



# Chapter 8

# Parallel Algorithms

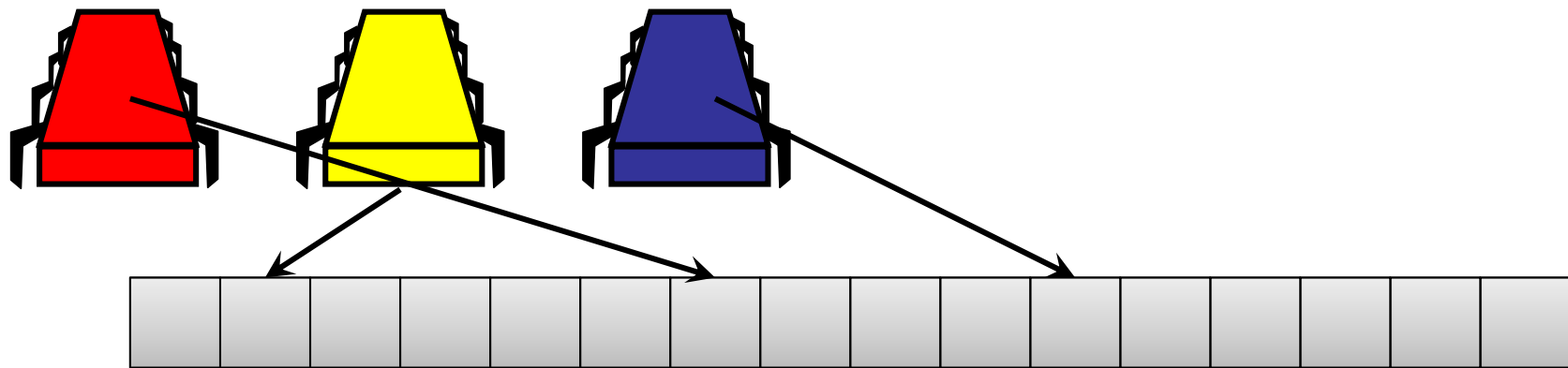
## Parallel Prefix Sums

**Algorithm Theory**  
**WS 2012/13**

**Fabian Kuhn**

# PRAM

- Parallel version of RAM model
- $p$  processors, shared random access memory



- Basic operations / access to shared memory cost 1
- Processor operations are synchronized
- **Focus on parallelizing computation** rather than cost of communication, locality, faults, asynchrony, ...

# Brent's Theorem

**Brent's Theorem:** On  $p$  processors, a parallel computation can be performed in time

$$T_p \leq \frac{T_1 - T_\infty}{p} + T_\infty.$$

**Proof:**

- Greedy scheduling achieves this...
- #operations scheduled with  $\infty$  processors in round  $i$ :  $x_i$

# Prefix Sums

- The following works for any associative binary operator  $\oplus$ :

**associativity:**  $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

**All-Prefix-Sums:** Given a sequence of  $n$  values  $a_1, \dots, a_n$ , the all-prefix-sums operation w.r.t.  $\oplus$  returns the sequence of prefix sums:

$$s_1, s_2, \dots, s_n = a_1, a_1 \oplus a_2, a_1 \oplus a_2 \oplus a_3, \dots, a_1 \oplus \dots \oplus a_n$$

- Can be computed efficiently in parallel and turns out to be an important building block for designing parallel algorithms

