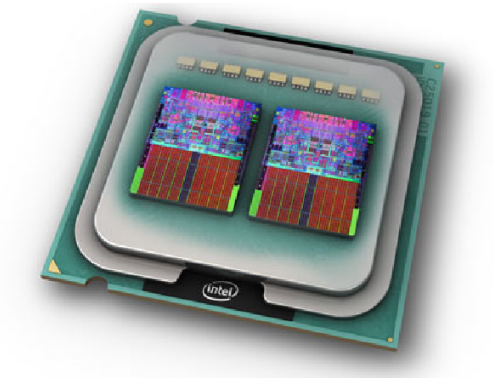
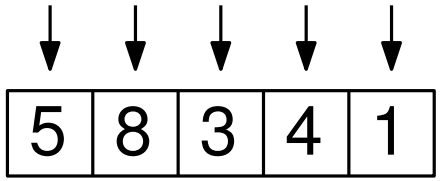
ICS 621: Analysis of Algorithms

Prof. Nodari Sitchinava



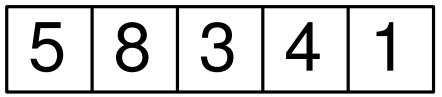


Parallel Algorithms

Course Evaluations

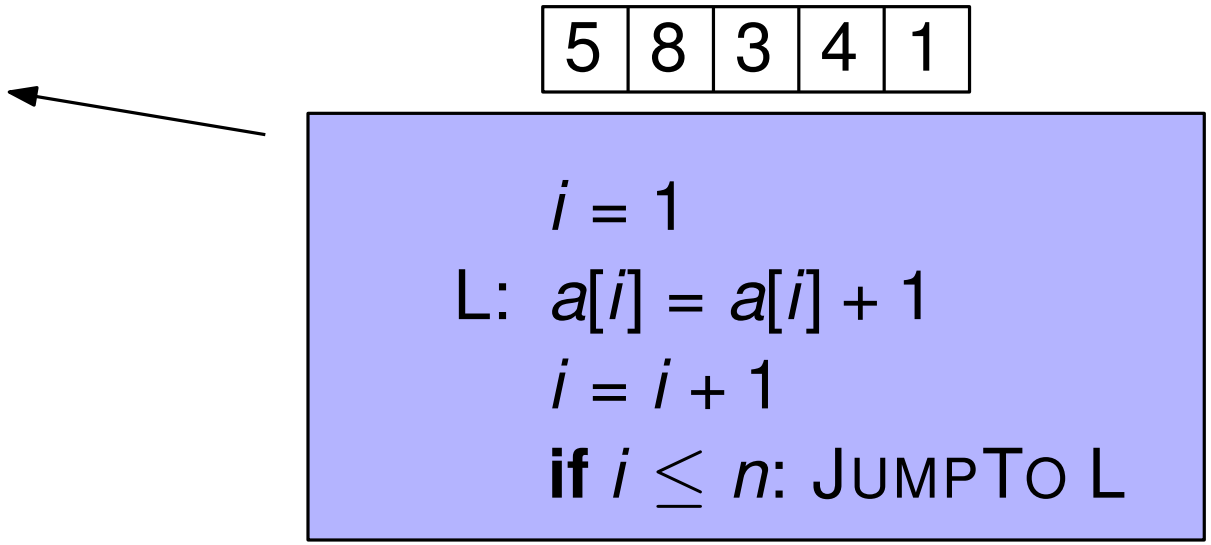
Simple example

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



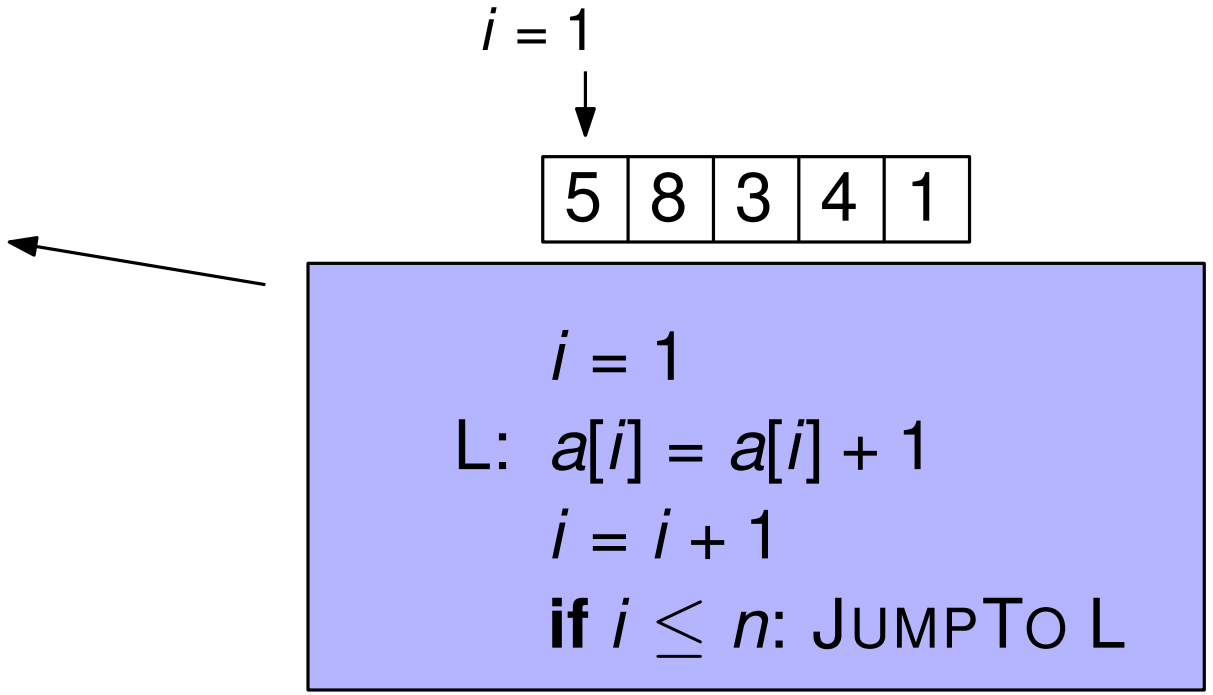
Simple example

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



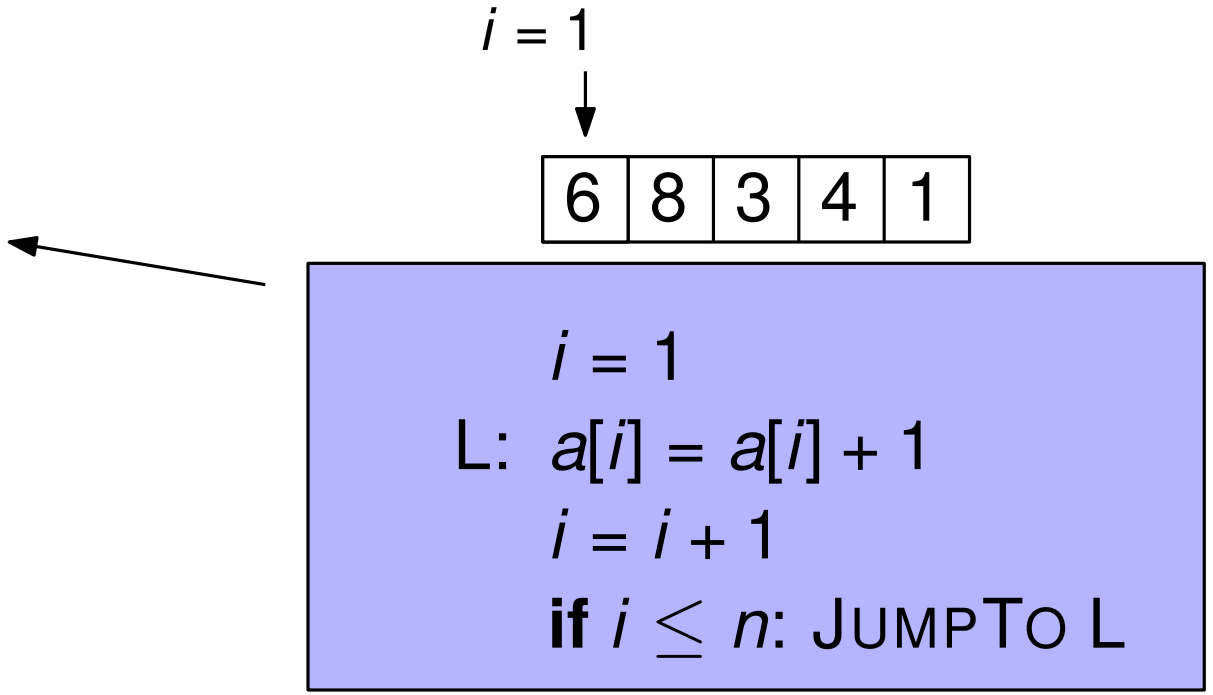
Simple example

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



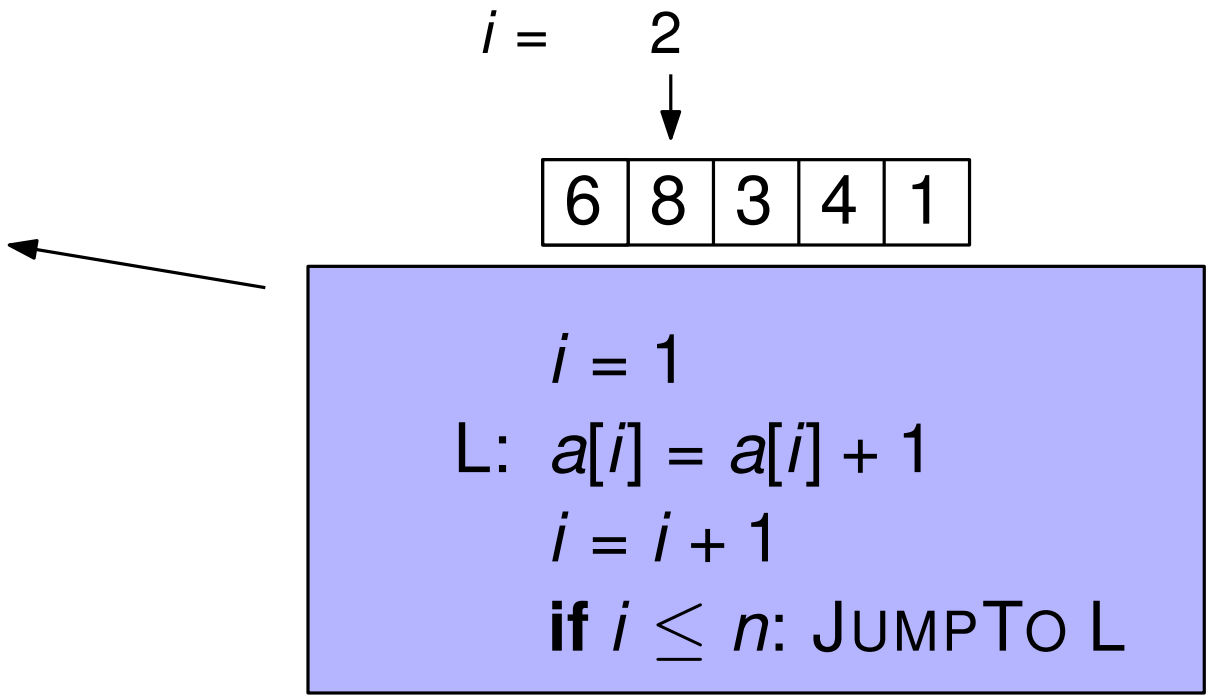
Simple example

