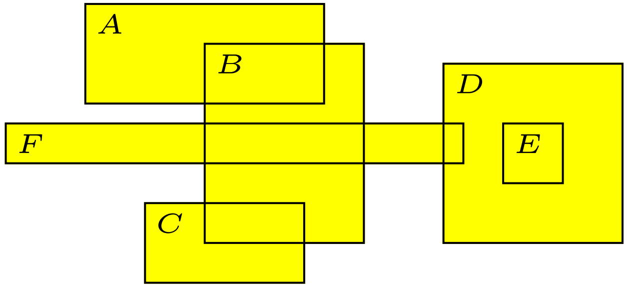
Buffer Trees

Lars Arge. *The Buffer Tree: A New Technique for Optimal I/O Algorithms*. In Proceedings of Fourth Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Computer Science Vol. 955, Springer-Verlag, 1995, 334-345.

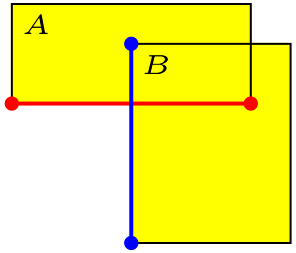
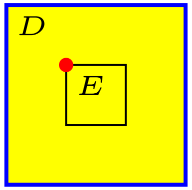
Computational Geometry

Pairwise Rectangle Intersection

Input N rectangles
Output all R pairwise intersections
Example (A, B) (B, C) (B, F) (D, E) (D, F)



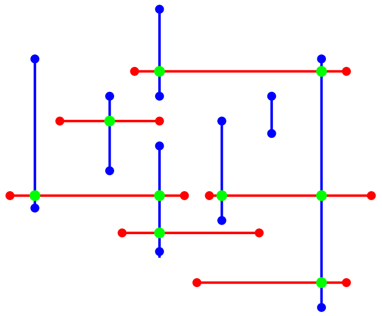
Intersection Types

Intersection	Identified by...
	Orthogonal Line Segment Intersection on $4N$ rectangle sides
	Batched Range Searching on N rectangles and N upper-left corners

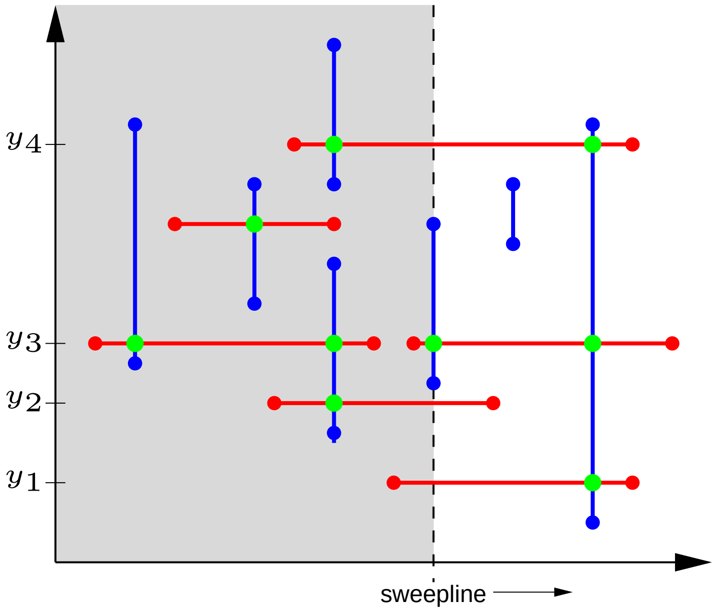
Algorithm Orthogonal Line Segment Intersection
$+$ Batched Range Searching $+$ Duplicate removal

Orthogonal Line Segment Intersection

Input N segments, vertical and horizontal
Output all R intersections



Sweepline Algorithm

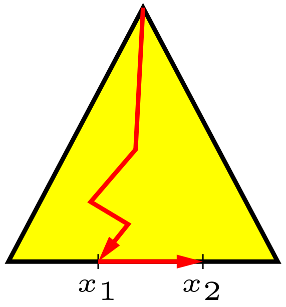


- Sort all endpoints w.r.t. x-coordinate
- Sweep left-to-right with a range tree T storing the y-coordinates of horizontal segments intersecting the sweepline
- Left endpoint $\Rightarrow$ insertion into T
- Right endpoint $\Rightarrow$ deletion from T
- Vertical segment $[y_1, y_2] \Rightarrow$ report $T \cap [y_1, y_2]$

Total (internal) time $O(N \cdot \log_2 N + R)$

Range Trees

Create	Create empty structure
Insert(x)	Insert element x
Delete(x)	Delete the inserted element x
Report(x_1, x_2)	Report all $x \in [x_1, x_2]$



	Binary search trees (internal)	**B-trees** (# I/Os)
Updates	$O(\log_2 N)$	$O(\log_B N)$
Report	$O(\log_2 N + R)$	$O(\log_B N + \frac{R}{B})$

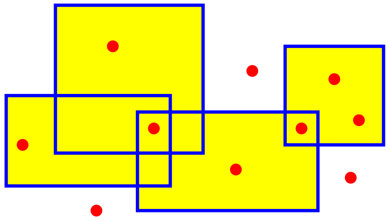
Orthogonal Line Segment Intersection using B-trees

$O(\text{Sort}(N) + N \cdot \log_B N + \frac{R}{B})$ I/Os . . .

