CSE 590: Special Topics Course
(Supercomputing)

Lecture 4
(Analyzing Multithreaded Algorithms)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Spring 2016

The Master Theorem

A Useful Recurrence

Consider the following recurrence:

$$T(n) = \begin{cases} \Theta(1), & if\ n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise; \end{cases}$$

where, $a \geq 1$ and $b > 1$.

Arises frequently in the analyses of *divide-and-conquer* algorithms.

Consider the following recurrences from previous lectures.

Karatsuba's Algorithm: $T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$

Strassen's Algorithm: $T(n) = 7T\left(\frac{n}{2}\right) + \Theta\left(n^2\right)$

Fast Fourier Transform: $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

How the Recurrence Unfolds

$$T(n) = \begin{cases} \Theta(1), & if\ n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$

How the Recurrence Unfolds

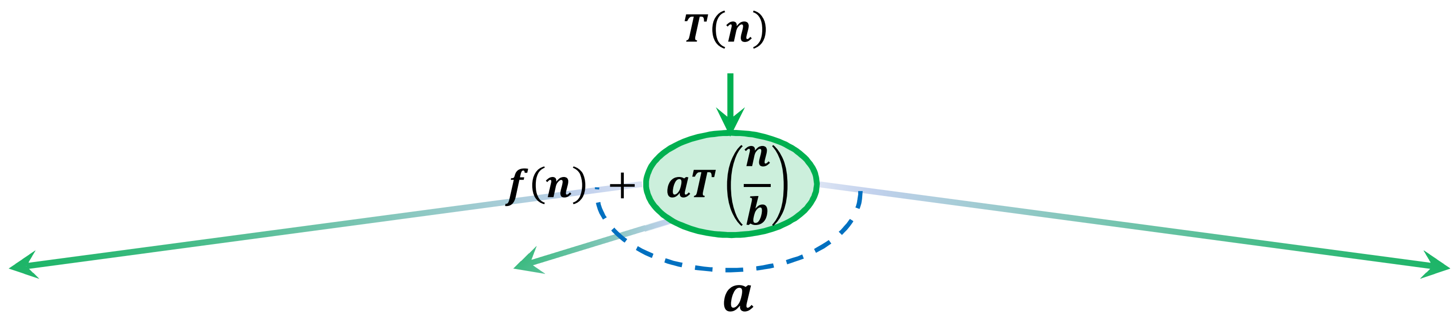
$$T(n) = \begin{cases} \Theta(1), & if\ n \le 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$

$$\boldsymbol{T(n)}$$

$$\boldsymbol{f(n) + aT\left(\frac{n}{b}\right)}$$

How the Recurrence Unfolds

$$T(n) = \begin{cases} \Theta(1), & if\ n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$



How the Recurrence Unfolds

$$T(n) = \begin{cases} \Theta(1), & if\ n \le 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$

$\boldsymbol{T(n)}$

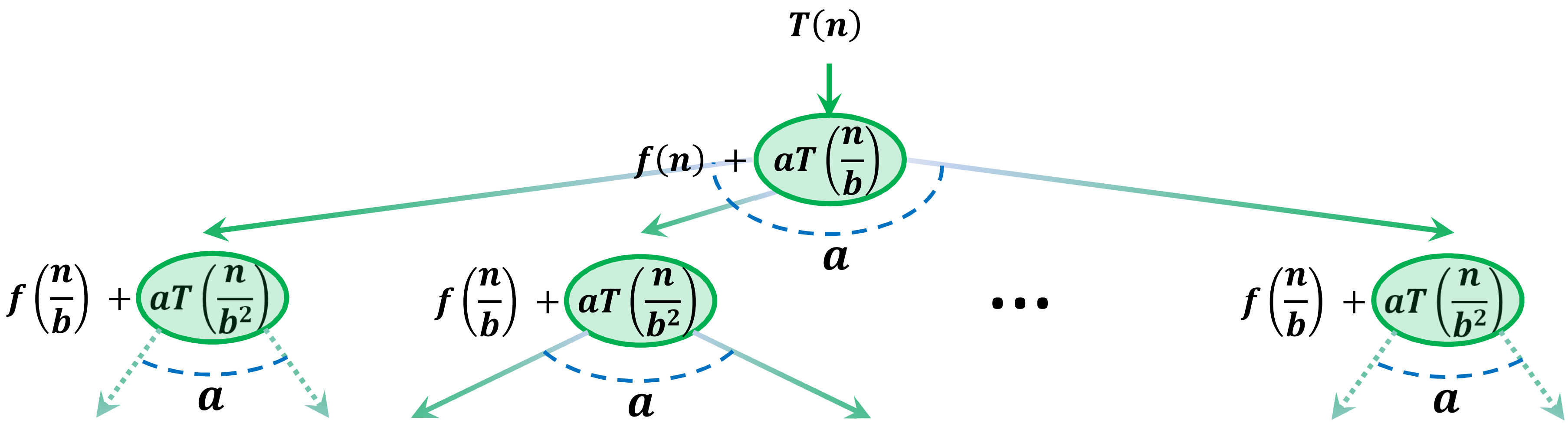
$\boldsymbol{f(n) + aT\left(\frac{n}{b}\right)}$

$\boldsymbol{a}$

$\boldsymbol{f\left(\frac{n}{b}\right) + aT\left(\frac{n}{b^2}\right)}$ $\quad$ $\boldsymbol{f\left(\frac{n}{b}\right) + aT\left(\frac{n}{b^2}\right)}$ $\quad \cdots \quad$ $\boldsymbol{f\left(\frac{n}{b}\right) + aT\left(\frac{n}{b^2}\right)}$

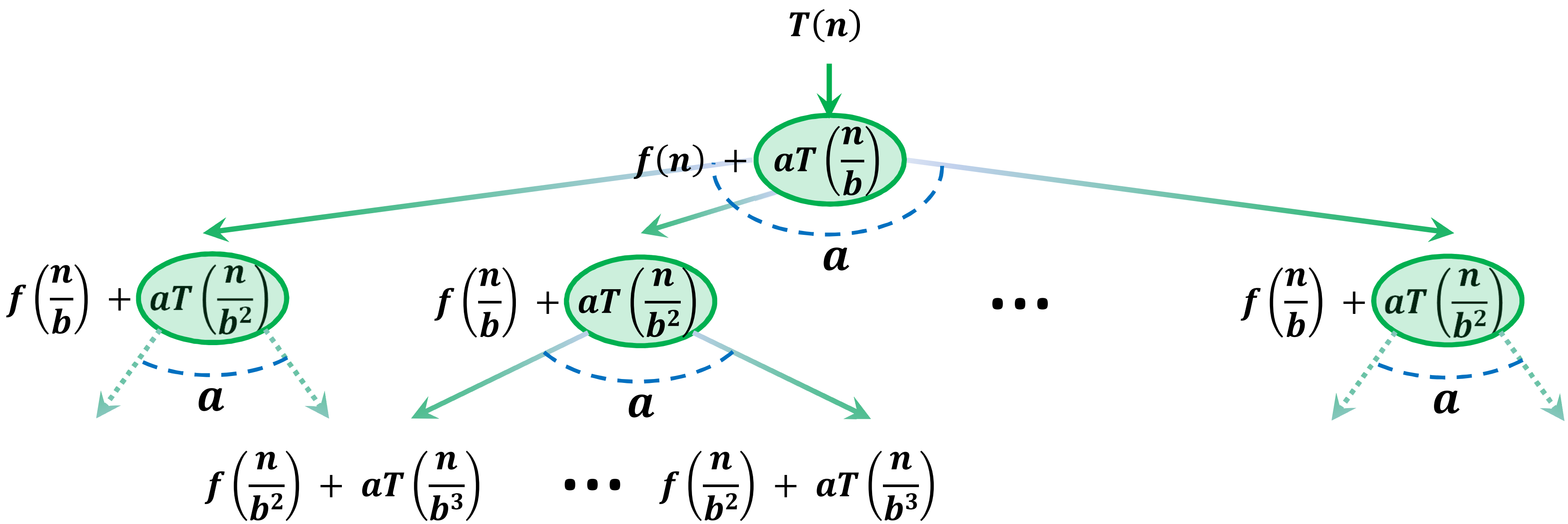
How the Recurrence Unfolds

$$T(n) = \begin{cases} \Theta(1), & if\ n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$



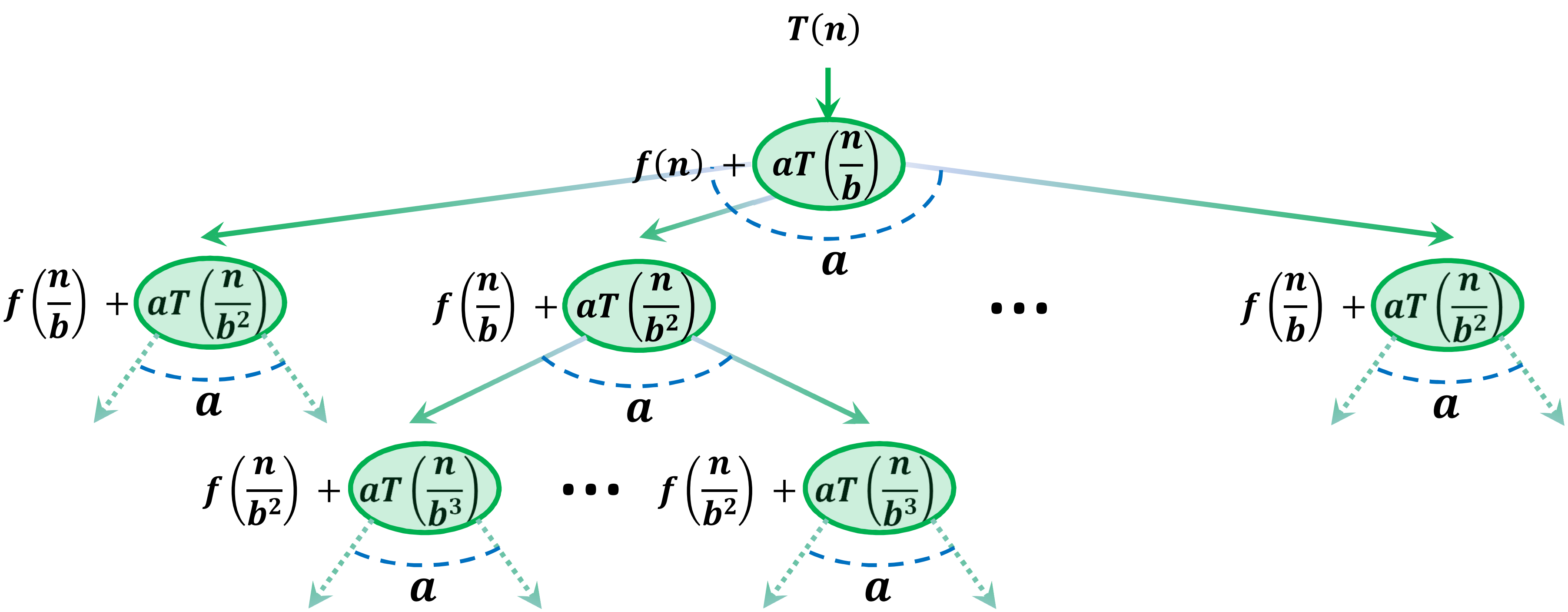
How the Recurrence Unfolds

$$T(n) = \begin{cases} \Theta(1), & if\ n \le 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$



