Dynamic Data Structures: The Interplay of Invariants and Algorithm Design

Casper Kejlberg-Rasmussen
PhD Defense, 18th of November 2013
Outline

- Introduction to Algorithms and Data Structures
- Implicit Working-Set Dictionaries
- Skyline Queries
- Catenable Priority Queues with Attrition
- Conclusion
What are Algorithms and Data Structures?
What are Algorithms and Data Structures?

Algorithm ≈ Recipe

Data ≈ Ingredients
What are Algorithms and Data Structures?

Algorithm \( \equiv \) Recipe

Data \( \equiv \) Ingredients

Data Structure \( \equiv \) Organization of Ingredients

Updates \( \equiv \) Ingredients Updates

Queries \( \equiv \) Follow Recipes
What are Algorithms and Data Structures?

Algorithm ≈ Recipe
Data ≈ Ingredients

Data Structure ≈ Organization of Ingredients
Updates ≈ Ingredients Updates
Queries ≈ Follow Recipes

Design Criteria
What are Algorithms and Data Structures?

Algorithm ≅ Recipe
Data ≅ Ingredients

Data Structure ≅ Organization of Ingredients
Updates ≅ Ingredients Updates
Queries ≅ Follow Recipes

Design Criteria

- Fast
- Low Space Usage
What are Algorithms and Data Structures?

Algorithm $\cong$ Recipe

Data $\cong$ Ingredients

Data Structure $\cong$ Organization of Ingredients

Updates $\cong$ Ingredients Updates

Queries $\cong$ Follow Recipes

Design Criteria

- Fast
- Low Space Usage

- Fast queries and updates
- Low Space Usage
What are Computational Models?
What are Computational Models?

Reality
What are Computational Models?

Reality

- CPU
- L1
- L2
- L3
- Memory
- Harddisk
- etc.
What are Computational Models?

**Reality**

- CPU
- L1
- L2
- L3
- Memory
- Harddisk
- etc.

**Complexity**

\[
(\text{CPUSpeed} \cdot L1 \cdot \left\lfloor \frac{L2}{L1} \right\rfloor \cdot \left\lfloor \frac{L3}{L2} \right\rfloor \cdot \left\lfloor \frac{n}{L3} \right\rfloor) \log n
\]
What are Computational Models?

Reality

CPU
L1
L2
L3

Memory

Harddisk

Models

CPU
Memory

Count operations

RAM Model

Complexity

\[ (\text{CPUSpeed} \cdot L1 \cdot \left\lceil \frac{L2}{L1} \right\rceil \cdot \left\lceil \frac{L3}{L2} \right\rceil \cdot \left\lceil \frac{n}{L3} \right\rceil) \log n \]
What are Computational Models?

Reality

<table>
<thead>
<tr>
<th>CPU</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>Memory</th>
<th>Harddisk</th>
</tr>
</thead>
</table>

Models

RAM Model

Complexity

\[(CPUSpeed \cdot L1 \cdot \frac{L2}{L1} \cdot \frac{L3}{L2} \cdot \frac{n}{L3}) \log n\]

\[O(n \log n)\]
What are Computational Models?

Reality

Models

CPU

L1

L2

L3

CPU

Memory

Harddisk

Count operations

Count disk accesses

CPU

Memory

Harddisk

Complexity

\[
(CPU\text{Speed} \cdot L1 \cdot \left[\frac{L2}{L1}\right] \cdot \left[\frac{L3}{L2}\right] \cdot \left[\frac{n}{L3}\right]) \log n
\]

\[O(n \log n)\]
Static and Dynamic Problems

Static

Car offers

Price

Min

Max

Quality

Low

High

5/36
We want to buy a new car!

We have a list of offers

We want to find the *undominated* offers, i.e. unmatched in price and quality
We want to buy a new car!

We have a list of offers

We want to find the undominated offers, i.e. unmatched in price and quality
Static and Dynamic Problems

Static

Price
Min
Max
Low
High

Quality

Dynamic

Price
Min
Max
Low
High

Quality

Car offers

- We want to buy a new car!
- We have a list of offers
- We want to find the undominated offers, i.e. unmatched in price and quality

Car offers

- In the dynamic setting, we will receive new offers continuously
- New offers might change the set of undominated offers
We want to buy a new car!

We have a list of offers

We want to find the undominated offers, i.e. unmatched in price and quality

In the dynamic setting, we will receive new offers continuously

New offers might change the set of undominated offers
We want to buy a new car!

We have a list of offers

We want to find the undominated offers, i.e. unmatched in price and quality

In the dynamic setting, we will receive new offers continuously

New offers might change the set of undominated offers
We want to buy a new car!

We have a list of offers

We want to find the *undominated* offers, i.e. unmatched in price and quality

In the dynamic setting, we will receive new offers continuously

New offers might change the set of *undominated* offers
What are Invariants?

An *Invariant* is a logical statement about the structural properties of a data structure.
What are Invariants?

An *Invariant* is a logical statement about the structural properties of a data structure

- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the *car offers problem* from before
What are Invariants?

An Invariant is a logical statement about the structural properties of a data structure

- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the car offers problem from before
What are Invariants?

An *Invariant* is a logical statement about the structural properties of a data structure

- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the *car offers problem* from before
What are Invariants?

An *Invariant* is a logical statement about the structural properties of a data structure

- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the *car offers problem* from before

$c_2$ dominates $c_1$
What are Invariants?

An Invariant is a logical statement about the structural properties of a data structure

- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the car offers problem from before

\[ C_2 \text{ dominates } C_1 \]
What are Invariants?

An **Invariant** is a logical statement about the structural properties of a data structure

- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the *car offers problem* from before

\[ c_2 \text{ dominates } c_1 \]
\[ c_1 \text{ and } c_3 \text{ are incomparable} \]
What are Invariants?

An *Invariant* is a logical statement about the structural properties of a data structure

- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the *car offers problem* from before

\[
\begin{align*}
&c_2 \text{ dominates } c_1 \\
&c_1 \text{ and } c_3 \text{ are incomparable} \\
&c_2 \text{ and } c_3 \text{ are incomparable}
\end{align*}
\]

![Diagram showing car offers with price and quality axes, and three car offers c1, c2, c3, with c2 dominating c1 and c2 and c3 being incomparable.](image)
What are Invariants?

An *Invariant* is a logical statement about the structural properties of a data structure

- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the *car offers problem* from before

\[
\begin{align*}
&c_2 \text{ dominates } c_1 \\
&c_1 \text{ and } c_3 \text{ are incomparable} \\
&c_2 \text{ and } c_3 \text{ are incomparable}
\end{align*}
\]

- We notice that the *undominated* \( (c_2 \text{ and } c_3) \) offers are sorted both according to price and quality simultaneously
What are Invariants?
What are Invariants?

• From our observation we store all undominated points in a search tree sorted simultaneously on price and quality

• This gives us the following data structure and invariant
What are Invariants?

- From our observation we store all undominated points in a search tree sorted simultaneously on price and quality.
- This gives us the following data structure and invariant:

  Invariant: All undominated offers are stored in the search tree $T$ and are sorted simultaneously on price and quality.
What are Invariants?

- From our observation we store all undominated points in a search tree sorted simultaneously on price and quality.
- This gives us the following data structure and invariant:

  Invariant: All undominated offers are stored in the search tree $T$ and are sorted simultaneously on price and quality.
What are Invariants?

- From our observation we store all undominated points in a search tree sorted simultaneously on price and quality.
- This gives us the following data structure and invariant:

Invariants: All **undominated** offers are stored in the search tree $T$ and are sorted simultaneously on price and quality.
What are Invariants?

- From our observation we store all undominated points in a search tree sorted simultaneously on price and quality.
- This gives us the following data structure and invariant:

  Invariant: All undominated offers are stored in the search tree $T$ and are sorted simultaneously on price and quality.
What are Invariants?

