



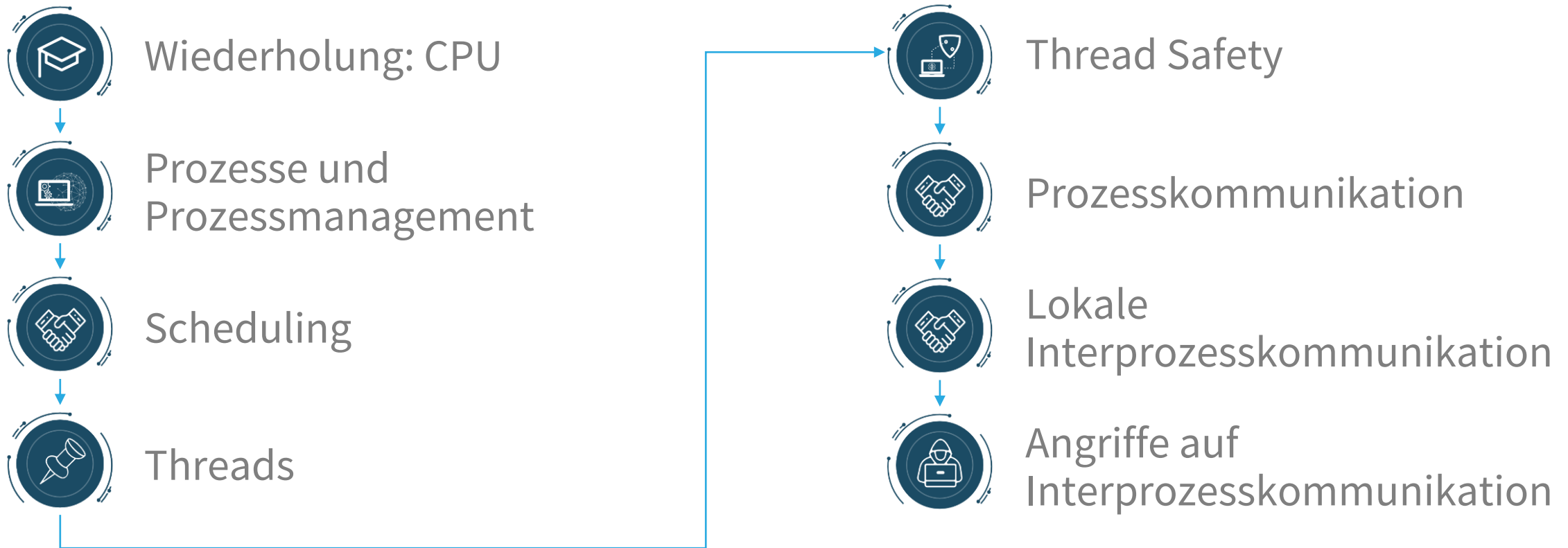
Operation System Security

13 – Threads und Prozesse



SECURNITE

OSSEC 13 – Threads und Prozesse





Wiederholung: CPU



Central Processing Unit (CPU)

Zentrale **Verarbeitungseinheit**

- ist der Teil des Computers, der die Hauptarbeit erledigt und alle anderen Komponenten **steuert**
- liest **Programmcode** (Anweisungen, Befehle) aus dem **Hauptspeicher**, **entschlüsselt** den Programmcode
- **führt** den Programmcode **aus** und **modifiziert** dabei evtl. Daten (Zahlen, Zeichen) und Programmcode im Hauptspeicher
- kann Eingaben **lesen** und Resultate **ausgeben**
- Die Geschwindigkeit wird in **Hertz** gemessen



Wesentliche CPU Bestandteile

- **Control Unit:** liest Programmbefehle aus und führt sie aus
- **Internes Register / Speicher:** Hält Befehle und Daten
- **Arithmetic Logic Unit:** Führt logische (Vergleiche) oder arithmetische (mathematische) Instruktionen aus

Leistung / Durchsatz einer CPU

$$\frac{\text{Befehle}}{\text{Programm}} \times \frac{\text{Taktzyklen}}{\text{Befehl}} \times \frac{\text{Ausführungszeit}}{\text{Taktzyklus}} = \text{NIT} \times \text{CPI} \times \text{CCT}$$

NIT (Number of Instructions per Task) je nach Compilertechnologie, Algorithmen, Betriebssystem

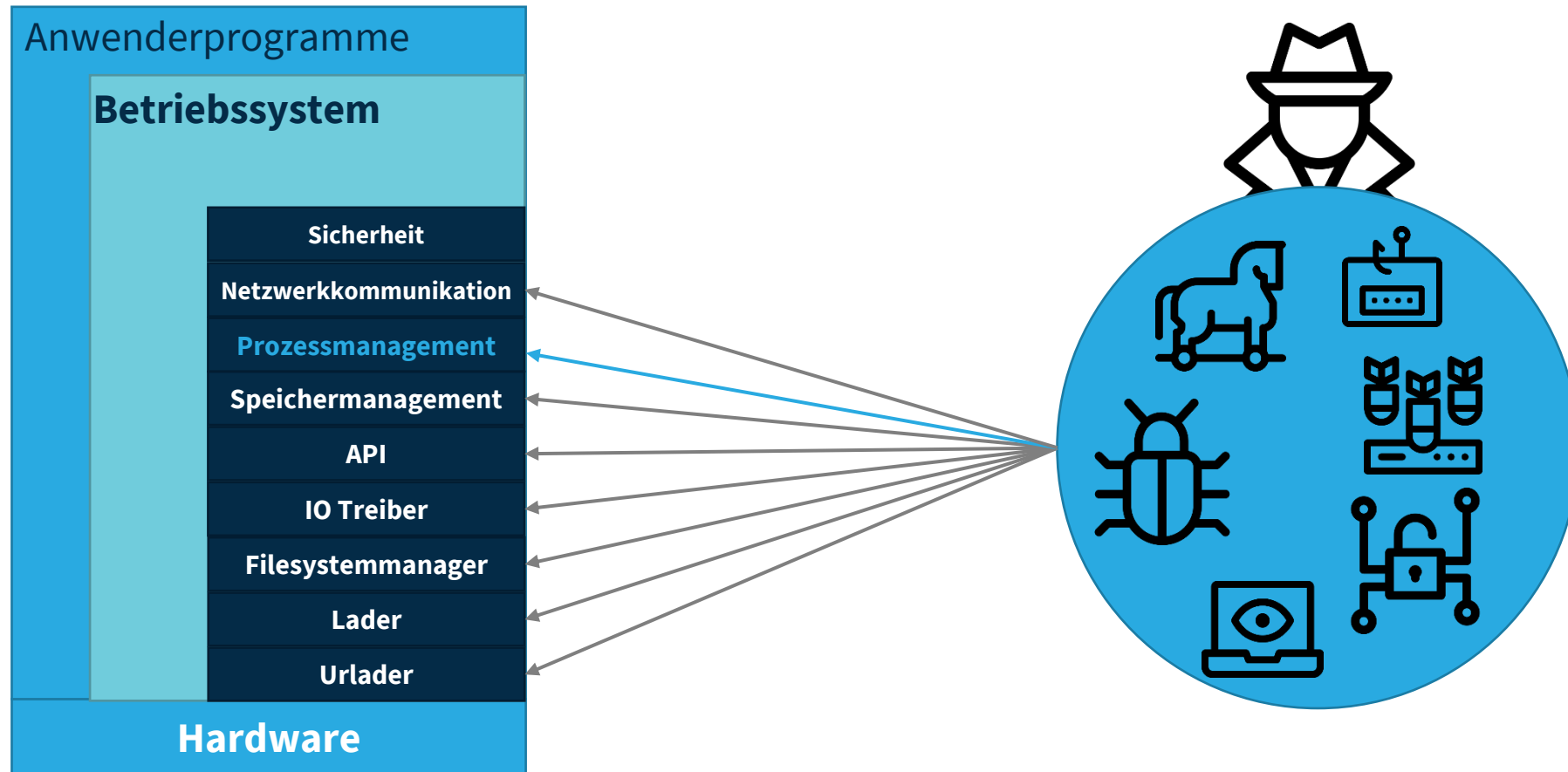
CPI (Clockcycles Per Instruction) je nach Architektur des Befehlssatzes, Pipelining, Anzahl Ausführungseinheiten

CCT (Clock Cycle Time) je nach verwendeter Hardwaretechnologie, Architekturtechniken

Wiederholung

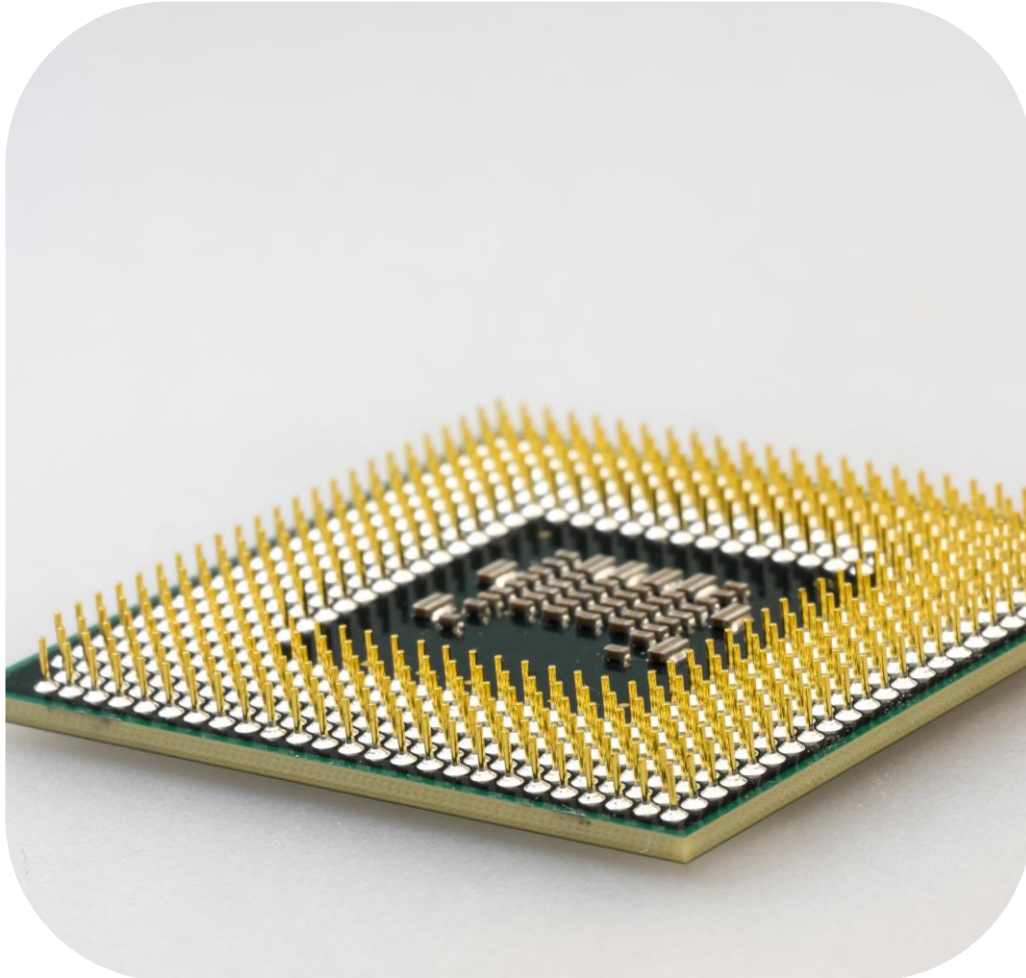


Einführung in die Funktionsweise von Computern





Prozesse und Prozessmanagement



Definition

- **Vorgang**, der **Berechnungen** und **Anweisungen** auf einem Prozessor(kern) ausführt
- **Instanz** eines Programms
- Manchmal auch **Task** genannt (zum Beispiel im Windows Task Manager)



