

# Themen zur Computersicherheit

## Autorisierung HW & Unix Mechanismen

PD Dr. Reinhard Bündgen  
[buendgen@de.ibm.com](mailto:buendgen@de.ibm.com)

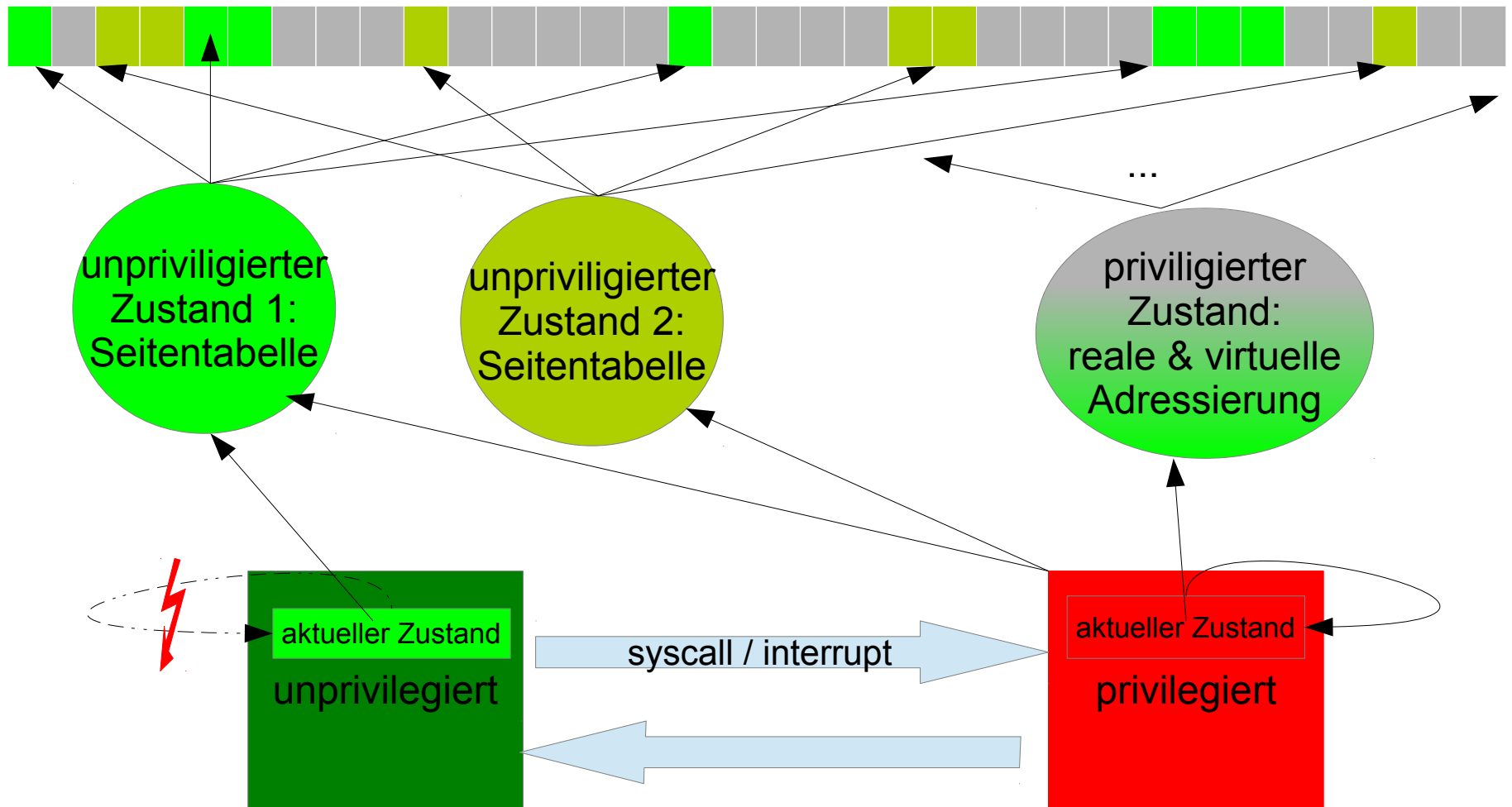
# CPU Modi und Speicherzugriff

- im unprivilegierten CPU Modus
  - nur virtueller Adressmodus
  - alle Instruktionen nutzen virtuelle Adressen
  - nur ein Teil der Instruktionen sind unterstützt
    - keine Instruktionen zur Verwaltung von Seitentabellen
  - alle Programme/Prozesse laufen in unprivilegiertem Modus
  - ein *Systemruf* fordert einen privilegierten Dienst an
    - führt zu einem Kontextwechsel in privilegierten Modus
- im privilegierten CPU Modus
  - alle Adressierungsmodi (real & virtuell)
  - alle Instruktionen unterstützt
    - einschließlich Instruktionen zu Verwaltung von Seitentabellen
  - nur der Betriebssystemkern läuft im privilegiertem Modus