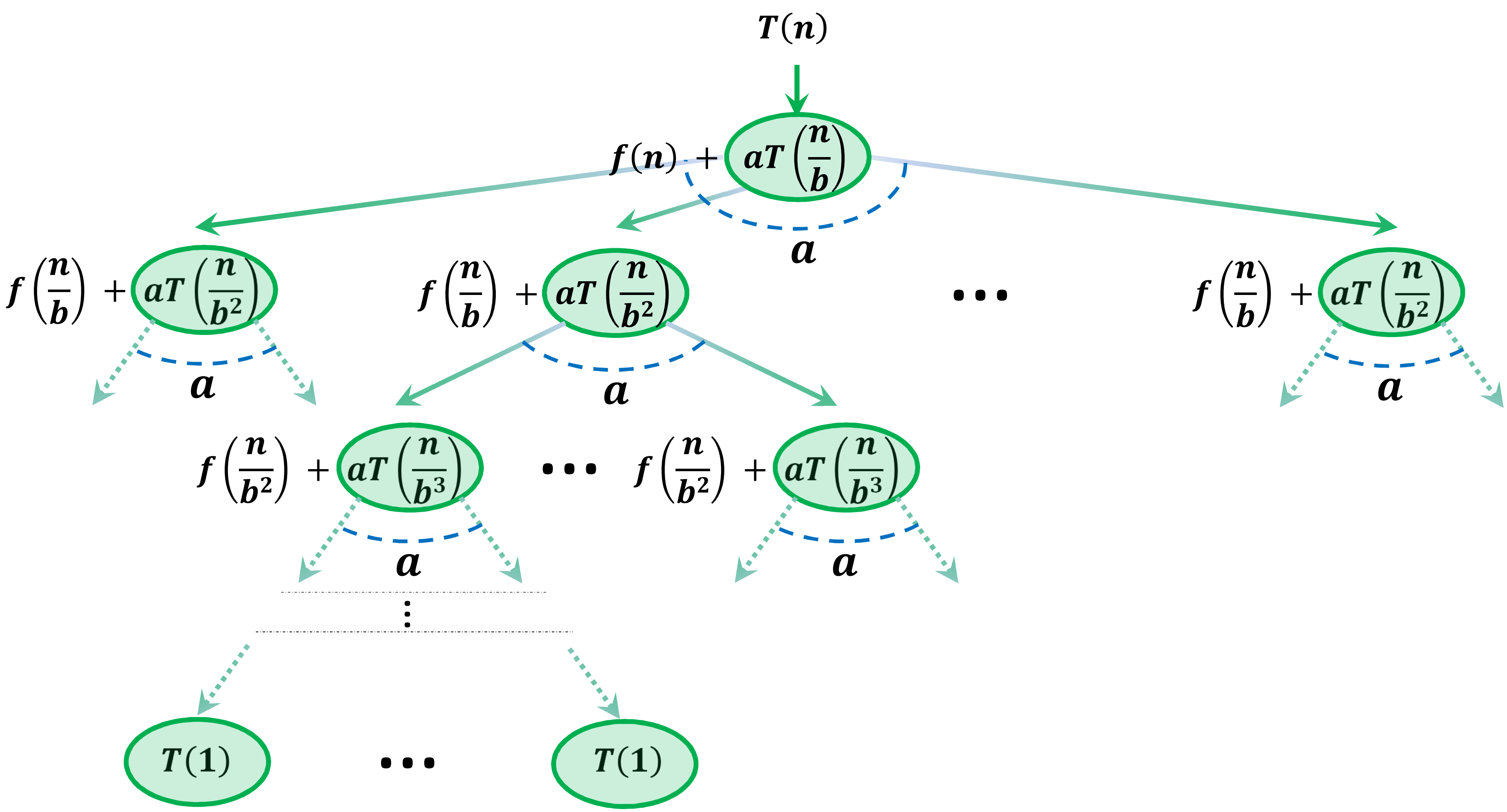
How the Recurrence Unfolds

$$T(n) = \begin{cases} \Theta(1), & if\ n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$



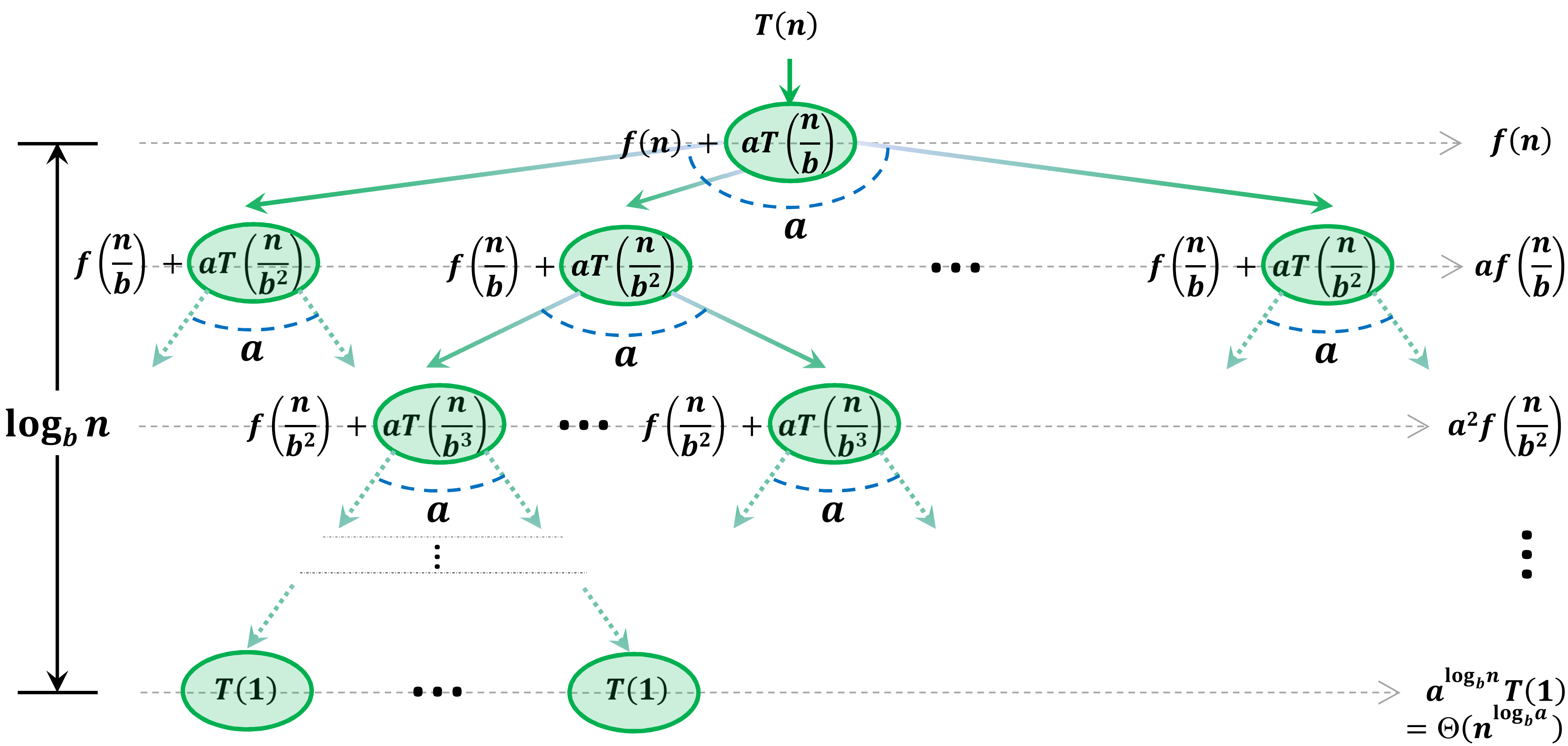
How the Recurrence Unfolds

$$T(n) = \begin{cases} \Theta(1), & \text{if } n \le 1, \\ aT\left(\frac{n}{b}\right) + f(n), & \text{otherwise.} \end{cases}$$



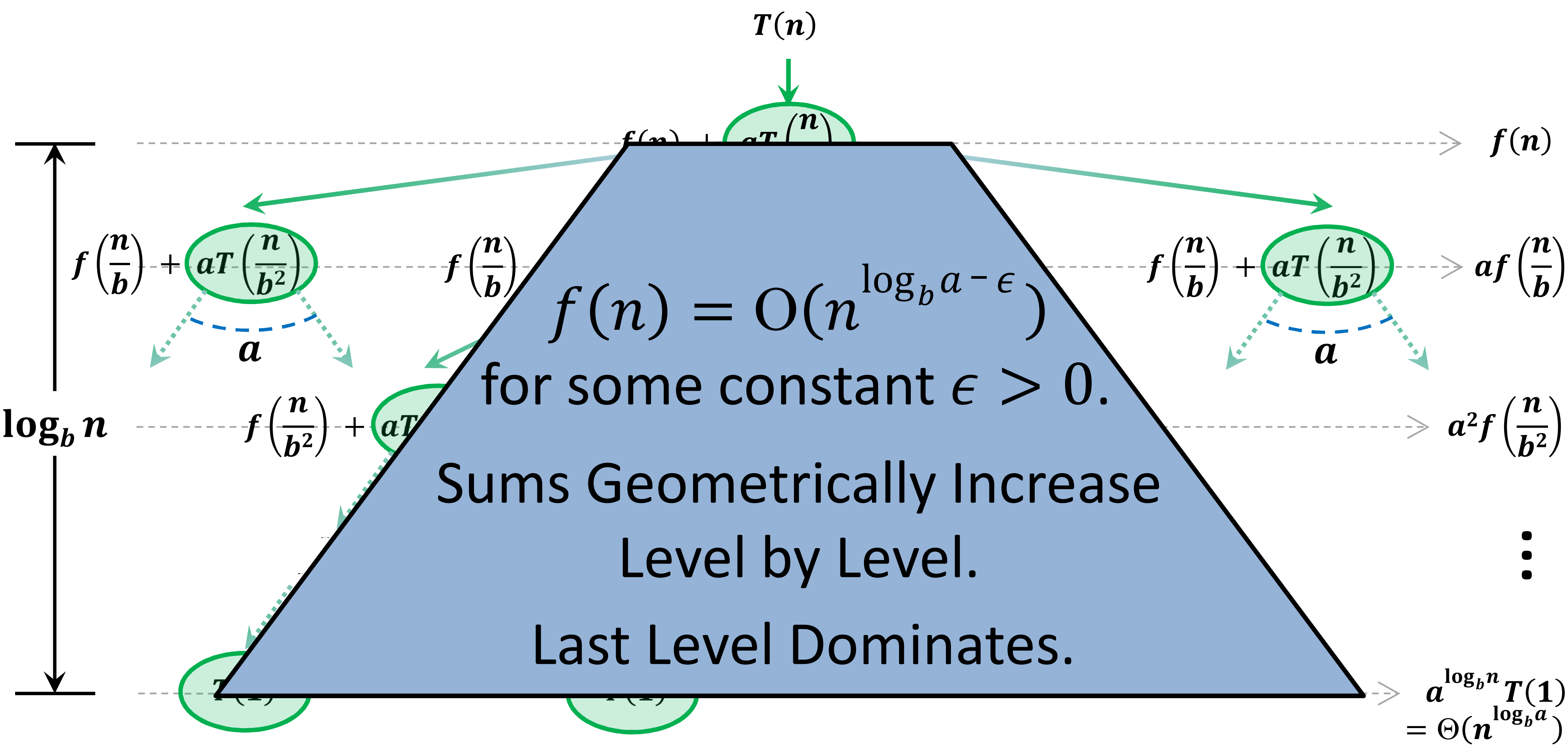
How the Recurrence Unfolds

$$T(n) = \begin{cases} \Theta(1), & \text{if } n \le 1, \\ aT\left(\frac{n}{b}\right) + f(n), & \text{otherwise.} \end{cases}$$



