



Technische Fakultät
Universität Bielefeld

RBG-Seminar

Virtualisierung - Was kommt nach Xen?

Dr. Carsten Gnörlich

Rechnerbetriebsgruppe

Übersicht

- eingesetzte Virtualisierungstechniken in der RBG
- Hintergrundinfos zu KVM
- Leistungsvergleich
- KVM-Vorführung
- Diskussion



Eingesetzte Virtualisierungstechniken

Eingesetzte Virtualisierungstechniken

Server (X4100 und größer)

- Xen 3.1 / 3.2
- KVM
- Linux Vserver (chroot mit Kernelunterstützung)

Desktop (PC, Laptop)

- VirtualBox

(noch) dominant: Xen

- zeitweise bis zu 10 Xen-Maschinen
- einige wichtige Dienste
 - Mailserver (X4200)
 - Netboot-NFS-Server (X4100)
 - Linux-Abbild an bis zu 200 Clients verteilt
 - Linux Compute-Server (X4100)
 - Linux SunRay-Server (FSC 2x Quadcore-Xeon)
- uptimes von mehreren 100 Tagen

Insgesamt

- Erfahrungen mit Xen sind sehr gut
- ➔ wir können auf Virtualisierung nicht verzichten
- ➔ aber Xen bietet nicht mehr den Supportlevel den wir benötigen



Warum wir von Xen wegmigrieren
werden

Xen ist keine freie Software mehr

- XenSource wurde Oktober 2007 von Citrix übernommen
 - ➔ Hypervisor bleibt frei (GPLv2)
 - ➔ *keine Kernelpatches* mehr, letzter offizieller Kernel ist 2.6.18
 - extreme Sicherheitslücken
 - keine Unterstützung aktueller Hardware mehr
- Angebot von Citrix: XenServer 4 / 5
 - 30 Tage-Testversion, XenServer Express frei wie in Freibier
 - käufliche Versionen: Listenpreis für 5 Jahre mehr als eine X4100
 - ➔ für uns sind das alles keine Optionen

für 500Mio Dollar

Inoffizielle Kernelpatches

- Xen ist nicht offizieller Bestandteil des Linux-Kernels
 - kernel.org-Versionen nur als DomU (Gast) nutzbar
 - im Prinzip ein Fork, nicht stabil
- inoffizielle Pflege in aktuellen Kernen durch Team von
 - Red Hat, SuSE, Canonical (Ubuntu) und anderen
 - funktionieren mal gut mal schlecht
 - (noch) in Debian lenny vorhanden (2.6.26), wird voraussichtlich gedroppt

RedHat und Sun

- RedHat wird / muß Xen bis 2014 unterstützen (7 Jahre nach RHEL 5!)
 - seit der Übernahme von Qumranet (kvm) im September 2008:
 - „vergeßt Xen, kvm ist die Zukunft“ (zwischen den Zeilen gelesen)
- Sun: xVM Server
 - Xen 3.1 Hypervisor
 - Dom0 übernimmt komplettes System + Festplatte(n)
 - nur Web-GUI, keine Kommandozeile und login in Dom0 (!)
 - ➔ das kann nicht ernstgemeint sein
 - ➔ Sun scheint sich auf xVM VirtualBox zu konzentrieren (gute Idee ;-)



Linux kvm

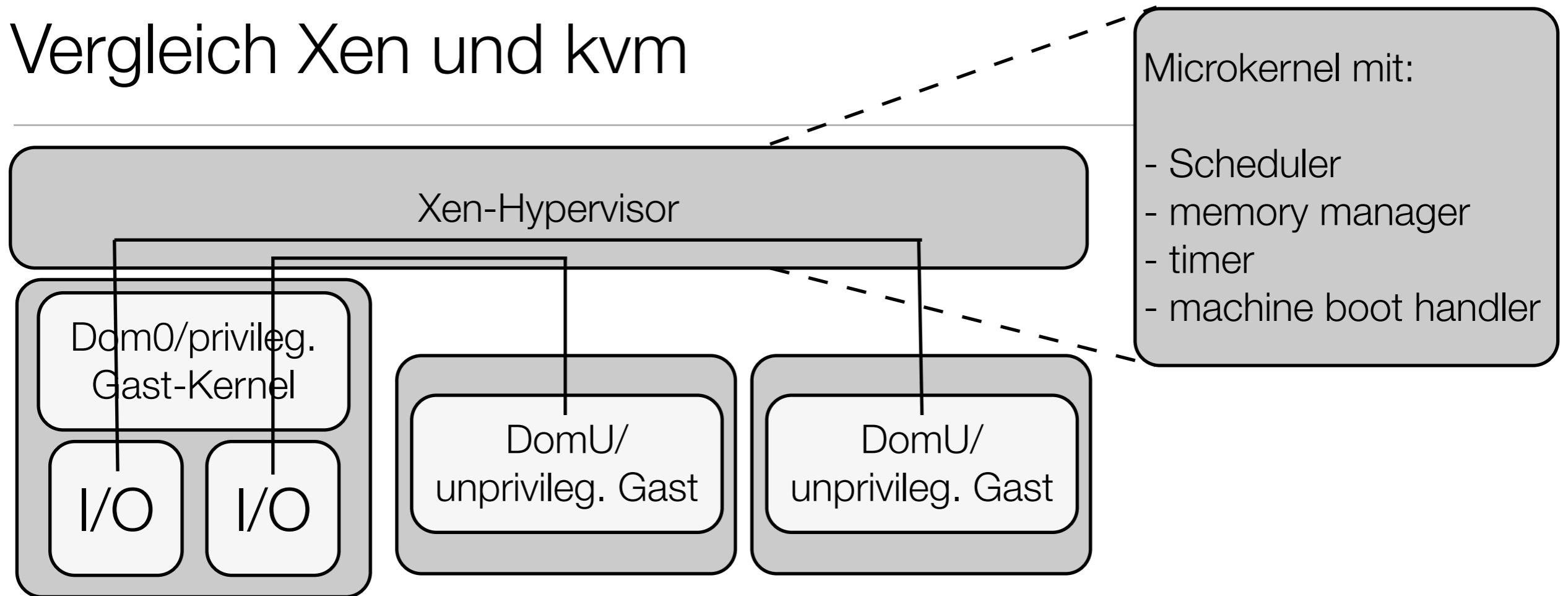
Historie von kvm

- Firma: Qumranet
- Co-Founder: Moshe Bar (openMosix, XenSource)
- Entwickler: Avi Kivity
- Beginn Mitte 2006
- im Kernel-Tree seit 2.6.20
- ab September 2008 bei Red Hat
 - unproblematisch, da Red Hat Open Source-Company ist (vgl. CentOS)
 - alle kvm-Komponenten sind LGPLv2 oder besser
 - wahrscheinlich einfacher weiterzuentwickeln als Xen

Komponenten von kvm

- CPU mit Hardware-Virtualisierung (Intel-VT oder AMD SVM)
- Kernel-Module: kvm.ko, kvm-intel.ko, kvm-amd.ko
 - ca. 10.000 Quellcode-Zeilen
- eine modifizierte Qemu