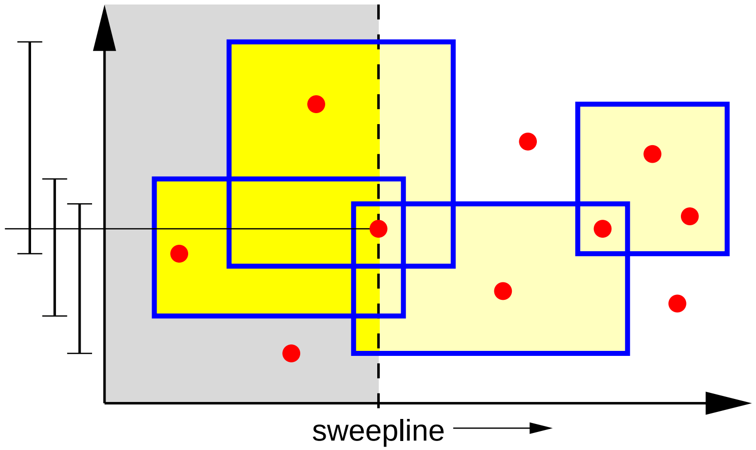
Batched Range Searching

Input N rectangles and points

Output all R (r,p) where point p is within rectangle r



Sweepline Algorithm



- Sort all points and left/right rectangle sides w.r.t. x-coordinate
- Sweep left-to-right while storing the y-intervals of rectangles intersecting the sweepline in a segment tree T
- Left side $\Rightarrow$ insert interval into T
- Right side $\Rightarrow$ delete interval from T
- Point (x,y) $\Rightarrow$ stabbing query : report all $[y_1,y_2]$ where $y \in [y_1,y_2]$

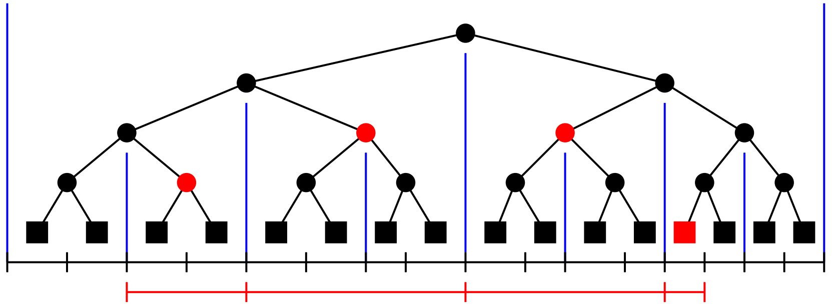
Total (internal) time $O(N \cdot \log_2 N + R)$

Segment Trees

Create	Create empty structure
Insert(x_1, x_2)	Insert segment $[x_1, x_2]$
Delete(x_1, x_2)	Delete the inserted segment $[x_1, x_2]$
Report(x)	Report the segments $[x_1, x_2]$ where $x \in [x_1, x_2]$

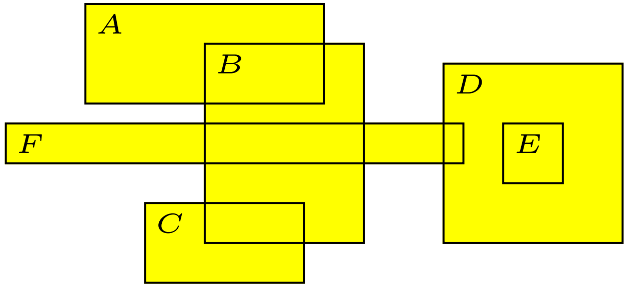
Assumption The endpoints come from a fixed set S of size $N + 1$

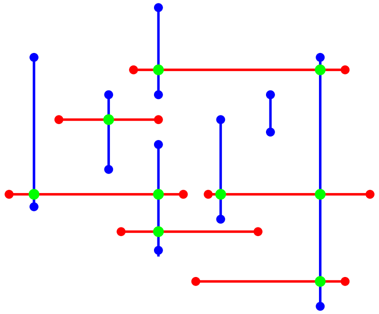
- Construct a balanced binary tree on the N intervals defined by S
- Each node spans an interval and stores a linked list of intervals
- An interval I is stored at the $O(\log N)$ nodes where the node intervals $\subseteq I$ but the intervals of the parents are not

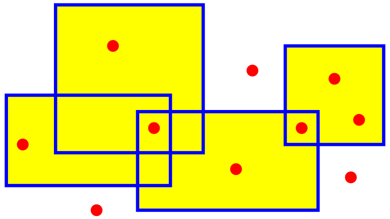


Create	$O(N \log_2 N)$
Insert	$O(\log_2 N)$
Delete	$O(\log_2 N)$
Report	$O(\log_2 N + R)$

Computational Geometry – Summary







Pairwise Rectangle Intersection

Orthogonal Line Segment Intersection

Batched Range Searching

$$\left.\begin{array}{l}\\ \\ \\ \end{array}\right\} O(N \cdot \log_2 N + R)$$

Range Trees

Segment Trees

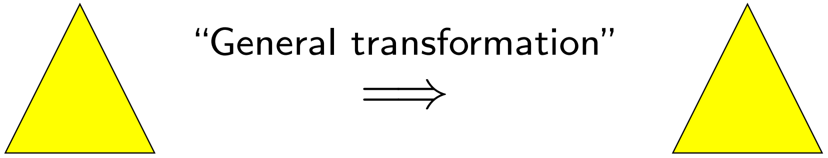
Updates	$O(\log_2 N)$
Queries	$O(\log_2 N + R)$

