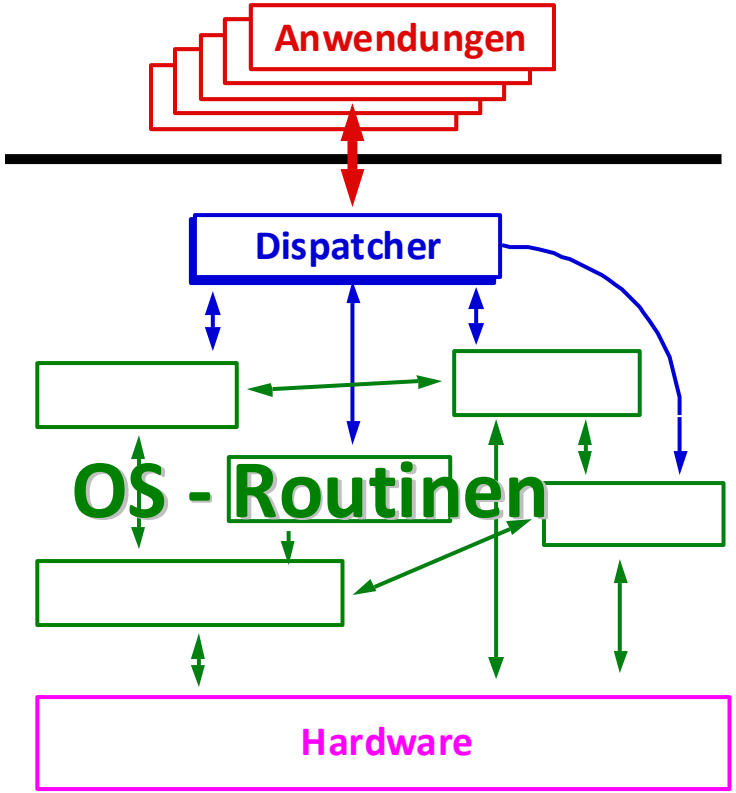
15. Architekturen

Michael Schöttner
Betriebssysteme und Systemprogrammierung

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

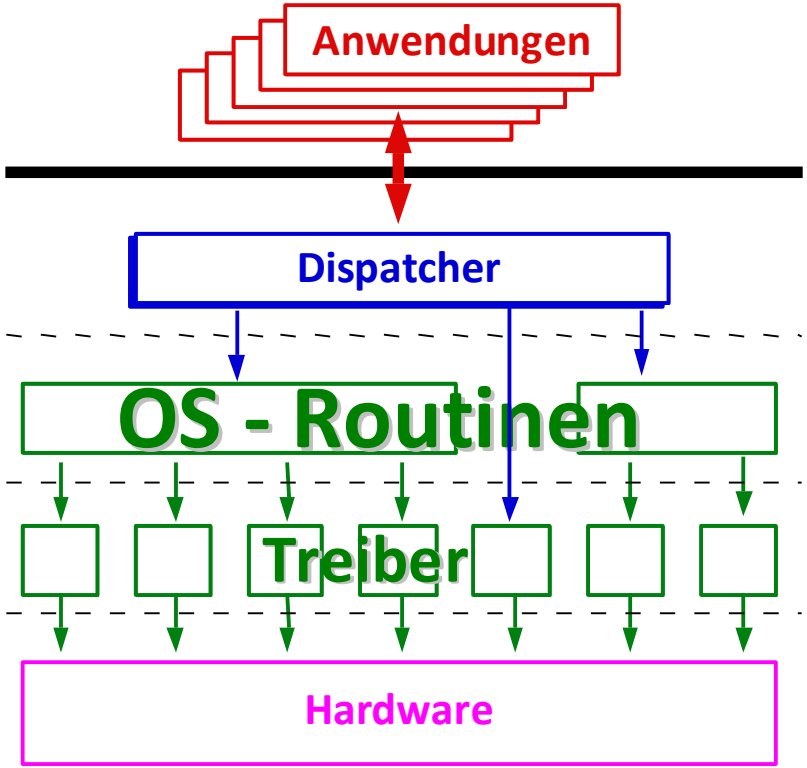
15.1 Monolithische Betriebssysteme

- Sammlung von Routinen, ohne Hierarchie, und Schichtung.
- Jede Prozedur kann beliebige andere aufrufen und Datenstrukturen ändern.
- Erweiterungen sind schwer abzuschätzen.
- Zum Beispiel:
  - Ältere UNIX-Systeme
    - auch Linux zu Beginn
  - Batch-Betriebssysteme
  - IBM OS/360 ...



