(Heim-)Server-Virtualisierung

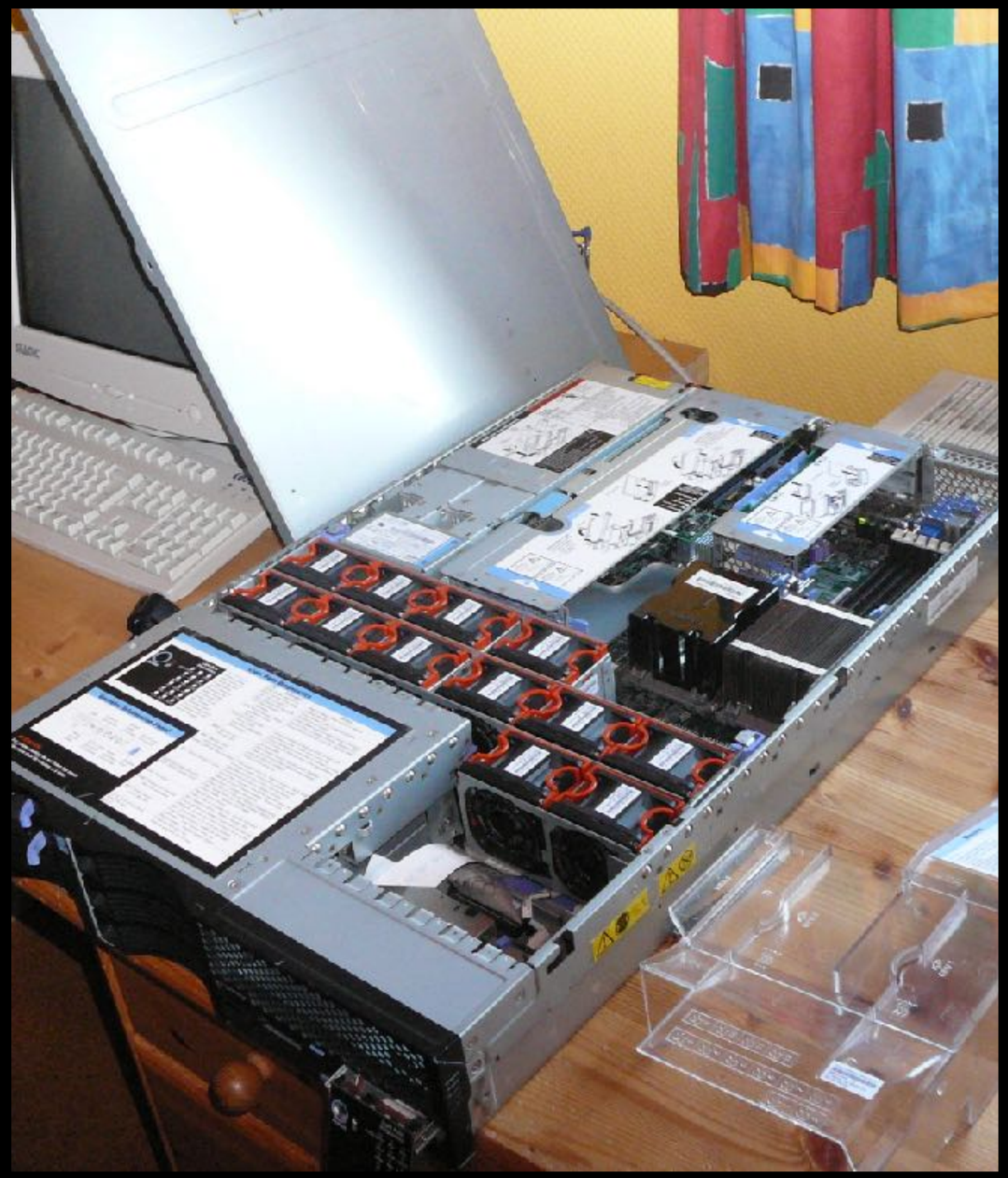
“Warum?” und “Wie?” mit QEMU/KVM und ZFS

Roter Faden

1. Warum?
2. Technische Grundlagen
3. Vorstellung: QEMU / KVM
4. Vorstellung: ZFS
5. Aufbau und Betrieb des SW-Stacks
6. Entscheidungen für einen eigenen Server
7. Feldstudie: 3 Jahre virtualisierter Heimserver

Hintergrund

- Erste 19”-Server im “Kinderzimmer”
- früh in den Linux-Topf gefallen
- Schwerpunkt: Server, Netzwerk-Infrastruktur & Virtualisierung
- B.o.Sc. Informatik FH Trier
- aktuell: M.o.Eng. Informatik - IT-Sicherheit und Zuverlässigkeit Uni Lübeck
- HiWi Administration IMBS



1. Warum?

1.1 Konsolidierung

- Im RZ:
  - Reduktion von Komplexität
  - Effiziente Nutzung von Ressourcen
- Zu Hause:
  - Management des heimischen Technik-Zoos
  - Bietet sicheren & kostenlosen Spielplatz

1.2 HW- & Strom-Kosten

- Im RZ:
  - Lasten zeitlich selten stabil
  - viele Server überdimensioniert
  - Kabel und Platz

1.2 HW- & Strom-Kosten

- Zu Hause:
  - ein PI kommt selten allein
  - 99% Idle
  - einzelner HW-Upgrade-Path

1.3 Skalierbarkeit

- Im RZ:
  - Ressourcen-Pooling
  - Leistung / Server Mausklick
    - keine harten Limits
- Zu Hause:
  - Projektwachstum
  - einfache Ressourcen-Umverteilung

