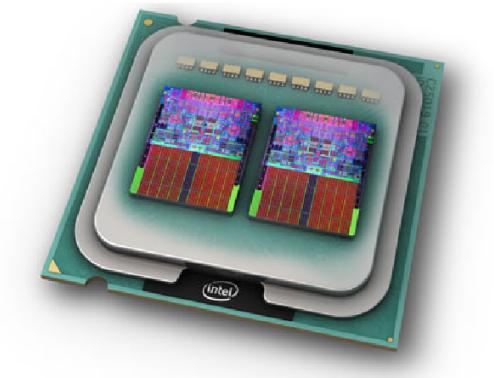
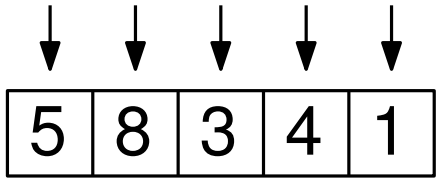
INFORMATION & COMPUTER SCIENCES
UNIVERSITY of HAWAI'I at MĀNOA

AlgoPARC

ICS 443: Parallel Algorithms

Prof. Nodari Sitchinava





Lecture 5: Prefix Sums Applications

Reminder: Prefix Sums

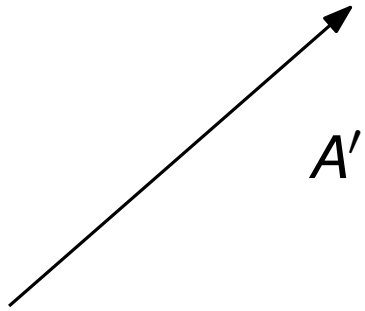
A :	3	1	5	2	3	4	1	5	7	2	1	5	2	1	6	9
A' :	3	4	9	11	14	18	19	24	31	33	34	39	41	42	48	57

$$A'[i] = \sum_{k=1}^{i} A[i]$$

Reminder: Prefix Sums

A : 3 1 5 2 3 4 1 5 7 2 1 5 2 1 6 9

A' : 3 4 9 11 14 18 19 24 31 33 34 39 41 42 48 57



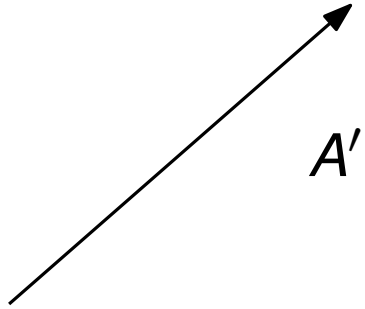
$$A'[i] = \sum_{k=1}^{i} A[i]$$

$$A'[6] = A[1] + A[2] + A[3] + A[4] + A[5] + A[6]$$

Reminder: Prefix Sums

A : 3 1 5 2 3 4 1 5 7 2 1 5 2 1 6 9

A' : 3 4 9 11 14 18 19 24 31 33 34 39 41 42 48 57



$$A'[i] = \sum_{k=1}^{i} A[i]$$

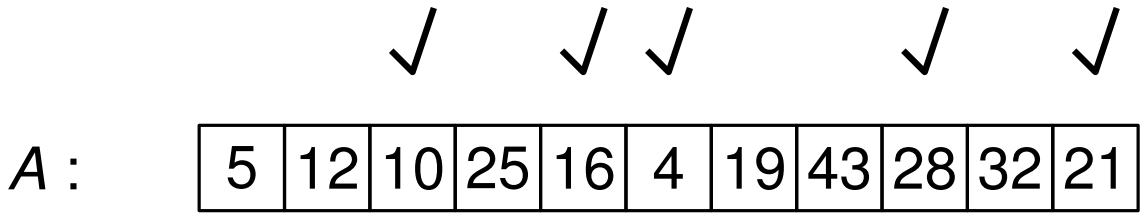
$$A'[6] = A[1] + A[2] + A[3] + A[4] + A[5] + A[6]$$

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

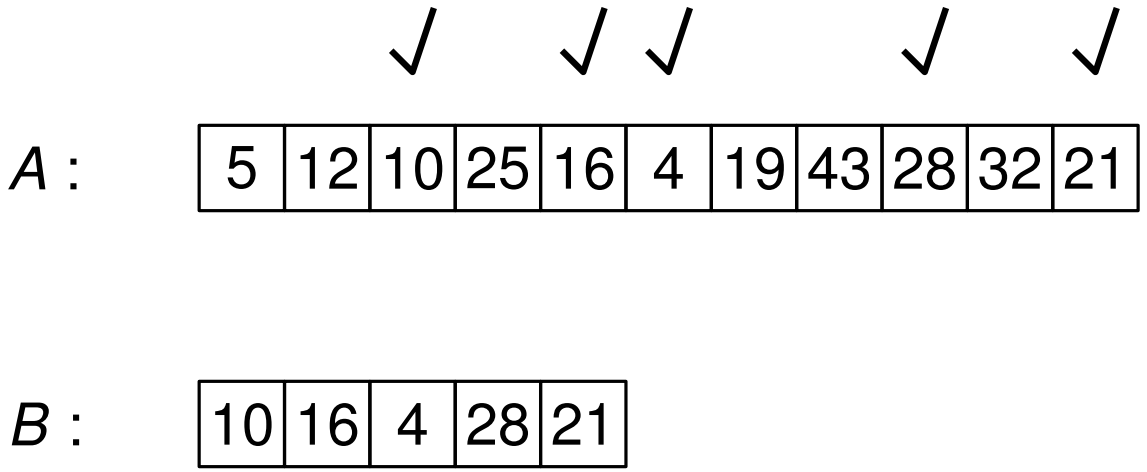
Application: FILTER

A :

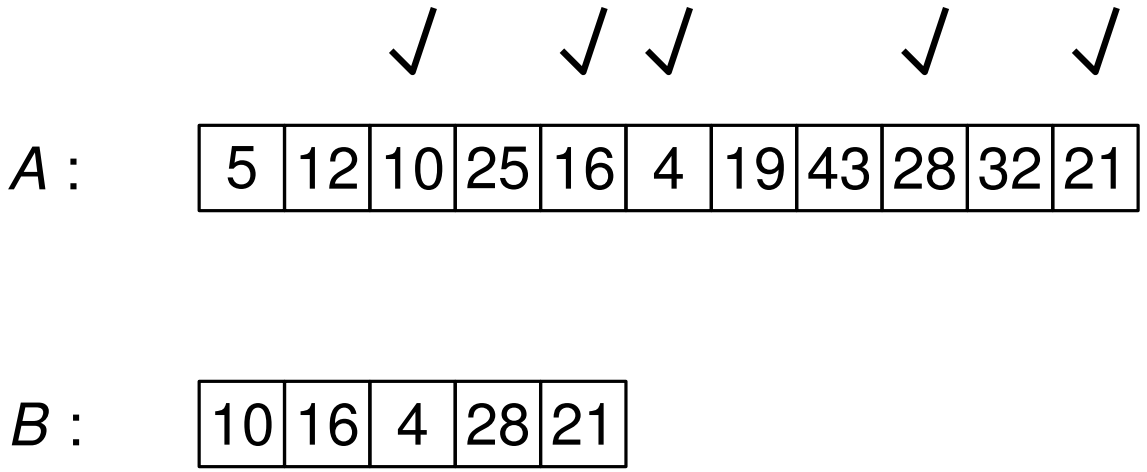
Application: FILTER



Application: FILTER



Application: FILTER




```
procedure FILTER(A[1..n])
    k = 0
    for i = 1 to n do
        if A[i] is checked then
            k = k + 1
            B[k] = A[i]
    return B[1..k]
```

Application: FILTER

flags :	0	0	1	0	1	1	0	0	1	0	1
A :	5	12	10	25	16	4	19	43	28	32	21

B :	10	16	4	28	21

```
procedure FILTER(A[1..n])
    k = 0
    for i = 1 to n do
        if A[i] is checked then
            k = k + 1
            B[k] = A[i]
    return B[1..k]
```

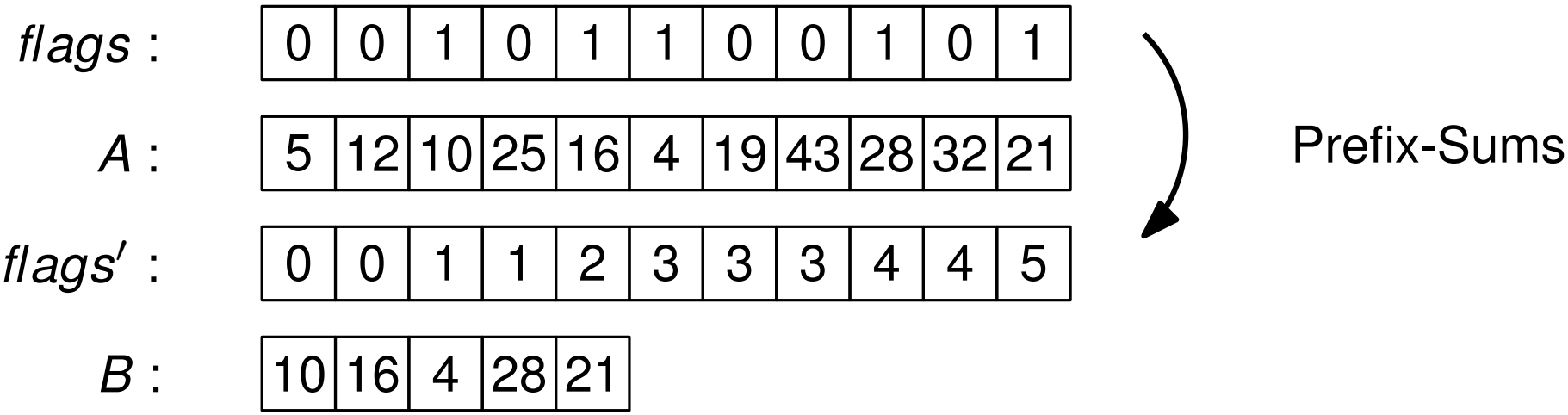
Application: FILTER

flags :	0	0	1	0	1	1	0	0	1	0	1
A :	5	12	10	25	16	4	19	43	28	32	21

B :	10	16	4	28	21

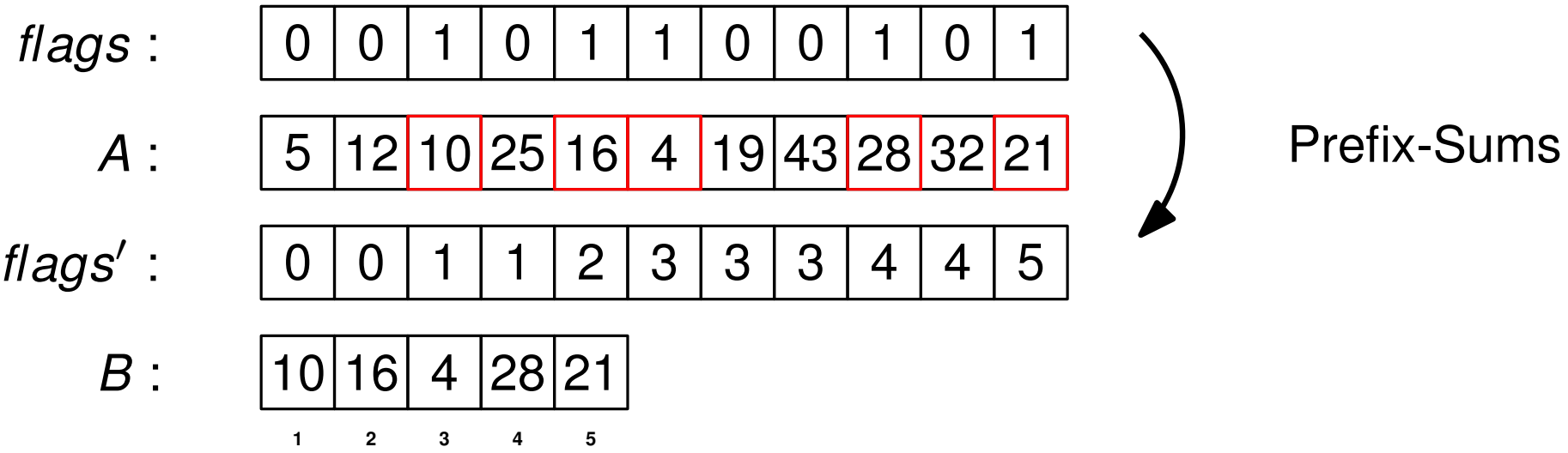
```
procedure FILTER(A[1..n], flags[1..n])
    k = 0
    for i = 1 to n do
        if flags[i] = 1 then
            k = k + 1
            B[k] = A[i]
    return B[1..k]
```

Application: FILTER



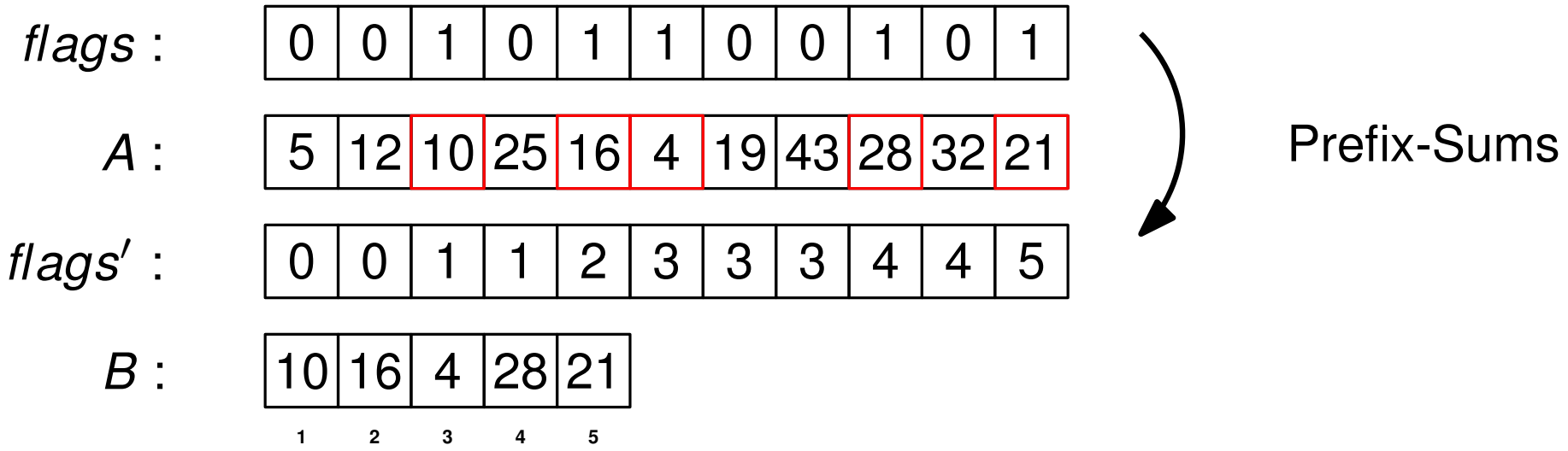

```
procedure FILTER(A[1..n], flags[1..n])
    k = 0
    for i = 1 to n do
        if flags[i] = 1 then
            k = k + 1
            B[k] = A[i]
    return B[1..k]
```

Application: FILTER



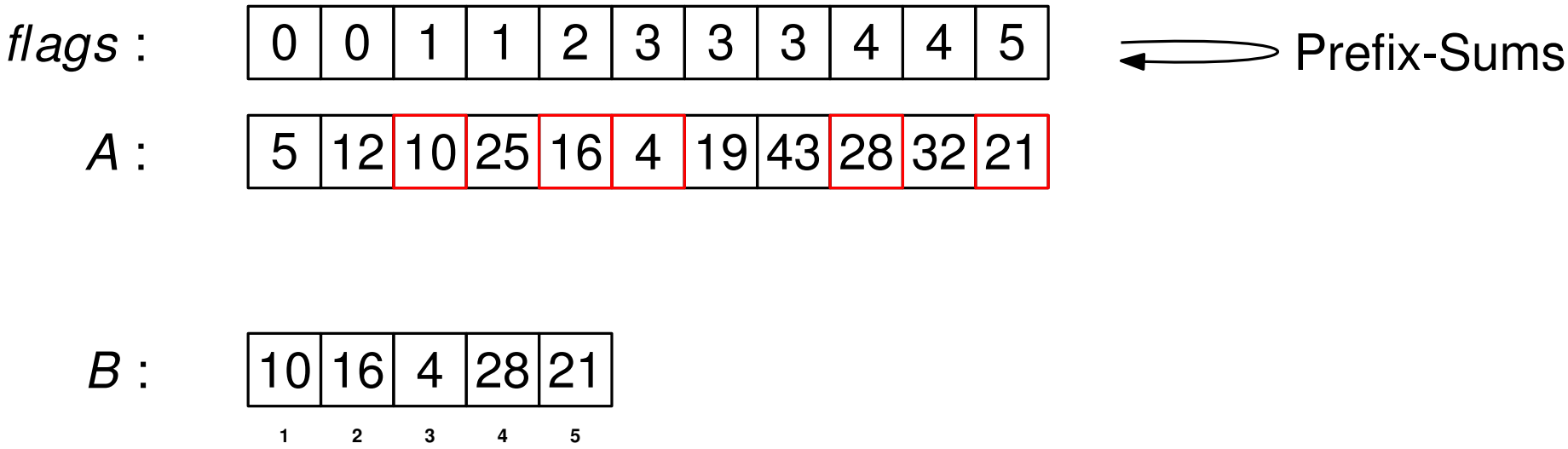

```
procedure FILTER(A[1..n], flags[1..n])
    k = 0
    for i = 1 to n do
        if flags[i] = 1 then
            k = k + 1
            B[k] = A[i]
    return B[1..k]
```

Application: FILTER



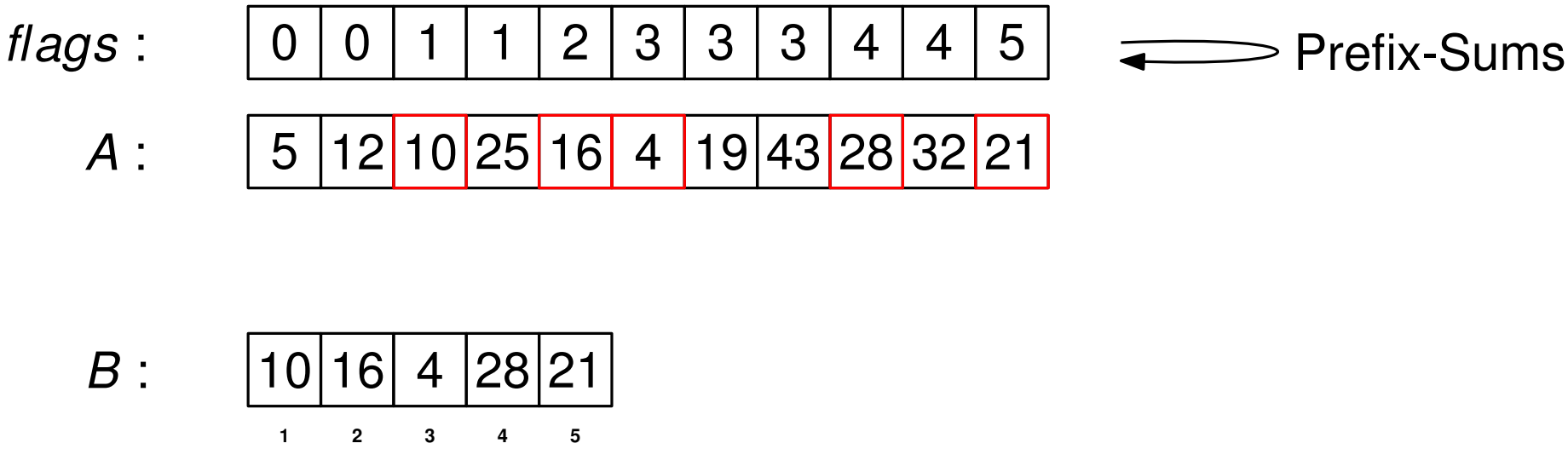

```
procedure FILTER(A[1..n], flags[1..n])
    flags′[1..n] = PREFIX-SUMS(flags[1..n])
    for i = 1 to n in parallel do
        if flags[i] = 1 then
            B[flags′[i]] = A[i]
    return B[1..flags′[n]]
```

Application: FILTER



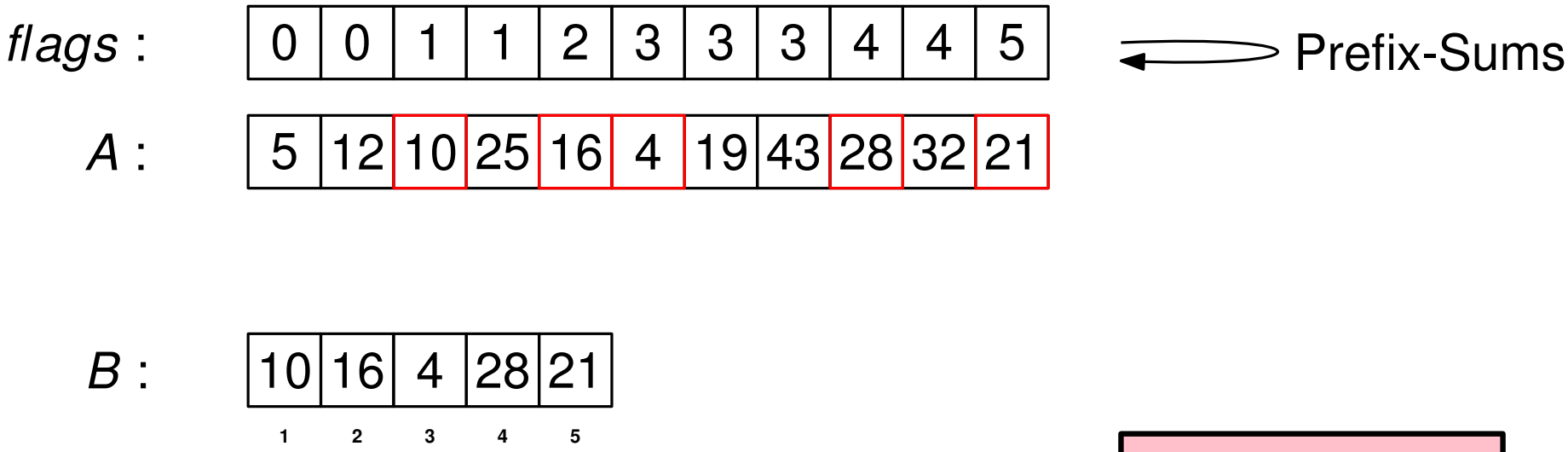

```
procedure FILTER(A[1..n], flags[1..n])
    flags'[1..n] = PREFIX-SUMS(flags[1..n])
    for i = 1 to n in parallel do
        if flags[i] = 1 then
            B[flags'[i]] = A[i]
    return B[1..flags'[n]]
```

Application: FILTER




```
procedure FILTER(A[1..n], flags[1..n])
    PREFIX-SUMS(flags[1..n])
    for i = 1 to n in parallel do
        if flags[i] ≠ flags[i − 1] then
            B[flags[i]] = A[i]
    return B[1..flags[n]]
```

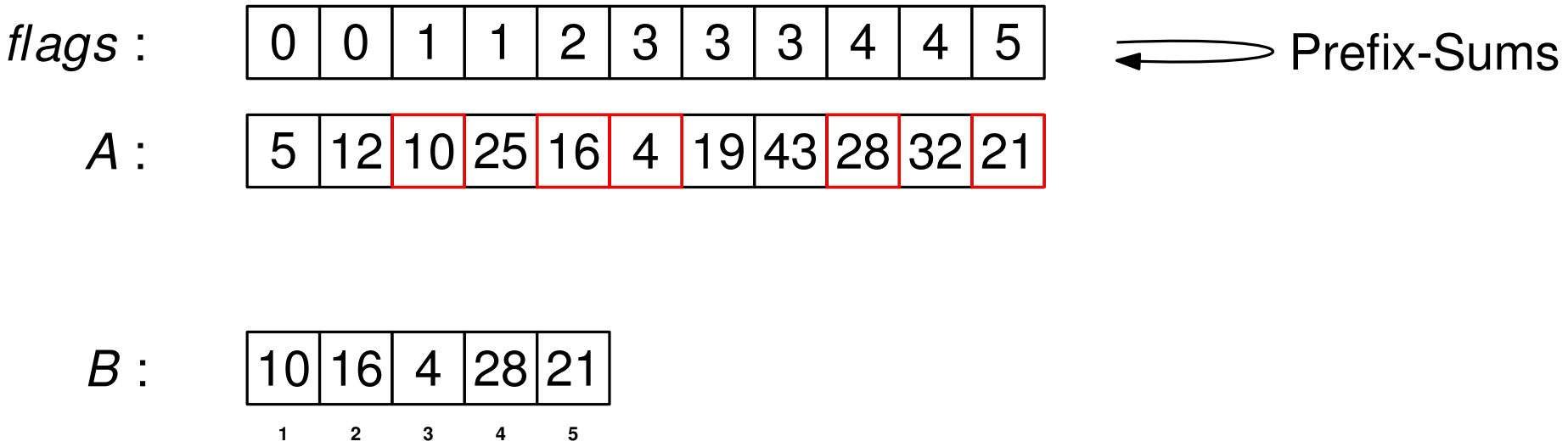
Application: FILTER




```
procedure FILTER(A[1..n], flags[1..n])
    PREFIX-SUMS(flags[1..n])
    for i = 1 to n in parallel do
        if flags[i] ≠ flags[i − 1] then
            B[flags[i]] = A[i]
    return B[1..flags[n]]
```

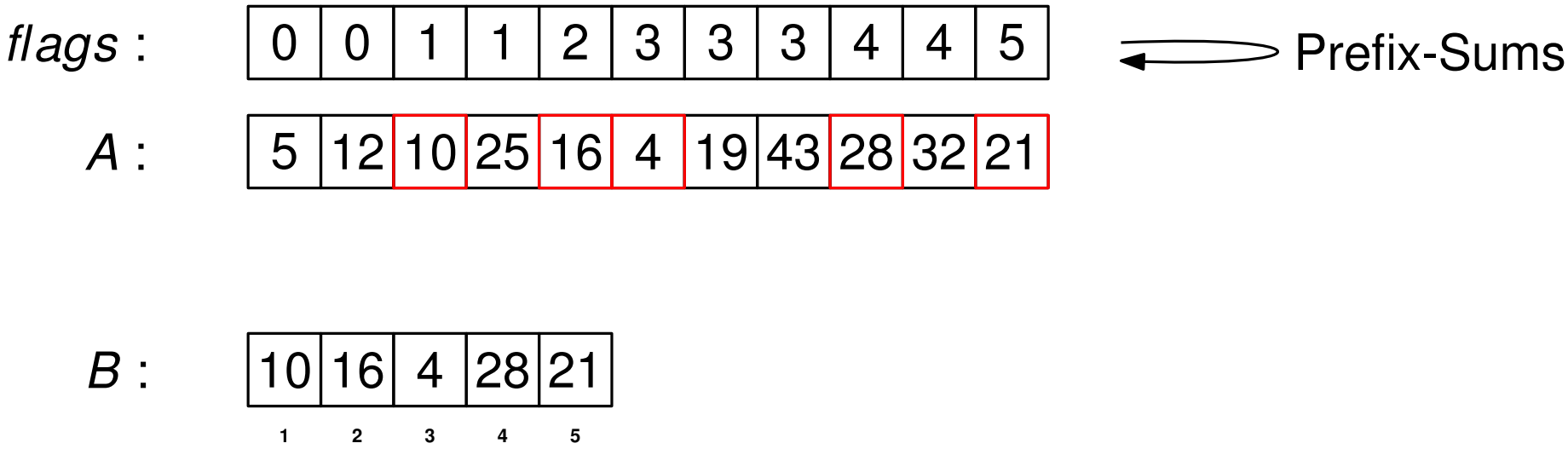
Array out-of-bounds error

Application: FILTER




```
procedure FILTER(A[1..n], flags[1..n])
    PREFIX-SUMS(flags[1..n])
    for i = 2 to n in parallel do
        if flags[i] ≠ flags[i − 1] then
            B[flags[i]] = A[i]
    if flags[1] == 1 then
        B[1] = A[1]
    return B[1..flags[n]]
```

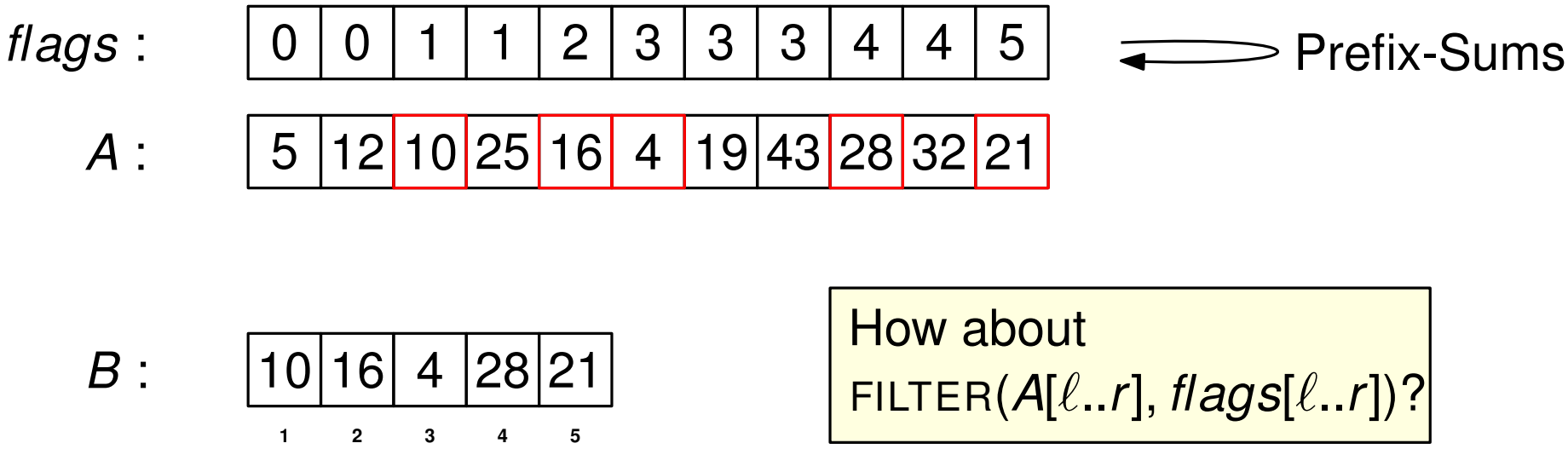
Application: FILTER




```
procedure FILTER(A[1..n], flags[1..n])
    PREFIX-SUMS(flags[1..n])
    for i = 2 to n in parallel do
        if flags[i] ≠ flags[i − 1] then
            B[flags[i]] = A[i]
    if flags[1] == 1 then
        B[1] = A[1]
    return B[1..flags[n]]
```

size of B

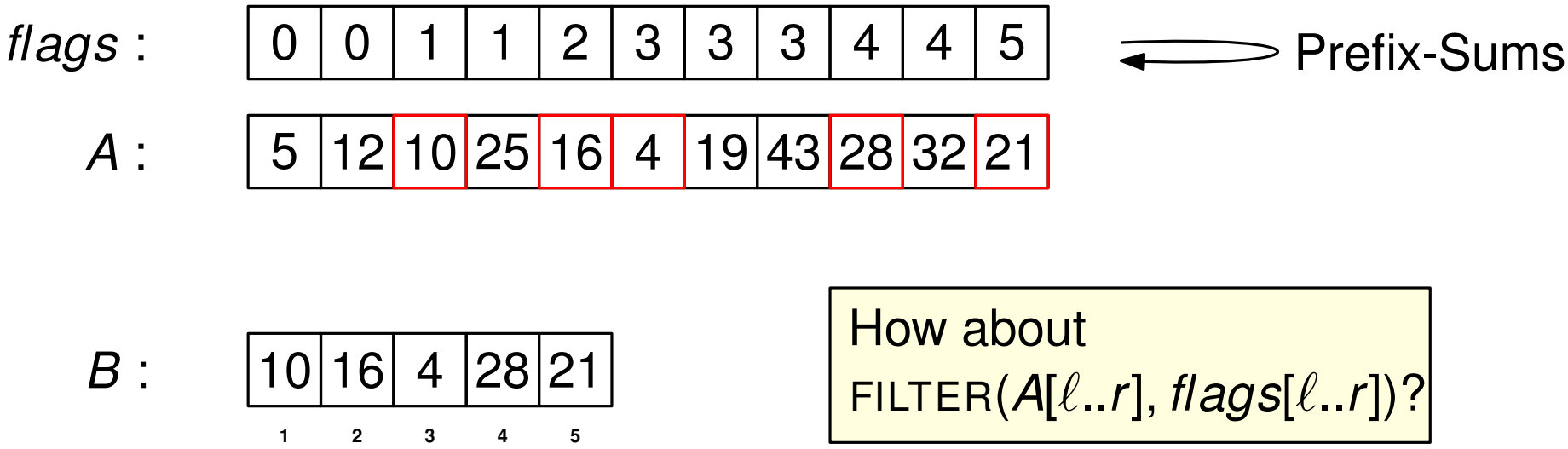
Application: FILTER



How about FILTER($A[\ell..r]$, $flags[\ell..r]$)?

procedure FILTER($A[1..n]$, $flags[1..n]$)
    PREFIX-SUMS($flags[1..n]$)
    for $i = 2$ to n **in parallel do**
        if $flags[i] \neq flags[i-1]$ **then**
            $B[flags[i]] = A[i]$
    if $flags[1] == 1$ **then**
        $B[1] = A[1]$
    return $B[1..flags[n]]$

