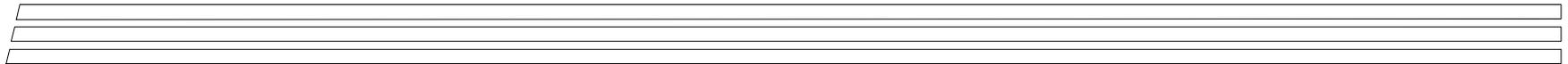
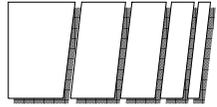


Parallele Algorithmen

- bereits behandelt:
 - paralleles Sortieren mit Ranksort
 - parallele Matrixmultiplikation nach Gentleman
 - numerisches Iterationsverfahren nach Jacobi
- Matrixmultiplikation im Hypercube (DNS-Verfahren)
- Paralleles Sortieren
 - Bitonisches Mischsortieren
 - Hyper-Quicksort
 - PSRS-Verfahren (Parallel Sorting by Regular Sampling)
- Dynamische Aufgabenverwaltung
- Terminationserkennung in verteilten Systemen



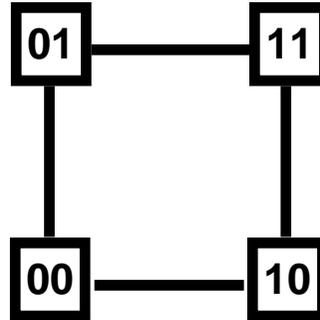


Aufbau von Hypercubes

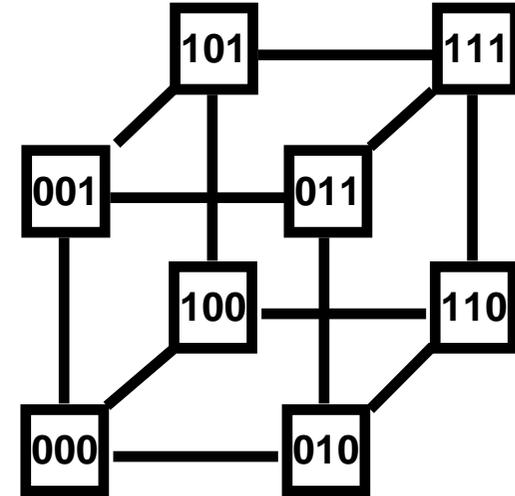
k=1



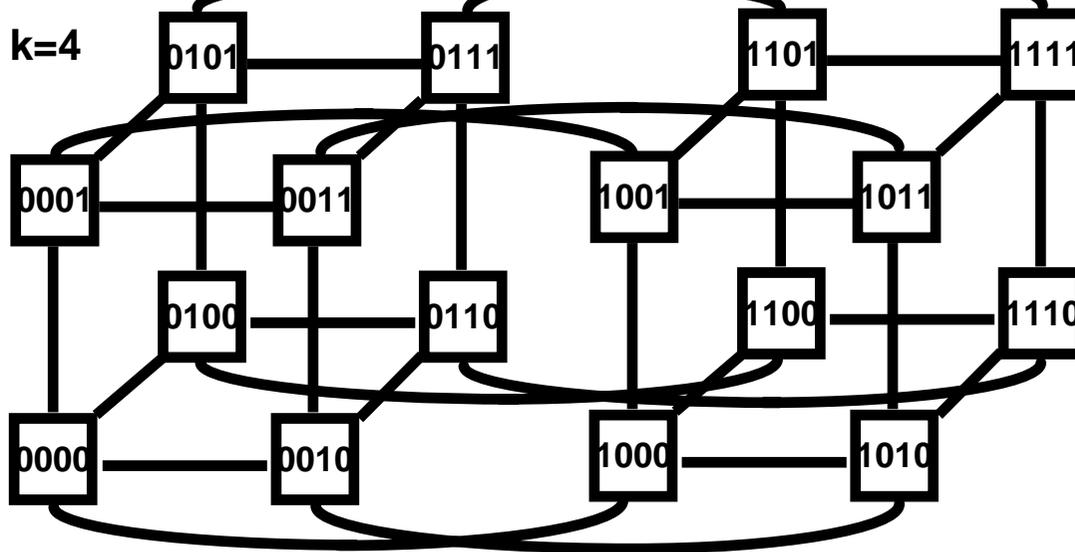
k=2



k=3

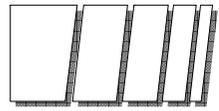


k=4



allgemein:
Hypercube der
Dimension k mit
 2^k Knoten und
Verbindungsgrad k

Ein Hypercube der Dim. k erlaubt Broadcast- und Reduktionsop. in k Schritten. 154



Matrixmultiplikation im Hypercube

[Dekel, Nassimi, Sahni -SIAM Journal on Computing 1981]

Grundidee:

Führe alle n^3 Multiplikationen in einem parallelen Schritt durch.

Algorithmus:

Gegeben Matrizen (a_{ij}) , (b_{ij}) mit $0 \leq i, j \leq n-1$. Sei $n = 2^q$.

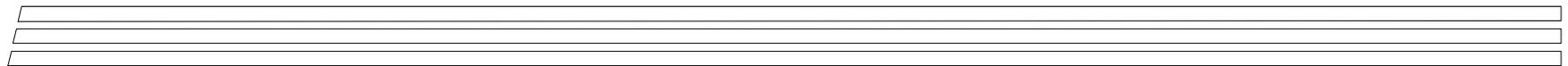
Identifikation der Prozesse im Hypercube der Dimension $3q$ durch

$$\text{id} = (b_{3q-1} \dots b_0)_2$$

Prozess $P(\text{id})$ habe lokale Variablen a , b , c .

Der Algorithmus hat 3 Phasen:

- **Laden und Verteilen der Matrixelemente im Hypercube**
- **parallele Multiplikation in allen PE's**
- **Akkumulation und Summation der Produkte**



Phase I: Broadcast der Eingabematrizen

- Laden der $n^2 = 2^{2q}$ Matrixelemente in Teilhypercube der Dim. $2q$:

$$a_{ij}, b_{ij} \rightarrow P(2^q i+j) \quad \text{Form der Pid:} \quad \frac{0 \dots 0}{q \text{ Bits}} \quad \frac{i}{q \text{ Bits}} \quad \frac{j}{q \text{ Bits}}$$

- Broadcast der Elemente in $2^q - 1 = n - 1$ weitere Teilhypercubes der Dimension 2^q :

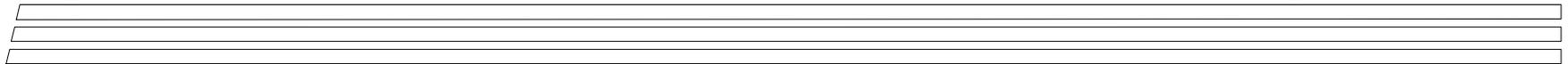
$$a_{ij}, b_{ij} \rightarrow P(2^{2q} k + 2^q i + j) \text{ f\u00fcr alle } 1 \leq k \leq n - 1$$

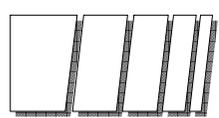
- Umspeicherung der Matrixelemente, so dass in jedem Prozess genau ein Produkt berechnet werden kann:

- $a_{i1} \rightarrow P(2^{2q} 1 + 2^q i + j)$ f\u00fcr $0 \leq j \leq n - 1$, $b_{1j} \rightarrow P(2^{2q} 1 + 2^q i + j)$ f\u00fcr $0 \leq i \leq n - 1$
- Broadcast von a_{i1} [aus $P(2^{2q} 1 + 2^q i + 1)$] in Dimensionen $0 \dots q - 1$
- Broadcast von b_{1j} [aus $P(2^{2q} 1 + 2^q 1 + j)$] in Dimensionen $q \dots 2q - 1$