- From our observation we store all undominated points in a search tree sorted simultaneously on price and quality.
- This gives us the following data structure and invariant:

  **Invariant:** All undominated offers are stored in the search tree $T$ and are sorted simultaneously on price and quality.
What are Invariants?

- From our observation we store all undominated points in a search tree sorted simultaneously on price and quality.

- This gives us the following data structure and invariant:

  Invariant: All undominated offers are stored in the search tree $T$ and are sorted simultaneously on price and quality.
What are Invariants?

- From our observation we store all undominated points in a search tree sorted simultaneously on price and quality.
- This gives us the following data structure and invariant:

  **Invariant**: All undominated offers are stored in the search tree $T$ and are sorted simultaneously on price and quality.

- Where $k$ is the number of undominated car offers out of all offers.
- Inserting a new car offer takes $O(\log k)$ time.
- Reporting the undominated offers takes $O(k)$ time.
Designing Invariants and Dynamic Data Structures
Designing Invariants and Dynamic Data Structures

- Dynamic data structure and invariant design follows a cycle:
  - Observe properties of the problem
  - Formulate invariants
  - Check if the invariants are strong enough to support queries
  - Check if the invariants can be maintained under updates
- The process is similarly to suitcase packing:
  1. We place our stuff in the suitcase
  2. We check if the lid can be closed
- When everything fits inside the suitcase, we are done!
Designing Invariants and Dynamic Data Structures

- Dynamic data structure and invariant design follows a cycle:
  - Observe properties of the problem
  - Formulate invariants
  - Check if the invariants are strong enough to support queries
  - Check if the invariants can be maintained under updates

- The process is similarly to suitcase packing:
  1. We place our stuff in the suitcase
  2. We check if the lid can be closed

- When everything fits inside the suitcase, we are done!
Designing Invariants and Dynamic Data Structures

- Dynamic data structure and invariant design follows a cycle:
  - Observe properties of the problem
  - Formulate invariants
  - Check if the invariants are strong enough to support queries
  - Check if the invariants can be maintained under updates
- The process is similarly to suitcase packing:
  1. We place our stuff in the suitcase
  2. We check if the lid can be closed
- When everything fits inside the suitcase, we are done!
Outline

- Introduction to Algorithms and Data Structures
- Implicit Working-Set Dictionaries
- Skyline Queries
- Catenable Priority Queues with Attrition
- Conclusion
Implicit Model
Implicit Model

- All operations from the RAM
- It is not allowed to *create* words, only to *move* them
- All \( n \) words have to be in contiguous positions

- Often it is assumed that all elements are distinct
- Fundamental trick: encode a bit in a pair of adjacent and distinct elements
Implicit Model

- All operations from the RAM
- It is not allowed to *create* words, only to *move* them
- All $n$ words have to be in contiguous positions

![Diagram showing contiguous positions of words](image)

- Often it is assumed that all elements are distinct
- Fundamental trick: encode a bit in a pair of adjacent and distinct elements
Implicit Model

- All operations from the RAM
- It is not allowed to *create* words, only to *move* them
- All $n$ words have to be in contiguous positions

Often it is assumed that all elements are distinct

**Fundamental trick:** encode a bit in a pair of adjacent and distinct elements

\[
\begin{align*}
    b &= \begin{cases} 
    0 & \text{if } x = \min(x, y) \\
    1 & \text{if } x = \max(x, y) 
    \end{cases} 
\end{align*}
\]
The Working-Set Property
The Working-Set Property

\[ l_x : \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ a & b & c & d & e & f \end{array} \]
The Working-Set Property

\[ l_x : \begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
 a & b & c & d & e & f \\
\end{array} \]
The Working-Set Property

- Element $x$ has a working-set number of $l_x$ iff:
  $l_x$ elements different from $x$ have been searched for since we last searched for $x$

\[
\begin{array}{cccccc}
  l_x : & 0 & 1 & 2 & 3 & 4 & 5 \\
  a & b & c & \textbf{d} & e & f \\
\end{array}
\]

- An Implicit Dictionary with the Working-Set Property:
  - Insert($x$): insert element $x$ into the dictionary and set $l_x = 0$
  - Delete($x$): delete element $x$ from the dictionary
  - Search($x$): determine if $x$ is in the dictionary and set $l_x = 0$
  - Predecessor($x$): find the address of the predecessor of $x$
  - Successor($x$): find the address of the successor of $x$
# Previous and Our Results

<table>
<thead>
<tr>
<th>Ref.</th>
<th>WS prop.</th>
<th>Insert/Delete(e)</th>
<th>Search(e)</th>
<th>Predecessor/Successor(e)</th>
<th>Additional Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1989</td>
<td>-</td>
<td>$O(\log^2 n)$</td>
<td>$O(\log^2 n)$</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>FGMP2002</td>
<td>-</td>
<td>$O(\log^2 n / \log \log n)$</td>
<td>$O(\log^2 n / \log \log n)$</td>
<td></td>
<td>None</td>
</tr>
<tr>
<td>FG2006</td>
<td>-</td>
<td>$O(\log n)$ amor.</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>None</td>
</tr>
<tr>
<td>FG2003</td>
<td>-</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>None</td>
</tr>
<tr>
<td>I2001</td>
<td>+</td>
<td>$O(\log n)$</td>
<td>$O(\log l_{e^*})$</td>
<td>$O(\log l_{e^*})$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>BHM2009</td>
<td>+</td>
<td>$O(\log n)$</td>
<td>$O(\log l_{e^*})$ exp.</td>
<td>$O(\log n)$</td>
<td>$O(\log \log n)$</td>
</tr>
<tr>
<td>BHM2009</td>
<td>+</td>
<td>$O(\log n)$</td>
<td>$O(\log l_{e^*})$ exp.</td>
<td>$O(\log l_{e^*})$ exp.</td>
<td>$O(\sqrt{n})$</td>
</tr>
<tr>
<td>BKT2010</td>
<td>+</td>
<td>$O(\log n)$</td>
<td>$O(\log l_{e^*})$</td>
<td>$O(\log n)$</td>
<td>None</td>
</tr>
<tr>
<td>BK2011</td>
<td>+</td>
<td>$O(\log n)$</td>
<td>$O(\log \min(l_{p(e)}, l_{e^*}, l_s))$</td>
<td>$O(\log l_{e^*})$</td>
<td>None</td>
</tr>
</tbody>
</table>

*e* is the predecessor/successor of e
Implicit Moveable Dictionaries

\[ i \quad \text{gray} \quad j \]
Implicit Moveable Dictionaries

- A dictionary laid out in memory addresses \([i,j]\)

\[
\begin{array}{c}
\text{i} \\
\end{array}
\quad \begin{array}{c}
\text{j} \\
\end{array}
\]

- **Interface:**
  - Insert-left/right\((e)\): insert element \(e\) into the dictionary which grows to the left/right
  - Delete-left/right\((e)\): delete element \(e\) from the dictionary which shrinks from the left/right
  - Search\((e)\): finds the address of \(e\) if \(e\) is in the dictionary
  - Predecessor\((e)\): finds the address of the predecessor of \(e\)
  - Successor\((e)\): finds the address of the successor of \(e\)
  - Can be constructed from \(O(1)\) FG dictionaries used as black boxes
Implicit Moveable Dictionaries

- A dictionary laid out in memory addresses $[i,j]$.