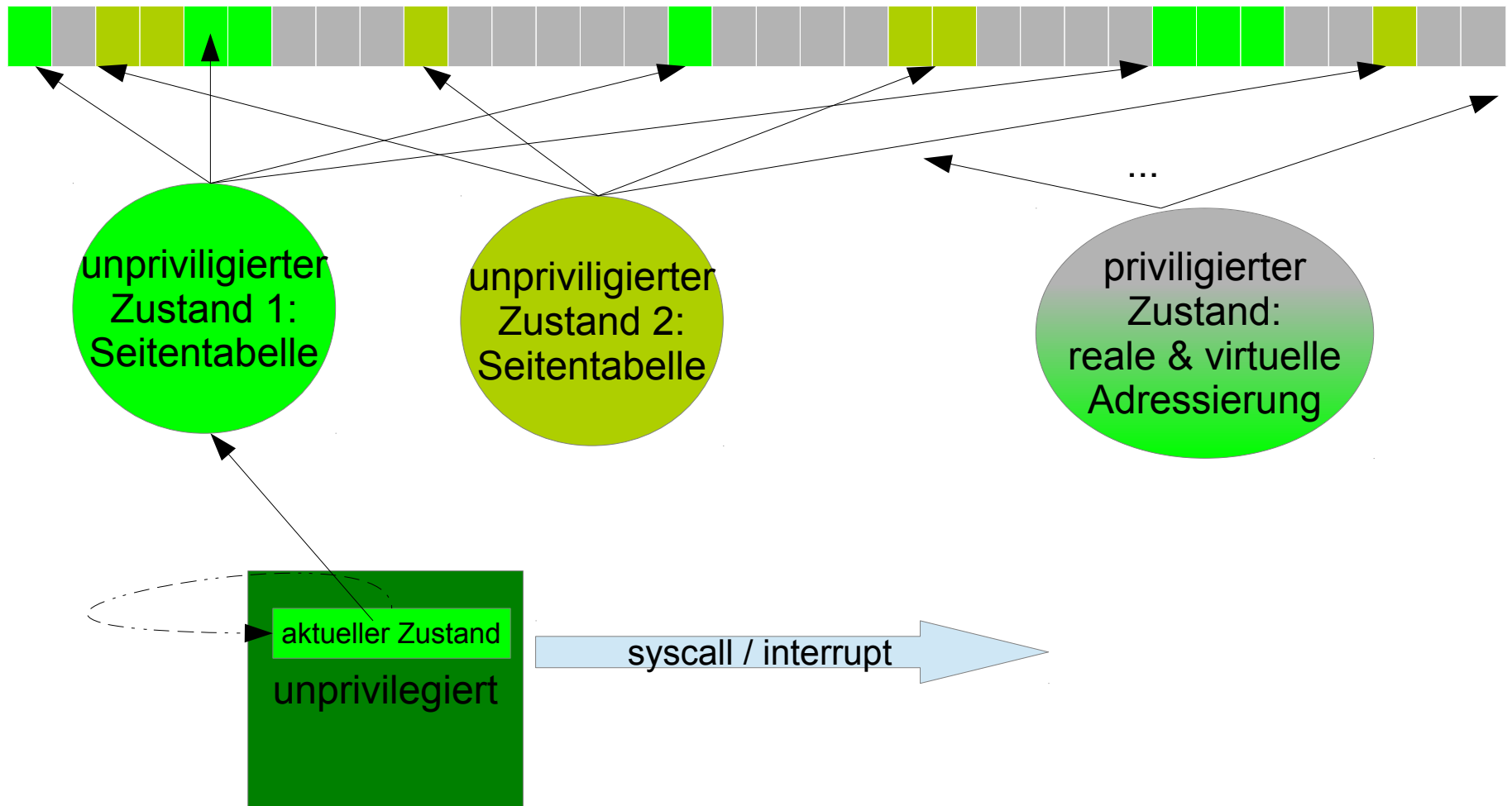
# HW Kontrollmechanismen (MAC)

- privilegierter/nicht privilegierter Modus der CPU
- Virtuelle Adressräume
- Unified Extensible Firmware Interface (UEFI) Secure Boot
- Trusted Platform Module (TPMs)
- Hardware Security Module (HSMs)
- Geschützte Berechnungen: SGX, SME, SEV

