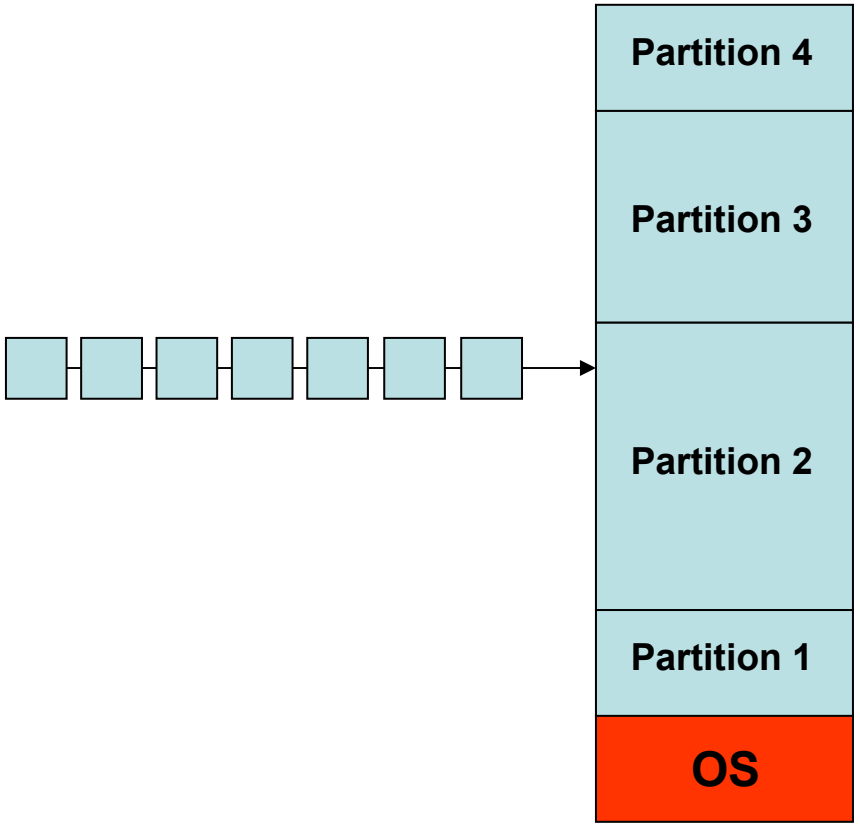
Verwaltung fester Partitionen

mehrere Warteschlangen



Jedes Element einer WS hat spez. Speicher-bedarf, der genau mit der zugehörigen Partition übereinstimmt.

eine Warteschlange



Speicherverwaltung mit Partitionen

Relokation und Zugriffsschutz

Relokation:

Probleme: Programme müssen in beliebigen Speicherbereichen funktionieren.

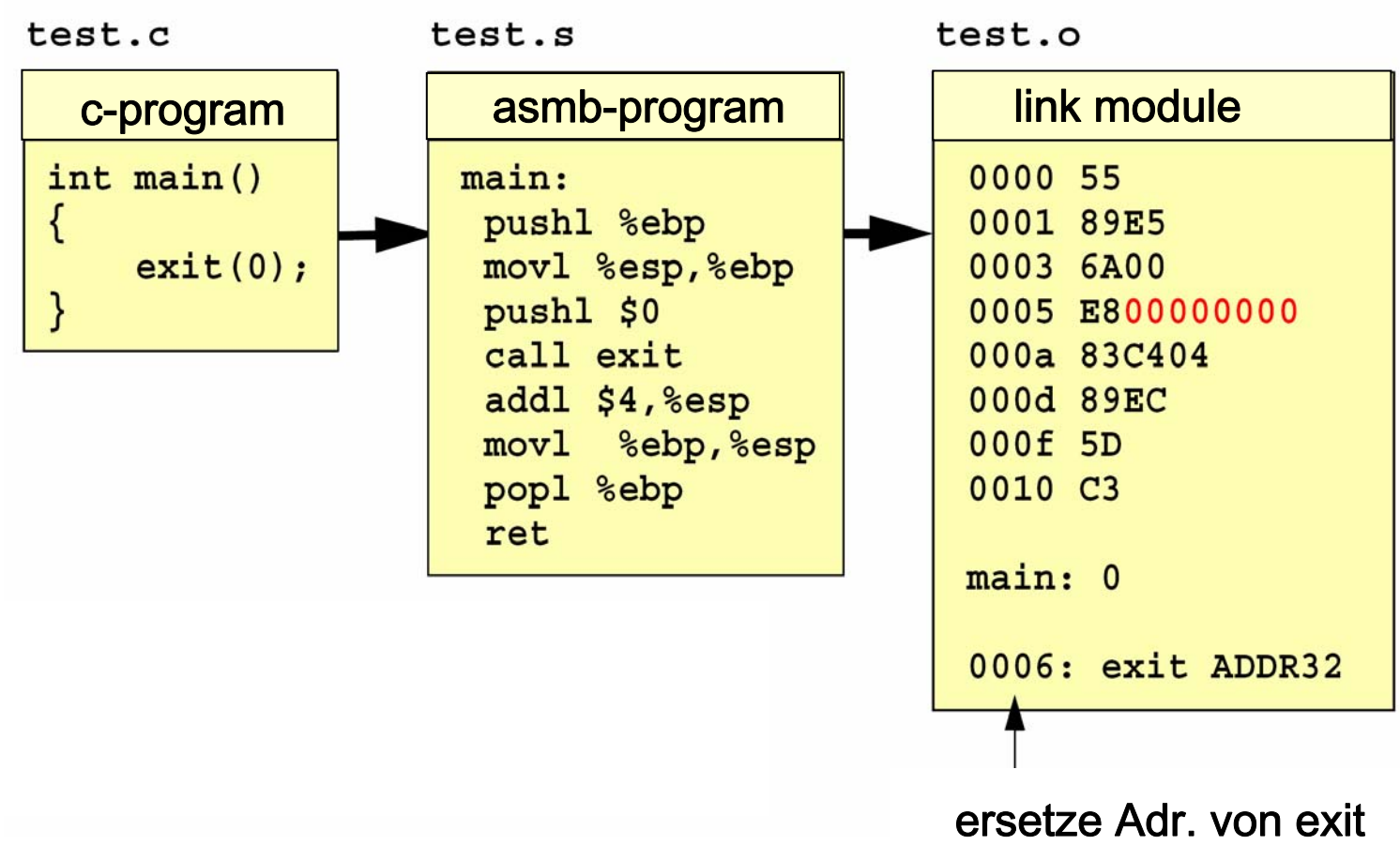
Mechanismen:
1. Statisches Binden von Speicheradressen zur Compilierungszeit 
2. Relokation beim Einlagern in den Hauptspeicher → loader/linker
3. Relokation zur Laufzeit → benötigt positionsunabhängigen Code
   → setzt Unterstützung durch die Rechnerarchitektur voraus.

Zugriffsschutz:

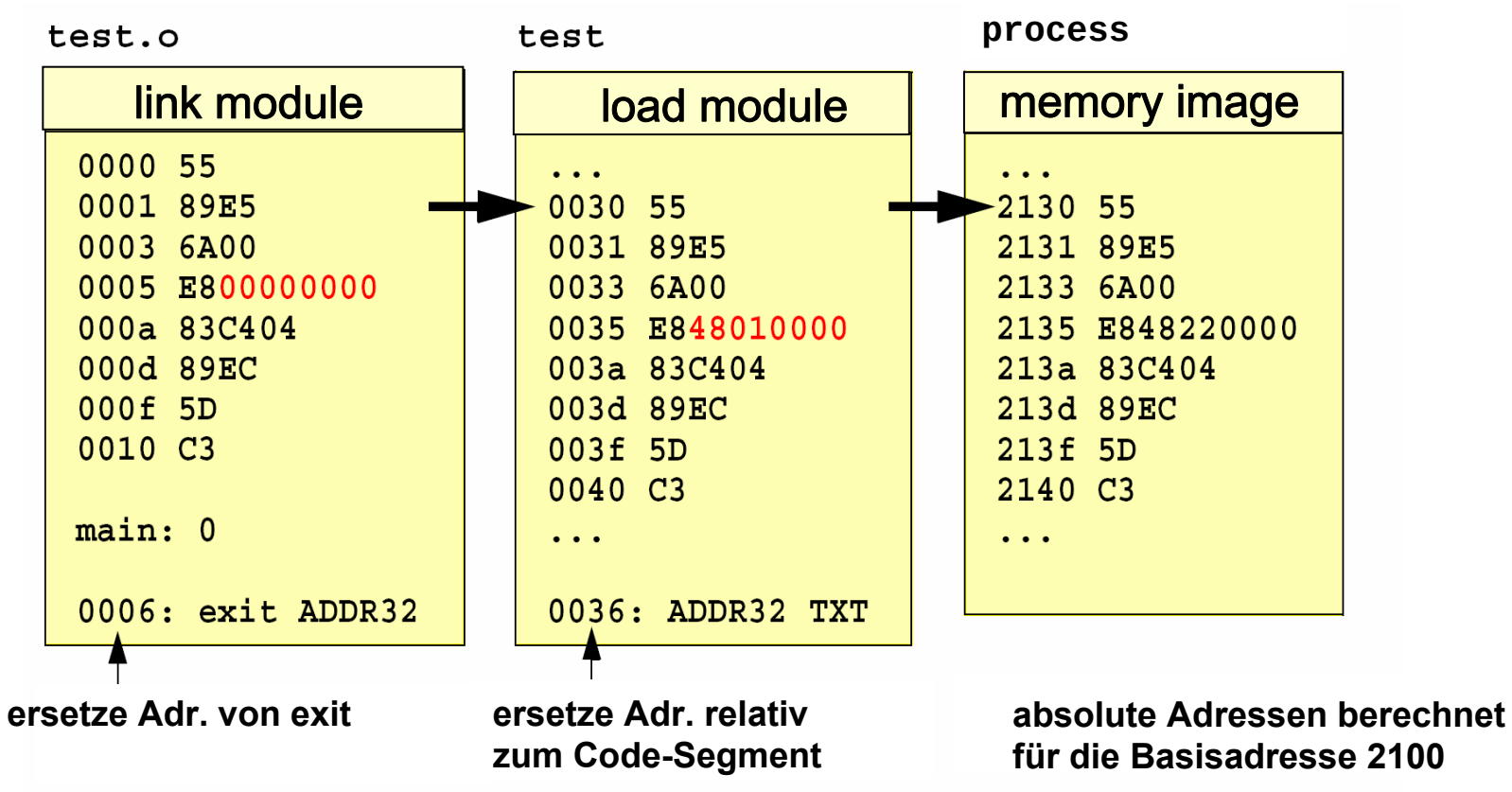
Probleme: Beliebige Referenzen auf Bereiche ausserhalb der Partition.

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

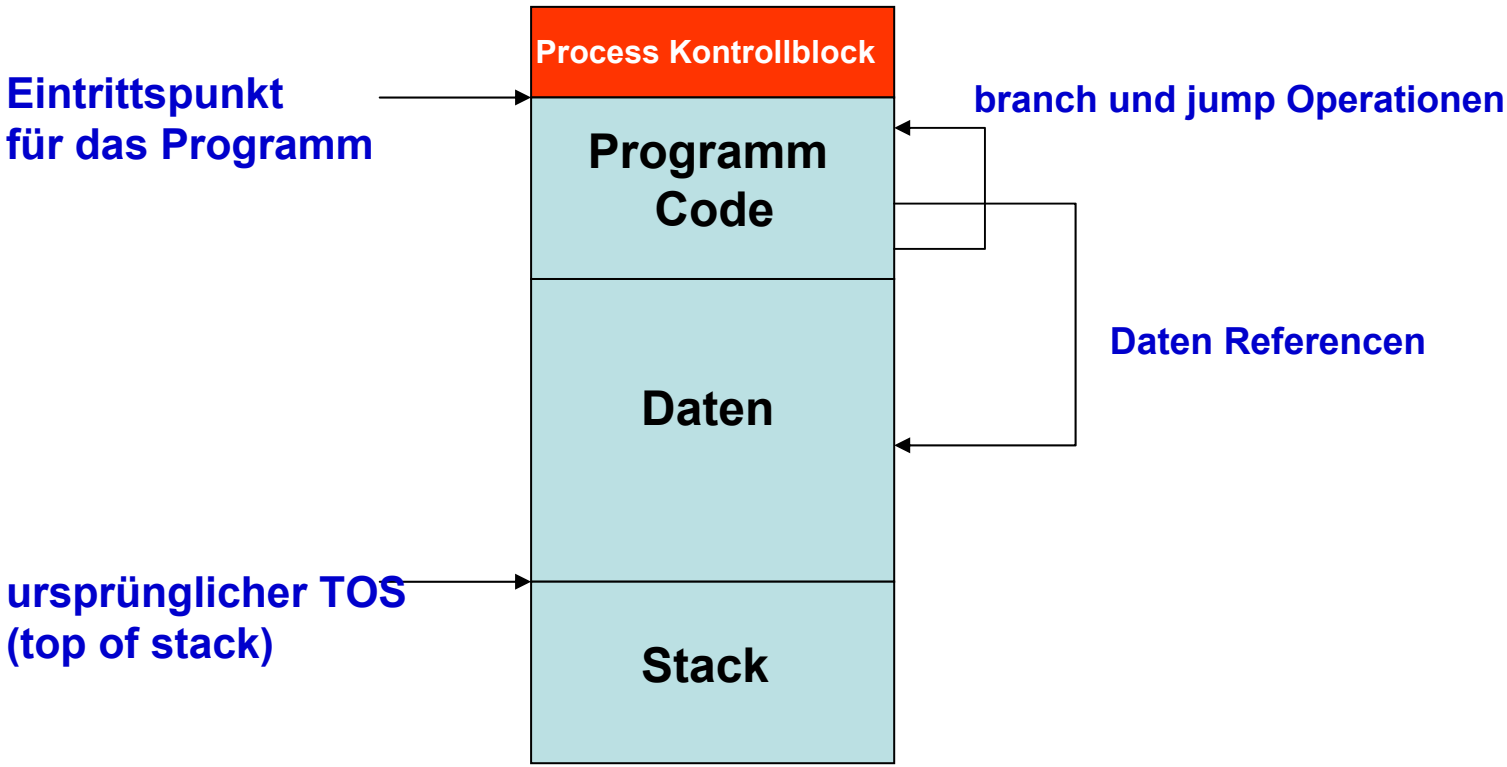

test.c
c-program
int main()
{
exit(0);
}
test.s
asmb-program
main:
pushl %ebp
movl %esp,%ebp
pushl $0
call exit
addl $4,%esp
movl %ebp,%esp
popl %ebp
ret
test.o
link module
0000 55
0001 89E5
0003 6A00
0005 E800000000
000a 83C404
000d 89EC
000f 5D
0010 C3
main: 0
0006: exit ADDR32
ersetze Adr. von exit

Relokation

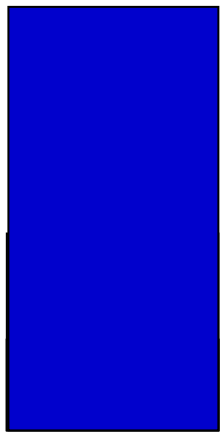

test.o
link module
0000 55
0001 89E5
0003 6A00
0005 E800000000
000a 83C404
000d 89EC
000f 5D
0010 C3
main: 0
0006: exit ADDR32
test
load module
...
0030 55
0031 89E5
0033 6A00
0035 E848010000
003a 83C404
003d 89EC
003f 5D
0040 C3
...
0036: ADDR32 TXT
process
memory image
...
2130 55
2131 89E5
2133 6A00
2135 E848220000
213a 83C404
213d 89EC
213f 5D
2140 C3
...
ersetze Adr. von exit
ersetze Adr. relativ zum Code-Segment
absolute Adressen berechnet für die Basisadresse 2100

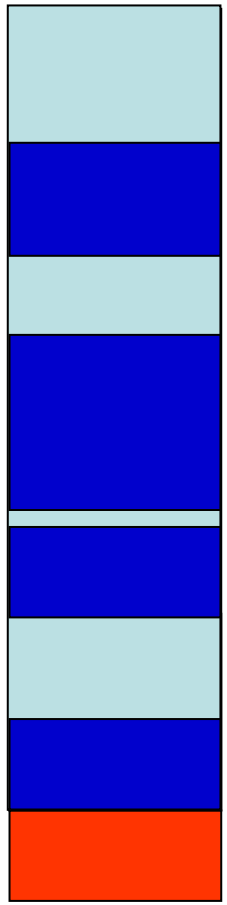
Dynamische Partitionen and Ein-/Auslagern (Swapping)

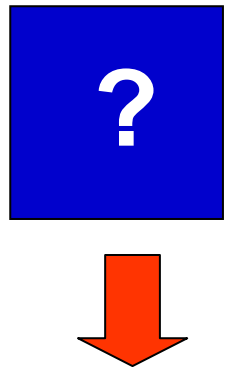
Ein-/Auslagern: Jeder Prozess wird vollständig vom Speicher auf die Platte geschoben und umgekehrt, mit seinen Code-, Daten- and Stackbereichen.