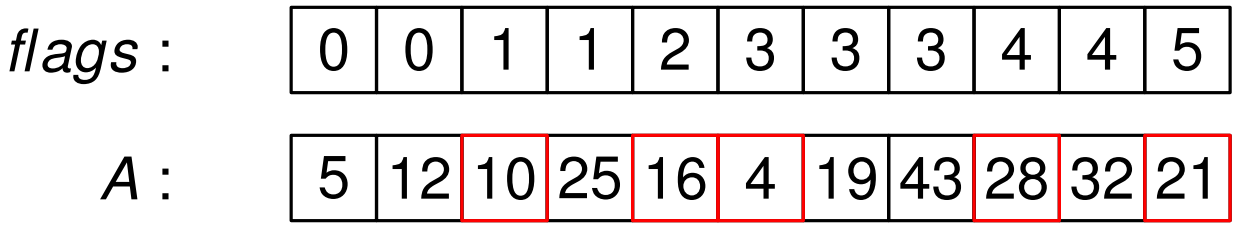
Application: FILTER

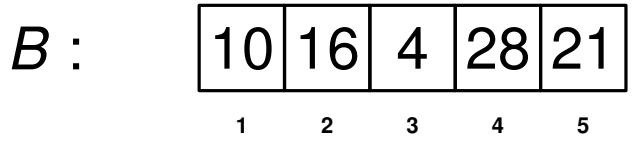


How about FILTER($A[\ell..r]$, $flags[\ell..r]$)?

procedure FILTER($A[\ell..r]$, $flags[\ell..r]$)
  PREFIX-SUMS($flags[\ell..r]$)
  for $i = \ell + 1$ to r **in parallel do**
    if $flags[i] \neq flags[i-1]$ **then**
      $B[flags[i]] = A[i]$
  if $flags[\ell] = 1$ **then**
    $B[1] = A[\ell]$
  return $B[1..flags[r]]$

Application: FILTER



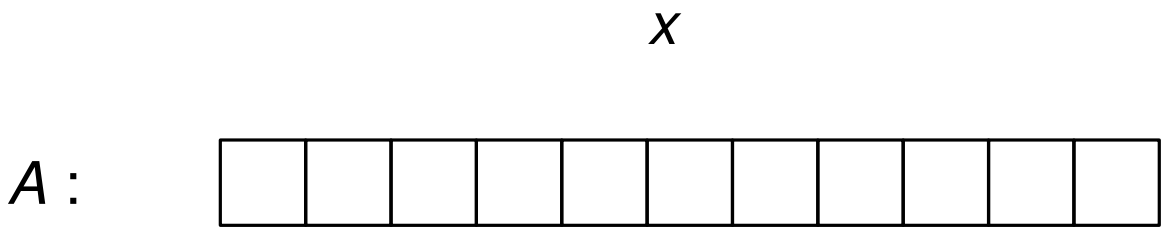


How about FILTER($A[\ell..r]$, $flags[\ell..r]$)?

```
procedure FILTER(A[ℓ..r], flags[ℓ..r])
    PREFIX-SUMS(flags[ℓ..r])
    for i = ℓ + 1 to r in parallel do
        if flags[i] ≠ flags[i − 1] then
            B[flags[i]] = A[i]
    if flags[ℓ] = 1 then
        B[1] = A[ℓ]
    return B[1..flags[r]]
```

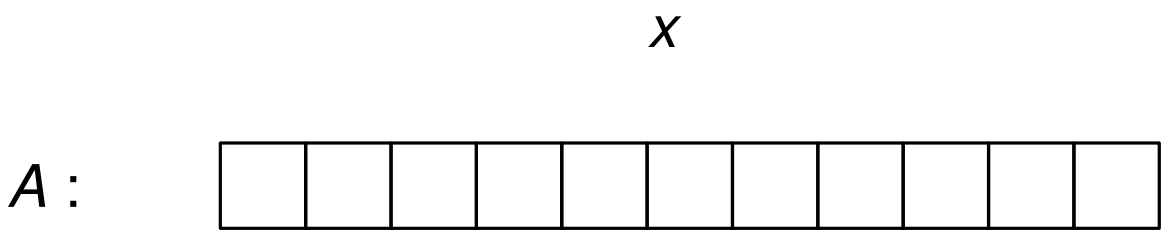
Prefix sums on a subarray

Application: BROADCAST



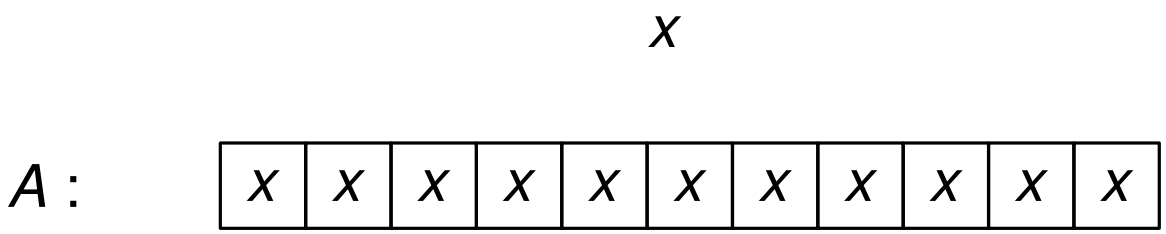
Application: BROADCAST

BROADCAST(x, $A[1..n]$)



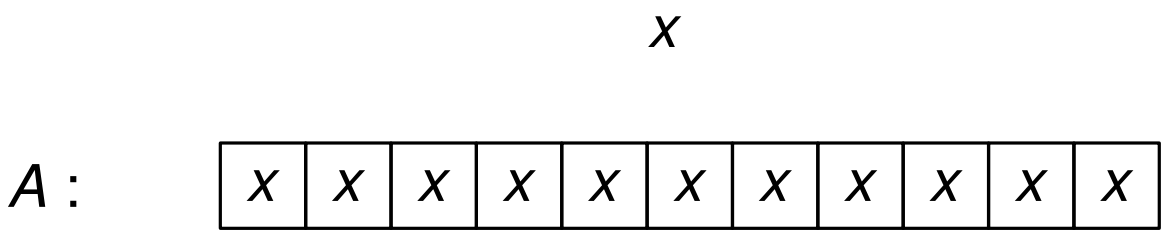
Application: BROADCAST

BROADCAST(x, $A[1..n]$)



Application: BROADCAST

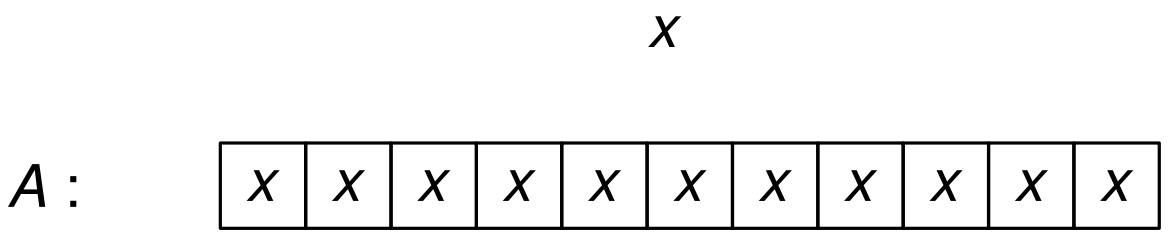
BROADCAST(x, $A[1..n]$)



CREW PRAM

Application: BROADCAST

BROADCAST(x, $A[1..n]$)

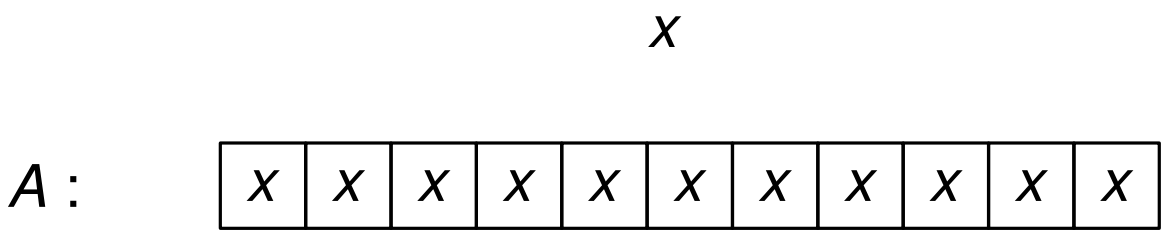


CREW PRAM

procedure CREW-BROADCAST(x, $A[1..n]$)
    for i = 1 to n **in parallel do**
        $A[i] = x$

Application: BROADCAST

BROADCAST(x, $A[1..n]$)



CREW PRAM

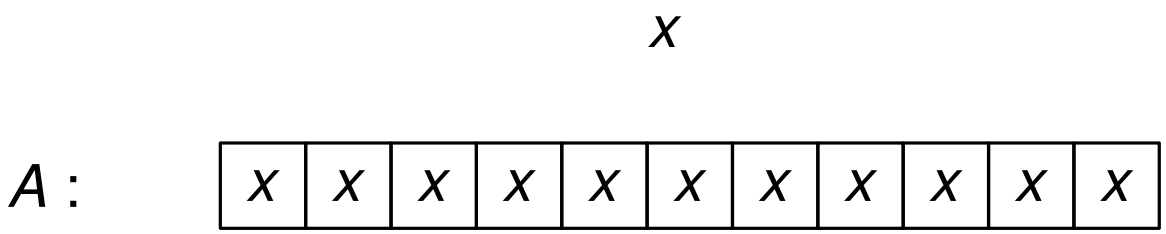
procedure CREW-BROADCAST(x, $A[1..n]$)
  for i = 1 to n **in parallel do**
    $A[i] = x$

Time $T(n) = O(1)$

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

Application: BROADCAST

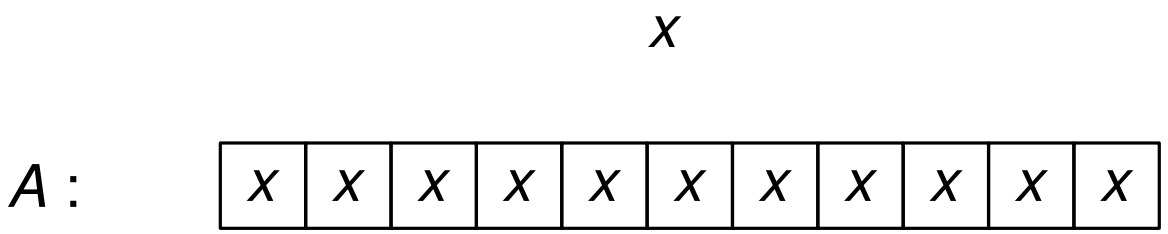
BROADCAST(x, $A[1..n]$)



EREW PRAM

Application: BROADCAST

BROADCAST(x, $A[1..n]$)

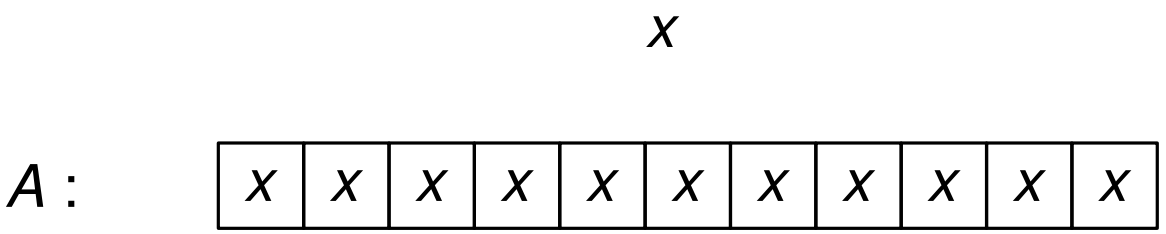


EREW PRAM

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)

Application: BROADCAST

BROADCAST(x, $A[1..n]$)



EREW PRAM

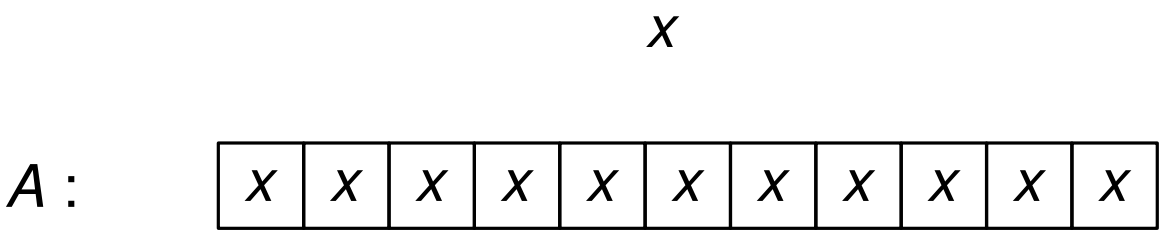
procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)

Time:

Work:

Application: BROADCAST

BROADCAST(x, $A[1..n]$)



EREW PRAM

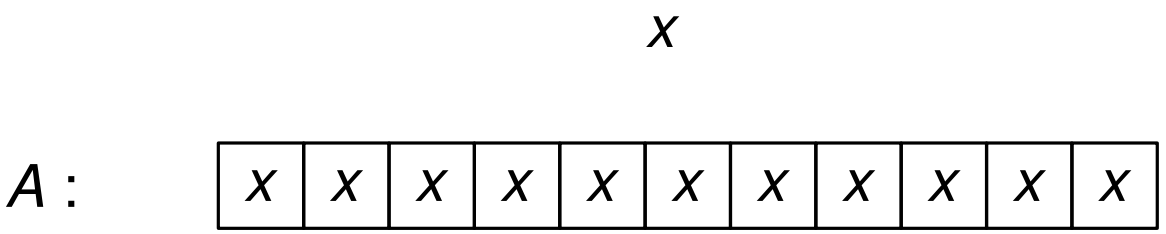
procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)

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

Work:

Application: BROADCAST

BROADCAST(x, $A[1..n]$)



EREW PRAM

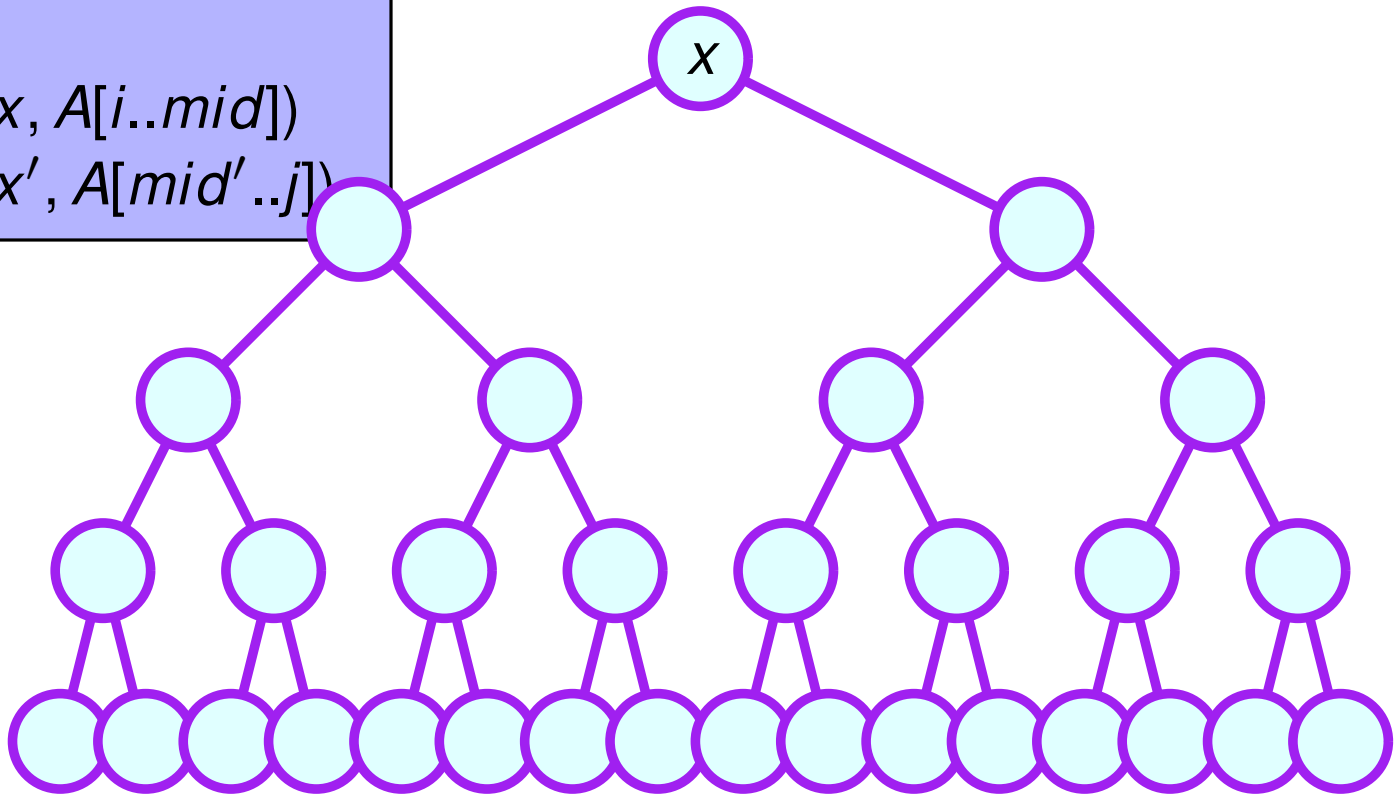
procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)

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

Work: $W(n) = 2W(n/2) + O(1)$
$= O(n)$

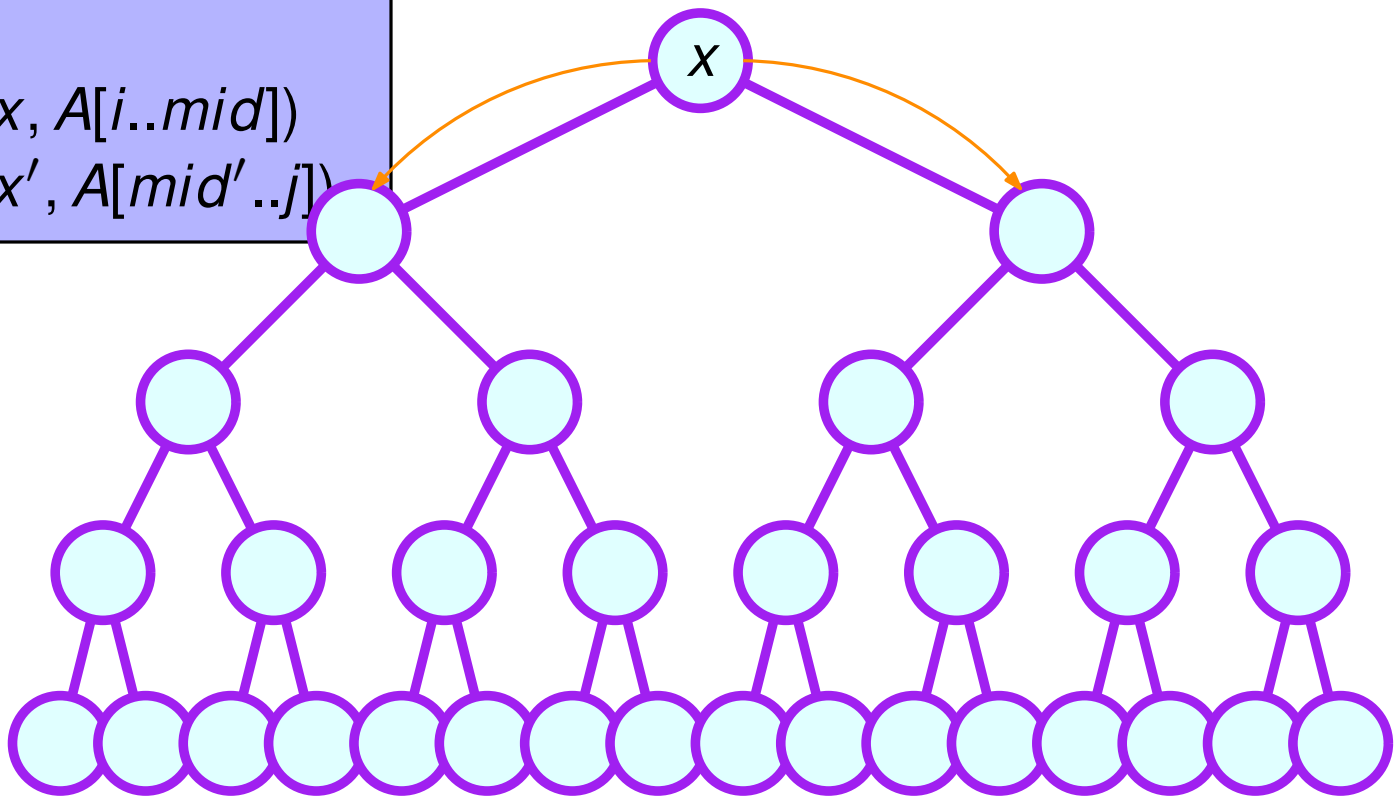
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



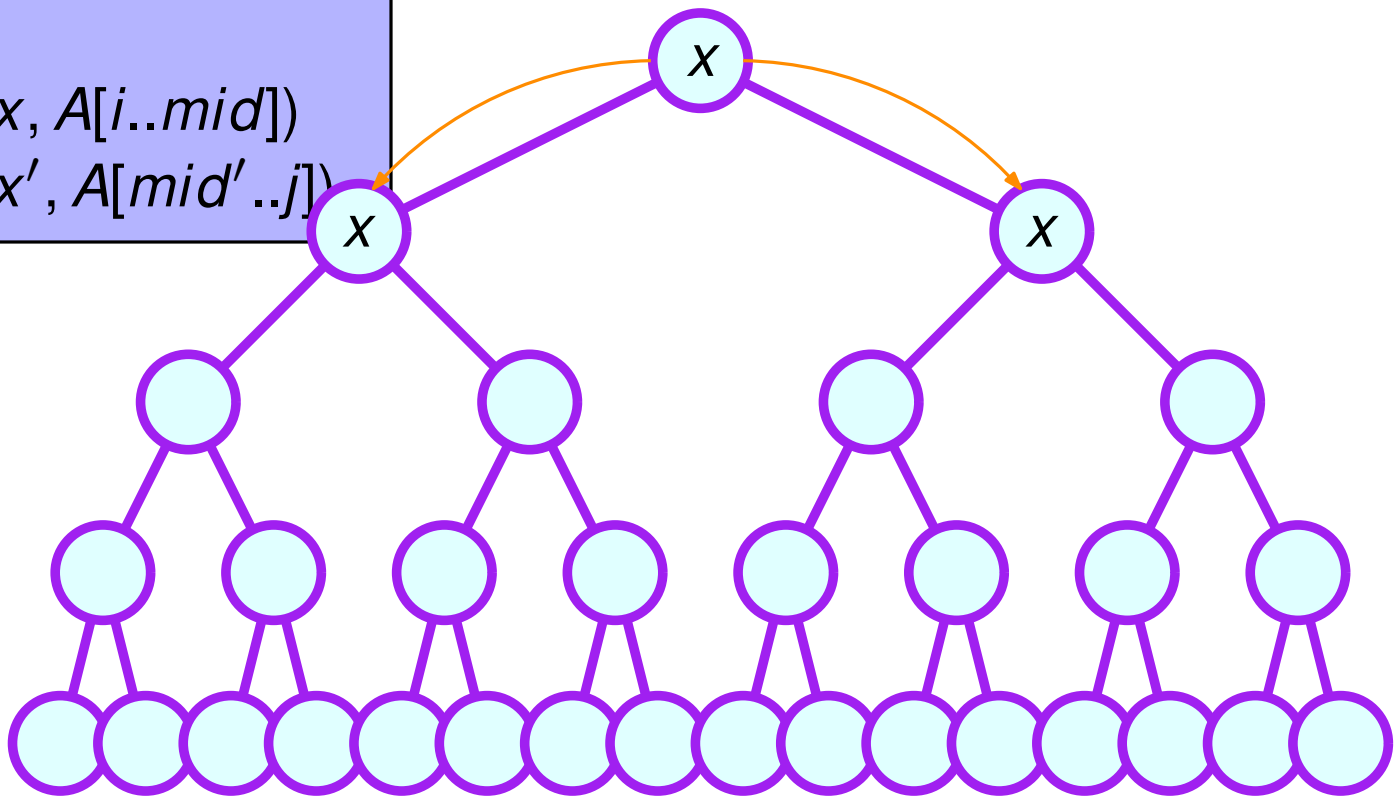
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



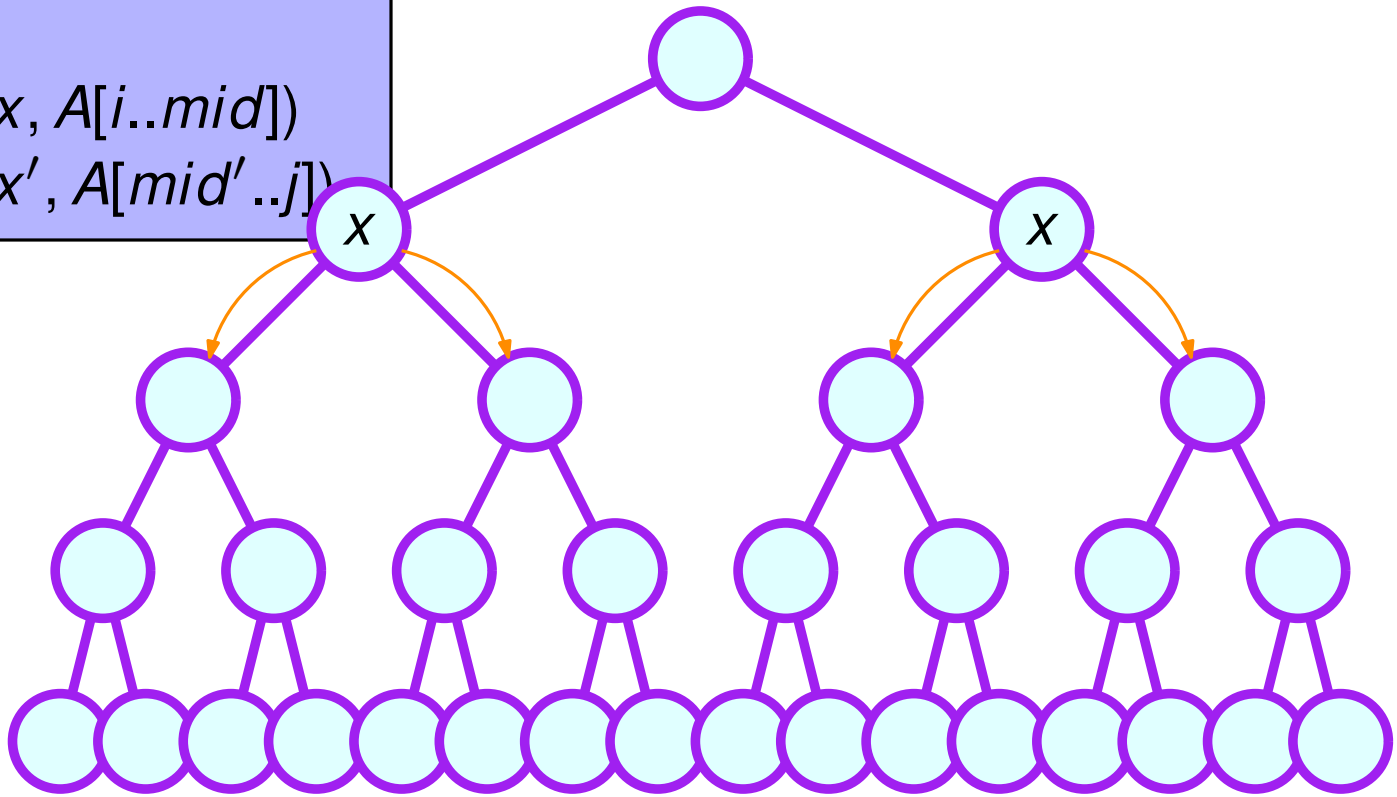
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



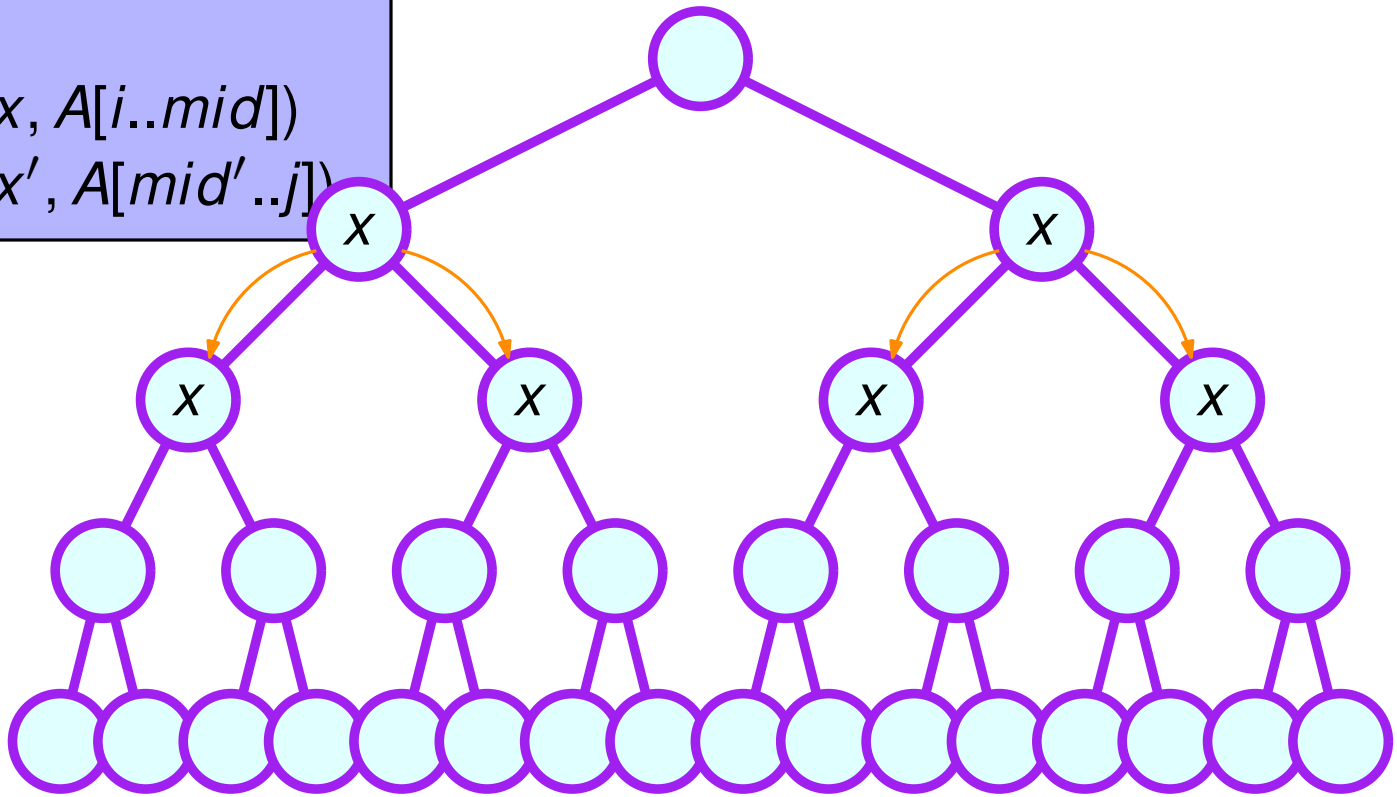
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



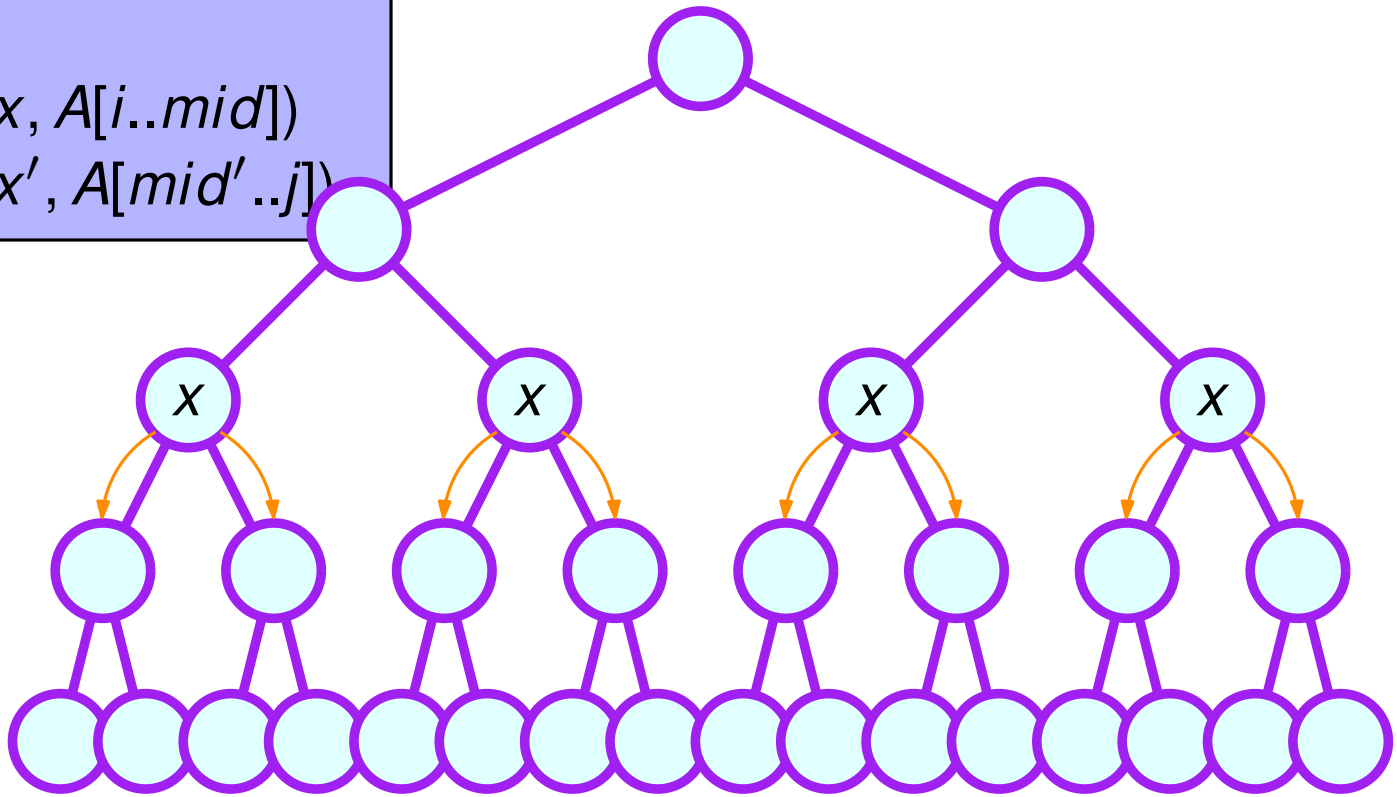
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



