



# **Chapter 10**

# **Parallel Algorithms**

**Algorithm Theory**  
**WS 2019/20**

**Fabian Kuhn**

# Parallel Computations

$T_p$ : time to perform comp. with  $p$  procs

- **Lower Bounds:**

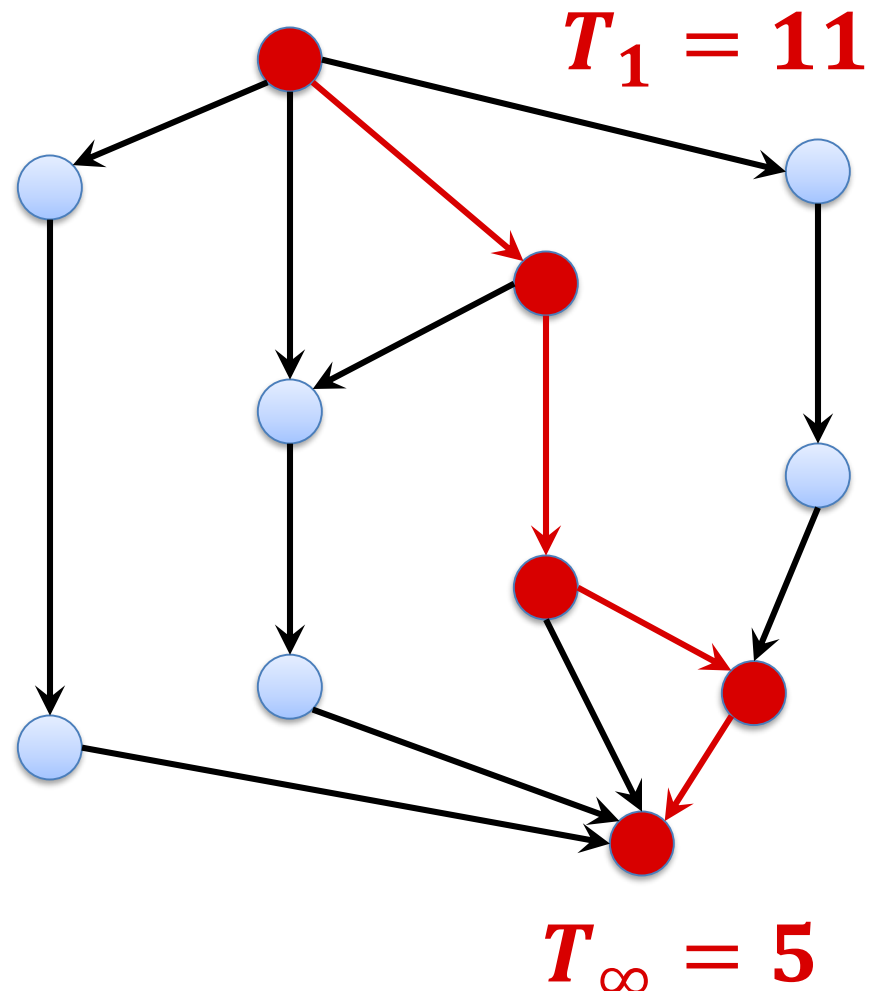
$$T_p \geq \frac{T_1}{p}, \quad T_p \geq T_\infty$$

