Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Kap. 3: Prozesse und Threads

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Konzept des sequentiellen Prozesses:

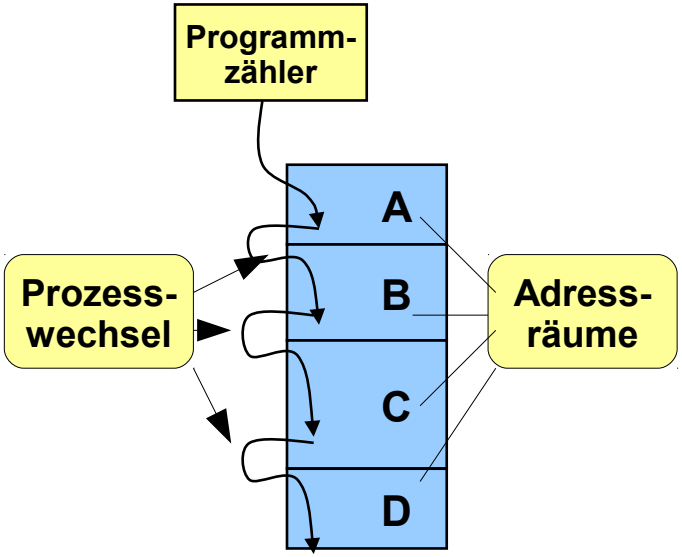
Def

- Def: Ein Prozess ist ein sich in Ausführung befindliches Programm (einschl. seiner aktuellen Werte des Programmzählers, der Register, Speichervariablen, Stack).
- Ein Prozess besitzt einen privaten Adressraum
  - Menge von (virtuellen) Adressen, von Prozess zugreifbar
  - Programm und Daten in Adressraum sichtbar.
- Verhältnis Prozess - Prozessor
  - Prozess besitzt konzeptionell einen eigenen virtuellen Prozessor
  - reale(r) Prozessor(en) zwischen den virtuellen Prozessoren umgeschaltet (Mehrprogrammbetrieb).
  - Umschaltungseinheit heißt Scheduler oder Dispatcher, Scheduling-Algorithmus legt Regeln fest.
  - Umschaltungsvorgang heißt Prozesswechsel oder Kontextwechsel (Context Switch).
  - Multicore-Prozessoren enthalten mehrere Prozessoren auf einem Chip

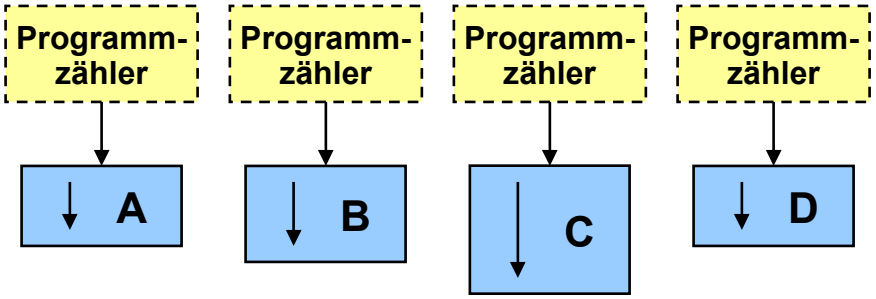
Verdeutlichung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

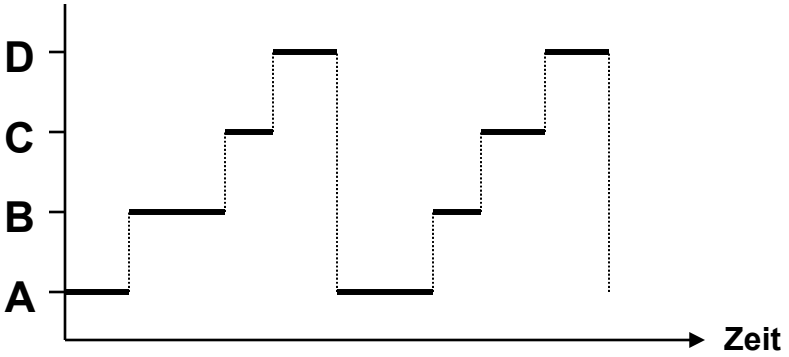
(a) Mehrprogrammbetrieb
mit 4 Programmen



(b) Konzeptionelles Modell:
4 unabhängige, sequenzielle Prozesse



(c) Gantt-Diagramm



nur ein Programm ist zu jedem Zeitpunkt aktiv

Beachte:

- Programm:
  - statische Beschreibung eines sequentiellen Algorithmus
  - dasselbe Programm kann mehrmals (auch gleichzeitig!) innerhalb verschiedener Prozesse ausgeführt werden.
- Prozess:
  - Ausführungsgeschwindigkeit eines Prozesses ist nicht gleichmäßig und nicht reproduzierbar.

    ⇒ Bei der Programmierung sind keine a-priori-Annahmen über den zeitlichen Verlauf zulässig.

    ⇒ Bei zeitkritische Anforderungen, z.B. Realzeit-System, sind besondere Vorkehrungen im Scheduling-Algorithmus notwendig.

Prozessfamilie, Prozesshierarchie

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- In einfachen Systemen (z.B. Gerätesteuerung):
  - Menge der zur Laufzeit existierenden Prozesse häufig statisch
  - wird bei Systemstart erzeugt.
- Allg. Fall:
  - Mechanismus notwendig, um Prozesse dynamisch (zur Laufzeit) durch andere Prozesse erzeugen zu können.

Def

- Erzeugender Prozess heißt Elternprozess (*parent process*), erzeugte Prozesse heißen Kindprozesse (*child processes*).
- Wiederholte Prozesserzeugung durch Kindprozesse

  $\Rightarrow$ baumartig strukturierte Prozessmenge.
- Prozess mit all seinen direkten und indirekten Nachfahren heißt Prozessfamilie.
- Jede baumartig strukturierte Menge von Prozessen heißt Prozesshierarchie (ohne notwendigerweise durch Prozesserzeugung aus der Wurzel entstanden zu sein).

Motivation:

- Prozesse, obwohl unabhängige Einheiten, können aufgrund des Algorithmus logisch voneinander abhängig sein:
  Beispiel

  UNIX shell: `cat datei1 datei2 datei3 | grep egon`
- In Abhängigkeit von den relativen Ausführungsgeschwindigkeiten kann ein Prozess warten müssen, bis eine Eingabe vorliegt.
- Allgemeiner sagt man:
  Er blockiert und wartet auf ein (für ihn externes) Ereignis.
- Einprozessor-System: Prozessor wird dann unmittelbar einem anderen Prozess zugeordnet. Entzug des Prozessors (Suspendierung) in diesem Fall problembegründet.
- Auch möglich: Scheduler entscheidet auf Prozesswechsel, obwohl der erste Prozess weiter ausgeführt werden könnte (Preemption).

Prozesszustände (2)

Damit sinnvolle Prozesszustände:

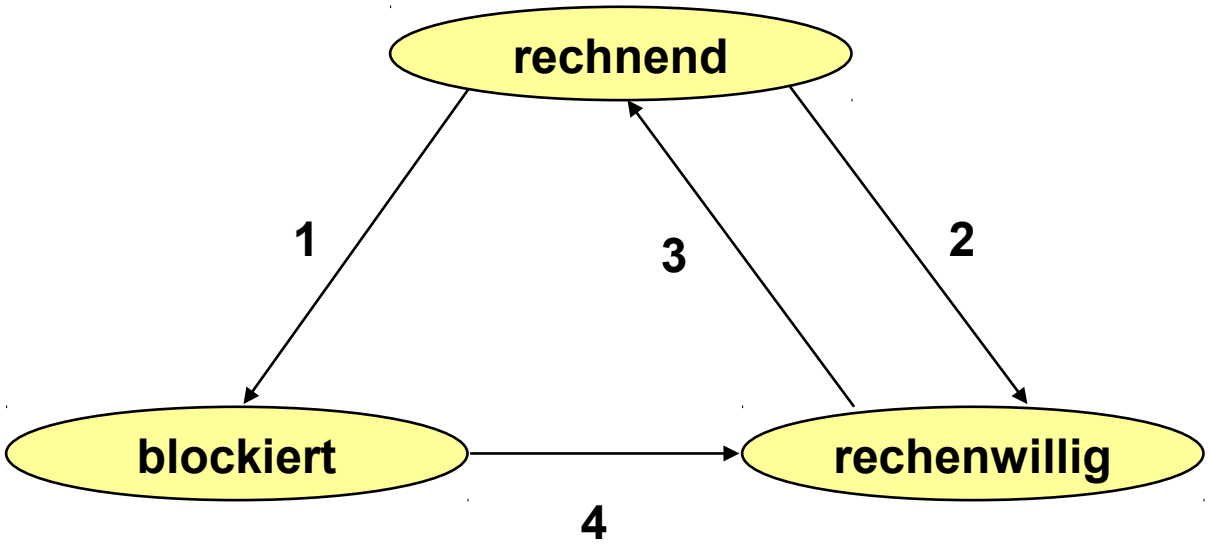
- rechnend (oder aktiv): dem Prozess ist ein Prozessor zugeordnet, der das Programm vorantreibt.
- rechenwillig (oder bereit): Prozess ist ausführbar, aber Prozessor ist anderem Prozess zugeordnet (bzw. alle verfügbaren Prozessoren sind anderen Prozessen zugeordnet).
- blockiert (oder schlafend): Prozess wartet auf Ereignis.
  Er kann solange nicht ausgeführt werden, bis das Ereignis eintritt.

