4.5 Capabilities

Konzeptionell speichert das Schutzsystem die Schutzmatrix und fungiert als Überwachungsinstanz *(reference monitor)* für die Zugriffsversuche.

De facto wird die Matrix – die dünn besetzt ist! – nicht als solche gespeichert, sondern effizient in unterschiedlicher Weise repräsentiert – in Abhängigkeit von den Änderungsoperationen, die effizient unterstützt werden sollen.

Schutzstatus speichert die Rechte **spaltenweise** – bei den Objekten

Capabilities *(auch Berechtigungen, Anrechte, Zugriffsausweise)* speichern die Rechte **zeilenweise** – bei den Subjekten

Def.: **Capability** = *Objektverweis + Rechte*

wobei der Objektverweis *fälschungssicher (unforgeable)* ist
und die Rechte häufig (nicht immer) *typgebunden* sind

Repräsentation der Zugriffsmatrix:

Jedem Subjekt ist **Berechtigungsliste** *(capability list, CL)* zugeordnet,
z.B. bei generischen Rechten R,W:

(Index identifiziert Berechtigung)

	Objekt	R	W
0			
1			
2			
3			

4.5.1 Operationen auf Berechtigungen

Änderung des Schutzzustands durch Änderung von Berechtigungslisten –
Erzeugen, Löschen, Kopieren, Modifizieren von Berechtigungen:

- **Erzeugen** bei Objekterzeugung
- **Löschen** durch Überschreiben, z.B. mit *null*
- **Kopieren** durch spezielle Kopieroperation
  ermöglicht Weitergabe an anderes Subjekt
- **Einschränken** durch spezielle Modifizieroperation
  ermöglicht *(nur)* Verringerung der Rechte,
  z.B. vor Weitergabe an anderes Subjekt
- **Erweiterung** *(capability amplification)* durch *templates*
  in aufgerufenem Code

Benutzung einer Berechtigung beim Objektzugriff
(in hochsprachlicher Formulierung):

objektorientiert:

```
Object obj;
.....
obj.op(args);
```

wird erlaubt, wenn in `obj` das `op`-Bit gesetzt ist

prozedural:

```
Object obj;
.....
proc(obj,args);
```

wird erlaubt, wenn in `obj` das `proc`-Bit gesetzt ist

Vorteile von Berechtigungen gegenüber *Schutzstatus*:

→ restriktive Rechtevergabe: Subjekt hat nur die Rechte,
die ihm bei seiner Erzeugung übergeben werden
bzw. die er später von anderen Subjekten erhält

→ einfache, „natürliche" Weitergabe von Rechten

→ kontrollierte Rechte-Modifikation

➜ Sicherheitsprinzipien
- *need-to-know,*
- *attenuation of privileges*
- *controlled amplification*

Nachteil:

→ Entzug von Berechtigungen *(capability revocation)*
ist nicht möglich – oder schwierig zu realisieren

→ Proliferation von Berechtigungen schwer zu kontrollieren

4.5.2 Hardware Capabilities

Capability-Based Addressing (Dennis/van Horn, MIT 1966!):