Bestandteile eines Prozesses

Befehlsabfolge des Programms

Benötigt **Zugriff** auf die Ressourcen des Prozessors

Addressbereich im Arbeitsspeicher

Stack Speicher



Verarbeitung im Prozessorkern

Zu **jedem Zeitpunkt** kann auf dem Prozessor oder dem Prozessorkern nur **ein Prozess** verarbeitet werden

Was passiert wenn **mehrere Prozesse** bzw. **Programme Prozessorzeit** benötigen?



(Quasi) Parallelverarbeitung von Prozessen

Prozessorzeit muss zugeteilt werden

Algorithmus entscheidet, welcher Prozess wann Prozessorzeit zugeteilt bekommt

- **Scheduling** Verfahren
- **Priorisierung**

Einführung



Prozess



(Quasi) Parallelverarbeitung von Prozessen

Alle anderen Prozesse werden **angehalten** und **warten** in einer Warteschlange auf Prozessorzeit

Speicherabbild und **Registerbelegung** werden gespeichert

Wechsel von Prozessen bleibt aufgrund Prozessorgeschwindigkeit **unbemerkt**

Einführung

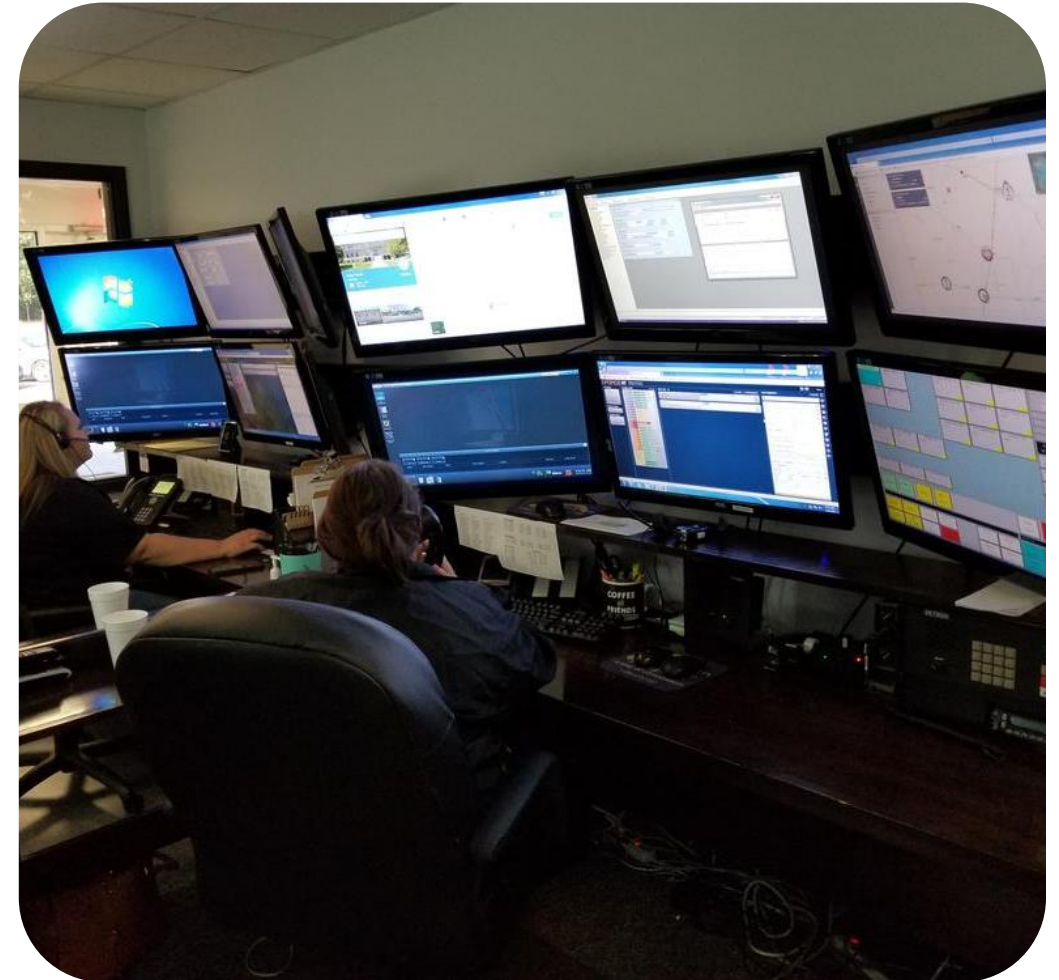


Scheduler & Dispatcher

Zuteilung von Prozessorzeit

Funktion wird von **Scheduler** und **Dispatcher** übernommen

Scheduler: Komponente, welche die **Zuteilung** der Prozessorzeit übernimmt



Einführung



Scheduler & Dispatcher

Zuteilung von Prozessorzeit

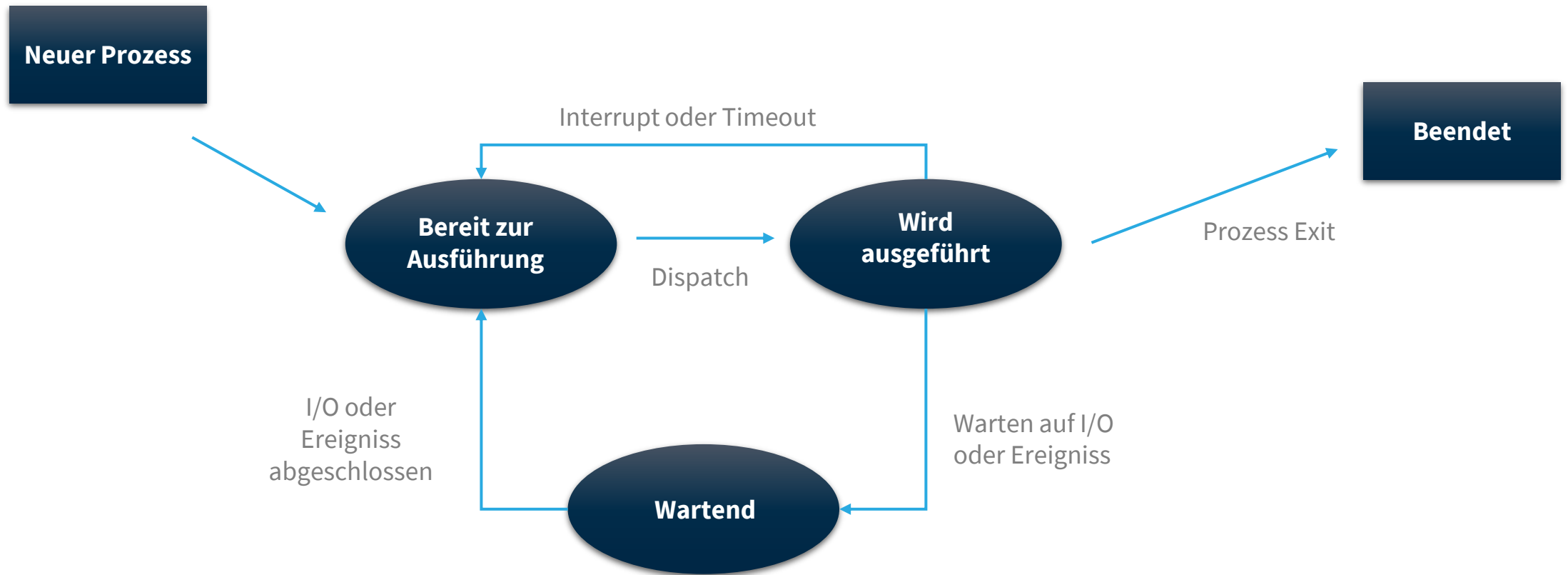
Dispatcher: Für den **Prozesswechsel** zuständige Komponente

Beide Komponenten sind Teile des **Prozessmanagers**



Einführung

Prozessmanagement





Scheduling



Scheduler

- Legt fest, welcher **Prozess** als Nächstes **Zeit** auf dem Prozessorkern bekommt
- Zwei grundsätzliche Scheduling **Verfahren**
 - Non-Preemptive (Nicht verdrängend)
 - Preemptive (verdrängend)



Non-Preemptive Scheduling

- Altes Verfahren
- Prozess wird **nicht unterbrochen** bis er fertig ist
- Nicht geeignet für **konkurrierende Benutzer** und **Echtzeitverarbeitung**
- **Beispiel:** MS-DOS



