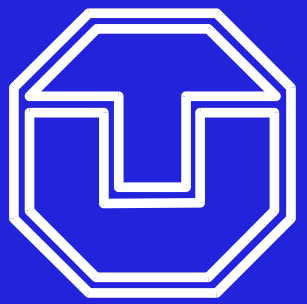
Sicherheit @ Systemarchitektur

Betriebssysteme, WS 2016/17

Hermann Härtig
TU Dresden



Angriffe und Aufwand

- Effektiver Schutz erfordert klare Vorstellung über Angriffe und deren Aufwand, die man abwehren können möchte.
- Kosten und Nutzen für Schutzmaßnahmen müssen in sinnvoller Relation stehen.
- Schutzziele sind manchmal (zu Recht ?) anderen Erwägungen untergeordnet.

“It is meaningless to achieve perfection” (Butler Lampson, SOSP 2015)

Lernziele

- Prinzipien der Konstruktion sicherer Systeme
- Integration (Platzierung)
  von Sicherheitsmechanismen in Systemarchitektur
- Access Control Lists ./. Capabilities
- Regelbasierte Sicherheitspolitik (mandatory security policy)
- Verdeckte Kanäle (covert channels)

Konstruktion sicherer Systeme

Entwurfsprinzipien – SALTZER und SCHRÖDER (schon 1975!)

Prinzip der geringst-möglichen Privilegierung
nur die Rechte einräumen, die für die zu erbringende Funktionalität erforderlich sind

- Verbot als Normalfall
- Whitelisting vs. Blacklisting
- "Brutales" Gegenbeispiel: Unix „root" & Unix SetUID

Sichere Standardeinstellungen („fail-safe defaults")

Separierung von Privilegien (Separation of duty)

- mehrfache Bedingungen für die Zulassung einer Op.

Konstruktion sicherer Systeme

So einfach wie möglich

- reduziert Fehlermöglichkeiten bei Implementierung und Einsatz

Offenheit

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

Psychologisch akzeptabel

Konstruktion sicherer Systeme

Vollständige Prüfungen (complete mediation):

- jeder Aktion, die potentiell ein Schutzziel verletzt, sollte kontrolliert werden
- Gegenbeispiel: Zugriff auf offene Unix Dateien

Aktualität von Prüfungen

TOCTTOU: time of check to time of use

Systemarchitektur - Basiselemente

- Isolation
- Kommunikation
- Access Control
- “Policy”

- Intrusion Detection

Sicherheit und Betriebssysteme

Benutzer:

- erzeugen Prozesse und lassen sie für sich arbeiten
- speichern ihre längerfristigen Daten in Dateien/ Speicherobjekten

Administratoren/Benutzer/System-Software:

- steuern die Zugriffsrechte auf Objekte

Sicherheit und Systemebenen

Umgebung:	physische Sicherung
HW:	User/Kernel-Modi Adressraumseparierung Sicherer Neustart (secure booting)
BS-Kern:	Kapselung von Prozessen (Isolation)
Kern oder Server:	Virtuelle Maschinen Zugriffssteuerung
Serverprozesse:	Authentifikation Zugriffssteuerung (im verteilten System)

Protokolle im verteilten System: …

Benutzer/Administration: Einsatz der Mechanismen

Isolation

Adressräume hardwarebasiert:

- Kernel/User-Mode und MMU
- Beispiele:
  Windows, Linux, L4, …

Sprachbasiert:

- alles in einer sicheren HLL codiert
- Beispiele:
  Burroughs 1700, 5500, (1975 ff)
  Microsoft Singularity
  Oberon

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

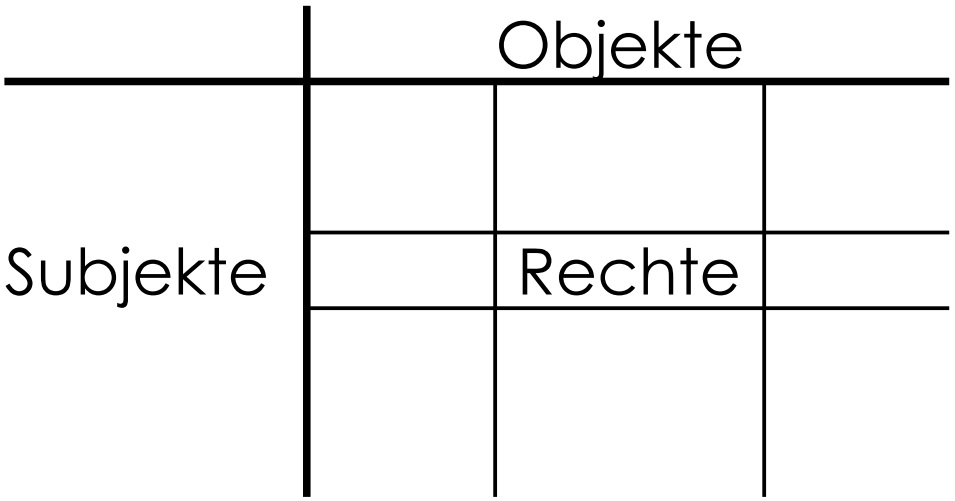
Objekte:

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

ACL – Access Control List (identitätsbasiert)

- bei jedem Zugriff wird beim Objekt auf der **Basis der Identität des Absenders** dessen Berechtigung geprüft
- "spaltenweise" Festlegung durch Besitzer der Objekte

Sandboxing, Capabilities

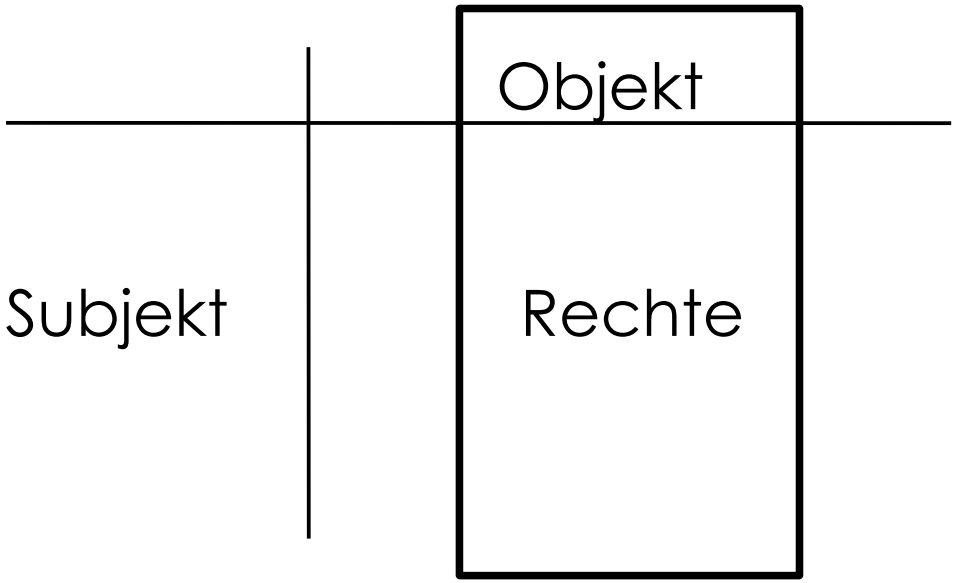
- bei jedem Zugriff wird etwas geprüft, was Subjekte "besitzen" und bei Bedarf weitergeben können
- Beschränkung durch Festlegungen seitens der Subjekte

Regelbasiert („mandatory access control"):

- bei jedem Zugriff werden Regeln ausgewertet

Schutzmatrix spaltenweise: ACL

- Festlegung für jedes Objekt,
  „was welches Subjekt damit tun darf “
- Basiert auf **Identität** des Subjekts!



