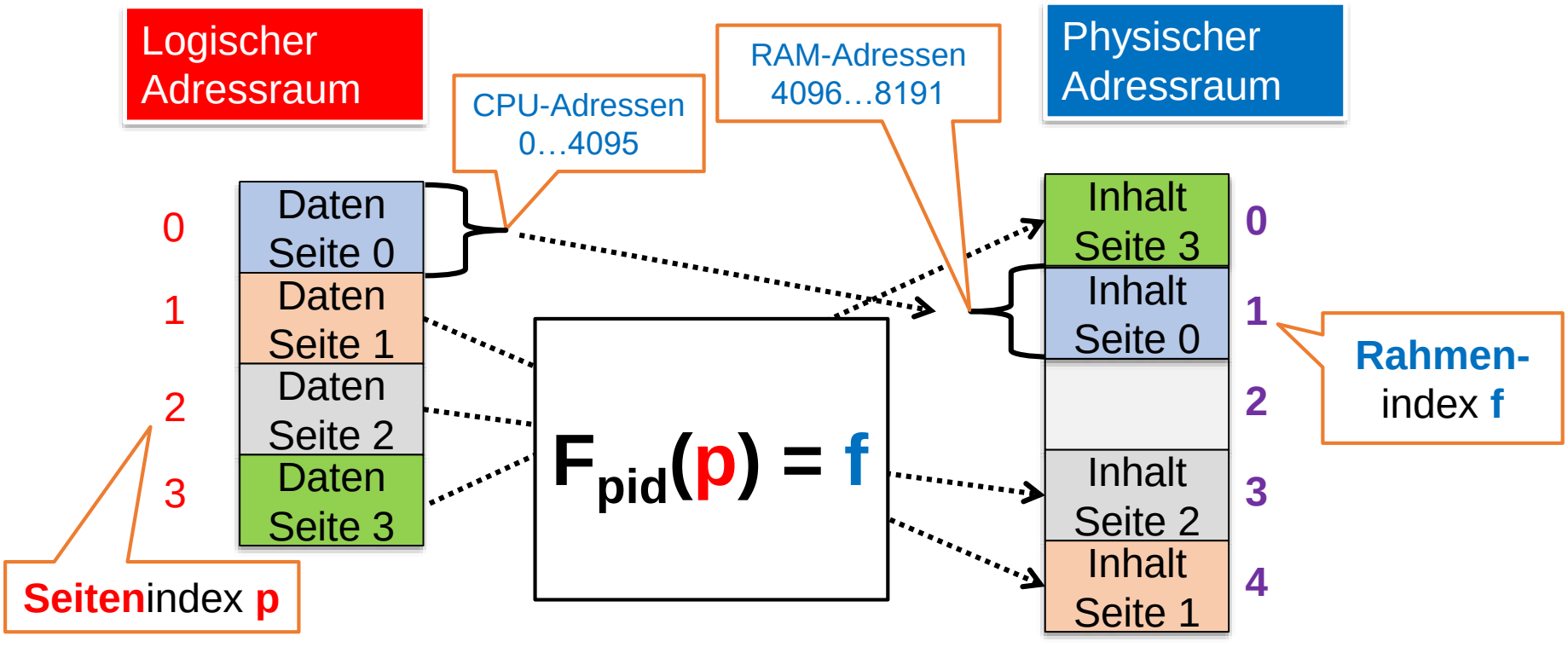
Betriebssysteme und Netzwerke Vorlesung 10

Artur Andrzejak

Umfragen: https://pingo.coactum.de/301541

Paging – Grundlagen (Fortsetzung)

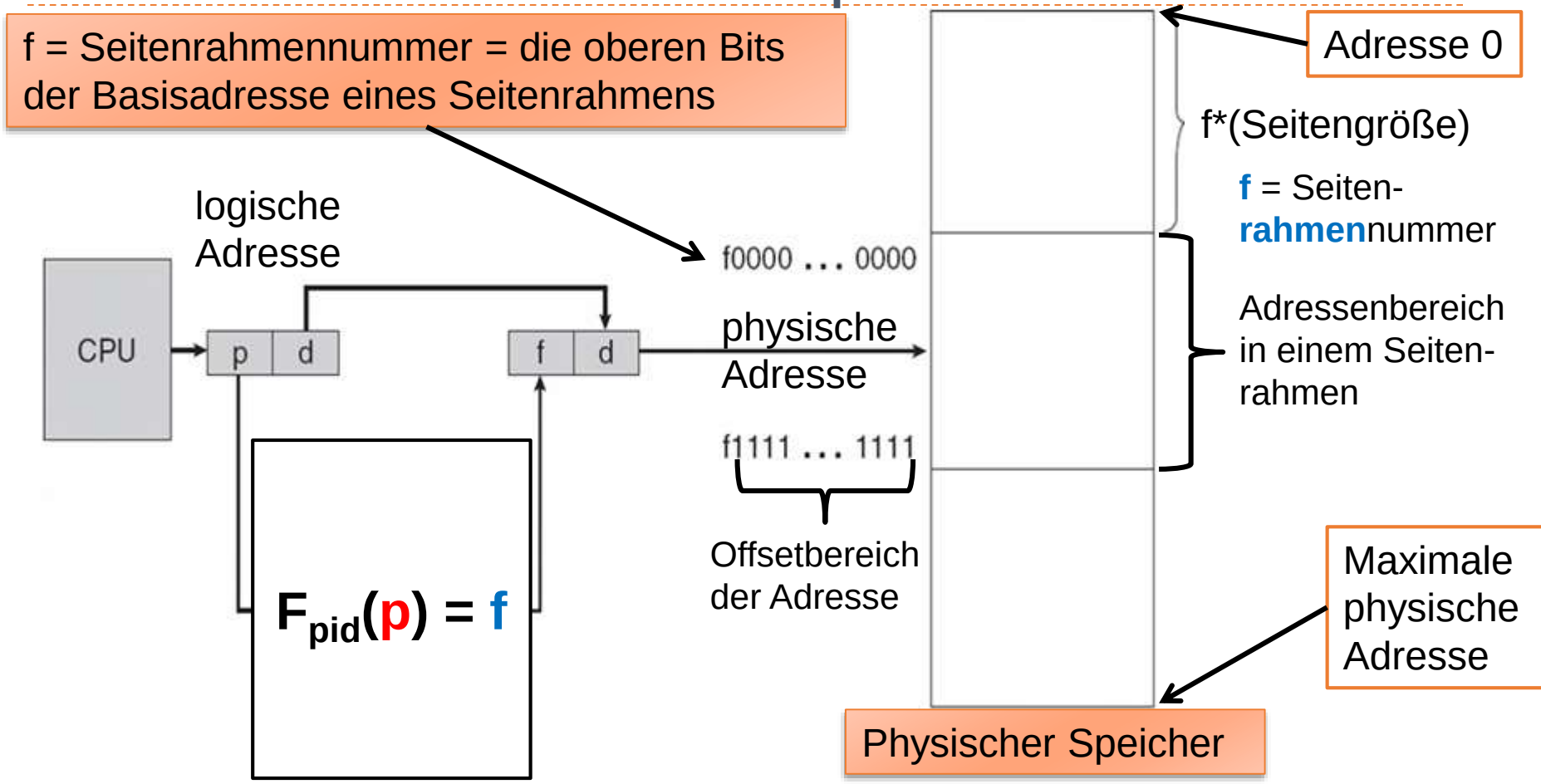
Paging: Adressenübersetzung in „Kacheln“



- **Paging** übersetzt “Kacheln” der logischen Adressen auf “Kacheln” der physischen Adressen
- Der Kern ist eine effiziente Funktion $F_{pid}(p) = f$, die Seitenindex **p** auf Rahmenindex **f** abbildet

pid = Prozess ID des aktuellen Prozesses

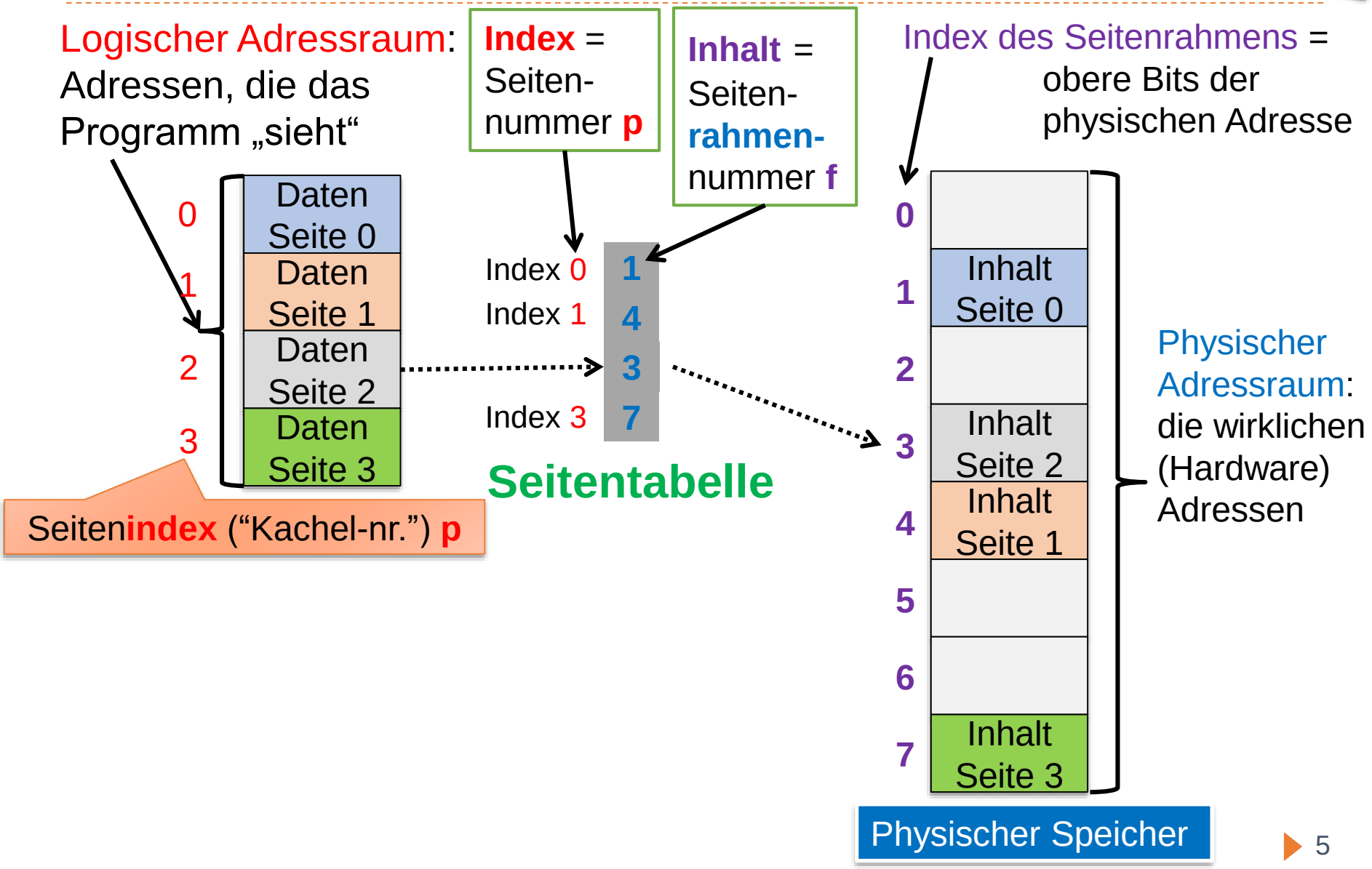
Adressübersetzung mit $F_{pid}(p) = f$



- Nachdem $F_{pid}(p) = f$ berechnet ist, ist die Übersetzung sehr einfach: obere Bits (= Wert **p**) einer logischen Adresse werden durch Bits mit Wert **f** ersetzt

