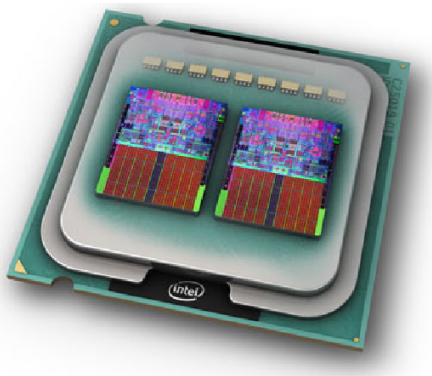
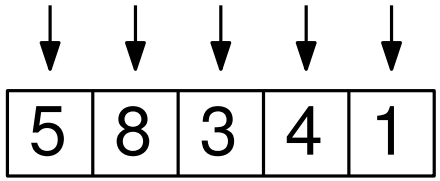
ICS 443: Parallel Algorithms

Prof. Nodari Sitchinava





Lecture 3: Brent's Scheduling Principle

Last Time: EREW Minimum

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

  if $i = j$ **then**

    return $a[i]$

  else

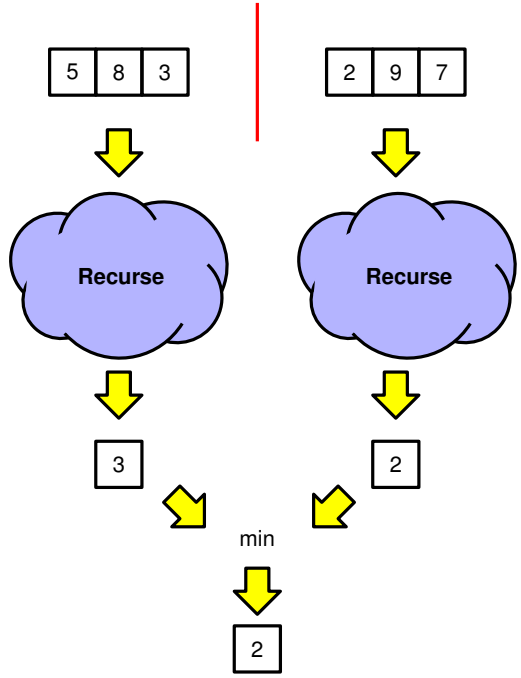
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$; $rmid = mid + 1$ — $O(1)$

    in parallel do — $T(n)$

      $left$ = MIN($a[i..mid]$) — $T(n/2)$

      $right$ = MIN($a[rmid..j]$) — $T(n/2)$

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



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

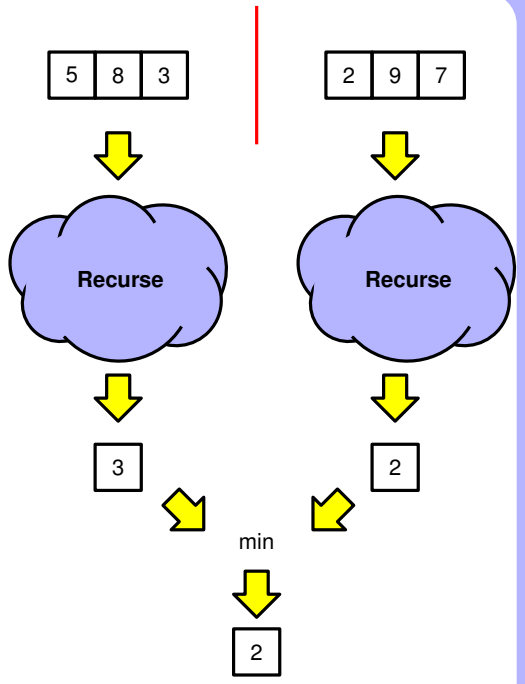
$$T(n) = O(1) + \max \left\{ \begin{array}{c} T(n/2) \\ T(n/2) \end{array} \right\} + O(1)$$

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

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

Saving Space

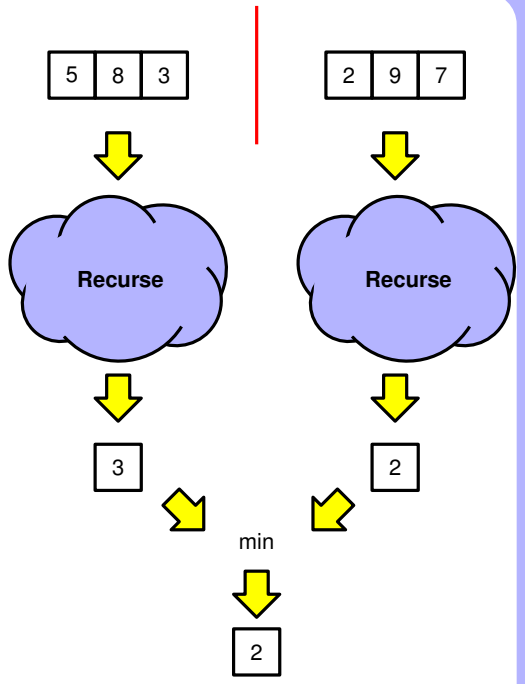
```
procedure MIN(a[i..j])
  if i = j then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋; rmid = mid + 1
      in parallel do
          left = MIN(a[i..mid])
          right = MIN(a[rmid..j])
      return min(left, right)
```



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

Saving Space

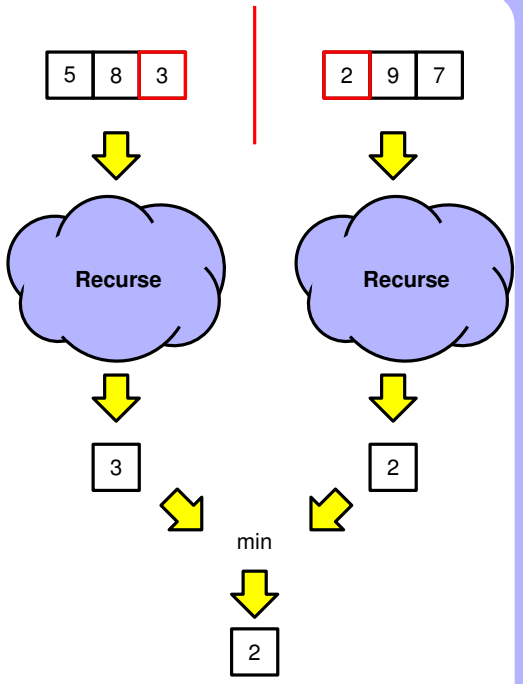
procedure MIN($a[i..j]$)
  if $i = j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[mid]$ = MIN($a[i..mid]$)
      $a[rmid]$ = MIN($a[rmid..j]$)
    return min($left$, $right$)



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

Saving Space

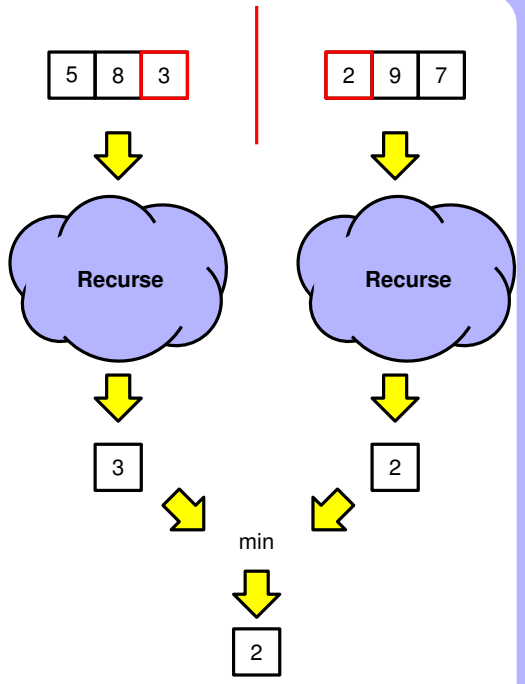
procedure MIN($a[i..j]$)
  if $i = j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[mid]$ = MIN($a[i..mid]$)
      $a[rmid]$ = MIN($a[rmid..j]$)
    return min($left$, $right$)



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

Saving Space

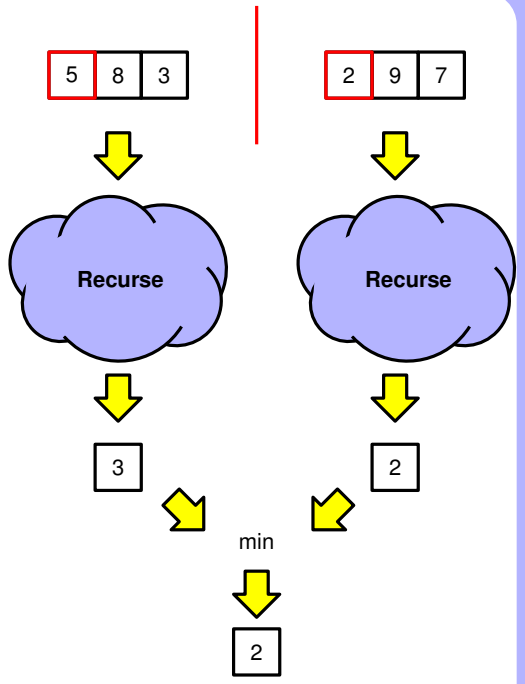
procedure MIN($a[i..j]$)
  if $i = j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[mid]$ = MIN($a[i..mid]$)
      $a[rmid]$ = MIN($a[rmid..j]$)
    return min($a[mid], a[rmid]$)



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

Saving Space

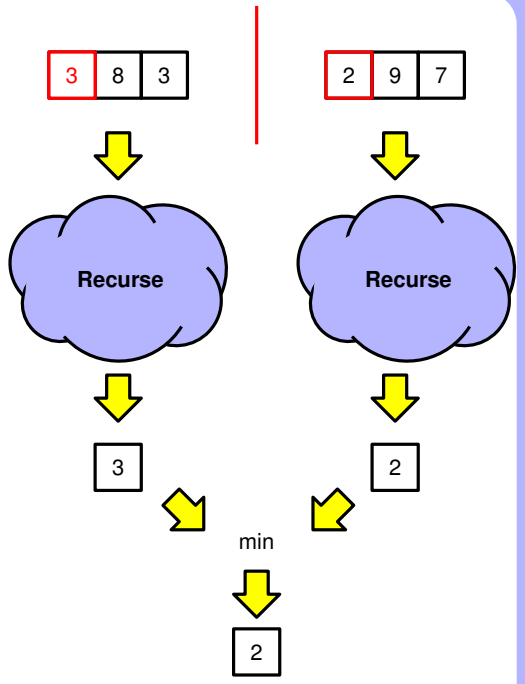
procedure MIN($a[i..j]$)
  if $i = j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[i]$ = MIN($a[i..mid]$)
      $a[rmid]$ = MIN($a[rmid..j]$)
    return min($a[i]$, $a[rmid]$)



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

Saving Space

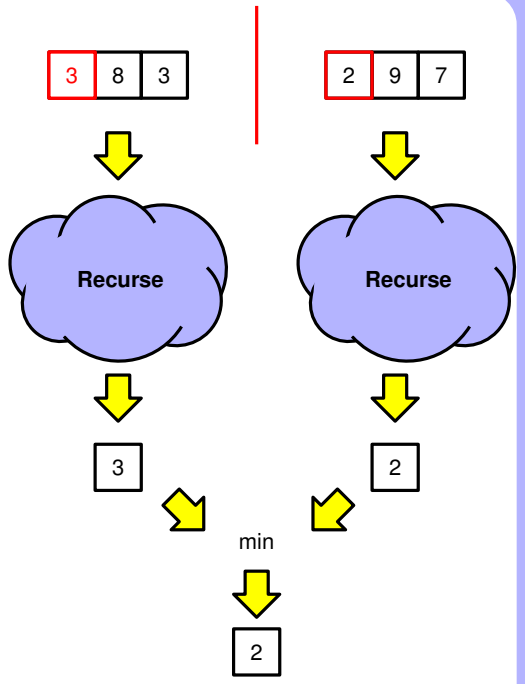
procedure MIN($a[i..j]$)
  if $i = j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[i]$ = MIN($a[i..mid]$)
      $a[rmid]$ = MIN($a[rmid..j]$)
    return min($a[i]$, $a[rmid]$)



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

Saving Space

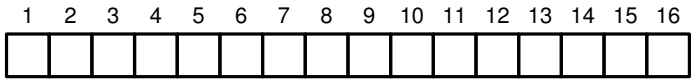
procedure MIN($a[i..j]$)
  if $i = j$ **then**
    return $a[i]$
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[i]$ = MIN($a[i..mid]$)
      $a[rmid]$ = MIN($a[rmid..j]$)
    return min($a[i]$, $a[rmid]$)



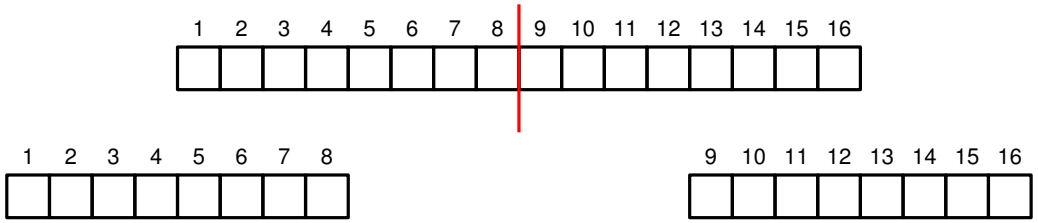
Runtime of MIN($a[1..n]$)?

Can we compute *mid* and *rmid* on the fly?