Dynamische Partitionen and Ein-/Auslagern (Swapping)



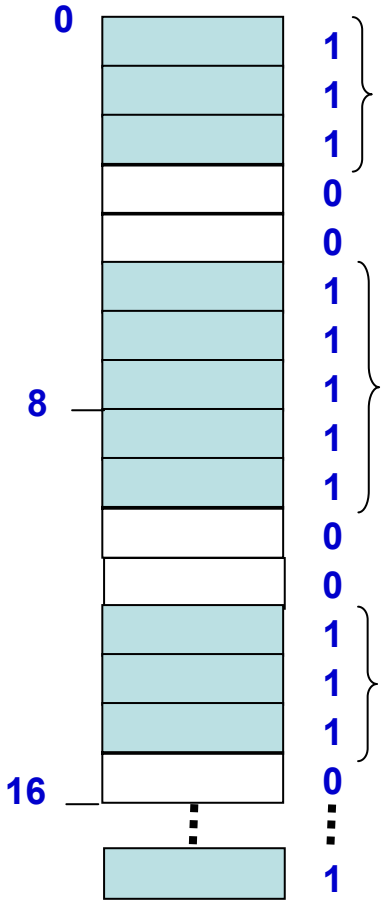




- Freispeicherverwaltung
- Ersetzungsstrategie

Dynamische Partitionen and Ein-/Auslagern (Swapping)

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

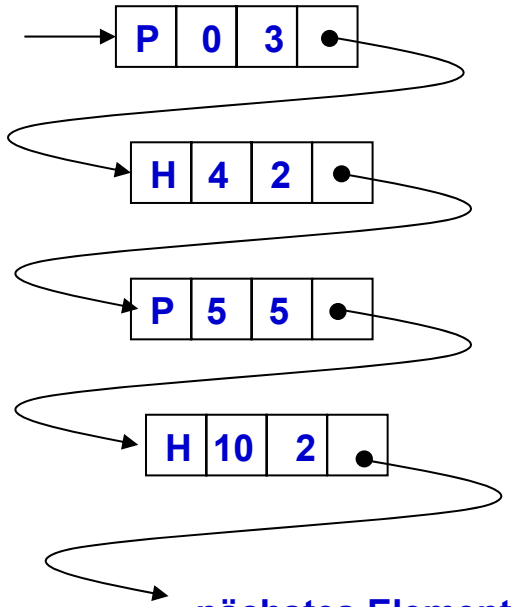


Bitmap-darstellung:

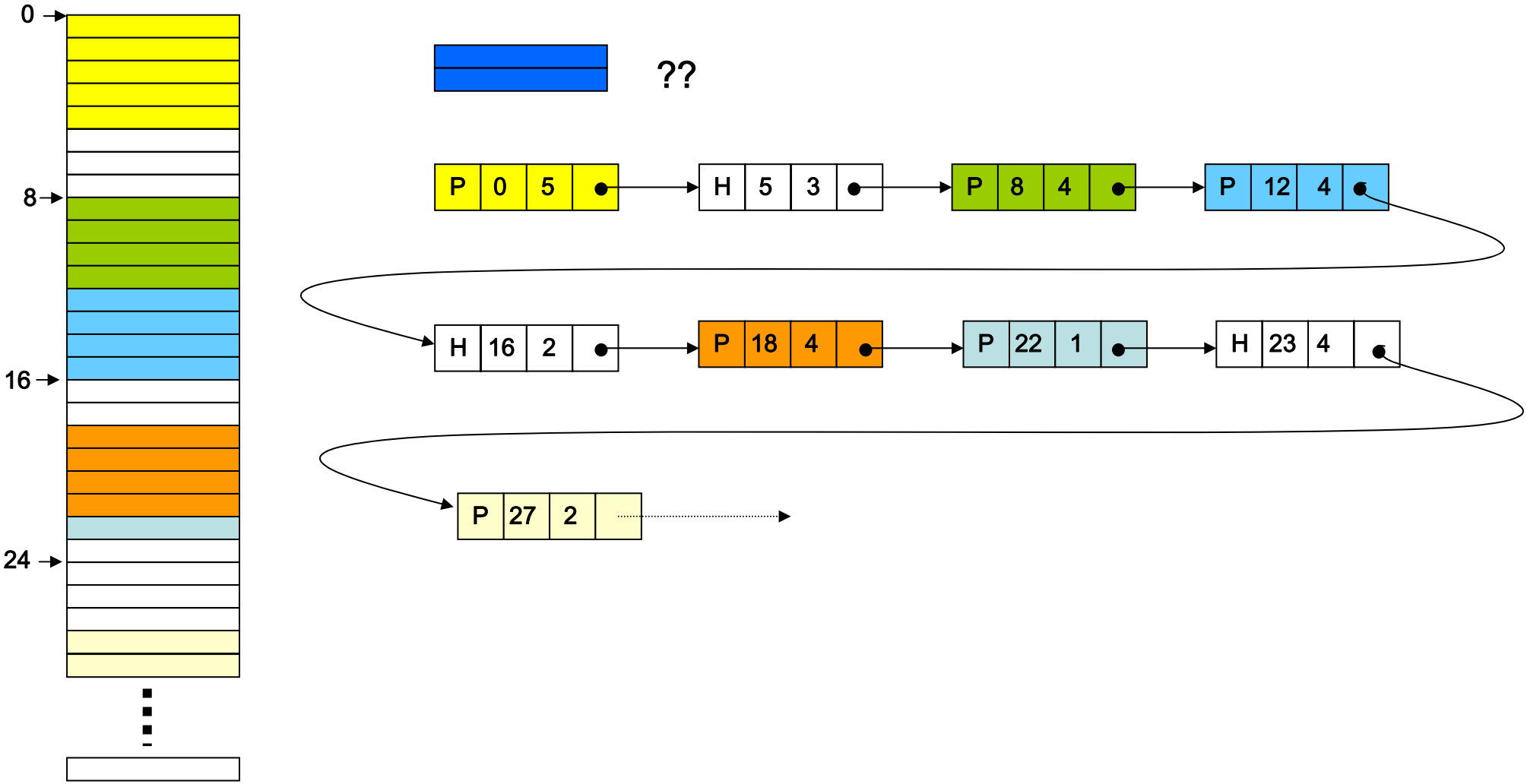
1110011
1110011
10.....

1 = allozierter Block
0 = Loch

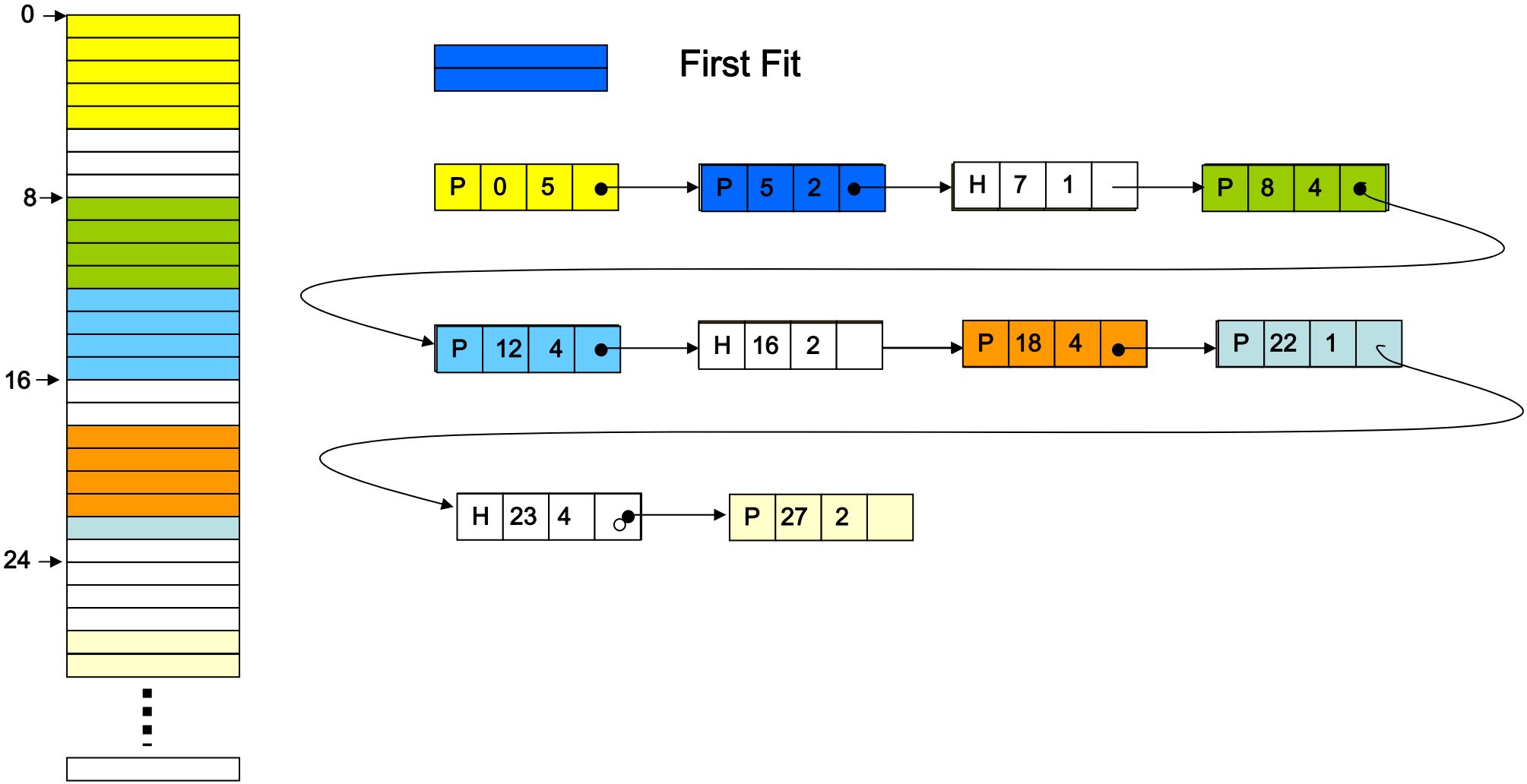
Darstellung durch verkettete Liste:



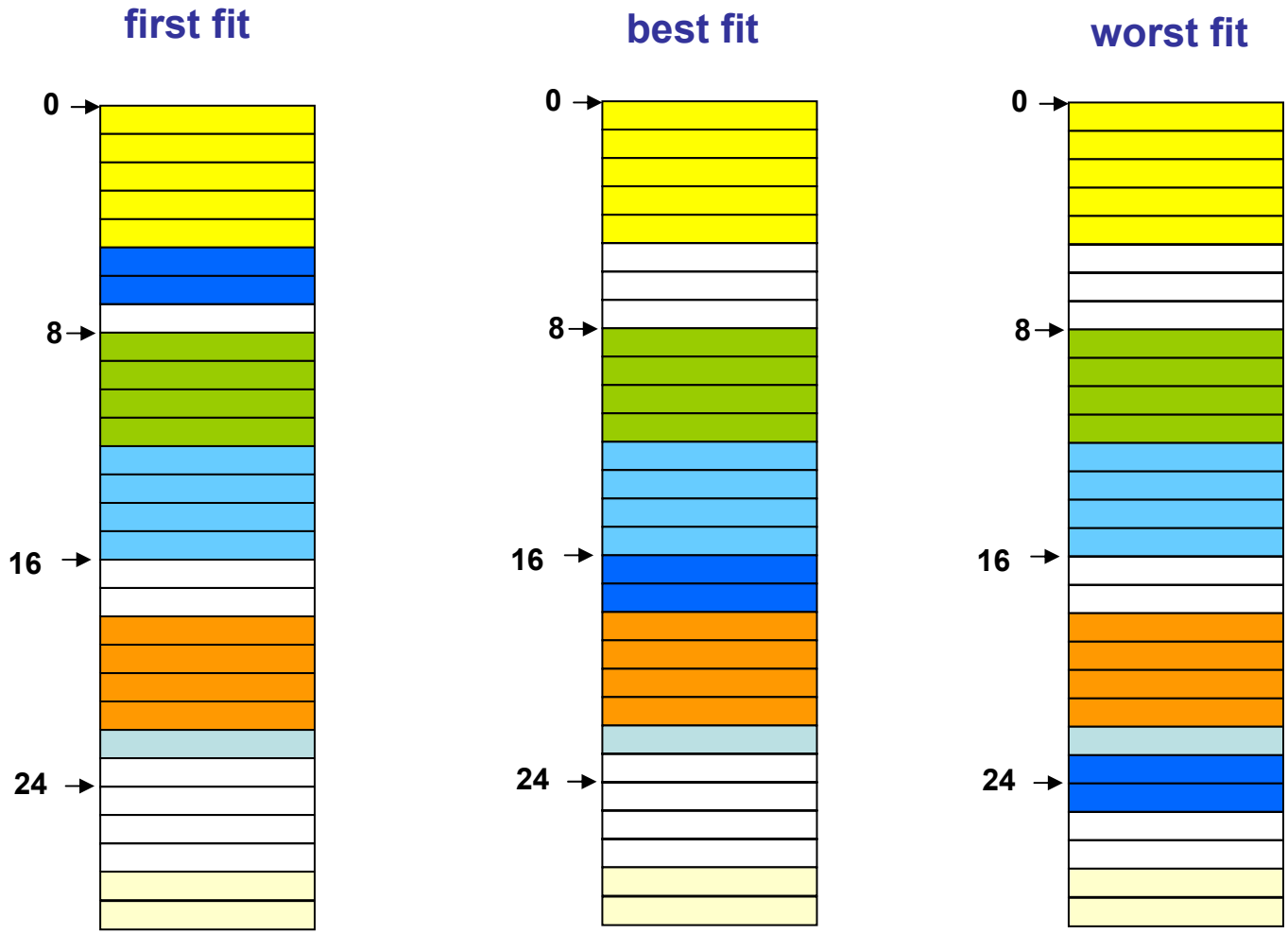
Dynamische Partitionen and Ein-/Auslagern (Swapping)



Dynamische Partitionen and Ein-/Auslagern (Swapping)



Dynamische Partitionen and Ein-/Auslagern (Swapping)



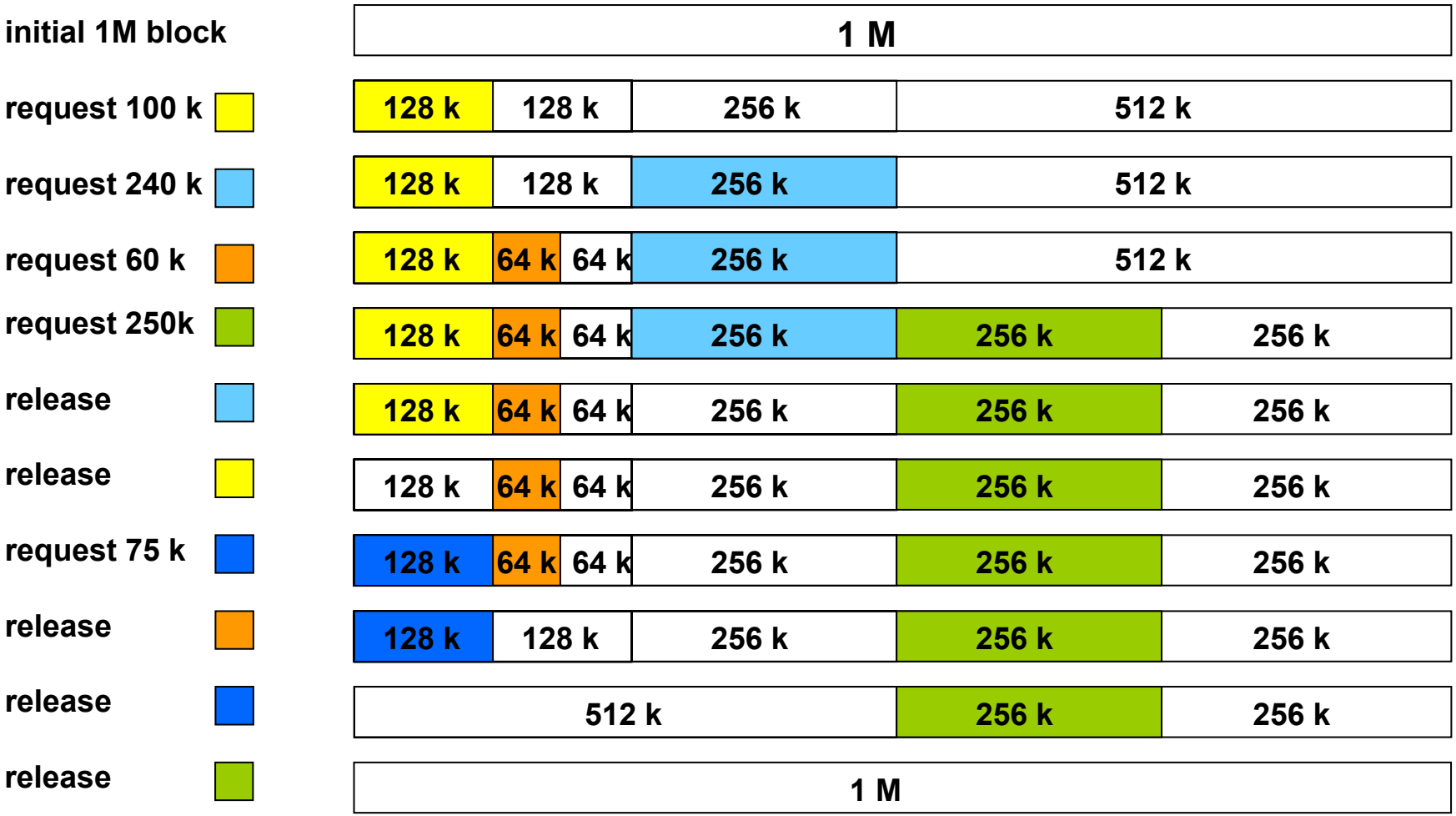
next fit: Sartpunkt ist die Position des zuletzt gefundenen Blocks

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

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**

Das Buddy-System



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 können größer sein als der Realspeicher.
- Zugriffsschutz, wenn mehrere Prozesse in einem Adreßraum ablaufen.