Gelegentlich noch folgende Zustände:

- initiiert: in Vorbereitung (Anfangszustand).
- terminiert: Prozess ist beendet (Endzustand).

Zustandsübergangsdiagramm

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



Zustandsübergänge:

1 rechnend → blockiert: Versetzung in den Wartezustand, (Warten auf Ereignis).

2 rechnend → rechenwillig: Scheduler entzieht den Prozessor.

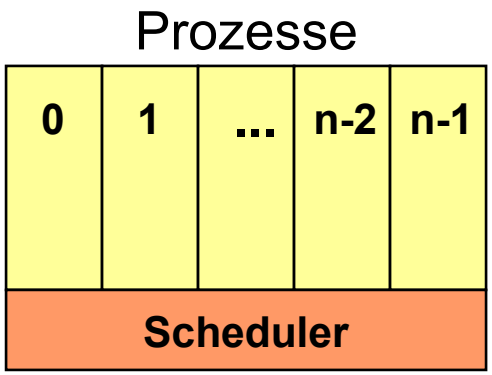
3 rechenwillig → rechnend: Scheduler teilt Prozessor zu.

4 blockiert → rechenwillig: Ereignis tritt ein.

Zusammenfassung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Das Prozessmodell vereinfacht die Beschreibung der Aktivität des Rechensystems.
- Die ineinander verwobene Aktivität des Systems wird durch eine Menge von sequentiellen Prozessen beschrieben.
- Die unterste Schicht eines Betriebssystems behandelt die Unterbrechungen und ist für das Scheduling verantwortlich. Der Rest des Systems besteht aus sequentiellen Prozessen.

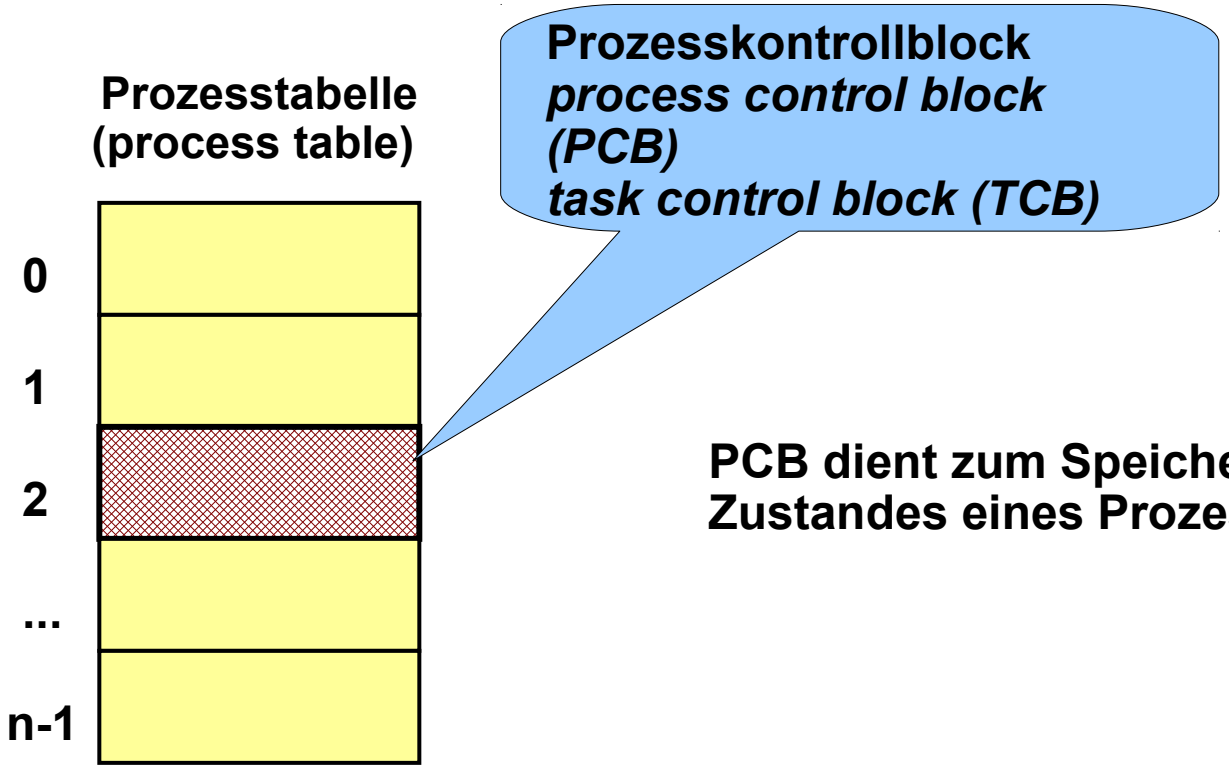


Anmerkungen

- Mechanismen zur Synchronisation und Kommunikation von Prozessen sind notwendig → Kap. 5
- Programmierung von Anwendungen aus mehreren nebenläufigen Prozessen heißt Concurrent Programming.
- Klass. Prozessmodell verfeinert durch Einführung sog. Leichtgewichtsprozesse (light weight processes, oder Threads = Fäden), die mehrere Aktivitätsträger in einem einzigen Adressraum darstellen → 3.3 .

3.2. Implementierung von Prozessen

Datenstrukturen im BS-Kern zur Prozessverwaltung:



PCB dient zum Speichern des gesamten Zustandes eines Prozesses

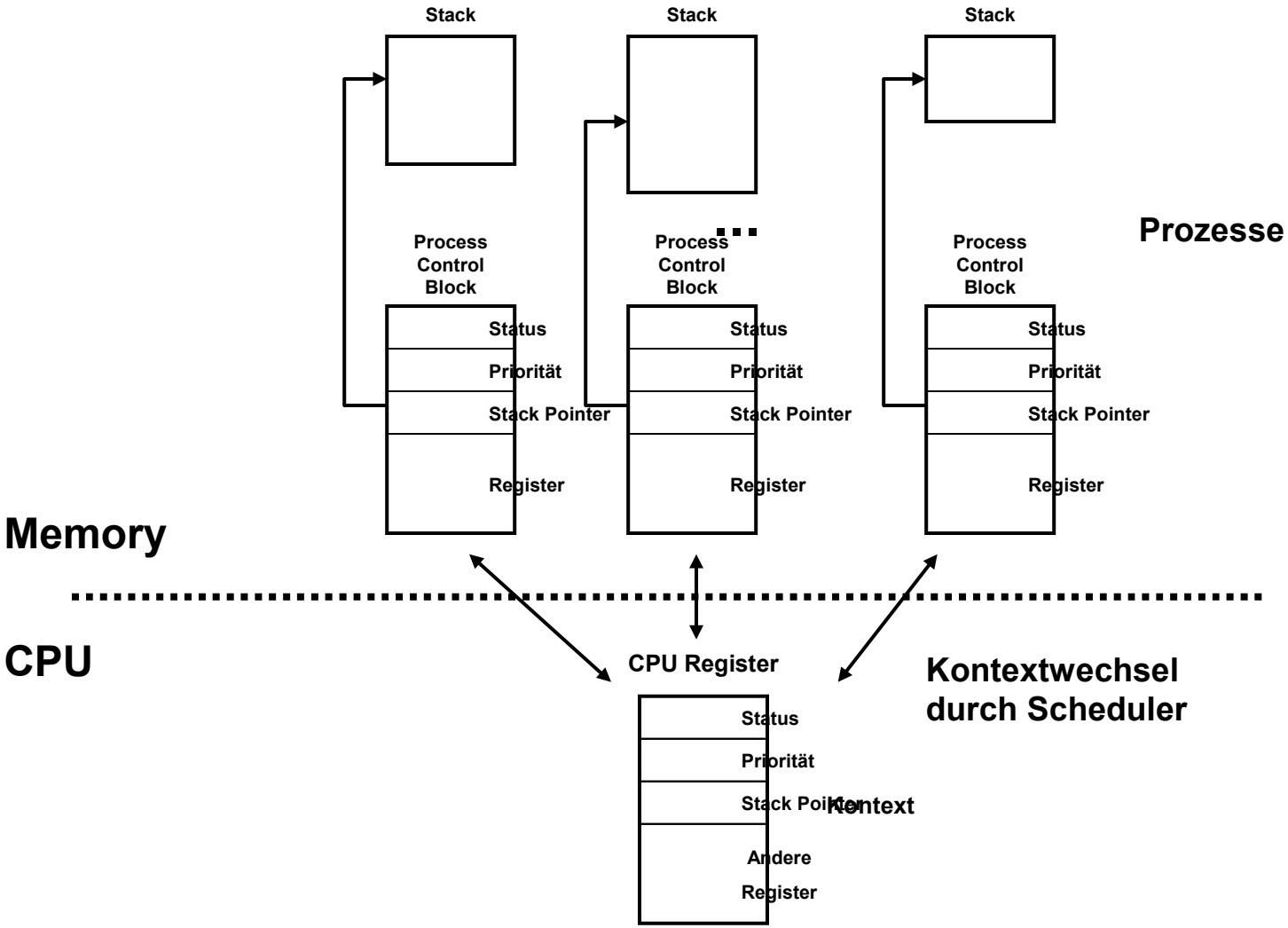
Prozesskontrollblock

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Typische Felder in einem Prozesskontrollblock:

Prozessverwaltung	Speicherverwaltung	Dateisystem
Register	Zeiger auf Textsegment	Wurzelverzeichnis
Programmzähler	Zeiger auf Datensegment	aktuelles Verzeichnis
Programmstatuswort	Zeiger auf BSS-Segment	UMASK-Maske
Stack-Zeiger	Prozessgruppe	offene Dateideskriptoren
Prozesszustand	reale UID	effektive UID
Prozessnummer	effektive UID	effektive GID
Elternprozessnummer	reale GID	Aufrufparameter
Prozesserzeugungszeitpunkt	effektive GID	verschiedene Flags
Terminierungsstatus	verschiedene Flags	
verbrauchte Prozessorzeit		
Prozessorzeit der Kinder		
Alarm-Zeitpunkt		
Signalstatus		
Signalmaske		
unbearbeitete Signale		
Zeiger auf Nachrichten		
verschiedene Flags		

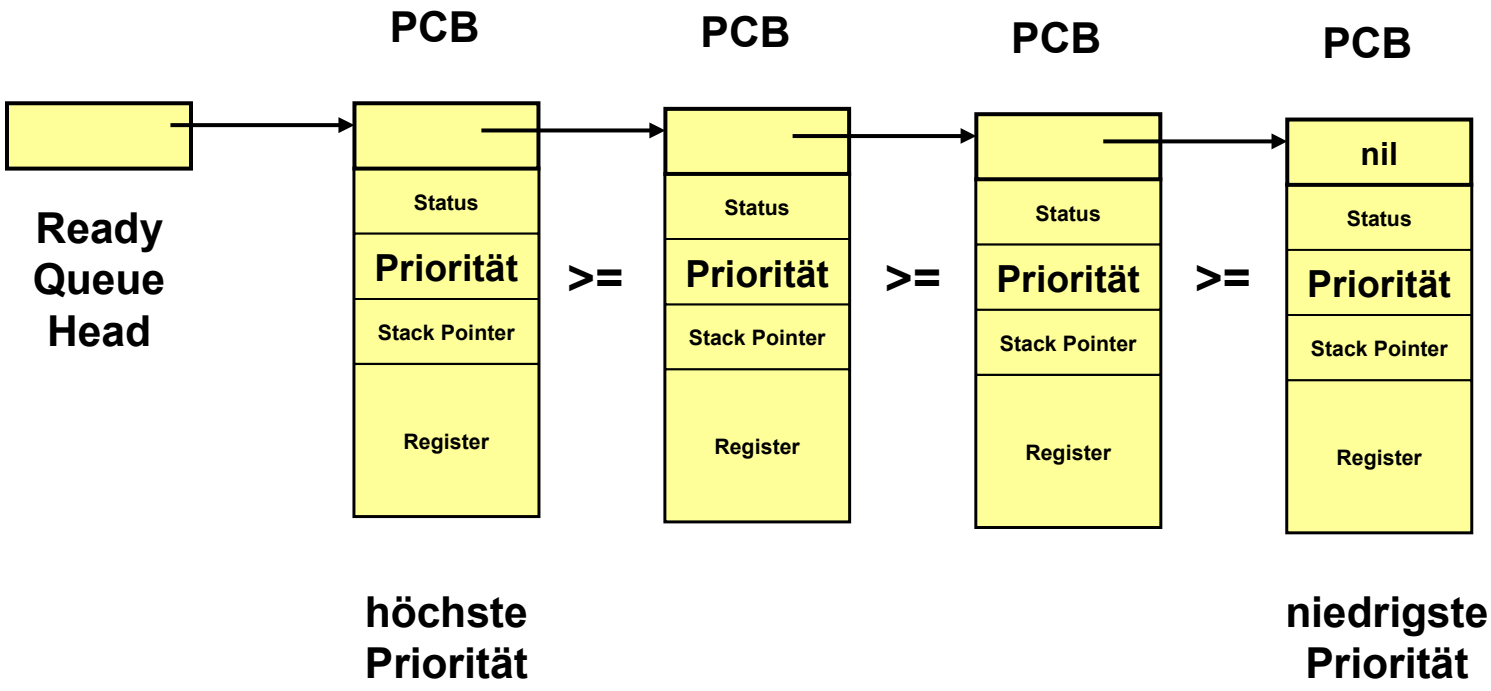
zusätzlich: Zeiger zur Verkettung des PCB in Warteschlangen


Stack
Stack
Stack
Process Control Block
Process Control Block
Process Control Block
Status
Priorität
Stack Pointer
Register
Prozesse
Memory
CPU
CPU Register
Kontextwechsel durch Scheduler
Kontext
Andere Register

Warteschlangenstruktur

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Einfache Struktur der Liste der rechenwilligen Prozesse (Bereit-Liste oder Ready Queue):

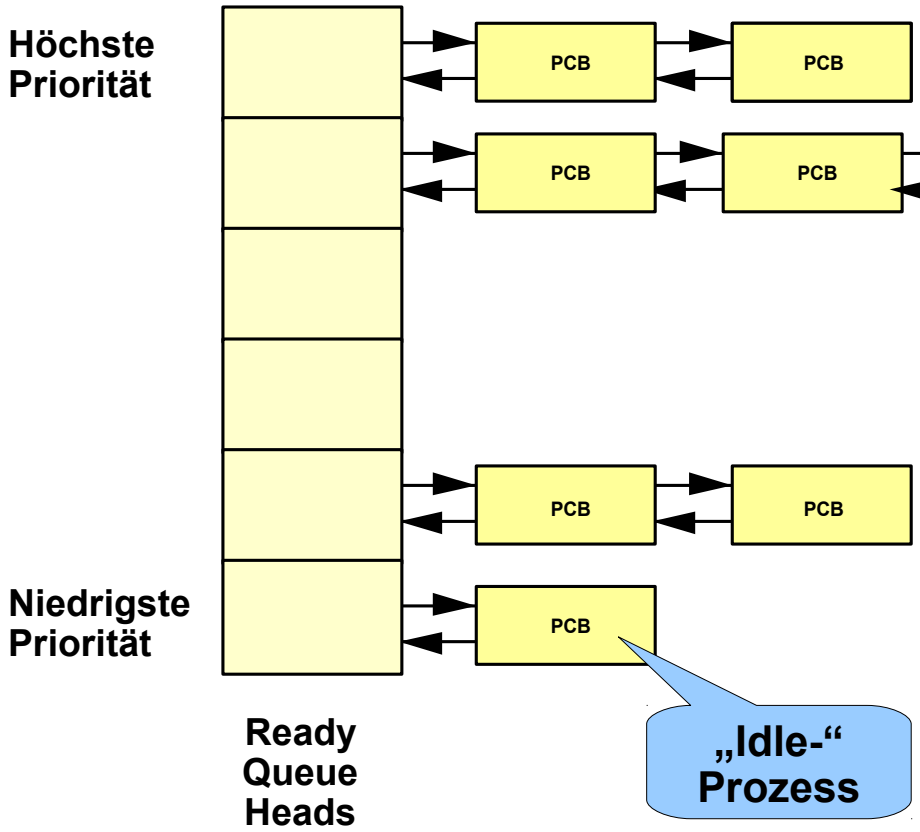


Bei gleicher Priorität: Einreihen nach „first in / first out“
==> Nachteil: Laufzeit abhängig von Prozessanzahl

