Dynamic Data Structures: The Interplay of Invariants and Algorithm Design

Casper Kejlberg-Rasmussen

PhD Defense, 18th of November 2013

MADALGO
CENTER FOR MASSIVE DATA ALGORITHMICS

Danmarks Grundforskningsfon
Danish National Research Foundation

AARHUS UNIVERSITY

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 ≋ Recipe

Data ≋ Ingredients

Data Structure ≋ Organization of Ingredients

Updates ≋ Ingredients Updates

Queries ≋ 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 ≋ Recipe

Data ≋ Ingredients

Data Structure ≋ Organization of Ingredients

Updates ≋ Ingredients Updates

Queries ≋ Follow Recipes

Design Criteria

- Fast
- Low Space Usage

- Fast queries and updates
- Low Space Usage

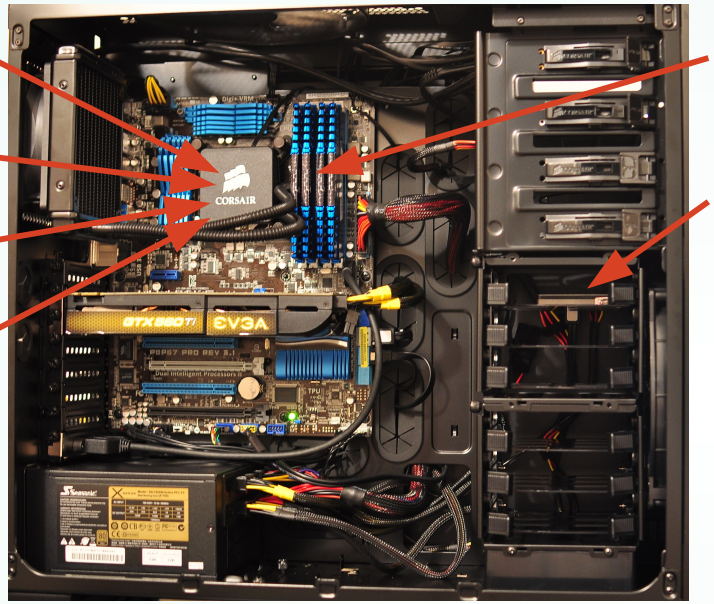
What are Computational Models?

What are Computational Models?

Reality

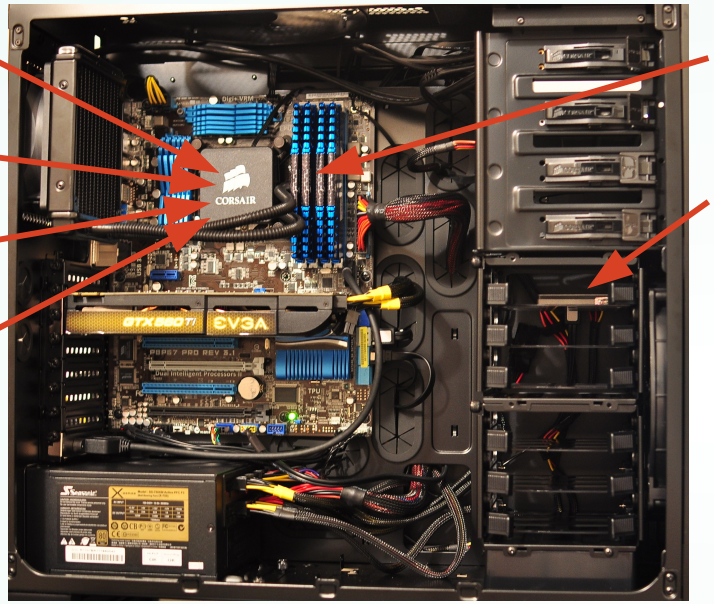
What are Computational Models?

Reality



What are Computational Models?

Reality

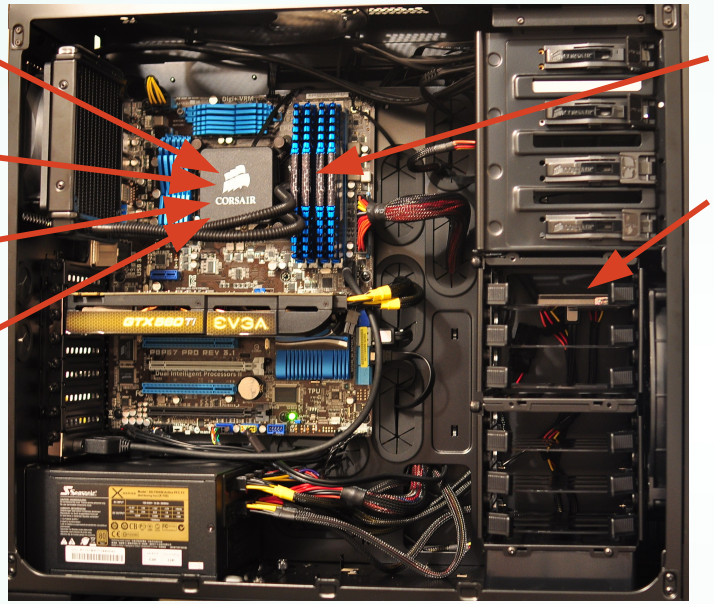


Complexity

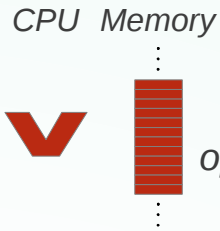
$$\left(CPUSpeed \cdot L1 \cdot \lceil \frac{L2}{L1} \rceil \cdot \lceil \frac{L3}{L2} \rceil \cdot \lceil \frac{n}{L3} \rceil\right) \log n$$

What are Computational Models?

Reality



Models



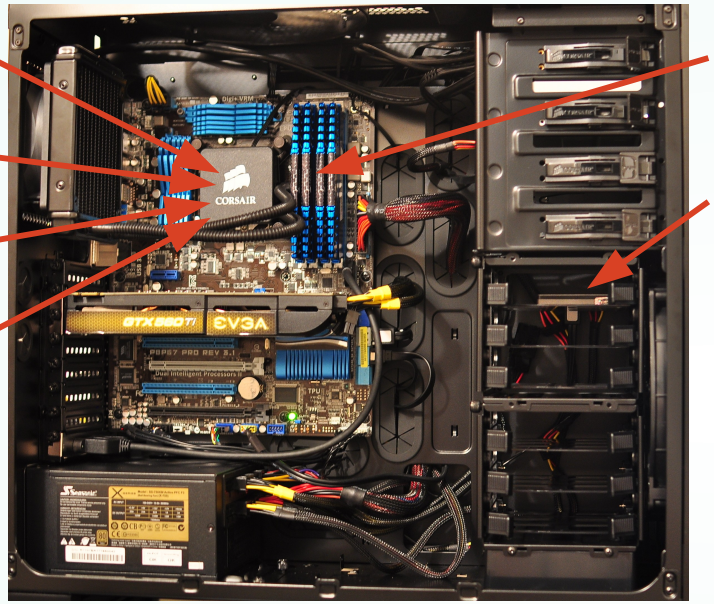
RAM Model

Complexity

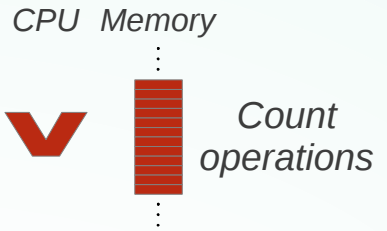
$$\left(CPUSpeed \cdot L1 \cdot \lceil \frac{L2}{L1} \rceil \cdot \lceil \frac{L3}{L2} \rceil \cdot \lceil \frac{n}{L3} \rceil\right) \log n$$

What are Computational Models?

Reality



Models



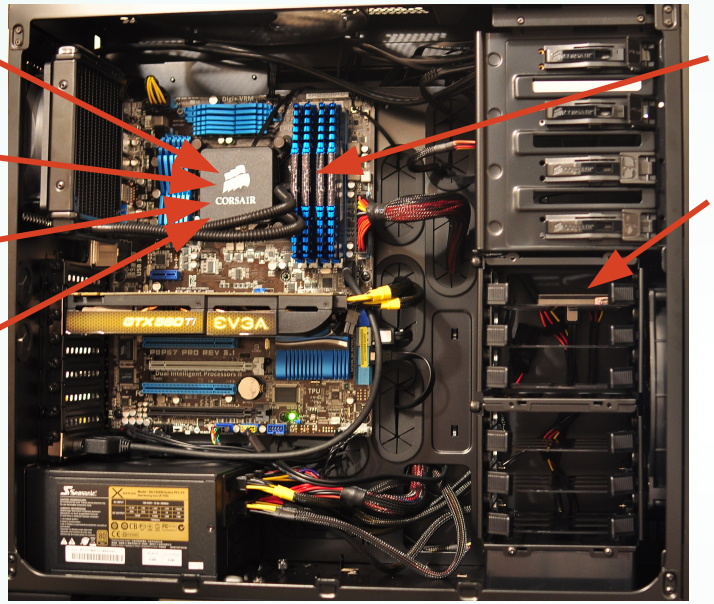
RAM Model

Complexity

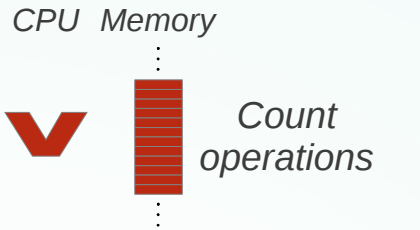
$$\left(CPUSpeed \cdot L1 \cdot \lceil \frac{L2}{L1} \rceil \cdot \lceil \frac{L3}{L2} \rceil \cdot \lceil \frac{n}{L3} \rceil\right) \log n \qquad O(n \log n)$$

What are Computational Models?

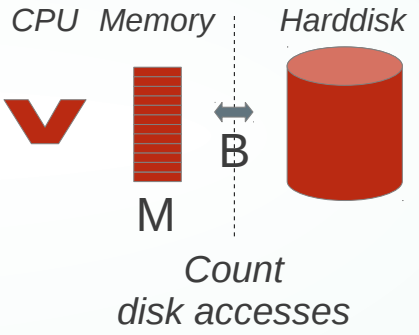
Reality



Models



RAM Model



EM Model

Many more

Complexity

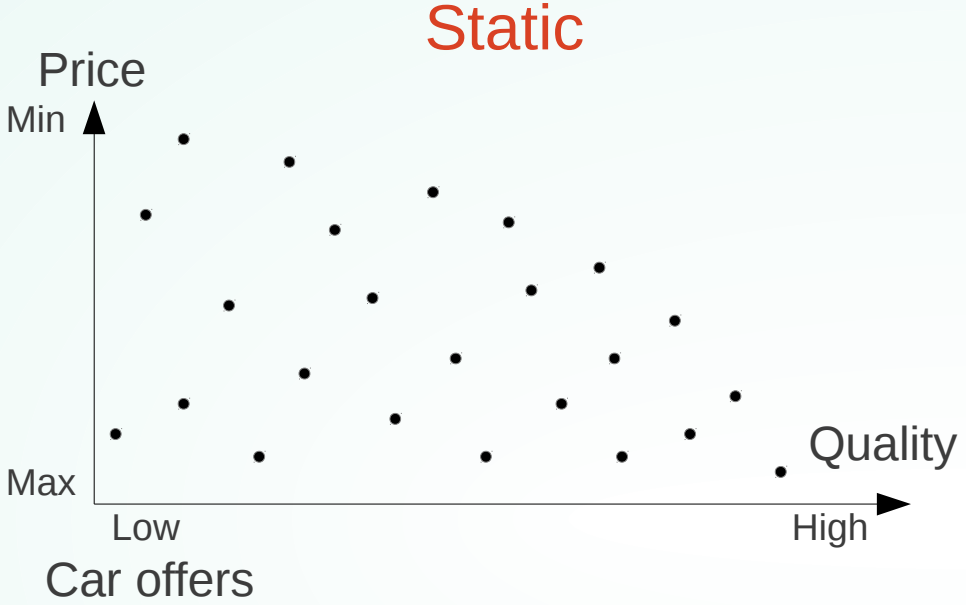
$$\left(CPUSpeed \cdot L1 \cdot \lceil \frac{L2}{L1} \rceil \cdot \lceil \frac{L3}{L2} \rceil \cdot \lceil \frac{n}{L3} \rceil\right) \log n$$

$$O(n \log n)$$

Static and Dynamic Problems

Static

Static and Dynamic Problems

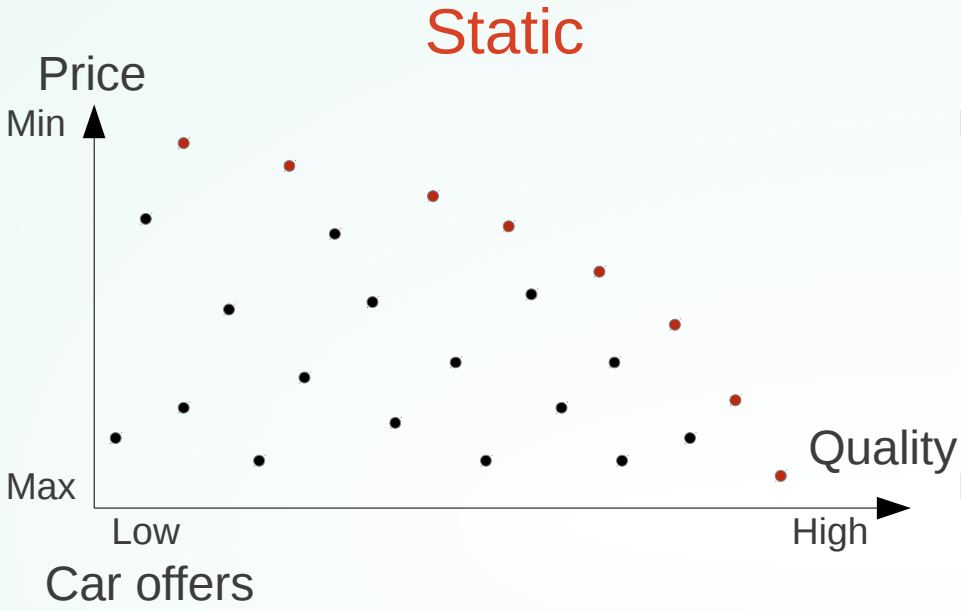


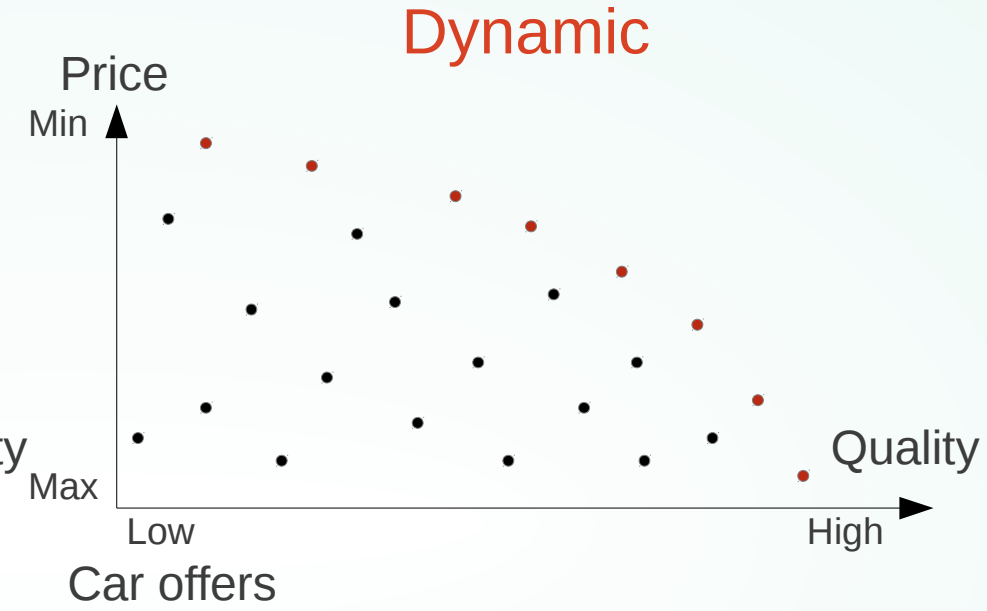
- We want to buy a new car!
- We have a list of offers
- We want to find the *undominated* offers, i.e. unmatched in price and quality

Static and Dynamic Problems

- 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

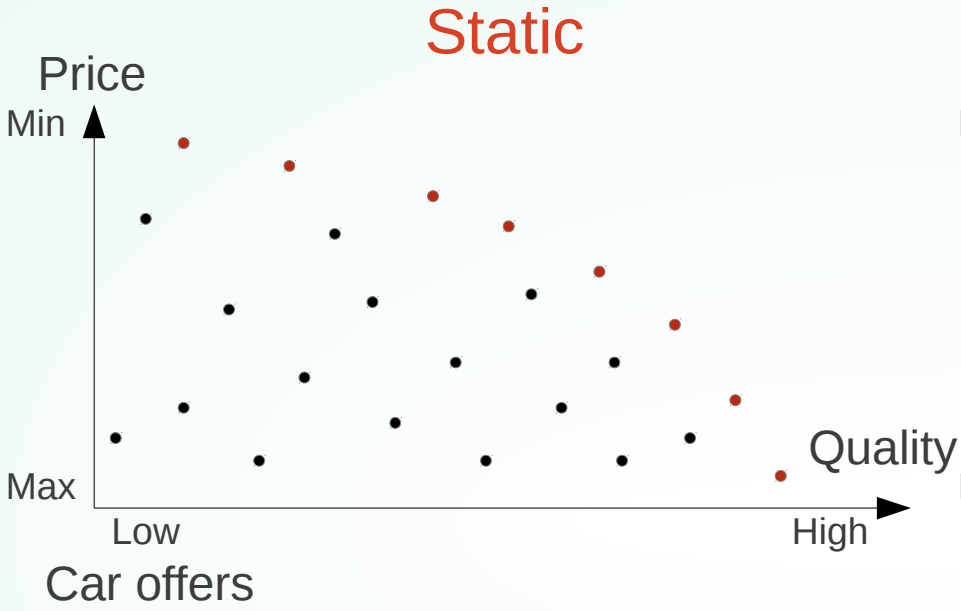


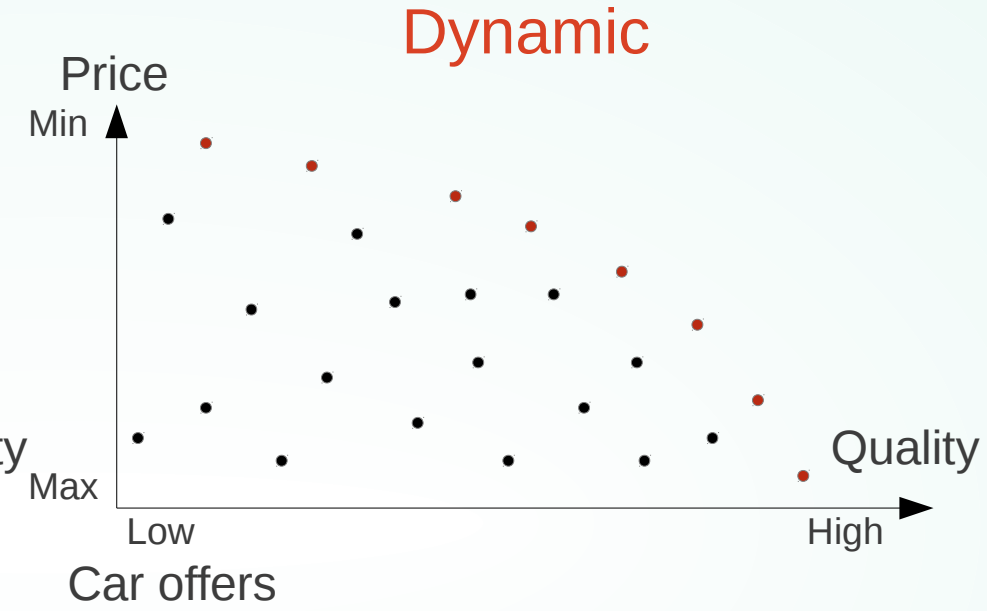


- We want to buy a new car!
- We have a list of offers
- We want to find the *undominated* offers, i.e. unmatched in price and quality

- In the dynamic setting, we will receive new offers continously
- New offers might change the set of undominated offers

Static and Dynamic Problems

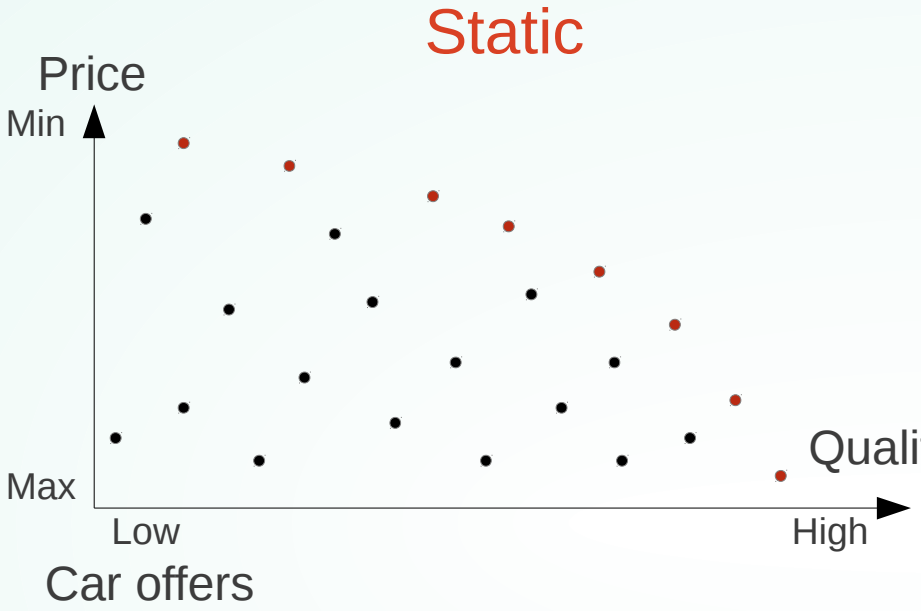


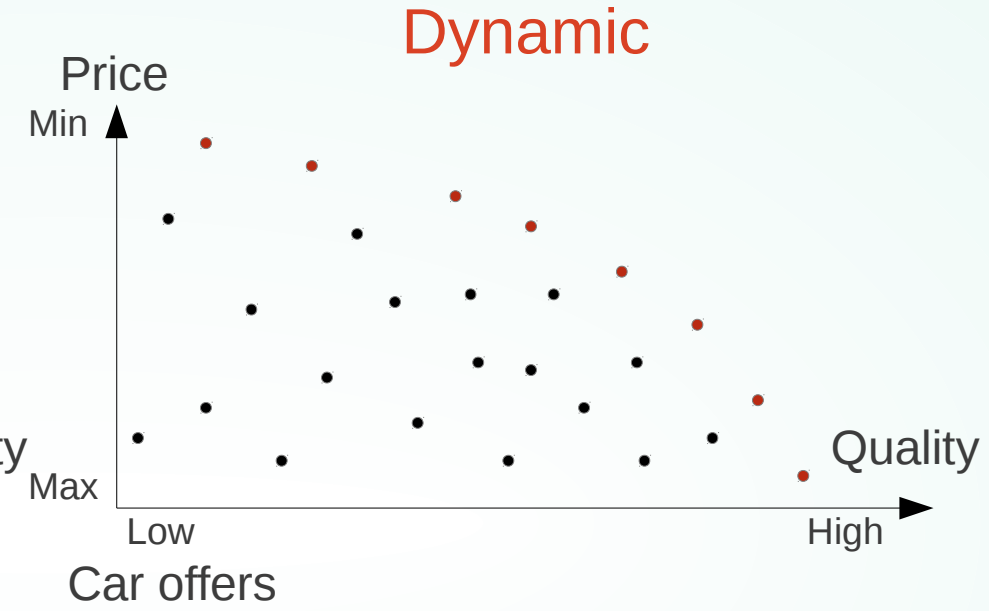


- We want to buy a new car!
- We have a list of offers
- We want to find the *undominated* offers, i.e. unmatched in price and quality

- In the dynamic setting, we will receive new offers continously
- New offers might change the set of undominated offers

Static and Dynamic Problems

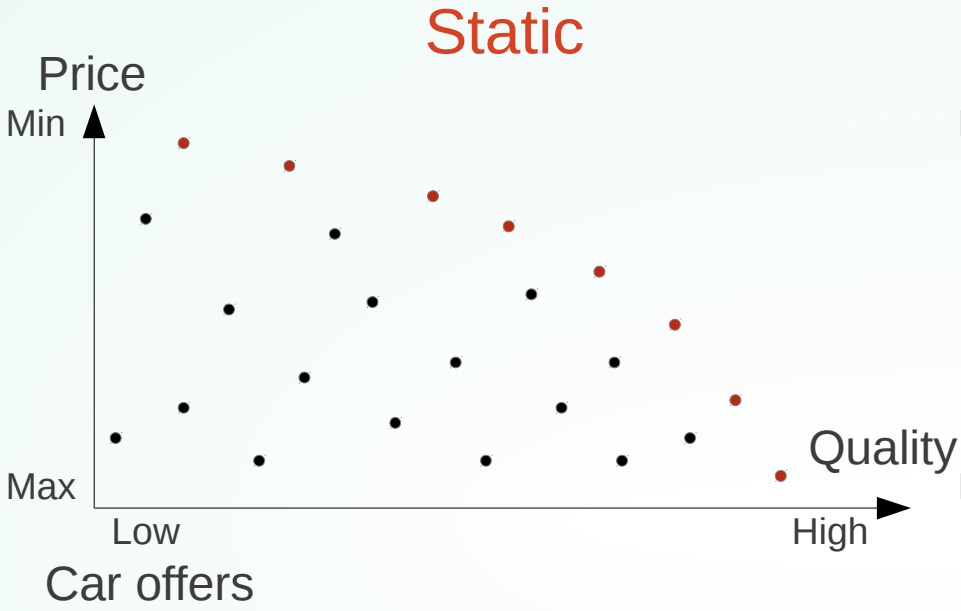


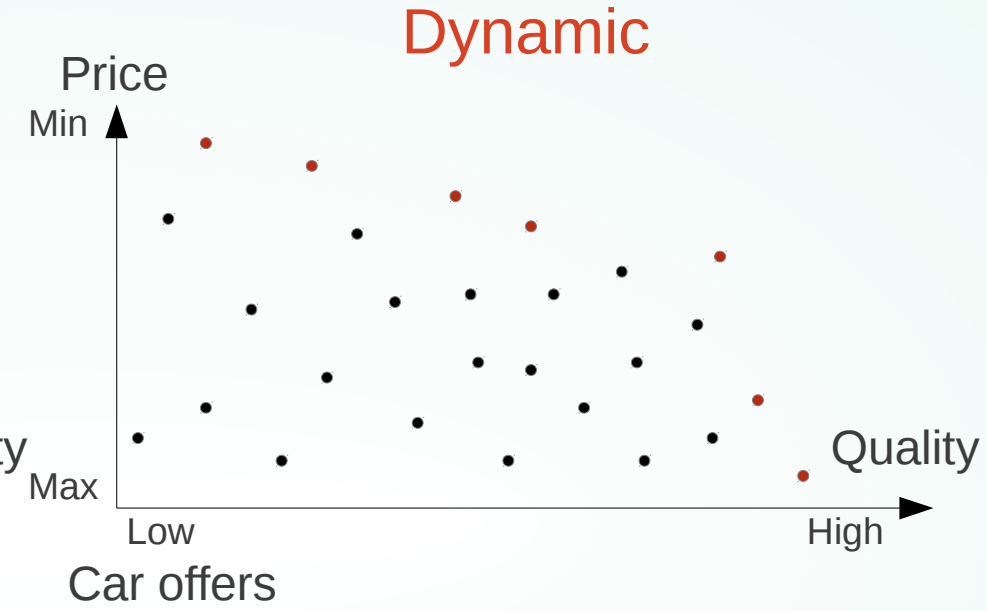


- We want to buy a new car!
- We have a list of offers
- We want to find the *undominated* offers, i.e. unmatched in price and quality

- In the dynamic setting, we will receive new offers continously
- New offers might change the set of undominated offers

Static and Dynamic Problems





- 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 continously
- 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?

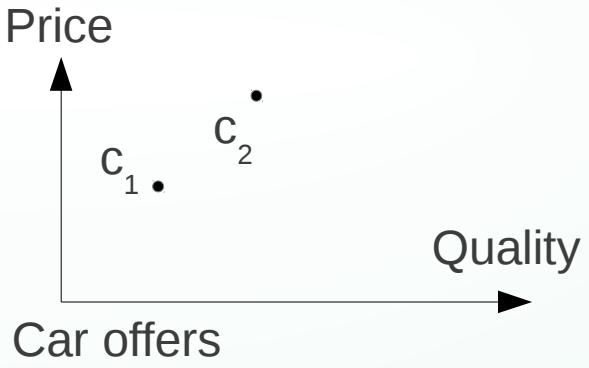
An *Invariant* is a logical statement about the structural properties of a data structure

- 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

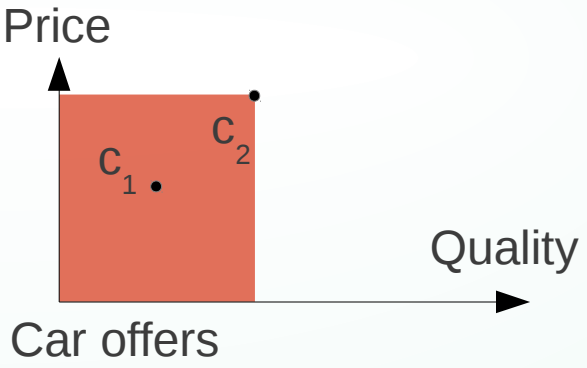
- Invariants are designed from observations of the problem we try to solve
- Consider two cars from the *car offers problem* from before



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

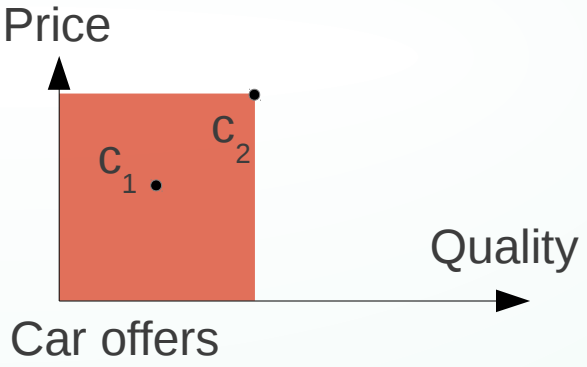


What are Invariants?

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

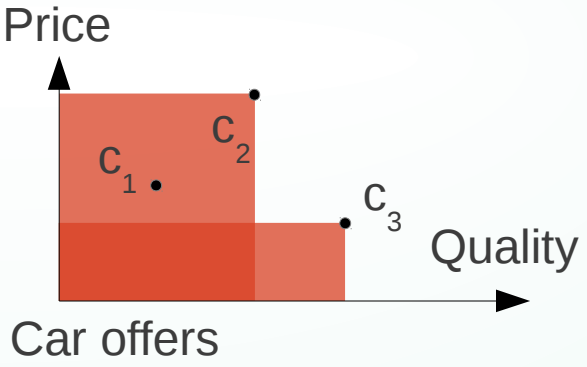


What are Invariants?

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 dominates c_1



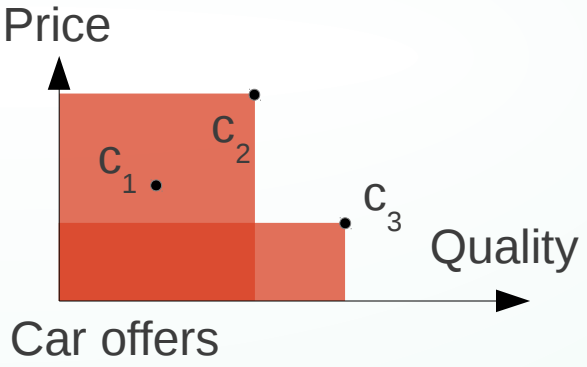
What are Invariants?

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 dominates c_1

c_1 and c_3 are incomparable



What are Invariants?

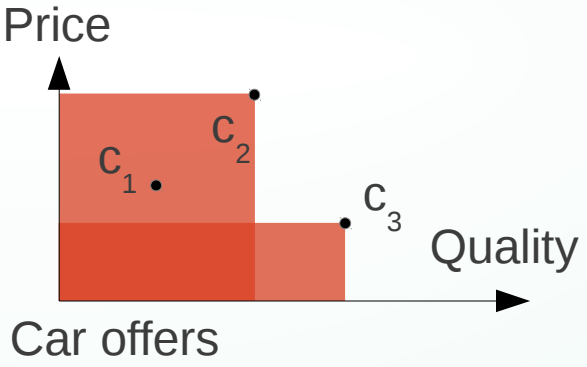
An *Invariant* is a logical statement about the structural properties of a data structure

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

c_2 dominates c_1

c_1 and c_3 are incomparable

c_2 and c_3 are incomparable



What are Invariants?

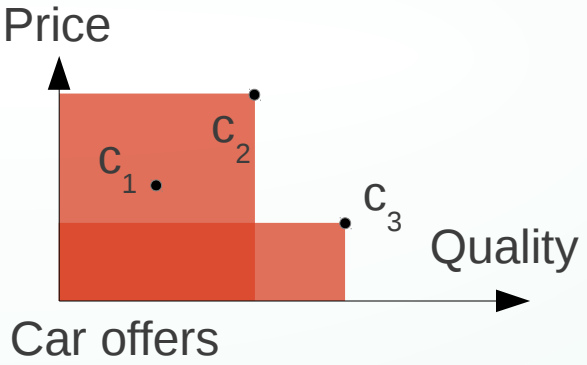
An *Invariant* is a logical statement about the structural properties of a data structure

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

c_2 dominates c_1

c_1 and c_3 are incomparable

c_2 and c_3 are incomparable



- We notice that the *undominated* (c_2 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?

