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

    * **Typ geforderte Rechte** *
    *

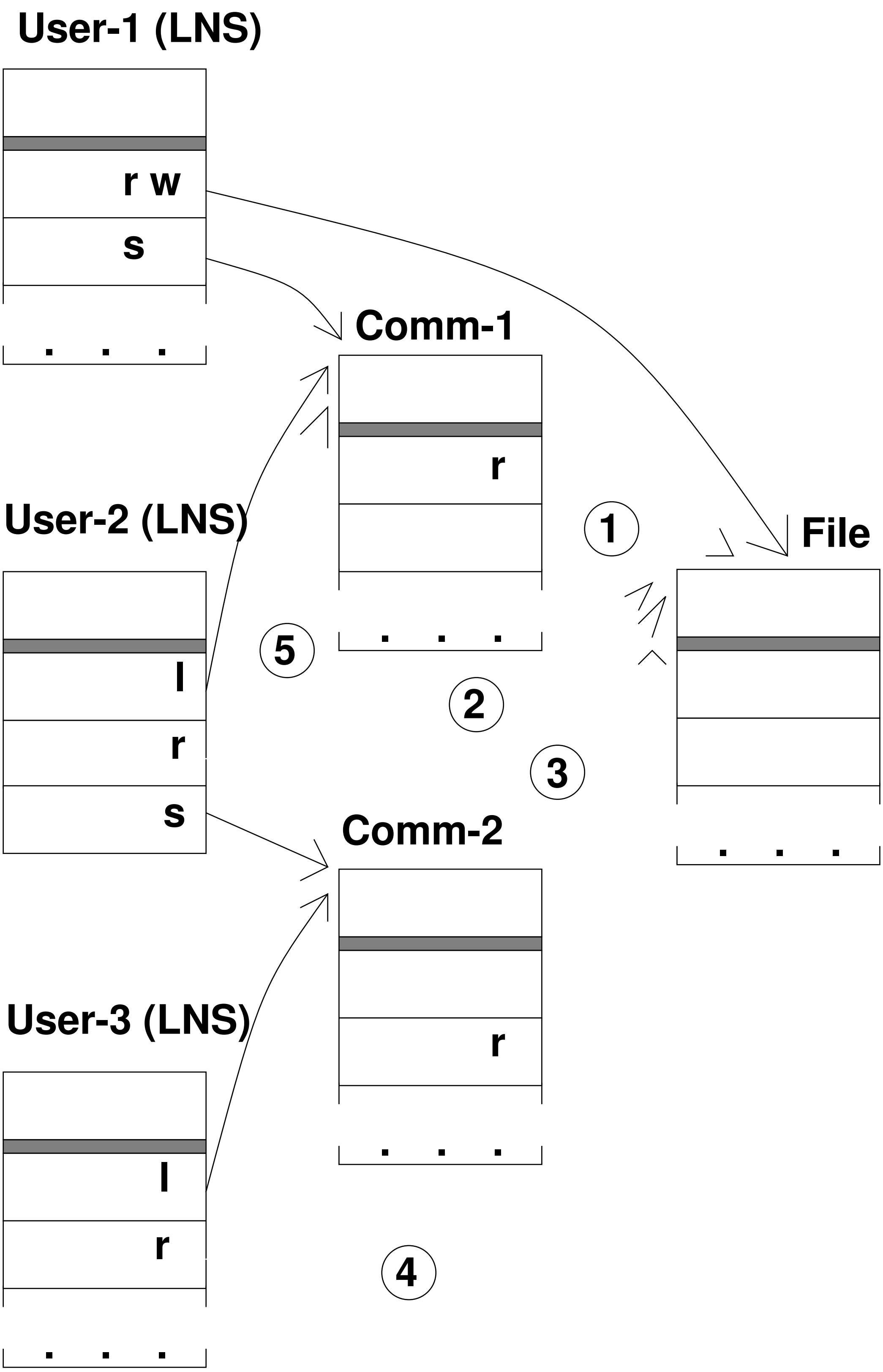
  - Erweiterungsschablonen bestehend aus Typ, geforderten Rechten und neuen Rechten