- **Interface:**
  - Insert-left/right($e$): insert element $e$ into the dictionary which grows to the left/right.
  - Delete-left/right($e$): delete element $e$ from the dictionary which shrinks from the left/right.
  - Search($e$): finds the address of $e$ if $e$ is in the dictionary.
  - Predecessor($e$): finds the address of the predecessor of $e$.
  - Successor($e$): finds the address of the successor of $e$.
  - Can be constructed from $O(1)$ FG dictionaries used as black boxes.
Implicit Moveable Dictionaries

- A dictionary laid out in memory addresses \([i,j]\)

\[
\begin{array}{c}
\text{i} \\
\text{j}
\end{array}
\]

- **Interface:**
  - Insert-left/right\((e)\): insert element \(e\) into the dictionary which grows to the left/right
  - Delete-left/right\((e)\): delete element \(e\) from the dictionary which shrinks from the left/right
  - Search\((e)\): finds the address of \(e\) if \(e\) is in the dictionary
  - Predecessor\((e)\): finds the address of the predecessor of \(e\)
  - Successor\((e)\): finds the address of the successor of \(e\)
  - Can be constructed from \(O(1)\) FG dictionaries used as black boxes
Implicit Working-Set Dictionaries

$|B_i| = \Theta(2^{i+k})$

$m = O(\log \log n)$
Implicit Working-Set Dictionaries

- Exponential layout

\[ |B_i| = \Theta(2^{2^{i+k}}) \]

- \( B_i \) consists of \( O(1) \) moveable dictionaries

- All elements \( e \) in \( B_i \) have \( l_e \geq 2^{2^{i-1+k}} \) or \( l_e \geq 2^{2^{i+k}} \)

- Searched and inserted elements are moved into \( B_0 \) (overflows)

- These are the ideas we used in the ISAAC 2010 paper

- Only gives \( O(\log n) \) bounds for predecessor and successor searches as all \( B_i \) have to be searched: the invariants do not relate \( e \) to its prede/suc-cessor

\[ m = O(\log \log n) \]
Implicit Working-Set Dictionaries

- Exponential layout

- $B_i$ consists of $O(1)$ moveable dictionaries
- All elements $e \in B_i$ have $l_e \geq 2 \cdot 2^{i-1+k}$ or $l_e \geq 2 \cdot 2^{i+k}$
- Searched and inserted elements are moved into $B_0$ (overflows)
- These are the ideas we used in the ISAAC 2010 paper
- Only gives $O(\log n)$ bounds for predecessor and successor searches as all $B_i$ have to be searched: the invariants do not relate $e$ to its predecessor

<table>
<thead>
<tr>
<th>$B_0$</th>
<th>$B_1$</th>
<th>$B_2$</th>
<th>...</th>
<th>$B_i$</th>
<th>...</th>
<th>$B_{m-1}$</th>
<th>$B_m$</th>
</tr>
</thead>
</table>

Summary of Invariants

- Blocks of fixed size: easy word/pointer encoding
- Elements in each block are divided according to working-set number
Implicit Predecessor/Successor Working-Set Dictionaries

- Intervals to solve the predecessor and successor problems
## Implicit Predecessor/Successor
### Working-Set Dictionaries

- Intervals to solve the predecessor and successor problems

<table>
<thead>
<tr>
<th>$B_m$</th>
<th>⋮ ⋮ ⋮ ⋮ ⋮</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{m-1}$</td>
<td>⋮ ⋮ ⋮ ⋮</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
</tr>
<tr>
<td>$B_2$</td>
<td>⋮ ⋮ ⋮</td>
</tr>
<tr>
<td>$B_1$</td>
<td>⋮ ⋮ ⋮</td>
</tr>
<tr>
<td>$B_0$</td>
<td>⋮ ⋮ ⋮ ⋮</td>
</tr>
</tbody>
</table>

---

Casper Kejlberg-Rasmussen

[madalgo]

15/36
Implicit Predecessor/Successor Working-Set Dictionaries

- Intervals to solve the predecessor and successor problems

\[
\begin{array}{c|c|c|c}
\hline
B_m & \cdots & \cdots & \cdots \\
\hline
B_{m-1} & \cdots & \cdots & \cdots \\
\vdots & \cdots & \cdots & \cdots \\
B_2 & \cdots & \cdots & \cdots \\
B_1 & \cdots & \cdots & \cdots \\
B_0 & \cdots & \cdots & \cdots \\
\hline
\end{array}
\]

- Divide the key-space into mutually disjoint intervals aligned with the points/elements
- Invariant: any point/element, intersecting an interval at level \( i \), lies in block \( B_i \)
- Predecessor/Successor(e) searches can terminate when an interval at level \( i \) is intersected
Divide the key-space into mutually disjoint intervals aligned with the points/elements.

Invariant: any point/element, intersecting an interval at level $i$, lies in block $B_i$.

Predecessor/Successor($e$) searches can terminate when an interval at level $i$ is intersected.
Implicit Predecessor/Successor Working-Set Dictionaries

- Intervals to solve the predecessor and successor problems

- Divide the key-space into mutually disjoint intervals aligned with the points/elements
- Invariant: any point/element, intersecting an interval at level $i$, lies in block $B_i$
- Predecessor/Successor($e$) searches can terminate when an interval at level $i$ is intersected
Implicit Representation of Intervals

- Representing the intervals implicitly

![Diagram showing intervals and related parameters]

- $B_0, B_{m-1}, B_m$
- $D_i, A_i, R_i, W_i, H_i, C_i, G_i, B_{i-1}, B_{i+1}$
- $l_e \geq 2^{i-1+k}$
- $l_e \geq 2^{i+k}$
- $l_e \geq 2^{\max(i,j)-1+k}$
Implicit Representation of Intervals

- Representing the intervals implicitly
Implicit Representation of Intervals

- Representing the intervals implicitly

\[ B_m, B_{m-1}, \ldots, B_1, B_0 \]

\[ B_{i-1}, D_i, A_i, R_i, W_i, H_i, C_i, G_i, B_{i+1} \]

- Arriving: \( l_e \geq 2^{i-1+k} \)
- Resting: \( l_e \geq 2^{i+k} \)
- Waiting: \( l_e \geq 2^{j+k} \)
- Helping: \( l_e \geq 2^{\max(i,j)-1+k} \)
- Climbing
- Guarding

Casper Kejlberg-Rasmussen

Danmarks Grundforskningsfond
Danish National Research Foundation

MADELGO
CENTER FOR MASSIVE DATA ALGORITHMS

AARHUS UNIVERSITY

16/36
Implicit Representation of Intervals

- Representing the intervals implicitly

\[ B_0, B_1, \ldots, B_{m-1}, B_m \]

- Representing the intervals implicitly

\[ B_{i-1} D_i A_i R_i W_i H_i C_i G_i B_{i+1} \]

- Representing the intervals implicitly

\[ \text{Arriving, Resting, Waiting, Helping, Climbing, Guarding} \]

\[ l_e \geq 2^{i-1+k}, l_e \geq 2^{i+k}, l_e \geq 2^{\max(i,j)-1+k} \]
Implicit Representation of Intervals

- Representing the intervals implicitly
Implicit Representation of Intervals

- Representing the intervals implicitly

\[ B_m \]
\[ B_{m-1} \]
\[ \vdots \]
\[ B_2 \]
\[ B_1 \]
\[ B_0 \]

\[ B_{i-1} \quad D_i \quad A_i \quad R_i \quad W_i \quad H_i \quad C_i \quad G_i \quad B_{i+1} \]

Arriving: \[ l_e \geq 2^{i-1+k} \]
Resting: \[ l_e \geq 2^{i+k} \]
Helping: \[ l_e \geq 2^{\max(i,j)-1+k} \]
Guarding: \[ l_e \]
Implicit Representation of Intervals

- Representing the intervals implicitly

\[
B_0, B_1, B_{m-1}, B_m
\]

\[
B_{i-1}, D_i, A_i, R_i, W_i, H_i, C_i, G_i, B_{i+1}
\]

- \( l_e \geq 2^{i-1+k} \)
- \( l_e \geq 2^{i+k} \)
- \( l_e \geq 2^{\max(i,j)-1+k} \)
Implicit Representation of Intervals

- Representing the intervals implicitly

Summary of Invariants

- Blocks of fixed size: easy word/pointer encoding
- Elements in each block are divided into types according to their working-set number and
- According to the types of neighboring elements
Outline

- Introduction to Algorithms and Data Structures
- Implicit Working-Set Dictionaries
- Skyline Queries
- Catenable Priority Queues with Attrition
- Conclusion
What are Skyline Queries?
What are Skyline Queries?