- **Parallelism:**  $\frac{T_1}{T_\infty}$

– maximum possible speed-up

- **Linear Speed-up:**

$$\frac{T_p}{T_1} = \Theta(p)$$



# Brent's Theorem

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

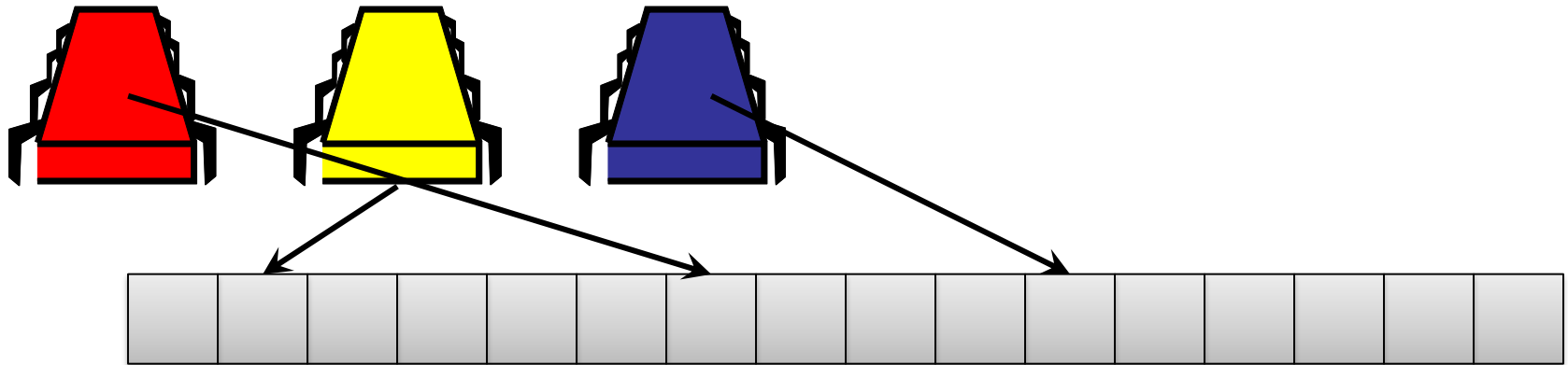
$$T_p \leq \frac{T_1 - T_\infty}{p} + T_\infty.$$

## Proof:

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

# 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, ...

Back to the PRAM:

- Shared random access memory, synchronous computation steps
- The PRAM model comes in variants...

## **EREW (exclusive read, exclusive write):**

- Concurrent memory access by multiple processors is not allowed
- If two or more processors try to read from or write to the same memory cell concurrently, the behavior is not specified

## **CREW (concurrent read, exclusive write):**

- Reading the same memory cell concurrently is OK
- Two concurrent writes to the same cell lead to unspecified behavior
- This is the first variant that was considered (already in the 70s)

The PRAM model comes in variants...

## **CRCW (concurrent read, concurrent write):**

- Concurrent reads and writes are both OK
- Behavior of concurrent writes has to be specified
  - Weak CRCW: concurrent write only OK if all processors write 0
  - Common-mode CRCW: all processors need to write the same value
  - Arbitrary-winner CRCW: adversary picks one of the values
  - Priority CRCW: value of processor with highest ID is written
  - Strong CRCW: largest (or smallest) value is written

- The given models are ordered in strength:

**weak  $\leq$  common-mode  $\leq$  arbitrary-winner  $\leq$  priority  $\leq$  strong**

# Some Relations Between PRAM Models



**Theorem:** A parallel computation that can be performed in time  $t$ , using  $p$  proc. on a strong CRCW machine, can also be performed in time  $O(t \log p)$  using  $p$  processors on an EREW machine.

- Each (parallel) step on the CRCW machine can be simulated by  $O(\log p)$  steps on an EREW machine

**Theorem:** A parallel computation that can be performed in time  $t$ , using  $p$  probabilistic processors on a strong CRCW machine, can also be performed in expected time  $O(t \log p)$  using  $O(p/\log p)$  processors on an arbitrary-winner CRCW machine.

- The same simulation turns out more efficient in this case

# Some Relations Between PRAM Models



**Theorem:** A computation that can be performed in time  $t$ , using  $p$  processors on a strong CRCW machine, can also be performed in time  $O(t)$  using  $O(p^2)$  processors on a weak CRCW machine

**Proof:**

- **Strong:** largest value wins, **weak:** only concurrently writing 0 is OK



# Some Relations Between PRAM Models



**Theorem:** A computation that can be performed in time  $t$ , using  $p$  processors on a strong CRCW machine, can also be performed in time  $O(t)$  using  $O(p^2)$  processors on a weak CRCW machine

**Proof:**

- **Strong:** largest value wins, **weak:** only concurrently writing 0 is OK

# Computing the Maximum

**Given:**  $n$  values

**Goal:** find the maximum value

**Observation:** The maximum can be computed in parallel by using a binary tree.

# Computing the Maximum

**Observation:** On a strong CRCW machine, the maximum of a  $n$  values can be computed in  $O(1)$  time using  $n$  processors

- Each value is concurrently written to the same memory cell

**Lemma:** On a **weak CRCW** machine, the **maximum of  $n$  integers between 1 and  $\sqrt{n}$**  can be computed in **time  $O(1)$**  using  **$O(n)$  proc.**

**Proof:**

- We have  $\sqrt{n}$  memory cells  $f_1, \dots, f_{\sqrt{n}}$  for the possible values
- Initialize all  $f_i := 1$
- For the  $n$  values  $x_1, \dots, x_n$ , processor  $j$  sets  $f_{x_j} := 0$ 
  - Since only zeroes are written, concurrent writes are OK
- Now,  $f_i = 0$  iff value  $i$  occurs at least once
- Strong CRCW machine: max. value in time  $O(1)$  w.  $O(\sqrt{n})$  proc.
- Weak CRCW machine: time  $O(1)$  using  $O(n)$  proc. (prev. lemma)

# Computing the Maximum

**Theorem:** If each value can be represented using  $O(\log n)$  bits, the maximum of  $n$  (integer) values can be computed in time  $O(1)$  using  $O(n)$  processors on a weak CRCW machine.

