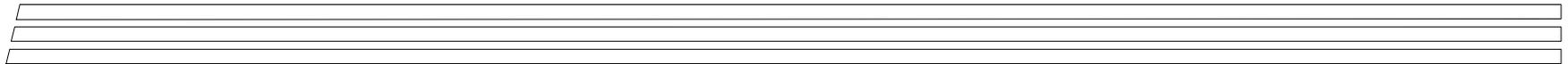


Parallele Programmierung

Prof. Dr. Rita Loogen

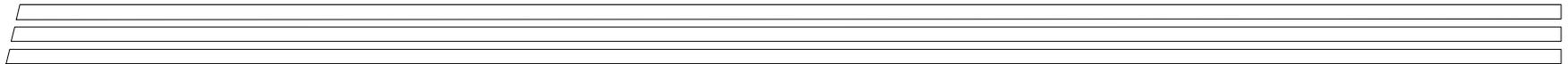
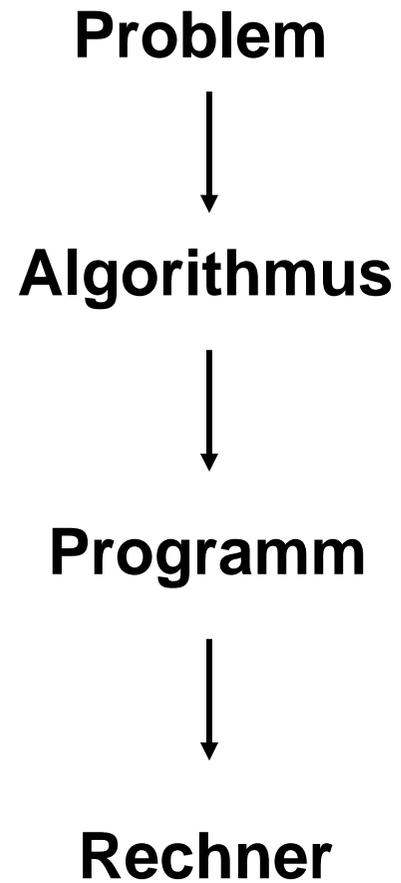
Fachbereich Mathematik und Informatik

WS 2008/09





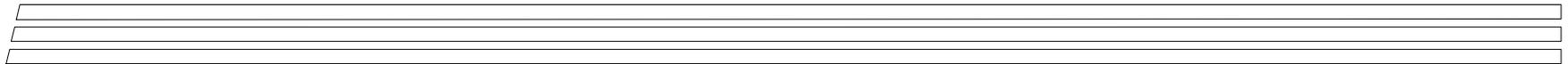
Vom Problem zum Programm





Sequentialität

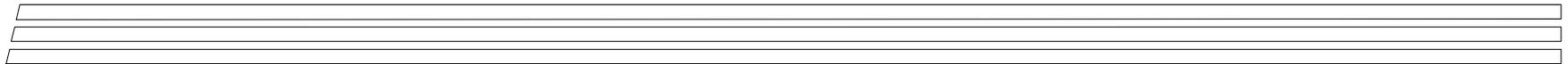
- Sequenz:
 - natürliches Konzept zur Beschreibung von Abläufen jeglicher Art
 - Basis unseres Zeitbegriffs
 - => seq. Algorithmen, Ursprung in Mathematik, Berechnungen als Sequenz von Operationen
- sequentielle Rechner nach von-Neumann-Prinzip
 - => seq. Programmierung:
 - Beschreibung von Berechnungsabläufen

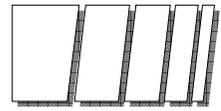




Parallelität

- "Die Welt ist hochgradig parallel."
- Natur:
simultane Weiterentwicklung, Pflanzenwachstum, Wetter
- dezentrale Organisationen
- ...
- Viele Probleme sind ihrem Ursprung nach parallel,
Sequentialisierung stellt eine künstliche Einschränkung dar.
- **These:** **Parallelverarbeitung** ist die natürliche
Form der Informationsverarbeitung.



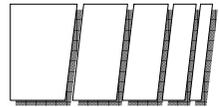


Erwartungen an die Parallelverarbeitung

- besser verständliche Programme
- höherer Informationsgehalt über die Problemstruktur
- mehr Rechenleistung >> “grand challenges”
- besseres Preis/Leistungsverhältnis:

Die Parallelverarbeitung auf vielen kleinen, billigen Rechnern ist kostengünstiger als der Einsatz eines größeren, viel teureren Supercomputers.

- bessere Verfügbarkeit durch Komponentenredundanz
- ...



Parallelität

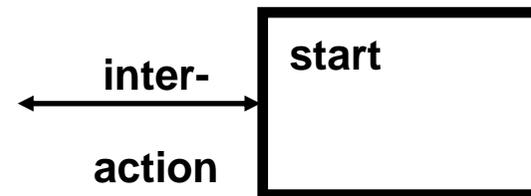
vs. Nebenläufigkeit

- Kooperation von abhängigen, kommunizierenden Prozessen zur Lösung eines Problems
- transformationelle Systeme, parallele Algorithmen

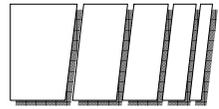


- ⇒ Leistungssteigerung durch Zeitersparnis
- ⇒ Ziele: Beschleunigung, Skalierbarkeit (größere Probleme)

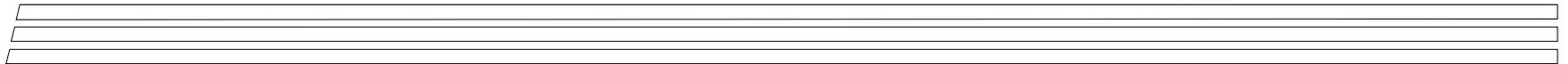
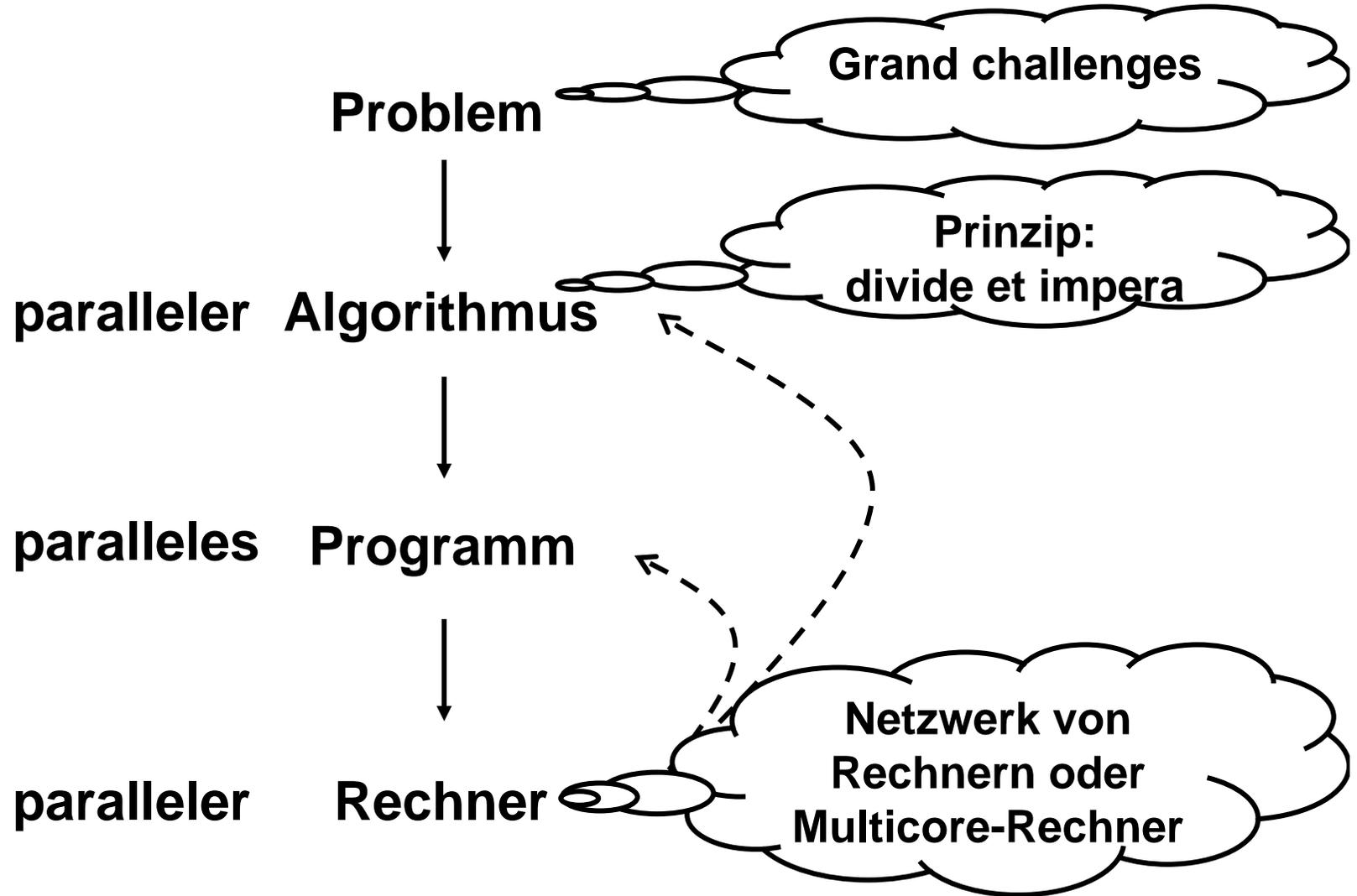
- Kooperation von unabhängigen Prozessen, die verschiedene Aufgaben haben
- reaktive bzw. verteilte Systeme, z.B. GUIs, Betriebssysteme



