Prozesse und Threads

Peter Puschner
Institut für Technische Informatik
peter@vmars.tuwien.ac.at

Ziel:

“Gleichzeitiges”, kontrolliertes Ausführen von Programmen auf einem Rechner

- Welche Mechanismen sind nötig?
- Welche Datenstrukturen brauchen wir?

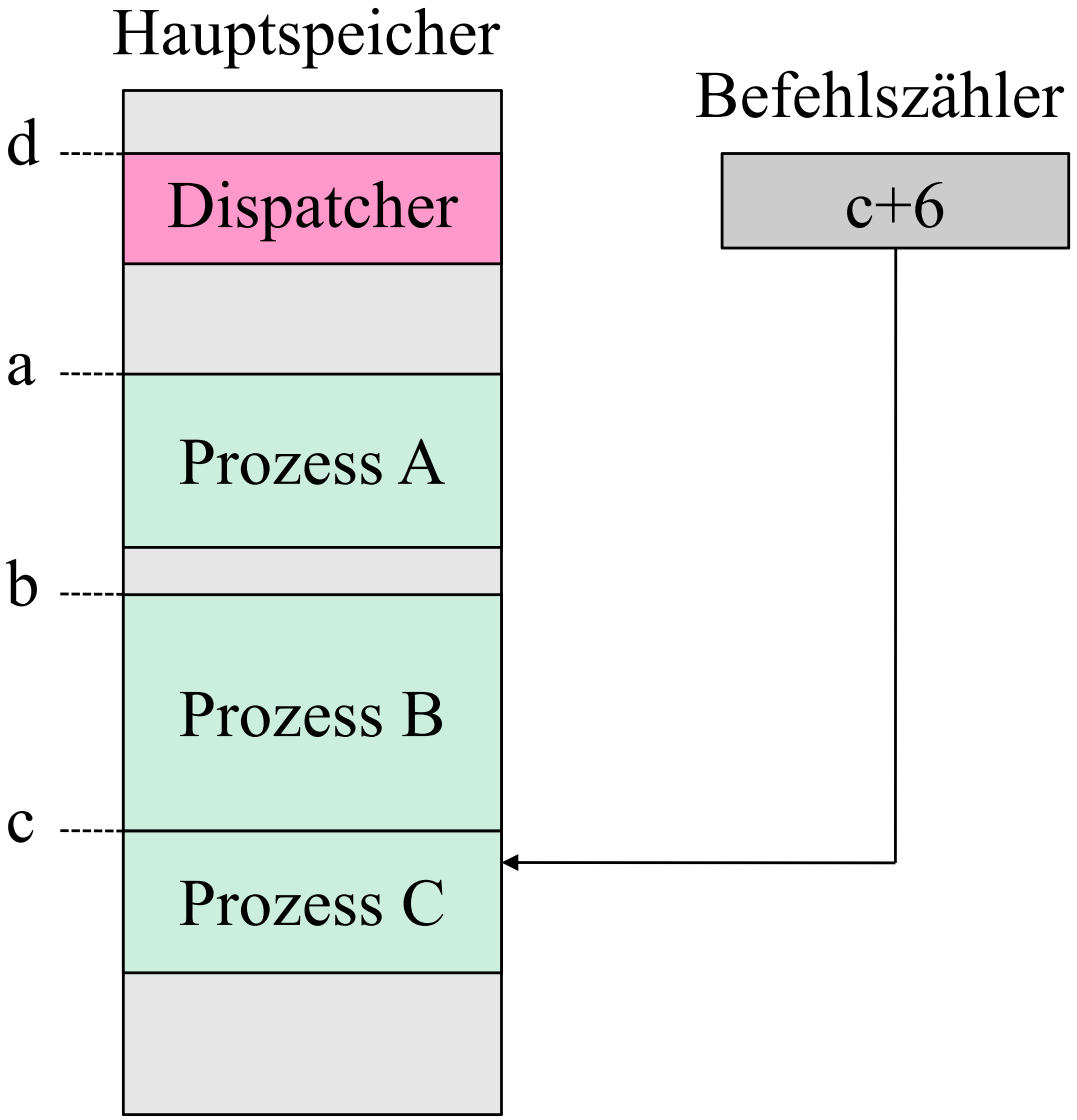
- Begriffe und Konzepte
- Prozesszustände
- Kontrollstrukturen des BS
- Prozesse – BS-Implementierungen
- Threads

Prozess

"Animated Spirit of a program"

- ausführbares Programm
- zugehörige Daten (Variable, Stack, Puffer, etc.)
- Kontext
  - akt. Zustand des Prozesses (PC, Register, etc.)
  - Daten zur Prozessverwaltung (Wartebedingung, Priorität, etc.)

