6.172 Performance Engineering of Software Systems

Lecture 9

Cache-Efficient Algorithms II

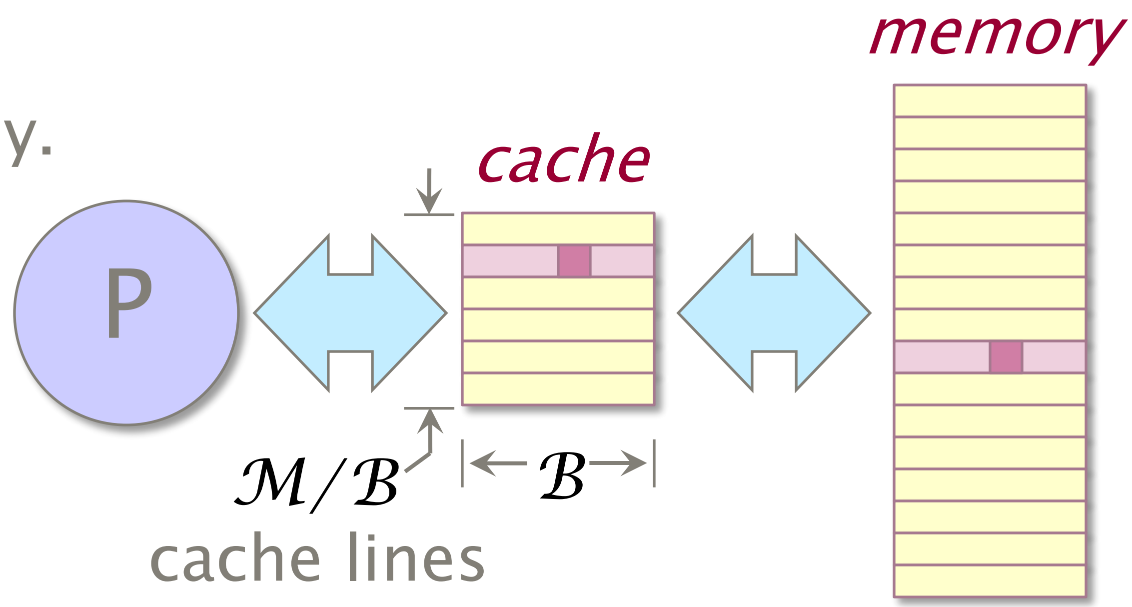
Charles E. Leiserson

October 7, 2010

Ideal-Cache Model

Features

- Two-level hierarchy.
- Cache size of $\mathcal{M}$ bytes.
- Cache-line length of $\mathcal{B}$ bytes.
- Fully associative.
- Optimal, omniscient replacement.

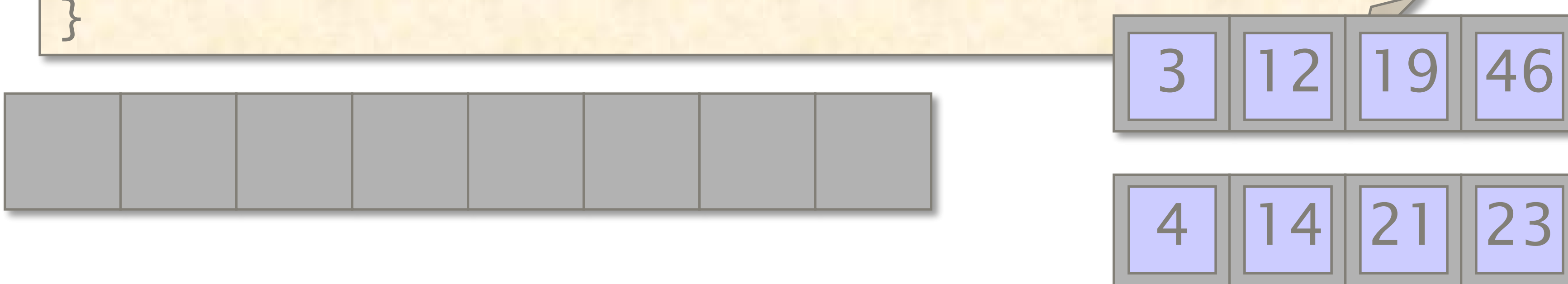


Performance Measures

- ***work*** W (ordinary running time).
- ***cache misses*** Q.

Merging Two Sorted Arrays

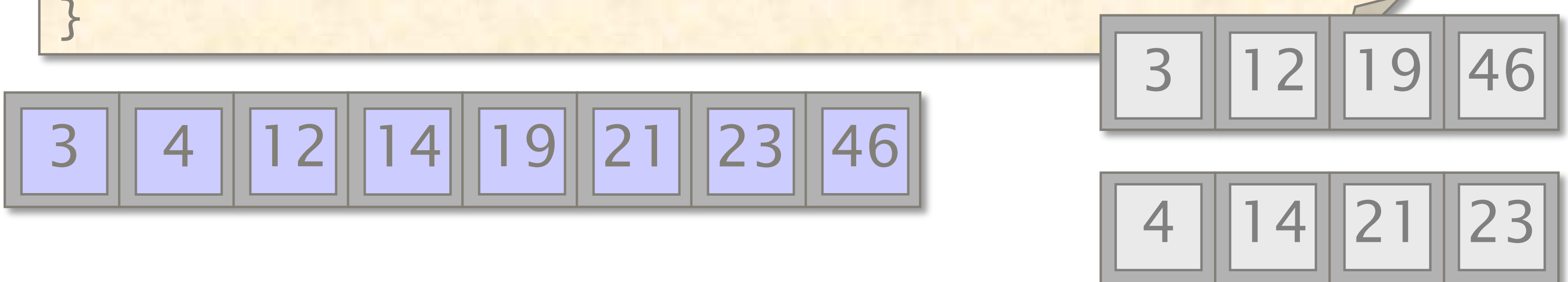
```
void Merge(double *C, double *A, double *B,
           int na, int nb) {
  while (na>0 && nb>0) {
    if (*A <= *B) {
      *C++ = *A++; na--;
    } else {
      *C++ = *B++; nb--;
    }
  }
  while (na>0) {
    *C++ = *A++; na--;
  }
  while (nb>0) {
    *C++ = *B++; nb--;
  }
}
```



Merging Two Sorted Arrays

```
void Merge(double *C, double *A, double *B,
           int na, int nb) {
  while (na>0 && nb>0) {
    if (*A <= *B) {
      *C++ = *A++; na--;
    } else {
      *C++ = *B++; nb--;
    }
  }
  while (na>0) {
    *C++ = *A++; na--;
  }
  while (nb>0) {
    *C++ = *B++; nb--;
  }
}
```

Time to merge n elements = $\Theta(n)$.



Merge Sort

```
void MergeSort(double *B, double *A, int n) {
  if (n==1) {
    B[0] = A[0];
  } else {
    double *C = malloc(n*sizeof(double));
    MergeSort(C, A, n/2);
    MergeSort(C+n/2, A+n/2, n-n/2);
    Merge(B, C, C+n/2, n/2, n-n/2);
} }
```

19	3	12	46	33	4	21	14

Merge Sort

```
void MergeSort(double *B, double *A, int n) {
  if (n==1) {
    B[0] = A[0];
  } else {
    double *C = malloc(n*sizeof(double));
    MergeSort(C, A, n/2);
    MergeSort(C+n/2, A+n/2, n-n/2);
    Merge(B, C, C+n/2, n/2, n-n/2);
} }
```

19	3	12	46	33	4	21	14

recursively sort *recursively sort*

3	12	19	46	4	14	21	33

merge

3	4	12	14	19	21	33	46

Work of Merge Sort

```
void MergeSort(double *B, double *A, int n) {
  if (n==1) {
    B[0] = A[0];
  } else {
    double *C = malloc(n*sizeof(double));
    MergeSort(C, A, n/2);
    MergeSort(C+n/2, A+n/2, n-n/2);
    Merge(B, C, C+n/2, n/2, n-n/2);
  }
}
```

$$W(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2W(n/2) + \Theta(n) & \text{otherwise.} \end{cases}$$

Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.

Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.

$$W(n)$$

Recursion tree

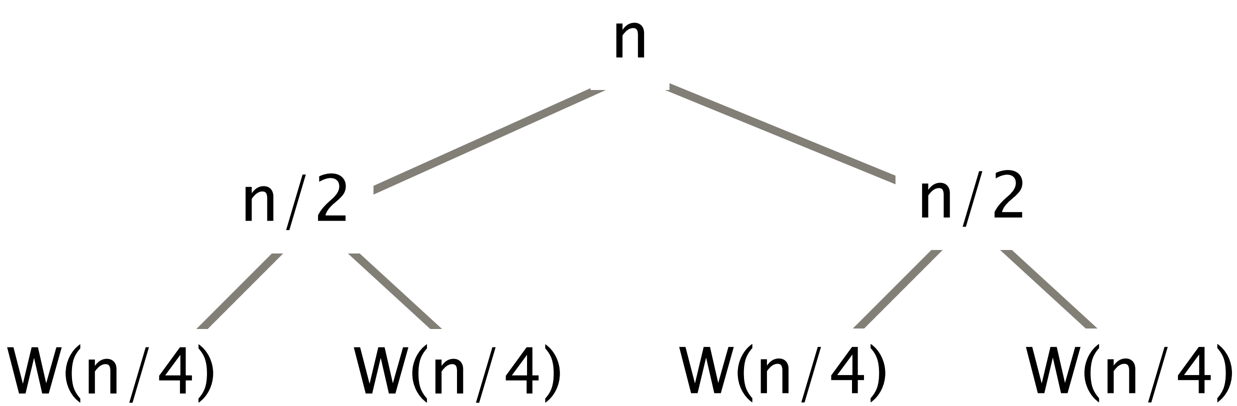
Solve $W(n) = 2W(n/2) + \Theta(n)$.

$$n$$

$$W(n/2) \qquad\qquad W(n/2)$$

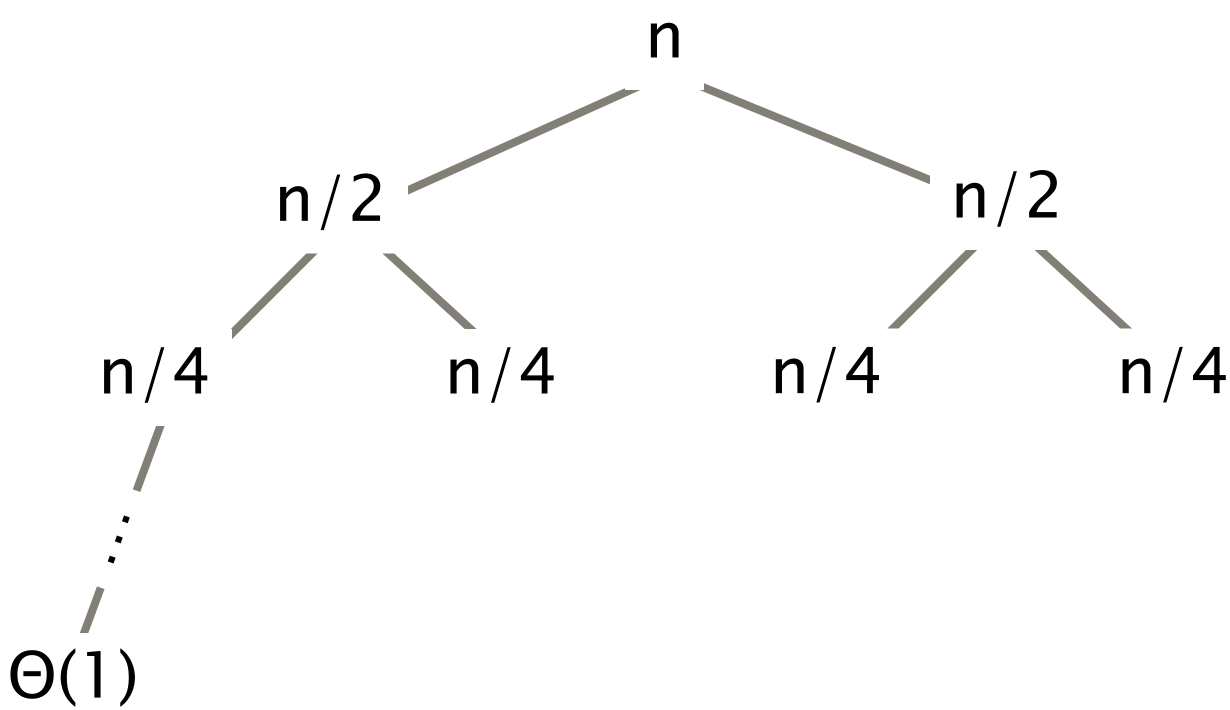
Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.