Diskussion:

Adreßraum eines Prozessors:

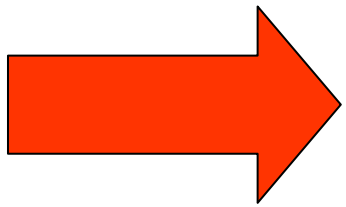
32 Bit = 4.294.967.296

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-Aulagern 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

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

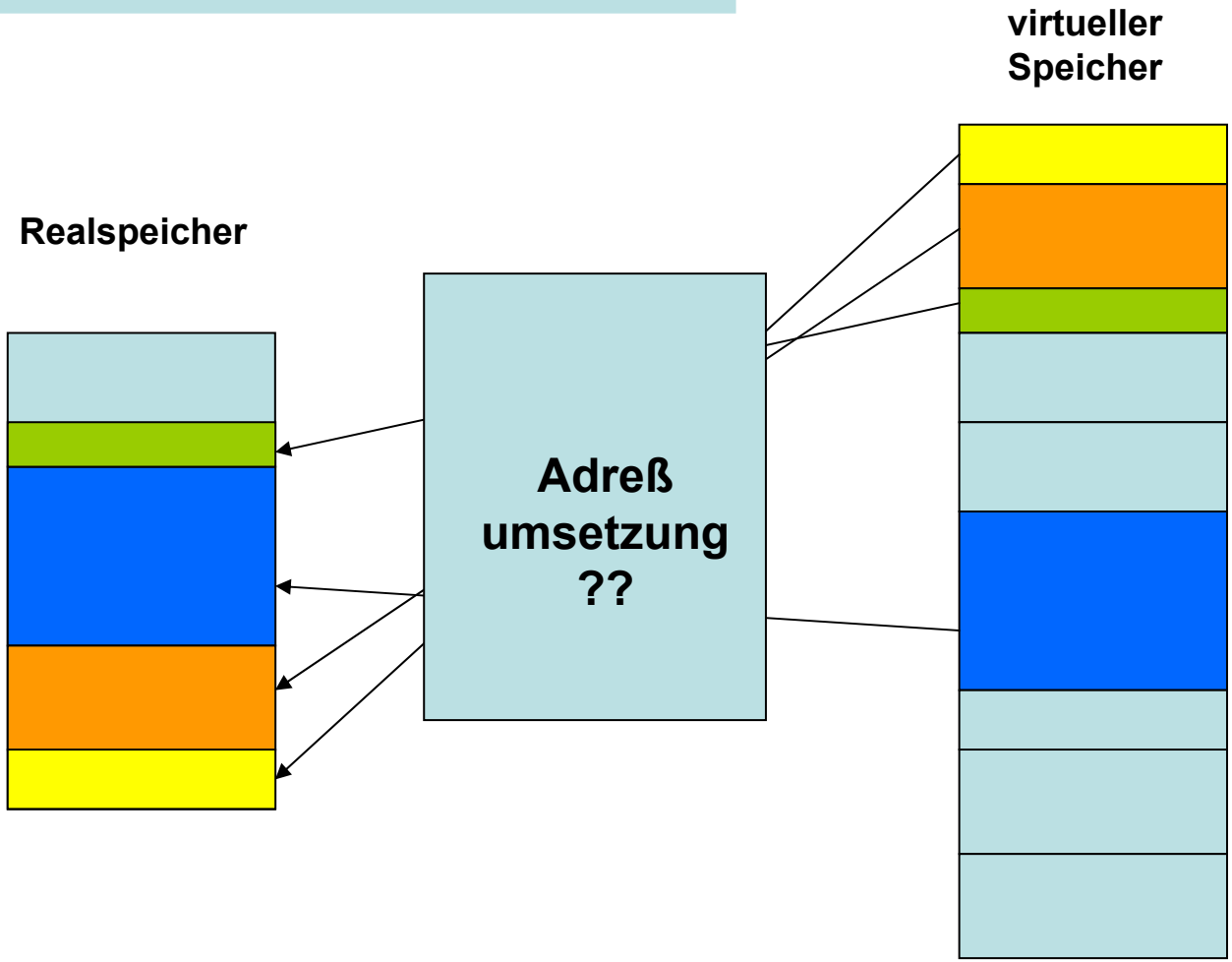
By T. Kilburn, D. J. Howarth, R. B. Payne and F. H. Sumner

Introduction

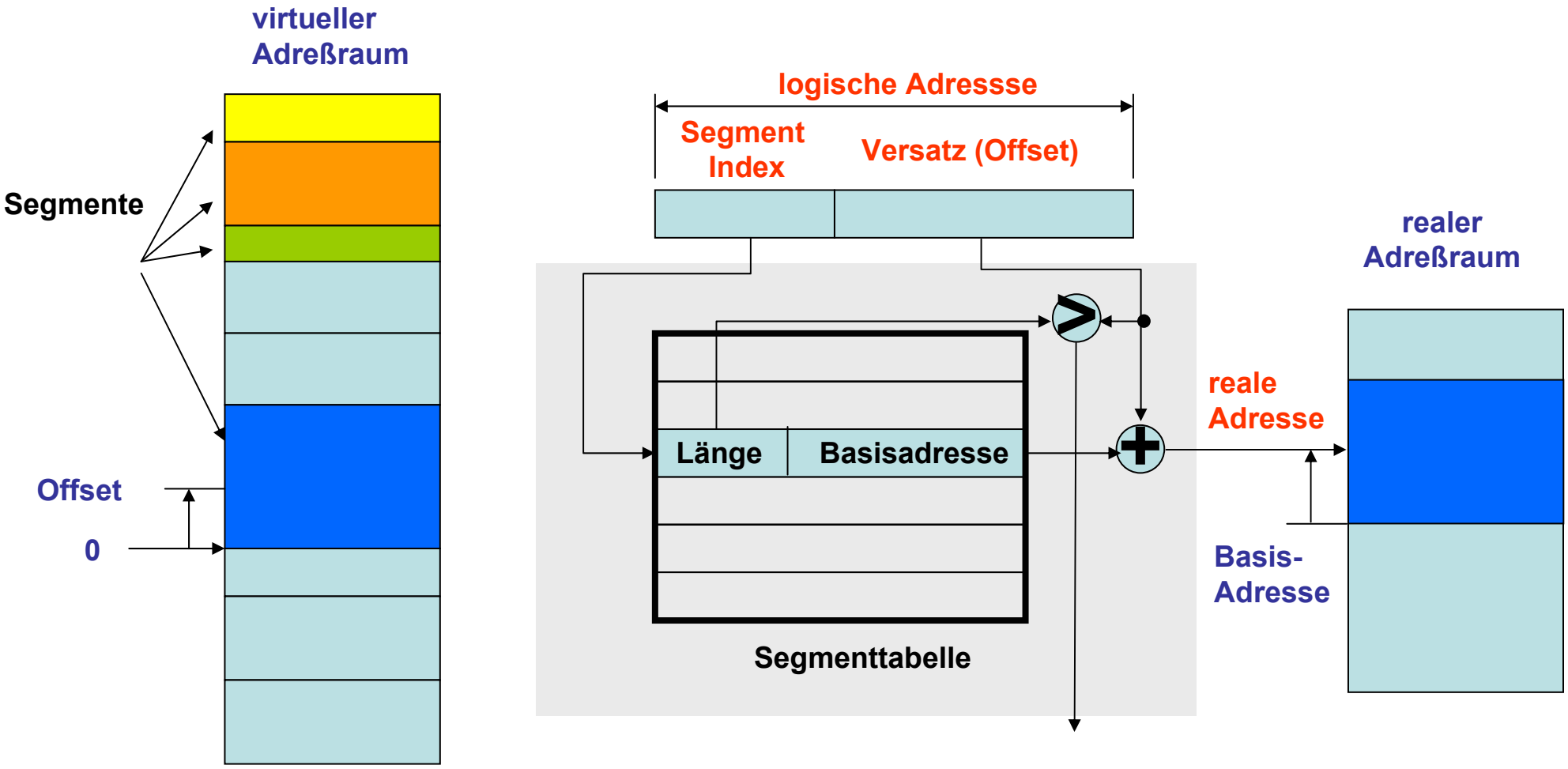
Atlas* is the name given to a comprehensive computer system designed by a joint team of Ferranti Ltd. and Manchester University engineers. The computer system comprises the central computer, fixed store, core store, magnetic drum store, magnetic tapes, and a large quantity and variety of peripheral equipments 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

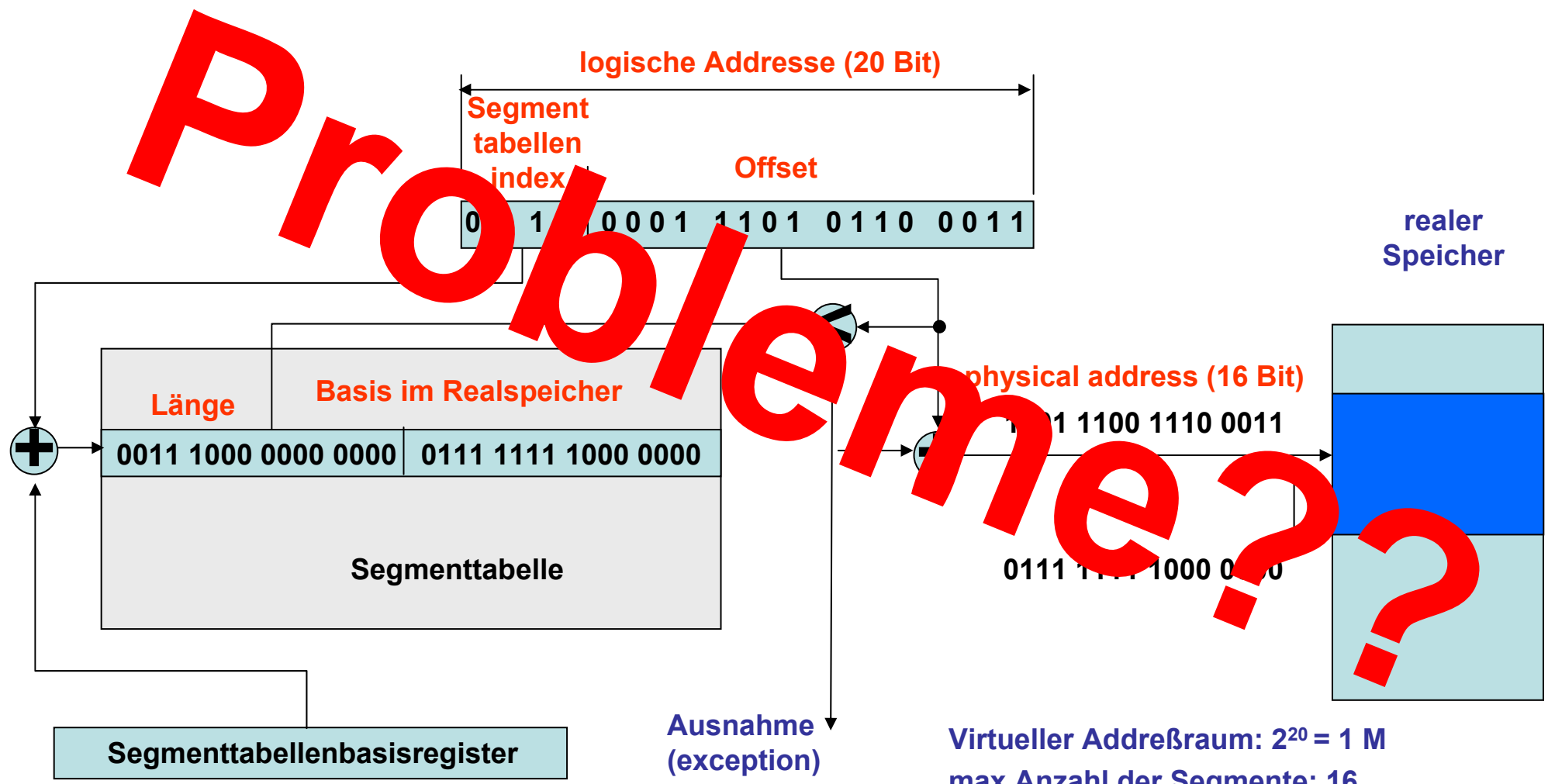
virtual memory



"segmentierter" virtueller Speicher



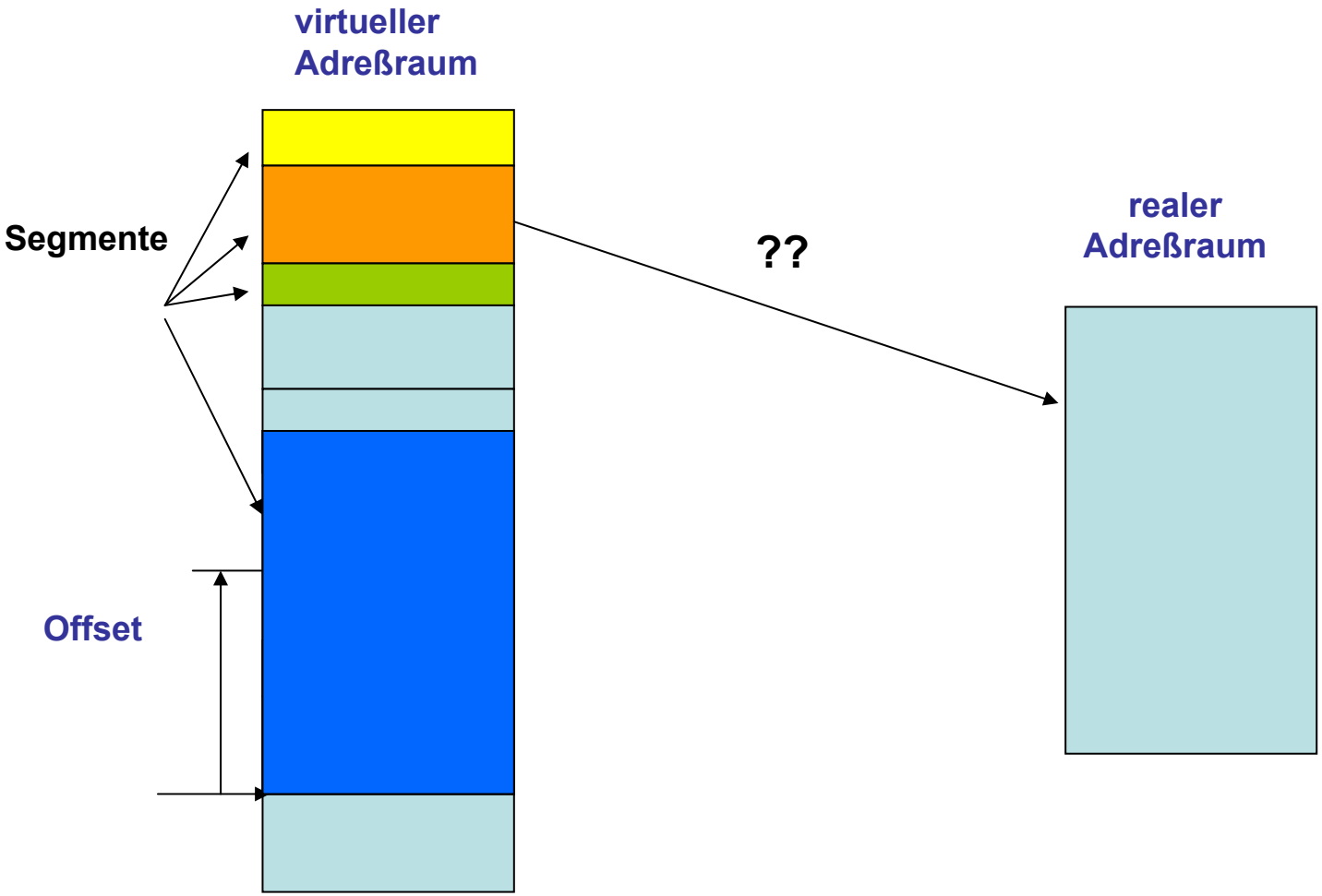
Segmentierter Speicher - Adreßumsetzung



Probleme??