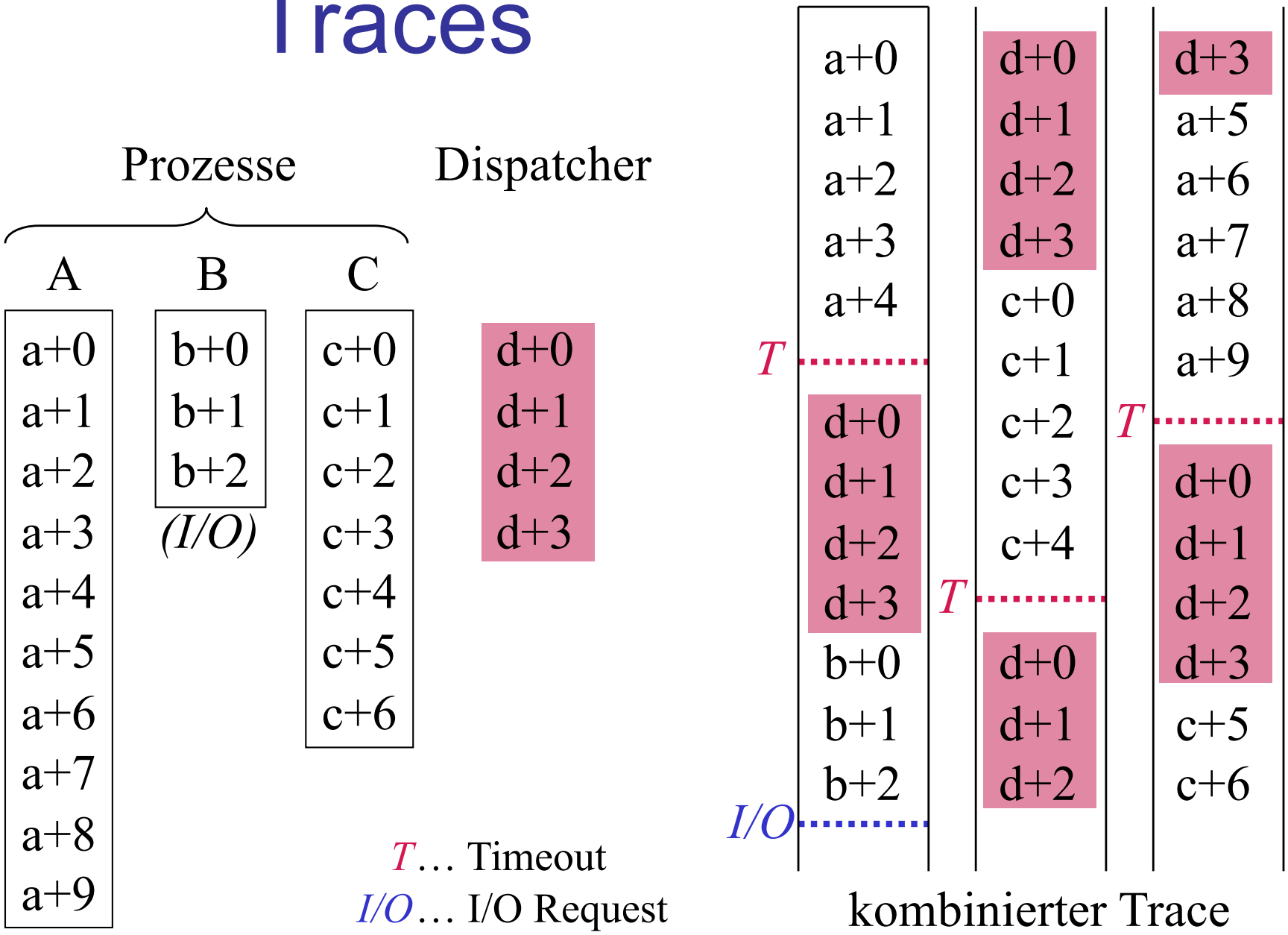
Prozesse im BS



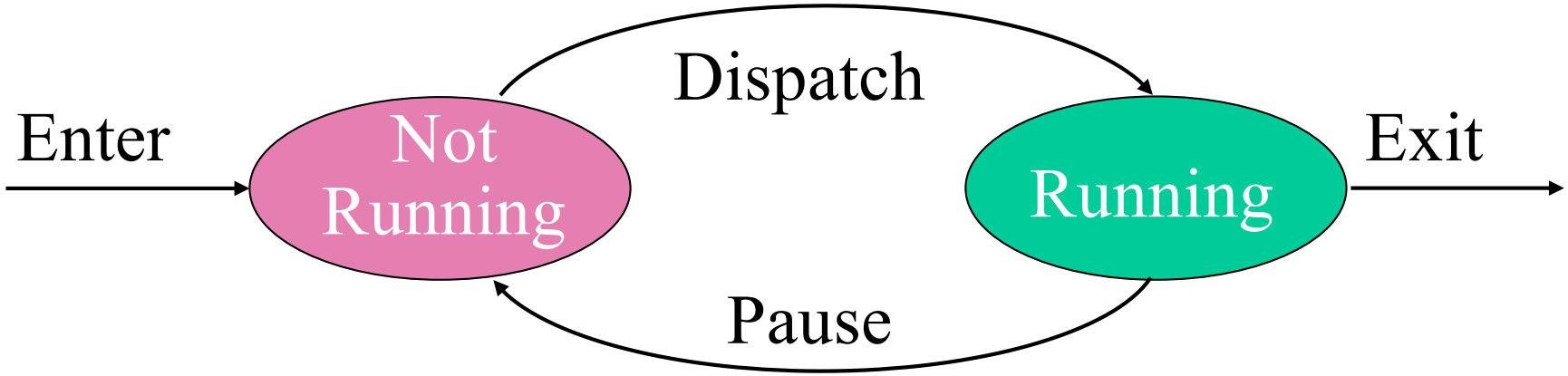
Trace

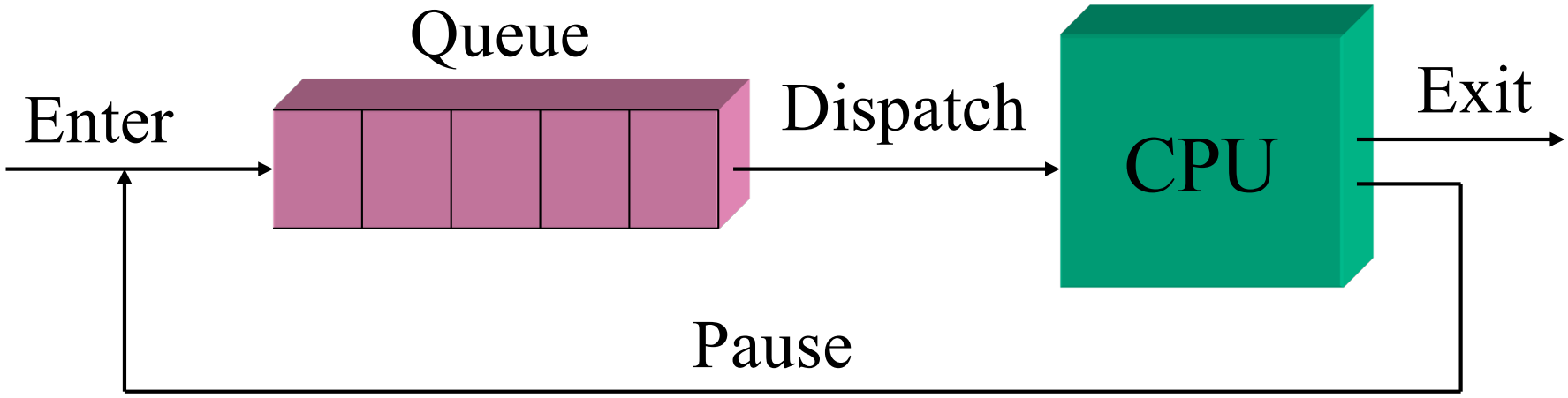
- Charakterisiert das Verhalten eines Prozesses
- Sequenz der Instruktionen, die für einen Prozess ausgeführt werden
- Überlappung von Traces verschiedener Prozesse charakterisieren das Prozessorverhalten

Traces


Prozesse
Dispatcher
A
B
C
a+0
a+1
a+2
a+3
a+4
a+5
a+6
a+7
a+8
a+9
b+0
b+1
b+2
(I/O)
c+0
c+1
c+2
c+3
c+4
c+5
c+6
d+0
d+1
d+2
d+3
T… Timeout
I/O… I/O Request
a+0
a+1
a+2
a+3
a+4
T
d+0
d+1
d+2
d+3
b+0
b+1
b+2
I/O
d+0
d+1
d+2
d+3
c+0
c+1
c+2
c+3
c+4
T
d+0
d+1
d+2
d+3
a+5
a+6
a+7
a+8
a+9
T
d+0
d+1
d+2
d+3
c+5
c+6
kombinierter Trace

Einfachstes Prozessmodell





BS und Prozesse

- BS kontrolliert die Ausführung der Prozesse (Ausführungsmuster, Ressourcenzuteilung)
- Repräsentation von Prozessen muss Aufgaben des BS unterstützen
  - Zustand des Prozesses
  - vom Prozess belegter Speicherbereich
  - ⇨ Verwaltung von Prozessen in Datenstrukturen (Queues) entsprechend ihren Zuständen

Erzeugung von Prozessen

BS baut Datenstrukturen auf und allokiert notwendigen Speicher.

Wann wird ein Prozess erzeugt?

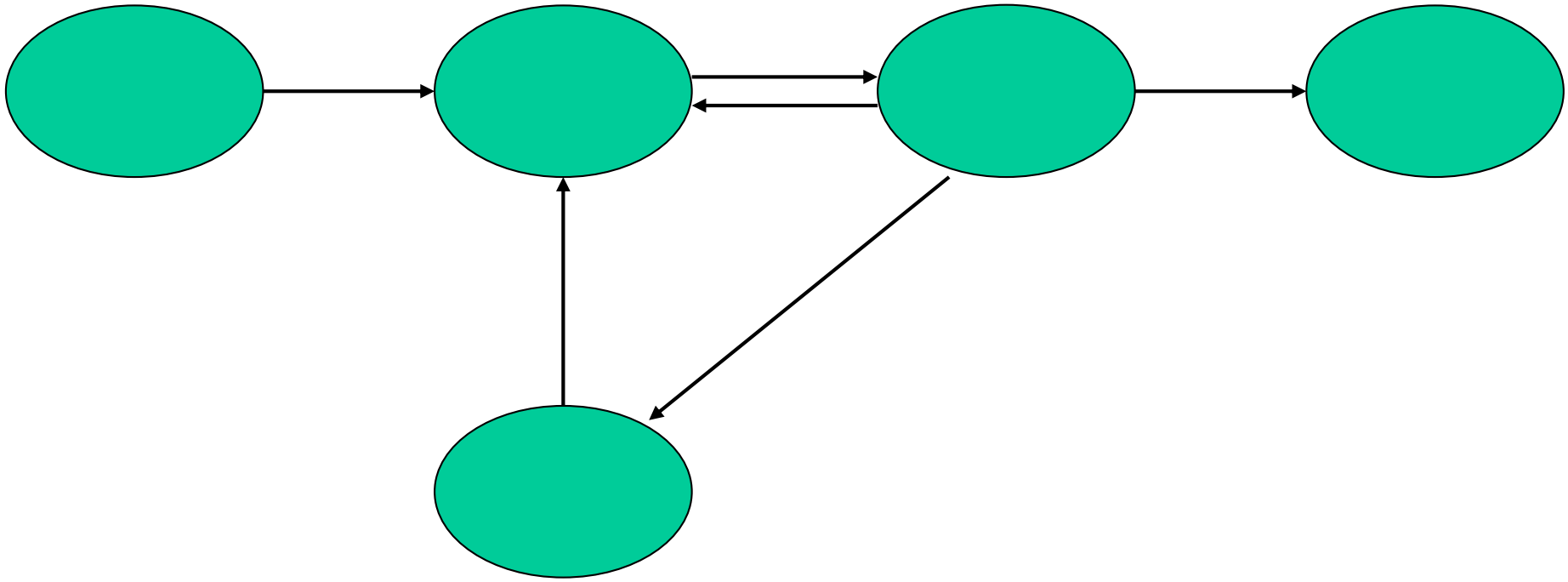
- Login eines interaktiven Benutzers
- BS: Ausführung eines Services
- Erzeugung durch einen Benutzerprozess (*Process Spawning*: *Parent* bzw. *Child*)
- Absetzen eines neuen Jobs

Beendigung von Prozessen

Prozess zeigt Beendigung an

- Logout durch den Benutzer
- Service Request an das BS
- Auftreten eines Fehlers bei der Abarbeitung eines Prozesses
- Halt-Instruktion eines Jobs