Warteschlangenstruktur (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

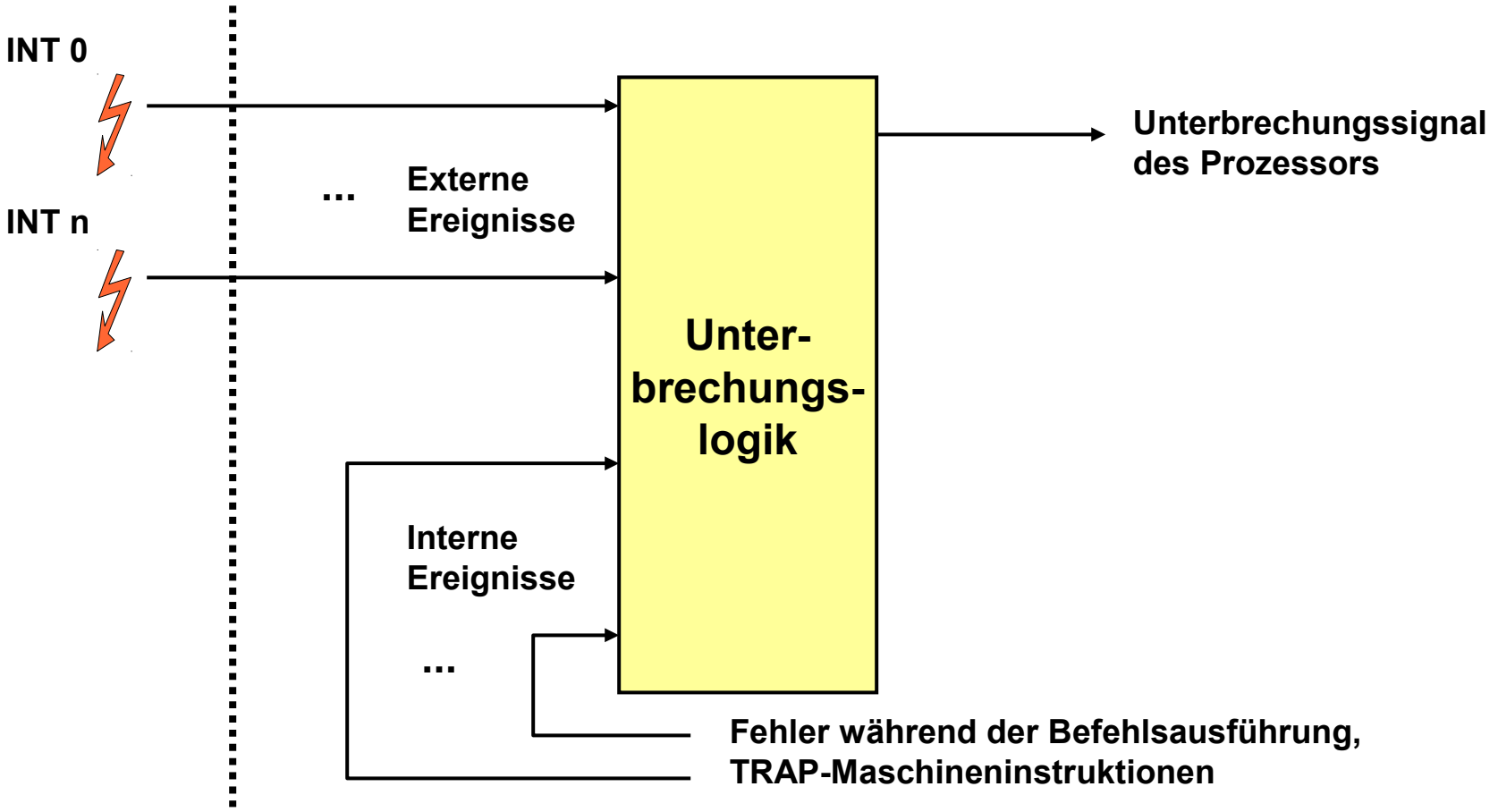
Typische Struktur der Liste der rechenwilligen Prozesse (Bereit-Liste oder Ready Queue):



- **Eine** Ready-Queue **pro Prioritätsstufe**
- Vorteil: Schnelles Einreihen in konstanter Laufzeit („O(1)-Scheduler“)
- Nachteil: Feste Anzahl an Prioritäten

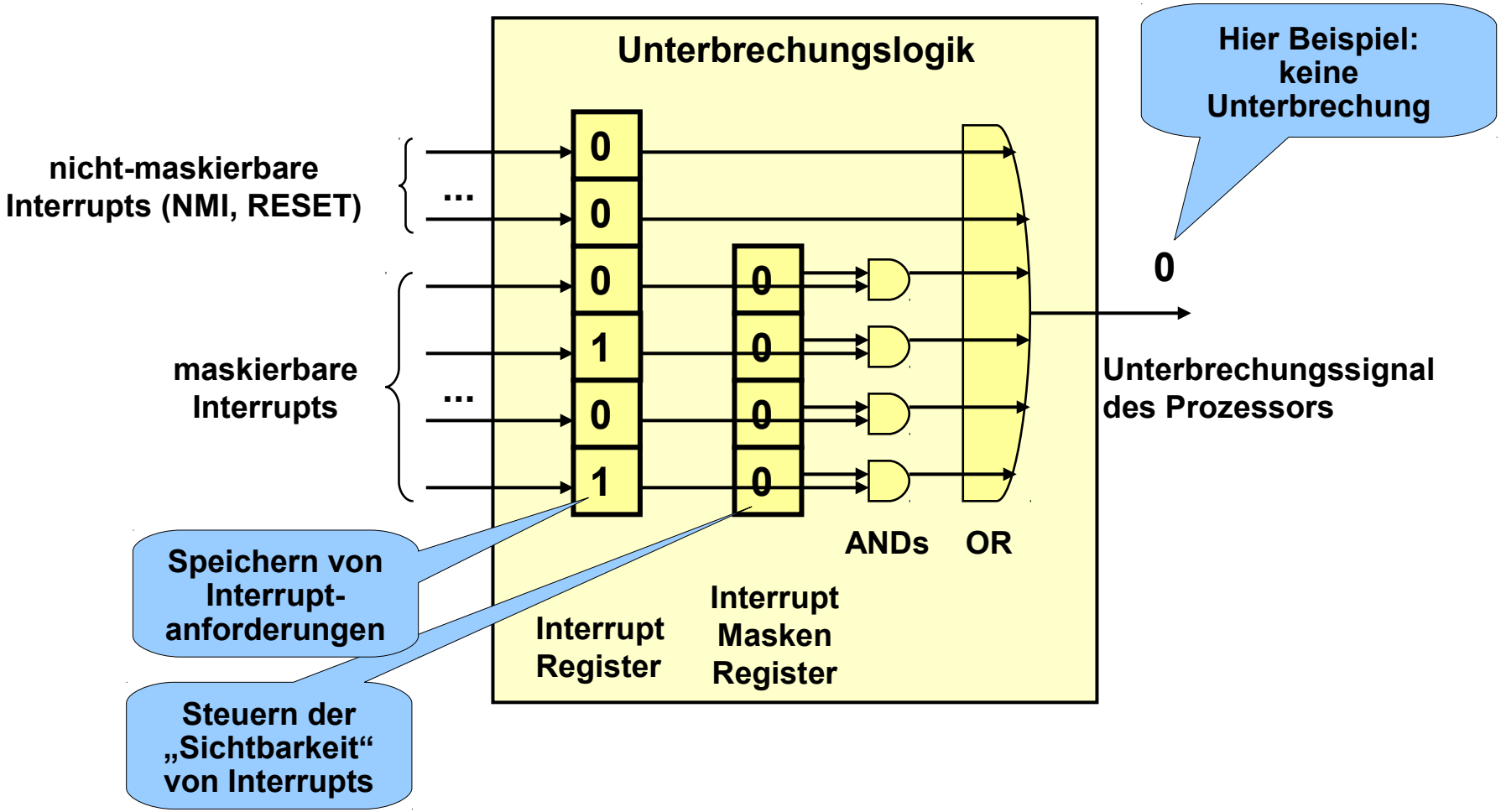
Unterbrechungslogik

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



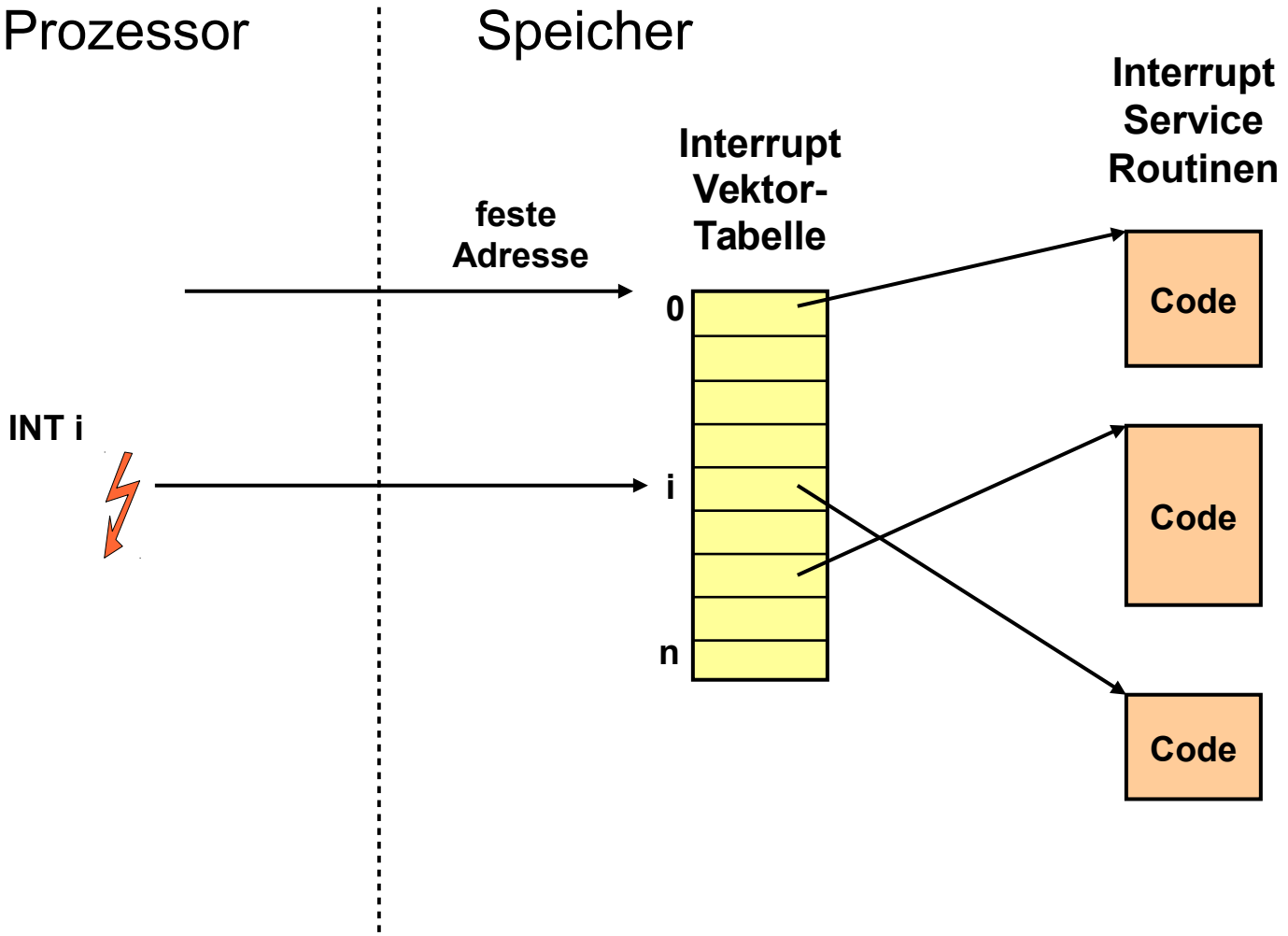
Unterbrechungslogik (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



