Course 16
Geometric Data Structures for Computer Graphics
Generic Dynamization

Dr. Elmar Langetepe

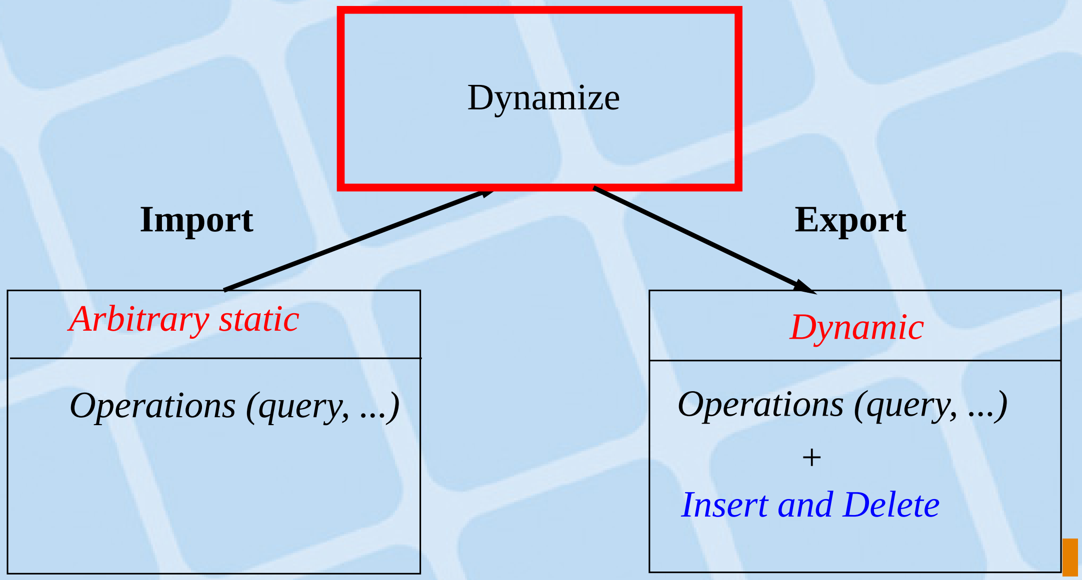
Institut für Informatik I
Universität Bonn

Motivation

- Some geometric data structures were considered
- So far we assumed that the structure is *static*
- Fixed set of objects
- Objects may changes over time, Delete, Insert
- Need special update techniques for Insert and Delete for every data structure
- Now: One Delete/Insert techniques suitable for many data structures
- Generic dynamization

Motivation

- Consider *Static* geometric data structure for n *fixed* objects
- Assume: Static version is easy to implement
- Assume: Efficient *Range Queries* on *fixed* set
- Assume: Objects changes over time, Delete, Insert



Implement Delete/Insert only once

For many data structures (no special dynamization)

Model: $V \in TStat$

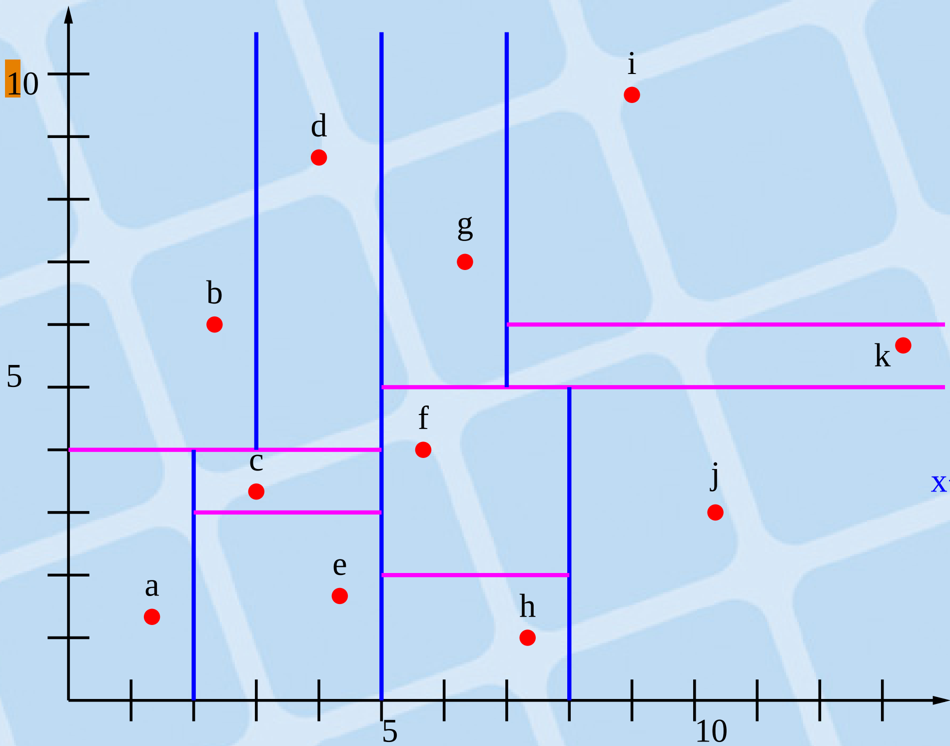
V is a static geometric data structure of Type $TStat$
With object set D
Operations:

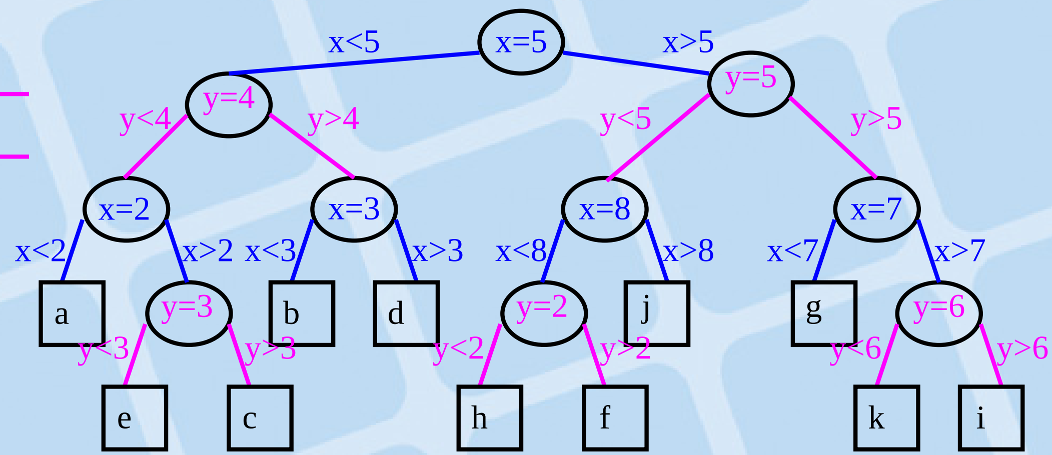
build(V, D): Build the structure *V* of type *TStat* with all data objects in the set *D*.

query(V, q): Gives the answer (objects of *D*) to a query to *V* with query object q.

extract(V, D): Collects all data objects D of V in a single set and returns a pointer to it.

Example: Balanced k-d-tree





Balanced k-d-tree of set D:

build: $O(n \log n)$

query, orthogonal range: $O(\sqrt{n} + a)$

Dynamic structure $W \in TDyn$

SIGGRAPH 2003 SAN DIEGO

Operations:

Build(W, D): Build the structure *W* of type *TDyn* with data objects in the set *D*.