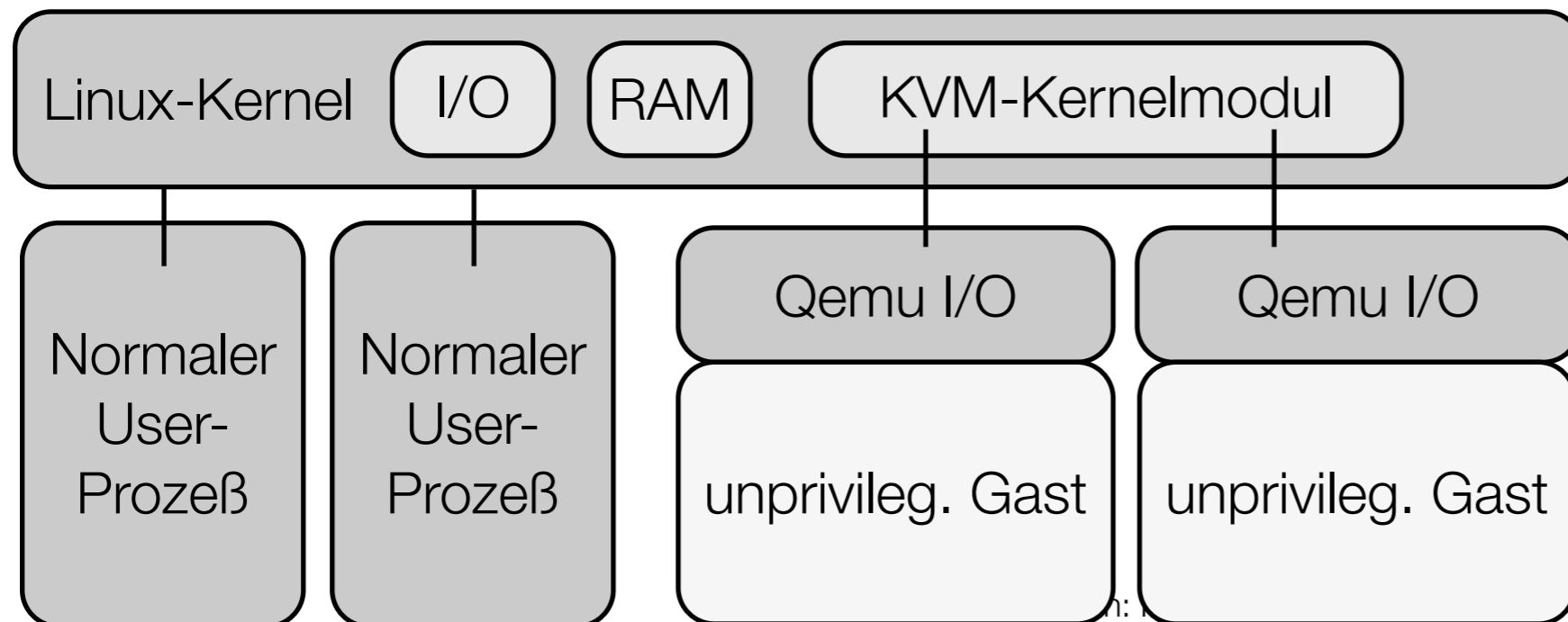
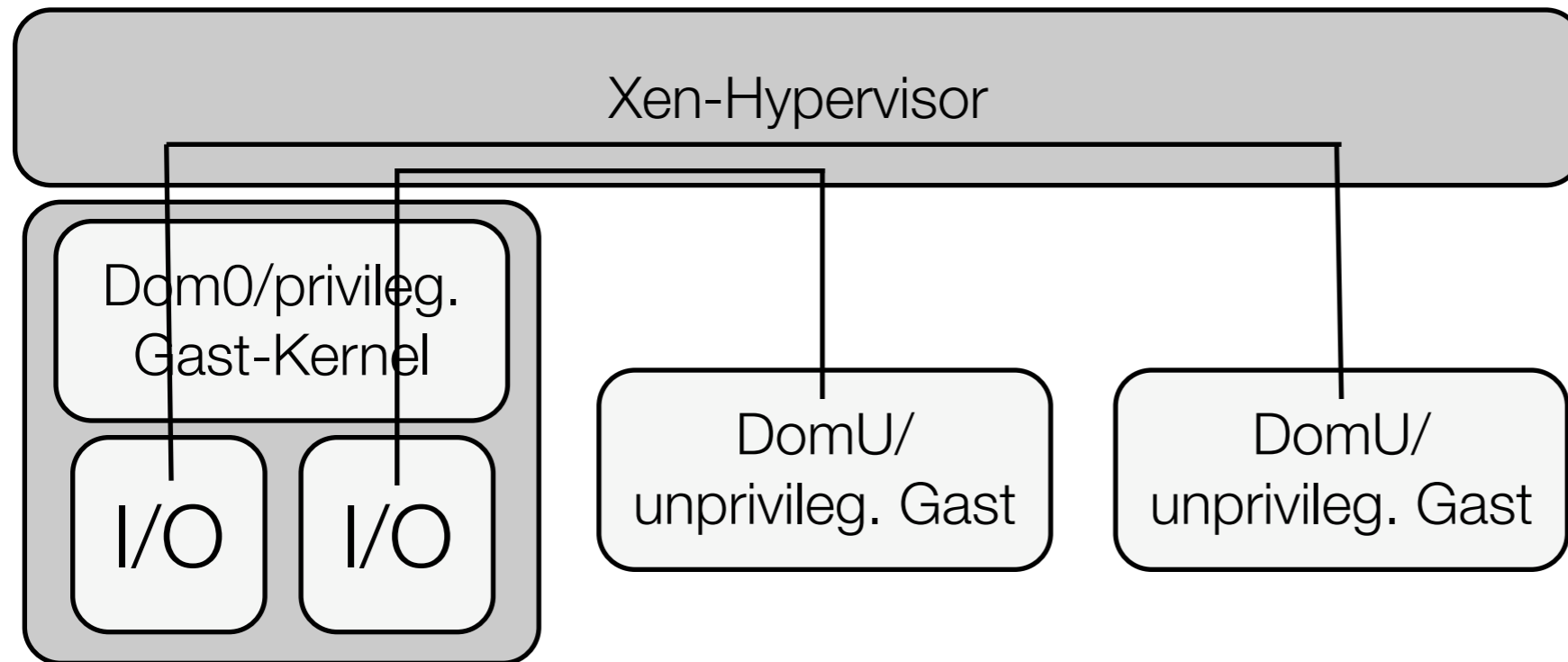
Vergleich Xen und kvm