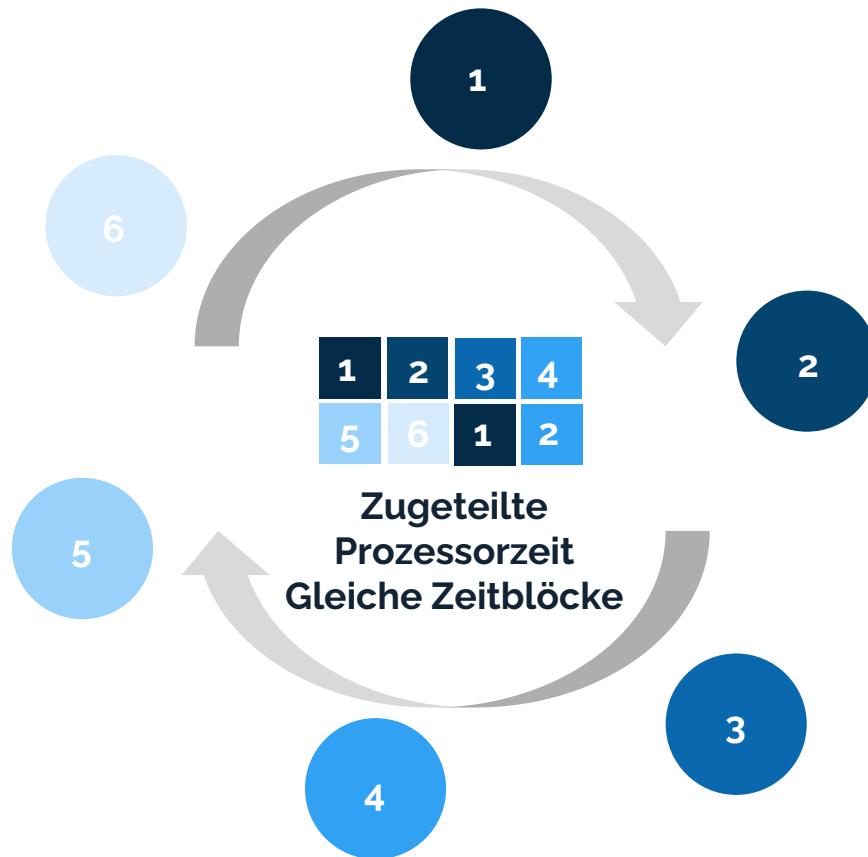
Preemptive Scheduling

- Vorrangunterbrechung
- Prozesse werden **suspendiert** und wieder **aktiviert**
- Geeignet für **konkurrierende Benutzer**
- **Timesharing** Techniken erforderlich



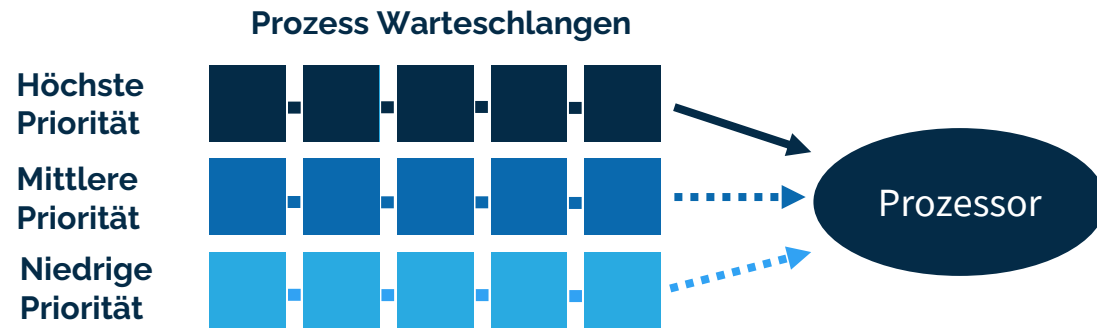
Scheduling Algorithmen

- **First Come First Serve:** Prozesse werden nach der **Reihenfolge** des Eintreffens zugeteilt
- **Shortest – Job – First:** Der **kürzeste** Prozess wird zuerst ausgeführt



Scheduling Algorithmen

- **Round Robin** (Rundlauf):
 - **First Come First Serve** in Verbindung mit einem **Timesharing** Verfahren
 - Jeder Prozess erhält einen **gleichen Zeitblock** auf dem Prozessor
 - Nach Ablauf des Zeitblocks wird der Prozess **angehalten** und der nächste Prozess in der Warteschlange wird **aktiviert**



Scheduling Algorithmen

- **Priority** Scheduling:
 - Priorisierung von Prozessen
 - Prozesse mit **höherer Priorität** werden zuerst ausgeführt
 - Oft in Verbindung mit mehreren **Warteschlangen**



Threads

Threads



Single Threads



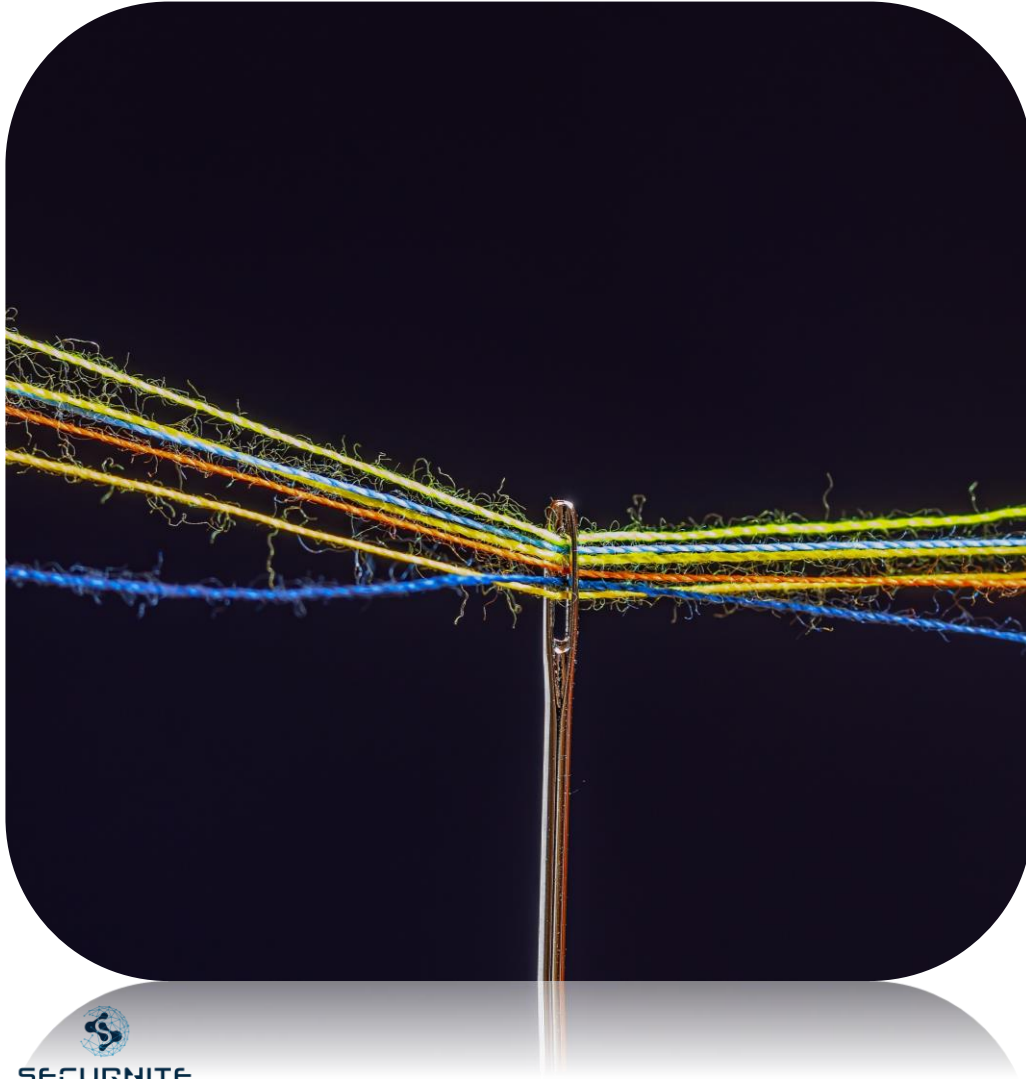
Trace / Single Thread

- Jeder Prozess hat einen **einzigsten Ausführungsablauf**
- Jeder Prozess hat einen **eigenen Adressraum**
- Nur jeweils **ein** Prozess hat Zugriff auf **eine I/O Schnittstelle**

Threads



Multi Threads



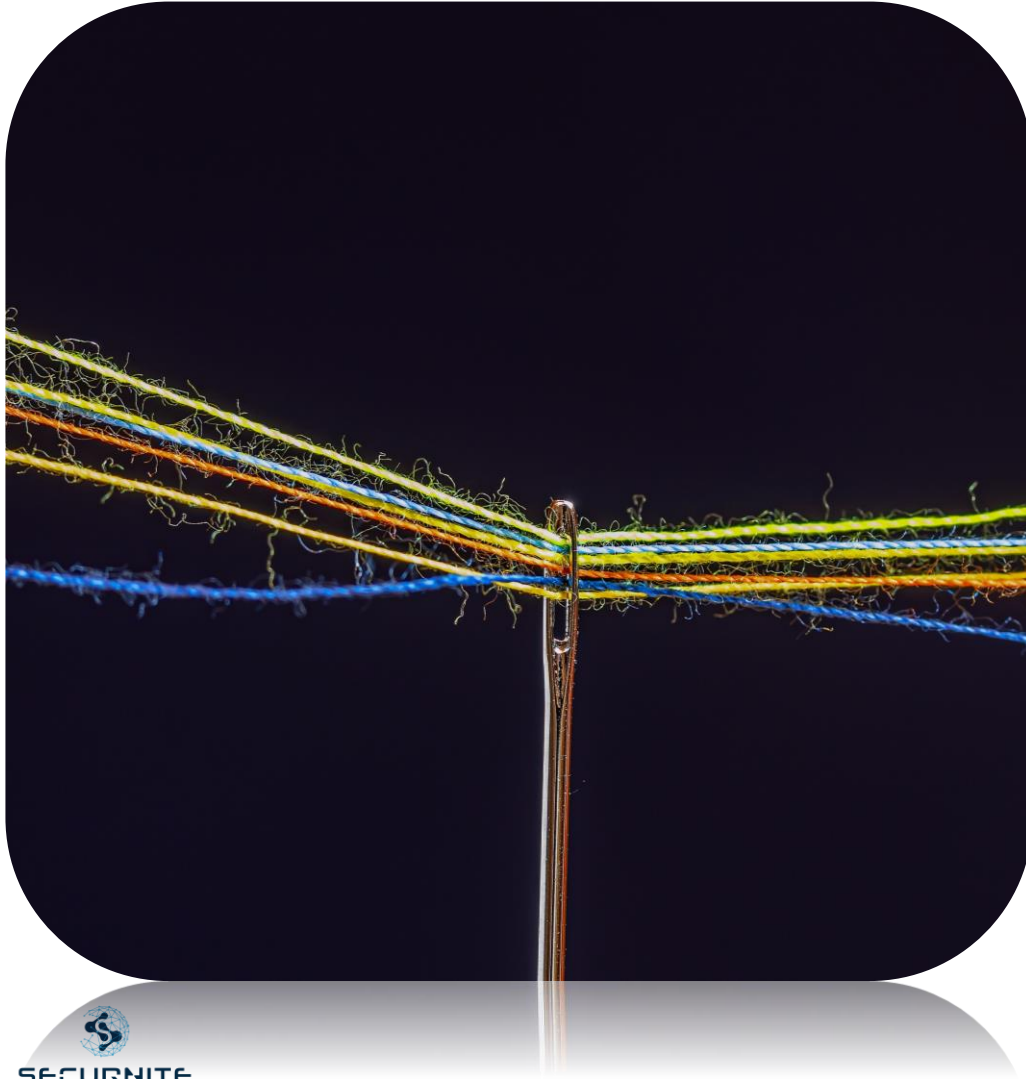
Multi Thread

- Multi Thread Prozesse haben **mehrere Ausführungsabläufe** (Threads)
- Erlaubt mehrere Teile eines Programms **gleichzeitig** auszuführen
- Threads **teilen** sich einen **Adressraum** innerhalb eines Prozess