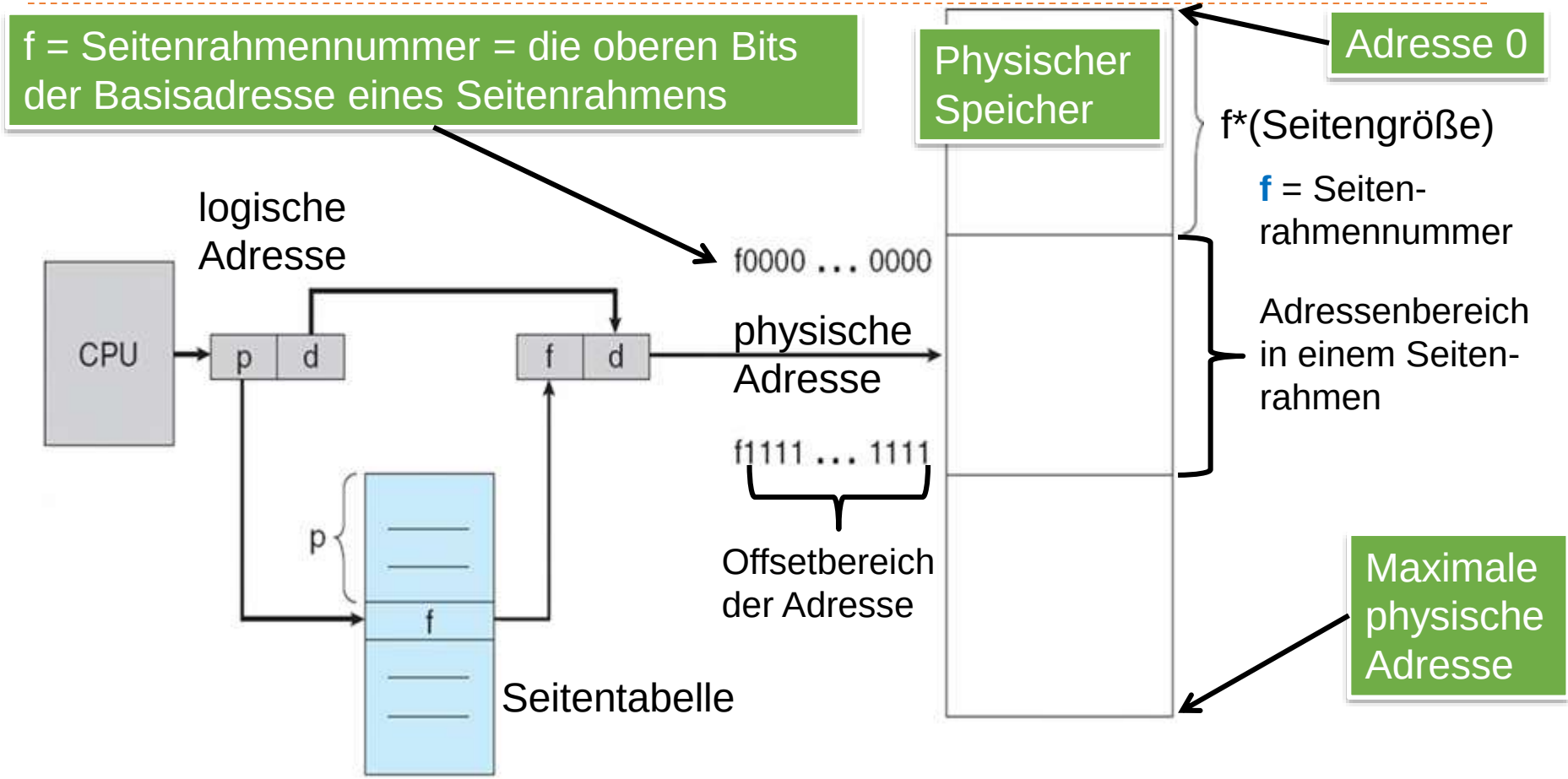
Wiederholung

Adressübersetzung durch Seitentabelle



Wiederholung

Adressübersetzung durch Seitentabelle



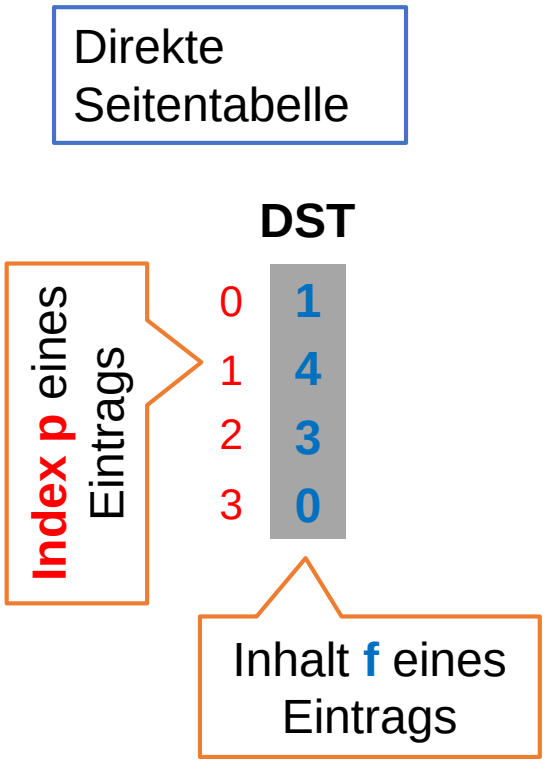
logische Adresse

Seitennummer	Seitenoffset
p	**d**

physische Adresse

Seiten**rahmen**nummer	Seitenoffset
f	**d**

Direkte (d.h. “Normale”) Seitentabellen



- Wie berechnet man $F_{pid}()$ mit einer direkten Seitentabelle?
- $F_{pid}()$ ist implementiert als ein Tabellen-**look-up**: sehr schnell
- In **DST**, Eintrag mit Index **p** enthält Wert **f** mit: $\mathbf{F_{pid}(p) = f}$
  - Index **p** = **Seiten**nummer
  - Inhalt **DST[p]** = **Rahmen**nummer
- Index **p** wird in DST nicht gespeichert, das wäre redundant
- Jeder Prozess (identifiziert via pid) braucht eine eigene Tabelle

Hat jeder Prozess eine eigene Seitentabelle?

- Verschiedene Prozesse benutzen i.A. denselben logischen Adressraum (der oft bei 0 anfängt) mit versch. Daten. Zugleich wird der Eintrag in der Seitentabelle durch (obere) Bits einer logischen Adresse ausgewählt. Gäbe es nur eine Seitentabelle, würden verschiedene Prozesse ggf. den gleichen Eintrag bei Adressenübersetzung der Seitentabelle holen (=> zur gleichen physischen Adresse übersetzen), was nicht gewünscht ist.
- Die Seitentabelle könnte aber so implementiert sein, dass man bei jedem Eintrag zwischen verschiedenen Prozessen unterscheidet, d.h. ein "Prozess-ID-Label" den Index der Seitentabelle erweitert. Bei den gängigen Prozessoren wie die x86-Architektur ist das aber nicht der Fall, das zeigt ein Blick auf die MMU von x86 (z.B. hier - auch eine interessante Lektüre an sich).
- Es wird bei Linux tatsächlich der Inhalt des SP3 Systemregisters bei einem Taskwechsel ggf. ausgetauscht (siehe hier unter 6.3, "change Memory context (change CR3 value)"). Dieser Register enthält die Basisadresse der Seitentabelle (d.h. PTBR, siehe hier).
- Es ist aber denkbar, dass andere Architekturen / BS mit einer einzigen Seitentabelle und Labels pro Prozess arbeiten (das wird z.T. auch bei *invertierten* Seitentabellen gemacht) (siehe hier).
- Wichtig ist nur, dass es für jeden Prozess eine separate *Abbildung* von logischen auf die physische Adressen möglich ist. Wir können also vereinfachend sagen, dass es eine Seitentabelle pro Prozess gibt.

Das ist Antwort auf eine Frage aus dem WS 2011/12 – hier als Ergänzung

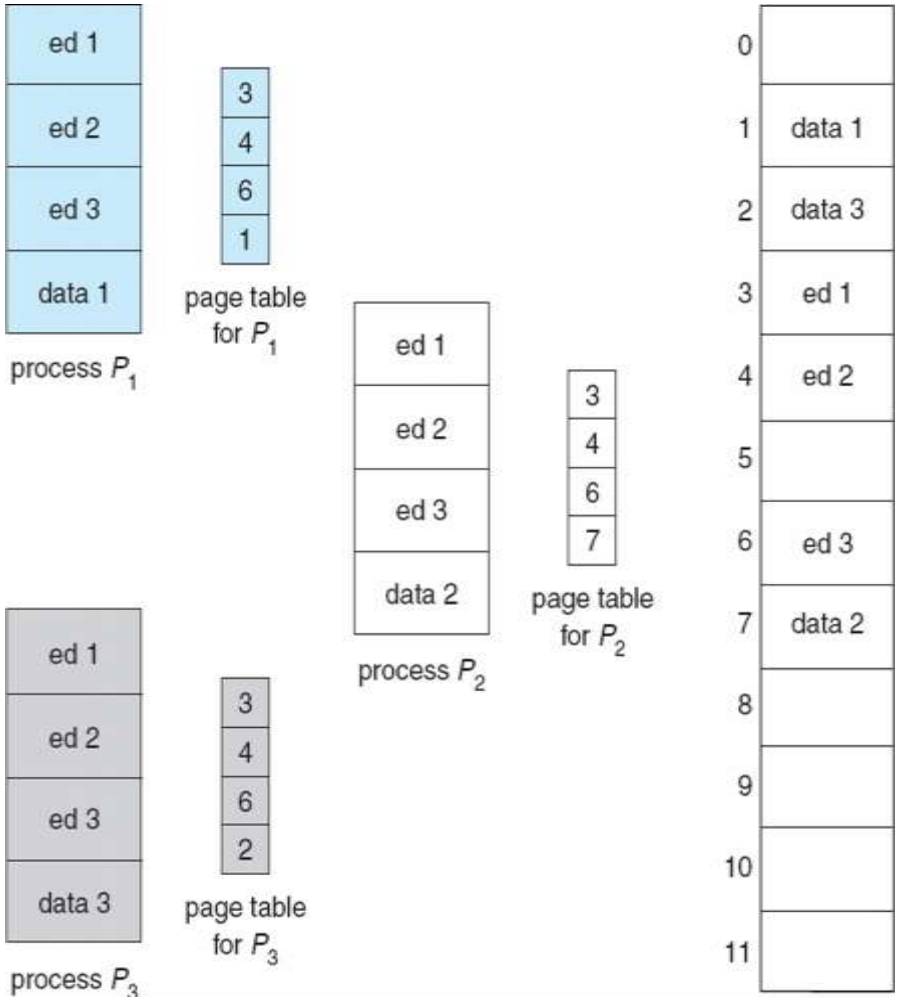
Paging und Prozesse

- Mit jedem Prozess wird eine neue Seitentabelle nur für diesen Prozess erzeugt
  - Die nötigen Seitenrahmen können überall liegen
- Aber das BS muss die Verwendung des physischen Speichers nachverfolgen
- Dazu dient die (globale) **frame table** (**Seitenrahmen-tabelle**) mit Informa-tionen wie …
- Welche Seitenrahmen sind belegt und welche frei?
- Falls belegt, zu welchem Prozess / welchen Prozessen gehört der Rahmen?

vor dem Laden nach dem Laden

Gemeinsam Benutzte Seiten (Shared Pages)

- Nur-Lese Code (**reentranter Code**) kann gemeinsam von mehreren Prozessen genutzt werden
  - Z.B. DLL‘s
- Auch gemeinsame Daten:
  - Shared-Memory Bereiche für IPC
- Auch hier wird Paging benutzt:
  - Mehrere Prozesse haben Seitentabellen mit gleichen Seitenrahmennummern (hier ed 1 -- ed 3)



Paging – Geschwindigkeit der Übersetzung

Seitentabellen – wo befinden sie sich?