    Darstellung

    * **Typ geforderte Rechte** *
    neue Rechte

Prozeduraufrufmechanismus

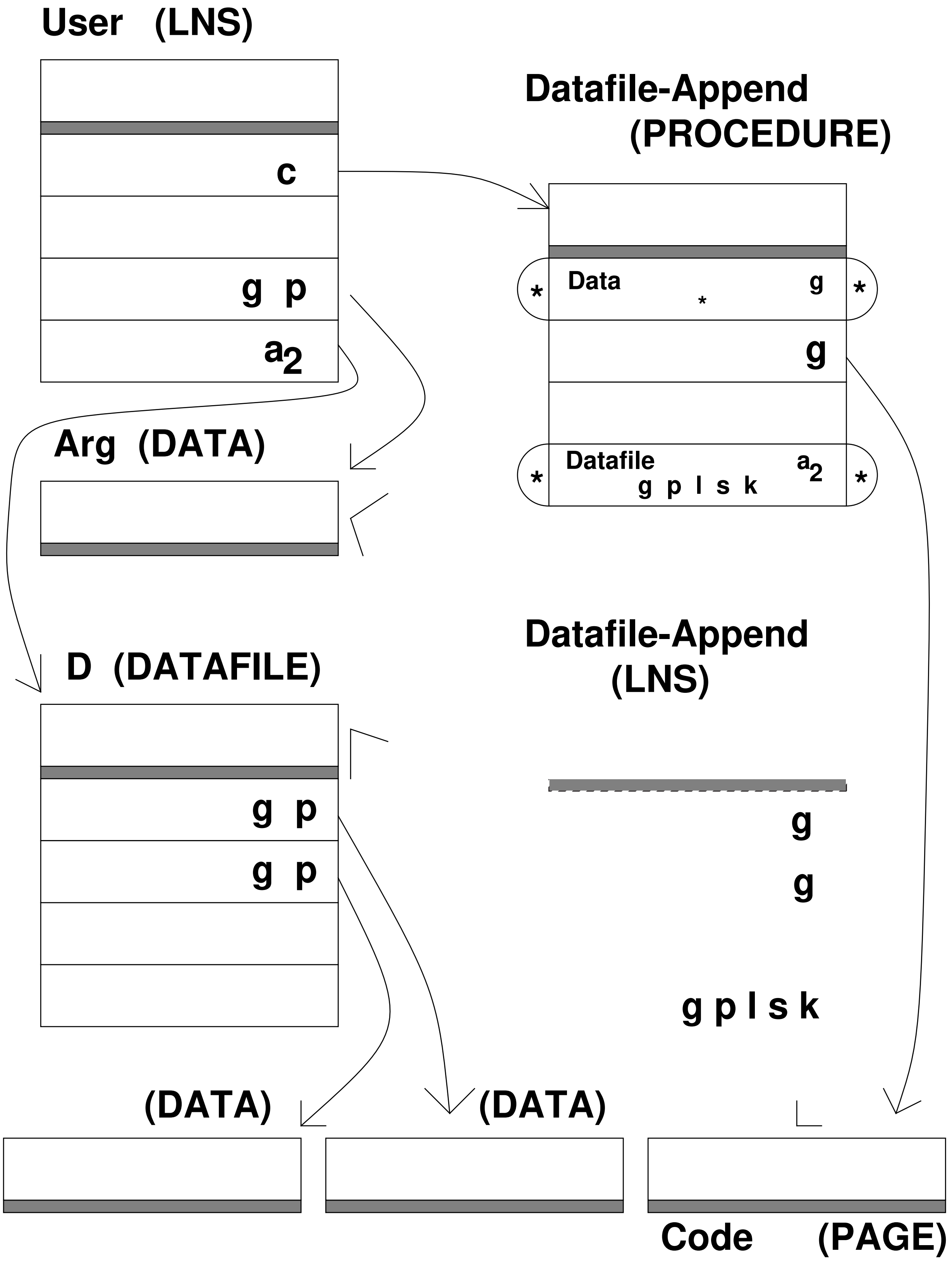
- Call(cproc, return_slot, cp1, mask1, ..., cpn, maskn)
  - C-Liste wird in neuen LNS kopiert.
  - Parameter- und Eweiterungsschablonen werden der Reihe nach den Parameter-Capabilities im Aufruf zugeordnet.
  - Parameterschablonen werden nach Maskierung und erfolgreicher Typ- und Rechteprüfung durch 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':

