Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps

William F. Jones and **Gregory M. Kapfhammer**

Allegheny College
http://www.cs.allegheny.edu/~gkapfham/

20th International Conference on Software Engineering and Data Engineering, June 20 - 22, 2011

Overview of the Presentation

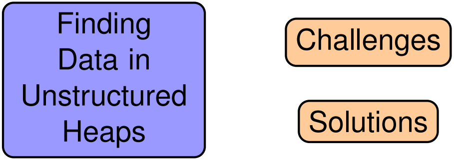
Finding
Data in
Unstructured
Heaps

Overview of the Presentation

Finding Data in Unstructured Heaps

Challenges

Overview of the Presentation



Overview of the Presentation

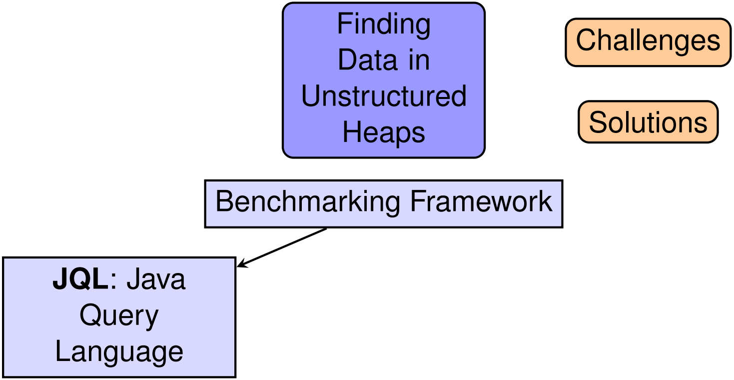
Finding Data in Unstructured Heaps

Challenges

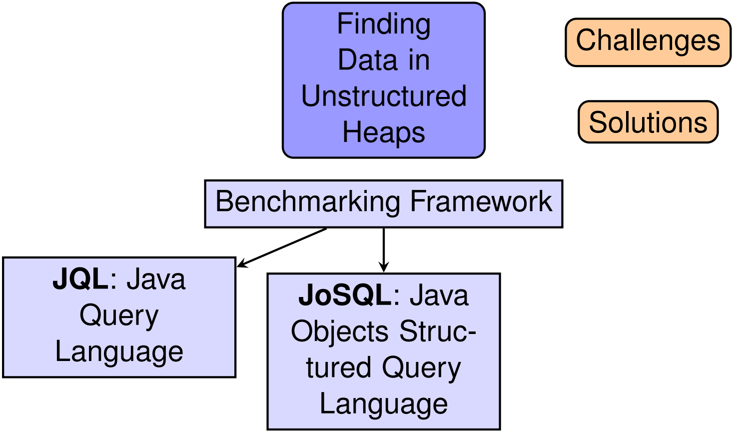
Solutions

Benchmarking Framework

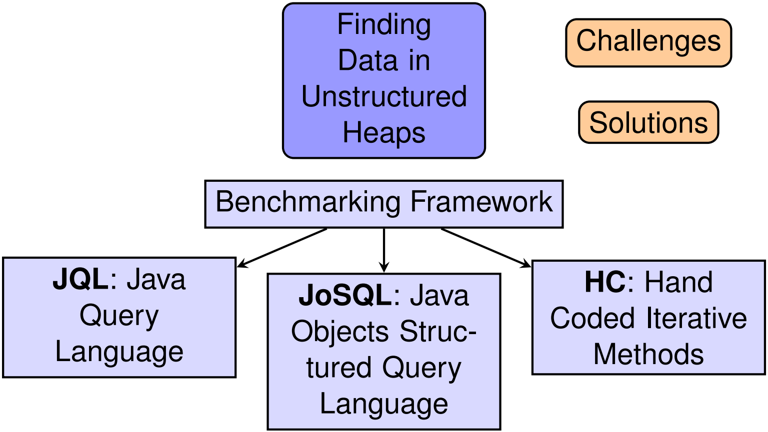
Overview of the Presentation



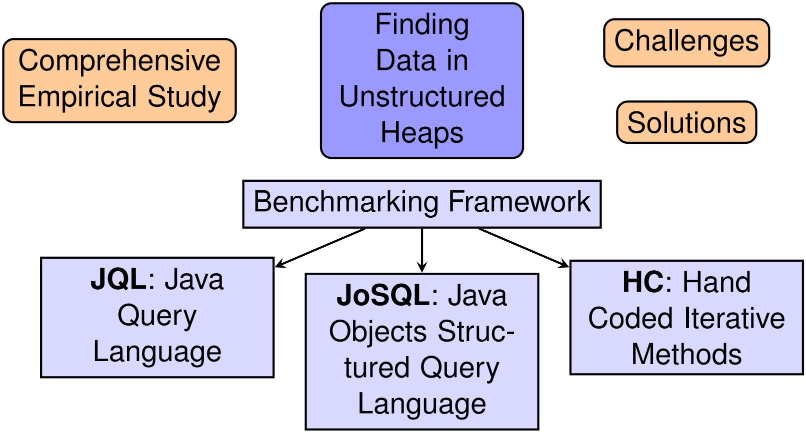
Overview of the Presentation



Overview of the Presentation



Overview of the Presentation



Overview of the Presentation

Experiments Reveal Trade-offs in Performance and Overall Viability

Comprehensive Empirical Study

Finding Data in Unstructured Heaps

Challenges

Solutions

Benchmarking Framework

- **JQL**: Java Query Language
- **JoSQL**: Java Objects Structured Query Language
- **HC**: Hand Coded Iterative Methods

Correctly and Efficiently Finding Objects in the Heap

The unstructured heap in a Java virtual machine stores objects that are connected in complex and unpredictable ways (Xu and Rountev, ICSE 2008)

Correctly and Efficiently Finding Objects in the Heap

The unstructured heap in a Java virtual machine stores objects that are connected in complex and unpredictable ways (Xu and Rountev, ICSE 2008)

When is an Object Allocated to the Heap?

```
LinkedList list = new LinkedList()
```

Correctly and Efficiently Finding Objects in the Heap

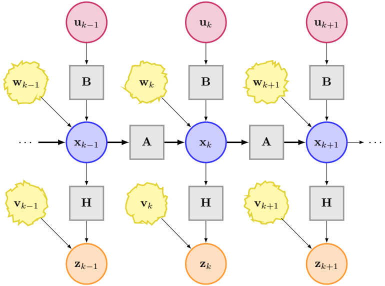
The unstructured heap in a Java virtual machine stores objects that are connected in complex and unpredictable ways (Xu and Rountev, ICSE 2008)

When is an Object Allocated to the Heap?

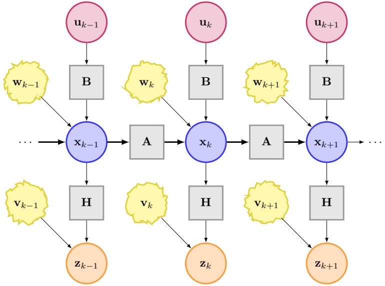
```
LinkedList list = new LinkedList()
```

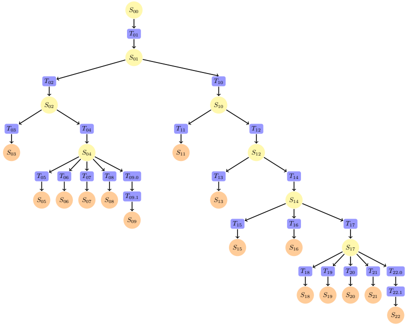
Let's Allocate Some Objects to the Heap!