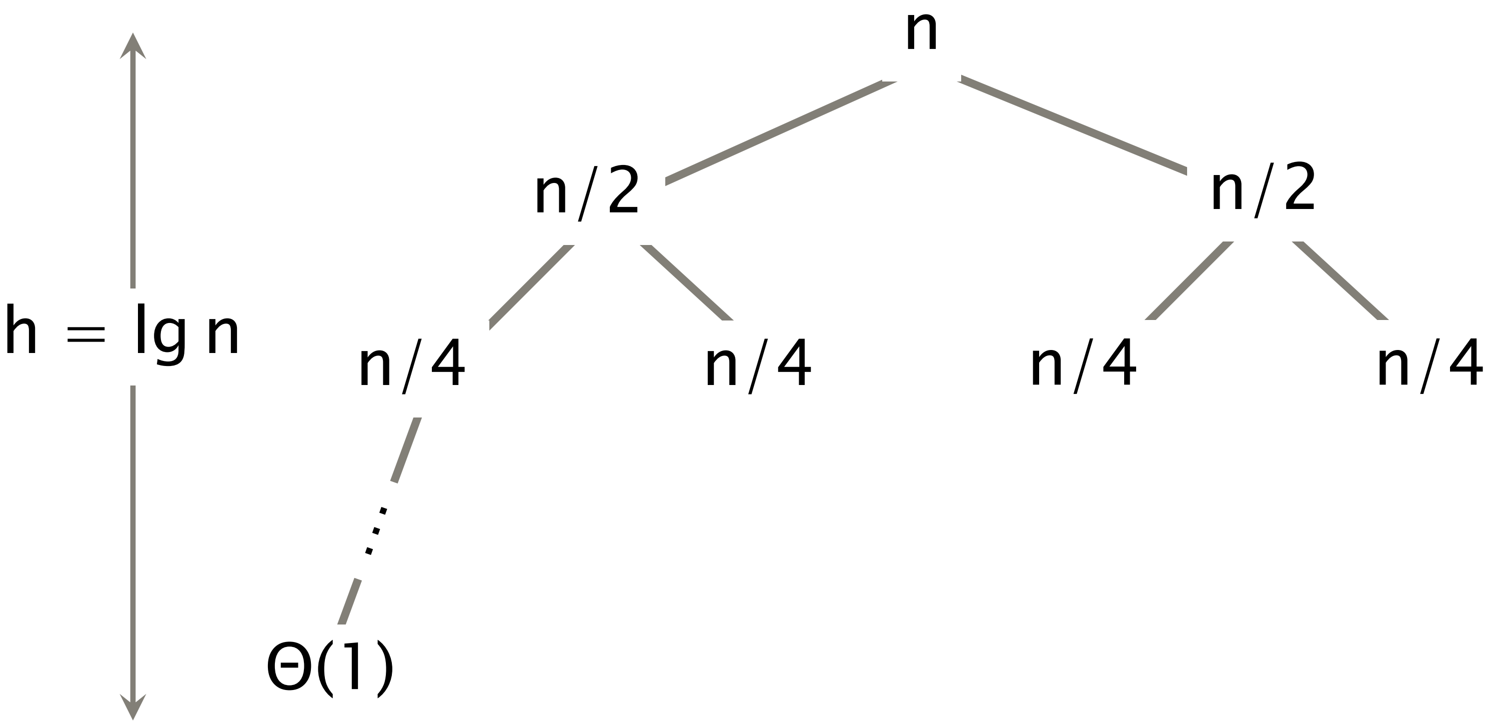
Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.



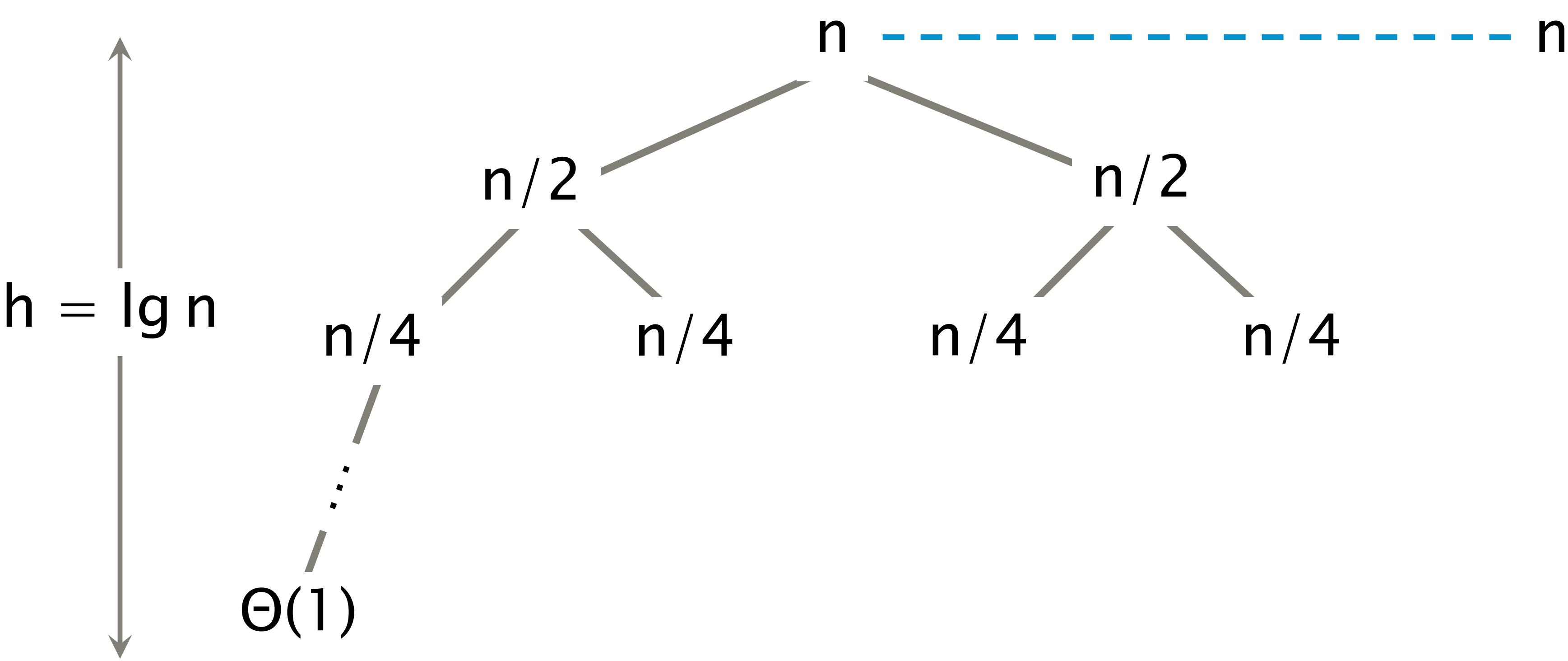
Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.



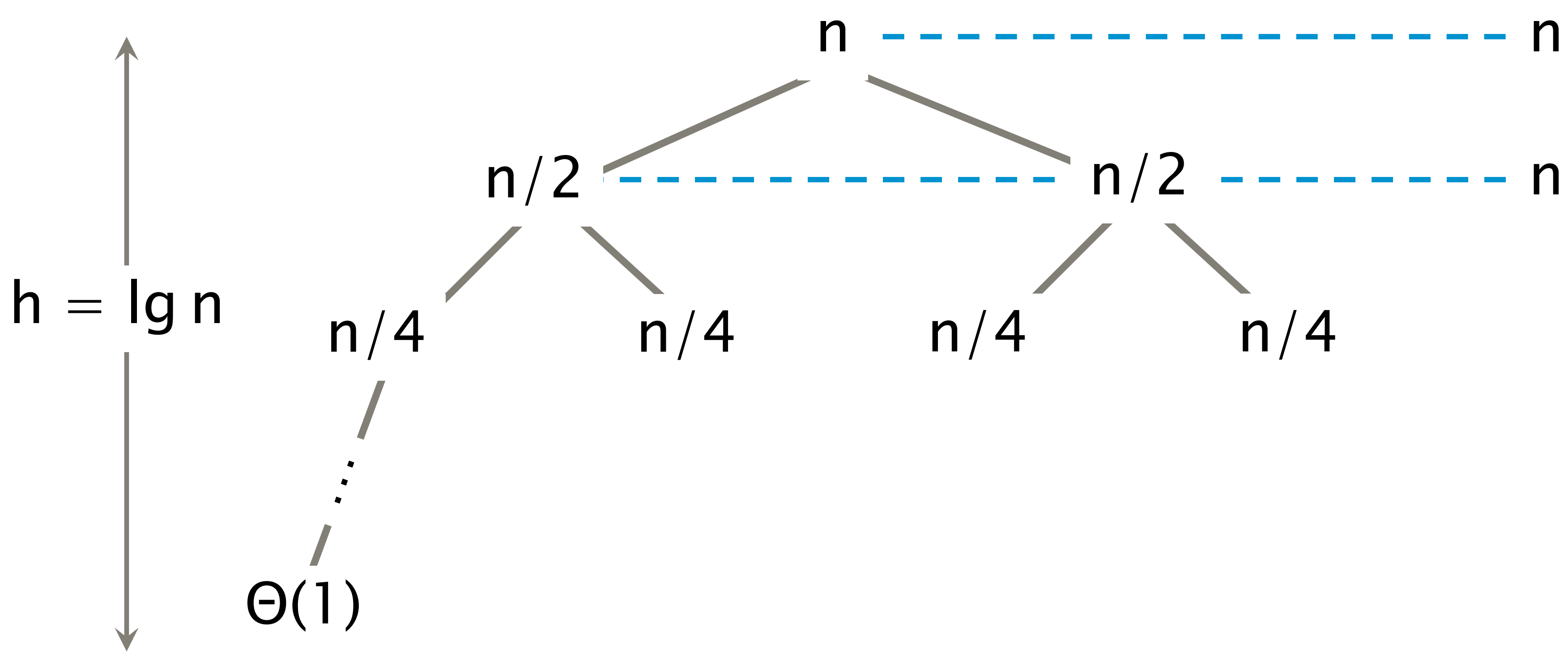
Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.



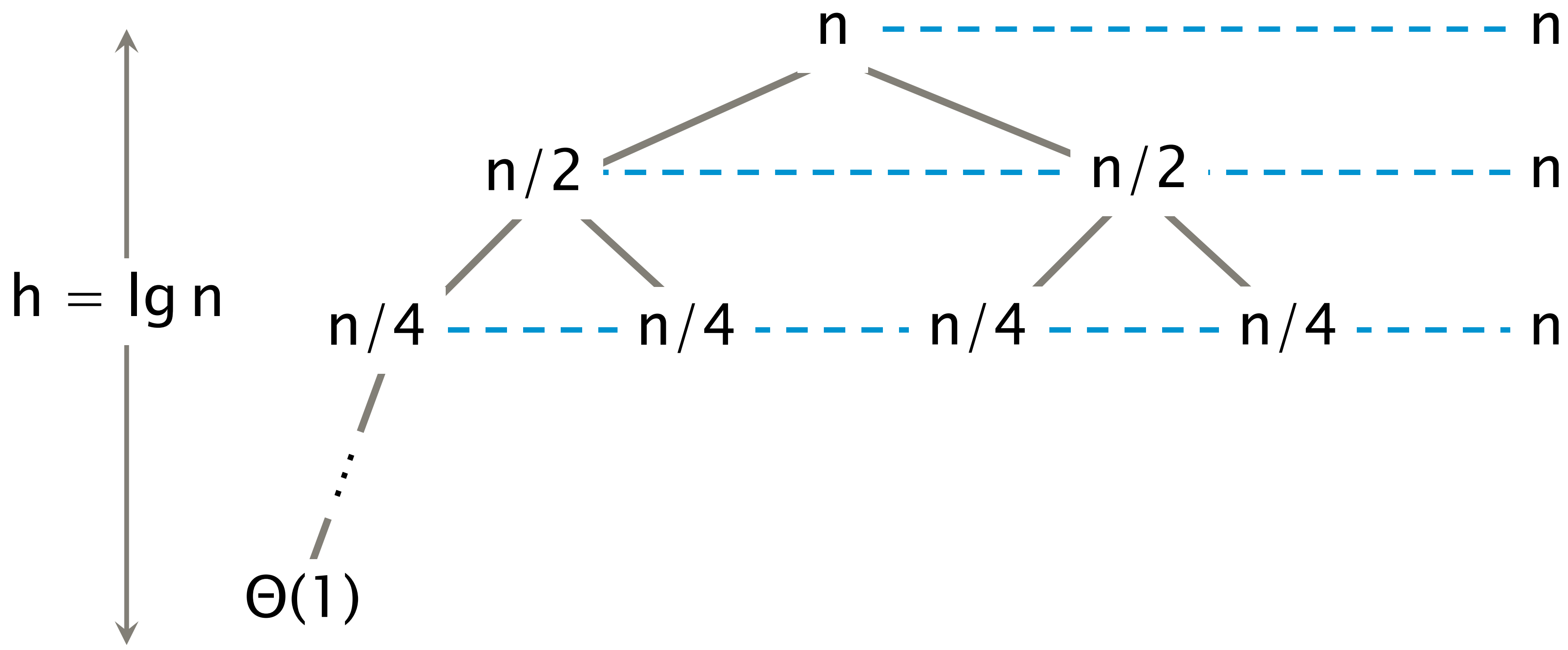
Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.