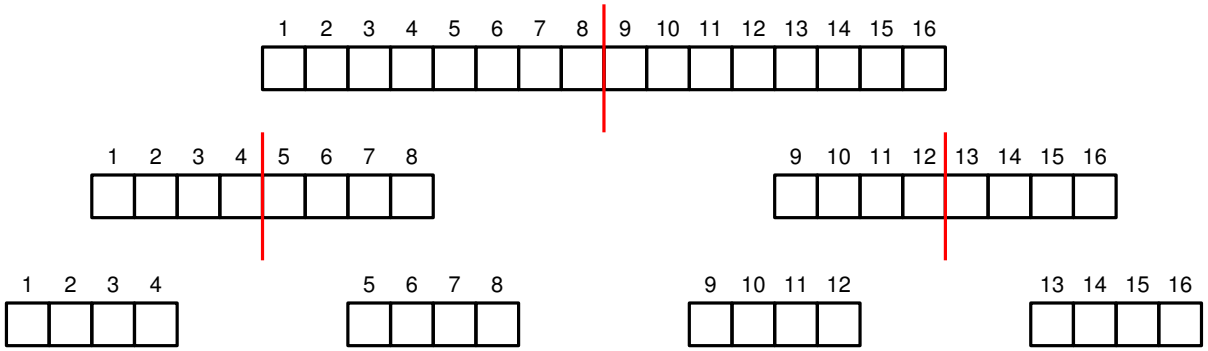
Computing *mid*



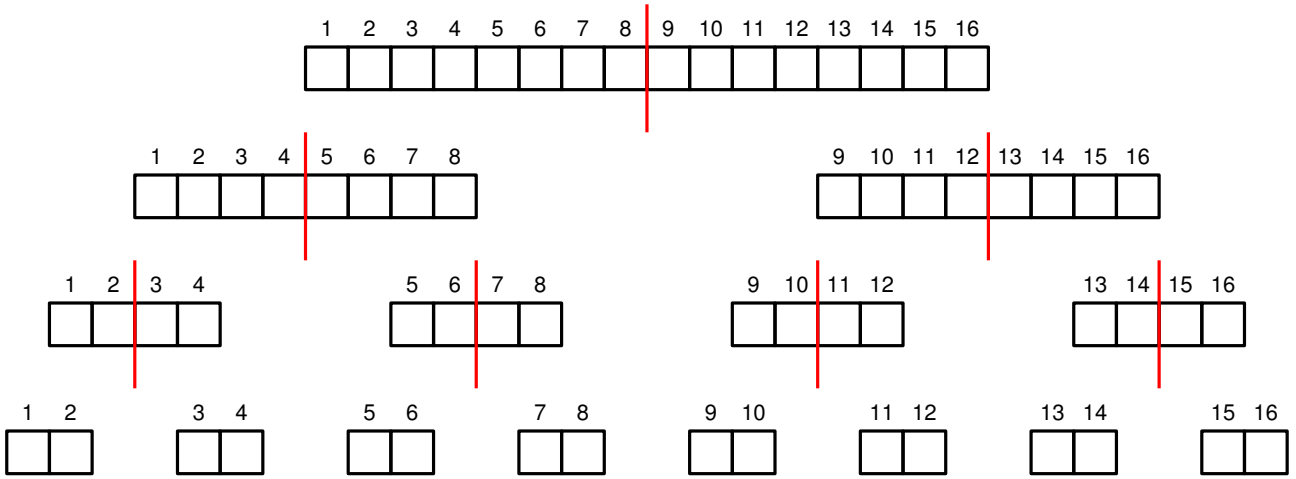
Computing *mid*



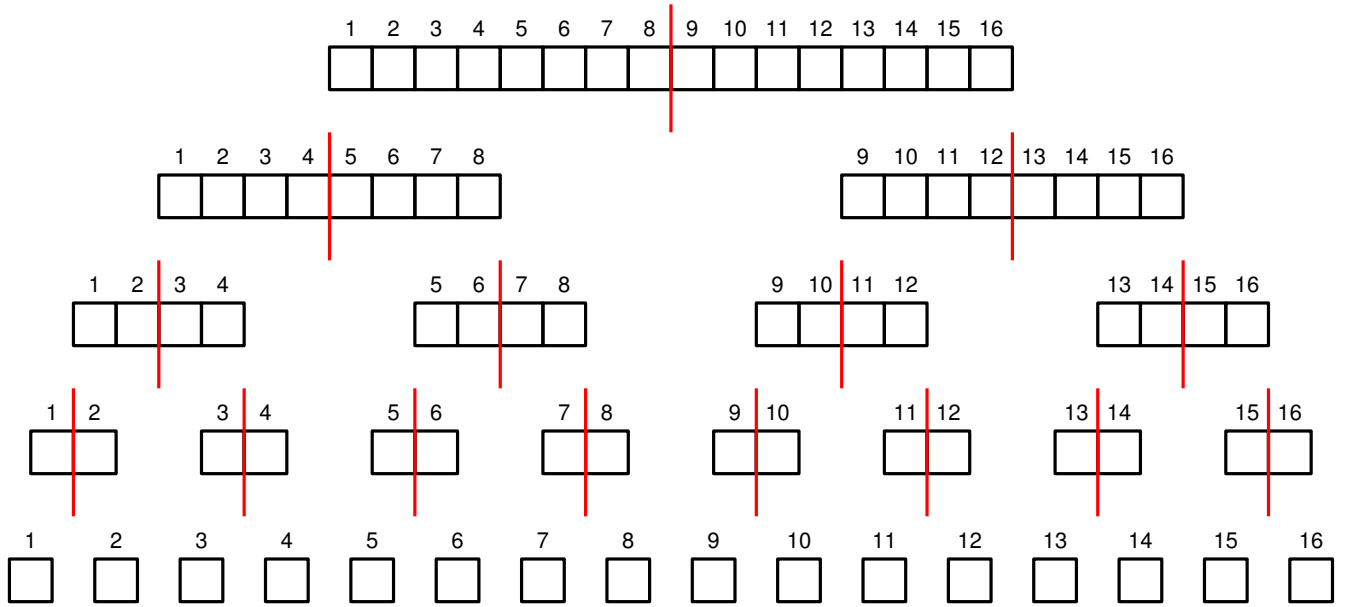
Computing *mid*



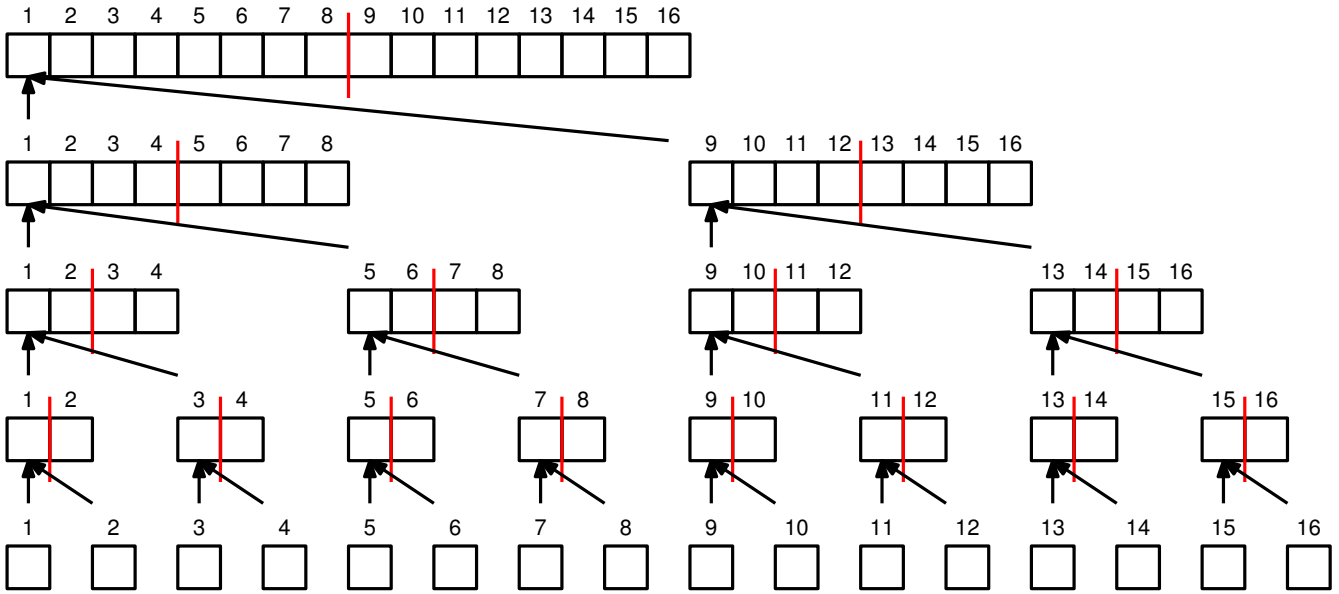
Computing *mid*



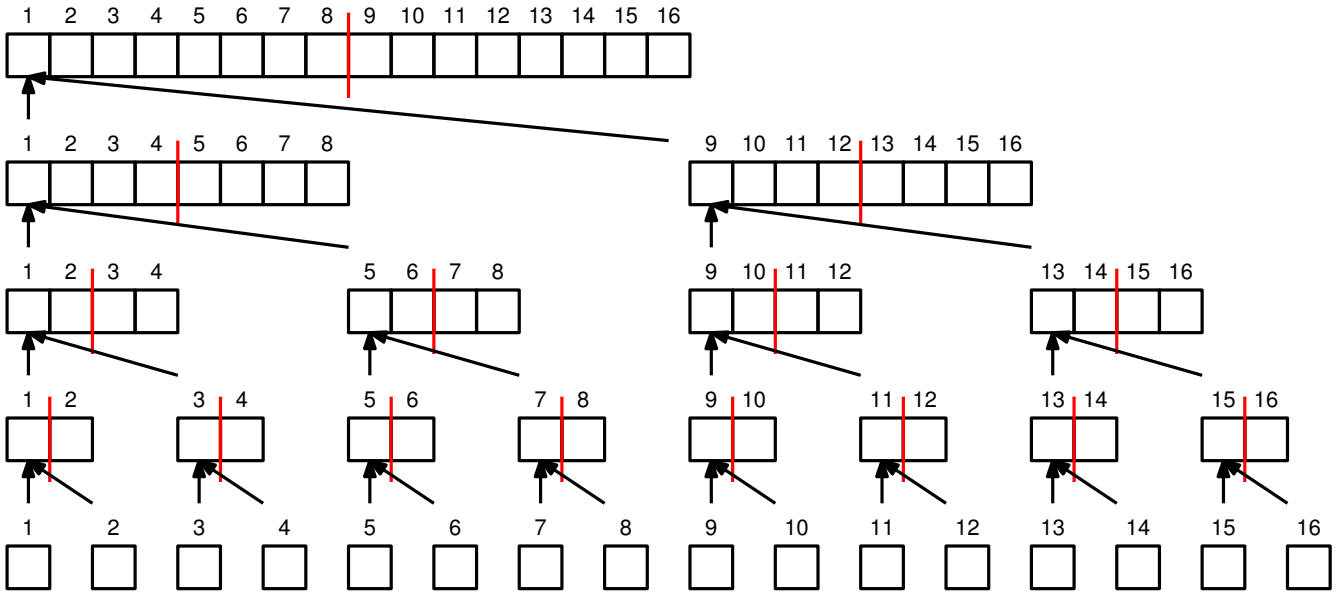
Computing *mid*



Computing *mid*

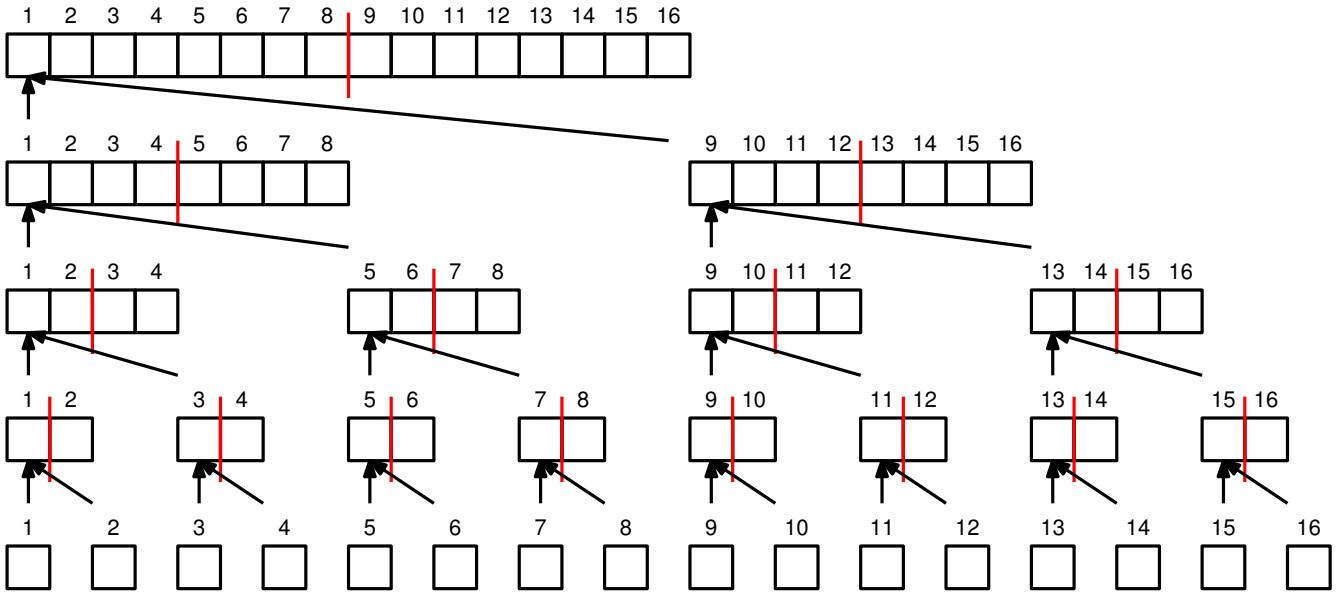


Computing *mid*



for $i = 1; i \leq n; i = i + 2$ **do**

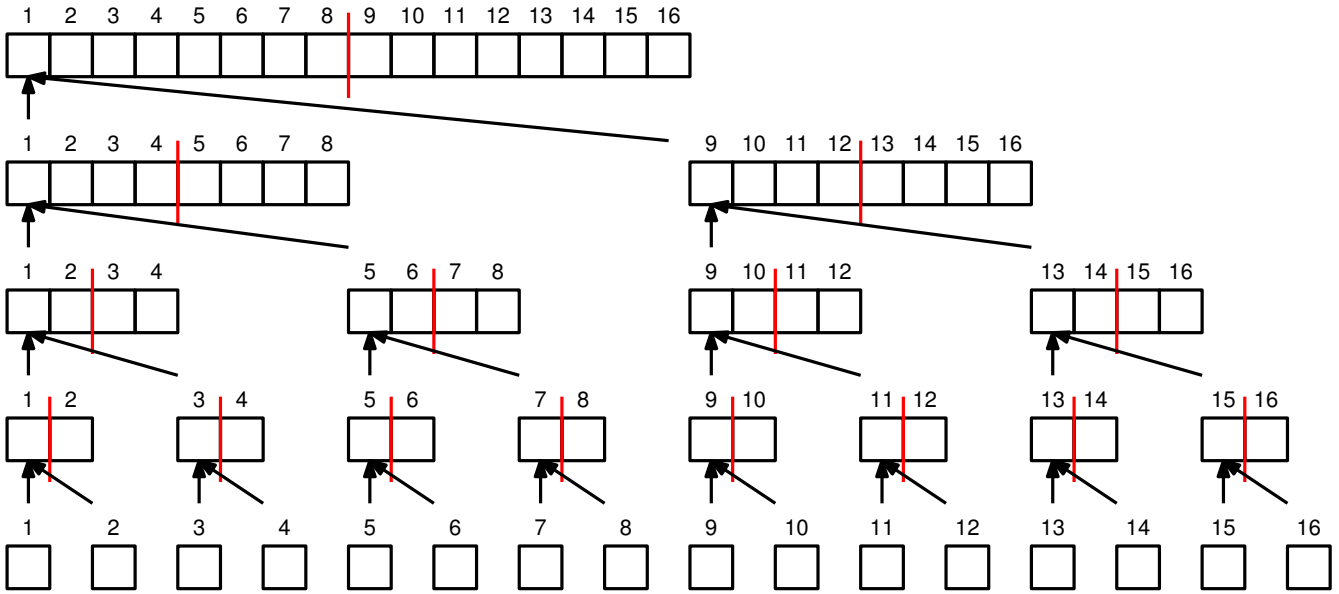
Computing *mid*



for $i = 1; i \leq n; i = i + 4$ **do**

for $i = 1; i \leq n; i = i + 2$ **do**

Computing *mid*

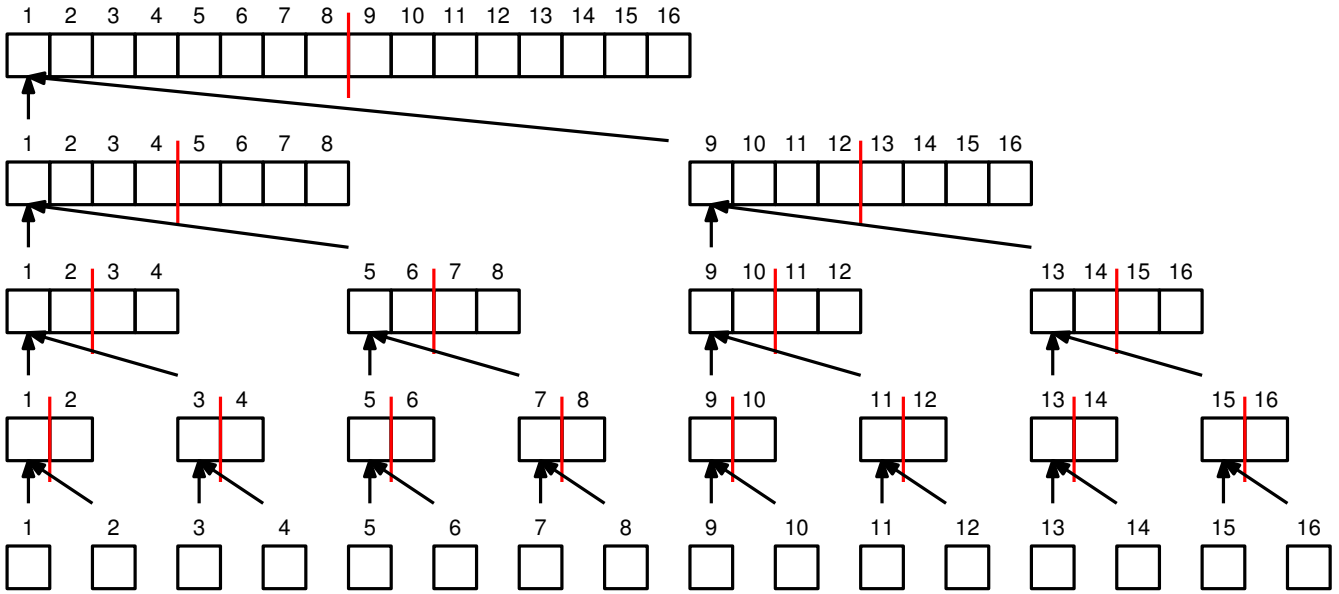


for $i = 1; i \leq n; i = i + 8$ **do**

for $i = 1; i \leq n; i = i + 4$ **do**

for $i = 1; i \leq n; i = i + 2$ **do**

Computing *mid*



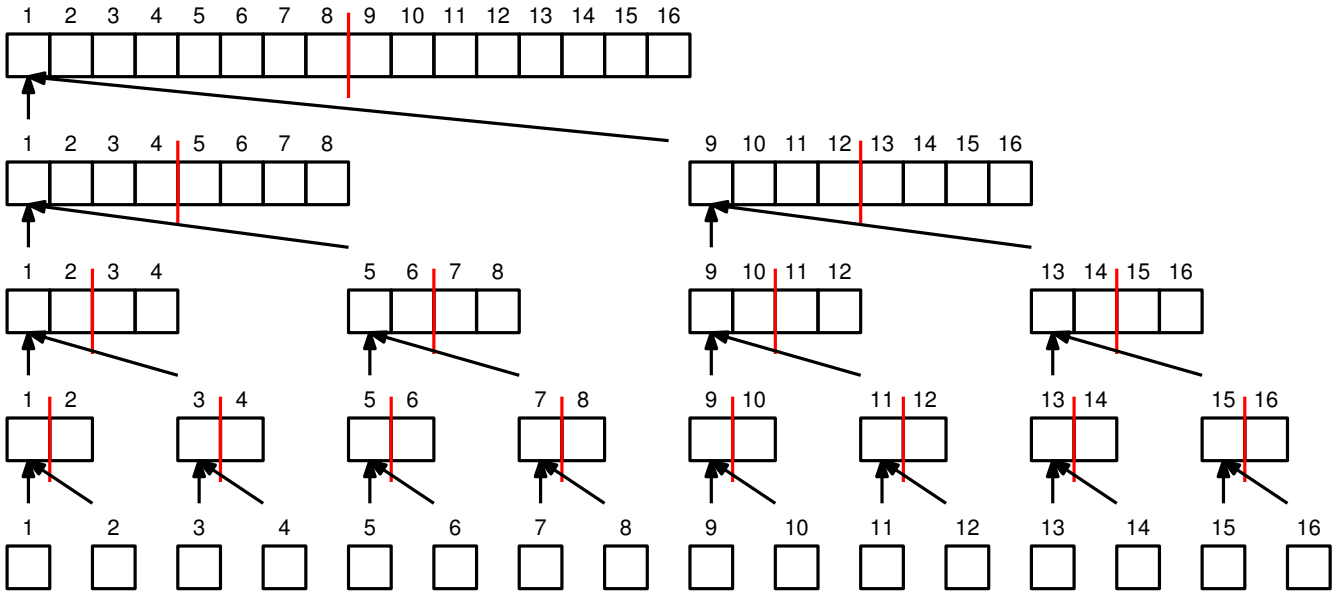
for $i = 1; i \leq n; i = i + 16$ **do**

for $i = 1; i \leq n; i = i + 8$ **do**

for $i = 1; i \leq n; i = i + 4$ **do**

for $i = 1; i \leq n; i = i + 2$ **do**

Computing *mid*



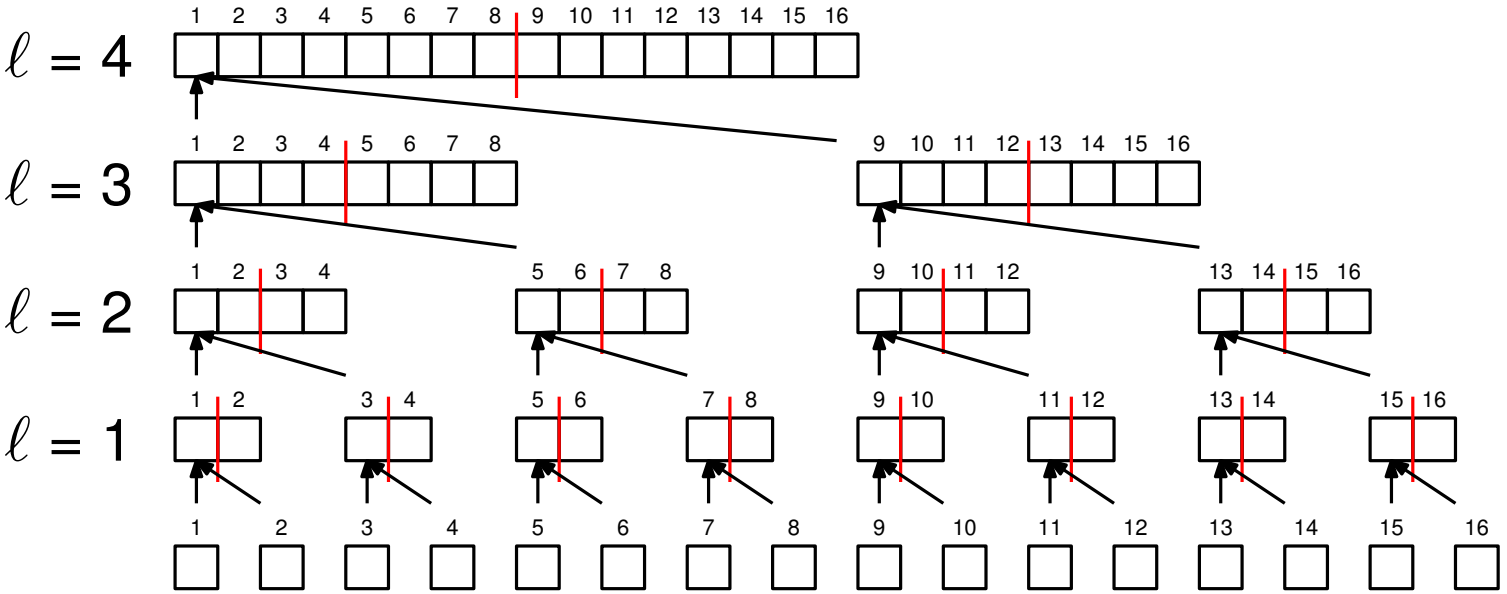
for $i = 1; i \leq n; i = i + 2^4$ **do**