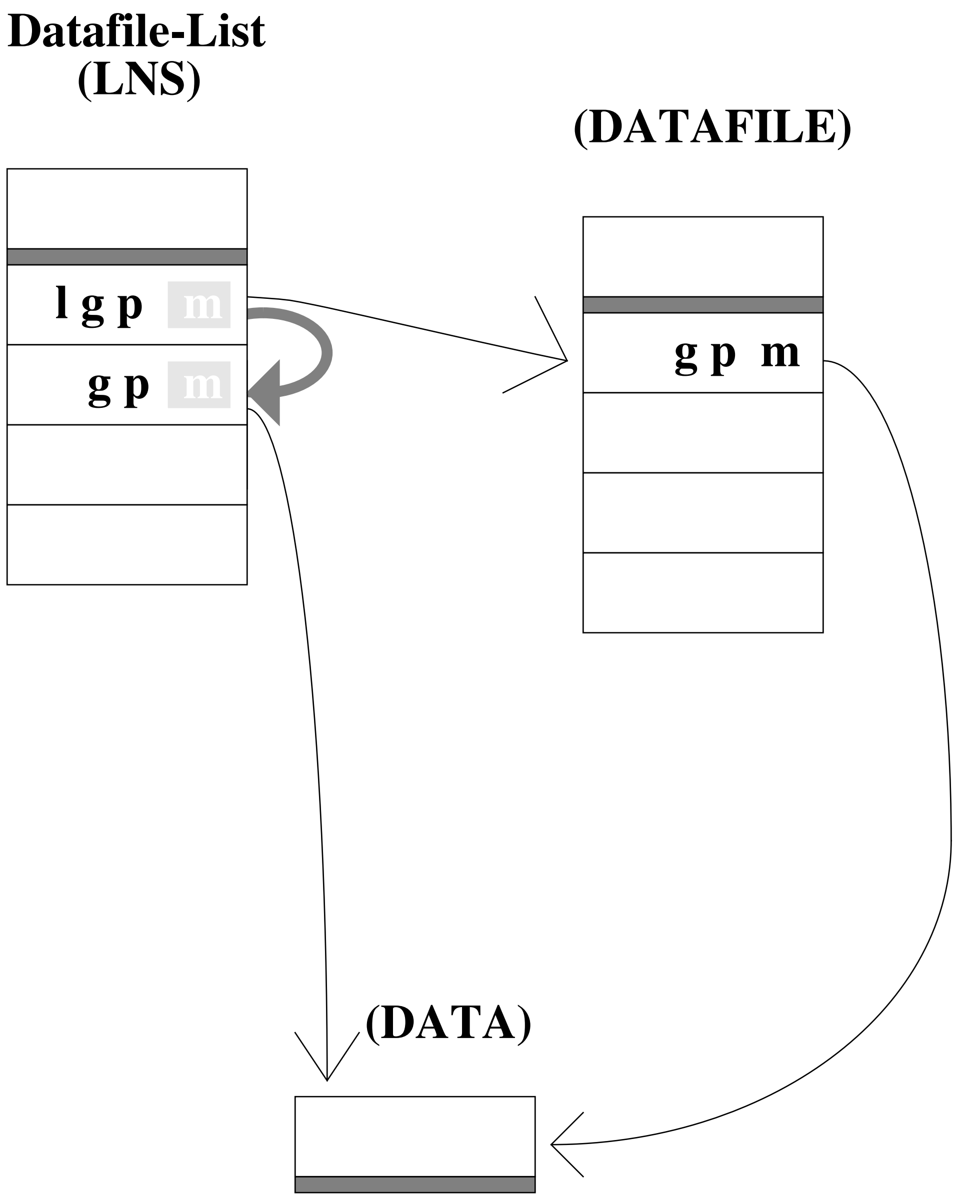
a_2 Anfügerecht

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

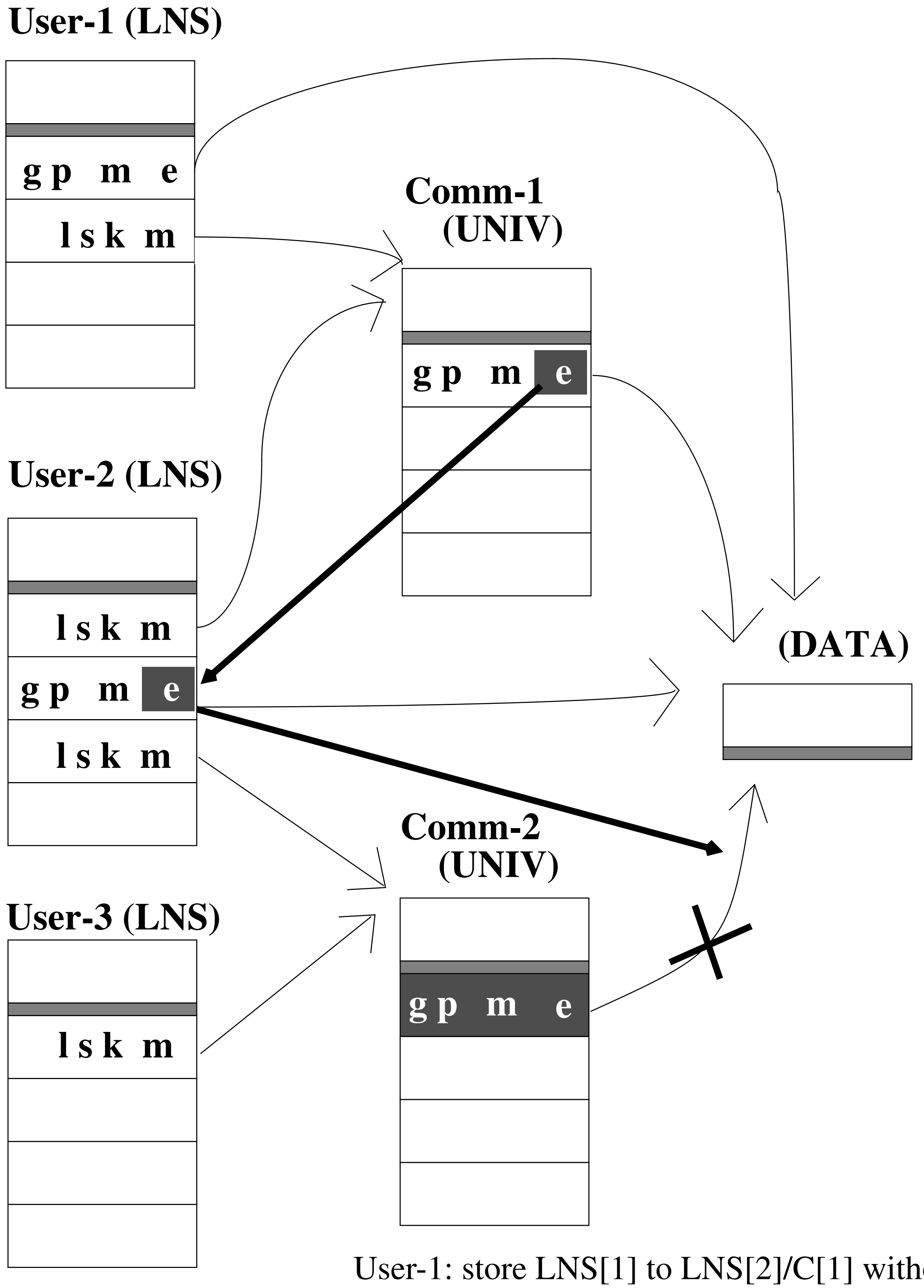
- Aufgabenstellung
  - Es soll trotz des Mechanismus der Rechteerweiterung möglich sein, eine aufgerufene Prozedur an der Modifikation von Objekten, die als Parameter übergeben wurden, zu hindern.
- Lösung
  - Einführung des zusätzlichen Rechtes MDFYRTS, das folgende Eigenschaften besitzt:
    - Die 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:

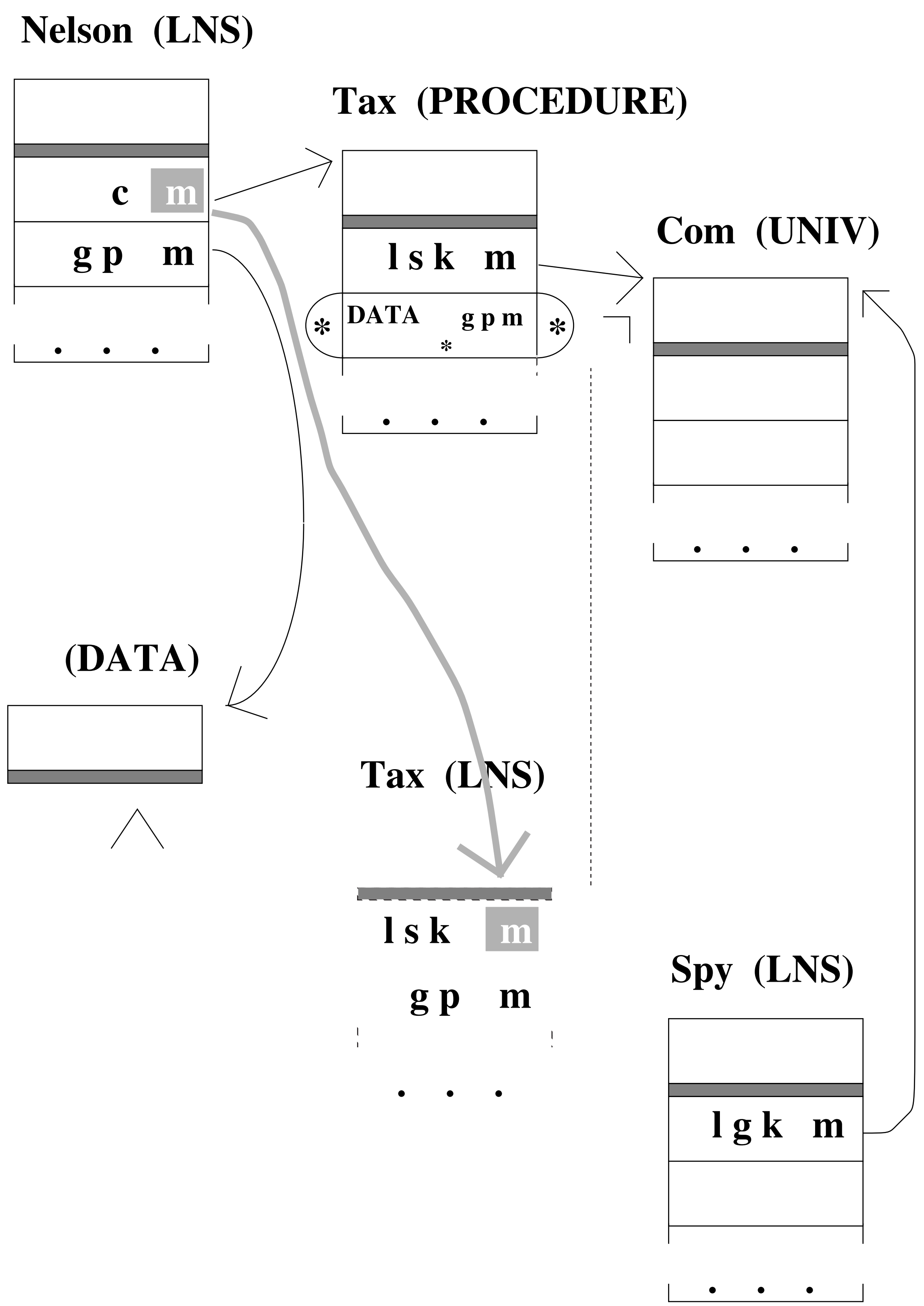
User-2: store LNS[2] to LNS[3]/C[1]