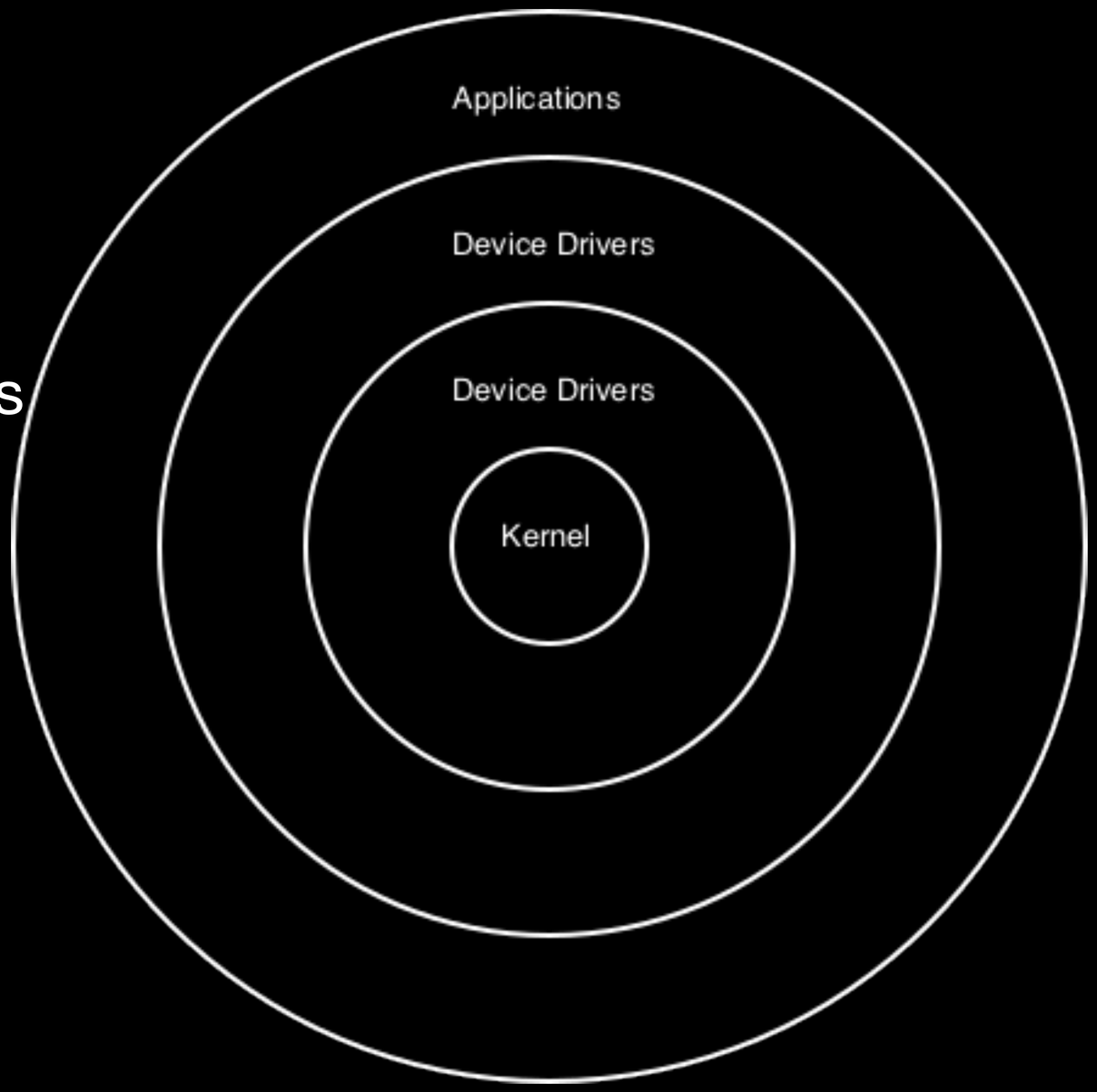
1.4 Portabilität

- Im RZ:
  - Plattform-Abstraktion
  - bringt Legacy in die Zukunft
    - Live-Migration nicht verteilter Anwendungen
- Zu Hause:
  - Hardware-Unabhängigkeit
  - Run where-ever you please

2. Technische Grundlagen

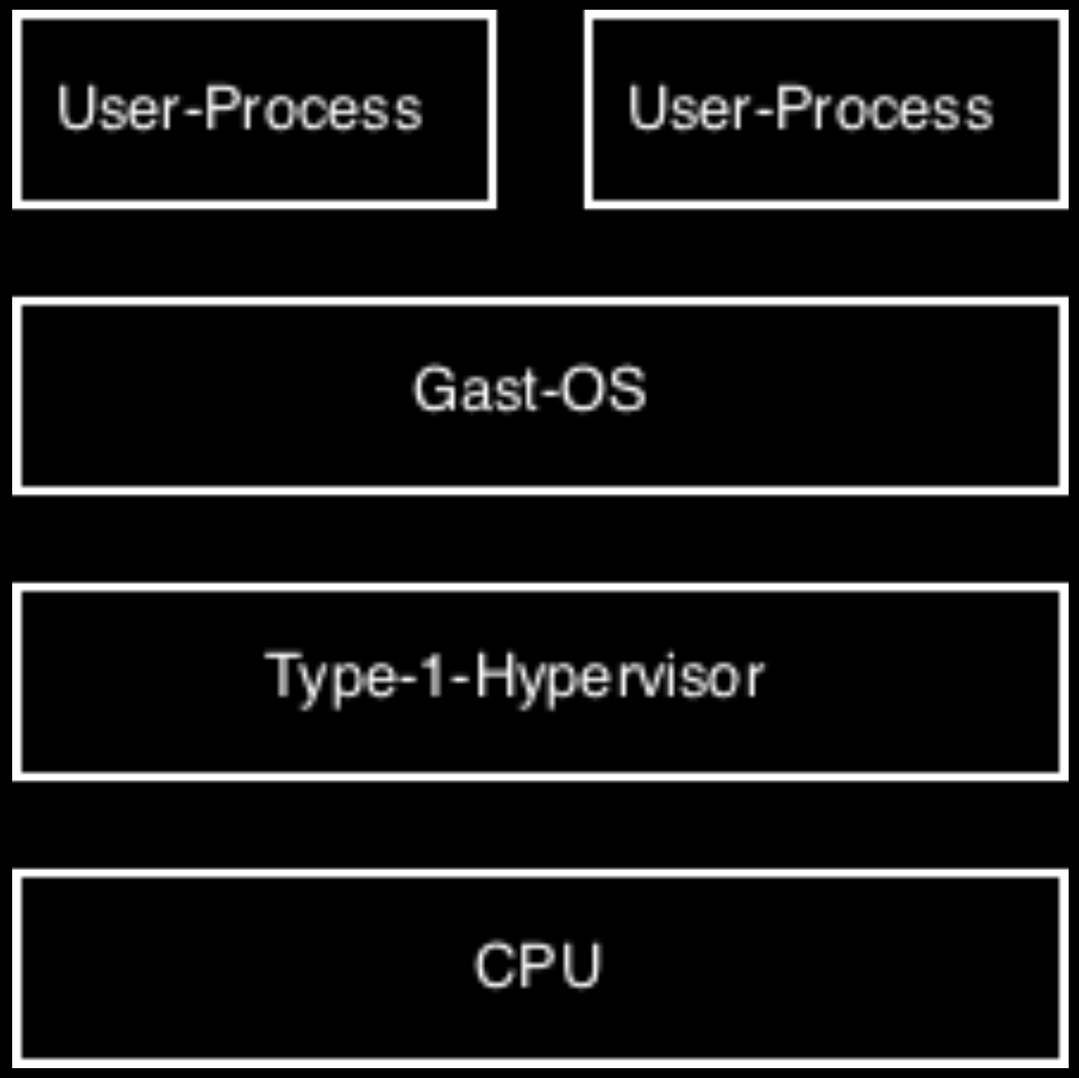
2.1 Privilegien im BS

- Ringe / Schichten
- Kommunikation zwischen Schichten über APIs
- System Calls / Traps / Interrupts



