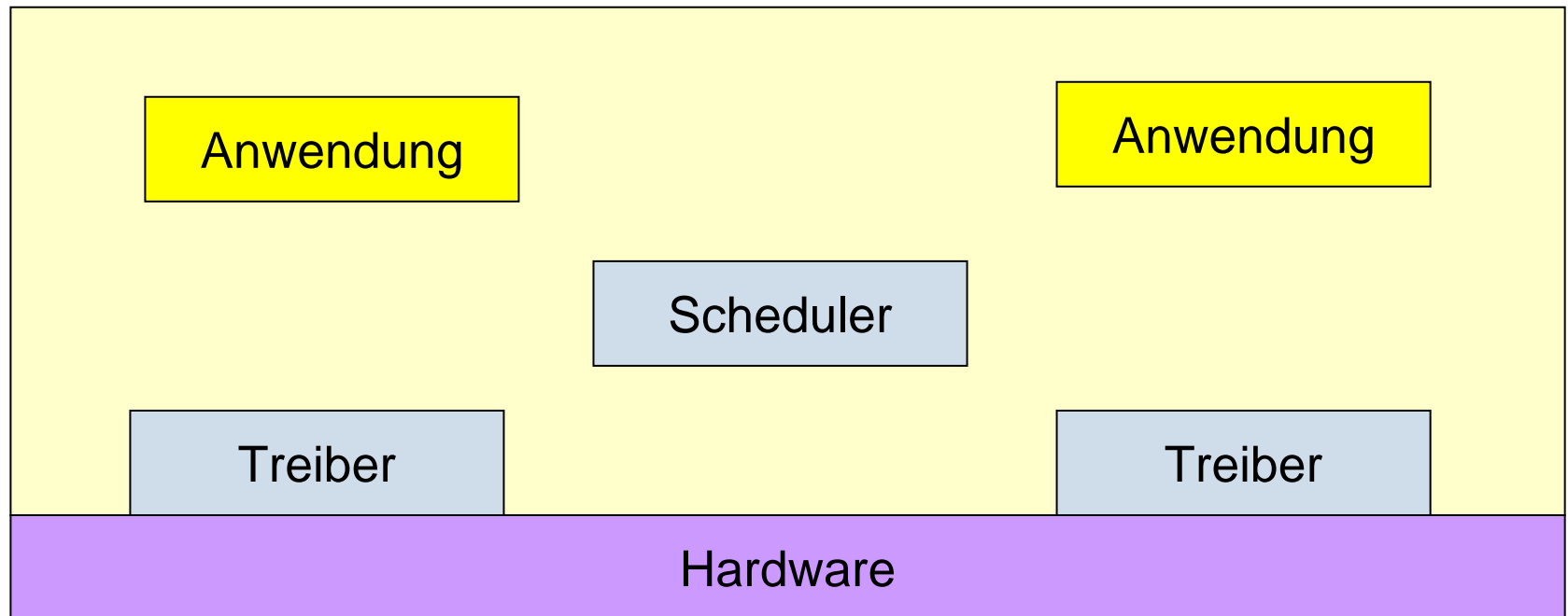


Betriebssysteme

Aufbau eines Betriebssystems

Prof. Dr.-Ing. Torben Weis
Universität Duisburg-Essen

Monolithische Systeme



- Keine Trennung zwischen OS und Anwendung
- Geeignet für kleine eingebettete Systeme
- Beispiel: TinyOS

