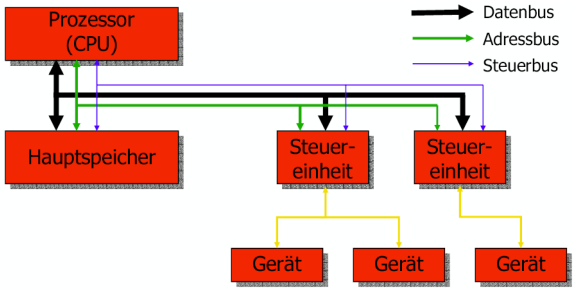
Konzepte und Methoden der Systemsoftware, Prof. Dr. Odej Kao, Zusammenfassung von Florian Schoppmann

Zusammenfassung KMS

1. Technische Grundlagen

1.1 Rechnerarchitektur

- **Von Neumann-Architektur**



- **BUS (Bidirectional Universal Switch)** Übertragungsmedium, das von mehreren angeschlossen Einheiten gemeinsam benutzt wird
- **Operationsprinzip** Verfahren, nach denen Komponenten zusammengefügt werden sowie die Wirkungsweise der Komponenten in ihrer Gesamtheit
  Bei Von-Neumann-Architekturen fehlt die Typenbezeichnung (semantische Lücke)
  Kontrollstruktur: Takt 1 – Befehlszustand, Takt 2 – Datumszustand.

1.2 Ein- und Ausgabearchitekturen

- Ansteuerung
  - Speicherbasierte E/A (Memory-mapped I/O)
  - Dedizierter E/A-Bus zur Ansteuerung
  - E/A-Bussteuereinheiten (PCI, SCSI, ...)
- Unterschied: Programmed I/O (**PIO**) ↔ **DMA**

1.3 Unterbrechungsbehandlung

- **Privilegierter Zustand**: Unterscheidung zwischen zwei Zuständen oder Modi (Bit im Prozessorstatuswort)
- Prioritäten bei der Interruptverarbeitung

1.4 Parallele Architekturen

- **Implizite und explizite** Parallelverarbeitung
- Grundstrukturen
  SISD (Single Instruction, Single Data): sequentielle Architektur
  SIMD (Single Instruction, Multiple Data): Array- und Vektorrechner
  Abwandlung: **SPMD** (Single Program, Multiple Data)
  MISD (Multiple Instruction, Single Data): eventuell Pipeline Rechner
  MIMD (Multiple Instruction, Multiple Data): allgemeine Form, da mehrere Prozessoren zusammen oder unabhängig voneinander an gleichen oder verschiedenen Daten arbeiten können
- Wichtigstes Merkmal: physikalische Speicheranordnung
  Bei gemeinsamem Speicher (z. B. symmetrisches Multi-Processing):
  - Gleichförmiger Speicherzugriff
  - Ungleichförmiger Speicherzugriff
  - Cache-Speicherzugriffe
- Bewertung paralleler Programme:
  Speedup:
  $S_p = \frac{\text{Rechenzeit 1 CPU}}{\text{Rechenzeit } p \text{ CPUs}} \in (0, p]$
  Auslastung:
  $E_p = \frac{\text{Speedup bei } p \text{ CPUs}}{p} = \frac{S_p}{p} \in (0, 1]$

1.5 Rechnernetze

- Netzwerkverbunde: Repeater, Brücken (Koppelelemente zur Verbindung von Segmenten auf der Ebene der physikalischen Adressierung), Router
- Leitungsvermittlung (z. B. öffentliches Telefonnetz), Paketvermittlung (z. B. Internet)

1.6 ISO/OSI Modell

- Ausgeschrieben: ISO Basic Reference Model for Open Systems Interconnection
- Schichten:
  1. Bitübertragungsschicht (ISDN, Modem, LAN)
  2. Sicherungsschicht (CAPI, PPP, SLIP, LLC)
     Medienzuteilung bspw. über Token-Vergabe, Statisches Multiplexen oder CSMA/DA (carrier sense multiple access / collision detection)

3. Vermittlungsschicht (IP)
4. Transportschicht (TCP)
5. Kommunikationsschicht
6. Darstellungsschicht
7. Anwendungsschicht

Da das TCP/IP Modell lange vor dem ISO/OSI 7 Schichten Modell definiert wurde, bildet es das ISO/OSI Modell nicht ganz ab. Im TCP/IP Modell werden die Schichten 5-7 zur Anwendungs-Schicht zusammengefasst. Schicht 4 entspricht TCP und UDP und wird Transport-Schicht genannt. Die 3. Schicht entspricht IP, ICMP und ARP und wird Netzwerk-Schicht genannt. Die Schichten 1 und 2 werden zusammengefasst und heißen Link Layer.

