Systeme I: Betriebssysteme

Wiederholung wichtiger Inhalte der Kapitel 3-7

Maren Bennewitz

Version 12.02.2014

Hinweis

- Diese Folien haben keinen Anspruch auf Vollständigkeit, was den Inhalt der Prüfung angeht
- Sie dienen nur dem Zweck der (kurzen) Wiederholung wichtiger Inhalte, die Sie gelernt haben sollten
- Insbesondere wird selbstverständlich auch der Inhalt von Kapitel 8 in der Prüfung abgefragt

Dateisysteme

Dateisysteme

- Dateisystem = Betriebssystemteil, der sich mit Dateien befasst
- Externe Sicht: Vom Dateisystem angebotene Funktionalität
- Interne Sicht: Implementierung von Dateisystemen

Dateiattribute

- Informationen über eine Datei, die das Betriebssystem speichert
- Beispiele:
  - Entstehungszeitpunkt (Datum, Uhrzeit)
  - Dateigröße
  - Zugriffsrechte

Zugriffsrechte

- Sicht des Benutzers (Bsp. Unix / Linux):

  drwxr-xr-x 6 maren users 238 Oct 18 12:13 systeme-slides
  drwxr-x–- 1 maren users 204 Oct 2 11:10 systeme-uebungen
  -rw------- 1 maren users 29696 Feb 29 2012 zeitplan.xls

- Bedeutung der Felder:
  - Typ des Eintrags: Datei, Verzeichnis
  - Rechte: Besitzer, Gruppenbesitzer, alle anderen
  - Anzahl Einträge (Verzeichnis)/Linkzähler (Datei)
  - Besitzer, Gruppenbesitzer
  - Speicherplatzverbrauch
  - Datum der letzten Modifikation
  - Name

Sonderrechte Unix (1)

SUID (set user ID):

- Erweitertes Zugriffsrecht für Dateien
- Unprivilegierte Benutzer erlangen kontrollierten Zugriff auf privilegierte Ressourcen
- Ausführung mit den Rechten des Besitzers der Datei (anstatt mit den Rechten des ausführenden Benutzers)
- Optische Notation bei ls:

  -rwsr-x---

Sonderrechte Unix (2)

SGID (set group ID)

- Ausführung mit den Rechten der Gruppe, der die Datei/das Verzeichnis gehört (anstatt mit den Rechten der Gruppe, die ausführt)
- Verzeichnisse: Neu angelegte Dateien gehören der Gruppe, der auch das Verzeichnis gehört (anstatt der Gruppe, die eine Datei erzeugt)
- Optische Notation bei ls:

  drwxrws---

Sequentieller Dateizugriff

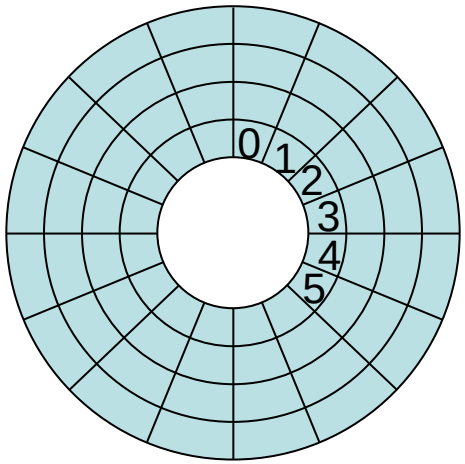
- Alle Bytes können nur nacheinander vom Datenspeicher gelesen werden
- Kein Überspringen möglich
- Um auf einen bestimmten Datensatz zugreifen zu können, müssen alle Datensätze zwischen Start- und Zielposition besucht werden
- Die Zugriffszeit ist von der Entfernung der Datensätze abhängig

Wahlfreier Dateizugriff

- Zugriff auf ein beliebiges Element in konstanter Zeit
- Bytes/Datensätze können in beliebiger Reihenfolge ausgelesen werden
- Durch Hardware realisiert

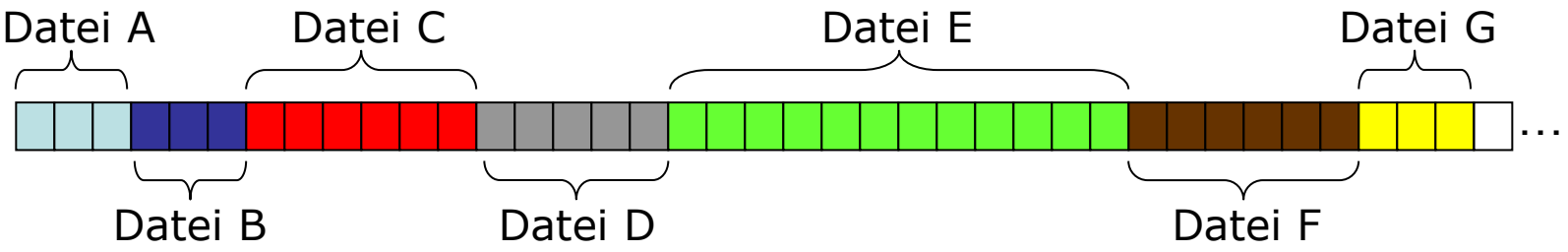
Implementierung von Dateisystem

- Festplatten sind eingeteilt in Blöcke, die durchnummeriert sind



Zusammenhängende Belegung

- Abspeicherung von Dateien durch zusammenhängende Menge von Plattenblöcken



- Vorteil: Lesegeschwindigkeit (eine einzige Suchoperation für gesamte Datei), effizienter wahlfreier Zugriff
- Nachteil: Externe Fragmentierung