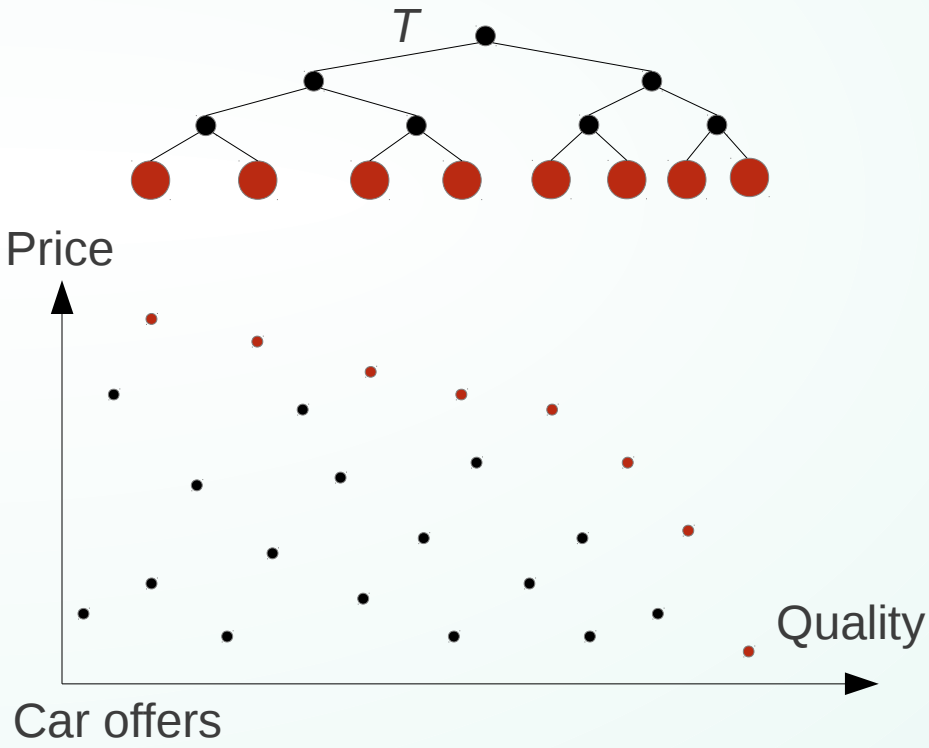
- From our observation we store all undominated points in a search tree sorted simultaneously on price and quality
- This gives us the following data structure and invariant

  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

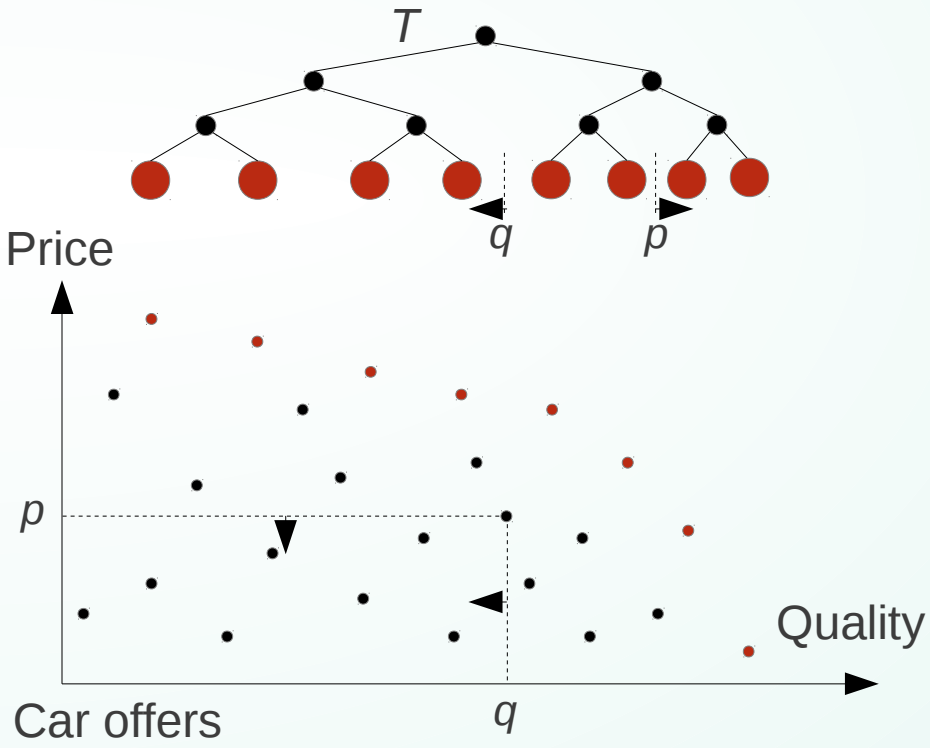
Invariant: All undominated offers are stored in the search tree *T* and are sorted simultaneously on price and quality



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

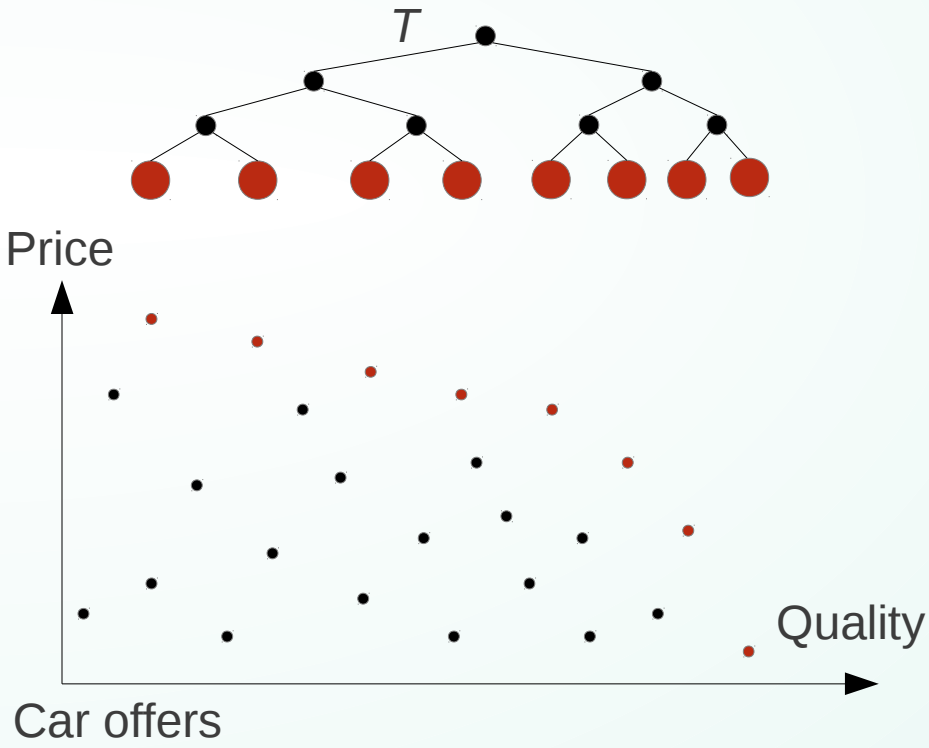
Invariant: All undominated offers are stored in the search tree *T* and are sorted simultaneously on price and quality



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

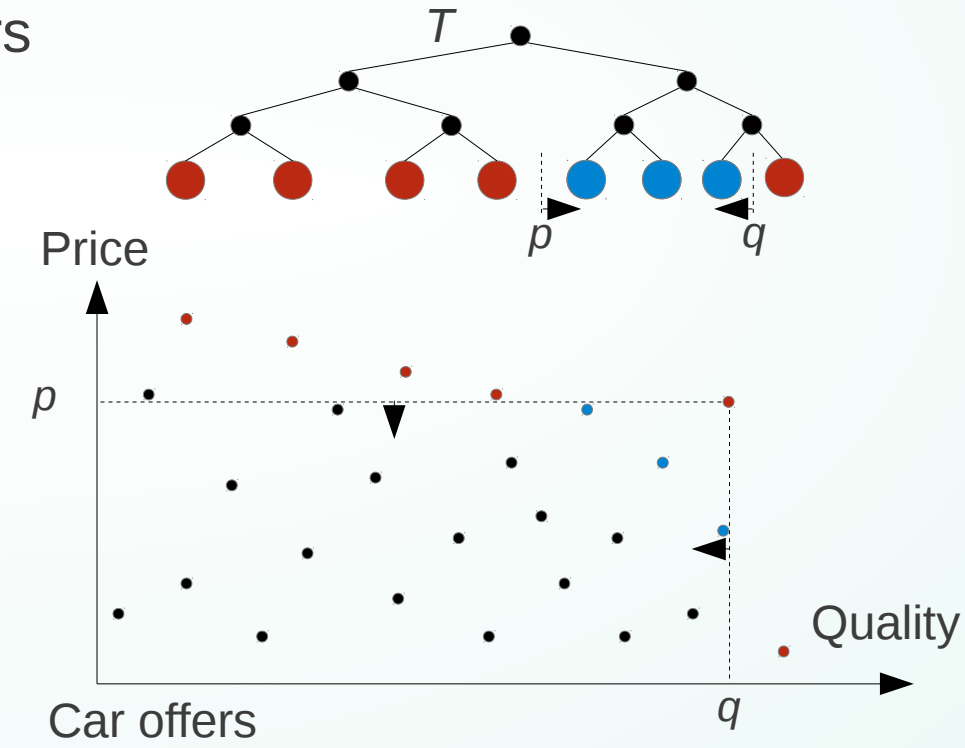
Invariant: All undominated offers are stored in the search tree *T* and are sorted simultaneously on price and quality



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

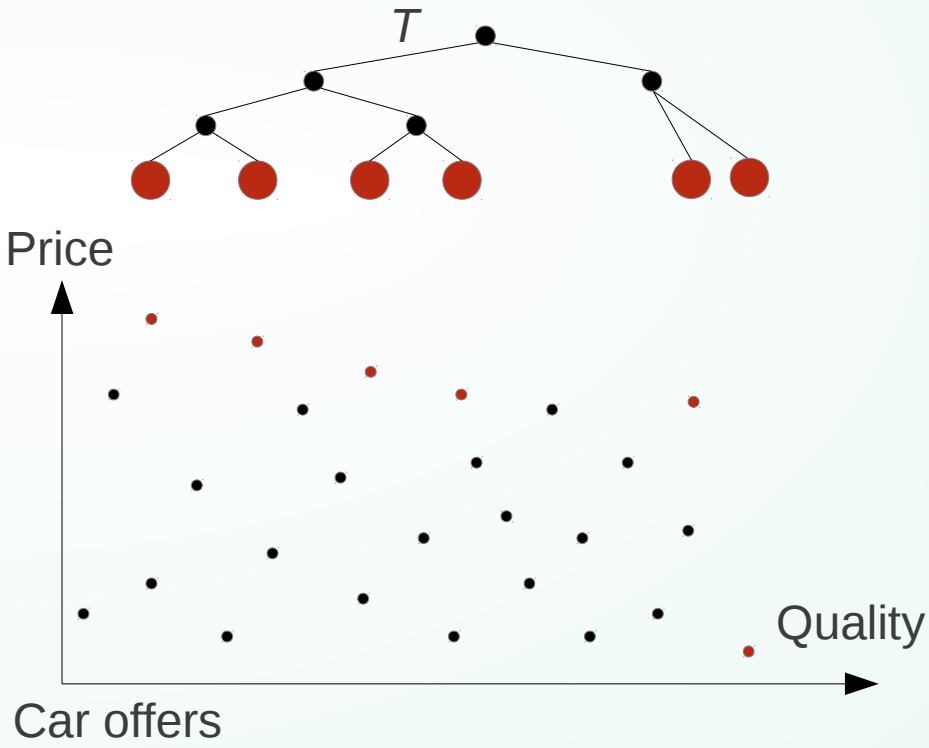
Invariant: All undominated offers are stored in the search tree *T* and are sorted simultaneously on price and quality



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

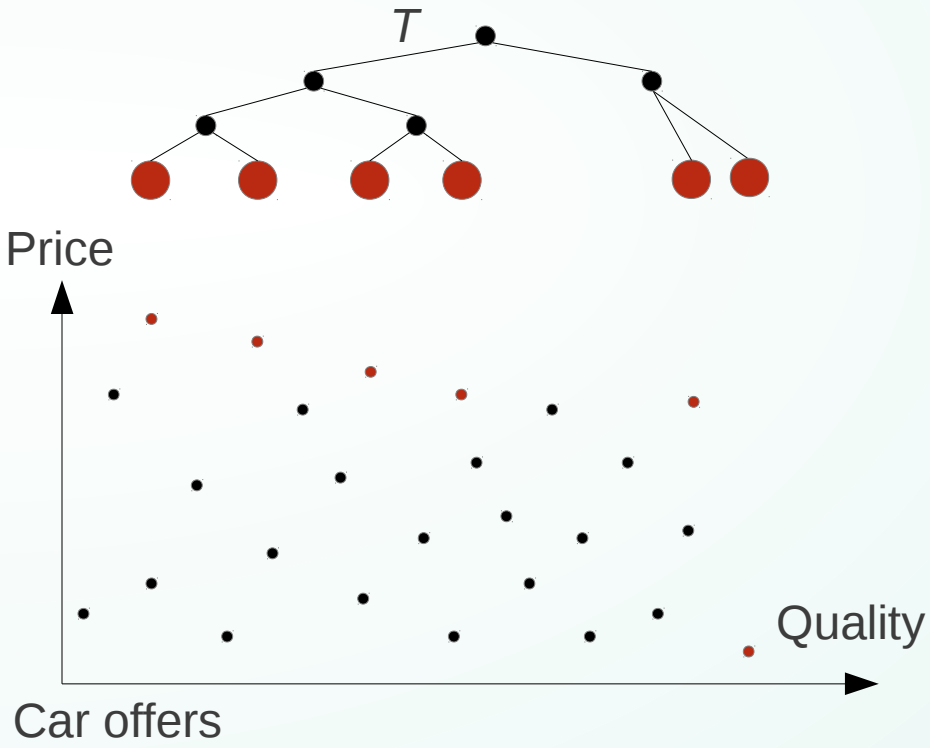
Invariant: All undominated offers are stored in the search tree *T* and are sorted simultaneously on price and quality



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!

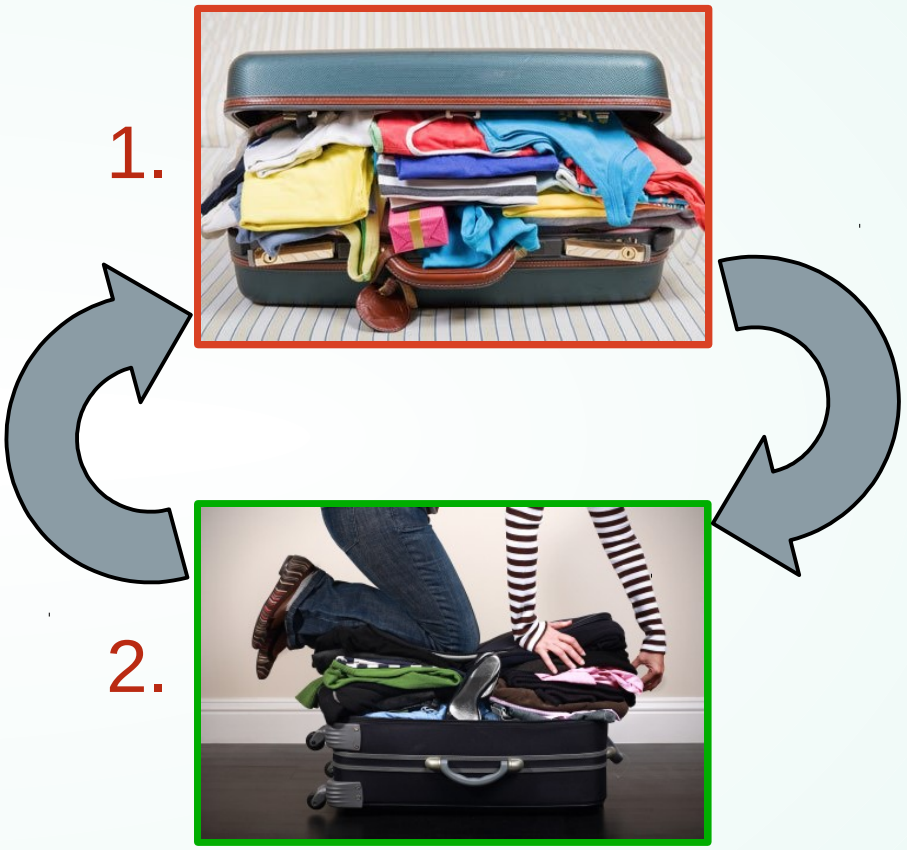
Designing Invariants and Dynamic Data Structures

- 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!

1.

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

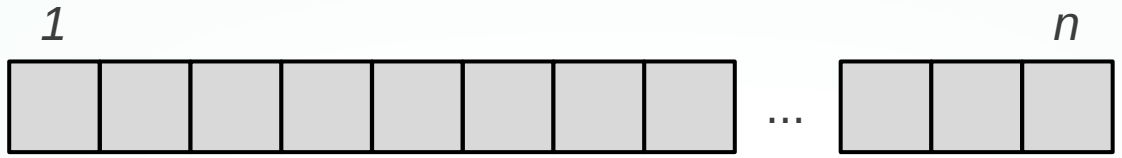
Implicit Model

- 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

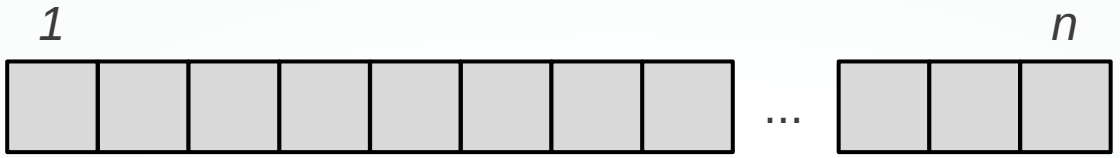
- All operations from the RAM
- It is not allowed to *create* words, only to *move* them
- All n words have to be in contiguous positions



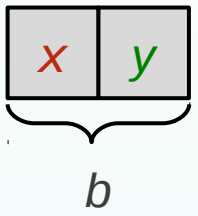
- Often it is assumed that all elements are distinct
- Fundamental trick: encode a bit in a pair of adjacent and distinct elements

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



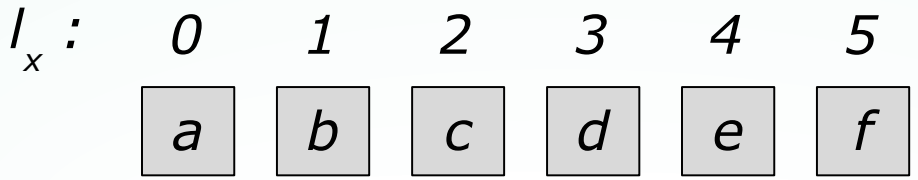
- Often it is assumed that all elements are distinct
- Fundamental trick: encode a bit in a pair of adjacent and distinct elements



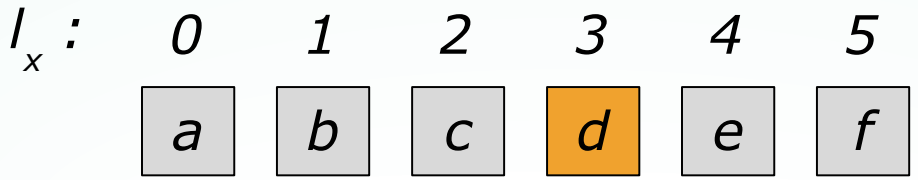
$$b = \begin{cases} 0 \text{ if } x=\min(x,y) \\ 1 \text{ if } x=\max(x,y) \end{cases}$$

The Working-Set Property

The Working-Set Property



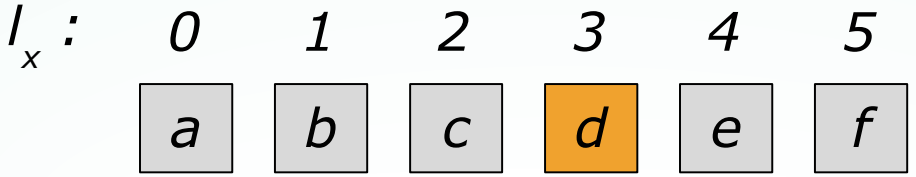
The Working-Set Property



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



- *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

Ref.	WS prop.	Insert/Delete(e)	Search(e)	Predecessor/ Successor(e)	Additional Words
M1989	-	$O(\log^2 n)$	$O(\log^2 n)$		None
FGMP2002	-	$O(\log^2 n/\log\log n)$	$O(\log^2 n/\log\log n)$		None
FG2006	-	$O(\log n)$ amor.	$O(\log n)$	$O(\log n)$	None
FG2003	-	$O(\log n)$	$O(\log n)$	$O(\log n)$	None
I2001	+	$O(\log n)$	$O(\log l_e)$	$O(\log l_{e^*})$	$O(n)$
BHM2009	+	$O(\log n)$	$O(\log l_e)$ exp.	$O(\log n)$	$O(\log\log n)$
BHM2009	+	$O(\log n)$	$O(\log l_e)$ exp.	$O(\log l_e)$ exp.	$O(\sqrt{n})$
BKT2010	+	$O(\log n)$	$O(\log l_e)$	$O(\log n)$	None
BK2011	+	$O(\log n)$	$O(\log \min(l_{p(e)}, l_e, l_{s(e)}))$	$O(\log l_{e^*})$	None

e^* is the predecessor/successor of e

Implicit Moveable Dictionaries

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*]

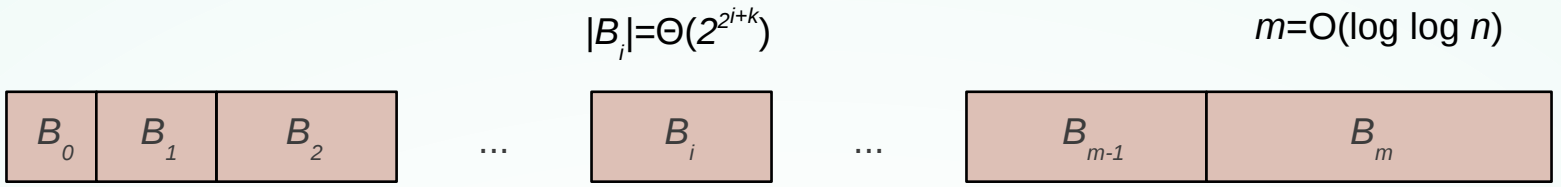
- *Interface:*
  - Insert-left/right(*e*): insert element *e* into the dictionary which grows to the left/right
  - Delete-left/right(*e*): delete element *e* from the dictionary which shrinks from the left/right
  - Search(*e*): finds the address of *e* if *e* is in the dictionary
  - Predecessor(*e*): 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*]

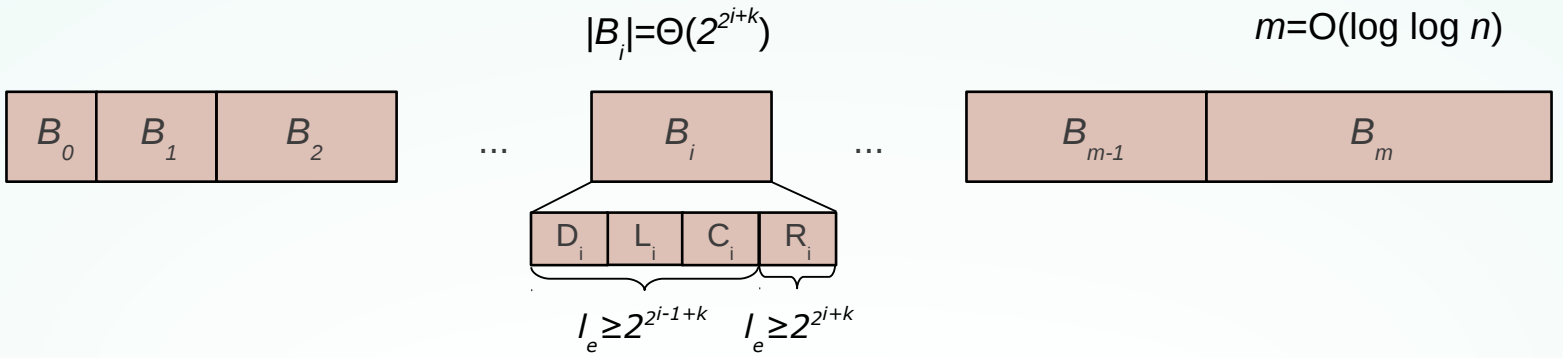
- *Interface:*
  - Insert-left/right(*e*): insert element *e* into the dictionary which grows to the left/right
  - Delete-left/right(*e*): delete element *e* from the dictionary which shrinks from the left/right
  - Search(*e*): finds the address of *e* if *e* is in the dictionary
  - Predecessor(*e*): 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 Working-Set Dictionaries



Implicit Working-Set Dictionaries

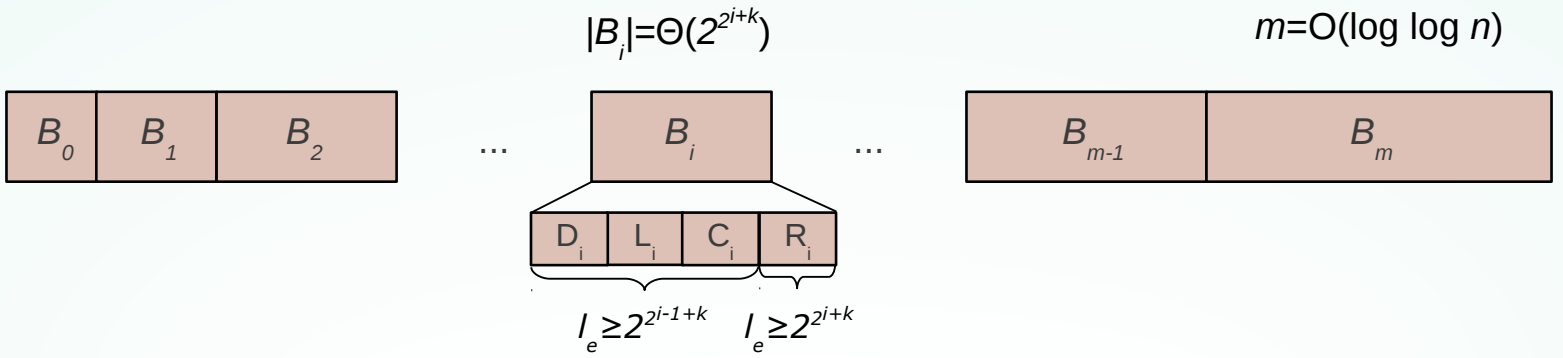
- Exponential layout



- B_i consists of O(*1*) moveable dictionaries
- All elements e in B_i have $l_e \geq 2^{2i-1+k}$ or $l_e \geq 2^{2i+k}$
- 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

Implicit Working-Set Dictionaries

- Exponential layout



- B_i consist
- All eleme
- Searched rflows)
- These ar
- 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

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

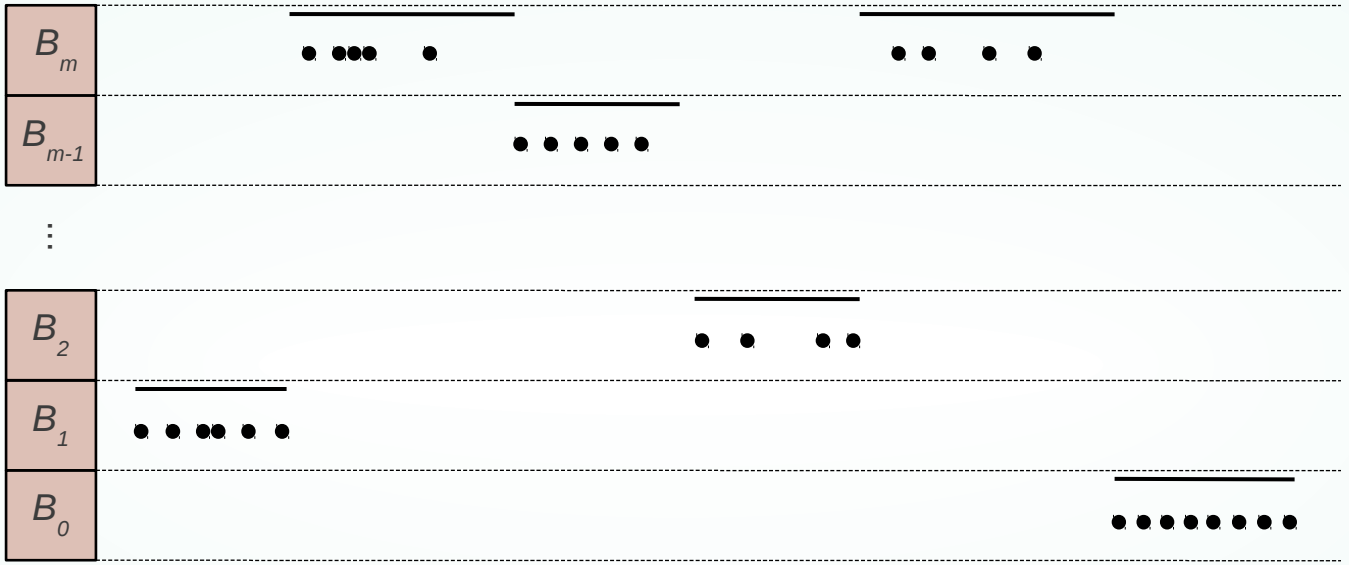
- Intervals to solve the predecessor and successor problems

Implicit Predecessor/Successor Working-Set Dictionaries

- Intervals to solve the predecessor and successor problems

Implicit Predecessor/Successor Working-Set Dictionaries

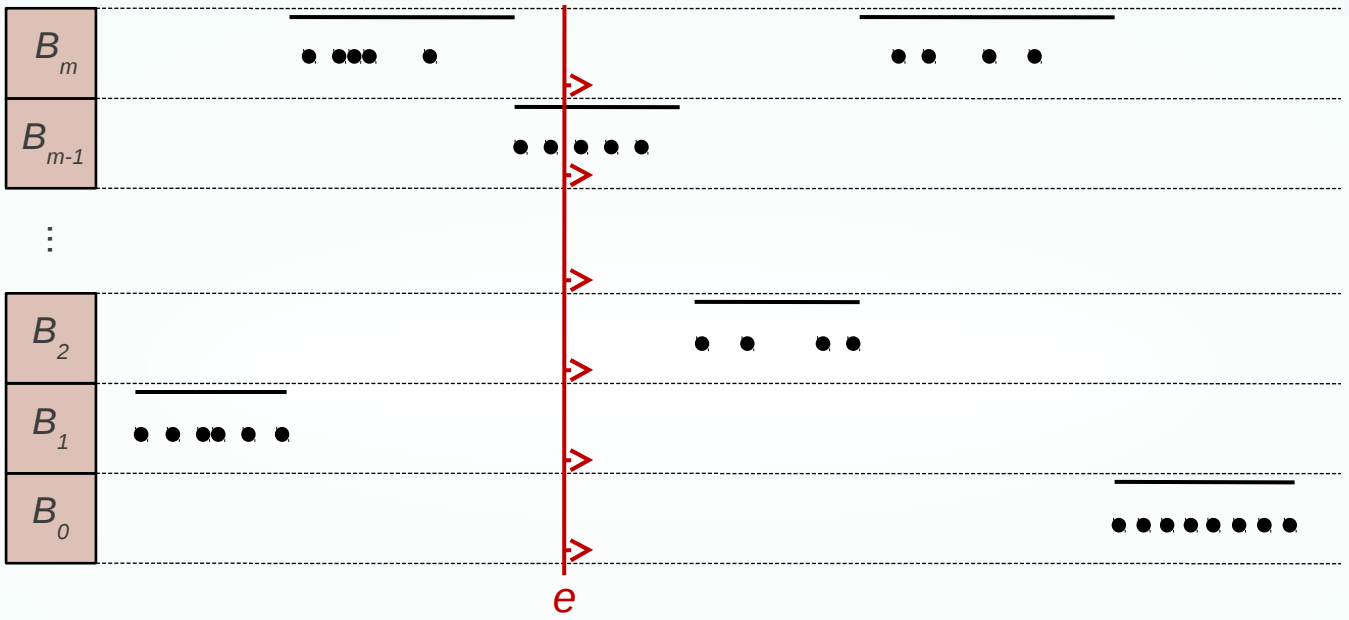
- Intervals to solve the predecessor and successor problems



- 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 Predecessor/Successor Working-Set Dictionaries

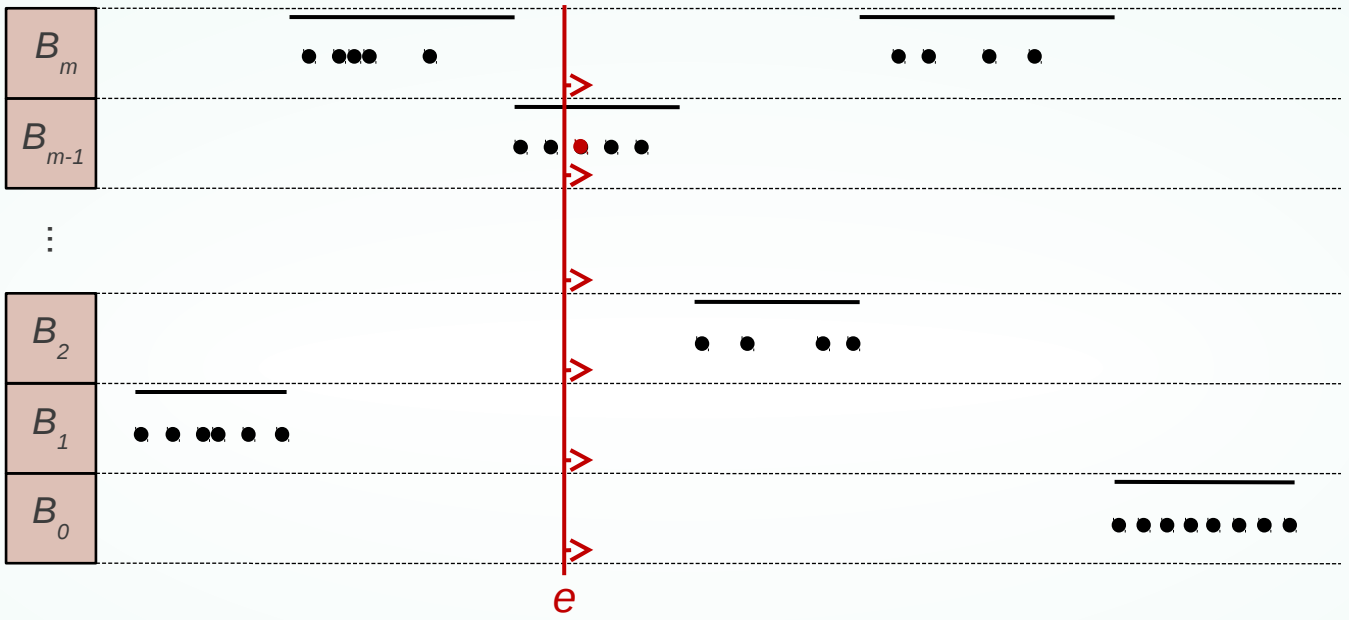
- Intervals to solve the predecessor and successor problems



- 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 Predecessor/Successor Working-Set Dictionaries

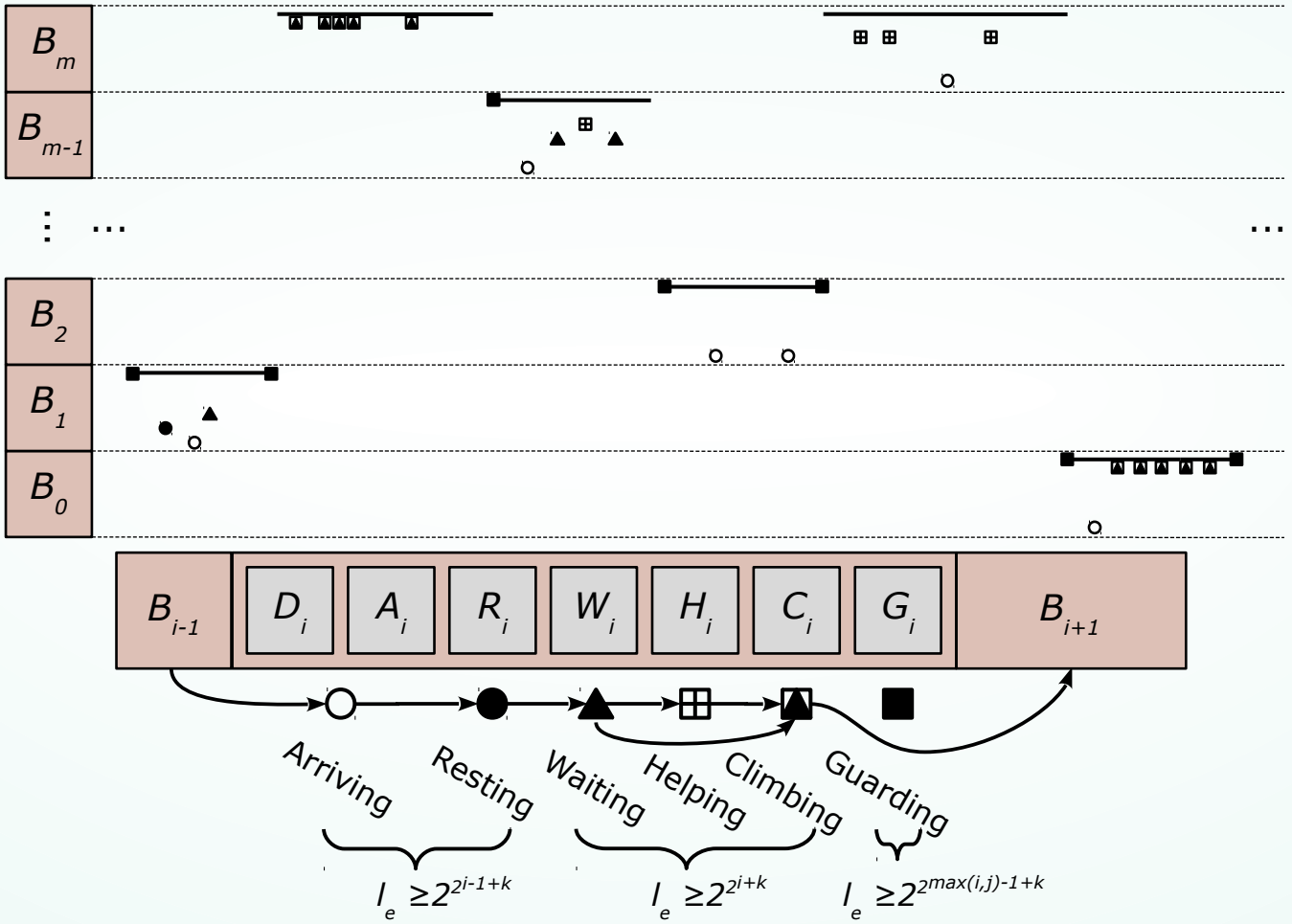
- Intervals to solve the predecessor and successor problems