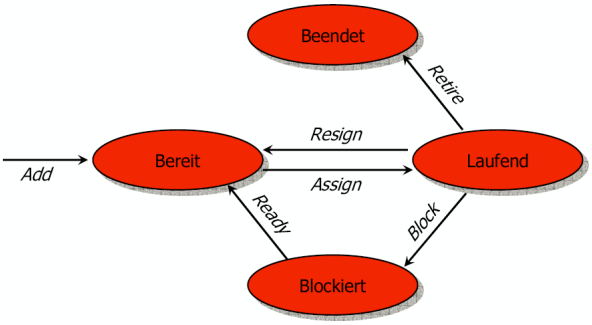
2. Grundbegriffe der Systemsoftware

2.1 Anwender- und Systemsoftware

- **Anwenderprogramme** / **Systemprogramme** (werden oft in einer Einheit zusammengefasst und als Betriebssystem bezeichnet)
- Elementare Aufgaben eines **Betriebssystems**:
  - Unterbrechungsverarbeitung (interrupt handling)
  - Verteilung (dispatching): Umschaltung des Prozessors von einem Auftrag zum anderen
  - Betriebsmittelverwaltung (resource management)
  - Programmallokation (program allocation)
  - Dateiverwaltung (file management)
  - Auftragsteuerung (job control)
  - Zuverlässigkeit (reliability)

2.2 Threads, Prozesse und Prozesszustände

- **Prozess** kann als virtueller Rechner spezialisiert zur Ausführung eines bestimmten Programms angesehen werden
- **Prozesszustände**: Running, Ready, Waiting, Terminated



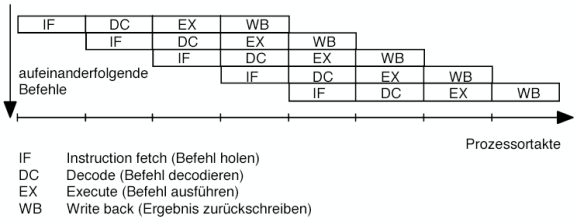
- Implementierung von Prozessen in Betriebssystemen durch Datenstruktur **Prozesskontrollblock**
- **Prozesskontext** Beschreibung des Prozesszustandes im größeren Detail: **Ablaufumgebung**: Befehlszähler, Befehlsregister etc. sowie Adressraum und **Verknüpfungsumgebung**: Datenregistern, Indexregistern, Stapelzeiger etc.

2.3 Thread- und Prozessumschaltung

- Arten der Prozessumschaltung:
  - Explizit
  - Unbedingt: gezielte Übergabe/Auswahl durch übergeordnete Instanz
  - Bedingt: Umschaltung erst bei Erfüllung einer Bedingung möglich
  - Automatisch: Prozessumschaltung durch ein äußeres Ereignis (Interrupt)
- Unterschied **Kernel-Level-Threads** (Heavyweight) / **User-Level-Threads** (Lightweight)

2.4 Grundlegende parallele und nebenläufige Programmiersprachenkonstrukte

- Parallelität ist eine Teilmenge der Nebenläufigkeit
- UNIX-Konzept `fork/join` (auch `fork/wait`) ermöglicht die Erzeugung einer perfekten Kopie (Child) des aufrufenden Prozesses (Parent), Zusammenführung mit `join`
- Pipelining von Prozessoren:



3. Scheduling

3.1 Einführung

- Beispiele: Codeerzeugung für superskalare Prozessoren, Abspielen von Videos im Internet, Montageroboter
- Unterschied: Offline-/Online-Scheduling

3.2 Gestaltungsparameter für Scheduling

- Gestaltungsparameter: Anz. Prozessoren, Prozessmenge statisch/dynamisch, Off- / Online, Abhängigkeiten, Ausführungszeiten, Verdrängung (Preemption), Synchronisation, Prioritäten, Sollzeitpunkte, periodische Prozesse, Ziel (bspw. Länge des Plans, Durchsatz, etc.)