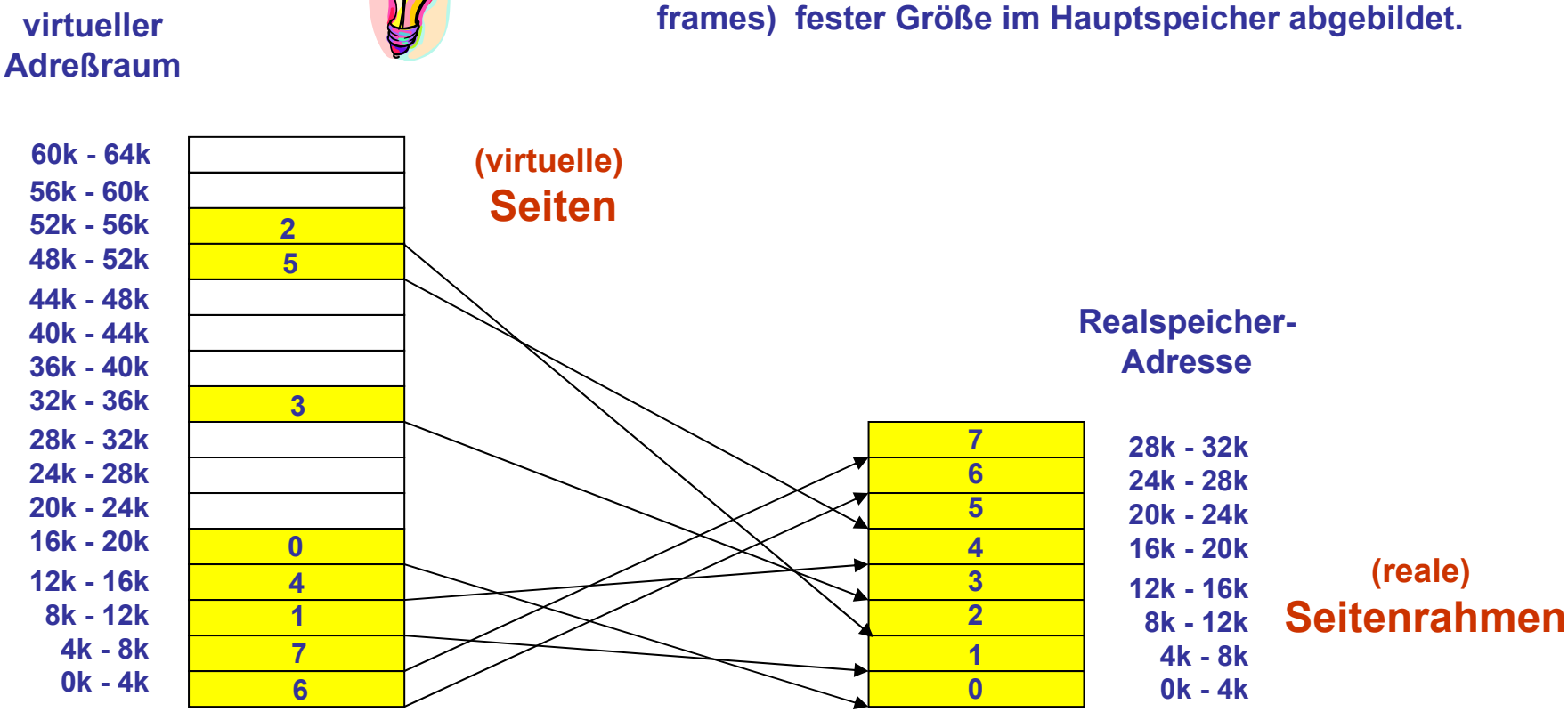
Virtueller Addreßraum: 2^{20} = 1 M
max Anzahl der Segmente: 16
max Größe eines Segments: 64 k

"segmentierter" virtueller Speicher

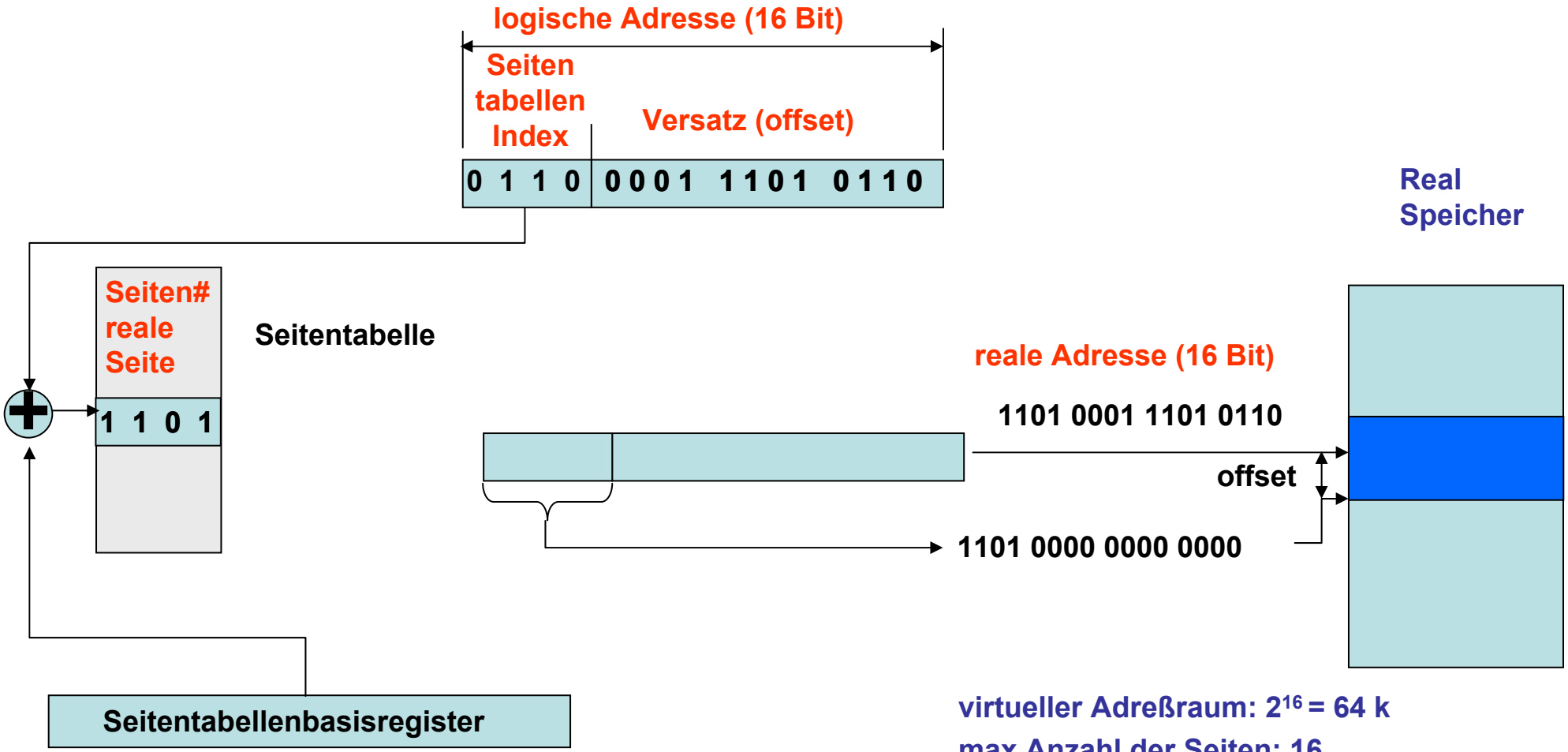


Seitenorientierter virtueller Speicher

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



Adreßumsetzung mit Seiten

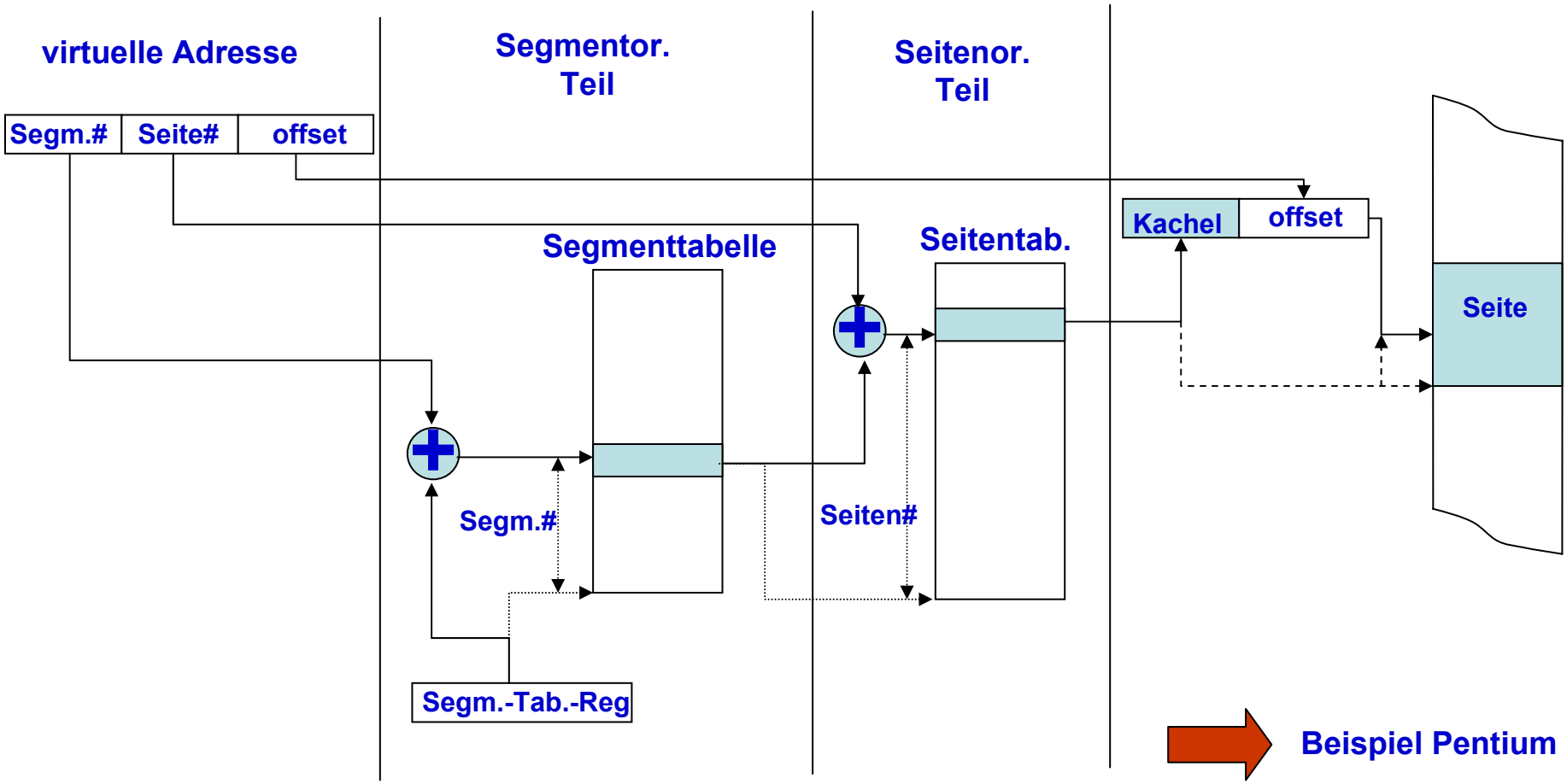


virtueller Adreßraum: 2^{16} = 64 k
max Anzahl der Seiten: 16
max Größe der Seiten: 4 k

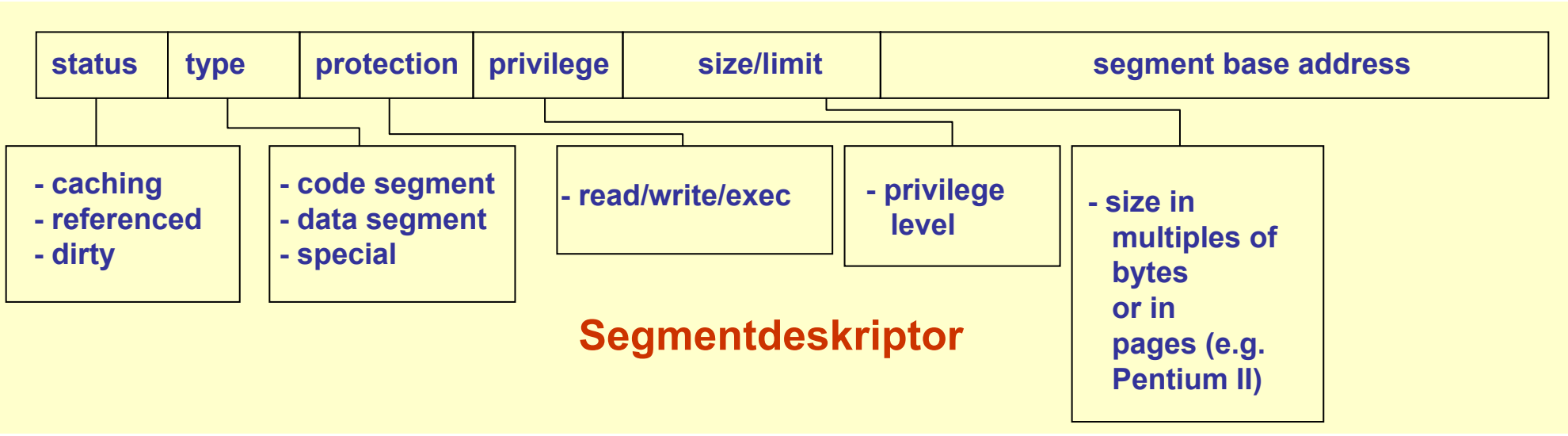
Diskussion: Segmente gegen Seiten

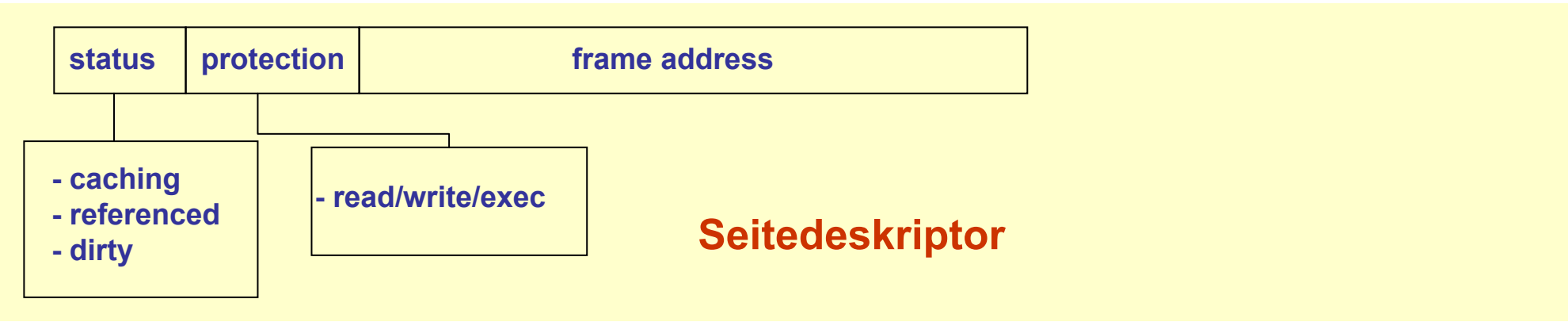
	Transparenz für Progr.	Anzahl d. Adreß-räume	virt. Adr. raum > realer Speicher	variable Objektgr.	Frag-ment.	Verw. overhead	Hauptgrund für Einführung
Seitenor.		ein			intern		unendlich viel Speicher
Segmentor.		viele			extern		mehrere Adr. räume

Kombination von Segmenten und Seiten



Struktur eines Tabelleneintrags





Größe der Seitentabellen

32-Bit Adreßraum: 4G Addresses — Seitetabellengröße @ 4k: 1M @ 4k entries

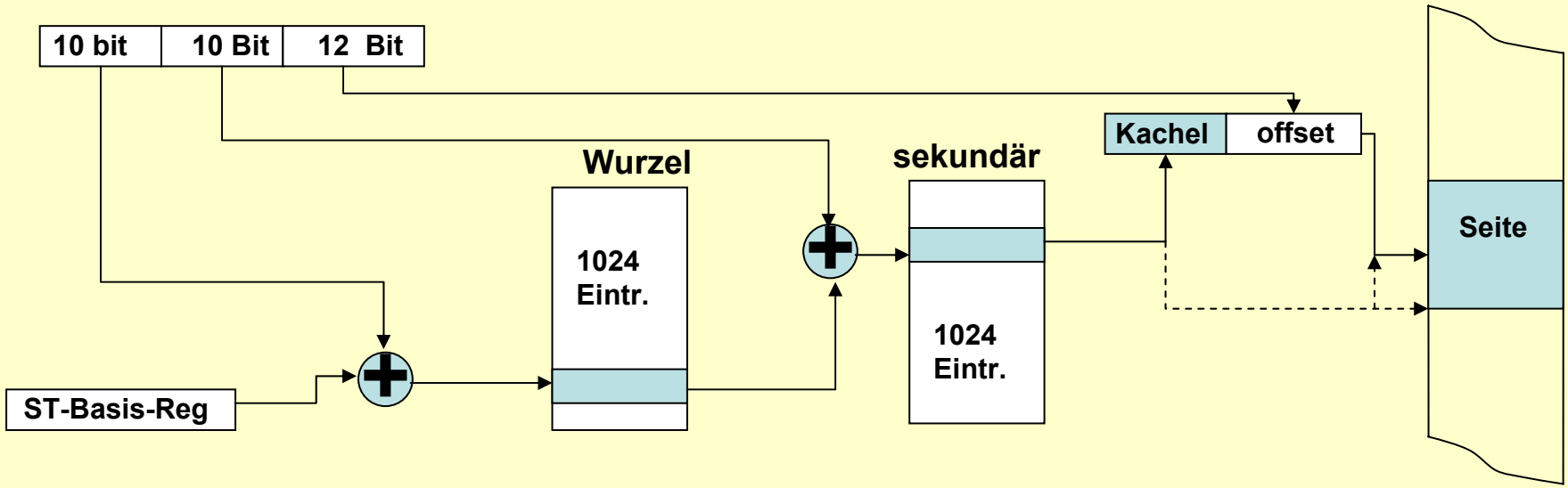
64-Bit Adreßraum: 4G·4G Addresses — Seitetabellengröße @ 4k: 4G·1M entries