Correctly and Efficiently Finding Objects in the Heap

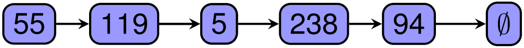


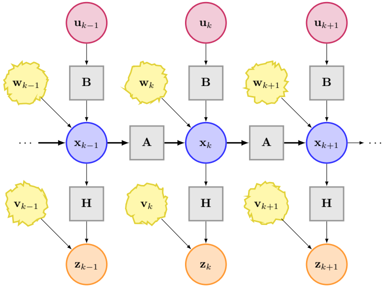
Correctly and Efficiently Finding Objects in the Heap

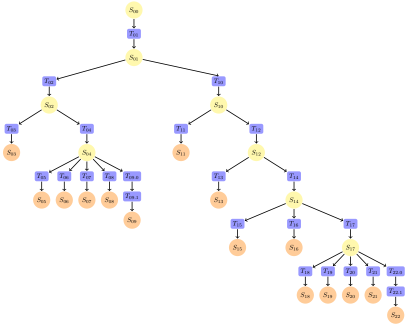




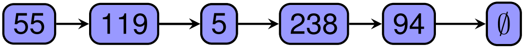
Correctly and Efficiently Finding Objects in the Heap

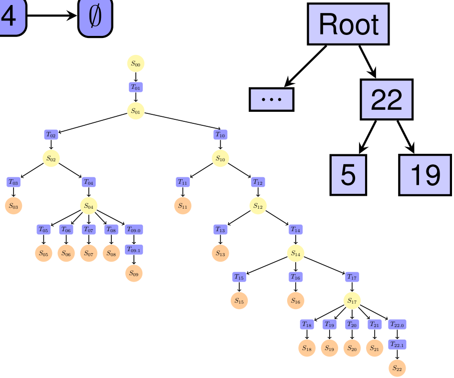


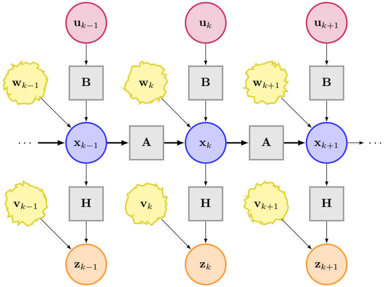




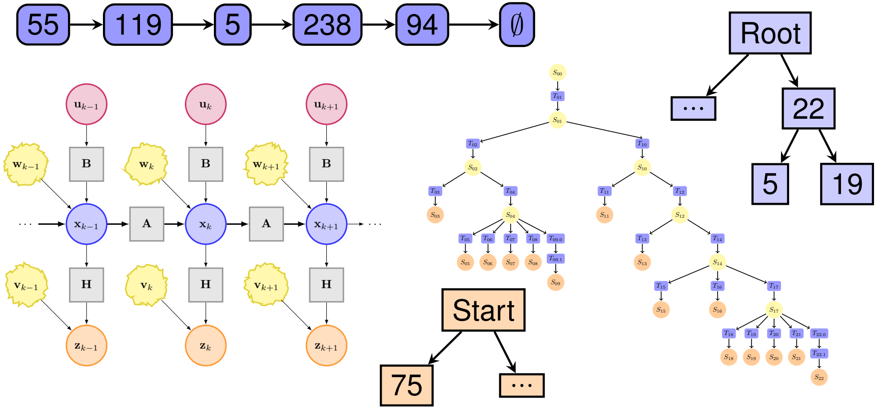
Correctly and Efficiently Finding Objects in the Heap



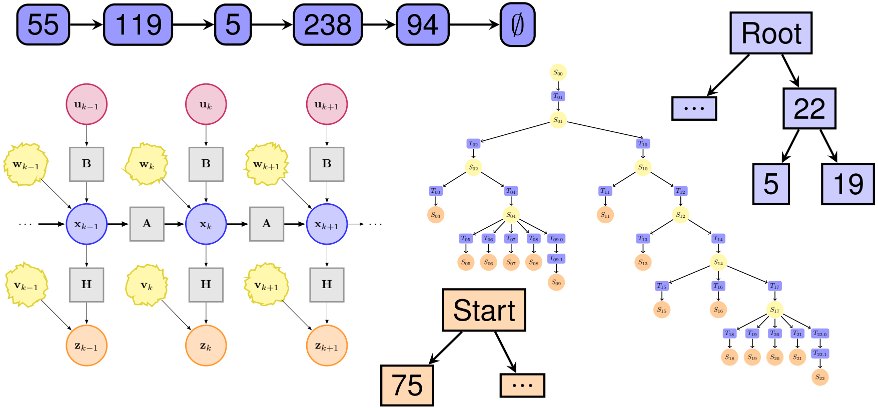




Correctly and Efficiently Finding Objects in the Heap

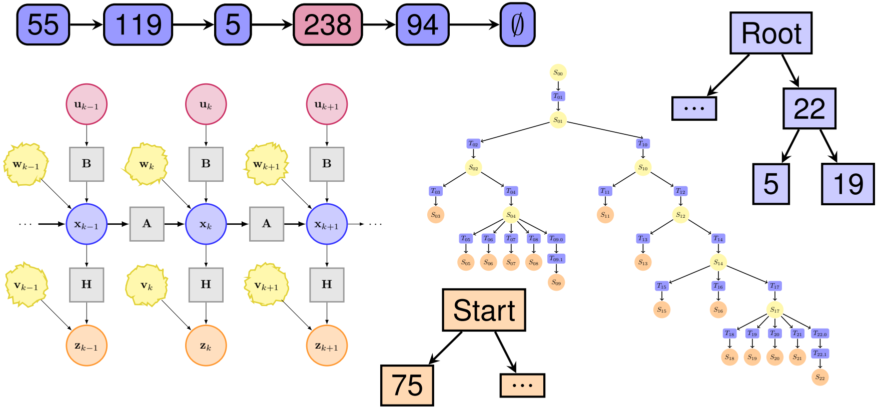


Correctly and Efficiently Finding Objects in the Heap



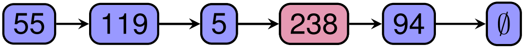
`LinkedList` Node(s) with Values Greater Than Those in the Trees

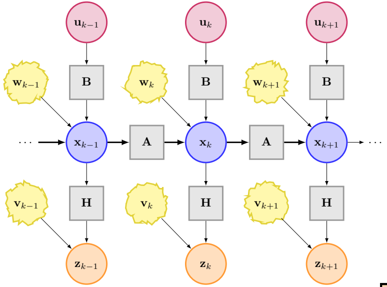
Correctly and Efficiently Finding Objects in the Heap

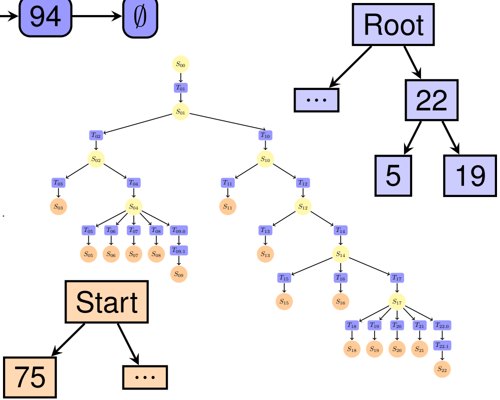


`LinkedList` Node(s) with Values Greater Than Those in the Trees

Correctly and Efficiently Finding Objects in the Heap

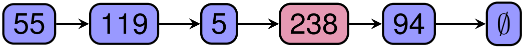


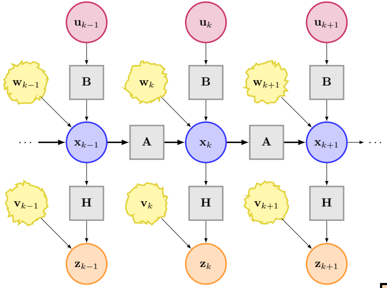


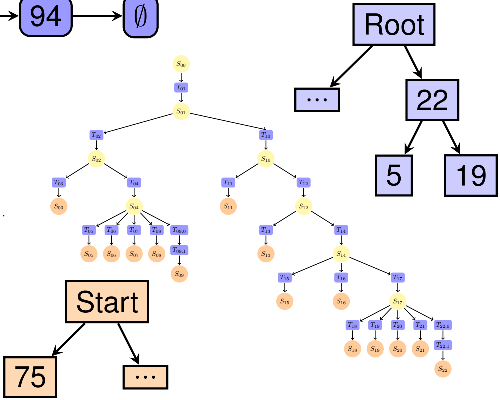


How Do We Find These Nodes?