3.3 Schedulingstrategien

FCFS First Come First Served

LCFS Last Come First Served

LCFS-PR Last Come First Served – Pre-emptive Resume

RR Round Robin

PRIO-NP Priorities – Nonpreemptive

PRIO-P Priorities – Preemptive

SJN Shortest Job Next

SRTN Shortest Remaining Time Next

HRN Highest Response Ratio Next, wobei Response Ratio $r = \frac{\text{Wartezeit}+\text{Bedienzeit}}{\text{Bedienzeit}}$, nicht verdrängend

3.4 Multilevel-Scheduling

- **Multilevel-Feedback-Scheduling**: dynamisch, Zeitscheibenlänge wird angepasst, stufenweise Prioritätenreduktion

3.5 Scheduling mit Sollzeitpunkten

- Strikte/Schwache Echtzeitsysteme
- **EDD** (Earliest Due Date): Alle Prozesse können zu jedem Zeitpunkt beginnen
- **EDF** (Earliest Deadline First): Vorgegebene Startzeitpunkte, Verdrängung

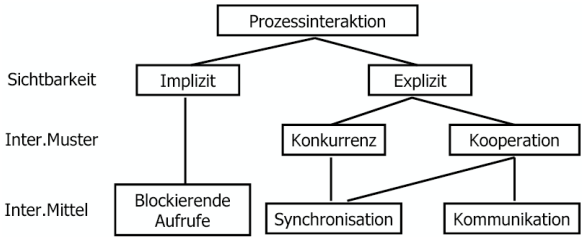
3.6 Scheduling abhängiger Prozesse

- **List-Scheduling**:
  Eingabe: DAG-Abhängigkeitsgraph, Knoten gewichtet mit Prioritäten, zusätzlich markiert mit Ausführungszeiten
  1. Füge alle Quellknoten (Knoten ohne Vorgänger) in Liste ein
  2. Setze $Zyklus = 0$
  3. Solange (nicht alle Knoten bearbeitet):
     4. Solange (Prozessoren frei und aktuelle Liste nicht leer):
        5. Nimm Knoten mit höchster Priorität aus der Liste
        6. Wähle einen leeren Prozessor, um diesen Knoten auszuführen und belege ihn für die Laufzeit des Prozesses
     7. $Zyklus = Zyklus + 1$
     8. Einfügen der Nachfolger in die Liste: Bestimme die Menge der Nachfolger für die aktuelle Runde und sortiere diese entsprechend ihrer Priorität in die Liste ein
- **Pfadlänge in einem Abhängigkeitsgraph**: Summe aller Knotengewichte (inkl. Anfangs- und Endknoten)
- **Level eines Knotens**: Länge des längsten Pfades bis zu einer Senke
- **Co-Level eines Knotens**: Länge des längstens Pfades bis zu einer Quelle
- **HLF** (Highest Level First): Variante des List-Scheduling, der Knoten mit längster Pfadlänge hat höchste Priorität

4. Koordination nebenläufiger Prozesse

4.1 Elementare Koordinationsoperationen

- **Prozesssynchronisation**: zeitliche Abstimmung konkurrierender Prozesse



- Operationen `signal(s)` und `wait(s)`, wobei `s` Sperrflag.

4.1.1 Signalisierung

- **Rendezvous**: Fortsetzung erst, wenn beide Prozesse den **Synchronisationspunkt** erreicht haben.

