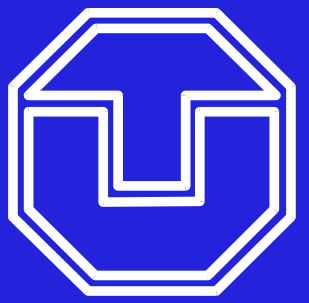
# Sicherheits-mechanismen

## Betriebssysteme

Hermann Härtig
TU Dresden



# Konstruktion sicherer Systeme

**Entwurfsprinzipien** – SALTZER und SCHRÖDER (schon 1975!)

- Prinzip der geringst-möglichen Privilegisierung
- nur die Rechte einräumen, die für die zu erbringende Funktionalität erforderlich sind
  - Verbot als Normalfall
  - Whitelisting vs. Blacklisting
  - Gegenbeispiel: Unix „root“
- Sichere Standardeinstellungen („fail-safe defaults“)
- Separierung von Privilegien (Separation of duty)
  - mehrfache Bedingungen für die Zulassung einer Operation

# Konstruktion sicherer Systeme

**So einfach wie möglich**

- reduziert Fehlermöglichkeiten bei Implementierung und Einsatz
- klares mentales Modell bei Nutzern erzeugen

**Offenheit**

- Sicherheit darf nicht auf der Geheimhaltung von Entwurf und Implementierung basieren

**Psychologisch akzeptabel**

- Verantwortungsbereiche der Entwickler und Nutzer sinnvoll abwägen
- Nutzer werden das System für sie „ökonomisch“ bedienen

# Konstruktion sicherer Systeme

**Vollständige Kontrolle (complete mediation)**

- jede Aktion, die potentiell ein Schutzziel verletzt, sollte kontrolliert werden
- mehrschichtige Absicherung (security in depth)
- Gegenbeispiel: Zugriff auf offene Dateien in Unix
- Aktualität von Prüfungen
  TOCTTOU: time of check to time of use

**Basiselemente**

- Ausgangspunkt: Isolation
- gezieltes Durchbrechen der Isolation: Kommunikation
- Regeln für diese Durchbrüche

# Sicherheit und Systemebenen

| | |
|---|---|
| Umgebung: | physische Sicherung |
| HW: | User/Kernel-Modi<br>Adressraumseparierung<br>Sicheres Booten<br>Verschlüsselter Speicher<br>Schlüsselverwaltung |
| BS-Kern: | Kapselung (Isolation) von Prozessen,<br>Virtuellen Maschinen, Containern<br>Sichere Kommunikationskanäle |
| Systemdienste: | Authentifikation<br>Zugriffssteuerung<br>Protokollierung |
| Benutzer/Entwickler: | Einsatz der Mechanismen |

# Sicherheitsakteure

**Benutzer**

- erzeugen Prozesse und lassen sie für sich arbeiten
- speichern langlebige Daten in Dateien/Dokumenten/ Speicherobjekten
- steuern die Zugriffsrechte auf Objekte

**Entwickler**

- schreiben Anwendungen, welche die Benutzer installieren
- wissen welche Ressourcen ihren Anwendungen im Normalbetrieb benötigen
- jede Anwendung ist ein potentielles Sicherheitsrisiko

# Prozesse und ihre Interaktion

**Grundsätzliche Annahme**

Prozesse versuchen nicht erlaubte Operation durchzuführen

- Dateien lesen/schreiben
- Ressourcen benutzen (z.B. Netzwerk)
- Informationen austauschen

**Fragestellungen:**

- Wie legt man Rechte fest?
- Wie setzt man deren Einhaltung durch?

# Subjekte und Objekte

Operation(Subjekt, Objekt) zulässig?

**Subjekte**

- Prozesse („Security Domains“)

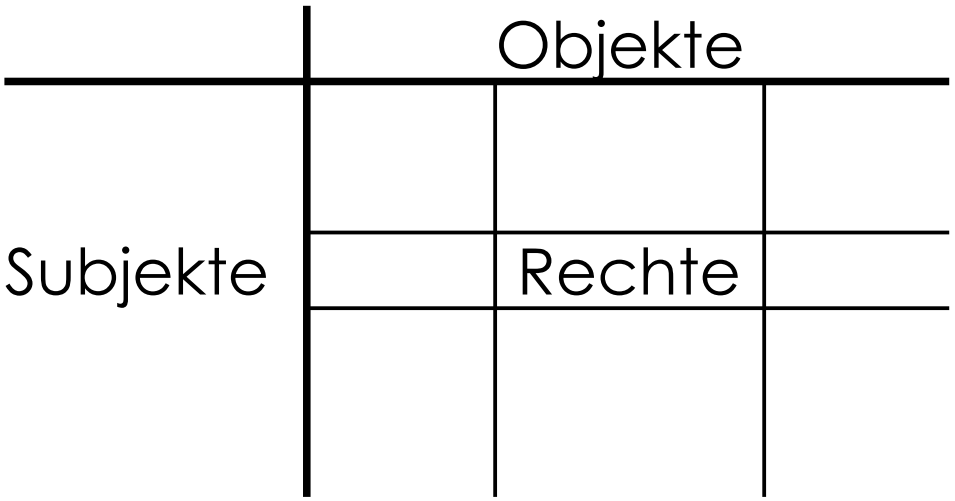
**Objekte**

- Dateien, Geräte,
  Prozesse, Kommunikationskanäle,
  physischer Speicher, virtueller Speicher
- Bankkonten, …

**Operationen**

- Lesen, Schreiben, Löschen, Ausführen
- Einzahlen, Abheben, …

# Schutzmatrix




**f(Subjekt, Objekt, Operation) → zulässig/verboten**

# Ausprägungen der Schutzmatrix

**Spaltenweise: ACL – Access Control List**

- bei jedem Zugriff wird beim Objekt auf der Basis der Identität des Aufrufers dessen Berechtigung geprüft
- Festlegung durch Besitzer der Objekte

**Zeilenweise: Capabilities**

- bei jedem Zugriff wird etwas geprüft, was Subjekte besitzen und bei Bedarf weitergeben können
- Beschränkung durch initiale Vergabe der Capabilities und Festlegungen seitens der Subjekte