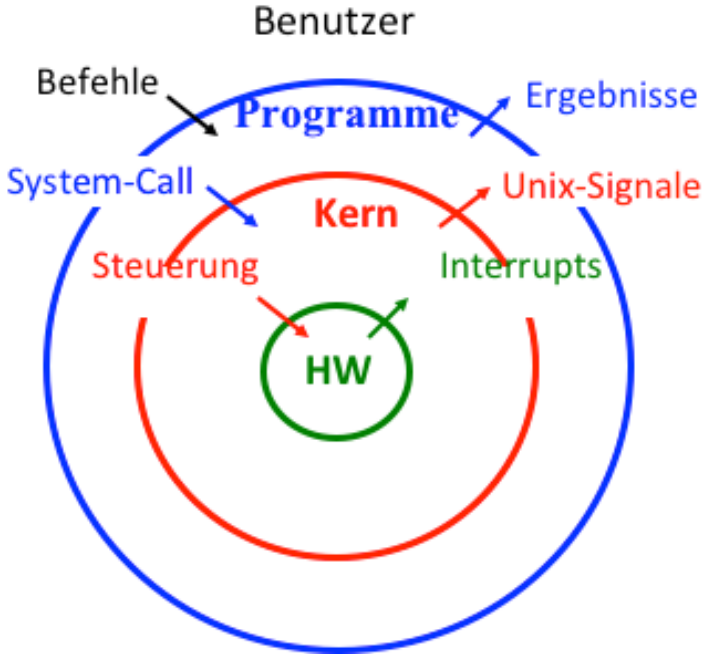
15.2 Geschichtete Architekturen

- Verschiedene Abstraktionsebenen respektive Schnittstellen:
  - Programmierschnittstelle für die Anwendungsprogramme
  - Treiberschnittstelle
  - Hardwareschnittstellen