4.1.2 Kritische Abschnitte

- Die Unteilbarkeit **Atomarität** der Kombination (Überprüfe Bedingung / Setze Bedingung) muss bei Mehrprozessoren durch einen unteilbaren Maschinenbefehl sichergestellt werden

4.1.3 Semaphore

- Betriebssystemunterstützung unumgänglich für Vermeidung von **busy waiting**
- Grundoperationen von Semaphoren: Passieren `P(s)` und Verlassen `V(s)`, wobei `s` Semaphore.
- Für die Echtzeitverarbeitung Einführung von Prioritäten möglich.
  Konzept **Prioritätsvererbung**: der Prozess im kritischen Bereich bekommt während dieser Zeitspanne die Priorität desjenigen Prozesses, welcher auf den Eintritt wartet und die höchste Priorität hat

4.1.4 Monitore

- Die in einem kritischen Abschnitt bearbeiteten Daten werden zusammen mit den Zugriffsalgorithmen in einer sprachlichen Einheit – **Monitor** – zusammengefasst
- Monitorbenutzung durch Aufruf der Form `Monitorname.Funktionsname (aktuelle Parameter);`
- Bedingungsvariablen mit Operationen `wait` und `signal`

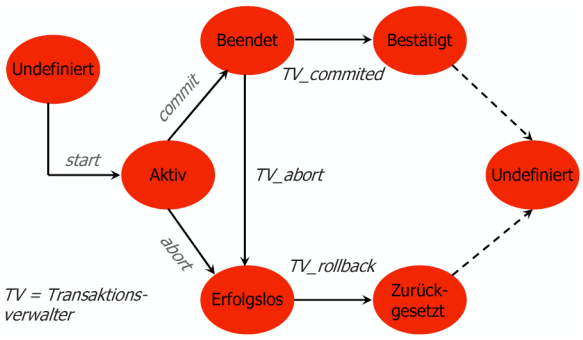
4.2 Transaktionen

4.2.1 ACID-Eigenschaften

- Eine **Transaktion** ist eine Operationsfolge, die in ihrer Wirkung atomar ist
- **ACID**: Atomicity, Consistency, Isolation (Zwischenergebnisse einer Transaktion sind nicht nach außen sichtbar), Durability (Beendete und bestätigte Transaktionen können nicht rückgängig gemacht werden)
- Weitere Anforderungen: Concurrency, Recovery, Versioning, Security

4.2.2 Architektur von Transaktionssystemen

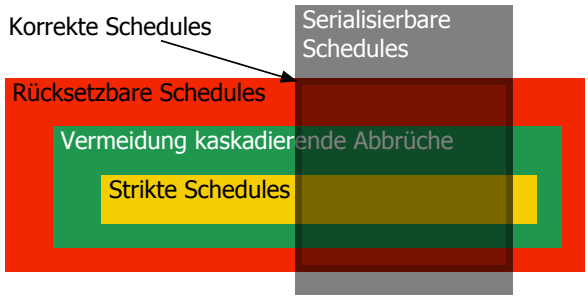
- Transaktionszustände:



- **Schedule**: Vorschrift für die nebenläufige und verzahnte Ausführung mehrerer Transaktionen
- Zwei Transaktion T_i und T_j stehen in **Konflikt**, wenn mind. 2 Teiloperationen $p(x) \in T_i$ und $q(y) \in T_j$ in Konflikt zueinander stehen $(p \leftrightarrow q)$, d. h. wenn gilt:
  $x = y$ und $i \neq j$ und (p oder w Schreiboperation)
- Ein **Schedule** S für eine Menge von Transaktionen $\{T_1, \ldots, T_n\}$ ist eine partiell geordnete Menge aller Transaktionsoperationen mit:
  1. $\forall p, q \in T_i : p <_{T_i} q \Rightarrow p <_S q$
  2. $\forall p, q \in S : p \leftrightarrow q \Rightarrow$ entweder $p <_S q$ oder $q <_S p$
- Unterschied: Vollständige Schedules vs. Schedules (die nur Anfangsstücke vollständiger Schedules sind).
- **Bestätigte Projektion** $C(S) \subseteq S$ eines Schedules S

