Netzwerktechnik und Verteile Systeme

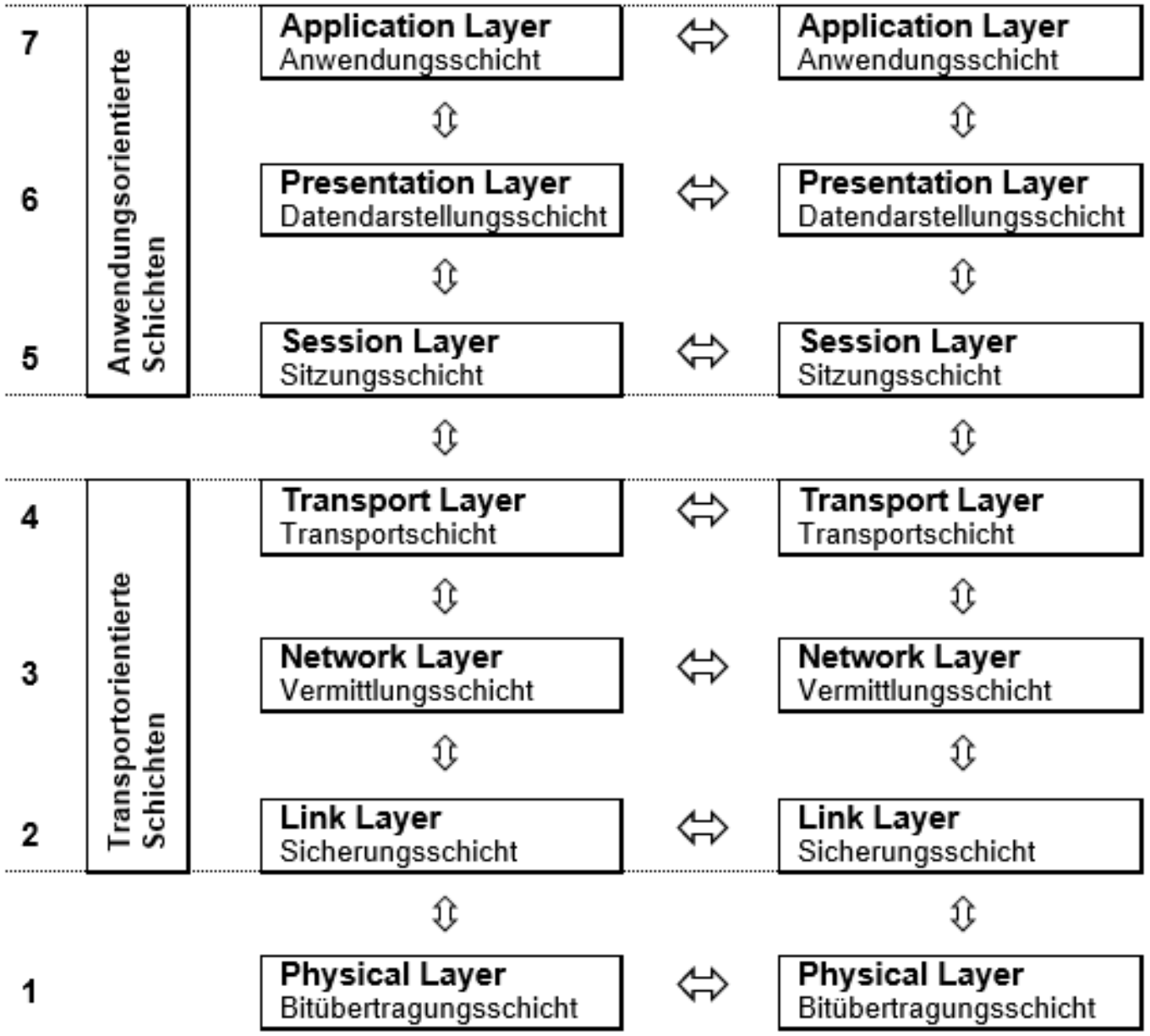
Sebastian Simon

Rekapitulation Netzwerke

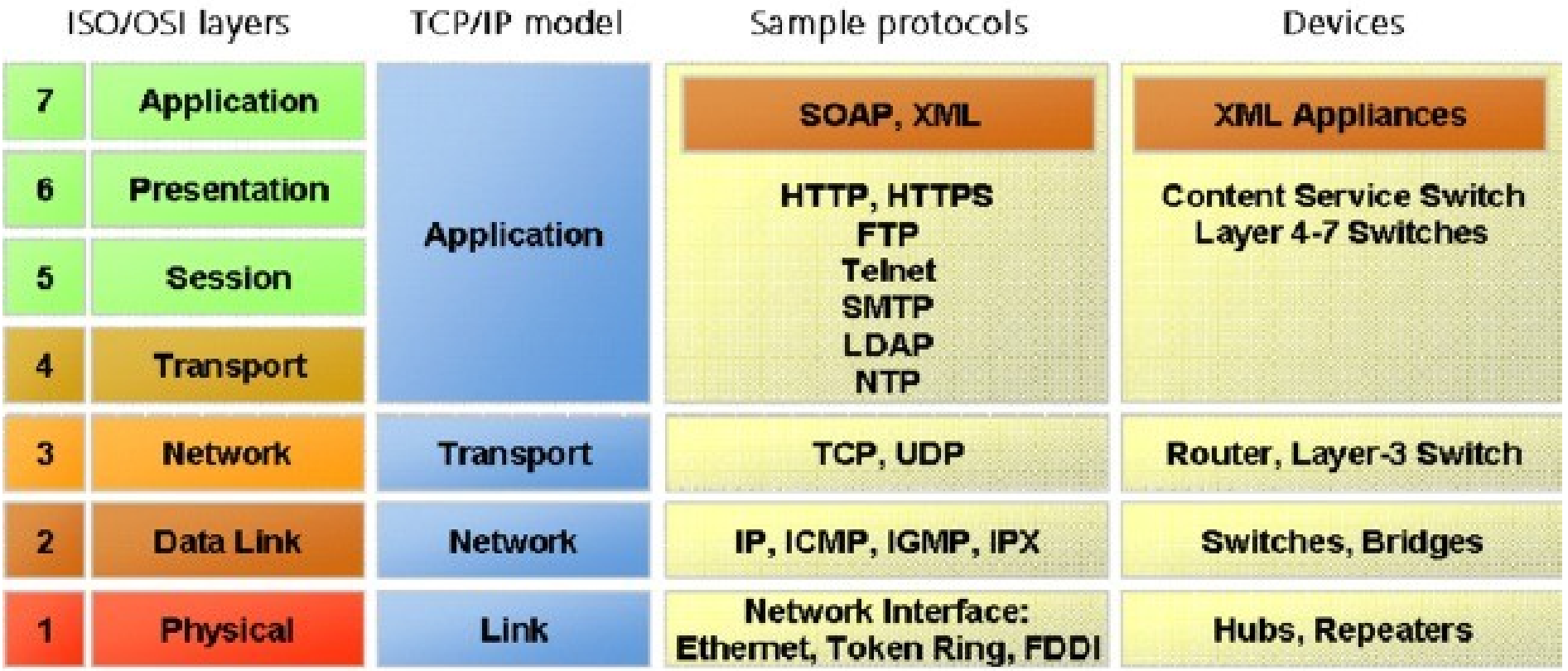
- Netzwerke verbinden Computer
- Ermöglichen Informationsaustausch
- Verschiedene Verbindungsformen
  - LAN-Kabel (Ethernet, Tokenring)
  - Radiowellen (Wireless)
  - Glasfaser (Für Langstrecken)

- Physische Verbindung sind unsicher
  - Datenpaket können verloren gehen oder fehlerhaft sein
  - Netzwerkprotokolle müssen damit umgehen

ISO/-OSI Modell



Umsetzung in der Praxis



Teil 1: Einführung

Definition von Verteilten Systemen

Was sind verteilte Systeme?

Definition eines Verteilten Systems (1)

Eine Sammlung von unabhängigen, vernetzen Computern die dem Benutzers wie ein einzelnes zusammenhängendes System erscheinen.