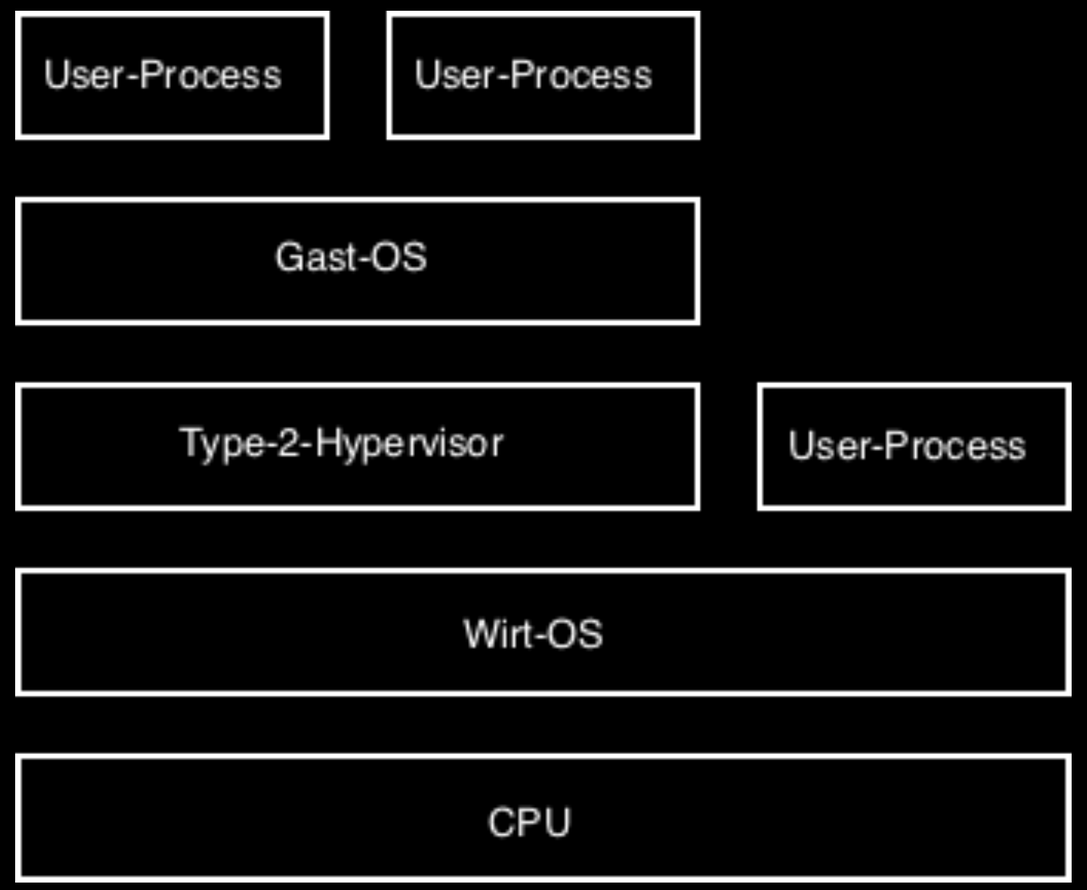
2.2 Type-1-Virtualisierung

- “Bare Metal” = Hypervisor ist BS
- z.B. VMWare ESX(i), MS Hyper-V, …
- Hardware muss unterstützt werden
- Address-Translation
- Maskierung von Traps



2.3 Type-2-Virtualisierung

- Läuft im Userspace eines BSs
- Address-Translation
- Dynamic Binary Translation
- Hypervisor nicht privilegiert
- Trap aus Hypervisor notwendig
- auch Emulation möglich



2.4 Paravirtualisierung

- OS muss angepasst sein
- Static Binary Translation
- Keine System-Calls / kein virtueller Kernel-Modus
- stattdessen API des Hypervisors
- API ähnlich mächtig wie HW-Interface

2.5 Nicht-Virtualisierungen

- z.B. Container / Docker / chroot / …
- reine Maskierung
- ein Kernel
- BS-Funktionen

3. Vorstellung: QEMU / KVM

Der Schlüssel zu zwei schlagenden Herzen

3.1 QEMU

- “Quick Emulator”
- Type-2 Hypervisor
- Emuliert verschiedene Architekturen (ARM, MIPS, PowerPC…)
  - Meist verwendet in x86-auf-x86
  - Unterstützt Hardware-Virtualisierung
- verfügbar auf Linux, BSD, MS Windows, macOS

3.1 QEMU

- Beinhaltet para-virtualisierte HW
- ermöglicht direkten Zugriff auf HW
- eigenes Image-Format mit Snapshots
- Überwiegend GPL, teilweise andere

