

---

# Optimizing Concurrent Storage and Retrieval Operations for Real-Time Surveillance Applications

**Jacob Nikom**

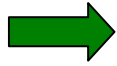
MIT Lincoln Laboratory

March 26, 2009

06:53:12

# Outline

---



- **Introduction**
  - **Data Sources**
  - **Data Rates**
  - **Archiving Data Flow**
- **Hardware Selection**
- **Database Server Selection**
- **Schema Design**
- **Optimizing Search Operations**
- **Gauging Concurrent Insert and Search Operations**
- **Summary**

# Data Sources

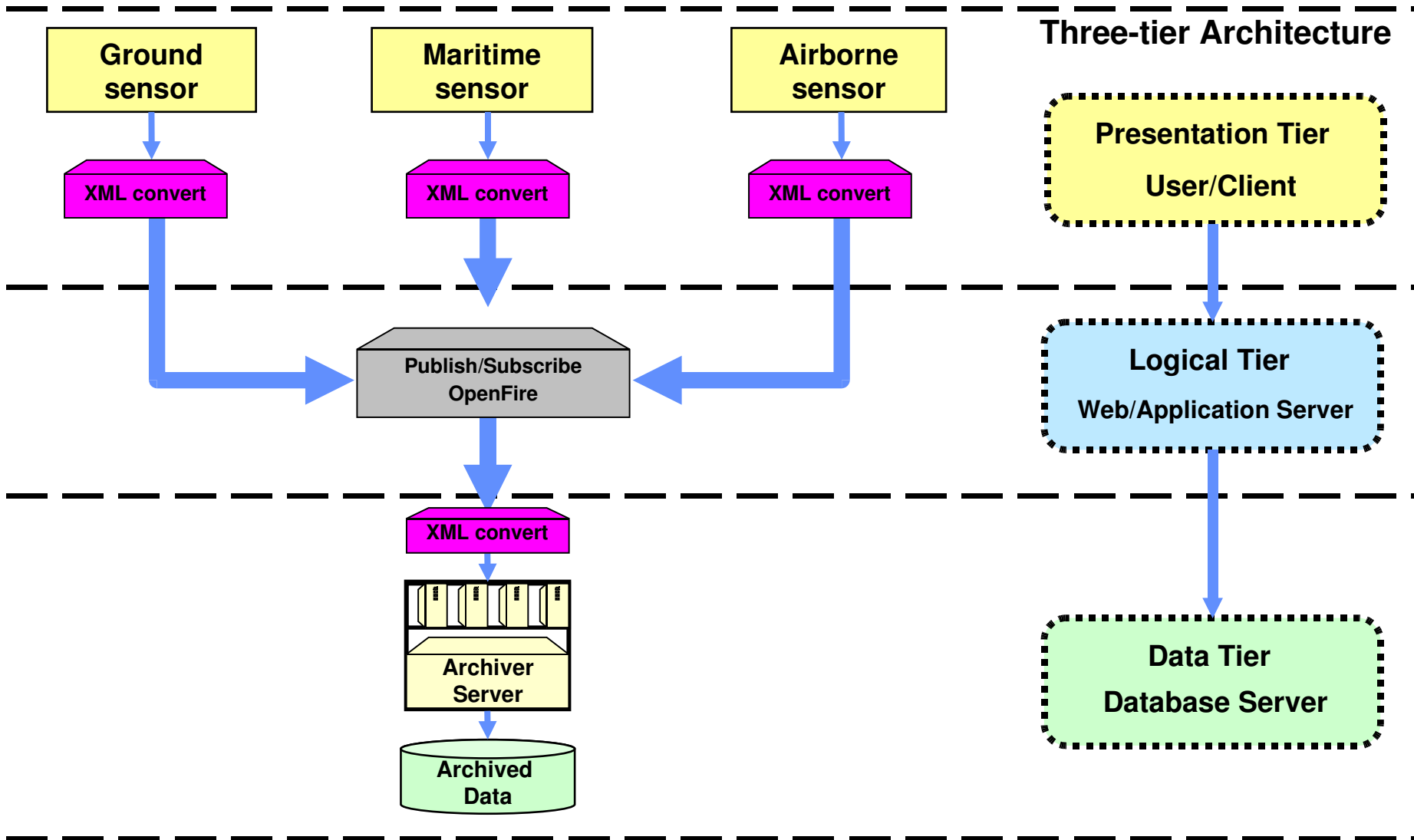


# Data Rates

Max Feed Rates (msg/sec)	Message Size (Numerical) (bytes)	Numerical Data Throughput (KB/sec)	Message Size (XML) (bytes)	XML Data Throughput (KB/sec)
453	300	132.4	3000	1359


**Incoming data have to be stored in real time!**

# Archiving Data Flow



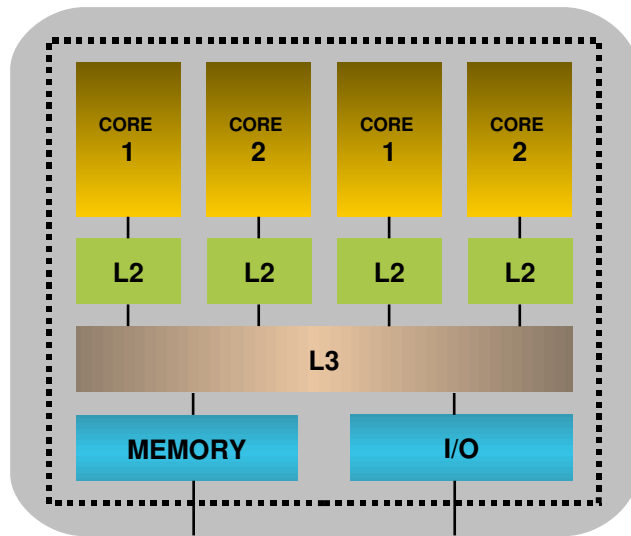
# Outline

---

- Introduction
-  • **Hardware Selection**
  - Hardware Architecture Comparison
  - Opteron — Xeon Conversion Performance
- Database Server Selection
- Schema Design
- Optimizing Search Operations
- Gauging Concurrent Insert and Search Operations
- Summary

