Binary Search Trees

Key set $K = \{3, 7, 12, 18, 25, 29, 37, 43, 51, 55, 61, 71\}$

Size $n = |K|$

Binary Search Trees

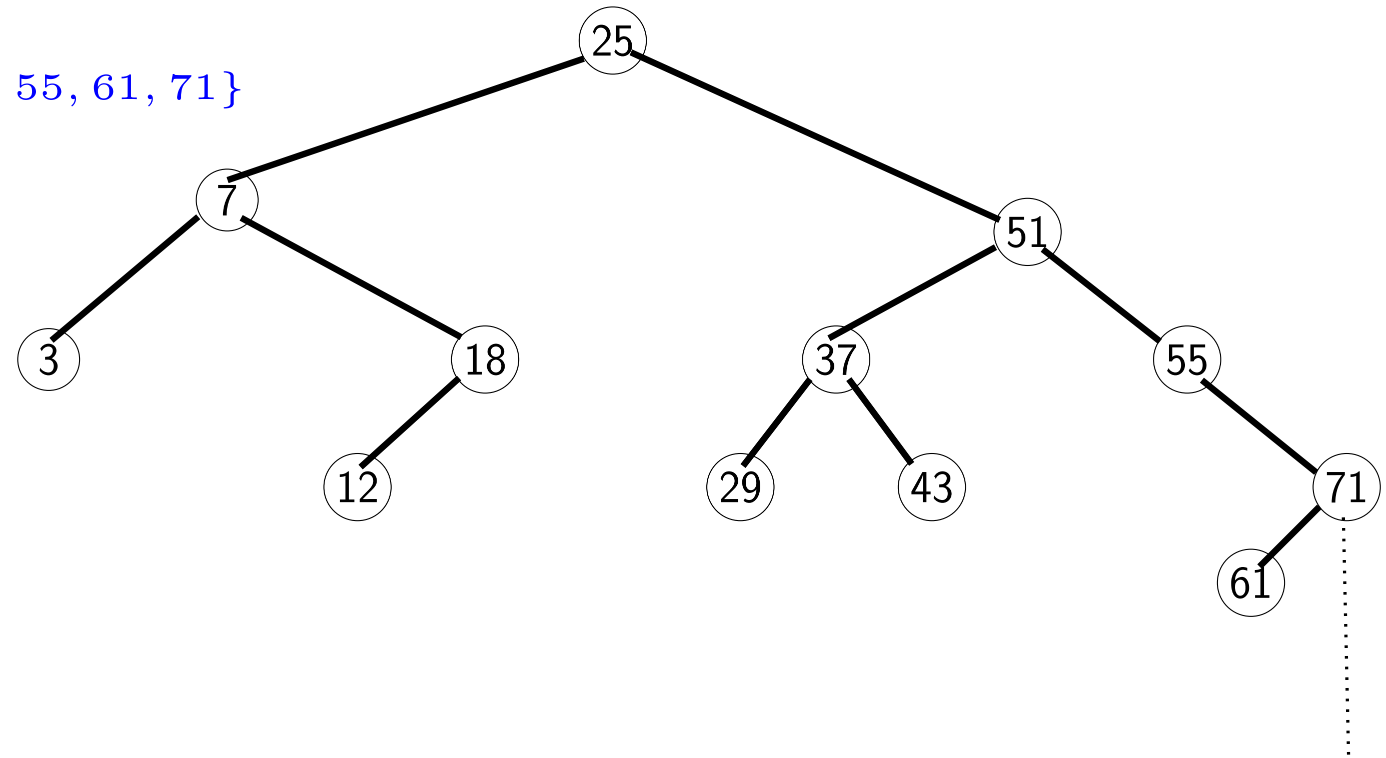
Key set $K = \{3, 7, 12, 18, 25, 29, 37, 43, 51, 55, 61, 71\}$

Size $n = |K|$

T **Binary Search Tree** for K

v node of T: v.LC, v.RC, v.PAR, v.key

- T_v subtree rooted at v
- K_v keys in T_v



Binary Search Trees

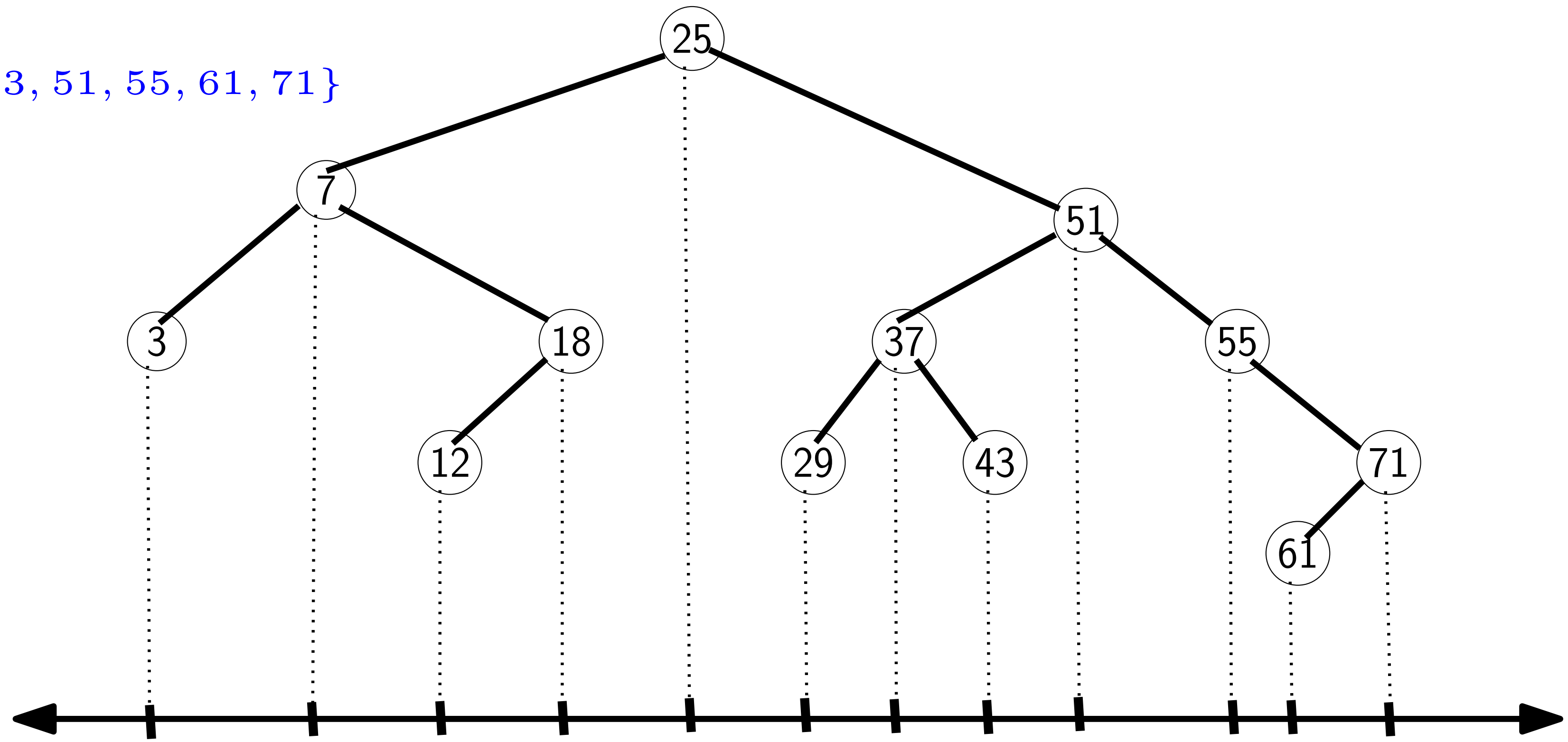
Key set $K = \{3, 7, 12, 18, 25, 29, 37, 43, 51, 55, 61, 71\}$