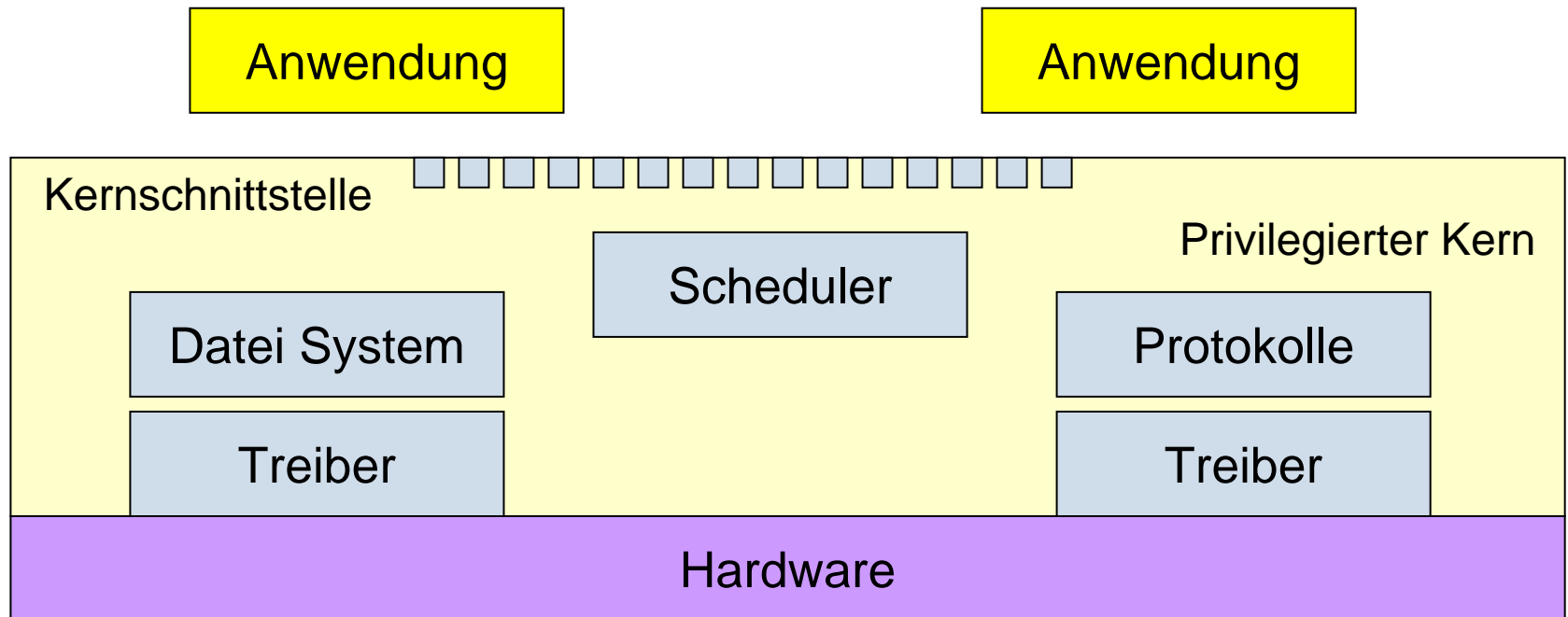
Monolithische Systeme

- Mobiltelefon, PDA, Autos, Maschinen, Sensoren
 - Bedarf an eingebetteten Systemen
 - -> spezielle, maßgeschneiderte Betriebssysteme
 - Weitaus größere Stückzahlen als z.B. PCs
- Unterschiedlichste Anforderungen
 - Echtzeit
 - Zuverlässigkeit
 - Sparsamkeit
 - Sicherheit
- Beispiel
 - TinyOS