Threads



Multi Threads



Multi Thread

- Zugriff auf **gleiche Ressourcen** möglich
- Jeder Thread hat **Zugriff** auf den **Speicher** eines anderen Threads, sofern sie sich im **gleichen Prozess** befinden

Threads



Multi Threads



Vorteile

- **Effiziente** Kommunikation
- Zwischen zwei Threads innerhalb eines Prozesses können sehr einfach **Speicheradressen übergeben** werden
- Ausnutzung von **Multi Prozessor Architekturen**

Threads



Multi Threads



Vorteile

- Threads eines Prozesses können auf mehreren Prozessoren / Prozessorkernen **echt parallel** ausgeführt werden
 - **Reaktivität**
 - Programm kann weiter ausgeführt werden, obwohl ein Teil **blockiert** ist
- Beispiel:** Webserver kommuniziert mit Client und lädt gleichzeitig Bilder von einem Server.

Threads



Multi Threads



Vorteile

- **Ökonomie:** Die CPU kann durch die nebenläufige Ausführung von Aufgaben (Multitasking) **maximal ausgelastet** werden



Thread Safety

Thread Safety



- Threads **teilen** sich den **Adressraum** eines Prozesses und können **nicht** auf den Speicher eines anderen Prozesses **zugreifen**
- Threads dürfen sich gegenseitig **nicht behindern**
- **Koordinierung** von Threads nötig, damit sie **nicht gleichzeitig Zugriff** auf einen bestimmten Speicher haben



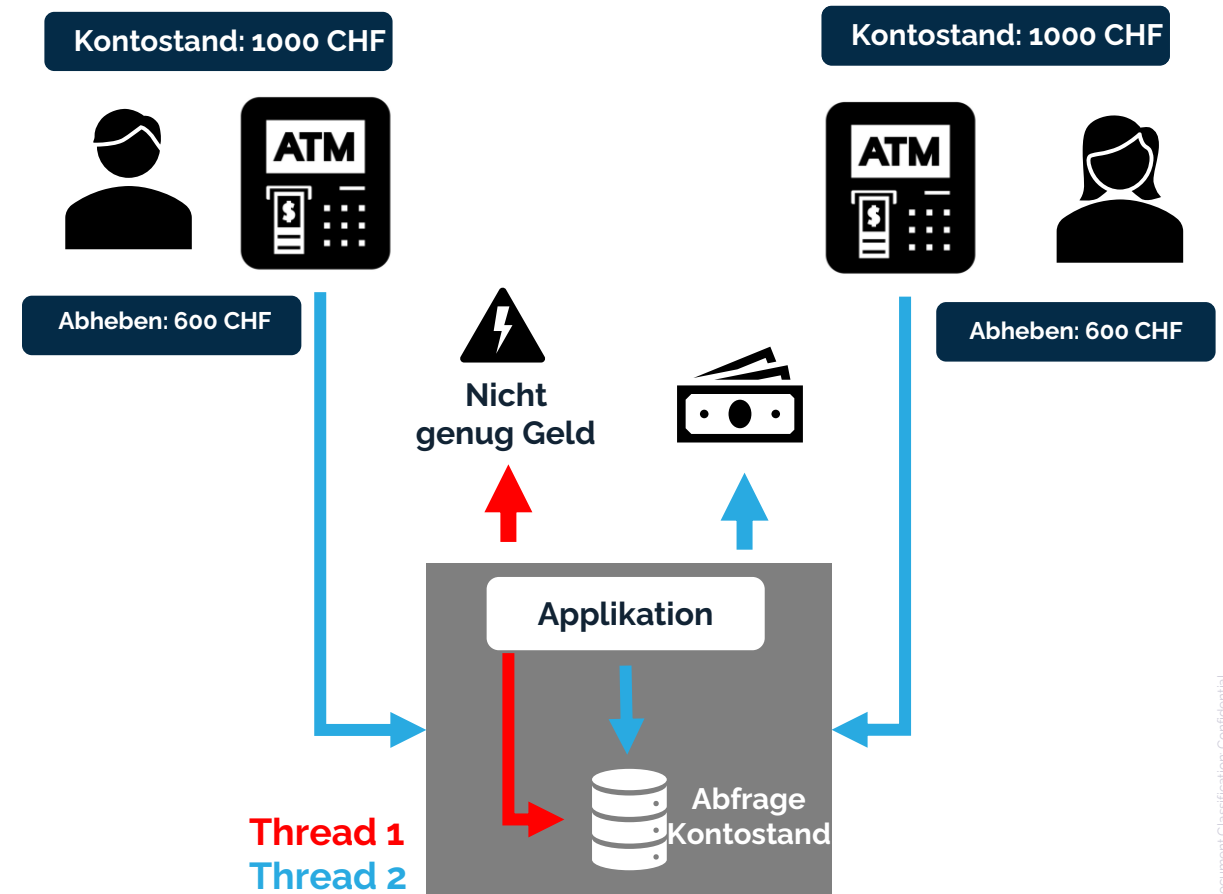
Thread Safety



Beispiel

Gleichzeitiges Geldabheben

- **Zwei Personen** wollen gleichzeitig Geld vom **gleichen Konto** abheben
- Anzeige des **gleichen Kontostands**
- Beide **Transaktionen** starten nacheinander einen Thread im Applikationsprozess



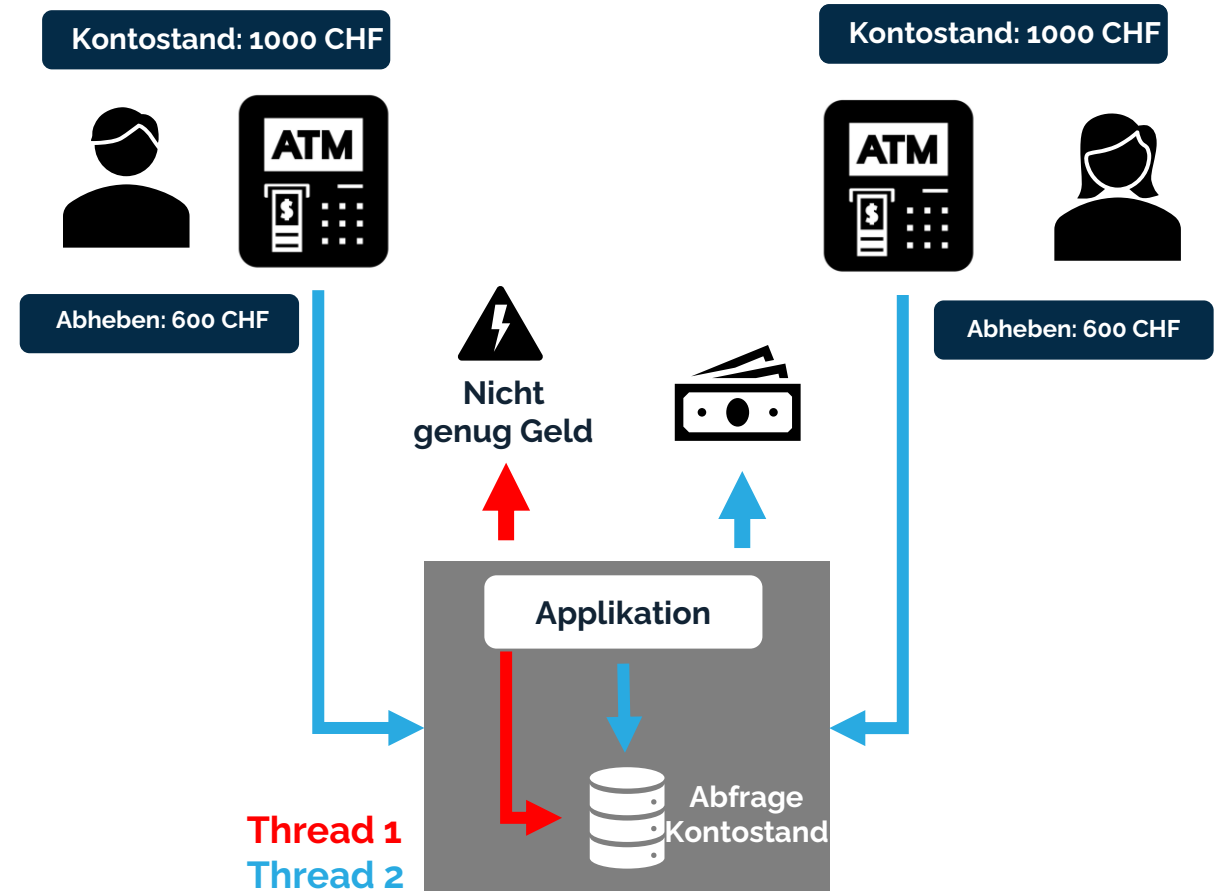
Thread Safety



Beispiel

Gleichzeitiges Geldabheben

- **Thread Safety** Implementierung verhindert, dass Thread 1 während der Transaktion von Thread 2 auf den **Datenspeicher** zugreifen kann
- Nach **Abschluss** der **Transaktion** (Thread 2) und **Aktualisierung** des Kontostands ist somit richtigerweise nicht mehr genug Geld vorhanden, um die Transaktion **abzuschliessen**