for $i = 1; i \leq n; i = i + 2^3$ **do**

for $i = 1; i \leq n; i = i + 2^2$ **do**

for $i = 1; i \leq n; i = i + 2^1$ **do**

Computing *mid*



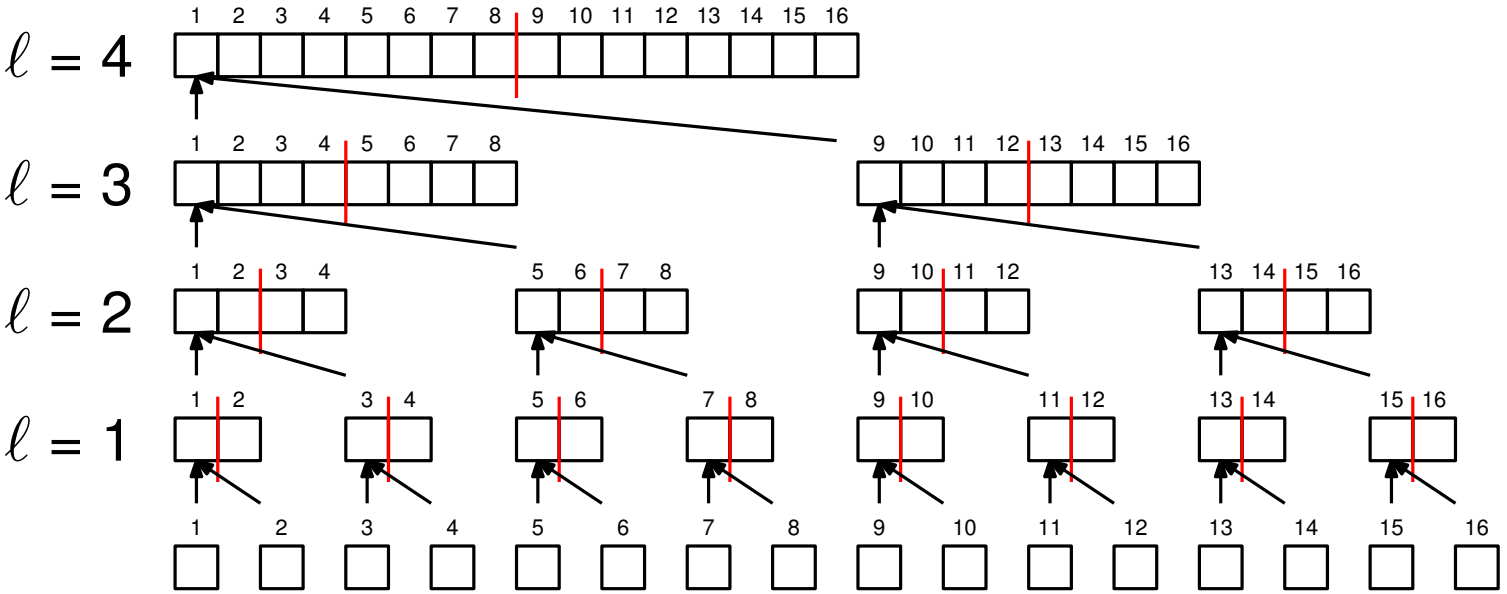
for $i = 1; i \leq n; i = i + 2^4$ **do**

for $i = 1; i \leq n; i = i + 2^3$ **do**

for $i = 1; i \leq n; i = i + 2^2$ **do**

for $i = 1; i \leq n; i = i + 2^1$ **do**

Computing *mid*



for $i = 1; i \leq n; i = i + 2^4$ **do**

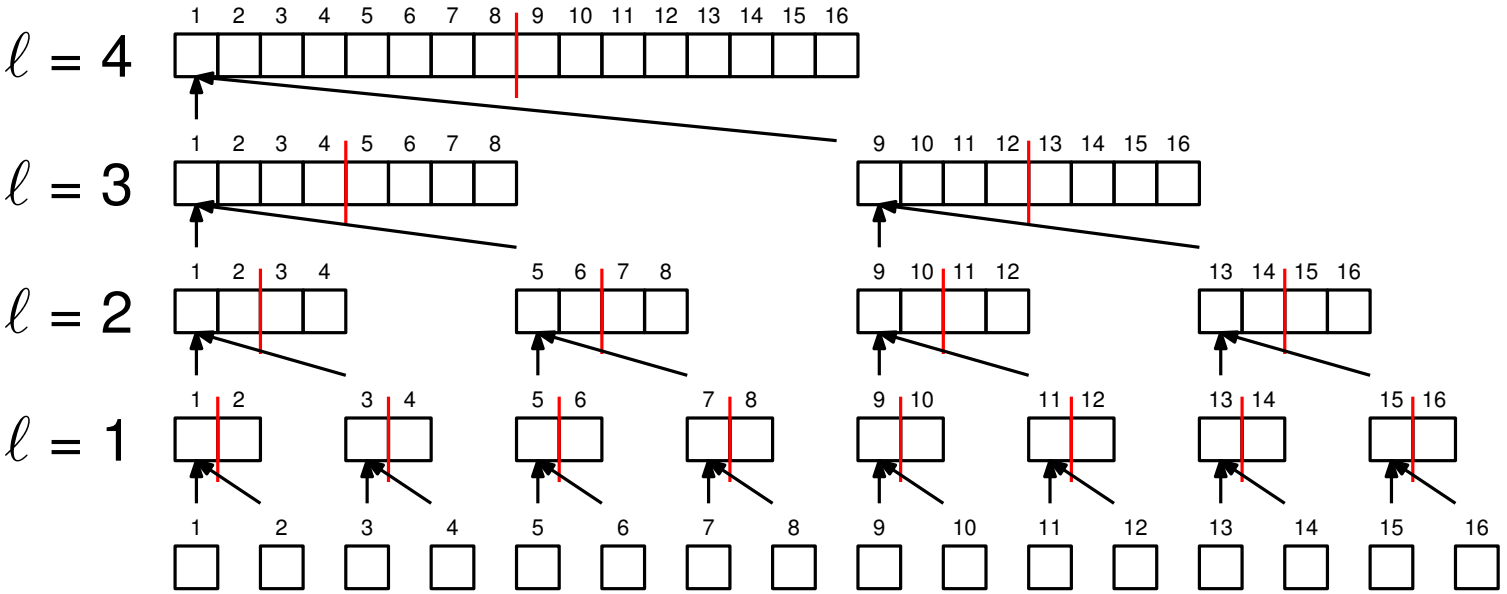
for $i = 1; i \leq n; i = i + 2^3$ **do**

for $i = 1; i \leq n; i = i + 2^2$ **do**

for $i = 1; i \leq n; i = i + 2^1$ **do**

for $i = 1; i \leq n; i = i + 2^{\ell}$ **do**

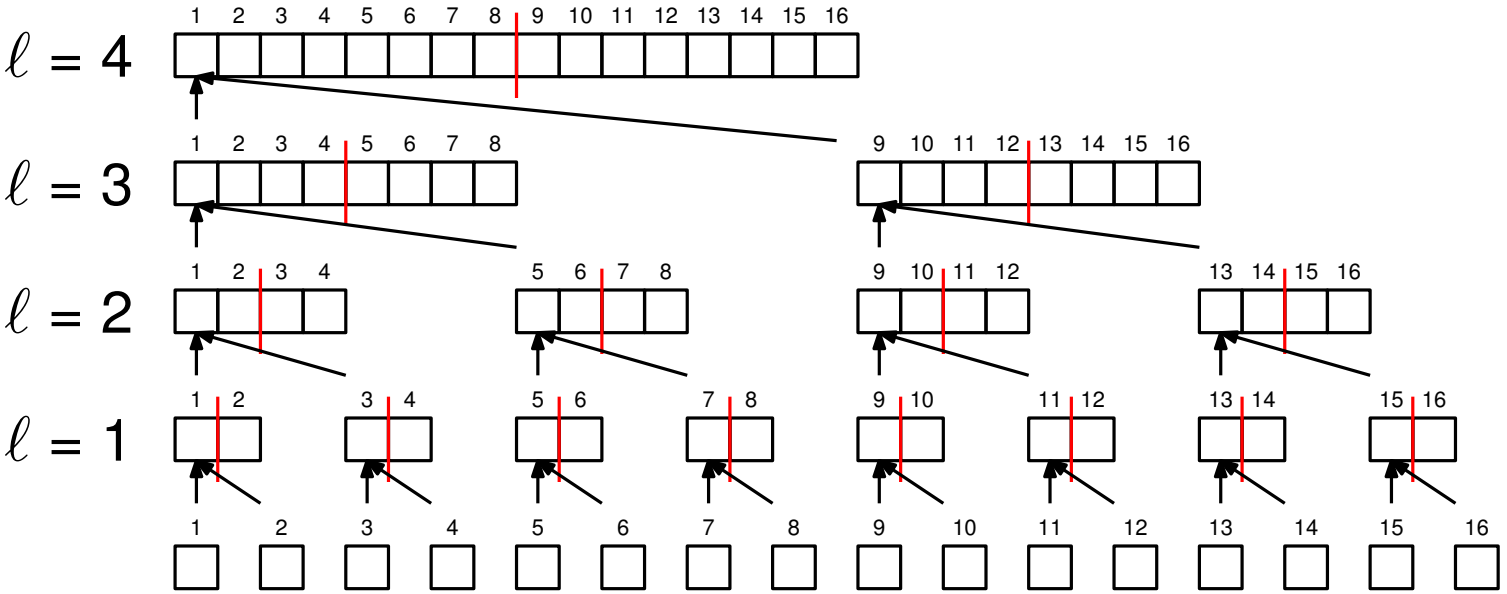
Computing *mid*



for $\ell = 1$ to $\log n$ **do**

  for $i = 1; i \leq n; i = i + 2^{\ell}$ **do**

Computing *mid*



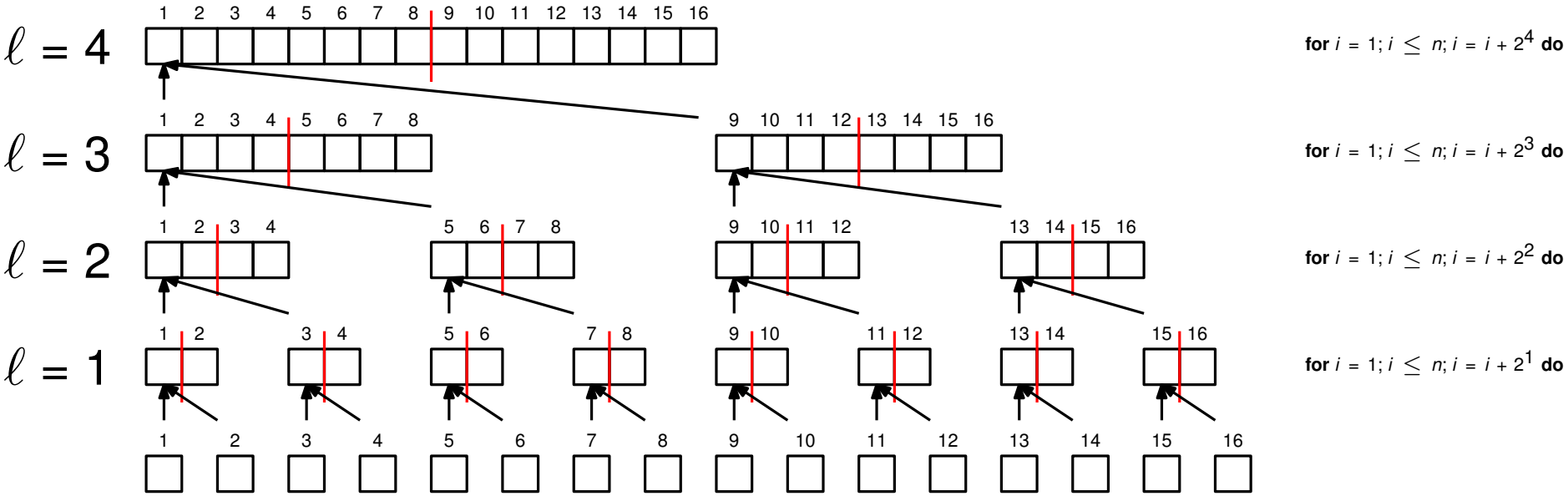
for $\ell = 1$ to $\log n$ **do**

  for $i = 1; i \leq n; i = i + 2^{\ell}$ **do**

    if $i + 2^{\ell-1} \leq n$ **then**

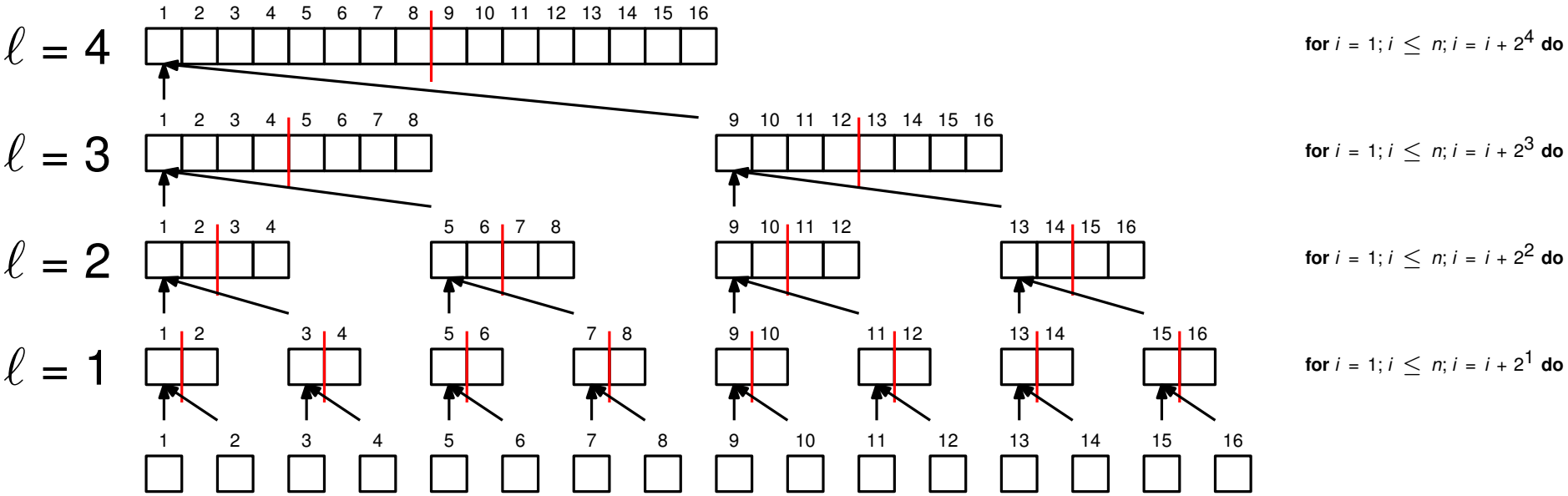
      $a[i] = \min(a[i], a[i + 2^{\ell-1}])$

Computing *mid*



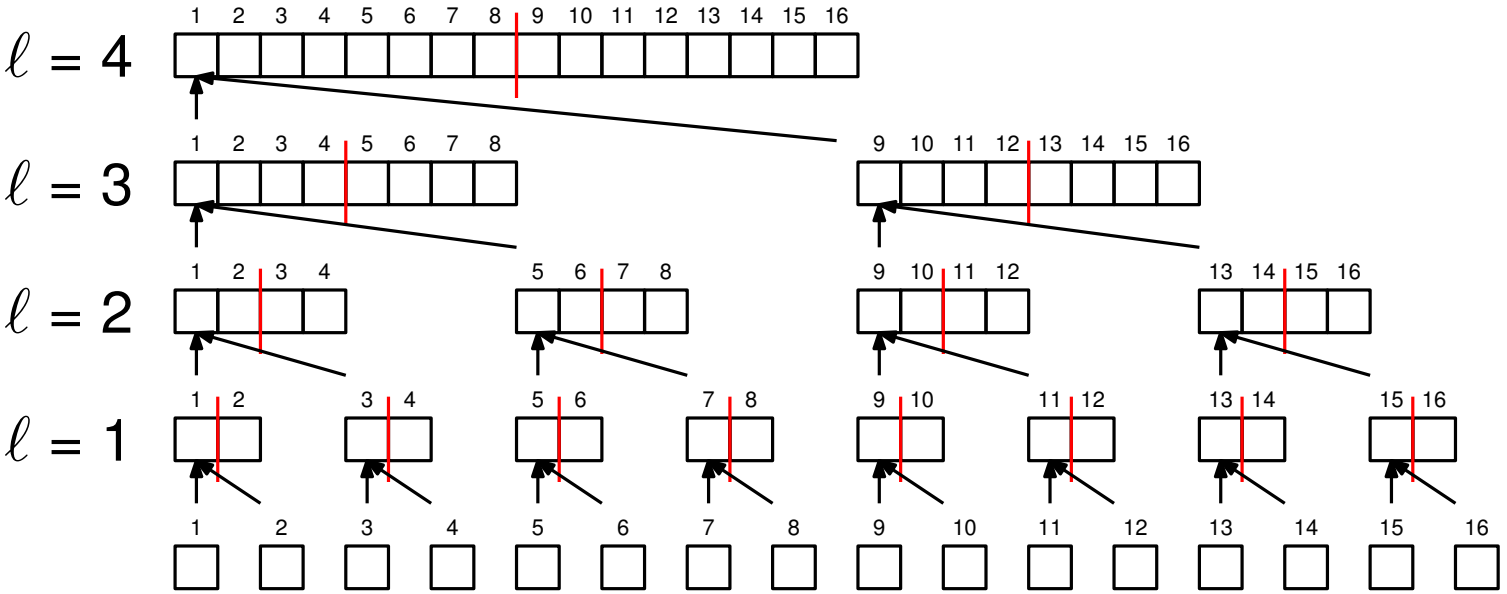
procedure MIN($a[1..n]$)
  for $\ell = 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

