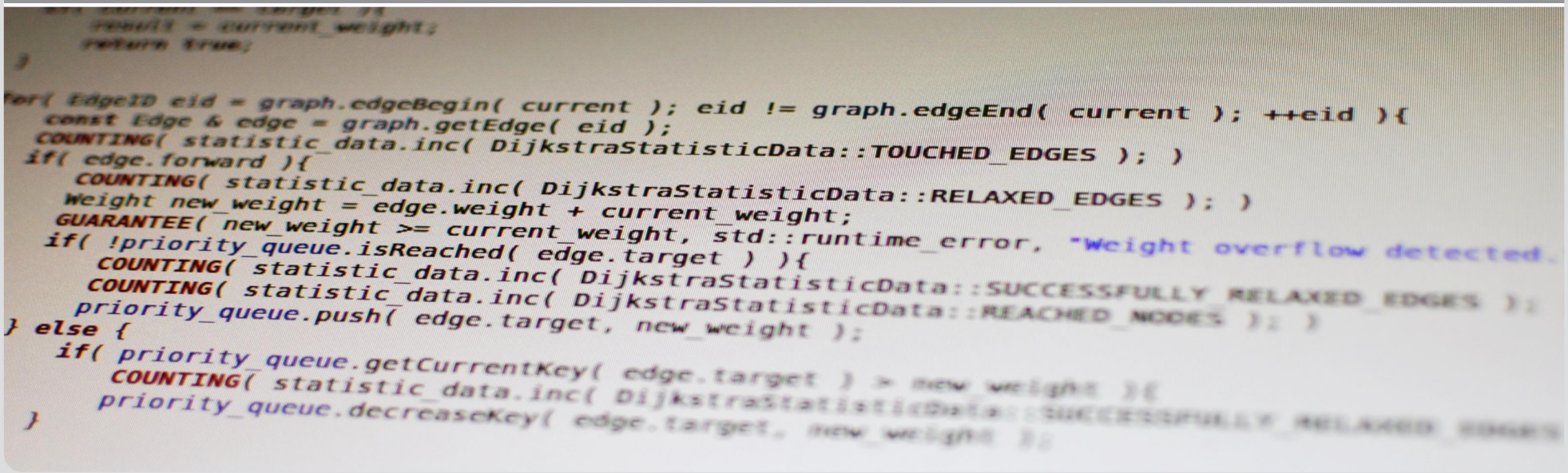
Karlsruher Institut für Technologie

Übung 8 – Algorithmen II

Moritz Laupichler, Nikolai Maas – *{moritz.laupichler, nikolai.maas}@kit.edu*

`https://algo2.iti.kit.edu/AlgorithmenII_WS23.php`

Institut für Theoretische Informatik - Algorithm Engineering



Themenübersicht

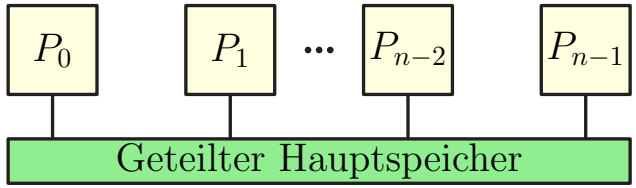
- Parallele Algorithmen
- Externe Algorithmen

Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

BulkSynchronousParallel

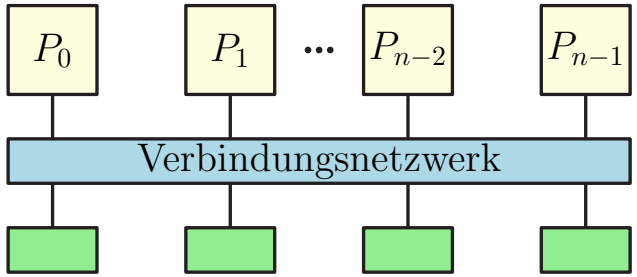
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

Bulk**S**ynchronous**P**arallel

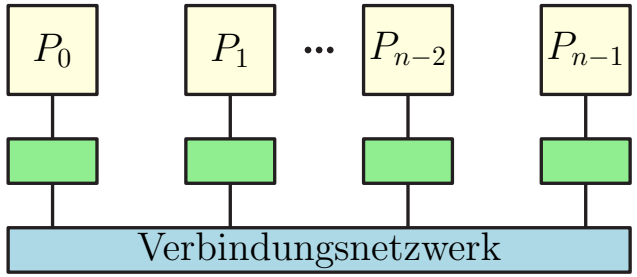
- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Parallelverarbeitung

Modelle

PRAM (Shared Memory)

- synchrone Prozessoren
- gemeinsamer Speicher
- Speicherkonflikte



Gemeinsamer Speicher

Verteilter Speicher (Distributed Memory)

Bulk**S**ynchronous**P**arallel

- kollektiver Nachrichtenaustausch aller Rechner
- BSP* berücksichtigt Nachrichtenlänge

Verbindungsnetzwerke

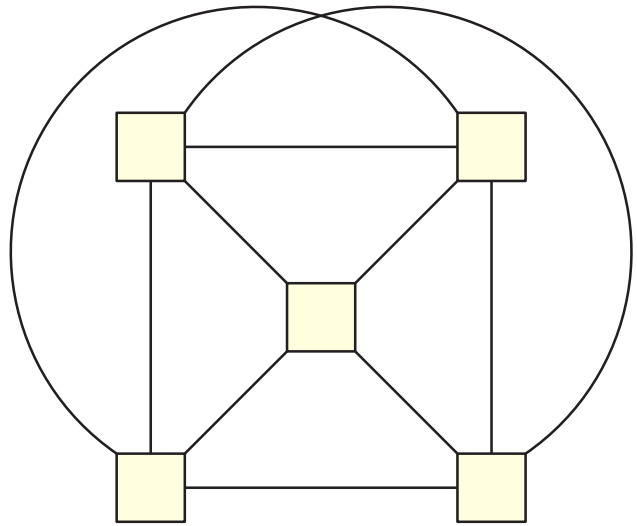
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex
  - Telefon
  - Duplex

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

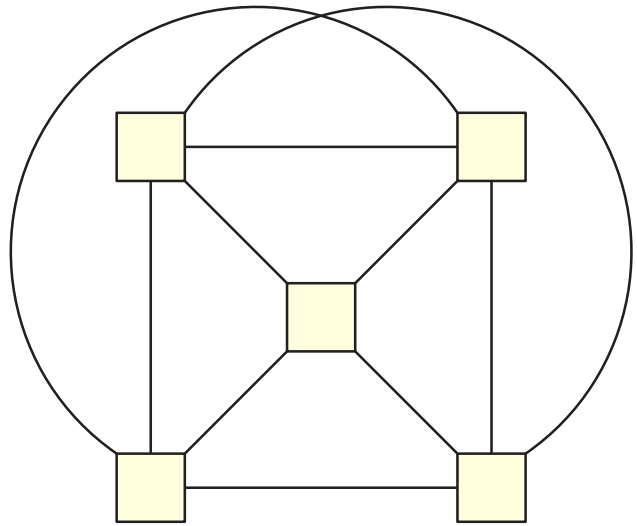
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$

Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

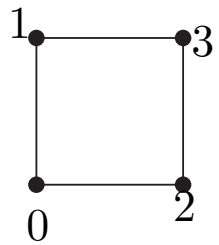
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

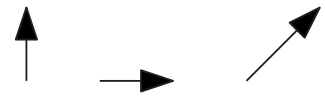
- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

Struktur

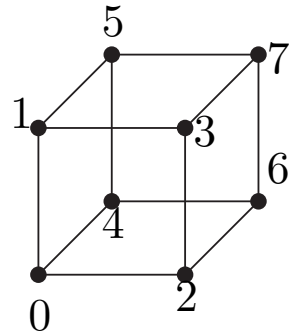
Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$



Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $***1** \leftrightarrow ***0**$



Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Verbindungsnetzwerke

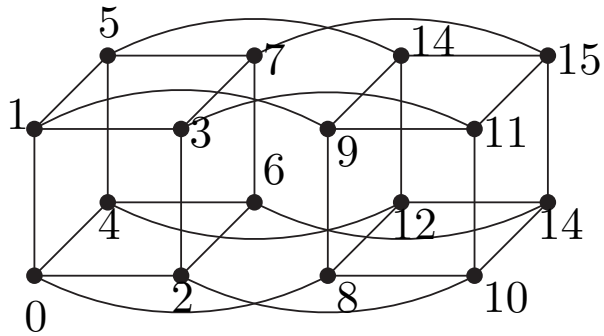
Struktur

Vollverkabelt

- nur für geringe Anzahl an Rechnern
- $\frac{p \cdot (p-1)}{2}$ Verbindungen nötig
- Varianten
  - Simplex $i \to j$
  - Telefon $i \leftrightarrow j$
  - Duplex $i \to j, k \to i$

Hyperwürfel

- $p \log p$ Verbindungen
- klare Nummerierung von Nachbarn
  $* * *1 * * \leftrightarrow * * *0 * *$

