Rechnerstrukturen

Vorlesung im Sommersemester 2006

Prof. Dr. Wolfgang Karl

Universität Karlsruhe (TH)

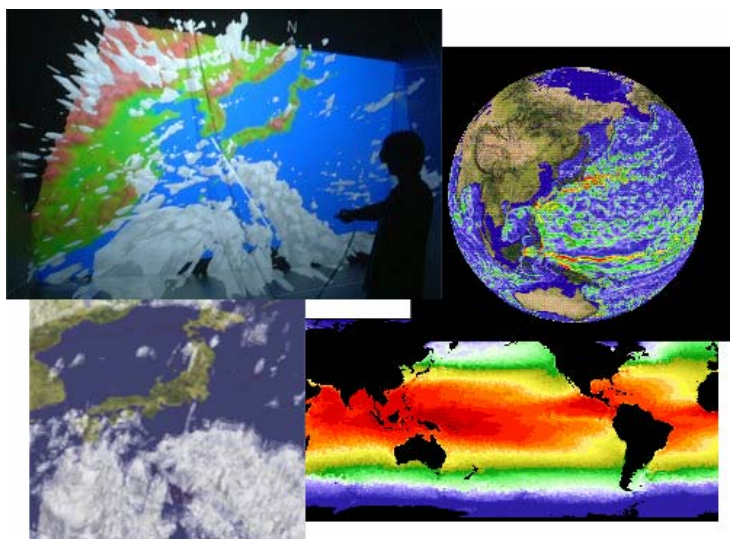
Fakultät für Informatik

Institut für Technische Informatik

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.1: Motivation

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan, Platz 7 (TOP500, Nov. 05)
      - Anzahl Prozessoren: 5120
      - Leistung: 35,86 TFLOPS (Linpack),
      - Anwendung: Klimaforschung



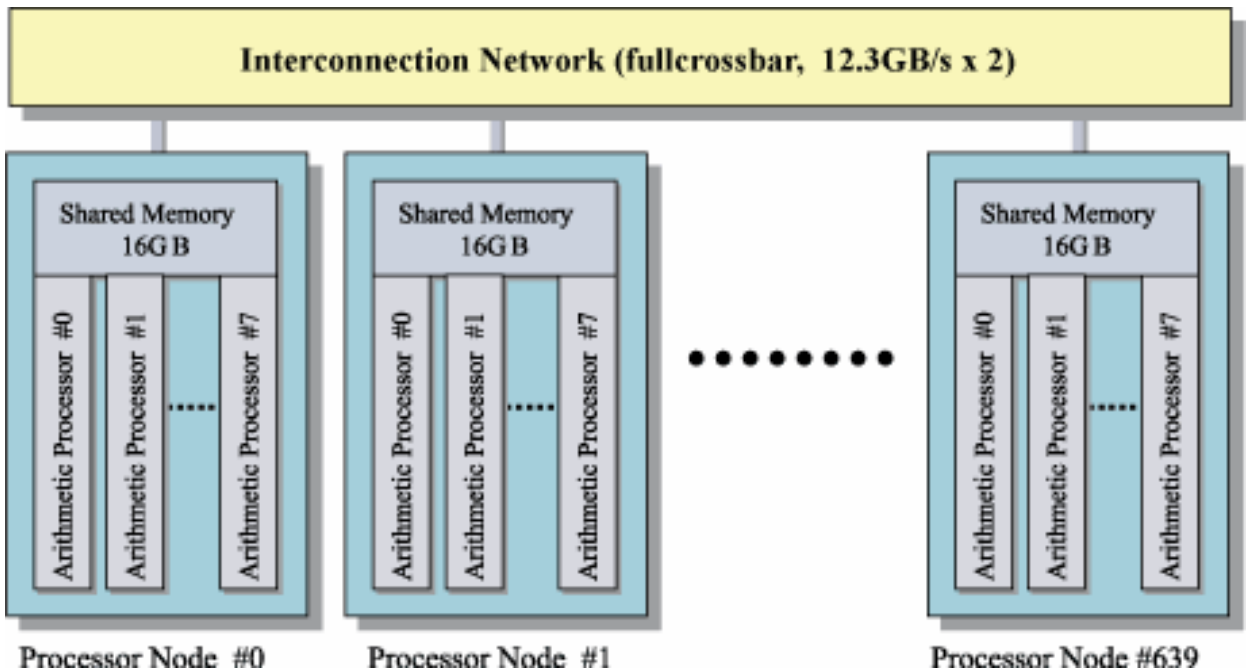
Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan, Platz 7 (TOP500, Nov. 05)
      - Ziel des Earth Simulator Project:
        - » „The Earth Simulator Project will create a "virtual earth" on a supercomputer to show what the world will look like in the future by means of advanced numerical simulation technology.“
        - » „Achievement of high-speed numerical simulations with processing speed of 1000 times higher than that of the most frequently used supercomputers in 1996.“

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan, Platz 7 (TOP500, Nov. 05)
      - "Understanding and Prediction of Global Climate Change
        - » Occurrence prediction of meteorological disaster
        - » Occurrence prediction of El Niño
        - » Understanding of effect of global warming
        - » Establishment of simulation technology with 1km resolution"

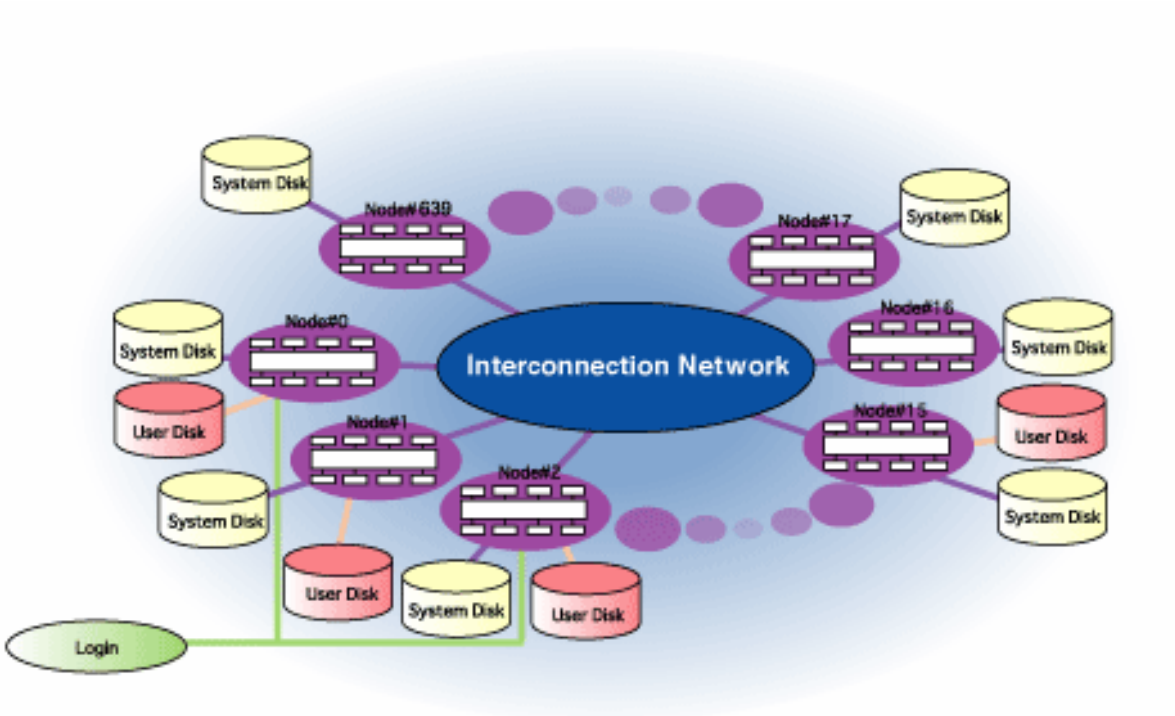
- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan, Platz 7 (TOP500, Nov. 05)
      - "Understanding of Plate Techtonics
        - » Understanding of long-range crustal movements
        - » Understanding of mechanism of seismicity
        - » Understanding of migration of underground water and materials transfer in strata"