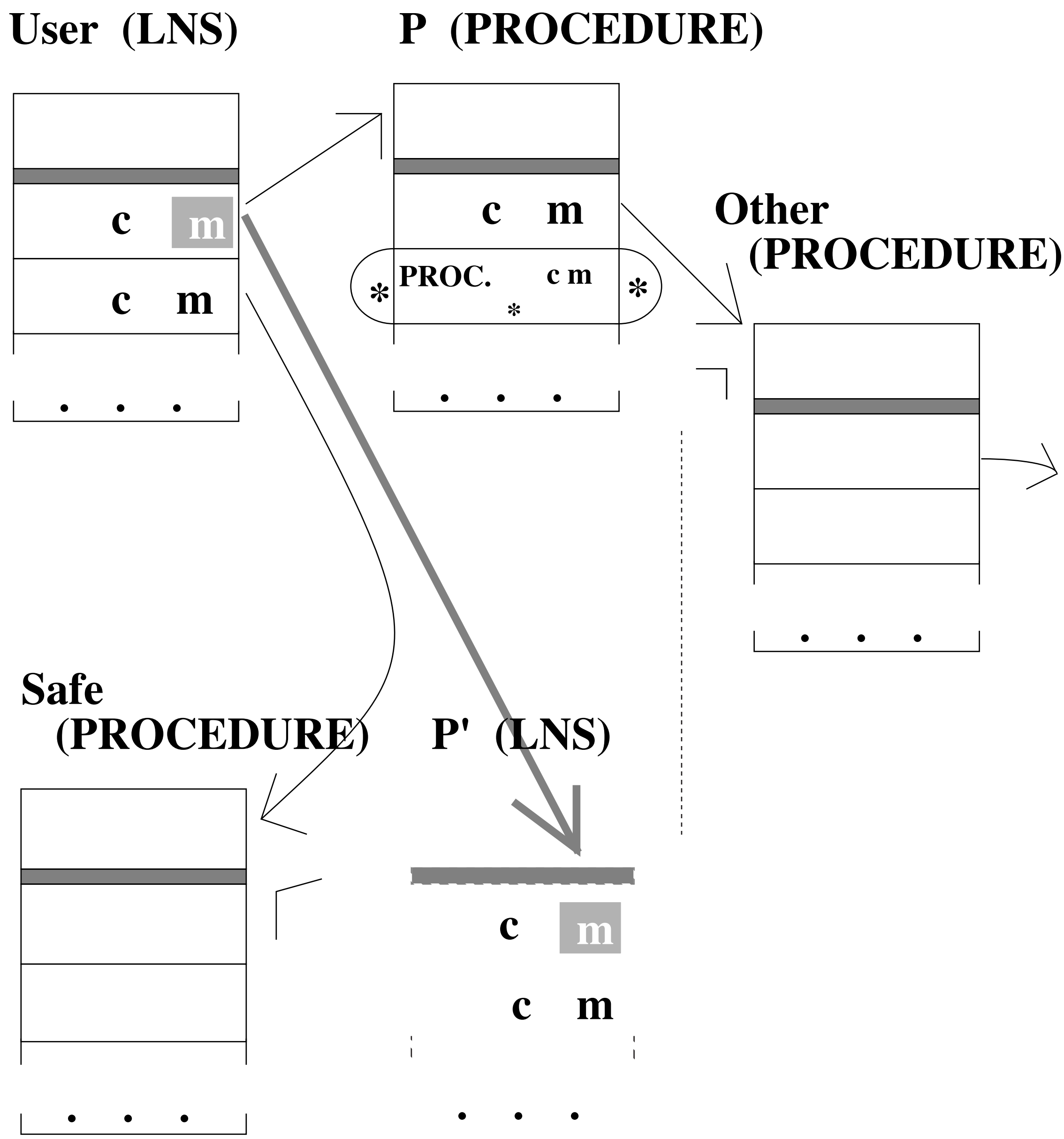
Bewahrung (Conservation)

- 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)