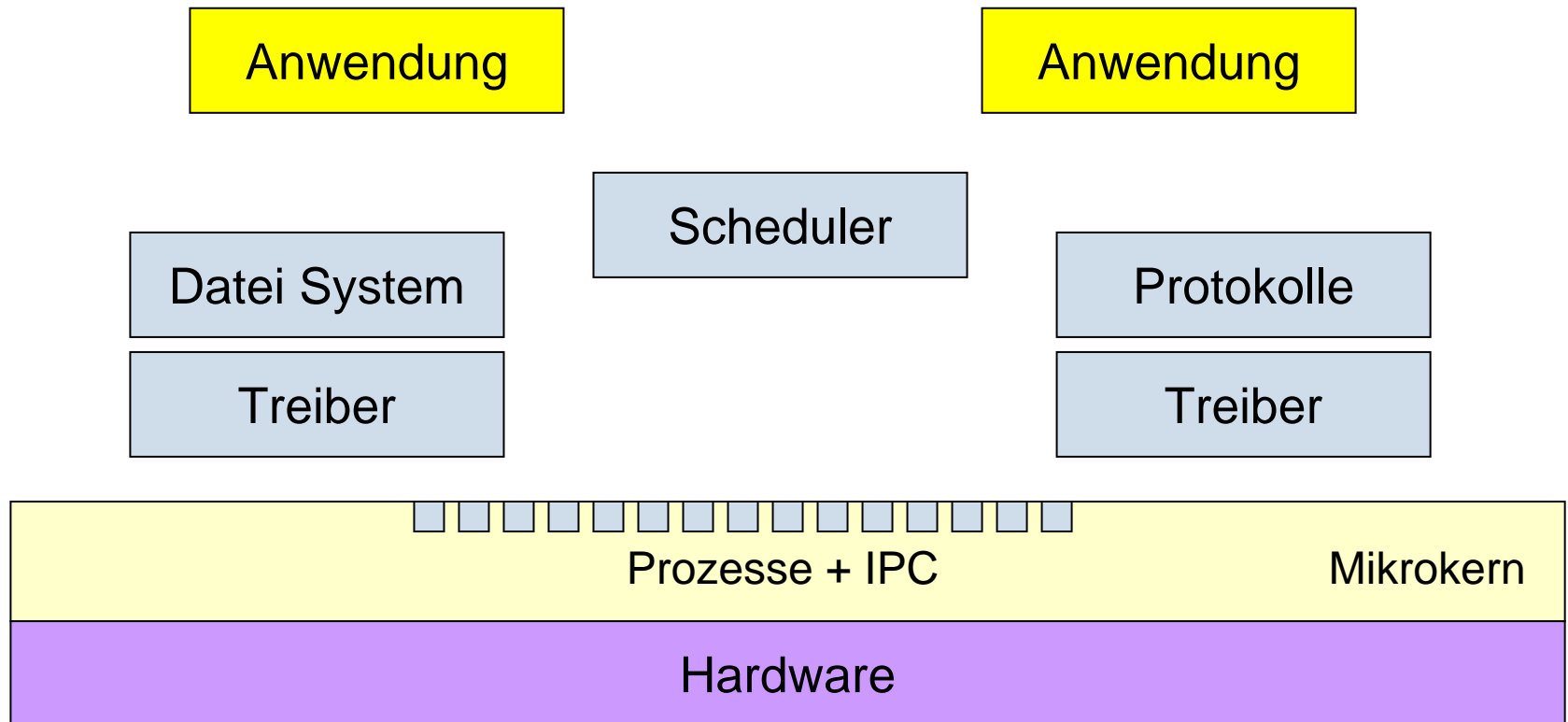
ESB
Sensor Board
FU Berlin

Monolithisches Betriebssystem



- Trennung zwischen Anwendung und Kern
- Kein Schutz der Kernkomponenten untereinander
- Der ganze Kern muss vertrauenswürdig sein
- Keine Rettung bei Absturz einer Kernkomponente

Mikrokern Betriebssystem



- Der Kern umfasst nur Prozessmanagement (Scheduling, Dispatching) und Interprozesskommunikation.

Vorzüge der Mikrokernarchitektur

- Klare Kernschnittstelle begünstigt modulare Struktur
- Realisierung der Dienste außerhalb des Kerns
 - Schafft mehr Sicherheit und Stabilität, da der Kern durch fehlerhafte Dienste nicht in Mitleidenschaft gezogen wird
 - Verbessert die Flexibilität und Erweiterbarkeit, da Dienste beliebig hinzugefügt oder weggenommen werden können, selbst im laufenden Betrieb.
- Der sicherheitskritische Teil des Systems (Kern) ist relativ klein und kann daher besser verifiziert werden
- Nur der Kern läuft im privilegierten Zustand
- Koexistenz mehrerer alternativer Schnittstellen zwischen Betriebssystem und Anwendungen

Nachteile der Mikrokernarchitektur

- i.d.R. langsam

Warum?

- Zusammenspiel der Komponenten außerhalb des Kerns erfordert mehr Interprozesskommunikation und daher mehr Systemaufrufe und Prozesswechsel

Allgemeine Entwurfsprinzipien

- KISS (keep it small and simple; keep it simple, stupid)
- Occam's Razor: "Plurality should not be assumed without necessity."
(William of Ockham, ca. 1285–1349)
- "There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."
(C.A.R. Hoare)
- "Everything should be made as simple as possible, but no simpler"
(Albert Einstein, 1879–1955)
- "Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away."
(Antoine de Saint-Exupéry, 1900–1944)