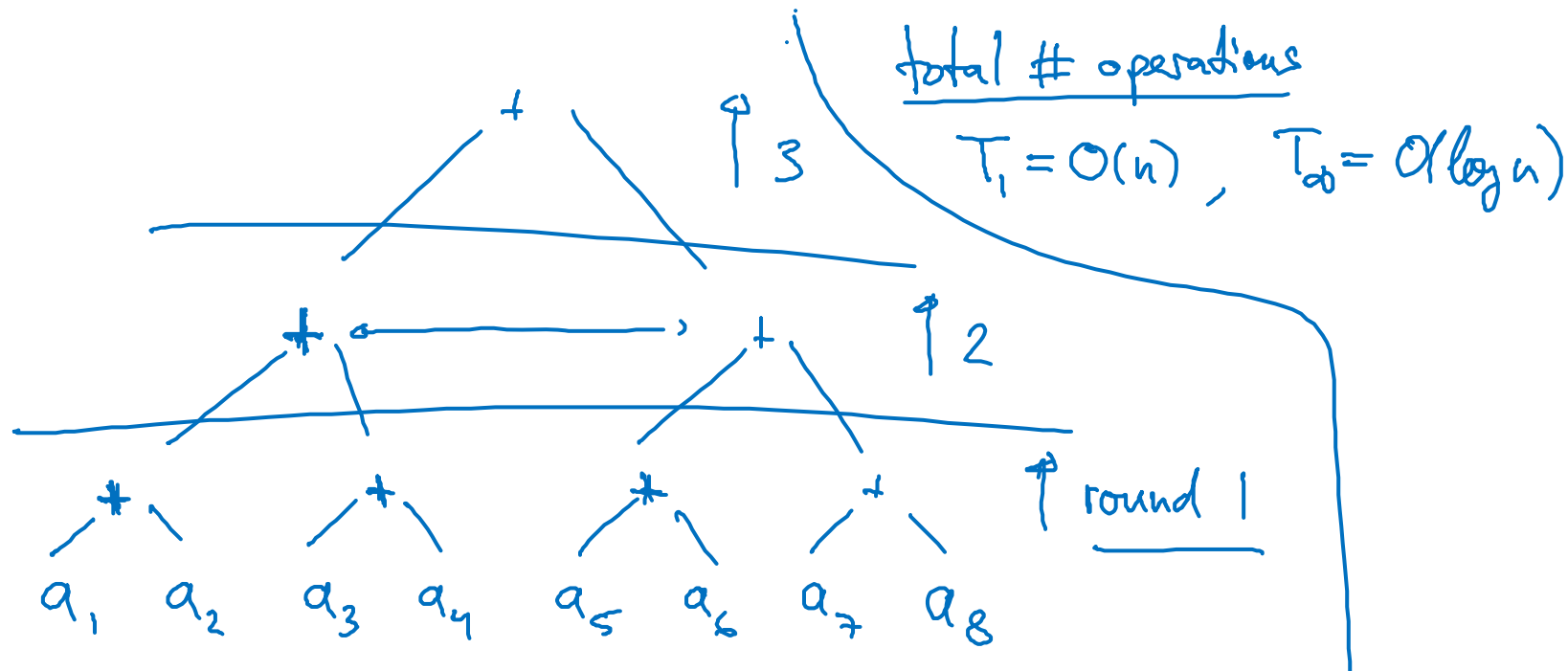
**Example:** Operator:  $+$ , input:  $a_1, \dots, a_8 = 3, 1, 7, 0, 4, 1, 6, 3$

$$s_1, \dots, s_8 = 3, 4, 11, 11, 15, 16, 22, 25$$

# Computing the Sum

- Let's first look at  $s_n = a_1 \oplus a_2 \oplus \dots \oplus a_n$
- Parallelize using a binary tree:

$$\left( (a_1 + a_2) + (a_3 + a_4) \right) + \left( (a_5 + a_6) + (a_7 + a_8) \right)$$



# Computing the Sum

**Lemma:** The sum  $s_n = a_1 \oplus a_2 \oplus \dots \oplus a_n$  can be computed in time  $O(\log n)$  on an EREW PRAM. The total number of operations (total work) is  $O(n)$ .