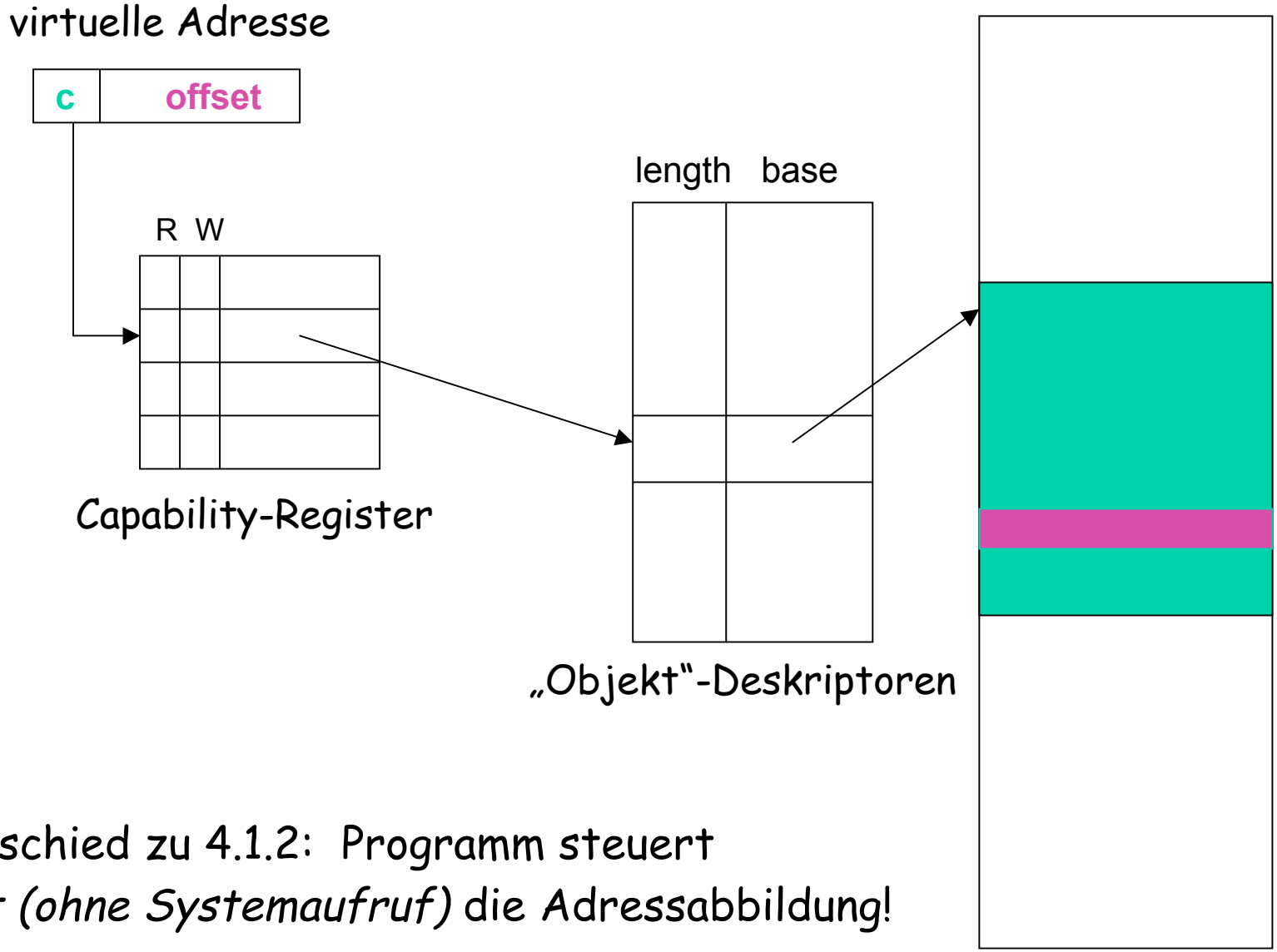
- ☛ capability = Nummer eines Speicherbereich-Deskriptors
  (= Adresse + Länge)
  + Rechte R/W

- ☛ capabilities sind hardware-getypt *(tagged architecture),*

- ☛ capabilities werden von speziellen Maschinenbefehlen manipuliert

- ☛ capabilities werden vom Programm nach Belieben
  in Capability-Register geladen,

- ☛ wirken dort bei der Umsetzung der virtuellen Adressen mit:
  „Segmentnummer" ist hier Capability-Register-Nummer



Unterschied zu 4.1.2: Programm steuert
selbst (ohne Systemaufruf) die Adressabbildung!

4.5.3 Software Capabilities

bedeutet: - Betriebssystem verwaltet Berechtigungen für Objekte,
- stellt Systemaufrufe für ihre Manipulation zur Verfügung

Schutz der Berechtigungen vor direkter Manipulation durch Speicherung

❶ bei prozessspezifischen Systemdaten (Berechtigungsliste) oder

❷ (flexibler) in verschatteten Bereichen der Prozesssegmente oder

❸ (sehr flexibel) kryptographisch versiegelt

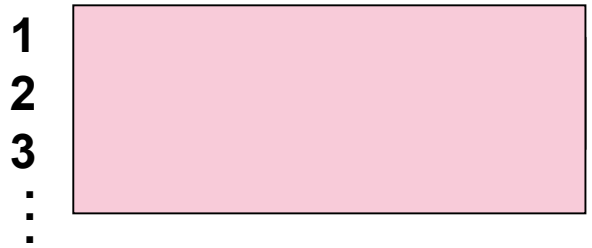
Identifikation einer Berechtigung durch

❶ Index in Berechtigungsliste bzw.