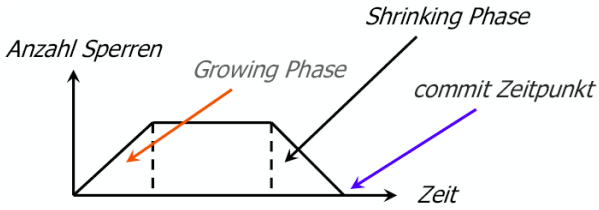
4.2.3 Serialisierbarkeit

- Zwei Schedules S und S' heißen **äquivalent**, wenn sie dieselben Ausgabewerte liefern und denselben Datenzustand zurücklassen
- Ein Schedule S heißt **serialisierbar**, wenn seine bestätigte Projektion $C(S)$ zu einem seriellen Plan äquivalent ist
- S und S' heißen **konfliktäquivalent**, wenn sie aus denselben Operationen bestehen und Konflikte in gleicher Weise auflösen, d.h.:
  $\forall p \leftrightarrow q : p <_S q \Leftrightarrow p <_{S'} q$
  Konfliktäquivalente Schedules liefern sie auch das gleiche Ergebnis.
- Ein Schedule S heißt **konfliktserialisierbar**, wenn seine bestätigte Projektion $C(S)$ zu einem seriellen Plan konfliktäquivalent ist.
  Es gilt für einen Schedule S: S konfliktserialisierbar $\Rightarrow$ S serialisierbar
- Ein Schedule S heißt **rücksetzbar** (recoverable), wenn jede Transaktion T_i erst dann bestätigt wird, wenn alle Transaktionen T_j, von denen sie gelesen hat, bereits bestätigt sind oder abgebrochen wurden.
- **Kaskadierender Abbruch** Transaktionen, die ungültig gewordene Daten gelesen haben, müssen abgebrochen werden.
- Ein Schedule S vermeidet einen kaskadierenden Abbruch, wenn keine der Transaktion aus S unbestätigte Daten liest.
- Ein Schedule S heißt **strikt**, wenn von keiner der darin enthaltenen Transaktion unbestätigte Daten gelesen oder überschrieben werden.



4.2.4 Koordination und Sperrprotokolle

- Zulässige Aktivitäten des Transaktionsplaners: Sofortige Ausführung, Verzögerung, Abweisen
- **Zweiphasensperren**: Zweiphasensperren (exklusive locks, shared locks)
- Ein **sperrender Scheduler** erweitert jede Transaktion um die Befehle `rl` / `wl` / `ru` / `wu`
- 2-Phasen-Sperrprotokoll:



- Varianten von 2PL: Freigabe erst am Ende (strikt) oder setzen aller Sperren zu Anfang (konvervativ)

4.2.5 Wiederherstellung

- Sicherungstechniken: Logging (ggf. mit **Checkpoints**), **Schattenspeicher**
- Pufferverwaltung: Operationen: `flush(c)`, `fetch(x)`, `pin(c)` (Verhinderung des Auslagerns des Inhalts von `c`), `unpin(c)`
- Realisierung Schattenspeicher: Jede Transaktion unterhält zwei Verzeichnisse, die Verweise auf die Datenwerte enthalten

5. Betriebsmittelverwaltung und Verklemmungen

5.1 Betriebsmittel: Definition und Klassifikation

- Grundsätzliche Vorgehensweise: Zentrale Instanz, Verständigung der Teilnehmer (Protokolle) oder unkoordinierte Nutzung (bspw. CSMA/CD)
- **Betriebsmittel**: Unterscheidung Betriebsmittelform (Einzel-, Mehrfachexemplar), Persistenz (Wiederverwendbarkeit), Kapazität real, virtuell, logisch (bspw. Datei)

5.2 Ziele einer Betriebsmittelverwaltung

- Anforderungen: Korrekte Abläufe, keine Verklemmung, kein Verhungern, hohe Nebenläufigkeit

5.3 Auswahlstrategien

- FCFS/FIFO (First-In-First-Out): erste Anforderung

- First-Fit-Request: erste erfüllbare Anforderung
- Best-Fit-Request: Anforderung, die die Restkapazität minimiert
- Vermeidung des Verhungerns bei First/Best-Fit: Betrachte immer nur „Fenster“ (d. h. immer nur die ersten L Anforderungen) in der Warteliste. Nach jeder erfolgreichen Belegung setze L auf: $L-1$, wenn $L > 1$ und erste Anforderung noch nicht berücksicht, sonst auf L_{max}.