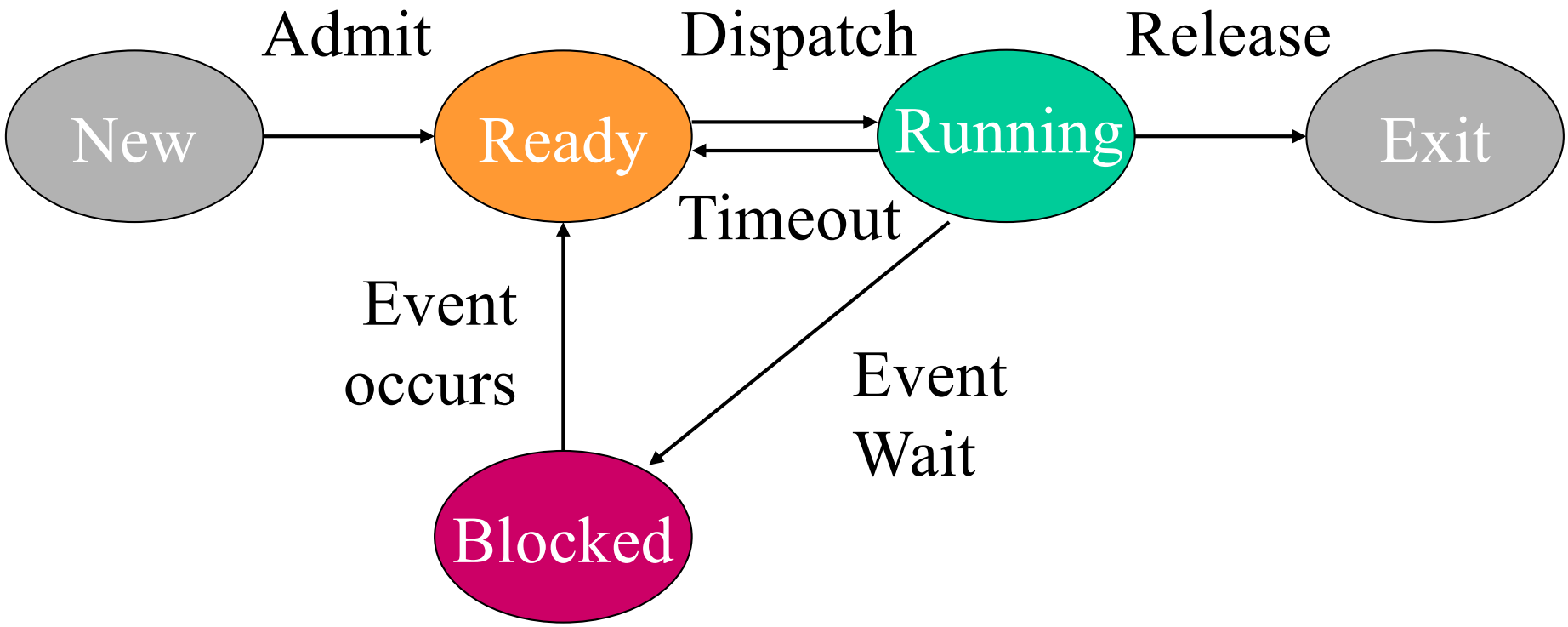
Prozesszustände



Wesentliche Prozesszustände

- *Running*
  - Prozess ist im Besitz der CPU und wird ausgeführt
- *Ready*
  - Prozess ist bereit zur Ausführung (wartet nur auf Zuteilung der CPU)
- *Blocked / Waiting*
  - Prozess wartet auf ein Ereignis (z.B. Beendigung von I/O), ist nicht laufbereit

5 Zustands-Prozessmodell



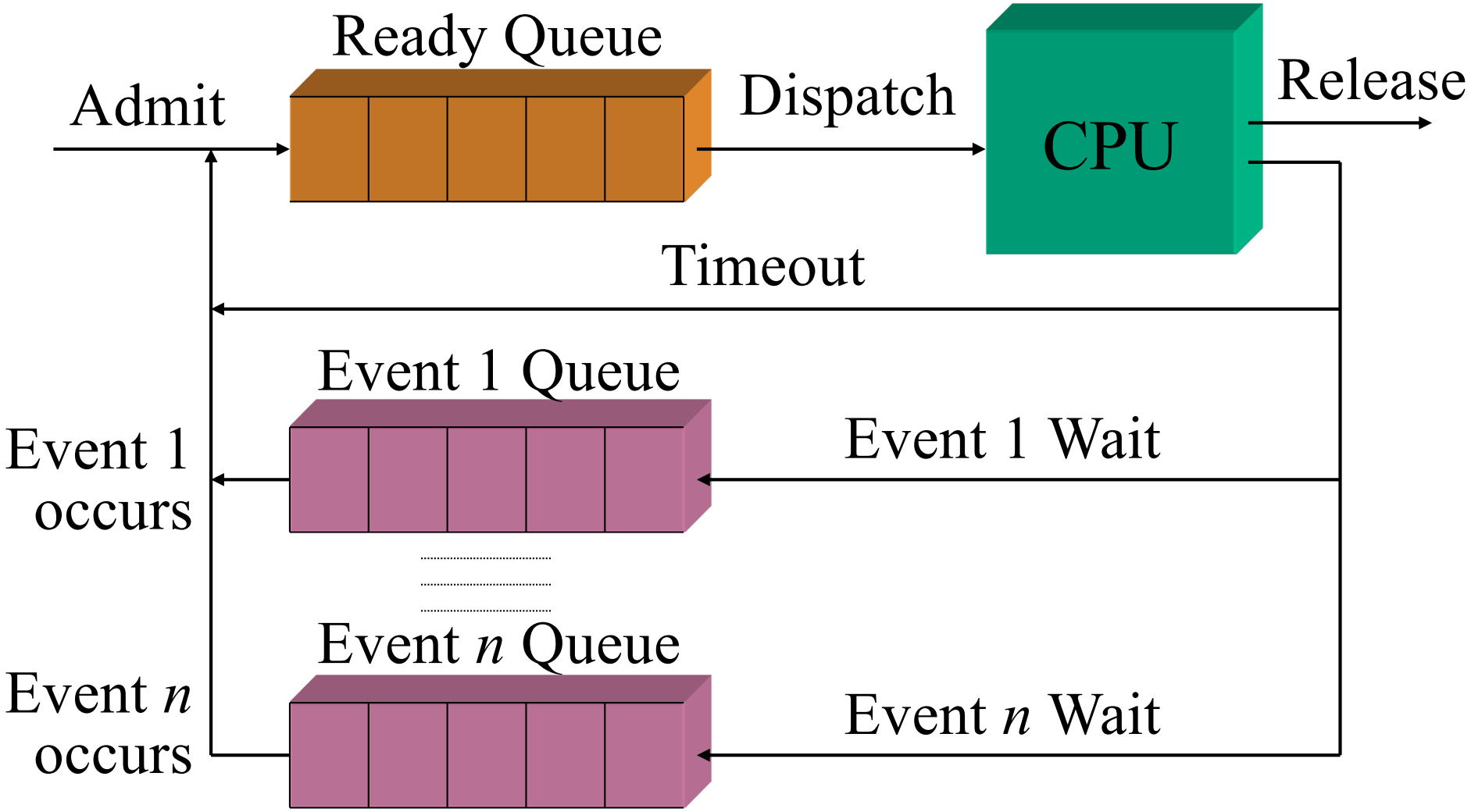
Wesentliche Prozesszustände

- *New*
  - BS hat den Prozess kreiert:
    - Prozessnummer (process identifier)
    - Tabellen und Tabelleneinträge zur Prozessverwaltung
  - der Prozess ist jedoch noch nicht bereit zur Ausführung
    - Vermeidung der Ressourcenüberlastung durch Zulassung zuvieler Prozesse

Wesentliche Prozesszustände

- *Exit*
  - Zustand wird durch Terminierung erreicht
  - Prozess wird nicht mehr weiter ausgeführt
  - Prozessinformationen (Tabellen) werden von Hilfsprogrammen verwendet (z.B. Accounting, Debugging)
  - Die Prozessinformationen, -tabellen werden gelöscht, wenn sie nicht mehr benötigt werden