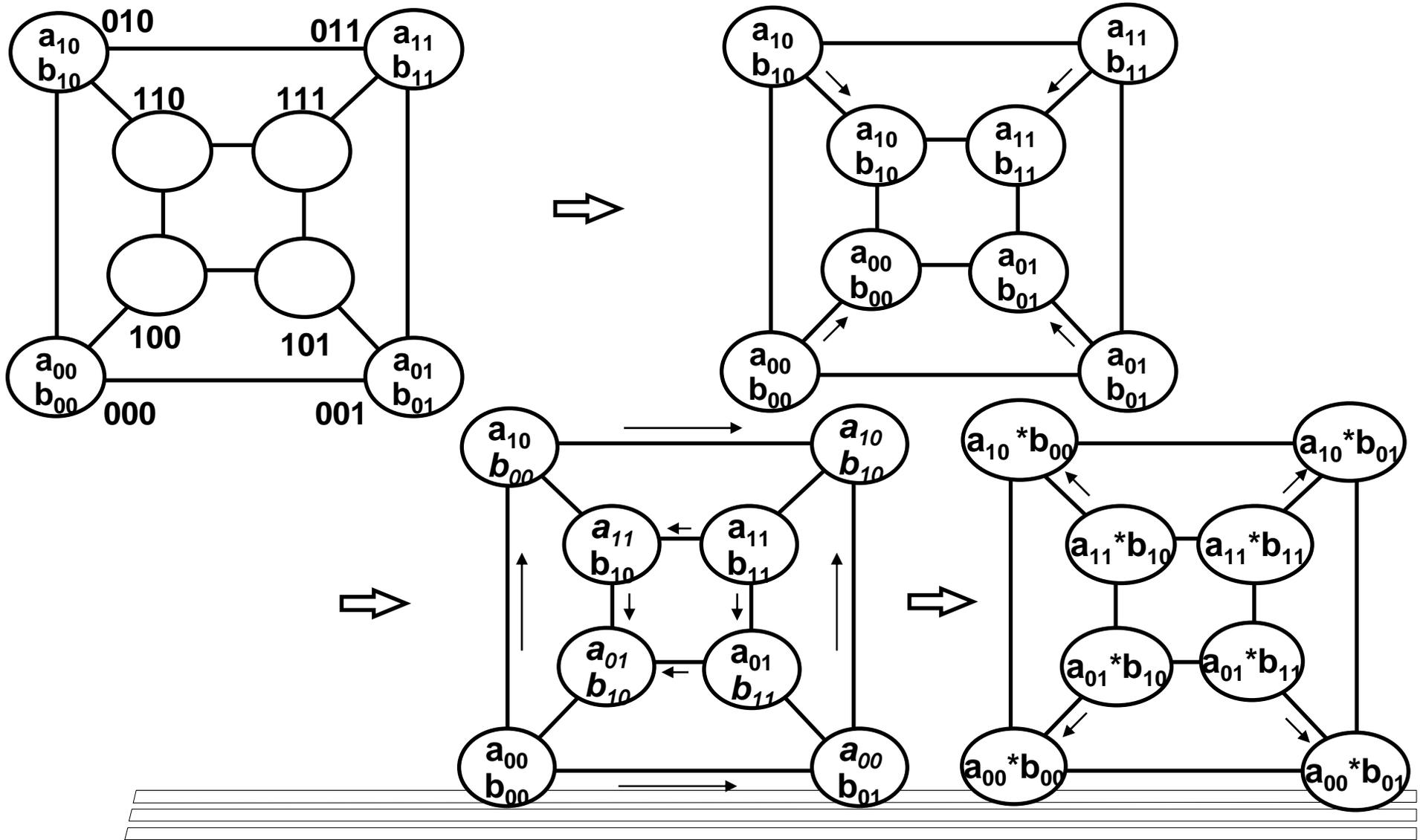
in Phase III:

“umgekehrter” Broadcast in oberen q Dimensionsverbindungen





Beispiel: $\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} * \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} a_{00}b_{00} + a_{01}b_{10} & a_{00}b_{01} + a_{01}b_{11} \\ a_{10}b_{00} + a_{11}b_{10} & a_{10}b_{01} + a_{11}b_{11} \end{pmatrix}$



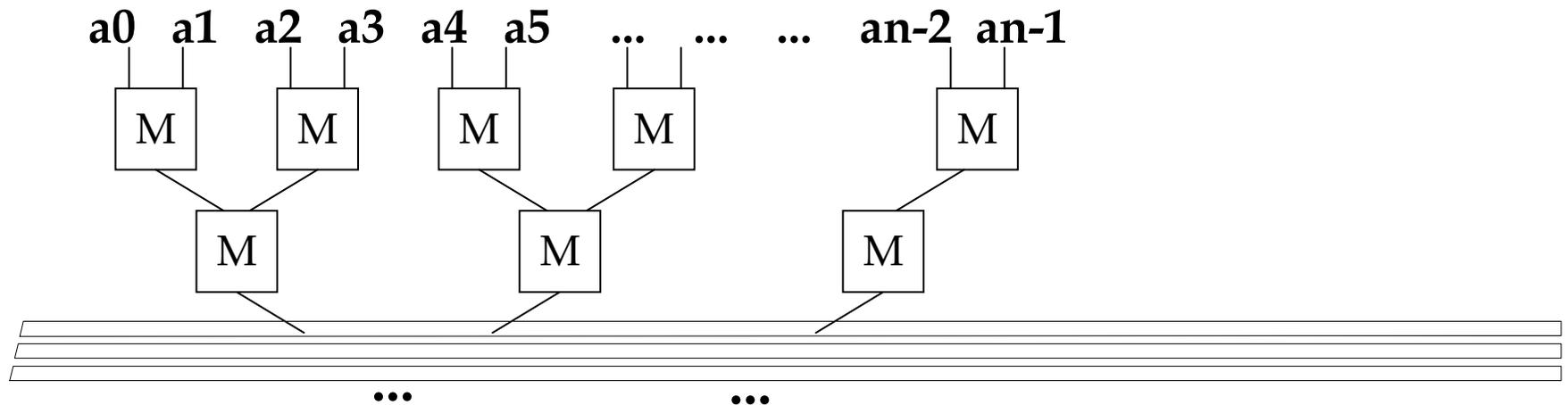
Paralleles Sortieren

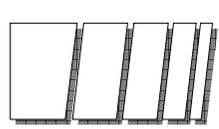
- **RankSort: $O(n^2)$ Vergleiche auf n Prozessoren**
 => $O(n)$ parallele Schritte (ohne Kommunikation)
- **paralleles BubbleSort: Odd-Even-Transposition-Sort**



=> $O(n)$ parallele Schritte (inkl. Kommunikation)

- **Mischsortieren (seq. Aufwand $O(n \log n)$, par. Aufwand: $O(n)$)**

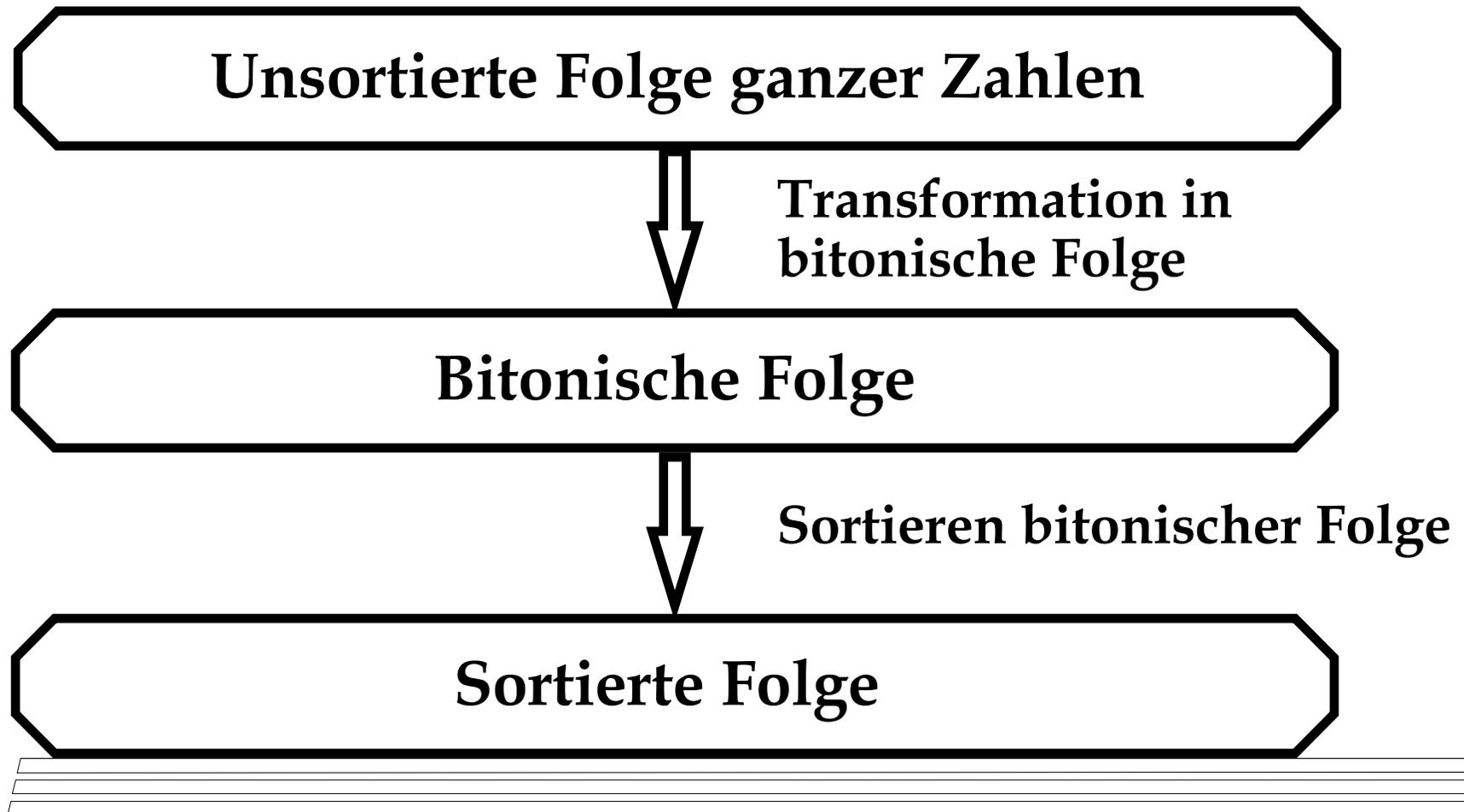


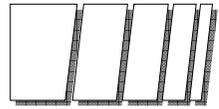


Bitonisches Mischsortieren

(Batcher 1968)

paralleles Sortierverfahren mit Komplexität $O(\log^2 n)$





Bitonische Folgen

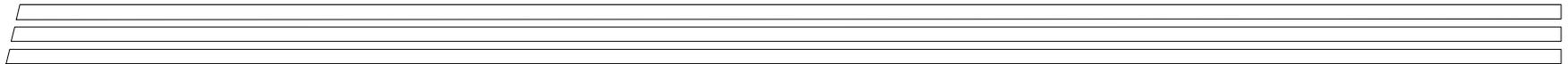
Eine Folge ganzer Zahlen a_0, \dots, a_{n-1} heißt bitonisch, falls