Einfaches Modell: Datei- und Prozess-Attribute

- Festlegungen in Bezug auf Benutzer:
  - für welchen Benutzer arbeitet ein Prozess
  - welchem Benutzer gehört eine Datei (owner)
  - welche Rechte räumt ein Benutzer anderen und sich selbst an „seiner“ Datei ein
    - → Rechte eines Prozesses an einer Datei
  - Attribute von Prozessen: UserId
  - Attribute von Dateien: OwnerId

Schutzmatrix :

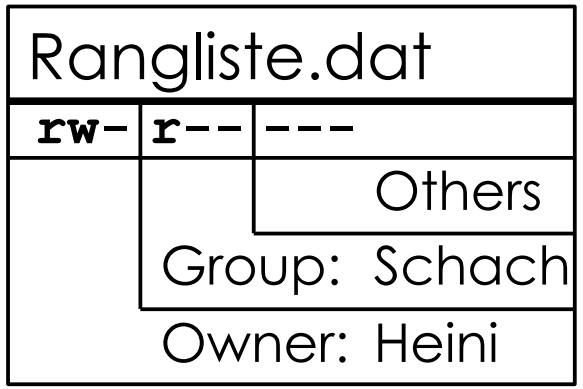
	File1	File2	File3	File4	File5	File6	File7	File8
User1								
User2								
User3			read					
User4								

ACLs in Unix

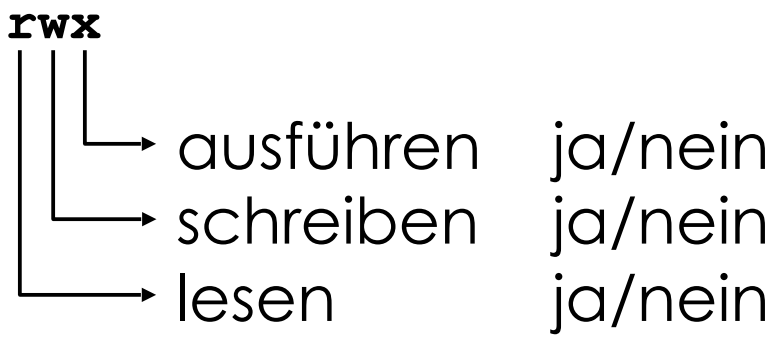
Unix-Original: rudimentäre Zugriffssteuerlisten

- Prozess: UserId, GroupId
- Datei: Owner, Group

→ Rechte in Bezug auf Owner, Group, Others



Dateiattribute:



Rechte-Änderung per SETUID: Unix

- Programme haben einen **Owner** und
- erhalten Attribut **SETUID**
- ein Prozess erhält dann die Rechte des Owners
  nicht des Starters

Access Control Lists (Zugriffssteuerlisten)

Setzen der ACLs darf,

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

Multics

File 0 (Jens, *, RWX)
File 1 (Jens, system, RWX)
File 2 (Jens, *, RW-), (Else, staff, R --), (Meike, *, RW-)
File 3 (*, student, R--)
File 4 (Paul, *, ---), (*, student, R--)

Windows NT

Objekt: allow, deny
full control, modify, read&execute, ...

aus Tanenbaum

ACL - Limitationen

- identitätsbasierte Rechtevergabe
  -> zu große Rechte
  z.B. alle Rechte von user haertig für alle seine Apps

Schadsoftware !!

Abhilfe: siehe nächste Folien

Abhilfe 1

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

- erzeuge eine neue Benutzerkennung per 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