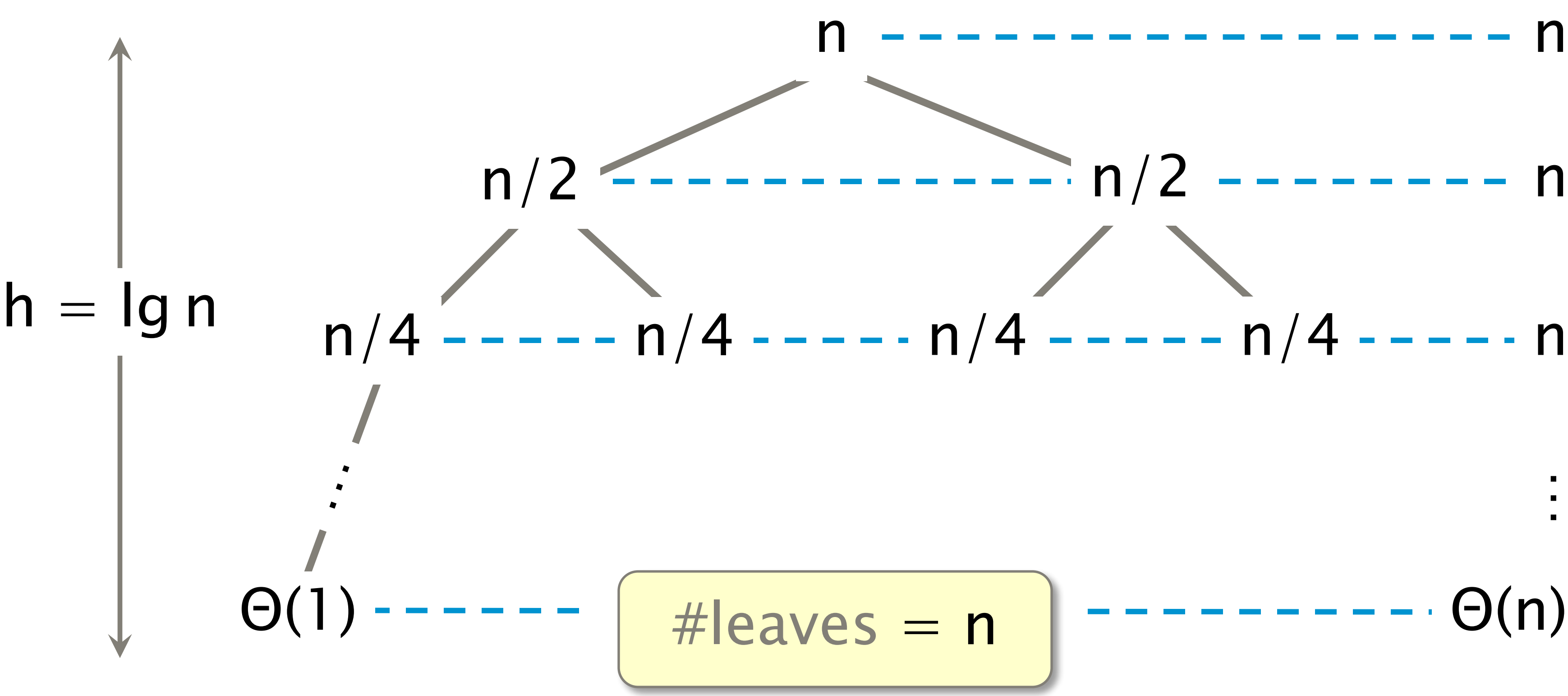
Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.



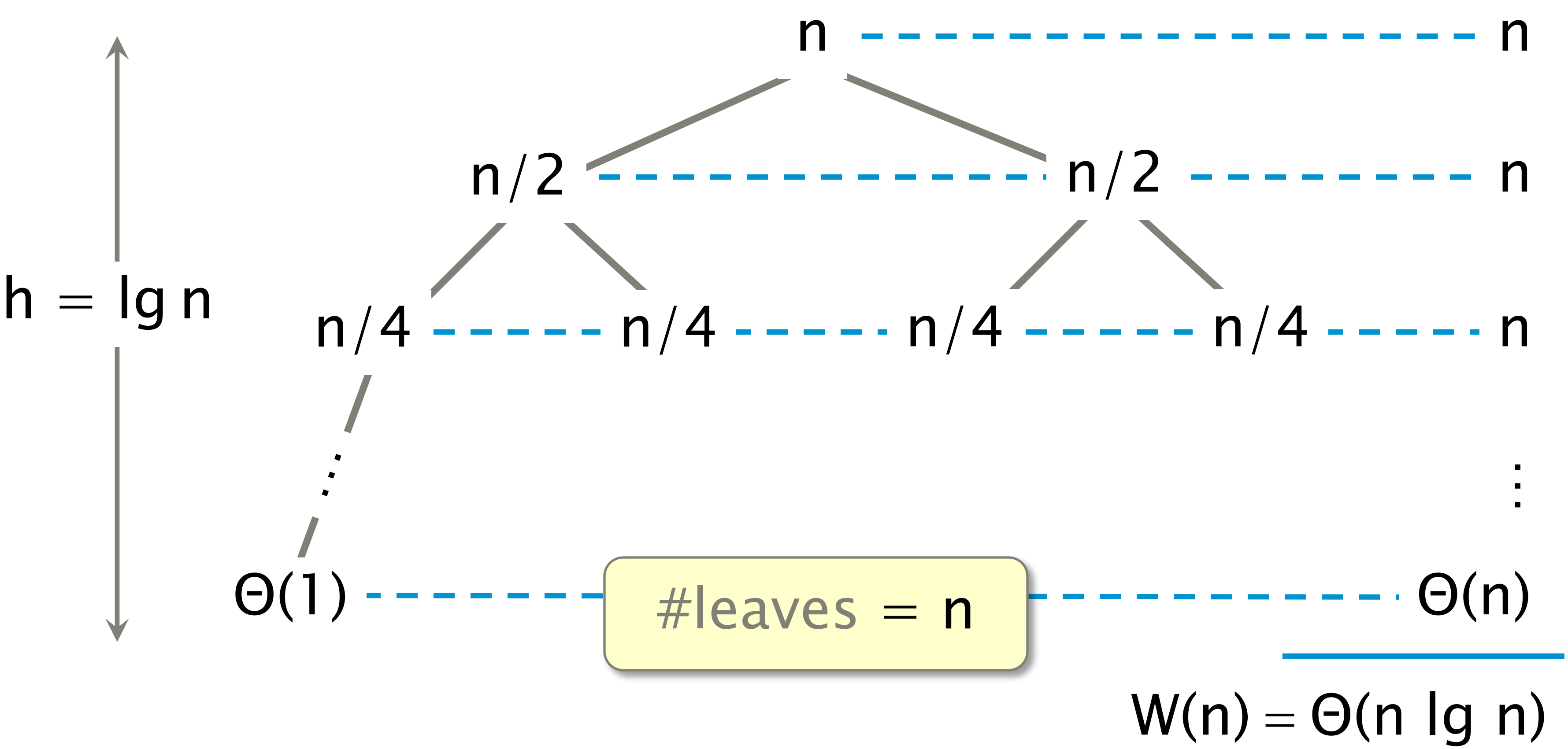
Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.



Recursion tree

Solve $W(n) = 2W(n/2) + \Theta(n)$.



Now with Caching

Merge subroutine

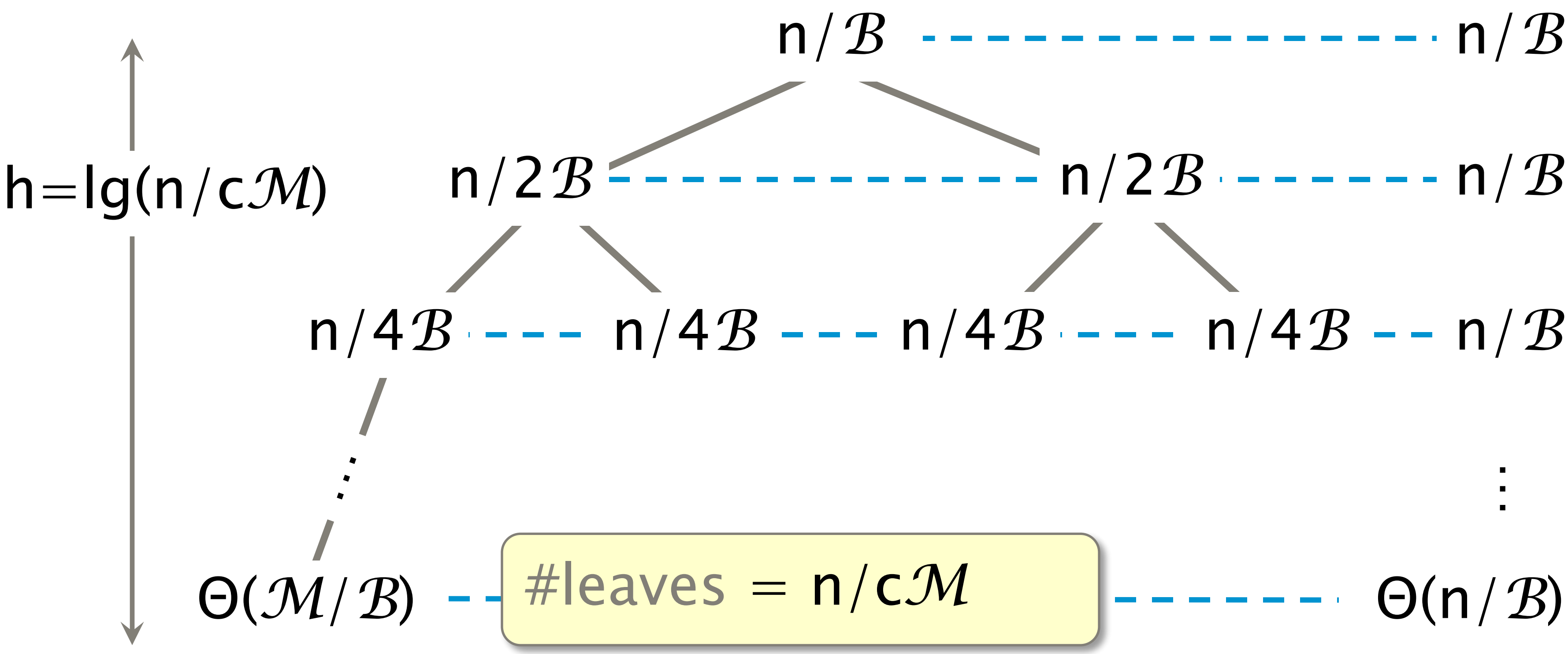
$$Q(n) = \Theta(n/\mathcal{B}).$$

Merge sort

$$Q(n) = \begin{cases} \Theta(n/\mathcal{B}) & \text{if } n \le c\mathcal{M}, \text{ const } c \le 1. \\ 2Q(n/2) + \Theta(n/\mathcal{B}) & \text{otherwise.} \end{cases}$$

Recursion tree

$$Q(n) = \begin{cases} \Theta(n/\mathcal{B}) & \text{if } n \le c\mathcal{M}, \text{ const } c \le 1. \\ 2Q(n/2) + \Theta(n/\mathcal{B}) & \text{otherwise.} \end{cases}$$



$$Q(n) = \Theta((n/\mathcal{B}) \lg(n/\mathcal{M}))$$

Bottom Line for Merge Sort

$$Q(n) = \begin{cases} \Theta(n/\mathcal{B}) & \text{if } n \le c\mathcal{M}, \text{ const } c \le 1. \\ 2Q(n/2) + \Theta(n/\mathcal{B}) & \text{otherwise.} \end{cases}$$

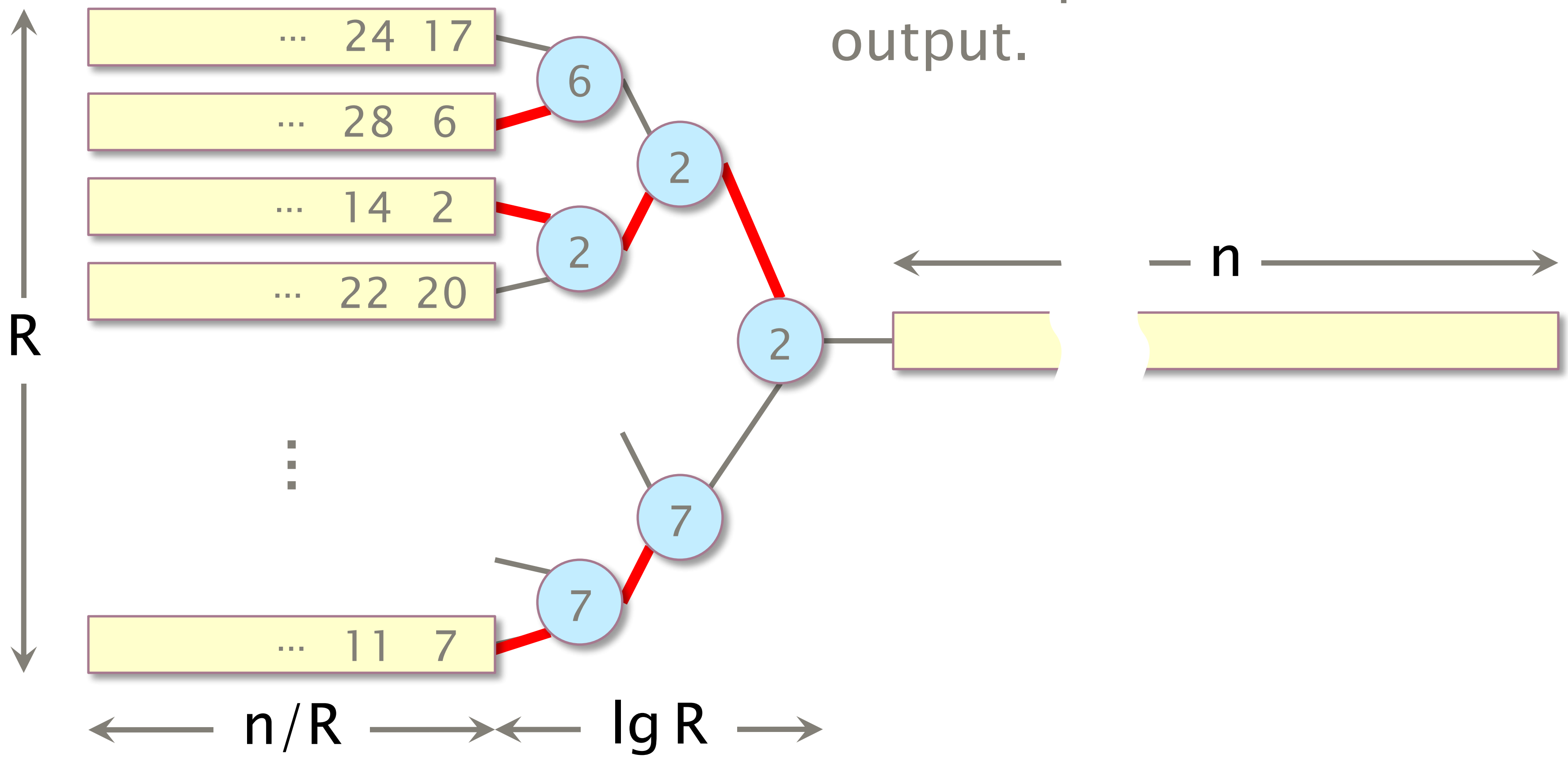
$$= \Theta(\ (n/\mathcal{B}) \lg(n/\mathcal{M})\ )$$

- For $n \gg \mathcal{M}$, we have $\lg(n/\mathcal{M}) \approx \lg n$, and thus $W(n)/Q(n) \approx \Theta(\mathcal{B})$.
- For $n \approx \mathcal{M}$, we have $\lg(n/\mathcal{M}) \approx \Theta(1)$, and thus $W(n)/Q(n) \approx \Theta(\mathcal{B} \lg n)$.