Observations on Range and Segment Trees

- Only inserted elements are deleted, i.e. Delete does not have to check if the elements are present in the structure
- Applications are off-line, i.e. amortized performance is sufficient
- Queries to the range trees and segment trees can be answered lazily, i.e. postpone processing queries until there are sufficiently many queries to be handled simultaneously
- Output can be generated in arbitrary order, i.e. batched queries
- The deletion time of a segment in a segment tree is known when the segment is inserted, i.e. no explicit delete operation required

Assumptions for buffer trees

Buffer Trees



“General transformation”

$\Longrightarrow$

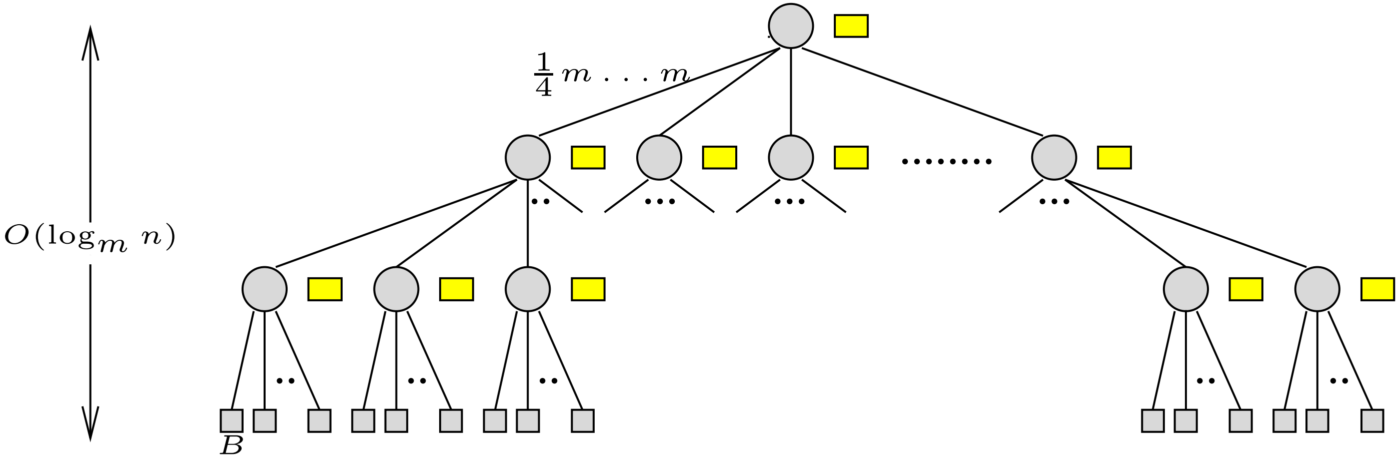
On-line Internal

Batched External

Buffer Trees

- (a, b)-tree, $a = m/4$ and $b = m$
- Buffer at internal nodes m blocks
- Buffers contain delayed operations, e.g. Insert(x) and Delete(x)
- Internal memory buffer containing $\leq B$ last operations

  Moved to root buffer when full
- Invariant Buffers at internal nodes contain $\leq mB/2$ elements



Buffer Emptying : Insertions Only

Emptying internal node buffers

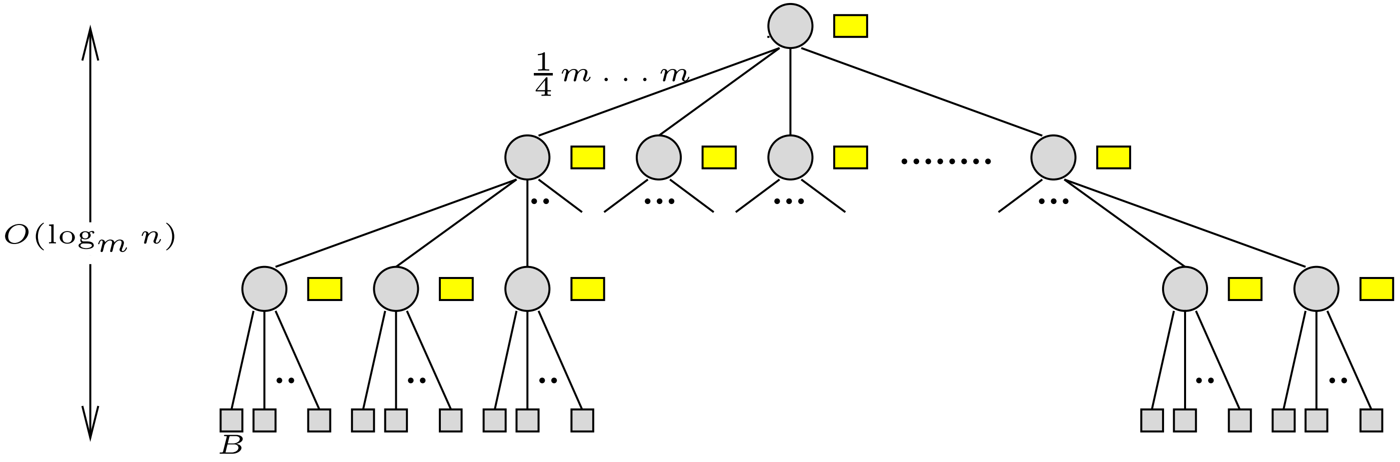
- Distribute $\leq m/2$ blocks of elements to children
- For each child with $m/2$ blocks of elements recursively empty buffer
- If buffer non-empty repeat