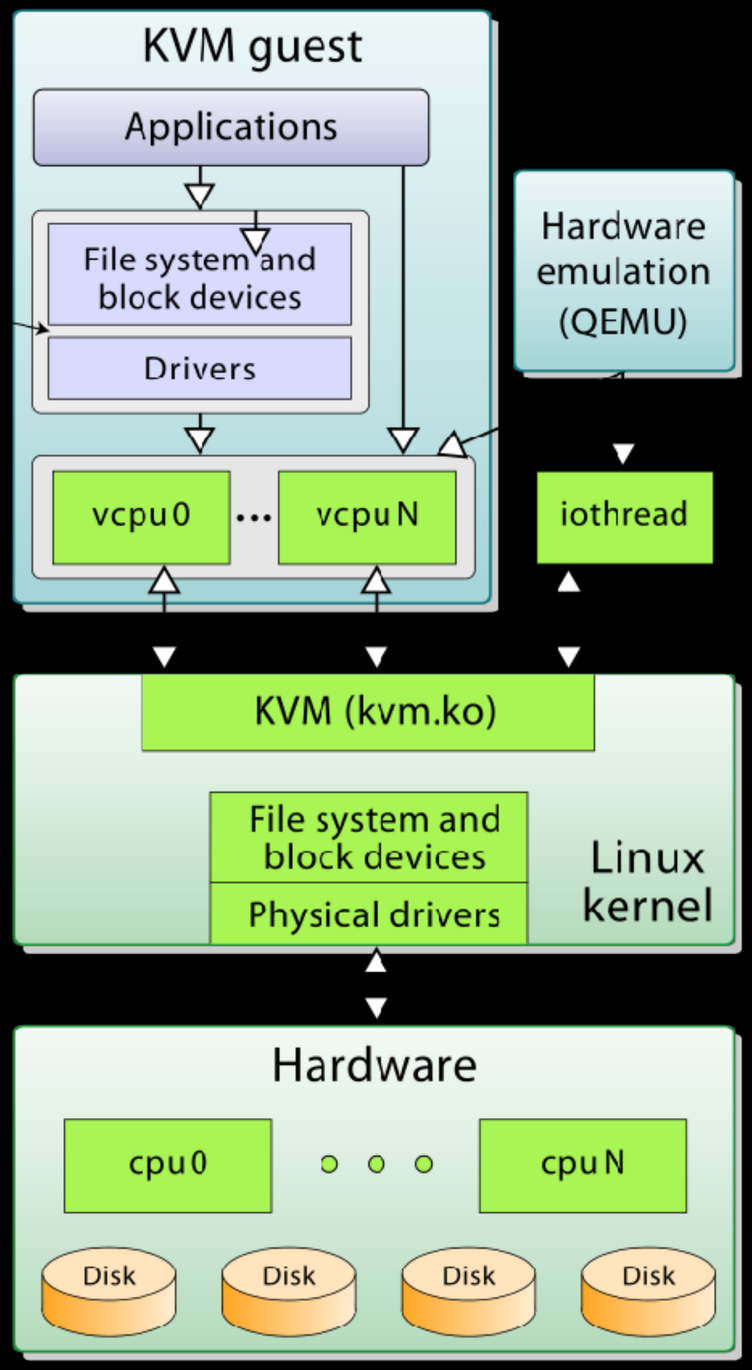
3.2 Kernel-based Virtual Machine

- In Kernel integrierte Virtualisierungs-Infrastruktur
- Bestandteile:
  - QEMU (= Gast)
  - Kernel-Modul (= Hypervisor)
  - Userspace-Interface

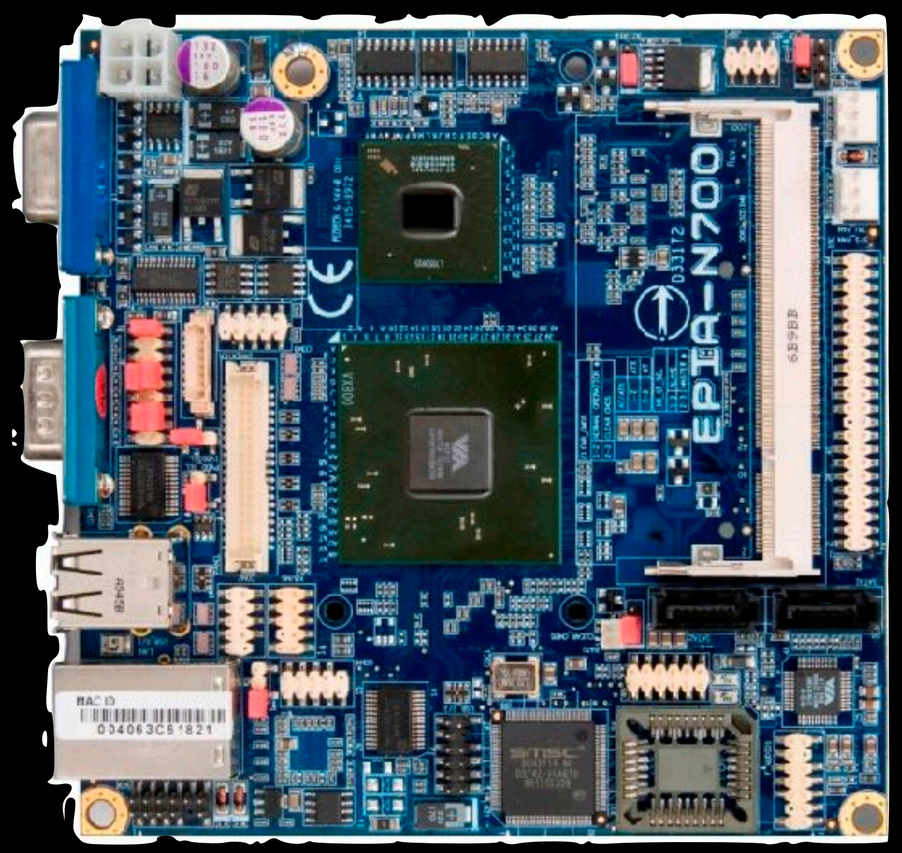
3.2 Kernel-based Virtual Machine

- Kernel zu Hypervisor
- gibt QEMU Hardware-Virtualisierung
- Seit Linux-Kernel 2.6.20 Mainstream
- weitestgehend unter LGPL
- Interface über libvirt
  - Zentrales Virtualisierungsframework

3.2 Kernel-based Virtual Machine



KVM



CC O.T.S.U.

CC VIA Gallery from Hsintien, Taiwan

CC Agiorgio

4. Vorstellung:
ZFS

Das eierlegende Wollmilch-Dateisystem

4. ZFS

- Zettabyte File System
- maximal Größe: 2^128 bytes (256 zebibytes)
- recht jung
  - Erste Version: Juni 2006
- Ursprünglich von Sun unter Open Source
  - Seit 2010 unter Oracle Closed Source
  - führte (indirekt) zu OpenZFS

4. ZFS

- Features:
  - Snapshots / CoW
  - integrierter Software-RAID
  - integriertes LVM
  - Checksumming
  - Komprimierung

