8 Datensicherheit, Zugriffsschutz

8.1 Erläuterung der Problemstellung

8.2 Maßnahmen auf der logischen Ebene

8.3 Maßnahmen auf der physikalischen Ebene

8.4 Betriebssystemstrukturen

8.5 Kryptographische Maßnahmen

8.1 Erläuterung der Problemstellung

Umgebung: Naturkatastrophen

Böswillige Angriffe

Unbefugter Zutritt zu den Räumen des RZ

Sicherheitsbeauftragter:

Fehlerhafte Spezifikation der Sicherheitspolitik

Systemsoftware:

Versagen der Schutzmechanismen

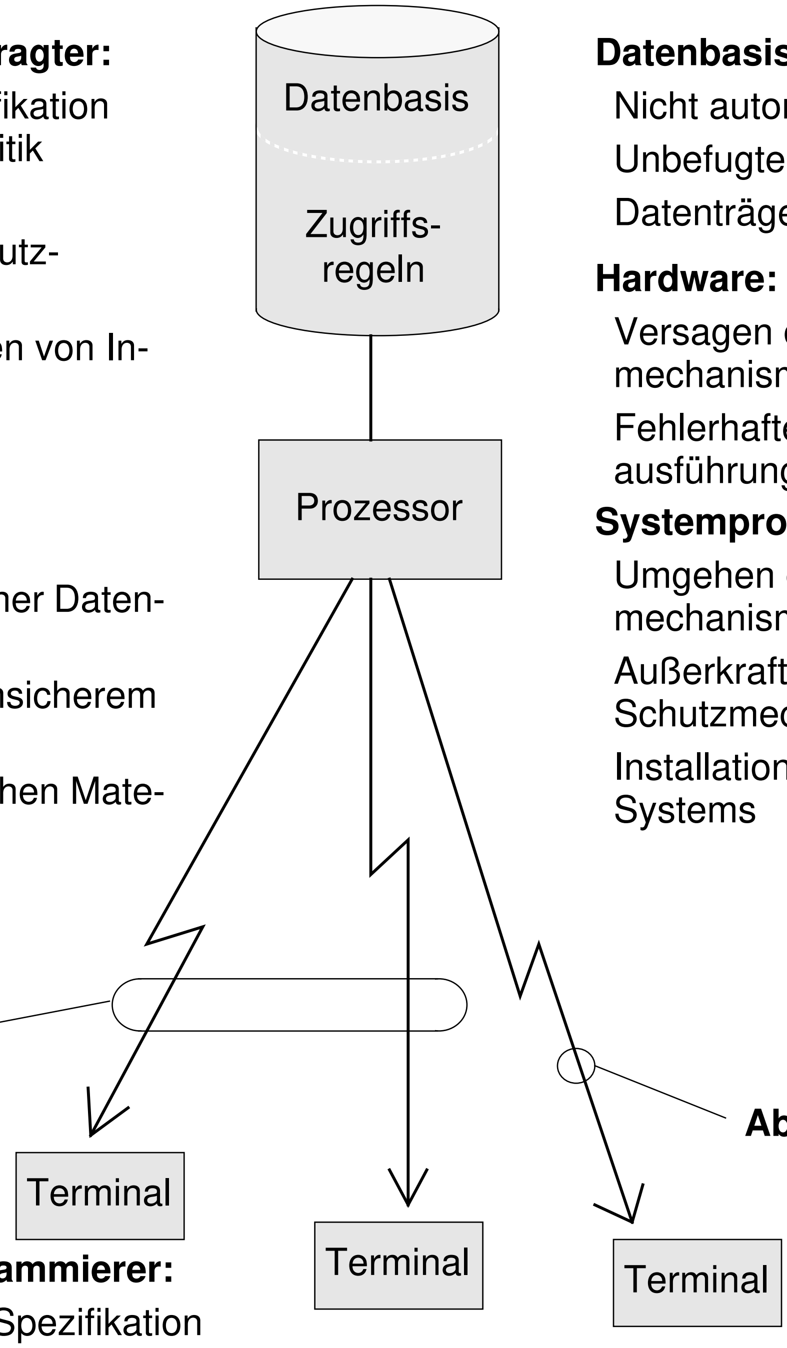
Durchsickernlassen von Information

Operator:

Kopieren vertraulicher Datenträger

Initialisierung mit unsicherem Zustand

Diebstahl vertraulichen Materials



Konspiration

Abhören

Datenbasis:

Nicht autorisierter Zugriff

Unbefugtes Kopieren

Datenträgerdiebstahl

Hardware:

Versagen der Schutzmechanismen

Fehlerhafte Befehlsausführung

Systemprogrammierer:

Umgehen der Schutzmechanismen

Außerkraftsetzen der Schutzmechanismen

Installation eines unsicheren Systems

Anwendungsprogrammierer:

Nichteinhalten der Spezifikation

Terminal:

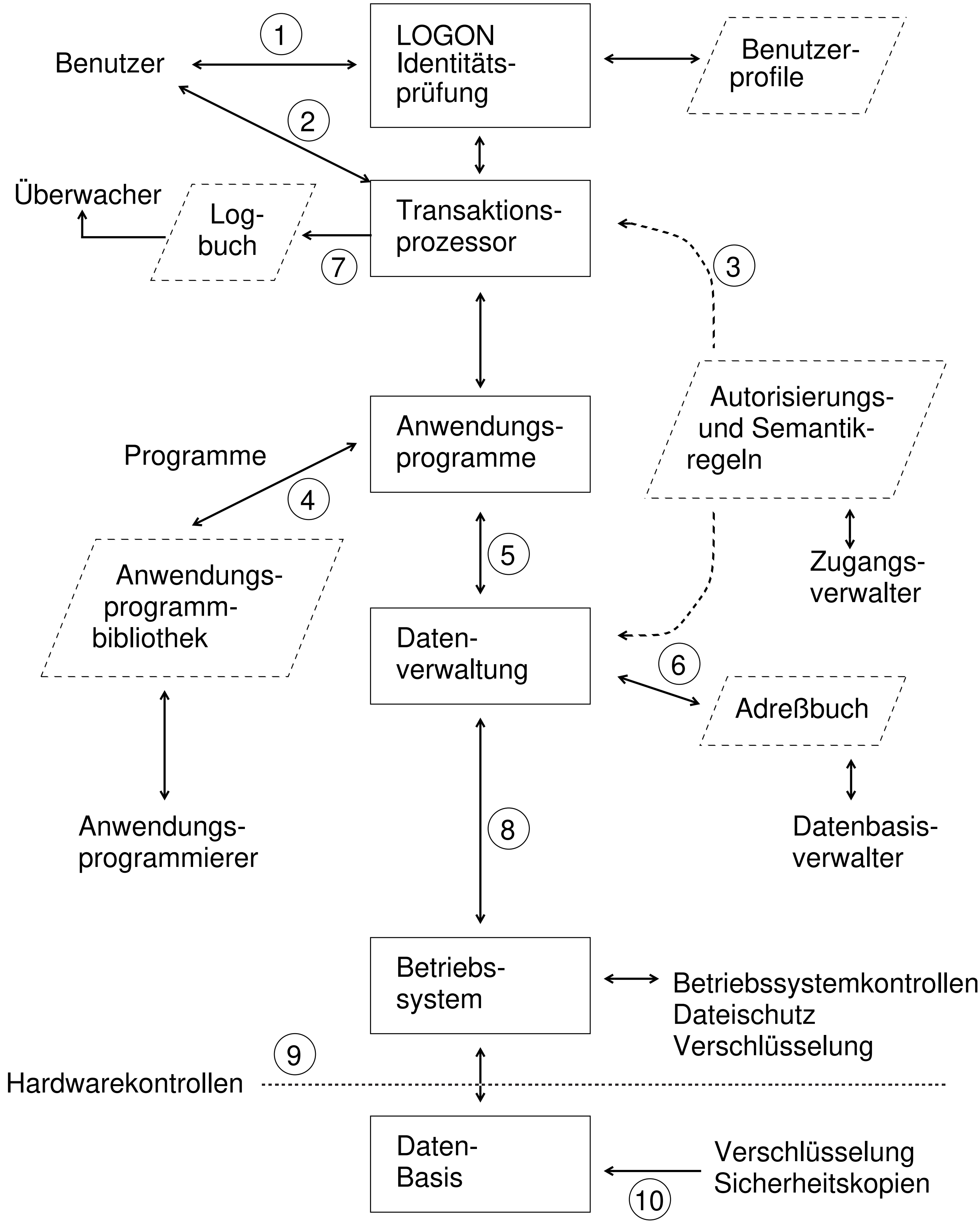
Ungeschützter Zugang

Benutzer:

Falsche Identifikation

Illegale Weitergabe von Information

8.2 Maßnahmen auf der logischen Ebene



Nr.	(eindeutiger) Name	Geschlecht	Fach	Stellung	Gehalt (KDM)	Spenden (DM)
1	Albrecht	m	Inf.	Prof.	60	150
2	Bergner	m	Math.	Prof.	45	300
3	Cäsar	w	Math.	Prof.	75	600
4	David	w	Inf.	Prof.	45	150
5	Engel	m	Stat.	Prof.	54	0
6	Frech	w	Stat.	Prof.	66	450
7	Groß	m	Inf.	Angest.	30	60
8	Hausner	m	Math.	Prof.	54	1500
9	Ibel	w	Inf.	Stud.	9	30
10	Jost	m	Stat.	Angest.	60	45
11	Knapp	w	Math.	Prof.	75	300
12	Ludwig	m	Inf.	Stud.	9	0

Fragen 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

Schlußfolgerung:

Spende(’David’) = 306 * 5 - 345 * 4
= 1530 - 1380
= 150

Personaldatensatz

- Name
- Abteilung
- Personalnummer
- Lohn- oder Gehaltsgruppe

Personaldateien

- O_1: Personaldaten der leitenden Angestellten
- O_2: Personaldaten der Angestellten
- O_3: Personaldaten der Arbeiter

Prozeduren

- P_1: Lesen von Pers.-Nr. und Lohn-/Gehaltsgr. aus O_1
- P_2: Lesen von Pers.-Nr. und Lohn-/Gehaltsgr. aus O_2 und O_3
- Q : Lesen von Name, Abteilung und Pers.-Nr.

Benutzer

- S_1: Leiter des Personalbüros
  Besitzer aller Dateien und Prozeduren
  Lese- und Schreibrecht für alle Dateien
  Aufrufrecht für alle Prozeduren
- S_2: Sachbearbeiter für leitende Angest., stellvertr. Leiter
  Lese- und Schreibrecht für O_2 und O_3
  Aufrufrecht für P_1
- S_3: Sachbearbeiter für Angestellte und Arbeiter
  Aufrufrecht für P_2
- S_4: Poststelle
  Aufrufrecht für Q

Schutzmatrix

	O_1	O_2	O_3	P_1	P_2	Q
S_1	B, L, S	B, L, S	B, L, S	B, A	B, A	B, A
S_2		L, S	L, S	A		
S_3					A	
S_4						A
P_1	L					
P_2		L	L			
Q	L	L	L			

B S ist Besitzer des Objektes O

L S darf aus O lesen mit der Operation read

S S darf O modifizieren mit Operation write

A S darf O als Operation ausführen

Das Schutzmodell von Bell und La Padula

- Sicherheitsgrade

  Jedem Subjekt und jedem Objekt wird ein **Sicherheitsgrad** zugeordnet bestehend aus einer **Geheimhaltungsstufe** a und einer **Schutzkategorie** b.

  - Die Geheimhaltungsstufe ist Element einer systemspezifischen, vollständig geordneten Menge von Geheimhaltungsstufen.
  - Die Schutzkategorie ist Teilmenge einer systemspezifischen Menge von Sachgebieten.

  Es ist

  $$(a;b) \leq (a';b') \Leftrightarrow (a < a' \wedge b \subseteq b') \vee (a \leq a' \wedge b \subset b')$$

- Sicherheitseigenschaft

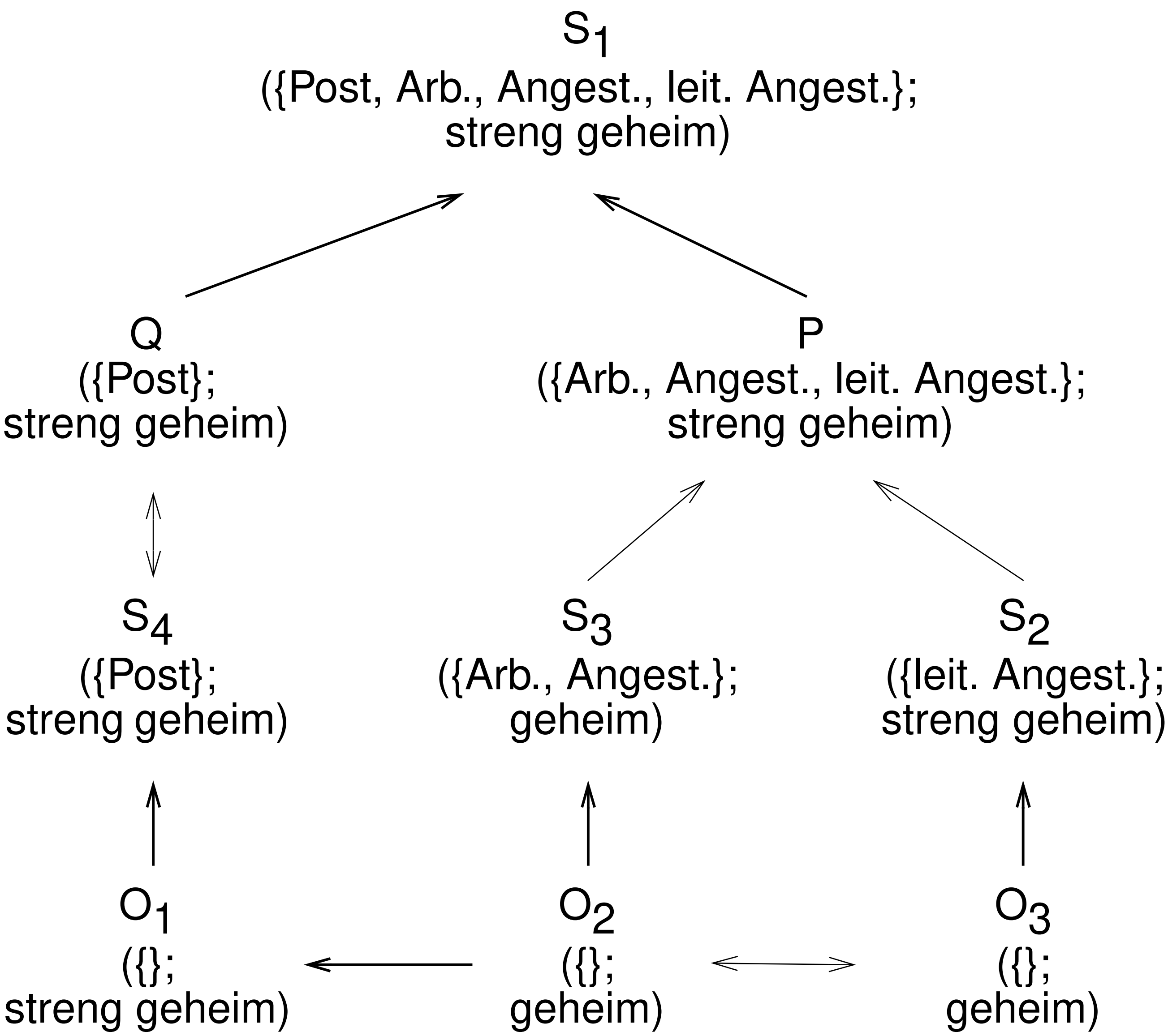
  Ein Subjekt kann nur auf Objekte mit gleichem oder niedrigerem Sicherheitsgrad lesend oder schreibend zugreifen.