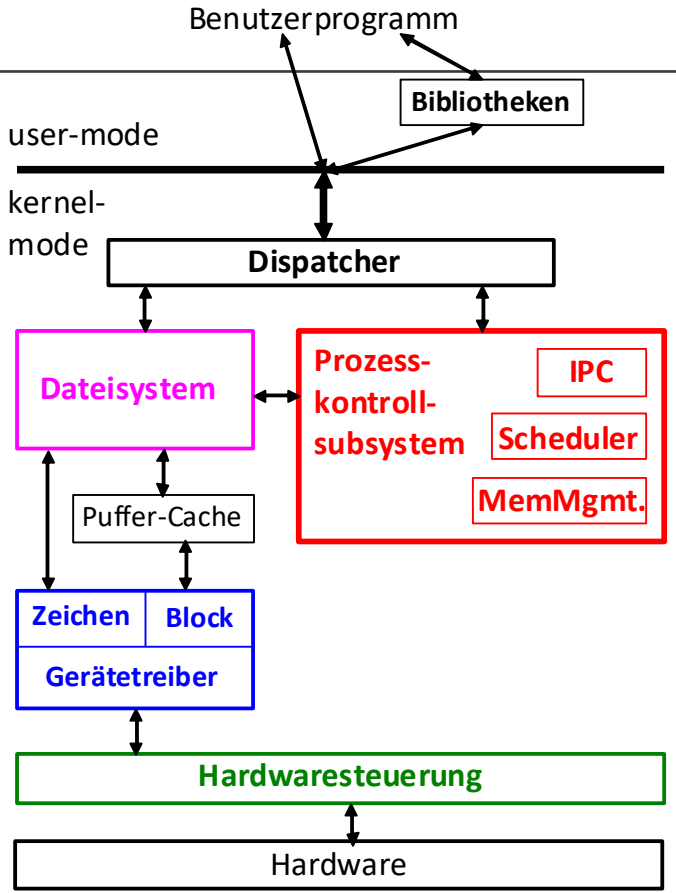
Hierarchisches Modell

- Auch **Schalenmodell** genannt
- Anordnung der Funktionen
  in einer Folge konzentrischer Ringe.
- Innere Ringe sind privilegierter als Äußere
- Aufrufe über Ringgrenzen hinweg
  mithilfe eines Software-Interrupts:
  - kontrollierter Übergang zw. Privilegstufen
  - Überprüfung der Parameter
  - Unterstützung durch HW
  - x86 CPUs max. 4 Ringe
    (i.d.R. nur 2 genutzt)



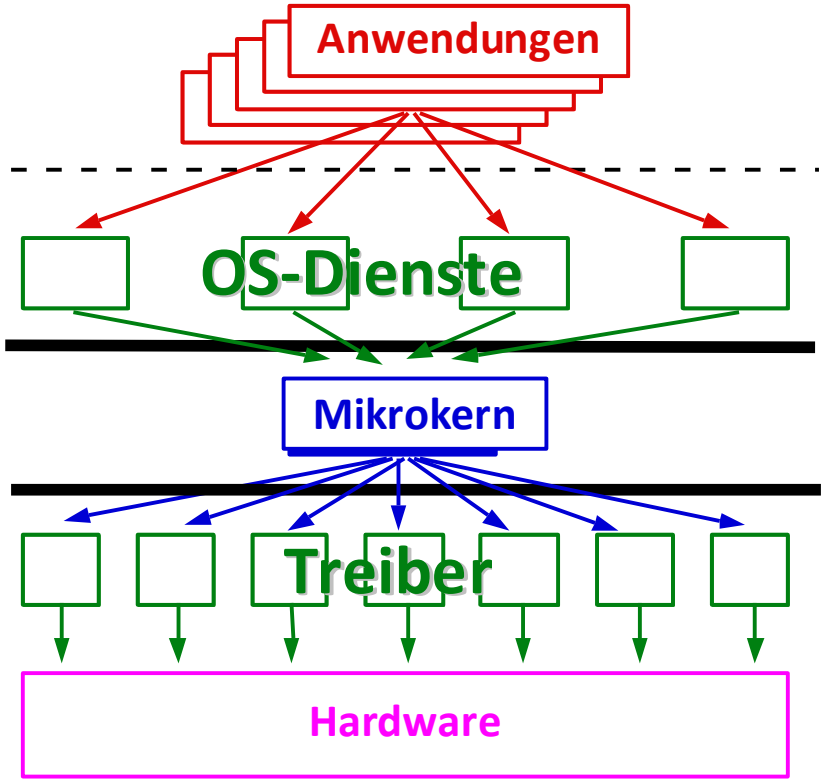
Traditioneller UNIX-Kern

- Trennung User-/Kernel Space
- Monolithischer Aufbau
- Linux bietet zusätzlich dynamisch ladbare Module
- Hardwareabstraktion