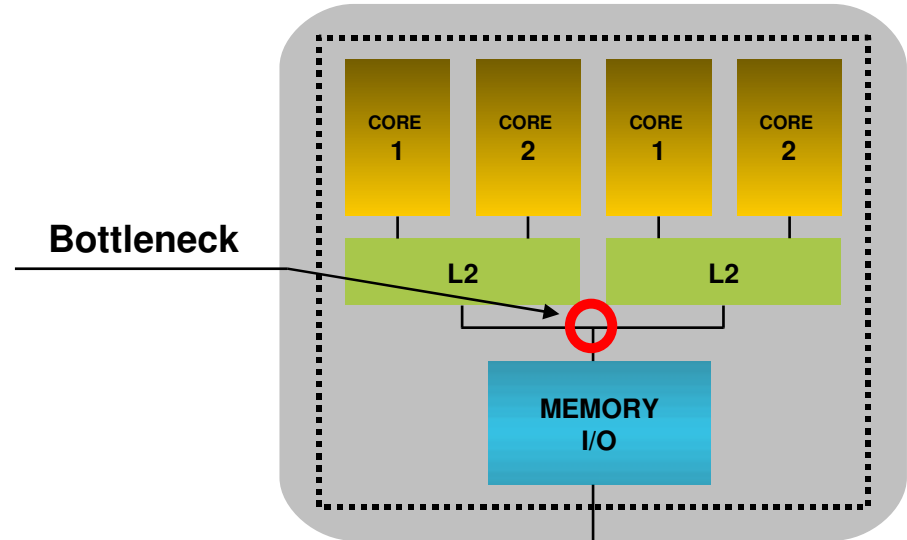
# Hardware Architecture Comparison

## Opteron



Each core connects directly to the memory using Direct Connect architecture

## Xeon



As the number of processing cores increases, so does competition for access to main memory

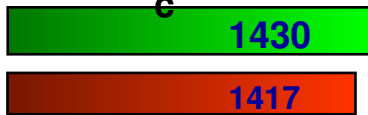
# Opteron—Xeon Conversion Performance

AMD 2.5 GHz [65nm](#) Barcelona vs Intel 3.0 GHz [45nm](#) Xeon [AnandTech 11/27/2007]

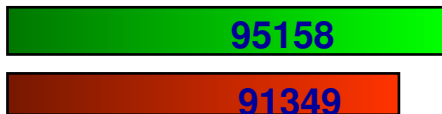
Dual Opteron 2360 2.5 GHz ■

Dual Xeon 5472 3.0 GHz ■

Queries/se

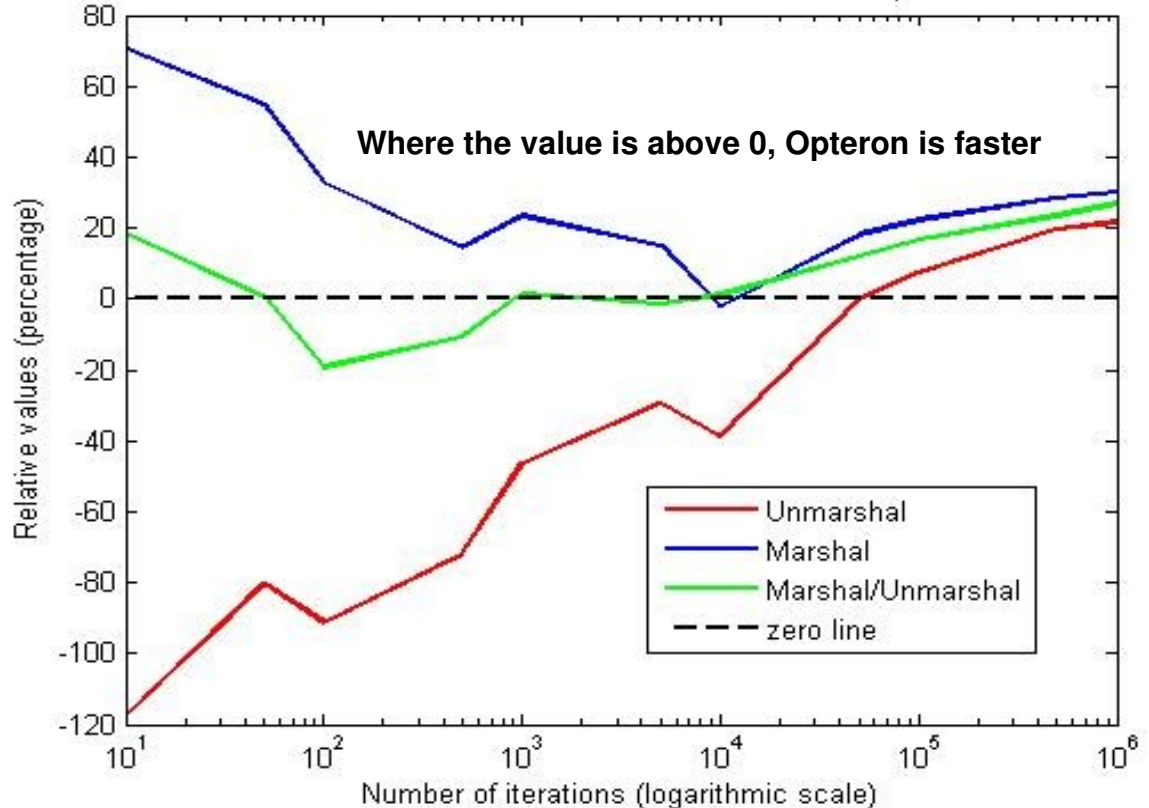


SPECjbb2005



64-bit MySQL 5.1.22,  
SLES 10, SUN JDK 1.6.0\_02


Relative differences in CPU time for JAXB conversion between Opteron and Xeon CPUs