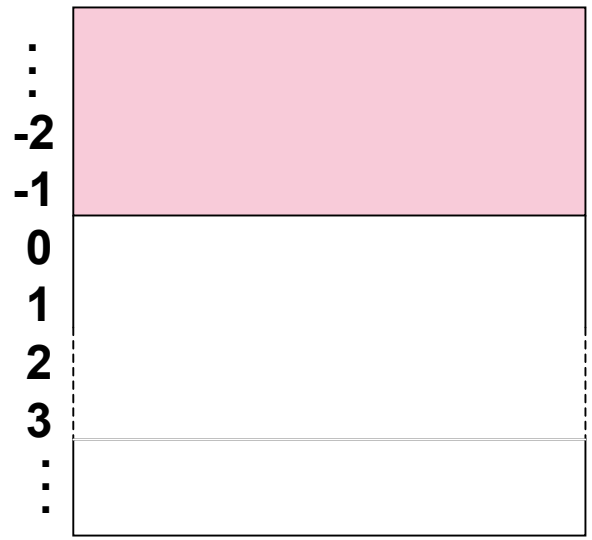
❷ Segmentnummer + Index bzw.

❸ normale Adresse

❶ *Berechtigungsliste:*

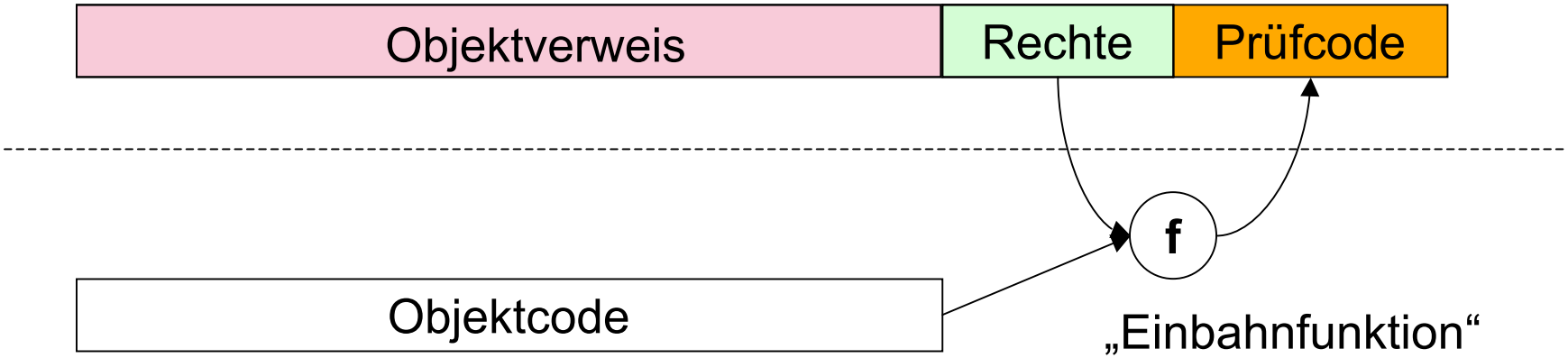


❷ *Verschattete Bereiche:* innerhalb eines Segments Identifizierung „negativ interpretierten" Indizes:



❸ *Kryptographische Versiegelung:*

Betriebssystem führt für jedes Objekt (geheimen) *Objektcode*



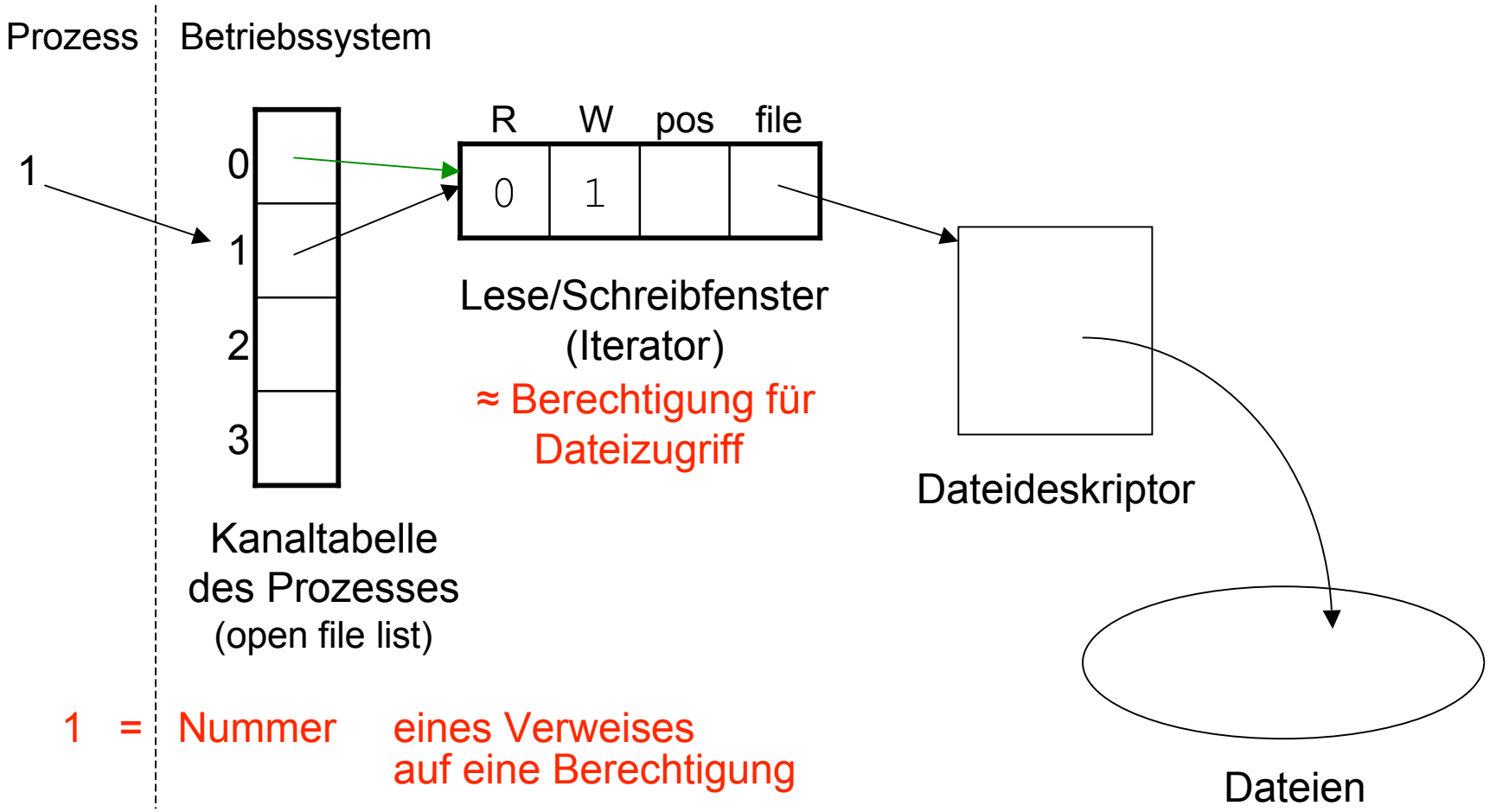
Manipulierte Berechtigung wird vom Betriebssystem erkannt, indem f angewendet und das Ergebnis mit dem Prüfcode verglichen wird.

Eignung der drei Alternativen für

	Rechte-Weitergabe über		Vererbung an Kindprozess
	Interprozess-kommunikation	gemeinsames Segment	
❶	-	-	✓
❷	-	✓	✓
❸	✓	✓	-

4.5.3.1 Beispiel Unix

Zur Erinnerung: Unix-Kanalnummer *(file descriptor)*,
z.B. 0,1 für Standard-Ein/Ausgabe:



Erzeugen einer solchen „Quasi-Berechtigung" (Iterator mit Rechten):

```
fildes = open(path, oflag)
```

- setzt R,W gemäß `oflag`,
- setzt Lese/Schreibposition pos auf 0,
- setzt Verweis auf Deskriptor der Datei `path`
- erzeugt und liefert Verweis auf Iterator