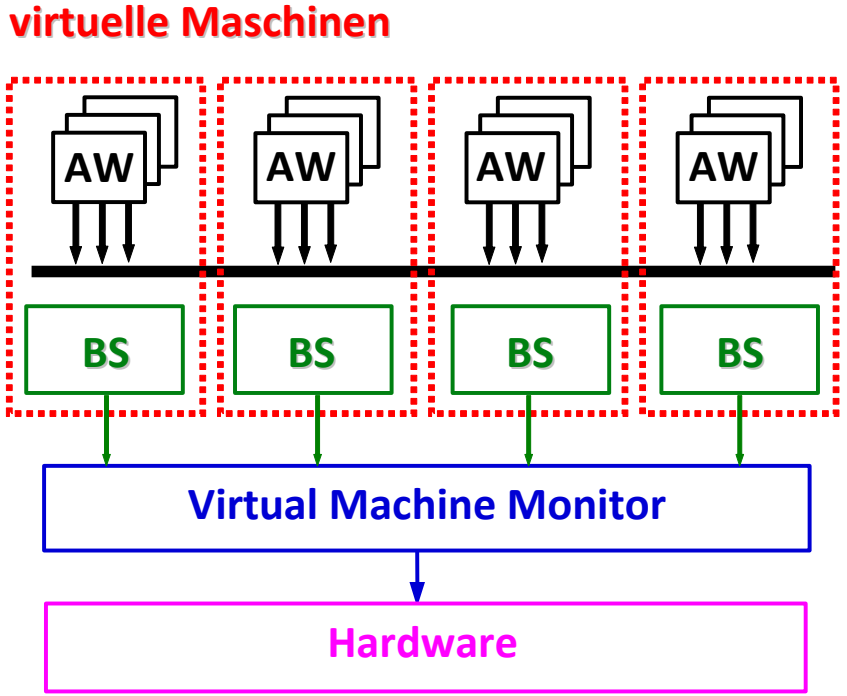
15.3 Mikrokern Systeme

- Zuverlässiger (kleiner) Mikrokern:
  - Prozessumschaltung & IPC
  - Speicherverwaltung & Sicherheit ...
- Treiber
  - geringer privilegiert als Kern
- OS-Dienste: im User-Mode
  - z.B. Dateisystem, GUI, …
- Beispiel: Mach, L4, TUDOS
- Mehr Stabilität, aber erheblich mehr Aufwand durch häufige Interprozesskommunikation.



15.4 Virtuelle Maschinen

- **Mehrere Gast-Betriebssysteme** auf einer Maschine ausführbar
- Virtuelle Maschinen sind Kopien der zugrunde liegenden Hardware:
  - CPU, Interrupts, E/A,
  - Kern- & Benutzermodus, ...
  - Nachteil: Effizienzverlust
- **Virtual Machine Monitor** (Host-BS):
  - läuft direkt auf Hardware
  - fängt Zugriffe auf Hardware ab und übernimmt deren Ausführung.
  - Heute **Typ-1-Hypervisor genannt**.



Voraussetzung für Virtualisierung

- Erste systematische Untersuchung von Popek & Goldberg 1974
  → Prozessor muss “virtualisierbar“ sein
- Gast-Betriebssystem muss trotz Ausführung im User-Mode sensitive Instruktionen ausführen können.
- **Beim Ausführung eines privilegierten Befehls im User-Mode muss ein Trap erzeugt werden.**
- Die „alte“ IBM-Hardware besaß diese Eigenschaft schon 1972
- Intel-CPUs hatten diese Eigenschaft lange nicht. Gelöst wurde das Problem ab 2005 durch Einführung von Intel-VT und AMD-V.