Dom0 ist im Wesentlichen

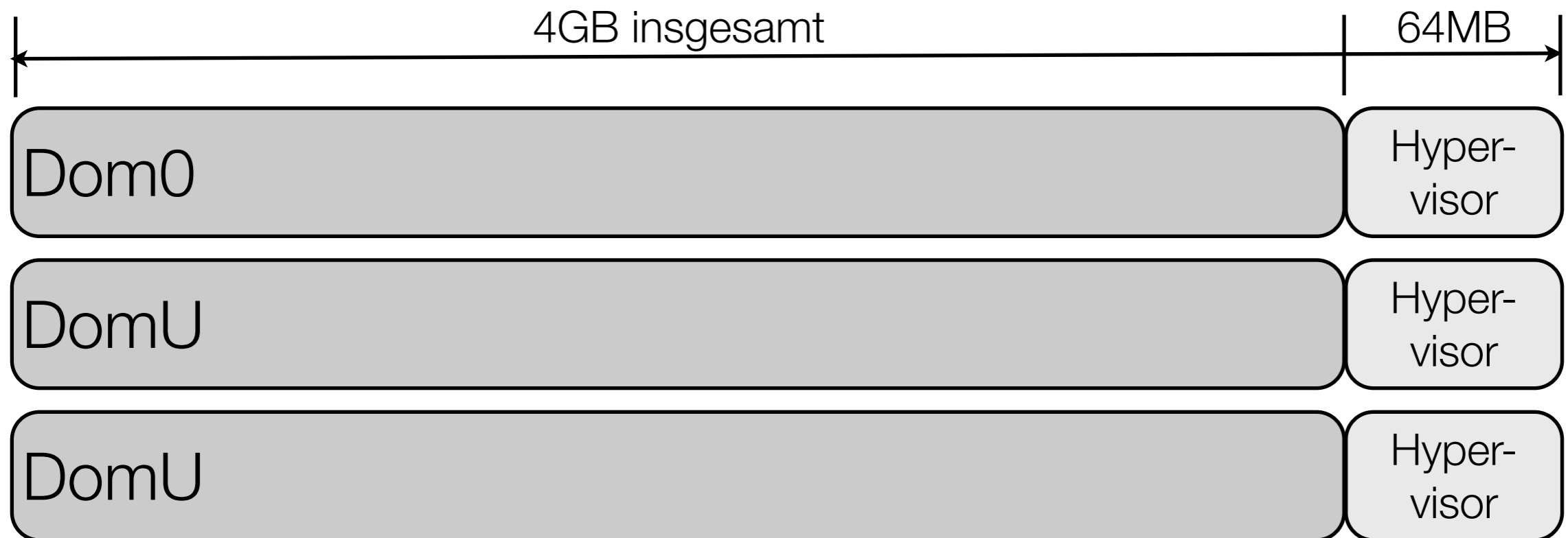
- I/O-Treiber
- Konsole

Vergleich Xen und kvm



Warum das komplexe Xen-Design?

- Beschränkungen der x86-32Bit-Architektur, insbesondere:
- 4GB Adreßraum + keine MMU-Virtualisierung



- spätere Hardware-Virtualisierung hingegen gibt Wirt und Gast getrennte Adreßräume

zu wenig für
komplettes
Linux-System!

Was macht Qemu?

- *Fabrice Bellard, ab 2005*

Qemu *emuliert* einen kompletten x86-PC, ca. Pentium-II-Ära:

- Mainboard
 - Speichercontroller, PCI-Bus, IDE-Kontroller
 - BIOS (aus Bochs)
 - VGA-Karte, Soundkarte, Netzwerkkarte
 - x86-CPU
- ➡ reine Emulation im user space
- ➡ könnte man nutzen um Windows unter Sparc auszuführen

Was macht kvm-qemu?

- Ersetzt simulierte CPU durch virtualisierte („echte“) CPU
- Paravirtualisierte Treiber für Festplatte und Netzwerk (Ingo Molnar)
- ➔ Performanzgewinn bei nativer x86-Plattform

- Weil das alles ist was kvm macht
- ➔ kompaktes Kernel-Modul

Unterschiede im Userland

Xen:

- komplexes python-basiertes Userland
- xm-Frontend
- xend, shared memory-Datenbank zur internen Kommunikation

kvm:

- ps, top, kill, brctl um virtuelle Maschinen zu manipulieren
- + ein paar Shellskripte zum Aufrufen von kvm-qemu
- vm ist halt einfach ein Linux-Prozeß im Wirtssystem



Benchmarks I kvm und VirtualBox

Testumgebung 1:

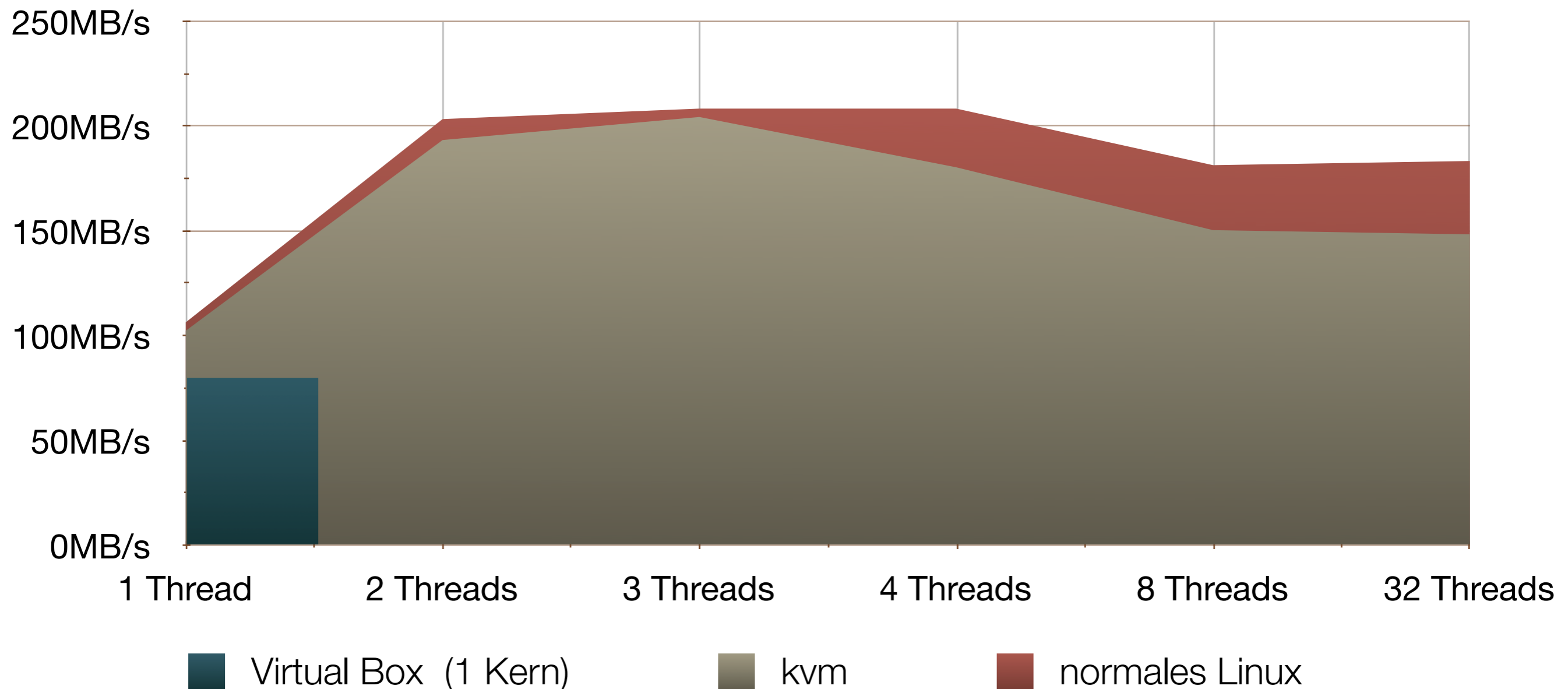
Hardware: Xeon 5405 (4x2Ghz), 8GB RAM,
ARC-1680 mit 6xWD1500HLFS 10K SATA

Benchmark: Reed-Solomon-Encoder

- berechnet fehlerkorrigierende Codes (vgl. CD, DVD, RAID)
- erzeugt extrem hohe Integer- und IO-Last
- je nach Einstellung CPU-/Memory-/IO-Beschränkt
- ich hatte ohnehin gerade daran zu benchmarken ;-)
 - ➡ wird mal Teil des RS03-Codecs in dvdaster 0.80

