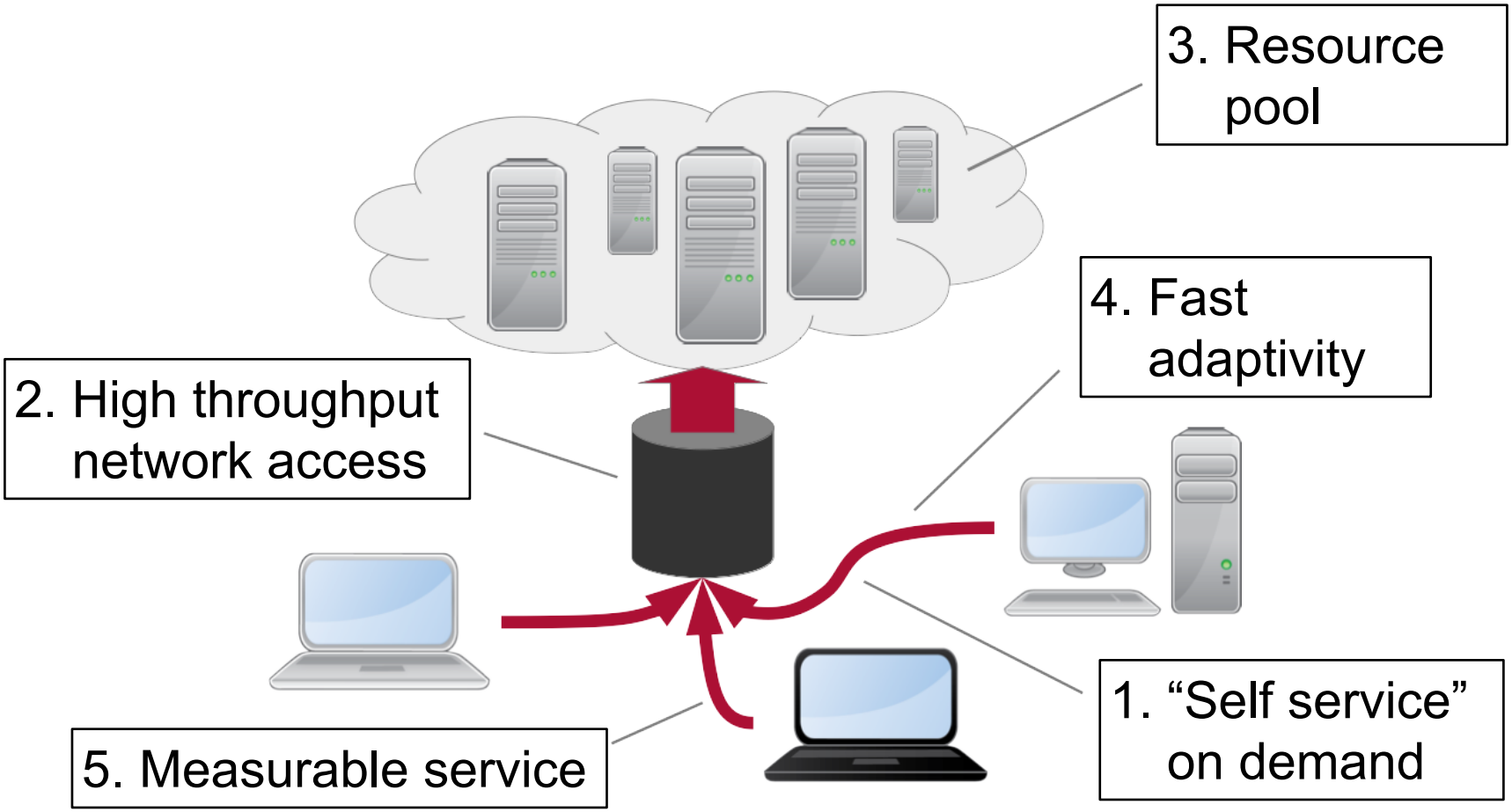
Operating Systems

Lecture 18: Cloud Operating Systems

Michael Engel

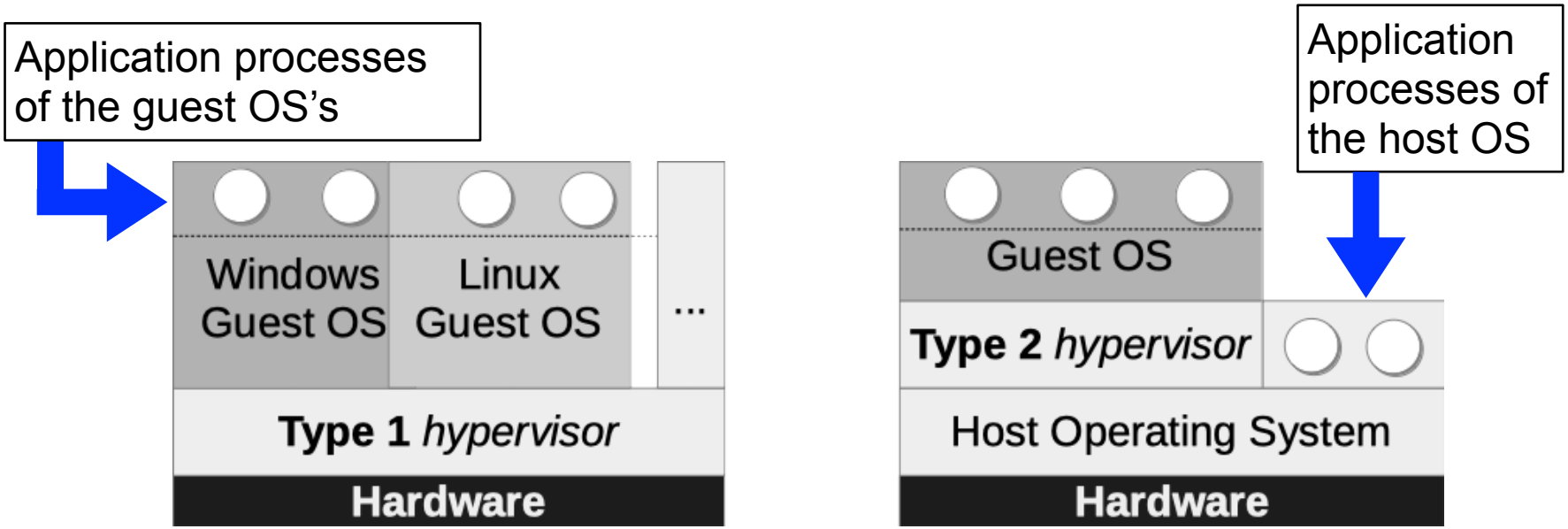
Cloud computing

- According to the US *National Institute of Standards and Technology*, a **Cloud** has five properties:



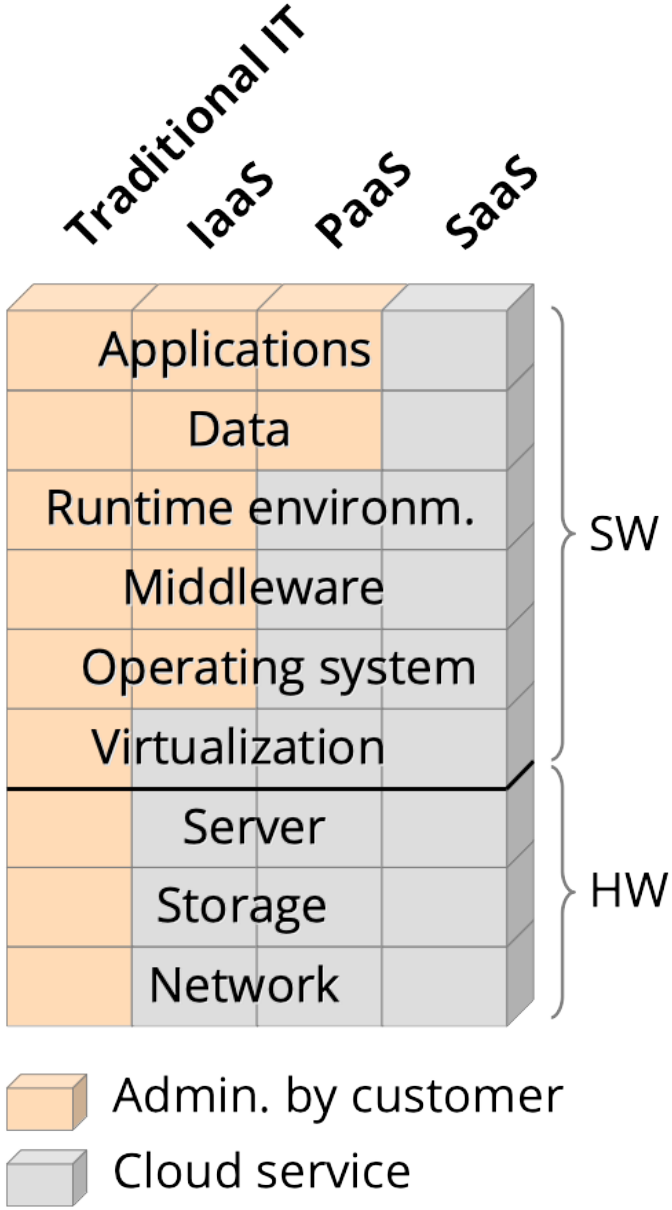
Hardware virtualization

- …enables the creation of multiple **virtual machines** on one physical computer. Each virtual machine can have its own OS.
- Important foundation technology for Cloud computing and server consolidation
- Technical basis: **hypervisor** / **virtual machine monitor**



Cloud service models

- **SaaS** – Software-as-a-Service
  - Cloud service provider offers a complete application
  - e.g. Office365, Gmail, Zoom
- **PaaS** – Platform-as-a-Service
  - Execution environment for applications including the OS and runtime environment (depending on the programming language)
  - e.g. Engine Yard, Google App Engine
- **IaaS** – Infrastructure-as-a-Service
  - (Virtual) hardware platform
  - e.g. Amazon EC2, Microsoft Azure



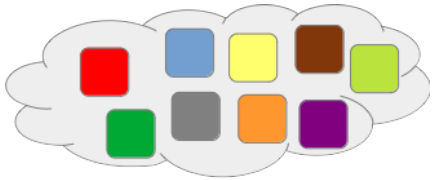
After an idea in Stallings' "*Operating Systems*"

Discussion: Cloud disadvantages

- Cloud-Computing has a number of advantages, but can also cause problems that must not be ignored…
  - **Data protection and privacy**
    - Where are the data of my users/customers located? Which data protection laws apply in the respective country? (➛ GDPR)
    - Is the cloud service provider trustworthy?
  - **Vendor lock-in**
    - Can I retrieve my data (for a reasonable amount of money) if I want to change the provider? If yes, in which format?
  - **Quality of service**
    - Which guarantees are offered by the provider?

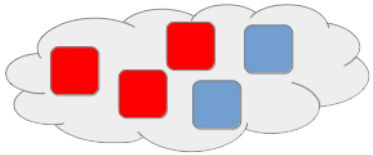
Provisioning models

- **Public Cloud**
  - Cloud Service Provider (CSP) has arbitrary customers

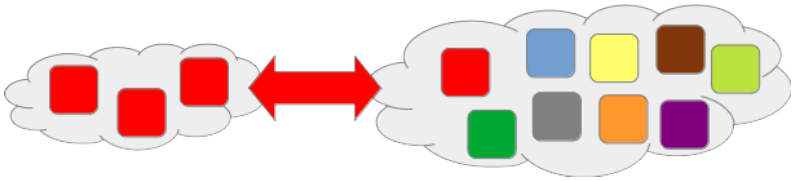


- **Private Cloud**
  - A cloud infrastructure for a (large) company, which can use the company's own or rented resources.
    ➛ more control

- **Community Cloud**
  - Multiple customers with the same requirements share a cloud infrastructure