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan)
      - Systemkonfiguration



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/eng/Hardware/system.html

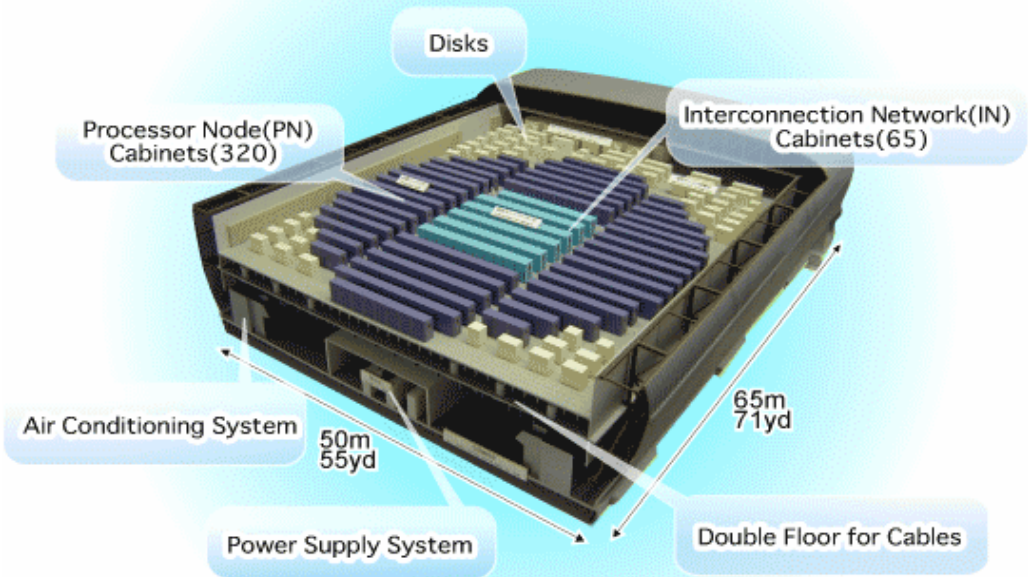
- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan)



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

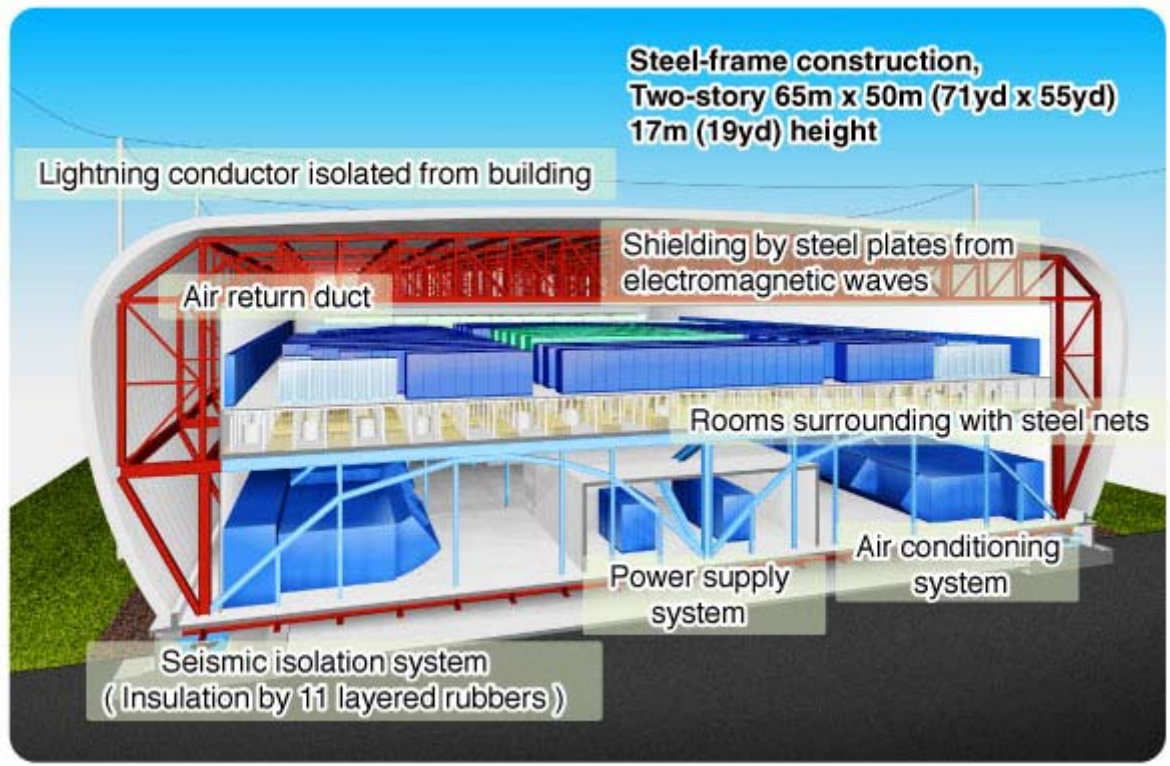
- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan)

-



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

- Motivation
  - Höchstleistungsrechner:
    - Earth Simulator (Japan)