- *-Eigenschaft

  Ein Subjekt kann

  nur dann gleichzeitig zu einem Objekt A lesenden und zu einem Objekt B schreibenden Zugriff haben,

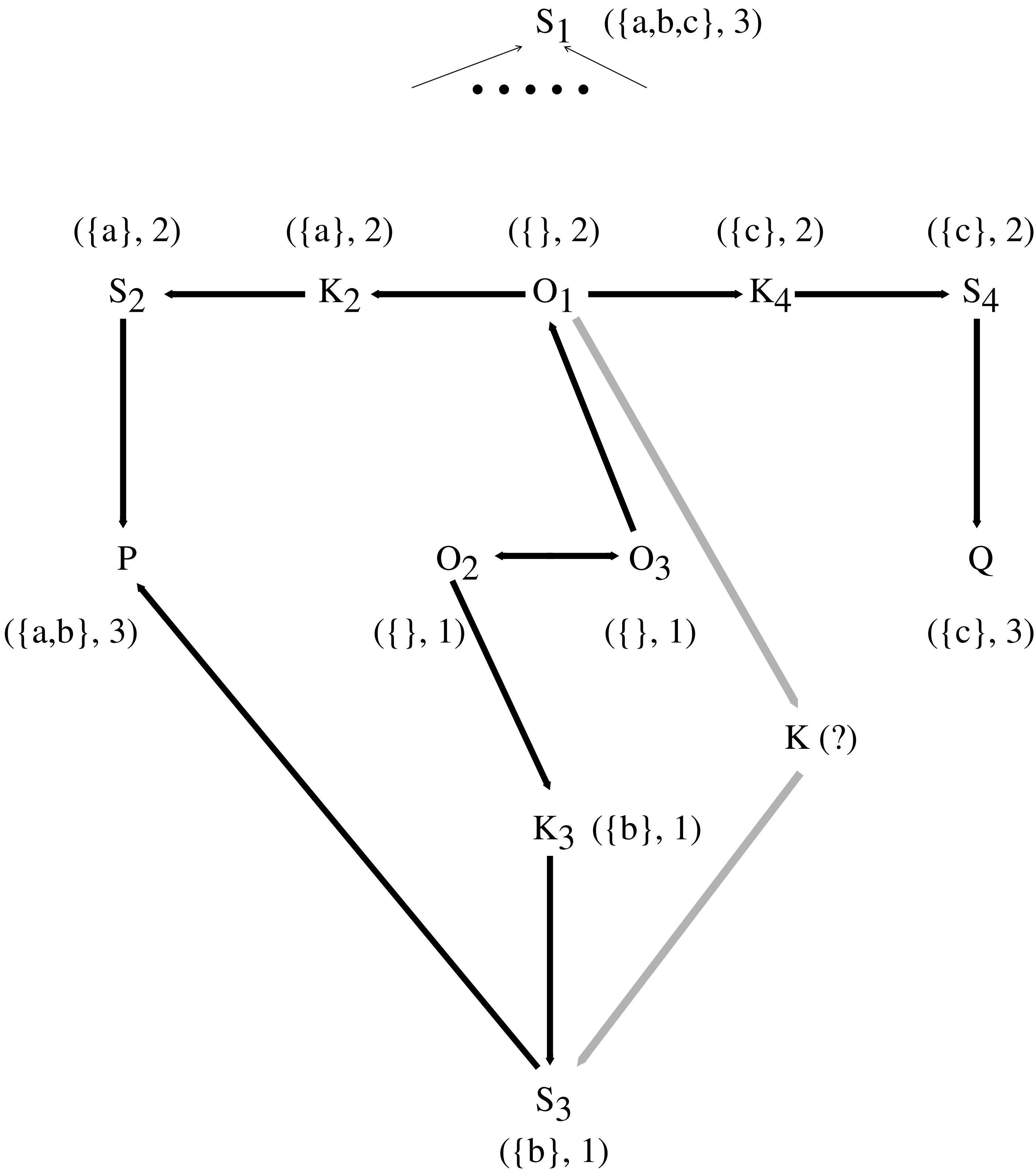
  wenn B den gleichen oder einen höheren Sicherheitsgrad besitzt als A.



Informationsflußkontrollgraph für das Beispiel 'Personaldatei'.

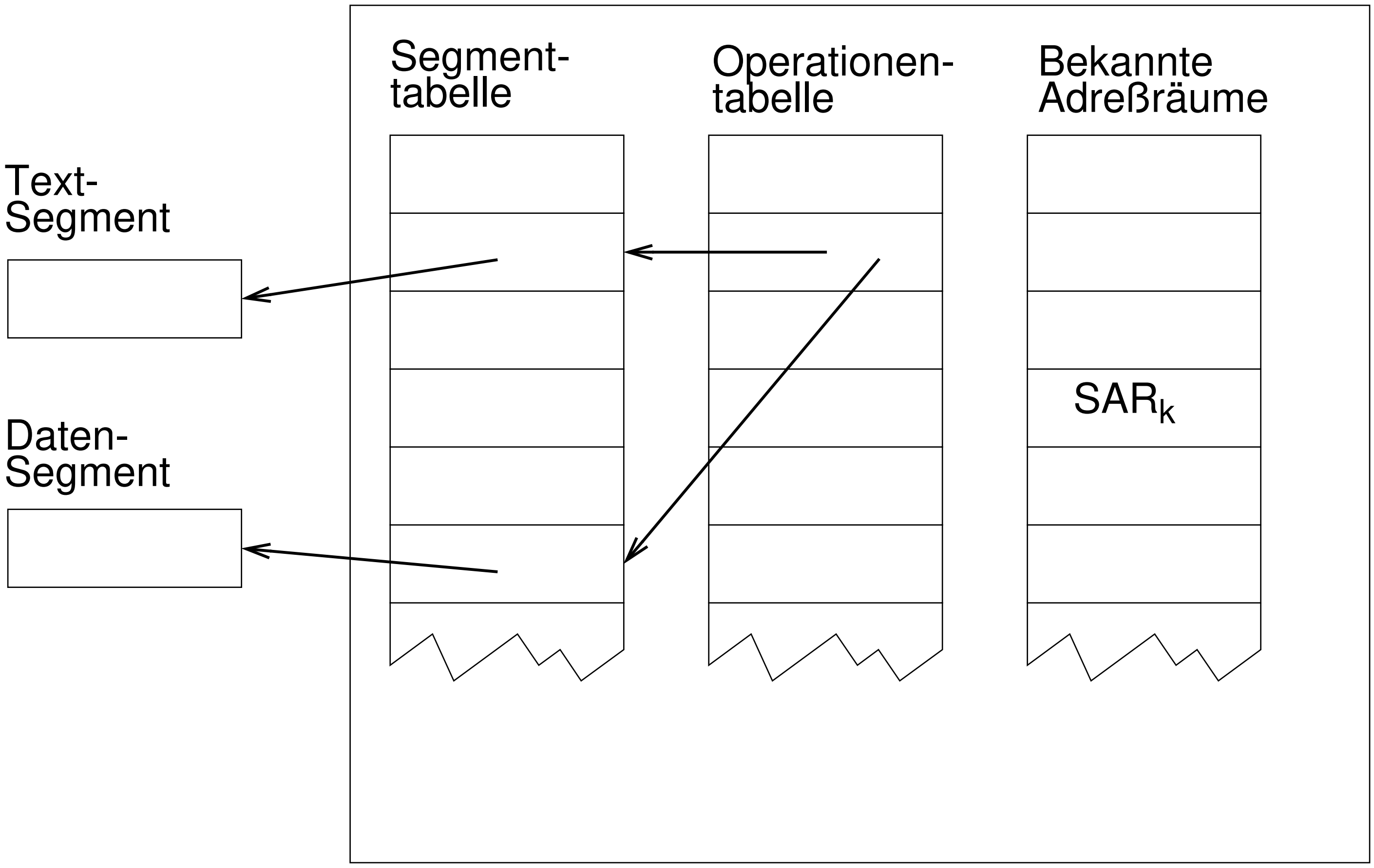
Die Sicherheitsgrade haben die Gestalt (a; b) mit a $\subseteq$ {*Arbeiter*, *Angestellte*, *leitende Angestellte*, *Post*}, b $\in$ {*geheim*, *streng geheim*} und *geheim* < *streng geheim*.

Da es in diesem Falle nicht notwendig ist, P_1 und P_2 zu unterscheiden, sind sie durch P ersetzt.


S1 ({a,b,c}, 3)
.....
({a}, 2)
({a}, 2)
({}, 2)
({c}, 2)
({c}, 2)
S2
K2
O1
K4
S4
P
O2
O3
Q
({a,b}, 3)
({}, 1)
({}, 1)
({c}, 3)
K (?)
K3 ({b}, 1)
S3
({b}, 1)

8.3 Maßnahmen auf der physikalischen Ebene

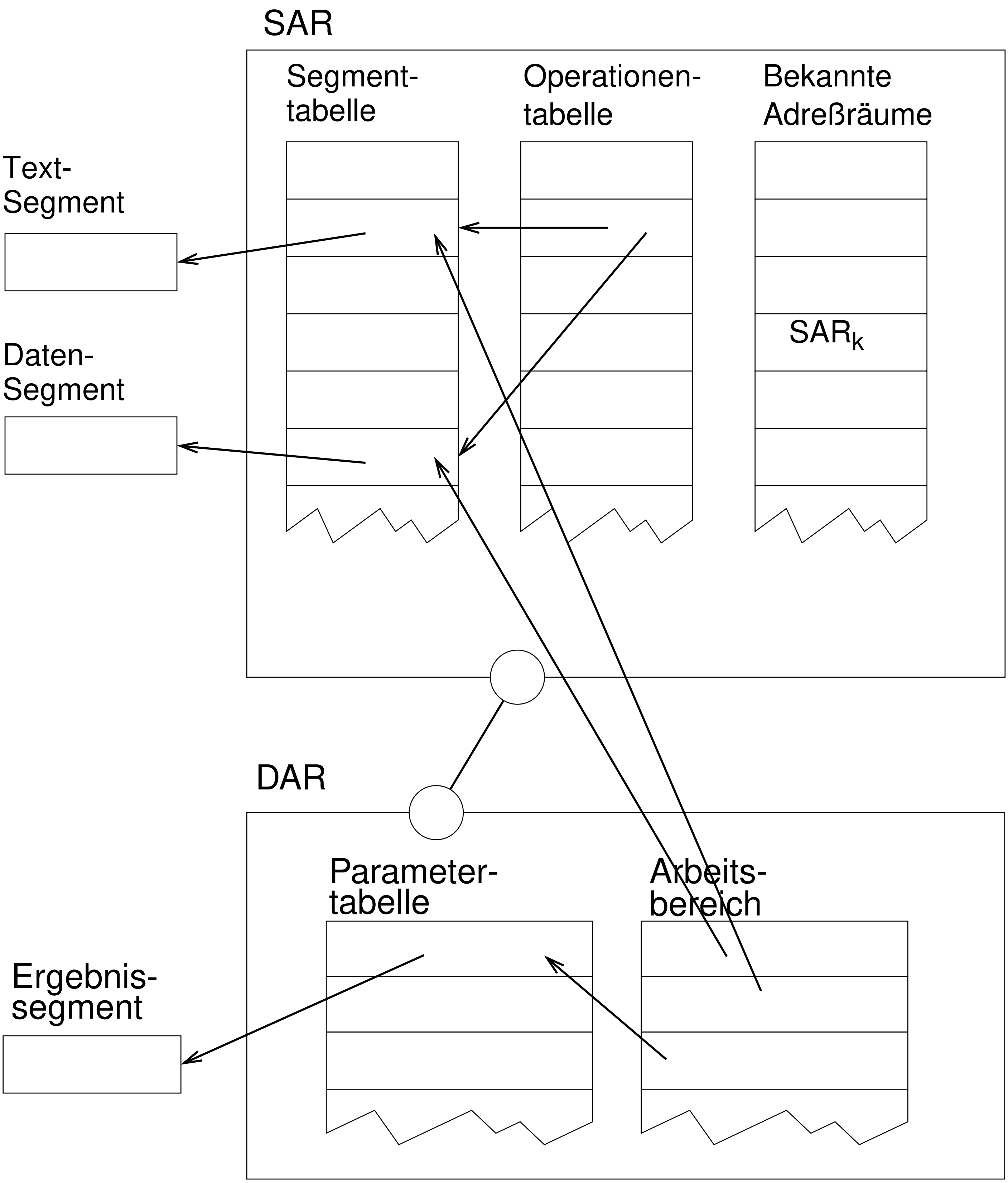
Statischer Adreßraum (SAR)



Angaben für ASCALL

1. Index eines bekannten Adreßraums
2. Index einer Operation
3. Parameterliste in Form von Indizes für Segment- und Parametertabelle
4. Index für Segment- oder Parametertabelle zur Übergabe des Ergebnissegmentes