Definition eines Verteilten Systems (2)

Du weißt, dass es sich um eines handelt, wenn der Crash eines Computers, von dem du noch nie gehört hast, deine Arbeit aufhält. (Leslie Lamport)

Beispiele für Verteilte Systeme

- Webapplikation
  - Online-Banking, Online-Shops, ...
  - Webmail, Moodle, ...
- Mobile Apps (sofern mit Backend vernetzt)
  - Fahrplanauskunft, Messengerdienste, ...
- Peer-to-Peer (P2P)
  - BitTorrent
- Vollautomatische Produktionsstraßen
- Internet of Things (IoT)

Warum werden Systeme verteilt?

- Resourcen und Services werden mit Benutzern verbunden
  - Basisfunktion eines verteilten Systems
- Verfügbarkeit und Ausfallsicherheit
- Performance
  - Geringere Latenzzeit, höherer Durchsatz, …
- Grundsätzlich gilt: Nur verteilen, wenn notwendig
  - Programme werden viel komplexer, fehleranfälliger

8 Trugschlüsse Verteilter Systeme

- Das Netzwerk ist verlässlich
- Die Latenzzeit ist null
- Brandbreite ist unendlich
- Das Netzwerk ist sicher
- Die Topologie ändert sich nicht
- Es gibt einen Administrator
- Transportkosten sind null
- Das Netzwerk ist homogen

Praktisch jeder der zum ersten Mal eine verteile Applikation entwickelt hat diese 8 Vorstellungen im Kopf. Diese erweisen sich langfristig aber als falsch und verursachen viel Ärger und einen schmerzhaften Lernprozess. (Peter Deutsch)