Vor.: Dateizugriff gemäß `oflag` *erlaubt (4.3.1.3) !*

Benutzung durch Lese/Schreiboperationen:

```
read(fildes, &buffer, bufsize)
```

Vor.: R gesetzt in `fildes`

```
write(fildes, &buffer, bufsize)
```

Vor.: W gesetzt in `fildes`

Kopieren eines Berechtigungs-*Verweises (!)*:

`newfildes = dup(oldfildes)`

liefert Nummer der Kopie von `oldfildes`

Vor.: `oldfildes` *ist nicht* `null`

`dup2(oldfildes, newfildes)`

kopiert nach `newfildes`

Vor.: `oldfildes` *ist nicht* `null`

`close(fildes)`

ersetzt Verweis durch `null` und . . .

Vor.: *Verweis ist nicht* `null`

... und von Prozess zu Prozess bei `fork()`:

Erzeuger-Prozess *vererbt* Kanaltabelle an erzeugten Prozess

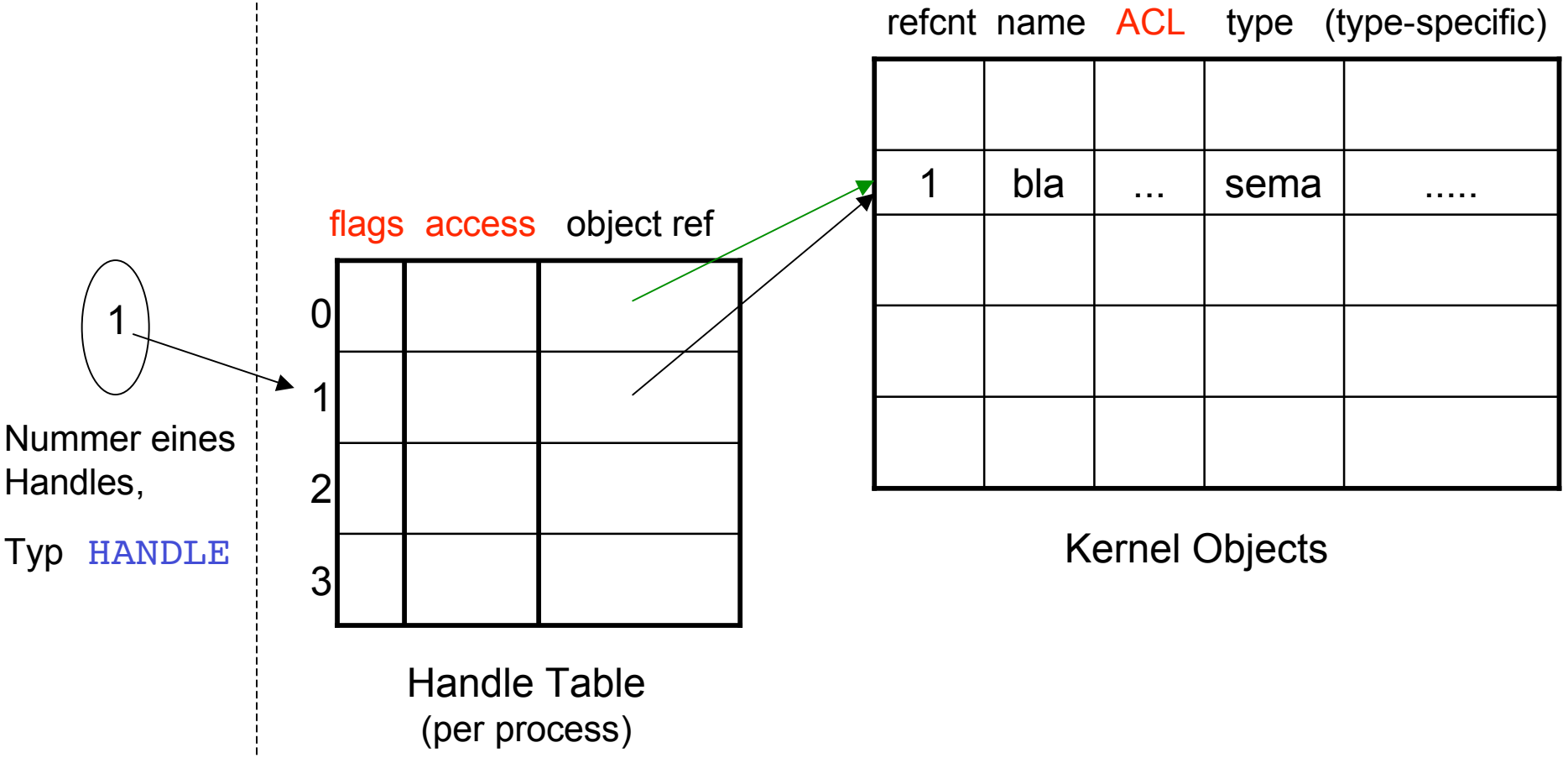
4.5.3.2 Beispiel Windows

- Zugriff auf **Systemobjekte** über **Handles** = Berechtigungen
  mit *typgebundenen*, *generischen* und *halbgenerischen* Rechten
  (mit benannten Zahlen durchnumeriert)

- **Typen** der Objekte: Prozess, Semaphor, Ereignis, Kanal, ...

- Objekt kann bei Erzeugung mnemonischen **Namen** erhalten
  (kein Pfadname, aber Präfix `Global\` oder `Local\` möglich)

- **Öffnen** eines benannten Objekts liefert Handle



-1 bezeichnet *Pseudo Handle* für laufenden Prozess:
beinhaltet *alle Rechte* am Prozess

Generische Rechte bei Handles:

objektbezogen (vgl. 4.3.2.2):

delete	`DELETE`
readControl	`READ_CONTROL`
writeACL	`WRITE_DAC`
writeOwner	`WRITE_OWNER`

handle-bezogen (! vgl. 4.4.2):

inherit	`HANDLE_FLAG_INHERIT`
close	`HANDLE_FLAG_PROTECT_FROM_CLOSE`

Halbgenerische Rechte bei Handles:

Beispiel für Synchronisationsobjekte:

synchronize	`SYNCHRONIZE`

Beispiel Semaphor:

```
HANDLE    CreateSemaphore(LPSECURITY_ATTRIBUTES mode,
                          LONG init, LONG max, LPCTSTR name)
```

erzeugt und *öffnet* Objekt vom Typ *Semaphore*
mit *Schutzstatus* `mode` und *Namen* `name` und