Size $n = |K|$

T **Binary Search Tree** for K

v node of T: v.LC, v.RC, v.PAR, v.key

- T_v subtree rooted at v
- K_v keys in T_v



Binary Search Trees

Key set $K = \{3, 7, 12, 18, 25, 29, 37, 43, 51, 55, 61, 71\}$

Size $n = |K|$

T **Binary Search Tree** for K

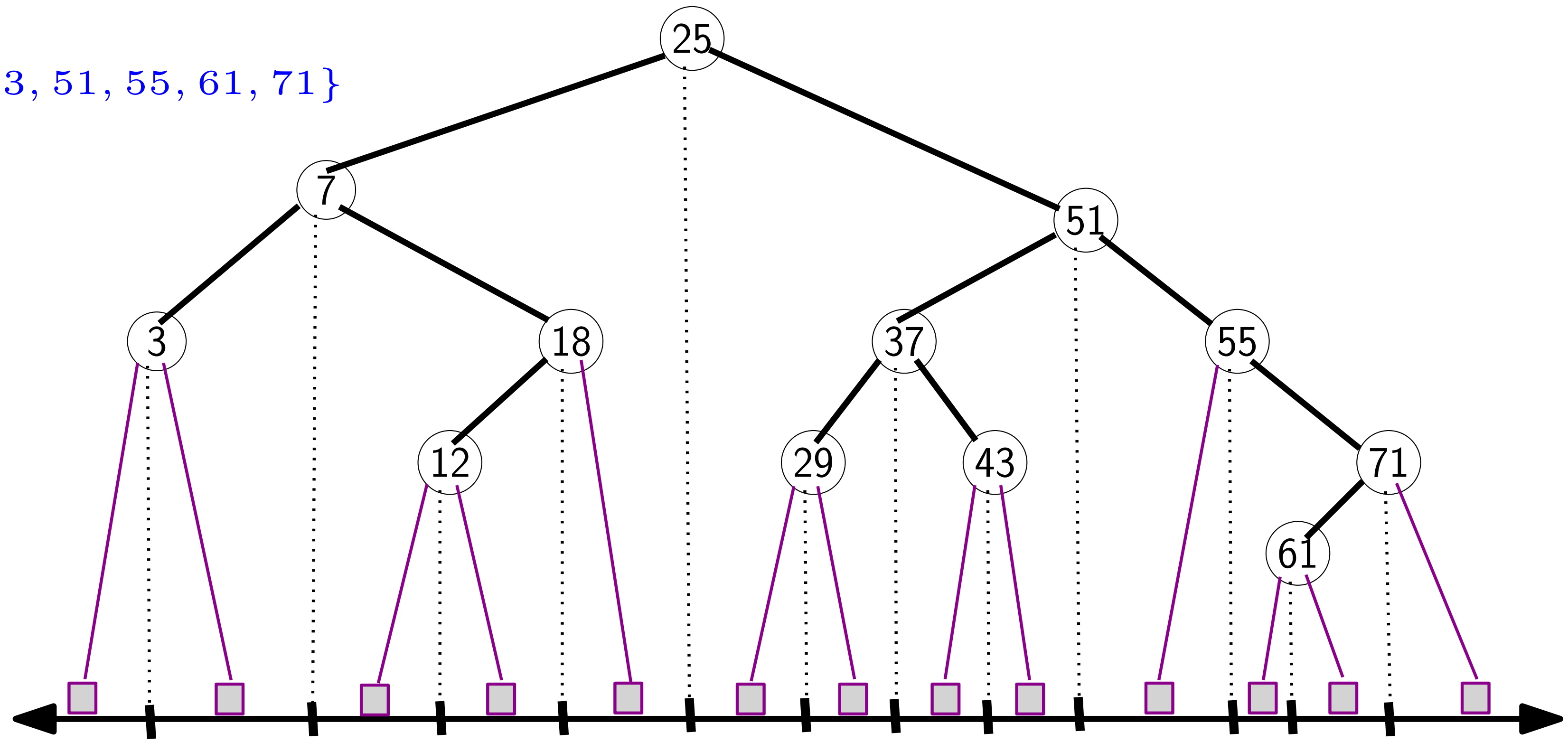
v node of T: v.LC, v.RC, v.PAR, v.key

T_v subtree rooted at v

K_v keys in T_v

Additional leaf for each primitive interval

$\overline{T}$ **Extended Binary Search Tree** for K



Binary Search Trees

Key set $K = \{3, 7, 12, 18, 25, 29, 37, 43, 51, 55, 61, 71\}$

Size $n = |K|$

T **Binary Search Tree** for K

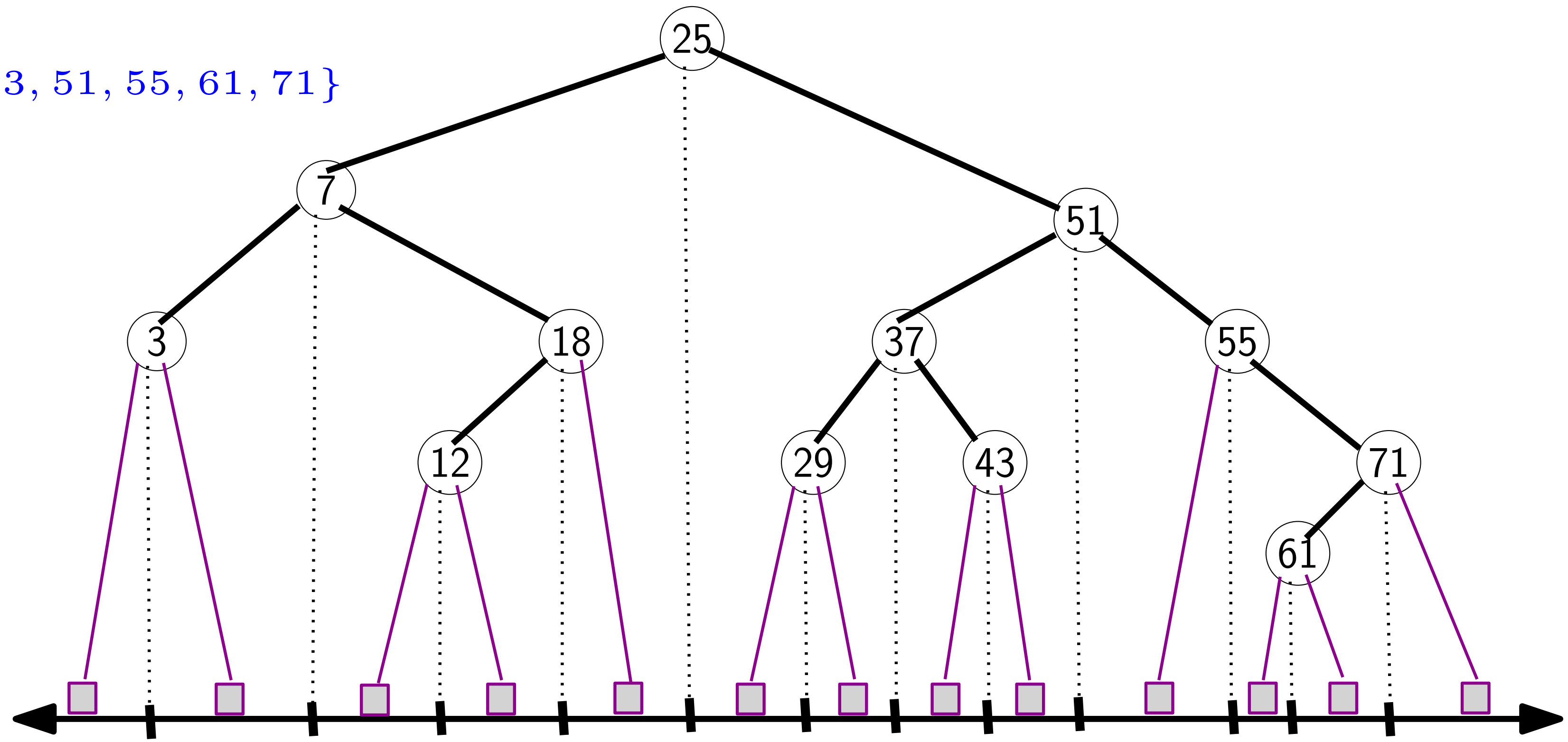
v node of T: v.LC, v.RC, v.PAR, v.key

T_v subtree rooted at v

K_v keys in T_v

Additional leaf for each primitive interval

$\overline{T}$ **Extended Binary Search Tree** for K



I_v is the union of the primitive intervals associated with the leaves of $\overline{T_v}$ together with K_v

Binary Search Trees

Key set $K = \{3, 7, 12, 18, 25, 29, 37, 43, 51, 55, 61, 71\}$

Size $n = |K|$

T **Binary Search Tree** for K

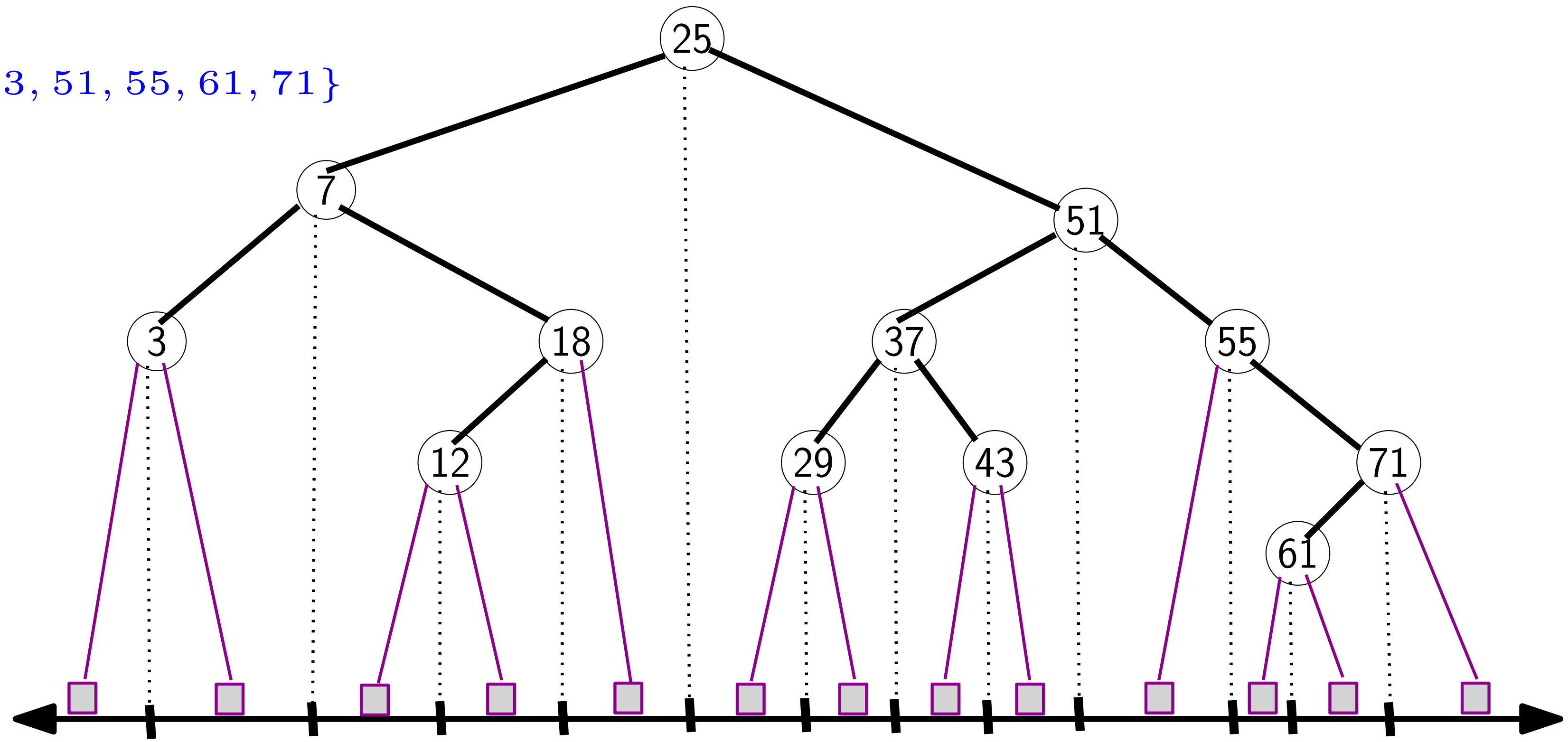
v node of T: v.LC, v.RC, v.PAR, v.key

T_v subtree rooted at v

K_v keys in T_v

Additional leaf for each primitive interval

$\overline{T}$ **Extended Binary Search Tree** for K



I_v is the union of the primitive intervals associated with the leaves of $\overline{T_v}$ together with K_v

key $x \in \mathbb{R}$:

$\mathrm{path}(x) = \{v \in \overline{T} | x \in I_v\}$

Binary Search Trees

Key set $K = \{3, 7, 12, 18, 25, 29, 37, 43, 51, 55, 61, 71\}$

Size $n = |K|$

T **Binary Search Tree** for K

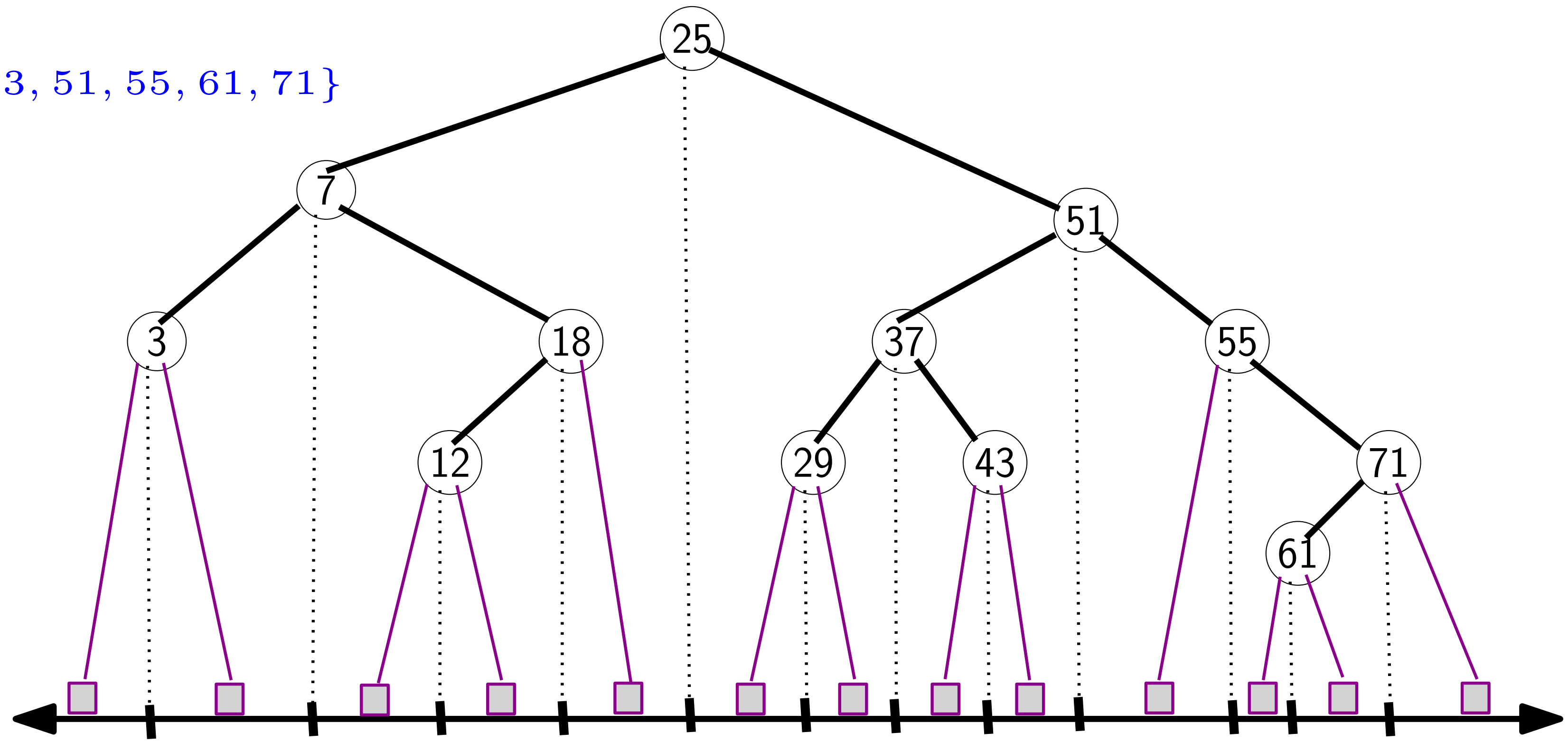
v node of T: v.LC, v.RC, v.PAR, v.key

T_v subtree rooted at v

K_v keys in T_v

Additional leaf for each primitive interval

$\overline{T}$ **Extended Binary Search Tree** for K



I_v is the union of the primitive intervals associated with the leaves of $\overline{T_v}$ together with K_v

key $x \in \mathbb{R}$:

$$\mathrm{path}(x) = \{v \in \overline{T} | x \in I_v\}$$

interval $[\alpha, \beta]$ with $\alpha, \beta \in K$:

$$\mathrm{span}[\alpha, \beta] = \{v \in \overline{T} | I_v \subseteq [\alpha, \beta] \text{ but } I_{v.\mathrm{PAR}} \not\subseteq [\alpha, \beta]\}$$