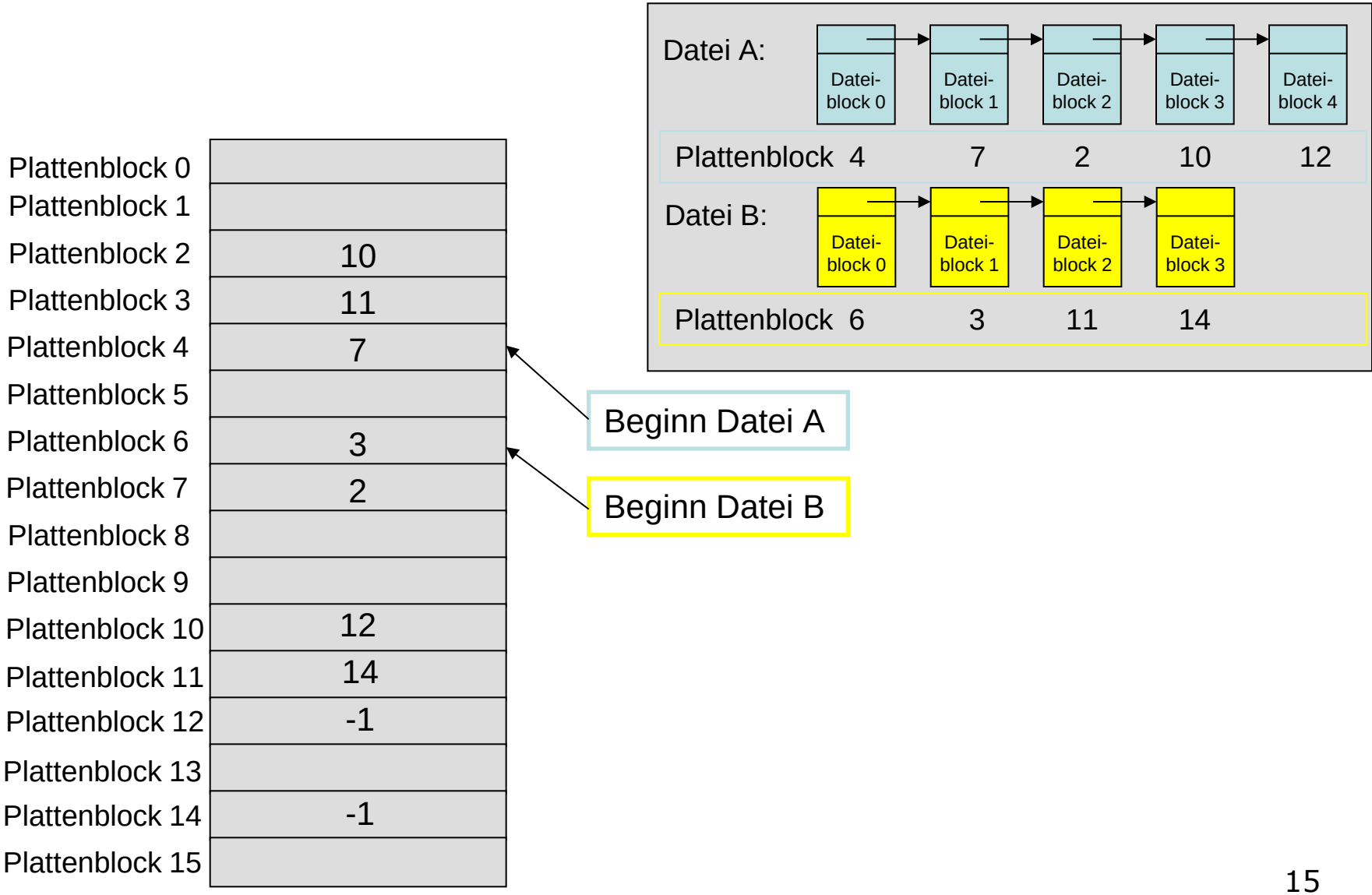
Verkettete Listen

- Dateien gespeichert als verkettete Listen von Plattenblöcken: Zeiger auf nächsten Block
- Vorteil: Fragmentierung der Festplatte führt nicht zu Verlust von Speicherplatz
- Nachteil: Bei Zugriff auf Dateiblock n sind n-1 Lesezugriffe auf die Platte nötig, um den Block zu lokalisieren

FAT-System

- Datei-Allokationstabelle (FAT=File Allocation Table)
- Information über Verkettung der Blöcke im Hauptspeicher
- z.B. FAT-16: 16 Bit zur Adressierung
- Bei wahlfreiem Zugriff auf Block n muss nur die Kette von Verweisen im Hauptspeicher verfolgt werden (nicht auf der Platte)
- Nachteil: Anzahl der Einträge ist immer gleich der Gesamtzahl der Plattenblöcke

FAT Beispiel



Beispiel: FAT-32

- 28 Bit zur Adressierung:
  2^{28} verschiedene Zeiger, je 4 Byte groß
- Größe der FAT: $2^{28} * 4$ Byte = 2^{30}
  = 1 GiB!

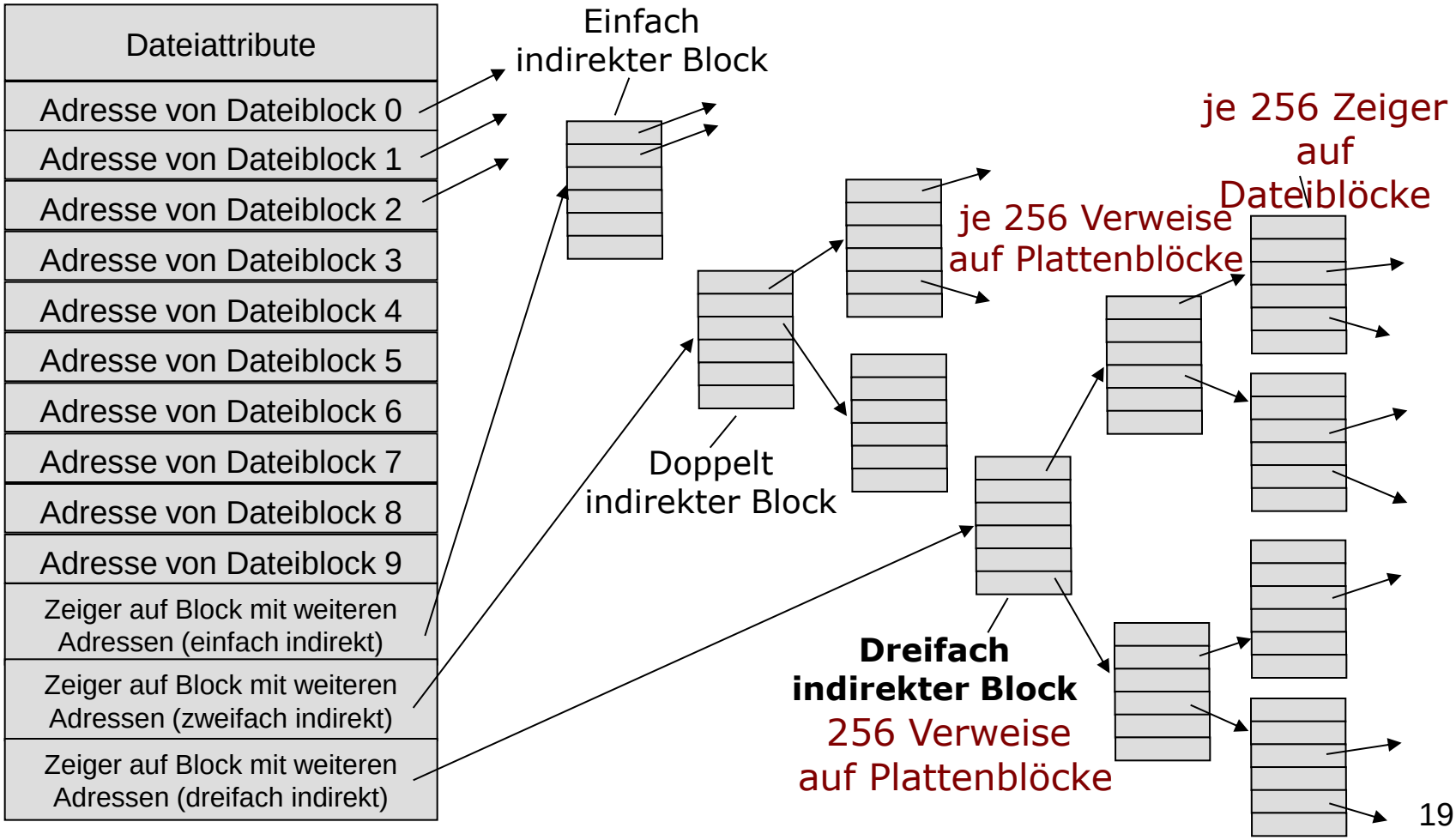
FAT-System

- Kleinere Blöcke führen zu weniger verschwendetem Platz pro Datei (interne Fragmentierung)
- Je kleiner die Blockgröße, desto mehr Zeiger, desto größer die FAT im Hauptspeicher
- Maximale Größe des ganzen Dateisystems wird durch Blockgröße begrenzt
- FAT-16 muss z.B. für eine 2 GiB Partition eine Blockgröße von 32 KiB verwenden
- Andernfalls kann mit den 2^{16} verschiedenen Zeigern nicht die ganze Partition adressiert werden