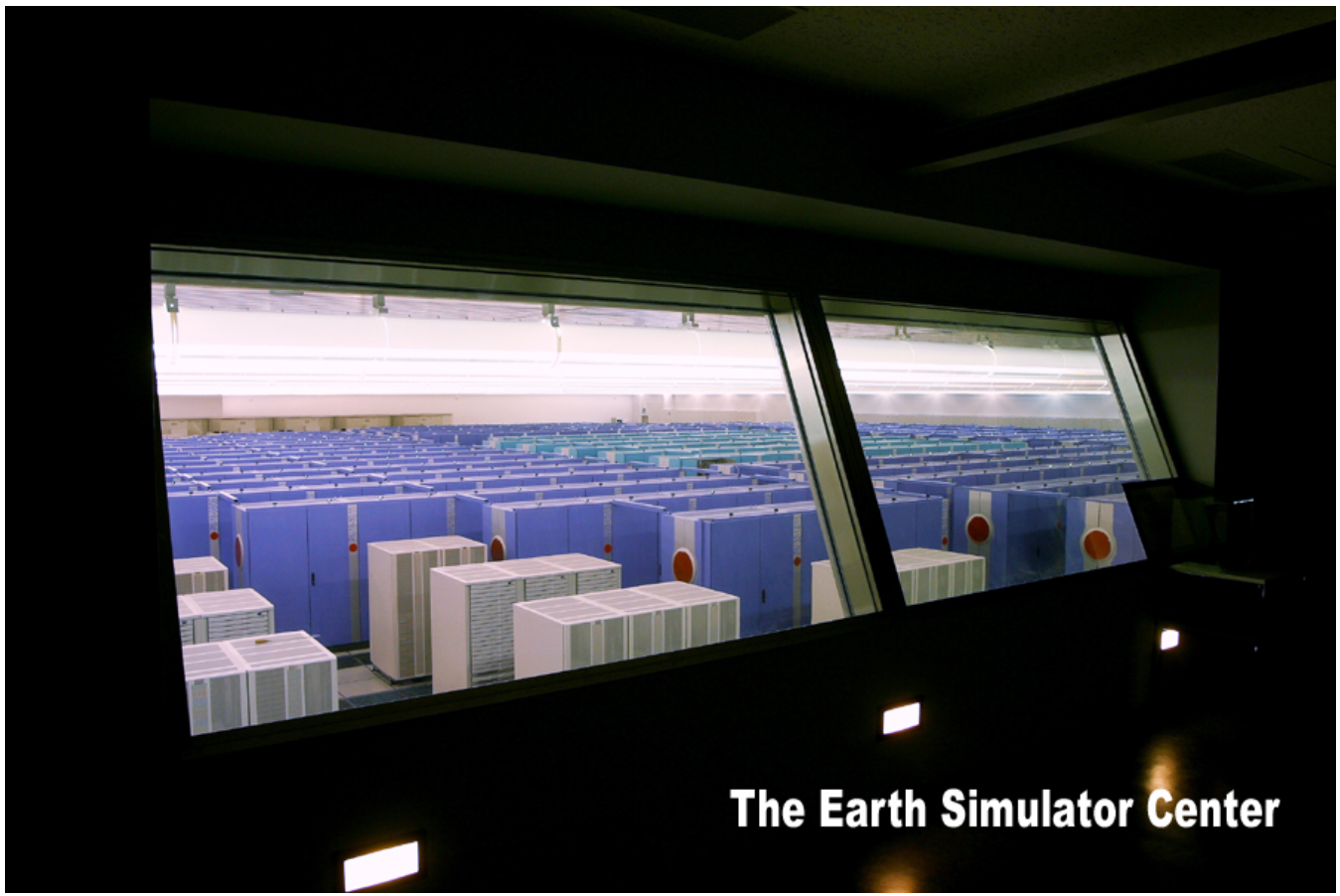


Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

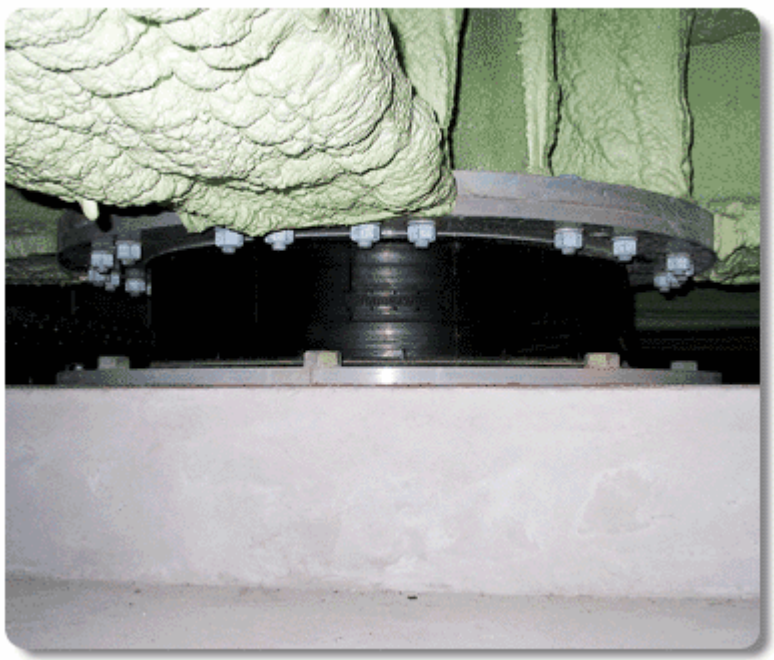
- Earth Simulator

Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/eng/GC/index.html

- Earth Simulator



- Earth Simulator
  - Erdbebenschutz:



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

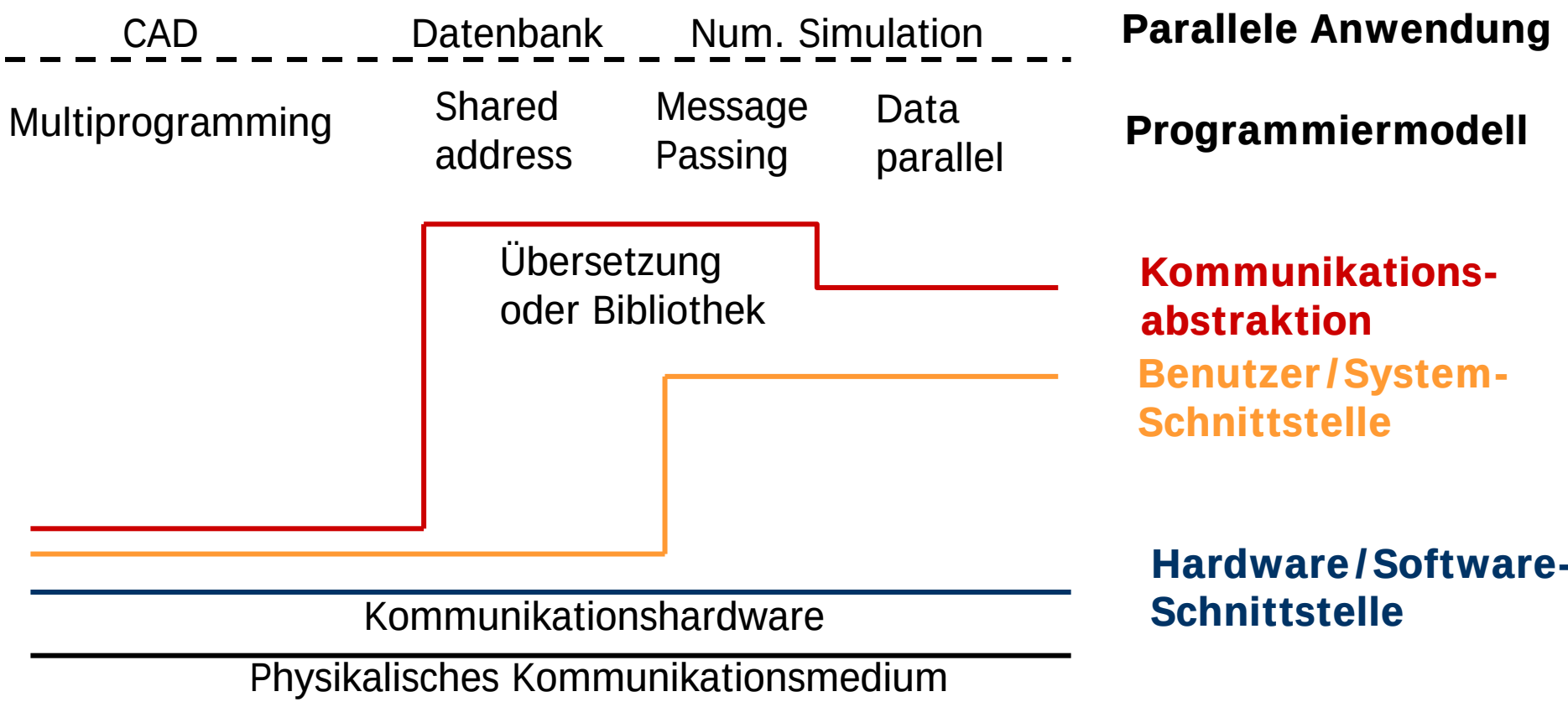
3.2: Allgemeine Grundlagen

- Parallele Architekturen
  - Definition:
    - Parallelrechner:
      - *„A collection of processing elements that communicate and cooperate to solve large problems“* (Almase and Gottlieb, 1989)
      - Betrachtung einer parallelen Architektur als eine Erweiterung des Konzepts einer konventionellen Rechnerarchitektur um eine Kommunikationsarchitektur

- Rechnerarchitektur
  - Abstraktion
    - Benutzer-/System-Schnittstelle
    - Hardware-/Software-Schnittstelle
  - Architektur
    - Spezifiziert die Menge der Operationen an den Schnittstellen und die Datentypen, auf denen diese operieren
  - Organisation
    - Realisierung der Abstraktionen

- Kommunikationsarchitektur
  - Abstraktion
    - Benutzer-/System-Schnittstelle
    - Hardware-/Software-Schnittstelle
  - Architektur
    - Spezifiziert die Kommunikations- und Synchronisationsoperationen
  - Organisation
    - Realisierung dieser Operationen