How the Recurrence Unfolds: Case 1

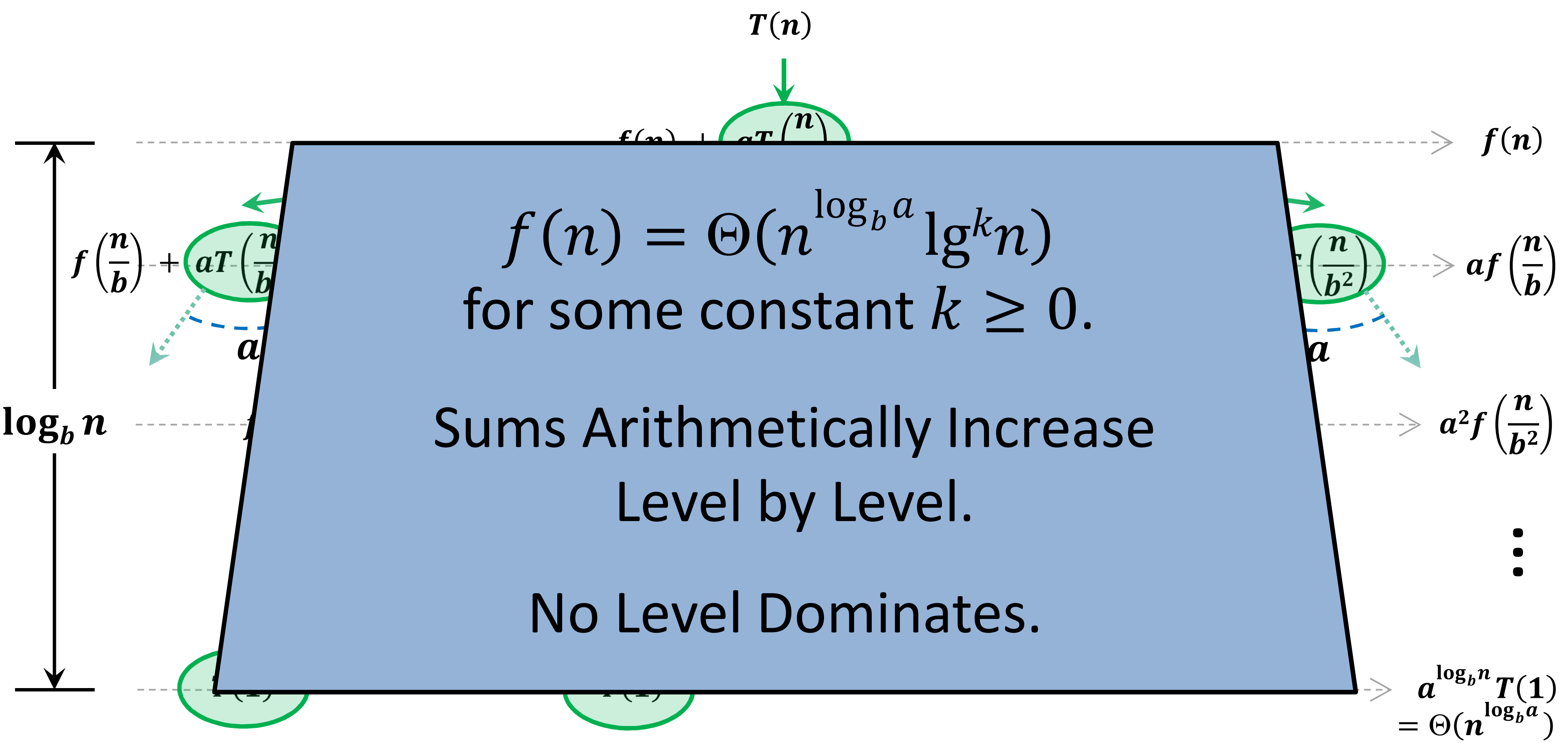
$$T(n) = \begin{cases} \Theta(1), & if\ n \le 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$



$$\boxed{T(n) = \Theta(n^{\log_b a})}$$

How the Recurrence Unfolds: Case 2

$$T(n) = \begin{cases} \Theta(1), & \text{if } n \leq 1, \\ aT\left(\frac{n}{b}\right) + f(n), & \text{otherwise.} \end{cases}$$



$f(n) = \Theta(n^{\log_b a} \lg^k n)$
for some constant $k \geq 0$.

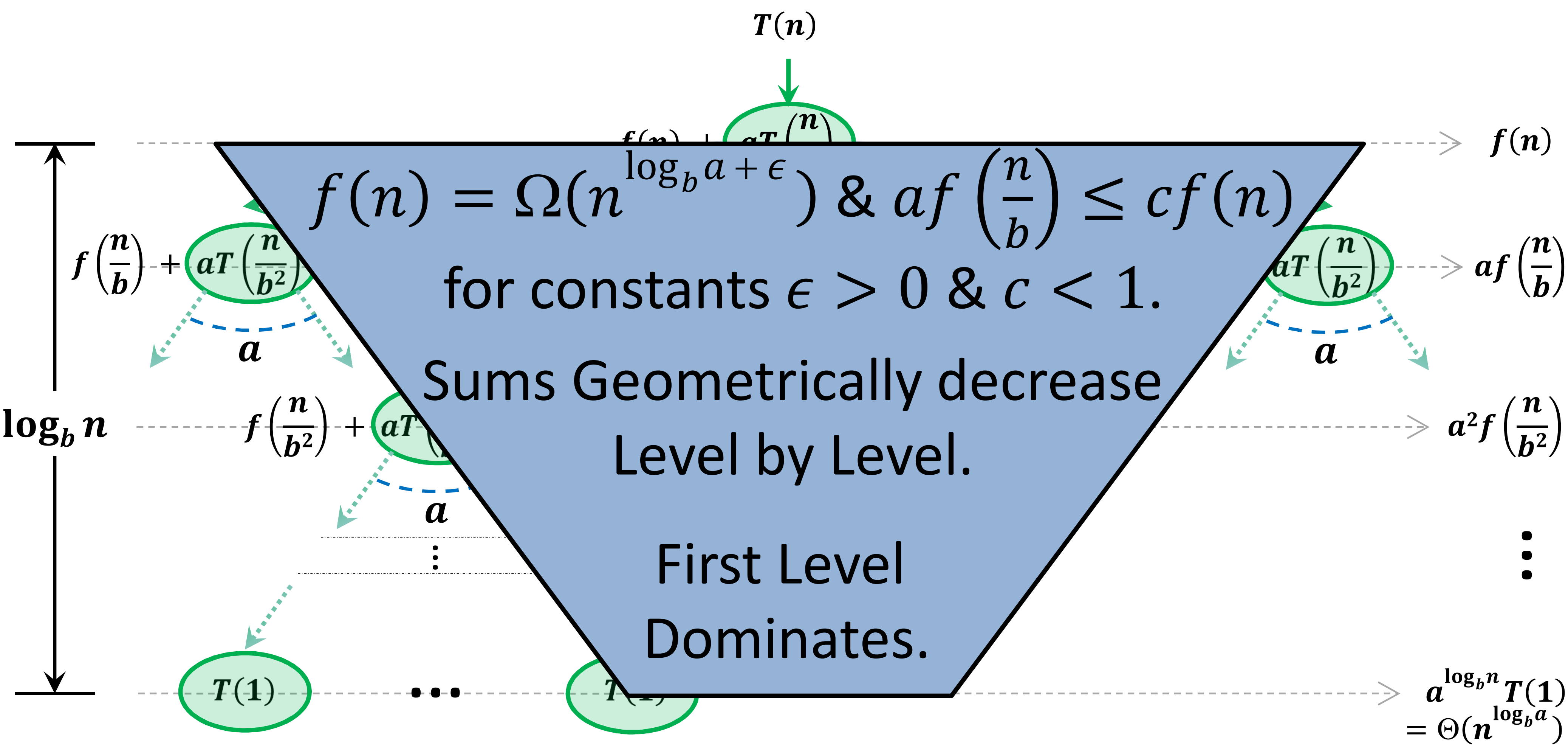
Sums Arithmetically Increase Level by Level.

No Level Dominates.

$$\boldsymbol{T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)}$$

How the Recurrence Unfolds: Case 3

$$T(n) = \begin{cases} \Theta(1), & if\ n \le 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise. \end{cases}$$



$f(n) = \Omega(n^{\log_b a + \epsilon})$ & $af\left(\frac{n}{b}\right) \le cf(n)$

for constants $\epsilon > 0$ & $c < 1$.

Sums Geometrically decrease Level by Level.

First Level Dominates.

$$\boxed{T(n) = \Theta(f(n))}$$

The Master Theorem

$$T(n) = \begin{cases} \Theta(1), & if\ n \le 1, \\ aT\left(\frac{n}{b}\right) + f(n), & otherwise\ (a \ge 1, b > 1). \end{cases}$$

Case 1: $f(n) = \mathrm{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$

$$\boxed{T(n) = \Theta(n^{\log_b a})}$$

Case 2: $f(n) = \Theta\left(n^{\log_b a} \lg^k n\right)$ for some constant $k \ge 0$.

$$\boxed{T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)}$$

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af\left(\frac{n}{b}\right) \le cf(n)$

for constants $\epsilon > 0$ and $c < 1$.

$$\boxed{T(n) = \Theta\left(f(n)\right)}$$

Example Applications of Master Theorem

Example 1: $T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$

Master Theorem Case 1: $T(n) = \Theta\left(n^{\log_2 3}\right)$

Example 2: $T(n) = 7T\left(\frac{n}{2}\right) + \Theta\left(n^2\right)$

Master Theorem Case 1: $T(n) = \Theta\left(n^{\log_2 7}\right)$

Example 3: $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

Master Theorem Case 2: $T(n) = \Theta(n\log n)$

Assuming that we have an infinite number of processors, and each recursive call in example 2 above can be executed in parallel:

Example 4: $T(n) = T\left(\frac{n}{2}\right) + \Theta\left(n^2\right)$

Master Theorem Case 3: $T(n) = \Theta\left(n^2\right)$

Recurrences not Solvable using the Master Theorem

Example 1: $T(n) = \sqrt{n}\,T\left(\frac{n}{2}\right) + n$

$a = \sqrt{n}$ is not a constant

Example 2: $T(n) = 2T\left(\frac{n}{\log n}\right) + n^2$

$b = \log n$ is not a constant

Example 3: $T(n) = \frac{1}{2}T\left(\frac{n}{2}\right) + n^2$

$a = \frac{1}{2}$ is not ≥ 1

Example 4: $T(n) = 2T\left(\frac{4n}{3}\right) + n$

$b = \frac{3}{4}$ is not > 1.

Recurrences not Solvable using the Master Theorem

Example 5: $T(n) = 3T\left(\frac{n}{2}\right) - n$

$f(n) = -n$ is not positive

Example 6: $T(n) = 2\,T\left(\frac{n}{2}\right) + n^2 \sin n$

violates regularity condition of case 3

Example 7: $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

$f(n) = O\left(n^{\log_b a}\right)$, but $\neq O\left(n^{\log_b a - \epsilon}\right)$ for any constant $\epsilon > 0$

Example 8: $T(n) = T\left(\frac{n}{2}\right) + 2T\left(\frac{n}{4}\right) + n$

a and b are not fixed

Multithreaded
Matrix Multiplication

Iterative Matrix Multiplication

$$z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj}$$

$$\begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1n} \\ z_{21} & z_{22} & \cdots & z_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nn} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{bmatrix} \times \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nn} \end{bmatrix}$$

```
Iter-MM ( Z, X, Y )          { X, Y, Z are n × n matrices,
                               where n is a positive integer }
1. for i ← 1 to n do
2.     for j ← 1 to n do
3.         Z[ i ][ j ] ← 0
4.         for k ← 1 to n do
5.             Z[ i ][ j ] ← Z[ i ][ j ] + X[ i ][ k ] · Y[ k ][ j ]
```

Parallel Iterative MM

```
Iter-MM ( Z, X, Y )            { X, Y, Z are n × n matrices,
                                 where n is a positive integer }
for i ← 1 to n do
    for j ← 1 to n do
        Z[ i ][ j ] ← 0
        for k ← 1 to n do
            Z[ i ][ j ] ← Z[ i ][ j ] + X[ i ][ k ] · Y[ k ][ j ]
```

```
Par-Iter-MM ( Z, X, Y )        { X, Y, Z are n × n matrices,
                                 where n is a positive integer }
parallel for i ← 1 to n do
    parallel for j ← 1 to n do
        Z[ i ][ j ] ← 0
        for k ← 1 to n do
            Z[ i ][ j ] ← Z[ i ][ j ] + X[ i ][ k ] · Y[ k ][ j ]
```

Parallel Iterative MM

```
Par-Iter-MM ( Z, X, Y )          { X, Y, Z are n × n matrices,
                                   where n is a positive integer }
parallel for i ← 1 to n do
    parallel for j ← 1 to n do
        Z[ i ][ j ] ← 0
        for k ← 1 to n do
            Z[ i ][ j ] ← Z[ i ][ j ] + X[ i ][ k ] · Y[ k ][ j ]
```

Work: $T_1(n) = \Theta(n^3)$

Span: $T_\infty(n) = \Theta(n)$

Parallel Running Time: $T_p(n) = O\left(\frac{T_1(n)}{p} + T_\infty(n)\right) = O\left(\frac{n^3}{p} + n\right)$

Parallelism: $\frac{T_1(n)}{T_\infty(n)} = \Theta(n^2)$

Parallel Recursive MM

$$\overset{Z}{\begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix}} = \overset{X}{\begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}} \times \overset{Y}{\begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix}}$$

$$= \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{bmatrix}$$

Parallel Recursive MM

```
Par-Rec-MM ( Z, X, Y )    { X, Y, Z are n × n matrices,
                             where n = 2^k for integer k ≥ 0 }
if n = 1 then
    Z ← Z + X · Y
else
    spawn Par-Rec-MM ( Z11, X11, Y11 )
    spawn Par-Rec-MM ( Z12, X11, Y12 )
    spawn Par-Rec-MM ( Z21, X21, Y11 )
          Par-Rec-MM ( Z21, X21, Y12 )
    sync
    spawn Par-Rec-MM ( Z11, X12, Y21 )
    spawn Par-Rec-MM ( Z12, X12, Y22 )
    spawn Par-Rec-MM ( Z21, X22, Y21 )
          Par-Rec-MM ( Z22, X22, Y22 )
    sync
endif
```