![Diagram of Skyline Queries](image)
What are Skyline Queries?

- Given two points $p, q \in P \subseteq \mathbb{R}^2$ we say $p$ dominates $q$ iff $p_x \geq q_x$ and $p_y \geq q_y$

- The *maximal/skyline* points of a point set $P \subseteq \mathbb{R}^2$ are the undominated points

- Given a *dynamic* point set $P \subseteq \mathbb{R}^2$ we want to be able to find the skyline for a given query range $Q = [x_1, x_2] \times [y_1, y_2]$
What are Skyline Queries?

- Given two points $p, q \in P \subseteq \mathbb{R}^2$ we say $p$ dominates $q$ iff $p_x \geq q_x$ and $p_y \geq q_y$

- The maximal/skyline points of a point set $P \subseteq \mathbb{R}^2$ are the undominated points.

- Given a dynamic point set $P \subseteq \mathbb{R}^2$ we want to be able to find the skyline for a given query range $Q = [x_1, x_2] \times [y_1, y_2]$.
Special Cases of Skyline Queries

$x_1 = y_1 = -\infty \quad x_2 = y_2 = \infty$
Special Cases of Skyline Queries

Skyline

Top-Open

\[ x_1 = y_1 = -\infty \quad x_2 = y_2 = \infty \]

\[ y_2 = \infty \]
Special Cases of Skyline Queries

Skyline

$x_1 = y_1 = -\infty$

$y_2 = \infty$

Top-Open

$x_2 = y_2 = \infty$

$x_2 = \infty$

Right-Open
Special Cases of Skyline Queries

**Skyline**

$x_1 = y_1 = -\infty \quad x_2 = y_2 = \infty$

**Top-Open**

$y_2 = \infty$

**Right-Open**

$x_2 = \infty$

**Bottom-Open**

$y_1 = -\infty$
Special Cases of Skyline Queries

- **Skyline**: $x_1 = y_1 = -\infty$ and $x_2 = y_2 = \infty$
- **Top-Open**: $y_2 = \infty$
- **Right-Open**: $x_2 = \infty$
- **Bottom-Open**: $y_1 = -\infty$
- **Left-Open**: $x_1 = -\infty$
Special Cases of Skyline Queries

- **Skyline**: $x_1 = y_1 = -\infty$, $x_2 = y_2 = \infty$

- **Top-Open**: $y_2 = \infty$

- **Right-Open**: $x_2 = \infty$

- **Bottom-Open**: $y_1 = -\infty$

- **Left-Open**: $x_1 = -\infty$

- **Dominance**: $x_2 = y_2 = \infty$
Special Cases of Skyline Queries

Skyline

Top-Open

Right-Open

Bottom-Open

Left-Open

Dominance

Anti-Dominance

$x_1 = y_1 = -\infty$, $x_2 = y_2 = \infty$

$y_2 = \infty$

$x_2 = \infty$

$y_1 = -\infty$

$x_1 = -\infty$

$x_2 = y_2 = \infty$

$x_1 = y_1 = -\infty$
Special Cases of Skyline Queries

- **Skyline**
  - $x_1 = y_1 = -\infty$
  - $x_2 = y_2 = \infty$

- **Top-Open**
  - $y_2 = \infty$

- **Right-Open**
  - $x_2 = \infty$

- **Bottom-Open**
  - $y_1 = -\infty$

- **Left-Open**
  - $x_1 = -\infty$

- **Dominance**
  - $x_2 = y_2 = \infty$

- **Anti-Dominance**
  - $x_1 = y_1 = -\infty$

- **Contour**
  - $x_1 = y_1 = -\infty$
  - $y_2 = \infty$
EM Model

CPU

Memory
EM Model

CPU

Memory

RAM Model
EM Model

CPU \quad Memory \quad \text{Hard disk}

Count disk accesses

B

M
EM Model

- We count the number of disk accesses, not CPU instructions.
- When reading/writing from/to disk, we can access $B$ consecutive elements in one I/O.
- Our algorithms should spend $O(1/B)$ I/Os to access one element.
- Scanning uses $O(n/B)$ I/Os and search trees uses $O(\log_B n)$ I/Os.
## Previous and Our Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Space</th>
<th>Pre-proces</th>
<th>Query</th>
<th>Query</th>
<th>Update</th>
<th>Domain</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-Open</td>
<td>O(n)</td>
<td>O(n log n)</td>
<td>O(1+k)</td>
<td>-</td>
<td>-</td>
<td>R</td>
<td>YA10</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n)</td>
<td>O(n log n)</td>
<td>O(log n+k)</td>
<td>-</td>
<td>O(log n)</td>
<td>R</td>
<td>BT11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n)</td>
<td>O(n log n/log log n)</td>
<td>O(log n/log log n+k)</td>
<td>-</td>
<td>O(log n/log log n)</td>
<td>R</td>
<td>BT11</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>O(log n+k)</td>
<td>-</td>
<td>-</td>
<td>R</td>
<td>KDKS11</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n log n/log log n)</td>
<td>O(n log n/log log n)</td>
<td>O(n log n/log log n+k)</td>
<td>-</td>
<td>-</td>
<td>Rank</td>
<td>GKASK97</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>O(log^2 n+k)</td>
<td>-</td>
<td>O(log^2 n)</td>
<td>R</td>
<td>BT11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(n/B log_{M/B} n/B)</td>
<td>O(n/B)</td>
<td>-</td>
<td>-</td>
<td>R</td>
<td>PTFS05</td>
</tr>
<tr>
<td>Top-Open</td>
<td>Heuristics, various update types: HKIT06, PTFS05, TO06, WAEA07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ST11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(n/B*)</td>
<td>O(log_B n+k/B)</td>
<td>-</td>
<td>-</td>
<td>R</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(n/B*)</td>
<td>O(loglog_B U+k/B)</td>
<td>-</td>
<td>-</td>
<td>U</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(n/B*)</td>
<td>O(1+k/B)</td>
<td>-</td>
<td>-</td>
<td>Rank</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n/B)</td>
<td>O(n/B*)</td>
<td>O((n/B)^ε+k/B)</td>
<td>Ω((n/B)^ε+k/B)</td>
<td>-</td>
<td>R</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(n/B*)</td>
<td>O(log_{26^n} n/B+k/B^{εε})</td>
<td>-</td>
<td>O(log_{26^n} n/B)</td>
<td>R</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n/B)</td>
<td>O(n/B*)</td>
<td>O((n/B)^ε+k/B)</td>
<td>Ω((n/B)^ε+k/B)</td>
<td>O(log n/B)</td>
<td>R</td>
<td>KTTTY13</td>
</tr>
</tbody>
</table>