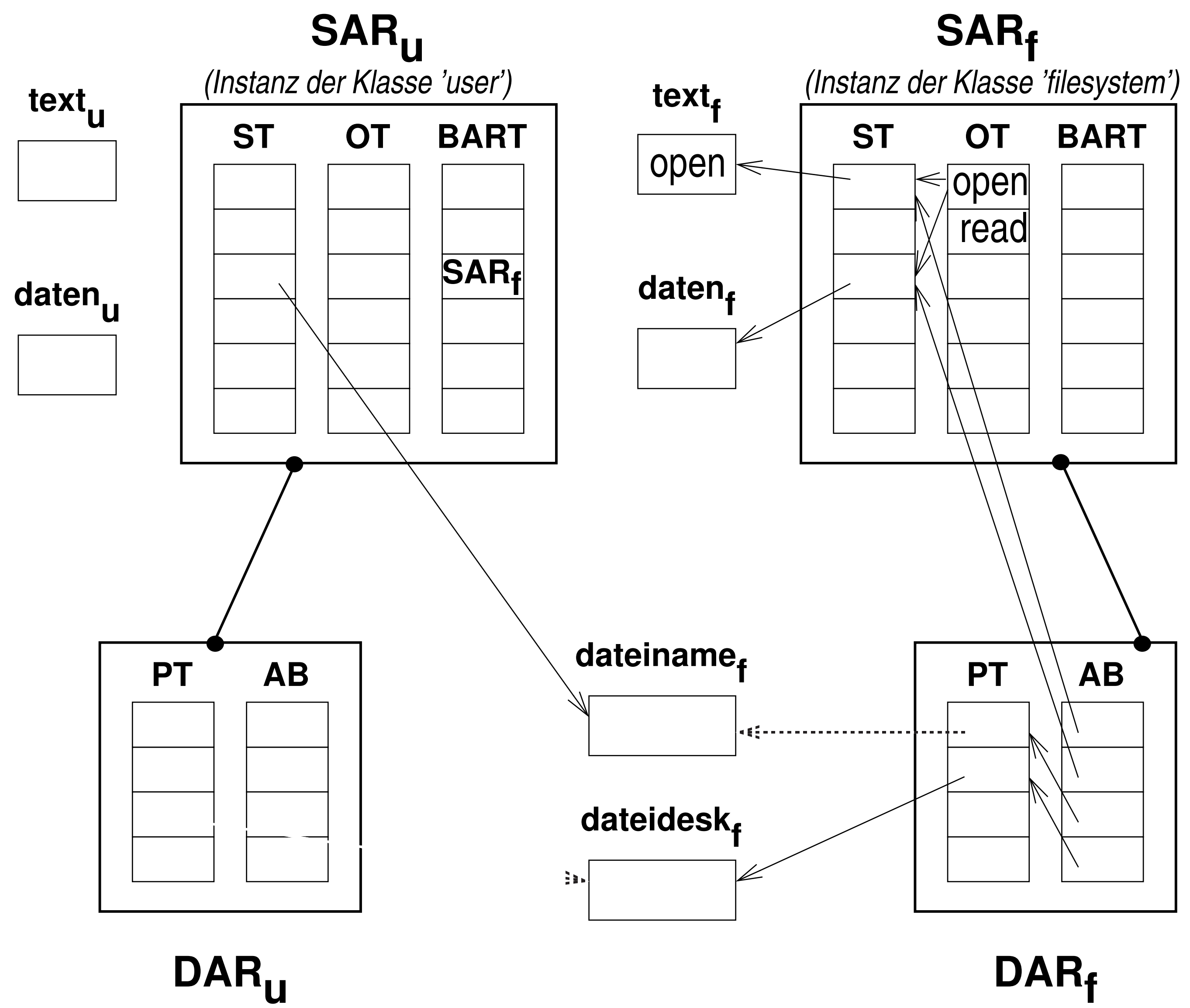
Dynamischer Adreßraum (DAR)

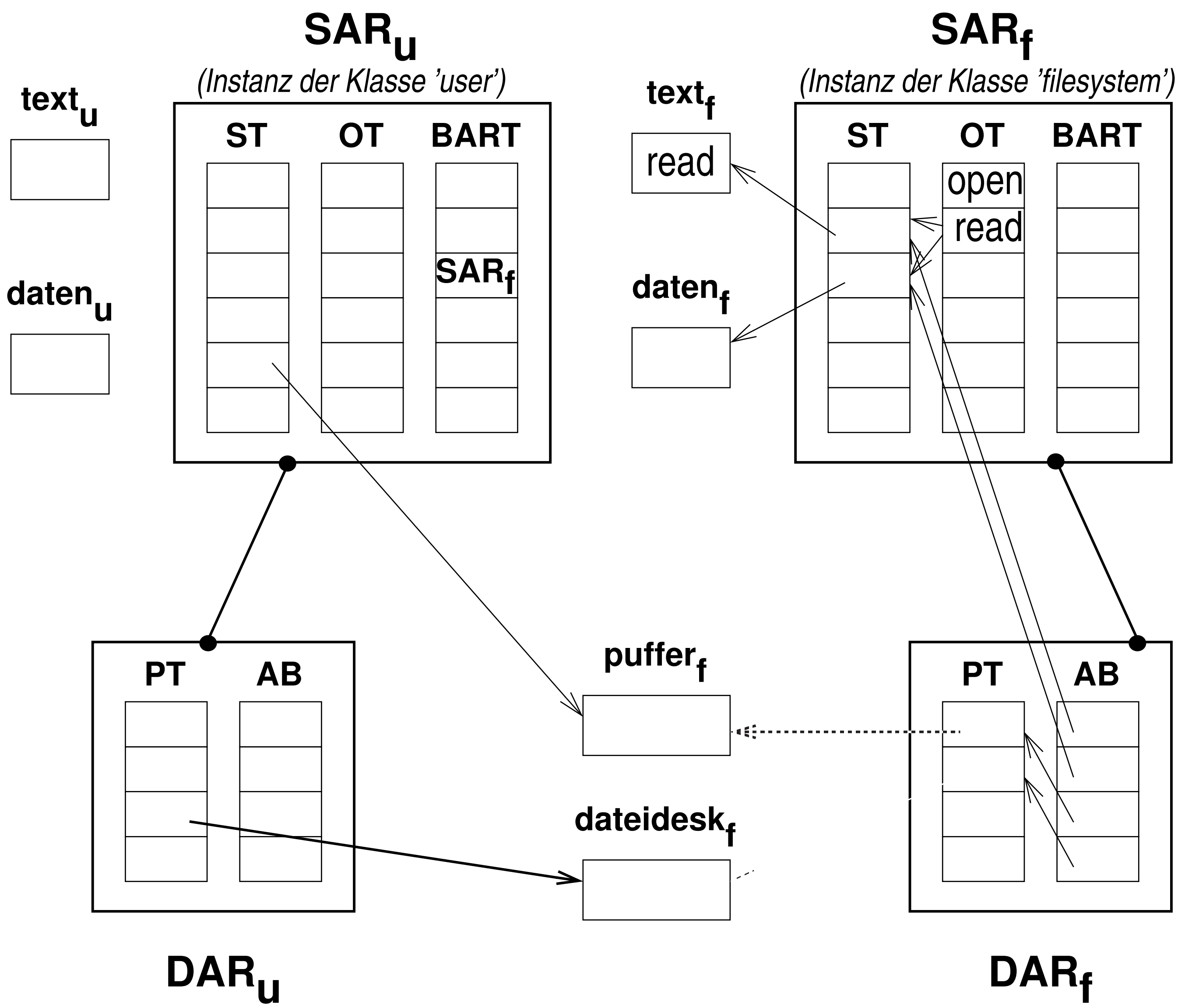


```
MODULE user(ds: &filesystem)
DECLARATIONS
   dateidesk: dateideskriptor;
   dateiname: SEQ OF CHAR;
   ...
OPERATIONS
   ...
      EFFECTS
         dateidesk=ds->open('dateiname)
   ...
```

```
MODULE filesystem;
DECLARATIONS
   ...
OPERATIONS
   ...
   open(fname)
            ->fdes: dateideskriptor
   ...
```

ASCALL(BART[3], 1, (ST[3]), PT[3])





Lesen aus einer Datei, für die ein Verweis auf den Dateideskriptor in PT[3] und ein Verweis auf einen Puffer in ST[5] steht:

ASCALL(BART[3], 2, (PT[3], ST[5]), NIL)

Grundlegende Schutzmechanismen

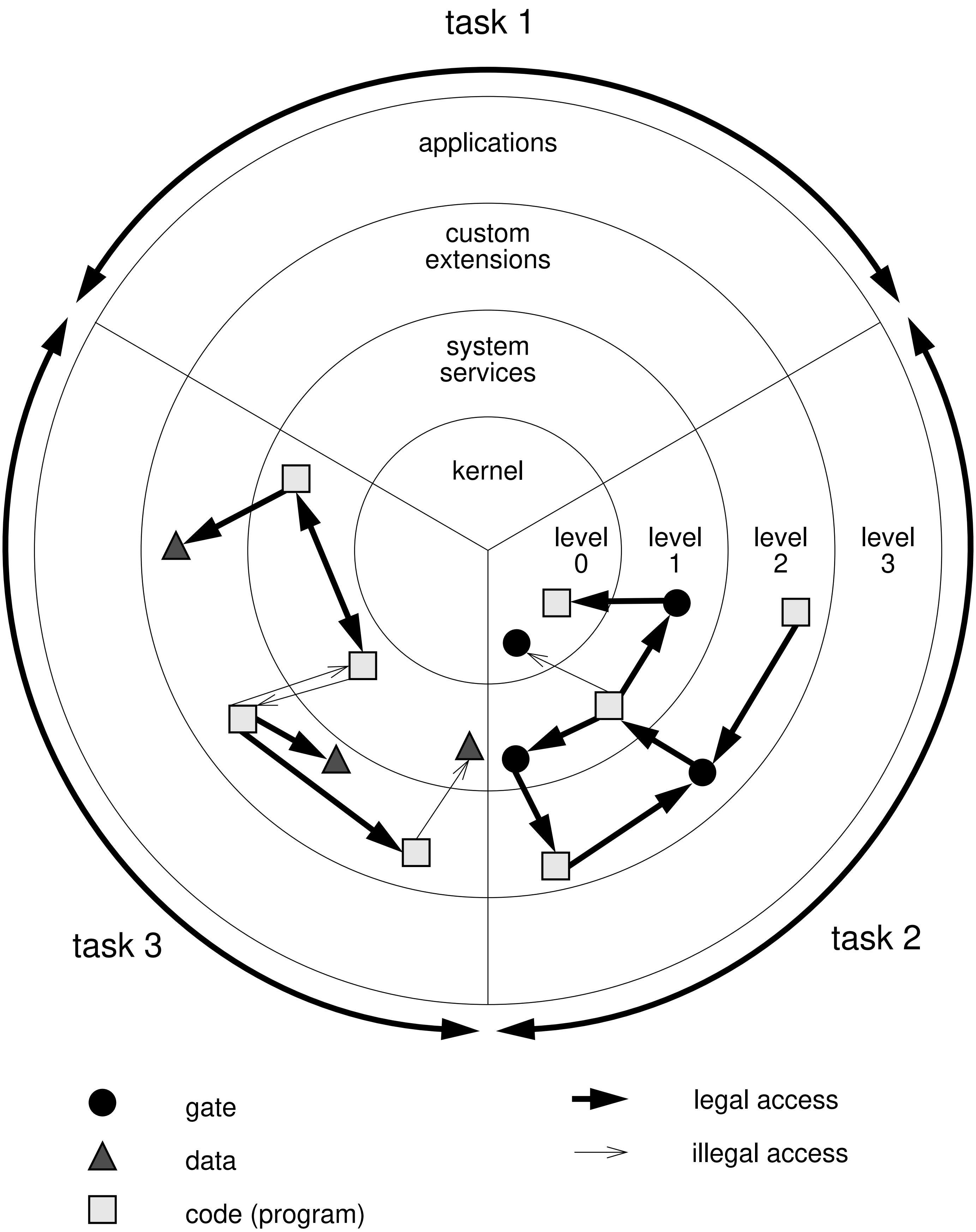
(1) Typ-Prüfung
(Datensegmente, Text(Befehls)segmente, ...)

(2) Grenzen-Prüfung
(Segmentgrenzen)

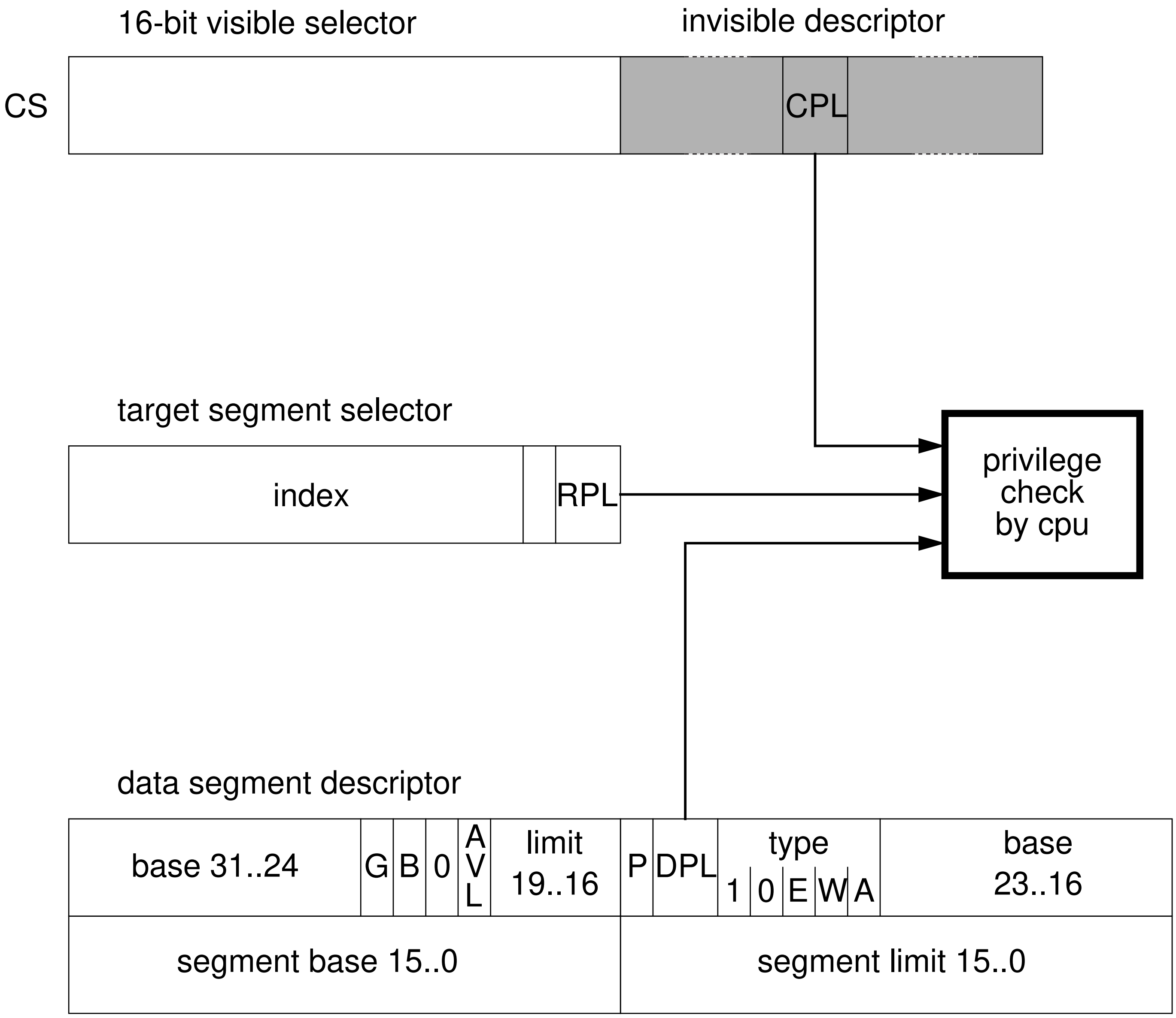
(3) Einschränkung des adressierbaren Bereichs
(prozeßspezifische Segmenttabellen)