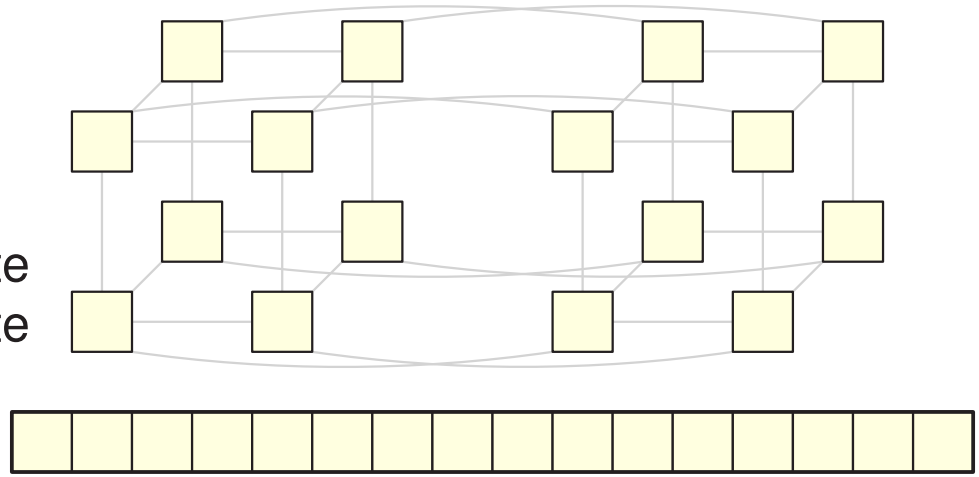


Kosten

- Kostenmaß Kommunikation $T_{comm} = T_{start} + l \cdot T_{byte}$

Präfixsumme

Hypercube

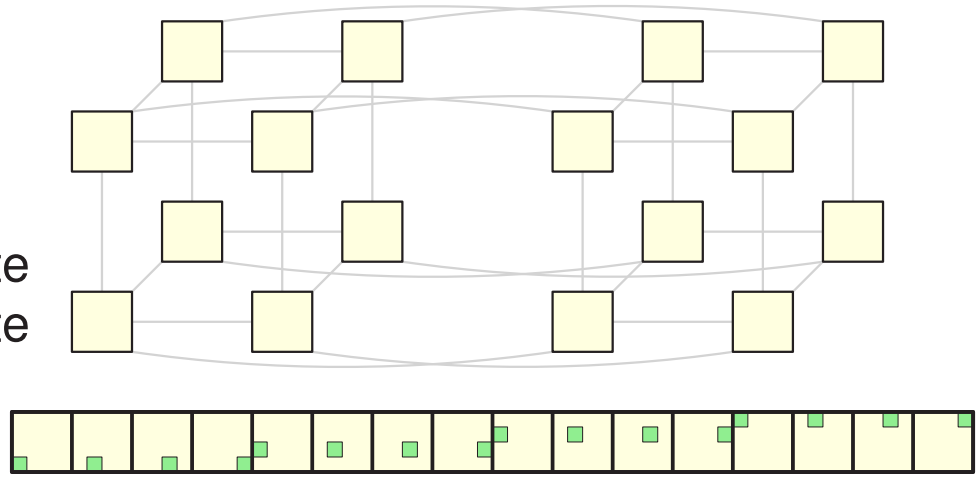


- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

Hypercube

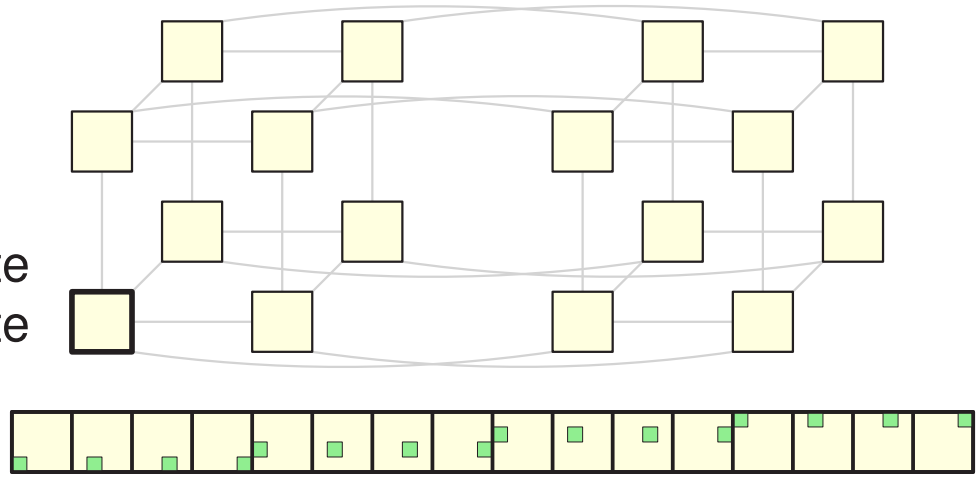


- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

Hypercube

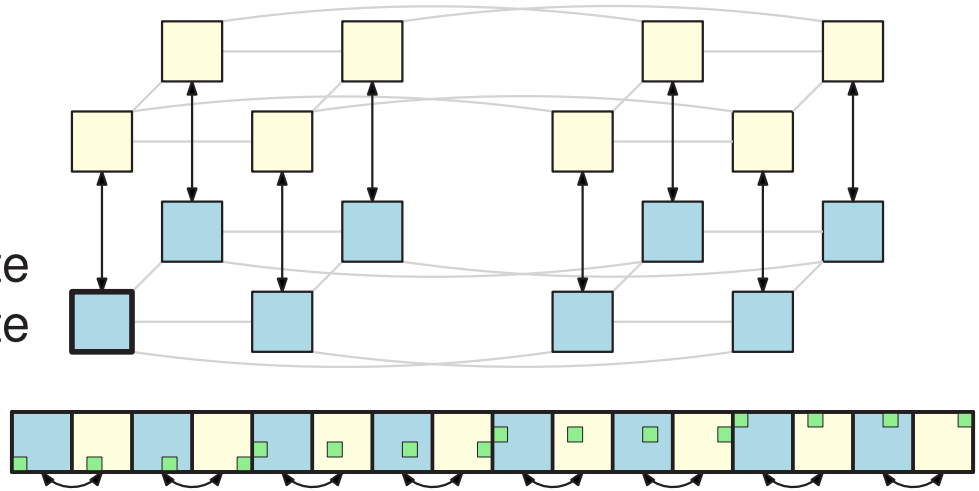


- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

Hypercube

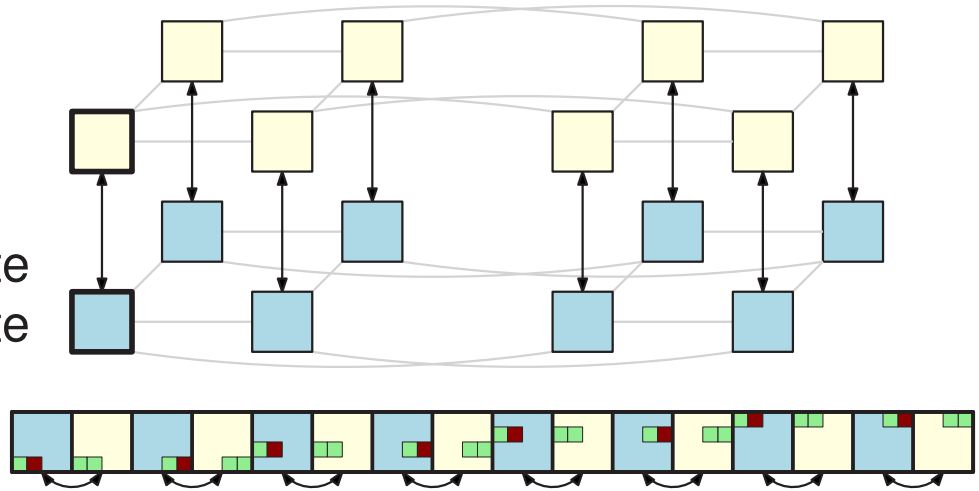


- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

Hypercube

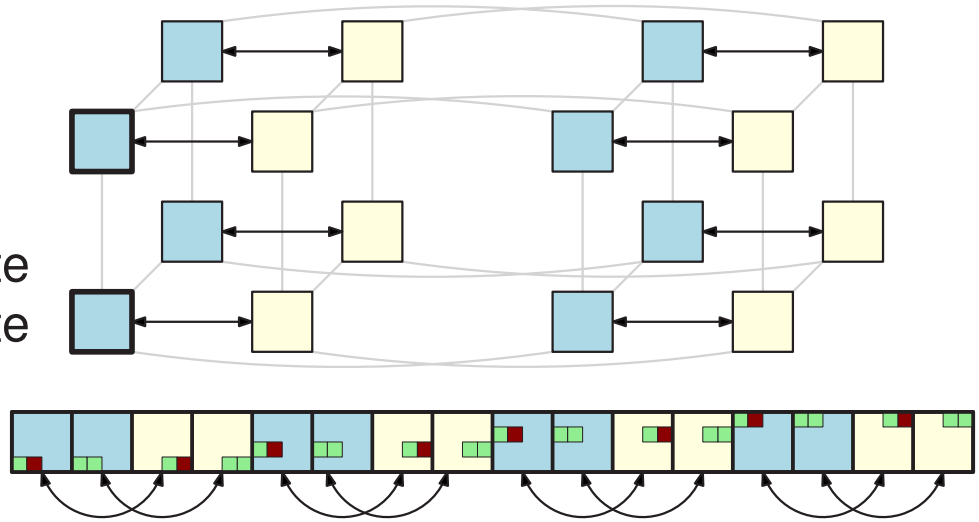


- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

Hypercube

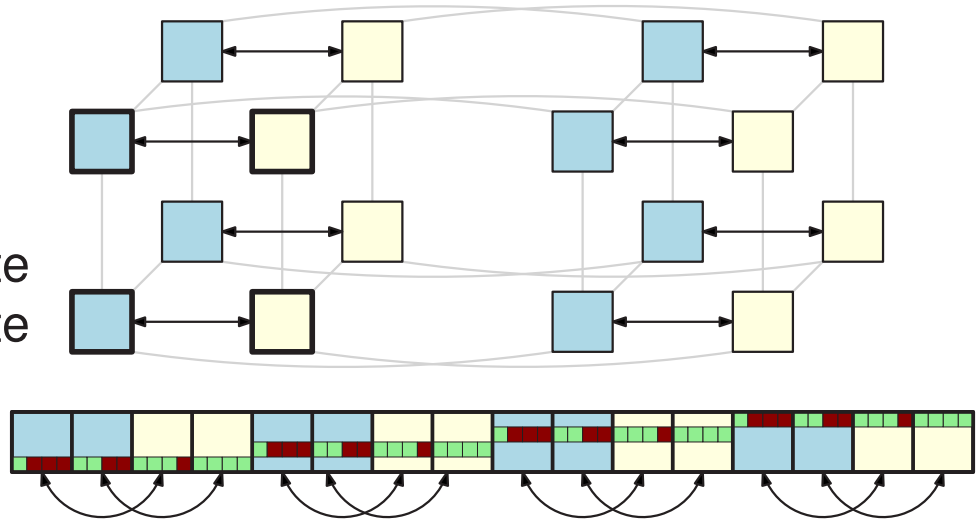


- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

Hypercube

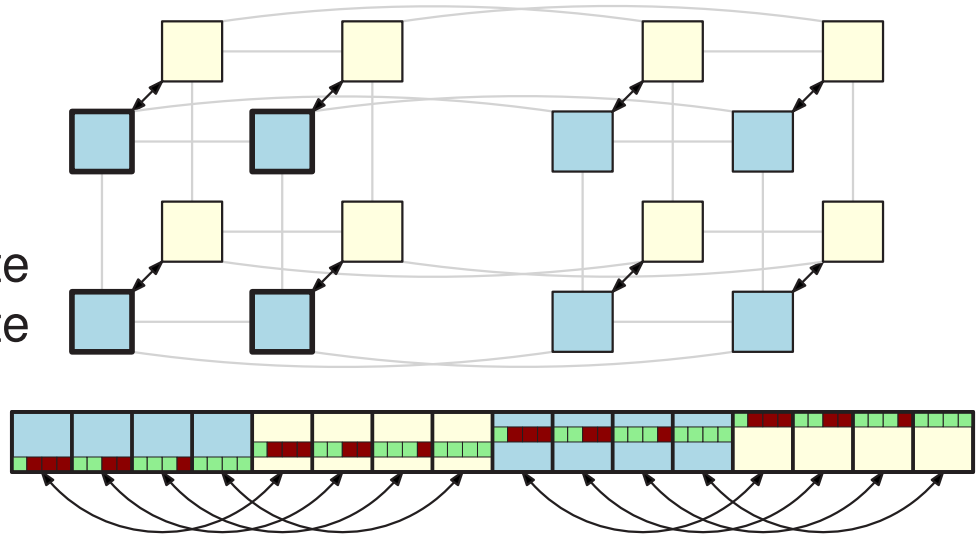


- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID

- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

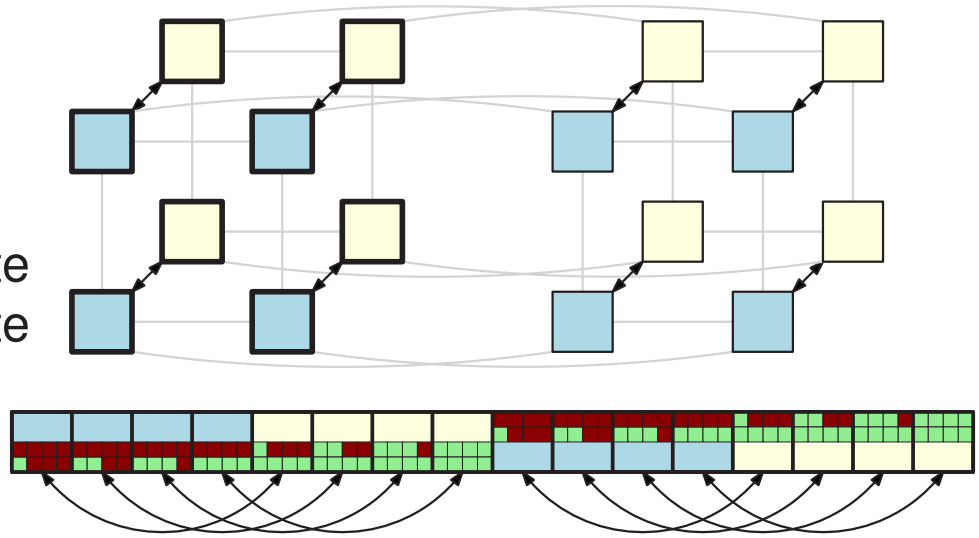
Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

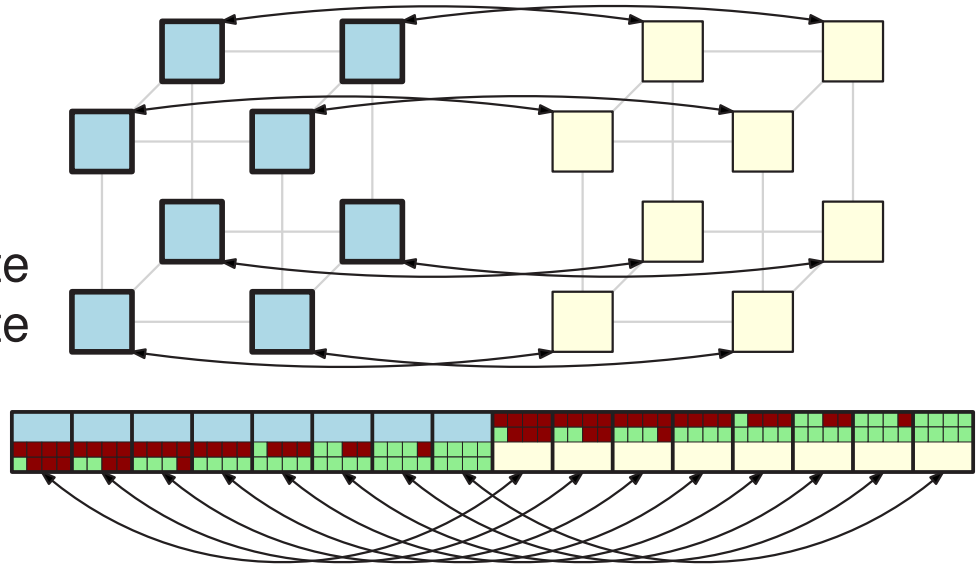
Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

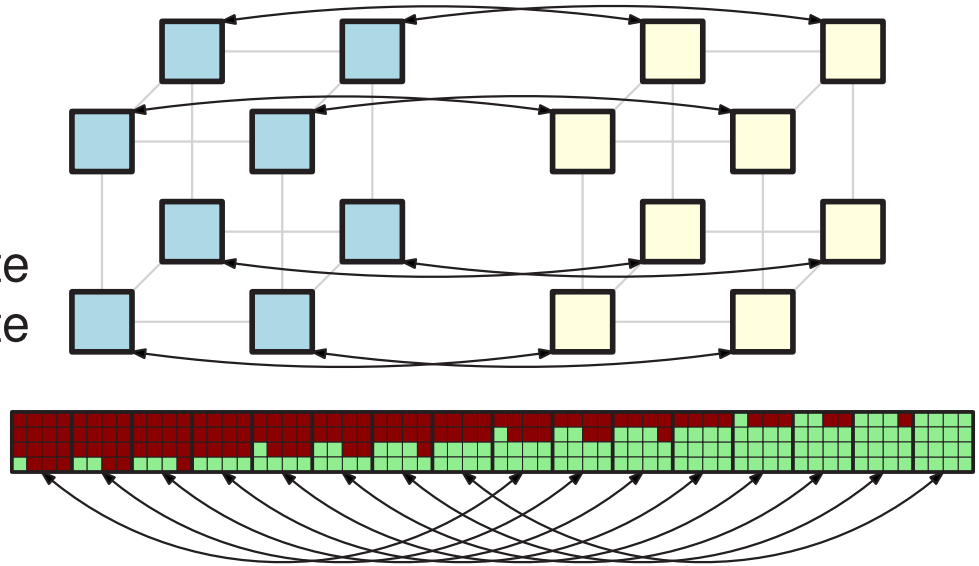
Hypercube



- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

Hypercube



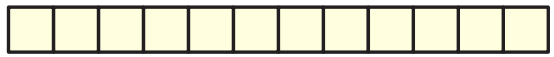
- Jede CPU speichert zwei Werte
  1. Summe aller bekannten Elemente
  2. Summe aller bekannten Elemente von CPUs mit kleinerer ID
- Schritt k: CPUs tauschen entlang k-ter Dimension ihre Summen aus
  - $\mathrm{ID}[k] = 1$: Daten kommen von CPUs mit kleinerer ID
    $\rightarrow$ gehört zur Präfixsumme
  - $\mathrm{ID}[k] = 0$: Daten kommen von CPUs mit größerer ID
    $\rightarrow$ gehört nicht zur Präfixsumme

Präfixsumme

Reduktionsbaum

- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase



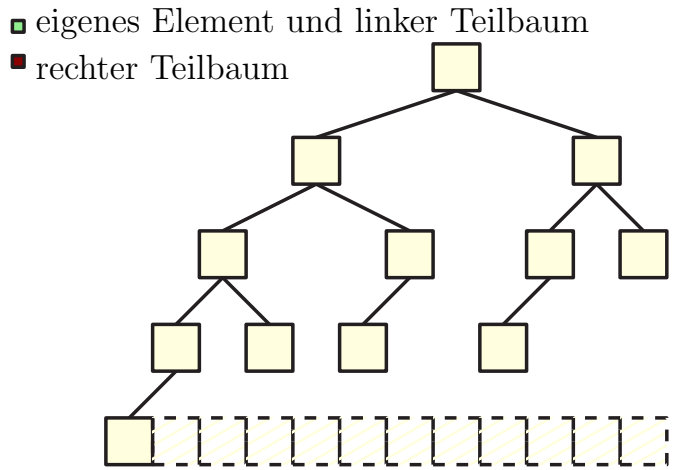
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

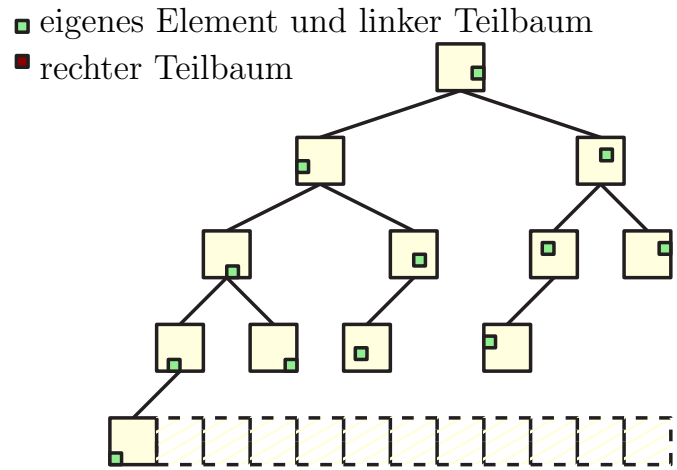
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

