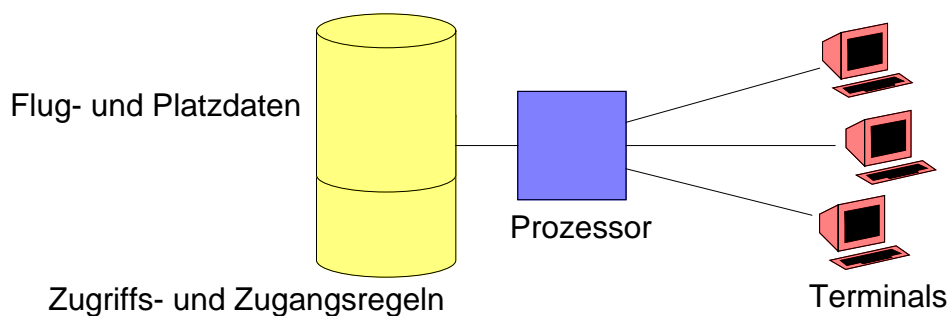


I Datensicherheit und Zugriffsschutz

I Datensicherheit und Zugriffsschutz

1 Problemstellung

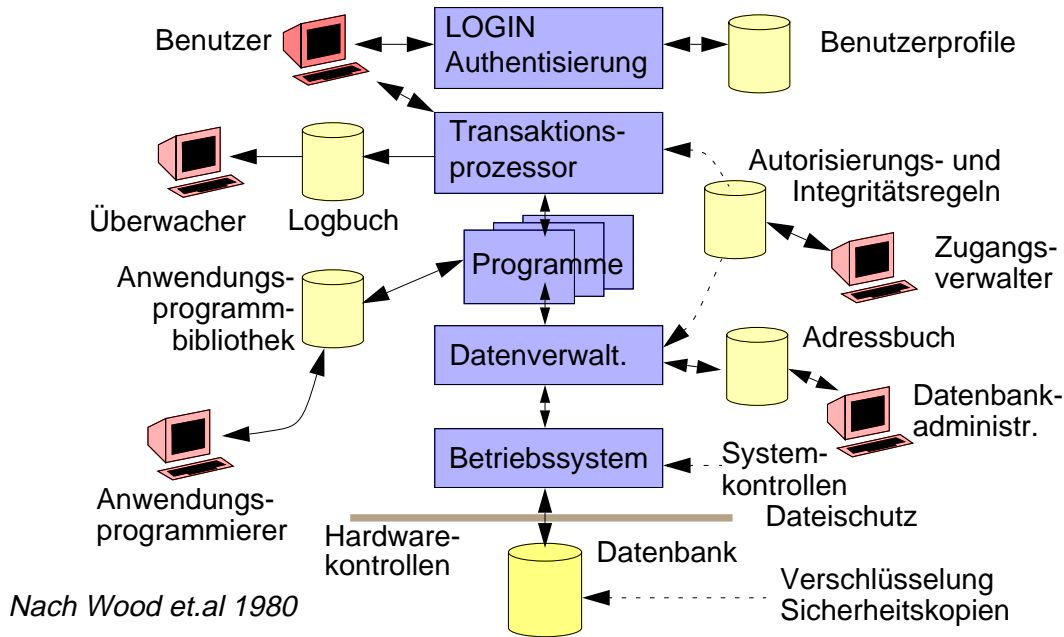
- Beispiel: Zugang zu einer Datenbank zur Flugreservierung und -buchung



- ◆ Was sind mögliche Beeinträchtigungen der Datensicherheit?

1 Problemstellung (2)

■ Überprüfungen beim Transaktionsbetrieb (Datenbankanwendung)



1 Problemstellung (3)

- **Illegaler Datenzugriff**
 - ◆ Daten sind zugreifbar, die vertraulich behandelt werden sollen
- **Illegales Löschen von Daten**
 - ◆ Kein Zugriff, aber Daten werden gelöscht
- **Illegales Manipulieren von Daten**
 - ◆ Daten werden in böswilliger Absicht verändert
- **Zerstörung von Rechensystemen**
 - ◆ physisches Zerstören von Teilen der Rechenanlage

1.1 Umgebung der Rechenanlage

- ▲ Naturkatastrophen
 - ◆ Erdbeben, Vulkanausbrüche etc. können Rechenanlage und Datenbestand zerstören
- ▲ Unfälle
 - ◆ Gasexplosion, Kühlwasserlecks in der Klimaanlage oder Ähnliches zerstören Rechner und Daten
- ▲ Böswillige Angriffe
 - ◆ Zerstörung der Rechenanlage und des Datenbestands durch Sabotage (Bombenanschlag, Brandanschlag etc.)
- ▲ Unbefugter Zutritt zu den Räumen des Rechenzentrums
 - ◆ Diebstahl von Datenträgern
 - ◆ Zerstörung von Daten
 - ◆ Zugang zu vertraulichen Daten

1.2 Systemsoftware

- ▲ Versagen der Schutzmechanismen
 - ◆ System lässt Unbefugte auf Daten zugreifen oder Operationen ausführen
- ▲ Durchsickern von Informationen
 - ◆ Anwender können anhand scheinbar unauffälligen Systemverhaltens Rückschlüsse auf vertrauliche Daten ziehen (*Covert channels*)
- Beispiel: verschlüsselt abgespeicherte Passwörter sind zugänglich
 - ◆ Entschlüsselungsversuch der Passwörter außerhalb der Rechenanlage
 - ◆ "Wörterbuchattacke": Raten von Passwörtern möglich

1.3 Systemprogrammierer

- ▲ Umgehen oder Abschalten der Schutzmechanismen
- ▲ Installation eines unsicheren Systems
 - ◆ erlaubt dem Systemprogrammierer die Schutzmechanismen von außen zu umgehen
- ▲ Fehler beim Nutzen von Bibliotheksfunktionen innerhalb sicherheitskritischer Programme
 - ◆ S-Bit Programme unter UNIX laufen mit der Benutzererkennung des Dateibesitzers, nicht unter der des Aufrufers
 - Fehlerhafte S-Bit Programme können zur Ausführung von Code unter einer fremden Benutzererkennung gebracht werden
 - S-Bit Programme mit "root-Rechten" besonders gefährlich

1.3 Systemprogrammierer (2)

- ◆ Buffer-Overflow Fehler bei Funktionen
`gets()`, `strcpy()`, `strcat()`, `sprintf("%s",...)`
 - Zu lange Eingaben überschreiben den Stackspeicher des Prozessors
 - Mit genauer Kenntnis ist das Ausführen beliebigen Codes erzwingbar
- ◆ Fehlerhafte Parameterprüfung beim Aufruf von Funktionen
`system()`
- ◆ Beispiel: Löschen einer Datei
 - Name der Datei wurde in Variable `file` eingelesen
 - Aufruf von `system` mit Parameter `strcat("rm ", file)`
 - Gibt man für den Dateinamen den String
`"fn ; xterm -display myhost:0 &"`
ein, bekommt man ein Fenster auf der aufgebrochenen Maschine

1.4 Rechnerhardware

- ▲ Versagen der Schutzmechanismen
 - ◆ erlauben nicht-autorisierten Zugriff
- ▲ Fehlerhaft Befehlsausführung
 - ◆ Zerstörung von wichtigen Daten
- ▲ Abstrahlungen
 - ◆ erlaubt Ausspähen von Daten

1.5 Datenbasis

- ▲ Falsche Zugriffsregeln
 - ◆ erlauben nicht-autorisierten Zugriff

1.6 Operateur

- ▲ Kopieren vertraulicher Datenträger
- ▲ Diebstahl von Datenträgern
- ▲ Initialisierung mit unsicherem Zustand
 - ◆ Operateur schaltet beispielsweise Zugriffskontrolle ab
- ▲ Nachlässige Rechnerwartung
 - ◆ Nachbesserungen der Systemsoftware (*Patches*) werden nicht eingespielt
 - Sicherheitslücken werden nicht gestopft

1.7 Sicherheitsbeauftragter

- ▲ Fehlerhafte Spezifikation der Sicherheitspolitik
 - ◆ dadurch Zugang für Unbefugte zu vertraulichen Daten oder
 - ◆ Änderungen von Daten durch Unbefugte möglich
- ▲ Unterlassene Auswertung von Protokolldateien
 - ◆ Einbrüche und mögliche Sicherheitslücken werden nicht rechtzeitig entdeckt

1.8 Kommunikationssystem

- ▲ Abhören der Kommunikationsleitungen (*Sniffing*)
 - ◆ z.B. Telefonverbindung bei Modemnutzung oder serielle Schnittstellen
 - ◆ z.B. Netzwerkverkehr auf einem Netzwerkstrang
 - ◆ Ermitteln von Passwörtern und Benutzerkennungen
 - manche Dienste übertragen Passwörter im Klartext (z.B. ftp, telnet, rlogin)
 - ◆ Zugriff auf vertrauliche Daten
 - ◆ unbefugte Datenveränderungen
 - Verfälschen von Daten
 - Übernehmen von bestehenden Verbindungen (*Hijacking*)
 - Vorspiegeln falscher Netzwerkadressen (*Spoofing*)

1.8 Kommunikationssystem (2)

- ▲ Illegale Nutzung von Diensten über das Netzwerk
 - ◆ Standardsysteme bieten eine Menge von Diensten an (z.B. ftp, telnet, rwho u.a.)
 - ◆ Sicherheitslücken von Diensten werden publik gemacht und sind auch von "dummen" Hackern nutzbar (*Exploit scripts*)
<http://www.rootshell.org>
 - ◆ Auch bei temporär am Netzwerk angeschlossenen Computern eine Gefahr
 - z.B. Linux-Maschine mit PPP-Verbindung an das Uni-Netz
 - Voreinstellungen der Standardinstallation meist unsicher

1.9 Terminal

- ▲ Ungeschützter Zugang zum Terminal
 - ◆ Nutzen einer fremden Benutzererkennung
 - ◆ Zugriff auf vertrauliche Daten
 - ◆ unbefugte Datenveränderungen

1.10 Benutzer

- ▲ Nutzen anderer Kennungen
 - ◆ erlauben nicht-autorisierten Zugriff
 - ◆ unbefugte Datenveränderungen
 - ◆ unbefugte Weitergabe von Informationen
- ▲ Einbruch von Innen
 - ◆ leichter Zugang zu möglichen Sicherheitslöchern (z.B. bei Diensten)

1.11 Anwendungsprogrammierung

- ▲ Nichteinhalten der Spezifikation
 - ◆ Umgehen der Zugriffskontrollen
- ▲ Einfügen von „böartigen“ Befehlsfolgen
 - ◆ *Back door*: Hintertür gibt dem Programmierer im Betrieb Zugang zu vertraulichen Daten oder illegalen Operationen
 - ◆ *Trojan horse*: Unter bestimmten Bedingungen werden illegale Operationen ohne Trigger von außen angestoßen

1.12 "Tracker Queries"

- Beispiel: Datenbanksysteme
 - ◆ Zugriff auf Einzelinformationen ist verboten (Vertraulichkeit)
 - ◆ statistische Informationen sind erlaubt
- ▲ Grenzen möglicher Sicherheitsmaßnahmen:
Zugriff auf Einzelinformationen dennoch möglich
 - ◆ geeignete Anfragen kombinieren (*Tracker queries*)
- Beispiel: Gehaltsdatenbank

1.12 "Tracker Queries" (2)

- Tabelle der Datenbankeinträge:

Nr.	Name	Geschl.	Fach	Stellung	Gehalt	Spenden
1	Albrecht	m	Inf.	Prof.	60.000	150
2	Bergner	m	Math.	Prof.	45.000	300
3	Cäsar	w	Math.	Prof.	75.000	600
4	David	w	Inf.	Prof.	45.000	150
4	Engel	m	Stat.	Prof.	54.000	0
5	Frech	w	Stat.	Prof.	66.000	450
6	Groß	m	Inf.	Angest.	30.000	60
8	Hausner	m	Math.	Prof.	54.000	1500
9	Ibel	w	Inf.	Stud.	9.000	30
10	Jost	m	Stat.	Angest.	60.000	45
11	Knapp	w	Math.	Prof.	75.000	300
12	Ludwig	m	Inf.	Stud.	9.000	0

1.12 "Tracker Queries" (3)

- Anfragen und Antworten:

◆ Anzahl('w'): 5

◆ Anzahl('w' und (nicht 'Inf' oder nicht 'Prof.')): 4

◆ mittlere Spende('w'): 306

◆ mittlere Spende('w' und (nicht 'Inf.' oder nicht 'Prof.')): 345

- Berechnung:

◆ Spende('David'): $306 * 5 - 345 * 4 =$
 $1530 - 1380 =$
150

2 Zugriffslisten

- Identifikation von Subjekten, Objekten und Berechtigungen
 - ◆ Subjekt: Person oder Benutzererkennung im System (repräsentiert jemanden, der Aktionen ausführen kann)
 - ◆ Objekt: Komponente des Systems (repräsentiert Ziel einer Aktion)
 - ◆ Berechtigung: z.B. Leseberechtigung auf einer Datei (repräsentiert die Erlaubnis für die Ausführung einer Aktion)
- Erfassung der Berechtigungen in einer Subjekt-Objekt-Matrix: Zugriffsliste (*Access control list, ACL*)

2.1 Beispiel für Zugriffslisten

- Personaldatensatz
 - ◆ besteht aus: Name, Abteilung, Personalnummer, Lohn- oder Gehaltsgruppe
- Personaldateien (Objekte)
 - ◆ D_{LA} : Personaldaten der leitenden Angestellten
 - ◆ D_{AN} : Personaldaten der sonstigen Angestellten
 - ◆ D_{AR} : Personaldaten der Arbeiter
- Prozeduren (gehören zu den Aktionen)
 - ◆ R_{LA} : Lesen von Pers.-Nr. und Lohn-/Gehaltsgr. aus D_{LA}
 - ◆ $R_{AN/AR}$: Lesen von Pers.-Nr. und Lohn-/Gehaltsgr. aus D_{AN} oder D_{AR}
 - ◆ R_{post} : Lesen von Name, Abteilung und Pers.-Nr.

2.1 Beispiel für Zugriffslisten (2)

- Benutzer (Subjekte)
 - ◆ S_{pers} : Leiter des Personalbüros
 - Besitzer aller Dateien und Prozeduren
 - Lese- und Schreibrecht für alle Dateien
 - Aufrufrecht für alle Prozeduren
 - ◆ S_{stellv} : Sachbearb. leitende Angestellte, stellvertr. Leiter Personalbüro
 - Lese- und Schreibrecht für D_{AN} und D_{AR}
 - Aufrufrecht für R_{LA}
 - ◆ S_{sach} : Sachbearbeiter Angestellte u. Arbeiter
 - Aufrufrecht für $R_{AN/AR}$
 - ◆ S_{post} : Poststelle
 - Aufrufrecht für R_{post} auf alle Dateien

2.1 Beispiel für Zugriffslisten (3)

- Berechtigungen werden in Matrix ausgedrückt:

	D_{LA}	D_{AN}	D_{AR}	R_{LA}	$R_{AN/AR}$	R_{post}
S_{pers}	O, R, W	O, R, W	O, R, W	O, I	O, I	O, I
S_{stellv}		R, W	R, W	I		
S_{sach}					I	
S_{post}						I
R_{LA}	R					
$R_{AN/AR}$		R	R			
R_{post}	R	R	R			

- O = *Owner*; Besitzer der Datei oder Prozedur
- R = *Read*; volle Leseberechtigung
- W = *Write*; volle Schreibberechtigung
- I = *Invoke*; Aufrufberechtigung

2.2 Beispiel: UNIX

- Zugriffslisten für
 - ◆ Dateien und Geräte
 - ◆ Shared-Memory-Segmente
 - ◆ Message-Queues
 - ◆ Semaphore
 - ◆ etc.
- Berechtigungen:
 - ◆ Lesen (*read*), Schreiben (*write*), Ausführen (*execute*)
 - ◆ für Besitzer, Gruppe und alle anderen unterscheidbar
- Subjekte:
 - ◆ Prozesse
 - ◆ Besitzer (Benutzer) und Zugehörigkeit zu einer oder mehreren Gruppen

2.2 Beispiel: UNIX

- Superuser
 - ◆ Benutzer *root* hat automatisch alle Zugriffsrechte
- S-Bit-Programme
 - ◆ S-Bit ist ein besonderes Recht auf der Binärdatei des Programms
 - ◆ Besitzer der Datei wird bei der Ausführung auch Besitzer des Prozess (sonst wird Aufrufer Besitzer des Prozess)
- ★ Vorteil
 - ◆ Bereitstellen von Prozessen, die kontrolliert Aufrufern höhere Zugriffsberechtigungen erlauben
- ▲ Nachteil
 - ◆ Fehler im Prozess gibt Aufrufer volle Rechte des Programmbesitzers
 - ◆ fatal, falls das Programm *root* gehört

2.3 Implementierung

- Globale Tabelle/Matrix
 - ◆ System hält eine Datenstruktur und prüft im betreffenden Eintrag die Berechtigungen
 - ◆ Tabelle üblicherweise recht groß: passt evtl. nicht in den Speicher
- Zugriffslisten an den Objekten
 - ◆ jedes Objekt hält eine Liste der Berechtigungen (z.B. Unix Datei: Inode)
 - ◆ verringert üblicherweise den Platzbedarf für die Einträge (unnötige Felder der Matrix werden nicht repräsentiert)
- Zugriffslisten an den Subjekte
 - ◆ jedes Subjekt hält eine Liste von Objekten und den Berechtigungen, die das Subjekt für das Objekt hat
 - ◆ ähnlich Capabilities

3 Schutz durch Speicherverwaltung

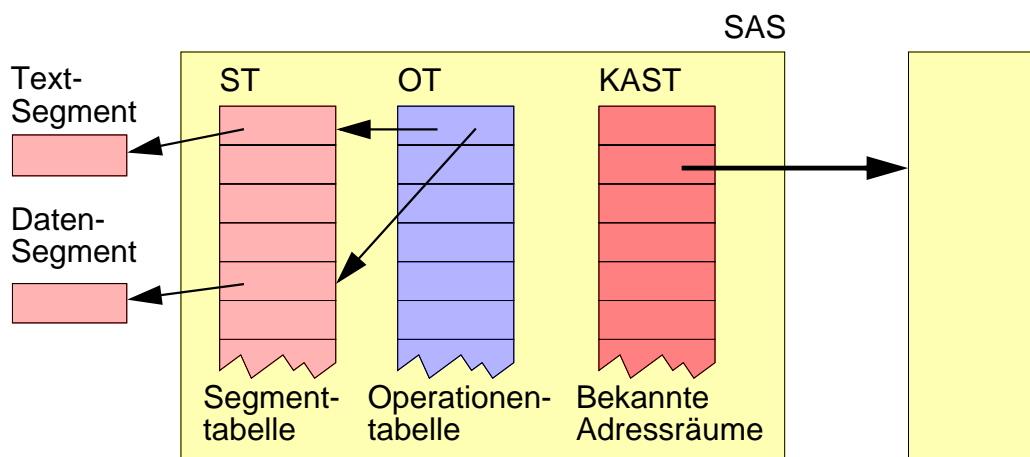
- Schutz vor gegenseitigem Speicherzugriff
 - ◆ Segmentierung und Seitenadressierung erlauben es, jedem Prozess nur den benötigten Speicher einzublenden
 - ◆ Segmentverletzung löst Unterbrechung aus
- Systemaufrufe
 - ◆ definierter Weg von einer Schutzumgebung (der des Prozesses) in eine andere (der des Betriebssystems)
- Erweiterung dieses Konzepts:
 - ◆ allgemeine Prozeduraufrufe zwischen verschiedenen Schutzumgebungen, realisiert mit der Speicherverwaltung und deren Hardware (MMU)

3.1 Modulkonzept von Habermann

- Idee (von 1976)
 - ◆ Adressräume (Module) bilden Schutzumgebungen
 - ◆ Adressräume bieten definierte Operationen an (ähnlich wie das Betriebssystem Systemaufrufe anbietet)
 - ◆ Parameter werden in speziellen Segmenten übergeben
- ★ Bietet allgemeinen Schutz der Module und erlaubt kontrollierte Interaktionen
- Module besitzen einen statischen Adressraum (*SAS, Static address space*)
 - ◆ enthält Liste von Segmenten, die zu dem Modul gehören bzw. von dem Modul zugegriffen werden dürfen
 - ◆ enthält Liste von angebotenen Operationen mit den Angaben, welche Segmente jede Operation benötigt (u.a. Segment für die auszuführenden Instruktionen)

3.1 Modulkonzept von Habermann (2)

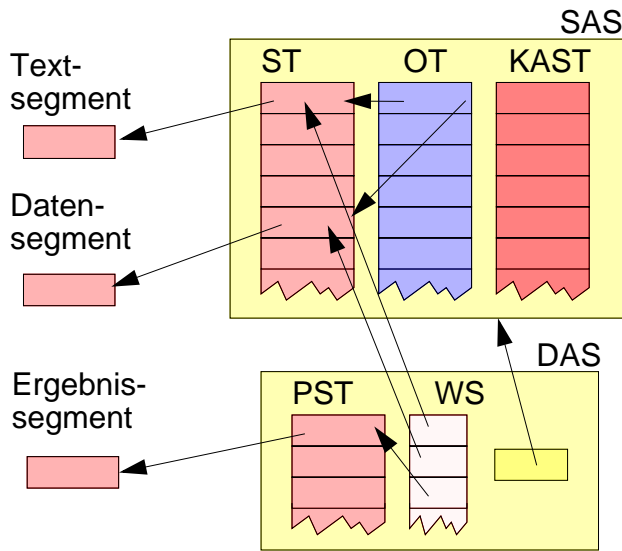
- ◆ enthält Liste von bekannten Adressräumen anderer Module (dort können dann Operationen aufgerufen werden)



KAST = *Known address space table*

3.1 Modulkonzept nach Habermann (3)

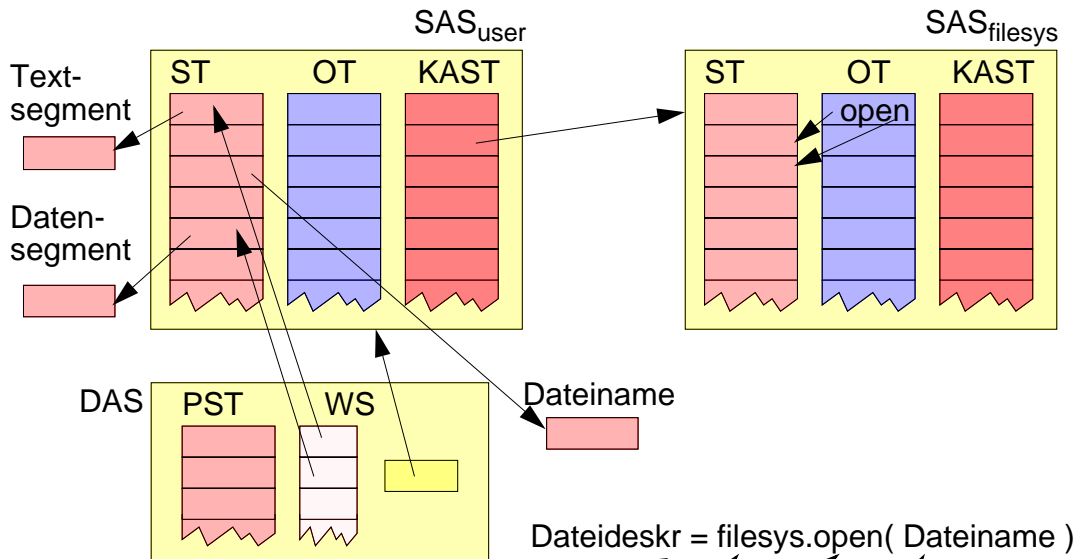
- Aktivitätsträger sind einem dynamischen Adressraum zugeordnet (*DAS, Dynamic address space*)