Queuing Modell



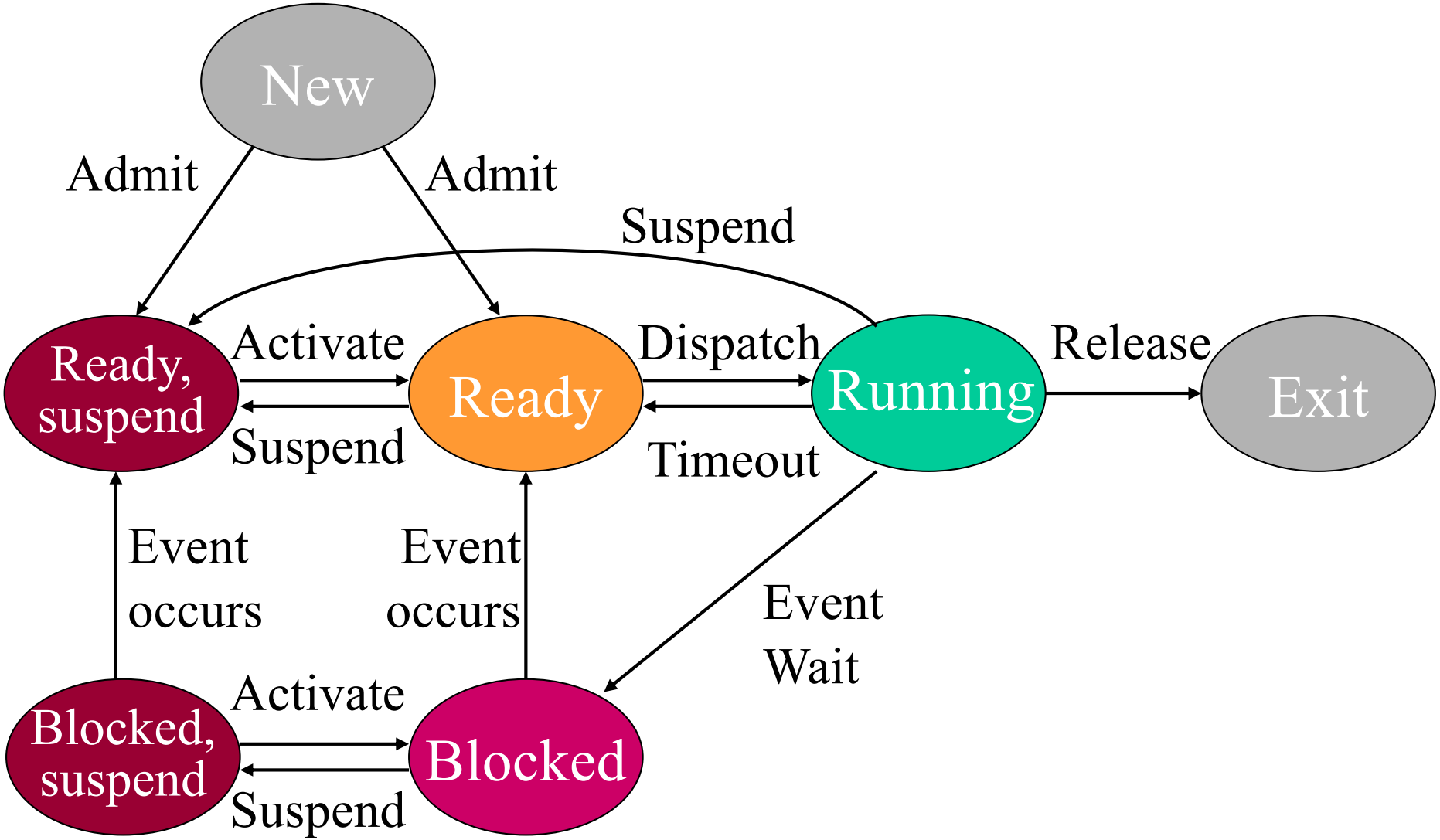
Process Switch

- Umschalten des aktiven Prozesses
- wenn das BS im Besitz der CPU ist:
  - Supervisor Call
    expliziter Aufruf durch das Programm (I/O, …)
  - Trap
    bedingt durch aktuelle Instruktion (z.B. Auftreten eines Fehlers)
  - Interrupt
    Ursache liegt außerhalb des Prozesses, Kontrolle geht an Interrupt Handler und BS

Swapping

- zuviele Prozesse im Hauptspeicher führen zu einer Verschlechterung der Performance
- Abhilfe: *Swapping* = Auslagern von Prozessen auf einen Sekundärspeicher
- zur Realisierung im BS: zwei neue Prozesszustände und Queues
  - *Ready, suspend*
  - *Blocked, suspend*

Prozesszustände mit Suspend



Ursachen für Suspend

- Swapping
- BS lagert Hintergrundprozess, Utility, oder problembehafteten Prozess aus
- Interaktiver Request (z.B. für Debugging)
- Timing: periodischer Prozess kann zwischen Aktivierungen ausgelagert werden
- Parent Request: Untersuchung, Modifikation, Koordination von Kindprozessen

Kontrollstrukturen für die Prozessverwaltung

Kontrollstrukturen des BS

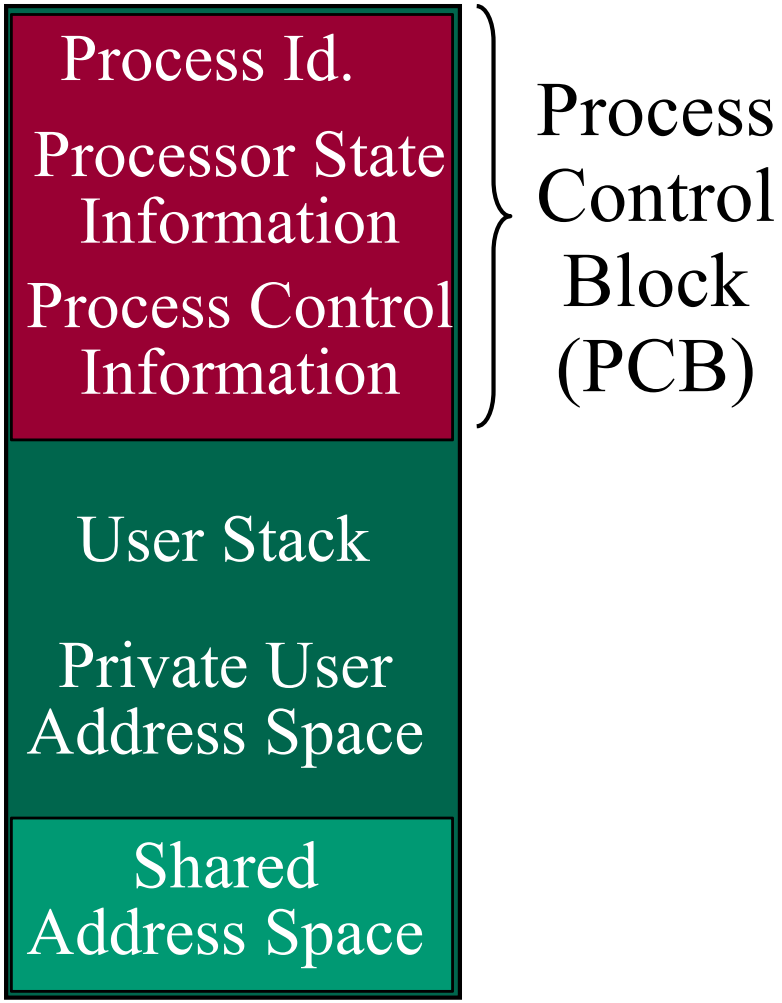
Das BS verwaltet die folgenden Tabellen für Prozesse und Ressourcen

- Memory Tables
- I/O Tables (für Geräte und Kanäle)
- File Tables
- *Process Tables*

Process Image

- User Program
- User Data
  - modifizierbarer Bereich des User Space (Daten, User Stack, modifizierbare Programme)
- System Stack
  - Parameter und Calling Addr. von System Calls
- Process Control Block (Execution Context)
  - Process identification, processor state information, process control information

Process Image



- befindet sich im Virtual Memory
  - muss nicht zusammen-hängend sein
- Primary Process Table enthält Verweis auf Process Image
- BS benötigt Teile des Images zur Prozess-verwaltung im Haupt-speicher

PCB: Process Identification

- Eindeutige Prozessnummer (Process identifier)
  - = Index in der Primary Process Table
- Benutzerkennung (User identifier)
  - – Benutzer, dem der Prozess gehört
- Nummer des Prozesses, der den Prozess generiert hat (Parent Process Identifier)

PCB: Processor State Information

- Registerinhalte
- Kontroll- und Statusregister
  - Befehlszähler
  - Program Status Word (PSW)
    - Kontroll-, Modusinformation, Status-Bits
- Stack Pointers