1. Seitengröße erhöhen: e.g. UltraSPARC II supports 8k, 64k, 512k and 4M pages
   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. Umsetzen von realen auf virtuelle Adressen: Invertierte Seitentabellen
   Grund: Der reale Adreßraum ist klein im Vergleich zum virtuellen.
   Problem: viel mehr Seiten als Seitenrahmen → keine eindeutige Abbildung

mehstufige hierarchische Seitentabellen



Seitengröße

für n Segmente mit p bytes ist die

interne Fragmentierung: n·p/2

Zielkonflikte:

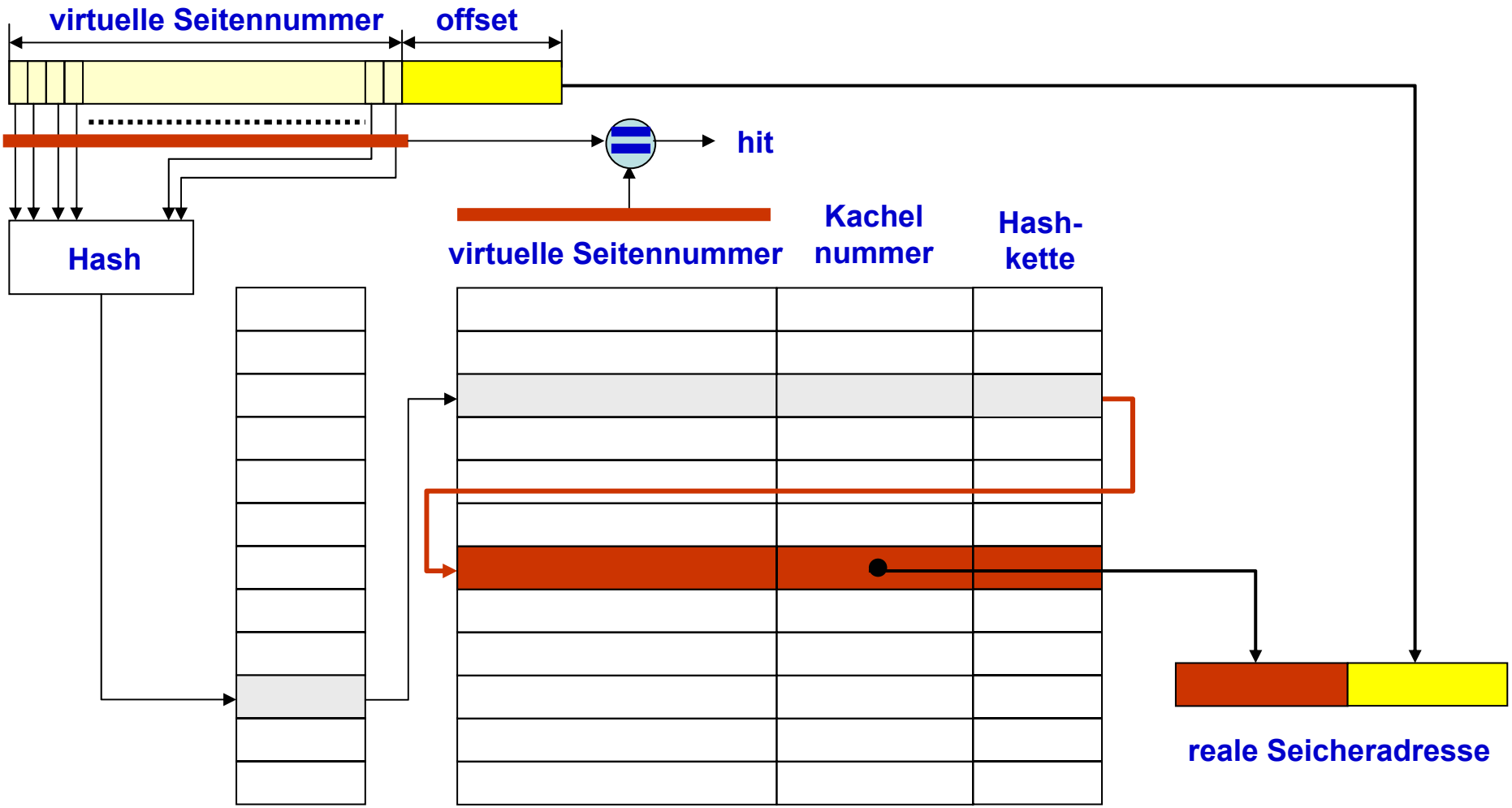
Seitengröße:	large	small
interne Fragmentierung (n·p/2):		
Seitentabellengröße+Management Ovh.		
Ladezeit von der Platte		

Übliche Seitengrößen liegen zwischen 512 Bytes und 64KB.

Heute werden Seitengrößen von 4KB oder 8KB benutzt.

invertierte Seitentabelle

PPC, AS/400



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.**

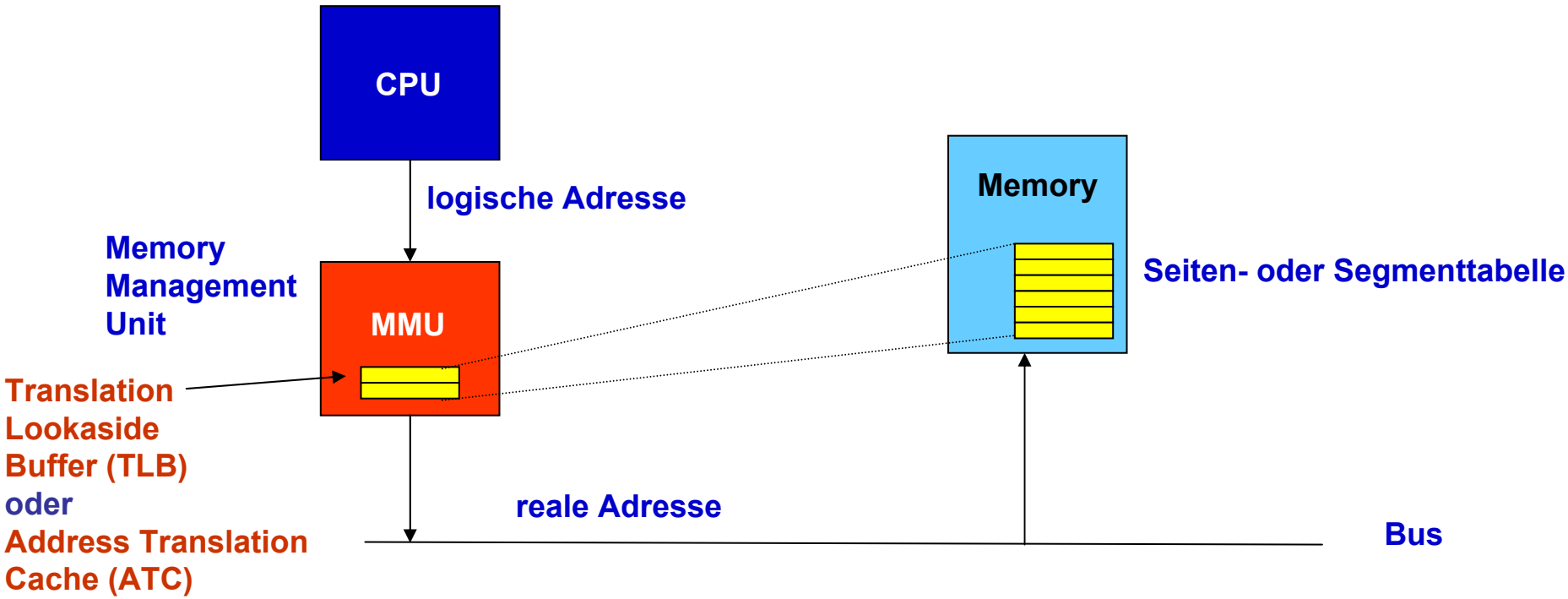
Hardwareunterstützung wird in allen Fällen benötigt !

Die MMU: Speicherverwaltungseinheit

Adreßübersetzung benötigt mehrere Ebenen der Indirektion

➡ **negative Auswirkung auf Leistung !**

➡ **benötigt Hardwareunterstützung zur Beschleunigung.**



Das Lokalitätsprinzip

Lokalitätsprinzip:

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

Das Lokalitätsprinzip ist die Basis für jeglichen Caching-Mechanismus!

Verwaltung des virtuellen Speichers

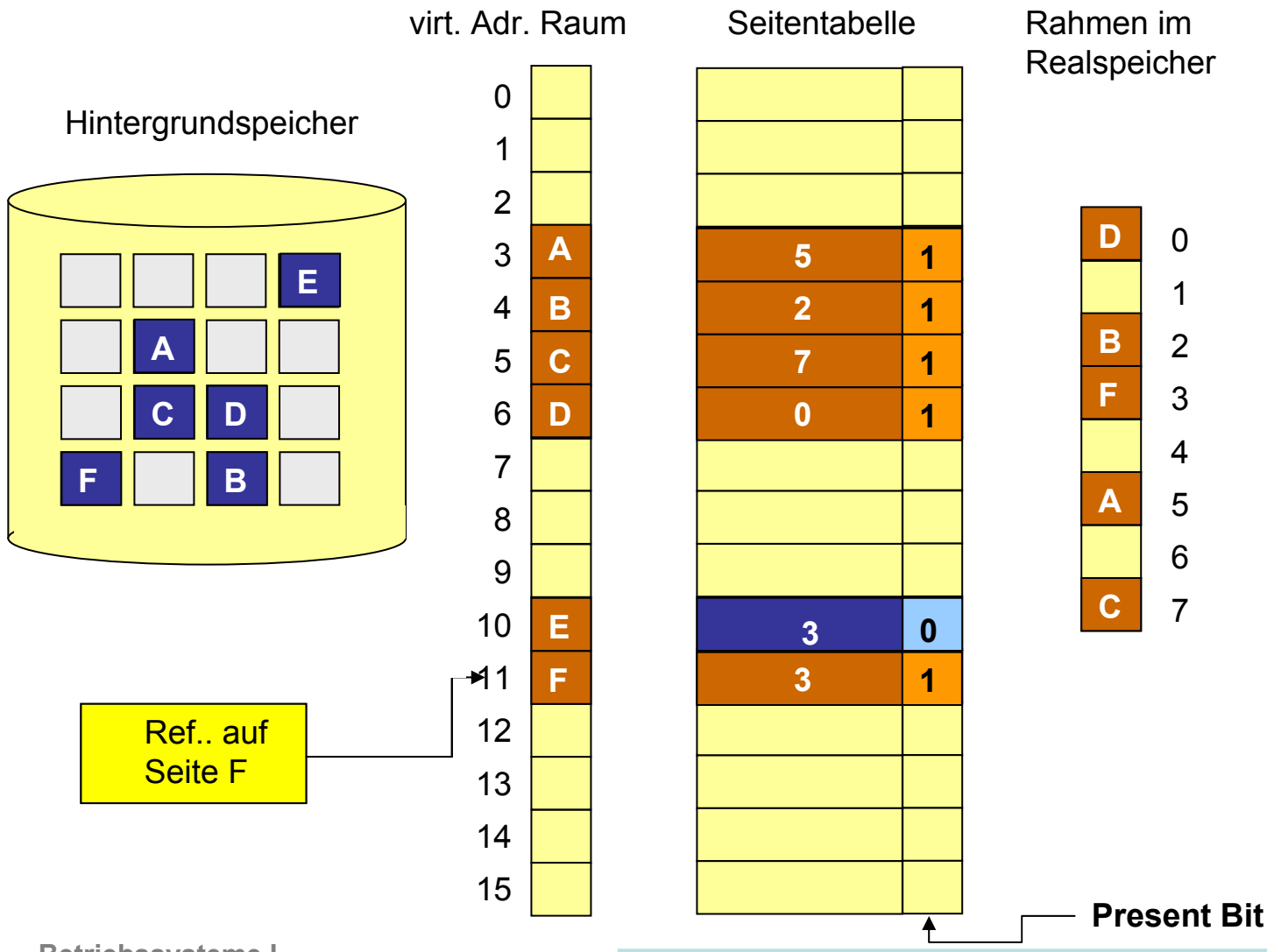
Woher weiss man, wie viele Seiten benötigt werden?

➡ **einlagern von Seiten auf Anforderung**

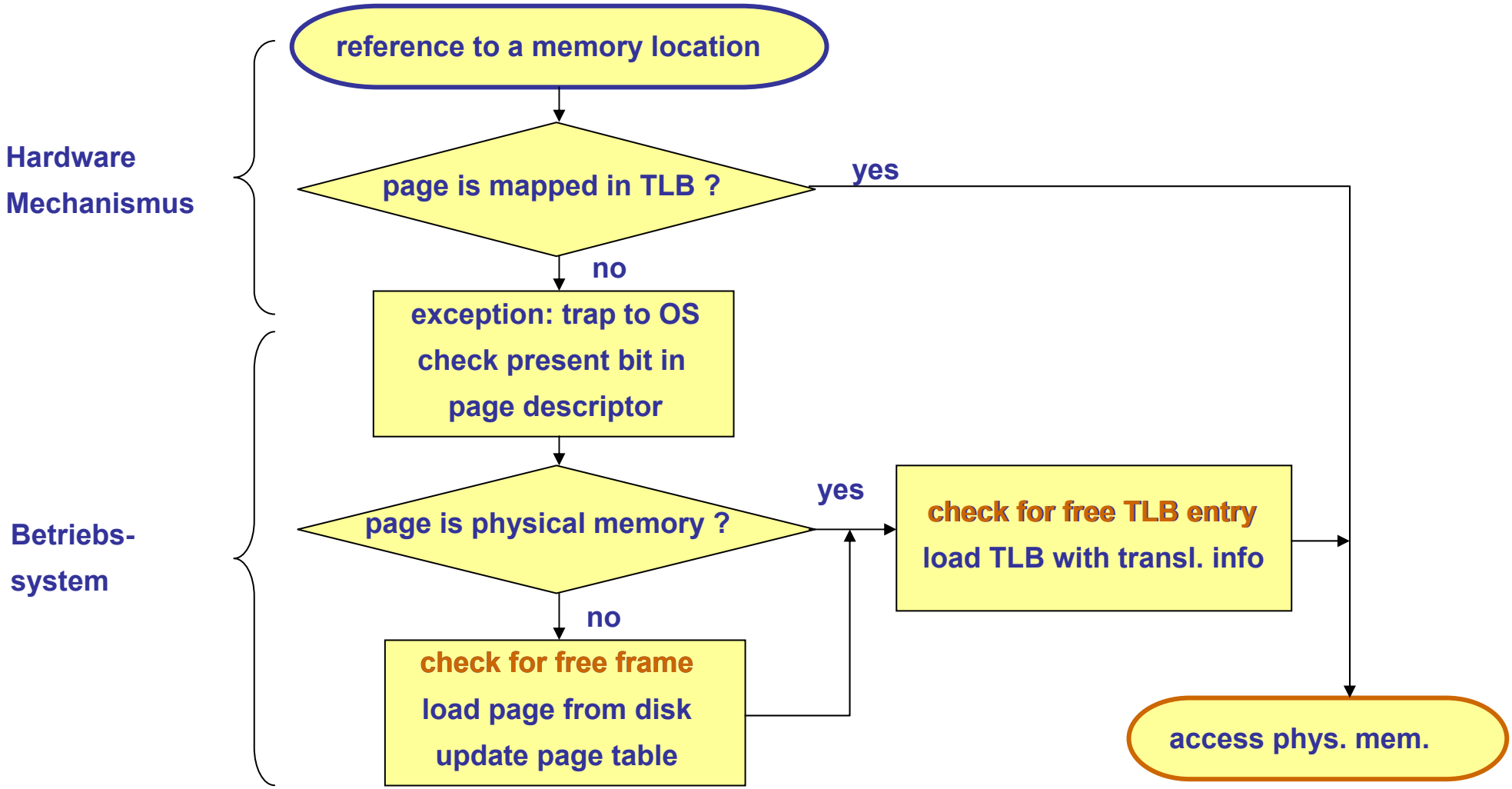
Was tun, wenn mehr Seiten benötigt werden als Seitenrahmen zur verfügung stehen?

➡ **Seitenersetzungsmechanismen**

Demand Paging



Demand Paging



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:

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

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!**

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	R	D	P	protect.	frame number

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**

Problem

NRU ersetzt eine beliebige Seite aus der niedrigsten nichtleeren Klasse

sweep sweep

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**	**1**	**1**	**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.