Computing *mid*



procedure MIN($a[1..n]$)
  for $\ell = 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

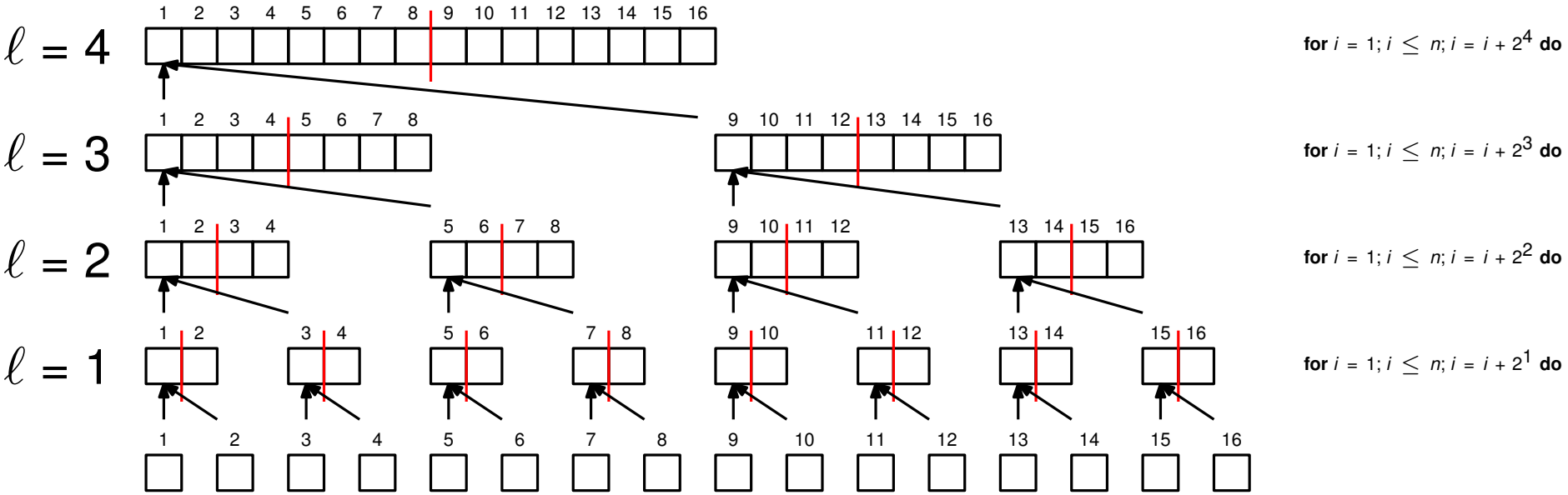
Computing *mid*



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

procedure MIN($a[1..n]$)
  for $\ell = 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

Computing *mid*



procedure MIN($a[1..n]$)
  for $\ell = 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

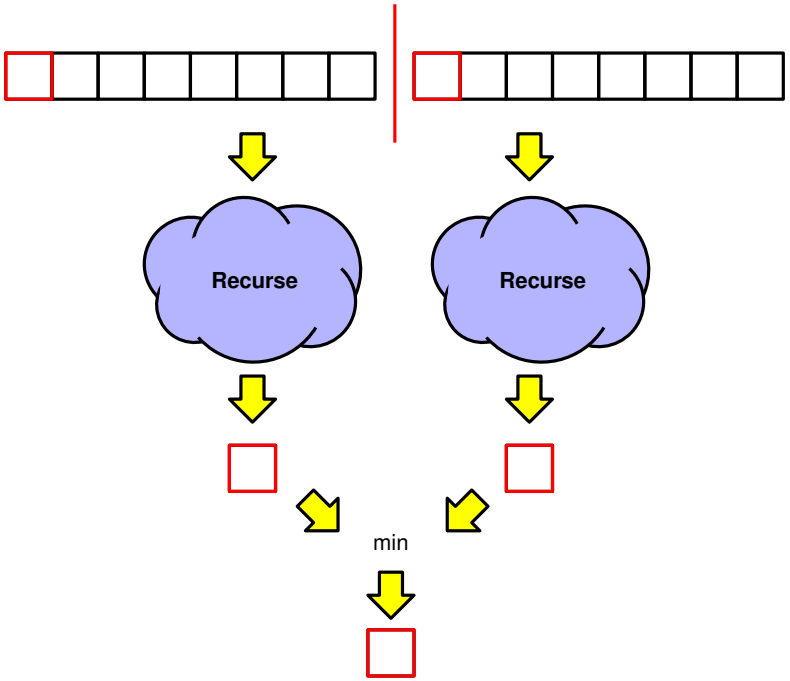
$O(1)$

$\sum_{\ell=1}^{\log n} O(1) = O(\log n)$

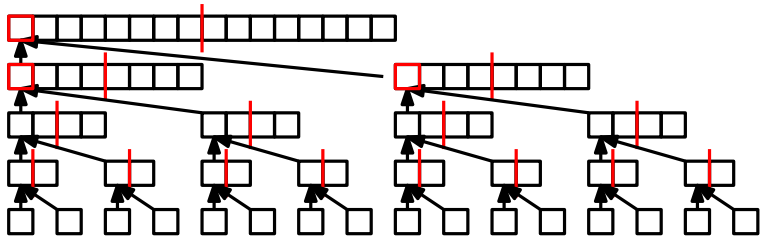
Runtime of MIN($a[1..n]$)?

Recursion vs. parallel **for** loop

```
procedure MIN(a[i..j])
  if i = j then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid])
          a[rmid] = MIN(a[rmid..j])
      return min(a[i], a[rmid])
```




```
procedure MIN(a[1..n])
   for ℓ = 1 to log n do
      for i = 1; i ≤ n; i = i + 2^ℓ in parallel do
         if i + 2^(ℓ−1) ≤ n then
            a[i] = min(a[i], a[i + 2^(ℓ−1)])
   return a[1]
```



Advantages of recursion vs. parallel **for** loop

- Every recursive program can be converted into an iterative one (and vice versa)

Advantages of recursion vs. parallel **for** loop

- Every recursive program can be converted into an iterative one (and vice versa)
- Recursion is slightly slower — $O(1)$ factor

Advantages of recursion vs. parallel **for** loop

- Every recursive program can be converted into an iterative one (and vice versa)
- Recursion is slightly slower — $O(1)$ factor
- Recursion is often easier to design and understand

Advantages of recursion vs. parallel **for** loop

- Every recursive program can be converted into an iterative one (and vice versa)
- Recursion is slightly slower — $O(1)$ factor
- Recursion is often easier to design and understand
- Analysis of recursive algorithm requires solving recurrences

Advantages of recursion vs. parallel **for** loop

- Every recursive program can be converted into an iterative one (and vice versa)
- Recursion is slightly slower — $O(1)$ factor
- Recursion is often easier to design and understand
- Analysis of recursive algorithm requires solving recurrences
- A programming language might support one more efficiently than another

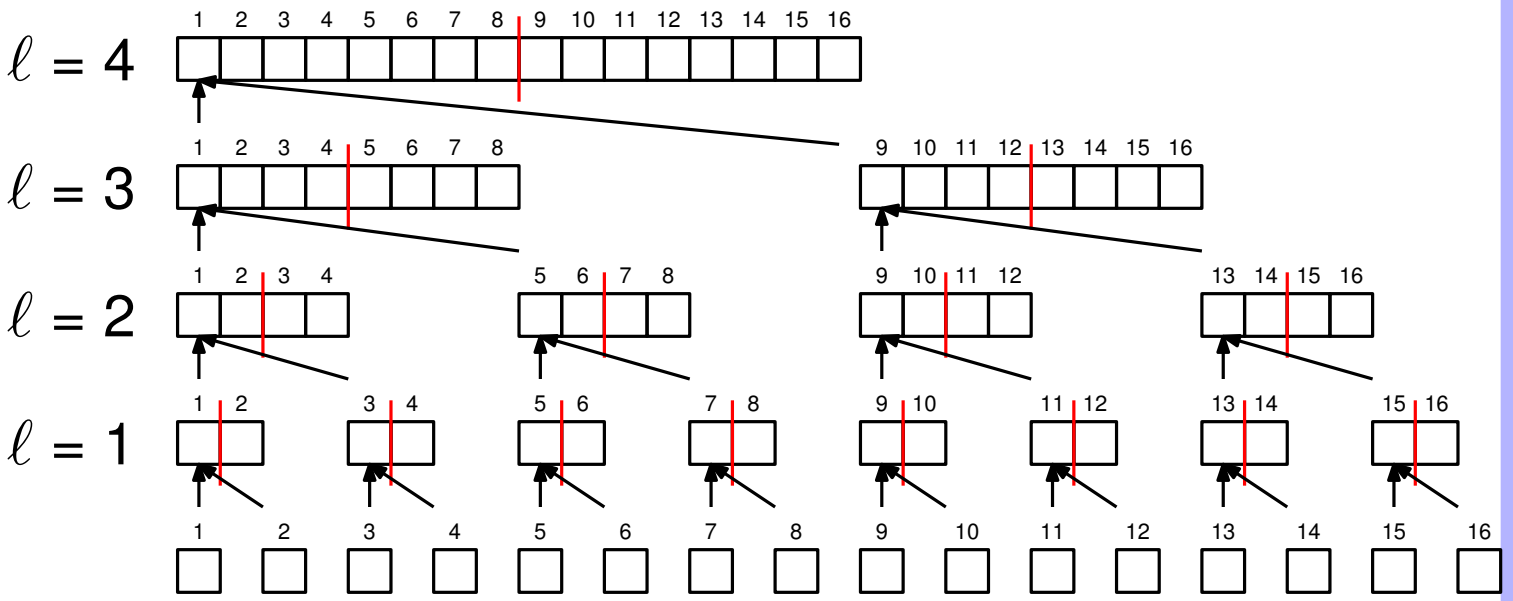
Advantages of recursion vs. parallel **for** loop

- Every recursive program can be converted into an iterative one (and vice versa)
- Recursion is slightly slower — $O(1)$ factor
- Recursion is often easier to design and understand
- Analysis of recursive algorithm requires solving recurrences
- A programming language might support one more efficiently than another
- Spawning n threads might be slow (for large n)
  - Hidden cost of the parallel **for** loop
  - At most $O(\log n)$, but often negligible

Fewer than n processors?

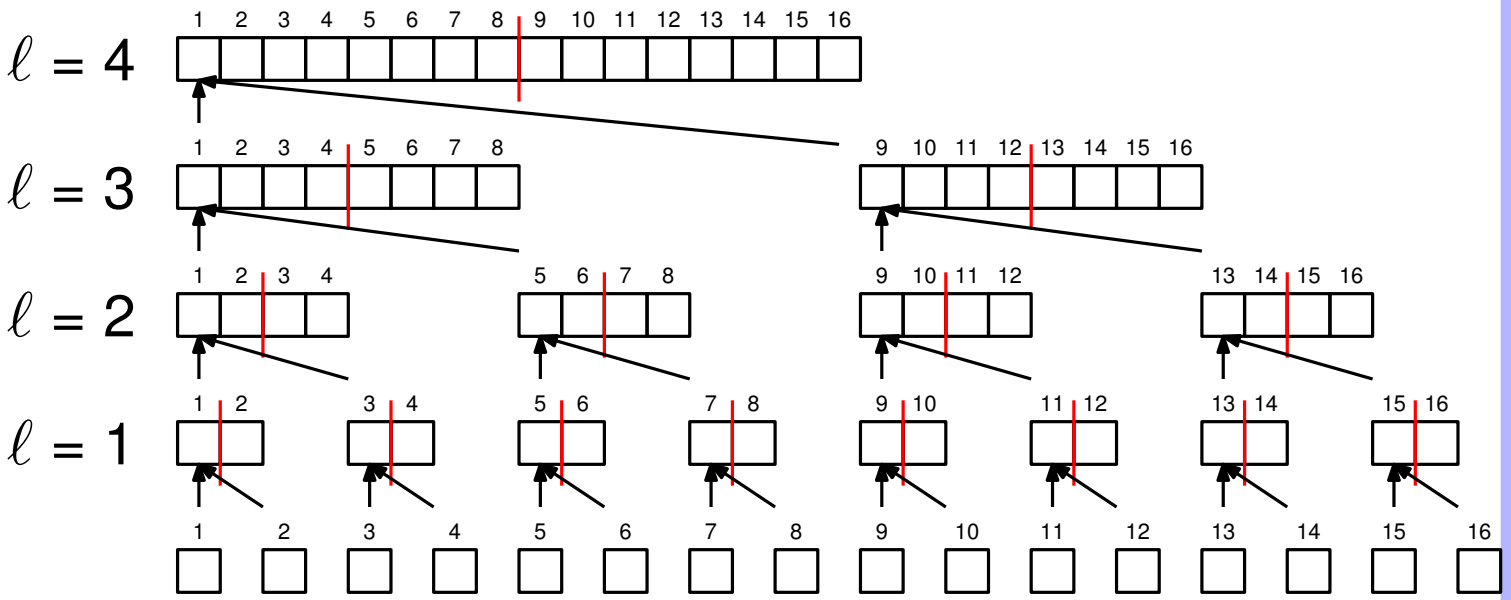
Fewer than n processors?

$n = 16$



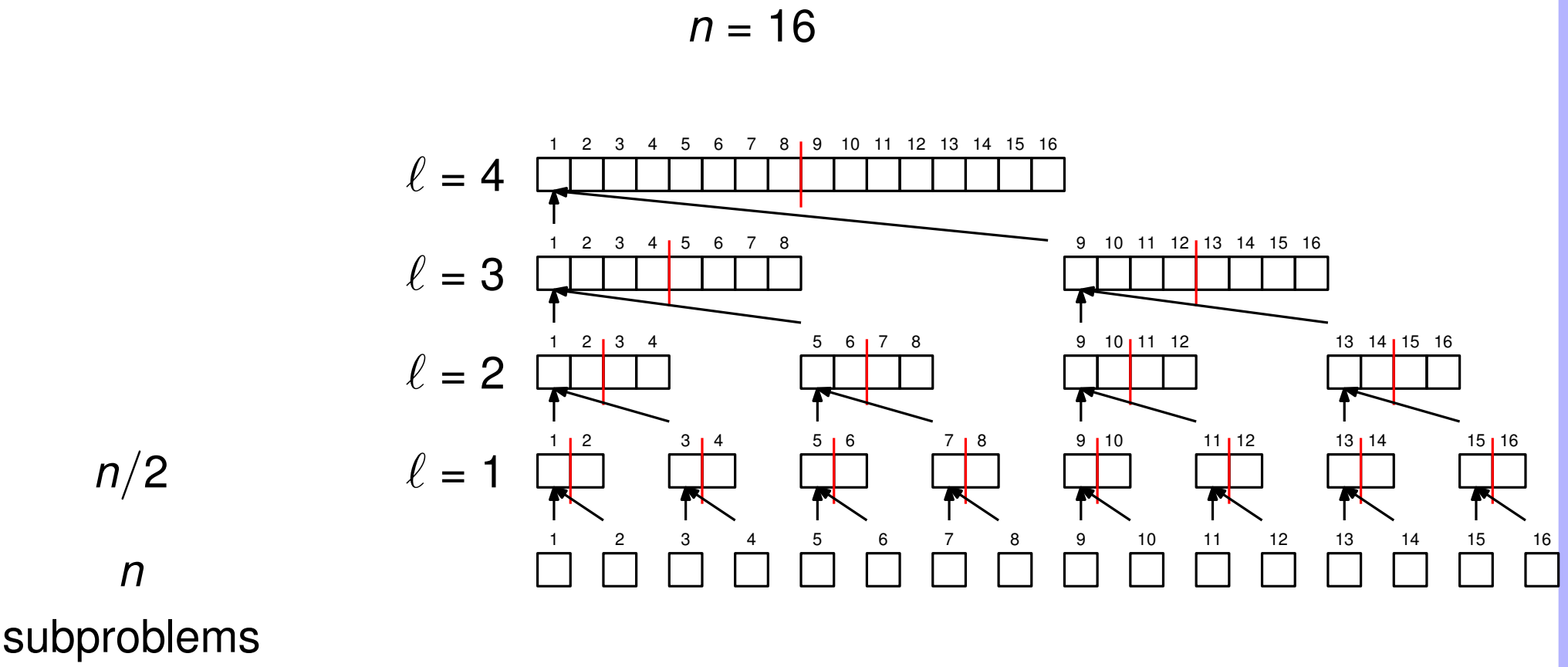
Fewer than n processors?