* Assumes pre-sorting
## Previous and Our Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Space</th>
<th>Pre-proces</th>
<th>Query</th>
<th>Query</th>
<th>Update</th>
<th>Domain</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-Open</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
<td>$O(1+k)$</td>
<td>-</td>
<td>-</td>
<td>ℝ</td>
<td>YA10</td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
<td>$O(\log n+k)$</td>
<td>-</td>
<td>$O(\log n)$</td>
<td>ℝ</td>
<td>BT11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n)$</td>
<td>$O(n \log n/\log \log \log n)$</td>
<td>$O(\log n/\log \log n+k)$</td>
<td>-</td>
<td>$O(\log n/\log n)$</td>
<td>ℝ</td>
<td>BT11</td>
</tr>
<tr>
<td>4-sided</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(\log n+k)$</td>
<td>-</td>
<td>-</td>
<td>ℝ</td>
<td>KDKS11</td>
</tr>
<tr>
<td>4-sided</td>
<td>$O(n \log n/\log \log n)$</td>
<td>$O(n \log n/\log \log n)$</td>
<td>$O(n \log n/\log \log n+k)$</td>
<td>-</td>
<td>-</td>
<td>Rank</td>
<td>GKASK97</td>
</tr>
<tr>
<td>4-sided</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(\log^2 n+k)$</td>
<td>-</td>
<td>$O(\log^2 n)$</td>
<td>ℝ</td>
<td>BT11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n/B)$</td>
<td>$O(n/B \log_{M/B} n/B)$</td>
<td>$O(n/B)$</td>
<td>-</td>
<td>-</td>
<td>ℝ</td>
<td>PTFS05 ST11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>Heuristics, various update types: HKIT06, PTFS05, TO06, WAEA07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ℝ</td>
<td></td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n/B)$</td>
<td>$O(n/B^*)$</td>
<td>$O(\log_B n+k/B)$</td>
<td>$\Omega((n/B)^\epsilon+k/B)$</td>
<td>$\Omega((n/B)^\epsilon+k/B)$</td>
<td>ℝ</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n/B)$</td>
<td></td>
<td>$O((n/B)^{\epsilon}+k/B)$</td>
<td>$\Omega((n/B)^{\epsilon}+k/B)$</td>
<td>$\Omega((n/B)^{\epsilon}+k/B)$</td>
<td>U</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n/B)$</td>
<td></td>
<td>$\Omega(\log_{2B^\epsilon} n/B+k/B^\epsilon)$</td>
<td>-</td>
<td>$O(\log_{2B^\epsilon} n/B)$</td>
<td>ℝ</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>4-sided</td>
<td>$O(n/B)$</td>
<td></td>
<td>$O((n/B)^{\epsilon}+k/B)$</td>
<td>$\Omega((n/B)^{\epsilon}+k/B)$</td>
<td>$O(\log n/B)$</td>
<td>ℝ</td>
<td>KTTTY13</td>
</tr>
</tbody>
</table>

* Assumes pre-sorting

**Indexability Model**

**Indivisibility Assumption**
## Previous and Our Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Space</th>
<th>Pre-proce</th>
<th>Query</th>
<th>Query</th>
<th>Update</th>
<th>Domain</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-Open</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
<td>$O(1+k)$</td>
<td>-</td>
<td>-</td>
<td>$\mathbb{R}$</td>
<td>YA10</td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
<td>$O(\log n+k)$</td>
<td>-</td>
<td>$O(\log n)$</td>
<td>$\mathbb{R}$</td>
<td>BT11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n)$</td>
<td>$O(n \log n/\log \log n)$</td>
<td>$O(\log n/\log \log n+k)$</td>
<td>-</td>
<td>$O(\log n/\log n)$</td>
<td>$\mathbb{R}$</td>
<td>BT11</td>
</tr>
<tr>
<td>4-sided</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(\log n+k)$</td>
<td>-</td>
<td>-</td>
<td>$\mathbb{R}$</td>
<td>KDKS11</td>
</tr>
<tr>
<td>4-sided</td>
<td>$O(n \log n/\log \log n)$</td>
<td>$O(n \log n/\log \log n)$</td>
<td>$O(n \log n/\log \log n+k)$</td>
<td>-</td>
<td>-</td>
<td>Rank</td>
<td>GKASK97</td>
</tr>
<tr>
<td>4-sided</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(\log^2n+k)$</td>
<td>-</td>
<td>$O(\log^2n)$</td>
<td>$\mathbb{R}$</td>
<td>BT11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>$O(n/B)$</td>
<td>$O(n/B \log_{M/B} n/B)$</td>
<td>$O(n/B)$</td>
<td>-</td>
<td>-</td>
<td>$\mathbb{R}$</td>
<td>PTFS05 ST11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>Heuristics, various update types: HKIT06, PTFS05, TO06, WAEA07</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\mathbb{R}$</td>
<td></td>
</tr>
</tbody>
</table>

- **4-sided**
- **21/36**

* Assumes pre-sorting

**Casper Kejlberg-Rasmussen**

---

Danmarks Grundforskningsfond
Danes National Research Foundation

**madalgo**

CENTER FOR MASSIVE DATA ALGORITHMS

AARHUS UNIVERSITY
## Previous and Our Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Space</th>
<th>Pre-proces</th>
<th>Query</th>
<th>Update</th>
<th>Domain</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-Open</td>
<td>O(n)</td>
<td>O(n log n)</td>
<td>O(1+k)</td>
<td>-</td>
<td>-</td>
<td>YA10</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n)</td>
<td>O(n log n)</td>
<td>O(log n+k)</td>
<td>O(log n)</td>
<td>R</td>
<td>BT11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n)</td>
<td>O(n log n/log log n)</td>
<td>O(log n/log n+k)</td>
<td>-</td>
<td>O(log n/log n)</td>
<td>R</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>O(log n+k)</td>
<td>-</td>
<td>-</td>
<td>KDKS11</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n log n/log log n)</td>
<td>O(n log n/log log n)</td>
<td>O(n log n/log n+k)</td>
<td>-</td>
<td>-</td>
<td>Rank</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
<td>O(log^2n+k)</td>
<td>-</td>
<td>O(log^2n)</td>
<td>R</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(n/B log_{M/B} n/B)</td>
<td>O(n/B)</td>
<td>-</td>
<td>-</td>
<td>PTFS05 ST11</td>
</tr>
<tr>
<td>Top-Open</td>
<td>Heuristics, various update types: HKIT06, PTFS05, TO06, WAEA07</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(log_B n+k/B)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(loglog_B U+k/B)</td>
<td>-</td>
<td>-</td>
<td>U</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O(1+k/B)</td>
<td>-</td>
<td>-</td>
<td>Rank</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n/B)</td>
<td>O((n/B)^{ε}+k/B)</td>
<td>Ω((n/B)^{ε}+k/B)</td>
<td>-</td>
<td>-</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>Top-Open</td>
<td>O(n/B)</td>
<td>O((n/B)^{ε}+k/B)</td>
<td>Ω((n/B)^{ε}+k/B)</td>
<td>O(log n/B)</td>
<td>-</td>
<td>KTTTY13</td>
</tr>
<tr>
<td>4-sided</td>
<td>O(n/B)</td>
<td>O((n/B)^{ε}+k/B)</td>
<td>Ω((n/B)^{ε}+k/B)</td>
<td>O(log n/B)</td>
<td>-</td>
<td>KTTTY13</td>
</tr>
</tbody>
</table>

* Assumes pre-sorting

For Today! 21/36
Observation for Top-Open Queries
Observation for Top-Open Queries

- Consider mirroring the point set in the $y$-axis.
- Let $x$ represent the insertion time and $y$ the key space.
- When inserting element $e$ it deletes/attributes all elements inserted before it with a larger key.
- Non-attrited points and undominated points are equivalent.
Observation for Top-Open Queries

- Consider mirroring the point set in the \( y \)-axis.
- Let \( x \) represent the insertion time and \( y \) the key space.
- When inserting element \( e \) it deletes/attrites all elements inserted before it with a larger key.
- Non-attrited points and undominated points are equivalent.
Observation for Top-Open Queries

- Consider mirroring the point set in the $y$-axis
- Let $x$ represent the insertion time and $y$ the key space
- When inserting element $e$ it deletes/attrites all elements inserted before it with a larger key
- Non-attrited points and undominated points are equivalent
Priority Queues with Attrition