Correctly and Efficiently Finding Objects in the Heap



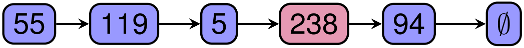


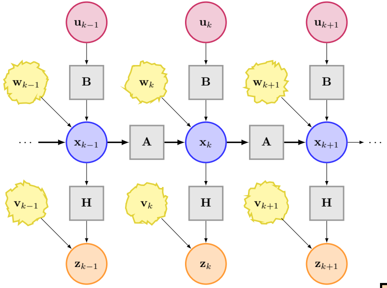


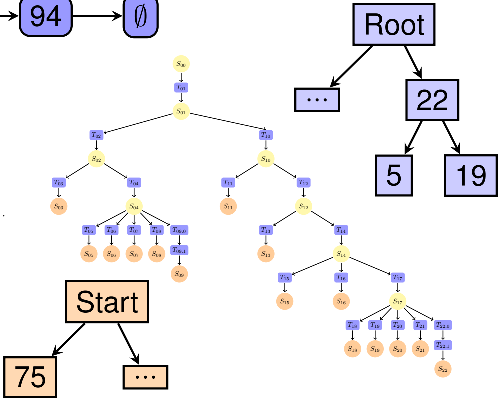
How Do We Find These Nodes?

Imperative - Give the Procedure

Correctly and Efficiently Finding Objects in the Heap



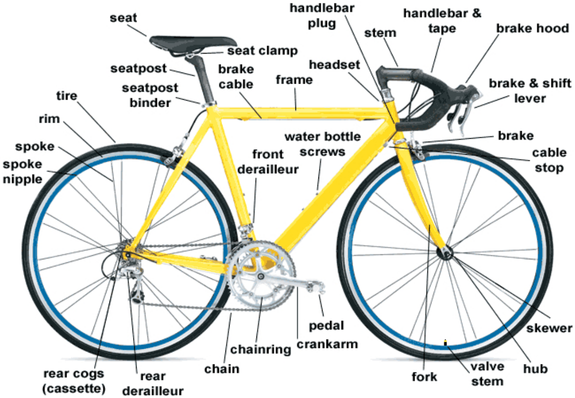




How Do We Find These Nodes?

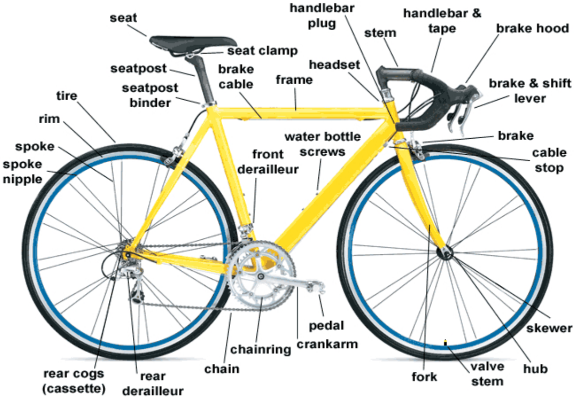
Declarative - Give the Specification

Object Query Languages and Bicycles



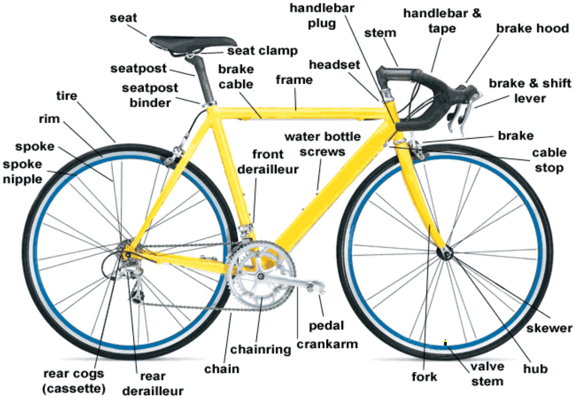
Efficiency - Bicycle: Low wind resistance and time to destination

Object Query Languages and Bicycles



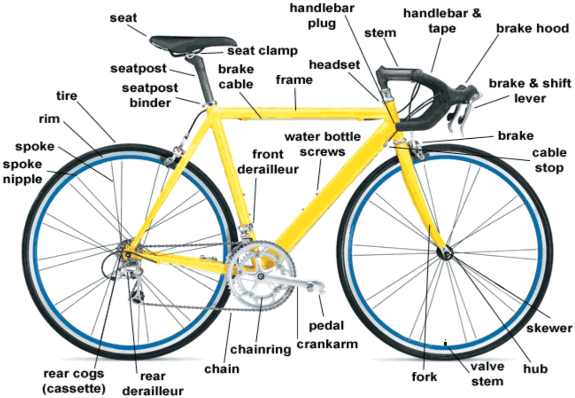
Efficiency - Query: Minimal space overhead and a low response time

Object Query Languages and Bicycles



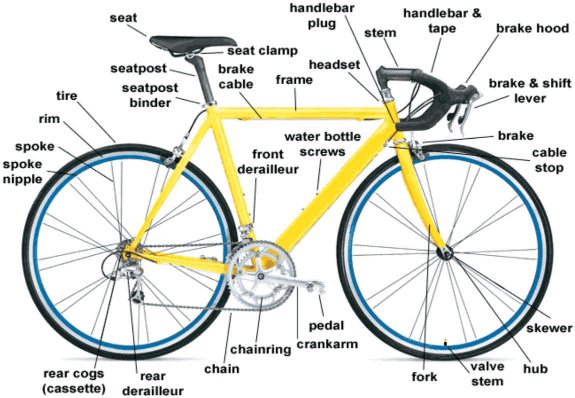
Effectiveness - Bicycle: Transports all item(s) with no break downs

Object Query Languages and Bicycles



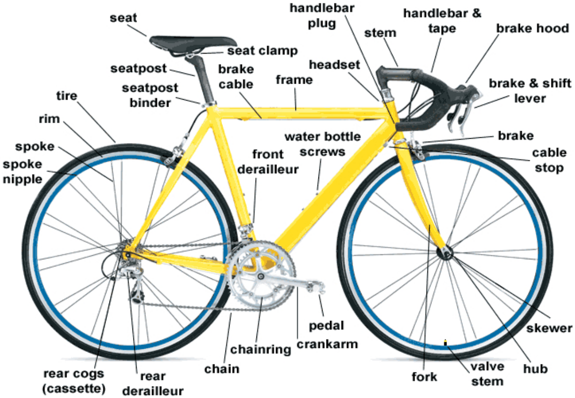
Effectiveness - Query: Always returns the correct result(s) to a query