- ⇒ Verwaltung komplexer Systeme durch Abstraktion und Strukturierung
- ⇒ Ziele: Sicherheit, Fehlertoleranz



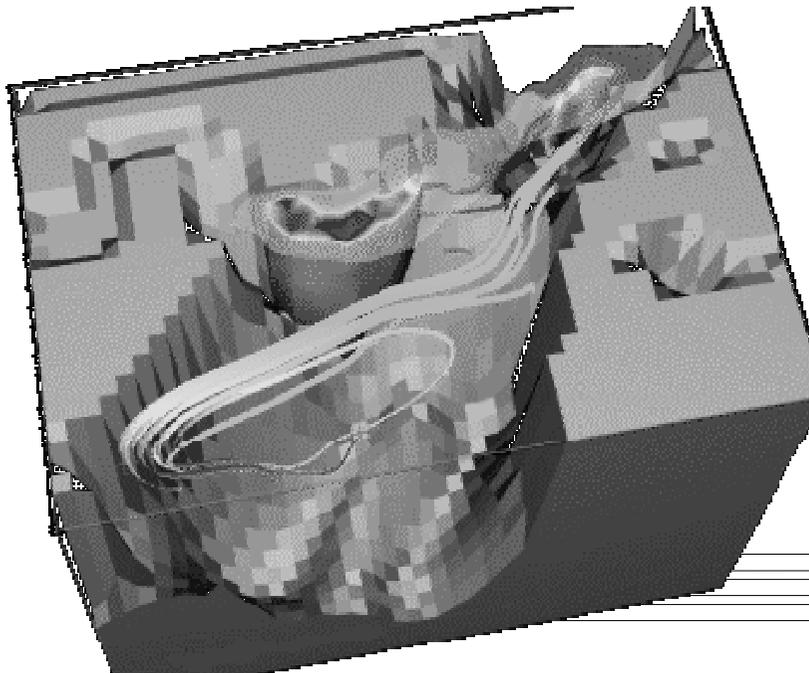
Vom Problem zum Programm



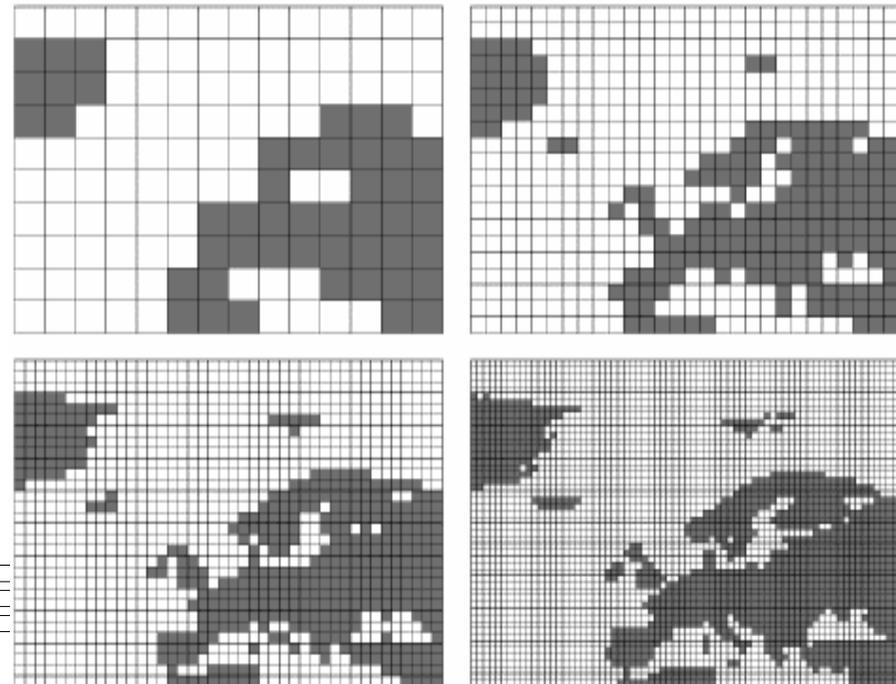
Grand Challenges

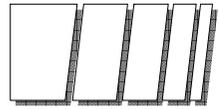
“... fundamentale Probleme in Wissenschaft und Technik, deren Lösung die Anwendung von Höchstleistungsrechnern erfordert.”

Strömungsmodelle:
Golfstrom



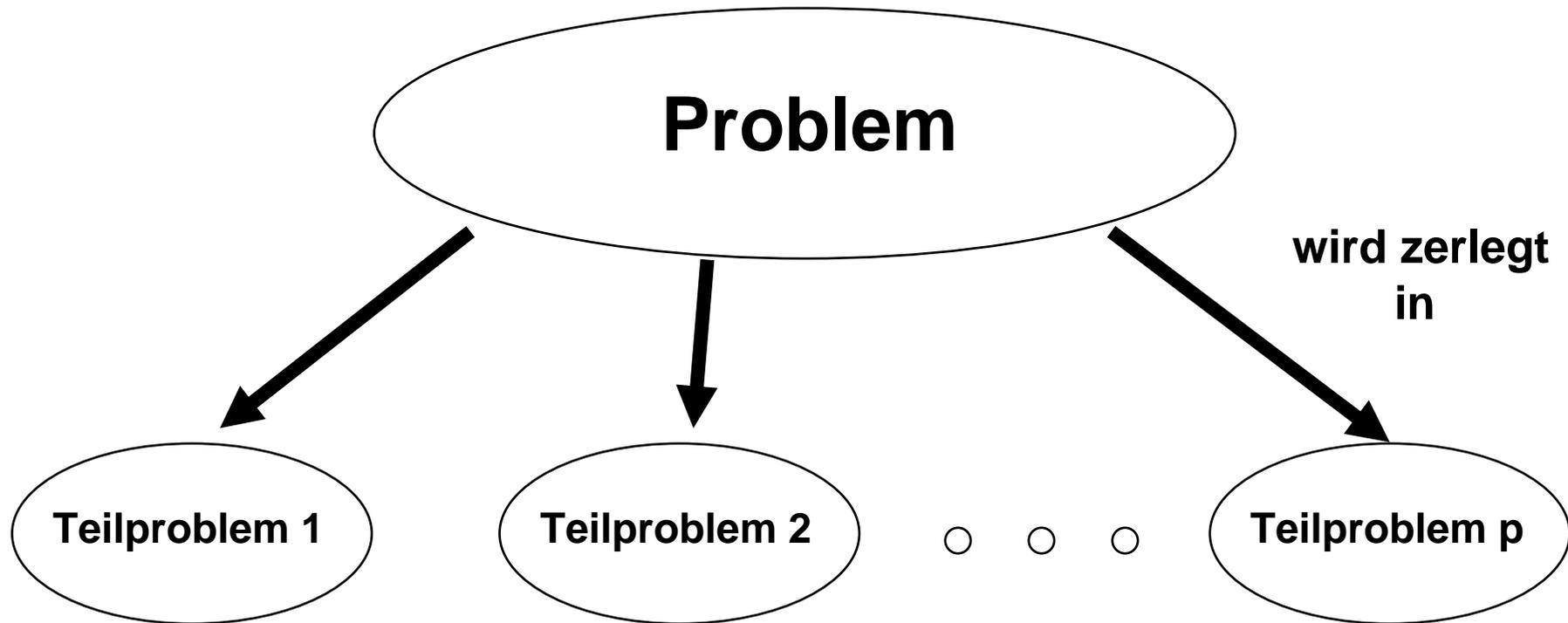
Wettervorhersage:
Europa



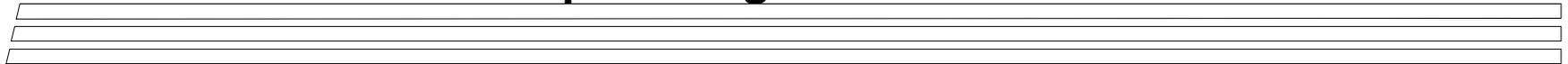


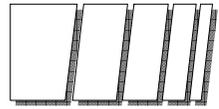
Grundprinzip paralleler Algorithmen

divide et impera



Zerlege Problem in Teile, die unabhängig sind und somit parallel gelöst werden können



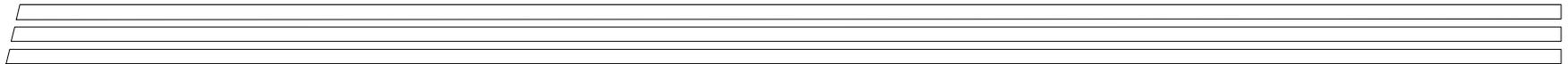


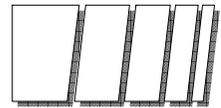
Vektoraddition

$$\begin{pmatrix} a_1 \\ a_2 \\ \cdot \\ \cdot \\ \cdot \\ a_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{pmatrix} = \begin{pmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \cdot \\ \cdot \\ \cdot \\ a_n + b_n \end{pmatrix}$$