Max

Key

Min

Head

Time

Tail
Priority Queues with Attrition

- **DeleteMin()**: Deletes the head/minimum element of the queue
- **InsertAndAttrite(e)**: Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- **ConcatenateAndAttrite(Q_1, Q_2)**: Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
Priority Queues with Attrition

- **DeleteMin()**: Deletes the head/minimum element of the queue
- **InsertAndAttrite(e)**: Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- **ConcatenateAndAttrite(Q₁, Q₂)**: Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂
Priority Queues with Attrition

- **DeleteMin():** Deletes the head/minimum element of the queue
- **InsertAndAttrite(e):** Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- **ConcatenateAndAttrite(Q₁, Q₂):** Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂
Priority Queues with Attrition

- **DeleteMin()**: Deletes the head/minimum element of the queue
- **InsertAndAttrite(e)**: Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- **ConcatenateAndAttrite(Q_1, Q_2)**: Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
Data Structure and Invariants

PQAs of size $[B, 2B]$

Fanout $[2B^\varepsilon, 4B^\varepsilon]$

PQA Buffer size $O(B^{1-\varepsilon})$

PQAs of size $[B, 2B]$
Data Structure and Invariants

- A \((2B^\epsilon,4B^\epsilon)\)-tree augmented with PQAs
- Internal node have between \(2B^\epsilon\) and \(4B^\epsilon\) children and stores a PQA which is the concatenation of its childrens PQAs
- Leaf stores a PQA over the \(B\) to \(2B\) elements it contains
- Updating element \(e\) discards the PQAs on the path to \(e\) and rebuilds them again

\[\text{Fanout } [2B^\epsilon,4B^\epsilon]\]

\[\text{PQA Buffer size } O(B^{1-\epsilon})\]
Data Structure and Invariants

- A $(2B^\varepsilon, 4B^\varepsilon)$-tree augmented with PQAs
- Internal node have between $2B^\varepsilon$ and $4B^\varepsilon$ children and stores a PQA which is the concatenation of its childrens PQAs
- Leaf stores a PQA over the $B$ to $2B$ elements it contains
- Updating element $e$ discards the PQAs on the path to $e$ and rebuilds them again

PQA Buffer size $O(B^{1-\varepsilon})$

Fanout $[2B^\varepsilon, 4B^\varepsilon]$

PQAs of size $[B, 2B]$
Data Structure and Invariants

- A $(2B^\varepsilon, 4B^\varepsilon)$-tree augmented with PQAs
- Internal node have between $2B^\varepsilon$ and $4B^\varepsilon$ children and stores a PQA which is the concatenation of its childrens PQAs
- Leaf stores a PQA over the $B$ to $2B$ elements it contains
- Updating element $e$ discards the PQAs on the path to $e$ and rebuilds them again
Data Structure and Invariants

• A \((2B^\varepsilon, 4B^\varepsilon)\)-tree augmented with PQAs

• Internal node have between \(2B^\varepsilon\) and \(4B^\varepsilon\) children and stores a PQA which is the concatenation of its children’s PQAs.

• Leaf nodes store a PQA over the \(B\) to \(2B\) elements it contains.

• Updating element \(e\) discards the PQAs on the path to \(e\) and rebuilds them again.

Summary of Invariants

- Nodes have a bounded degree
- All leaves have the same depth
- Nodes are augmented with PQAs
Top-Open Skyline Queries
Top-Open Skyline Queries

- We find the leaves of $x_1$ and $x_2$ and make two PQAs of the elements within $[x_1, x_2]$ called $Q_1$ and $Q_2$

- We concatenate $Q_1$, all PQAs of subtrees inside $[x_1, x_2]$ and $Q_2$ into one PQA $Q$ (Divide and conquer)

- We call DeleteMin on $Q$ and report the returned element $e$ unless $e$ has $y$-value larger than $-y$
Top-Open Skyline Queries

- We find the leafs of $x_1$ and $x_2$ and make two PQAs of the elements within $[x_1, x_2]$ called $Q_1$ and $Q_2$.

- We concatenate $Q_1$, all PQAs of subtrees inside $[x_1, x_2]$ and $Q_2$ into one PQA $Q$ (Divide and conquer).

- We call DeleteMin on $Q$ and report the returned element $e$ unless $e$ has $y$-value larger than $-y$. 
Top-Open Skyline Queries

- We find the leaves of $x_1$ and $x_2$ and make two PQAs of the elements within $[x_1, x_2]$ called $Q_1$ and $Q_2$

- We concatenate $Q_1$, all PQAs of subtrees inside $[x_1, x_2]$ and $Q_2$ into one PQA $Q$ (Divide and conquer)

- We call DeleteMin on $Q$ and report the returned element $e$ unless $e$ has $y$-value larger than $-y$
Outline

- Introduction to Algorithms and Data Structures
- Implicit Working-Set Dictionaries
- Skyline Queries
- Catenable Priority Queues with Attrition
- Conclusion
## Previous and Our Results

<table>
<thead>
<tr>
<th>Authors</th>
<th>Find-Min</th>
<th>Delete-Min</th>
<th>Insert-And-Attrite</th>
<th>Catenate-And-Attrite</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun89</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td></td>
<td>PM</td>
</tr>
<tr>
<td>Sun89</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td></td>
<td>EM</td>
</tr>
<tr>
<td>KTTTY13</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>PM</td>
</tr>
<tr>
<td>KTTTY13</td>
<td>$O(1/B)$</td>
<td>$O(1/B)$</td>
<td>$O(1/B)$</td>
<td>$O(1/B)$</td>
<td>EM</td>
</tr>
</tbody>
</table>
Previous and Our Results

<table>
<thead>
<tr>
<th>Authors</th>
<th>Find-Min</th>
<th>Delete-Min</th>
<th>Insert-And-Attrite</th>
<th>Catenate-And-Attrite</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun89</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td></td>
<td>PM</td>
</tr>
<tr>
<td>Sun89</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td></td>
<td>EM</td>
</tr>
<tr>
<td>KTTTY13</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>PM</td>
</tr>
<tr>
<td>KTTTY13</td>
<td>O(1/B)</td>
<td>O(1/B)</td>
<td>O(1/B)</td>
<td>O(1/B)</td>
<td>EM</td>
</tr>
</tbody>
</table>

- In External Memory our O(1/B) I/O bounds assume that for $k$ PQAs we keep O(1) blocks in memory for each PQA
- Hence we require that $M = \Omega(kB)$ when maintaining $k$ PQAs
- We can concatenate an arbitrary number of PQAs into one in O(1) I/Os if we maintain an extra invariant
Observations and Invariants

- **Max**
- **Key**
- **Min**

- **Head**
- **Time**
- **Tail**
Observations and Invariants

- InsertAndAttrite(e): Inserts $e$ and deletes/attrites all elements before $e$ with a key larger or equal to $e$
- ConcatenateAndAttrite($Q_1, Q_2$): Deletes/attrites all elements in $Q_1$ with a key larger or equal to $\min(Q_2)$ and appends $Q_2$
- DeleteMin($Q$): Deletes the head/minimum element $e$ of the queue $Q$
Observations and Invariants

- **InsertAndAttrite(e):** Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- **ConcatenateAndAttrite(Q₁, Q₂):** Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂
- **DeleteMin(Q):** Deletes the head/minimum element e of the queue Q
Observations and Invariants

- **InsertAndAttrite(e):** Inserts e and deletes/attrites all elements before e with a key larger or equal to e.
- **ConcatenateAndAttrite(Q₁,Q₂):** Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂.
- **DeleteMin(Q):** Deletes the head/minimum element e of the queue Q.
Observations and Invariants

- **InsertAndAttrite(e)**: Inserts $e$ and deletes/attrites all elements before $e$ with a key larger or equal to $e$