Emptying leaf buffers

- Sort buffer
- Merge buffer with leaf blocks
- Rebalance by splitting nodes bottom-up (where buffers are empty)

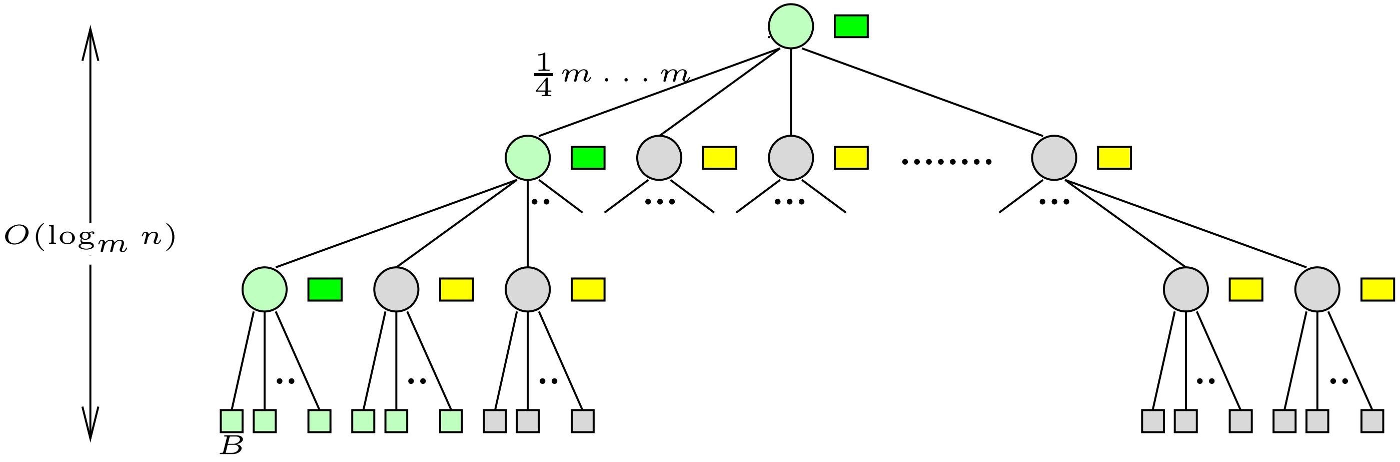
$O(\frac{n}{m})$ buffer empty operations per internal level, each of $O(m)$ I/Os $\Rightarrow$ in total $O(\text{Sort}(N))$ I/Os

Corollary Optimal sorting by top-down emptying all buffers



Priority Queues

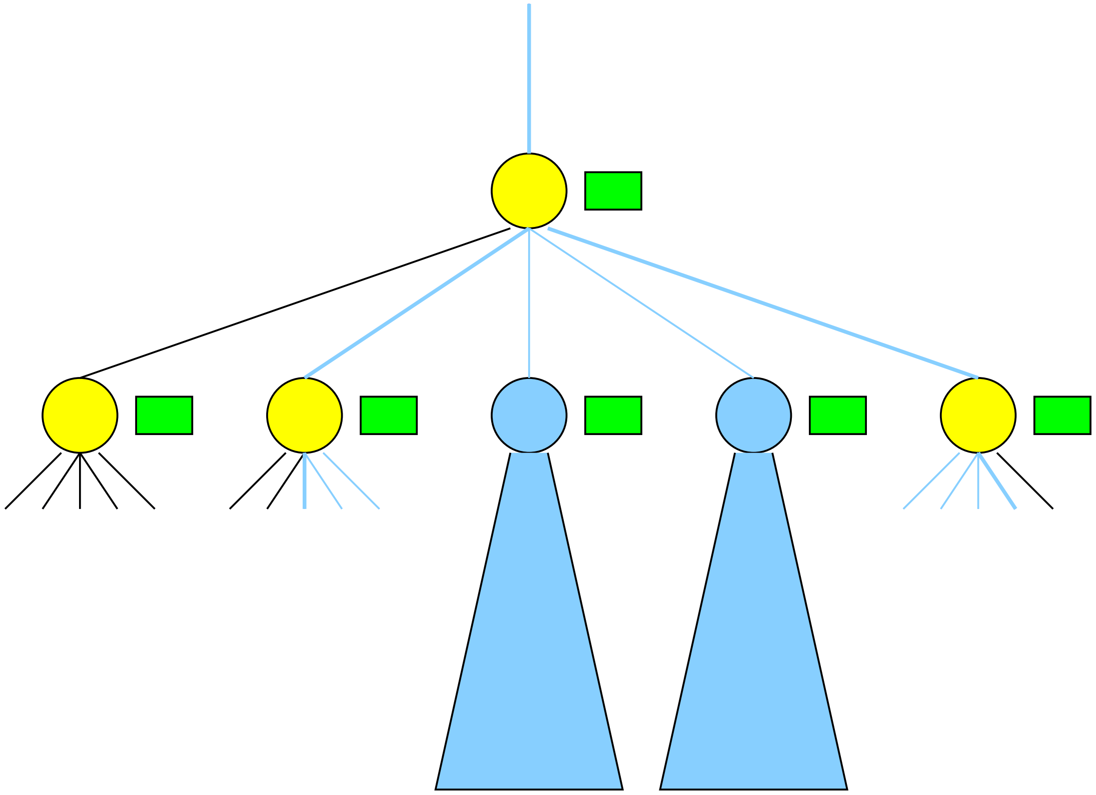
- Operations : Insert(x) and DeleteMin
- Internal memory min-buffer containing the $\frac{1}{4}mB$ smallest elements
- Allow nodes on leftmost path to have degree between 1 and m
  ⇒ rebalancing only requires node splittings
- Buffer emptying on leftmost path
  ⇒ two leftmost leaves contain $\geq mB/4$ elements
- Insert and DeleteMin amortized $O(\frac{1}{B}\log_{M/B}\frac{N}{B})$ I/Os