5.4 Verklemmungen

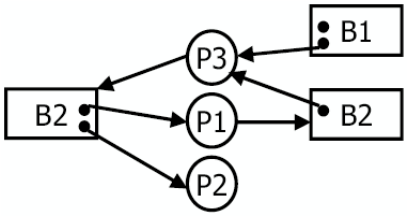
- Notwendige und hinreichende Bedingungen für Deadlocks:
  1. mutual exclusion
  2. hold-and-wait
  3. no pre-emption
  4. circular wait

5.5 Behandlung von Verklemmungen

- Vorbeugung: eigener Verwaltungsprozess (daemon), Totalfreigabe, Anforderung von BM nur in einer festgelegten Reihenfolge
- Vermeidung: Anforderung nur dann gewähren, wenn sie in eine sichere Situation führt
- Entdeckung und Auflösung: In regelmäßigen Abständen/bei jeder Belegung wird die aktuelle Belegungssituation analysiert

5.6 Betriebsmittelgraphen zur Modellierung von Belegungssituationen

- **Betriebsmittelgraph**: Bi-partiter Graph, Maximaler Ausgangsgrad eines Knotens = Insgesamt verfügbare Einheiten eines BMs, hier durch schwarze Punkte symbolisiert:



- BM-Graph heißt **reduzierbar**, wenn ein Prozess existiert, dessen Anforderungen sofort erfüllbar sind und somit alle seine Kanten entfernt werden können. Er heißt **vollständig reduzierbar**, wenn es eine Folge von Reduktionen gibt, so dass am Ende alle Kanten entfernt sind.
- **Banker-Algorithmus**: Restforderungen $R \in \mathbb{N}^{m \times n}$, freie BM $f \in \mathbb{N}^{1 \times n}$, $m \mathrel{\widehat{=}}$ Anz. Prozesse, $n \mathrel{\widehat{=}}$ Anz. BM
  1. Wähle eine Zeile aus der Matrix R so aus, dass die notierten Restforderungen kleiner oder gleich der freien Ressourcen in f sind. Falls eine solche Zeile nicht vorliegt, wird das System u. U. in ein Deadlock enden, da kein Prozess beendet werden kann.
  2. Nehme an, der zugehörige Prozess ist terminiert und gebe seine belegten Ressourcen frei, d. h. aktualisiere den Vektor f.
  3. Wiederhole die Schritte 1 und 2, bis alle Prozesse terminiert sind (sicherer Zustand) oder eine Verklemmung auftritt.

5.7 Erkennung und Beseitigung von Verklemmungssituationen

- Restanforderungen nicht bekannt $\Rightarrow$ Vermeidung von Verklemmungen nicht möglich
- Daher betrachte nur die aktuellen Forderungen und definiere: Eine Menge von Prozessen heißt **verklemmungsfrei** genau dann, wenn eine Beendigungsreihenfolge derart existiert, dass jede Anforderung durch die von terminierten Prozessen freigegebenen BM erfüllt werden kann.
- Banker Algorithmus wird zur Verklemmungs entdeckungeingesetzt: Ersetze Restanforderungen durch aktuelle Anforderungen
- Beseitigung von Verklemmungen: Prozesse abbrechen, Prozesse zurücksetzen (letzter Checkpoint), BM entziehen.

6. Speicherverwaltung

6.1 Einführende Definitionen

- Wachstum von Heap und Stapel:

6.2 Speicherbelegung

- **Interner Verschnitt**: Nicht benutzter Speicher innerhalb zugeteilter Segmente, **Exter-**

ner Verschnitt Zu kleine Segmente können nicht mehr belegt werden. Insg. wäre aber genug Speicher vorhanden.

- Auslastung:
  $\eta = \frac{\text{Größe des belegten Speichers}}{\text{Größe des gesamten Speichers}}$