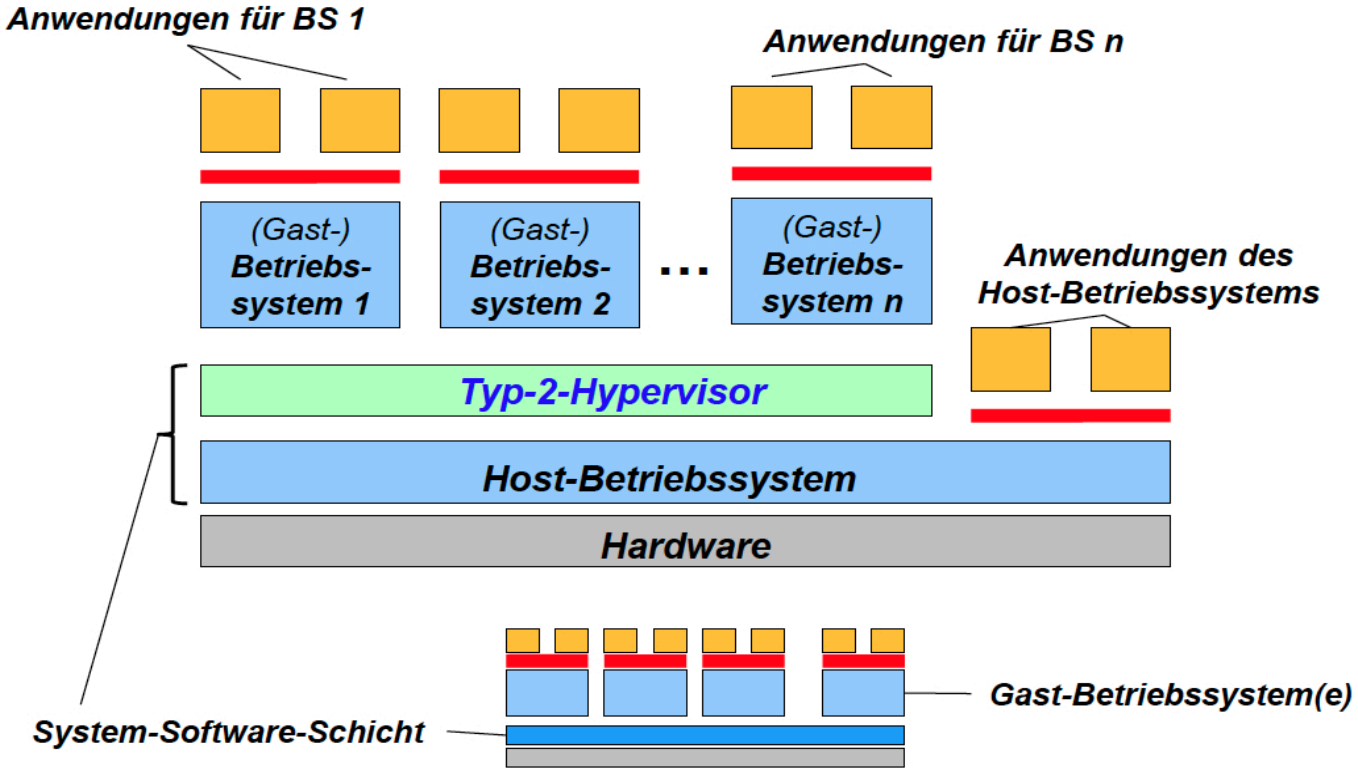
Typ-1 Hypervisor

- Hypervisor wurde früher Virtual Machine Monitor (VMM) genannt
- **Hypervisor läuft direkt auf Hardware, fängt Zugriffe auf Hardware ab und übernimmt deren Ausführung.**
- Bietet mehrmals die unmittelbare Hardware-Schnittstelle nach oben an, von denen jede eine virtuelle Maschine jeweils mit Hardware-Eigenschaften darstellt (Kern- und Nutzer-Modus, Ein-/Ausgaben usw.).
- Beispiele: VM/370 (IBM, 1972), VMware ESX Server, Microsoft Hyper-V, …

Typ-2 Hypervisor

- Hypervisor läuft wie das Gast-BS im User-Mode und nutzt die Ressourcen, welche das Host-OS bereitstellt.
- Hypervisor analyisert Binärcode und übersetzt privilegierte Befehle in Aufrufe an Funktionen des Hypervisors der die Befehle emuliert.
- Diese dynamische Übersetzung von sensitiven Befehlen wird auch als „binary translation“ bezeichnet
- Beispiele: VMWare Workstation, Qemu,, ...
- Bem.: erlaubt Virtualisierung ohne HW-Unterstützung