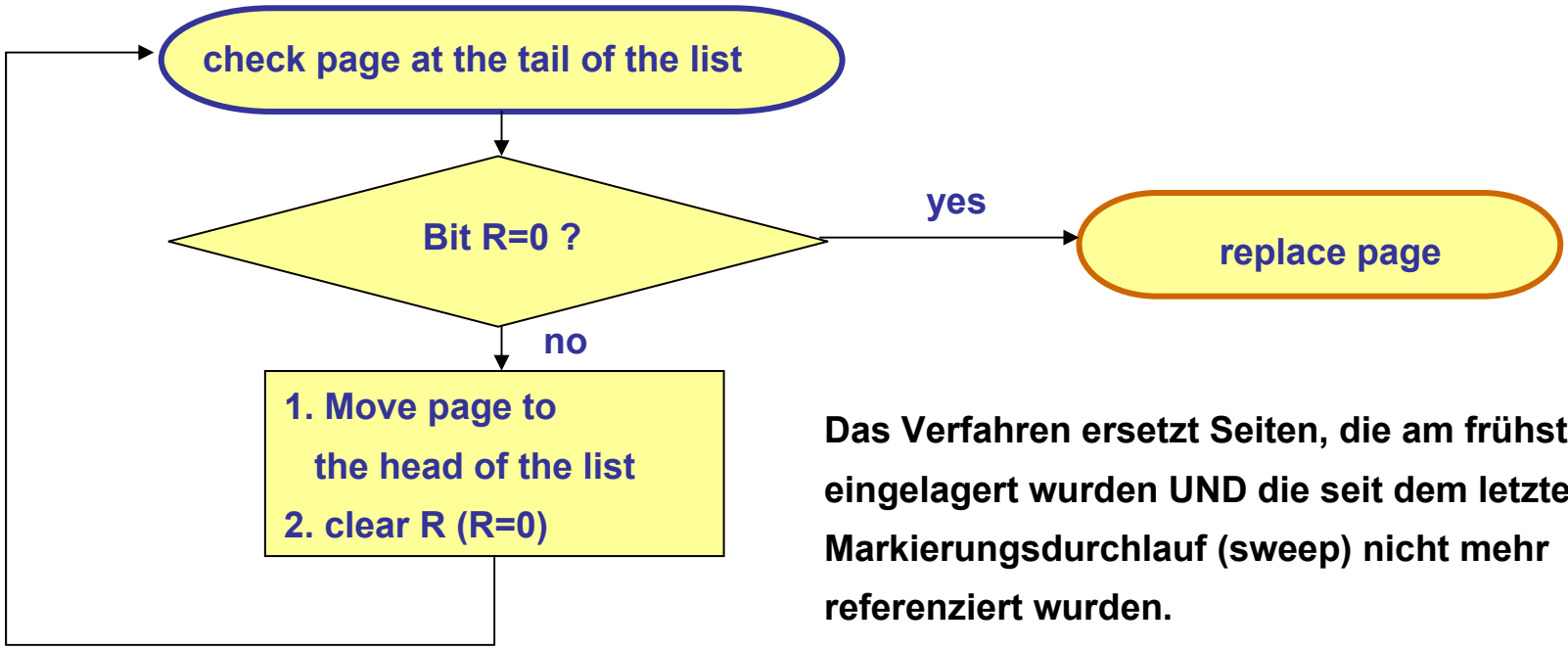
Ref. Folge		1	2	3	4	1	2	5	1	2	3	4	5
Kachelzuordnung im Realspeicher	frame 1	1	1	1	4	4	4	5	5	5	5	5	5
	frame 2		2	2	2	1	1	1	1	1	3	3	3
	frame 3			3	3	3	2	2	2	2	2	4	4
Kontrollstatus: Alter der Seite	frame 1	0	1	2	0	1	2	0	1	2	3	4	5
	frame 2	-	0	1	2	0	1	2	3	4	0	1	2
	frame 3	-	-	0	1	2	0	1	2	3	4	0	1
		P	P	P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

9 Seitenfehler

Seitenersetzungsstrategien

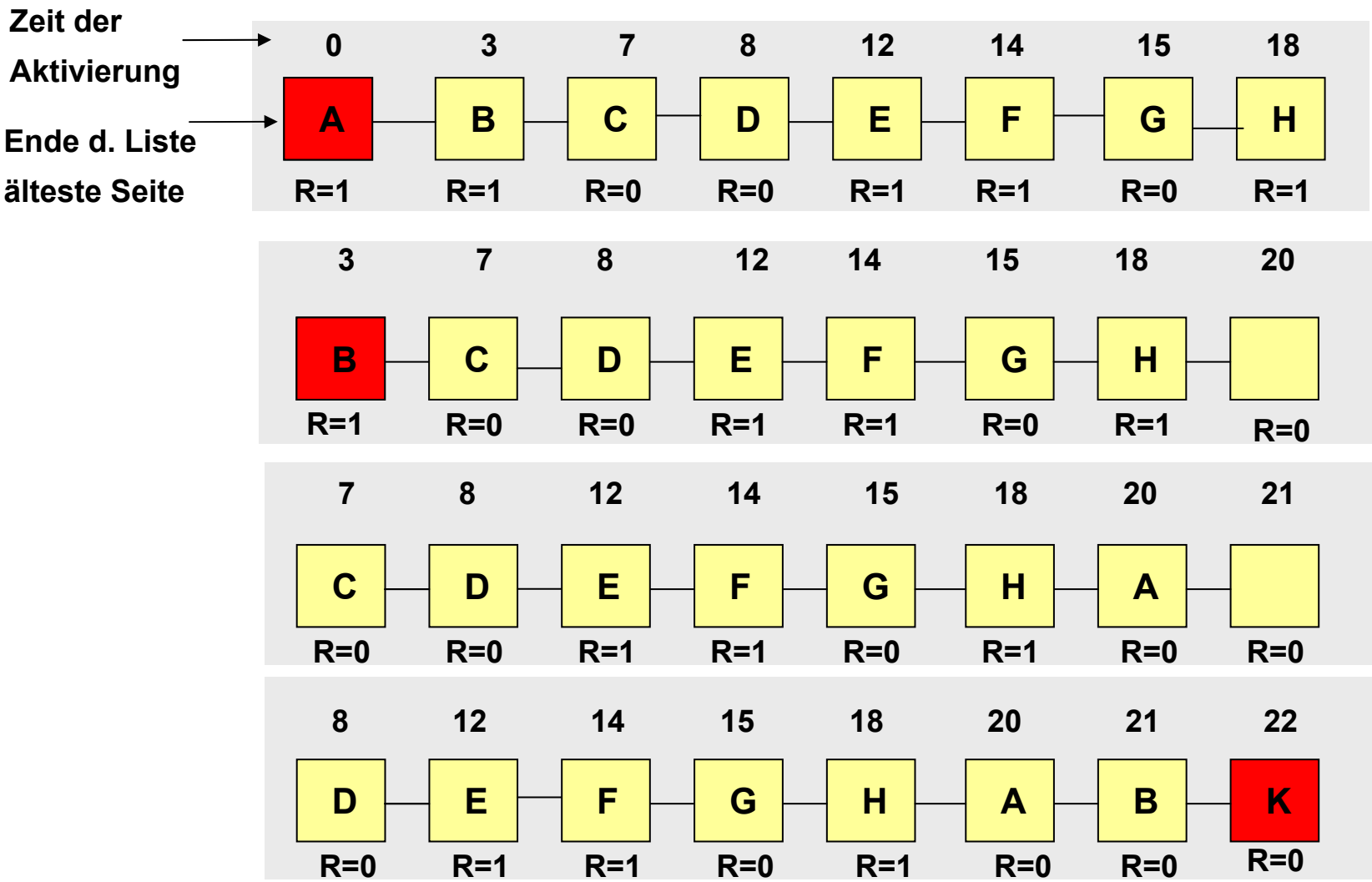
Variation von FIFO: Der "Second Chance" Algorithmus

Die Seiten sind in einer Liste nach FIFO geordnet



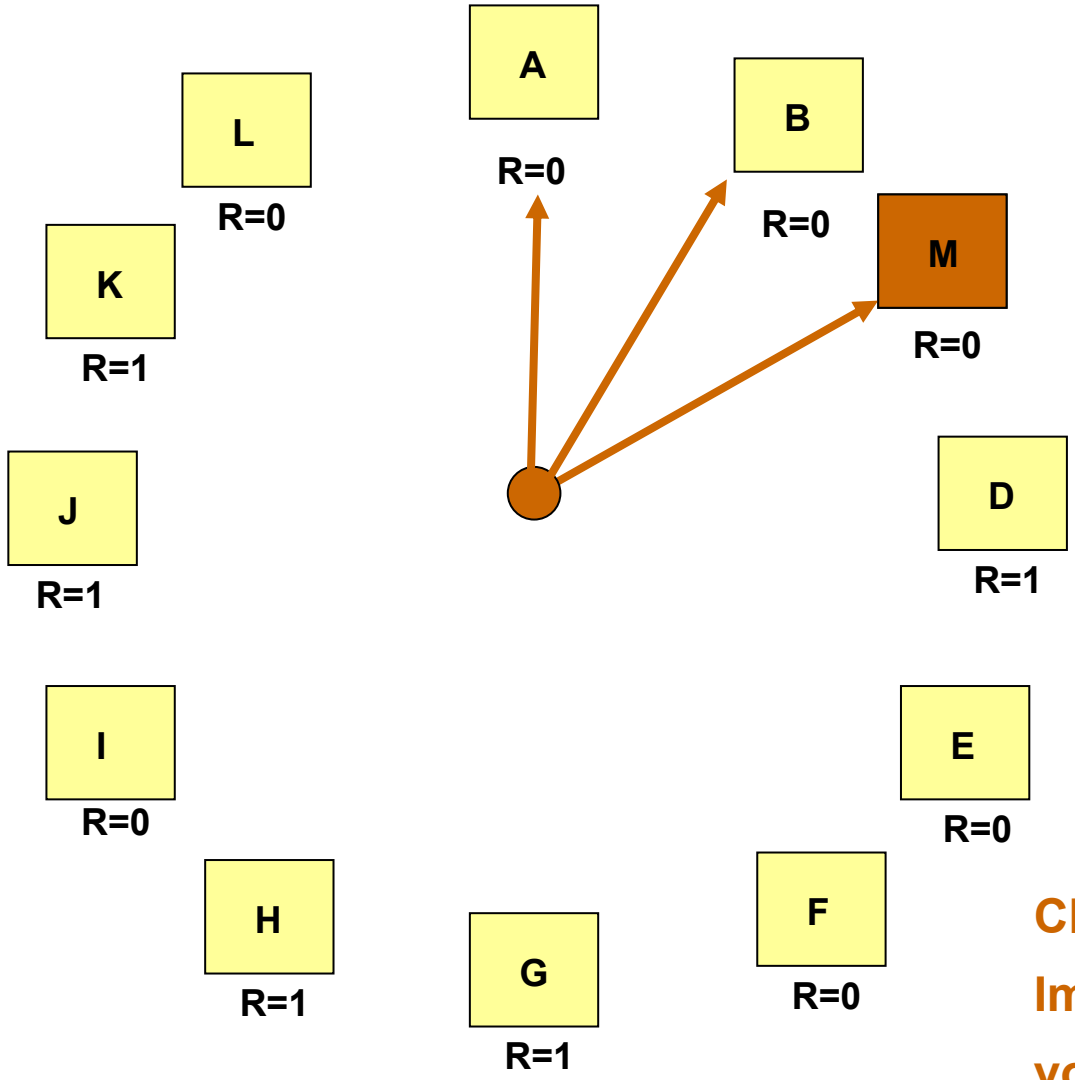
Das Verfahren ersetzt Seiten, die am frühsten eingelagert wurden UND die seit dem letzten Markierungsdurchlauf (sweep) nicht mehr referenziert wurden.

Der "Second Chance" Algorithmus



Seiten sind nach FIFO geordnet

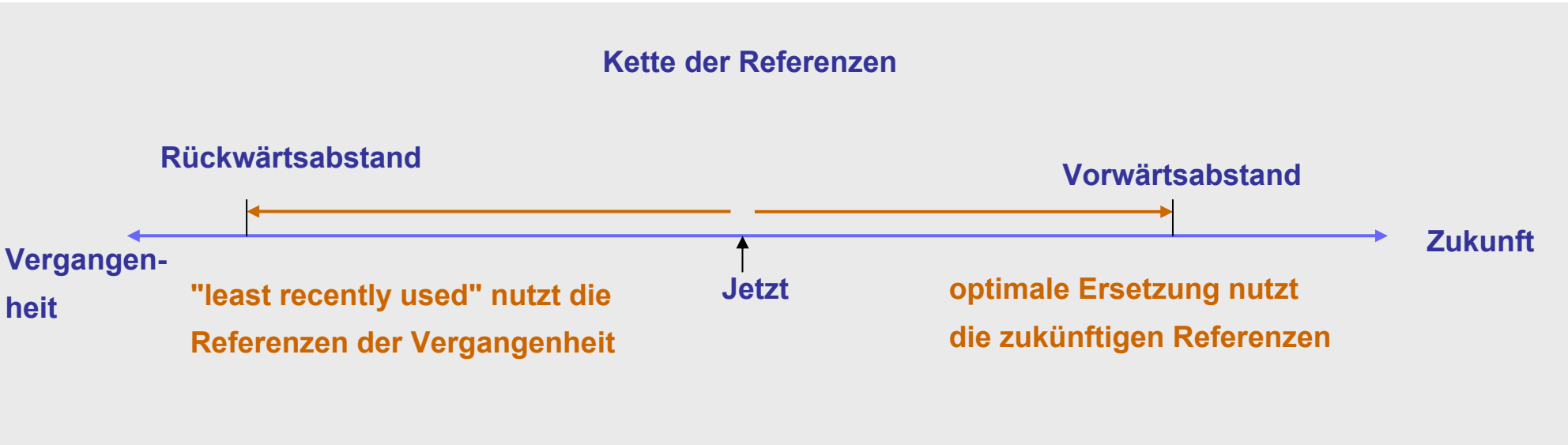
Implementierung von "second chance": Der "Clock" Algorithmus



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

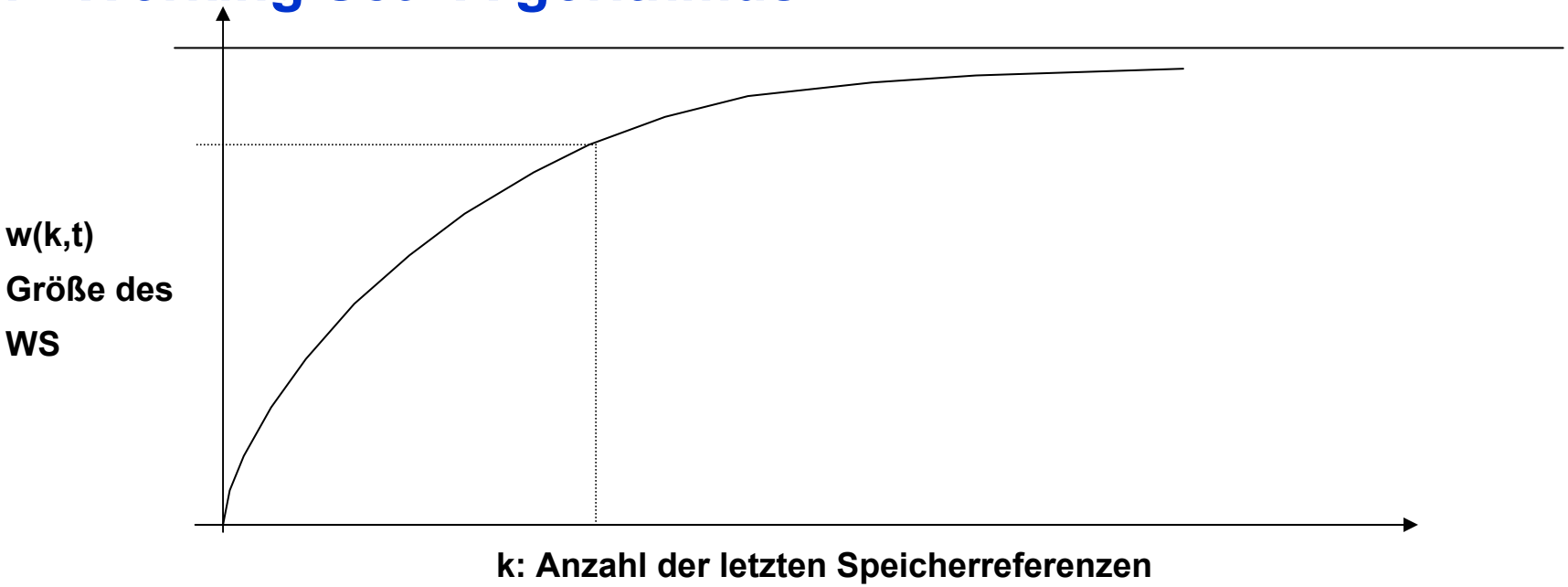
Least-Recently-Used

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

8 page faults

Seitenersetzungsstrategien

Der "Working Set" Algorithmus



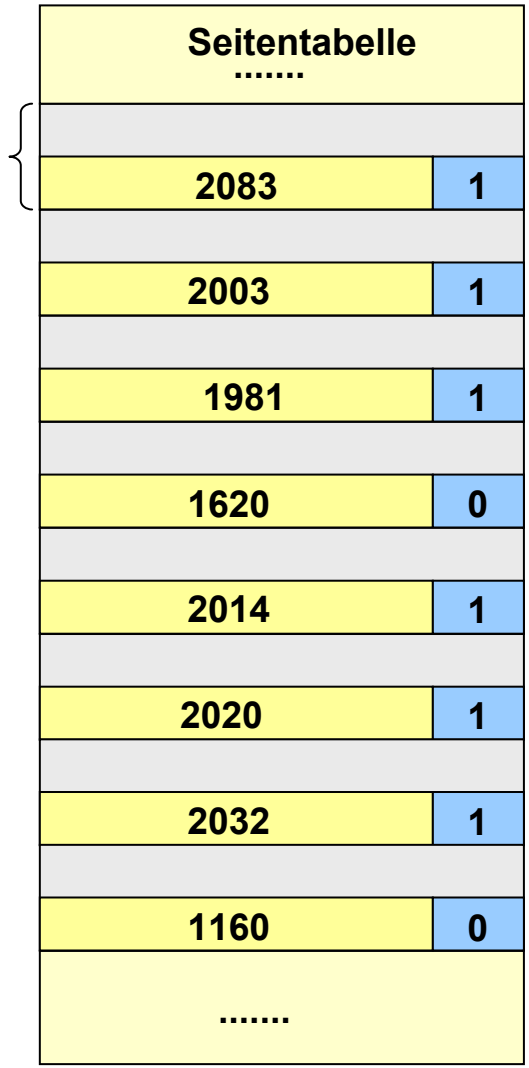
Die Menge der Seiten, die von einem Prozess in einem bestimmten Zeitfenster benutzt werden, wird als Working Set (WS) bezeichnet.