Modularisierung

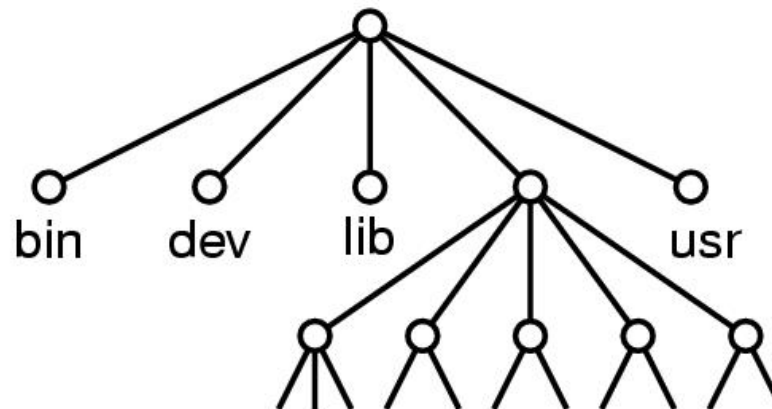
- **Das System wird zerlegt in eine Menge Module derart, dass**
 - die Interaktion (Informations- und Kontrollflüsse) innerhalb des Moduls hoch ist,
 - die Interaktion zwischen den Modulen gering ist,
 - die Schnittstellen zwischen den Modulen einfach sind,
 - die Module in ihrer Größe und Komplexität überschaubar und beherrschbar sind
- **Das Prinzip kann hierarchisch angewendet werden**

Modularisierung des Kerns

- Monolithischer Kern
 - Sichtweise zur Zeit der Ausführung
- Modularisierter Monolithischer Kern
 - Sichtweise zur Zeit der Implementierung
 - Bessere Wartbarkeit des Codes
 - Einfachere Fehlersuche
- Beispiel
 - Linux Kernel
 - Linux kann Module auch zur Laufzeit nachladen
 - -> Ähnlich der Mikrokernarchitektur
 - Aber nicht so langsam wie Mikrokerne

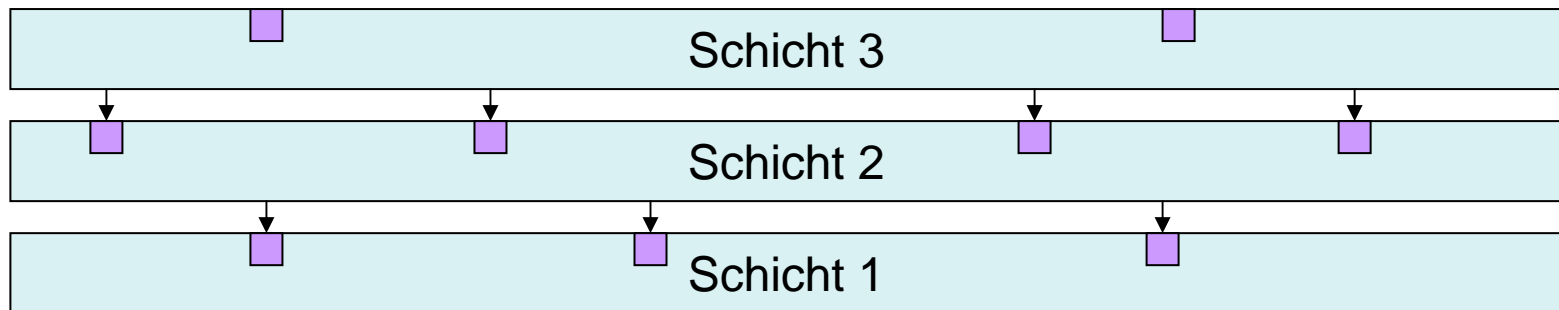
Hierarchisierung

- Baumartige Organisation von gleichartigen Elementen
- Ziele
 - Skalierbarkeit
 - Beherrschung der Komplexität



Schichtung (Layering)