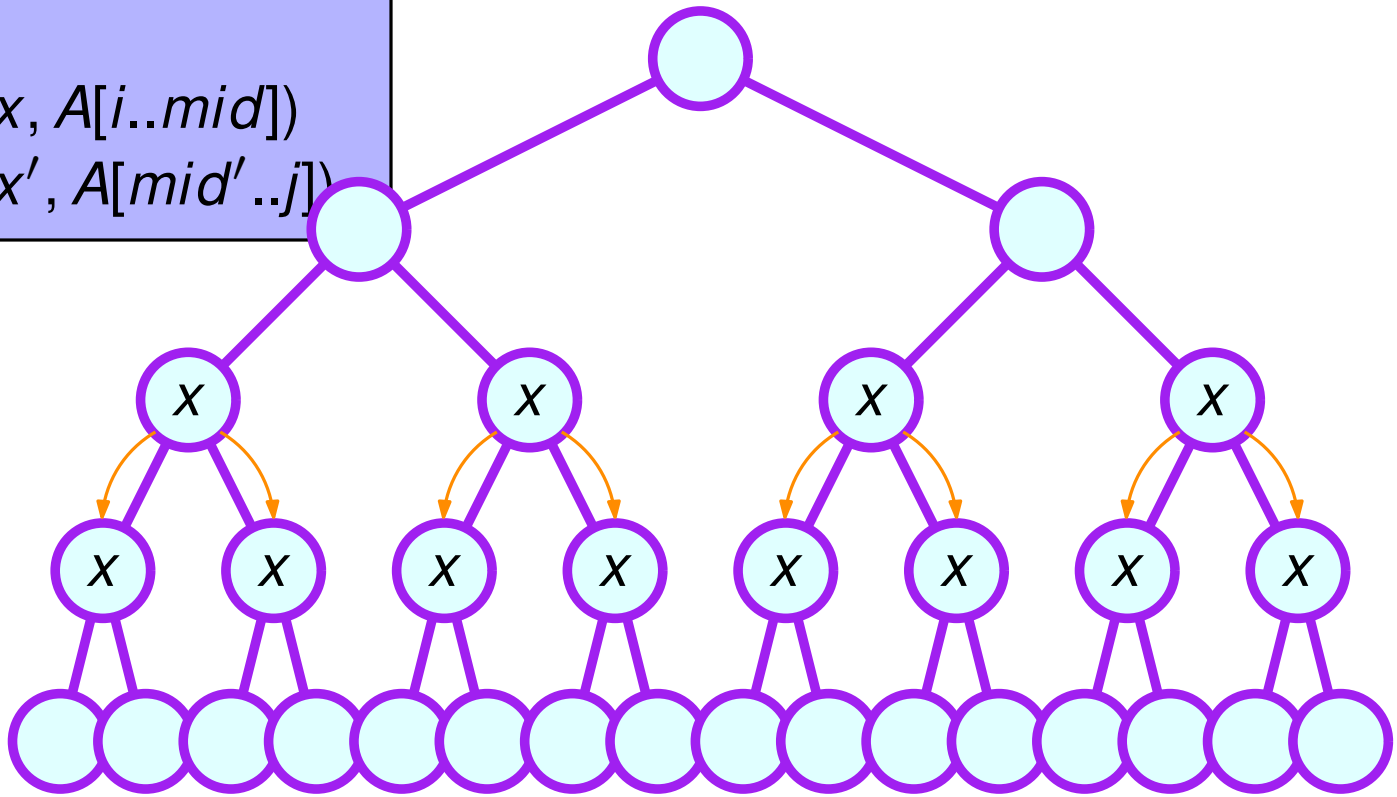
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



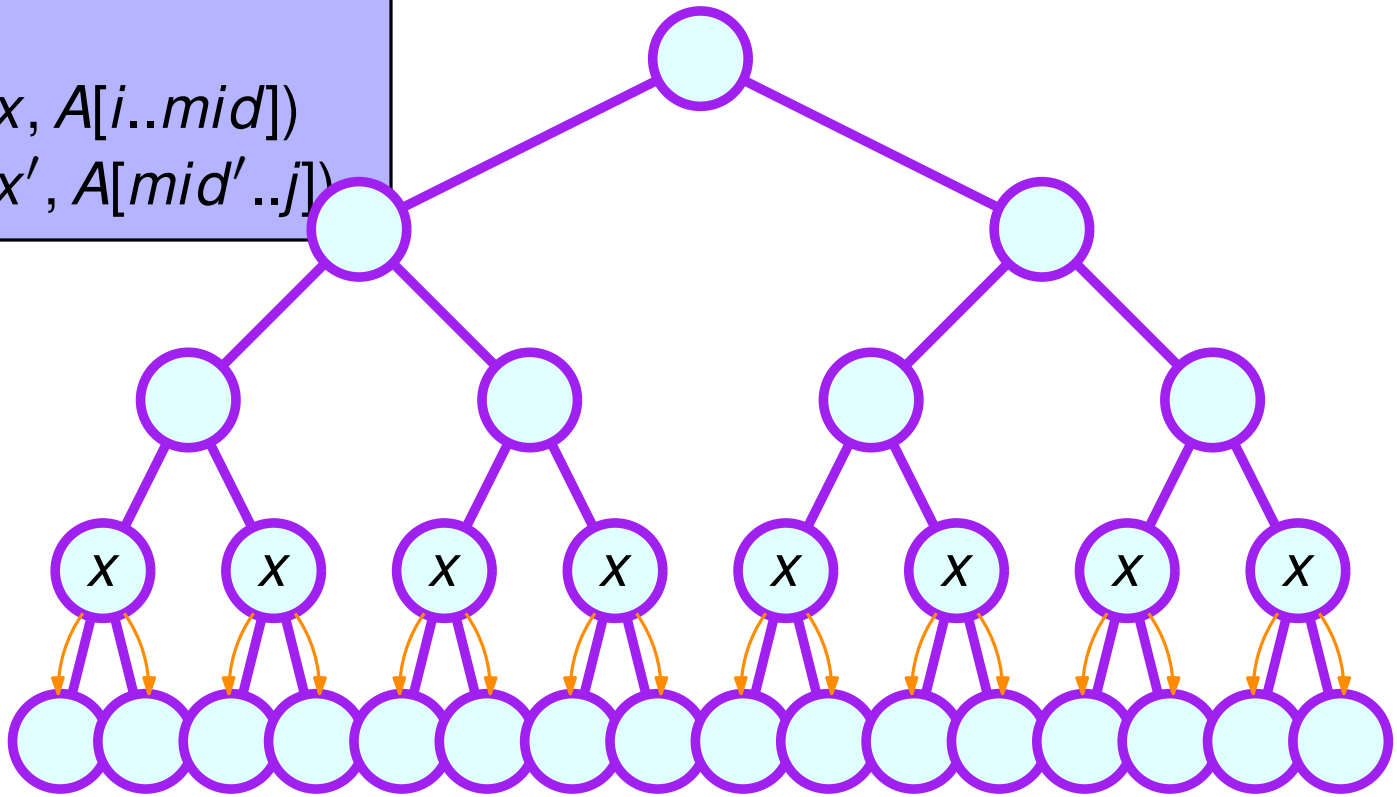
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



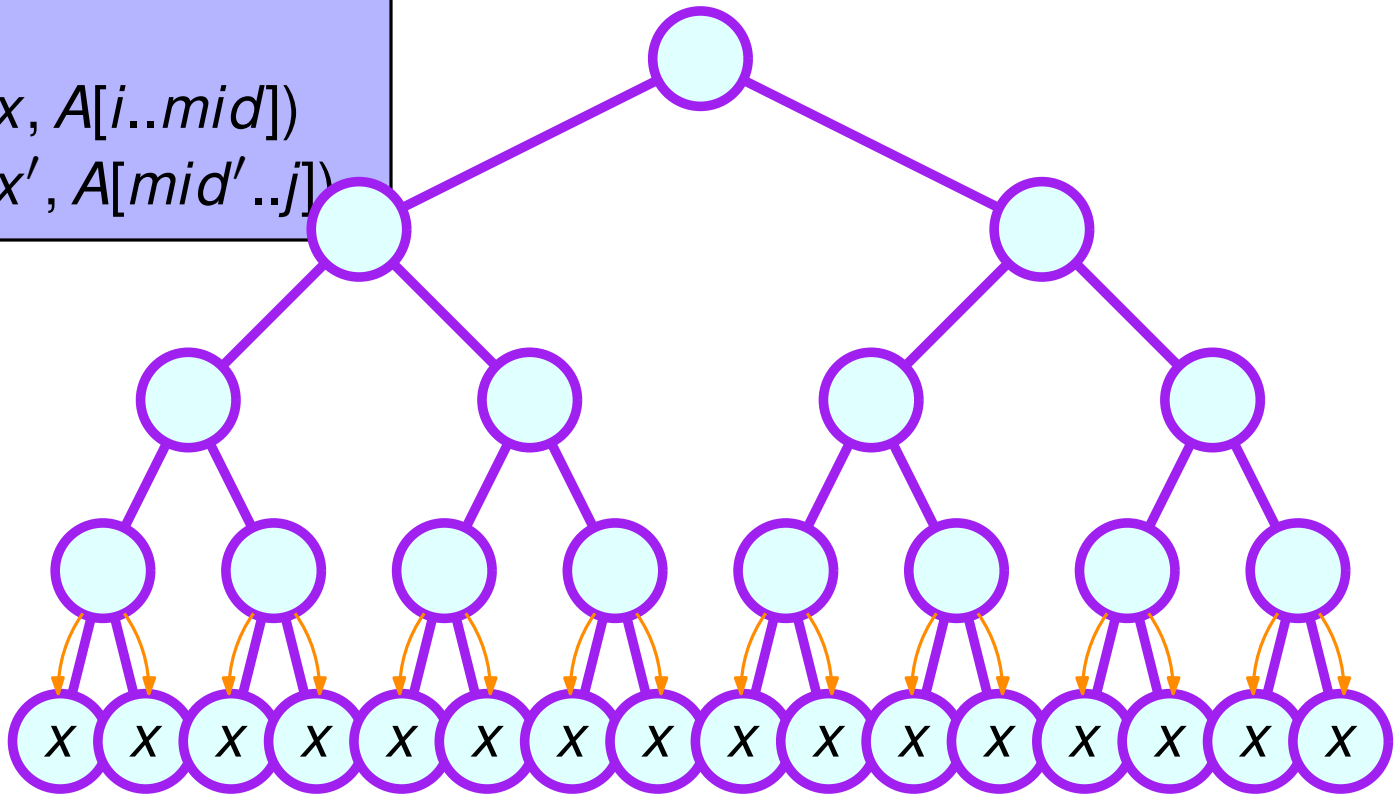
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



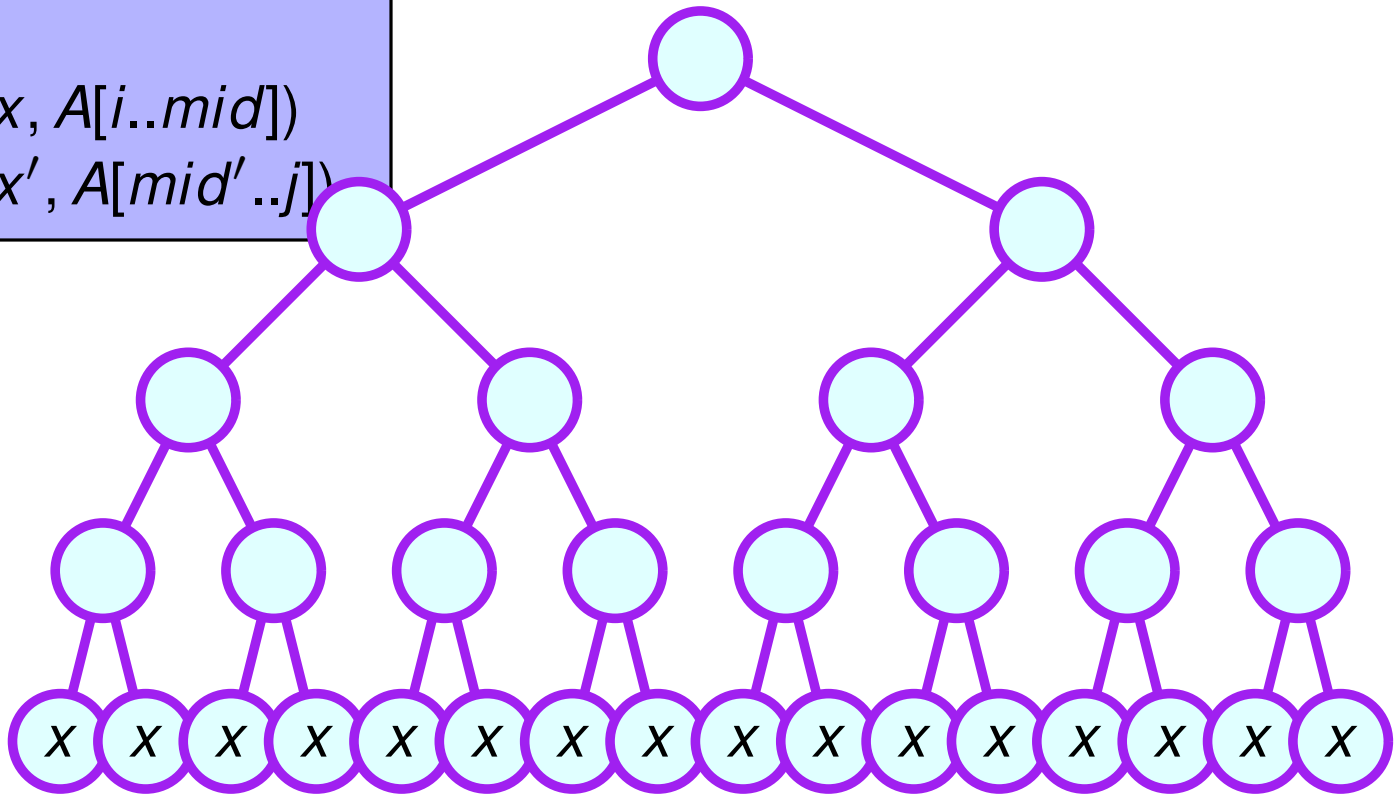
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



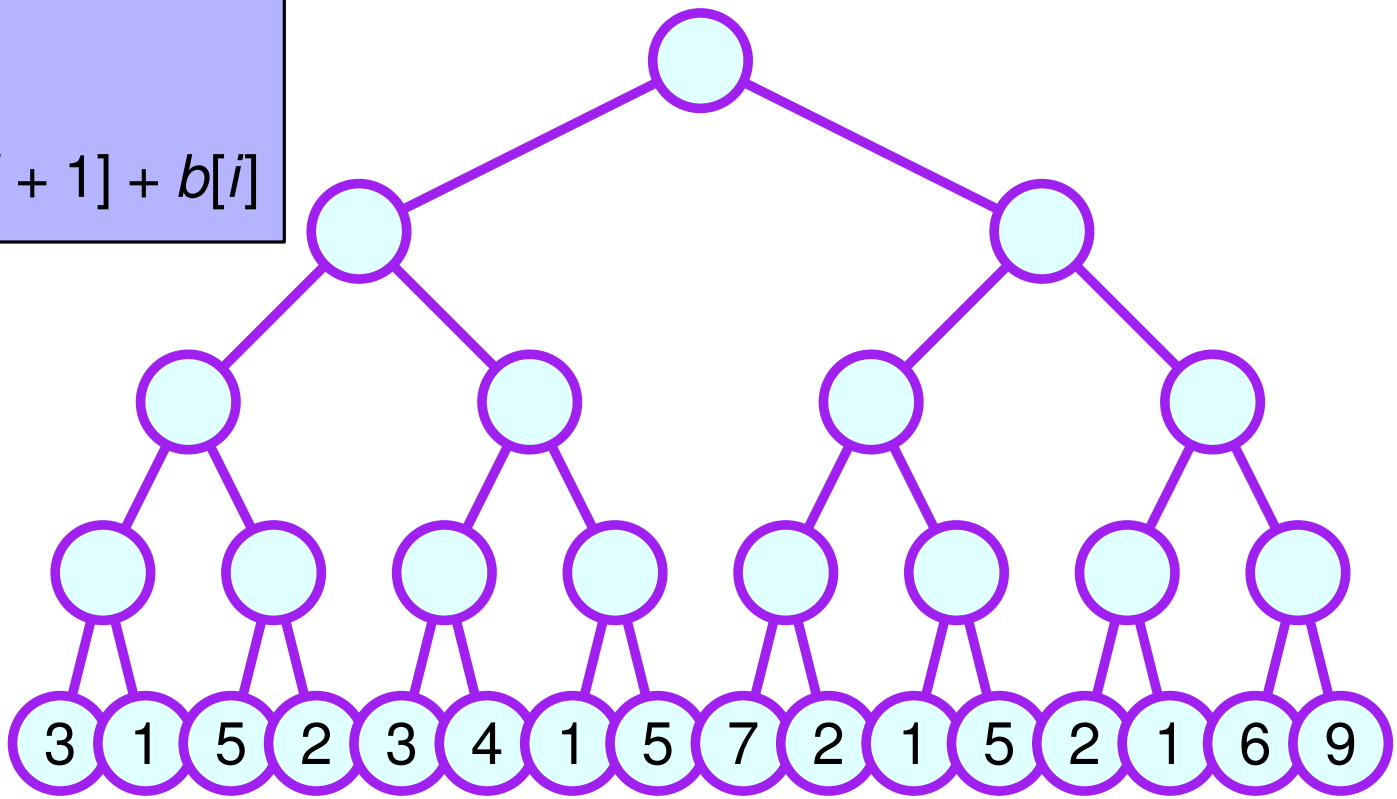
Broadcast visualization

procedure BROADCAST(x, $A[i..j]$)
  if i = j **then**
    $A[i] = x$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $mid' = mid + 1$
    $x' = x$
    in parallel do
      BROADCAST(x, $A[i..mid]$)
      BROADCAST(x', $A[mid'..j]$)