(4) Einschränkung der Methoden-Einsprungpunkte
(Tore oder ’gates’)

(5) Einschränkung der ausführbaren Befehle
(privilegierter Befehle)


task 1
applications
custom
extensions
system
services
kernel
level
0
level
1
level
2
level
3
task 3
task 2
gate
data
code (program)
legal access
illegal access

Datenzugriff

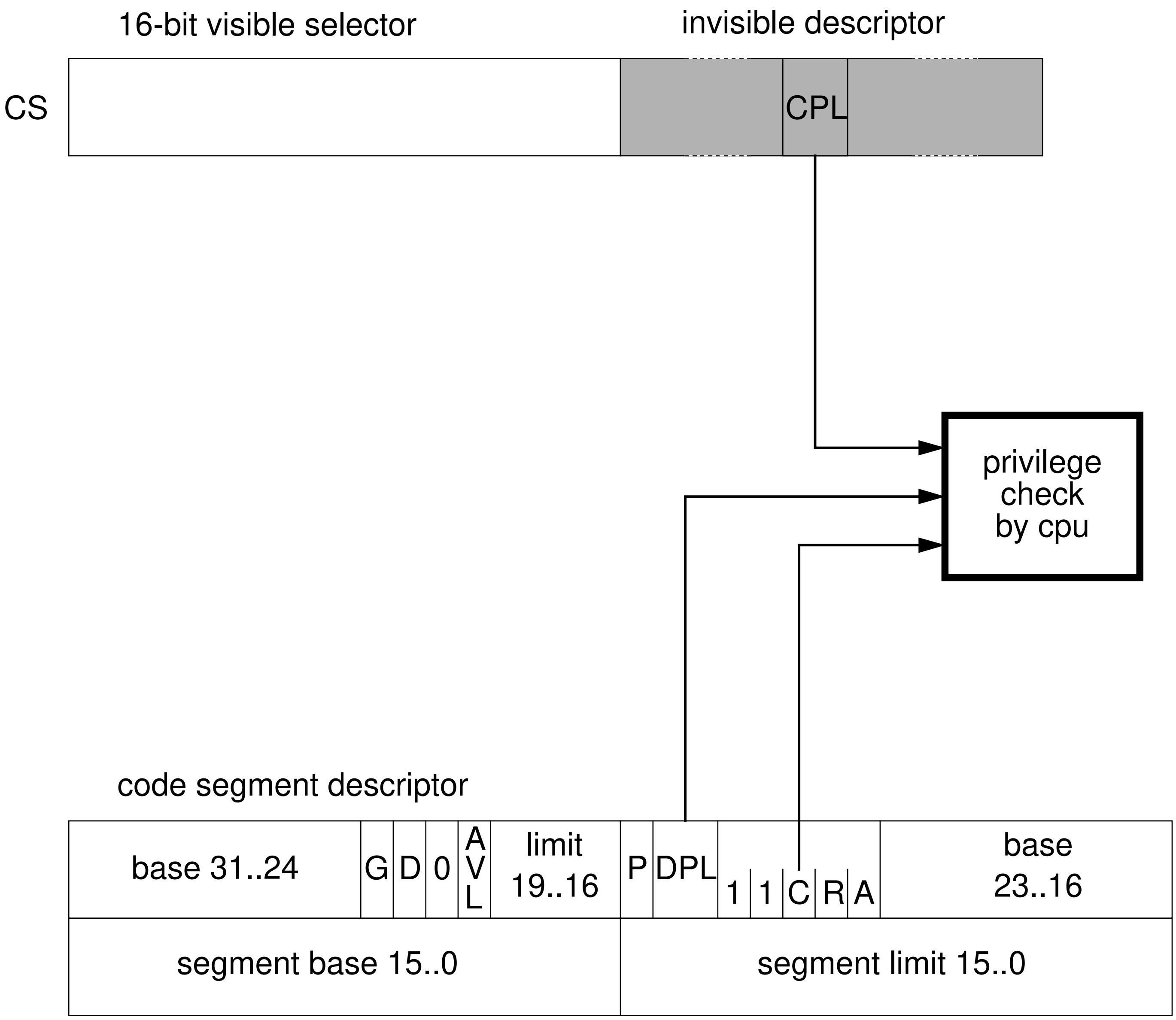


CPL - current privilege level
RPL - requestor's privilege level
DPL - descriptor privilege level

Zugangsbedingung:

$$DPL \geq \max(CPL, RPL)$$

Kontrolltransfer ohne 'gate'

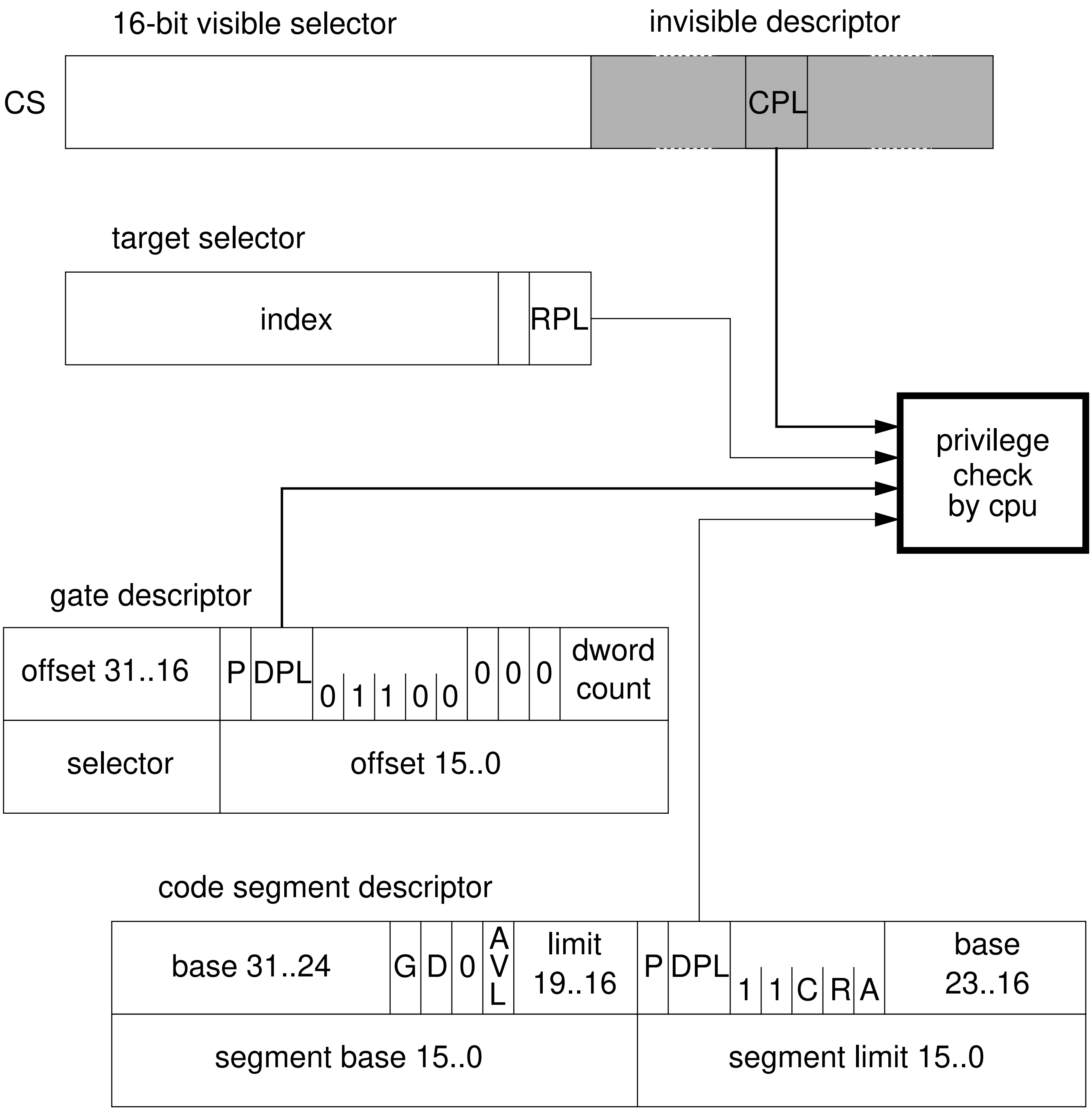


CPL - current privilege level
DPL - descriptor privilege level
C - conforming bit

Zugangsbedingung:

DPL = CPL
oder
C = 1 und DPL ≤ CPL

Kontrolltransfer mit 'gate'



Zugangsbedingung:

CPL - current privilege level
RPL - requestor's privilege level
DPL - descriptor privilege level

gate DPL $\geq$ max(CPL, RPL)

und

CPL $\geq$ target segment DPL

8.4 Betriebssystemstrukturen

H Y D R A

Hardware

- Multiprozessor
- prozessorspezifische Segmentierung
- kleine, in Spezialregistern untergebrachte Segmenttabelle

Grundprinzipien

- Objektschutz
- Typisierung
- Capabilities
- need-to-know
- information-hiding

Objekte, Capabilities

- Objekt: Name, Typ
- Capability: Objektname, Zugriffsattribute
- Objektdarstellung: Datenteil, C-Liste

LNS und Pfade

- LNS definiert aktuelle Schutzumgebung eines Prozesses
- Pfade: LNS[i1]/C[i2]/...
  Interpretation beginnt grundsätzlich im aktuellen LNS

Allgemeine Operationen

-	getdata	Kopieren eines Auschnitts aus dem Datenbereich eines Objektes in einen Bereich des lokalen Adreßraums
-	putdata	Kopieren eines Ausschnitts aus dem lokalen Adreßraum in einen Bereich des Datenteils eines Objektes
-	adddata	Anfügen der Kopie eines Ausschnitts des lokalen Adreßraums an den Datenteil eines Objektes
-	load	Kopieren einer Capability aus der C-Liste eines Objektes in den LNS
-	store	Kopieren einer Capability des LNS in die C-Liste eines Objektes
-	append	Anfügen der Kopie einer Capability des LNS an die C-Liste eines Objektes
-	delete	Tilgen einer Capability im LNS oder der C_Liste eines Objektes
-	destroy	Tilgen eines Objektes
-	copy	Kopie eines Objektes in einem neuen Objekt anlegen
-	create	Erzeugen eines (leeren) Objektes
-	call	Aufrufen einer Prozedur (Ausführen eines PROC-Objektes, Methodenaufruf)

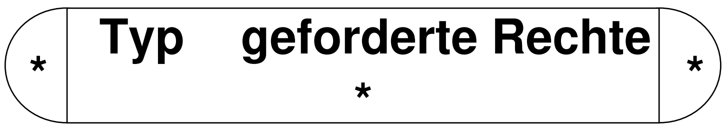
Allgemeine Zugriffsattribute (bei HYDRA 'rights' genannt)

GETRTS	erlaubt es, dem Datenteil eines Objektes Daten zu entnehmen.
PUTRTS	erlaubt es, den Datenteil eines Objektes zu beschreiben.
ADDRTS	erlaubt es, an den Datenteil eines Objektes Daten anzufügen.
LOADRTS	erlaubt es, eine Capability aus der C-Liste eines Objektes in einen lokalen Namensraum (LNS) zu kopieren.
STORTS	erlaubt es, eine Capability aus einem lokalen Namensraum (LNS) in die C-Liste eines Objektes zu kopieren.
APPRTS	erlaubt es, an die C-Liste eines Objektes Capabilities anzufügen.
KILLRTS	erlaubt es, eine Capability aus der C-Liste eines Objektes zu entfernen.
COPYRTS	erlaubt es, ein Objekt zu kopieren.
OBJRTS	erlaubt es, ein Objekt zu eliminieren.
CALRTS	erlaubt es, eine Prozedur aufzurufen

Schablonen

- in der C-Liste abgelegt
- identifizieren kein Objekt
- als Muster für die Schaffung einer Capability auffassbar
- Schablonenarten
  - Erzeugungsschablonen, bestehend aus Typ und neuen Rechten
  - Paramererschablonen bestehend aus Typ und geforderten Rechten