- **Hybrid Cloud**
  - Mixed approach

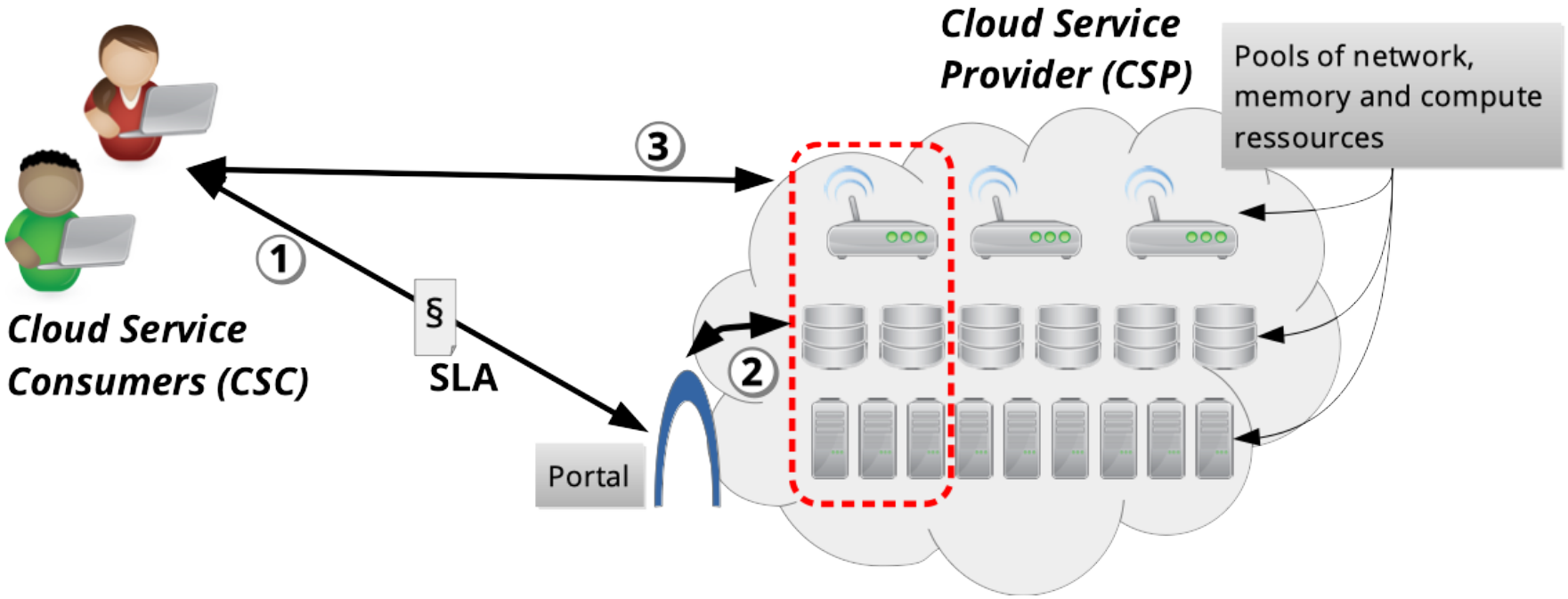


Comparison of provisioning models

- Stallings: "Operating Systems: Internals and Design Principles"

	Private cloud	Community cloud	Public cloud	Hybrid cloud
Scalability	restricted	restricted	very high	very high
Data protection Security	most secure option	very secure	moderately secure	very secure
Performance	very good	very good	low to medium	good
Reliability	very high	very high	medium	medium to high
Costs	high	medium	low	medium

Application example / Requirements



- Secure access via portal
- Function selection (e.g. VM templates)
- Choice of a service
- Observation and adaptation

1

- Provisioning of the required resources
- Confirmation of a **service level agreement** (SLA) and related costs