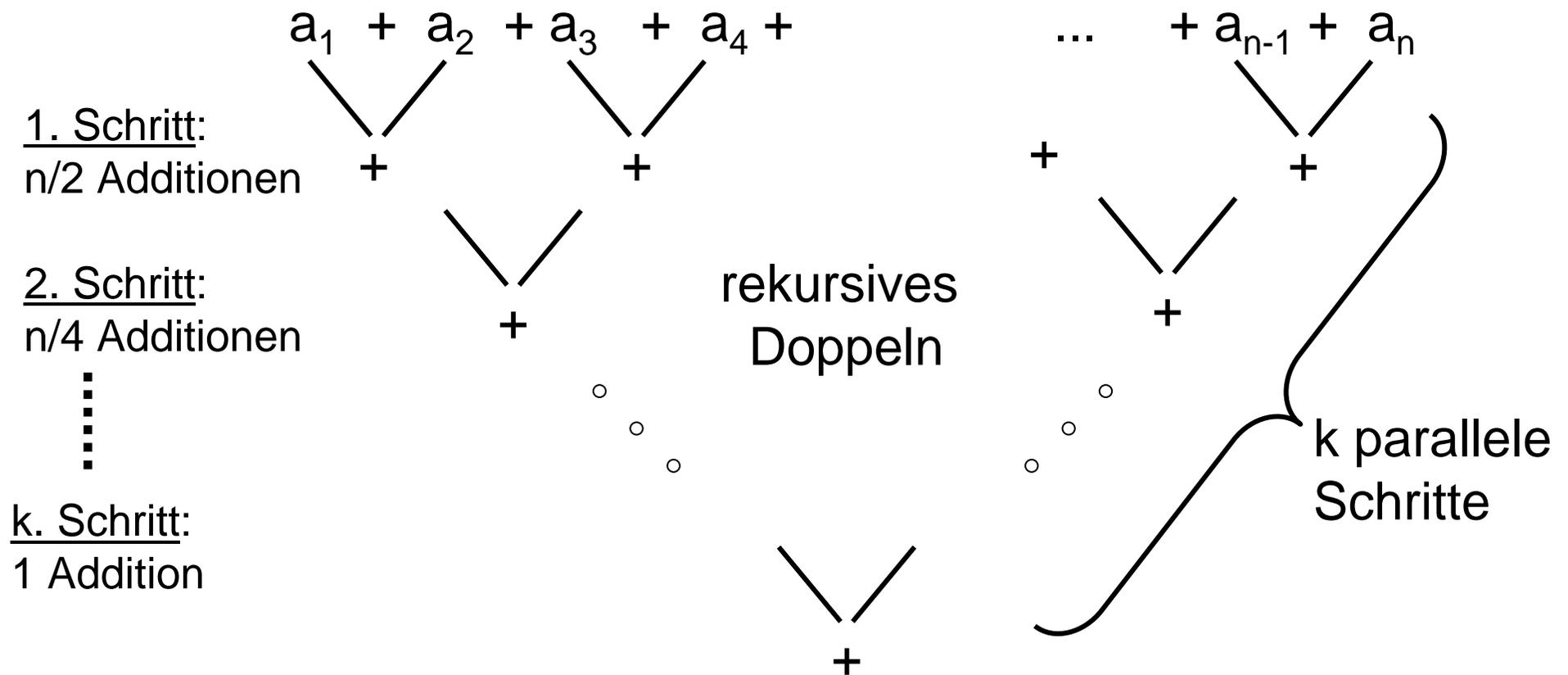
n unabhängige Teilaufgaben

=> Datenparallelität, feine Granularität

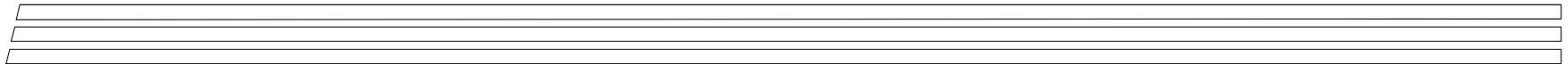


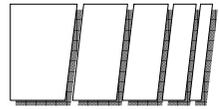


Summation von n Zahlen (mit $n=2^k$)



=> Kontrollparallelität, feine Granularität



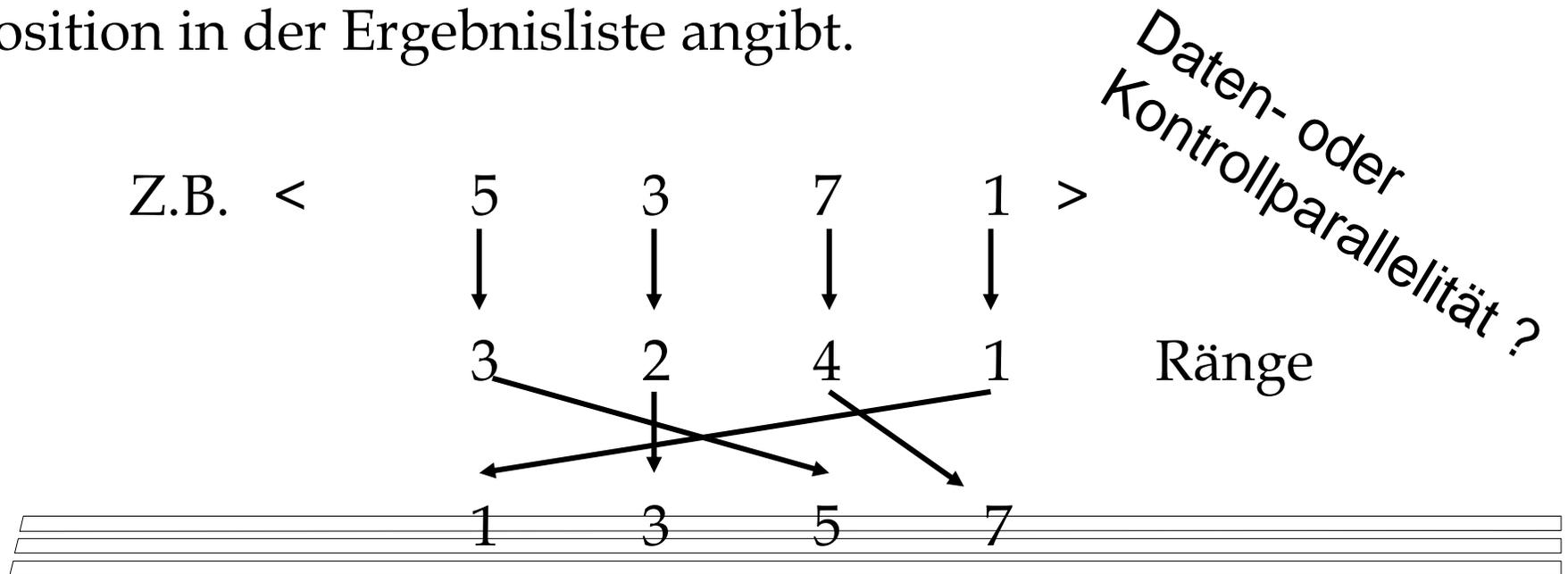


Sortieren von n Zahlen: RankSort

Sei $\hat{a} = \langle a_j \mid 1 \leq i \leq n \rangle, n \geq 1$ eine Folge von paarweise verschiedenen Zahlen.

Definiere $\text{rank}_{\hat{a}}(a_i) := |\{a_j \mid a_j \leq a_i\}|$ als Rang des i -ten Elementes von \hat{a} .

Berechne zu jedem Element parallel den Rang, der die Position in der Ergebnisliste angibt.



Ein inhärent sequentielles Problem

Berechnung großer Potenzen $y = x^{2^k}$ mit $k \gg 1$

Algorithmus: sukzessives Quadrieren:

$$(1) \quad x * x = x^2$$

$$(2) \quad x^2 * x^2 = x^4$$

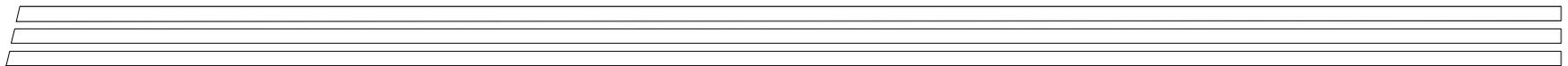
$$(3) \quad x^4 * x^4 = x^8$$

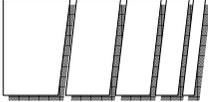
...

...

$$(k) \quad x^{2^{k-1}} * x^{2^{k-1}} = x^{2^k}$$

=> k sequentielle Schritte

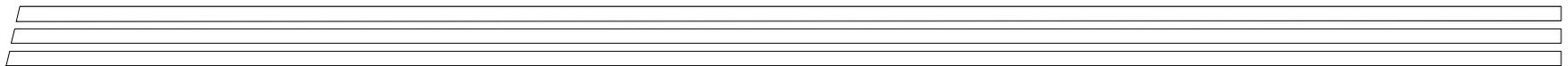




Begriffe

- Daten- vs. Kontrollparallelität
 - Datenparallelität basiert auf einer Zerlegung der Datenbasis.
 - Kontrollparallelität zerlegt die durchzuführenden Berechnungen.
- feinkörnige vs. grobkörnige Parallelität
 - Bei feinkörniger Parallelität (fine grain parallelism) bestehen die parallelen Aufgaben meist nur aus einer Instruktion/Operation.
 - Bei grobkörniger Parallelität (coarse grain parallelism) sind die parallelen Aufgaben komplexer.