    Darstellung

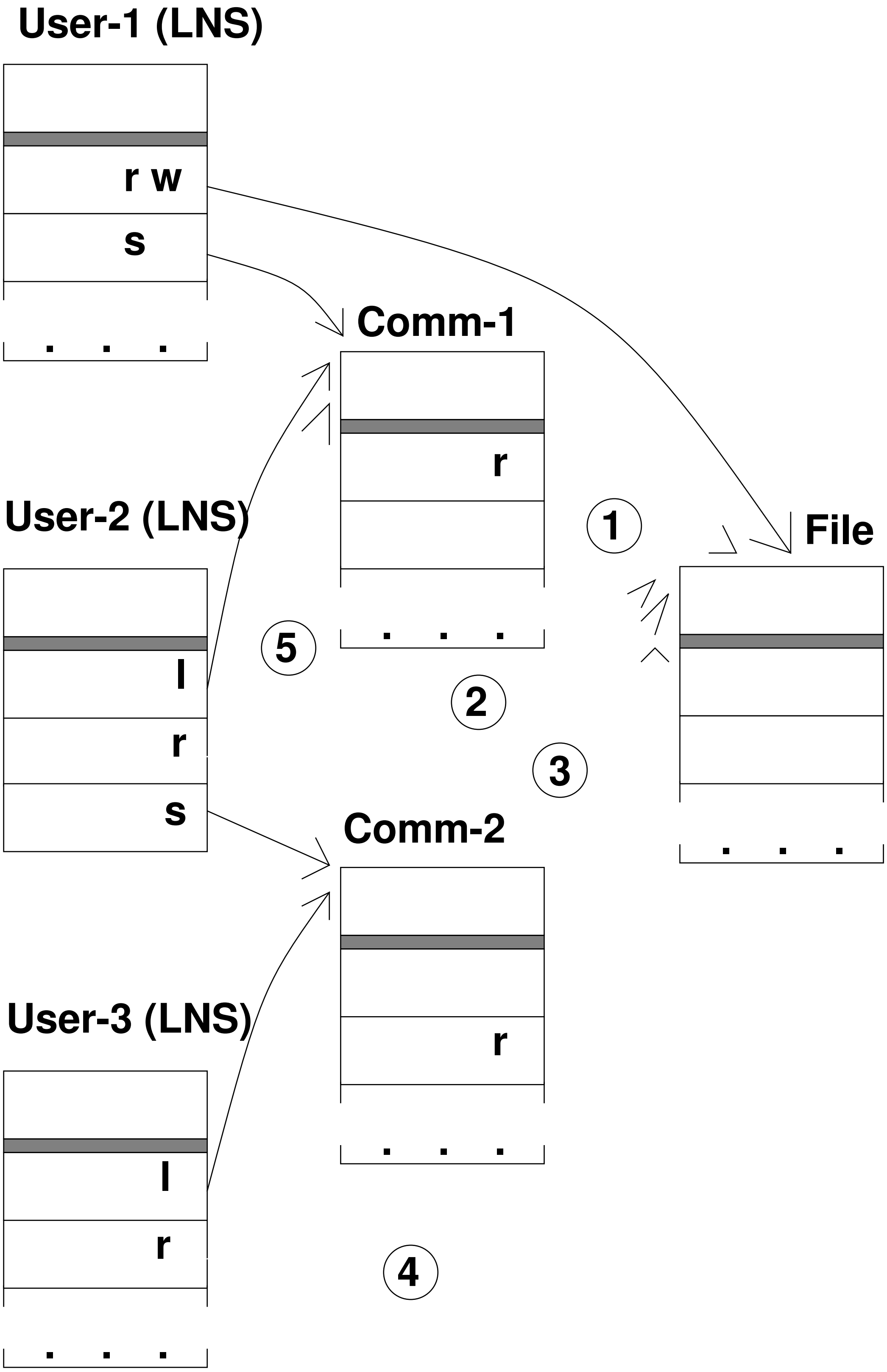


  - Erweiterungsschablonen bestehend aus Typ, geforderten Rechten und neuen Rechten

    Darstellung

Prozeduraufrufmechanismus

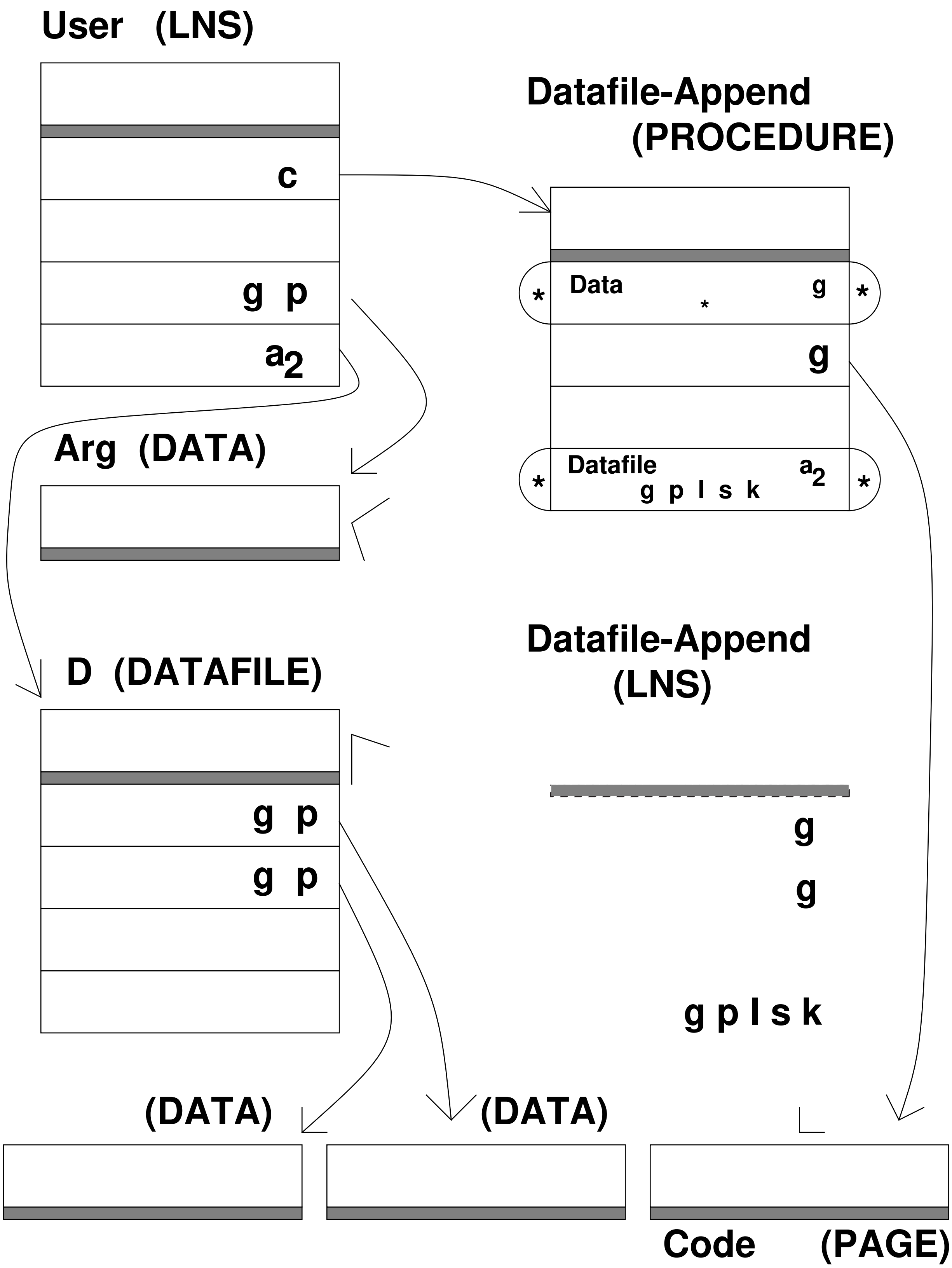
- Call(cproc, return_slot, cp1, mask1, ..., cpn, maskn)
  - C-Liste wird in neuen LNS kopiert.
  - Parameter- und Eweiterungsschablonen werden der Reihe nach den Parameter-Capabilities im Aufruf zugeordnet.
  - Parameterschablonen werden nach Maskierung und erfolgreicher Typ- und Rechteprüfung durch den maskierten Aktualparameter ersetzt
  - Erweiterungsschablonen werden nach Maskierung und erfolgreicher Typ- und Rechteprüfung durch den Aktualparameter ersetzt, wobei statt der alten die neuen Rechte eingetragen werden.



Spezielle Rechte:

r Leserecht für Objekte vom Typ 'File'

w Leserecht für Objekte vom Typ 'File'



Wirkung von Parameter- und Erweiterungsschablonen beim Prozeduraufruf
Call(LNS[0], LNS[2], g, LNS[3], all)

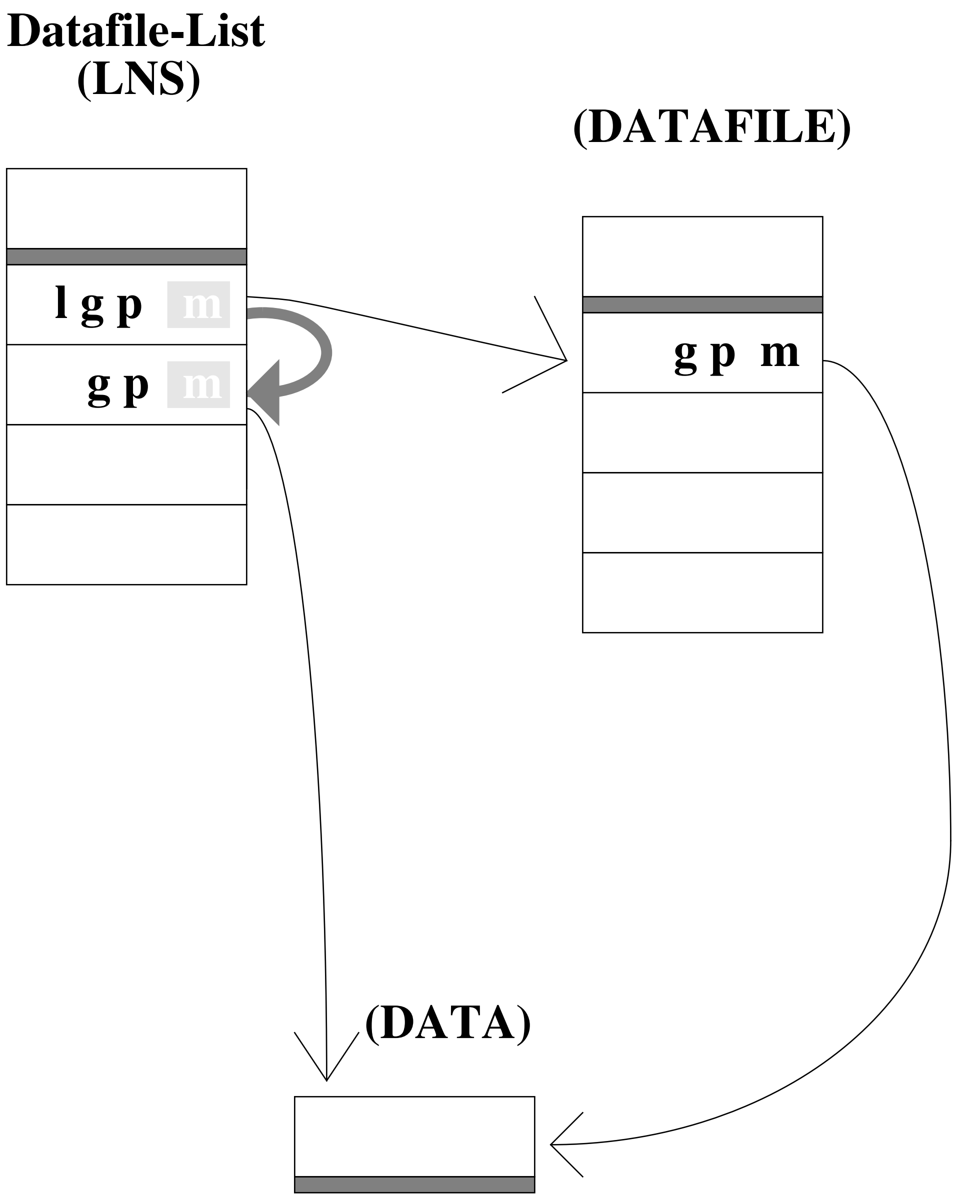
Spezielles Recht für Objekte vom Typ 'DATAFILE':
a_2 Anfügerecht

Gegenseitiges Mißtrauen

- Aufgabenstellung
  - Der Benutzer einer Prozedur soll in der Lage sein, den Zugang zu seinen Objekten auf die als Parameter übergebenen Objekte zu beschränken
  - Die Prozedur soll in der Lage sein sicherzustellen, daß Benutzer zu prozedurlokalen Objekten nur während der Ausführung der Prozedur Zugang haben.
- Lösung
  - Direkte Lösung durch den Mechanismus des Prozeduraufrufs.

Modifikation

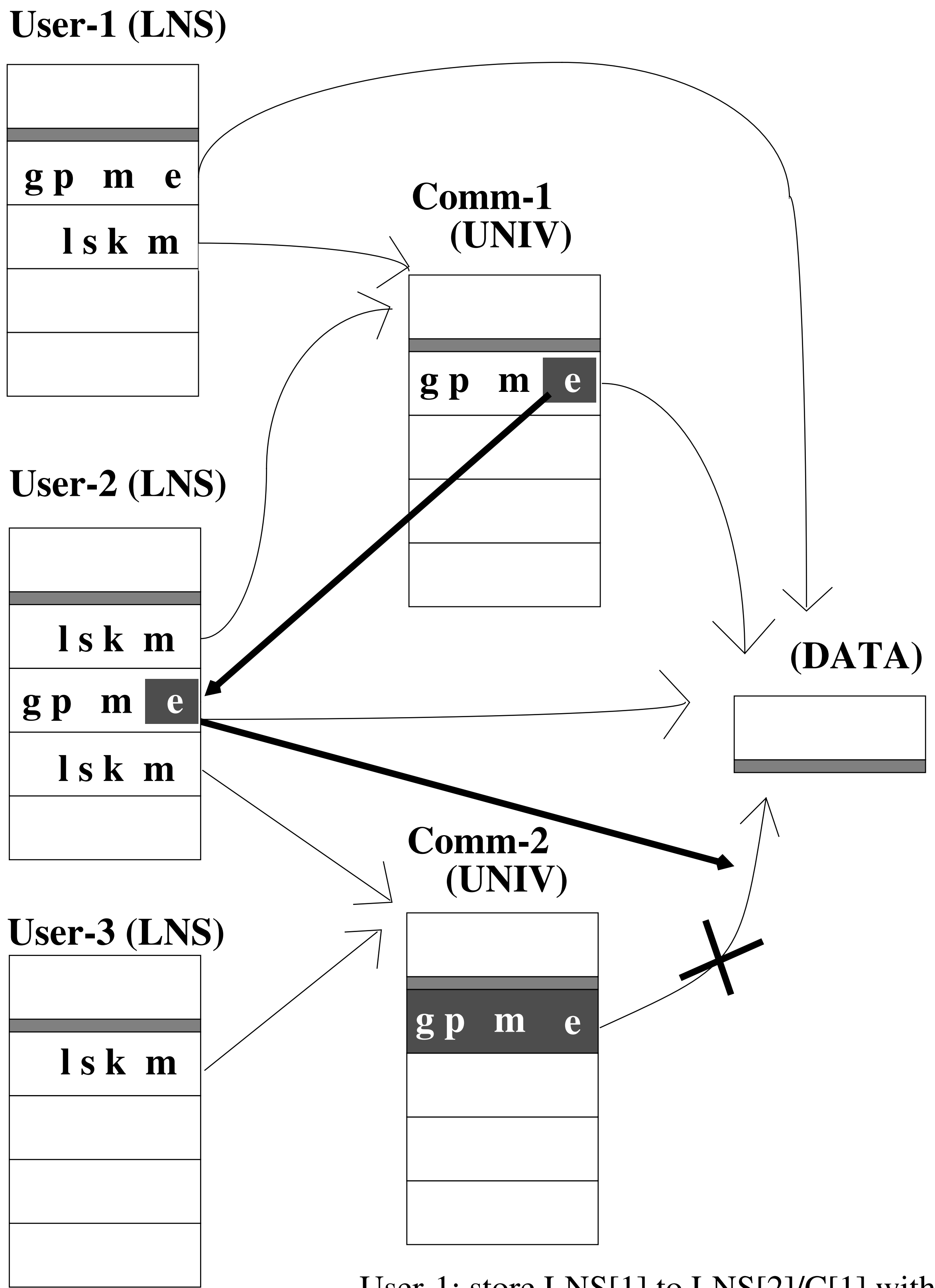
- Aufgabenstellung
  - Es soll trotz des Mechanismus der Rechteerweiterung möglich sein, eine aufgerufene Prozedur an der Modifikation von Objekten, die als Parameter übergeben wurden, zu hindern.
- Lösung
  - Einführung des zusätzlichen Rechtes MDFYRTS, das folgende Eigenschaften besitzt:
    - Die Ausübung von Rechten, die eine Modifikation des entsprechenden Objektes erlauben, ist nur dann möglich, wenn alle Capabilities des Pfades, der zum Zugriff auf das Objekt benutzt wird, zusätzlich über MDFYRTS verfügen.
    - MDFYRTS kann nicht im Rahmen einer Rechteerweiterung erworben werden.
    - Die Kernoperation load blendet beim Laden von Capabilities in den LNS ein eventuell vorhandenes MDFYRTS aus, wenn MDFYRTS in einer der Capabilities des Pfades fehlt, der zum Zugriff auf das Objekt benutzt wird.



