Prozesse I - Grundlagen

Prozesse

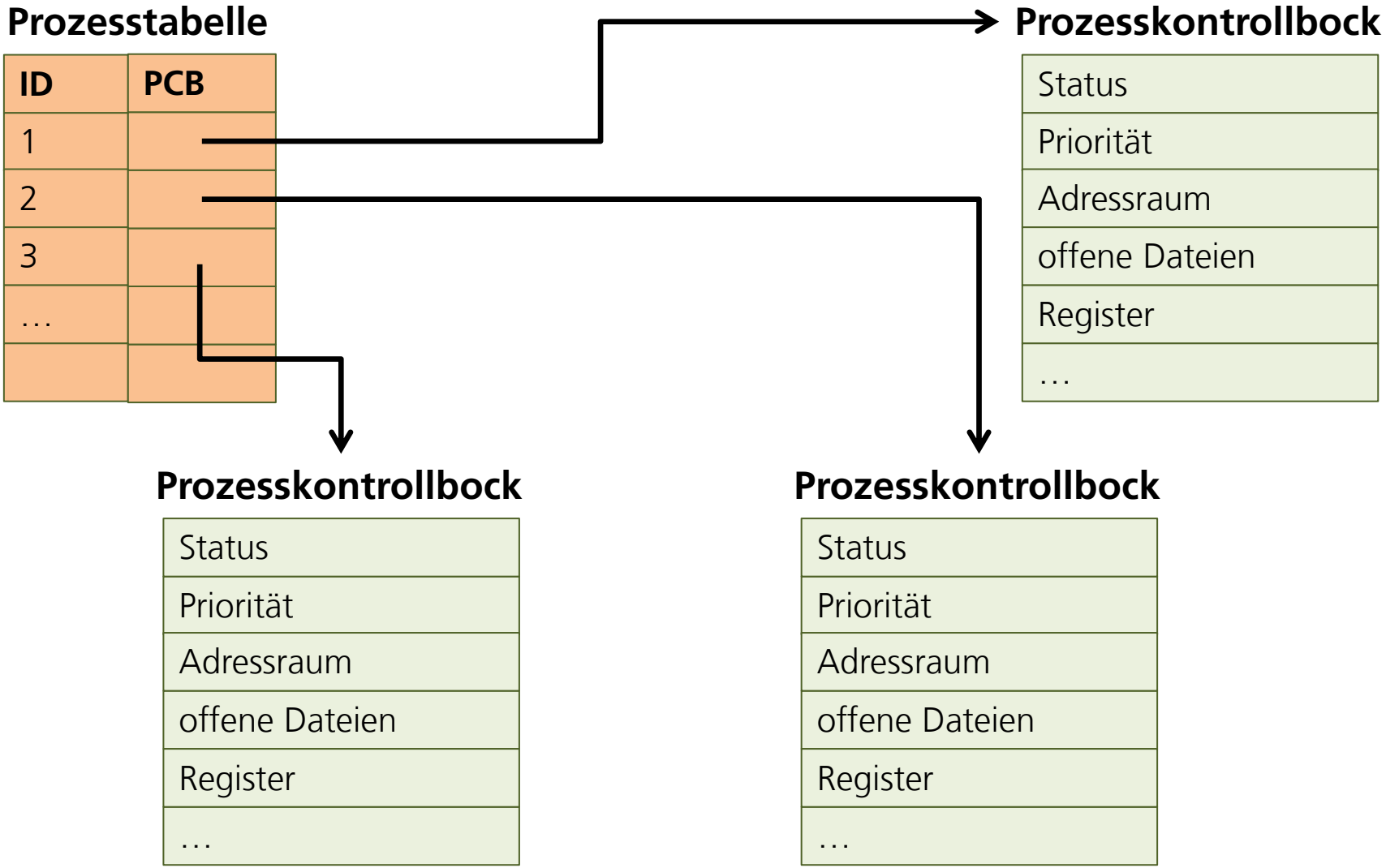
- Ein **Prozess** ist ein eindeutig identifizierbarer *Ablauf eines Programms*, d.h. ein Programm in Ausführung mit allen dazu notwendigen Daten*).
- Das Betriebssystem verwaltet eine Tabelle über alle Prozesse, die **Prozesstabelle**.
- Jeder **Eintrag** ist genau einem **Prozess** zugeordnet.
- Ein Eintrag wird als **Prozesskontrollblock** bezeichnet (fehlerhafte Übersetzung von *Process Control Block* = Prozess**steuerungs**datensatz), kurz *PCB*.