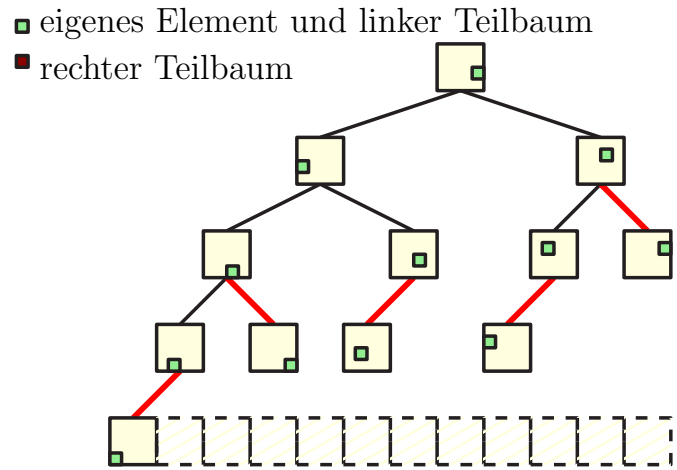
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

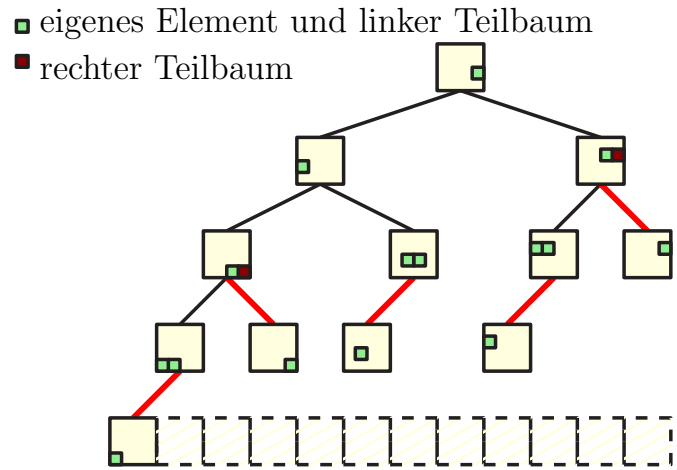
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

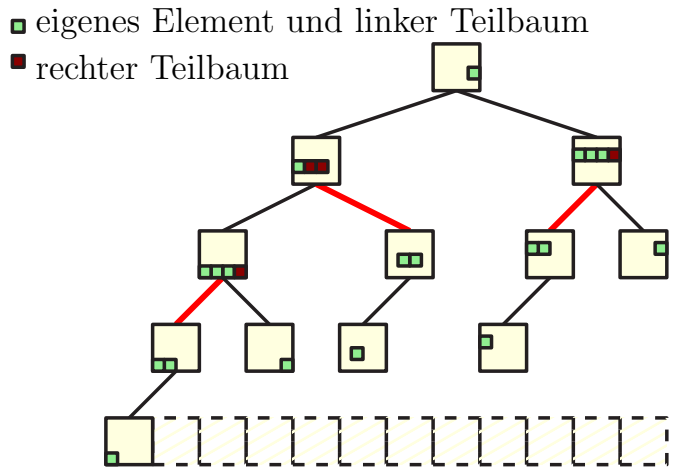
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

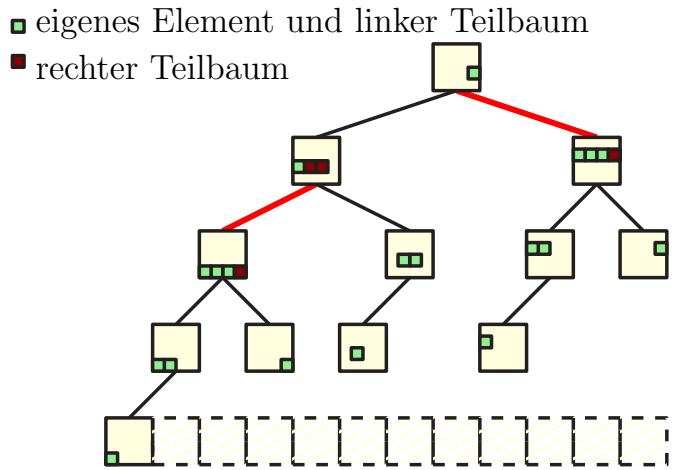
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

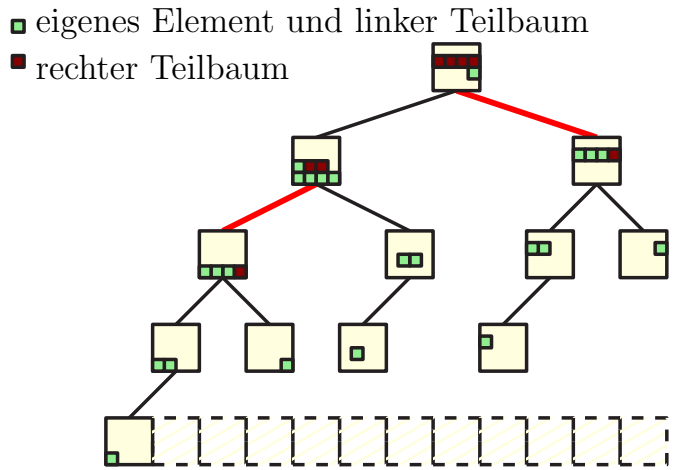
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

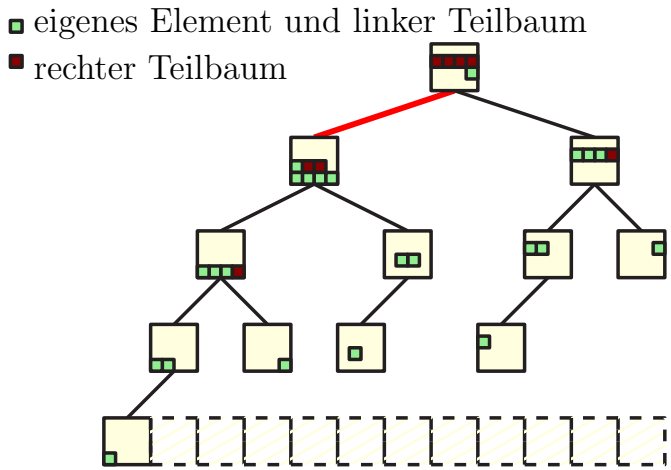
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

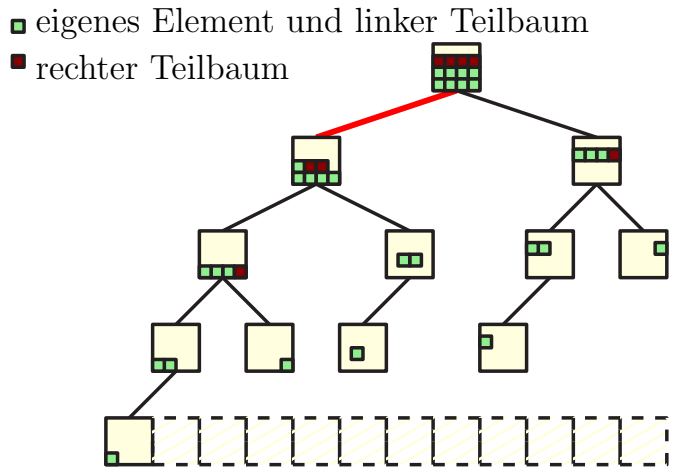
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

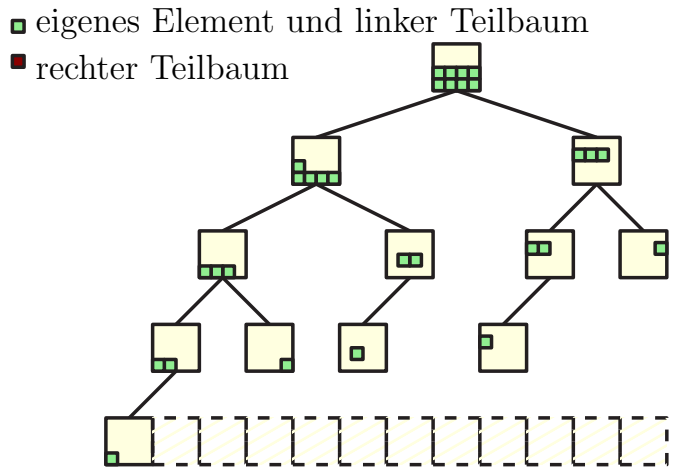
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

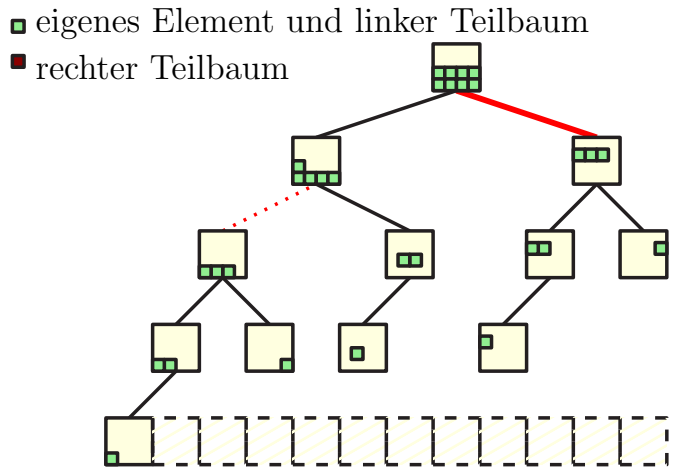
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

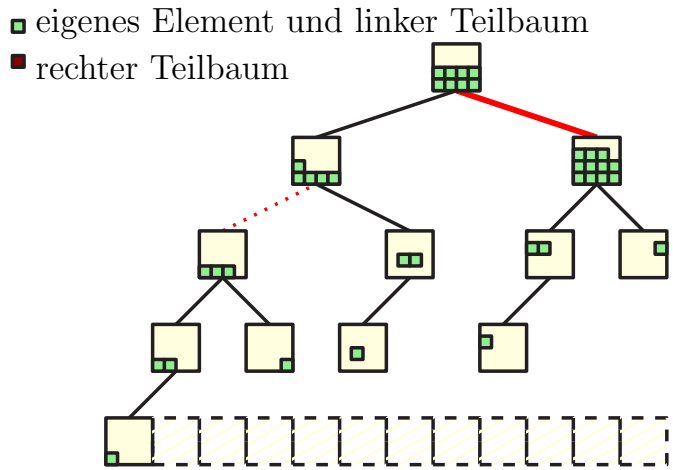
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

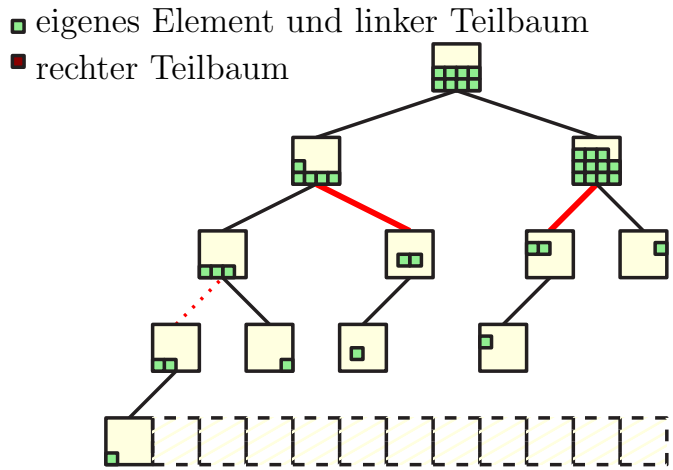
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

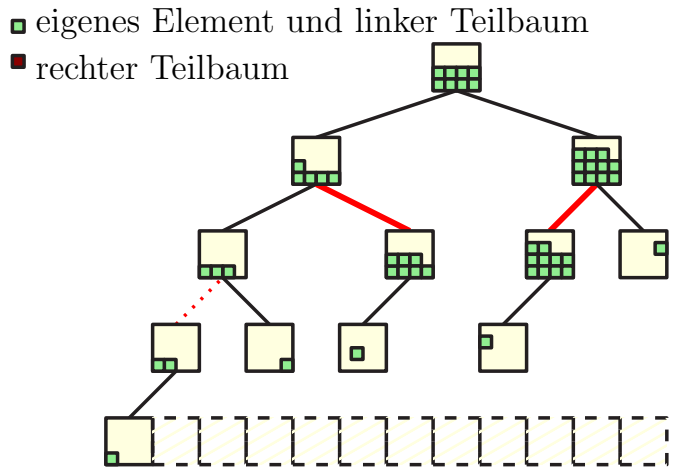
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

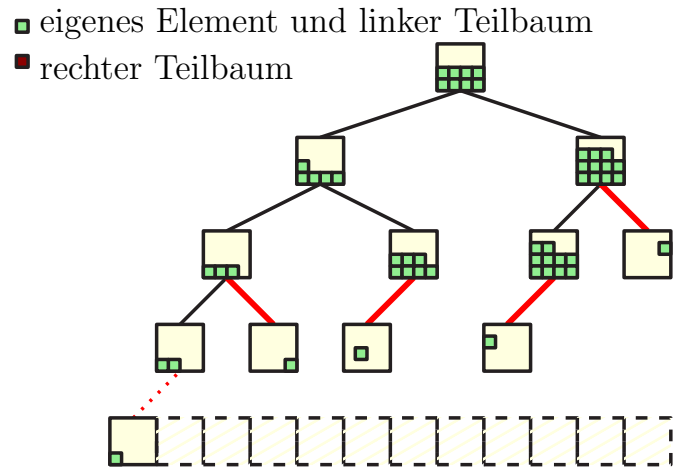
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

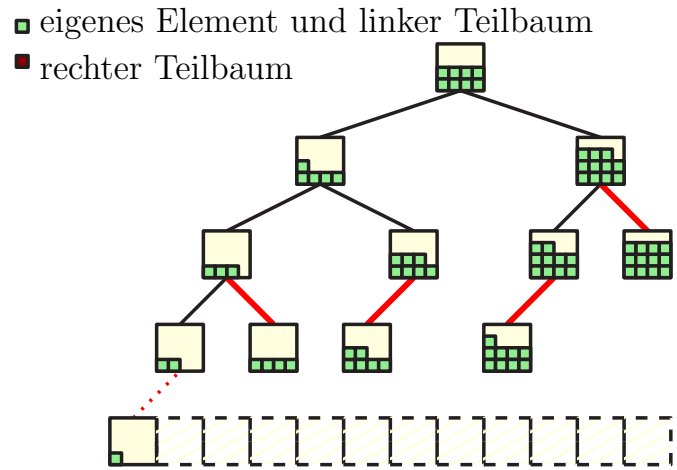
- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

Präfixsumme

Reduktionsbaum



- viele komplexere Algorithmen nutzen Reduktionsschemata in Baumform
- hier: Präfixsumme via Fibonacci-Baum

Aufwärtsphase

- aggregiere Daten aus Teilbäumen
- speichere Summe kleinerer und größerer Elemente getrennt voneinander
- leite Summe aller Elemente an Vorgängerknoten

Abwärtsphase

- gesammelte Daten werden verteilt
- eigene Daten und Daten des linken Teilbaums nur nach rechts