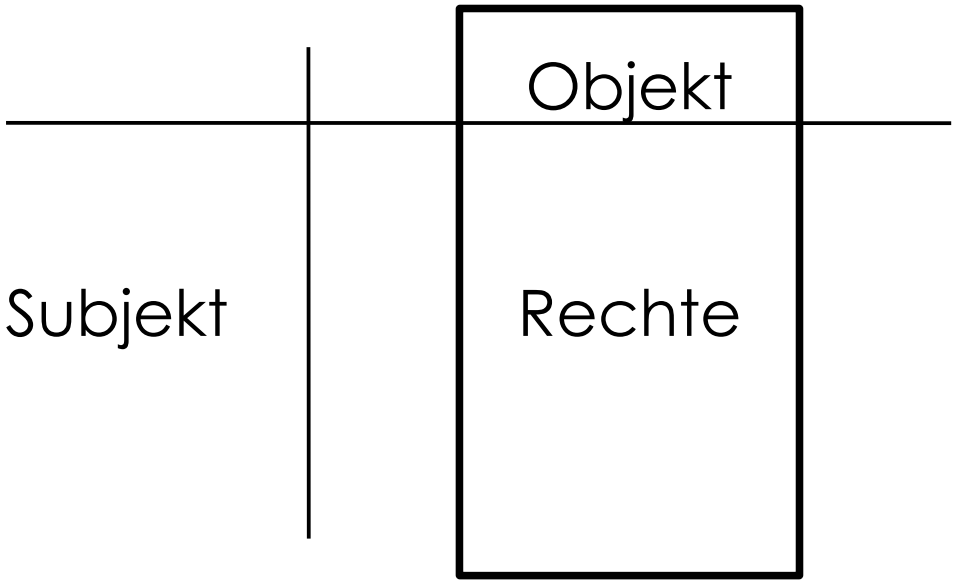
**Regelbasiert: MAC – Mandatory Access Control**

- bei jedem Zugriff werden Regeln ausgewertet

# Schutzmatrix spaltenweise: ACL

**Spaltenweise Darstellung: Access Control Lists**

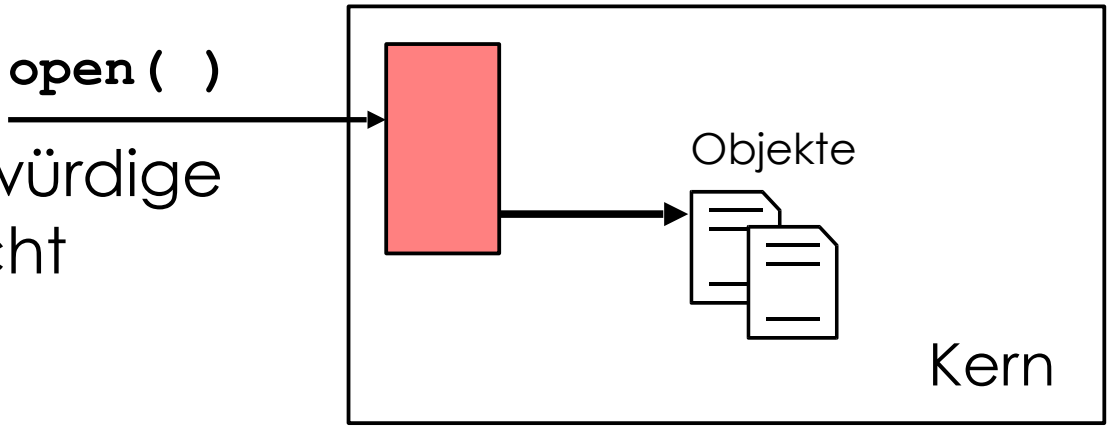
- Festlegung für jedes Objekt, was welches Subjekt damit tun darf
- basiert auf der Identität des Subjekts

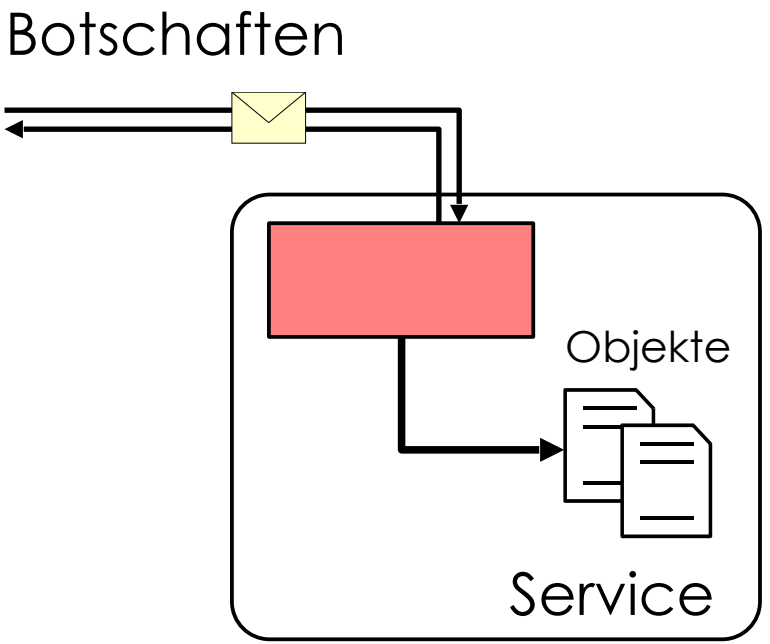


# Durchsetzung von ACLs

**Schutz der ACLs**

1. durch eine vertrauenswürdige Einheit, an der man nicht vorbeigehen kann:
   - Unix: Kern
   - Windows: Security Manager
2. durch Dienste, die Objekte implementieren;
   dabei muss Integrität der Botschaften gewährleistet sein



# Access Control Lists

Setzen der ACLs darf,

- Erzeuger eines Objekts
- wer einen ACL Eintrag für dieses Recht hat

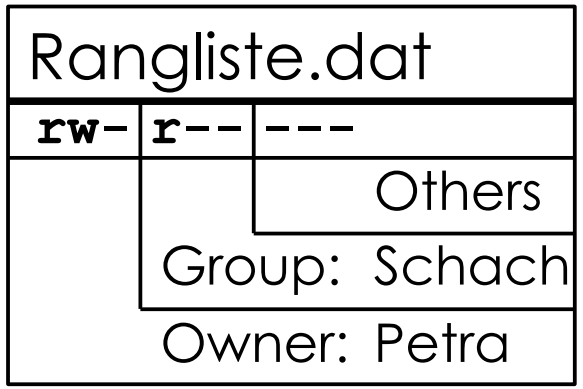
**Windows**

- Objekt: allow, deny
  full control, modify, read&execute, …
- „whitelist" von Programmen, die zugreifen dürfen