Unterbrechungsbehandlung

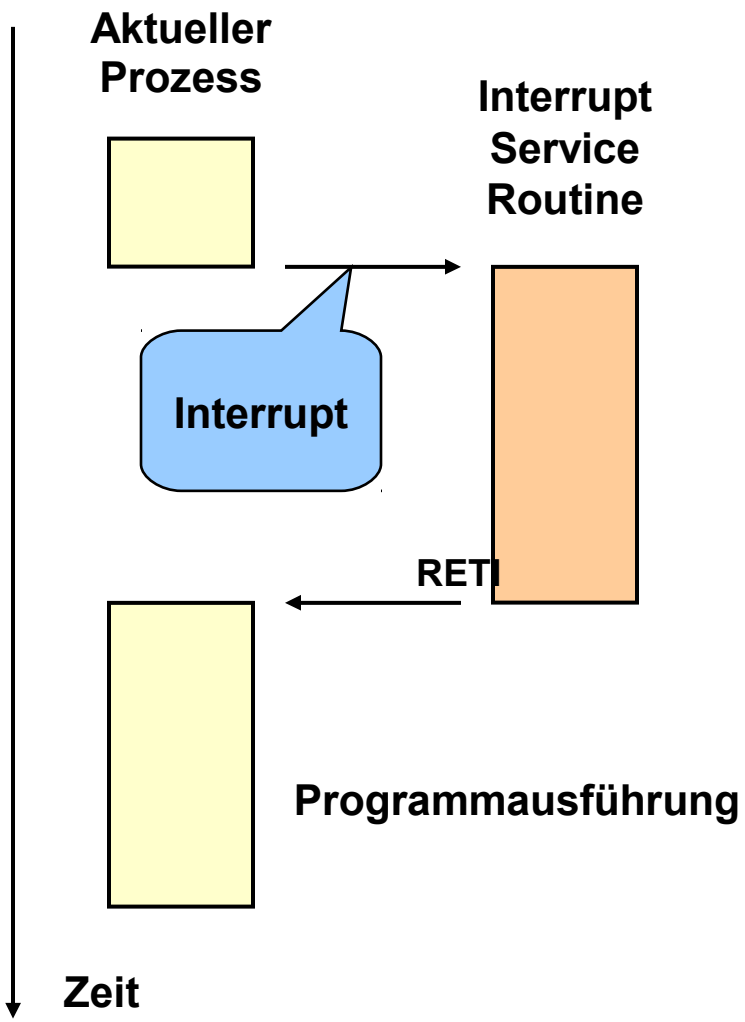
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



Unterbrechungsbehandlung (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Prinzipieller Ablauf



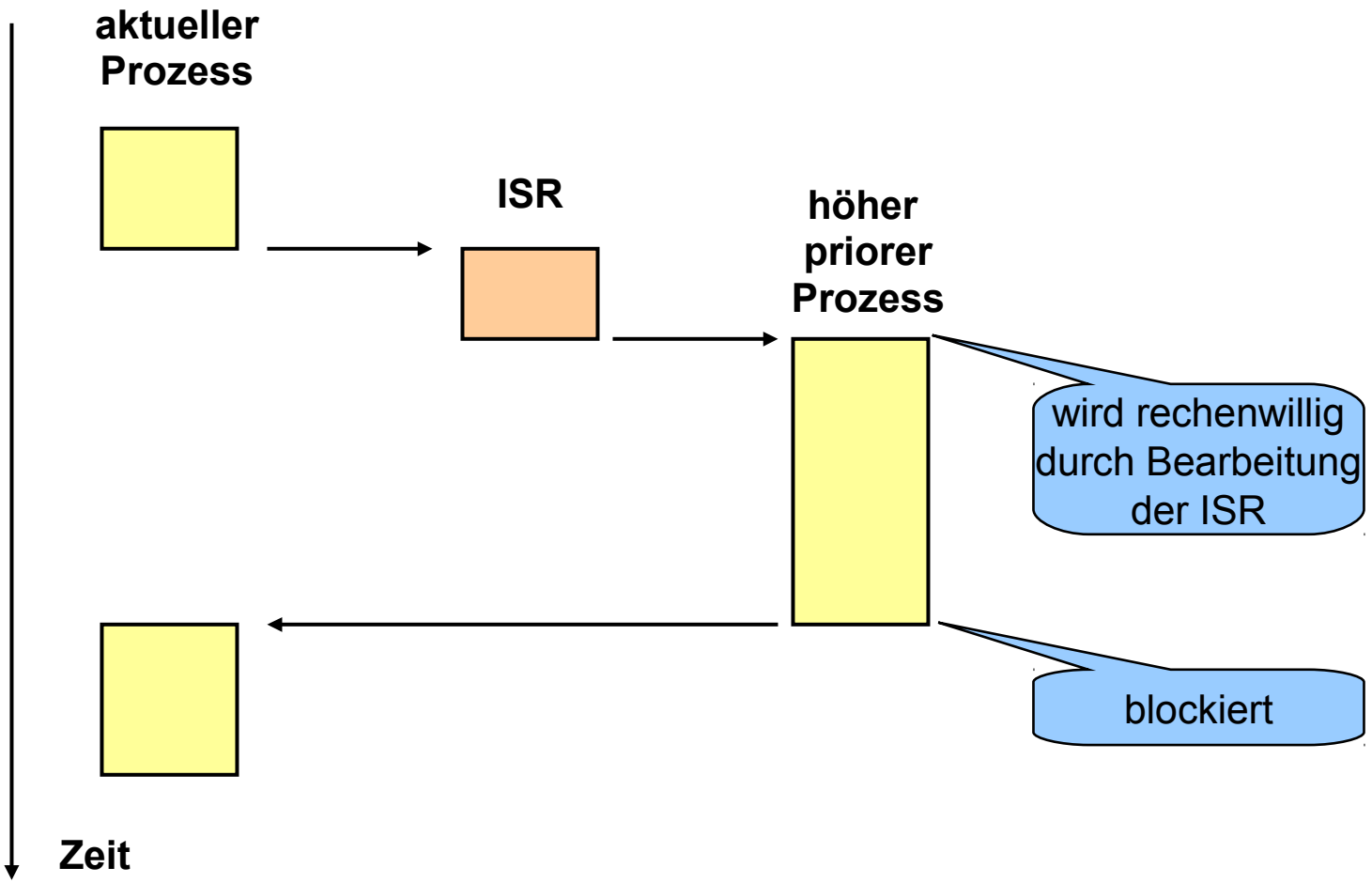
Typische Ausführungsschritte:

1. Programmzähler (u.a.) wird durch Hardware auf dem Stack abgelegt.
2. Hardware lädt den neuen Programmzählerinhalt aus dem Unterbrechungsvektor.
3. Eine Assembler-Routine rettet die Registerinhalte.
4. Eine Assembler-Routine bereitet den neuen Stack vor.
5. Eine C-Prozedur markiert den unterbrochenen Prozess als rechenwillig.
6. Der Scheduler bestimmt den Prozess, der als nächster ausgeführt werden soll.
7. Die C-Prozedur gibt die Kontrolle an die Assembler-Routine zurück.
8. Die Assembler-Routine startet den ausgewählten Prozess.

Unterbrechungsbehandlung (4)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Damit Umschaltung zwischen Benutzerprogrammen:



Beispiel UNIX

Systemaufrufe zur Prozessverwaltung:

`pid_t fork(void)`

Erzeugen einer Kopie des Prozesses (Child), Parent erhält pid des Kindes zurück oder -1 bei Fehler, Kind erhält 0 als Ergebnis.