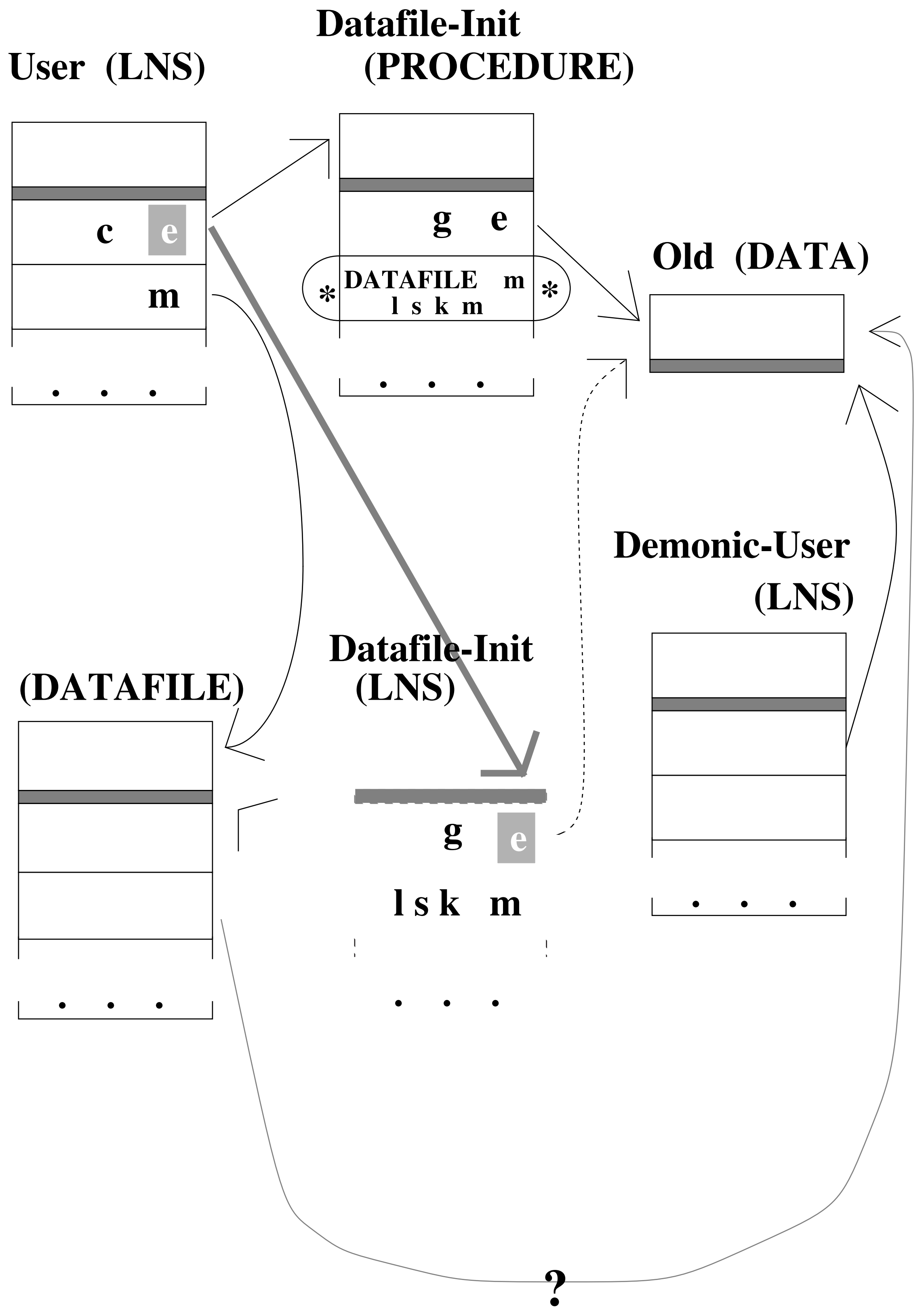
- Aufgabenstellung
  - Selektive Informationsflußbegrenzung
    - Es soll keine Information, die in bestimmten Parameterobjekten enthalten ist, zurückbehalten werden können.
  - Globale Informationsflußbegrenzung
    - Eine aufgerufene Prozedur soll keine Informationen aus Parameterobjekten zurückbehalten 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)
c m
c m
. . .
P (PROCEDURE)
c m
PROC. c m
* * *
. . .
Other
(PROCEDURE)
. . .
Safe
(PROCEDURE)
. . .
P' (LNS)
c m
c m
. . .