## Proof:

- First look at  $\frac{\log_2 n}{2}$  highest order bits
- The maximum value also has the maximum among those bits
- There are only  $\sqrt{n}$  possibilities for these bits
- max. of  $\frac{\log_2 n}{2}$  highest order bits can be computed in  $O(1)$  time
- For those with largest  $\frac{\log_2 n}{2}$  highest order bits, continue with next block of  $\frac{\log_2 n}{2}$  bits, ...

# 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:  $a_1, \dots, a_8 = 3, 1, 7, 0, 4, 1, 6, 3$

$$s_1, \dots, s_8 =$$

# 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:

# 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:**

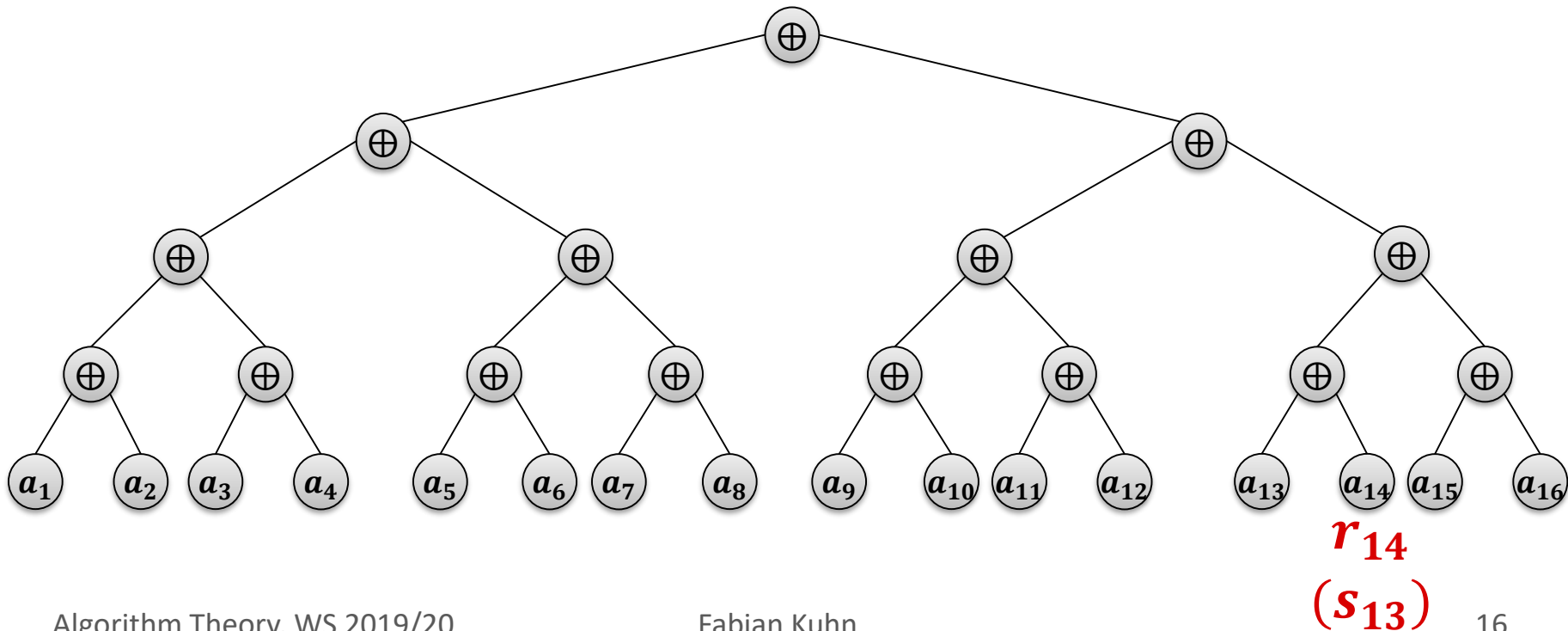
**Corollary:** The sum  $s_n$  can be computed in time  $O(\log n)$  using  $O(n/\log n)$  processors on an EREW PRAM.

