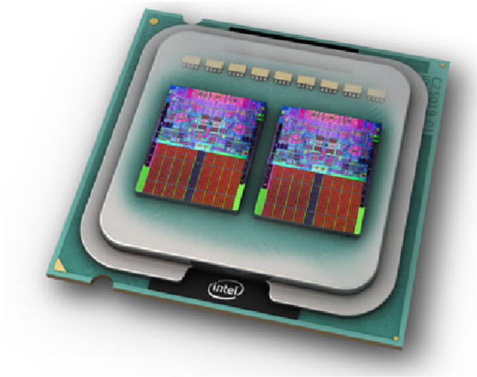
INFORMATION & COMPUTER SCIENCES
UNIVERSITY of HAWAIʻI at MĀNOA

AlgoPARC

ICS 621: Analysis of Algorithms

Prof. Nodari Sitchinava



5	8	3	4	1

Parallel Algorithms

Understanding (Parallel) Runtime

procedure FOO()
  a = 7
  b = 20
  c = 14
  d = 27
  e = 15

Understanding (Parallel) Runtime

procedure FOO()

→ a = 7

b = 20

c = 14

d = 27

e = 15

Understanding (Parallel) Runtime

procedure FOO()
a = 7
→ b = 20
c = 14
d = 27
e = 15

Understanding (Parallel) Runtime

procedure FOO()
    a = 7
    b = 20
→ c = 14
    d = 27
    e = 15

Understanding (Parallel) Runtime

```
procedure FOO()
    a = 7
    b = 20
    c = 14
→   d = 27
    e = 15
```

Understanding (Parallel) Runtime

procedure FOO()

a = 7

b = 20

c = 14

d = 27

→ e = 15

Understanding (Parallel) Runtime

procedure FOO()
    a = 7
    b = 20
    c = 14
    d = 27
    e = 15

Runtime = 5 steps

Understanding (Parallel) Runtime

procedure FOO()
  a = 7
  b = 20
  c = 14
  d = 27
  e = 15

Runtime = 5 steps

procedure PARALLELFOO()
  a = 7
  spawn {
    b = 20
    c = 14
  }
  d = 27
  e = 15
  sync

Understanding (Parallel) Runtime

procedure FOO()
a = 7
b = 20
c = 14
d = 27
e = 15

Runtime = 5 steps

procedure PARALLELFOO()
a = 7
spawn
b = 20
c = 14
d = 27
e = 15
sync

Understanding (Parallel) Runtime

procedure FOO()
  a = 7
  b = 20
  c = 14
  d = 27
  e = 15

Runtime = 5 steps

procedure PARALLELFOO()
  a = 7
  spawn
    b = 20
    c = 14
  d = 27
  e = 15
  sync

Understanding (Parallel) Runtime

procedure FOO()
a = 7
b = 20
c = 14
d = 27
e = 15

Runtime = 5 steps

procedure PARALLELFOO()
a = 7
spawn
b = 20
c = 14
d = 27
e = 15
sync

Understanding (Parallel) Runtime

procedure FOO()
  a = 7
  b = 20
  c = 14
  d = 27
  e = 15

Runtime = 5 steps

procedure PARALLELFOO()
  a = 7
  spawn
    b = 20
    c = 14
  d = 27
  e = 15
  sync

Understanding (Parallel) Runtime

procedure FOO()
  a = 7
  b = 20
  c = 14
  d = 27
  e = 15

Runtime = 5 steps

procedure PARALLELFOO()
  a = 7
  spawn
    b = 20
    c = 14
  d = 27
  e = 15
  sync

Understanding (Parallel) Runtime

procedure FOO()
  a = 7
  b = 20
  c = 14
  d = 27
  e = 15

Runtime = 5 steps

procedure PARALLELFOO()
  a = 7
  spawn
      b = 20
      c = 14
  d = 27
  e = 15
  sync

Understanding (Parallel) Runtime

procedure FOO()
a = 7
b = 20
c = 14
d = 27
e = 15

Runtime = 5 steps

procedure PARALLELFOO()
a = 7
spawn
b = 20
c = 14
d = 27
e = 15
sync

Runtime = 5 steps

Parallel Execution as a DAG

procedure PARALLELFOO()

  a = 7

  spawn

    b = 20

    c = 14

  d = 27

  e = 15

  sync

Parallel Execution as a DAG

procedure PARALLELFOO()

a = 7

spawn

b = 20

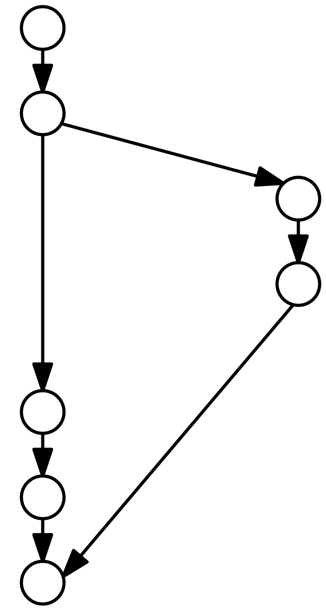
c = 14

d = 27

e = 15

sync

Dependency graph



Parallel Execution as a DAG

procedure PARALLELFOO()

a = 7

spawn

b = 20

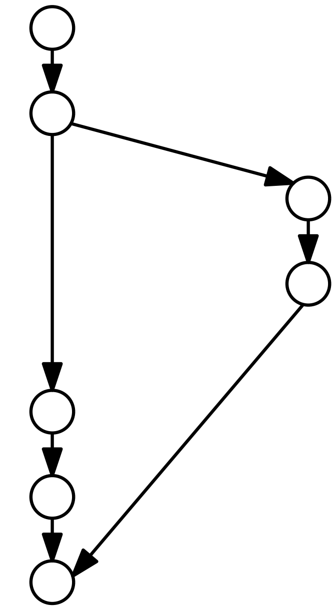
c = 14

d = 27

e = 15

sync

Dependency graph



- Parallel runtime: Longest path length in the dependency graph
  - $T(n)$

Parallel Execution as a DAG

procedure PARALLELFOO()

a = 7

spawn

b = 20

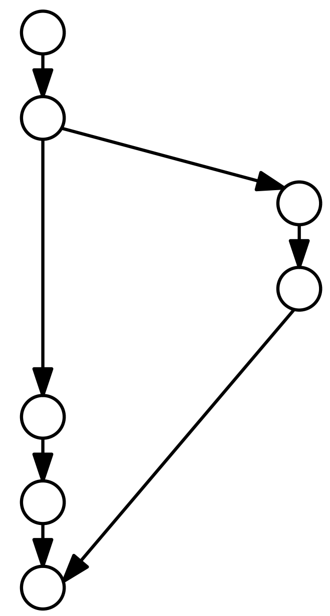
c = 14

d = 27

e = 15

sync

Dependency graph



- Parallel runtime: Longest path length in the dependency graph
  - $T(n)$
- Work: Total # of operations = Sequential runtime
  - $W(n)$

Understanding Parallel Runtime

procedure PARALLELFOO()
a = 7
spawn
b = 20
c = 14
d = 27
e = 15
sync

Understanding Parallel Runtime

procedure PARALLELFOO()
a = 7
spawn
b = 20
c = 14
d = 27
e = 15
sync

procedure ABOVE()
a = 7
procedure LEFT()
d = 27
e = 15
procedure RIGHT()
b = 20
c = 14
procedure BELOW()

Understanding Parallel Runtime

procedure ParallelFoo()
    Above()
    spawn

Left()   Right()

    sync
    Below()

procedure Above()
    a = 7
procedure Left()
    d = 27
    e = 15
procedure Right()
    b = 20
    c = 14
procedure Below()

Understanding Parallel Runtime

procedure ParallelFoo()
    Above()
    spawn
        Left()   Right()
    sync
    Below()

procedure Above()
    a = 7
procedure Left()
    d = 27
    e = 15
procedure Right()
    b = 20
    c = 14
procedure Below()

Parallel Runtime

$$T(\textsc{Above}) + 1 + \max(T(\textsc{Left}), T(\textsc{Right})) + 1 + T(\textsc{Below})$$

$$= T(\textsc{Above}) + \max\left\{\begin{array}{c} T(\textsc{Left}) \\ T(\textsc{Right}) \end{array}\right\} + T(\textsc{Below}) + O(1)$$

Proper Pseudocode

procedure PARALLELFOO()
  ABOVE()
  spawn

  LEFT()   RIGHT()

  sync
  BELOW()

Proper Pseudocode

procedure PARALLELFOO()
    ABOVE()
    spawn
    LEFT()   RIGHT()
    sync
    BELOW()

procedure PARALLELFOO()
    ABOVE()
    spawn RIGHT()
    LEFT()
    sync
    BELOW()

Proper Pseudocode

procedure ParallelFoo()
    Above()
    spawn

    Left()   Right()

    sync
    Below()

procedure ParallelFoo()
    Above()
    spawn Right()
    Left()
    sync
    Below()

procedure ParallelFoo()
    Above()
    in parallel do
        Right()
        Left()
    Below()

Taking a step further

```
procedure PARALLELFOO()
    ABOVE()
    in parallel do
        RIGHT()
        LEFT()
    BELOW()
```

```
procedure PARALLELFOO()
    ABOVE()
    in parallel do
        RIGHT()
        MIDDLE()
        LEFT()
    BELOW()
```

Taking a step further

procedure PARALLELFOO()
    ABOVE()
    in parallel do
        RIGHT()
        LEFT()
    BELOW()

procedure PARALLELFOO()
    ABOVE()
    in parallel do
        RIGHT()
        MIDDLE()
        LEFT()
    BELOW()

procedure PARALLELFOO()
    ABOVE()
    spawn

LEFT()   MIDDLE()   RIGHT()

sync
    BELOW()

Parallel **for** loop

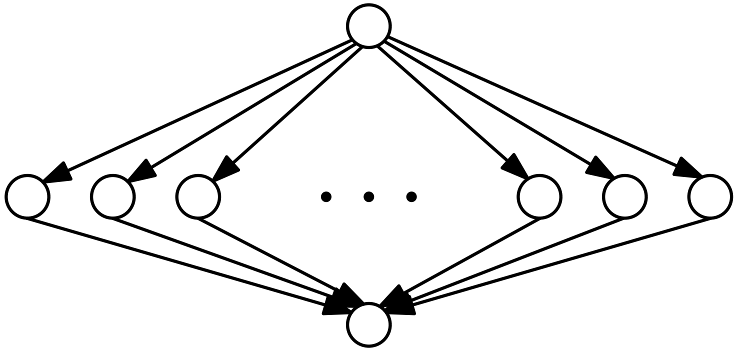
for i = 1 to n **in parallel do**
    $a[i] = a[i] + 1$

Spawn n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$
Synchronize all n threads

Parallel **for** loop

for i = 1 to n **in parallel do**
  $a[i] = a[i] + 1$

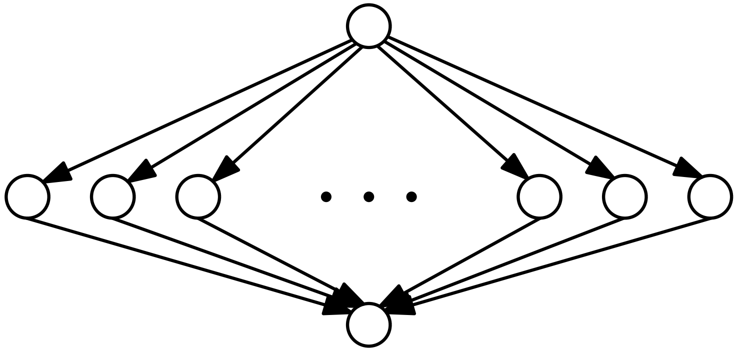
Spawn n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
  $a[i] = a[i] + 1$
Synchronize all n threads



Parallel **for** loop

for i = 1 to n **in parallel do**
    $a[i] = a[i] + 1$

Spawn n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$
Synchronize all n threads



Parallel Runtime: $T(n) = O(1)$

Simple example

for i = 1 to n **do**
    $a[i] = a[i] + 1$

5	8	3	4	1

Simple example

for $i = 1$ to n **do**

$\quad a[i] = a[i] + 1$

5	8	3	4	1

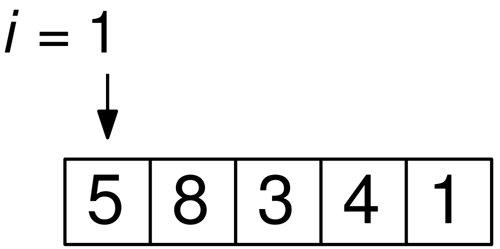
$\quad i = 1$

L: $a[i] = a[i] + 1$

$\quad i = i + 1$

$\quad$ **if** $i \leq n$: JUMPTO L

Simple example



for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

for i = 1 to n **do**
    $a[i] = a[i] + 1$

$i = 1$

6	8	3	4	1

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

$i =$ 2

6	8	3	4	1

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

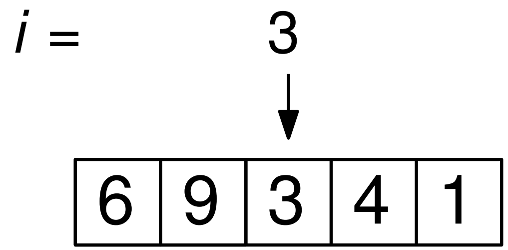
$i =$ 2

6	9	3	4	1

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example



for $i = 1$ to n **do**
    $a[i] = a[i] + 1$

```
    i = 1
L:  a[i] = a[i] + 1
    i = i + 1
    if i ≤ n: JUMPTO L
```

Simple example

for i = 1 to n **do**
    $a[i] = a[i] + 1$

$i =$ 3

6	9	4	4	1

```
   i = 1
L: a[i] = a[i] + 1
   i = i + 1
   if i ≤ n: JUMPTO L
```

Simple example

$i =$ 4

6	9	4	4	1

for $i = 1$ to n **do**
$\quad a[i] = a[i] + 1$

$$
\begin{array}{ll}
 & i = 1 \\
\text{L:} & a[i] = a[i] + 1 \\
 & i = i + 1 \\
 & \textbf{if } i \leq n\text{: } \textsc{JumpTo L}
\end{array}
$$

Simple example

for $i = 1$ to n **do**

   $a[i] = a[i] + 1$

$i =$ 4

6	9	4	5	1

```
    i = 1
L:  a[i] = a[i] + 1
    i = i + 1
    if i ≤ n: JUMPTO L
```

Simple example

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$i = \quad 5$

6	9	4	5	1

```
   i = 1
L: a[i] = a[i] + 1
   i = i + 1
   if i ≤ n: JUMPTO L
```

Simple example

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

$i =$ 5

6	9	4	5	2

$$
\begin{array}{ll}
 & i = 1 \\
\text{L:} & a[i] = a[i] + 1 \\
 & i = i + 1 \\
 & \textbf{if } i \le n\text{: }\textsc{JumpTo}\text{ L}
\end{array}
$$

Simple example

for i = 1 to n **do**
    $a[i] = a[i] + 1$

i = 5

6	9	4	5	2

Time

$O(n)$

$i = 1$
L: $a[i] = a[i] + 1$
$i = i + 1$
if $i \leq n$: JUMPTO L

Simple example

Time

for i = 1 to n **do**
    $a[i] = a[i] + 1$

6	9	4	5	2

$\boxed{O(n)}$

for i = 1 to n **in parallel do**
    $a[i] = a[i] + 1$

Simple example

Time

for $i = 1$ to n **do**
    $a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

for $i = 1$ to n **in parallel do**
    $a[i] = a[i] + 1$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Simple example

Time

for $i = 1$ to n **do**
$\quad a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

for $i = 1$ to n **in parallel do**
$\quad a[i] = a[i] + 1$

5	8	3	4	1

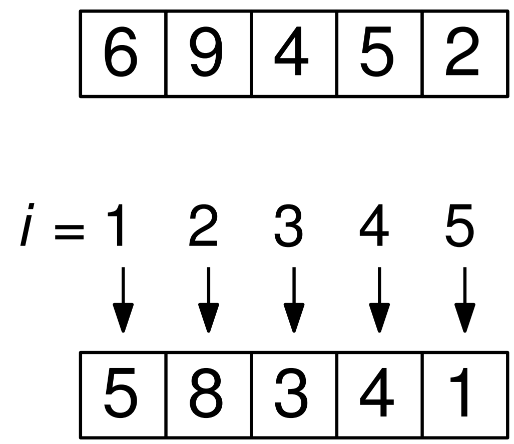
Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
$\quad a[i] = a[i] + 1$

Simple example

Time

for i = 1 to n **do**
  $a[i] = a[i] + 1$

$\boxed{O(n)}$



for i = 1 to n **in parallel do**
  $a[i] = a[i] + 1$

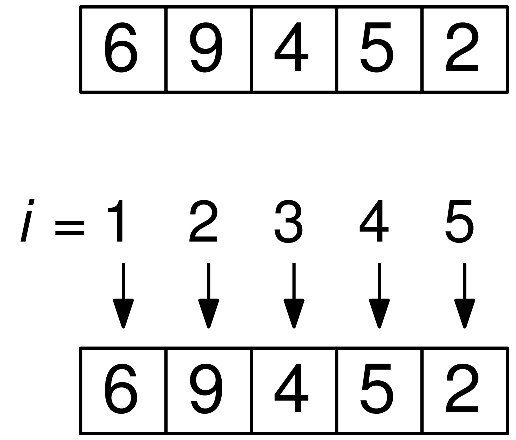
Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
  $a[i] = a[i] + 1$

Simple example

Time

for i = 1 to n **do**
$a[i] = a[i] + 1$

$O(n)$



for i = 1 to n **in parallel do**
$a[i] = a[i] + 1$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
$a[i] = a[i] + 1$

Simple example

Time

for $i = 1$ to n **do**
  $a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

$i = 1$ 2 3 4 5

6	9	4	5	2

for $i = 1$ to n **in parallel do**
  $a[i] = a[i] + 1$

$O(1)$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
  $a[i] = a[i] + 1$

Simple example

Time

for i = 1 to n **do**
  $a[i] = a[i] + 1$

6	9	4	5	2

$O(n)$

for i = 1 to n **in parallel do**
  $a[i] = a[i] + 1$

$i = 1$ 2 3 4 5

6	9	4	5	2

$O(1)$

Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
  $a[i] = a[i] + 1$

Parallel Time = time of the slowest thread

Simple example: Finding minimum

$a:$

5	8	3	2	9	7

Simple example: Finding minimum

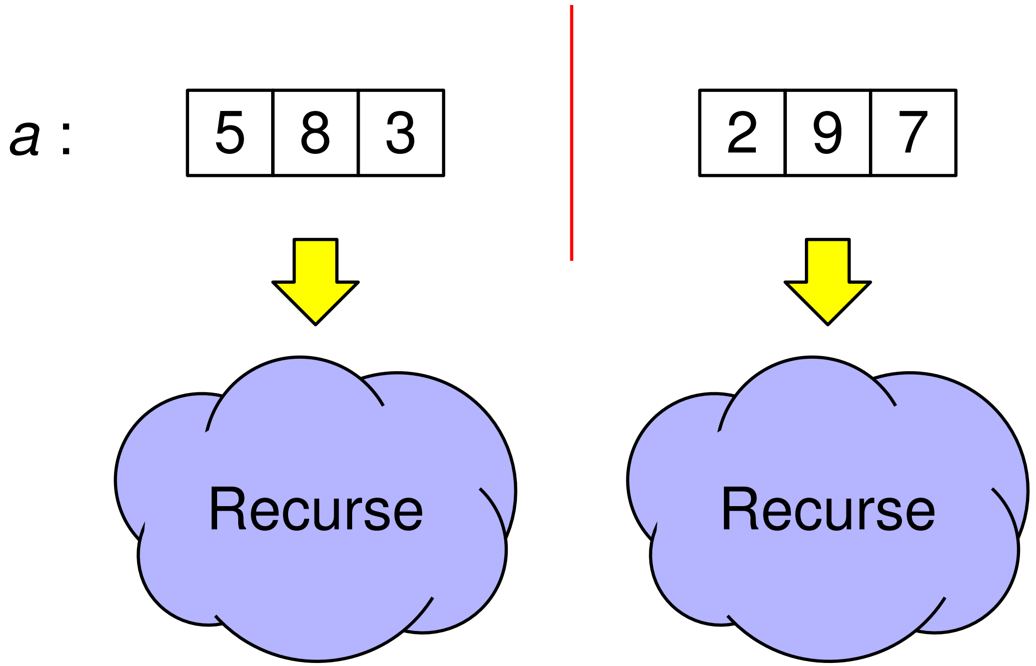
a :

5	8	3	2	9	7

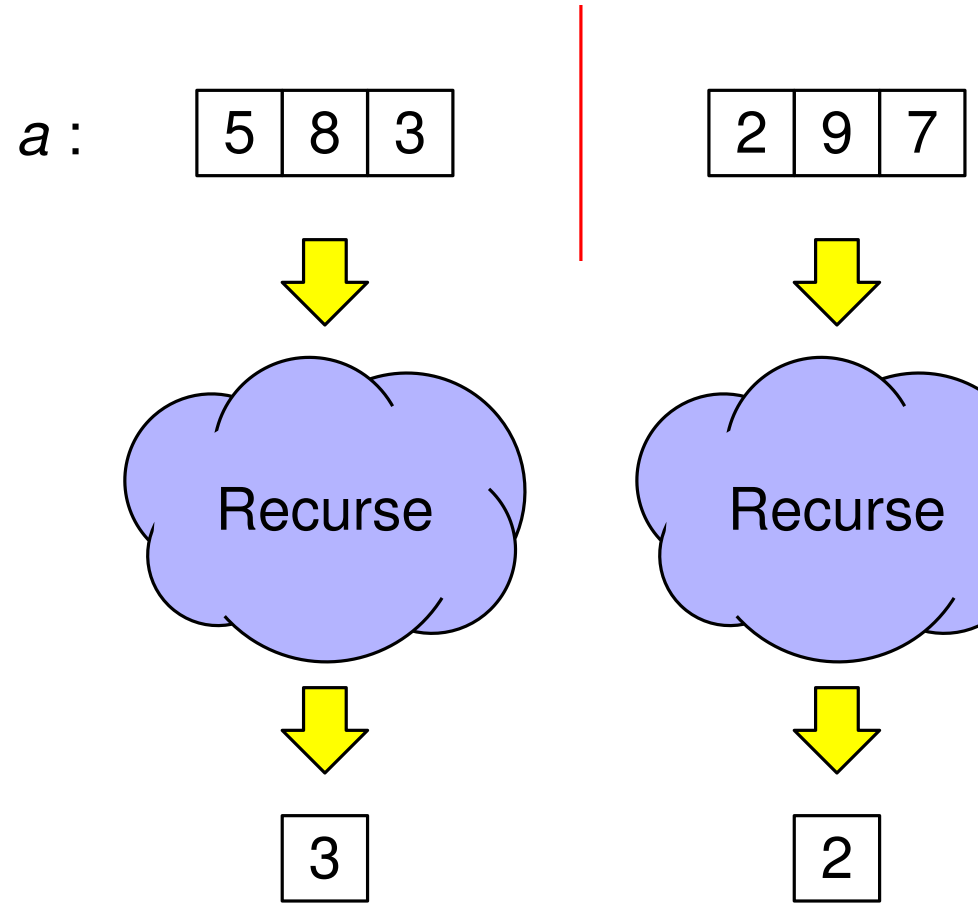
Simple example: Finding minimum

a :	5	8	3	2	9	7

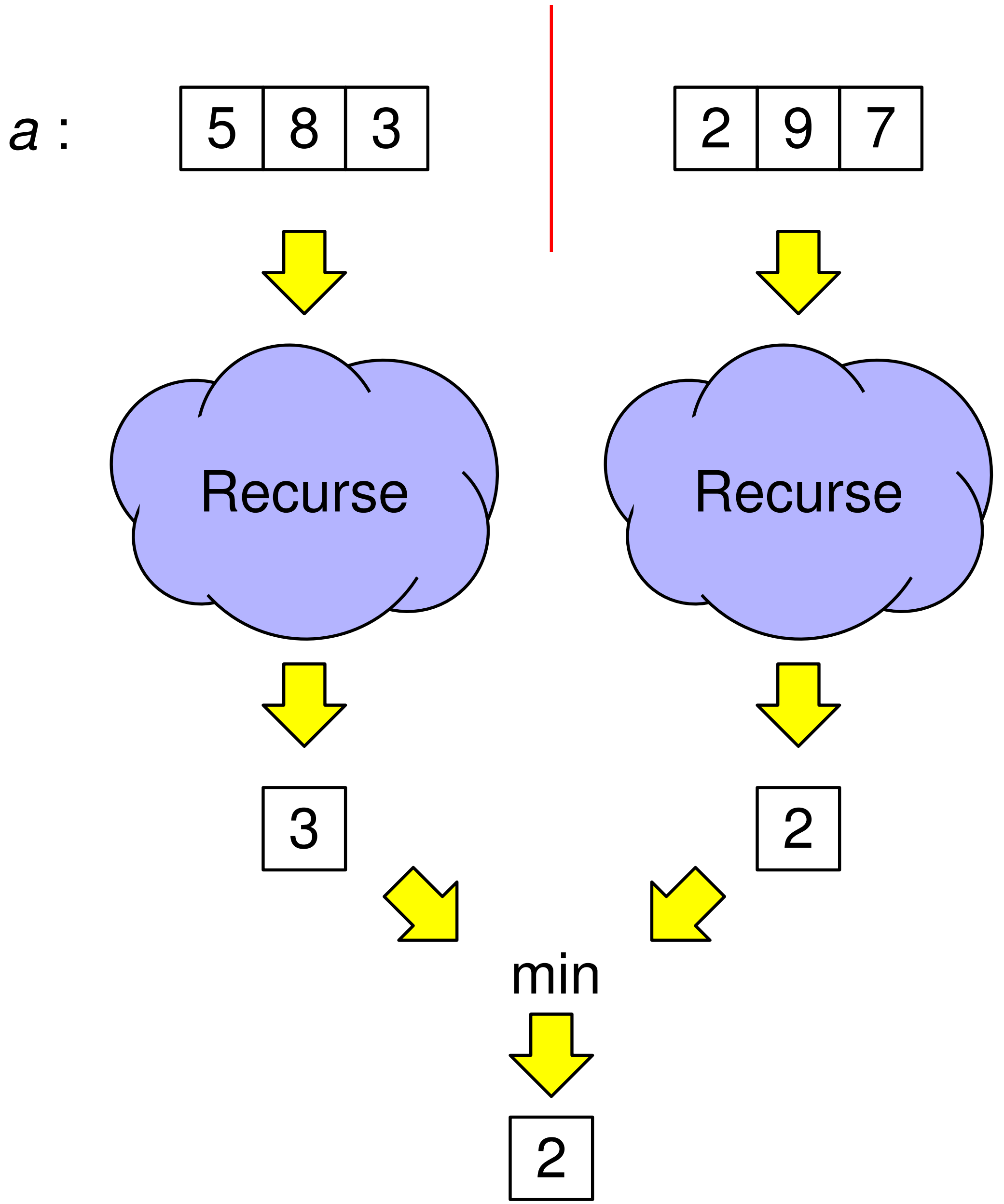
Simple example: Finding minimum



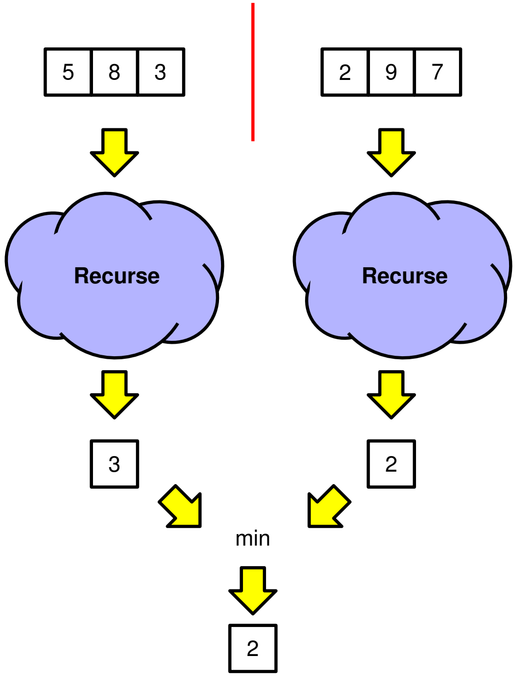
Simple example: Finding minimum



Simple example: Finding minimum

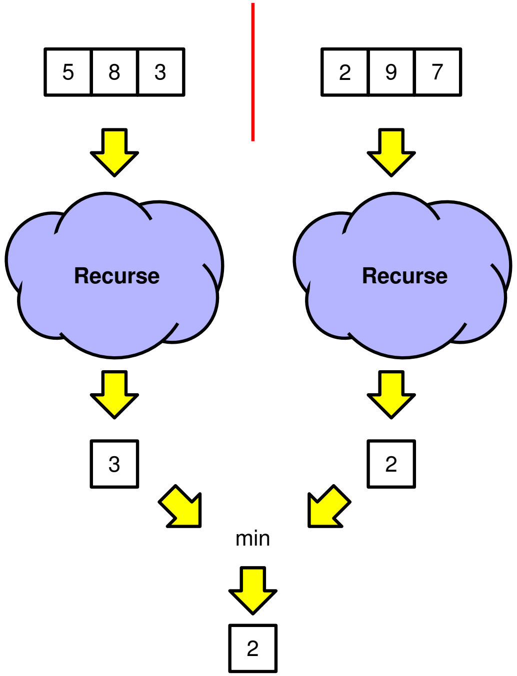


Finding Minimum



Finding Minimum

procedure MIN($a[i..j]$)

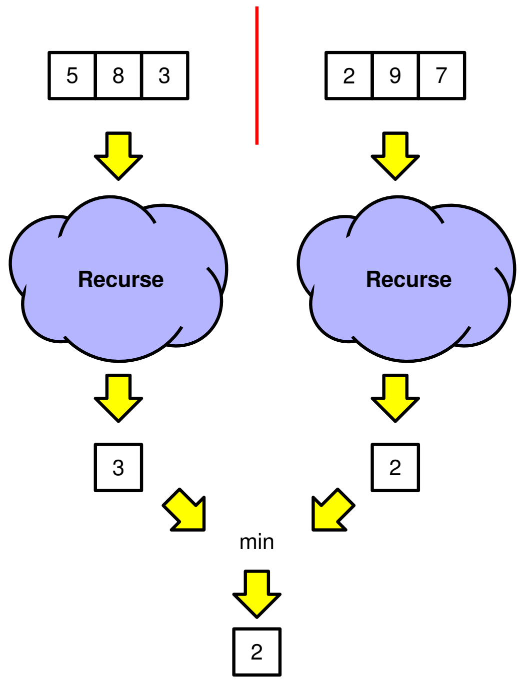


Finding Minimum

procedure $\text{MIN}(a[i..j])$

$$mid = \left\lfloor \frac{i+j}{2} \right\rfloor$$

$$left = \text{MIN}(a[i..mid])$$
$$right = \text{MIN}(a[mid+1..j])$$



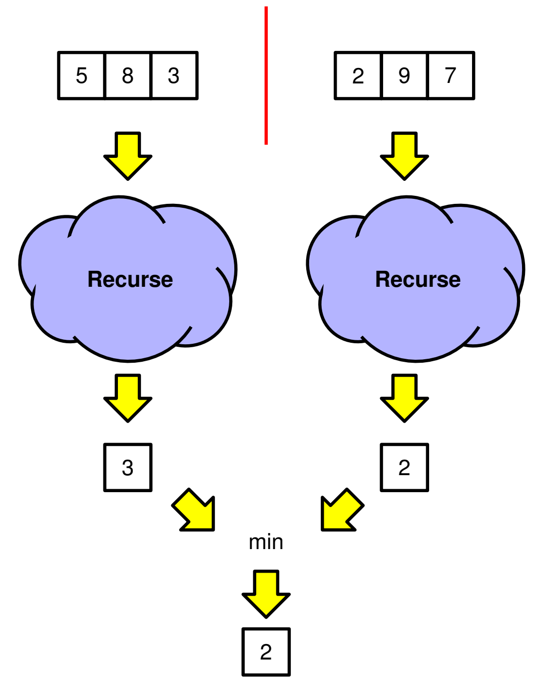
Finding Minimum

procedure $\textsc{Min}(a[i..j])$

$mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

$left = \textsc{Min}(a[i..mid])$
$right = \textsc{Min}(a[mid+1..j])$

return $\min(left, right)$



Finding Minimum

procedure $\text{MIN}(a[i..j])$

  if $i == j$ **then**

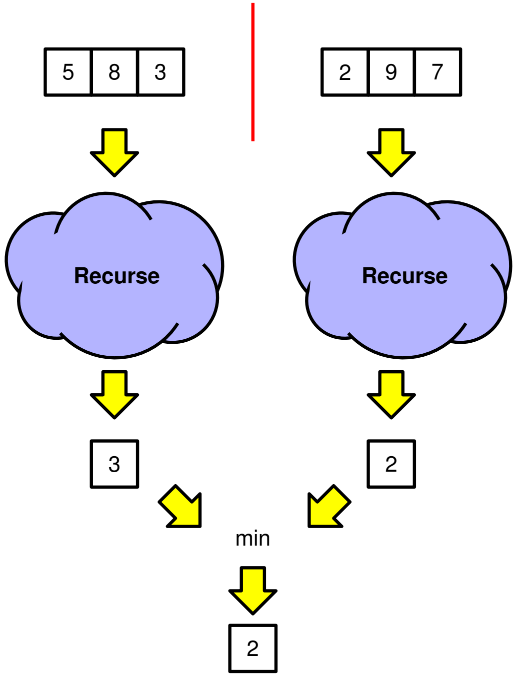
    return $a[i]$

  else

    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

    $left = \text{MIN}(a[i..mid])$
    $right = \text{MIN}(a[mid+1..j])$

    return $\min(left, right)$



Finding Minimum

procedure $\textsc{Min}(a[i..j])$

if $i == j$ **then**

return $a[i]$

else

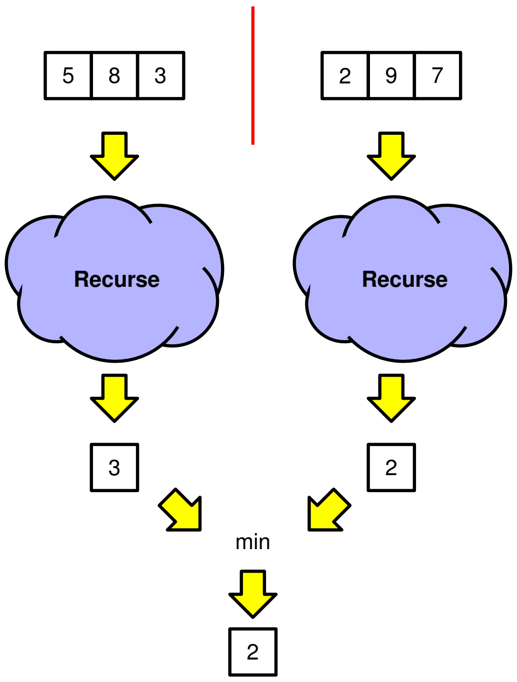
$mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

in parallel do

$left = \textsc{Min}(a[i..mid])$

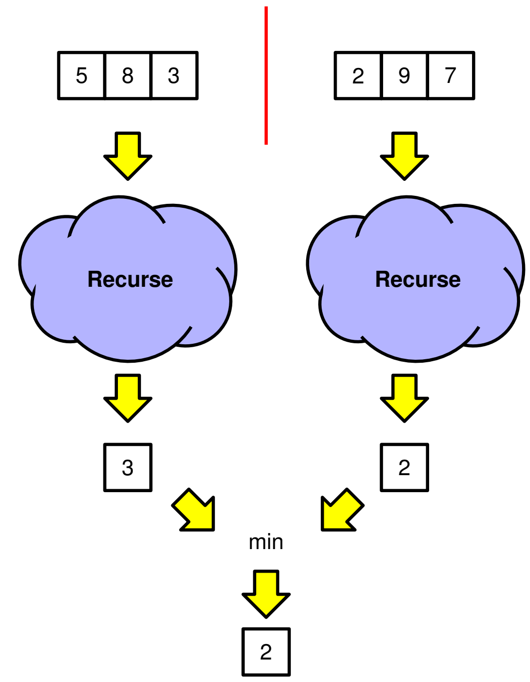
$right = \textsc{Min}(a[mid+1..j])$

return $\min(left, right)$



Finding Minimum

procedure $\textsc{Min}(a[i..j])$
  if $i == j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do
      $left = \textsc{Min}(a[i..mid])$
      $right = \textsc{Min}(a[mid+1..j])$
    return $\min(left, right)$



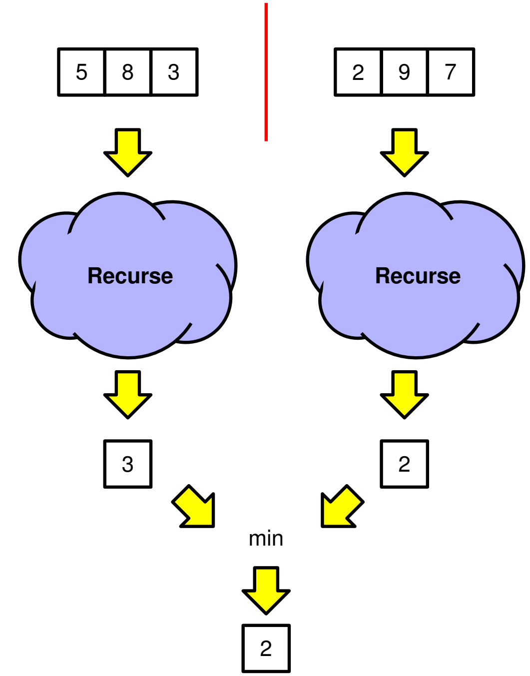
Runtime of $\textsc{Min}(a[1..n])$?