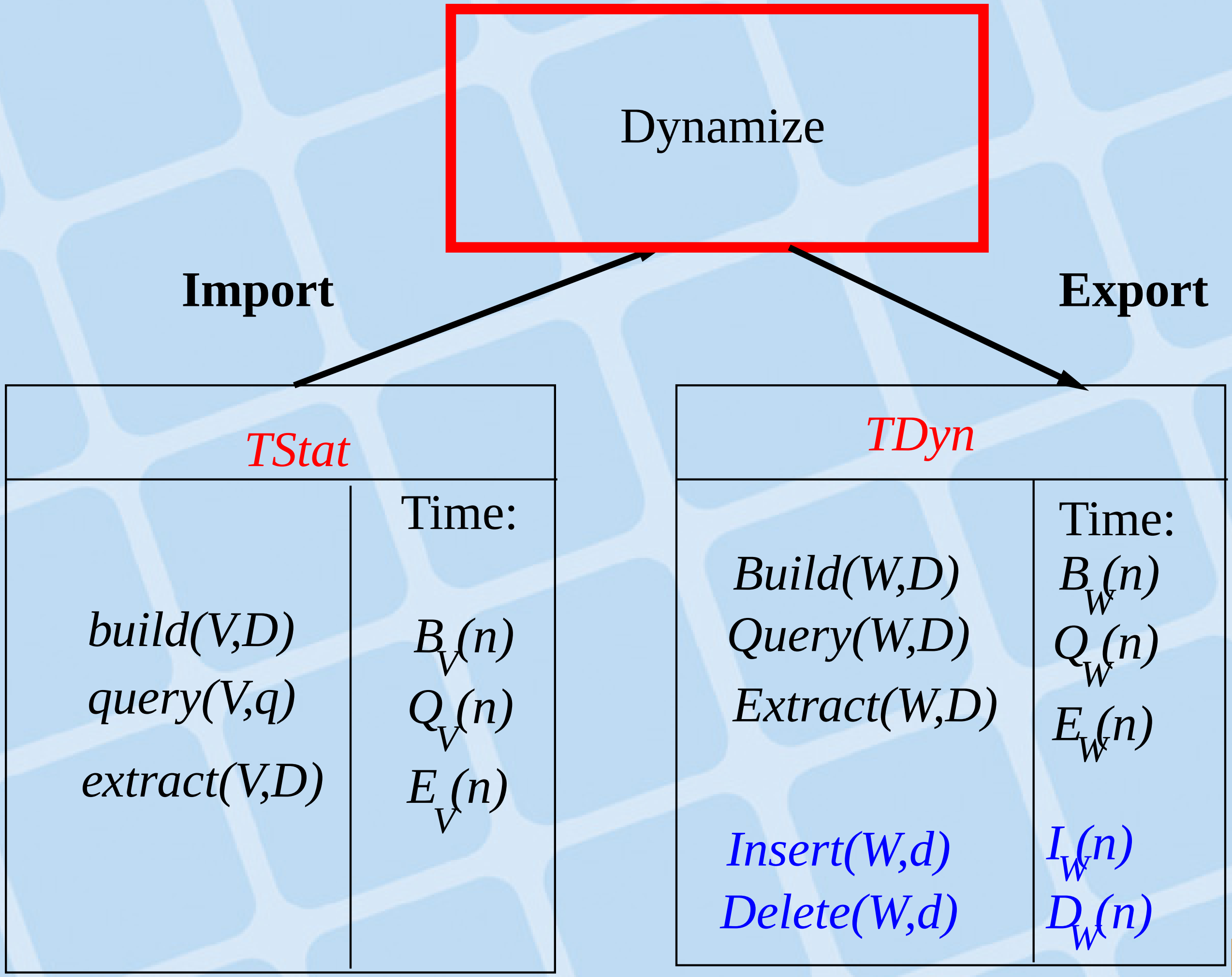
Query(W, q): Gives the answer (objects of *D*) to a query to *W* with query object q.

Extract(W, D): Collects all data objects D of *W* in a single set and returns a pointer to it.

Insert(W, d): Insert object d into W.

Delete(W, d): Delete d out of W.

Model of the dynamization



Simple *Throw-Away* solution

$$\textit{Insert}(W, d) : \quad \textit{extract}(W, D); \textit{build}(W, D \cup \{d\})$$
$$\textit{Delete}(W, d) : \quad \textit{extract}(W, D); \textit{build}(W, D \setminus \{d\}).$$

Ineffective, approach must cope with *small* changes

Requirements

- Query operation: Decomposable
  - $V = V_1 \cup V_2 \cup \cdots \cup V_j \Rightarrow (query(V_i, d) \Rightarrow query(V, d))$
  - Almost all geometric DS, Example: k-d tree
- Time-Functions increase monotonically in n
  - Functions like: n, $n \log n$, n^2, 2^n, $\sqrt{n}$
  - Example: k-d tree,
    $B_V(n) = O(n \log n)$, $E_V(n) = O(n)$, $Q_V(n) = O(\sqrt{n} + a)$
- A few others, normally fulfilled