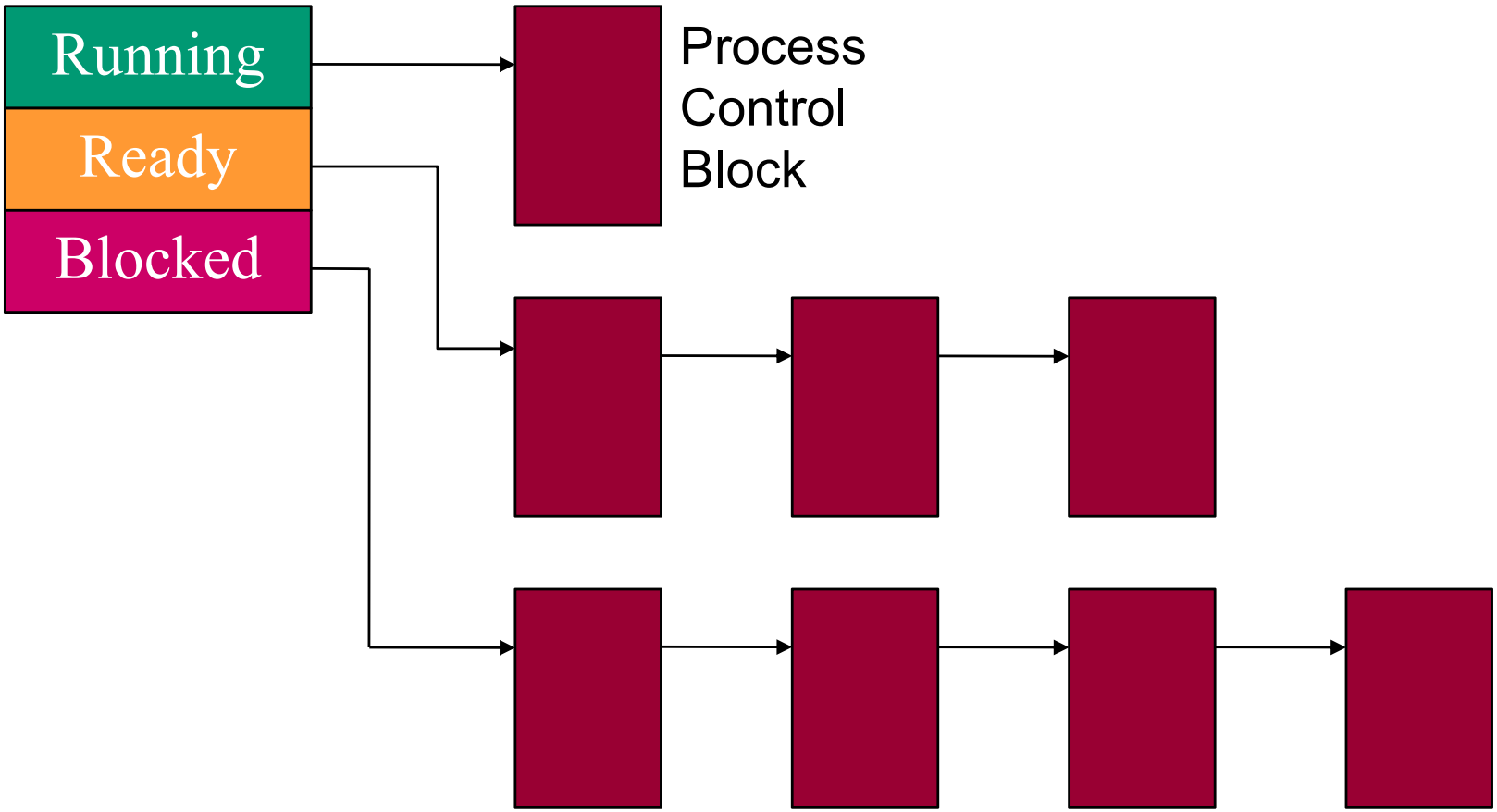
PCB: Process Control Information

- Scheduling und Zustandsinformation
  - Zustand, in dem sich der Prozess befindet
  - Priorität und Schedulinginformation
  - Ereignis, auf das der Prozess wartet
- Querverweise auf andere Prozesse
  - Realisierung von Prozess-Queues
  - Verweis auf Parent, Child, …

PCB: Process Control Information

- Interprozesskommunikation (IPC)
  - Flags, Signale, Verweise auf Nachrichten
- Privileges
- Memory Management
  - Verweise auf Segment- oder Seitentabellen
- Ressourcen
  - verwendete: geöffnete Files, I/O Devices
  - bisher konsumierte: CPU Zeit, I/O, etc.

Prozesslisten

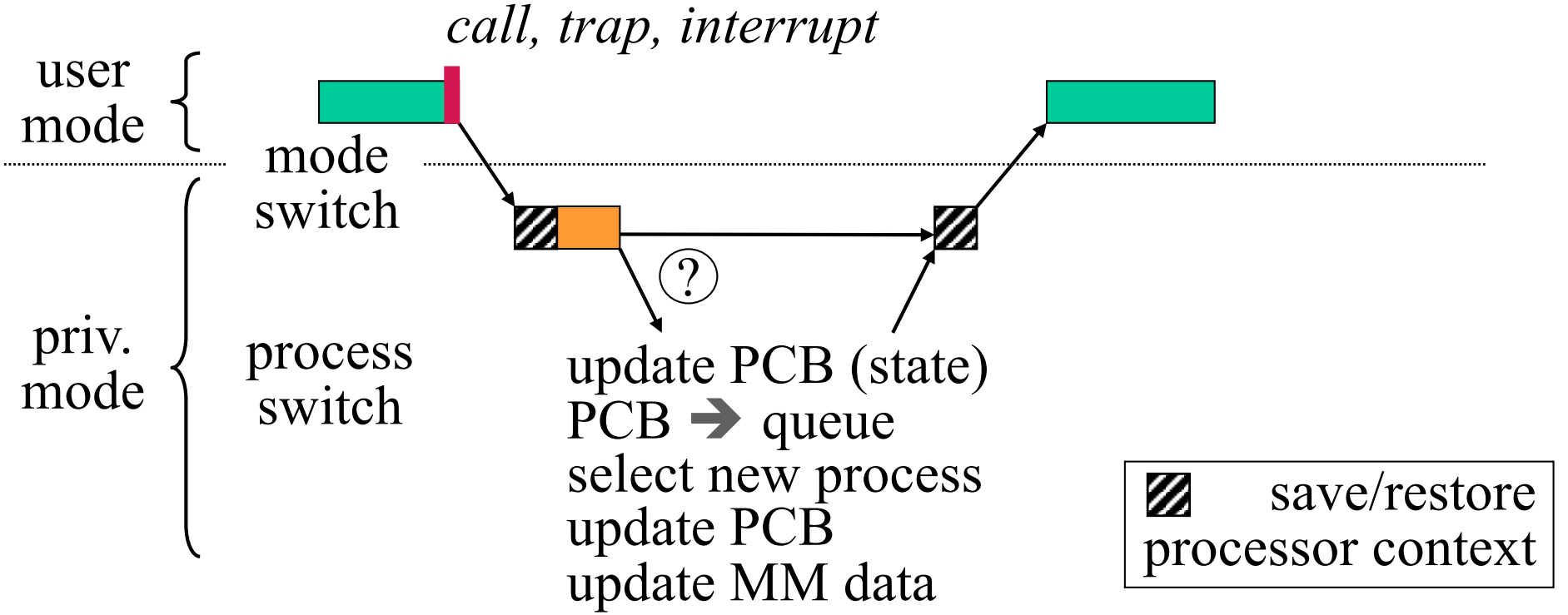


Execution Modes

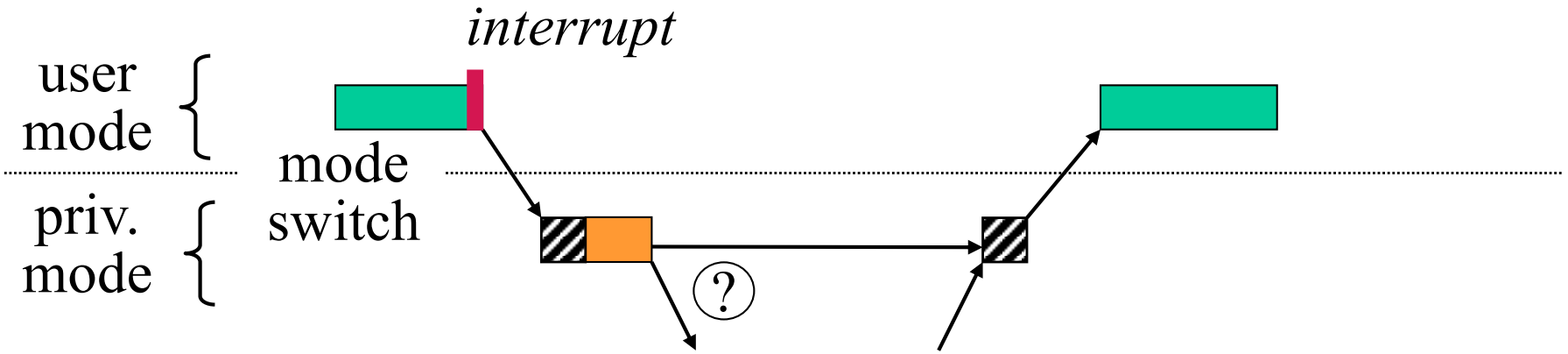
- Um die Datenstrukturen (z.B. PCB) des BS zu schützen, gibt es mindestens zwei *Execution Modes*
  - Privileged Mode (= system mode, kernel mode, supervisor mode, control mode)
    für Zugriff auf Kontrollregister, MM, I/O-Primitive
  - User Mode
- *Mode Switch* in BS-Routinen
- Execution Modes werden durch Mode-Bits der CPU unterstützt

Mode Switch vs. Process Switch

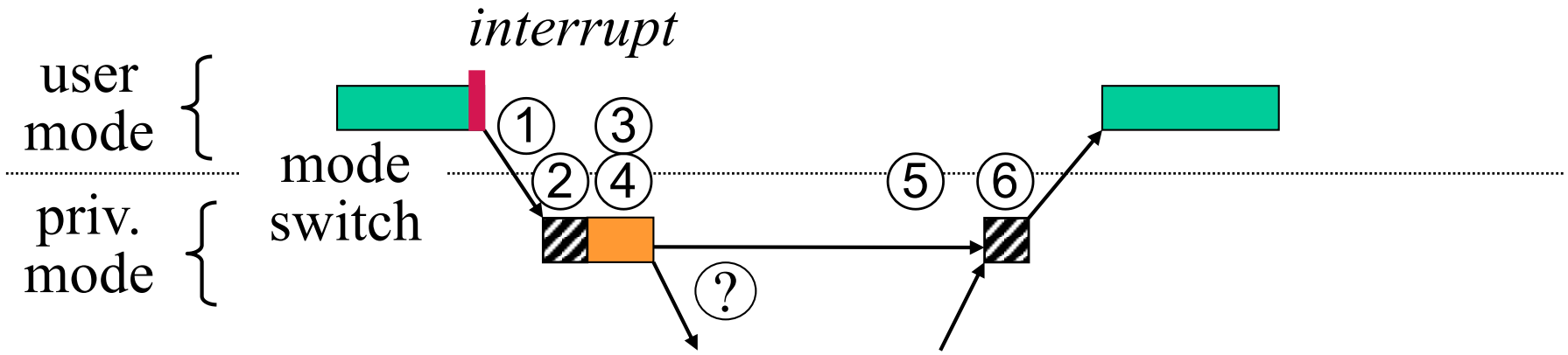
- Mode Switch ist Basis für Process Switch
- nicht jeder Mode Switch bewirkt Process Switch