Finding Minimum

procedure MIN($a[i..j]$)
  if $i == j$ **then**
    return $a[i]$
  else

    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do
      $left = \text{MIN}(a[i..mid])$
      $right = \text{MIN}(a[mid+1..j])$
    return $\min(left, right)$

$T(n)$



Runtime of MIN($a[1..n]$)?

Finding Minimum

procedure $\textsc{Min}(a[i..j])$

  if $i == j$ **then**

    return $a[i]$

  else

    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

    in parallel do

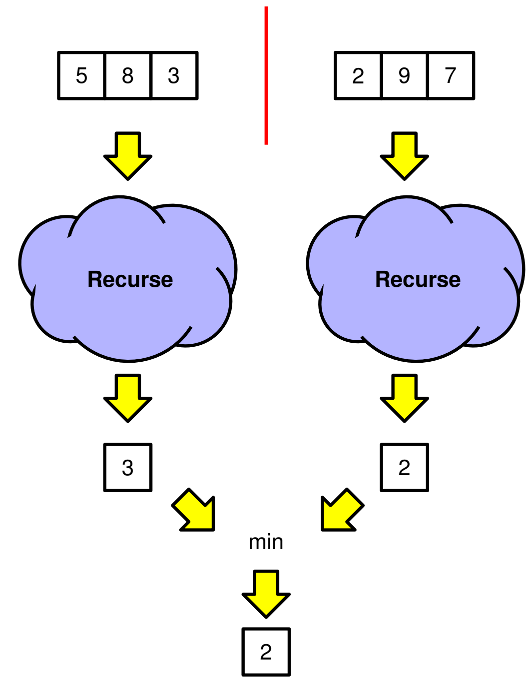
      $left = \textsc{Min}(a[i..mid])$

      $right = \textsc{Min}(a[mid+1..j])$

    return $\min(left, right)$

$O(1)$

$T(n)$



Runtime of $\textsc{Min}(a[1..n])$?

Finding Minimum

procedure $\text{MIN}(a[i..j])$

if $i == j$ **then**

  return $a[i]$

else

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  in parallel do

    $left = \text{MIN}(a[i..mid])$

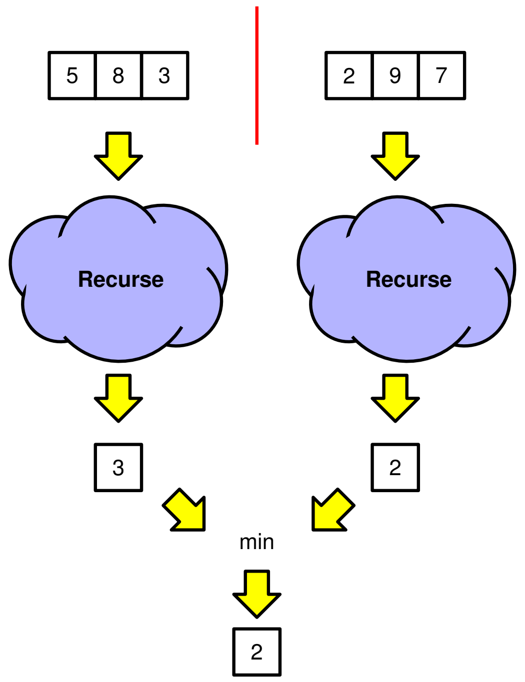
    $right = \text{MIN}(a[mid+1..j])$

  return $\min(left, right)$

$O(1)$

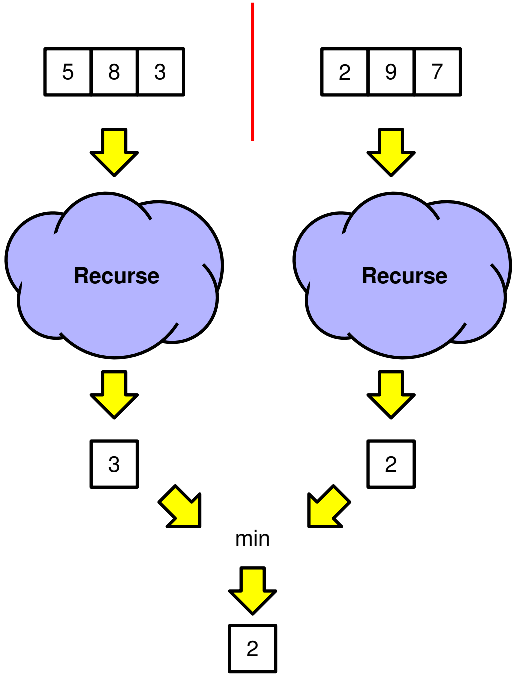
$T(n/2)$

$T(n)$



Runtime of $\text{MIN}(a[1..n])$?

Finding Minimum



procedure $\text{MIN}(a[i..j])$

  if $i == j$ **then**

      return $a[i]$

  else

      $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

      in parallel do

          $left = \text{MIN}(a[i..mid])$ $T(n/2)$

          $right = \text{MIN}(a[mid+1..j])$ $T(n/2)$

      return $\min(left, right)$

$O(1)$

$T(n)$

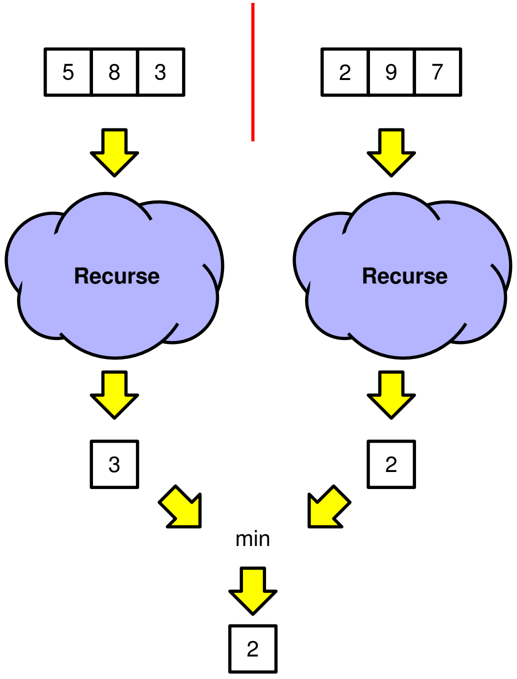
Runtime of $\text{MIN}(a[1..n])$?

Finding Minimum

procedure $\textsc{Min}(a[i..j])$
  if $i == j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do
      $left = \textsc{Min}(a[i..mid])$ — $T(n/2)$
      $right = \textsc{Min}(a[mid+1..j])$ — $T(n/2)$
    return $\min(left, right)$ — $O(1)$

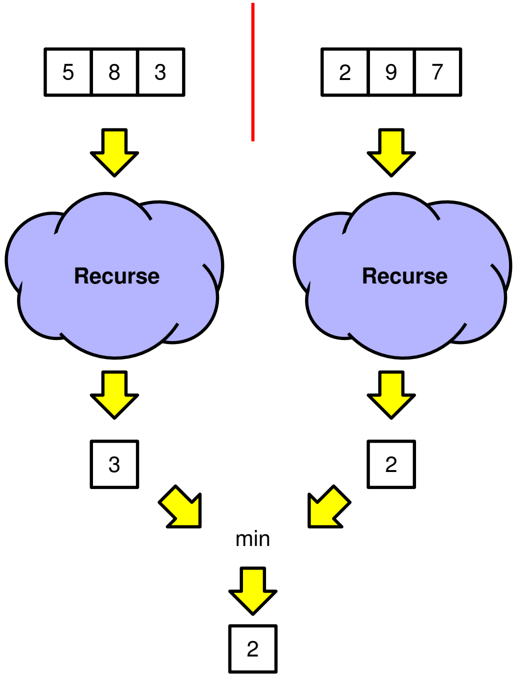
$O(1)$

$T(n)$



Runtime of $\textsc{Min}(a[1..n])$?

Finding Minimum



procedure $\textsc{Min}(a[i..j])$

if $i == j$ **then**

return $a[i]$

else

$mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

in parallel do

$left = \textsc{Min}(a[i..mid])$ $T(n/2)$

$right = \textsc{Min}(a[mid+1..j])$ $T(n/2)$

return $\min(left, right)$ $O(1)$

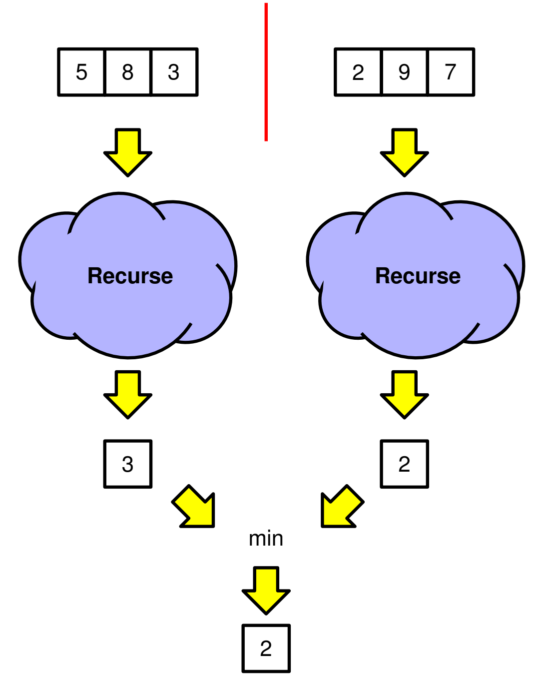
$O(1)$

$T(n)$

Runtime of $\textsc{Min}(a[1..n])$?

$$T(n) = O(1) + \max \left\{ \begin{array}{c} T(n/2) \\ T(n/2) \end{array} \right\} + O(1)$$

Finding Minimum



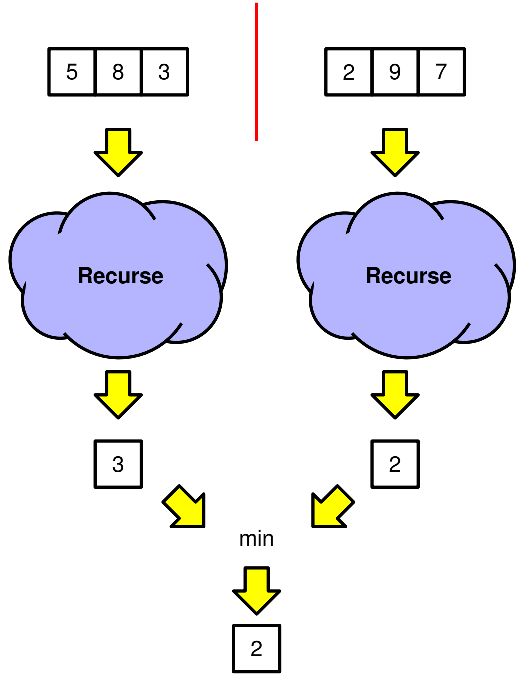
procedure $\textsc{Min}(a[i..j])$
  if $i == j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do
      $left = \textsc{Min}(a[i..mid])$
      $right = \textsc{Min}(a[mid+1..j])$
    return $\min(left, right)$

$O(1)$ — lines from **if** through mid
$T(n/2)$ — $left$
$T(n/2)$ — $right$
$O(1)$ — **return** $\min(left, right)$
$T(n)$ — whole procedure

Runtime of $\textsc{Min}(a[1..n])$?

$$T(n) = O(1) + \max\left\{\begin{array}{c} T(n/2) \\ T(n/2) \end{array}\right\} + O(1)$$
$$= T(n/2) + O(1)$$

Finding Minimum



procedure $\textsc{Min}(a[i..j])$
  if $i == j$ **then**
      return $a[i]$
  else
      $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$ — $O(1)$
      in parallel do
          $left = \textsc{Min}(a[i..mid])$ — $T(n/2)$
          $right = \textsc{Min}(a[mid+1..j])$ — $T(n/2)$
      return $\min(left, right)$ — $O(1)$

$T(n)$

Runtime of $\textsc{Min}(a[1..n])$?

$$T(n) = O(1) + \max \left\{ \begin{array}{c} T(n/2) \\ T(n/2) \end{array} \right\} + O(1)$$

$$= T(n/2) + O(1)$$

$$= \Theta(\log n)$$

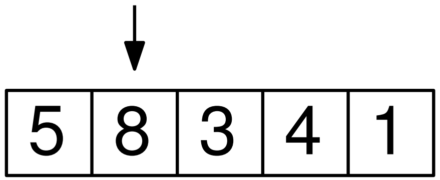
Not-so-simple example: Prefix Sums

for $i = 2$ to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	8	3	4	1

Not-so-simple example: Prefix Sums

for i = 2 to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$



Not-so-simple example: Prefix Sums

for $i = 2$ to n **do**

$\quad a[i] = a[i] + a[i-1]$

return $a[n]$

5	13	3	4	1

Not-so-simple example: Prefix Sums

for $i = 2$ to n **do**

    $a[i] = a[i] + a[i-1]$

return $a[n]$

5	13	3	4	1

Not-so-simple example: Prefix Sums

for $i = 2$ to n **do**

$a[i] = a[i] + a[i-1]$

return $a[n]$

5	13	16	4	1

Not-so-simple example: Prefix Sums

for i = 2 to n **do**

   $a[i] = a[i] + a[i-1]$

return $a[n]$

5	13	16	4	1

Not-so-simple example: Prefix Sums

for i = 2 to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	1

Not-so-simple example: Prefix Sums

for i = 2 to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

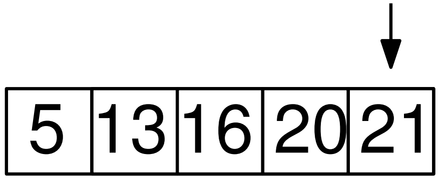
5	13	16	20	1

Not-so-simple example: Prefix Sums

for i = 2 to n **do**

  $a[i] = a[i] + a[i-1]$

return $a[n]$



Not-so-simple example: Prefix Sums

Time

for $i = 2$ to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	21

$O(n)$

Not-so-simple example: Prefix Sums

5	8	3	4	1

Time

for $i = 2$ to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	21

$\boxed{O(n)}$

Not-so-simple example: Prefix Sums

5	8	3	4	1

Time

for $i = 2$ to n **do**
  $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	21

$\boxed{O(n)}$

for $i = 2$ to n **in parallel do**
  $a[i] = a[i] + a[i-1]$
return $a[n]$

5	8	3	4	1

Not-so-simple example: Prefix Sums

5	8	3	4	1

Time

for i = 2 to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	21

$\boxed{O(n)}$

for i = 2 to n **in parallel do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	8	3	4	1

Not-so-simple example: Prefix Sums

5	8	3	4	1

Time

for $i = 2$ to n **do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	16	20	21

$\boxed{O(n)}$

for $i = 2$ to n **in parallel do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$

5	13	11	7	5

Not-so-simple example: Prefix Sums

5	8	3	4	1

Time

for i = 2 to n **do**
  $a[i] = a[i] + a[i - 1]$
return $a[n]$

5	13	16	20	21

$O(n)$

for i = 2 to n **in parallel do**
  $a[i] = a[i] + a[i - 1]$
return $a[n]$

5	13	11	7	5

$O(1)$

Parallel Prefix Sums

5	8	3	4	1

Parallel Prefix Sums

5	8	3	4	1

Parallel Prefix Sums

5	8	3	4	1

Parallel Prefix Sums

5	8	3	4	1

Parallel Prefix Sums

5	8	3	4	1

5	8	3

4	1

Parallel Prefix Sums

5	8	3	4	1

5	13	16

4	5

Parallel Prefix Sums

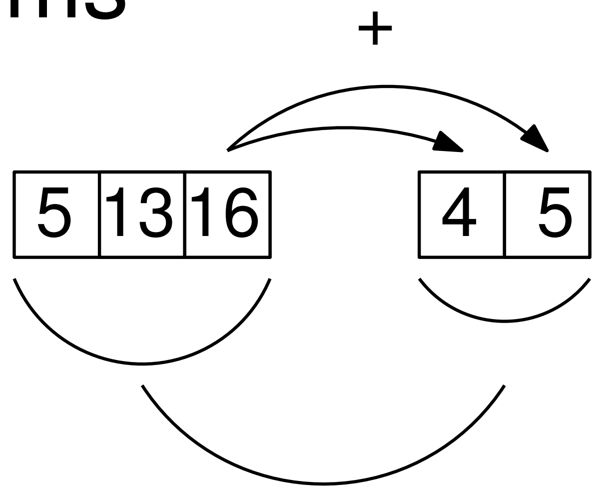
5	8	3	4	1

5	13	16

4	5

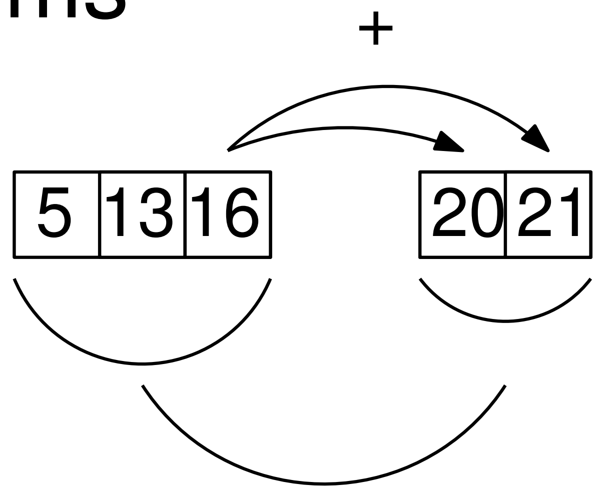
Parallel Prefix Sums

5	8	3	4	1

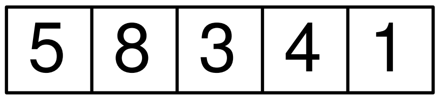


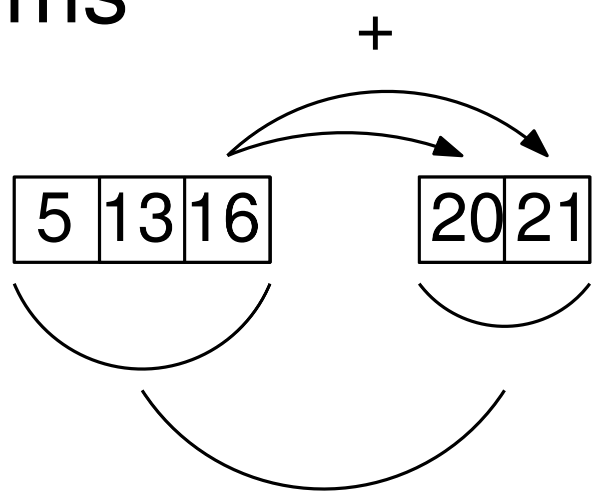
Parallel Prefix Sums

5	8	3	4	1



Parallel Prefix Sums





function PREFIX-SUMS(A, i, j)

  if $i \geq j$ **then return** ▷ Base case

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

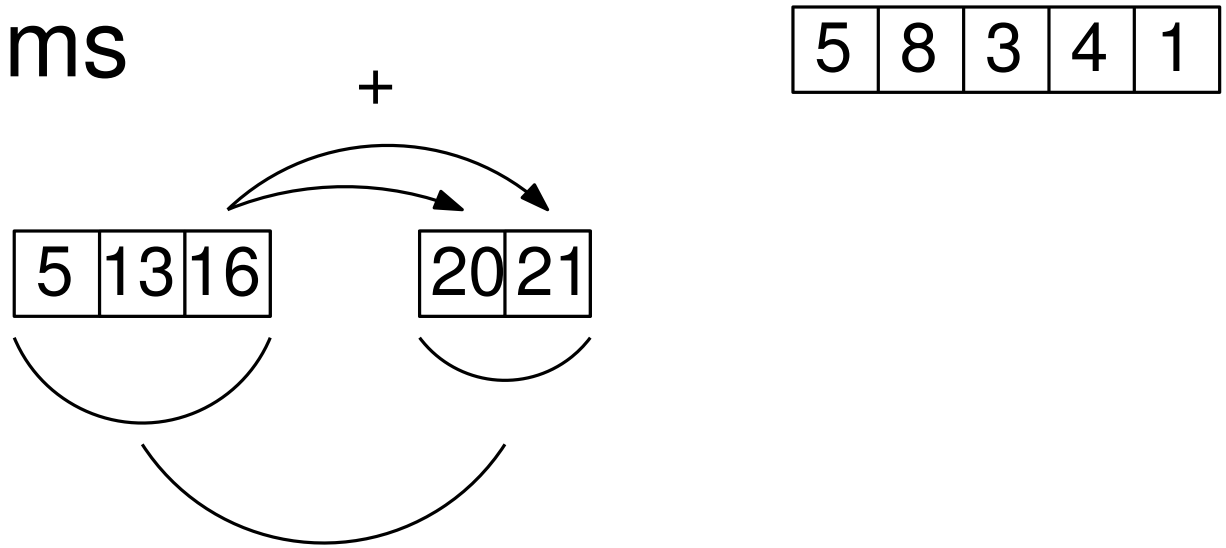
  PREFIX-SUMS(A, i, mid)

  PREFIX-SUMS($A, mid + 1, j$)

  for $k = mid + 1$ to j **do**

    $A[k] = A[k] + A[mid]$

Parallel Prefix Sums



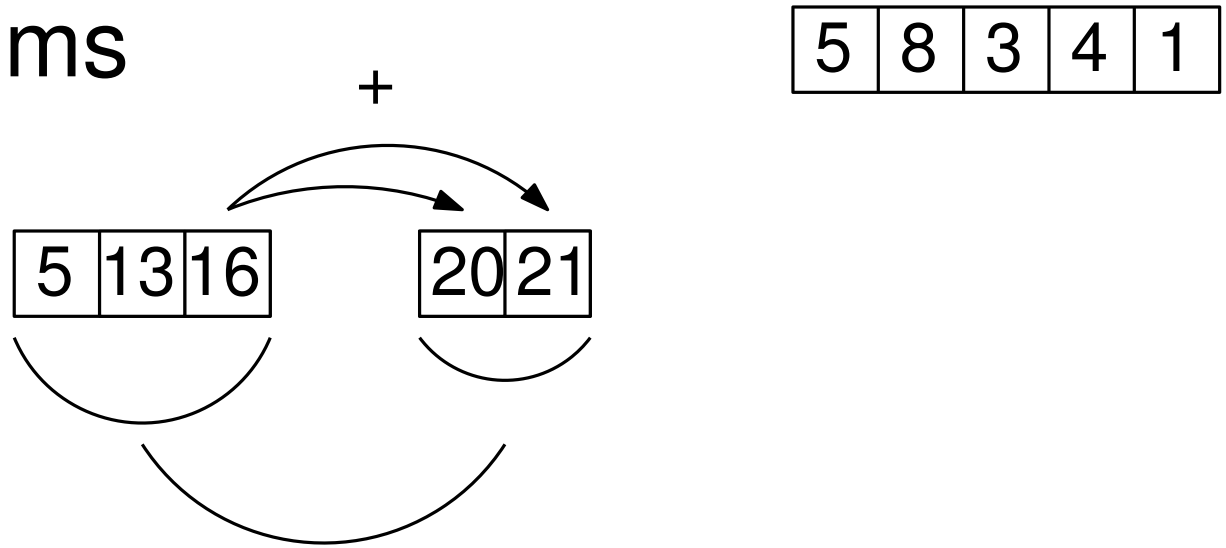
function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS($A, mid + 1, j$)

    for $k = mid + 1$ to j **do**
        $A[k] = A[k] + A[mid]$

$$\begin{aligned} T(n) &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Parallel Prefix Sums



function PREFIX-SUMS(A, i, j)

  if $i \geq j$ **then return** ▷ Base case

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  PREFIX-SUMS(A, i, mid)

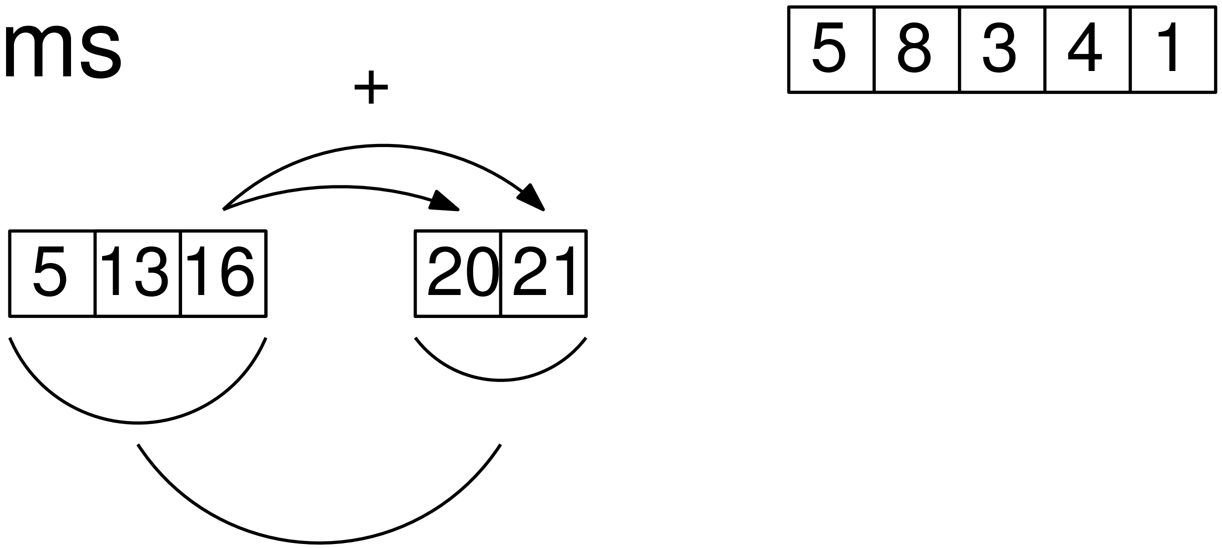
  PREFIX-SUMS($A, mid + 1, j$)

  for $k = mid + 1$ to j **do**

    $A[k] = A[k] + A[mid]$

$$\begin{aligned} W(n) &= 2W(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Parallel Prefix Sums



function PREFIX-SUMS(A, i, j)

  if $i \geq j$ **then return** ▷ Base case

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  PREFIX-SUMS(A, i, mid)

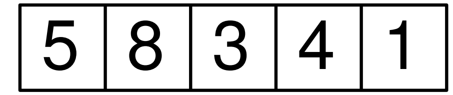
  PREFIX-SUMS($A, mid + 1, j$)

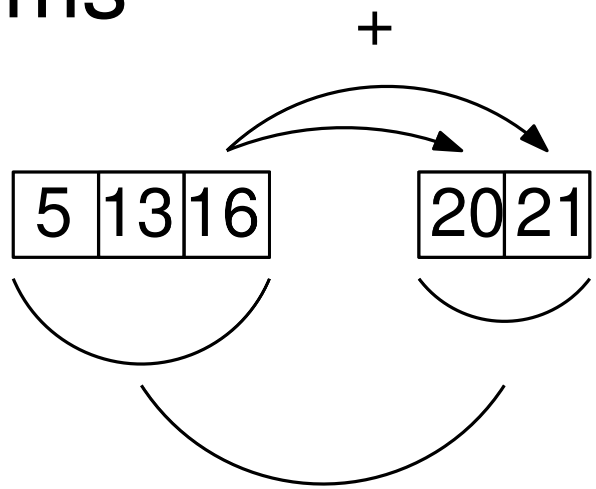
  for $k = mid + 1$ to j **in parallel do**

    $A[k] = A[k] + A[mid]$

$$\begin{aligned} W(n) &= 2W(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Parallel Prefix Sums





function PREFIX-SUMS(A, i, j)

  if $i \geq j$ **then return** ▷ Base case

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  PREFIX-SUMS(A, i, mid)