Thread Safety



Prozess Synchronisation

- **Koordinierung** von Threads:
 - Threads dürfen sich gegenseitig **nicht behindern**
 - Kein gleichzeitiger **Zugriff** auf **Speicher** möglich
- Realisierung z.B. durch Implementierung von **Semaphoren** (Signalgeber)



Thread Safety

Semaphore

- Abstrakte Datenstrukturen, die als Signalgeber zur **Verfügbarkeit** von **Ressourcen** verwendet werden
- Implementierung von **Locks** möglich
- Arbeitet auf einer **privilegierten Schicht** des **Betriebssystems**
- Ist **nicht** Teil der Prozesse



Thread Safety

Semaphore

- **Binäre Sempahore:**
 - Werte **0** oder **1**
 - beschreibt die Freigabe einer Ressource



Thread Safety

Semaphore

- **Zählende Semaphore:**
 - Datensatz für die **verfügbare Anzahl** einer Ressource
 - Funktionen zur sicheren (im Sinne von Thread Safety) **Aktualisierung** des **Zählstandes**
 - Erlauben Zählerstand größer als 1
 - Managen eines Ressourcen Pools



Synchronisation



Locks

Binäre Semaphore

- **Sperren** des Betriebsmittels, solange ein Thread / Prozess darauf zugreift
- **Read-Lock:**
 - Thread **liest** nur von der Ressource
 - Andere Threads dürfen lesen, aber die Ressource **nicht verändern**



Synchronisation



Locks

Binäre Semaphore

- **Write Lock:**
 - Thread **verändert** die Ressource
 - Andere Threads dürfen daher **weder** die Ressource **lesen** noch **verändern**
- Lock **Freigabe:** Wenn ein Prozess oder Thread mit der **Bearbeitung** fertig ist, muss die Ressource wieder **freigegeben** werden



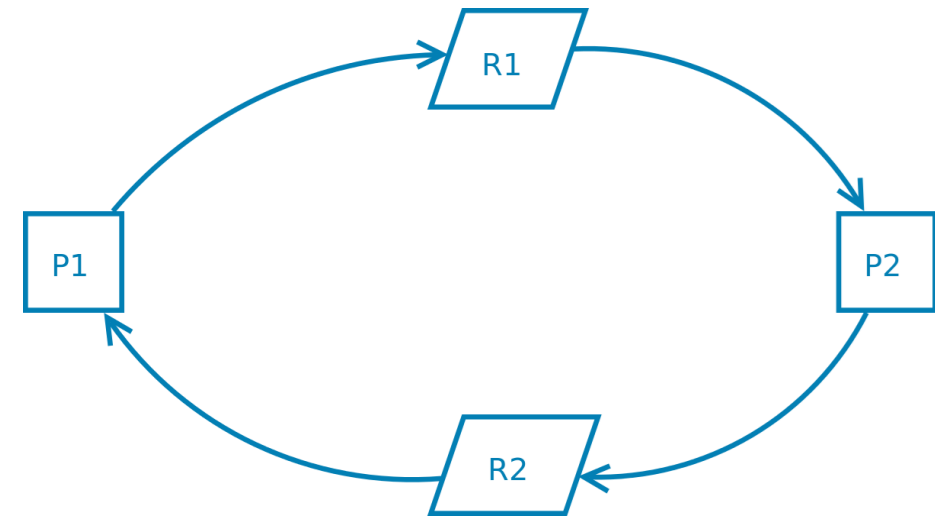
Synchronisation



Locks

Binäre Semaphore


- **Starving („verhungern“):**
 - Warten auf Freigabe einer Ressource
 - diese wird von einem anderen Thread nicht freigegeben
- **Dead Lock:**
 - Gegenseitiges Blockieren
 - Threads blockieren **Ressourcen**, die jeweils von anderen Threads **benötigt** werden

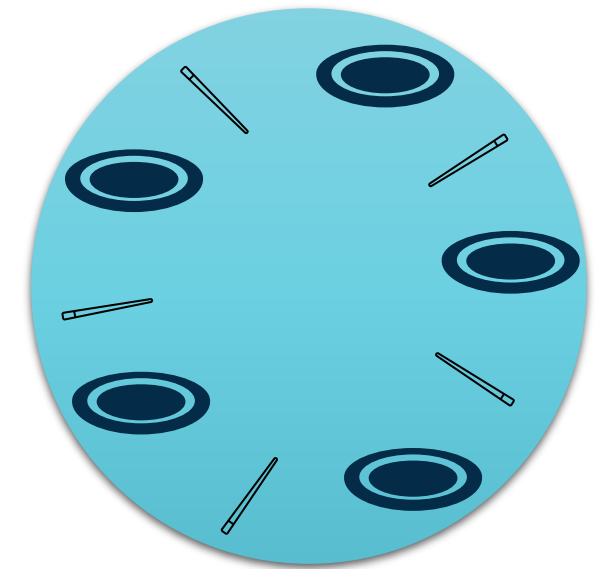


Gruppenübung – Philosophenproblem



15 Minuten

- **Fünf** Philosophen sitzen an einem runden Tisch und denken. Vor jedem Philosoph steht ein Teller mit Reis. Zwischen zwei Tellern liegt jeweils **ein Stäbchen**. Die Philosophen benutzen die Stäbchen gemeinsam (**nacheinander**). Ab und zu, müssen die Philosophen das Denken unterbrechen und etwas essen. Bevor ein Philosoph essen kann, muss er **beide Stäbchen**  neben seinen Teller bekommen.
- **Diskutieren** Sie:
 - Welche **Probleme** können auftreten?
 - Welche **Lösungen** sind denkbar?
- Stellen Sie Ihre Erkenntnisse vor (max. **5 Minuten**)





Prozesskommunikation



Interprozesskommunikation

- **Austausch** von **Informationen** zwischen Prozessen / Threads
- **Lokal:** Innerhalb einer Maschine
- **Verteilt:** Computer übergreifend



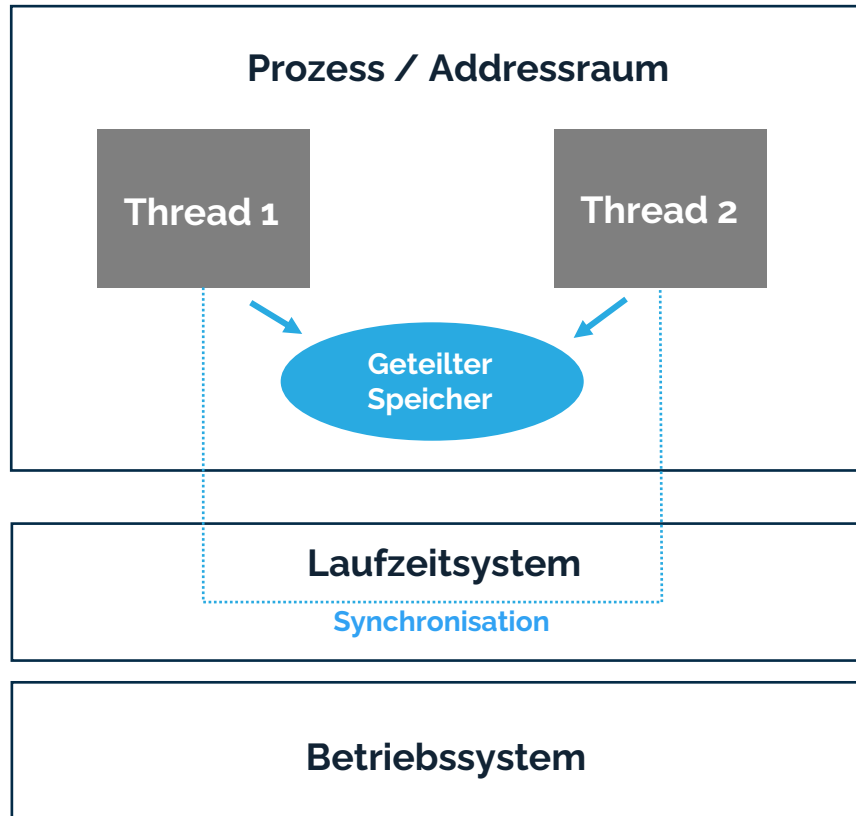
Konzepte

- Speicherbasierte bzw. Nachrichtenbasierte Kommunikation
- Verbindungsorientiert bzw. Verbindungslos
- Synchrone bzw. Asynchrone Kommunikation
- **Senderichtung:** Simplex, Halbduplex, Vollduplex

Prozesskommunikation



Speicherbasierte Kommunikation



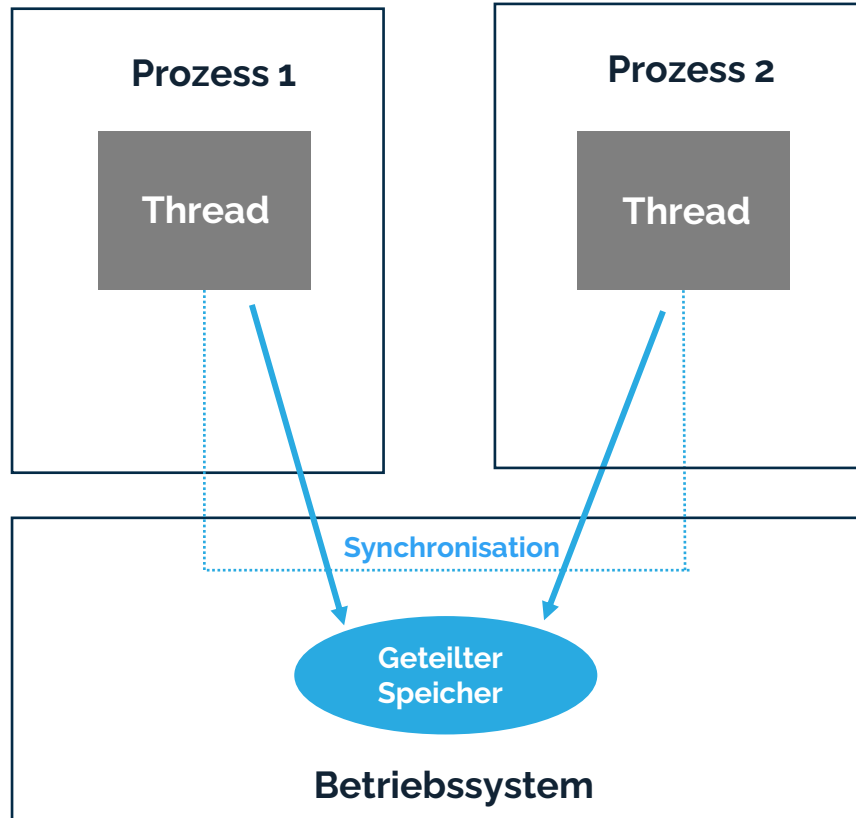
Kommunikation über einen Adressraum

- Zwei Threads eines Prozesses kommunizieren direkt über **geteilten Speicher** im gleichen Adressraum