Realisierung von Dateien
I-Nodes (1)

- I-Node enthält: Metadaten und Adressen von Plattenblöcken
- I-Node ermöglicht Zugriff auf alle Blöcke der Datei
- I-Node muss nur dann im Speicher sein, wenn eine Datei offen ist
- Die Größe eines I-Nodes ist fix und relativ klein
- Auf kleinere Dateien kann direkt zugegriffen werden

Realisierung von Dateien
I-Nodes (2)



Beispiel: Maximale Dateigröße

- Blockgröße 1 KiB, Zeigergröße 4 Bytes
- 10 direkte Zeiger des I-Nodes: 10 KiB Daten lassen sich speichern
- Einfach indirekter Zeiger verweist auf einen Plattenblock, der maximal 1 KiB/4 Byte Zeiger verwalten kann, also 256 Zeiger
- Indirekt: 1 KiB * 256 = 256 KiB Daten
- Zweifach indirekter Zeiger:
  1 KiB * 256 * 256 = 64 MiB Daten
- Dreifache indirekter Zeiger:
  1 KiB * 256 * 256 * 256 = 16 GiB Daten

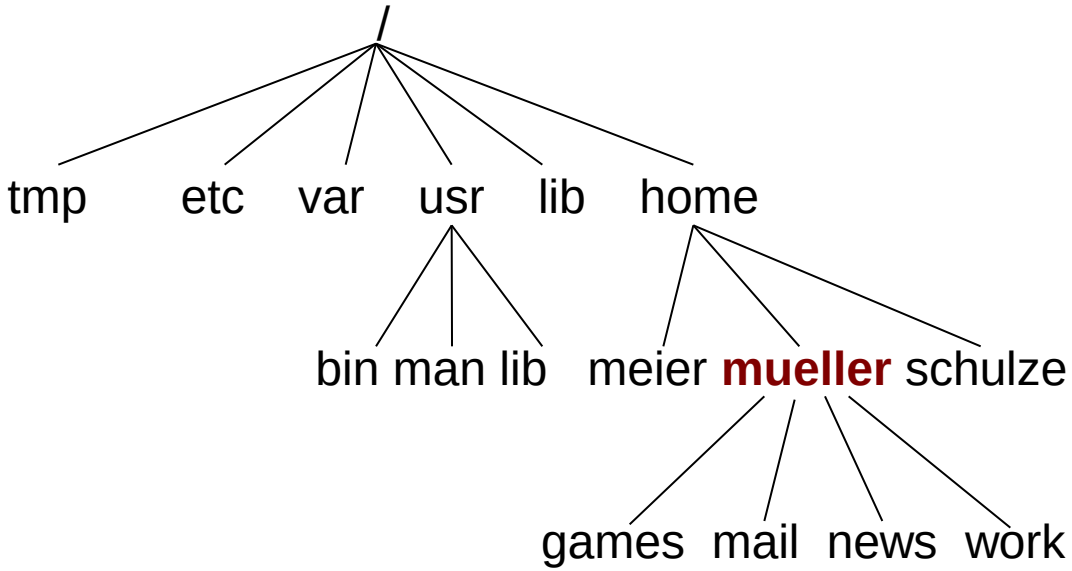
I-Node-Tabelle

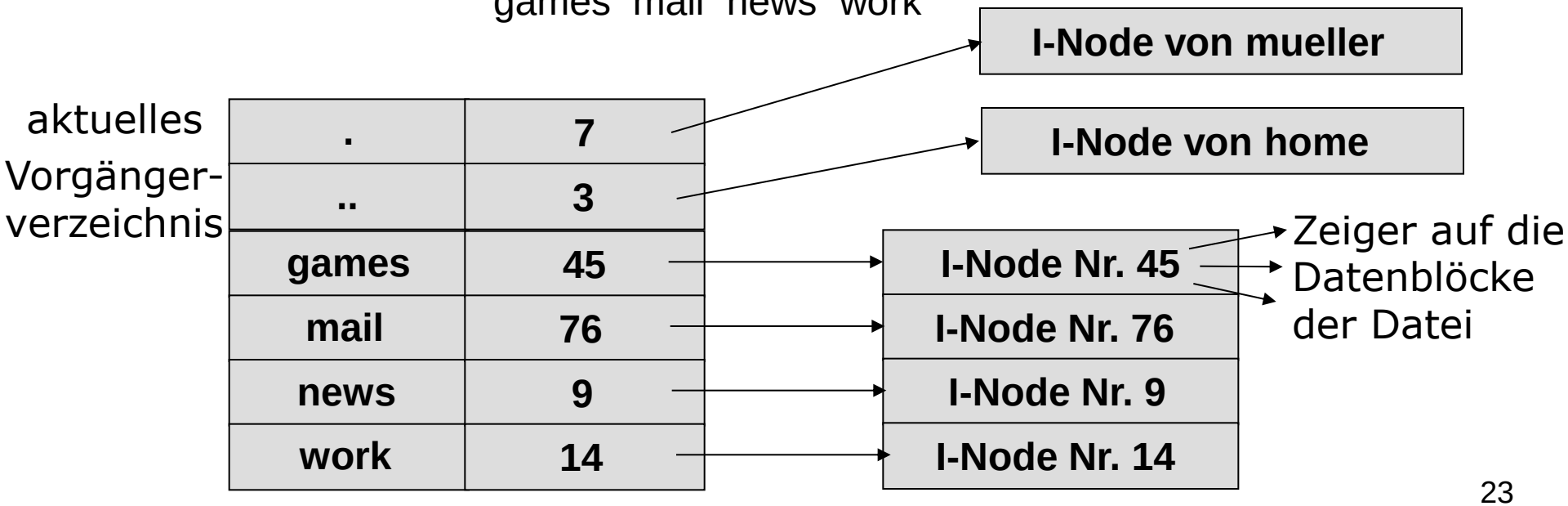
- I-Node-Tabelle auf Festplatte: Enthält alle I-Nodes mit Speicheradresse
- Größe der I-Node-Tabelle wird beim Anlegen des Dateisystems festgelegt
- Es muss eine ausreichende Anzahl von I-Nodes eingeplant werden: Jeder Datei ist ein eindeutiger I-Node zugeordnet
- Wenn verfügbarer Plattenplatz oder alle I–Nodes belegt sind: Dateisystem voll
- Es muss eine ausreichende Anzahl von I-Nodes eingeplant werden

Realisierung von Verzeichnissen (1)

- Verzeichnisse sind ebenfalls Dateien
- Verzeichnis liefert eine Abbildung von Datei- bzw. Verzeichnisnamen auf I-Node-Nummern
- Jeder Verzeichniseintrag ist ein Paar aus Name und I-Node-Nummer
- Über die I-Nodes kommt man dann zu Dateiinhalten