Batched Range Trees

Delayed operations in buffers : Insert(x), Delete(x), Report(x_1, x_2)

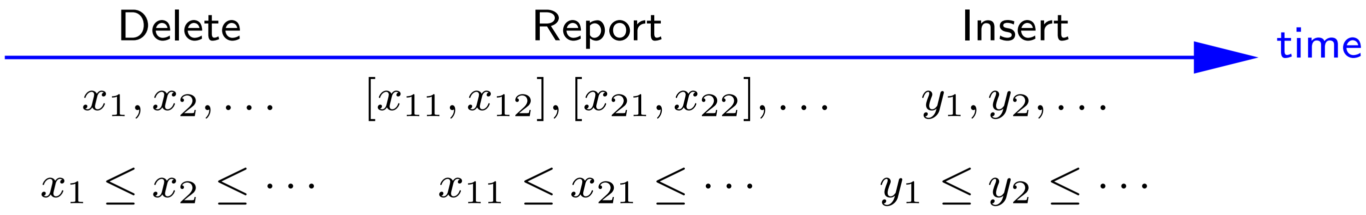
Assumption : Only inserted elements are deleted



Time Order Representation

Definition A buffer is in time order representation (TOR) if

1. Report queries are older than Insert operations and younger than Delete operations
2. Insertions and deletions are in sorted order
3. Report queries are sorted w.r.t. x_1



time

Delete	Report	Insert
$x_1, x_2, \ldots$	$[x_{11}, x_{12}], [x_{21}, x_{22}], \ldots$	$y_1, y_2, \ldots$
$x_1 \leq x_2 \leq \cdots$	$x_{11} \leq x_{21} \leq \cdots$	$y_1 \leq y_2 \leq \cdots$

Constructing Time Order Representations

Lemma A buffer of $O(M)$ elements can be made into TOR using $O(\frac{M+R}{B})$ I/Os where R is the number of matches reported

Proof

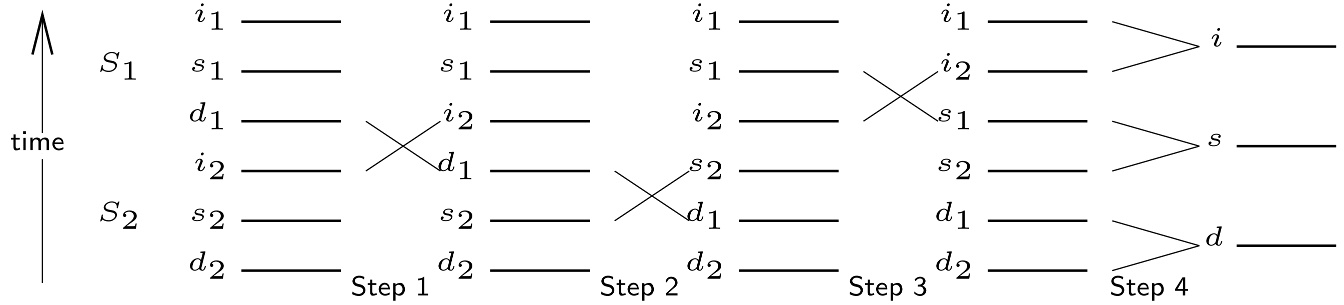
- Load buffer into memory
- First Inserts are shifted up thru time
  - If Insert(x) passes Report(x_1, x_2) and $x \in [x_1, x_2]$ then a match is reported
  - If Insert(x) meets Delete(x), then both operations are removed
- Deletes are shifted down thru time
  - If Delete(x) passes Report(x_1, x_2) and $x \in [x_1, x_2]$ then a match is reported
- Sort Deletions, Reports and Insertion internally
- Output to buffer $\square$

Merging Time Order Representations

Lemma Two list S_1 and S_2 in TOR where the elements in S_2 are older than the elements in S_1 can be merged into one time ordered list in $O(\frac{|S_1|+|S_2|+R}{B})$ I/Os

Proof

1. Swap i_2 and d_1 and remove canceling operations
2. Swap d_1 and s_2 and report matches
3. Swap i_2 and s_1 and report matches
4. Merge lists