`int execve(char* name, char* argv[], char* envp[])`

Überlagern des ausgeführten Programms eines Prozesses (Code, Daten, Stack) durch neues Programm. Andere Varianten:
`execl, execle, execlp, execv, execvp.`

`pid_t getpid(void)`

Rückgabe der eigenen Process Id.

`pid_t getppid(void)`

Rückgabe der Process Id des Elternprozesses.

Systemaufrufe zur Prozessverwaltung (2):

`exit(status)` — Beende den laufenden Prozess und stelle dem Parent den Exit-Status zur Verfügung.

`pid_t wait(int* status)`
Warten auf die Beendigung eines Kindprozesses. Dessen Id wird über den Rückgabewert, sein Status über `status` zurückgegeben.

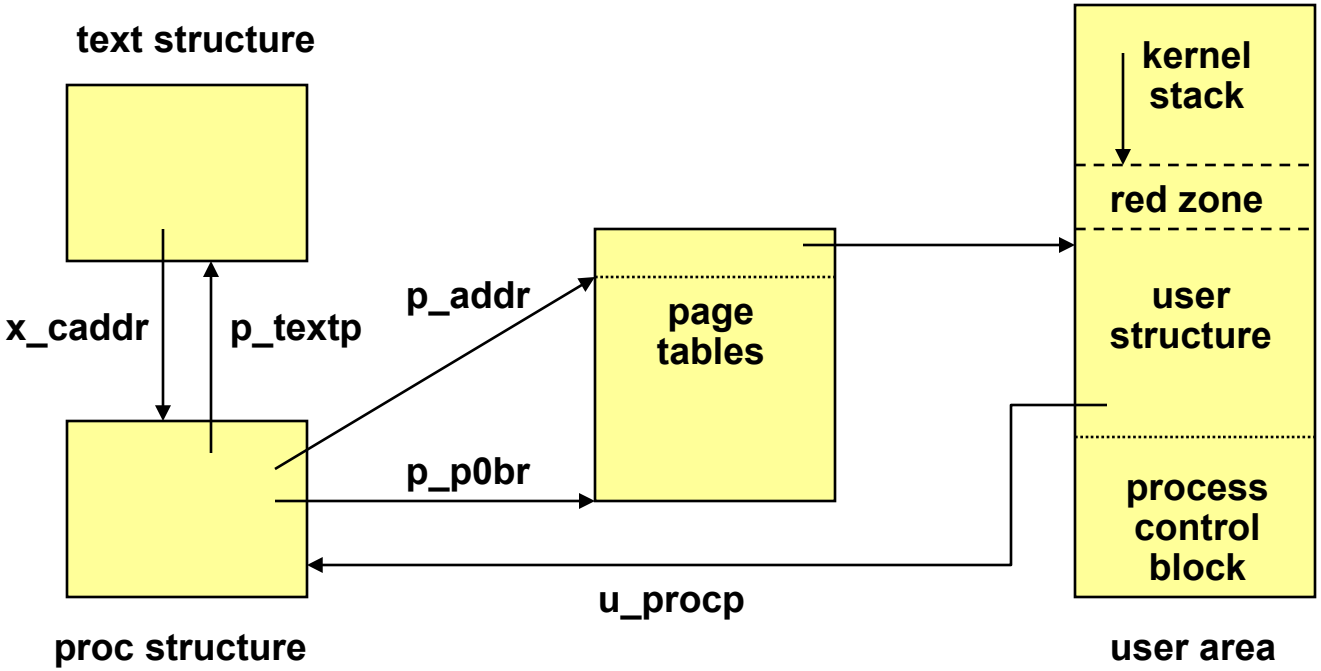
`pid_t waitpid(pid_t pid, int* status, int opts)`
Warten auf das Ende eines bestimmten Kindprozesses, dessen Id über den Parameter `pid` übergeben wird.

(wird im Praktikum vertieft)

Beispiel UNIX (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Kerndatenstrukturen zur Repräsentierung eines Prozesses in 4.3BSD UNIX (Überblick):



aus Leffler et al: Design and Implementation of the 4.3BSD UNIX Operating System

Beispiel UNIX (3)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

proc structure (residenter Teil der Prozessbeschreibung):

Kategorie	Name	Bedeutung
scheduling	p_pri	current priority
	p_userpri	user priority based on p_cpu and p_nice
	p_nice	user-requested scheduling priority
	p_cpu	recent CPU usage
	p_slptime	amount of time sleeping
identifiers	p_pid	unique process identifier
	p_ppid	parent process identifier
	p_uid	user identifier
memory management	p_textp	pointer to description of executable file
	p_P0br	pointer to process page tables
	p_szpt	size of process page tables
	p_addr	location of user area
	p_swaddr	location of user area when swapped out
synchronisation	p_wchan	event process is awaiting
signals	p_sig	mask of signals pending delivery
	p_sigignore	mask of signals being ignored
	p_sigcatch	mask of signals being caught
	p_pgrp	process-group identifier
resource accounting	p_rusage	pointer to rusage structure
	p_quota	pointer to disk quota structure
timer management	p_time	amount of real-time until timer expires

aus Leffler et al: Design and Implementation of the 4.3BSD UNIX Operating System

Beispiel UNIX (4)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Prozesszustände (process states):

`SSLEEP`	schlafend = blockiert, wartend auf ein Ereignis
`SRUN`	runnable = rechenwillig und rechnend
`SIDL`	initiiert, transienter Zustand während Erzeugung
`SZOMB`	terminiert, transienter Zustand nach Beendigung
`SSTOP`	angehalten oder in trace mode

Beispiel UNIX (5)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

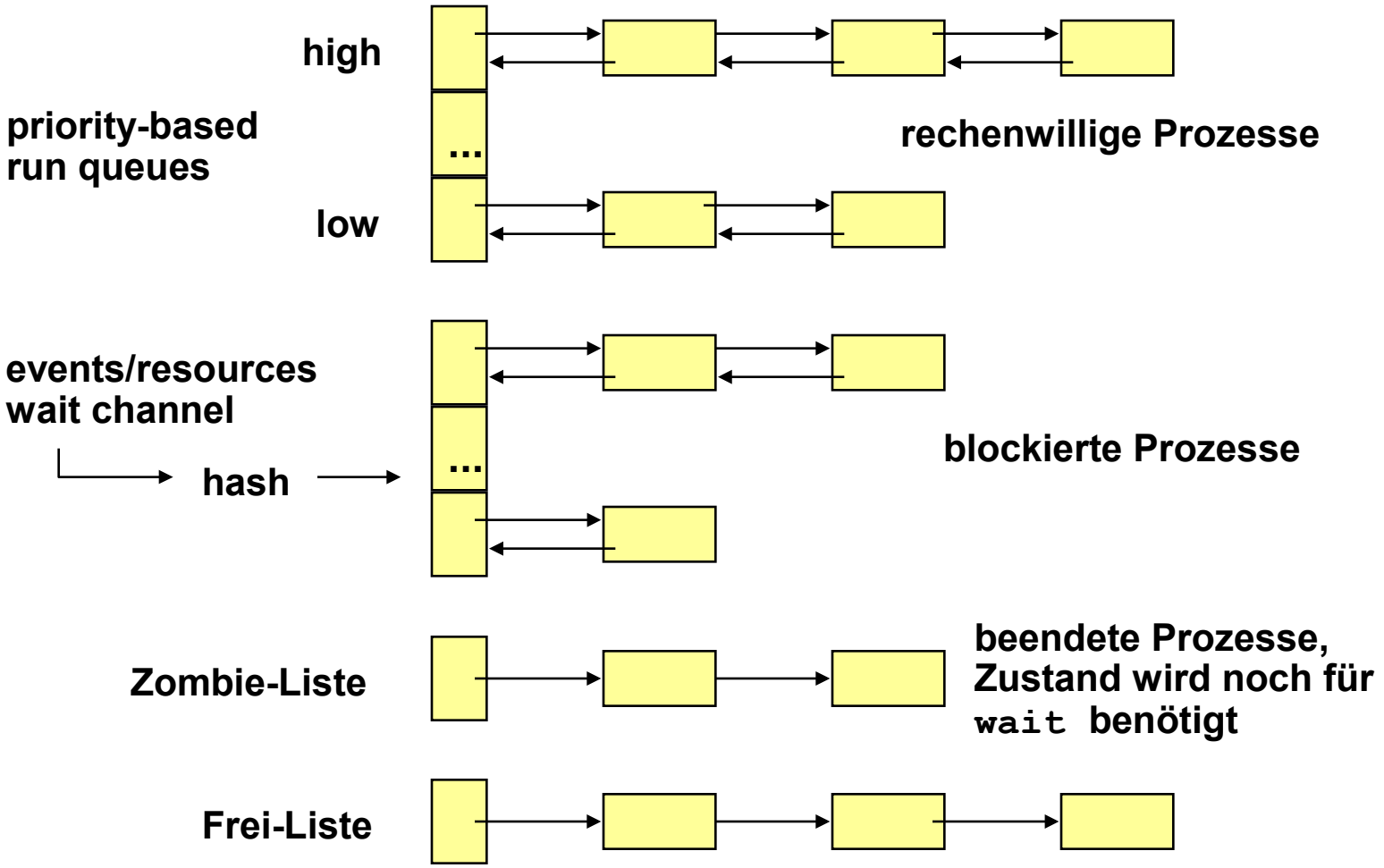
user structure
(auslagerbarer Teil der Prozessbeschreibung):

- **Prozessorzustand (Registerinhalte usw. = eigentlicher PCB, getrennt für user mode und kernel mode).**
- **Zustand bzgl. aktueller Systemaufrufe.**
- **Descriptor-Tabelle**
  (Liste der geöffneten Objekte des Prozesses).
- **Accounting-Information.**
- **Kernel Stack des Prozesses**
  (Stack für user mode ist im Prozessadressraum).