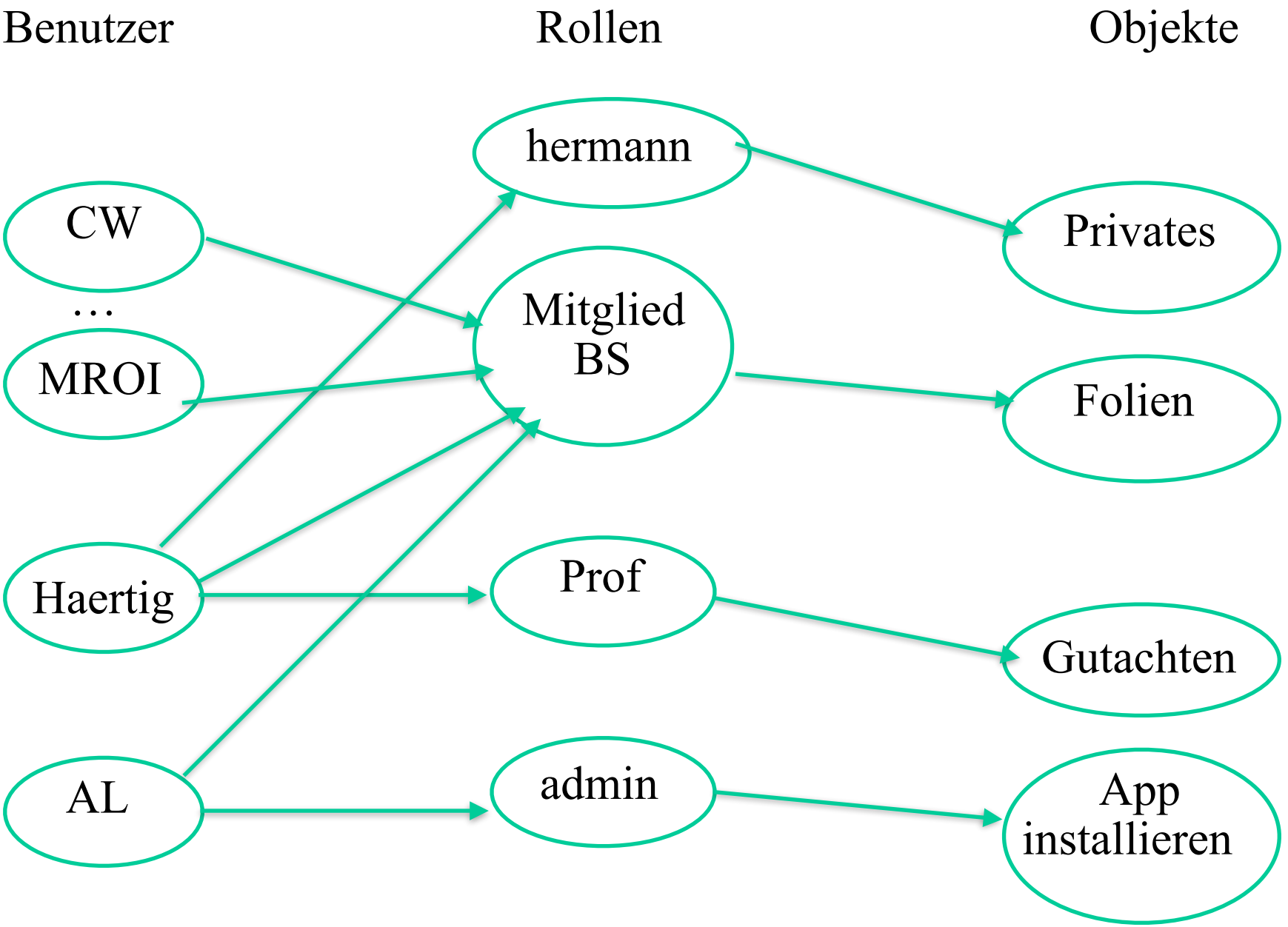
Abhilfe 2

Weitere Elemente in ACL

- Id/Hashcode des Programm-Codes
- ID eines Prozesses
- Zeiten

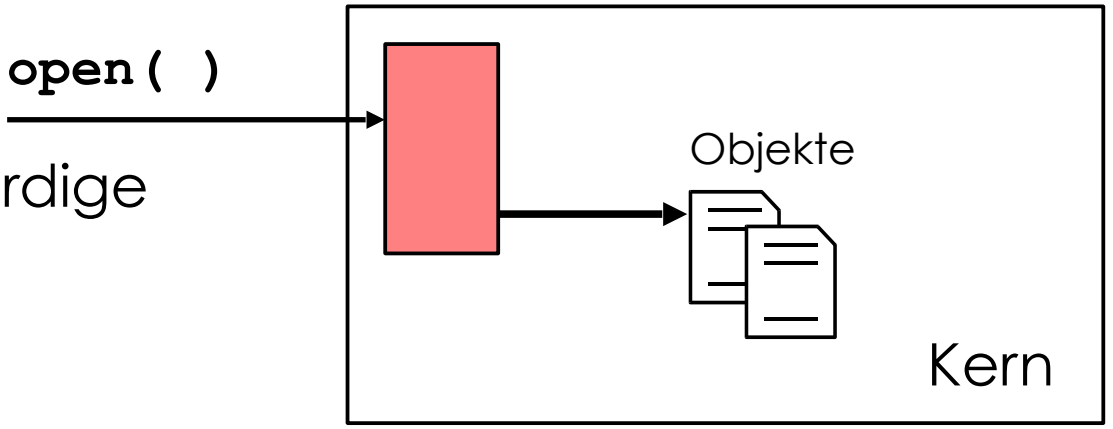
- keine praktische Bedeutung !!

Abhilfe 3: "Role Based" Access Control

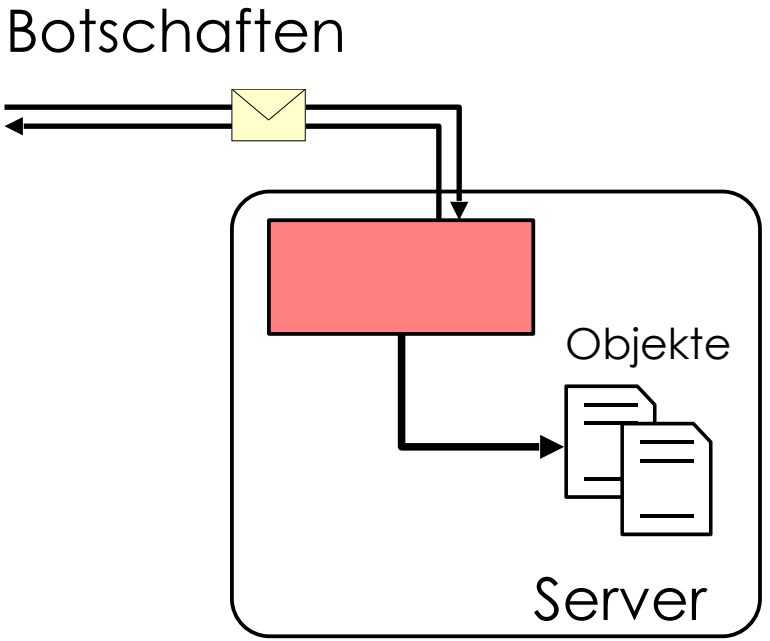


Durchsetzung von ACLs

Kapselung der ACLs



1. durch eine vertrauenswürdige Einheit, an der man nicht vorbeigehen kann:
   - Unix: der Kern
   - WinX: Security Manager



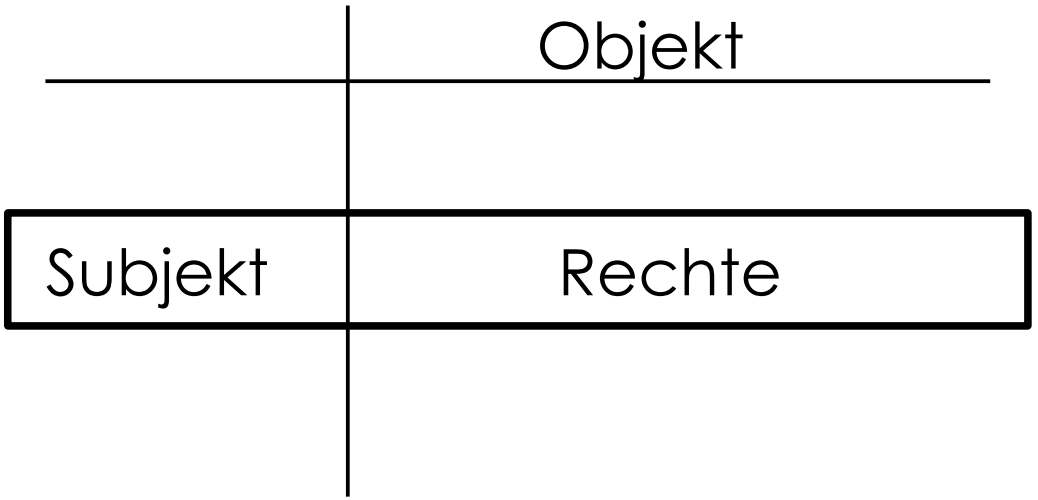
2. durch die Server, die Objekte implementieren
   dabei muss Integrität der Botschaften gewährleistet sein

