Präfix-Summe

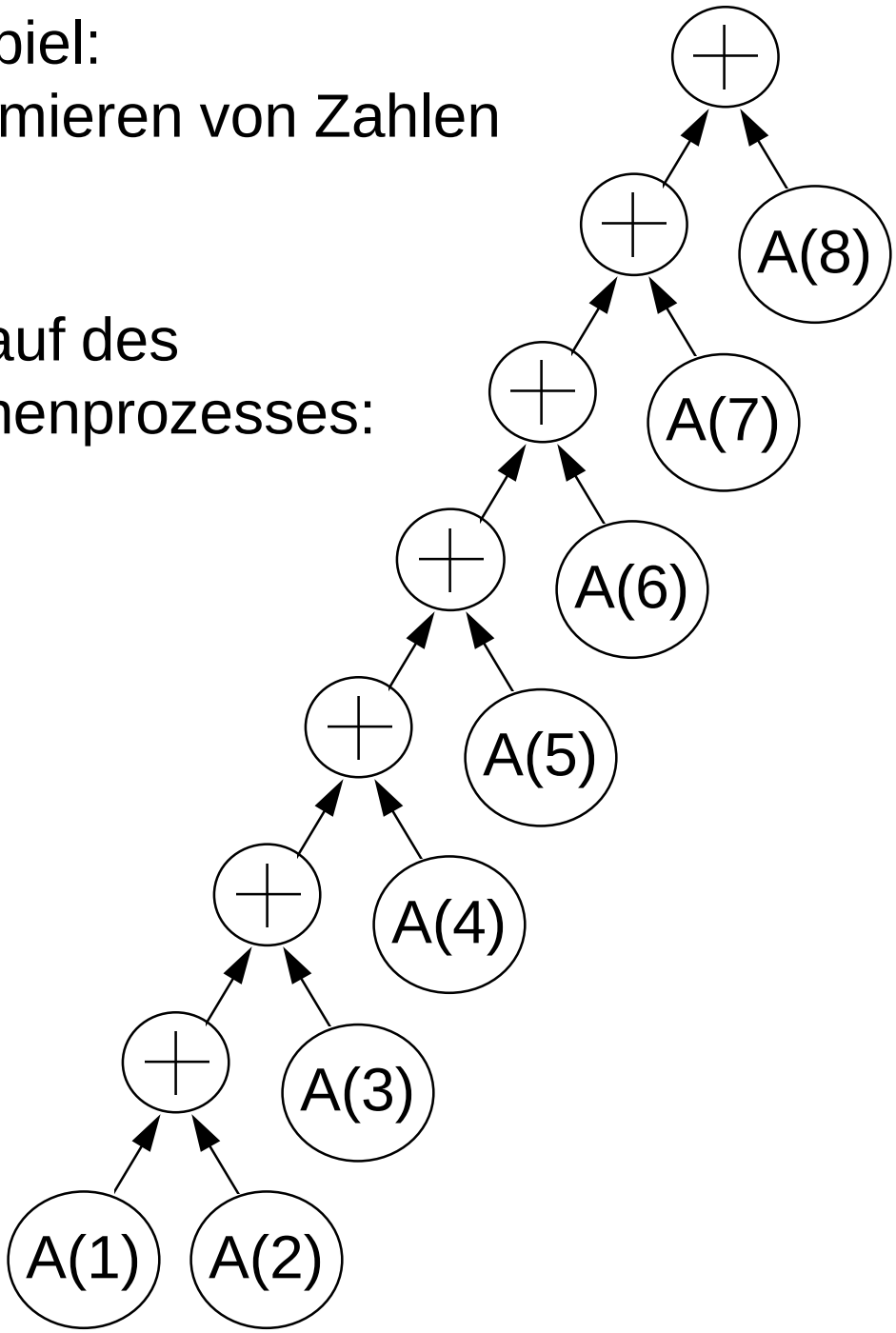
Das Schweizer
Offiziersmesser der
Parallelen Algorithmen

