

# L Betriebssystemarchitekturen

---

## L.1 Entwicklung der Betriebssystemstrukturierung

---

- Monolithische Kerne
- Minimalkerne
- Objektbasierte, offene Betriebssysteme

# 1 Literatur

---

- BAL+90.** B. N. Bershad, T. E. Anderson, E. D. Lazowska and Henry M. Levy, "Lightweight Remote Procedure Call", *ACM Transactions on Computer Systems*, Vol. 8, No. 1, pp. 37-55, Feb. 1990.
- DLA+91.** P. Dasgupta, R. J. LeBlanc, M. Ahamed and U. Ramachandran, "The CLOUDS Distributed Operating System", *IEEE Computer*, Apr. 1991.
- GFWK02.** Michael Golm, Meik Felser Christian Wawersich, and Jürgen Kleinöder. The JX Operating System. Proc. of the 2002 USENIX Annual Technical Conference General Track 2002 USENIX Annual Technical Conference, June 10-15, 2002, Monterey, CA, pp. 45-58.
- HaK93.** G. Hamilton and P. Kougiouris, "The Spring nucleus: A micokernel for objects", Proceedings of the *Summer 1993 USENIX Technical Conference*, pp. 147 - 159, Cincinnati (OH, USA), Jun. 1993.
- Laz93.** E. Lazowska, "Recent Trends in Experimental Operating Systems Research", *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*, pp. 13 - 19, Ithaca (NY, USA), Aug. 1993.
- Yok92.** Yasuhiko Yokote, "The Apertos Reflective Operating System: The Concept and Its Implementation", *OOPSLA '92 – Conference Proceedings*, pp. 414-434, Vancouver (BC, Canada), published as *SIGPLAN Notices*, Vol. 27, No. 10, Oct. 1992.

## L.2 Monolithische Betriebssystemkerne

---

- Alle Betriebssystemkomponenten zu einem Kern zusammengefasst
- Ausführung generell im privilegierten Modus

### 1 Vorteile

---

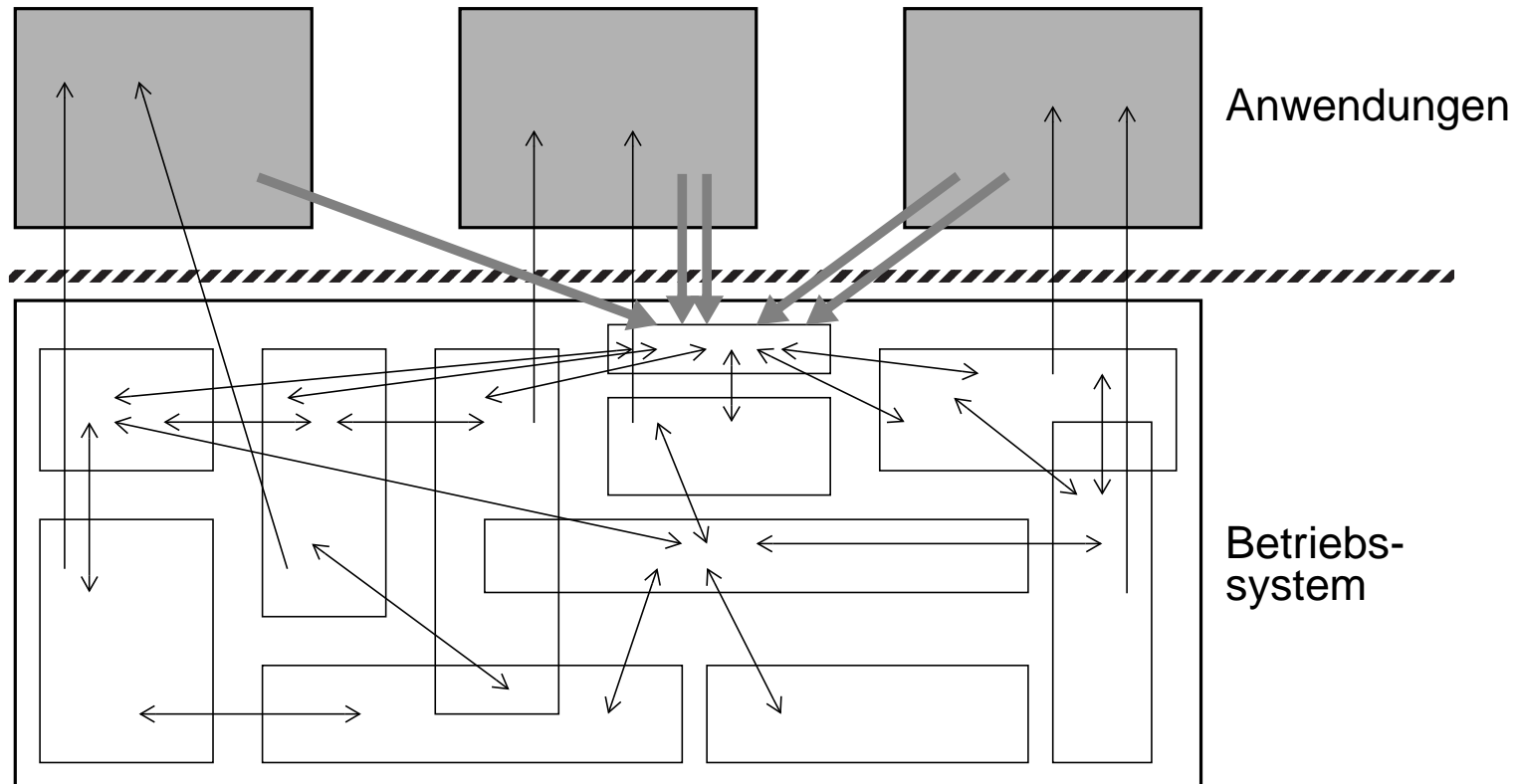
- + Effiziente Kommunikation zwischen den Komponenten des Kerns durch Prozeduraufrufe
- + Direkter Zugriff auf alle Datenstrukturen des Kerns jederzeit möglich
- + Privilegierte Befehle jederzeit aufrufbar

## 2 Nachteile

---

- Mechanismen strikt vorgegeben; individuelle Anpassung an Anwendungen, Erweiterung oder Reduzierung nicht möglich
- Strukturierung meist schlecht → änderungsunfreundlich und fehleranfällig
- Kein Schutz gegenüber Kernkomponenten möglich (kritisch z. B. bei zugekauften Treibern!)
- Niedriges Abstraktionsniveau  
Programmierung “direkt auf der nackten Hardware”

### 3 Gesamtablauf



→ billige, ungeschützte Interaktion  
 → teuere, gesicherte Interaktion

■ Anwendung  
 □ Betriebssystemfunktion

## L.3 Minimalkerne

---

- Auslagerung von Betriebssystemkomponenten
  - Platzierung der Komponenten in eigenen Adressräumen
  - Ausführung im Benutzermodus
  
- Reduktion des Kerns auf Basisfunktionalität und die Mechanismen, für die privilegierter Modus notwendig ist

# 1 Vorteile

---

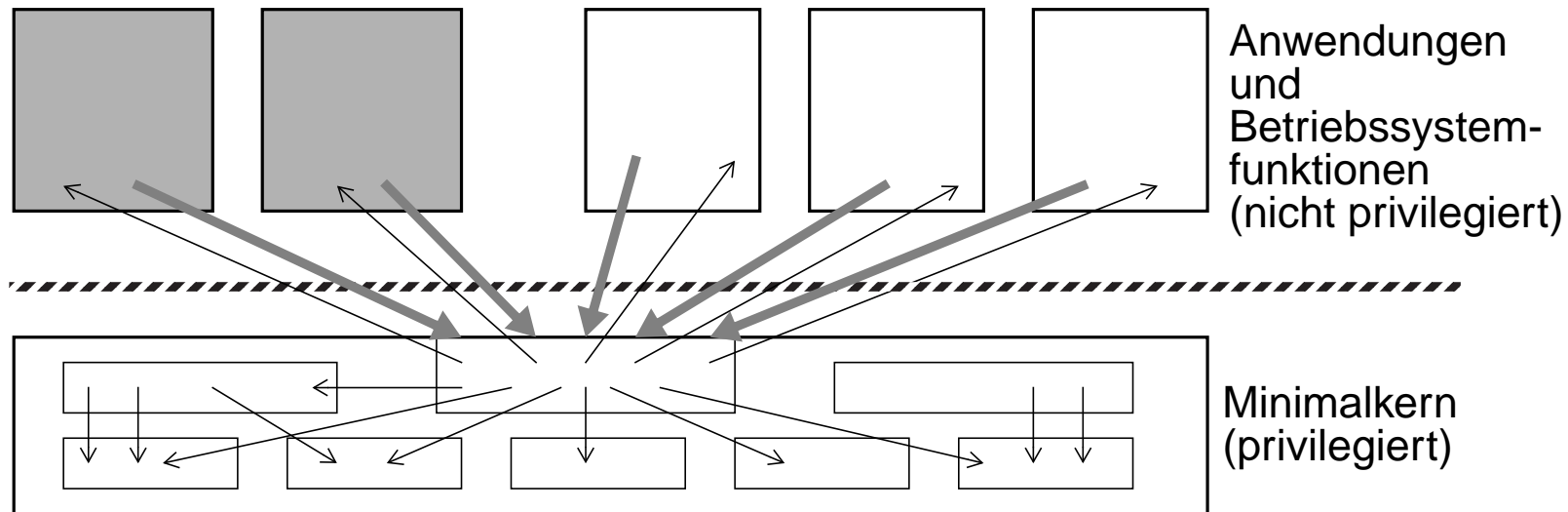
- + Bessere Modularisierung
  
- + Ausgelagerte Teile leichter individuell zu gestalten
  - zusätzlich angepasste Module erzeugbar
  - nicht benötigte Module können einfach weggelassen werden
  