- ◆ enthält Segmenttabelle für Parameter und Ergebnisse (*PST*)
- ◆ enthält Verweise auf Segmenttabellen, die den Adressraum zusammensetzen (*WS= Working space*)

3.2 Beispielaufruf

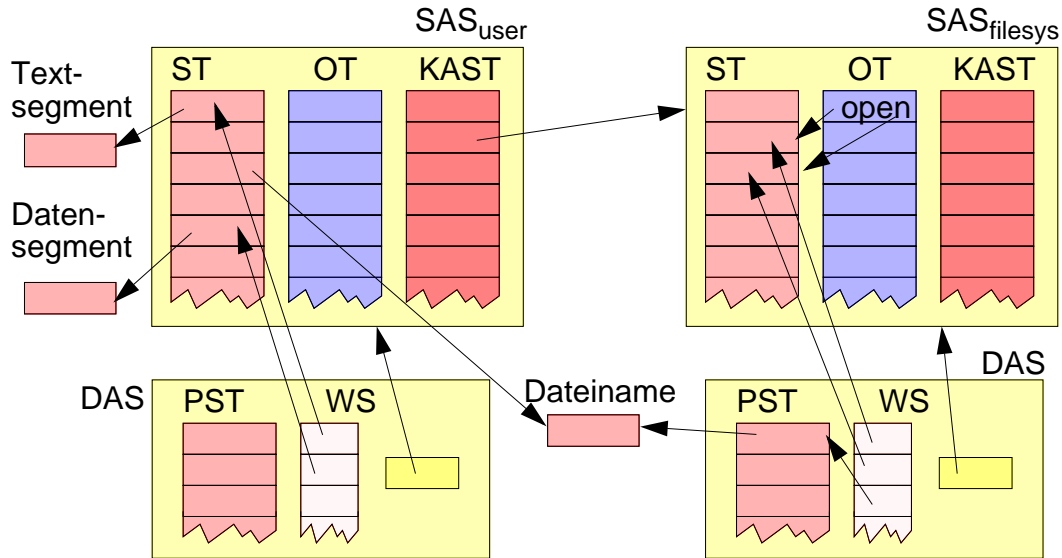
- SAS des Benutzers ruft Operation „open“ des Dateisystem-SAS auf



Dateideskr = filesys.open(Dateiname)
ASCALL(KAST[1], OT[0], (ST[2]), PST[2])

3.2 Beispielaufruf (2)

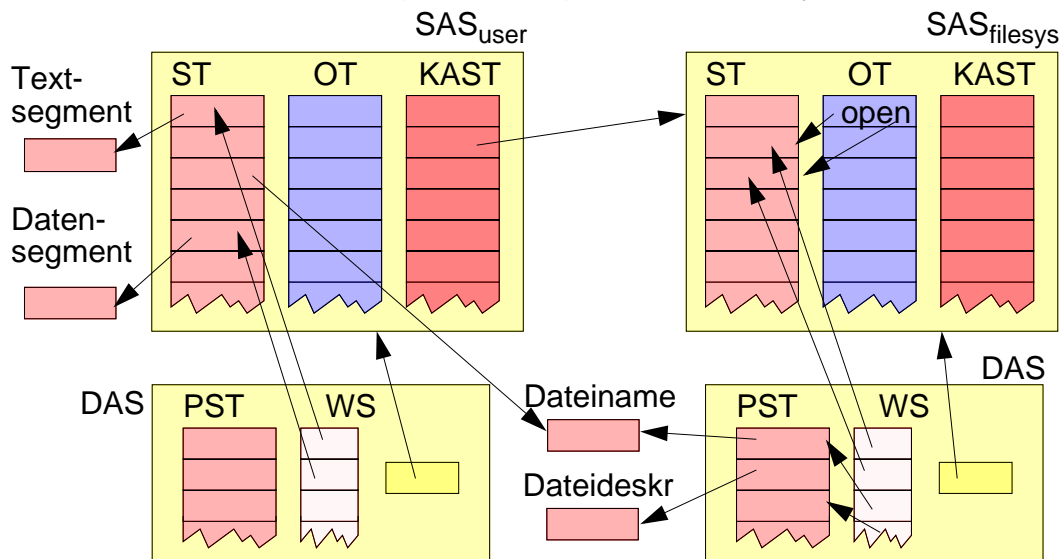
- SAS des Benutzers ruft Operation „open“ des Dateisystem-SAS auf



- ◆ für den Aufruf von „open“ wird ein neuer DAS erzeugt

3.2 Beispielaufruf (3)

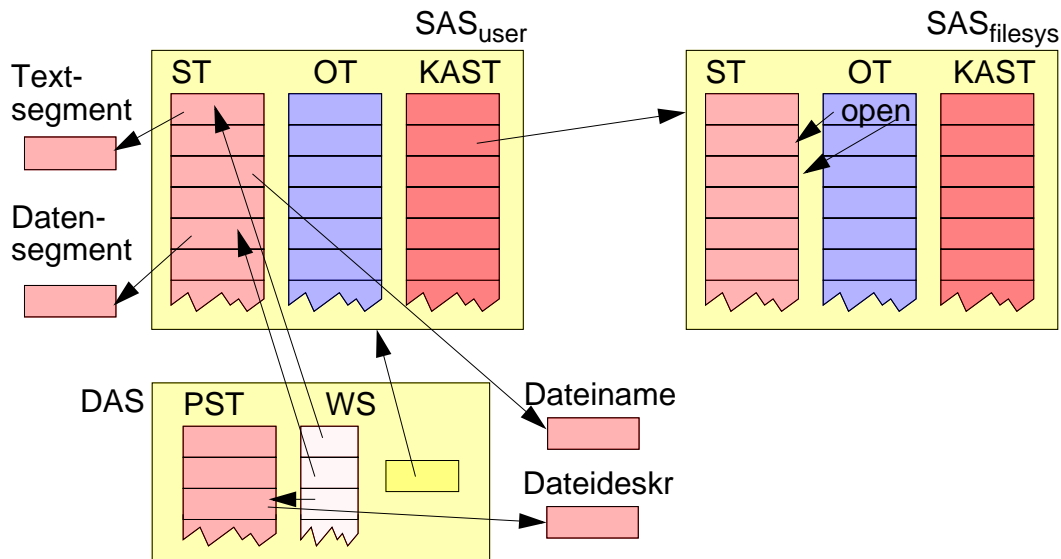
- SAS des Benutzers ruft Operation „open“ des Dateisystem-SAS auf



- ◆ „open“ erzeugt neues Segment für den Dateideskriptor

3.2 Beispielaufruf (4)

- SAS des Benutzers ruft Operation „open“ des Dateisystem-SAS auf



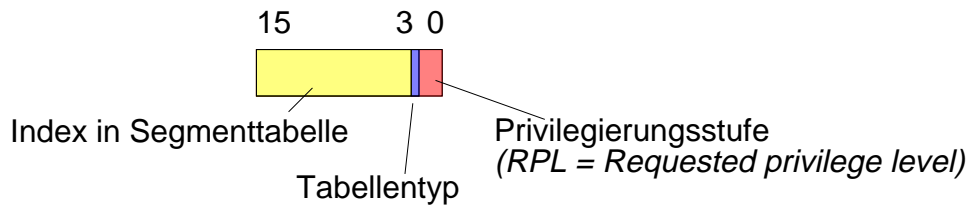
- ◆ Ergebnissegment wird an den Aufrufer zurückgegeben

3.3 Beispiel: Pentium

- Privilegierungsstufen
 - ◆ Stufe 0: höchste Privilegien (privilegierte Befehle, etc.): BS Kern
 - ◆ Stufe 1: BS Treiber
 - ◆ Stufe 2: BS Erweiterungen
 - ◆ Stufe 3: Benutzerprogramme
- Merke:
 - ◆ kleine Stufennummer: hohe Privilegien
 - ◆ große Stufennummer: kleine Privilegien

3.3 Beispiel: Pentium (2)

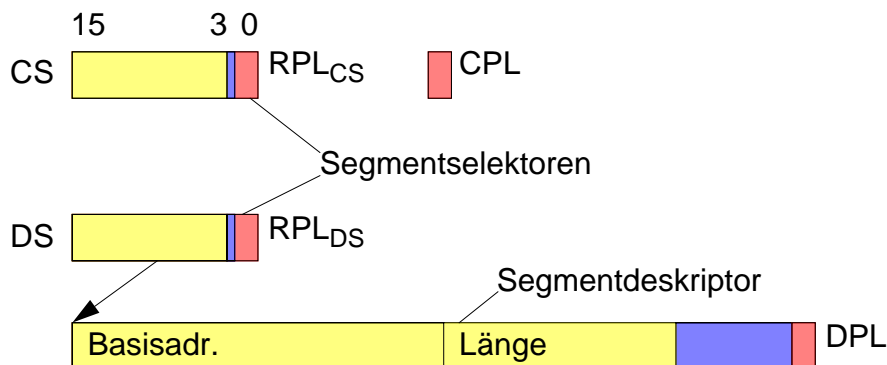
- Segmentsselektoren enthalten Privilegierungsstufe



- Codesegmentselektor (CS) bestimmt aktuelle Privilegierungsstufe
- ◆ CPL = *Current privilege level*

3.3 Beispiel: Pentium (3)

- Datenzugriff (z.B. auf Datensegment DS)



- ◆ DPL = *Descriptor privilege level*
- ◆ CPL ist normalerweise gleich RPL_{CS}
- ◆ Zugriff wird erlaubt, wenn: $DPL \geq \max(CPL, RPL_{DS})$
- ◆ ansonsten wird Unterbrechung ausgelöst (Schutzverletzung)

3.3 Beispiel: Pentium (4)

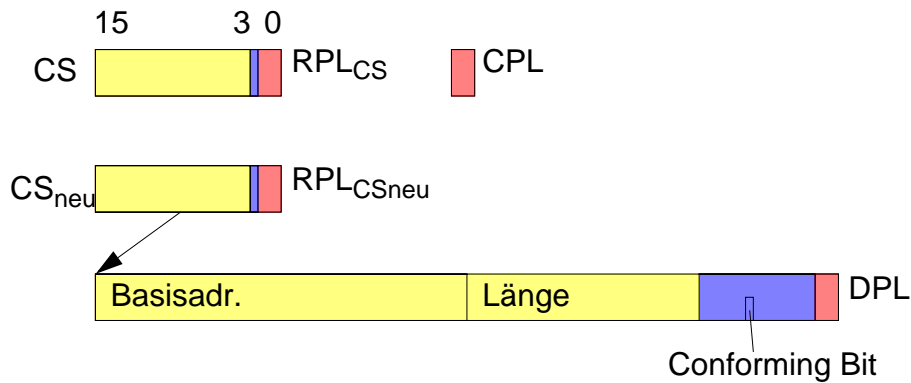
- Erläuterung:
 - ◆ $DPL < CPL$: *Schutzverletzung*
augenblickliche Privilegierungsstufe hat weniger Privilegien als im Deskriptor verlangt (CPL hat höhere Stufenr. als der Deskriptor)
 - ◆ $DPL \geq CPL$: OK
augenblickliche Privilegierungsstufe ist mindestens so hoch wie die im Deskriptor
- ★ Segment kann nur angesprochen werden, wenn augenblickliche Stufe die gleichen oder mehr Privilegien beinhaltet.

