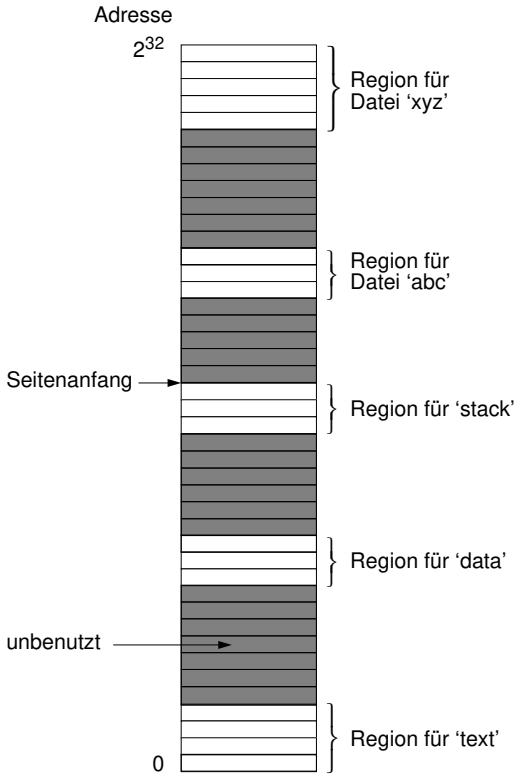
4.3 Arbeitsspeicherverwaltung

Arbeitsspeicherverwaltung in Mach

- ❏ flexible Verwaltung basierend auf Seitenadressierung
- ❏ Trennung in maschinenabhängigen und maschinenunabhängigen Teil auf eine sehr innovative Weise
- ❏ Drei Module mit klar definierten Schnittstellen:
  - Das erste Modul läuft im privilegierten Modus und verwaltet die MMU, ist daher hochgradig maschinenabhängig.
  - Das zweite Modul läuft ebenfalls im privilegierten Modus, ist aber maschinenunabhängig. Es behandelt Seitenfehler, Adreßräume und Seitenersetzung.
  - Das dritte Modul läuft als Benutzerprozeß. Es behandelt hauptsächlich den logischen Teil der Speicherverwaltung insbesondere die Hintergrundspeicher (memory manager, external pager).
  - Kommunikation zwischen Modulen durch Nachrichten an ‘ports’, die an Module gebunden sind
- ❏ Virtueller Speicher
  - Adreßräume sind linear
  - Effiziente Handhabung dünn besetzter Adreßräume durch das Konzept der Regionen (regions)
  - Regionen können zugeteilt und freigegeben werden, Anfangsadresse ist bei Zuteilung angebbar.

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig



Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

Das ‘virtual memory interface’ besteht aus folgenden Methoden (Benutzerfäden an Kern)

Methode	Beschreibung
vm_allocate target_task address size anywhere	Eine Region virtueller Adressen zugänglich machen
vm_deallocate target_task address sizw	Eine Region von Adressen unzugänglich machen
vm_map target_task address size mask anywhere memory_object offset copy cur_protection max_protection inheritance	Ein Speicherobjekt in den virtuellen Adreß-raum abbilden
vm_copy target_task source_address count dest_address	Kopie einer Region an einer anderen Stelle des virtuellen Adreßraums anlegen
vm_inherit target_task address sixe new_inheritance	Vererbungsattribut für eine Region setzen

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

Methode	Beschreibung
vm_read target_task address size data data_count	Einen Bereich aus dem virtuellen Adreß-raum eines anderen Prozesses lesen
vm_write target_task address data data_count	Einen Bereich in den virtuellen Adreßraum eines anderen Prozesses kopieren

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

Nachrichten des Kerns an den ‘external pager’ (memory_object_)