**Proof:**

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

# Getting The Prefix Sums

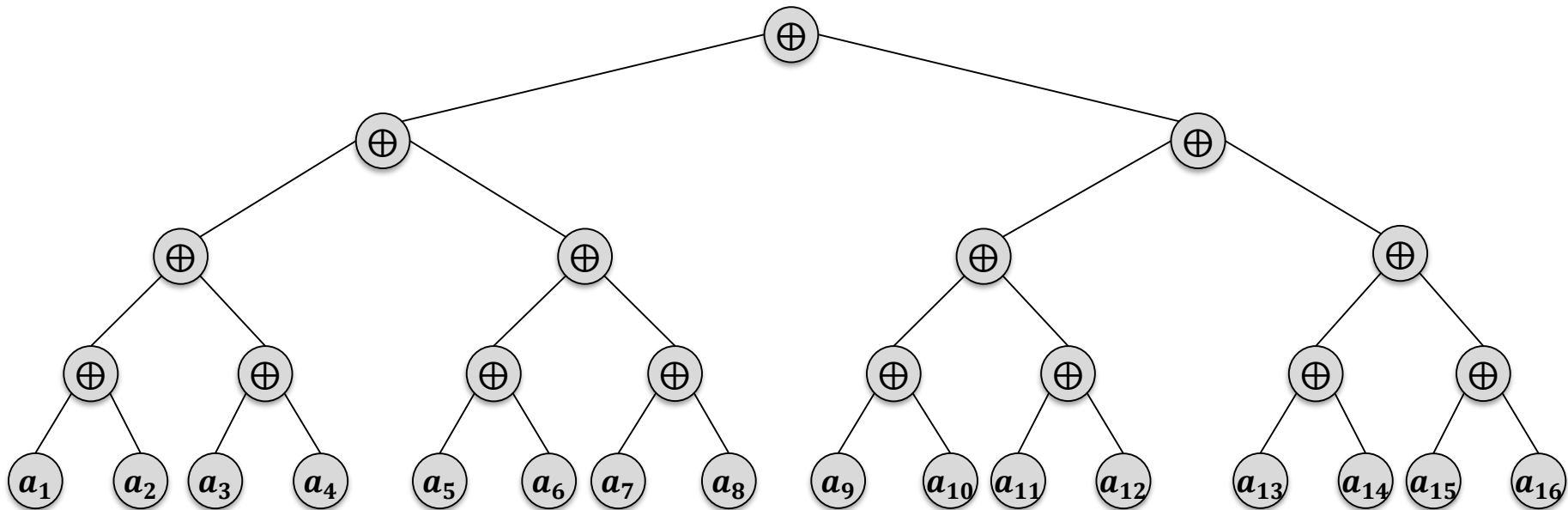
- Instead of computing the sequence  $s_1, s_2, \dots, s_n$  let's compute  $r_1, \dots, r_n = 0, s_1, s_2, \dots, s_{n-1}$  (0: neutral element w.r.t.  $\oplus$ )  
 $r_1, \dots, r_n = 0, a_1, a_1 \oplus a_2, \dots, a_1 \oplus \dots \oplus a_{n-1}$
- Together with  $s_n$ , this gives all prefix sums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