**Proof:**

EREW: excl. read, excl. write

**Corollary:** The sum  $s_n$  can be computed in time  $O(\log n)$  using  $O(n/\log n)$  processors on an EREW PRAM.

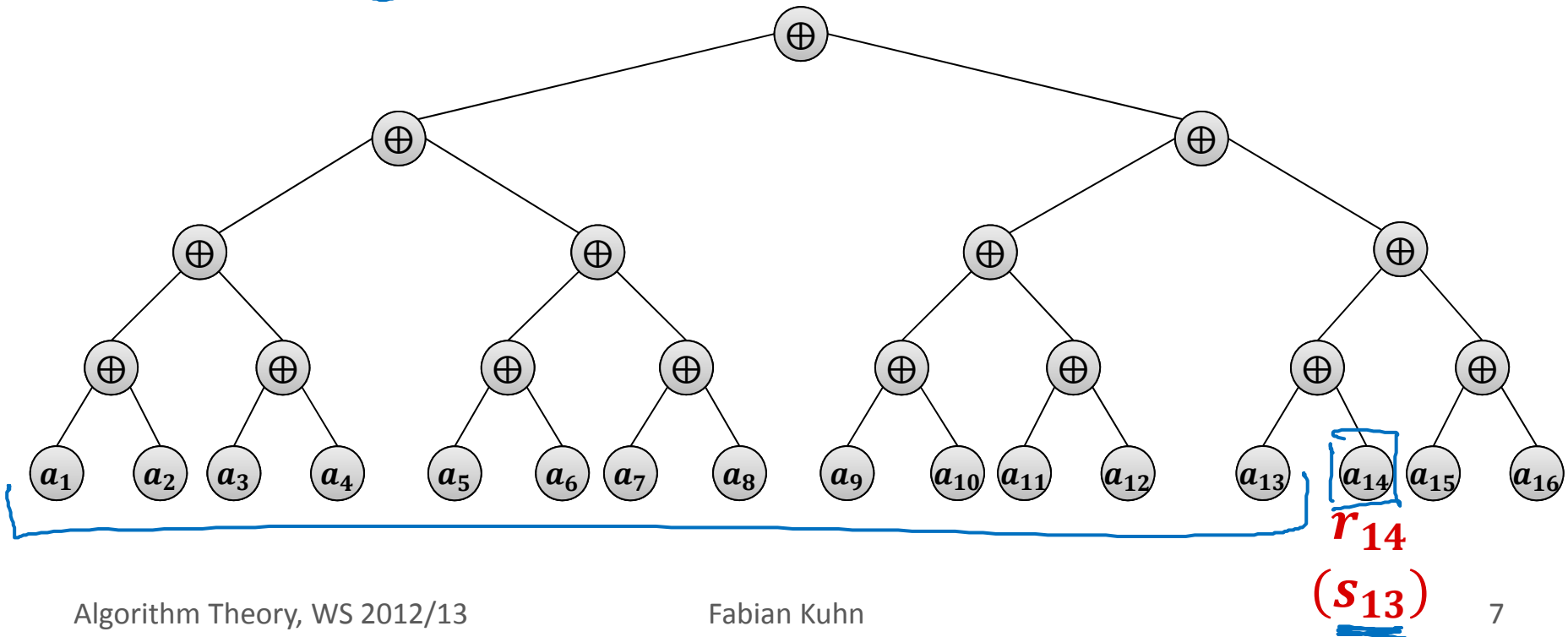
**Proof:**

- Follows from Brent's theorem ( $T_1 = O(n)$ ,  $T_\infty = O(\log n)$ )