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.



R-Bit

virtual time: vt

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

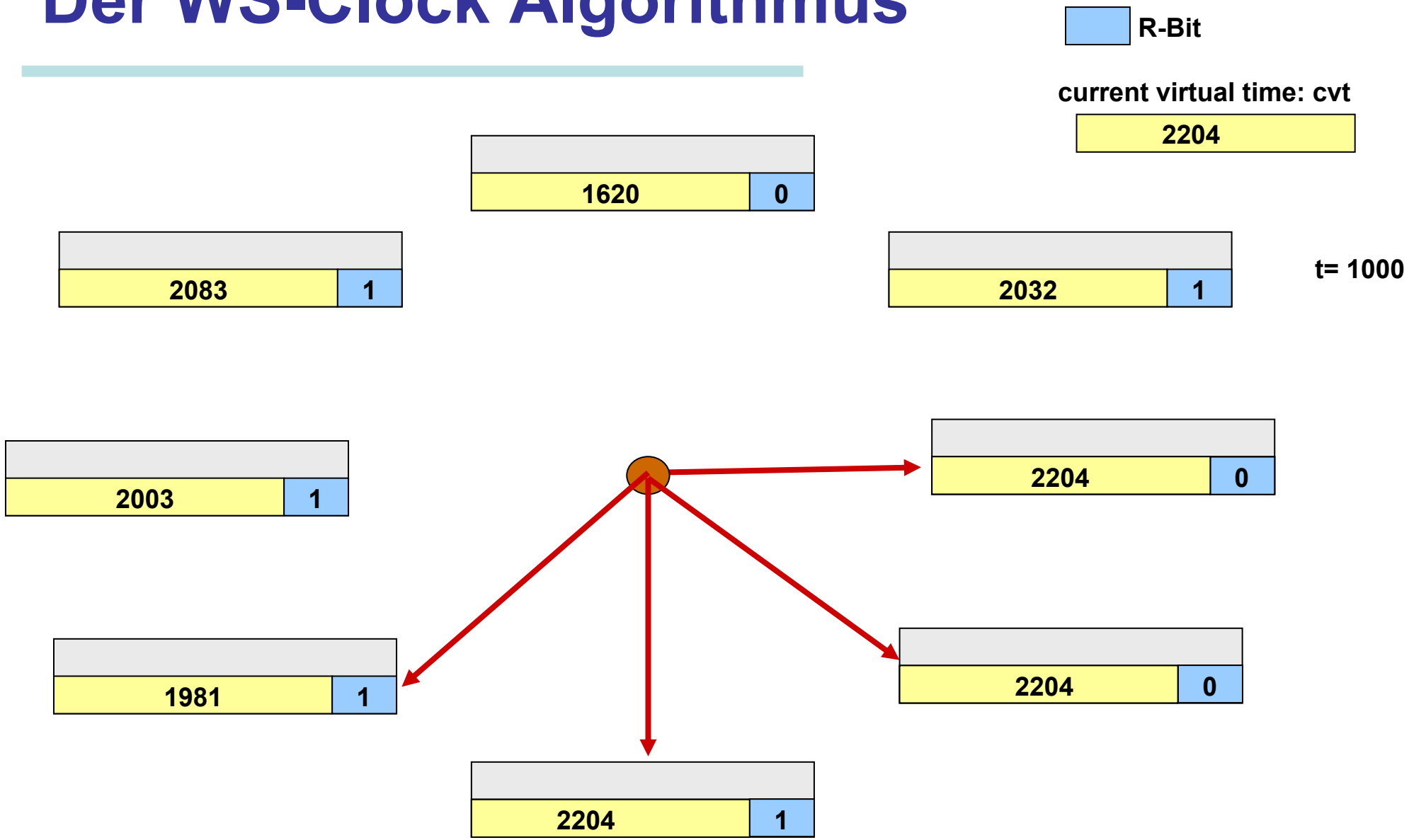
"virtual time" ist eine Prozess-lokale Repräsentation der Zeit, die mit Prozessbeginn startet.

current virtual time: cvt

2204

t = 800

Der WS-Clock Algorithmus



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, Implementatierungsprobleme
WSClock:	😈 😈 😈	😈 😈	gut + effizient

Seitenersetzungsstrategien

FIFO: Belady's Anomalie

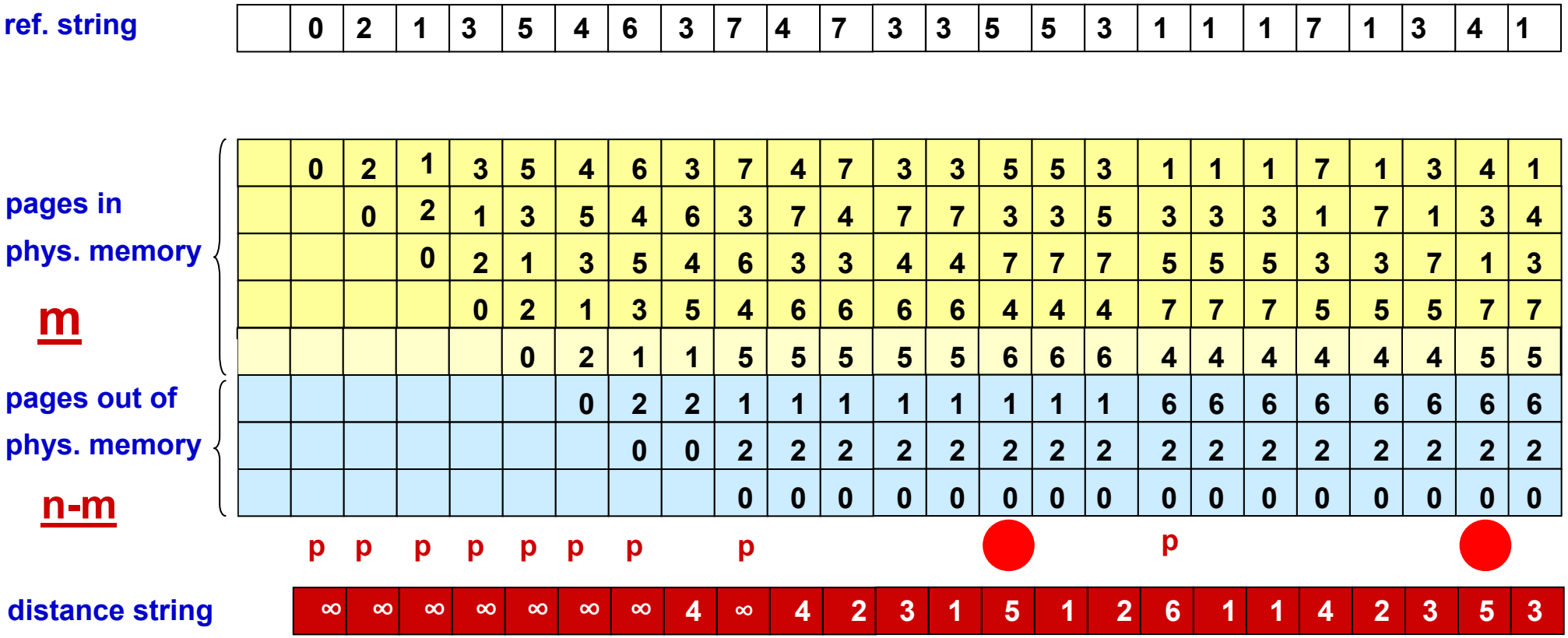
Ref. Folge		1	2	3	4	1	2	5	1	2	3	4	5
Kachelzuordnung im Realspeicher	frame 1	1	1	1	1	1	1	5	5	5	5	4	4
	frame 2		2	2	2	2	2	2	1	1	1	1	5
	frame 3			3	3	3	3	3	3	2	2	2	2
	frame 4				4	4	4	4	4	4	3	3	3
Kontrollstatus: Alter der Seite	frame 1	0	1	2	3	4	5	0	1	2	3	0	1
	frame 2	-	0	1	2	3	4	5	0	1	2	3	0
	frame 3	-	-	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. Das gilt auch für 2nd Chance.

the class of stack algorithms



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

m: # of frames, r: distance index

analysis of the distance chain

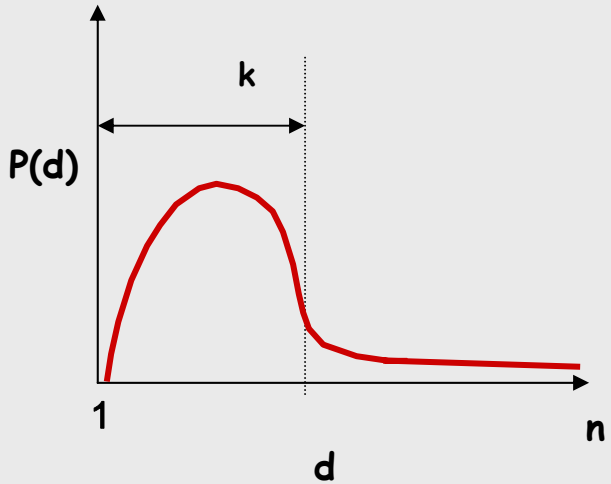
distance string

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

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

the distance string depends: 1.) on the reference string

2.) on the page replacement strategy

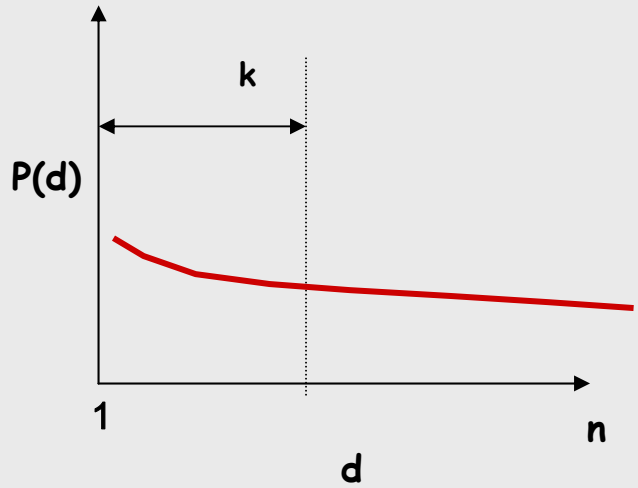


2 examples

k : number of frames

n: number of pages

d: distance string



predicting page fault rate

distance string

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

$C_1 \ldots C_\infty$

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

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

$F_1 = 20$	$C_2 + \ldots + C_\infty$
$F_2 = 17$	$C_3 + \ldots + C_\infty$
$F_3 = 14$	$C_4 + \ldots + C_\infty$
$F_4 = 11$	$C_5 + \ldots + C_\infty$
$F_5 = 9$	$C_6 + \ldots + C_\infty$
$F_6 = 8$	$C_7 + \ldots + C_\infty$
$F_7 = 8$	$C_8 + \ldots + C_\infty$
$F_8 = 8$	C_∞
$F_\infty = 8$	

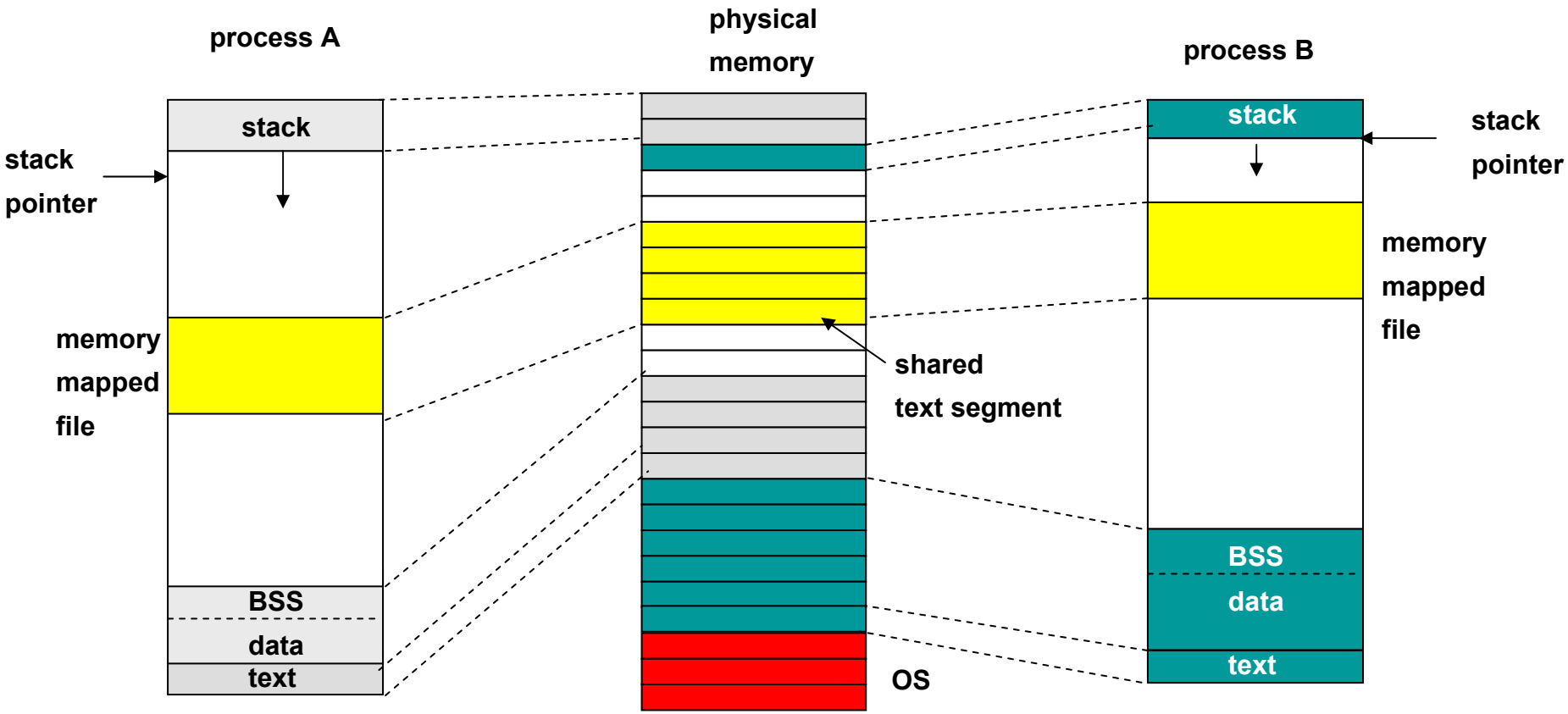
design issues for paging

- 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

Examples

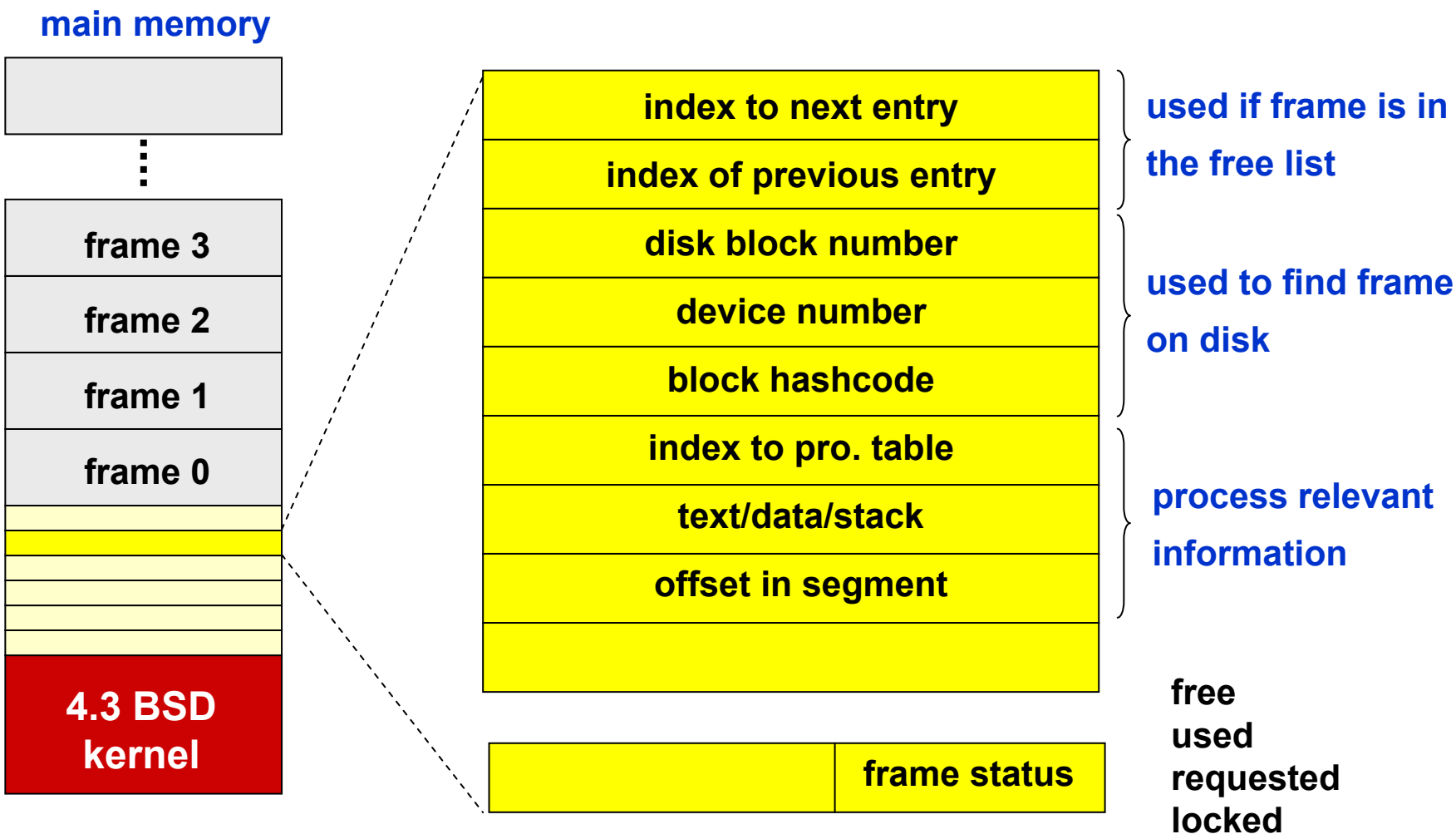
Unix "et al."