- Rechner mit wenig Speicher, z.B. **DEC PDP-11**
  - 16-Bit Adressraum, Seitengröße 8kB
    - => insgesamt nur 8 Einträge in der Seitentabelle
  - Die Seitentabelle = ein Satz von CPU Registern
- Bei **IA32**-Architekturen (d.h. ab Intel Pentium)
  - Bei dieser Architektur hat ein Eintrag der Seitentabelle immer 4 Bytes (32 Bits), siehe https://wiki.osdev.org/Paging
  - Bei 4 GB Speicher (2^{32}) und 4kB Seitengröße (2^{12}) …
    - => 2^{20} ~ 1 Mio. (bzw. 1 „MB“) Einträge der Seitentabelle
  - Je Eintrag 4 Bytes => 4 MB *pro Prozess* nötig!
  - Die Seitentabelle muss also im Hauptspeicher liegen!
    - Und nicht als Register der CPU/MMU

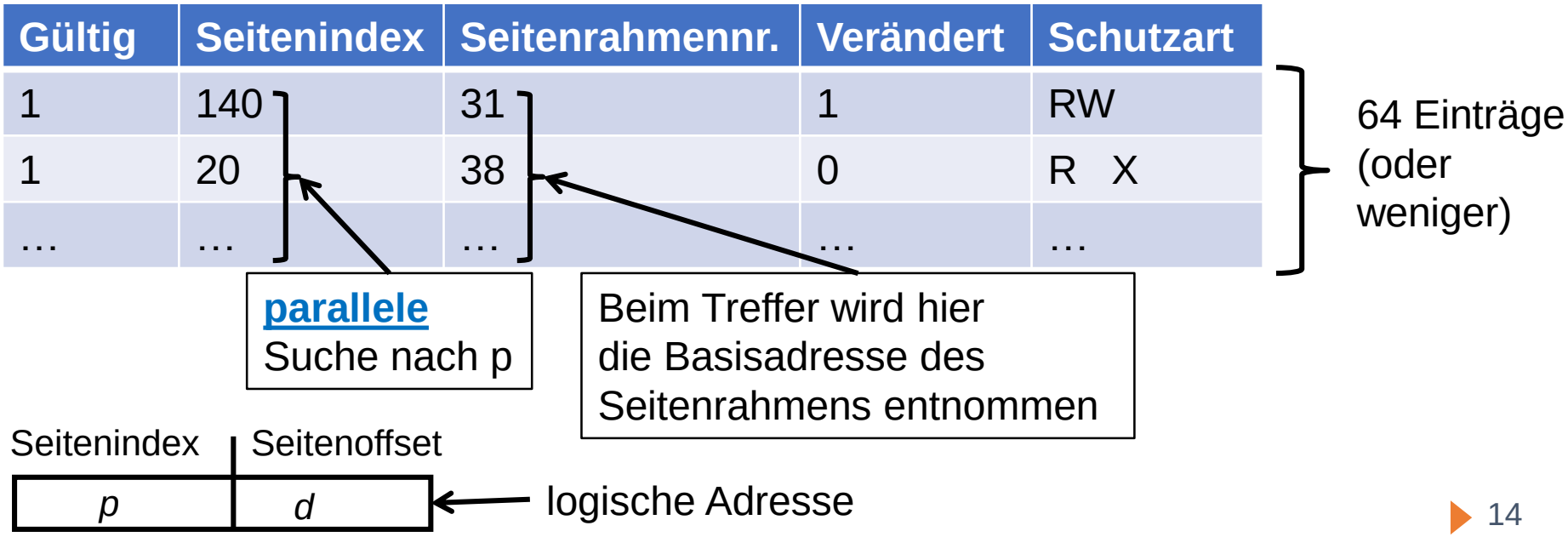
Die Seitentabelle im Hauptspeicher

- Die Seitentabelle muss im Hauptspeicher (und nicht in der CPU) gespeichert sein
- Zwei Probleme:
  - A. Erhöhte Zugriffszeit auf den Speicher
  - B. Hoher Speicherbedarf einer Seitentabelle

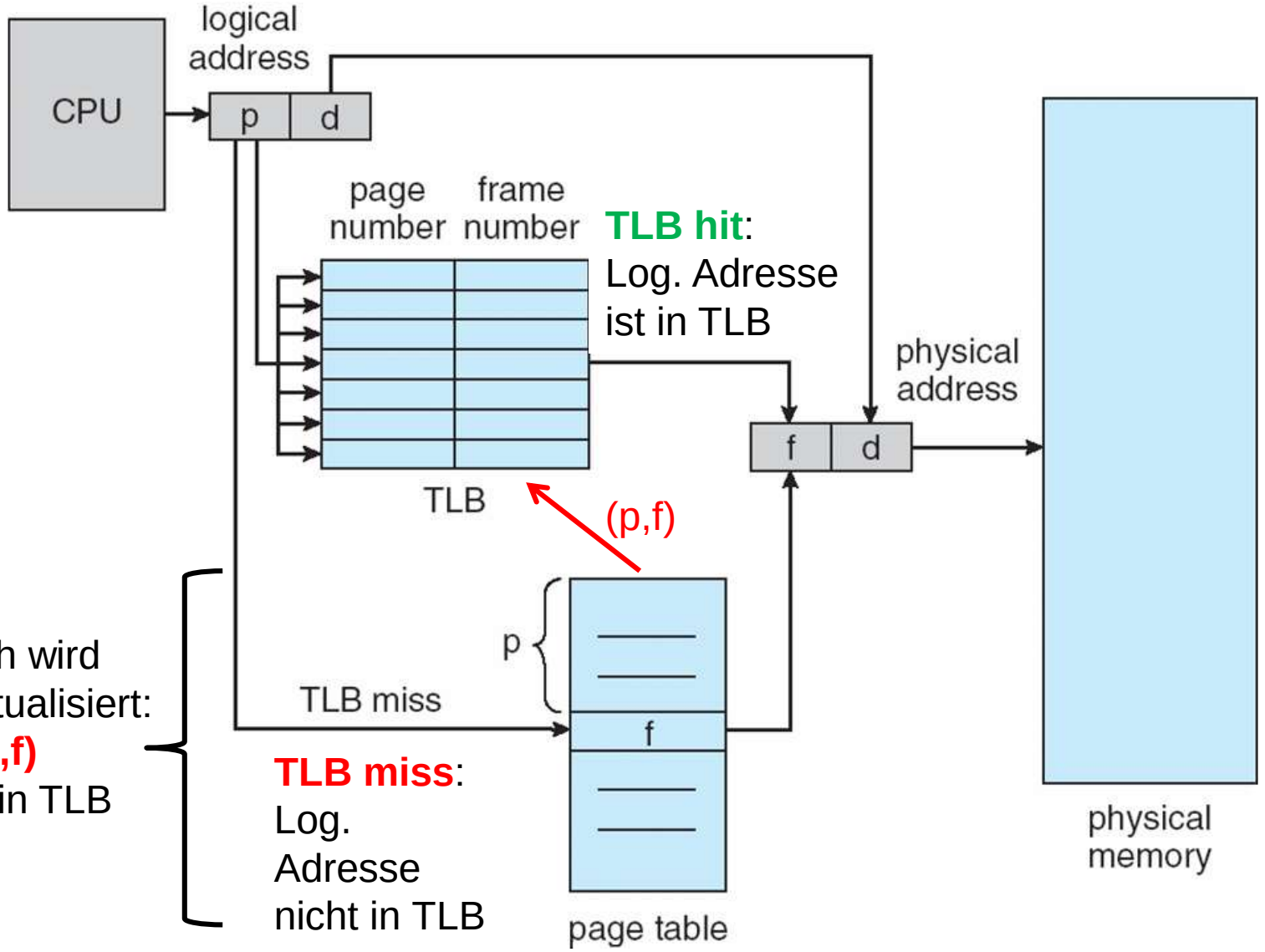
- Problem: Jeder Speicherzugriff muss in zwei Zugriffe übersetzt werden:
  - (1) Lesen aus der Seitentabelle und
  - (2) Eigentliches Lesen/Schreiben auf den Speicher
- Naiv umgesetzt - das dauert zu lange!

Beschleunigen der Adressenübersetzung

- Die Lösung nutzt spacial locality aus:
  - Sehr wenige Seiten werden sehr häufig verwendet
- Man verwendet einen Cache für die Einträge der Seitentabelle (spezielle Hardware in MMU)
  - Den **Translation Look-Aside Buffer** (**TLB**)



TLB – Umsetzung in Hardware



TLB – Umsetzung in Software

- Falls der Seitenindex p in TLB nicht vorhanden ist, muss in der Seitentabelle nachgeschaut werden
- Behandlung durch Software, in nur wenigen Zyklen:
  - Das BS holt den Index f des Seitenrahmens zu p (d.h. den p-ten Eintrag der Seitentabelle) aus der Seitentabelle
  - BS ersetzt einen Eintrag von TLB durch (p,f, flags…)
  - Startet den Befehl neu, der den TLB-Fehler auslöste
- Enge Verschmelzung von Hardware (MMU) und BS
  - In vielen modernen RISC-Prozessoren: SPARC, MIPS, Alpha, HP PA

Effektive Zugriffszeit

- T = Übersetzungszeit durch TLB
- h = hit ratio = Prozentsatz der Fälle, bei denen der Seitenindex in TLB gefunden wird
- Annahme: Speicherzugriff = eine (1) Zeiteinheit
- Die **effektive Zugriffszeit** (**Effective Access Time**) ist der Erwartungswert der Zufallsvariable X, die die (komplette) Zugriffszeit auf den Speicher modelliert
  - Fall A - Seitenindex in TLB: Prob(A) = h, X = ?
  - Fall B - Seitenindex nicht in TLB: Prob(B) = 1-h, X = ?
- Fall A: Prob(A) = h, X = 1+T
- Fall B: Prob(B) = 1-h, X=2+T
- Damit: **EAT** = E[X] = (1+T)h + (2+T)(1-h) = **2 + T - h**

Paging – Größe der Seitentabelle

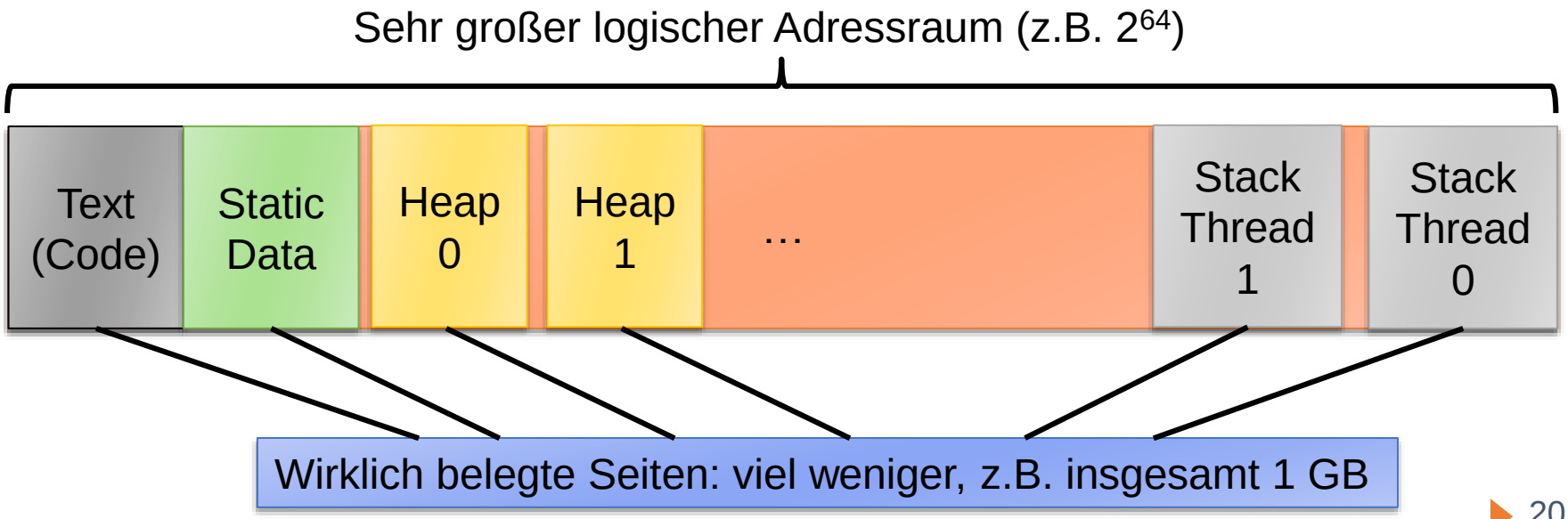
Problem: Größe der Seitentabelle (ST)

- Logischer Adressraum bei IA32
  - 4kB Seitengröße (2^{12}), 4 GB Speicher (2^{32}) => 2^{20} ~ 1 Mio. (bzw. 1 „MB“) Einträge der Seitentabelle
  - Jeder ST-Eintrag hat 4 Bytes (https://wiki.osdev.org/Paging)
    - => Vollständige Seitentabelle hat 4 MB *pro Prozess*!