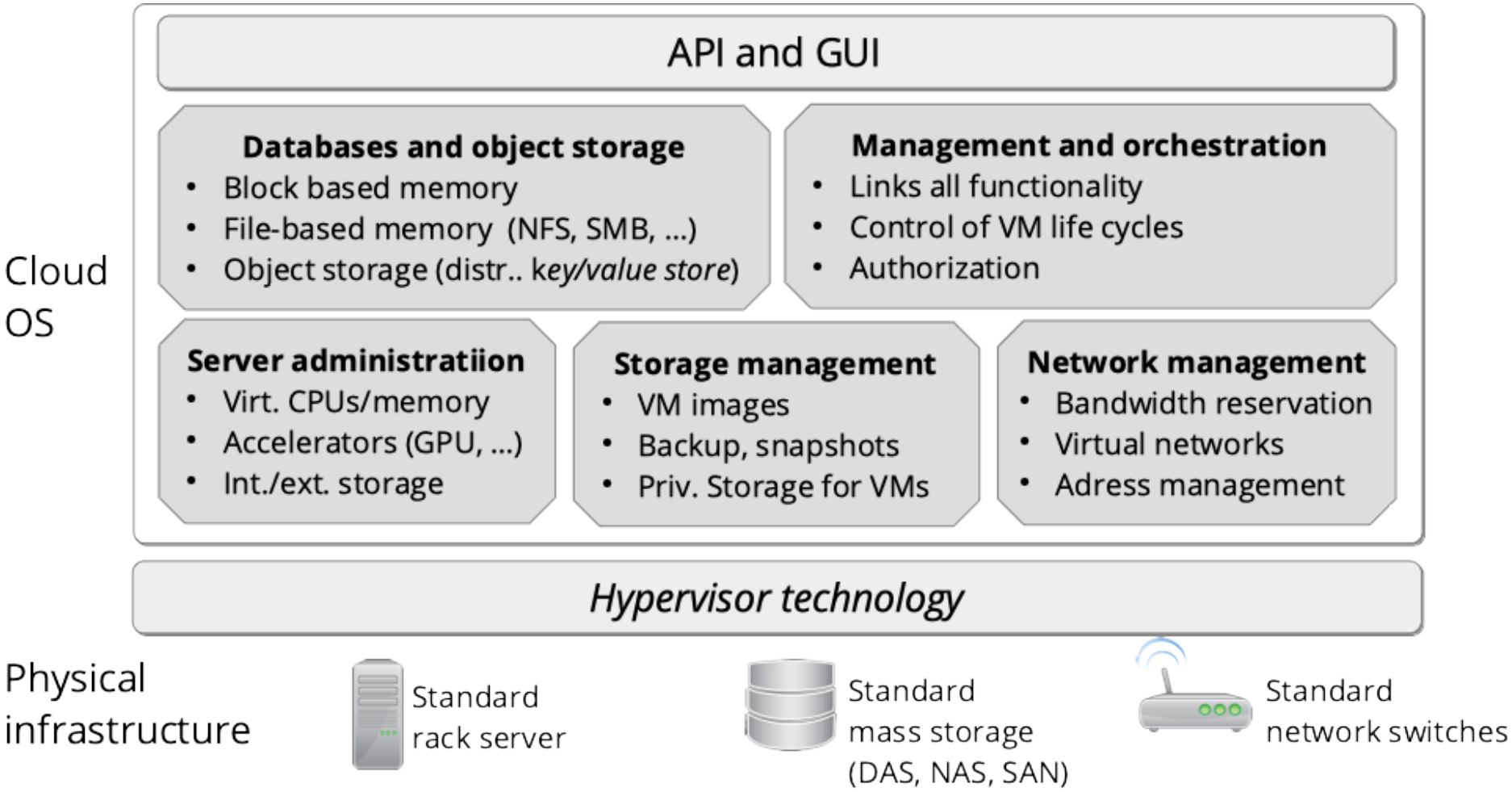
2

- Use (CSC) and observation (CSP)
- **Management** (CSP) (migration, redundancy, energy optimization, extensibility, ...)

3

General architecture of a cloud OS

- ***All*** resources are virtualized → IaaS is basis of all services



Strategic decisions

- Where to place the VMs? When should they be migrated?
- How to minimize SLA violations? How much **overbooking**?
- Does it make sense to release and switch off single computers?

Different strategies are possible:

Scheduler	Migrations	Violations			Penalties	Costs	Profit	Margin (in %)
		CPU	RAM	UT				
FF	28.326	12	370	3.350	87.792,85	31.537,03	-58.523,54	-96,2
HPGBF	5.355	401	212	60	90.558,46	30.381,79	-60.133,91	-98,9
HPGOP	2.372	114	101	33	30.942,97	31.093,17	-1.229,80	-2,0
HPGWF	3.705	49	141	0	24.760,00	30.781,68	5.264,65	8,7
MMBF	1.111	6	69	40	10.516,92	31.350,98	18.938,43	31,1
MMOP	825	2	34	7	4.934,46	32.066,84	23.805,03	39,1
MMWF	890	2	34	0	4.400,00	31.736,97	24.669,36	40,6
MMWF + LB	871	3	26	0	3.600,00	32.326,96	24.879,37	41,0

Evaluation of different VM configurations (initial distribution according to RAM resources) for the BitBrains RnD trace (month 1) with 500 VMs

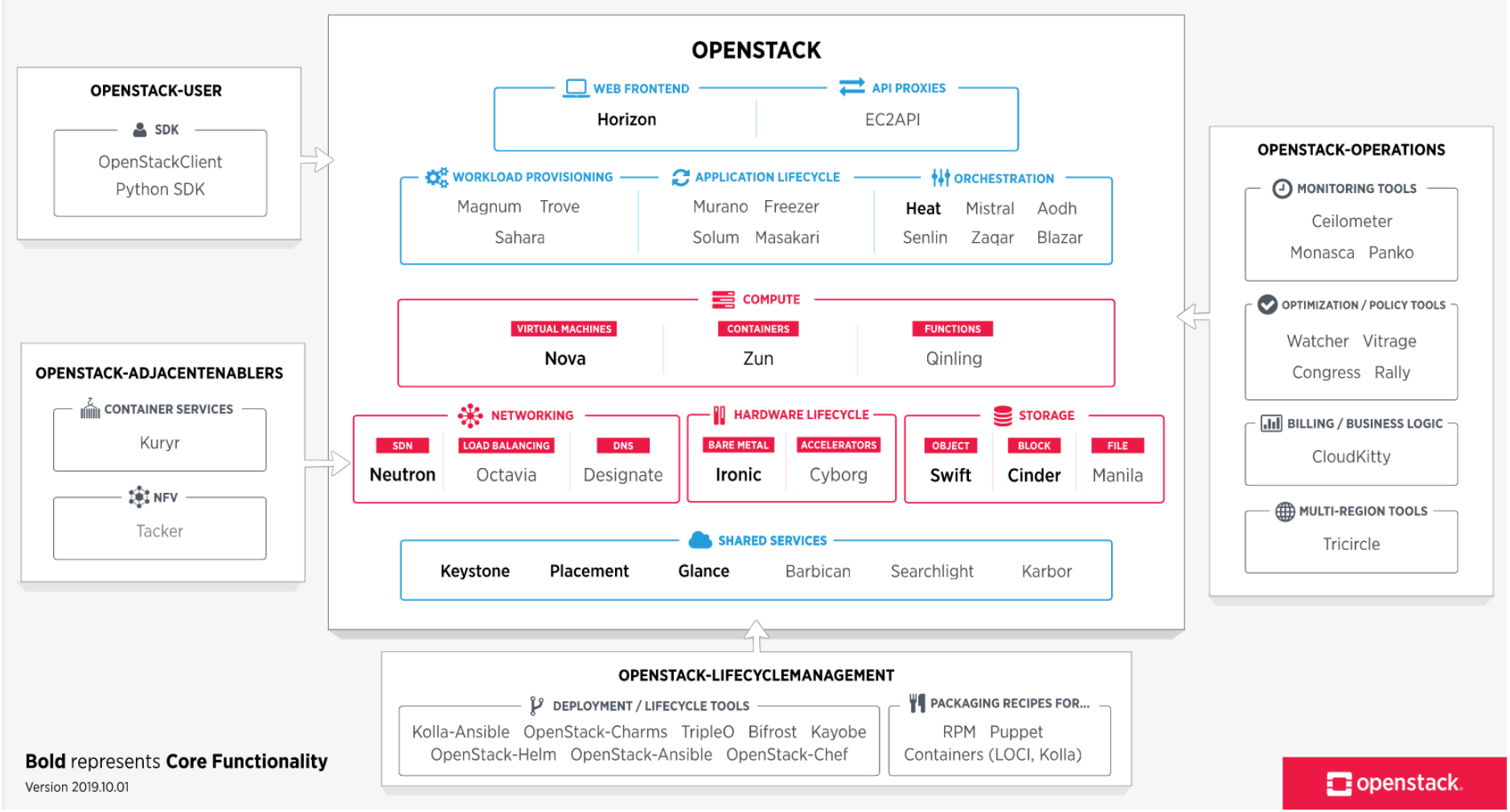
More resources were sold than are physically available