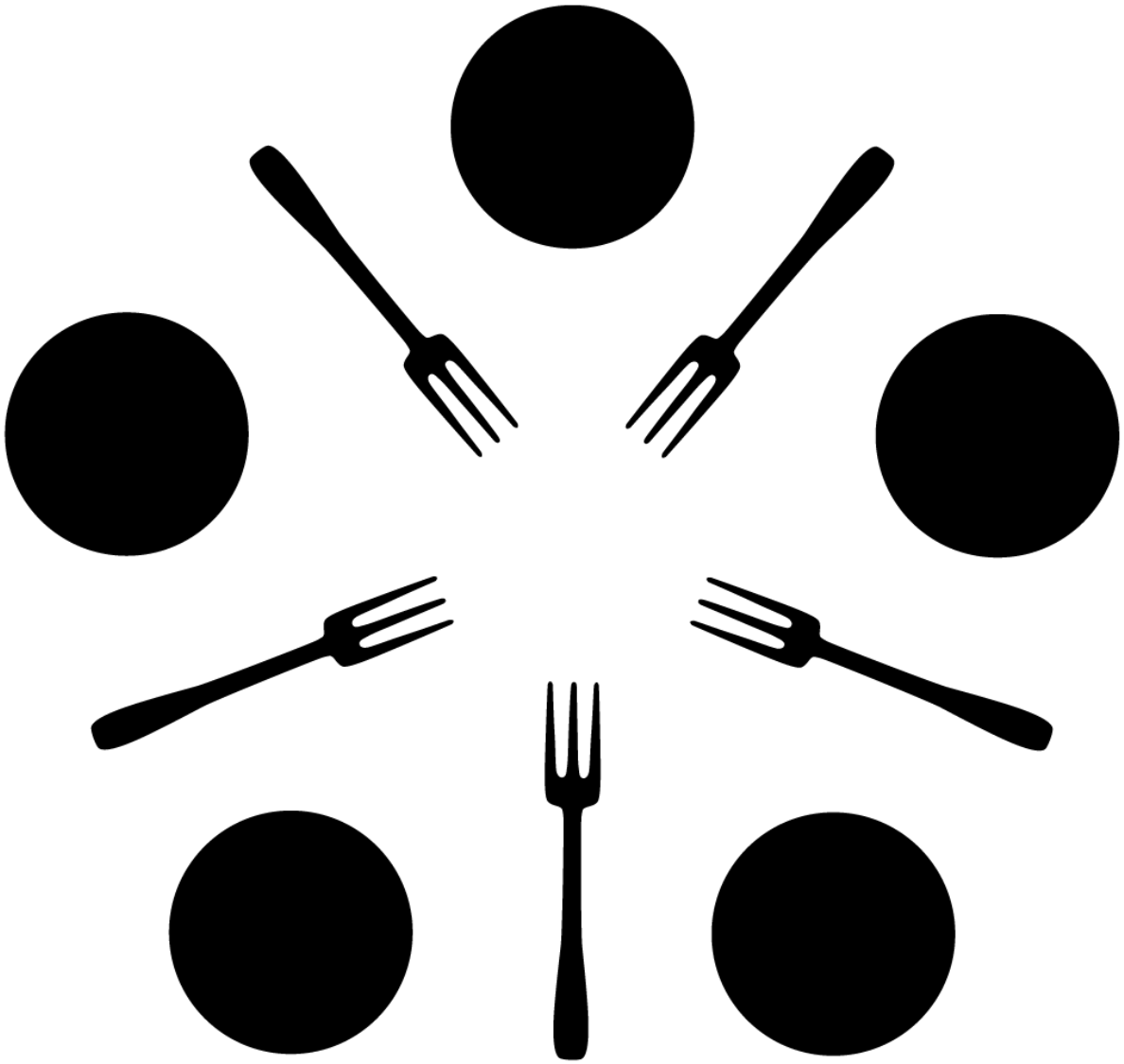
Design-Ziele verteilter Systeme

- Resourcen teilen
- Nebenläufigkeit
- Transparenz
  - Verstecken interner Strukturen und Komplexität
- Offenheit
  - Portabilität, Interoperabilität
  - Services werden gemäß Standards angeboten
- Skalierbarkeit
  - Die Fähigkeit das System einfach zu erweitern
- Fehlertoleranz

Resourcen teilen

- Eine Resource soll von mehreren Usern bzw. Services genutzt werden können
  - zB Informationen über aktuelle Zugverspätungen soll auf Bahnhöfen und in App abrufbar sein
- Viele Resourcen sind aber nur exklusiv nutzbar
  - zB Netzwerkdrucker, Schreiben in DB-Tabelle
  - Lange Wartezeiten können entstehen
    ►die Resource wird zum “Flaschenhals”
  - Exklusiv nutzbare Resource führen auch zu anderen Problemen (siehe nächste Folie)

Dining Philosophers



Nebenläufigkeit (1)

- Auch Resourcen die parallel nutzbar sind, können zum Flaschenhals werden
  - zB Zentrales Service für Zugverspätungen für alle Bahnhöfe, Apps, Webseiten, internen Applikationen
- Viele Anfragen auf eine einzige zentrale Resource überlasten diese, wodurch auch die anderen Services zum Stillstand kommen
- Lösung: Mehrere Anfragen müssen parallel abgearbeit werden

Nebenläufigkeit (2)

- Threads ermöglichen Nebenläufigkeit auf einem Server
- Mehrere Threads laufen gleichzeitig auf verschiedenen Prozessorkernen
- Somit können mehrere Anfragen zur selben Zeit bearbeitet werden

- Wenn das immer noch nicht reicht, müssen mehrere Server parallel arbeiten
- Erfordert jedoch aufwändige Synchronisation

Transparenz

- Konzept: Verstecke die verschiedenen Aspekte der Verteilung vor dem Client
- Aspekte, die versteckt werden können:
  - Zugriff – wie auf Resourcen zugegriffen wird
  - Standort – die Lage der Resourcen
  - Replikation – das Vorhandensein von Kopien
  - Nebenläufigkeit – Resourcen werden unter vielen Usern geteilt
  - Störungen – Ausfälle von Resourcen
- Achtung: Nicht alles kann/muss versteckt werden

Offenheit (1)

- Services werden gemäß gängigen Standards angeboten
- Diese Standards sind in Protokollen formalisiert
- Klar definierte, gut dokumentierte Schnittstellen

- Ziel ist das System flexibel zu halten
  - Erleichtert Komposition, Konfiguration, Erweiterung und Austausch

Offenheit (2)

- Beispiele aus dem WWW
  - Interoperabilität zwischen verschiedene Webservices und Web-Browser funktioniert
  - Neue Browser können entwickelt werden und funktionieren mit bestehenden Servern (und vice versa)
- Beispiel für offene Service-Schnittstellen
  - REST-Schnittstelle, Datenaustausch im JSON-Format, Schnittstellenbeschreibung mit SWAGGER
  - SOAP-Schnittstelle, Datenaustausch in XML, Schnittstellendefinition mittels WSDL