- 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

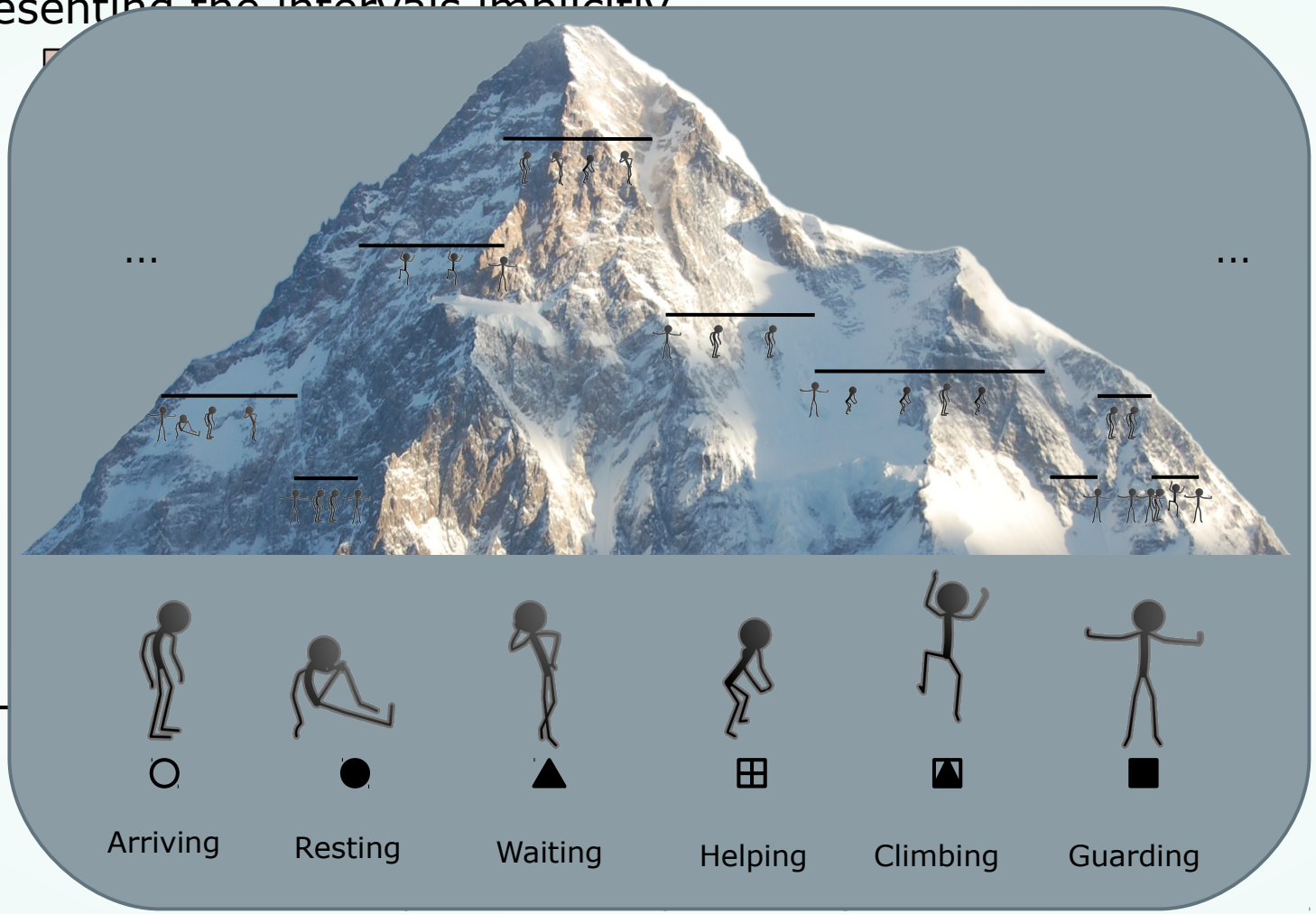
Implicit Representation of Intervals

- Representing the intervals implicitly



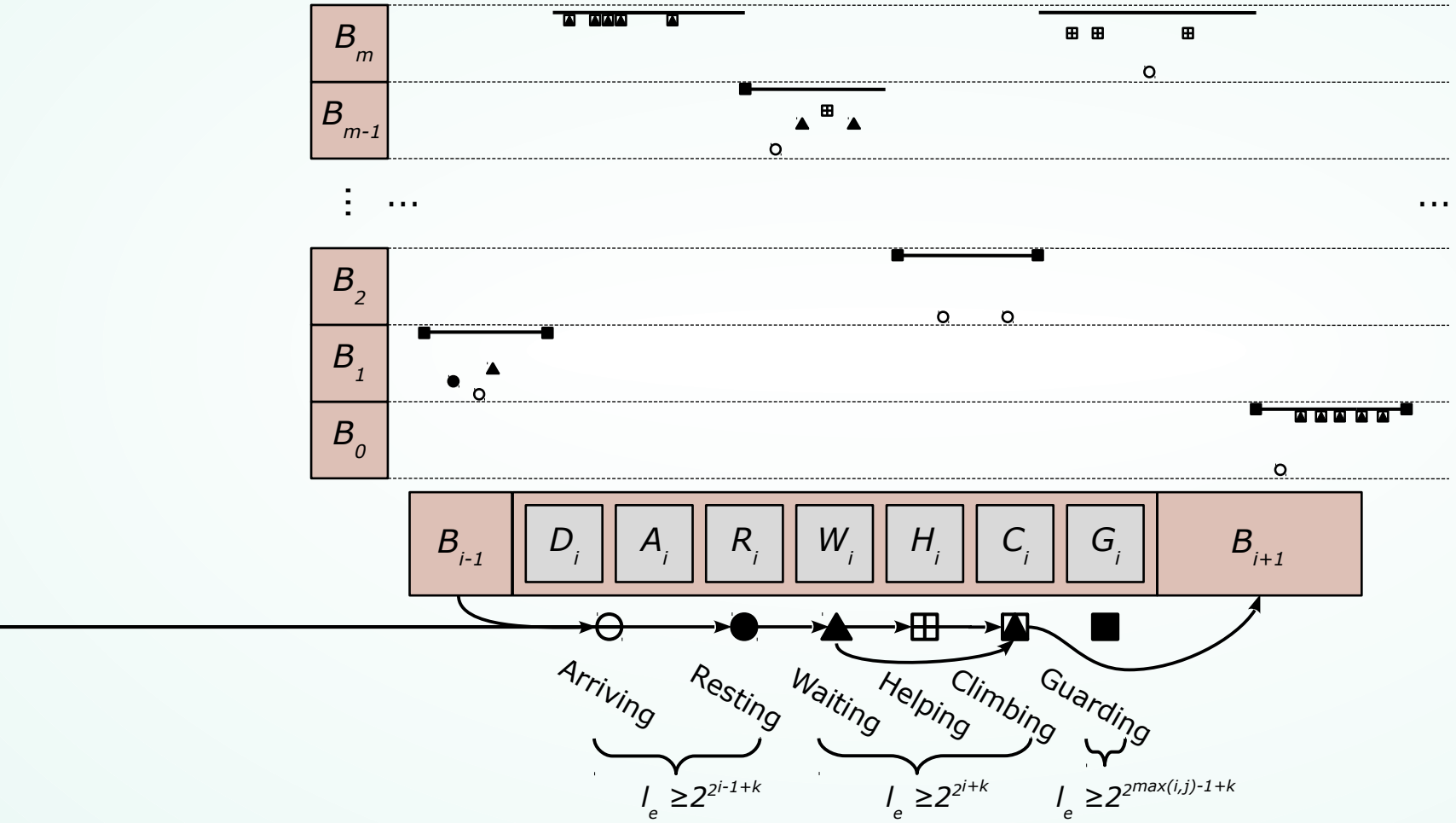
Implicit Representation of Intervals

- Representing the intervals implicitly



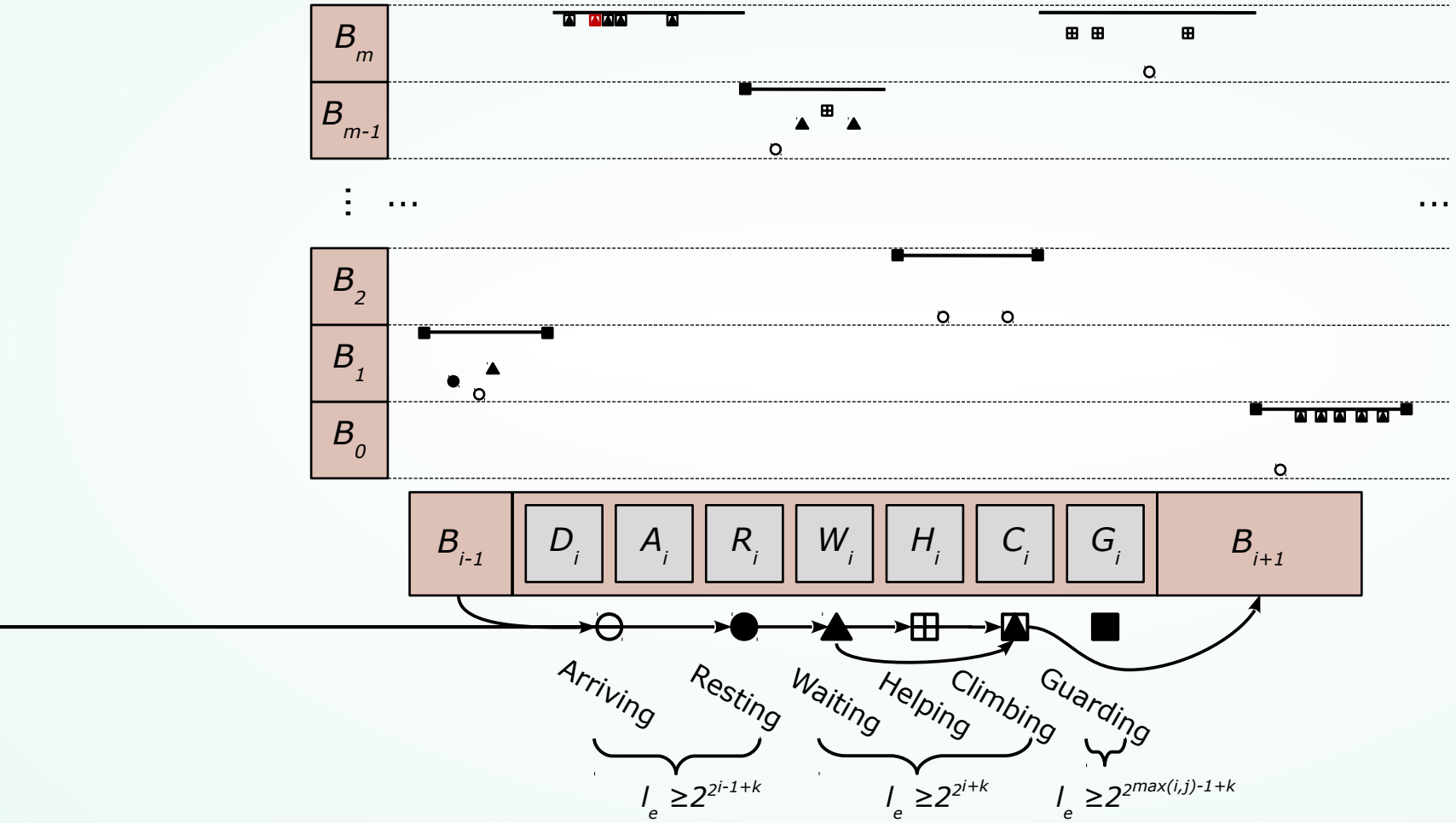
Implicit Representation of Intervals

- Representing the intervals implicitly



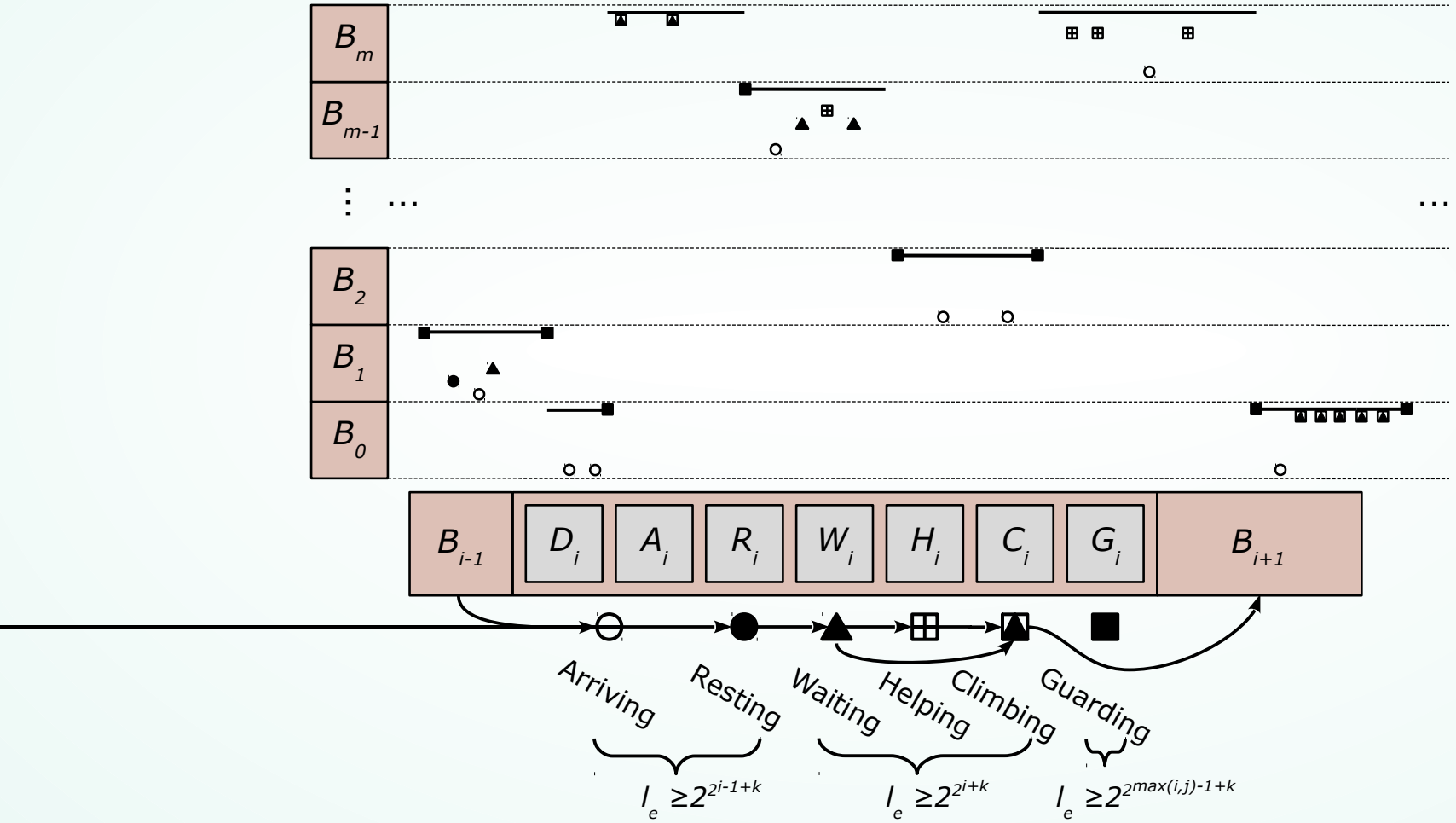
Implicit Representation of Intervals

- Representing the intervals implicitly



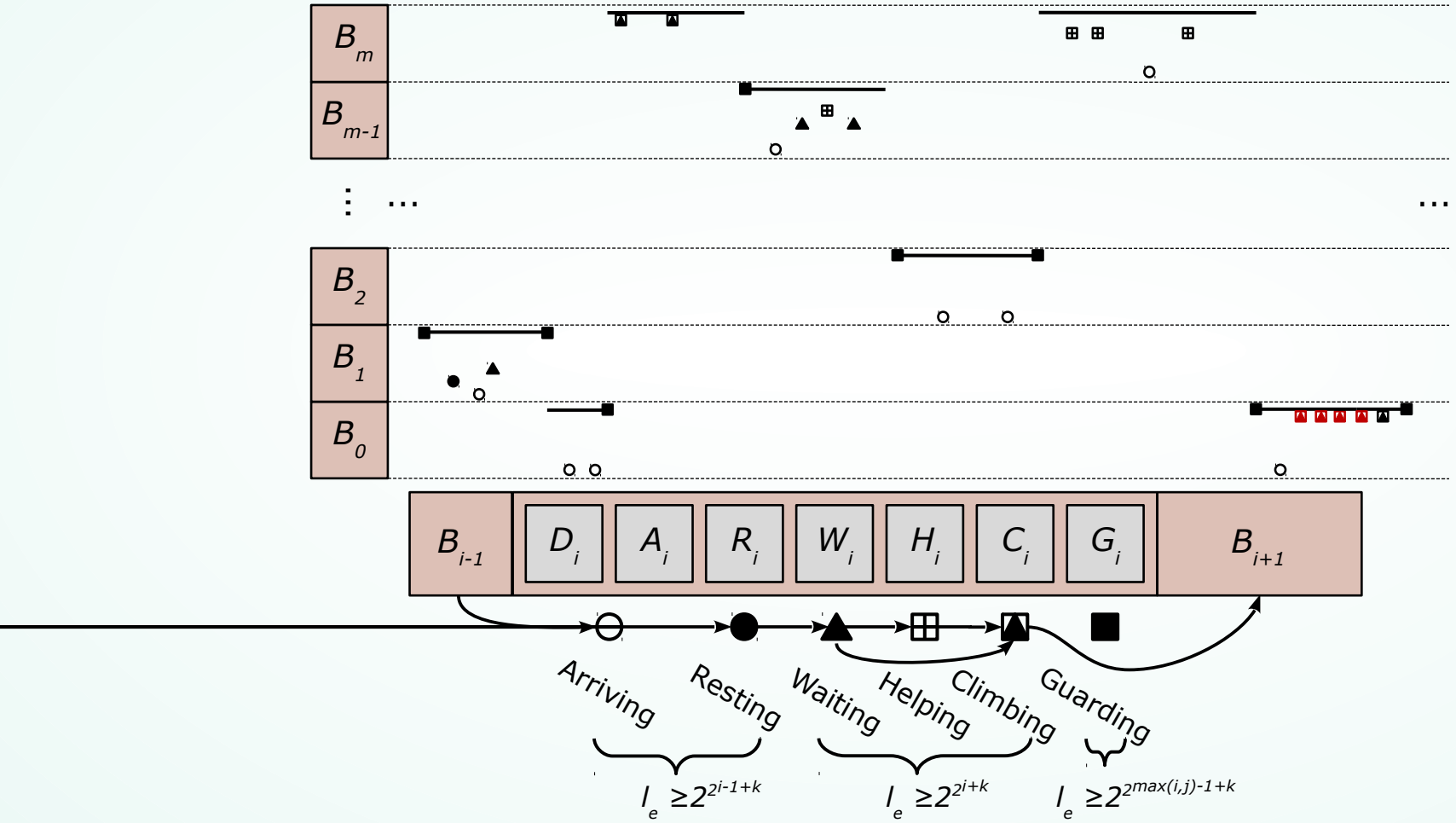
Implicit Representation of Intervals

- Representing the intervals implicitly



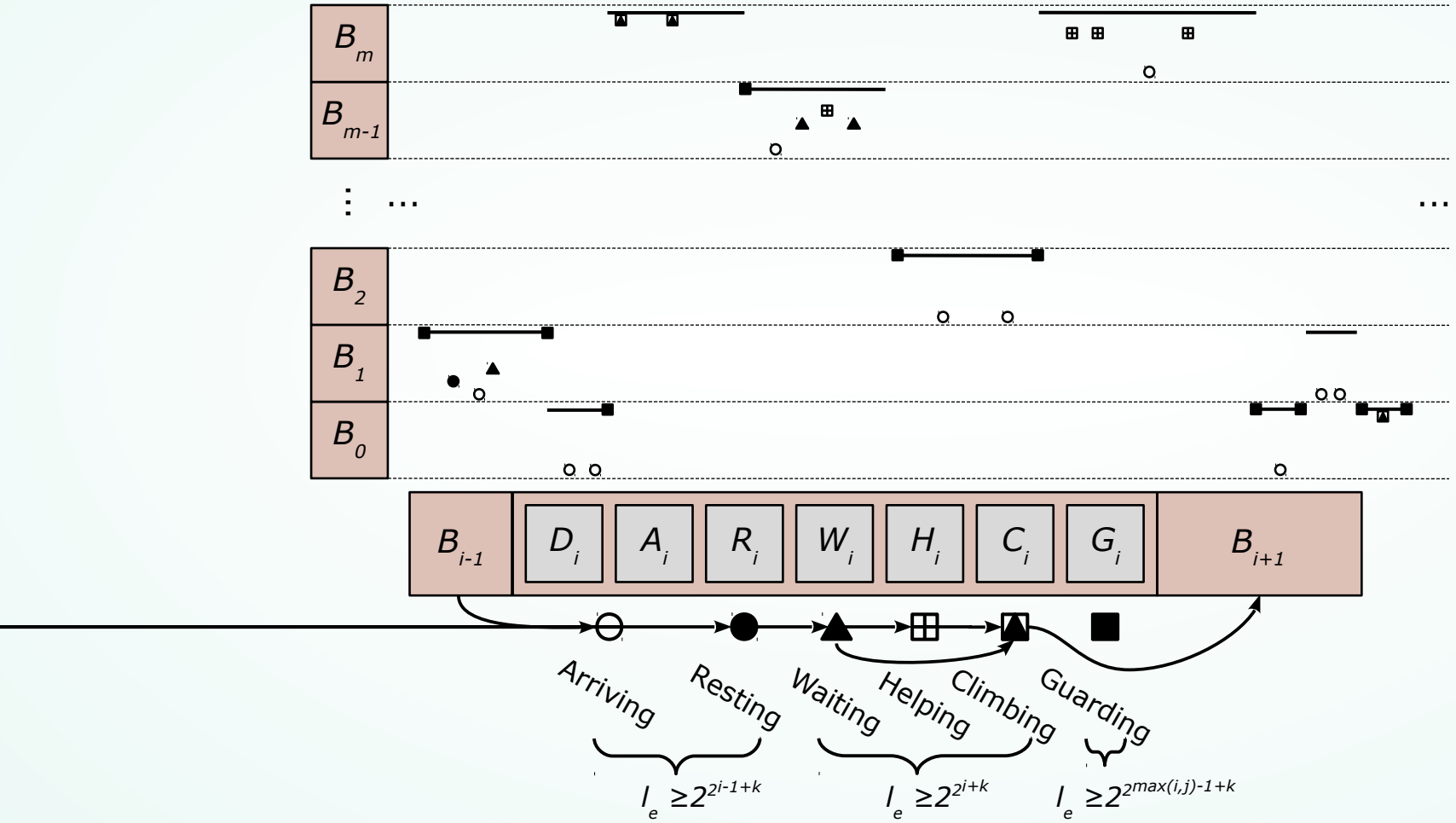
Implicit Representation of Intervals

- Representing the intervals implicitly



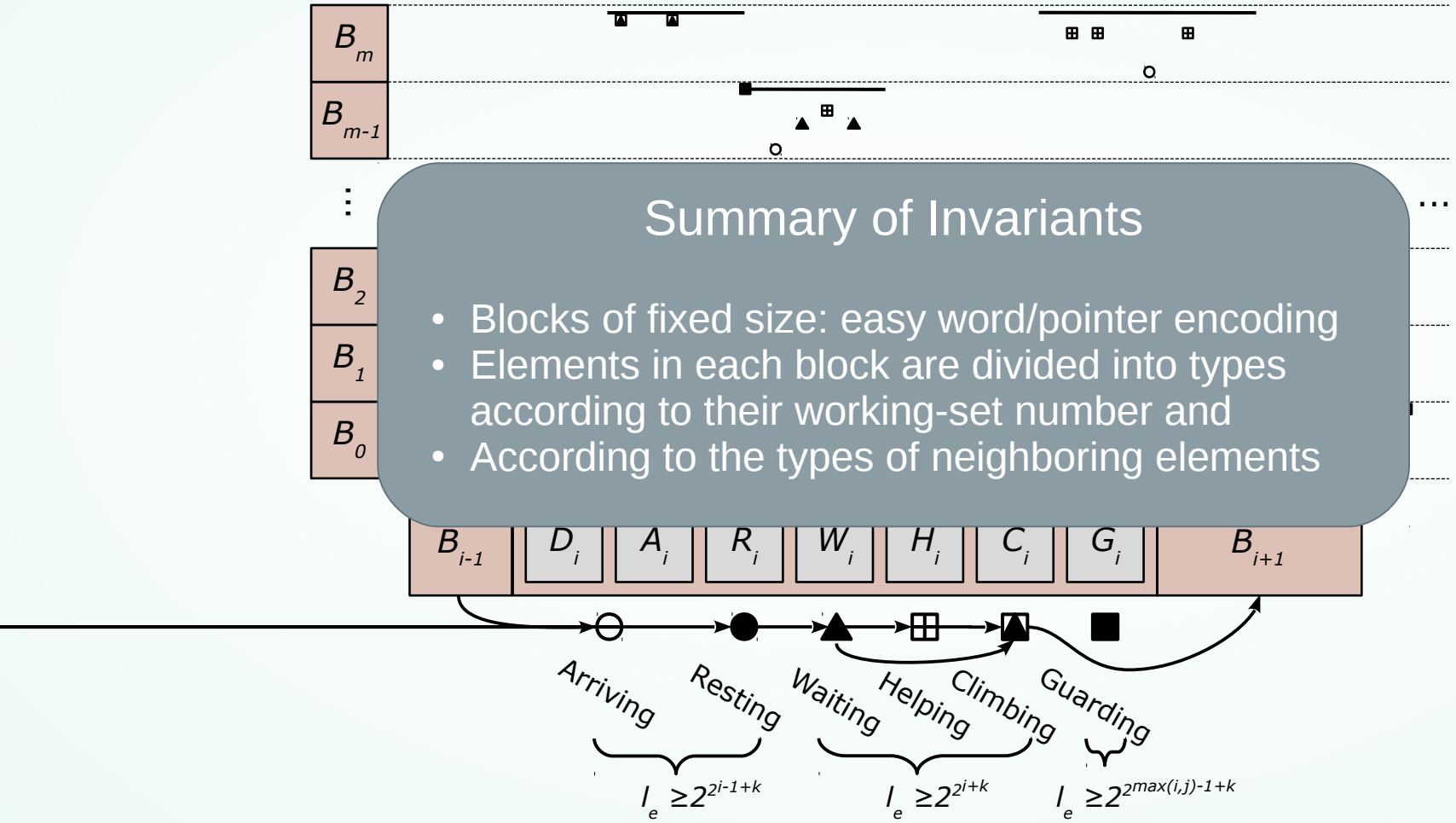
Implicit Representation of Intervals

- Representing the intervals implicitly



Implicit Representation of Intervals

- Representing the intervals implicitly



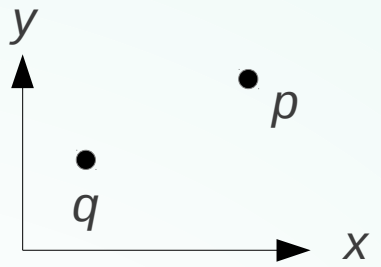
Summary of Invariants

- 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

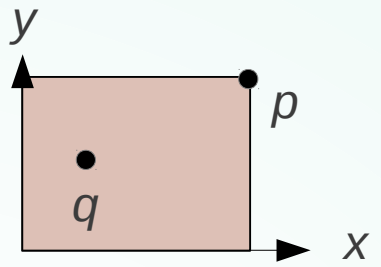
Outline

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

What are Skyline Queries?

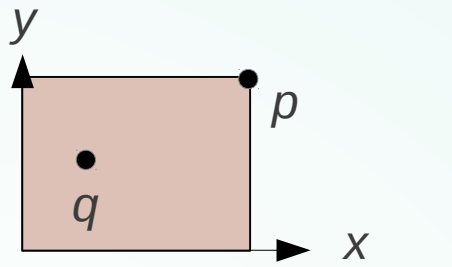


What are Skyline Queries?

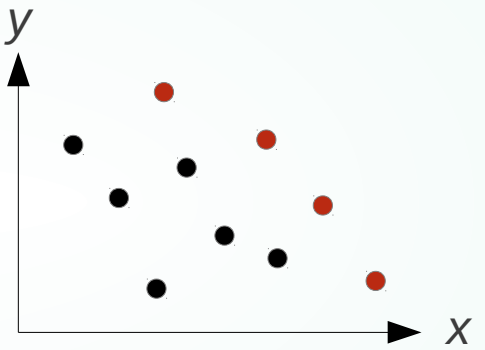


What are Skyline Queries?

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



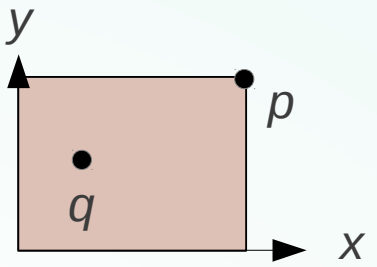
- The *maximal/skyline* points of a point set $P \subseteq \mathbb{R}^2$ are the undominated points



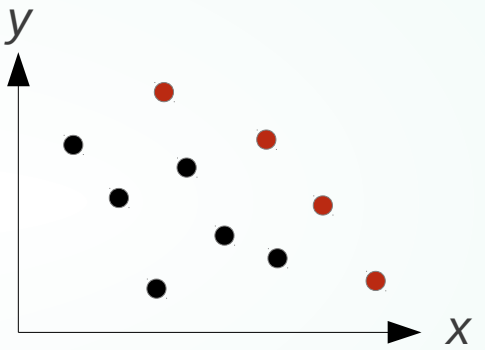
- Given a *dynamic* point set $P \subseteq \mathbb{R}^2$ we want to be able to find the skyline for a given query range $Q = [x_1,x_2] \times [y_1,y_2]$

What are Skyline Queries?

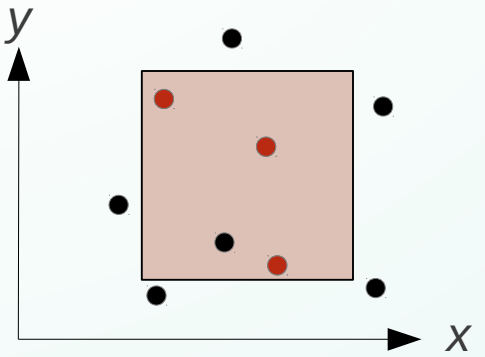
- Given two points $p,q \in P \subseteq \mathbb{R}^2$ we say p *dominates* q iff $p_x \geq q_x$ and $p_y \geq q_y$



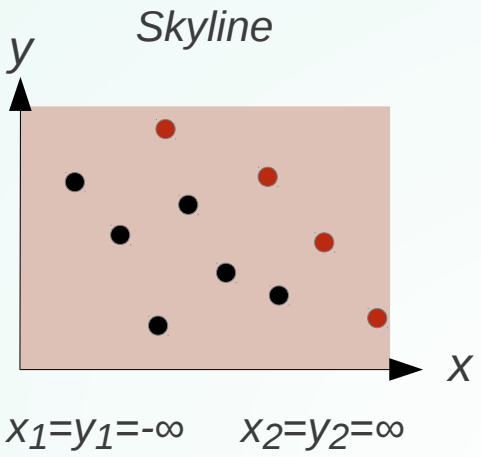
- The *maximal/skyline* points of a point set $P \subseteq \mathbb{R}^2$ are the undominated points



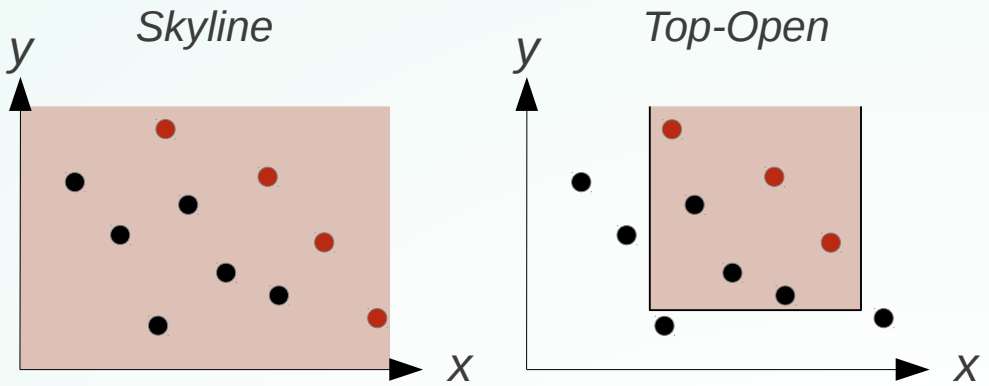
- Given a *dynamic* point set $P \subseteq \mathbb{R}^2$ we want to be able to find the skyline for a given query range $Q = [x_1,x_2] \times [y_1,y_2]$