Multiway Merging

Idea: Merge $R < n$ subarrays with a tournament.

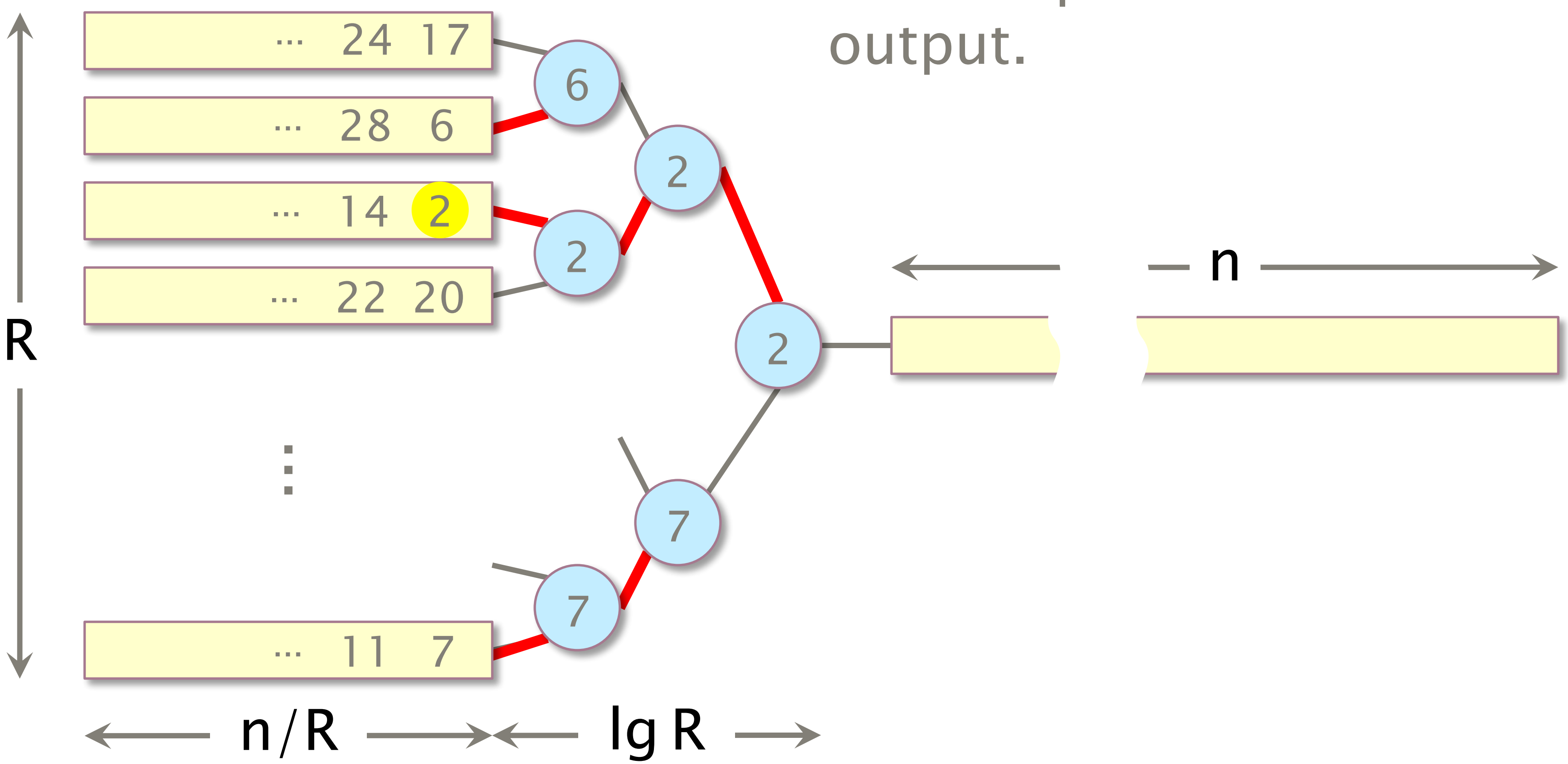
- Tournament takes $\Theta(R)$ work to produce the first output.



Multiway Merging

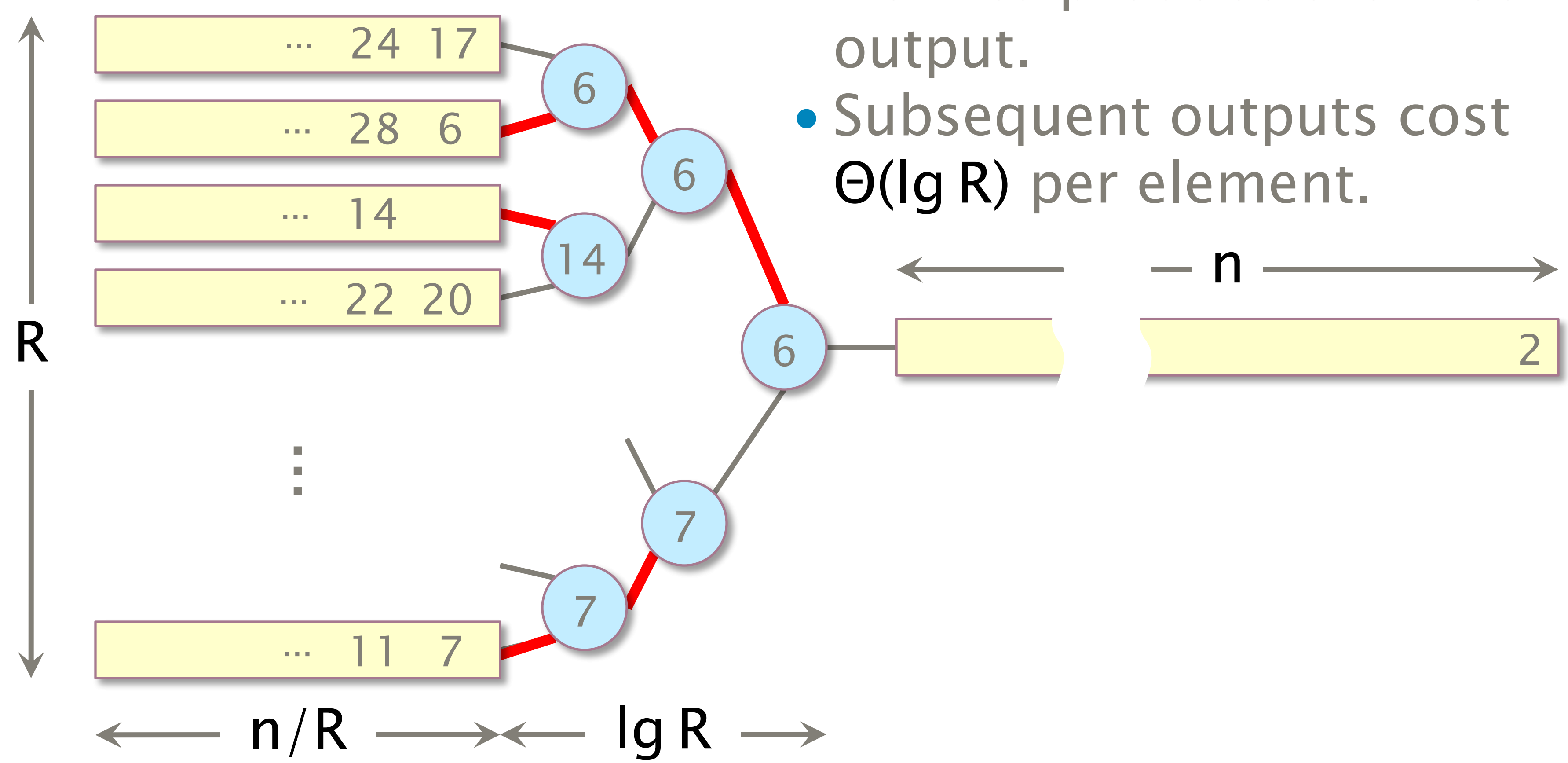
Idea: Merge $R < n$ subarrays with a tournament.

- Tournament takes $\Theta(R)$ work to produce the first output.



Multiway Merging

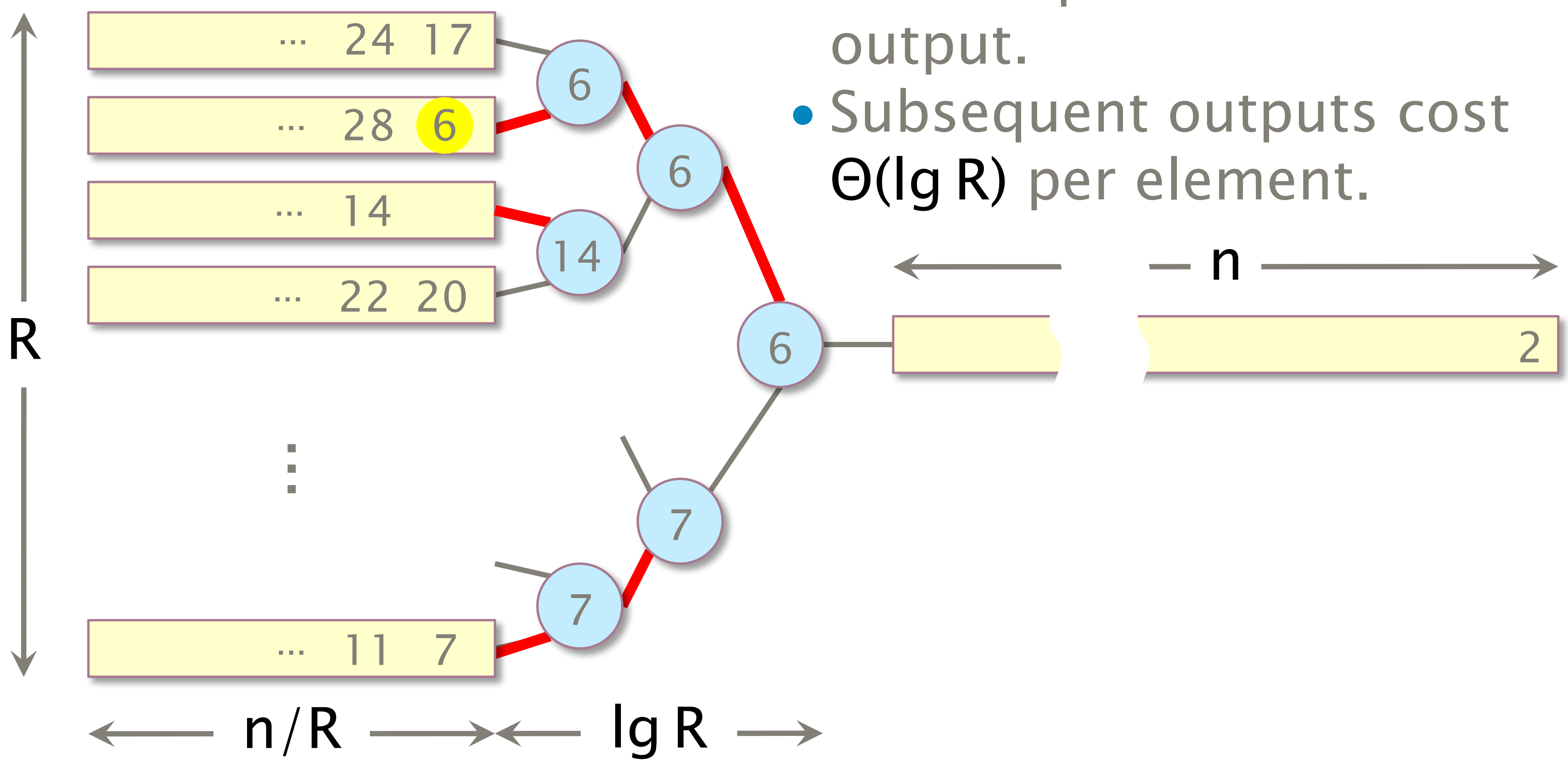
Idea: Merge R < n subarrays with a tournament.



- Tournament takes Θ(R) work to produce the first output.
- Subsequent outputs cost Θ(lg R) per element.