Beispiel UNIX (6)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Organisation der Warteschlangen:



Beispiel UNIX (7)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Wesentliche interne Kern-Routinen zur Prozessumschaltung:

`swtch()` **Aufruf des Schedulers zur Auswahl des rechenwilligen Prozesses mit der höchsten Priorität und darin Aufruf von** `resume()`.

`resume(...)`
Lade Prozessor-Register mit ausgewähltem Prozesskontrollblock und führe diesen Prozess fort.

`sleep(wait channel, priority)`
Freiwillige Abgabe des Prozessors bei Übergang nach blockiert und Zielpriorität bei Erwachen.

`wakeup(wait channel)`
Setze alle Prozesse rechenwillig, die auf das Ereignis blockiert warten.

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Motivation

- Prozesserzeugung, Prozessumschaltung und Prozesskommunikation sind teuer
  - = rechenzeitaufwendig zur Laufzeit
  - Verluste auch durch Cache-Misses
- Wie nutzt man mehrere Prozessoren eines Multiprozessors für eine Applikation?
  - Z.B. mehrere kooperierende Prozesse auf verschiedenen Prozessoren
  - Muss vom Applikationsprogrammierer ausprogrammiert werden !
- Wie strukturiert man einen Server-Prozess, der Anforderungen von mehreren Klienten bedienen kann?
  - Ein Server-Prozess = keine Parallelität
  - Multiplexing für verschiedene Klienten von Hand
    = komplexe Programmierung

Lösung

- Einführung von billiger Nebenläufigkeit in einem Prozessadressraum durch „Leichtgewichtsprozesse", sogenannte Threads.

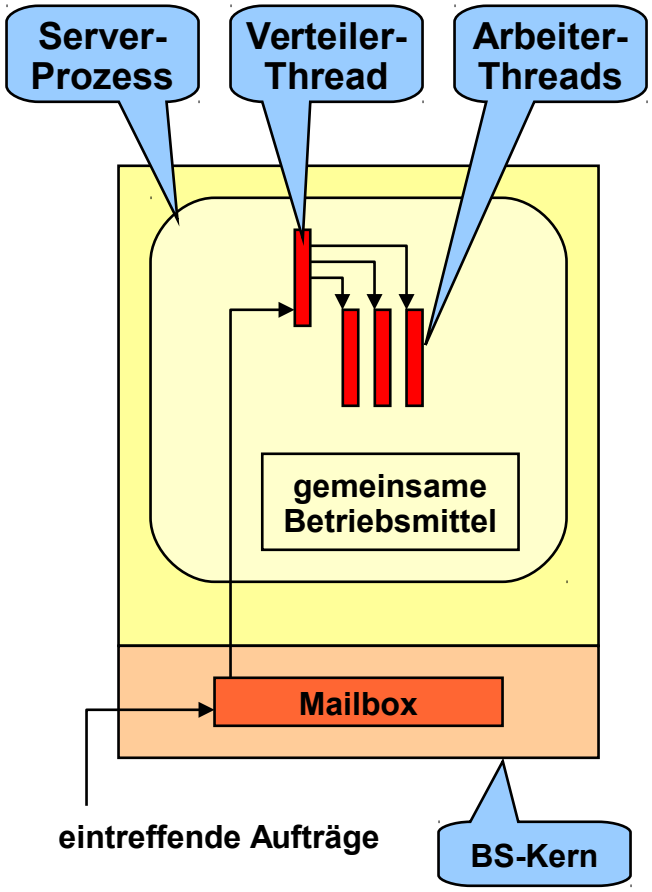
Definition

- Prozess (Einheit der BM-Verwaltung)
  - ausführbares Programm, das Code und globale Daten definiert.
  - privater Adressraum.
    Code und Daten über Adressraum zugreifbar.
  - Menge von Betriebsmitteln
    - geöffnete Objekte, Betriebssystem-Objekte wie z.B. Timer, Signale, Semaphore
    - dem Prozess durch das BS als Folge der Programmausführung zugeordnet.
  - Menge von Threads.
- Threads (Aktivitätsträger)
  - Idee einer „parallel ausgeführten Programmfunktion“
  - Eigener Prozessor-Context (Registerinhalte usw.)
  - Eigener Stack (i.d.R. zwei, getrennt für user und kernel mode)
  - Eigener kleiner privater Datenbereich (Thread Local Storage)
  - Threads eines Prozesses nutzen gemeinsam Programm, Adressraum und alle Betriebsmittel.

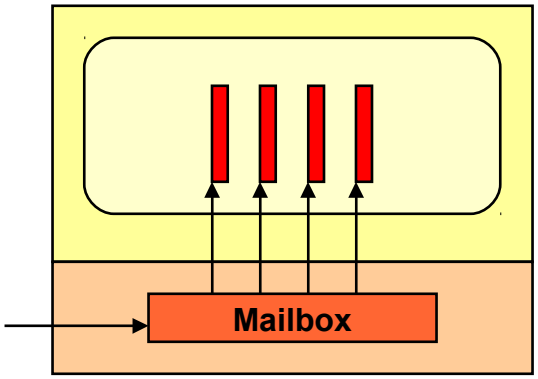
Kooperationsformen

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

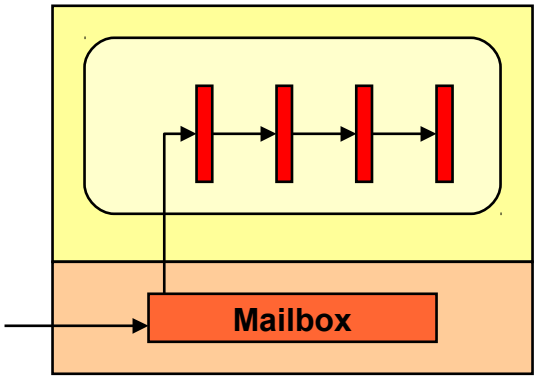
(a) Verteiler/Arbeiter-Modell



(b) Team-Modell



(c) Fließband-Modell



aus Tanenbaum: Moderne Betriebssysteme

Schnittstellen

- Nicht einheitlich
- Bsp: Java: Klasse Thread in java.lang
  - Z.B. Implementierung der Schnittstelle java.lang.Runnable und Implementierung von Methode run().
  - Thread-Modell in JVM implementiert
- Bsp: C++: Boost Threads verbreitet
- Bsp: Windows
  - C-Schnittstelle für Windows API, u.a. CreateThread(...)
- Bsp: LinuxThreads (veraltet)
  - clone() system call
    - Erzeugung eines Prozesses mit Angabe detaillierter Flags, was gemeinsam mit erzeugendem Prozess genutzt werden soll
  - Gehört nicht zum Unix Standard

  ⇒ Programme nicht portierbar

Schnittstellen (2)

- POSIX Threads (Pthreads)
  - Verbreitete Standard-Programmierschnittstelle für Threading und entsprechende Synchronisation
  - Standardisiert in IEEE POSIX 1003.1c (1995)
  - Pthreads: Implementierungen dieses Standards
  - Auf vielen Systemen, insbesondere auch Multiprozessorsystemen, z.B. Linux, Solaris, MacOS X, FreeBSD
  - Neben C / C++ auch für andere Prgrammiersprachen: z.B. Fortran
  - Teilweise Bestandteil der libc
  - Ca. 50 Funktionen

- API-Aufrufe zum Thread-Management

  (wird fortgesetzt für andere Funktionsbereiche wie datenbezogene Synchronisation)

```
#include <pthread.h>

int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
```
Erzeugen eines Threads

```
void pthread_exit(void *retval);
```
Sich selbst beenden

```
pthread_t pthread_self(void);
```
Thread-Identifier des aktuellen Threads ermitteln

```
int pthread_join(pthread_t th, void **thread_return);
```
Warten auf Beendigung eines Threads