External Memory

Speichermodell

Latenzen

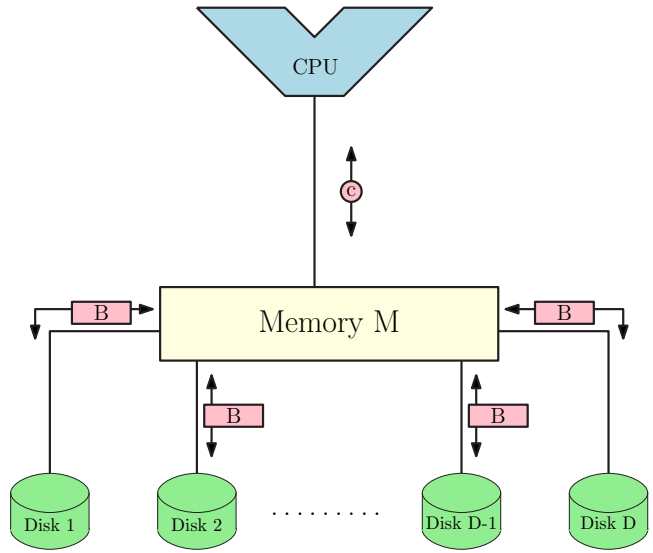
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 µs	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia

Speichermodell

- **External Memory Model**
- Allgemeiner: **Parallel Disk Model**

- Speicherzugriffe in Blöcken
- Blockzugriffe minimieren $\rightarrow$ Datenlokalität

- Muster wiederholt sich in Speicherhierarchie immer wieder

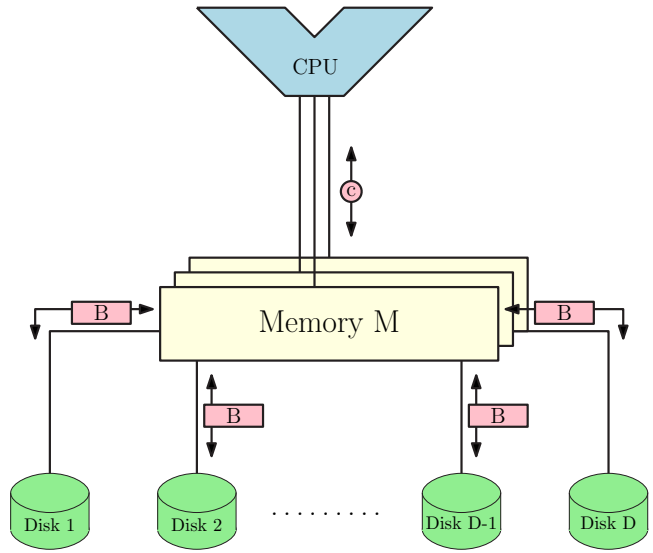


Speichermodell

- **External Memory Model**
- Allgemeiner: **Parallel Disk Model**

- Speicherzugriffe in Blöcken
- Blockzugriffe minimieren $\rightarrow$ Datenlokalität

- Muster wiederholt sich in Speicherhierarchie immer wieder

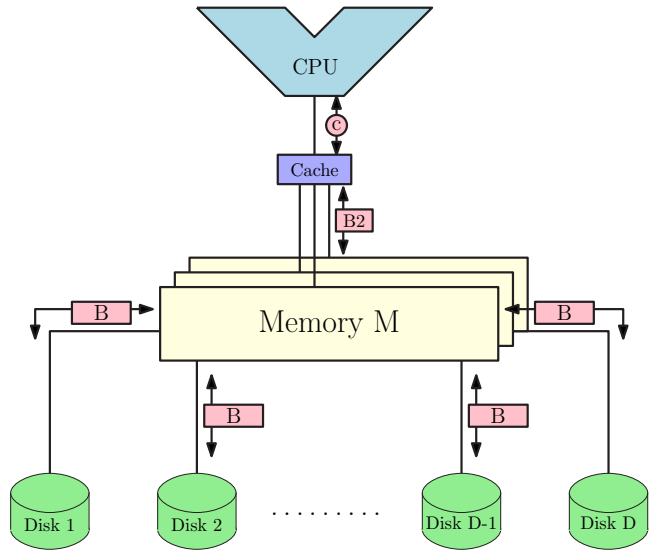


Speichermodell

- **External Memory Model**
- Allgemeiner: **Parallel Disk Model**

- Speicherzugriffe in Blöcken
- Blockzugriffe minimieren $\rightarrow$ Datenlokalität

- Muster wiederholt sich in Speicherhierarchie immer wieder

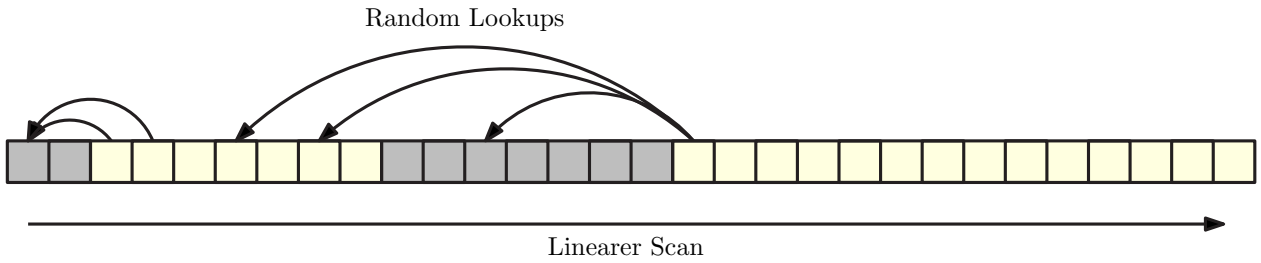


I/O-effizientes Design

- nicht nur relevant bei Disk-I/O
- Beispiel: kürzeste Wege-Bäume auf großen Straßennetzen
  (z.B. Europa)
  - Dijkstra (annähernd linear): $\approx 3 - 5$ Sek.
  - Breitensuche (untere Schranke?): ≈ 2 Sek.

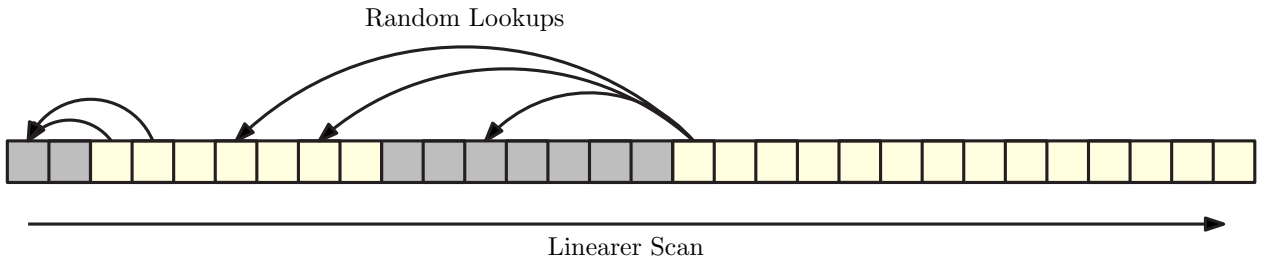
I/O-effizientes Design

- nicht nur relevant bei Disk-I/O
- Beispiel: kürzeste Wege-Bäume auf großen Straßennetzen
  (z.B. Europa)
  - Dijkstra (annähernd linear): $\approx 3 - 5$ Sek.
  - Breitensuche (untere Schranke?): ≈ 2 Sek.
  - PHAST (linearer Scan): < 0.2 Sek.



I/O-effizientes Design

- nicht nur relevant bei Disk-I/O
- Beispiel: kürzeste Wege-Bäume auf großen Straßennetzen
  (z.B. Europa)
  - Dijkstra (annähernd linear): $\approx 3 - 5$ Sek.
  - Breitensuche (untere Schranke?): ≈ 2 Sek.
  - PHAST (linearer Scan): < 0.2 Sek.



- Strukturierter Zugriff als wichtiges Designprinzip

I/O-effizientes Design

Grundlegende Techniken

- Zugriffsmuster
  - Random Access erwartet $\mathcal{O}(n)$ I/Os
  - Linearer Scan $\mathcal{O}(n/B)$ I/Os
- Stack / Queue
- Sortieren
  - $\mathcal{O}\left(\frac{2n}{B}\left(1+\lceil\log_{M/B}\frac{n}{M}\rceil\right)\right)$ I/Os
  - oft vorbereitend für linearen Scan
- Prioritätswarteschlangen
  - $\approx \text{sort}(n)$ I/Os
  - nutzbar als Warteliste: „Speicherzugriff auf später verschieben“
- Externe Suchbäume

I/O-effizientes Design

Grundlegende Techniken

- Zugriffsmuster
  - Random Access erwartet $\mathcal{O}(n)$ I/Os
  - Linearer Scan $\mathcal{O}(n/B)$ I/Os
- Stack / Queue
- Sortieren
  - $\mathcal{O}\left(\frac{2n}{B}\left(1+\lceil\log_{M/B}\frac{n}{M}\rceil\right)\right)$ I/Os
  - oft vorbereitend für linearen Scan
- Prioritätswarteschlangen
  - $\approx \text{sort}(n)$ I/Os
  - nutzbar als Warteliste: „Speicherzugriff auf später verschieben“
- Externe Suchbäume

I/O-effizientes Design

Grundlegende Techniken

- Zugriffsmuster
  - Random Access erwartet $\mathcal{O}(n)$ I/Os
  - Linearer Scan $\mathcal{O}(n/B)$ I/Os
- Stack / Queue
- Sortieren
  - $\mathcal{O}\left(\frac{2n}{B}\left(1+\lceil\log_{M/B}\frac{n}{M}\rceil\right)\right)$ I/Os
  - oft vorbereitend für linearen Scan
- Prioritätswarteschlangen
  - $\approx \text{sort}(n)$ I/Os
  - nutzbar als Warteliste: „Speicherzugriff auf später verschieben“
- Externe Suchbäume

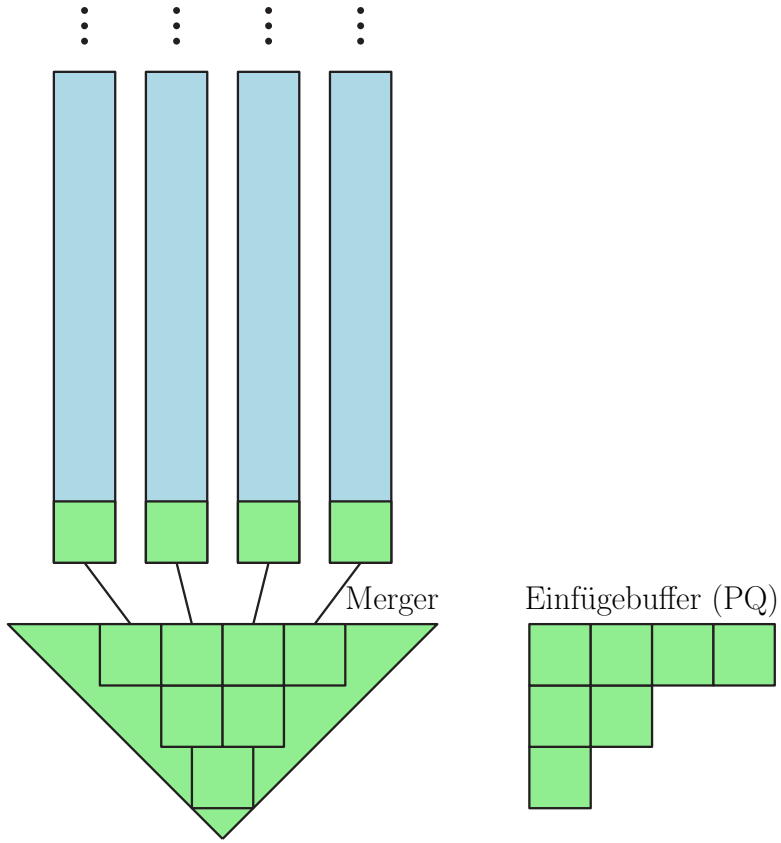
I/O-effizientes Design

Grundlegende Techniken

- Zugriffsmuster
  - Random Access erwartet $\mathcal{O}(n)$ I/Os
  - Linearer Scan $\mathcal{O}(n/B)$ I/Os
- Stack / Queue
- Sortieren
  - $\mathcal{O}\left(\frac{2n}{B}\left(1+\lceil\log_{M/B}\frac{n}{M}\rceil\right)\right)$ I/Os
  - oft vorbereitend für linearen Scan
- Prioritätswarteschlangen
  - $\approx \text{sort}(n)$ I/Os
  - nutzbar als Warteliste: „Speicherzugriff auf später verschieben“
- Externe Suchbäume

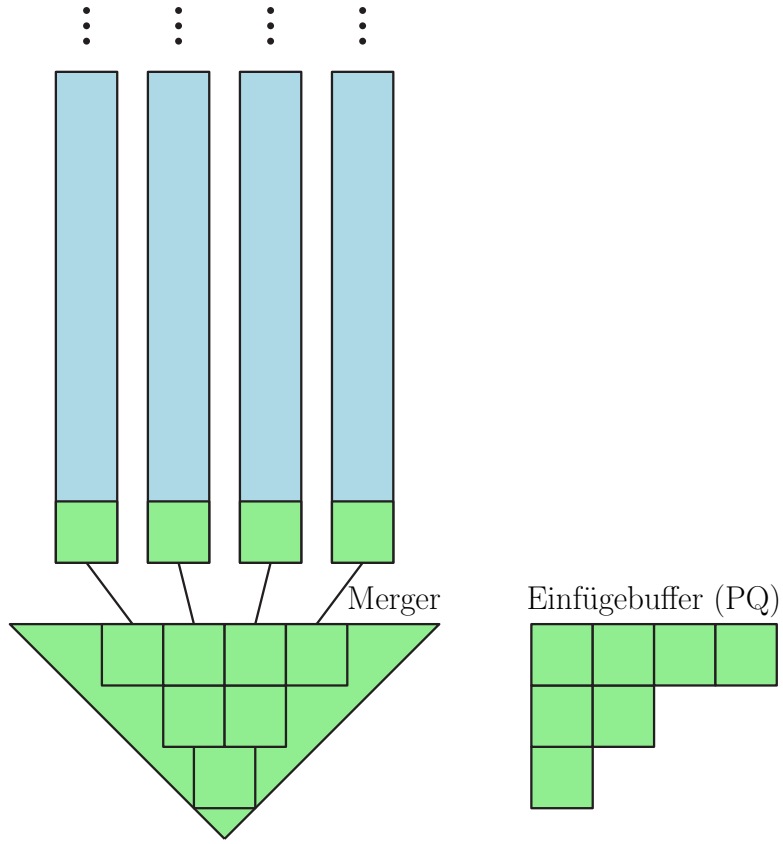
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