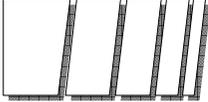
(1) ein Index i existiert, so daß

$$a_0 \leq \dots \leq a_{i-1} \leq a_i \geq a_{i+1} \geq \dots \geq a_{n-1}$$

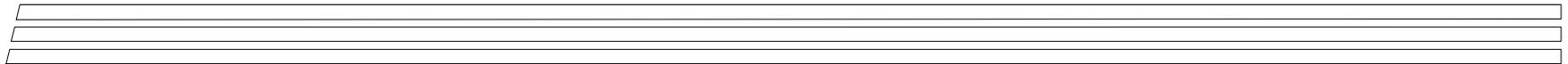
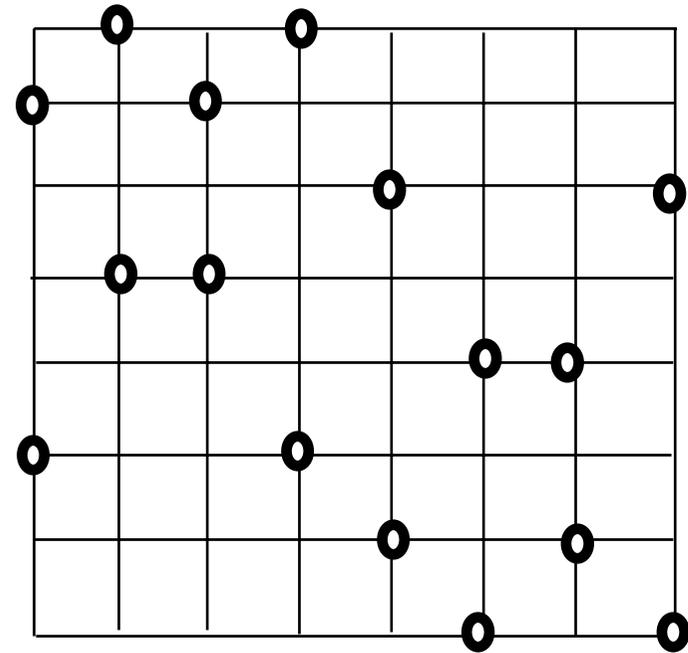
oder

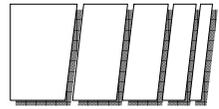
(2) Bedingung (1) durch eine zyklische Verschiebung der Folgenindizes erfüllt werden kann.



 *Beispielfolgen*

- **3, 5, 7, 8, 6, 4, 2, 1**
- **7, 8, 5, 3, 1, 2, 4, 6**





Zerlegen bitonischer Folgen

Lemma: Seien $n=2^k$ mit $k>0$ und (a_0, \dots, a_{n-1}) eine bitonische Folge mit

$$a_0 \leq a_1 \leq a_2 \leq \dots \leq a_{n/2} \quad \text{und} \quad a_{n/2} \geq a_{n/2+1} \geq \dots \geq a_{n-1}$$

Dann sind die Folgen

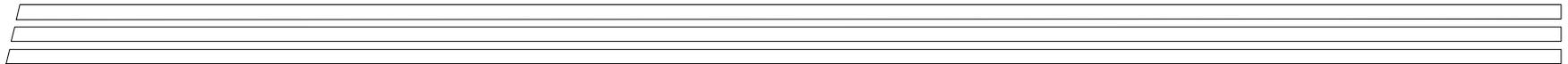
$$\min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}) \quad \dots \quad \min(a_{n/2-1}, a_{n-1})$$

und

$$\max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}) \quad \dots \quad \max(a_{n/2-1}, a_{n-1})$$

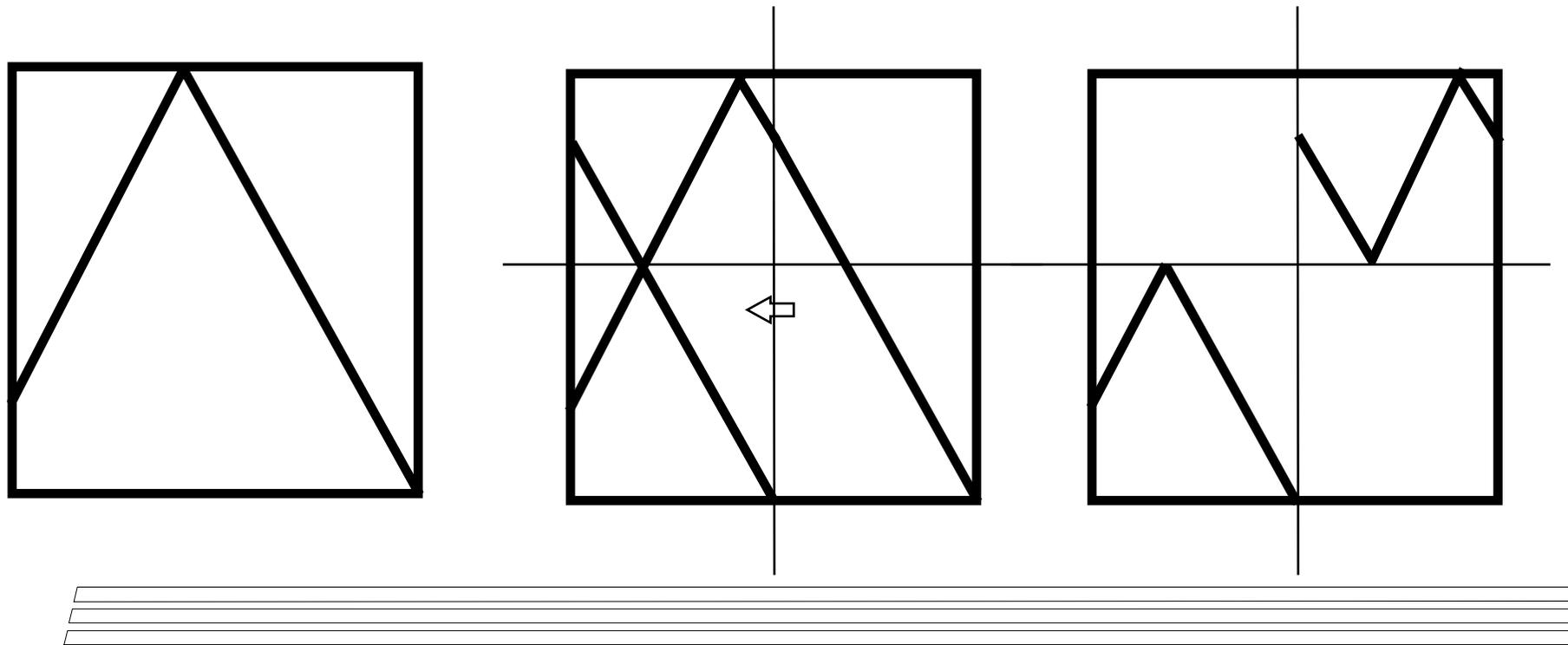
bitonisch und jedes Element der Minimumfolge ist kleiner oder gleich zu jedem Element der Maximumfolge.

Bem: Diese Aussage gilt für beliebige bitonische Listen, wird aber in der Vorlesung zur Vereinfachung nur für obigen Spezialfall formal bewiesen.

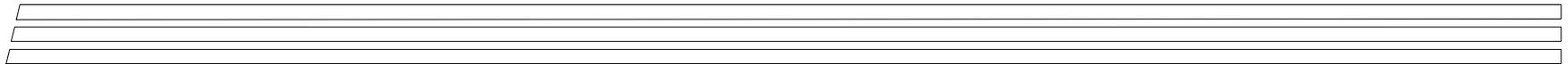
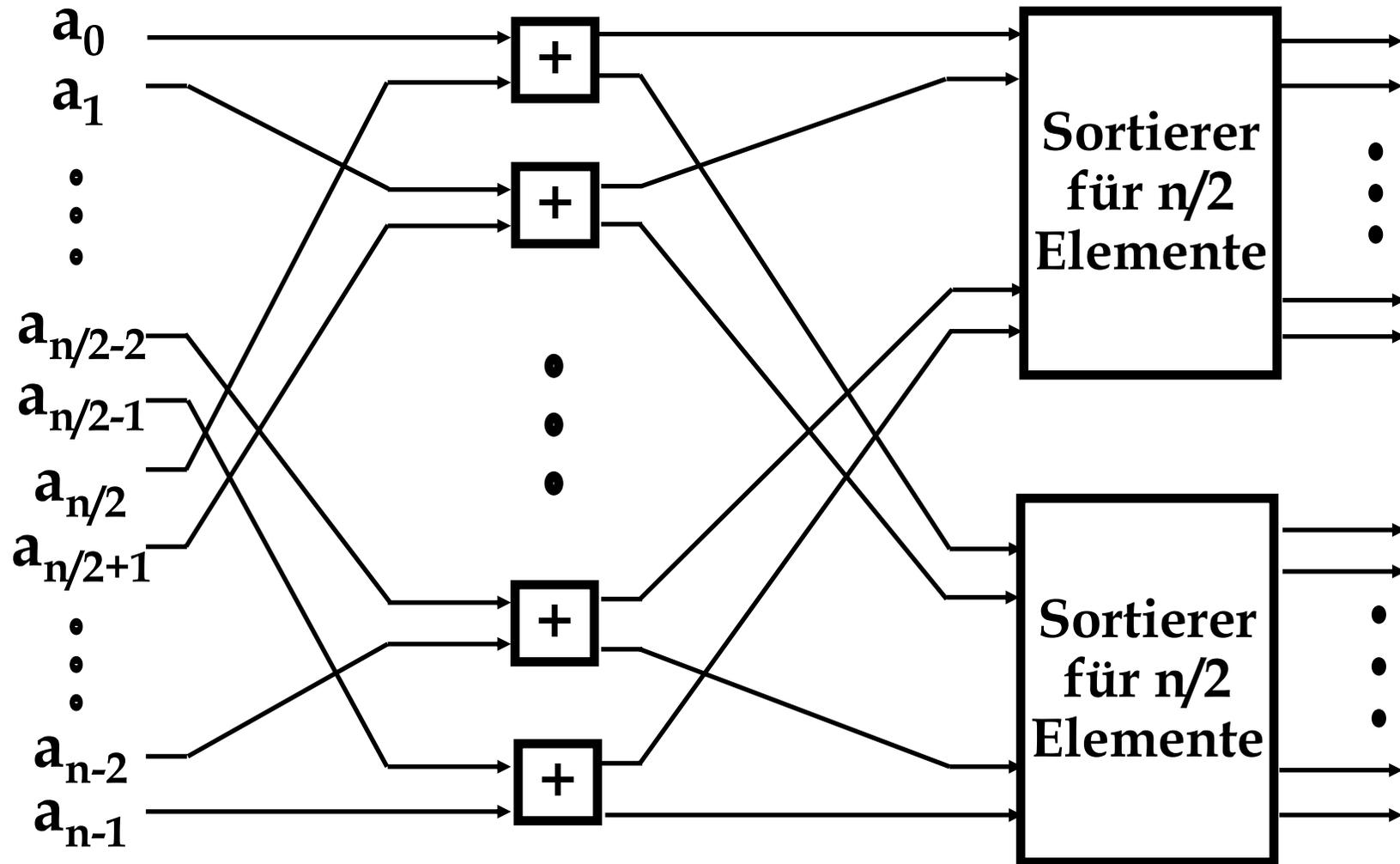