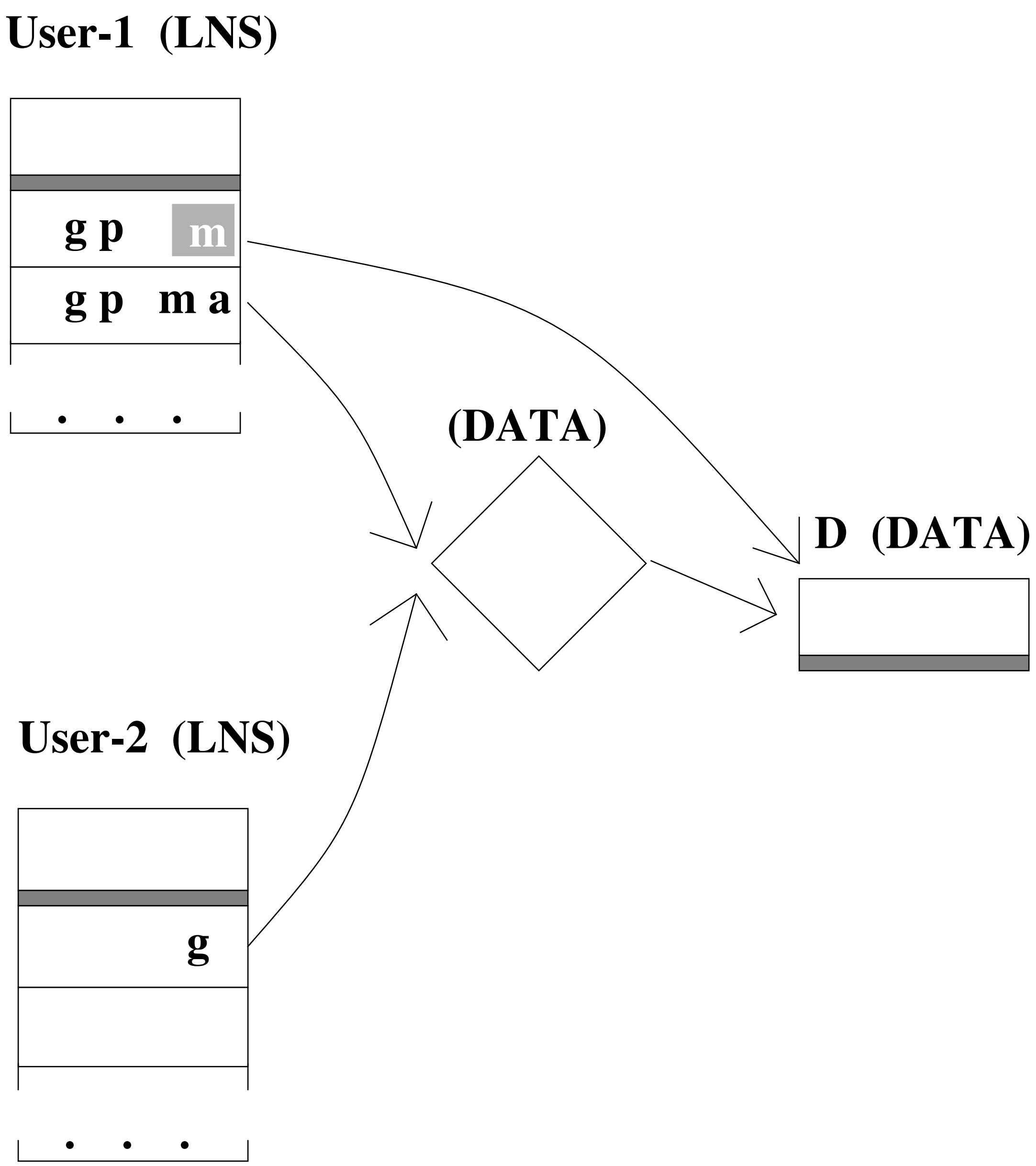
Initialisierung

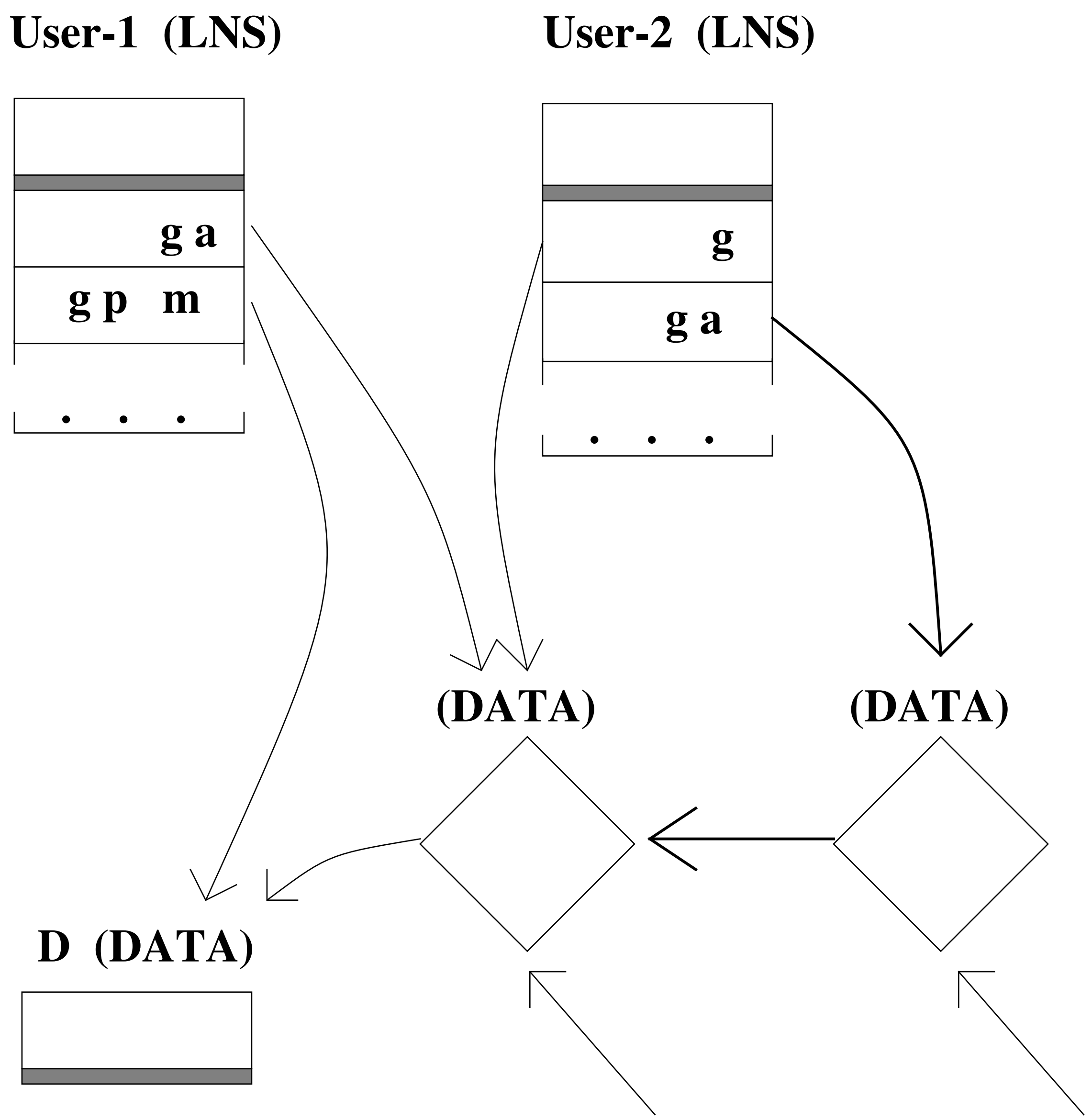
- Aufgabenstellung
  - Es soll sichergestellt werden, daß bei der Schaffung neuer Objekte weder Capabilities für das Objekt selbst noch für Objekte, die über seine C-Liste erreichbar sind, einem anderen Benutzer zugänglich gemacht wurden.
- Lösung
  - Aufruf der Erzeugungsprozedur ohne MDFYRTS und ohne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