Typ-2 Hypervisor



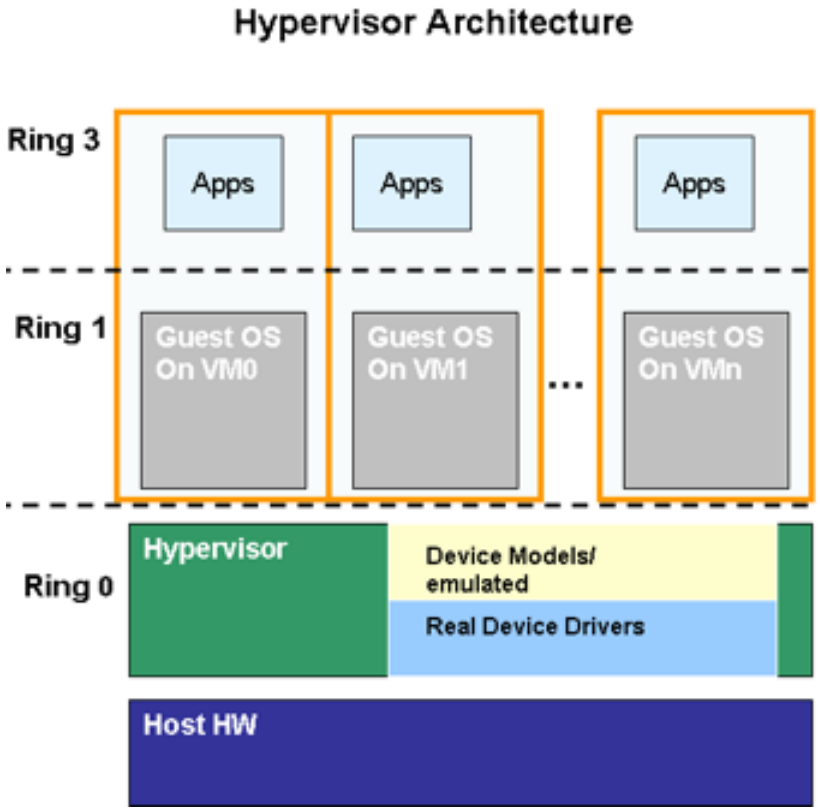
Dr. Hans-Albrecht Schindler, TU Illmenau

Virtualisierung auf Betriebssystem-Ebene

- Ein Betriebssystem-Kernel stellt mehrere identische Laufzeitumgebungen (Container) bereit
- Host-BS virtualisiert einen physikalischen Server, sodass mehrere virtuelle Server auf einer Maschine, isoliert voneinander ausgeführt werden können.
- Beispiele: Docker, Linux VServer, OpenVZ, ...
- Vorteil: sehr effizient
- Nachteil:
  - keine grundlegend anderes Betriebssystem ausführbar
  - kein eigener Kernel und keine eigenen Treiber für Gast-Betriebssysteme

Para-Virtualisierung

- Quelltext des Gast-BS wird modifiziert:
  - führt privilegierte Befehle in Ring-1 oder 3 aus, indem ein spezielles API des VMM genutzt wird
  - VMM muss daher diese Befehle nicht mehr erkennen und übersetzen
- VMM verarbeitet die Virtualisierungs-befehle u. übersetzt diese auf die echte Hardware.



Para-Virtualisierung

- Zwei Stufen:
  - 1. Installation von Para-Virtualisierungstreiber:
    → für einige Geräte, z.B. für Grafikkarten, Beispiel: VMWare
  - 2. Gast-Kernel modifizieren:
    → Kern mit Para-Virtualisierungstreibern kompilieren, z.B. Linux auf XEN

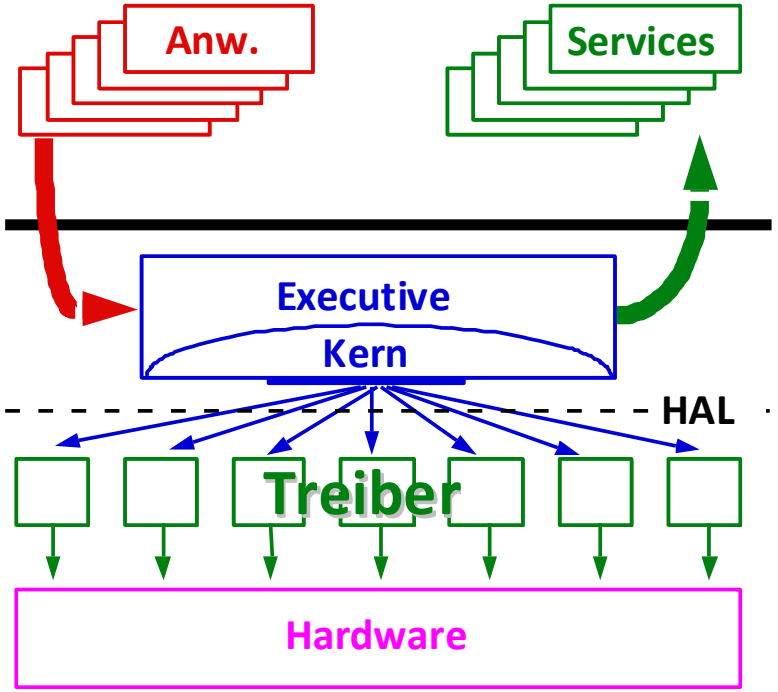
- Bewertung: schneller, aber Gast-BS muss modifiziert werden