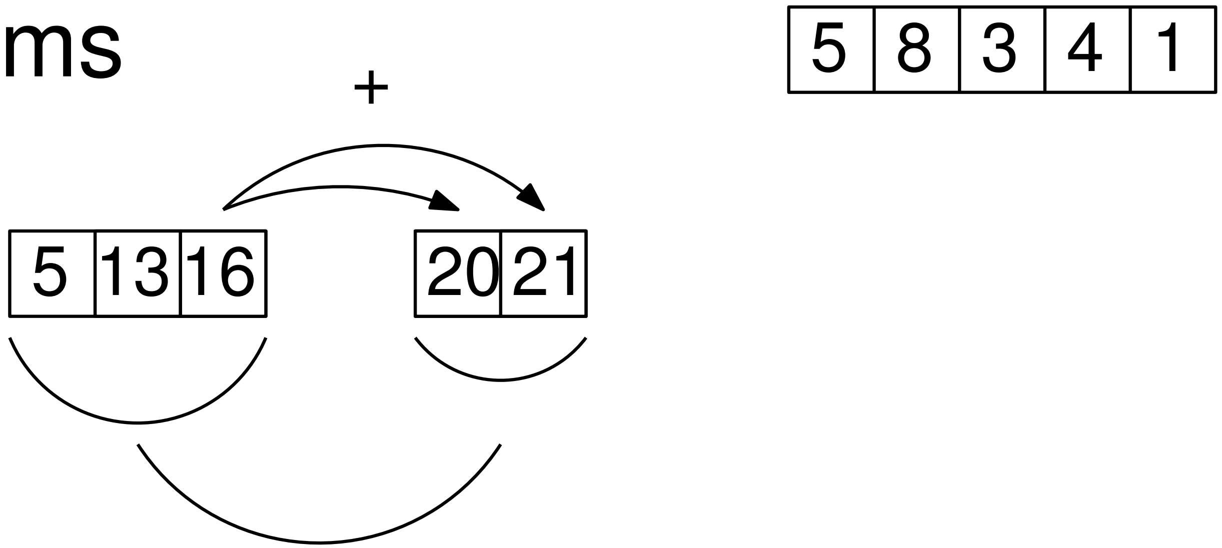
  PREFIX-SUMS($A, mid+1, j$)

  for $k = mid + 1$ to j **in parallel do**

    $A[k] = A[k] + A[mid]$

$$\begin{aligned} T(n) &= 2T(n/2) + O(1) \\ &= O(n) \end{aligned}$$

Parallel Prefix Sums

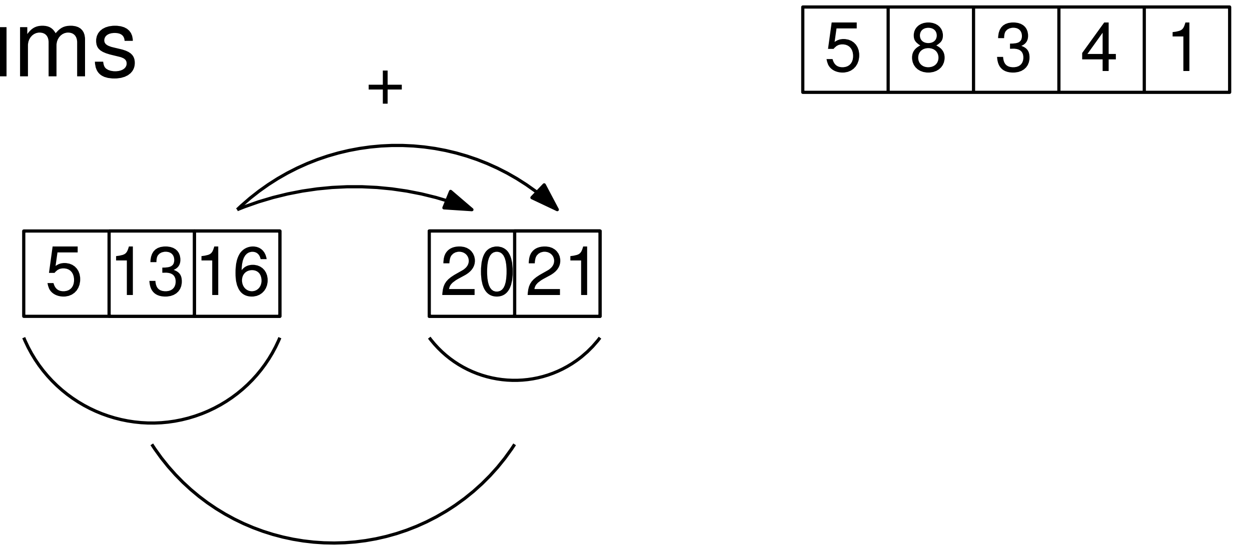


function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS($A, mid+1, j$)

$$\begin{aligned} T(n) &= 2T(n/2) + O(1) \\ &= O(n) \end{aligned}$$

    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

Parallel Prefix Sums



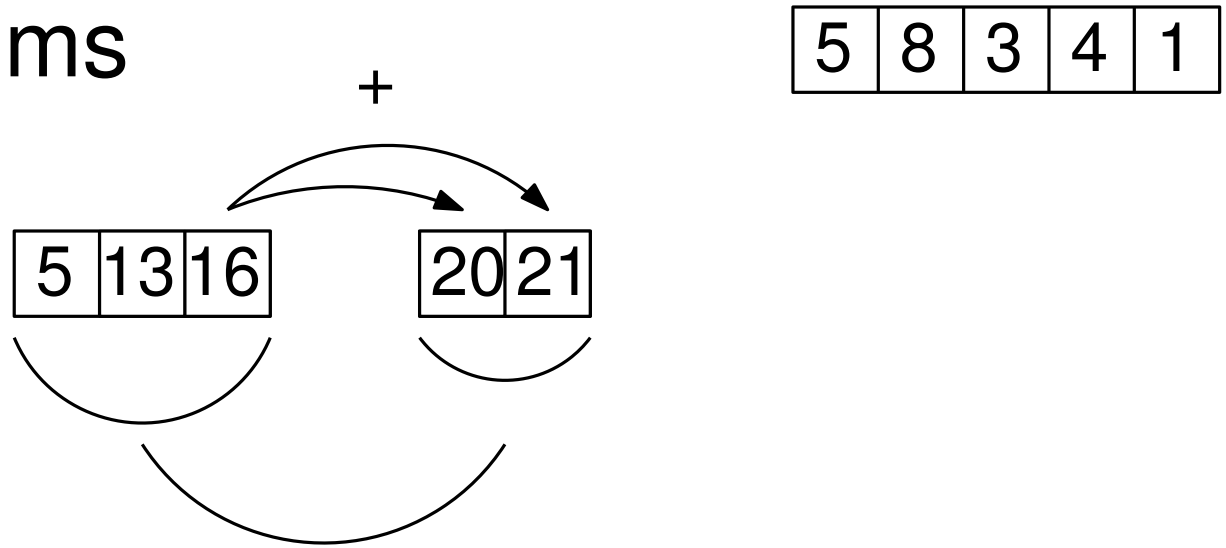
function $\textsc{Prefix-Sums}(A, i, j)$
    if $i \geq j$ **then return**
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do
        $\textsc{Prefix-Sums}(A, i, mid)$
        $\textsc{Prefix-Sums}(A, mid + 1, j)$

    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

$(t_1, t_2) = \textsc{StartTwoThreads}()$
t_1 **do**: $\textsc{Prefix-Sums}(A, i, mid)$
t_2 **do**: $\textsc{Prefix-Sums}(A, mid + 1, j)$
$\textsc{WaitUntilFinished}(t_1, t_2)$

$$
\begin{aligned}
T(n) &= 2T(n/2) + O(1) \\
&= O(n)
\end{aligned}
$$

Parallel Prefix Sums



function $\textsc{Prefix-Sums}(A, i, j)$

  if $i \geq j$ **then return**

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  in parallel do

    $\textsc{Prefix-Sums}(A, i, mid)$

    $\textsc{Prefix-Sums}(A, mid + 1, j)$

  for $k = mid + 1$ to j **in parallel**

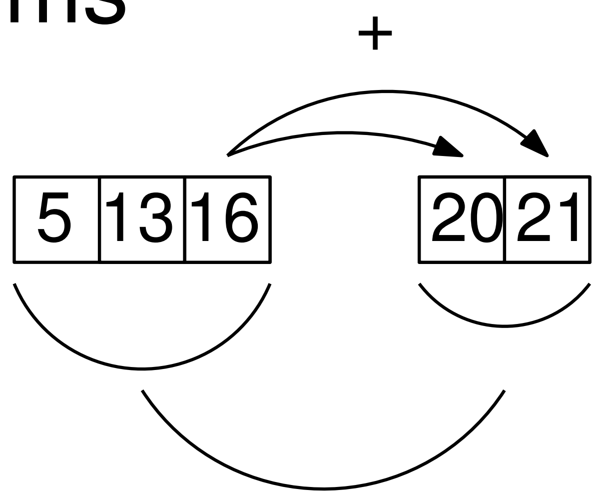
    $A[k] = A[k] + A[mid]$

$(t_1, t_2) = \textsc{StartTwoThreads}()$
t_1 **do**: $\textsc{Prefix-Sums}(A, i, mid)$
t_2 **do**: $\textsc{Prefix-Sums}(A, mid + 1, j)$
$\textsc{WaitUntilFinished}(t_1, t_2)$

$$T(n) = \max\left\{T\left(\left\lceil \frac{n}{2} \right\rceil\right), T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right\} + O(1)$$
$$\leq \quad T(n/2) + O(1)$$

Parallel Prefix Sums

5	8	3	4	1



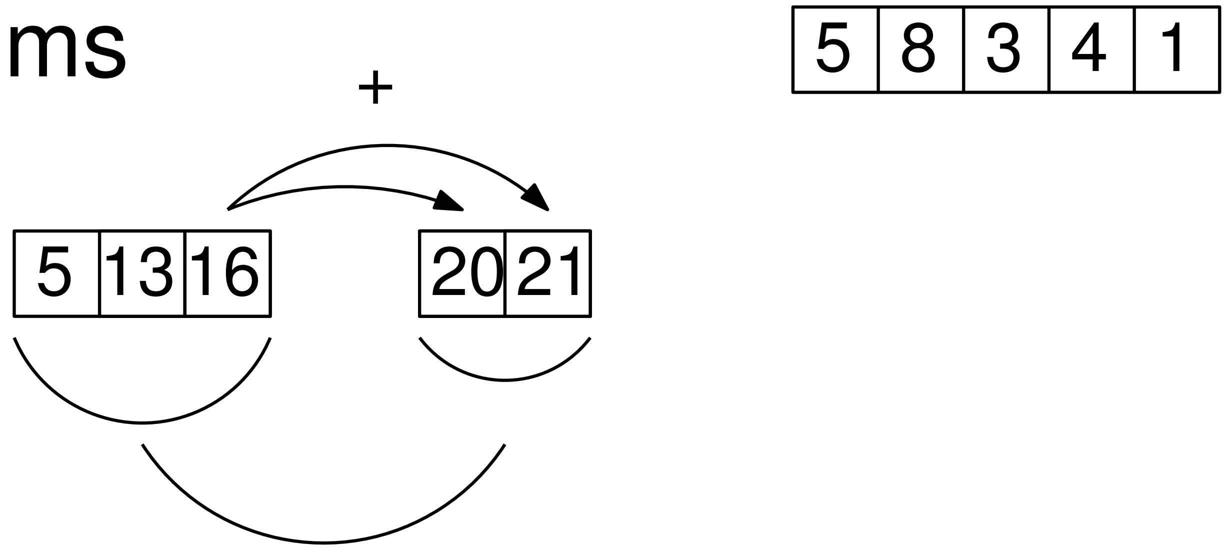
function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return**
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS($A, mid + 1, j$)

    for $k = mid + 1$ to j **in parallel**
        $A[k] = A[k] + A[mid]$

(t_1, t_2) = STARTTWOTHREADS()
t_1 **do**: PREFIX-SUMS(A, i, mid)
t_2 **do**: PREFIX-SUMS($A, mid + 1, j$)
WAITUNTILFINISHED(t_1, t_2)

$$\begin{aligned} T(n) &= \max\left\{ T\left(\left\lceil \tfrac{n}{2} \right\rceil\right), T\left(\left\lfloor \tfrac{n}{2} \right\rfloor\right) \right\} + O(1) \\ &\leq T(n/2) + O(1) \\ &= O(\log n) \end{aligned}$$

Parallel Prefix Sums



function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return**
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS($A, mid + 1, j$)

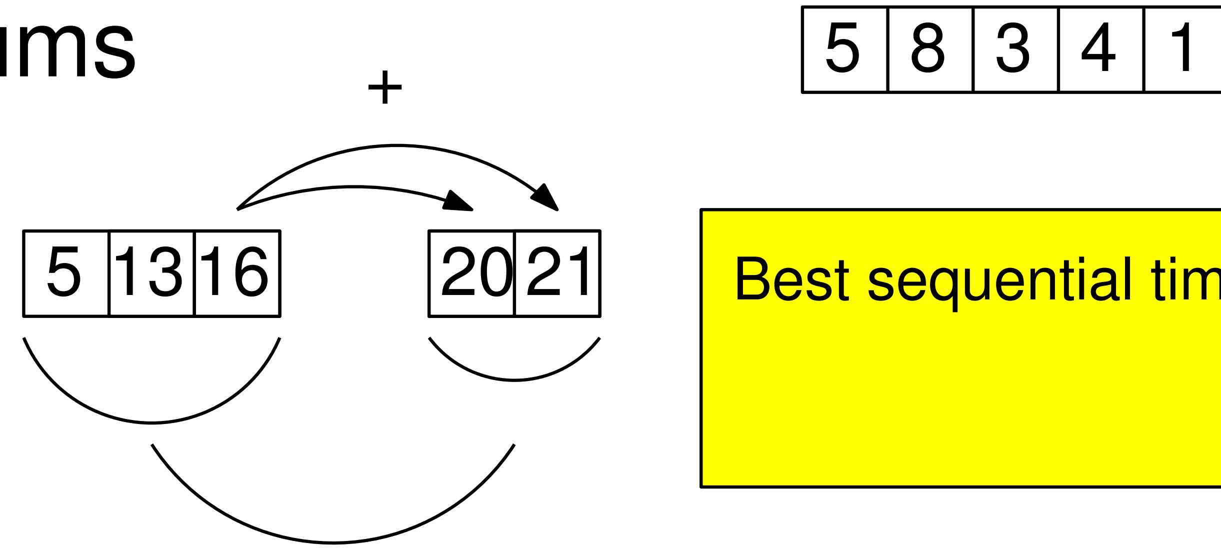
    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

(t_1, t_2) = STARTTWOTHREADS()
t_1 **do**: PREFIX-SUMS(A, i, mid)
t_2 **do**: PREFIX-SUMS($A, mid + 1, j$)
WAITUNTILFINISHED(t_1, t_2)

Work: $W(n) = O(n \log n)$

Time: $T(n) = O(\log n)$

Parallel Prefix Sums



Best sequential time

function PREFIX-SUMS(A, i, j)

  if $i \geq j$ **then return**

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  in parallel do

    PREFIX-SUMS(A, i, mid)

    PREFIX-SUMS($A, mid + 1, j$)

  for $k = mid + 1$ to j **in parallel do**

    $A[k] = A[k] + A[mid]$

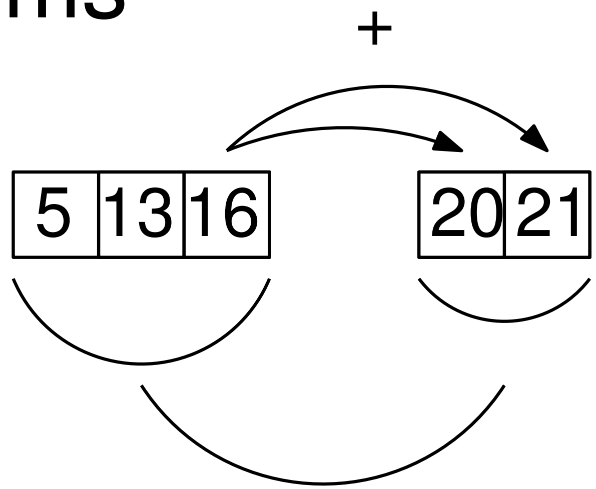
(t_1, t_2) = STARTTWOTHREADS()
t_1 **do**: PREFIX-SUMS(A, i, mid)
t_2 **do**: PREFIX-SUMS($A, mid + 1, j$)
WAITUNTILFINISHED(t_1, t_2)

Work: $W(n) = O(n \log n)$

Time: $T(n) = O(\log n)$

Parallel Prefix Sums

5	8	3	4	1



Best sequential time

$T(n) = O(n)$

function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return**
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS$(A, mid + 1, j)$

  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

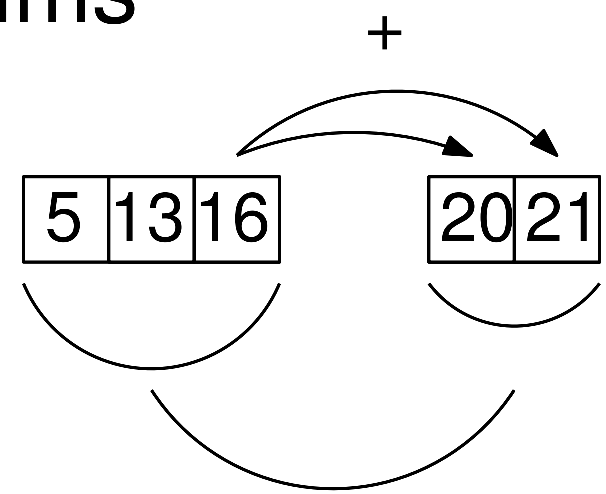
(t_1, t_2) = STARTTWOTHREADS()
t_1 **do**: PREFIX-SUMS(A, i, mid)
t_2 **do**: PREFIX-SUMS$(A, mid + 1, j)$
WAITUNTILFINISHED(t_1, t_2)

Work: $W(n) = O(n \log n)$

Time: $T(n) = O(\log n)$

Parallel Prefix Sums

5	8	3	4	1



Best sequential time

$$T(n) = O(n)$$

function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return**
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do
    PREFIX-SUM ... $, mid)$
    PREFIX ... $(A, mid + 1, j)$

Not work-efficient!

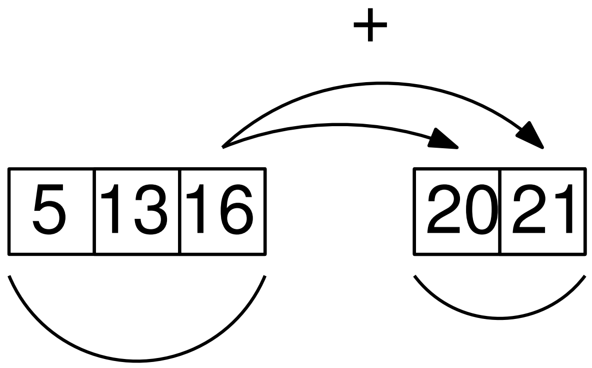
  for ... $mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

(t_1, t_2) = STARTTWOTHREADS()
t_1 **do**: PREFIX-SUMS(A, i, mid)
t_2 **do**: PREFIX-SUMS($A, mid + 1, j$)
WAITUNTILFINISHED(t_1, t_2)

Work: $W(n) = O(n \log n)$

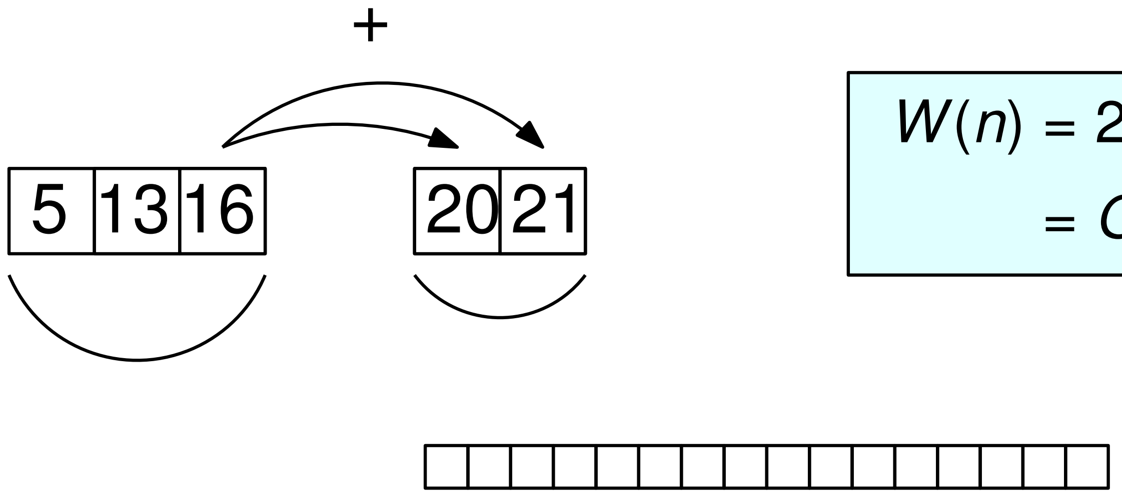
Time: $T(n) = O(\log n)$

Work-efficient Prefix Sums



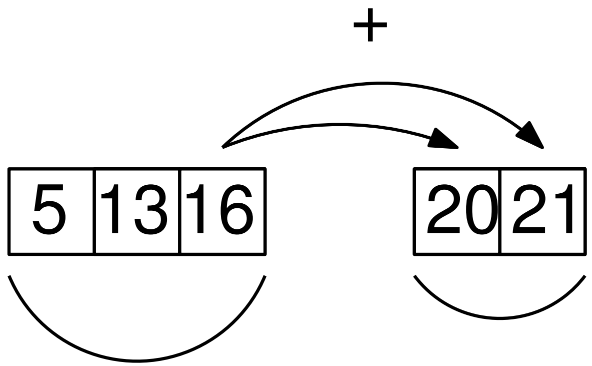
$$W(n) = 2W(n/2) + O(n) = O(n \log n)$$

Work-efficient Prefix Sums

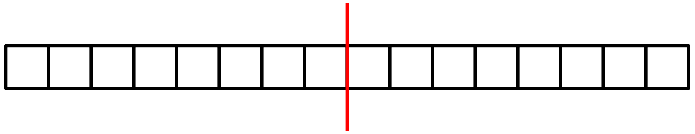


$$\boxed{\begin{aligned} W(n) &= 2W(n/2) + O(n) \\ &= O(n \log n) \end{aligned}}$$

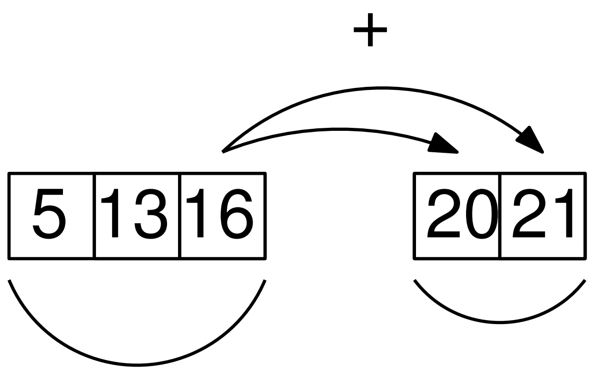
Work-efficient Prefix Sums



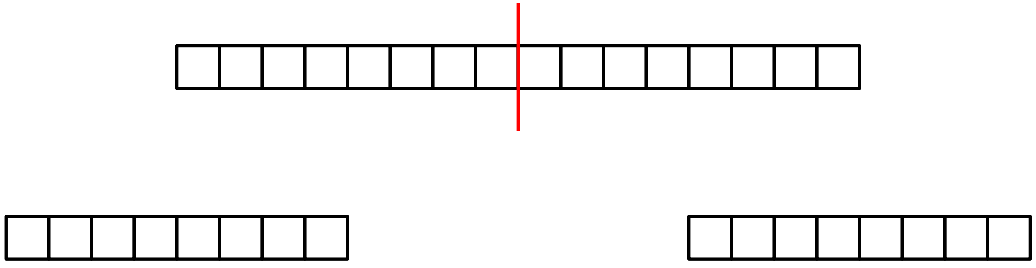
$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$



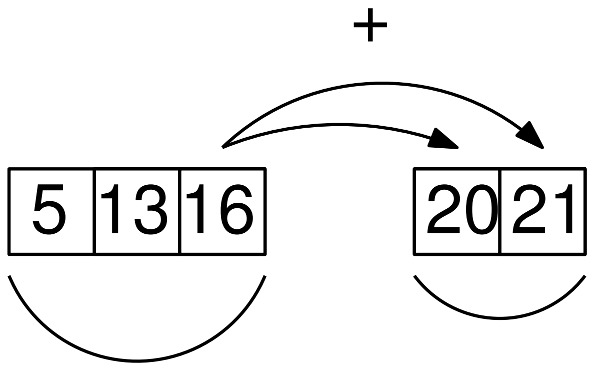
Work-efficient Prefix Sums



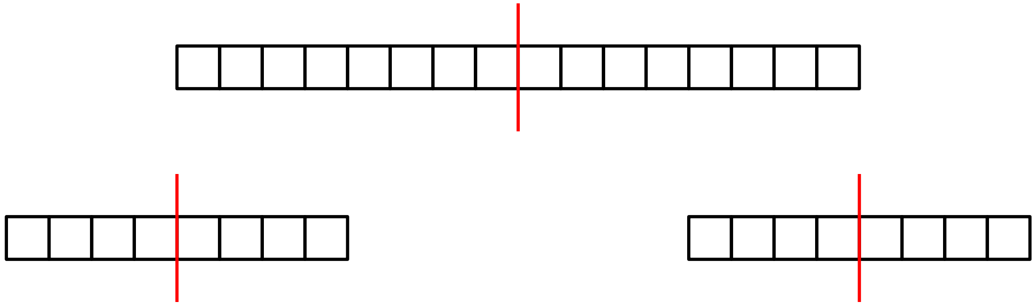
$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$



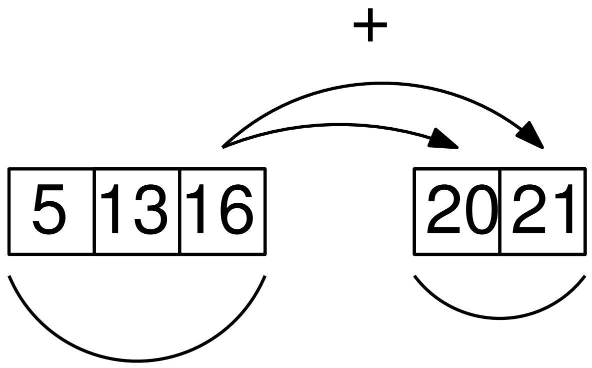
Work-efficient Prefix Sums



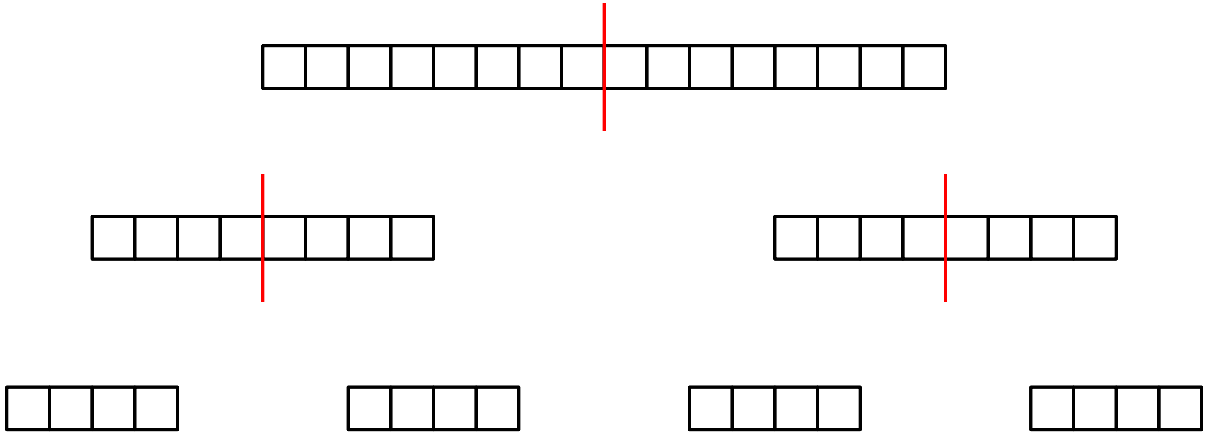
$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$



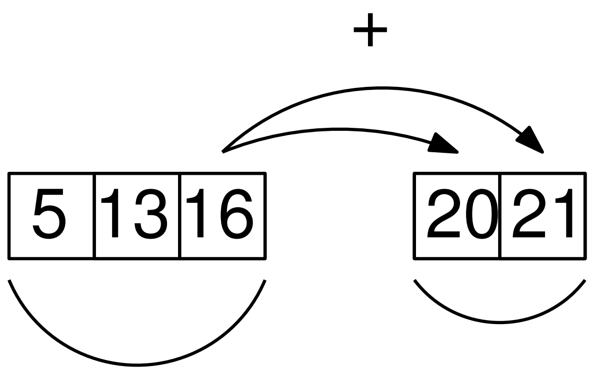
Work-efficient Prefix Sums



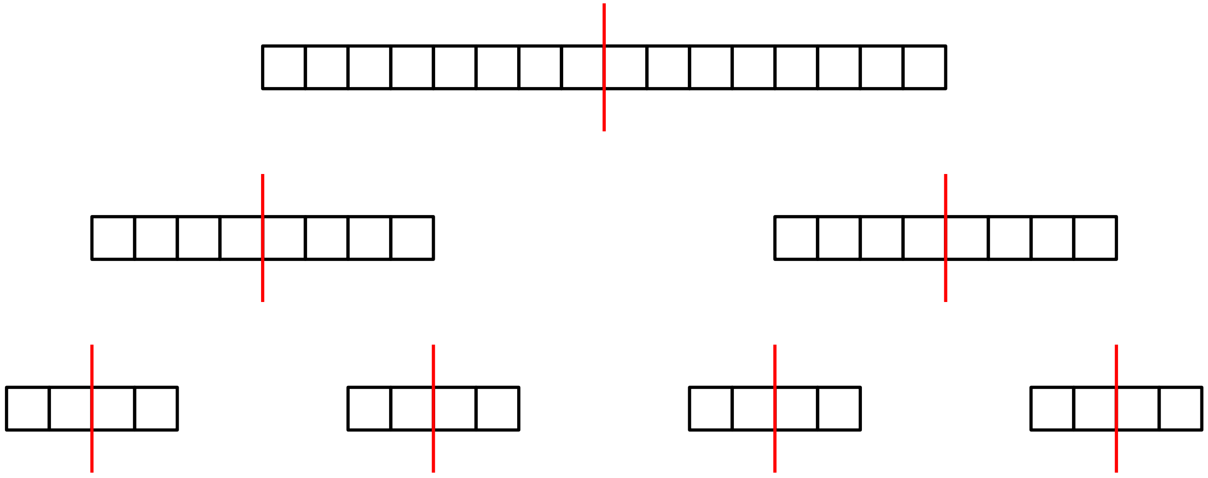
$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$



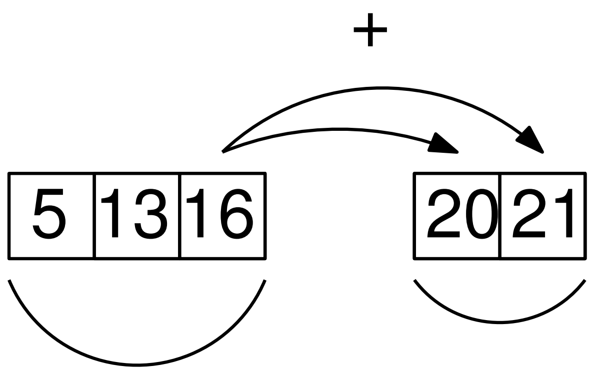
Work-efficient Prefix Sums



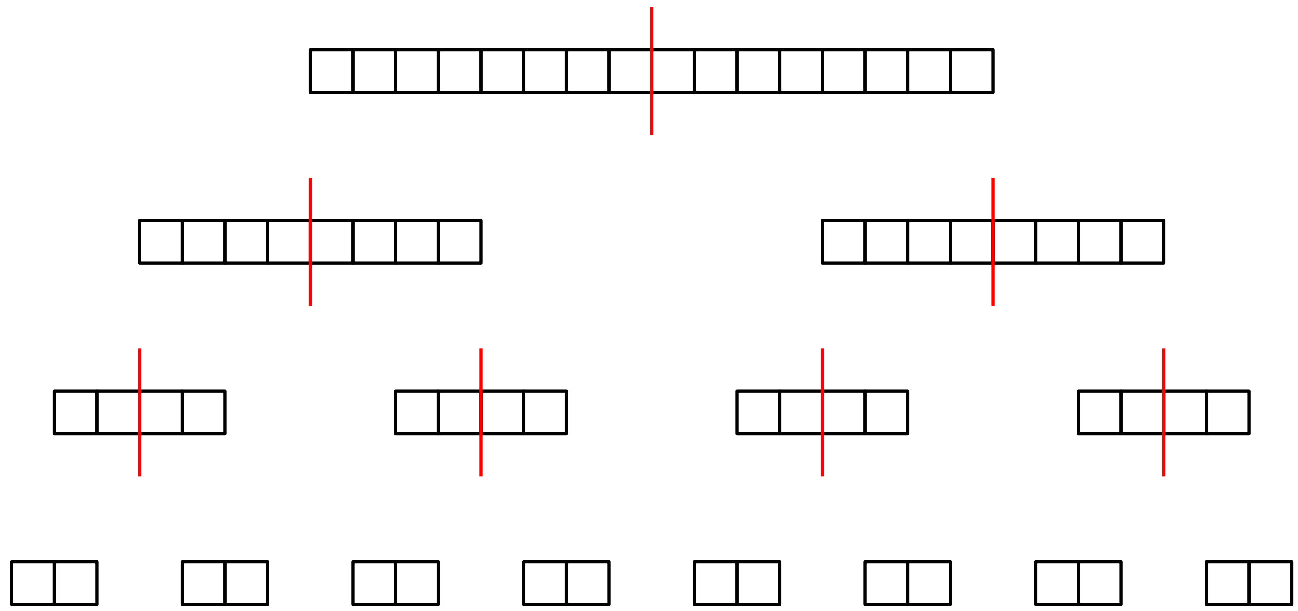
$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$