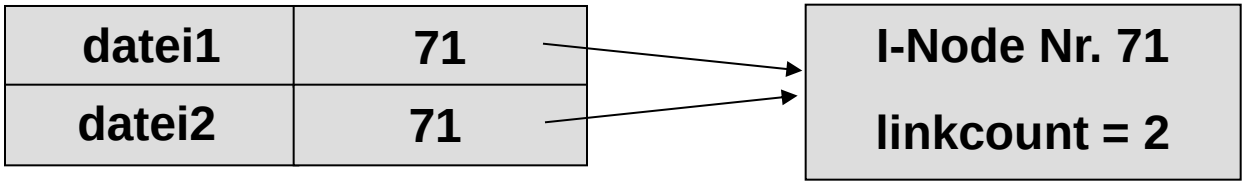
Realisierung von Verzeichnissen (2)





Implementierung von Hardlinks

```
$ ls –l
-rw-r----- 1 meier users … datei1
$ ln datei1 datei2
$ ls –l
-rw-r----- 2 meier users … datei1
-rw-r------ 2 meier users … datei2
```



Implementierung symbolischer Links

$ ls –l

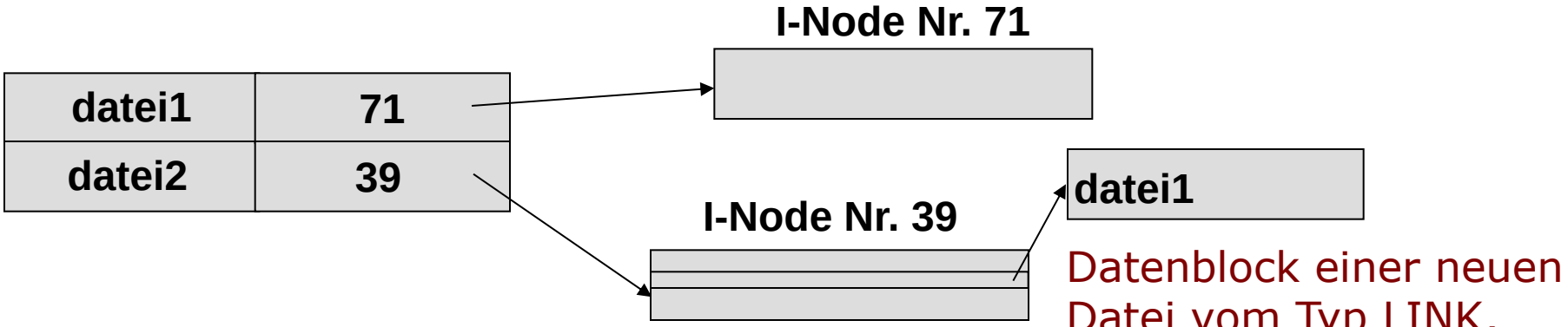
-rw-r----- **1** meier users … datei1

$ ln –s datei1 datei2

$ ls –l

-rw-r----- **1** meier users ... datei1

lrwxrwxrwx **1** meier users ... datei2 -> datei1



Prozesse

„Programm in Ausführung“

- Prozess = Instanz eines Programms mit
  - Aktuellem Wert vom Befehlszähler
  - Registerinhalten
  - Belegung von Variablen
- Multitasking-Betriebssysteme: Mehrere Prozesse können „pseudo-parallel“ ausgeführt werden

Motivation Multitasking

- „Gleichzeitiges“ Ausführen von Aufgaben
- Mehrere Benutzer teilen sich einen Rechner
- Wartezeiten auf Ein-/Ausgaben kann durch Abarbeitung anderer Prozesse genutzt werden

Prozesswechsel

- Nicht-präemptive Betriebssysteme: Prozessen kann nur dann der Prozessor entzogen werden, wenn sie ihn selbst abgeben
- Präemptive Betriebssysteme: Aktiver Prozess kann unterbrochen werden vom Betriebssystem
- Mehr Verwaltung, aber bessere Auslastung der CPU bei präemptiven Systemen

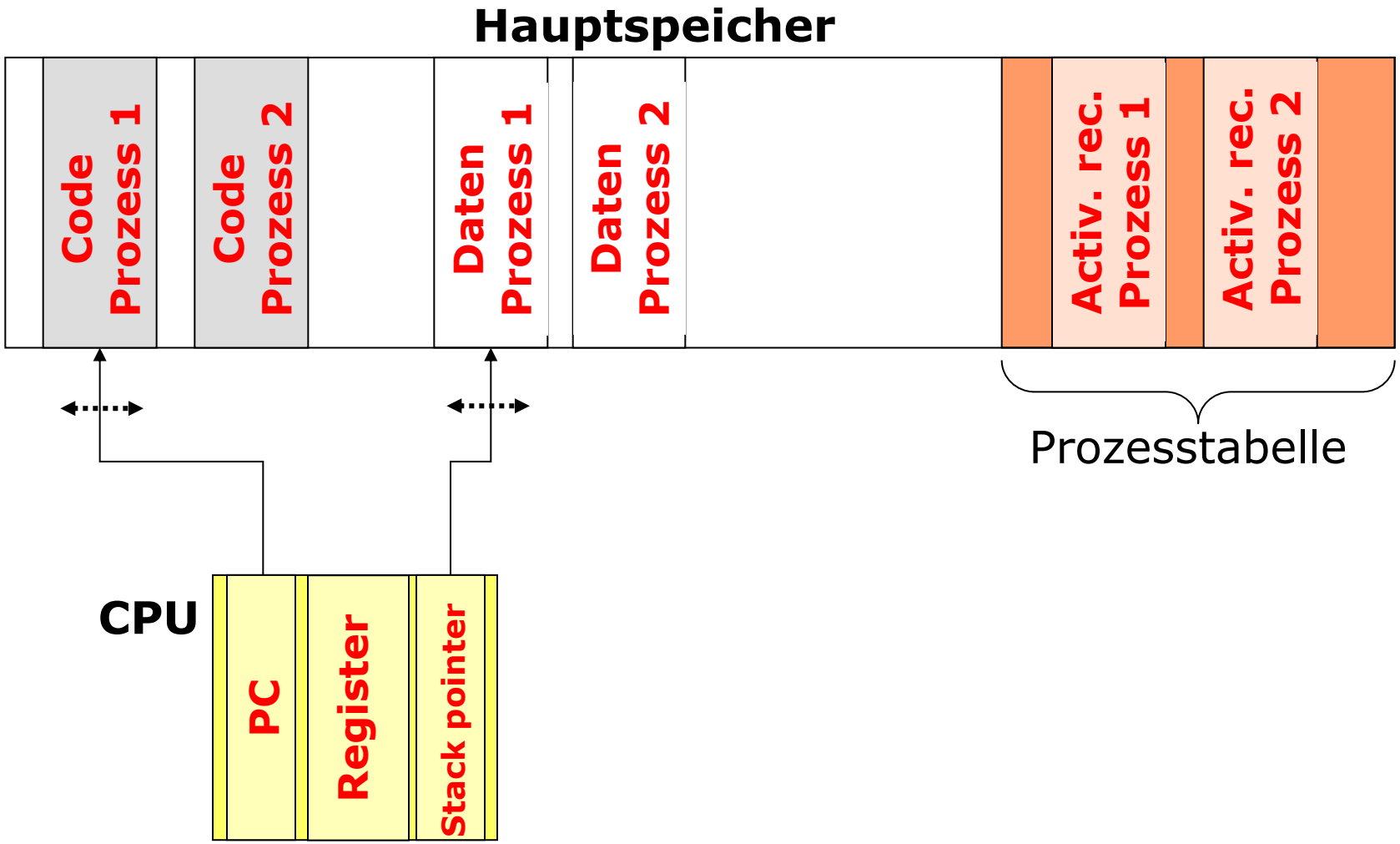
Adressraum

- Jeder Prozess hat einen Adressraum im Hauptspeicher
- Speicherzellen, in denen der Prozess lesen und schreiben darf
- Adressraum enthält
  - Ausführbares Programm
  - Programmdaten
  - Kellerspeicher (“Stack”, für lokale Variablen)