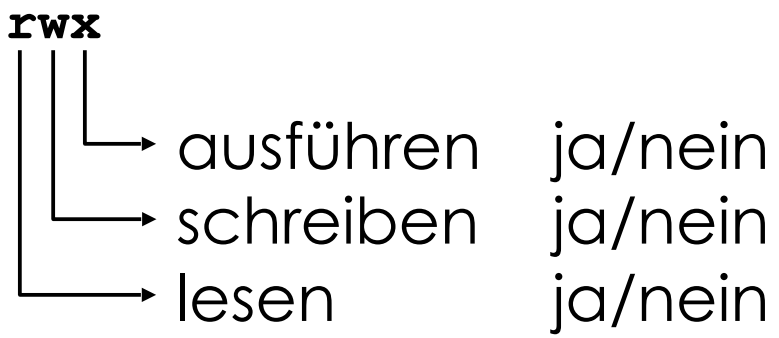
# ACLs in Unix

Unix: rudimentäre Zugriffssteuerlisten

- Prozess: UserID, GroupID
- Datei: Owner, Group
- Rechte werden festgelegt in Bezug auf:
  Owner, Group, Others (= alle anderen)




Dateiattribute:



# Benutzer und Prozesse

Prozesse repräsentieren Benutzer im System

- jeder Prozess hat Attribute UserID, GroupID
- IDs werden vom Eltern-Prozess übernommen
- Spezialfall: Login-Dienst

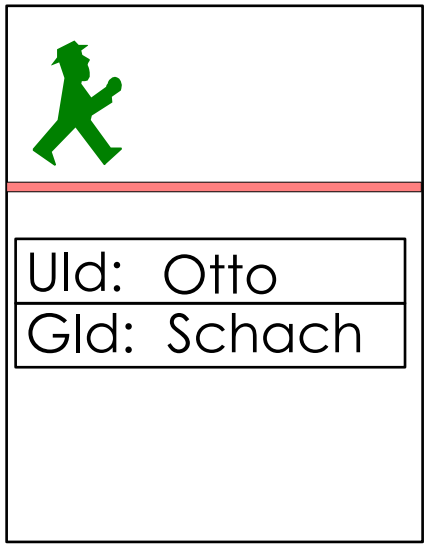

Beim Zugriff auf eine Datei:

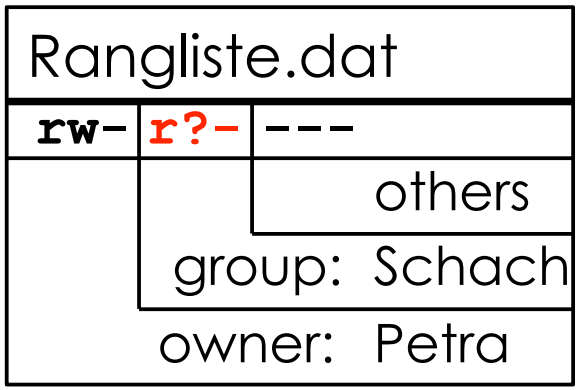
- stimmt UserID mit Datei-Owner überein?
  → dann gelten Owner-Rechte
- ist Prozess Mitglied in der Gruppe der Datei?
  → dann gelten Gruppen-Rechte
- sonst gelten Other-Rechte

# Problem: Rechteerweiterung

**Beispiel: Schachrangliste**

- jeder Spieler soll lesen können
- jeder Spieler soll seine Ergebnisse schreiben können
- kein Teilnehmer soll darüber hinaus schreiben dürfen (Fälschung der Rangliste)

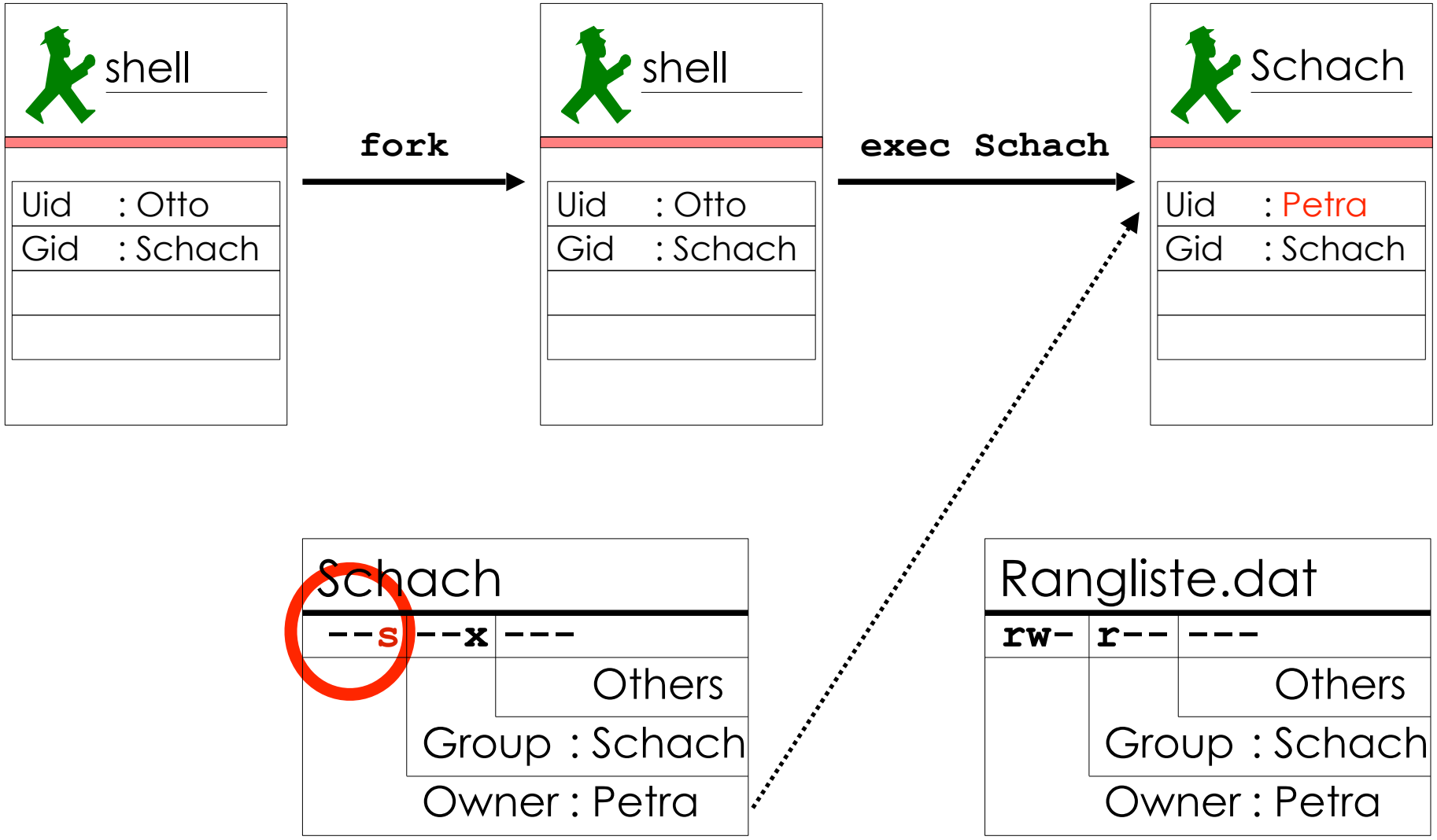


# Unix-Lösung: Set-UID-Mechanismus

- Zugriffsrechte abhängig vom dem Programm machen, mit dem man auf Datei zugreift
- Datei, die vertrauenswürdigen Programmcode (z. B. Schach) enthält, besitzt Kennzeichnung als „Set-UID“
- Bei **`exec`** mit einer Set-UID-Programmdatei erhält der ausführende Prozess die UID des Eigentümers (Owner) der Programmdatei