$$T_p \leq \frac{T_1}{p} + T_\infty = \frac{O(n)}{p} + O(\log n)$$

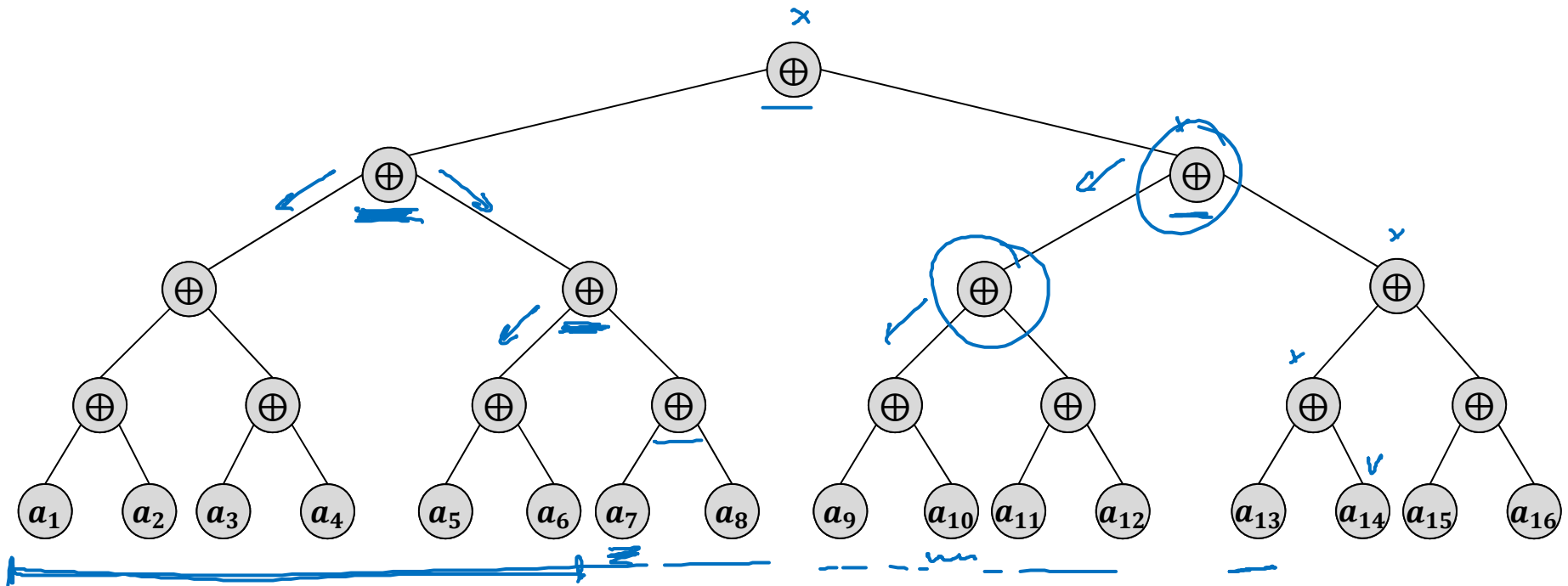
# Getting The Prefix Sums

- Instead of computing the sequence  $s_1, s_2, \dots, s_n$  let's compute  $r_1, \dots, r_n = 0, s_1, s_2, \dots, s_{n-1}$  (0: neutral element w.r.t.  $\oplus$ )  
 $r_1, \dots, r_n = 0, a_1, a_1 \oplus a_2, \dots, a_1 \oplus \dots \oplus a_{n-1}$
- Together with  $s_n$ , this gives all prefix sums  $s_i = r_i + a_i$
- Prefix sum  $r_i = s_{i-1} = a_1 \oplus \dots \oplus a_{i-1}$ :



# Getting The Prefix Sums

**Claim:** The prefix sum  $r_i = a_1 \oplus \dots \oplus a_{i-1}$  is the sum of all the leaves in the left sub-tree of ancestor  $u$  of the leaf  $v$  containing  $a_i$  such that  $v$  is in the right sub-tree of  $u$ .





# Computing The Prefix Sums

For each node  $v$  of the binary tree, define  $r(v)$  as follows:

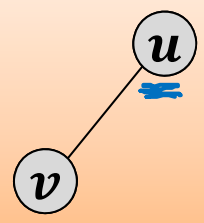
- $r(v)$  is the sum of the values  $a_i$  at the leaves in all the left sub-trees of ancestors  $u$  of  $v$  such that  $v$  is in the right sub-tree of  $u$ .

For a leaf node  $v$  holding value  $a_i$ :  $r(v) = r_i = s_{i-1}$



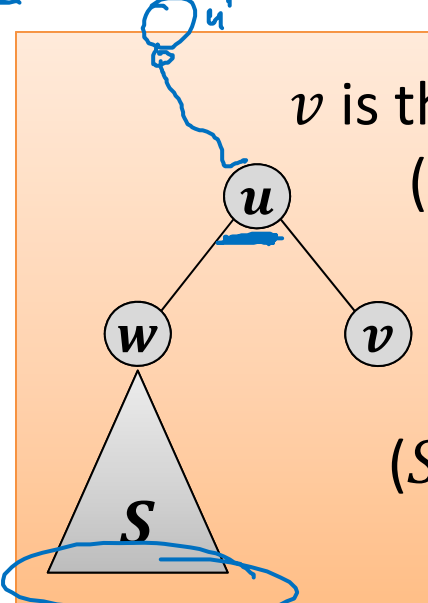
For the root node:  $r(\text{root}) = 0$