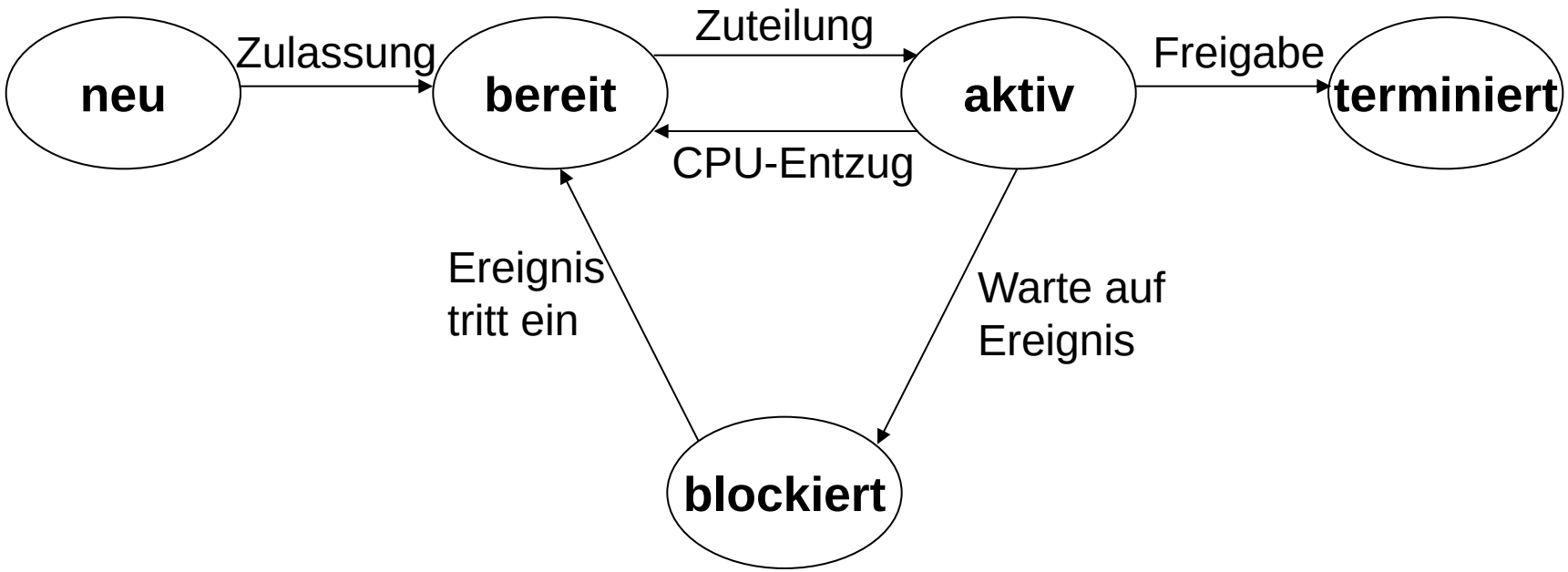
Prozessinformationen

- Prozesstabelle enthält individuelle Informationen
  - Befehlszähler
  - Allgemeine Register
  - Stack pointer (zeigt auf oberstes Kellerelement)
- Wird bei Bedarf in Prozessorregister geladen

Beispiel: Prozesswechsel



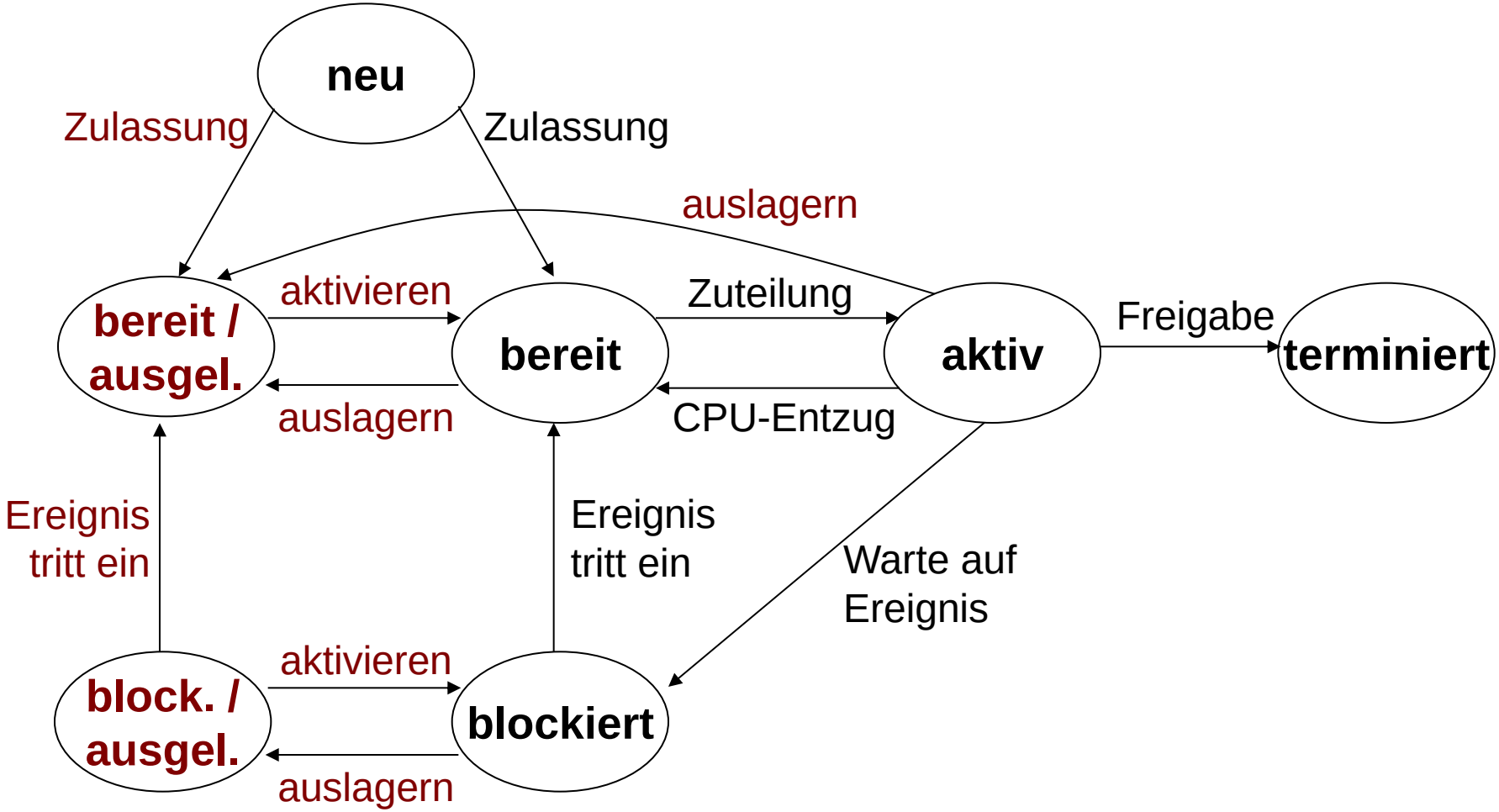
Prozessmodell mit 5 Zuständen



Swapping (Auslagern)