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



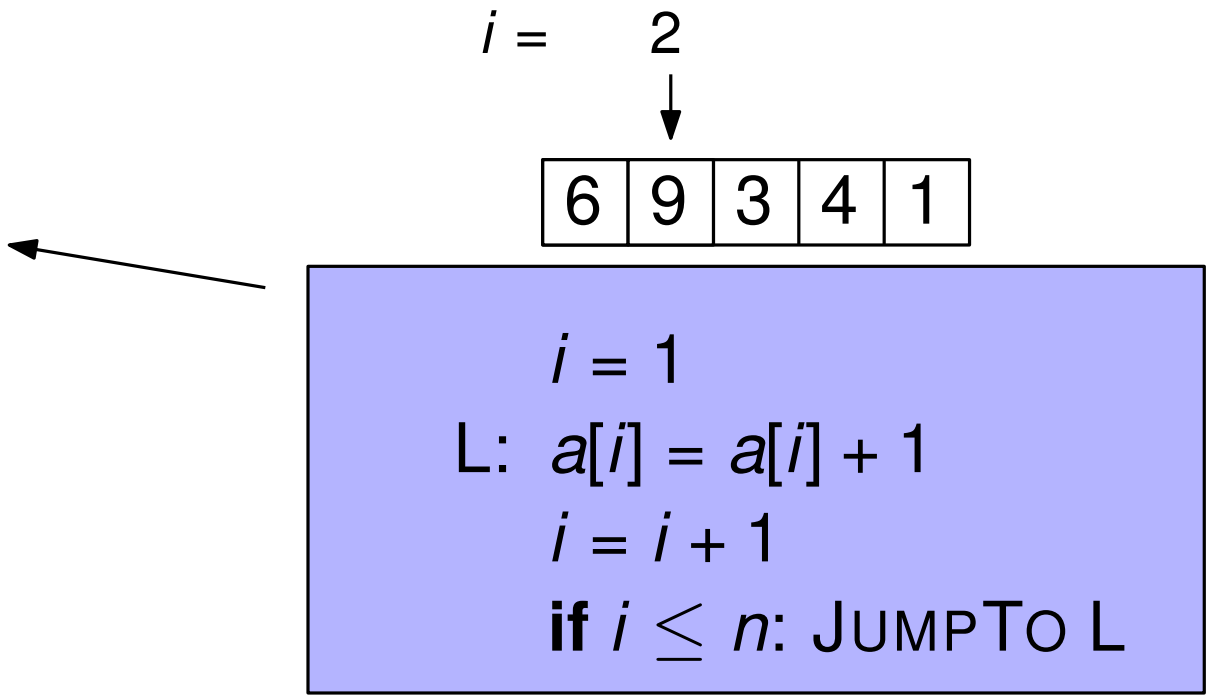
Simple example

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



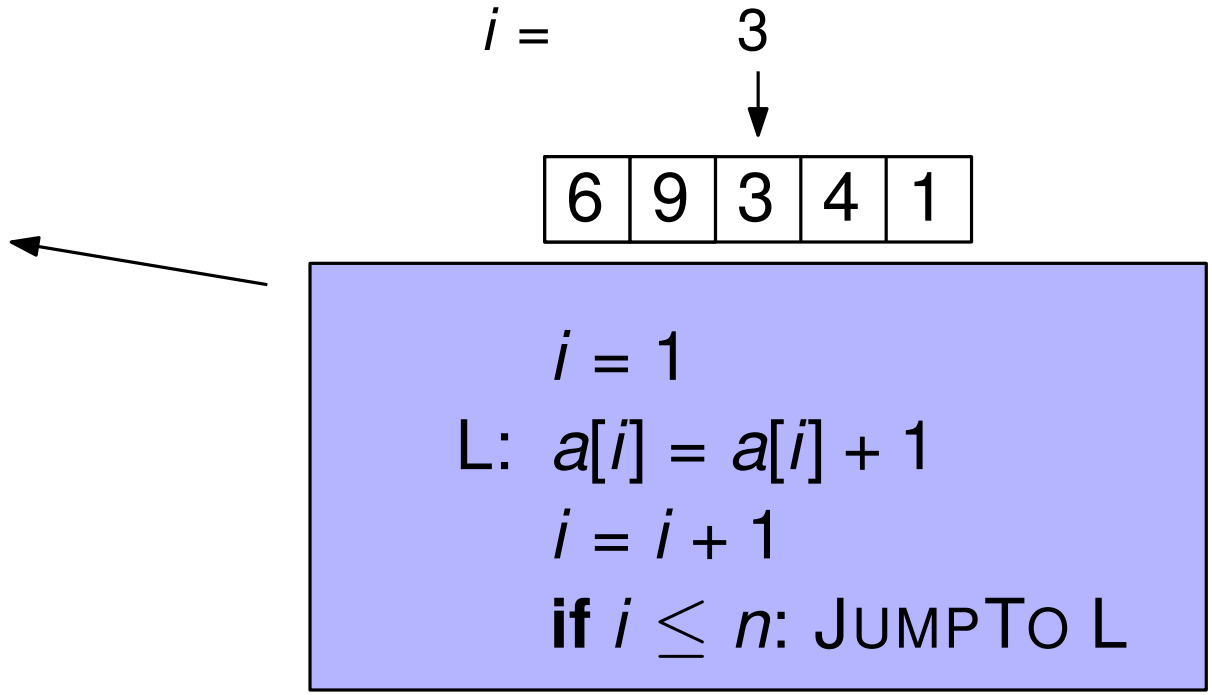
Simple example

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



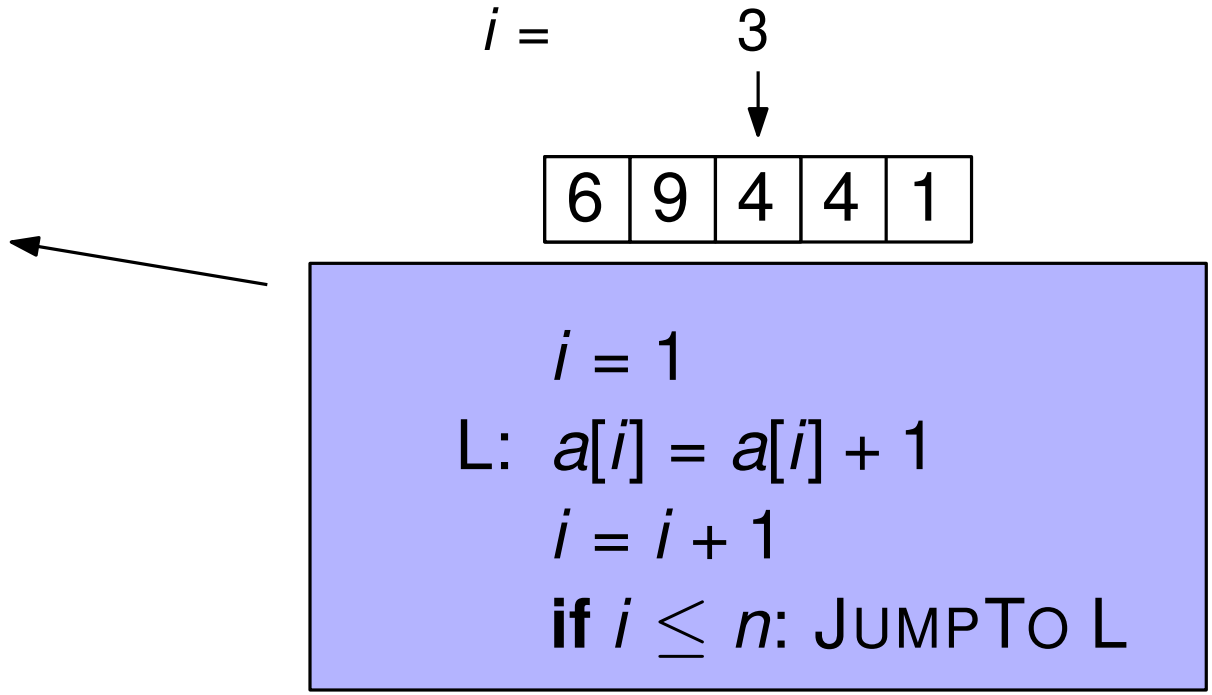
Simple example

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



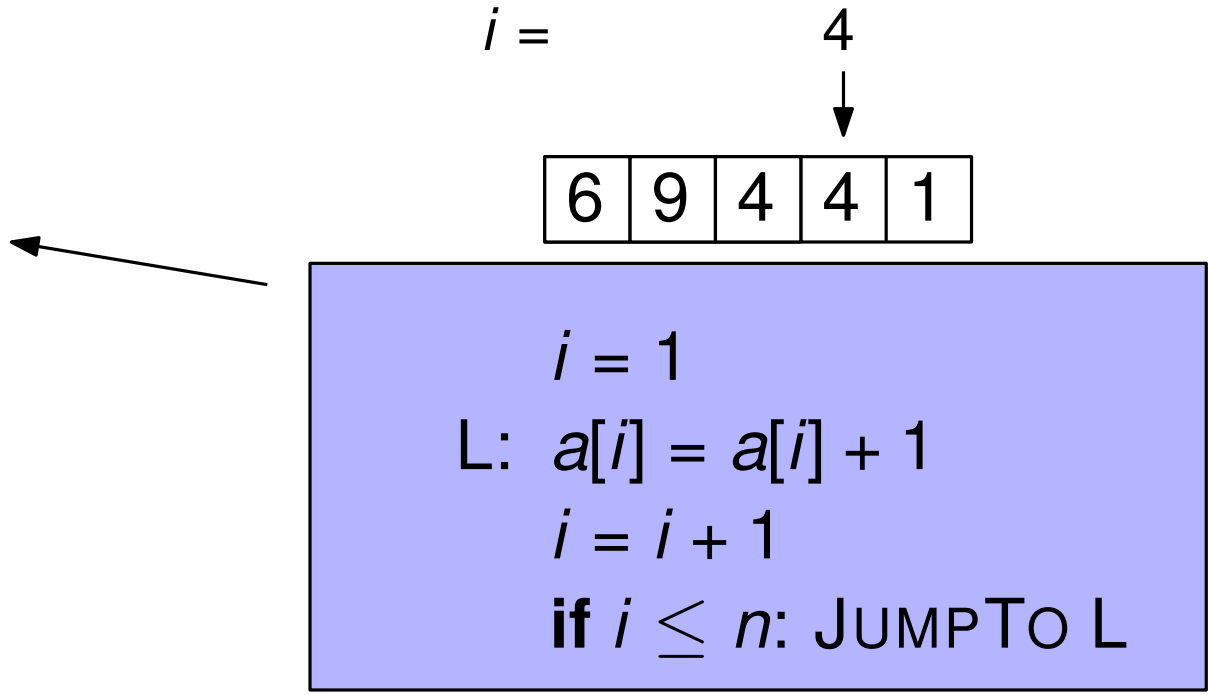
Simple example

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



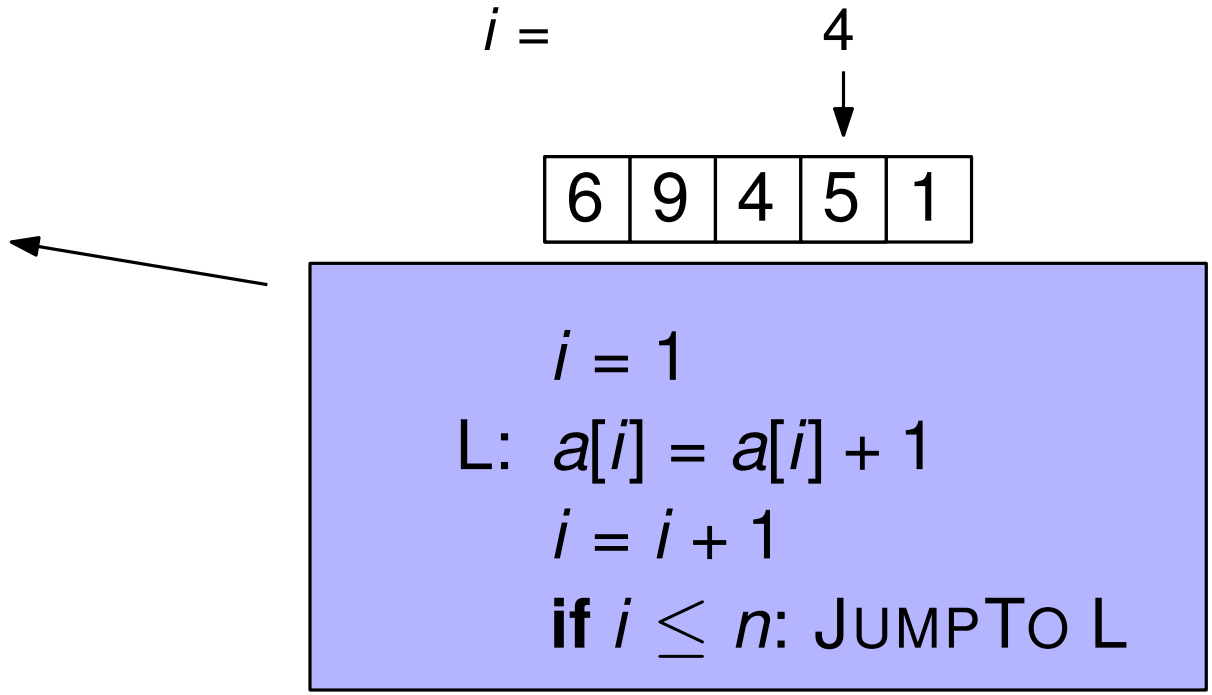
Simple example

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



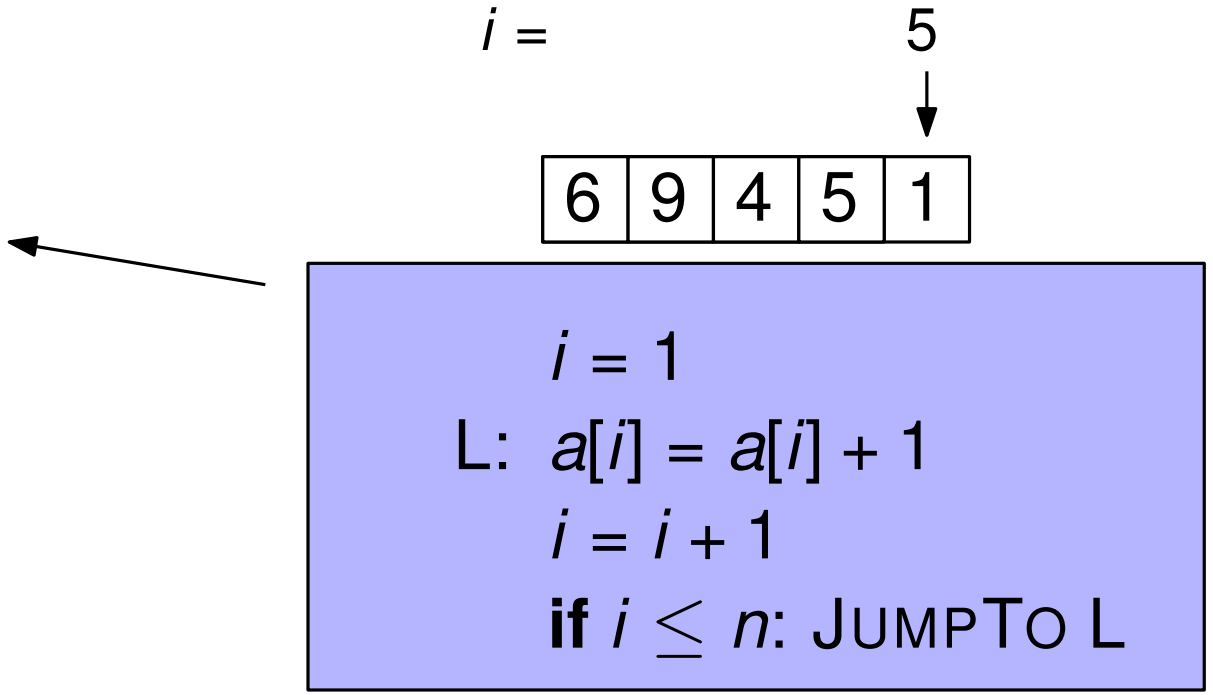