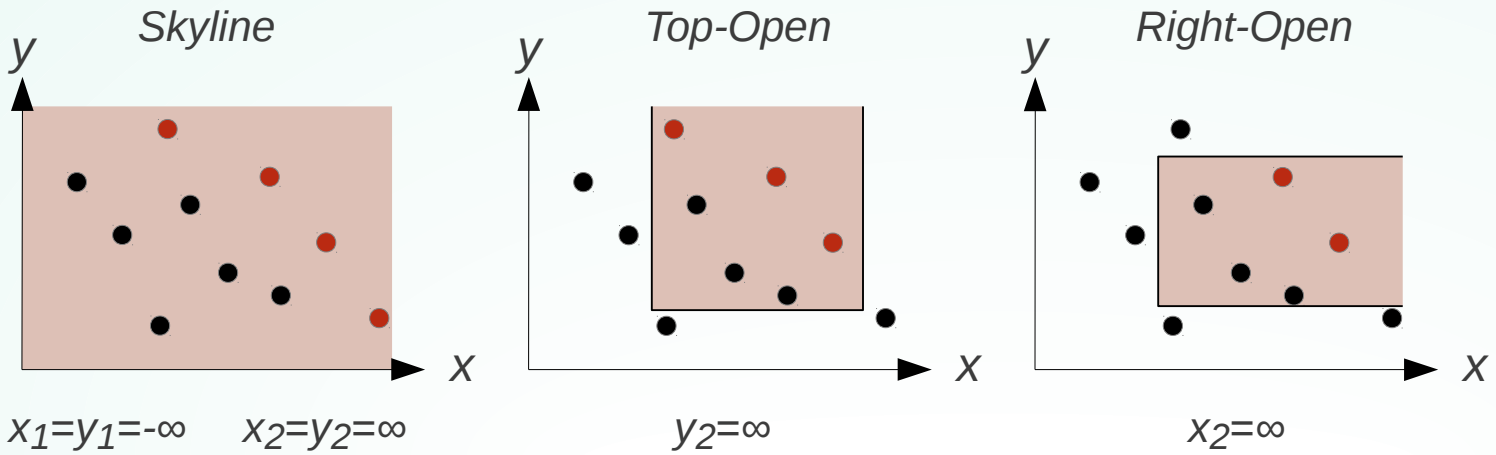
Special Cases of Skyline Queries



Special Cases of Skyline Queries

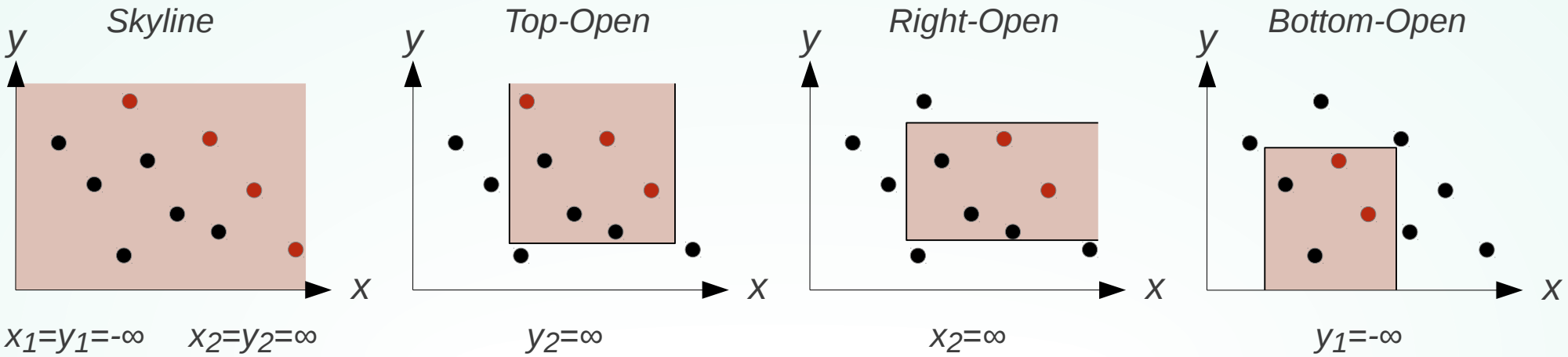


Special Cases of Skyline Queries

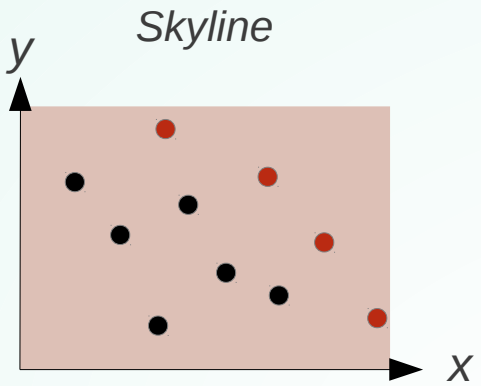


$x_1=y_1=-\infty$ $x_2=y_2=\infty$ $y_2=\infty$ $x_2=\infty$

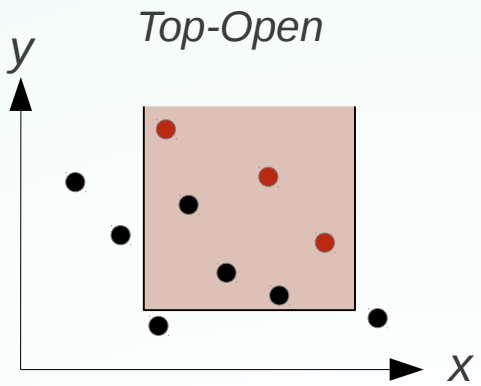
Special Cases of Skyline Queries



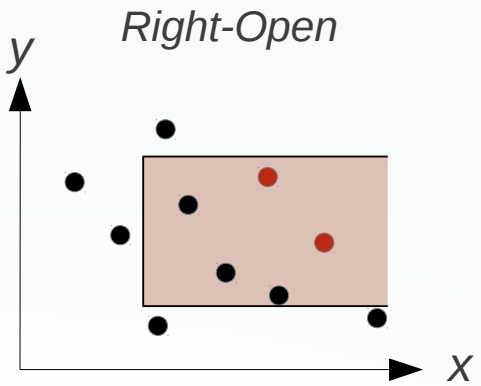
Special Cases of Skyline Queries



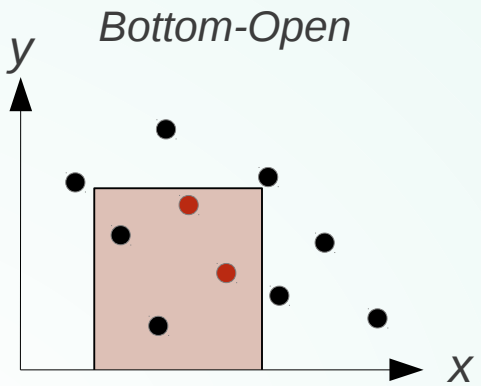
$x_1=y_1=-\infty$ $x_2=y_2=\infty$



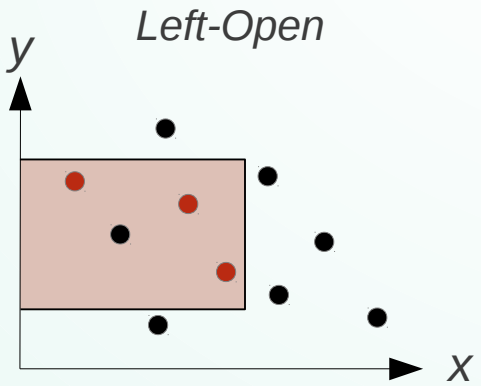
$y_2=\infty$



$x_2=\infty$

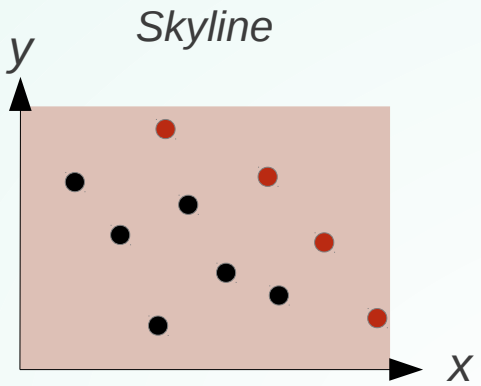


$y_1=-\infty$

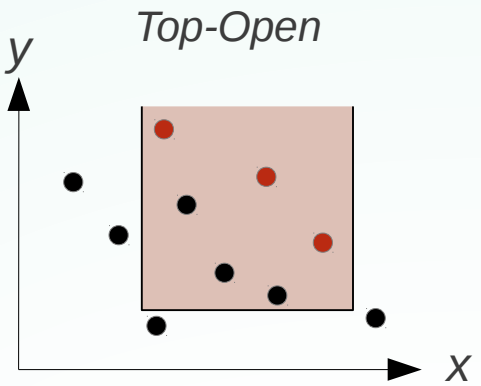


$x_1=-\infty$

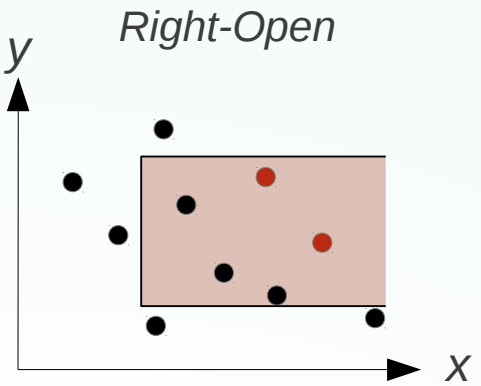
Special Cases of Skyline Queries



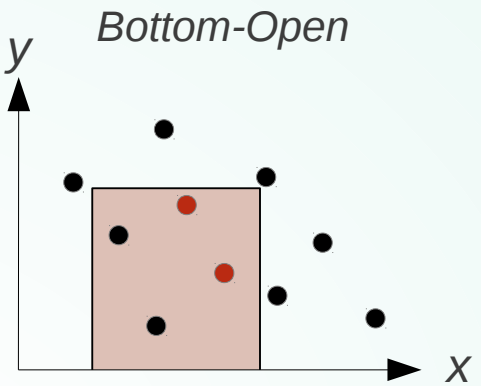
$x_1=y_1=-\infty$ $x_2=y_2=\infty$



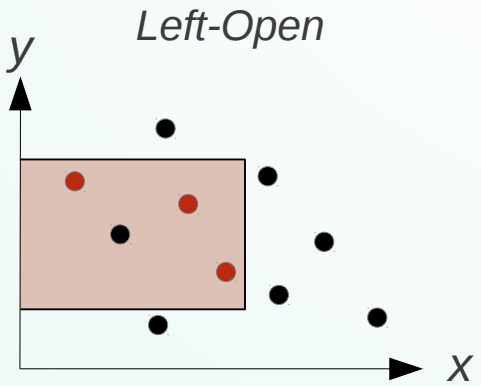
$y_2=\infty$



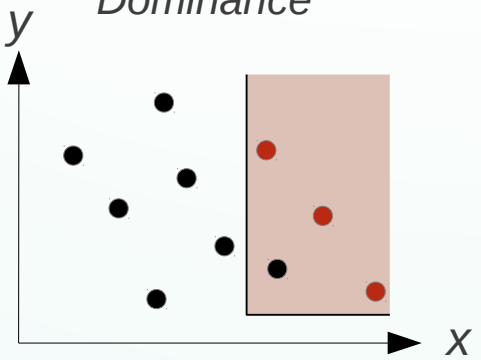
$x_2=\infty$



$y_1=-\infty$

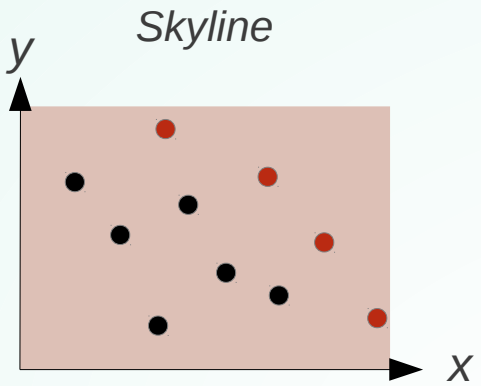


$x_1=-\infty$

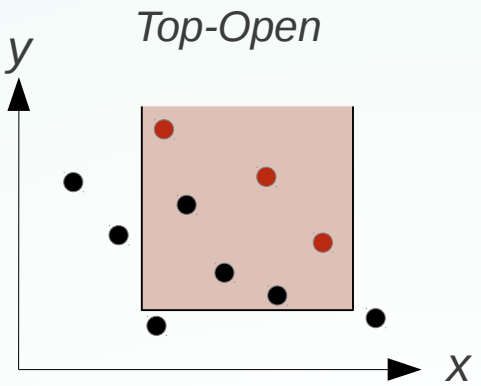


$x_2=y_2=\infty$

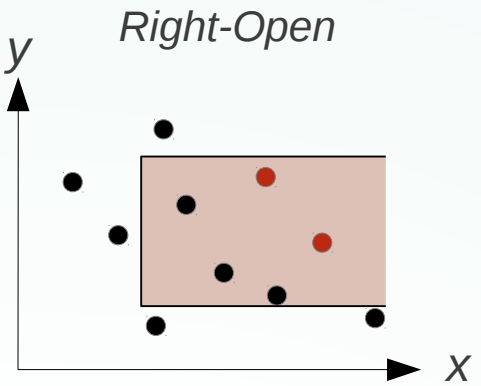
Special Cases of Skyline Queries



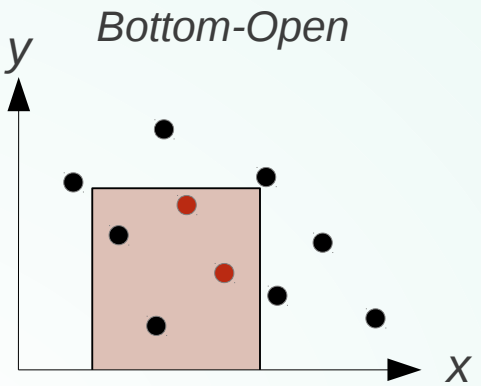
$x_1=y_1=-\infty$ $x_2=y_2=\infty$



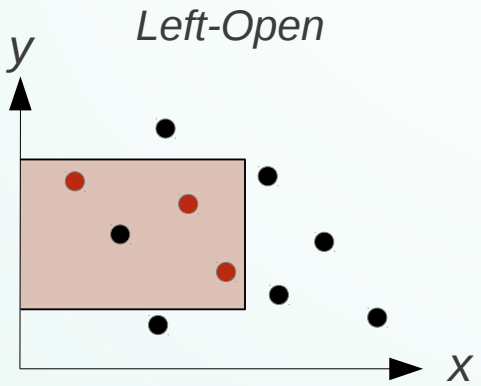
$y_2=\infty$



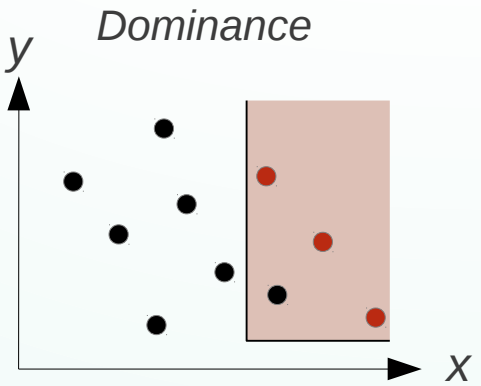
$x_2=\infty$



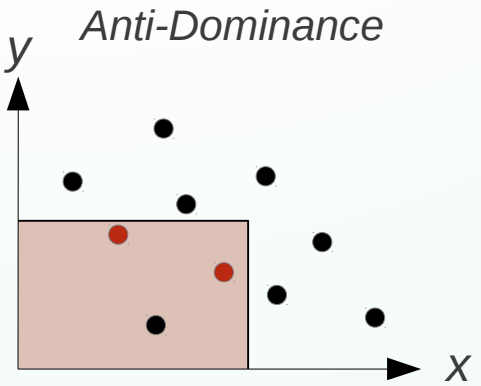
$y_1=-\infty$



$x_1=-\infty$

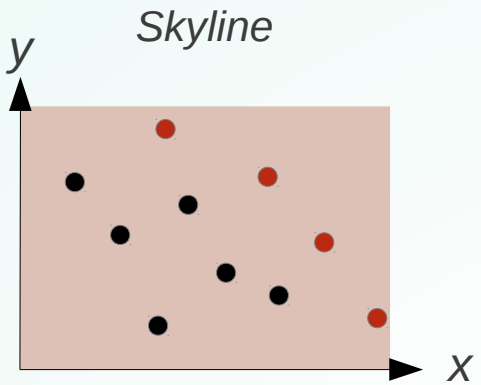


$x_2=y_2=\infty$

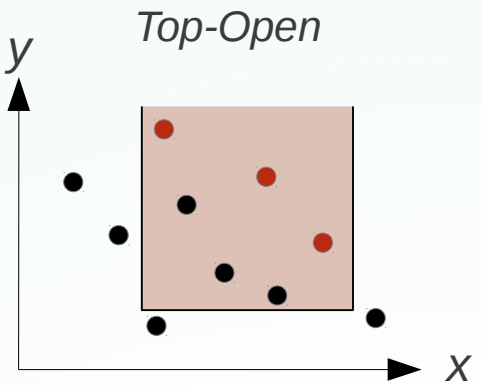


$x_1=y_1=-\infty$

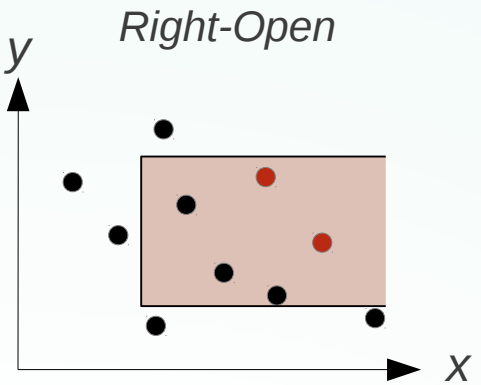
Special Cases of Skyline Queries



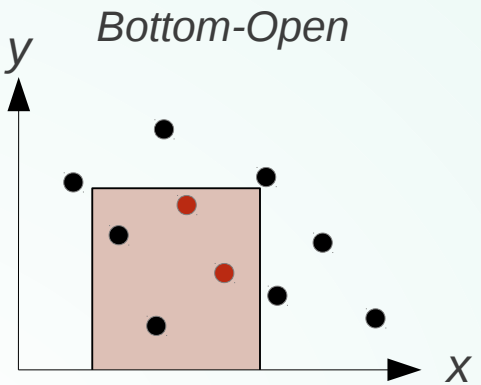
$x_1=y_1=-\infty$ $x_2=y_2=\infty$



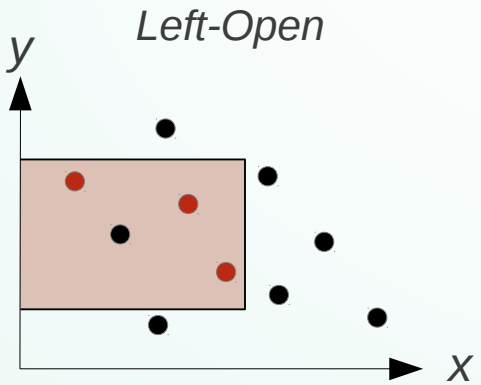
$y_2=\infty$



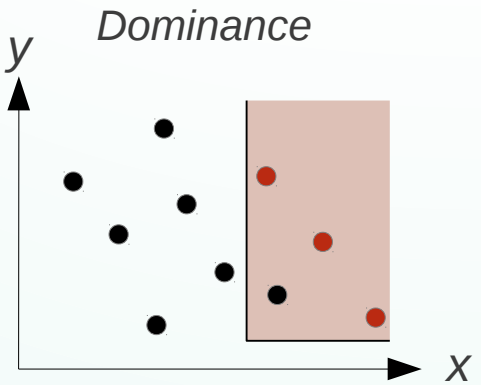
$x_2=\infty$



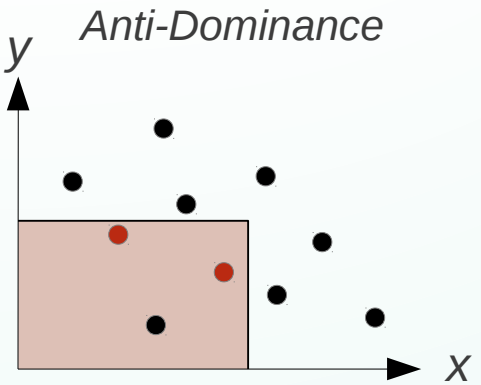
$y_1=-\infty$



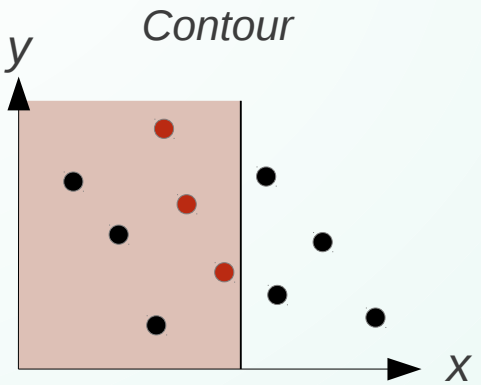
$x_1=-\infty$



$x_2=y_2=\infty$

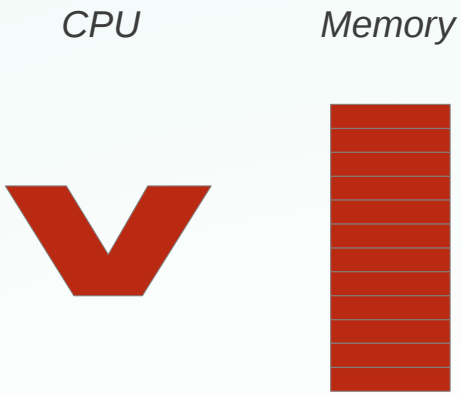


$x_1=y_1=-\infty$

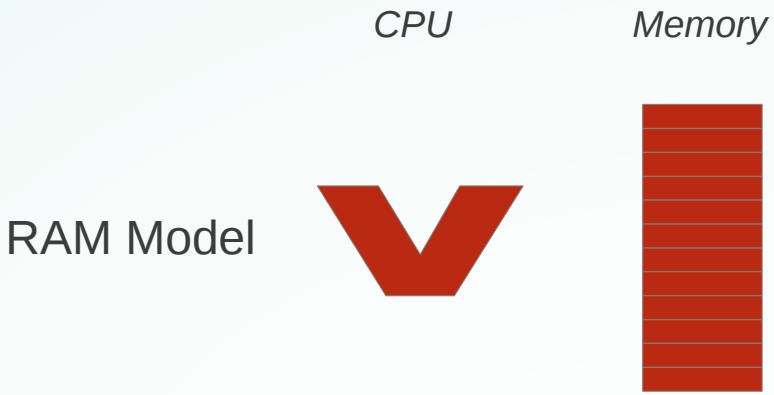


$x_1=y_1=-\infty$ $y_2=\infty$

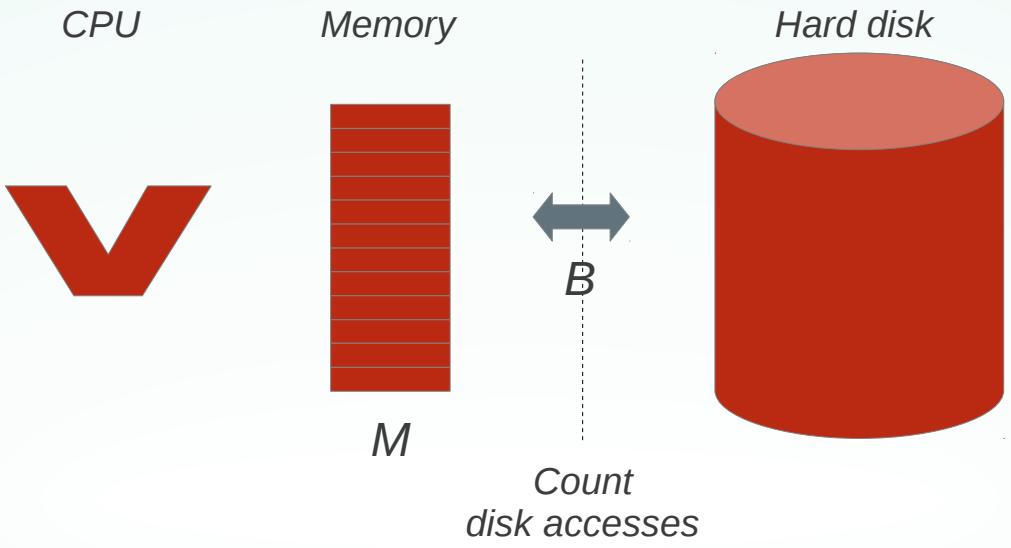
EM Model



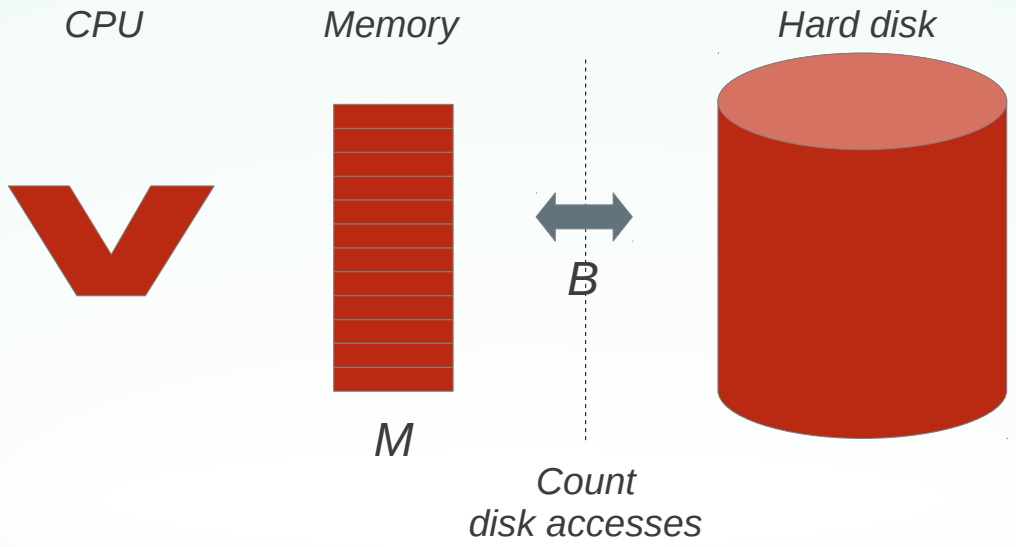
EM Model



EM Model

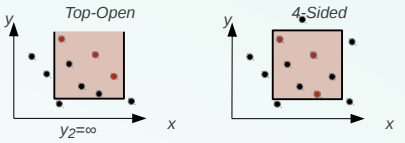


EM Model



- 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 spent $O(1/B)$ I/Os to access one element
- Scanning uses $O(n/B)$ I/Os and search trees uses $O(\log_B n)$ I/Os

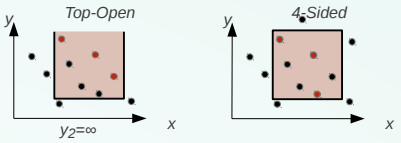
Previous and Our Results



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

*** Assumes pre-sorting**

Previous and Our Results

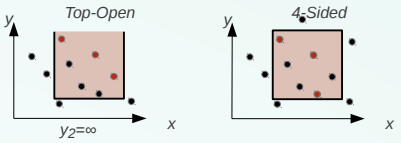


Problem	Space	Pre-proces	Query	Query	Update	Domain	Source
Top-Open	$O(n)$	$O(n \log n)$	$O(1+k)$	-	-	$\mathbb{R}$	YA10
Top-Open	$O(n)$	$O(n \log n)$	$O(\log n+k)$	-	$O(\log n)$	$\mathbb{R}$	BT11
Top-Open	$O(n)$	$O(n \log n/\log\log n)$	$O(\log n/\log\log n+k)$	-	$O(\log n/\log\log n)$	$\mathbb{R}$	BT11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

Indexability Model
Indivisibility Assumption

*** Assumes pre-sorting**

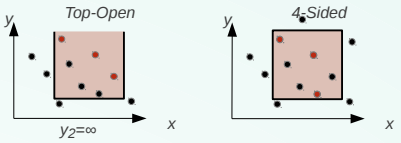
Previous and Our Results



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

*** Assumes pre-sorting**

Previous and Our Results

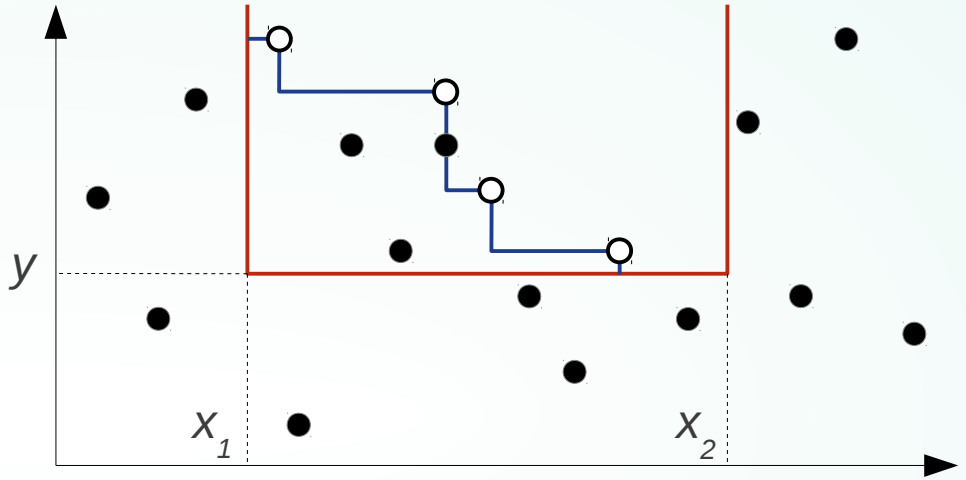


Problem	Space	Pre-proces	Query	Query	Update	Domain	Source
Top-Open	$O(n)$	$O(n \log n)$	$O(1+k)$	-	-	$\mathbb{R}$	YA10
Top-Open	$O(n)$	$O(n \log n)$	$O(\log n+k)$	-	$O(\log n)$	$\mathbb{R}$	BT11
Top-Open	$O(n)$	$O(n \log n/\log\log n)$	$O(\log n/\log\log n+k)$	-	$O(\log n/\log\log n)$	$\mathbb{R}$	BT11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

For Today!

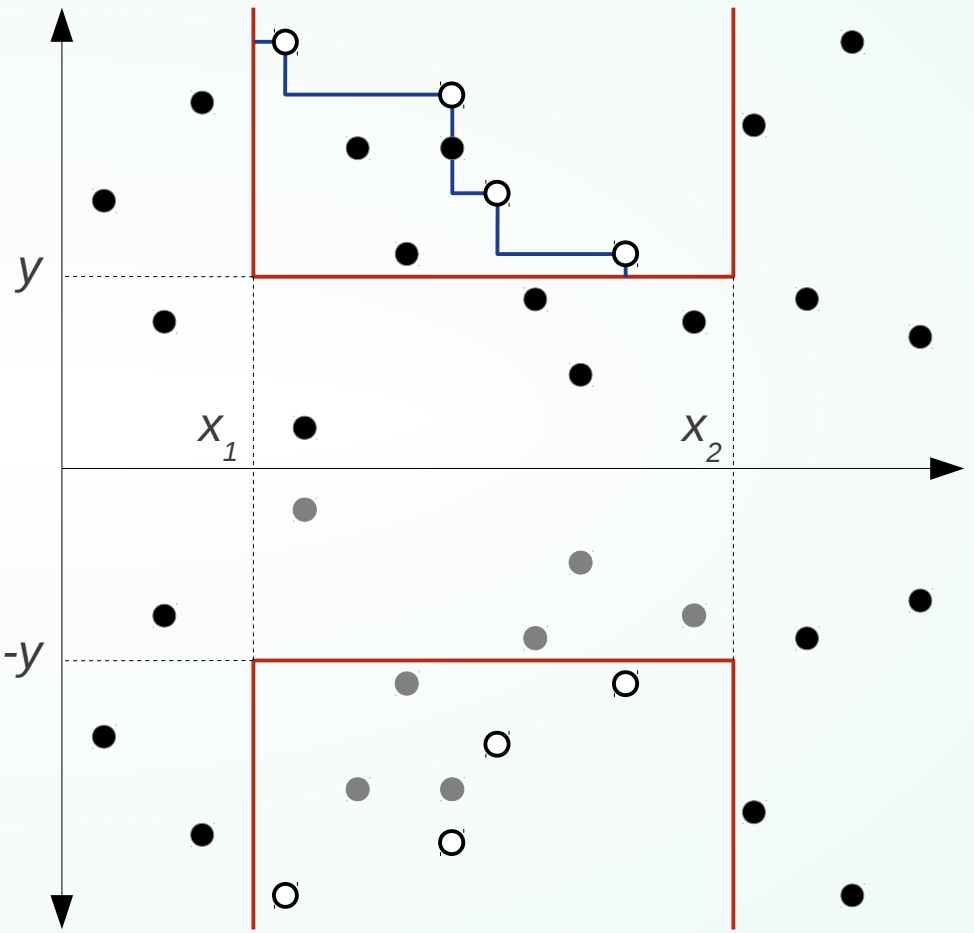
* Assumes pre-sorting

Observation for Top-Open Queries



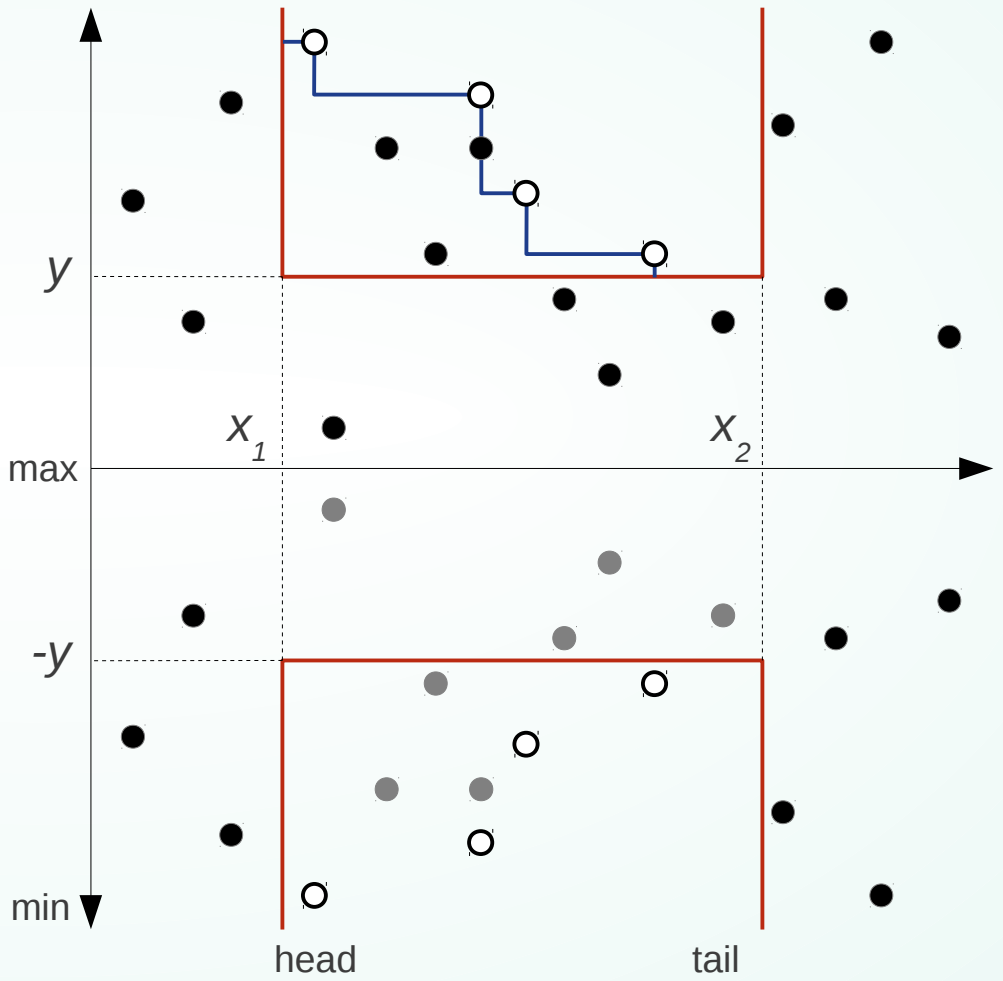
Observation for Top-Open Queries

- Consider mirroing 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



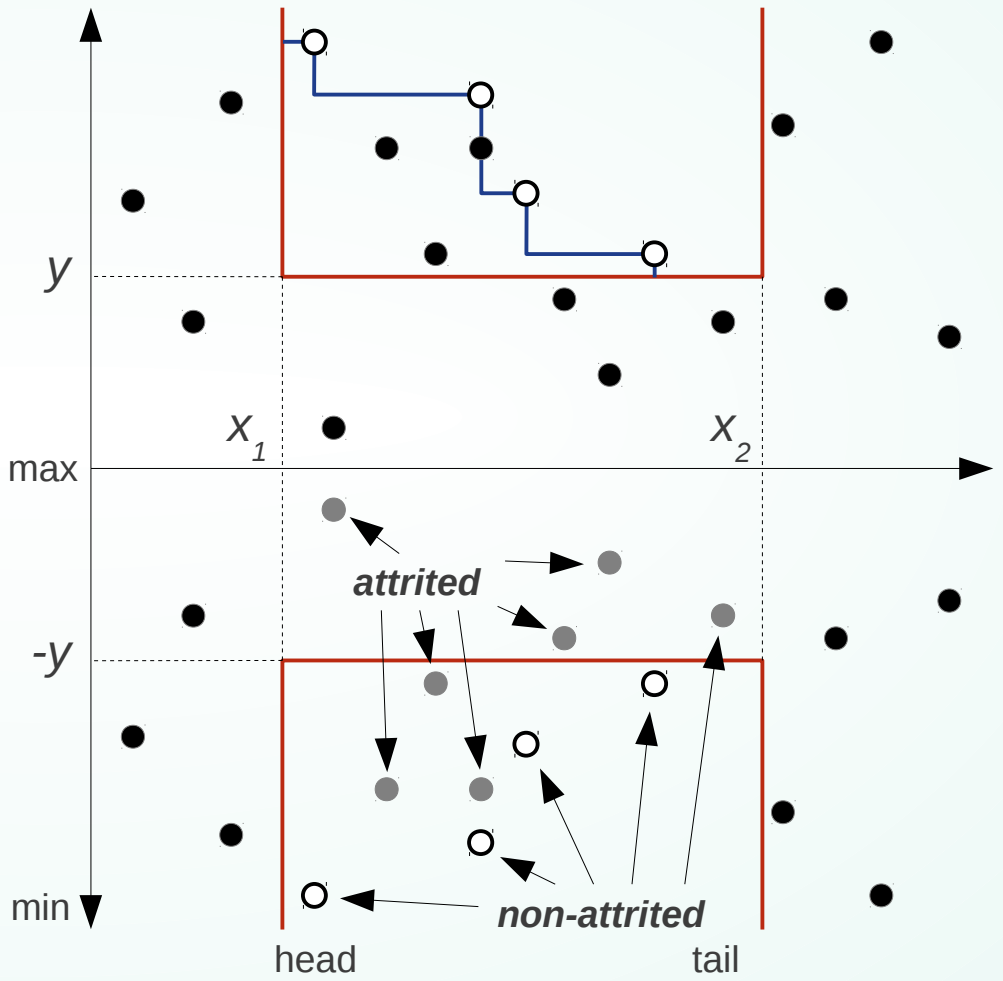
Observation for Top-Open Queries

- Consider mirroing 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

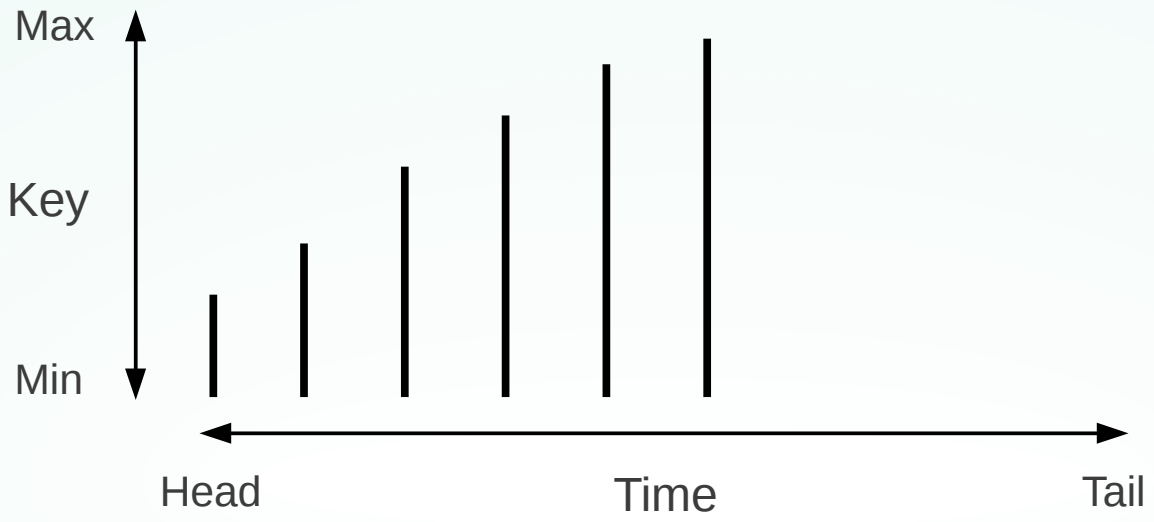


Observation for Top-Open Queries

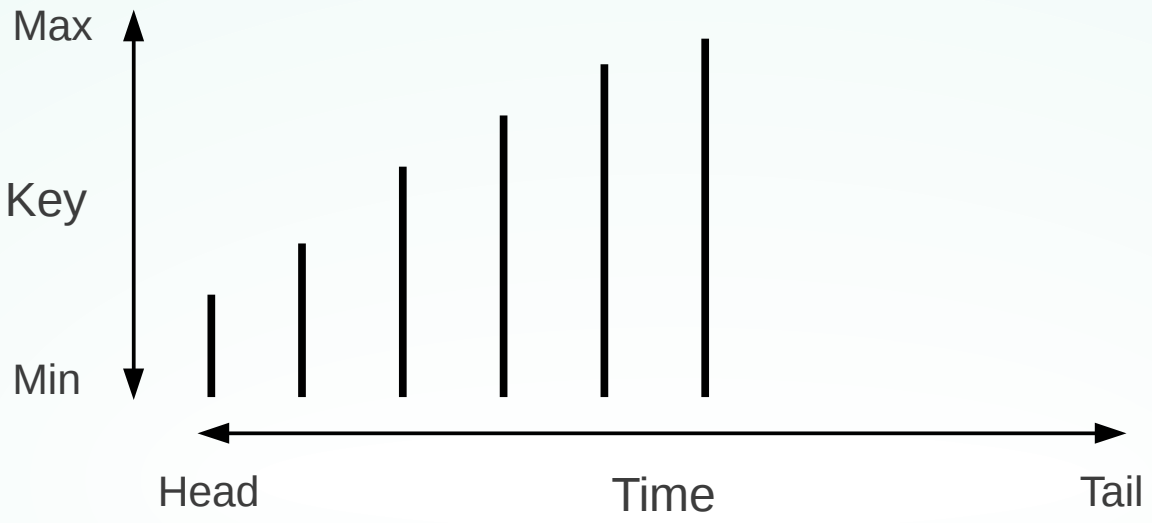
- Consider mirroing the point set in the *y*-axis
- Let *x* represent the insertion time and *y* the key space
- When inserting element *e* it deletes/*attrites* all elements inserted before it with a larger key
- Non-attrited points and undominated points are equivalent



Priority Queues with Attrition

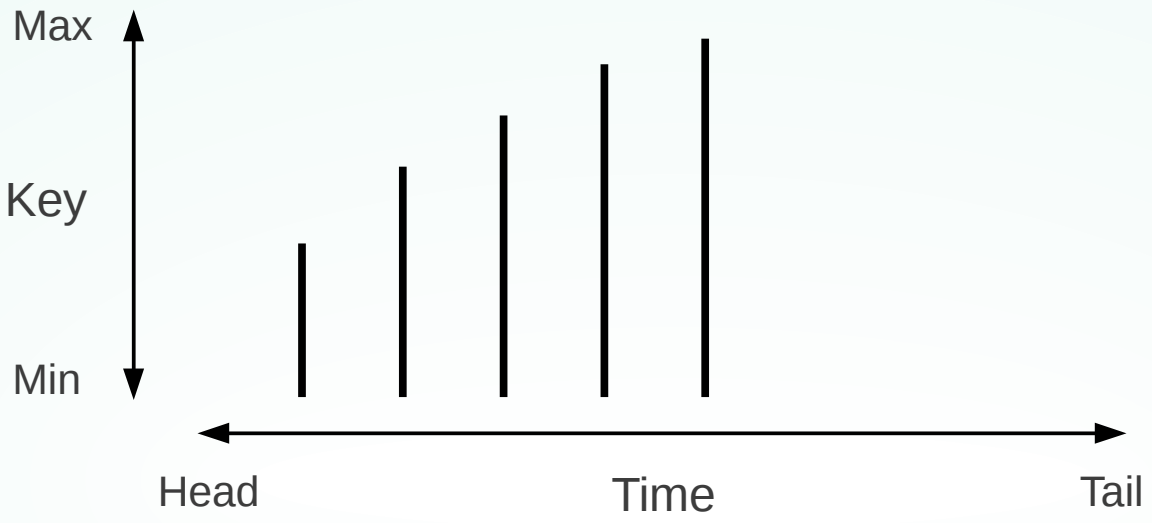


Priority Queues with Attrition



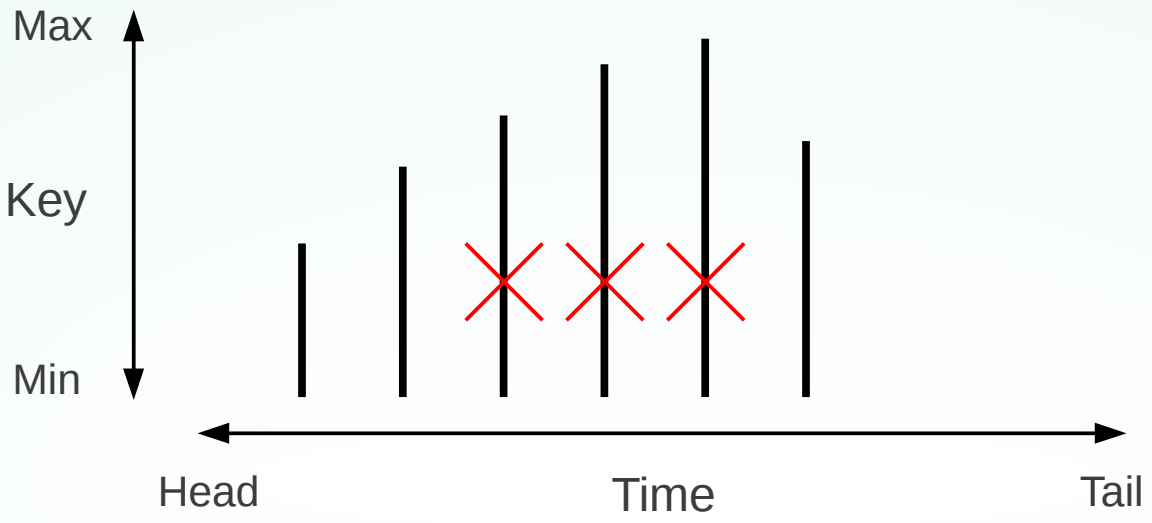
- DeleteMin(): Deletes the head/minimum element of the queue
- InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2