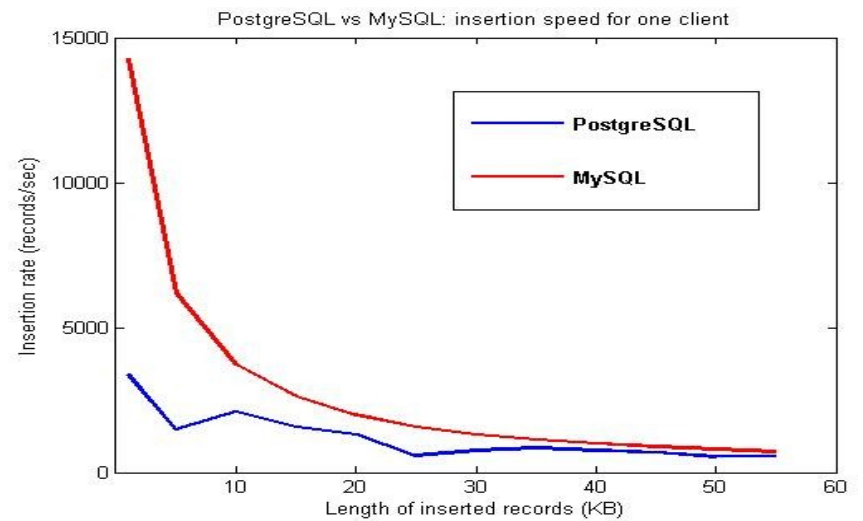
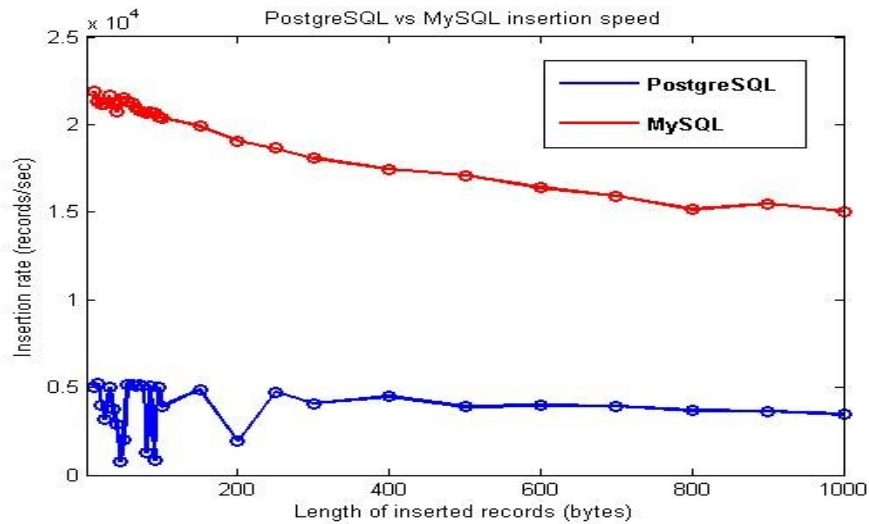
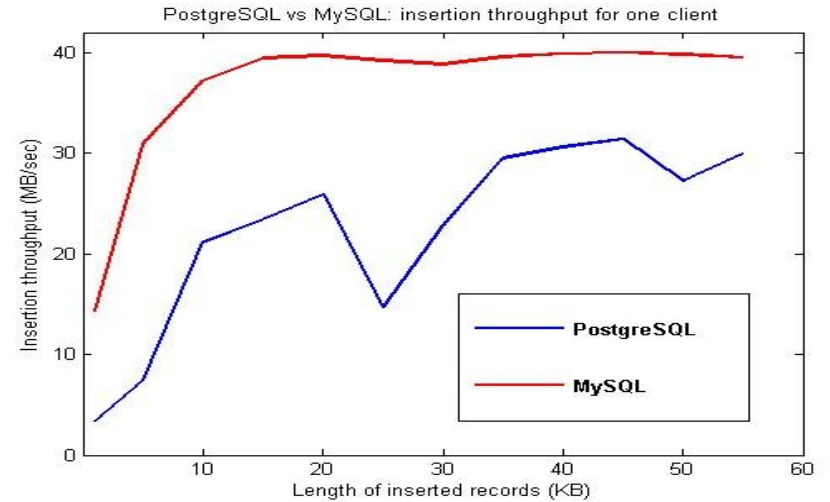
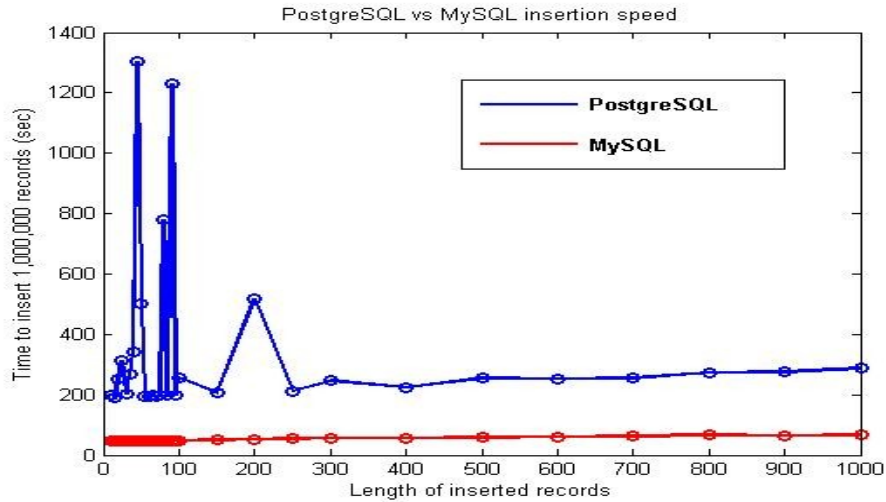
**Opteron-based systems deliver better I/O with comparable CPU power**

# Outline

---

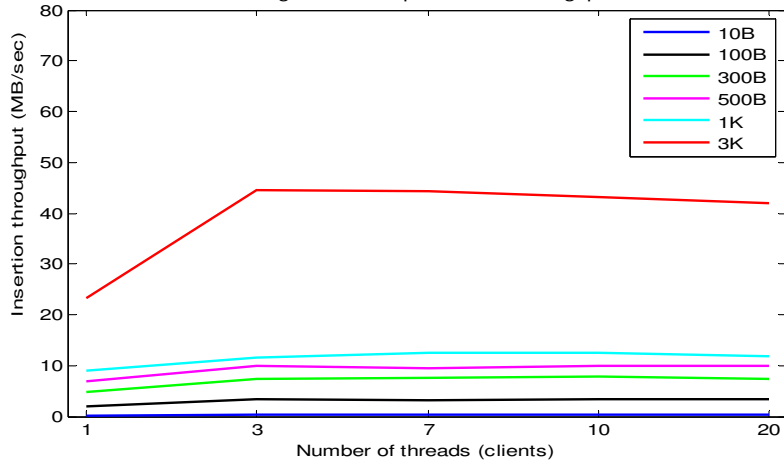
- Introduction
- Hardware Selection
-  • Database Server Selection
  - Single Client Performance
  - Multiple Clients Performance
- Schema Design
- Optimizing Search Operations
- Gauging Concurrent Insert and Search Operations
- Summary