Simple example

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



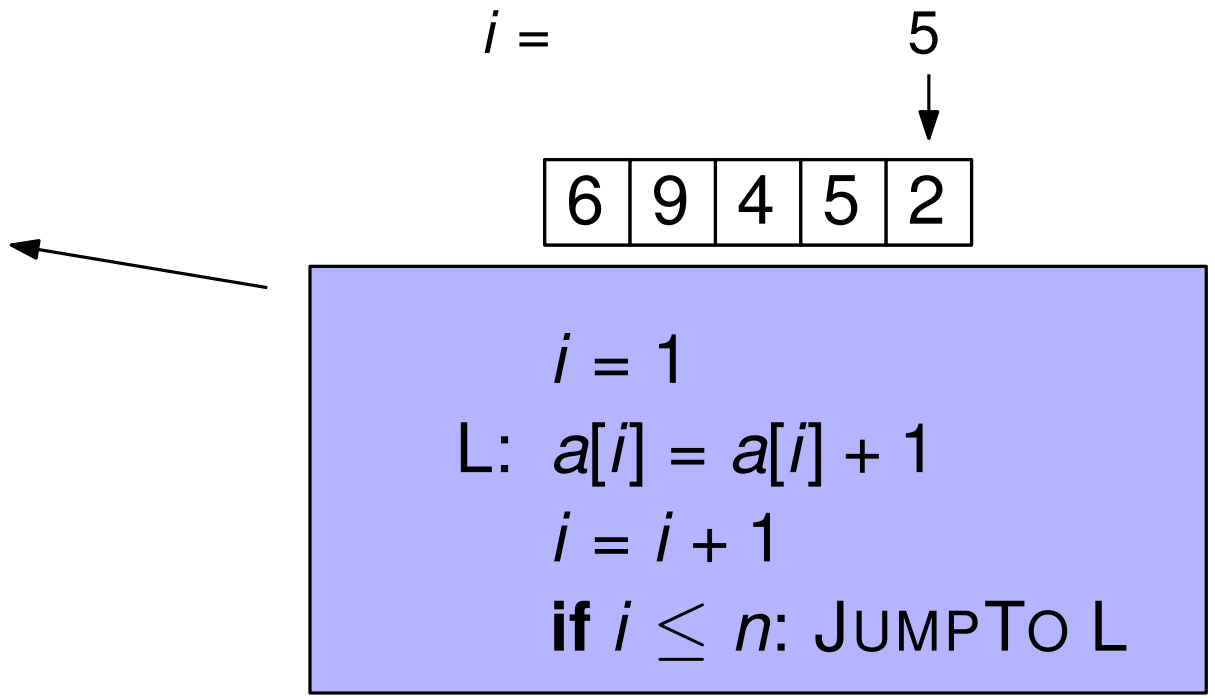
Simple example

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



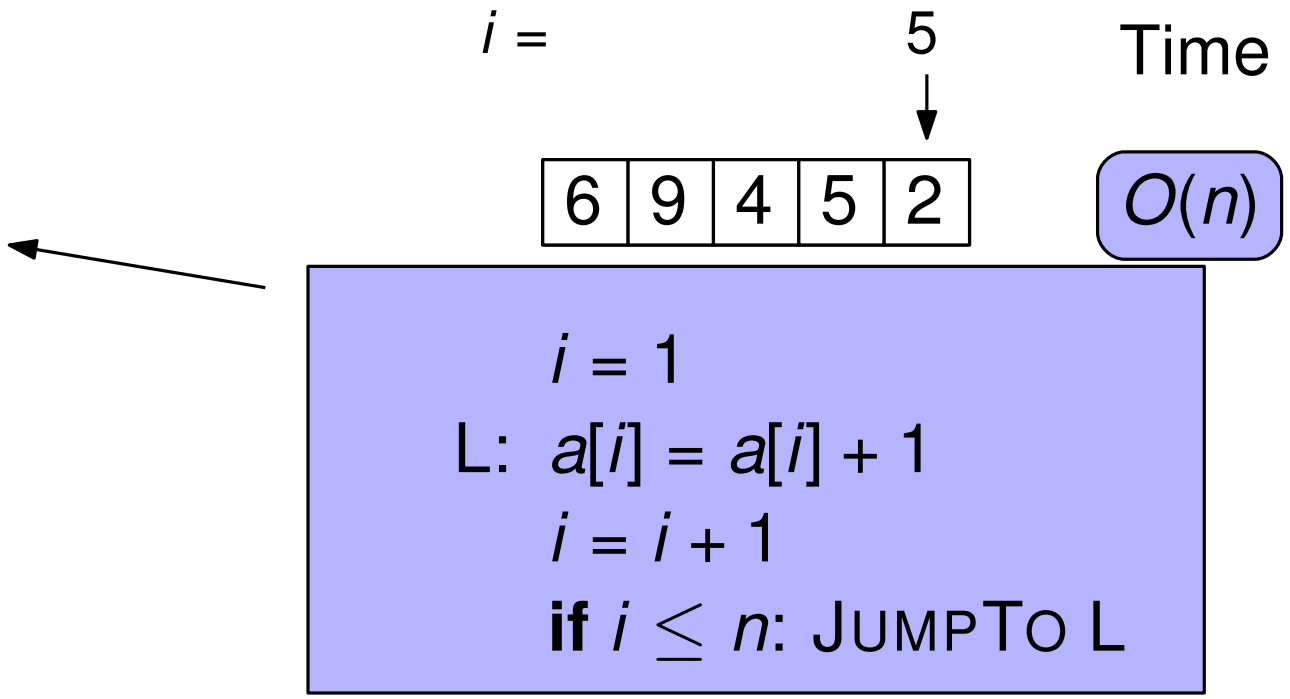
Simple example

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



Simple example

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

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

6	9	4	5	2

$O(n)$

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

5	8	3	4	1

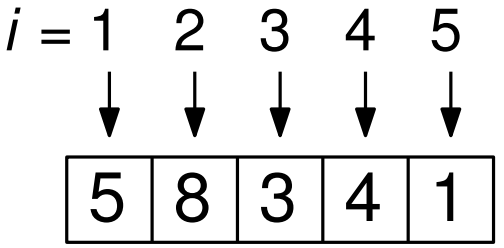
Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Simple example