# Set-UID am Beispiel Rangliste



# Set-UID

Erweiterung der Rechte eines Benutzers genau
für den Fall der Benutzung dieses Programms.

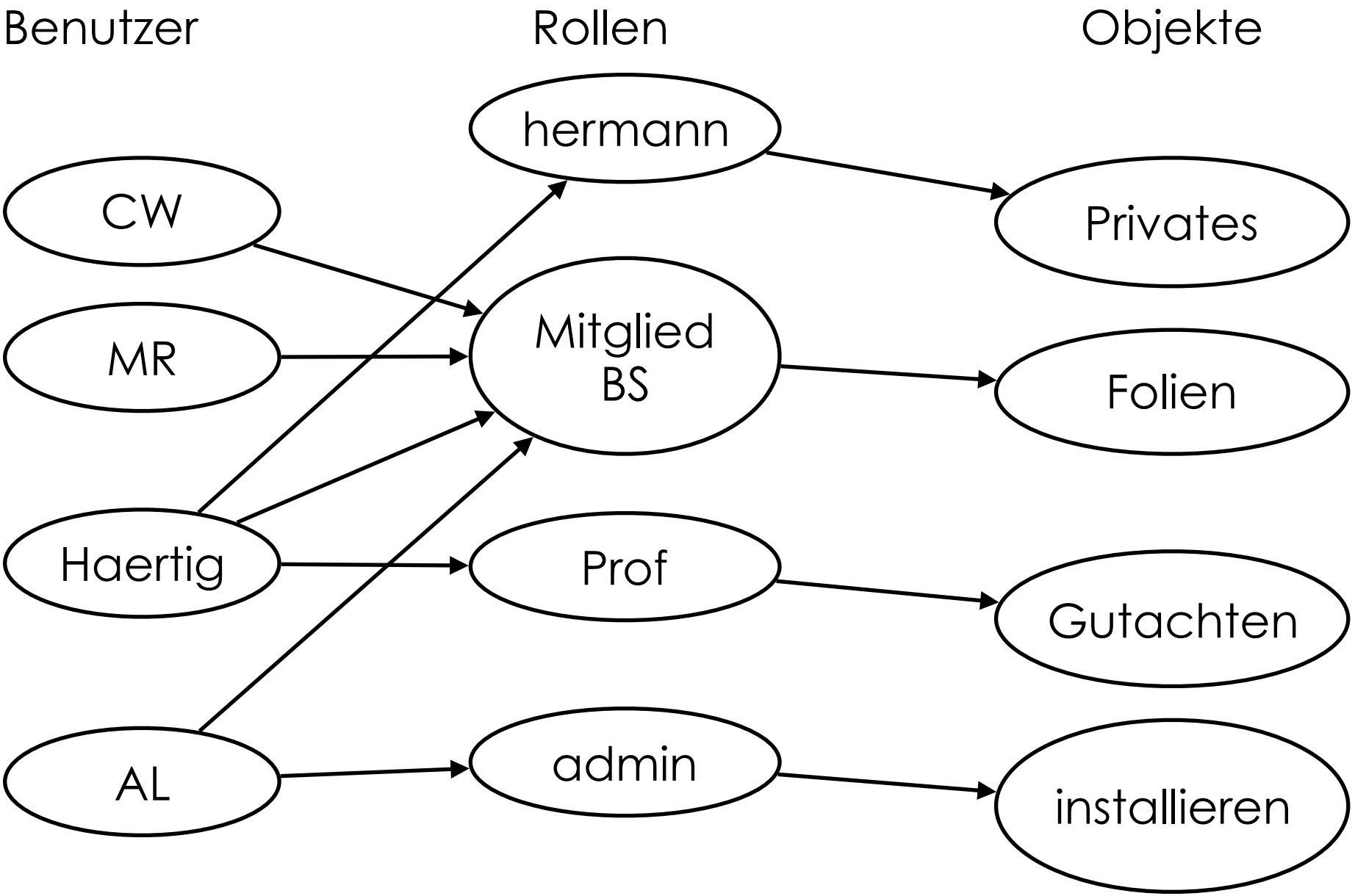
**Probleme**

- Set-UID vergibt in der Praxis zu viele Rechte
  (Set-UID auf root-Benutzer)
- Programmfehler führen zu unerwünschter
  Rechteerweiterung
  - Start anderer Programme aus einem Set-UID-Programm
  - Überschreiben von Dateien

# Limitationen von ACLs

- identitätsbasierte Rechtevergabe
  → Erkennen und Einordnen von Identitäten
- häufige Praxis: Zusammenfassen von Identitäten
  (zum Beispiel zu Benutzern und Gruppen wie in Unix)
  → zu weitreichende Rechte
- alle von einem Nutzer gestarteten Programme haben alle dazugehörigen Rechte
  → viele Möglichkeiten für Schadsoftware

# Abhilfe 1: Role Based Access Control



# Abhilfe 2: viele UIDs

**App-“Sandbox” in Android (Versionen vor 4.3)**

- erzeuge eine neue Benutzerkennung pro App
- füge für neuen Benutzer genau die ACLs ein, die für die Funktion des Programms nötig sind
- starte das Programm mit der Nutzerkennung des neuen Nutzers

# Abhilfe 3: Code-Identifikatoren

**Identifikation von konkreten Programmen als Subjekte**

- Programm-Name offensichtlich ungeeignet: Fälschungen
- sicherer (kryptografischer) Hash der Programmdatei: nicht robust gegenüber Updates
- Lösung: Code-Signaturen

**Ergänzung in ACL**

- Identifikator der Programmdatei als Subjekt
- bei Zugriffen muss sicherer Rückschluss von Prozess zu Identifikator möglich sein