- Parallele Rechenmodelle
- Präfix-Summe
- Brents Lemma
- Anwendungen

Parallele Rechenmodelle

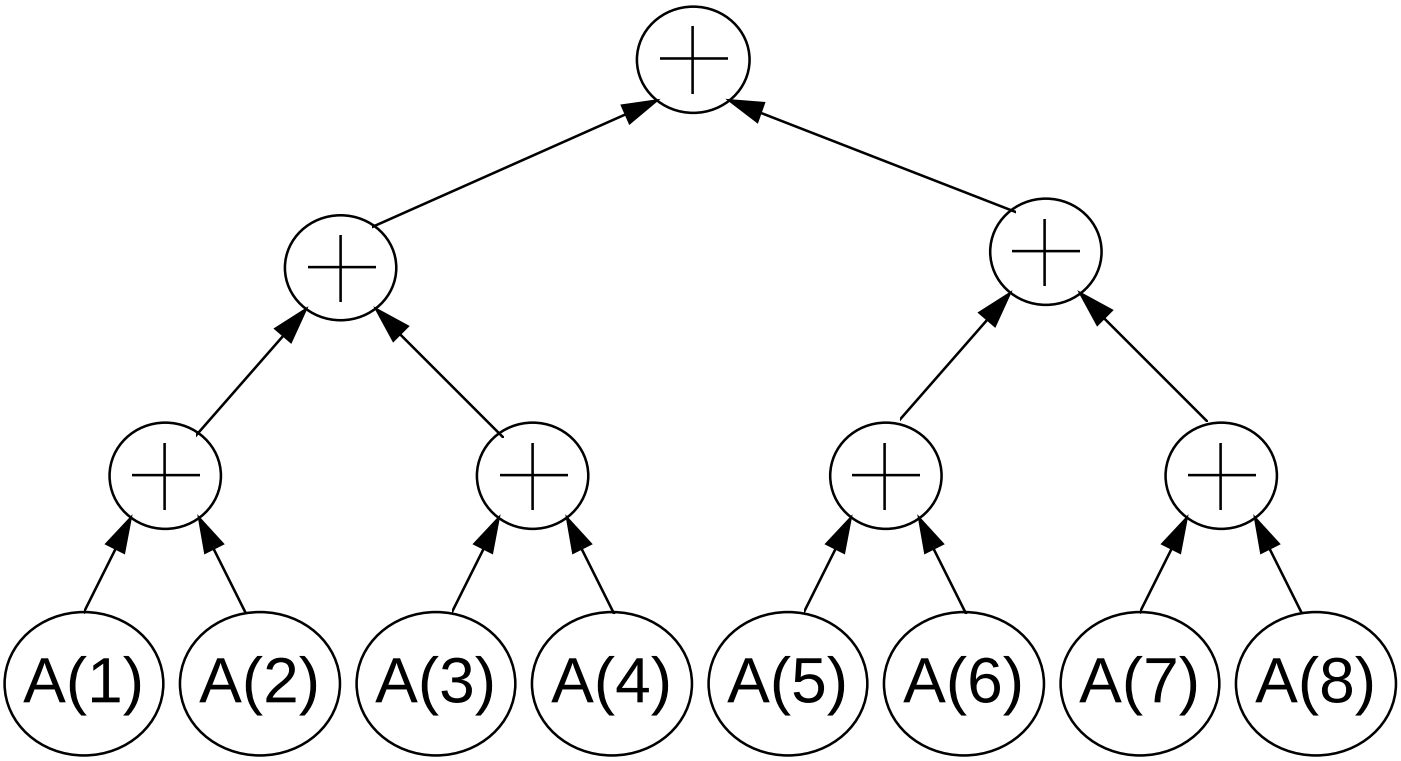
Beispiel:
Summieren von Zahlen

Verlauf des Rechenprozesses:

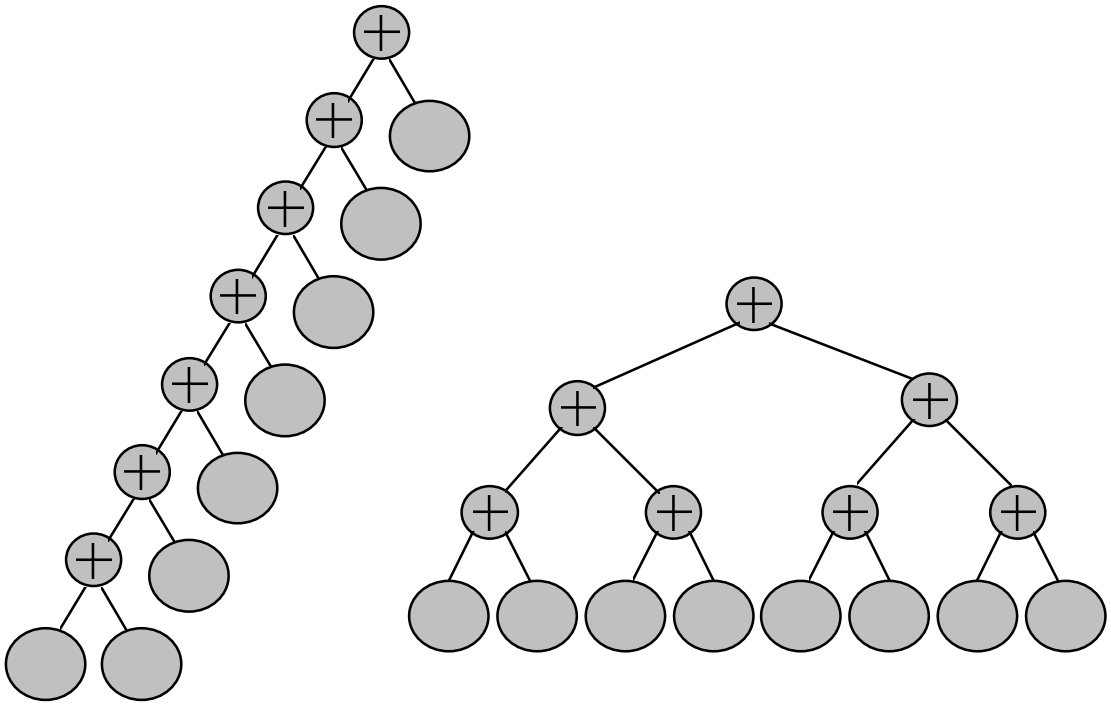


Parallele Rechenmodelle

Alternativer Verlauf:



Was ist besser?



Prozessorzahl:

1	n / 2

Laufzeit:

n - 1	log n

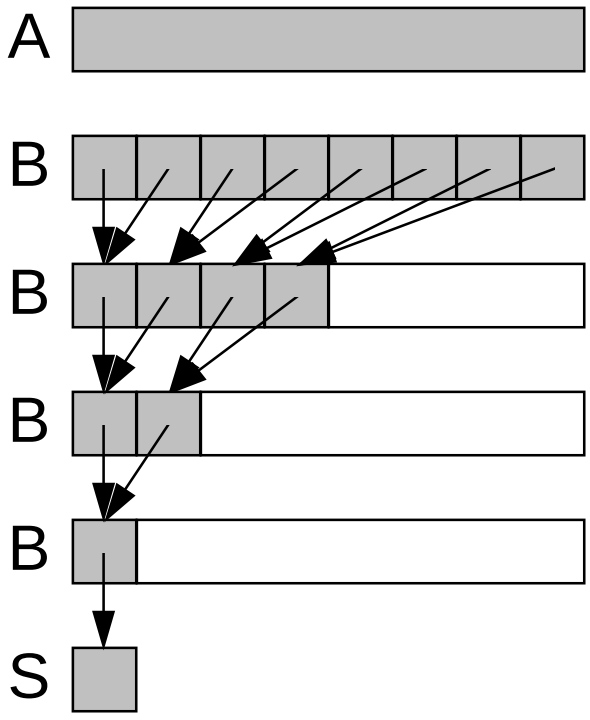
Summe in einem Array

Input: Ein Array A der Grösse $n = 2^k$, abgelegt im gemeinsamen Speicher von n Prozessoren.

Output: Die Summe der Einträge von A in der Speicherzelle S.