Multiway Merging

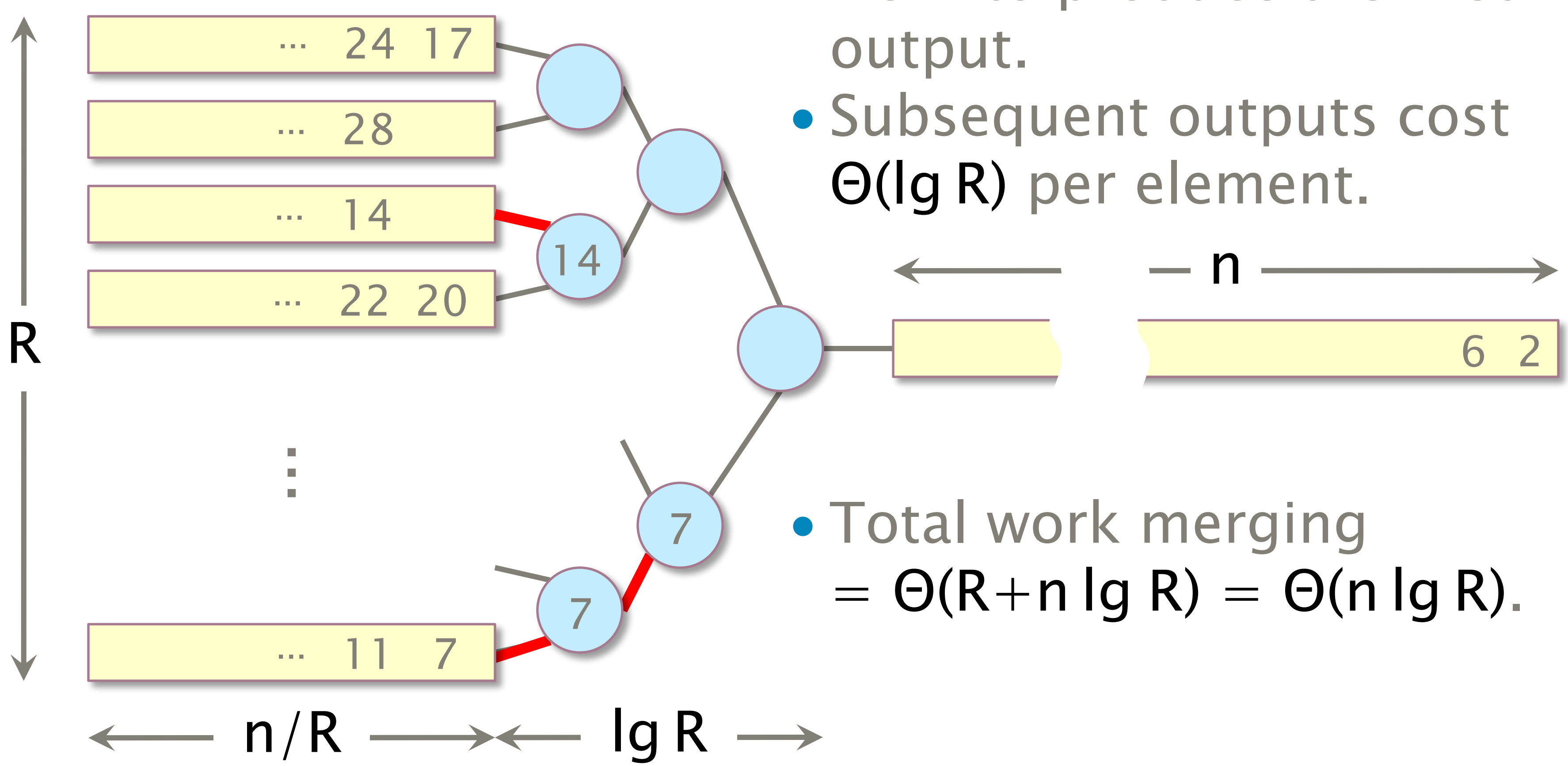
Idea: Merge R < n subarrays with a tournament.



- Tournament takes Θ(R) work to produce the first output.
- Subsequent outputs cost Θ(lg R) per element.

Multiway Merging

Idea: Merge $R < n$ subarrays with a tournament.



- Tournament takes $\Theta(R)$ work to produce the first output.
- Subsequent outputs cost $\Theta(\lg R)$ per element.
- Total work merging $= \Theta(R+n \lg R) = \Theta(n \lg R)$.

Multiway Merge Sort

$$W(n) = \begin{cases} \Theta(1) \text{ if } n = 1, \\ R \cdot W(n/R) + \Theta(n \lg R) \text{ otherwise.} \end{cases}$$

recursion tree

$n \lg R$ ---------- $n \lg R$

R

$\log_R n$ $(n/R) \lg R$ $(n/R) \lg R$ -- $(n/R) \lg R$ --- $n \lg R$

R

$(n/R^2) \lg R$ $(n/R^2) \lg R$ -- $(n/R^2) \lg R$ ---------- $n \lg R$

#leaves $= n$

$\Theta(1)$ -- $\Theta(n)$

$$\begin{aligned} W(n) &= \Theta((n \lg R) \log_R n + n) \\ &= \Theta((n \lg R)(\lg n)/\lg R + n) \\ &= \Theta(n \lg n) \end{aligned}$$

Same as binary merge sort.

Cache Recurrence

Assume that $R < c\mathcal{M}/\mathcal{B}$ for suff. small const $c \leq 1$.

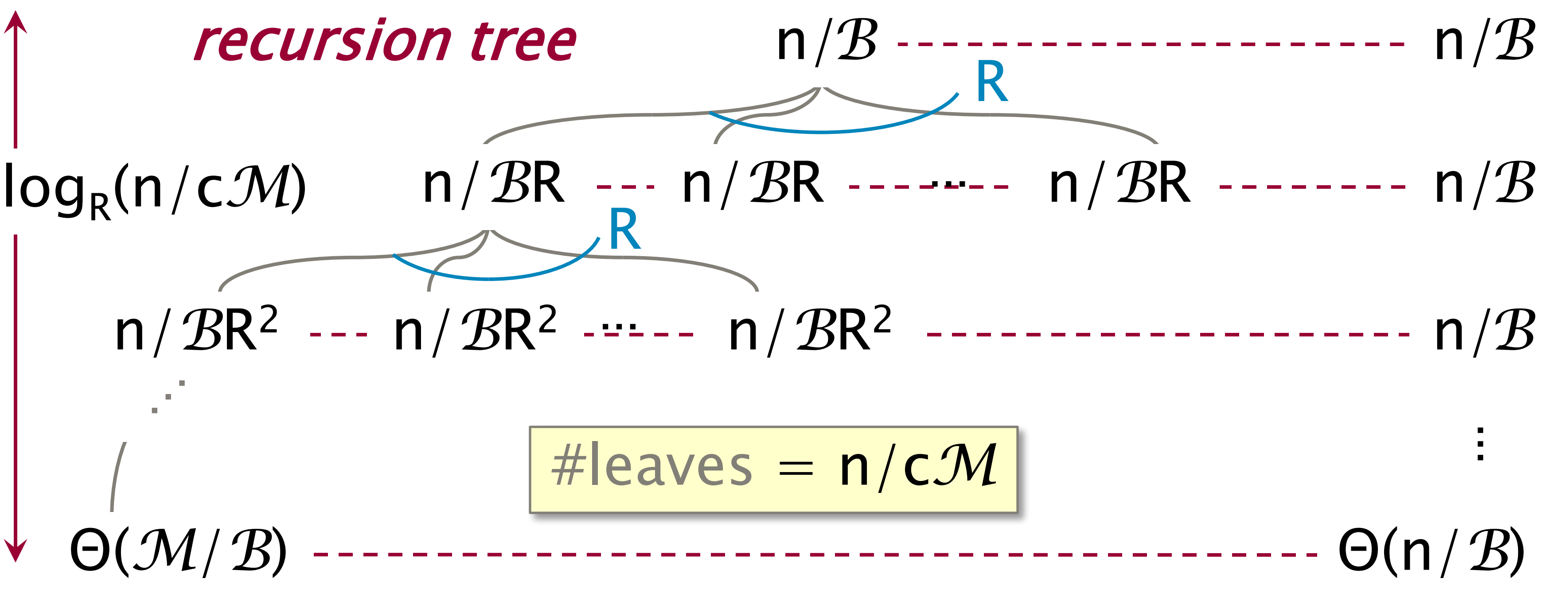
Consider the R–way merging of contiguous arrays of total size n. If $R < c\mathcal{M}/\mathcal{B}$, the entire tournament plus 1 block from each array can fit in cache.
$\Rightarrow Q(n) \leq \Theta(n/\mathcal{B})$.

R–way merge sort

$$Q(n) \leq \begin{cases} \Theta(n/\mathcal{B}) & \text{if } n < c\mathcal{M}, \\ R \cdot Q(n/R) + \Theta(n/\mathcal{B}) & \text{otherwise.} \end{cases}$$

Cache Analysis

$$Q(n) \le \begin{cases} \Theta(n/\mathcal{B}) & \text{if } n < c\mathcal{M}, \\ R \cdot Q(n/R) + \Theta(n/\mathcal{B}) & \text{otherwise.} \end{cases}$$



$$Q(n) = \Theta((n/\mathcal{B}) \log_R(n/\mathcal{M}))$$

Tune

We have

$$Q(n) = \Theta((n/\mathcal{B}) \log_R(n/\mathcal{M})) \ ,$$

which decreases as R increases. $\therefore$ Choose R as big as possible $\Rightarrow R = \Theta(\mathcal{M}/\mathcal{B})$.

By the tall-cache assumption and the fact that $\log_{\mathcal{M}}(n/\mathcal{M}) = \Theta((\lg n)/\lg \mathcal{M})$, we have

$$\begin{aligned} Q(n) &= \Theta((n/\mathcal{B}) \log_{\mathcal{M}/\mathcal{B}}(n/\mathcal{M})) \\ &= \Theta((n/\mathcal{B}) \log_{\mathcal{M}}(n/\mathcal{M})) \\ &= \Theta((n \lg n)/\mathcal{B} \lg \mathcal{M}) \ . \end{aligned}$$

Hence, we have $W(n)/Q(n) \approx \Theta(\mathcal{B} \lg \mathcal{M})$.

Multiway versus Binary Merge Sort

We have

$$Q_{\text{multiway}}(n) = \Theta((n \lg n)/\mathcal{B} \lg \mathcal{M})$$

versus

$$Q_{\text{binary}}(n) = \Theta((n/\mathcal{B}) \lg (n/\mathcal{M})).$$

If $n \gg \mathcal{M}$, then $\lg (n/\mathcal{M}) \approx \lg n$, and thus multiway merge sort saves a factor of $\Theta(\lg \mathcal{M})$ in cache misses.

Example

- L1-cache: $\mathcal{M} = 2^{15}$, $\mathcal{B} = 2^6 \Rightarrow 9\times$ savings.
- L2-cache: $\mathcal{M} = 2^{18}$, $\mathcal{B} = 2^6 \Rightarrow 12\times$ savings.
- L3-cache: $\mathcal{M} = 2^{23}$, $\mathcal{B} = 2^6 \Rightarrow 17\times$ savings.

Optimal Cache-Oblivious Sorting

Funnelsort [FLPR99]

1. Recursively sort $n^{1/3}$ groups of $n^{2/3}$ items.
2. Merge the sorted groups with an $n^{1/3}$-funnel.

A *k-funnel* merges k^3 items in k sorted lists, incurring at most

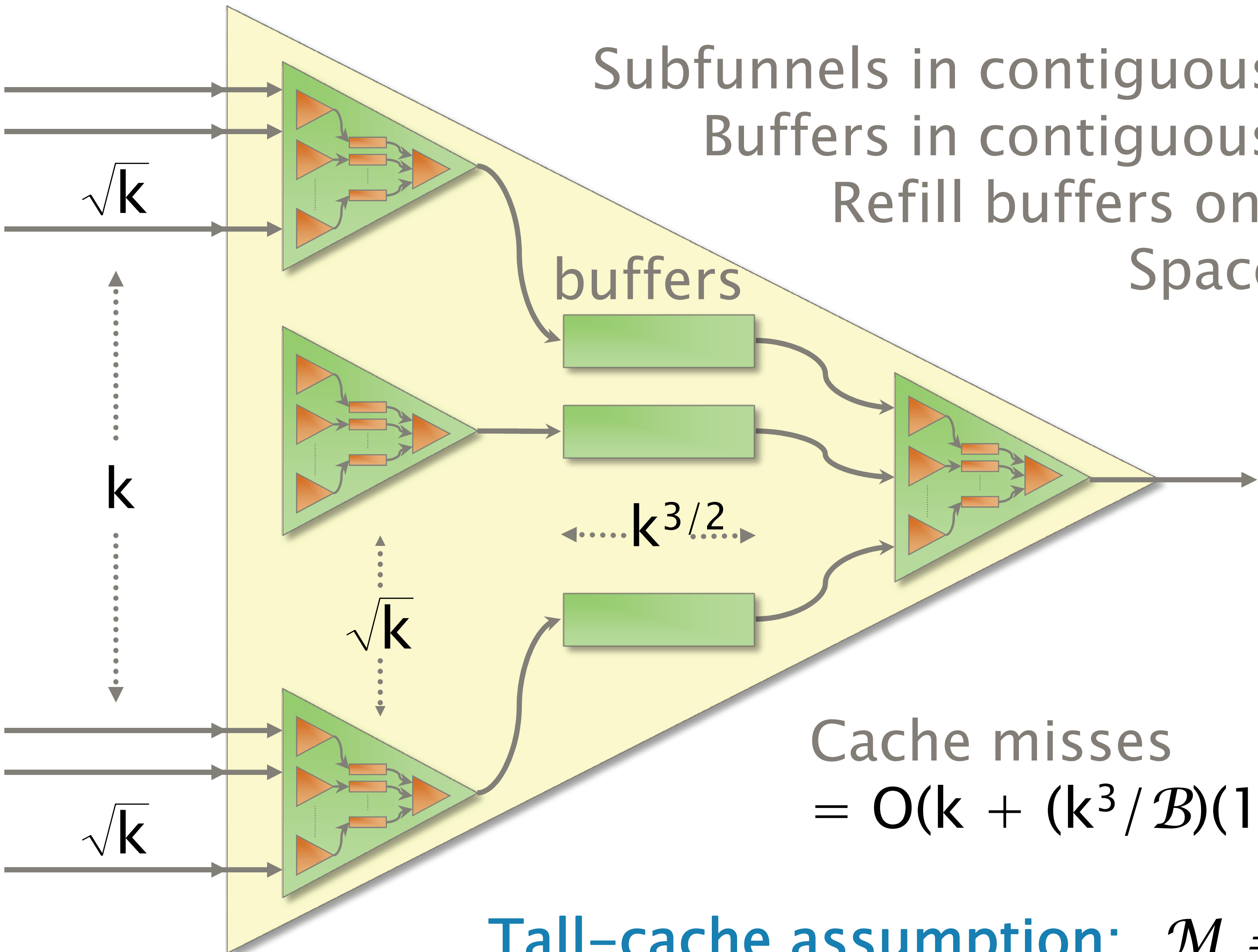
$$\Theta\left(k + (k^3/\mathcal{B})(1 + \log_{\mathcal{M}} k)\right)$$

cache misses. Thus, funnelsort incurs

$$\begin{aligned} Q(n) &\le n^{1/3} Q\left(n^{2/3}\right) + \Theta\left(n^{1/3} + (n/\mathcal{B})(1 + \log_{\mathcal{M}} n)\right) \\ &= \Theta\left(1 + (n/\mathcal{B})(1 + \log_{\mathcal{M}} n)\right), \end{aligned}$$

cache misses, which is asymptotically optimal [AV88].