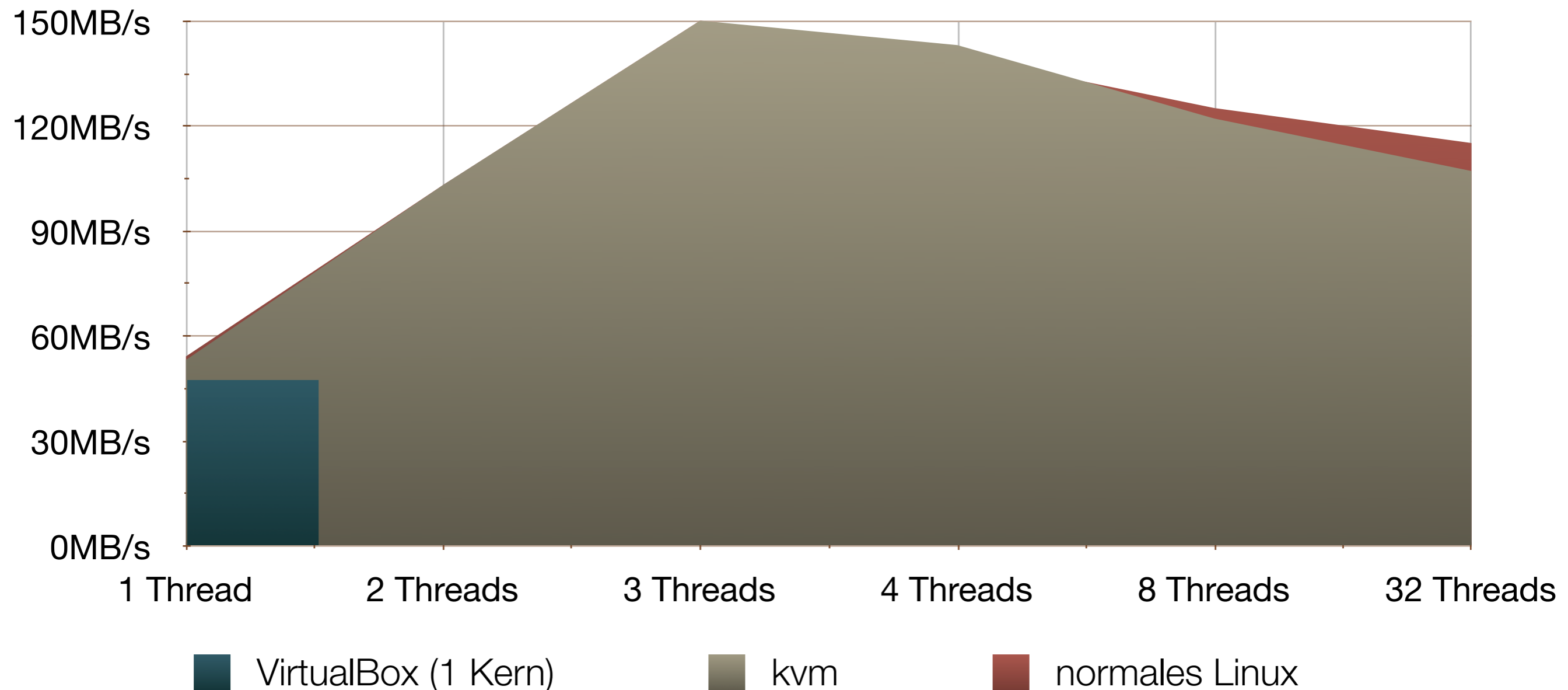
Memory-Bound, ohne I/O via /dev/shm

- testet nur CPU und Scheduling (nroots=8)



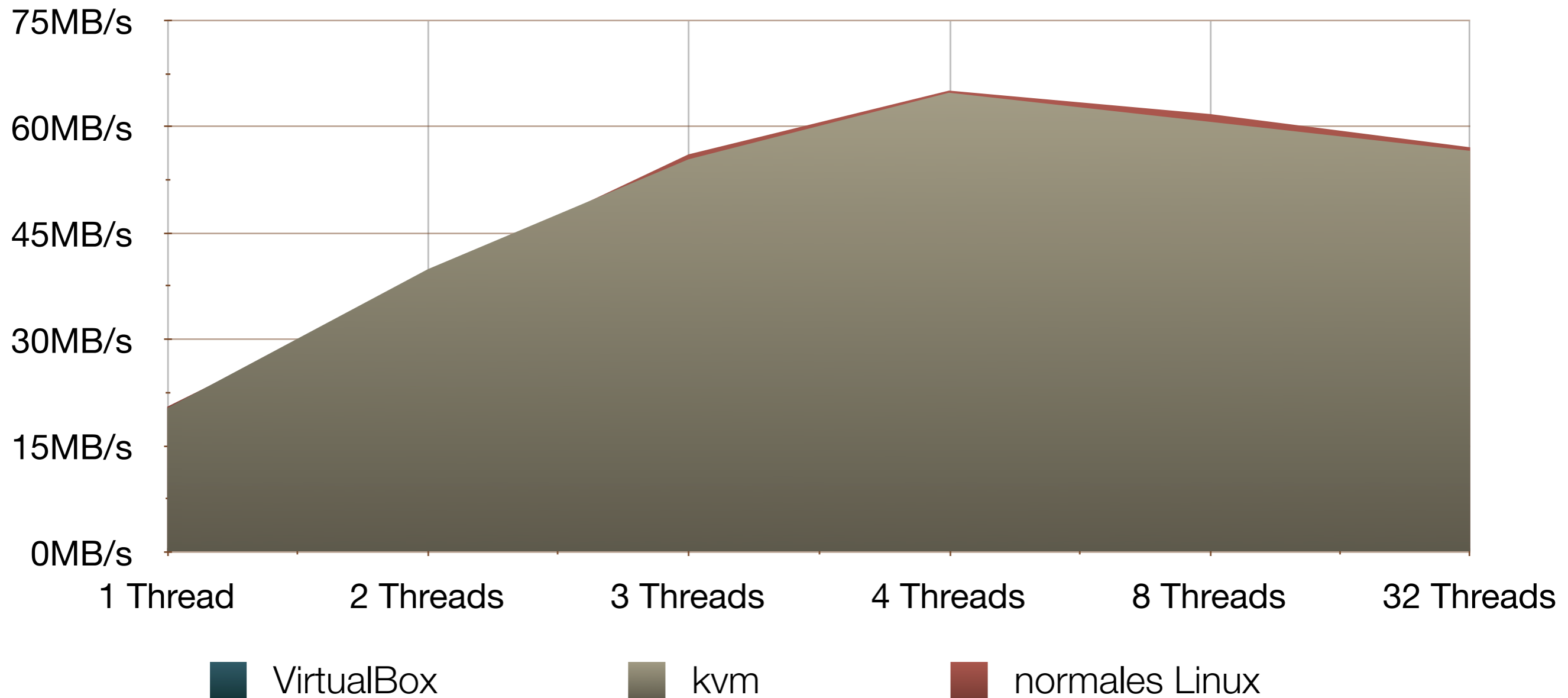
CPU-Bound, leichte Kost, ohne I/O via /dev/shm

- testet nur CPU und Scheduling (nroots=32)



CPU-Bound, schwere Kost, ohne I/O via /dev/shm

- testet nur CPU und Scheduling (nroots=128)