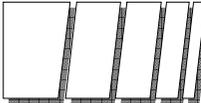
Zerlegen bitonischer Folgen (allg. Fall)

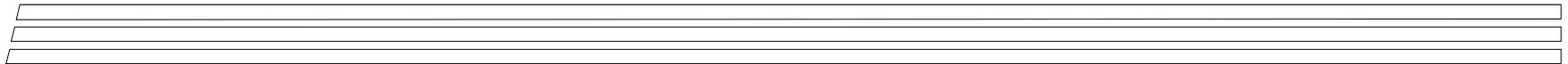
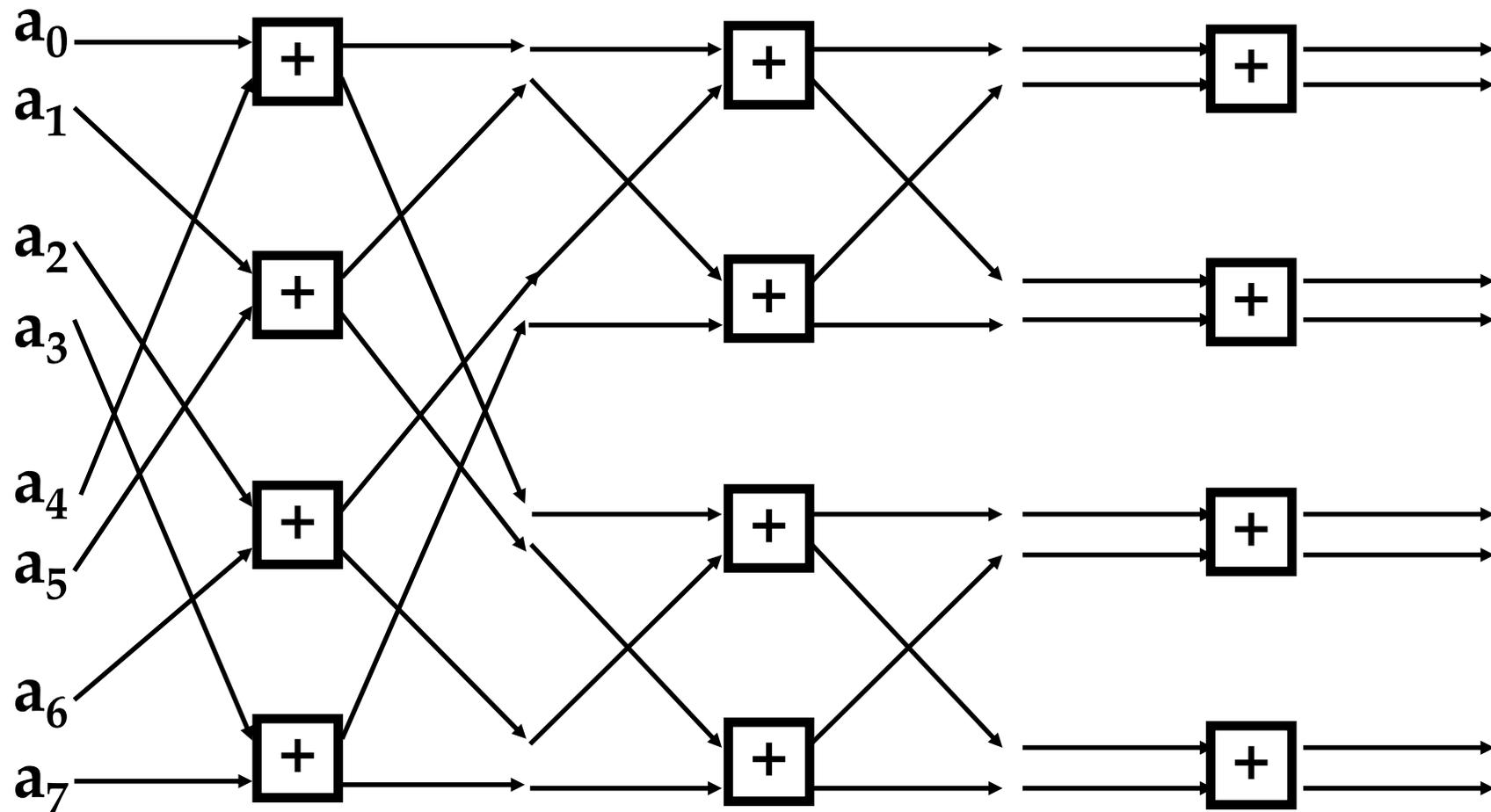
Zerlege eine bitonische Folge der Länge n
in zwei bitonische Folgen der Länge $n/2$

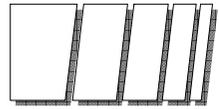


Bitonischer Sortierer

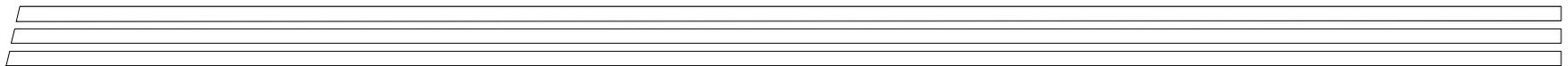
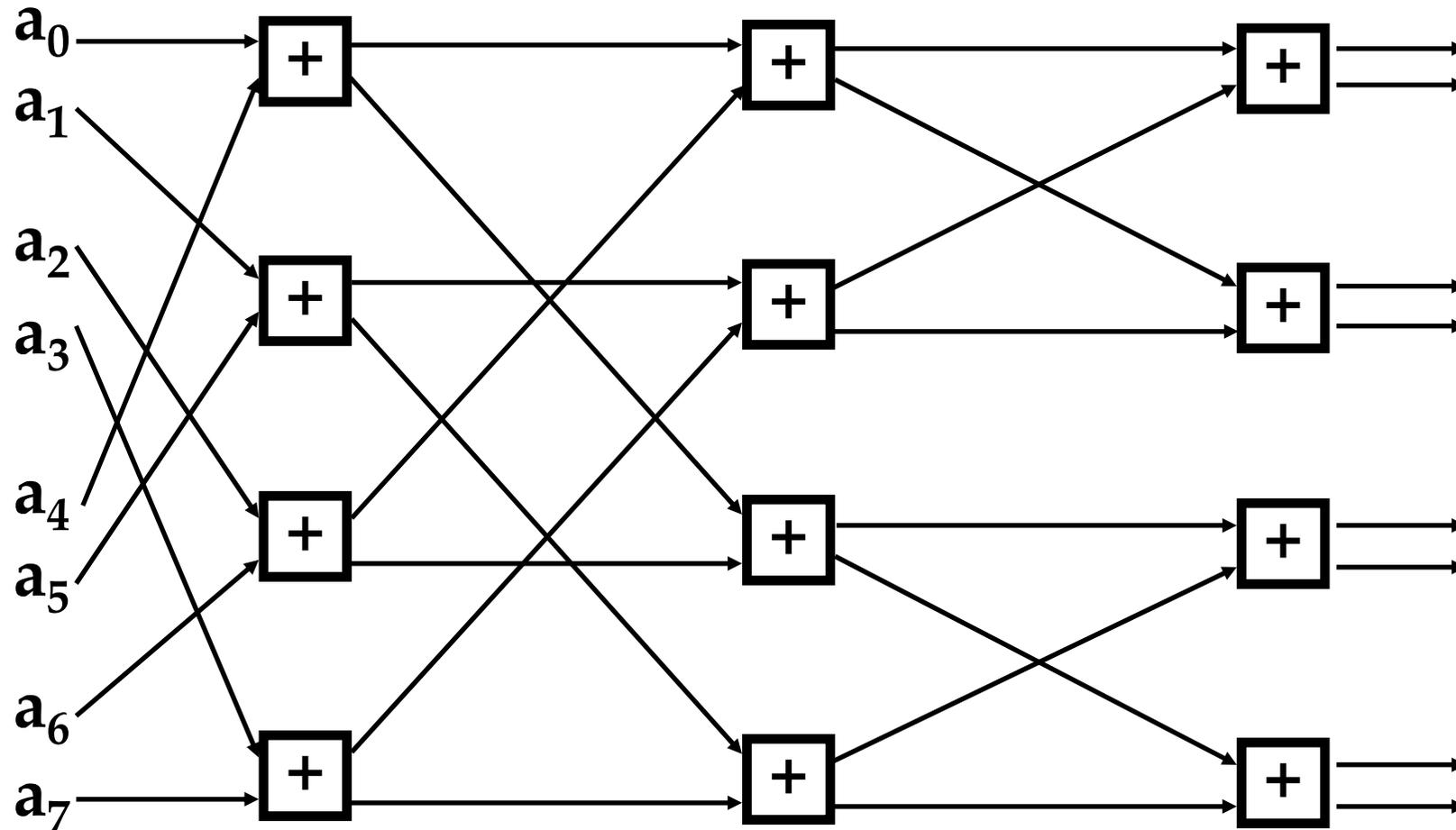