Die Grenzen zwischen diesen Konzepten sind fließend.



Bewertung paralleler Algorithmen

Parameter:

- Problemgröße n (Größe und Anzahl der Eingaben)
- Anzahl der Prozessoren p

Zeitkomplexität / Laufzeit $T_p(n)$, auch $T_p(n, A)$ oder T_p :

= Anzahl der Schritte eines Verfahrens A bei Eingaben
der Größe n auf p Prozessoren
(=> Laufzeit auf p Prozessoren)

Sequentielle Zeit: $T_1(n)$

=> Algorithmus auf einem Prozessor oder
=> optimales (?) sequentielles Verfahren

In realen Messungen besteht $T_p(n)$ aus

Berechnungszeit + Kommunikationszeit + Leerlaufzeit

Beschleunigung (*Speedup*)

$$S_p(n) = T_1(n) / T_p(n)$$

Zentrale Größe, da Maß für Zeitgewinn durch Parallelisierung

absoluter Speedup: T_1 = Zeit des (opt.) seq. Algorithmus

relativer Speedup: T_1 = Zeit des par. Algorithm. auf 1 Prozessor

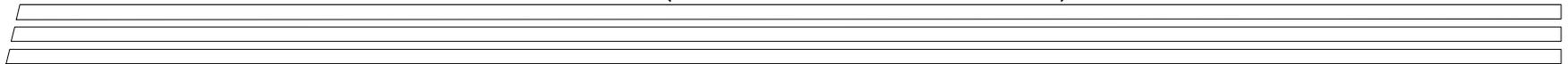
Normalerweise gilt: $1 \leq S_p(n) \leq p$

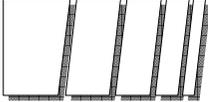
- Slowdown ($S_p(n) < 1$)

möglich wenn Mehraufwand der Parallelisierung Gewinn übersteigt

- superlinearer Speedup ($S_p(n) > p$)

Cache-Effekte, Suchverfahren (branch and bound)



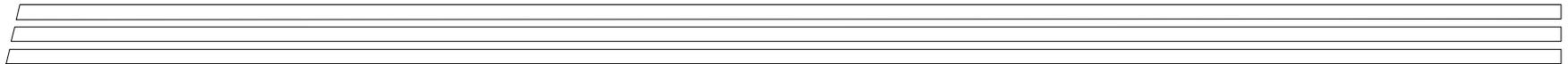


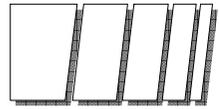
Effizienz

$$E_p(n) = S_p(n) / p$$

- Quotient aus erreichtem und theor. max. Speedup
- Maß für die Prozessorauslastung
(bei identischer Anzahl von Schritten im seq. und par. Fall)

$$E_p(n) = T_1(n) / (T_p(n) * p)$$





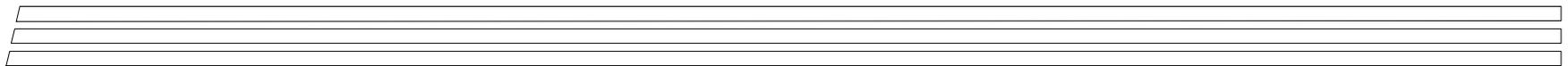
Weitere Kriterien

Kommunikationskomplexität: $C_p(n)$

- Maß für Datentransferaufwand bei parallelem Algorithmus
- hängt nicht nur vom Algorithmus, sondern insbesondere auch von der Rechnerarchitektur ab.
(=> starke Wechselwirkung zwischen Algorithmus und Rechnerarchitektur)

Skalierbarkeit

- Wie verhält sich der Algorithmus bei einer Erhöhung der Problemgröße oder Prozessoranzahl?
- Wie stark ist die Abhängigkeit des Algorithmus von der Rechnerkonfiguration?



Amdahls Gesetz

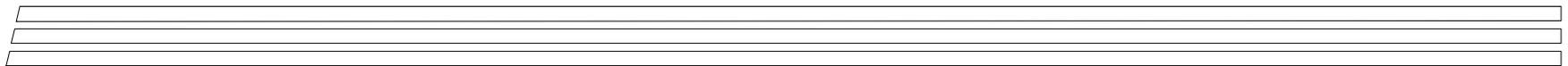
Sei A ein Algorithmus mit n Operationen, von denen ein Anteil f mit $0 < f < 1$ sequentiell ausgeführt werden muss. Sei $p > 1$.

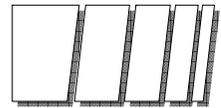
Dann gilt:

$$S_p(n) \leq \frac{1}{f + (1-f)/p} \leq \frac{1}{f}$$

Konsequenz: Der Anteil nicht-parallelisierbarer Berechnungen eines Algorithmus muss sehr klein sein.

Amdahl-Effekt: Bei fester Prozessoranzahl steigt die Beschleunigung mit wachsender Problemgröße.





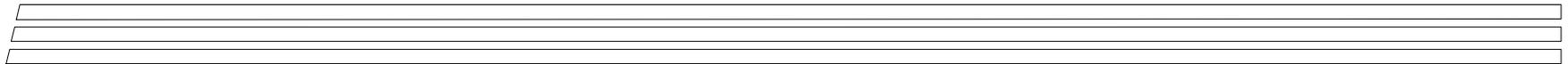
Gesetz von Gustafson und Barsis

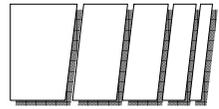
Verbesserung der Genauigkeit/Problemgröße eines parallelen Verfahrens durch Steigerung der Prozessorzahl bei gleich bleibender Ausführungszeit

Gegeben sei ein paralleles Programm mit der parallelen Laufzeit $T_p(n)$ auf p Prozessoren.

Sei s der Anteil der parallelen Ausführungszeit, der der Ausführung sequentiellen Codes gewidmet ist. Dann gilt:

$$S_p(n) \leq p + (1-p)*s$$

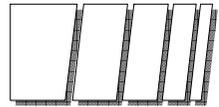




Rechnerklassifikation

Flynn 1966: grobe Einteilung nach Anzahl von Kontroll- bzw. Datenflüssen in einem Rechnersystem

Datenfluß Kontrollfluß	single data stream (SD)	multiple data stream (MD)
single instruction stream (SI)	SISD: serielle Monoprozessoren	SIMD: Vektor- bzw. Arrayrechner
multiple instruction stream (MI)	MISD: faktisch leer	MIMD: Mehrprozessorsysteme & verteilte Systeme



Parallele Rechner

Multiprozessorsysteme (MIMD)

speichergekoppelt

- gemeinsamer Adressbereich für alle Prozessoren
- Kommunikation und Synchronisation über gemeinsame Variablen

nachrichtengekoppelt

- prozessorlokale Adressbereiche; kein gemeinsamer Speicher
- Kommunikation und Synchronisation über Nachrichtenaustausch

➔ Abstraktion zu parallelen Programmiermodellen

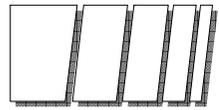
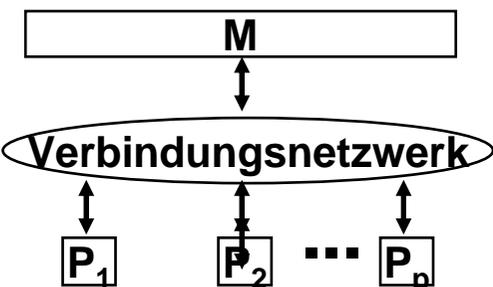
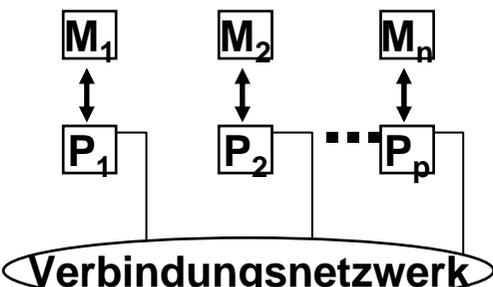
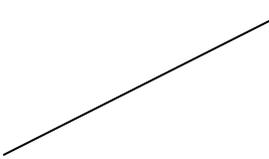
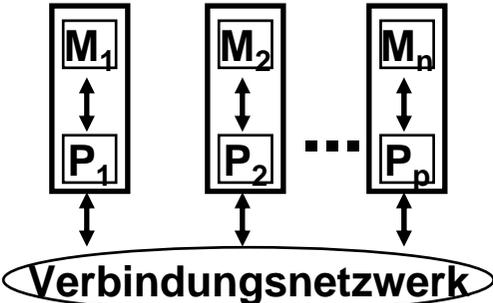
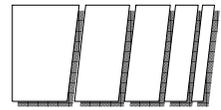