LOAD LNS[1]/C[1] to LNS[2]

Ausbreitungsbegrenzung

- Aufgabenstellung
  - Ein Benutzer soll verhindern können, daß eine Capability, die er einem anderen Benutzer übergeben hat, von letzterem an einen dritten Benutzer weitergegeben wird.
- Lösung
  - Einführung des zusätzlichen Rechtes ENVRTS, das folgende Eigenschaften besitzt:
    - Eine Capability kann nur dann mittels store oder append in die C-Liste eines Objektes eingetragen werden, wenn sie über ENVRTS verfügt.
    - ENVRTS kann nicht im Rahmen einer Rechteerweiterung erworben werden.
    - Die Kernoperation load blendet bei der in den LNS zu ladenden Capability ein eventuell vorhandenes ENVRTS aus, wenn ENVRTS in einer der Capabilities des Pfades fehlt, der zum Zugriff auf das Objekt benutzt wird.



User-1: store LNS[1] to LNS[2]/C[1] without 'e'

User-2: load LNS[1]/C[1] to LNS[2]

Danach nicht möglich:

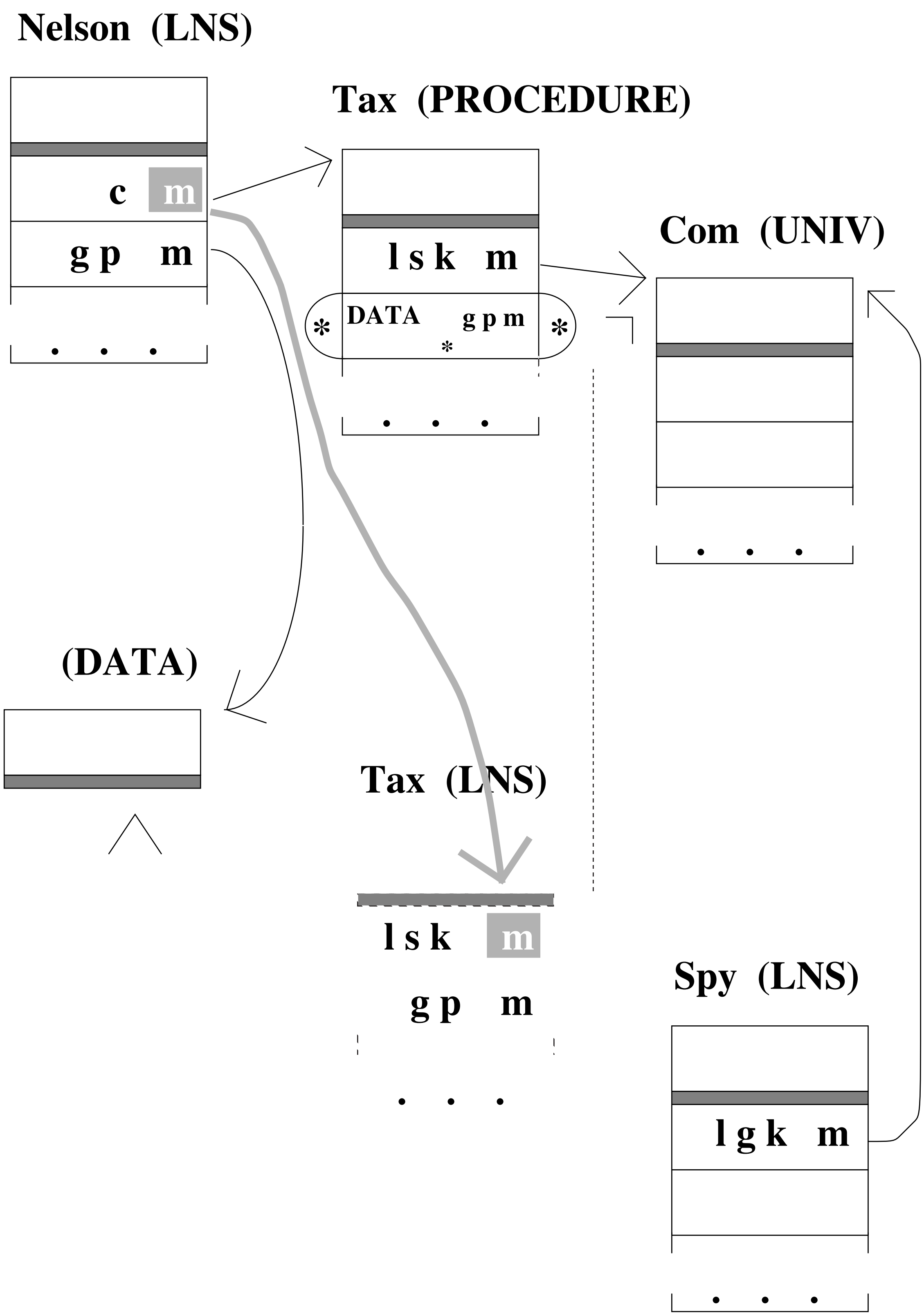
User-2: store LNS[2] to LNS[3]/C[1]

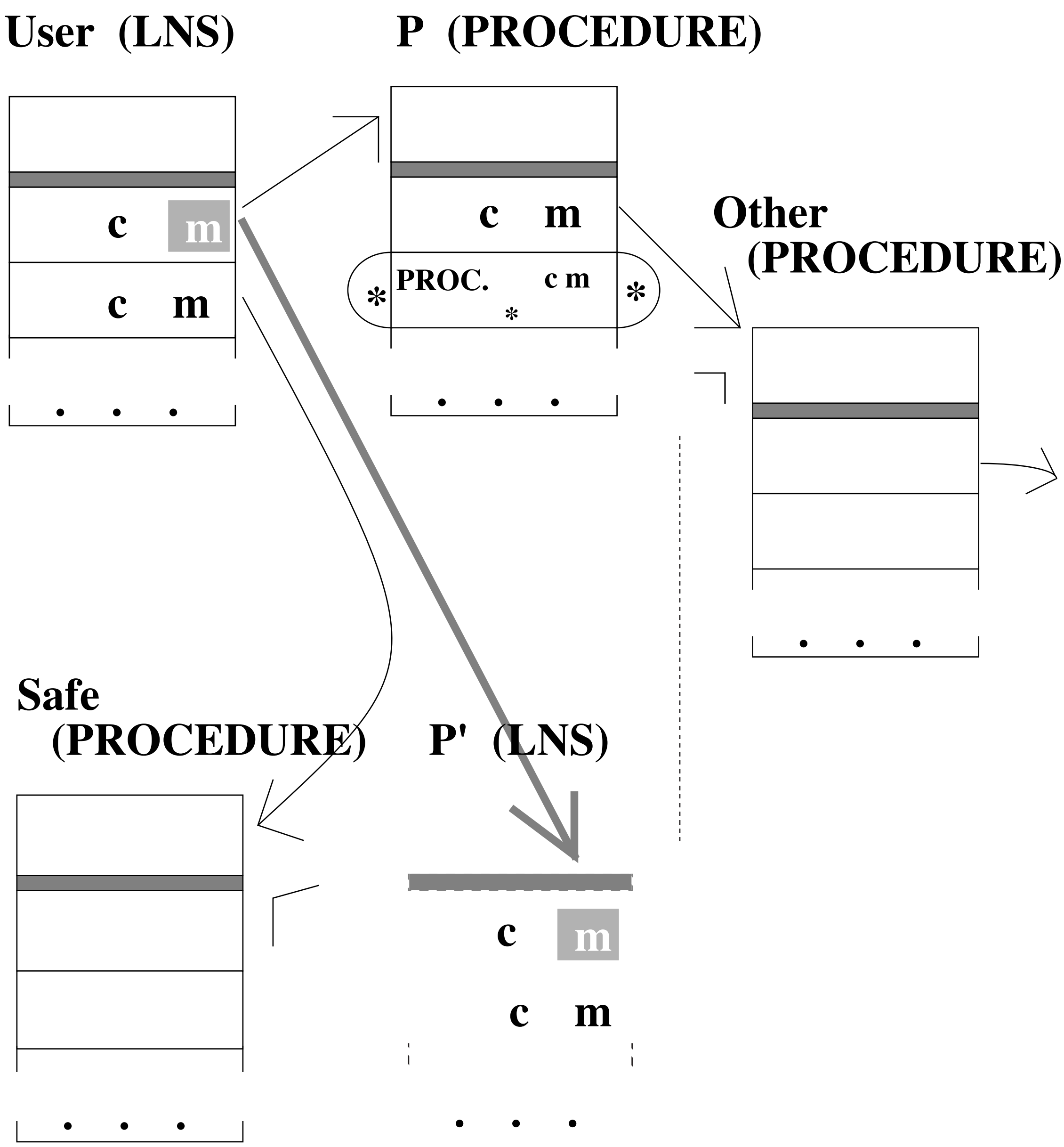
Bewahrung (Conservation)

- Aufgabenstellung:
  - Es soll unterbunden werden können, daß eine als Parameter übergebene Capability von der aufgerufenen Prozedur oder von einer durch sie aufgerufenen, nach Beendigung des Aufrufs zurückbehalten wird.
- Lösung:
  - ENVRTS zusammen mit dem Prozeduraufrufmechanismus.

Informationsflußbegrenzung (Confinement)

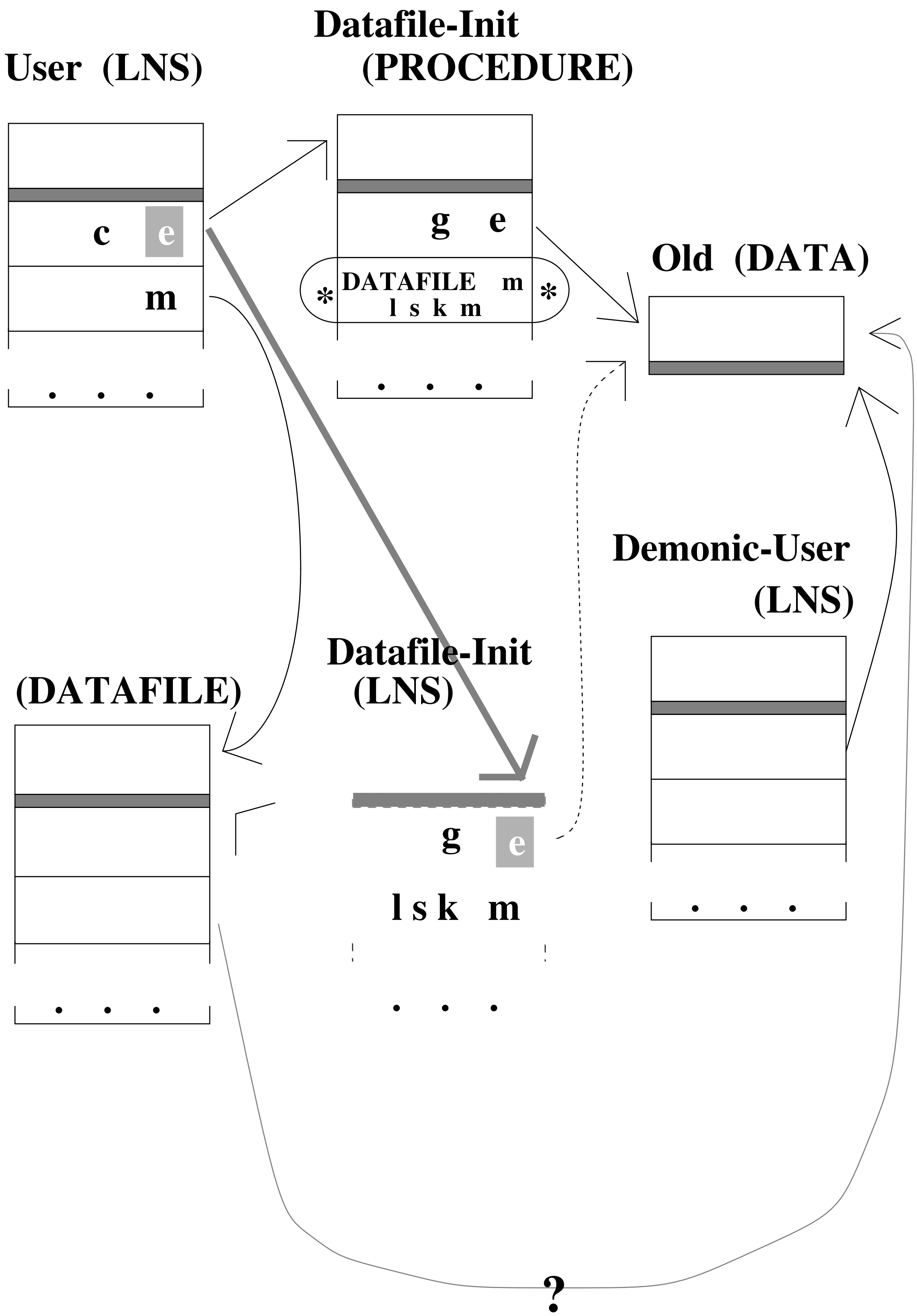
- Aufgabenstellung
  - Selektive Informationsflußbegrenzung
    - Es soll keine Information, die in bestimmten Parameterobjekten enthalten ist, zurückbehalten werden können.
  - Globale Informationsflußbegrenzung
    - Eine aufgerufene Prozedur soll keine Informationen aus Parameterobjekten zurückbehalten können.
- Lösung
  - Selektive Informationsflußbegrenzung
    - Keine direkte Lösung für die selektive Informationsflußbegrenzung.
  - Globale Informationsflußbegrenzung
    - Fehlendes MDFYRTS beim Aufruf einer Prozedur hat zur Folge, daß bei der Bildung des neuen LNS für prozedureigene Capabilities MDFYRTS ausgeblendet wird.