We do not go into details of the strategies here

Source:
Ph.D. thesis of A. Kohne,
“SLA-basierte VM-Scheduling-Verfahren für Cloud-Föderationen”

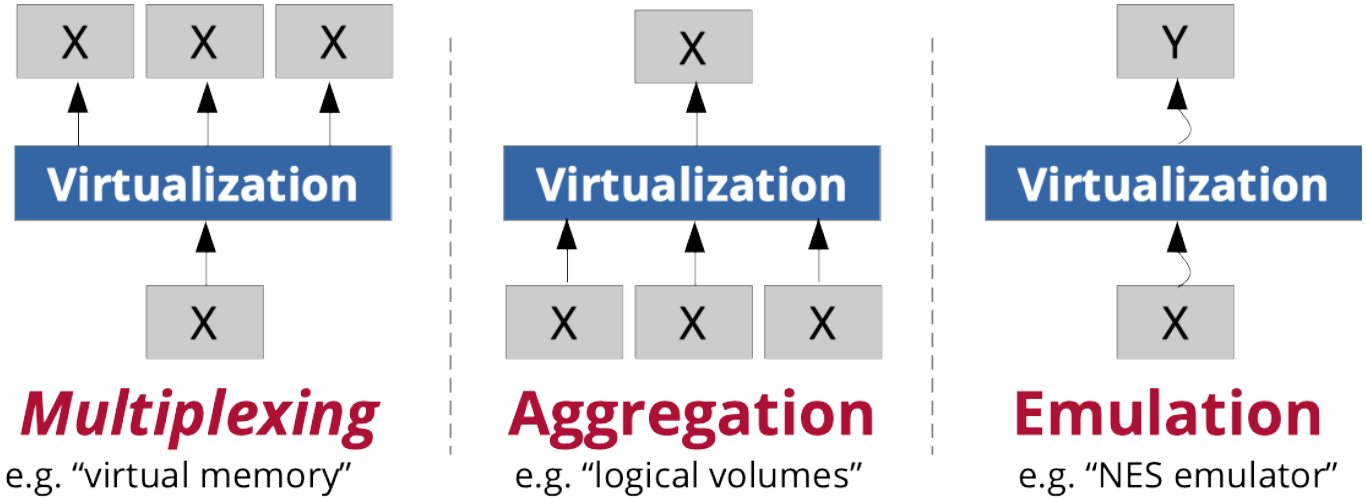
Example: OpenStack

- Open source cloud OS: www.openstack.org



Relevance and use of virtualization

- Enforces strict adherence to a layer structure through control and intervention possibilities for resource accesses by a VM
- Basis for...



X and Y are types of resources, e.g. RAM, disks, I/O devices

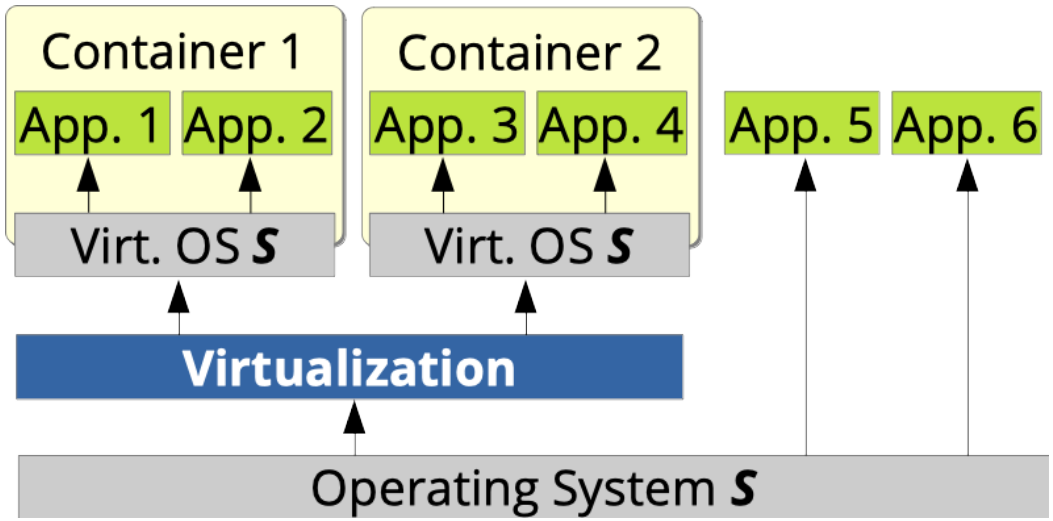
Source: [1]

- This construction principle can be replicated on different layers and for different resources

Container base virtualization

or simply ***containers***

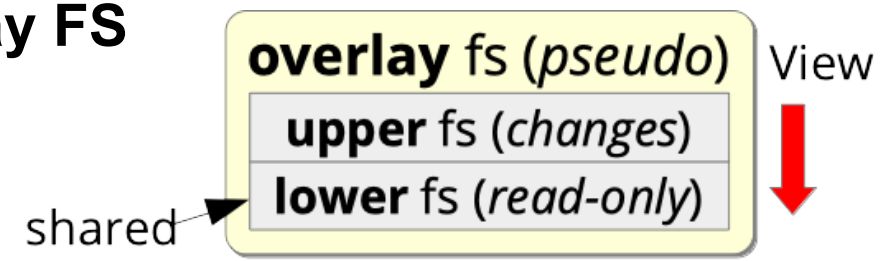
- The OS **kernel** is virtualized
  - Containers share a kernel
  - Libraries and system processes can be different



- The virtualization component takes care of...
  - **Separate views**, e.g. each container sees only its "own" processes
  - **Resource partitioning**, e.g. CPU time
  - **Efficient sharing**, e.g. avoid duplication of files