Nachricht	Beschreibung
init memory_object memory_control memory_object_name memory_object_page_size	Initialisieren eines neu abgebildeten Speicherobjektes
data_request memory_object memory_control offset length desired_access	Dem Kern ist die wegen eines Seitenfehlers benötigte Seite zu übergeben
data_write memory_object memory_control offset data data_count	Rückgabe einer modifizierten Seite
data_unlock memory_object memory_control offset length desired_access	Anforderung von Zugriffsattributen durch den Kern
lock_completed memory_object memory_control offset length	Eine frühere Aufforderung zum Rückschreiben modifizierter Seiten ist vollzogen
terminate memory_object seqno memory_control memory_object_name	Anzeige, daß das Objekt zurückgegeben ist und nicht mehr benutzt wird

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

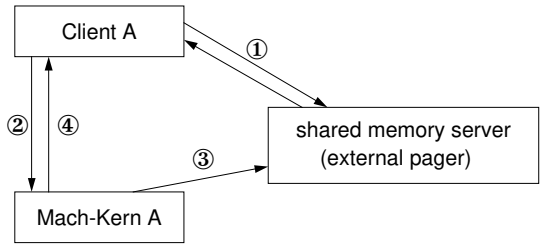
Nachrichten des ‘external pager’ an den Kern (memory_object_)

Nachricht	Beschreibung
set_attributes memory_control object_ready may_cache_object copy_strategy	Übergabe von Attributen an den Kern z. B. zur Steuerung der Ersetzungsstrategie
data_provided memory_control offset data data_count lock_value	Übergabe der angeforderten Seite (Antwort auf data_request)
data_unavailable memory_control offset size	Angeforderte Seite ist nicht verfügbar (Antwort auf data_request)
lock_request memory_control offset size should_return should_flush lock_value reply_to	Aufforderung an den Kern modifizierte Seiten zurückzuschreiben, zu invalidieren oder Zugriffsattribute zu ändern (wird beantwortet mit lock_completed)
destroy memory_control reason	Aufforderung an den Kern, das Objekt zurückzugeben (wird beantwortet mit terminate)

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

Beispiel: Initialisierung

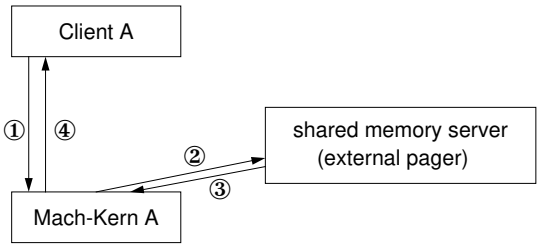


① object_create
② vm_allocate + vm_map
③ memory_object_init
④ *client resumed*

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

Seitenfehler



① *Zugriff mit Seitenfehler*
② memory_object_data_request
③ memory_object_data_provided
④ *client resumed*

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

Verteilter, gemeinsam genutzter Speicher

❑ Anforderungen

- Beispiel: parallele Bearbeitung von Satellitendaten mit unterschiedlichen Programmen
- Differenzierte Behandlung der Regionen bei fork: 'text'-Segmente können gemeinsam genutzt werden
- manche Regionen (z. B. solche die Dateien beherbergen) können für den erzeugten Prozeß irrelevant sein
- bei manchen Regionen kann der Anfangszustand relevant sein, aber es sollen erzeugender und erzeugter Prozeß eigene Kopien besitzen

❑ Antwort in Mach: Vererbungsattribute

- Region wird vom erzeugten Prozeß nicht benötigt
- Region wird von erzeugendem und erzeugtem Prozeß kooperativ benutzt
- erzeugter Prozeß erhält Kopie der Region

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

❑ Organisation

- externer Speicherverwalter, läuft auf Benutzerebene
- drei ports
  - 'object port' beim Speicherverwalter
  - 'control port' im Kern als Adressat für Nachrichten eines Speicherverwalters
  - 'name port' wird als eine Art Name für das Speicherobjekt benutzt (z. B. kann zu einer vorgebbaren virtuellen Adresse erfragt werden, zu welchem Speicherobjekt sie gehört. Die Antwort besteht aus einem Zeiger auf den 'name port'.)