Amortized Insert: Binary structure

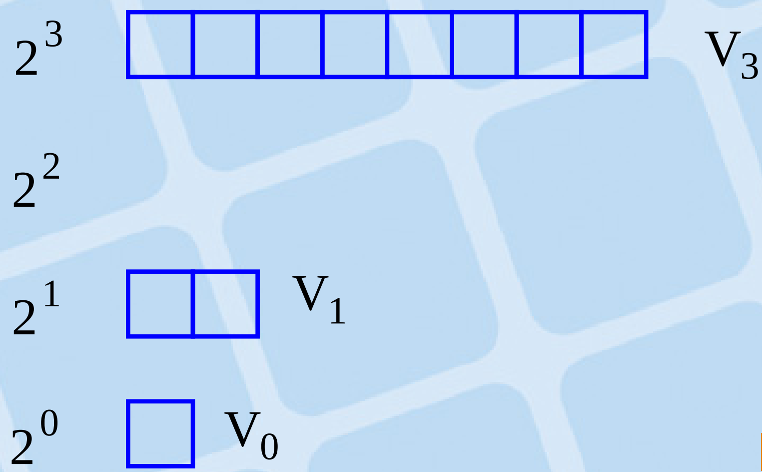
- Set D of objects, $|D| = n$
- Decompose D into sets D_i
- Binary representation of n:
  - $n = a_l 2^l + a_{l-1} 2^{l-1} + \ldots + a_1 2 + a_0$ mit $a_i \in \{0, 1\}$
  - Binary representation: $a_l a_{l-1} \ldots a_1 a_0$
- $a_i = 1 \Rightarrow$ build static structure V_i with 2^i elements of D.

Amortized Insert: Binary structure

SIGGRAPH 2003 SAN DIEGO

Set D of objects, $|D| = n$

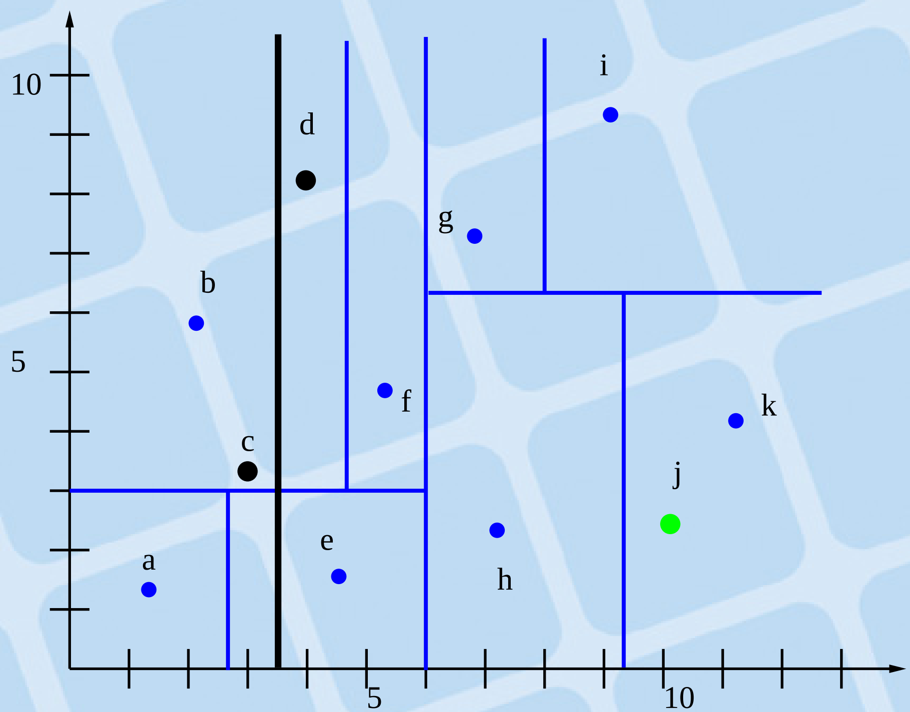
$$|D| = n = 11 = \mathbf{1} \cdot \mathbf{2^3} + 0 \cdot 2^2 + 1 \cdot \mathbf{2^1} + 1 \cdot \mathbf{2^0}$$