- Parallele Architekturen
  - Abstraktion

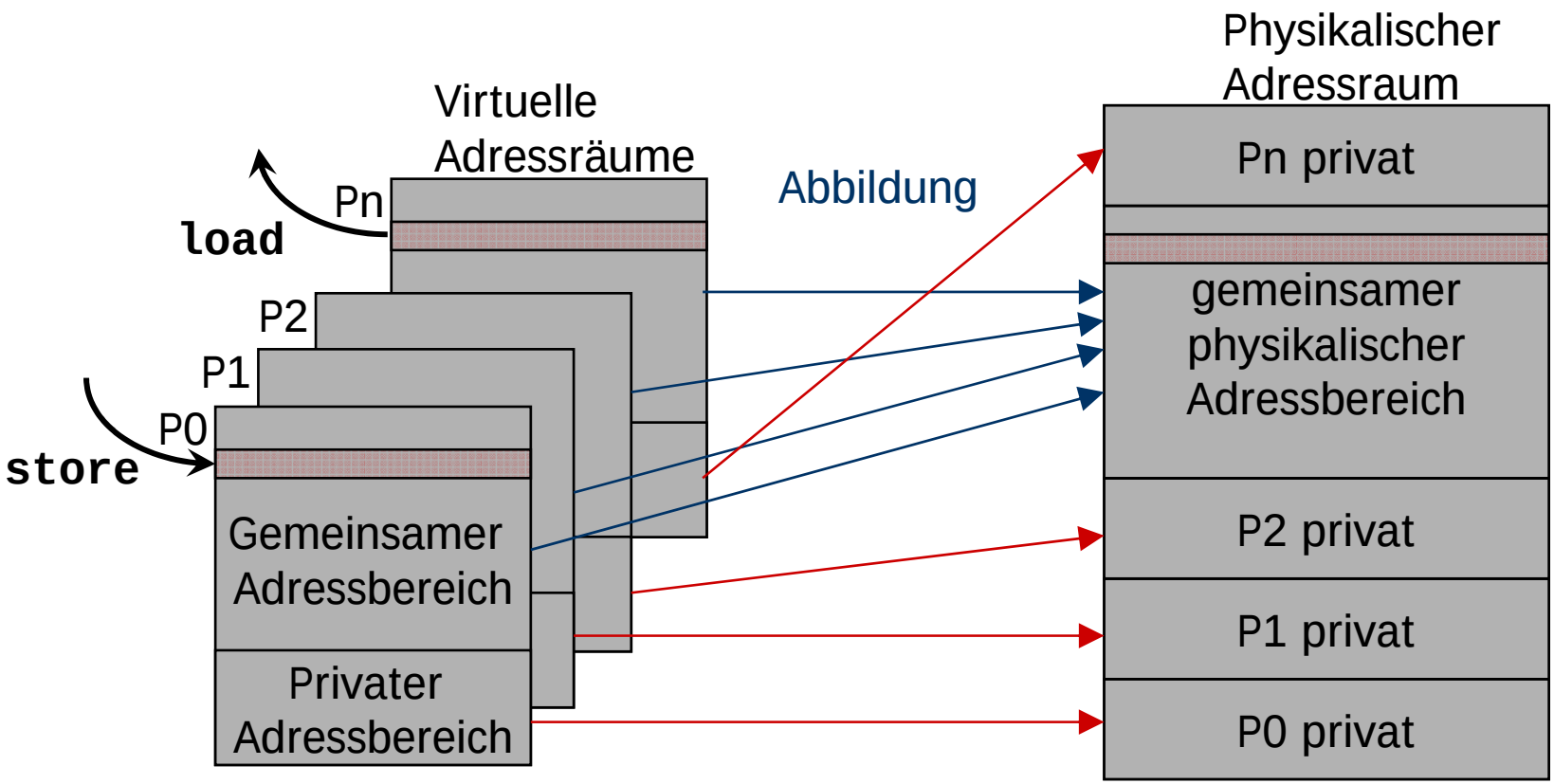


- Parallele Architekturen
  - Programmiermodell
    - Abstraktion einer parallelen Maschine, auf der der Anwender sein Programm formuliert
    - Spezifiziert, wie Teile des Programms parallel abgearbeitet werden, wie Informationen ausgetauscht werden und welche Synchronisationsoperationen verfügbar sind, um die Aktivitäten zu koordinieren
    - Anwendungen werden auf der Grundlage eines parallelen Programmiermodells formuliert

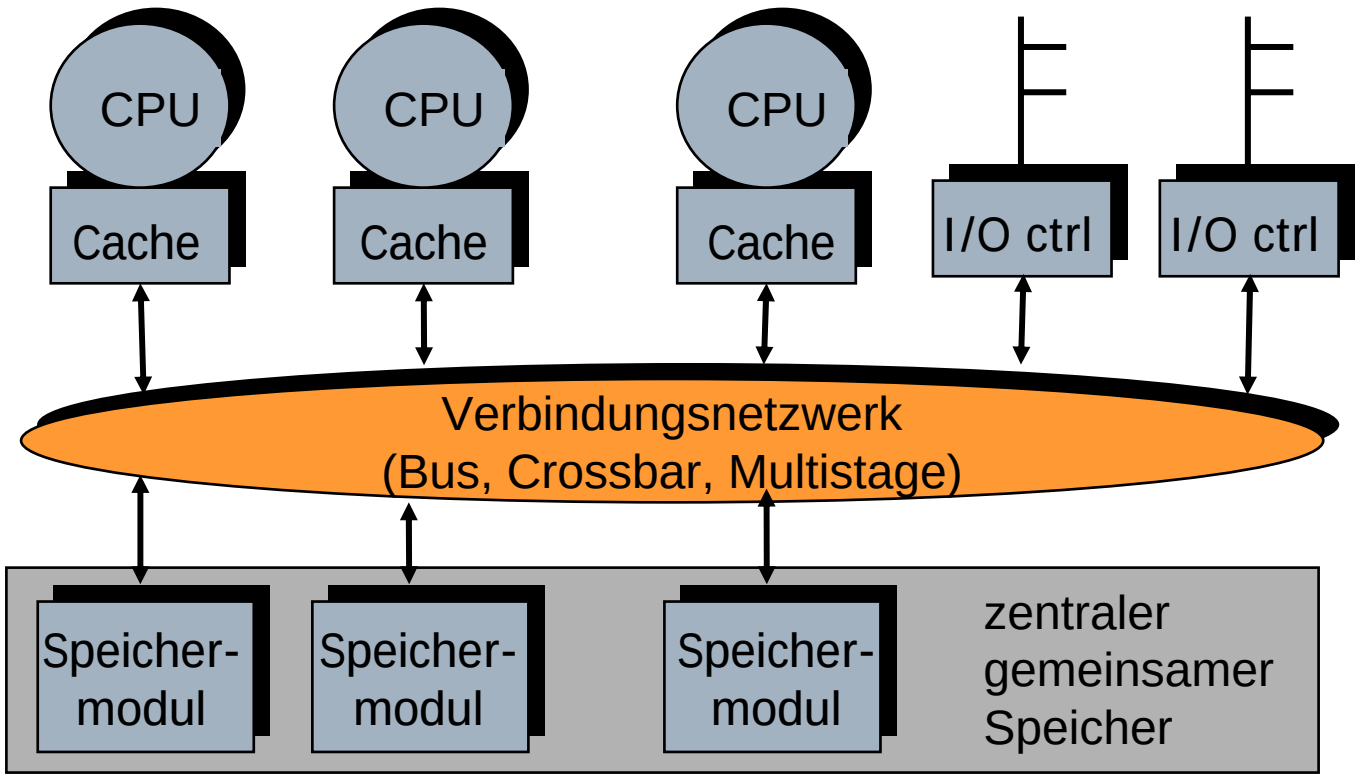
- Parallele Architekturen
  - Programmiermodell
    - Multiprogramming
      - Menge von unabhängigen sequentiellen Programmen
      - Keine Kommunikation oder Koordination