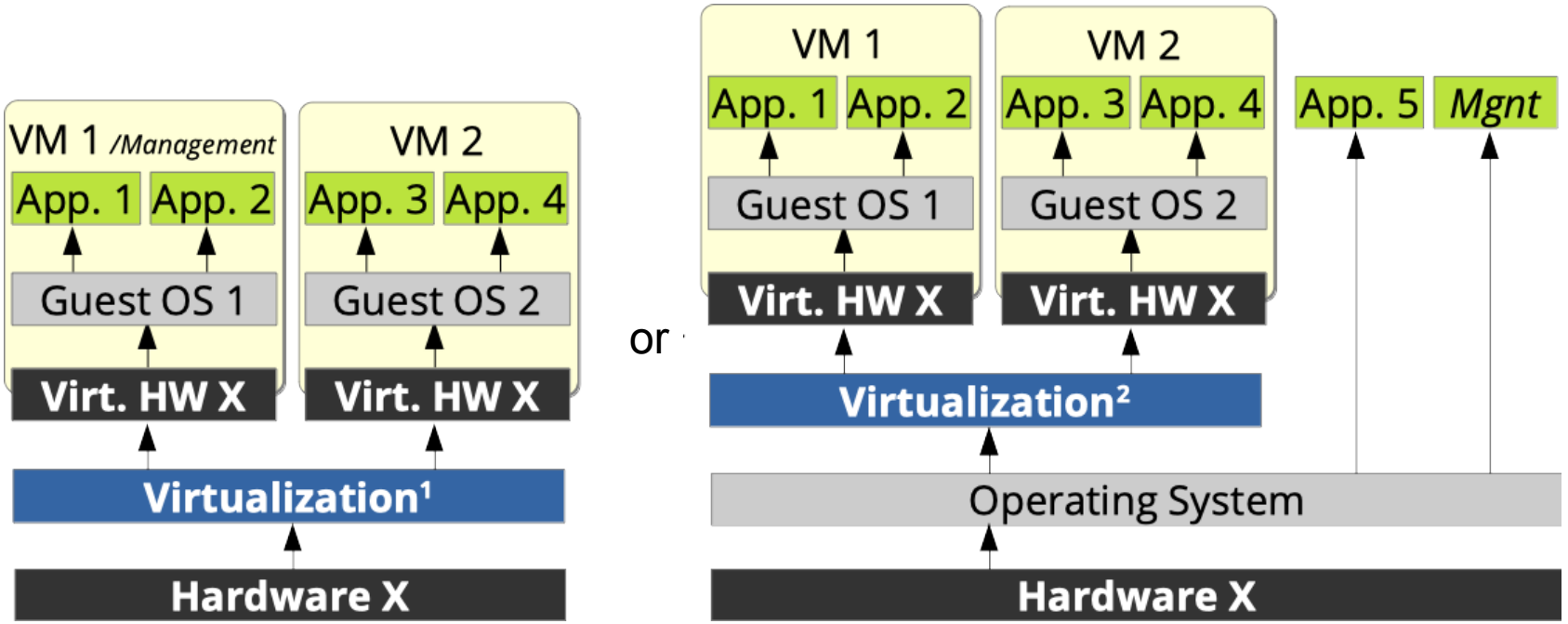
Example: Linux container support

- Integrated in the Linux kernel
  - Container solutions only provide management tasks
- Separate views: ***Name spaces*** per task
  - …for computer names (“UTS”), processes (“PID”), mount points (“mount”), network devices and configuration (“network”), IPC objects (“IPC”), control groups (“Cgroup”) and system time (“Time”)
- Resource partitioning: **Control groups** (cgroups)
  - Container shares of CPU time, memory and I/O bandwidth
  - Configuration interface: pseudo file system **cgroupfs**
- Efficient sharing (of files): **Overlay FS**
  - Overlay of directory trees



Hardware virtualization

- A complete computer (CPU, memory, I/O devices) is virtualized:



[1] Type-1 hypervisors provide virtual machines without the support of an operating system (directly on the hardware, "bare metal")

[2] Type-2 hypervisors work on top of a "host operating system".
They can use its capabilities, e.g. virtual memory

CPU virtualization (1)

- Most simple approach: **CPU emulation** (+ multiplexing)
  - **Interpretation** or **Just-in-time translation** (JIT) of the instructions of the emulated processor
  - Examples: Bochs, QEMU, MAME
- Imitates an ***arbitrary*** CPU Y with the help of a CPU X
- Problem: slow execution speed

Execution Mode	T1FAST.EXE time	T1SLOW.EXE time
Native	0.26	0.26
QEMU 0.9.0	10.5	12
Bochs 2.3.5	25	31
Bochs 2.3.7	8	10

Table 3.2: Execution time in seconds of Win32 test program

Conclusion:
avoid CPU emulation where possible

FAST/SLOW: with/without code optimization

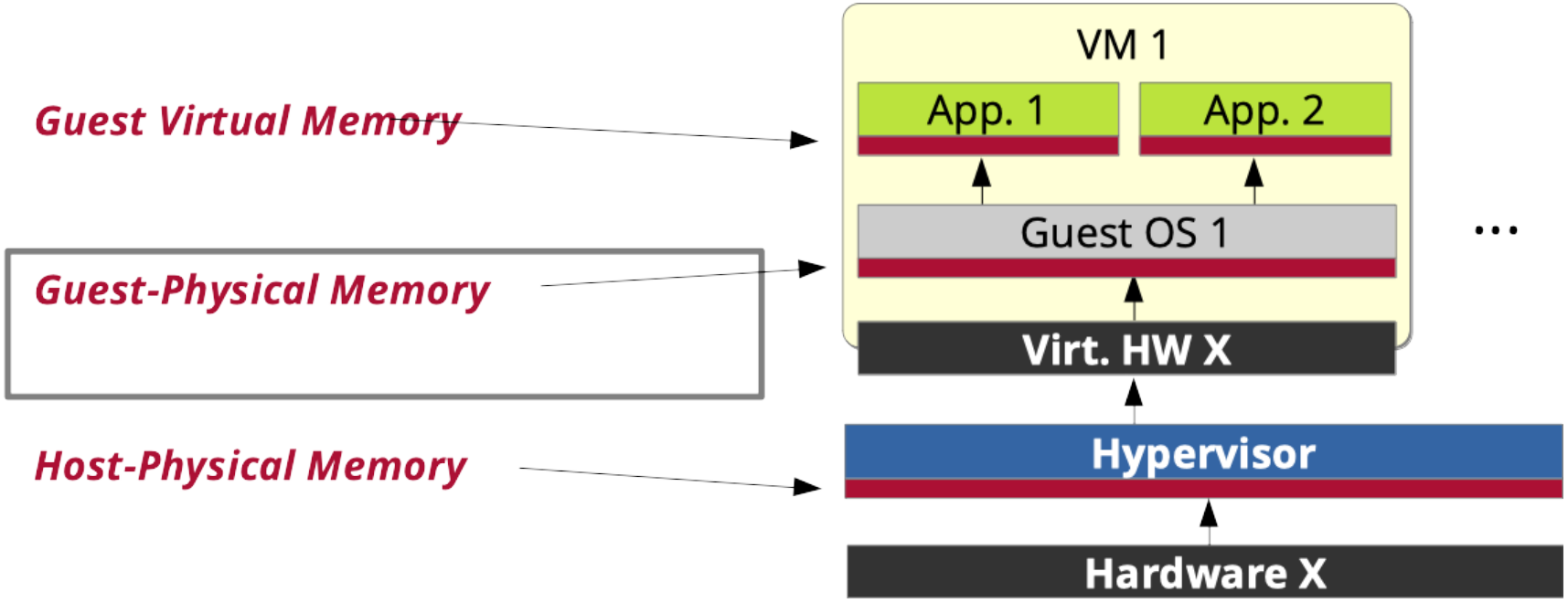
```
static int foo(int i) {
  return(i+1);
}
int main(void) {
  … <start timer>
  for(i=0; i<100000000; i++)
    t += foo(i);
  … <stop timer>
}
```