Nelson (LNS)
c m
g p m
. . .
Tax (PROCEDURE)
l s k m
DATA g p m
*
*
*
. . .
Com (UNIV)
. . .
(DATA)
Tax (LNS)
l s k m
g p m
. . .
Spy (LNS)
l g k m
. . .


User (LNS)
P (PROCEDURE)
c m
c m
. . .
c m
PROC.
c m
* * *
Other
(PROCEDURE)
Safe
(PROCEDURE)
P' (LNS)
c m
c m
. . .

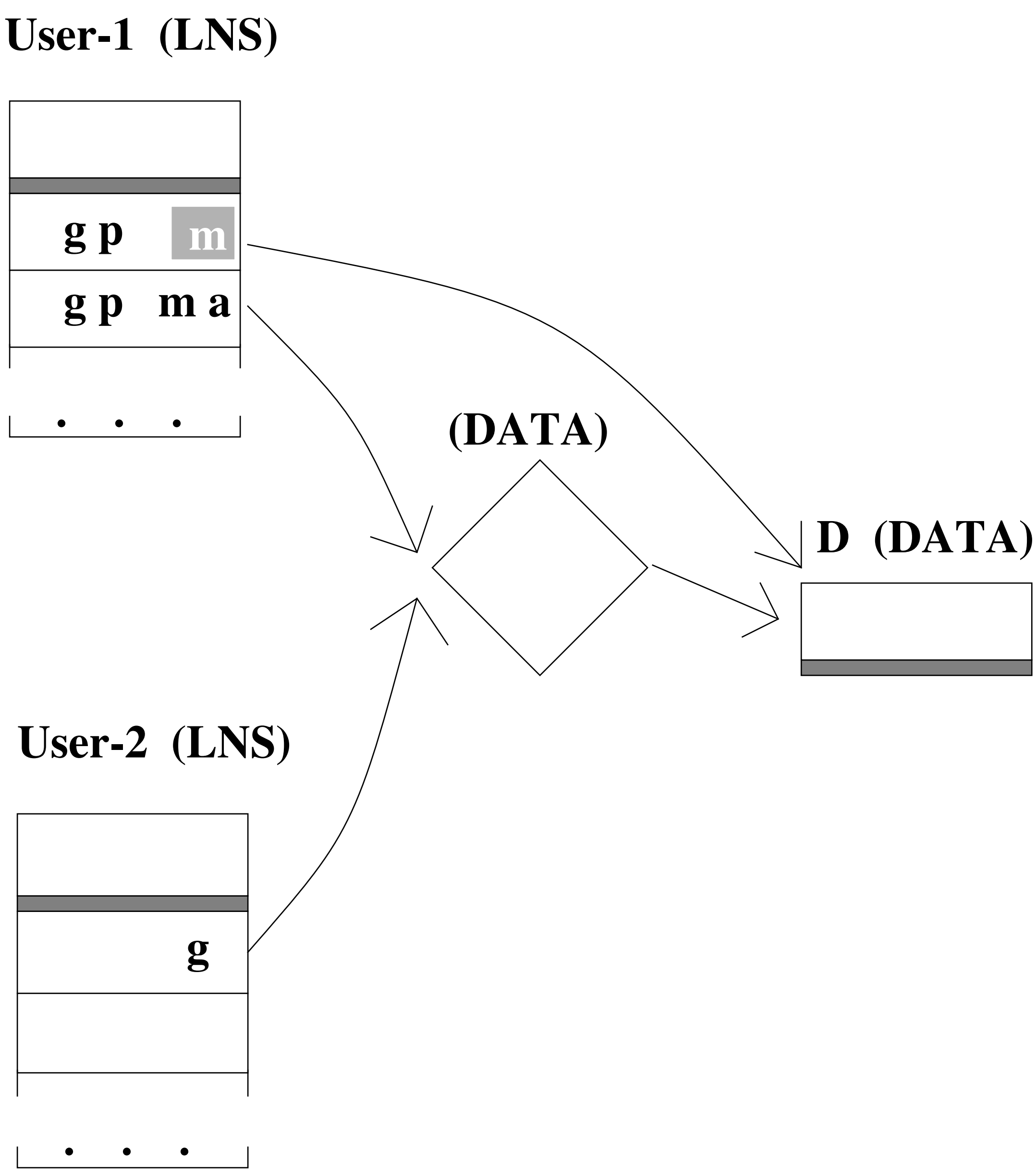
Initialisierung

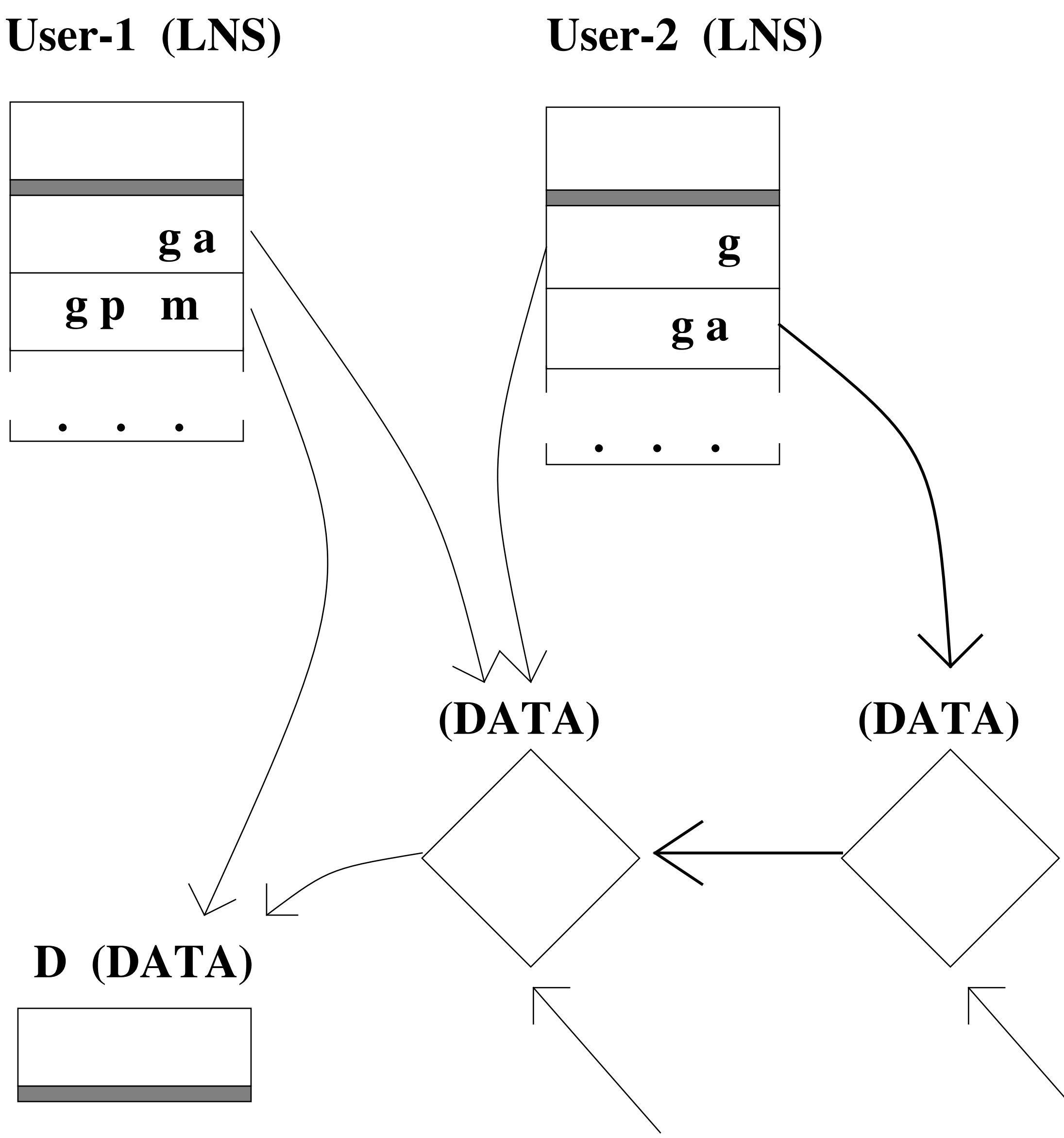
- Aufgabenstellung
  - Es soll sichergestellt werden, daß bei der Schaffung neuer Objekte weder Capabilities für das Objekt selbst noch für Objekte, die über seine C-Liste erreichbar sind, einem anderen Benutzer zugänglich gemacht wurden.
- Lösung
  - Aufruf der Erzeugungsprozedur ohne MDFYRTS und ohne ENVRTS.
  - Fehlendes ENVRTS beim Aufruf einer Prozedur hat zur Folge, daß bei der Bildung des neuen LNS für prozedureigene Capabilities ENVRTS ausgeblendet wird.

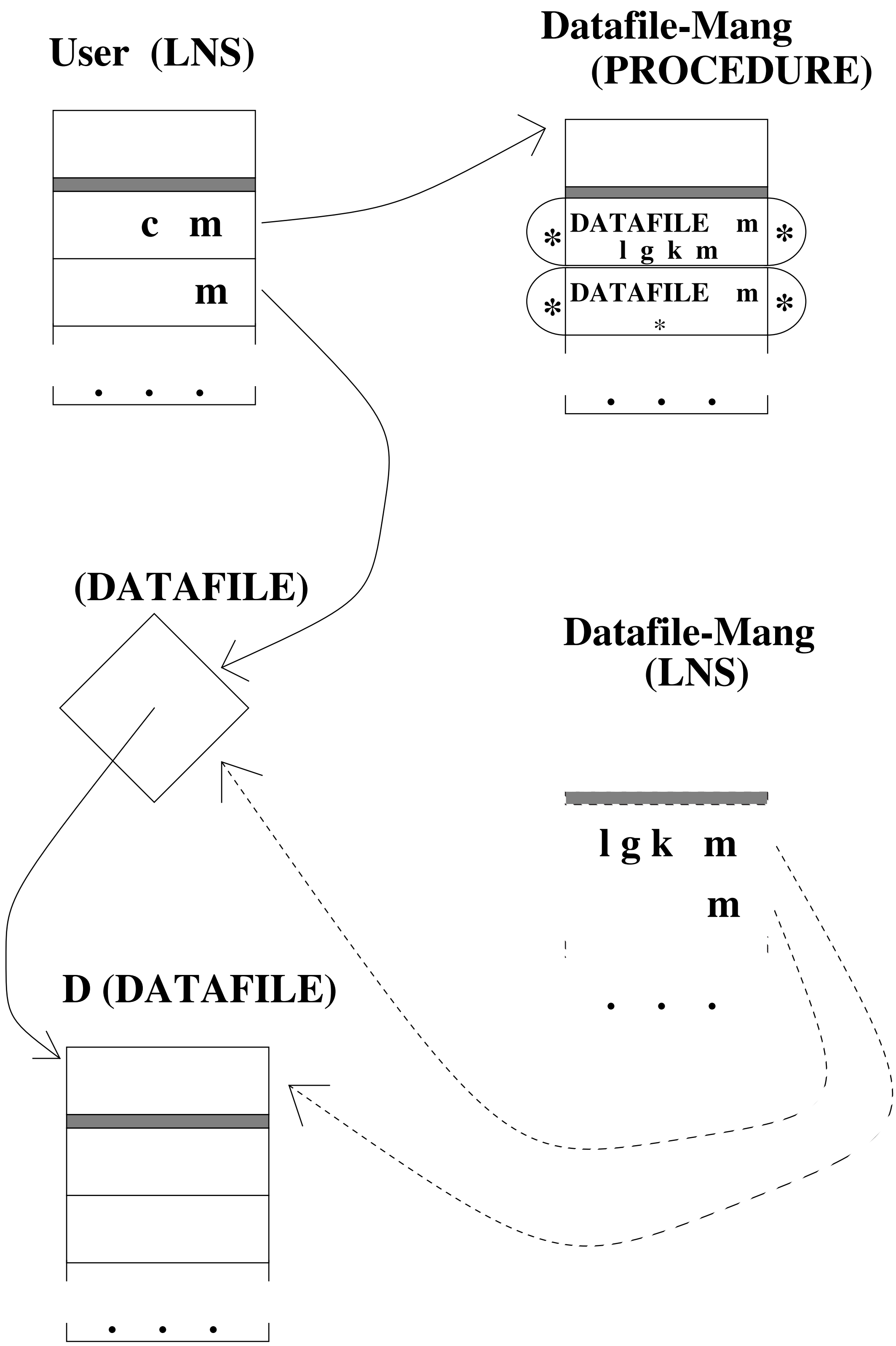

Datafile-Init
(PROCEDURE)
User (LNS)
c e
m
. . .
g e
DATAFILE m
l s k m
* *
. . .
Old (DATA)
Demonic-User
(LNS)
. . .
Datafile-Init
(LNS)
(DATAFILE)
g e
l s k m
. . .
. . .
?

Rückruf

- Aufgabenstellung
  - Sofortiger Rückruf
  - Dauerhafter Rückruf
  - Selektiver Rückruf
  - Partieller Rückruf
  - Weitergabe und Rückruf von Rückrufrechten
  - Schutz gegen einseitigen Rückruf
- HYDRA stellt nur Mechanismen zur Lösung der Aufgabenstellung des sofortigen, des selektiven und des zeitlich begrenzten Rückrufs zur Verfügung sowie zum Schutz vor einseitigem Rückruf.
- Lösung
  - Neuer Objekttyp 'Alias' als eine Art 'indirekte' Capability. Bei der Schaffung eines LNS wird bei Erweiterungsschablonen die Indirektion beseitigt, d. h. in den LNS wird eine 'direkte' Capability eingetragen.
  - ALLYRTS erlaubt das Unterbrechen und Wiederherstellen der Indirektion
  - FRZRTS stellt sicher, daß das Objekt nicht modifiziert werden kann; kann nur gesetzt werden, wenn alle Capabilities der C-Liste des bezeichneten Objektes bereits FRZRTS gesetzt haben; kann nicht im Rahmen einer Rechteerweiterung gewonnen werden; hat Entfernung von MDFYRTS zur Folge; kann nicht in Capabilities für Aliases gesetzt werden.