# Schutzmatrix zeilenweise: Capabilities

**Zeilenweise Darstellung: Capabilities**

- Festlegung für jedes Subjekt, wie es auf welche Objekte zugreifen darf
- Subjekte können Rechte gezielt weitergeben
- Erzeuger eines Objekts hat zunächst alle Rechte und vergibt diese an andere Subjekte

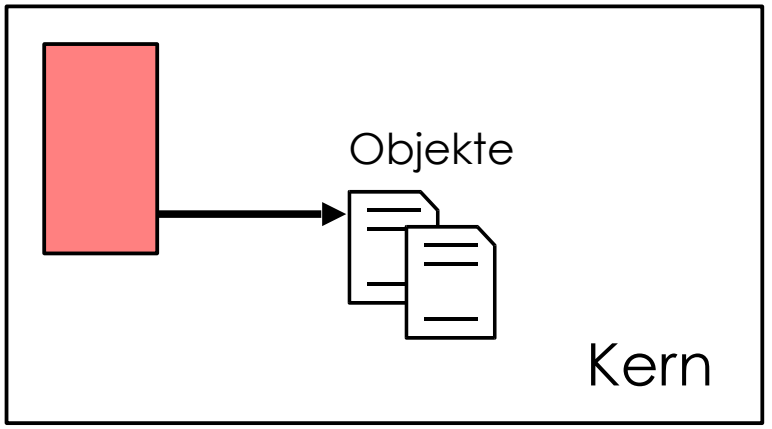
# Durchsetzung von Capabilities

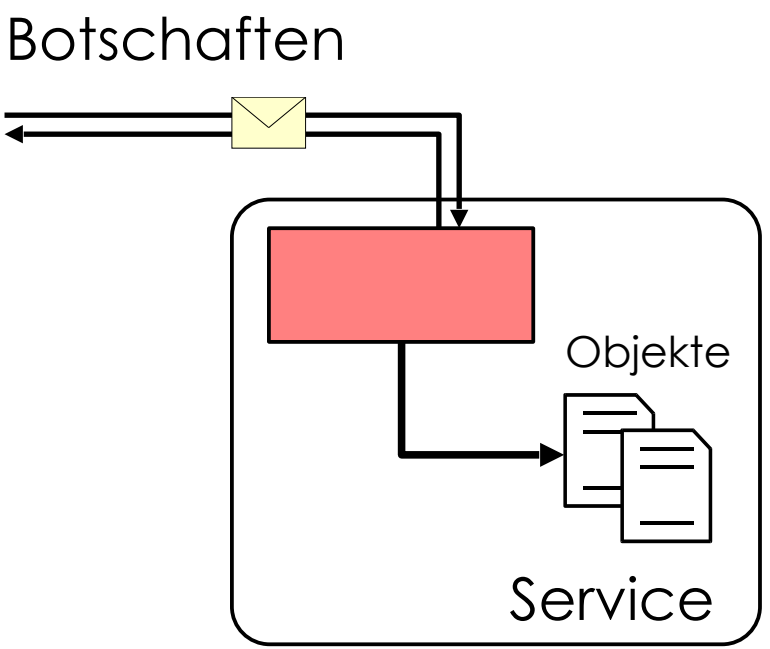
**Schutz der Capabilities**

1. durch eine vertrauenswürdige Einheit, an der man nicht vorbeigehen kann
2. durch Dienste, die Objekte implementieren und kryptografische Verfahren:

   Vergabe via “Tickets” in Botschaften

   - Kerberos
   - OAuth

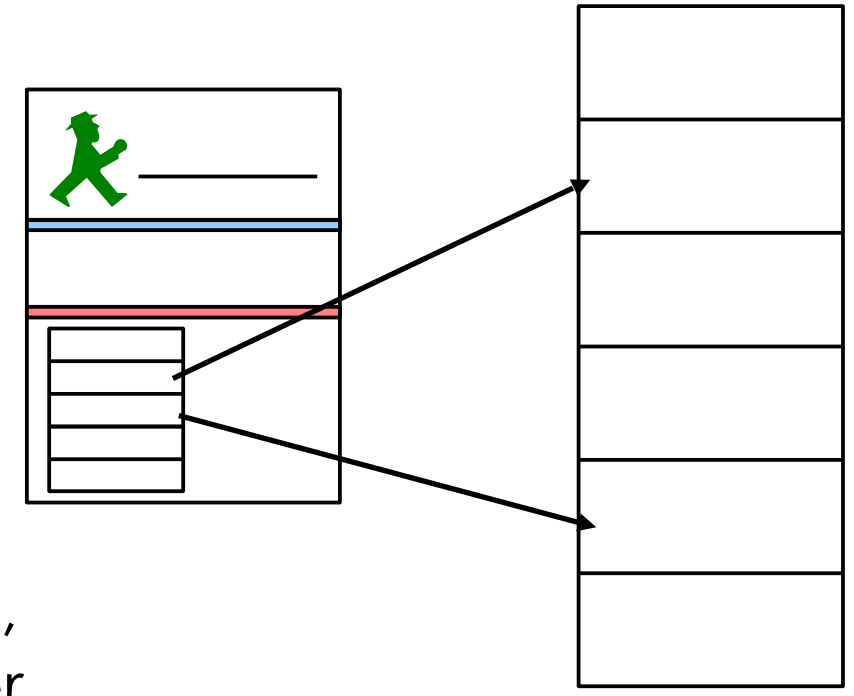


# Capabilities

**Prozesse**

- Tabelle mit Capabilities durch Kern geschützt
- Zugriffs auf Objekte durch Index in Capability-Array