*) Auf diese Definition haben wir bereits im Grundlagenkapitel zurückgegriffen

Datenstrukturen zur Prozessverwaltung

- Der PCB enthält alle relevanten Daten zur Verwaltung des Prozesses, also
  - Prozessidentifikation und Prozessstatus
  - Prozessrechte (Privilegien)
  - Prozessstrukturinformationen (Elternprozesse, Kindprozesse, etc.)
  - Informationen zu Kommunikation mit anderen Prozessen
  - ggf. zusätzliche Daten wie Laufzeit, zugewiesene Bearbeitungszeiten, etc.

Prozesstabelle und Prozesskontrollblöcke



Datenstrukturen zur Prozessverwaltung

- Als **Prozesskontext** wird derjenige Teil der Prozessdaten bezeichnet, der beim Umschalten der CPU auf einen anderen Prozess gesichert werden muss, um ihn später transparent fortführen zu können. Dazu gehören
  - aktuelle Werte der Register*) der CPU
  - Referenz auf aktuell verwendeten Speicher
  - ggf. weitere Steuerungsdaten
- Der Prozesskontext ist Teil des Prozesskontrollblocks.

*) Als Register werden die sehr kleinen (aktuell meist 64 Bit) Speichereinheiten der CPU bezeichnet, auf denen die Befehle der CPU letztlich operieren.

Prozesserzeugung

Prozesse werden stets durch andere Prozesse erzeugt.

Ausnahme: Beim Einschalten des Rechners wird ein Prozess aus dem Nur-Lese-Speicher (ROM) gestartet, der für den Startvorgang zuständig ist und im Zuge dessen weitere Prozesse startet.

Prozesserzeugung wird initialisiert

1. durch Benutzeranfrage (Applikation starten)
2. automatisch durch bereits laufenden Prozess
3. durch Kontrollprozess eines Batch-Jobs mit mehreren Teilaufgaben

Prozesstermination

Ereignisse, die zur **Beendigung** eines Prozesses führen, sind

1. Prozessaufgabe ist ausgeführt, Prozess führt Systemruf zur Beendigung aus (bspw. *ExitProcess* in Windows) (*freiwillig*)

2. Prozess stellt einen Fehler fest und beendet als Konsequenz seine Ausführung (*freiwillig*)

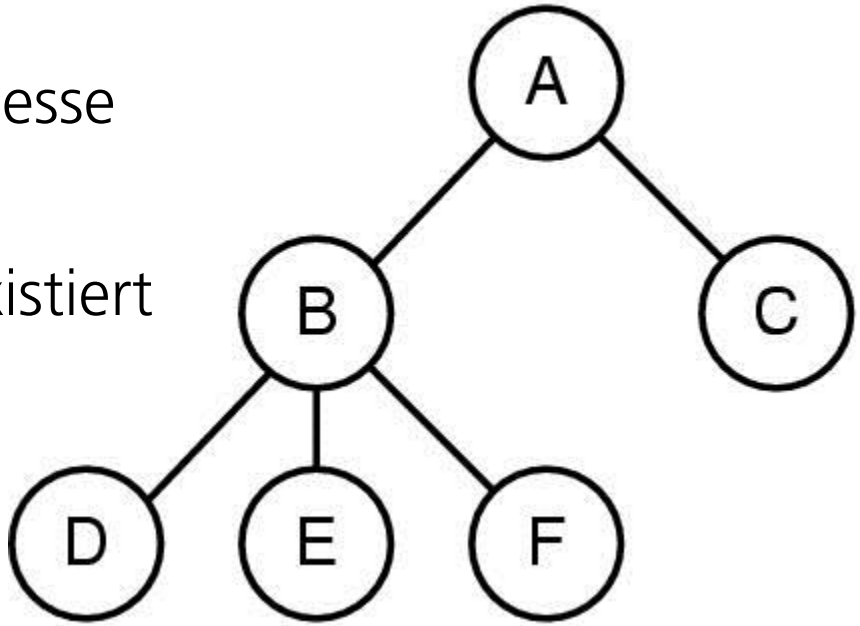
3. Schwerwiegender Fehler löst Unterbrechung aus, Betriebssystem entscheidet auf Beendigung (*unfreiwillig*)