- Ein 64-Bit Prozessor hat bis zu 2^{64} Adressen
  - Bei 4 kB Seitengröße hätte eine vollständige Seitentabelle $2^{(64-12)}=2^{52}$ Einträge, je mit 64 Bits => 2^{55} Bytes Speicher!
  - Hinzu kommt: Der logische Adressraum übersteigt die Größe von heutigen RAMs um ein Mehrfaches

Problem: Größe der Seitentabelle

- Brauchen wir wirklich eine vollständige Seitentabelle?
  - Wir können nicht mehr Seiten haben, als es physische Rahmen gibt (später: doch!)
- Idee: wir geben den Prozessen die Freiheit bei der Verwendung des logischen Addressraumes
- … ABER die Gesamtanzahl der verwendeten logischen Seiten soll (pro Prozess) beschränkt sein



Neuartige Seitentabelle – mit „log. Lücken“

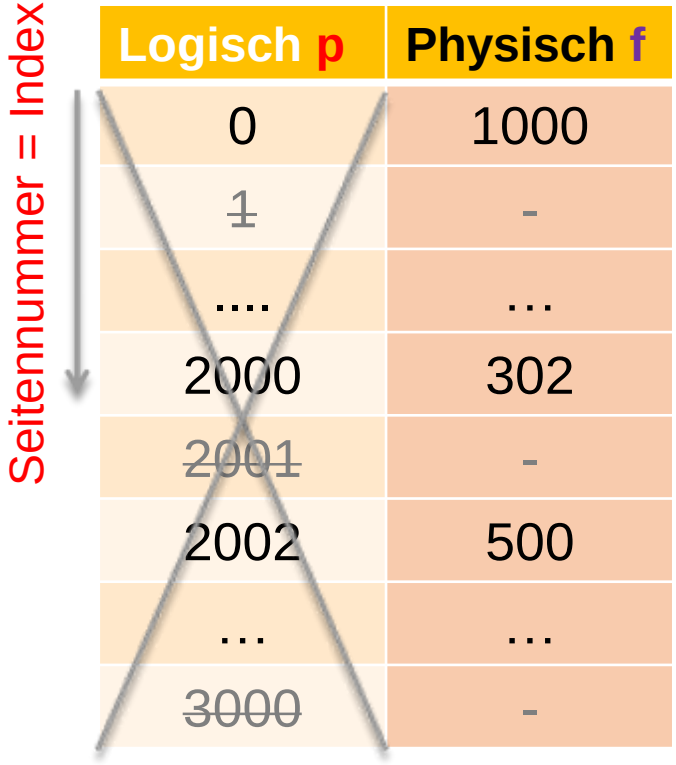
- Idee: wir halten in einer „neuartigen“ Seitentabelle die Einträge nur zu solchen Seiten, die überhaupt verwendet wurden
- Für eine zu übersetzende Seitennummer **p** durchsuchen wir die 1. Spalte der Tabelle bis zum Erfolg, holen dann die Rahmen# **f**
- Wie unterscheidet eine direkten Seitentabelle von dieser neuen Tabelle?

Logisch p	Physisch f
0	1000
~~1~~	-
....	…
2000	302
~~2001~~	-
2002	500
…	…
~~3000~~	-

Logische Seiten nicht verwendet

Direkte vs. „Neuartige“ Seitentabelle

- *Wie unterscheidet sich diese Tabelle von der direkten Seitentabelle?*
- Bei der direkten Seitentabelle fehlt die 1. Spalte, da wir dort durchgehende Seitennummern annehmen
- Und statt nach **p** zu suchen, können wir einfach die Zeile **p** indexieren



Neuartige Seitentabelle – mit „log. Lücken“/2

- Wie können wir die „neuartige“ Seitentabelle verkleinern?
- 1. Einträge entfernen, die nicht verwendet werden
- 2. Können wir auch die 2. Spalte „wegrationalieren“?

Logisch p	Physisch f
....	…
2000	302
…	…
2002	500
…	…
0	1000

Logisch p	Physisch f
0	1000
~~1~~	-
....	…
2000	302
~~2001~~	-
2002	500
…	…
~~3000~~	-

Logische Seiten nicht verwendet

Neuartige Seitentabelle /3

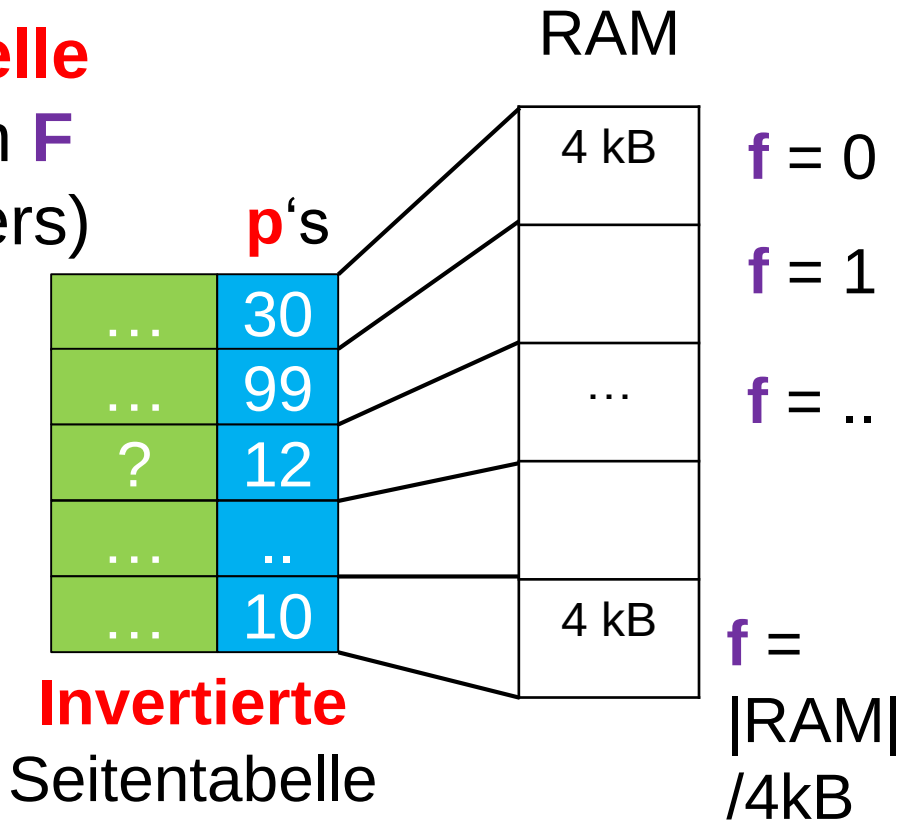
- Wir sortieren die Einträge nach der 2. Spalte (= Rahmennr. **f**)
- … Und führen zu jeder möglichen Rahmennummer **f** eine Zeile ein
- Da die Rahmennummern nun durchgehen sind, können wir die 2. Spalte streichen!
- Speicherbedarf?

Logisch p	Physisch f
59	0
71	1
....	…
2000	300
…	301
60	302
…	…
0	1000

- Die Größe der Seitentabelle ist nun proportional zum vorhandenen physischen Speicher und nicht zum logischen Adressraum – ist das viel?

Ergebnis: Invertierte Seitentabellen

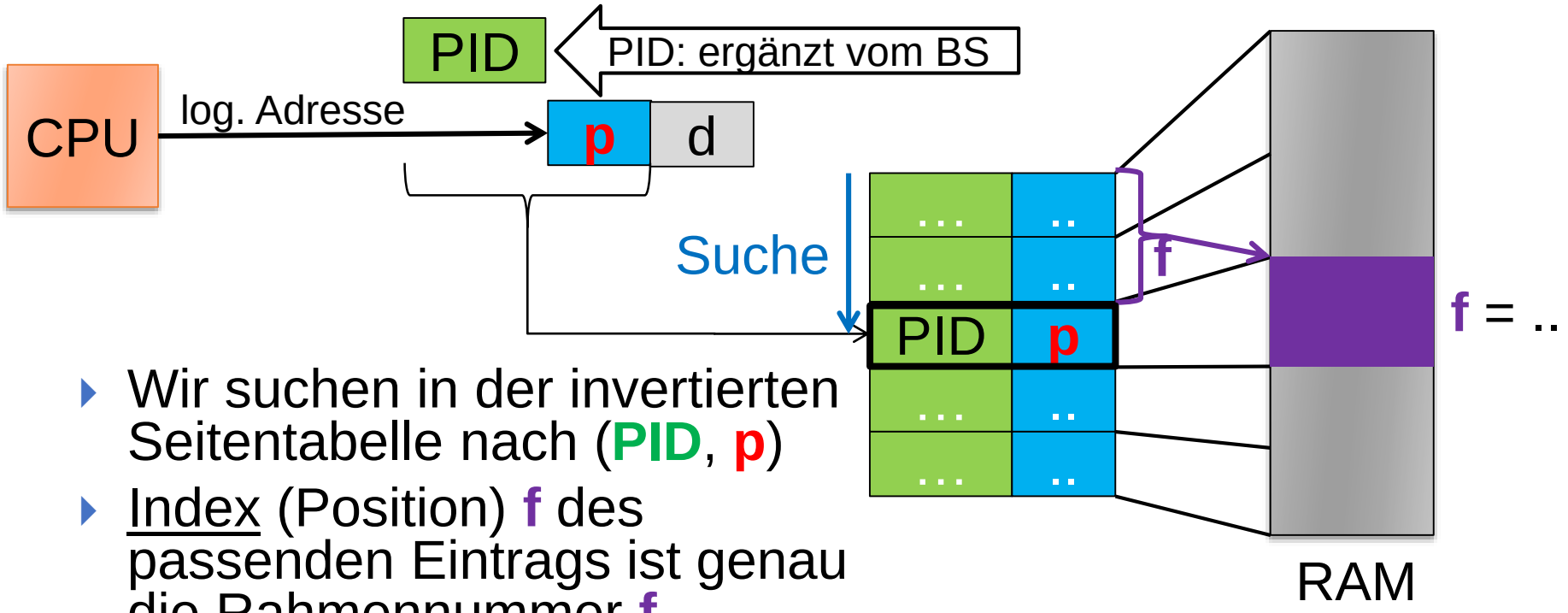
- Eine **invertierte Seitentabelle** speichert zu jedem Rahmen **F** (=Kachel des phys. Speichers) die Seitennummer **p** der Seite (=Kachel des log. Speichers), die auf **F** abgebildet wird
- Wie viele solcher Tabellen brauchen wir?