- Parallele Architekturen
  - Programmiermodell
    - Gemeinsamer Speicher (Shared Address Space)
      - Kommunikation und Koordination von Prozessen (Threads) über gemeinsame Variablen und Zeiger, die gemeinsame Adressen referenzieren
      - Kommunikationsarchitektur
        - » Verwendung konventioneller Speicheroperationen für die Kommunikation über gemeinsame Adressen
        - » Atomare Synchronisationsoperationen

- Parallele Architekturen
  - Gemeinsamer Speicher (Shared Address Space)

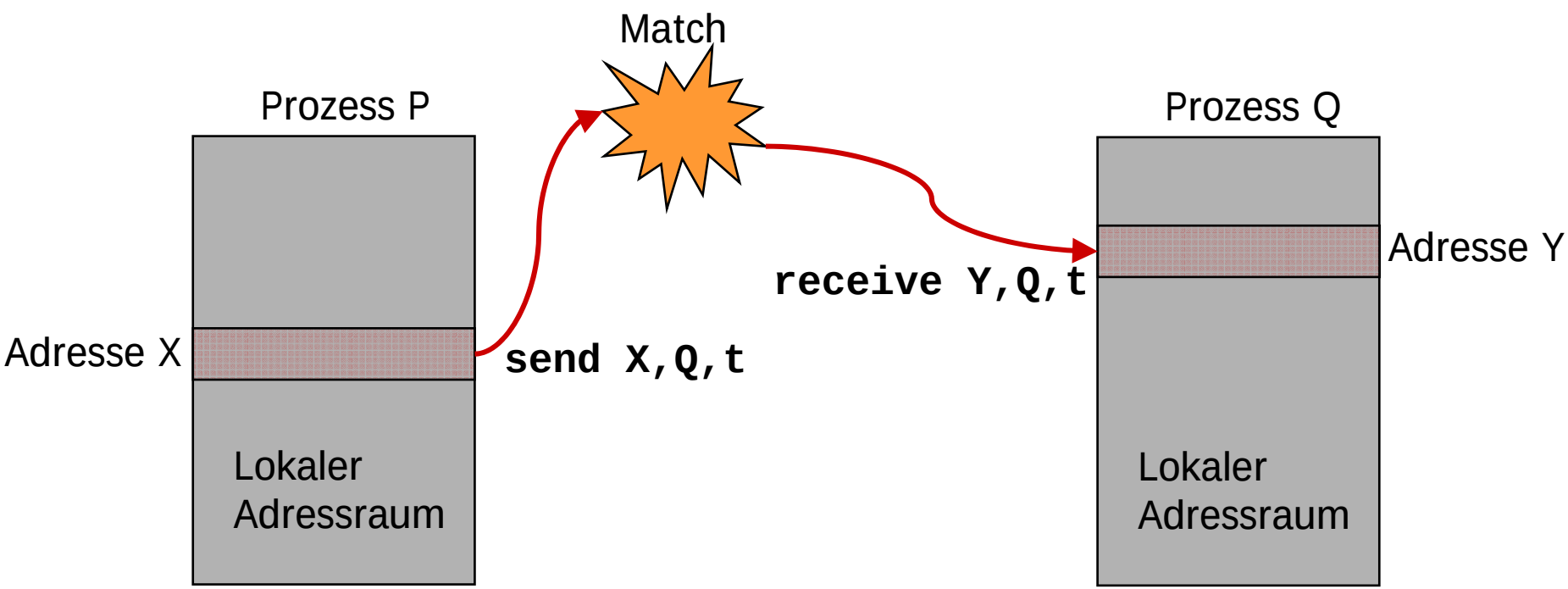


- Parallele Architekturen
  - Multiprozessor mit gemeinsamem Speicher

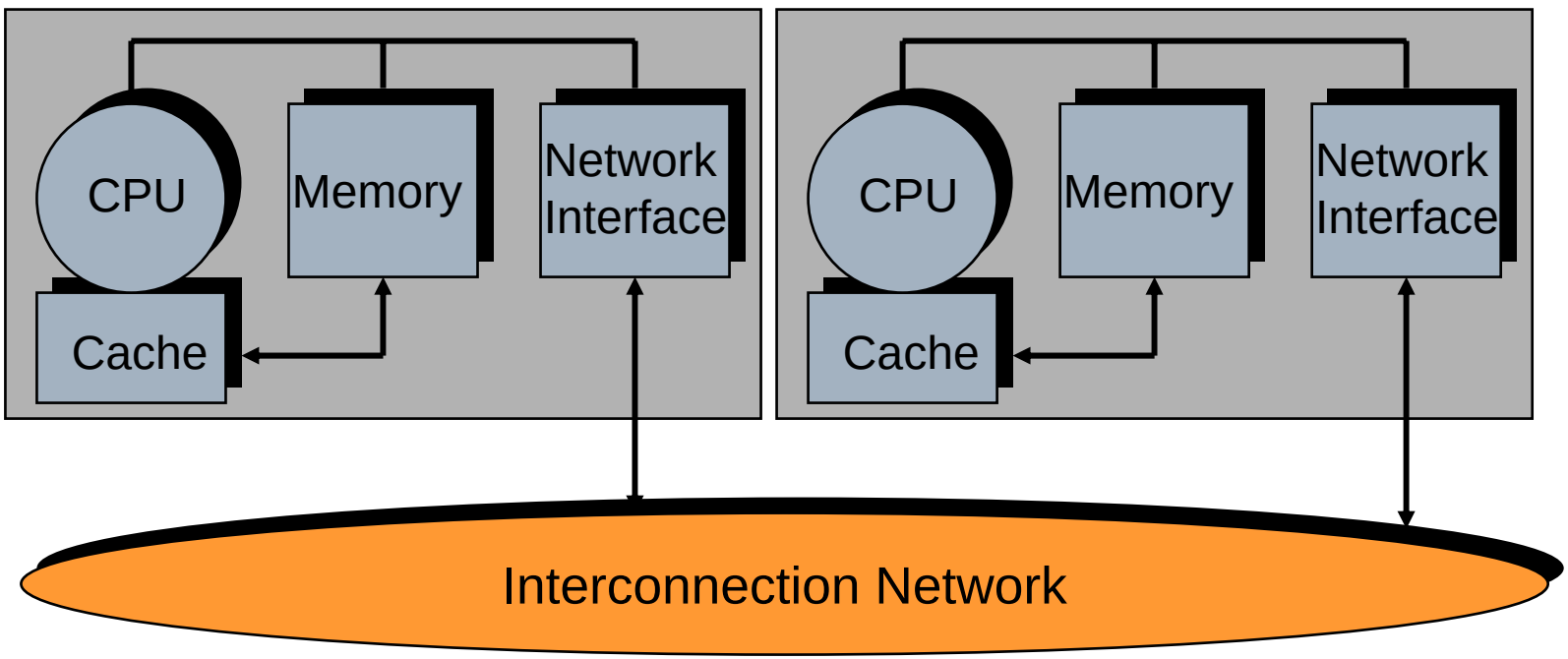


- Parallele Architekturen
  - Programmiermodell
    - Nachrichtenorientiertes Programmiermodell (Message Passing)
      - Kommunikation der Prozesse (Threads) mit Hilfe von Nachrichten
        - » Kein gemeinsamer Adressbereich
      - Kommunikationsarchitektur
        - » Verwendung von korrespondierenden Send- und Receive-Operationen
        - » Send: Spezifikation eines lokalen Datenpuffers und eines Empfangsprozesses (auf einem entfernten Prozessor)
        - » Receive: Spezifikation des Sende-Prozesses und eines lokalen Datenpuffers, in den die Daten ankommen