4. Beendigungsaufforderung durch einen anderen Prozess, bspw. durch Elternprozess oder Nutzeranfrage (*unfreiwillig*)

Kindprozesse und Prozessbaum

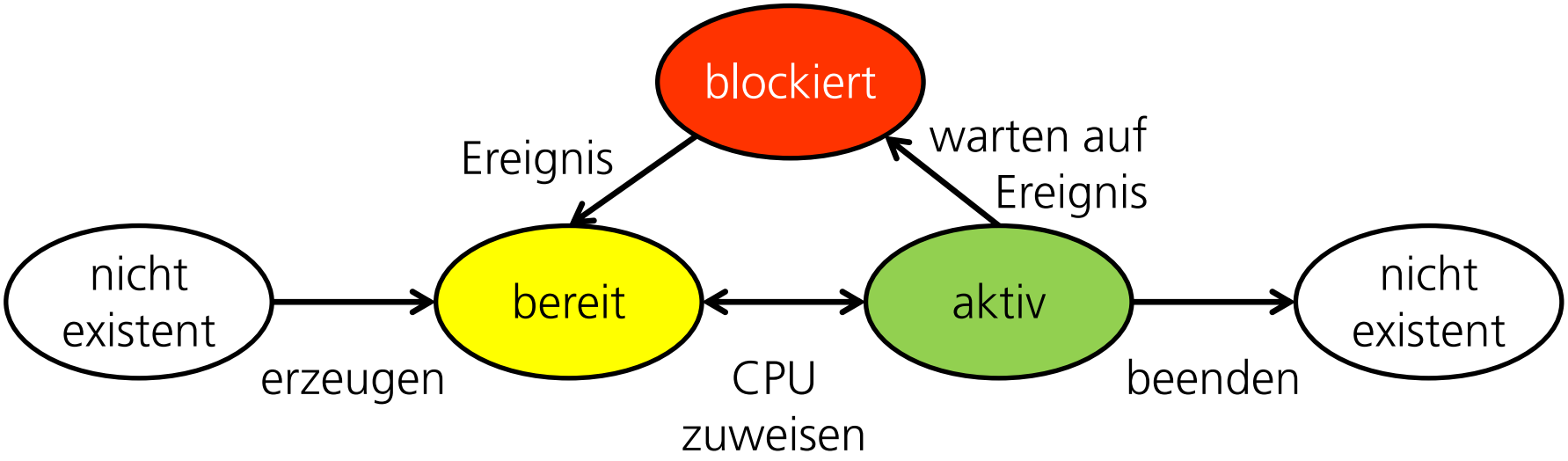
Zur Erinnerung:

- Prozesse können selbst neue Prozesse erzeugen (Kindprozesse).
- Beim Start des Betriebssystems existiert nur ein einziger Prozess (*root process*).
- Dieser erzeugt schrittweise Kindprozesse, die wiederum selbst Kindprozesse erzeugen können, es entsteht ein Prozessbaum*).



*) Windows kennt keine solche Prozesshierarchie

Prozesszustände – Grundvariante



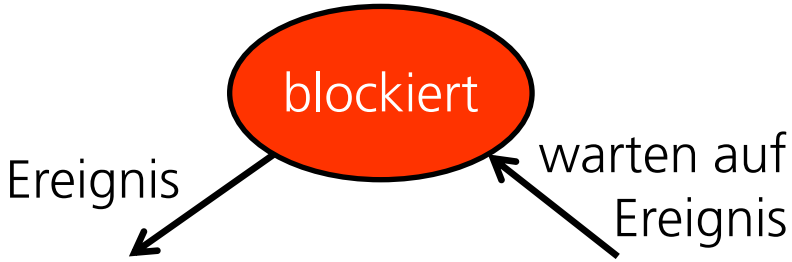
Zustände existierender Prozesse:

bereit: kann rechnen
wird bei Prozessorzuteilung berücksichtigt

aktiv: Prozess rechnet

blockiert: kann nicht rechnen, wartet auf Ereignis
wird bei Prozessorzuteilung nicht berücksichtigt