Binary Search Trees

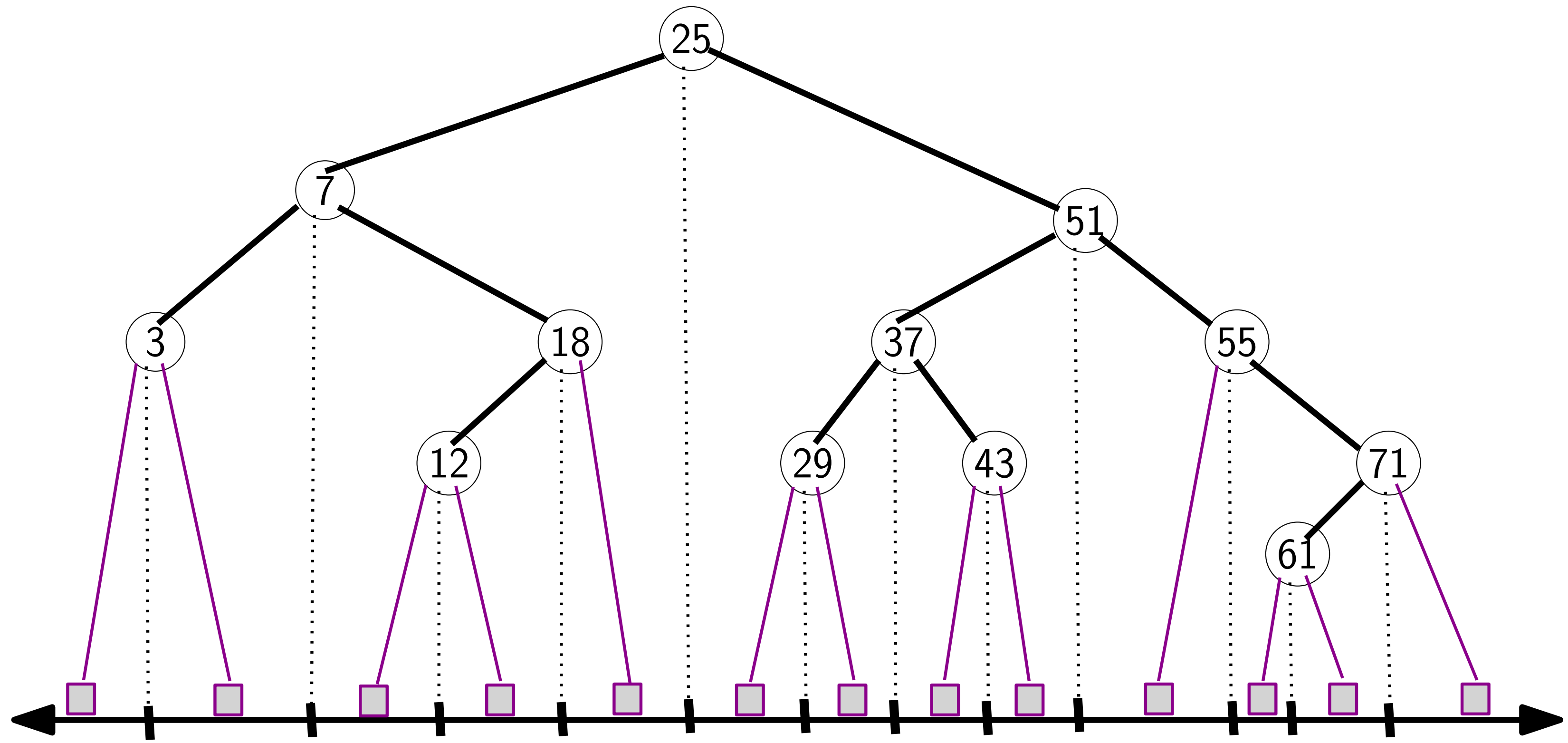
Size $n = |K|$

v node of T:

T_v subtree rooted at v

K_v keys in T_v

I_v is the union of the primitive intervals associated with the leaves of $\overline{T_v}$ together with K_v



key $x \in \mathbb{R}$:

$$\text{path}(x) = \{v \in \overline{T} | x \in I_v\}$$

interval $[\alpha, \beta]$ with $\alpha, \beta \in K$:

$$\text{span}[\alpha, \beta] = \{v \in \overline{T} | I_v \subseteq [\alpha, \beta] \text{ but } I_{v.\text{PAR}} \not\subseteq [\alpha, \beta]\}$$

Lemma: T binary tree for n keys with height $O(\log n)$.

- for any key x we have $|\text{path}(x)| = O(\log n)$
- for any interval $[\alpha, \beta]$ we have $|\text{span}[\alpha, \beta]| = O(\log n)$
- If $\alpha, \beta \in K$ then $[\alpha, \beta] = \dot{\bigcup}\{I_v | v \in \text{span}[\alpha, \beta]\}$.
- $\text{path}(x)$ and $\text{span}[\alpha, \beta]$ can be found in $O(\log n)$ time.

Range Trees

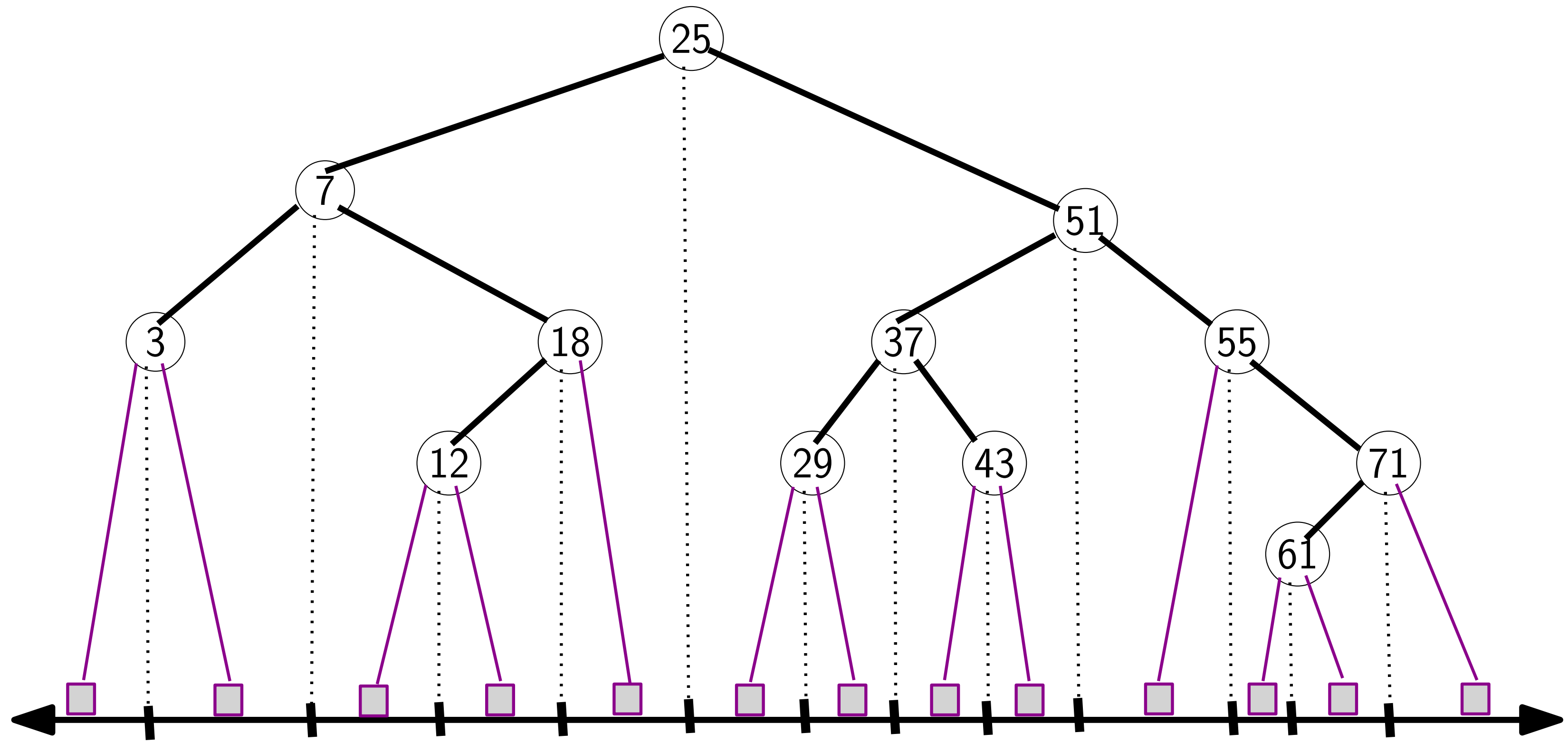
Size $n = |K|$

v node of T:

T_v subtree rooted at v

K_v keys in T_v

I_v is the union of the primitive intervals associated with the leaves of $\overline{T_v}$ together with K_v

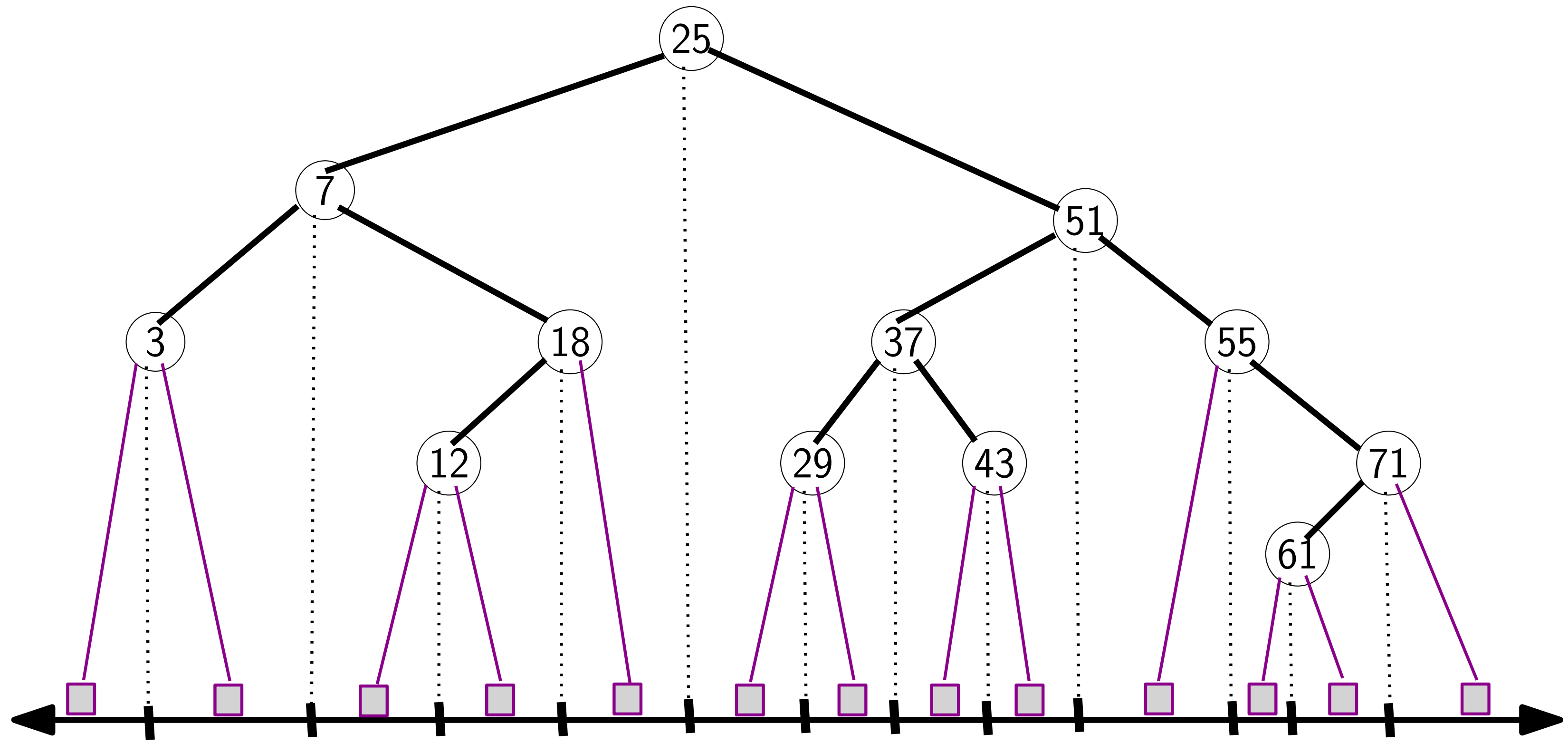


A set of objects, each $a \in A$ has a value $a.\mathrm{key}$ associated with it.