- Parallele Architekturen
  - Nachrichtenorientiertes Programmiermodell (Message Passing)



- Parallele Architekturen
  - Multiprozessor mit verteiltem Speicher



- Parallele Architekturen
  - Programmiermodell
    - Datenparallelismus
      - Gleichzeitige Ausführung von Operationen auf getrennten Elementen einer Datenmenge (Feld, Matrix)
      - Typischerweise in Vektorprozessoren

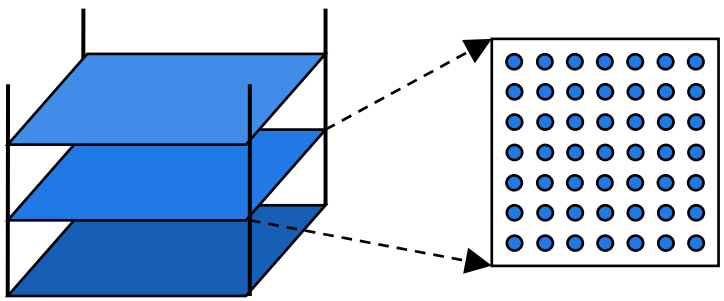
- **Kapitel 3: Multiprozessoren – Parallelismus auf Prozess/Thread-Ebene**

3.3: Parallele Programmierung

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Benchmark-Programm aus der SPLASH-Benchmark-Suite
  - Modellierung des Erdklimas
    - Gegenseitige Beeinflussung der Atmosphäre und der Ozeane, die ¾ der Erdoberfläche ausmachen
  - Simulation der Bewegung der Wasserströmung im Ozean
    - Strömung entwickelt sich unter dem Einfluss mehrerer physikalischer Kräfte, einschließlich atmosphärischer Effekte, dem Wind und der Reibung am Grund des Ozeans;
    - Vertikale Reibung an den „Rändern“: führt zu Wirbelströmung
    - Ziel: Simulation dieser Wirbelströme über der Zeit

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Grundsätzliche Probleme:
    - Gute Modelle für die Beschreibung des Verhaltens sind kompliziert:
      - Vorhersage des Zustands eines Ozeans zu einem Zeitpunkt erfordert die Lösung eines komplexen Gleichungssystems
      - Nur mit numerischen Verfahren möglich
    - Das physikalische Problem ist kontinuierlich über Raum und Zeit:
      - Diskretisierung über beide Dimensionen

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Diskretisierung:
    - Raum:
      - Modellierung des Ozeansbeckens als ein Gitter von diskreten Punkten
      - Jede Variable (Druck, Geschwindigkeit, ...) hat einen Wert an jedem Gitterpunkt
      - Zweidimensionales Gitter:



Modellierung des Ozeans
in einem rechteckigen Becken

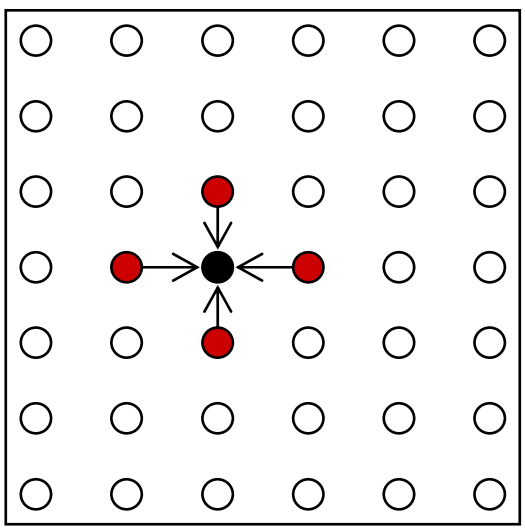
    - Zeit:
      - Endliche Folge von Zeitschritten

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Lösung der Bewegungsgleichungen:
    - An allen Gitterpunkten in einem Zeitschritt
    - In jedem Zeitschritt werden die Variablen neu berechnet
    - Wiederholung der Berechnung mit jedem Zeitschritt
    - Jeder Zeitschritt besteht aus mehreren Phasen

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Lösung der Bewegungsgleichungen:
    - Je mehr Gitterpunkte verwendet werden, desto feiner ist die Auflösung der Diskretisierung und desto genauer ist die Simulation
    - Für einen Ozean wie den Atlantik, der etwa eine Fläche von 2000km x 2000km umspannt bedeutet ein Gitter mit 100 x 100 Punkten eine Distanz von 20 km in jeder Dimension
    - Kürzere physikalische Intervalle zwischen den Zeitschritten führen zu einer höheren Simulationsgenauigkeit
    - Simulation der Ozeanbewegung über einen Zeitraum von 5 Jahren mit einer Aktualisierung des Zustands alle 8 Stunden erfordert 5500 Zeitschritte

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Vereinfachung: Parallelisierung des Gleichungslösers
    - Lösung einer einfachen partiellen Differentialgleichung auf einem Gitter mit Hilfe einer finiten Differenzenmethode (Gauss-Seidel-Verfahren)
    - Reguläres zweidimensionales Gitter mit *(n+2)*(n+2)* Punkten (eine Ebene des Ozeanbeckens
    - Randwerte sind fest
    - Die inneren *n*n* Gitterpunkte werden mit Hilfe des Lösers berechnet, ausgehend von Anfangswerten

- Fallstudie: OCEAN - Simulation der Ozean-Strömung
  - Vereinfachung: Parallelisierung des Gleichungslösers
    - Gitter:



Berechnungsvorschrift für einen Gitterpunkt:

```
A[i,j]=0,2 x (A[i,j]+
             +A[i,j-1]+A[i-1,j]
             +A[i,j+1]+A[i+1,j]
```

Wiederholte Berechnung, bis Verfahren konvergiert

- Sequentielle Version des Lösers:

```
(1) int n,                                  /*size of matrix: (n+2)x(n+2)
(2) float **A, diff=0;
(3) main()
(4) begin
(5)    read(n)                              /*read input parameters*/
(6)    A←malloc (2-d array of size n+2 by n+2 doubles)
(7)    initialize(A);                              /*initialize matrix A*/
(8)    Solve (A);
(9)  end main
```

- Sequentielle Version des Lösers:

```
(1) procedure Solve (A)                  /*solve equation system*/
(2)   float **A;
(3) begin
(4)   int i,j,done=0
(5)   float temp;
(6)   while (!done) do                   /*outermost loop over sweeps*/
(7)     diff=0;                          /*initialize maximum diff*/
(8)     for i←1 to n do                  /*sweep over nonborder points*/
(9)       for j←1 to n do
(10)        temp=A[i,j];
(11)        A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(12)        diff += abs(A[i,j]–temp);
(13)      end for
(14)    end for
(15)    if (diff/(n*n) < TOL) then done=1;
(16)  end while
(17) end procedure
```

- Parallelisierungsprozess
  - Festlegen der Aufgaben, die parallel ausgeführt werden kann
    - Aufteilen der Aufgaben und der Daten auf Verarbeitungsknoten
      - Berechnung
      - Datenzugriff
      - Ein-/Ausgabe
    - Verwalten des Datenzugriffs, der Kommunikation und der Synchronisation
  - Ziel: Hohe Leistung
    - Schnellere Lösung der parallelen Version gegenüber der sequentiellen Version
      - Ausgewogene Verteilung der Arbeit unter den Verarbeitungsknoten
      - Reduzierung des Kommunikations- und Synchronisationsaufwandes

- Parallelisierungsprozess
  - Ausführung der Schritte bei der Parallelisierung
    - Durch den Programmierer
    - Auf den verschiedenen Ebenen der Systemsoftware
      - Compiler
      - Laufzeitsystem
      - Betriebssystem
    - Ideal:
      - Automatische Parallelisierung
      - Sequentielles Programm wird automatisch in ein effizientes paralleles Programm transformiert
      - Parallelisierende Compiler
      - Parallele Programmiersprachen
      - Noch nicht vollständig möglich!