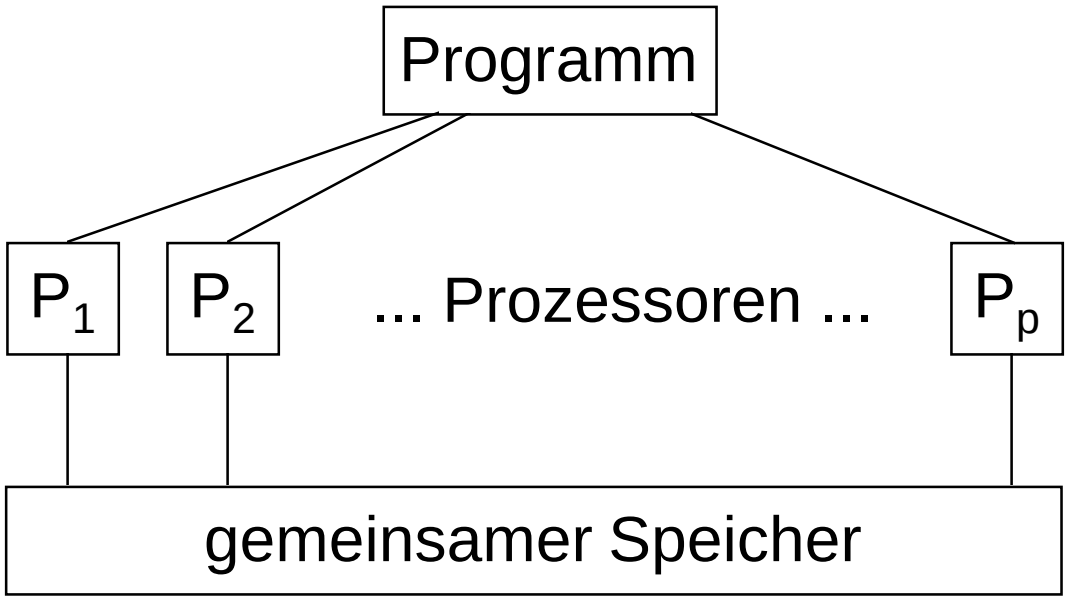
Algorithmus für Prozessor i:

```
begin
    B[i] := A[i];
    for h = 1 to log n do
        if (i <= n/2^h) then
            x := B[2i-1];
            y := B[2i];
            z := x + y;
            B[i] := z;
        end;
    end;
    if i = 1 then S := z end;
end
```



Laufzeit ~ 2 log n

Definition PRAM



Schritt ist
1. Lese-Phase: 1 Speicherwort
2. Rechen-Phase: konstant viele Operationen im lokalen Speicher
3. Schreib-Phase: 1 Speicherwort

Beispiel: Prozessor i führt aus: a[i] := a[a[i]]

Effizienz von PRAM Algorithmen

n	Problemgrösse
$p(n)$	Prozessorzahl
$t_p(n)$	Laufzeit
$w_p(n) = t_p(n) \cdot p(n)$	Arbeit
$\frac{\text{Arbeit sequentiell}}{\text{Arbeit parallel}}$	Effizienz
$\frac{\text{Laufzeit sequentiell}}{\text{Laufzeit parallel}}$	Speedup

Effizienz von PRAM Algorithmen

Ein Algorithmus ist

- optimal, falls $t_p(n) = \text{polylog}(n)$
  und $w_p(n) = O(t_1(n))$

- effizient, falls $t_p(n) = \text{polylog}(n)$
  und $w_p(n) = O(t_1(n))\ \text{polylog}(n)$

$t_1(n)$ ist die Laufzeit des schnellsten sequentiellen Algorithmus.

Wie schnell kann eine parallele Rechnung sein?

Input: Ein Array x[1..n] mit n Zahlen.

Output: Variable min

p(n) = n^2 Prozessoren P_{ij} (i=1,...,n, j=1,...,n)

Algorithmus:

1. forall i P_{i1}: h[i] := 1;
2. forall i,j P_{ij}: if x[i] < x[j] then h[j] := 0 end;
3. forall i,j P_{ij}: if (h[i] = 1) and (i < j) then
   h[j] := 0
   end;
4. forall i P_{i1}: if h[i] = 0 then min := x[i] end;

Beispiel:

x	8	5	2	4	6	7	9	5
h								

Wie schnell kann eine parallele Rechnung sein?

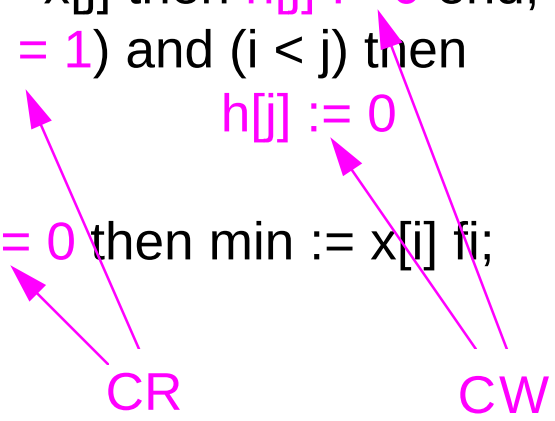
Input: Ein Array x[1..n] mit n Zahlen.