A **range tree** for A is a balanced binary search tree T whose key set K contains $\{a.\mathrm{key} | a \in A\}$ and that stores for each node v of T the set $A_v = \{a \in A | a.\mathrm{key} \in K_v\}$.

Saarland Informatics Campus

Range Trees



Size $n = |K|$

v node of T:

T_v subtree rooted at v

K_v keys in T_v

I_v is the union of the primitive intervals associated with the leaves of $\overline{T_v}$ together with K_v

A set of objects, each $a \in A$ has a value $a.\mathrm{key}$ associated with it.

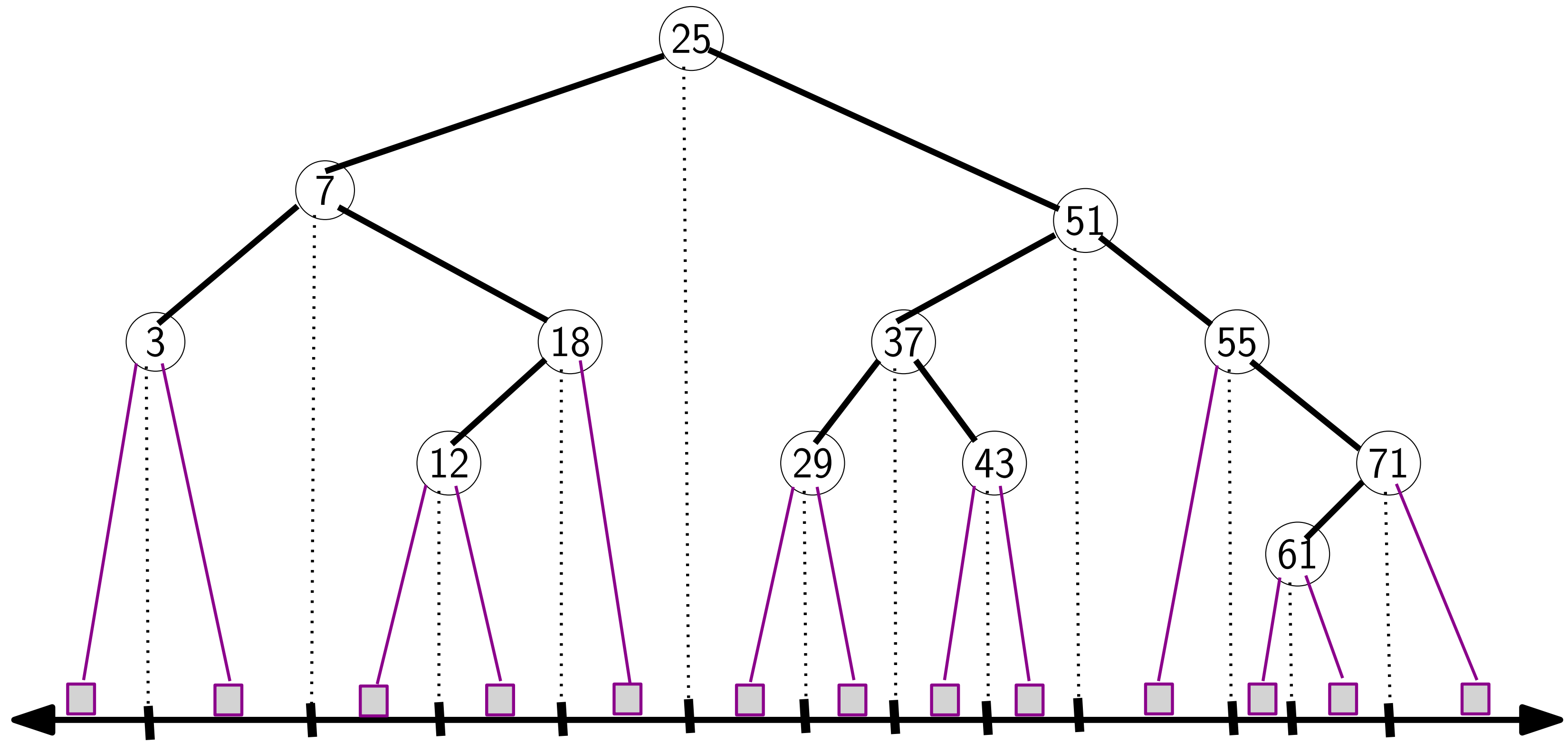
A **range tree** for A is a balanced binary search tree T whose key set K contains $\{a.\mathrm{key} | a \in A\}$ and that stores for each node v of T the set $A_v = \{a \in A | a.\mathrm{key} \in K_v\}$.

Lemma: Let A be a set of objects with keys in K, and $n = |K|$. Let T be a range tree for A with key set K

- $\sum_{v \in T} |A_v| = O(|A| \log n)$
- Given interval $[\alpha, \beta]$ the set $\{a \in A | a.\mathrm{key} \in [\alpha, \beta]\}$ can be found as a disjoint union of $O(\log n)$ blocks in $O(\log n)$ time.
- If $|A| = O(n)$ and the A_v's are stored in data structures that admit updates in time $O(\log^k n)$ then the range tree can be updated in time $O(\log^{k+1} n)$.

SIC Saarland Informatics Campus

Segment Trees



Size $n = |K|$

v node of T:

T_v subtree rooted at v

K_v keys in T_v

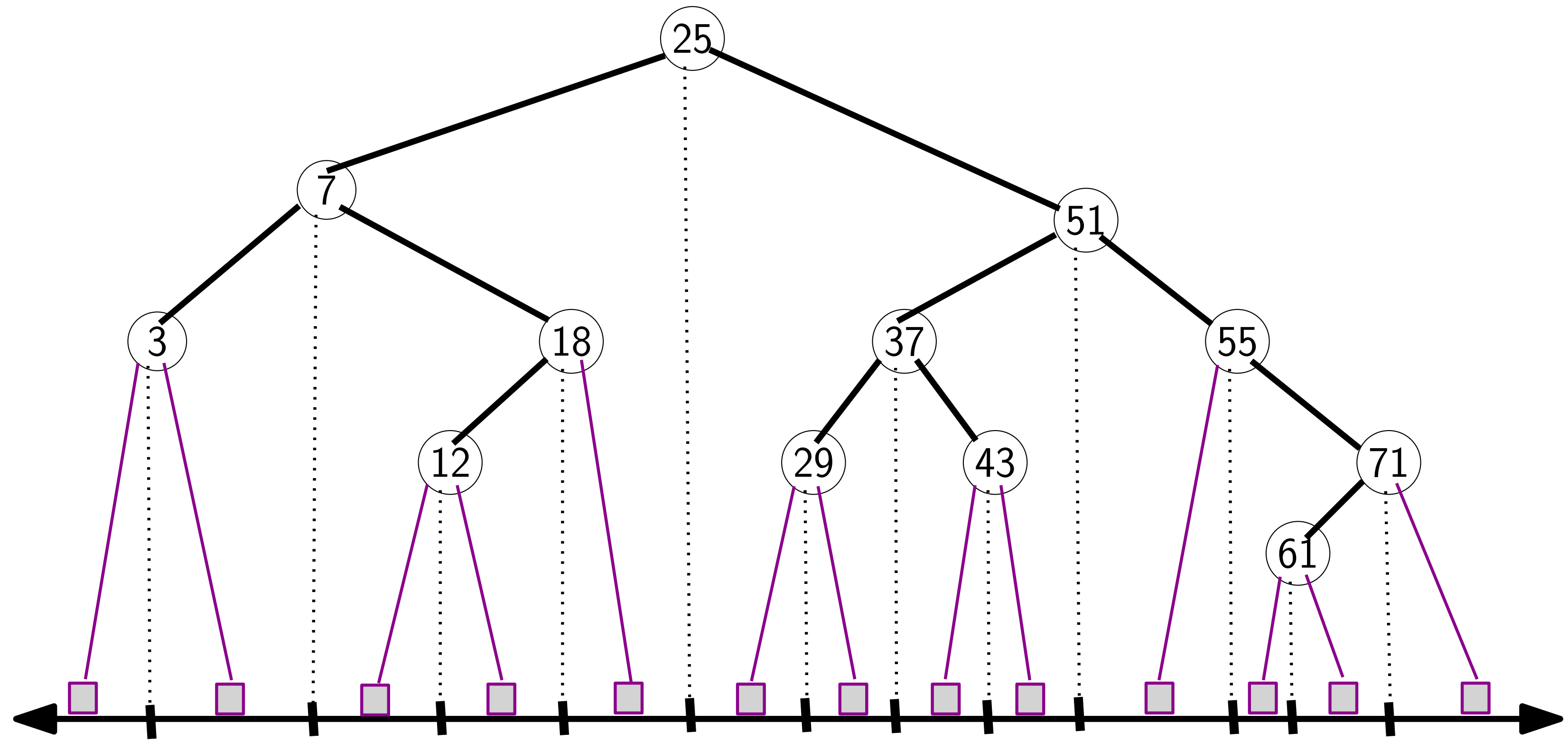
I_v is the union of the primitive intervals associated with the leaves of $\overline{T_v}$ together with K_v

A set of objects, each $a \in A$ has a segment $a.\mathrm{seg}$ associated with it.

A **segment tree** for A is a balanced binary search tree T whose key set K contains all endpoints of segments $\{a.\mathrm{seg} | a \in A\}$ and that stores for each node v of T the set $S_v = \{a \in A | v \in \mathrm{span}(a.\mathrm{seg})\}$.

SIC Saarland Informatics Campus

Segment Trees



Size $n = |K|$

v node of T:

T_v subtree rooted at v

K_v keys in T_v

I_v is the union of the primitive intervals associated with the leaves of $\overline{T_v}$ together with K_v

A set of objects, each $a \in A$ has a segment $a.\mathrm{seg}$ associated with it.

A **segment tree** for A is a balanced binary search tree T whose key set K contains all endpoints of segments $\{a.\mathrm{seg} | a \in A\}$ and that stores for each node v of T the set $S_v = \{a \in A | v \in \mathrm{span}(a.\mathrm{seg})\}$.

Lemma: Let A be a set of objects each associated with a segment with endpoints in K. Let $n = |K|$ and let T be a segment tree for A with key set K

- $\sum_{v \in T} |S_v| = O(|A| \log n)$
- Given key $x \in \mathbb{R}$ the set $\{a \in A | x \in a.\mathrm{seg}\}$ can be found as a disjoint union of $O(\log n)$ blocks in $O(\log n)$ time.
- If $|A| = O(n)$ and the S_v's are stored in data structures that admit updates in time $O(\log^k n)$ then the segment tree can be updated in time $O(\log^{k+1} n)$.

Hierarchies of Range and Segment Trees

Example 1:

A a set of n objects each having an xkey and ykey.
Build a data structure for A so that for any axis-parallel rectangle $B = \mathrm{xseg} \times \mathrm{yseg}$ you can tell quickly for which objects in A you have $(a.\mathrm{xkey}, a.\mathrm{ykey}) \in B$.

Example 2:

A a set of n objects each having an xseg and yseg, defining the axis-parallel rectangle $a.Box = \mathrm{xseg} \times \mathrm{yseg}$.
Build a data structure for A so that for any query point $q \in \mathbb{R}^2$ you can determine quickly for which objects in A you have $q \in a.Box$.

Example 3:

A a set of n horizontal segments $a.\mathrm{xseg}$.

Build a data structure for A so that for any vertical query segment s you can determine quickly the segments in A that intersect q.