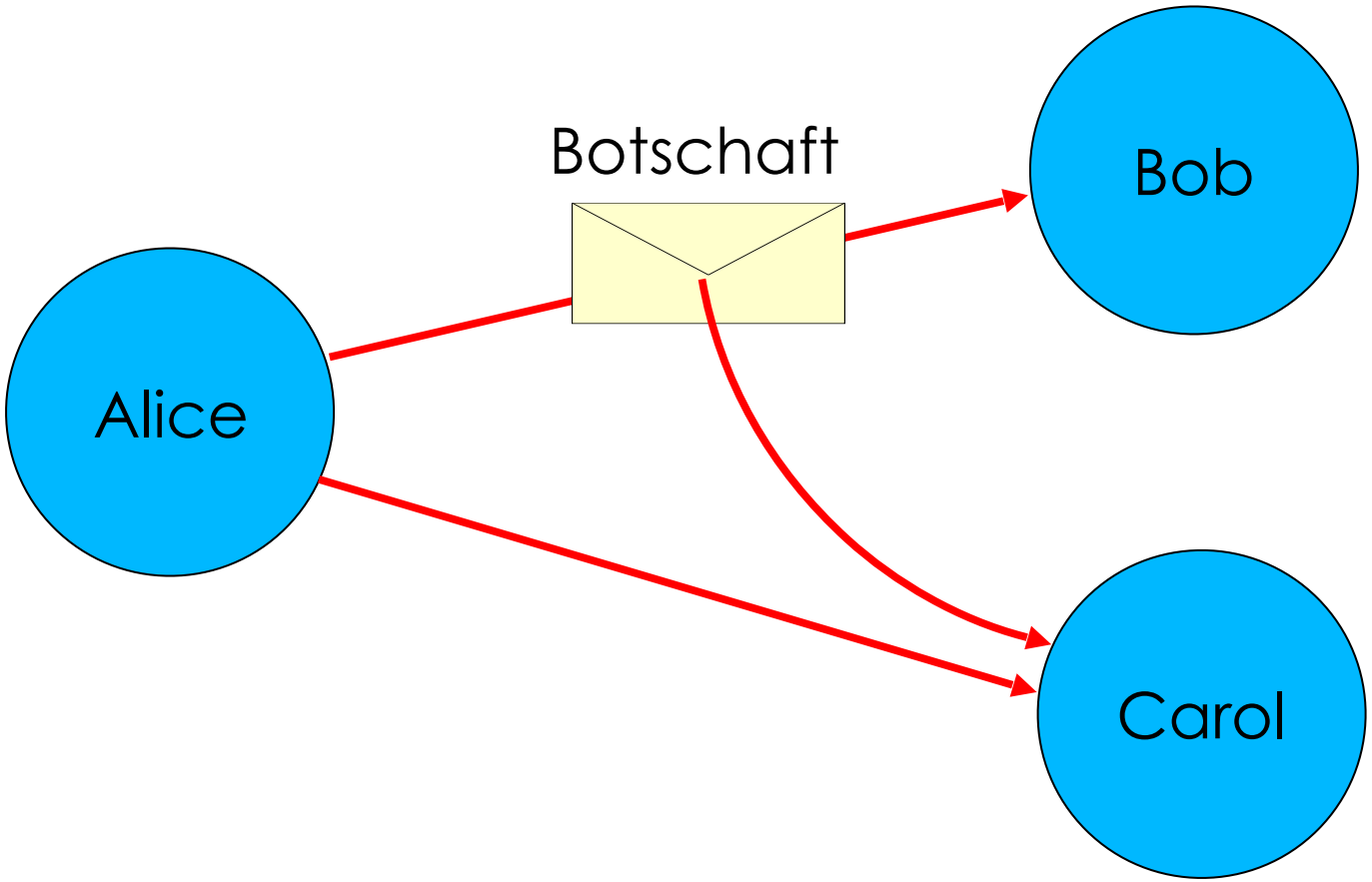


**Capability**

- “Pointer” auf Objekt
- Rechte
- Beispiel: Unix-Dateideskriptoren, Weitergabe durch **`fork()`** oder über Prozesskommunikation (Sockets)

# Rechte-Änderungen bei Capabilities

**Weitergabe von Capabilities durch Botschaften**



# Rechteerweiterung durch Powerbox

- Beispiel: Foto-Programm hat Zugriff auf die Foto-Bibliothek
- aber zur Laufzeit wird Zugriff auf weitere Fotos benötigt (zum Beispiel für Import neuer Bilder)
- Lösung: Anforderung über Zwischenprozess („Powerbox“)
- Öffnen-Dialog wird nicht vom Prozess des Foto-Programms bereitgestellt, sondern von einem Systemdienst mit Zugriff auf alle Dateien
- nur die vom Nutzer explizit ausgewählten Dateien werden dem Foto-Programm zur Verfügung gestellt
- in Unix zum Beispiel durch Übertragen der Dateideskriptoren

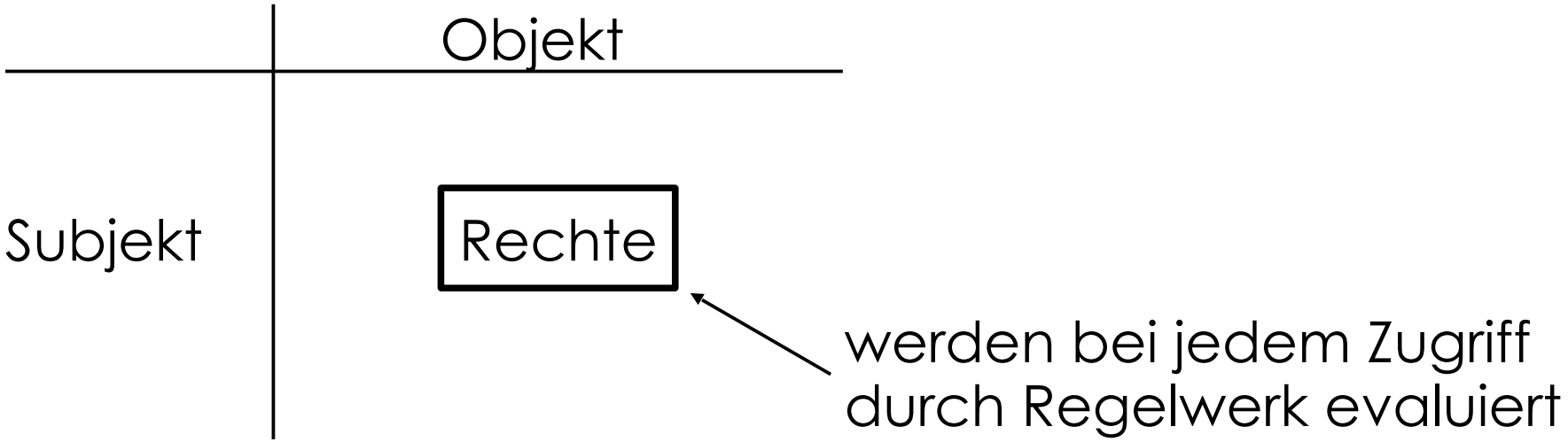
# Implementierungen von Capability-Systemen

- Unix File-Descriptors (rudimentär):
  langfristige Rechtsvergabe: ACL, kurzfristig: Capabilities
- Mikrokerne: KeyKOS, L4-Mikrokern
  (fortgeschrittene Vorlesung im Master-Studium)
- Tagged Architectures (Hardware):
  „fat pointer“ mit Rechten, Längenbeschränkungen
  (Burroughs 5500 ca 1970, Intel 432, CHERI-Projekt)

# Schutzmatrix regelbasiert

**Mandatory Access Control (regelbasierte Zugriffssteuerung)**

- Subjekte und Objekte haben Attribute („labels“)
- Entscheidung über Zugriff anhand von Regeln