Construction of a k-funnel



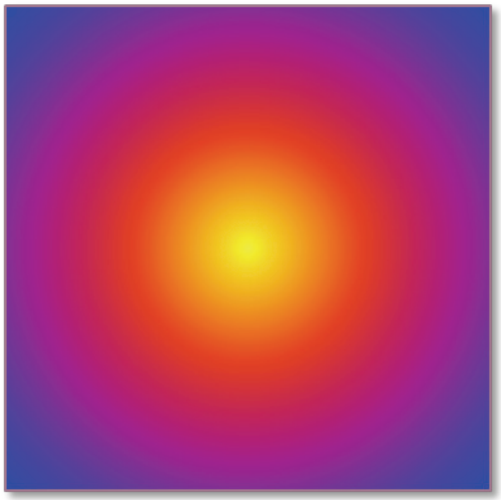
Subfunnels in contiguous storage.
Buffers in contiguous storage.
Refill buffers on demand.
Space = $O(k^2)$.

Cache misses
$= O(k + (k^3/\mathcal{B})(1+\log_{\mathcal{M}} k))$

Tall-cache assumption: $\mathcal{M} = \Omega(\mathcal{B}^2)$.

Heat Diffusion

2D heat equation



Let u(t,x,y) = temperature at time t at point (x,y).

$$\frac{\partial u}{\partial t} = \alpha\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

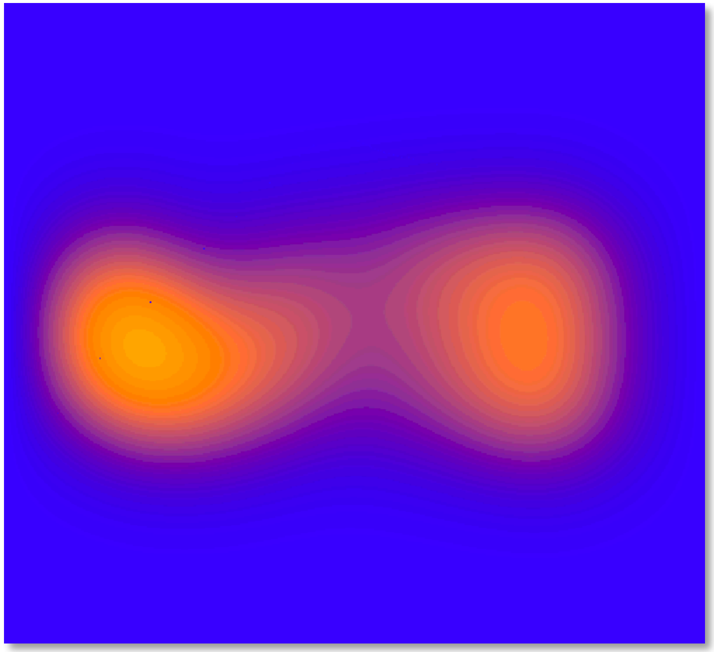
α is the ***thermal diffusivity***.

Acknowledgment

These stencil slides were heavily inspired by originals due to Matteo Frigo.

2D Heat-Diffusion Simulation

Before



After

1D Heat Equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

Finite-Difference Approximation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

$$\frac{\partial u}{\partial t}(t,x) \approx \frac{u(t+\Delta t, x) - u(t,x)}{\Delta t},$$

$$\frac{\partial u}{\partial x}(t,x) \approx \frac{u(t, x+\Delta x/2) - u(t, x-\Delta x/2)}{\Delta x},$$

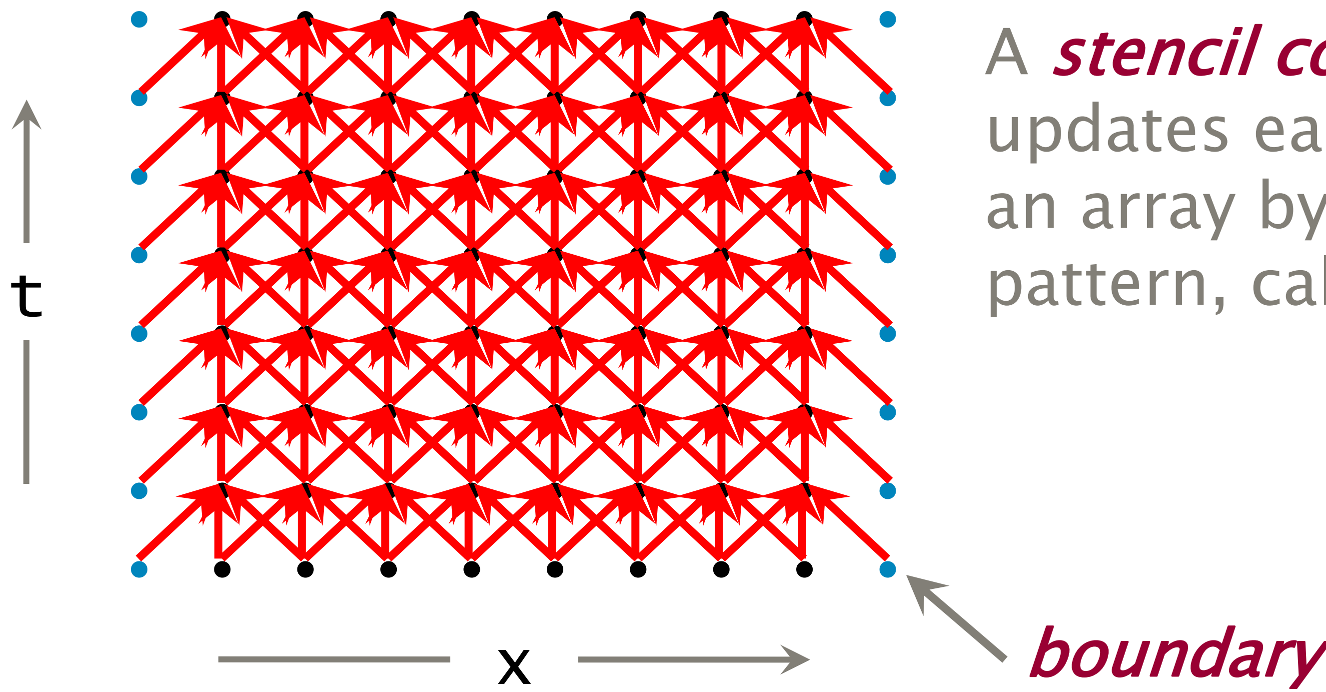
$$\frac{\partial^2 u}{\partial x^2}(t,x) \approx \frac{(\partial u/\partial x)(t, x+\Delta x/2) - (\partial u/\partial x)(t, x-\Delta x/2)}{\Delta x}$$

$$\approx \frac{u(t, x+\Delta x) - 2u(t,x) + u(t, x-\Delta x)}{(\Delta x)^2}.$$

The 1D heat equation thus reduces to

$$\frac{u(t+\Delta t, x) - u(t,x)}{\Delta t} = \alpha \frac{u(t, x+\Delta x) - 2u(t,x) + u(t, x-\Delta x)}{(\Delta x)^2}.$$

3-Point Stencil



A ***stencil computation*** updates each point in an array by a fixed pattern, called a ***stencil***.

Update rule

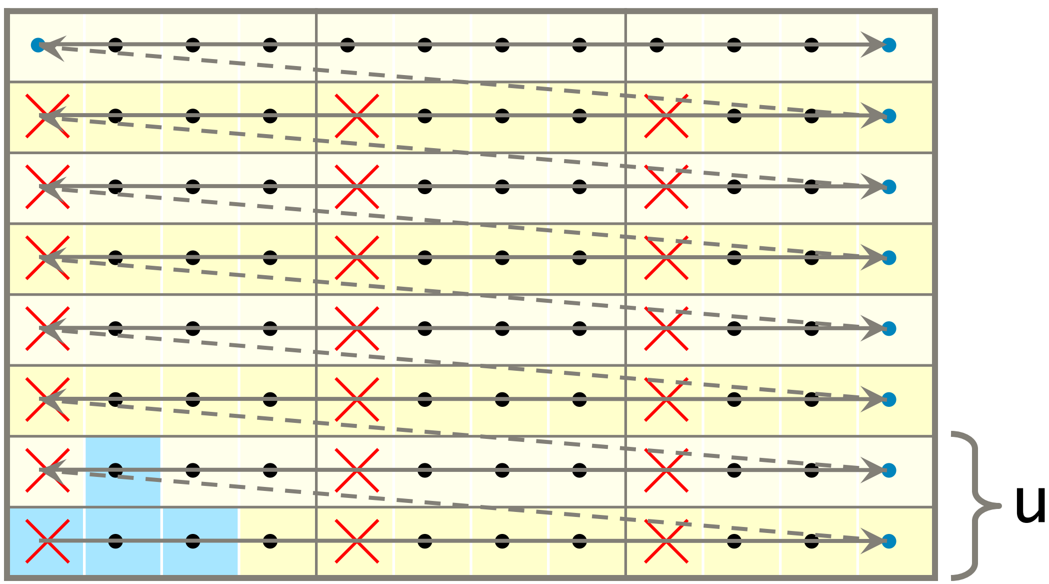
$$u[t+1][x] = u[t][x] + \frac{\alpha \Delta t}{(\Delta x)^2}(u[t][x+1] - 2u[t][x] + u[t][x-1]),$$

Cache Behavior of Looping

```
double u[2][N]; // even-odd trick

static inline double kernel(double * u) {
    return u[0] + ALPHA * (u[-1] - 2*u[0] + u[1]);
}

for (int t = 1; t < T-1; ++t) { // time loop
  for(int x = 1; x < N-1; ++x) // space loop
    u[(t+1)%2][x] = kernel( &u[t%2][x] );
```

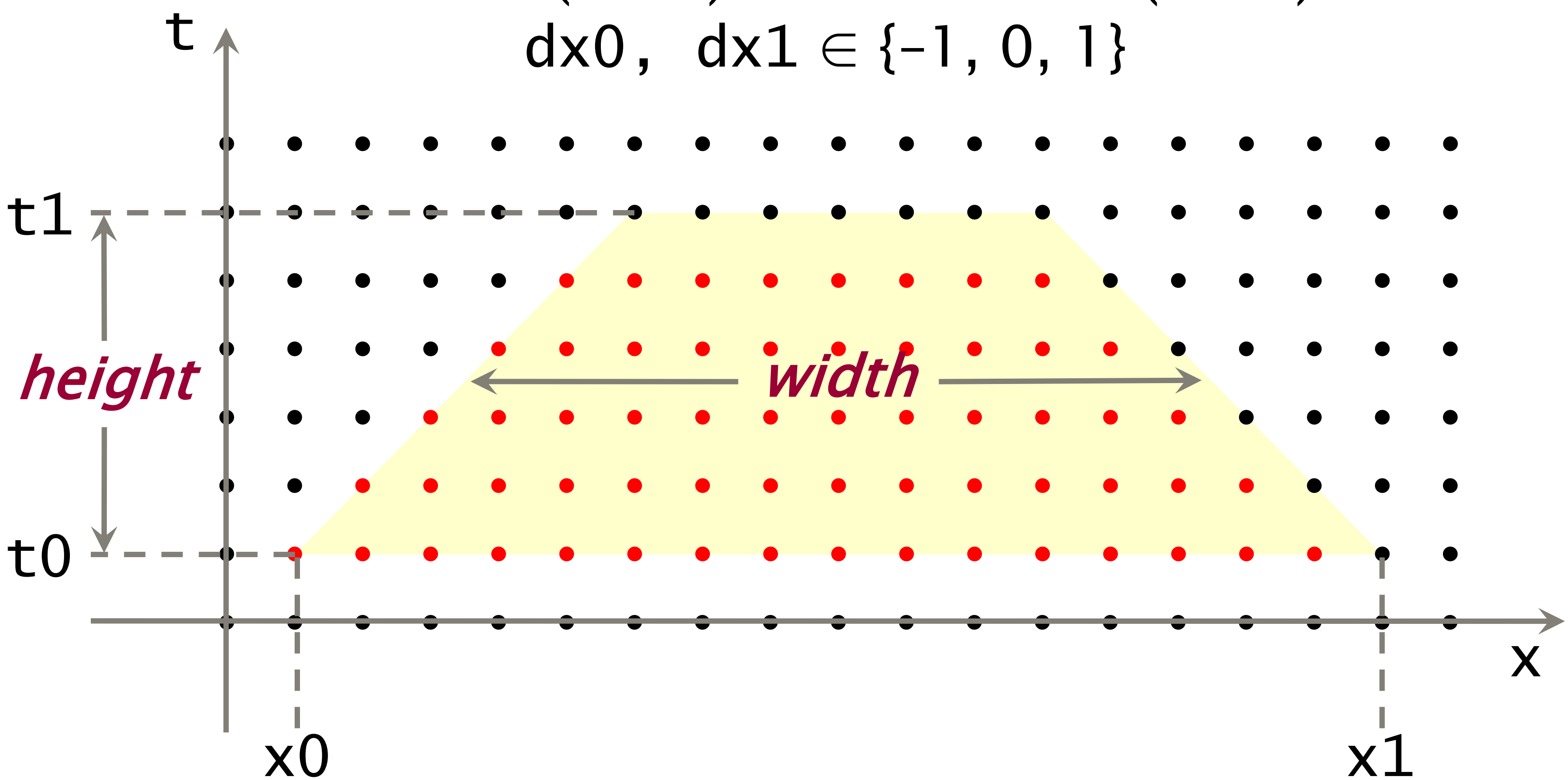


Assuming LRU, if $N > \mathcal{M}$, then $Q = \Theta(NT/\mathcal{B})$.