- Es reicht eine einzige, wenn wir zu jeder Zeile (= log. Seite in Verwendung) auch die Prozessnummer abspeichern

Invertierte Seitentabellen - Details

- Der Eintrag mit Index **f** besteht aus:
  - Prozess-Nummer **PID** des „Besitzer-Prozesses“ P des Seitenrahmens und …
  - …Der Seitennummer **p** der Seite im logischen Addressraum (von P), die auf Rahmen mit Nummer **f** abbildet



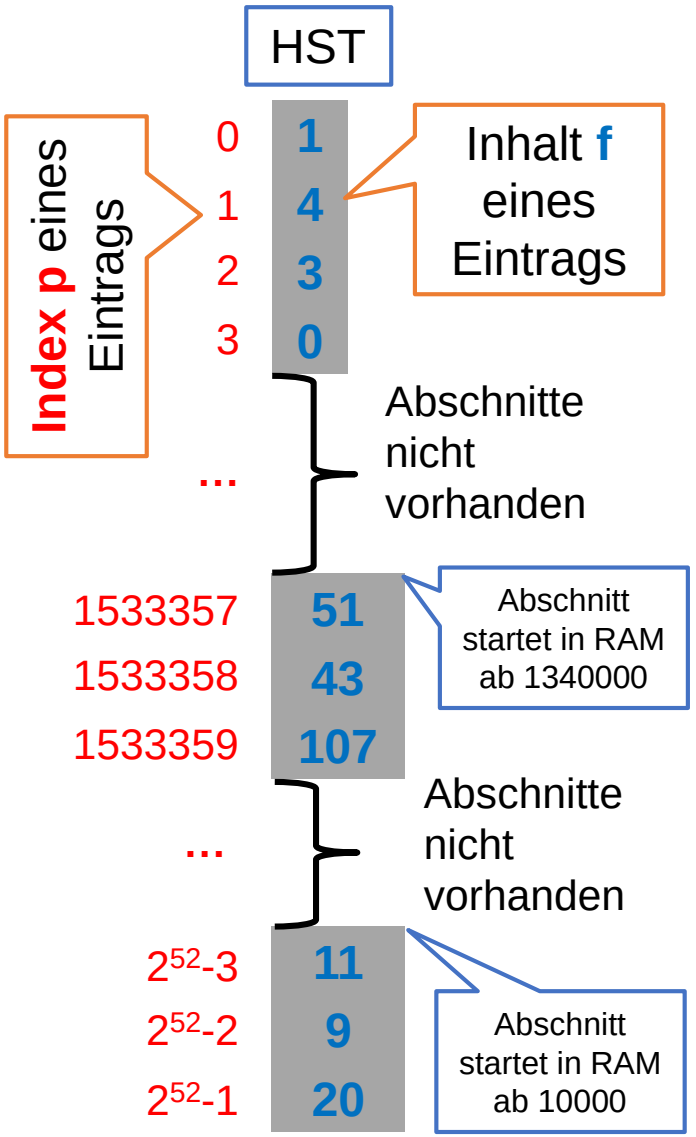
- Wir suchen in der invertierten Seitentabelle nach (**PID**, **p**)
- Index (Position) **f** des passenden Eintrags ist genau die Rahmennummer **f**

Invertierte Seitentabellen - Beispiel

- Die Zeit für die Adressenübersetzung steigt
  - Hier wird wieder mit TLB gearbeitet
  - Zusätzlich wird eine Hashtabelle benutzt, um bei einem TLB-Fehler den Eintrag in Zeit *O*(1) zu finden
  - Sonst lineare Suchzeit - im schlimmsten Falle durch alle Tabelleneinträge!
- Wie groß ist nun die Seitentabelle - in % des Gesamtspeichers?
  - Annahmen: PID hat 12 Bits, ein Rahmen (== Seite) hat 4kByte und wir haben 64 Bit-Adressen
  - Dann ist d = 12 Bits, p hat (64-12)=52 Bits; wir brauchen also 64 Bits (8 Bytes) pro Eintrag, welcher eine Seite mit 4 kByte abdeckt => 8/4096 = 1/512 ~ 0.2%

Paging – Mehrstufige Seitentabelle

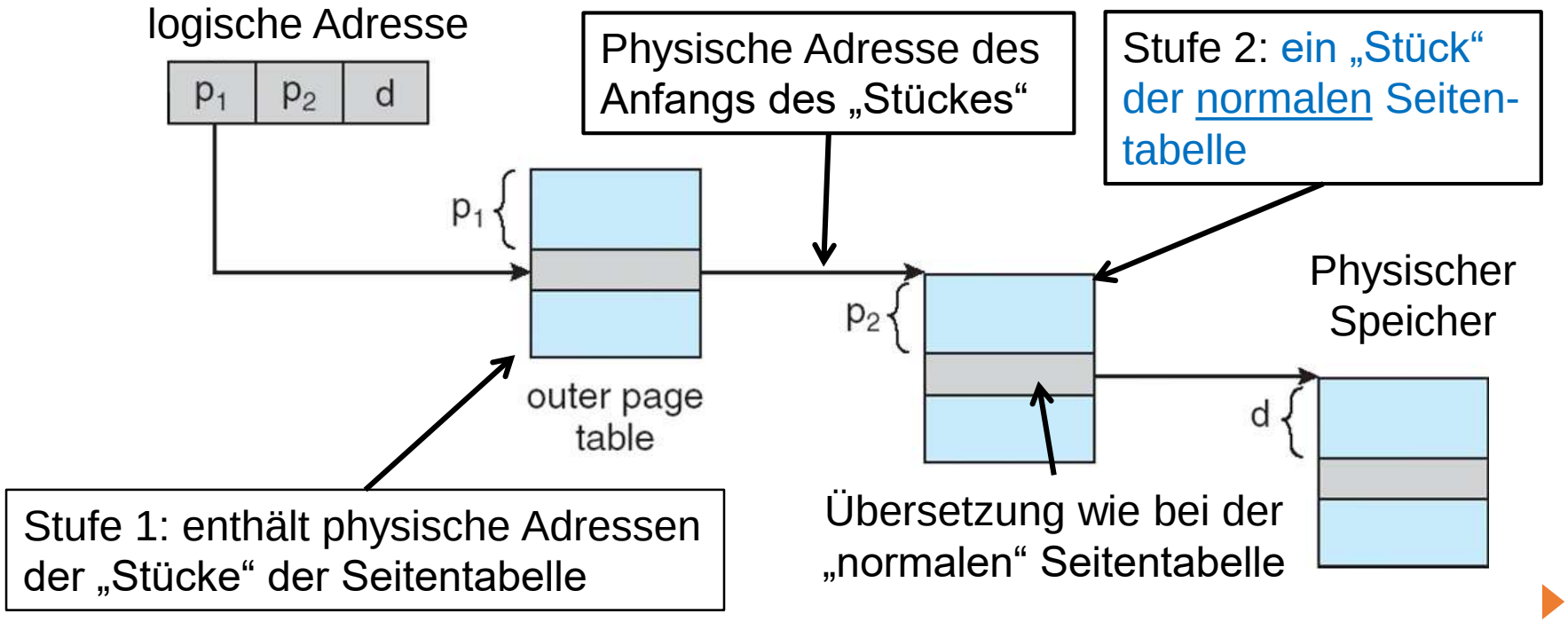
Mehrstufige/Hierarchische Seitentabellen



- Eine hierarchische Tabelle HST ist zunächst wie eine direkte Seitentabelle (nicht invertiert)
- Aber sie ist zerteilt in Abschnitte mit je 2^k Einträgen (z.B. k=9), und nur manche dieser Abschnitte existieren
- Jeder Abschnitt kann „irgendwo“ im phys. Speicher anfangen
  - Es werden nur solche Abschnitte angelegt, die die tatsächlich verwendeten logischen Adressenbereiche abdecken

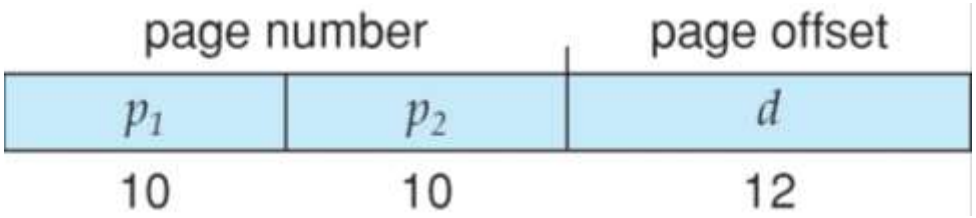
Mehrstufige/Hierarchische Seitentabellen

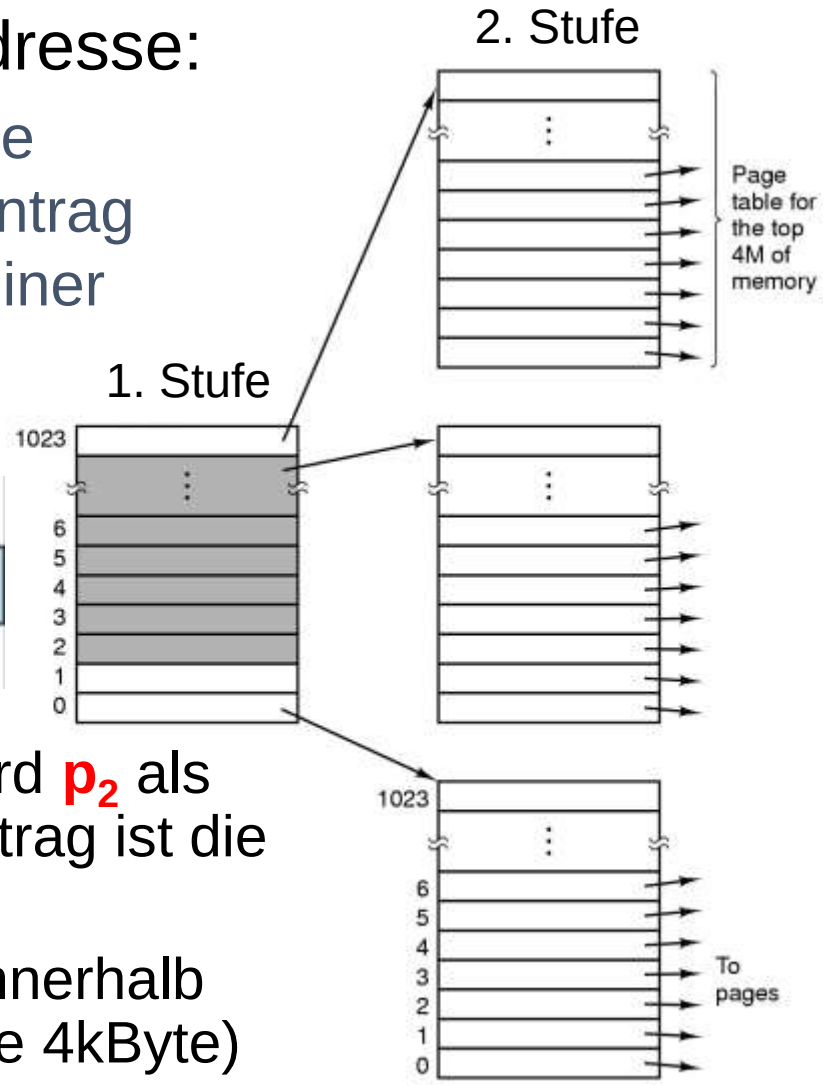
- Wir müssen wissen, wo im Speicher welche Stücke (der Seitentabelle) liegen
- Diese Anfangsadressen werden in einer Tabelle höherer Stufe (d.h. Stufe 1) gehalten („page directory“)



Mehrstufige Seitentabellen - Beispiel

- Umrechnung der logischen Adresse:
  - **p_1** wird als Index in die 1. Tabelle genutzt; der von P_1 indizierte Eintrag enthält die physische Adresse einer Tabelle der 2. Stufe