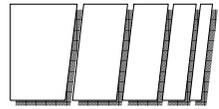
 **Bitonischer Sortierer für 8 Elemente**
(rekursiver Aufbau)





Bitonischer Sortierer für 8 Elemente



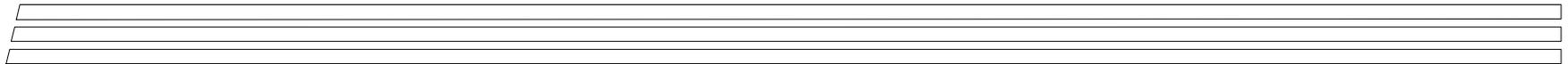


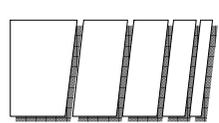
Satz von Batcher (1968)

Eine unsortierte Liste mit $n=2^k$ Elementen kann mit einem Netzwerk aus insgesamt $2^{k-2} * k * (k+1)$ Komparatoren in der Zeit $O((\log n)^2) = O(k^2)$ sortiert werden.

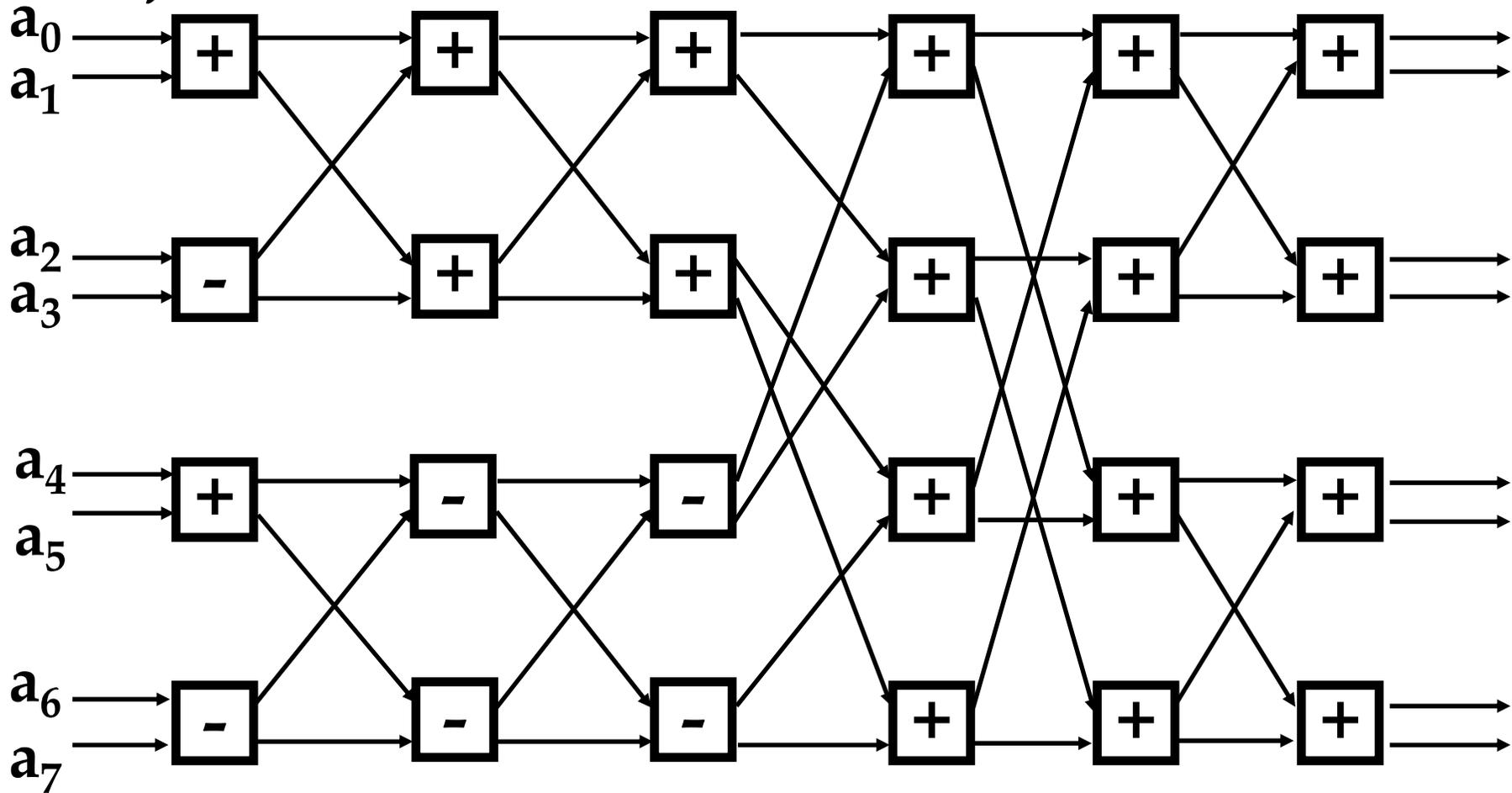
Beweisidee: unsortierte Liste der Länge n
= n sortierte Listen der Länge 1
= $n/2$ bitonische Listen der Länge 2

allgemein: aufsteigend sortierte Liste der Länge 2^m ++
absteigend sortierte Liste der Länge 2^m
= bitonische Liste der Länge 2^{m+1}
=> sortierbar mit $(m+1)$ -stufigem Netzwerk mit 2^m Komparatoren je Stufe





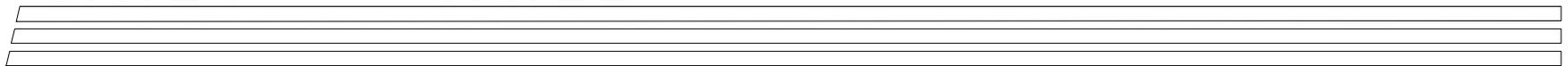
Batchers Sortiernetzwerk (für 8 Elemente)