- Prozessen werden komplett auf die Festplatte ausgelagert
- Motivation:
  - Platz schaffen
  - Prozessor kann sich trotz Multitasking die meiste Zeit im Leerlauf befinden

Prozessmodell mit 7 Zuständen



Nebenläufigkeit und wechselseitiger Ausschluss

Anforderungen an Lösungen für das Problem der krit. Region

- Keine zwei Prozesse dürfen gleichzeitig in ihren kritischen Regionen sein (wechselseitiger Ausschluss)
- Es dürfen keine Annahmen über Geschwindigkeit und Anzahl der Rechenkerne gemacht werden
- Kein Prozess, der außerhalb seiner kritischen Regionen läuft, darf andere Prozesse blockieren
- Kein Prozess sollte ewig darauf warten müssen, in seine kritische Region einzutreten

Lösungen für wechselseitigen Ausschluss

- Software-Lösungen: Verantwortlichkeit bei Prozessen, Anwendungsprogramme gezwungen sich zu koordinieren
- Hardware-Unterstützung: Spezielle Maschinenbefehle, reduzieren Verwaltungsaufwand
- Ins Betriebssystem integrierte Lösungen

Wechselseitiger Ausschluss in Software

- Wechselseitiger Ausschluss ist in Software schwer zu realisieren
- Alles was einfacher ist als Petersons Algorithmus ist höchstwahrscheinlich falsch
- Formale Beweise sind unabdingbar!
- Software-Lösungen für wechselseitigen Ausschluss benötigen aktives Warten

Wechselseitiger Ausschluss in Hardware (1)

- Atomare Operationen (Hardware garantiert atomare Ausführung)
- Testen und Setzen zusammen bilden eine atomare Operation:
  - Befehl TSL (Test and Set Lock)
  - Prozesswechsel kann nicht zwischen Testen und Setzen erfolgen

Wechselseitiger Ausschluss in Hardware (2)

- Vorteil: Wechselseitiger Ausschluss ist garantiert
- Nachteil: Aktives Warten wie bei Software-Lösungen

Wechselseitiger Ausschluss, ins Betriebssystem integriert

- Prozesse blockieren statt warten
- Systemaufruf sleep zum Blockieren von Prozessen
- Systemaufruf wakeup nach Verlassen des kritischen Abschnitts

Mutexe (1)

- Mutex m besteht aus einer binären Lock-Variable $lock_m$ und einer Warteschlange $queue_m$
- Vor Eintritt in die kritische Region: Aufruf von mutex_lock(m)
- Darin: Überprüfung, ob die kritische Region schon belegt ist
  - Falls ja: Prozess blockiert (sleep) und wird in Warteschlange eingefügt
  - Falls nein: Lock-Variable wird gesetzt und Prozess darf in kritische Region eintreten

Mutexe (2)

- Zwei Zustände: gesperrt / nicht gesperrt
- Kein aktives Warten, CPU wird abgegeben
- Der Prozess, der den Mutex gesperrt hat, gibt ihn auch wieder frei

Semaphore

- Wert des Semaphors repräsentiert die Anzahl der Weckrufe, die ausstehen
- Drei mögliche Situationen für den Wert eines Semaphors:
  - Wert < 0: Weckrufe stehen schon aus, nächster Prozess legt sich auch schlafen
  - Wert > 0: frei, nächster Prozess darf fortfahren
  - Wert = 0: keine Weckrufe bisher gespeichert, nächster Prozess legt sich schlafen

Semaphor: down/up Operationen

- Initialisiere Zähler des Semaphors: $count_s = m$
- down-Operation:
  - Verringere den Wert von $count_s$ um 1
  - Wenn $count_s < 0$, blockiere den aufrufenden Prozess, sonst fahre fort
- up-Operation:
  - Erhöhe den Wert von $count_s$ um 1
  - Wenn $count_s <= 0$, wecke einen der blockierten Prozesse auf

Deadlocks

Deadlocks

- Ressourcen, die zu einem Zeitpunkt jeweils nur ein Prozess benutzen kann
- Ressourcen werden nach und nach von den Prozessen angefordert
- Dabei kann es zu einem Deadlock kommen

Voraussetzungen für Ressourcen-Deadlocks

- Wechselseitiger Ausschluss:
  Jede Ressource ist entweder verfügbar oder genau einem Prozess zugeordnet

Voraussetzungen für Ressourcen-Deadlocks

- Wechselseitiger Ausschluss
- Besitzen und Warten:
  Prozesse, die schon Ressourcen reserviert haben, können noch weitere Ressourcen anfordern

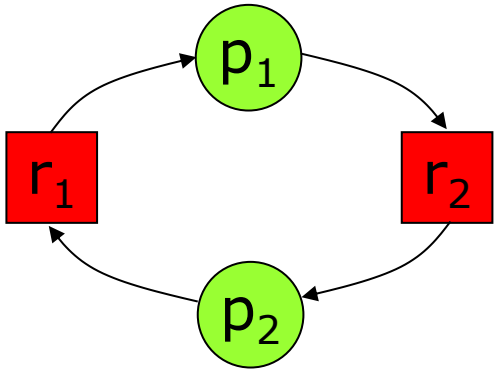
Voraussetzungen für Ressourcen-Deadlocks

- Wechselseitiger Ausschluss
- Besitzen und Warten
- Kein Ressourcenentzug:
  Ressourcen, die einem Prozess bewilligt wurden, können nicht gewaltsam wieder entzogen werden

Voraussetzungen für Ressourcen-Deadlocks

- Wechselseitiger Ausschluss
- Besitzen und Warten
- Kein Ressourcenentzug
- Zyklisches Warten:
  Es gibt eine zyklische Kette von Prozessen, von denen jeder auf eine Ressource wartet, die dem nächsten Prozess in der Kette gehört