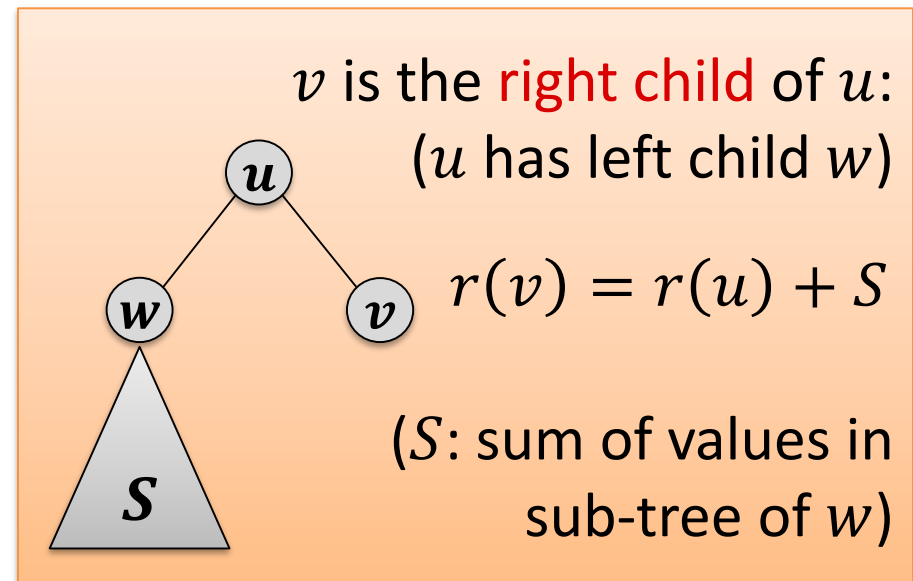
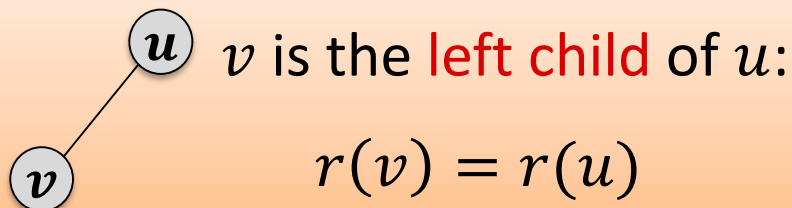
For each node  $v$  of the binary tree, define  $r(v)$  as follows:

- $r(v)$  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  $v$ :



# Computing The Prefix Sums

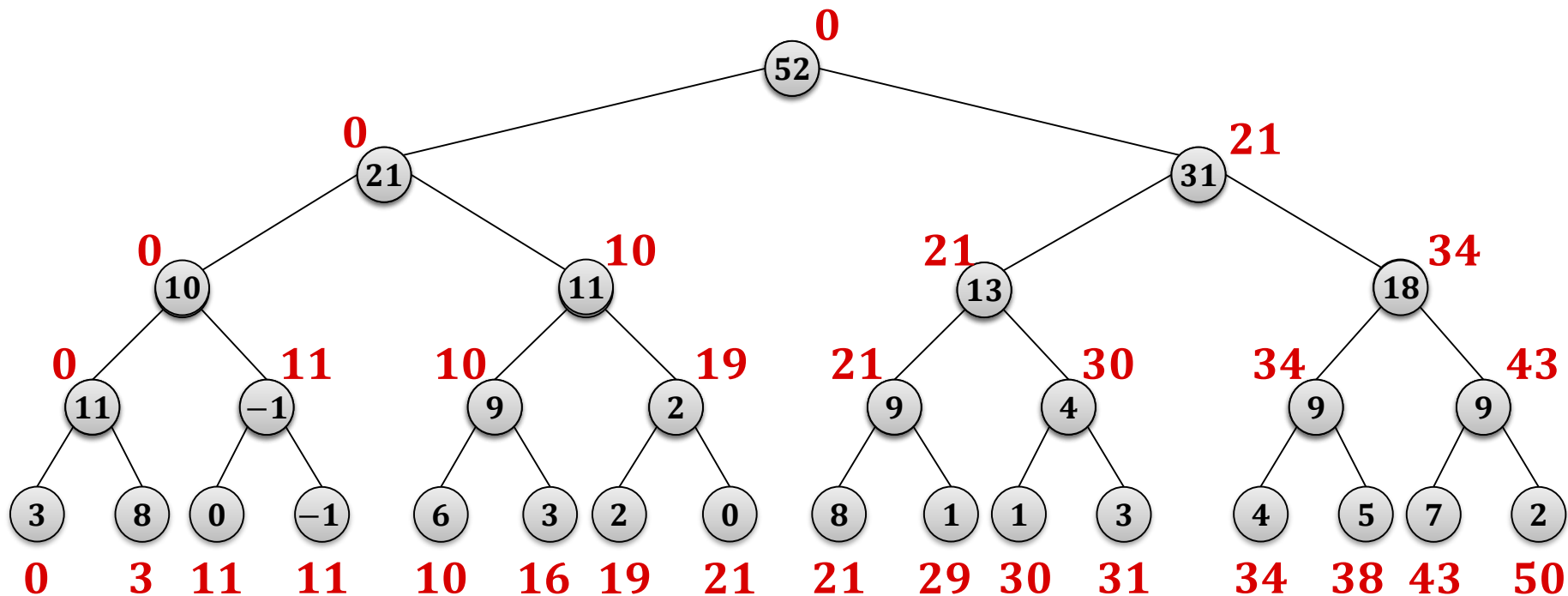
- leaf node  $v$  holding value  $a_i$ :  $r(v) = r_i = s_{i-1}$
- root node:  $r(\text{root}) = 0$
- Node  $v$  is the left child of  $u$ :  $r(v) = r(u)$
- Node  $v$  is the right child of  $u$ :  $r(v) = r(u) + S$ 
  - Where:  $S$  = sum of values in left sub-tree of  $u$