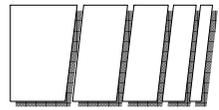


STUFE 1

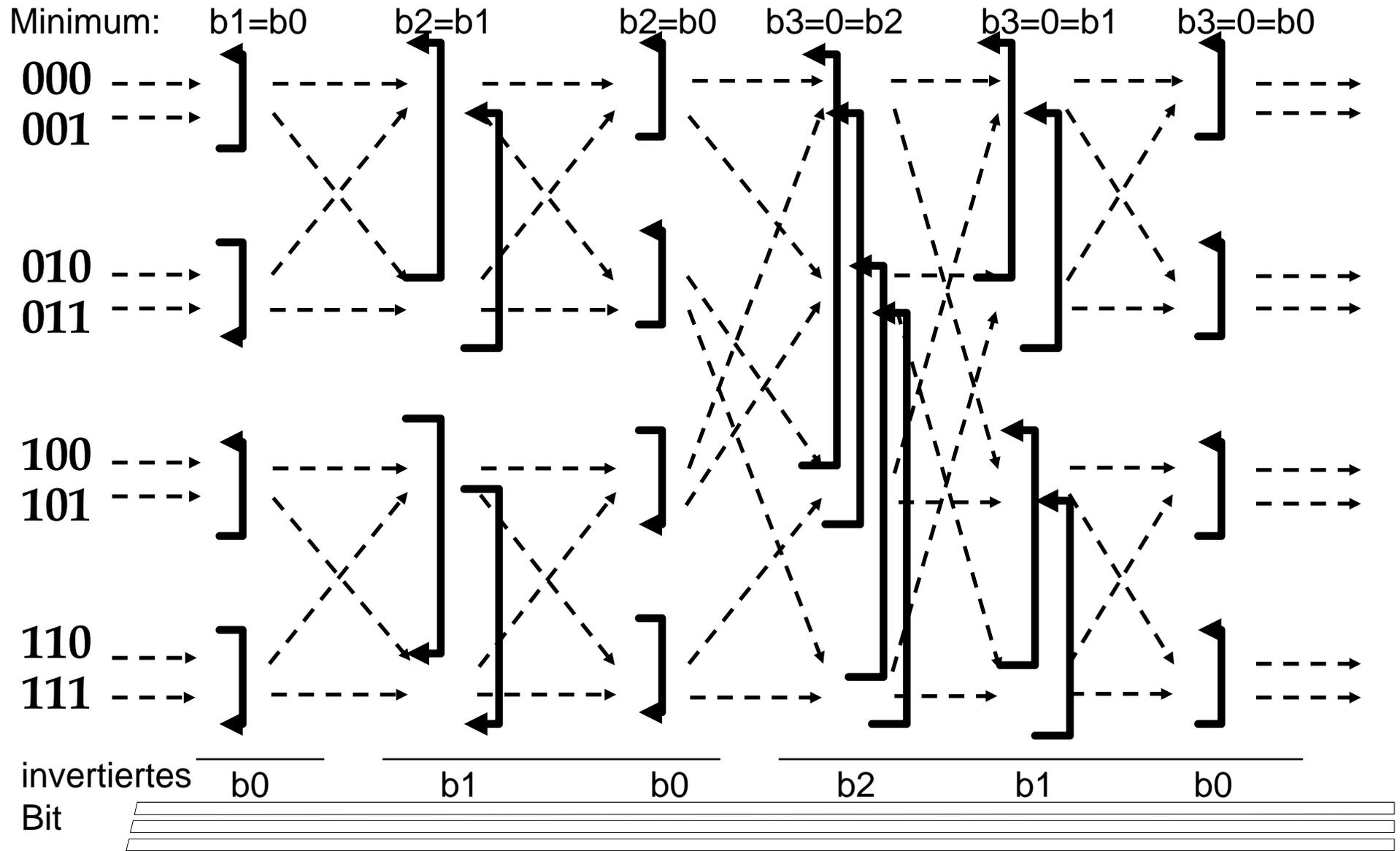
STUFE 2

STUFE 3





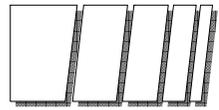
Kommunikationsmuster



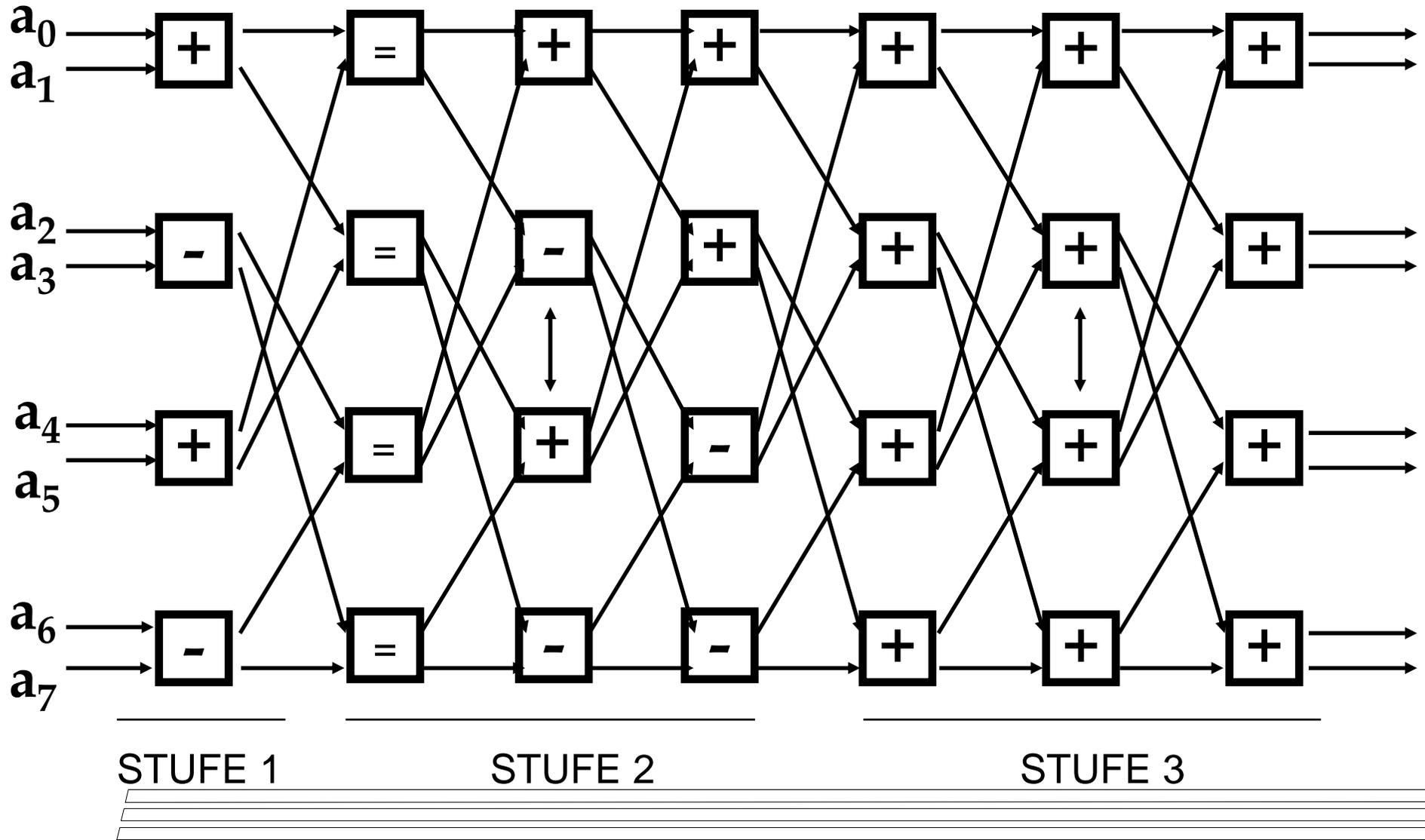
Auszug aus parallelem Programm

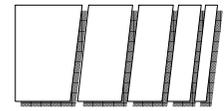
. . .

```
op int chan [0:n-1, 0:k-1]
process p [myid=0 to n-1] {
  int my_element, partner_element
  my_element = a[myid]
  for [i=1 to k] { (* k Stufen des Sortieralgorithmus *)
    for [j=i-1 downto 0] {
      (* Sortieren bitonischer Listen der Länge 2i *)
        . . . (* Bestimme Kommunikationspartner durch
          Invertieren des j-ten Bits von myid*)
        send chan[myid,j](my_element)
        receive chan[partner_id,j](partner_element)
        if (bit(i,myid) == bit(j,myid))
          { my_element = min(my_element,partner_element) }
        else { my_element = max(my_element,partner_element) }
      } }
  }
```

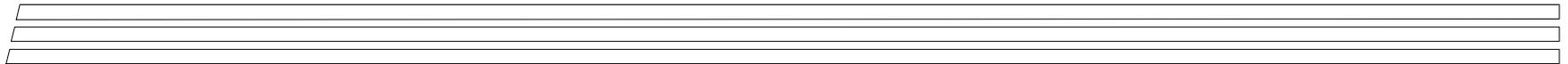
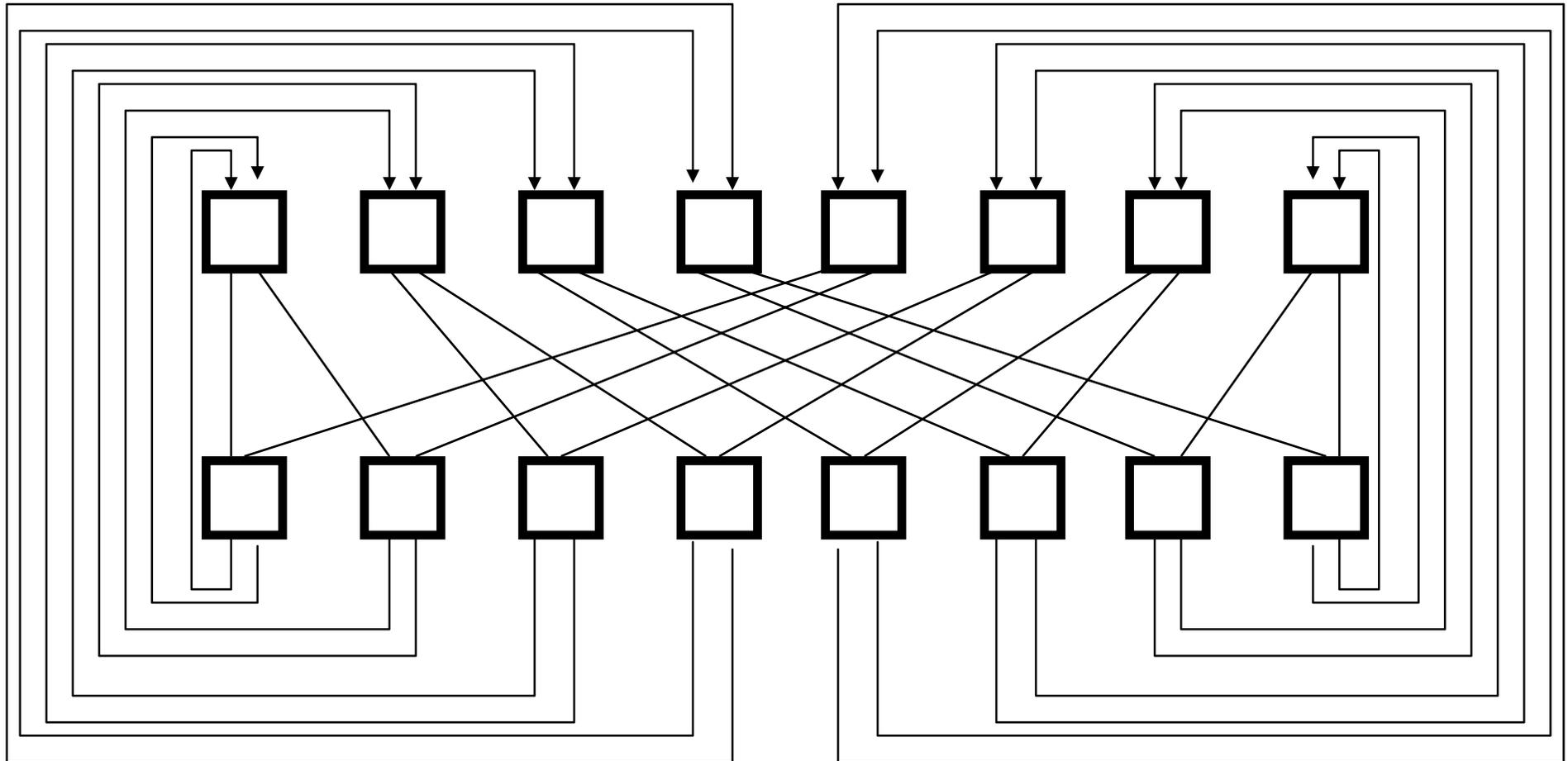


Sortiernetzwerk von Stone





Sortiermaschine nach Stone

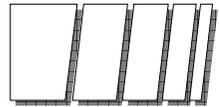


Hyper-QuickSort

- **Initialisierung:**
Eine Liste von n Werten wird gleichmäßig auf die 2^k Knoten einer Hypercube-Struktur verteilt.
=> Jeder Knoten erhält $n/2^k$ Listenelemente.
- **Ziele:**
 1. Die Teilliste auf jedem Prozessorknoten ist sortiert.
 2. Alle Elemente auf P_i sind kleiner oder gleich zu allen Elementen auf P_{i+1} ($0 \leq i \leq p-2 = 2^k-2$).