Prozesszustände



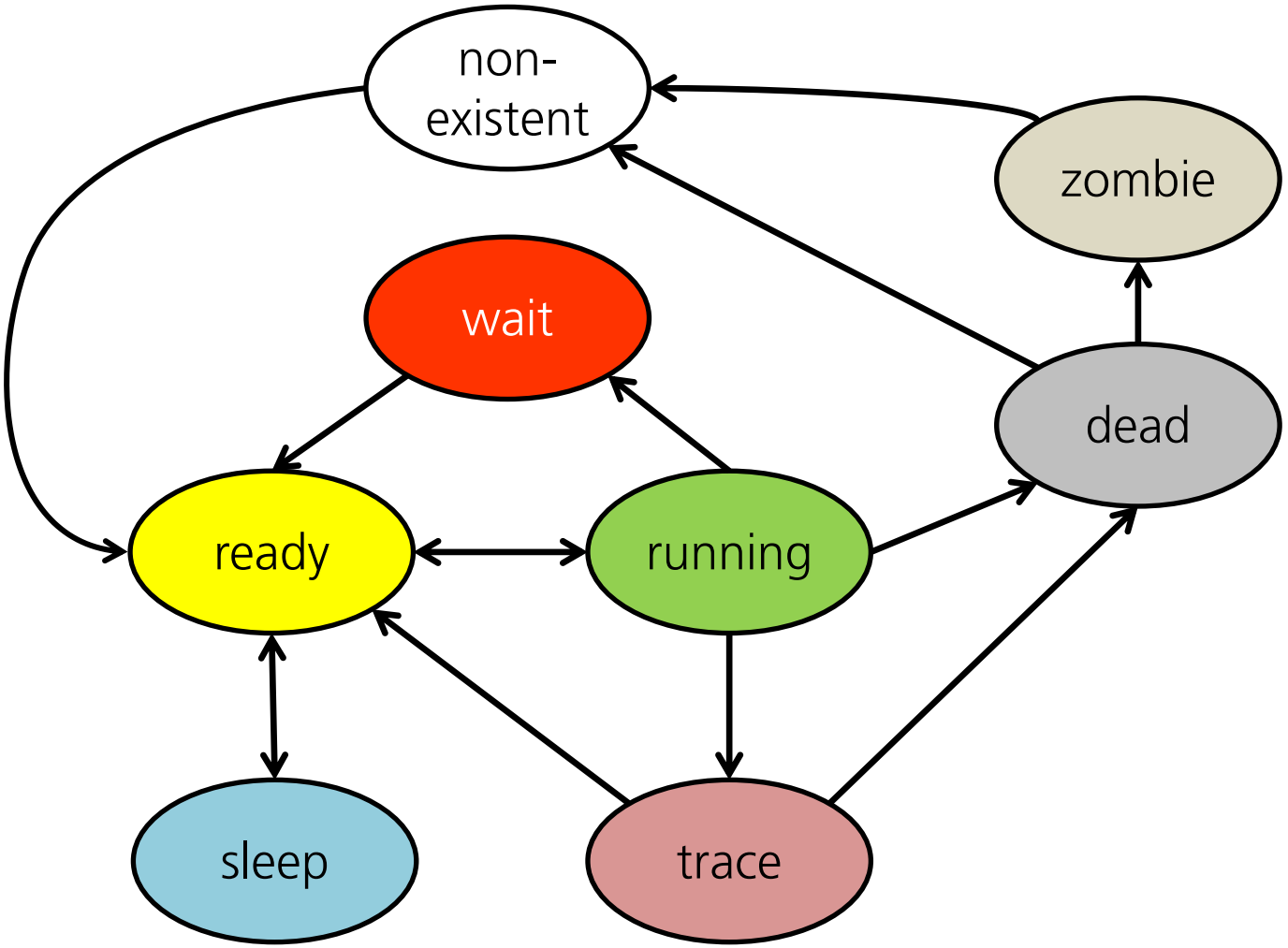
Typische **Blockadegründe**:

1. Ein durch den Prozess ausgelöster Systemruf stellt fest, dass eine Weiterverarbeitung erst bei Eintritt eines externen Ereignisses möglich ist.