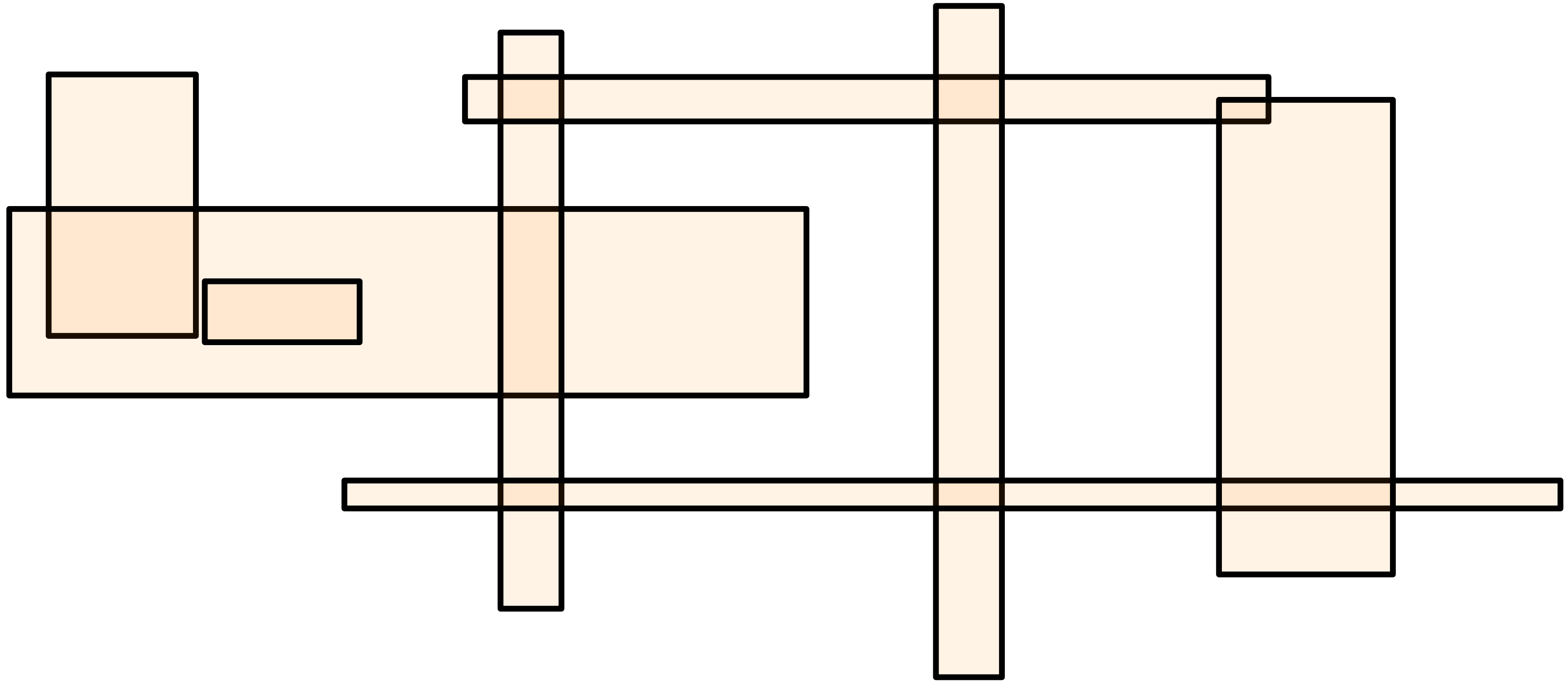
Example 3:

A a set of n horizontal segments $a.\mathrm{xseg}$.

Build a data structure for A so that for any vertical query segment s you can determine quickly the segments in A that intersect q.

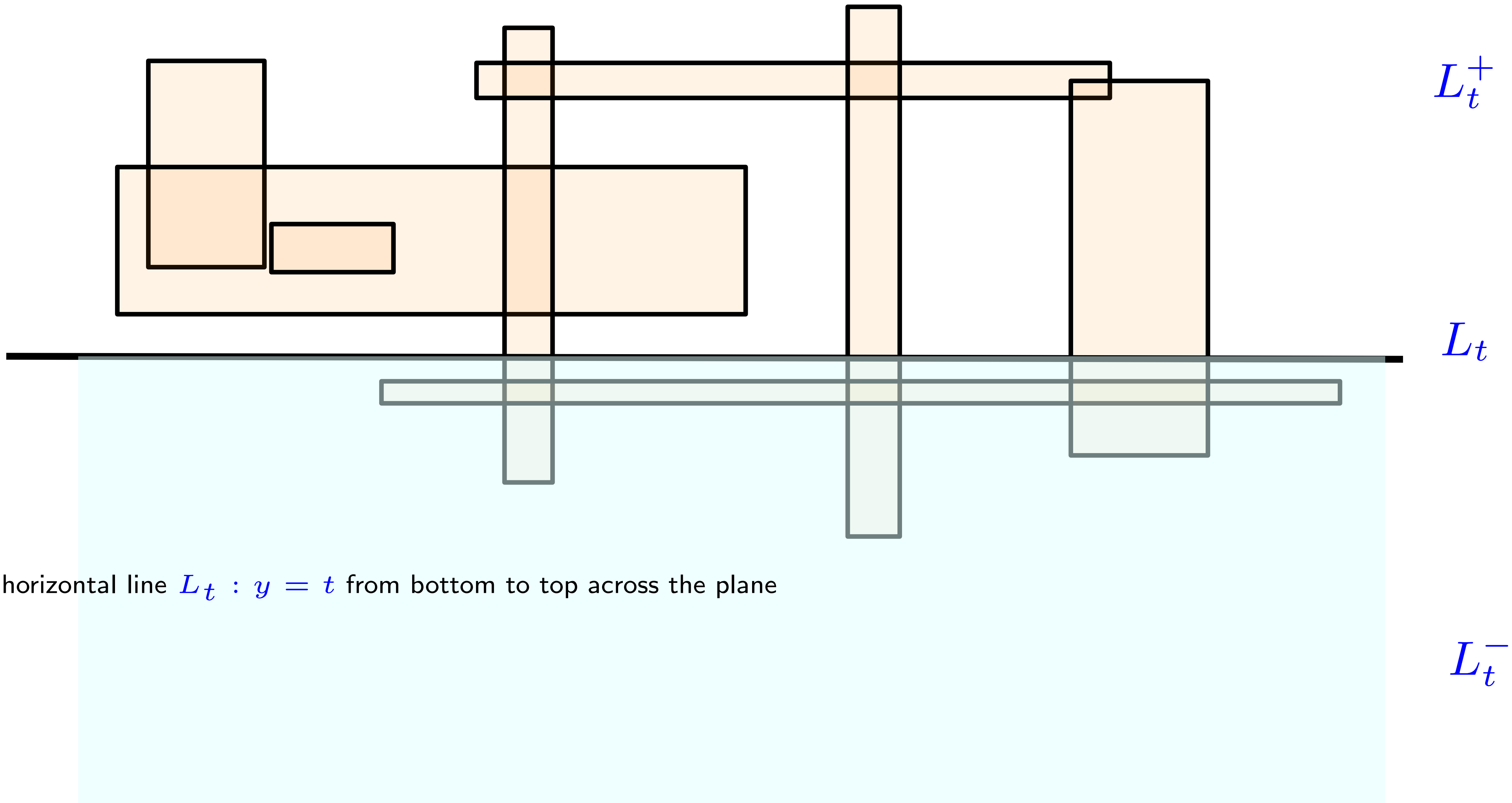
Sweep Algorithms

Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.



Sweep Algorithms

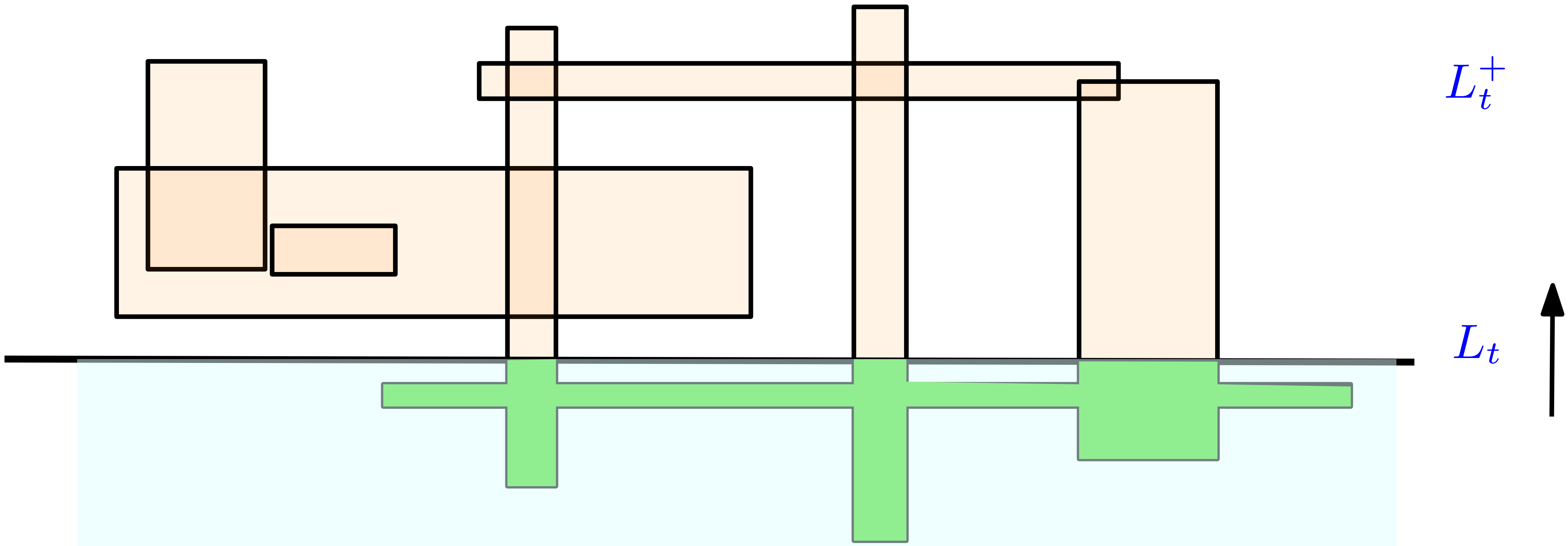
Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.



Sweep horizontal line $L_t : y = t$ from bottom to top across the plane

Sweep Algorithms

Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.

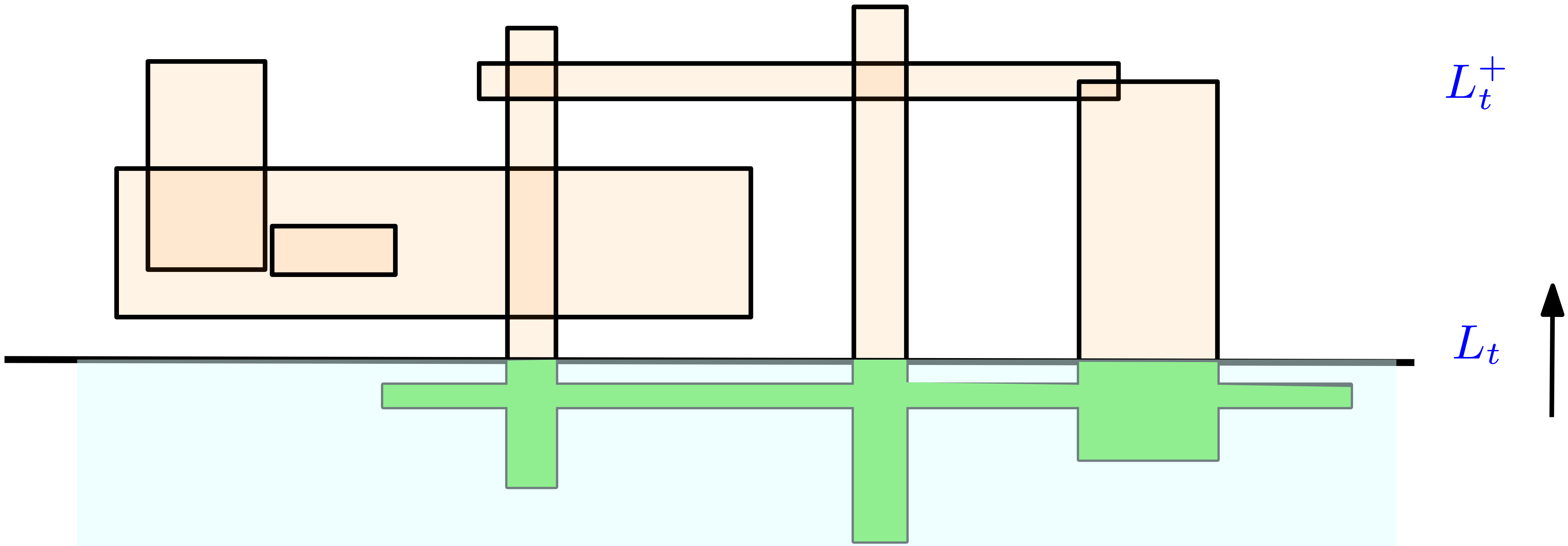


Sweep horizontal line $L_t : y = t$ from bottom to top across the plane

and maintain an Invariant so that in the end the veracity of the invariant implies correctness of the computation

Sweep Algorithms

Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.



