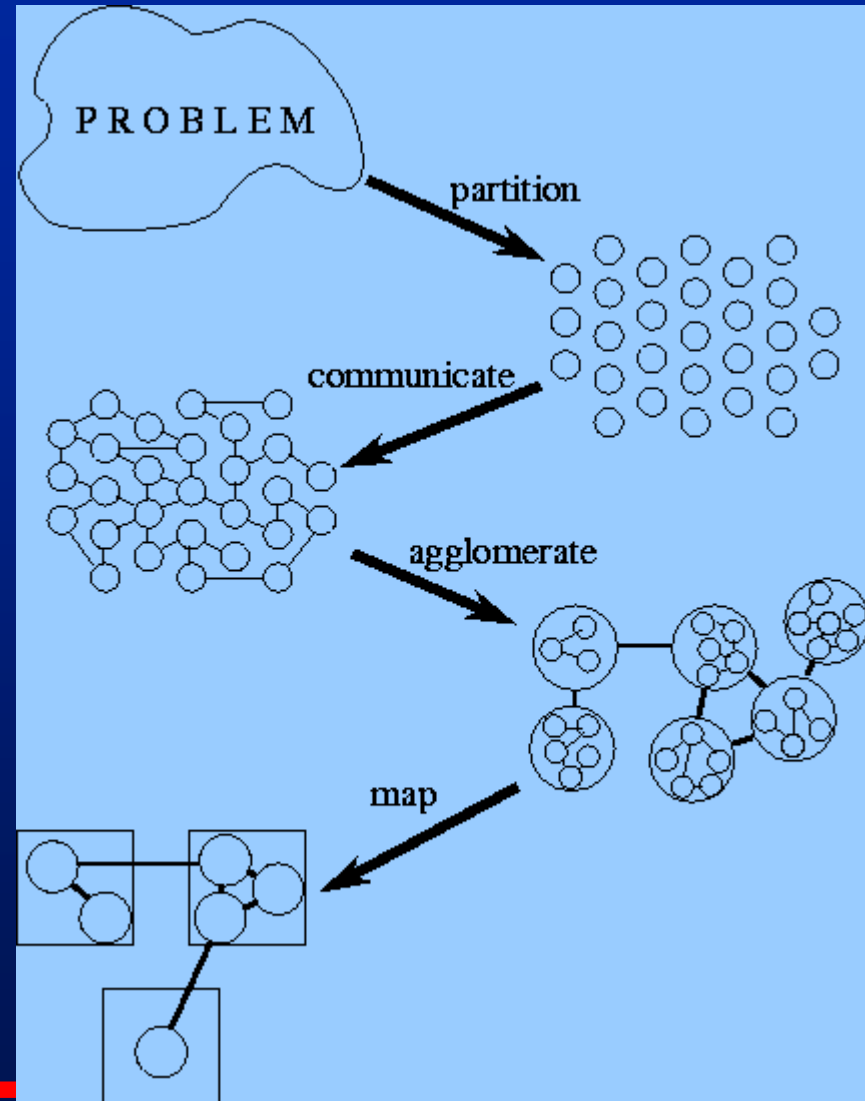


Paralleler Programmmentwurf nach Foster

Die PCAM-Methode

- **P**artitionierung
 - ermittle maximale Parallelität
- **C**ommunication
 - ermittle Datenabhängigkeiten
- **A**gglomeration
 - erhöhe die Granularität der Aufgaben
- **M**apping
 - Prozessorplatzierung



Partitionierung

Ziele:

- möglichst feinkörnige problemabhängige
Zerlegung der Berechnung und der Daten in Teile
(tasks)
ohne Berücksichtigung der zur Verfügung
stehenden Prozessoren
=> inhärente Parallelität, Skalierbarkeit
- Bestimmung der maximal vorhandenen Parallelität
- Vermeidung der Duplizierung von
Daten/Berechnungen

Beispiel: Matrixmultiplikation

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \cdot & \cdot & \cdot & \cdot \\ \mathbf{a_{i1}} & \mathbf{a_{i2}} & \dots & \mathbf{a_{in}} \\ \cdot & \cdot & \cdot & \cdot \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} & \dots & \mathbf{b_{1k}} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & \mathbf{b_{2k}} & \dots & b_{2n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ b_{i1} & b_{i2} & \dots & \mathbf{b_{ik}} & \dots & b_{in} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ b_{n1} & b_{n2} & \dots & \mathbf{b_{nk}} & \dots & b_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} c_{11} & \dots & c_{1k} & \dots & c_{1n} \\ c_{21} & \dots & c_{2k} & \dots & c_{2n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ c_{i1} & \dots & \mathbf{c_{ik}} & \dots & c_{in} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ c_{n1} & \dots & c_{nk} & \dots & c_{nn} \end{pmatrix} \quad \text{mit } \mathbf{c_{ik}} = \sum_{j=1}^n a_{ij} * b_{jk}$$