Parallel Recursive MM

```
Par-Rec-MM ( Z, X, Y )    { X, Y, Z are n × n matrices,
                             where n = 2^k for integer k ≥ 0 }
if n = 1 then
    Z ← Z + X · Y
else
    spawn Par-Rec-MM ( Z11, X11, Y11 )
    spawn Par-Rec-MM ( Z12, X11, Y12 )
    spawn Par-Rec-MM ( Z21, X21, Y11 )
          Par-Rec-MM ( Z21, X21, Y12 )
    sync
    spawn Par-Rec-MM ( Z11, X12, Y21 )
    spawn Par-Rec-MM ( Z12, X12, Y22 )
    spawn Par-Rec-MM ( Z21, X22, Y21 )
          Par-Rec-MM ( Z22, X22, Y22 )
    sync
endif
```

Work:

$$T_1(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ 8T_1\left(\frac{n}{2}\right) + \Theta(1), & otherwise. \end{cases}$$

$$= \Theta(n^3) \qquad \text{[MT Case 1]}$$

Span:

$$T_\infty(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ 2T_\infty\left(\frac{n}{2}\right) + \Theta(1), & otherwise. \end{cases}$$

$$= \Theta(n) \qquad \text{[MT Case 1]}$$

Parallelism: $\frac{T_1(n)}{T_\infty(n)} = \Theta(n^2)$

Additional Space:

$$s_\infty(n) = \Theta(1)$$

Recursive MM with More Parallelism

$$Z = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} = \begin{bmatrix} X_{11}Y_{11} + X_{12}Y_{21} & X_{11}Y_{12} + X_{12}Y_{22} \\ X_{21}Y_{11} + X_{22}Y_{21} & X_{21}Y_{12} + X_{22}Y_{22} \end{bmatrix}$$

$$= \begin{bmatrix} X_{11}Y_{11} & X_{11}Y_{12} \\ X_{21}Y_{11} & X_{21}Y_{12} \end{bmatrix} + \begin{bmatrix} X_{12}Y_{21} & X_{12}Y_{22} \\ X_{22}Y_{21} & X_{22}Y_{22} \end{bmatrix}$$

Recursive MM with More Parallelism

```
Par-Rec-MM2 ( Z, X, Y )      { X, Y, Z are n × n matrices,
                               where n = 2^k for integer k ≥ 0 }

if n = 1 then
    Z ← Z + X · Y
else            { T is a temporary n × n matrix }
    spawn Par-Rec-MM2 ( Z11, X11, Y11 )
    spawn Par-Rec-MM2 ( Z12, X11, Y12 )
    spawn Par-Rec-MM2 ( Z21, X21, Y11 )
    spawn Par-Rec-MM2 ( Z21, X21, Y12 )
    spawn Par-Rec-MM2 ( T11, X12, Y21 )
    spawn Par-Rec-MM2 ( T12, X12, Y22 )
    spawn Par-Rec-MM2 ( T21, X22, Y21 )
          Par-Rec-MM2 ( T22, X22, Y22 )
    sync
    parallel for i ← 1 to n do
        parallel for j ← 1 to n do
            Z[ i ][ j ] ← Z[ i ][ j ] + T[ i ][ j ]
endif
```

Recursive MM with More Parallelism

```
Par-Rec-MM2 ( Z, X, Y )      { X, Y, Z are n × n matrices,
                               where n = 2^k for integer k ≥ 0 }
if n = 1 then
    Z ← Z + X · Y
else        { T is a temporary n × n matrix }
    spawn Par-Rec-MM2 ( Z11, X11, Y11 )
    spawn Par-Rec-MM2 ( Z12, X11, Y12 )
    spawn Par-Rec-MM2 ( Z21, X21, Y11 )
    spawn Par-Rec-MM2 ( Z21, X21, Y12 )
    spawn Par-Rec-MM2 ( T11, X12, Y21 )
    spawn Par-Rec-MM2 ( T12, X12, Y22 )
    spawn Par-Rec-MM2 ( T21, X22, Y21 )
          Par-Rec-MM2 ( T22, X22, Y22 )
    sync
    parallel for i ← 1 to n do
        parallel for j ← 1 to n do
            Z[ i ][ j ] ← Z[ i ][ j ] + T[ i ][ j ]
endif
```

Work:

$$T_1(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ 8T_1\left(\frac{n}{2}\right) + \Theta\left(n^2\right), & otherwise. \end{cases}$$

$$= \Theta\left(n^3\right) \qquad \text{[MT Case 1]}$$

Span:

$$T_\infty(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ T_\infty\left(\frac{n}{2}\right) + \Theta(\log n), & otherwise. \end{cases}$$

$$= \Theta(\log^2 n) \qquad \text{[MT Case 2]}$$

Parallelism: $\frac{T_1(n)}{T_\infty(n)} = \Theta\left(\frac{n^3}{\log^2 n}\right)$

Additional Space:

$$s_\infty(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ 8s_\infty\left(\frac{n}{2}\right) + \Theta\left(n^2\right), & otherwise. \end{cases}$$

$$= \Theta\left(n^3\right) \qquad \text{[MT Case 1]}$$

Parallel Merge Sort

Parallel Merge Sort

```
Merge-Sort ( A, p, r )        { sort the elements in A[ p … r ] }

if p < r then
    q ← ⌊ ( p + r ) / 2 ⌋
    Merge-Sort ( A, p, q )
    Merge-Sort ( A, q + 1, r )
    Merge ( A, p, q, r )
```

```
Par-Merge-Sort ( A, p, r )    { sort the elements in A[ p … r ] }

if p < r then
    q ← ⌊ ( p + r ) / 2 ⌋
    spawn Merge-Sort ( A, p, q )
          Merge-Sort ( A, q + 1, r )
    sync
    Merge ( A, p, q, r )
```

Parallel Merge Sort

Par-Merge-Sort (*A*, *p*, *r*) { *sort the elements in A*[*p* … *r*] }

1. *if* $p < r$ *then*
2. $q \leftarrow \lfloor (p + r) / 2 \rfloor$
3. *spawn Merge-Sort* (*A*, *p*, *q*)
4. *Merge-Sort* (*A*, *q* + 1, *r*)
5. *sync*
6. *Merge* (*A*, *p*, *q*, *r*)

Work: $$T_1(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ 2T_1\left(\frac{n}{2}\right) + \Theta(n), & otherwise. \end{cases}$$

$$= \Theta(n \log n) \qquad \text{[MT Case 2]}$$

Span: $$T_\infty(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ T_\infty\left(\frac{n}{2}\right) + \Theta(n), & otherwise. \end{cases}$$

$$= \Theta(n) \qquad \text{[MT Case 3]}$$

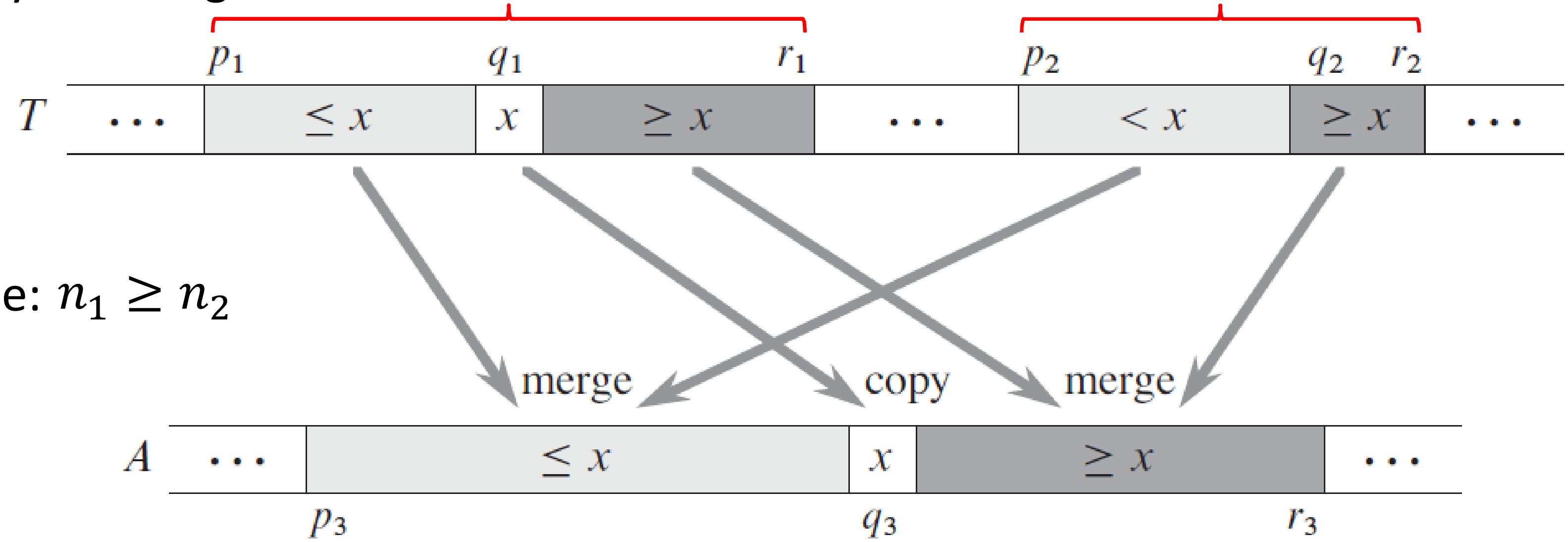
Parallelism: $\frac{T_1(n)}{T_\infty(n)} = \Theta(\log n)$