Cache-Oblivious 3-Point Stencil

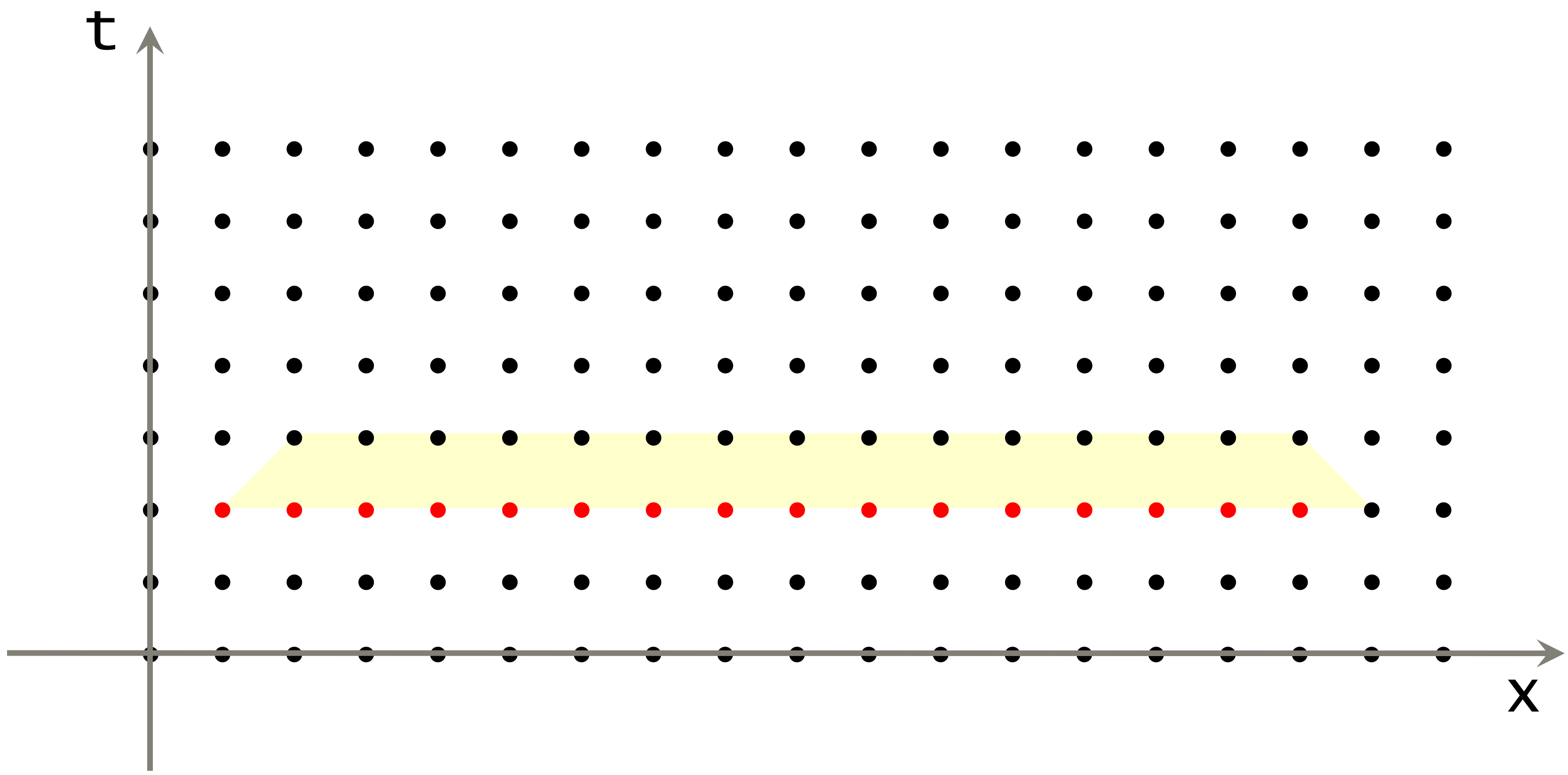
Recursively traverse trapezoidal regions of space-time points `(t,x)` such that

$$t0 \le t < t1$$
$$x0+dx0(t-t0) \le x < x1+dx1(t-t0)$$
$$dx0,\ dx1 \in \{-1, 0, 1\}$$



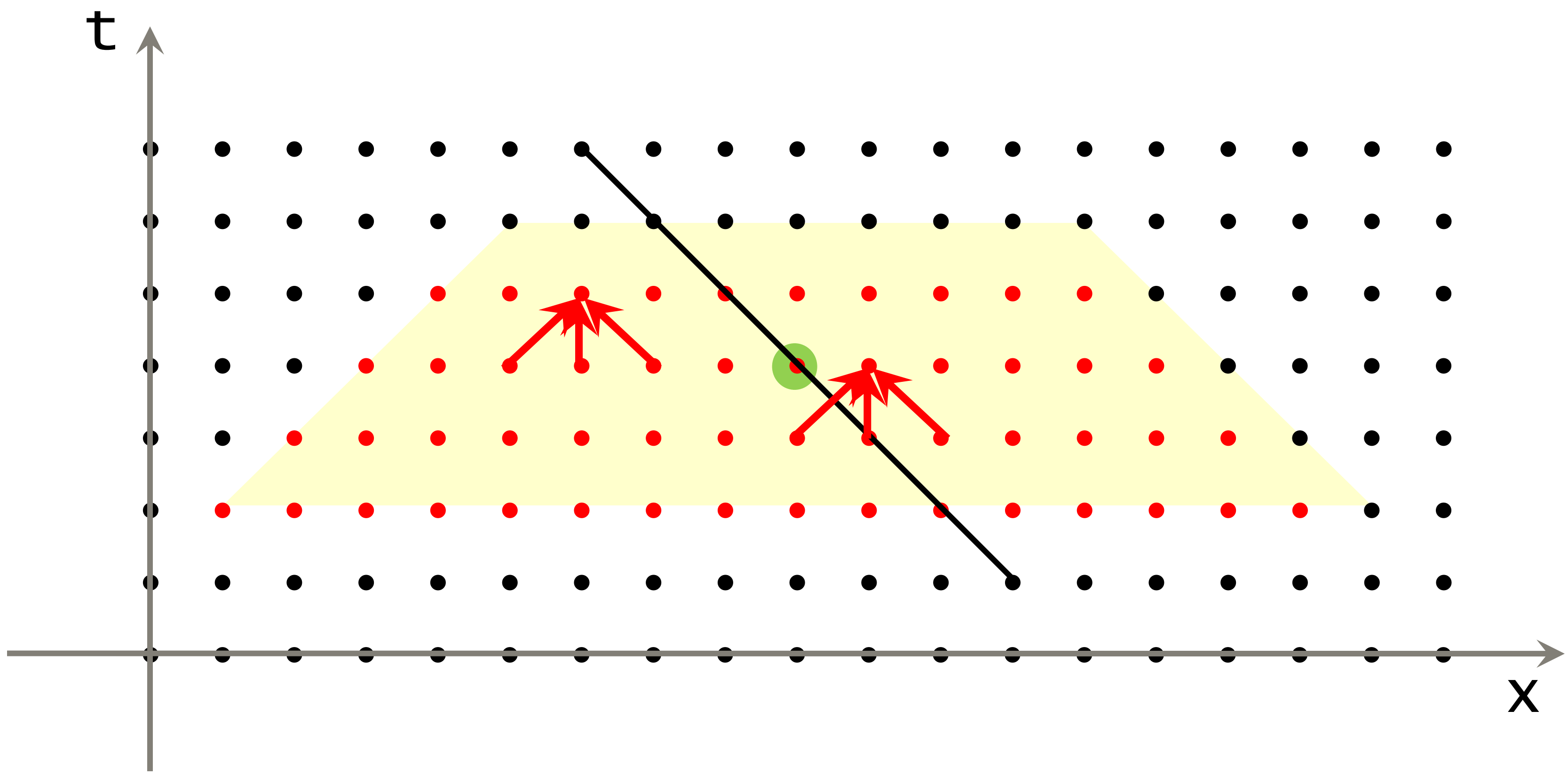
Base Case

If height = 1, compute all space-time points in the trapezoid. Any order of computation is valid, since no point depends on another.



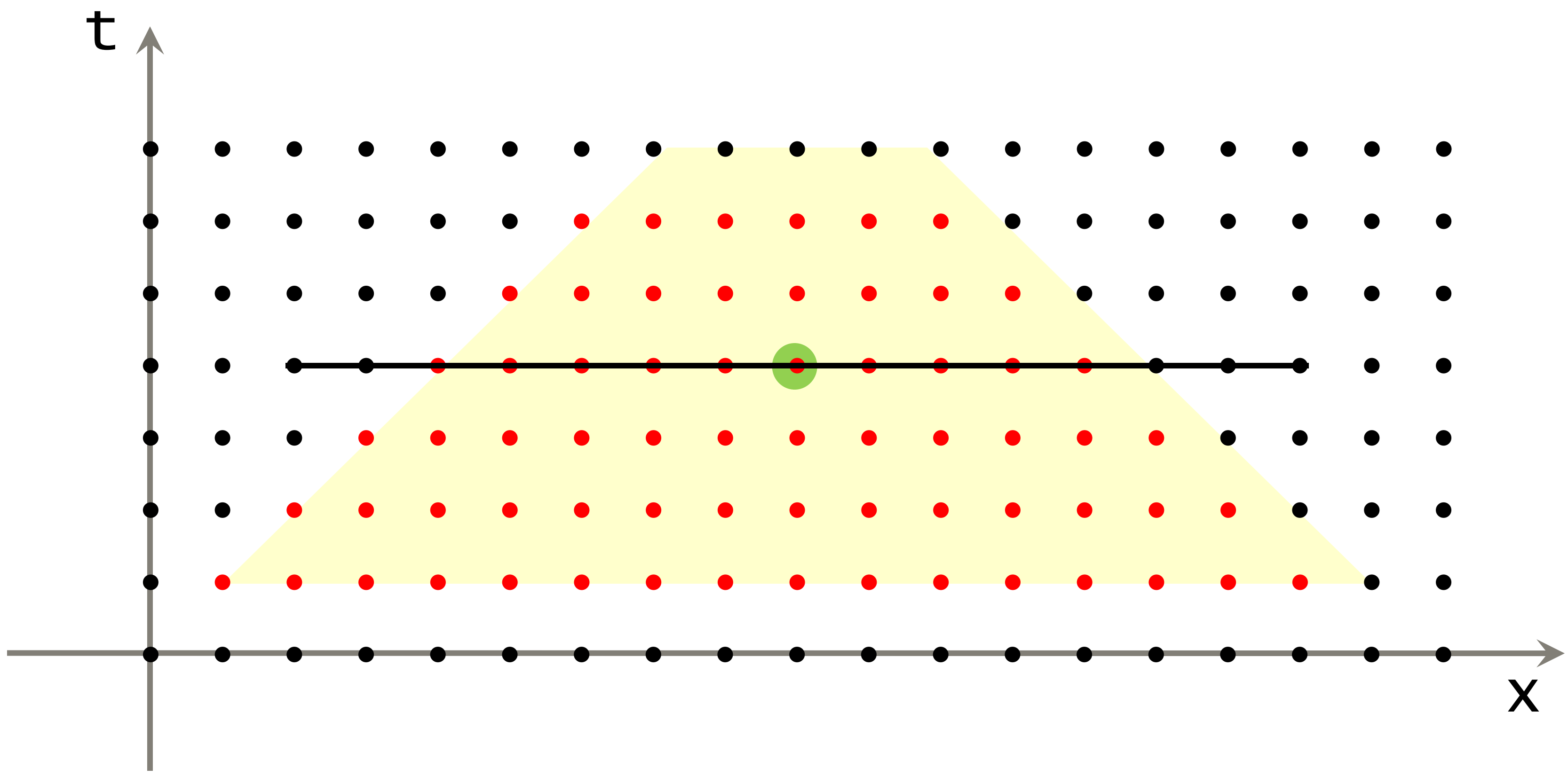
Space Cut

If width $\geq$ 2·height, cut the trapezoid with a line of slope −1 through the center. Traverse the trapezoid on the left first, and then the one on the right.