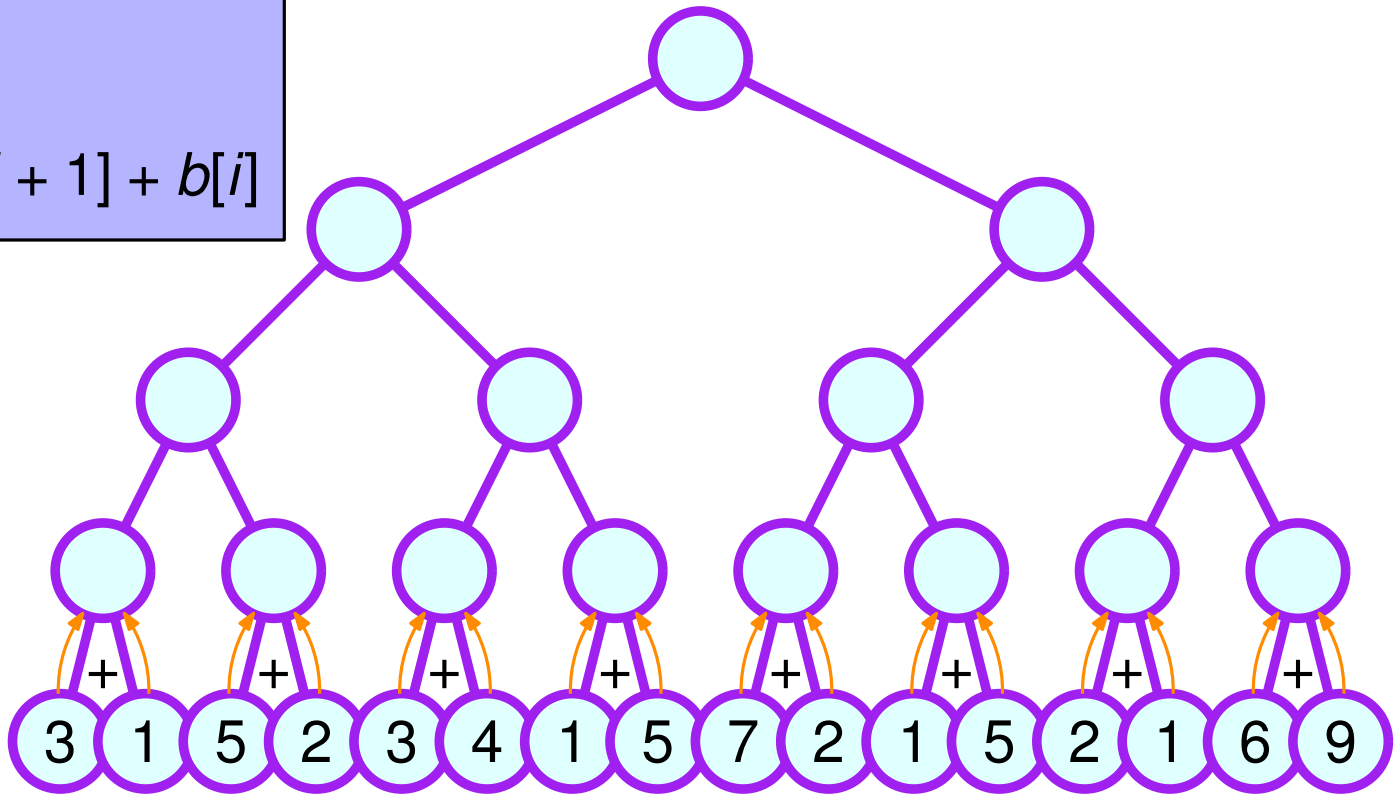
Prefix Sums

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



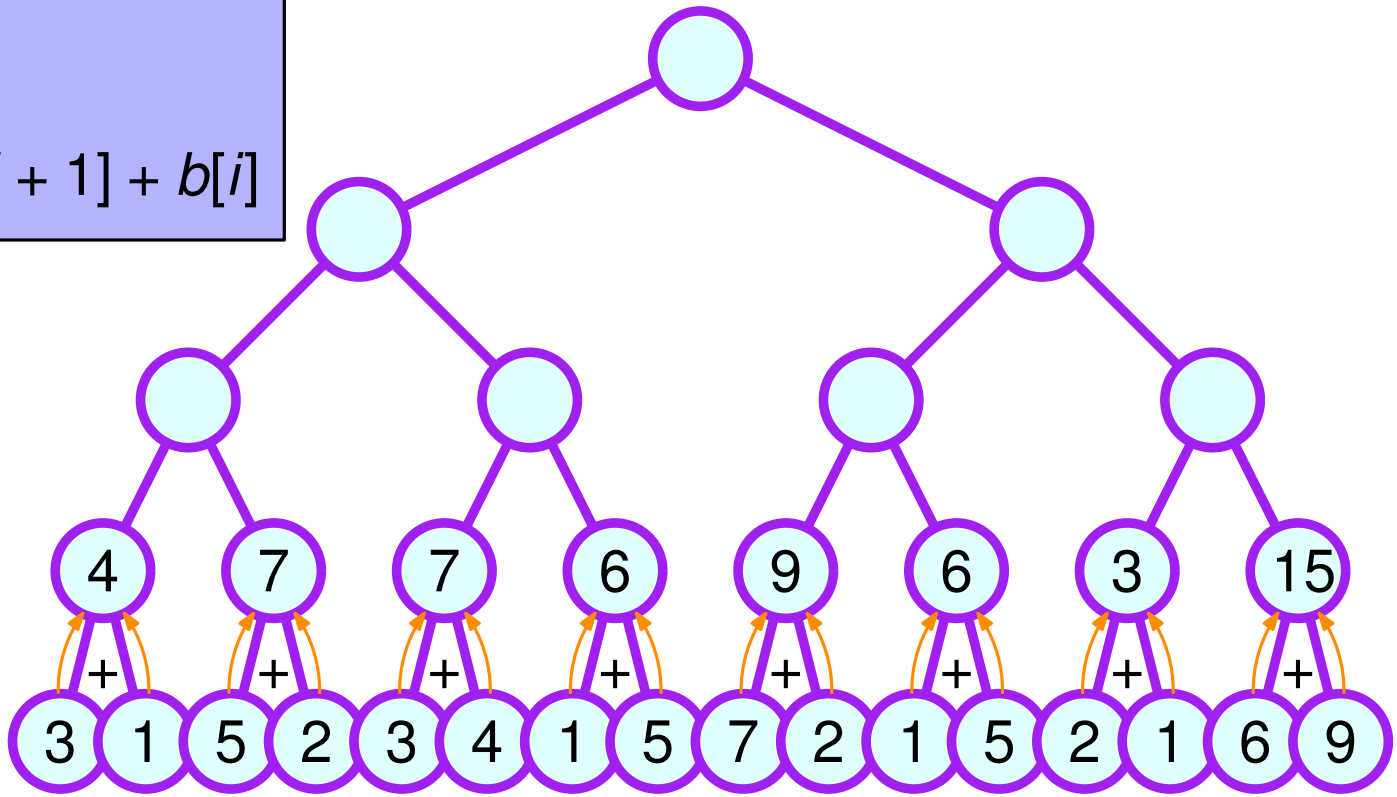
Prefix Sums

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



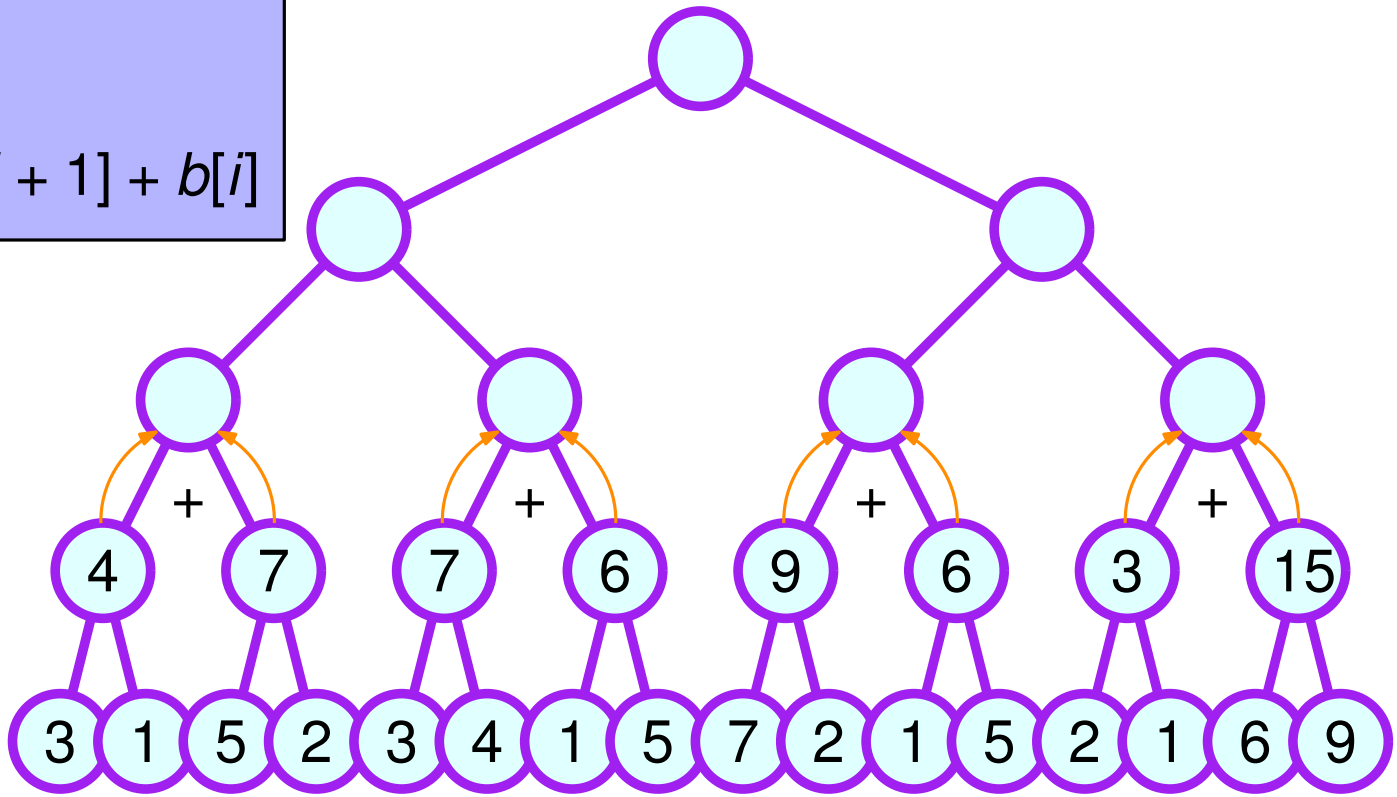
Prefix Sums

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



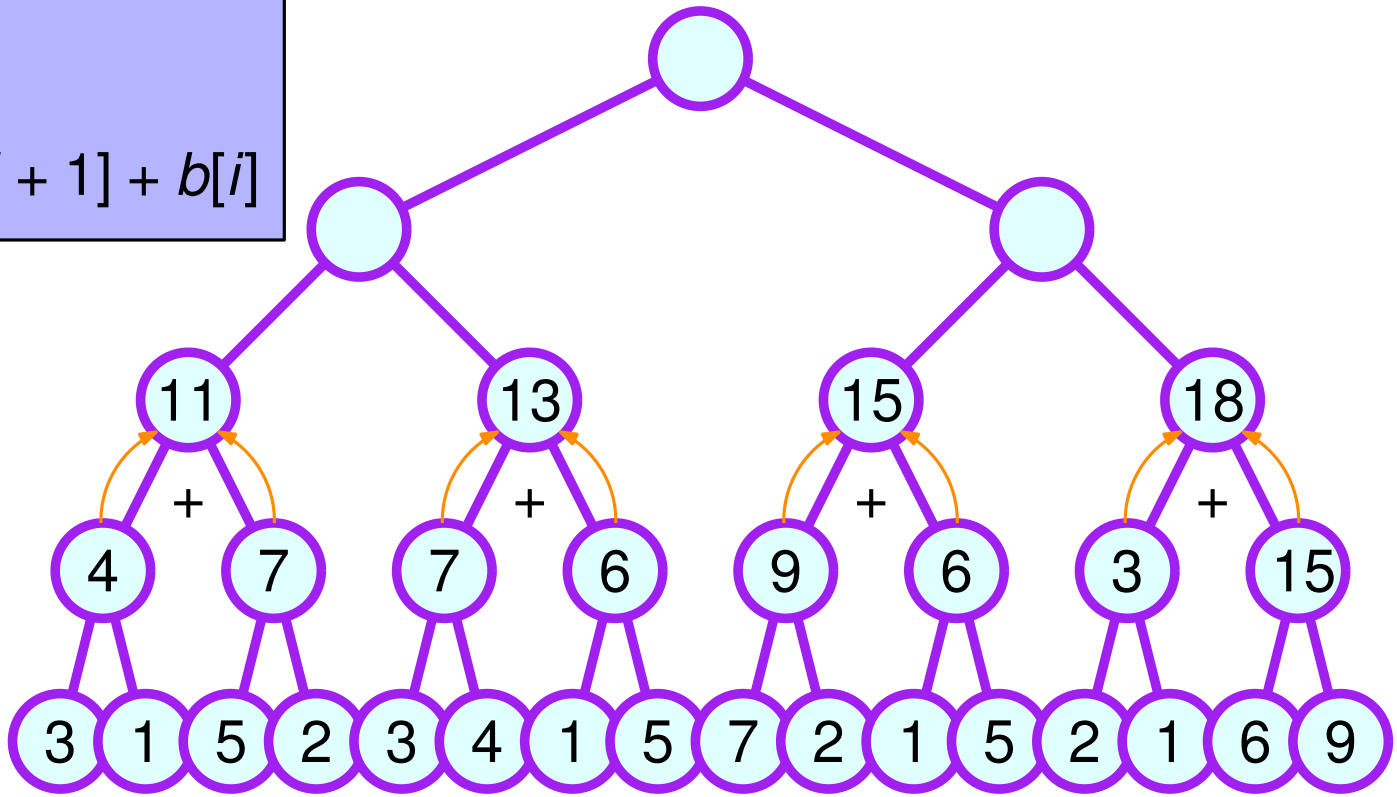
Prefix Sums

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



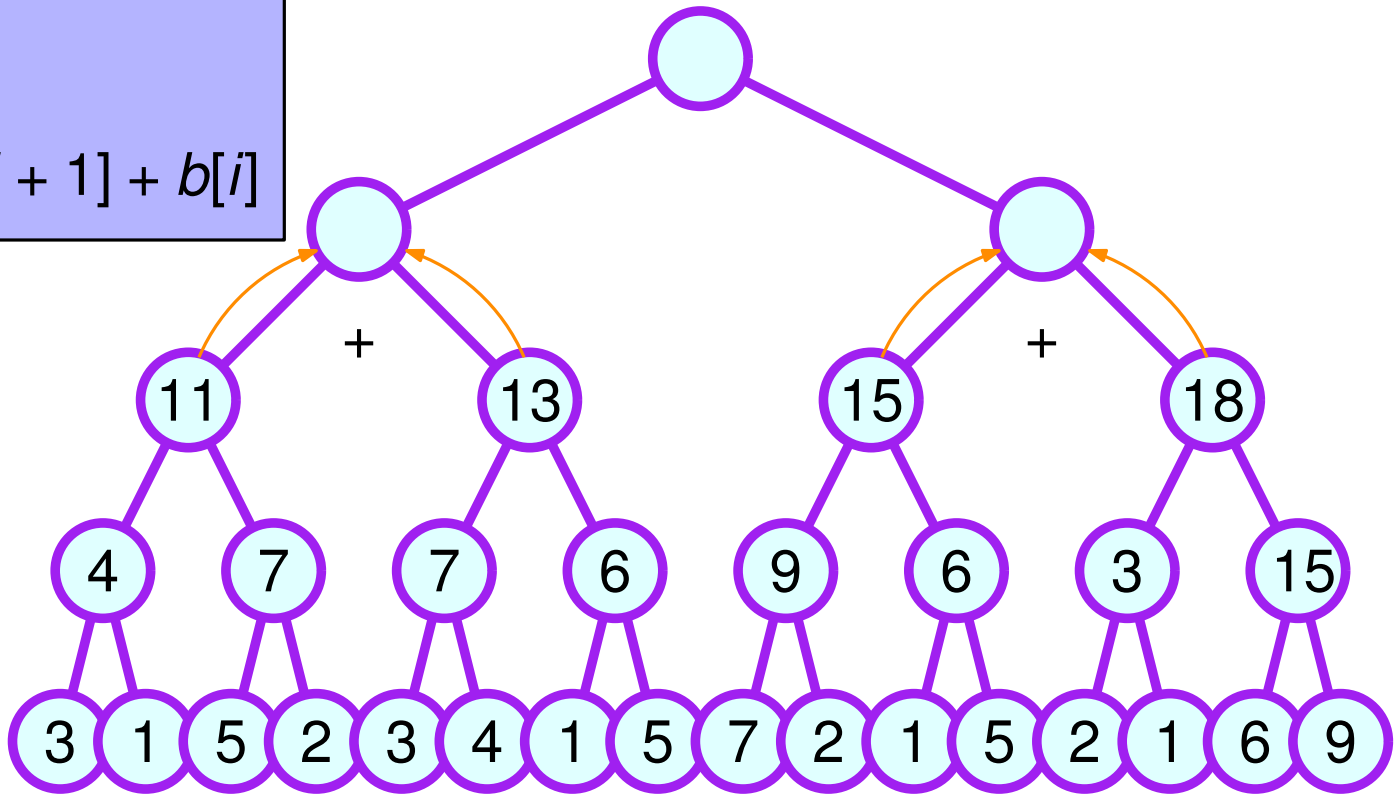
Prefix Sums

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



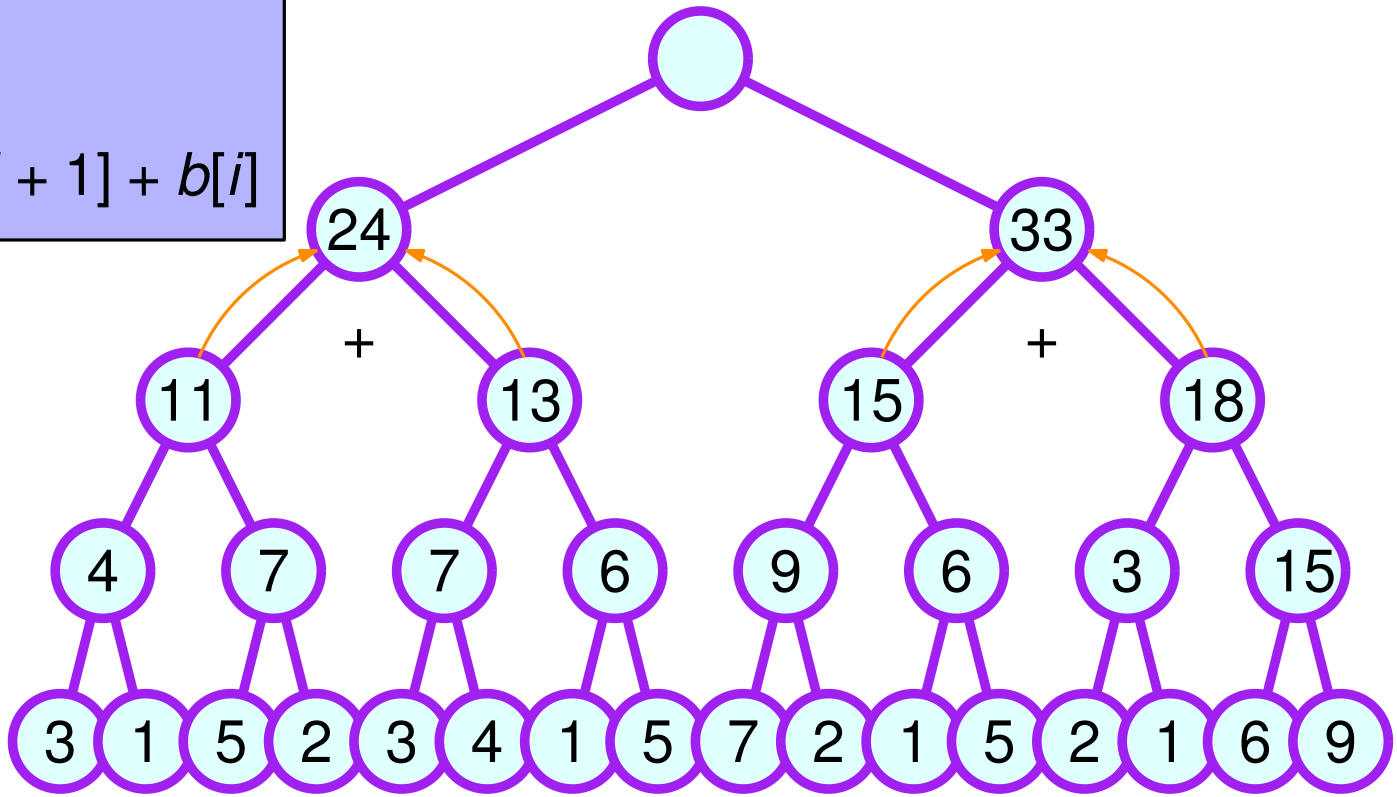
Prefix Sums

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



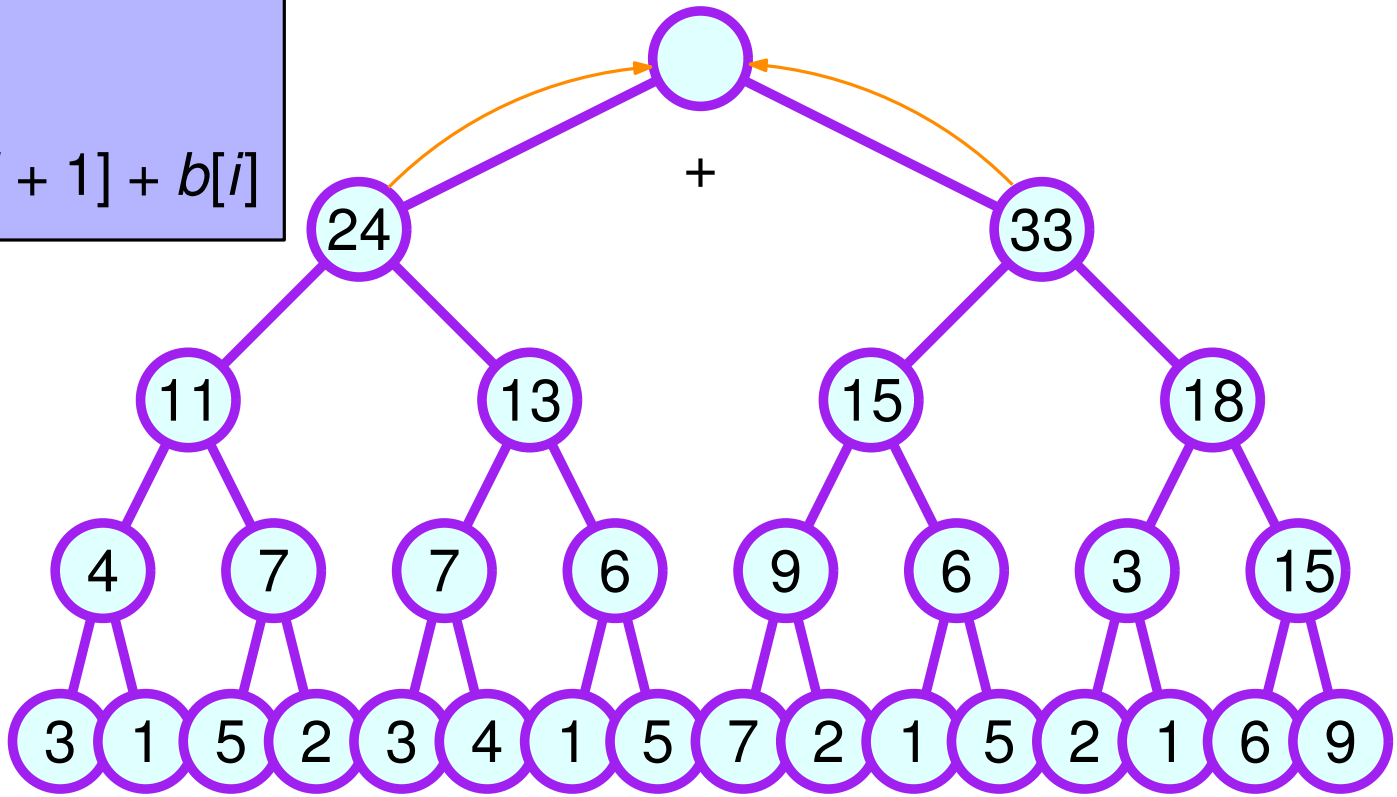
Prefix Sums

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



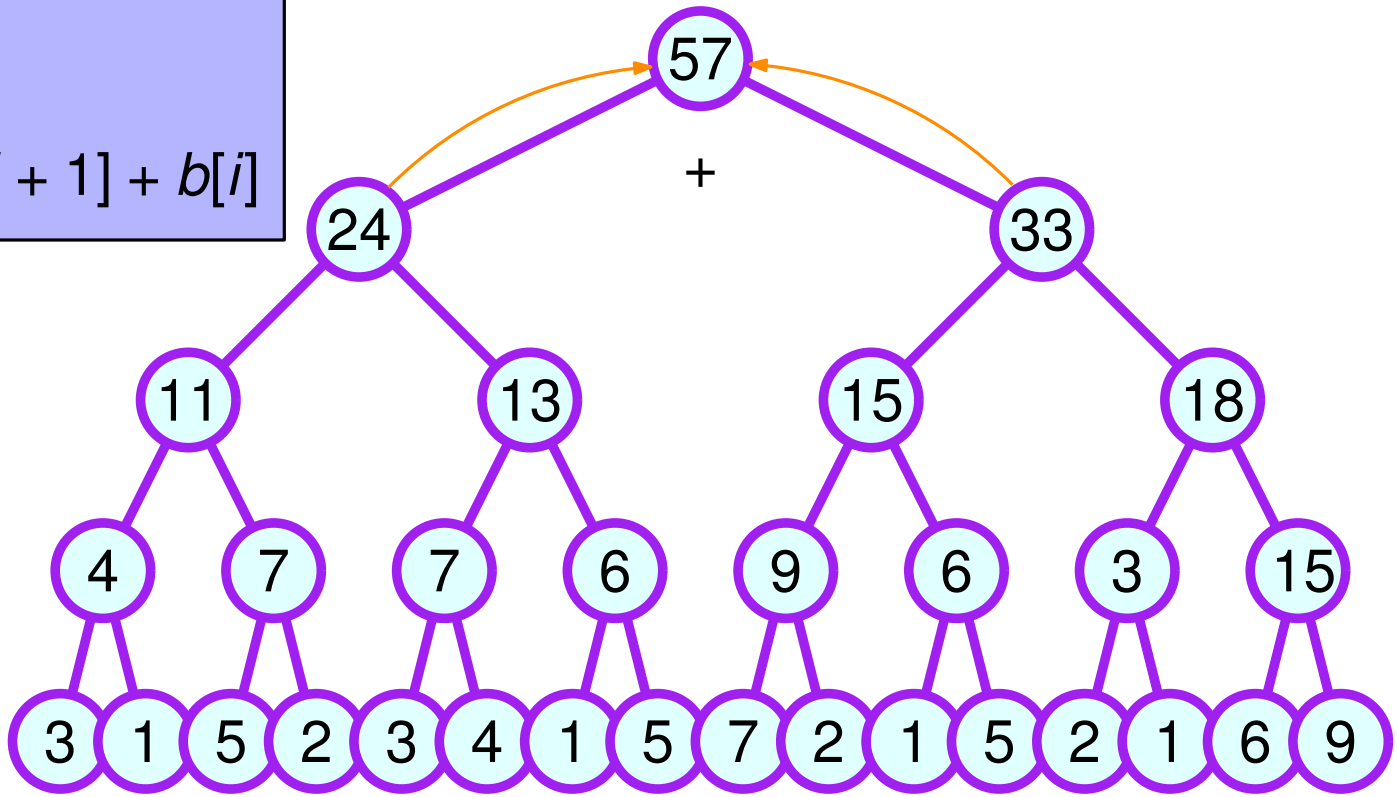
Prefix Sums

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



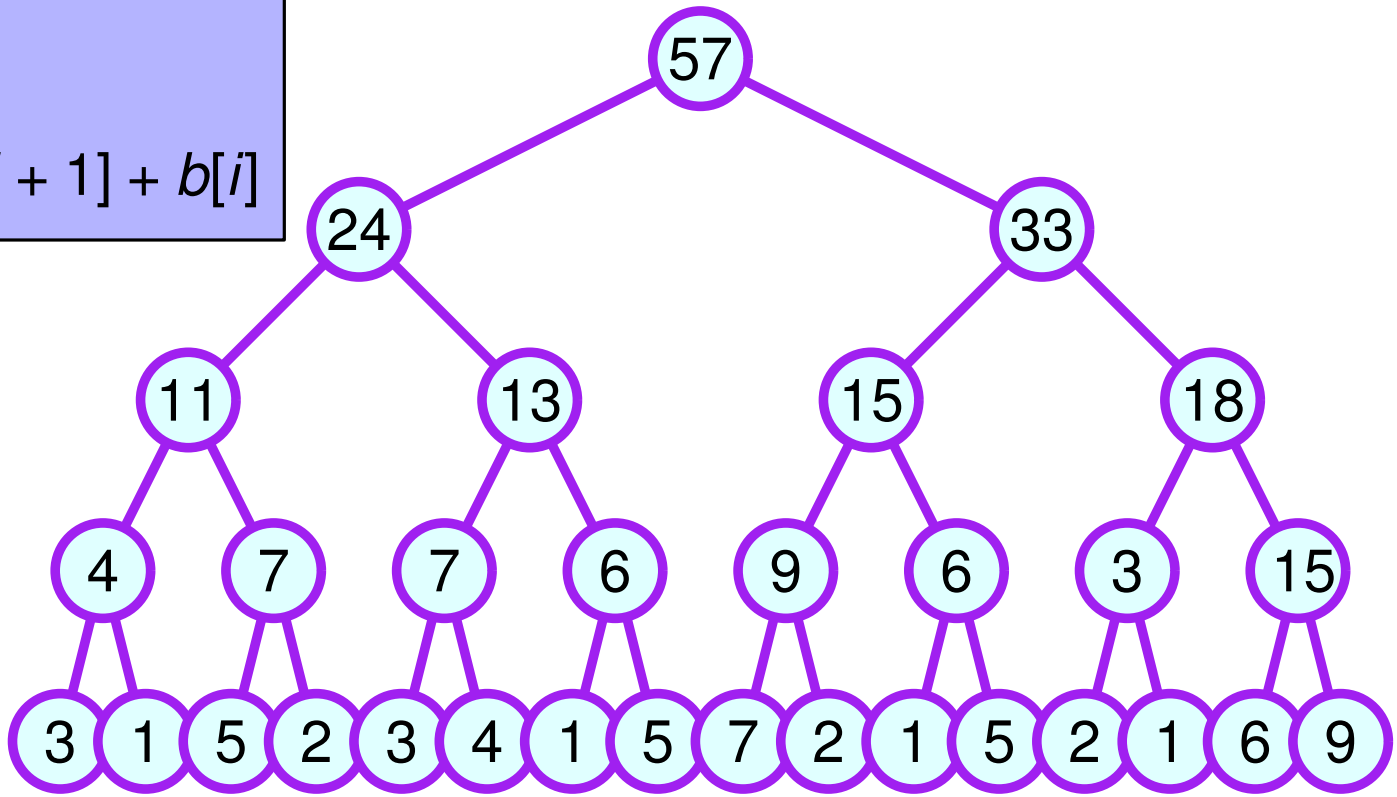
Prefix Sums

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



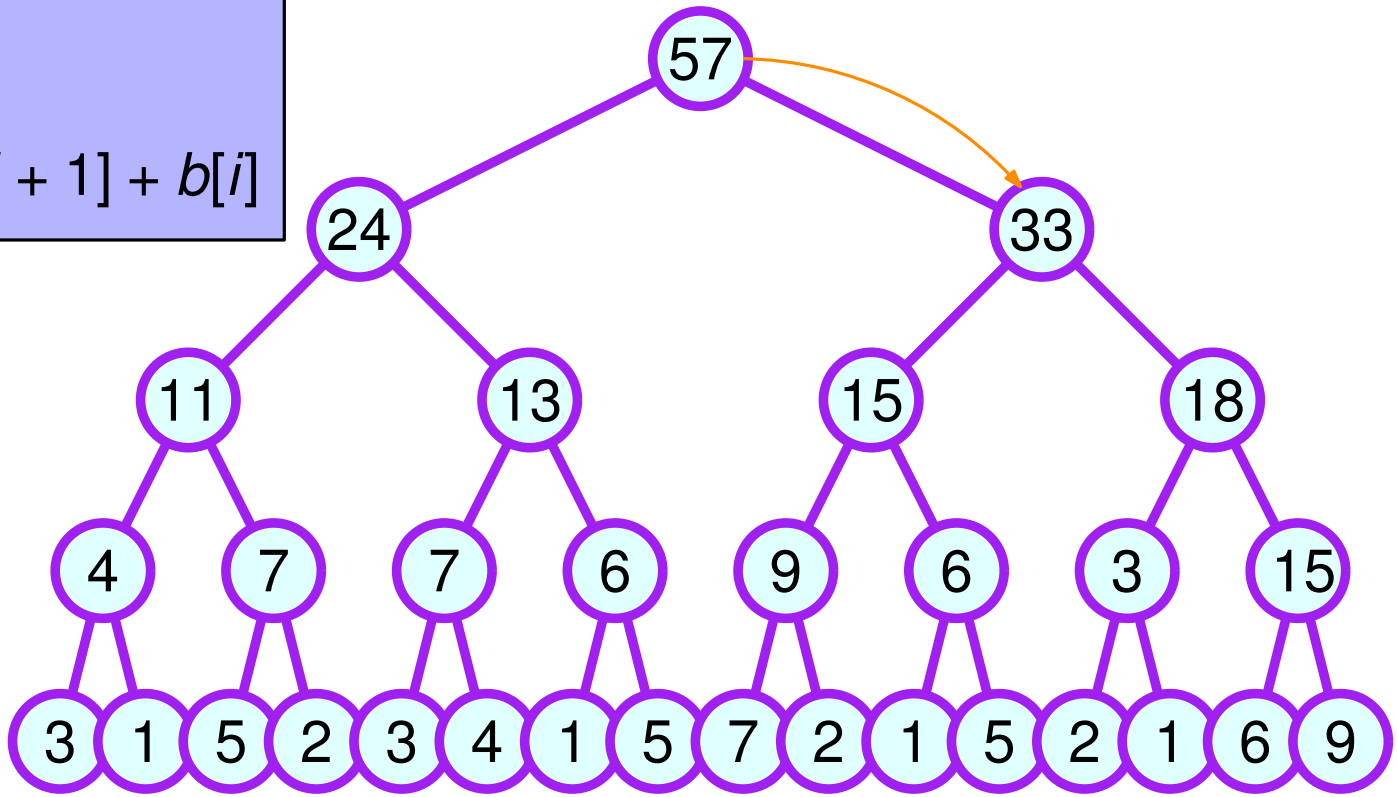
Prefix Sums

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



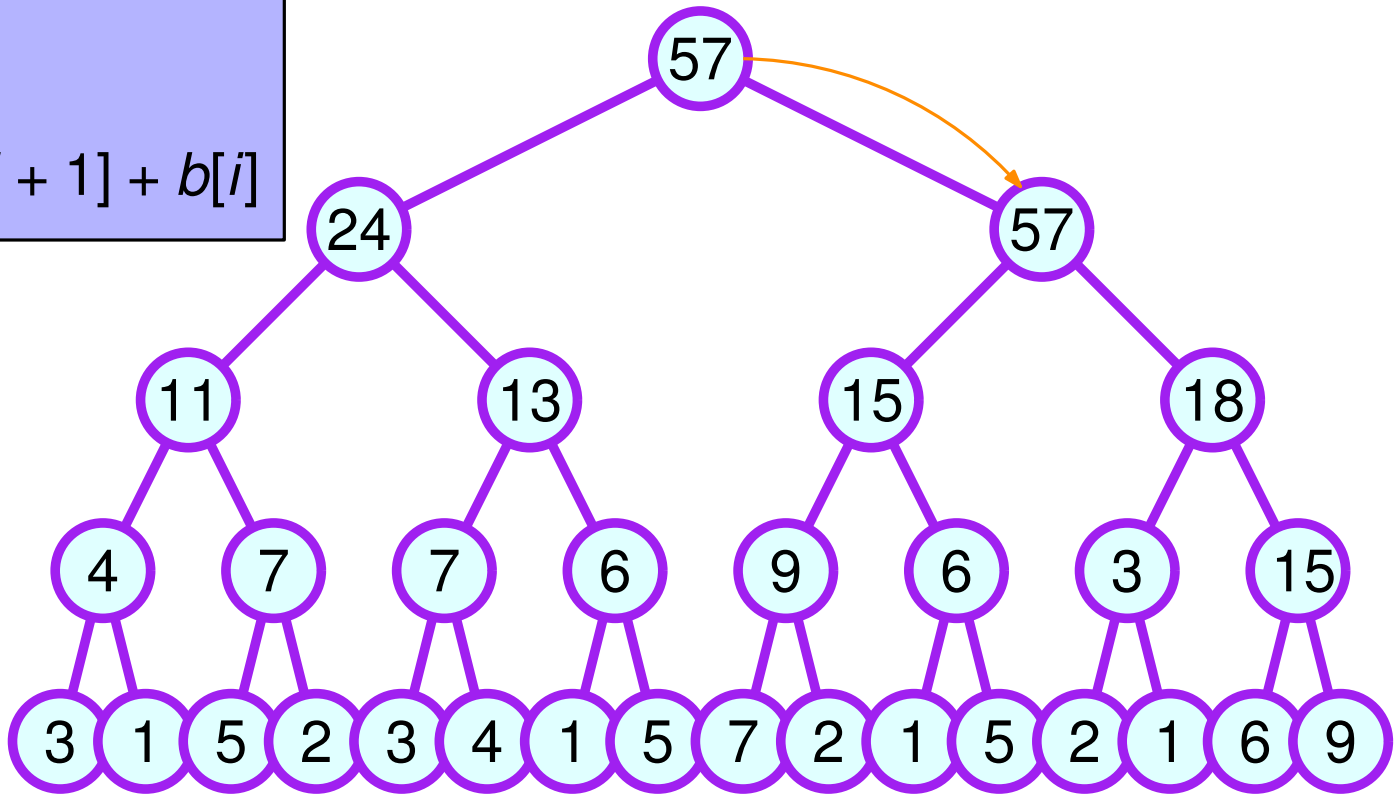
Prefix Sums

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



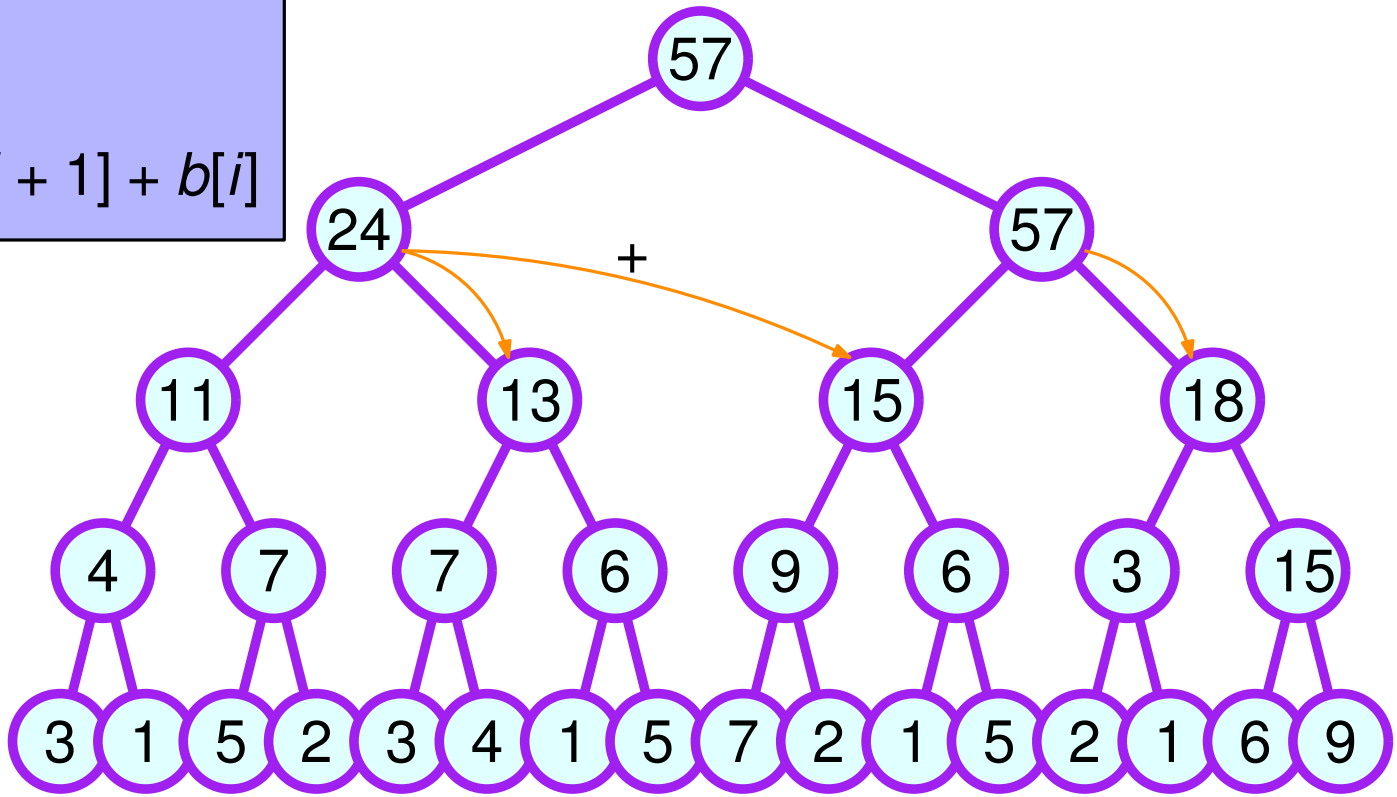
Prefix Sums

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



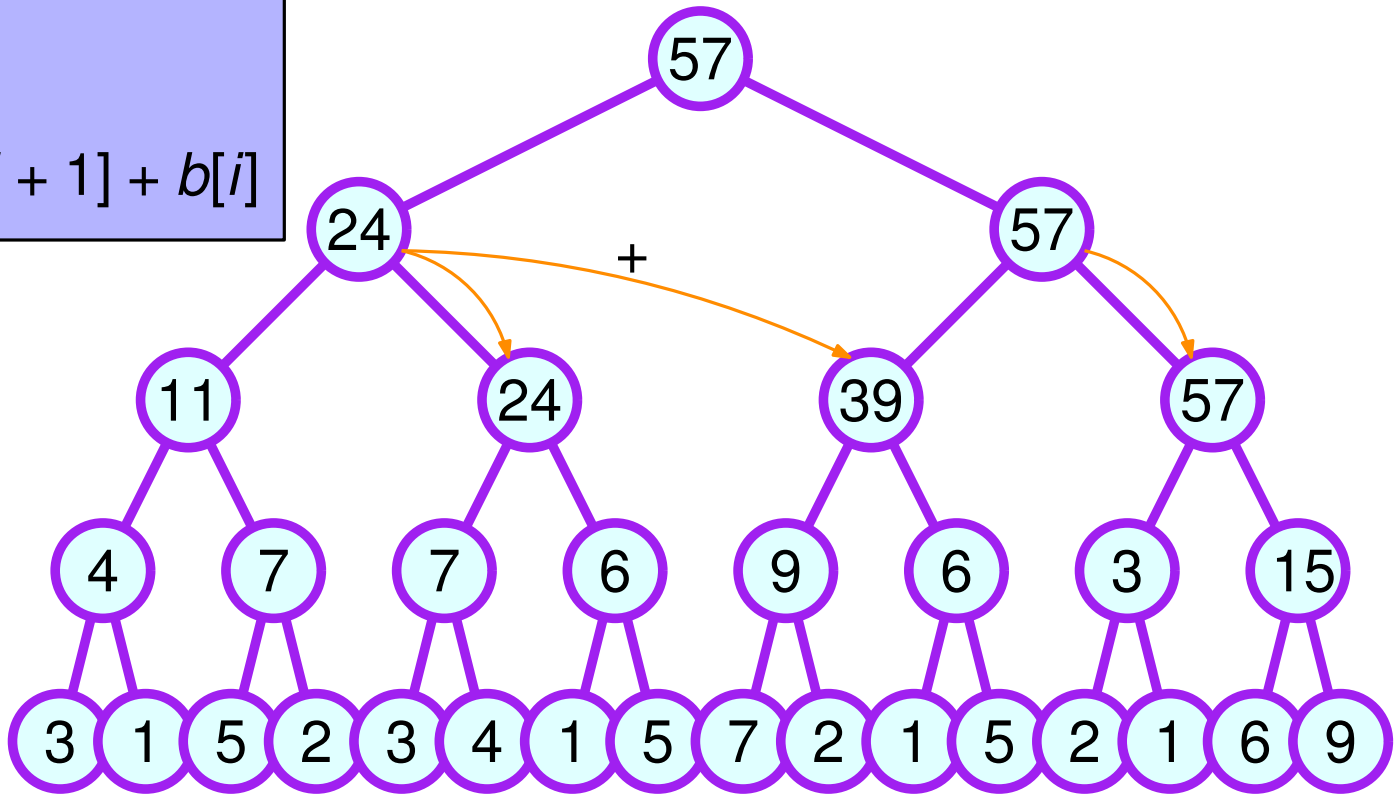
Prefix Sums

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



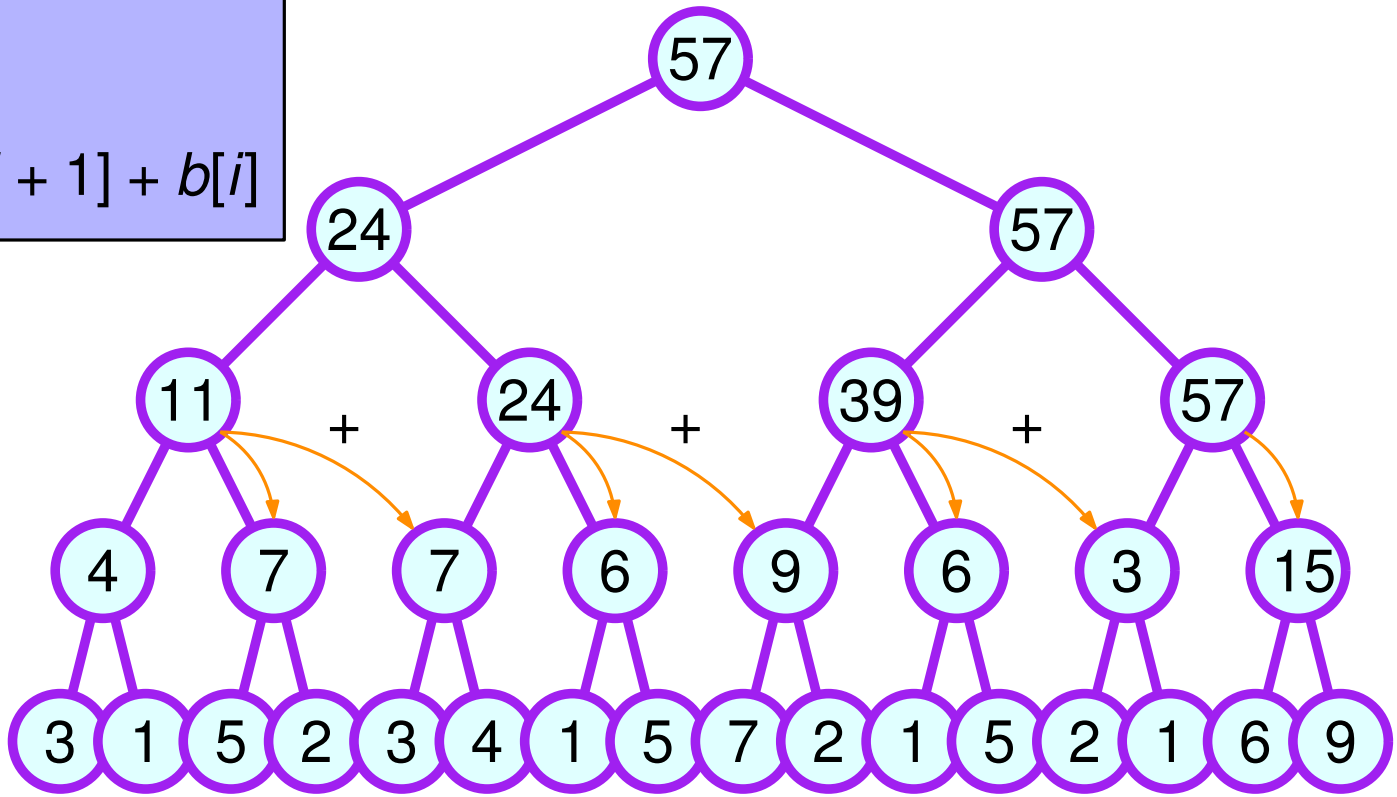
Prefix Sums

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



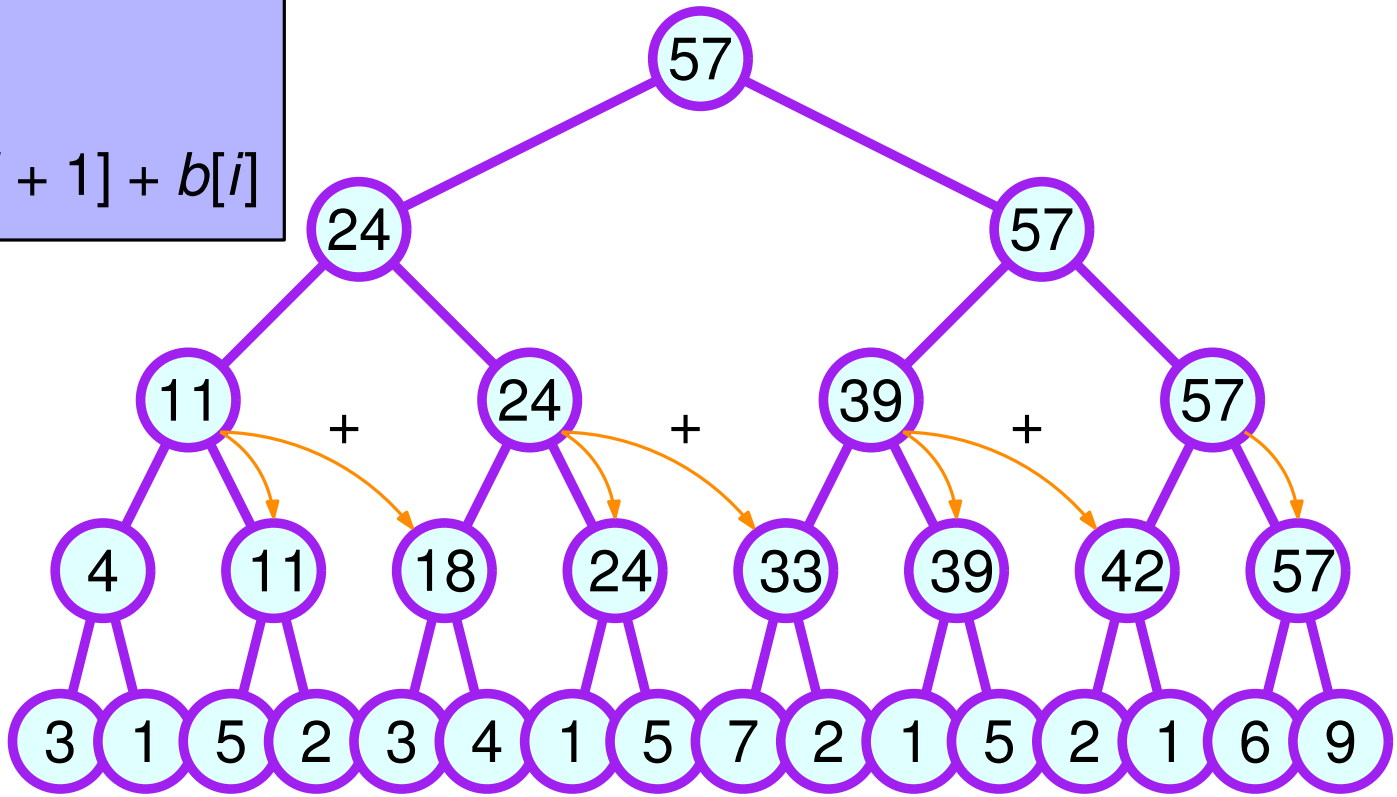
Prefix Sums

```
procedure PREFIX-SUMS(a[1..n])
    if n ≤ 1 then return
    for i = 1 to n/2 in parallel do
        b[i] = a[2i − 1] + a[2i]
    PREFIX-SUMS(b[1..n/2])
    for i = 1 to n/2 in parallel do
        a[2i] = b[i]
        if i ≠ n/2 then
            a[2i + 1] = a[2i + 1] + b[i]
```