liefert Berechtigung *mit allen Rechten*

```
HANDLE    OpenSemaphore(DWORD access, BOOL inherit,
                        LPCTSTR name)
```

öffnet Objekt vom Typ *Semaphore* und
liefert Berechtigung mit Rechten gemäß `access`

Vor.: Objekttyp ist tatsächlich *Semaphore* und
Schutzstatus erlaubt gewünschten `access`

Nichtgenerische **Rechte** für Semaphore: `SYNCHRONIZE`
`SEMAPHORE_MODIFY_STATE`

Typische **Operationen**:

```
DWORD WaitForSingleObject(HANDLE handle,
                          DWORD timeout)
```

ist P-Operation, falls `handle` auf ein Semaphor verweist

Vor.: `handle` beinhaltet Recht `SYNCHRONIZE`

```
BOOL ReleaseSemaphore(HANDLE handle,
                      LONG   count,
                      LPLONG oldcount)
```

ist verallgemeinerte V-Operation

Vor.: `handle` verweist auf ein Semaphor und
beinhaltet Recht `SEMAPHORE_MODIFY_STATE`

Generische Operationen (*auf Handles* von Objekten):

```
BOOL CloseHandle(HANDLE handle)
```

löscht die Berechtigung `handle`
(und gegebenenfalls das Objekt)

```
BOOL GetHandleInformation(HANDLE handle,
                          LPWORD flags)
```

liefert in `flags:` `HANDLE_FLAG_INHERIT`
`HANDLE_FLAG_PROTECT_FROM_CLOSE`

```
BOOL SetHandleInformation(HANDLE handle,
                          DWORD mask, DWORD flags)
```

setzt/löscht die angegebenen Flags

... und **Kopieren/Weitergeben** von Handles:

```
BOOL DuplicateHandle(HANDLE sourceProcessHandle,
                     HANDLE sourceHandle,
                     HANDLE targetProcessHandle,
                     LPHANDLE targetHandle,
                     DWORD access,
                     BOOL inherit,
                     DWORD options)
```

erzeugt für den Prozess hinter `targetProcessHandle`

eine Kopie des `sourceHandle` im Prozess `sourceProcessHandle`,
eventuell mit geänderten Rechten/Flags!