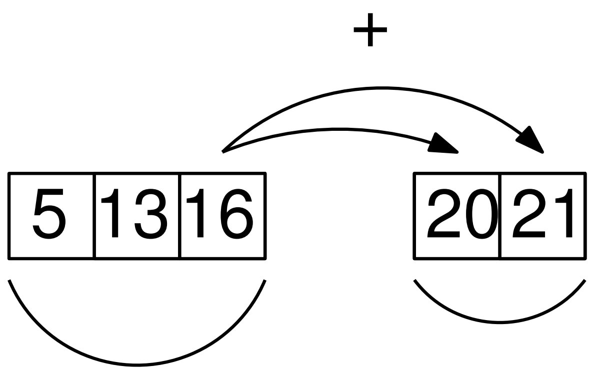
Work-efficient Prefix Sums



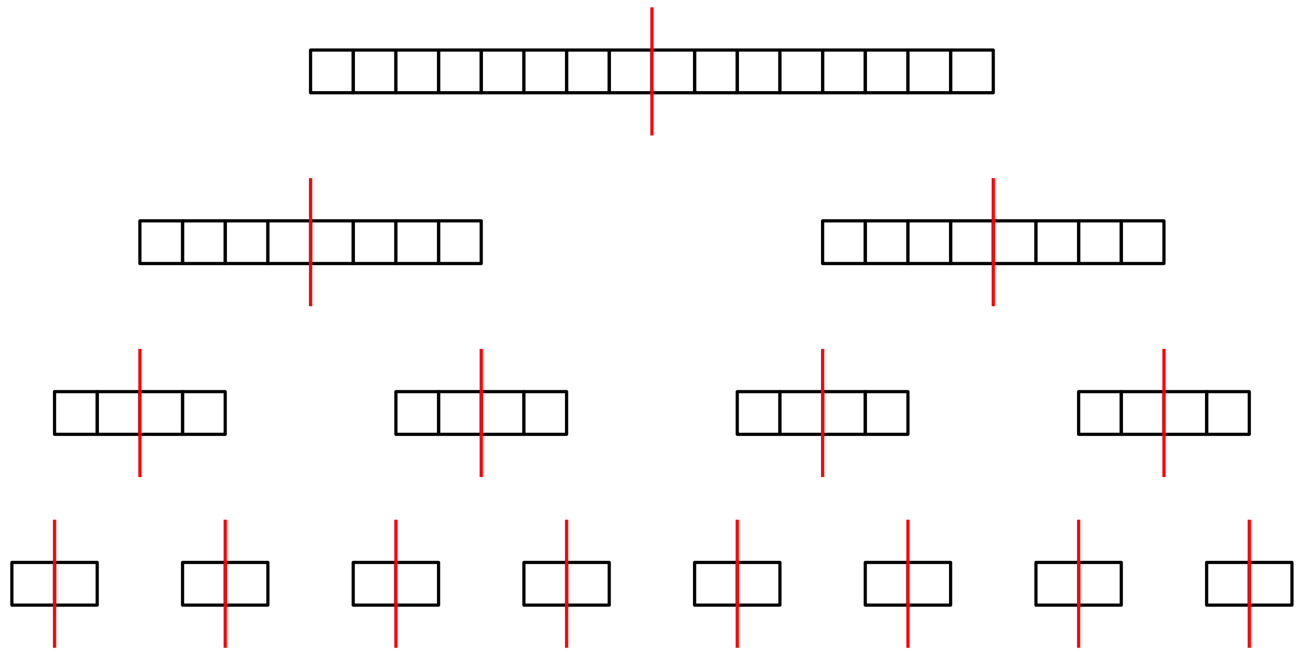
$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$



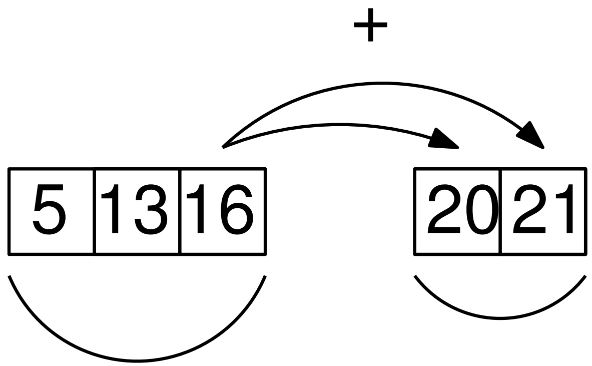
Work-efficient Prefix Sums



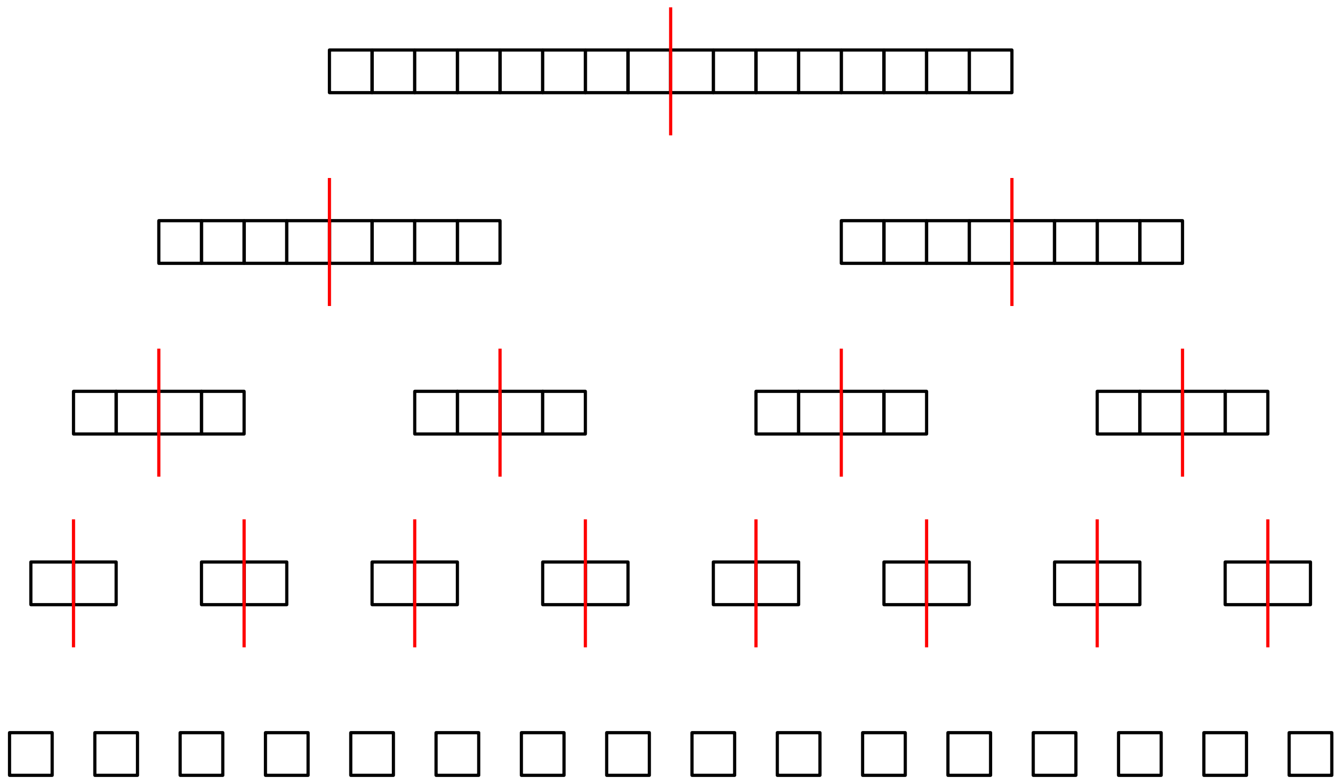
$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$



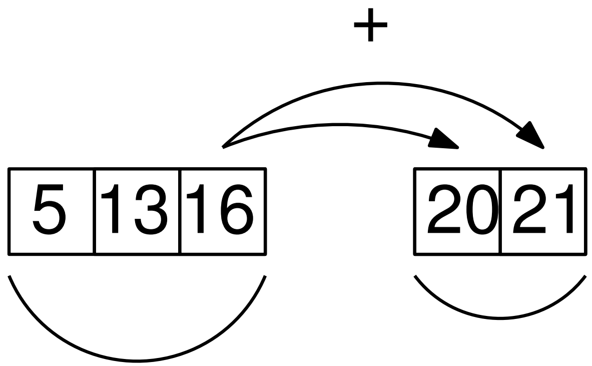
Work-efficient Prefix Sums



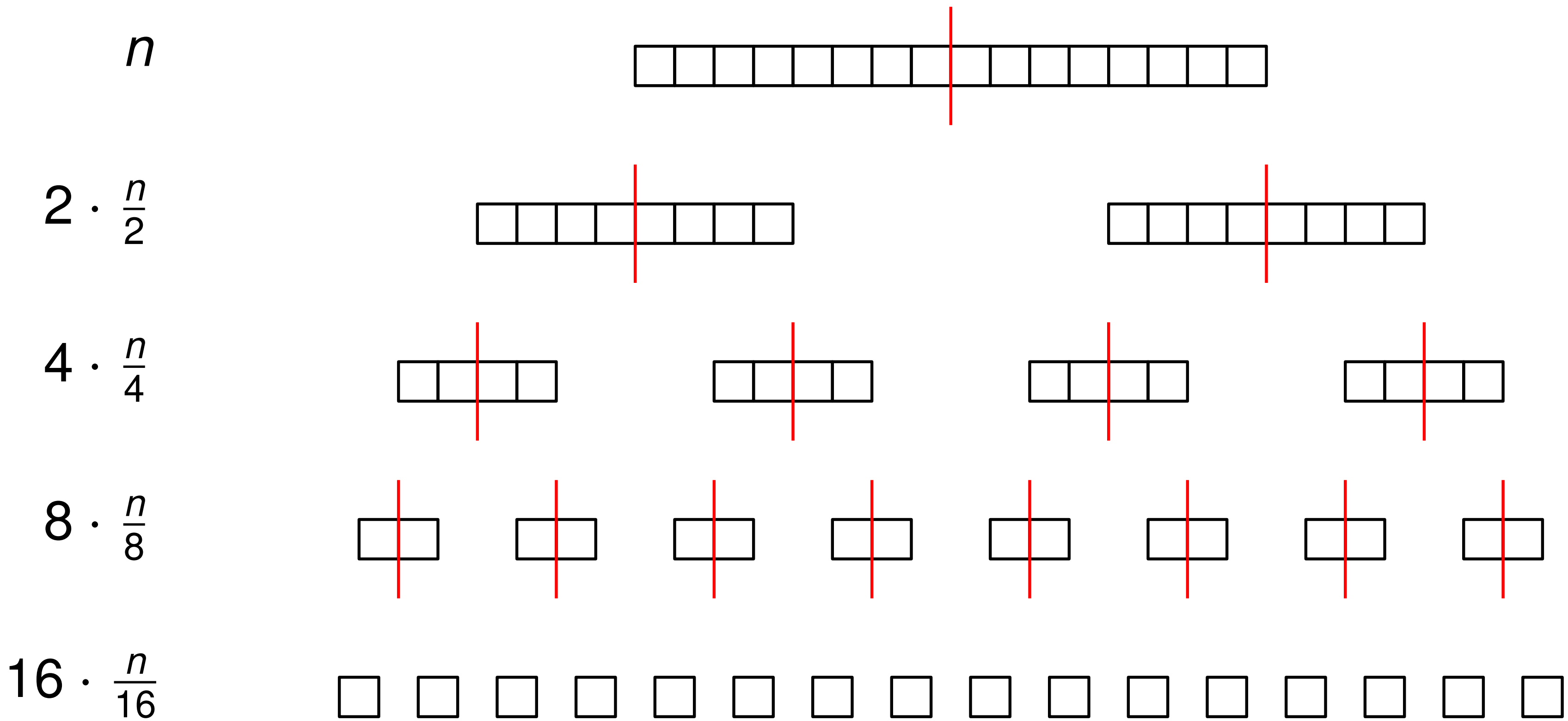
$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$



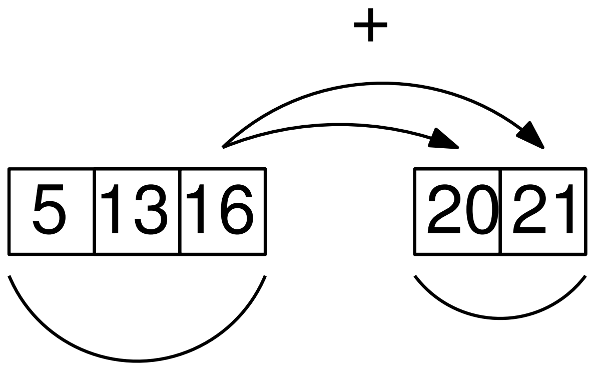
Work-efficient Prefix Sums



$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$

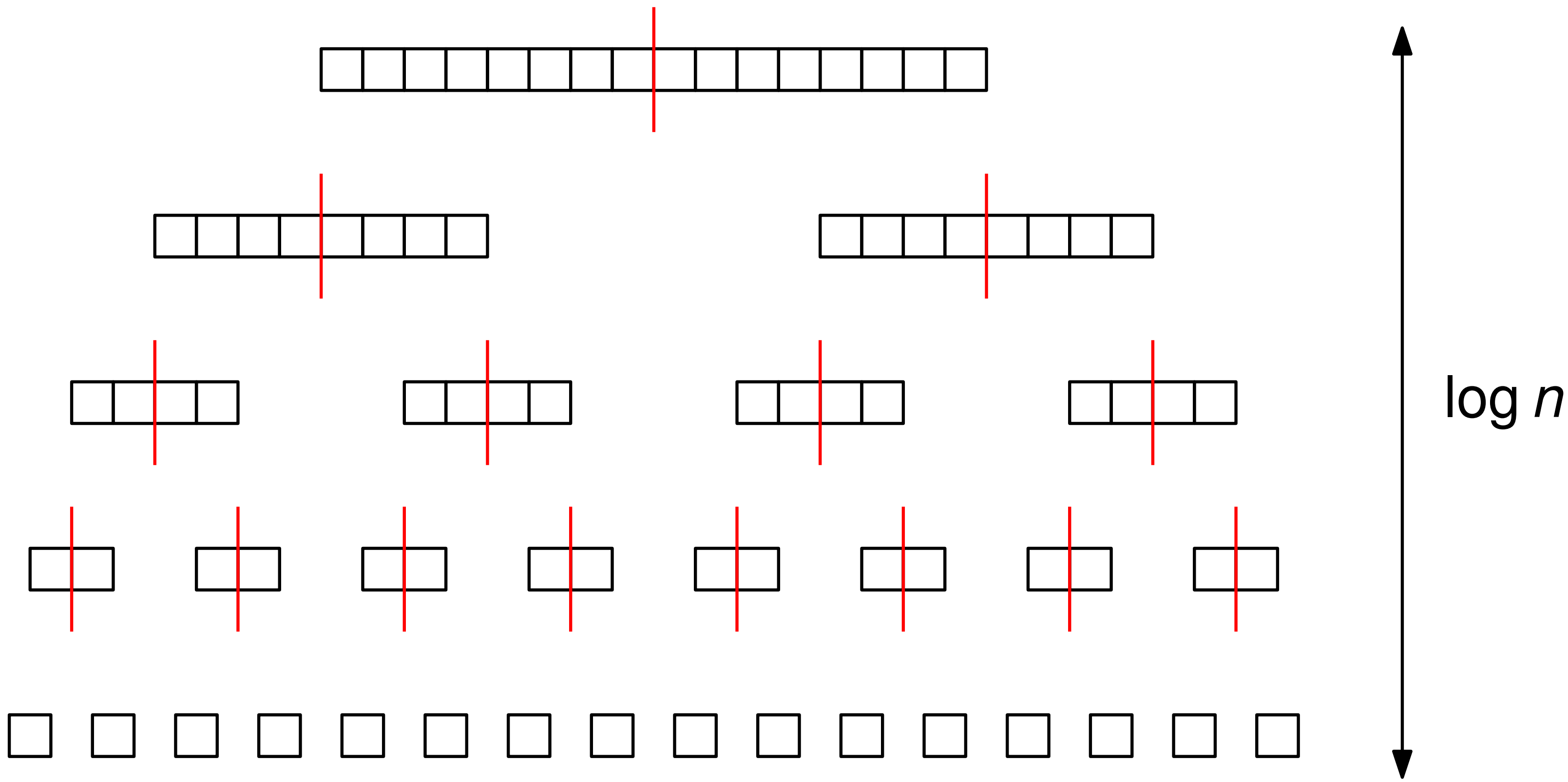


Work-efficient Prefix Sums

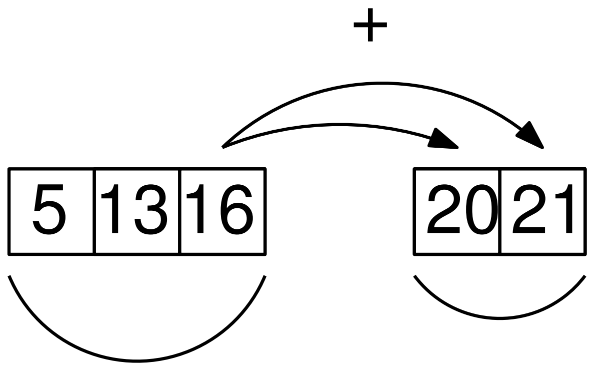


$$W(n) = 2W(n/2) + O(n)$$
$$= O(n \log n)$$

$$n = n$$
$$2 \cdot \tfrac{n}{2} = n$$
$$4 \cdot \tfrac{n}{4} = n$$
$$8 \cdot \tfrac{n}{8} = n$$
$$16 \cdot \tfrac{n}{16} = n$$