Priority Queues with Attrition



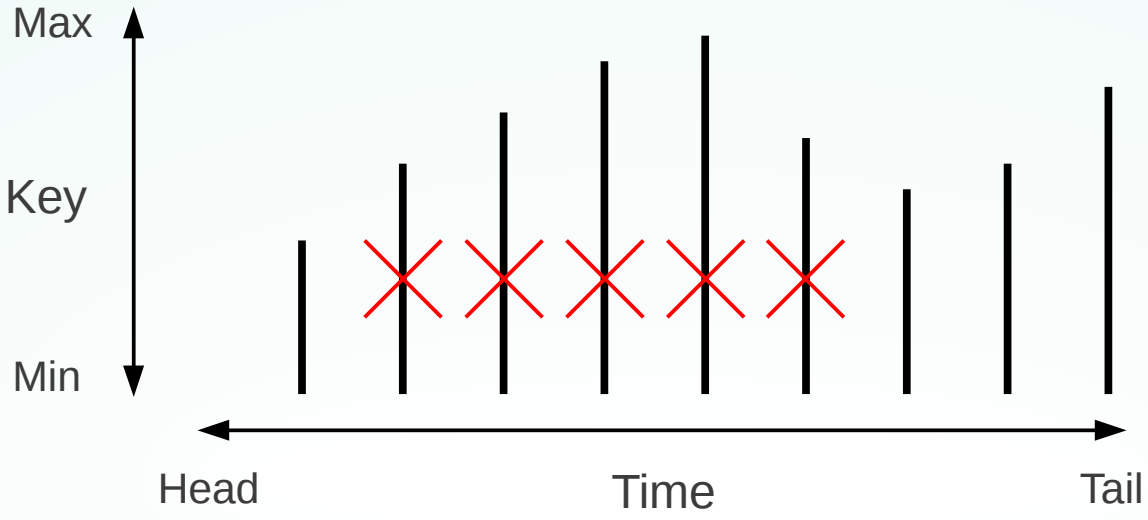
- DeleteMin(): Deletes the head/minimum element of the queue
- InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2

Priority Queues with Attrition



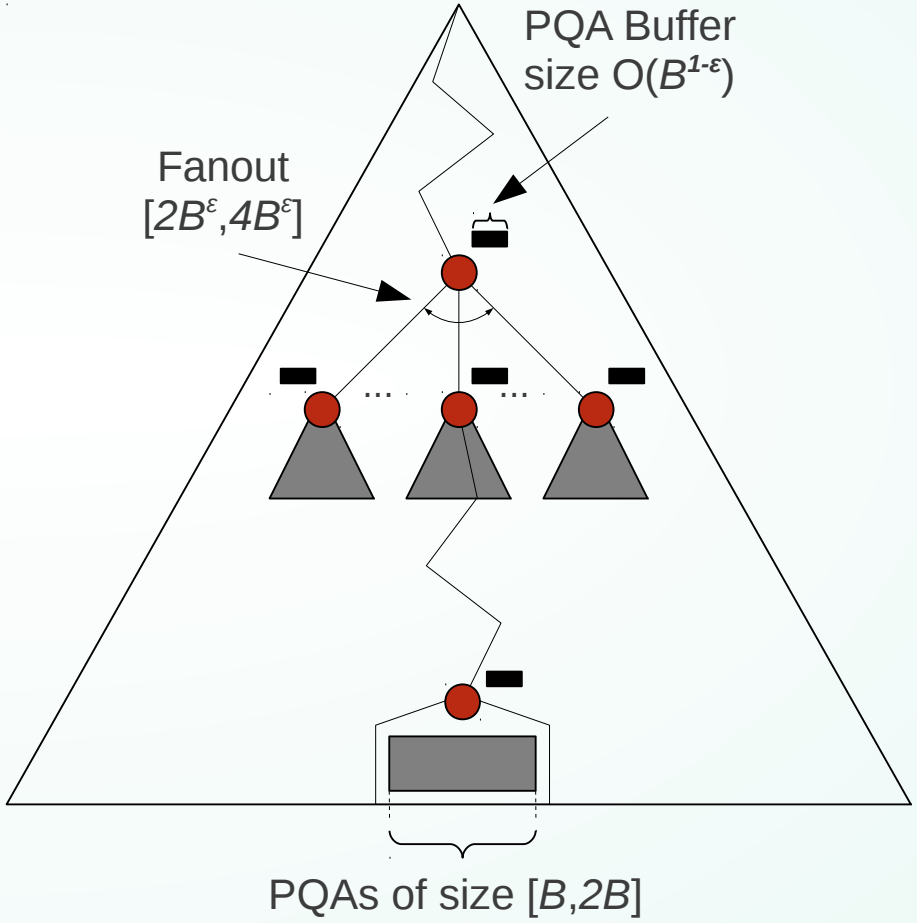
- DeleteMin(): Deletes the head/minimum element of the queue
- InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2

Priority Queues with Attrition



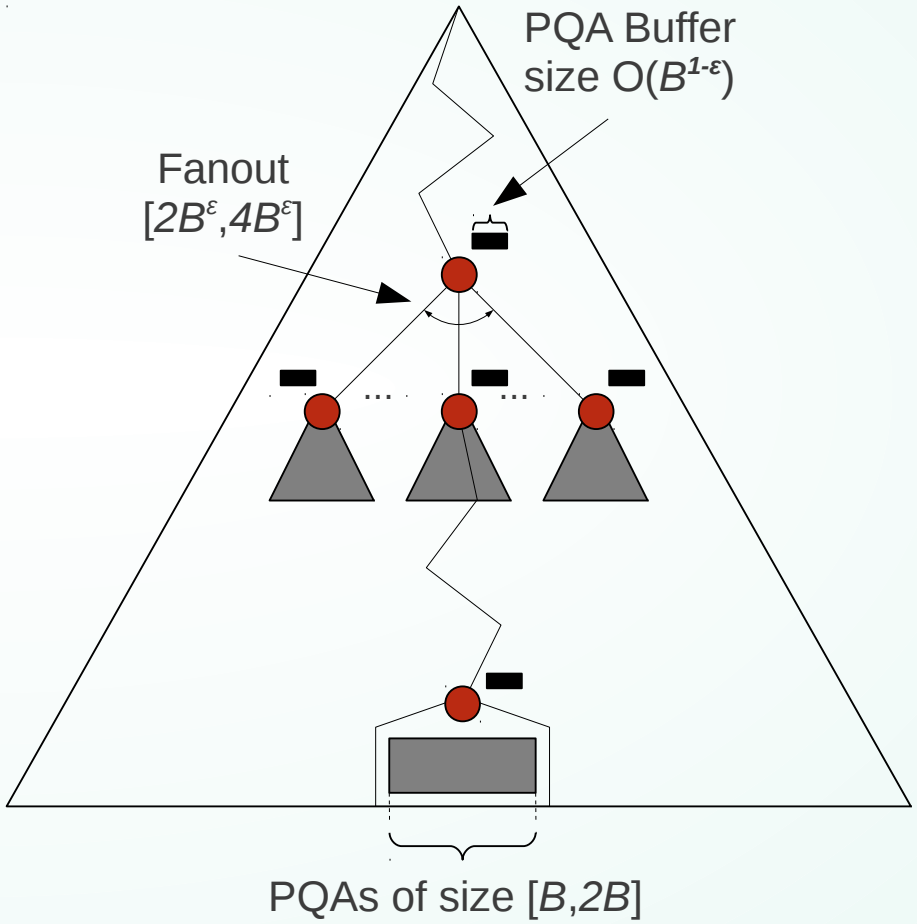
- DeleteMin(): Deletes the head/minimum element of the queue
- InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2

Data Structure and Invariants



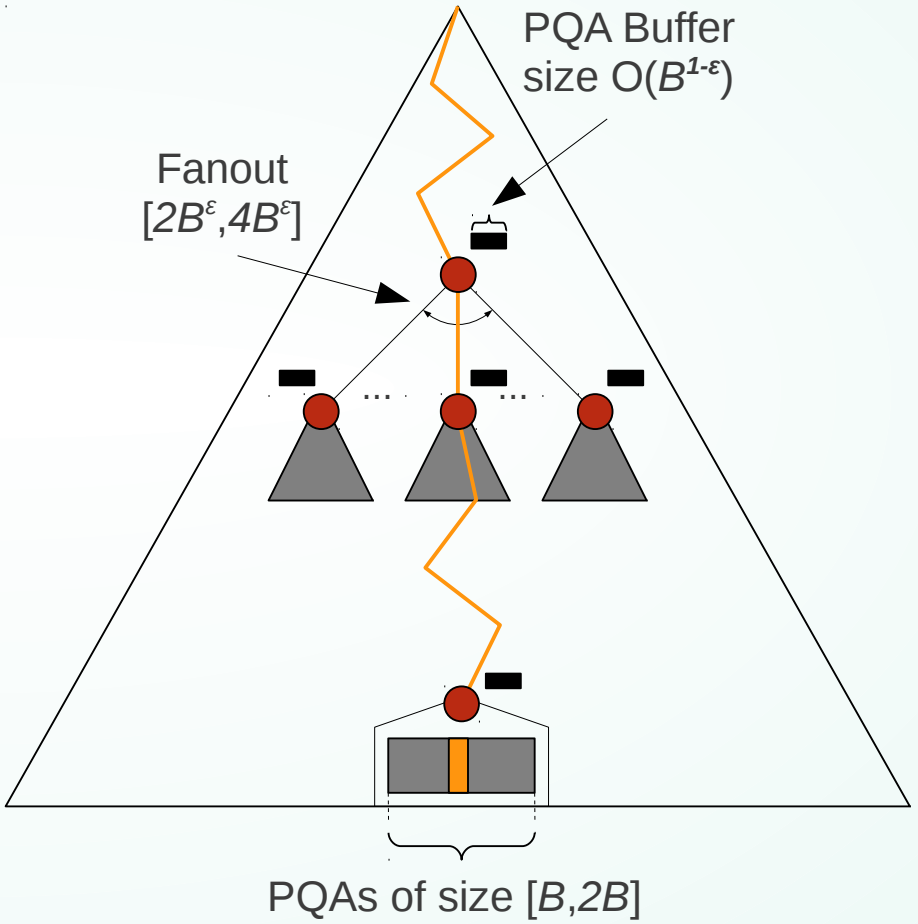
Data Structure and Invariants

- 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
- Leafs 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



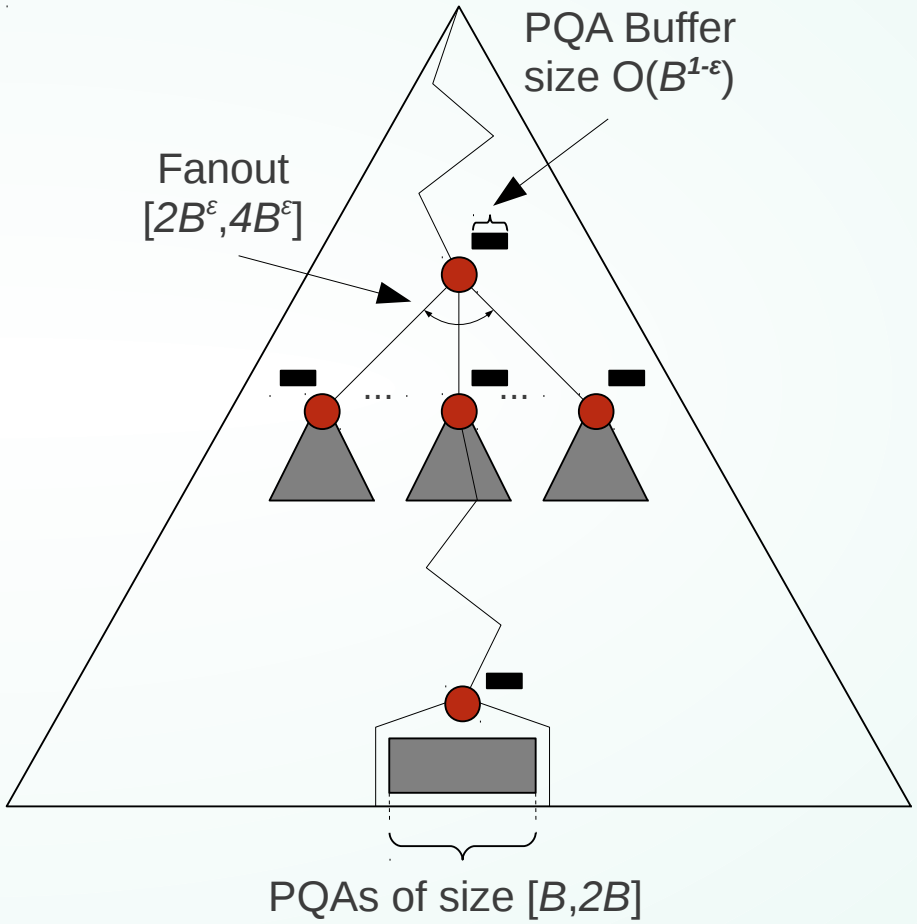
Data Structure and Invariants

- 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 childrens PQAs
- Leafs 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 childrens PQAs
- Leafs 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

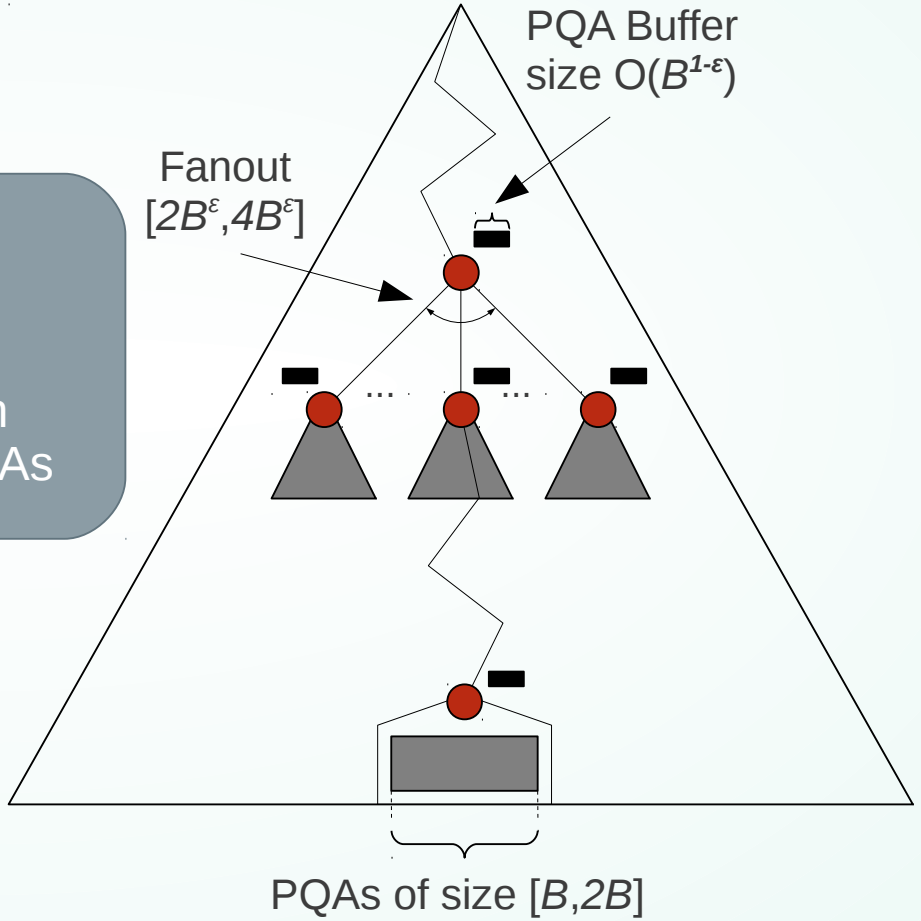


Data Structure and Invariants

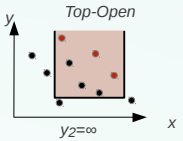
- A $(2B^{\varepsilon},4B^{\varepsilon})$-tree augmented with PQAs
- Internal node have between $2B^{\varepsilon}$ a
  stor
  conc
  PQA
- Leaf
  to $2B$ elements it contains
- Updating element e discards the PQAs on the path to e and rebuilds them again

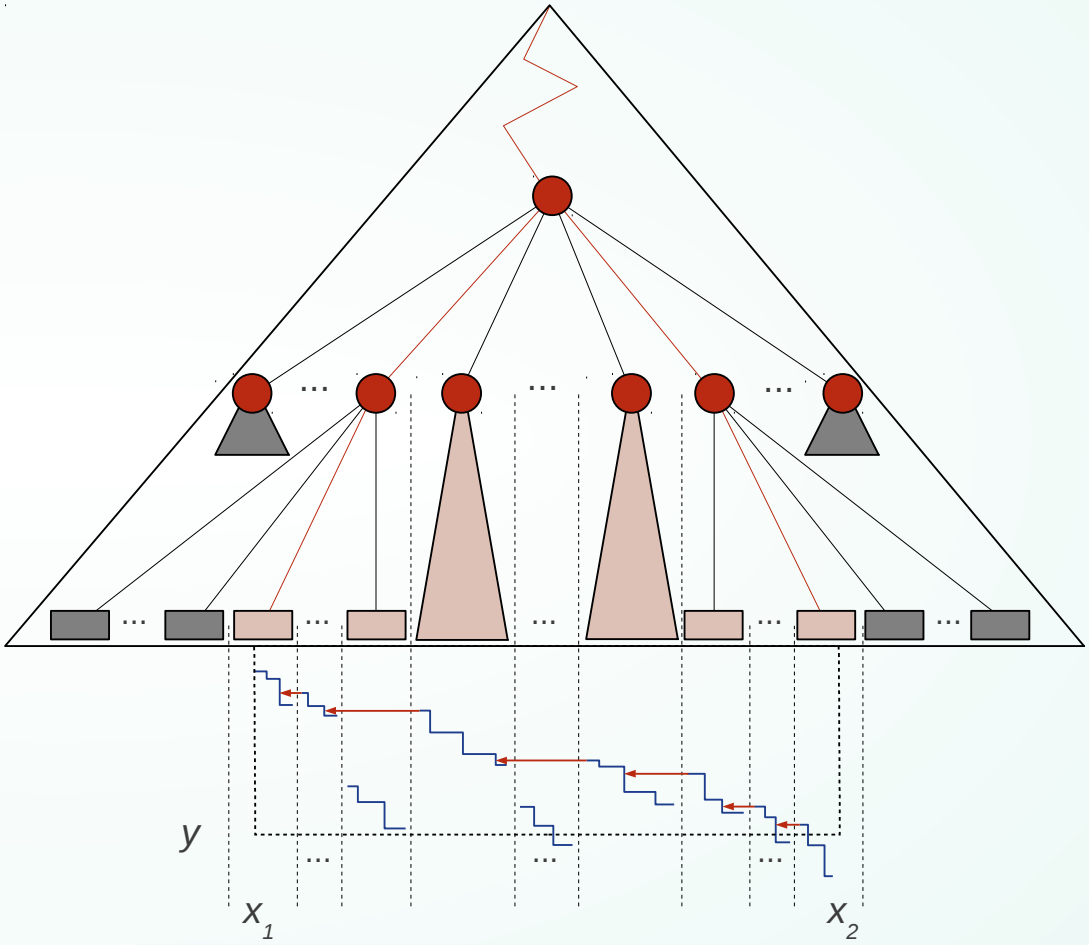
Summary of Invariants

- Nodes have a bounded degree
- All leaves have the same depth
- Nodes are augmented with PQAs

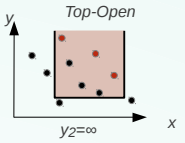


Top-Open Skyline Queries

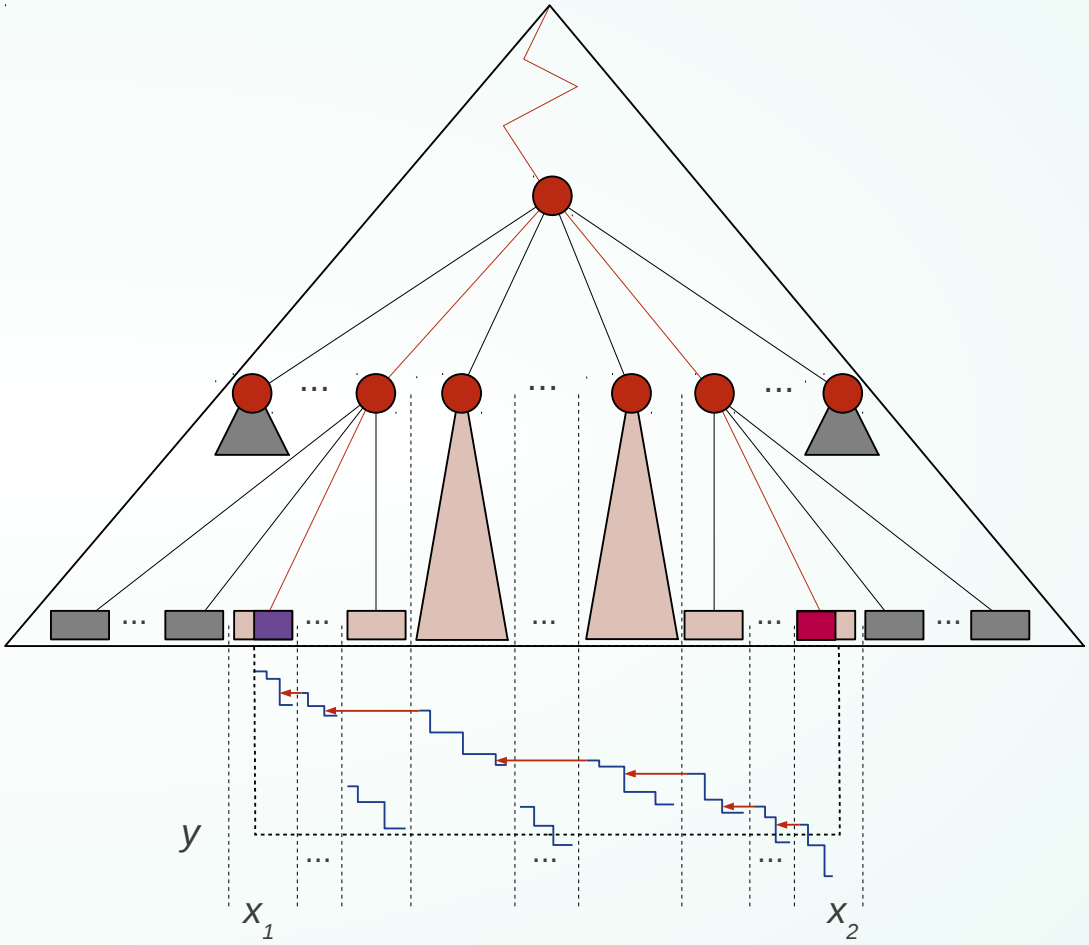




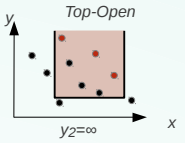
Top-Open Skyline Queries



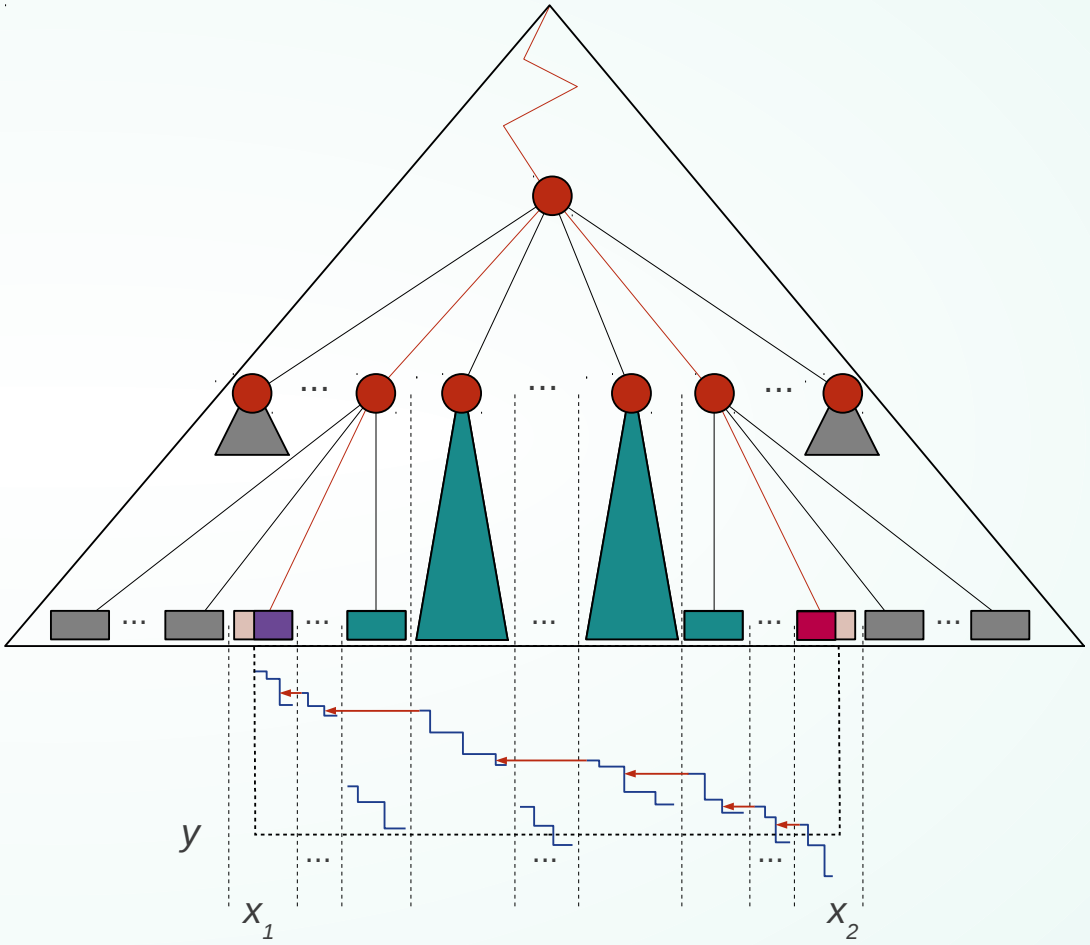
- We find the leafs of x_1 and x_2 and make two PQAs of the elements within $[x_1,x_2]$ called Q_1 and Q_2
- We concatenate Q_1, all PQAs of subtrees inside $[x_1,x_2]$ and Q_2 into one PQA Q (Divide and conquer)
- We call DeleteMin on Q and report the returned element e unless e has y-value larger than $-y$



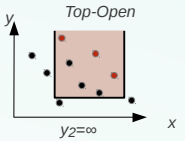
Top-Open Skyline Queries



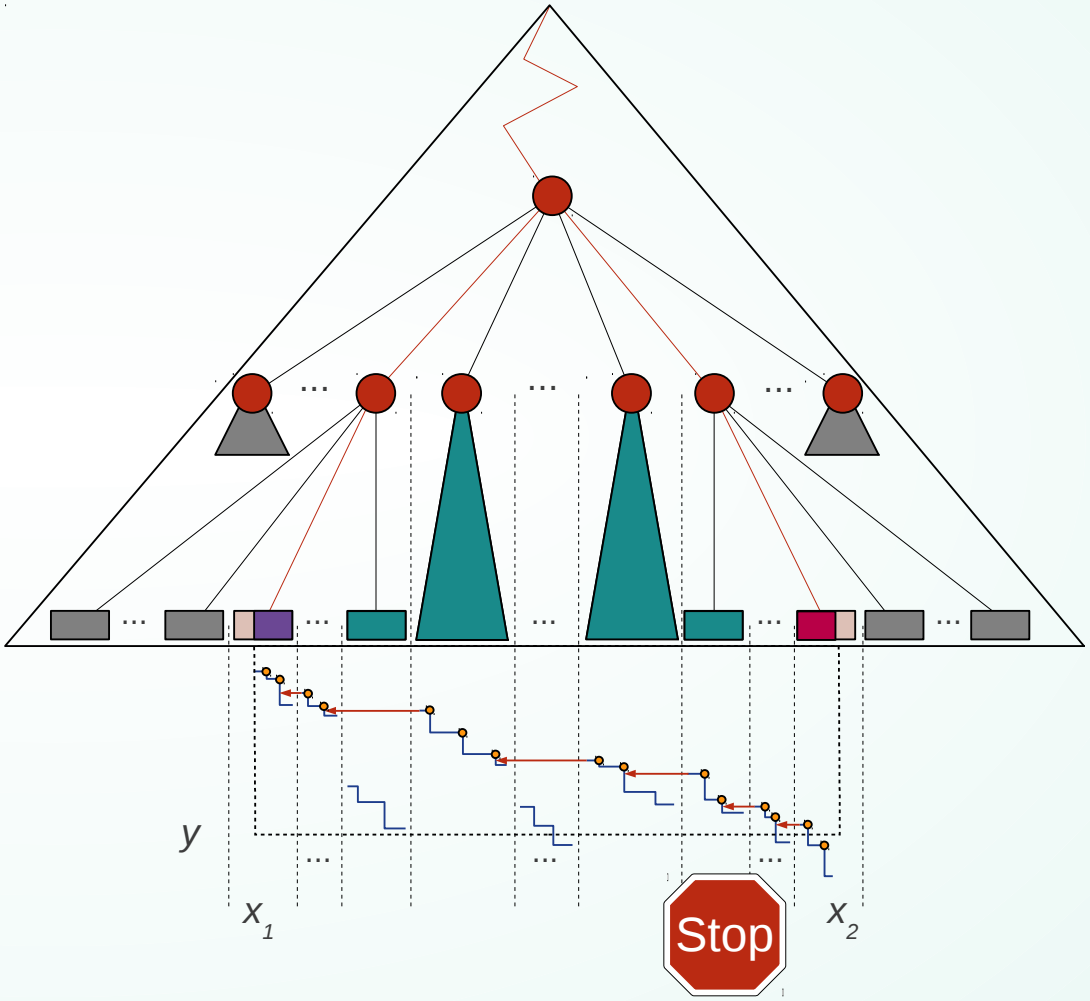
- We find the leafs of x_1 and x_2 and make two PQAs of the elements within $[x_1,x_2]$ called Q_1 and Q_2
- We concatenate Q_1, all PQAs of subtrees inside $[x_1,x_2]$ and Q_2 into one PQA Q (Divide and conquer)
- We call DeleteMin on Q and report the returned element e unless e has y-value larger than $-y$



Top-Open Skyline Queries



- We find the 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 $-y$



Outline

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

Previous and Our Results

Authors	Find-Min	Delete-Min	Insert-And-Attrite	Catenate-And-Attrite	Model
Sun89	O(*1*)	O(*1*)	O(*1*)		PM
Sun89	O(*1*)	O(*1*)	O(*1*)		EM
KTTTY13	O(*1*)	O(*1*)	O(*1*)	O(*1*)	PM
KTTTY13	O(*1/B*)	O(*1/B*)	O(*1/B*)	O(*1/B*)	EM

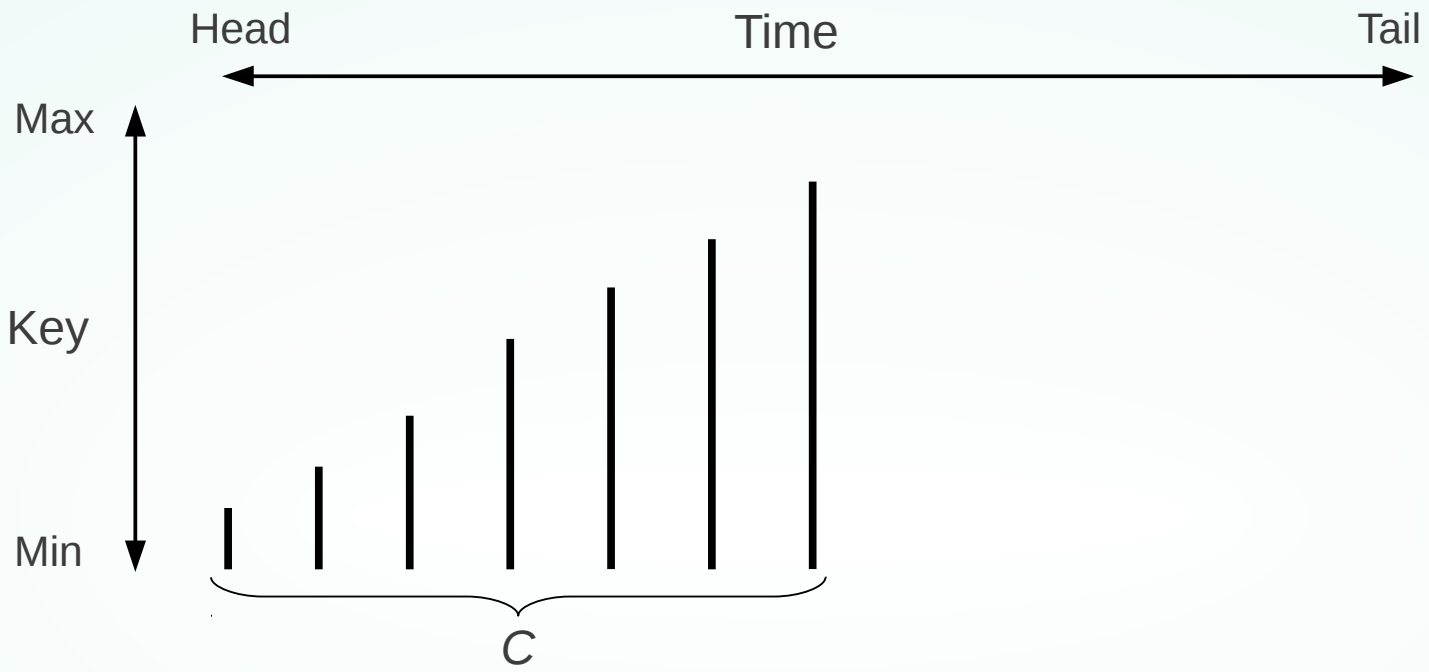
Previous and Our Results

Authors	Find-Min	Delete-Min	Insert-And-Attrite	Catenate-And-Attrite	Model
Sun89	O(*1*)	O(*1*)	O(*1*)		PM
Sun89	O(*1*)	O(*1*)	O(*1*)		EM
KTTTY13	O(*1*)	O(*1*)	O(*1*)	O(*1*)	PM
KTTTY13	O(*1/B*)	O(*1/B*)	O(*1/B*)	O(*1/B*)	EM

- 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

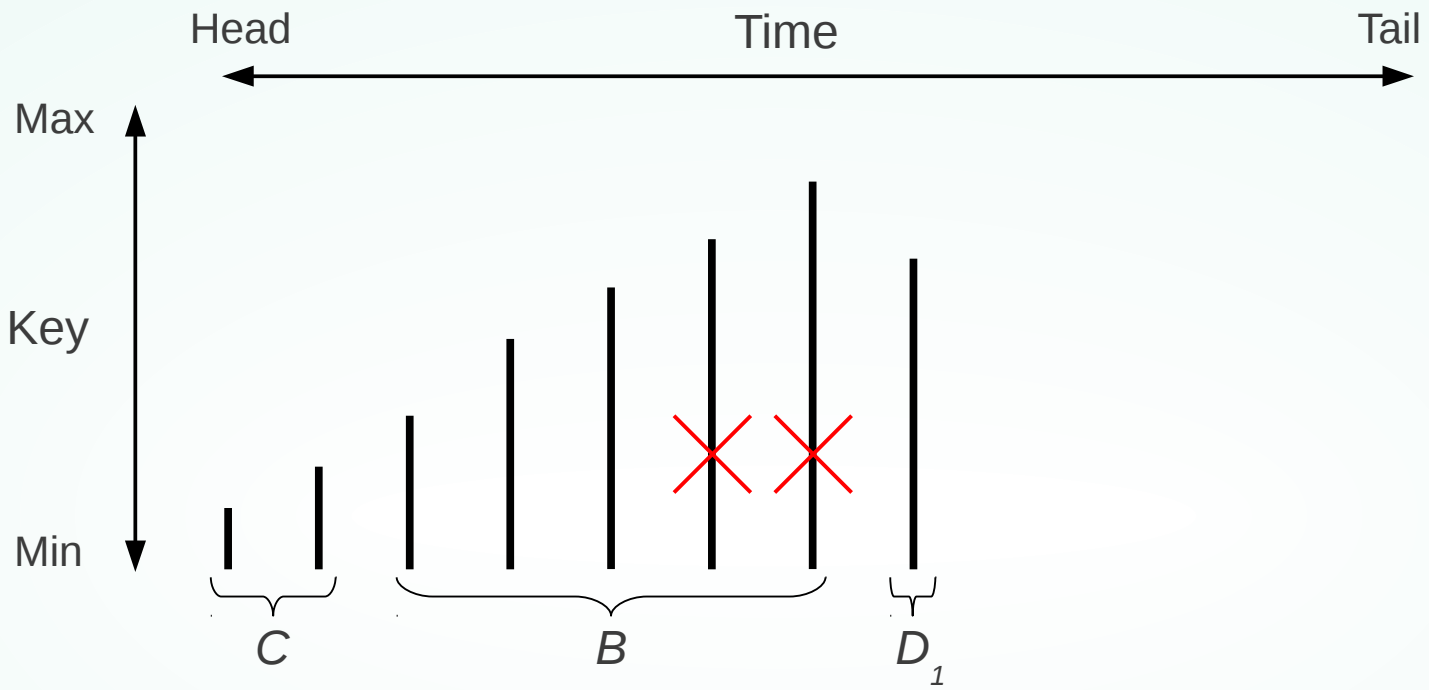
Observations and Invariants

Observations and Invariants



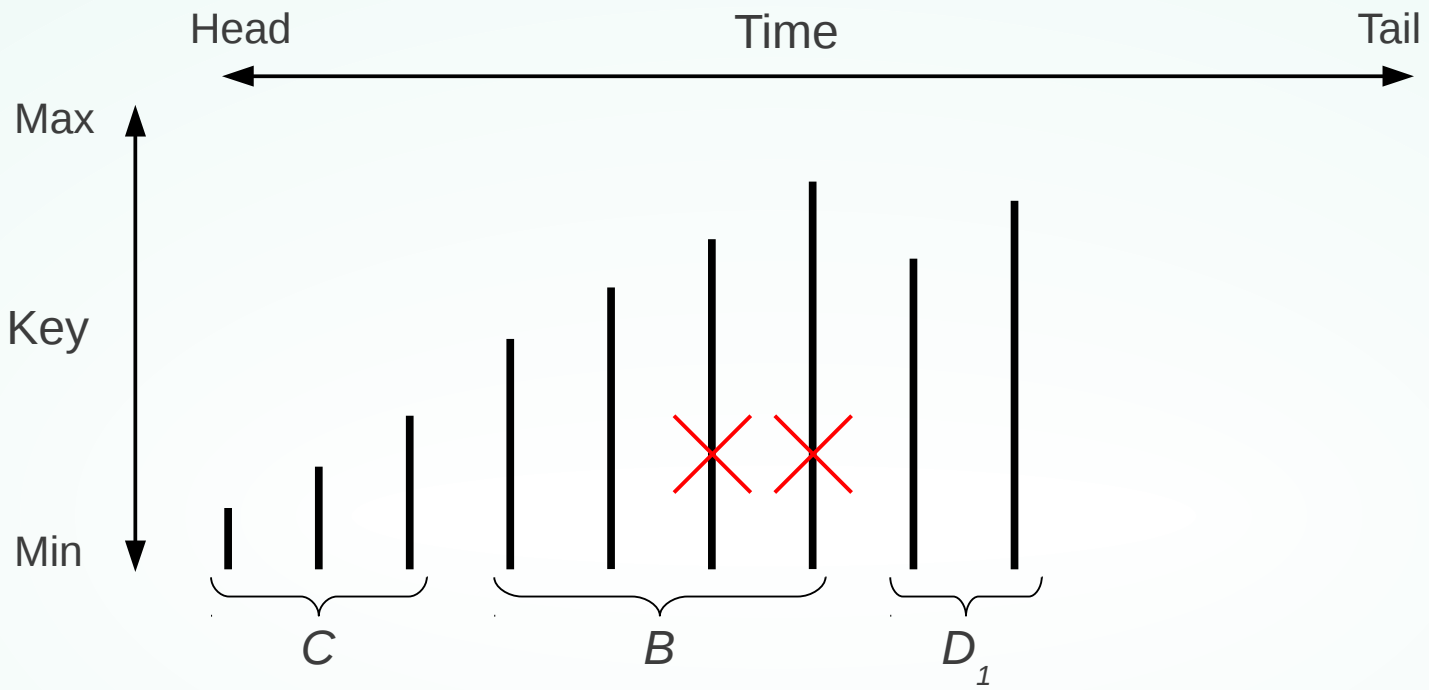
- InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element e of the queue Q