Parallel Merge

$n_1 = r_1 - p_1 + 1$ $\quad$ $n_2 = r_2 - p_2 + 1$

subarrays to merge: $\underbrace{T[p_1..r_1]}$ $\quad$ $T[p_2..r_2]$

suppose: $n_1 \geq n_2$



merged output: $A[p_3..r_3]$

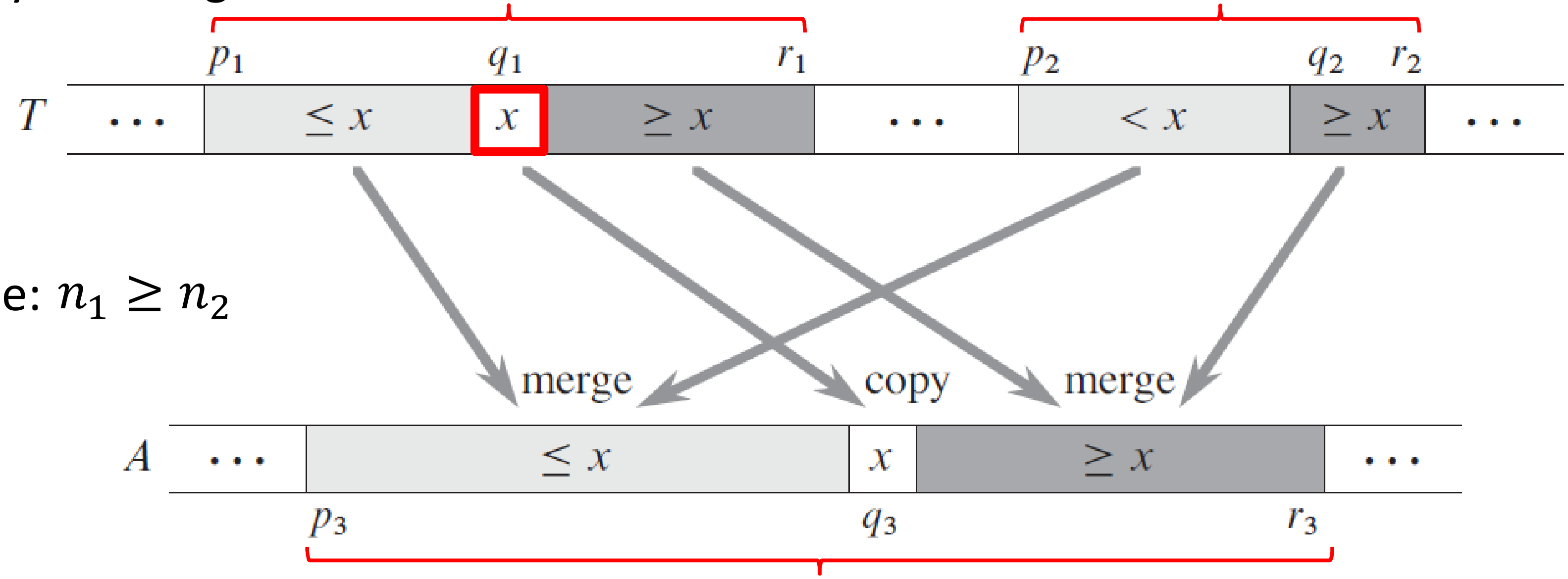
$n_3 = r_3 - p_3 + 1 = n_1 + n_2$

Source: Cormen et al., "Introduction to Algorithms", 3rd Edition

Parallel Merge

$n_1 = r_1 - p_1 + 1$ $\qquad$ $n_2 = r_2 - p_2 + 1$

subarrays to merge: $T[p_1 \,..\, r_1]$ $\qquad$ $T[p_2 \,..\, r_2]$



suppose: $n_1 \geq n_2$

merged output: $A[p_3 \,..\, r_3]$

$n_3 = r_3 - p_3 + 1 = n_1 + n_2$

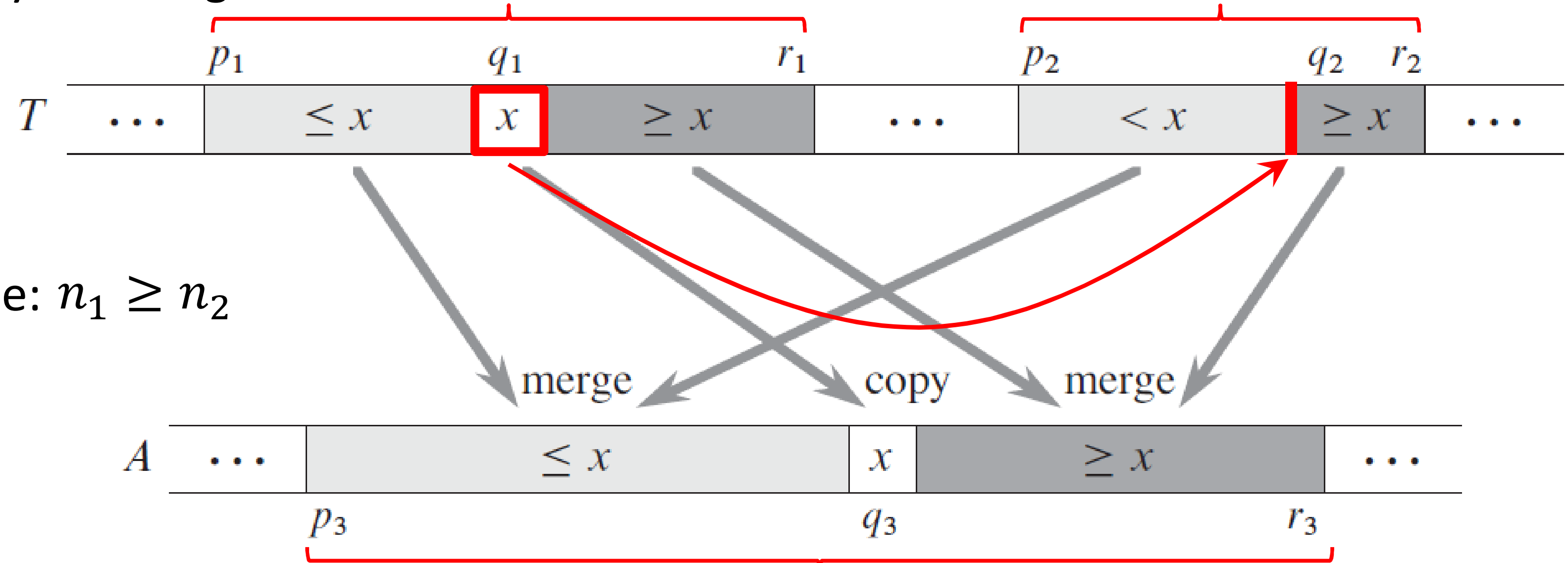
Step 1: Find $x = T[q_1]$, where q_1 is the midpoint of $T[p_1 \,..\, r_1]$

Source: Cormen et al., "Introduction to Algorithms", 3rd Edition

Parallel Merge

$n_1 = r_1 - p_1 + 1$ $\qquad$ $n_2 = r_2 - p_2 + 1$

subarrays to merge: $T[p_1..r_1]$ $\qquad$ $T[p_2..r_2]$



suppose: $n_1 \geq n_2$

merged output: $A[p_3..r_3]$

$n_3 = r_3 - p_3 + 1 = n_1 + n_2$

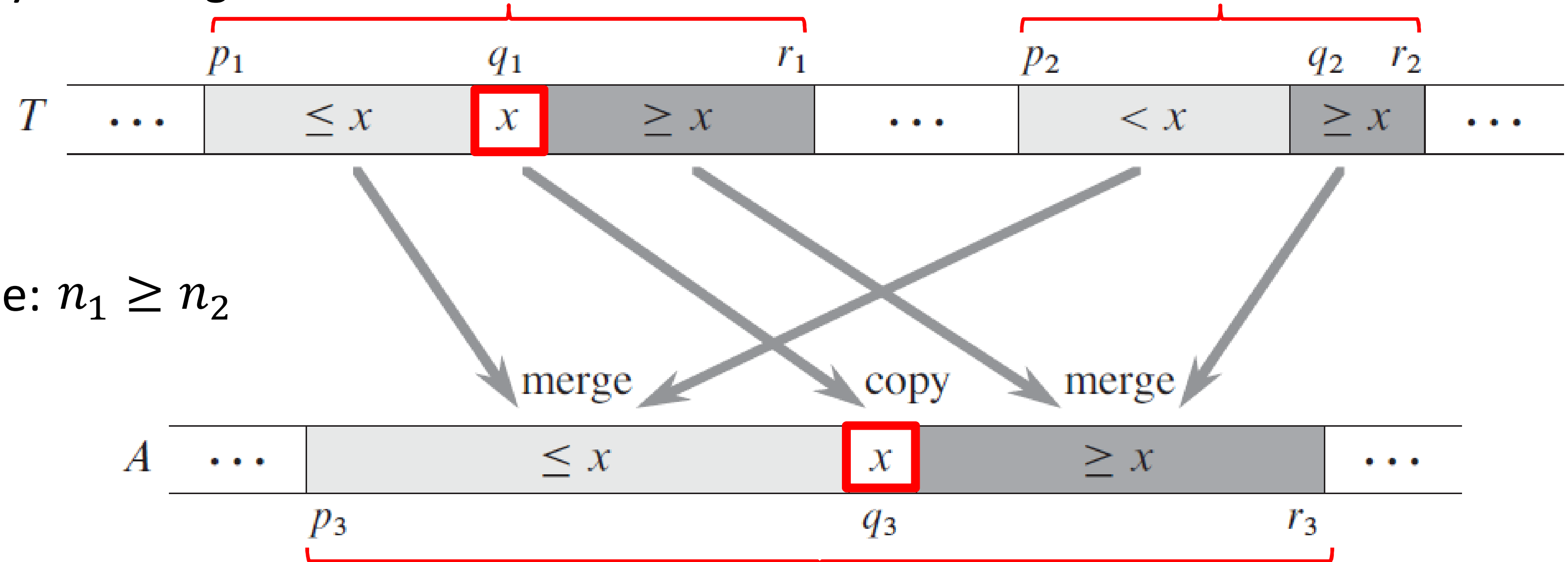
Source: Cormen et al., "Introduction to Algorithms", 3rd Edition

Step 2: Use binary search to find the index q_2 in subarray $T[p_2..r_2]$ so that the subarray would still be sorted if we insert x between $T[q_2 - 1]$ and $T[q_2]$

Parallel Merge

$$n_1 = r_1 - p_1 + 1 \qquad\qquad n_2 = r_2 - p_2 + 1$$

subarrays to merge: $T[p_1 \,..\, r_1]$ $\quad T[p_2 \,..\, r_2]$



suppose: $n_1 \geq n_2$

merged output: $A[p_3 \,..\, r_3]$

$$n_3 = r_3 - p_3 + 1 = n_1 + n_2$$

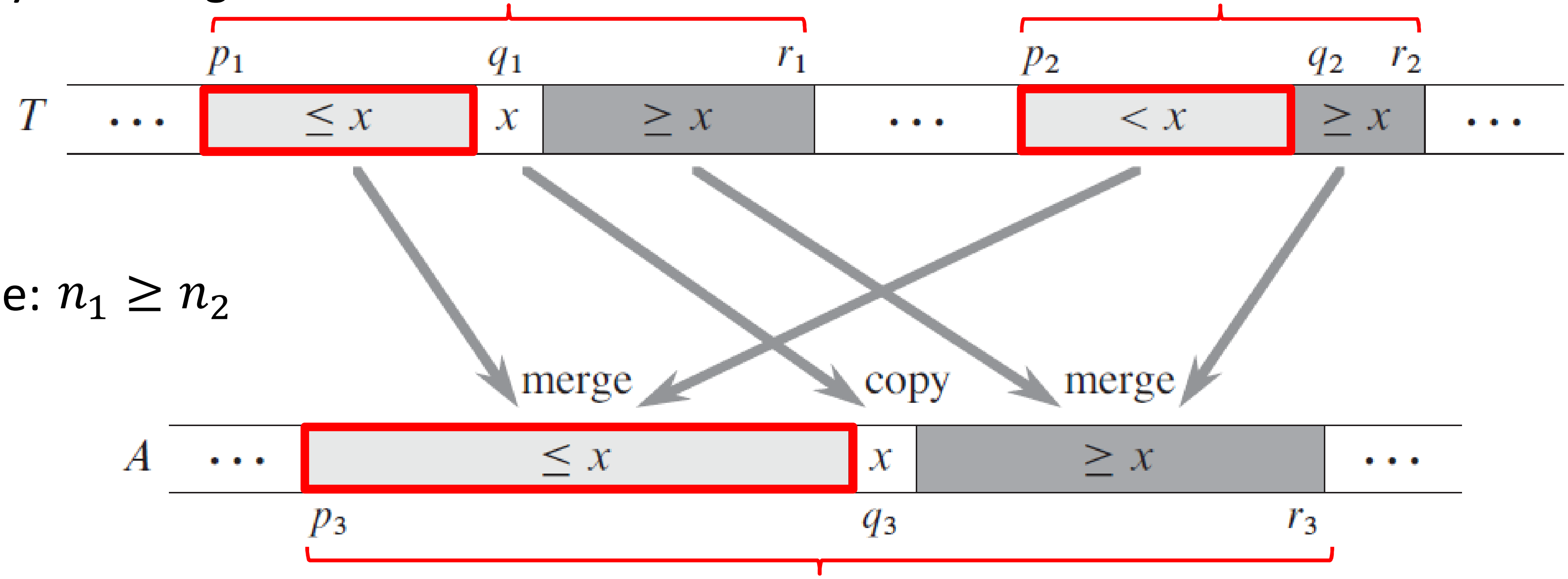
Source: Cormen et al., "Introduction to Algorithms", 3rd Edition