Work-efficient Prefix Sums



$$W(n) = 2W(n/2) + O(n) = O(n \log n)$$

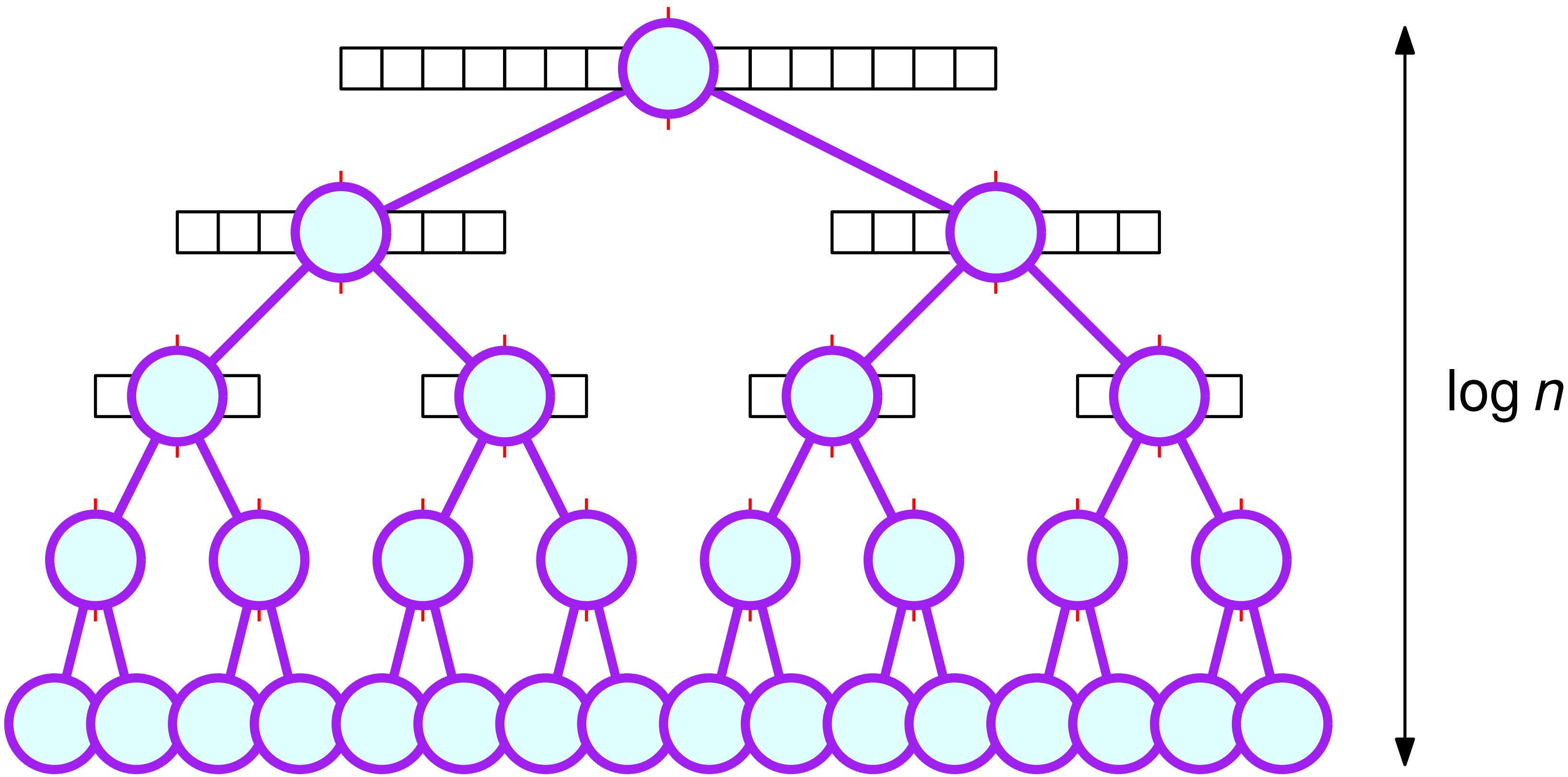
$$n = n$$

$$2 \cdot \tfrac{n}{2} = n$$

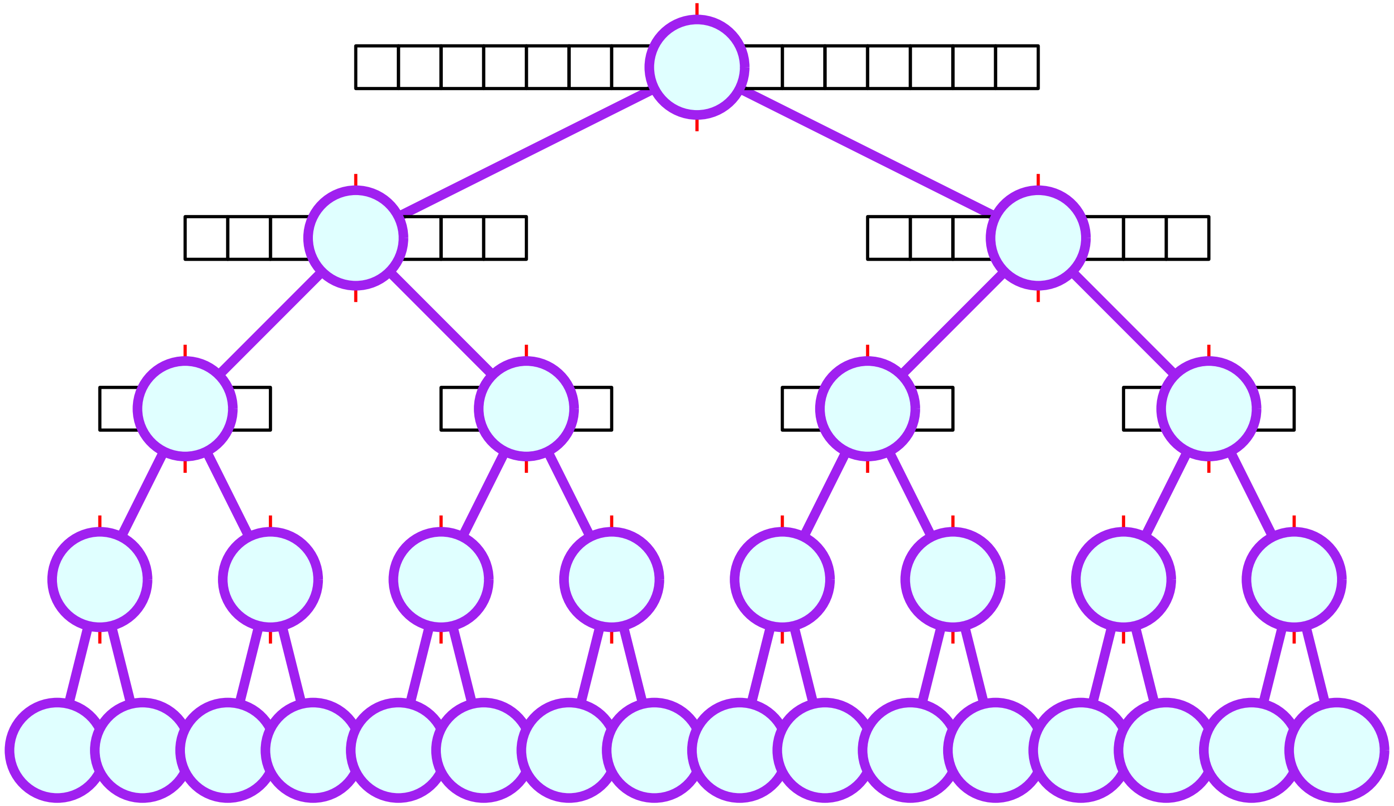
$$4 \cdot \tfrac{n}{4} = n$$

$$8 \cdot \tfrac{n}{8} = n$$

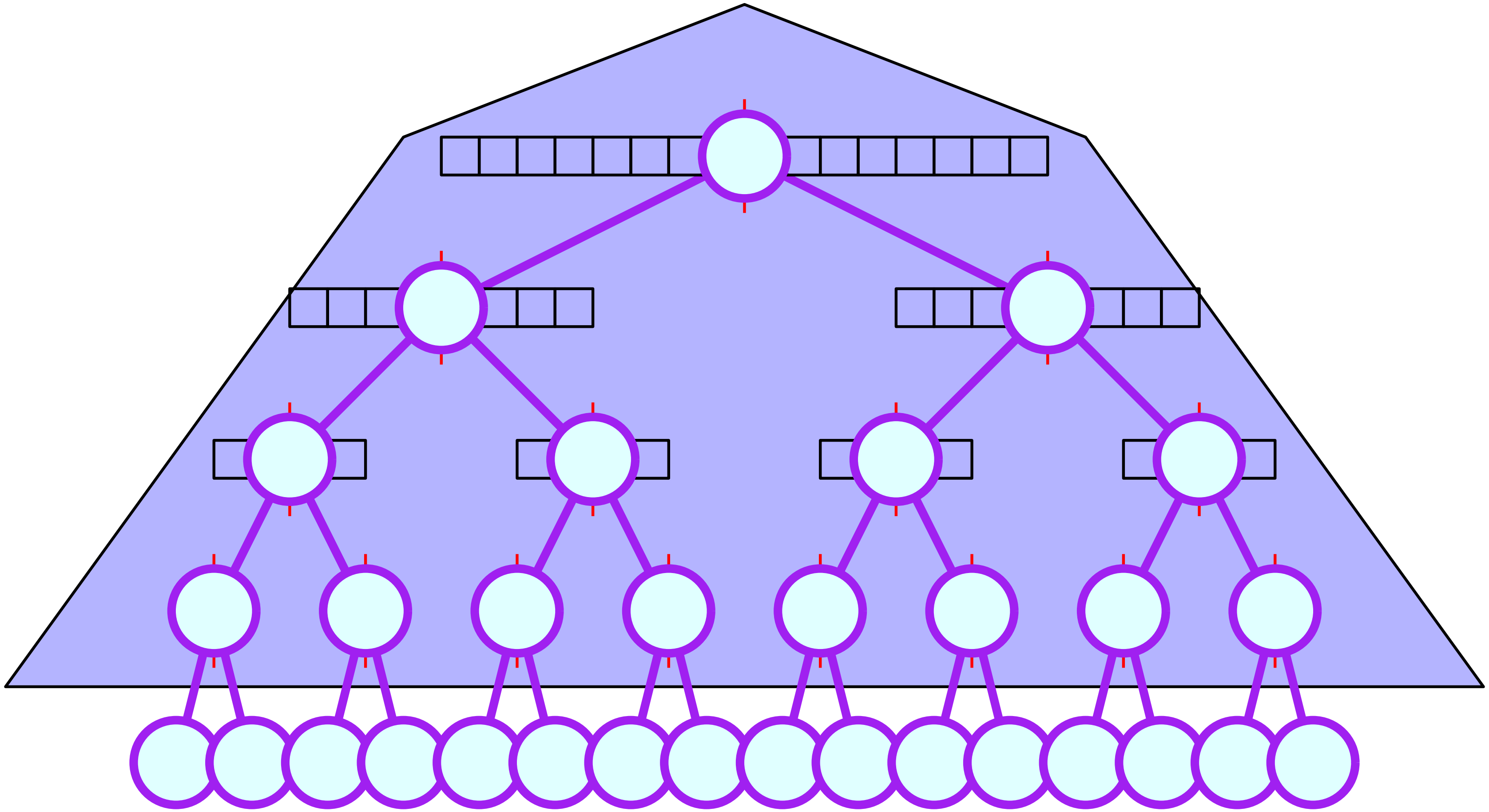
$$16 \cdot \tfrac{n}{16} = n$$



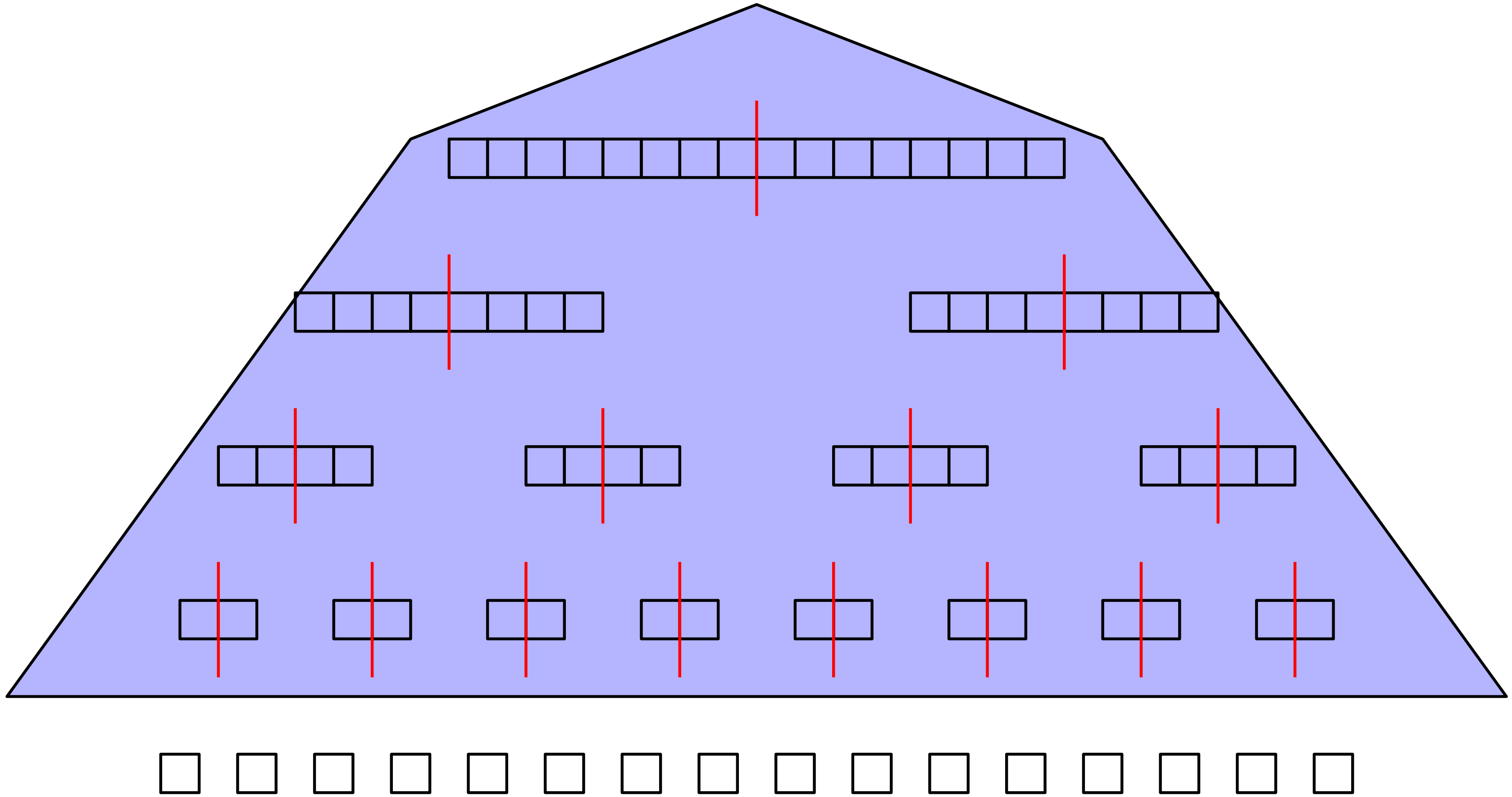
Work-efficient Prefix Sums



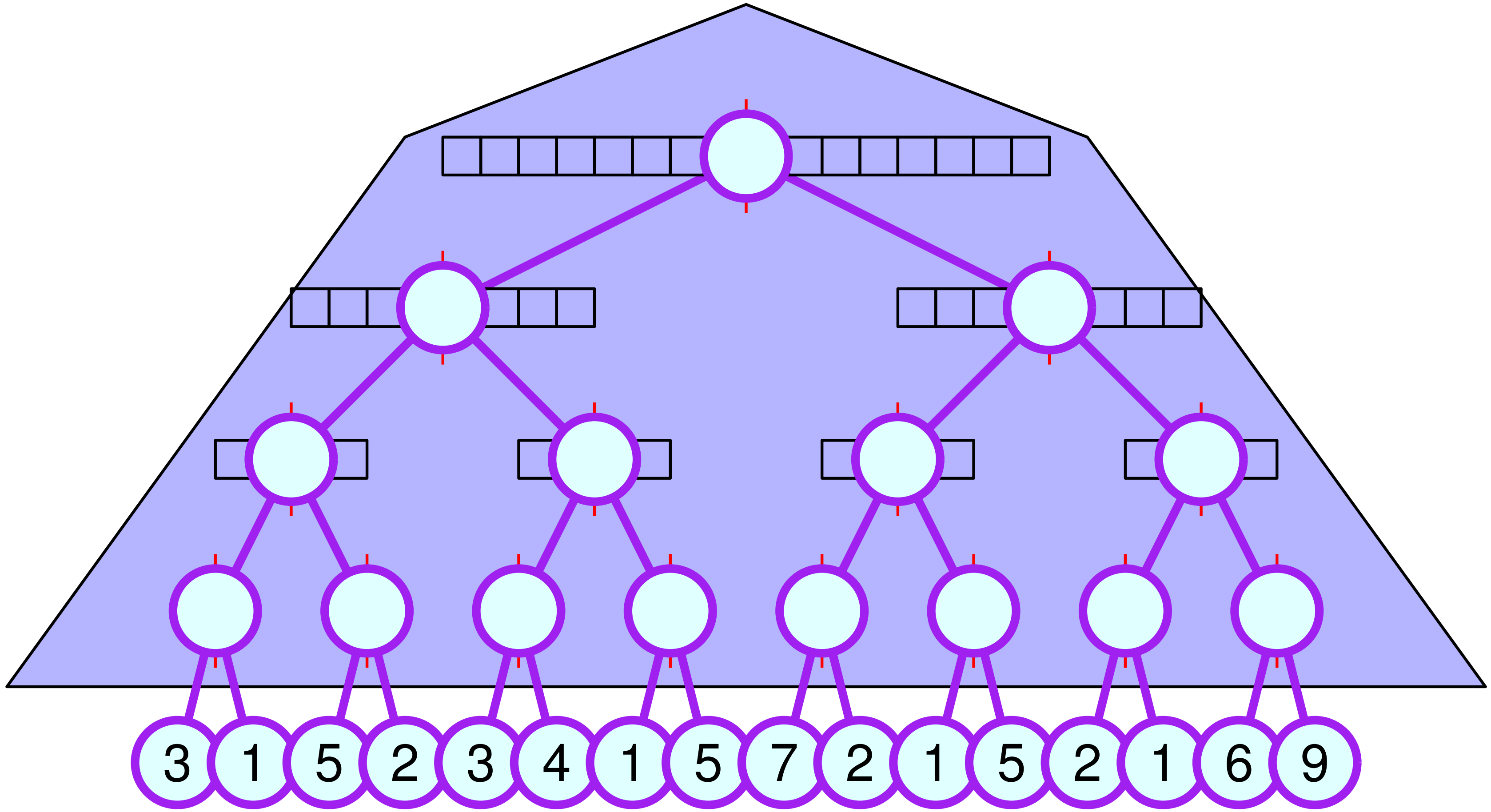
Work-efficient Prefix Sums



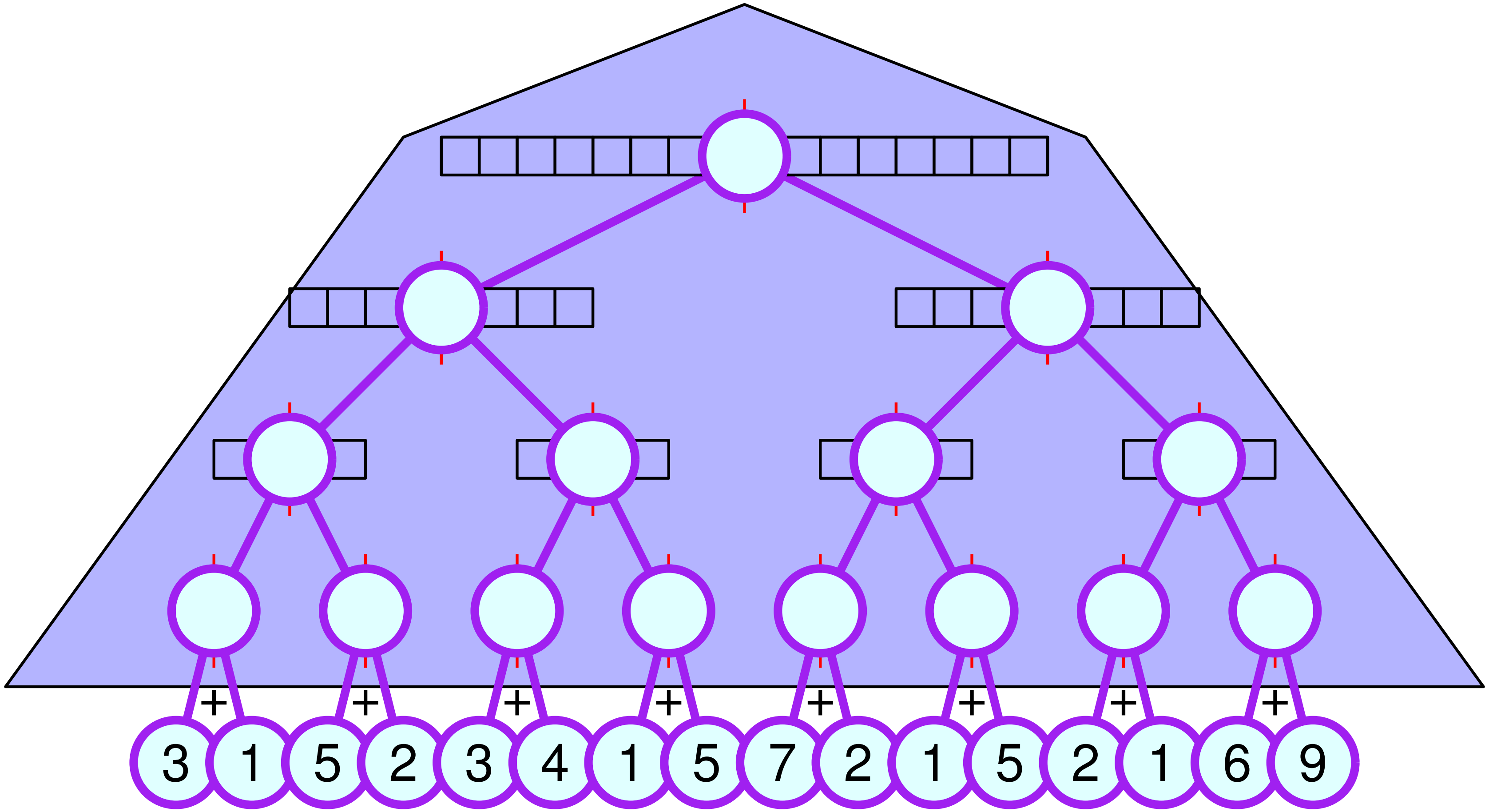
Work-efficient Prefix Sums



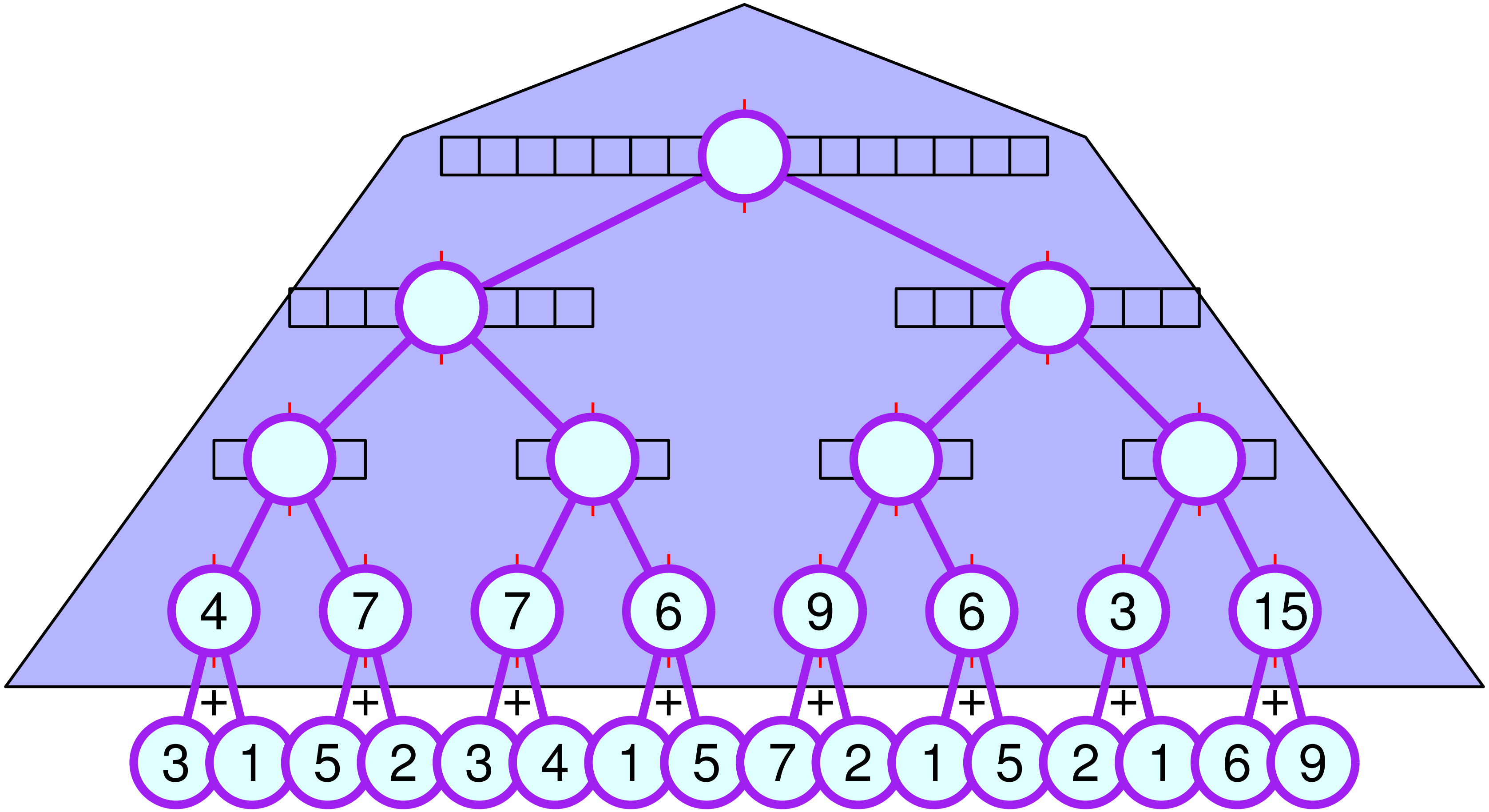
Work-efficient Prefix Sums



Work-efficient Prefix Sums



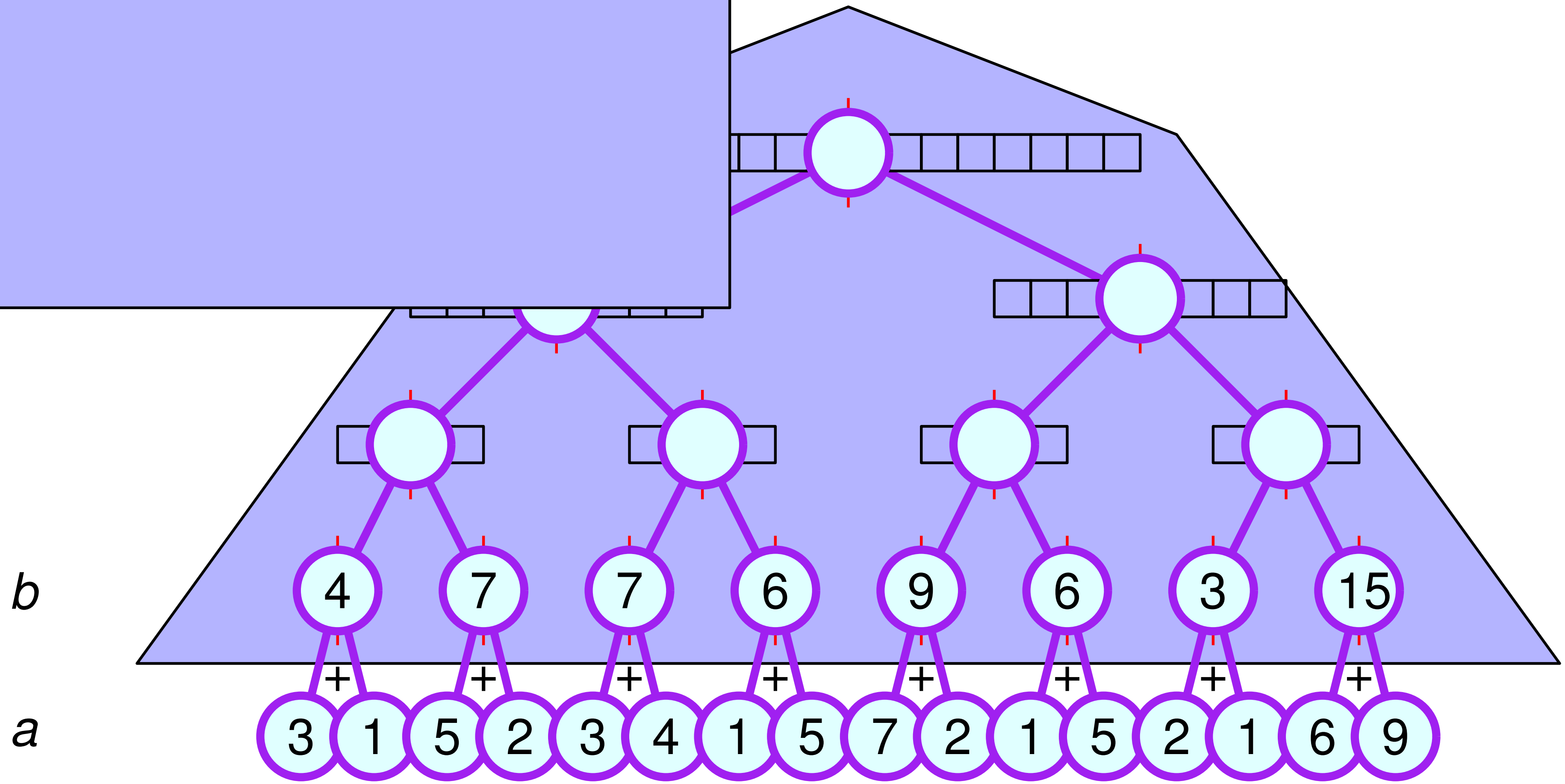
Work-efficient Prefix Sums



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**
$b[i] = a[2i - 1] + a[2i]$



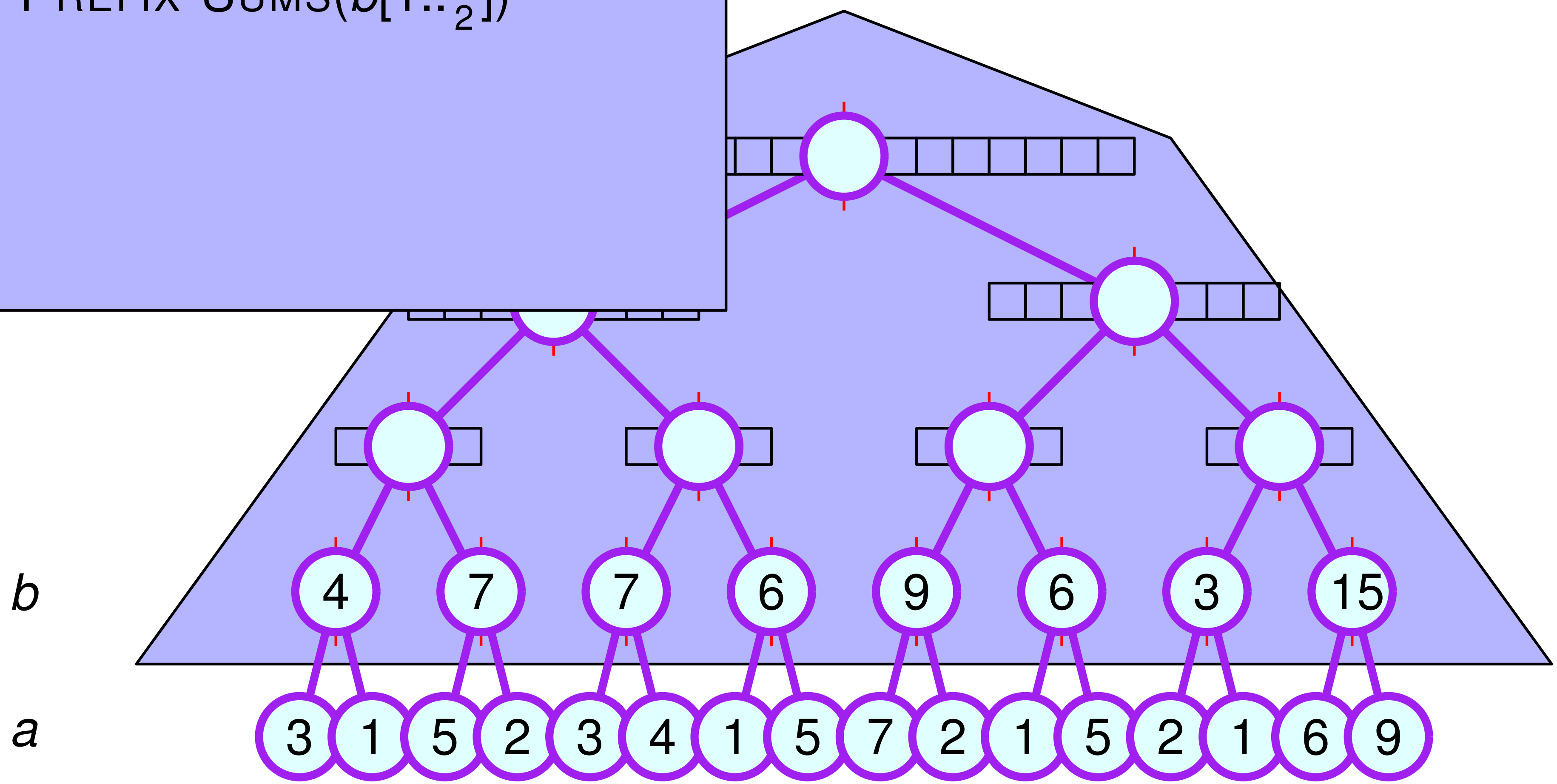
Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

  for $i = 1$ to $\frac{n}{2}$ **in parallel do**

    $b[i] = a[2i-1] + a[2i]$

  PREFIX-SUMS($b[1..\frac{n}{2}]$)



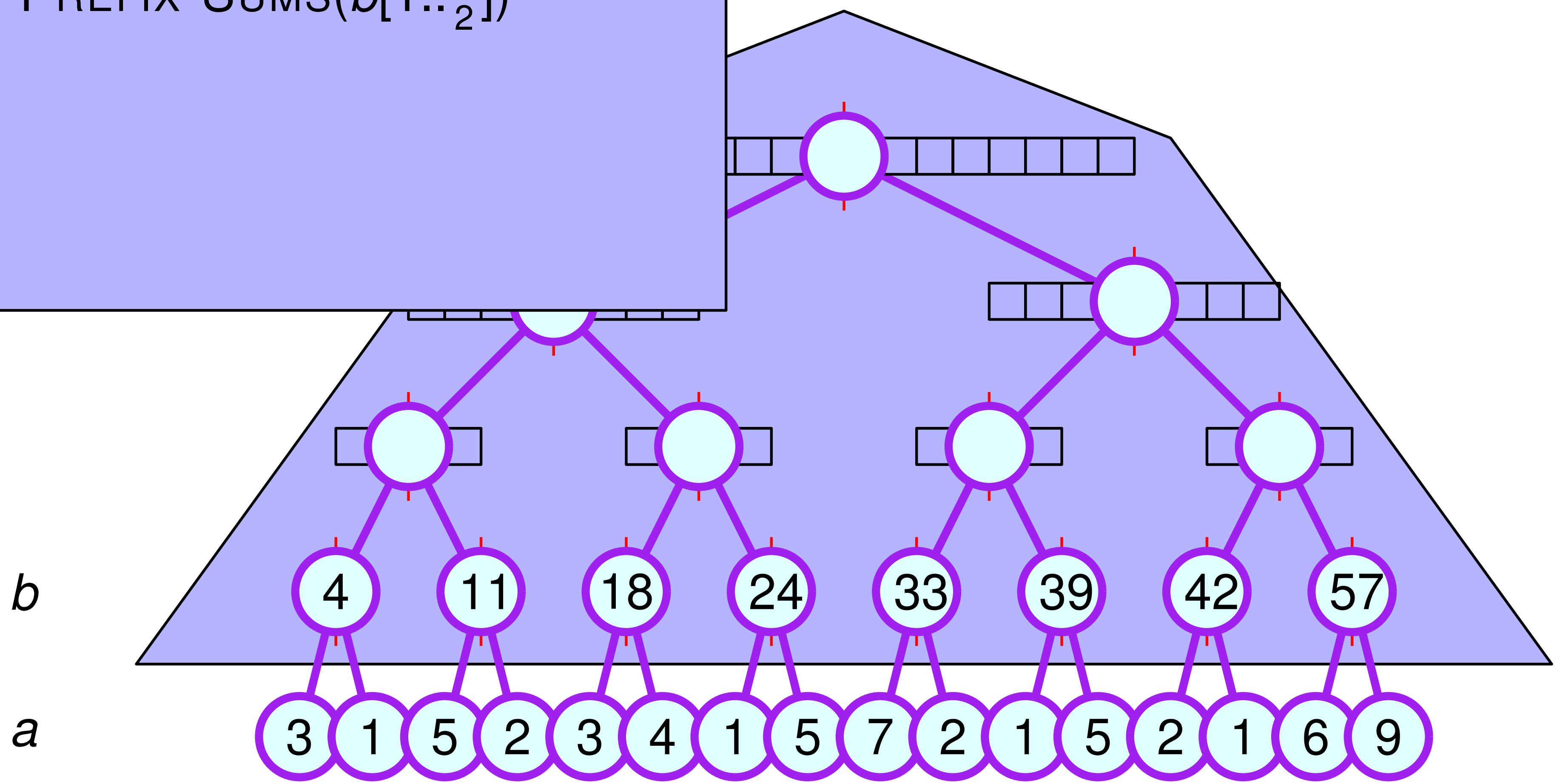
Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

  for $i = 1$ to $\frac{n}{2}$ **in parallel do**

    $b[i] = a[2i-1] + a[2i]$

  PREFIX-SUMS($b[1..\frac{n}{2}]$)

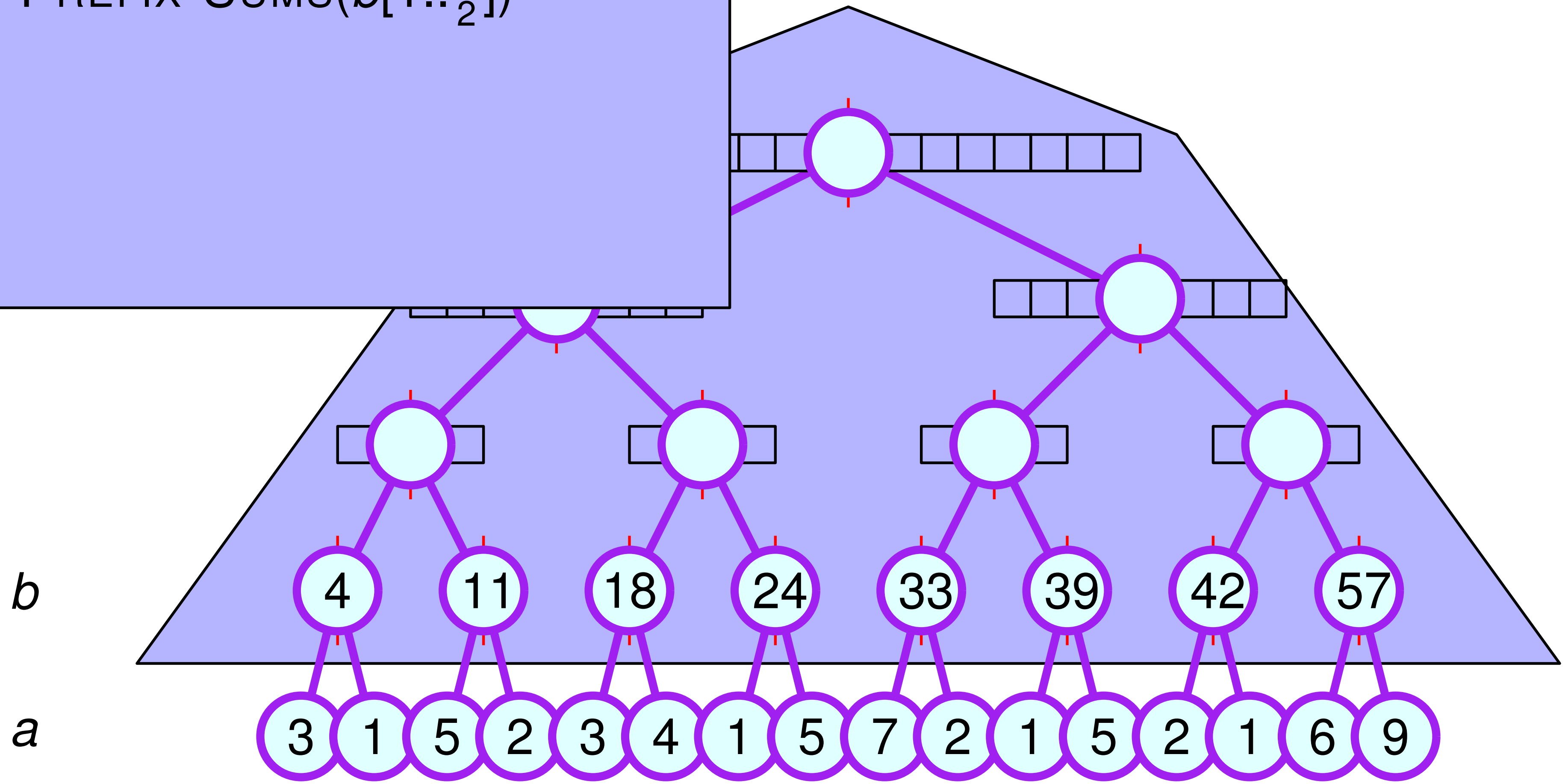


Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**
$b[i] = a[2i-1] + a[2i]$
PREFIX-SUMS($b[1..\frac{n}{2}]$)

Claim. $b[i] = \sum_{k=1}^{2i} a[k]$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

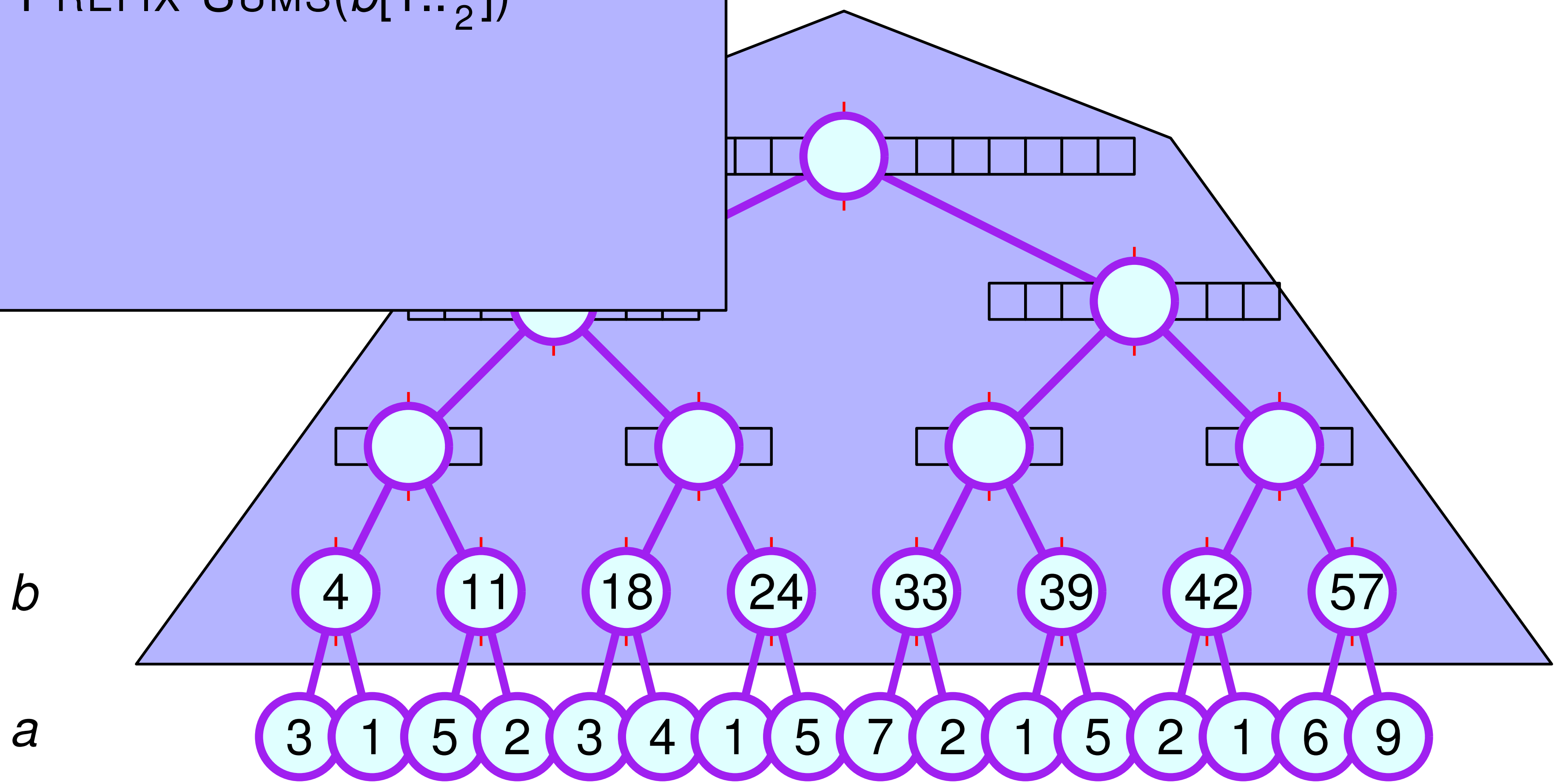
for $i = 1$ to $\frac{n}{2}$ **in parallel do**

$b[i] = a[2i-1] + a[2i]$

PREFIX-SUMS($b[1..\frac{n}{2}]$)

Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$



Work-efficient Prefix Sums

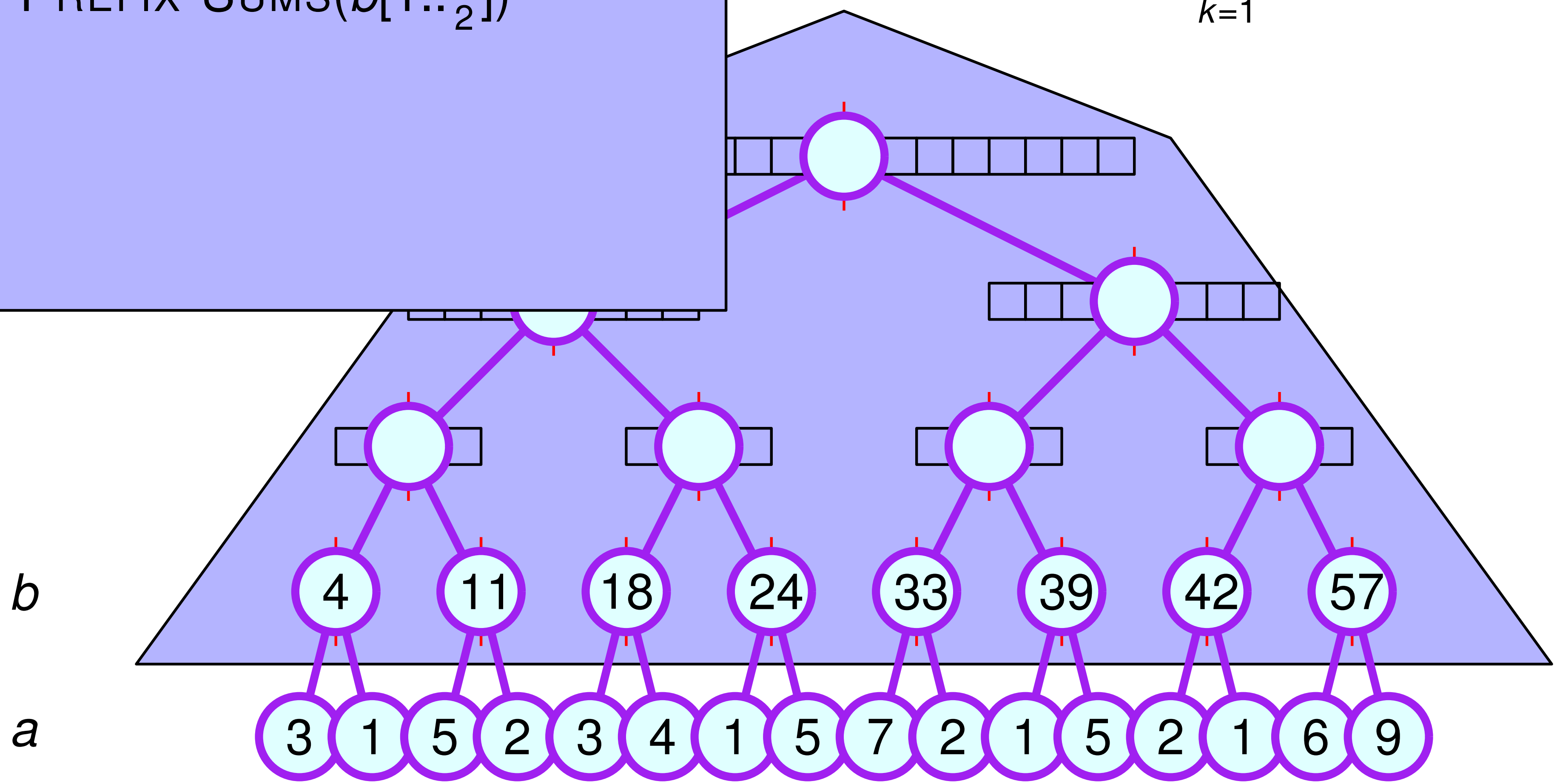
procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $b[i] = a[2i-1] + a[2i]$
 PREFIX-SUMS($b[1..\frac{n}{2}]$)

Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

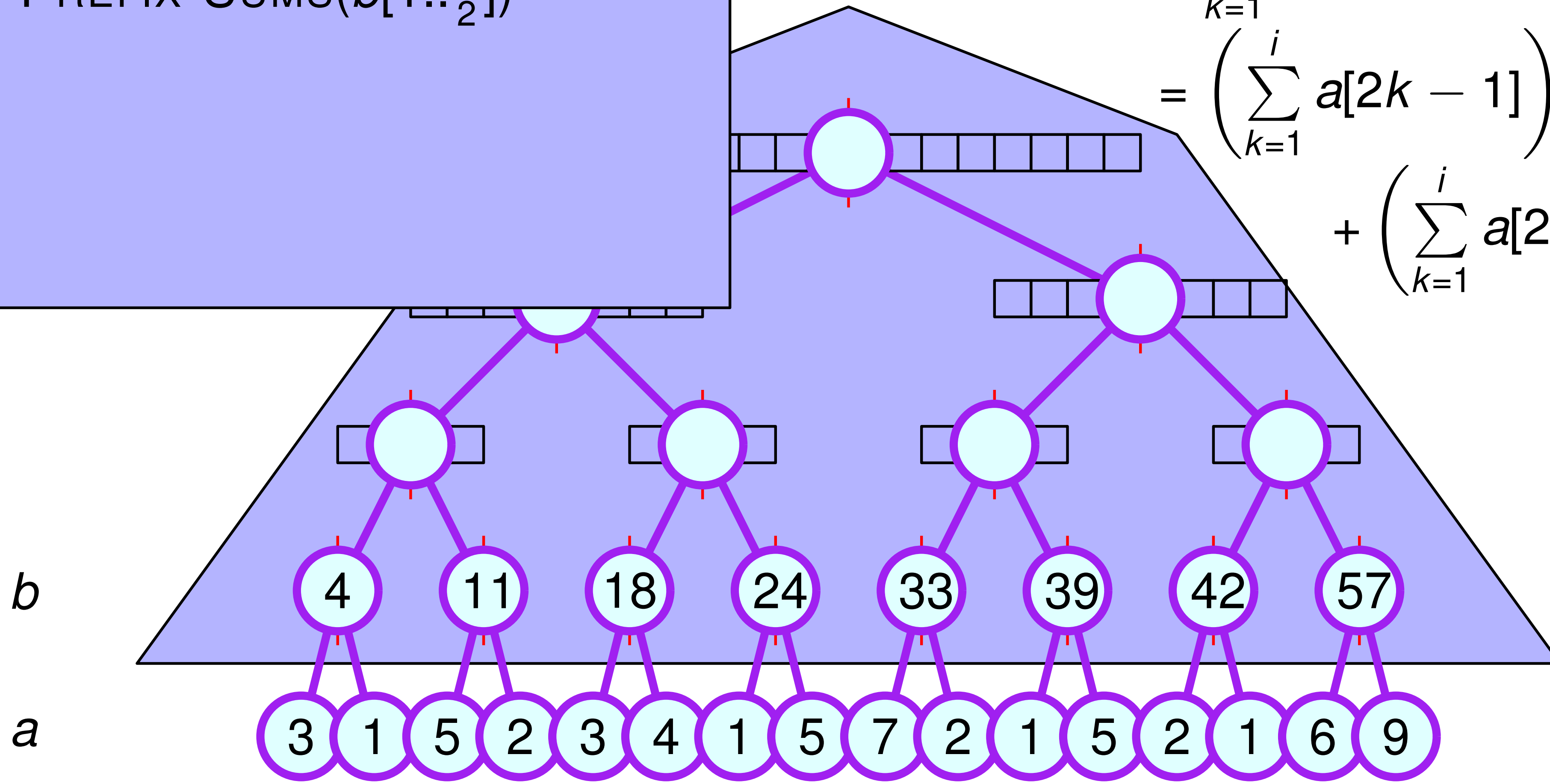
for $i = 1$ to $\frac{n}{2}$ **in parallel do**
$b[i] = a[2i-1] + a[2i]$
PREFIX-SUMS($b[1..\frac{n}{2}]$)

Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $b[i] = a[2i-1] + a[2i]$
 PREFIX-SUMS($b[1..\frac{n}{2}]$)

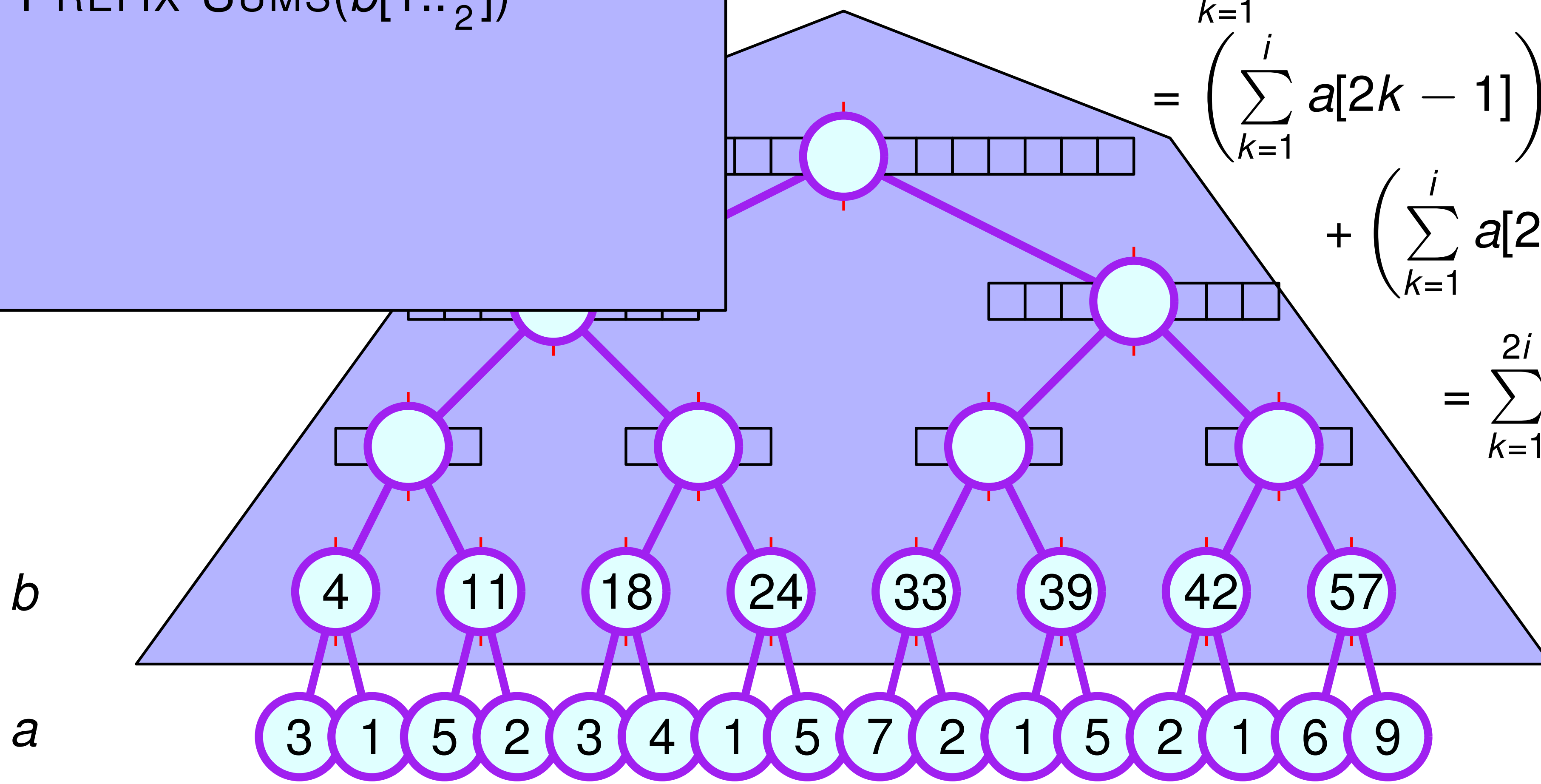
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $b[i] = a[2i-1] + a[2i]$
PREFIX-SUMS($b[1..\frac{n}{2}]$)

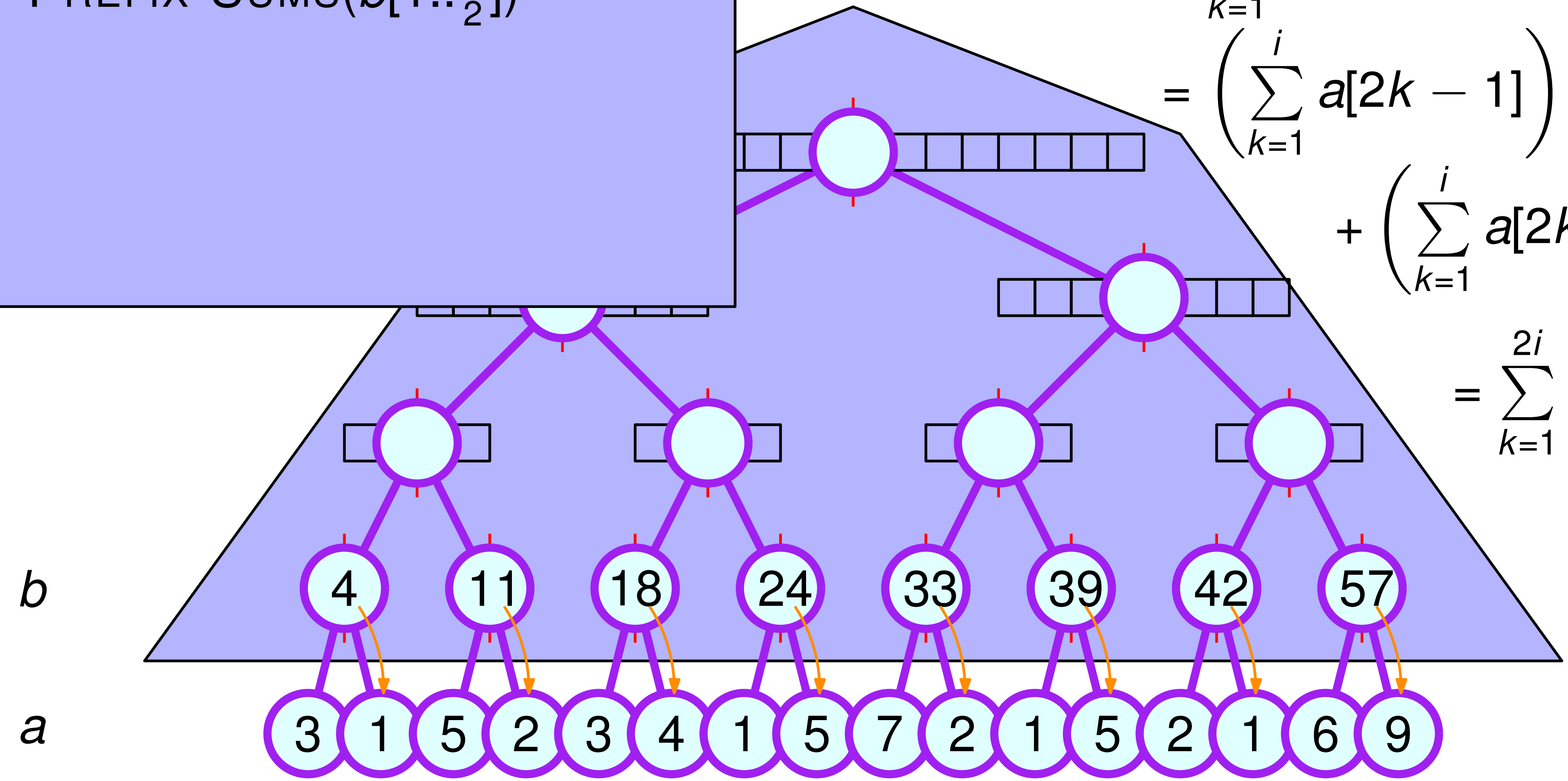
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**

$b[i] = a[2i-1] + a[2i]$

PREFIX-SUMS($b[1..\frac{n}{2}]$)

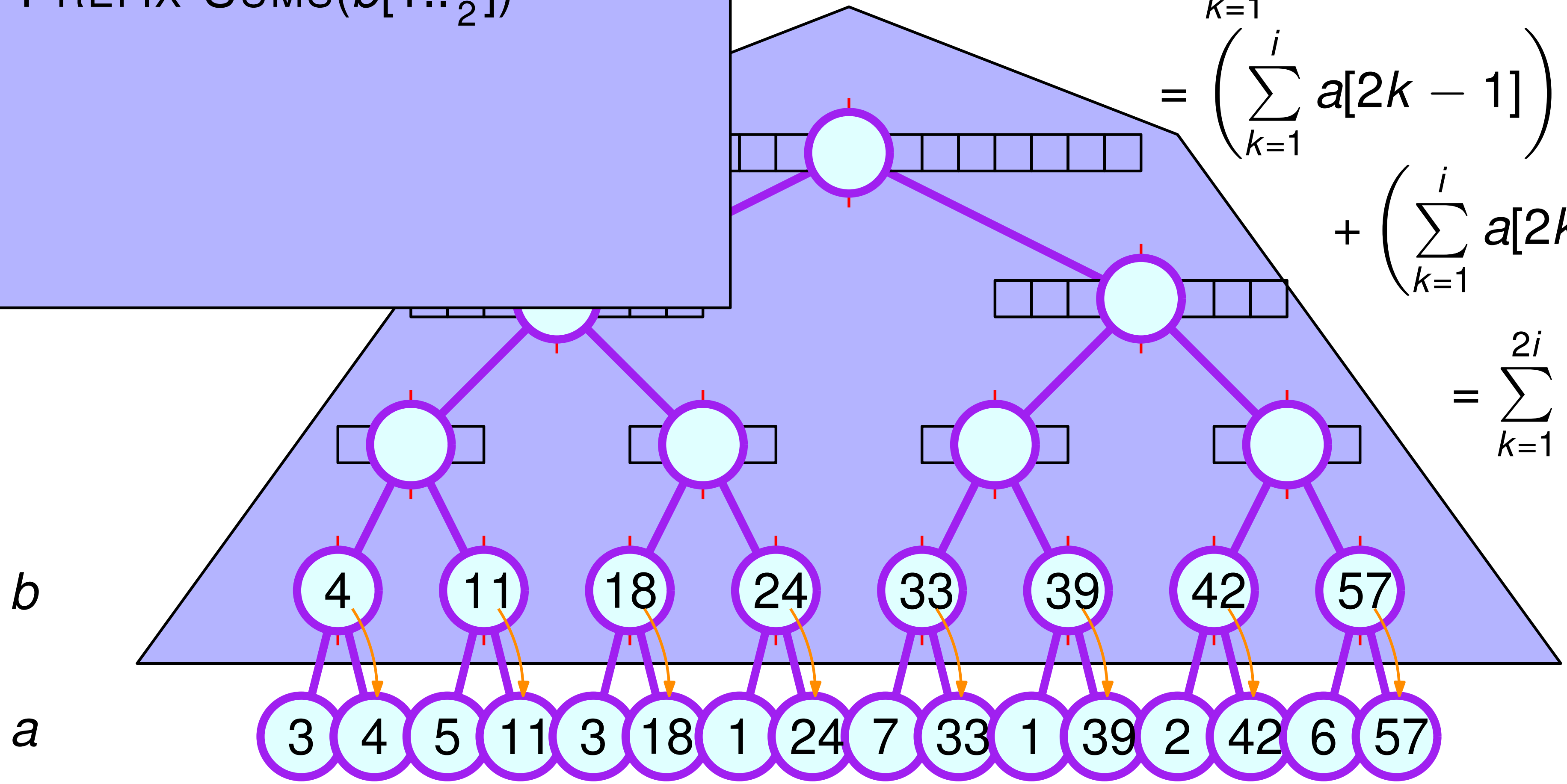
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $b[i] = a[2i-1] + a[2i]$
PREFIX-SUMS($b[1..\frac{n}{2}]$)
for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $a[2i] = b[i]$

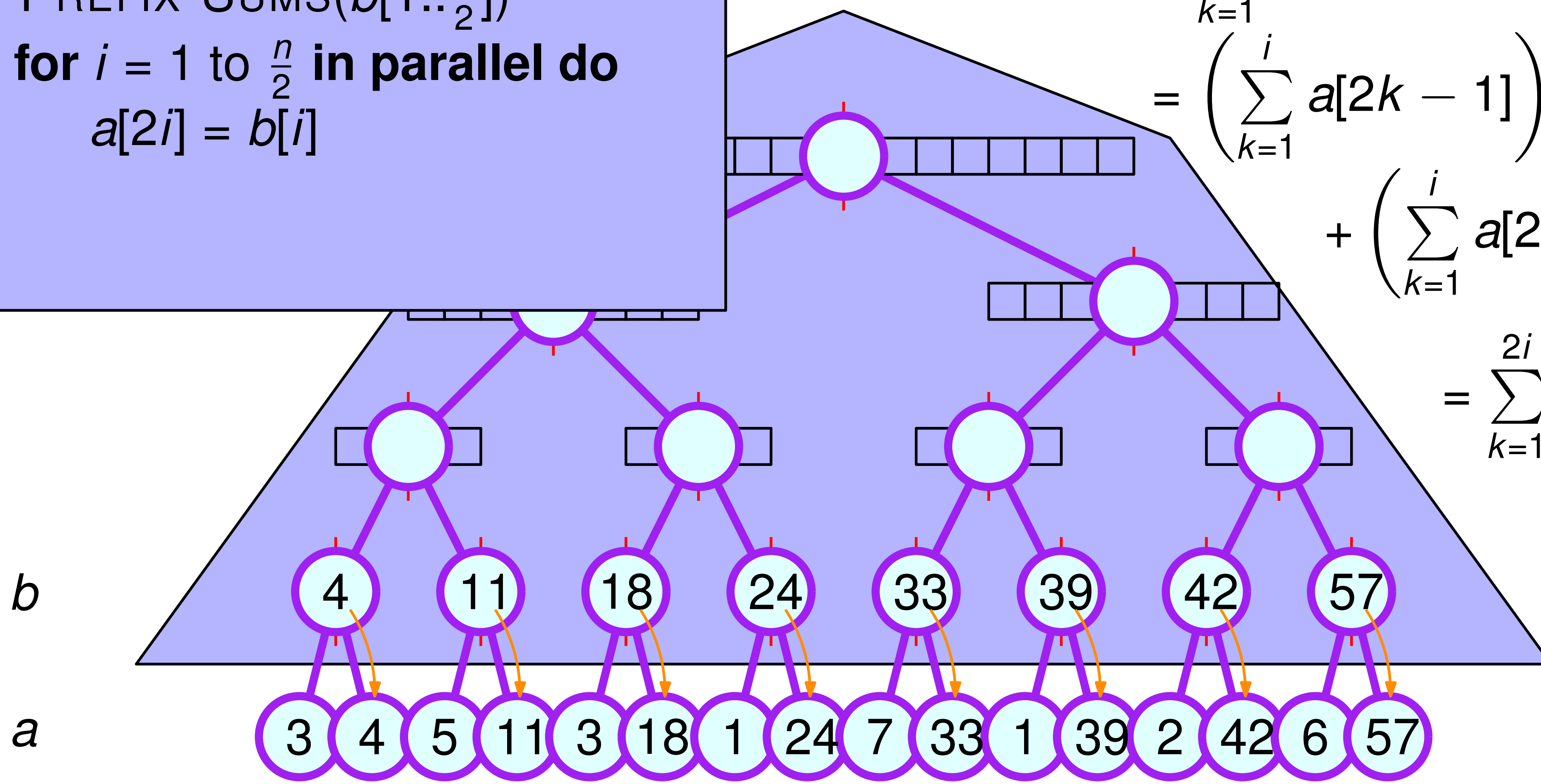
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i-1] + a[2i]$
  PREFIX-SUMS($b[1..\frac{n}{2}]$)
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$

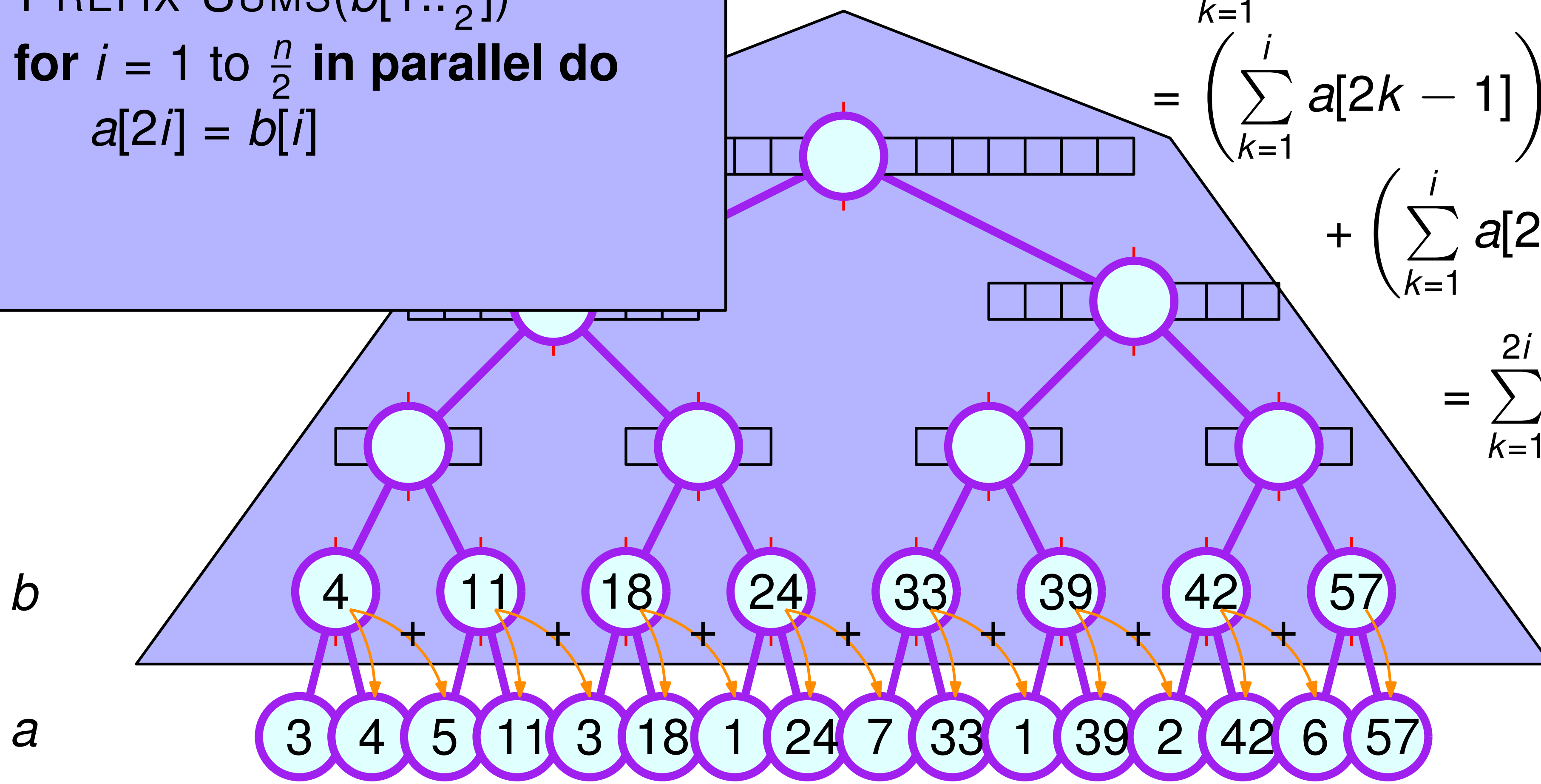
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $b[i] = a[2i-1] + a[2i]$
PREFIX-SUMS($b[1..\frac{n}{2}]$)
for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $a[2i] = b[i]$

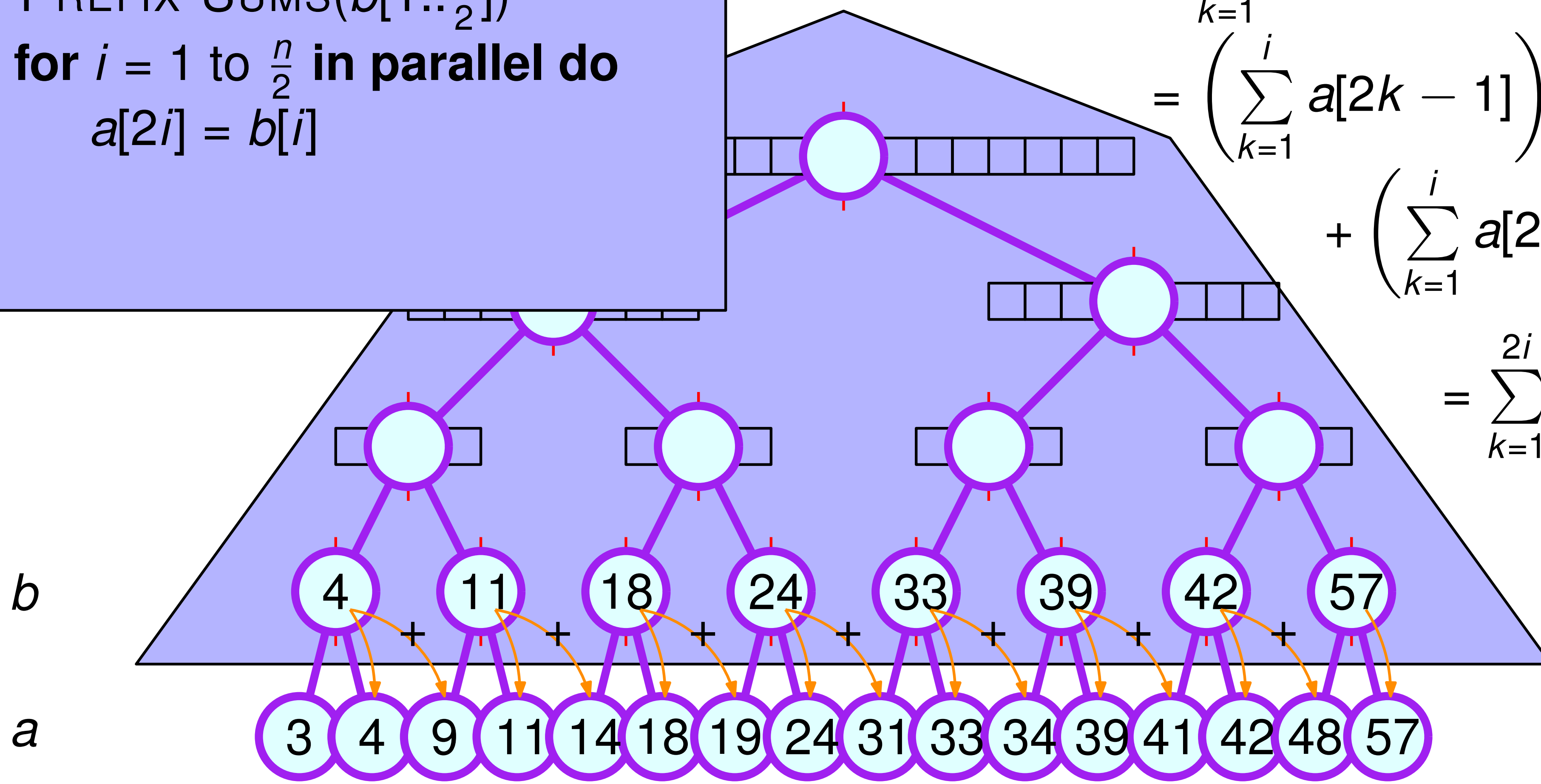
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $b[i] = a[2i-1] + a[2i]$
PREFIX-SUMS($b[1..\frac{n}{2}]$)
for $i = 1$ to $\frac{n}{2}$ **in parallel do**
  $a[2i] = b[i]$
  $a[2i+1] = a[2i+1] + b[i]$

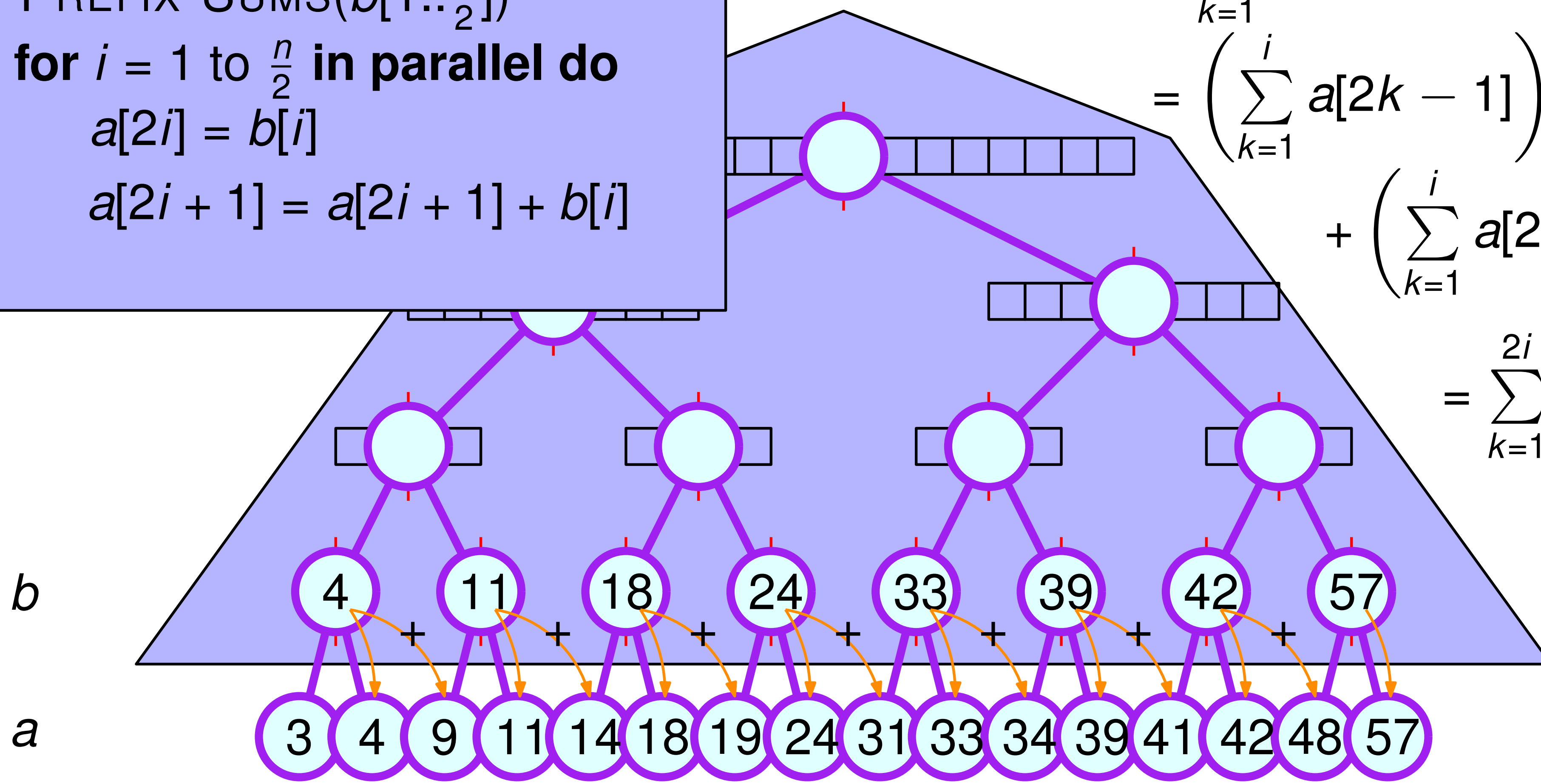
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)

  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i - 1] + a[2i]$
  PREFIX-SUMS($b[1..\frac{n}{2}]$)
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i + 1] = a[2i + 1] + b[i]$