Prozesskommunikation



Speicherbasierte Kommunikation

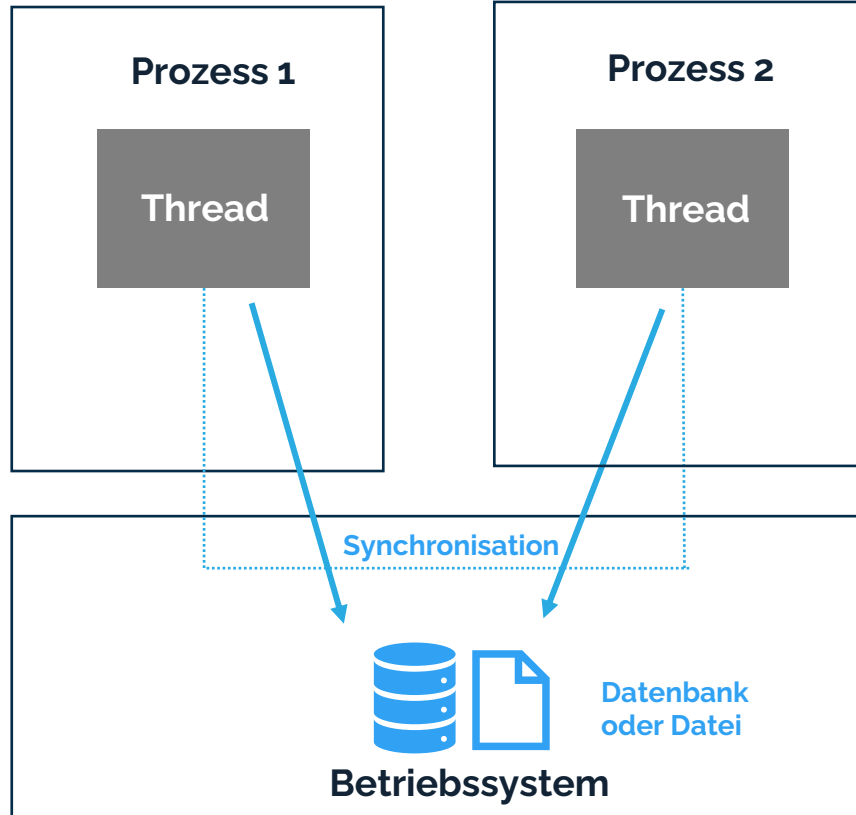


Kommunikation über getrennte Adressräume

- Zwei Threads verschiedener Prozesse kommunizieren über **geteilten Speicherbereich** im Hauptspeicher
- Speicher wird in die Adressräume beider Prozesse **kopiert**

Prozesskommunikation

Speicherbasierte Kommunikation



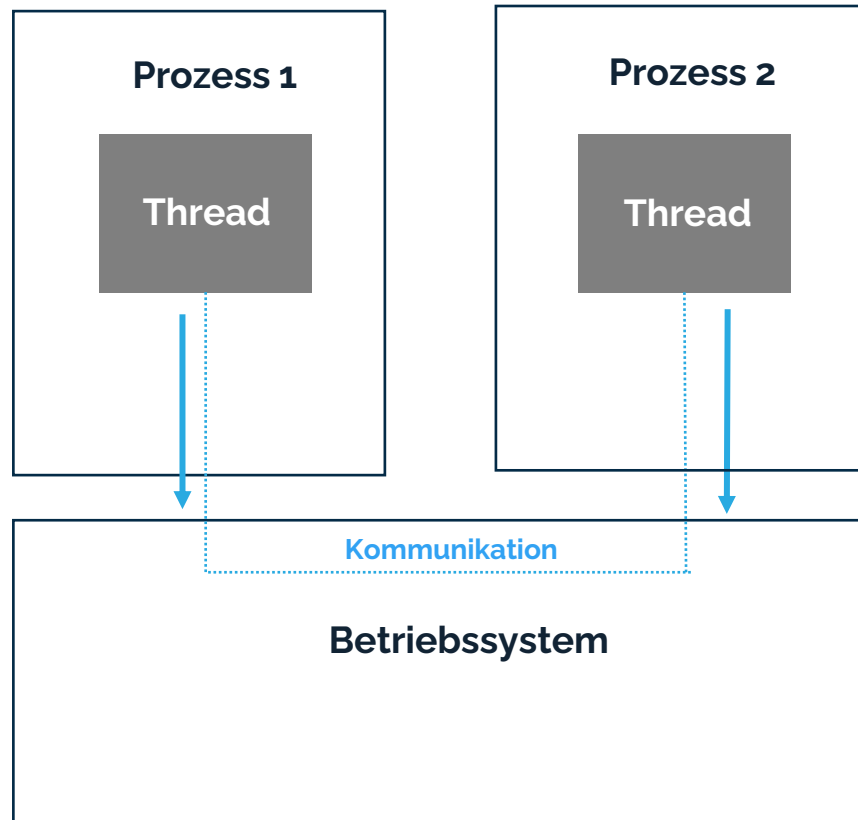
Kommunikation über gemeinsamen Dateizugriff

- Zwei Threads verschiedener Prozesse können über **Dateiaustausch** miteinander kommunizieren
- Zugriff auf **Datenbank**
- **Gemeinsam** genutzte Datei

Prozesskommunikation



Nachrichtenbasierte Kommunikation



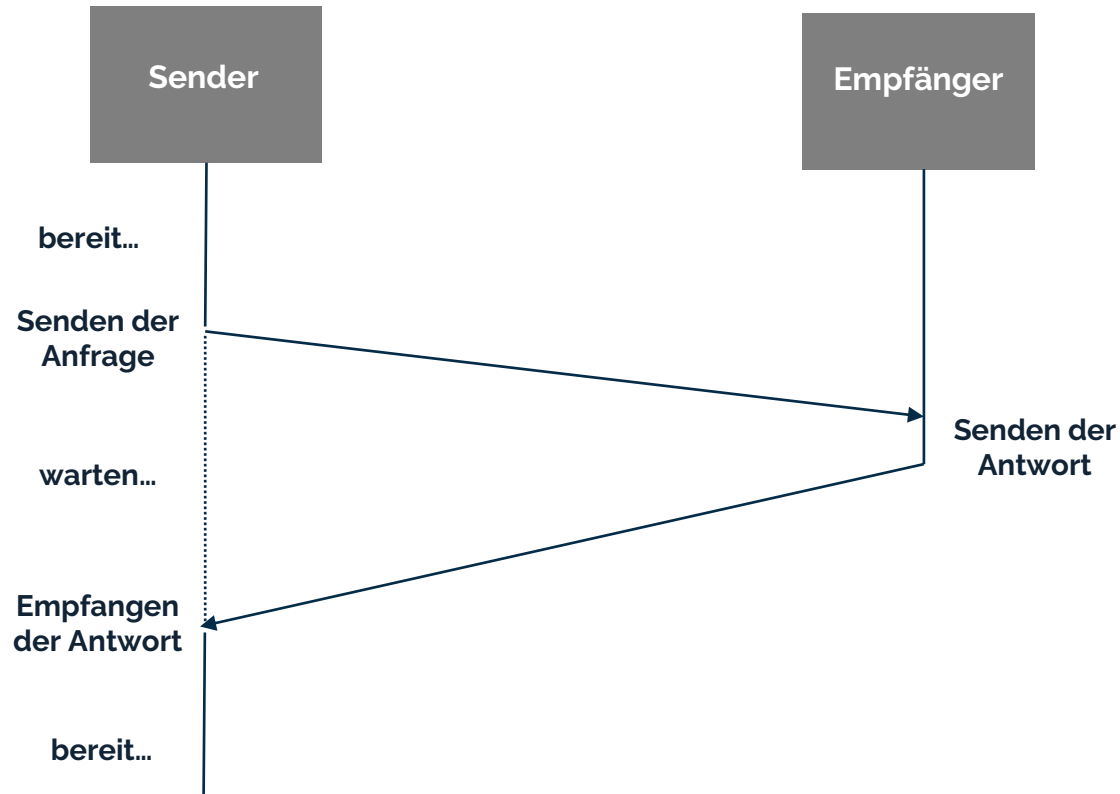
Nachrichtenbasierte Kommunikation

- Zwei Threads aus **verschiedenen Prozessen** / Adressräumen kommunizieren über **Nachrichtendienste** bzw. **Protokolle**
- **Beispiele**
 - TCP oder UDP
 - Sockets
 - Named oder unnamed Pipes