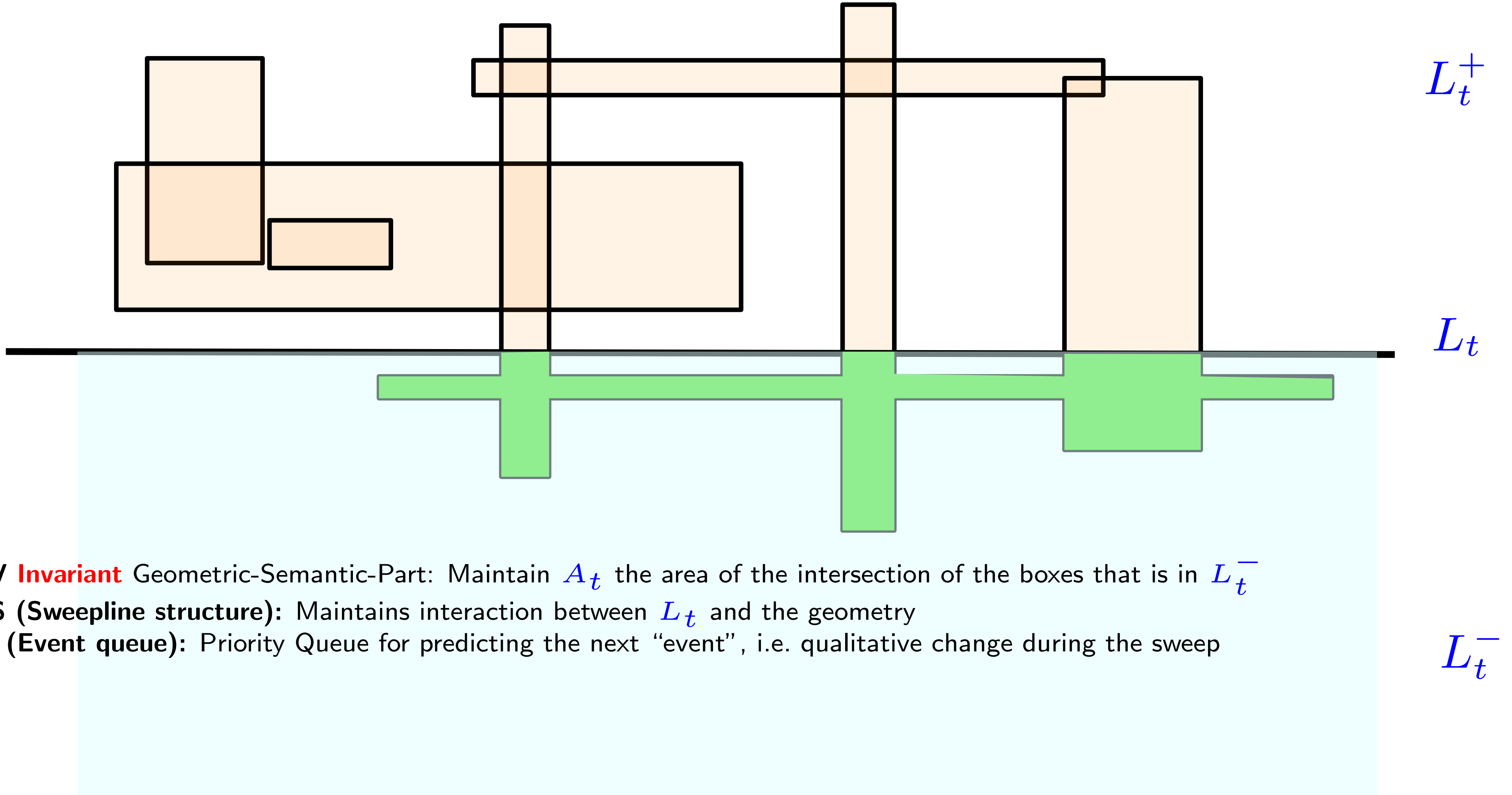
INV Invariant

SLS (Sweepline structure): Maintains interaction between L_t and the geometry

EQ (Event queue): Priority Queue for predicting the next "event", i.e. qualitative change during the sweep

Sweep Algorithms

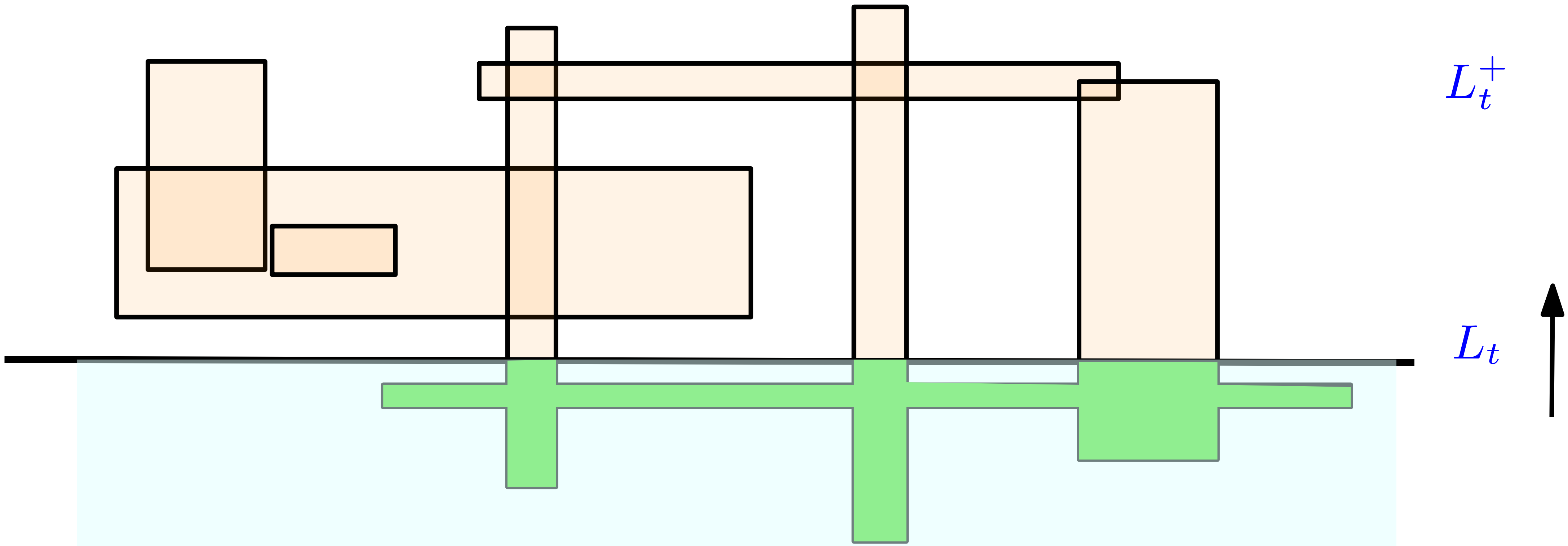
Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.



INV **Invariant** Geometric-Semantic-Part: Maintain A_t the area of the intersection of the boxes that is in L_t^-
SLS (Sweepline structure): Maintains interaction between L_t and the geometry
EQ (Event queue): Priority Queue for predicting the next "event", i.e. qualitative change during the sweep

Sweep Algorithms

Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.



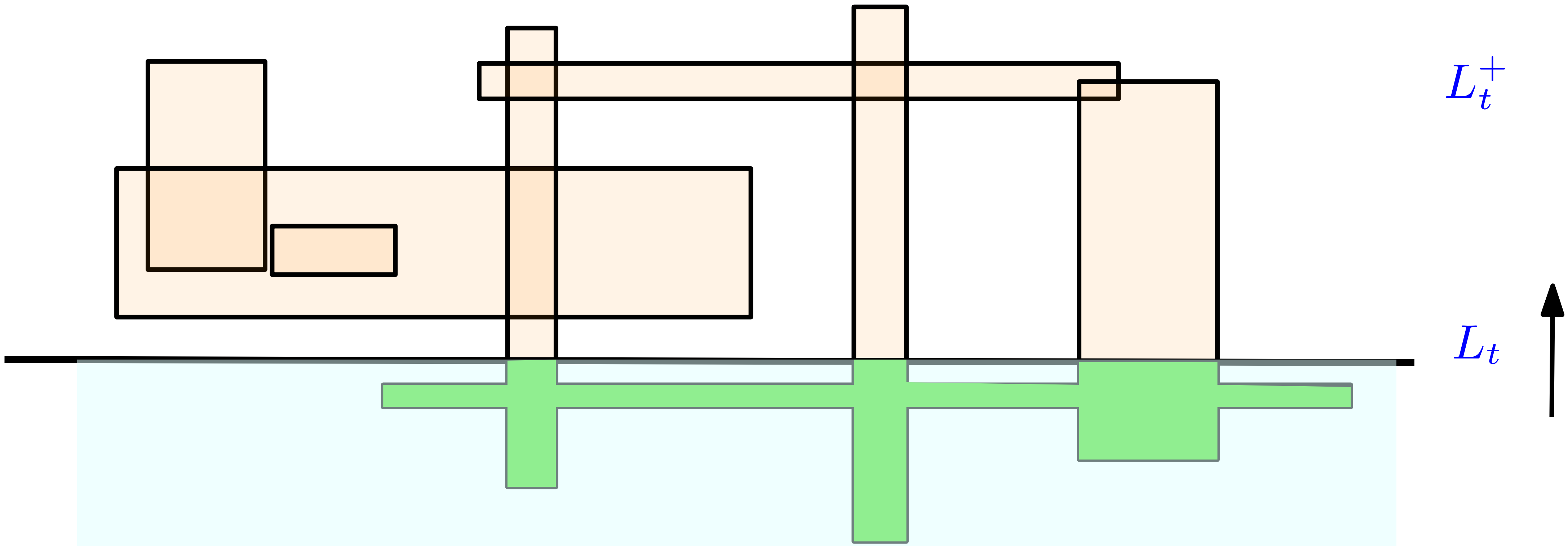
SLS (Sweepline structure): Maintains interaction between L_t and the geometry

Let B_t the boxes in B that intersect L_t. SLS stores the interval set $\{b \cap L_t | b \in B_t\}$ in a strcuture that allows updates and queries for the lenght of the union of all intervals in the structrue.

L_t^-

Sweep Algorithms

Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.

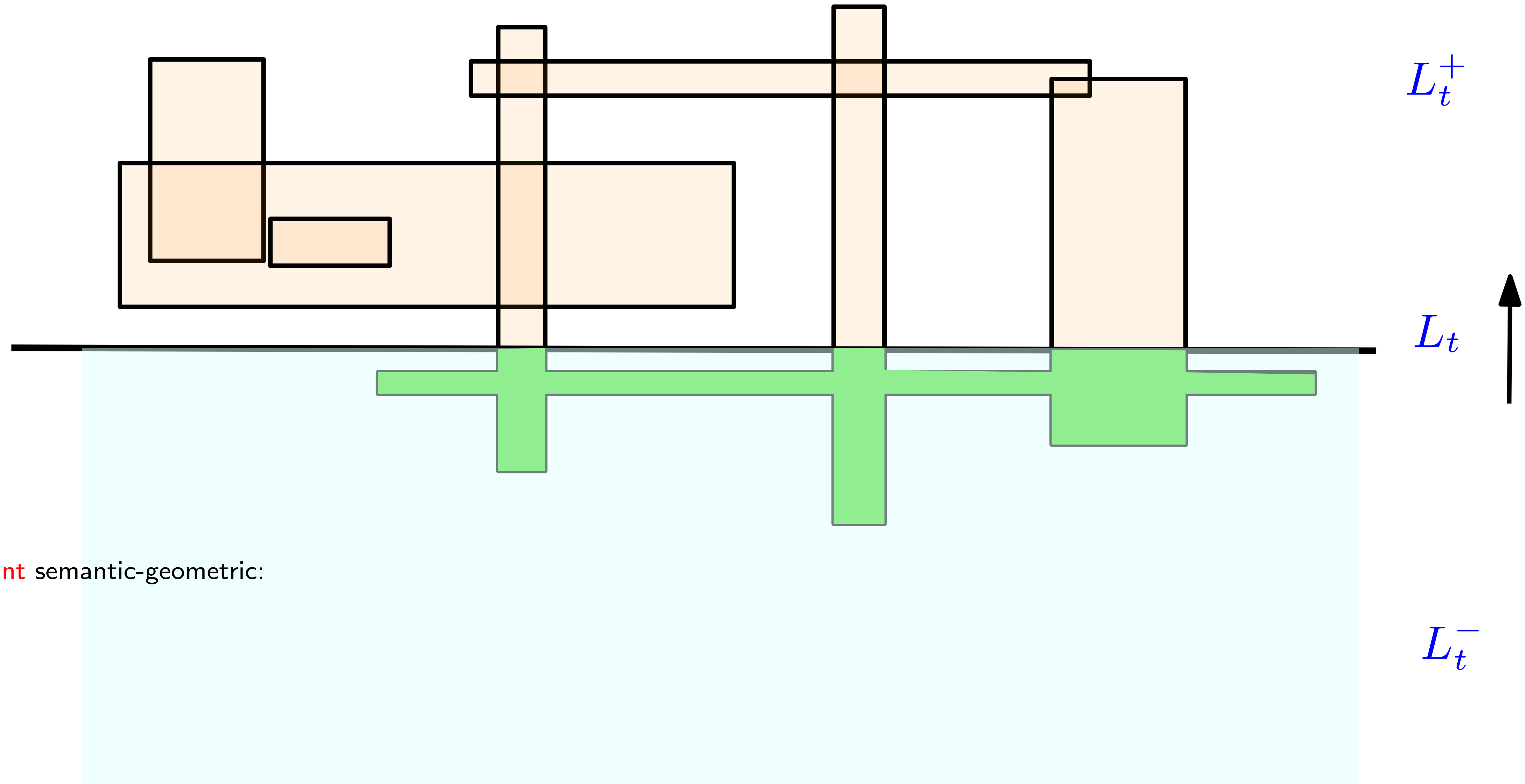


EQ (Event Queue): Events happen when L_t meets a lower or upper edge of a box in B.
There are two types: lower and upper.