Time

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

$O(n)$



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

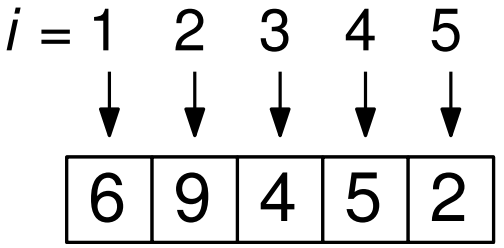
Start n threads $t_1, t_2, \ldots t_n$
Each thread t_i (where $i = 1, 2, \ldots, n$) **do**:
    $a[i] = a[i] + 1$

Simple example

Time

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

$O(n)$



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

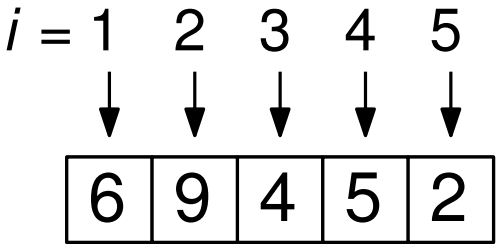
Simple example

Time

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

$O(n)$

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



$O(1)$

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

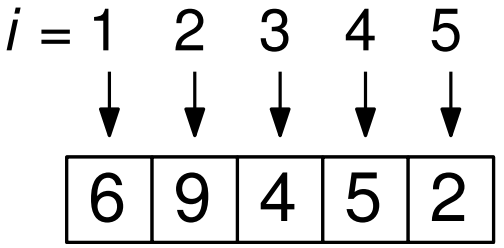
Simple example

Time

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

$O(n)$

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

Parallel Time = time of the slowest thread

More complex example: Prefix Sums

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

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

return $a[n]$

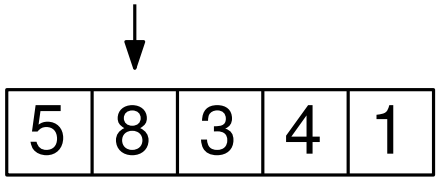
5	8	3	4	1

More complex example: Prefix Sums

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

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

return $a[n]$

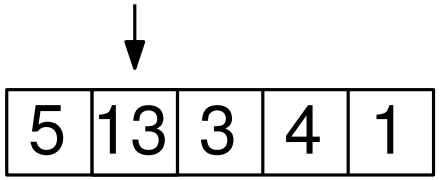


More complex example: Prefix Sums

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

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

return $a[n]$

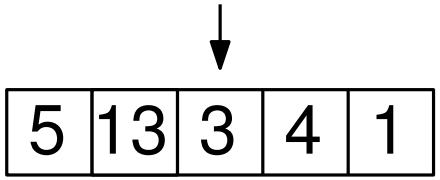


More complex example: Prefix Sums

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

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

return $a[n]$

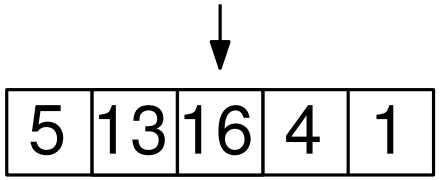


More complex example: Prefix Sums

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

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

return $a[n]$

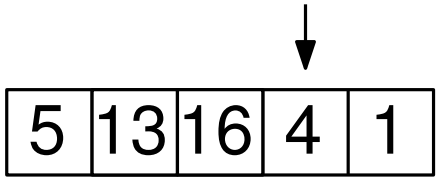


More complex example: Prefix Sums

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

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

return $a[n]$

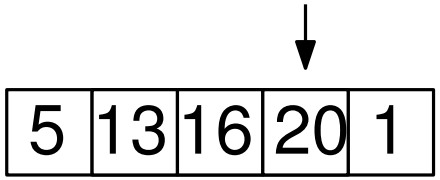


More complex example: Prefix Sums

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

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

return $a[n]$

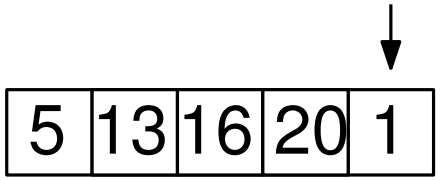


More complex example: Prefix Sums

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

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

return $a[n]$

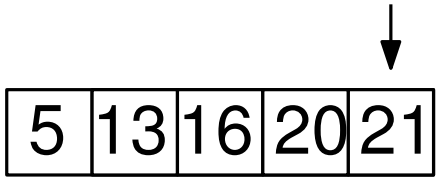


More complex example: Prefix Sums

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

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

return $a[n]$



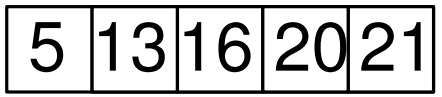
More complex example: Prefix Sums

Time

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

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

return $a[n]$



$O(n)$

More complex 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

$O(n)$

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

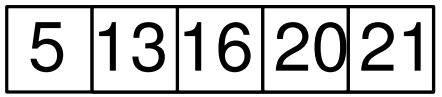
5	8	3	4	1

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

More complex example: Prefix Sums

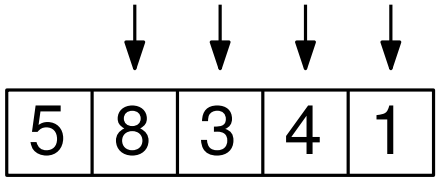
Time

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



$O(n)$

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

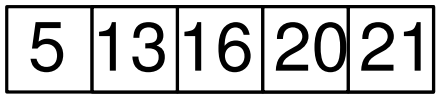


Start $n-1$ threads $t_2, \ldots t_n$
Each thread t_i (where $i = 2, \ldots, n$) **do**:
    $a[i] = a[i] + a[i-1]$

More complex example: Prefix Sums

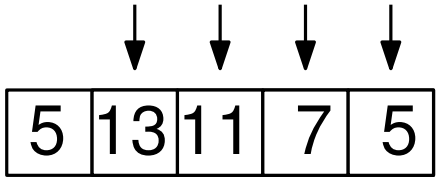
Time

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



$O(n)$

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

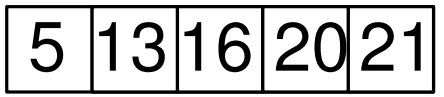


Start $n-1$ threads $t_2, \ldots t_n$
Each thread t_i (where $i = 2, \ldots, n$) **do**:
    $a[i] = a[i] + a[i-1]$

More complex example: Prefix Sums

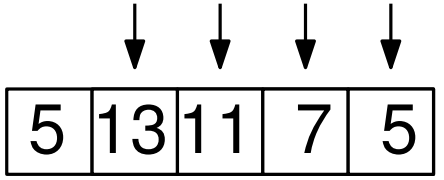
Time

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



$O(n)$

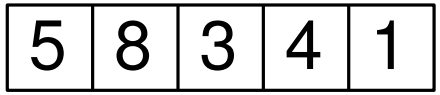
for $i = 2$ to n **in parallel do**
    $a[i] = a[i] + a[i-1]$
return $a[n]$