CPU virtualization (2)

- Efficient approach: **CPU multiplexing** (CPU X_1 , …, X_N on X)
- Desired properties ("virtualization criteria")
  - **Equivalence**: a VM behaves identical to the real machine
  - **Security**: a VM is isolated. The hypervisor has full control
  - **Performance**: virtual CPUs are not significantly slower than the real one
- *Question: which architectures are "virtualizable" in this way?*
- Answer (Popek and Goldberg, 1974 [3]):
  - CPUs have "sensitive" instructions which depend on the privileged mode of the CPU (user/supervisor mode, memory mapping, …) or switch its mode
  - All sensitive instructions must generate a ***trap*** when executed in user mode. This allows the hypervisor to emulate the instruction
- The "rest" works like a regular OS: VM scheduling

Memory virtualization (1)

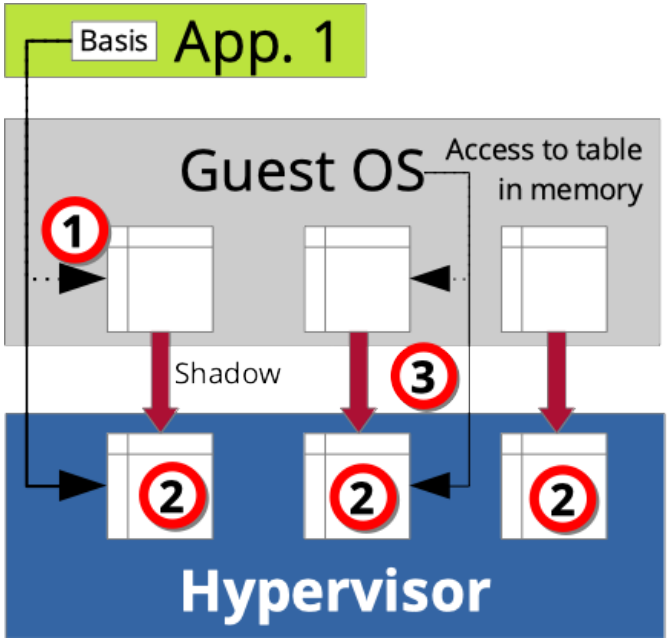
- Problem: additional memory mapping layer



Guest operating systems ***assume*** that they have complete control over the hardware. They use arbitrary page frames. Without the additional mapping layer, conflicts with other guest OSes could occur!

Memory virtualization (2)

- Solution 1: ***Shadow page tables***
  - Require no special hardware virtualization support
- Idea:
  1. Do ***not*** use the guest OS page tables
  2. Hypervisor keeps a ***shadow page table*** for each guest page table
  3. Shadow table must be kept ***up to date***!
     - Version 1: intercept and interpret all accesses to memory which stores part of a page table
     - Version 2: ignore changes, update tables when a page fault occurs

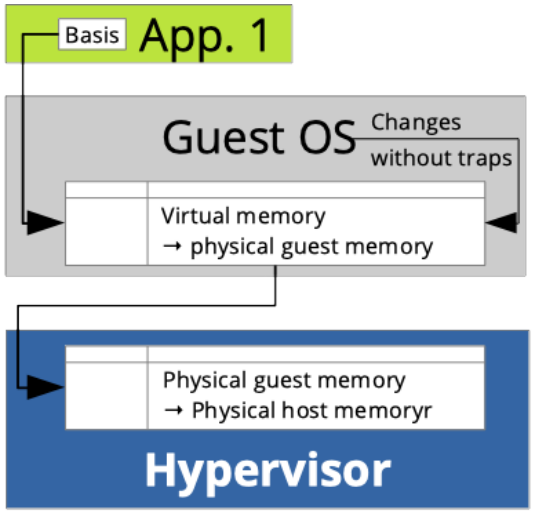


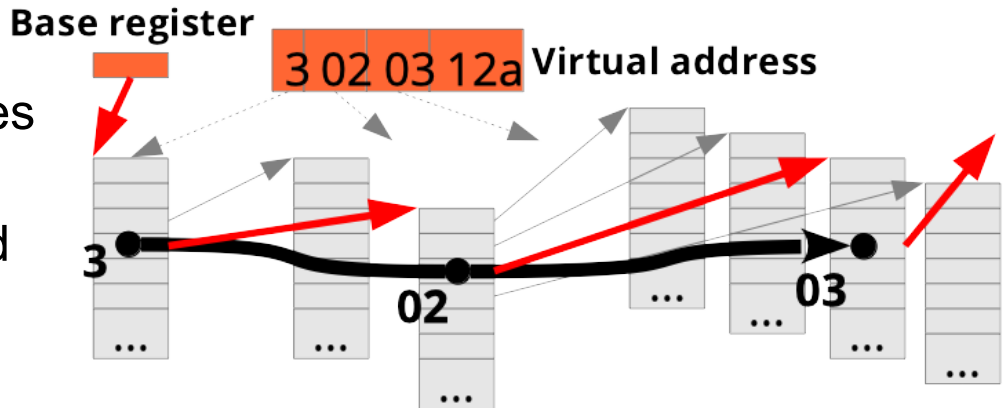
- Both variants result in many traps to the hypervisor ➛ **overhead**

Shadow page tables are expensive. Lower costs are possible using paravirtualization or hardware support

Memory virtualization (3)

- Solution 2: ***Nested page tables*** (AMD; Intel: "extended page tables"
- Idea:
  - Hardware is responsible for the complete memory mapping
  - Guest OS can change "its" page table as required
  - ***Page table walk*** is more expensive → greater relevance of the TLB
    - Page tables have tree structure
    - Pointers to tables are physical guest addresses
    - Translation to physical host addresses required (here: **4 translations**!)