Externe Priority Queue

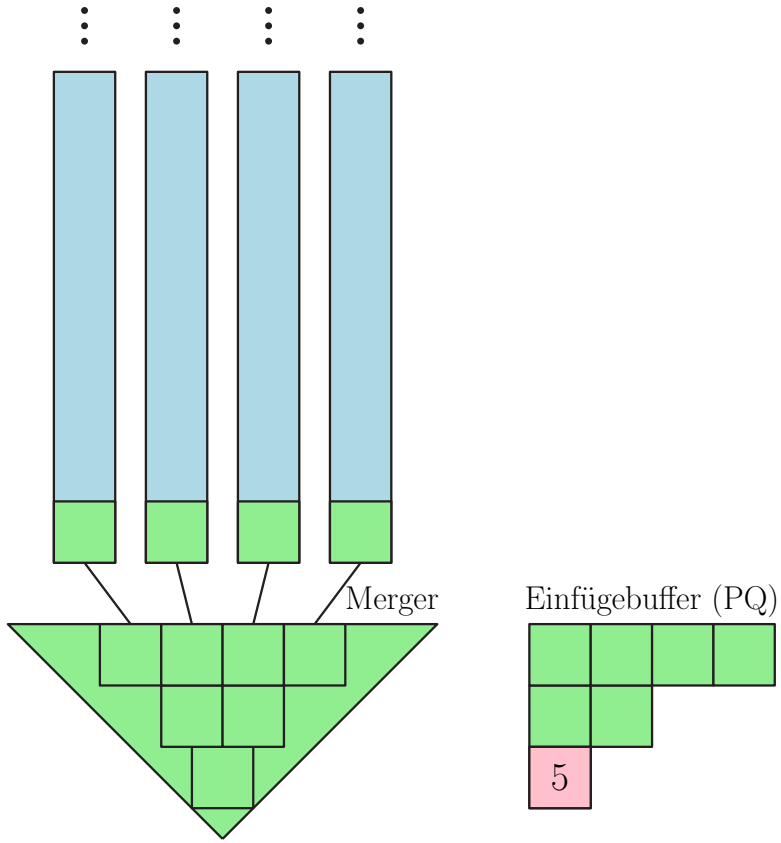
- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



5

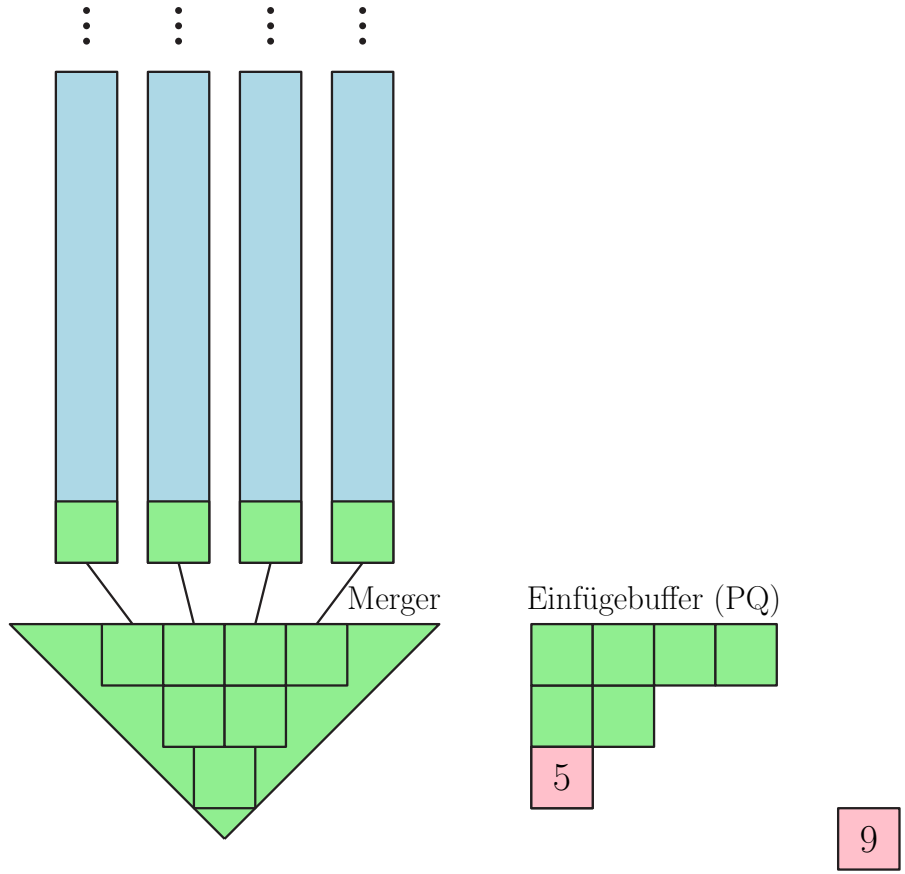
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



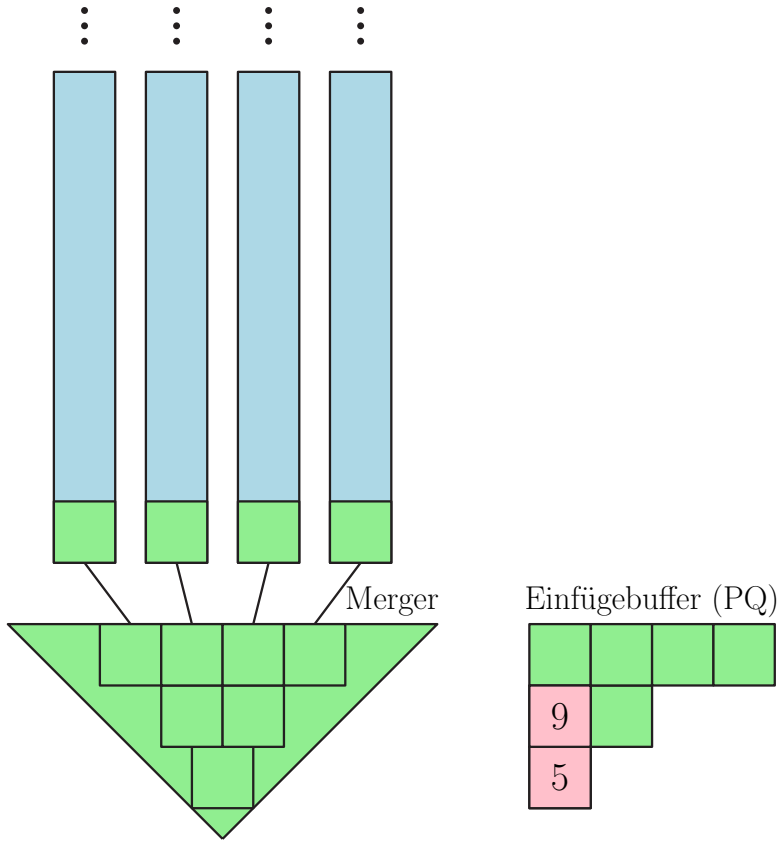
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



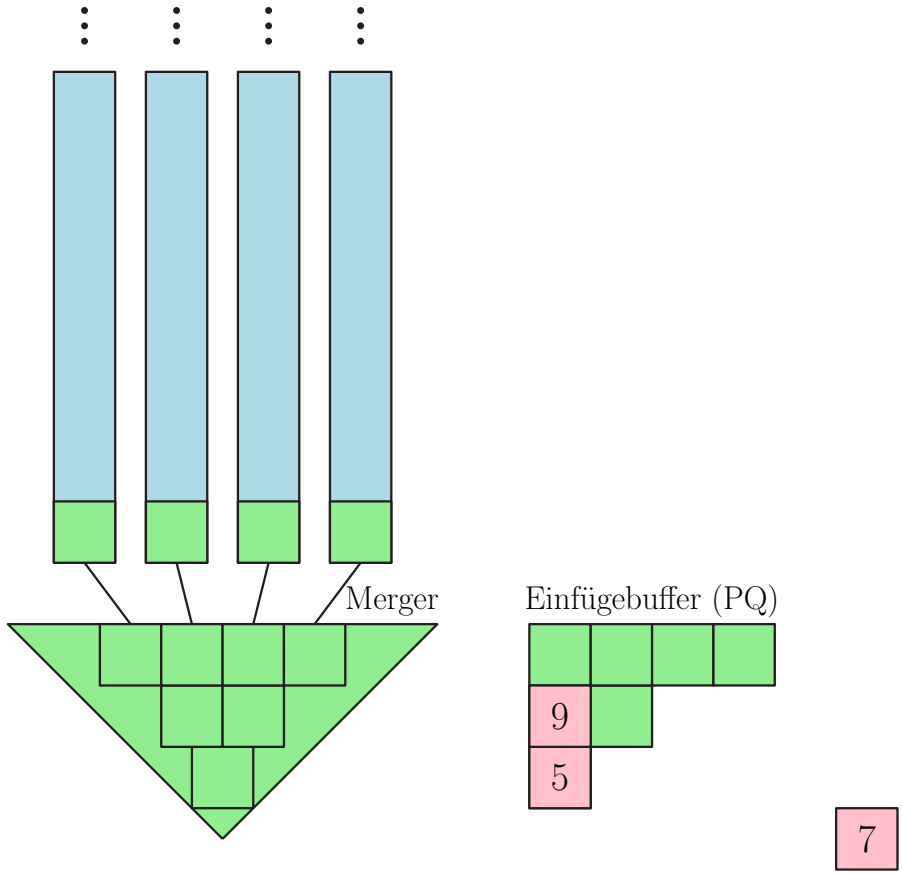
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



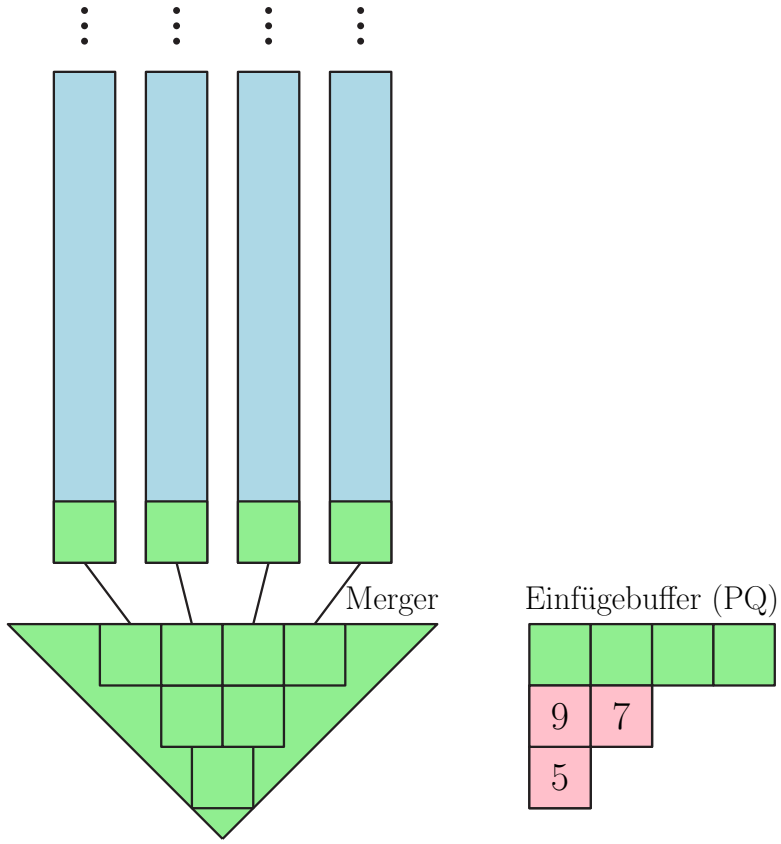
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



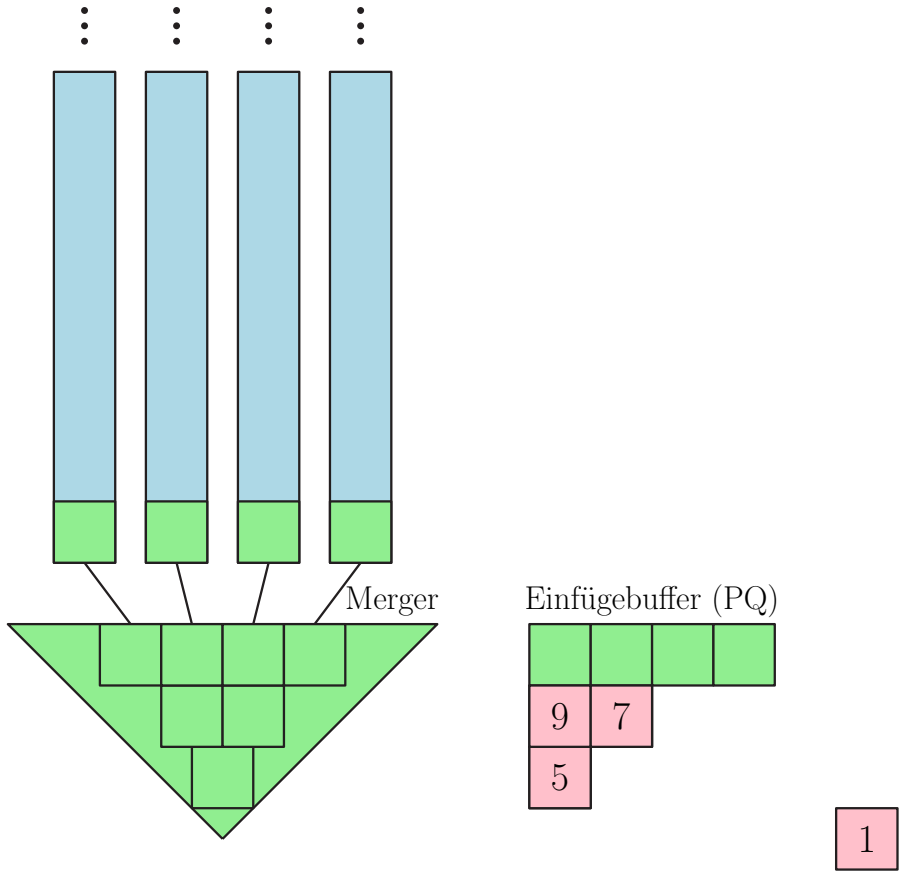
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



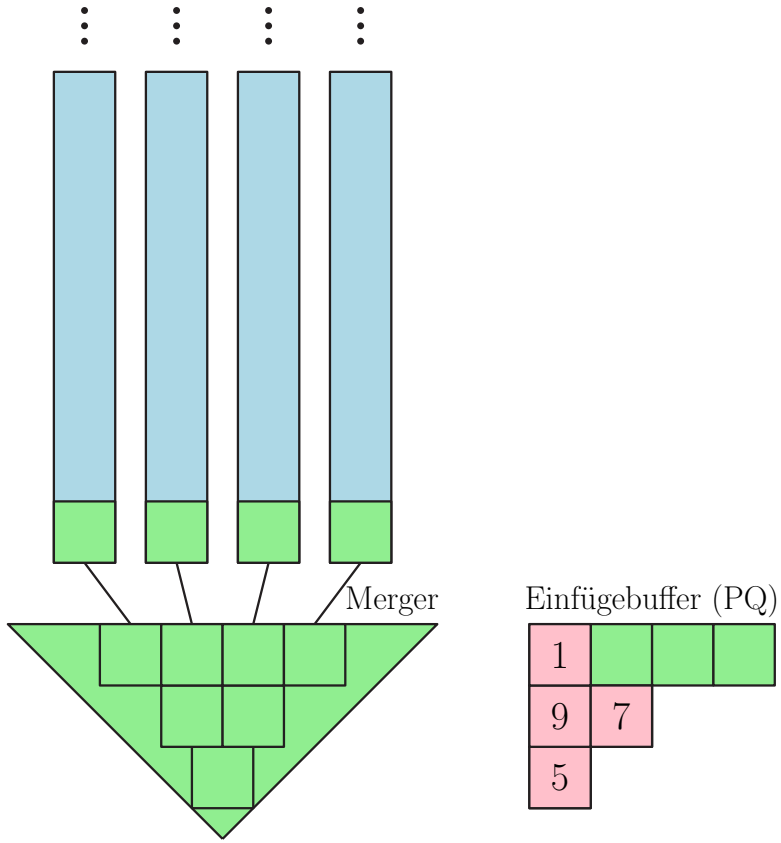
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