Step 3: Copy x to $A[q_3]$, where $q_3 = p_3 + (q_1 - p_1) + (q_2 - p_2)$

Parallel Merge

$n_1 = r_1 - p_1 + 1$ $\quad$ $n_2 = r_2 - p_2 + 1$

subarrays to merge: $T[p_1 \,..\, r_1]$ $\quad$ $T[p_2 \,..\, r_2]$



suppose: $n_1 \ge n_2$

merged output: $A[p_3 \,..\, r_3]$

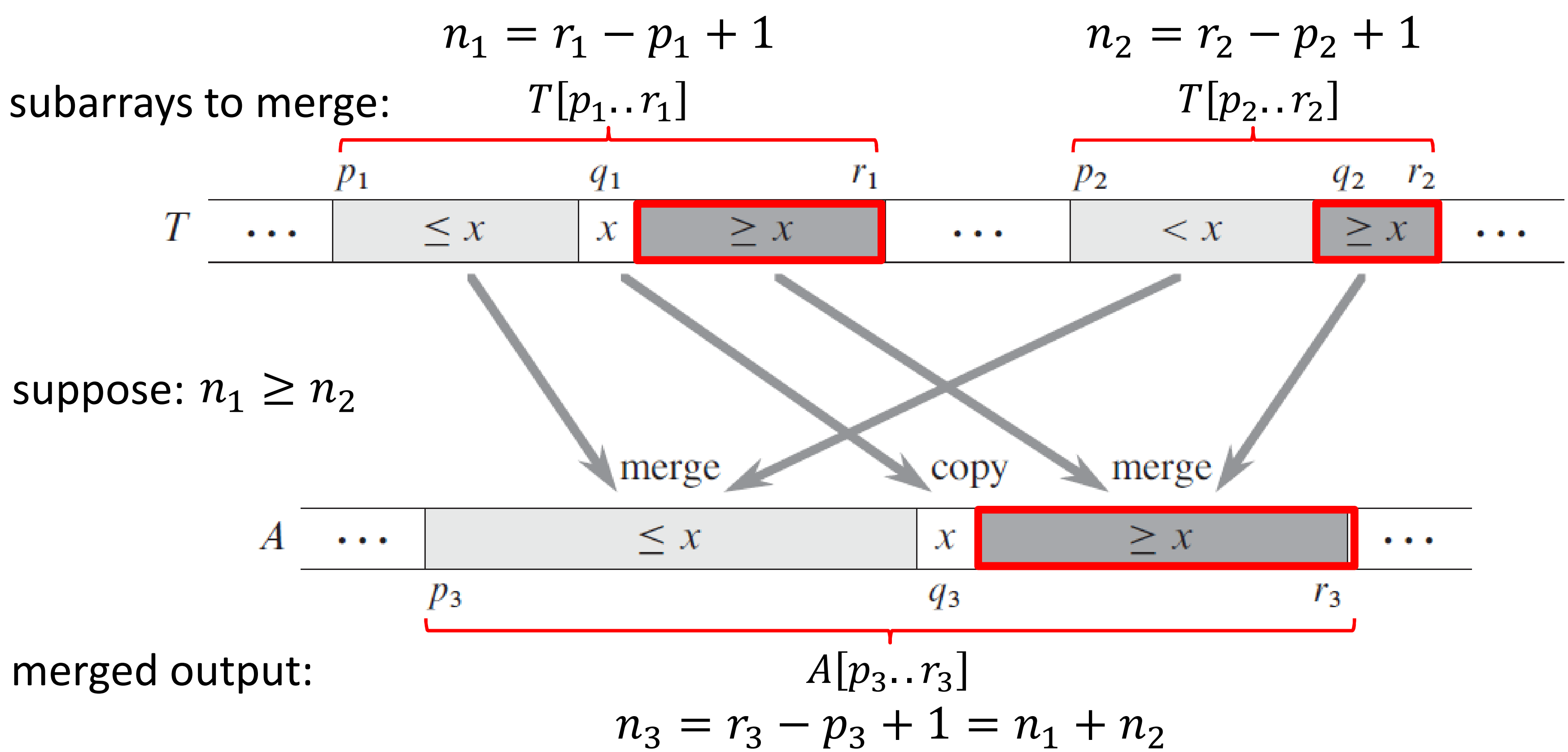
$n_3 = r_3 - p_3 + 1 = n_1 + n_2$

Source: Cormen et al., "Introduction to Algorithms", 3rd Edition

Perform the following two steps in parallel.

Step 4(a): Recursively merge $T[p_1 \,..\, q_1 - 1]$ with $T[p_2 \,..\, q_2 - 1]$, and place the result into $A[p_3 \,..\, q_3 - 1]$

Parallel Merge



Source: Cormen et al., "Introduction to Algorithms", 3rd Edition

Perform the following two steps in parallel.

Step 4(a): Recursively merge $T[p_1 \,..\, q_1 - 1]$ with $T[p_2 \,..\, q_2 - 1]$, and place the result into $A[p_3 \,..\, q_3 - 1]$

Step 4(b): Recursively merge $T[q_1 + 1 \,..\, r_1]$ with $T[q_2 + 1 \,..\, r_2]$, and place the result into $A[q_3 + 1 \,..\, r_3]$

Parallel Merge

Par-Merge ($T, p_1, r_1, p_2, r_2, A, p_3$)

$n_1 \leftarrow r_1 - p_1 + 1, \quad n_2 \leftarrow r_2 - p_2 + 1$

if $n_1 < n_2$ *then*

    $p_1 \leftrightarrow p_2, \quad r_1 \leftrightarrow r_2, \quad n_1 \leftrightarrow n_2$

if $n_1 = 0$ *then return*

else

    $q_1 \leftarrow \lfloor (p_1 + r_1) / 2 \rfloor$

    $q_2 \leftarrow$ *Binary-Search* ($T[q_1], T, p_2, r_2$)

    $q_3 \leftarrow p_3 + (q_1 - p_1) + (q_2 - p_2)$

    $A[q_3] \leftarrow T[q_1]$

    spawn *Par-Merge* ($T, p_1, q_1-1, p_2, q_2-1, A, p_3$)

        Par-Merge ($T, q_1+1, r_1, q_2+1, r_2, A, q_3+1$)

    sync

Parallel Merge

```
Par-Merge ( T, p1, r1, p2, r2, A, p3 )
1.  n1 ← r1 - p1 + 1,   n2 ← r2 - p2 + 1
2.  if n1 < n2 then
3.      p1 ↔ p2,  r1 ↔ r2,  n1 ↔ n2
4.  if n1 = 0 then return
5.  else
6.      q1 ← ⌊ ( p1 + r1 ) / 2 ⌋
7.      q2 ← Binary-Search (  T[ q1 ],  T,  p2, r2 )
8.      q3 ← p3 + ( q1 - p1 ) + ( q2 - p2 )
9.      A[ q3 ] ← T[ q1 ]
10.     spawn Par-Merge ( T, p1, q1-1, p2, q2-1, A, p3 )
11.           Par-Merge ( T, q1+1, r1, q2+1, r2, A, q3+1 )
12.     sync
```

We have,

$n_2 \leq n_1 \Rightarrow 2n_2 \leq n_1 + n_2 = n$

In the worst case, a recursive call in lines 9-10 merges half the elements of $T[p_1..r_1]$ with all elements of $T[p_2..r_2]$.

Hence, #elements involved in such a call:

$$\left\lfloor \frac{n_1}{2} \right\rfloor + n_2 \leq \frac{n_1}{2} + \frac{n_2}{2} + \frac{n_2}{2} = \frac{n_1 + n_2}{2} + \frac{2n_2}{4} \leq \frac{n}{2} + \frac{n}{4} = \frac{3n}{4}$$

Parallel Merge

```
Par-Merge ( T, p1, r1, p2, r2, A, p3 )
    n1 ← r1 - p1 + 1,   n2 ← r2 - p2 + 1
    if n1 < n2 then
        p1 ↔ p2,  r1 ↔ r2,  n1 ↔ n2
    if n1 = 0 then return
    else
        q1 ← ⌊ ( p1 + r1 ) / 2 ⌋
        q2 ← Binary-Search (  T[ q1 ],  T,  p2, r2 )
        q3 ← p3 + ( q1 - p1 ) + ( q2 - p2 )
        A[ q3 ] ← T[ q1 ]
        spawn Par-Merge ( T, p1, q1-1, p2, q2-1, A, p3 )
              Par-Merge ( T, q1+1, r1, q2+1, r2, A, q3+1 )
        sync
```

Span:

$$T_\infty(n) = \begin{cases} \Theta(1), & \text{if } n = 1, \\ T_\infty\left(\frac{3n}{4}\right) + \Theta(\log n), & \text{otherwise}. \end{cases}$$

$$= \Theta(\log^2 n) \qquad \text{[MT Case 2]}$$

Work:

Clearly, $T_1(n) = \Omega(n)$

We show below that, $T_1(n) = \mathrm{O}(n)$

For some $\alpha \in \left[\frac{1}{4}, \frac{3}{4}\right]$, we have the following recurrence,

$$T_1(n) = T_1(\alpha n) + T_1\big((1-\alpha)n\big) + \mathrm{O}(\log n)$$

Assuming $T_1(n) \le c_1 n - c_2 \log n$ for positive constants c_1 and c_2, and substituting on the right hand side of the above recurrence gives us: $T_1(n) \le c_1 n - c_2 \log n \ = \mathrm{O}(n)$.

Hence, $T_1(n) = \Theta(n)$.

Parallel Merge Sort with Parallel Merge

Par-Merge-Sort (*A*, *p*, *r*) { *sort the elements in A*[*p* … *r*] }

1. *if* $p < r$ *then*
2. $q \leftarrow \lfloor (p + r) / 2 \rfloor$
3. *spawn* *Merge-Sort* (*A*, *p*, *q*)
4. *Merge-Sort* (*A*, *q* + 1, *r*)
5. *sync*
6. *Par-Merge* (*A*, *p*, *q*, *r*)

Work: $$T_1(n) = \begin{cases} \Theta(1), & \text{if } n = 1, \\ 2T_1\left(\frac{n}{2}\right) + \Theta(n), & \text{otherwise.} \end{cases}$$

$$= \Theta(n \log n)$$ [MT Case 2]

Span: $$T_\infty(n) = \begin{cases} \Theta(1), & \text{if } n = 1, \\ T_\infty\left(\frac{n}{2}\right) + \Theta(\log^2 n), & \text{otherwise.} \end{cases}$$

$$= \Theta(\log^3 n)$$ [MT Case 2]

Parallelism: $$\frac{T_1(n)}{T_\infty(n)} = \Theta\left(\frac{n}{\log^2 n}\right)$$

Parallel Prefix Sums

Parallel Prefix Sums

Input: A sequence of n elements $\{x_1, x_2, \dots, x_n\}$ drawn from a set S with a binary associative operation, denoted by $\oplus$.

Output: A sequence of n partial sums $\{s_1, s_2, \dots, s_n\}$, where $s_i = x_1 \oplus x_2 \oplus \dots \oplus x_i$ for $1 \leq i \leq n$.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
5	3	7	1	3	6	2	4

$\oplus$ = binary addition

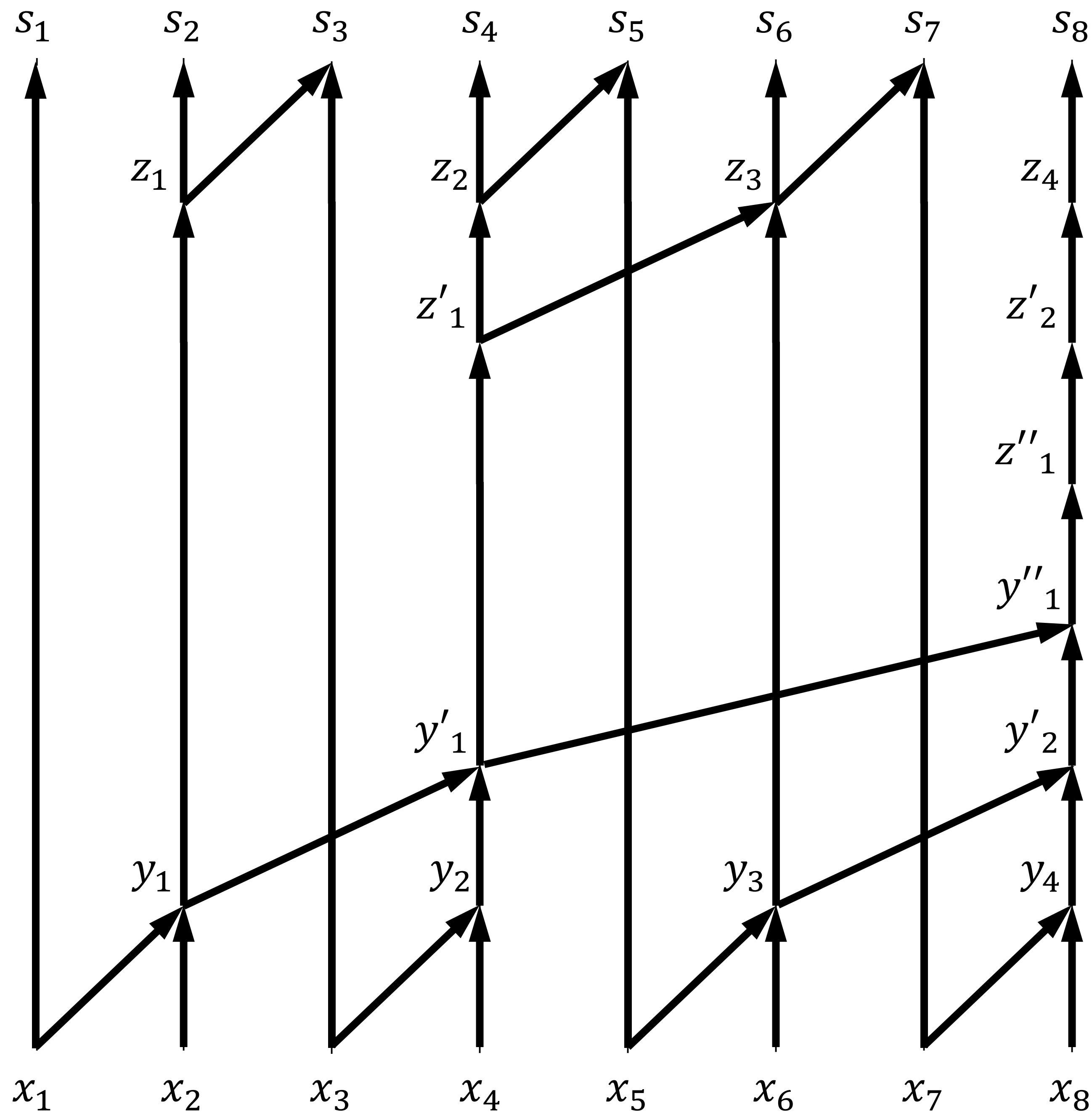
5	8	15	16	19	25	27	31
s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8