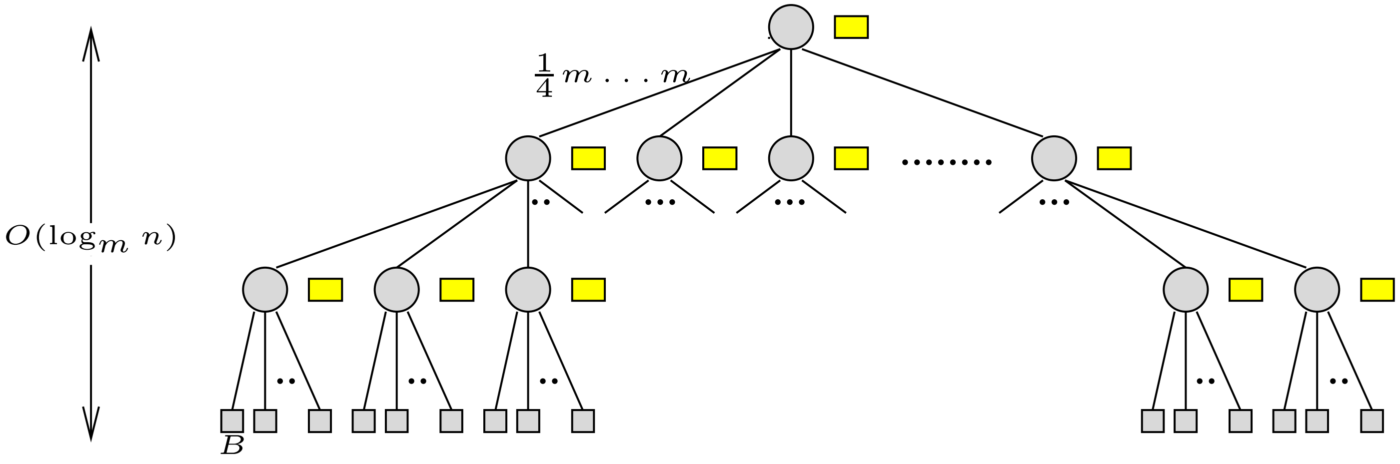
□

Emptying All Buffers

Lemma Emptying all buffers in a tree takes $O(\frac{N+R}{B})$ I/Os

Proof

- Make all buffers into time order representation, $O(\frac{N+R}{B})$ I/Os
- Merge buffers top-down for complete layers ⇒ since layer sizes increase geometrically, #I/Os dominated by size of lowest level, i.e $O(\frac{N+R}{B})$ I/Os



□

Note The tree should be rebalanced afterwards

Emptying Buffer on Overflow

Invariant Emptying a buffer distributes information to children in TOR

1. Load first m blocks in and make TOR and report matches
2. Merge with result from parent in TOR that caused overflow
3. Identify which subtrees are spanned completely by a Report(x_1, x_2)
4. Empty subtrees identified in **??**.
    - Merge with Delete operations
    - Generate output for the range queries spanning the subtrees
    - Merge Insert operations
5. Distribute remaining information to trees not found in **??**.

Batched Range Trees - The Result

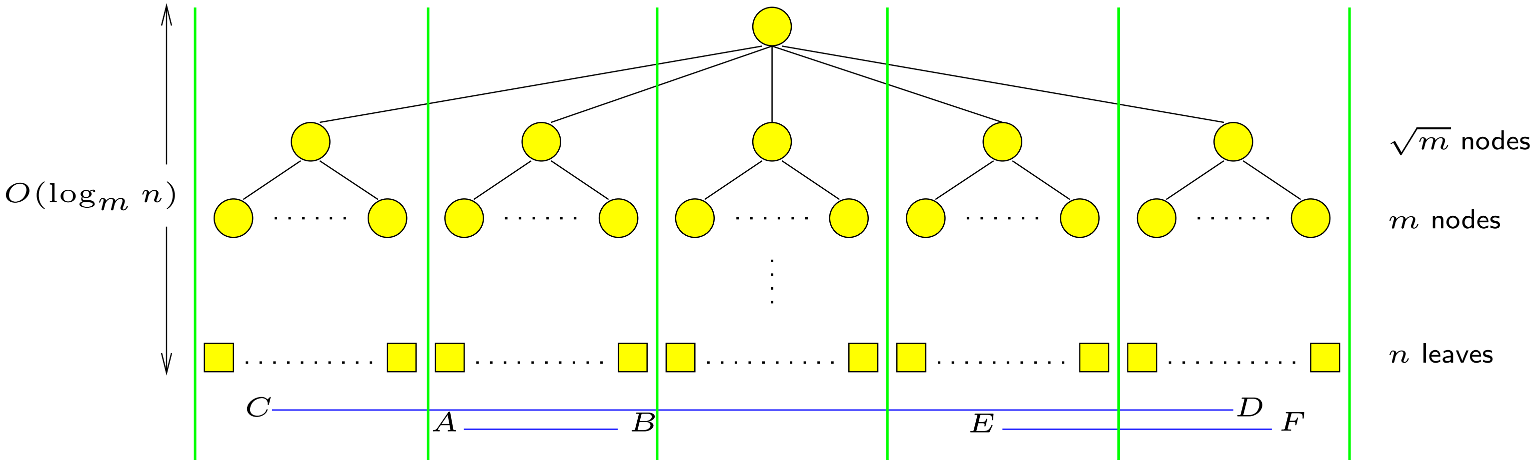
Rebalancing As in (a, b)-trees, except that buffers must be empty. For Fusion and Sharing a forced buffer emptying on the sibling is required, causing $O(m)$ addtional I/Os. Since at most $O(n/m)$ rebalancing steps done $\Rightarrow$ $O(n)$ additional I/Os.