3.3 Beispiel: Pentium (5)

- Erläuterung:
 - ◆ $DPL < RPL(ds)$: *Schutzverletzung*
Selektor hat weniger Privilegien als der Deskriptor (Selektor hat größere Stufe als der Deskriptor)
 - ◆ $DPL \geq RPL(ds)$: OK
Selektor hat mindestens die gleiche Privilegierungsstufe wie der Deskriptor
- ★ Selektor darf keine geringeren Privilegien versprechen als wirklich verlangt sind.

3.3 Beispiel: Pentium (5)

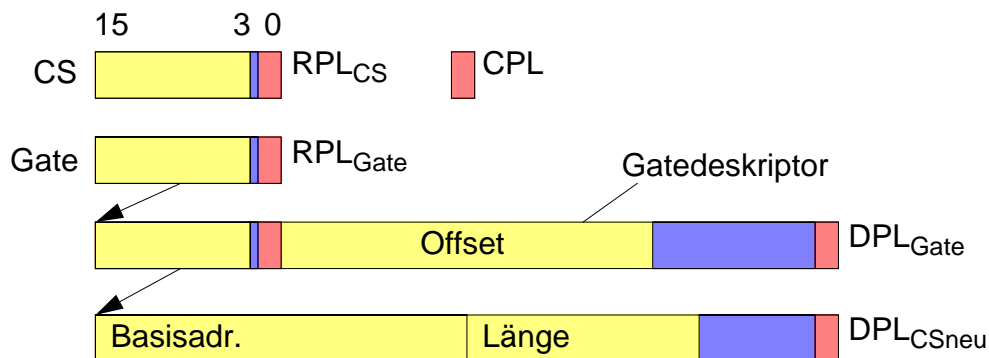
■ Sprünge in andere Codesegmente (*Far Call*)



- ◆ Sprung wird erlaubt, falls: $DPL = CPL$ oder Conforming Bit gesetzt und $DPL \leq CPL$
- ◆ Im Falle von $DPL \leq CPL$ wird jedoch CPL nicht geändert (Codesegment hat höheres Privileg, CPL bleibt aber unverändert)

3.3 Beispiel: Pentium (6)

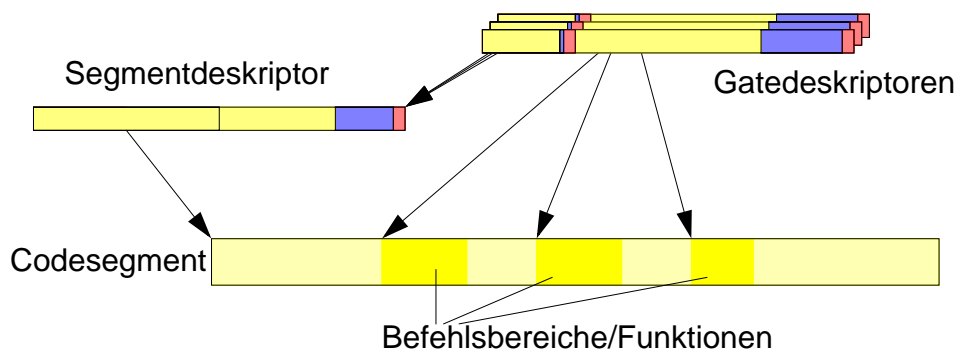
■ Kontrolltransfer mit einem *Gate*



- ◆ Gatedeskriptoren stehen wie Segmentdeskriptoren in der Segmenttabelle
- ◆ Gatedeskriptor enthält Segmentselektor für das Codesegment und einen Offset zu diesem Segment, an dem der Einsprungpunkt liegt
- ◆ Kontrolltransfer (*CALL* Aufruf) wird erlaubt, falls:
 $DPL_{Gate} \geq \max(CPL, RPL_{Gate})$ und $DPL_{CSneu} \leq CPL$

3.3 Beispiel: Pentium (7)

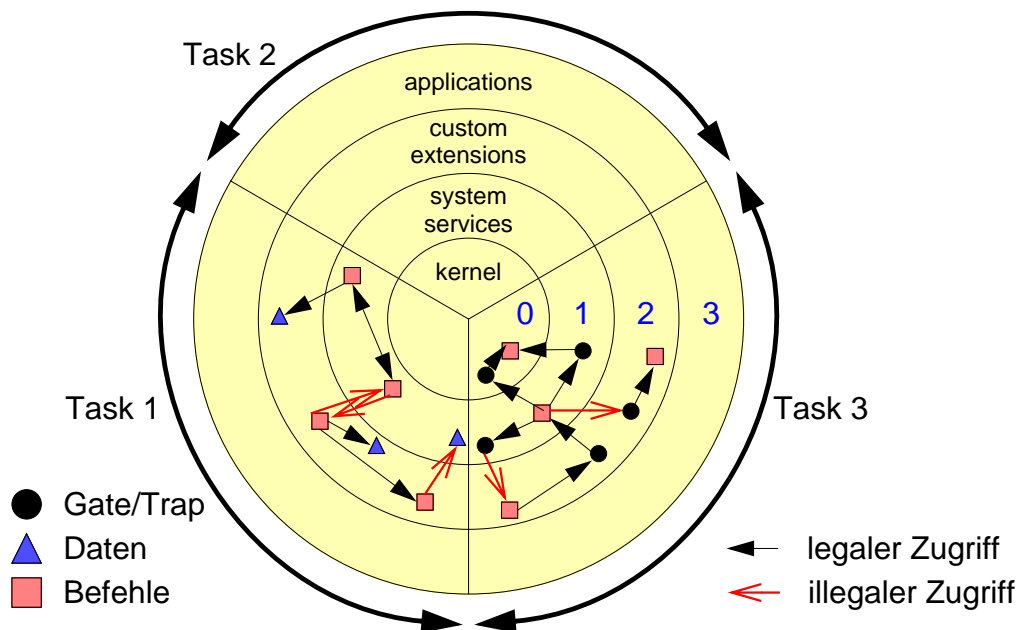
- Gates erlauben den kontrollierten Sprung in ein privilegierten Befehlsbereich



- ◆ es existiert auch der entsprechende Rücksprung
- ◆ für jede Privilegierungsstufe gibt es einen eigenen Stack; dieser wird mit umgeschaltet
- ◆ Parameter werden automatisch auf den neuen Stack kopiert (Anzahl wird im Gatedeskriptor vermerkt)

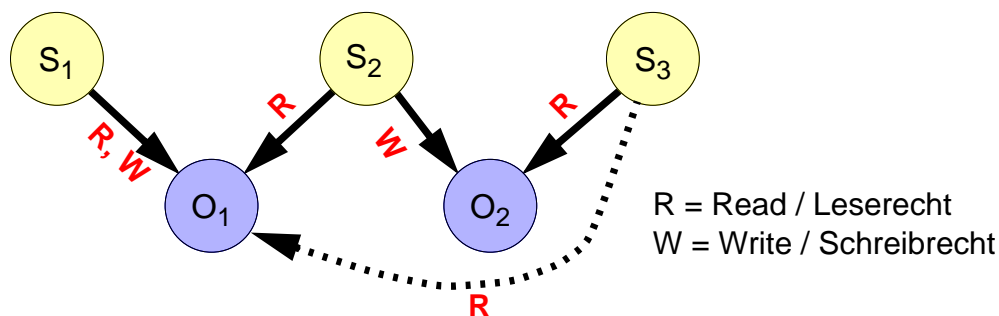
3.3 Beispiel: Pentium (8)

- Beispiel eines realisierten Schutzsystems



4 Capability-basierte Systeme

- Ein Benutzer (Subjekt) erhält eine Referenz auf ein Objekt
 - ◆ die Referenz enthält alle Rechte, die das Subjekt an dem Objekt besitzt
 - ◆ bei der Nutzung der Capability (Zugriff auf das Objekt) werden die Rechte überprüft



Subjekte und Objekte; Weitergabe einer Capability (O₁ von S₂ nach S₃)

4 Capability-basierte Systeme (2)

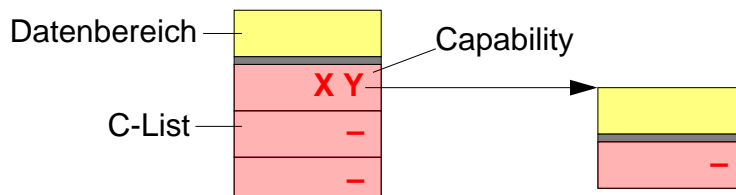
- ★ Vorteile
 - ◆ keine Speicherung von Rechten beim Objekt oder Subjekt nötig; Capability enthält Zugriffsrechte
 - ◆ leichte Vergabe von individuellen Rechten
 - ◆ einfache Weitergabe von Zugriffsrechten möglich
- ▲ Nachteile
 - ◆ Weitergabe nicht kontrollierbar
 - ◆ Rückruf von Zugriffsrechten nicht möglich
 - ◆ Capability muss vor Fälschung und Verfälschung geschützt werden (z.B. durch kryptographische Mittel oder durch Speicherverwaltung)

4.1 Beispiel: Hydra

- Hydra ist ein Capability-basiertes Betriebssystem
 - ◆ entwickelt Mitte der Siebziger Jahre an der Carnegie-Mellon University
 - ◆ lief auf einem speziellen Multiprozessor namens **C.mmp**
 - ◆ Capability-Mechanismen sind integraler Bestandteil des Betriebssystems
- Objekte in Hydra werden durch Capabilities angesprochen und geschützt
 - ◆ Objekte haben einen Typ (z.B. Prozeduren, Prozesse/LNS, Semaphore, Datei etc.)
 - ◆ Capabilities haben entsprechenden Typ
 - ◆ benutzerdefinierte Typen sind möglich

4.1 Beispiel: Hydra (2)

- ◆ generische Operationen für alle Typen implementiert durch das Betriebssystem
- ◆ Objekte besitzen eine Liste von Capabilities auf andere Objekte (genannt *C-List*)
- ◆ Capabilities enthalten Rechte
- ◆ Objekte besitzen einen Datenbereich (implementiert durch geschütztes Segment)



4.1 Beispiel: Hydra (3)

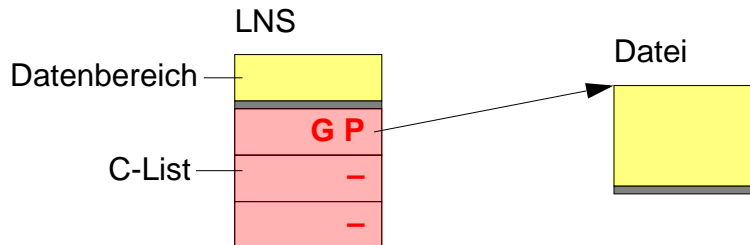
- Prozesse (Subjekte)
 - ◆ Prozesse besitzen einen aktuellen Kontext, den LNS (*Local name space*)
 - ◆ LNS ist ein Objekt
 - ◆ zum LNS gehört einen Aktivitätsträger (Thread)
 - ◆ LNS kann nur auf Objekte zugreifen, die in seiner C-List stehen (mehrstufige Zugriffe, z.B. auf die C-List eines Objekts, dessen Capabilities in der C-List des LNS steht, sind möglich; Pfad zur eigentlichen Capability)
- Capabilities
 - ◆ Prozesse können nur über Systemaufrufe ihre Capabilities bzw. ihre C-List bearbeiten
 - ◆ Capabilities können nicht gefälscht oder verfälscht werden
 - ◆ Betriebssystem kann sicheres Schutzkonzept basierend auf Capabilities implementieren