- + Ausgelagerte Teile werden nicht privilegiert ausgeführt
  - mehr Sicherheit

## 2 Nachteile

---

- Kommunikation zwischen Modulen teuer  
(RPC, 20- bis 70-facher Aufwand eines Prozeduraufrufs)
  - Module werden aus Effizienzgründen möglichst groß gehalten
  - innerhalb der Module Strukturierung beliebig
  - aus Effizienzgründen verbleiben Komponenten im Minimalkern, die nicht dorthin gehören

### 3 Gesamtablauf



—> billige, ungeschützte Interaktion  
 —> teure, gesicherte Interaktion

■ Anwendung  
 □ Betriebssystemfunktion

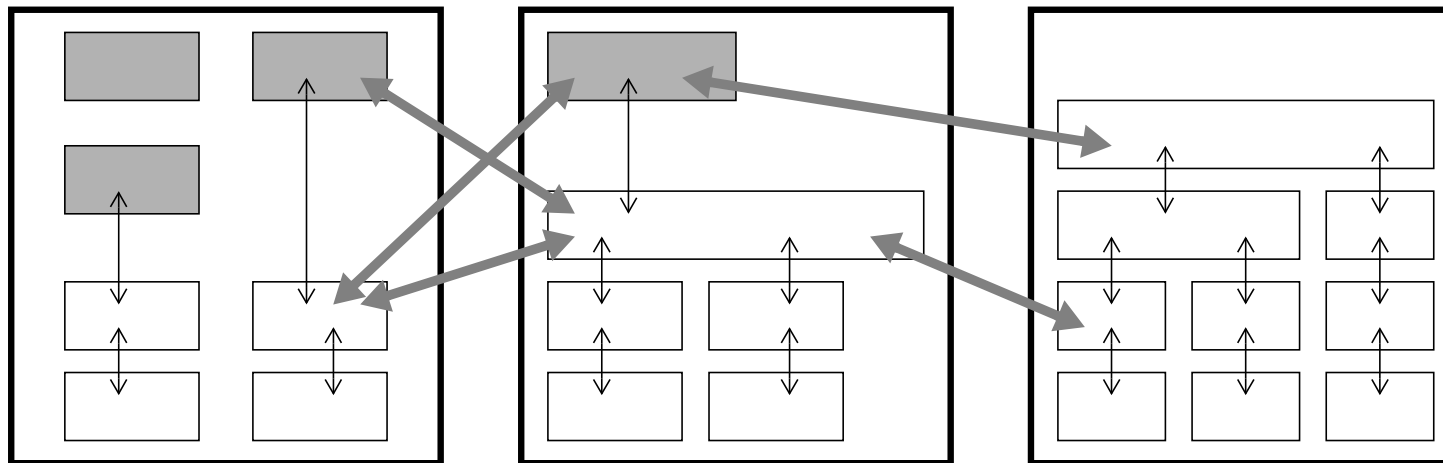
## L.4 Objektbasierte, offene Betriebssysteme

---

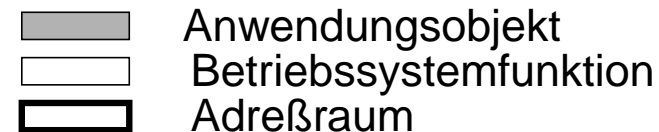
- Fortführung des Wegs 'monolithischer Kern → Minimalkern'
  
- Weitere Modularisierung nur möglich, wenn Inter-Modul-Kommunikationskosten reduziert werden
  - Adressraumkapselung muss aufgegeben werden
  - Verlust an Sicherheit
  - Sicherheit muß anders gewährleistet werden
  - Sprachebene + typsichere objektorientierte Programmierung
  
- Wenn Sicherheit auf Sprachebene garantiert wird, kann auf Adreßraumgrenzen verzichtet werden, sonst nicht!
  - gleiche Modul-Interaktion Adreßraum-lokal und Adreßraum-übergreifend wünschenswert

## L.4 Objektbasierte, offene Betriebssysteme (2)

- + Strukturierungsmöglichkeiten von Systemen mit ausgelagerten Komponenten mit Effizienz eines monolithischen Kerns kombinierbar
- + Individuelle Anpassbarkeit wie bei einem Minimal kern möglich  
→ durch feinere Granularität der Module jedoch weit flexibler



- billige, durch Objekt-Kapselung geschützte Interaktion
- teure, durch Adressraumgrenze geschützte Interaktion



# L.5 Hardware- vs. Softwareschutzkonzepte

---

## 1 Hardware-Schutz

---

- Virtuelle Adressräume durch MMU
  - + sehr effiziente Code-Ausführung
    - Überprüfung der Schutzkonzepte durch die Hardware
  - sehr grob-granular
    - Einheit: Seiten
    - Rechte: Lesen, Schreiben, Ausführen
  - kaum Information über Ausführungssemantik
    - nur primitive Operation (Lesen, Schreiben, Ausführen)
    - nur Überprüfung der Ortsinformation und der Art der Operation

## 2 Software-Schutzkonzepte

---

### ■ zentrale Grundlagen

#### ◆ Referenzen

- regeln **wo** man zugreifen darf

#### ◆ Typen

- regeln **was** man damit tun darf

### + sehr flexibel

- beliebig fein-granulare Einheiten
- beliebige Operationen abprüfbar

### – erhöhter Laufzeitaufwand

- Prüfung nur zum Teil statisch möglich
- Compiler muss Code für dynamische Prüfungen einflechten
- Laufzeitsystem muss evtl. zusätzliche Unterstützung leisten

### – Probleme mit niederen programmiersprachlichen Konzeptionen

# L.6 Objektorientierte BS: JX

---

## 1 Grundkonzepte

---

- Java-basiert
- Single-Adress-Space Betriebssystem
  - ◆ Schutz durch Sprachmechanismen statt durch MMU
    - Typkonzept
    - erweiterter Byte-Code-Verifier
    - spezielle Compiler-Unterstützung
  - ↳ feingranulare, flexible und effiziente Einheiten
- Betriebssystem dynamisch erweiterbar
  - ◆ Kapselung auch zwischen Betriebssystemmodulen
    - ↳ Erweiterungen muss nicht blind vertraut werden

## 2 Architektur

---

### ■ Domains

- Schutzeinheit
- Ressourcenverwaltung
- Eingrenzung von Fehlverhalten
- unabhängige Terminierung

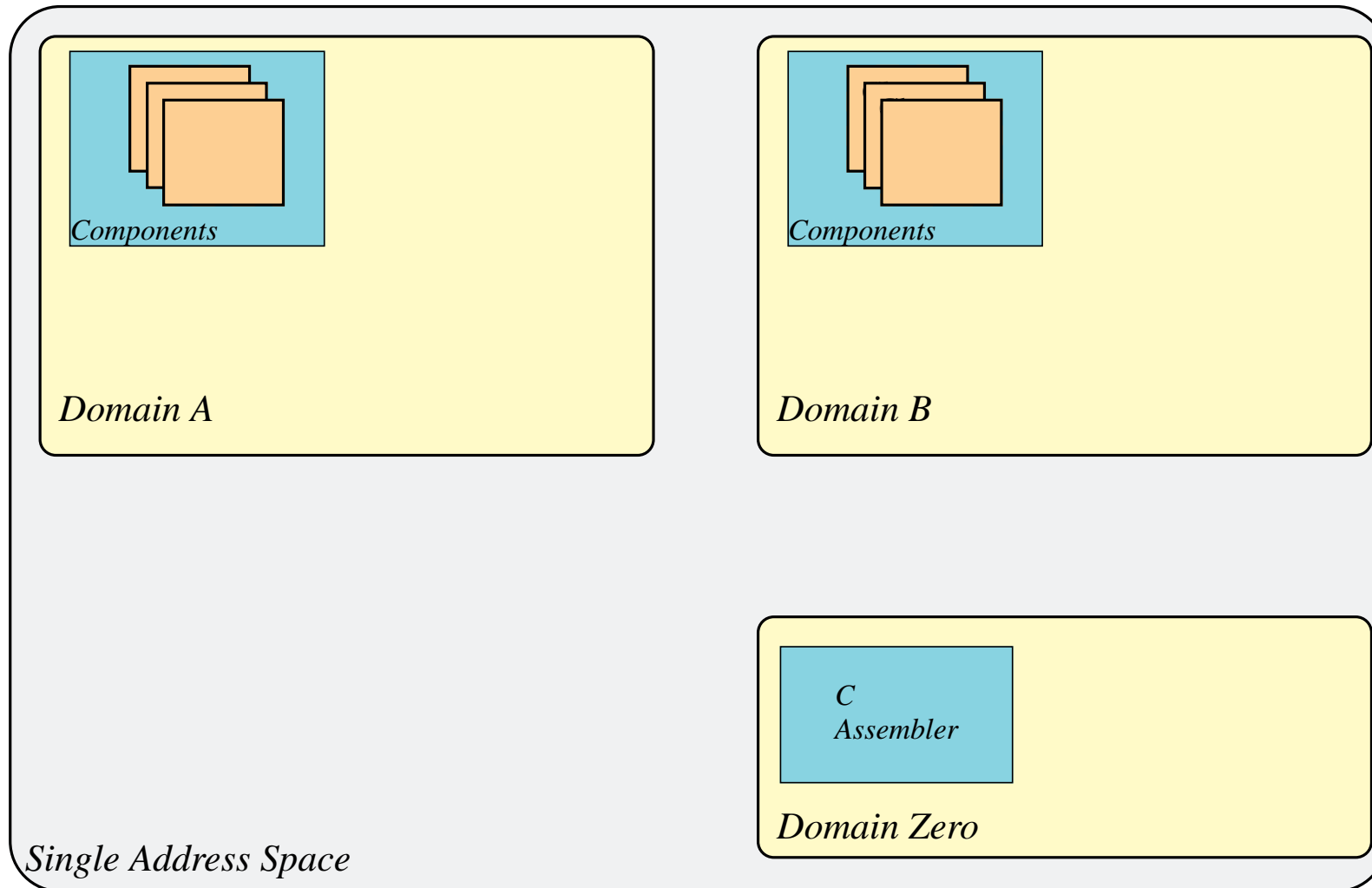
### ■ Komponenten

- Code-Module (Java-Bytecode), werden in Domain geladen
- Bytecode wird durch erweiterten Verifier überprüft
- werden zur Ladezeit in Maschinencode übersetzt