- Parallelisierungsprozess
  - Definitionen
    - Task
      - Beliebige Aufgabe, die durch ein Programm auszuführen ist
      - Kleinste Parallelisierungseinheit
      - Möglichkeiten beim Beispiel Ocean:
        - » Berechnung eines Gitterpunkts in jeder Berechnungsphase,
        - » die Berechnung einer Reihe von Gitterpunkten,
        - » die Berechnung einer beliebigen Teilmenge von Gitterpunkten
      - Granularität
        - » grobkörnig
        - » feinkörnig

- Parallelisierungsprozess
  - Definitionen
    - Prozess oder Thread
      - Paralleles Programm setzt sich aus mehreren kooperierenden Prozessen zusammen, von denen jeder eine Teilmenge der Tasks ausführt
      - Tasks werden über Prozessen zugewiesen
      - Beispiel Ocean:
        - » Falls die Berechnung einer Reihe von Gitterpunkten als Task angesehen wird, dann kann eine feste Anzahl von Reihen einem Prozess zugewiesen werden
        - » Aufteilung einer Ebene in mehrere Streifen
      - Kommunikation der Prozesse untereinander und Synchronisation
    - Prozessor
      - Ausführung eines Prozesses

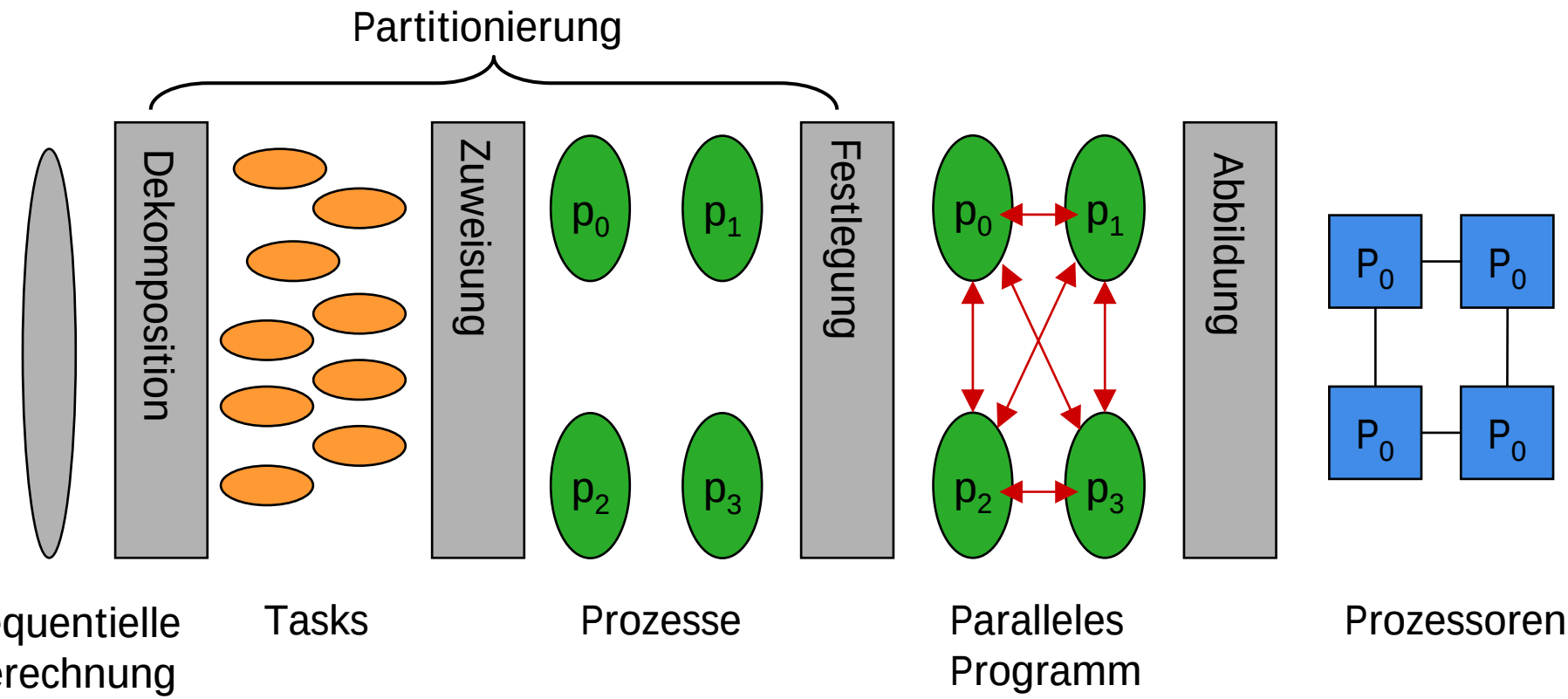
- Parallelisierungsprozess
  - Definitionen
    - Unterscheidung Prozess und Prozessor
      - Prozessor:
        - » Physikalische Ressource
      - Prozess
        - » Abstraktion, Virtualisierung von einem Multiprozessor
        - » Anzahl der Prozesse muss nicht gleich der Anzahl der Prozessoren eines Multiprozessorsystems sein

- Parallelisierungsprozess
  - Schritte bei der Parallelisierung
    - Ausgangspunkt ist ein sequentielles Programm
    - Aufteilung oder Dekomposition
      - der Berechnung in Tasks
    - Zuweisung
      - der Tasks zu Prozessen

    } Partitionierung

    - Festlegung (Orchestration)
      - des notwendigen Datenzugriffs, der Kommunikation und der Synchronisation zwischen den Prozessen
    - Abbildung
      - der Prozesse an die Prozessoren

- Parallelisierungsprozess
  - Schritte bei der Parallelisierung und die Beziehung zwischen Tasks, Prozessen und Prozessoren



- Parallelisierungsprozess
  - Dekomposition
    - Aufteilung der Berechnung in eine Menge von Tasks
      - Tasks können dynamisch während der Ausführung generiert werden
      - Anzahl der Tasks kann während der Ausführung variieren
      - Maximale Anzahl der Tasks, die zu einem Zeitpunkt zur Ausführung verfügbar sind, ist eine obere Grenze für die Anzahl der Prozesse, die effektiv genutzt werden können
    - Ziel:
      - Finden von parallel ausführbaren Anteilen
      - Verwaltungsaufwand (Overhead) gering halten

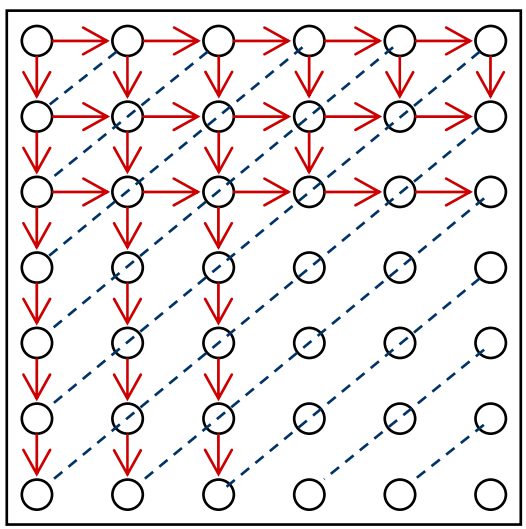
- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - Programm strukturiert in geschachtelten Schleifen
        - » Betrachtung einzelner Schleifen oder der geschachtelten Schleifen
        - » Können Iterationen parallel ausgeführt werden?
        - » Betrachtung über Schleifengrenzen