Output: Variable min

p(n) = n^2 Prozessoren P_{ij} (i=1,...,n, j=1,...,n)

Algorithmus:

1. forall i P_{i1}: h[i] := 1;
2. forall i,j P_{ij}: if x[i] < x[j] then h[j] := 0 end;
3. forall i,j P_{ij}: if (h[i] = 1) and (i < j) then
   h[j] := 0
   end;
4. forall i P_{i1}: if h[i] = 0 then min := x[i] fi;



Modell CRCW

Analyse:

p(n) = n^2
$t_p(n) \sim 1$
$w_p(n) \sim n^2$

Effizienz ~ 1/n
Speedup ~ n

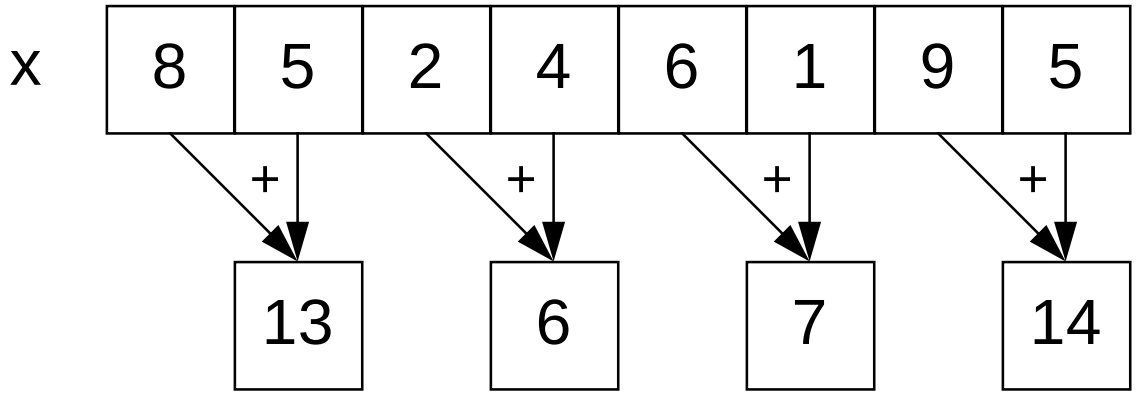
PRAM Modelle

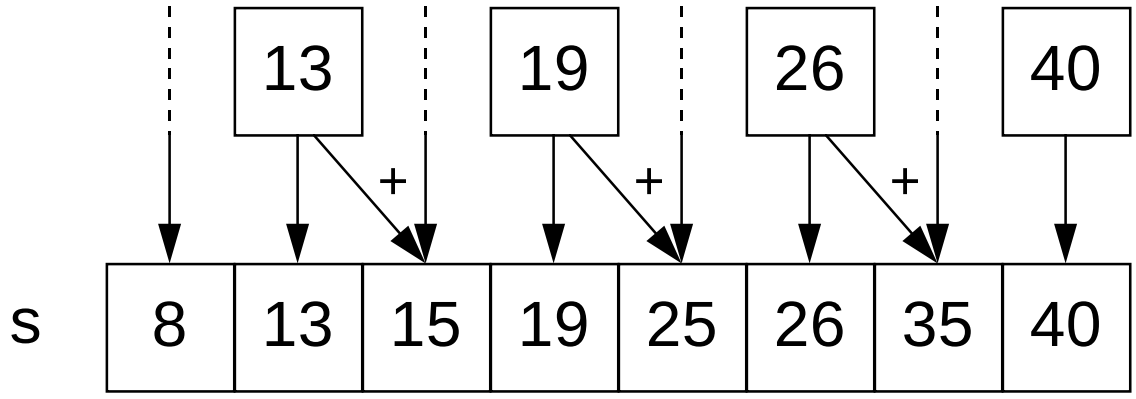
- EREW
- CREW
- CRCW:
  - **weak**: nur gleichzeitig 0 schreiben
  - **common**: nur gleichzeitig denselben Wert schreiben
  - **arbitrary winner**
  - **priority**: grösste Prozessor-Id gewinnt
  - **strong**: grösster Wert gewinnt