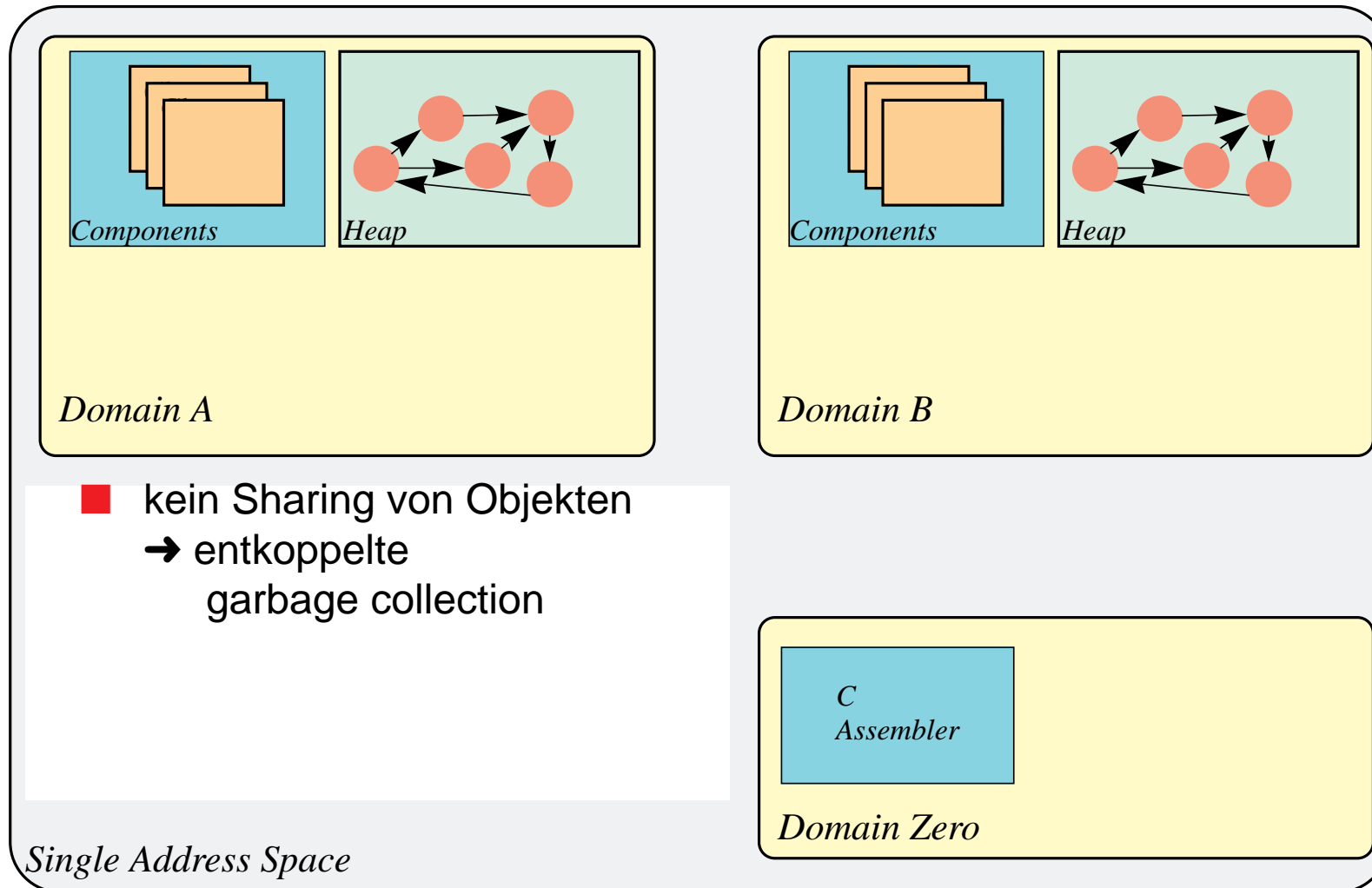
### ■ DomainZero

- enthält in C und Assembler geschriebene Komponenten
- kein Kern im klassischen Sinn — andere Domains setzen nicht darauf auf, sondern nutzen seine Dienste

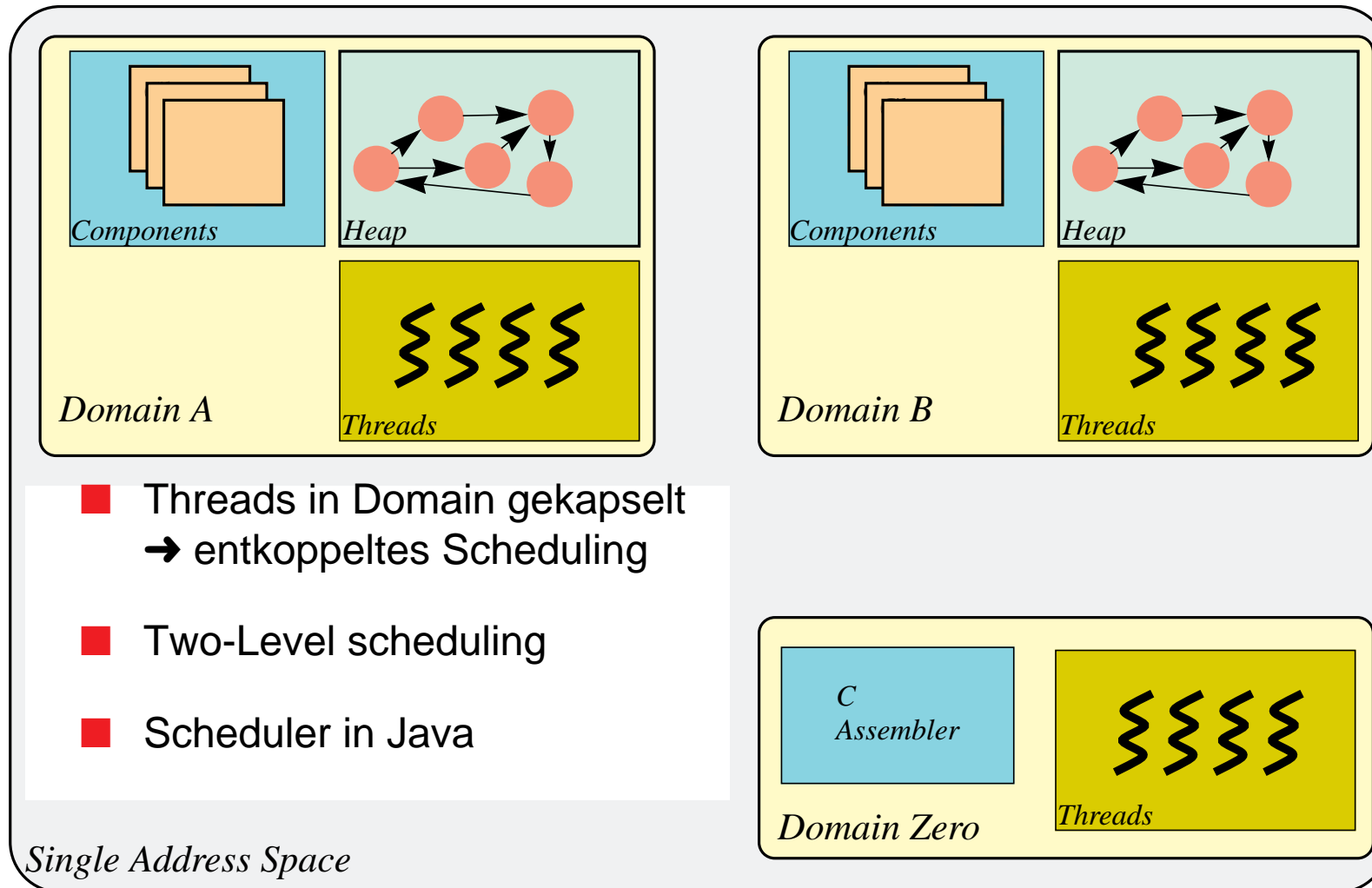
## 2 Architektur (2) — Domains und Komponenten