❑ Gemeinsam genutzter Speicher

Sowohl für NORMA- als auch für NUMA-Architekturen von Interesse

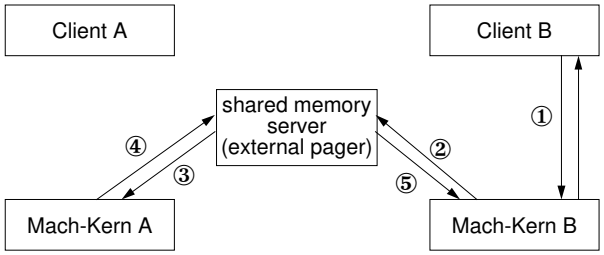
Wichtige Gesichtspunkte

- Struktur und Granularität
- Kohärenz-Semantik
- Skalierbarkeit
- Implementierung
  - Datenlokalisierung und Zugriff
  - Kohärenzprotokoll (write-invalidate, write-update)
  - Ersetzungsstrategie
  - Thrashing
  - Interaktionsmechanismen

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- Gemeinsam genutzter Speicher,von A angesprochene Seite wird von B lesend angesprochen

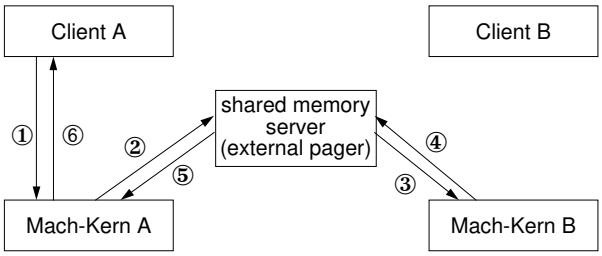


① *Zugriff mit Seitenfehler*
② memory_object_data_request
③ memory_object_lock_request (read only)
④ memory_object_lock_completed
⑤ memory_object_data_provided (read only)

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- Gemeinsam genutzter Speicher, Seite wird von B schreibend angesprochen



① *Zugriff mit Seitenfehler wegen Schreibens*
② memory_object_data_unlock
③ memory_object_lock_request
④ memory_object_lock_completed
⑤ memory_object_data_provided
⑥ *client resumed*

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

Speicherkohärenz

Strikte Konsistenz

Lesen liefert zuletzt geschriebenen Wert

Sequentielle Konsistenz

Die Ausführung erscheint als zeitliche Durchmischung der Operationen, wie sie bei Mehrprozeßbetrieb auftreten kann.

Prozessorkonsistenz

Schreibaufrufe eines Prozessors werden immer in der Reihenfolge ihrer Anforderung sichtbar. Bei Anforderungen von verschiedenen Prozessoren kann keine Aussage über die Reihenfolge gemacht werden, in der ihre Ausführung sichtbar wird.

Schwache Konsistenz

Koordinierungsoperationen sind sequentiell konsistent.

Vor Zugriff auf Koord.-Variablen müssen alle Schreibvorgänge abgeschlossen sein.

Zugriff auf Daten nur, wenn alle vorherigen Zugriffe auf Koord.-Variablen abgeschlossen sind.

Freigabe (Release) Konsistenz

Jede der beiden Koordinierungsoperationen acquire und release ist prozessorkonsistent

Vor Ausführung von release müssen alle vorherigen Schreib- und Lesevorgänge des Prozesses abgeschlossen sein

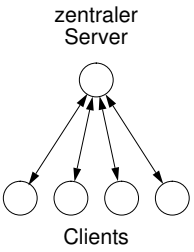
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

• Algorithmen für die Plazierung von Seiten

	nicht vervielfachend	vervielfachend
Nicht migrierend	zentralisiert (central server)	allgemein vervielfachend (full-replication)
migrierend	migrierend (migration)	vervielfachend für Lesezugriff (read-replication)