4.2 Datenzugriff in Hydra

- Operationen auf dem Datenbereich
 - ◆ *Getdata*: kopiere Abschnitt aus dem Datenbereich eines Objekts in den Datenbereich des LNS
 - ◆ *Putdata*: kopiere Abschnitt aus dem Datenbereich des LNS in den Datenbereich eines Objekts
 - ◆ *Adddata*: füge Daten zu dem Datenbereich eines Objekts hinzu
- Dazugehörige Rechte:
 - ◆ **GETRTS**: erlaubt den Aufruf von *Getdata*
 - ◆ **PUTRTS**: erlaubt den Aufruf von *Putdata*
 - ◆ **ADDRTS**: erlaubt den Aufruf von *Adddata*
- Rechte müssen in der Capability zum Objekt gesetzt sein

4.2 Datenzugriff in Hydra (2)

- Beispiel: Implementierung von Dateien
 - ◆ *Getdata* erlaubt das Lesen von Daten
 - ◆ *Putdata* erlaubt das Schreiben von Daten
 - ◆ *Adddata* erlaubt das Anhängen von Daten
- Entsprechende Rechte können pro Capability gesetzt werden

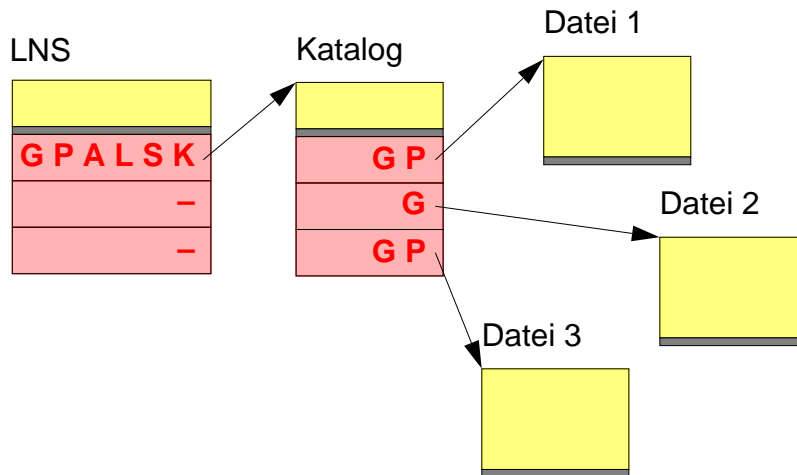


4.3 Zugriff auf Capabilities in Hydra

- Operationen auf der C-List:
 - ◆ *Load*: kopieren einer Capability aus der C-List eines Objekts in die C-List des LNS
 - ◆ *Store*: kopieren einer Capability aus der C-List des LNS in die C-List eines Objekts (dabei können Rechte maskiert werden)
 - ◆ *Append*: anfügen einer Capability in die C-List eines Objekts
 - ◆ *Delete*: löschen einer Capability aus der C-List eines Objekts
- Rechte:
 - ◆ **LOADRTS**: erlaubt Aufruf von *Load*
 - ◆ **STORTS**: erlaubt Aufruf von *Store*
 - ◆ **APPRTS**: erlaubt Aufruf von *Append*
 - ◆ **KILLRTS**: erlaubt Aufruf von *Delete*

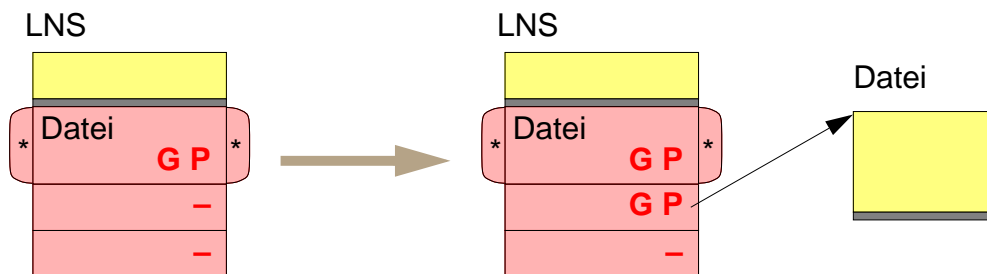
4.3 Zugriff auf Capabilities in Hydra (2)

- Beispiel: Implementierung von Katalogen
 - ◆ *Load* erlaubt das Auflösen von Namen (Aufrufer bekommt die Capability)
 - ◆ *Store* und *Append* erlauben das Hinzufügen von Dateien zum Katalog
 - ◆ *Delete* erlaubt das Austragen von Dateien aus dem Katalog



4.4 Objekterzeugung in Hydra

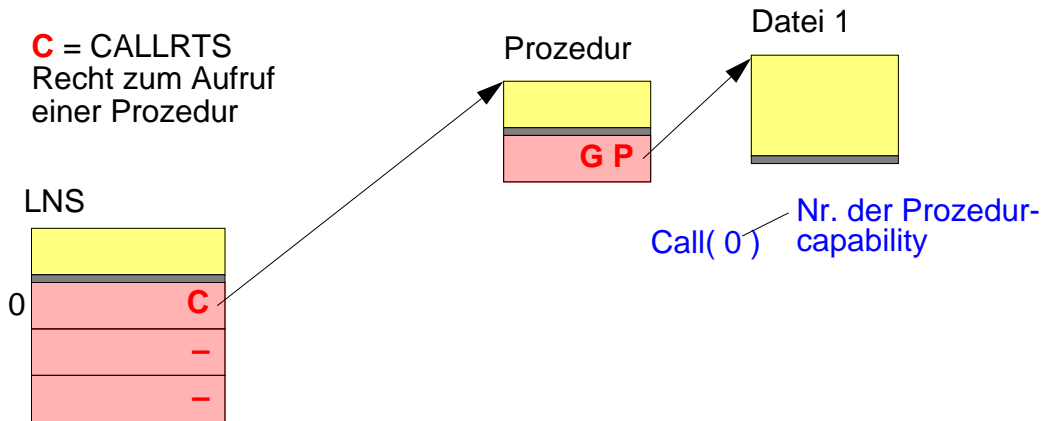
- Objekterzeugung über Erzeugungsschablonen (*Creation Templates*)
 - ◆ Erzeugungsschablone enthält den Typ des neu zu erzeugenden Objektes und eine Rechtemaske
 - ◆ nur die in der Maske angeschalteten Rechte werden dem Aufrufer in einer neuen Capability gegeben



4.5 Prozeduraufruf in Hydra

- Prozedur ist ein Objekt, aus dem beim Aufruf ein LNS des laufenden Prozesses erzeugt wird
 - ◆ neuer LNS wird aktueller Kontext (alte LNS stehen auf einem Stack; sie werden wieder aktiviert, wenn Prozedur zu Ende)

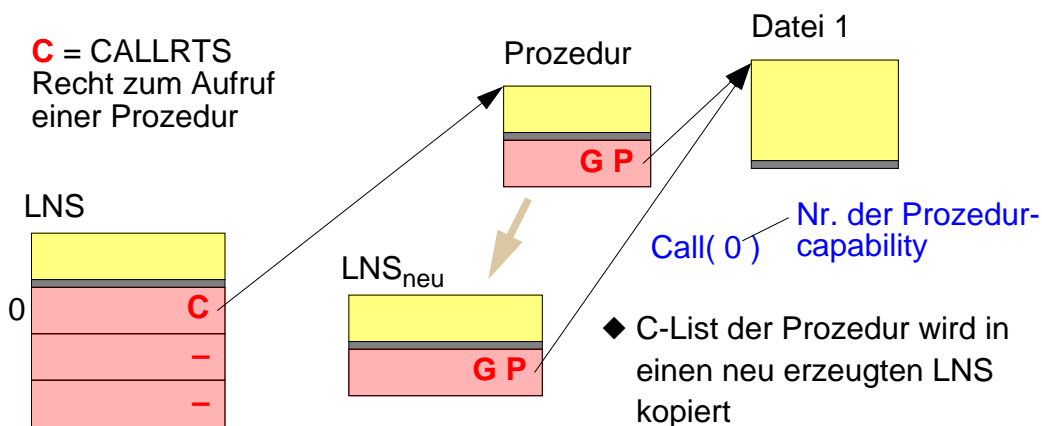
■ Aufruf einer Prozedur



4.5 Prozeduraufruf in Hydra

- Prozedur ist ein Objekt, aus dem beim Aufruf ein LNS des laufenden Prozesses erzeugt wird
 - ◆ neuer LNS wird aktueller Kontext (alte LNS stehen auf einem Stack; sie werden wieder aktiviert, wenn Prozedur zu Ende)

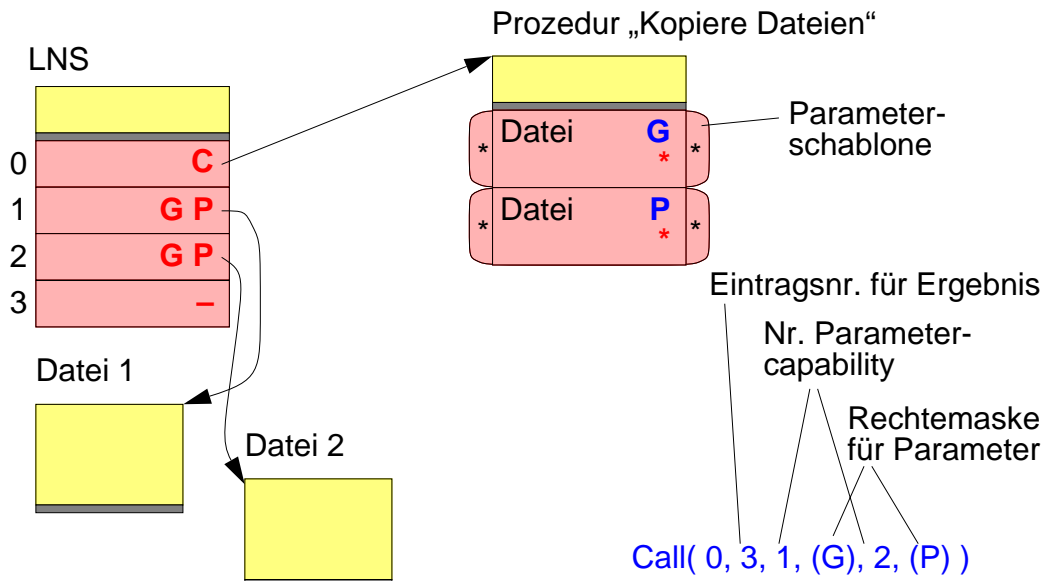
■ Aufruf einer Prozedur



4.5 Prozeduraufruf in Hydra (2)

■ Übergabe von Parametern

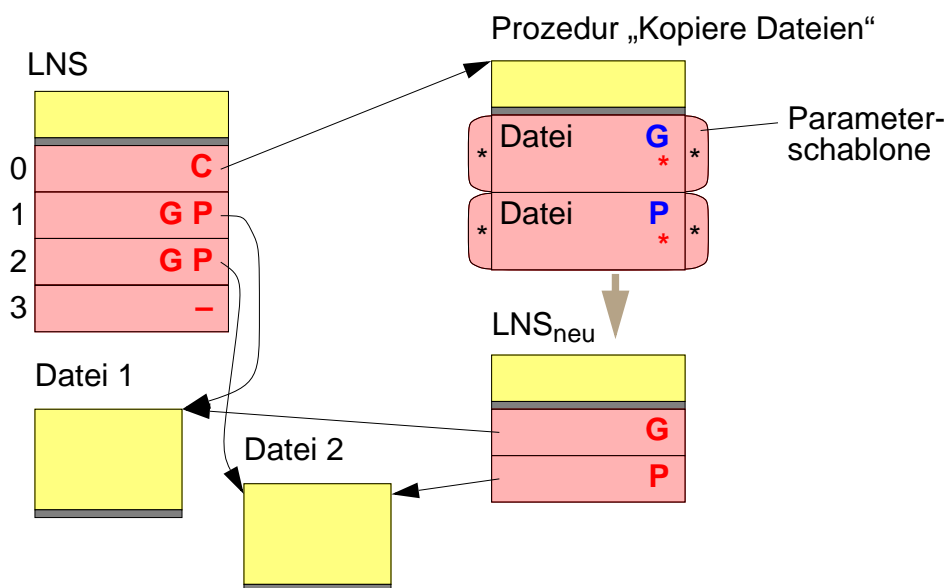
◆ Beispiel: Prozedur zum Kopieren von Dateiinhalten