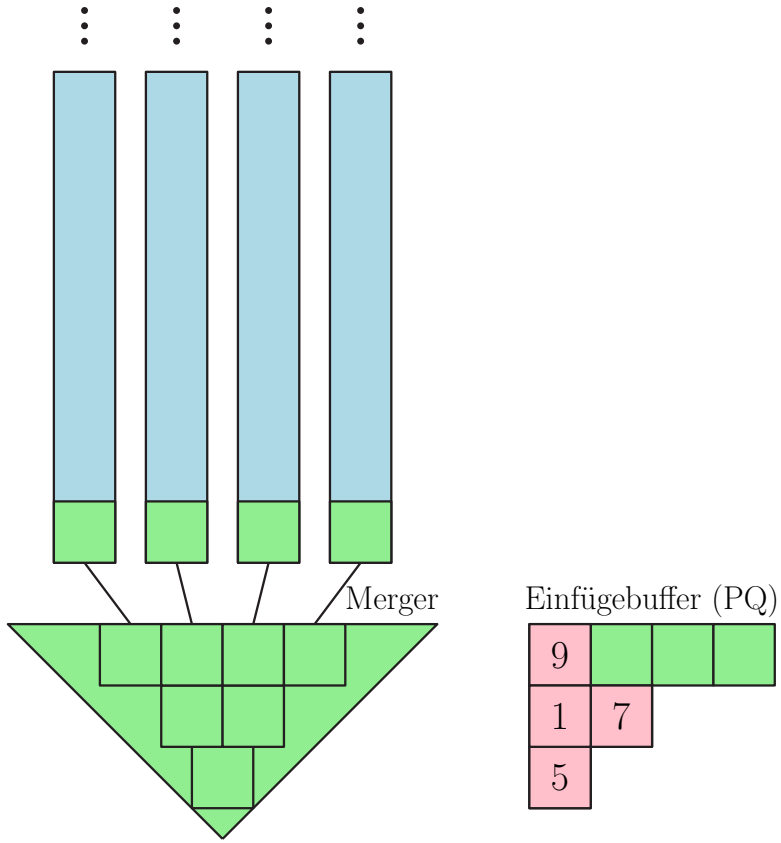
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



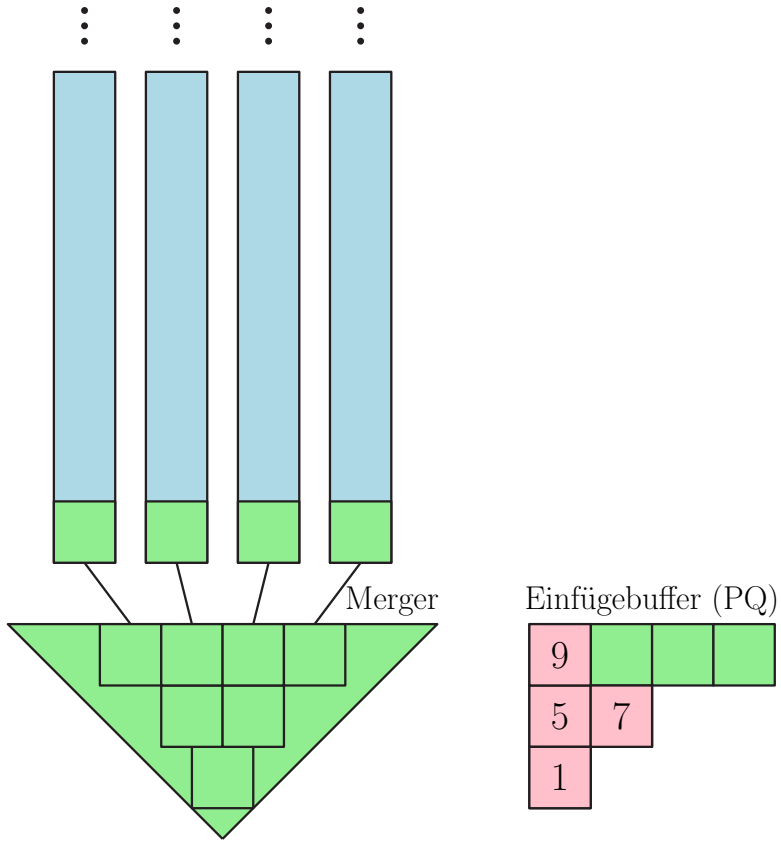
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



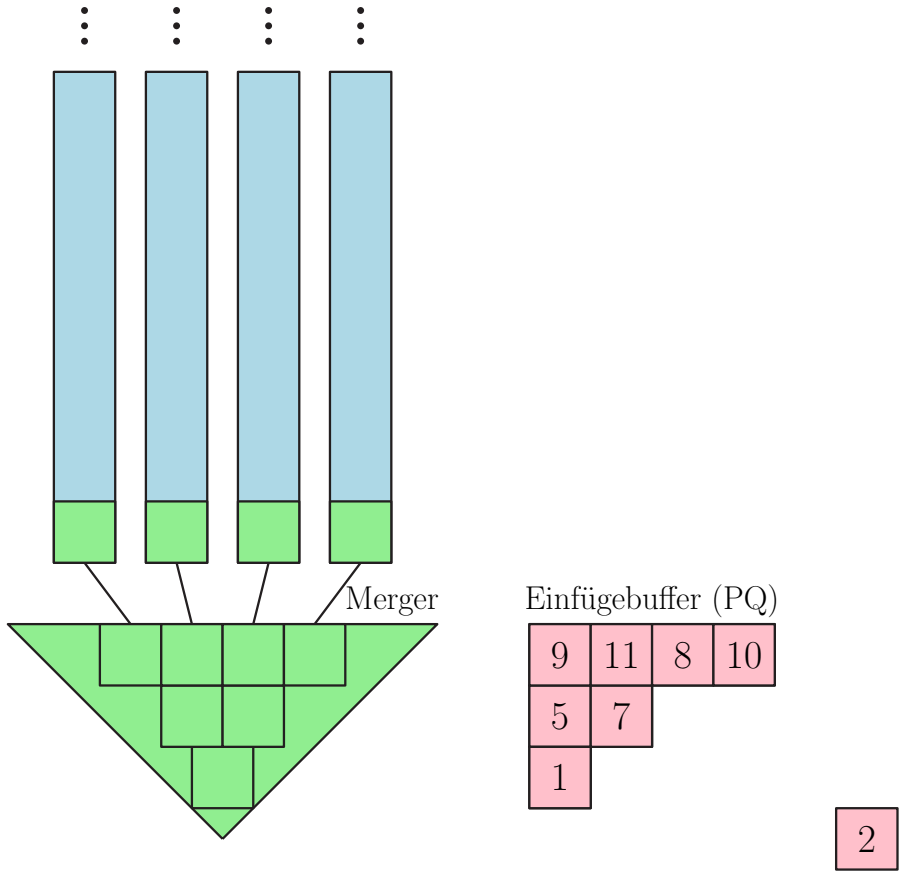
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



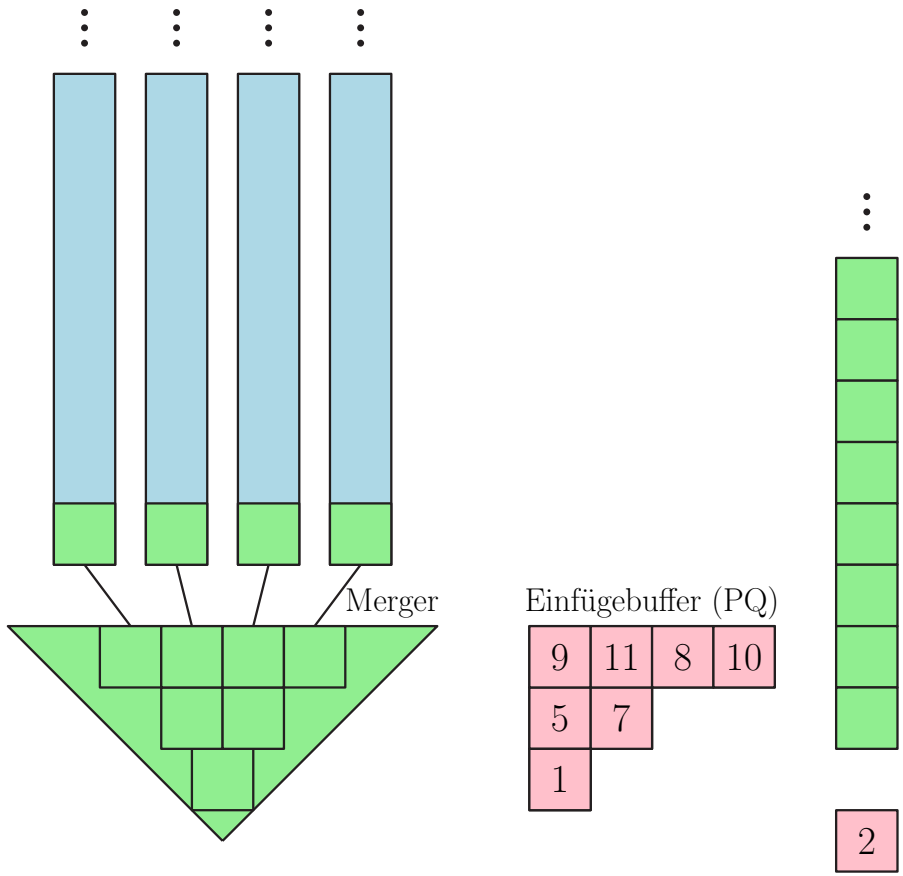
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



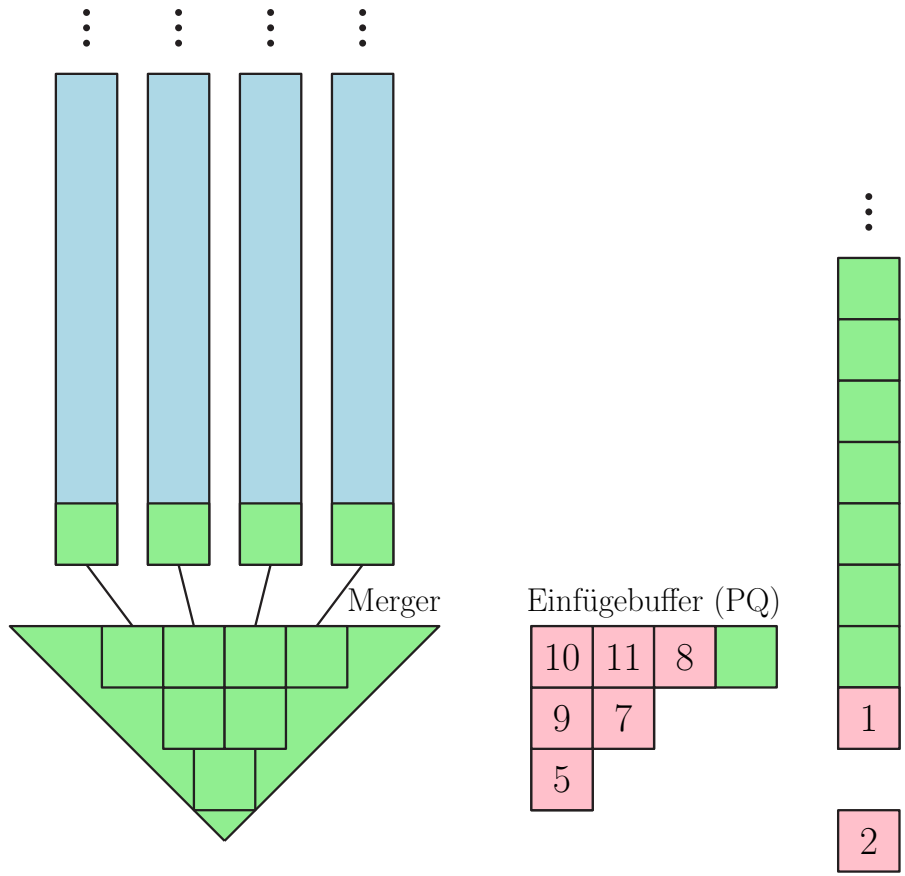
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



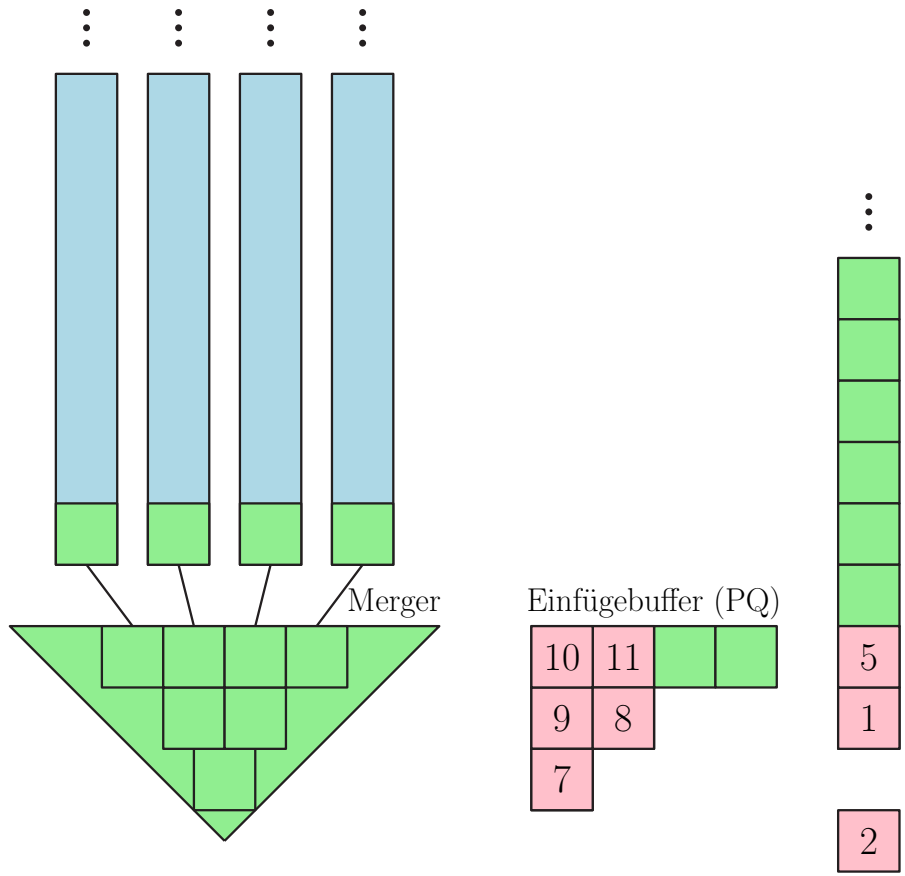
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