6.3 Auswahlstrategien

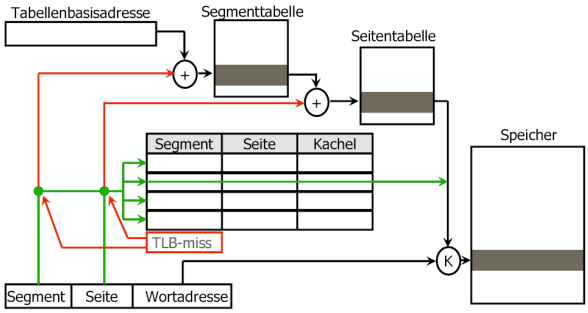
- **First-Fit**
- **Next-Fit**, **Rotating First-Fit**
- **Nearest-Fit** Von einer gwünschten Adresse aus First-Fit-Suche
- **Best-Fit**: Das kleinste aber hinreichend große Stück
- **Worst-Fit**
- **Quick-Fit**: Für häufig vorkommende Belegungen wird eine Extraliste unterhalten
- **Buddy-Systeme**: Für mehrere Belegungsgrößen – Speicher in den Größen von 2er Potenzer – Zweierpotenzen – wird eine eigene Liste angelegt. Erwartungswert für internen Verschnitt $f_{int} = 25\ \%$.
- **Vektorverfahren**: Stückgröße $= k \cdot$ Grundeinheit, Speicherung für jedes Stück (in einer Bitmap), ob frei oder belegt.
- **Randkennzeichnungsverfahren**: Integration von Belegungsdarstellung und Stückverwaltung, doppelte Verkettung nach Größe

6.4 Adressierungsarten

- **Virtueller Speicher**: Abhhilfe zur vermeidung unverhältnismäßig großer Seitentabellen (insb. bei 64 bit-Prozessoren) nötig
- **Invertierte Seitentabellen**
- **mehrstufige Seitentabellen**: Beispiel bei 32 bit-Adresse:

10	10	12
P1	P2	Offset

  PT1 bestimmt die Second-Level Seitentabelle, PT2 die Startadresse der Seite und Offset die Wortadresse innerhalb der Seite.
- Segmentadressierung mit Seitenadressierung und **Translation Lookaside Buffer**:



- Probleme des TLB: Wie L1/L2-Cache: Adressraumwechsel invalidiert die Einträge in TLB, Abhilfe bspw. Sichern des aktuellen TLB im Hauptspeicher und Restauration beim Adressraumwechsel

6.5 Speicherallokation

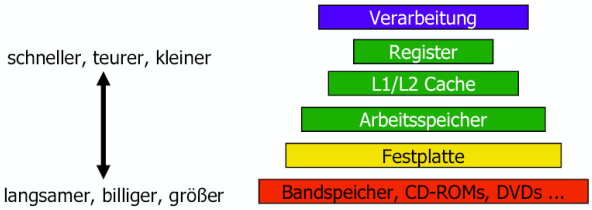
- **Namensfunktion** N: Transformation der symbolischen Namen in eindeutige logische Adressen des Arbeitsspeichers
  Umsetzung durch Compiler und Linker (kombiniert einzelne Objektmodule miteinander zu einem Lademodul mit eindeutigen logischen Adressen)
- **Speicherfunktion** S: logische Adresse → physikalische Adresse
  Durchführung: Allokation des Arbeitsspeichers und Transfer des Programms vom sekundären Speicher in den RAM
- **Inhaltsfunktion** I: Speicheradresse → enthaltener Wert
- **Speicherallokation**: statisch und dynamisch denkbar

6.6 Verdrängungs- und Nachschubstrategien

- Nachschubstrategien: Auf Verlangen (**demand paging**), vorgeplant (**pre-paging**), kombinierte Ansätze
- Verdrängungsstrategien: Lokale Auswahlstrategie, globale Auswahlstrategie: Als am sinnvollsten erweist sich die LRU (**Least Recently Used**)-Strategie.
- **Second-Chance-Algorithmus**: FIFO-Modifikation

6.7 Speicherhierarchien und Lokalität

- Speicherhierarchie:

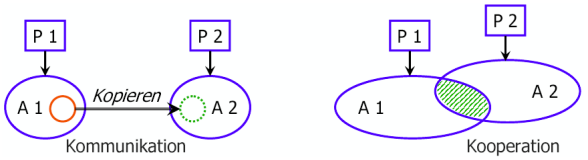


- **Lokalitätsprinzip**: Räumlich/Zeitlich
- Unterschied **Caching** ↔ **Virtualisierung**: benötigte Daten in einer höheren Schicht → Beschleunigter Zugriff durch Caching, benötigten Daten auf einer niedrigeren Schicht → Vergrößerung der Speicherkapazität der Schicht durch Virtualisierung
- **Cachekohärenzproblem** durch nach oben verzweigte Speicherhierarchien