4.5 Prozeduraufruf in Hydra (3)

■ Übergabe von Parametern

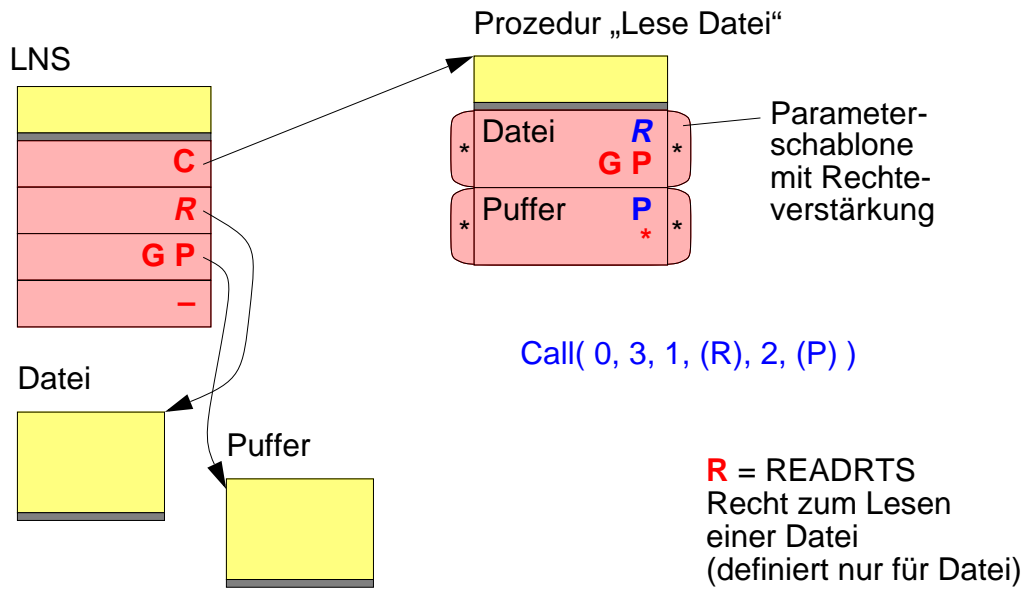
◆ Beispiel: Prozedur zum Kopieren von Dateiinhalten



4.5 Prozeduraufruf in Hydra (3)

■ Verstärken von Rechten

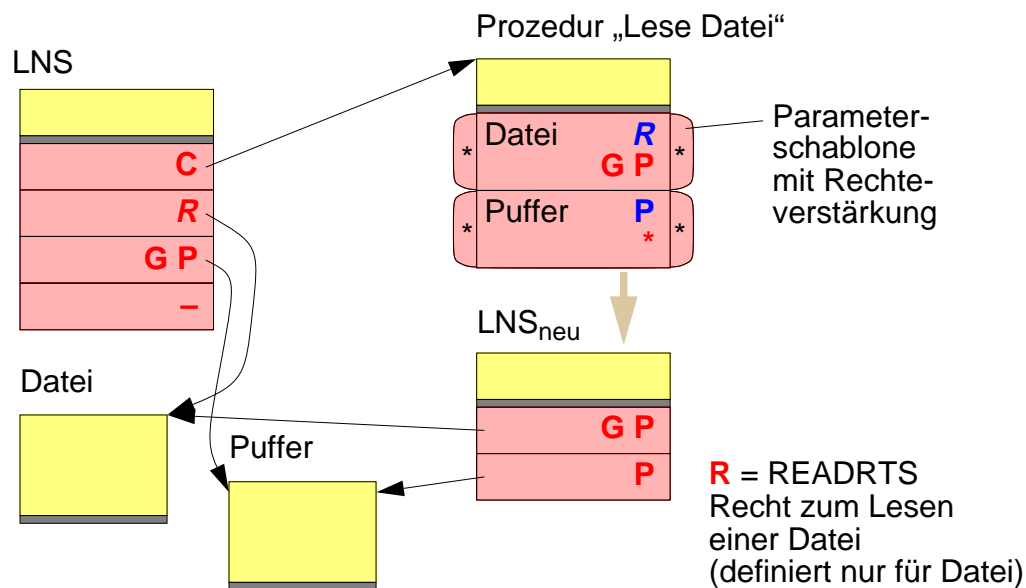
- ◆ Beispiel: Prozedur zum Lesen von Dateiinhalten in einen Puffer



4.5 Prozeduraufruf in Hydra (3)

■ Verstärken von Rechten

- ◆ Beispiel: Prozedur zum Lesen von Dateiinhalten in einen Puffer



4.6 Problem: Gegenseitiges Misstrauen

- Aufrufer misstraut einer Prozedur
 - ◆ Aufrufer möchte der Prozedur nur soviel Rechte einräumen wie nötig
- Aufgerufene Prozedur misstraut dem Aufrufer
 - ◆ Aufrufer soll nur soviel Rechte und Zugang bekommen wie erforderlich
- ★ Hydra Prozeduraufruf unterstützt diese Forderungen direkt
 - ◆ Aufrufer übergibt Capabilities, die nötig sind
 - ◆ Aufrufer kann Rechte bei der Übergabe maskieren und damit ausschalten
 - ◆ Aufrufer erhält nur Zugang zu einem definierten Ergebnis
 - ◆ Prozedur kann eigene Capabilities besitzen, die einem LNS zur Verfügung stehen und die dem Aufrufer verborgen bleiben können

4.6 Problem: Gegenseitiges Misstrauen (2)

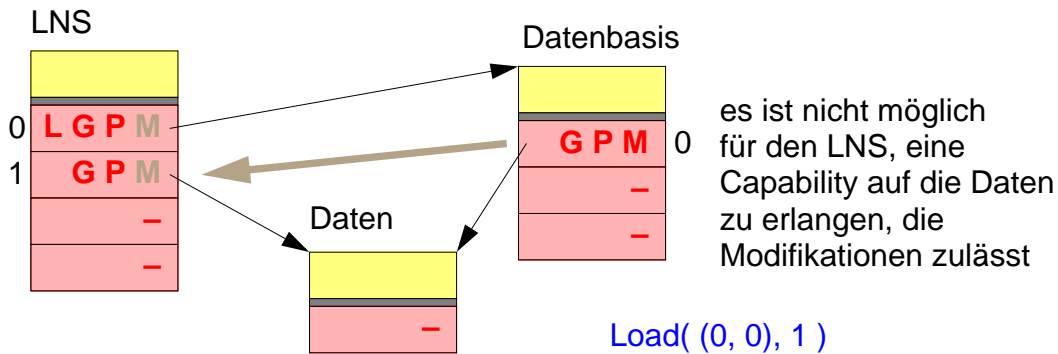
- ▲ Rechteverstärkung als Sicherheitslücke?
 - ◆ Verstärkungsschablone wird nur an vertrauenswürdige Prozeduren ausgegeben und kann nicht einfach erzeugt werden

4.7 Problem: Modifikationen

- Aufrufer möchte Modifikationen an und über Parameter ausschließen
 - ◆ eine Prozedur soll nichts verändern können
- Wegnehmen der entsprechenden Rechte langt nicht
 - ◆ Prozedur kann lesend zu neuen Capabilities gelangen und über diese Änderungen vornehmen (Transitivität)
 - ◆ Rechteverstärkung könnte angewandt werden

4.7 Problem: Modifikation (2)

- ★ Einführung des Modifikationsrechts **MDFYRTS**
 - ◆ für alle modifizierenden Operationen an Datenbereichen und C-Lists muss zusätzlich das Modifikationsrecht für das Objekt vorhanden sein
 - ◆ Modifikationsrecht wird automatisch gelöscht, wenn eine Capability über einen Pfad geladen wird, auf dem eine der Capabilities kein Modifikationsrecht besitzt
 - ◆ Modifikationsrecht kann nicht über Rechteverstärkung erlangt werden



4.7 Problem: Modifikation (3)

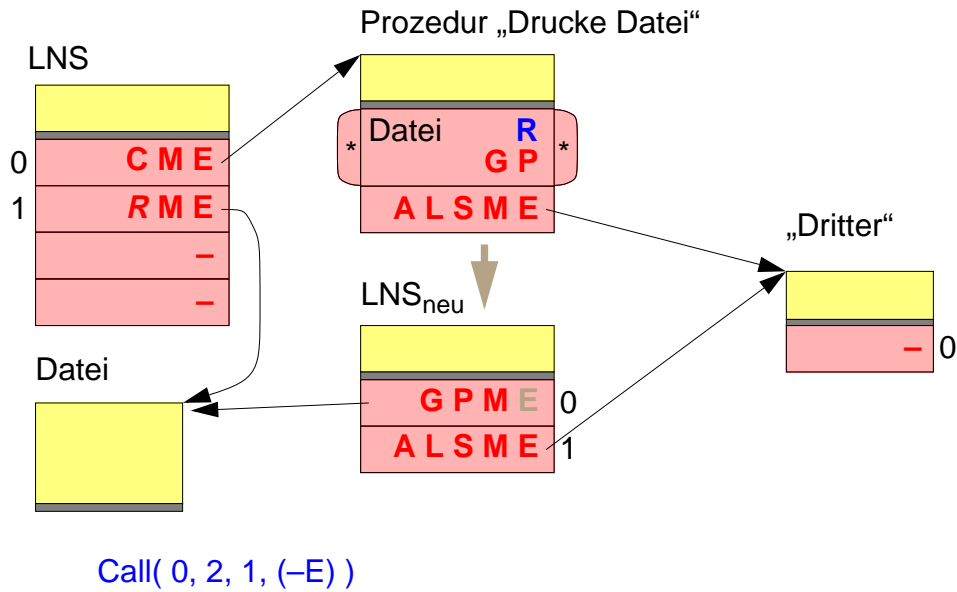
- Parameterübergabe
 - ◆ Wegnahme des Modifikationsrecht bei Parametern stellt sicher, dass die aufgerufene Prozedur keinerlei Veränderungen beim Aufrufer durchführen kann

4.8 Problem: Ausbreitung von Capabilities

- Aufrufer will verhindern, dass eine übergebene Capability vom Aufgerufenen an einen Dritten weitergegeben wird (*Propagation Problem*)
 - ◆ Beispiel: Prozedur „Drucken“ soll niemandem eine Referenz auf die zu druckenden Daten weitergeben können

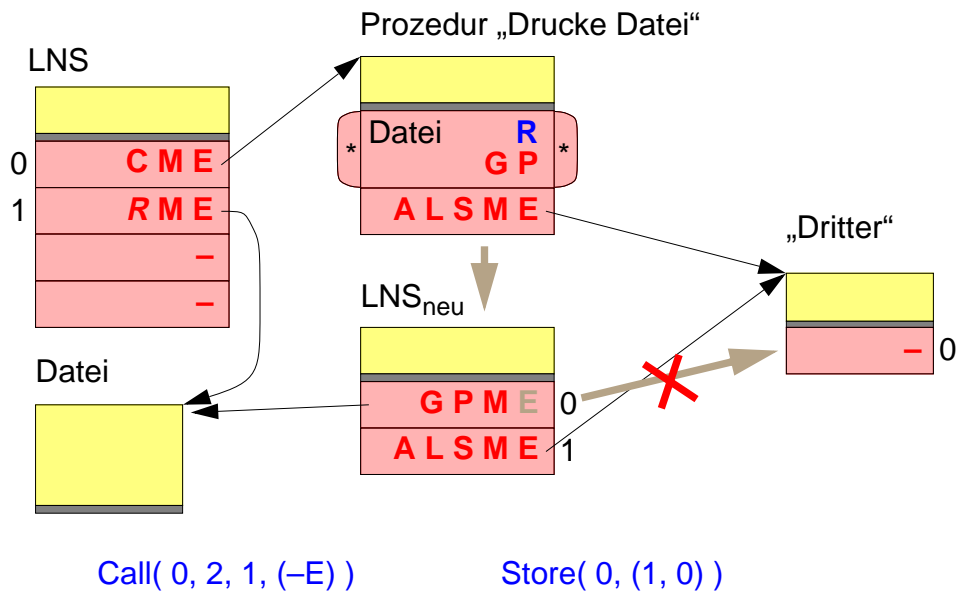
4.8 Problem: Ausbreitung von Capabilities (3)