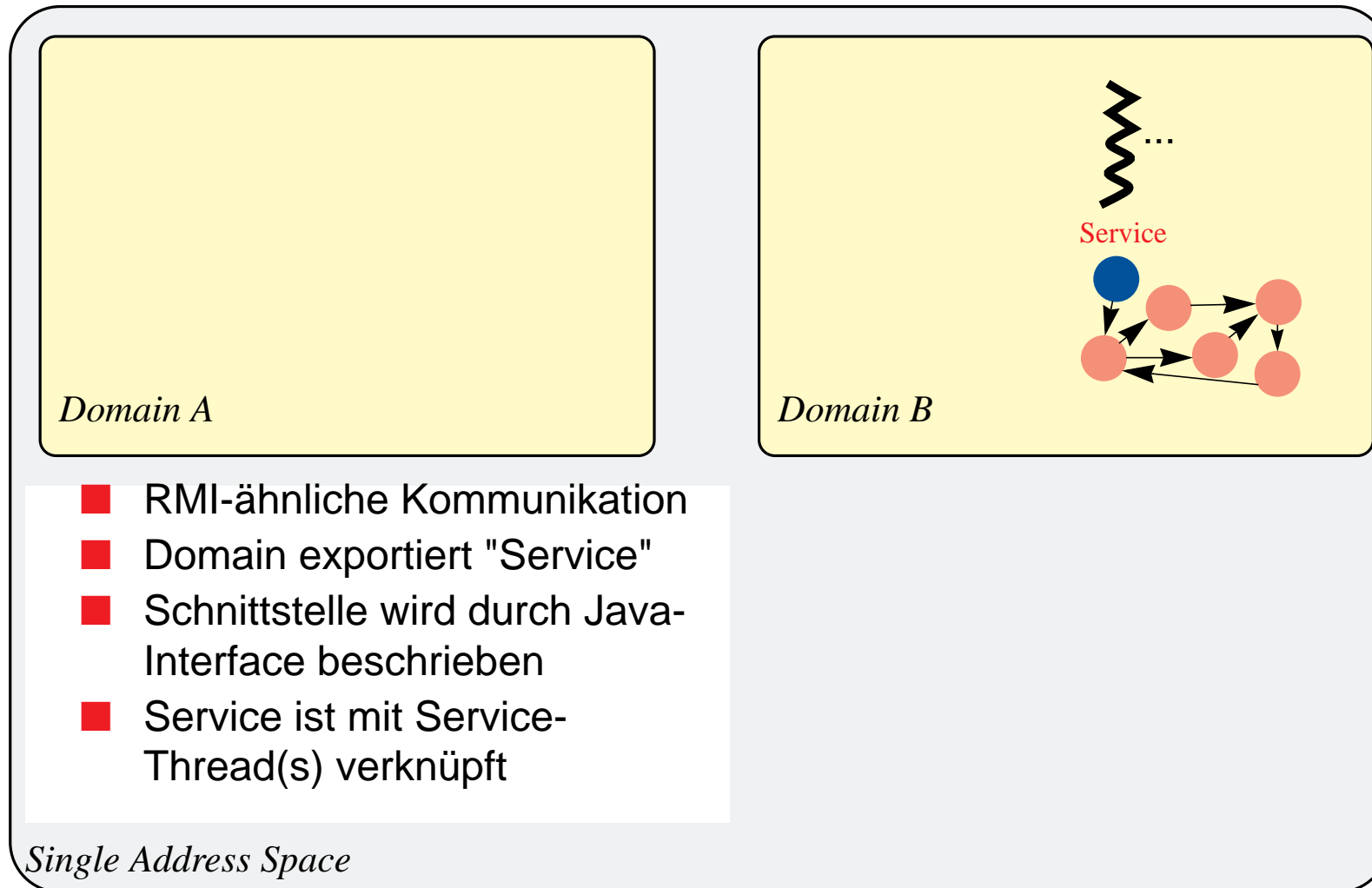
## 2 Architektur (3) — Objekte und Heap



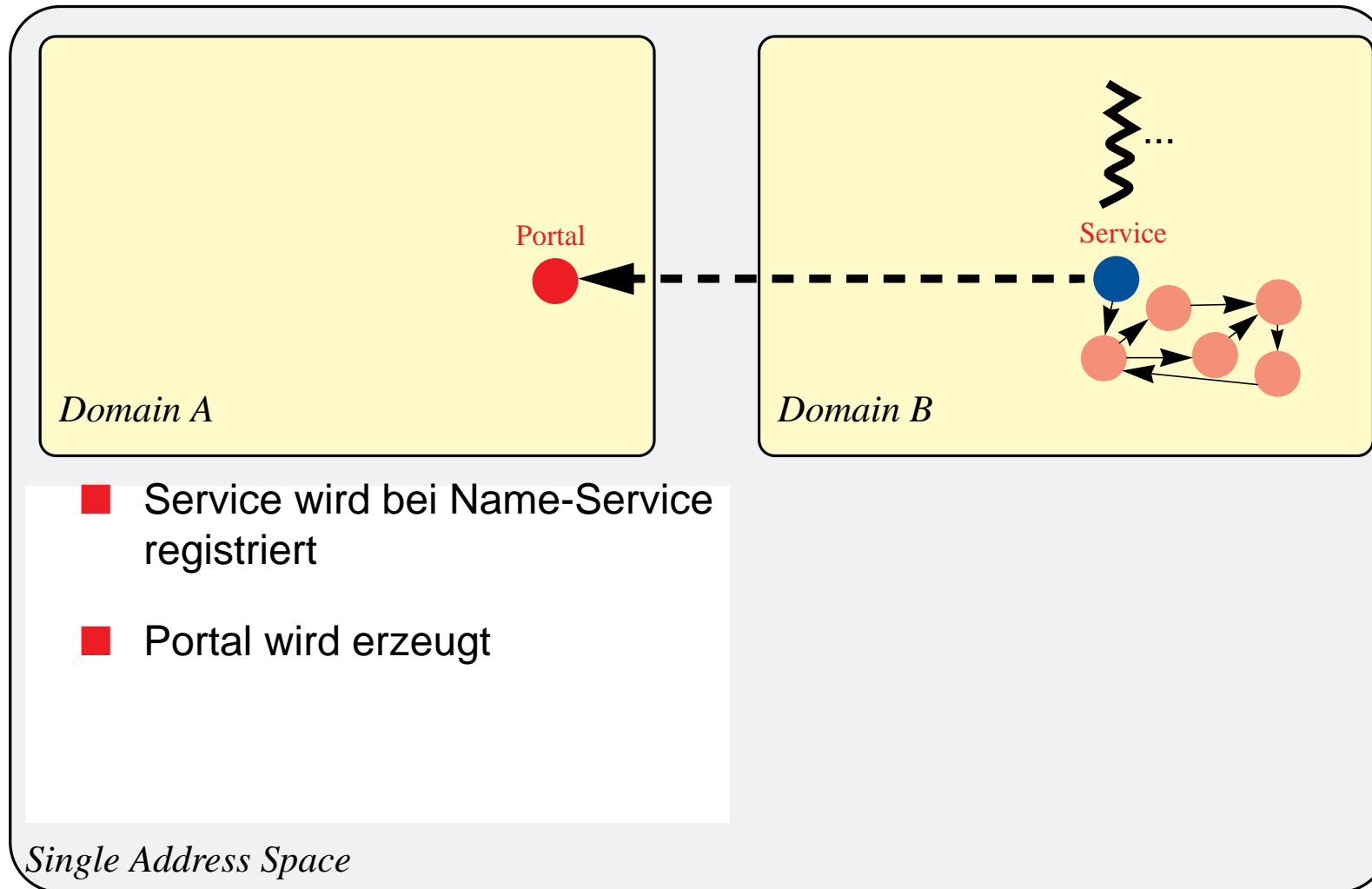
## 2 Architektur (4) — Threads



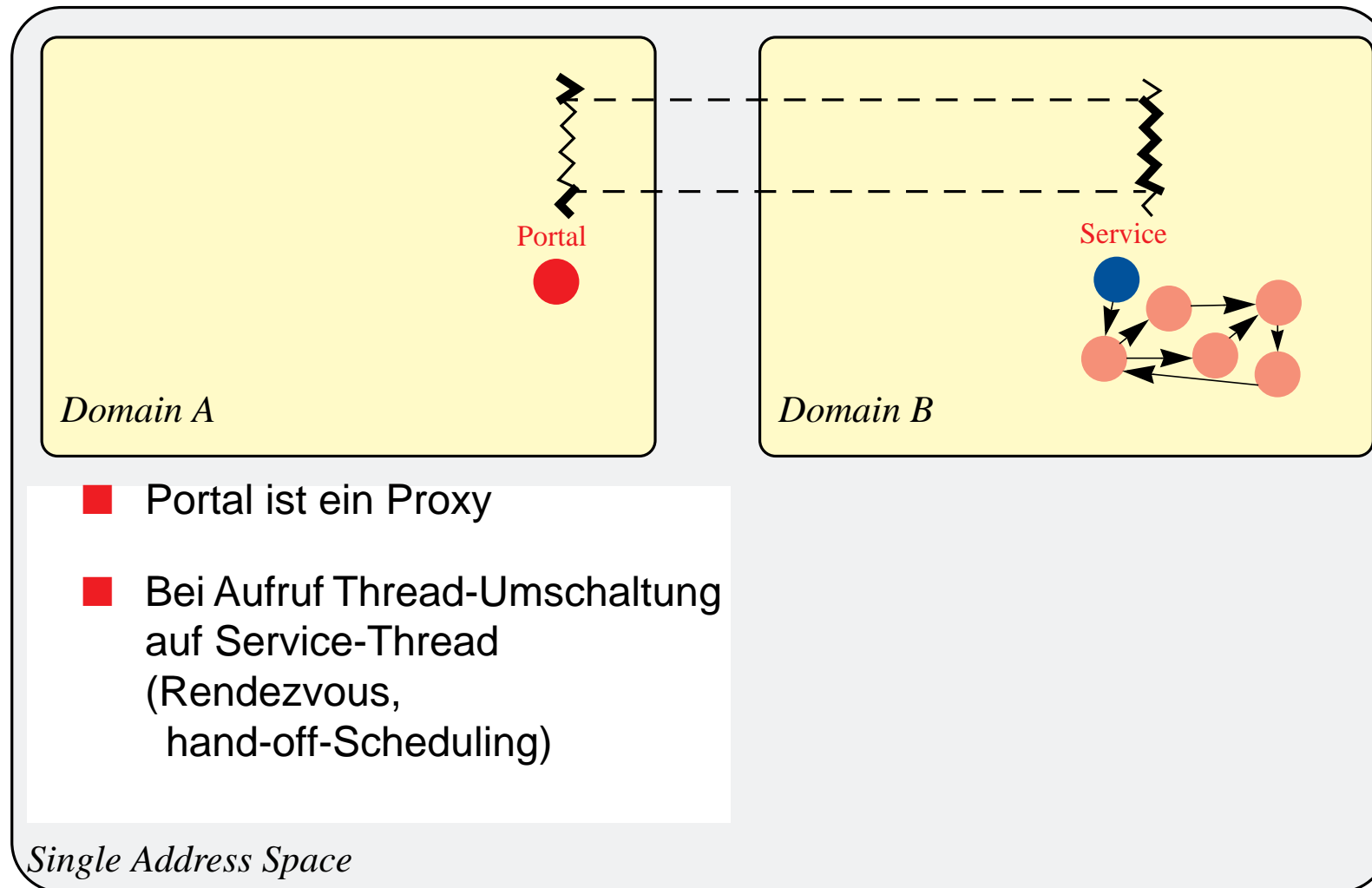
### 3 Inter-Domain-Kommunikation — Portale