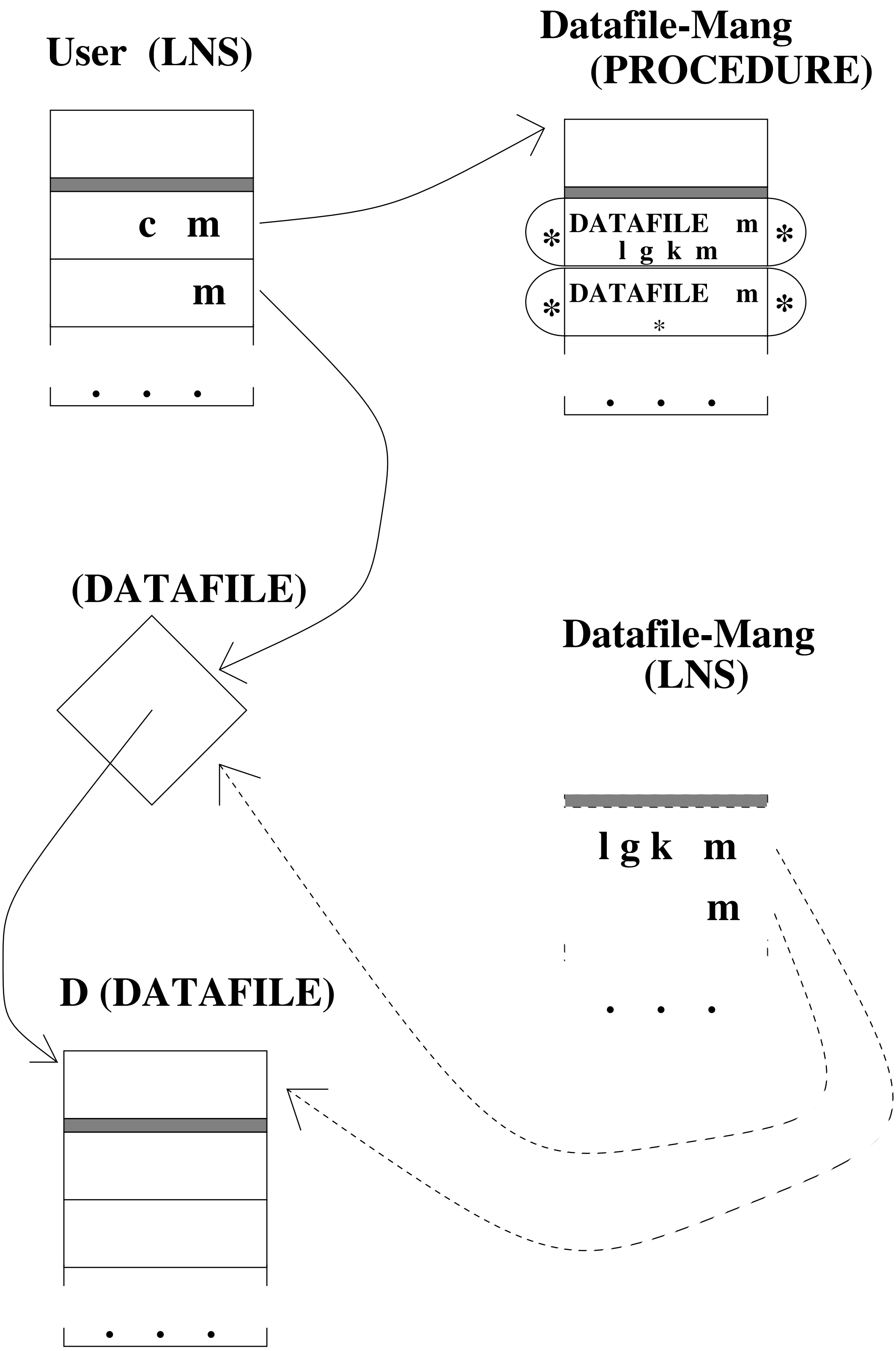

Datafile-Init
(PROCEDURE)
User (LNS)
c e
m
g e
DATAFILE m
l s k m
Old (DATA)
Demonic-User
(LNS)
Datafile-Init
(LNS)
(DATAFILE)
g e
l s k m
?

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)
User-2 (LNS)
g a
g p m
. . .
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

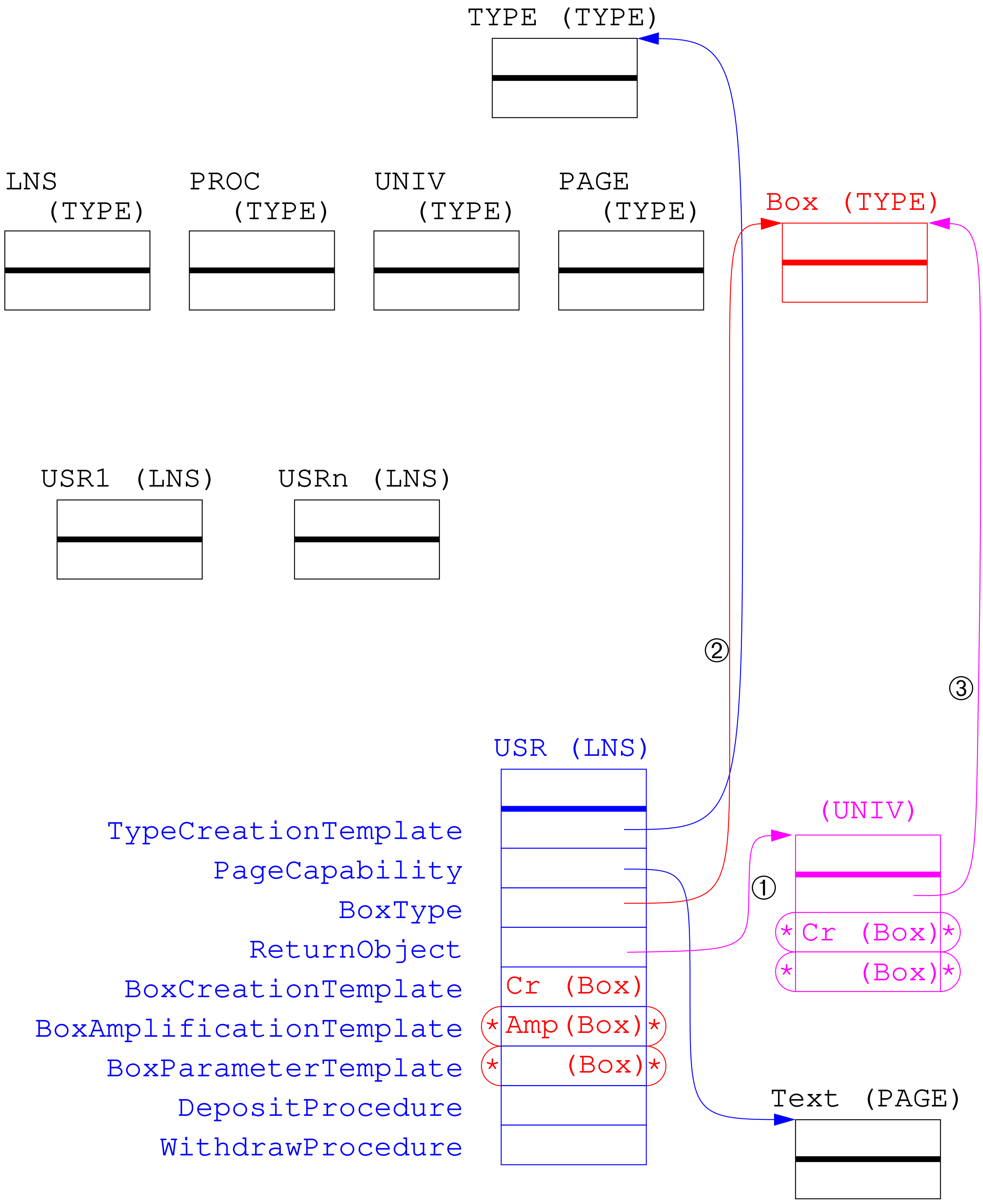
```
const
    AllRights: Rights := [[1,1,1,...,1],[1,1,1,1,1,1,1,1]];
    NoAuxRights: AuxiliaryRights := [0,0,0,0,0,0,0,0];
    NoRestrictions: Rights := AllRights;
```