Belegungs-Anforderungs-Graph

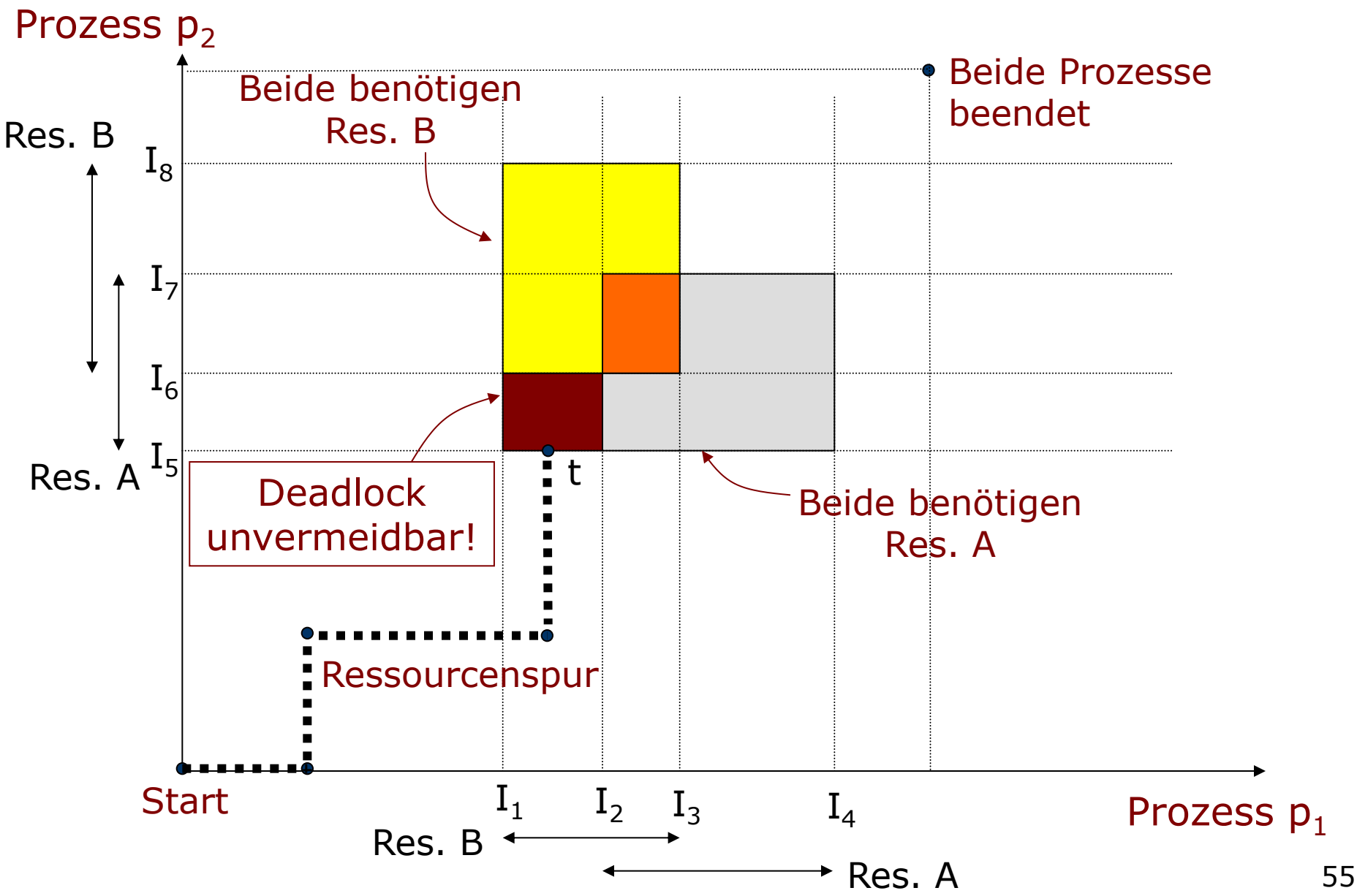


- Zyklen im Belegungs-Anforderungsgraphen repräsentieren Deadlocks!

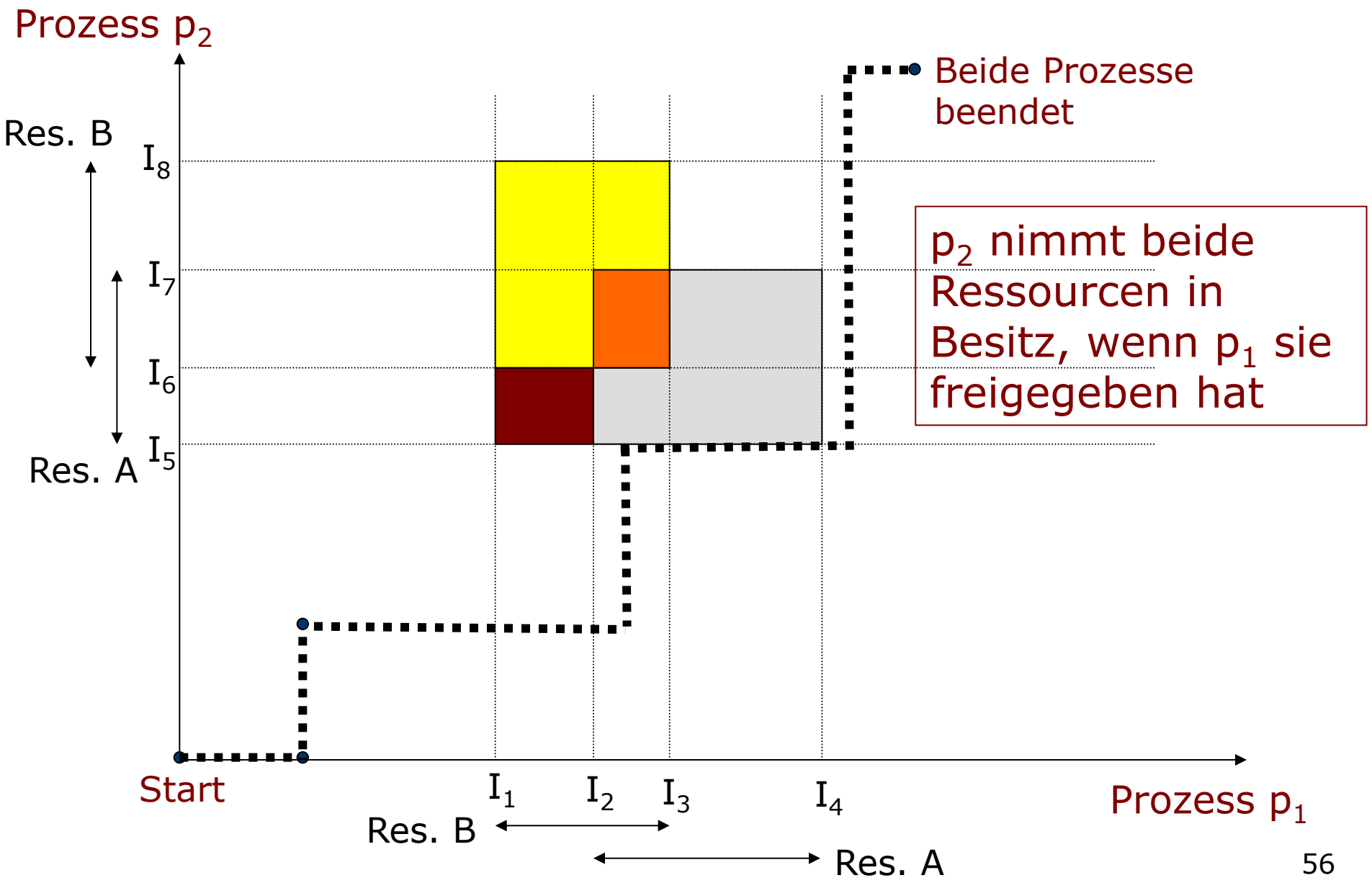
Ressourcendiagramm

- Diagramm zur Visualisierung der Ressourcenanforderungen über die Zeit
- Dient zur Erkennung von potentiellen Deadlocks
- Zeitachsen: Prozessfortschritt
- Ressourcenspur: Eine mögliche Ausführungsreihenfolge der Anweisungen