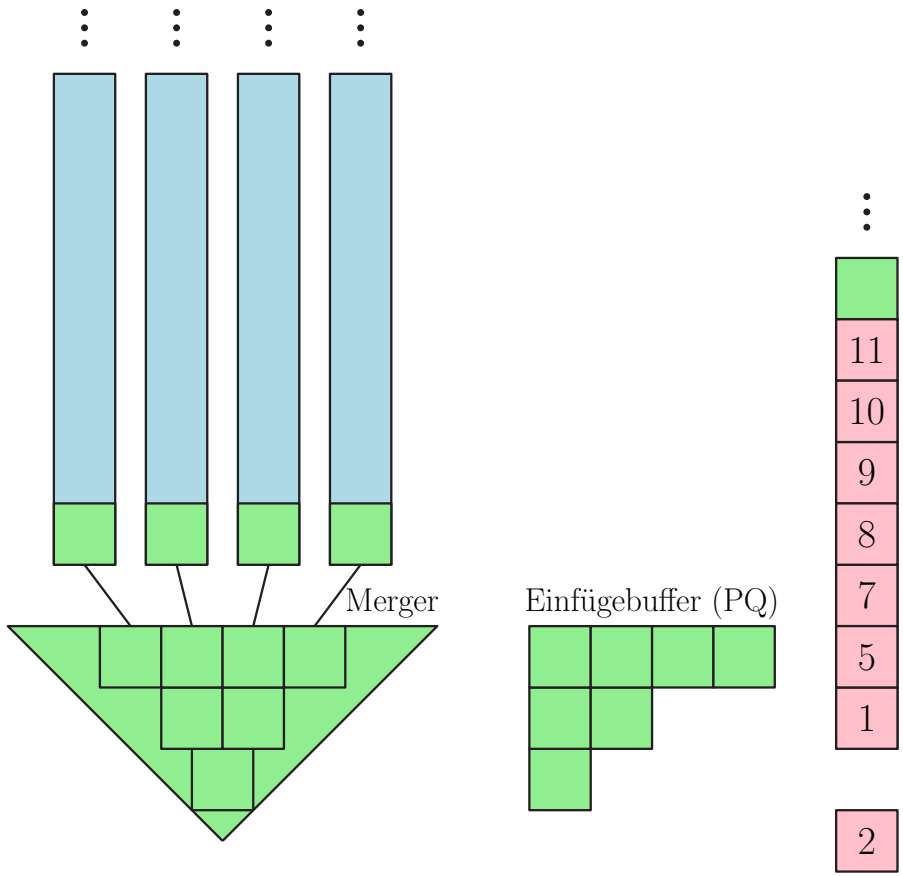
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



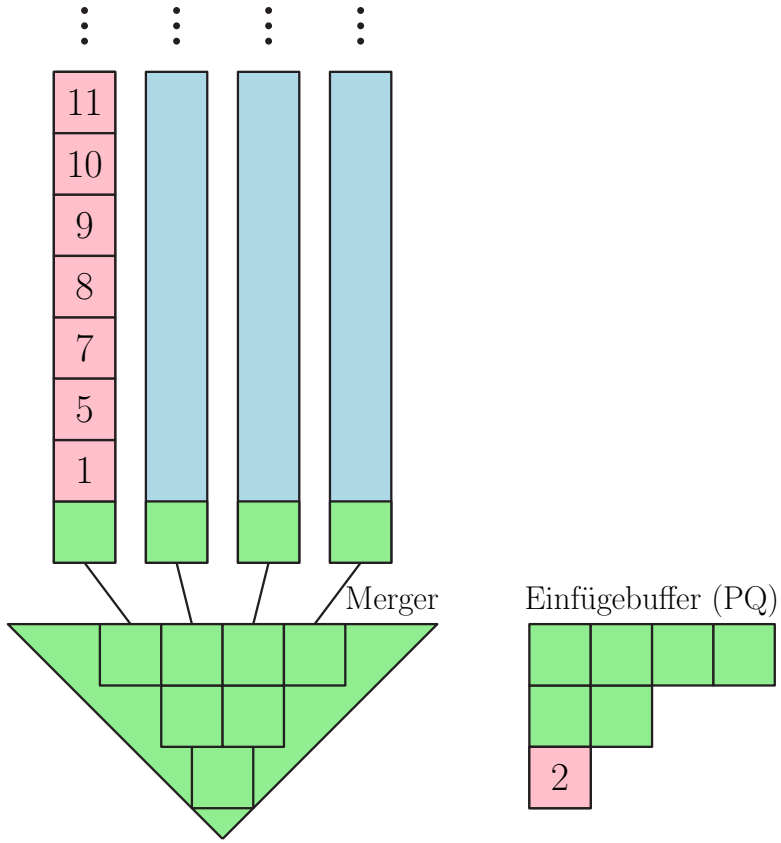
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



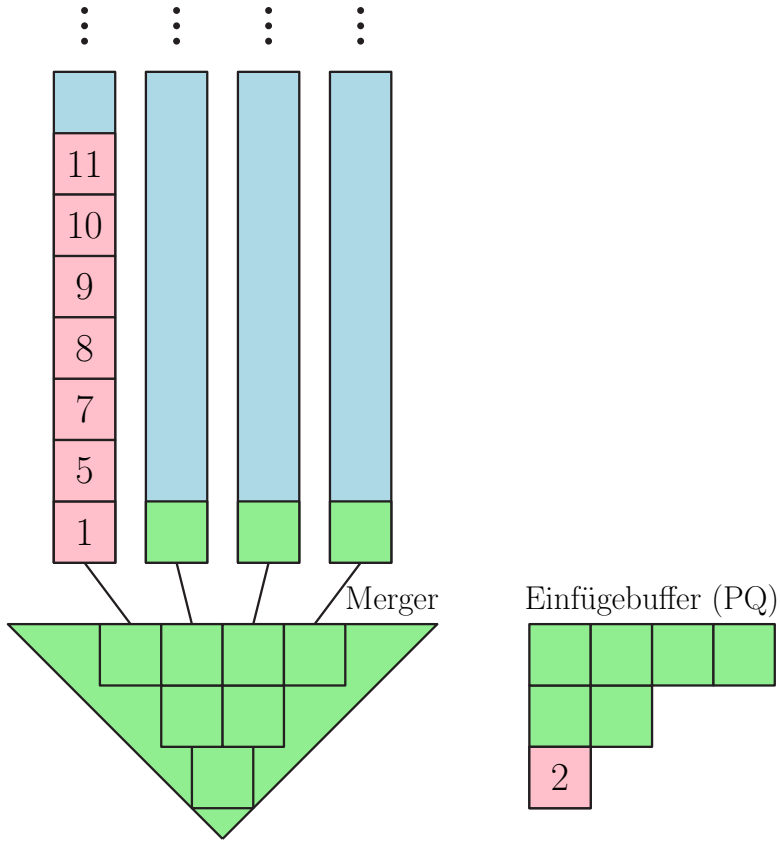
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



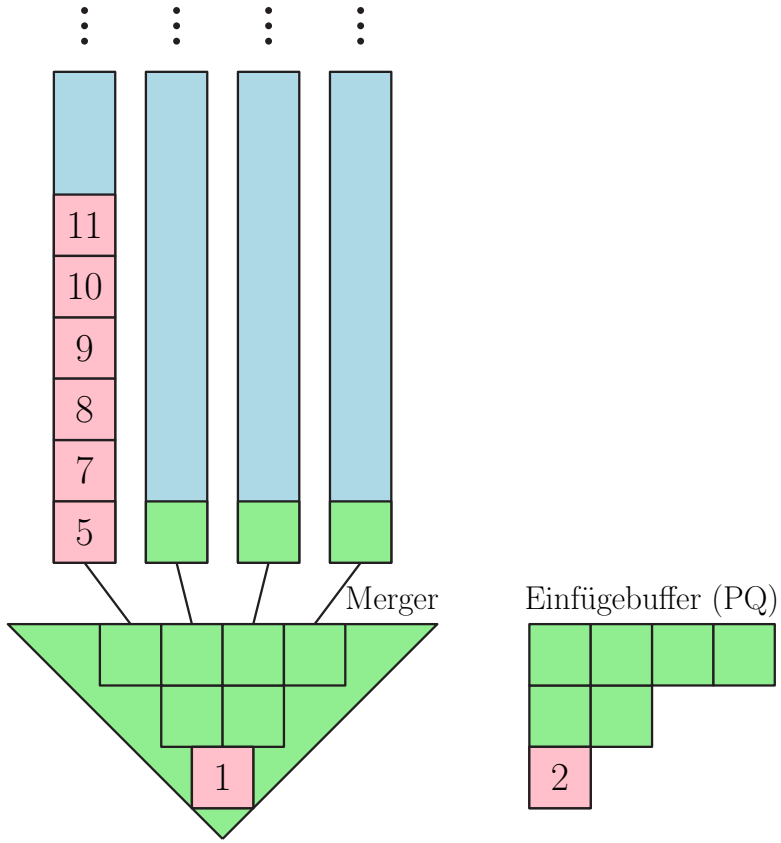
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



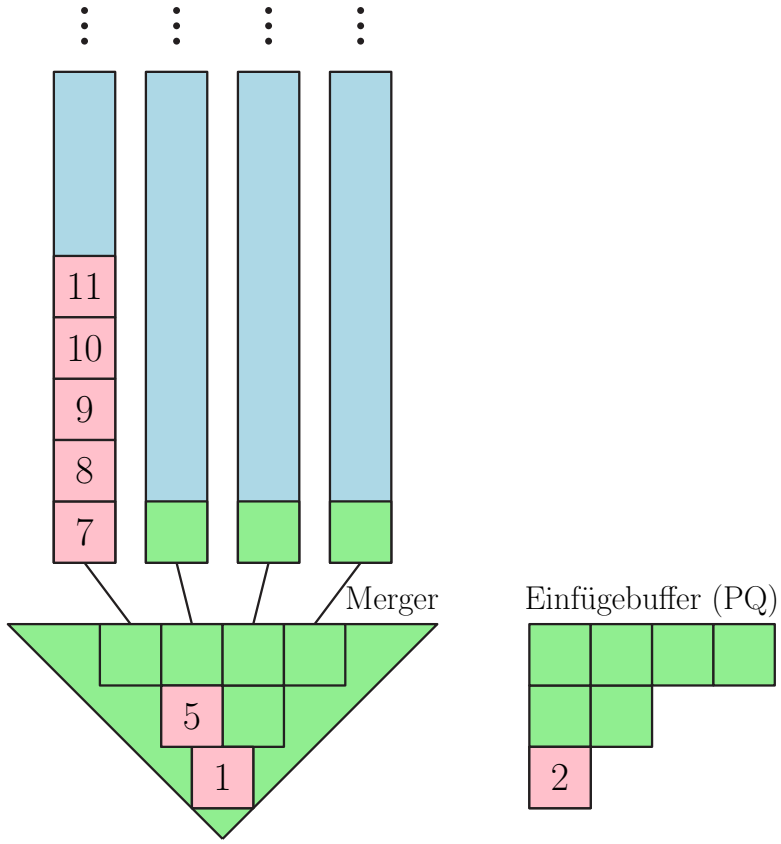
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



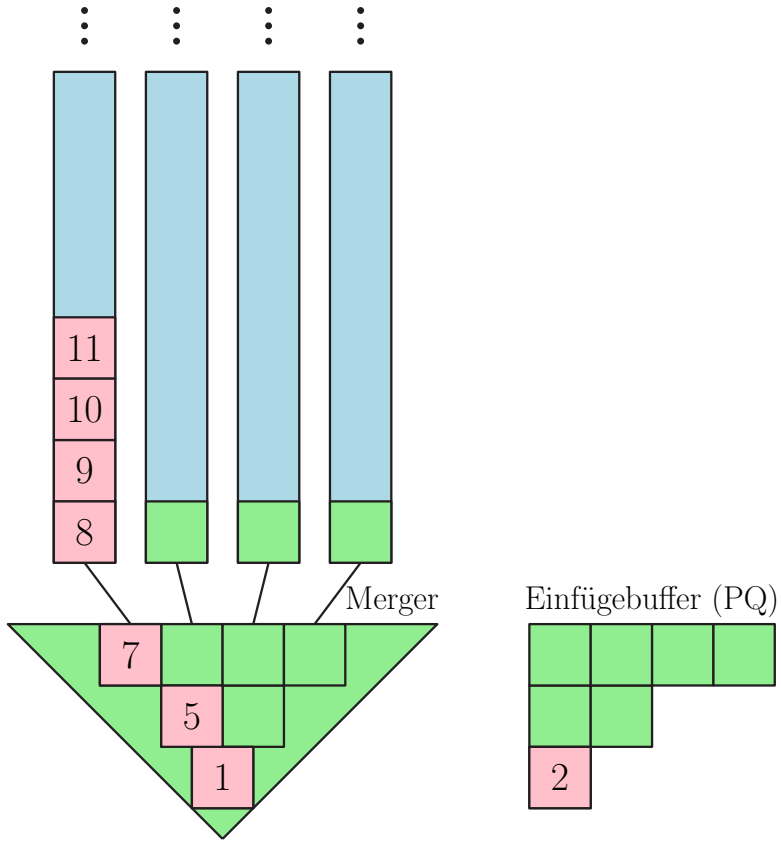
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



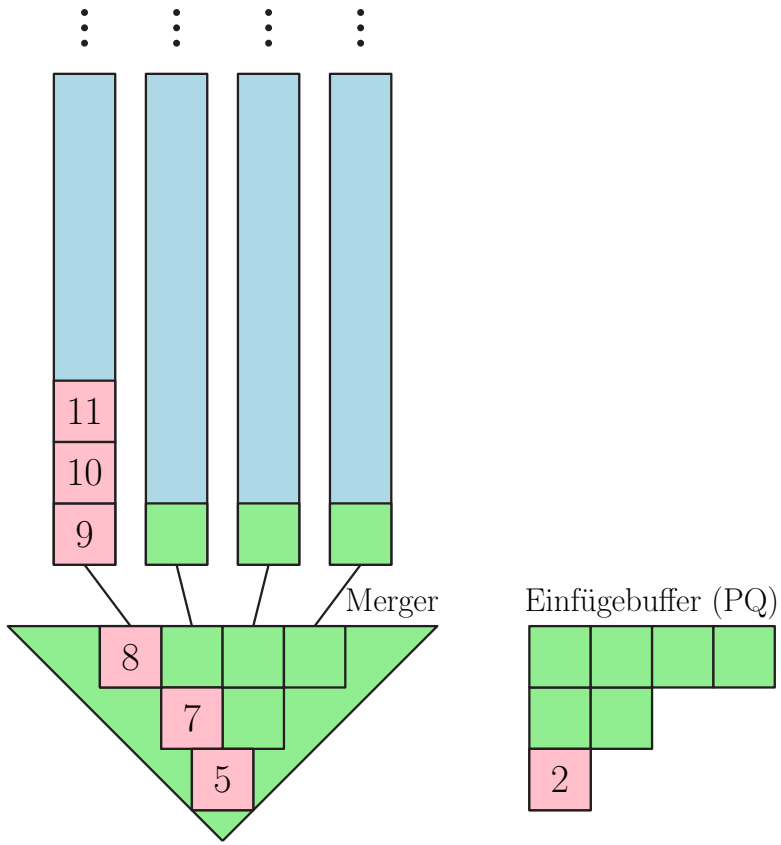
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