$n = 16$

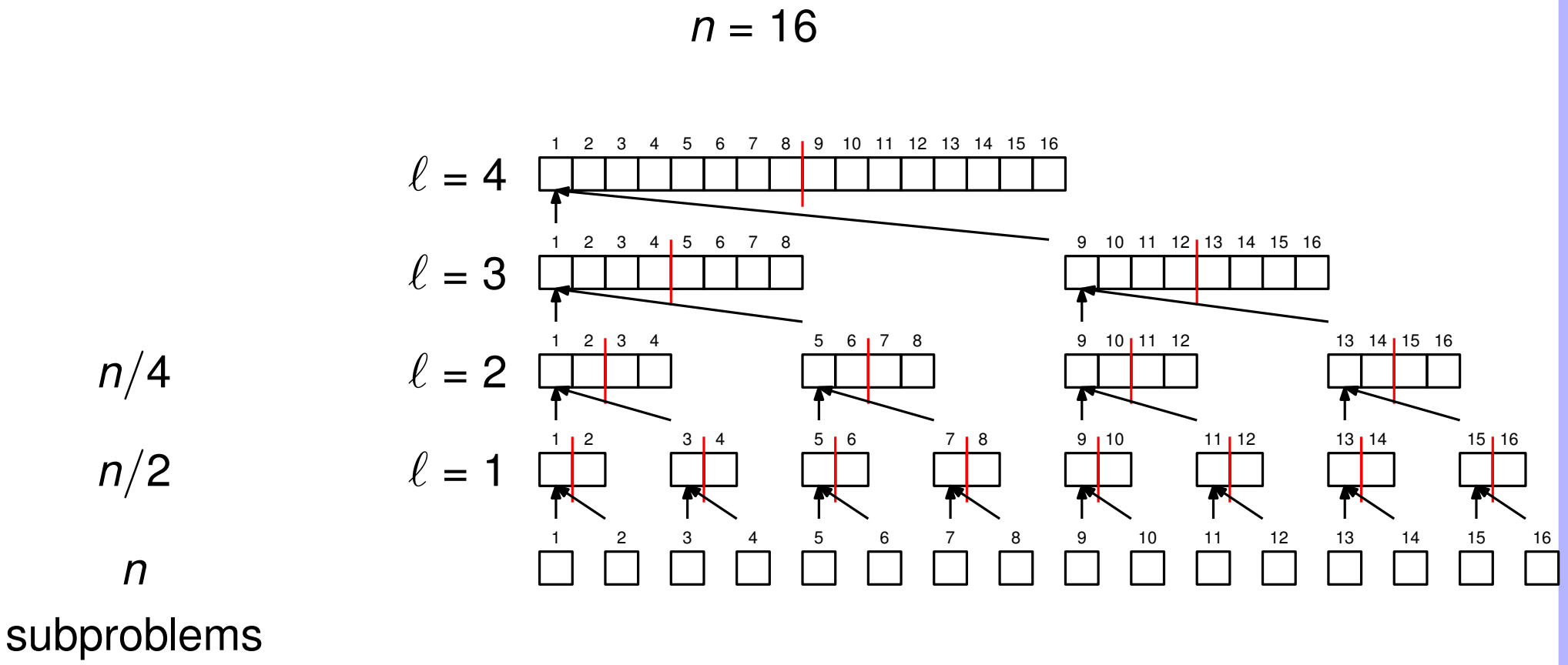


n subproblems

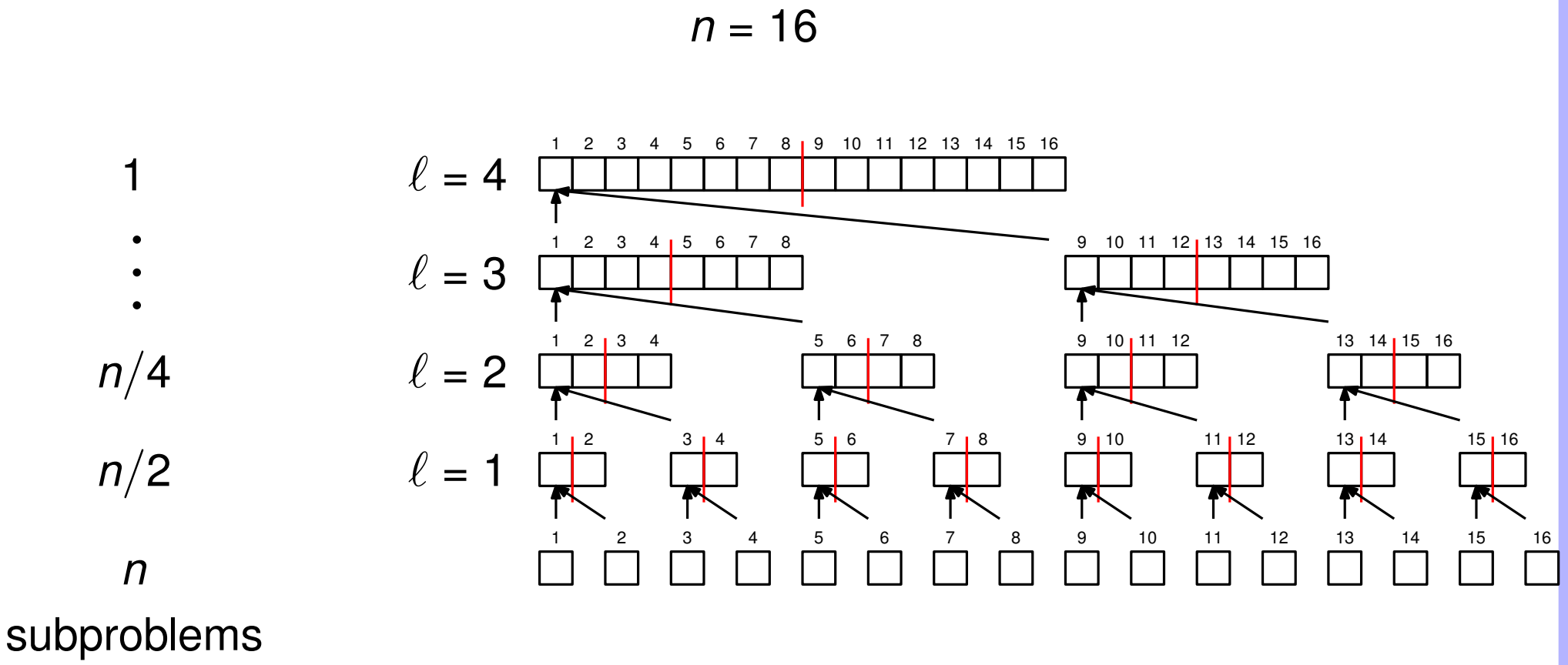
Fewer than n processors?



Fewer than n processors?

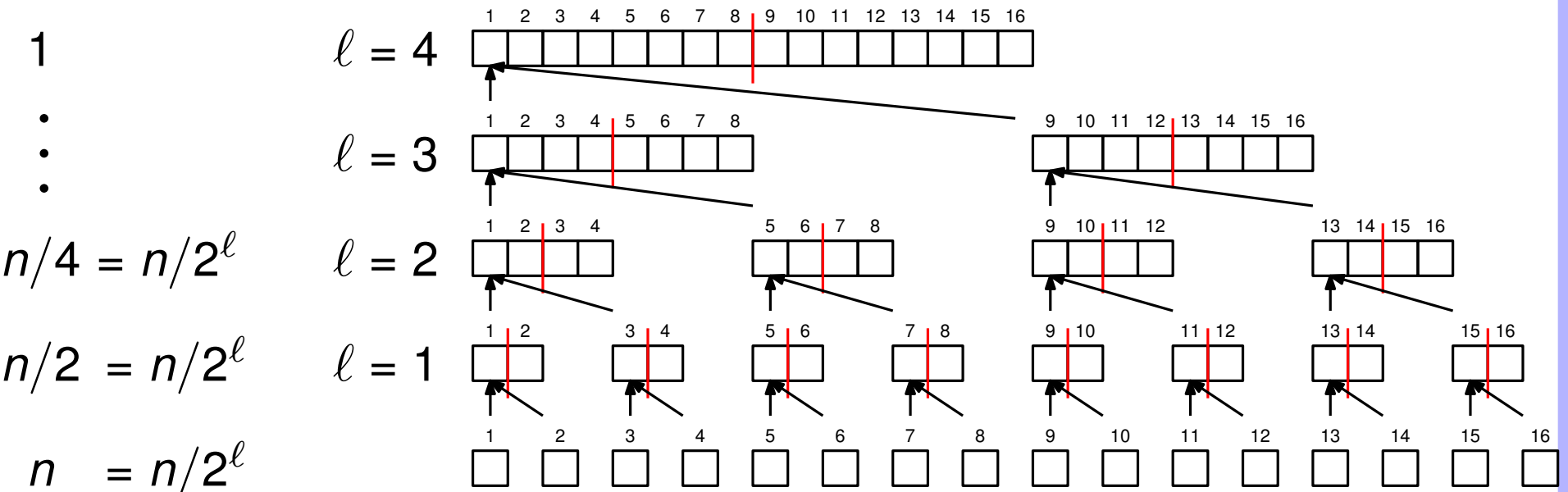


Fewer than n processors?



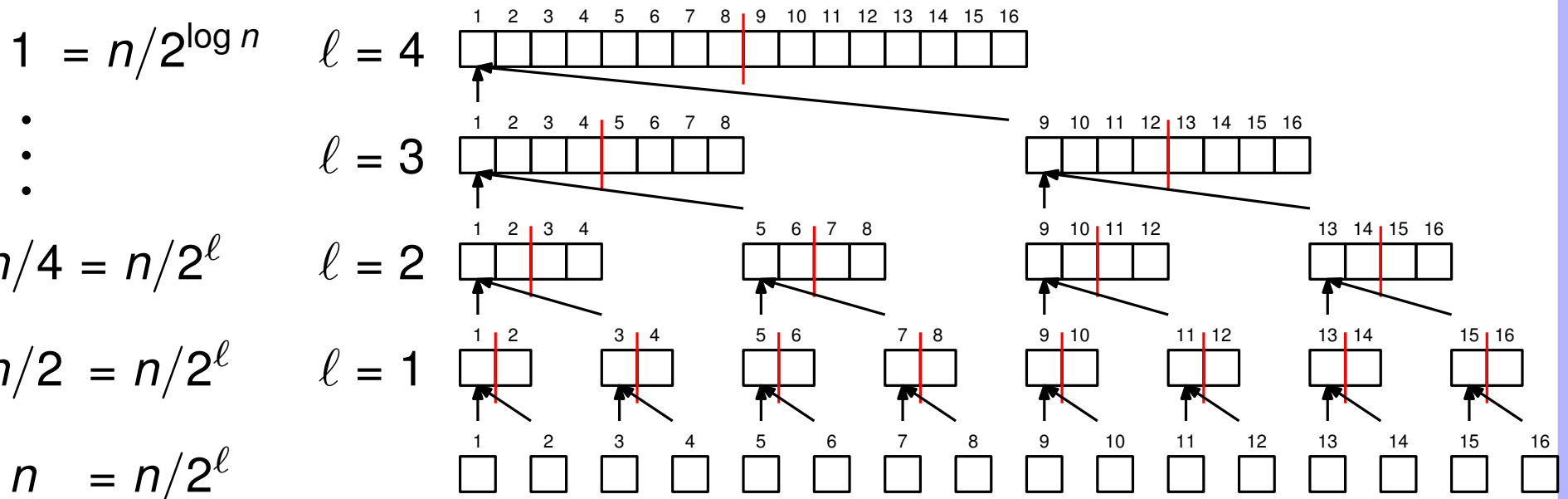
Fewer than n processors?

$n = 16$



Fewer than n processors?

$n = 16$



subproblems

Fewer than n processors?

$n = 16$

$P = 1 = n/2^{\log n}$ $\ell = 4$

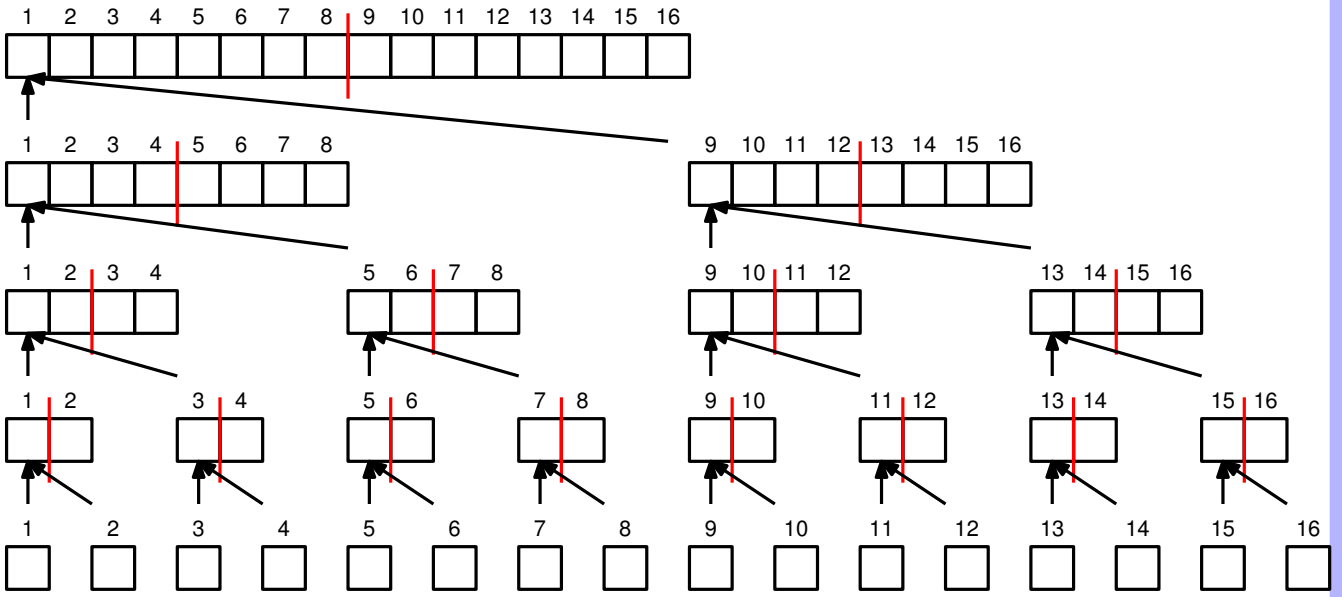
$\vdots$ $\ell = 3$

$P = n/4 = n/2^{\ell}$ $\ell = 2$

$P = n/2 = n/2^{\ell}$ $\ell = 1$

$P = n = n/2^{\ell}$

subproblems



Fewer than n processors?

$n = 16$

$P = 1 = n/2^{\log n}$ $\quad \ell = 4$

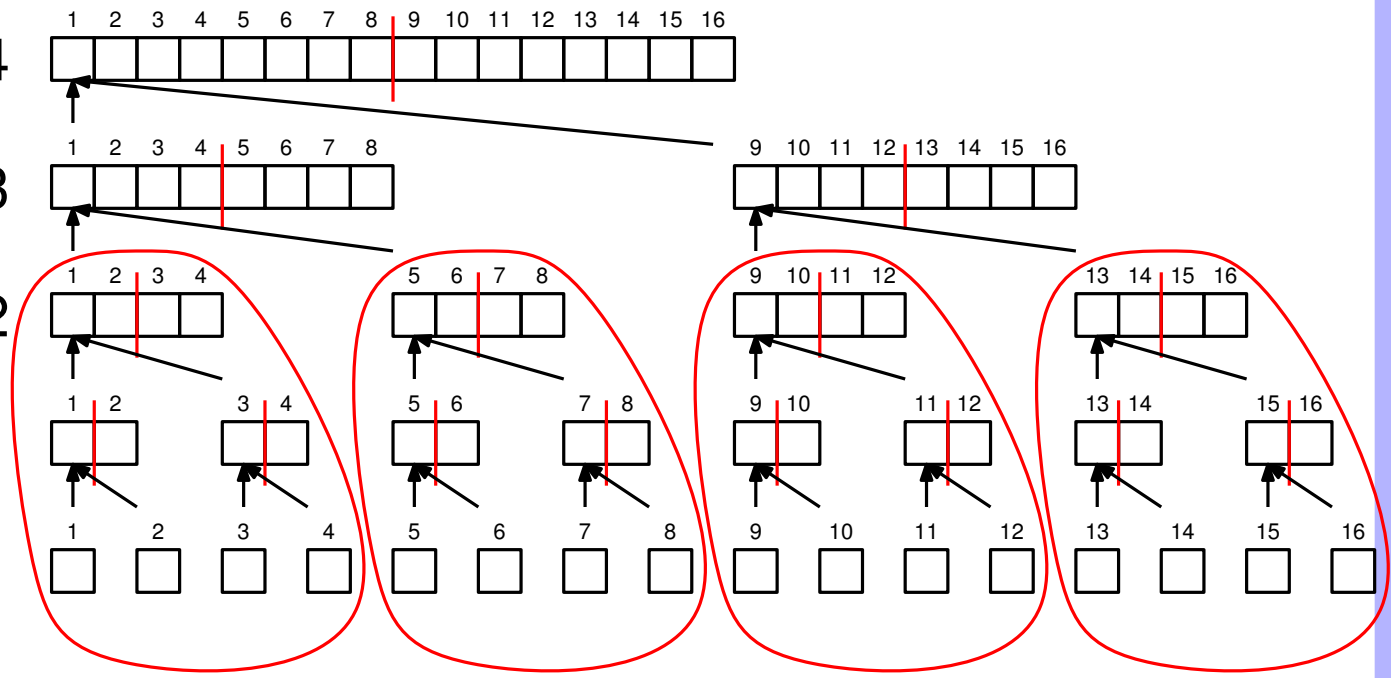
$\vdots$ $\quad \ell = 3$

$P = n/4 = n/2^{\ell}$ $\quad \ell = 2$

$P = n/2 = n/2^{\ell}$ $\quad \ell = 1$

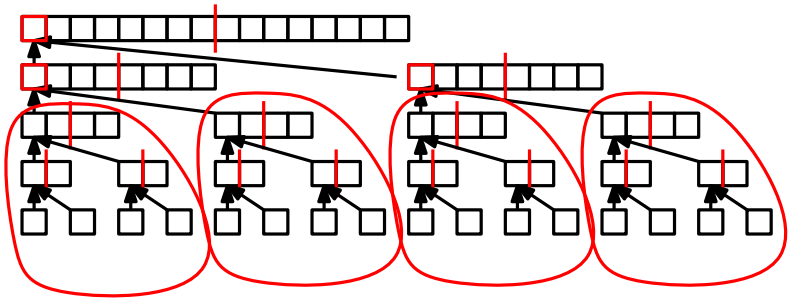
$P = n = n/2^{\ell}$

subproblems



Fewer than n processors?