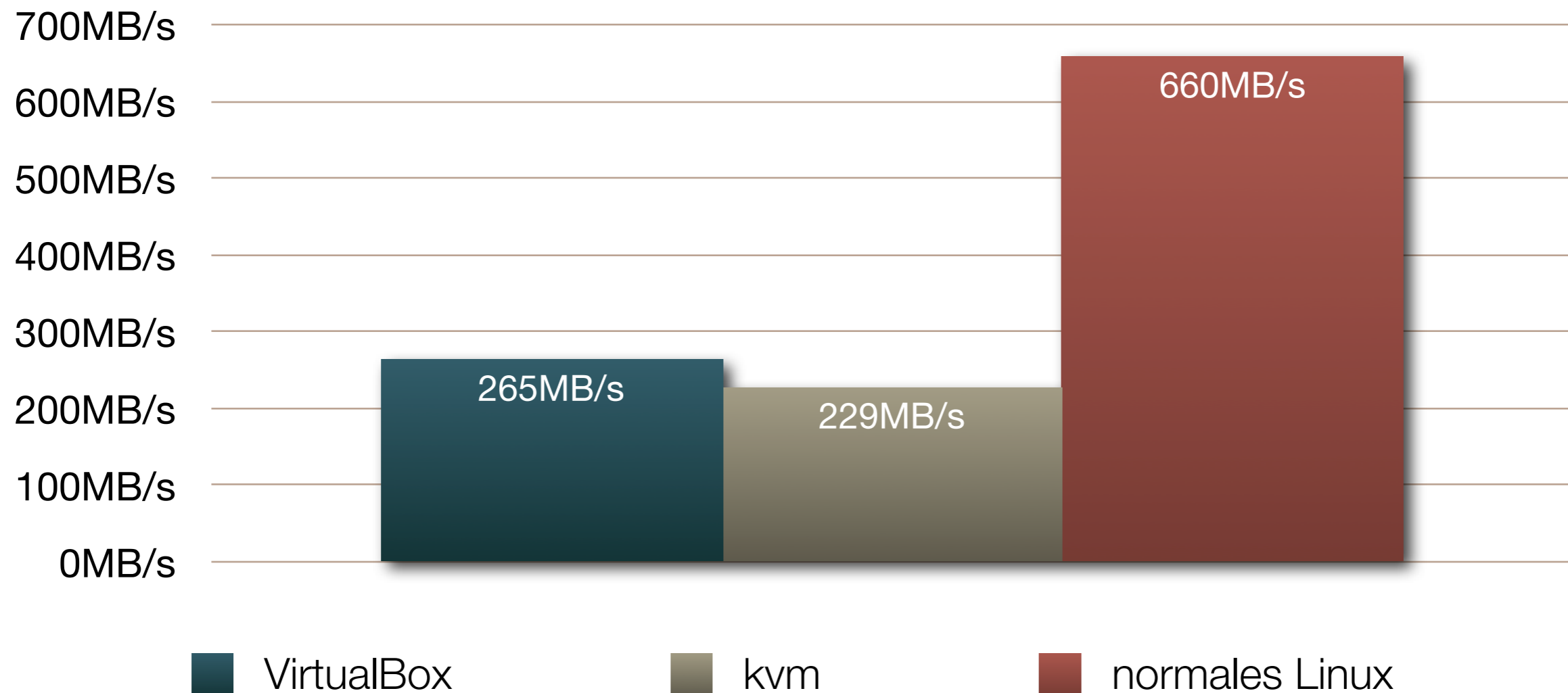


Schlußfolgerung aus diesen Messungen

- Virtualisierung kostet kaum CPU-Leistung
- allenfalls kleine Scheduling-Verluste
 - ➔ extremes Multithreading in der VM vermeiden