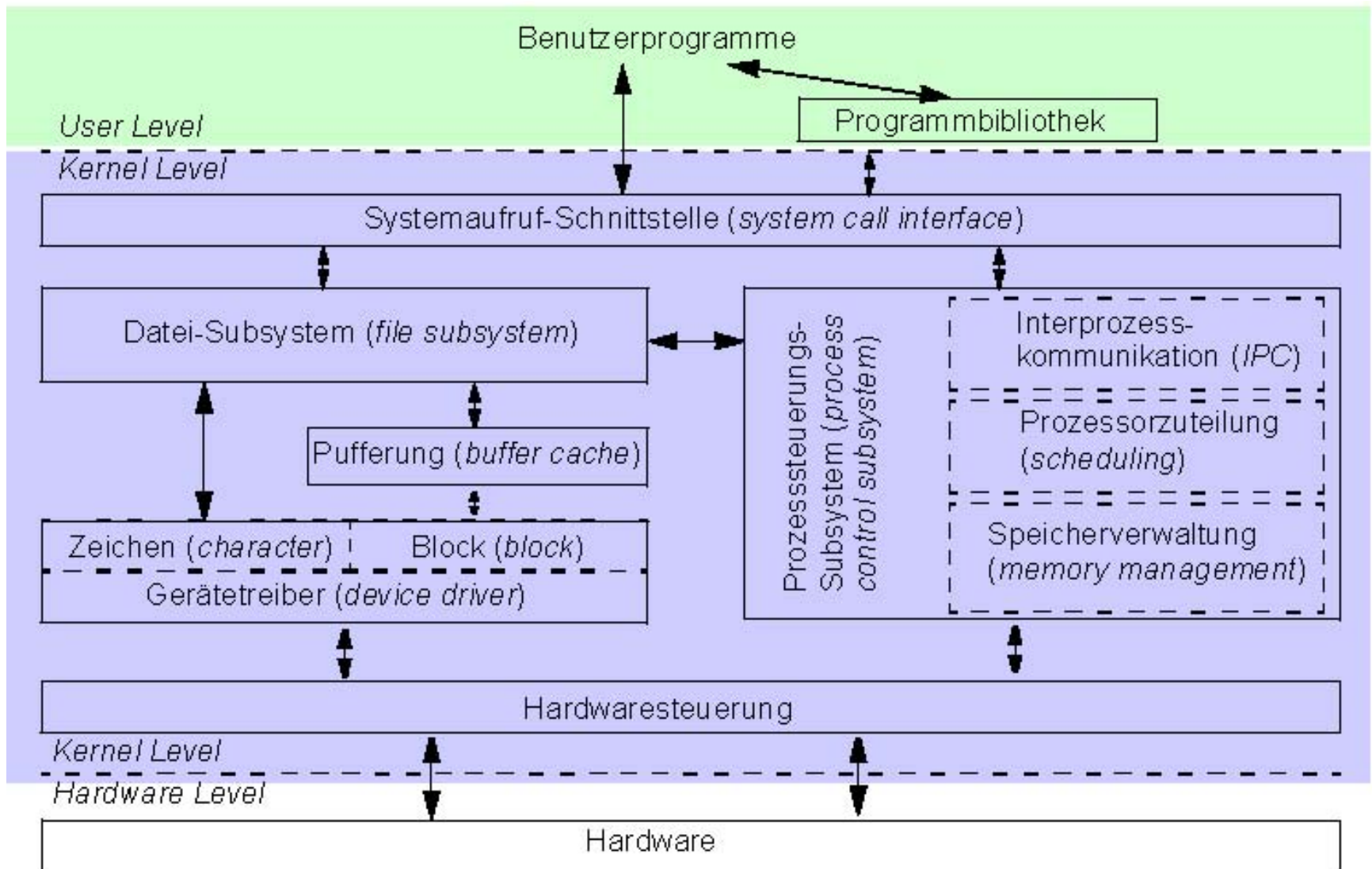
- Zerlegung des Systems in Schichten
 - Einfachere, universelle Funktionen weiter unten
 - Komplexere, spezifische Funktionen weiter oben
- Jede Schicht ist eine Abstraktion tiefer liegender Schichten
- Jede Schicht besitzt eine Schnittstelle, die von höheren Schichten benutzt werden können