Präfix-Summe
Beispiel

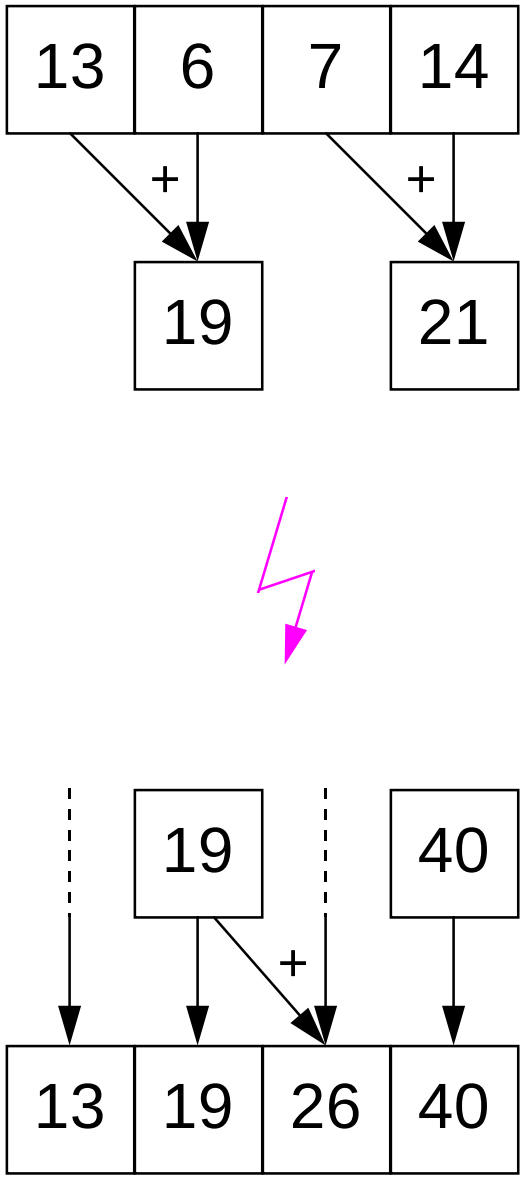
Input: Ein Array ganzer Zahlen x[1..n]

Output: $s_i := x_1 + x_2 + ... + x_i$ (für jedes i=1,...,n)

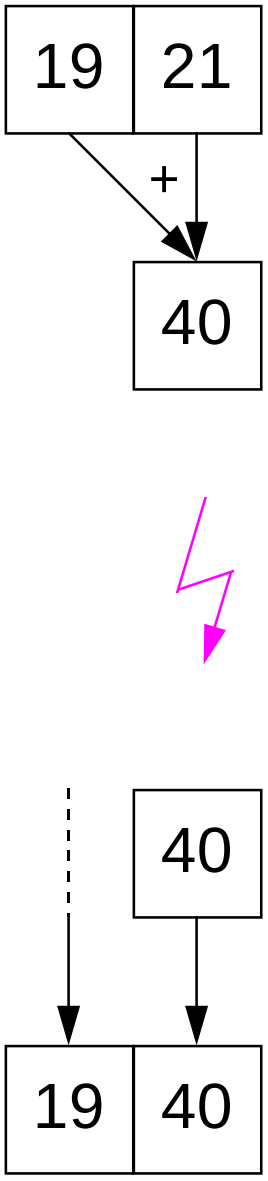




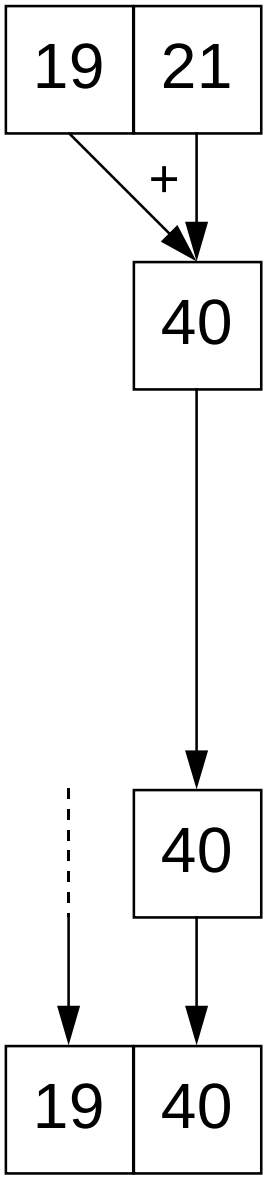
Präfix-Summe
Beispiel



Präfix-Summe
Beispiel



Präfix-Summe
Beispiel



Präfix-Summe Algorithmus

Input: Ein Array ganzer Zahlen x[1..n]

Output: $s_i := x_1 + x_2 + ... + x_i$ (für jedes i=1,...,n)

p(n) = n Prozessoren P_i (i=1,...,n)

Algorithmus:

0. if n = 1 then s[1]:= x[1]; *halt* end;

1. forall i: if i gerade then y[i/2] := x[i-1] + x[i] fi;
   (für Hilfsarray y[1..n/2])

2. forall i = 1,...,n/2:
   berechne rekursiv $z_i := y_1 + y_2 + ... + y_i$
   (für Hilfsarray z[1..n/2])

3. forall i if i gerade then
   s[i] := z[i/2]
   else
   s[i] := z[(i-1)/2] **+** x[i]
   end;

Präfix-Summe Analyse

- EREW
- Schritt 0: konstante Zeit
  Schritt 1: konstante Zeit bei n/2 Proz.
  Schritt 2: Rekursion auf n/2 Werte
  Schritt 3: konstante Zeit bei n/2 Proz.