For all other nodes  $v$ :



$v$  is the left child of  $u$ :

$r(v) = r(u)$



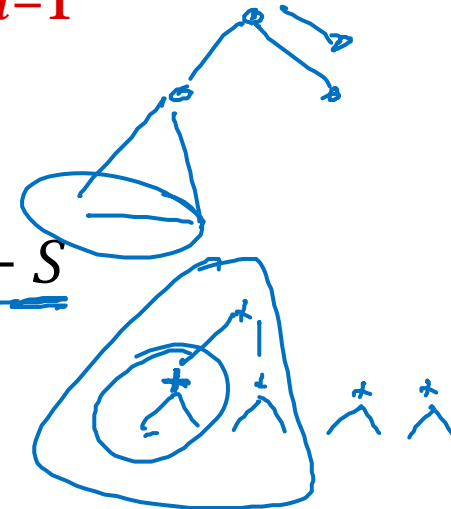
$v$  is the right child of  $u$ :  
( $u$  has left child  $w$ )

$r(v) = r(u) + S$

( $S$ : sum of values in sub-tree of  $w$ )

# Computing The Prefix Sums

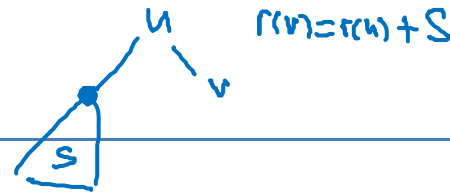
- leaf node  $v$  holding value  $a_i$ :  $\underline{r(v)} = \underline{r_i} = \underline{s_{i-1}}$
- root node:  $\underline{r(\text{root})} = \underline{0}$
- Node  $v$  is the left child of  $u$ :  $\underline{r(v)} = \underline{r(u)}$
- Node  $v$  is the right child of  $u$ :  $\underline{r(v)} = \underline{r(u) + S}$ 
  - Where:  $S$  = sum of values in left sub-tree of  $u$



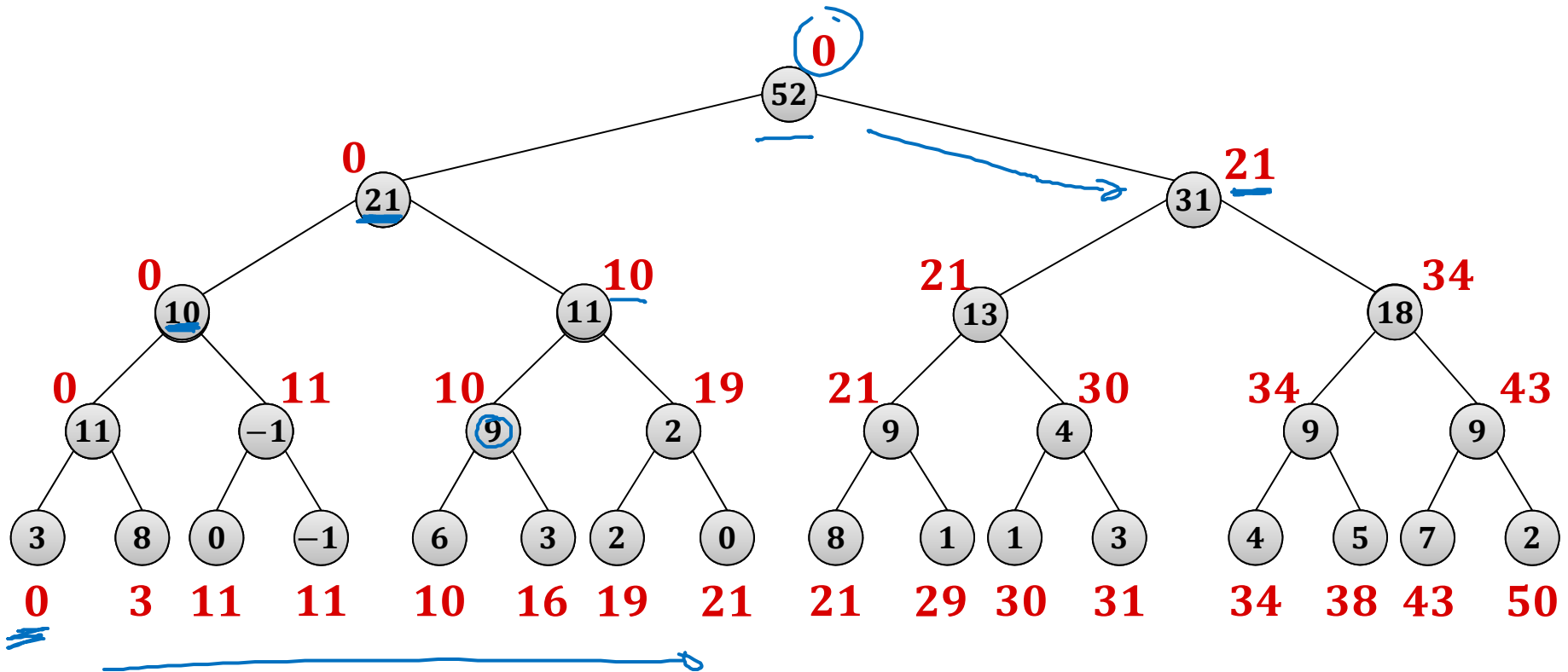
## Algorithm to compute values $r(v)$ :