Schichtung des Betriebssystems

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

- THE-System, E.W. Dijkstra (1968)
- Technische Hogeschool Eindhoven



Trennung von Strategie und Mechanismus

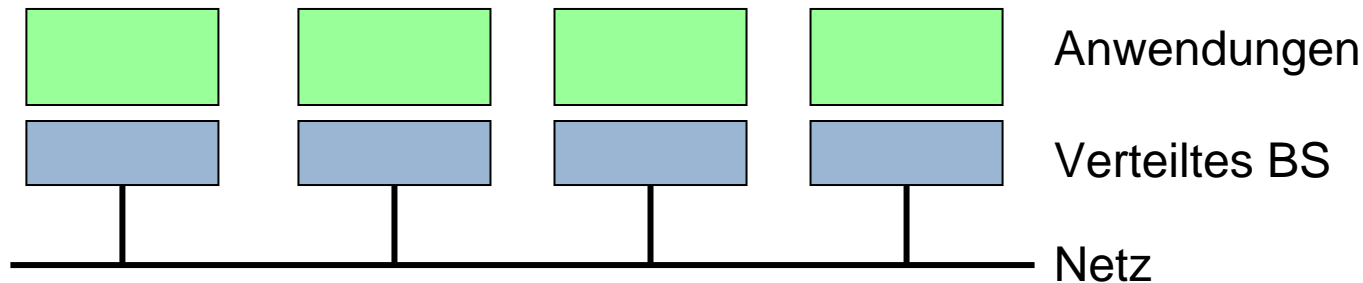
- Mechanismen stellt das BS bereit
 - Scheduling
 - Paging
 - Security
- Strategie (Policy) nutzt Mechanismen
 - Video Player bekommt mehr Rechenzeit
 - Systemtabellen nicht auslagern
 - Fritz darf auf gar nichts zugreifen
- Anwendungsneutralität erfordert Kompromisse
- Untere Schichten implementieren Mechanismen, die in höheren Schichten anwendungsspezifisch parametrisiert werden können

SPOT-Rule (Single Point of Truth)

- Keine Kopien oder Wiederholungen
- Für Code: Jede Funktionalität wird genau einmal implementiert
- Für Daten: Jede Information, die das System verwaltet, hat genau eine Repräsentation.
- Anwendung von SPOT vermeidet Inkonsistenzen

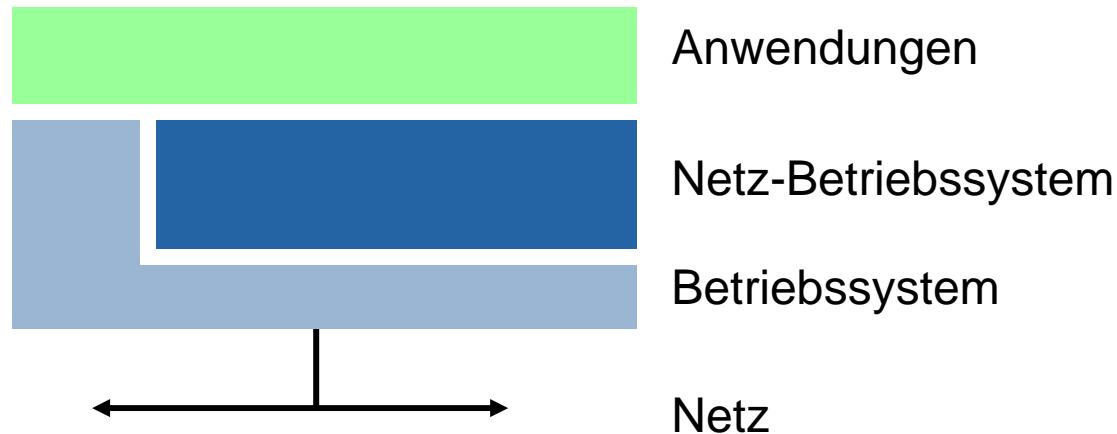
Verteilte Betriebssysteme

- Kommunikation über Rechnernetz
- Transparenz der Verteilung
- Knoten sind nur bedingt autark