# Single Client Performance

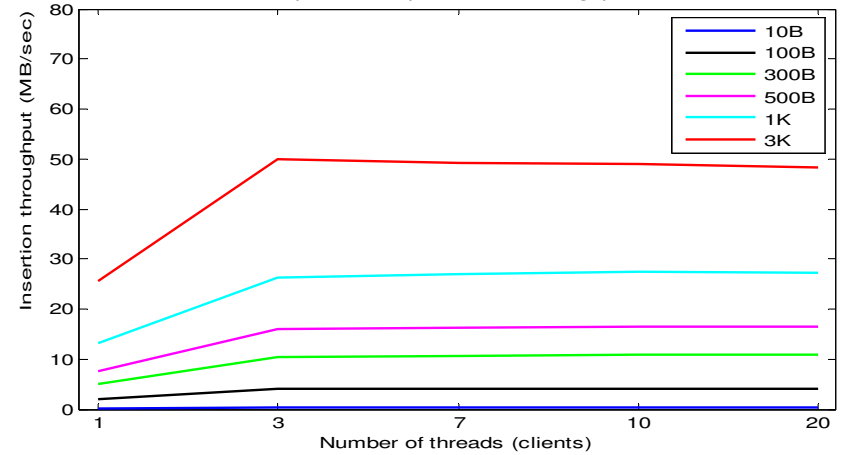


# Multiple Clients Performance

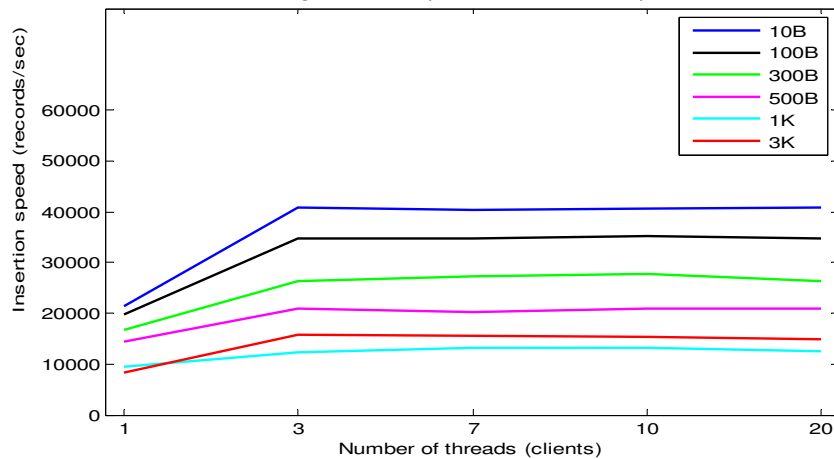
PostgreSQL Multiple Clients Throughput



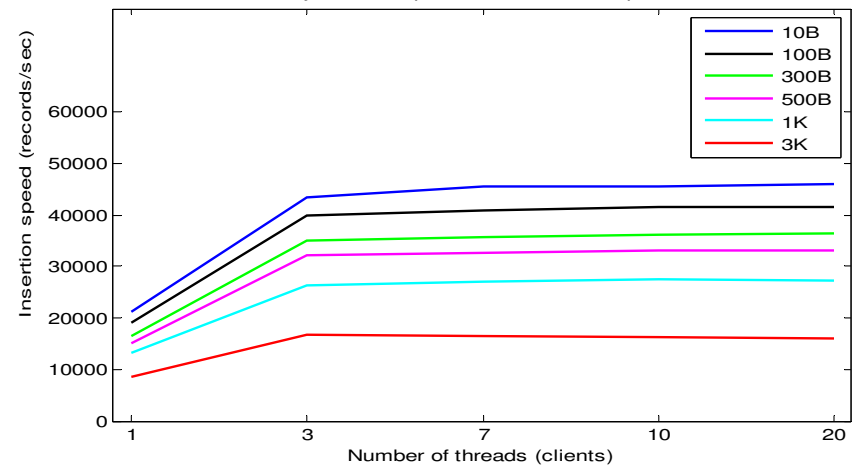
MySQL Multiple Clients Throughput



PostgreSQL Multiple Clients Insertion speed




MySQL Multiple Clients Insertion speed



**MySQL server (MyISAM tables) has higher insertion rate and throughput**

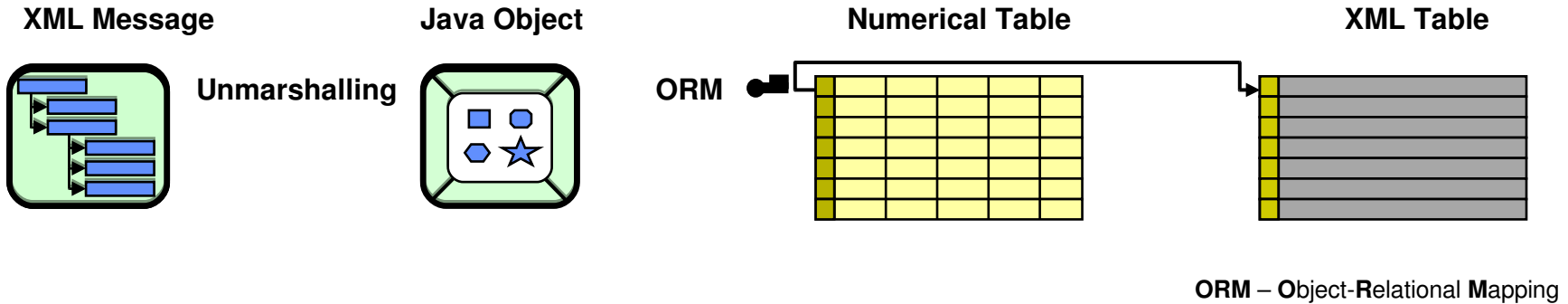
# Outline

---

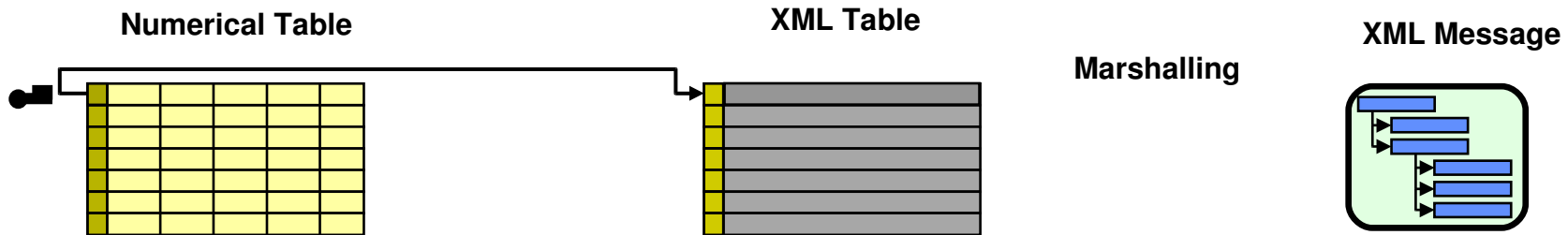
- Introduction
- Hardware Selection
-  • Database Server Selection
- Schema Design
  - Initial Proposals
  - Alternative Proposals
  - JAXB Conversion Time Measurements
- Optimizing Search Operations
- Gauging Concurrent Insert and Search Operations
- Summary