Hardware-unterstützte Virtualisierung

- Gast-BS läuft wie gewohnt in Ring-0
- VMM benutzt CPU-Erweiterungen, um
  privilegierte Befehle abzufangen und zu emulieren
- VMM läuft in einem höher-privilegiertem Ring, dem virtuellen Ring -1
- Beispiele: Intel-VT und AMD-V
- Vorteile:
  - kann Gast-OS ohne Modifikationen ausführen
  - vergleichsweise schnell
- Nachteile:
  - für optimale Geschwindigkeit muss Gast-BS
    aber dennoch modifiziert werden

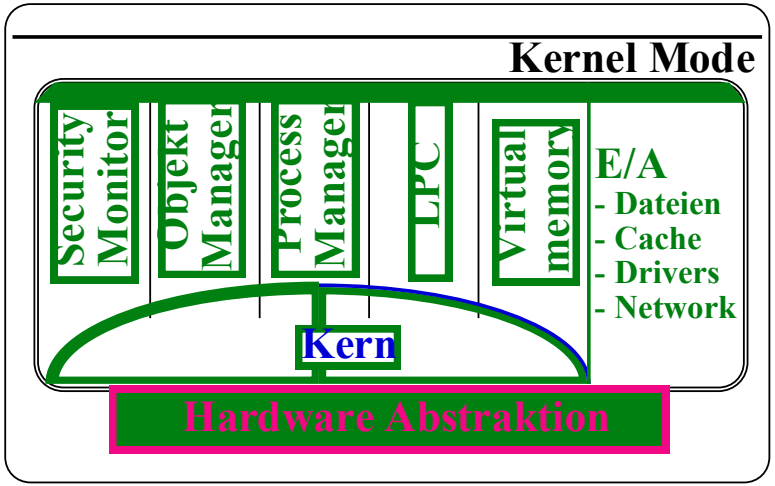
15.5 Client / Server Architekturen

- Aufteilung in Client- und Server-Prozesse.
  - Keine Privilegien für Dienste im User-Mode
- Validierung der Aufrufparameter zum Dienst in Executive:
  - gültige Handles
  - Zugriffsrechte, ...
- Aber Aufruf von Diensten ist aufwändig
  - Mehrfaches Umschalten zwischen User-/Kernel-Mode



Fallbeispiel: „Executive“ in Windows NT

- BS-Funktionen im Kern
- Security Monitor kooperiert mit Security-Subsystem.
- Object Manager verwaltet Betriebssystem-Ressourcen.
- "Local Procedure Calls" (LPC) für Aufrufe zw. Adressräumen (über Shared-Memory und Events)
- E/A-Manager: verwaltet und ruft Treiber:
  - asynchrone Kommunikation & Umkopieren von Daten
  - optional Direct-I/O: schneller, da ohne Umkopieren



Subsysteme in Windows NT

- Kommunikation mit Subsystemen über den LPC Manager in der Executive.
- Kommunikation zw. Subsystemen ebenfalls per LPC (Local Procedure Call).
- Win32-Subsystem:
  - DLLs beim Klienten: User32, Kernel32, GDI32
  - LPC zum Win32-Subsystem-Prozess CSRSS.exe (= Client/Server Runtime Subsystem process)