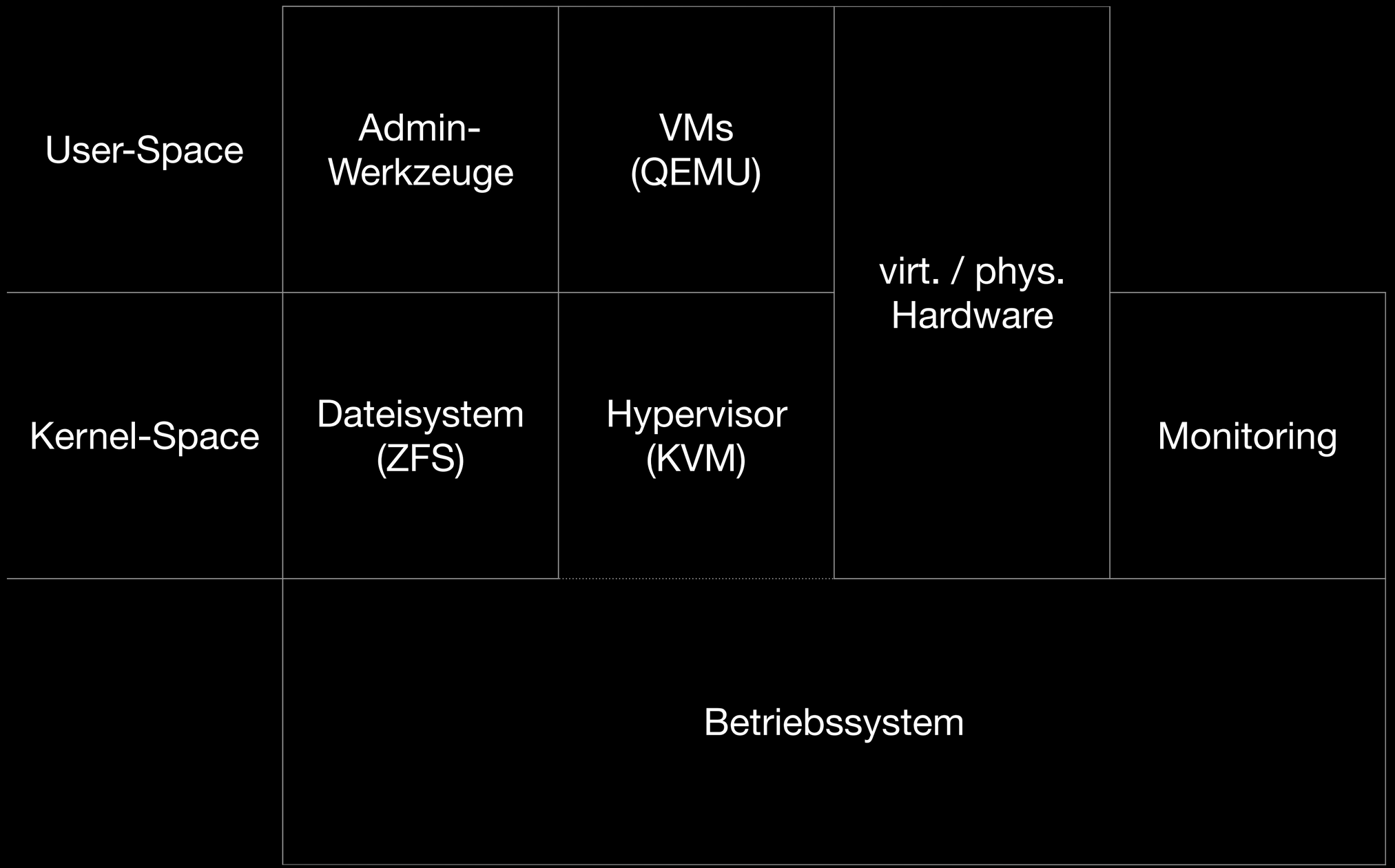
4. ZFS

- Verschlüsselung
- Deduplizierung
- Multi-Level-Caching
- Features (de)aktivierbar / Verhalten konfigurierbar
- dynamischer Umbau live möglich
- umfangreiche Admin-Tools integriert

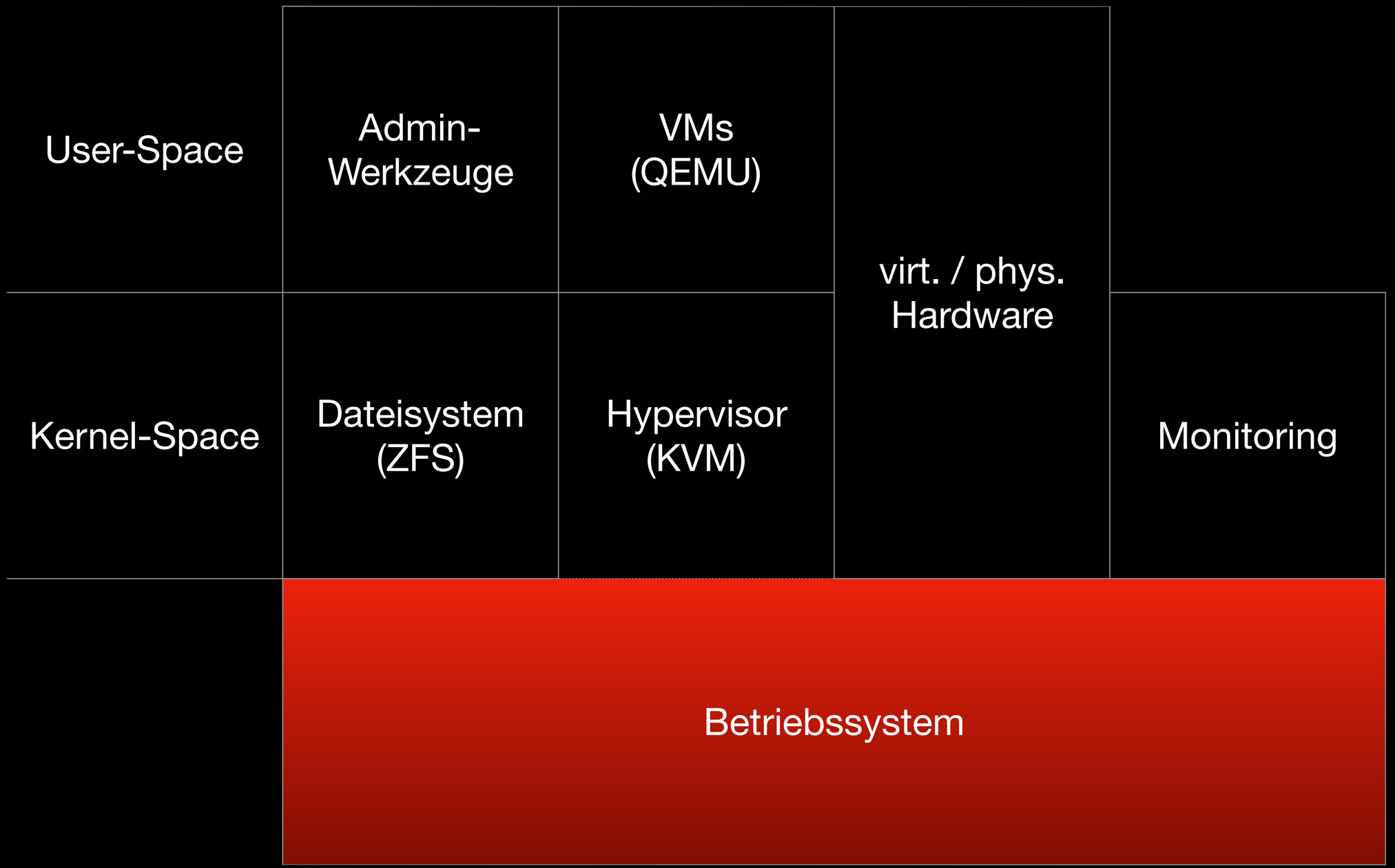
ZFS-Live-Demo

5.1 Aufbau des SW-Stacks

5.1.1 SW-Stack


User-Space
Admin-Werkzeuge
VMs (QEMU)
virt. / phys. Hardware
Kernel-Space
Dateisystem (ZFS)
Hypervisor (KVM)
Monitoring
Betriebssystem