EQ maintains all these events in a priority queue.

Sweep Algorithms

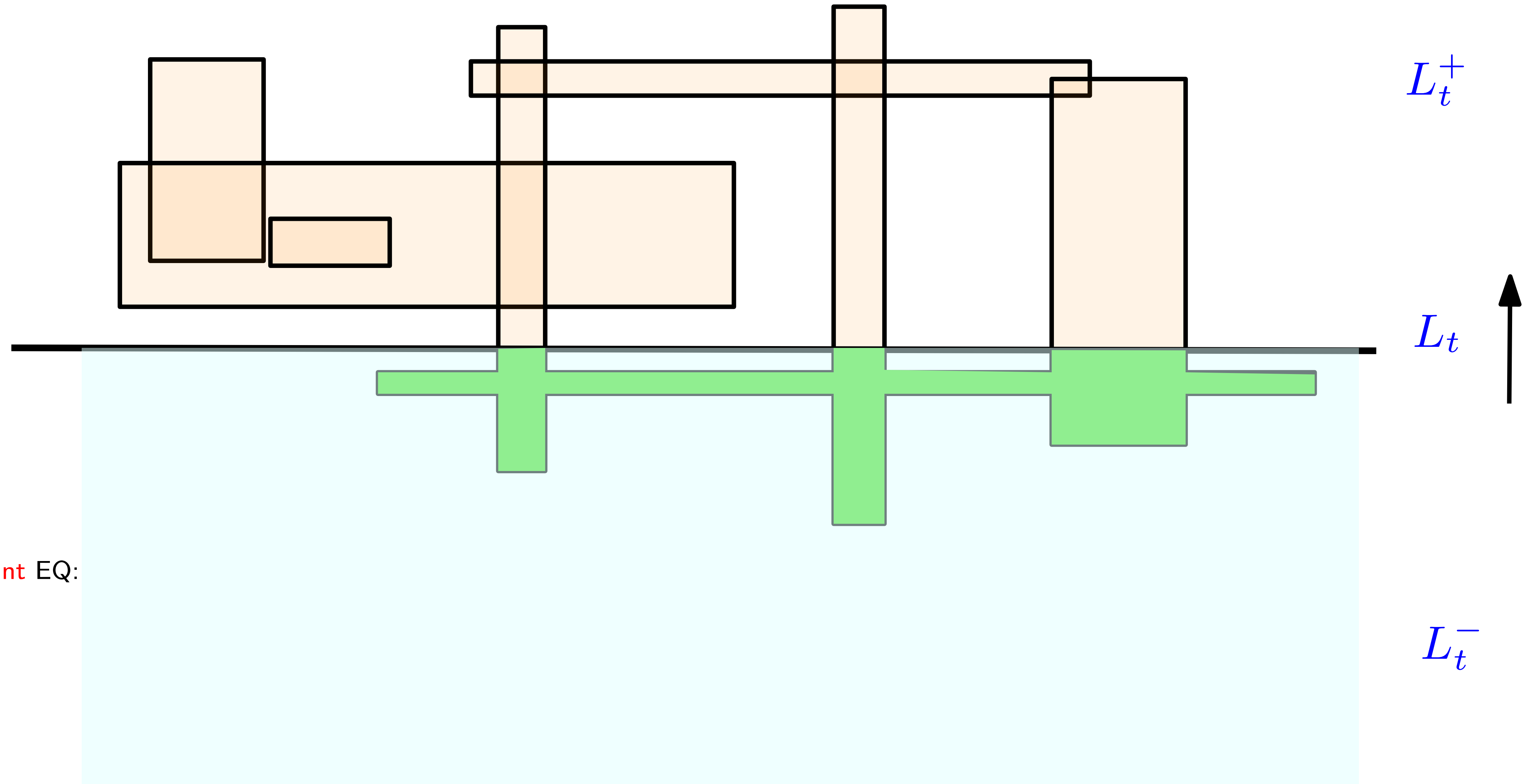
Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.



Invariant semantic-geometric:

Sweep Algorithms

Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.

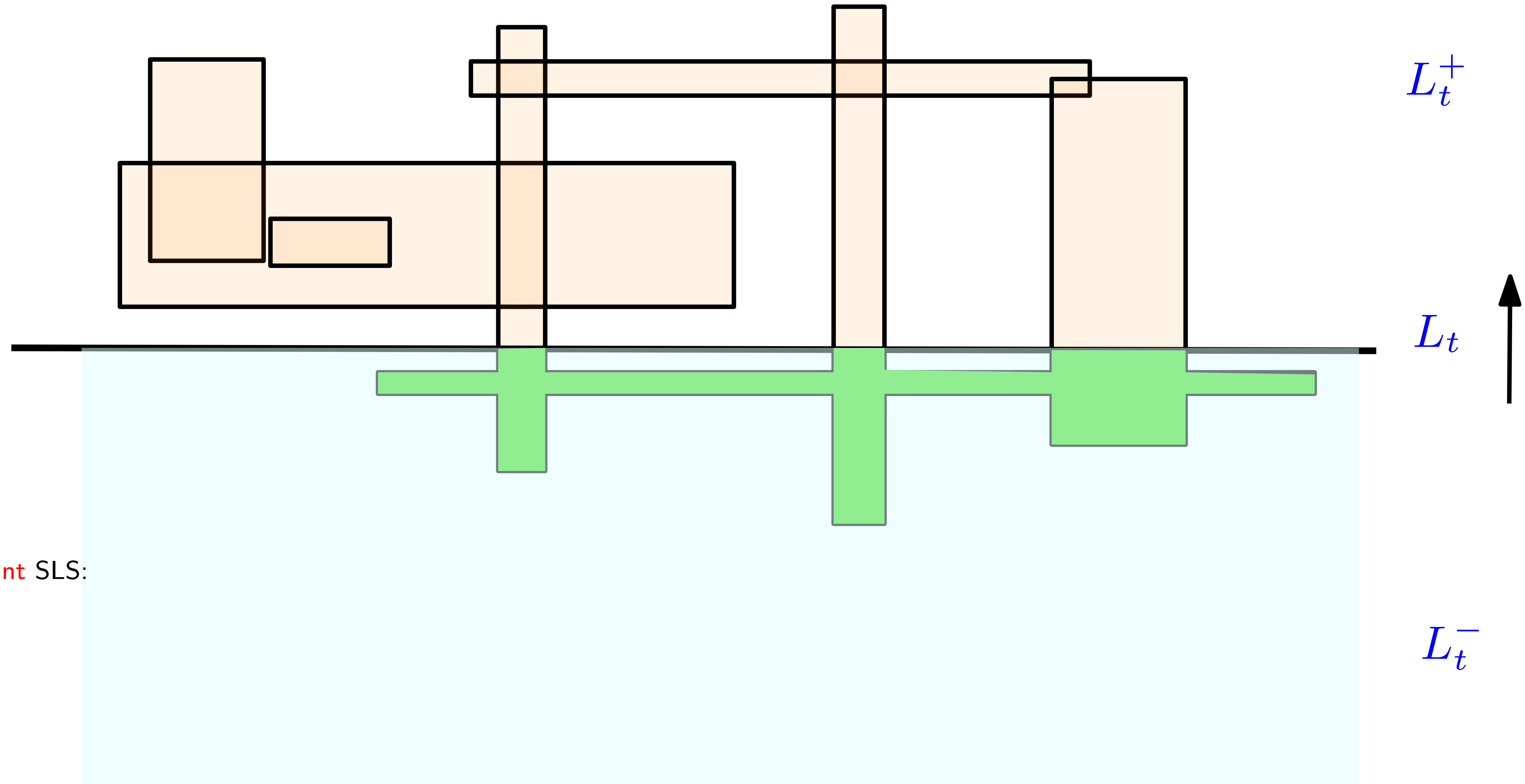


Invariant EQ:

SIC Saarland Informatics Campus

Sweep Algorithms

Example: Given a set of axis parallel boxes in $\mathbb{R}^2$ compute area of their union.

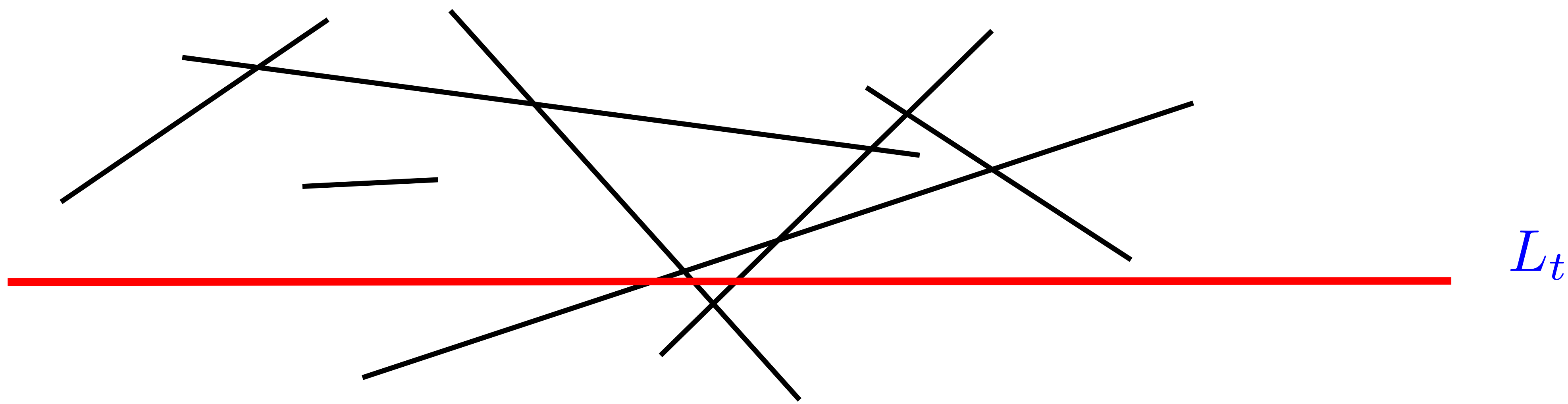


Invariant SLS:

Sweep Algorithms

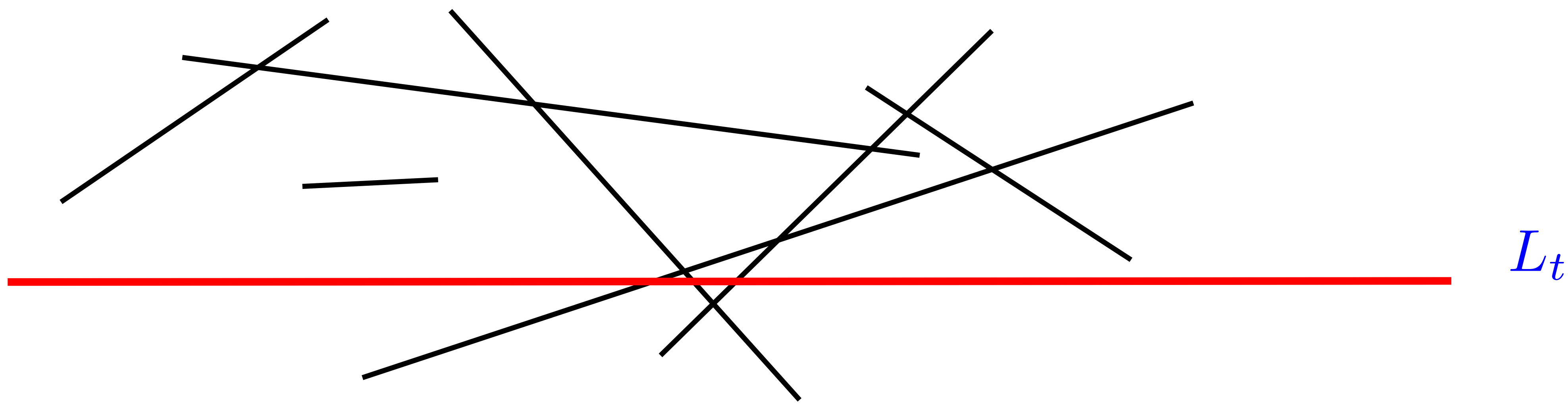
Sweep Algorithms

Example: Given a set S of n non-horitontal segements in the plane, report all their pairwise intersections.