Example: Interrupt Handling



Example: Interrupt Handling



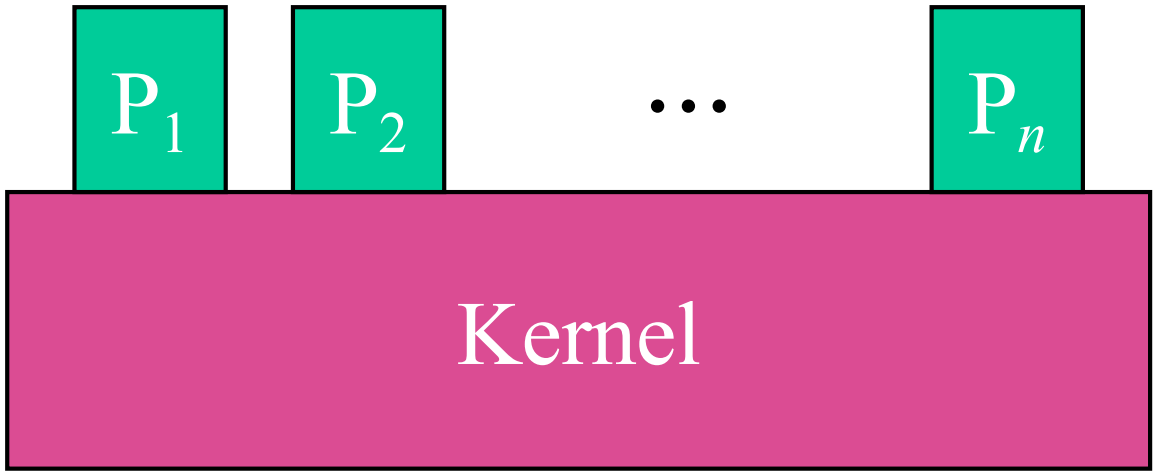
1. PC → stack, ...; load PC from interrupt vector [HW]
2. Save registers, setup new stack [assembly code]
3. Interrupt service, e.g., read/buffer inputs [C code]
4. Scheduler decides on next process [C code]
5. Return to assembly code [C code]
6. Setup for process continuation [assembly code]

Prozesse im Betriebssystem

BS und Prozesse

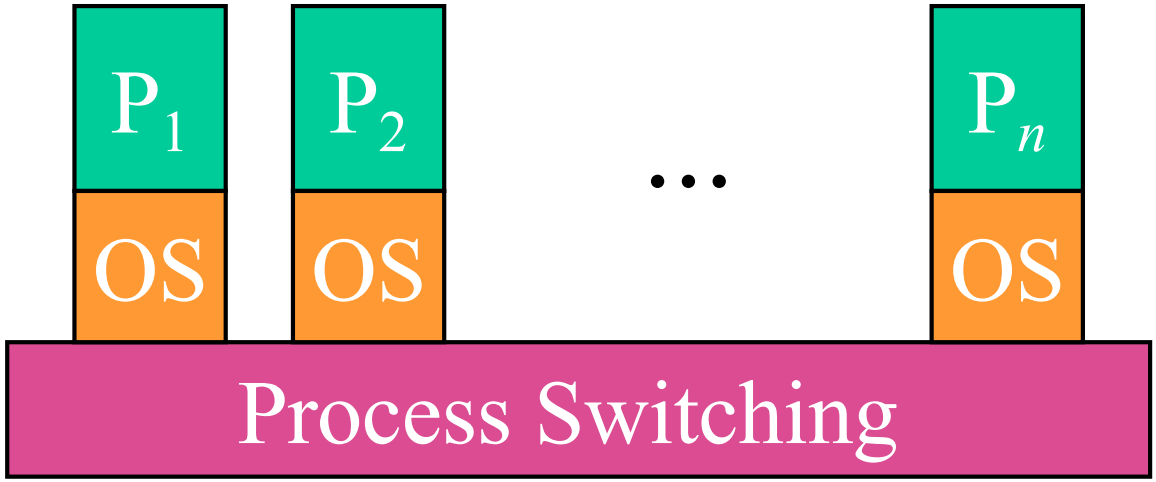
- Verschiedene Möglichkeiten, ein BS zu implementieren
  - Strikte Trennung von Kernel und Prozessen
  - BS exekutiert innerhalb von Benutzerprozessen
  - Prozessbasiertes BS

Nonprocess Kernel



- Prozessbegriff nur für Benutzerprogramme
- Verlassen des Prozesskontexts bei BS-Aktivität
- BS arbeitet von Prozessen getrennt im Privileged Mode (eigener Speicherbereich, eigener Stack für BS)

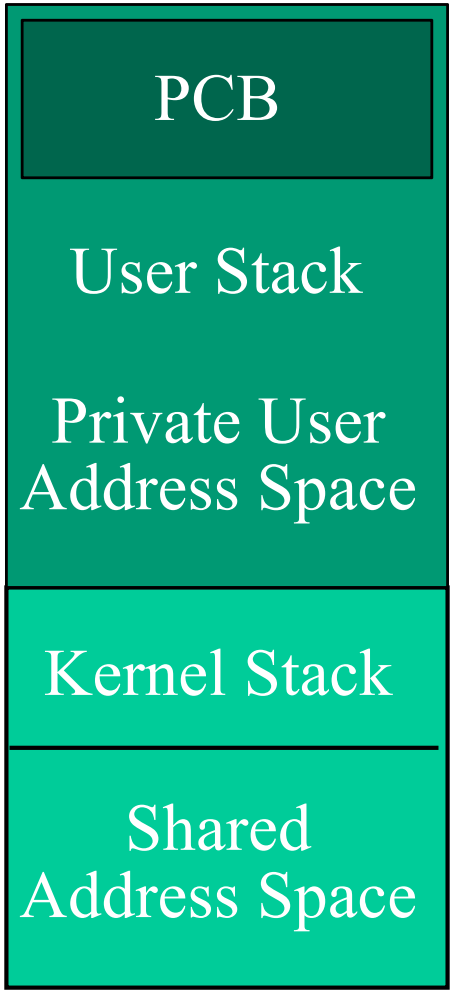
BS-Ausführung in User-Prozessen



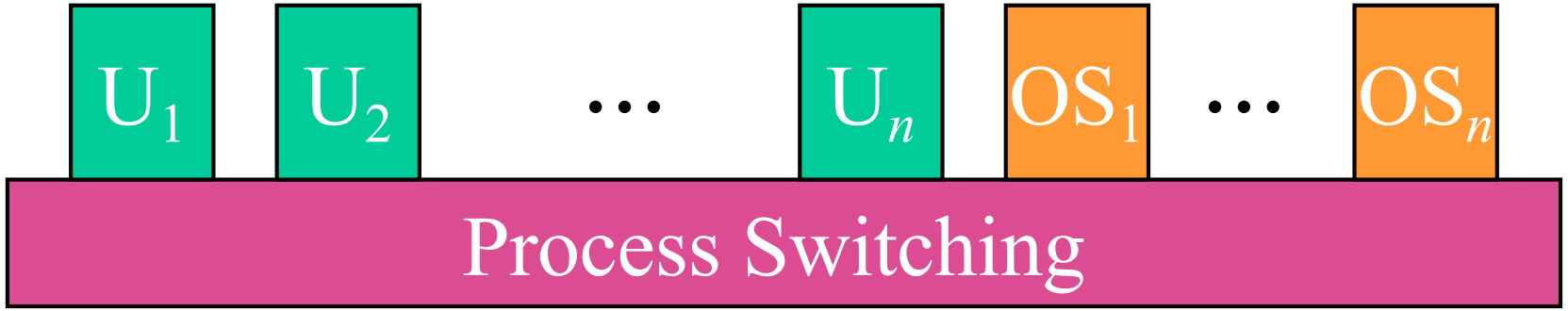
- BS ist eine Sammlung von Routinen, die vom Benutzerprogramm aufgerufen werden
- Fast alle BS-Routinen laufen im Prozesskontext
- Verlassen des Prozesskontexts nur bei Process Switch

BS-Ausführung in User-Prozessen

- BS Code und Daten werden im Shared Address Space abgelegt
- Getrennter Kernel Stack für Kernel Mode
- Im Prozess laufen sowohl User-Programm als auch BS-Routinen
  ➔ Programm vs. Prozess



Prozessbasiertes BS



- BS ist eine Sammlung von Systemprozessen
- Nur Basisservices (Process Switching, einfaches Memory Management, IPC, Interrupts und I/O) sind nicht als Prozesse realisiert

Kernel-Architekturen

Monolithic Operating System

- Betriebssystem = Menge von Prozeduren
- Jede Prozedur kann jede andere Prozedur aufrufen