- Versuchte Weitergabe einer Capability an einen Dritten



4.8 Problem: Ausbreitung von Capabilities (3)

- Versuchte Weitergabe einer Capability an einen Dritten



4.9 Problem: Aufbewahrung von Capabilities

- Aufrufer möchte sicher sein, dass Aufgerufener keine Capabilities nach der Bearbeitung des Aufrufs zurückbehalten kann (*Conservation Problem*)
- ★ Environment-Recht zusammen mit dem Aufrufmechanismus genügt
 - ◆ Aufgerufener kann Capability ohne ENVRTS nicht weitergeben und folglich nicht abspeichern
 - ◆ der LNS des Aufrufs wird mit Beendigung des Aufrufs vernichtet, so dass die übergebenen Capabilities nicht zurückbehalten werden können
 - ◆ ENVRTS wirkt transitiv, so dass auch die über eine Parameter-Capability gewonnenen Capabilities nicht weitergegeben werden können

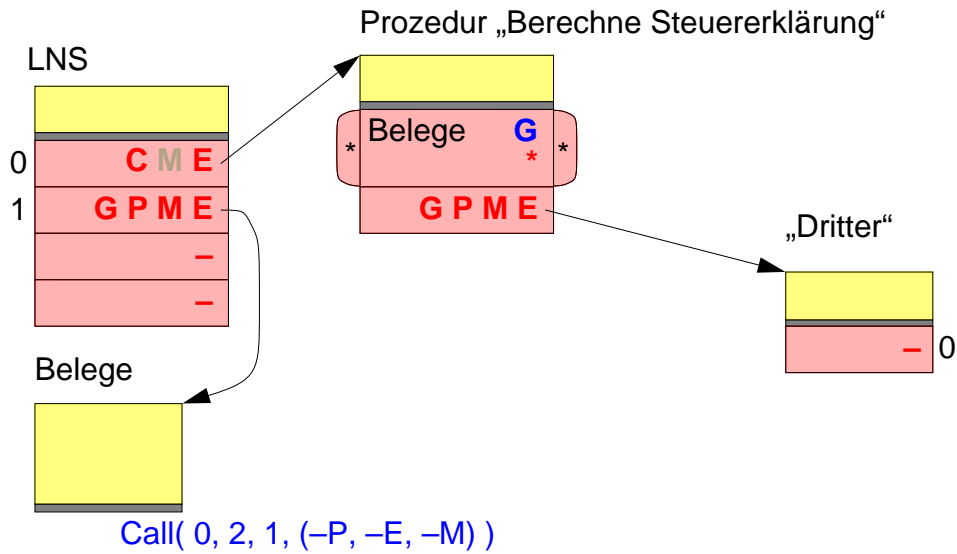
4.10 Problem: Informationsflussbegrenzung

- Aufrufer möchte die Verbreitung von Informationen aus übergebenen Parametern einschränken (*Confinement Problem*)
 - ◆ selektiv: bestimmte Informationen sollen nicht nach außen gelangen
 - ◆ global: gar keine Informationen sollen nach außen gelangen
 - ◆ ENVRTS ist nicht ausreichend, da Prozedur den Dateninhalt von Parameterobjekten kopieren könnte (ENVRTS wirkt nur auf die Weitergabe von Capabilities)
- Hydra realisiert nur globale Informationsflussbegrenzung
- ★ Modifikationsrecht auf der Prozedur-Capability
 - ◆ wenn kein Modifikationsrecht vorhanden ist, werden bei allen in den LNS übernommenen Capabilities die Modifikationsrechte ausgeschaltet (gilt jedoch nicht für Parameter)

4.10 Problem: Informationsflussbegrenzung (2)

■ Beispiel: Prozedur zur Steuerberechnung

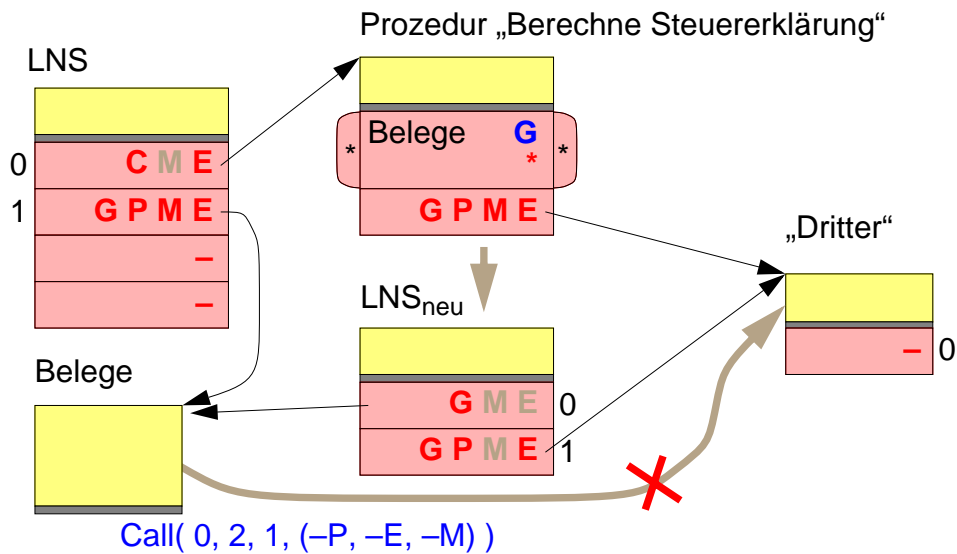
- ◆ die übergebenen Beleg- und Buchhaltungsdaten sollen nicht weitergegeben werden können



4.10 Problem: Informationsflussbegrenzung (2)

■ Beispiel: Prozedur zur Steuerberechnung

- ◆ die übergebenen Beleg- und Buchhaltungsdaten sollen nicht weitergegeben werden können



4.11 Problem: Initialisierung

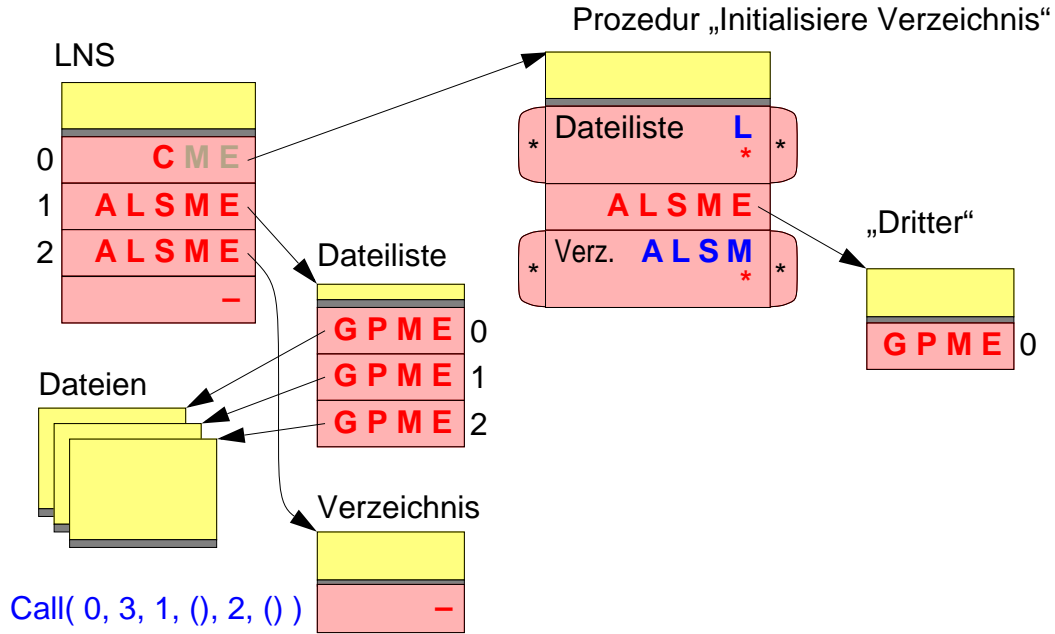
- Initialisierung von Objekten durch Prozeduren
 - ◆ Übergabe eines Objekts und verschiedener Capabilities, mit denen das Objekt initialisiert werden soll
 - ◆ Problem: Parameter-Capabilities müssen Environment-Recht besitzen (sonst ist das zu initialisierende Objekt nicht arbeitsfähig), gleichzeitig soll aber die Ausbreitung solcher Capabilities eingeschränkt werden
 - Lösung: Wegnahme des Modifikationsrechts auf der Prozedurcapability
 - ◆ Problem: Es muss verhindert werden, dass die Prozedur in das zu initialisierende Objekt eigene oder fremde Capabilities einsetzt, so dass es später Einfluss auf das zu initialisierende Objekt nehmen kann
- Beispiel: Prozedur zur Initialisierung eines Katalogs bekommt Capabilities auf die entsprechenden Dateien
 - ◆ es soll sichergestellt werden, dass Prozedur keine eigenen Dateicapabilities in den Katalog einfügt

4.11 Problem Initialisierung (2)

- ★ Environment-Recht auf der Prozedur-Capability
 - ◆ wenn kein Environment-Recht vorhanden ist, werden bei allen in den LNS übernommenen Capabilities die Environment-Rechte ausgeschaltet (gilt jedoch nicht für Parameter)
 - ◆ durch das fehlende Environment-Recht können alle bereits vorhandenen Capabilities nicht in das zu initialisierende Objekt gespeichert werden

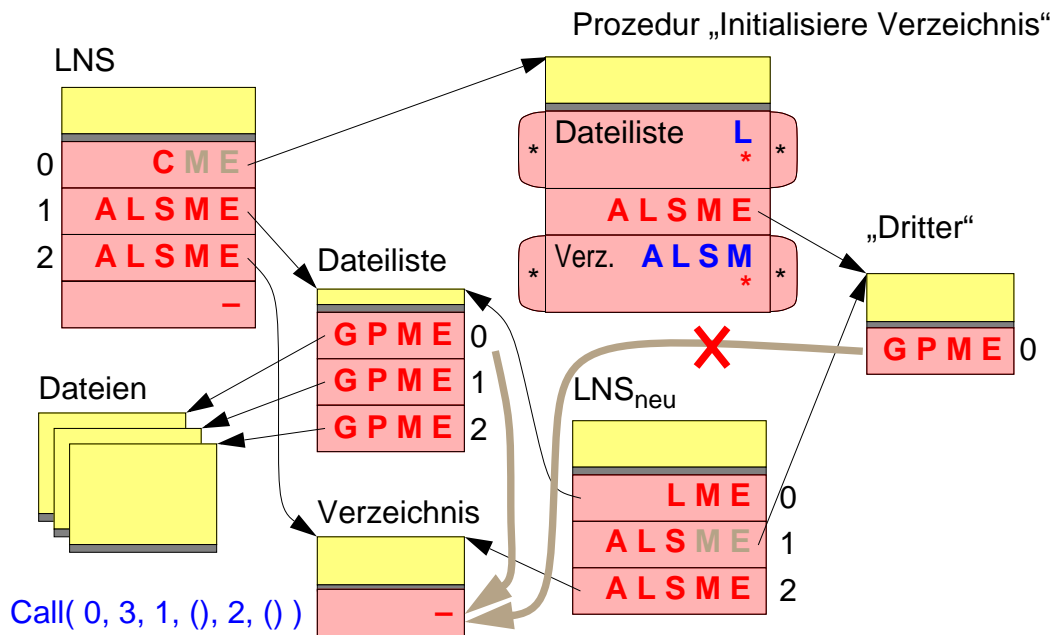
4.11 Problem: Initialisierung (3)