User-1 (LNS)
g p m
g p m a
. . .
(DATA)
D (DATA)
User-2 (LNS)
g
. . .


User-1 (LNS)
g a
g p m
. . .
User-2 (LNS)
g
g a
. . .
(DATA)
(DATA)
D (DATA)


User (LNS)
c m
m
. . .
Datafile-Mang
(PROCEDURE)
* DATAFILE m *
l g k m
* DATAFILE m *
*
. . .
(DATAFILE)
Datafile-Mang
(LNS)
l g k m
m
. . .
D (DATAFILE)
. . .

❑ Allgemeine Vereinbarungen

```
type
    Slot = integer;
    Path = array 1..* of Slot;
    KernelRights = array 1..16 of boolean;
    AuxiliaryRights = array 1..8 of boolean;
    Rights = record
        KernRts: KernelRights;
        AuxRts: AuxiliaryRights;
        end;

function PutCapa(D:Path; S:Slot; R:Rights): integer;
...
```

❑ Allgemeine Rechte

```
const
    PutCapaRts: KernelRights := [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
    GetCapaRts: KernelRights := [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
    ...
```

❑ Operationen an Rechten

```
var PutAndGetRts: KernelRights;

PutAndGetRts := PutCapaRts + GetCapaRts;
```

❑ Weitere Vereinbarungen

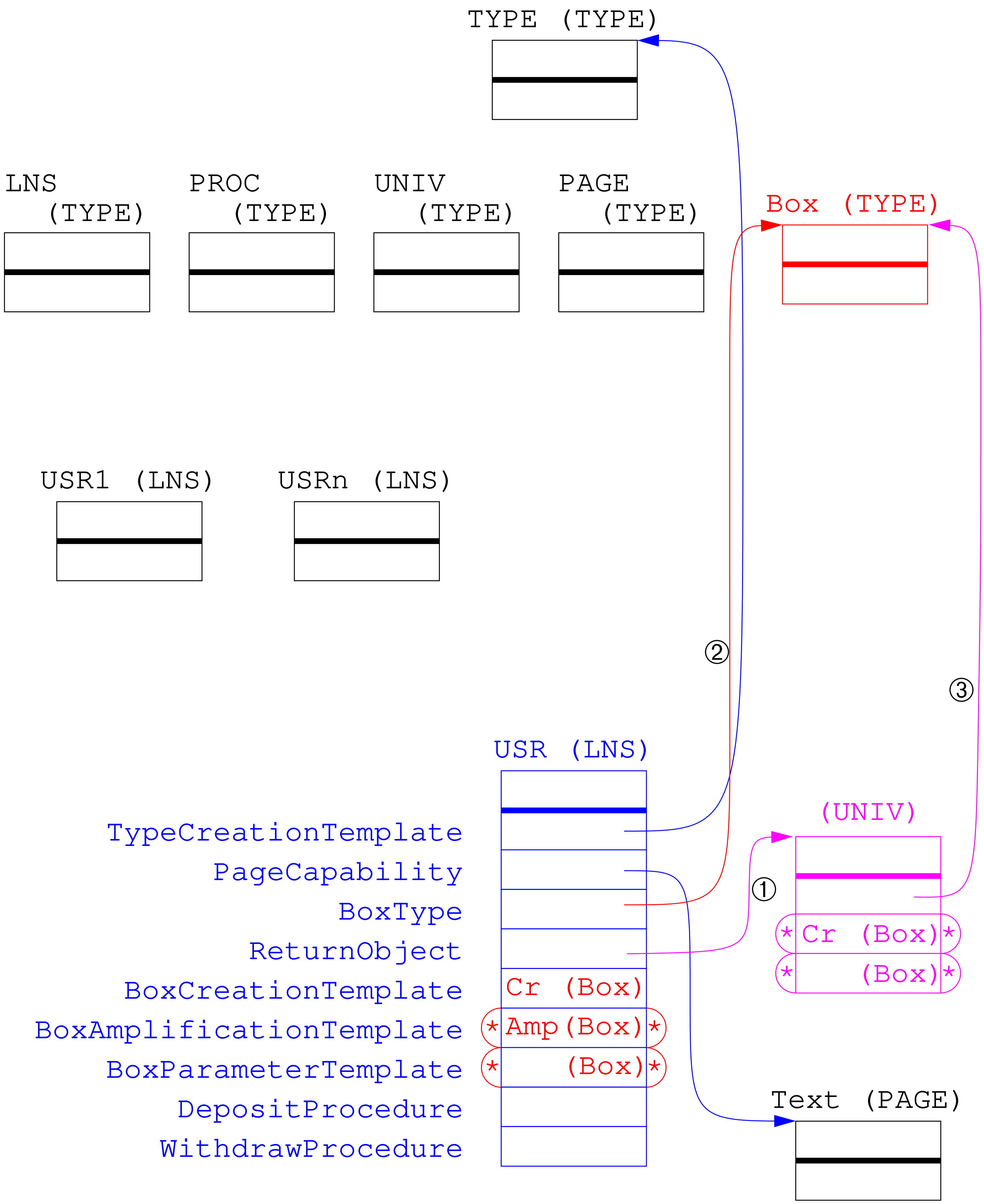
```
const
    AllRights: Rights := [[1,1,1,...,1],[1,1,1,1,1,1,1,1]];
    NoAuxRights: AuxiliaryRights := [0,0,0,0,0,0,0,0];
    NoRestrictions: Rights := AllRights;
```

❑ CreateBoxSubsystem

```
const
    BoxDepositRts: AuxiliaryRights := [1,0,0,0,0,0,0,0];
    BoxWithdrawRts: AuxiliaryRights := [0,1,0,0,0,0,0,0];

    TypeCreationTemplate = 1;
    PageCapability = 2;

const
```

```
    BoxType = 3;
    ReturnObject = 4;
    BoxCreationTemplate = 5;
    BoxAmplificationTemplate = 6;
    BoxParameterTemplate = 7;
    DepositProcedure = 8;
    WithdrawProcedure = 9;

procedure CreateBoxSubsystem;
begin
    MakeUniversal(ReturnObject);  // Zur Aufnahme der Ergebnisse
    Create(BoxType, TypeCreationTemplate, "Box");
    PutCapa(Path(ReturnObject, 1), BoxType, NoRestrictions);
    MakeCreationTemplate(BoxCreationTemplate, BoxType);
    MakeParameterTemplate(BoxParameterTemplate, BoxType);
    MakeAmplificationTemplate(BoxAmplificationTemplate, BoxType);
    PutCapa(Path(ReturnObject, 2), BoxCreationTemplate,
                [CreateRts+DeleteRts+ModifyRts+EnvRts+UncfRts;
                     BoxDepositRts+BoxWithdrawRts]);
    PutCapa(Path(ReturnObject, 3), BoxParameterTemplate, AllRights);
```

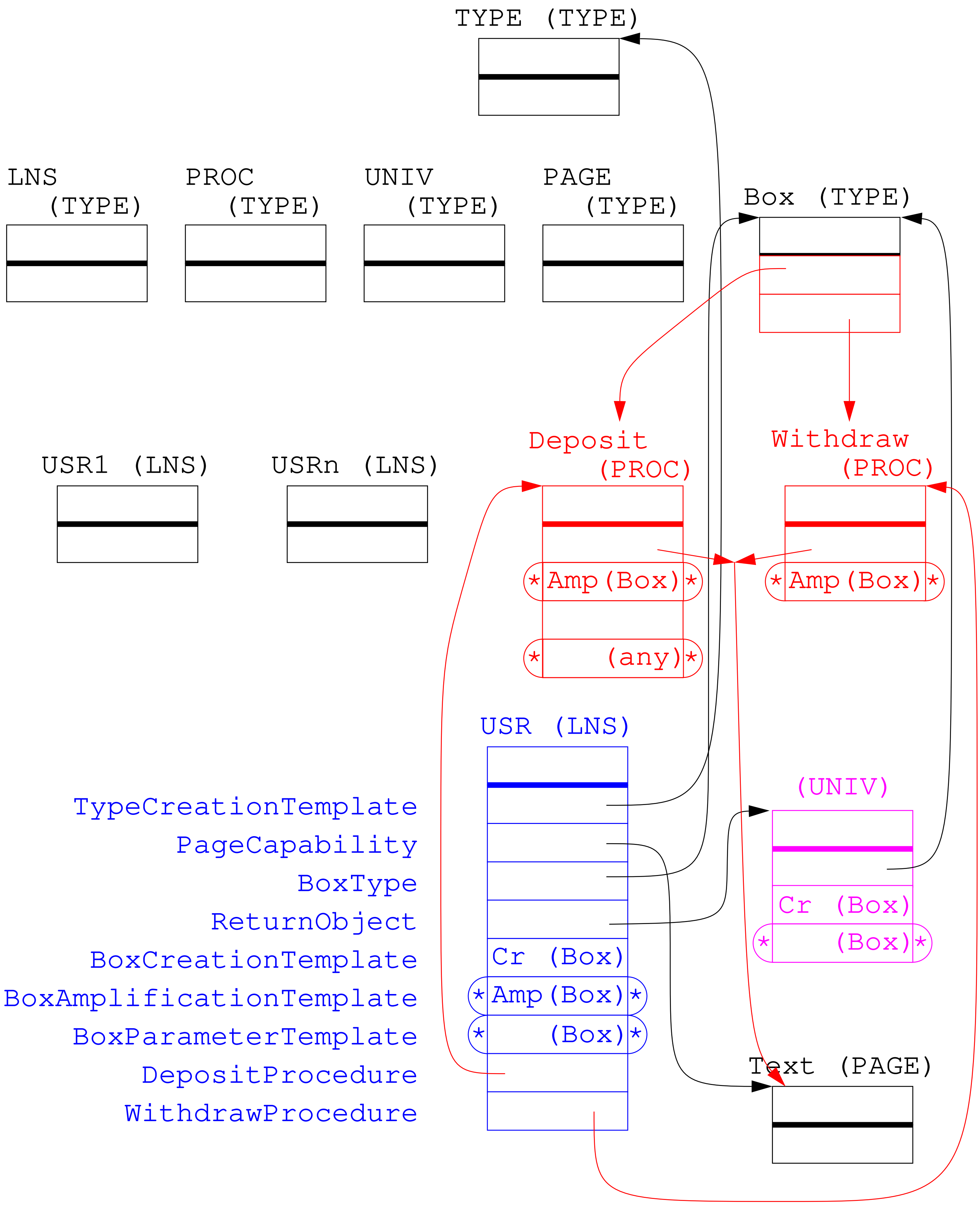
- Erzeugung der Prozedurobjekte



TYPE (TYPE)
LNS (TYPE)
PROC (TYPE)
UNIV (TYPE)
PAGE (TYPE)
Box (TYPE)
USR1 (LNS)
USRn (LNS)
②
③
USR (LNS)
(UNIV)
TypeCreationTemplate
PageCapability
BoxType
ReturnObject
BoxCreationTemplate
BoxAmplificationTemplate
BoxParameterTemplate
DepositProcedure
WithdrawProcedure
①
Cr (Box)
* (Box)*
Cr (Box)
Amp (Box)
* (Box)*
Text (PAGE)

```
const
    DepositPage = 1;
    WithdrawPage = 1;

MakeProcedure(DepositProcedure);
MakeProcedure(WithdrawProcedure);
PutCapa(Path(DepositProcedure, DepositPage),
          PageCapability, NoRestrictions);
SetStartingAddress(DepositProcedure, Deposit);
PutCapa(Path(WithdrawProcedure, WithdrawPage),
          PageCapability, NoRestrictions);
SetStartingAddress(WithdrawProcedure, Withdraw);
```

- Erzeugung ihrer Parameterlisten


TYPE (TYPE)
LNS (TYPE)
PROC (TYPE)
UNIV (TYPE)
PAGE (TYPE)
Box (TYPE)
Deposit (PROC)
Withdraw (PROC)
USR1 (LNS)
USRn (LNS)
* Amp (Box) *
* (any) *
* Amp (Box) *
USR (LNS)
TypeCreationTemplate
PageCapability
BoxType
ReturnObject
BoxCreationTemplate
BoxAmplificationTemplate
BoxParameterTemplate
DepositProcedure
WithdrawProcedure
Cr (Box)
* Amp (Box) *
* (Box) *
(UNIV)
Cr (Box)
* (Box) *
Text (PAGE)

```
const
DepositBoxParameter = 2;
WithdrawBoxParameter = 2;

PutCapa(Path(DepositProcedure, DepositBoxParameter),
          BoxAmplificationTemplate, NoRestrictions);
SetRequiredRights(Path(DepositProcedure, DepositBoxParameter),
          [ModifyRts+EnvRts+UncfRts; BoxDepositRts]);
PutCapa(Path(WithdrawProcedure, WithdrawBoxParameter),
          BoxAmplificationTemplate, NoRestrictions);
SetRequiredRights(Path(WithdrawProcedure, WithdrawBoxParameter),
          [ModifyRts+EnvRts+UncfRts; BoxWithdrawRts]);

const
AnyObjectParameter = 4;

MakeNullParameterTemplate(Path(DepositProcedure, AnyObjectParameter));
SetRequiredRights(Path(DepositProcedure, AnyObjectParameter),
          [EnvRts; NoAuxRights]);

const
```

```
DepositIndex = 1;
WithdrawIndex = 2;


PutCapa(Path(BoxType, DepositIndex),
          DepositProcedure, NoRestrictions);
PutCapa(Path(BoxType, WithdrawIndex),
          WithdrawProcedure, NoRestrictions);

Return(0, ReturnObject, NoRestrictions);
```

- Prozedur Deposit

```
procedure Deposit;
begin
    InterchangeCapa(Path(DepositBoxParameter, 1),
                        AnyObjectParameter, NoRestrictions);
    Update(DepositBoxParameter); // Sicherung im stabilen Speicher
    Return(0,0,0);
end;
```

- Prozedur Withdraw

```
procedure Withdraw;
begin
    TakeCapa(AnyObjectParameter, Path(WithdrawBoxParameter, 1));
    Update(WithdrawBoxParameter);
    Return(0, AnyObjectParameter, AllRights);
end;
```

- Benutzung des Subsystems

```
Create(BoxObject, BoxTemplate)

TypeCall(0, BoxObject, DepositIndex, BoxObject, Capa);

TypeCall(NewCapa, BoxObject, WithdrawIndex, BoxObject);
```

```
Create(BoxObject, BoxTemplate)

TypeCall(0,
           BoxObject, DepositIndex, BoxObject, Capa);
TypeCall(NewCapa,
           BoxObject, WithdrawIndex, BoxObject);
```