Partitionierung

- Grundtechniken
 - Bereichszerlegung → Datenparallelität
 - funktionale Zerlegung → Kontrollparallelität
- im Beispiel Matrixmultiplikation:
 - Bereichszerlegung:
 - Ausgabematrix → n^2 Aufgaben
 - funktionale Zerlegung:
 - n^3 Multiplikationen, $n^2 * (n-1)$ Additionen

Checkliste Partitionierung

- # Tasks \gg # Prozessoren? \Rightarrow Flexibilität
- keine redundanten Berechnungen,
Speicheranforderungen? \Rightarrow Skalierbarkeit
- vergleichbare Taskgröße? \Rightarrow Lastausgleich
- Steigt die Anzahl der Tasks mit der Problemgröße?
(nicht die Größe der Tasks!) \Rightarrow Skalierbarkeit
- alternative Partitionierungen? \Rightarrow Flexibilität

Kommunikation

Ziele:

- Identifikation der Kanalstruktur, die erforderlich ist, um Tasks mit den von ihnen benötigten Daten zu versorgen
- Identifikation der zu transportierenden Daten

Kommunikationsmuster:

- lokal vs global
- strukturiert vs unstrukturiert
- statisch vs dynamisch
- synchron vs asynchron

Beispiel Matrixmultiplikation

Bestimmung notwendiger Kommunikationen

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & \dots & a_{in} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} * \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2k} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{i1} & b_{i2} & \dots & b_{ik} & \dots & b_{in} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nk} & \dots & b_{nn} \end{pmatrix}$$