$P = n/2^{\ell}$

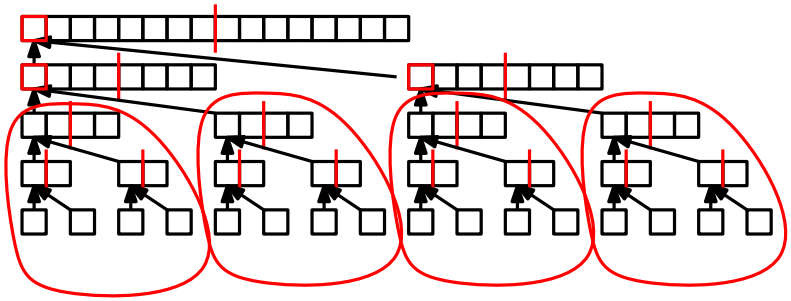


procedure MIN($a[1..n]$)

for $\ell = 1$ to $\log n$ **do**
  for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
    if $i + 2^{\ell-1} \leq n$ **then**
      $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
return $a[1]$

Fewer than n processors?

$$\ell = \log \frac{n}{P} \longleftarrow P = n/2^{\ell}$$

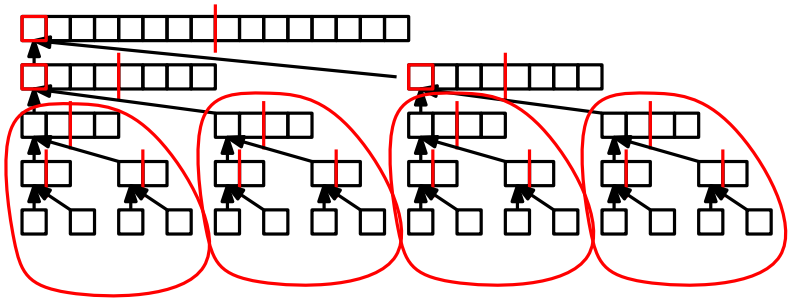


procedure MIN($a[1..n]$)

for $\ell = 1$ to $\log n$ **do**
  for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
    if $i + 2^{\ell-1} \leq n$ **then**
      $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
return $a[1]$

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



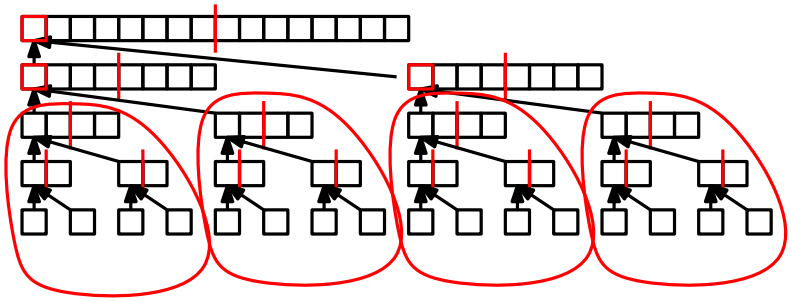
procedure MIN($a[1..n]$)

for $\ell = 1$ to $\log n$ **do**
  for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
    if $i + 2^{\ell-1} \leq n$ **then**
      $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
return $a[1]$

Upper part

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



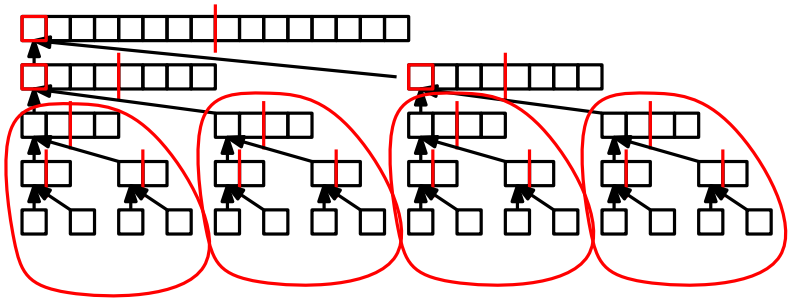
procedure MIN($a[1..n]$)

for $\ell = \log \left\lceil \frac{n}{P} \right\rceil + 1$ to $\log n$ **do**
  for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
    if $i + 2^{\ell-1} \leq n$ **then**
      $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
return $a[1]$

Upper part

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



procedure MIN($a[1..n]$)

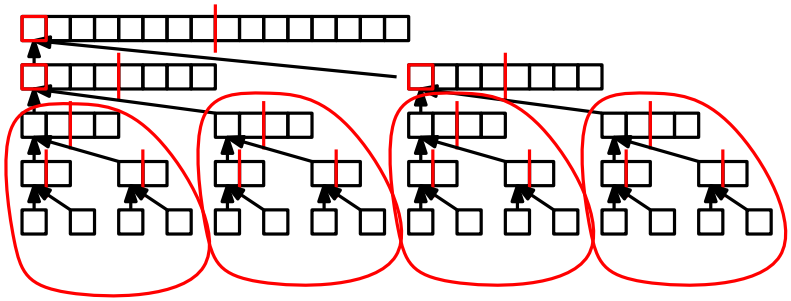
  for $i = 1$ to P **in parallel do**
    $start = \ldots$
    for $k = start + 1$ to $\ldots$ **do**
      if $a[k] < a[start]$ **then**
        $a[start] = a[k]$
  for $\ell = \log \left\lceil \frac{n}{P} \right\rceil + 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

Lower part

Upper part

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



procedure MIN($a[1..n]$)

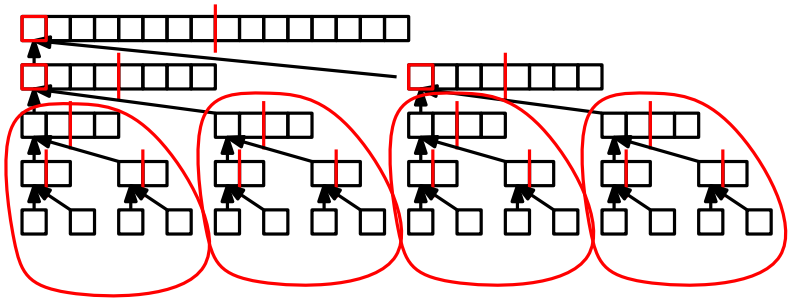
  for $i = 1$ to P **in parallel do**
    $start = \ldots$
    for $k = start + 1$ to $start + \lceil n/P \rceil - 1$ **do**
      if $a[k] < a[start]$ **then**
        $a[start] = a[k]$
  for $\ell = \log \left\lceil \frac{n}{P} \right\rceil + 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

Lower part

Upper part

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



procedure MIN($a[1..n]$)

  for $i = 1$ to P **in parallel do**
    $start = (i-1) \cdot \lceil n/P \rceil + 1$
    for $k = start + 1$ to $start + \lceil n/P \rceil - 1$ **do**
      if $a[k] < a[start]$ **then**
        $a[start] = a[k]$

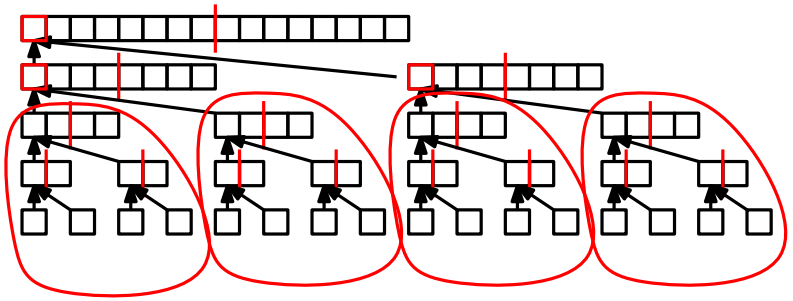
  for $\ell = \log \left\lceil \frac{n}{P} \right\rceil + 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

Lower part

Upper part

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



procedure MIN($a[1..n]$)

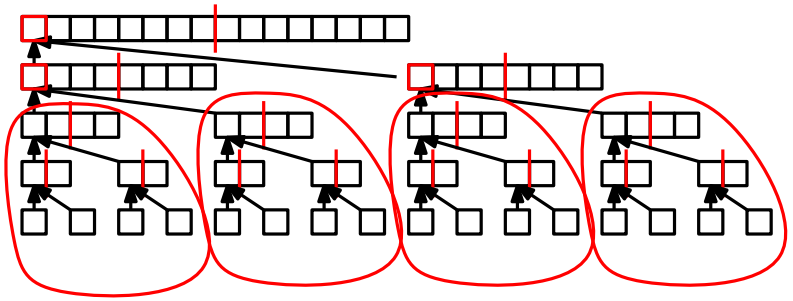
  for $i = 1$ to P **in parallel do**
    $start = (i-1) \cdot \lceil n/P \rceil + 1$
    for $k = start + 1$ to $start + \lceil n/P \rceil - 1$ **do**
      if $a[k] < a[start]$ **then**
        $a[start] = a[k]$

  for $\ell = \log \left\lceil \frac{n}{P} \right\rceil + 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

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

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



procedure MIN($a[1..n]$)

  for $i = 1$ to P **in parallel do**
    $start = (i-1) \cdot \lceil n/P \rceil + 1$
    for $k = start + 1$ to $start + \lceil n/P \rceil - 1$ **do**
      if $a[k] < a[start]$ **then**
        $a[start] = a[k]$

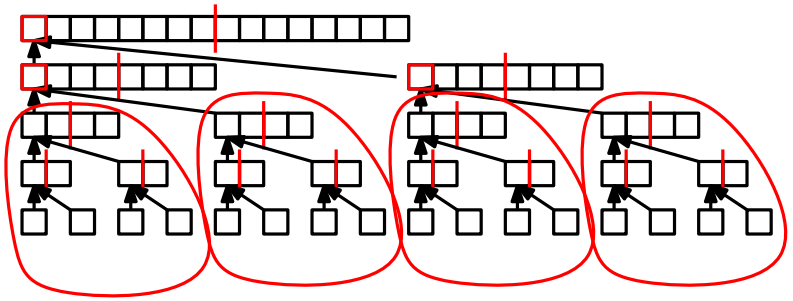
  for $\ell = \log \lceil \frac{n}{P} \rceil + 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

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

$O(n/P)$

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



procedure MIN($a[1..n]$)

  for $i = 1$ to P **in parallel do**
    $start = (i-1) \cdot \lceil n/P \rceil + 1$
    for $k = start + 1$ to $start + \lceil n/P \rceil - 1$ **do**
      if $a[k] < a[start]$ **then**
        $a[start] = a[k]$

  for $\ell = \log \left\lceil \frac{n}{P} \right\rceil + 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

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

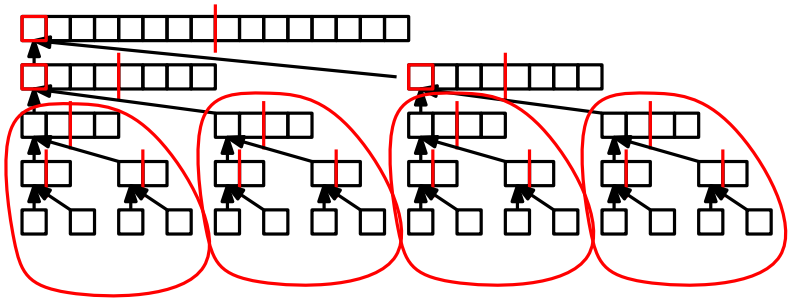
$O(n/P)$

$O\left(\log n - \left(\log \frac{n}{p} + 1\right)\right)$

$= O(\log P)$

Fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



procedure MIN($a[1..n]$)

  for $i = 1$ to P **in parallel do**
    $start = (i-1) \cdot \lceil n/P \rceil + 1$
    for $k = start + 1$ to $start + \lceil n/P \rceil - 1$ **do**
      if $a[k] < a[start]$ **then**
        $a[start] = a[k]$

  for $\ell = \log \left\lceil \frac{n}{P} \right\rceil + 1$ to $\log n$ **do**
    for $i = 1; i \leq n; i = i + 2^{\ell}$ **in parallel do**
      if $i + 2^{\ell-1} \leq n$ **then**
        $a[i] = \min(a[i], a[i + 2^{\ell-1}])$
  return $a[1]$

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

$O(n/P)$

$O\left(\log n - \left(\log \frac{n}{p} + 1\right)\right)$

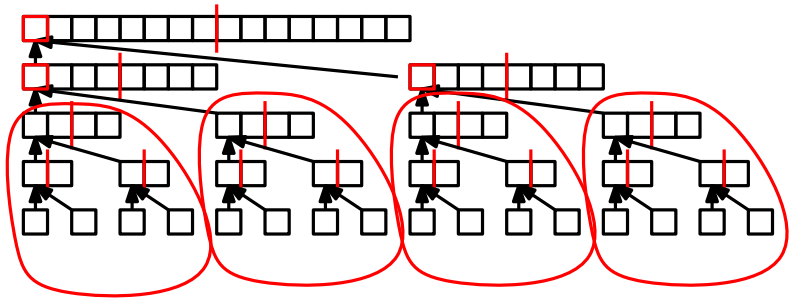
$= O(\log P)$

Total:

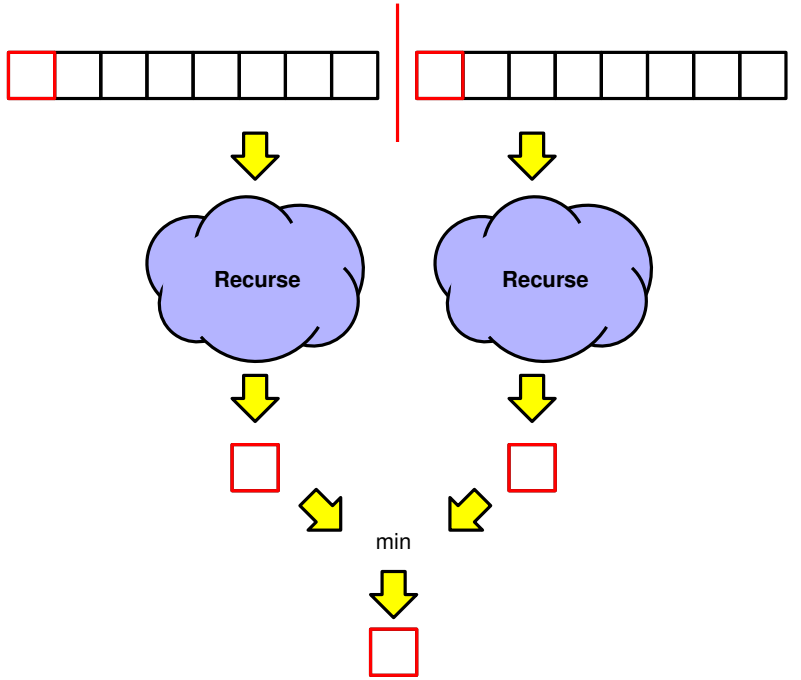
$T_P(n) = O(\frac{n}{P} + \log P)$

Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ← $P = n/2^{\ell}$

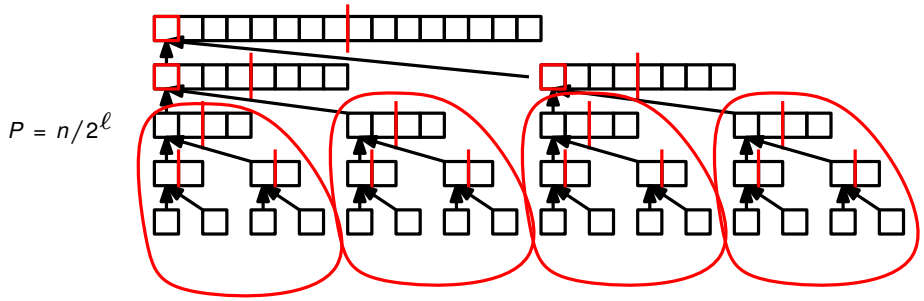


```
procedure MIN(a[i..j])
  if i = j then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid])
          a[rmid] = MIN(a[rmid..j])
      return min(a[i], a[rmid])
```

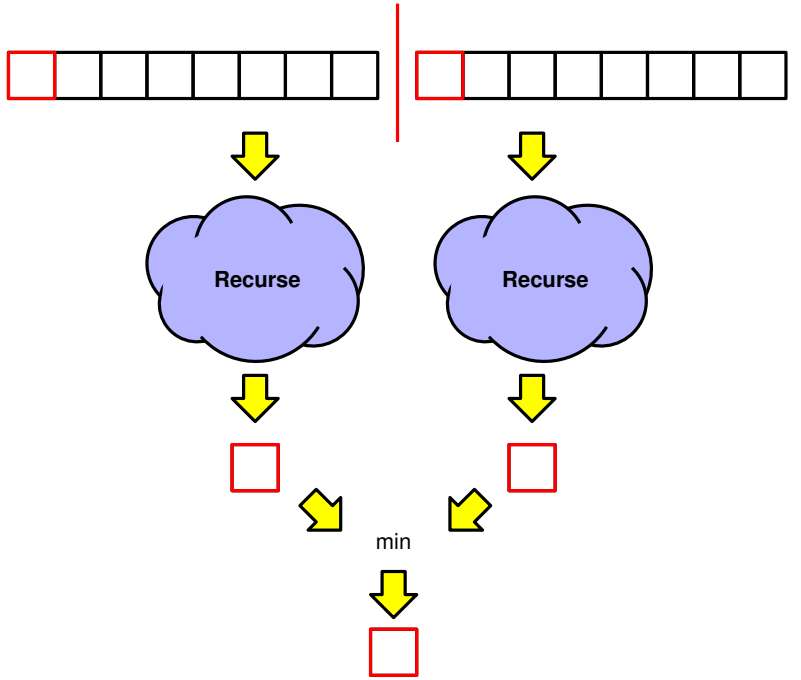


Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$

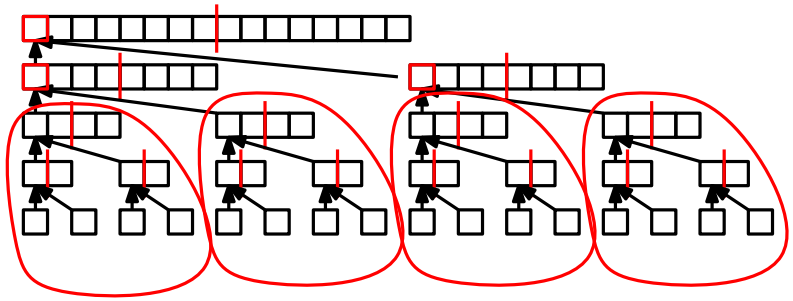



```
procedure MIN(a[i..j])
  if j − i + 1 = 1 then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid])
          a[rmid] = MIN(a[rmid..j])
      return min(a[i], a[rmid])
```

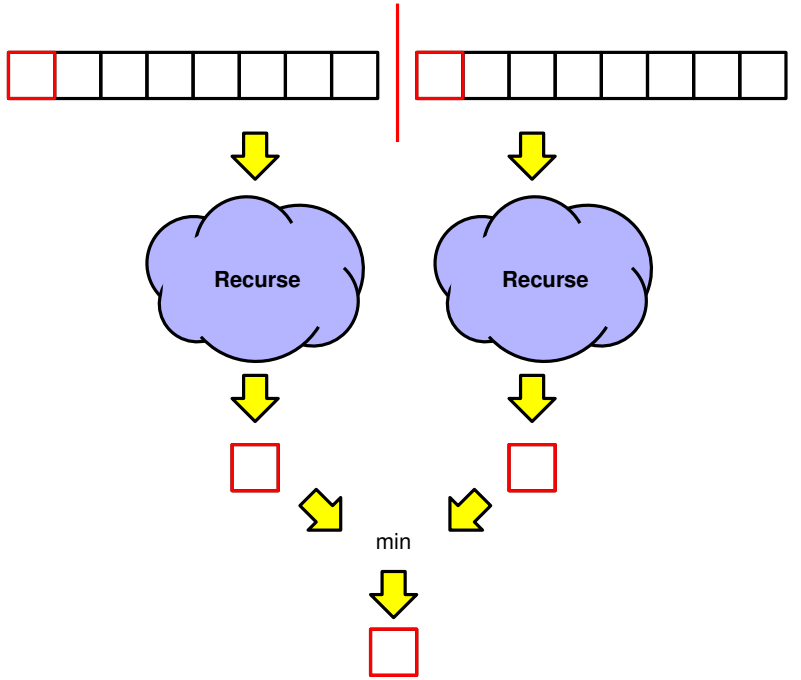


Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$

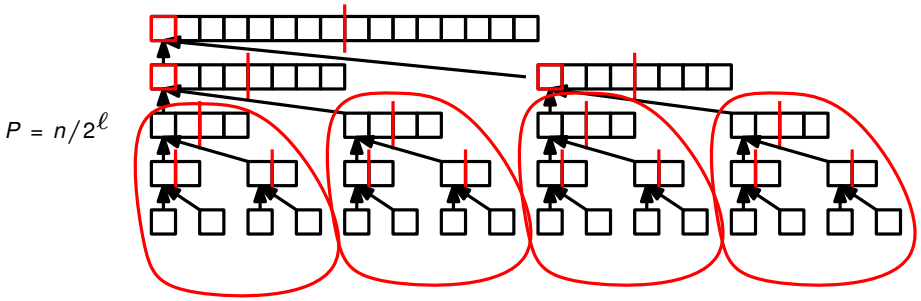


procedure MIN($a[i..j], n, P$)
  if $j - i + 1 \leq n/P$ **then**
    return SEQMIN($a[i..j]$)
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[i]$ = MIN($a[i..mid], n, P$)
      $a[rmid]$ = MIN($a[rmid..j], n, P$)
    return min($a[i], a[rmid]$)



Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ←

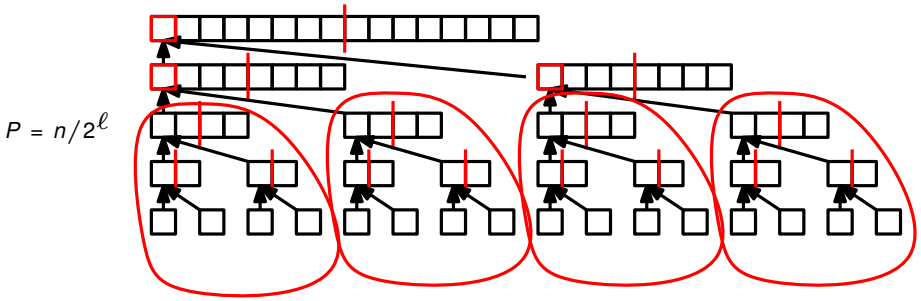



```
procedure MIN(a[i..j], n, P)
  if j − i + 1 ≤ n/P then
      return SEQMIN(a[i..j])
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid], n, P)
          a[rmid] = MIN(a[rmid..j], n, P)
      return min(a[i], a[rmid])
```

```
procedure SEQMIN(a[i..j])
    for k = i + 1 to j do
        if a[k] < a[i] then
            a[i] = a[k]
    return a[i]
```

Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ←

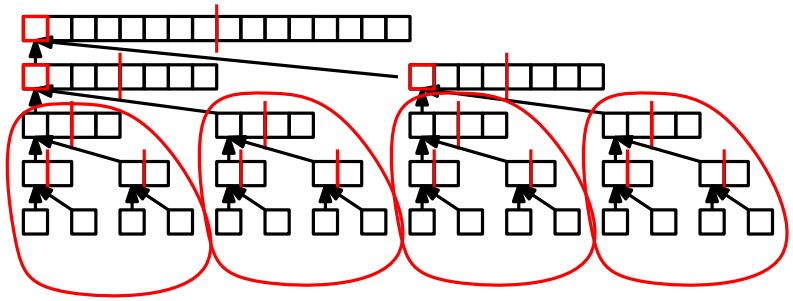



```
procedure MIN(a[i..j], n, P)
  if j − i + 1 < 2n/P then
      return SEQMIN(a[i..j])
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid], n, P)
          a[rmid] = MIN(a[rmid..j], n, P)
      return min(a[i], a[rmid])
```

```
procedure SEQMIN(a[i..j])
    for k = i + 1 to j do
        if a[k] < a[i] then
            a[i] = a[k]
    return a[i]
```

Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



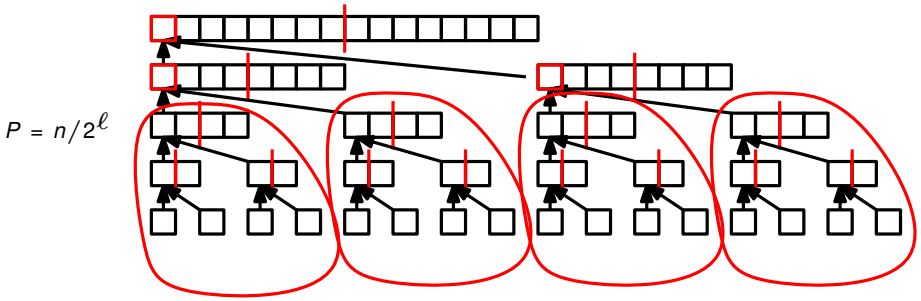
procedure MIN($a[i..j]$, n, P)
  if $j - i + 1 < 2n/P$ **then**
    return SEQMIN($a[i..j]$)
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[i]$ = MIN($a[i..mid]$, n, P)
      $a[rmid]$ = MIN($a[rmid..j]$, n, P)
    return min($a[i]$, $a[rmid]$)

procedure SEQMIN(a[i..j])
  for $k = i + 1$ to j **do**
    if $a[k] < a[i]$ **then**
      $a[i] = a[k]$
  return $a[i]$

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

Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵




```
procedure MIN(a[i..j], n, P)
  if j − i + 1 < 2n/P then
      return SEQMIN(a[i..j])
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid], n, P)
          a[rmid] = MIN(a[rmid..j], n, P)
      return min(a[i], a[rmid])
```

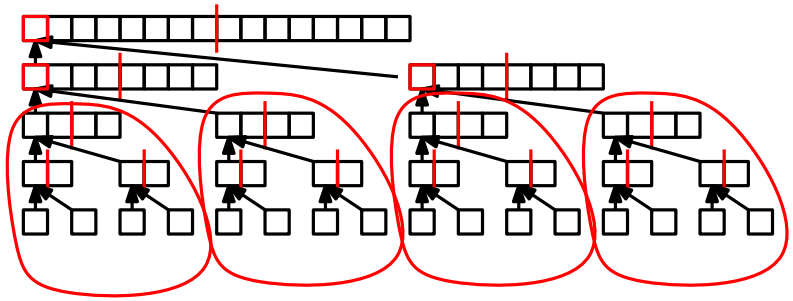
```
procedure SEQMIN(a[i..j])
    for k = i + 1 to j do
        if a[k] < a[i] then
            a[i] = a[k]
    return a[i]
```

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

$$T_P(n) = \begin{cases} T_P(n/2) + O(1) & \text{if } n \geq \frac{2n}{P} \\ O(n) & \text{if } n < \frac{2n}{P} \end{cases}$$

Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



```
procedure MIN(a[i..j], n, P)
  if j − i + 1 < 2n/P then
      return SEQMIN(a[i..j])
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid], n, P)
          a[rmid] = MIN(a[rmid..j], n, P)
      return min(a[i], a[rmid])
```

```
procedure SEQMIN(a[i..j])
    for k = i + 1 to j do
        if a[k] < a[i] then
            a[i] = a[k]
    return a[i]
```

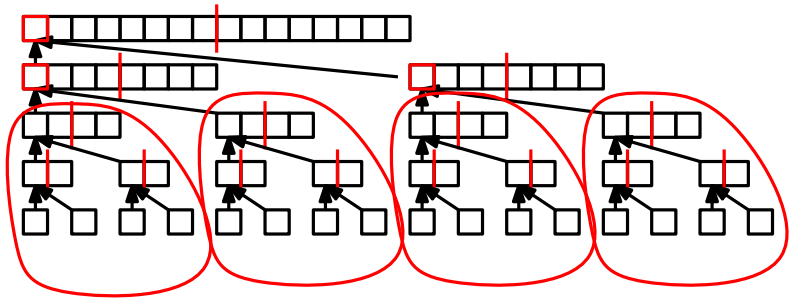
Runtime of MIN($a[1..n]$)?

same n?!

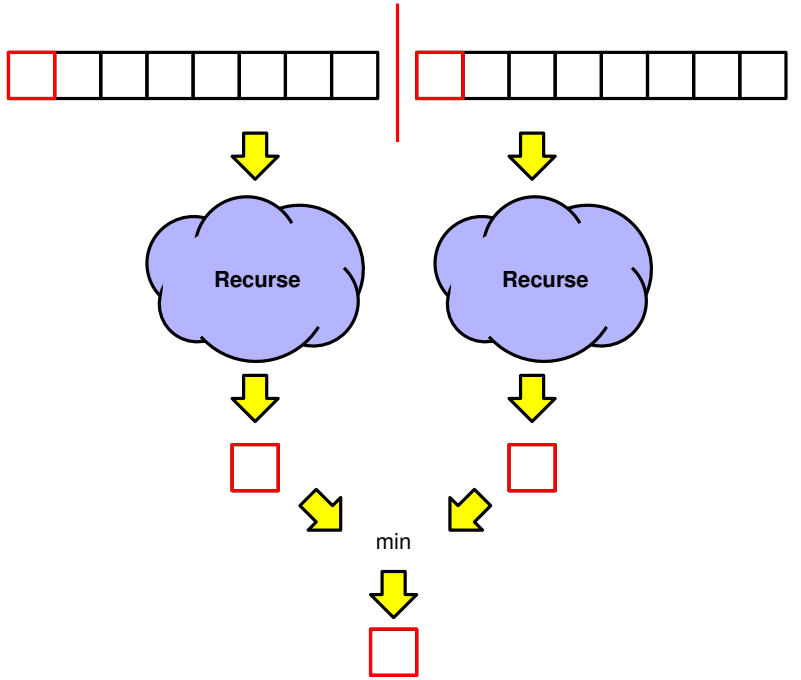
$$T_P(n) = \begin{cases} T_P(n/2) + O(1) & \text{if } n \geq \frac{2n}{P} \\ O(n) & \text{if } n < \frac{2n}{P} \end{cases}$$

Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$

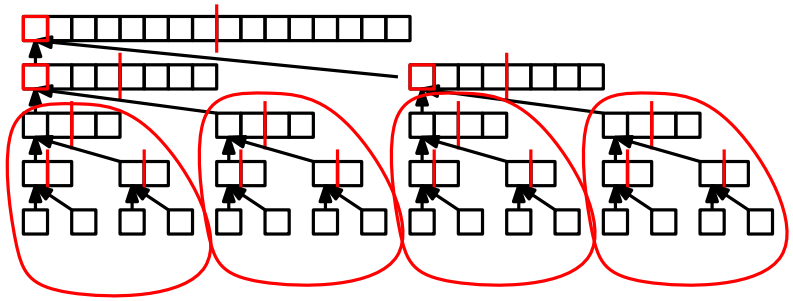


procedure MIN($a[i..j], n, P$)
  if $j - i + 1 < 2n/P$ **then**
    return SEQMIN($a[i..j]$)
  else
    $mid = \left\lfloor \frac{i+j}{2} \right\rfloor$, $rmid = mid + 1$
    in parallel do
      $a[i]$ = MIN($a[i..mid], n, P$)
      $a[rmid]$ = MIN($a[rmid..j], n, P$)
    return min($a[i], a[rmid]$)

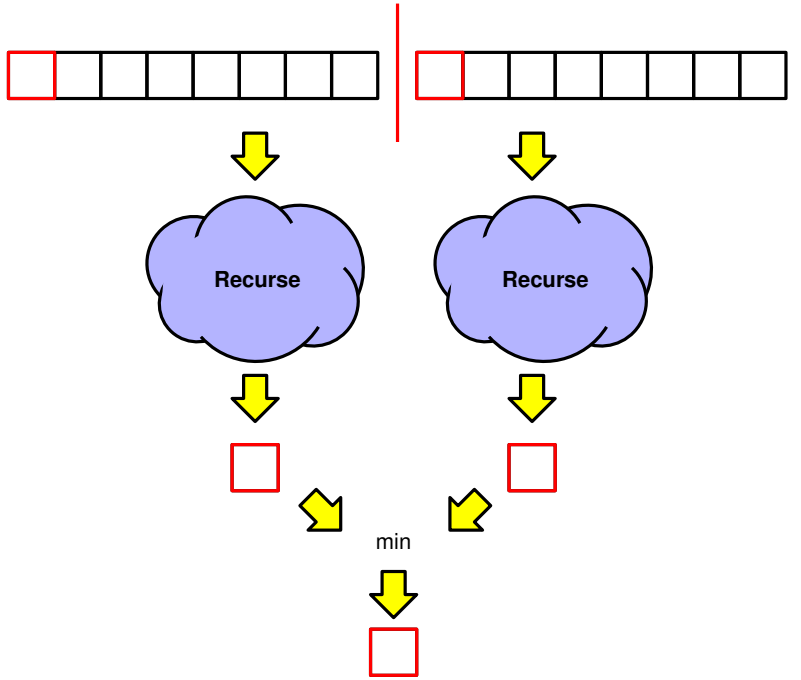


Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ← $P = n/2^{\ell}$

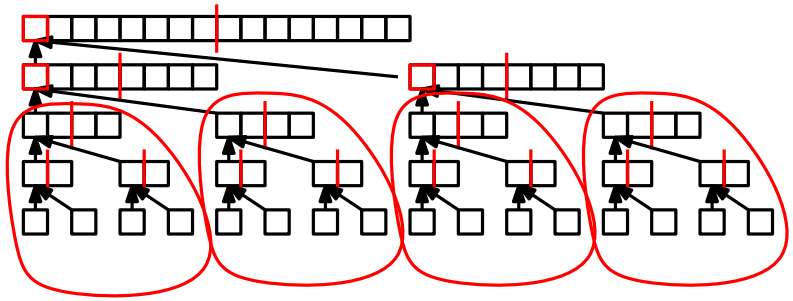


```
procedure MIN(a[i..j], P)
  if P = 1 then
      return SEQMIN(a[i..j])
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid], ⌈P/2⌉)
          a[rmid] = MIN(a[rmid..j], ⌊P/2⌋)
      return min(a[i], a[rmid])
```



Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$

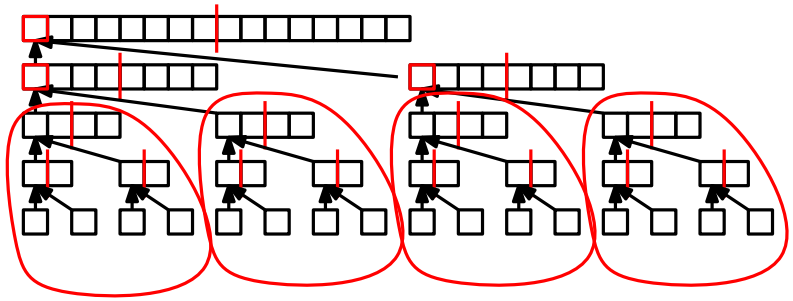


```
procedure MIN(a[i..j], P)
  if P = 1 then
      return SEQMIN(a[i..j])
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid], ⌈P/2⌉)
          a[rmid] = MIN(a[rmid..j], ⌊P/2⌋)
      return min(a[i], a[rmid])
```

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

Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



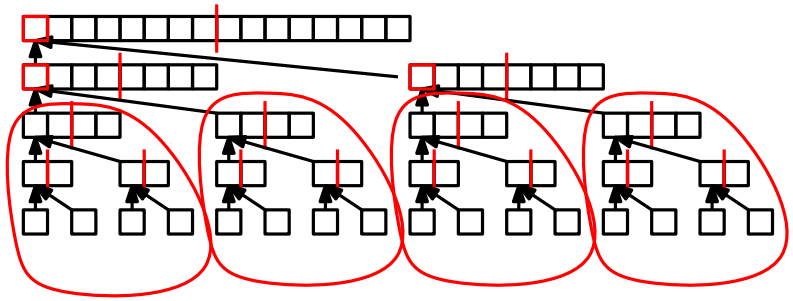
```
procedure MIN(a[i..j], P)
  if P = 1 then
      return SEQMIN(a[i..j])
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid], ⌈P/2⌉)
          a[rmid] = MIN(a[rmid..j], ⌊P/2⌋)
      return min(a[i], a[rmid])
```

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

$$T_P(n) = \begin{cases} T_{P/2}(n/2) + O(1) & \text{if } P > 1 \\ O(n) & \text{if } P = 1 \end{cases}$$

Recursion with fewer than n processors?

$\ell = \log \frac{n}{P}$ ⟵ $P = n/2^{\ell}$



```
procedure MIN(a[i..j], P)
  if P = 1 then
      return SEQMIN(a[i..j])
  else
      mid = ⌊(i+j)/2⌋, rmid = mid + 1
      in parallel do
          a[i] = MIN(a[i..mid], ⌈P/2⌉)
          a[rmid] = MIN(a[rmid..j], ⌊P/2⌋)
      return min(a[i], a[rmid])
```

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

$$T_P(n) = \begin{cases} T_{P/2}(n/2) + O(1) & \text{if } P > 1 \\ O(n) & \text{if } P = 1 \end{cases}$$
$$= O\left(\frac{n}{P} + \log P\right)$$

Runtime of MIN on P processors

$$T_P(n) = O\left(\frac{n}{P} + \log P\right)$$

Runtime of MIN on P processors

$$T_P(n) = O\left(\frac{n}{P} + \log P\right)$$

If $P = 1$: $T_1(n) = O(n)$

If $P = n$: $T_n(n) = O(\log n)$

Runtime of MIN on P processors

$$T_P(n) = O\left(\frac{n}{P} + \log P\right)$$

If $P = 1$: $T_1(n) = O(n)$ ← Sequential runtime for finding minimum

If $P = n$: $T_n(n) = O(\log n)$

Runtime of MIN on P processors

$$T_P(n) = O\left(\frac{n}{P} + \log P\right)$$

If $P = 1$: $T_1(n) = O(n)$ ← Sequential runtime for finding minimum

If $P = n$: $T_n(n) = O(\log n)$ ← Parallel runtime with n processors

Work-Depth Analysis

We use two metrics to analyze parallel algorithms

Work-Depth Analysis

We use two metrics to analyze parallel algorithms

Work: $W(n)$

(Parallel) Time: $T(n)$

Work-Depth Analysis

We use two metrics to analyze parallel algorithms

Work: $W(n)$

Also denoted as:

- $T_1(n)$: sequential time

(Parallel) Time: $T(n)$

Also denoted as:

- $D(n)$: depth
- $S(n)$: span
- $T_\infty(n)$ $(P = \infty)$

Work-Depth Analysis

We use two metrics to analyze parallel algorithms

Work: $W(n)$

Also denoted as:

- $T_1(n)$: sequential time

(Parallel) Time: $T(n)$

Also denoted as:

- $D(n)$: depth
- $S(n)$: span
- $T_\infty(n)$ $(P = \infty)$

Work-efficient parallel algorithm:

$W(n) = O(\text{time of the fastest sequential algorithm})$

Analysis Example

```
procedure MIN(a[i..j])
  if i = j then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋; rmid = mid + 1
      in parallel do
          left = MIN(a[i..mid])
          right = MIN(a[rmid..j])
      return min(left, right)
```