- Beispiel: Initialisierung eines Verzeichnis



4.11 Problem: Initialisierung (3)

- Beispiel: Initialisierung eines Verzeichnis

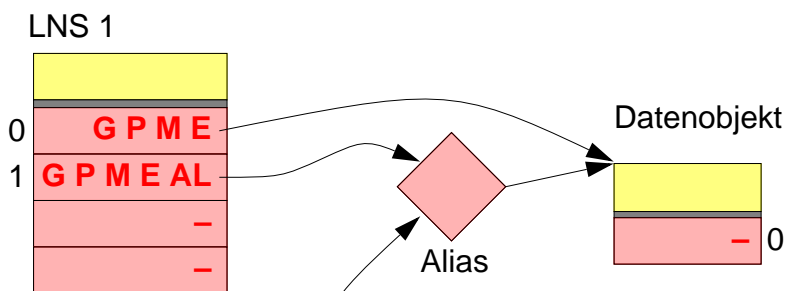


4.12 Rückruf von Capabilities

- Anwender möchte ausgegebene Capabilities für ungültig erklären
 - ◆ sofortiger Rückruf — Rückruf nach einiger Zeit erst wirksam
 - ◆ dauerhafter Rückruf — Rückruf nur zeitlich begrenzt wirksam
 - ◆ selektiver Rückruf — Rückruf für alle Benutzer eines Objekts
 - ◆ partieller Rückruf — Rückruf aller Rechte an einem Objekt
 - ◆ Recht zum Rückruf; Rückruf des Rückrufrechts
- ★ Hydra setzt sogenannte Aliase ein
 - ◆ Alias ist eine Indirektionsstufe zu Capabilities
 - ◆ Statt auf ein Objekt können Capabilities auf Aliase verweisen und diese wiederum auf andere Aliase oder schließlich auf das eigentliche Objekt
 - ◆ Verbindung vom Alias zum Objekt kann gelöst werden: Fehler beim Zugriff
 - ◆ Recht zum Lösen der Verbindung **ALLYRTS** (engl. *ally* = verbünden)

4.12 Rückruf von Capabilities (2)

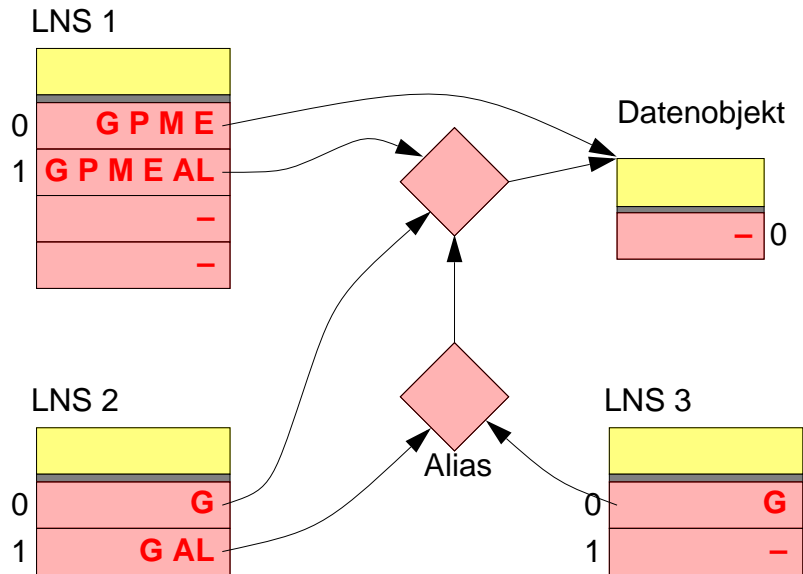
- Beispiel: Weitergabe einer rückrufbaren Capability



- ◆ LNS 1 kann Verbindung des Alias lösen (ALLYRTS nötig)
- ◆ LNS 2 kann dann nicht mehr zugreifen
- ◆ Wiederverbindung möglich (Originalcapability nötig)
- ◆ Aliase können hintereinander auftreten

4.12 Rückruf von Capabilities (3)

■ Beispiel: Aliasketten

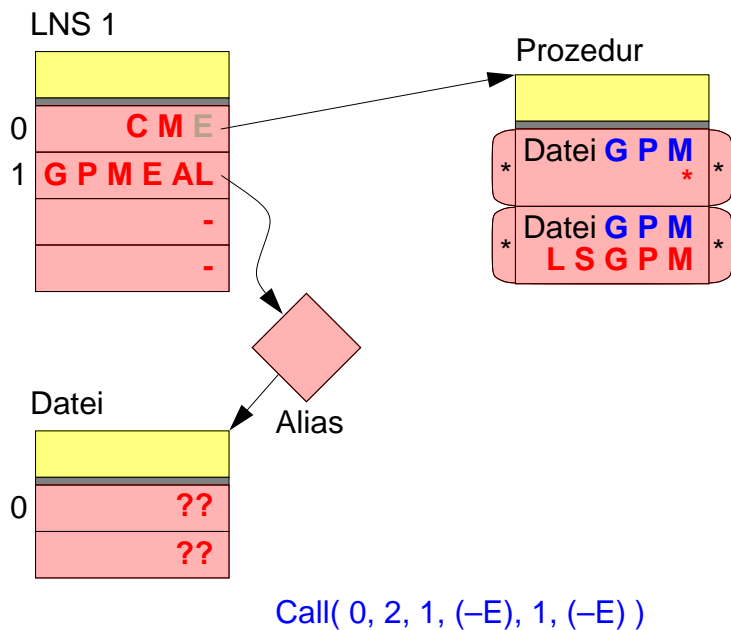


4.12 Rückruf von Capabilities (4)

- ▲ Problem: Rückruf während der Bearbeitung eines Objekts
 - ◆ inkonsistente Zustände möglich
- ★ Lösung in Hydra
 - ◆ Parameter-Capabilities, die durch eine rechteverstärkende Parameterschablone angenommen werden, zeigen auf das Originalobjekt
- ▲ Nachteil
 - ◆ nicht vertrauenswürdige Prozeduren können rückruffreie Capability erlangen
 - ◆ Problem fällt in die selbe Kategorie wie rechteverstärkende Parameterschablonen an sich

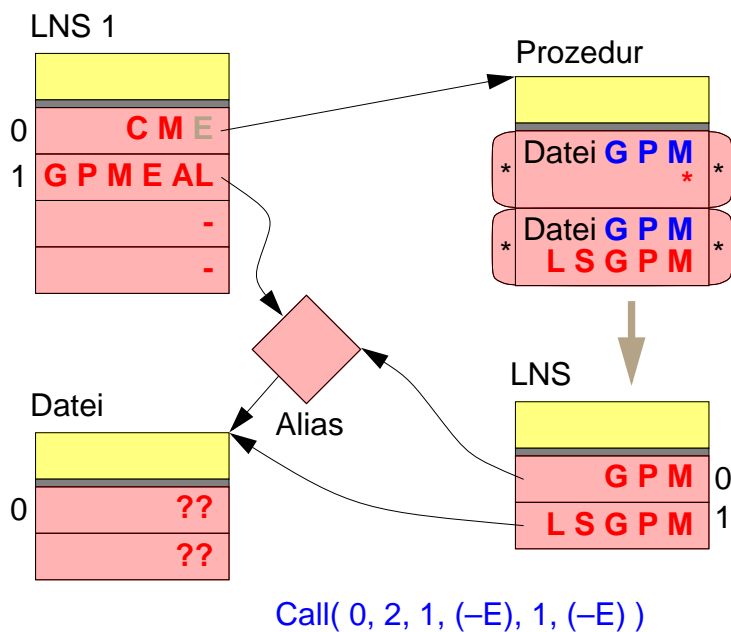
4.12 Rückruf von Capabilities (5)

■ Beispiel:



4.12 Rückruf von Capabilities (5)

■ Beispiel:

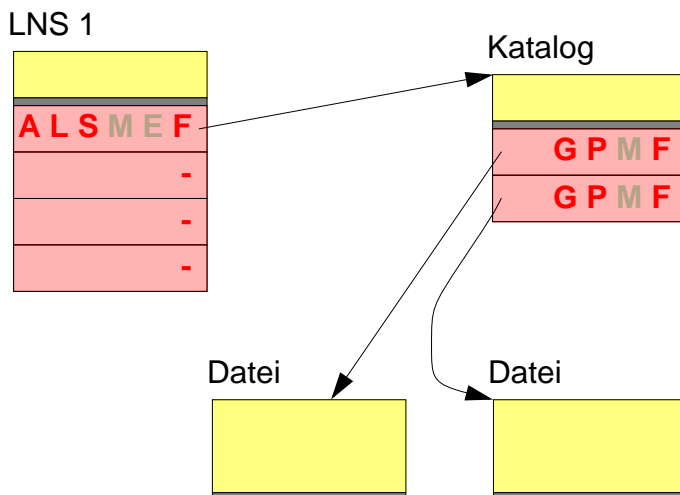


4.13 Garantierter Zugriff

- Schutz vor Rückruf
 - ◆ Modifizierende Benutzer eines Objekts
 - kooperierende Benutzer: Rückruf keine Gefahr
 - nicht kooperierende Benutzer: Rückruf nötig
 - ◆ Benutzer eines Objekts ohne modifizierende Zugriffe
 - Aufruf von Prozeduren ist unkritisch, da Aufrufe nicht rückrufbar sind
 - lesende Zugriffe auf Objekte sind kritisch:
 - Verhindern des Rückrufs ist jedoch nicht ausreichend
 - Daten im Objekt könnten gelöscht oder verfälscht werden
 - Capabilities im Objekt oder deren Rechte könnten entfernt werden
- ★ Hydra führt das Einfrierrecht (*Freeze right FRZRTS*) ein
 - ◆ Einfrieren nimmt Modifikationsrecht weg
 - ◆ ein Objekt kann nur gefroren werden, wenn alle Capabilities in der C-List bereits das Einfrierrecht haben

4.13 Garantierter Zugriff (2)

- Beispiel:



4.14 Bewertung von Hydra

- Hydra demonstrierte die Beherrschbarkeit einer ganzen Reihen von Sicherheitsproblemen
 - ◆ Ergebnisse flossen in eine ganze Reihe von Systemen
 - ◆ reine Capability-basierte Systeme haben sich jedoch nie durchgesetzt
- Hydras Probleme
 - ◆ lagen im wesentlichen nicht am Capability-Mechanismus
 - ◆ es gabe keine vernünftigen Editoren und Compiler
 - ◆ Hardware besaß keine Spezialhardware zur Unterstützung von Paging

5 Kryptographische Maßnahmen

- Verschlüsseln und Entschlüsseln vertraulicher Daten
 - ◆ aus den verschlüsselten Daten soll die Originalinformation nur mit Hilfe eines Schlüssels restauriert werden können
 - ◆ Schlüssel bleibt geheim
- Authentisierung
 - ◆ Empfänger kann verifizieren, wer der Absender ist
- Sicherer Kanal
 - ◆ gesendete Informationen können nicht gefälscht und verfälscht werden
 - ◆ nur der adressierte Empfänger kann die Informationen lesen
 - ◆ Empfänger kann den Absender authentisieren