- **ConcatenateAndAttrite($Q_1, Q_2$)**: Deletes/attrites all elements in $Q_1$ with a key larger or equal to $\min(Q_2)$ and appends $Q_2$

- **DeleteMin($Q$)**: Deletes the head/minimum element $e$ of the queue $Q$
Observations and Invariants

- **InsertAndAttrite(e):** Inserts $e$ and deletes/attrites all elements before $e$ with a key larger or equal to $e$
- **ConcatenateAndAttrite($Q_1, Q_2$):** Deletes/attrites all elements in $Q_1$ with a key larger or equal to $\text{min}(Q_2)$ and appends $Q_2$
- **DeleteMin($Q$):** Deletes the head/minimum element $e$ of the queue $Q$
Observations and Invariants

- **InsertAndAttrite(e)**: Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- **ConcatenateAndAttrite(Q₁,Q₂)**: Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂
- **DeleteMin(Q)**: Deletes the head/minimum element e of the queue Q
Observations and Invariants

• InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e

• ConcatenateAndAttrite(Q₁,Q₂): Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂

• DeleteMin(Q): Deletes the head/minimum element e of the queue Q

Casper Kejlberg-Rasmussen
Observations and Invariants

- **InsertAndAttrite(e):** Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- **ConcatenateAndAttrite(Q₁,Q₂):** Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂
- **DeleteMin(Q):** Deletes the head/minimum element e of the queue Q
Observations and Invariants

- **InsertAndAttrite**\((e)\): Inserts \(e\) and deletes/attrites all elements before \(e\) with a key larger or equal to \(e\).

- **ConcatenateAndAttrite**\((Q_1, Q_2)\): Deletes/attrites all elements in \(Q_1\) with a key larger or equal to \(\min(Q_2)\) and appends \(Q_2\).

- **DeleteMin**\((Q)\): Deletes the head/minimum element \(e\) of the queue \(Q\).
Observations and Invariants

- InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- ConcatenateAndAttrite(Q₁,Q₂): Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂
- DeleteMin(Q): Deletes the head/minimum element e of the queue Q
Observations and Invariants

Summary of Invariants

- The clean queue holds non-attrited elements and is larger than the combined size of all dirty queues + #dirty queues
- The buffer queue holds both (non)-attrited elements
- The dirty queues might attrite into each other
- Each element in a dirty queue might contain another PQA

Observations:
- InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- ConcatenateAndAttrite(Q₁,Q₂): Deletes/attrites all elements in Q₁ with a key larger or equal to min(Q₂) and appends Q₂
- DeleteMin(Q): Deletes the head/minimum element e of the queue Q
Concatenable PQA: Data Structure
Concatenable PQA: Data Structure

- A PQA consists of $2+k_Q$ deques $C, B, D_1, ..., D_{k_Q}$ of records and buffers $F$ and $L$
- A record $r=(l,p)$ contains a buffer $l$ of $[b,4b]$ elements and a pointer $p$ to a PQA, if $p$ is nil then $r$ is simple
- A PQA is a tree of unbounded degree with PQAs as internal and leaf nodes
Concatenable PQA: Data Structure

- A PQA consists of $2+k_Q$ deques $C$, $B$, $D_1$, ..., $D_{k_Q}$ of records and buffers $F$ and $L$
- A record $r=(l,p)$ contains a buffer $l$ of $[b,4b]$ elements and a pointer $p$ to a PQA, if $p$ is nil then $r$ is simple
- A PQA is a tree of unbounded degree with PQAs as internal and leaf nodes
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\[ C(Q) \text{ and } B(Q) \text{ are simple} \]

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \quad |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

DeleteMin

\[ Q \]

\[ F \quad C \quad B \quad D_1 \quad D_{k_{Q_1}} \quad L \]
Invariants and Operations

\[\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1\]

\[C(Q) \text{ and } B(Q) \text{ are simple}\]

\[\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))\]

\[|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q\]

DeleteMin

\[\text{max}(C(Q)) < \text{min}(B(Q)) < \text{min}(D_1(Q)) < \text{min}(D_i(Q)), \text{ for } i > 1\]

\[C(Q) \text{ and } B(Q) \text{ are simple}\]

\[\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))\]

\[|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q\]
Invariants and Operations

\[
\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1
\]

\[
C(Q) \text{ and } B(Q) \text{ are simple}
\]

\[
\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))
\]

DeleteMin

\[
|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q
\]
Invariants and Operations

\[
\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1
\]

\[
C(Q) \text{ and } B(Q) \text{ are simple}
\]

\[
\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))
\]

DeleteMin

\[
|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q
\]
Invariants and Operations

\[
\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1
\]

\[C(Q) \text{ and } B(Q) \text{ are simple}\]

\[
\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))
\]

\[\left| C(Q) \right| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q\]

DeleteMin

Bias to the rescue!
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\[ C(Q) \text{ and } B(Q) \text{ are simple} \]

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

CatenateAndAttrite
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\( C(Q) \) and \( B(Q) \) are simple

\[ \max(F(Q)) < \min(C(Q)) < \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

CatenateAndAttrite

\( Q_2 \)

\[ \begin{array}{cccccc}
F & C & B & D_1 & D_k_{Q_2} & L \\
\end{array} \]

\( Q_1 \)

\[ \begin{array}{cccccc}
F & C & B & D_1 & D_k_{Q_1} & L \\
\end{array} \]

\( Q \)

\[ \begin{array}{cccccc}
B & D_1 & D_k_{Q_2} & ... & C & B & D_1 & D_k_{Q_2} & ... \\
\end{array} \]
Invariants and Operations

\[
\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1
\]

\[C(Q) \text{ and } B(Q) \text{ are simple}\]

\[
\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))
\]

\[
|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q
\]
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\( C(Q) \) and \( B(Q) \) are simple

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

CatenateAndAttrite

Bias to the rescue!
Invariants and Operations

\[
\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1
\]

\[C(Q) \text{ and } B(Q) \text{ are simple}\]

\[
\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))
\]

\[|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q\]

CatenateAndAttrite
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\[ C(Q) \text{ and } B(Q) \text{ are simple} \]

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\( C(Q) \) and \( B(Q) \) are simple

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\[ C(Q) \text{ and } B(Q) \text{ are simple} \]

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

CatenateAndAttrite

\[ F \quad C \quad B \quad D_1 \quad D_{k_Q_2} \]

\[ F \quad C \quad B \quad D_1 \quad D_{k_Q_1} \]

\[ F \quad C \quad B \quad D_1 \quad D_{k_Q_1+1} \]

\[ Q \]

Casper Kejlberg-Rasmussen

Danmarks Grundforskningsfond
Danish National Research Foundation

Madalgo
Center for Massive Data Algorithmics

Aarhus University

31/36
Invariants and Operations

\[
\max(C(Q)) \,<\, \min(B(Q)) \,<\, \min(D_1(Q)) \,<\, \min(D_i(Q)), \text{ for } i > 1
\]

\[
C(Q) \text{ and } B(Q) \text{ are simple}
\]

\[
\max(F(Q)) \,<\, \min(C(Q)) \quad \min(D_1(Q)) \,<\, \min(L(Q))
\]

\[
|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q
\]
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\[ C(Q) \text{ and } B(Q) \text{ are simple} \]

\[ \max(F(Q)) < \min(C(Q)) \quad \text{min}(D_1(Q)) < \text{min}(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

CatenateAndAttrite

Bias to the rescue!
Invariants and Operations

\[
\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1
\]

\(C(Q)\) and \(B(Q)\) are simple

\[
\max(F(Q)) < \min(C(Q)) < \min(D_1(Q)) < \min(L(Q))
\]

\[
|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q
\]