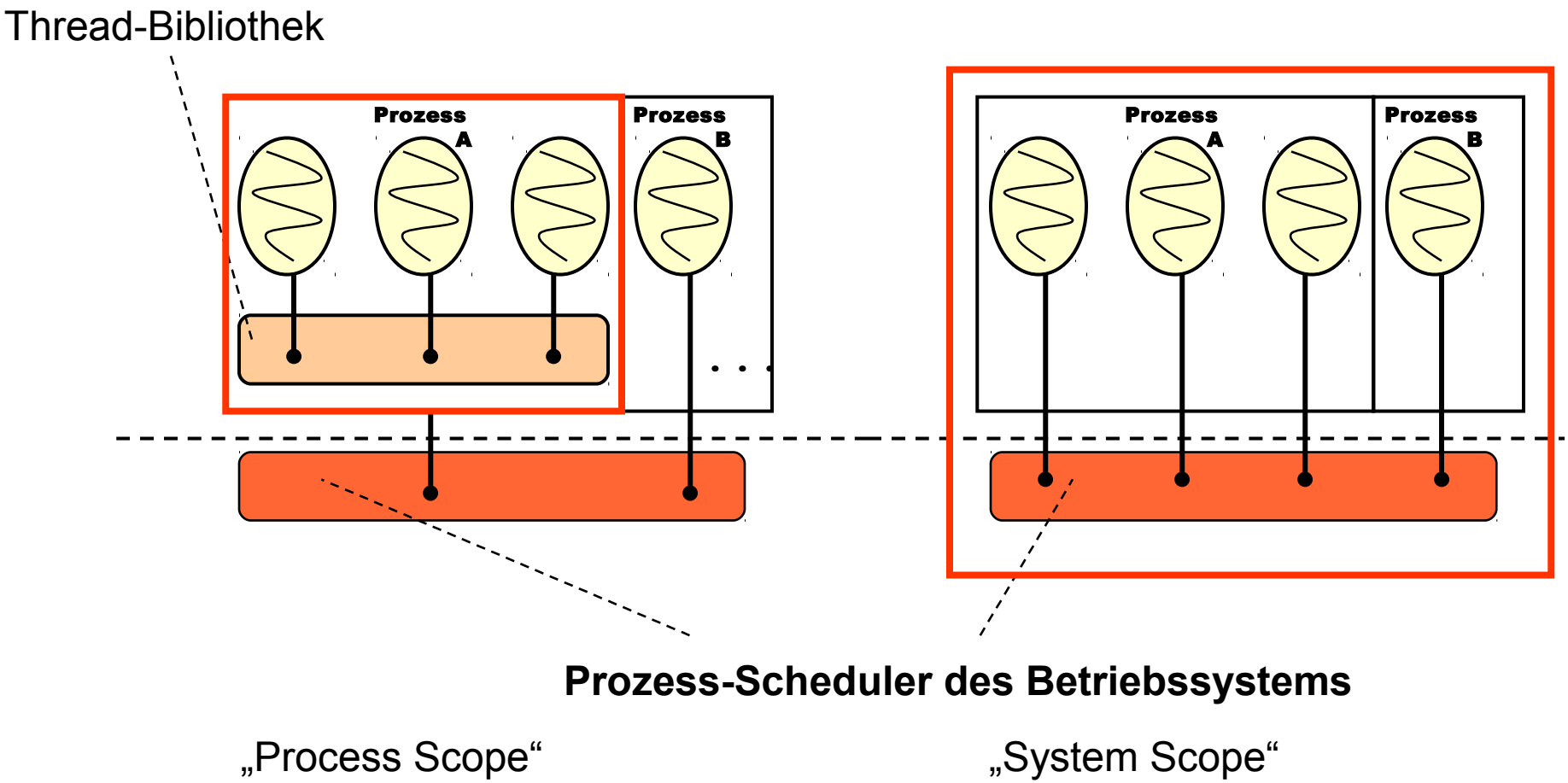
```
int pthread_cancel(pthread_t thread);
```
Beenden eines anderen Threads

Implementierung von Threads

- Im BS-Kern:
  genannt Kernel Threads:
  - Betriebssystem unterstützt Threads
  - Threads sind die Einheiten, denen Prozessoren zugeordnet werden (Schedulable Entities).
  - 1:1-Zuordnung
  - Beispiele: Windows NT, Mach, Chorus, Amoeba
- Nur in Thread Library:
  genannt User-Level Threads
  - Programmierung mit Threads, aber BS-Scheduler kennt nur übliche Prozesse.
  - 1:n-Zuordnung (allg. m:n)
  - Beispiel:
    POSIX Pthreads in OSF/DCE

Kernel Thread vs. User level Thread

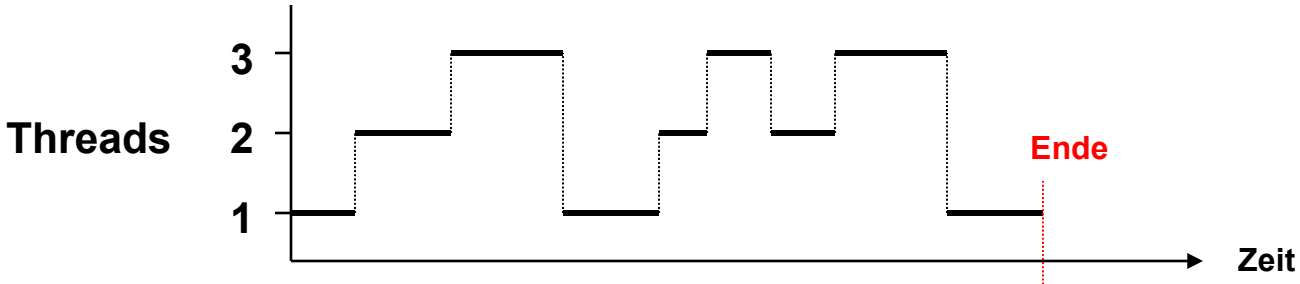
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



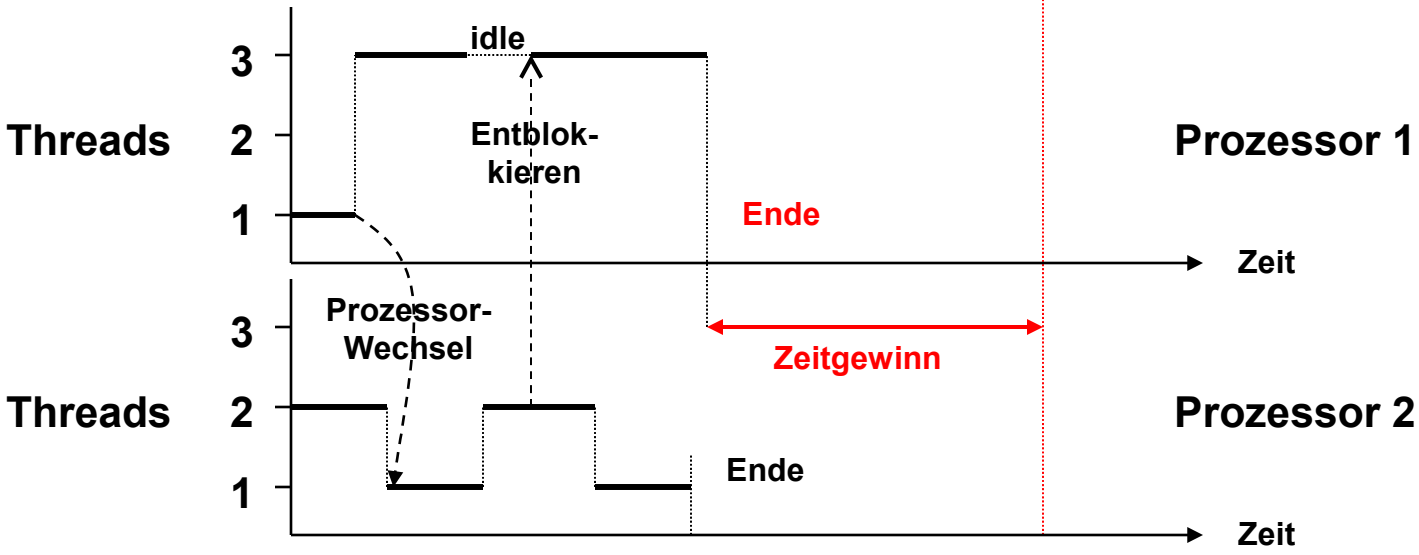
Ausführung von Kernel-Threads

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Ein-Prozessor-System:



Zwei-Prozessor-System:



Beispiel: Linux (ab Kernel 2.6)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Native POSIX Thread Library (NPTL)
- Federführung Red Hat
- Ziele:
  - Konformität zu POSIX Pthreads
  - Gute Multiprozessor-Performance
  - Niedrige Erzeugungskosten
  - Kompatibilität zu LinuxThreads
- 1:1-Implementierung
  - Kernel verwaltet Prozesse
  - pthread_create führt zu neuem Prozess unter Nutzung von clone()
  - Spezielle Kernel-Unterstützung und viel Optimierung im Kernel (z.B. sog. Futexes)
- Nutzung zusammen mit GNU libc (glibc)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Was haben wir in Kap. 3 gemacht?

- Konzept des sequentiellen Prozesses (Wichtig!).
- Strukturierung von Aktivität durch eine Menge von sequentiellen Prozessen, die zueinander nebenläufig ausgeführt werden.
- Betriebssystem bietet Anwendungsprogrammierern ein solches Prozesskonzept an der Dienstschnittstelle zur Strukturierung von Anwendungen.
- Das Betriebssystem kann Prozesse auch intern zur Strukturierung höherer Funktionalität nutzen.
- Ansätze besprochen, wie Betriebssystem das Prozesskonzept implementiert (prinzipiell und speziell am Beispiel UNIX).
- Thread-Konzept als performante „Leichtgewichtsprozesse“ vorgestellt.