Time Cut

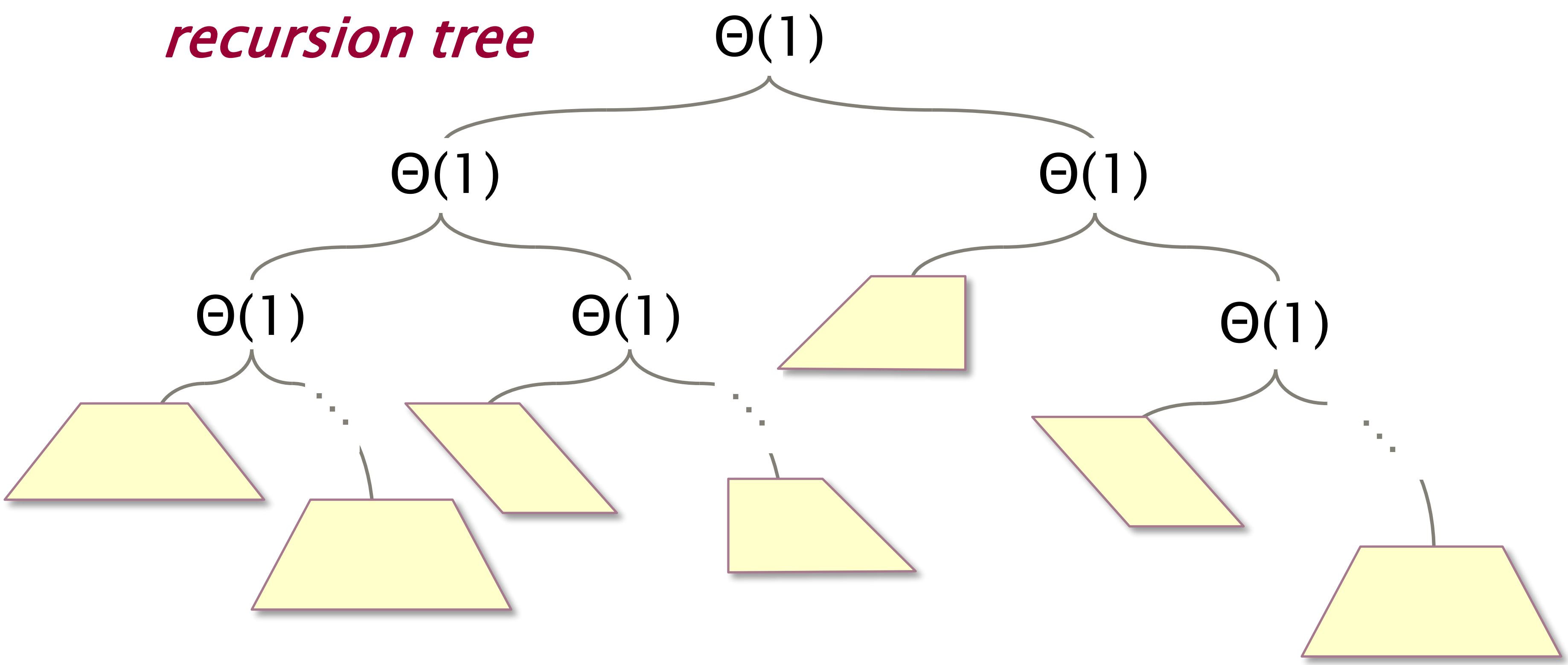
If width < 2·height, cut the trapezoid with a horizontal line through the center. Traverse the bottom trapezoid first, and then the top one.



C Implementation

```
void trapezoid(int t0, int t1, int x0, int dx0,
               int x1, int dx1)
{
  int lt = t1 - t0;
  if (lt == 1) {
      for (int x = x0; x < x1; x++)
        kernel(t, x);
  } else if (lt > 1) {
    if (2 * (x1 - x0) + (dx1 - dx0) * lt >= 4 * lt) {
      int xm = (2 * (x0 + x1) + (2 + dx0 + dx1) * lt) / 4;
      trapezoid(t0, t1, x0, dx0, xm, -1);
      trapezoid(t0, t1, xm, -1, x1, dx1);
    } else {
      int halflt = lt / 2;
      trapezoid(t0, t0 + halflt, x0, dx0, x1, dx1);
      trapezoid(t0 + halflt, t1, x0 + dx0 * halflt,
                dx0, x1 + dx1 * halflt, dx1);
    }
  }
}
```

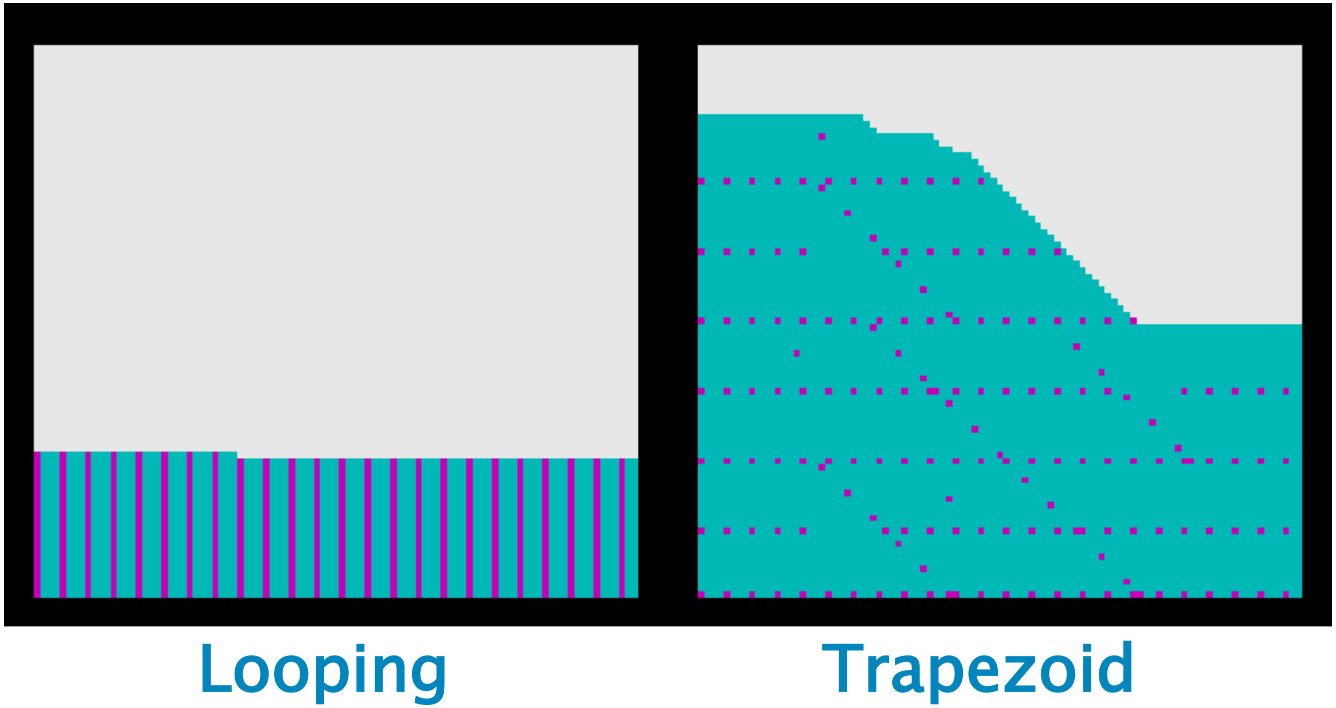
Cache Analysis



- Each leaf represents $\Theta(hw)$ points where $h = \Theta(w)$.
- Each leaf incurs $\Theta(w/\mathcal{B})$ misses where $w = \Theta(\mathcal{M})$.
- $\Theta(NT/hw)$ leaves.
- #internal nodes = #leaves – 1 do not contribute substantially to Q.
- $Q = \Theta(NT/hw)\cdot\Theta(w/\mathcal{B}) = \Theta(NT/\mathcal{M}^2)\cdot\Theta(\mathcal{M}/\mathcal{B}) = \Theta(NT/\mathcal{M}\mathcal{B})$.

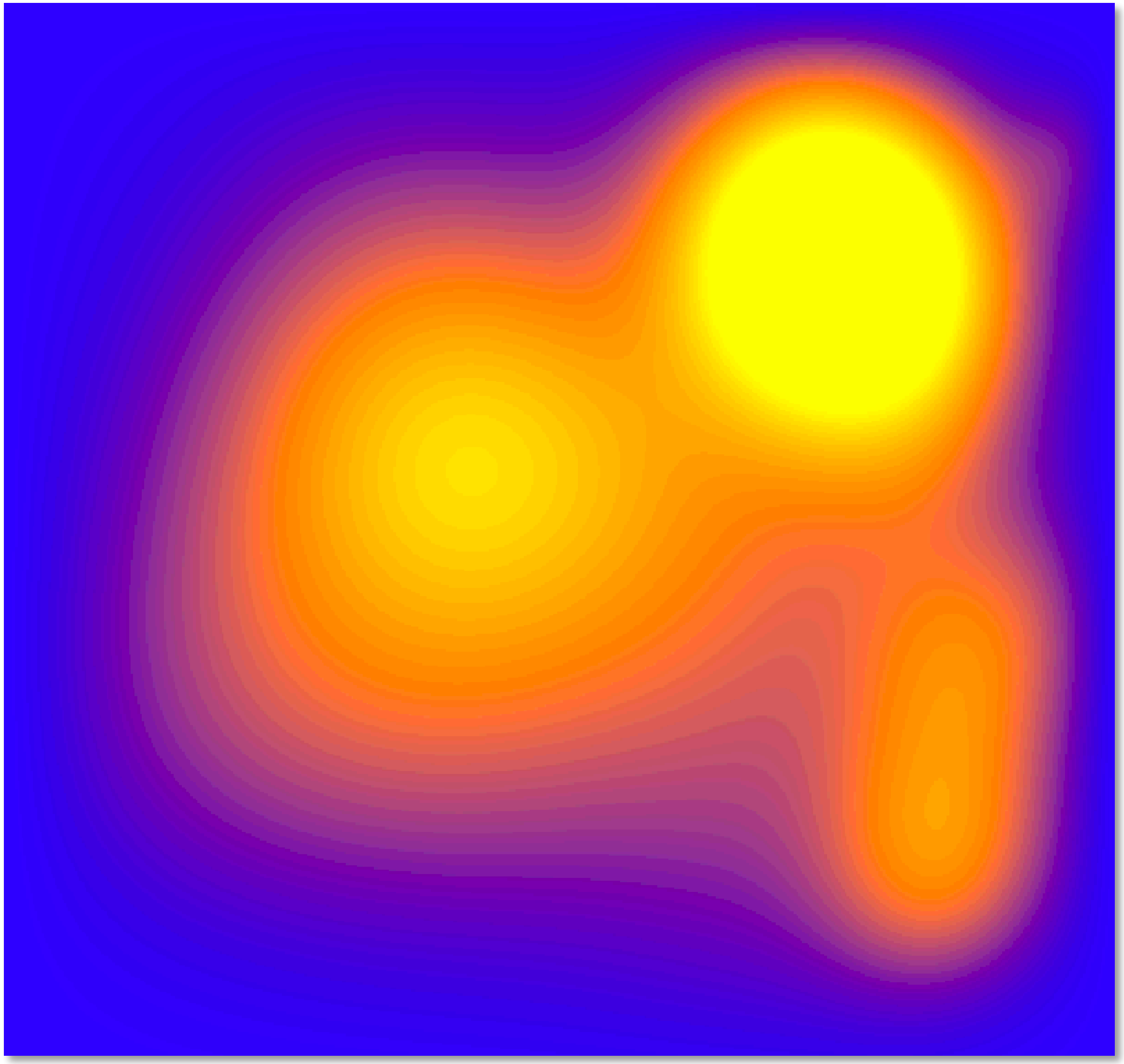
Simulation: 3-Point Stencil

- Rectangular region
  - $N = 95$
  - $T = 87$



- Fully associative LRU cache
  - $\mathcal{B} = 4$ points
  - $\mathcal{M} = 32$
- Cache-hit latency = 1 cycle
- Cache-miss latency = 10 cycles

Looping v. Trapezoid for Real



Other C-O Algorithms

Matrix Transposition/Addition $\Theta(1+mn/\mathcal{B})$

Straightforward recursive algorithm.

Strassen's Algorithm $\Theta(n + n^2/\mathcal{B} + n^{\lg 7}/\mathcal{B}\mathcal{M}^{(\lg 7)/2 - 1})$

Straightforward recursive algorithm.

Fast Fourier Transform $\Theta(1 + (n/\mathcal{B})(1 + \log_{\mathcal{M}} n))$

Variant of Cooley-Tukey [CT65] using cache-oblivious matrix transpose.

LUP-Decomposition $\Theta(1 + n^2/\mathcal{B} + n^3/\mathcal{B}\mathcal{M}^{1/2})$

Recursive algorithm due to Sivan Toledo [T97].

C–O Data Structures

Ordered-File Maintenance $O(1 + (\lg^2 n)/\mathcal{B})$

INSERT/DELETE or delete anywhere in file while maintaining $O(1)$-sized gaps. Amortized bound [BDFC00], later improved in [BCDFC02].

B-Trees

INSERT/DELETE: $O(1+\log_{\mathcal{B}+1} n+(\lg^2 n)/\mathcal{B})$
SEARCH: $O(1+\log_{\mathcal{B}+1} n)$
TRAVERSE: $O(1+k/\mathcal{B})$

Solution [BDFC00] with later simplifications [BDIW02], [BFJ02].

Priority Queues $O(1+(1/\mathcal{B})\log_{\mathcal{M}/\mathcal{B}}(n/\mathcal{B}))$

Funnel-based solution [BF02]. General scheme based on buffer trees [ABDHMM02] supports INSERT/DELETE.