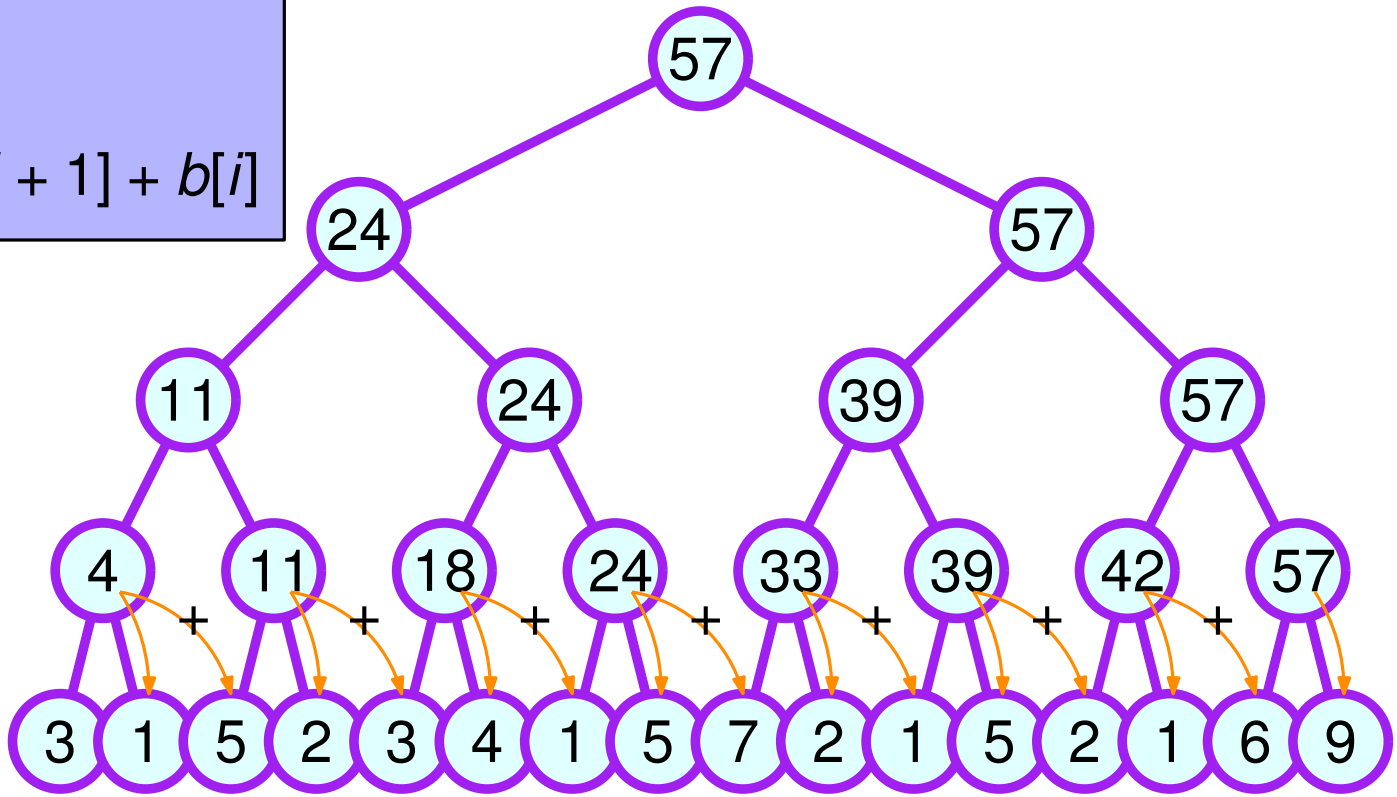
Prefix Sums

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



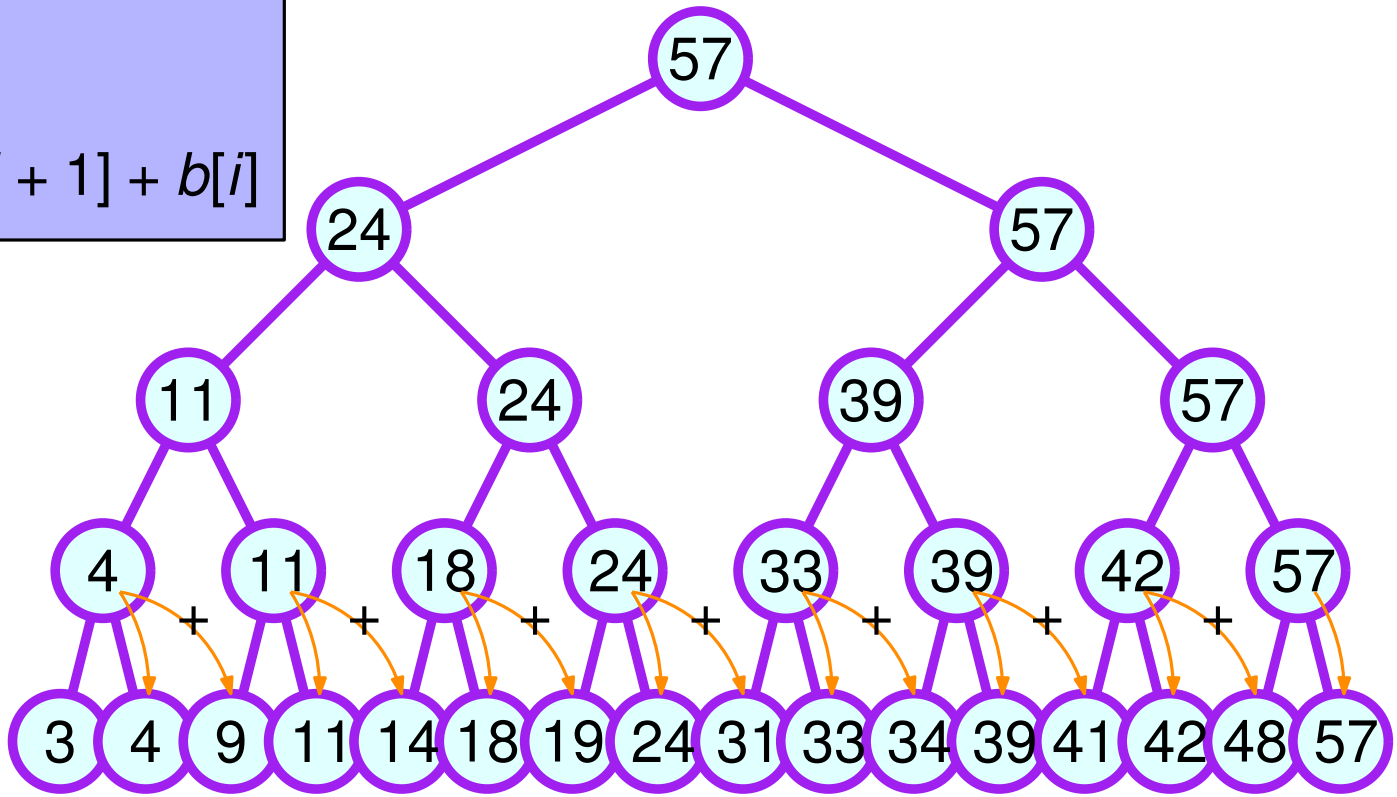
Prefix Sums

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



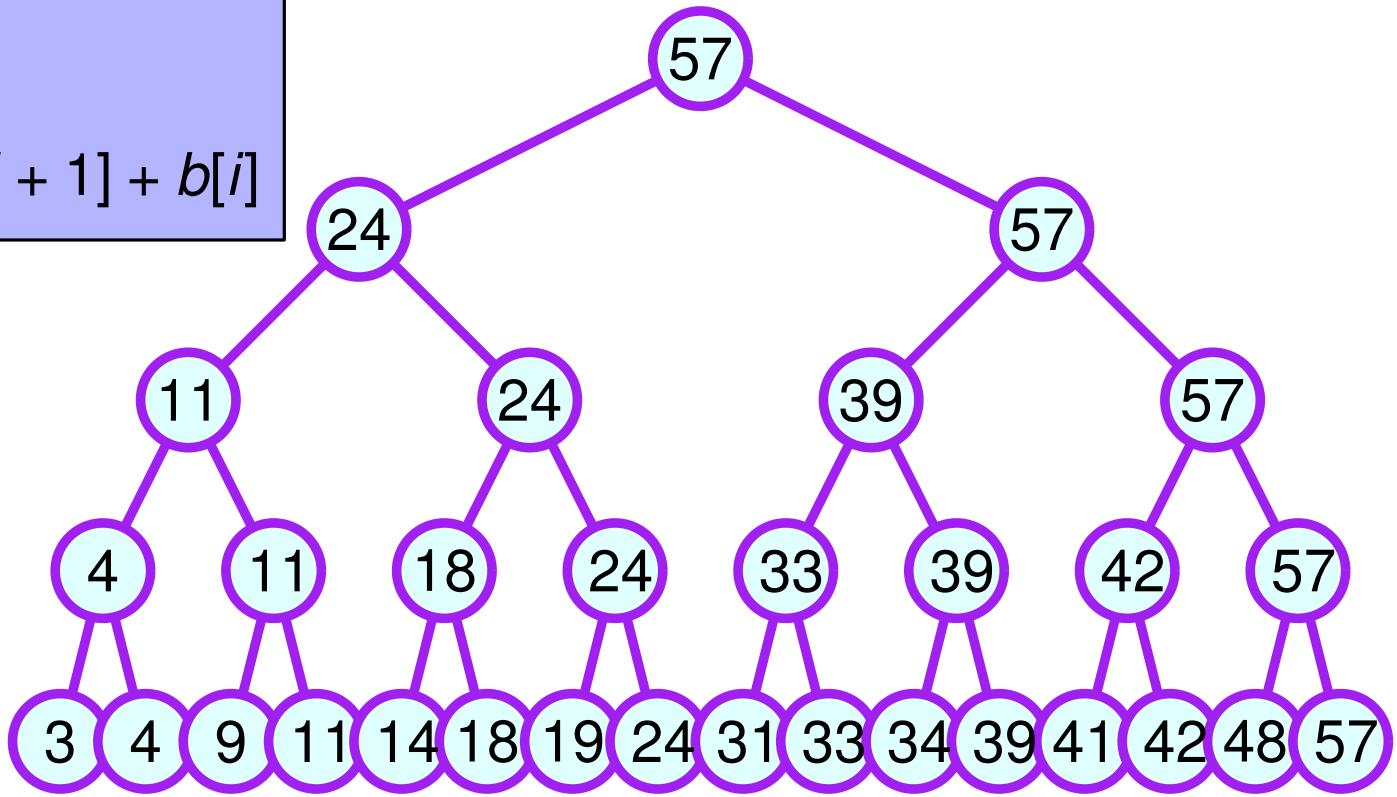
Prefix Sums

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



Prefix Sums

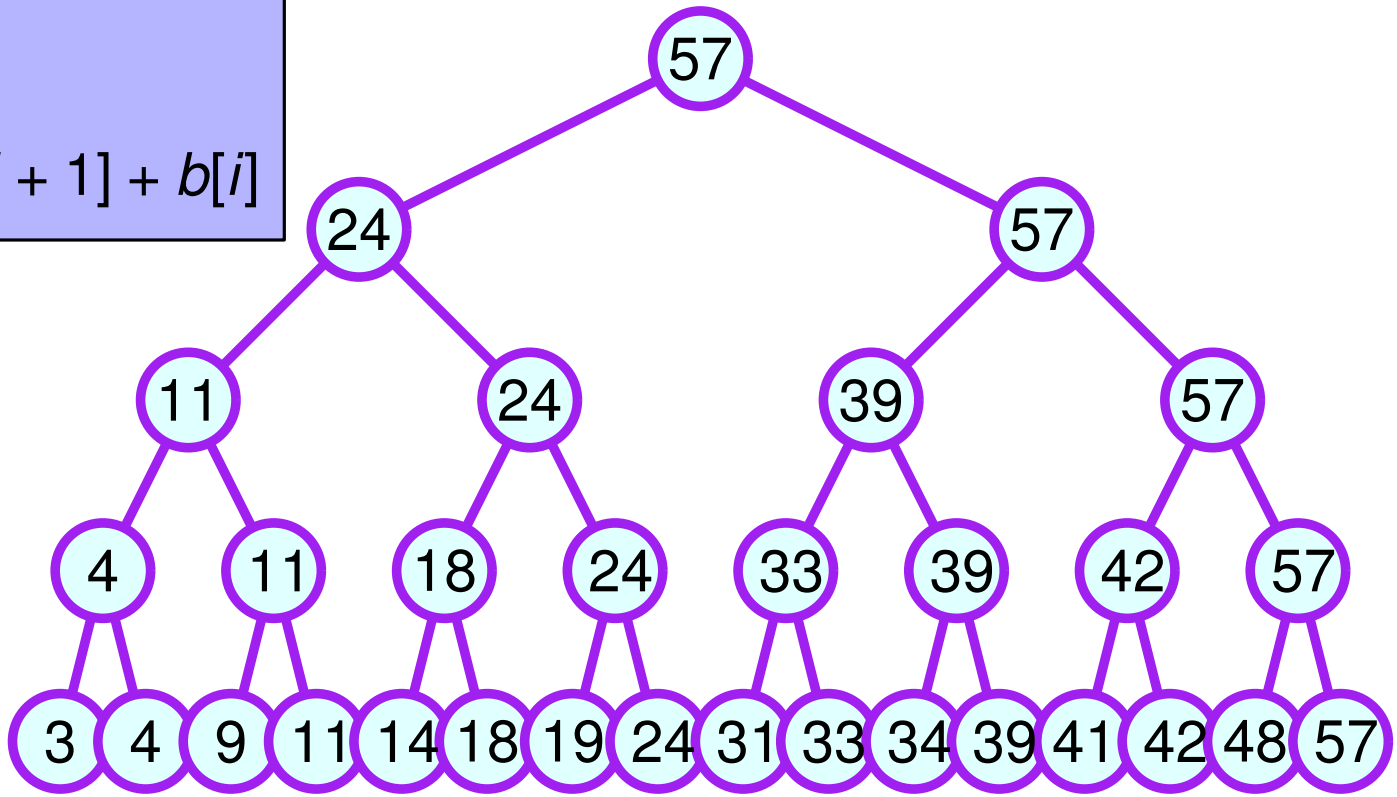
procedure PREFIX-SUMS($a[1..n]$)
  if $n \leq 1$ **then return**
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i - 1] + a[2i]$
  PREFIX-SUMS($b[1..\frac{n}{2}]$)
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i +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



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

Works with any ***associative*** operation



Prefix Sums

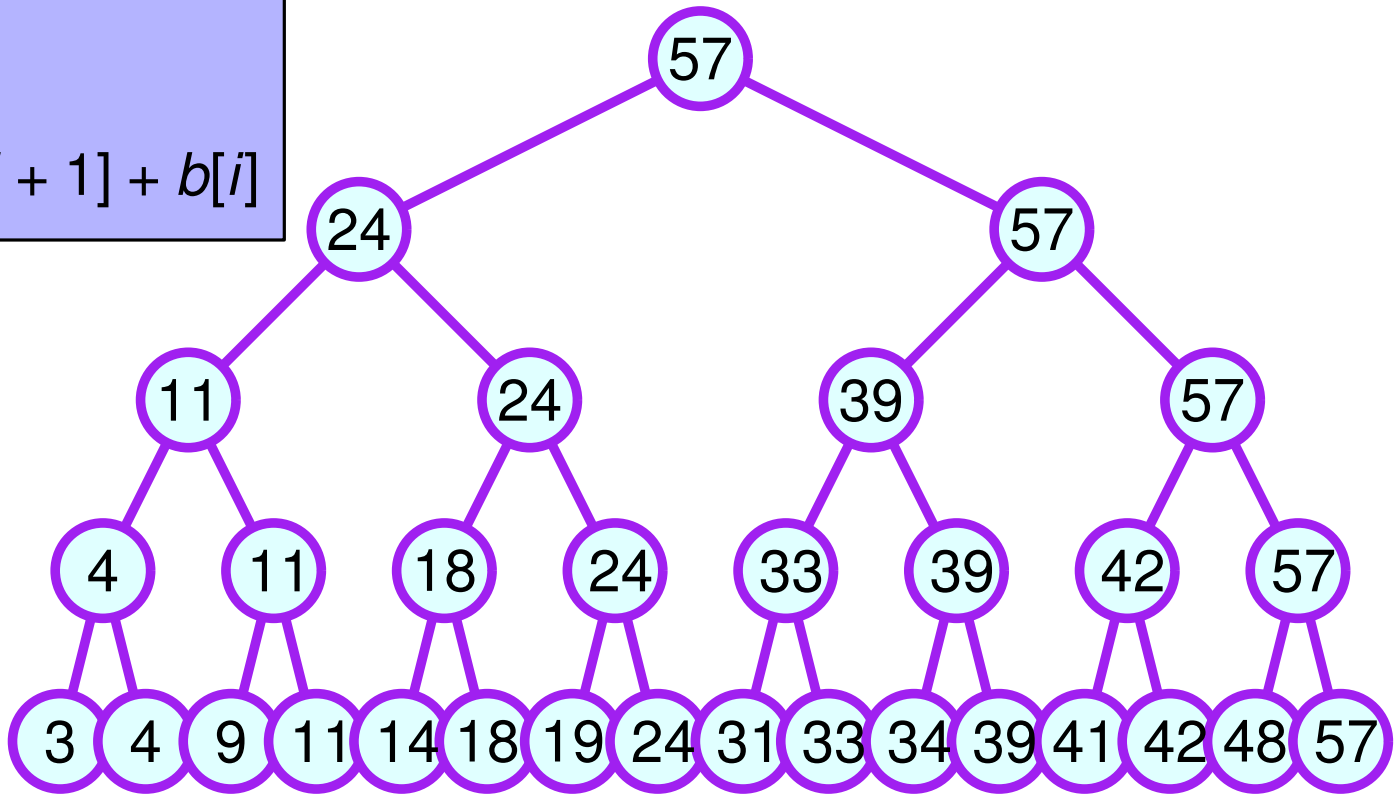
procedure PREFIX-SUMS($a[1..n]$)
  if $n \leq 1$ **then return**
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $b[i] = a[2i - 1] + a[2i]$
  PREFIX-SUMS($b[1..\frac{n}{2}]$)
  for $i = 1$ to $\frac{n}{2}$ **in parallel do**
    $a[2i] = b[i]$
    if $i \neq \frac{n}{2}$ **then**
      $a[2i +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

Works with any ***associative*** operation

$$(a + b) + c = a + (b + c)$$

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

$$\min(\min(a, b), c) = \min(a, \min(b, c))$$



Prefix Sums

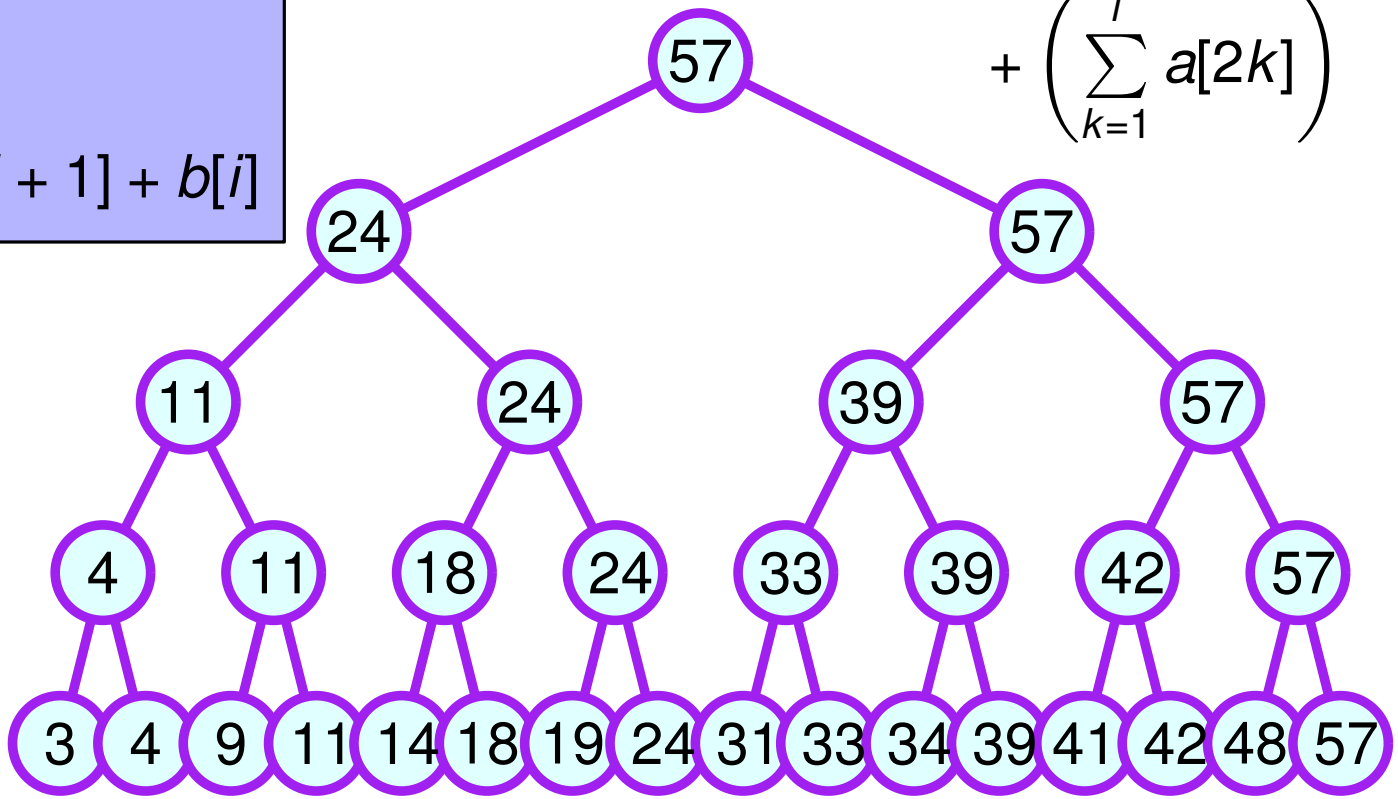
```
procedure PREFIX-SUMS(a[1..n])
    if n ≤ 1 then return
    for i = 1 to n/2 in parallel do
        b[i] = a[2i − 1] + a[2i]
    PREFIX-SUMS(b[1..n/2])
    for i = 1 to n/2 in parallel do
        a[2i] = b[i]
        if i ≠ n/2 then
            a[2i + 1] = a[2i + 1] + b[i]
```

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

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

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

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



Broadcast via Prefix Sums

"sum" is a binary operator $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- takes two real arguments, e.g., *a* and *b*
- returns a real number equal to the sum of *a* and *b*

Broadcast via Prefix Sums

"sum" is a binary operator $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- takes two real arguments, e.g., a and b
- returns a real number equal to the sum of a and b

$$a + b = sum(a, b) = +(a, b)$$

infix notation

prefix notation

Broadcast via Prefix Sums

"sum" is a binary operator $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- takes two real arguments, e.g., a and b
- returns a real number equal to the sum of a and b

$$a + b = sum(a, b) = +(a, b)$$

Similary, define a binary operator $\text{COPY} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- takes two real arguments, e.g., a and b
- returns a real number, the first one of the two arguments

Broadcast via Prefix Sums

"sum" is a binary operator $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- takes two real arguments, e.g., a and b
- returns a real number equal to the sum of a and b

$$a + b = sum(a, b) = +(a, b)$$

Similary, define a binary operator $\text{COPY} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- takes two real arguments, e.g., a and b
- returns a real number, the first one of the two arguments

E.g.:

- $\text{COPY}(7, 2)$ returns 7
- $\text{COPY}(12, 55)$ returns 12
- $\text{COPY}(a, b)$ returns a

Broadcast via Prefix Sums

"sum" is a binary operator $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- takes two real arguments, e.g., a and b
- returns a real number equal to the sum of a and b

$$a + b = sum(a, b) = +(a, b)$$

Similary, define a binary operator $\text{COPY} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

- takes two real arguments, e.g., a and b
- returns a real number, the first one of the two arguments

E.g.:

- $\text{COPY}(7, 2)$ returns 7
- $\text{COPY}(12, 55)$ returns 12
- $\text{COPY}(a, b)$ returns a

Claim. COPY *operator is associative, i.e.,*

$$\text{COPY}(a, \text{COPY}(b, c)) = \text{COPY}(\text{COPY}(a, b), c)$$

COPY is associative

Claim. COPY *operator is associative, i.e.,*

$$\text{COPY}(a, \text{COPY}(b, c)) = \text{COPY}(\text{COPY}(a, b), c)$$

COPY is associative

Claim. COPY *operator is associative, i.e.,*

$$\text{COPY}(a, \text{COPY}(b, c)) = \text{COPY}(\text{COPY}(a, b), c)$$

Proof.

$$\text{COPY}(a, \text{COPY}(b, c)) = \text{COPY}(\text{COPY}(a, b), c)$$

COPY is associative

Claim. COPY *operator is associative, i.e.,*

$$\text{COPY}(a, \text{COPY}(b, c)) = \text{COPY}(\text{COPY}(a, b), c)$$

Proof.

$$\text{COPY}(a, \overbrace{\text{COPY}(b, c)}^{b}) = \text{COPY}(\overbrace{\text{COPY}(a, b)}^{a}, c)$$

COPY is associative

Claim. COPY *operator is associative, i.e.,*

$$\text{COPY}(a, \text{COPY}(b, c)) = \text{COPY}(\text{COPY}(a, b), c)$$

Proof.

$$\overbrace{\text{COPY}(a, \text{COPY}(b, c))}^{b} = \overbrace{\text{COPY}(\text{COPY}(a, b), c)}^{a}$$

$$\Updownarrow \qquad\qquad \Updownarrow$$

$$\text{COPY}(a, b) \qquad\qquad \text{COPY}(a, c)$$

COPY is associative

Claim. COPY *operator is associative, i.e.,*

$$\text{COPY}(a, \text{COPY}(b, c)) = \text{COPY}(\text{COPY}(a, b), c)$$

Proof.

$$\overbrace{\text{COPY}(a, \text{COPY}(b, c))}^{b} = \overbrace{\text{COPY}(\text{COPY}(a, b), c)}^{a}$$

$$\Updownarrow \qquad\qquad \Updownarrow$$

$$\text{COPY}(a, b) \qquad\qquad \text{COPY}(a, c)$$

$$\Updownarrow \qquad\qquad \Updownarrow$$

$$a \qquad\qquad a$$

Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator

Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator

A :

5	12	10	25	16	4	19	8

PREFIX-SUMS($A[1..n]$)

PREFIX-COPY($A[1..n]$)

Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator

A :

5	12	10	25	16	4	19	8

PREFIX-SUMS($A[1..n]$)

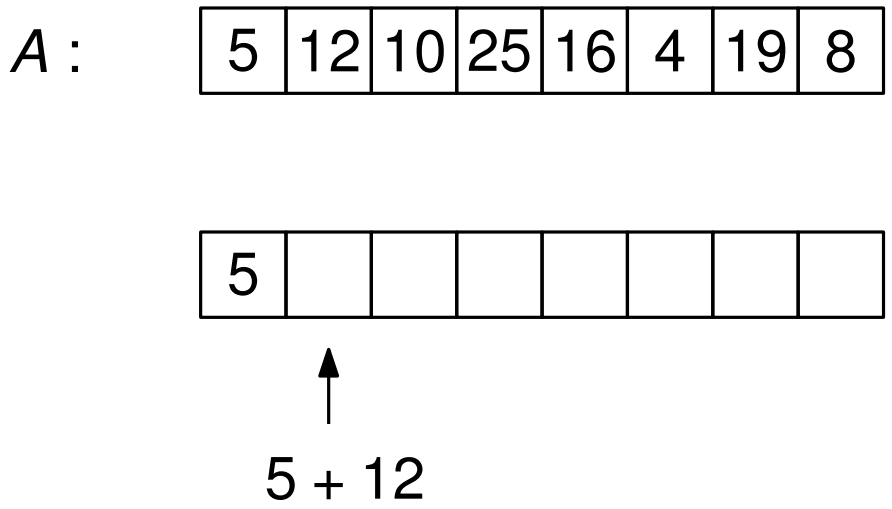
5							

PREFIX-COPY($A[1..n]$)

5							

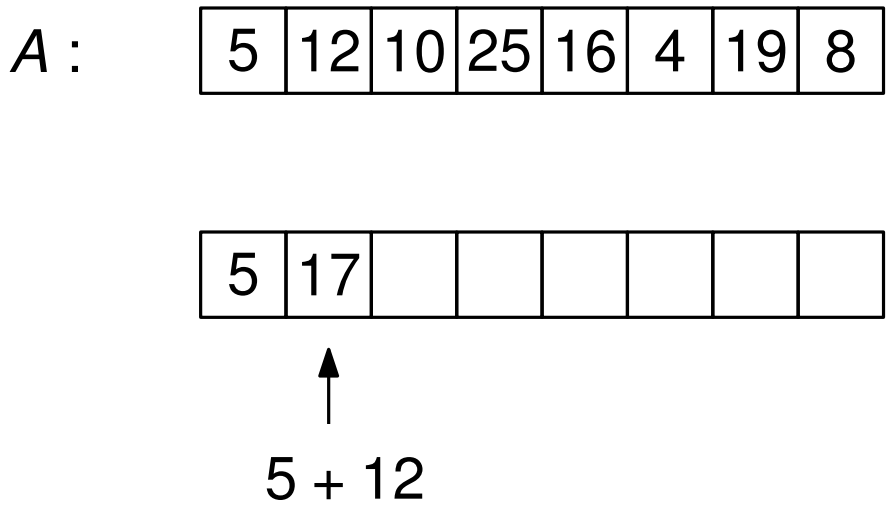
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



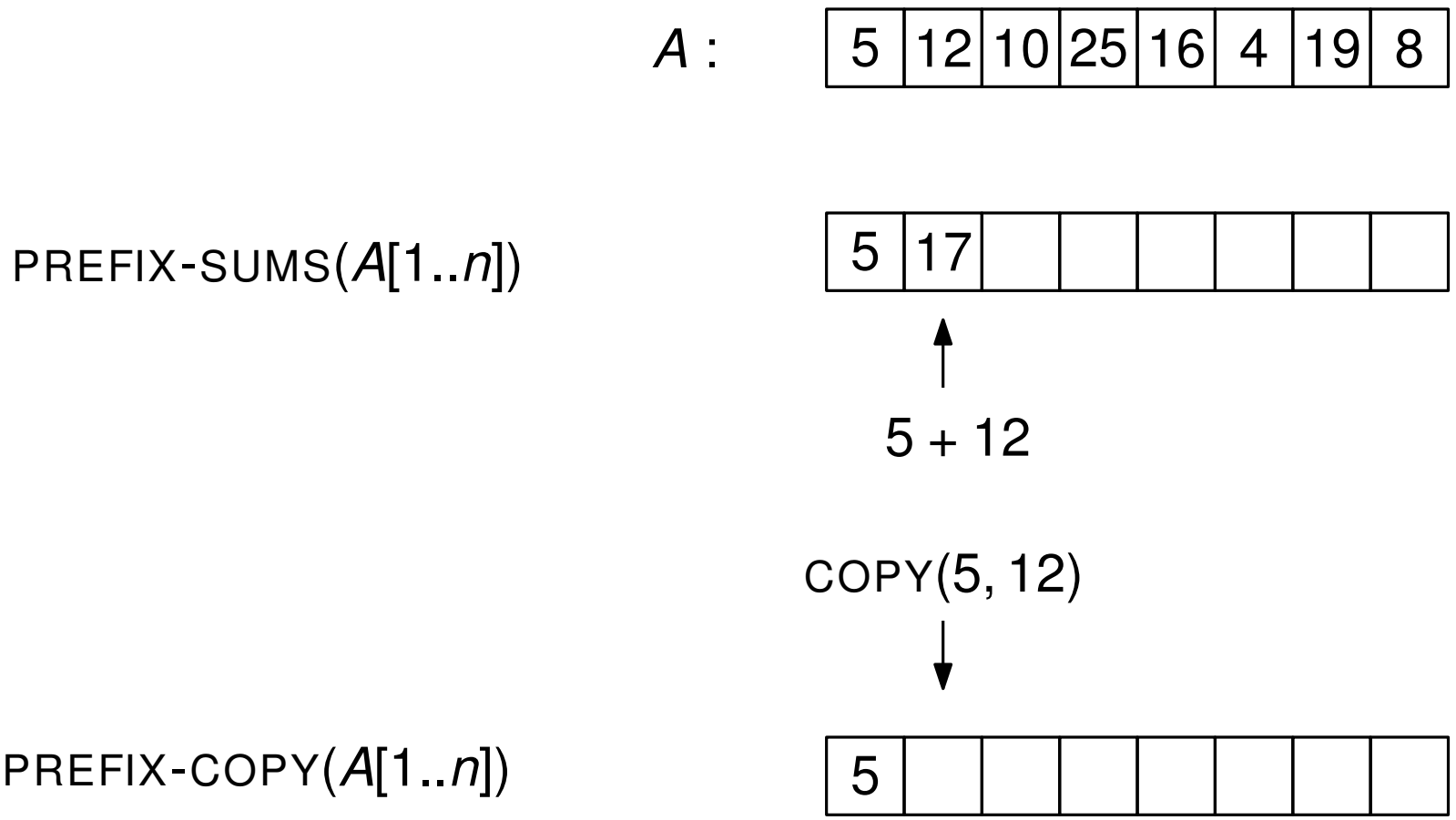
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