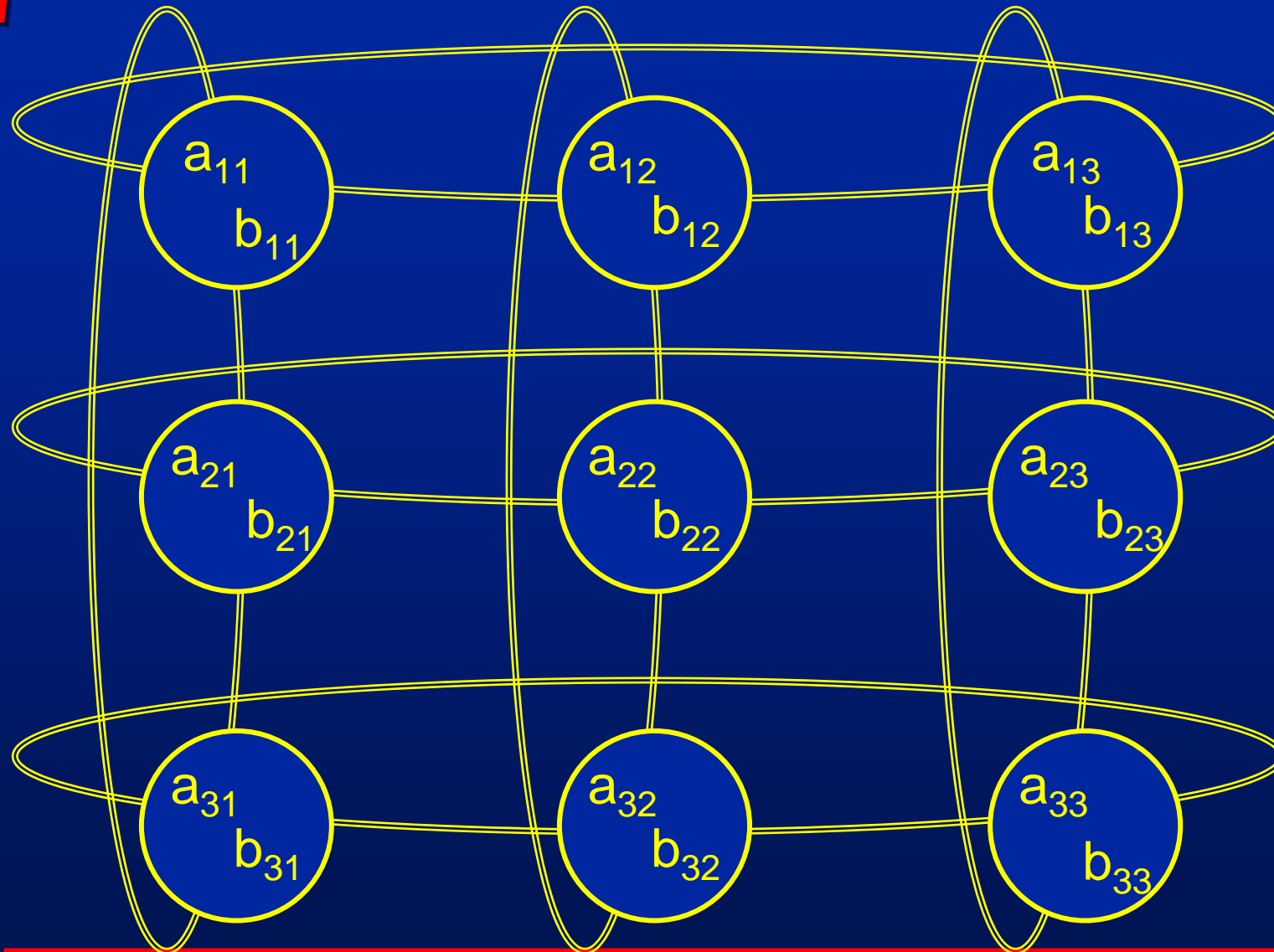
Bereichszerlegung

$$= \begin{pmatrix} c_{11} & \dots & c_{1k} & \dots & c_{1n} \\ c_{21} & \dots & c_{2k} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{i1} & \dots & c_{ik} & \dots & c_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & \dots & c_{nk} & \dots & c_{nn} \end{pmatrix}$$

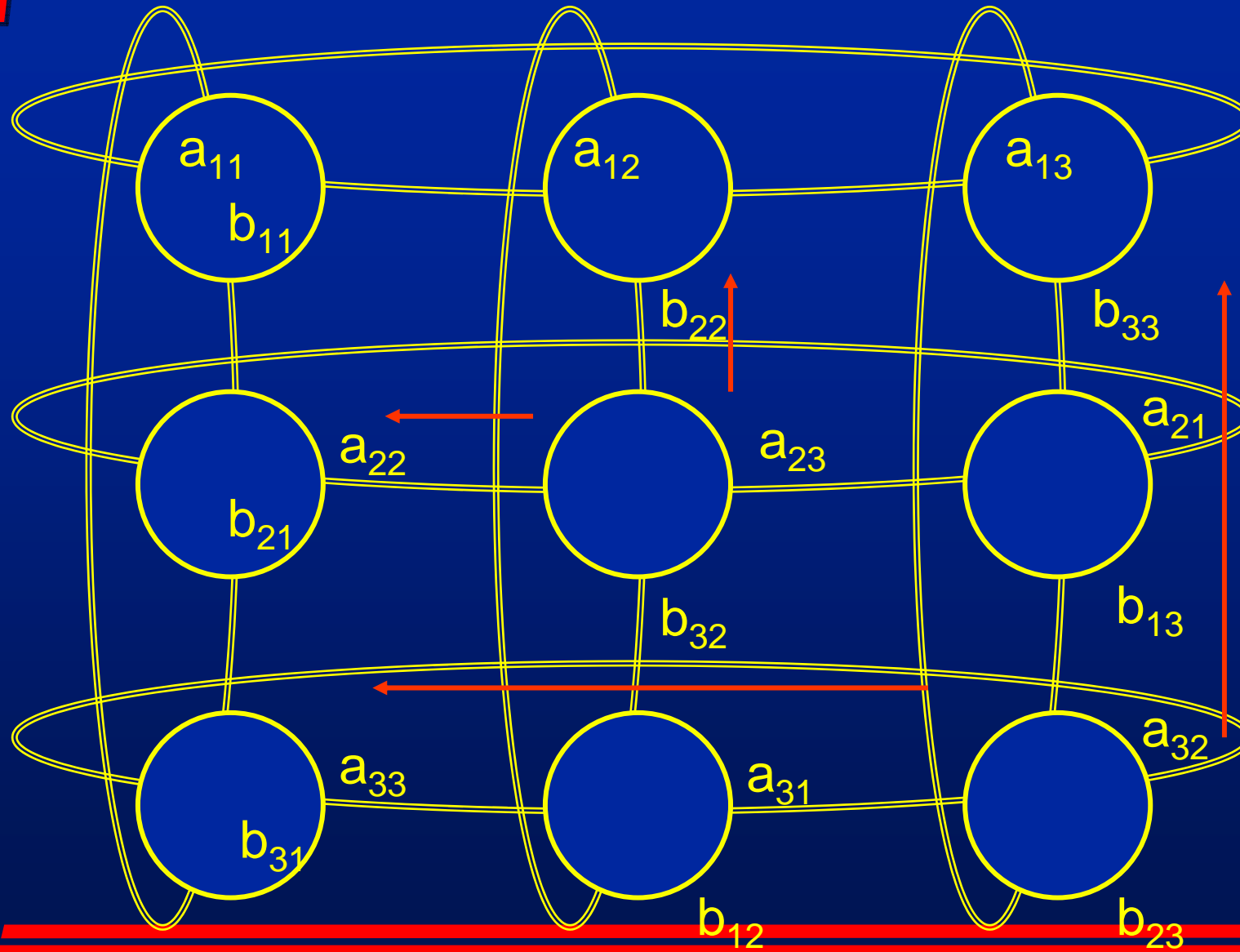
mit $c_{ik} = \sum_{j=1}^n a_{ij} * b_{jk}$