Sweep Algorithms

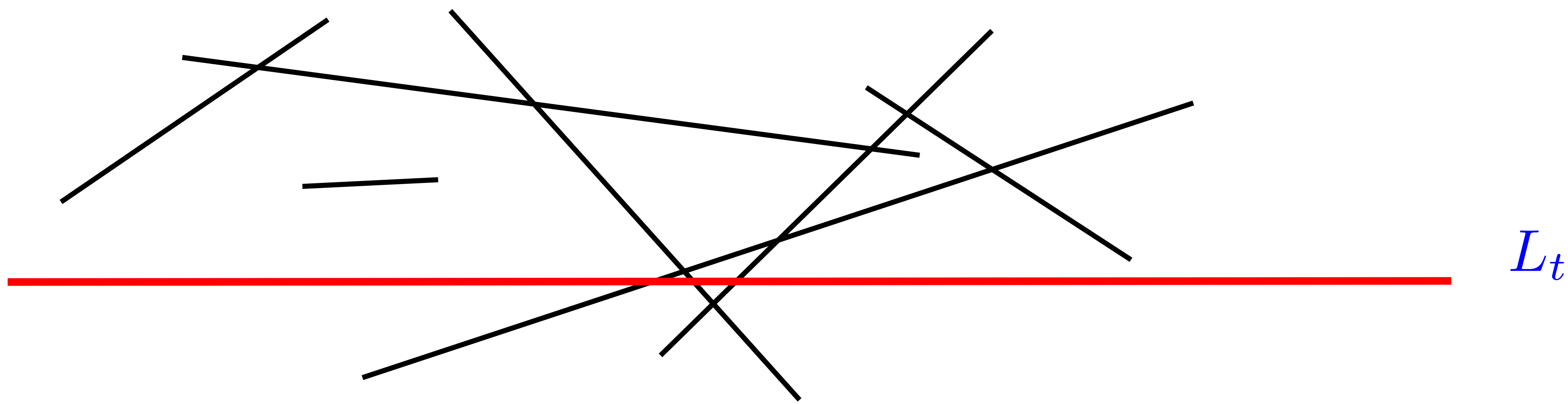
Example: Given a set S of n non-horitontal segements in the plane, report all their pairwise intersections.



Invariant semantic-geometric:

Sweep Algorithms

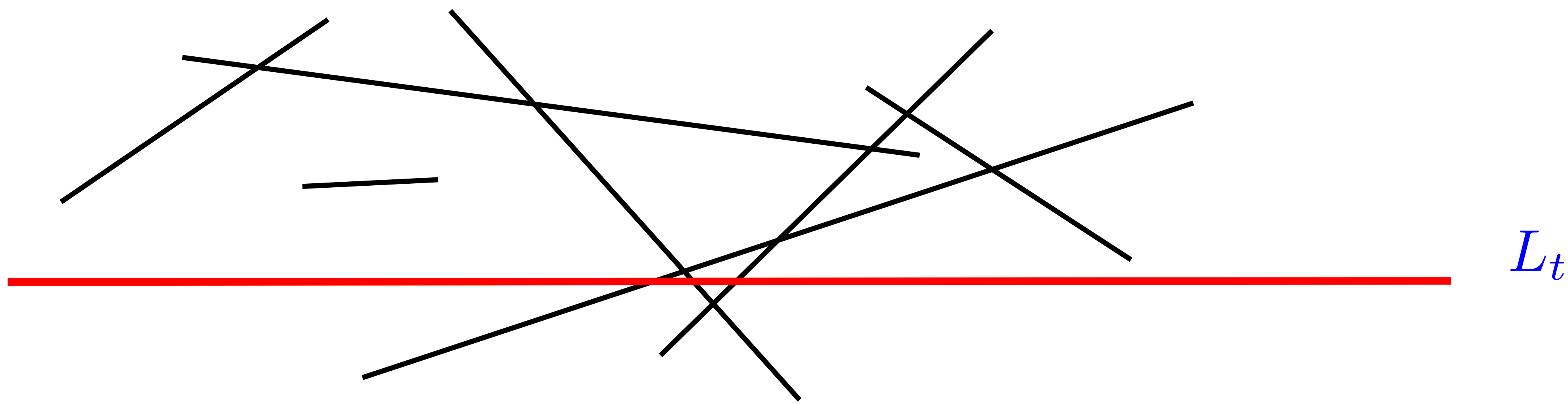
Example: Given a set S of n non-horitontal segements in the plane, report all their pairwise intersections.



Invariant SLS:

Sweep Algorithms

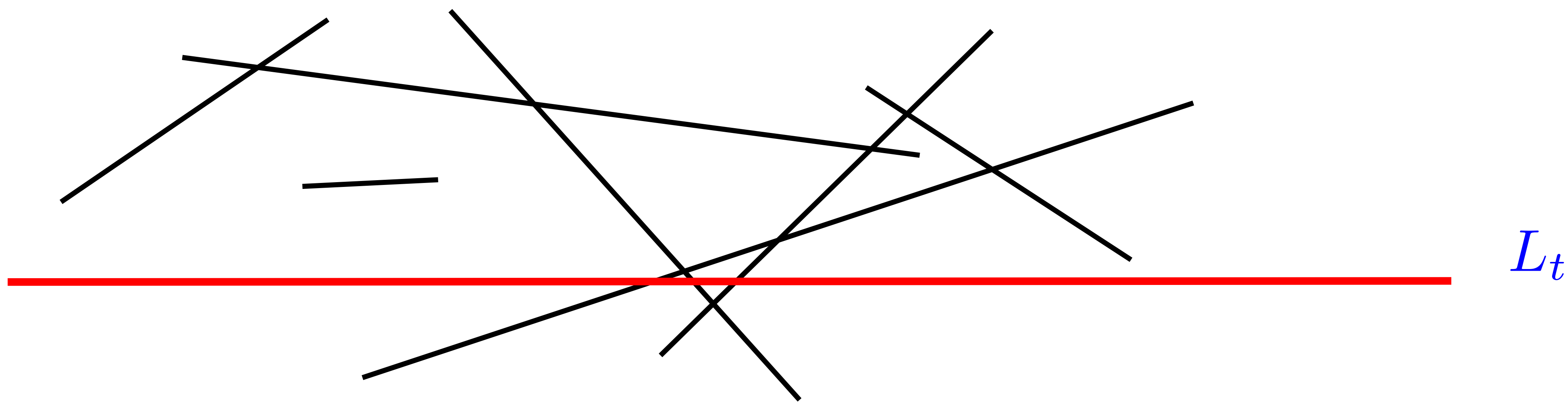
Example: Given a set S of n non-horitontal segements in the plane, report all their pairwise intersections.



Invariant EQ:

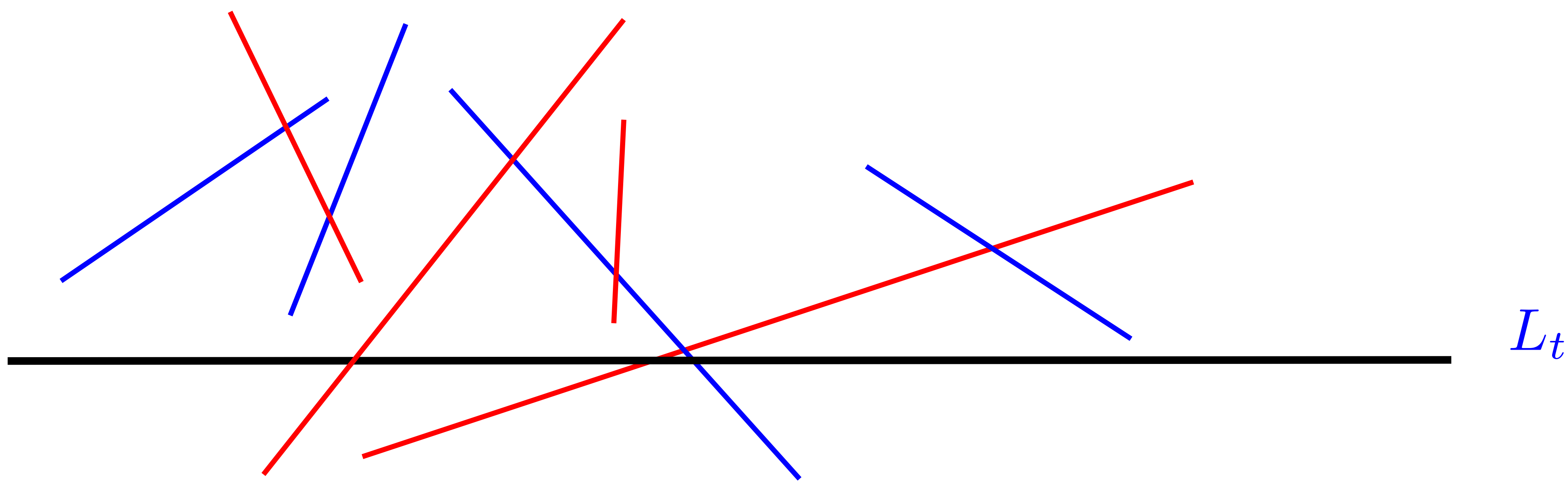
Sweep Algorithms

Example: Given a set S of n non-horitontal segements in the plane, report all their pairwise intersections.



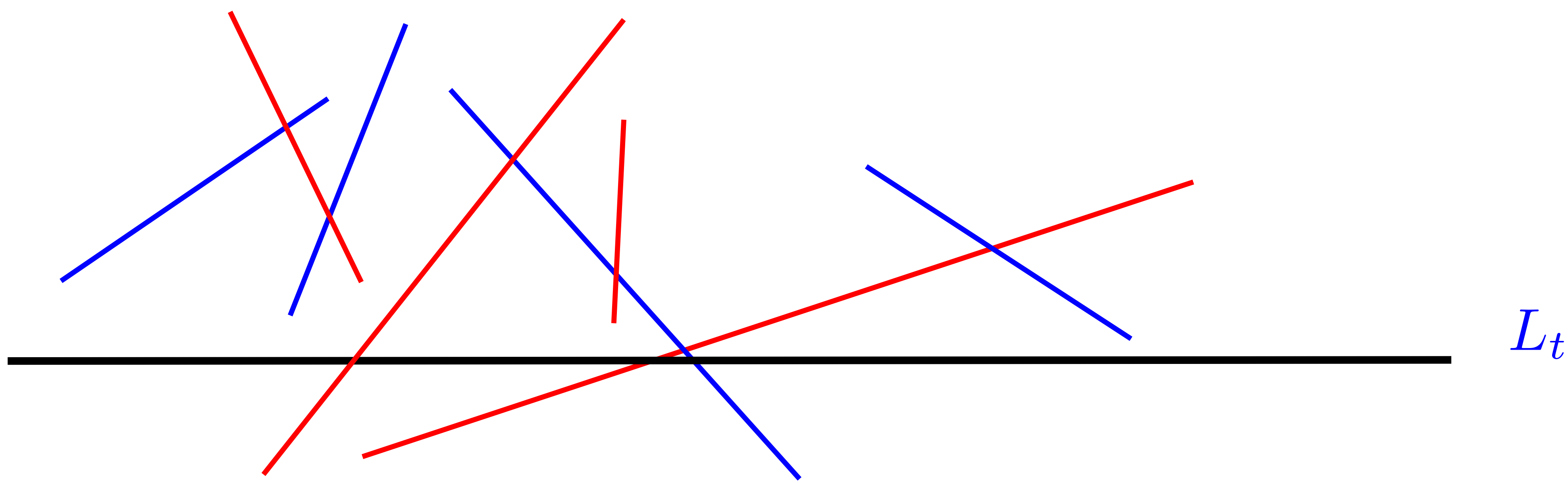
Sweep Algorithms

Example: Given a set B of n non-horizontal, non-intersecting blue segements in the plane and given a set R of n non-horizontal, non-intersecting red segments, report **the number** of red-blue intersections.



Sweep Algorithms

Example: Given a set B of n non-horizontal, non-intersecting blue segements in the plane and given a set R of n non-horizontal, non-intersecting red segments, report **the number** of red-blue intersections.



Sweep Algorithms

Example: Given a set B of n non-horizontal, non-intersecting blue segements in the plane and given a set R of n non-horizontal, non-intersecting red segments, report **the number** of red-blue intersections.