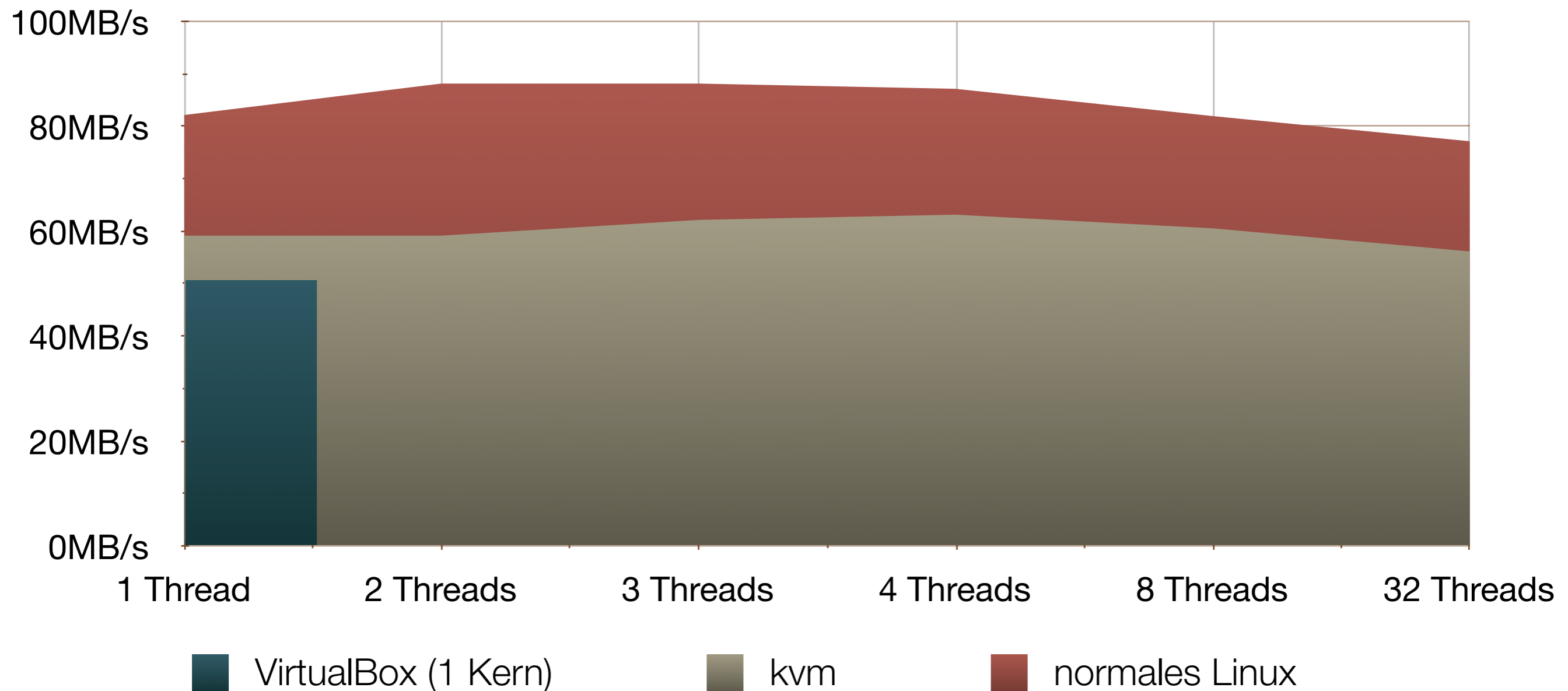
Sequentielle I/O-Leistung

Theoretische Bandbreite via `dd if=/dev/zero of=/dev/null bs=1M`



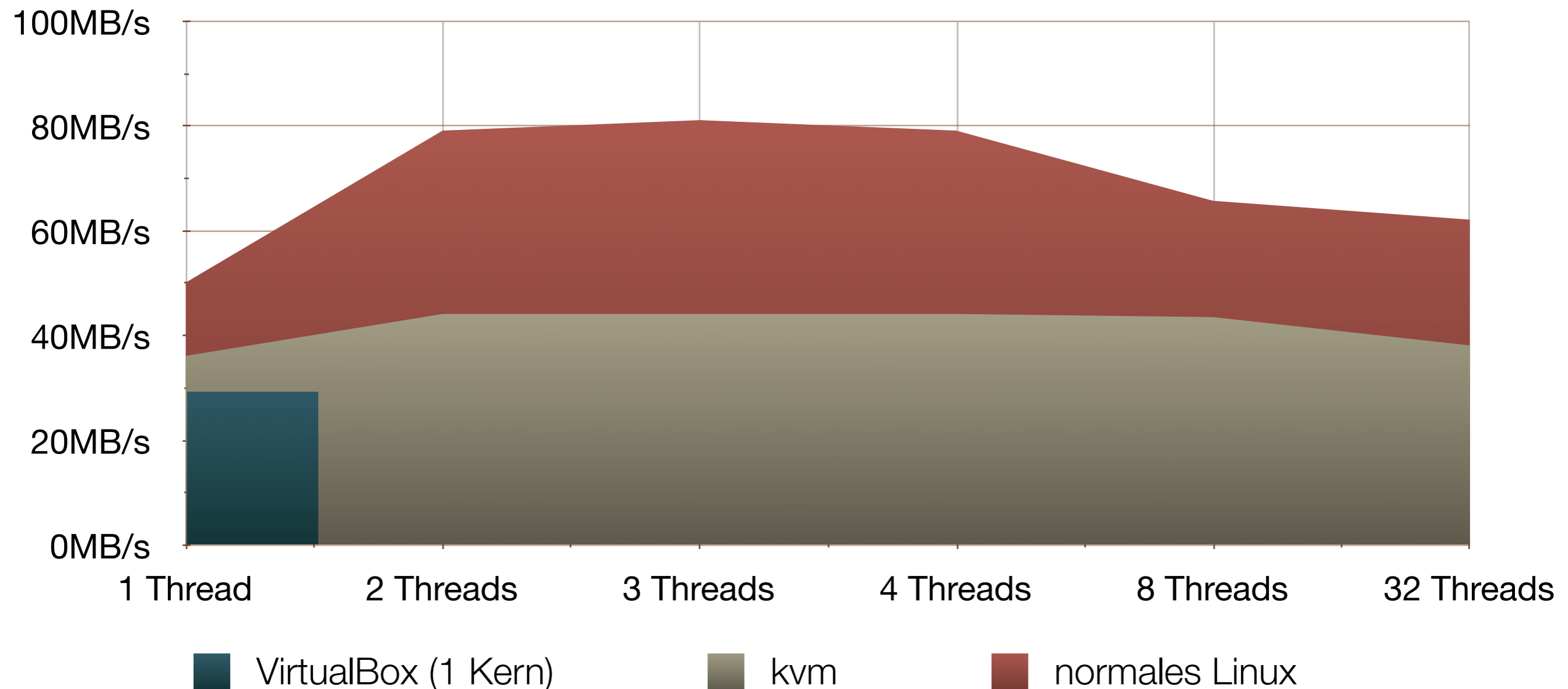
I/O-bound

- ermittelt maximalen random access-Durchsatz (nroots=8)



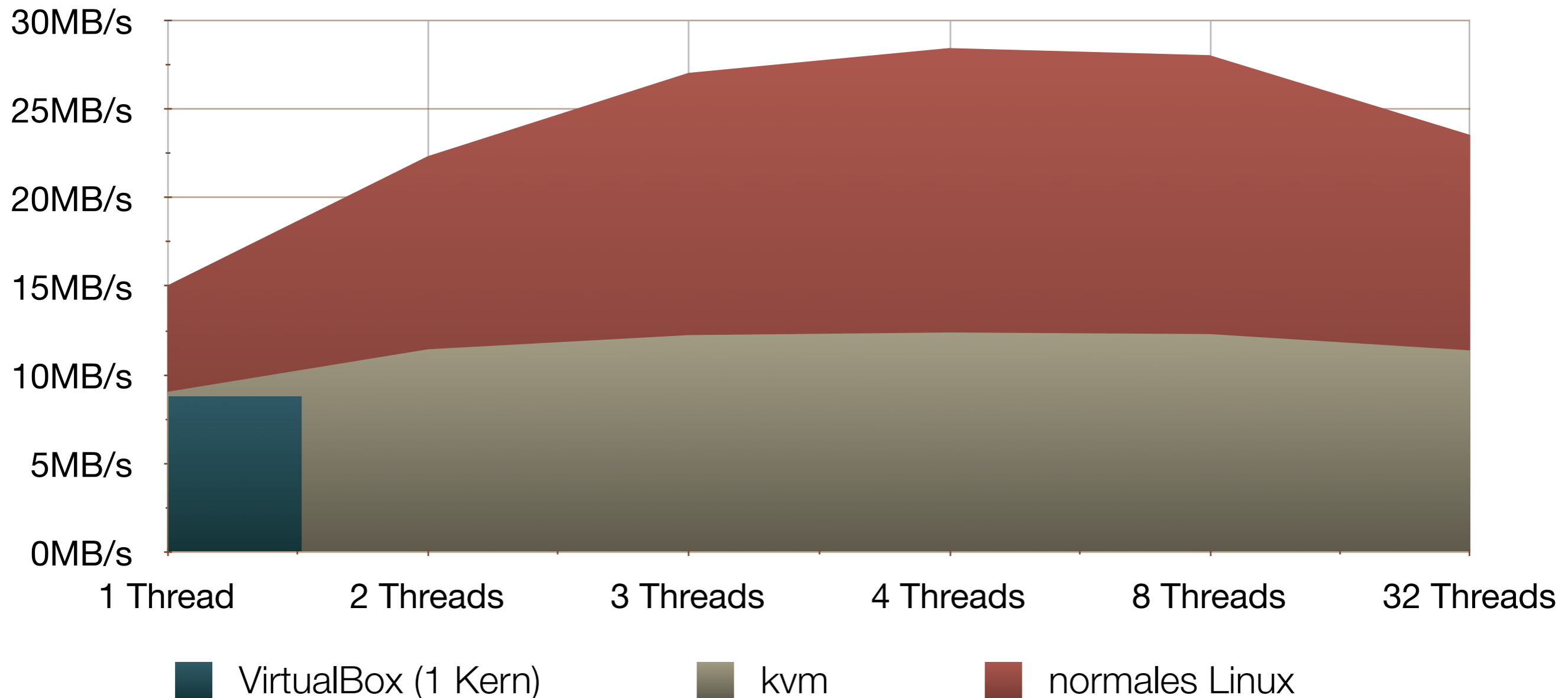
I/O-Bound mit leichter CPU-Last

- mittlere CPU-Last und viel random access (nroots=32)