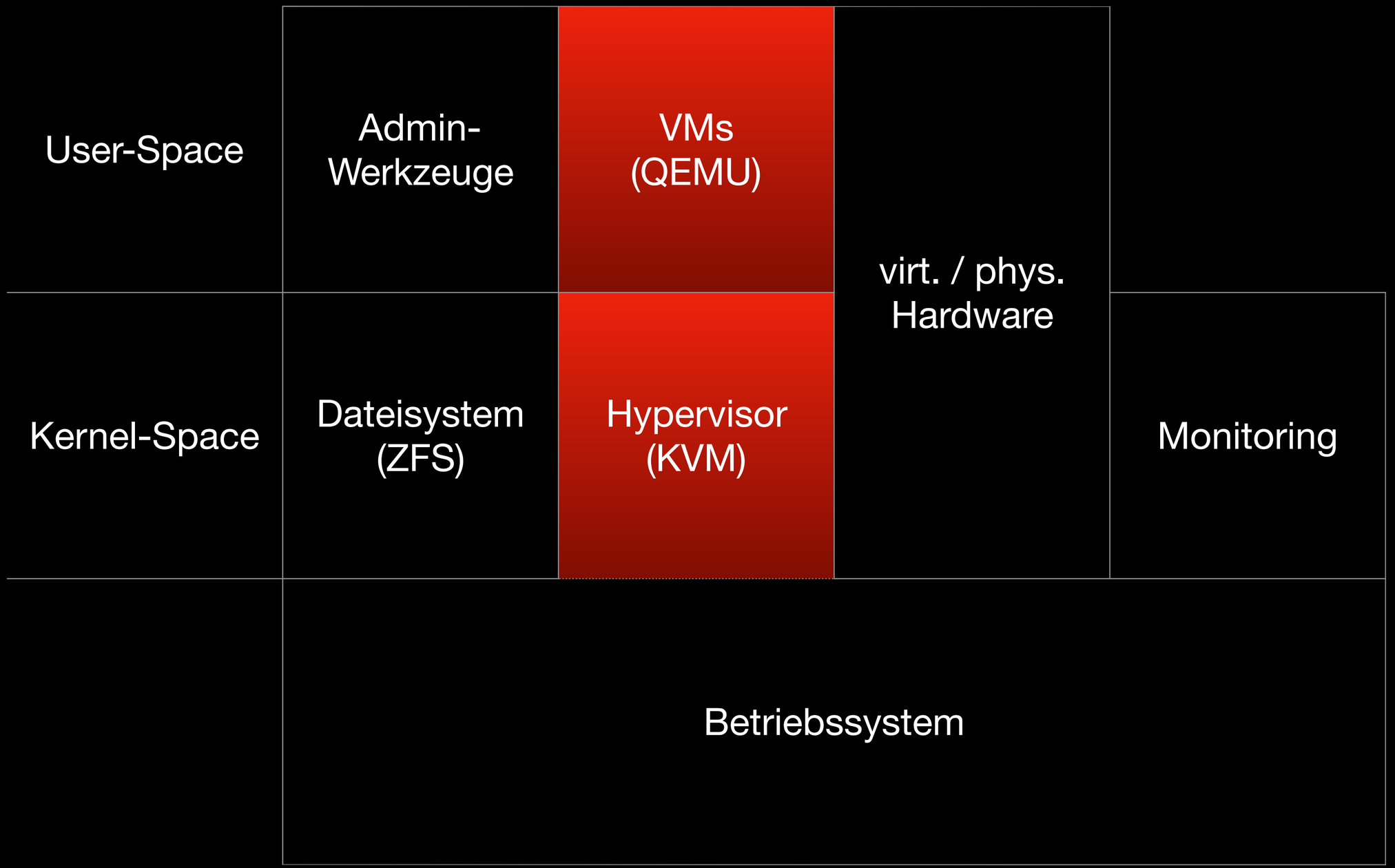
Schritt 1: System vorbereiten



Schritt 1: System vorbereiten

- Installieren & aktualisieren
- Verfügbarkeit von HW-Virtualisierung prüfen
  - Intel: VT-x, AMD: AMD-V
  - -> lscpu | grep “Virtualization”
- Automatische (Sicherheits-)Updates
- Härtung nach Anspruch