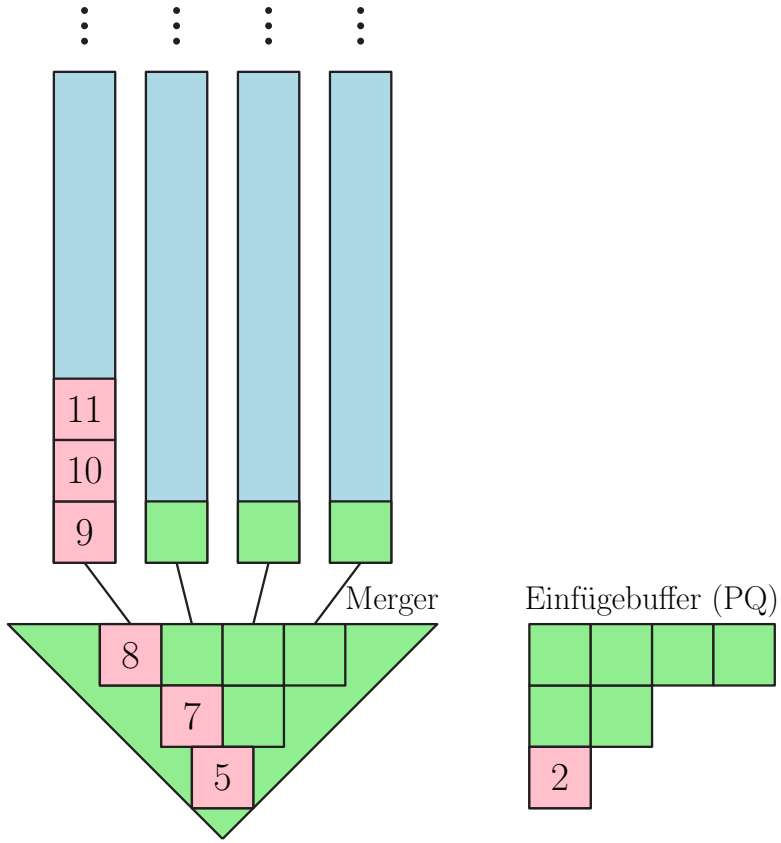
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



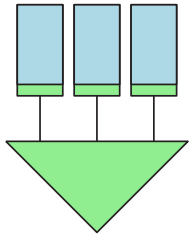
Externe Priority Queue

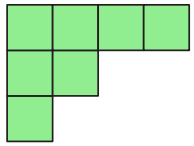
- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



Externe Priority Queue

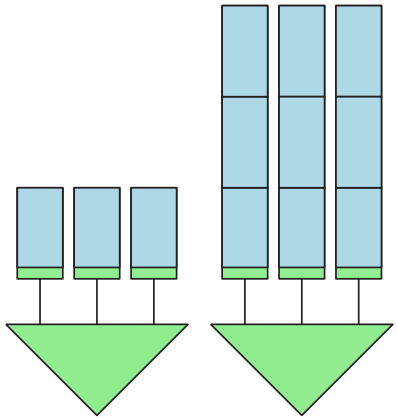
für sehr große Datenmengen

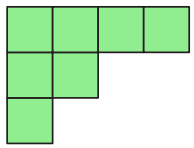




Externe Priority Queue

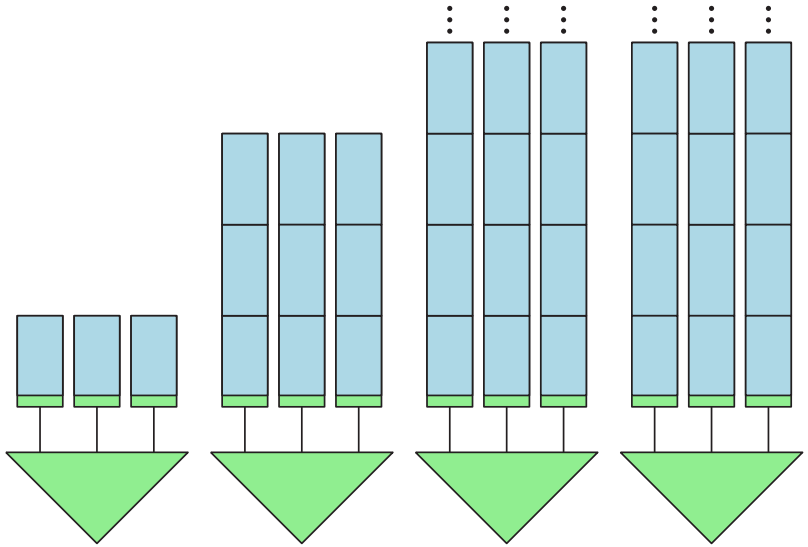
für sehr große Datenmengen

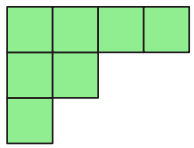




Externe Priority Queue

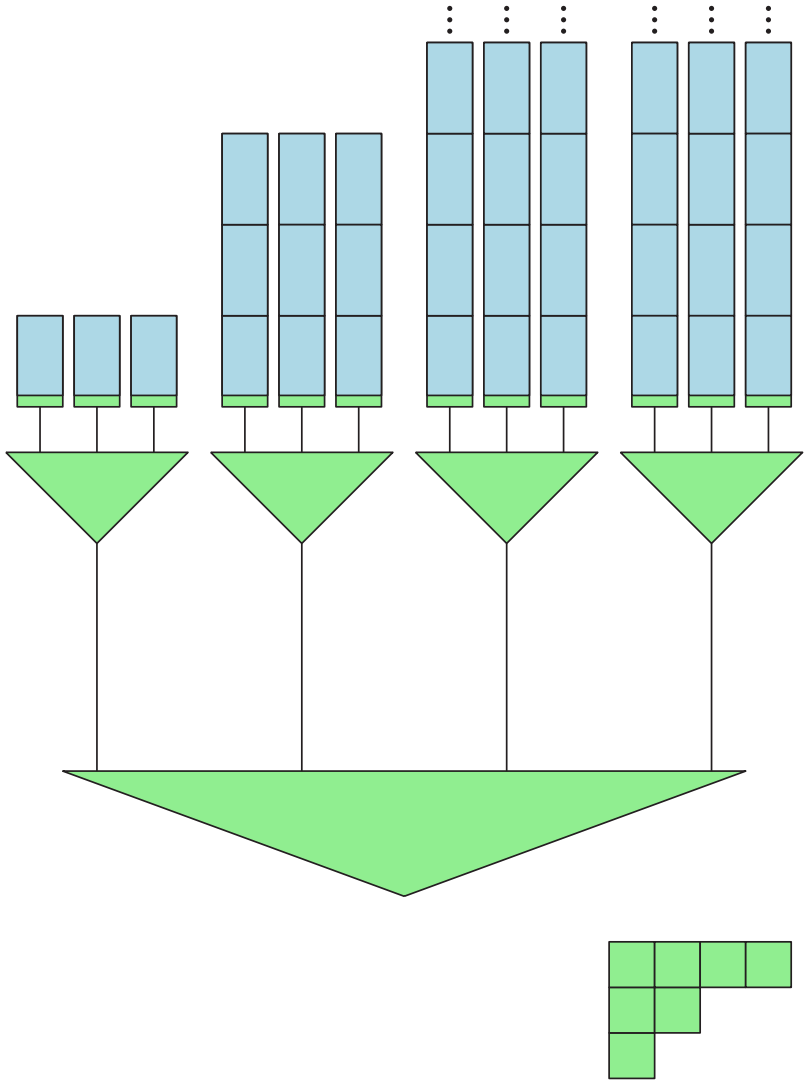
für sehr große Datenmengen





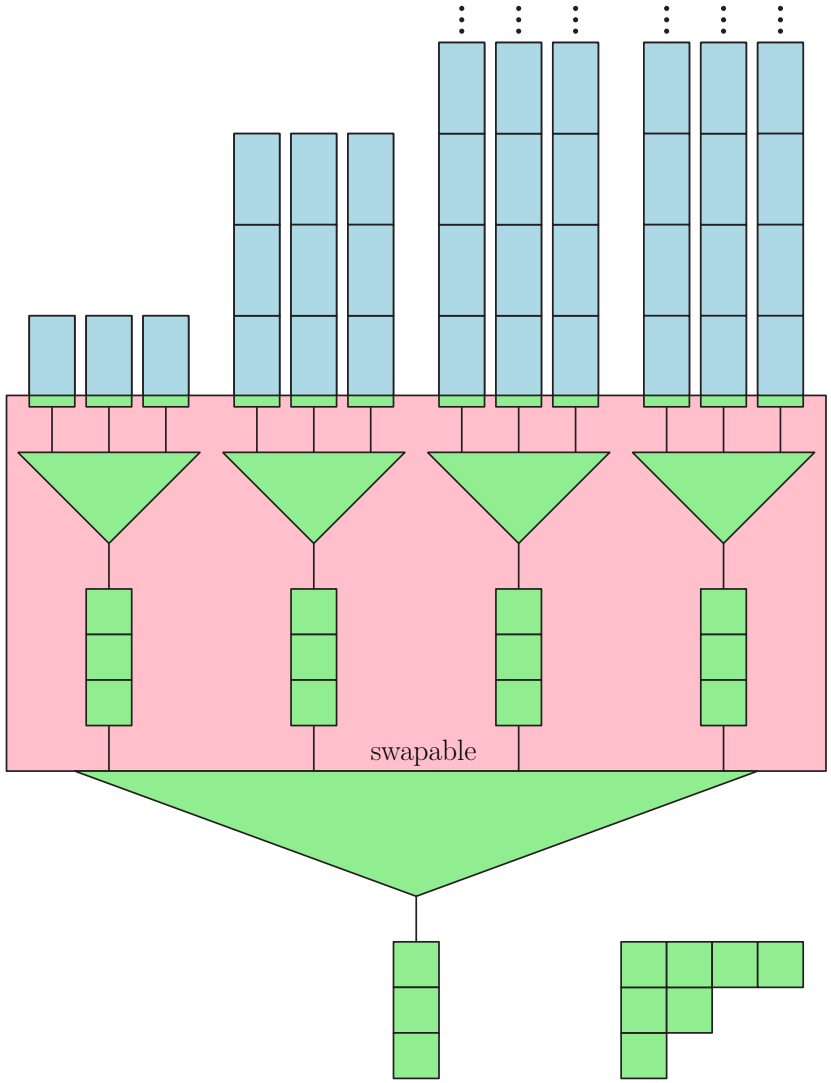
Externe Priority Queue

für sehr große Datenmengen



Externe Priority Queue

für sehr große Datenmengen



Externes Sortieren

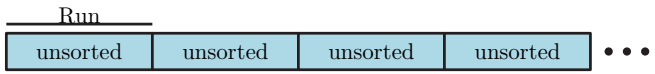
Zwei-Phasen Algorithmus

- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$

Externes Sortieren

Zwei-Phasen Algorithmus

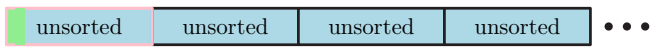
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
    $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

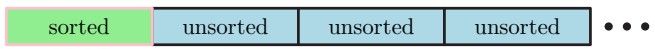
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
    $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$



Externes Sortieren

Zwei-Phasen Algorithmus

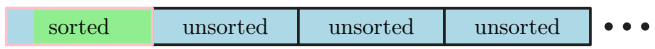
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
    $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$



Externes Sortieren

Zwei-Phasen Algorithmus

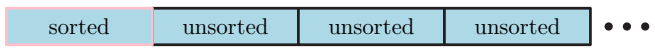
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

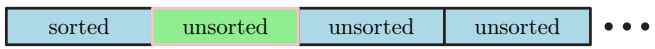
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

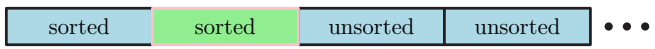
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

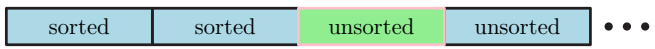
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$



Externes Sortieren

Zwei-Phasen Algorithmus

- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

Externes Sortieren

Zwei-Phasen Algorithmus

- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $$\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$$

Externes Sortieren

Zwei-Phasen Algorithmus

- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$

Externes Sortieren

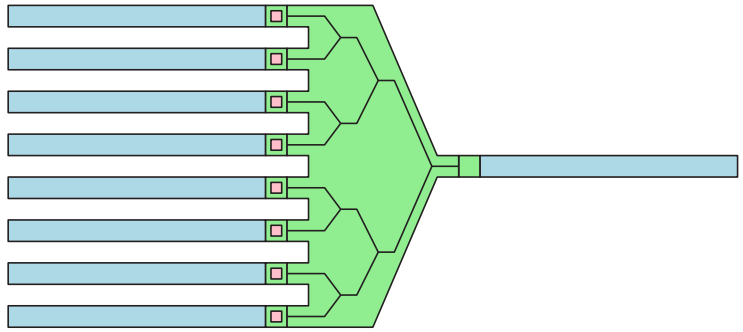
Zwei-Phasen Algorithmus

- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$

Externes Sortieren

Zwei-Phasen Algorithmus

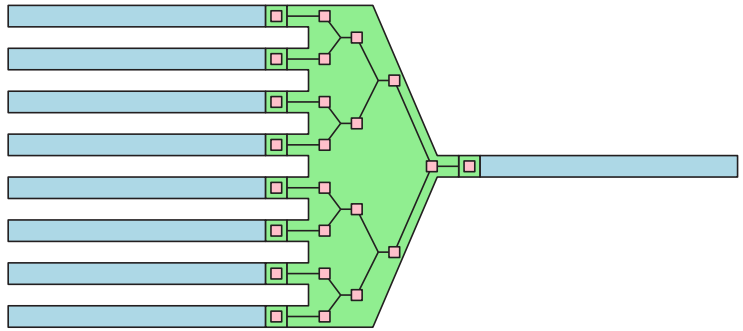
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$



Externes Sortieren

Zwei-Phasen Algorithmus

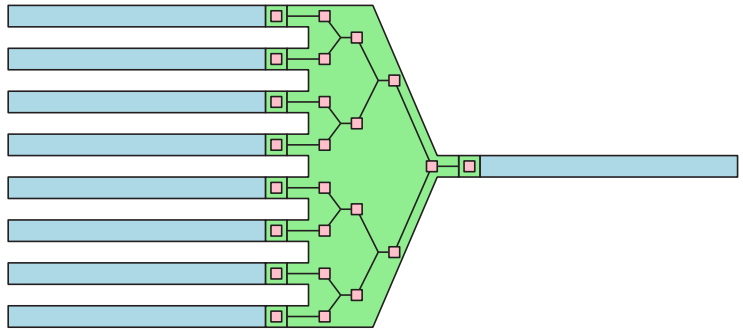
- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$



Externes Sortieren

Zwei-Phasen Algorithmus

- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen
    $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$



Ende!

Schöne Feiertage! 🎄