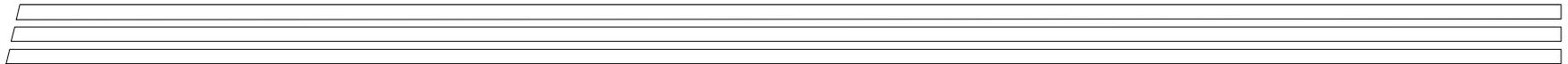
Abbildung von Adressräumen auf Speicher

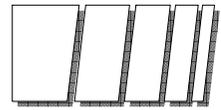
Speicher Adressraum	global	physikalisch verteilt
gemeinsam	<p>SMP - symmetrischer Multiproz.</p>  <p>UMA: uniform memory access</p>	<p>DSM - Distributed-Shared-Mem.</p>  <p>NUMA: non-uniform mem. access</p>
verteilt		<p>Multicomputer - Distributed Mem.</p> 



Gegenüberstellung und Zusammenfassung

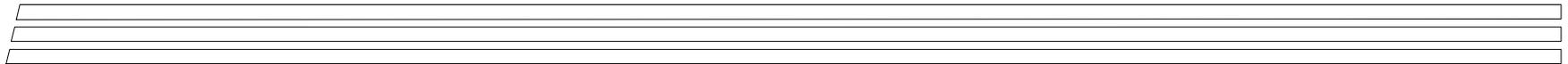
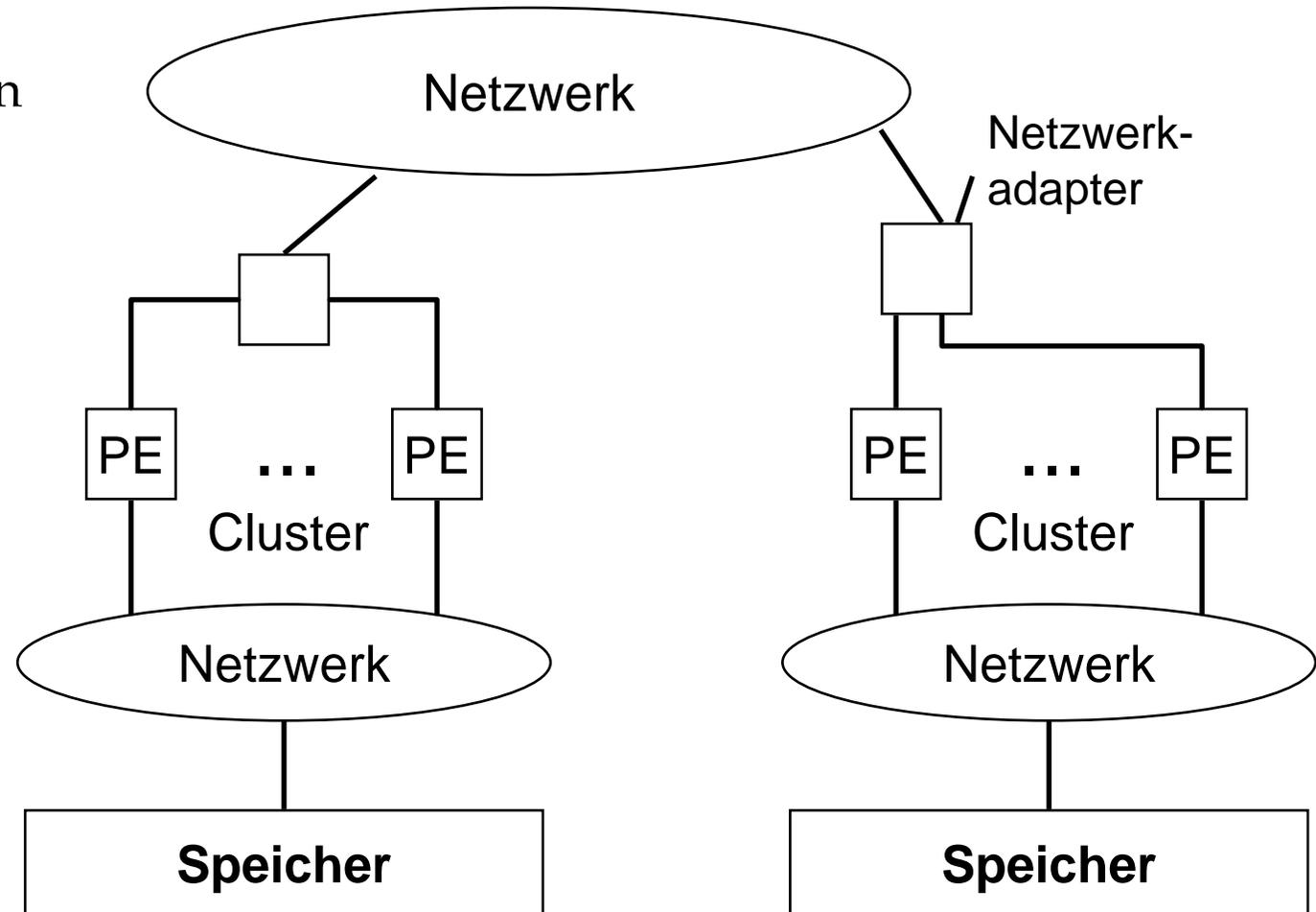
	SMP	DSM	DM
Adressraum	gemeinsam	gemeinsam	verteilt
Speicher	global	verteilt	verteilt
Komm./Synch.	gem. Vars	gem. Vars. (impl. Nachr.)	Nachr.
Skalierbarkeit	begrenzt	leicht	leicht
Lastverteilung	leicht	„leicht“	schwierig
Probleme	mangelnde Skalierbarkeit Speicherkonflikte	Cache- Kohärenz	globale Synchr. globale Inf.

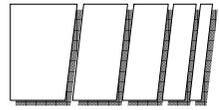




Mischformen, z.B. Clusterarchitekturen

verteilte Systeme von
Prozessorgruppen
mit gemeinsamem
Speicher



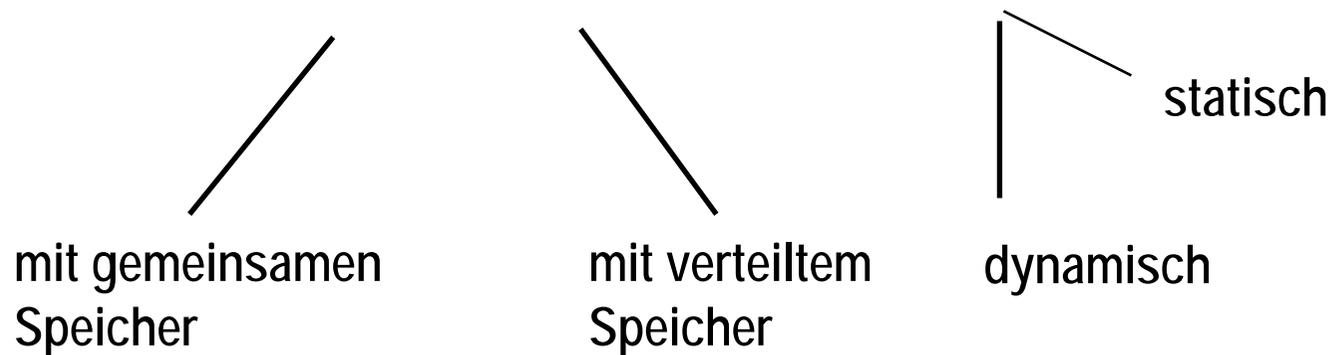


Paradigmen der Parallelverarbeitung

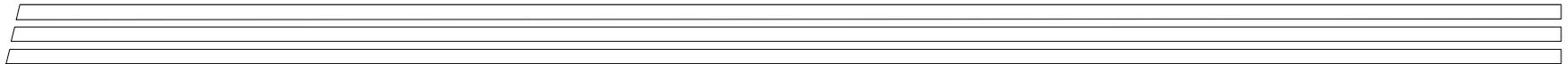
- **SIMD** → **synchrone Parallelität / Datenparallelität**
homogene Parallelität

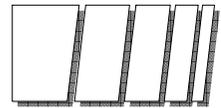
**"identische Prozesse, die auf verschiedenen Daten operieren",
automatische Synchronisation**

- **MIMD** → **asynchrone Parallelität, Prozessmodell**



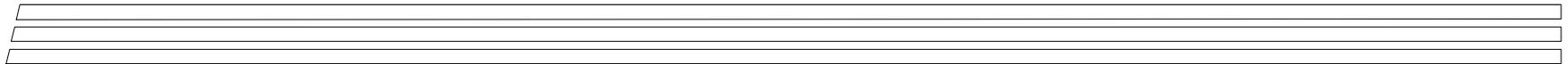
**"verschiedene Prozesse, die unabhängig unterschied-liche Daten
verarbeiten", explizite Synchronisation nötig**

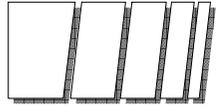




Parallele Programmiermodelle

- gemeinsamer Adressraum (shared memory)
 - Kommunikation über gemeinsame Variablen
 - Synchronisation über Schlossvariablen, Semaphore, Barrieren
 - Bsp: Linda, Java, SR, PRAM
- Nachrichtenkopplung: Datenaustausch über Nachrichten
 - Prozess-Kanal-Modell (Foster) Bsp: Occam, Eden
 - Prozess-Nachrichten-Modell Bsp: MPI
- Datenparallelität
 - Bereitstellung datenparalleler Grundoperationen
 - hoher Abstraktionsgrad
 - Bsp: HFP, Fortran90, NESL, Data-Parallel Haskell