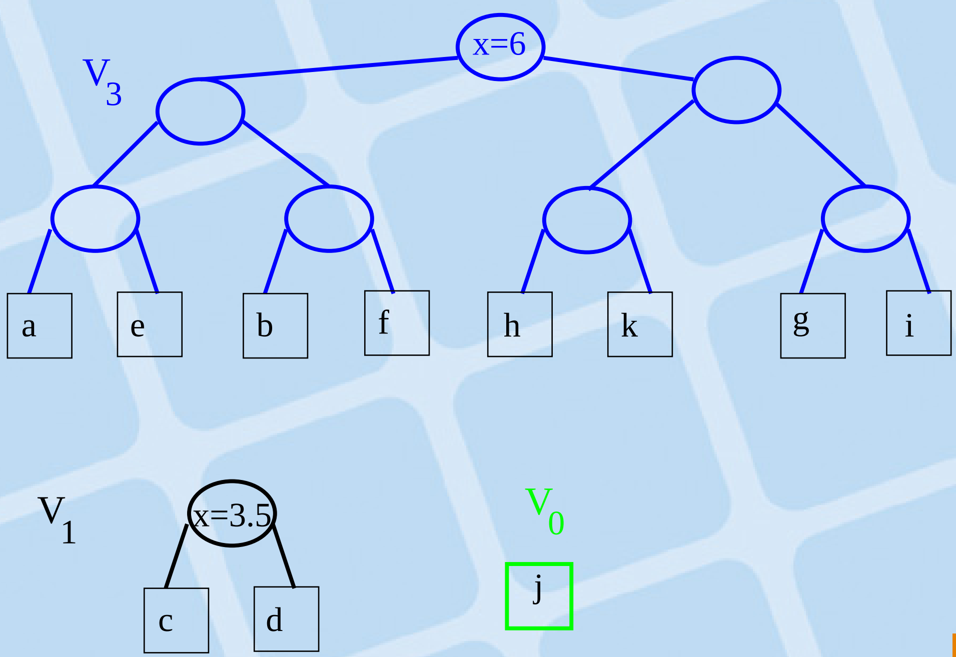


Binary structure W_n, consists of some static structures V_i

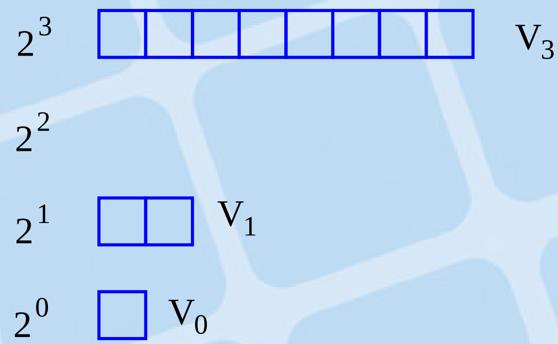
k-d tree Binary structure

$$11 = 1 \cdot \mathbf{2^3} + 0 \cdot 2^2 + 1 \cdot \mathbf{2^1} + 1 \cdot \mathbf{2^0}$$





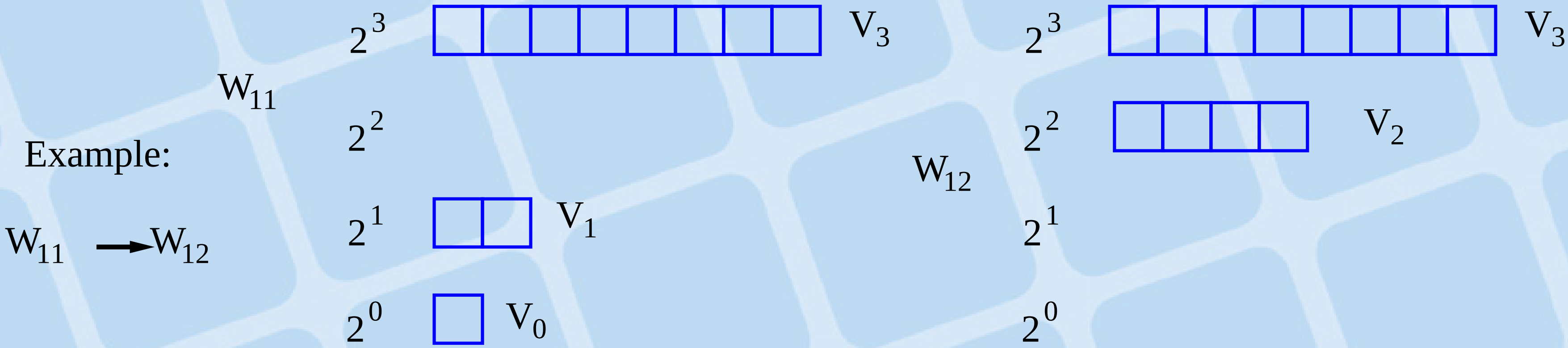
Binary structure: Operations cost



- *Build(W,D)*: $B_W(n) \in O(B_V(n))$ (k-d-tree: $O(n \log n)$)
- *Extract(W,D)*: $E_W(n) \leq \log n \; E_V(n)$ (k-d-tree: $O(n \log n)$)
- *Query(W,D)*: $Q_W(n) \leq \log n \; Q_v(n)$ (k-d-tree: $O(\log n(\sqrt{n} + a))$)

Amortized Insert

Insert new element d: Structural changes of the *binary representation*



To Do: $extract(V_0, D_0)$; $extract(V_1, D_1)$;
$D := D_0 \cup D_1 \cup \{d\}$; $build(V_2, D)$;