# Initial Proposals

## Data Insertion Operation



## Data Retrieval Operation



### Advantages

- Bypasses the marshalling of Java object into XML
- Storage of original XML messages
- Direct retrieval of XML messages

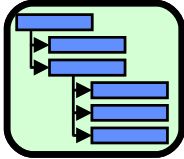
### Disadvantages

- More complex schema
- Requires transactions to synchronize the records

# Alternative Proposals

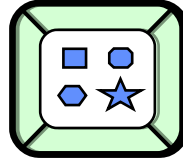
## Data Insertion

XML Message



Unmarshalling

Java Object



ORM

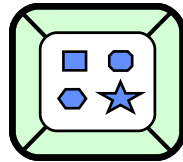
Numerical Table


## Data Retrieval

Numerical Table

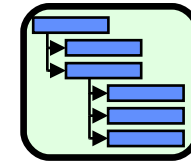

ORM

Java Object



Marshalling

XML Message



### Advantages

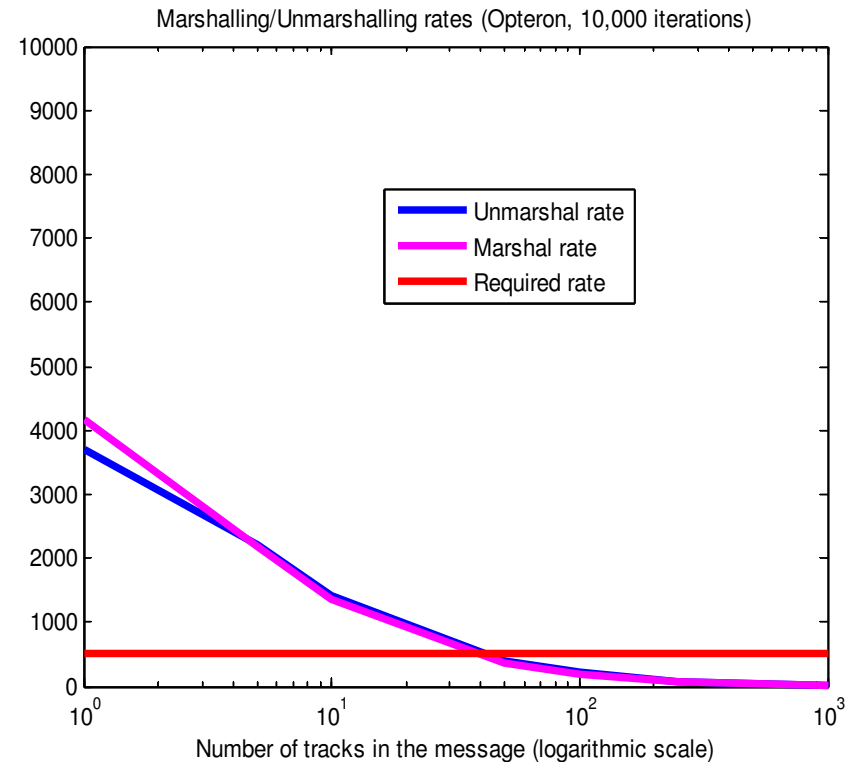
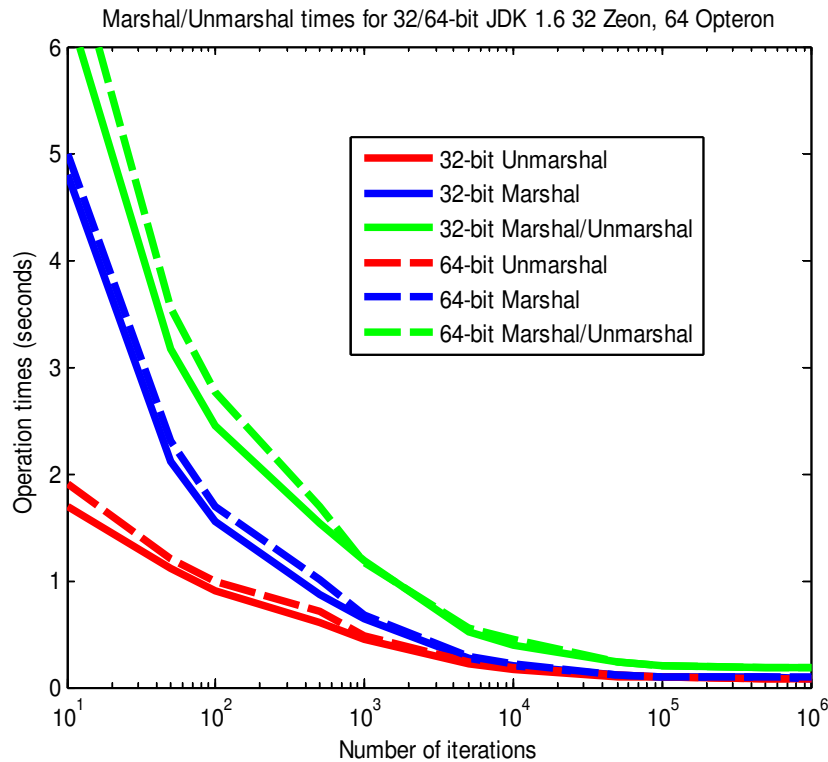
- Bypasses the storage of long XML messages in the table
- Simplifies schema (only one table)
- Simplifies and accelerates the data retrieval