In `targetHandle` wird die Nummer abgeliefert, mit der sich der Empfänger-Prozess auf das neue Handle beziehen kann.

! Information des Empfängers über Interprozesskommunikation !

Vor.: ☞ `sourceProcessHandle` beinhaltet das Recht `PROCESS_DUP_HANDLE`

☞ Rechte-Erweiterung nicht jenseits der gemäß Schutzstatus des Objekts gewährten Rechte

☞ ...

Beispiel **Dateizugriff**:

Datei öffnen = Objekt *Lese/Schreibfenster* erzeugen
und entsprechendes Handle liefern

```
HANDLE CreateFile(LPCTSTR path,
                  DWORD access, ...
                  LPSECURITY_ATTRIBUTES mode, ...)
```

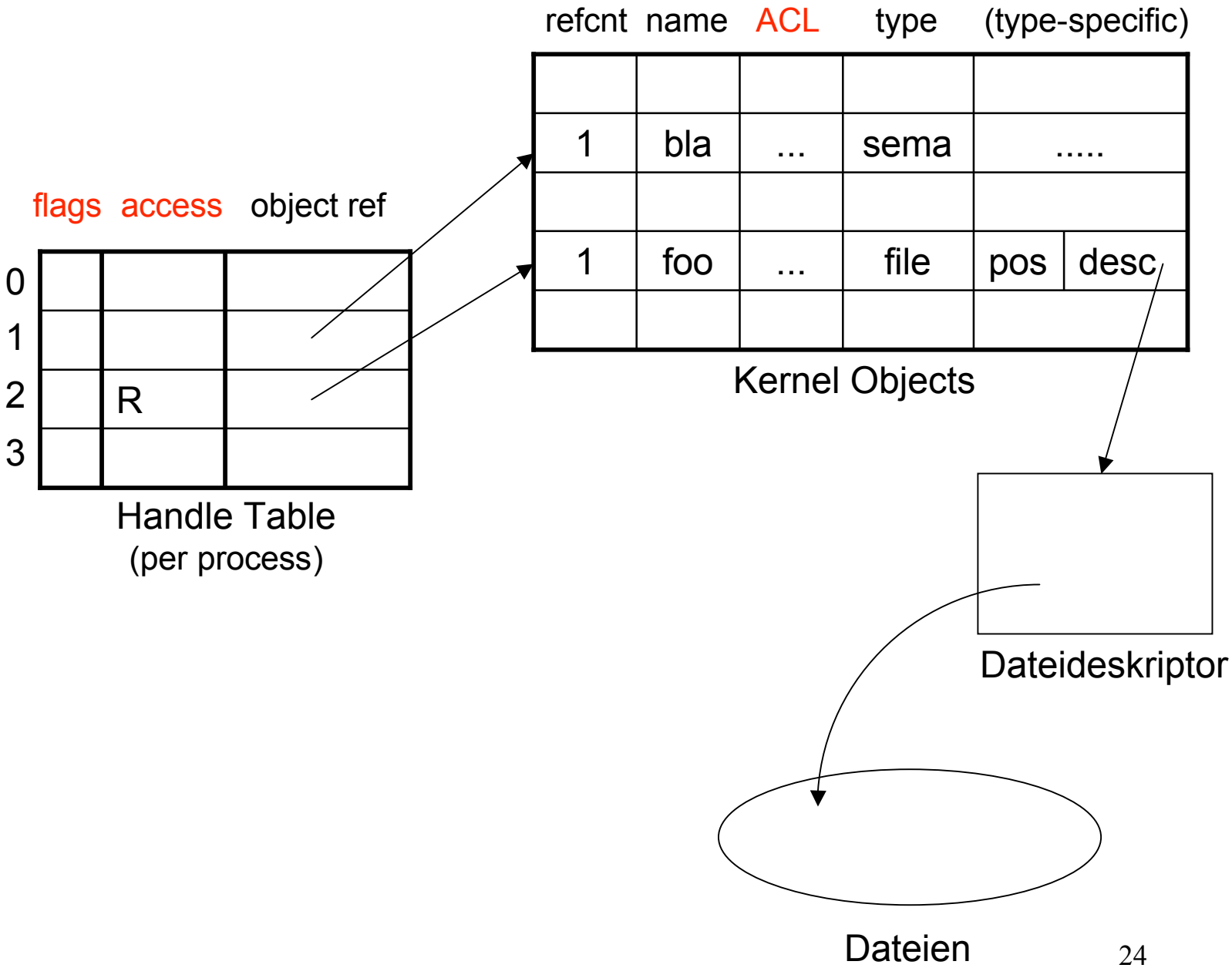
erzeugt und/oder öffnet Datei und erzeugt Handle