Prozesskommunikation



Synchrone vs Asynchrone Kommunikation



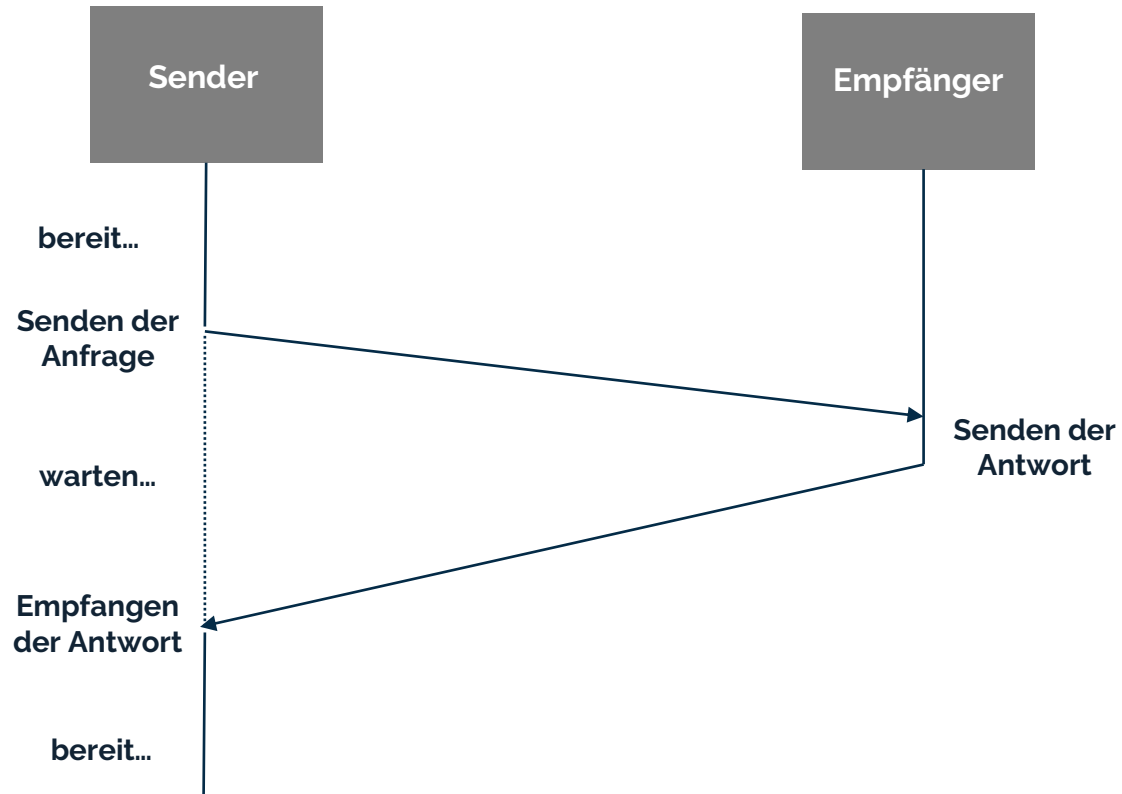
Synchrone Kommunikation

- Beide Prozesse **warten** (blockieren) beim Senden oder Empfangen von Daten bis die **Kommunikation abgeschlossen** ist
- Wird häufig bei Anfragen verwendet, wo **sofort** eine Antwort erwartet wird (Client – Server Prinzip)

Prozesskommunikation



Synchrone vs Asynchrone Kommunikation



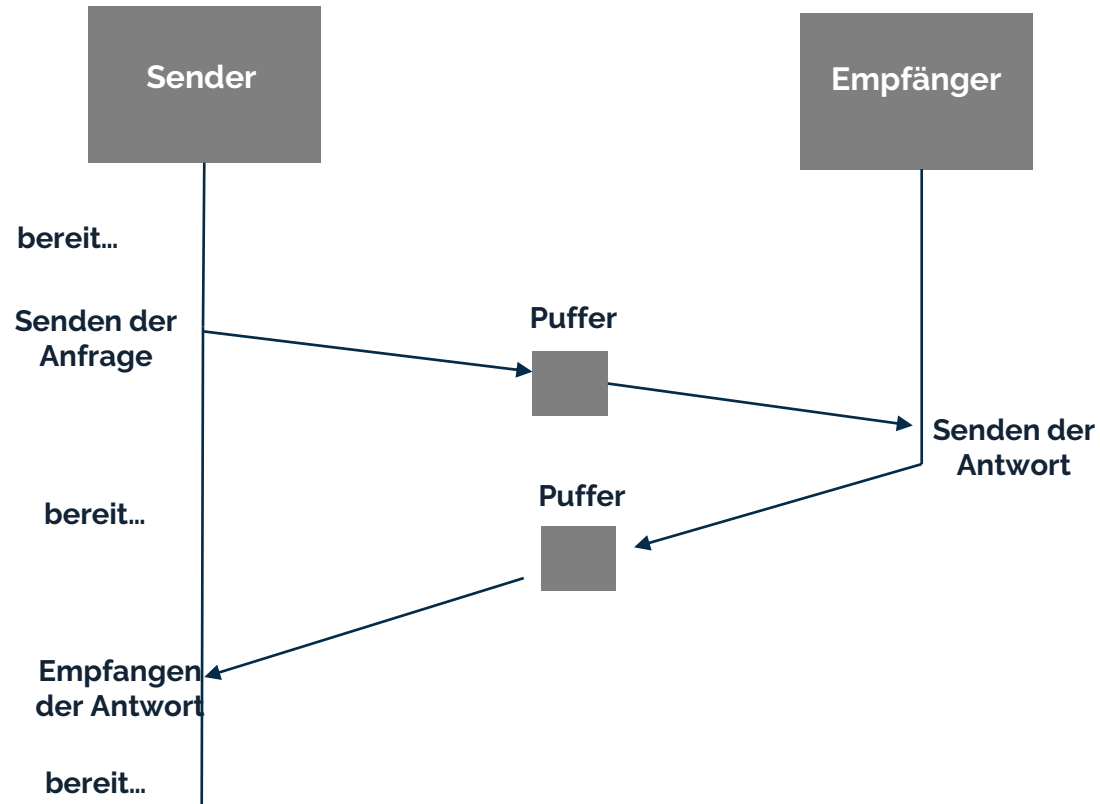
Synchrone Kommunikation

- Bei fehlender Antwort: **Timeout** Fehler
- **Beispiele:**
 - Remote Procedure Call (RPC)
 - HTTP Protokoll

Prozesskommunikation



Synchrone vs Asynchrone Kommunikation



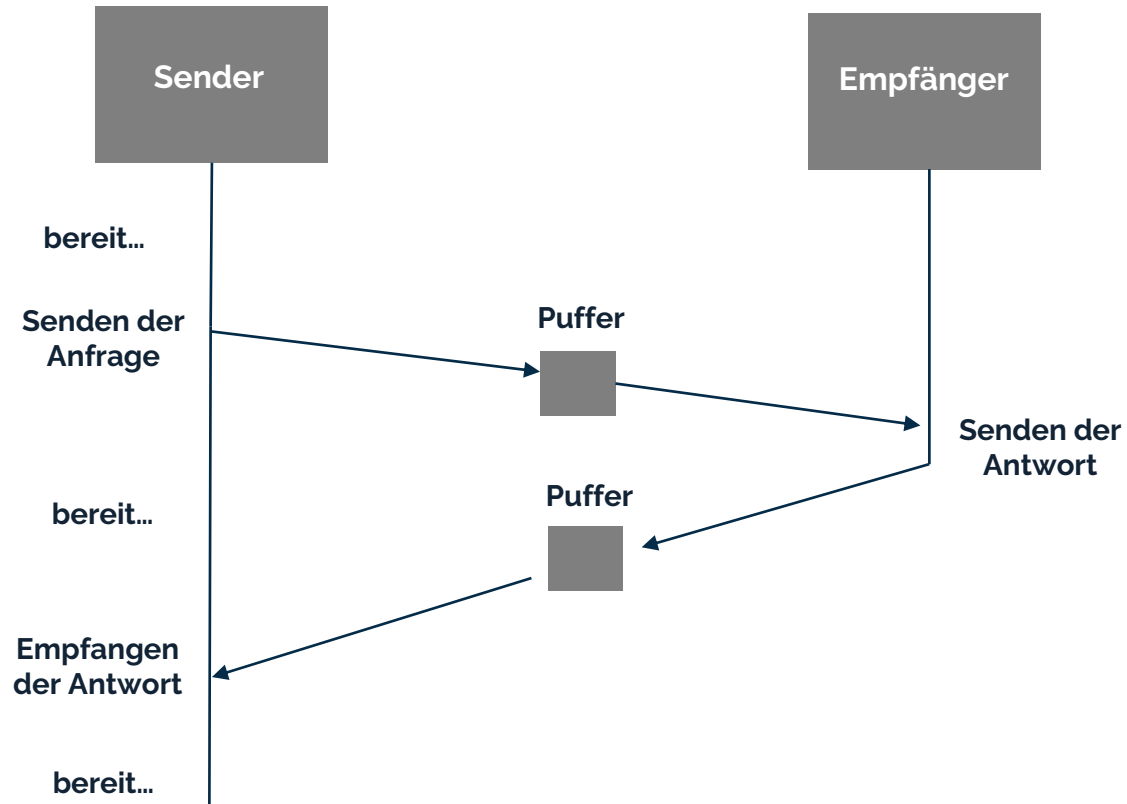
Asynchrone Kommunikation

- Senden und Lesen der Daten über einen **Puffer**
- Sendeprozess kann bis zum Empfang von Daten weiter ausgeführt werden, **blockiert** also **nicht**
- Verwendung z.B. bei **Webservern** zur Abfrage von Daten aus einer Datenbank

Prozesskommunikation



Synchrone vs Asynchrone Kommunikation



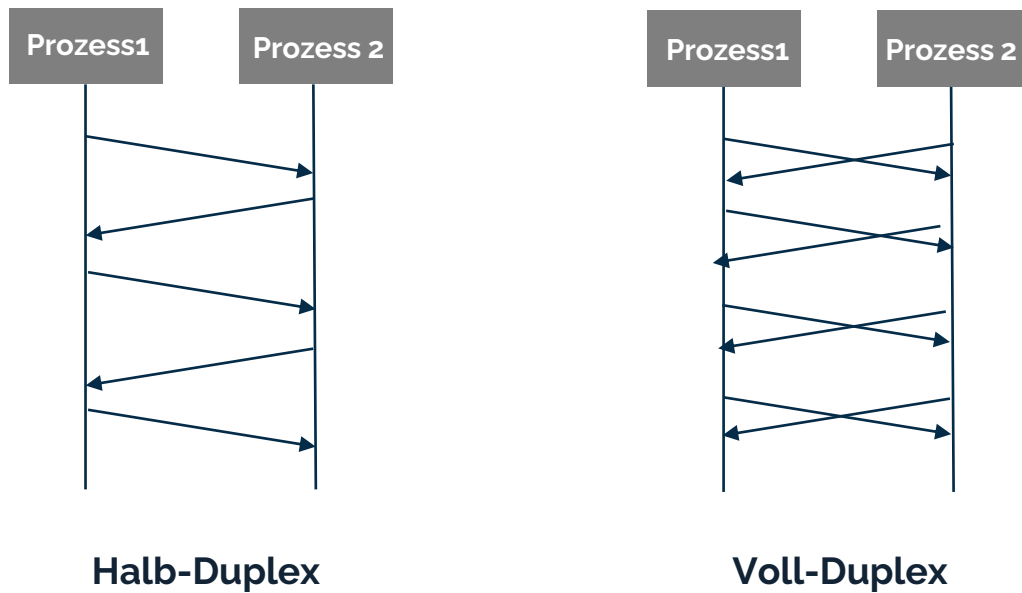
Asynchrone Kommunikation

- Implementierung z.B. durch JavaScript **AJAX** oder moderner mit Promises / `async await`