Ausschnitt WSDL

```
 <definitions name = "HelloService"
TargetNamespace = "http://www.examples.com/wsdl/HelloService.wsdl" (...)>

<message name = "SayHelloRequest">
   <part name = "firstName" type = "xsd:string"/>
</message>

<portType name = "Hello_PortType">
   <operation name = "sayHello">
      <input message = "tns:SayHelloRequest"/>
      <output message = "tns:SayHelloResponse"/>
   </operation>
</portType>

<binding name = "Hello_Binding" type = "tns:Hello_PortType">
   <soap:binding style = "rpc"  transport = "http://schemas.xmlsoap.org/soap/http"/>
   <operation name = "sayHello">
(...)
```

Skalierbarkeit

- Ist die Fähigkeit eines Systems zu wachsen um steigende Anforderungen zu erfüllen
- Wachstum gibt es mehreren Dimensionen
  - Größe (Benutzer und Resourcen)
  - Geografisch (zusätzliche Standorte)
  - Administrativ (zB neue Teilorganisationen)
- System bleibt leistungsfähig
- Keine Änderungen an der Software erforderlich

Fehlertoleranz

- Die Fähigkeit eines Systems im Falle von Fehlern in einer oder mehrerer Komponenten korrekt weiterzuarbeiten
  - Eine fehlerhafte Benutzereingabe wird erkannt und entsprechend behandelt
  - Software-Fehler werden auf einer anderen Ebene abgefangen
  - Bei Ausfall einer Hardwarekomponente übernimmt eine andere gleichwertige