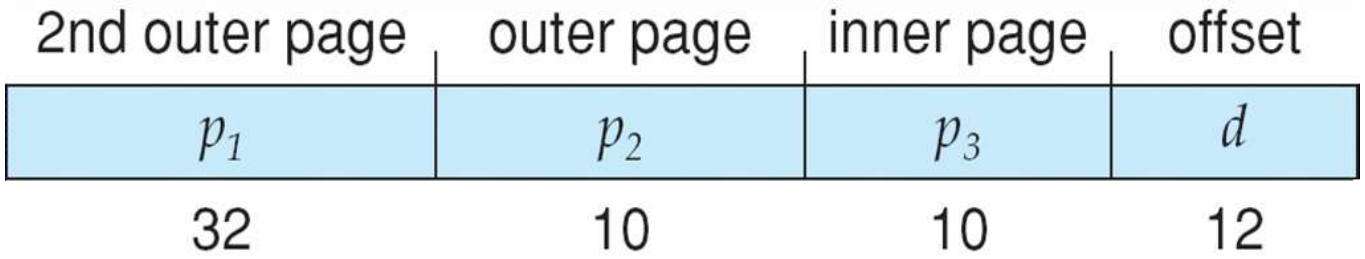




- Innerhalb der Tabelle der 2. Stufe wird **p_2** als Index verwendet; der gefundene Eintrag ist die Basisadresse eines Seitenrahmens
- Offset **d** (12 Bits) legt die Adresse innerhalb des Seitenrahmens fest (Seitengröße 4kByte)

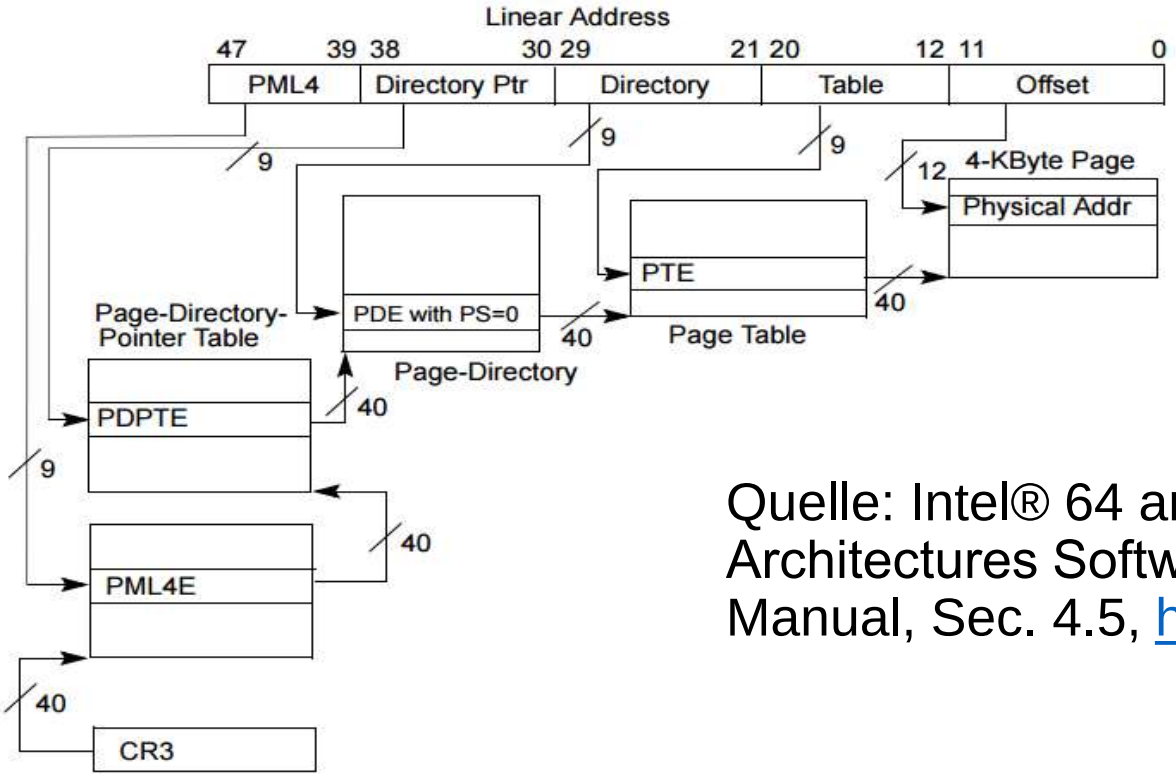
Mehrstufige Seitentabellen - Probleme

- Mehrstufige Tabellen erhöhen die Zugriffszeit
  - Aber mit TLB kann man dies kompensieren
- Wirkliches Problem entsteht bei 64-Bit Adressräumen
  - Bei 3 Stufigen Seitentabelle hätte die äußerste Tabelle (1. Stufe) immer noch 2^{32} Einträge => 2^{35} Bytes = 32 GB!



Mehrstufige Seitentabellen bei 64-Bit System

- Intel 64 CPUs in IA-32e-Modus haben 48-Bit log. Adressraum (256 TBytes), der zu 52-Bit physischen Addressraum (4 PBytes) übersetzt wird
- Es gibt 4 Stufen, je 9 Bits: PML4E, PDPTE, PDE, PTE



Quelle: Intel® 64 and IA-32 Architectures Software Developer's Manual, Sec. 4.5, http://goo.gl/ahtAvR

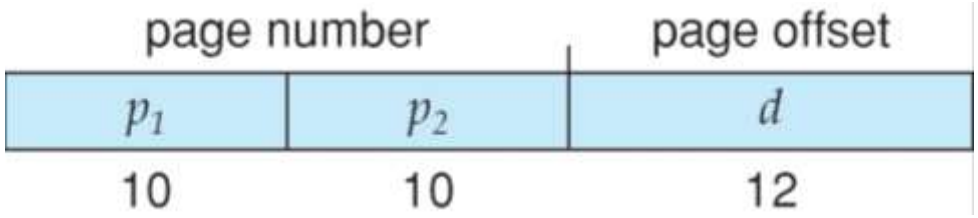
Video

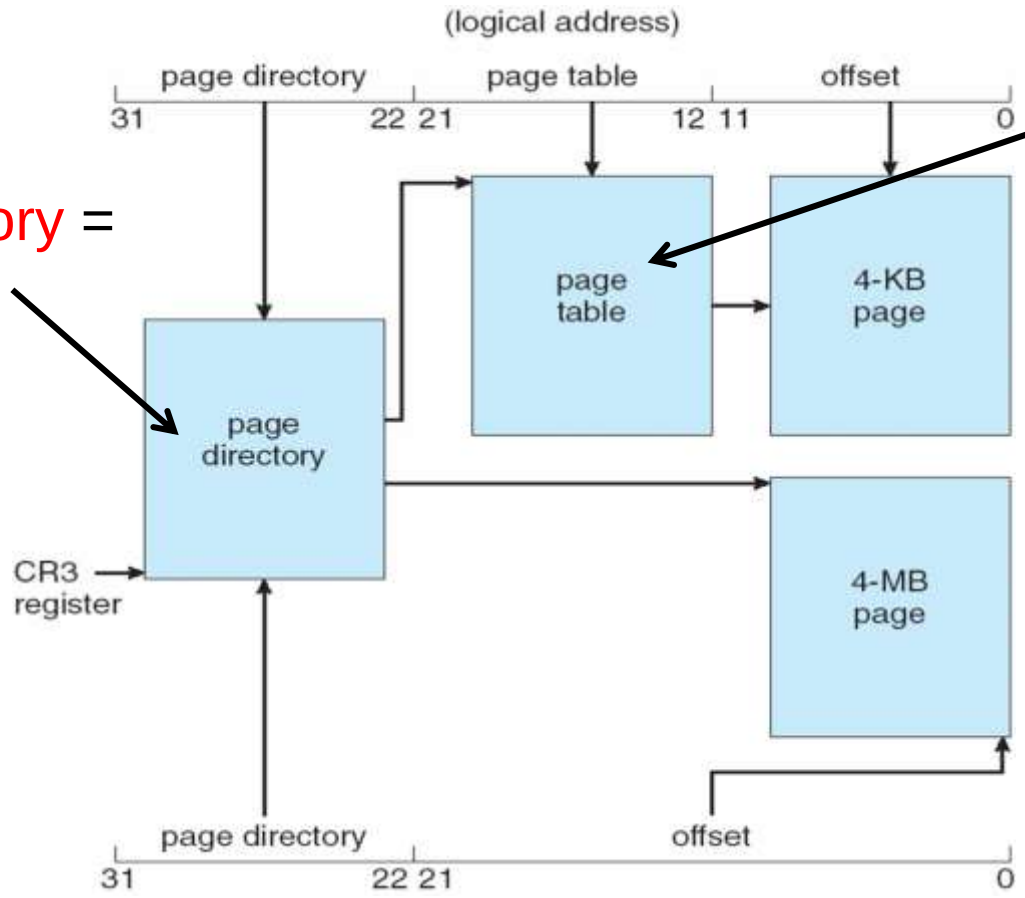
- Modern Page Tables: Multi-Level Paging
  - https://www.youtube.com/watch?v=pCgw4Pe-5jo
  - Von 0:00 bis 05:05 - Seitentabellen in IA 32 und Problem
  - Von 05:05 bis ca. 07:54: mögliche Lösungen
  - Von 07:54 bis ca. 08:58 - mehrstufige Seitentabellen
  - Von 14:00 bis Ende - IA32/x86 oder ARM?

Paging bei IA32-Architektur: Konkrete Details

Empfehlenswert zum Lesen:
https://wiki.osdev.org/Paging

Mehrstufige Seitentabellen bei IA-32 (Pentium)





Page Directory = Tabelle der 1. Stufe

Page Table = Tabelle der 2. Stufe (Stück der „normalen“ Seitentabelle)

Bei 4-MB ist Page Directory die tatsächliche Seitentabelle, da unser Offset 22 Bits hat => nur 2^{10} Einträge in der Seitentabelle

Konfigurieren der Seitentabellen bei IA32

- Die MMU wird durch spezielle CPU-Register bei jedem Prozesswechsel neu konfiguriert

- Register **page-table base register** (**PTBR**) zeigt auf den Anfang der Seitentabelle
  - Wird bei einem Prozesswechsel aktualisiert
  - Bei x86-CPUs ist das der **SP3-Systemregister**

- Register **page-table length register** (**PRLR**) beschreibt die Länge der Seitentabelle
  - Dient dem Schutz vor Zugriffen jenseits des Tabellenendes

Setzen und Einschalten der Seitentabelle

- 1. Lade in das Register **CR3** die Adresse des page directory (= oberste Stufe der IA32-Seitentabelle)
  - **mov** eax, page_directory
  - **mov** **cr3**, eax

- 2. Setze die entsprechenden Flags des Kontrollregisters **CR0**: Paging Bit (PG) auf 1
  - **mov** eax, **cr0**
  - **or** eax, 0x80000001
  - **mov** **cr0**, eax

Quelle: https://wiki.osdev.org/Paging#Enabling

Was enthält ein Tabelleneintrag (IA32)? /1

- G [global]: if set, prevents the TLB from updating the address in its cache if CR3 is reset
- D [dirty]: if set, then the page has been written to; once set will not unset itself
- A [accessed]: used to discover whether a page has been read or written to; if it has, then the bit is set
- C [cache]: the 'Cache Disable' bit; if the bit is set, the page will not be cached