$O(1)$

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

Parallel Prefix Sums



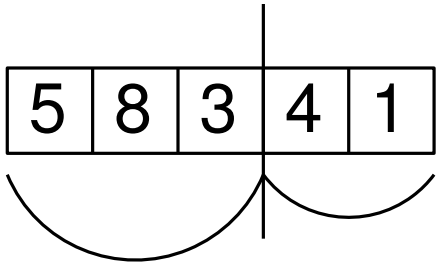
Parallel Prefix Sums

5	8	3	4	1

Parallel Prefix Sums

5	8	3	4	1

Parallel Prefix Sums



Parallel Prefix Sums

5	8	3	4	1

5	8	3

4	1

Parallel Prefix Sums

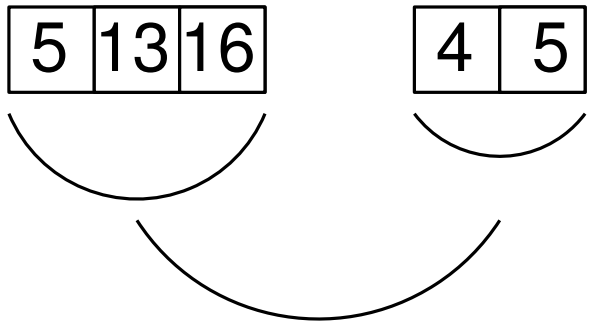
5	8	3	4	1

5	13	16

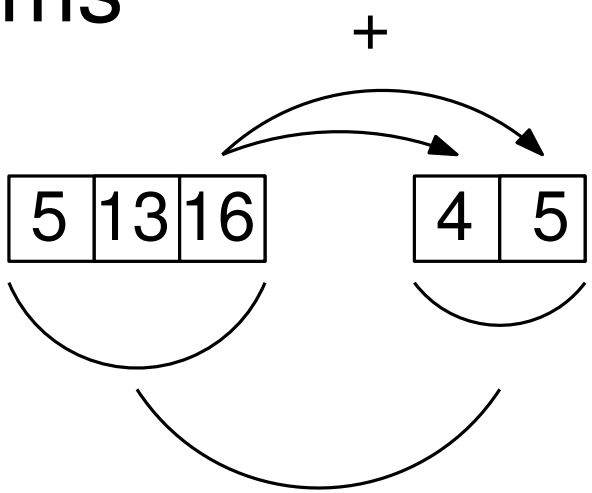
4	5

Parallel Prefix Sums

5	8	3	4	1

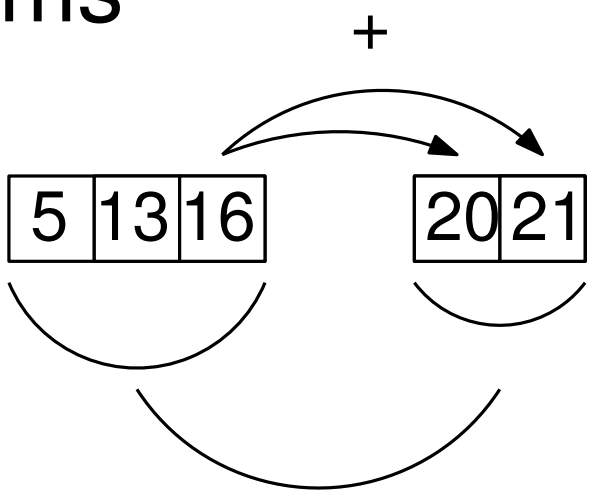


Parallel Prefix Sums



5	8	3	4	1

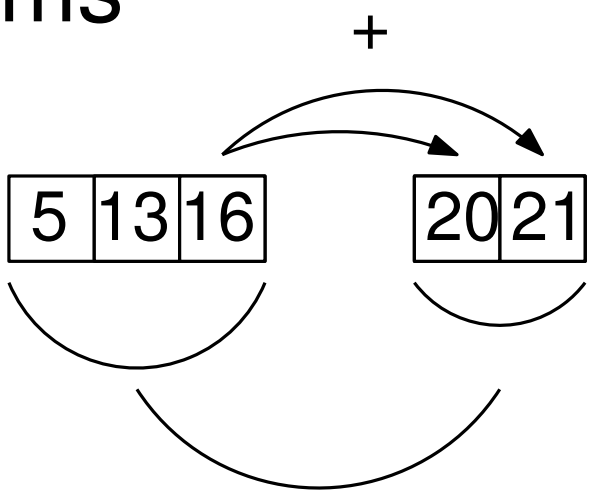
Parallel Prefix Sums



5	8	3	4	1

Parallel Prefix Sums

5	8	3	4	1



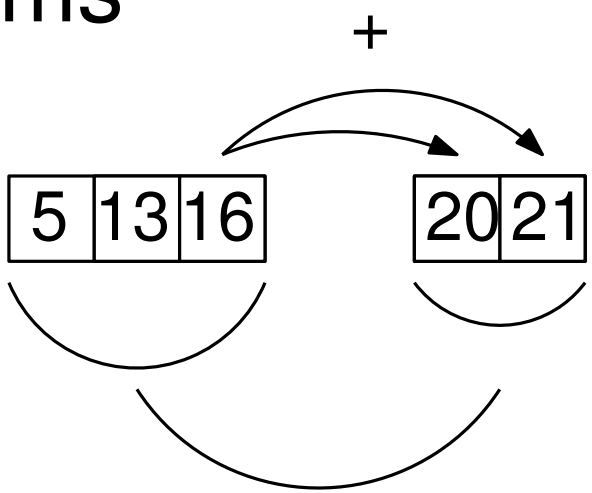
function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$

  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

5	8	3	4	1



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

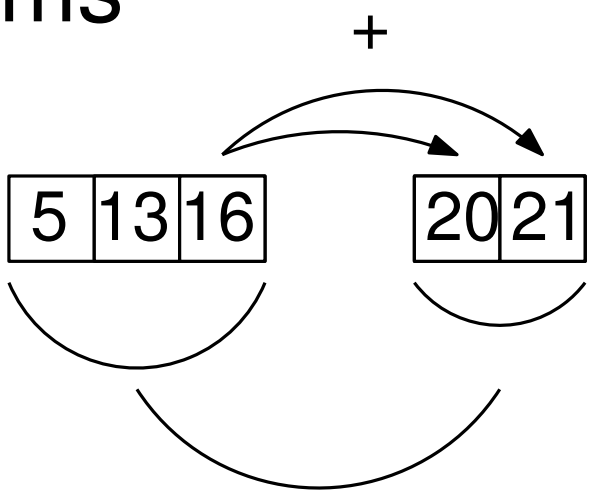
  PREFIX-SUMS(A, i, mid)
  PREFIX-SUMS($A, mid + 1, j$)

  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

5	8	3	4	1



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($A, mid + 1, j$)

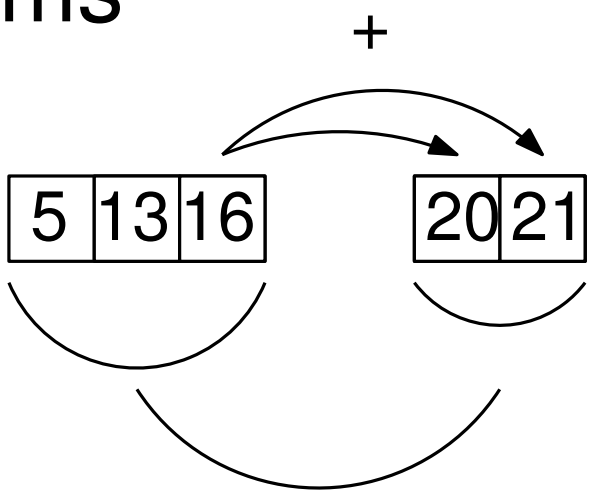
  for $k = mid + 1$ to j **in parallel do**

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

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

Parallel Prefix Sums

5	8	3	4	1



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($A, mid + 1, j$)

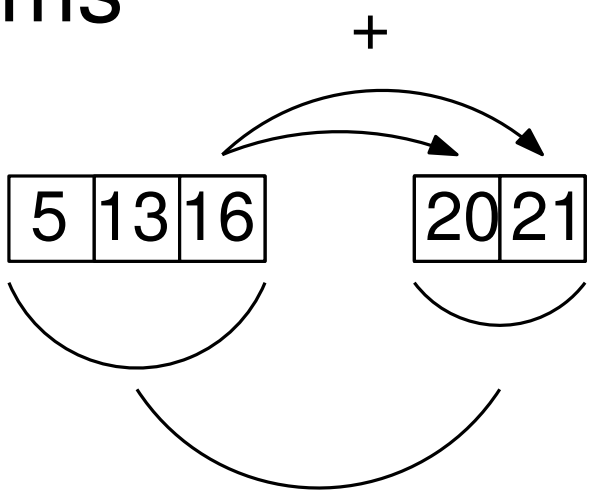
  for $k = mid + 1$ to j **in parallel do**

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

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

Parallel Prefix Sums

5	8	3	4	1

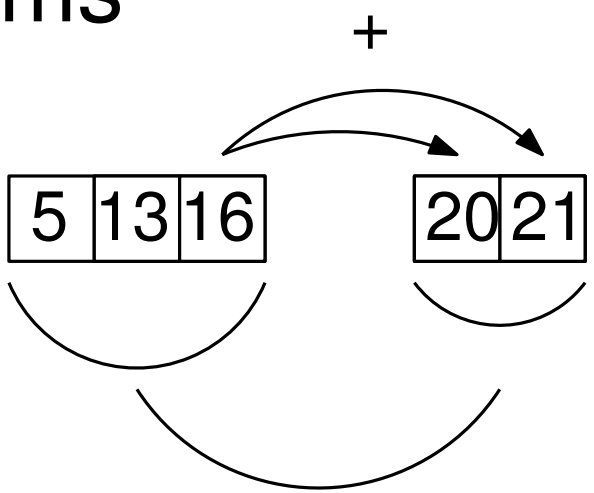


function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return** ▷ Base case
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS($A, mid + 1, j$)
  sync
  for $k = mid + 1$ to j **in parallel** **do**
    $A[k] = A[k] + A[mid]$

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

Parallel Prefix Sums

5	8	3	4	1



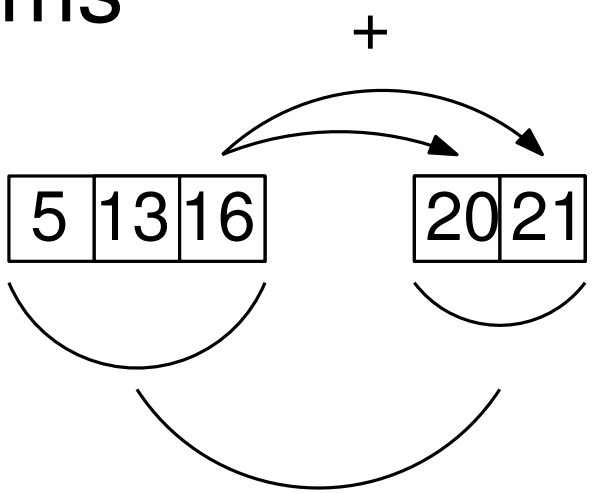
function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return**
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS($A, mid + 1, j$)
  sync
  for $k = mid + 1$ to j **in parallel do**
    $A[k] = A[k] + A[mid]$

(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)

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

Parallel Prefix Sums

5	8	3	4	1



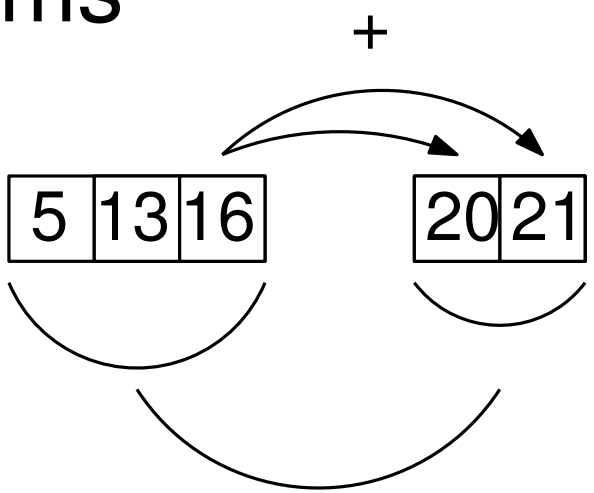
function PREFIX-SUMS(A, i, j)
  if $i \geq j$ **then return**
  $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$
  spawn
    PREFIX-SUMS(A, i, mid)
    PREFIX-SUMS($A, mid + 1, j$)
  sync
  for $k = mid + 1$ to j **in parallel**
    $A[k] = A[k] + A[mid]$

(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)

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

Parallel Prefix Sums

5	8	3	4	1



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

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

Recursion vs. parallel **for** loop

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** ▷ 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)

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

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)

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

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)

$$T(n) = \_T(n/2) + T_{\text{MERGE}}$$
$$= \_T(n/2) + O(n)$$

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)

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

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)

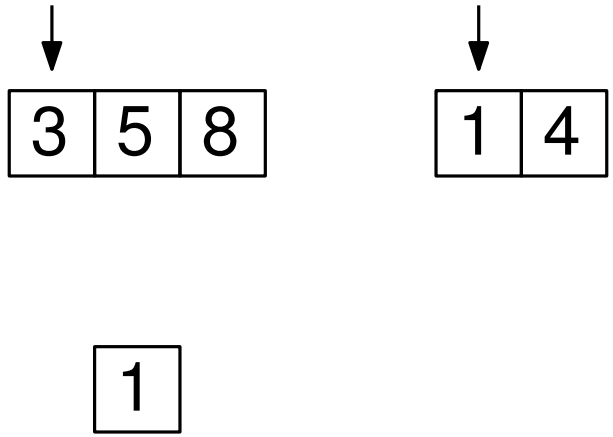
$$T(n) = T(n/2) + T_{\text{MERGE}}$$
$$= T(n/2) + O(\log n)$$
$$= O(\log^2 n)$$