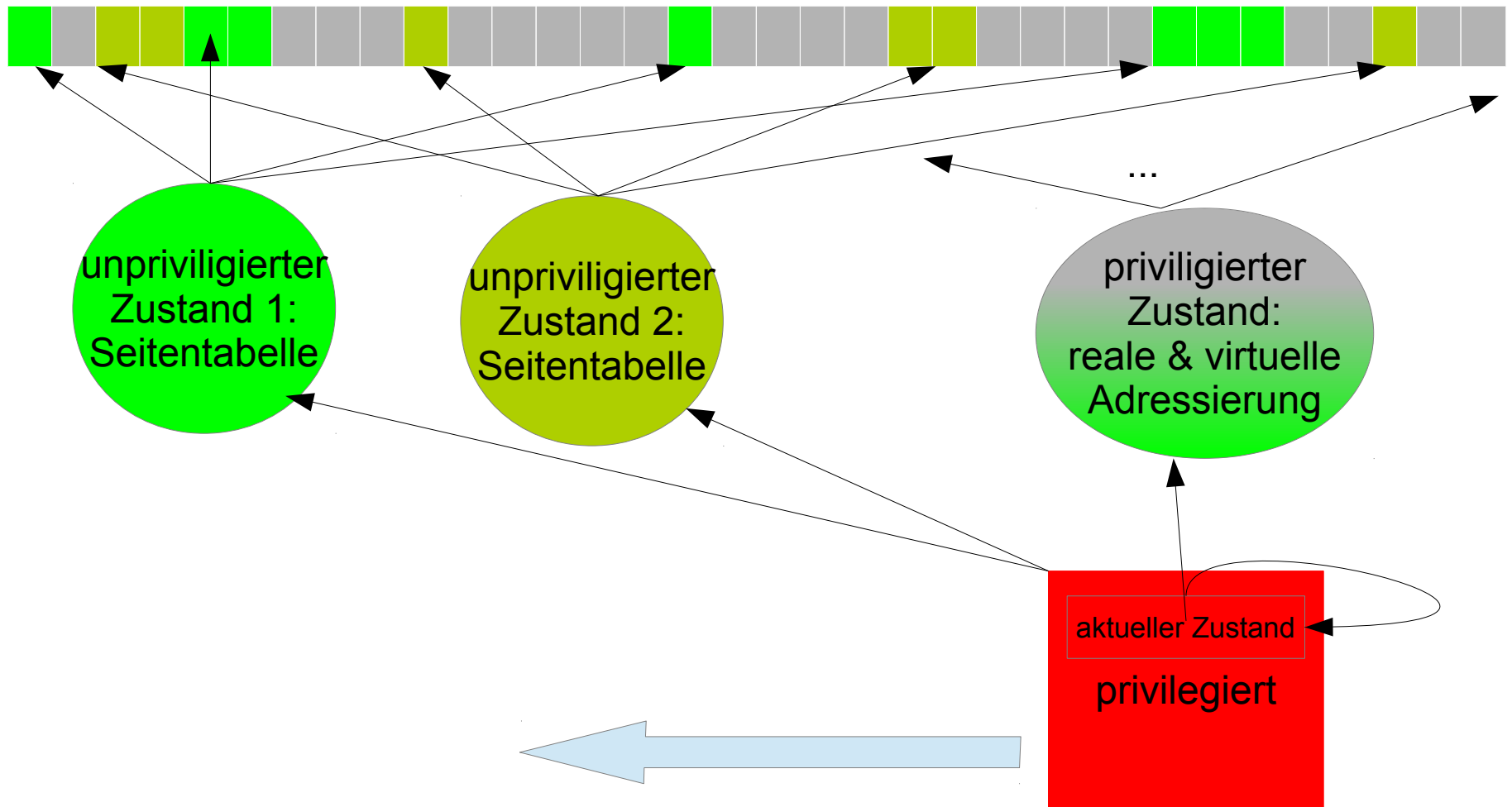
# CPU Modi & Virtueller Speicher



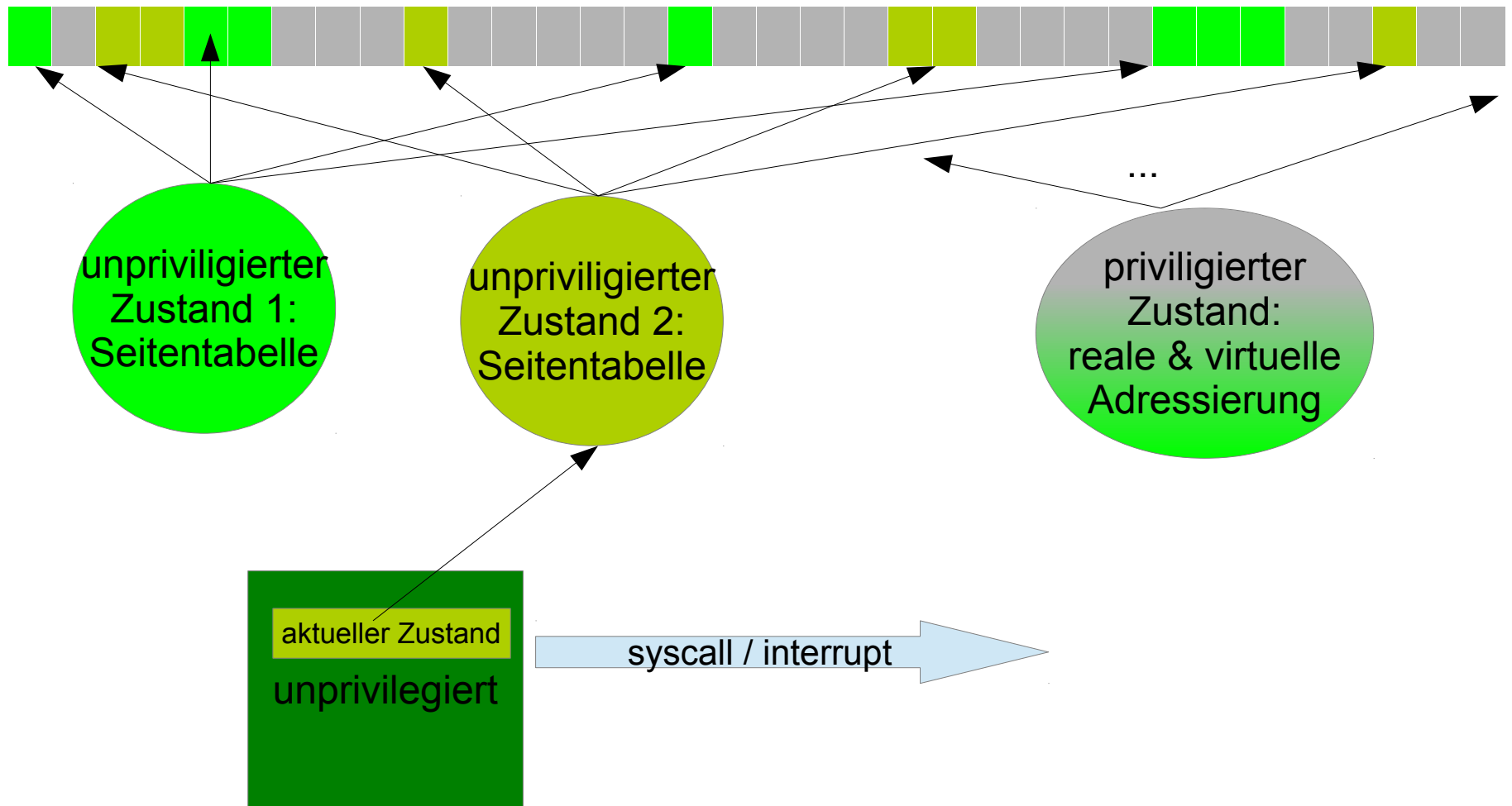
# CPU Modi & Virtueller Speicher



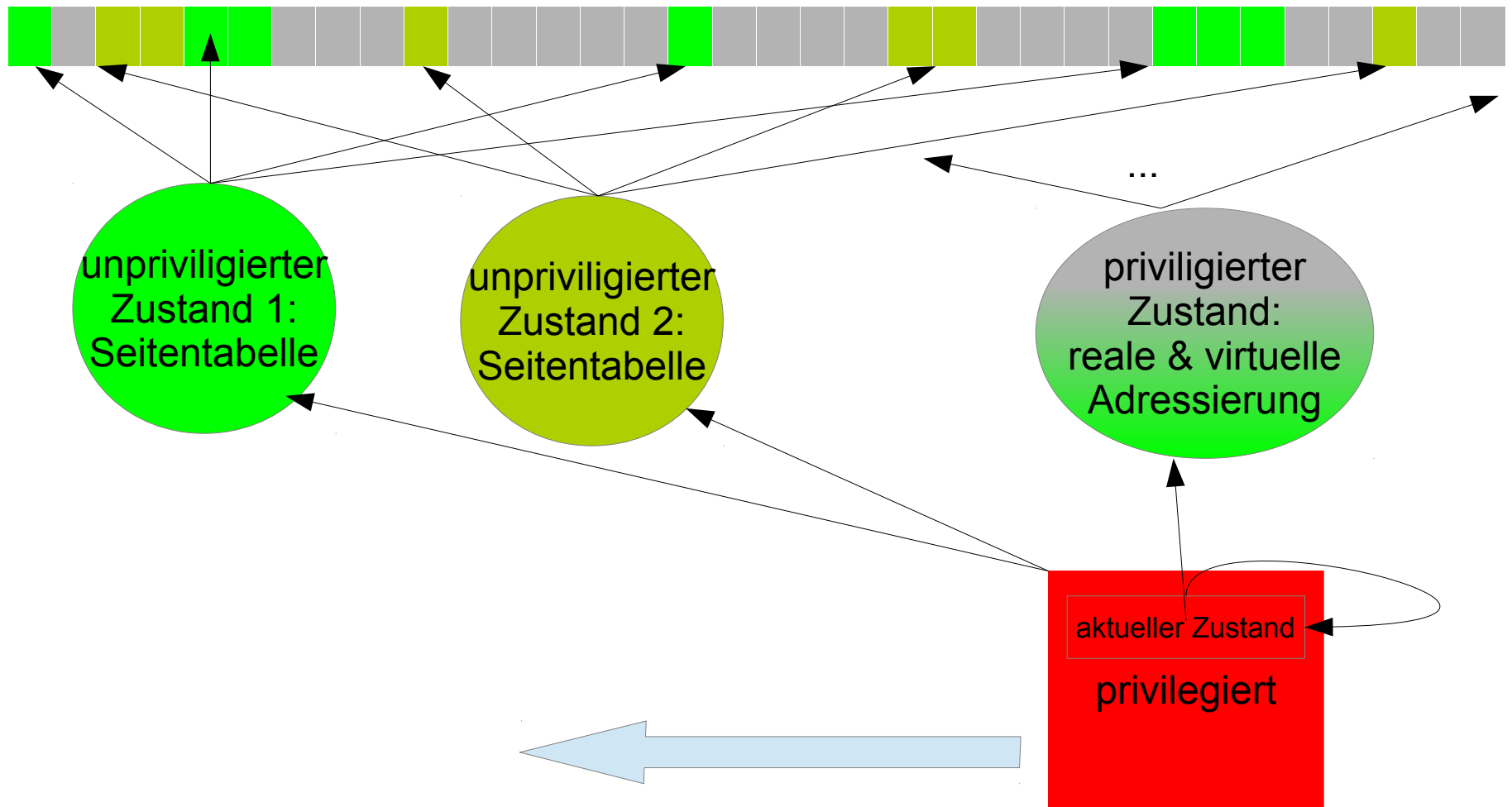
# CPU Modi & Virtueller Speicher



# CPU Modi & Virtueller Speicher

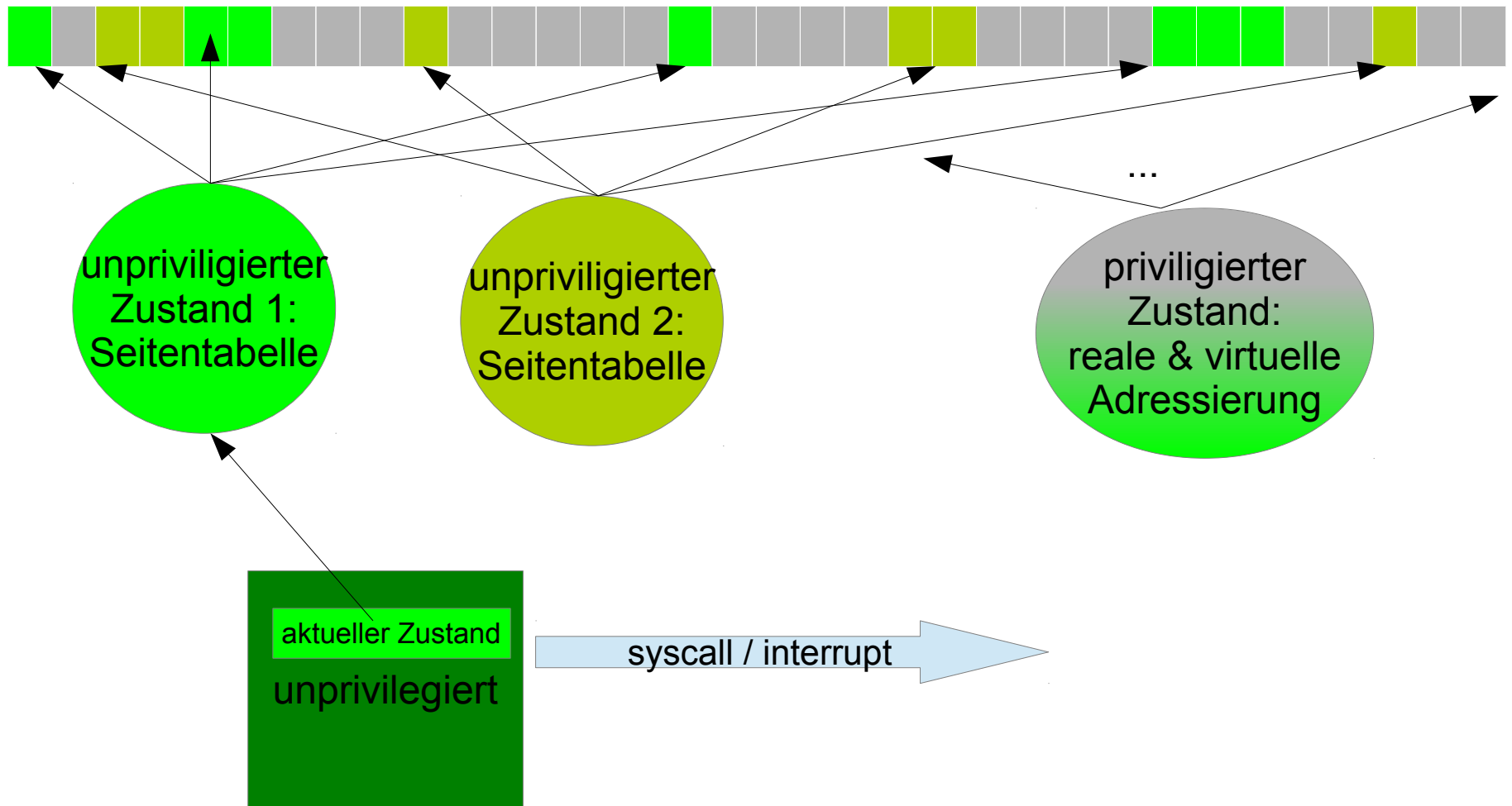


# CPU Modi & Virtueller Speicher





# CPU Modi & Virtueller Speicher



# Traditionelle Unix-Dateizugriffskontrolle

- Discretionary Access Control (DAC)
- Jede Datei hat einen Besitzer und eine Gruppe
- Jeder Teilnehmer ist Mitglied einer oder mehrerer Gruppen (/etc/group)
- Jede Datei kennt drei Zugriffsarten: read, write, execute (r,w,x)