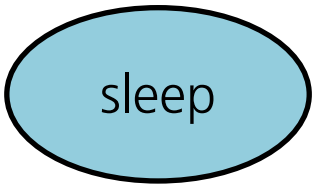
2. Der Prozess versucht, auf nicht eingelagerten Speicher zuzugreifen, der zunächst eingelagert werden muss.

In beiden Fällen wird der Prozess durch das **Betriebssystem** als **blockiert** markiert und erhält den Prozessor nicht.

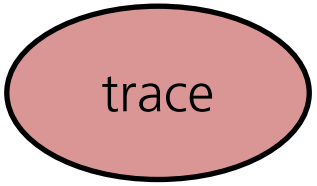
Erweiterte Prozesszustände – Beispiel Unix



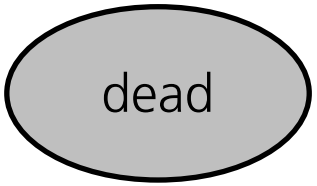
Erweiterte Prozesszustände – Beispiel Unix



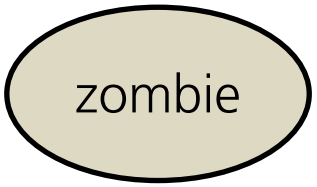
Prozess **wartet** auf **eigenen Wunsch**, kann auf Benachrichtigungen reagieren



Prozess wurde durch **Debugger** unterbrochen



Prozess ist **beendet**, belegt aber noch Speicher



Prozess ist beendet und aus dem Speicher entfernt, hat aber noch einen Eintrag in den **Verwaltungsstrukturen**

Weitere denkbare Prozesszustände

- **Ausgelagert und bereit:** Ein an sich zum Rechnen bereiter Prozess befindet sich nicht im Hauptspeicher, sondern ist auf einen Datenträger ausgelagert.
- **Ausgelagert und blockiert:** Ein blockierter Prozess befindet sich nicht im Hauptspeicher, sondern ist auf einen Datenträger ausgelagert.

Prozesswechsel

Ursachen für Prozesswechsel:

- Hardwareunterbrechungen (Zeitscheibe abgelaufen, Signal von Controller)
- Softwareunterbrechungen (Exception, Trap oder Systemaufruf)
- Freiwillige Abgabe der Kontrolle durch den Prozess (letztlich auch ein Systemaufruf)

→ Dies ist stets mit einem Wechsel in den **Kernel-Mode** verbunden, d.h. das Betriebssystem übernimmt die Kontrolle! Der **Dispatcher** (Umschalter, Verteiler) als Teil des Kernels ist zuständig für Wechsel des zur Zeit aktiven Prozesses.

Prozesswechsel

Aufgaben des **Dispatchers** beim Prozesswechsel:

1. **Sicherung** des Prozesskontextes
2. Eintragen des **Zustandes** in PCB
3. **Einreihung** des Prozesses in vorgesehene Warteschlange (bereit, blockiert, etc.)

} Aktueller Prozess

4. Für **nächsten** Prozess
    1. **Kontext** aus PCB auf CPU übertragen (restore)
    2. **PCB** aktualisieren
    3. Rückschalten in **User-Mode** und aktivieren

} Nächster Prozess

Zwischenfazit Prozesse

- Prozessverwaltung integraler Bestandteil des Betriebssystems
- Prozesszustand determiniert Zuteilungsmöglichkeiten der CPU, konkrete Implementierung abhängig vom Betriebssystem
- Prozessverwaltung in Prozesstabelle, Einträge sind Prozesskontrollblöcke (ggf. Verweise darauf)
- Prozesswechsel initiiert durch Interrupt, Dispatcher führt Wechsel durch

Threads

Prozesserzeugung, Prozessumschaltung und Interprozess-kommunikation sind **rechenzeitaufwendig**, weil stets komplette Prozesskontexte umgeschaltet/verwaltet werden müssen.