Prozesskommunikation



Simplex, Halb- und Voll Duplex



- Bezeichnet die **Kommunikationsrichtung**
- **Simplex** : eine Richtung
- **Halb- / Vollduplex**: beide Richtungen
 - **Halb-Duplex**: ein Kommunikationspartner sendet zu einer Zeit (Wechselbetrieb)
 - **Voll-Duplex**: Beide Partner können gleichzeitig senden



Lokale Interprozesskommunikation

Lokale Interprozesskommunikation



IPC

- Innerhalb des **gleichen Betriebssystems**
- Verschiedene Implementierungen je nach Betriebssystem
- **Beispiele** für Methoden unter Unix und Windows:
 - Named und unnamed **Pipes**
 - **Sockets** mit IP Loopback

Lokale Interprozesskommunikation



Pipes



Spezieller **unidirektionaler** Mechanismus zur Kommunikation zwischen Prozessen

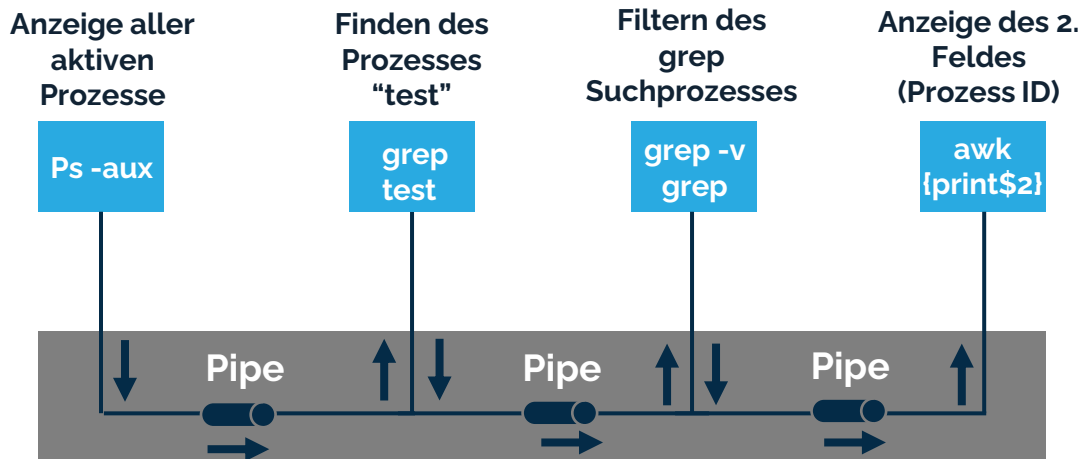
Datenstrom zwischen zwei Prozessen über einen **Puffer** nach dem First In First Out Prinzip (**FIFO**)

Bidirektionale Kommunikation durch Verwendung mehrerer Pipes möglich

Lokale Interprozesskommunikation



Pipes in Unix



Unix und Windows kennen sowohl **named** als auch **unnamed** pipes

Unnamed Pipe (“|” in Unix)

Verbindet Standard Ausgabe eines Prozesses mit der Standard Eingabe eines weiteren Prozesses

Verknüpft mehrere Programme

Beispiel: Ausgabe der Prozess ID des Prozesses “test”

```
Ps -aux | grep test | grep -v grep | awk {print $2}
```


Lokale Interprozesskommunikation



Pipes in Unix

Named Pipe

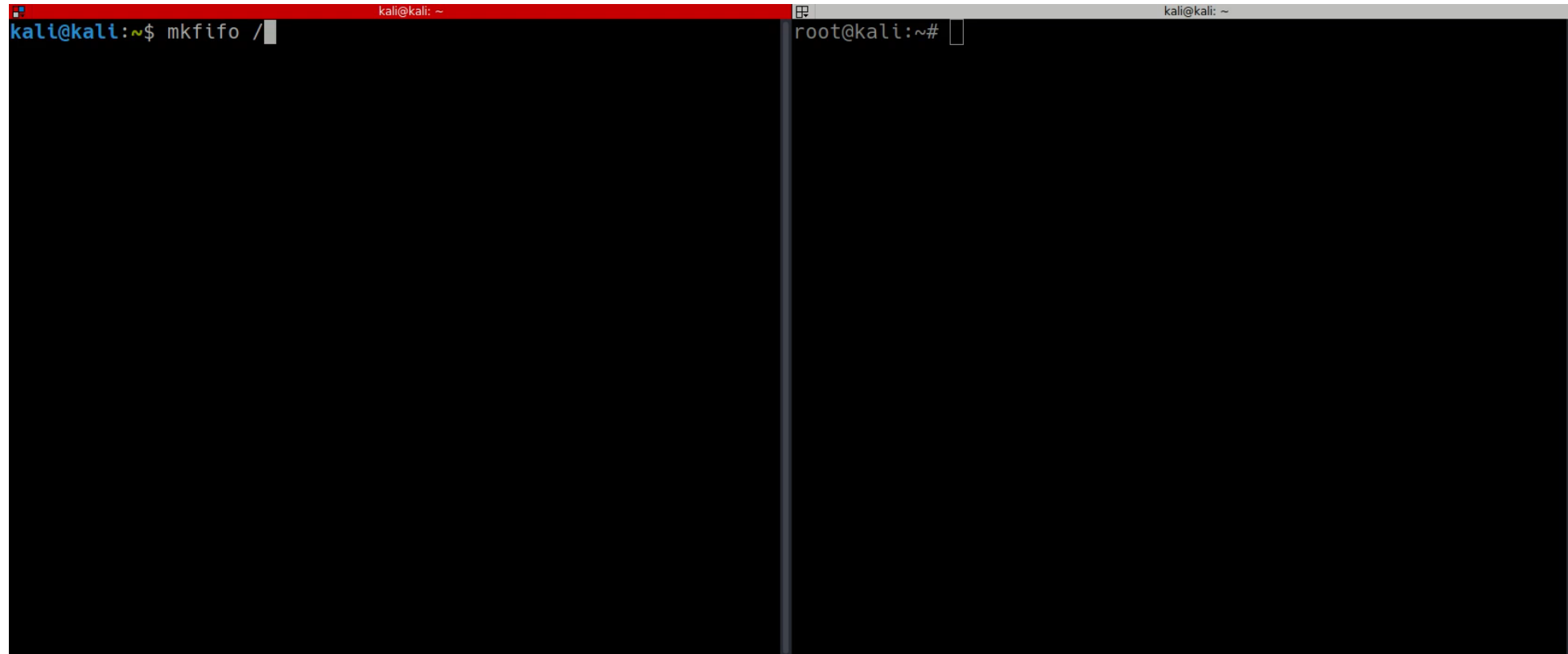
- Können mit dem command `mkfifo () <name>` erstellt werden
- Erzeugen einer „**Datei**“ mit einigen **Besonderheiten**
 - Schreib- und Leseprozess müssen die Datei **gleichzeitig** geöffnet haben
 - **Keine Speicherung** von **Daten** in der Datei
 - Datei ist nur **Symbol** mit ansprechbarem Namen für **unidirektionalen Datenstrom** zwischen beiden Prozessen
- Verwendung zum **Austausch** von **Daten** ohne temporäre Dateien oder wenn ein Prozess keine unnamed pipes unterstützt

Lokale Interprozesskommunikation



Named Pipes

Beispiel: Live Spiegeln des Terminals eines Users (kali) in einem anderen User Terminal (root) mittels "script" und named pipe



Lokale Interprozesskommunikation



Pipes in Windows



Bereitstellung über **Win32 API**

sowohl im **simplex** als auch **duplex**
Betrieb möglich

verwenden spezielles Filesystem **NPFS**
(named pipe filesystem)

Verwendung zur **lokalen** Kommunikation
von Prozessen aber auch zur **remote**
Kommunikation

Lokale Interprozesskommunikation



Pipes in Windows



Unnamed / Anonymous Pipes

Kommunikation eines **Kind** Prozesses mit seinem **Eltern** Prozess

Nur zur **lokalen** Kommunikation

Werden geschlossen, sobald einer der Prozesse abgeschlossen ist

Sind eigentlich named pipes mit **zufällig** vom System gewählten **Namen**

Lokale Interprozesskommunikation



Pipes in Windows



Named Pipes

Können von jedem Prozess, **lokal** und **remote**, angesprochen und zur Kommunikation genutzt werden

Abhängig von gesetzten **security flags**

Werden instanziiert mit jeweils eigenem **Puffer** und **Handle**

Lokale Interprozesskommunikation



Pipes in Windows



Named Pipes

Kommunizieren remote über **SMB** (Port 445) oder Distributed Computing Environment / Remote Procedure Calls **DCE/RPC** (Port 135)

Werden mit **speziellem Pfad** gemountet:

```
\\.\pipe\<<name der pipe> local  
\\<ip>\pipe\<<name der pipe> remote
```



Angriffe auf Interprozesskommunikation

Angriffsfläche



- **Betriebssystemmechanismen** (z.B. Shares) für Prozesskommunikation richtig konfigurieren
- Software muss interne **Kommunikation** (IPC-Kanal) vor anderen Prozessen **schützen**
- Beispiel:
 - **Passwortmanager** mit 2 Threads für Tresor und Browser Plugin
 - Bei fehlerhafter Programmierung können Prozesse anderer Benutzer auf den Kommunikationskanal zugreifen und z.B. **Zugangsdaten stehlen**

Gruppenübung



30 Minuten

- Recherchieren Sie
 - **Gruppe 1:** die Hintergründe zum **IPC\$ share Exploit**
 - **Gruppe 2:** die Hintergründe zur **Service Control Manager Named Pipe Impersonation Vulnerability**
- **Präsentieren** Sie Ihre Erkenntnisse (max. **5 Minuten**). Gehen Sie dabei besonders auf diese **Aspekte** ein:
 - Welche **Eigenschaft** oder **Funktionsweise** von Prozesskommunikation wird ausgenutzt?
 - Welche **Schwachstellen** wird ausgenutzt?
 - Welche **Auswirkungen** kann der Angriff für das Opfer haben?
 - Welche **Massnahmen** können getroffen werden?