Object Query Languages and Bicycles



Cost - Bicycle: Frame material(s) and components cause price to vary

Object Query Languages and Bicycles

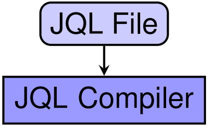


Cost - Query: Must consider installation and development challenges

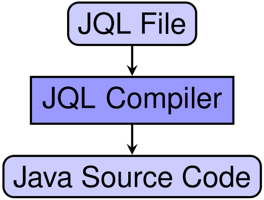
JQL: Java Query Language

JQL Compiler

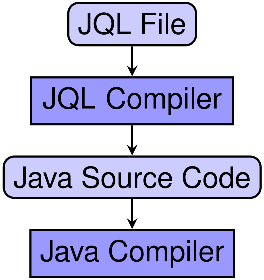
JQL: Java Query Language



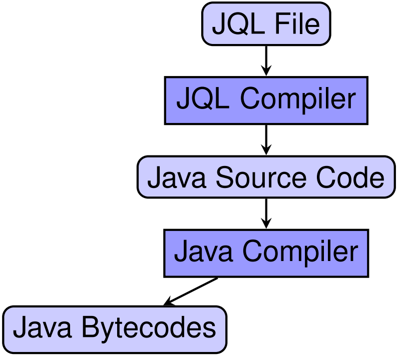
JQL: Java Query Language



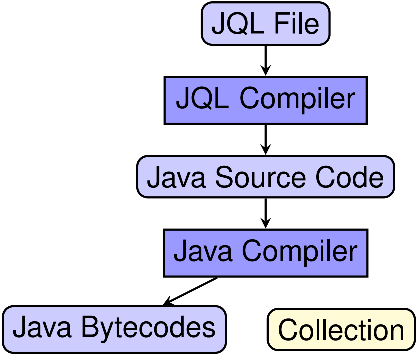
JQL: Java Query Language



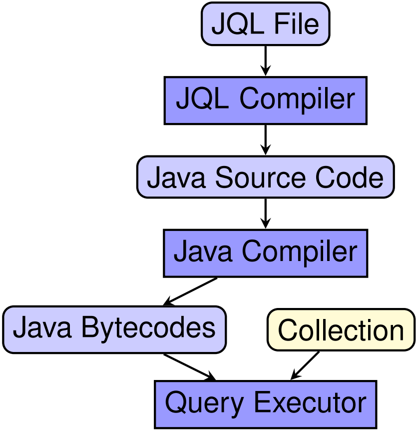
JQL: Java Query Language



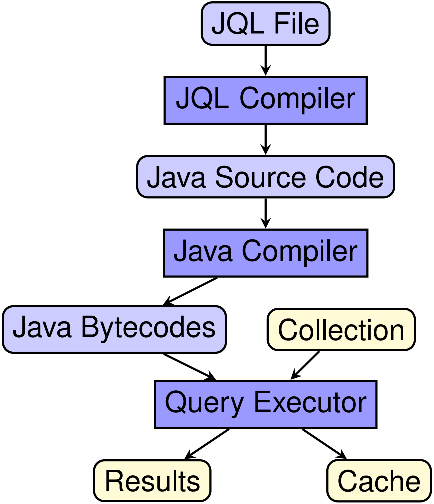
JQL: Java Query Language



JQL: Java Query Language



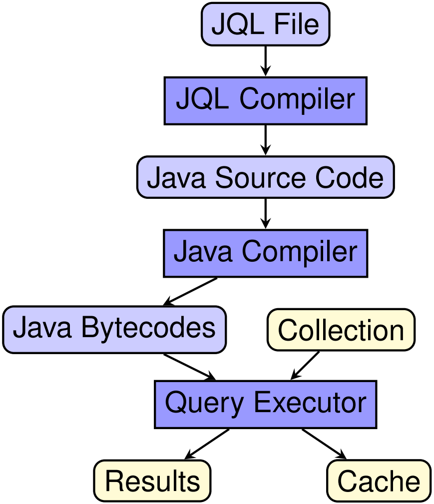
JQL: Java Query Language



JQL: Java Query Language

Features

- Pre-compilation
- AOP with AspectJ
- Method Queries
- Caching
- Optimizations



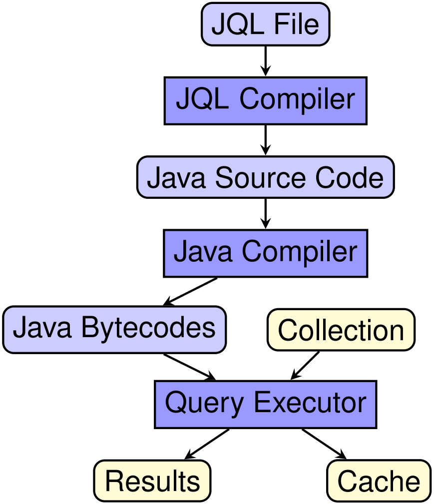
JQL: Java Query Language

Features

- Pre-compilation
- AOP with AspectJ
- Method Queries
- Caching
- Optimizations

References

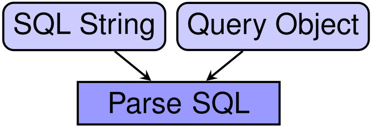
- (Willis et al. ECOOP 2006)
- (Willis et al. OOPSLA 2008)



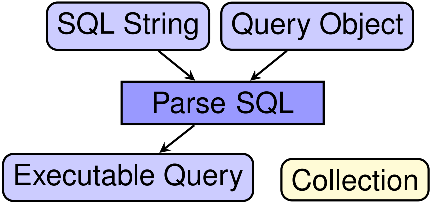
JoSQL: Java Objects Structured Query Language

Parse SQL

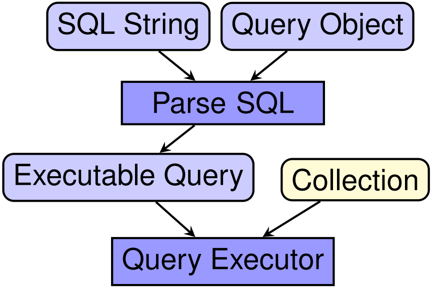
JoSQL: Java Objects Structured Query Language



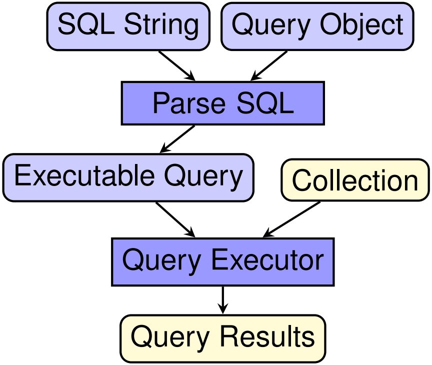
JoSQL: Java Objects Structured Query Language



JoSQL: Java Objects Structured Query Language



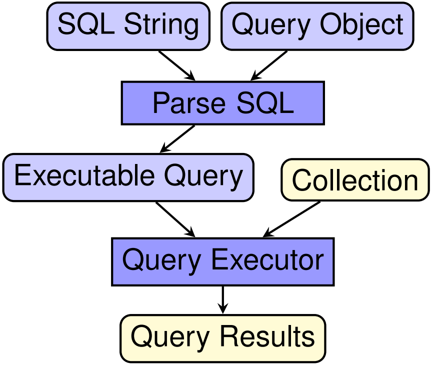
JoSQL: Java Objects Structured Query Language



JoSQL: Java Objects Structured Query Language

Features

- SQL Statements
- String Parsing
- Java Reflection
- Query Facilities



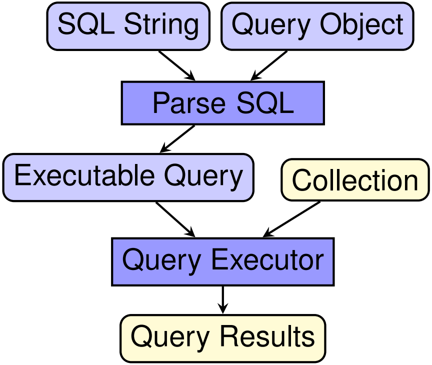
JoSQL: Java Objects Structured Query Language

Features

- SQL Statements
- String Parsing
- Java Reflection
- Query Facilities

Reference

- http://josql.sf.net



Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

JQL: Compile-Time

Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

JQL: Compile-Time

JoSQL: Run-Time

Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

JQL: Compile-Time

JoSQL: Run-Time

Performance Trade-Offs?