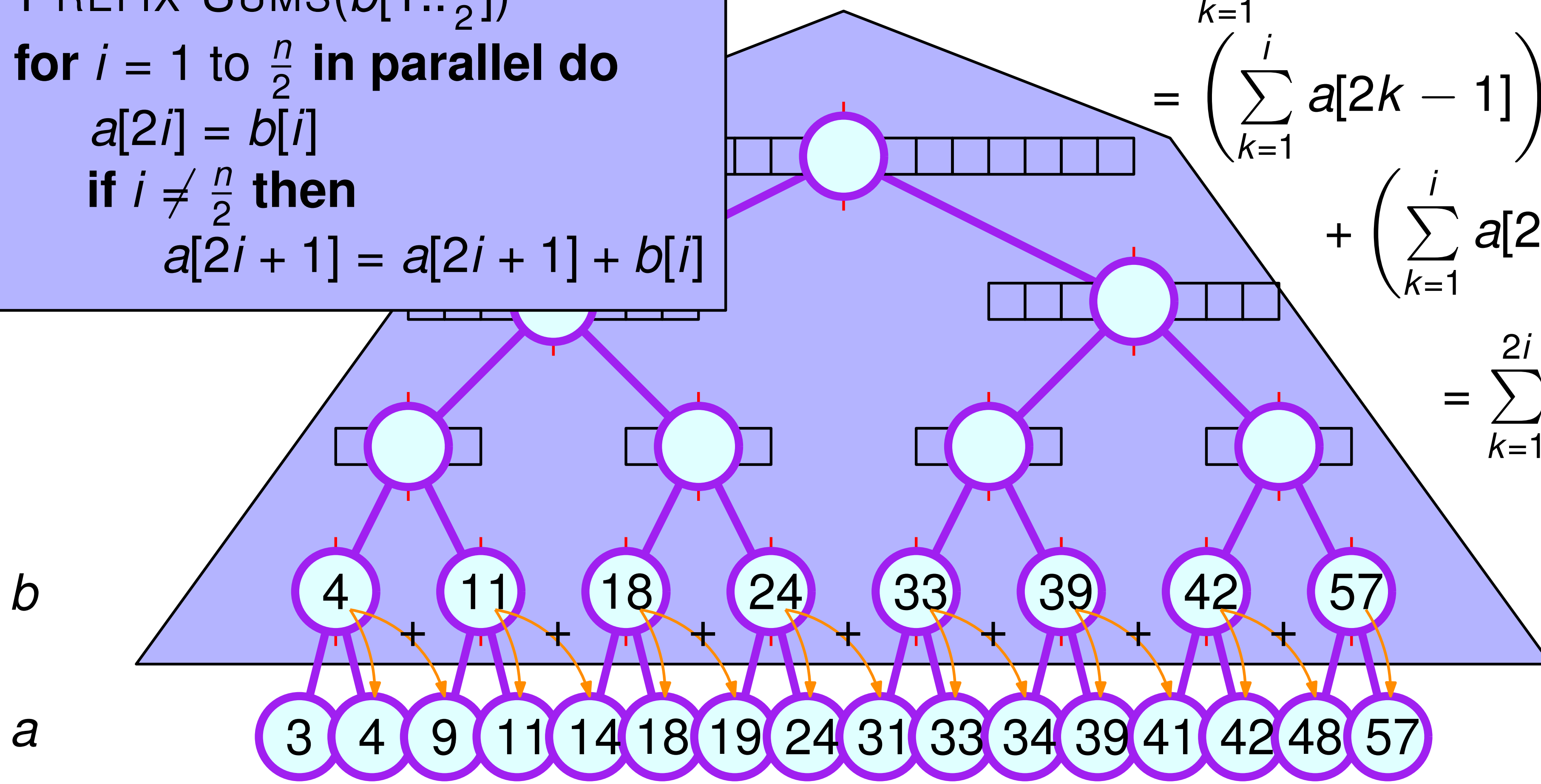
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k - 1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k - 1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



Work-efficient Prefix Sums

procedure $\textsc{Prefix-Sums}(a[1..n])$
  if $n \leq 1$ **then return**
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i-1] + a[2i]$
  $\textsc{Prefix-Sums}(b[1..\frac{n}{2}])$
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i+1] = a[2i+1] + b[i]$

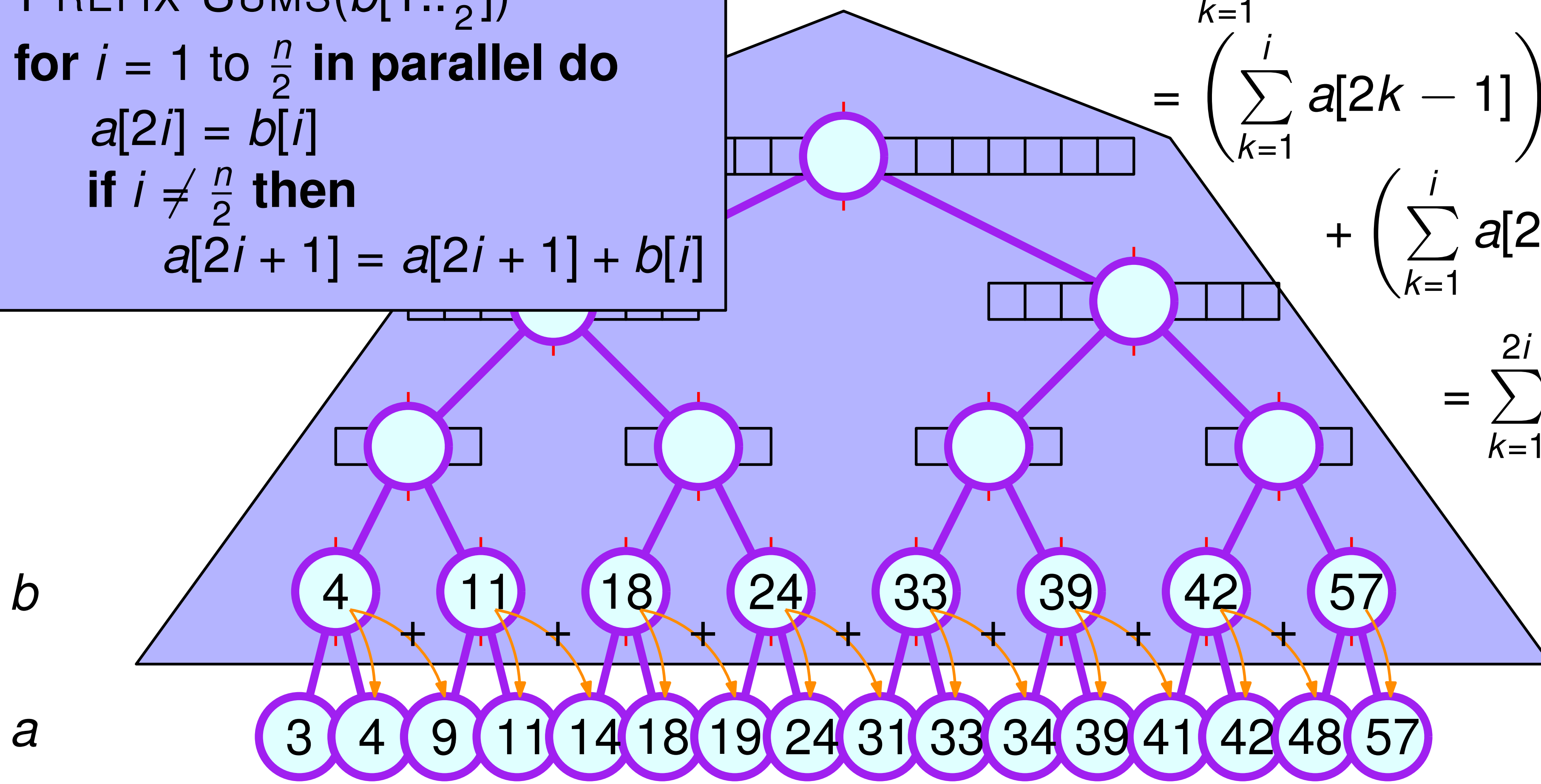
Claim. $b[i] = \sum_{k=1}^{2i} a[k]$

Proof. By I.H. $b[i] = \sum_{k=1}^{i} b[k]$

$$= \sum_{k=1}^{i} (a[2k-1] + a[2k])$$

$$= \left(\sum_{k=1}^{i} a[2k-1]\right) + \left(\sum_{k=1}^{i} a[2k]\right)$$

$$= \sum_{k=1}^{2i} a[k]$$



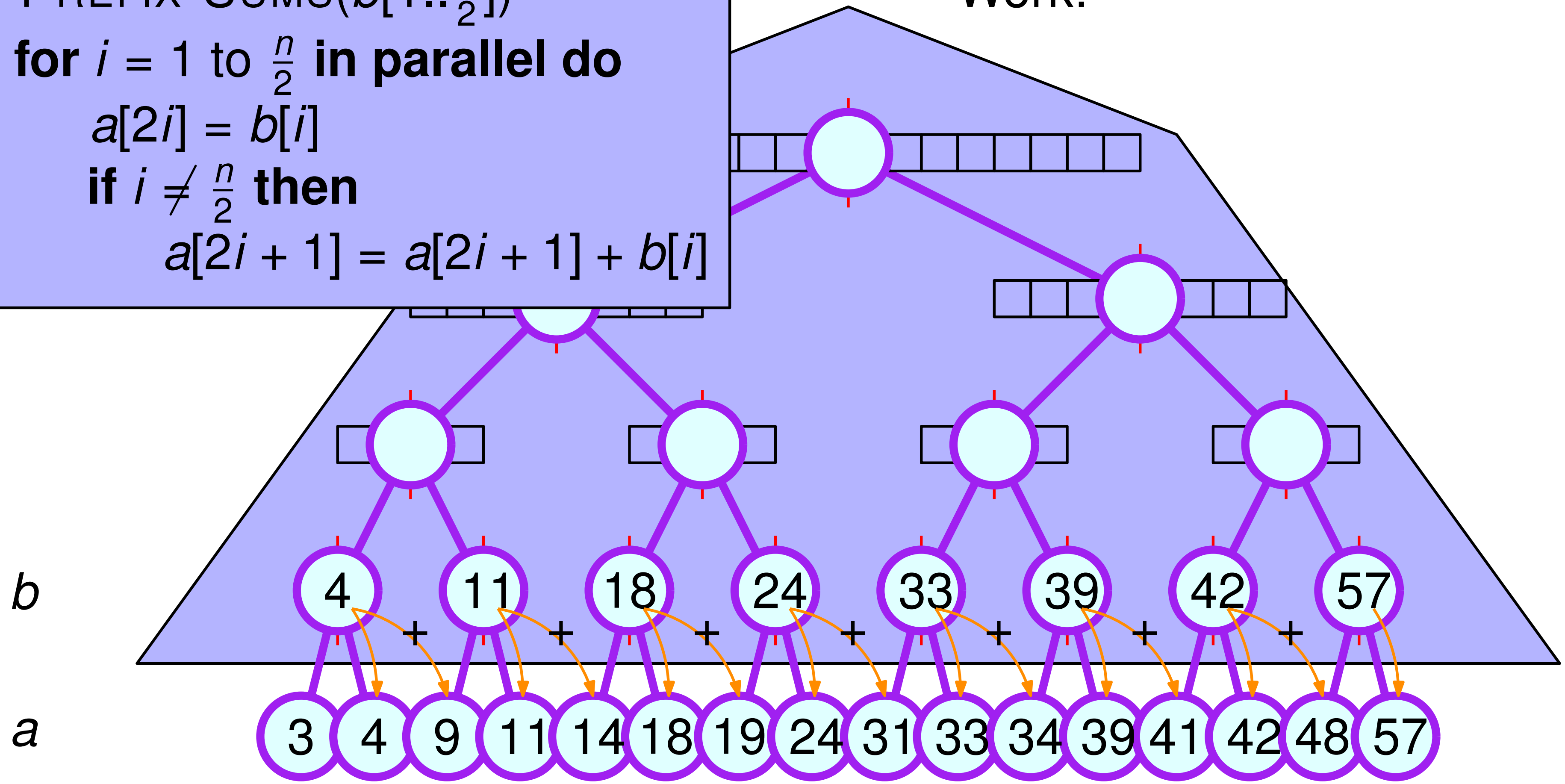
Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)
  if $n \leq 1$ **then return**
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i-1] + a[2i]$
  PREFIX-SUMS($b[1..\frac{n}{2}]$)
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i+1] = a[2i+1] + b[i]$

Analysis

Time:

Work:



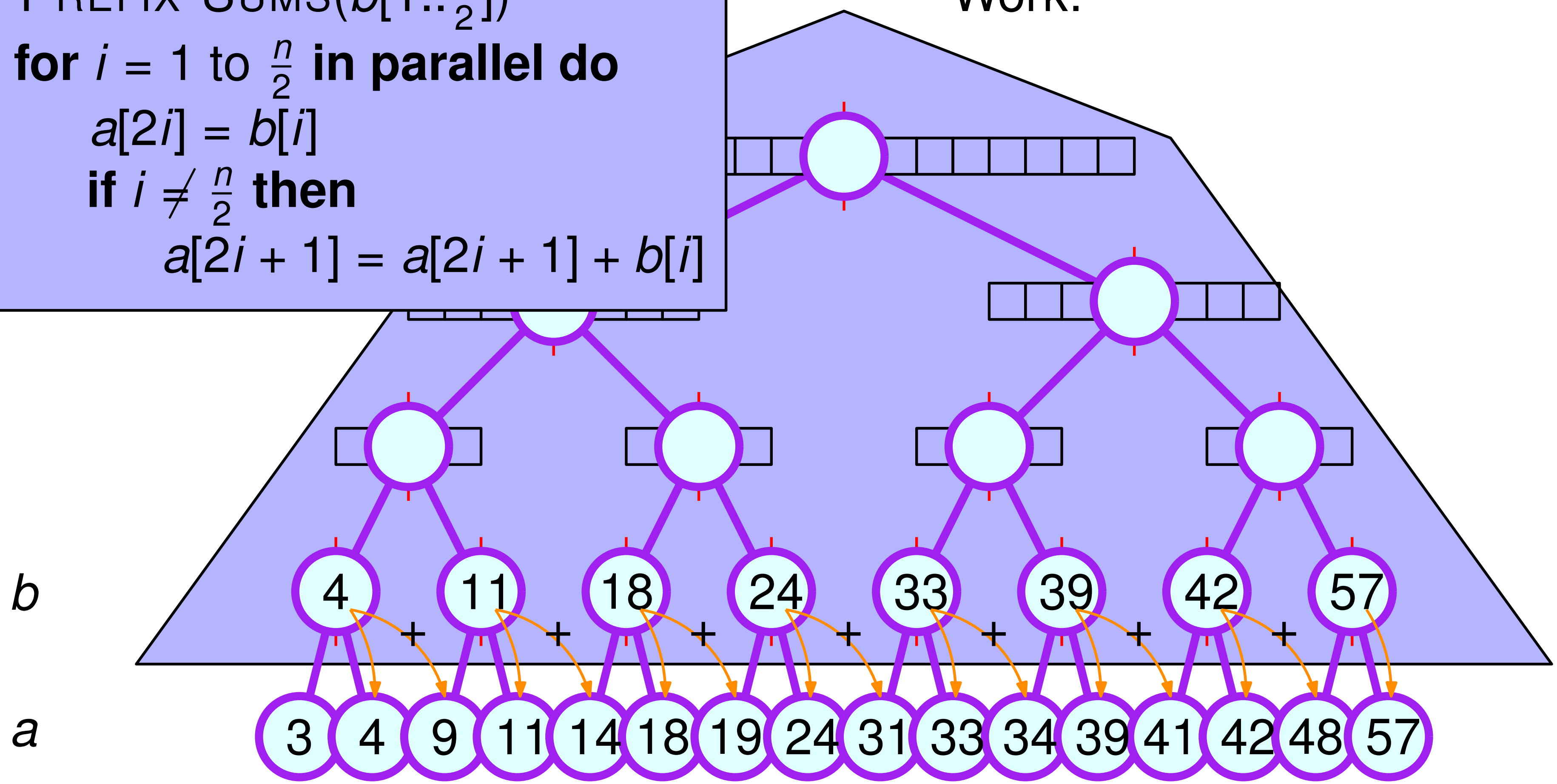
Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)
  if $n \leq 1$ **then return**
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i-1] + a[2i]$
  PREFIX-SUMS($b[1..\frac{n}{2}]$)
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i+1] = a[2i+1] + b[i]$

Analysis

Time: $T(n) = T(n/2) + O(1)$

Work:



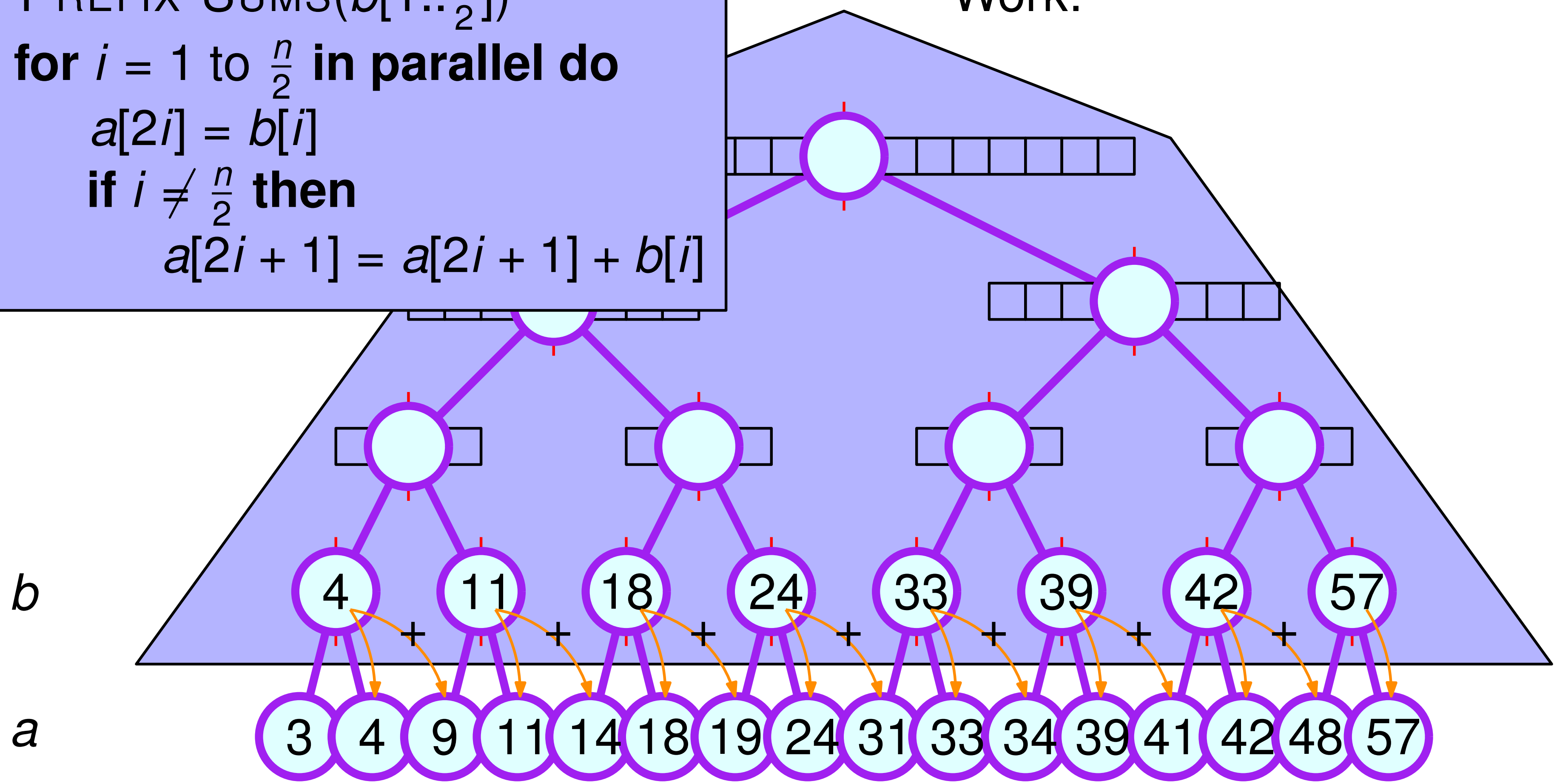
Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)
  if $n \leq 1$ **then return**
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i-1] + a[2i]$
  PREFIX-SUMS($b[1..\frac{n}{2}]$)
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i+1] = a[2i+1] + b[i]$

Analysis

Time:
$$T(n) = T(n/2) + O(1) = O(\log n)$$

Work:



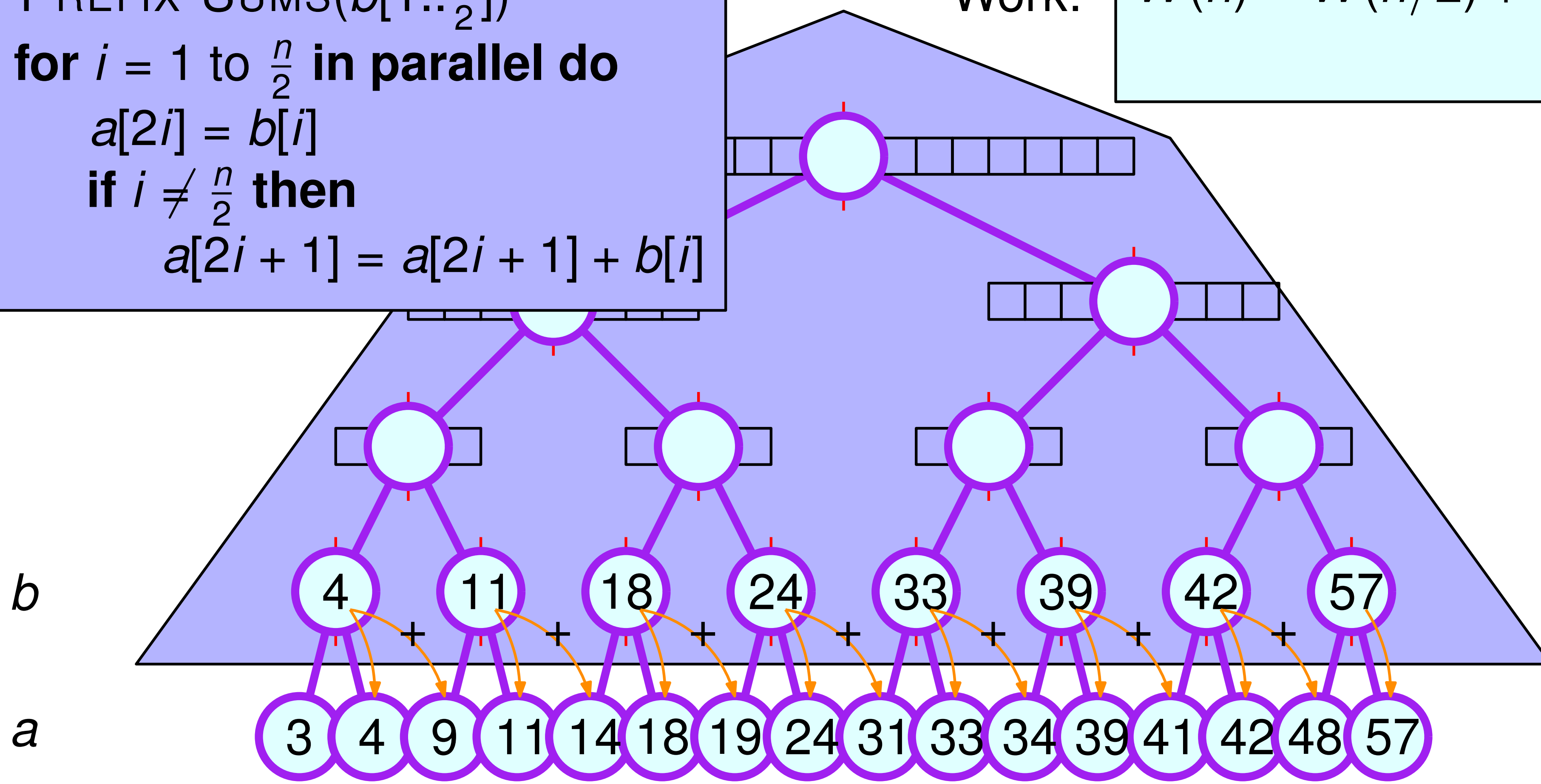
Work-efficient Prefix Sums

procedure $\textsc{Prefix-Sums}(a[1..n])$
  if $n \leq 1$ **then return**
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i-1] + a[2i]$
  $\textsc{Prefix-Sums}(b[1..\frac{n}{2}])$
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i+1] = a[2i+1] + b[i]$

Analysis

Time: $T(n) = T(n/2) + O(1) = O(\log n)$

Work: $W(n) = W(n/2) + O(n)$



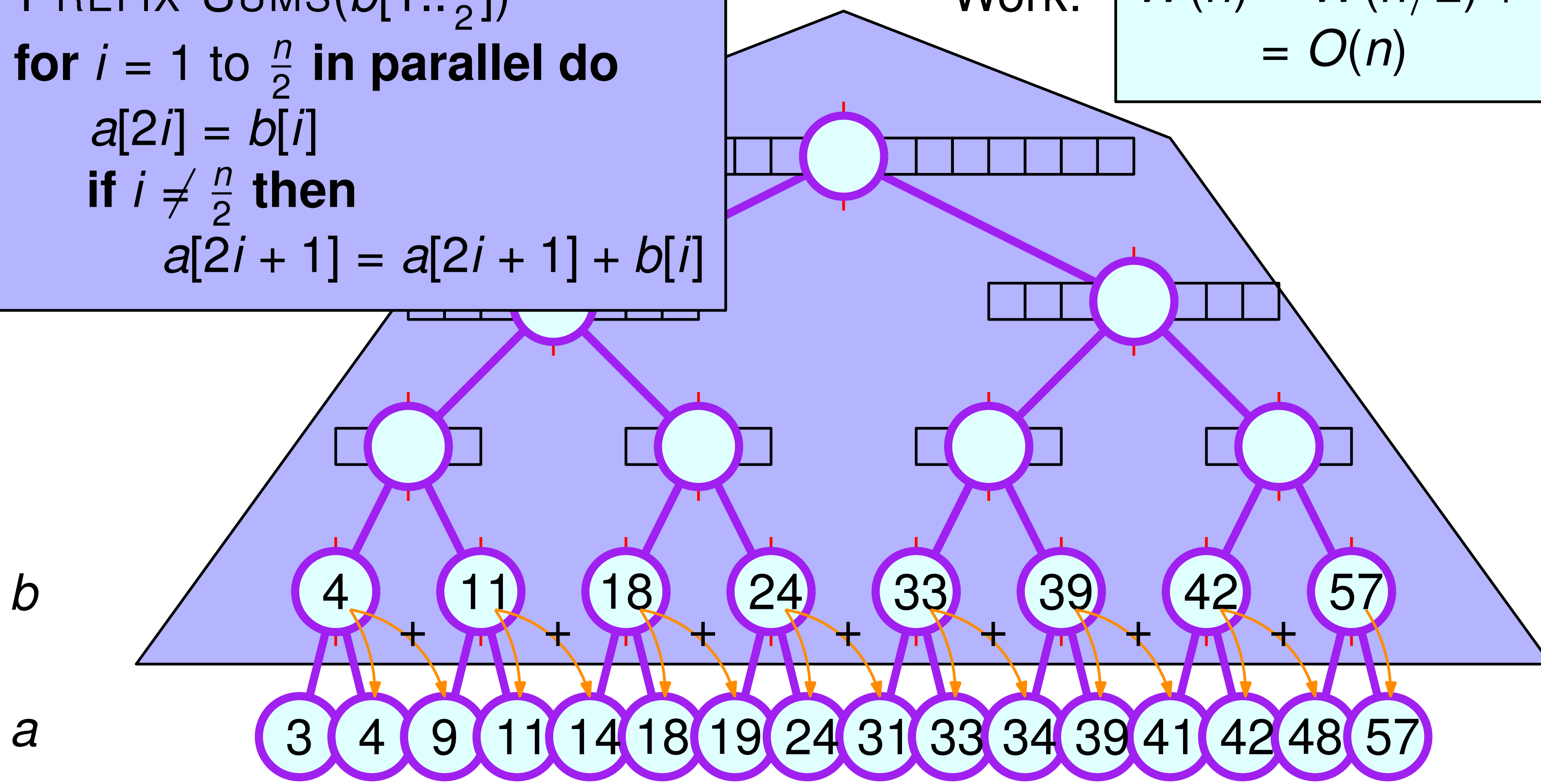
Work-efficient Prefix Sums

procedure PREFIX-SUMS($a[1..n]$)
  if $n \leq 1$ **then return**
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i-1] + a[2i]$
  PREFIX-SUMS($b[1..\frac{n}{2}]$)
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i+1] = a[2i+1] + b[i]$

Analysis

Time: $T(n) = T(n/2) + O(1) = O(\log n)$

Work: $W(n) = W(n/2) + O(n) = O(n)$



Recursion vs. parallel **for** loop

Recursion vs. parallel **for** loop

```
function PREFIX-SUMS(A, i, j)
    if i ≥ j then return                          ▷ Base case
    mid = ⌊(i+j)/2⌋
    spawn
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS(A, mid + 1, j)
    sync

    for k = mid + 1 to j in parallel do
        A[k] = A[k] + A[mid]
```

Recursion vs. parallel **for** loop

function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    for k = 1 to 2 **in parallel do**
        if k = 1 **then**
            PREFIX-SUMS(A, i, mid)
        else
            PREFIX-SUMS($A, mid + 1, j$)
    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

Parallel Sorting

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  MERGESORT(A, i, mid)
  MERGESORT($A, mid + 1, j$)

  MERGE(A, i, mid, j)

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** $\triangleright$ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  MERGESORT(A, i, mid)
  MERGESORT($A, mid + 1, j$)

  MERGE(A, i, mid, j)

$$\begin{aligned} T(n) &= 2T(n/2) + T_{\text{MERGE}} \\ &= 2T(n/2) + O(n) \end{aligned}$$

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  MERGESORT(A, i, mid)
  MERGESORT($A, mid + 1, j$)

  MERGE(A, i, mid, j)

$$
\begin{aligned}
T(n) &= 2T(n/2) + T_{\text{MERGE}} \\
&= 2T(n/2) + O(n) \\
&= O(n \log n)
\end{aligned}
$$

Parallel Sorting

function MergeSort(A, i, j)

  if $i \geq j$ **then return** ▷ Base case

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  in parallel do {

    MergeSort(A, i, mid)

    MergeSort($A, mid + 1, j$)

  }

  Merge(A, i, mid, j)

$$\begin{aligned} T(n) &= 2T(n/2) + T_{\textsc{Merge}} \\ &= 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Parallel Sorting

function MergeSort(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do $\{$
    MergeSort(A, i, mid)
    MergeSort($A, mid + 1, j$)
  $\}$
  Merge(A, i, mid, j)

$$\begin{aligned} T(n) &= \_\,T(n/2) + T_{\text{Merge}} \\ &= \_\,T(n/2) + O(n) \end{aligned}$$

Parallel Sorting

function MERGESORT(A, i, j)

  if $i \geq j$ **then return** ▷ Base case

  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  in parallel do $\{$

    MERGESORT(A, i, mid)

    MERGESORT($A, mid + 1, j$)

  $\}$

  MERGE(A, i, mid, j)

$$\begin{aligned} T(n) &= \_\,T(n/2) + T_{\text{MERGE}} \\ &= \_\,T(n/2) + O(n) \\ &= O(n) \end{aligned}$$

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do $\{$
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  $\}$
  MERGE(A, i, mid, j)

$$\begin{aligned} T(n) &= \_\,T(n/2) + T_{\text{MERGE}} \\ &= \_\,T(n/2) + O(\log n) \end{aligned}$$

With parallel merging

Parallel Sorting

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do {
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  }
  MERGE(A, i, mid, j)

$$
\begin{aligned}
T(n) &= \_\,T(n/2) + T_{\text{MERGE}} \\
&= \_\,T(n/2) + O(\log n) \\
&= O(\log^2 n)
\end{aligned}
$$