### Disadvantages

- Requires marshalling the object into XML string

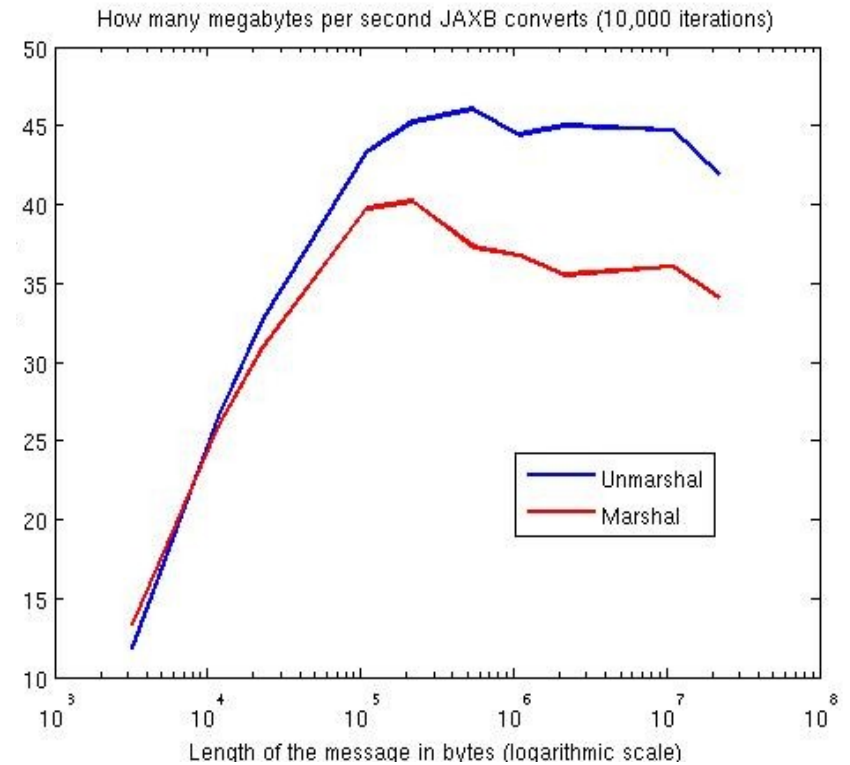
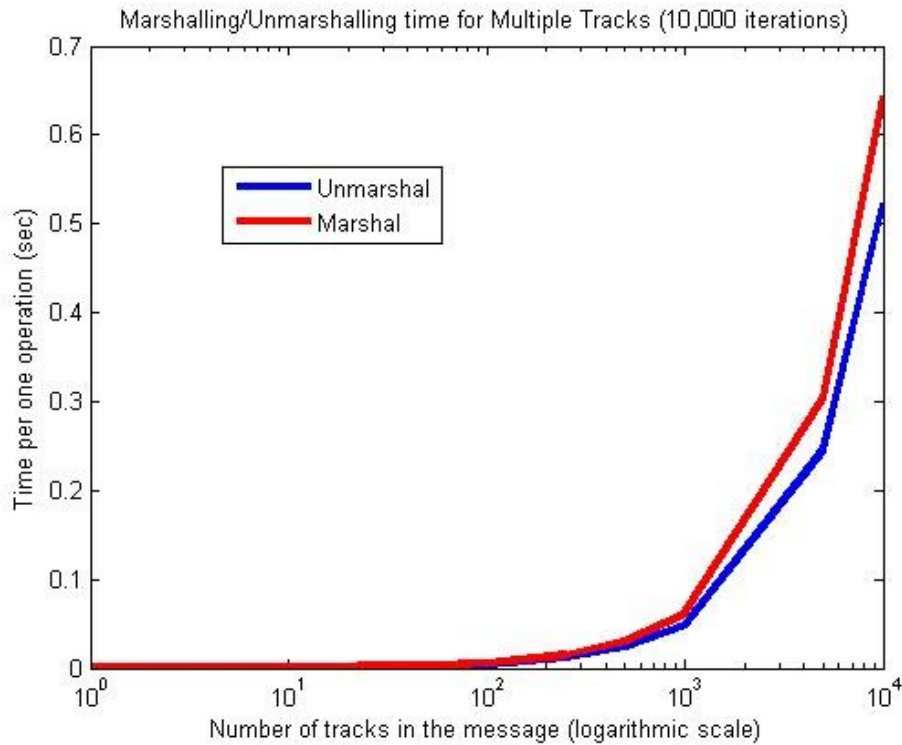
**Problem: Can we unmarshal the Java object into XML string quickly enough?**