Vor.: Schutzstatus erlaubt gewünschten `access`

```
BOOL ReadFile(HANDLE fileHandle, ...)
```

liest aus Datei hinter `fileHandle`

Vor.: `fileHandle` beinhaltet Recht `GENERIC_READ`


refcnt
name
ACL
type
(type-specific)
1
bla
...
sema
.....
1
foo
...
file
pos
desc
Kernel Objects
flags
access
object ref
0
1
2
3
R
Handle Table
(per process)
Dateideskriptor
Dateien

4.5.3.3 Capability-basierte Betriebssysteme

- erlauben Zugriffe grundsätzlich *nur über Berechtigungen*
- erlauben strikte Umsetzung des *need-to-know-Prinzips*
- findet man fast nur in (Labor-)Systemen:

  Hydra (Carnegie-Mellon Univ., 1970-75)
  Amoeba (Vrije Univ. Amsterdam, 1987-98)
  Birlix (GMD Birlinghoven, 1987-95)
  Monads (Univ. Bremen, Ulm; Monash Univ., 1980-95)
  Eros (Univ. of Pennsylvania, 1999)
  .....

  aber auch beim System IBM AS/400

- häufig gleichzeitig mit *Objektbsierung* für Benutzerprozesse

4.5.4 Typische Anwendungen

Voraussetzung: Unix-ähnliches System
mit kryptographisch geschützten Berechtigungen

- Erzeugung eines Systemobjekts liefert Berechtigung mit allen Rechten,
- **`exec`** kann Berechtigungen als *Parameter* übergeben,
- auch *Code/Konstanten*-Segmente können Berechtigungen enthalten -
  für ausgezeichnete Standard-Systemobjekte, z.B.
- Dateisystem: fungiert als Fabrikobjekt für Lese/Schreibfenster
  und ist selbst nur über eine Berechtigung `files` erreichbar;
- Login Shell erhält Berechtigung `files` für das Dateisystem
  als Parameter übergeben, mit Rechten u.a. für **`open`** -
  erzeugt Lese/Schreibfenster und liefert zugehörige Berechtigung

Beispiel 1: Verhindern des Szenarios „Trojanisches Pferd `cp`
kopiert bei `cp from to` zusätzlich nach `~spy/to` "

Shell: kennt Parametrisierung von `cp` und macht daher

```
...
old = open(files, from, READ);
new = open(files, to, WRITE);

child = fork();
if  (child==0)
      exec(files, "/bin/cp", old, new);
else ...
```

Erzeugter Prozess führt `cp` aus -
und `cp` kann nur mit `read(old,..)` und `write(new,..)` arbeiten!

Beispiel 2: Alternative zum setuid-Bit, z.B. bei `/bin/passwd` :

Capability im Code-Segment

```
...
const pw = fileCap("/etc/passwd", READ+WRITE);
...
inout  = openRW(pw);
...
write(inout,..);
...
```

liefert Datei-Berechtigung für **openRW**,
z.B. bereits bei Übersetzung (**const** !),
sofern der Übersetzende (hier `root`)
die beiden gewünschten Rechte hat.

4.5.5 Schutzstatus versus Berechtigungen

Kriterium	**Schutzstatus**	**Berechtigungen**
object sharing	+	-
revocation	+	-
least privilege	-	+
Rechteweitergabe	-	+
Berechtigte ermitteln	+	-
Rechte ermitteln	-	+