1. Compute sum of values in each sub-tree (bottom-up)
  - Can be done in parallel time  $O(\log n)$  with  $O(n)$  total work
2. Compute values  $r(v)$  **top-down** from root to leaves:
  - To compute the value  $r(v)$ , only  $r(u)$  of the parent  $u$  and the sum of the left sibling (if  $v$  is a right child) are needed
  - Can be done in parallel time  $O(\log n)$  with  $O(n)$  total work

# Example



1. Compute sums of all sub-trees
  - Bottom-up (level-wise in parallel, starting at the leaves)
2. Compute values  $r(v)$ 
  - Top-down (starting at the root)



# Computing Prefix Sums

**Theorem:** Given a sequence  $a_1, \dots, a_n$  of  $n$  values, all prefix sums  $s_i = a_1 \oplus \dots \oplus a_i$  (for  $1 \leq i \leq n$ ) can be computed in **time  $O(\log n)$**  using  **$O(n/\log n)$  processors** on an EREW PRAM.

## Proof:

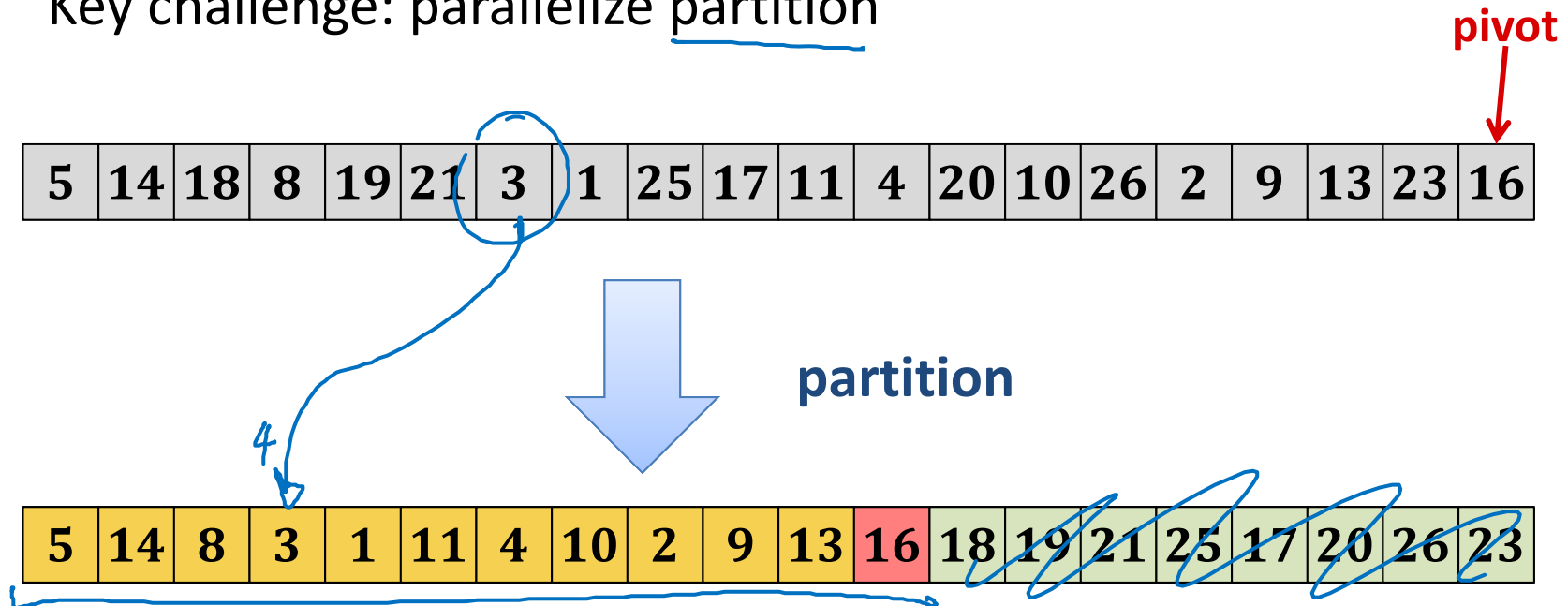
- Computing the sums of all sub-trees can be done in parallel in time  $O(\log n)$  using  $O(n)$  total operations.
- The same is true for the top-down step to compute the  $r(v)$
- The theorem then follows from Brent's theorem:

$$T_1 = O(n), \quad T_\infty = O(\log n) \quad \Rightarrow \quad T_p < T_\infty + \frac{T_1}{p}$$

**Remark:** This can be adapted to other parallel models and to different ways of storing the value (e.g., array or list)