Observations and Invariants



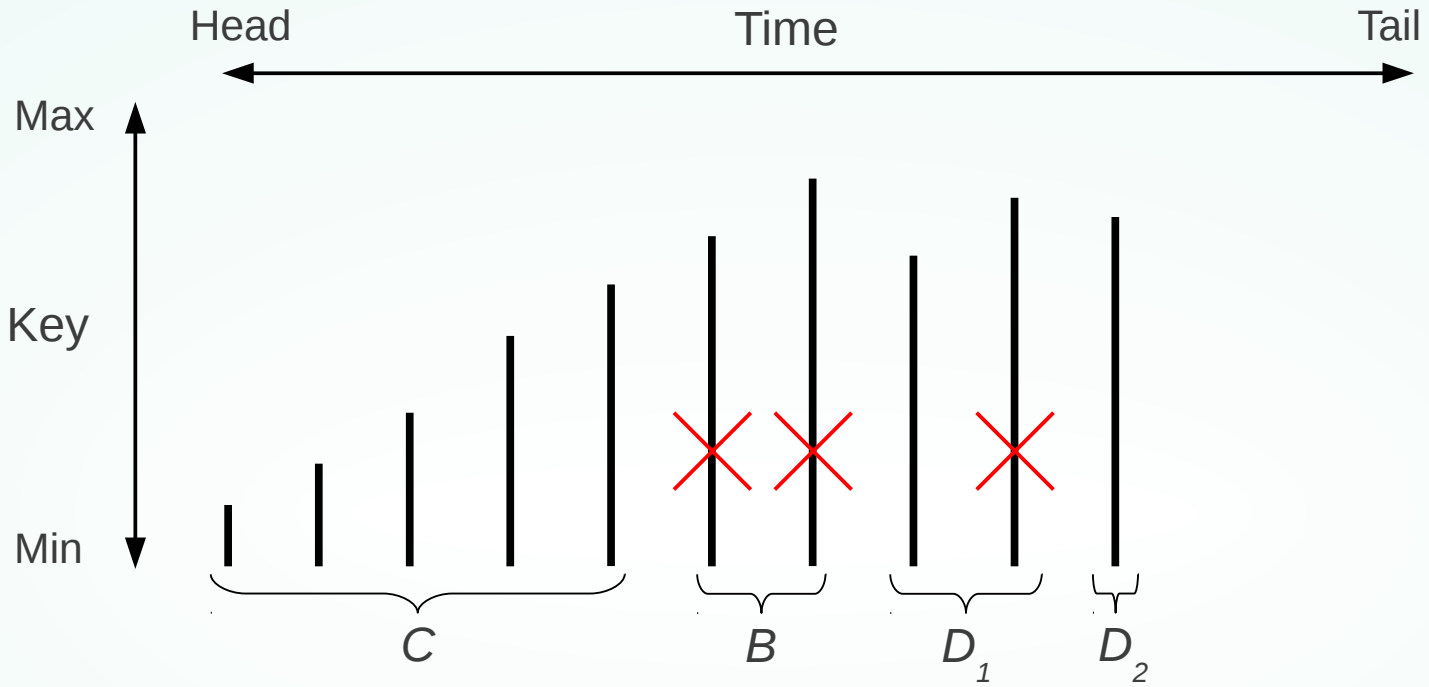
- InsertAndAttrite(*e*): Inserts *e* and deletes/attrites all elements before *e* with a key larger or equal to *e*
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element *e* of the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

Observations and Invariants



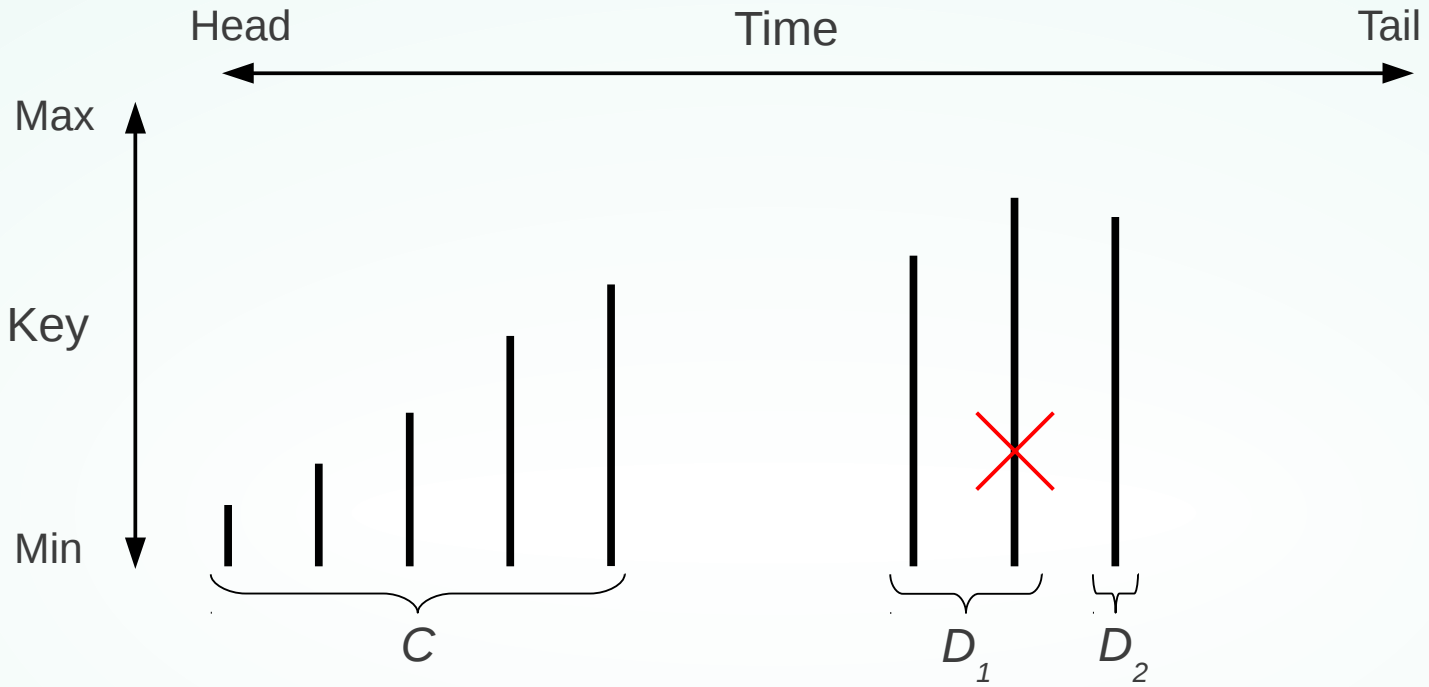
- InsertAndAttrite(e): Inserts e and deletes/attrites all elements before e with a key larger or equal to e
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element e of the queue Q

Observations and Invariants



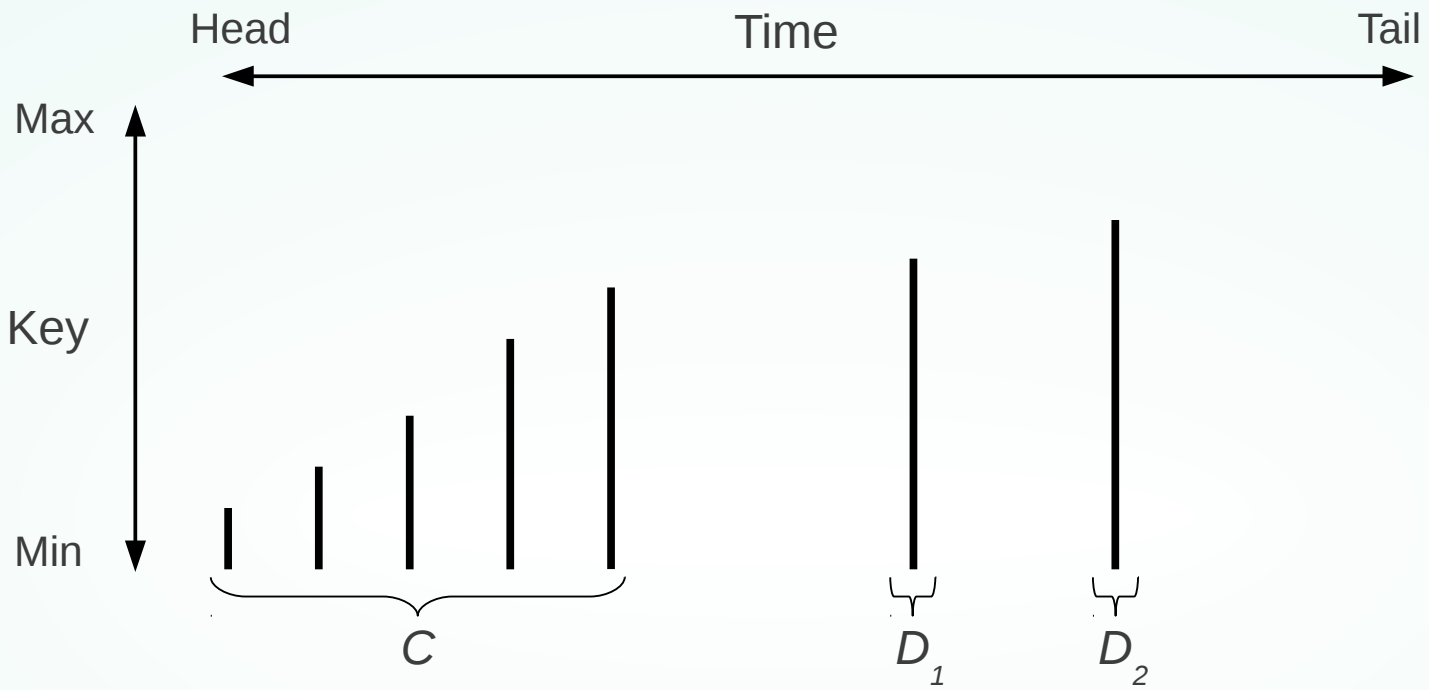
- InsertAndAttrite(*e*): Inserts *e* and deletes/attrites all elements before *e* with a key larger or equal to *e*
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element *e* of the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

Observations and Invariants



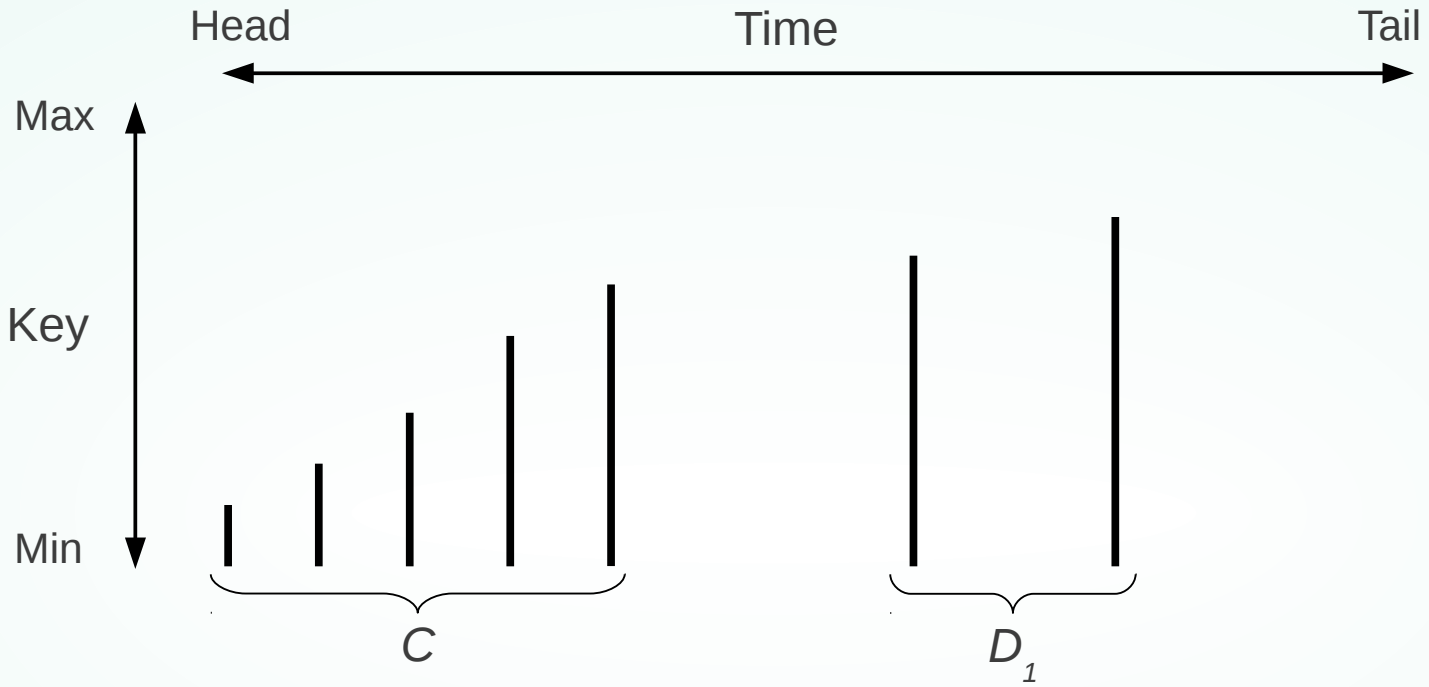
- InsertAndAttrite(*e*): Inserts *e* and deletes/attrites all elements before *e* with a key larger or equal to *e*
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element *e* of the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

Observations and Invariants



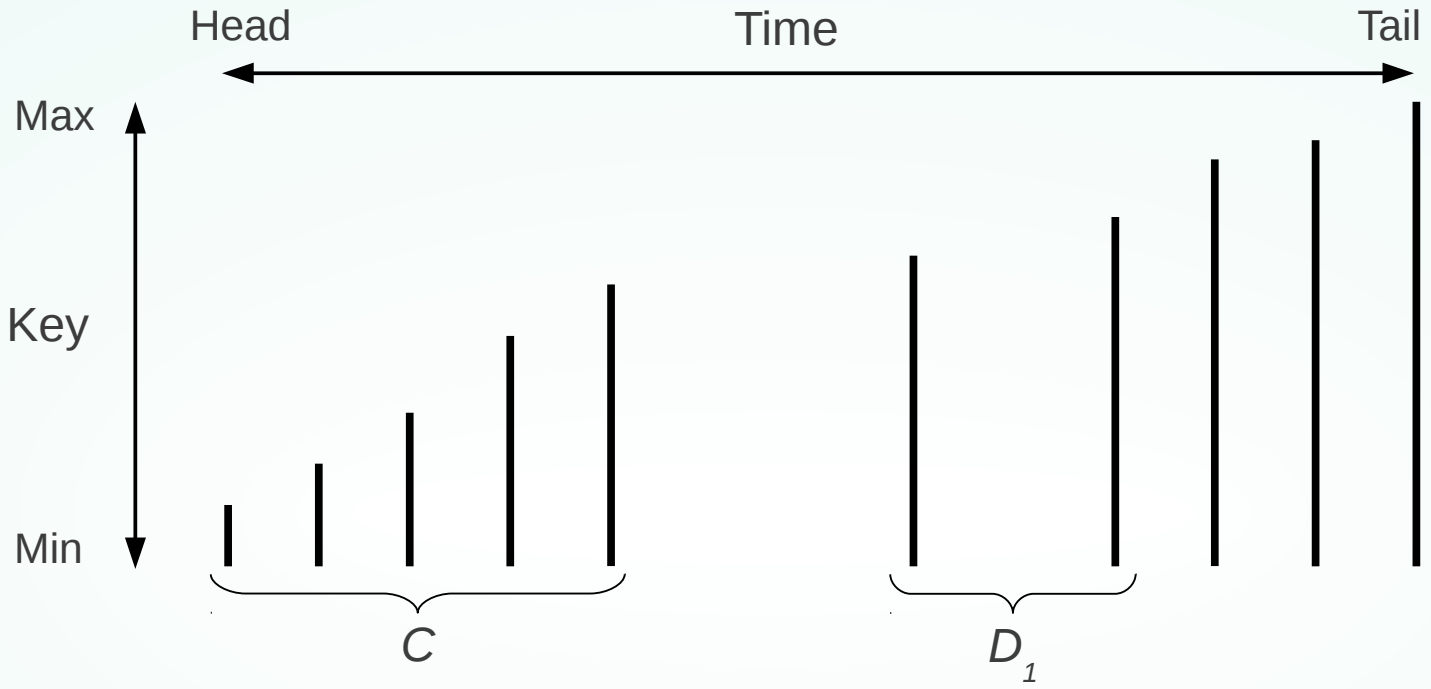
- InsertAndAttrite(*e*): Inserts *e* and deletes/attrites all elements before *e* with a key larger or equal to *e*
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element *e* of the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

Observations and Invariants



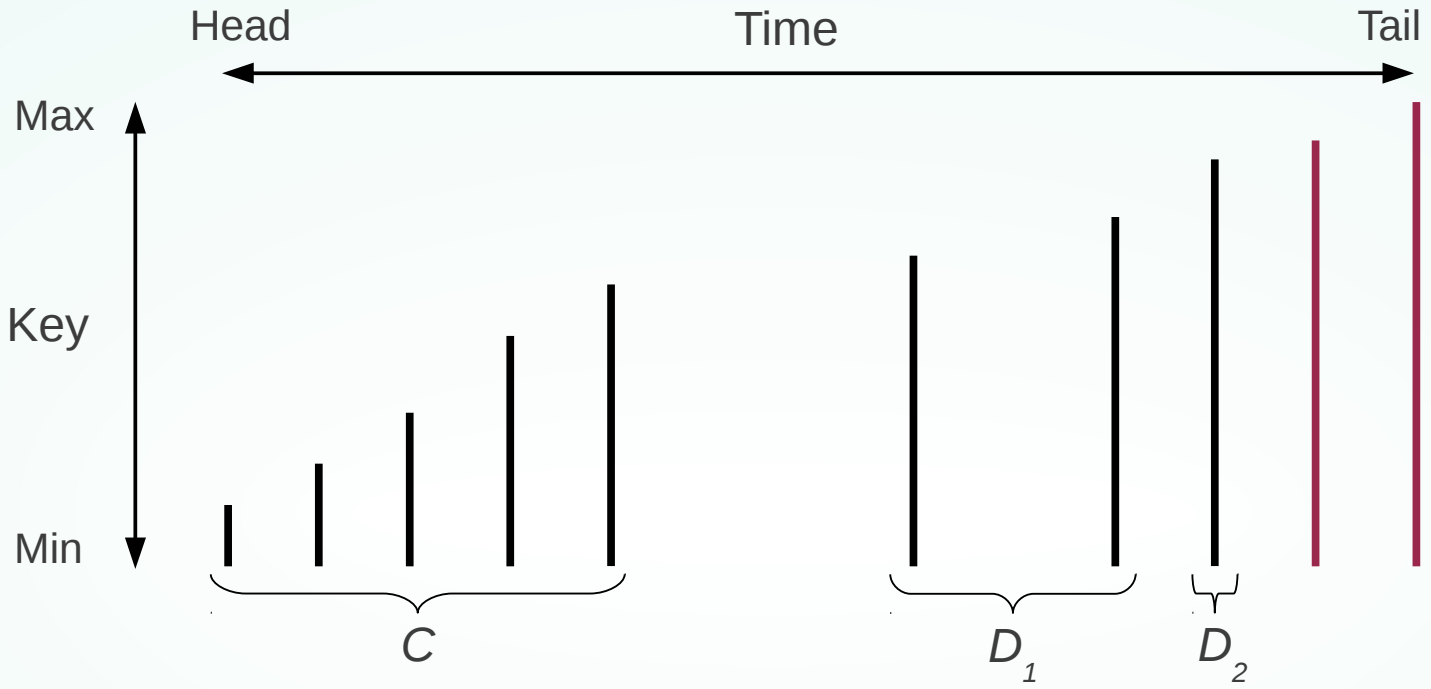
- InsertAndAttrite(*e*): Inserts *e* and deletes/attrites all elements before *e* with a key larger or equal to *e*
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element *e* of the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

Observations and Invariants



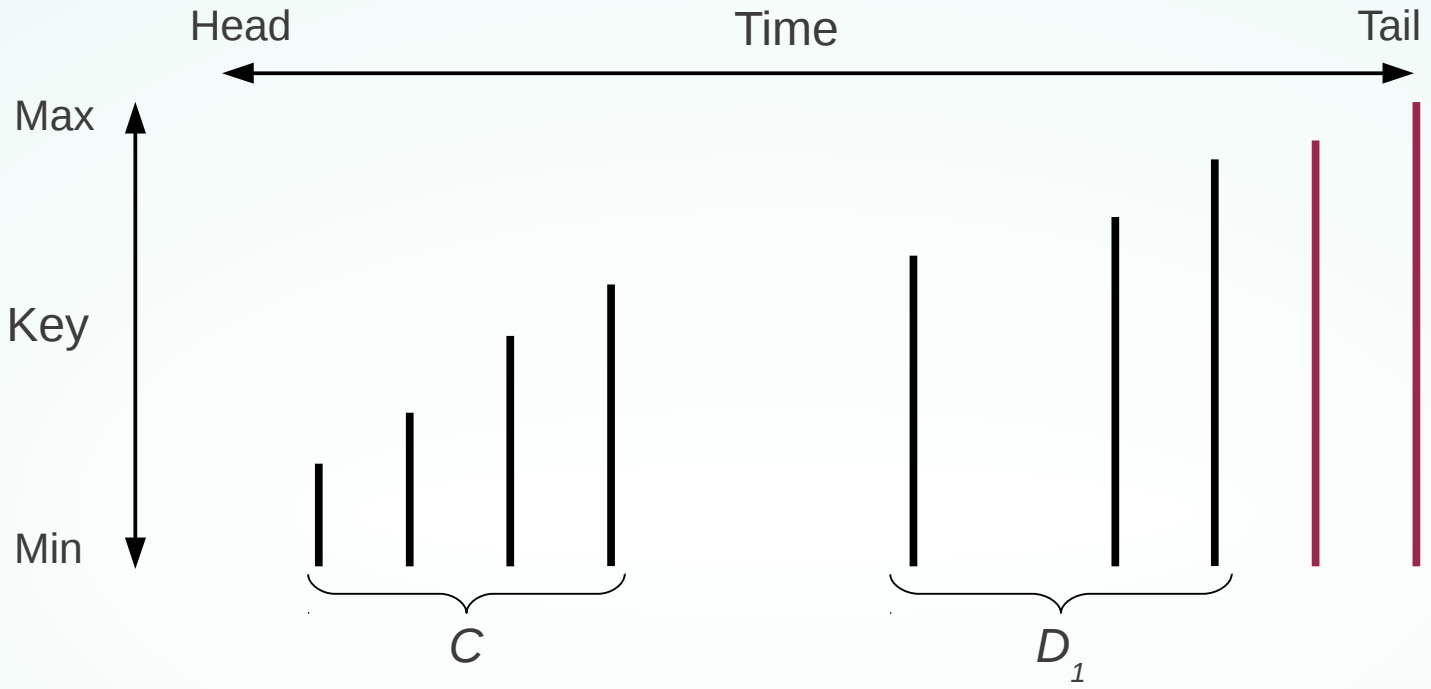
- InsertAndAttrite(*e*): Inserts *e* and deletes/attrites all elements before *e* with a key larger or equal to *e*
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element *e* of the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

Observations and Invariants



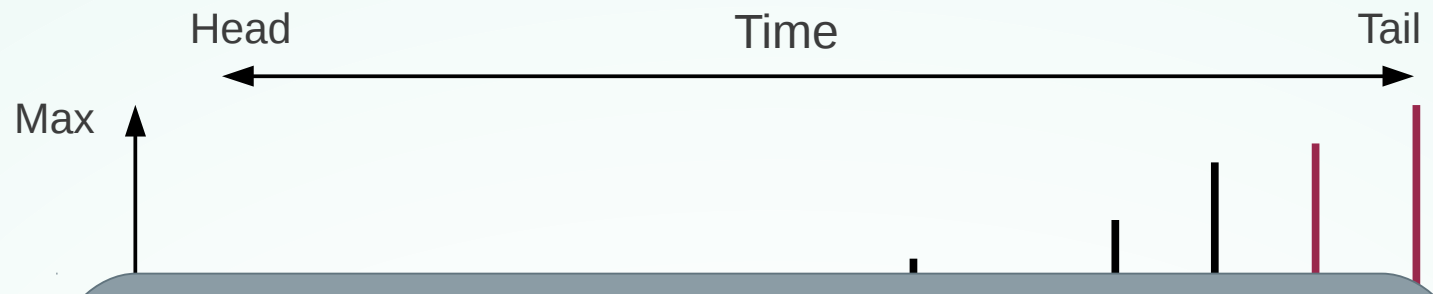
- InsertAndAttrite(*e*): Inserts *e* and deletes/attrites all elements before *e* with a key larger or equal to *e*
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element *e* of the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

Observations and Invariants

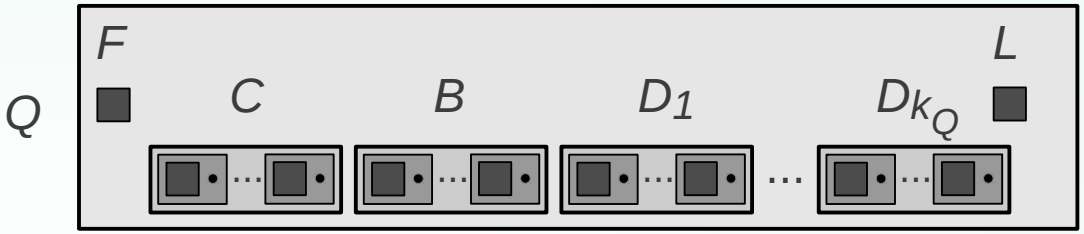


Summary of Invariants

- 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

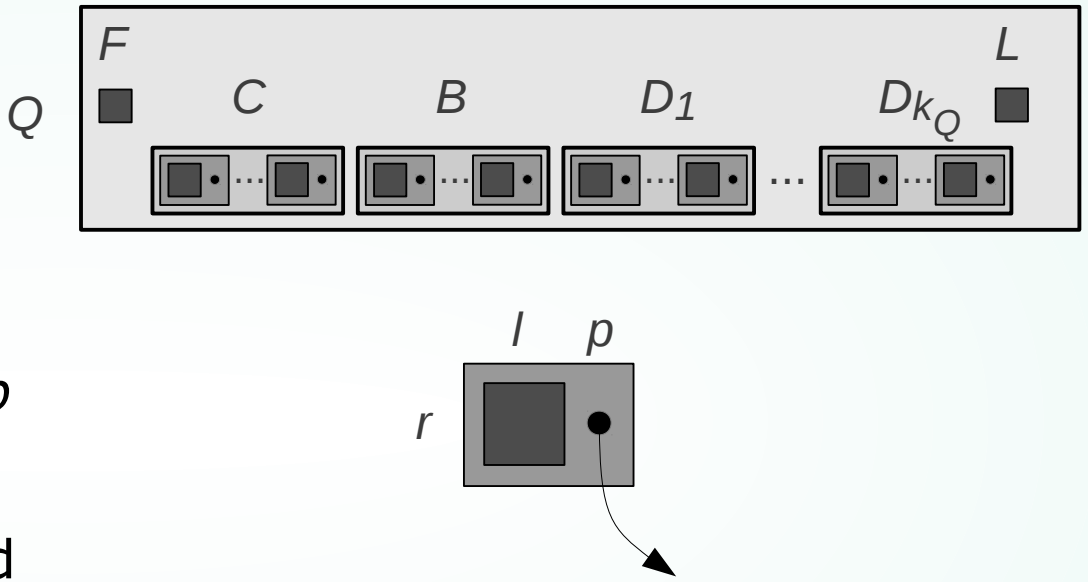
- Insert… before *e* with a key larger or equal to *e*
- ConcatenateAndAttrite(Q_1,Q_2): Deletes/attrites all elements in Q_1 with a key larger or equal to min(Q_2) and appends Q_2
- DeleteMin*(Q):* Deletes the head/minimum element *e* of the queue *Q*

Concatenable PQA: Data Structure



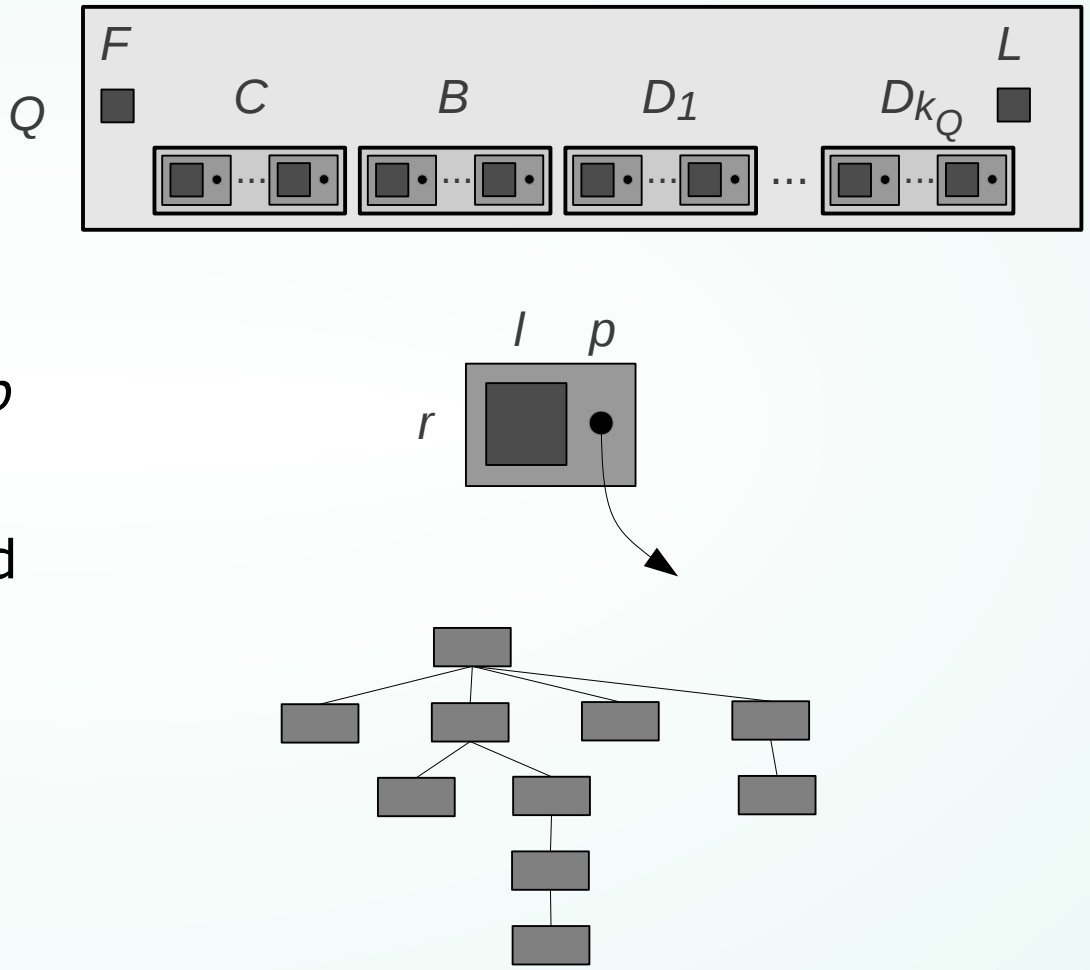
Concatenable PQA: Data Structure

- 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 *PQA*s as internal and leaf nodes



Concatenable PQA: Data Structure

- A *PQA* consists of $2+k_Q$ deques $C, B, D_1, \ldots, D_{k_Q}$ of records and buffers F and L
- A record $r=(I,p)$ contains a buffer I 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 *PQA*s as internal and leaf nodes



Invariants and Operations

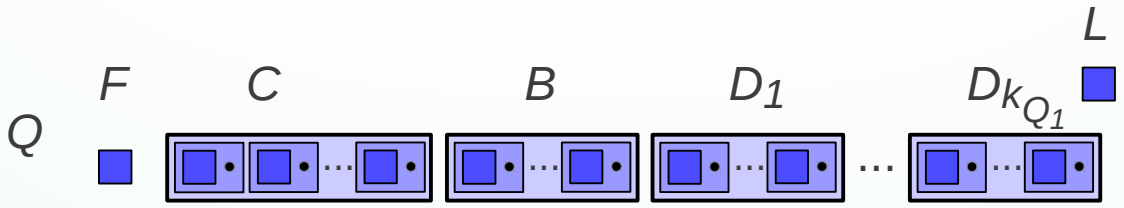
$$max(C(Q)) < min(B(Q)) < min(D_1(Q)) < min(D_i(Q)), \ for\ i > 1$$

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

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

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

DeleteMin



Invariants and Operations

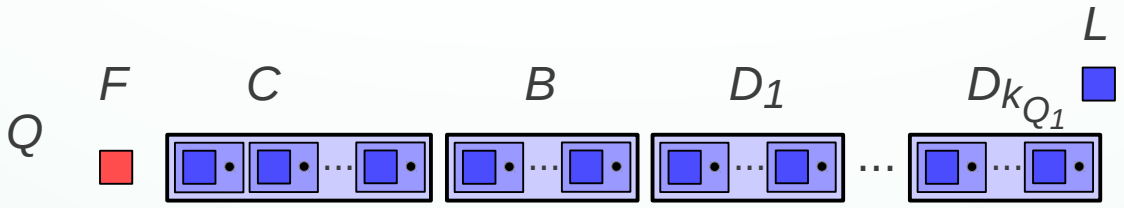
$$max(C(Q)) < min(B(Q)) < min(D_1(Q)) < min(D_i(Q)), \quad for\ i > 1$$

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

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

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

DeleteMin



Invariants and Operations

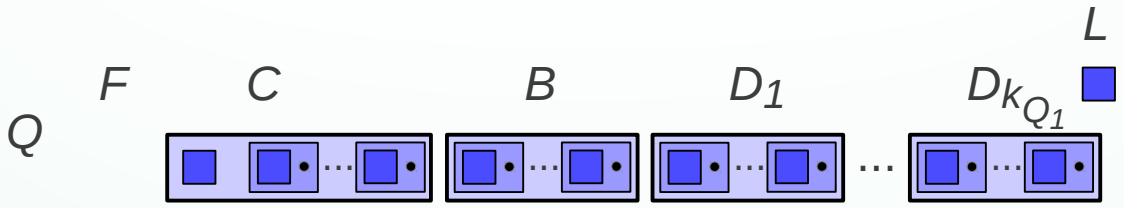
$$max(C(Q)) < min(B(Q)) < min(D_1(Q)) < min(D_i(Q)), \ for\ i > 1$$

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

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

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

DeleteMin



Invariants and Operations

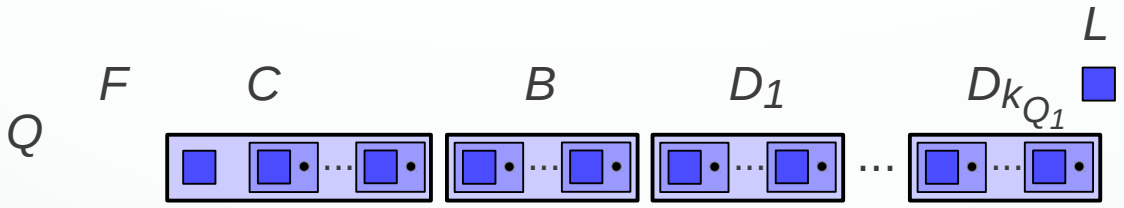
$$max(C(Q)) < min(B(Q)) < min(D_1(Q)) < min(D_i(Q)), \ for\ i > 1$$

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

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

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

DeleteMin



Invariants and Operations

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

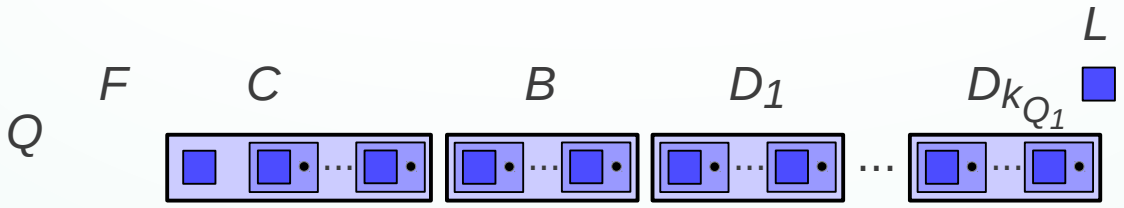
$$C(Q) \text{ and } B(Q) \text{ are simple}$$

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

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

DeleteMin

Bias to the rescue!