# Beispiel „Multilevel Security“ (Militär)

**Einstufung von Personen und Dokumenten**

- in eine Sicherheitsebene
  z. B. normal, vertraulich, geheim, streng geheim
- in eine oder mehrere Kategorien
  z.B. Nato, Atom, Crypto

**Begriffe**

- Personen: clearance
- Dokumente: classification

# Multilevel Security

Vorschrift 1: Sicherheitsebene (X,Y)

- Sicherheitsebenen sind geordnet
- X mindestens auf gleicher Sicherheitsebene wie Y

Vorschrift 2: Kategorien (X,Y)

- Kategorien sind unabhängig voneinander
- X muss alle Kategorien haben, die auch Y hat

Kombination von Sicherheitsebene und Kategorie:

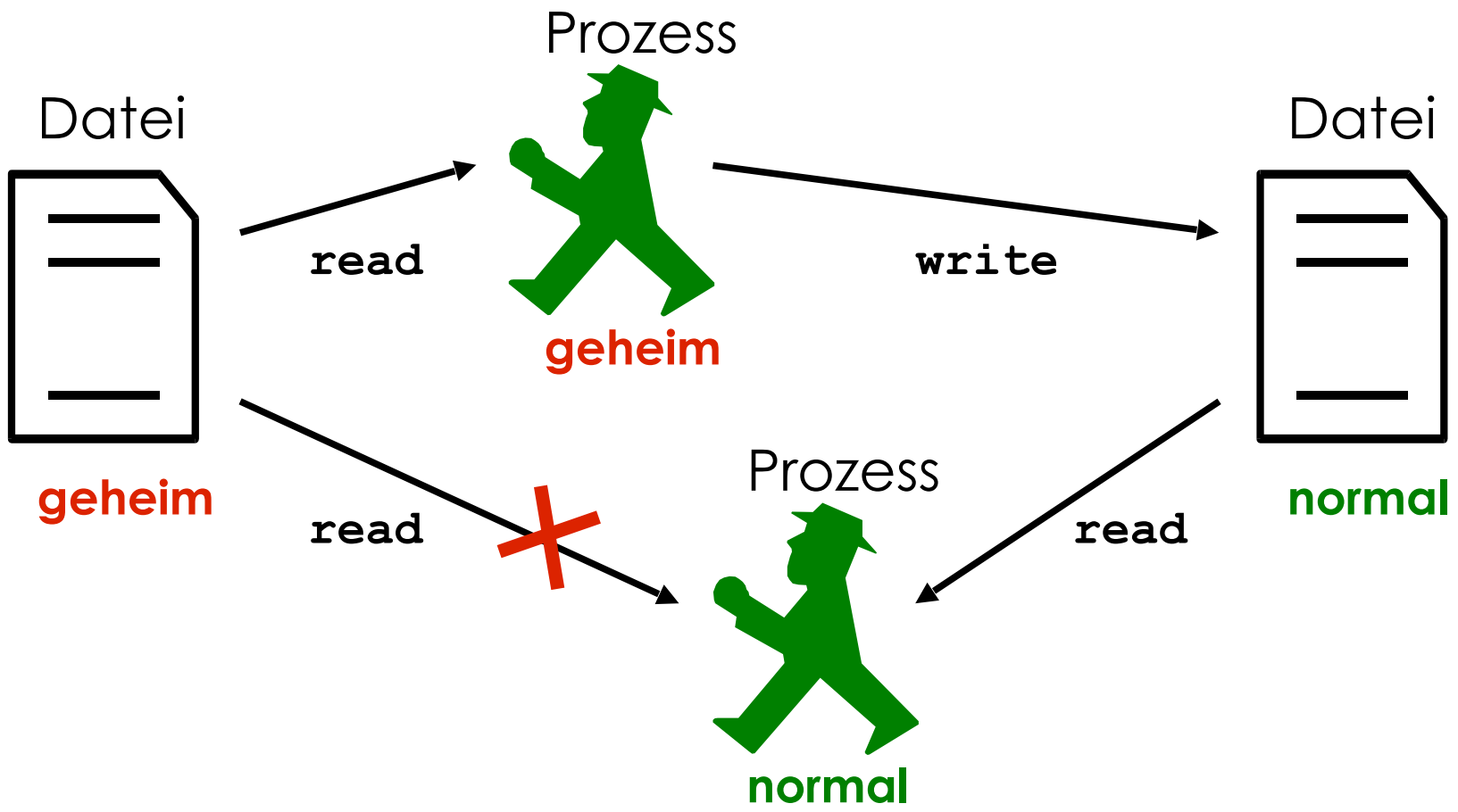
- Vorschrift 1(X,Y) und Vorschrift 2(X,Y) → X „dominiert“ Y

# Regeln der Multilevel Security

**Regel 1: „Simple Security“**

Ein Subjekt darf lesend auf ein Objekt nur dann zugreifen, wenn die Einstufung des Subjekts die des Objekts dominiert.

# Problem




als geheim eingestufter Prozess kann Informationen an Datei weitergeben, die als normal eingestuft ist

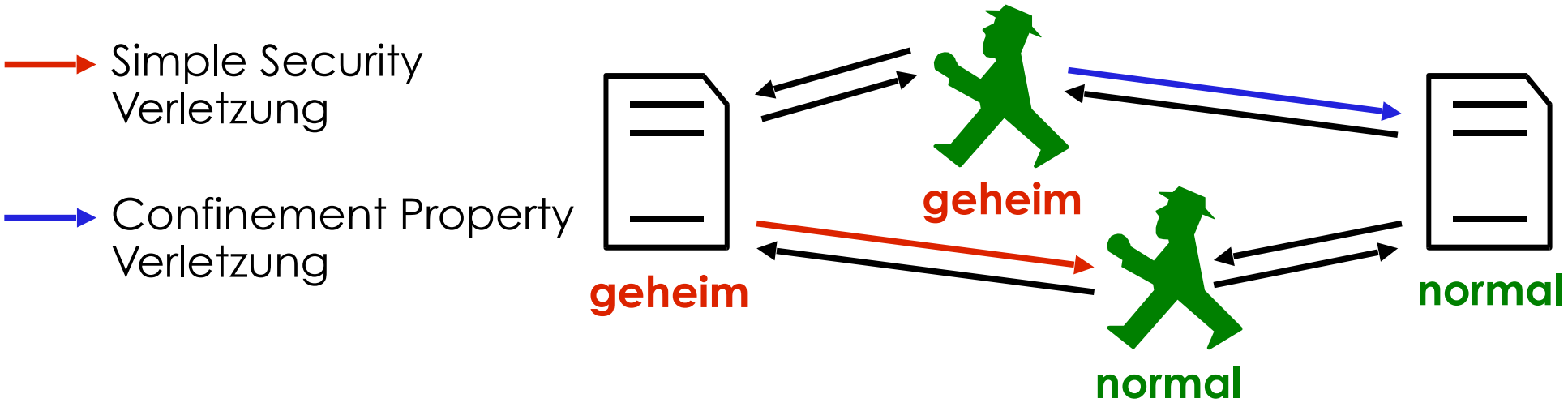
# Regeln der Multilevel Security

**Regel 1: „Simple Security“**

Ein Subjekt darf lesend auf ein Objekt nur dann zugreifen, wenn die Einstufung des Subjekts die des Objekts dominiert.