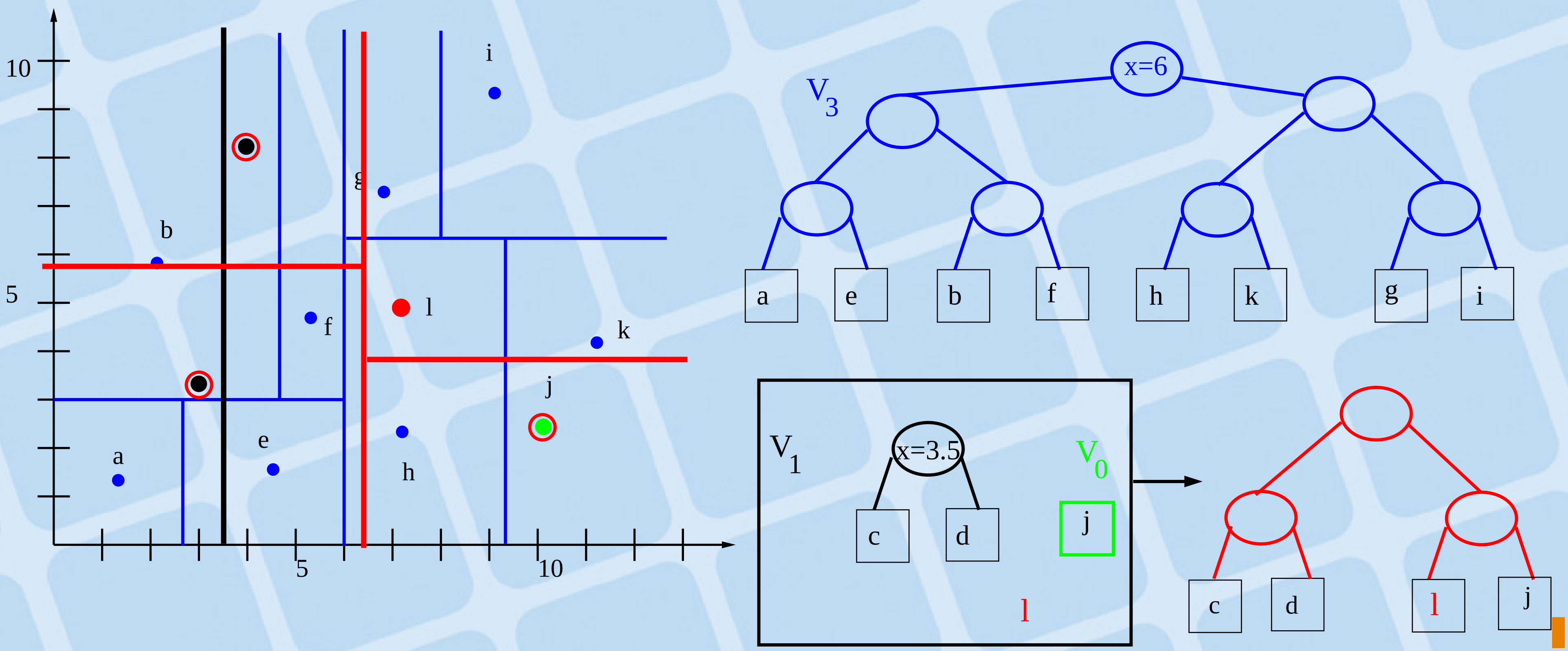
Next insert without changes of old parts!

Reconstruction partially!

Example: k-d tree Binary structure

$$|D| = 11 = 1 \cdot \mathbf{2^3} + 0 \cdot 2^2 + 1 \cdot \mathbf{2^1} + 1 \cdot \mathbf{2^0}$$

$$|D \cup \{l\}| = 12 = 1 \cdot \mathbf{2^3} + 1 \cdot \mathbf{2^2} + 0 \cdot 2^1 + 0 \cdot 2^0$$



Amortized Insert: Costs

Amortized time: Start with empty structure

Sequence S of $|S| = s$ operations, k *Insert* operations

$S = (Insert, op2, op1, Insert, Insert, Insert, op1, \ldots)$

$$\frac{\text{tot. cost of } k \text{ \textit{Insert} oper.}}{k} \leq \overline{I}(s)$$

Means: Insert in $\overline{I}(s)$ amortized time.