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: Core Map in 4BSD



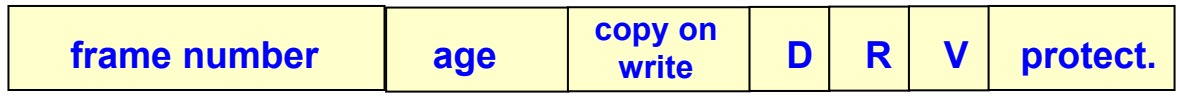
Example: Memory management in Unix et al.

Paging system

Data structures in System V Realease 4 (SVR4):

- Page table
- Disk block-descriptor table
- Frame data table
- Swap-use-table

Page table entry:

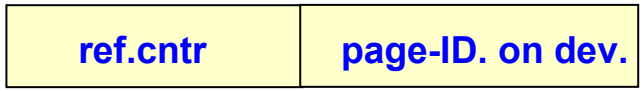


Disk block-descriptor:

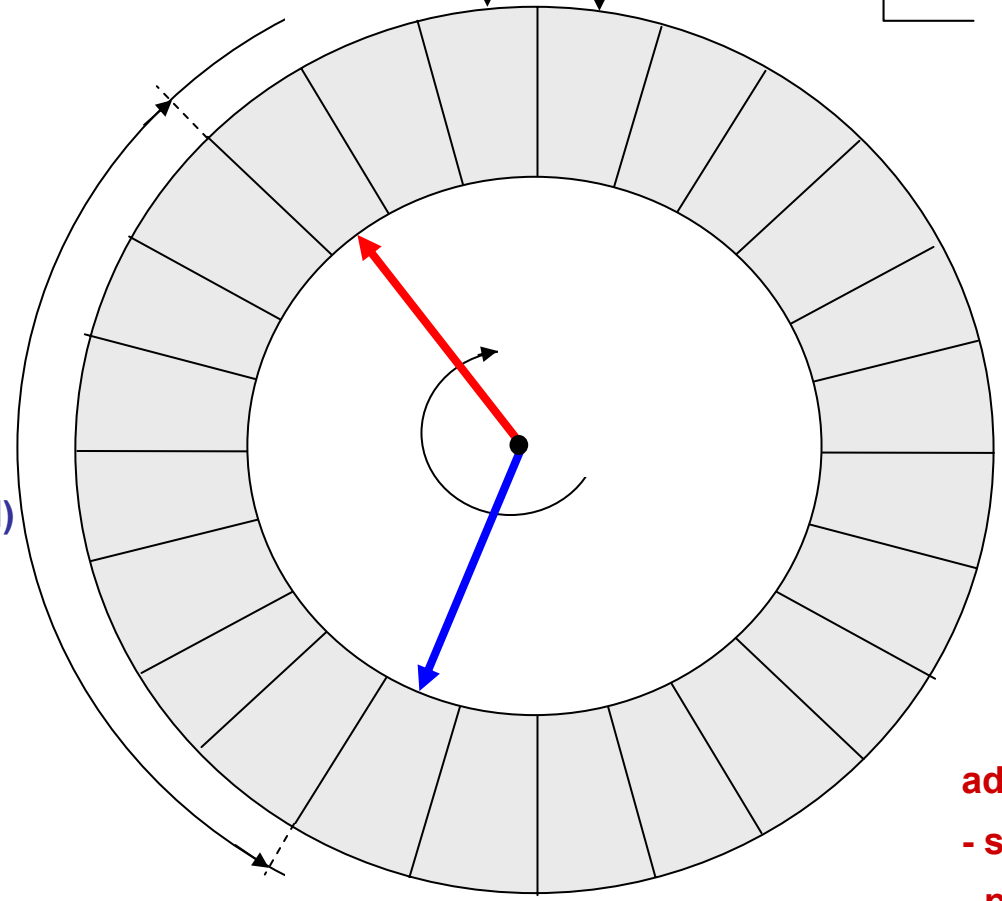
frame data table entry:

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

swap-use-table entry



releasing pages: clock with two pointers



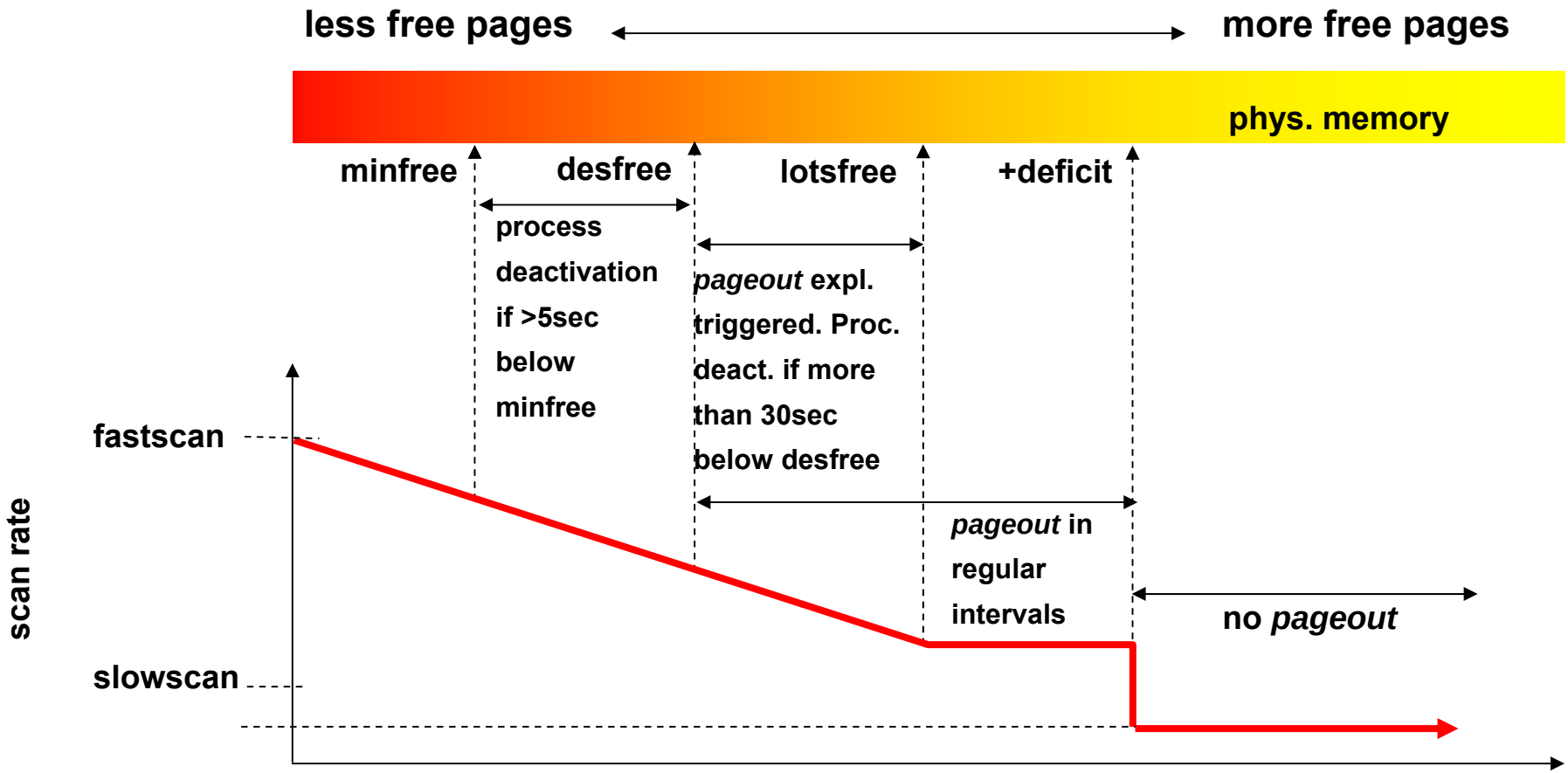
scans list of pages and clears the R-Bit

scans list of pages and moves pages with R=0 to the free list (pages that can be replaced) or replaces pages if necessary.

adjustable parameters for the algorithm:

- scan rate
- pointer distance

Memory management in Unix et al.

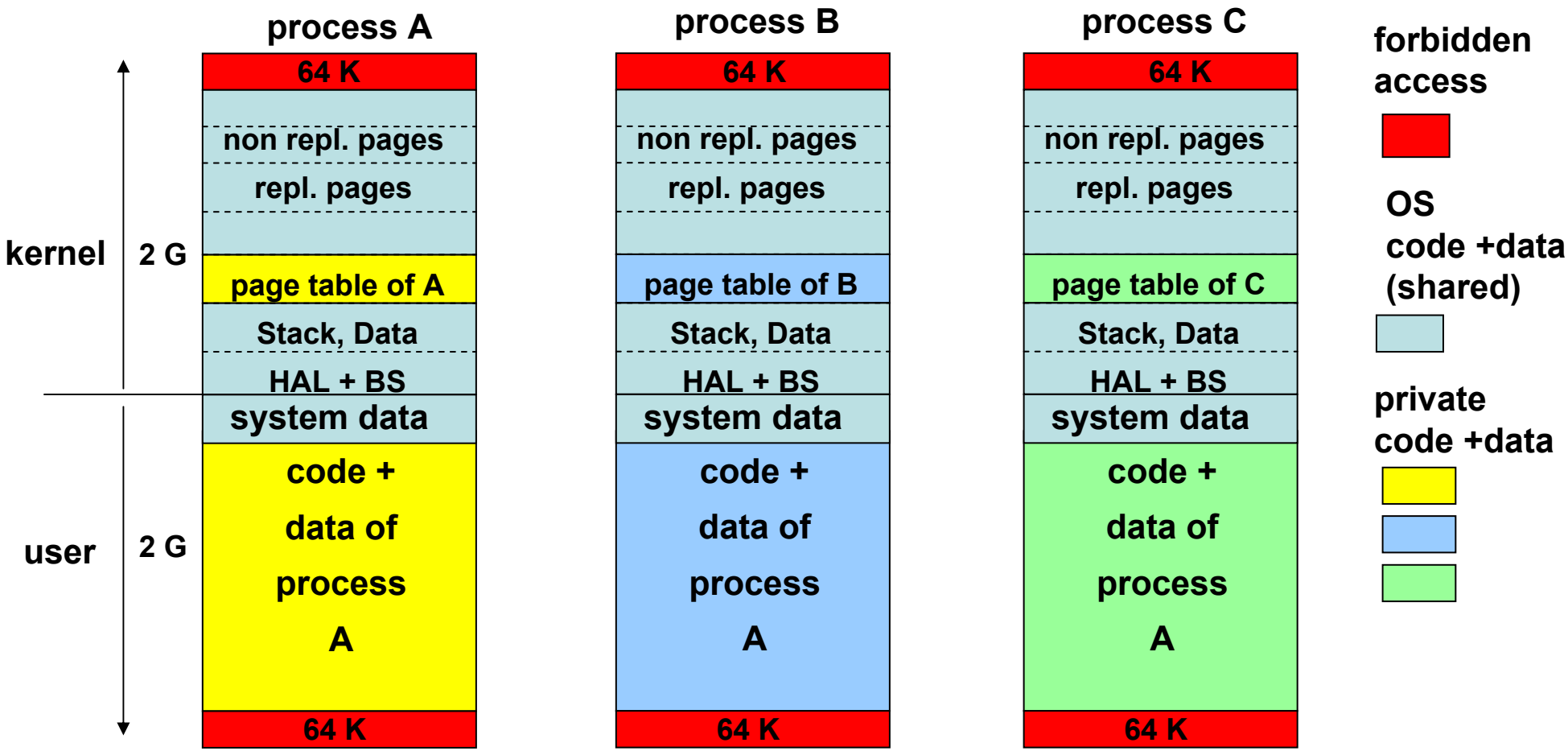


Examples

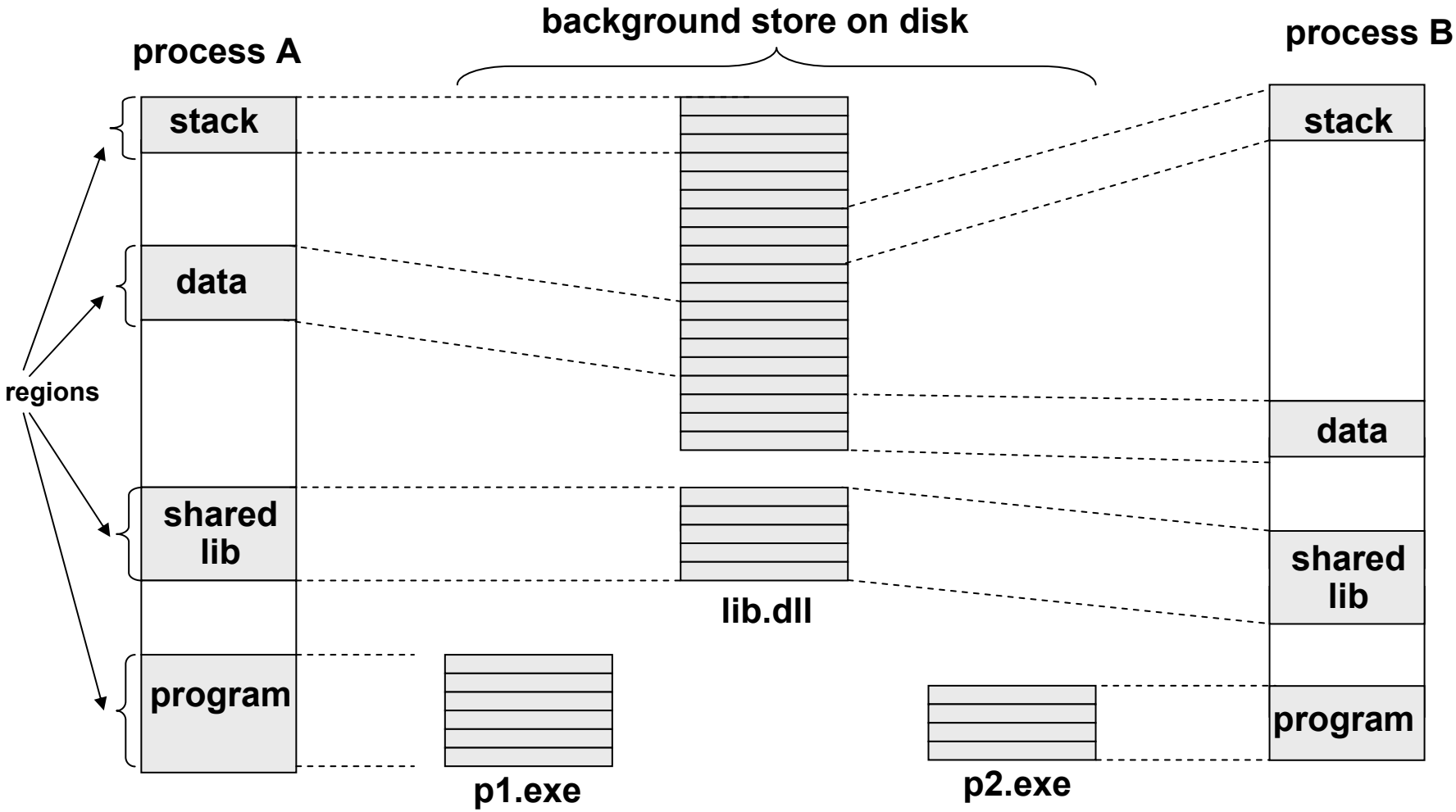
Windows 2000

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

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