Memory virtualization (4)

More approaches...

- **Ballooning**: "Trick" for dynamic allocation of memory to VMs
  - Small driver module communicates with the hypervisor
  - Can reserve memory of the OS kernel on demand
  - This memory can then be distributed to other VMs
- **Deduplication**: Detection and avoidance of duplicate page contents between VMs. Saves main memory, e.g. between identical guest OSes
- **VM migration**
  - Complete memory contents of a VM moved to other host system
  - Optimization: Transfer of pages while the VM is running
  - Recent changes are monitored using the dirty bit in the page table
- **VM replication**
  - Memory state changes are periodically transmitted to a backup host. Backup VM can replace one on a crashed/failed system quickly

I/O virtualization (1)

- Simple approach: **I/O emulation** (+ multiplexing)
  - Accesses to I/O device registers are privileged operations or can be intercepted by the hypervisor using the MMU (**"trap and emulate"**)
- Emulation of ***arbitrary*** I/O devices Y using I/O device X
  - e.g. in Oracle VirtualBox: PS/2 **mouse/keyboard**; IDE, SATA, SCSI, … **hard disk**; SVGA **graphics card**; different AMD and Intel **network controllers**; **USB** host controller; AC‘97, Intel HD or Soundblaster 16 **sound cards**
- Problem: I/O throughput
  - Even simple I/O operations require hundreds or thousands of I/O register accesses!

I/O emulation is expensive. Lower costs are (again) possible using paravirtualization or hardware support

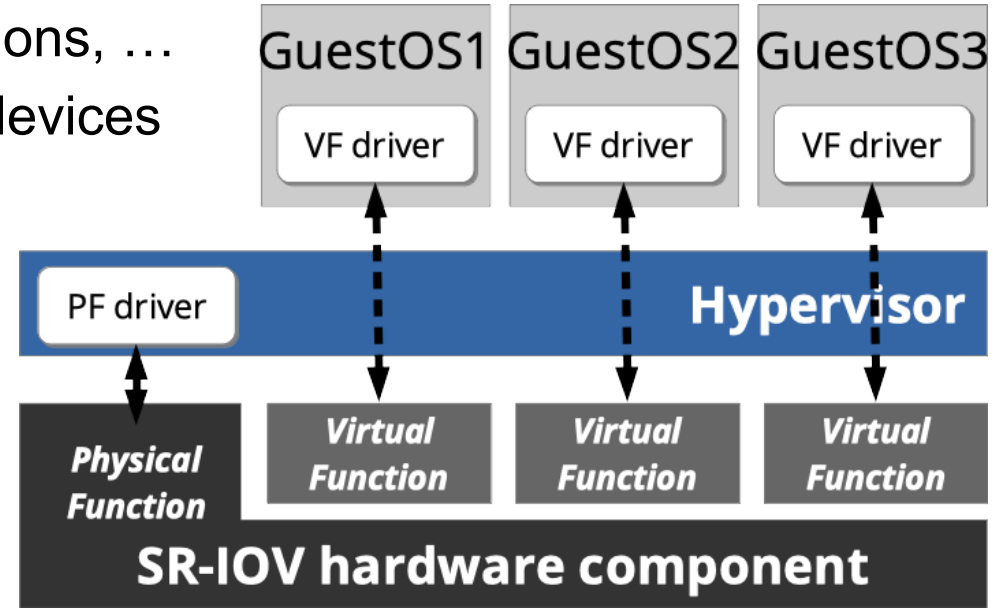
I/O virtualization (2)

- Alternative: do not use multiplexing – **device passthrough**
  - A device is exclusively assigned to exactly one VM
  - Arbitrary register accesses are permitted (without causing a trap)
- Problems:
  - DMA addresses are physical host addresses not known to the VM
  - This could be used to violate the VM isolation
  - Interrupts could be triggered on the "wrong" CPU
- Solution: **I/O MMU**
  - Hardware extension implemented in CPU or mainboard chip set
  - DMA uses an **address mapping** using tables
    - Acceleration using separate TLBs
  - **Interrupt remapping** is able to change the interrupt number and destination CPU

I/O virtualization (3)

- Alternative 2: **PCIe single root I/O virtualization** (SR-IOV)
- **Hardware mechanism**: One device appears as multiple virtual ones
  - Multiple I/O register sets, multiple interrupt configurations, …
- Hypervisor maps one of these devices to a VM and does not have to interfere further

- Possible problem:
  - Hardware takes care of the prioritization of VMs itself
    - e.g. round robin
  - Conflicts with priorities of the hypervisor are possible



Conclusion

- Virtualization is an important **architectural concept** recurring in the system software stack
  - Transparent: Multiplexing, aggregation, emulation

- Hardware virtualization (according to Popek/Goldberg)
  - Replaces inflexible connection of hardware and software
    - enables e.g. migration and replication of VMs at runtime
  - Technical basis for cloud computing

- Operating systems for clouds
  - Well-known functionality:
    Resource management and abstractions
  - …implemented on a higher layer

References

[1] Edouard Bugnion, Jason Nieh, and Dan Tsafrir. 2017. Hardware and Software Support for Virtualization. Morgan & Claypool Publishers, 2017.

[2] Mihočka, Darek, Stanislav Shwartsman and Intel Corp. Virtualization Without Direct Execution or Jitting: Designing a Portable Virtual Machine Infrastructure.”, 2008.

[3] Gerald J. Popek and Robert P. Goldberg. 1974. Formal requirements for virtualizable third generation architectures. Commun. ACM 17, 7 (July 1974), 412–421.DOI:https://doi.org/10.1145/361011.361073