## 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:

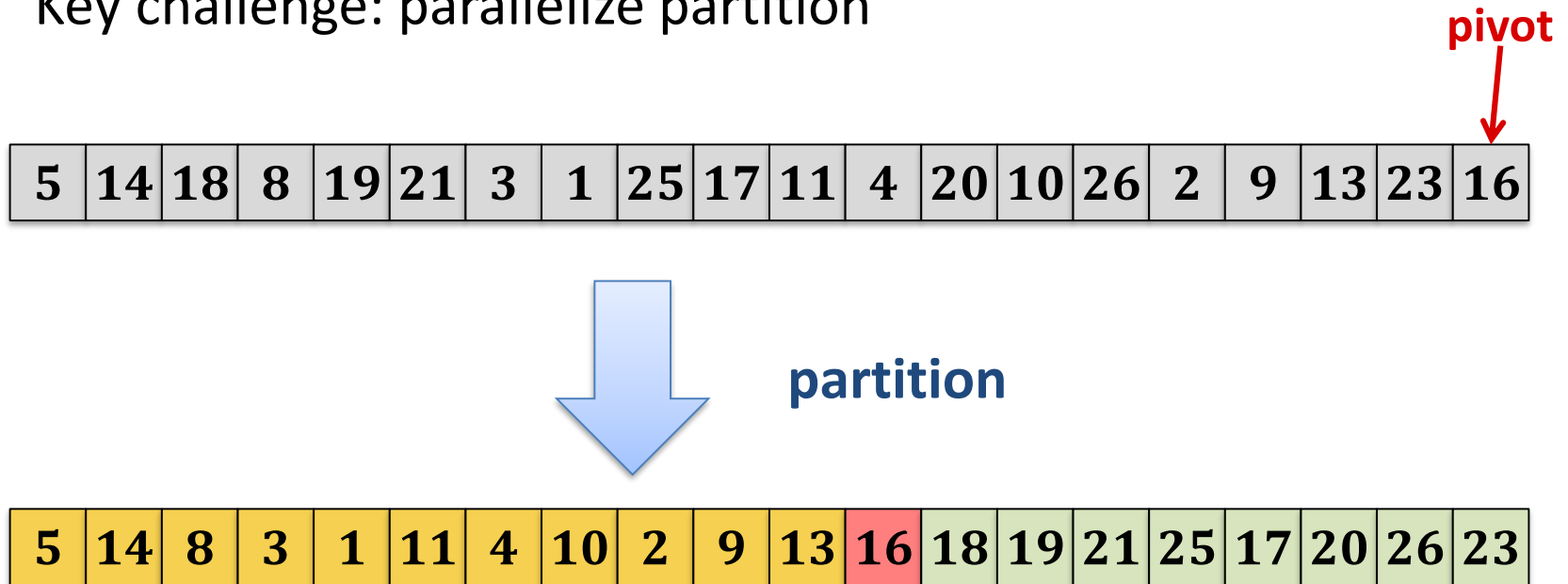
- Computing the sums of all sub-trees can be done in parallel in time  $O(\log n)$  using  $O(n)$  total operations.
- The same is true for the top-down step to compute the  $r(v)$
- The theorem then follows from Brent's theorem:

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

**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

**pivot**



5	14	18	8	19	21	3	1	25	17	11	4	20	10	26	2	9	13	23	16
---	----	----	---	----	----	---	---	----	----	----	---	----	----	----	---	---	----	----	----

1	1	0	1	0	0	1	1	0	0	1	1	0	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



**prefix sums**

1	2	2	3	3	3	4	5	5	5	6	7	7	8	8	9	10	11	11	12
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	----	----	----



**partition**

5	14	8	3	1	11	4	10	2	9	13	16	18	19	21	25	17	20	26	23
---	----	---	---	---	----	---	----	---	---	----	----	----	----	----	----	----	----	----	----

# 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:**

**Remark:**

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

# 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
- ...