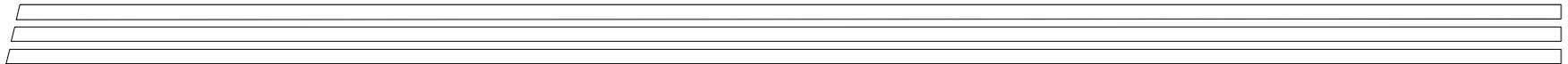


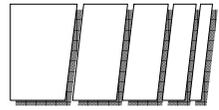


Prozess-Kanal- vs Prozess-Nachrichten-Modell

- Prozesse kapseln seq. Programme und Daten
- Prozessschnittstelle:
 - Eingangspforten (inports)
 - Ausgangspforten (outports)
- Grundaktionen eines Prozesses:
 - Senden und Empfangen von Nachrichten über die Pforten
 - lokale Berechnungen
 - neuen Prozess erzeugen -> dynamisches System
- Kanäle koppeln Aus- mit Eingangspforten
- Nachrichtenreihenfolge bleibt erhalten
- Prozesse werden auf Prozessoren abgebildet

- Prozesse werden über eine Prozessnummer identifiziert
- Sende-/ Empfangsanweisungen und Nachrichten enthalten Prozessnummern zur Angabe von Quelle und Ziel





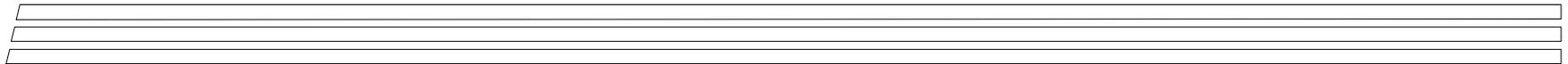
Das SPMD - Programmiermodell (Single Program Multiple Data)

Mischform SIMD - MIMD:



Dasselbe Programm wird zum Beginn der Berechnung auf eine feste Anzahl von Prozessoren geladen, arbeitet aber auf unterschiedlichen Datenbereichen, indem gemäß Prozessidentifikation verzweigt wird.

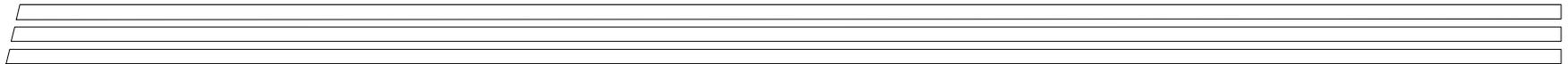
- statisches Prozesssystem
- Prozess-Nachrichten-Modell
- Modellierung von Datenparallelität





Prozesse vs Prozessoren

- Ein Prozess ist eine Instanz eines parallel ausführbaren Programmteils. Er beschreibt eine Teilaufgabe (task) des zu lösenden Problems.
- Ein Prozessor ist eine aktive Systemkomponente, die einen oder mehrere Prozesse ausführt.
- Die Abbildung Prozesse \rightarrow Prozessoren ist i.a. nicht injektiv. Sie bestimmt die Lastverteilung im parallelen System.



Der Prozessbegriff

Verschiedene Definitionen für den Begriff Prozess:

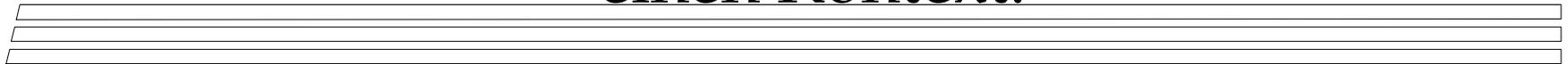
- sequentielle Folge von Aktivitäten, durch die eine in sich geschlossene Aufgabe bearbeitet wird

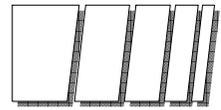
oder

- funktionale Einheit aus
 - zeitlich invariantem Programm
 - Satz von Daten
 - zeitlich variantem Zustand

=> in Bearbeitung befindliches Programm

Jeder Prozess besitzt eine Umgebung und einen Kontext.



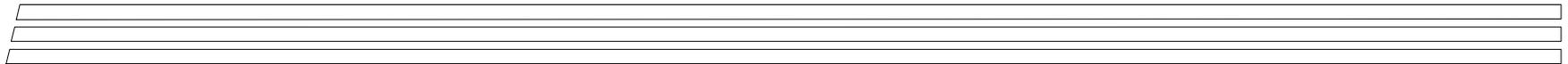


Umgebung und Kontext eines Prozesses

- Umgebung = geschützte Adressbereiche eines Prozesses
 - Code- und Datenbereiche im Speicher
 - geöffnete Dateien
 - Ressourcenverweise
- Kontext = Registerwerte des mit der Prozessausführung beschäftigten Prozessors
 - Befehlszähler
 - Zeiger auf Laufzeitkeller
 - Aktivierungsblock

Prozesswechsel => Umgebungs- und Kontextwechsel

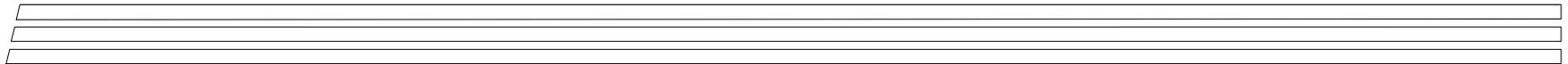
=> sehr aufwändig, daher Unterscheidung zwischen
Prozessen und Threads

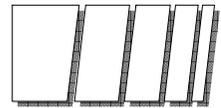




Prozesse vs Threads

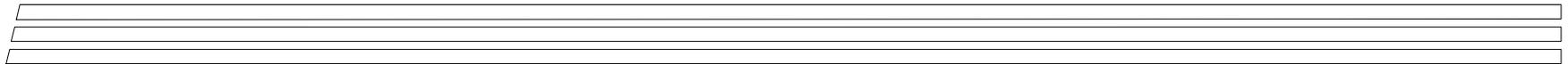
- Mehrere Threads (Kontrollfäden) teilen sich eine gemeinsame Umgebung.
Threadwechsel => nur Kontextwechsel
- Man nennt Threads daher auch leichtgewichtige Prozesse.
- Da Threads sich den Adressraum teilen, müssen ihre Speicherzugriffe synchronisiert werden.
- Moderne Betriebssysteme sind „multi-threaded“, unterstützen also das Threadkonzept.
- Seit 1996 gibt es eine standardisierte Threadschnittstelle des POSIX_Komitees.

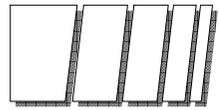




Historische Entwicklung: 60er Jahre

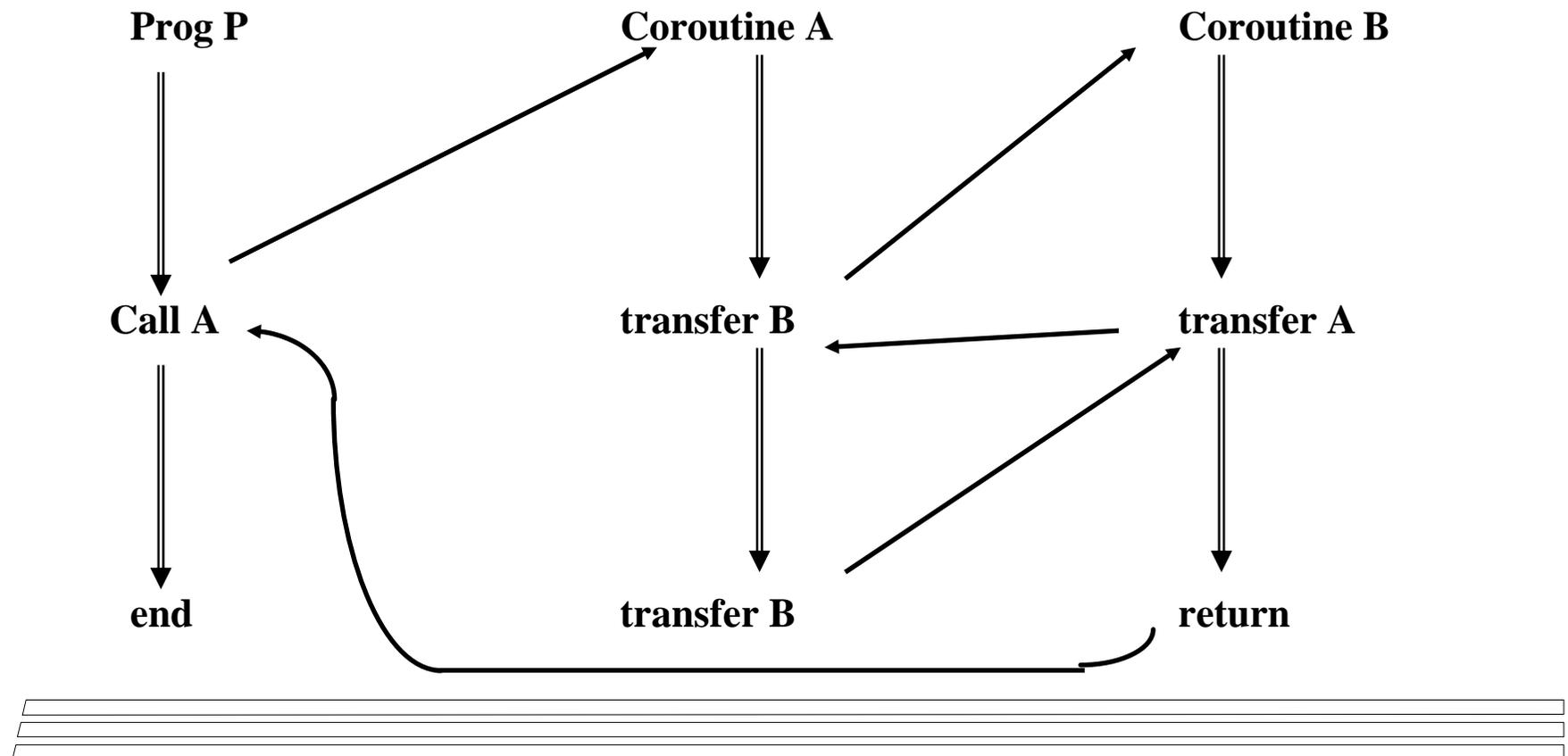
- Entwicklung erster Prozessmodelle und Konzepte zur Prozessinteraktion im Kontext von Mehrbenutzerbetriebssystemen (Multitasking)
- "Nebenläufigkeit" (Concurrency): unabhängige Ausführung verschiedener Jobs / Prozesse im Zeitscheibenverfahren auf einem Prozessor