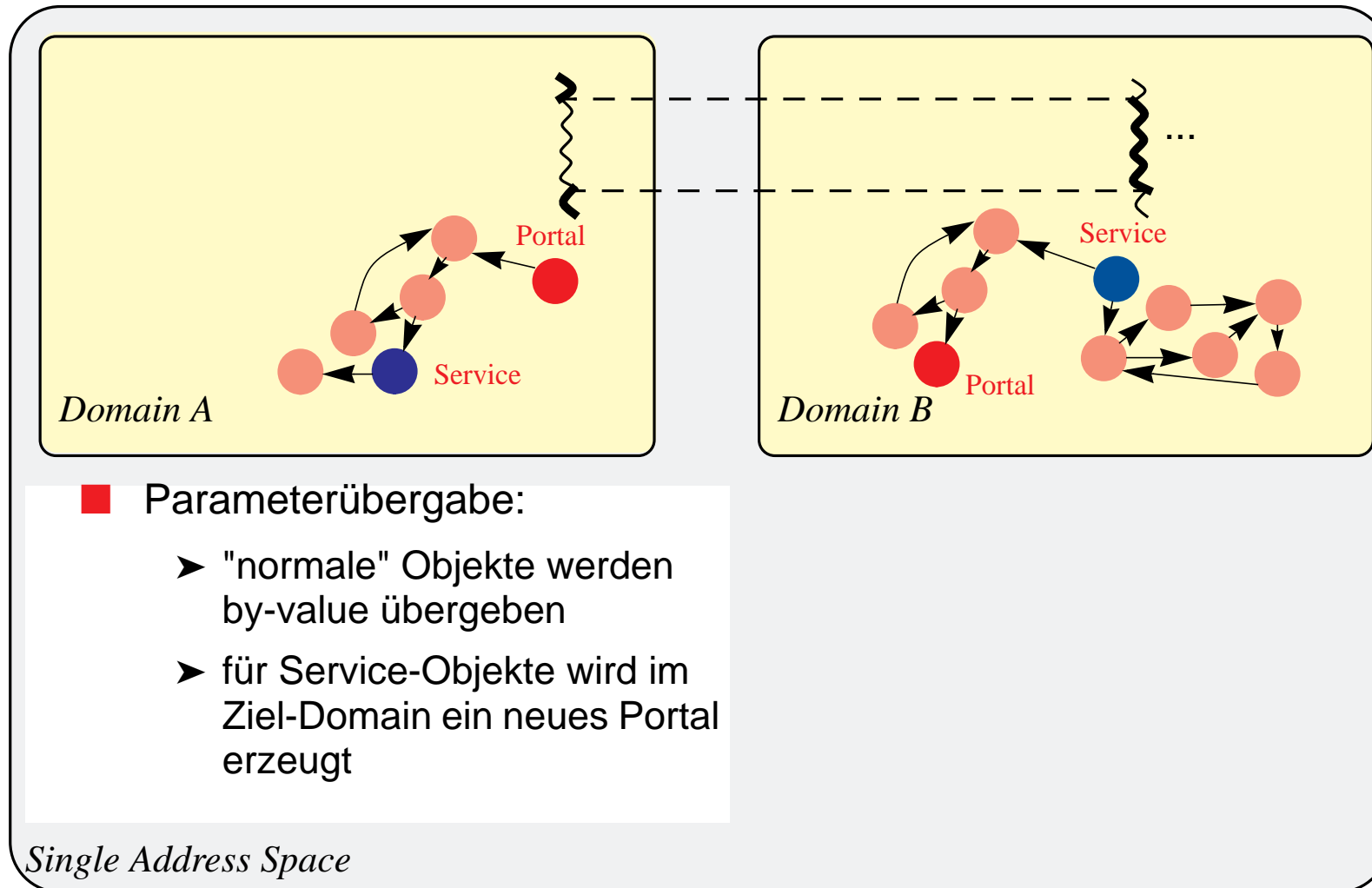
### 3 Inter-Domain-Kommunikation — Portale (2)



### 3 Inter-Domain-Kommunikation — Portale (3)



### 3 Inter-Domain-Kommunikation — Portale (4)



### 3 Inter-Domain-Kommunikation — Portale (5)

- Kosten im Vergleich zu IPC-Mechanismen in anderen Forschungsbetriebssystemen:

System	IPC (cycles)
L4KA (PIII, sysenter, sysexit) (PIII, 450MHz)	800
Fiasco/L4 (PIII 450 MHz)	2610
Alta/KaffeOS	27270
JX/native (PIII 500MHz)	691

- Kosten in JX immer noch Faktor 20 gegenüber normalem Methodenaufruf
  - aber reiner Softwareschutz
  - weitere Optimierungen möglich

## 4 Optimierungen: Fast Portals

---

- Methodenaufrufe an Domain Zero
  - ◆ Domain Zero muss ohnehin vertraut werden (C- oder Assembler-Code)
    - kontrollierte Domain-Grenze überflüssig
    - einfacher Methodenaufruf ohne Threadwechsel
  
- Inlining
  - ◆ Inlining von kleinen Methoden durch den Compiler
  
- Spezielle Memory-Objekte

## 5 Optimierungen: Memory Objekte

---

### ■ Probleme mit Arrays:

- Unterbereiche nur als Kopie
- keine Zugriffskontrolle möglich

### → Memory-Objekte

#### ◆ Interface **Memory**

◆ Memory-Interface darf von keiner Klasse implementiert werden (Verifiziert)

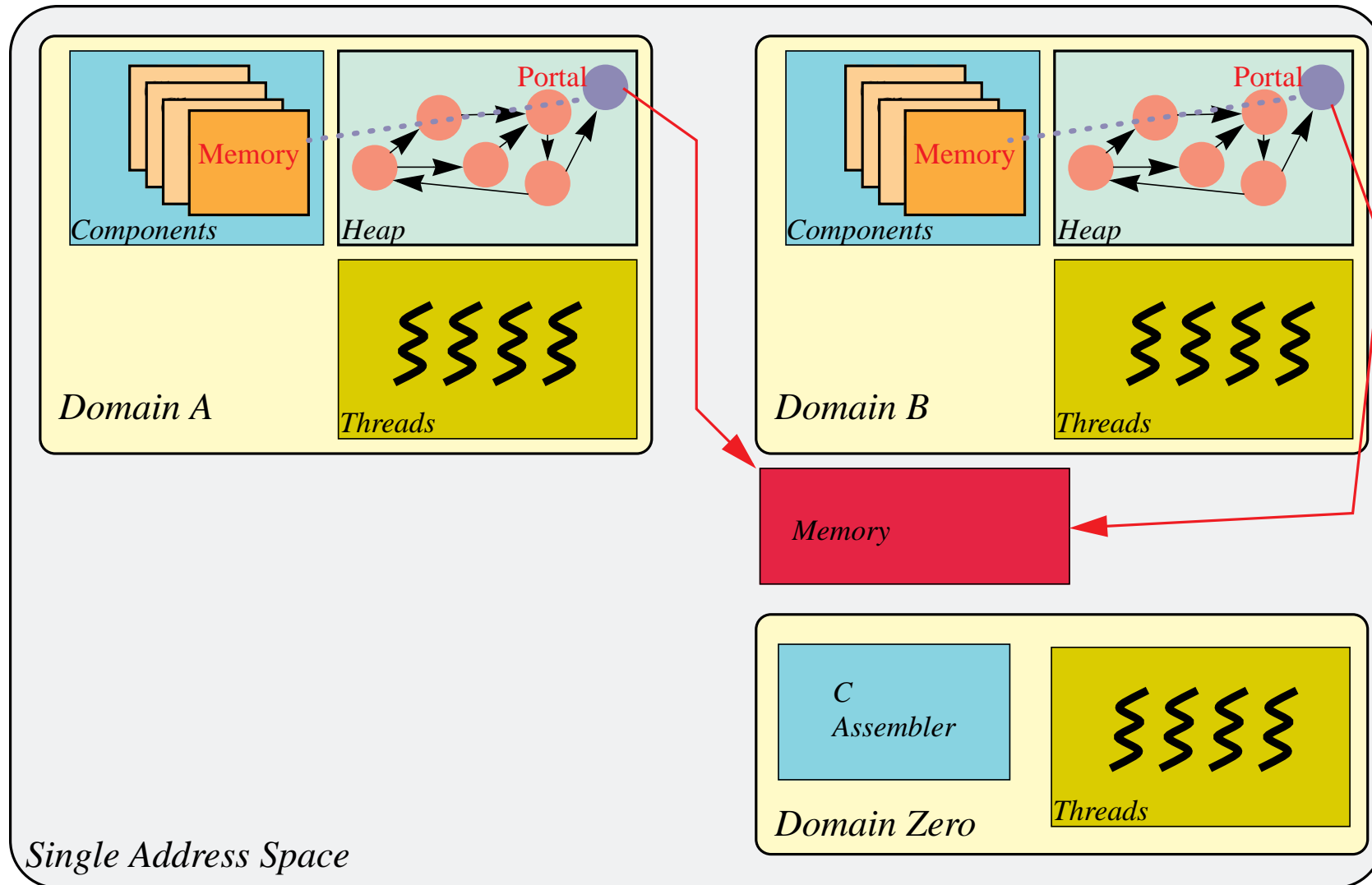
◆ Memory-Aufrufe werden vom Compiler als direkter Speicherzugriff übersetzt