With parallel merging

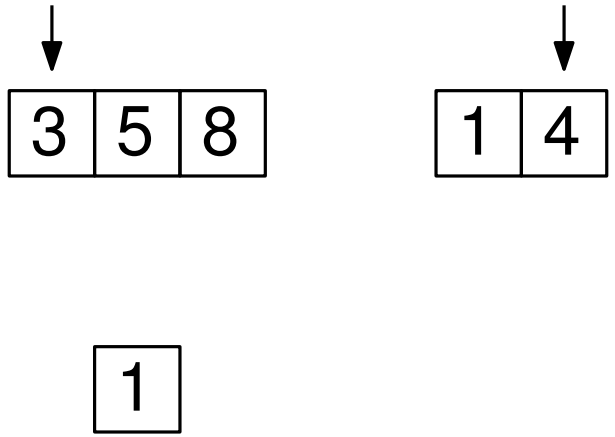
Parallel Merging

Parallel Merging

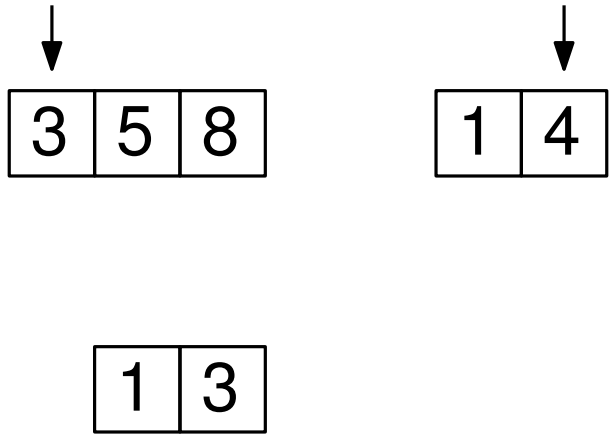
Parallel Merging



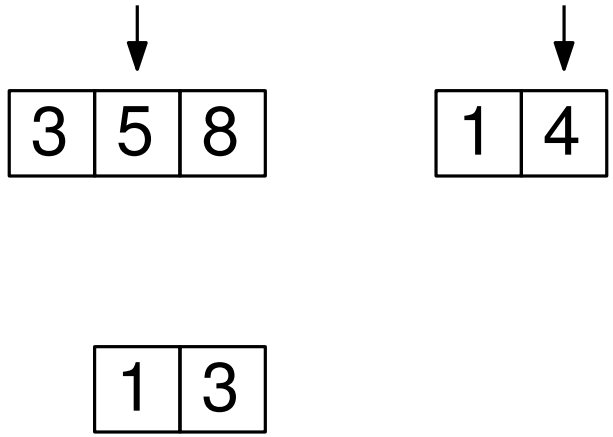
Parallel Merging



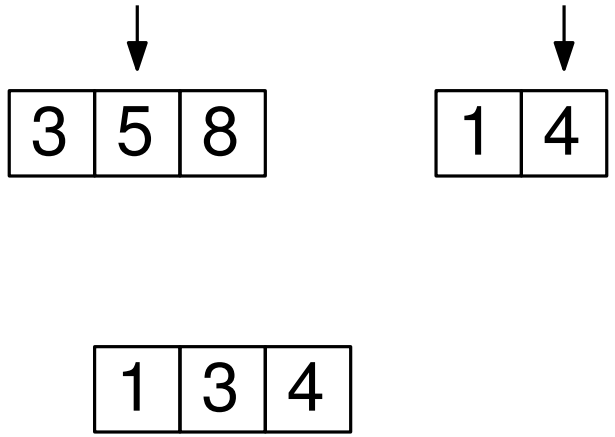
Parallel Merging



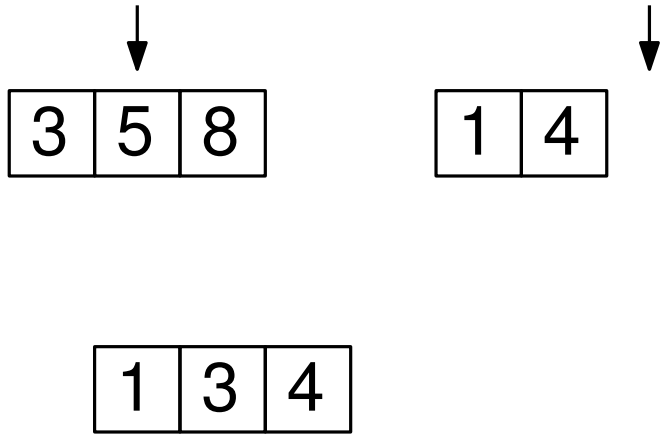
Parallel Merging



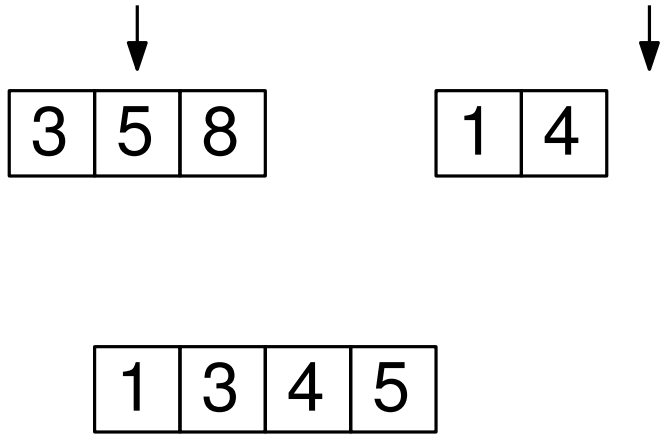
Parallel Merging



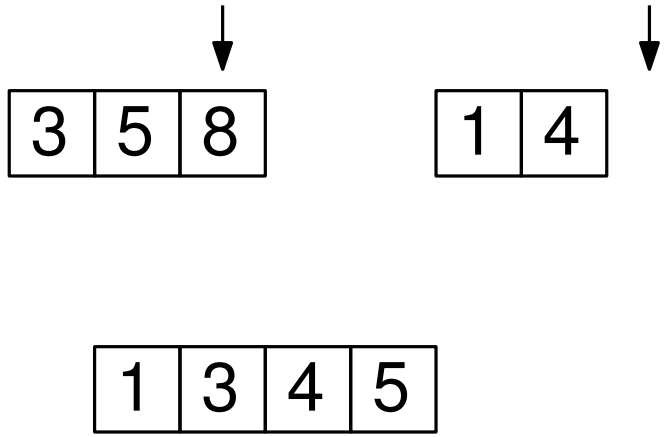
Parallel Merging



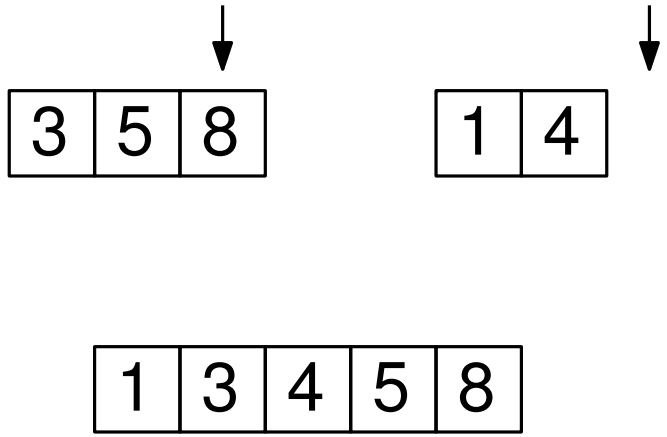
Parallel Merging



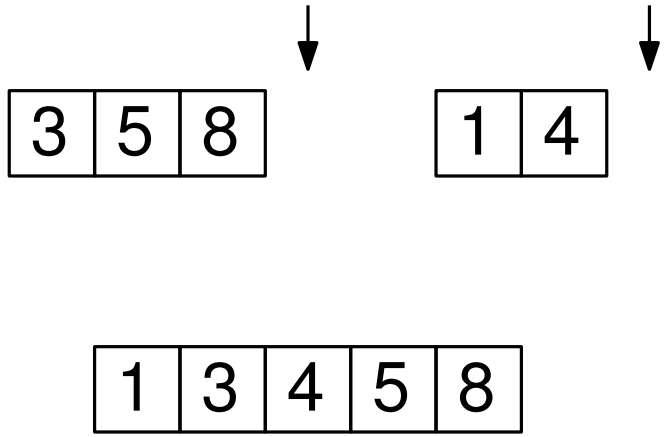
Parallel Merging



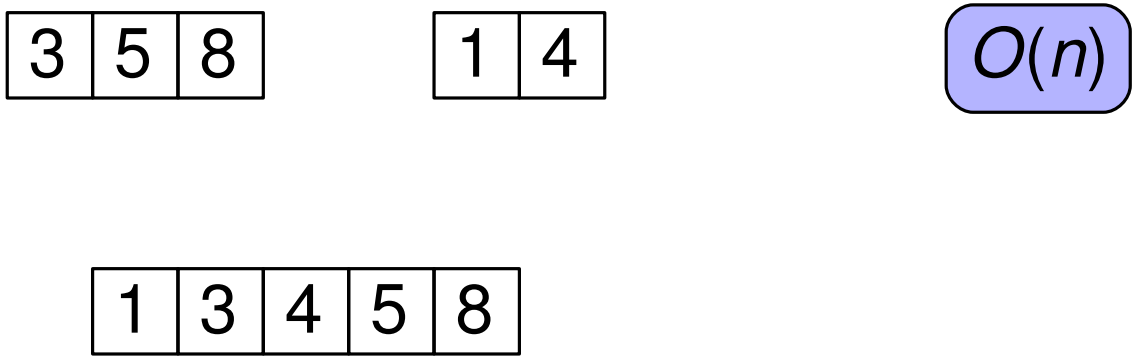
Parallel Merging



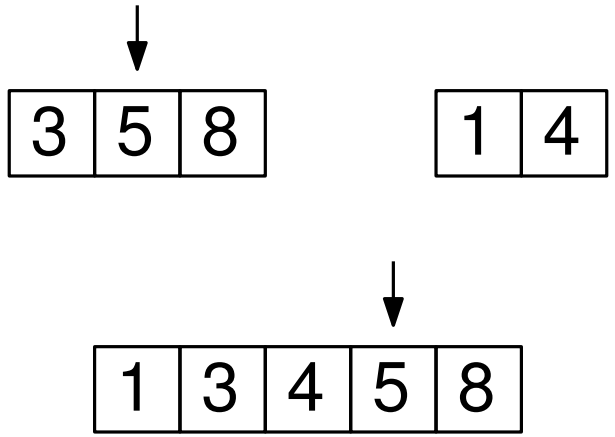
Parallel Merging



Parallel Merging

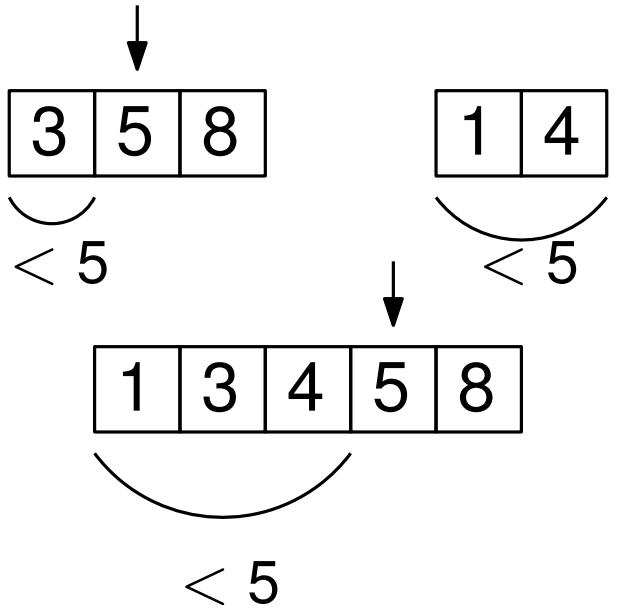


Parallel Merging



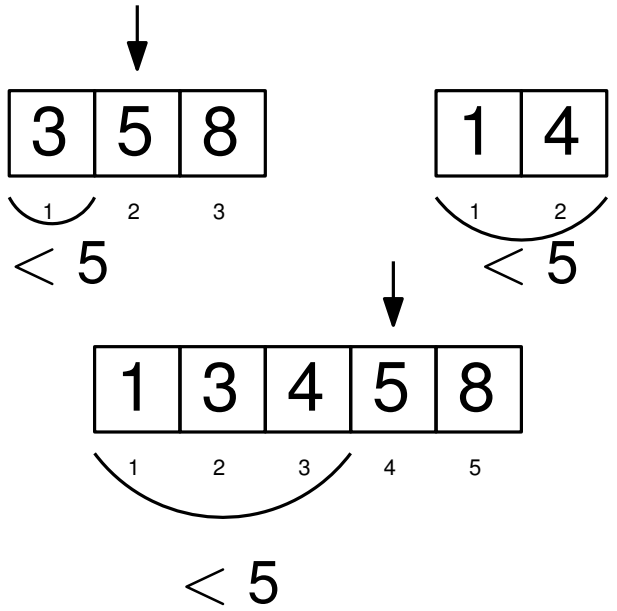
$O(n)$

Parallel Merging



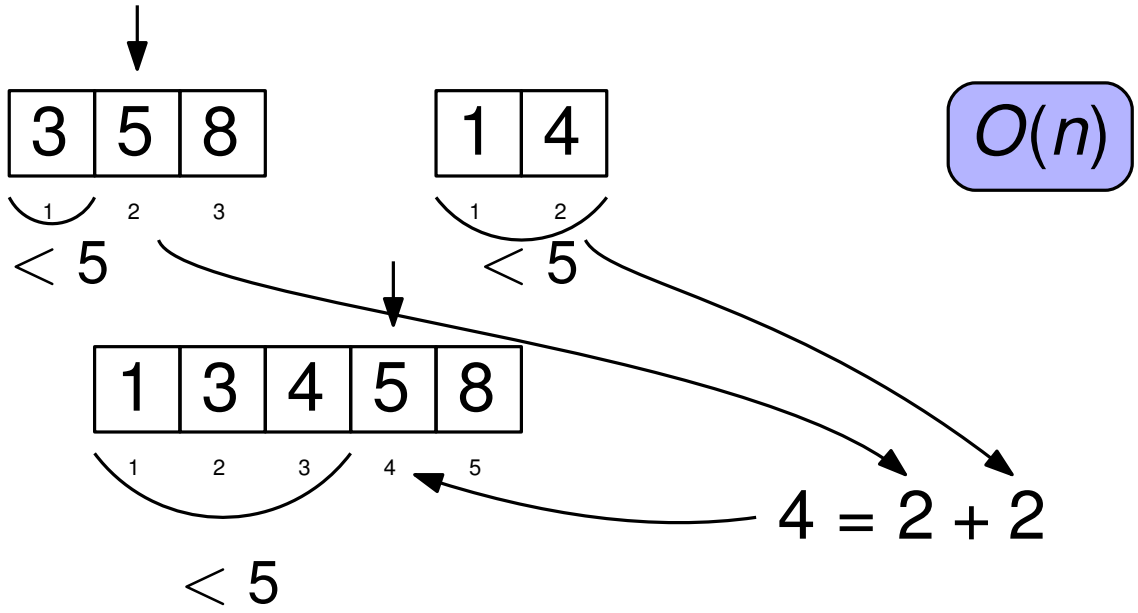
$O(n)$

Parallel Merging

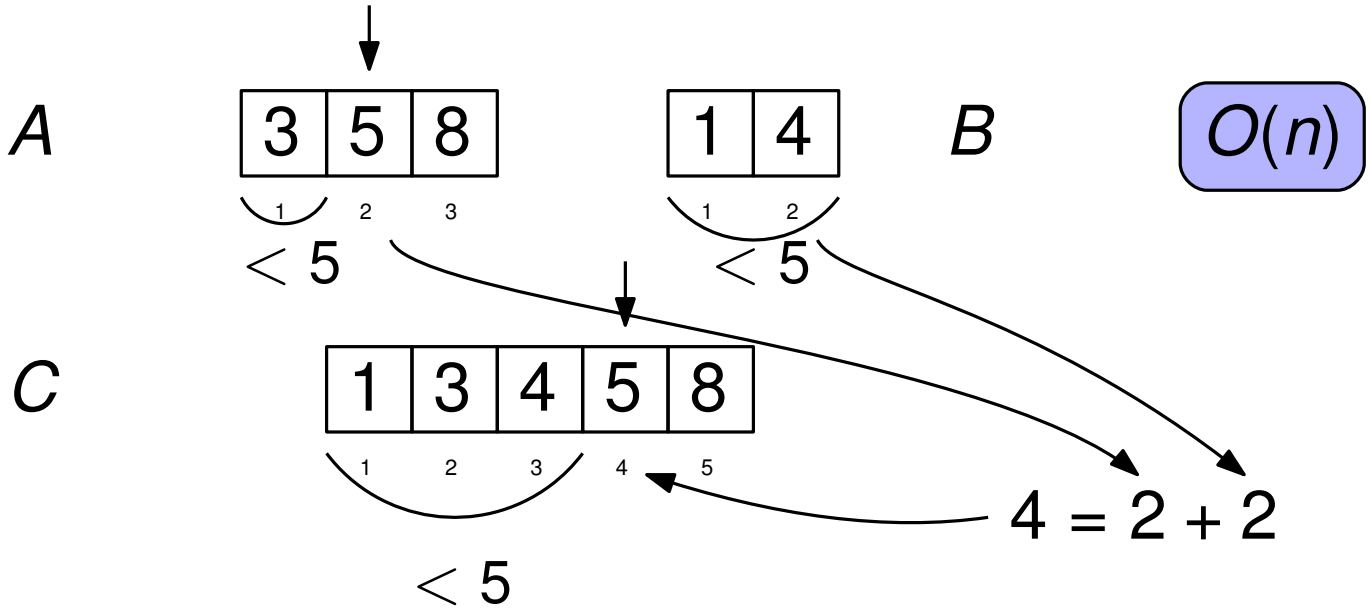


$O(n)$

Parallel Merging

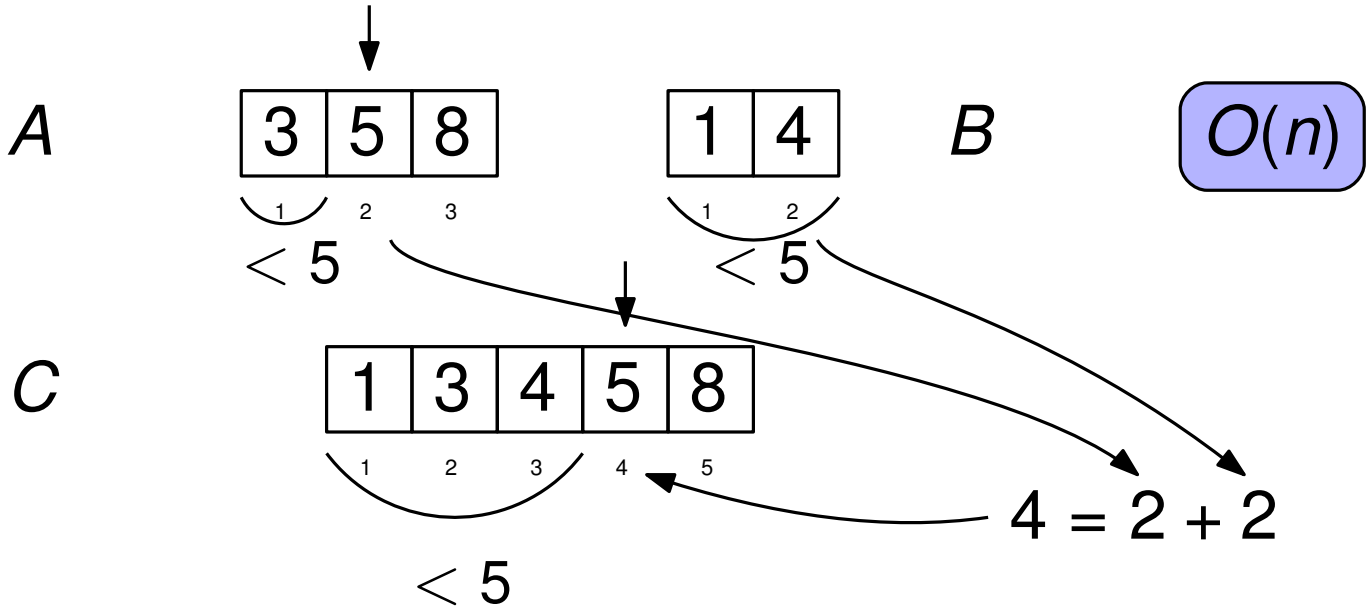


Parallel Merging



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

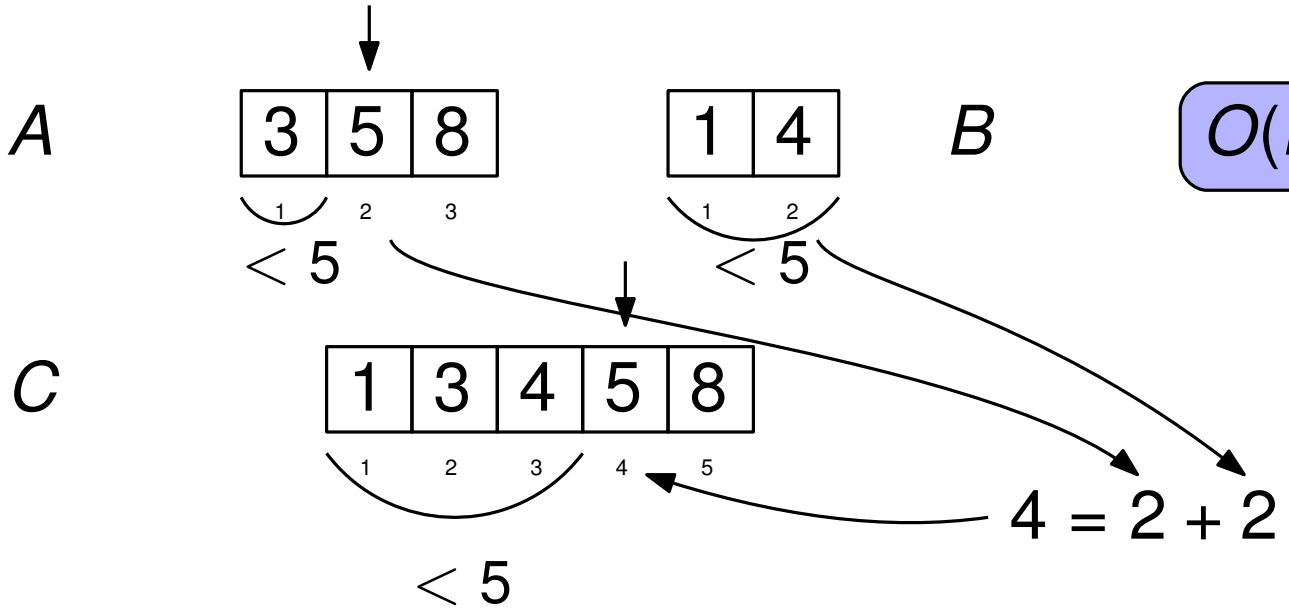
Parallel Merging



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

Parallel Merging

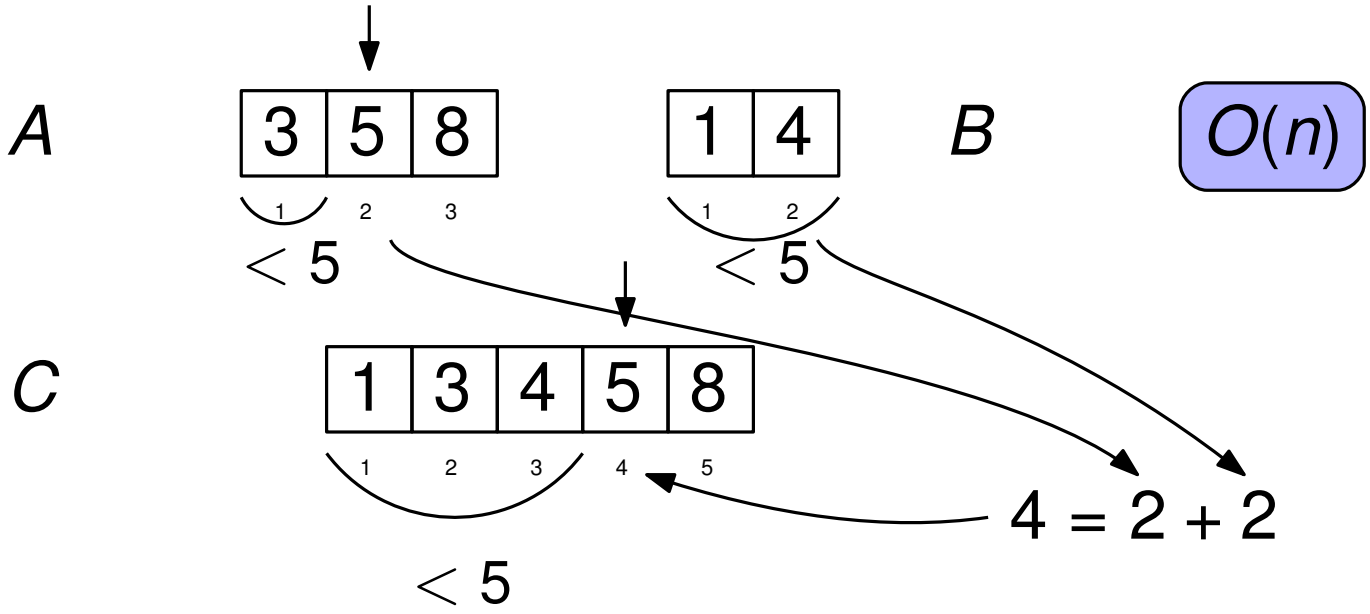


function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i +$ PREDECESSOR$(A[i], B)$