Ebenso ist bisher der Zugriff mehrerer Prozesse auf einen gemeinsamen Kontext nicht vorgesehen, was für einige Applikationen jedoch interessant wäre.

Threads

Beispiel: Wie strukturiert man einen **Webserver**, der Anforderungen von **mehreren Klienten** bedienen soll? Varianten:

1. mehrere **Kind-Prozesse** (Parallelität, Blockierungen erlaubt, gute Performance, hoher Aufwand für Prozesserzeugung und Datenaustausch)
2. **ein Prozess** mit nur einem Kontrollfluss (keine Parallelität, Blockierungen erlaubt, schlechte Performance)
3. ein Prozess mit manueller Blockadevermeidung (keine Parallelitat, keine Blockierungen, gute Performance, aufwändige und fehleranfällige Programmierung)

Threads

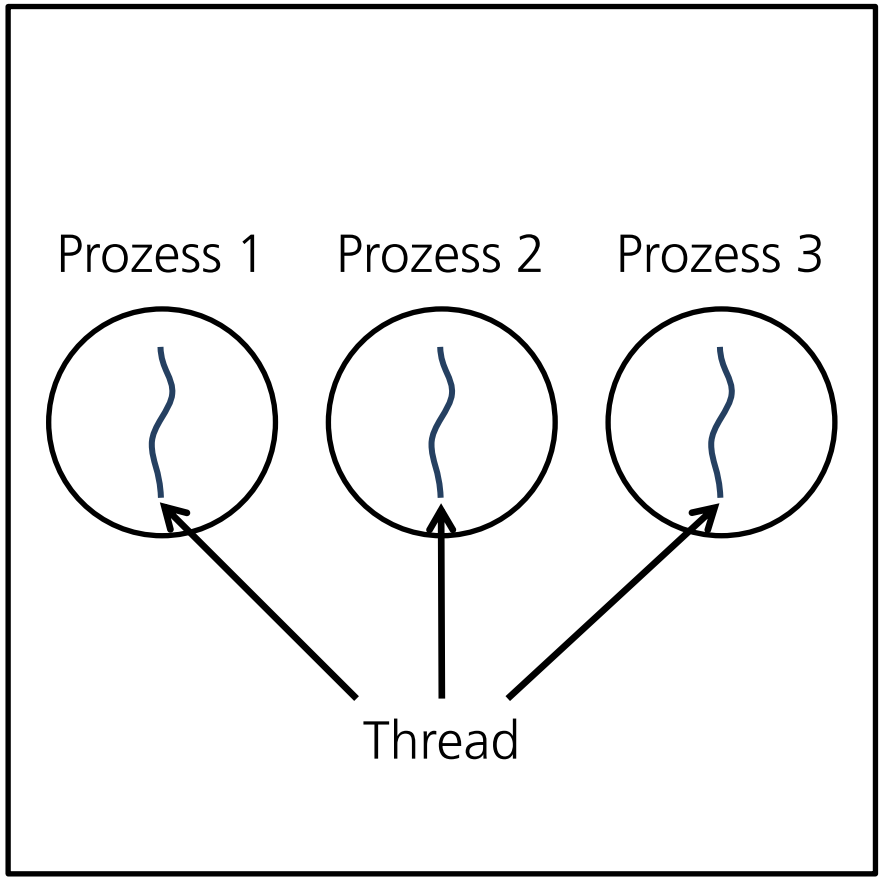
Lösung:

Einführung von Nebenläufigkeit **innerhalb eines Prozesses** durch parallele Ausführungsstränge, sog. **Threads** (*Fäden*).