# Parallel Quicksort

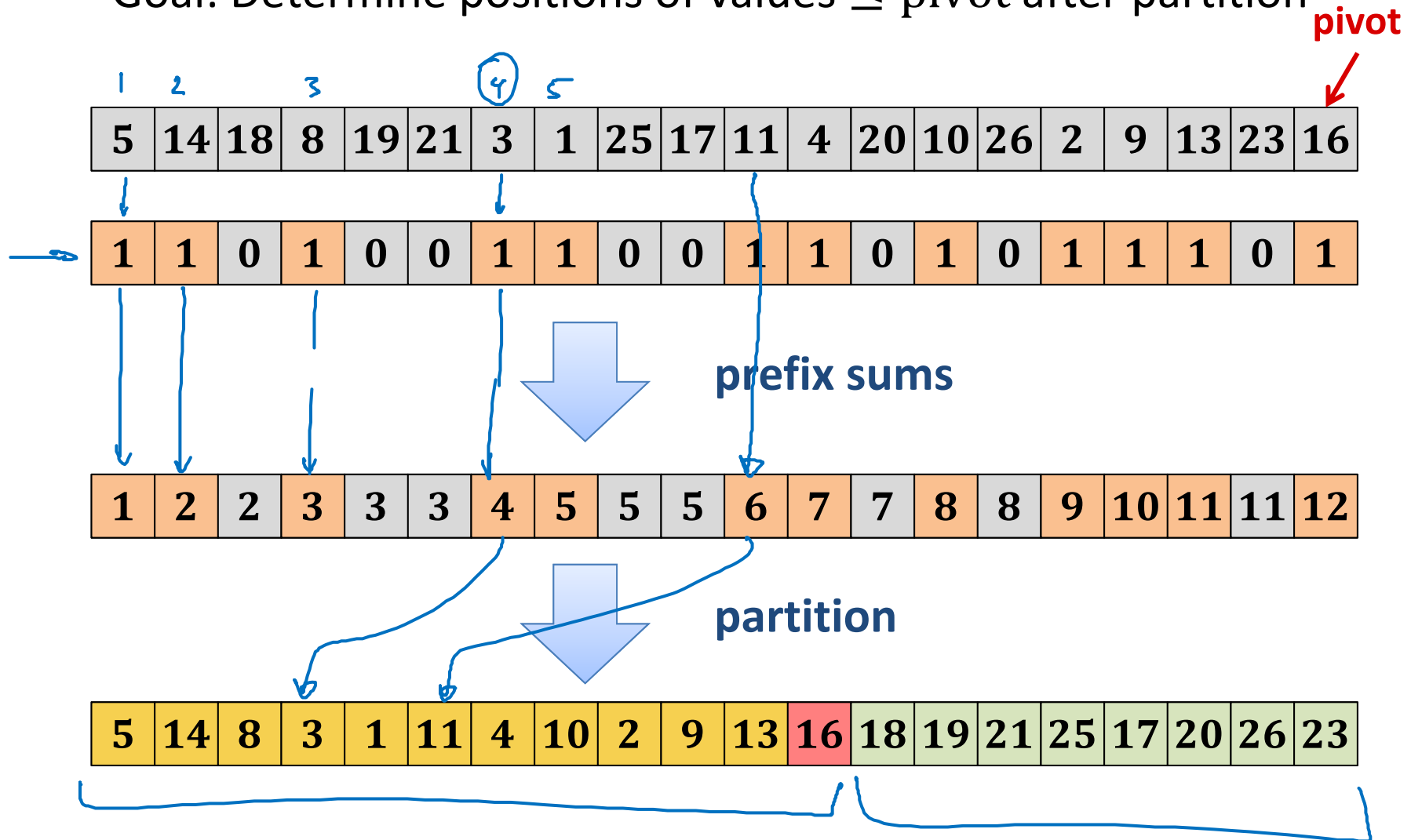
- Key challenge: parallelize partition



- How can we do this in parallel?
- For now, let's just care about the values  $\leq$  pivot
- What are their new positions

# Using Prefix Sums

- Goal: Determine positions of values  $\leq$  pivot after partition



# Partition Using Prefix Sums

- The positions of the entries  $>$  pivot can be determined in the same way
- **Prefix sums:**  $T_1 = O(n)$ ,  $T_\infty = O(\log n)$
- **Remaining computations:**  $T_1 = O(n)$ ,  $T_\infty = O(1)$
- **Overall:**  $T_1 = O(n)$ ,  $T_\infty = O(\log n)$

**Lemma:** The partitioning of quicksort can be carried out in parallel in time  $O(\log n)$  using  $O\left(\frac{n}{\log n}\right)$  processors.

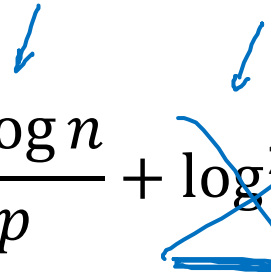
**Proof:**

- By Brent's theorem:  $T_p \leq \frac{T_1}{p} + T_\infty$

# Applying to Quicksort

**Theorem:** On an EREW PRAM, using  $p$  processors, randomized quicksort can be executed in time  $T_p$  (in expectation and with high probability), where

$$T_p = O\left(\frac{n \log n}{p} + \log^2 n\right).$$



**Proof:**

$$\left. \begin{array}{l}
 \text{1 recursion level: } T_1 = O(n), \quad T_{\infty} = O(\log n) \\
 O(\log n) \text{ rec. levels w.h.p. /...}
 \end{array} \right\} \begin{array}{l}
 \text{overall} \\
 T_1 = O(n \log n) \\
 T_{\infty} = O(\log^2 n)
 \end{array}$$

**Remark:**

- We get optimal (linear) speed-up w.r.t. to the sequential algorithm for all  $p = O(\underline{n/\log n})$ .



# Other Applications of Prefix Sums

---

- Prefix sums are a very powerful primitive to design parallel algorithms.
  - Particularly also by using other operators than  $+$

## **Example Applications:**

- Lexical comparison of strings
- Add multi-precision numbers
- Evaluate polynomials
- Solve recurrences
- Radix sort / quick sort
- Search for regular expressions
- Implement some tree operations
- ...