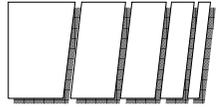




Conway 1963: Coroutinen

- keine echte Parallelverarbeitung, sondern Multiprogrammierung, d. h. mehrere Programme auf einem Prozessor
- kontrollierte Belegung und Freigabe des „Betriebsmittels“ Prozessor

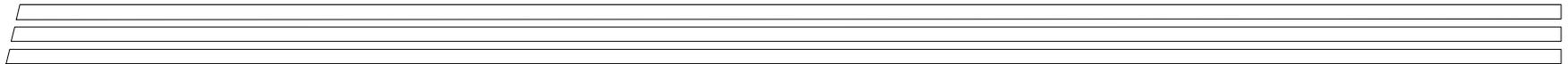
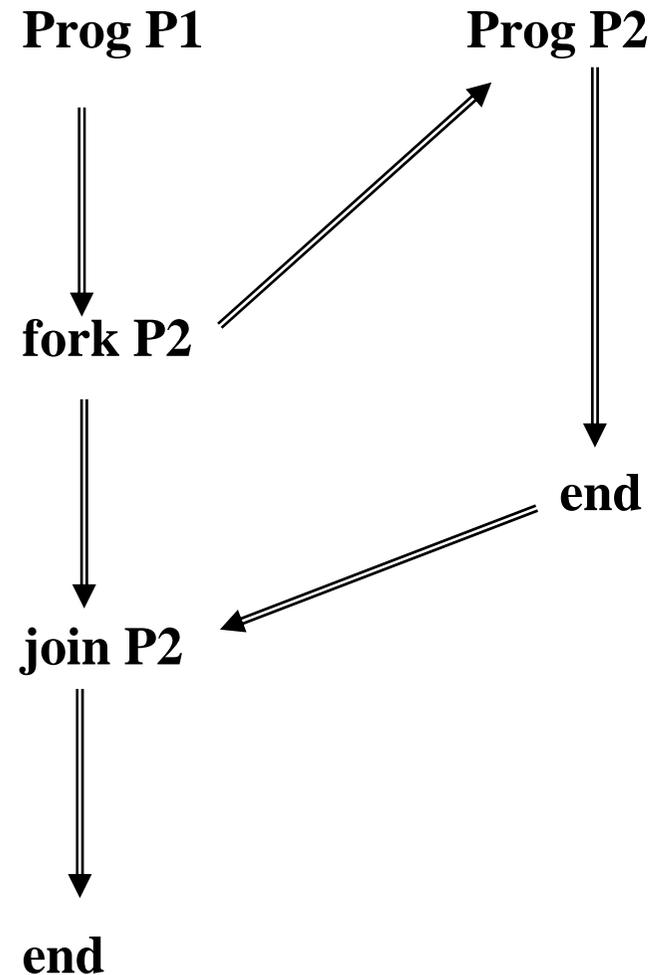


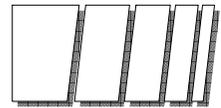


Dennis/van Horn 1966: fork/join

- fork startet einen parallelen Prozeß
- join zwingt das Programm, das den Prozess gestartet hat, zu warten, bis der parallele Prozess beendet ist
=> Synchronisation

UNIX verwendet Varianten von fork / join(wait).



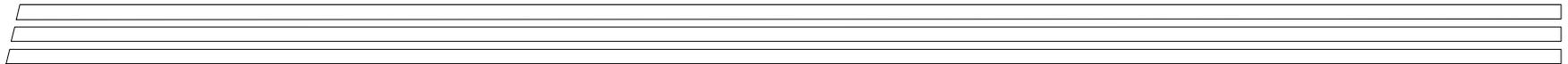


Dijkstra 1968: cobegin / coend

- Anweisungsblöcke, die mit cobegin/coend (oder parbegin/parend) geklammert sind, werden parallel ausgeführt:

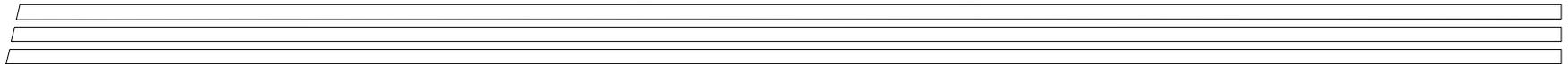
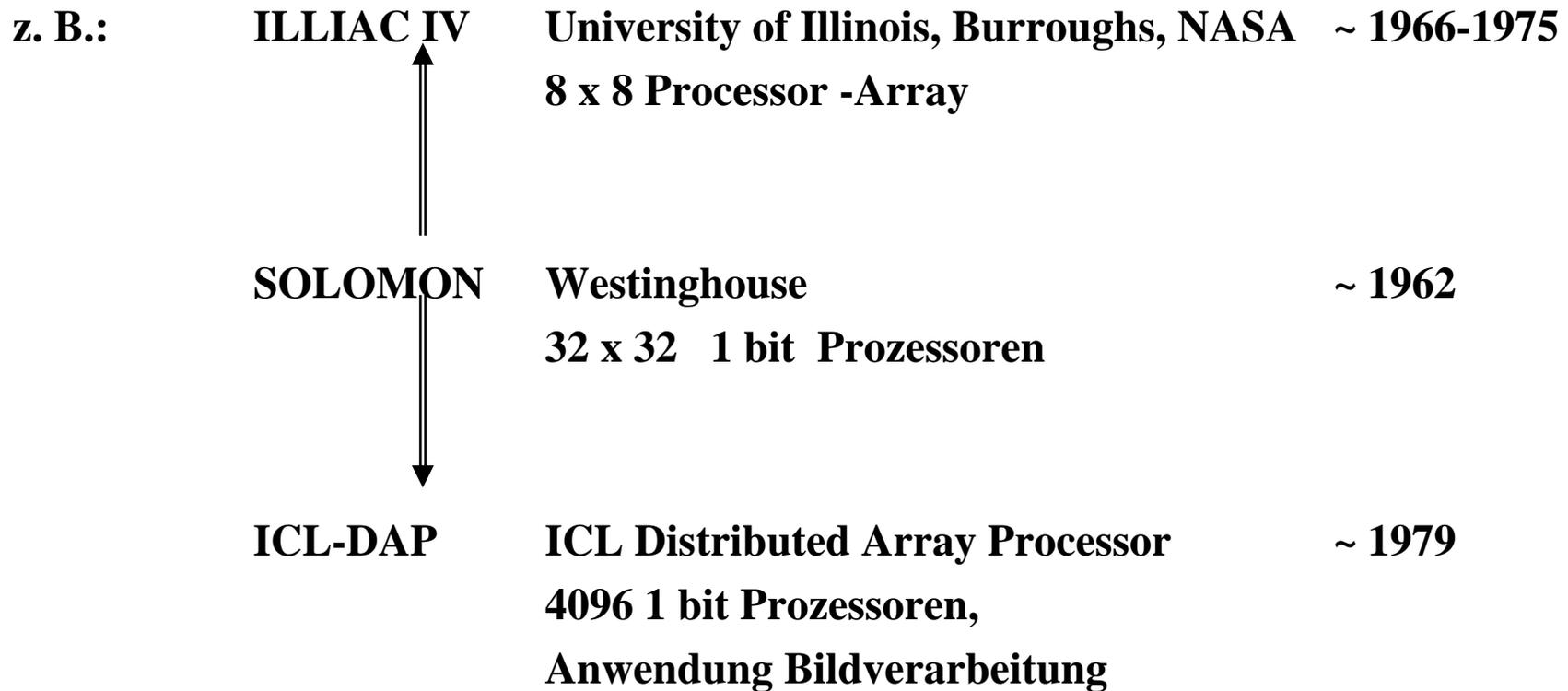
cobegin S1 | S2 | ... | Sm coend

- nicht so allgemein wie fork/join wegen fehlender Kommunikations- und Synchronisationskonzepte
- Mit entsprechenden Erweiterungen findet man ein ähnliches Konstrukt auch in der parallelen Programmiersprache MPD.