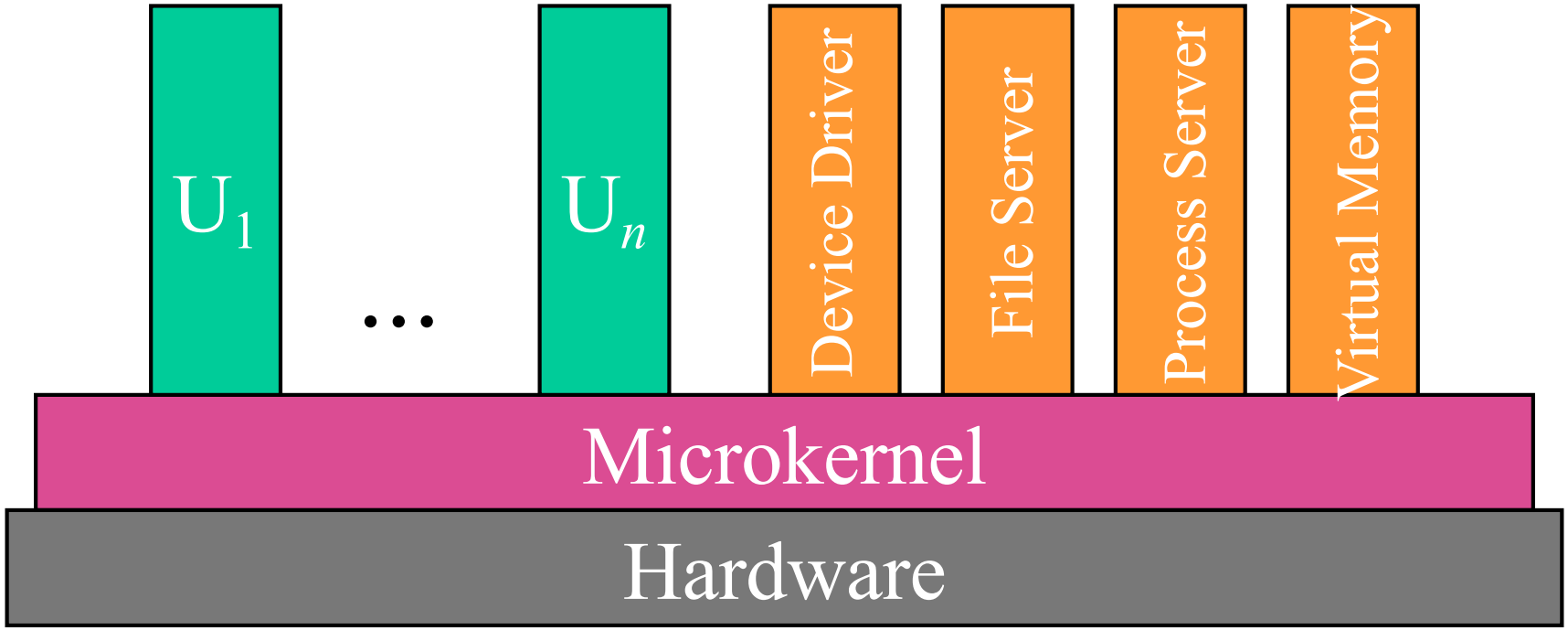
Layered Operating System

- Hierarchische Organisation der BS-Funktionen
- Interaktionen zwischen benachbarten Schichten
- Mehrheit der Schichten exekutieren im Kernel Mode

Modular Operating System

- Hardware Abstraction Layer
- Module realisieren verschiedene OS-Funktionalitäten

Microkernel Architektur



- Basisservices im Kernel
- nicht zentrale BS Services als Server Prozesse
- Nachrichtenkommunikation zw. BS/Prozessen

Microkernel Architektur

Microkernel Services

- Process Switching
- Basic Memory Management
- Interrupts und Hardware Access (I/O)
- Nachrichtenaustausch und –kontrolle

Eigenschaften

- Einheitliche Interfaces
- Flexibilität und Erweiterbarkeit
- Portabilität
- Unterstützung von Verteilung
- Kernelgröße: 300KB, 140 Sys. Calls (1st generation)
  12KB, 7 Sys. Calls (L4, 2nd generation)

Zusammenfassung Prozesse

- Prozess ist ein zentrales Konzept in BS
- BS kreiert, verwaltet und beendet Prozesse
- Prozess durchläuft verschiedene Zustände (Ready, Running, Blocked, Suspend, ...)
- Datenstrukturen zur Prozessverwaltung
  - Prozesstabelle
  - Process Image: belegter Adressbereich, PCB: ID, Zustand, Steuerinfo.; Ressourcen, Priorität, etc.

Zusammenfassung Prozesse

- Mode Switch zwischen User/Kernel Mode
  - Interrupt, Trap oder Supervisor Call
- Execution Mode versus Prozess/BS Kontext
- Unterschiedliche Ansätze der Implementierung von BS und Prozessen
  - strikte Trennung BS – Prozesse
  - BS-Funktionen in User-Prozessen abgearbeitet
  - Server-Prozesse zur Ausführung von BS-Funktionen

Threads

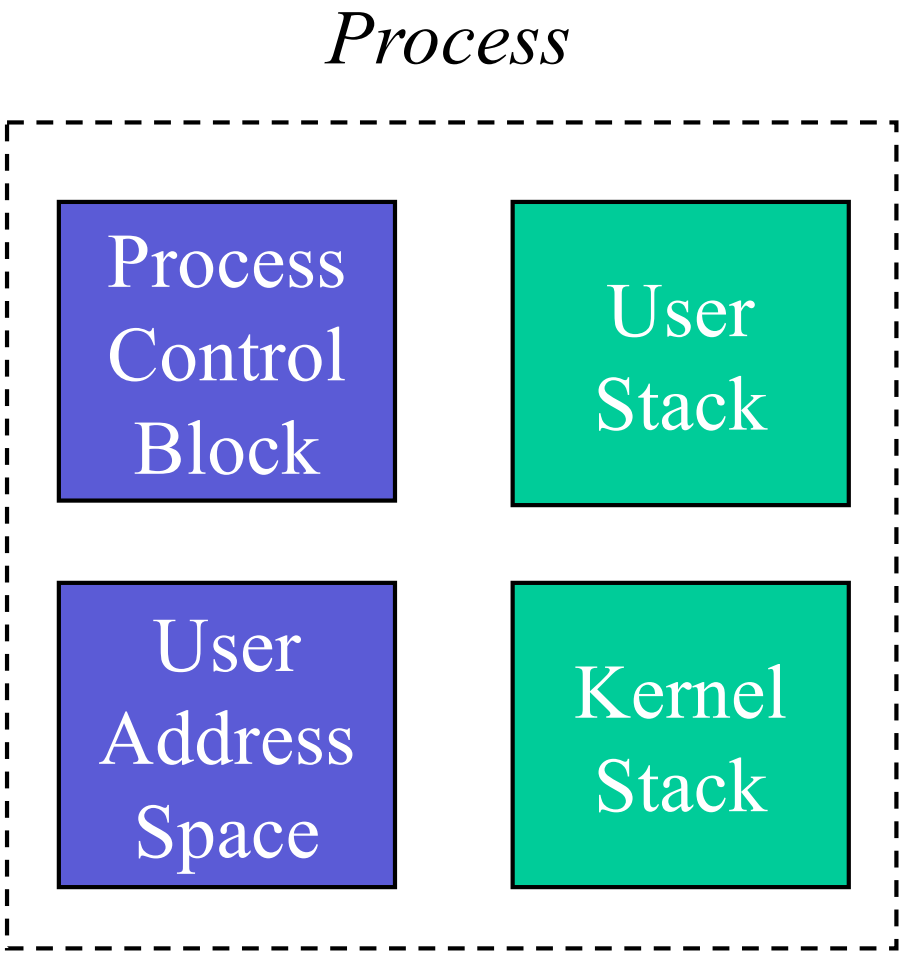
Motivation Threads

- Bisher betrachtete Prozesse bilden Einheit für
  1. Ressourcenverwaltung
  2. Dispatching (kurzfristiges Scheduling)
- Entkopplung von (1) und (2):
  - *Process (Task)*: Einheit der Ressourcenverwaltung
  - *Thread (Lightweight Process):* Einheit für das Dispatching
- *Multithreading*: *n* > 1 Threads pro Prozess