◆ Memory-Objekte sind nicht Teil des Objekt-Heaps

◆ Sharing von Memory-Objekten zwischen Domains möglich

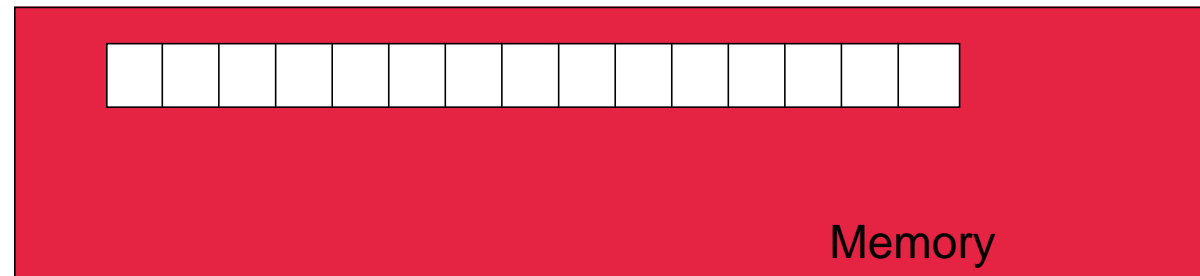
- Übergabe grosser Datenmengen zwischen Betriebssystem-Modulen und Anwendung (Netzwerk, Dateisystem, ...)

# 5 Optimierungen: Memory Objekte (2)



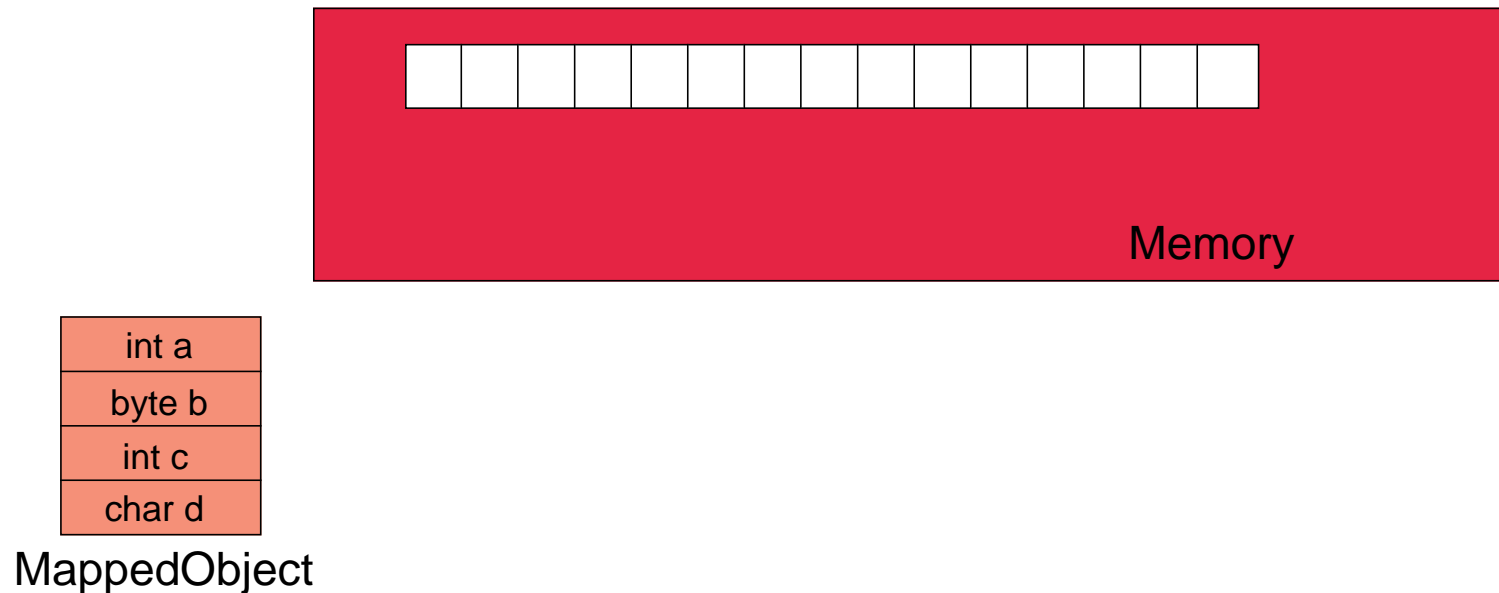
## 5 Optimierungen: Memory Objekte (3)

- Performance-Problem: Bereichsüberprüfung
- (teilweise) Lösung: Abbilden von speziellen Objekten auf Memory-Objekte (Mapping, Reifikation)



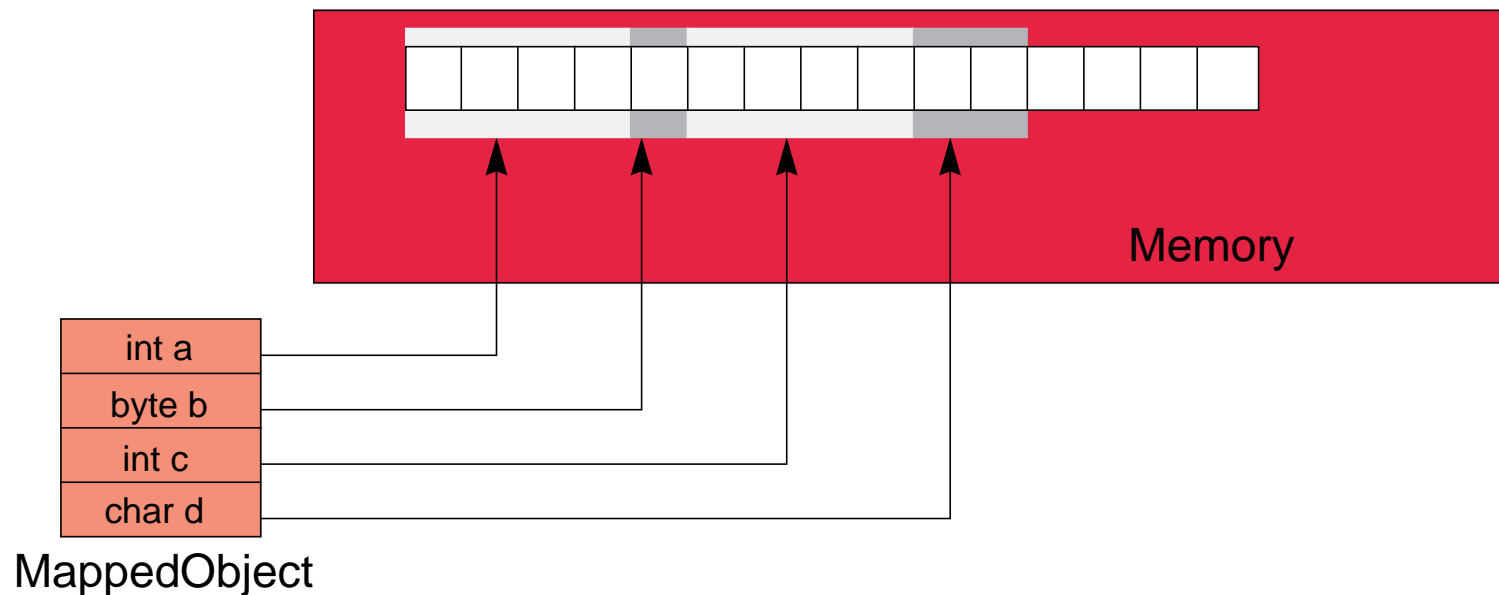
## 5 Optimierungen: Memory Objekte (4)

- Performance-Problem: Bereichsüberprüfung
- (teilweise) Lösung: Abbilden von speziellen Objekten auf Memory-Objekte (Mapping, Reifikation)