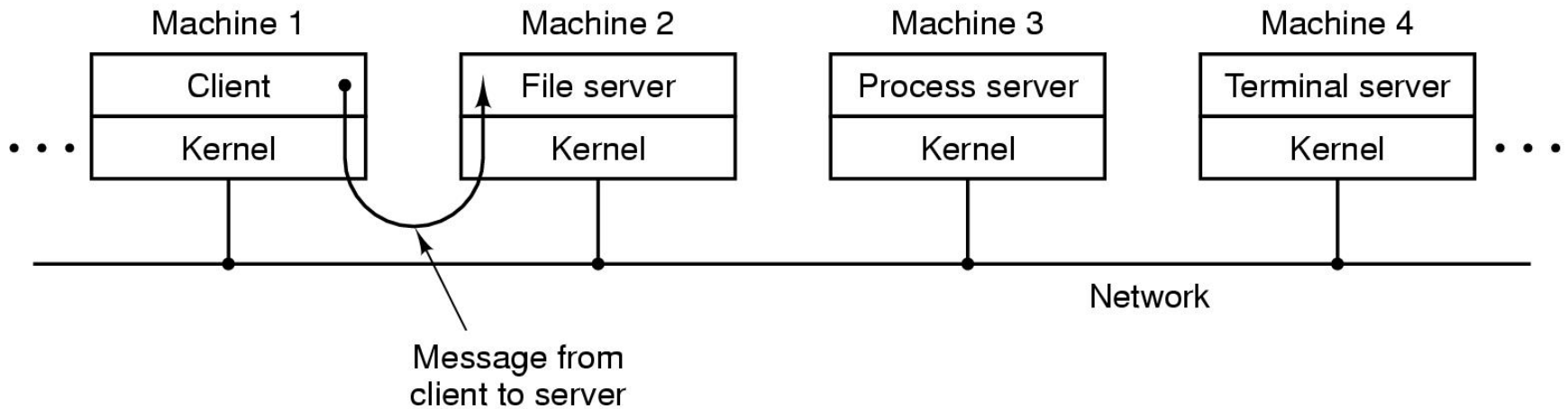


Netz-Betriebssystem

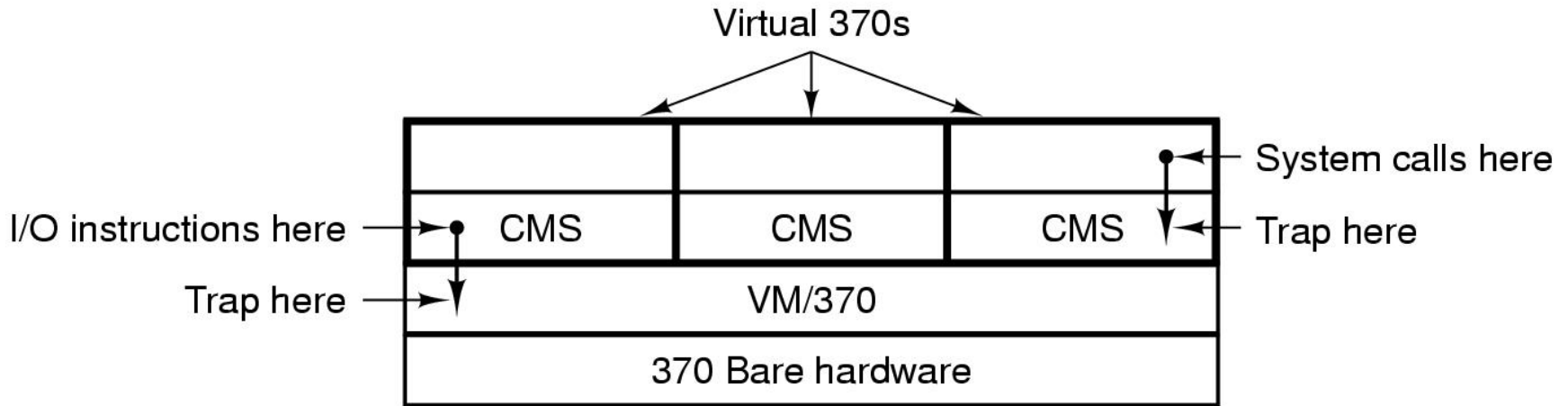
- Zusatz zu existierendem BS
- Meist praktikabler als pures Verteiltes BS



Client-Server Systeme



Virtualisierung



- Virtueller Computer wird zur Verfügung gestellt
 - Negativ: Host-Rechner braucht sehr viel Speicher
 - Positiv: Virtuelle Maschine kann man anhalten, kopieren, verschieben, ...

Virtualisierung

- .NET und Java haben auch eine virtuelle Maschine
- Es wird eine virtuelle (und einheitliche) Betriebssystem-Schnittstelle geboten
 - Vorteil: Anwendung wird BS-unabhängig
 - Nachteil: Kleinster gemeinsamer Nenner