Parallel Prefix Sums

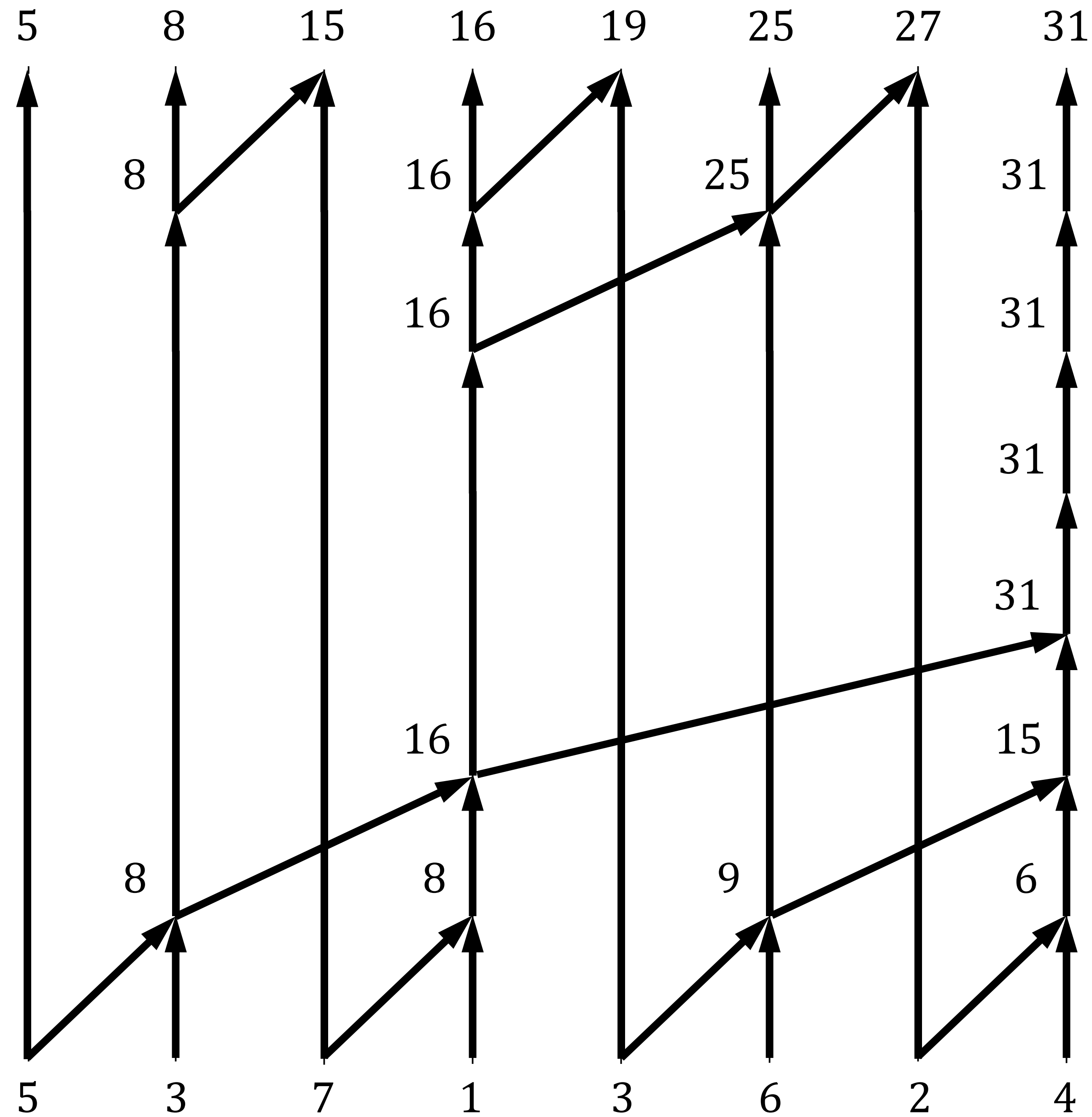
Prefix-Sum ($\langle x_1, x_2, \ldots, x_n \rangle, \oplus$) { $n = 2^k$ *for some* $k \geq 0$. *Return prefix sums* $\langle s_1, s_2, \ldots, s_n \rangle$ }

if $n = 1$ *then*
    $s_1 \leftarrow x_1$
else
    parallel for $i \leftarrow 1$ *to* $n/2$ *do*
        $y_i \leftarrow x_{2i-1} \oplus x_{2i}$
    $\langle z_1, z_2, \ldots, z_{n/2} \rangle \leftarrow$ *Prefix-Sum*($\langle y_1, y_2, \ldots, y_{n/2} \rangle, \oplus$)
    parallel for $i \leftarrow 1$ *to* n *do*
        if $i = 1$ *then* $s_1 \leftarrow x_1$
        else if $i = even$ *then* $s_i \leftarrow z_{i/2}$
            else $s_i \leftarrow z_{(i-1)/2} \oplus x_i$
return $\langle s_1, s_2, \ldots, s_n \rangle$

Parallel Prefix Sums



Parallel Prefix Sums



Parallel Prefix Sums

Prefix-Sum ($\langle x_1, x_2, \ldots, x_n \rangle$, $\oplus$)   { $n = 2^k$ *for some* $k \geq 0$. *Return prefix sums* $\langle s_1, s_2, \ldots, s_n \rangle$ }

if $n = 1$ *then*
    $s_1 \leftarrow x_1$
else
    parallel for $i \leftarrow 1$ *to* $n/2$ *do*
        $y_i \leftarrow x_{2i-1} \oplus x_{2i}$
    $\langle z_1, z_2, \ldots, z_{n/2} \rangle \leftarrow$ *Prefix-Sum*($\langle y_1, y_2, \ldots, y_{n/2} \rangle$, $\oplus$)
    parallel for $i \leftarrow 1$ *to* n *do*
        if $i = 1$ *then* $s_1 \leftarrow x_1$
        else if $i = even$ *then* $s_i \leftarrow z_{i/2}$
            else $s_i \leftarrow z_{(i-1)/2} \oplus x_i$
return $\langle s_1, s_2, \ldots, s_n \rangle$

Work:

$$T_1(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ T_1\left(\frac{n}{2}\right) + \Theta(n), & otherwise. \end{cases}$$

$$= \Theta(n)$$

Span:

$$T_\infty(n) = \begin{cases} \Theta(1), & if\ n = 1, \\ T_\infty\left(\frac{n}{2}\right) + \Theta(1), & otherwise. \end{cases}$$

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

Parallelism: $\frac{T_1(n)}{T_\infty(n)} = \Theta\left(\frac{n}{\log n}\right)$

Observe that we have assumed here that a *parallel for loop* can be executed in $\Theta(1)$ time. But recall that *cilk_for* is implemented using divide-and-conquer, and so in practice, it will take $\Theta(\log n)$ time. In that case, we will have $T_\infty(n) = \Theta(\log^2 n)$, and parallelism $= \Theta\left(\frac{n}{\log^2 n}\right)$.

Parallel Partition

Parallel Partition

Input: An array $A[\,q : r\,]$ of distinct elements, and an element x from $A[\,q : r\,]$.

Output: Rearrange the elements of $A[\,q : r\,]$, and return an index $k \in [\,q, r\,]$, such that all elements in $A[\,q : k - 1\,]$ are smaller than x, all elements in A$[\,k + 1 : r\,]$ are larger than x, and $A[\,k\,] = x$.

```
Par-Partition ( A[ q : r ], x )
n ← r − q + 1
if n = 1 then return q
array  B[ 0: n − 1 ], lt[ 0: n − 1 ], gt[ 0: n − 1 ]
parallel for i ← 0 to n − 1 do
     B[ i ] ← A[ q + i ]
     if B[ i ] < x then lt[ i ] ← 1 else lt[ i ] ← 0
     if B[ i ] > x then gt[ i ] ← 1 else gt[ i ] ← 0
lt [ 0: n − 1 ] ← Par-Prefix-Sum (  lt[ 0: n − 1 ],  + )
gt[ 0: n − 1 ] ← Par-Prefix-Sum (  gt[ 0: n − 1 ],  + )
k ← q + lt [ n − 1 ],  A[ k ] ← x
parallel for i ← 0 to n − 1 do
     if B[ i ] < x then A[ q + lt [ i ] − 1 ] ← B[ i ]
     else if B[ i ] > x then A[ k + gt [ i ] ] ← B[ i ]
return k
```

Parallel Partition

A:	9	5	7	11	1	3	8	14	4	21	$x = 8$

Parallel Partition

A:

9	5	7	11	1	3	8	14	4	21

***x* = 8**

B:

0	1	2	3	4	5	6	7	8	9
9	5	7	11	1	3	8	14	4	21

lt:

0	1	2	3	4	5	6	7	8	9
0	1	1	0	1	1	0	0	1	0

gt:

0	1	2	3	4	5	6	7	8	9
1	0	0	1	0	0	0	1	0	1

Parallel Partition

***A*:**	9	5	7	11	1	3	8	14	4	21

***x* = 8**

	0	1	2	3	4	5	6	7	8	9
***B*:**	9	5	7	11	1	3	8	14	4	21

	0	1	2	3	4	5	6	7	8	9
***lt*:**	0	1	1	0	1	1	0	0	1	0
***lt*:**	0	1	2	2	3	4	4	4	5	5

prefix sum

	0	1	2	3	4	5	6	7	8	9
***gt*:**	1	0	0	1	0	0	0	1	0	1
***gt*:**	1	1	1	2	2	2	2	3	3	4

prefix sum

Parallel Partition

***A*:**	9	5	7	11	1	3	8	14	4	21

***x* = 8**

	0	1	2	3	4	5	6	7	8	9
***B*:**	9	5	7	11	1	3	8	14	4	21

	0	1	2	3	4	5	6	7	8	9
***lt*:**	0	1	1	0	1	1	0	0	1	0
***lt*:**	0	1	2	2	3	4	4	4	5	5

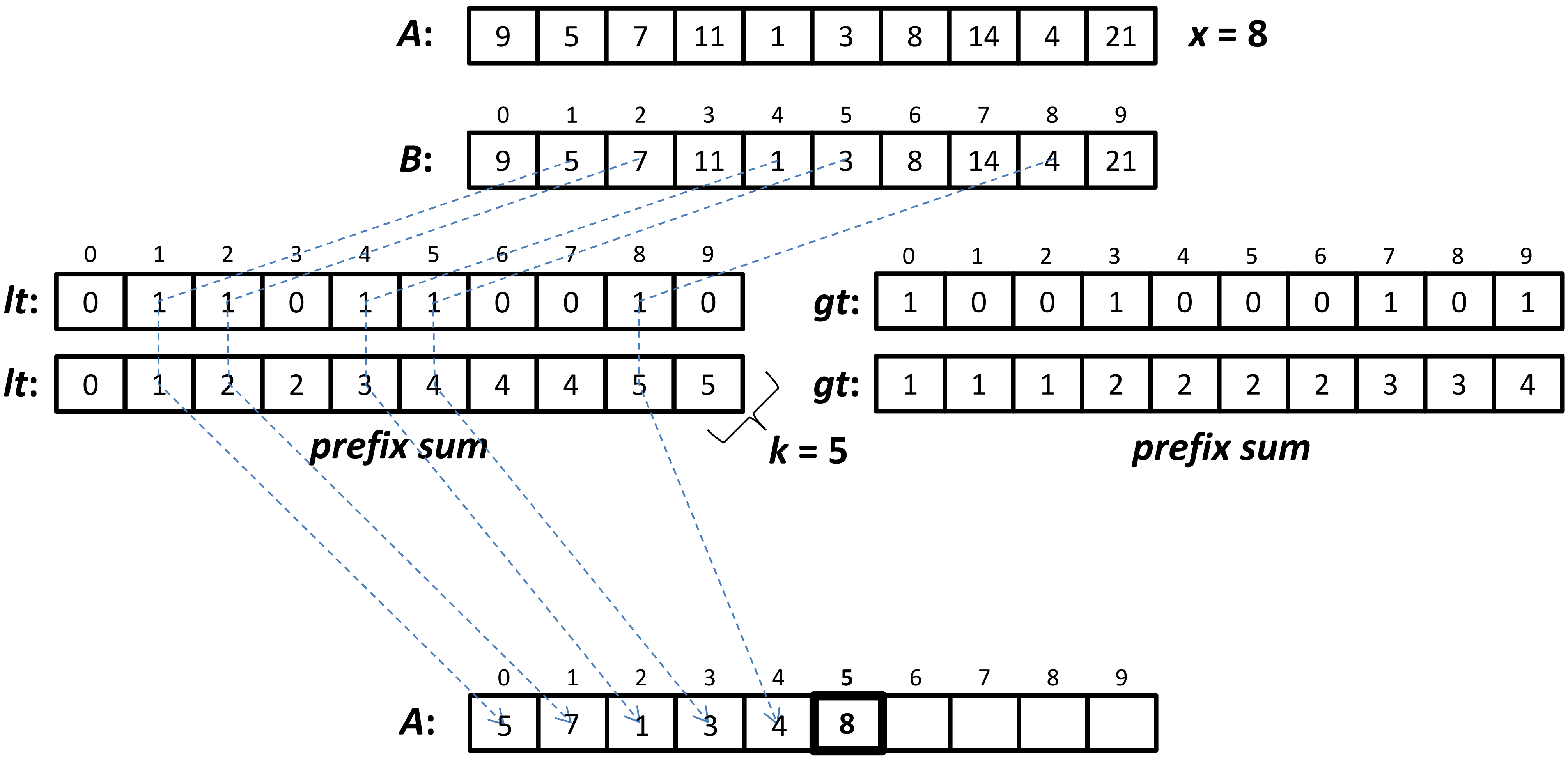
prefix sum

	0	1	2	3	4	5	6	7	8	9
***gt*:**	1	0	0	1	0	0	0	1	0	1
***gt*:**	1	1	1	2	2	2	2	3	3	4

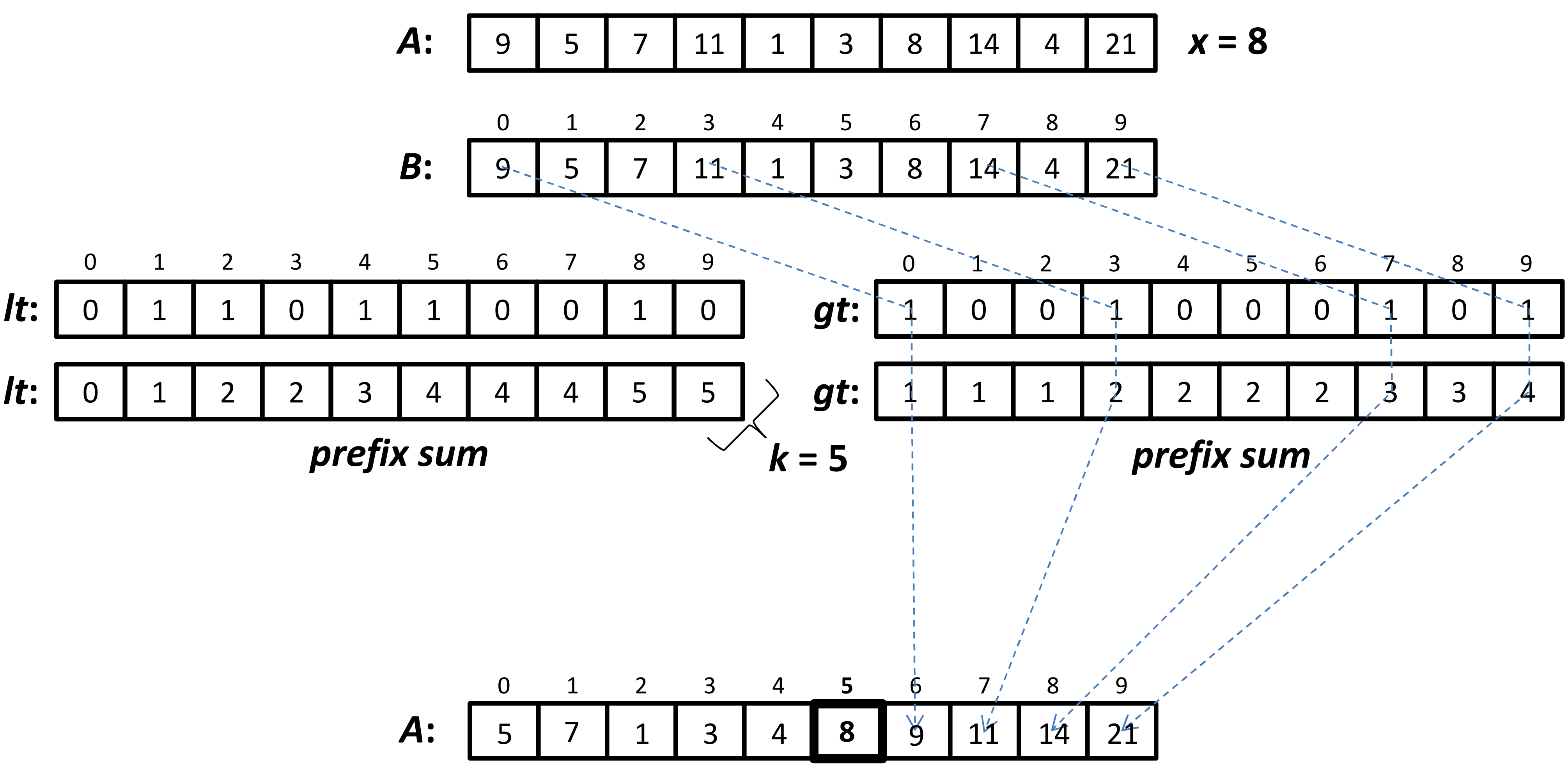
prefix sum

	0	1	2	3	4	5	6	7	8	9
***A*:**										

Parallel Partition



Parallel Partition



Parallel Partition

***A*:**

9	5	7	11	1	3	8	14	4	21

$x = 8$

***B*:**

0	1	2	3	4	5	6	7	8	9
9	5	7	11	1	3	8	14	4	21

	0	1	2	3	4	5	6	7	8	9
***lt*:**	0	1	1	0	1	1	0	0	1	0
***lt*:**	0	1	2	2	3	4	4	4	5	5

prefix sum

$k = 5$

	0	1	2	3	4	5	6	7	8	9
***gt*:**	1	0	0	1	0	0	0	1	0	1
***gt*:**	1	1	1	2	2	2	2	3	3	4

prefix sum

***A*:**

0	1	2	3	4	**5**	6	7	8	9
5	7	1	3	4	**8**	9	11	14	21

Parallel Partition: Analysis

```
Par-Partition ( A[ q : r ], x )
1.  n ← r − q + 1
2.  if n = 1 then return q
3.  array  B[ 0: n − 1 ], lt[ 0: n − 1 ], gt[ 0: n − 1 ]
4.  parallel for i ← 0 to n − 1 do
5.      B[ i ] ← A[ q + i ]
6.      if B[ i ] < x then lt[ i ] ← 1 else lt[ i ] ← 0
7.      if B[ i ] > x then gt[ i ] ← 1 else gt[ i ] ← 0
8.  lt [ 0: n − 1 ] ← Par-Prefix-Sum (  lt[ 0: n − 1 ],  + )
9.  gt[ 0: n − 1 ] ← Par-Prefix-Sum (  gt[ 0: n − 1 ],  + )
10. k ← q + lt [ n − 1 ],  A[ k ] ← x
11. parallel for i ← 0 to n − 1 do
12.     if B[ i ] < x then A[ q + lt [ i ] − 1 ] ← B[ i ]
13.     else if B[ i ] > x then A[ k + gt [ i ] ] ← B[ i ]
14. return k
```

Work:

$$
\begin{aligned}
T_1(n) &= \Theta(n) && [\text{ lines } 1-7\text{ }]\\
&\quad + \Theta(n) && [\text{ lines } 8-9\text{ }]\\
&\quad + \Theta(n) && [\text{ lines } 10-14\text{ }]\\
&= \Theta(n)
\end{aligned}
$$

Span:

Assuming $\log n$ depth for *parallel for* loops:

$$
\begin{aligned}
T_\infty(n) &= \Theta(\log n) && [\text{ lines } 1-7\text{ }]\\
&\quad + \Theta(\log^2 n) && [\text{ lines } 8-9\text{ }]\\
&\quad + \Theta(\log n) && [\text{ lines } 10-14\text{ }]\\
&= \Theta(\log^2 n)
\end{aligned}
$$

Parallelism: $\frac{T_1(n)}{T_\infty(n)} = \Theta\left(\frac{n}{\log^2 n}\right)$

Parallel Quicksort

Randomized Parallel QuickSort

Input: An array $A[\, q : r \,]$ of distinct elements.

Output: Elements of $A[\, q : r \,]$ sorted in increasing order of value.

```
Par-Randomized-QuickSort ( A[ q : r ] )
  n ← r − q + 1
  if n ≤ 30 then
      sort A[ q : r ] using any sorting algorithm
  else
      select a random element x from A[ q : r ]
      k ← Par-Partition ( A[ q : r ], x )
      spawn Par-Randomized-QuickSort ( A[ q : k − 1 ] )
      Par-Randomized-QuickSort ( A[ k + 1 : r ] )
      sync
```

Randomized Parallel QuickSort: Analysis

```
Par-Randomized-QuickSort ( A[ q : r ] )
1. n ← r − q + 1
2. if n ≤ 30 then
3.        sort A[ q : r ] using any sorting algorithm
4. else
5.        select a random element x from A[ q : r ]
6.        k ← Par-Partition ( A[ q : r ], x )
7.        spawn Par-Randomized-QuickSort ( A[ q : k − 1 ] )
8.        Par-Randomized-QuickSort ( A[ k + 1 : r ] )
9.        sync
```

Lines 1—6 take $\Theta(\log^2 n)$ parallel time and perform $\Theta(n)$ work.

Also the recursive spawns in lines 7—8 work on disjoint parts of $A[\, q : r \,]$. So the upper bounds on the parallel time and the total work in each level of recursion are $\Theta(\log^2 n)$ and $\Theta(n)$, respectively.

Hence, if D is the *recursion depth* of the algorithm, then

$$T_1(n) = \mathrm{O}(nD) \text{ and } T_\infty(n) = \mathrm{O}(D \log^2 n)$$

Randomized Parallel QuickSort: Analysis

```
Par-Randomized-QuickSort ( A[ q : r ] )
n ← r − q + 1
if n ≤ 30 then
      sort A[ q : r ] using any sorting algorithm
else
      select a random element x from A[ q : r ]
      k ← Par-Partition (  A[ q : r ],  x )
      spawn Par-Randomized-QuickSort ( A[ q : k − 1 ] )
      Par-Randomized-QuickSort ( A[ k + 1 : r ] )
      sync
```

We already proved that w.h.p. recursion depth, $D = \mathrm{O}(\log n)$.

Hence, with high probability,

$$T_1(n) = \mathrm{O}(n \log n) \text{ and } T_\infty(n) = \mathrm{O}(\log^3 n)$$