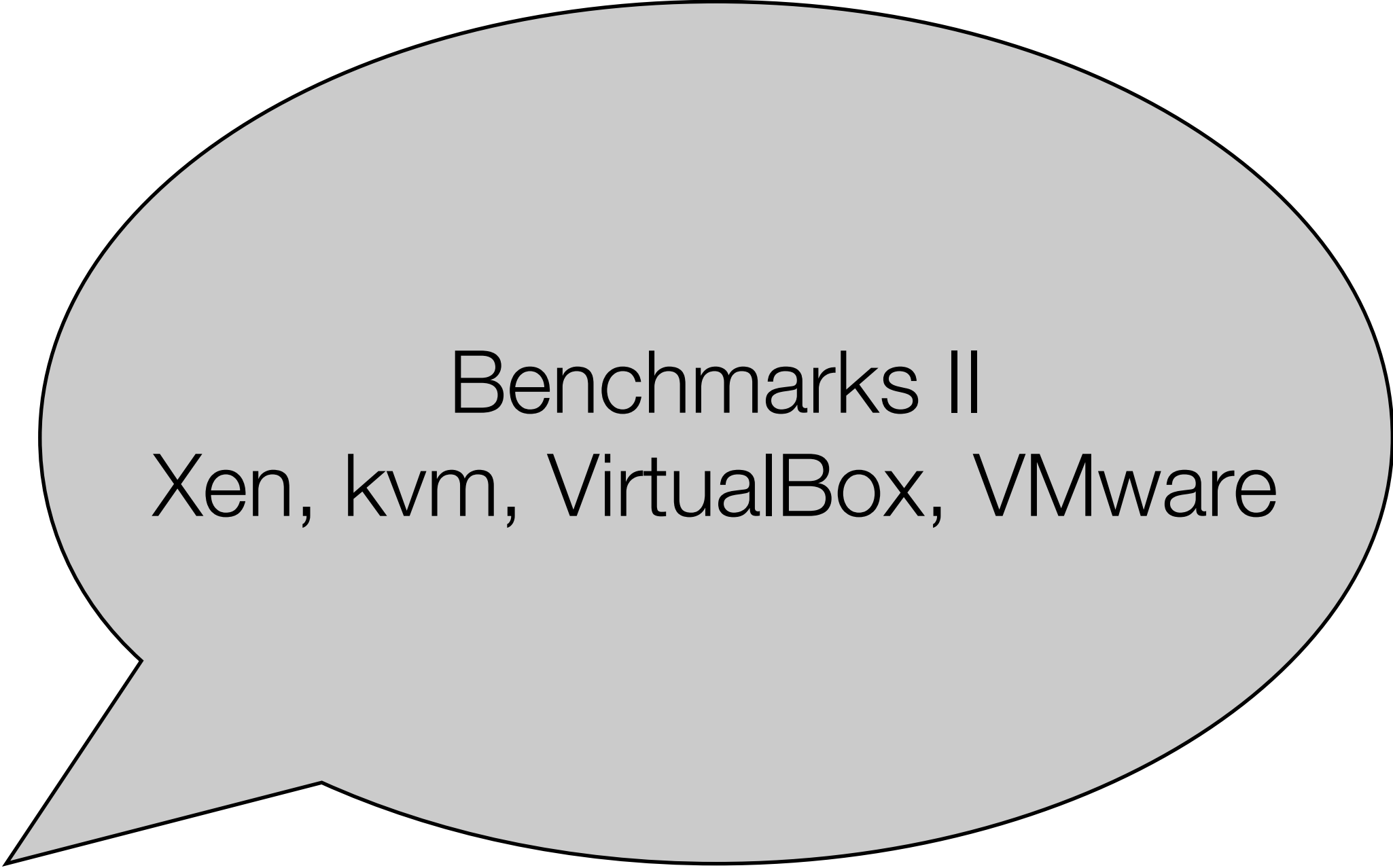
CPU-Bound

- hohe CPU-Last und etwas weniger random I/O (nroots=128)



Schlußfolgerung aus diesen Messungen

- I/O-Virtualisierung ist momentan noch ein Problem
 - analog für Netzwerk-I/O
 - ➔ für stark I/O-lastige Dienste ist Virtualisierung noch nichts
- Paravirtualisierte Treiber
 - waren bei dem Hardware-RAID kontraproduktiv
 - verbessern I/O bei Software-RAID (→ Testreihe 2)
 - bei unter 100MB/s RAW-Rate (einzelne Platte) kein Unterschied



Benchmarks II
Xen, kvm, VirtualBox, VMware

Testumgebung 2:

Hardware: Core 2 @2.4GHz, 2 GB RAM,

2x 80GB Barracuda.10, Software-Stripe

- studentisches Projekt
- Versionen der Virtualisierer und Benchmarks unterschiedlich zu Test 1
- ➡ Ergebnisse mit der anderen Maschine nicht vergleichbar
 - zu wenig RAM und I/O

Detaillierte Ergebnisse

- siehe Paper des studentischen Projektes



kvm quick and dirty
- Klon des Wirts booten

Vorbereitungen, Voraussetzungen

Wirtssystem:

- Debian Lenny, Kernel 2.6.26

Pakete:

```
host:/# apt-get install kvm virtio libsdl1.2debian-all
```

Root-Dateisystem des Wirts klonen und labeln:

```
host:/# lvcreate -s -L1G --name kvm_root /dev/internal/lenny_root
```

```
host:/# e2label /dev/internal/kvm_root kvm_root
```

Gast vorbereiten

```
host:/# mount /dev/internal/kvm_root /mnt
```

```
host:/# vi /mnt/etc/fstab
```

```
# <file system> <mount point> <type> <options> <dump> <pass>
proc /proc proc defaults 0 0
LABEL=kvm_root / ext3 errors=remount-ro 0 1
```

```
host:/# echo "arkani" >/mnt/etc/hostname
```

```
host:/# vi /mnt/etc/X11/xorg.conf
```

- nvidia-Treiber durch vesa ersetzen
- Auflösung runterstellen, z.B 800x600

```
host:/# umount /mnt
```

Skript zum Booten des Gastes erstellen

```
host:/# vi /root/kvm-test.bash
```

```
#!/bin/bash
```

```
KERNEL=2.6.26-1-amd64
```

```
NAME=arkani
```

```
kvm -smp 2 \
```

```
-drive if=ide,file=/dev/internal/kvm_root,boot=on \
```

```
-m 1024 \
```

```
-daemonize \
```

```
-name ${NAME} \
```

```
-kernel /boot/vmlinuz-${KERNEL} \
```

```
-initrd /boot/initrd.img-${KERNEL} \
```

```
-append "root=/dev/hda ro"
```

Gast booten

```
host:/root# ./kvm-test.bash
```

Aber Vorsicht:

- SDL-Fenster zu → Virtuelle Maschine aus!

Für den Produktiveinsatz:

- VNC
- serielle Konsole (→ nächstes Kapitel)



Server unter kvm einrichten

Dateisystem für Gast bootstrappen

```
host:/# apt-get install socat debootstrap
host:/# lvcreate -L4G --name kvm_guest stripe
host:/# mkfs.ext3 -L kvm_guest /dev/stripe/kvm_guest

host:/# mount LABEL=kvm_guest /mnt
host:/# touch /mnt/this-is-kvm-guest

host:/# debootstrap lenny /mnt file:/slow/deb-mount/d1
host:/# mkdir -p /mnt/export/iso
host:/# mount -o bind /export/iso /mnt/export/iso

host:/# chroot /mnt
```

Beispiel:
Bootstrappen aus
einem .iso
Einfacher geht es
aus dem Netz!

Gast in der chroot vorkonfigurieren

```
guest:~# passwd
```

```
guest:/# vi /etc/apt/sources.list
```

```
deb file:/export/iso/debian lenny main
```

```
guest:/# apt-get install udev locales initramfs-tools less ssh sudo
```

```
guest:/# dpkg-reconfigure locales
```

```
guest:/# echo "arkani" >hostname
```

```
guest:/# vi /etc/fstab
```

```
# <file system> <mount point> <type> <options> <dump> <pass>
```

```
proc /proc proc defaults 0 0
```

```
LABEL=kvm\_guest / ext3 errors=remount-ro 0 1
```

```
guest:/# vi hosts
```

```
127.0.0.1 localhost
```

```
127.0.1.1 arkani.techfak.uni-bielefeld.de arkani
```

```
guest:/# exit
```

Kernel für den Gast vorbereiten

```
host:/# cp -a /lib/modules/2.6.26-1-amd64 /mnt/lib/modules/.  
host:/# umount /mnt/export/iso/  
host:/# umount /mnt/                # Nicht vergessen!
```

Hilfreich: Kernel für die Gäste in ein Unterverzeichnis kopieren

→ ein apt-get upgrade im Wirt könnte den kernel updaten!

```
host:/boot# mkdir -p kvm/arkani  
host:/boot/kvm/arkani# cp /boot/*2.6.26-1* .
```

Modifiziertes Skript zum Booten des Gastes

```
host:/# vi /root/kvm-server.bash
KERNEL=2.6.26-1-amd64
NAME=arkani
```

```
kvm -smp 2 \
-drive if=ide,file=/dev/internal/kvm_guest,boot=on \
-m 1024 \
-daemonize \
-name ${NAME} \
-kernel /boot/kvm/${NAME}/vmlinuz-${KERNEL} \
-initrd /boot/kvm/${NAME}/initrd.img-${KERNEL} \
-append "root=/dev/hda ro"
```