Total #I/Os Bounded by generated output $O(\frac{R}{B})$, and $O(\frac{1}{B})$ I/O for each level an operation is moved down.

Theorem Batched range trees support

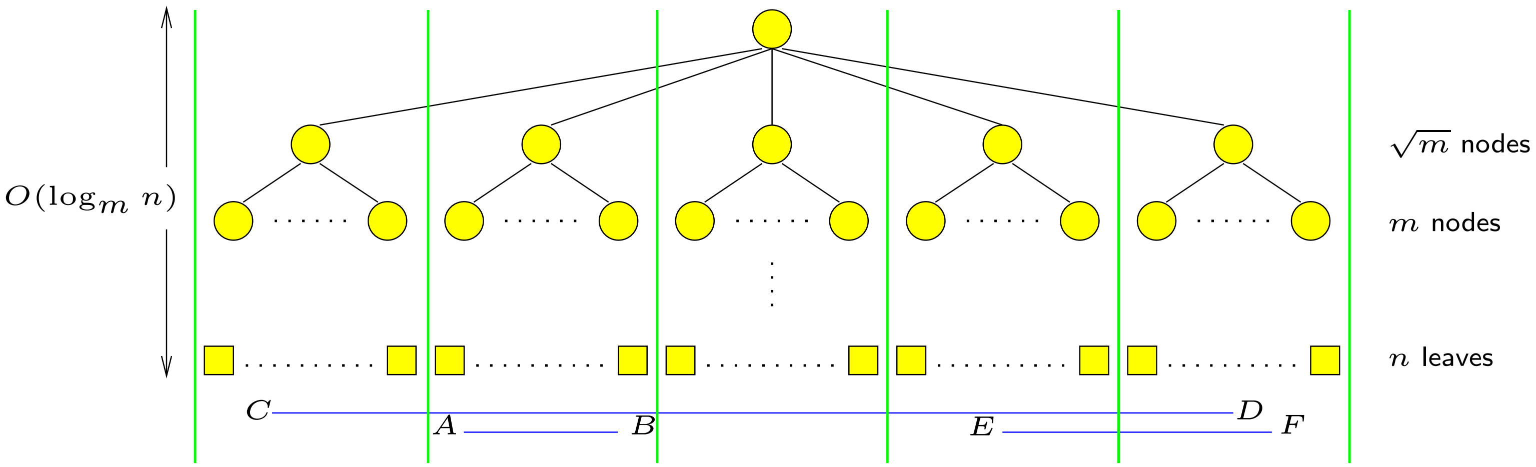
Updates	$O\left(\frac{1}{N}\text{Sort}(N)\right)$ amortized I/Os
Queries	$O\left(\frac{1}{N}\text{Sort}(N) + \frac{R}{B}\right)$ amortized I/Os

Batched Segment Trees



- Internal node:
  - Partition x-interval in $\sqrt{m}$ slabs/intervals
  - $O(m)$ multi-slabs defined by continuous ranges of slabs
  - Segments spanning at least one slab (long segment) stored in list associated with largest multi-slab it spans
  - Short segments, as well as ends of long segments, are stored further down the tree

Batched Segment Trees



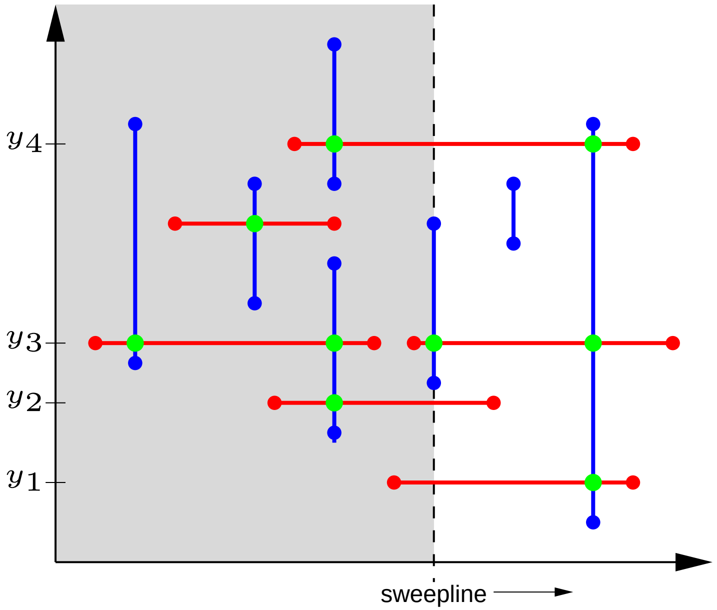
- Buffer-emptying process in $O(m + \frac{R}{B})$ I/Os:
  - Load buffer — $O(m)$
  - Store long segments from buffer in multi-slab lists — $O(m)$
  - Report “intersections” between queries from buffer and segments in relevant multi-slab lists — $O(\frac{R}{B})$
  - “Push” elements one level down — $O(m)$

Batched Segment Trees

Theorem Batched segment trees support

Updates	$O(\frac{1}{N}\text{Sort}(N))$ amortized I/Os
Queries	$O(\frac{1}{N}\text{Sort}(N) + \frac{R}{B})$ amortized I/Os

Orthogonal Line Segment Intersection



- Sort all endpoints w.r.t. x-coordinate — Sort(N)
- Sweep left-to-right with a batched range tree T — $O(\frac{N}{B})$
- Left endpoint ⇒ insertion into T
- Right endpoint ⇒ deletion from T