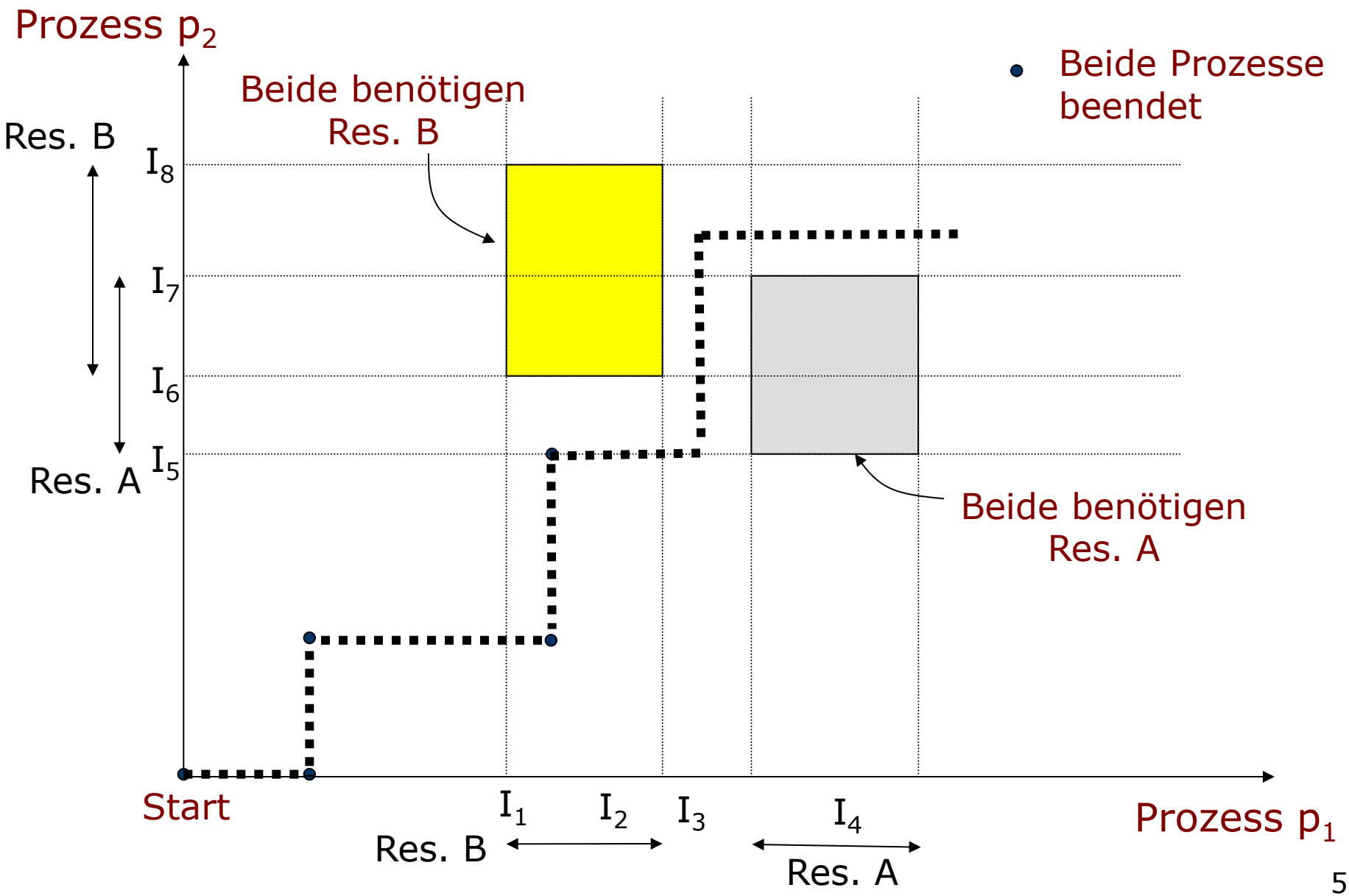
Ressourcendiagramm: Deadlock



Beispiel: Ohne Deadlock



Beispiel: Nie Deadlock



Verhindern von Deadlocks

- Bankier-Algorithmus: Ressourcenaufteilung an Prozesse, so dass Deadlocks garantiert verhindert werden
- Voraussetzungen:
  - Es ist im Voraus bekannt, welche und wie viele Ressourcen die einzelnen Prozesse maximal anfordern werden
  - Anforderung übersteigt für keinen Prozess die zur Verfügung stehenden Ressourcen

Bankier-Algorithmus (1)

Ein Zustand ist sicher, wenn

- Es auf jeden Fall eine deadlockfreie „Restausführung“ aller Prozesse gibt
- Unabhängig davon, wann die Prozesse in Zukunft ihre Ressourcenanforderungen und -freigaben durchführen
- Auch dann wenn gilt
  - Prozesse stellen ihre Restanforderungen jeweils auf einen Schlag
  - Freigaben erst bei Prozessbeendigung (Worst Case)

Bankier-Algorithmus (2)

- Wenn deadlockfreie Restausführung nicht garantiert werden kann: Zustand ist unsicher
- Beachte: Ein unsicherer Zustand muss nicht notwendigerweise zu einem Deadlock führen! (Wir betrachten den Worst Case)
- Bei sicheren Zuständen kann garantiert werden, dass es eine Reihenfolge gibt, s.d. alle Prozesse zu Ende laufen können
- Bei unsicheren gibt es keine Garantie

Bankier-Algorithmus (3)

- Strategie: Überführe das System immer nur in sichere Zustände
- Bei jeder Ressourcenanforderung eines Prozesses: Prüfe, ob das System bei Erfüllung der Anforderung in einen sicheren Zustand kommt
- Falls nein: Erfülle Anforderung nicht, stelle den Prozess zurück und mache mit einem anderen Prozess weiter; sonst erfülle sie

Bankier-Algorithmus (4)

Probleme bei der praktischen Anwendung:

- Prozesse können meist nicht im Voraus eine verlässliche Obergrenze für ihre Ressourcenanforderungen geben
- Anzahl der Prozesse ist nicht fest, sondern ändert sich ständig (z.B. durch Ein- / Ausloggen)

In der Praxis verzichtet man auf absolute Garantien für Deadlock-Freiheit

Scheduling

Kurzfristiges Scheduling

- Scheduling: Betriebssystem muss entscheiden, welche Prozesse Rechenzeit beanspruchen dürfen
- Kurzfristiges Scheduling: Zuweisung der CPU an konkurrierende Prozesse
- Verschiedene Scheduling-Algorithmen existieren für verschiedene Optimierungsziele

Scheduling-Algorithmen

- First Come First Served: Prozess, der bereits am längsten wartet
- Shortest Job First: Prozess mit kürzester erwarteter Rechenzeit
- Shortest Remaining Time: Prozess mit kürzester geschätzter Restlaufzeit
- Highest Response Ratio Next: Prozess mit größter normalisierter Durchlaufzeit

Feedback Queues

- Präemptiv (Zeitintervall), dynamische Prioritäten
- Bei Abgabe der CPU: Einreihen in Warteschlange mit der nächst geringeren Priorität
- Dadurch: Verbrauchte CPU-Zeit wird angenähert durch Anzahl erzwungener CPU-Abgaben
- Abarbeitung der Warteschlangen nach Priorität

Scheduling bei UNIX

- Es gibt Warteschlangen mit unterschiedlichen Prioritäten
- Anfangs: Erster Prozess der nichtleeren Warteschlange mit höchster Priorität ausgeführt
- Anschließend: Prozesse höchster Priorität werden untereinander nach Round Robin gescheduled
- Neuberechnung der Prioritäten in regelmäßigen Zeitabständen, Wartezeit geht in die Priorität ein

Klausur

- Datum: 14. März 2013, 9:00
- Bitte UniCard mitbringen
- Räume: Gebäude 101, Räume 00-026, 00-036 und 00-010/14
- Klausureinsicht: Wird auf Webpage bekannt gegeben
- Viel Glück!