Jetzt einmal testweise booten:

```
host:/# ./kvm-server.bash
```

Serielle Konsole einrichten

```
host:/# mount LABEL=kvm_guest /mnt
```

```
host:/# vi /mnt/etc/inittab
```

- folgendes hinzufügen:

```
T0:12345:respawn:/sbin/getty -L ttyS0 115200 vt100
```

```
host:/# vi /mnt/etc/sysctl.conf
```

- 'kernel-printk ...' auskommentieren

```
host:/# umount /mnt
```

```
host:/# mkdir /etc/kvm/consoles
```

Modifiziertes Skript für die serielle Konsole

```
host:/# vi /root/kvm-server.bash
```

```
KERNEL=2.6.26-1-amd64
```

```
NAME=arkani
```

```
kvm -smp 2 \
```

```
-drive if=ide,file=/dev/internal/kvm_guest,boot=on \
```

```
-m 1024 \
```

```
-nographic \
```

```
-daemonize \
```

```
-name ${NAME} \
```

```
-kernel /boot/kvm/${NAME}/vmlinuz-${KERNEL} \
```

```
-initrd /boot/kvm/${NAME}/initrd.img-${KERNEL} \
```

```
-append "root=/dev/hda ro console=ttyS0,115200" \
```

```
-serial mon:unix:/etc/kvm/soles/${NAME}.sock,server,nowait
```

Skript zum Verbinden mit der seriellen Konsole

```
#!/bin/bash

guestname=${1}

if [ -z "${guestname}" ]; then
    echo "Usage: `basename ${0}` GUESTNAME"
    echo ""

    exit 1
fi

echo ""
echo "-----"
echo " KVM guest console. Press <CTRL> + <Q> to quit."
echo "-----"
echo ""

socat -,echo=0,raw,intr=0x11,isig=1 UNIX-CONNECT:/etc/kvm/consoles/${guestname}.sock

echo ""
echo "KVM console closed."
echo ""
```

Booten und serielle Konsole verwenden

- erstmaliges Booten der VM:

```
./start-server2.bash && reset && ./kvmconsole.bash arkani
```

- Verlassen der Konsole mit CTRL-Q

- Erneutes Verbinden mit der Konsole:

```
./kvmconsole.bash arkani
```



Paravirtualisierte Treiber

Auf dem Client neue initrd bauen

```
guest:/# vi /etc/initramfs-tools/modules
```

```
virtio  
virtio_blk  
virtio_net  
virtio_rng  
virtio_ring  
virtio_pci  
virtio_balloon
```

hinzufügen

```
guest:/# update-initramfs -c -k 2.6.26-1-amd64
```

```
guest:/# halt
```

```
host:/# mount /dev/stripe/kvm_guest /mnt
```

```
host:/boot/kvm/arkani# cp /mnt/boot/initrd.img-2.6.26-1-amd64 .
```

```
host:/# umount /mnt
```

zum Wirt
hochkopieren

Modifiziertes Skript für paravirtualisierte Treiber

```
host:/# vi /root/kvm-server.bash
```

```
KERNEL=2.6.26-1-amd64
```

```
NAME=arkani
```

```
kvm -smp 2 \
```

```
-drive if=virtio,file=/dev/internal/kvm_guest,boot=on \
```

```
-m 1024 \
```

```
-nographic \
```

```
-daemonize \
```

```
-name ${NAME} \
```

```
-kernel /boot/kvm/${NAME}/vmlinuz-${KERNEL} \
```

```
-initrd /boot/kvm/${NAME}/initrd.img-${KERNEL} \
```

```
-append "root=/dev/vda ro console=ttyS0,115200" \
```

```
-serial mon:unix:/etc/kvm/soles/${NAME}.sock,server,nowait
```



Netzwerk einrichten

Szenario: VM auf primäres Interface des Hosts

```
host:/# vi /etc/network/interfaces
```

```
#iface eth0 inet dhcp
```

```
iface eth0 inet manual
```

```
auto br-kvm
```

```
iface br-kvm inet static
```

```
    address 192.168.1.1
```

```
    netmask 255.255.255.0
```

```
    bridge_ports eth0
```

```
    bridge_fd 9
```

```
    bridge_hello 2
```

```
    bridge_maxage 12
```

```
    bridge_stp off
```

```
    bridge_maxwait 5
```

problemlos durch dhcp ersetzbar

kvm braucht spezielles Skript

```
host:/# vi /etc/kvm/kvm-ifup-local
```

```
#!/bin/sh
```

```
switch=br-kvm
```

```
/sbin/ip link set $1 up
```

```
/usr/sbin/brctl addif ${switch} $1
```

```
exit 0
```

Modifiziertes Skript für Netzwerk

```
host:/# vi /root/kvm-server.bash
```

```
KERNEL=2.6.26-1-amd64
```

```
NAME=arkani
```

```
kvm -smp 2 \
```

```
-drive if=ide,file=/dev/internal/kvm_guest,boot=on \
```

```
-m 1024 \
```

```
-nographic \
```

```
-daemonize \
```

```
-name ${NAME} \
```

```
-kernel /boot/kvm/${NAME}/vmlinuz-${KERNEL} \
```

```
-initrd /boot/kvm/${NAME}/initrd.img-${KERNEL} \
```

```
-append "root=/dev/hda ro console=ttyS0,115200" \
```

```
-serial mon:unix:/etc/kvm/soles/${NAME}.sock,server,nowait \
```

```
-net nic,macaddr=00:1A:4A:00:8E:30,model=virtio \
```

```
-net tap,script=/etc/kvm/kvm-ifup-local
```

Gast booten, Netzwerk wie gewohnt hochfahren

```
guest:/# vi /etc/network/interfaces
```

```
# The loopback network interface
```

```
auto lo
```

```
iface lo inet loopback
```

```
auto eth0
```

```
iface eth0 inet static
```

```
    address 192.168.1.10
```

```
    netmask 255.255.255.0
```

```
    gateway 192.168.1.1
```

```
guest:/# ifup lo eth0
```



Aus ISO-Abbild installieren

Passendes Startskript

KERNEL=2.6.26-1-amd64

NAME=fukano

kvm -smp 2 \

-drive if=ide,file=/dev/internal/kvm_win,boot=on \

-drive if=ide,file=/export/slow/win.iso,media=cdrom,boot=off \

-boot d \

-m 2048 \

-daemonize \

-no-shutdown \

-name \${NAME} \

-net none

kein Netz konfiguriert

→ NAT auf das primäre Interface

→ reicht um nach Hause zu telefonieren

→ -net none um Netzwerk abzuschalten

Bootreihenfolge

nur über

-boot

verändern!

Zusammenfassung

- wir werden von Xen nach kvm migrieren
 - kvm performancemäßig unter den besten Lösungen
 - frei, keine eingeschränkte Funktionalität
 - beste Integration mit Linux
 - insbesondere im Kerneltree (keine Kernel mehr selbst patchen + bauen)
- warten auf den Durchbruch bei I/O-(Para)-Virtualisierung
- VirtualBox wegen fehlender SMP-Fähigkeit nur für Desktops interessant
- Linux Vserver für spezielle Lösungen, z.B. Webserver



Links

kvm und qemu

- <http://www.qumranet.com/>
- <http://bellard.org/qemu>

Die anderen Projekte

- [http://linux-vserver.org/Welcome to Linux-VServer.org](http://linux-vserver.org/Welcome_to_Linux-VServer.org)
- <http://www.virtualbox.org/>
- <http://www.sun.com/software/products/xvm/index.jsp>
- <http://www.xenserver5.com/>

Ende des Vortrags

Danke fürs Zuhören!