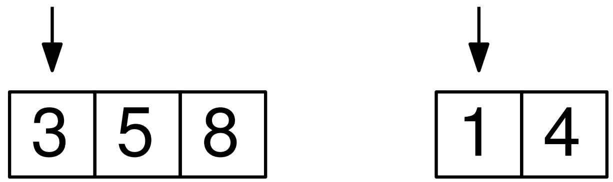
With parallel merging

Parallel Merging

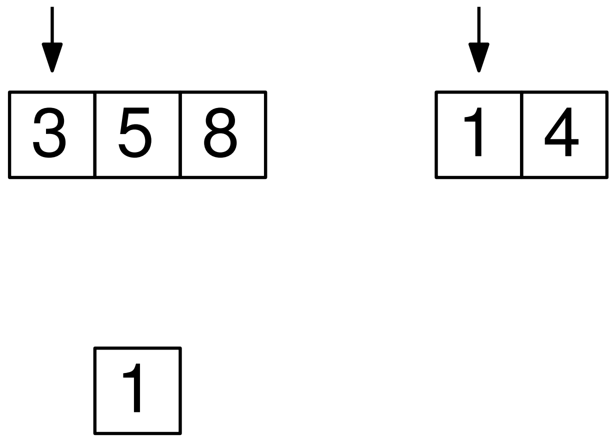
3	5	8

1	4

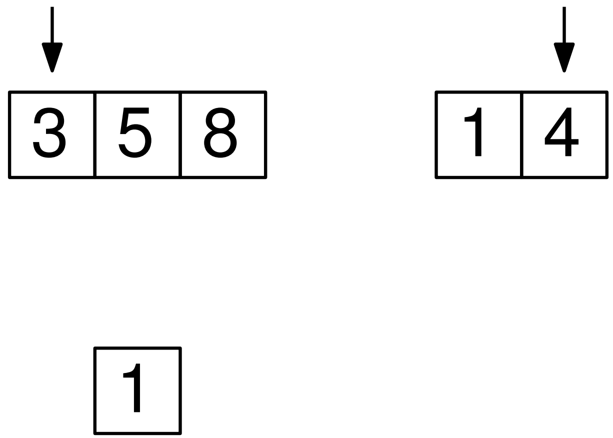
Parallel Merging



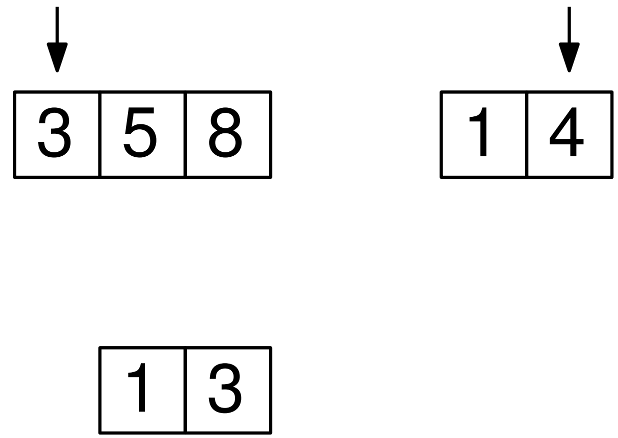
Parallel Merging



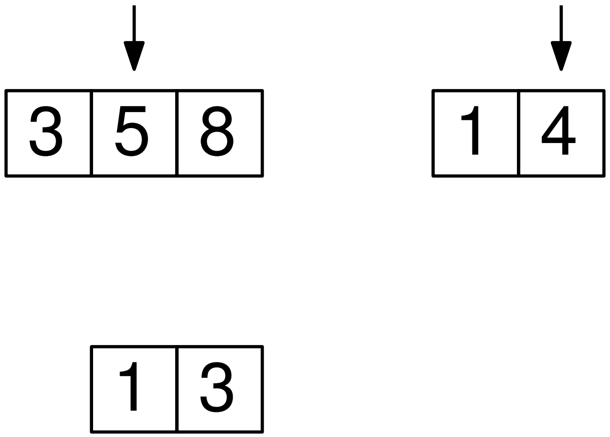
Parallel Merging



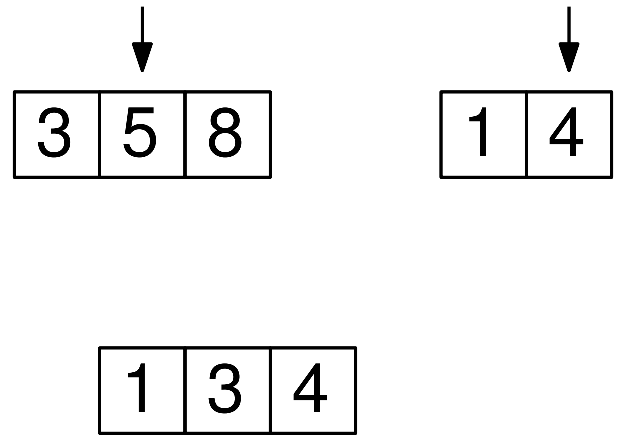
Parallel Merging



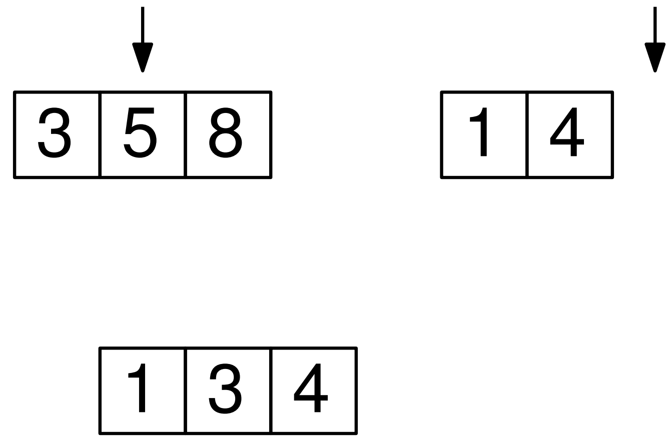
Parallel Merging



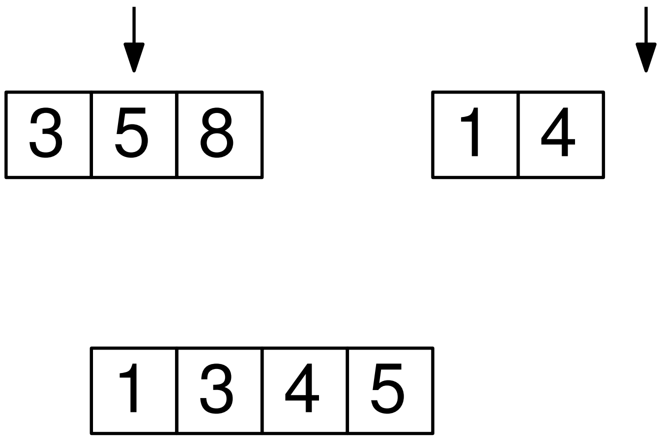
Parallel Merging



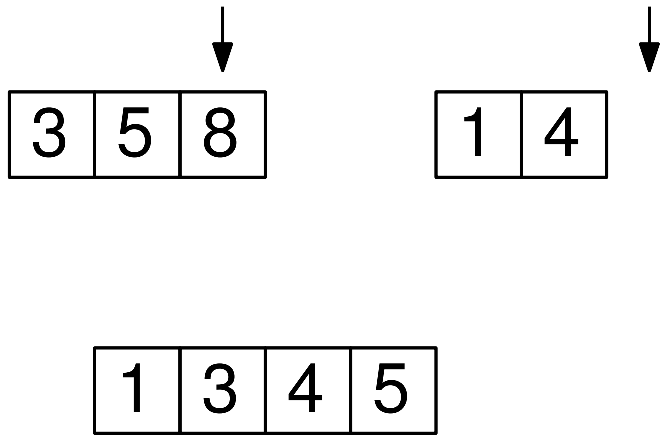
Parallel Merging



Parallel Merging



Parallel Merging



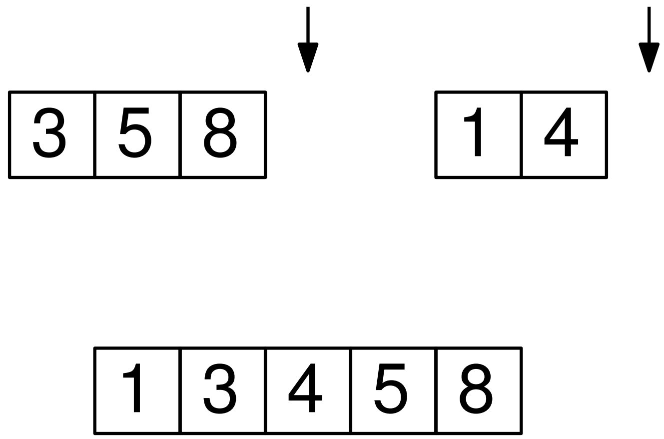
Parallel Merging

3	5	8

1	4

1	3	4	5	8

Parallel Merging



Parallel Merging

3	5	8

1	4

$O(n)$

1	3	4	5	8

Parallel Merging

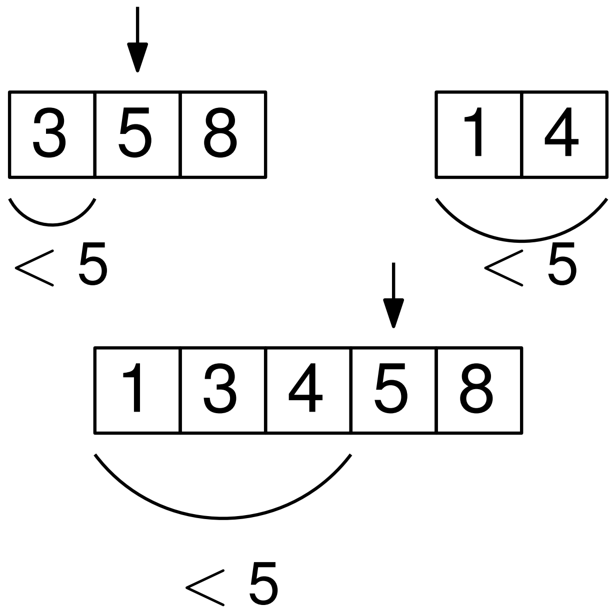
3	5	8

1	4

$O(n)$

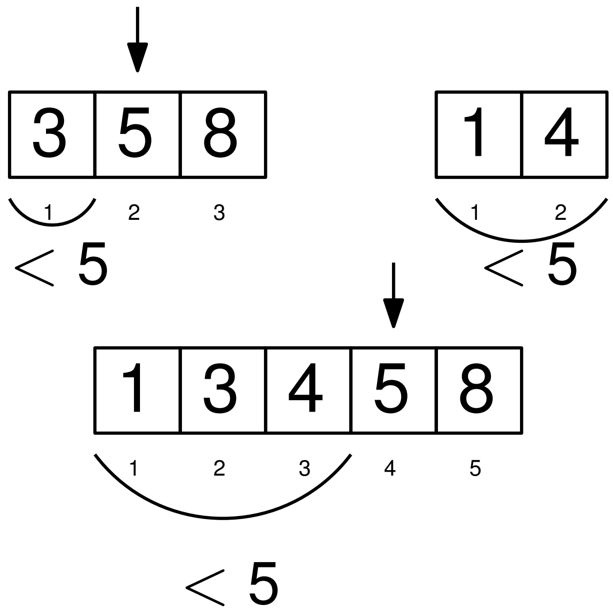
1	3	4	5	8

Parallel Merging



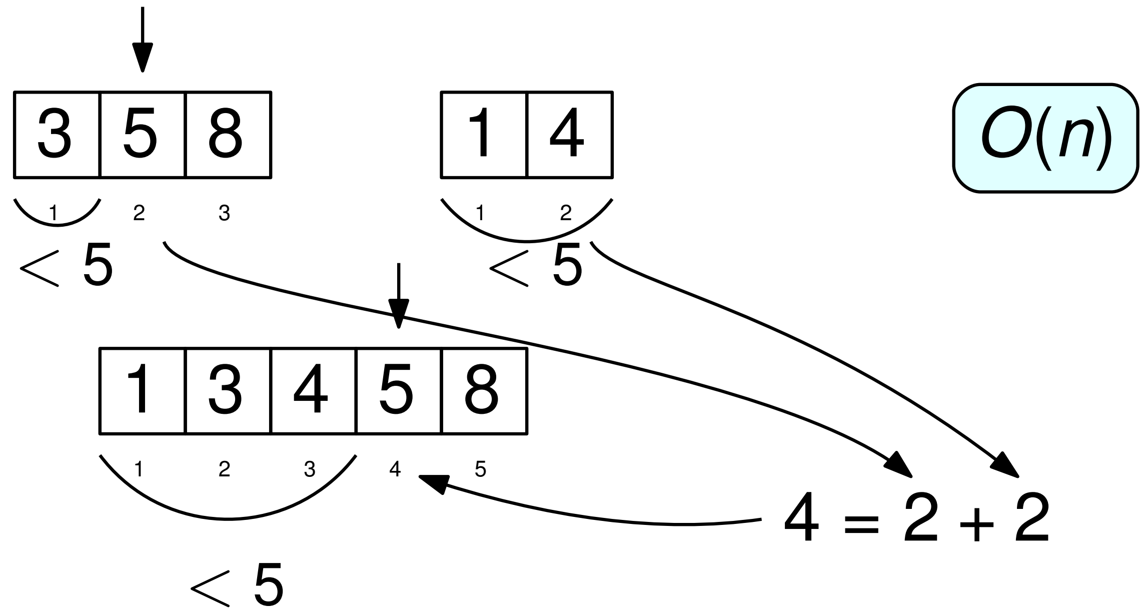
$O(n)$

Parallel Merging

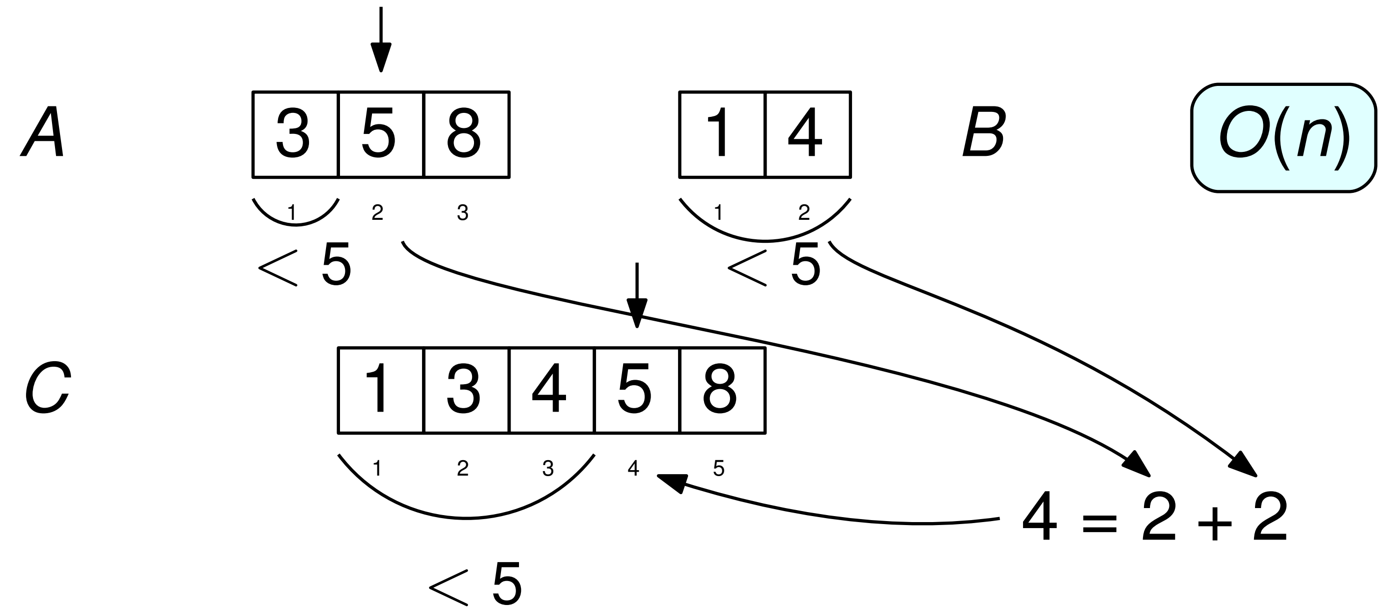


$O(n)$

Parallel Merging

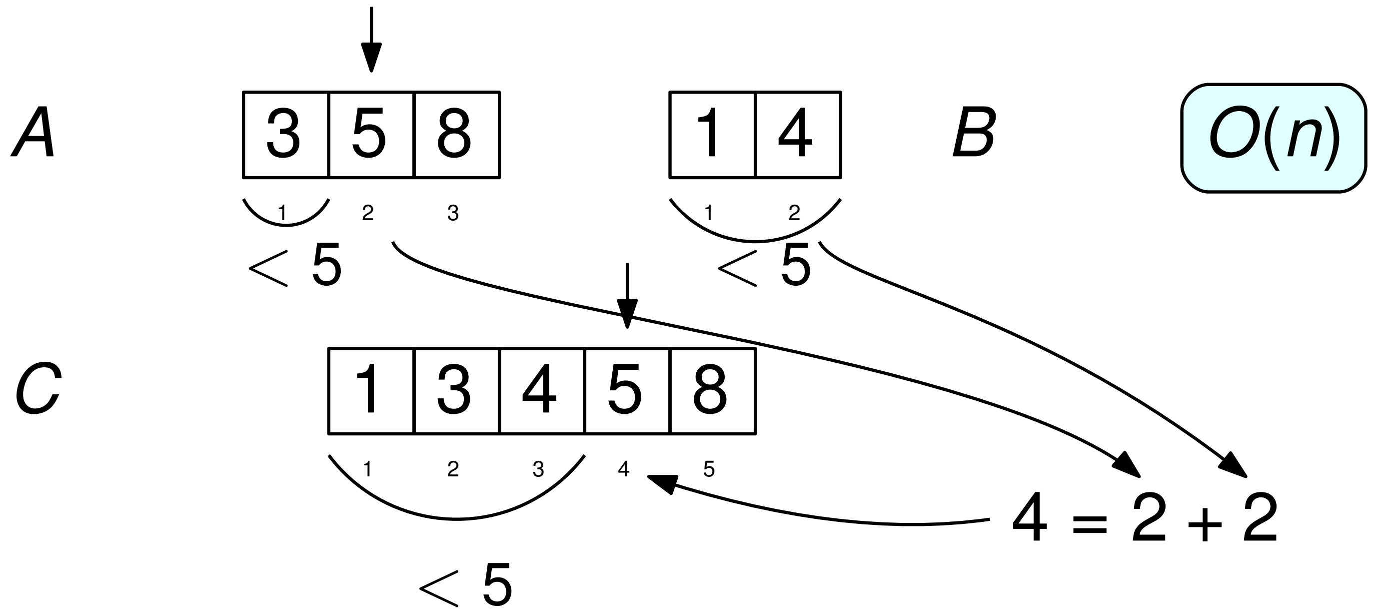


Parallel Merging



function MERGE(A, B, C)
    for $i = 1$ to $|A|$ **in parallel do**
        $k = i + \text{PREDECESSOR}(A[i], B)$
        $C[k] = A[i]$
    for $j = 1$ to $|B|$ **in parallel do**
        $k = j + \text{PREDECESSOR}(B[j], A)$
        $C[k] = B[j]$

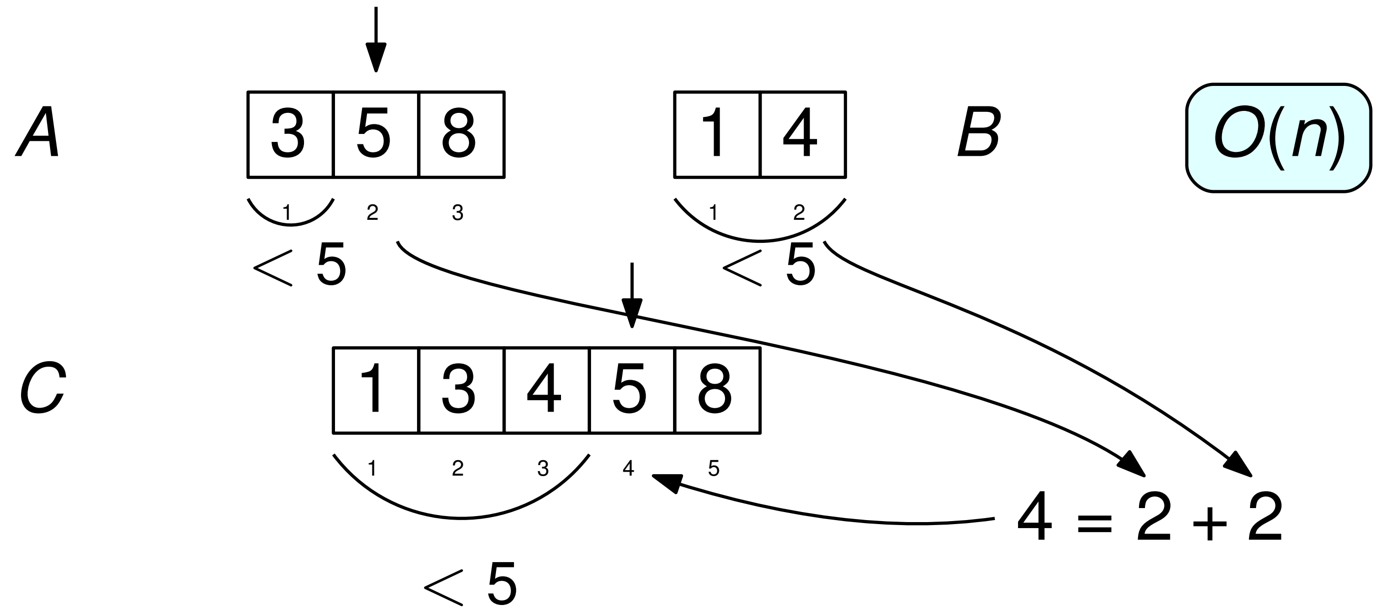
Parallel Merging



function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i + \text{PREDECESSOR}(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j + \text{PREDECESSOR}(B[j], A)$
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  for $i = 1$ to $|A|$ **do**
    if $A[i] > x$ **then**
      return $i - 1$
  return $|A|$

Parallel Merging

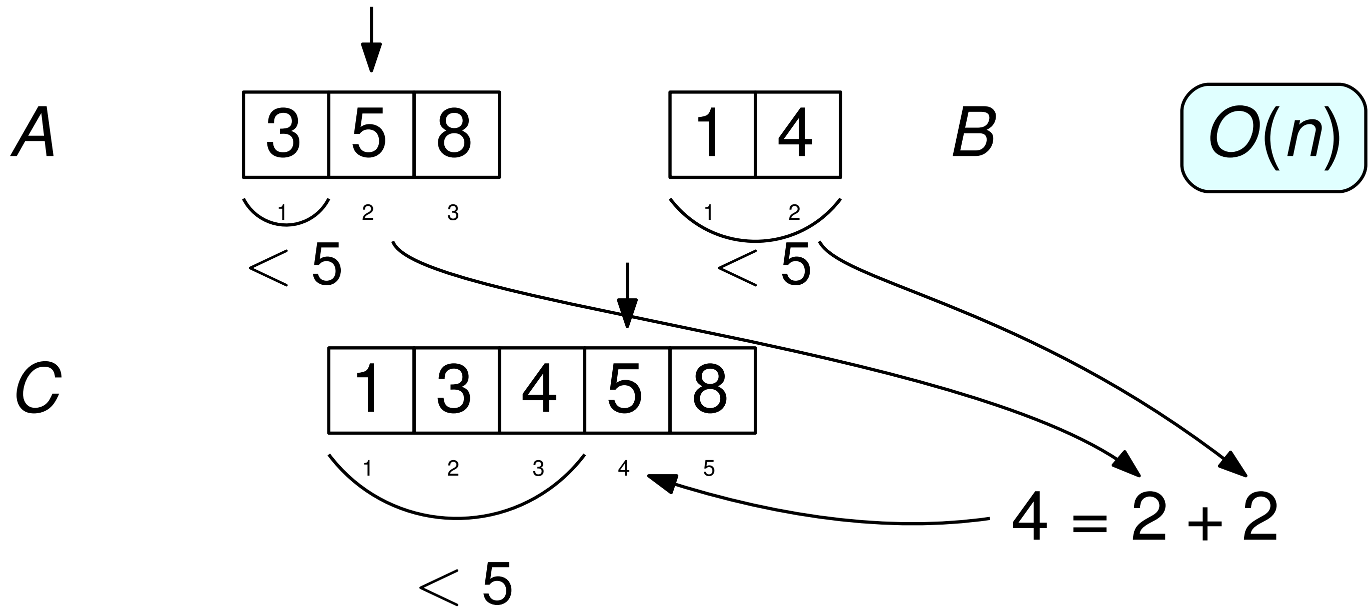


function MERGE(A, B, C)
    for $i = 1$ to $|A|$ **in parallel do**
        $k = i + \text{PREDECESSOR}(A[i], B)$
        $C[k] = A[i]$
    for $j = 1$ to $|B|$ **in parallel do**
        $k = j + \text{PREDECESSOR}(B[j], A)$
        $C[k] = B[j]$

function PREDECESSOR(x, A)
    for $i = 1$ to $|A|$ **do**
        if $A[i] > x$ **then**
            return $i - 1$
    return $|A|$

Still $O(n)$

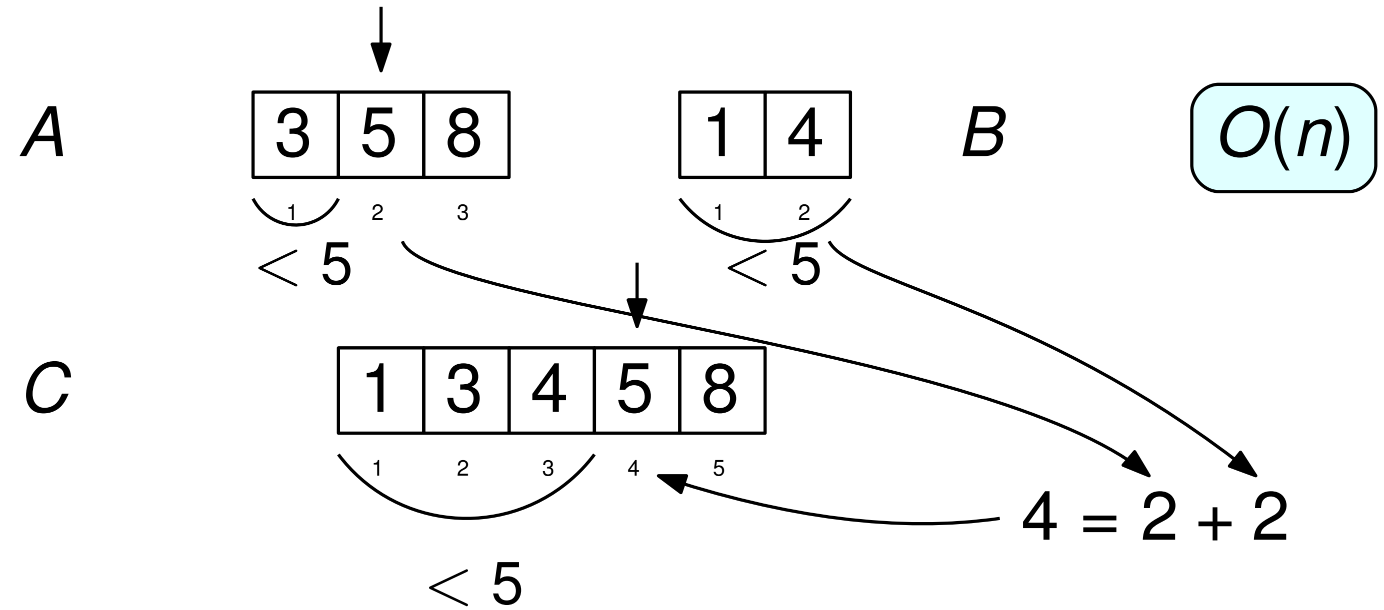
Parallel Merging



function MERGE(A, B, C)
  for i = 1 to $|A|$ **in parallel do**
    $k = i$ + PREDECESSOR($A[i]$, B)
    $C[k] = A[i]$
  for j = 1 to $|B|$ **in parallel do**
    $k = j$ + PREDECESSOR($B[j]$, A)
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

Parallel Merging

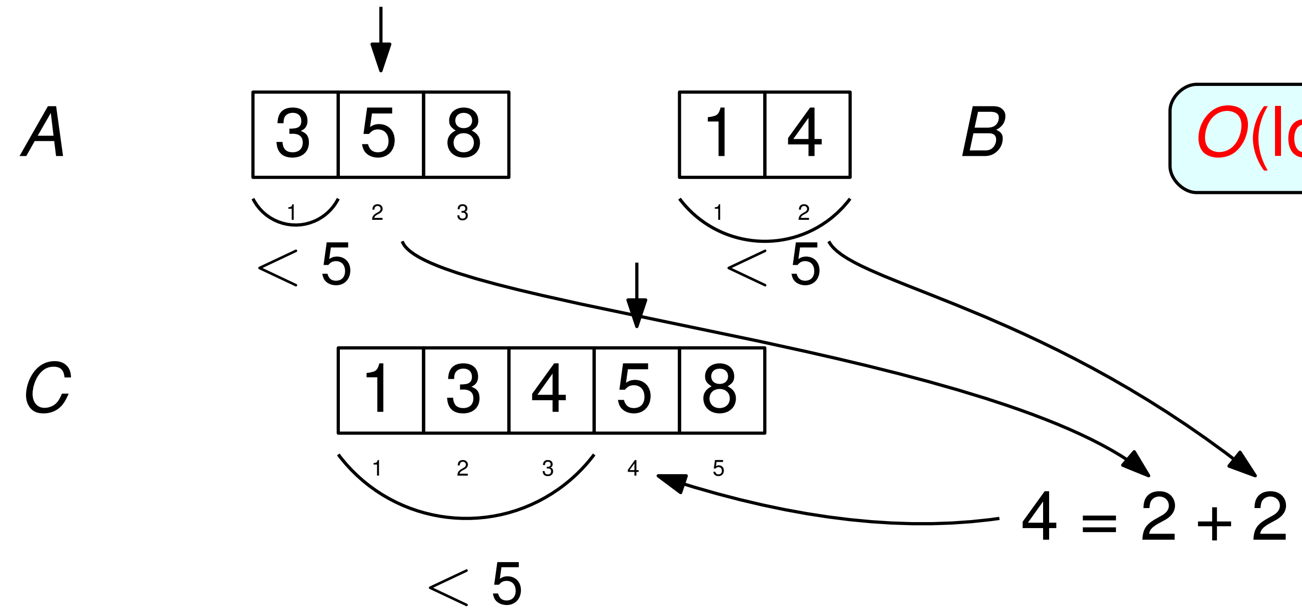


function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i +$ PREDECESSOR($A[i], B$)
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j +$ PREDECESSOR($B[j], A$)
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

$O(\log n)$

Parallel Merging



$O(\log n)$

function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i + \text{PREDECESSOR}(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j + \text{PREDECESSOR}(B[j], A)$
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

$O(\log n)$

Work vs Parallel Time

Time using p processors: $T_p(n)$?

Work vs Parallel Time

Time using p processors: $T_p(n)$?

- Work: Total # of operations = Sequential runtime: $W(n) = T_1(n)$
- (Parallel) Time: # of operations of slowest thread: $T(n) = T_\infty(n)$

Work vs Parallel Time

Time using p processors: $T_p(n)$?

- Work: Total # of operations = Sequential runtime: $W(n) = T_1(n)$
- (Parallel) Time: # of operations of slowest thread: $T(n) = T_\infty(n)$

Brent's Scheduling Principle:

$$T_p = O\left(\frac{W(n)}{p} + T_\infty(n)\right)$$

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

```
function PREFIX-SUMS(A, i, j)
    if i ≥ j then return                                   ▷ Base case
    mid = ⌊(i+j)/2⌋
    spawn
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS(A, mid + 1, j)
    sync
    for k = mid + 1 to j in parallel do
        A[k] = A[k] + A[mid]
```

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS($A, mid + 1, j$)
  sync
  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

$$\begin{aligned} W &= 2W(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function PREFIX-SUMS(A, i, j)
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    spawn
        PREFIX-SUMS(A, i, mid)
        PREFIX-SUMS($A, mid + 1, j$)
    sync
    for $k = mid + 1$ to j **in parallel do**
        $A[k] = A[k] + A[mid]$

$$\begin{aligned} W &= 2W(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

$$\begin{aligned} T_\infty(n) &= T_\infty(n/2) + O(1) \\ &= O(\log n) \end{aligned}$$

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS($A, mid + 1, j$)
  sync
  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

$$\begin{aligned} W &= 2W(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

$$\begin{aligned} T_\infty(n) &= T_\infty(n/2) + O(1) \\ &= O(\log n) \end{aligned}$$

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

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i + \text{PREDECESSOR}(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j + \text{PREDECESSOR}(B[j], A)$
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function MERGE(A, B, C)
    for $i = 1$ to $|A|$ **in parallel do**
        $k = i + \text{PREDECESSOR}(A[i], B)$
        $C[k] = A[i]$
    for $j = 1$ to $|B|$ **in parallel do**
        $k = j + \text{PREDECESSOR}(B[j], A)$
        $C[k] = B[j]$

$$W(n) = O(n \log n)$$
$$T_\infty(n) = O(\log n)$$

function PREDECESSOR(x, A)
    return BINARYSEARCH(x, A)

$$W(n) = O(\log n)$$
$$T_\infty(n) = O(\log n)$$

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i +$ PREDECESSOR$(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j +$ PREDECESSOR$(B[j], A)$
    $C[k] = B[j]$

function PREDECESSOR(x, A)
  return BINARYSEARCH(x, A)

$$W(n) = O(\log n)$$
$$T_\infty(n) = O(\log n)$$

$$W(n) = O(n \log n)$$
$$T_\infty(n) = O(\log n)$$

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

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function MERGESORT(A, i, j)
    if $i \geq j$ **then return** $\triangleright$ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do {
        MERGESORT(A, i, mid)
        MERGESORT($A, mid + 1, j$)
    }
    MERGE(A, i, mid, j)

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function MERGESORT(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  in parallel do {
    MERGESORT(A, i, mid)
    MERGESORT($A, mid + 1, j$)
  }
  MERGE(A, i, mid, j)

$$\begin{aligned} W(n) &= 2W(n) + O(n \log n) \\ &= O(n \log^2 n) \end{aligned}$$

$$\begin{aligned} T_\infty(n) &= T_\infty(n/2) + O(\log n) \\ &= O(\log^2 n) \end{aligned}$$

Time using p processors

$$\boxed{T_p(n) = O\left(\frac{W(n)}{p} + T_\infty(n)\right)}$$

function MERGESORT(A, i, j)
    if $i \geq j$ **then return** ▷ Base case
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
    in parallel do {
        MERGESORT(A, i, mid)
        MERGESORT($A, mid + 1, j$)
    }
    MERGE(A, i, mid, j)

$$\begin{aligned} W(n) &= 2W(n) + O(n \log n) \\ &= O(n \log^2 n) \end{aligned}$$

$$\begin{aligned} T_\infty(n) &= T_\infty(n/2) + O(\log n) \\ &= O(\log^2 n) \end{aligned}$$

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