**Bias**

- \(B > 0\)
- \(B = 0\) and \(k_Q > 1\)
- \(B = 0\) and \(k_Q = 1\)
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\( C(Q) \) and \( B(Q) \) are simple

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

Bias

B > 0

B = 0 and \( k_Q > 1 \)

B = 0 and \( k_Q = 1 \)
Invariants and Operations

\[
\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1
\]

\[
C(Q) \text{ and } B(Q) \text{ are simple}
\]

\[
\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))
\]

\[
|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q
\]

**Bias**

B > 0

B = 0 and \(k_Q > 1\)

B = 0 and \(k_Q = 1\)

\[Q\]

\[F\]

\[C\]

\[B\]

\[D_1\]

\[D_{k_Q-1}\]

\[D_{k_Q}\]

\[L\]
Invariants and Operations

$$\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1$$

$$C(Q) \text{ and } B(Q) \text{ are simple}$$

$$\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))$$

$$|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

**Bias**

B > 0

B = 0 and k_Q > 1

B = 0 and k_Q = 1
**Invariants and Operations**

\[
\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1
\]

\[
C(Q) \text{ and } B(Q) \text{ are simple}
\]

\[
\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))
\]

\[
|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q
\]

**Bias**

- B > 0
- B = 0 and \(k_Q > 1\)
- B = 0 and \(k_Q = 1\)
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\( C(Q) \) and \( B(Q) \) are simple

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

**Bias**

B > 0

B = 0 and \( k_Q > 1 \)

B = 0 and \( k_Q = 1 \)
**Invariants and Operations**

$\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q))$, for $i > 1$

$C(Q)$ and $B(Q)$ are simple

$\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))$

$|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$

**Bias**

B > 0

B = 0 and $k_Q > 1$

B = 0 and $k_Q = 1$
Invariants and Operations

$$\max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1$$

$C(Q)$ and $B(Q)$ are simple

$$\max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q))$$

$$|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

Bias

B>0

B=0 and $k_Q>1$

B=0 and $k_Q=1$

![Diagram showing invariants and operations with sets F, C, D_1, and L.]
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \text{ for } i > 1 \]

\[ C(Q) \text{ and } B(Q) \text{ are simple} \]

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

**Bias**

B > 0

B = 0 and k_Q > 1

B = 0 and k_Q = 1
Invariants and Operations

\[ \max(C(Q)) < \min(B(Q)) < \min(D_1(Q)) < \min(D_i(Q)), \quad \text{for } i > 1 \]

\( C(Q) \) and \( B(Q) \) are simple

\[ \max(F(Q)) < \min(C(Q)) \quad \min(D_1(Q)) < \min(L(Q)) \]

\[ |C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q \]

**Bias**

B > 0

B = 0 and \( k_Q > 1 \)

B = 0 and \( k_Q = 1 \)
Outline

- Introduction to Algorithms and Data Structures
- Implicit Working-Set Dictionaries
- Skyline Queries
- Catenable Priority Queues with Attrition
- Conclusion
Conclusion
Conclusion

- We have seen how invariants are formed and used in dynamic data structures to give the three data structures:
  - Implicit Cache-Oblivious Working-Set Dictionaries
  - 2D Skyline Data Structures in External Memory
  - Catenable Priority Queues with Attrition in External Memory

- Open problems:
  - Can we change the insert($e$) operation of the working set dictionary so that $e$ gets a working set value of $n$ instead of 0?
  - In what other problems does attrition occur as a subproblem?
  - Can the PQA be modified to solve other skyline related problems like Top-k Domination and variants?
  - Using PQAs for High-dimensional Skyline Structures?
Thank you :) 

References

1) A Cache-Oblivious Implicit Dictionary with the Working Set Property
   • Gerth Stlting Brodal, Casper Kejlberg-Rasmussen, Jakob Truelsen
   • ISAAC 2010
   • Available at http://dx.doi.org/10.1007/978-3-642-17514-5_4

2) Cache-Oblivious Implicit Predecessor Dictionaries with the Working-Set Property
   • Gerth Stølting Brodal, Casper Kejlberg-Rasmussen
   • STACS 2012
   • Available at http://dx.doi.org/10.4230/LIPIcs.STACS.2012.112

3) I/O-Efficient Planar Range Skyline and Attrition Priority Queues
   • Casper Kejlberg-Rasmussen, Yufei Tao, Konstantinos Tsakalidis, Kostas Tsichlas, Jeonghun Yoon
   • PODS 2013
   • Available at http://doi.acm.org/10.1145/2463664.2465225
Extra Slides
Implicit Moveable Dictionaries
Implicit Moveable Dictionaries

- Uses $O(1)$ FG dictionaries as black boxes.
- Recall the FG interface:
  - Insert-right($e$): insert element $e$ into the dictionary which grows to the right.
  - Delete-right($e$): delete element $e$ from the dictionary which shrinks from the right.
  - Search($e$): finds the address of $e$ if $e$ is in the dictionary.
  - Predecessor($e$): finds the address of the predecessor of $e$.
  - Successor($e$): finds the address of the successor of $e$.
Implicit Moveable Dictionaries

- L
- C
- R
Implicit Moveable Dictionaries

- $L$ and $R$ will shrink and grow over time
  - $L/R$ might get too small or
  - $L/R$ might get too large compared to $C$
- We introduce the notion of *jobs*
  - Grow-left/right – Counters when $L/R$ gets too small
  - Shrink-left/right – Counters when $L/R$ gets too large
  - Jobs run $O(1)$ steps every operation: searches, updates
 Implicit Moveable Dictionaries

- $L$ and $R$ will shrink and grow over time
  - $L/R$ might get too small or
  - $L/R$ might get too large compared to $C$
- We introduce the notion of jobs
  - Grow-left/right – Counters when $L/R$ gets too small
  - Shrink-left/right – Counters when $L/R$ gets too large
  - Jobs run $O(1)$ steps every operation: searches, updates
Implicit Moveable Dictionaries

- $L$ and $R$ will shrink and grow over time
  - $L/R$ might get too small or
  - $L/R$ might get too large compared to $C$
- We introduce the notion of *jobs*
  - Grow-left/right – Counters when $L/R$ gets too small
  - Shrink-left/right – Counters when $L/R$ gets too large
- Jobs run $O(1)$ steps every operation: searches, updates
Implicit Moveable Dictionaries

- $L$ and $R$ will shrink and grow over time
- $L/R$ might get too small or too large compared to $C$

We introduce the notion of jobs:
- Grow-left/right – Counters when $L/R$ gets too small
- Shrink-left/right – Counters when $L/R$ gets too large
- Jobs run $O(1)$ steps every operation: searches, updates

Summary of Invariants
- We ensure that both $L$ and $R$ are not too small/large
- We have queued at most 2 jobs
- All jobs finish before their deadline
Implicit Moveable Dictionaries
Implicit Moveable Dictionaries

L  C  R
L' C  R

Grow-left
Implicit Moveable Dictionaries

Grow-left

L  C  R

L'  C  R

L  L'  C  R
Implicit Moveable Dictionaries

![Diagram of Implicit Moveable Dictionaries]

Grow-left
Implicit Moveable Dictionaries

Grow-left

Address-mapping

L L' C R

L L' C R

L L' C R

L L' C R

k k' i i'

i j
Implicit Moveable Dictionaries
Implicit Moveable Dictionaries
Implicit Moveable Dictionaries
Implicit Moveable Dictionaries
Implicit Moveable Dictionaries
Implicit Moveable Dictionaries

Grow-left

Shrink-left

Casper Kejlberg-Rasmussen
Implicit Moveable Dictionaries

Grow-left

Shrink-left
Implicit Moveable Dictionaries

<table>
<thead>
<tr>
<th>Grow-left</th>
<th>Shrink-left</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="diagram1.png" alt="Diagram" /></td>
<td><img src="diagram2.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>