Comparison of Data Finding Methods

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

JQL: Compile-Time

JoSQL: Run-Time

Performance Trade-Offs?

Effectiveness Concerns?

Comparison of Data Finding Methods

Benchmarking Framework Helps to Answer These Questions

As the number of collections and objects increases, imperative programming may lead to applications that are complicated, error-prone, and hard to maintain (Xu and Rountev, ICSE 2008)

JQL: Compile-Time

JoSQL: Run-Time

Performance Trade-Offs?

Effectiveness Concerns?

Benchmarking Framework to Evaluate Query Methods

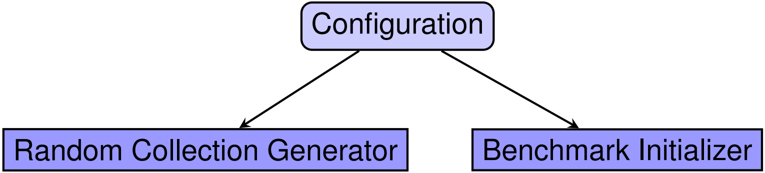
Random Collection Generator

Benchmarking Framework to Evaluate Query Methods

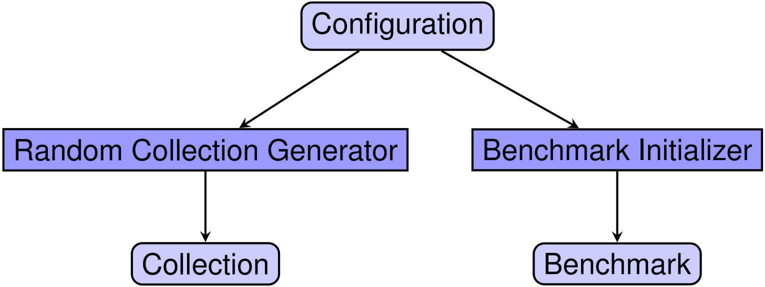
Random Collection Generator

Benchmark Initializer

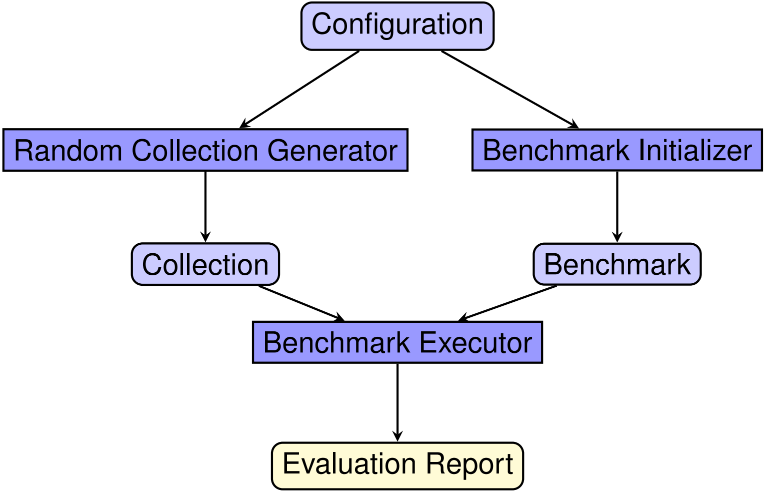
Benchmarking Framework to Evaluate Query Methods



Benchmarking Framework to Evaluate Query Methods



Benchmarking Framework to Evaluate Query Methods



Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

Sizes

Methods

Explored a **wide variety** of benchmark **configurations**

Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

Sizes

Methods

What **operations** do we run to **evaluate** the query methods?

Configuration of the Benchmarking Framework

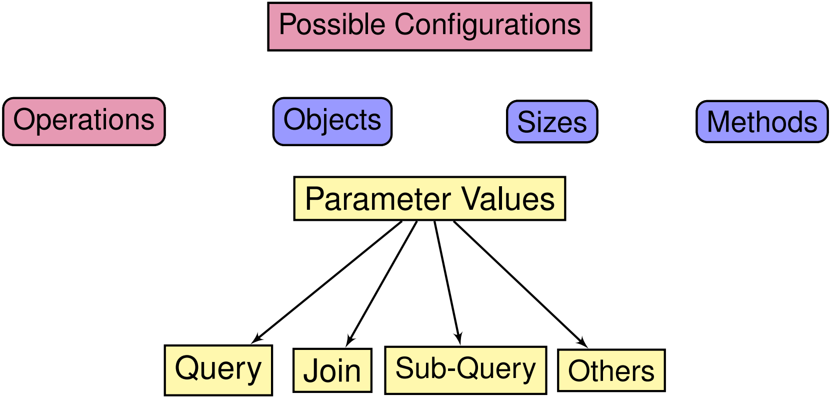
Possible Configurations

Operations Objects Sizes Methods

Parameter Values

What **operations** do we run to **evaluate** the query methods?

Configuration of the Benchmarking Framework



What **operations** do we run to **evaluate** the query methods?

Configuration of the Benchmarking Framework

Possible Configurations

Operations | Objects | Sizes | Methods

What **objects** will we **allocate** to the JVM's **heap**?

Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

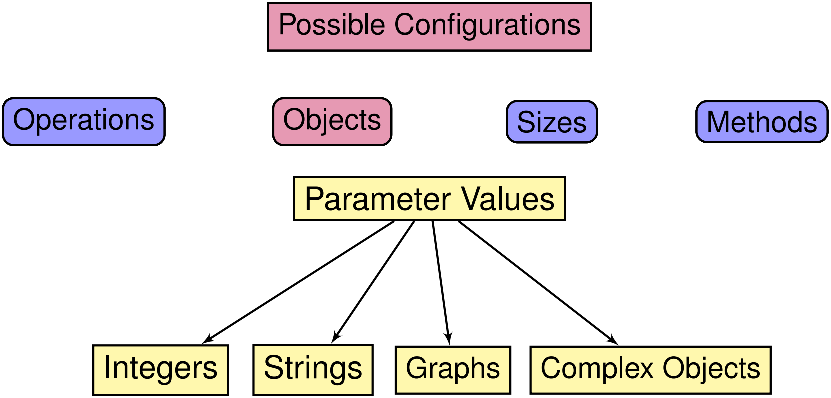
Sizes

Methods

Parameter Values

What **objects** will we **allocate** to the JVM's **heap**?

Configuration of the Benchmarking Framework



What **objects** will we **allocate** to the JVM's **heap**?

Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

Sizes

Methods

How **big** should we make the **objects** and the **collections**?

Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

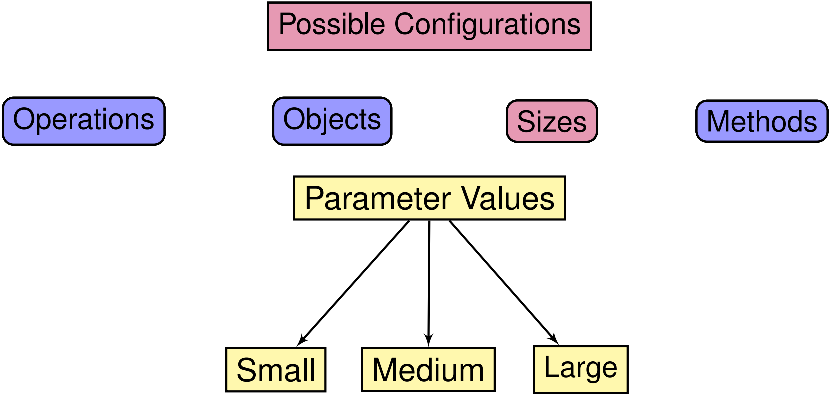
Sizes

Methods

Parameter Values

How **big** should we make the **objects** and the **collections**?