Schritt 2: Hypervisor installieren



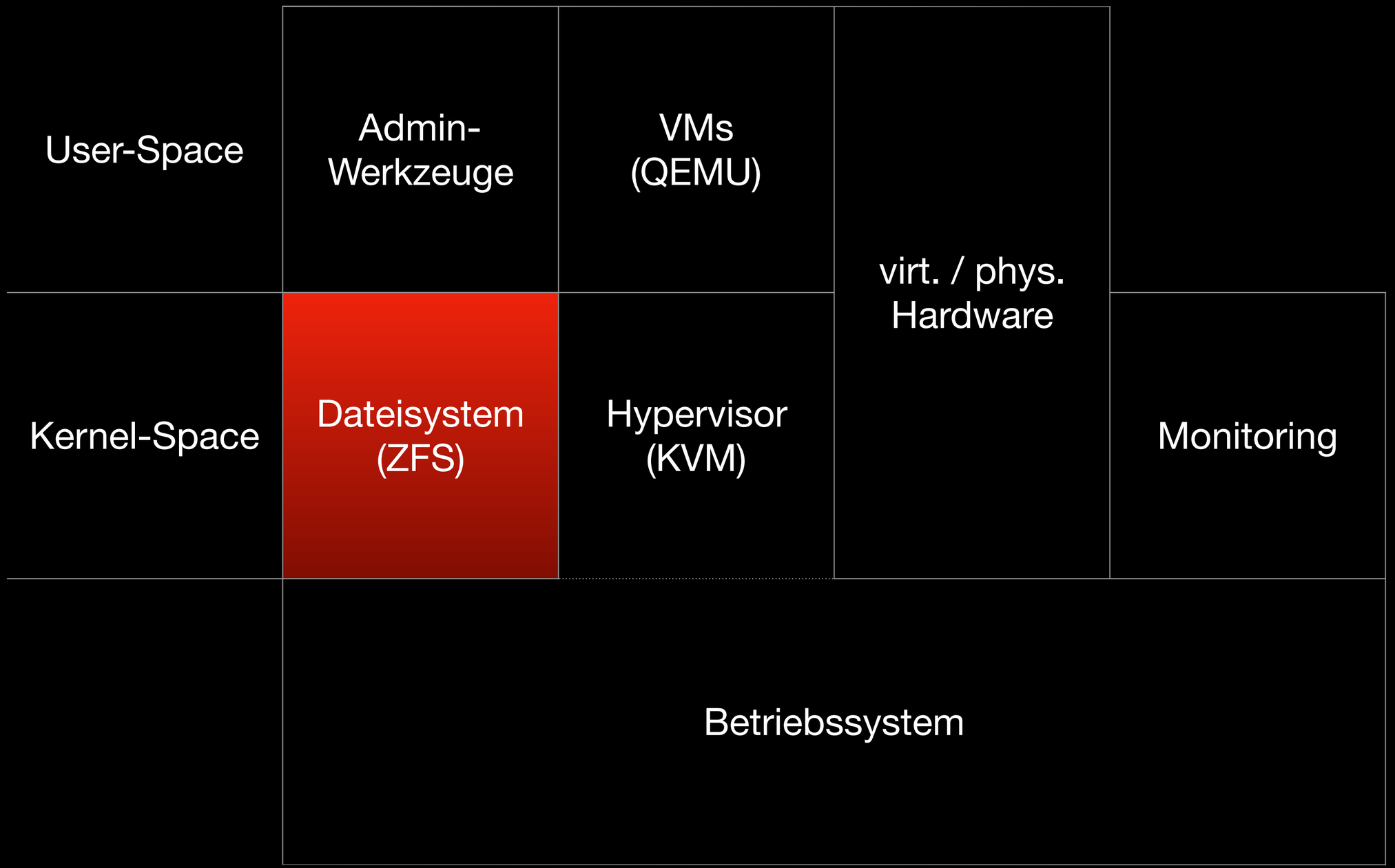
Schritt 2: Hypervisor installieren

- Kern-Paket: qemu-kvm
- nun verfügbar: /dev/kvm
- Kernelmodul: kvm
  - -> less /proc/modules | grep kvm

Schritt 2: Hypervisor installieren

- optionale Pakete:
  - qemu-system-*
  - qemu
  - je nach Repo in Extras / EPEL

Schritt 3: Speicher vorbereiten



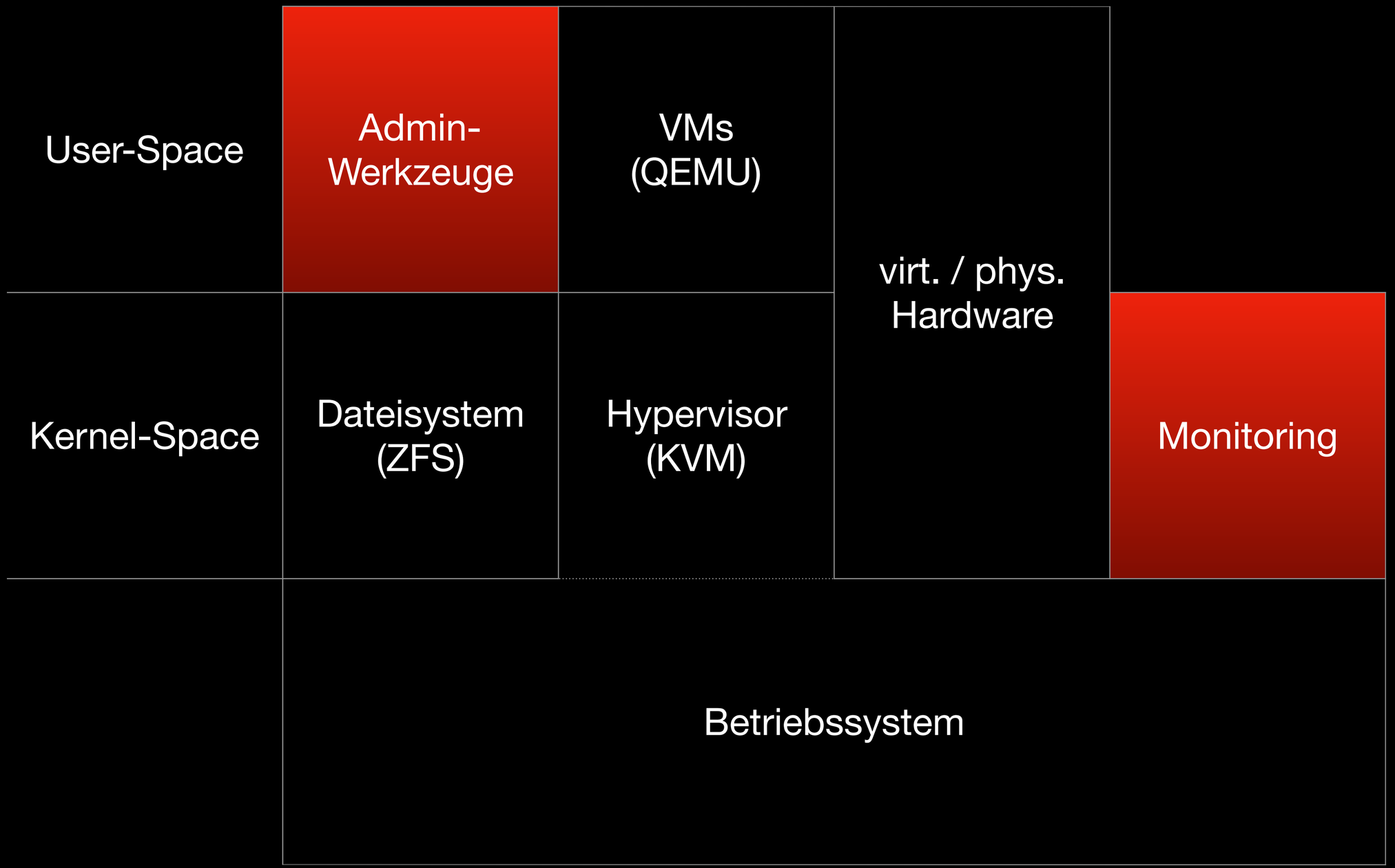
Schritt 3: Speicher vorbereiten

- Ubuntu seit 16.04 “universe”
- Sonst: OpenZFS-Repo hinzufügen
- DKMS vs. kmod
- ZFS installieren
- Kernelmodul laden: zfs

Schritt 3: Speicher vorbereiten

- Pool anlegen
  - Struktur festlegen
  - Features entscheiden
  - Mount bei Boot sicherstellen
- Dateisystem(e) konfigurieren
- Rohspeicher für VMs partitionieren / reservieren