One can show:

$$\overline{I}_W(s) \in O\left(\frac{\log s}{s} B_V(s)\right).$$

(k-d-tree: $O(\log^2 s)$)

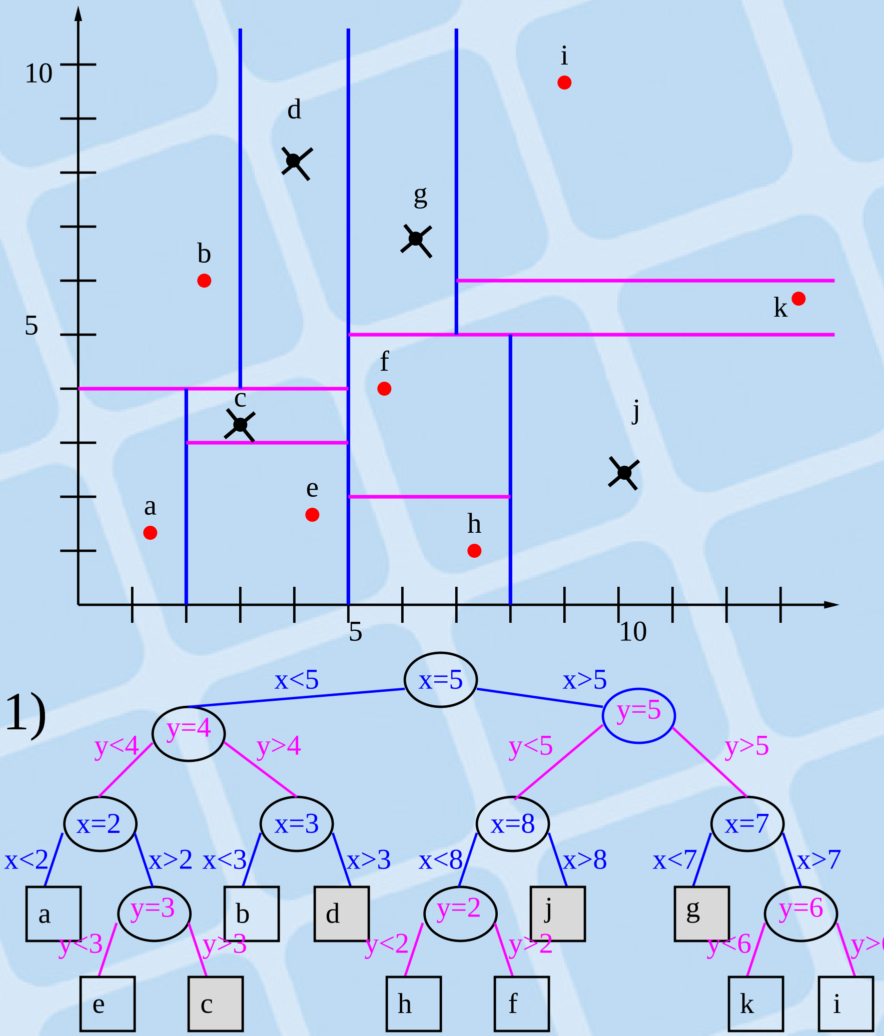
Amortized Delete

- First: Stand alone, without generic Insert(Bin. Str.)
- Weak-Delete operation on static structure
- Example: k-d-tree
  - Mark point as deleted
  - Proceed as before
  - Occasional reconstruct

Weak Delete: k-d-tree

|D|=7

|V|=O(11)



Reconstruct completely IFF D has only the half-size of V

Amortized Delete: Results

- Requires Static structure with $weak.Delete(V, d)$ operation
- Cost function: $WD_V(n)$ (k-d-tree: $O(\log n)$)
- r size of the actual data set, s length of operation sequence
- Dynamization by Occasional reconstruction

$$\begin{aligned} B_W(r) &= B_V(r) \text{(} k\text{-}d\text{-tree: } O(r\log r)) \\ E_W(r) &\in O\left(E_V(r)\right) \text{(} k\text{-}d\text{-tree: } O(r)) \\ Q_W(r) &\in O\left(Q_V(r)\right) \text{(} k\text{-}d\text{-tree: } O(\sqrt{r}+a)) \end{aligned}$$