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j +$ PREDECESSOR$(B[j], A)$
    $C[k] = B[j]$

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

Still $O(n)$

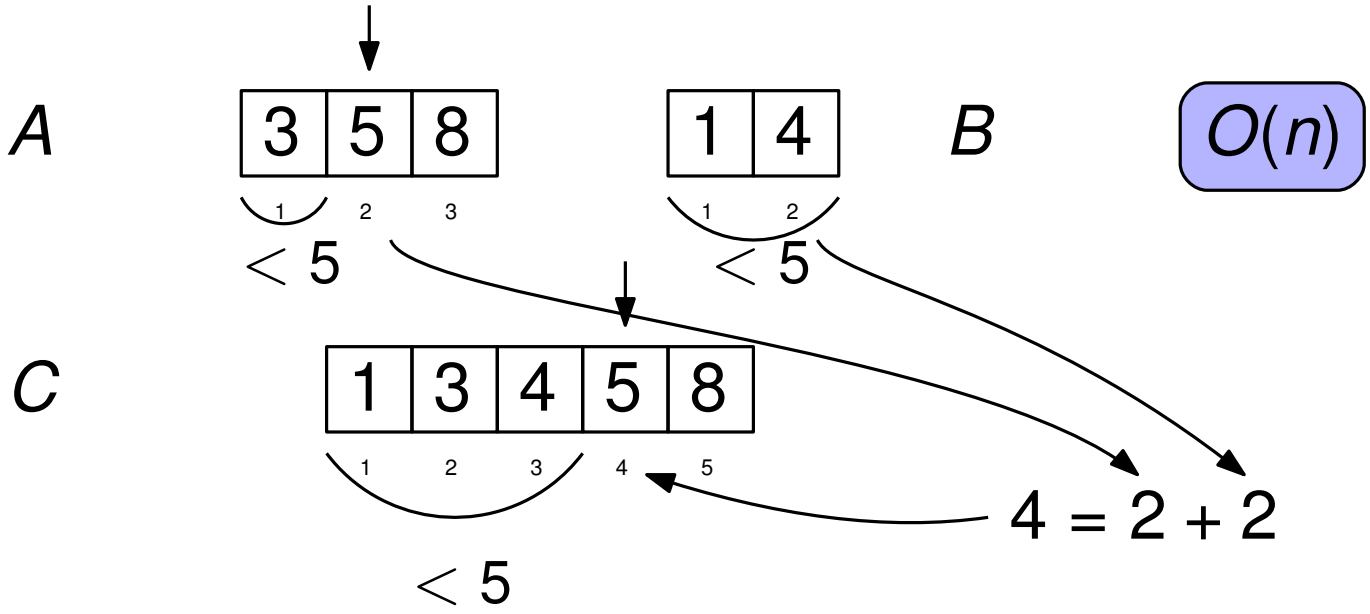
Parallel Merging



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

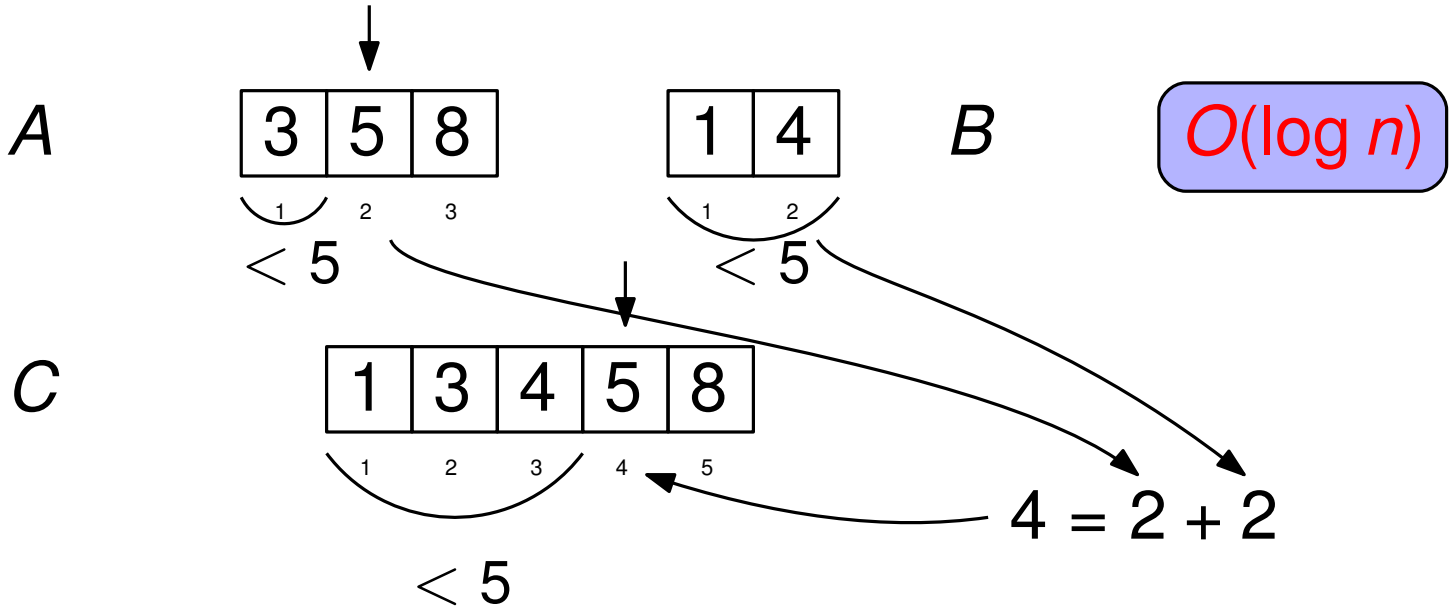


function MERGE(A, B, C)
  for $i = 1$ to $|A|$ **in parallel do**
    $k = i$ + PREDECESSOR($A[i]$, B)
    $C[k] = A[i]$
  for $j = 1$ to $|B|$ **in parallel do**
    $k = j$ + PREDECESSOR($B[j]$, A)
    $C[k] = B[j]$

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**
    $k = i$ + 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)

$O(\log n)$

Work vs Parallel Time

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

Work vs Parallel Time

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

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

Work vs Parallel Time

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

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

$W = O(n)$
$T_\infty = O(1)$

Work vs Parallel Time

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

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

Work vs Parallel Time

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

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

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

Work vs Parallel Time

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

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

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

Work vs Parallel Time

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

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

$$W = 2W(n/2) + O(n)$$
$$= O(n \log n)$$
$$T_\infty(n) = T_\infty(n/2) + O(1)$$

Work vs Parallel Time

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

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

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

$$T_\infty(n) = T_\infty(n/2) + O(1)$$
$$= O(\log n)$$

Work vs Parallel Time

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

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)

Work vs Parallel Time

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

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)

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

Work vs Parallel Time

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

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$(B[j], A)$
    $C[k] = B[j]$

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

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

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

Work vs Parallel Time

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

Work vs Parallel Time

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

$$T_p = O(\frac{W}{P} + T_\infty)$$