  - die jeweils für den Besitzer, die Gruppe der Datei oder alle Teilnehmer des Systems getrennt vergeben oder verwehrt werden
  - für Verzeichnisse beschreibt der execute Zugriff, den Durchgriff durch das Verzeichnis
- setuid und setgid Bits für ausführbare Dateien:
  - Prozess, der Datei ausführt, bekommt Dateibesitzer als effektive uid bzw Dateigruppe als effektive gid
- setgid Bit im Verzeichnis:
  - Dateien im Verzeichnis erben den Besitzer des Verzeichnisses
- sticky Bit im Verzeichnis:
  - Teilnehmer darf eine Datei in dem Verzeichnis nur dann löschen, wenn er Schreibzugriff auf das Verzeichnis hat und zusätzlich entweder Besitzer der Datei oder des Verzeichnisses ist oder Superuser ist.

# NFSv4 ACLs

- Viele neue Unix System unterstützen NFSv4 Zugriffssteuerungslisten (access control lists, ACLs) via extended attributes
- z.B. AIX, FreeBSD, MAC OS X, Solaris (ZFS), Linux (Ext3, Ext4, Btrfs)
- ext3:
  - Montageoption „-o acl“
  - setfacl Werkzeug verwaltet ACLs
    - `setfacl -m u:buendgen:rw /home/vorlesung/script.tex`
  - getfacl zeigt ACLs
- MAC OS X
  - `chmod +a`
  - `ls -l -e`

# Beschränkung der Allmacht von Root

- setuid Programme
- Systeme ohne Root Login
  - z.B. OS X, Ubuntu
  - sudo
- Linux Capabilities
- seccomp
  - Sandkasten: Einschränkung von Systemrufen
- Linux Security Modules,
  - z. B. SELinux, apparmor

# Programmstart in Unix

Erzeugen eines neuen  
Prozesses:

```
pid=fork();
if (pid == 0) {
    /* execute child code */
    ...
} else if ( pid > 0 ) {
    /* execute parent code */
    ...
} else {
    /* error */
}
```

Starten eine Programms

```
pid=fork();
if (pid == 0) {
    /* execute child code */
    /* program start: */
    execve(„/bin/ls“,...);
} else if ( pid > 0 ) {
    /* execute parent code */
} else {
    /* error */
}
```

fork() kopiert einen Prozess einschließlich einer Kopie des Adressraumes  
execve() lädt ein Programm in einen Adressraum und startet es