7. Kommunikation zwischen Prozessen

7.1 Einführung

- Unterscheidung:



- **Nachrichtentransaktion**: Abschicken, Zustellung, Verarbeitung, Quittung

7.2 Elementare Kommunikationsmuster

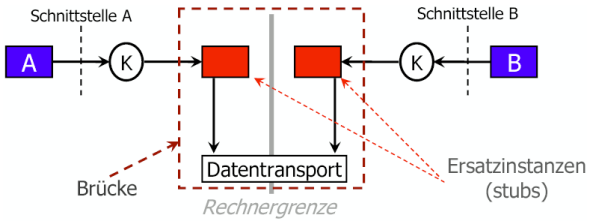
- Synchrone/Asynchrone Kommunikation
- **Kanal**: Abstraktion für die Kommunikation, d. h. Datenobjekt mit Operationen `send` und `receive`
- **Nachrichtenübergabe**: Wertübergabe, Referenzübergabe oder Behälterübergabe
- **Meldung**: Außer Quitting keine Rückmeldung an Empfänger, **Auftrag**: Empfänger startet Verarbeitung und gibt Ergebnis zurück an den Sender
- **1:n-Kanal**: mehrere Nachrichten werden gepuffert
- **n:1-Kanal**: z.B. ein Server mit mehreren replizierten Prozessen
- **Ports** sind Kanäle, WWW Port 80 wäre bspw. ein Eingangstor und n:1-Kanal
- **multicast**: Nachricht an viele, **broadcast**: Nachricht an alle

7.3 Dienstleistungsbeziehung (Client-Server-Prinzip)

- Der Client teilt dem Server bei der Auftragserteilung mit, an welchen **Rückkanal** das Ergebnis zu senden ist.
- **Sekretär**: Server mit mehreren Operationen, Art der Operation wird als Kennung in der Parameterliste übertragen
- **Team**: Server als Team von Prozessen, jeder Prozess ist für eine spezielle Operation zuständig und besitzt eigenen Eingangskanal
- Mechanismen zur Erhöhung des Durchsatzes:
  - Client und serverseitige **Pufferung**
  - Reproduktion (**Cloning**): Existenz mehrerer identischer Kopien eines Serverprozesses
  - Fließband (**Pipelining**): Zeitliche Überlagerung von Auftragsteilen beim Ausführen von mehreren Aufträgen
  - **Multiplexing**: Flexible Reaktion auf blockierte Prozesse, Verzahnen, event-programming

7.4 Rechnerübergreifende Kommunikation

- **Brückenprinzip**: Schaffung einer Illusion, dass der Kommunikationspartner in der lokalen Umgebung existiert:



- Aufgaben des **Transportsystems**:
  - Datentransport
  - Garantie: Sicherstellen des korrekten Datentransports
  - Routing: Versenden der Nachricht auf geeigneten kürzesten Wegen
  - Transportart: Verbindungsorientiert oder verbindungslos, Portionierung von Nachrichten
  - Einordnung in ISO/OSI

7.5 Fernprozeduraufruf (Remote Procedure Call, RPC)

- **RPC**: sprachorientierte Variante der synchronen, auftragsorientierten Kommunikation
- **Dynamische Bindung** der auszuführenden Prozedur P an den aufrufenden Prozess zur Laufzeit
- Call-by-reference nur schwer möglich, statt dessen call-by-copy / restore (= call-by-value-and-result)
- Implementierung: RPC-Dienstes für dynamische Zuweisung einer freien Portnummer zu einem lokalen Dienst (**Portmapper**)
- Zur Abhilfe für Fehlersituationen: **At-most-once-Semantik**: Auftrag wird höchstens einmal ausgeführt, **At-least-once-Semantik**, **Maybe-Semantik**: keine Aussage möglich
- Üblich sind Time-outs zur Erkennung von Problemen

7.6 Mobiler Code

- Beispiele: PostScript, Java-Applets, etc.
- Probleme: In heterogener Umgebung nur Interpretation möglich, Sicherheit, Kontrolle des Ressourcenverbrauchs