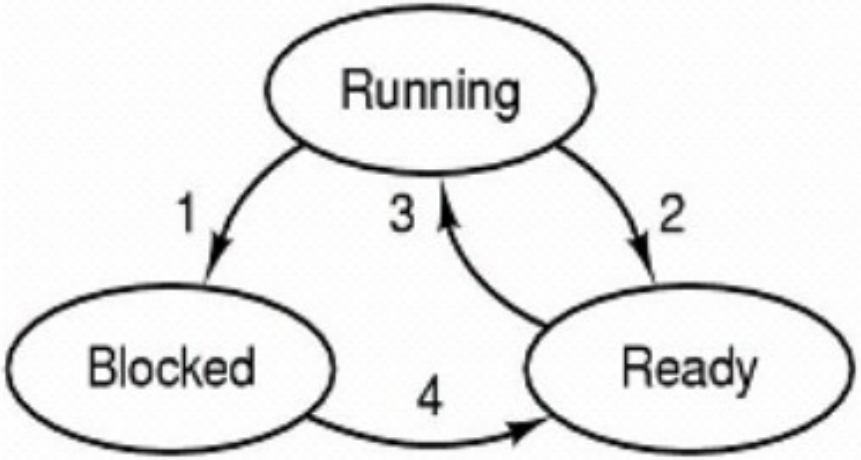
Appendix: Prozesse und Threads

Prozess

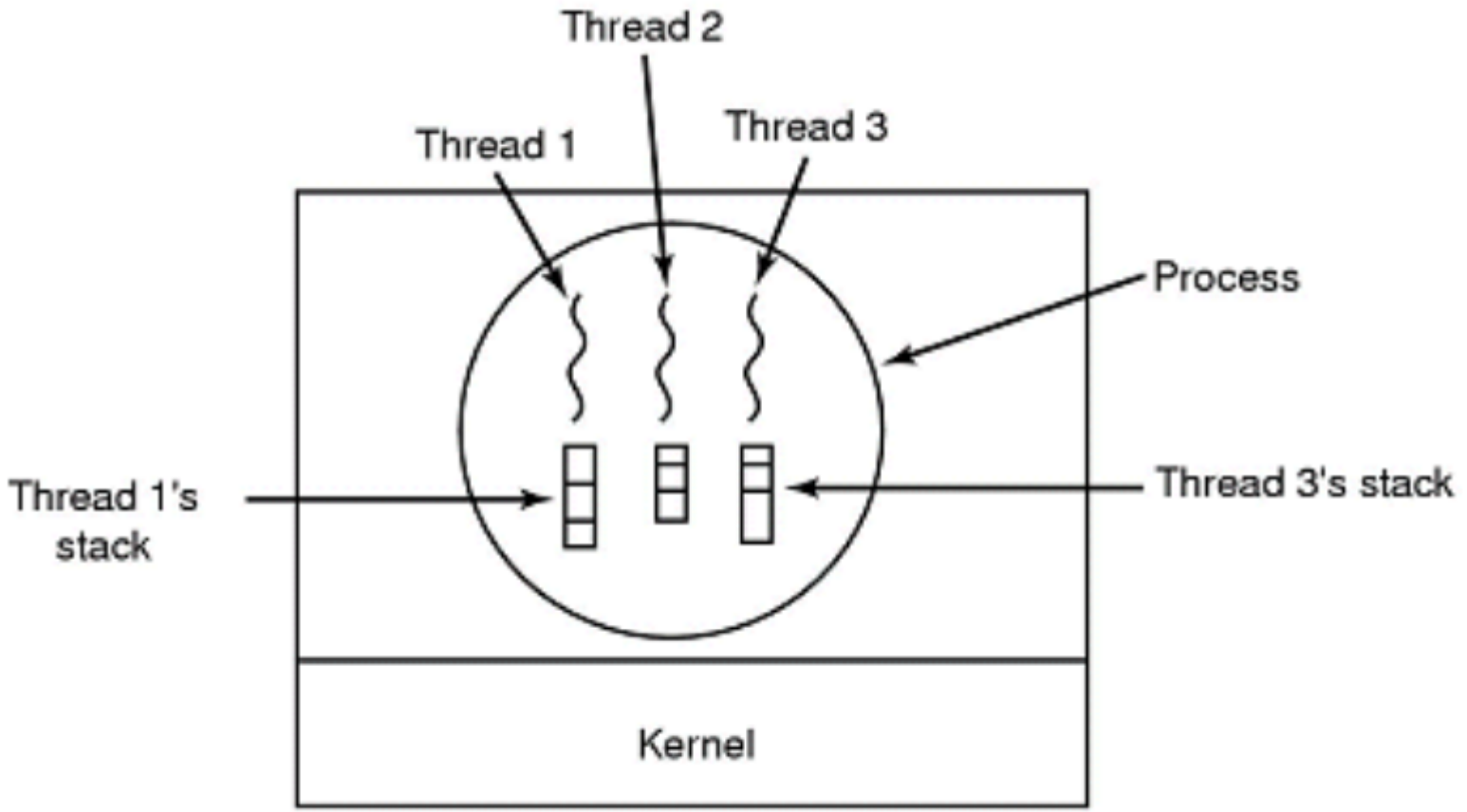
- Ein Prozess ist ein Computerprogramm zur Laufzeit
- Betriebssystem stellt Ablaufumgebung zur Verfügung
  - Befehlszeiger
  - Prozessor-Register
  - Stack (Variablen, Rücksprungadressen)
  - Heap (dynamischer Speicher)

Prozesszustände

- Running – Prozessor führt Prozess aus
- Ready – Prozess ist bereit und wartet auf Ausführung am Prozessor
- Blocked – Prozess ist blockiert, weil Hardware- oder Softwarebetriebsmittel fehlen
  - zB Daten von Festplatte müssen erst geladen werden



Threads (1)


Thread 2
Thread 1
Thread 3
Process
Thread 1's stack
Thread 3's stack
Kernel

Threads (2)

- Ein Prozess hat einen oder mehrere Threads
- Mehrere Threads ermöglichen Parallelisierung innerhalb eines Prozesses
- Fehlen Hardware- oder Softwarebetriebsmittel, dann wird einzelner Thread blockiert anstatt der ganze Prozess
- Teilen sich Speicher mit anderen Threads innerhalb des gleichen Prozesses
- Haben aber eigenen Befehlszeiger und Stack

Threads - Anwendungsbeispiele

- MS Word hat eine Thread der Text formatiert, einen der Eingaben verarbeitet, etc.
- Server weisen jede Anfrage einem Thread zu der diese abarbeitet
- Benutzer fordert Datenexport an, dieser wird im Hintergrund in neuem Thread abgearbeiet
- Cronjobs die stündlich aufwachen und eine Prozedur abarbeiten