# Linux Capabilities (I)

- Voraussetzungen zur Unterstützung von Capabilities:
  - Linuxkern überprüft für jede privilegierte Operation, ob aufrufender Thread über ausreichende capabilities verfügt
  - Linuxkern unterstützt Systemrufe um die Capabilities eines Threads zu inspizieren bzw zu ändern
  - Das Dateisystem muss die Assoziation von Capabilities mit ausführbaren Dateien unterstützen

# Linux Capabilities (II)

Capability: binärer Wert, der wenn gleich 1 eine Erlaubnis bezeichnet

- Thread Capabilities
  - permitted  $T(p)$ : maximal erreichbare Capabilities
  - inheritable  $T(i)$ : Capabilities, die bei einem `execve`-Aufruf geerbt werden können
  - effective  $T(e)$ : Capabilities, die der Kern überprüft bevor er eine Erlaubnis erteilt
  - `cap_bset` („bounding set“): beschränkt die durch `execve` erlangbaren capabilities
- File Capabilities (für ausführbare Programme)
  - permitted: fließen in die permitted capabilities des Threads ein
  - inheritable: ihre Schnittmenge mit den thread capabilities fließen in die permitted capabilities des threads ein
  - effective: 1 Bit, das wenn 1 besagt, dass die effective capabilities des Threads gesetzt werden sollen

# Capability Änderungen (III)

- **mit cap\_set\_proc**

- if  $\neg \text{CAP\_SETPCAP}$  then
  - $T'(i) \subseteq T(i) \cup T(p)$
  - $T'(i) \subseteq T(i) \cup T(b)$
  - $T'(e) \subseteq T(p)$
  - $T'(b) = T(b)$
- else
  - $T'(p) \subseteq T(p), T'(b) \subseteq T(b)$

- **UID Änderungen**

- alle uids nach nicht-0:
  - $T'(p) = \emptyset, T'(e) = \emptyset$
- euid nach nicht-0:
  - $T'(e) = \emptyset$
- nicht-0 euid nach 0:
  - $T'(e) = T(p)$

- **execve(F) (non-setuid Datei)**

- $T'(p) = (T(i) \cap F(i)) \cup (F(p) \cap T(b))$
- if F(e) then
  - $T'(e) = T'(p)$
- else
  - $T'(e) = \emptyset$
- $T'(i) = T(i)$
- $T'(b) = T(b)$

- **fork**

- $T'(p) = T(p)$
- $T'(e) = T(e)$
- $T'(i) = T(i)$
- $T'(b) = T(b)$

## Notation

T: Thread Capability vor Änderung  
T': Thread Capability nach Änderung  
F: Datei Capability vor Änderung  
F': Datei Capability nach Änderung



# Capabilities (IV)

- Problem: nicht-root Threads können keine Datei Capabilities erben, wenn  $F(i)$  leer
- neues „ambient“ Capability Set  $T(a)$  mit:
  - $T(a) \subseteq T(p) \cap T(i)$
- **execve(F)**
  - if F is setuid or setgid or has file caps then
    - $T'(a) = \emptyset$
  - else
    - $T'(a) = T(a)$
    - $T'(p) = (T(i) \cap F(i)) \cup (F(p) \cap T(b)) \cup T'(a)$
  - if F(e) then
    - $T'(e) = T'(p)$
  - else
    - $T'(e) = T'(a)$
    - $T'(i) = T(i)$
    - $T'(b) = T(b)$

# Dokumentation zu Linux Capabilities

- man (7) capabilities
- über das ambient capability set:
  - <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=58319057b7847667f0c9585b9de0e8932b0fdb08>
  - <http://lwn.net/Articles/632520/>