**Regel 2: „*-Property, Confinement Property“**

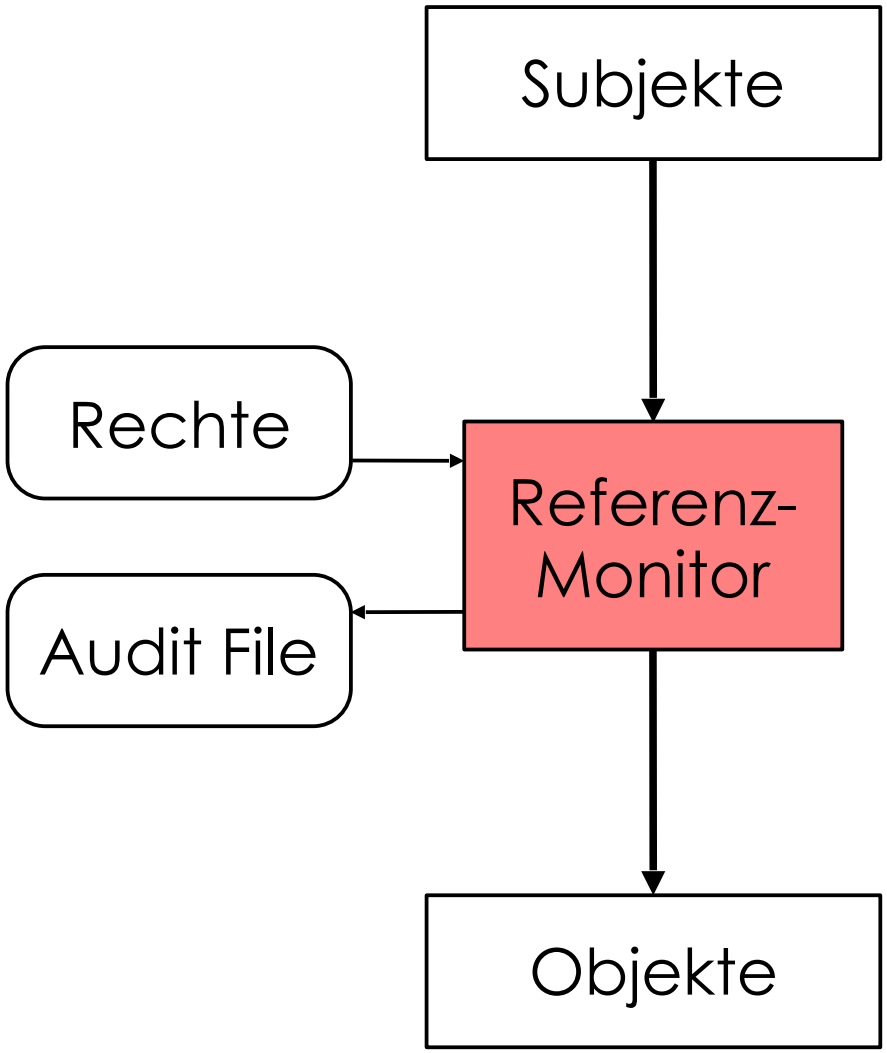
Ein Subjekt darf schreibend auf ein Objekt nur dann zugreifen, wenn die Einstufung des Subjekts von der des Objekts dominiert wird.



# Durchsetzung: Referenzmonitor

**Prinzipien**

- Vollständigkeit:
  kein Subjekt darf auf Objekt unter Umgehung des Referenzmonitors zugreifen
- Isolation:
  keine Modifikation des Referenzmonitors
- Verifizierbarkeit:
  Code-Inspektion, Test, formale Beweise



# Moderne MAC-Implementierungen

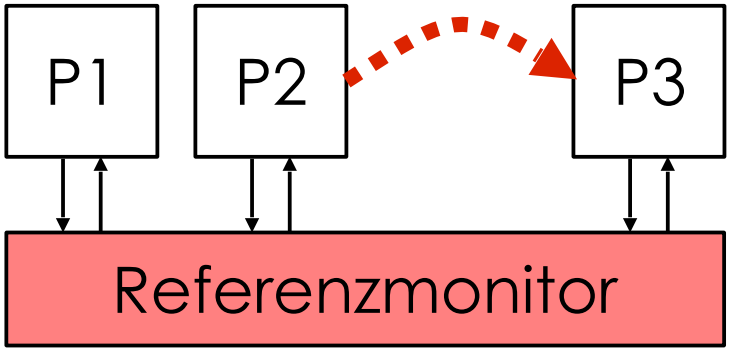
- SELinux, AppArmor, SecComp-Filter
  - regelbasiertes Filtern von Systemaufrufen
- TrustedBSD, Apple Seatbelt
  - Grundlage für die **App Sandbox** in iOS und OS X
- Windows 8 App Container

Kombination von **Entwicklerwissen** und **Nutzerinteressen:**

- Entwickler beschreibt über **Manifest,** was Prozess benötigt
- System setzt diese Beschränkungen durch
- auch bei einem Angriff werden nicht mehr Rechte erlangt
- Nutzer kann sensitive Resourcen (Ortung, Mikrofon, Kamera, …) durch **ACLs** verwalten

# Aber: Verdeckte Kanäle (Covert channels)

Informationsfluss an üblichen, durch Schutzmechanismen überwachten Schnittstellen vorbei




- Covert Storage Channel
  zum Beispiel über Existenz von Dateien (in **`/tmp`**)
- Covert Timing Channel
  zum Beispiel über Schwankungen der Rechenintensität eines Prozesses und Beobachtung durch einen anderen
- Energieverbrauch
  zum Beispiel zur Extraktion von Schlüsseln aus Smartcards

# Zusammenfassung

- Konstruktionsprinzipien für sichere Systeme
- Schutzmatrix
- ACLs, Unix-Rechteschema, Set-UID
- Capabilities, Powerbox
- Regelbasierte Zugriffssteuerung, Sandboxing
- Verdeckte Kanäle