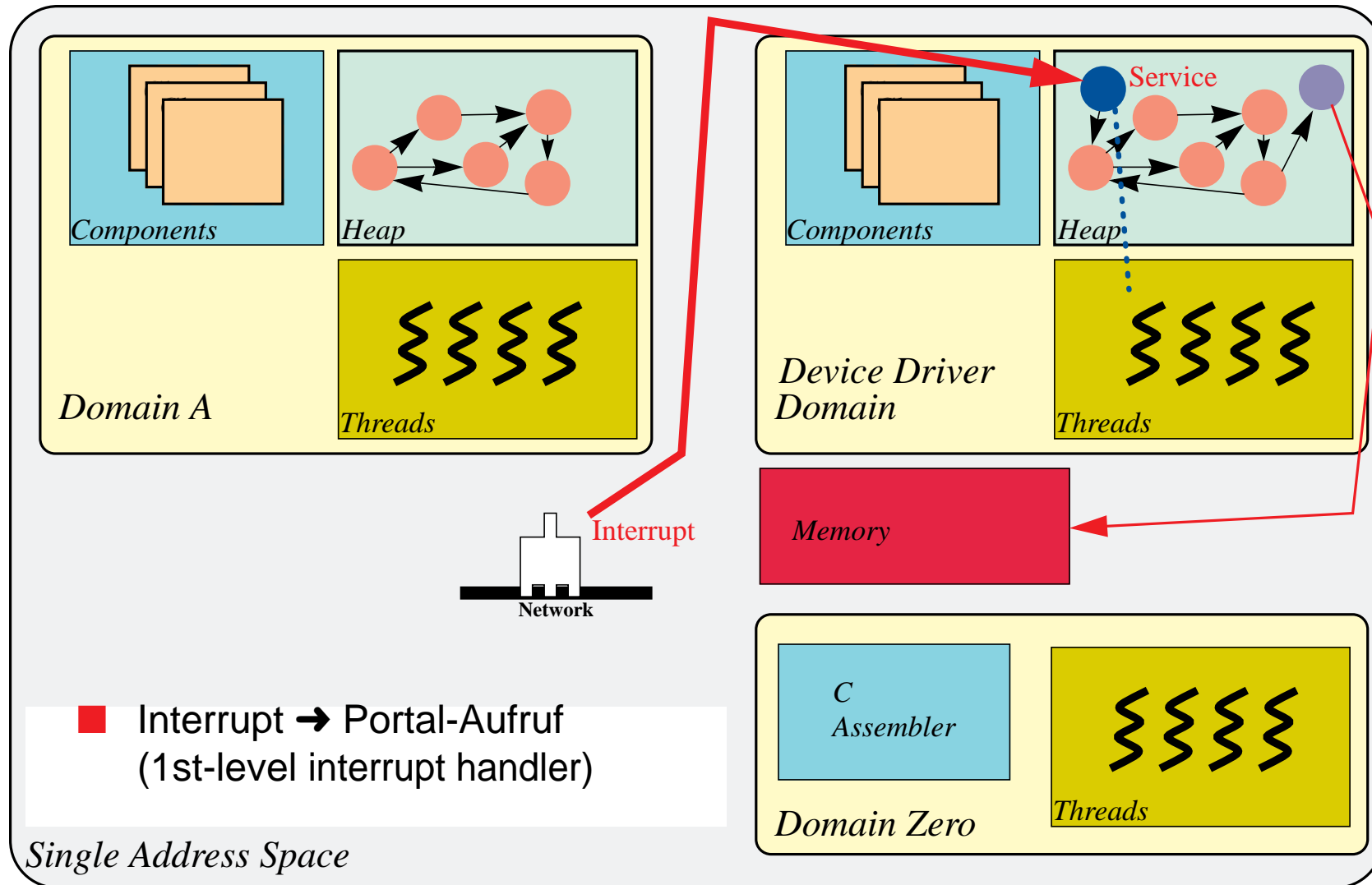


## 5 Optimierungen: Memory Objekte (5)

- Performance-Problem: Bereichsüberprüfung
- (teilweise) Lösung: Abbilden von speziellen Objekten auf Memory-Objekte (Mapping, Reifikation)



# 6 Gerätetreiber

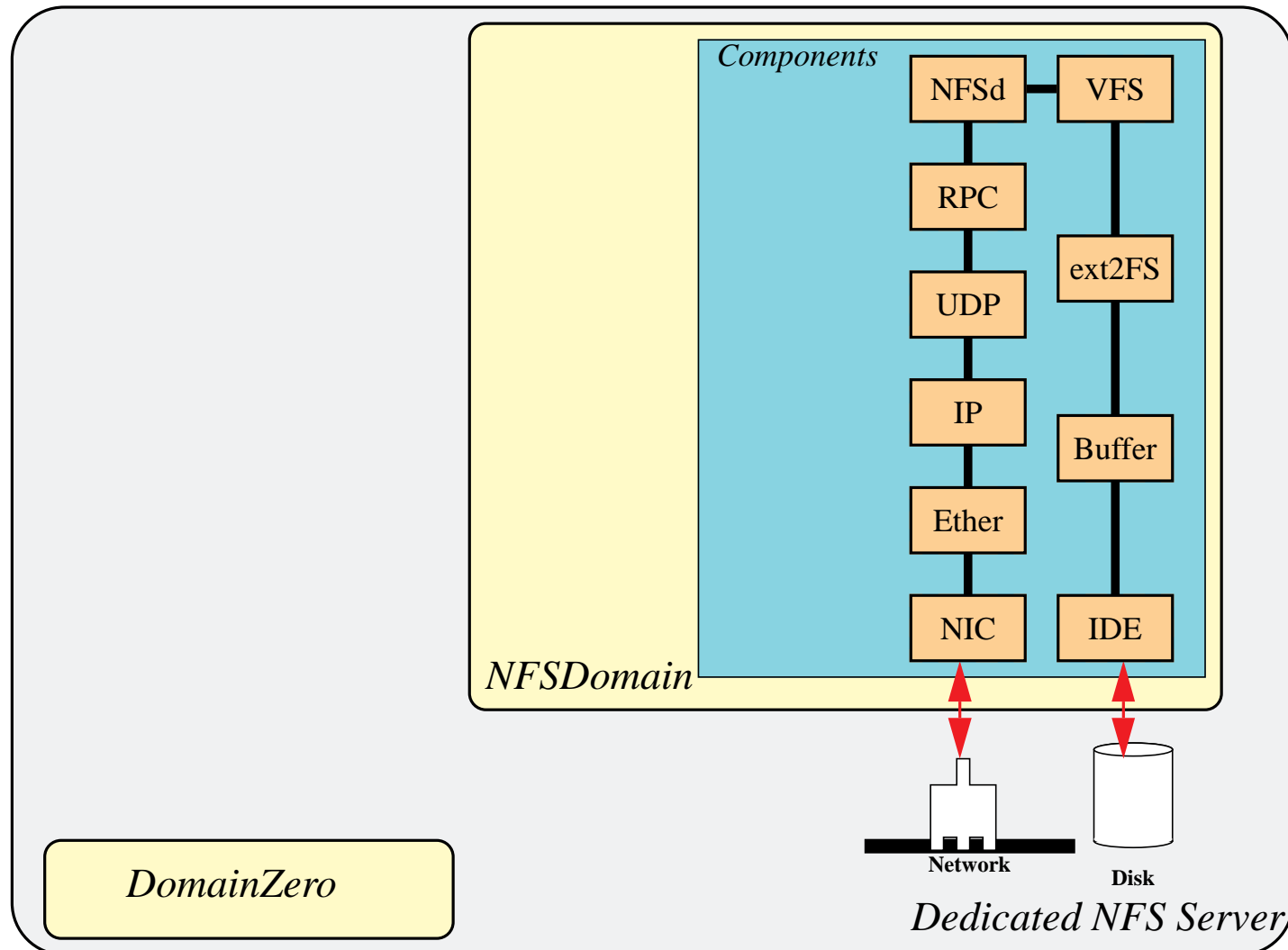


## 6 Gerätetreiber: Schutzproblematik

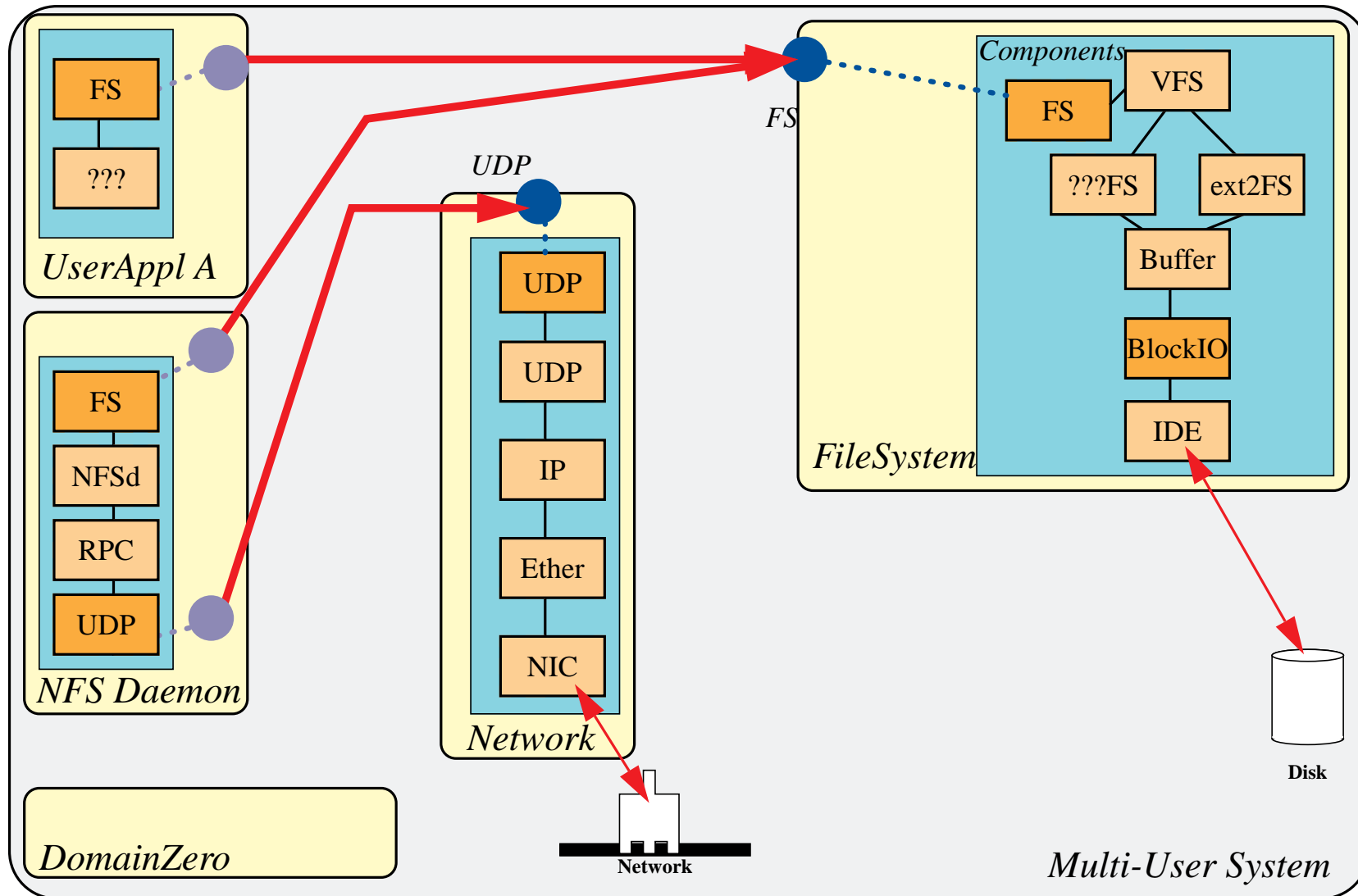
---

- Interrupts sind während der Ausführung des Interrupt-Handlers gesperrt
- Lösung:
  - ◆ Interrupt-Handler-Methoden müssen spezielles Interface implementieren
  - ◆ Verifier überprüft solche Methoden auf maximale Ausführungszeit
    - statische Analyse soweit möglich
    - Einfügen von Laufzeitüberprüfungen wo Ausführungszeit nicht entscheidbar
    - Laufzeitüberprüfungen können Interruptbehandlung abbrechen und Massnahmen gegen weiteres Fehlverhalten treffen (z. B. Interrupts von Gerät deaktivieren)
  - ◆ Aufteilen der Interrupt-Bearbeitung
    - kleiner vertrauenswürdiger 1st-level-Teil bearbeitet Initialisierung und DMA
    - komplexe Bearbeitung in JX-kontrolliertem 2nd-level-Teil