# Capabilities Beispiel

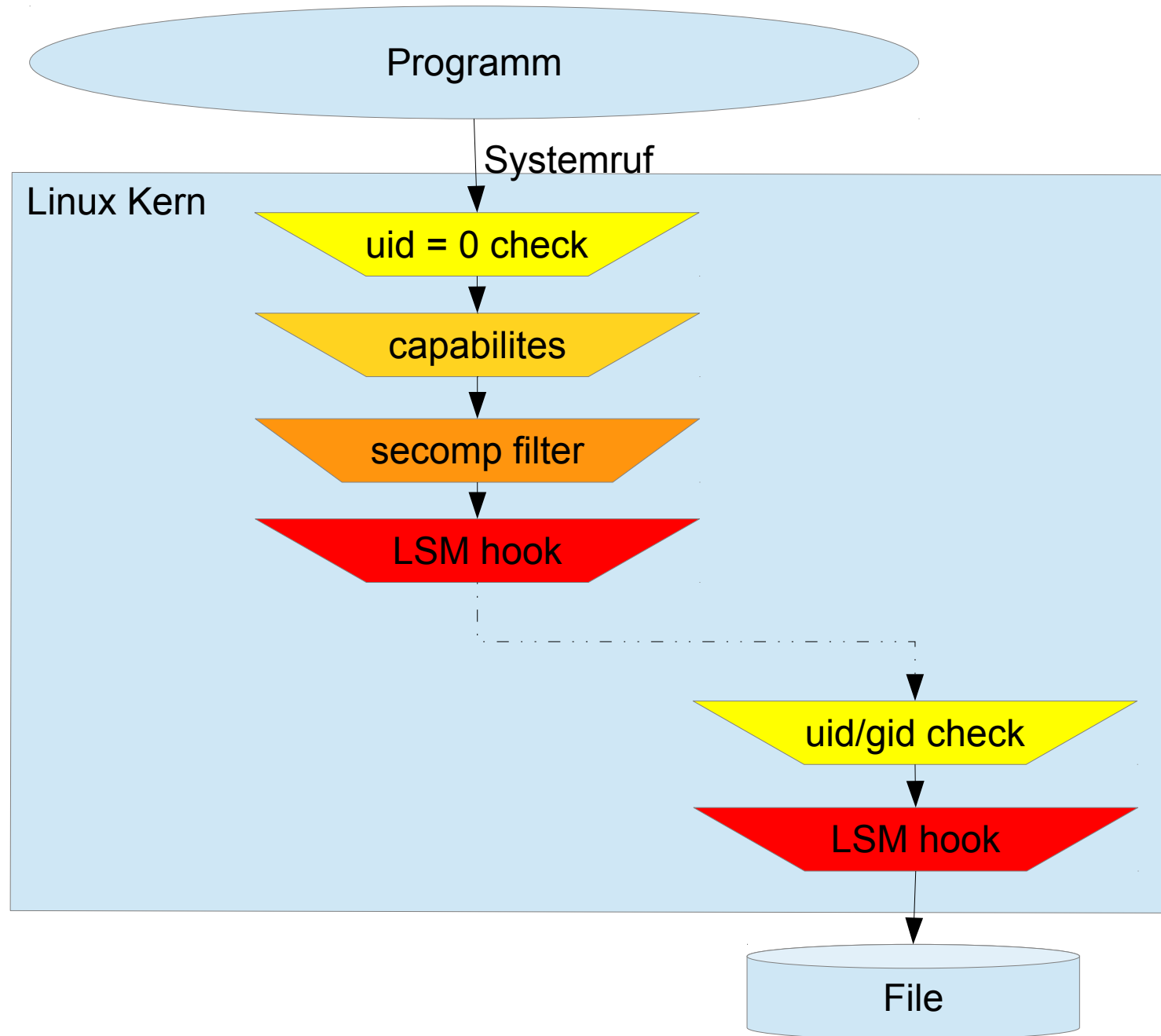
Ubuntu

```
reiner@MacLinux2: ~  
reiner@MacLinux2:~$ ls -l /bin/ping  
-rwsr-xr-x 1 root root 35712 Nov  8 2011 /bin/ping  
reiner@MacLinux2:~$
```

Fedora

```
Datei Bearbeiten Ansicht Suchen Terminal Hilfe  
[reiner@localhost ~]$ ls -l /usr/bin/ping  
-rwxr-xr-x. 1 root root 44776 17. Aug 2014 /usr/bin/ping  
[reiner@localhost ~]$ getcap /usr/bin/ping  
/usr/bin/ping = cap_net_admin,cap_net_raw+ep  
[reiner@localhost ~]$
```

# Zugriffskontrolle durch Linux Kern



# Aufgaben

- Warum kann ein Prozess nicht auf Speicher im Adressraum eines anderen Prozesses zugreifen?
- Wie unterscheiden sich die Rechte des Betriebssystemkerns von denen eines Prozesses mit vollen Administratorrechten?
- Wie muss ein Systemruf (bzw eine Unterbrechung) HW-seitig implementiert sein um nicht privilegierte Prozesse von anderen (nicht-privilegierten) Prozessen abzuschotten?
  - Was muss der Betriebssystemkern als Reaktion auf einen Systemruf (bzw eine Unterbrechung) tun?
- Beschreiben Sie, wie Sie mit Unix-Mitteln
  - ein Verzeichnis für andere unlesbar machen, aber die darin enthaltene Datei Übungsblatt für alle Mitglieder Ihrer Übungsgruppe lesbar machen
  - eine Datei mit einem Freund gemeinsam lesen und schreiben können und alle anderen vom Zugriff ausschließen?
- Kann mit Standard Unix-Zugriffsrechten RBAC implementiert werden?