Capabilities, Sandboxing

"**Zeilenweise"**

→ Festlegung für jedes Subjekt,
„wie es auf welche Objekte zugreifen darf"

- Subjekte können Rechte gezielt weitergeben
- Vergabe via "Tickets" in Botschaften
- Erzeuger eines Objekts hat alle Rechte,
  vergibt diese an andere "Subjekte"



Capabilities

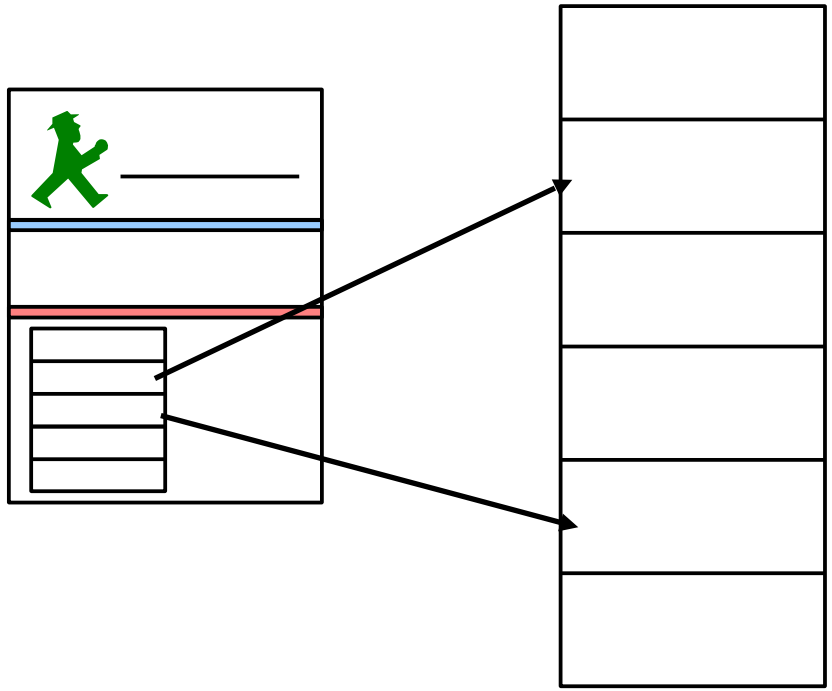
Prozesse:

- Tabelle mit Capabilities
- Zugriffs auf Objekte durch Index in array

Capability:

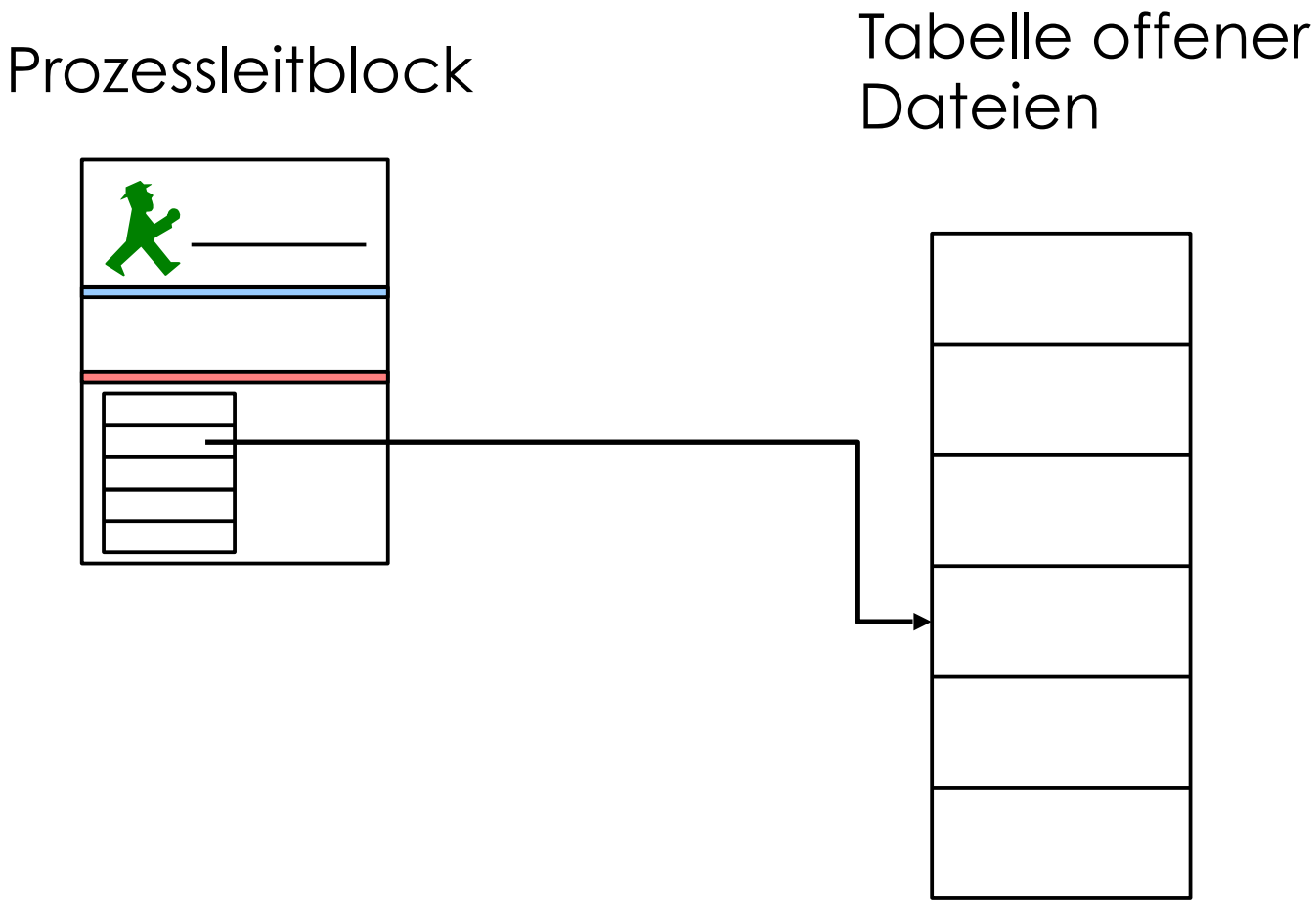
- Pointer auf Objekt
- Rechte
- “Tag”

Prozesse können capabilities weitergeben



Beipiel für rudimentäre Capabilities

- Rudimentäre Form: Unix Filedeskriptoren
- Weitergabe nur durch „FORK“ Operation