- $p(n) = n/2$
- $t_p(n) \sim 4 \log n$
- $w_p(n) \sim 2n \log n$

- effizient, aber nicht optimal

Brents Lemma

Idee: Simulation vieler Prozessoren durch wenige

Parallele Rechnung: Laufzeit **t**, **w** Operationen (davon w_i Operationen beim i-ten Schritt)

Simuliere diese parallele Rechnung mit **p** Prozessoren

Simuliere den i-ten parallelen Schritt in Laufzeit $\left\lceil \frac{w_i}{p} \right\rceil < \frac{w_i}{p} + 1$

Gesamtlaufzeit: $\sum_{i=1}^{t} \left(\frac{w_i}{p} + 1 \right) = \frac{w}{p} + t$

Voraussetzung:
Prozessorallokation
ist kein Problem

Präfix-Summe
Analyse mit Brent

Anzahl Operationen für n Zahlen:

- $w = n + n/2 + ... + 1 = 2n - 1 \sim n$

Anzahl Rekursionen für n Zahlen:

- $t \sim \log n$

da $t_p(n) = w / p + t$ wählt man:

- $p(n) = n / \log n$

damit wird:

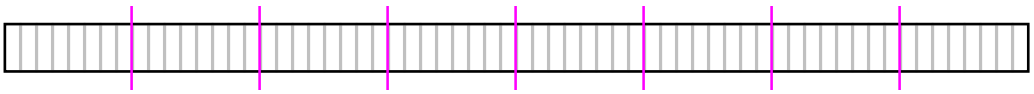
- $t_p(n) \sim \log n$
- $w_p(n) = t_p(n) \cdot p(n) \sim n$

- optimal

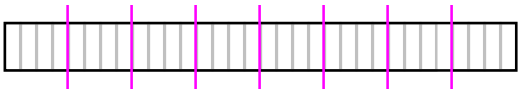
Präfix-Summe
Analyse mit Brent

- Problemgrösse n
- Prozessorzahl p(n) = n / log n

Laufzeit:

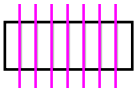


log n



log n/2

log n/4



⋮

1

1

1

Gesamtlaufzeit ~ log n

Anwendung 1

Komprimieren eines dünn besetzten Arrays

Input: dünnbesetztes Array

5	0	2	0	0	0	3	...

Output: komprimiertes Array

5	2	3	...

Algorithmus:

1. forall i:

```
if input[i] = 0 then
    x[i] := 0
else
    x[i] := 1
end;
```

2. **Präfix-Summe** ergibt
   für Array x das Resultat-Array s

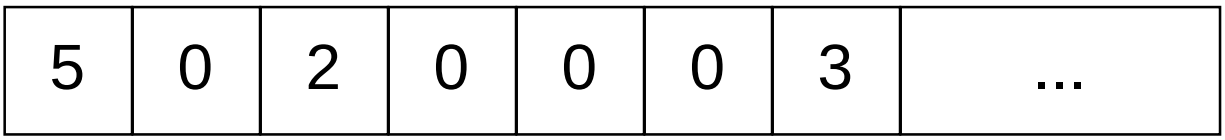
3. forall i

```
if input[i] ≠0 then
    output[s[i]] := input[i]
end;
```