Erste Parallelrechner

Ende der 60er Jahre: Start erster Projekte an Universitäten und Forschungslaboratorien zur Entwicklung von Parallelrechnern, erste Prototypen gab es erst im Laufe der 70er Jahre

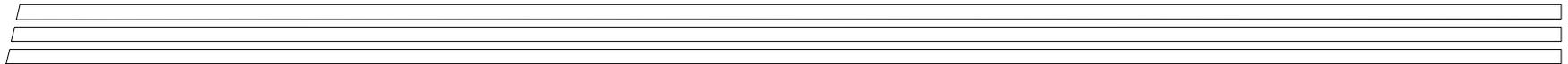


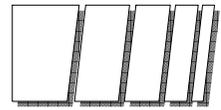
Amdahl 1967

Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, AFIPS Conf. Proc. 30: 483 – 485.

- Diese Arbeit enthält Argumente für die sequentielle und gegen die parallele Verarbeitung.
- Sie löste viele Diskussionen über Sinn und Wert der Parallelverarbeitung aus

$$\textit{Amdahls Gesetz: } S_p = \frac{1}{f + \frac{(1-f)}{p}}$$



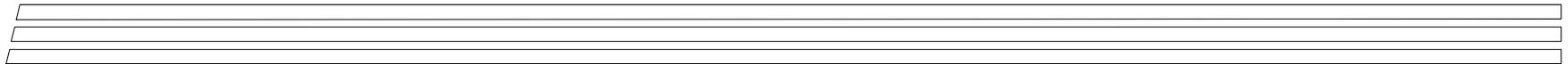


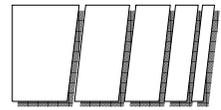
Historische Entwicklung: 70er Jahre

Entwicklung von Programmiersprachen mit Konzepten zur Erzeugung und Interaktion von Prozessen

(Semaphore, Monitore, Nachrichten, ...)

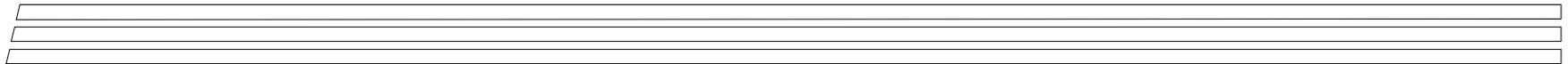
- *Brinch Hansen 1975*: Concurrent Pascal (= Pascal + Prozeß-Konzept)
- *Wirth 1977*: Modula
- *Hoare 1978*: Communicating Sequential Processes CSP
Vorläufer der Transputer-Sprache Occam (1984)
- *1973/74*: Untersuchungen zur parallelen Auswertung arithmetischer Ausdrücke
- *Fortune & Wyllie 1978*: Entwicklung der PRAM als Grundlage vieler theoretischer Komplexitätsuntersuchungen paralleler Algorithmen
- *VS DoD 1981*: Ada





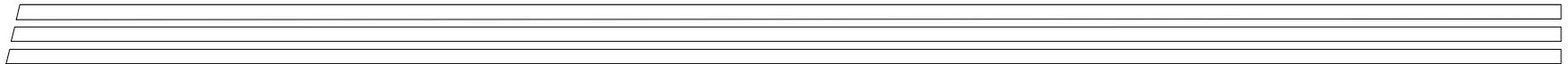
Parallelrechner Ende der 70er Jahre

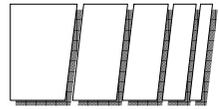
- Hochleistungsrechner, die auf Vektor- und Pipelineverarbeitung aufbauen, kommen auf den Markt:
 - CRAY-1 (Control Data) ~ 1976
 - CDC-Star, Cyber 205 ~ 1975/81
- außerdem verschiedene Parallelrechnersysteme:
 - BSP (Burrougs Scientific Processor) 1980
 - MPP (Goodyear Aerospace) 1979
 - massive parallel processors
 - Spezialsystem für Signal & Bildverarbeitung mit 128^2 (= 16384) Prozessoren



Historische Entwicklung: 80er Jahre

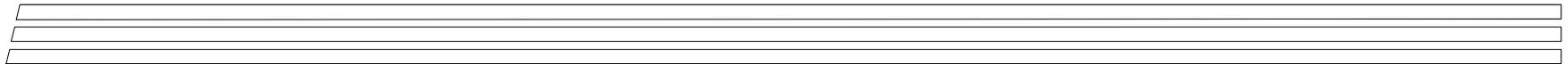
- Jahrzehnt des Super-Computing und damit auch der Parallelverarbeitung.
- maßgebliche Gründe dafür:
 - ▼ Verbreitung der VLSI-Technologie =>
1985: erster 32 Bit Mikroprozessor
 - ▼ Programmieretechniken wie Semaphore, Monitore, Signale ...
setzen sich als Standard durch => Real Time Programming
 - ▼ Vektor-Supercomputer bringen erhebliche Leistungssteigerungen,
aber auch Forderungen nach weiterer Leistung
 - ▼ erste Erfahrungen mit realen Parallelrechnern

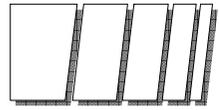




Einige Parallelrechner der 80er Jahre

- SIMD:
 - DAP - Weiterentwicklung des ICL-DAP durch AMT (Active Memory Technology)
 - MP1 - MasPar Computer Corporation, bis zu 16384 PE's
 - CM - Connection Machine, Thinking Machine Corp., Ph. D. Daniel Hillis 1985, bis zu 65536 PE's 1-bit
- SM-MIMD (Shared Memory):
 - Allient- FX2800 bis zu 14 PE's (7*2 Module), nicht busorientiert
 - Sequent Symmetry, bis zu 30 PE's, busorientiert
- DM-MIMD (Distributed Memory):
 - Intel iPSC/860, Hypercube-Organisation, bis 128 PE's = 2^7
 - Inmos Transputer basierte Systeme, bis 256 PE's

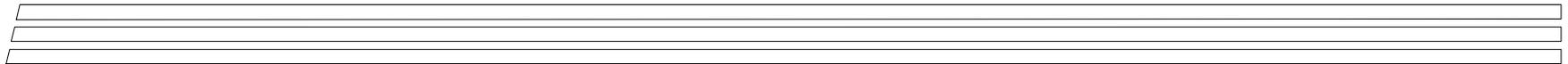


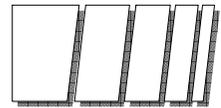


Historische Entwicklung: 90er Jahre

beginnende Konvergenz von Konzepten auf Hardware und Software-Ebene:

- Einsatz von Standard-Mikroprozessoren und Hochgeschwindigkeitsnetzwerken
- verteilt realisierte gemeinsame Speicher:
NUMA (Non-Uniform Memory Access) shared memory,
distributed shared memory
- Trend zur Unterstützung mehrerer Programmiermodelle
- “Standards” auf Software Ebene:
 - MPI - Message Passing Interface
 - HPF- High Performance Fortran Forum, Datenparallelität
 - OpenMP- Shared-Memory Compiler-Direktiven

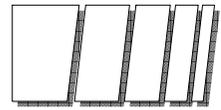




Einige Parallelrechner der 90er Jahre

- UMA (Uniform Memory Access)-Shared Memory:
 - Silicon Graphics Inc. (SGI) Challenge 1993, busorientiert, bis zu 16 Prozessoren
 - SUN Ultrasparc, 256 Bit breiter Bus (2.5 GB/s), bis zu 30 Prozessoren
- NUMA-Shared Memory:
 - KSR-1 (Kendall Square Research), 1993
 - Cray T3E(DEC Alpha Processor), dreidimensionales Gitter mit bis zu 1000 Prozessoren, globaler Adreßraum, keine Hardware zur Erhaltung der Cache-Konsistenz
- Distributed Memory:
 - IBM SP-2, 1993, RS 6000 Workstations mit Omega-Netzwerk
 - Intel Paragon, 1992, i860 Prozessoren, dreidimensionales Gitter
 - Beowulf, 1994, 16 Intel DX-Prozessoren mit Ethernet-Links,

nur COTS (commodity off-the-shelf) Komponenten



Entwicklung der Parallelrechner-Firmen

Firma/Land	Jahr	Status in 2001
Sequent/U.S.	1984	von IBM übernommen
Meiko /U.K.	1985	bankrott
nCube/U.S.	1985	Umorientierung
Parsytec/D.	1985	Umorientierung
Alliant/U.S.	1985	bankrott
Floating Point S./U.S.	1986	von Sun übernommen
Silicon Graphics/U.S.	1988	aktiv
KSR/U.S.	1992	bankrott
IBM/U.S.	1993	aktiv
Sun Micros./U.S.	1993	aktiv
Cray Research/U.S.	1993	aktiv

Aktueller Stand

- Die meisten Probleme sind inhärent parallel. Aber parallele Programmierung ist viel schwieriger als sequentielle.
- www.top500.org: Liste der schnellsten Rechner der Welt
- Mehrkernrechner als Knoten (seit Anfang 2000er Jahre)
=> Clusterarchitekturen mit Mehrkernrechnern
- Parallele Programmierung mit C, C++, Fortran plus
 - MPI
 - OpenMP
- Abstraktere Ansätze
 - Datenparallelität (HPF, Data Parallel Haskell)
 - Skelettbibliotheken (Algorithmische Skelette)
 - Parallele deklarative Sprachen