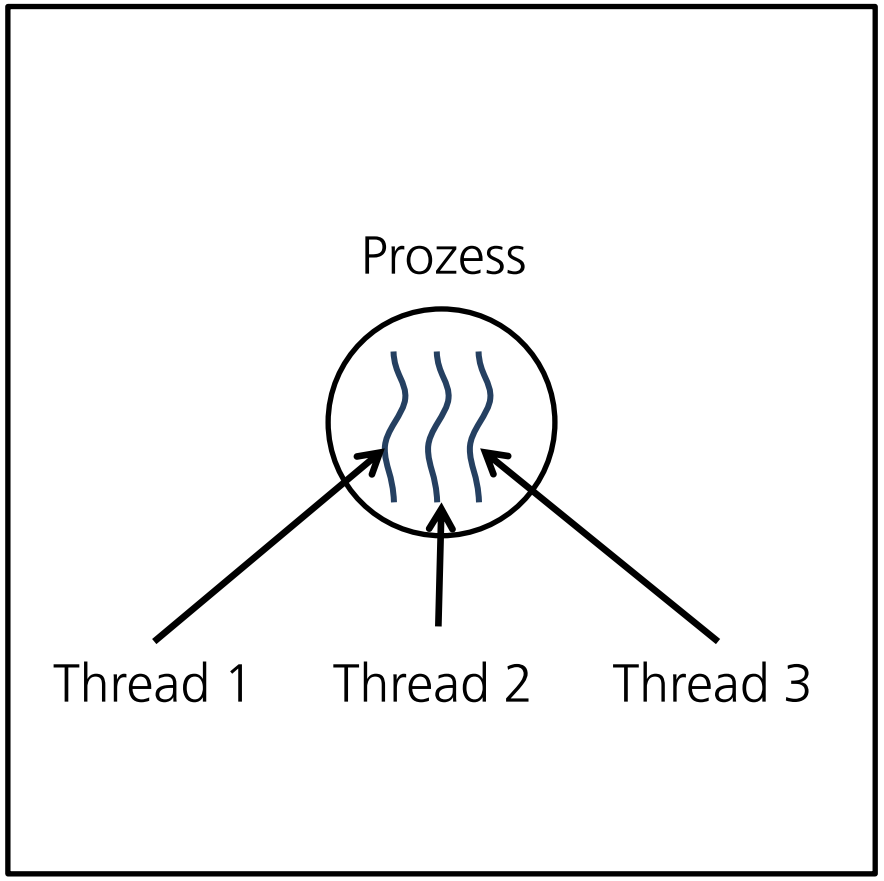
Folgen:

- **Einfachheit** des Prozessmodells bei Programmierung bleibt erhalten,
- **kein vollständiger Kontextwechsel** beim Umschalten zwischen Threads eines Prozesses notwendig,
- Inter-Thread **Kommunikation** von vornherein möglich

Prozesse und Threads



(a) Drei Prozesse,
je Prozess ein Thread



(b) Ein Prozess, drei Threads

Kernel- und User-Threads

Bisher betrachtet: Kernel-Threads

- Implementierung und Verwaltung (Dispatching, etc.) erfolgt analog zu Prozessen durch Betriebssystem

User-Thread:

- Nebenläufigkeit im User-Mode innerhalb eines Kernel-Threads oder Prozesses
- Verwaltung durch Benutzerprozess mittels entsprechender Programmierbibliotheken
- Beispiel: Threads in Java

Kernel- und User-Threads

Kernel-Threads

- per Systemaufruf erzeugt
- Threadwechsel langsam, weil Umschalten in Kernel-Mode notwendig
- Verteilung auf mehrere Prozessoren möglich
- blockierter Thread blockiert nicht den übergeordneten Prozess (echte Nebenläufigkeit)

User-Threads

- per Bibliotheksfunktion erzeugt
- Threadwechsel schnell, weil kein Umschalten in Kernel-Mode notwendig
- Verteilung auf mehrere Prozessoren nicht möglich
- blockierter Thread blockiert den gesamten Prozess

Zusammenfassung

- Prozesse = ausführbare Einheiten inklusive Verwaltungsinformationen
- Threads = Ausführungsstränge innerhalb eines Prozesses,
- Threaderzeugung und Threadumschaltung innerhalb eines Prozesses wesentlich schneller als bei Prozessen.