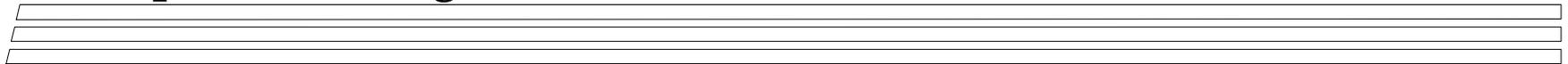
Eine gleichmäßige Verteilung der Listenelemente wird nicht gefordert.
- **1. Schritt: Jeder Knoten sortiert die ihm zugeordnete Teilliste mit einem optimalen sequentiellen Sortierverfahren.**

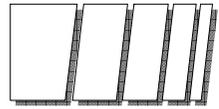
=====
=====
=====
=====
=> Ziel 1 ist erfüllt.



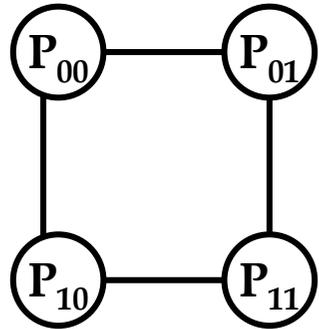
Rekursive divide-et-impera-Schritte

- Teilhypercubes der Dimension d werden in 2 Teilhypercubes der Dimension $d-1$ geteilt.
- Jeder Knoten in einem Teilhypercube sendet Werte zu seinem direkten Nachbarn im anderen Teilhypercube.
- Ziel ist es, in einem Teilhypercube die kleineren Werte und im anderen die größeren Werte bzgl eines Pivotelementes zu sammeln.
- Das Pivotelement wird etwa als mittleres Element eines ausgezeichneten Knotens gewählt, der dieses Element an alle übrigen Knoten eines Teilhypercubes per Broadcast verschickt.
- Jeder Knoten führt split-und-merge-Schritte durch:
 - split teilt gemäß Pivotelement aus und verschickt eine Hälfte an Partner
 - merge mischt verbleibende Hälfte mit den vom Partner erhaltenen Werten
- Nach k split-und-merge-Schritten ist auch Ziel 2 erreicht.





Beispiel: $n=32, k=2$



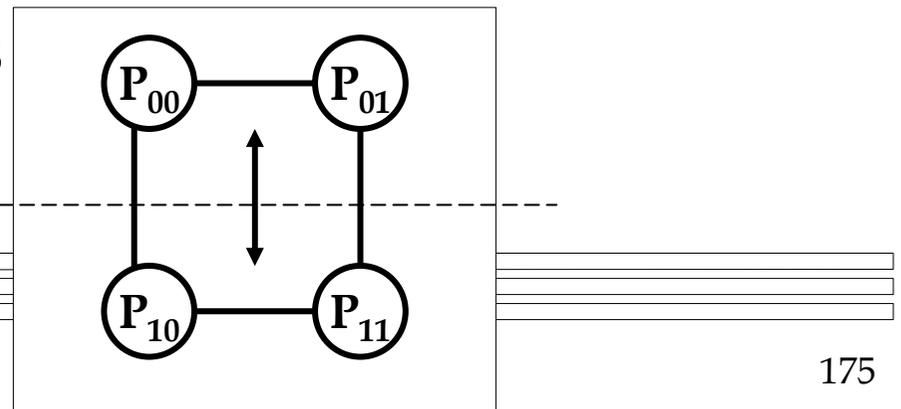
| | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|
| P_{00} : | 97 | 48 | 16 | 8 | 66 | 96 | 17 | 49 |
| P_{01} : | 58 | 76 | 54 | 39 | 82 | 47 | 65 | 51 |
| P_{10} : | 11 | 50 | 53 | 95 | 36 | 67 | 86 | 44 |
| P_{11} : | 35 | 16 | 81 | 1 | 44 | 23 | 15 | 5 |

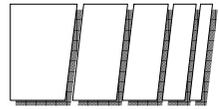
1. Schritt: lokales sequentielles Sortieren

| | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|
| P_{00} : | 8 | 16 | 17 | 48 | | 49 | 66 | 96 | 97 |
| P_{01} : | 39 | 47 | | 51 | 54 | 58 | 65 | 76 | 82 |
| P_{10} : | 11 | 36 | 44 | | 50 | 53 | 67 | 86 | 95 |
| P_{11} : | 1 | 5 | 15 | 16 | 23 | 35 | 44 | | 81 |

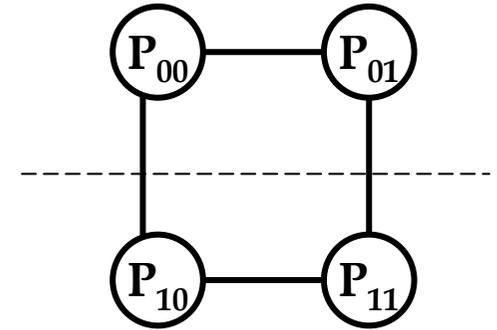
2. Schritt: P_{00} sendet mittleren Wert 48 an alle übrigen Prozesse.

Aufteilung in Teilhypercubes:





1. Split-und-merge-Schritt

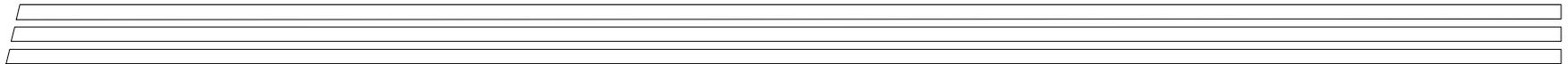


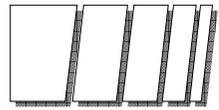
Datenaustausch zwischen Teilhypercubes (Split):

| | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|
| P_{00} : | 8 | 16 | 17 | 48 | 49 | 66 | 96 | 97 |
| P_{01} : | 39 | 47 | 51 | 54 | 58 | 65 | 76 | 82 |
| P_{10} : | 11 | 36 | 44 | 50 | 53 | 67 | 86 | 95 |
| P_{11} : | 1 | 5 | 15 | 16 | 23 | 35 | 44 | 81 |

Mischen von Teillisten (Merge):

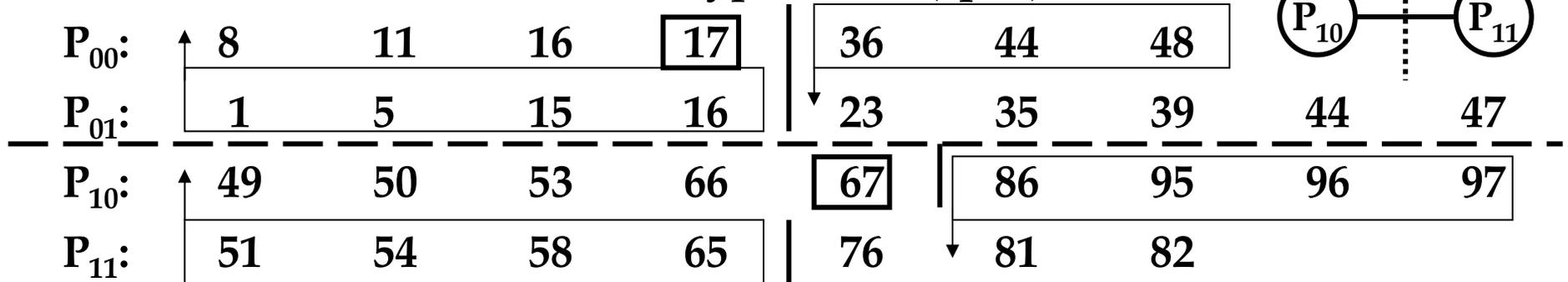
| | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|
| P_{00} : | 8 | 11 | 16 | 17 | 36 | 44 | 48 | | |
| P_{01} : | 1 | 5 | 15 | 16 | 23 | 35 | 39 | 44 | 47 |
| P_{10} : | 49 | 50 | 53 | 66 | 67 | 86 | 95 | 96 | 97 |
| P_{11} : | 51 | 54 | 58 | 65 | 76 | 81 | 82 | | |





2. Split-und-merge-Schritt

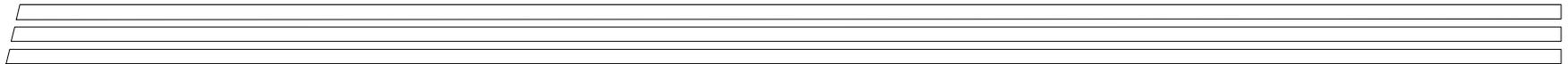
Datenaustausch zwischen Teilhypercubes (split)



Mischen der Teillisten (merge):

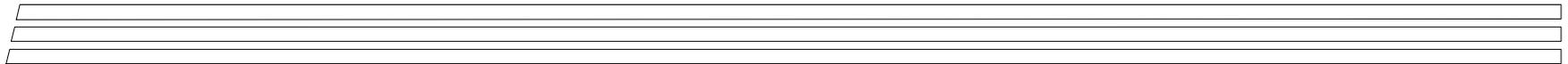
| | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|
| P_{00} : | 1 | 5 | 8 | 11 | 15 | 16 | 16 | 17 | |
| P_{01} : | 23 | 35 | 36 | 39 | 44 | 44 | 47 | 48 | |
| P_{10} : | 49 | 50 | 51 | 53 | 54 | 58 | 65 | 66 | 67 |
| P_{11} : | 76 | 81 | 82 | 86 | 95 | 96 | 97 | | |

=> Ziel 2 wurde ebenfalls erreicht.