❏ 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
```

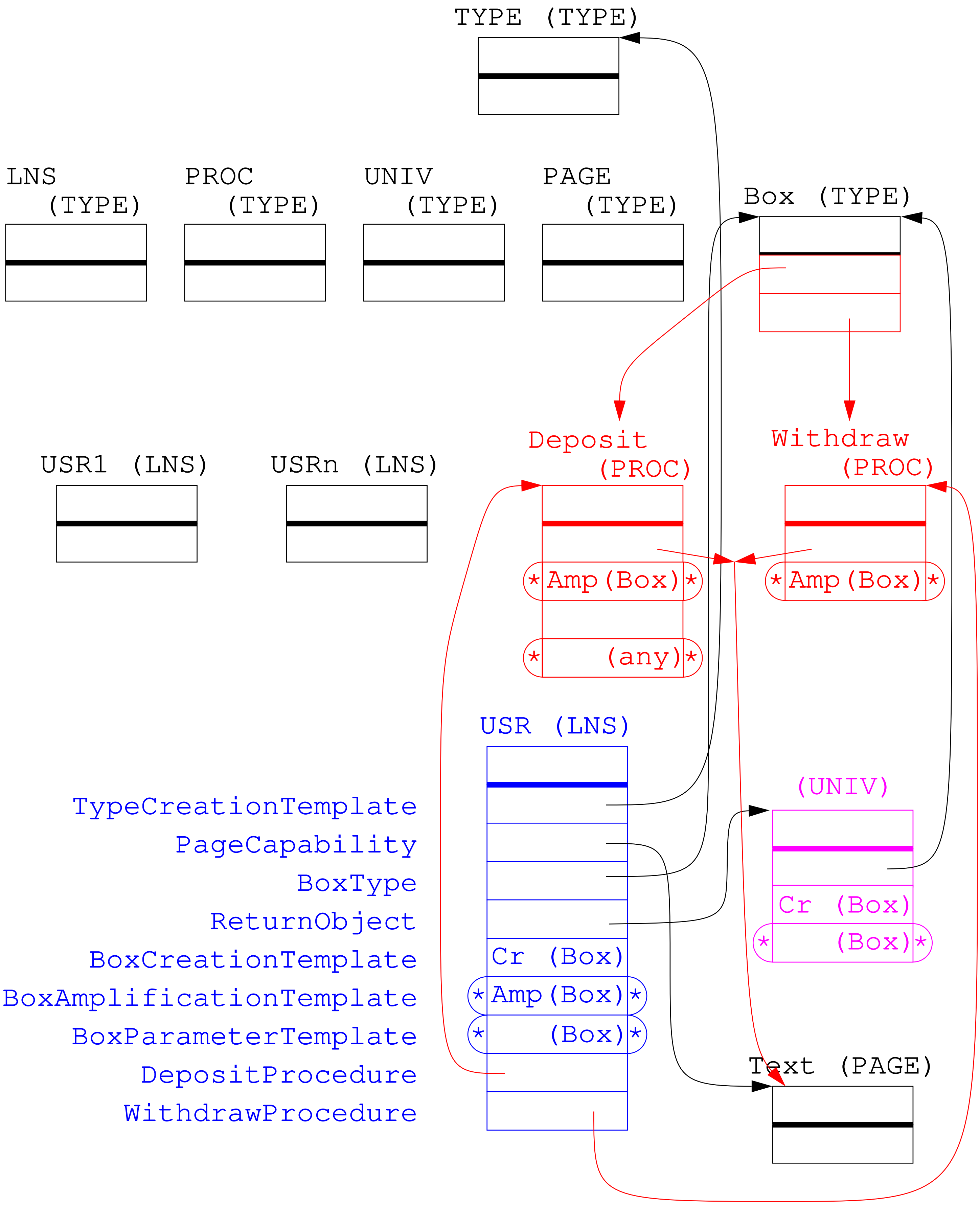
```
    BoxType = 3;
    ReturnObject = 4;
    BoxCreationTemplate = 5;
    BoxAmplificationTemplate = 6;
    BoxParameterTemplate = 7;
    DepositProcedure = 8;
    WithdrawProcedure = 9;

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)
TypeCreationTemplate
PageCapability
BoxType
ReturnObject
BoxCreationTemplate
BoxAmplificationTemplate
BoxParameterTemplate
DepositProcedure
WithdrawProcedure
Cr (Box)
* Amp (Box) *
* (Box) *
(UNIV)
①
* Cr (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)
USR1 (LNS)
USRn (LNS)
Deposit (PROC)
Withdraw (PROC)
* 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);
```