Invariants and Operations

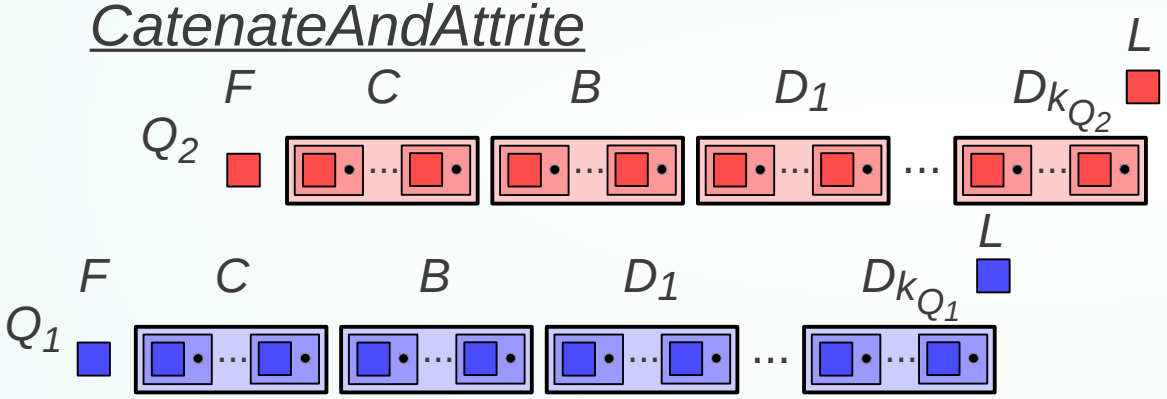
$$max(C(Q)) < min(B(Q)) < min(D_1(Q)) < min(D_i(Q)), \ for\ i > 1$$

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

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

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

<u>CatenateAndAttrite</u>



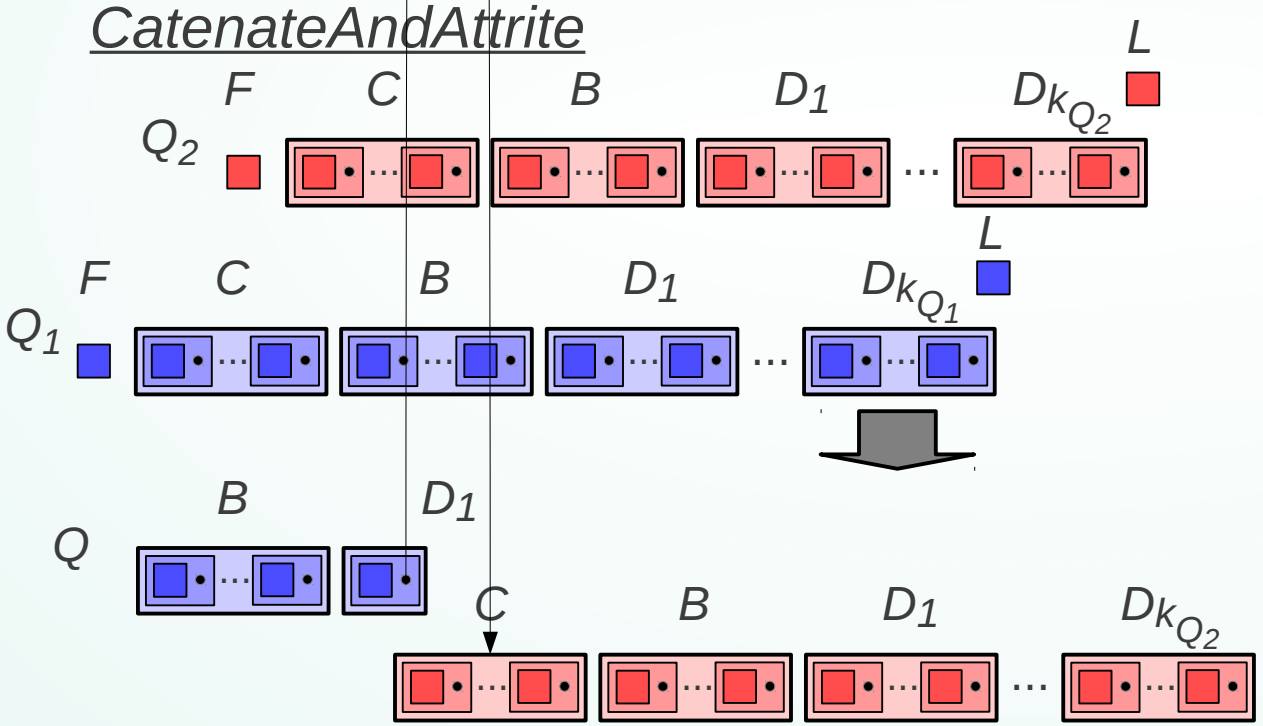
Invariants and Operations

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

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

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

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



Invariants and Operations

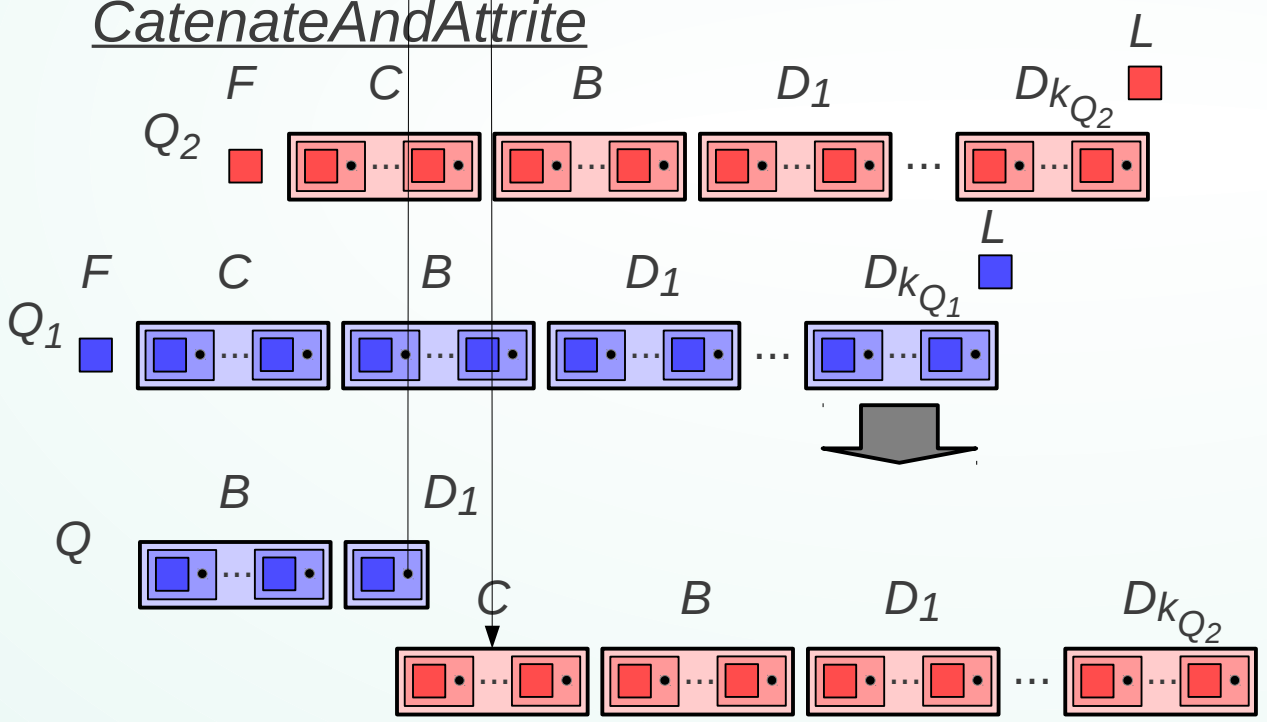
$$max(C(Q)) < min(B(Q)) < min(D_1(Q)) < min(D_i(Q)), \ for\ i > 1$$

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

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

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

CatenateAndAttrite



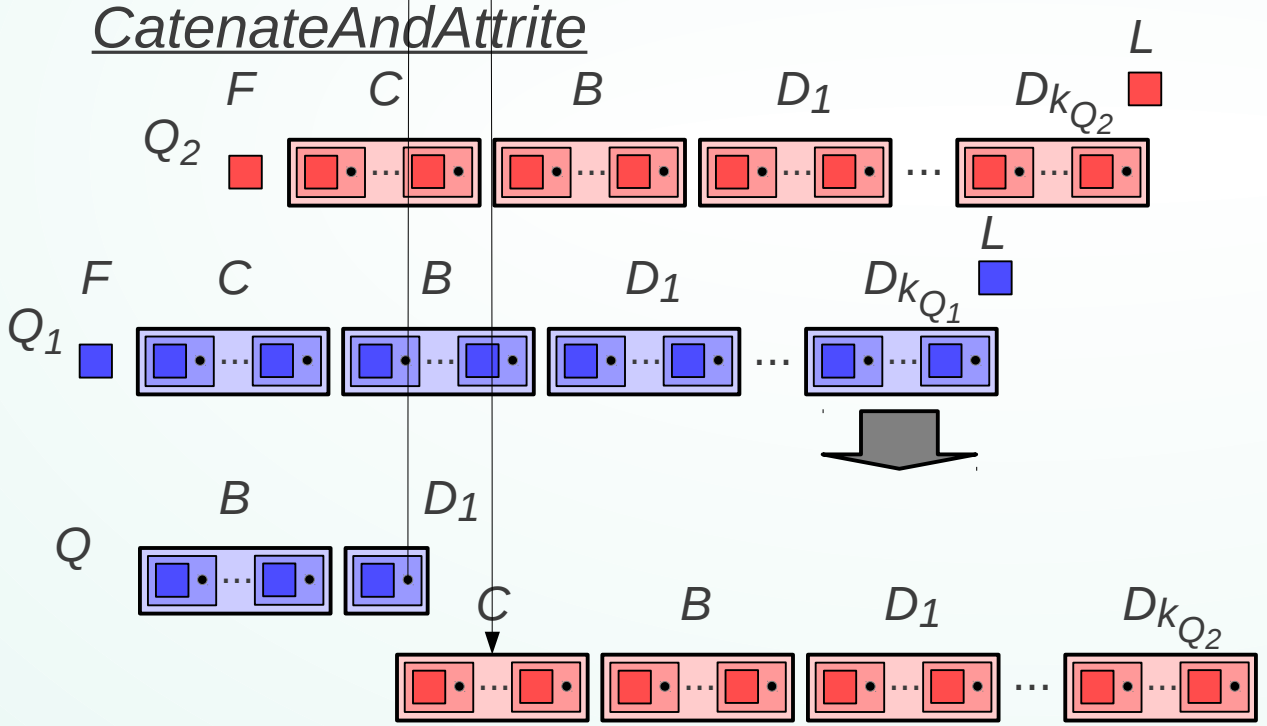
Invariants and Operations

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

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

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

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



Bias to the rescue!

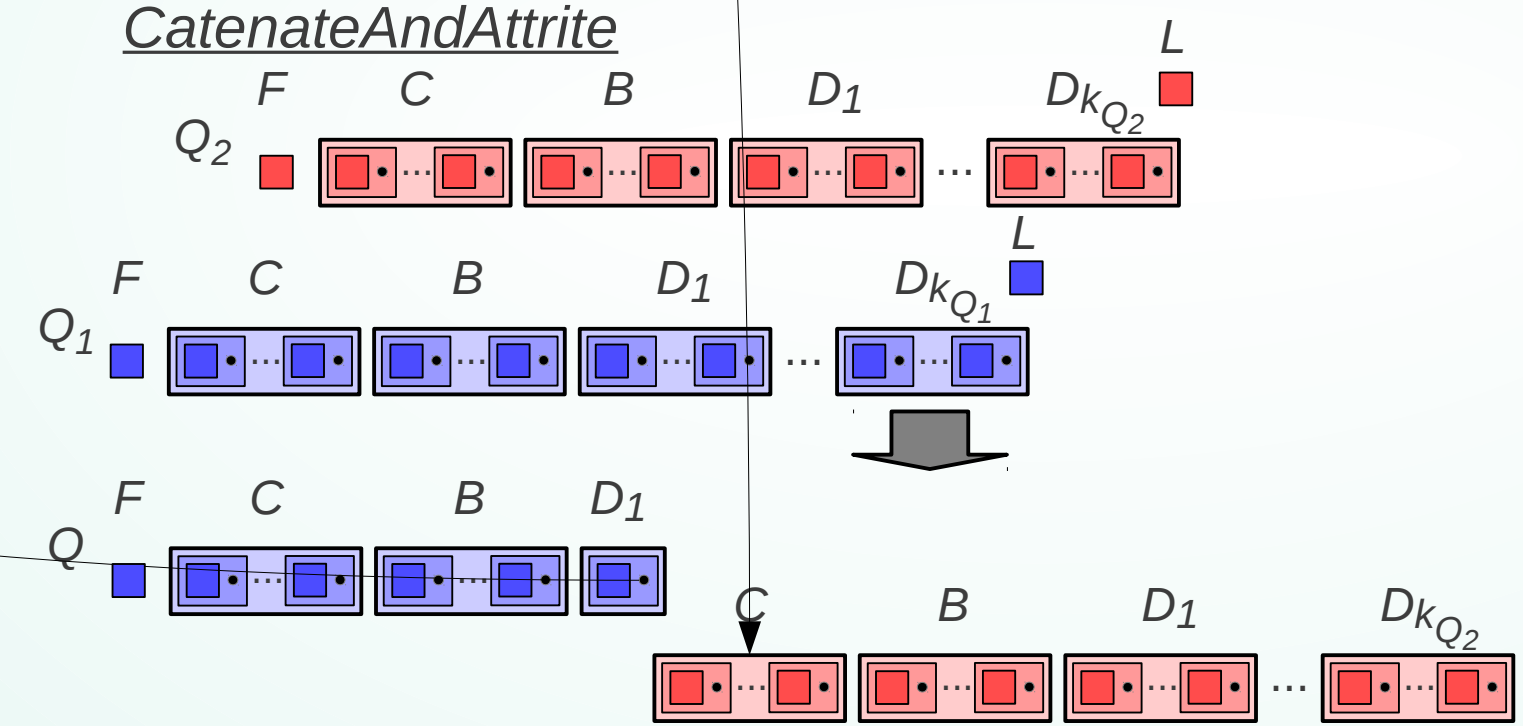
Invariants and Operations

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

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

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

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



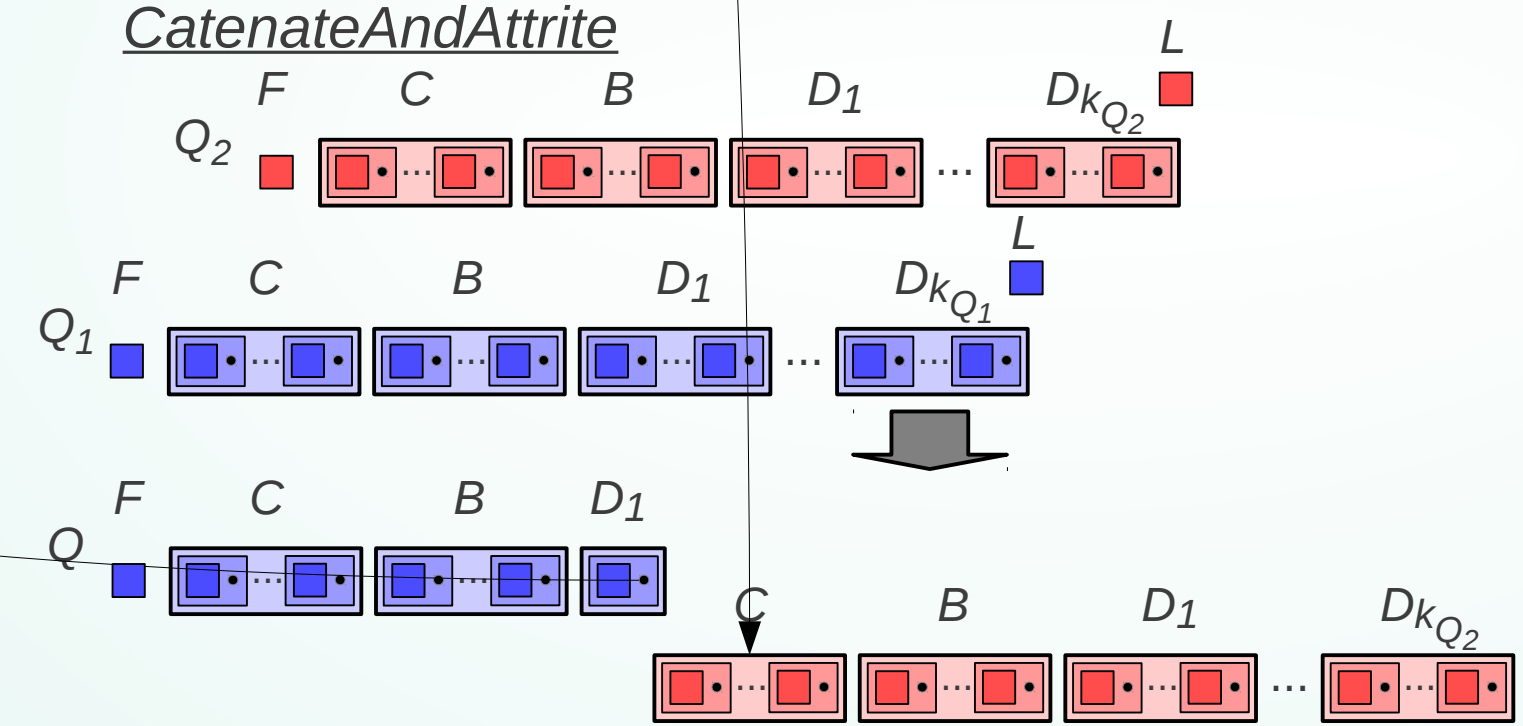
Invariants and Operations

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

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

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

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



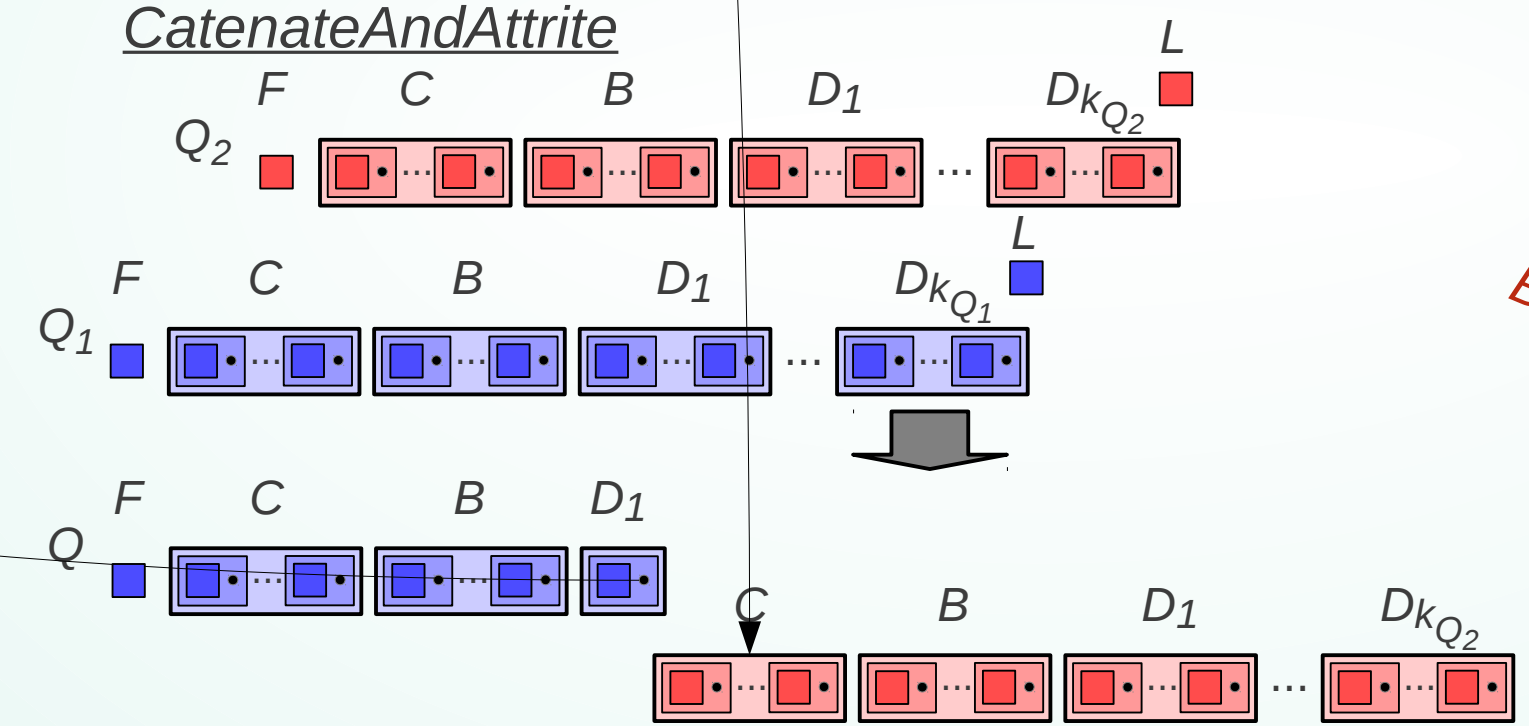
Invariants and Operations

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

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

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

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



Bias to the rescue!

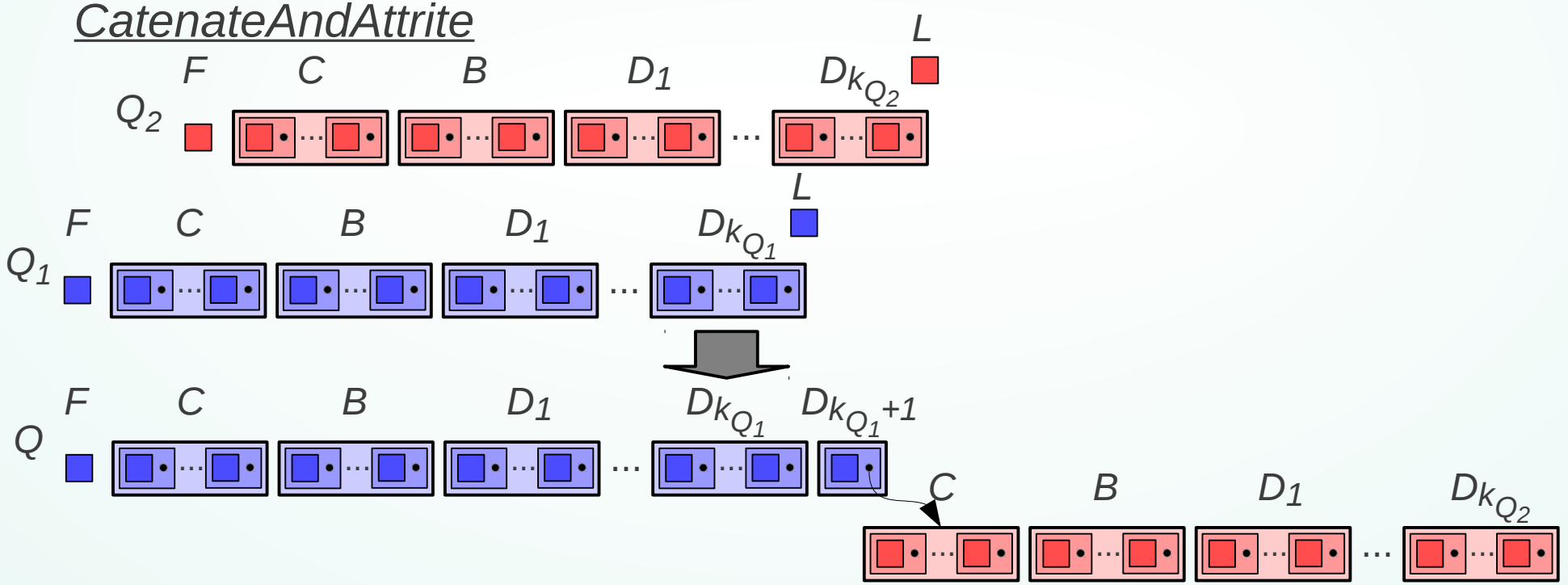
Invariants and Operations

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

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

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

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



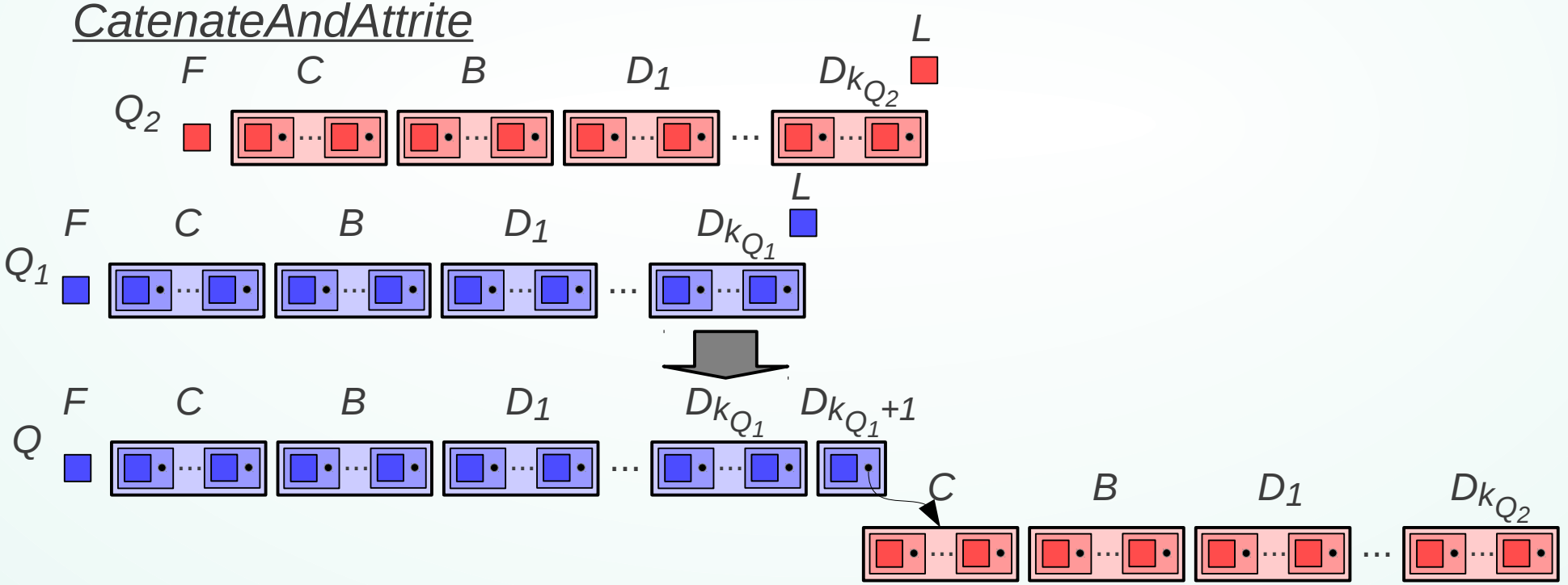
Invariants and Operations

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

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

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

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



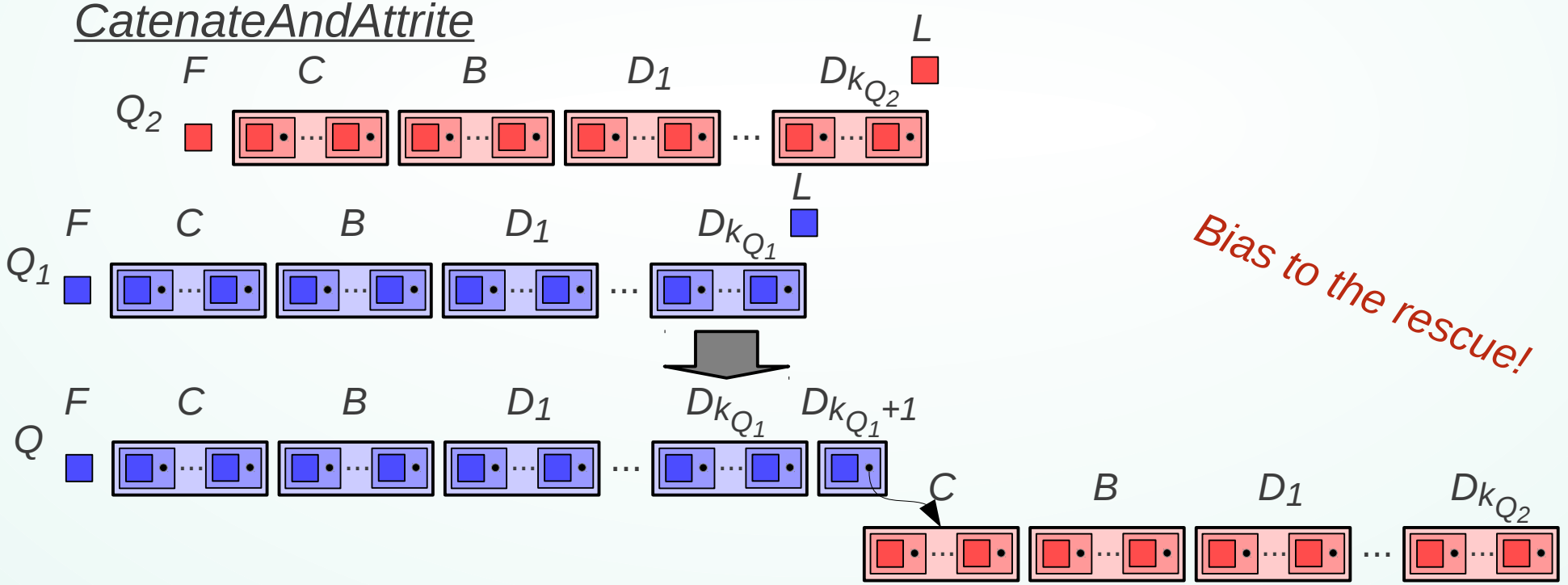
Invariants and Operations

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

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

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

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



Invariants and Operations

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

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

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

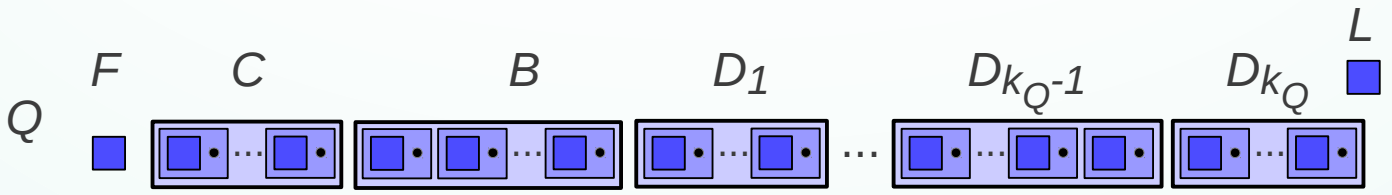
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1



Invariants and Operations

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

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

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

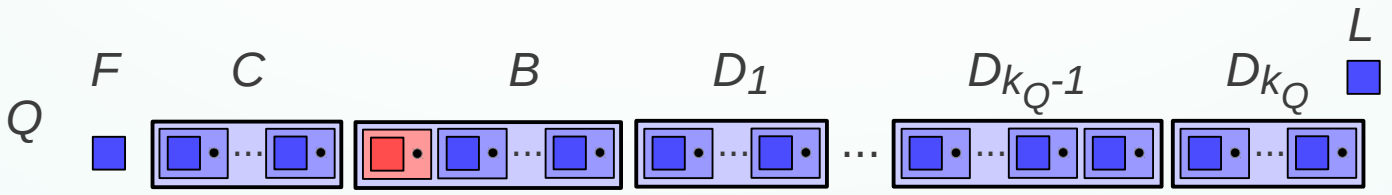
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1



Invariants and Operations

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

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

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

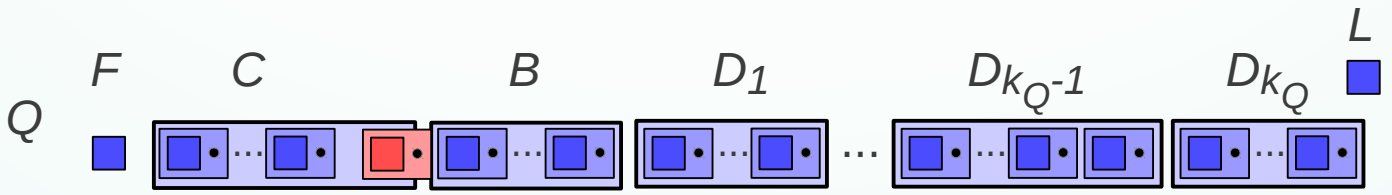
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1



Invariants and Operations

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

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

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

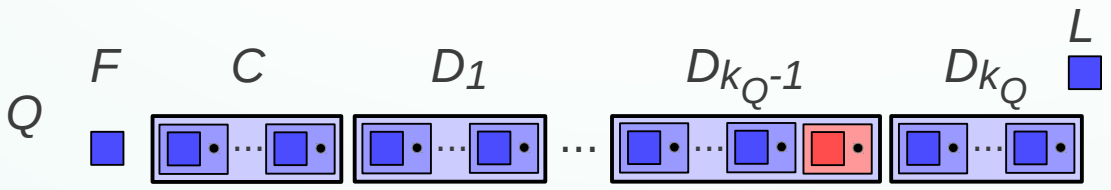
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1



Invariants and Operations

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

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

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

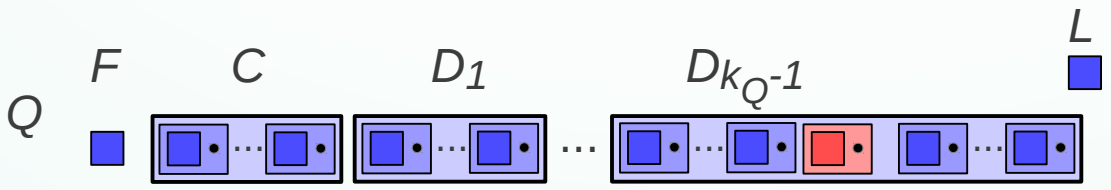
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1



Invariants and Operations

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

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

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

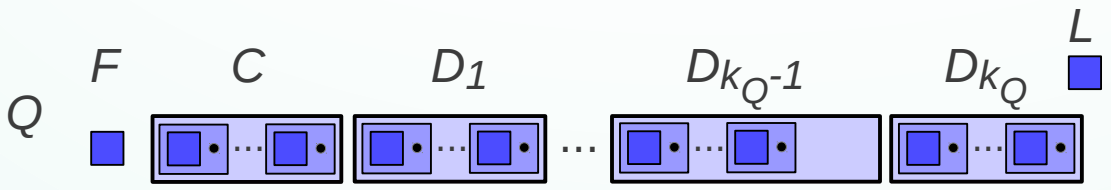
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1



Invariants and Operations

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

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

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

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

Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1

Invariants and Operations

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

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

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

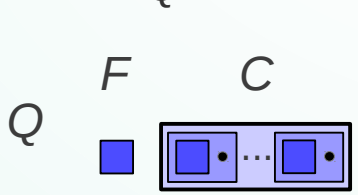
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

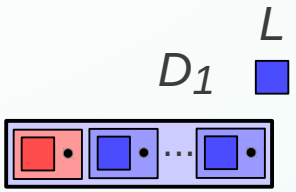
Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1





Invariants and Operations

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

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

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

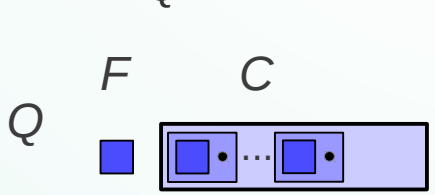
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

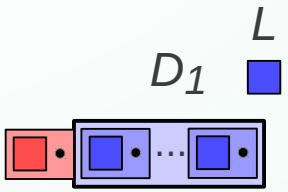
Bias

B>0

B=0 and k_Q>1

B=0 and k_Q=1





Invariants and Operations

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

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

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

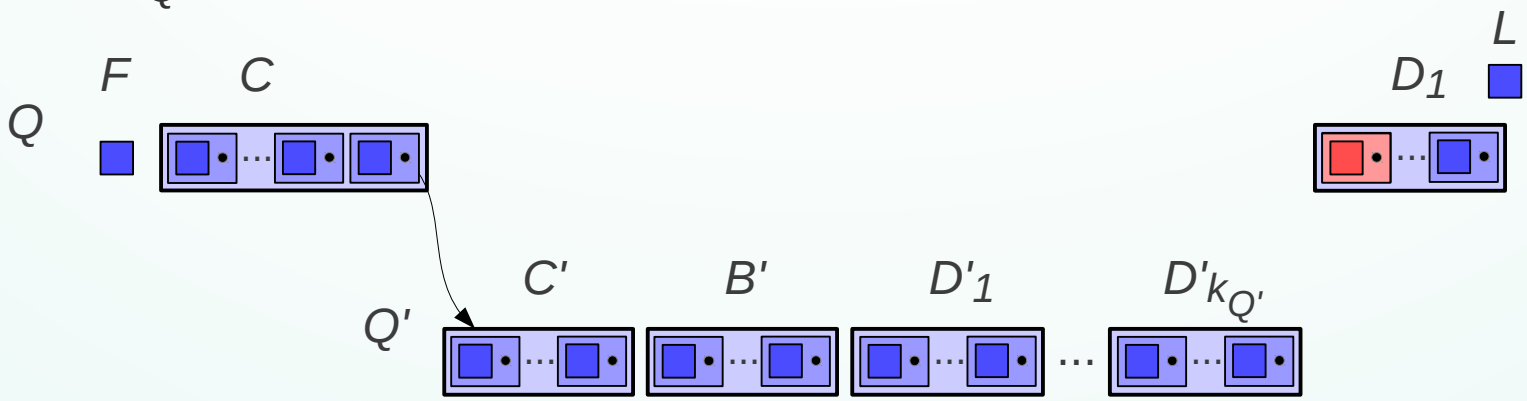
$$|C(Q)| \geqslant \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q$$

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?