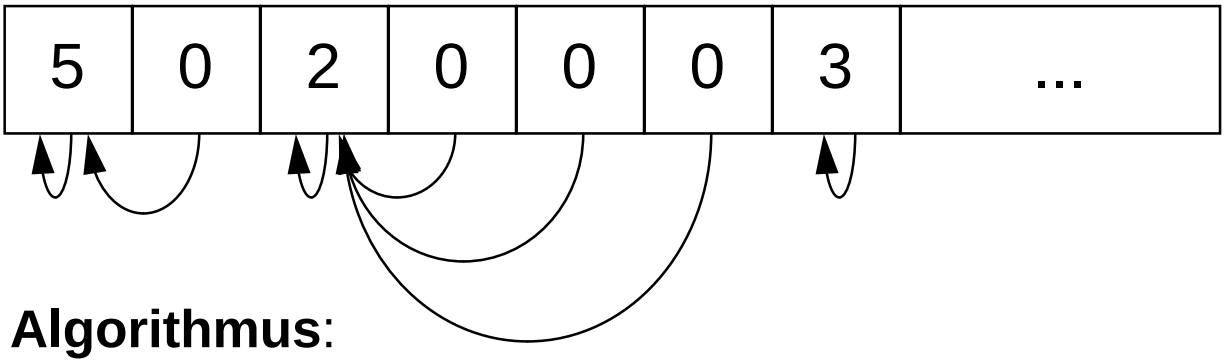
Anwendung 2

Komprimieren zu einer verketteten Liste

Input: dünnbesetztes Array



Output: Zeiger auf letztes Feld mit Inhalt ungleich 0



Algorithmus:

1. forall i: if input[i] = 0 then x[i] := 0 else x[i] := i end;

2. **Präfix-Summe**
Die Operation x[i] ⊕ x[i+1] ist definiert als:
if (x[i] > 0) and (x[i+1] = 0) then x[i] else x[i+1] end;

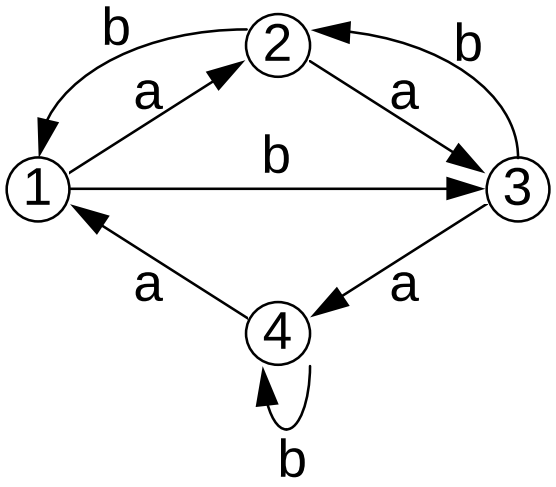
3. Im Resultat-Array stehen die Indexwerte

Allgemein: ⊕ muss assoziativ sein

Anwendung 3

Simulation eines endlichen Automaten

Präfix-Summe = **Zustand** des Automaten nach String Präfix



Input: a b b a a b b a

1. Ersetze Eingabezeichen durch Übergangsfunktion.
   a: 1→2, 2→3, 3→4, 4→1
   b: 1→3, 2→1, 3→2, 4→4

2. Berechne Präfix-Summe mit $\oplus$ als Zusammensetzung von Funktionen.
   ergibt für jeden Präfix die Gesamtübergangsfunktion.

3. Setze an jeder Stelle den Anfangszustand ein.
   ergibt erreichten Zustand für jeden Präfix.

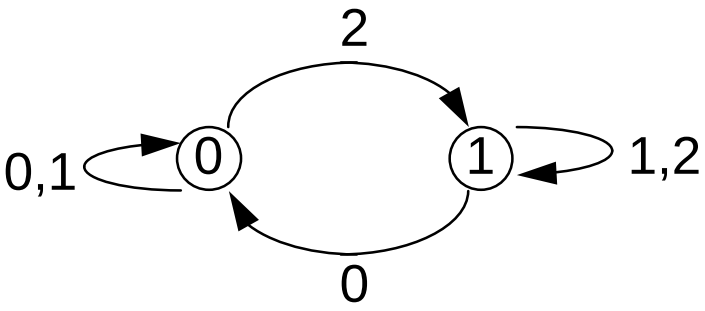
Anwendung 4

Addier-Schaltung

Input: zwei n-Bit-Integer a,b
Output: n+1-Bit-Integer a+b

a = 0 1 0 1 0 1 1 1
b = 0 0 0 1 0 0 1 1
~~0 0 1 0 1 1 1~~ = Übertrag
0 1 1 0 1 0 1 0 = Summe

Berechnung des Übertrages als endlichen Automaten realisieren:



Zustand = Übertrag
Übergangsfkt. = a_i+b_i

Zusammenfassung

- Präfix Summe ist vielseitig einsetzbar.
- Laufzeit $t_p(n) \sim \log n$
- Arbeit $w_p(n) \sim n$
- optimal