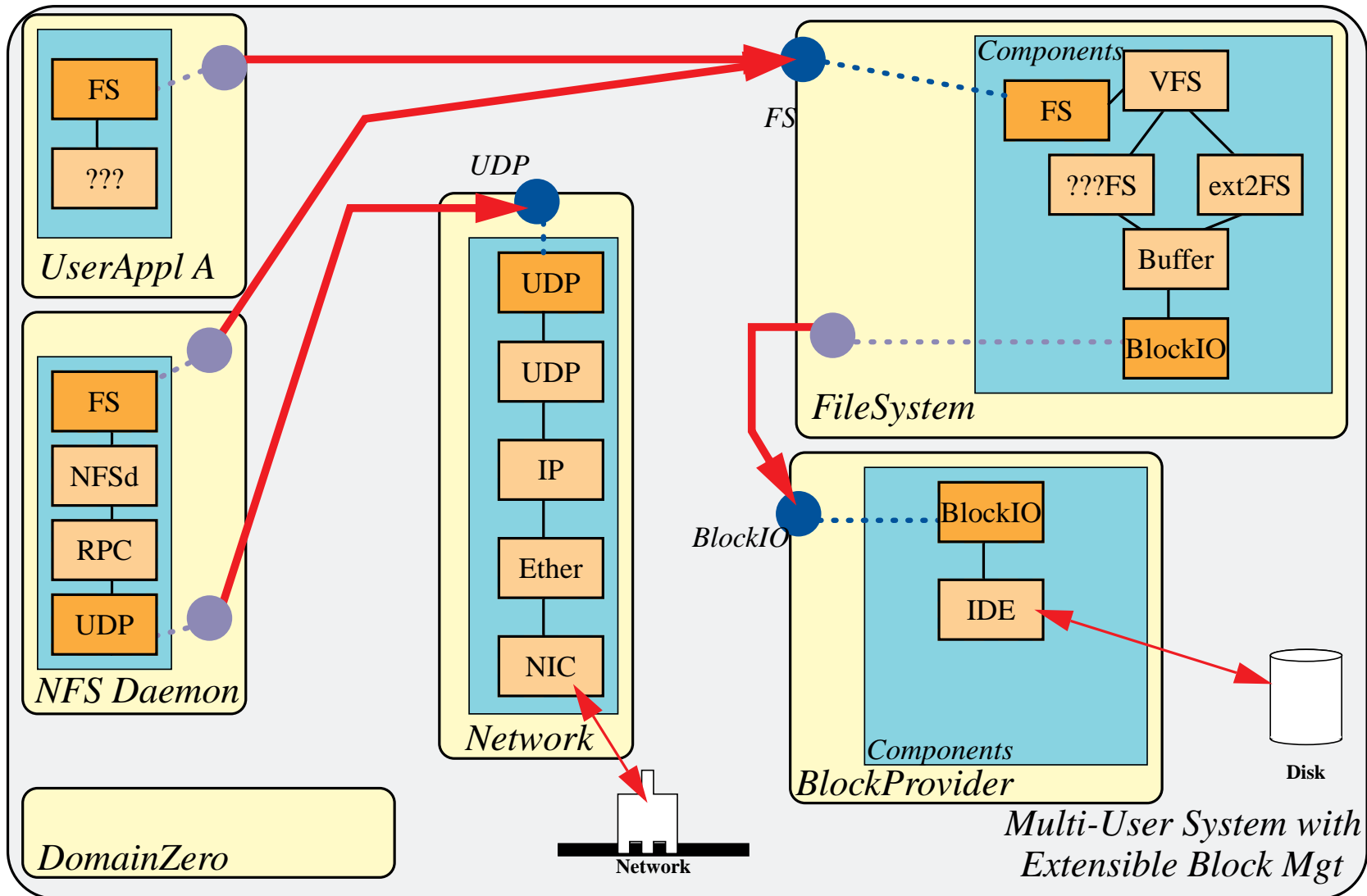
## 7 Betriebssystembaukasten: Spezialsystem



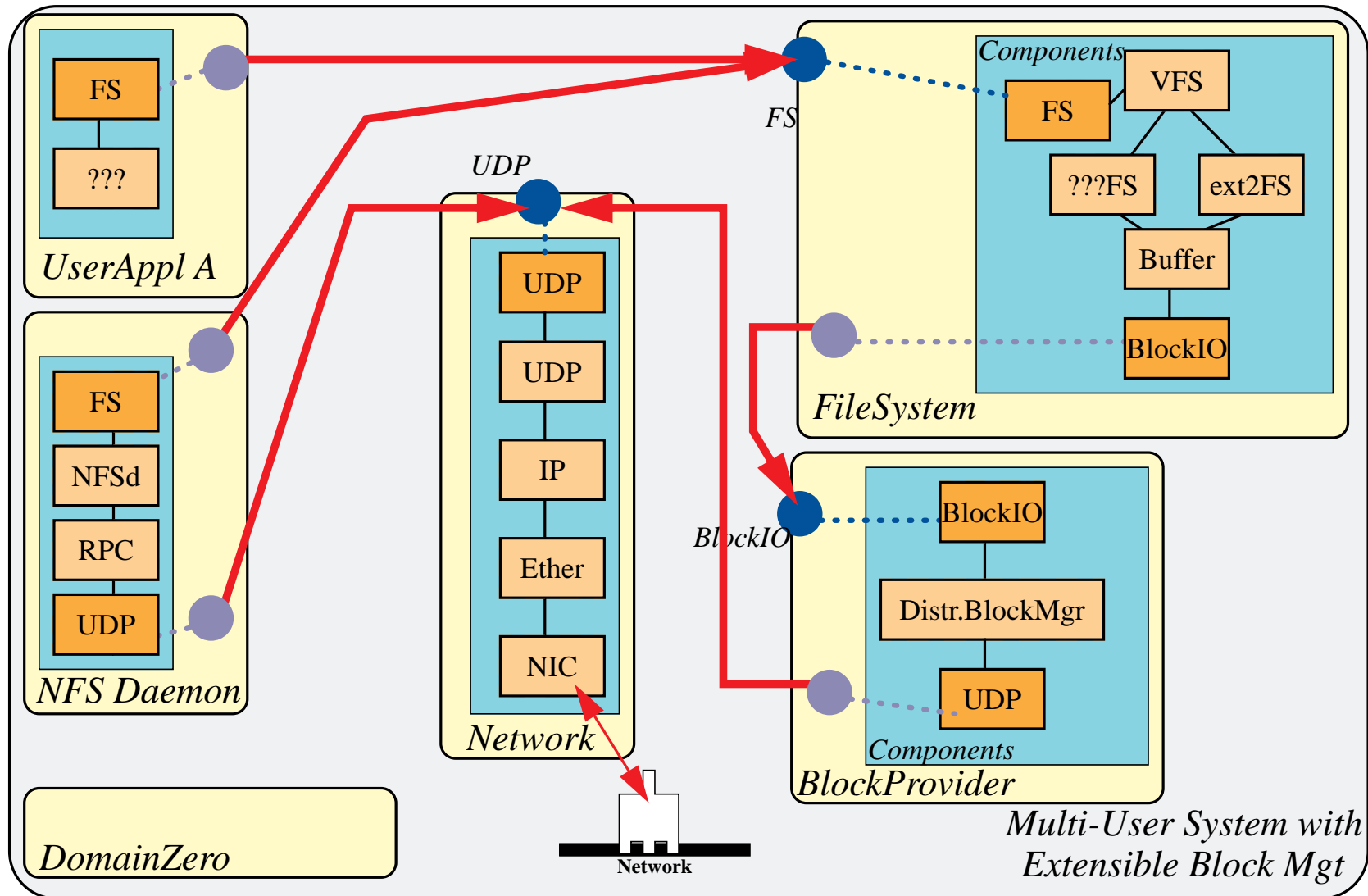
# 7 Betriebssystembaukasten: Multiuser-System



# 7 Betriebssystembaukasten: Erweiterbarkeit



# 7 Betriebssystembaukasten: Erweiterbarkeit (2)



Multi-User System with Extensible Block Mgt

## 8 Erweiterungsmöglichkeiten

---

- Betriebssystemmechanismen pro Domain
  - Scheduler
  - Garbage Collector
  - Compiler (aber: Sicherheitsproblem!)
  
- Erweiterter Portalaufruf
  - über Adressraumgrenzen
  - über Rechengrenzen
    - ↳ verteiltes Betriebssystem
  
- "Smart Portals"
  - ganz oder teilweise replizierte Remote-Objekte (Fragment-Konzept)