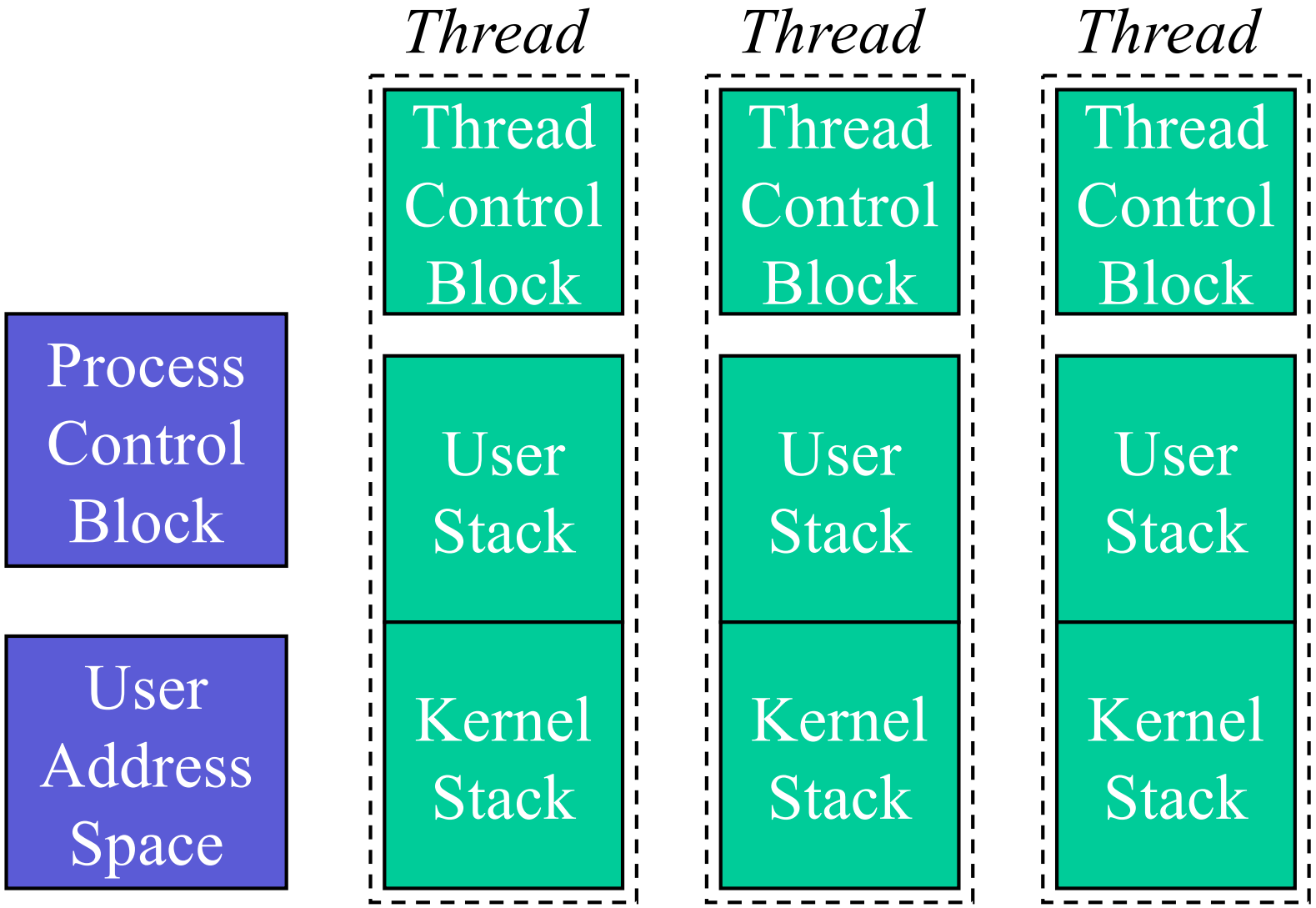
Prozesse und Threads

- Process
  - virtueller Adressraum mit Process Image
  - Speicherschutz, Files, I/O Ressourcen
- Thread
  - Ausführungszustand (Running, Ready, …)
  - Kontext (wenn nicht gerade laufend)
  - Stack
  - thread-lokale statische und lokale Variable
  - Zugriff auf Prozessspeicher und Ressourcen

Singlethreaded Process Model



Multithreaded Process Model



Vorteile von Thread vs. Prozess

- Thread-Erzeugung benötigt weniger Zeit
- Umschaltung zwischen Threads geht schneller als ein Process Switch
- Terminierung eines Threads benötigt weniger Zeit als Prozessterminierung
- Kommunikation zwischen Theads eines Prozesses ohne Einschaltung des Kernels, aber: Synchronisation notwendig!!!

Einsatzbereiche von Threads

Applikationen, die zusammengehörige Menge von Abarbeitungseinheiten bilden

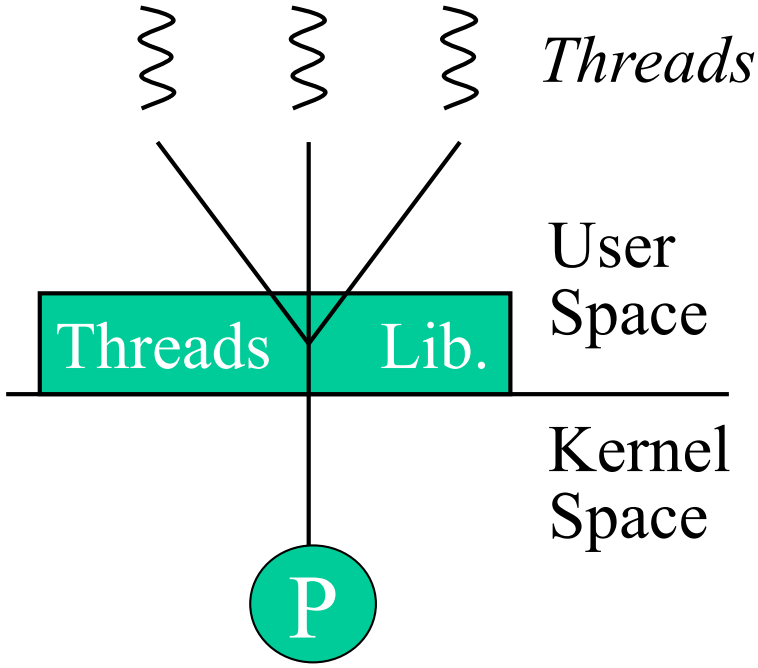
- Bsp: File Server in LAN
  - mehrere Requests in kurzer Folge
  - ein Thread für jeden Request
- Bsp: Spreadsheet-Programm
  - ein Thread zeigt Menüs an und liest Inputs
  - ein Thread führt Berechnungen und Updates aus

Thread-Zustände

- Wichtige Zustände: *Running, Ready, Blocked*
- Zustand *Suspend* existiert nicht für einzelne Threads - alle Threads eines Prozesses haben Zugriff auf den selben Adressraum
- Bei der Terminierung eines Prozesses terminieren alle zugehörigen Threads
- Was bedeutet das Blockieren eines Threads? Blockiert der Thread oder der ganze Prozess?

User-Level Threads (ULT)

- Threads sind für den Kernel unsichtbar
- Thread Management mittels Thread Library
- Thread Switching im User Mode (benötigt keinen Mode Switch)
- applikationsspezifisches Scheduling

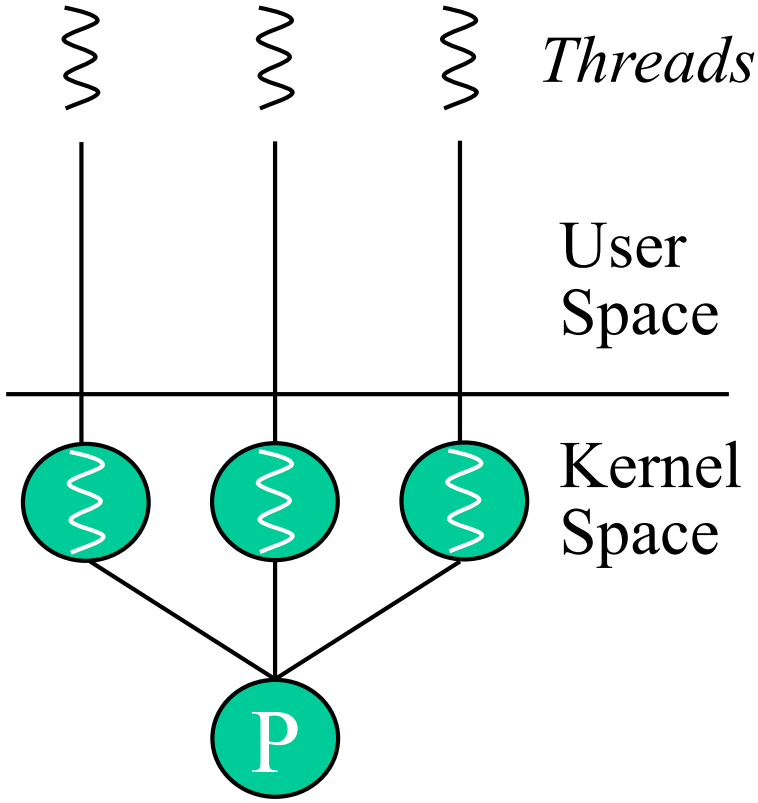


Threads Library

- enthält Code für
  - Erzeugung und Terminierung von Threads
  - Daten-, Nachrichtenaustausch zwischen Threads
  - Thread Scheduling
  - Sichern und Herstellen von Thread Kontexten
- Blockierender System Call blockiert *alle* ULTs eines Prozesses
- keine Verteilung auf mehrere Prozessoren

Kernel-Level Threads (KLT)

- Thread Management durch den Kernel
- Kernel Thread API, keine Library
- Thread Switching durch den Kernel
- Scheduling auf Thread-Basis



Kernel-Level Threads

- Thread-weises Blocking
- Kernel-Routinen multi-threaded
- gleichzeitiges Schedulen mehrerer Threads eines Prozesses (bei mehreren Prozessoren)
- Thread Switching innerhalb eines Prozesses über den Kernel benötigt 2 Mode Switches
  ➔ Verlangsamung gegenüber ULTs

➪ z.B. kombinierter (hybrider) ULT/KLT Ansatz

Zusammenfassung Threads

- Thread: Einheit für das Dispatching
- mehrere Threads pro Prozess möglich
- schnelleres Erzeugen und Umschalten als bei Prozessen
- verschiedene Implementierungen
  - user-level Threads
  - kernel-level Threads