Problem: Datenreplikation

Lösung: Rotation in Torusstruktur

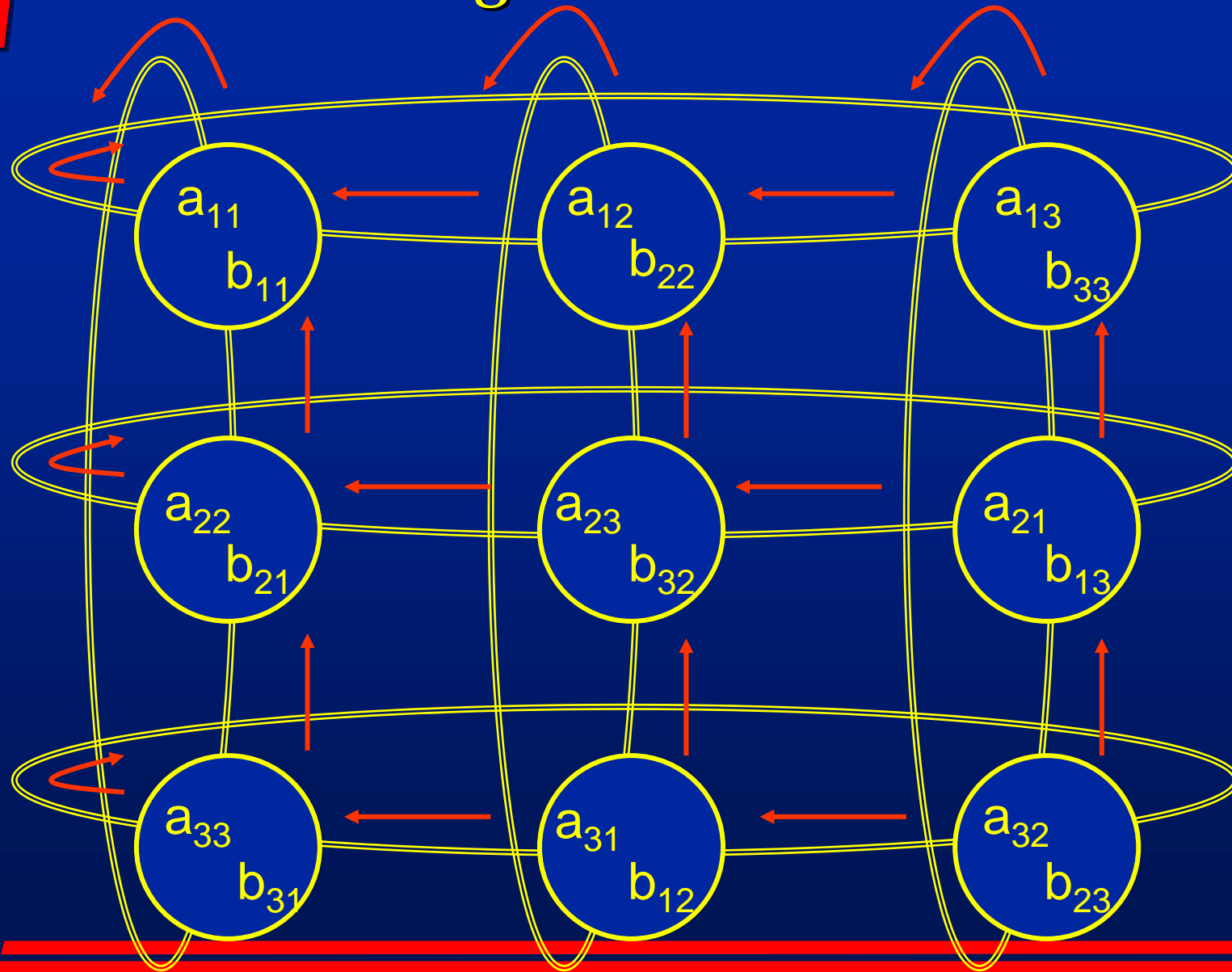


Vorrotation



Rotiere i-te Zeile von Matrix A um i-1 Positionen nach links
Rotiere j-te Spalte von Matrix B um j-1 Positionen nach oben

Gentleman-Algorithmus



Gentleman-Algorithmus

```
Nach Vorrotation:  
forall P(i,j) (1 <= i,j <= n) do  
  c := 0;  
  for i := 1 to n do  
    begin  
      sende a zu linkem Nachbarn  
      sende b zu oberem Nachbarn  
      c := c + a*b  
      empfangen a von rechtem Nachbarn  
      empfangen b von unterem Nachbarn  
    end
```

b_{31}

b_{12}

b_{23}

Checkliste Kommunikation

- # Kommunikationen in allen Tasks gleich?
=> Skalierbarkeit, Balance
- möglichst lokale Kommunikationen
=> Effizienz
- Kommunikationen nebenläufig?
- Kommunikation nebenläufig zu Berechnung?

Analyse des Gentleman-Algorithmus

- ursprünglich SIMD-Verfahren
- Tasks gleich komplex
- Anfangsverteilung kann durch geeignete Initialisierung hergestellt werden
- Kommunikation:
 - nur mit Nachbarn => lokal
 - strukturiert und statisch
 - nebenläufig in Zeilen und Spalten
 - überlappend mit Berechnung

Agglomeration

Ziele:

- Minimierung der Kommunikationskosten durch Zusammenfassung von stark interagierenden Teilaufgaben
- Vergrößerung der Aufgaben
- Verbesserung der Skalierbarkeit

Methoden:

- Replikation von Berechnungen
- Überlappung von Kommunikation und Berechnung

im Beispiel:

- Submatrizen (Teilblöcke) statt einzelner Matrixelemente multiplizieren und rotieren
- Verhältnis Kommunikationsaufwand/Berechnungsaufwand sinkt
=> gute Skalierbarkeit

Der Oberfläche-Volumen-Effekt

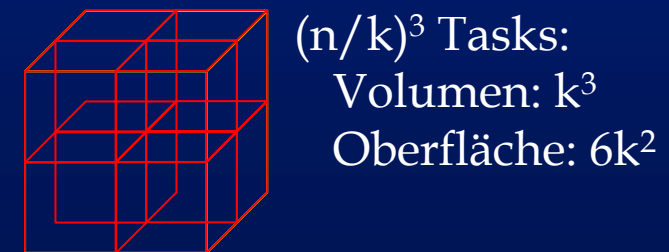
Bei regulären mehrdimensionalen Strukturen, wie Gittern, Würfeln etc. ist eine Agglomeration auf mehrere Weisen möglich:

- Dimensionsreduktion



- Blockaufteilung

Im dreidimensionalen Fall skalieren bei der Blockaufteilung das Volumen (= Berechnungsaufwand) kubisch und



die Oberfläche (= Kommunikationsaufwand) quadratisch.



Checkliste Agglomeration

- Reduktion der Kommunikationskosten durch Erhöhung der Lokalität?
- Mehraufwand durch Replikation von Daten/ Berechnungen gerechtfertigt?
- Skalierbarkeit?
- Verhältnis Kommunikations-/ Berechnungsaufwand?
- Task-Komplexität ausgeglichen?
- weitere Zusammenfassungen?

Mapping

Ziele:

- Zuordnung der parallelen Aufgaben zu Prozessoren (rechnerabhängig)
- Platzierung unabh. Tasks auf versch. Rechnern / Platzierung häufig komm. Tasks auf denselben Proz.

Methoden:

- statische Aufgabenverteilung vs dynamische Lastbalancierung (task scheduling)
=> Master-Worker-Strukturen

Checkliste:

- alle Alternativen berücksichtigt?
- Implementierungskosten?



Zusammenfassung

- **PCAM-Methode** erlaubt systematische parallele Programmentwicklung.
- Beispielalgorithmus: **Matrixmultiplikation** in einer Torusstruktur nach Gentleman