Configuration of the Benchmarking Framework



How **big** should we make the **objects** and the **collections**?

Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

Sizes

Methods

Which **methods** should be part of the **framework**?

Configuration of the Benchmarking Framework

Possible Configurations

Operations

Objects

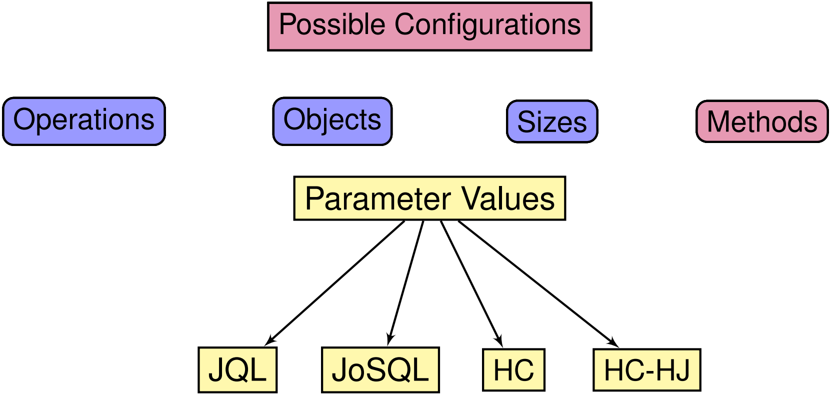
Sizes

Methods

Parameter Values

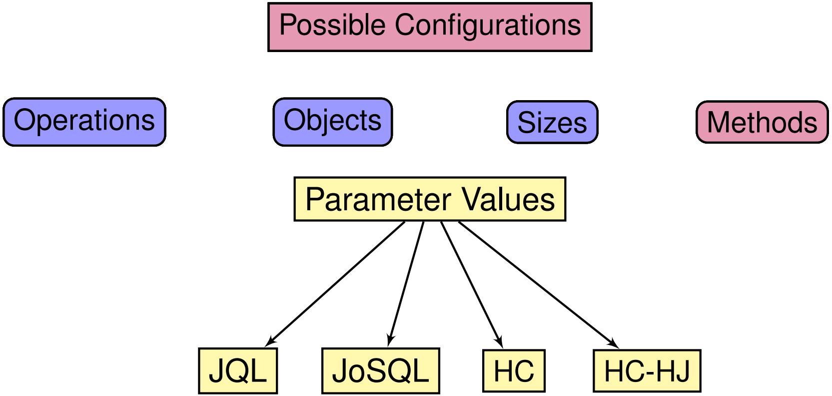
Which **methods** should be part of the **framework**?

Configuration of the Benchmarking Framework



Which **methods** should be part of the **framework**?

Configuration of the Benchmarking Framework



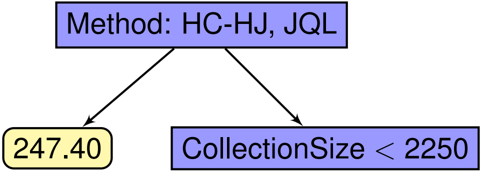
See the **paper** for further **operator** and **configuration** details

Analysis Techniques: Regression Tree Models

Method: HC-HJ, JQL

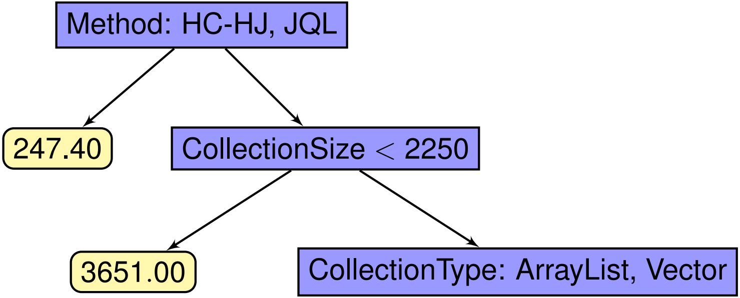
Tree Models: Recursive partitioning creates hierarchical view of data

Analysis Techniques: Regression Tree Models



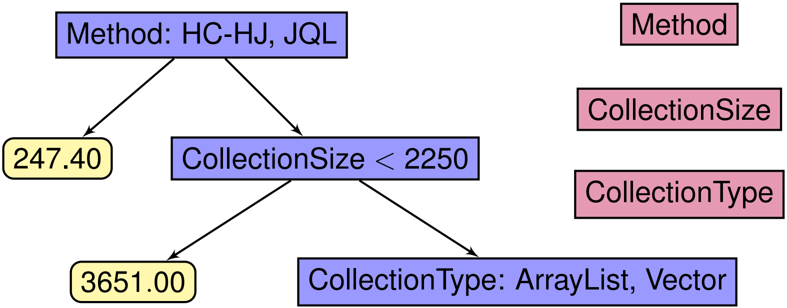
Tree Models: Recursive partitioning creates hierarchical view of data

Analysis Techniques: Regression Tree Models



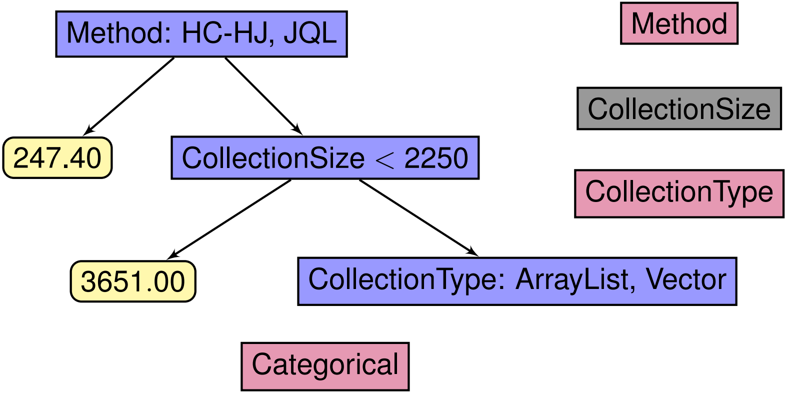
Tree Models: Recursive partitioning creates hierarchical view of data

Analysis Techniques: Regression Tree Models



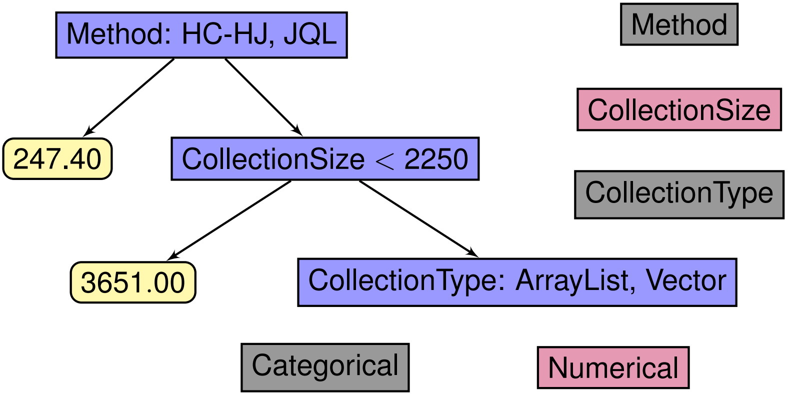
Explanatory Variable: Configuration of the benchmarking framework