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

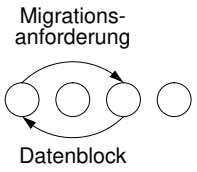
- Zentralisierter Server (Central-server algorithm)



Client	zentraler Server
Datenanforderung senden	
	Anforderung entgegennehmen Datenzugriff durchführen Antwort senden
Antwort entgegennehmen	

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

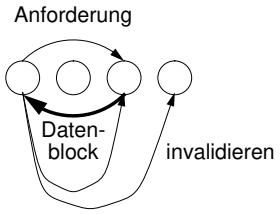
- Migrationsalgorithmus



Client	Verwalter
Falls Block nicht lokal, Speicherort ermitteln und Anforderung senden	
	Anforderung entgegennehmen Block senden
Antwort entgegennehmen Datenzugriff durchführen	

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- Vervielfachung für lesenden Zugriff

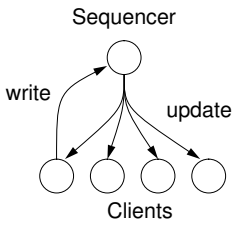


Client	Verwalter
Falls Block nicht lokal, Speicherort ermitteln und Anforderung senden	
	Anforderung entgegennehmen Block senden
Antwort entgegennehmen Invalidierung an alle anderen	
	Invalidierung vollziehen
Zugriff	

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- Allgemein vervielfachend



Client	Sequencer	Verwalter
Falls 'write' Daten an Sequencer		
	Sequenznummer anfügen Multicast	
		Daten empfangen 'update' lokal
Ausführung bestätigt		
'update' lokal		

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

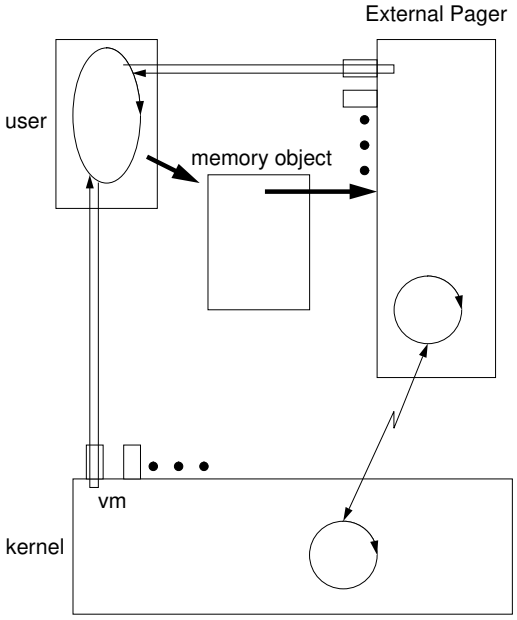
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- Vergleichende Analyse

  Stumm, M.; Zhou, S.: Algorithms Implementing Distributed Shared Memory. Computer, May 1990, pp. 54-64.

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig



© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- *resident page table*

  Tabelle zur Verwaltung von maschinenunabhängiger Information über Seiten

- *address map*

  Doppelt verkettete Liste, deren Einträge eine Abbildung eines Adressabschnitts zu einer Region eines *memory object* definieren

- *memory object*

  Hintergrundspeicher, der von einer Kern- oder Benutzer-*task* verwaltet wird.

- *pmap*

  Maschinenabhängige Speicherabbildung

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig


```
read_fault(page, kernel)
// memory_object_data_request
  switch (page->state) {
    case Read:
      memory_object_data_provided(kernel);
      break;
    case Write:
      page_state = ReadWait;
      memory_object_lock_request
         (page_owner, FLUSH(page), owner_self);
      break;
    default:
      enqueue();
  }
  set_add(page->readers, kernel);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig


```
write_fault(page, kernel)
  switch (page_state) {
    case Read:
      set_remove(page->readers, kernel);
      forall (readers)
        memory_object_lock_request
          (reader, FLUSH(page), owner_self);
      page->readers = empty_set;
      page->owner = kernel;
      if (needs_data)
        memory_object_data_provided(page->owner);
      else
        memory_object_data_unlock(page->owner);
      break;
    case Write:
      memory_object_lock_request
          (page_owner, FLUSH(page), owner_self);
      // weitermachen
    case WriteWait:
    case ReadWait:
      page->state = WriteWait;
      enqueue(kernel, page->writers);
  }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig


```
pageout(page, kernel, data)
  switch (page_state) {
    case Read:
      return;
    case Write:
      save(data);
      page->state = Read;
      page->owner = owner_self;
      break;
    case WriteWait:
      save(data);
      page->owner = dequeue(page->writers);
      memory_object_data_provided(page->owner);
      if (!page->writers)
        if (page->readers)
          page->state = ReadWait;
        else
          page_state = Write;
      if (page->readers || page->writers) {
        deschedule_myself();
        memory_object_lock_request
          (page->owner, FLUSH(page), owner_self);
      }
      break;
    case ReadWait:
      save(data);
      forall (readers)
        memory_object_data_provided(reader);
      page->state = Read;
      page->owner = owner_self;
  }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- Ein Lesezugriff (LOAD) ist von Prozessor P_i ausgeführt bezüglich Prozessor P_k:

  Die Ausführung eines Schreibzugriff durch P_k kann das Ergebnis des Lesezugriff nicht mehr beeinflussen.
- Ein Schreibzugriff (STORE) ist von P_i ausgeführt bezüglich P_k:

  Die Ausführung eines Lesezugriff durch P_k liefert den Wert, der durch Schreibzugriff geschrieben wurde.

- Ein Zugriff ist **ausgeführt**, wenn er bezüglich aller Prozessoren des Systems ausgeführt ist.

- Ein Lesezugriff ist **global ausgeführt**:

  Er ist ausgeführt und der Schreibzugriff, der das Ergebnis bestimmt, ist ausgeführt.

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- Sequentielle Konsistenz:
  - Bevor ein Lesezugriff bezüglich irgendeines anderen Prozessors ausgeführt sein kann, müssen alle früheren Lesezugriffe global ausgeführt sein und alle früheren Schreibzugriffe ausgeführt sein, und
  - bevor ein Schreibzugriff bezüglich irgendeines anderen Prozessors ausgeführt sein kann, müssen alle früheren Lesezugriffe global ausgeführt sein und alle früheren Schreibzugriffe ausgeführt sein.
- Prozessor-Konsistenz:
  - Bevor ein Lesezugriff bezüglich irgendeines anderen Prozessors ausgeführt sein kann, müssen alle früheren Lesezugriffe ausgeführt sein, und
  - bevor ein Schreibzugriff bezüglich irgendeines anderen Prozessors ausgeführt sein kann, müssen alle früheren Lese- und Schreibzugriffe ausgeführt sein.
- Schwache Konsistenz:
  - Bevor ein Lese- oder Schreibzugriff bezüglich irgendeines anderen Prozessors ausgeführt sein kann, müssen alle früheren Synchronisationszugriffe ausgeführt sein,
  - bevor ein Synchronisationszugriff ausgeführt sein kann, müssen alle früheren Lese- und Schreibzugriffe ausgeführt sein, und
  - Synchronisationszugriffe müssen untereinander sequentiell konsistent sein.

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig

- Freigabe-Konsistenz:
  - Bevor ein normaler Lese- oder Schreibzugriff bezüglich irgendeines anderen Prozessors ausgeführt sein kann, müssen alle früheren Belegungszugriffe ausgeführt sein,
  - bevor ein Freigabezugriff bezüglich irgendeines Prozessors ausgeführt sein kann, müssen alle früheren normalen Lese- und Schreibzugriffe ausgeführt sein, und
  - Belegungs- und Freigabezugriffe müssen untereinander sequentiell konsistent sein.

© 1996, Universität Erlangen-Nürnberg, IMMD IV, F. Hofmann

Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors nicht zulässig