Analysis Example

```
procedure MIN(a[i..j])
  if i = j then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋; rmid = mid + 1
      in parallel do
          left = MIN(a[i..mid])
          right = MIN(a[rmid..j])
      return min(left, right)
```

Span

Work

Analysis Example

```
procedure MIN(a[i..j])
  if i = j then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋; rmid = mid + 1
      in parallel do
          left = MIN(a[i..mid])
          right = MIN(a[rmid..j])
      return min(left, right)
```

Span

Work

$$T(n) = T(n/2) + \Theta(1)$$
$$= \Theta(\log n)$$

Analysis Example

```
procedure MIN(a[i..j])
  if i = j then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋; rmid = mid + 1
      in parallel do
          left = MIN(a[i..mid])
          right = MIN(a[rmid..j])
      return min(left, right)
```

Span

$$T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$$

Work

$$W(n) = 2W(n/2) + \Theta(1)$$

Analysis Example

```
procedure MIN(a[i..j])
  if i = j then
      return a[i]
  else
      mid = ⌊(i+j)/2⌋; rmid = mid + 1
      in parallel do
          left = MIN(a[i..mid])
          right = MIN(a[rmid..j])
      return min(left, right)
```

Span

$$T(n) = T(n/2) + \Theta(1) = \Theta(\log n)$$

Work

$$W(n) = 2W(n/2) + \Theta(1) = \Theta(n)$$

Analysis Example 2

```
procedure MIN(a[i..j])
    for ℓ = 1 to log n do
        for i = 1; i ≤ n; i = i + 2^ℓ in parallel do
            if i + 2^(ℓ−1) ≤ n then
                a[i] = min(a[i], a[i + 2^(ℓ−1)])
    return a[1]
```

Span

Work

Analysis Example 2

```
procedure MIN(a[i..j])
    for ℓ = 1 to log n do
        for i = 1; i ≤ n; i = i + 2^ℓ in parallel do
            if i + 2^(ℓ−1) ≤ n then
                a[i] = min(a[i], a[i + 2^(ℓ−1)])
    return a[1]
```

Span

$$T(n) = \sum_{\ell=1}^{\log n} \Theta(1)$$
$$= \Theta(\log n)$$

Work

Analysis Example 2

```
procedure MIN(a[i..j])
    for ℓ = 1 to log n do
        for i = 1; i ≤ n; i = i + 2^ℓ in parallel do
            if i + 2^(ℓ−1) ≤ n then
                a[i] = min(a[i], a[i + 2^(ℓ−1)])
    return a[1]
```

Span

$$T(n) = \sum_{\ell=1}^{\log n} \Theta(1)$$
$$= \Theta(\log n)$$

Work

$$T(n) = \sum_{\ell=1}^{\log n}$$

Analysis Example 2

```
procedure MIN(a[i..j])
    for ℓ = 1 to log n do
        for i = 1; i ≤ n; i = i + 2^ℓ in parallel do
            if i + 2^(ℓ−1) ≤ n then
                a[i] = min(a[i], a[i + 2^(ℓ−1)])
    return a[1]
```

Span

$$T(n) = \sum_{\ell=1}^{\log n} \Theta(1)$$
$$= \Theta(\log n)$$

Work

$$T(n) = \sum_{\ell=1}^{\log n} \frac{n}{2^\ell}$$

Analysis Example 2

```
procedure MIN(a[i..j])
    for ℓ = 1 to log n do
        for i = 1; i ≤ n; i = i + 2^ℓ in parallel do
            if i + 2^(ℓ−1) ≤ n then
                a[i] = min(a[i], a[i + 2^(ℓ−1)])
    return a[1]
```

Span

$$T(n) = \sum_{\ell=1}^{\log n} \Theta(1) = \Theta(\log n)$$

Work

$$T(n) = \sum_{\ell=1}^{\log n} \frac{n}{2^\ell} = \Theta(n)$$

Brent's Scheduling Principle

Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time*

$$T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$$

Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

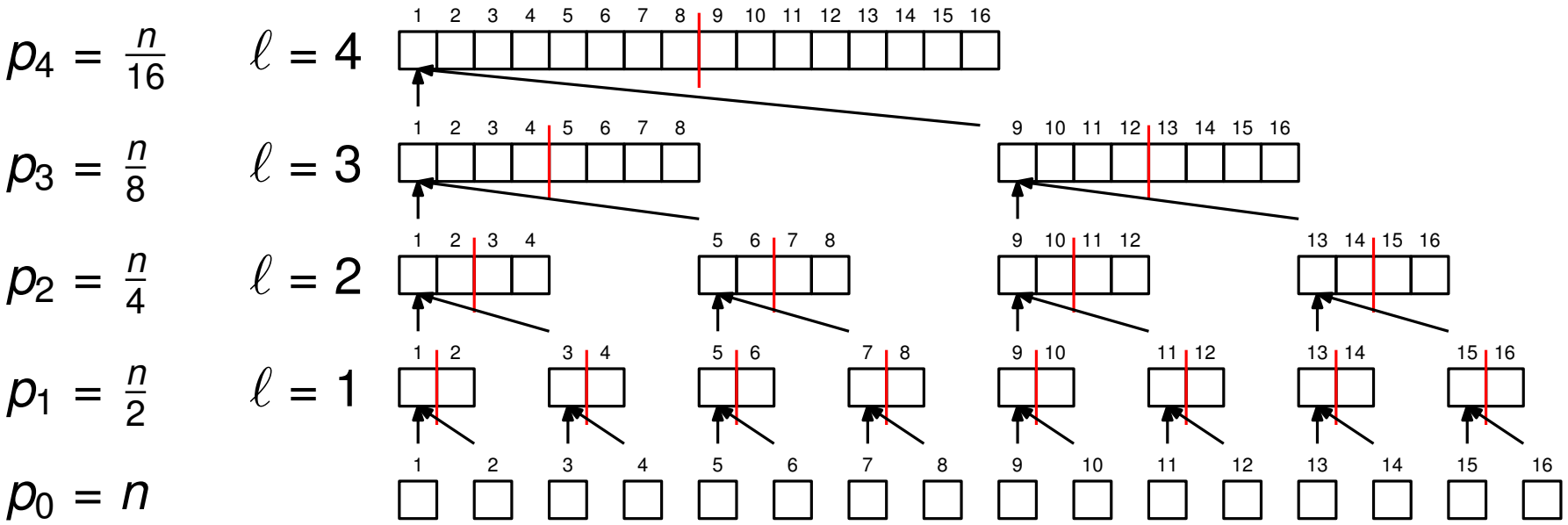
Proof. Let A be the original PRAM algorithm with work $W(n)$ and span $T(n)$. Let p_i be the number of "virtual" processors used by A in step i.

Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

Proof. Let A be the original PRAM algorithm with work $W(n)$ and span $T(n)$. Let p_i be the number of "virtual" processors used by A in step i.

Example

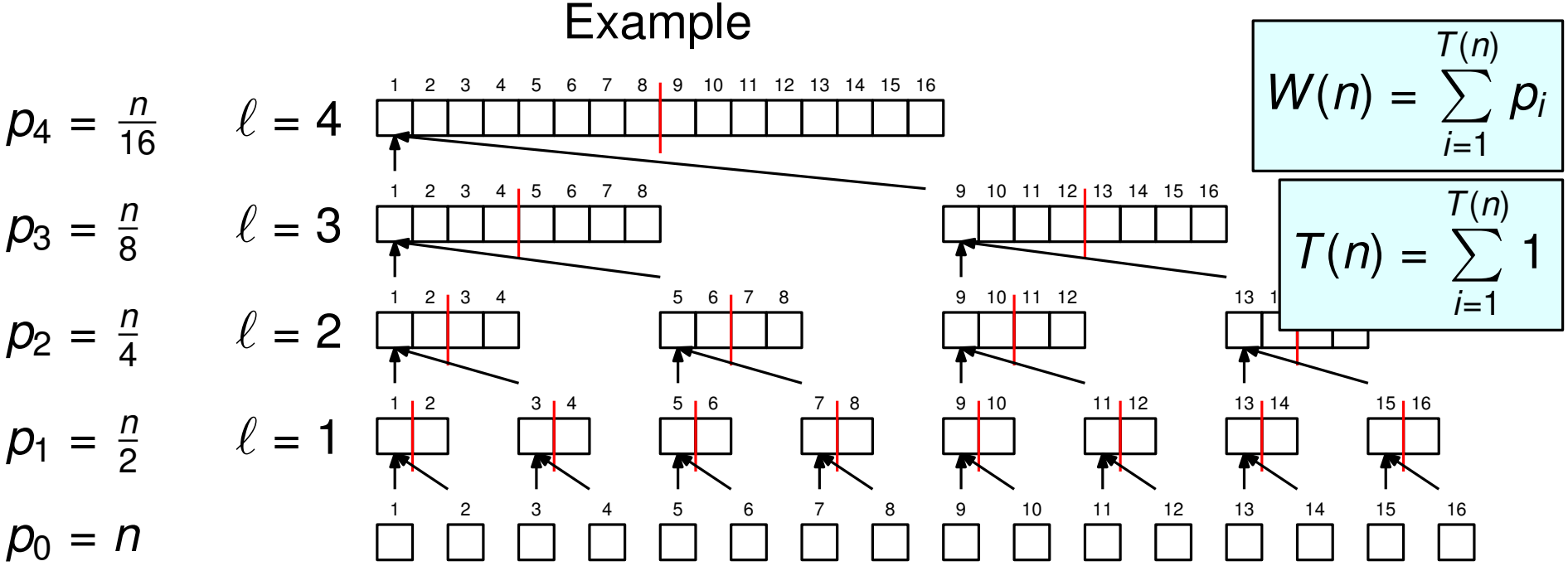


Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

Proof. Let A be the original PRAM algorithm with work $W(n)$ and span $T(n)$. Let p_i be the number of "virtual" processors used by A in step i.

Example



$$W(n) = \sum_{i=1}^{T(n)} p_i$$

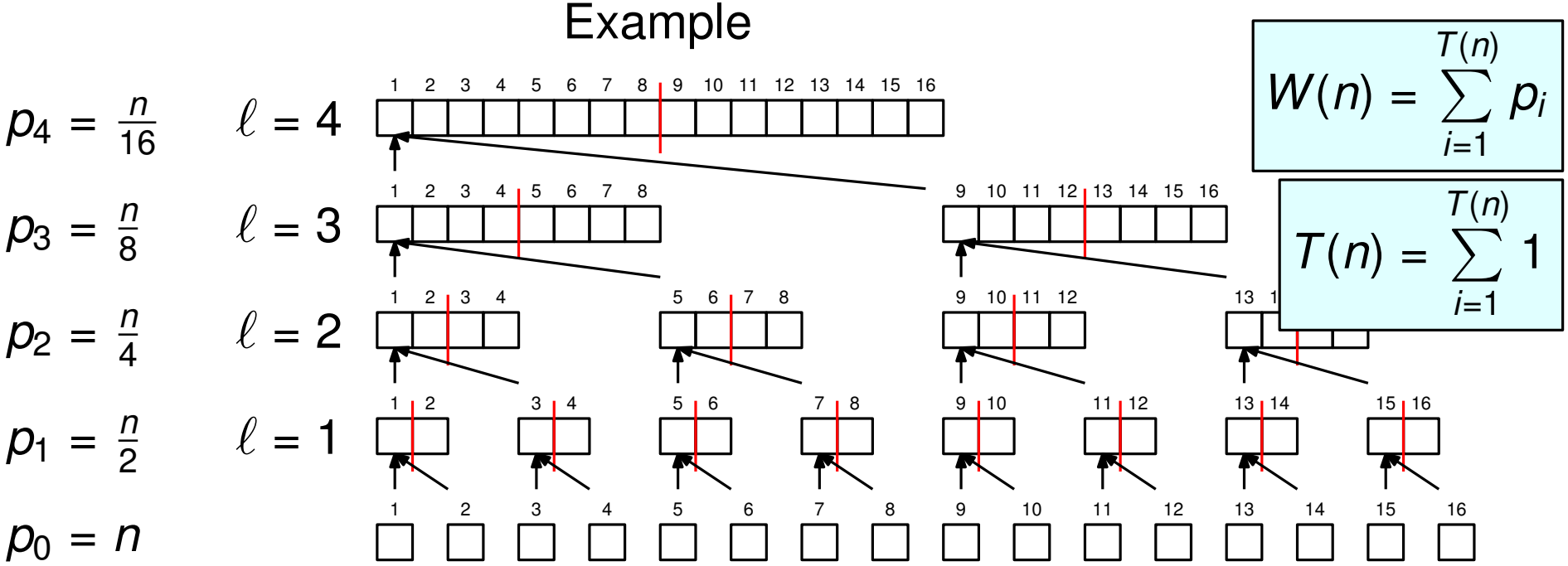
$$T(n) = \sum_{i=1}^{T(n)} 1$$

Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

Proof. Let A be the original PRAM algorithm with work $W(n)$ and span $T(n)$. Let p_i be the number of "virtual" processors used by A in step i.

Example



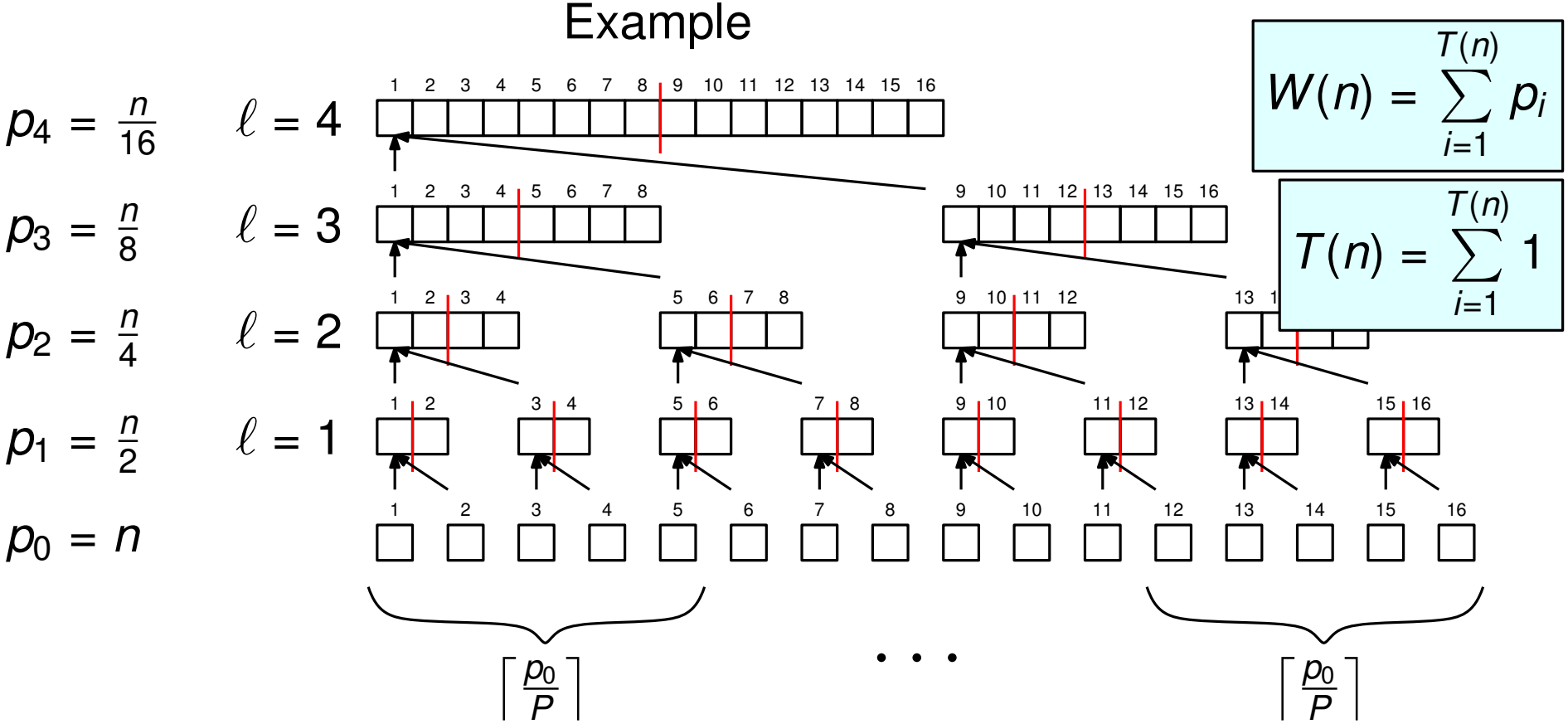
We design an algorithm A' that will run on P (physical) processors, executing the operations of $\left\lceil \frac{p_i}{P} \right\rceil$ "virtual" processors of A of step i.

Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

Proof. Let A be the original PRAM algorithm with work $W(n)$ and span $T(n)$. Let p_i be the number of "virtual" processors used by A in step i.

Example



Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

Thus, each step of A takes $\left\lceil \frac{p_i}{P} \right\rceil$ time on P (physical) processors, for a total of

$$T_P(n) = \sum_{i=1}^{T(n)} \left\lceil \frac{p_i}{P} \right\rceil \leq \sum_{i=1}^{T(n)} \left(\frac{p_i}{P} + 1 \right) = \frac{1}{P} \cdot \sum_{i=1}^{T(n)} p_i + \sum_{i=1}^{T(n)} 1$$

Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

Thus, each step of A takes $\left\lceil \frac{p_i}{P} \right\rceil$ time on P (physical) processors, for a total of

$$T_P(n) = \sum_{i=1}^{T(n)} \left\lceil \frac{p_i}{P} \right\rceil \leq \sum_{i=1}^{T(n)} \left(\frac{p_i}{P} + 1 \right) = \frac{1}{P} \cdot \sum_{i=1}^{T(n)} p_i + \sum_{i=1}^{T(n)} 1$$

$$W(n) = \sum_{i=1}^{T(n)} p_i$$

$$T(n) = \sum_{i=1}^{T(n)} 1$$

Brent's Scheduling Principle

Theorem. *Any PRAM algorithm with work $W(n)$ and span $T(n)$, can be implemented on a P-processor PRAM to run in time $T_P(n) = O\left(\frac{W(n)}{P} + T(n)\right)$*

Thus, each step of A takes $\left\lceil \frac{p_i}{P} \right\rceil$ time on P (physical) processors, for a total of

$$T_P(n) = \sum_{i=1}^{T(n)} \left\lceil \frac{p_i}{P} \right\rceil \leq \sum_{i=1}^{T(n)} \left(\frac{p_i}{P} + 1 \right) = \frac{1}{P} \cdot \sum_{i=1}^{T(n)} p_i + \sum_{i=1}^{T(n)} 1$$

$$= O\left(\frac{W(n)}{P} + T(n) \right)$$

Q.E.D.

$$W(n) = \sum_{i=1}^{T(n)} p_i$$

$$T(n) = \sum_{i=1}^{T(n)} 1$$