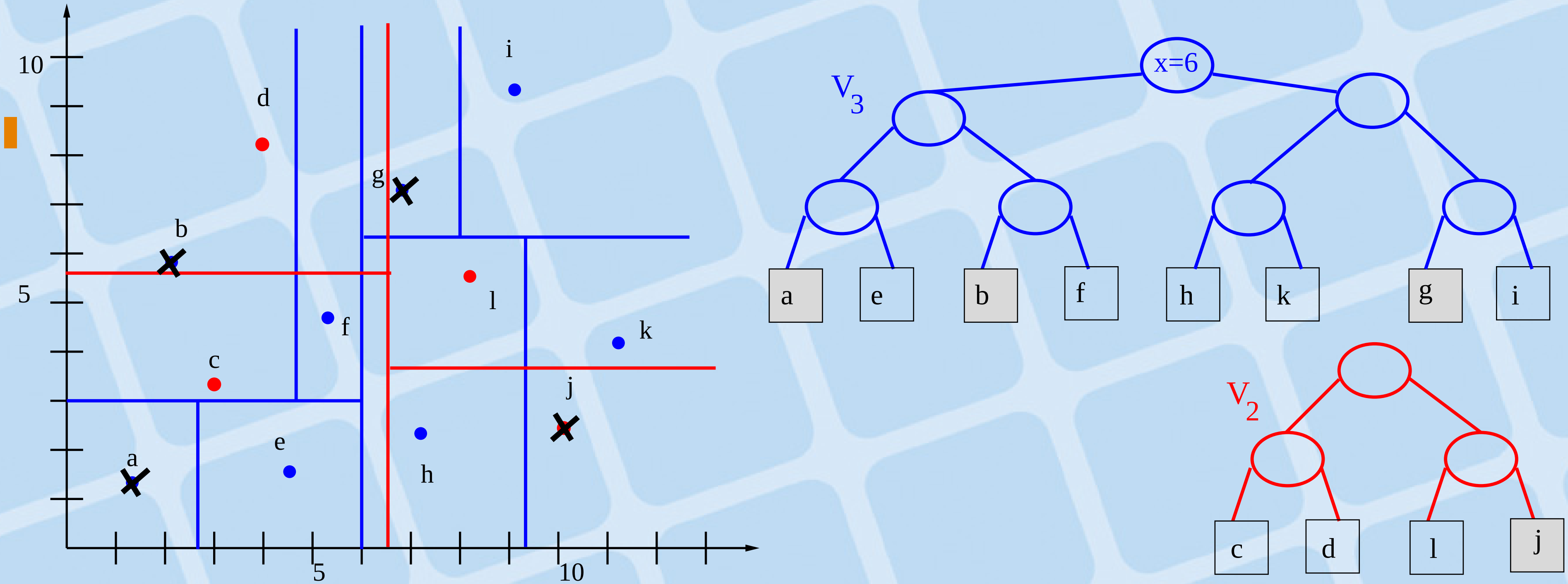
$S = (Insert, op1, Insert, op2, Delete, op2, Delete, op1, \ldots)$

Start with empty structure

Amortized Delete: $|S| = s$

$$\overline{D}_W(s) \in O\left(WD_V(s) + \frac{B_V(s)}{s}\right) \text{(} k\text{-}d\text{-tree: } O(\log s))$$

Combine: Amortized Insert/Delete



$Weak.Delete(W, d)$: Find the structure V_i of binary structure W

Implemented by a searchtree T for all elements

Results: Amortized Insert/Delete

r size of the actual data set, s length of operation sequence

$S = (Insert, op2, Insert, Delete, op2, Delete, Insert, op1, \ldots)$

Amortized time for insertion:

$$\overline{I}_W(s) \in O\left(\log s \frac{B_V(s)}{s}\right) \quad (k\text{-}d\text{-tree: } O(\log^2 s)),$$

Amortized time for deletion:

$$\overline{D}_W(s) \in O\left(\log s + W D_V(s) + \frac{B_V(s)}{s}\right) \quad (k\text{-}d\text{-tree: } O(\log s)).$$

Other operations:

$$\begin{aligned}
B_W(r) &= B_V(r) (\ k\text{-}d\text{-tree: } O(r \log r)) \\
E_W(r) &\in O(\log r \ E_V(r)) (\ k\text{-}d\text{-tree: } O(r \log r)) \\
Q_W(r) &\in O(\log r \ Q_V(r)) (\ k\text{-}d\text{-tree: } O(\log r \ (\sqrt{r} + a)))
\end{aligned}$$

Conclusion

- Simple generic dynamization techniques
- Easy to implement: Binary structure/Occasional reconstruction
- Amortized Delete and Insert
- Applicable for many geometric data structures
- Efficient: $\log$ factor
- Does not waste storage
- Worst-Case sensitive: Amortize the reconstruction itself