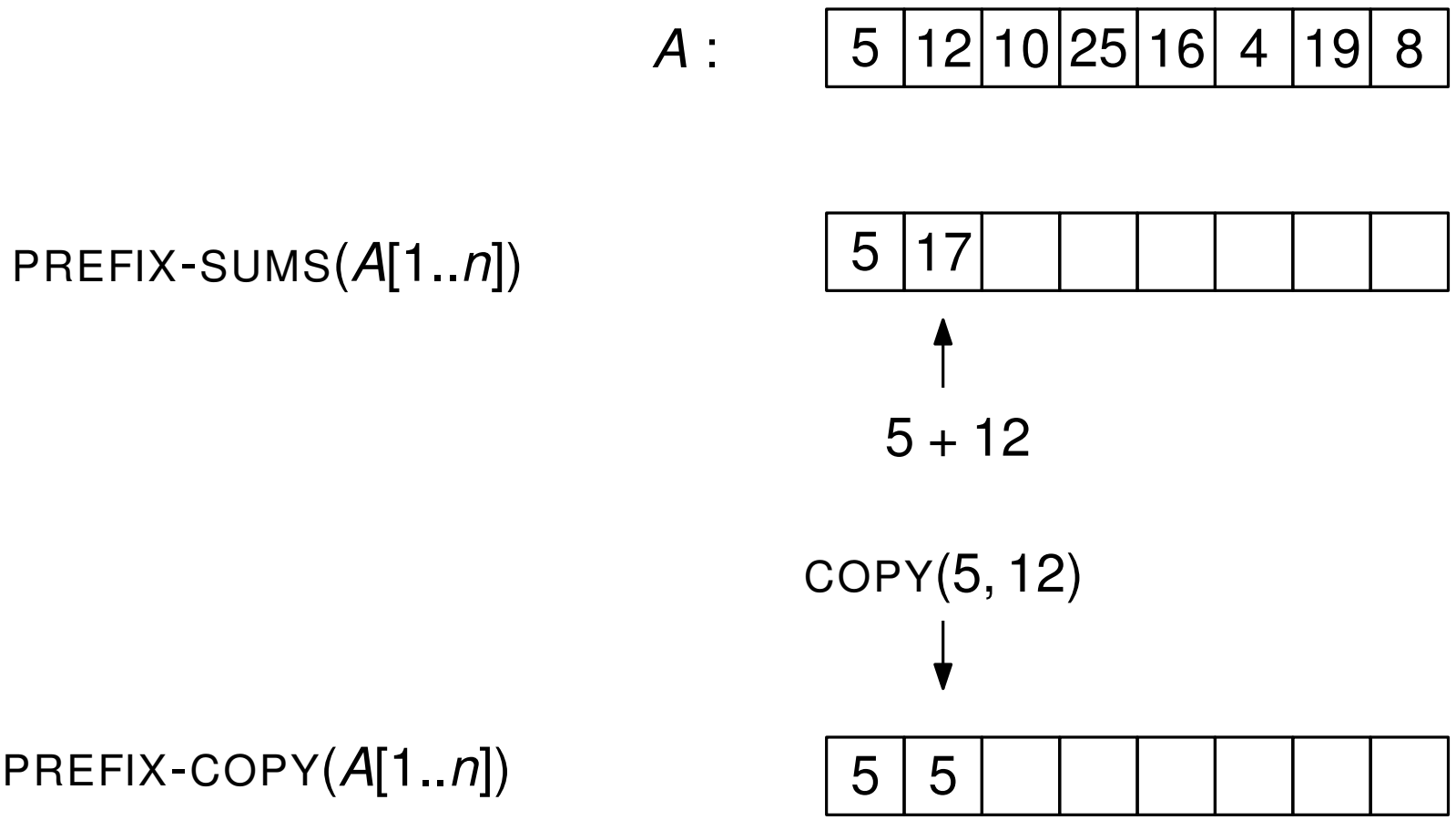
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



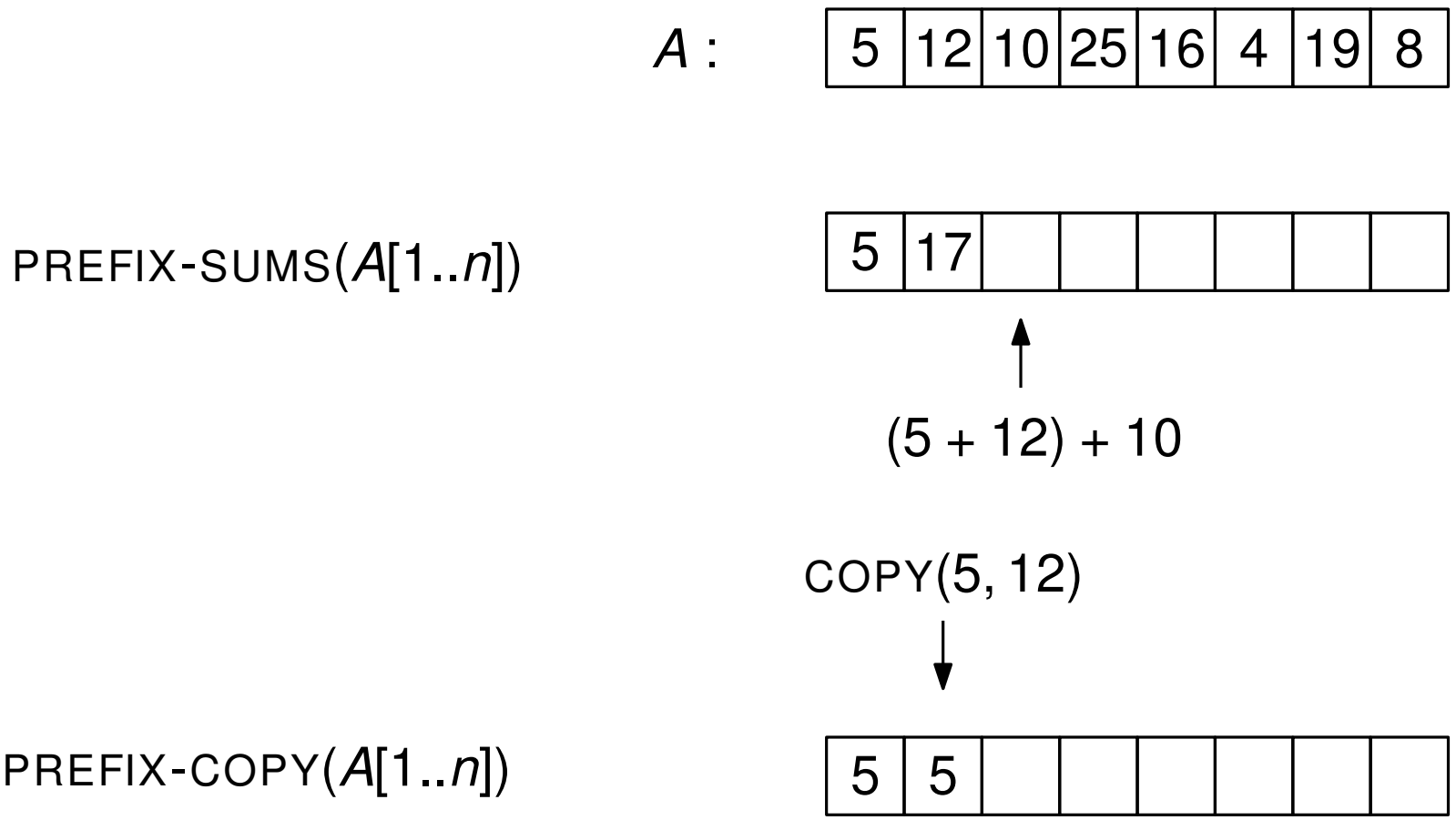
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



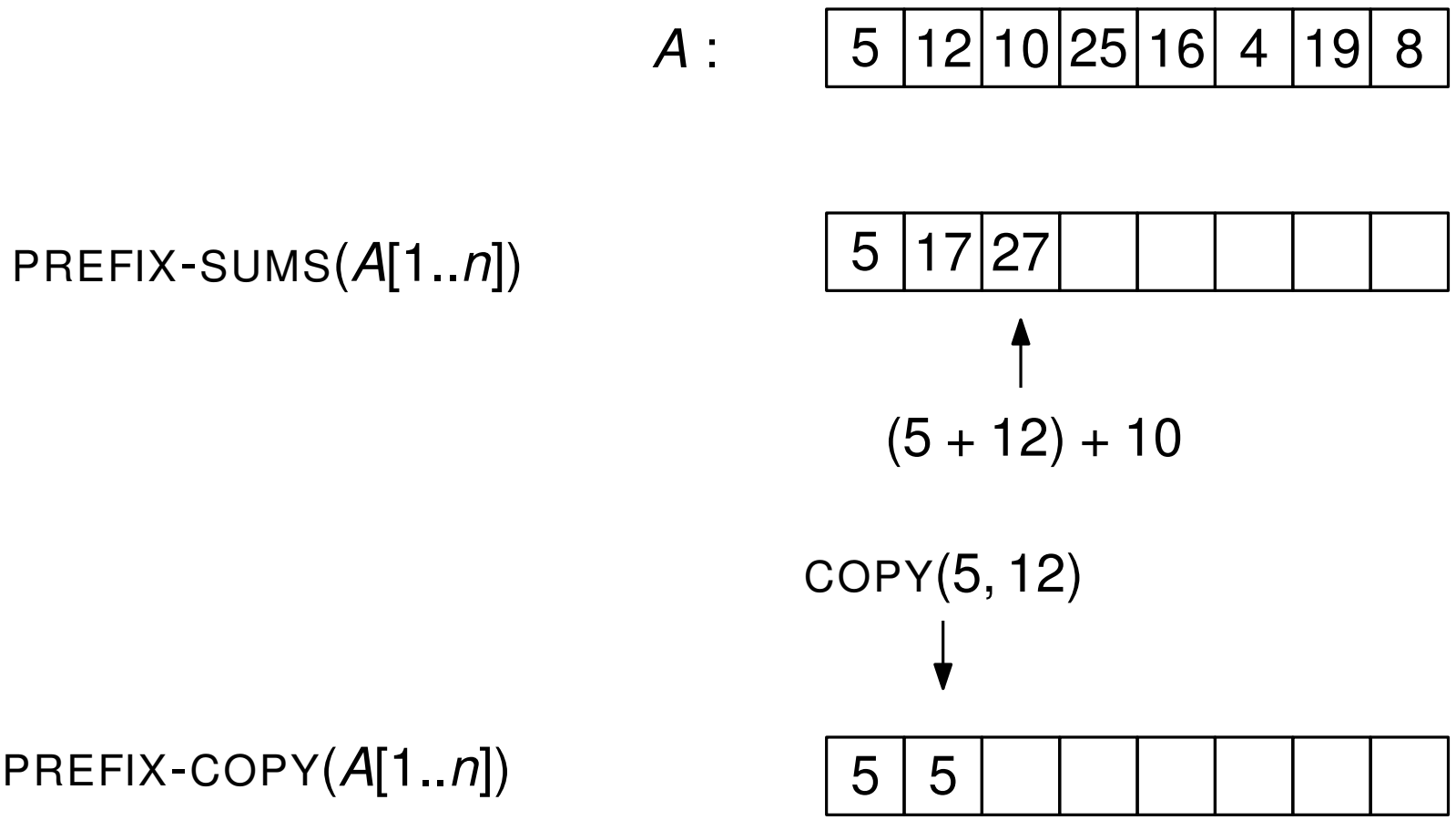
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



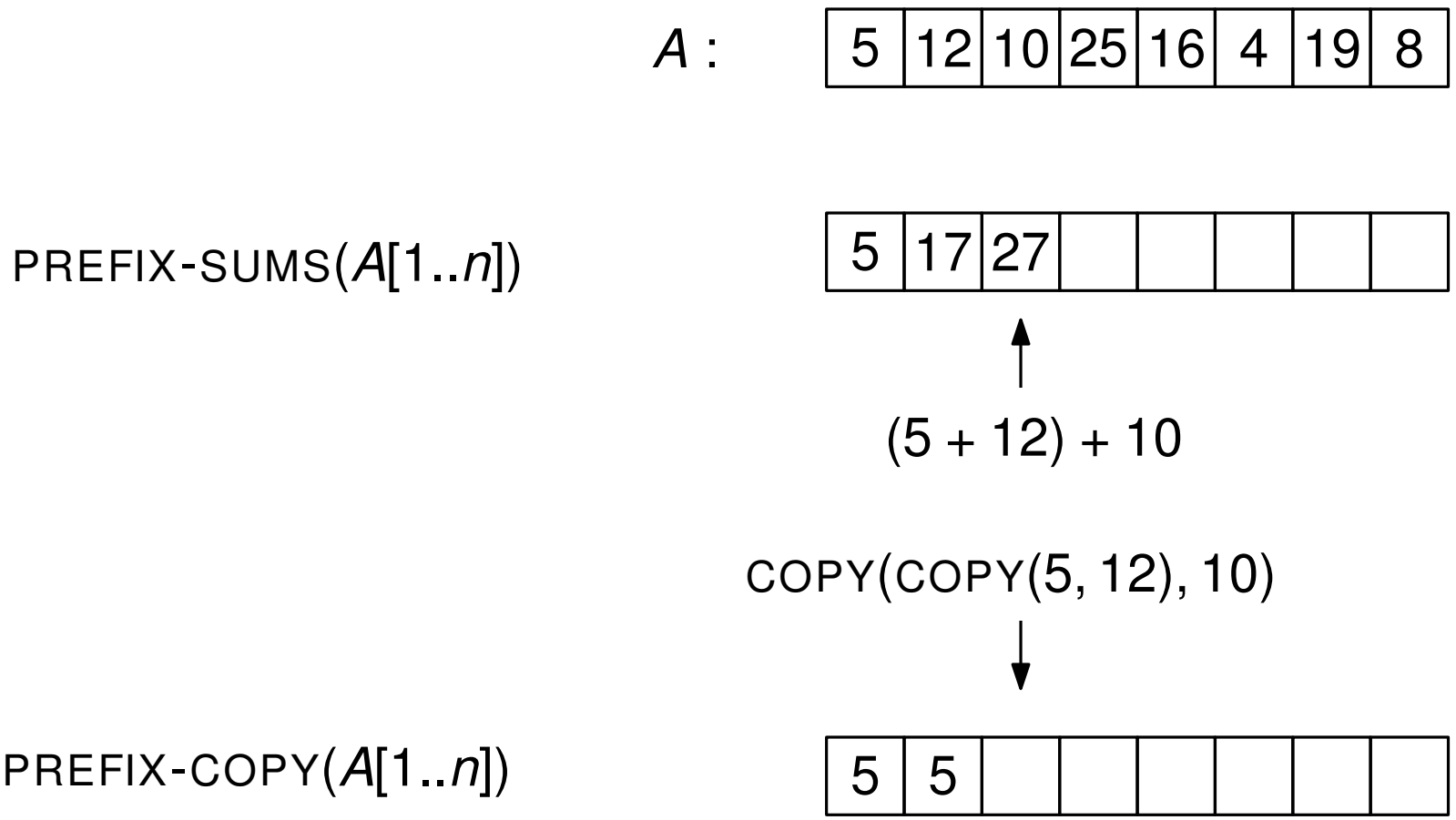
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



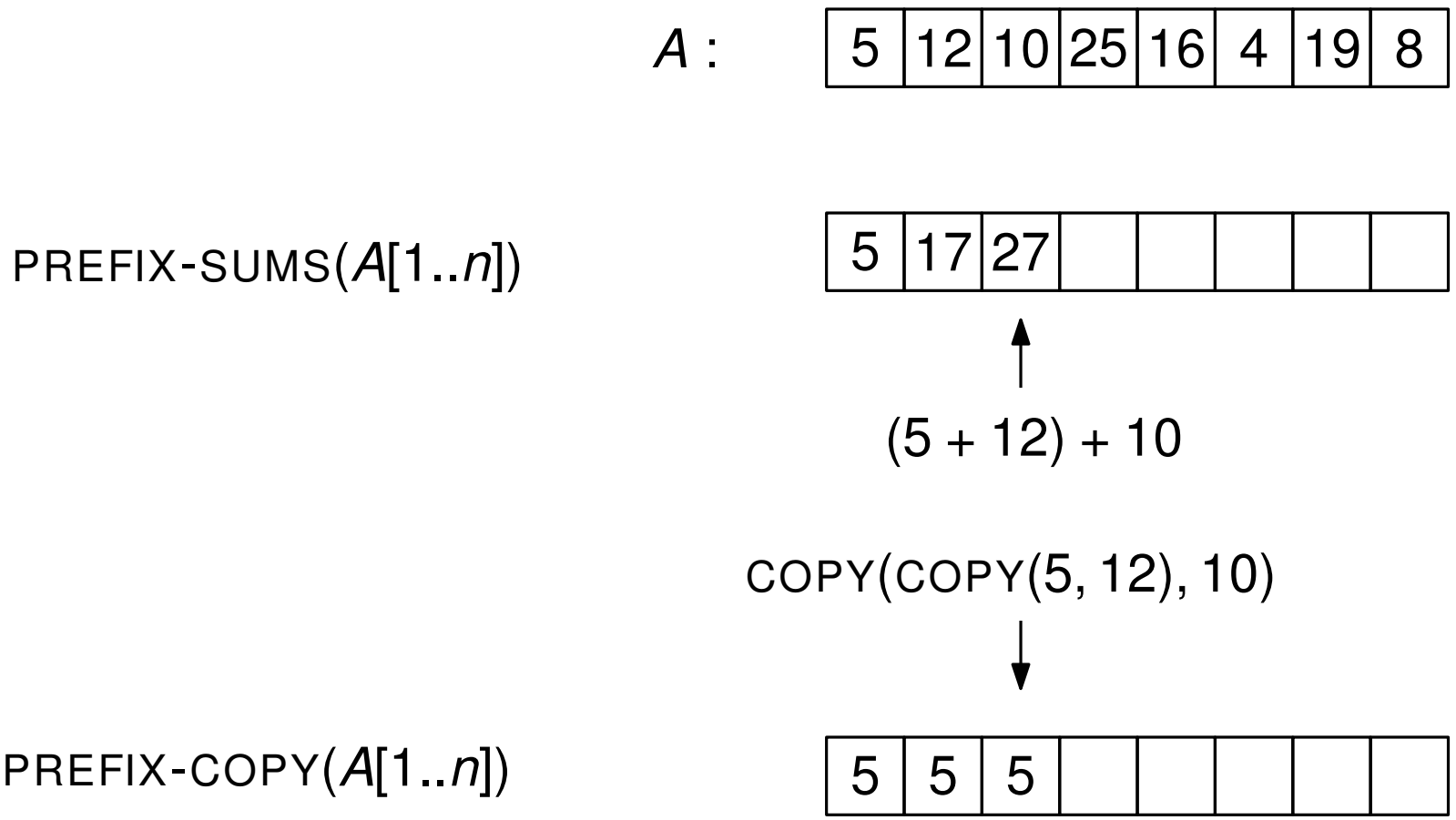
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



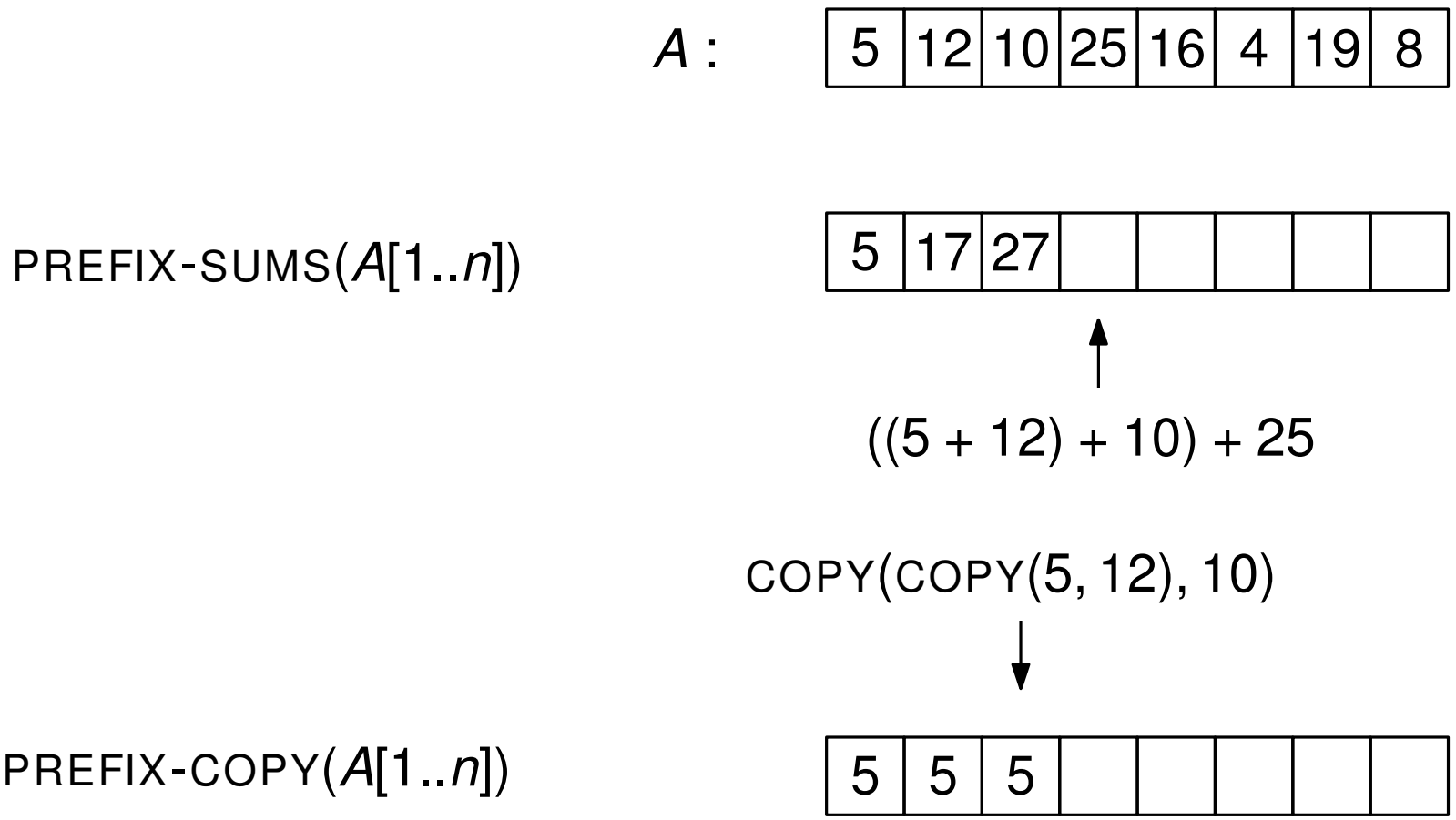
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



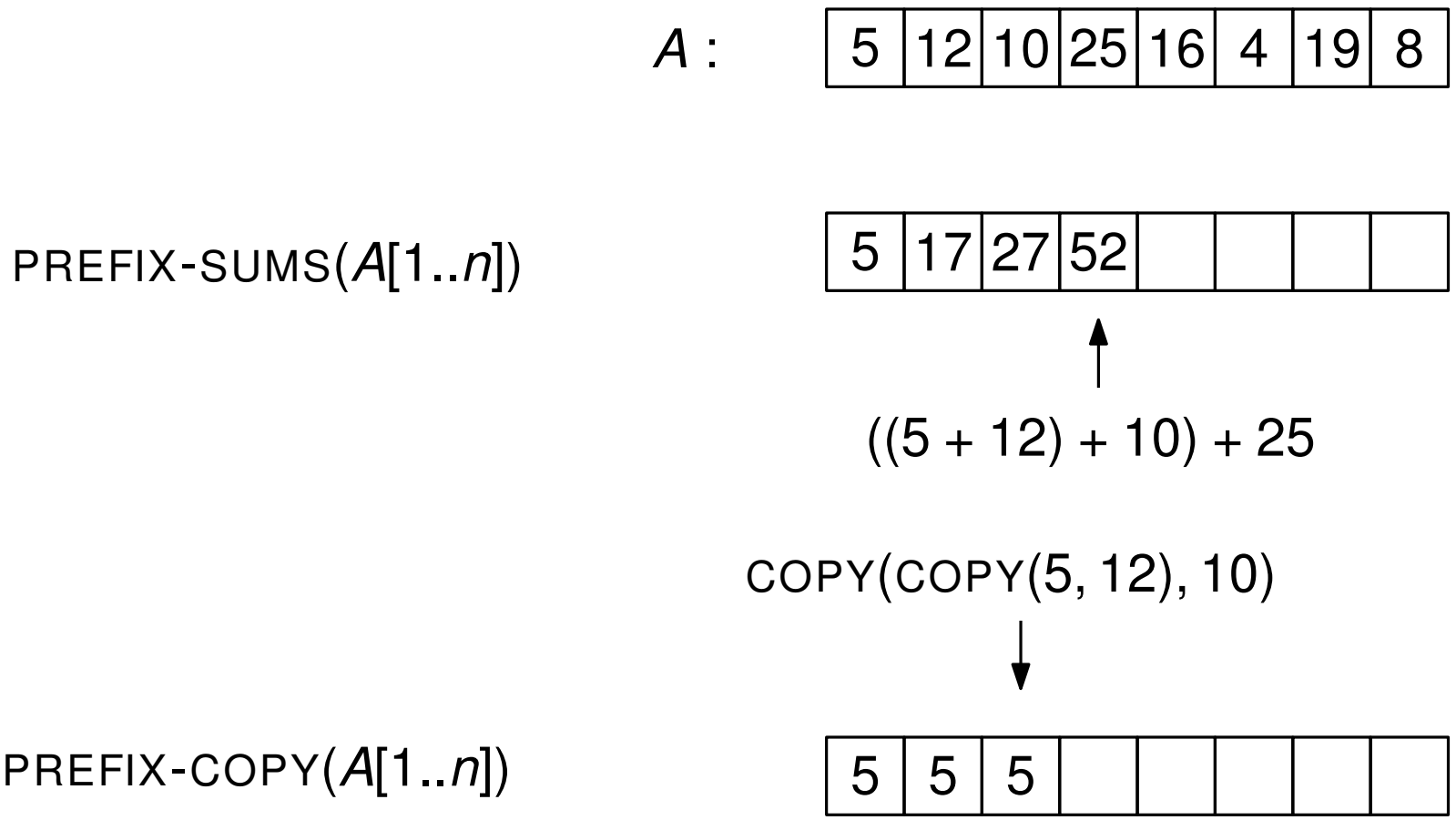
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



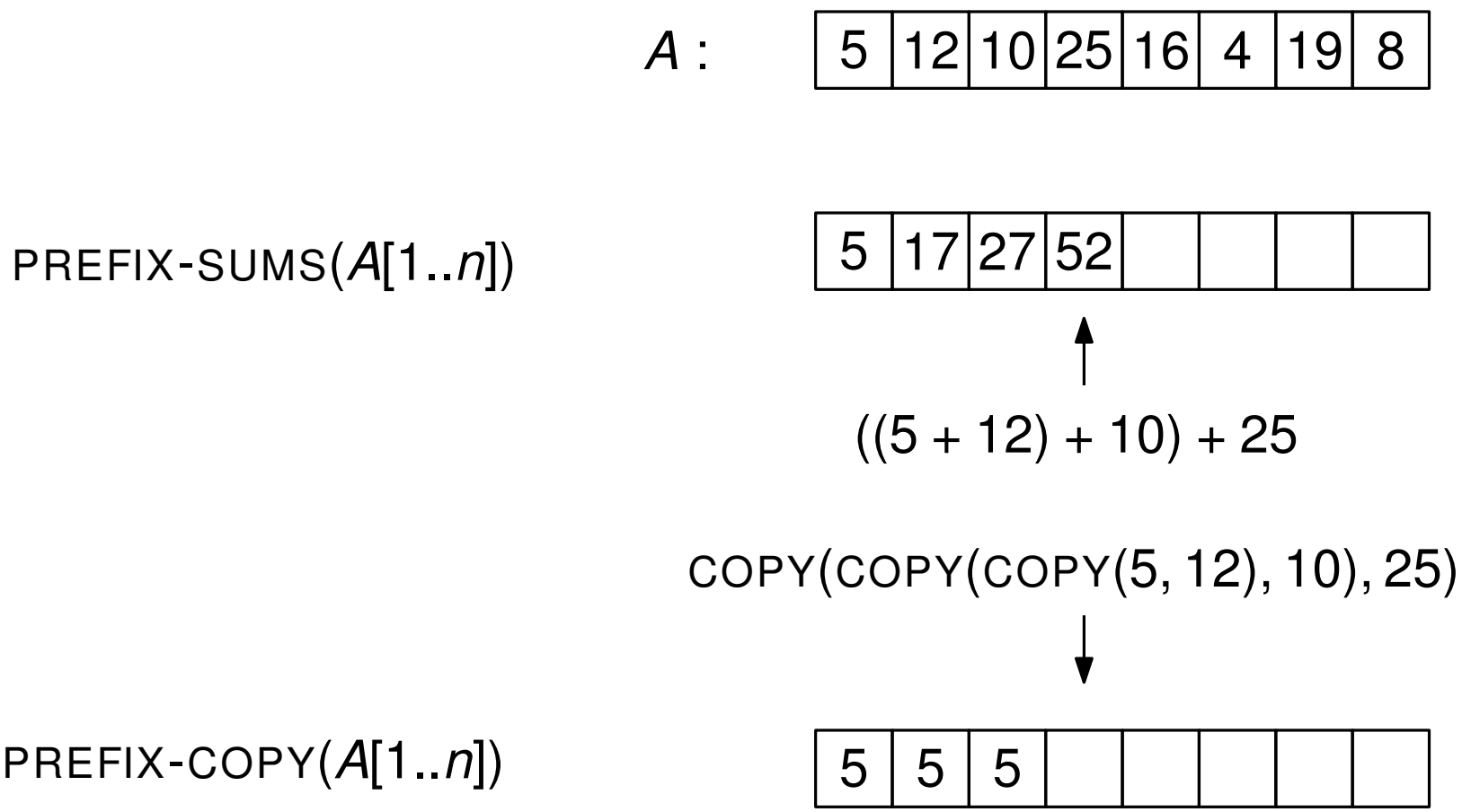
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



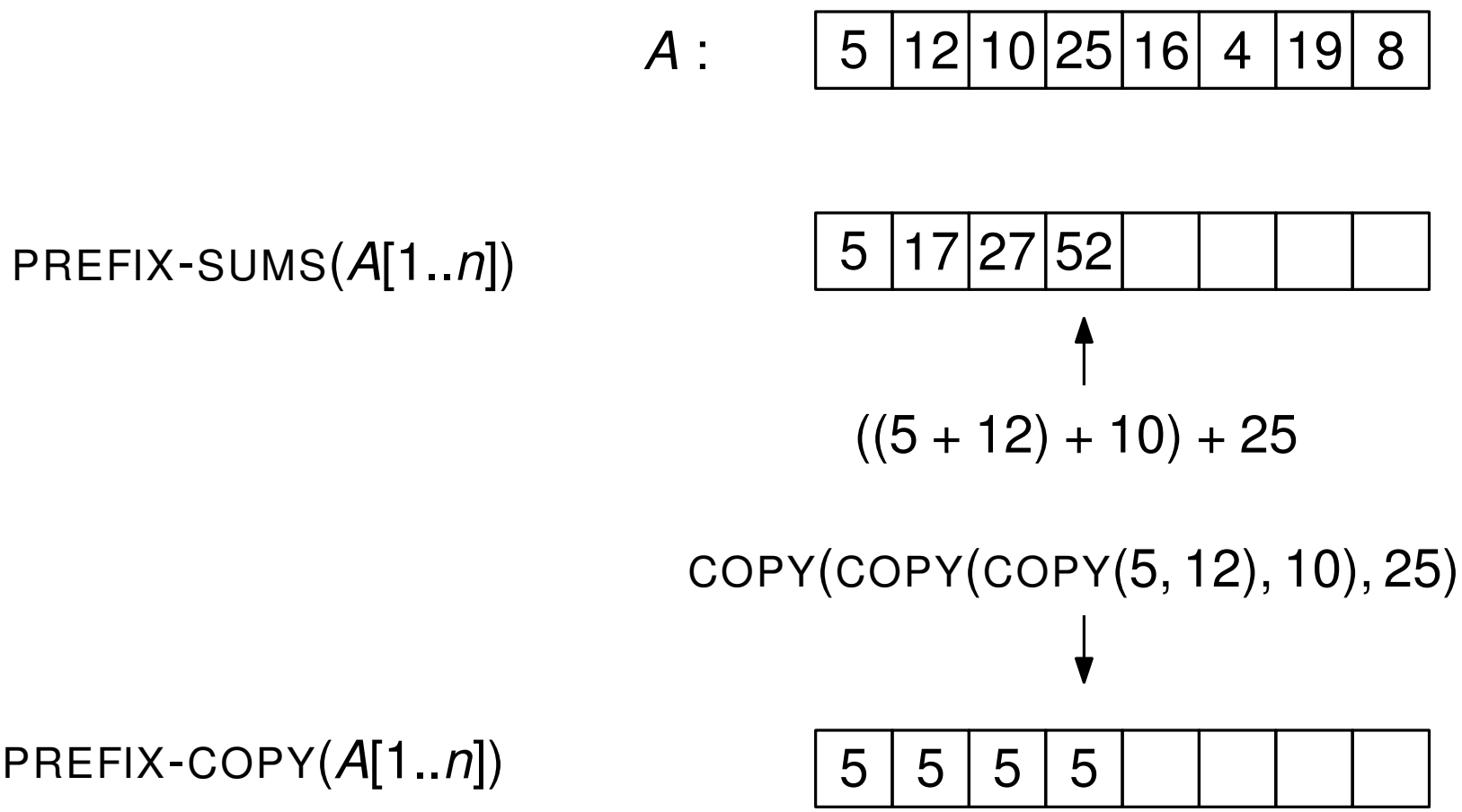
Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator



Broadcast = Prefix Copy

Prefix sums with COPY as the associative operator

A :

5	12	10	25	16	4	19	8

PREFIX-SUMS($A[1..n]$)

5	17	27	52	68	72	91	99

PREFIX-COPY($A[1..n]$)

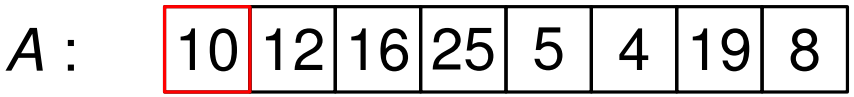
5	5	5	5	5	5	5	5

PARTITION($A[1..n]$)

Problem. *Given an array $A[1..n]$, partition it around the pivot $A[1]$: place all entries that are at most the pivot to the left of the pivot and all entries that are greater after it.*

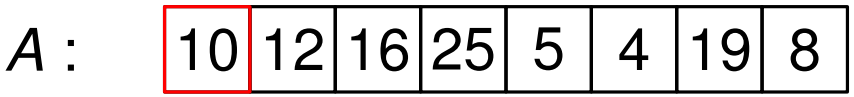
PARTITION($A[1..n]$)

Problem. *Given an array $A[1..n]$, partition it around the pivot $A[1]$: place all entries that are at most the pivot to the left of the pivot and all entries that are greater after it.*



PARTITION($A[1..n]$)

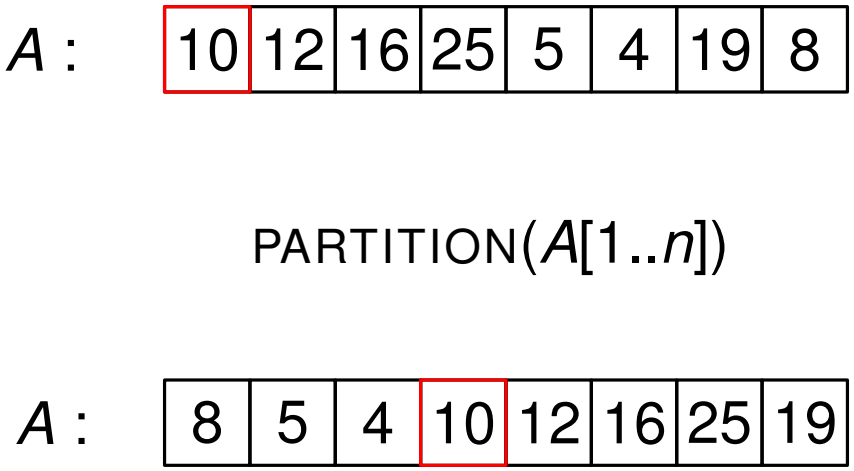
Problem. *Given an array $A[1..n]$, partition it around the pivot $A[1]$: place all entries that are at most the pivot to the left of the pivot and all entries that are greater after it.*



PARTITION($A[1..n]$)

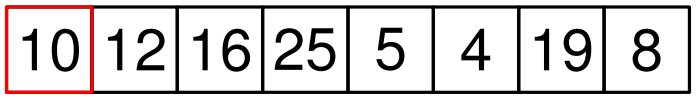
PARTITION($A[1..n]$)

Problem. *Given an array $A[1..n]$, partition it around the pivot $A[1]$: place all entries that are at most the pivot to the left of the pivot and all entries that are greater after it.*



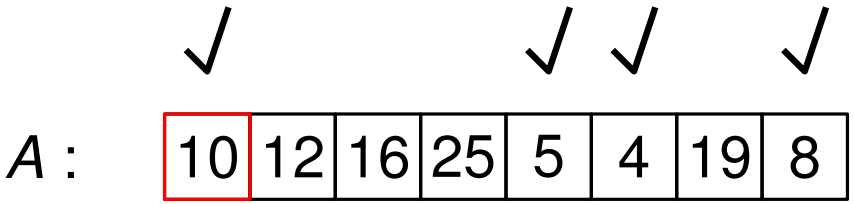
PARTITION($A[1..n]$)

A :



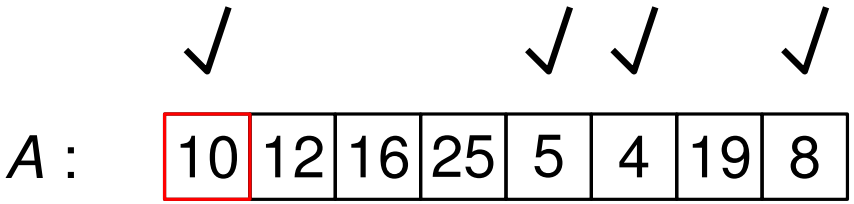
procedure PARTITION($A[1..n]$)

PARTITION($A[1..n]$)



procedure PARTITION($A[1..n]$)

PARTITION($A[1..n]$)



procedure PARTITION($A[1..n]$)

$flags[1..n]$ = MARK($A[1..n]$)

PARTITION($A[1..n]$)

flags :	1	0	0	0	1	1	0	1
A :	10	12	16	25	5	4	19	8

procedure PARTITION($A[1..n]$)

$flags[1..n]$ = MARK($A[1..n]$)

PARTITION($A[1..n]$)

flags :	1	0	0	0	1	1	0	1
A :	10	12	16	25	5	4	19	8

procedure PARTITION($A[1..n]$)

$flags[1..n]$ = MARK($A[1..n]$)

B = FILTER($A[1..n]$, $flags[1..n]$)

PARTITION($A[1..n]$)

flags :	1	0	0	0	1	1	0	1
A :	10	12	16	25	5	4	19	8
B :	10	5	4	8				

procedure PARTITION($A[1..n]$)

flags$[1..n]$ = MARK($A[1..n]$)

B = FILTER($A[1..n]$, *flags*$[1..n]$)

PARTITION($A[1..n]$)

flags :	1	0	0	0	1	1	0	1
A :	10	12	16	25	5	4	19	8
B :	10	5	4	8				

procedure PARTITION($A[1..n]$)

$flags[1..n]$ = MARK($A[1..n]$)

B = FILTER($A[1..n]$, $flags[1..n]$)

FLIP($flags[1..n]$)

PARTITION($A[1..n]$)

flags :	0	1	1	1	0	0	1	0
A :	10	12	16	25	5	4	19	8
B :	10	5	4	8				

procedure PARTITION($A[1..n]$)

flags$[1..n]$ = MARK($A[1..n]$)

B = FILTER($A[1..n]$, *flags*$[1..n]$)

FLIP(*flags*$[1..n]$)

PARTITION($A[1..n]$)

flags :	0	1	1	1	0	0	1	0
A :	10	12	16	25	5	4	19	8
B :	10	5	4	8				

procedure PARTITION($A[1..n]$)

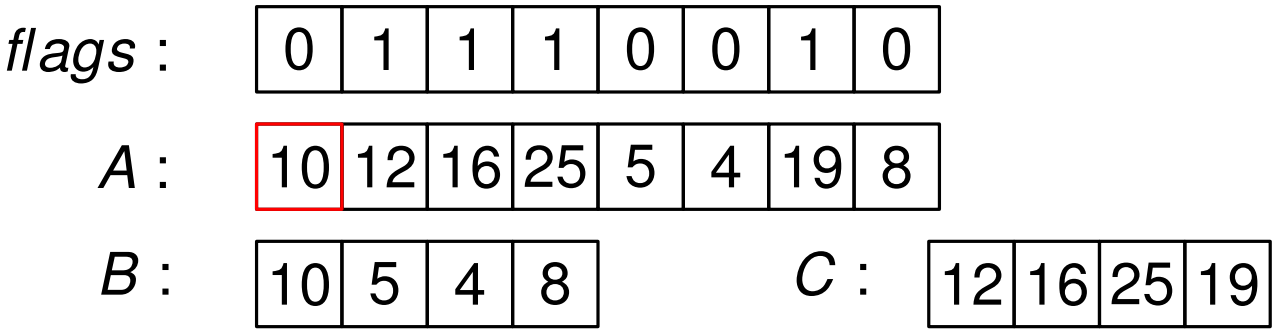
$flags[1..n]$ = MARK($A[1..n]$)

B = FILTER($A[1..n]$, $flags[1..n]$)

FLIP($flags[1..n]$)

C = FILTER($A[1..n]$, $flags[1..n]$)

PARTITION($A[1..n]$)



procedure PARTITION($A[1..n]$)

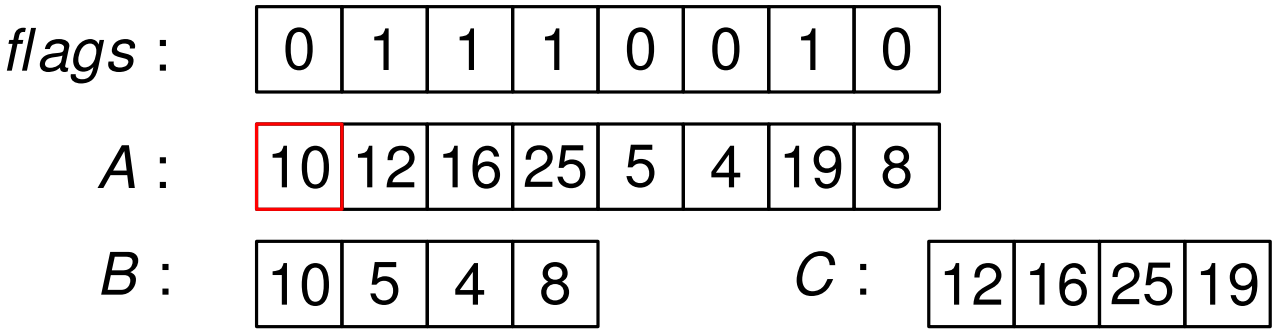
$flags[1..n]$ = MARK($A[1..n]$)

B = FILTER($A[1..n]$, $flags[1..n]$)

FLIP($flags[1..n]$)

C = FILTER($A[1..n]$, $flags[1..n]$)

PARTITION($A[1..n]$)



procedure PARTITION($A[1..n]$)

$flags[1..n]$ = MARK($A[1..n]$)

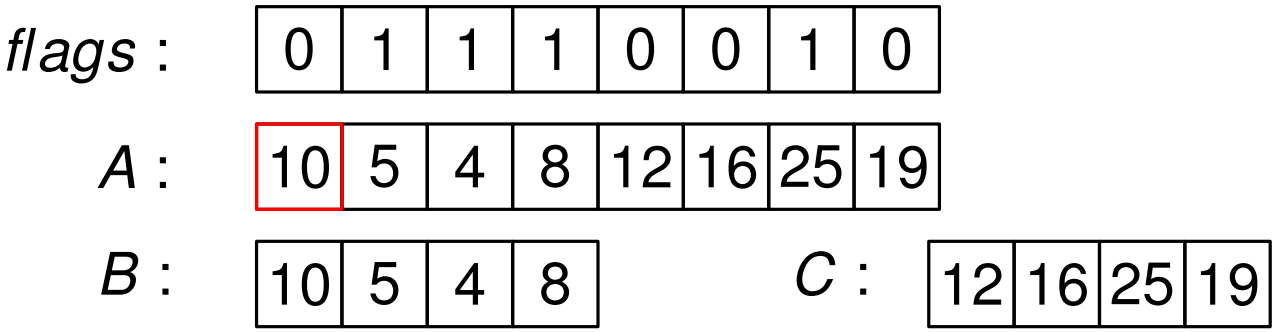
B = FILTER($A[1..n]$, $flags[1..n]$)

FLIP($flags[1..n]$)

C = FILTER($A[1..n]$, $flags[1..n]$)

COMBINE(B, C, A)

PARTITION($A[1..n]$)



procedure PARTITION($A[1..n]$)

$flags[1..n]$ = MARK($A[1..n]$)

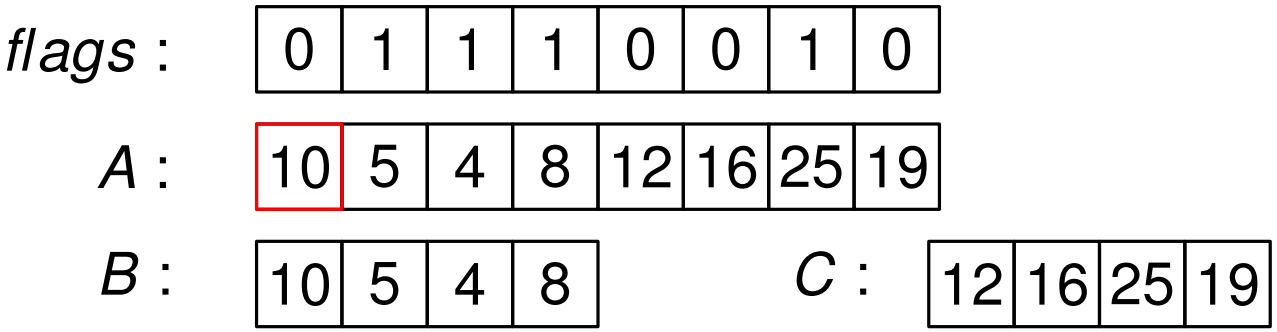
B = FILTER($A[1..n]$, $flags[1..n]$)

FLIP($flags[1..n]$)

C = FILTER($A[1..n]$, $flags[1..n]$)

COMBINE(B, C, A)

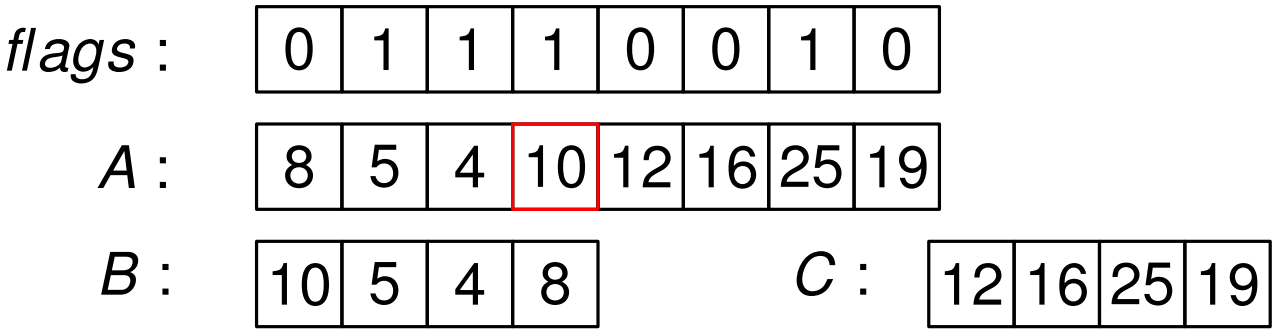
PARTITION($A[1..n]$)



procedure PARTITION($A[1..n]$)

$flags[1..n]$ = MARK($A[1..n]$)
B = FILTER($A[1..n]$, $flags[1..n]$)
FLIP($flags[1..n]$)
C = FILTER($A[1..n]$, $flags[1..n]$)
COMBINE(B, C, A)
SWAP($A[1]$, $A[B.size]$)

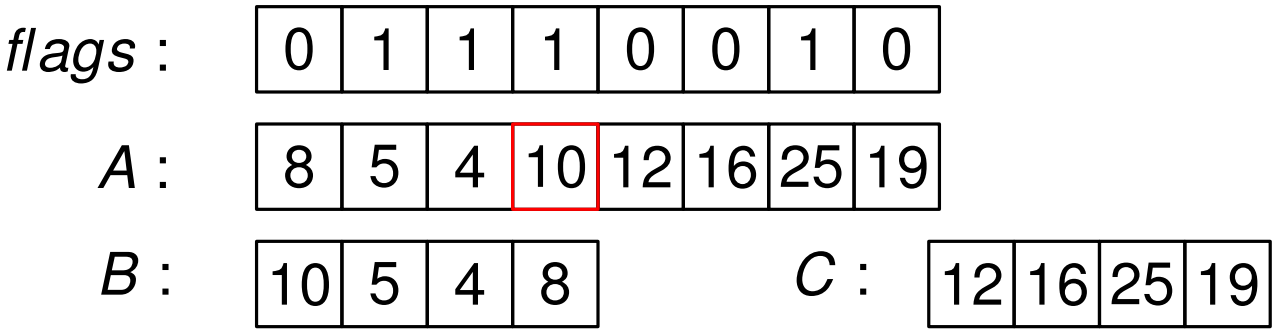
PARTITION($A[1..n]$)



procedure PARTITION($A[1..n]$)

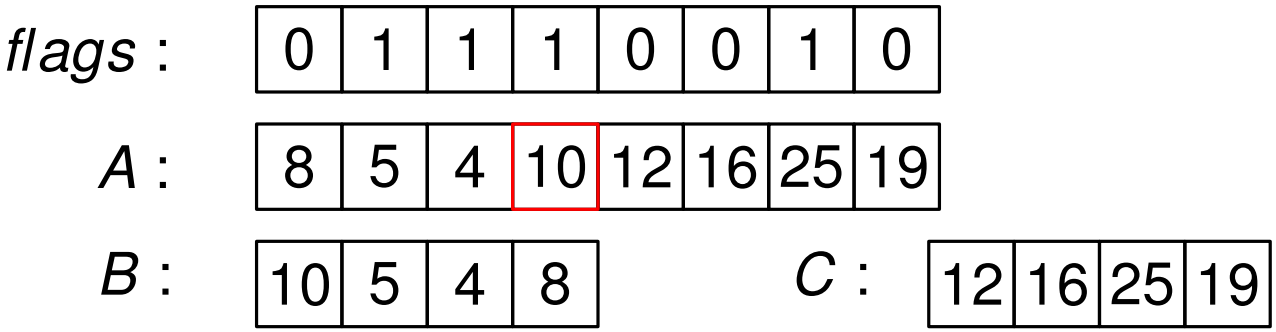
$flags[1..n]$ = MARK($A[1..n]$)
B = FILTER($A[1..n]$, $flags[1..n]$)
FLIP($flags[1..n]$)
C = FILTER($A[1..n]$, $flags[1..n]$)
COMBINE(B, C, A)
SWAP($A[1]$, $A[B.size]$)

PARTITION($A[1..n]$)




```
procedure PARTITION(A[1..n])
    flags[1..n] = MARK(A[1..n])
    B = FILTER(A[1..n], flags[1..n])
    FLIP(flags[1..n])
    C = FILTER(A[1..n], flags[1..n])
    COMBINE(B, C, A)
    SWAP(A[1], A[B.size])
    return B.size
```

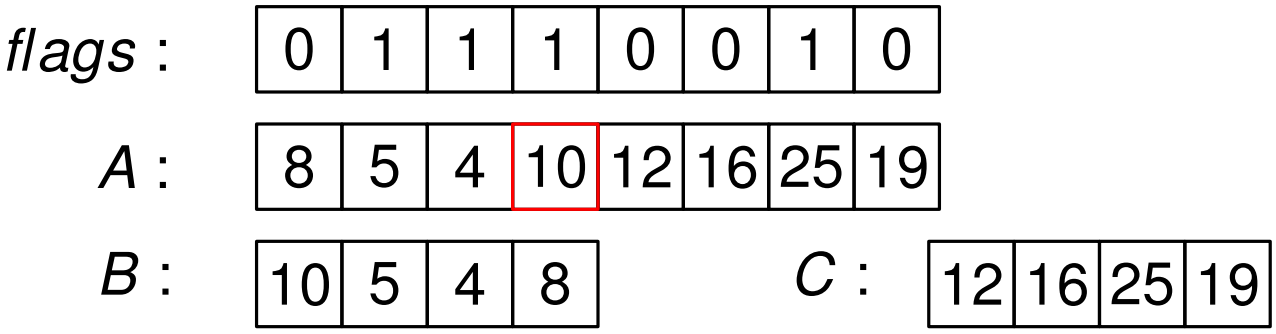
PARTITION($A[1..n]$)




```
procedure PARTITION(A[1..n])
    flags[1..n] = MARK(A[1..n])
    B = FILTER(A[1..n], flags[1..n])
    FLIP(flags[1..n])
    C = FILTER(A[1..n], flags[1..n])
    COMBINE(B, C, A)
    SWAP(A[1], A[B.size])
    return B.size
```

```
procedure MARK(A[1..n])
    flags = new array[n]
    for i = 1 to n in parallel do
        if A[1] ≤ A[i] then
            flags[i] = 1
        else
            flags[i] = 0
    return flags[1..n]
```

PARTITION($A[1..n]$)

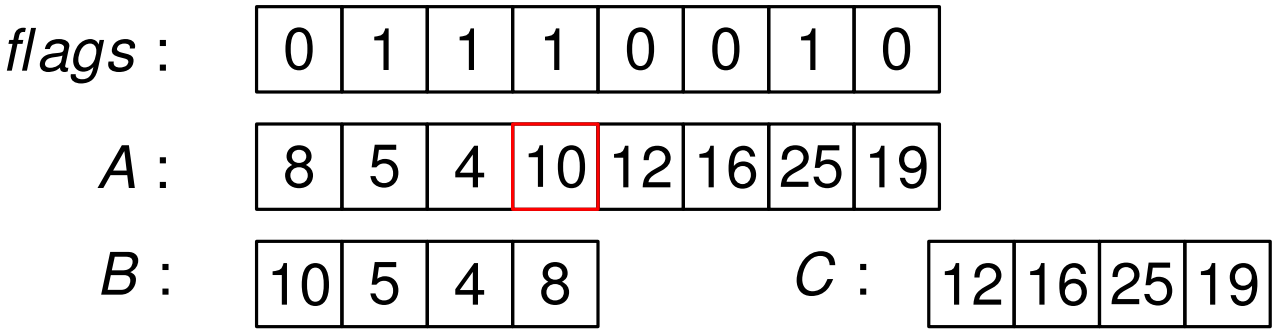



```
procedure PARTITION(A[1..n])
    flags[1..n] = MARK(A[1..n])
    B = FILTER(A[1..n], flags[1..n])
    FLIP(flags[1..n])
    C = FILTER(A[1..n], flags[1..n])
    COMBINE(B, C, A)
    SWAP(A[1], A[B.size])
    return B.size
```

```
procedure MARK(A[1..n])
    flags = new array[n]
    for i = 1 to n in parallel do
        if A[1] ≤ A[i] then
            flags[i] = 1
        else
            flags[i] = 0
    return flags[1..n]
```

not EREW

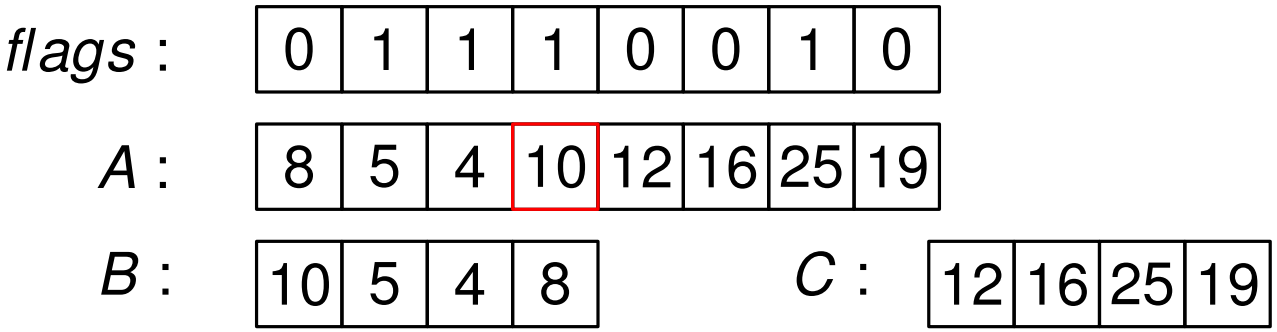
PARTITION($A[1..n]$)




```
procedure PARTITION(A[1..n])
    flags[1..n] = MARK(A[1..n])
    B = FILTER(A[1..n], flags[1..n])
    FLIP(flags[1..n])
    C = FILTER(A[1..n], flags[1..n])
    COMBINE(B, C, A)
    SWAP(A[1], A[B.size])
    return B.size
```

```
procedure MARK(A[1..n])
    flags, pivots = new array[n]
    pivots[1] = A[1]
    PREFIX-COPY(pivots[1..n])
    for i = 1 to n in parallel do
        if pivots[i] ≤ A[i] then
            flags[i] = 1
        else
            flags[i] = 0
    return flags
```

PARTITION($A[1..n]$)



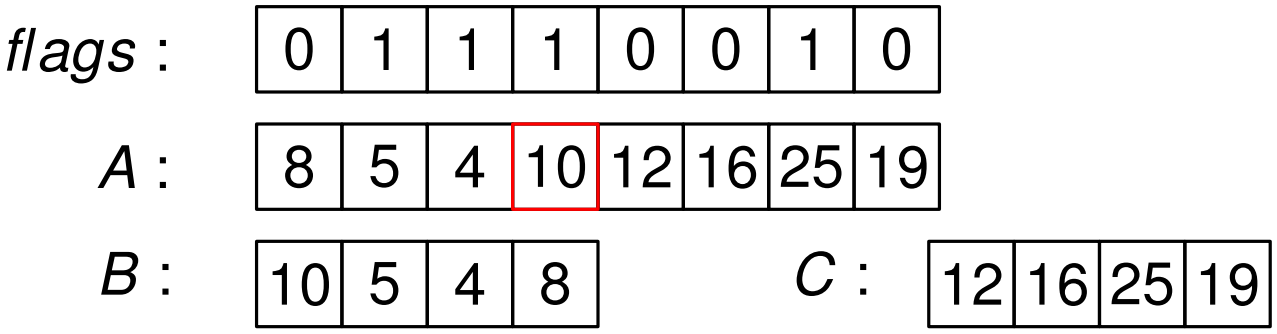
procedure PARTITION($A[1..n]$)
- $flags[1..n]$ = MARK($A[1..n]$)
- B = FILTER($A[1..n]$, $flags[1..n]$)
- FLIP($flags[1..n]$)
- C = FILTER($A[1..n]$, $flags[1..n]$)
- COMBINE(B, C, A)
- SWAP($A[1]$, $A[B.size]$)
- **return** $B.size$

procedure MARK($A[1..n]$)
- $flags$, $pivots$ = new array[n]
- $pivots[1] = A[1]$
- PREFIX-COPY($pivots[1..n]$)
- **for** $i = 1$ to n **in parallel do**
  - **if** $pivots[i] \leq A[i]$ **then**
    - $flags[i] = 1$
  - **else**
    - $flags[i] = 0$
- **return** $flags$

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

$W(n) = O(n)$

PARTITION($A[1..n]$)




```
procedure PARTITION(A[1..n])
    flags[1..n] = MARK(A[1..n])
    B = FILTER(A[1..n], flags[1..n])
    FLIP(flags[1..n])
    C = FILTER(A[1..n], flags[1..n])
    COMBINE(B, C, A)
    SWAP(A[1], A[B.size])
    return B.size
```

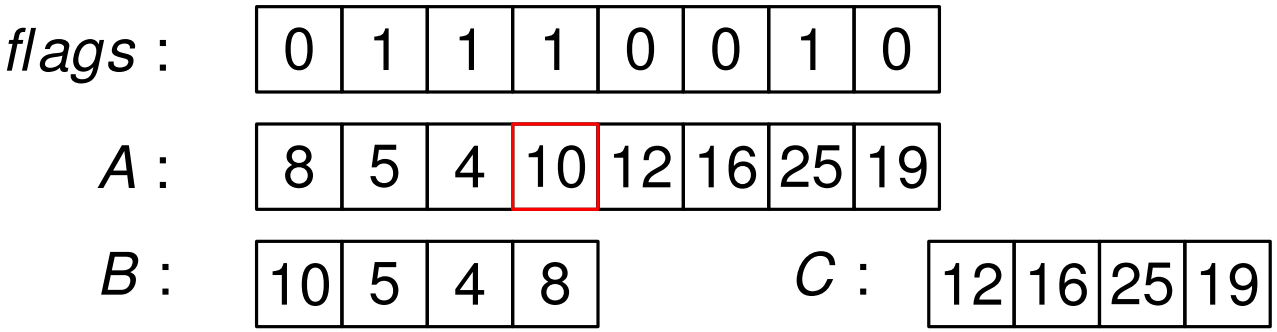
```
procedure MARK(A[1..n])
    flags, pivots = new array[n]
    pivots[1] = A[1]
    PREFIX-COPY(pivots[1..n])
    for i = 1 to n in parallel do
        if pivots[i] ≤ A[i] then
            flags[i] = 1
        else
            flags[i] = 0
    return flags
```

```
procedure FLIP(flags[1..n])
    for i = 1 to n in parallel do
        flags[i] = 1 − flags[i]
```

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

$W(n) = O(n)$

PARTITION($A[1..n]$)




```
procedure PARTITION(A[1..n])
    flags[1..n] = MARK(A[1..n])
    B = FILTER(A[1..n], flags[1..n])
    FLIP(flags[1..n])
    C = FILTER(A[1..n], flags[1..n])
    COMBINE(B, C, A)
    SWAP(A[1], A[B.size])
    return B.size
```

```
procedure MARK(A[1..n])
    flags, pivots = new array[n]
    pivots[1] = A[1]
    PREFIX-COPY(pivots[1..n])
    for i = 1 to n in parallel do
        if pivots[i] ≤ A[i] then
            flags[i] = 1
        else
            flags[i] = 0
    return flags
```

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

$W(n) = O(n)$

```
procedure FLIP(flags[1..n])
    for i = 1 to n in parallel do
        flags[i] = 1 − flags[i]
```

$T(n) = O(1)$

$W(n) = O(n)$

COMBINE(B, C, A)

COMBINE(B, C, A)

B :

10	5	4	8

C :

12	16	25	19

COMBINE(B, C, A)

A :

10	5	4	8	12	16	25	19

COMBINE(B, C, A)

B :

10	5	4	8

C :

12	16	25	19

COMBINE(B, C, A)

A :

10	5	4	8	12	16	25	19

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
    for $i = 1$ to k **in parallel do**
        $A[i] = B[i]$
    for $i = 1$ **to** m **in parallel do**
        $A[k+i] = C[i]$

COMBINE(B, C, A)

B :

10	5	4	8

C :

12	16	25	19

COMBINE(B, C, A)

A :