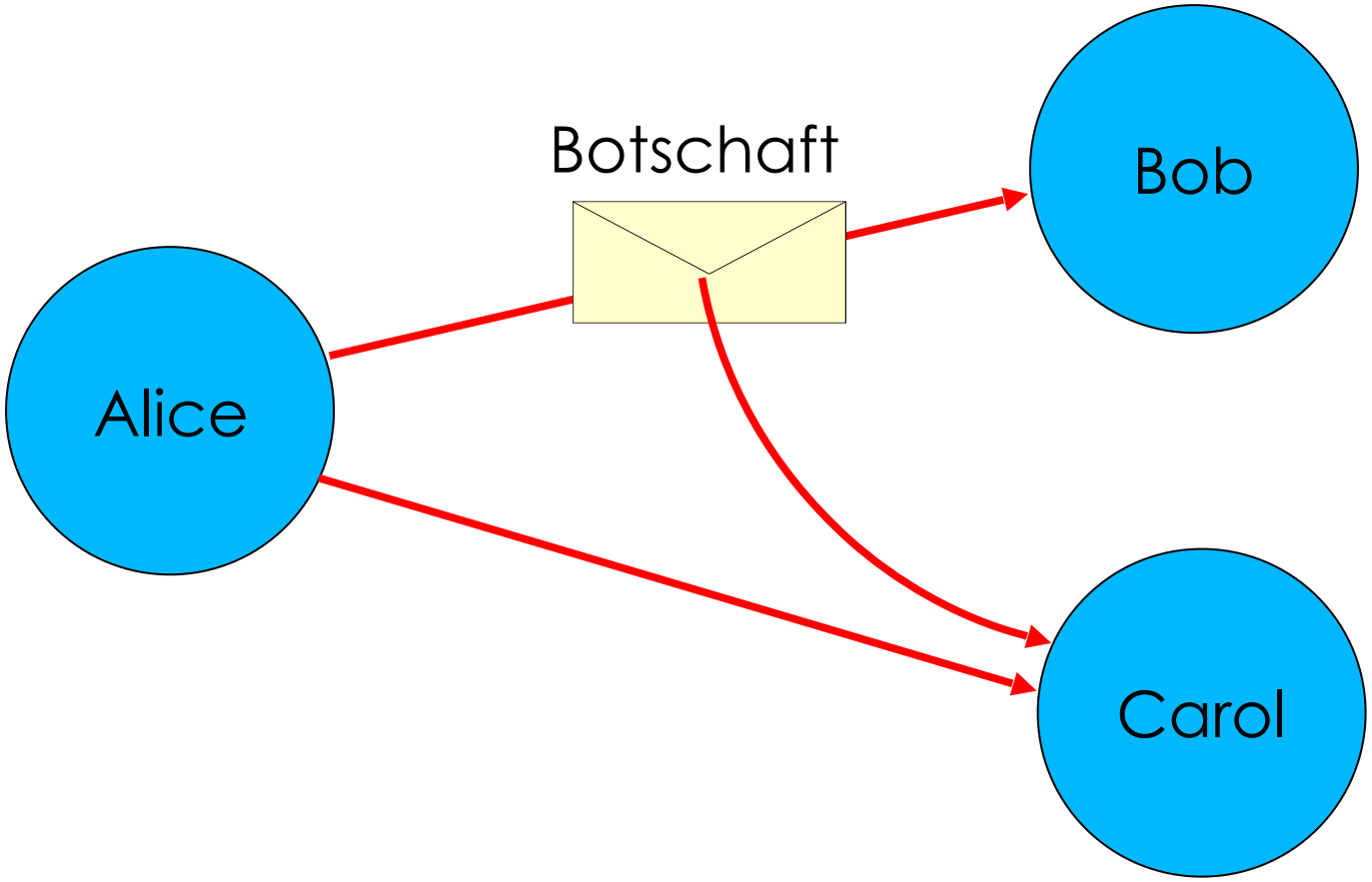


Rechte-Änderungen bei Capabilities

Weitergabe von Capabilities durch Botschaften

z.B.: Aufgabe von Namensdiensten

Granovetter-Diagramm nach Mark Miller:

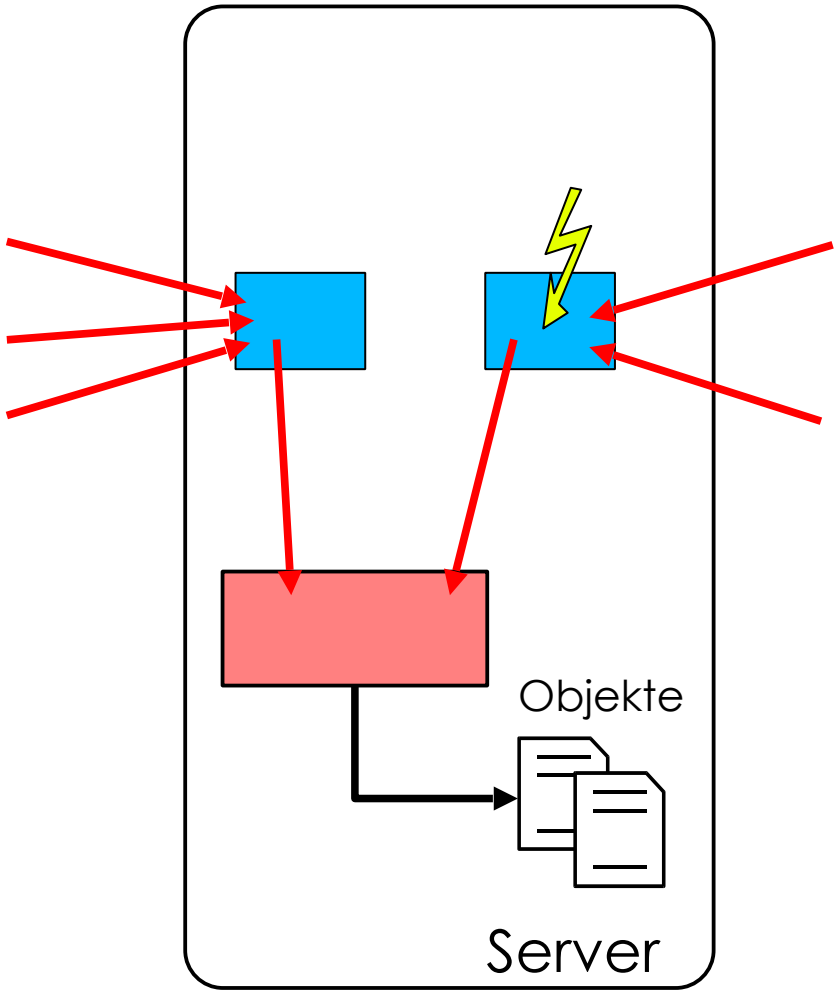


Entzug von Capabilities

- durch Buchführung
- durch Indirektion
  EROS, Keykos, Amoeba

viele weitere Fragestellungen zu Capabilities

→ Distributed OS (Sommersem.)