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - Betrachtung einzelner Schleifen oder der geschachtelten Schleifen

```
(1)  while (!done) do
(2)      diff=0;
(3)      for i←1 to n do
(4)         for j←1 to n do
(5)            temp=A[i,j];
(6)            A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(7)            diff += abs(A[i,j]–temp);
(8)         end for
(9)       end for
(10)      if (diff/(n*n) < TOL) then done=1;
(11)   end while
```

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - Betrachtung einzelner Schleifen oder der geschachtelten Schleifen
        - » Äußere Schleife (Zeile 1-11) durchläuft das gesamte Gitter → Iterationen sind nicht unabhängig, da Daten, die in einer Iteration geändert werden, in der nächsten Iteration gebraucht werden
        - » Innere Schleifen (Zeile 3-9)→Iterationen sind sequentiell abhängig, da in jeder inneren Schleife A[i,j-1] gelesen wird, der in der vorherigen Iteration geschrieben wurde

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - Betrachtung der Abhängigkeiten (Granularität Gitterpunkte)

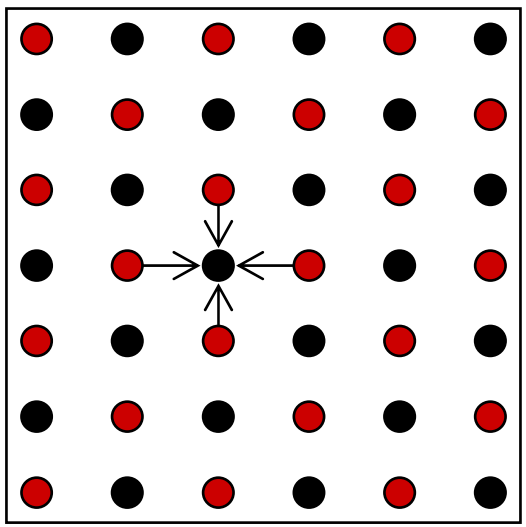


Abhängigkeiten

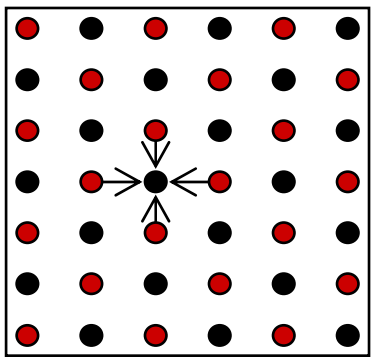
Verbinden Punkte, zwischen den keine Abhängigkeiten bestehen

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - 1. Möglichkeit:
        - » Aufteilen der Arbeit in einzelne Gitterpunkte, so dass die Aktualisierung eines Gitterpunktes eine Task ist
        - » Beibehalten der Schleifenstruktur
        - » erfordert Punkt-zu-Punkt-Synchronisation wegen Beachtung der Abhängigkeiten
        - » Hoher Aufwand!

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - 3. Möglichkeit:
        - » Änderung der Durchlaufordnung: Red-Black



- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - 3. Möglichkeit: Red-Black



» Durchlauf über ein Gitter wird in zwei Phasen aufgeteilt:
» Parallele Berechnung der $n^2/2$ roten Punkte
» Globale Synchronisation (konservativ)
» Berechnung der $n^2/2$ schwarzen Punkte
» Konvergiert mit mehr oder mit weniger Durchläufen
» Es können auch unterschiedliche Werte innerhalb der Toleranz berechnet werden

- Parallelisierungsprozess
  - Dekomposition
    - Beispiel: Ocean
      - 4. Möglichkeit: Asynchrone Methode ohne Aufteilung in rote und schwarze Punkte
        - » Ignorieren der Abhängigkeiten zwischen den Gitterpunkten für einen Durchlauf
        - » Globale Synchronisation zwischen den Iterationen, aber keine Änderung der Durchlaufordnung
        - » Prozess aktualisiert alle Punkte, sequentielle Ordnung
        - » Punkte können auf mehrere Prozesse aufgeteilt werden, dann ist die Ordnung nicht vorhersagbar, sondern hängt von der Zuteilung der Punkte zu Prozessen, der Anzahl der Prozesse und wie schnell die verschiedenen Prozesse relativ zueinander während der Laufzeit ausgeführt werden, ab
        - » Ausführung ist nicht deterministisch!
        - » Anzahl der Durchläufe bis zur Konvergenz kann von der Anzahl der Prozesse abhängen

- Parallelisierungsprozess
  - Zuweisung
    - Spezifikation des Mechanismus, mit dessen Hilfe die Tasks auf Prozesse aufgeteilt werden
    - Ziel:
      - ausgewogene Lastverteilung: Lastbalanzierung
      - Reduzierung der Interprozess-Kommunikation
      - Reduzierung des Aufwands zur Laufzeit
    - Statische oder dynamische Zuweisung

- Parallelisierungsprozess
  - Festlegung
    - Architektur und Programmiermodell sowie die Programmiersprache spielen eine Rolle
      - Um die zugewiesenen Tasks ausführen zu können, benötigen die Prozesse Mechanismen
        - » für den Zugriff auf die Daten,
        - » für die Kommunikation (Austausch von Daten)
        - » für die Synchronisation untereinander
      - Fragen
        - » Organisation der Datenstrukturen
        - » Ablauf der Tasks
        - » Explizite oder implizite Kommunikation
    - Ziel:
      - Reduzierung des Kommunikations- und Synchronisationsaufwandes (aus der Sicht des Prozessors)
      - Erhalten der Lokalität der Datenzugriffe, soweit möglich
      - Reduzierung des Parallelisierungsaufwandes
      - Rechnerarchitekt: bereitstellen effizienter Mechanismen, die die Festlegung vereinfachen

- Parallelisierungsprozess
  - Festlegung
    - Programmiermodelle:
      - Shared-Memory-Programmiermodell
      - Nachrichten-orientiertes Programmiermodell
      - Datenparalleles Programmiermodell

- Parallelisierungsprozess
  - Festlegung
    - Shared-Memory Programmiermodell
      - Primitive:

Name	Syntax	Funktion
CREATE	**CREATE(p,proc,args)**	Generiere Prozess, der die Ausführung bei der Prozedur **proc** mit den Argumenten **args** startet
G_MALLOC	**G_MALLOC(size)**	Allokation eines gemeinsamen Datenbereichs der Größe **size** Bytes
LOCK	**LOCK(name)**	Fordere wechselseitigen exklusiven Zugriff an
UNLOCK	**UNLOCK(name)**	Freigeben des Locks

- Parallelisierungsprozess
  - Festlegung
    - Shared-Memory Programmiermodell
      - Primitive:

Name	Syntax	Funktion
BARRIER	**BARRIER(name,number)**	Globale Synchronisation für **number** Prozesse
WAIT_FOR_END	**WAIT_FOR_END(number)**	Warten, bis **number** Prozesse terminieren
wait for flag	**while (!flag); or WAIT(flag)**	Warte auf gesetztes flag; entweder wiederholte Abfrage (spin) oder blockiere;
set flag	**flag=1; or SIGNAL(flag)**	Setze flag; weckt Prozess auf, der **flag** wiederholt abfragt

- Parallelisierungsprozess
  - Festlegung
    - Message Passing
      - Primitive:

Name	Syntax	Funktion
CREATE	**CREATE(procedure)**	Erzeuge Prozess, der bei **procedure** startet
SEND	**SEND(src_addr,size, dest,tag)**	Sende **size** Bytes von Adresse **src_addr** an **dest** Prozess mit **tag** Identifier
RECEIVE	**RECEIVE(buffer_addr ,size,src,tag)**	Empfange eine Nachricht mit der Kennung **tag** vom **src**-Prozess und lege **size** Bytes in Puffer bei **buffer_addr** ab
BARRIER	**BARRIER(name,number)**	Globale Synchronisation von **number** Prozessen