Analysis Techniques: Regression Tree Models



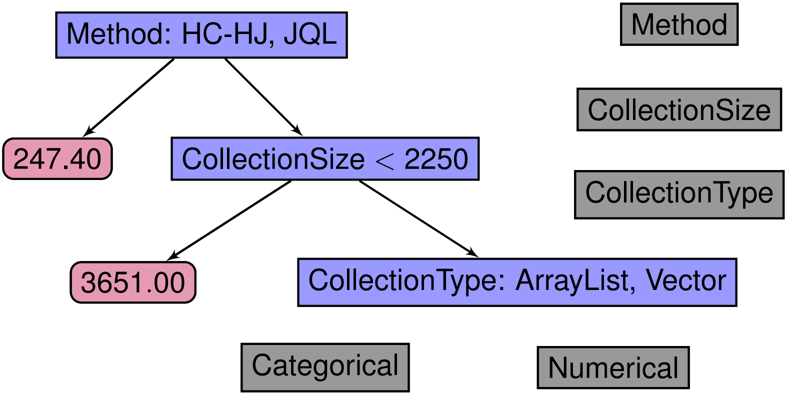
Non-parametric techniques that handles **different** variable types

Analysis Techniques: Regression Tree Models



Non-parametric techniques that handles **different** variable types

Analysis Techniques: Regression Tree Models

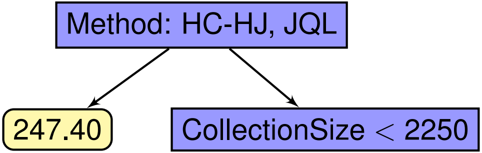


Response Variable: Response time of the benchmark

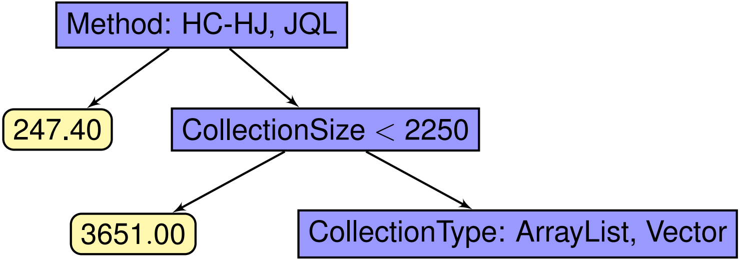
Join Benchmark with Integers and Strings

Method: HC-HJ, JQL

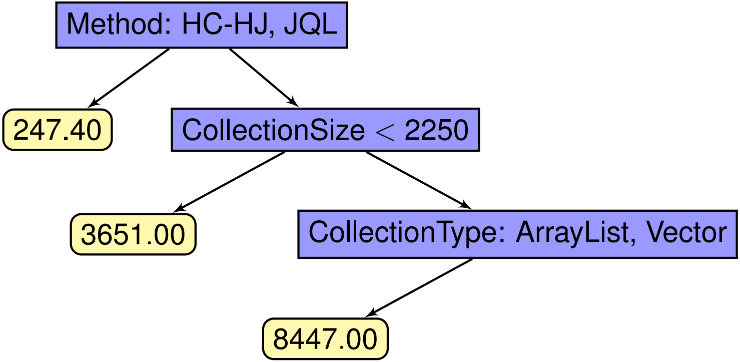
Join Benchmark with Integers and Strings



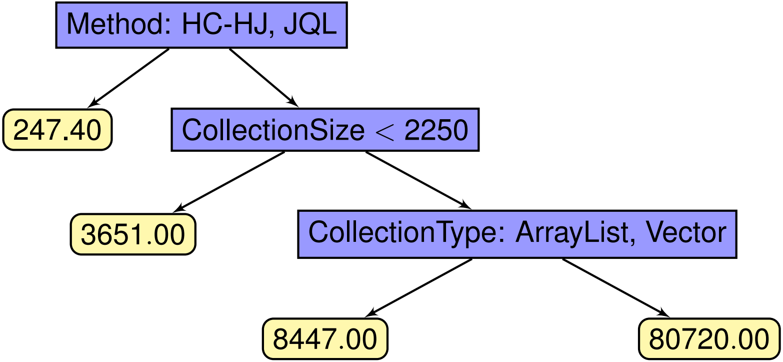
Join Benchmark with Integers and Strings



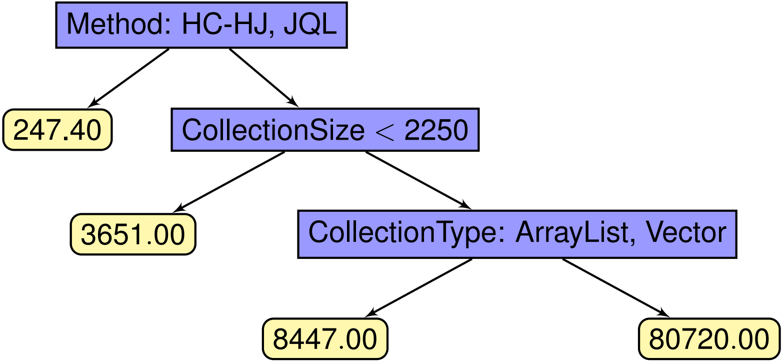
Join Benchmark with Integers and Strings



Join Benchmark with Integers and Strings

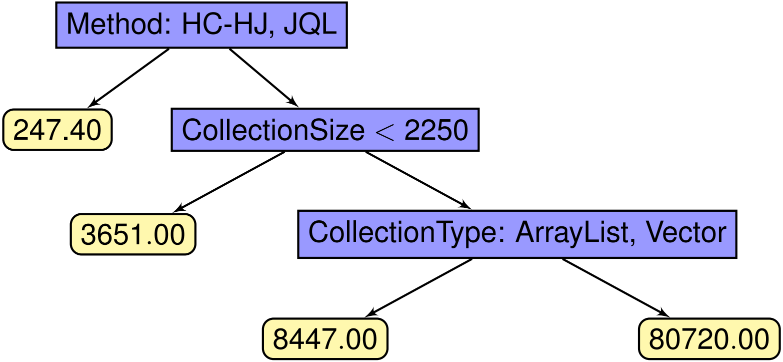


Join Benchmark with Integers and Strings



Reflection's Impact: HC-HJ and JQL exhibit lower values than JoSQL

Join Benchmark with Integers and Strings



Reflection's Impact: LinkedList further degrades JoSQL's performance

Impact of Object Size on Joining

Small Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	57.2	390.2	981.8
HC-HJ	69.3	378.1	923.5
JoSQL	997.3	3620.2	8823.1

Impact of Object Size on Joining

Small Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	57.2	390.2	981.8
HC-HJ	69.3	378.1	923.5
JoSQL	997.3	3620.2	8823.1

Large Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	35.4	80.8	255.4
HC-HJ	11.4	63.3	217.8
JoSQL	930.3	3107.3	8165.9

Impact of Object Size on Joining

Small Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	57.2	390.2	981.8
HC-HJ	69.3	378.1	923.5
JoSQL	997.3	3620.2	8823.1

Large Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	35.4	80.8	255.4
HC-HJ	11.4	63.3	217.8
JoSQL	930.3	3107.3	8165.9

Impact of Object Size on Joining

Small Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	57.2	390.2	981.8
HC-HJ	69.3	378.1	923.5
JoSQL	997.3	3620.2	8823.1

Large Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	35.4	80.8	255.4
HC-HJ	11.4	63.3	217.8
JoSQL	930.3	3107.3	8165.9