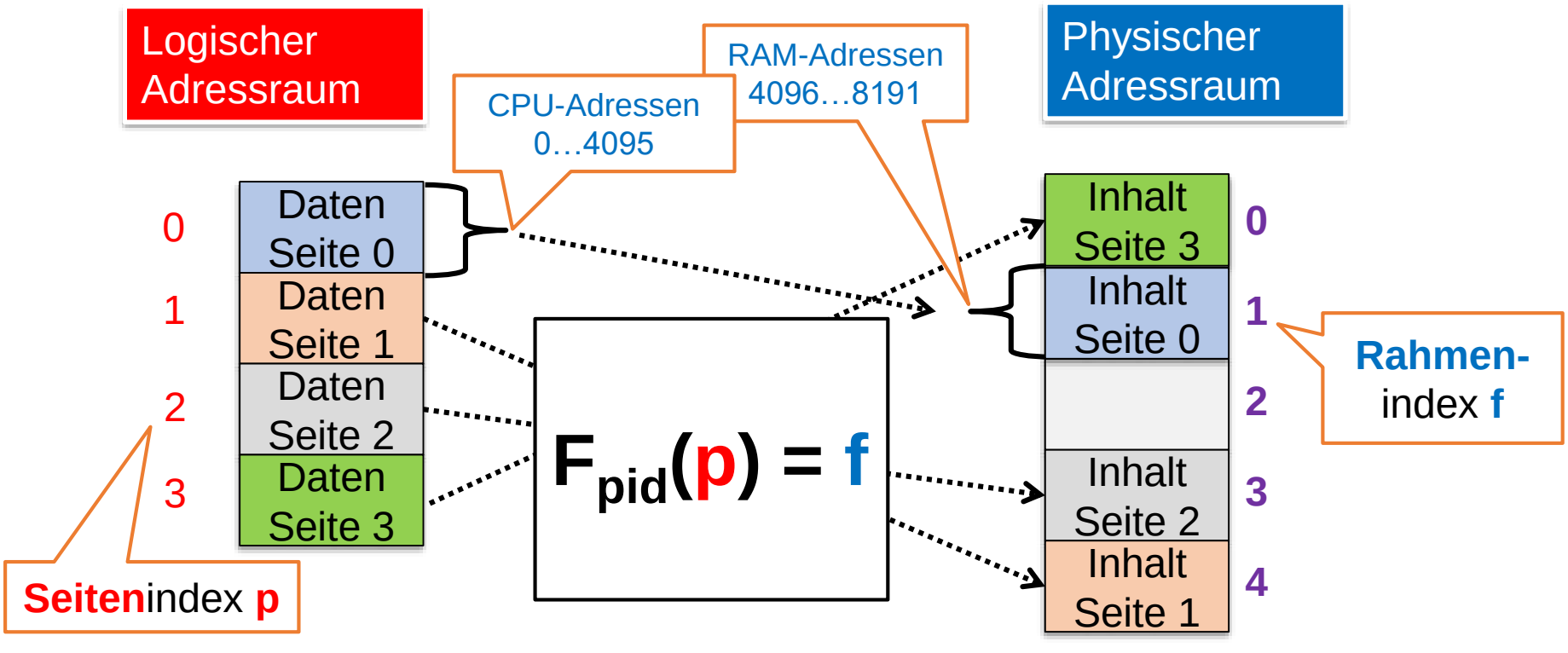
Was enthält ein Tabelleneintrag (IA32)? /2

- W [Write-Through]: if the bit is set, write-through caching is enabled
- U [User/Supervisor]: controls access to the page based on privilege level; if set, then the page may be accessed by all, otherwise only the supervisor can access it
- R [Read/Write]: if the bit is set, the page is read/write, otherwise the page is read-only
- P [Present]: If the bit is set, the page is actually in physical memory at the moment

Quelle: https://wiki.osdev.org/Paging#Page_Table

Zusammenfassung: Paging und Seitentabellen

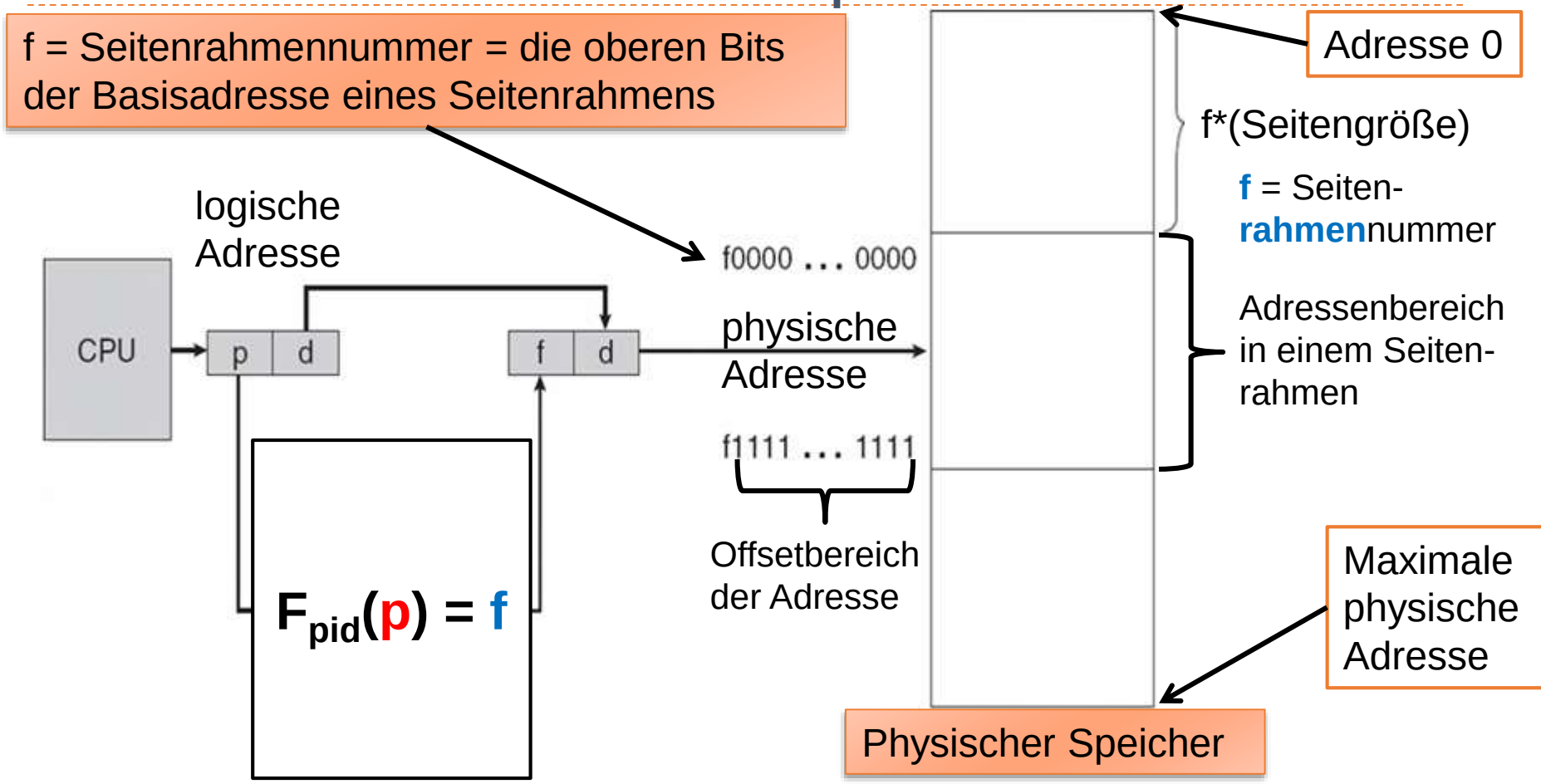
Paging: Adressenübersetzung in „Kacheln“



- **Paging** übersetzt “Kacheln” der logischen Adressen auf “Kacheln” der physischen Adressen
- Der Kern ist eine effiziente Funktion $F_{pid}(p) = f$, die Seitenindex **p** auf Rahmenindex **f** abbildet

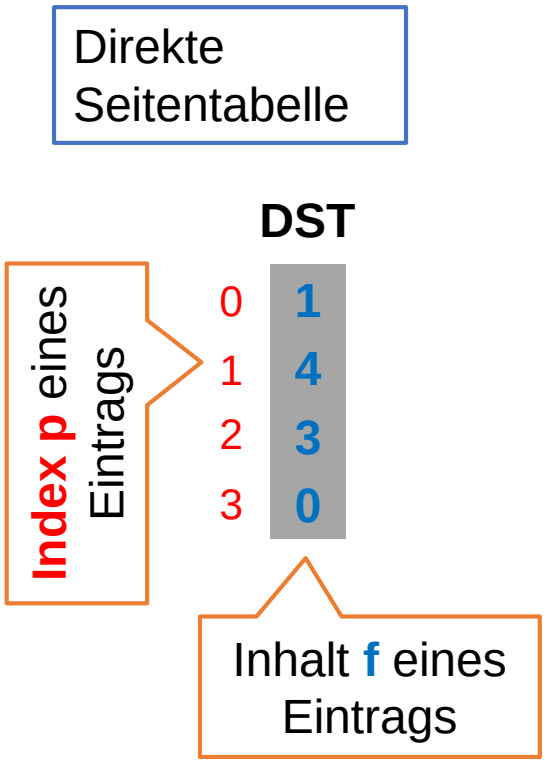
pid = Prozess ID des aktuellen Prozesses

Adressübersetzung mit $F_{pid}(p) = f$



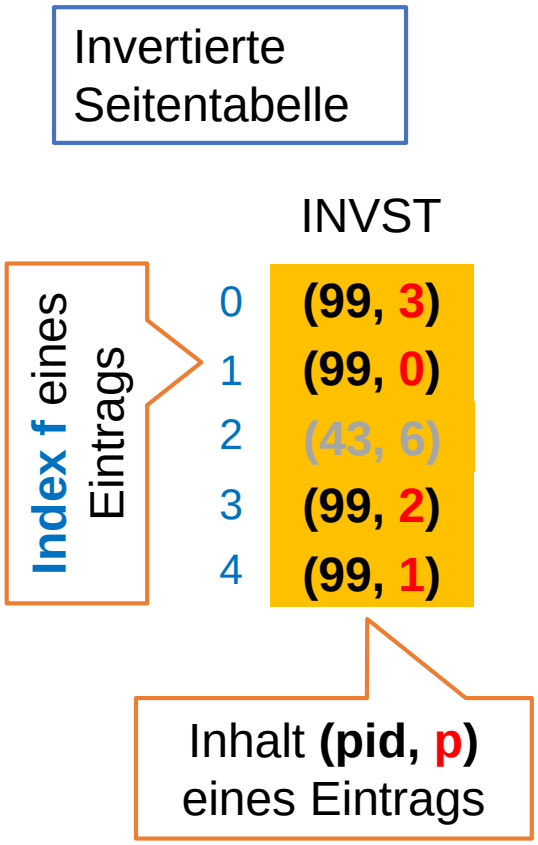
- Nachdem $F_{pid}(p) = f$ berechnet ist, ist die Übersetzung sehr einfach: obere Bits (= Wert **p**) einer logischen Adresse werden durch Bits mit Wert **f** ersetzt

Direkte (d.h. “Normale”) Seitentabellen



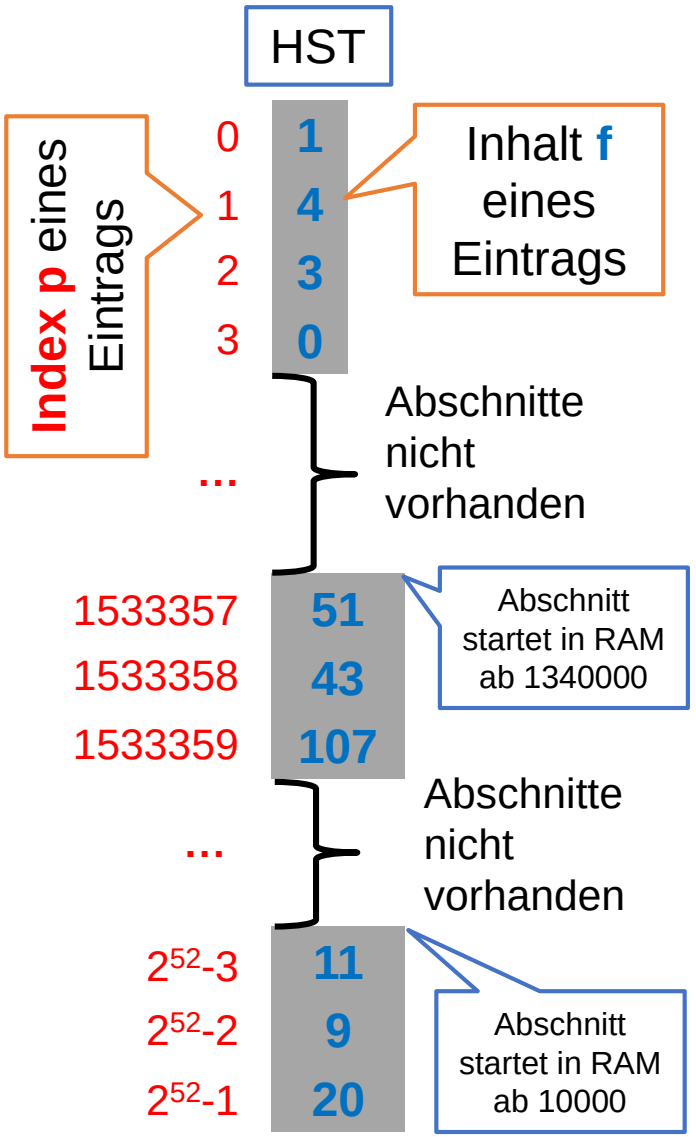
- Wie berechnet man $F_{pid}()$ mit einer direkten Seitentabelle?
- $F_{pid}()$ ist implementiert als ein Tabellen-**look-up**: sehr schnell
- In **DST**, Eintrag mit Index **p** enthält Wert **f** mit: $\mathbf{F_{pid}(p) = f}$
  - Index **p** = **Seiten**nummer
  - Inhalt **DST[p]** = **Rahmen**nummer
- Index **p** wird in DST nicht gespeichert, das wäre redundant
- Jeder Prozess (identifiziert via pid) braucht eine eigene Tabelle