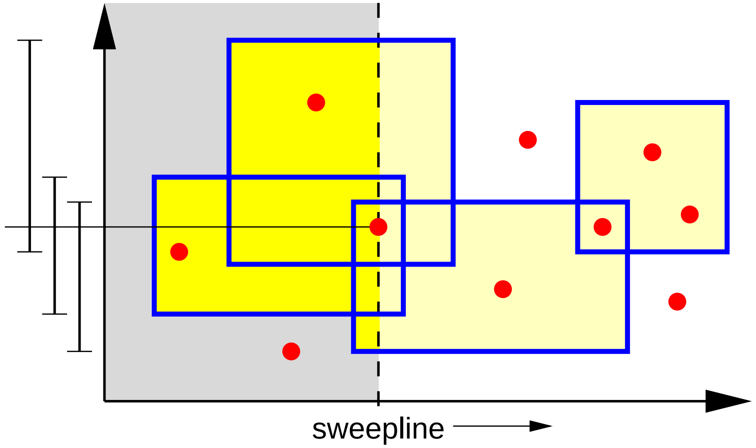
$$\Big\} \quad O(\tfrac{1}{B}\log_{M/B}\tfrac{N}{B})$$

- Vertical segment ⇒ batched report — $O(\frac{1}{B}\log_{M/B}\frac{N}{B}+\frac{R}{B})$

$$\overline{}$$

$O(\text{Sort}(N)+\frac{R}{B})$ I/Os

Batched Range Searching



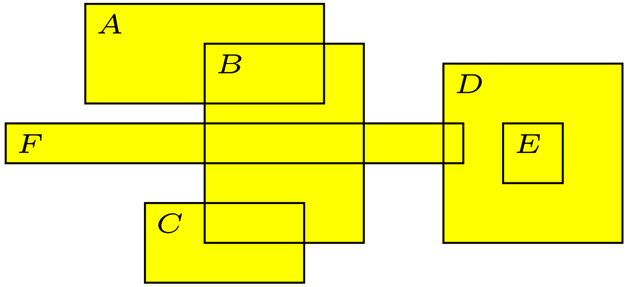
- Sort w.r.t. x-coordinate — Sort(N)
- Sweep left-to-right with a batched segment tree T — $O(\frac{N}{B})$
- Left side $\Rightarrow$ insert interval into T
- Right side $\Rightarrow$ delete interval from T

$$\Big\}\quad O(\tfrac{1}{B}\log_{M/B}\tfrac{N}{B})$$

- Point $\Rightarrow$ batched stabbing query — $O(\frac{1}{B}\log_{M/B}\frac{N}{B}+\frac{R}{B})$

$O(\text{Sort}(N)+\frac{R}{B})$ I/Os

Pairwise Rectangle Intersection

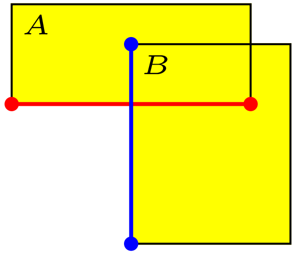


Orthogonal line segment intersection

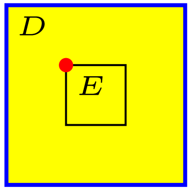
Batched range searching

Duplicate removal

$=$



$+$



$+$?

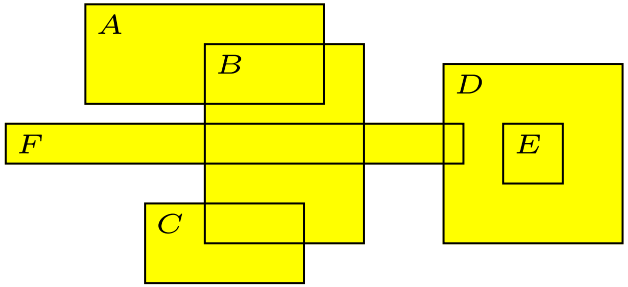
$4N$ rectangle sides

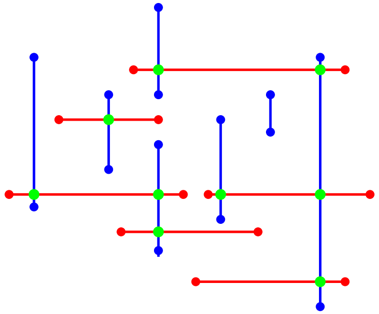
N rectangles and
N upper-left corners

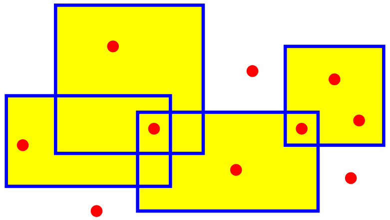
Trick Only generate one intersection between two rectangles

$\Rightarrow O(\text{Sort}(N) + \frac{R}{B})$ I/Os

Buffer Tree Applications – Summary







Pairwise Rectangle Intersection

Orthogonal Line Segment Intersection

Batched Range Searching

$$\left.\right\} \quad O(\text{Sort}(N) + \frac{R}{B})$$

Batched Range Trees

Batched Segment Trees

Updates	$O(\frac{1}{N}\text{Sort}(N))$
Queries	$O(\frac{1}{N}\text{Sort}(N) + \frac{R}{B})$

Priority Queues $O(\frac{1}{N}\text{Sort}(N))$