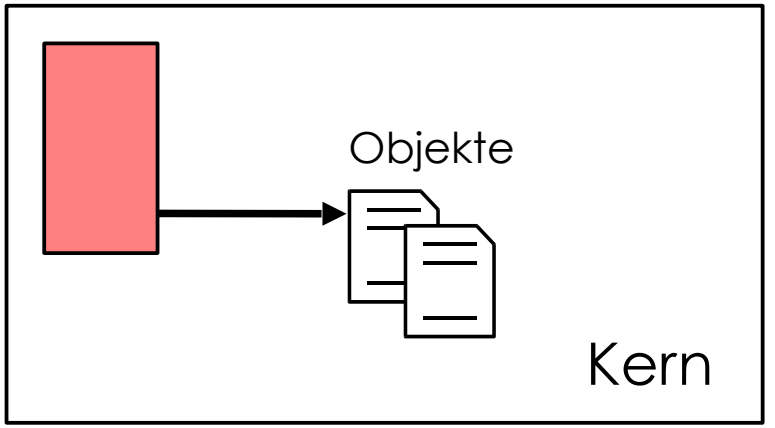


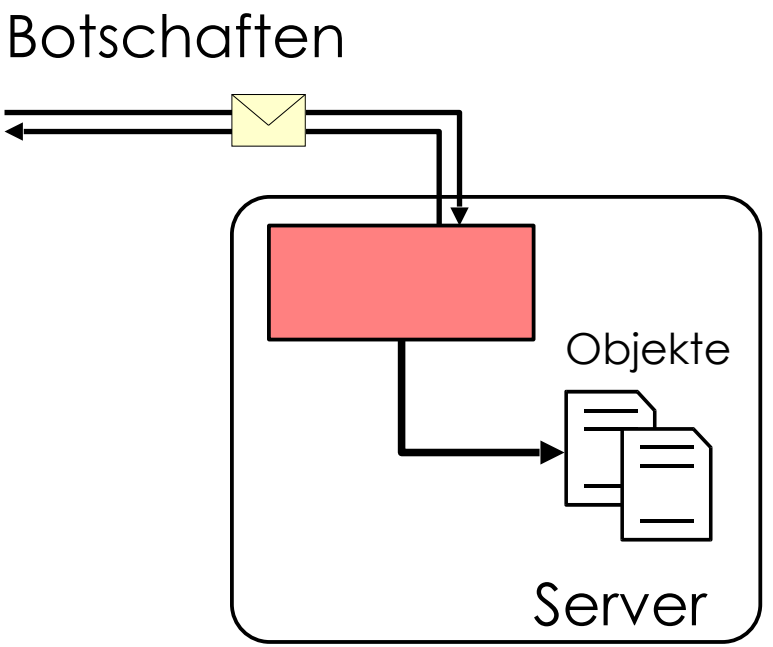
Durchsetzung von Capabilities

Schutz der Capabilities

1. durch EINE vertrauenswürdige Einheit, an der man nicht vorbeigehen kann:
   z.B. Betriebssystem-Kern

2. durch die Server,
   die Objekte implementieren,
   und kryptographische Verfahren
   (später dazu mehr)





Beipiele für Capability-Systeme

- Unix File-Descriptors (rudimentär):
  langfristige Rechtsvergabe: ACL, kurzfristig: Capabilities
- Android App Rechte (sehr elementar):
  z.B.: man kann bestimmte Rechte nicht entziehen
- Apple Sandboxes
- Mikrokerne: keykos,
  L4-Mikrokern (-> eigene Vorlesung im Hauptstudium)

- Hardware Tagged Architectures:
  ein Bit hinzufügen -> Interpretation als
  "FAT POINTER" mit Rechten, Längenbeschränkungen
  (Burroughs 5500 ca 1970, intel IA 432, Cheri-Cambridge)
  Forschungsprojekte

Beispiel: Schach Turnier

- Rangliste(Datei): /usr/henrike/spiele/schach/Rangliste
- Programm: /usr/henrike/bin/spiele/Schach
- alle Spieler benutzen ein Schachprogramm
- jeder Spieler soll seine Ergebnisse in die Rangliste eintragen können mittels dieses Schachprogramms
- Spieler sollen “Handycap” bekommen:
  je nach Erfahrung limitierte Denkzeit
  1: max 20 Minuten
  2: max 60 Minuten
  3: max 120 Minuten
  für ein ganzes Spiel

ACL

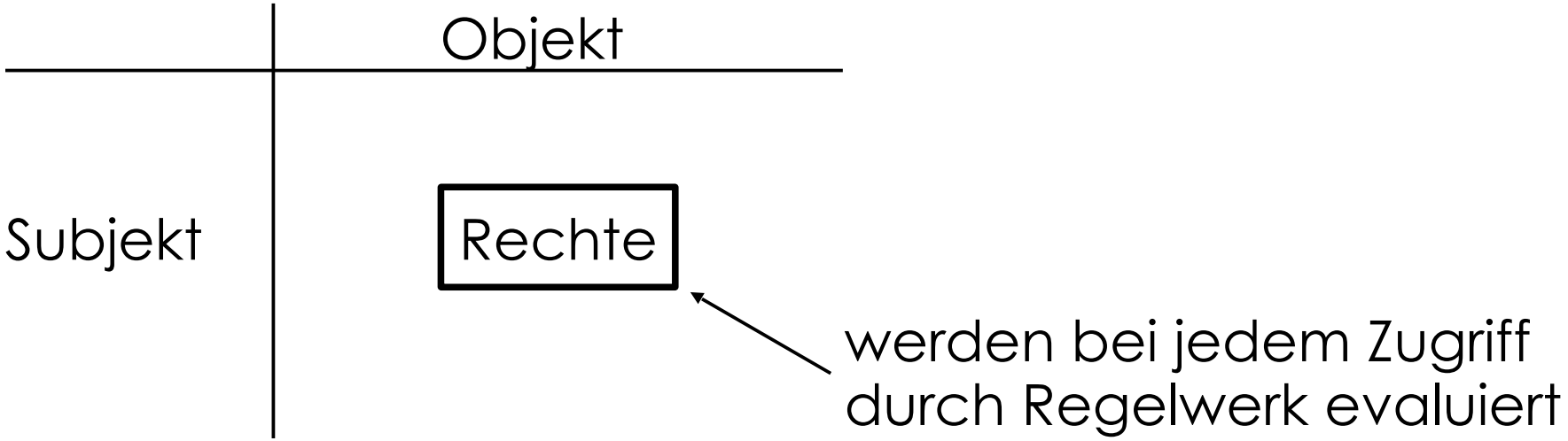
1. Erster Versuch:
   alle haben Schreibrecht → zu viele Rechte
   - (funktioniert nicht)
2. SetUID:
   - nur Henrike hat Schreibrecht;
   - Schachprogramm: „setuid“
     sobald ein Prozess das Schachprogramm aufruft,
     erhält es als UserId den Owner des Schachprogramms
   - Praktische Erfahrung: immer noch zu viele Rechte !!!