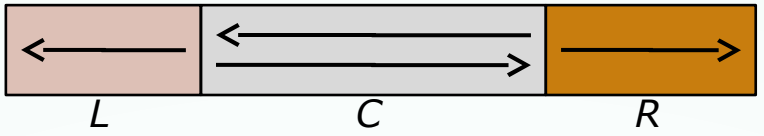
Thank you :)

References

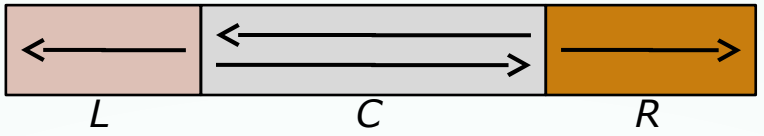
1) **A Cache-Oblivious Implicit Dictionary with the Working Set Property**
   - Gerth Stlting Brodal, Casper Kejlberg-Rasmussen, Jakob Truelsen
   - ISAAC 2010
   - *Available at http://dx.doi.org/10.1007/978-3-642-17514-5_4*
2) **Cache-Oblivious Implicit Predecessor Dictionaries with the Working-Set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

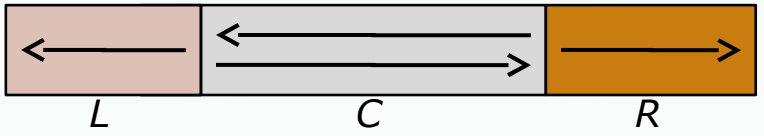


Implicit Moveable Dictionaries

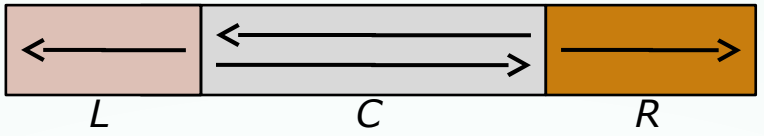


- Uses O(*1*) FG dictionaries as black boxes
- *Recall the FG interface:*
  - Insert-right(*e*): insert element *e* into the dictionary which grows to the right
  - Delete-right(*e*): delete element *e* from the dictionary which shrinks from the right
  - Search(*e*): finds the address of *e* if *e* 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

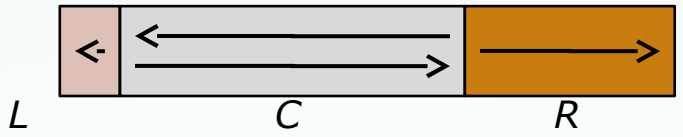


Implicit Moveable Dictionaries



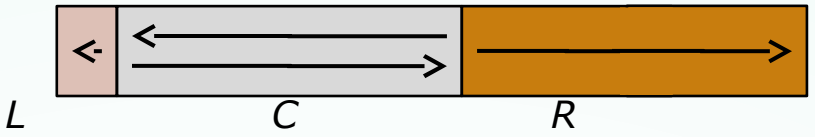
- *L* and *R* will shrink and grow over time
  - *L*/*R* might get too small or
  - *L*/*R* might get too large compared to *C*
- We introduce the notion of *jobs*
  - Grow-left/right – Counters when *L*/*R* gets too small
  - Shrink-left/right – Counters when *L*/*R* gets too large
  - Jobs run O(*1*)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

Implicit Moveable Dictionaries



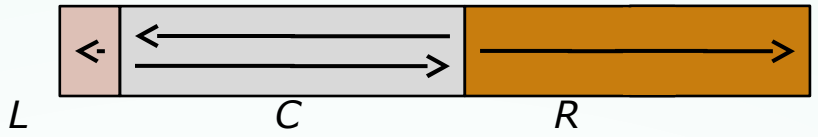
- *L* and *R* will shrink and grow over time
  - *L*/*R* might get too small or
  - *L*/*R* might get too large compared to *C*
- We introduce the notion of *jobs*
  - Grow-left/right – Counters when *L*/*R* gets too small
  - Shrink-left/right – Counters when *L*/*R* gets too large
  - Jobs run O(*1*) 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

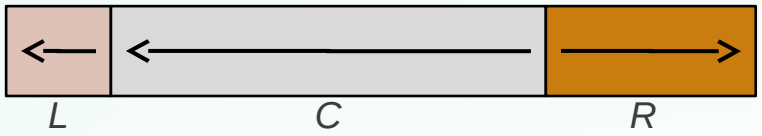


- *L* and *R*
  - *L*/*R*
  - *L*/*R*
- We intro
  - Grow-left/right – Counters when *L*/*R* gets too small
  - Shrink-left/right – Counters when *L*/*R* gets too large
  - Jobs run O(*1*) steps every operation: searches, updates

Summary of Invariants

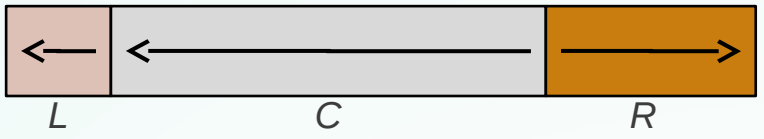
- We ensure that both L and R are not too small/large
- We have queued at most 2 jobs
- All jobs finish before their deadline

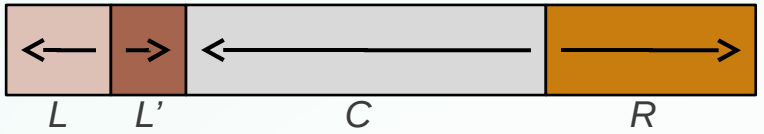
Implicit Moveable Dictionaries



Grow-left

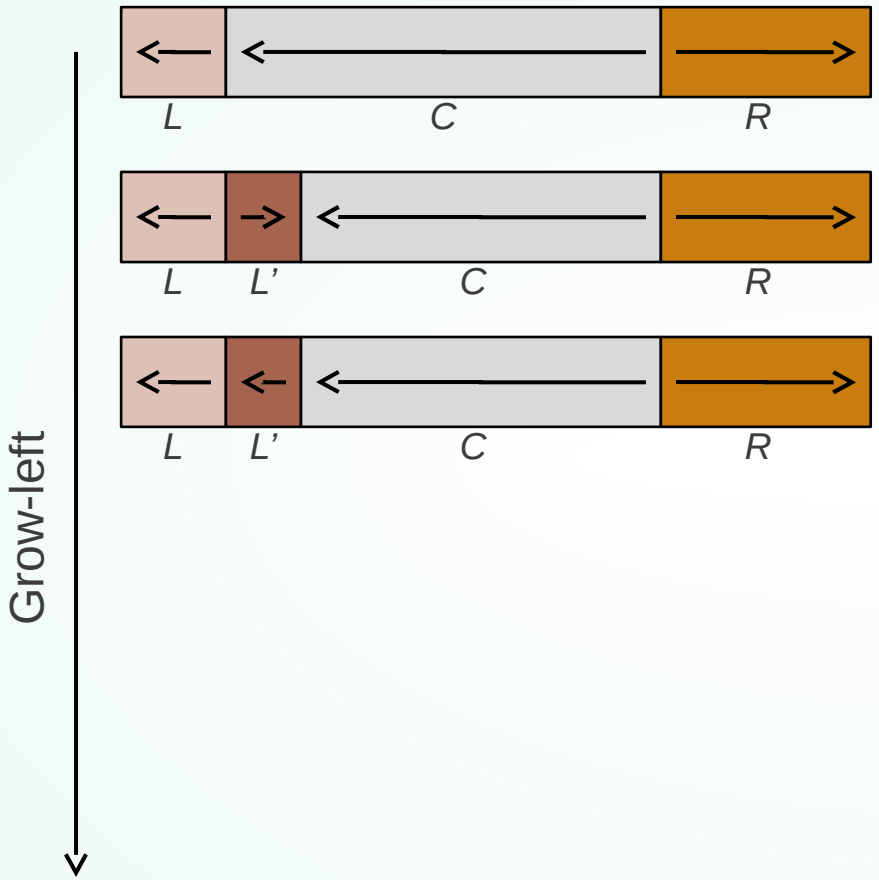
Implicit Moveable Dictionaries



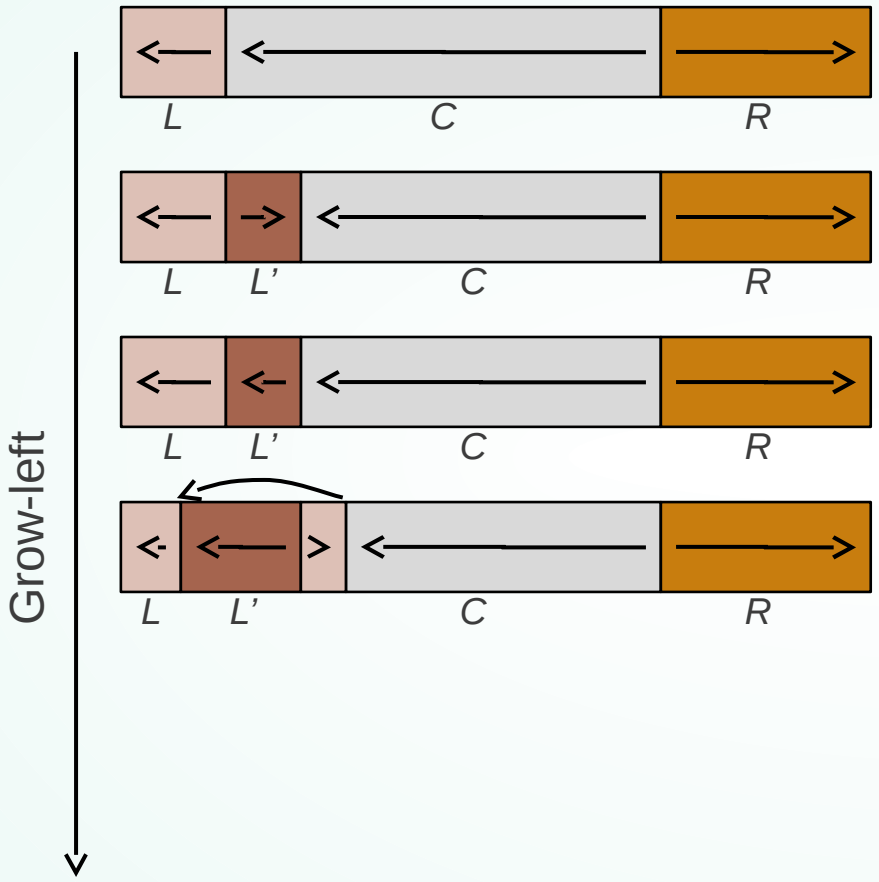


Grow-left

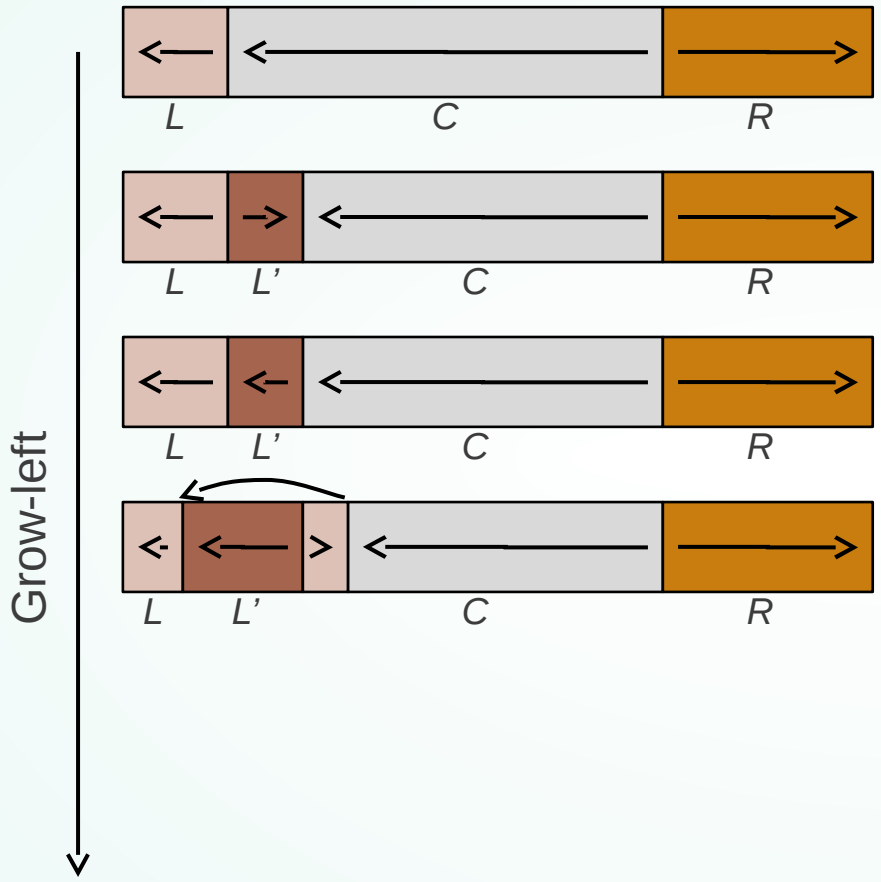
Implicit Moveable Dictionaries

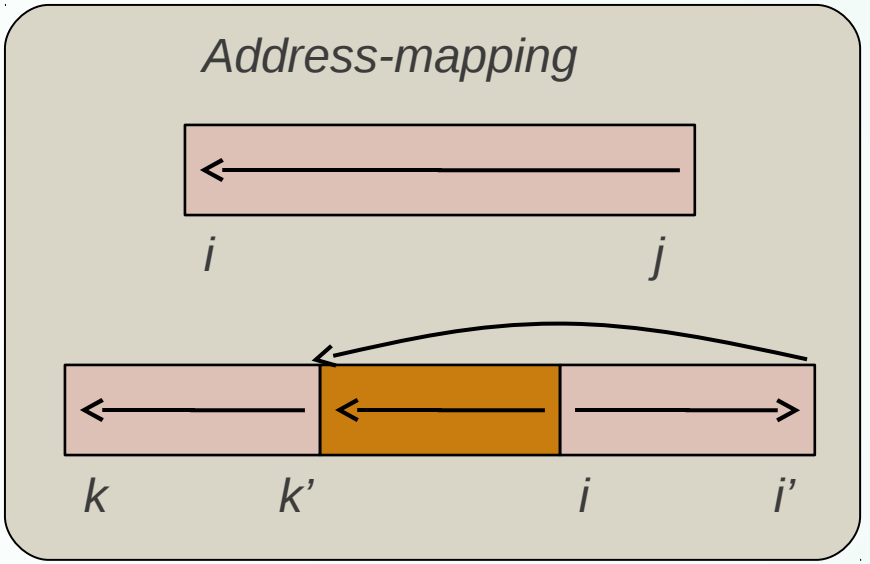


Implicit Moveable Dictionaries

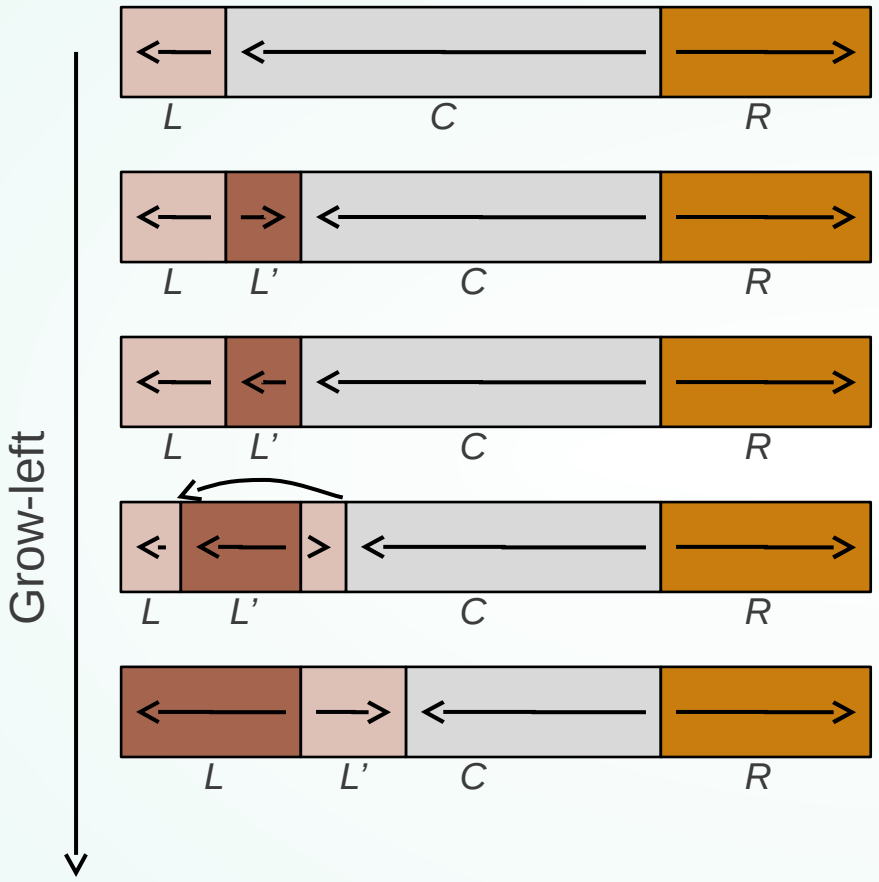


Implicit Moveable Dictionaries

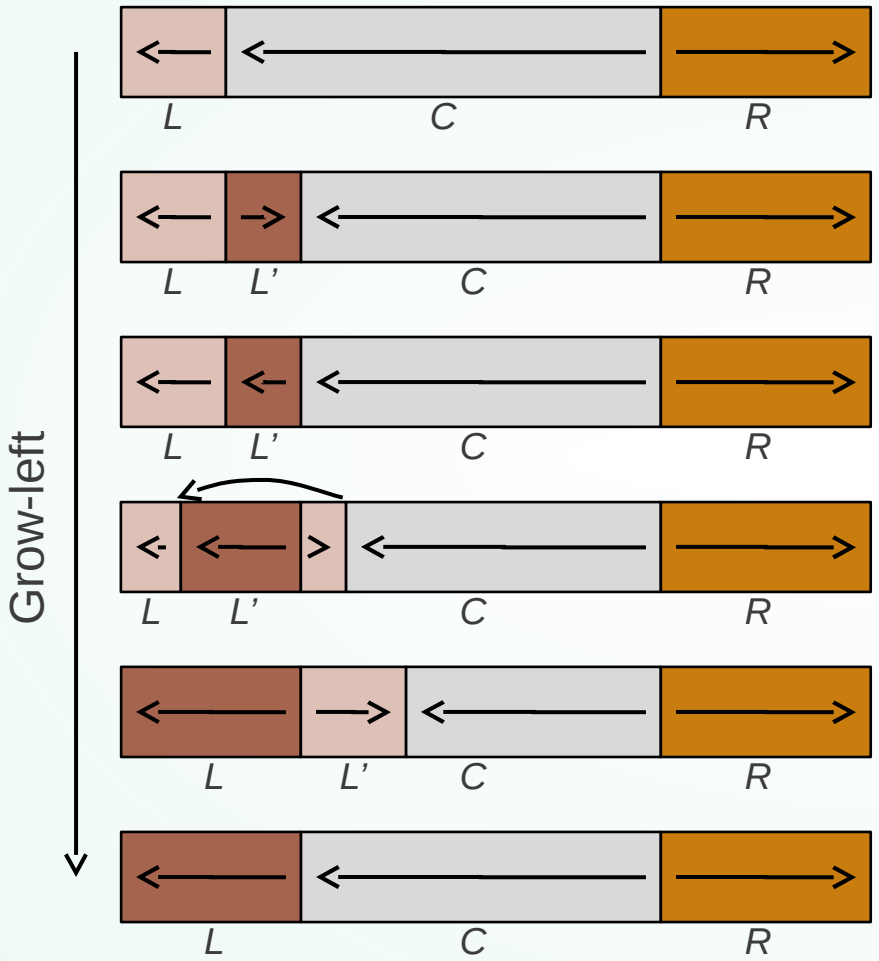




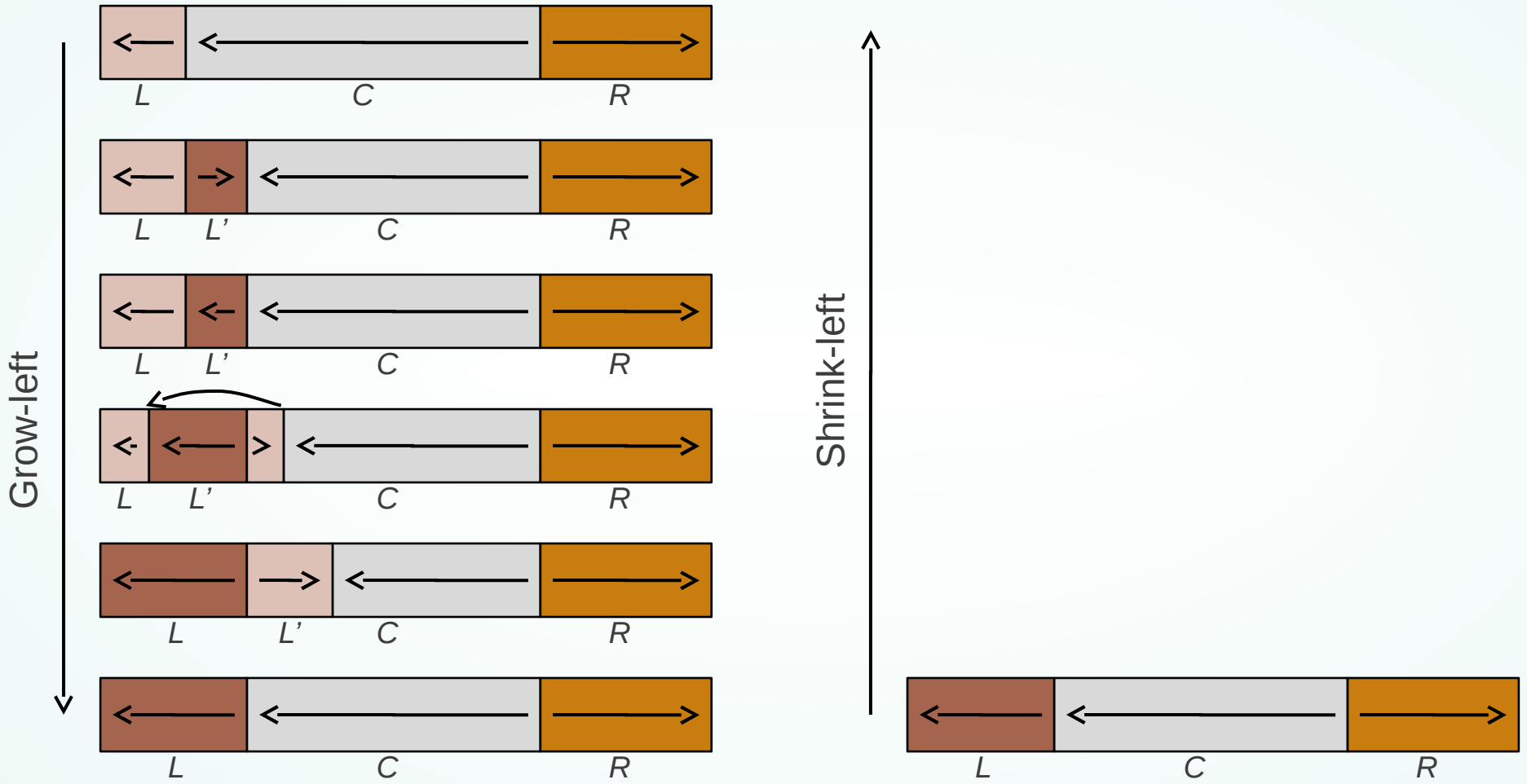
Implicit Moveable Dictionaries



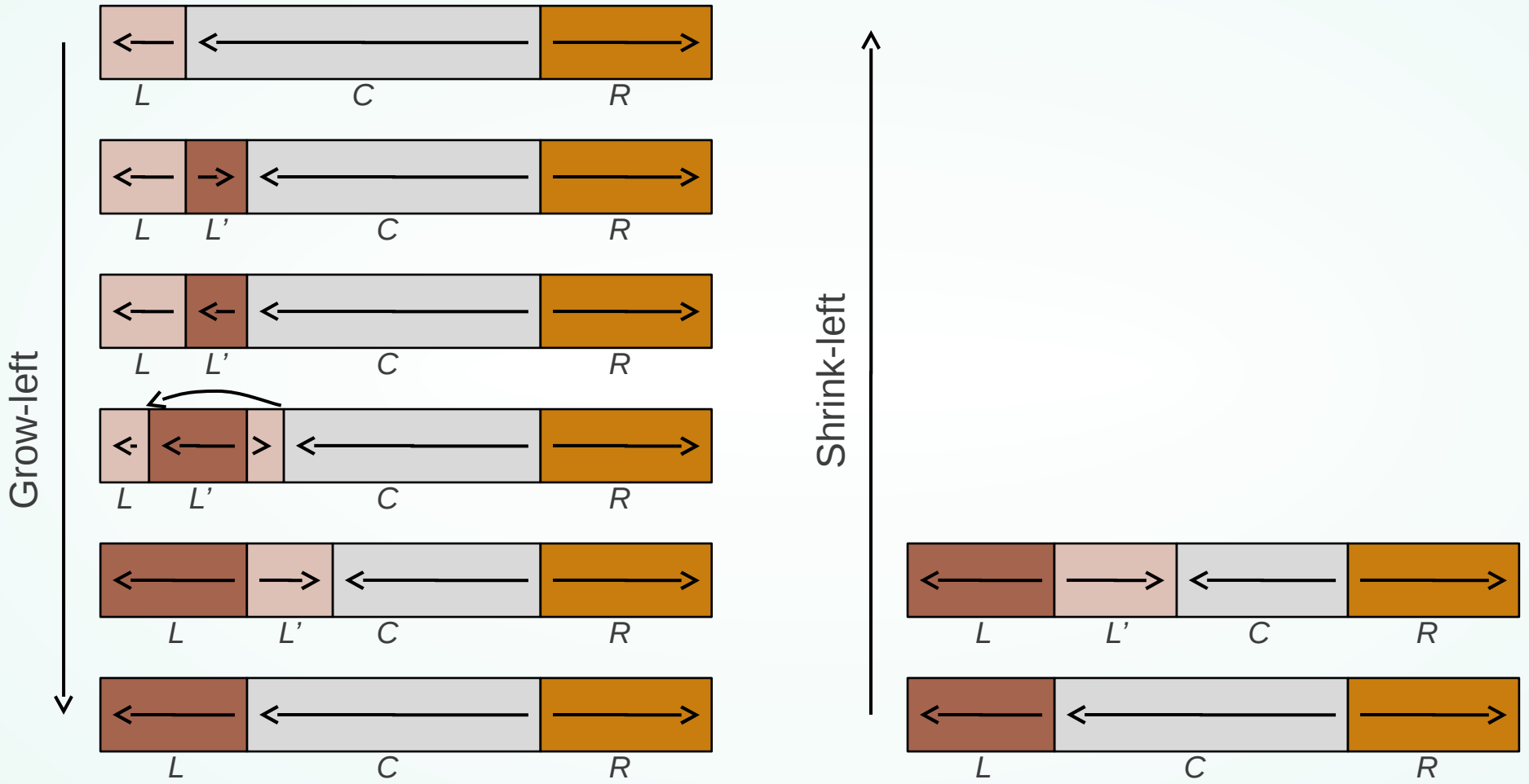
Implicit Moveable Dictionaries



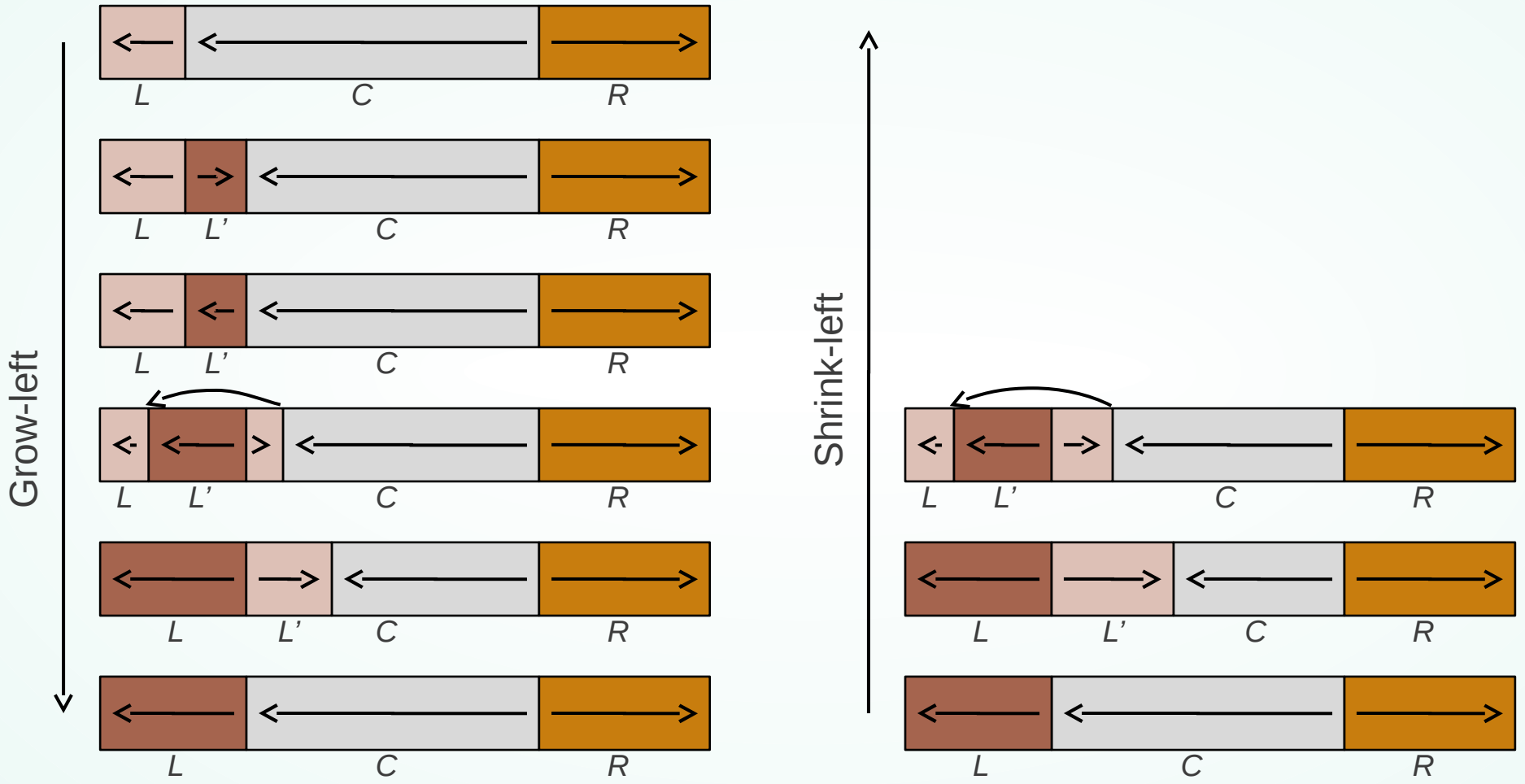
Implicit Moveable Dictionaries



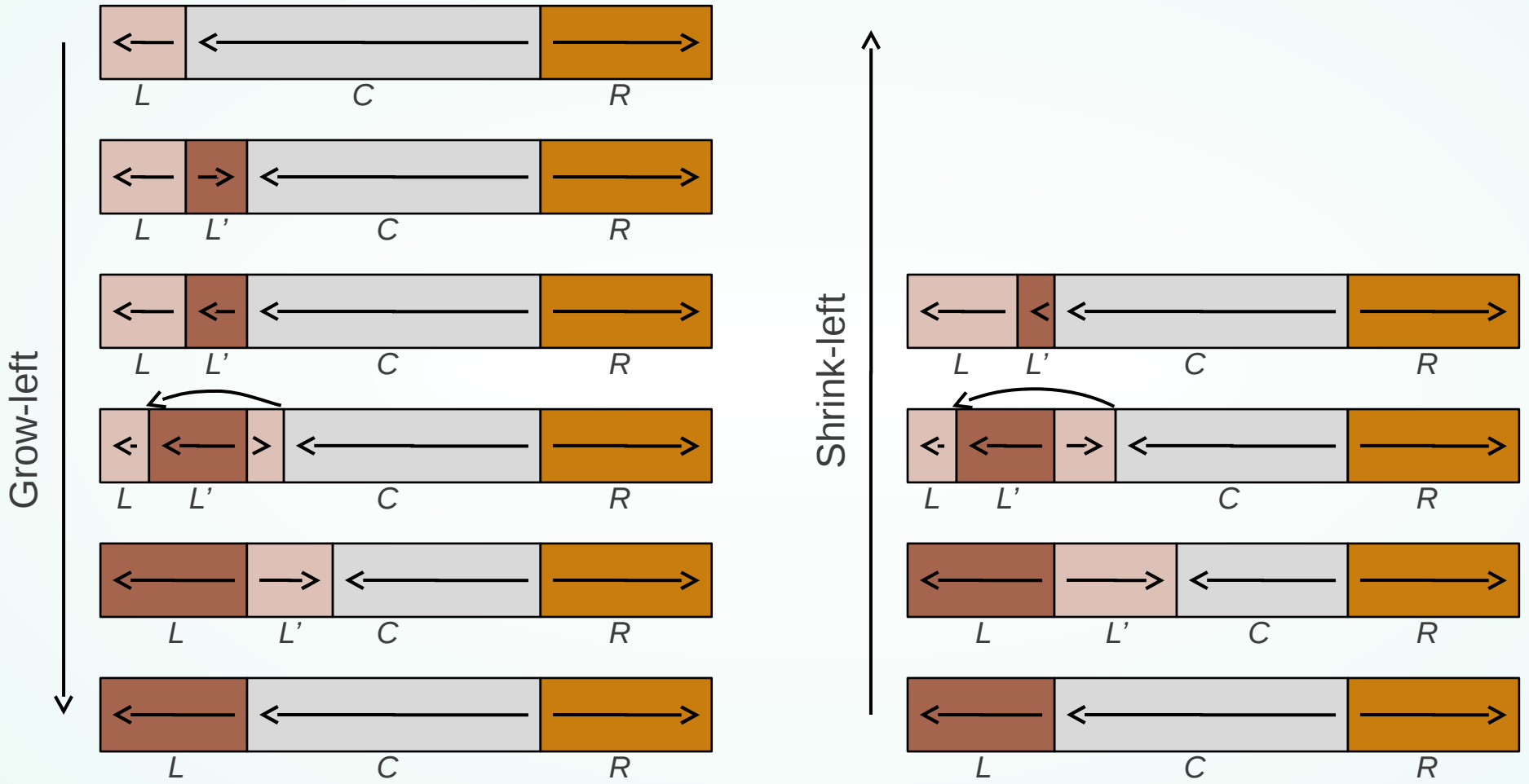
Implicit Moveable Dictionaries



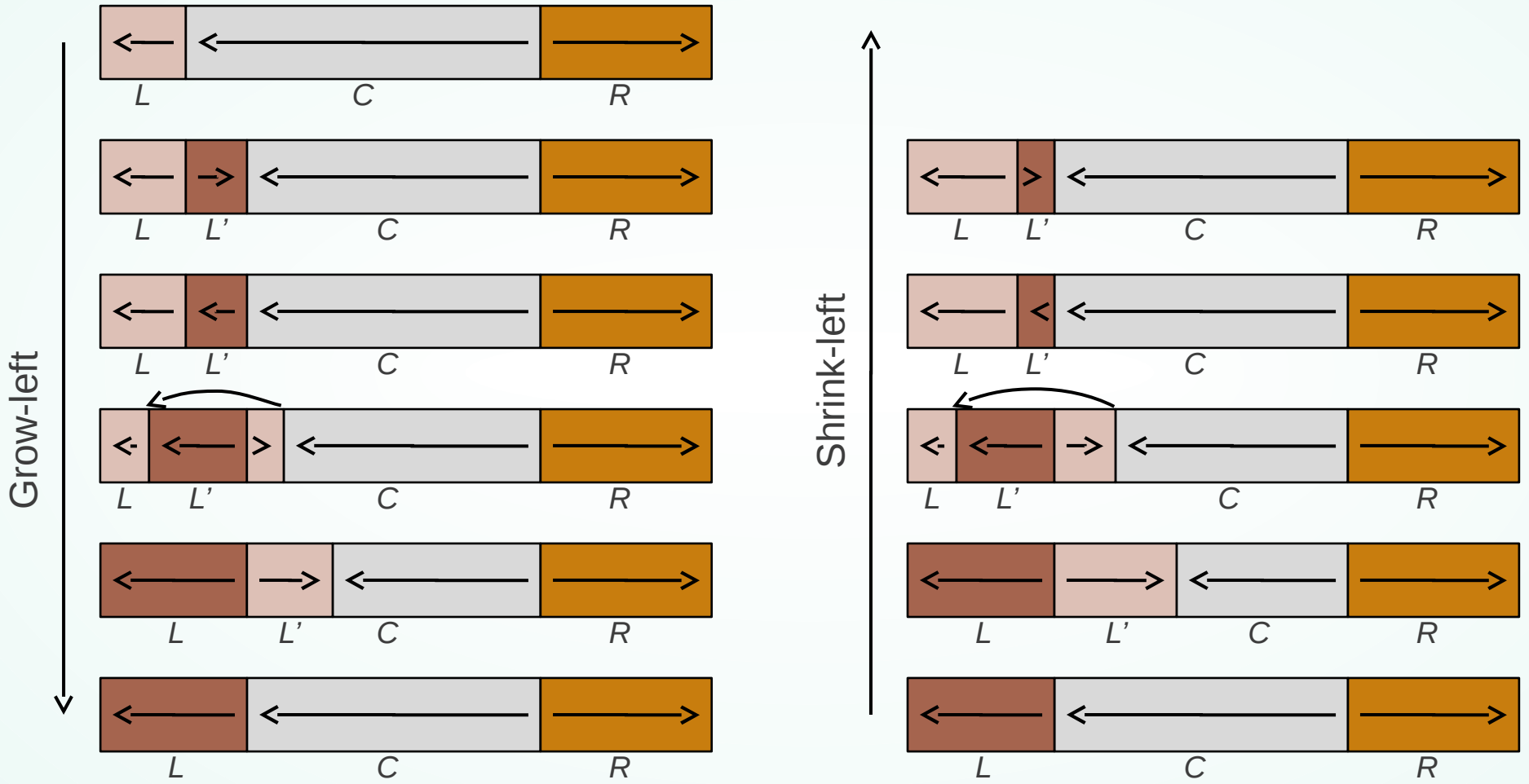
Implicit Moveable Dictionaries



Implicit Moveable Dictionaries



Implicit Moveable Dictionaries



Implicit Moveable Dictionaries