10	5	4	8	12	16	25	19

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$

$T(n) = O(1)$

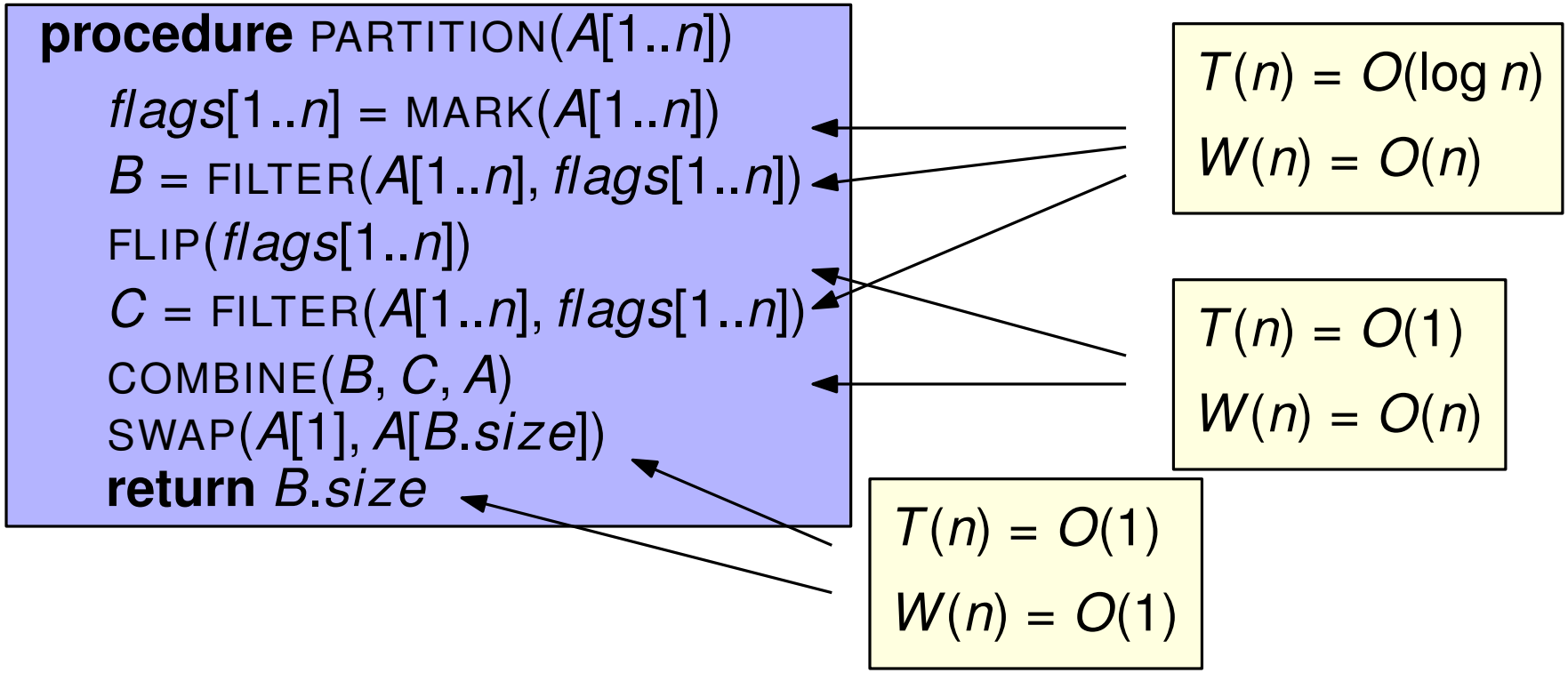
$W(n) = O(n)$

$n = k + m$

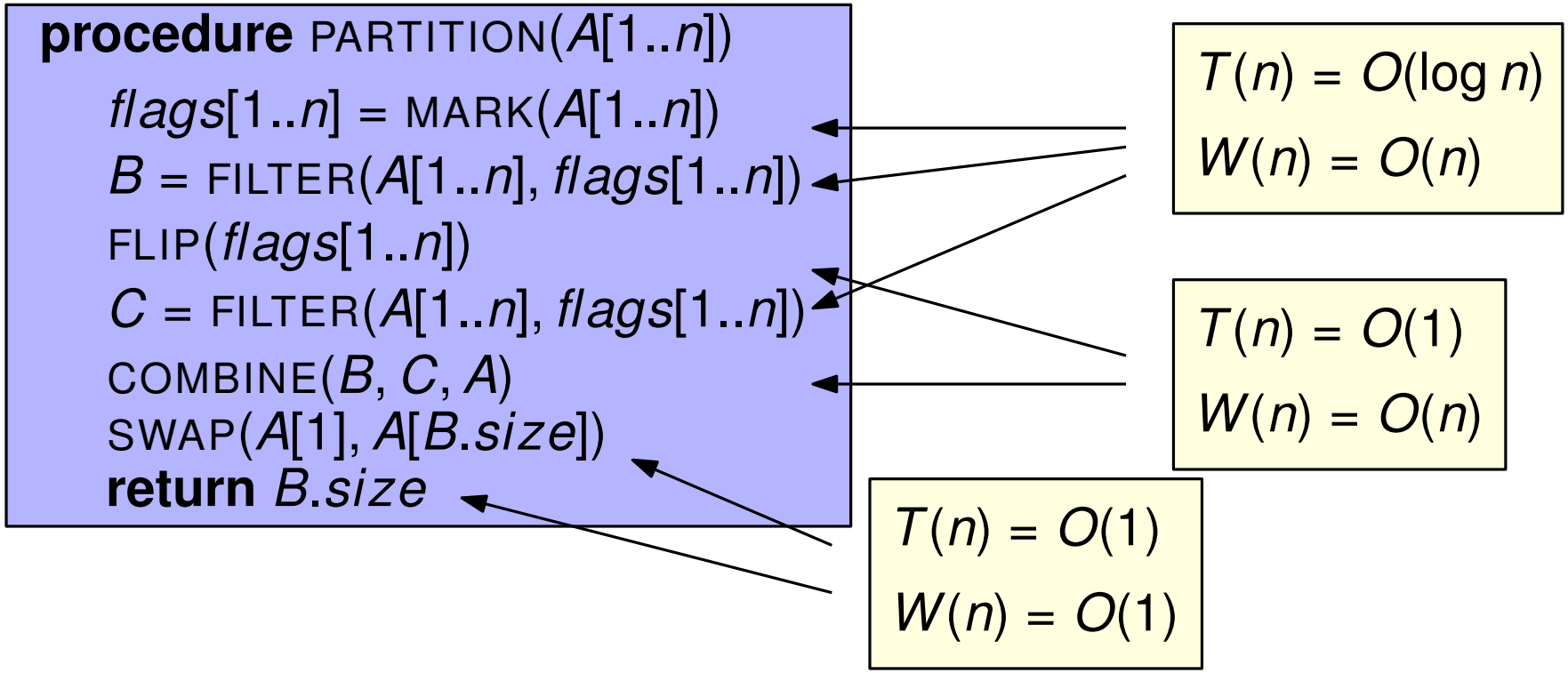
PARTITION($A[1..n]$) Analysis

```
procedure PARTITION(A[1..n])
    flags[1..n] = MARK(A[1..n])
    B = FILTER(A[1..n], flags[1..n])
    FLIP(flags[1..n])
    C = FILTER(A[1..n], flags[1..n])
    COMBINE(B, C, A)
    SWAP(A[1], A[B.size])
    return B.size
```

PARTITION($A[1..n]$) Analysis



PARTITION($A[1..n]$) Analysis



Total

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

$W(n) = O(n)$

QUICKSORT($A[1..n]$)

QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP($A[i], A[pivot]$)
    k = PARTITION($A[i..j]$)
    in parallel do
      QUICKSORT($A[i..k-1]$)
      QUICKSORT($A[k+1..j]$)

QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP($A[i], A[pivot]$)
    k = PARTITION($A[i..j]$)
    in parallel do
      QUICKSORT($A[i..k-1]$)
      QUICKSORT($A[k+1..j]$)

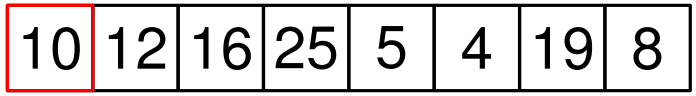
A :

10	12	16	25	5	4	19	8

QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP($A[i], A[pivot]$)
    k = PARTITION($A[i..j]$)
    in parallel do
      QUICKSORT($A[i..k-1]$)
      QUICKSORT($A[k+1..j]$)

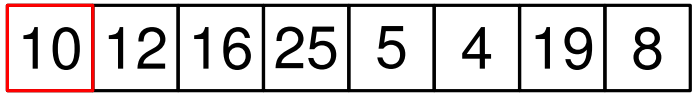
A :



QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP$(A[i], A[pivot])$
    k = PARTITION$(A[i..j])$
    in parallel do
      QUICKSORT$(A[i..k-1])$
      QUICKSORT$(A[k+1..j])$

A :

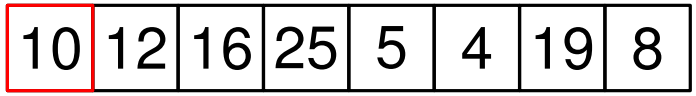


PARTITION$(A[1..n])$

QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP$(A[i], A[pivot])$
    k = PARTITION$(A[i..j])$
    in parallel do
      QUICKSORT$(A[i..k-1])$
      QUICKSORT$(A[k+1..j])$

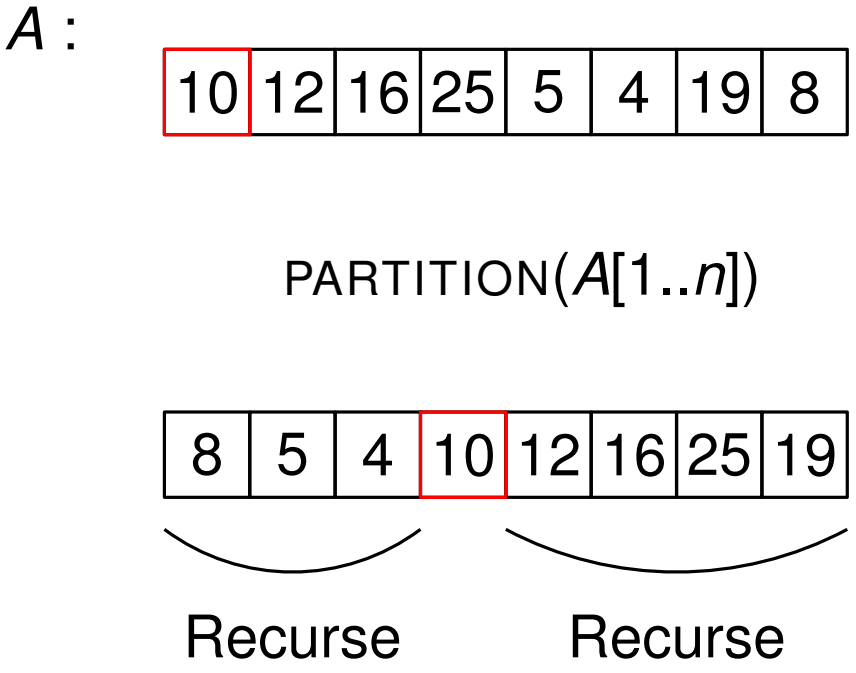
A :



PARTITION$(A[1..n])$

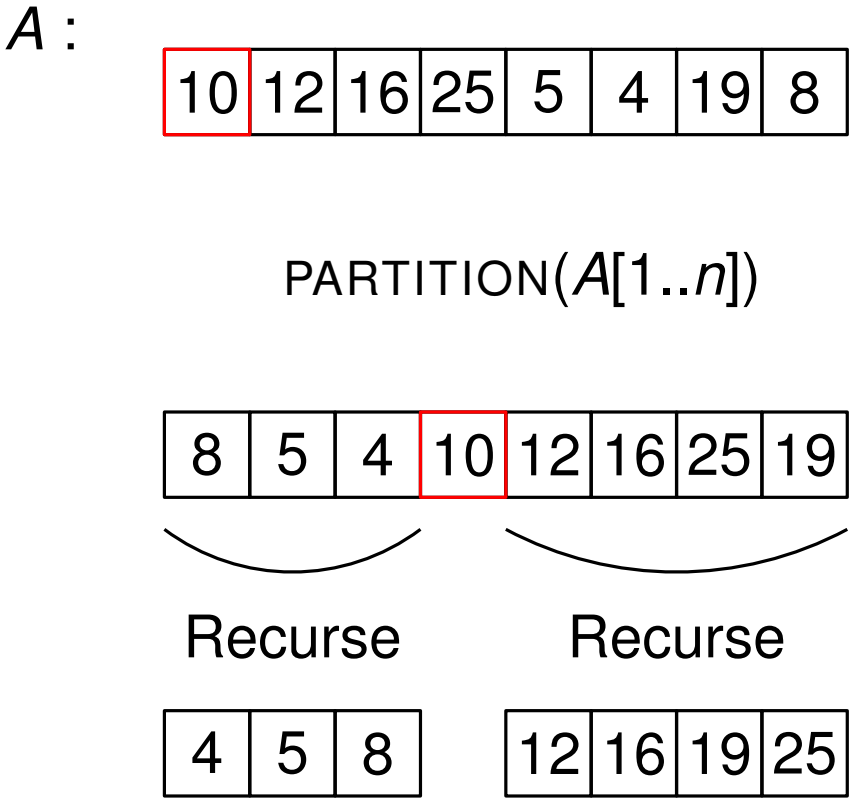
QUICKSORT($A[1..n]$)

```
procedure QUICKSORT(A[i..j])
  if i < j then
      pivot = RANDOM(i, j)
      SWAP(A[i], A[pivot])
      k = PARTITION(A[i..j])
      in parallel do
          QUICKSORT(A[i..k − 1])
          QUICKSORT(A[k + 1..j])
```



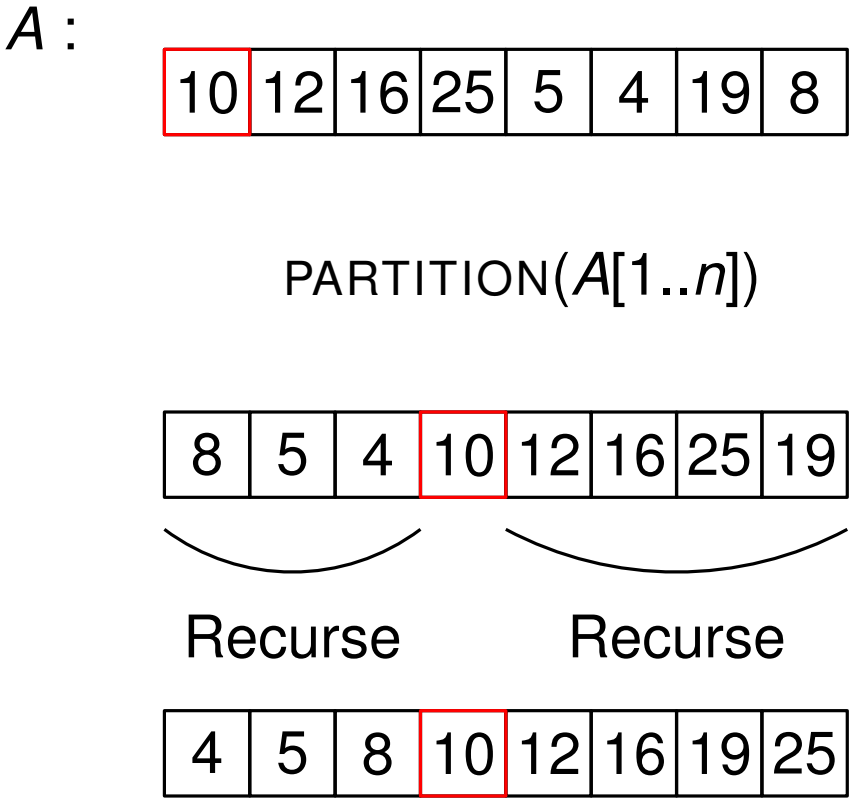
QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP($A[i], A[pivot]$)
    k = PARTITION($A[i..j]$)
    in parallel do
      QUICKSORT($A[i..k-1]$)
      QUICKSORT($A[k+1..j]$)



QUICKSORT($A[1..n]$)

```
procedure QUICKSORT(A[i..j])
  if i < j then
      pivot = RANDOM(i, j)
      SWAP(A[i], A[pivot])
      k = PARTITION(A[i..j])
      in parallel do
          QUICKSORT(A[i..k − 1])
          QUICKSORT(A[k + 1..j])
```

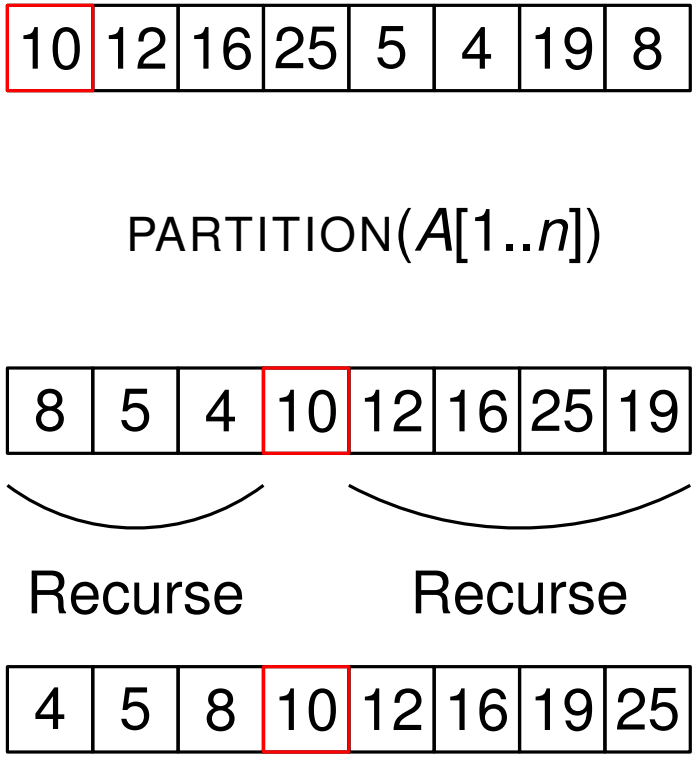


QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP$(A[i], A[pivot])$
    k = PARTITION$(A[i..j])$
    in parallel do
      QUICKSORT$(A[i..k-1])$
      QUICKSORT$(A[k+1..j])$

Analysis:

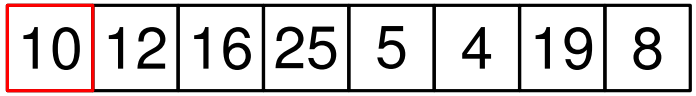
A :



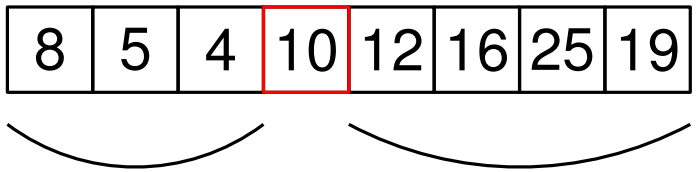
QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP$(A[i], A[pivot])$
    k = PARTITION$(A[i..j])$
    in parallel do
      QUICKSORT$(A[i..k-1])$
      QUICKSORT$(A[k+1..j])$

A :



PARTITION($A[1..n]$)



Recurse Recurse

Analysis:

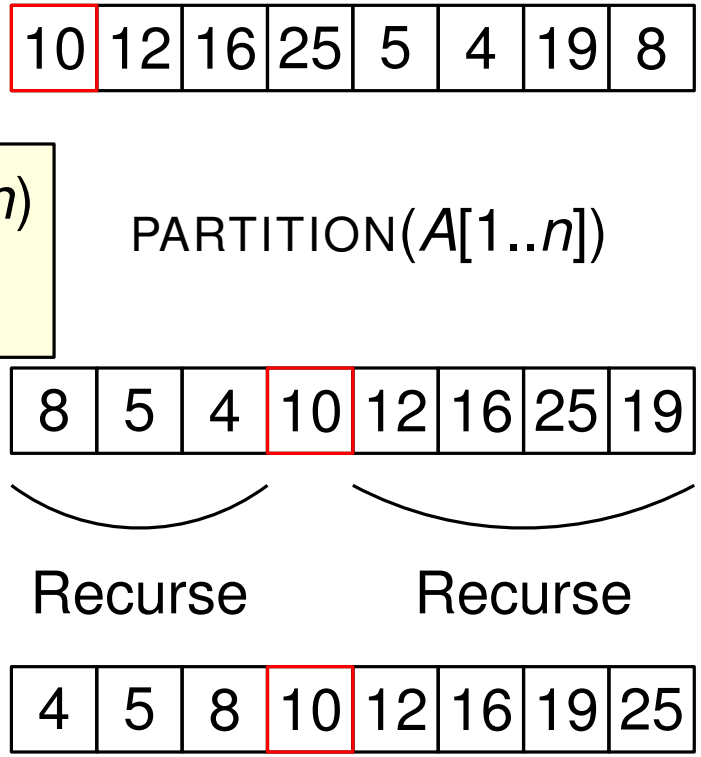
$O(\log n)$ recursive levels in expectation

QUICKSORT($A[1..n]$)

```
procedure QUICKSORT(A[i..j])
  if i < j then
      pivot = RANDOM(i, j)
      SWAP(A[i], A[pivot])
      k = PARTITION(A[i..j])
      in parallel do
          QUICKSORT(A[i..k − 1])
          QUICKSORT(A[k + 1..j])
```

$T(n) = O(1)$
$W(n) = O(1)$

$T(n) = O(\log n)$
$W(n) = O(n)$



Analysis:

$O(\log n)$ recursive levels in expectation

Each level:

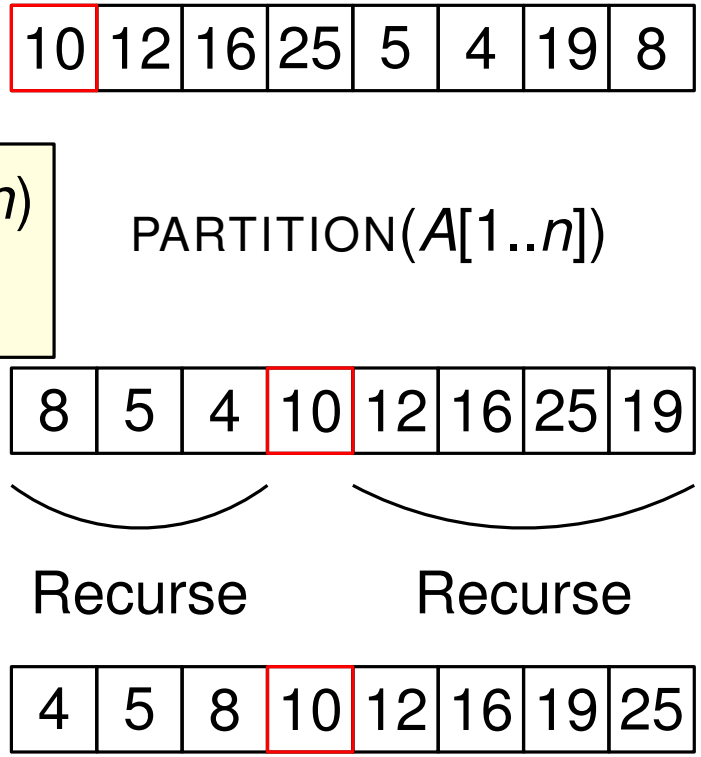
QUICKSORT($A[1..n]$)

procedure QUICKSORT($A[i..j]$)
  if $i < j$ **then**
    $pivot$ = RANDOM(i, j)
    SWAP($A[i]$, $A[pivot]$)
    k = PARTITION($A[i..j]$)
    in parallel do
      QUICKSORT($A[i..k-1]$)
      QUICKSORT($A[k+1..j]$)

$T(n) = O(1)$
$W(n) = O(1)$

$T(n) = O(\log n)$
$W(n) = O(n)$

PARTITION($A[1..n]$)



Analysis:

$O(\log n)$ recursive levels in expectation

Each level:

$T(n) = O(\log n)$
$W(n) = O(n)$

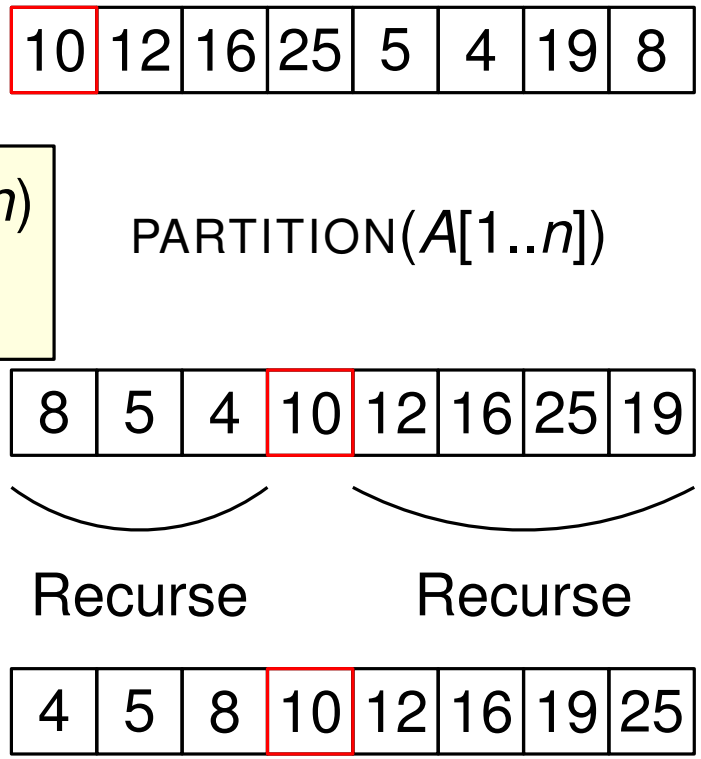
QUICKSORT($A[1..n]$)

```
procedure QUICKSORT(A[i..j])
  if i < j then
      pivot = RANDOM(i, j)
      SWAP(A[i], A[pivot])
      k = PARTITION(A[i..j])
      in parallel do
          QUICKSORT(A[i..k − 1])
          QUICKSORT(A[k + 1..j])
```

$T(n) = O(1)$
$W(n) = O(1)$

$T(n) = O(\log n)$
$W(n) = O(n)$

PARTITION($A[1..n]$)



Analysis:

$O(\log n)$ recursive levels in expectation

Each level:

$T(n) = O(\log n)$
$W(n) = O(n)$

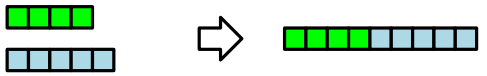
Expected

Total:

$T(n) = O(\log^2 n)$
$W(n) = O(n \log n)$

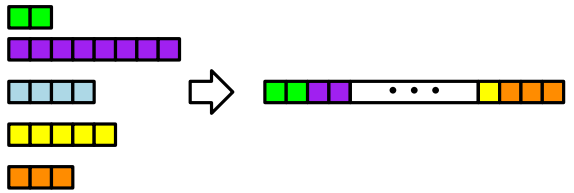
Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
    for $i = 1$ to k **in parallel do**
        $A[i] = B[i]$
    for $i = 1$ **to** m **in parallel do**
        $A[k+i] = C[i]$



Compact

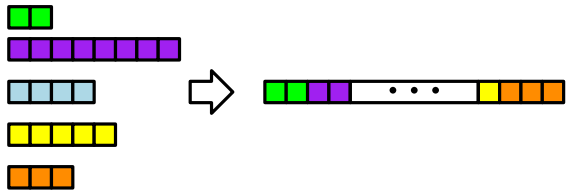
procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$



procedure COMPACT($A_1, A_2, \ldots, A_\ell, sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$

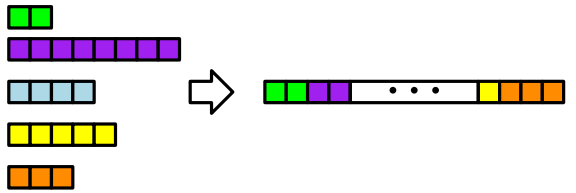


sizes : 2 8 4 5 3

procedure COMPACT($A_1, A_2, \ldots, A_\ell$, $sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$

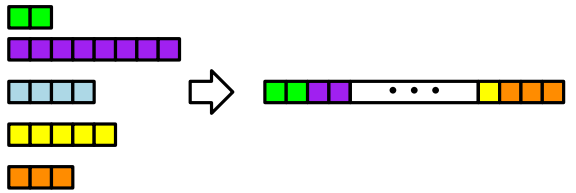


sizes : 2 8 4 5 3

procedure COMPACT($A_1, A_2, \ldots, A_\ell$, $sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$



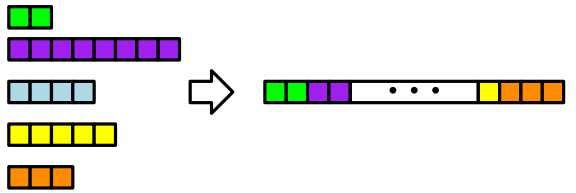
sizes : 0 2 8 4 5 3

procedure COMPACT($A_1, A_2, \ldots, A_\ell$, $sizes[1..\ell]$)
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)

▷ $n = \sum_{i=1}^{\ell} |A_i|$

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$

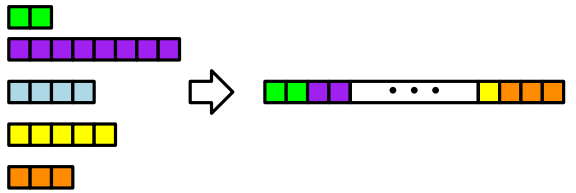


sizes : 0 2 10 14 19 22

procedure COMPACT($A_1, A_2, \ldots, A_\ell$, $sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$

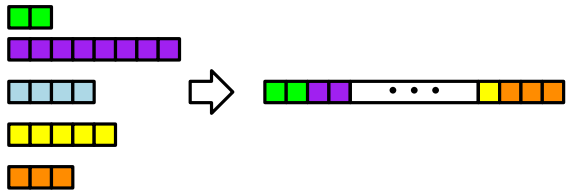


sizes : 0 2 10 14 19 22

procedure COMPACT($A_1, A_2, \ldots, A_\ell$, $sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)
  A = new array of size $n = sizes[\ell]$

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$

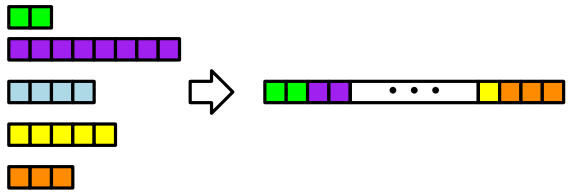


sizes : 0 2 10 14 19 22

procedure COMPACT($A_1, A_2, \ldots, A_\ell$, $sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)
  A = new array of size $n = sizes[\ell]$
  for $i = 1$ to ℓ **in parallel do**

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$

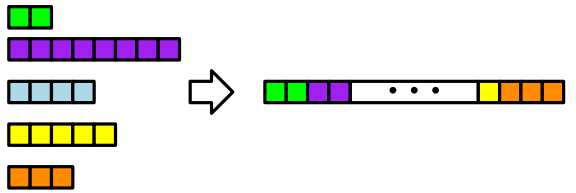


$sizes$: 0 2 10 14 19 22

procedure COMPACT($A_1, A_2, \ldots, A_\ell, sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)
  A = new array of size $n = sizes[\ell]$
  for $i = 1$ to ℓ **in parallel do**
    for $j = 1$ to $|A_i|$ **in parallel do**

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$

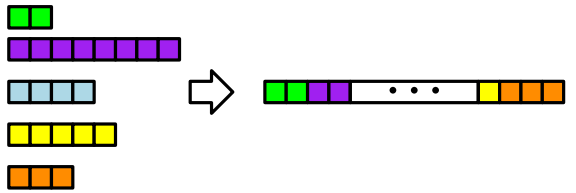


$sizes$: 0 2 10 14 19 22

procedure COMPACT($A_1, A_2, \ldots, A_\ell, sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)
  A = new array of size $n = sizes[\ell]$
  for $i = 1$ to ℓ **in parallel do**
    for $j = 1$ to $|A_i|$ **in parallel do**
      $A[sizes[i-1] + j] = A_i[j]$

Compact

```
procedure COMBINE(B[1..k], C[1..m], A[1..k + m])
    for i = 1 to k in parallel do
        A[i] = B[i]
    for i = 1 to m in parallel do
        A[k + i] = C[i]
```

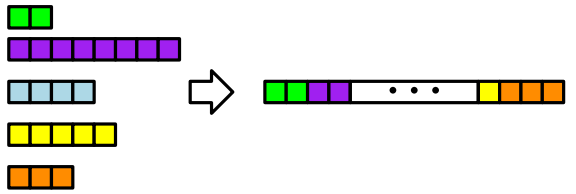


sizes : 0 2 10 14 19 22

```
procedure COMPACT(A_1, A_2, ..., A_ℓ, sizes[1..ℓ])          ▷ n = Σ_{i=1}^{ℓ} |A_i|
    sizes[0] = 0
    PREFIX-SUMS(sizes)
    A = new array of size n = sizes[ℓ]
    for i = 1 to ℓ in parallel do
        for j = 1 to |A_i| in parallel do
            A[sizes[i − 1] + j] = A_i[j]
    return A
```

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$



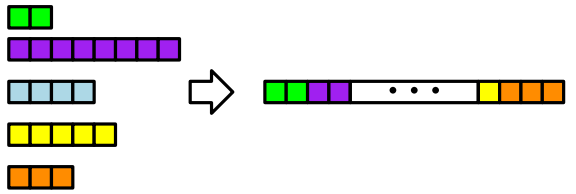
$sizes$: 0 2 10 14 19 22

procedure COMPACT($A_1, A_2, \ldots, A_\ell, sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)
  A = new array of size $n = sizes[\ell]$
  for $i = 1$ to ℓ **in parallel do**
    for $j = 1$ to $|A_i|$ **in parallel do**
      $A[sizes[i-1] + j] = A_i[j]$
  return A

$T(n) = O(\log \ell)$
$W(n) = O(\ell)$

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$



$sizes$: 0 2 10 14 19 22

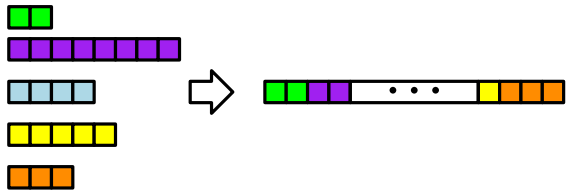
procedure COMPACT($A_1, A_2, \ldots, A_\ell, sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)
  A = new array of size $n = sizes[\ell]$
  for $i = 1$ to ℓ **in parallel do**
    for $j = 1$ to $|A_i|$ **in parallel do**
      $A[sizes[i-1] + j] = A_i[j]$
  return A

$T(n) = O(\log \ell)$
$W(n) = O(\ell)$

$T(n) = O(1)$
$W(n) = O(1)$

Compact

```
procedure COMBINE(B[1..k], C[1..m], A[1..k + m])
    for i = 1 to k in parallel do
        A[i] = B[i]
    for i = 1 to m in parallel do
        A[k + i] = C[i]
```



sizes : 0 2 10 14 19 22

```
procedure COMPACT(A_1, A_2, ..., A_ℓ, sizes[1..ℓ])        ▷ n = Σ_{i=1}^{ℓ} |A_i|
    sizes[0] = 0
    PREFIX-SUMS(sizes)
    A = new array of size n = sizes[ℓ]
    for i = 1 to ℓ in parallel do
        for j = 1 to |A_i| in parallel do
            A[sizes[i − 1] + j] = A_i[j]
    return A
```

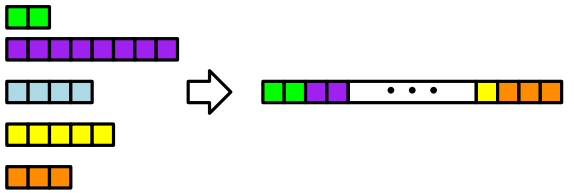
$T(n) = O(\log \ell)$
$W(n) = O(\ell)$

$T(n) = O(1)$
$W(n) = O(1)$

$T(n) = O(1)$
$W(n) = O(n)$

Compact

procedure COMBINE($B[1..k]$, $C[1..m]$, $A[1..k+m]$)
  for $i = 1$ to k **in parallel do**
    $A[i] = B[i]$
  for $i = 1$ **to** m **in parallel do**
    $A[k+i] = C[i]$



$sizes$: 0 2 10 14 19 22

procedure COMPACT($A_1, A_2, \ldots, A_\ell, sizes[1..\ell]$) ▷ $n = \sum_{i=1}^{\ell} |A_i|$
  $sizes[0] = 0$
  PREFIX-SUMS(sizes)
  A = new array of size $n = sizes[\ell]$
  for $i = 1$ to ℓ **in parallel do**
    for $j = 1$ to $|A_i|$ **in paralle**
      $A[sizes[i-1]+j] = A_i[$
  return A

$T(n) = O(\log \ell)$
$W(n) = O(\ell)$

$T(n) = O(1)$
$W(n) = O(1)$

$T(n) = O(1)$
$W(n) = O(n)$

Total
$T(n) = O(\log \ell)$
$W(n) = O(n)$

Summary

- FILTER
- BROADCAST (PREFIX-COPY)
- PARTITION
- QUICKSORT
- COMPACT