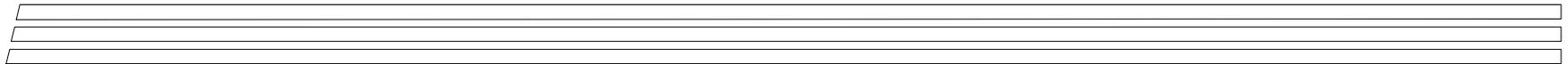
Analyse

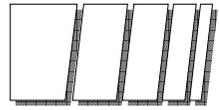
- **Rechenaufwand:**
 - **1. Schritt: optimales sequentielles Sortierverfahren:**
 $O(n/2^k \log (n/2^k)) = O(n/2^k (\log n - k))$
 - **2. - (k+1).Schritt: Aufteilung in Teilhypercubes der Dimensionen $k \Rightarrow k-1 \Rightarrow k-2 \Rightarrow \dots \Rightarrow 1 \Rightarrow 0$ im besten Fall der gleichmäßigen Aufteilung:**
 $O(n/2^k k)$ parallele Vergleichsschritte
 \Rightarrow insgesamt $O(n/2^k \log n)$ parallele Vergleichsschritte
- **Kommunikationsaufwand**
 - **Broadcast von Pivotelementen:** i Komm. in i -ter Iteration
 - **Elementaustausch, im besten Fall:** $n/2^{k+1}$
 - **pro Iteration:** $O(i+n/2^{k+1})$ bei k Iterationen
 \Rightarrow insgesamt $O(n \log p/p)$ pro Knoten



Nachteile von Hyper-Quicksort

- **hoher Kommunikationsaufwand**
 - Elemente „wandern“ über mehrere Zwischenknoten zur Zielposition
- **kritische Pivotwahl**
 - schlechte Lastbalancierung bei ungünstigem Pivotelement

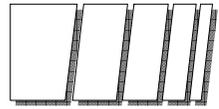




Der PSRS-Algorithmus (Li et al. 92) *(Parallel Sorting by Regular Sampling)*

- **Merkmale:**
 - bessere Pivotauswahl
 - Elemente werden höchstens einmal kommuniziert.
- **4 Phasen:**
 1. sequentielles Quicksort auf Teilsegmenten der zu sortierenden Liste => Auswahl von p Elementen (Probe)
 2. Ein Prozessor sammelt alle p^2 Probenelemente (je p Elemente von p Prozessoren) und sortiert diese.
=> Auswahl von $p-1$ Pivotelementen und Broadcast von diesen an alle Prozesse
 3. Jeder Prozess teilt seine Teilliste in p Teillisten gemäß der Pivotelemente und verschickt die j -te Partition an Prozess j für $1 \leq j \leq p$.
 4. Jeder Prozess mischt die ihm geschickten p Partitionen zu einer

sortierten Teilliste.



Phase I: sequentielles Sortieren

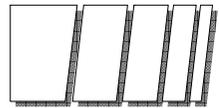
- Jeder Prozess erhält bis zu $\lceil n/p \rceil$ Elemente der Gesamtliste und sortiert mit sequentielltem Quicksort.
=> p sortierte Teillisten mit bis zu $\lceil n/p \rceil$ Elementen
- Auswahl von p Elementen an den Positionen
1, $\lceil n/p^2 \rceil + 1$, $2\lceil n/p^2 \rceil + 1$, $3\lceil n/p^2 \rceil + 1$, ..., $(p-1)\lceil n/p^2 \rceil + 1$
=> reguläre Probe aus sortierter Teilliste

Beispiel: $p=3, n=27 \Rightarrow n/p = 9, n/p^2 = 3 \Rightarrow$ Probepositionen: 1 4 7

| | | | | | | | | | |
|-----|----|----|----|----|----|----|----|----|----|
| P1: | 15 | 46 | 48 | 93 | 39 | 6 | 72 | 91 | 14 |
| P2: | 36 | 69 | 40 | 89 | 61 | 97 | 12 | 21 | 54 |
| P3: | 53 | 97 | 84 | 58 | 32 | 27 | 33 | 72 | 20 |

↓ PHASE 1 ↓

| | | | | | | | | | |
|-----|----|----|----|----|----|----|----|----|----|
| P1: | 6 | 14 | 15 | 39 | 46 | 48 | 72 | 91 | 93 |
| P2: | 12 | 21 | 36 | 40 | 54 | 61 | 69 | 89 | 97 |
| P3: | 20 | 27 | 32 | 33 | 53 | 58 | 72 | 84 | 97 |



Phase II: Auswahl von $p-1$ Pivotelementen

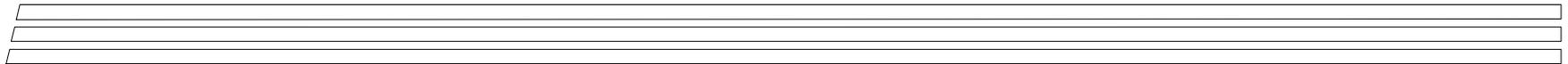
- Ein Prozess sammelt alle Proben aus jeweils p Elementen und sortiert diese.
=> sortierte Teilliste mit p^2 Elementen
- Auswahl von $p-1$ Pivotelementen aus dieser Listenprobe an den Stellen
 $p + \lfloor p/2 \rfloor, 2p + \lfloor p/2 \rfloor, \dots, (p-1)p + \lfloor p/2 \rfloor$
- Broadcast dieser Pivotelemente an alle Prozesse

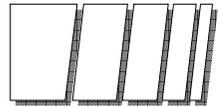
Beispiel (Forts.) $p + \lfloor p/2 \rfloor = 3+1 = 4 \Rightarrow$ Positionen Pivotelemente: 4 7

Probenelemente: 6 39 72 | 12 40 69 | 20 33 72

sortierte Probe: 6 12 20 33 39 40 69 72 72

2 Pivotelemente





Phase III: Partitionen kommunizieren

- Jeder Prozess teilt seine Teilliste mit bis zu $\lceil n/p \rceil$ Elementen in p Partitionen gemäß der Pivotelemente auf.
- Prozess i behält Partition i und verschickt die $p-1$ Partitionen j mit $j \neq i$ an Prozess j .

Beispiel (Forts.): Pivotelemente: 33 und 69

P1: 6 14 15 | 39 46 48 | 72 91 93

P2: 12 21 | 36 40 54 61 69 | 89 97

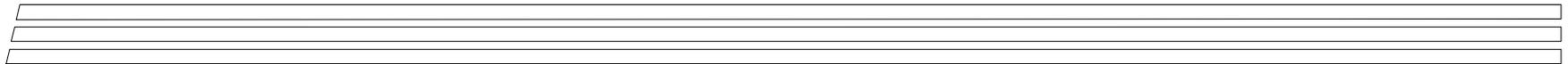
P3: 20 27 32 33 | 53 58 | 72 84 97

↓ PHASE III ↓

P1: 6 14 15 P2: 39 46 48 P3: 72 91 93

12 21 36 40 54 61 69 89 97

20 27 32 33 53 58 72 84 97



Analyse

- vereinfachende Annahmen:
 - p Prozesse
 - p geradzahlig
 - $n=p^2k$ paarweise verschiedene Elemente mit $k>1$

Berechnungsaufwand:

- Phase I:
paralleles sequentielles Sortieren von Teillisten der Länge n/p
 \Rightarrow Aufwand: $O(n/p \log n/p) = O(n/p (\log n - \log p))$
 - Phase II: Sortieren der Listenprobe mit p^2 Elementen
 \Rightarrow Aufwand: $O(p^2 \log p^2) = O(p^2 \log p)$
 - Phase III: Aufteilen der Partitionen \Rightarrow Aufwand: $O(n/p)$
 - Phase IV: Mischen von p sortieren Teillisten, Aufwand: $O(n/p \log p)$
- \Rightarrow Gesamtkomplexität: $O(n/p \log n + p^2 \log p)$

Falls $n \geq p^3$ dominiert der erste Term: $O(n \log n / p)$



Zum Aufwand von Phase IV

Satz: Jeder Prozess hat maximal $2n/p$ Elemente zu mischen.

Mit $\log p$ Mischstufen mit je $2n/p$ Vergleichen ergibt sich damit der oben angenommene Aufwand $O(n/p \log p)$.

Der obige Satz folgt aus dem folgenden Theorem:

Theorem: Bezeichnet φ_i die Anzahl der Listenelemente, die in Phase IV von Prozess i gemischt werden müssen, so gilt:

$$\max_{1 \leq i \leq p} \varphi_i \leq 2n/p - n/p^2 - p + 1$$