Schritt 4: Admin-Tools vorberieten



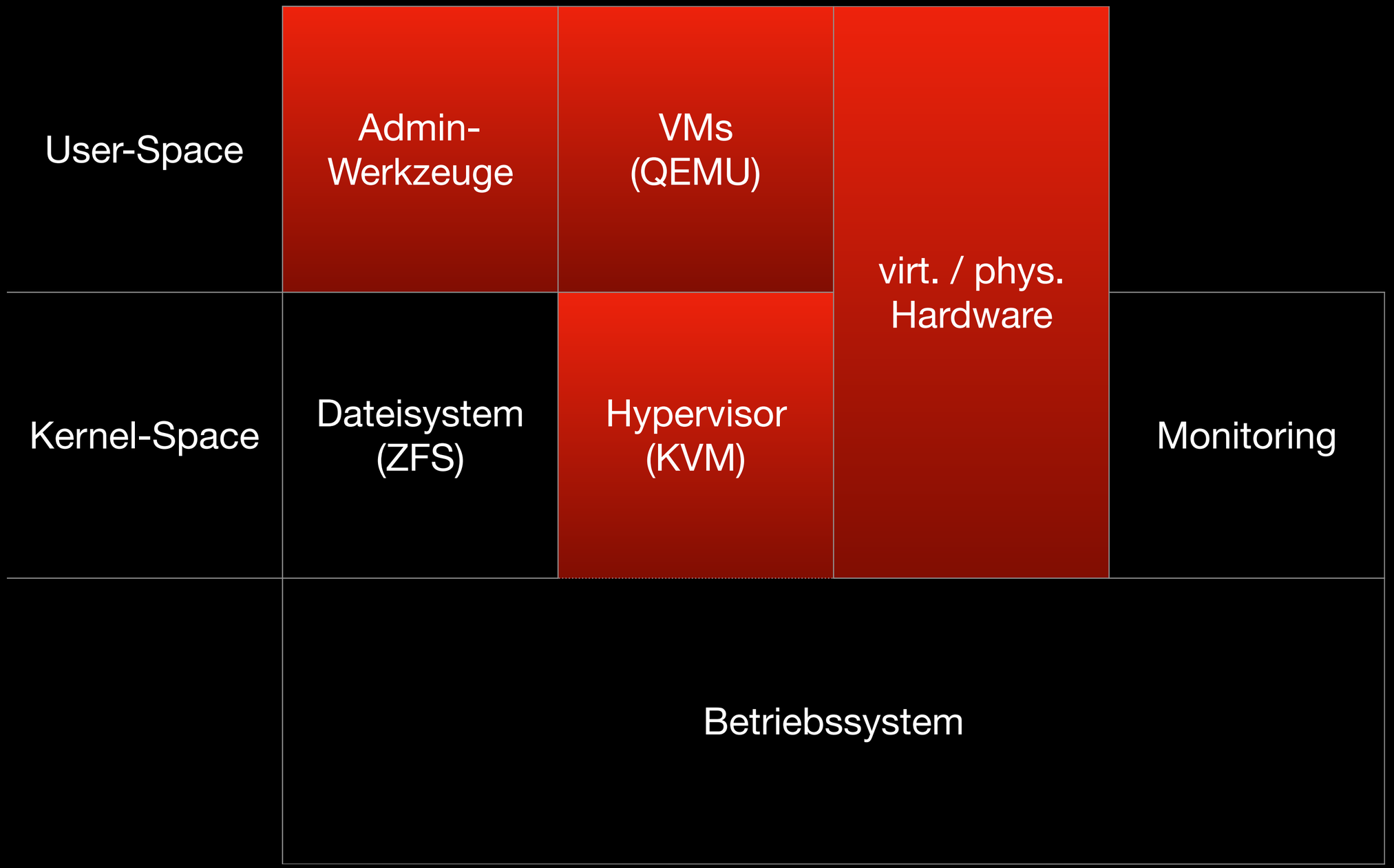
Schritt 4: Admin-Tools vorberieten

- libvirt
- virsh
- virt-manager
  - lokal oder remote
  - Nutzer muss Mitglied in “libvirt” sein
- alt. oVirt, GNOME Boxes, OpenStack, …

Schritt 4: Admin-Tools vorberieten

- {*x*top | *x* ∈ {h, io, if, …} }
- cockpit
- qemu-kvm-tools
- lm_sensors
- smartmontools
- Weitere Analyse nach Vorliebe

Schritt 5: VMs erstellen



Schritt 5: VMs erstellen

- Storage-Pools erstellen
  - KVM-Pools != ZFS-Pools
- event. Netwerk-Interfaces vorbereiten
  - nur bei Bridge
- Become a Keyboard-Warrior!
  - Or simply use the CLI / GUI

5.2 Betrieb des SW-Stacks

5.2 Betrieb des SW-Stacks

- Boot und Shutdown des Hosts
- Snapshoting
- Updates
- Hardware-Änderungen
  - Host
  - VM

5.2 Betrieb des SW-Stacks

- Trouble-Shooting
  - HW-Fehler
  - SW-Fehler
- Disaster-Mitigation
  - Snapshots != Backup
  - Host komplett generisch halten

6. Entscheidungen für den eigenen Server

6. Entscheidungen für den eigenen Server

- “from-scratch” vs. Frankenstein
- Anschaffungskosten vs. Betriebskosten
- Bleeding Edge vs. Stabilität
- passend vs. auf Zuwachs
- Verfügbarkeits-Anspruch
- Lautstärke
- Spezial-Hardware nötig?

7. Feldstudie

Anekdoten von Lachen und Weinen

7.1 “woodbox”

- “organisch gewachsen”
- mini-ITX
- AMD A4-5000 Quad-Core-APU
- 16 GB DDR3 (non-ECC)
- 5x HDD = 9,5TB (9TB für NAS)
- 60GB SSD OS, 256GB SSD VMs
- 64GB USB3 Stick
- 3x Gbit NIC

7.1 “woodbox”

- ~22W Idle (300W 80+ Bronze)
- CentOS 7.4
- 6VMs
  - YT-Autoloader
  - NAS
  - VPN-Server
  - Spiele-Server
  - CalDAV-Server
  - DNS-Resolver (+ Pi-Hole)

7.2 Weinen

- Wenn der RAM 1 sagt, aber 0 meint
- ZFS-Pool goes boom
- ZFS-Root-Pool
- Mainboard mag kein UEFI mehr

7.3 Lachen

- eigene Infrastruktur
- Spielplatz mit Sicherheitsnetz
- VM “zerbricht”
- “Wir brauchen einen ARK-Server”

Zusammenfassung

90min in 90sec

Zusammenfassung

- Geringe Anforderungen
- Großes Potential
- Flexibel
- Komplett (quell)offen
- Bringt das RZ nach Hause
- Das war nur die Spitze des Eisbergs

Bildquellen:

- https://de.wikipedia.org/wiki/Datei:Kernel-based_Virtual_Machine.svg
- https://de.wikipedia.org/wiki/Datei:VIA_EPIA_N700_Nano-ITX_Board-Top.jpg
- https://en.wikipedia.org/wiki/File:System_z_Frames.JPG
- https://en.wikipedia.org/wiki/File:Kvmbanner-logo2_1.png