Invertierte Seitentabellen



- In **INVST,** Eintrag mit Index **f** enthält Wert **(**pid,**p)** mit: $\mathbf{F_{pid}(p) = f}$
  - Index **f** = **Rahmen**nummer
  - Inhalt **INVST[f]** = (Prozess-ID **pid**, **Seiten**nummer **p**)
- D.h. um $F_{pid}()$ zu berechnen, muss man suchen: ggf. aufwändig
- Es gibt eine einzige Tabelle für alle Prozesse

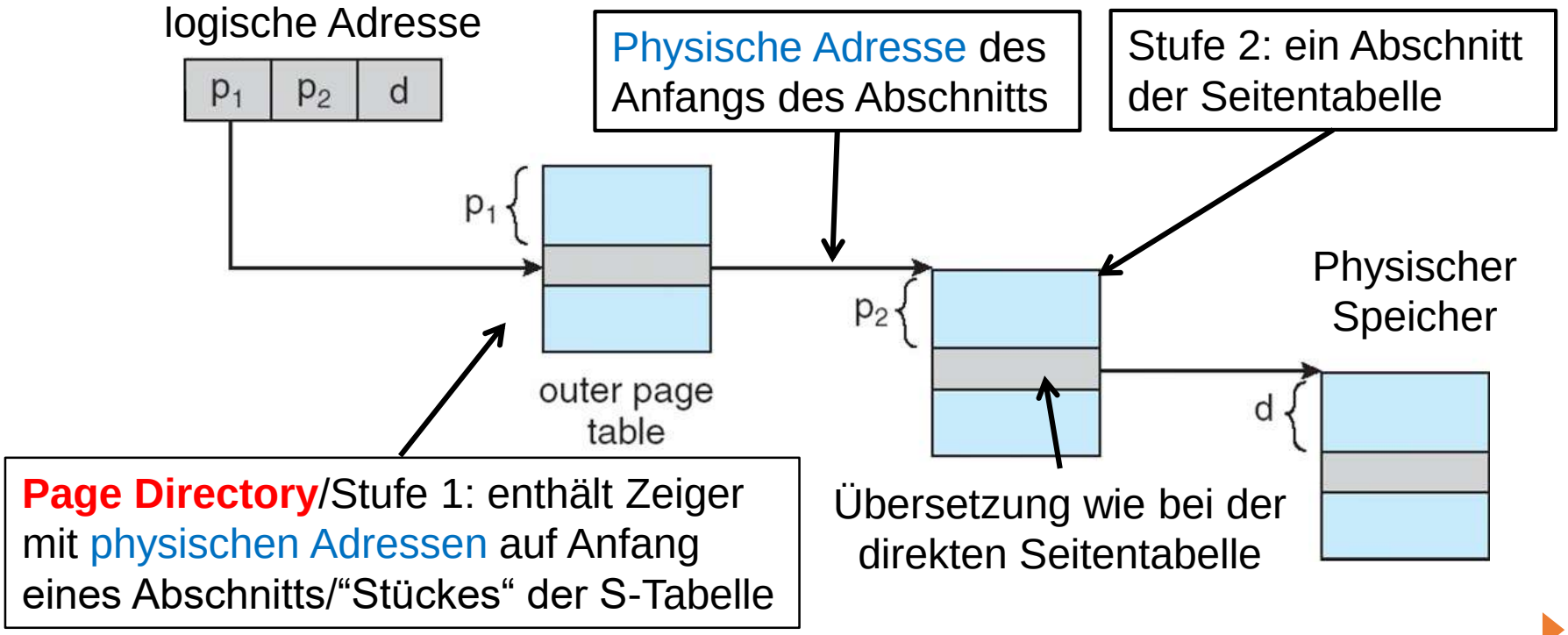
Hierarchische Seitentabellen



- Eine hierarchische Tabelle **HST** benutzt look-up (wie eine direkte Tabelle), d.h.: $\mathbf{F_{pid}(p) = HST_{pid}[p]}$
- Aber HST ist zerteilt in Abschnitte mit je 2^k Einträgen (z.B. k=10), und nur manche dieser Abschnitte existieren
- Jeder Abschnitt kann „irgendwo“ im Speicher anfangen
  - Es werden nur solche Abschnitte angelegt, die die tatsächlich verwendeten logischen Adressenbereiche abdecken

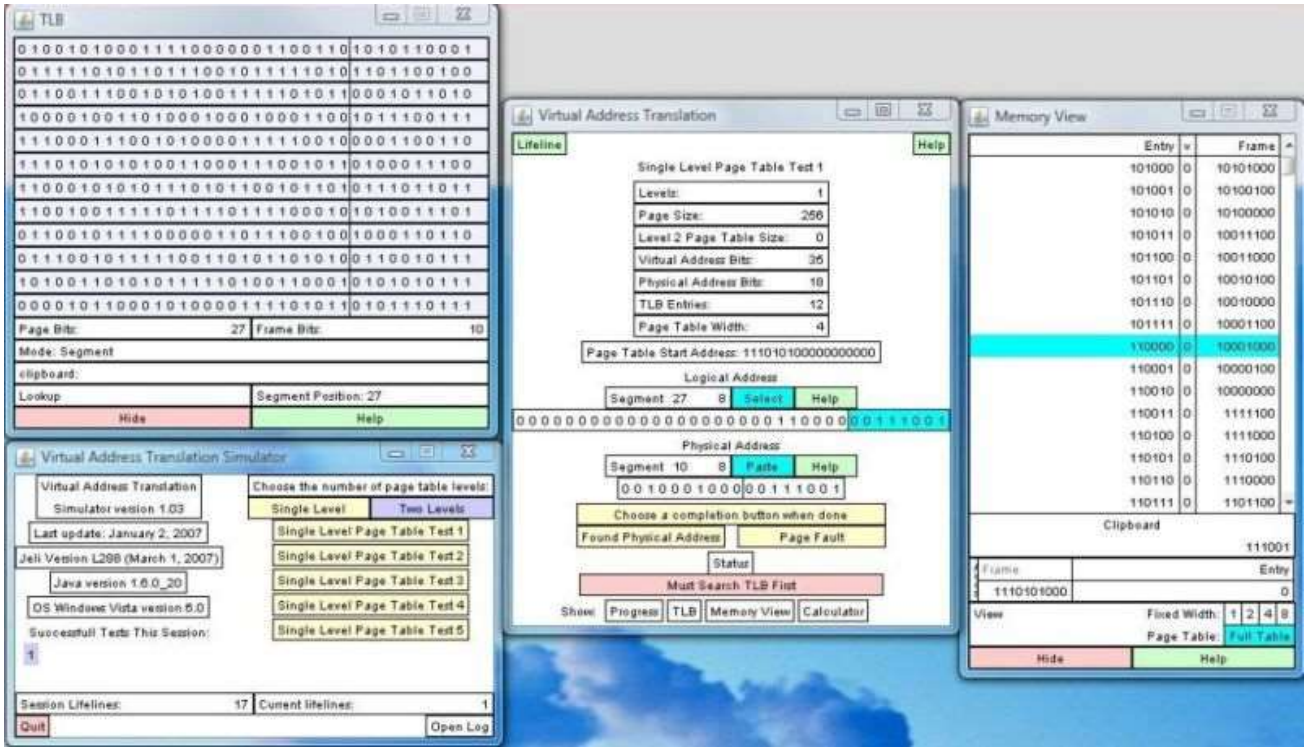
Hierarchische Seitentabellen bei IA32

- Oberste Bits (hier p_1) einer logischen Adresse indexieren die **Page Directory**, die Zeiger auf die Anfänge der einzelner Abschnitte enthält
  - Bei IA32e gibt es 3 Levels dieser Directories



Address Translation Simulator

- Simulator für Seitentabelle mit 1 oder 2 Stufen
- Download (address.zip)+ Doku:
  - https://heibox.uni-heidelberg.de/d/3f13bbd5d1/
- Originalquelle (Link kaputt):
  - http://vip.cs.utsa.edu/simulators/



Bitzählung: Nützliche Zusammenhänge /1

- Die Adressübersetzung funktioniert durch das Ersetzen (in der Adresse) der Seitennummer **p** durch eine (eigentlich beliebige) Seiten**rahmen**nummer **f**
- Die Bits des Seitenoffsets **d** werden direkt übernommen

- Sei **S** die Seitengröße (in Bytes), **N** die Busbreite (in Bits, z.B. N=32), **B**(..) die Anzahlen der Bits von p, f, d
- Was sind die Zusammenhänge zwischen S, N, B(p), B(d), B(f) usw.?

Bitzählung: Nützliche Zusammenhänge /2

- Da d die Offset-Adresse innerhalb einer Seite ist, muss $2^{B(d)}$ (=Anzahl dieser Adressen) mindestens S sein
  - => Annahme: **$S = 2^{B(d)}$**, bzw. B(d) = Zweierlogarithmus(S)
- Wie groß sind B(p) (und damit auch B(f))?
  - In einer N-Bit Adresse haben wir (neben B(d) Bits für Offset) noch genau N-B(d) Bits übrig, also **B(p) = N-B(f)**
- Wie viele Einträge hat eine vollständige direkte Seitentabelle?
  - Genau **$2^{B(p)}$** = Maximale Anzahl der Seitennummern

- [Wie viele Einträge hat eine invertierte Seitentabelle?
  - Maximalanzahl der Rahmen = **(Größe des RAMs) / S**]

Bitzählung: Beispiele

- Sei N=32 (z.B. IA32) und Seitengröße S = 4096 Bytes
  - Was sind B(d), B(p)=B(f) und die Anzahl der Einträge der direkten Seitentabelle?
  - Es ist 2^{12} = S, also B(d) = 12 (Bits)
  - Damit ist B(p) = 32 – 12 = 20 (Bits)
  - Anzahl der Einträge in Seitentabelle = $2^{B(p)} = 2^{20}$
- Sei N=64, Seitengröße=1024 Bytes, RAM-Größe=1GB und eine invertierte Seitentabelle
  - Was ist B(d), B(p), die Anzahl der Einträge der Seitentab.?
  - => B(d) = $\log_2(1024)$ = 10, B(p) = 64-10 = 54
  - Wir haben als Anzahl der Rahmen: 1 GB / 1 kB = 2^{20}, also genauso viele Einträge der Seitentabelle

Zusammenfassung

- Paging - Geschwindigkeit der Übersetung
  - Translation Look-Aside Buffer (TLB) für schnelleren Zugriff
- Reduktion der Größe der Seitentabellen
  - Invertierte Seitentabelle
  - Hierarchisches Paging
- Quellen: Silberschatz Kap. 8+9; Tanenbaum Kap. 3