Impact of Object Size on Joining

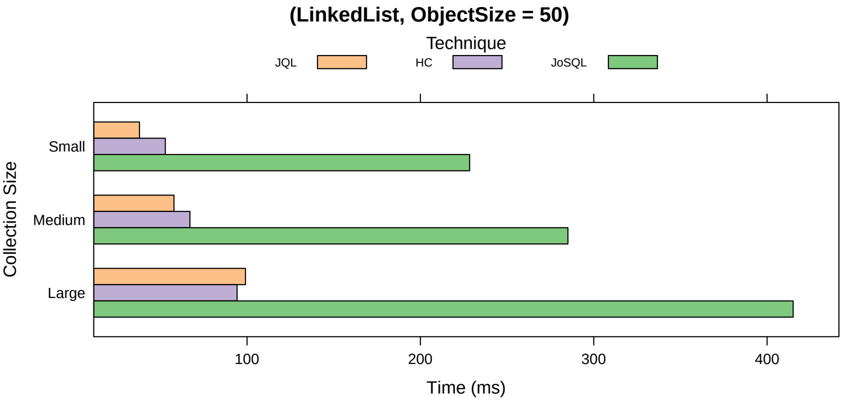
Small Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	57.2	390.2	981.8
HC-HJ	69.3	378.1	923.5
JoSQL	997.3	3620.2	8823.1

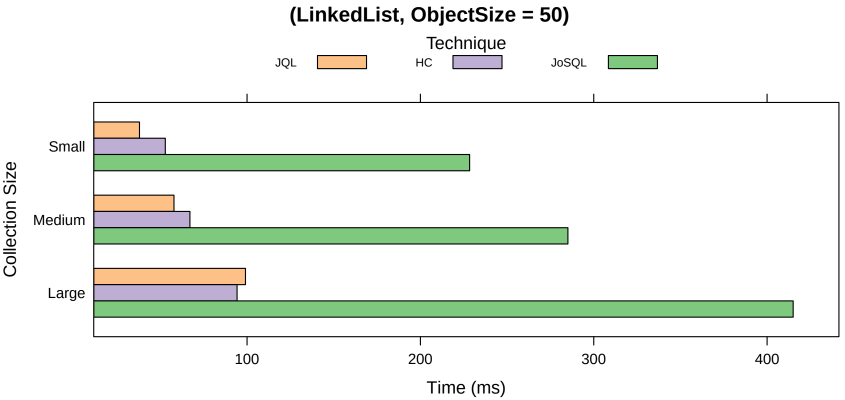
Large Objects

	Collection Size		
Method	**Small**	**Medium**	**Large**
JQL	35.4	80.8	255.4
HC-HJ	11.4	63.3	217.8
JoSQL	930.3	3107.3	8165.9

SubQuery Benchmark with Graphs Containing Strings

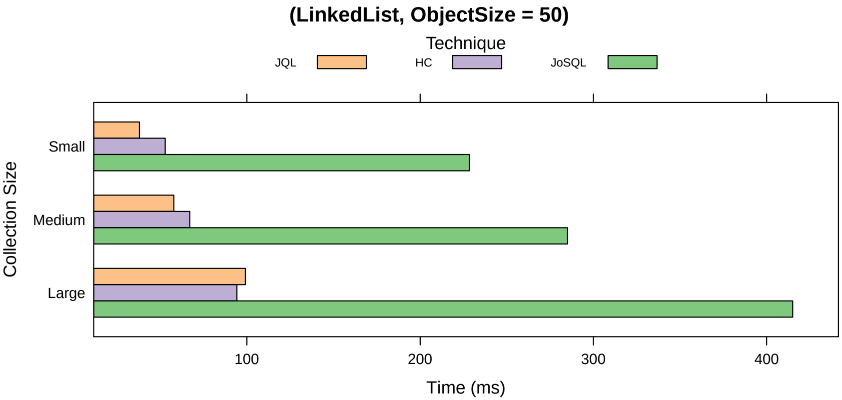


SubQuery Benchmark with Graphs Containing Strings



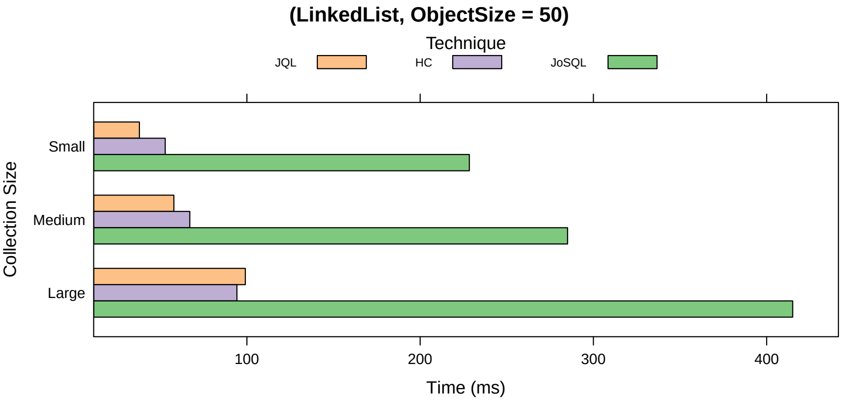
JQL is Faster Than HC When the Collection Size is Small and Medium

SubQuery Benchmark with Graphs Containing Strings



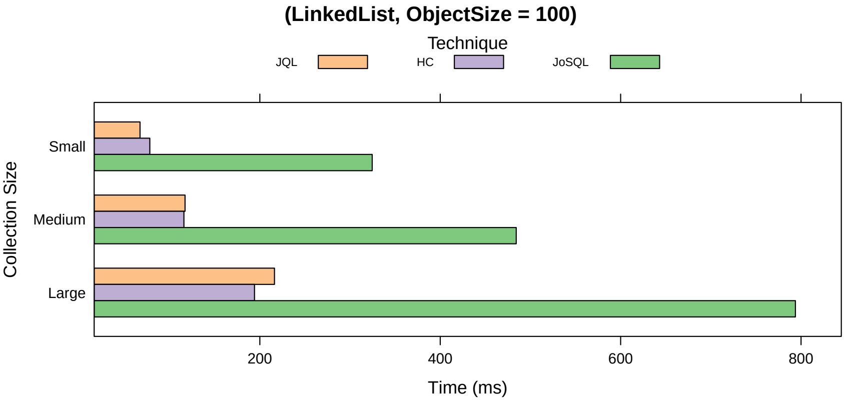
HC is Faster Than JQL When the Collection Size is Large

SubQuery Benchmark with Graphs Containing Strings



Why? JQL Must Track All of the Objects in the Heap

SubQuery Benchmark with Graphs Containing Strings



Trend is Even More Pronounced as the Object Size Increases

Conclusions and Future Work

Concluding Remarks

- Comprehensive empirical study of query methods
- Interesting trends concerning JQL, JoSQL, HC, and HC-HJ
- Refer to the paper for many more insights

Future Work

- Integrate new benchmarks and object types
- Consider different sizes of objects and collections
- Incorporate different data finding methods
- Leverage additional statistical analysis techniques

Conclusions and Future Work

Concluding Remarks

- Comprehensive empirical study of query methods
- Interesting trends concerning JQL, JoSQL, HC, and HC-HJ
- Refer to the paper for many more insights

Future Work

- Integrate new benchmarks and object types
- Consider different sizes of objects and collections
- Incorporate different data finding methods
- Leverage additional statistical analysis techniques

Ask and You Shall Receive: Empirically Evaluating Declarative Approaches to Finding Data in Unstructured Heaps

Thank you for your attention!
Questions?

"Ask, and you will receive. Search, and you will find. Knock, and the door will be opened for you."
Matthew 7:7 (GWT) http://bible.cc/matthew/7-7.htm