Schutzmatrix regelbasiert

Mandatory Access Control (Regelbasierte Zugriffssteuerung)

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

- Implementierung:
  sog. Sicherheitskerne



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}

→ z. B. (geheim; 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

Beispiel

Dokument (geheim; Nato, Atom)

Person 1: (geheim; Nato, Atom, Crypto)

Person 1 dominiert Dokument, da

geheim >= geheim

UND {Nato, Atom, Crypto} ⊃ {Nato, Atom}

Person 2: (streng geheim; Nato, Crypto)

Person 2 dominiert Dokument nicht, da

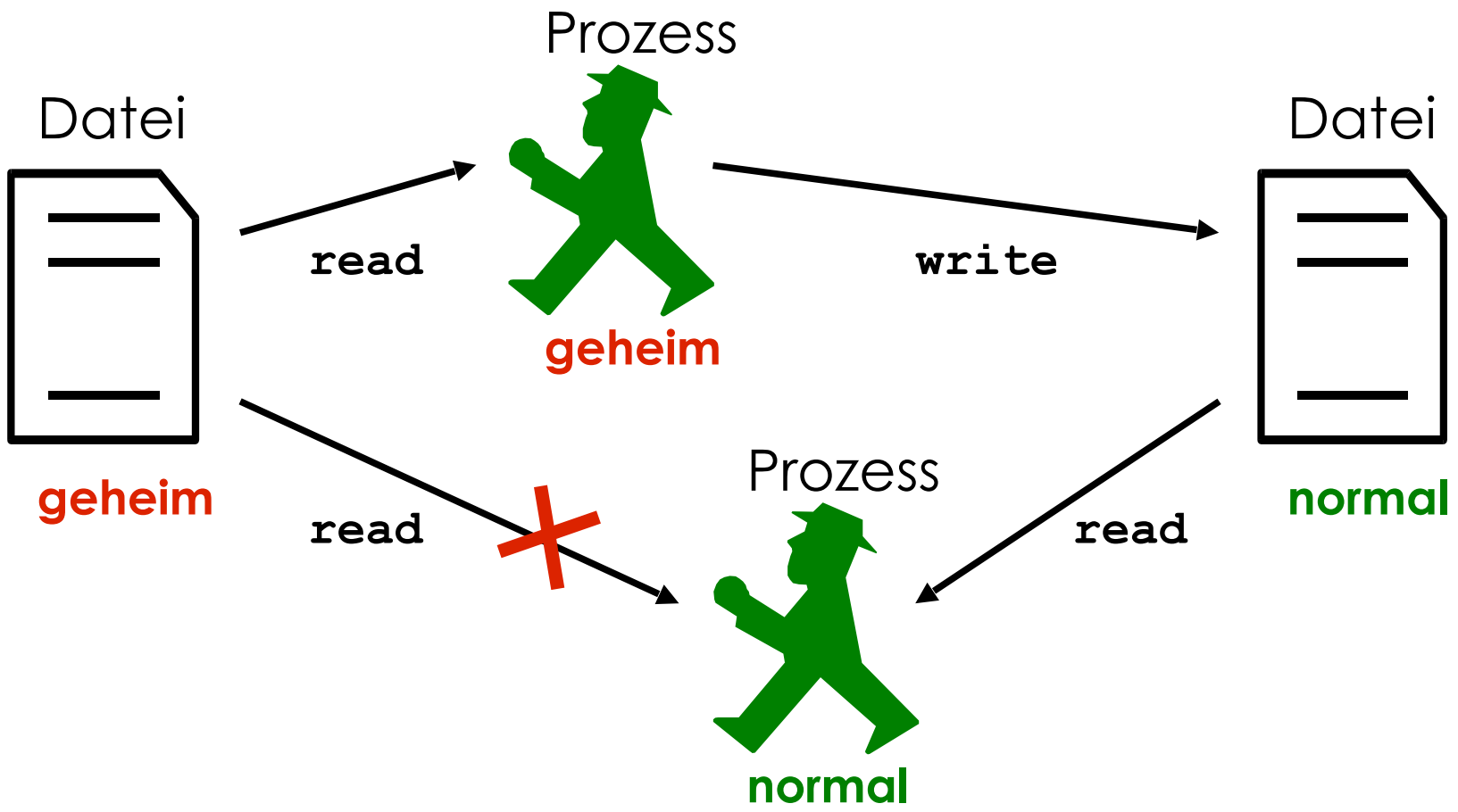
¬ ({Nato, Crypto} ⊃ {Nato, Atom})

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



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

→ zusätzliche Regel (*-property)

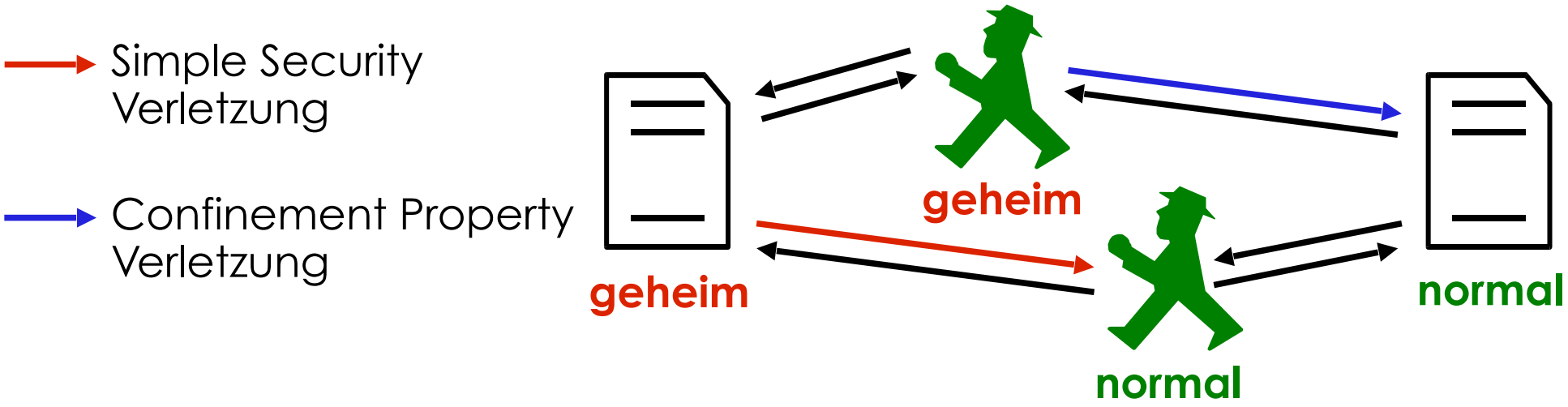
Regeln der Multilevel Security

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.



Formale Sicherheitsmodelle

Postulate

- präzise und eindeutige Beschreibung
- weitgehende Beschränkung auf Sicherheitseigenschaften

Modell

- Satz von Regeln

  Satz von Operationen im Kontext eines Betriebssystems
  z. B. erzeuge, lies, schreibe, ändere Attribut,...

Ziel: Beweis

- jede Folge von Operationen führt einen sicheren, d.h. den Regeln entsprechenden Zustand wieder in einen sicheren Zustand über
- Implementierung durch Betriebssystem entspricht Modell

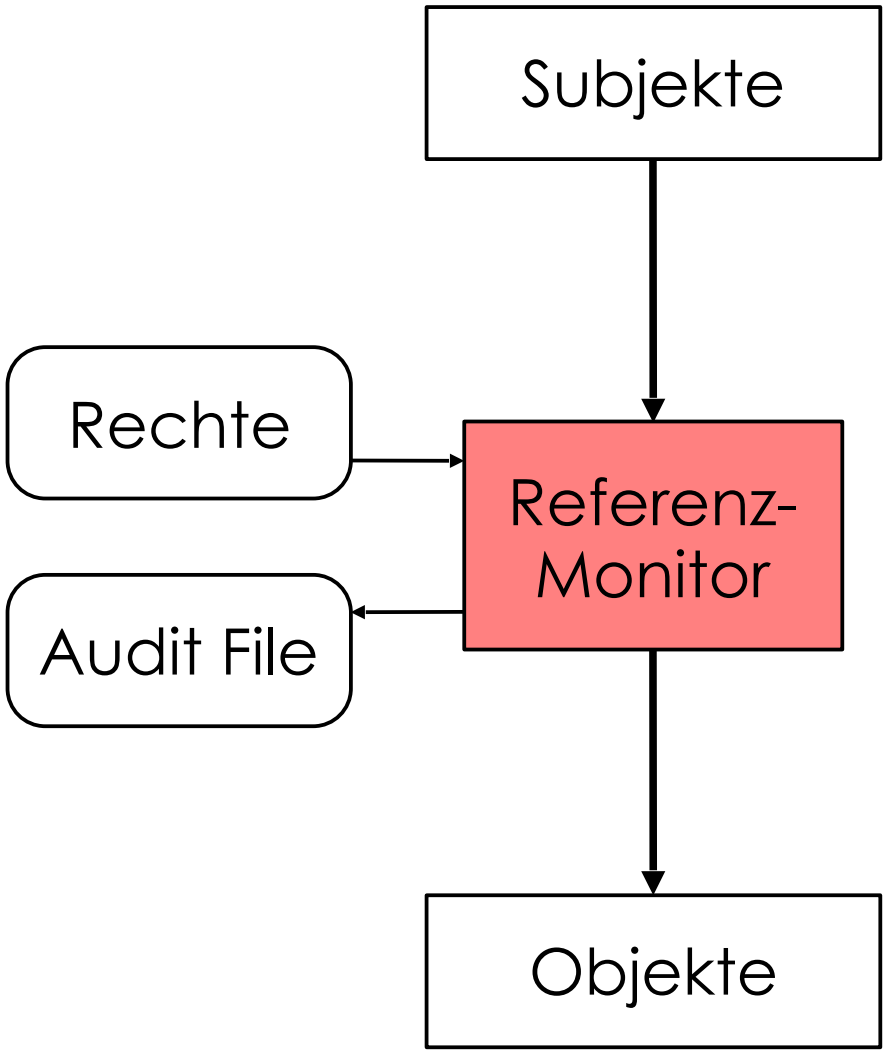
Begriff: Referenzmonitor

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
  - komplexe Regelsätze, schwer zu überblicken
- TrustedBSD, Apple Seatbelt
  - Grundlage für die **App Sandbox** in iOS und OS X
- Windows 8 App Container

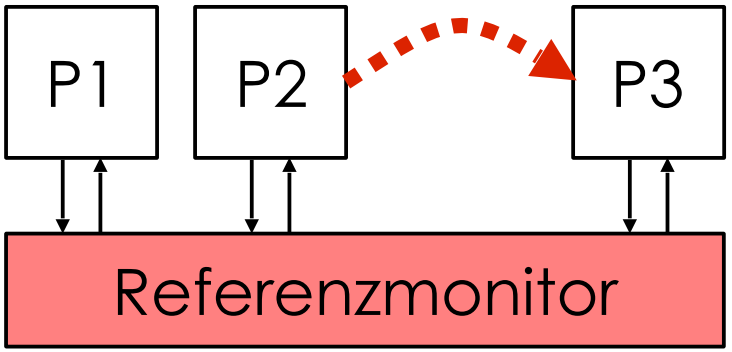
Kombination von **Entwicklerwissen** und **Nutzerinteressen:**

- Entwickler beschreibt über **Manifest**, was die Anwendung im Normalfall benötigt
- System setzt diese Beschränkungen durch
- auch bei einem Angriff werden nicht mehr Rechte erlangt
- Nutzer kann durch **ACLs** seine Dokumente verwalten

Aber: Verdeckte Kanäle (Covert channels)

Informationsfluss an veröffentlichten, durch Schutzmechanismen überwachten Schnittstellen vorbei

→ also etwa Info von P2 an P3,
obwohl von P2 nach P3 keine Info fließen darf



Speicherbasierte verdeckte Kanäle

"Covert Storage Cannels"

Notation für Beispiele im Folgenden

→ ein Prozess sendet,

← ein anderer Prozess empfängt

Beispiele

- Dateinamen und Attribute
  → neuer Dateiname mit n Zeichen
  ← Lesen des Verzeichnisses

  - Bandbreite: n Zeichen (pro Op)
  - einfach zu eliminieren (keine Leserechte für Verzeichnis)

Zeitbasierte verdeckte Kanäle

“Covert Timing Cannels”

- Anteil an CPU für einen Prozess

  → rechenintensiv/nicht rechenintensiv im Sekundenabstand

  ← Beobachtung, wie hoch der eigene Anteil

- Bandbreite: ???
- Voraussetzung: genaue Zeitmessung
- Gegenmaßnahme:
  - Ausschaltung des Zugriffs auf Uhren inklusive timeouts
  - absichtliche Ungenauigkeit von Uhren

Weitere Beispiele

- Steganografie nutzt verdeckte Kanäle
- Stromverbrauch
- Sensoren und physikalische Kanäle

Quellenangabe

- Vieles basiert auf dem Lehrbuch
- Andy Tanenbaum, Herbert Bos
  Modern Operating Systems

Zusammenfassung

- Konstruktionsprinzipien für sichere Systeme
- ACLs, Sandboxing, Capabilities
- Regelbasierte Zugriffssteuerung
- Verdeckte Kanäle

NICHT IN DIESER Vorlesung:

- Schutzziele etc (Prof Strufe und Dr Köpsell)
- Secure Booting (Hauptstudium)
- virtuelle Maschinen, Container, …
- Audit
- Intrusion Detection, Virenscanner, …