# JAXB Conversion and Insertion Rates



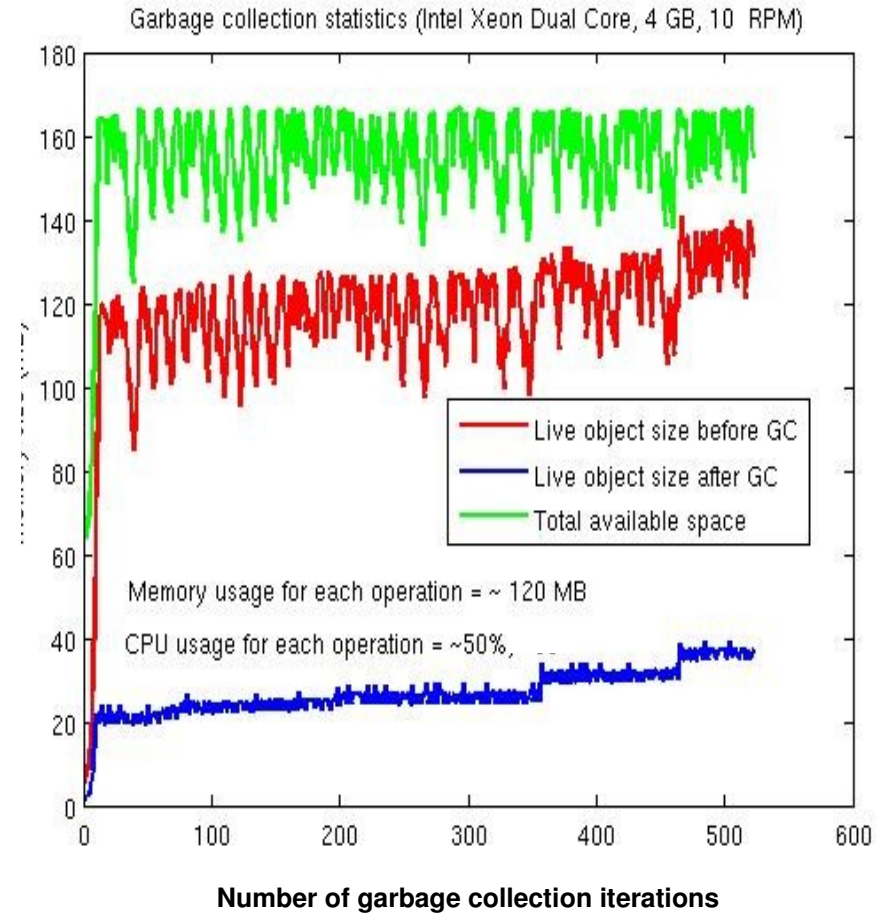
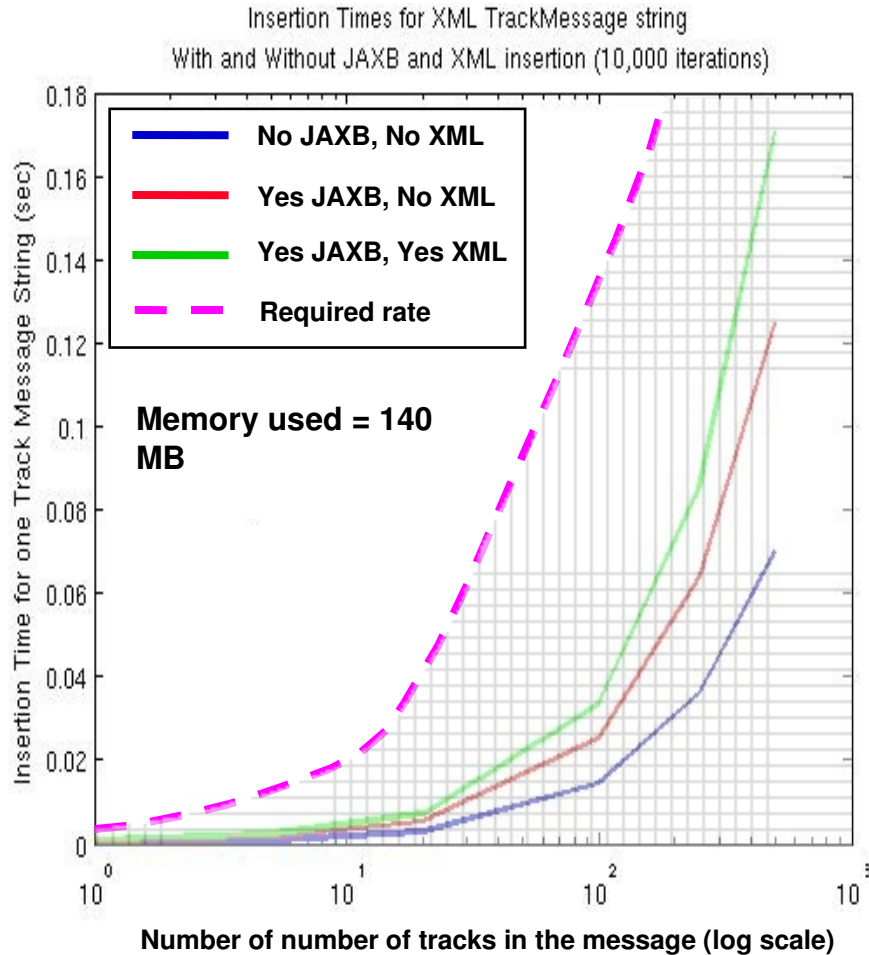
**After 100,000 conversions JAXB conversion time drops almost 1000 times!**

# JAXB Conversion Time Measurements



**Even in the case of 10,000 tracks in the message  
JAXB conversion takes less than a second**


# JAXB Conversion Time Measurements (cont.)



**Even in the worst case the insertion speed keeps up with real-time requirements**

# Outline

---

- Introduction
- Hardware Selection
- Database Server Selection
- Schema Design
-  • **Optimizing Search Operations**
  - Backtracking Operation
  - Search Data Flow
  - Real-time Data Indexation
  - Composite Indexes
- Gauging Concurrent Insert and Search Operations
- Summary

# Backtracking Operation

## How to find who launched the target?

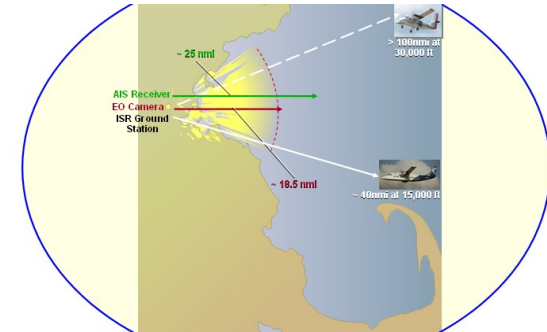
### First request type:

#### Specified:

- Object GlobalID
- Search Time Window

#### Returns:

- List of tracks with specified GlobalID
- within specified Time Window and default region boundaries



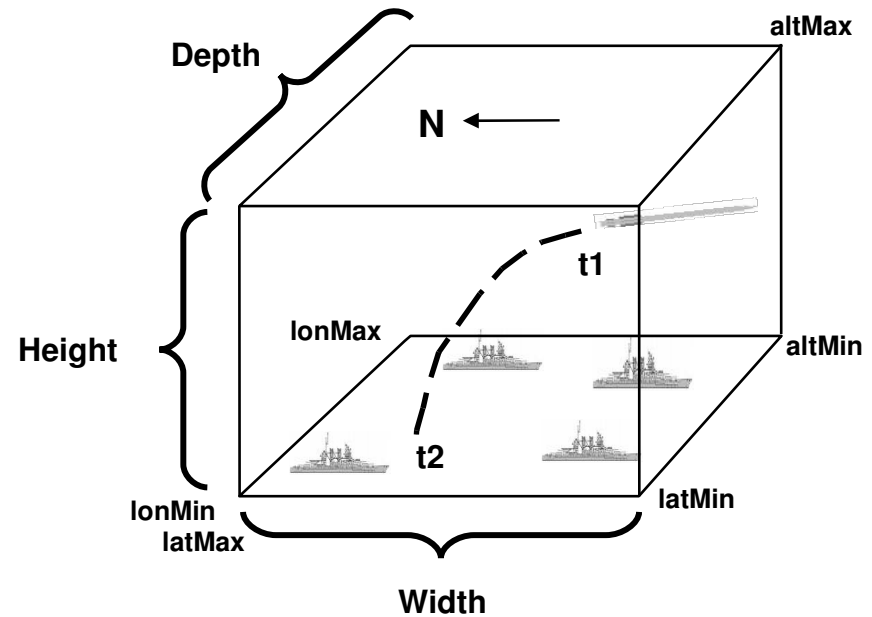
### Second request type:

#### Specified:

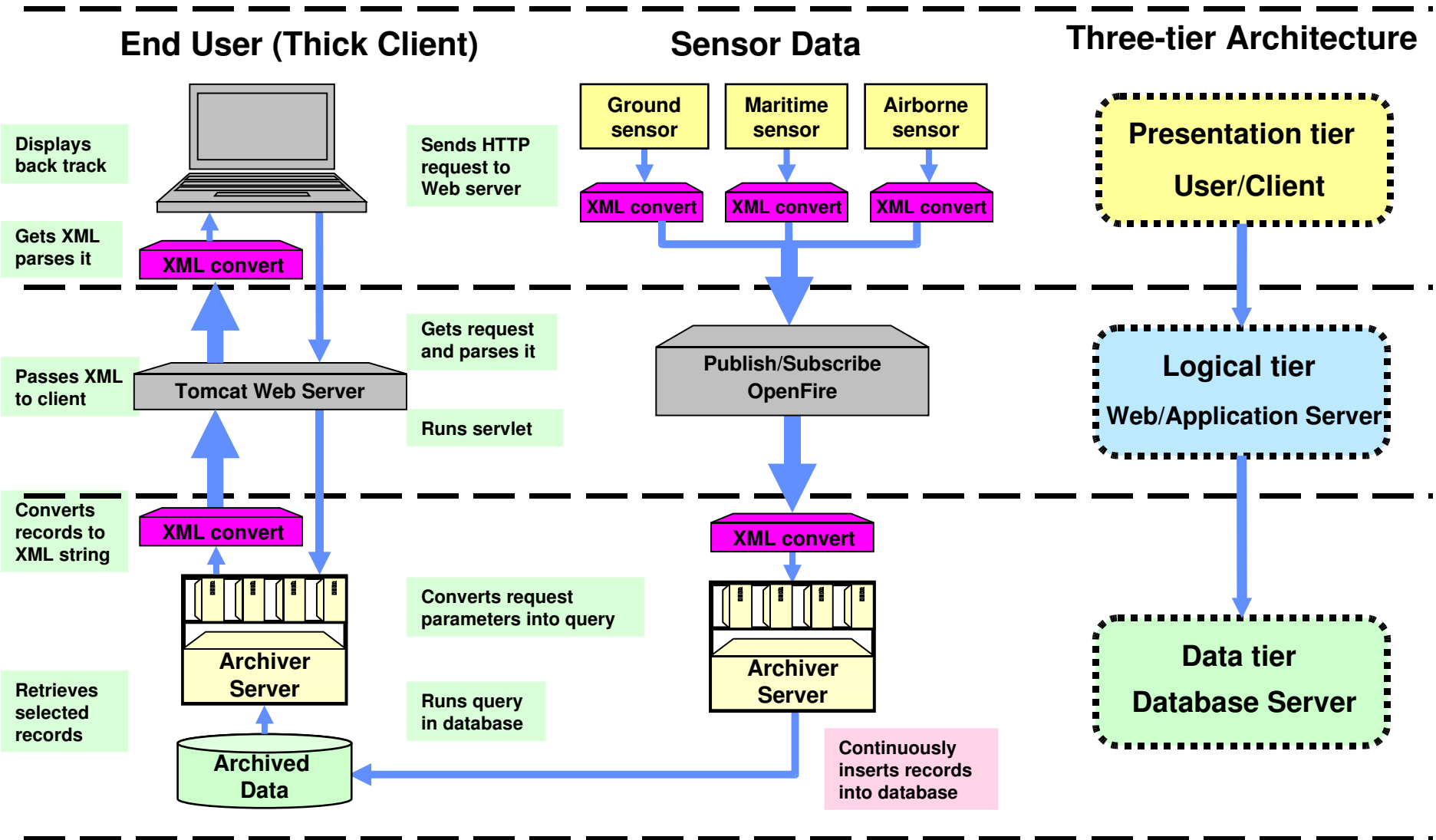
- Area boundaries
- Search Time

#### Returns:

- List of tracks containing the GlobalIDs within specified area at the specified time

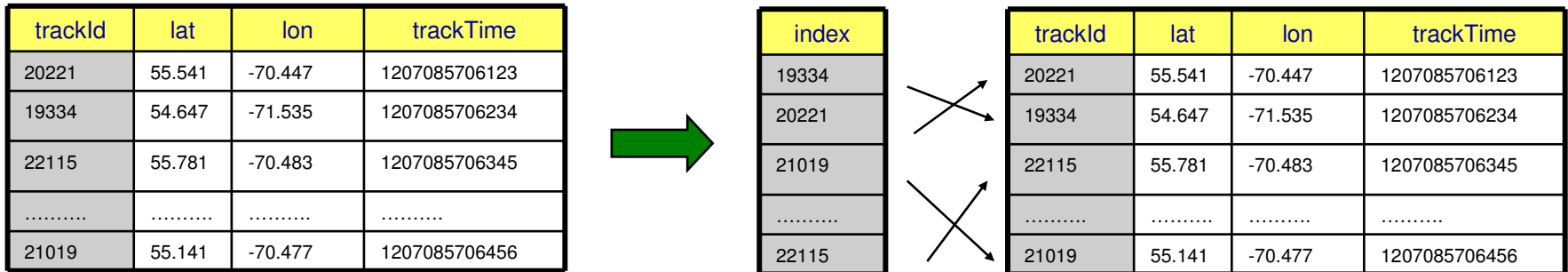


# Search Data Flow



# Real-time Data Indexation

Indexes help to find the matching rows very quickly by pre-sorting



## Traditional (batch) approach to indexing

1. Store data into the relational table
2. Add indexes to the table once the data accumulation is finished
3. Retrieve the data using added indexes

## Real-time (continuous) approach

- Proceed with data insertion into the relational table
- Allow indexes to be updated continuously
- Retrieve the data using continuously updated indexes

# Composite Indexes

**FIND** row **WHERE** trackId = 19334, trackTime = 1207085706144

**FIND** row **WHERE** lat = 55.559, lon = -70.478, alt = 15234 trackTime = 1207085706144

1. Different request types require different queries
2. Multiple query parameters require composite index
3. Due to leftmost index column rule each query needs its own composite index


Composite (multi-column) index			
19334	55.541	-70.447	1207085706123
19334	55.559	-70.345	1207085706224
19334	55.559	-70.478	1207085706012
19334	55.559	-70.478	1207085706144
22115	55.141	-70.477	1207085706456

2. Optimize search for multiple columns simultaneously
3. Stored as B-trees and use leftmost prefix rule
4. Have to be constantly updated due to the insertion
5. Have to be stored on the disk
6. Indexed columns are trackGlobalId, lat, lon, alt, trackType, trackTime

**Problem: How much real-time index updates slow down real-time insertions?**

# Outline

---

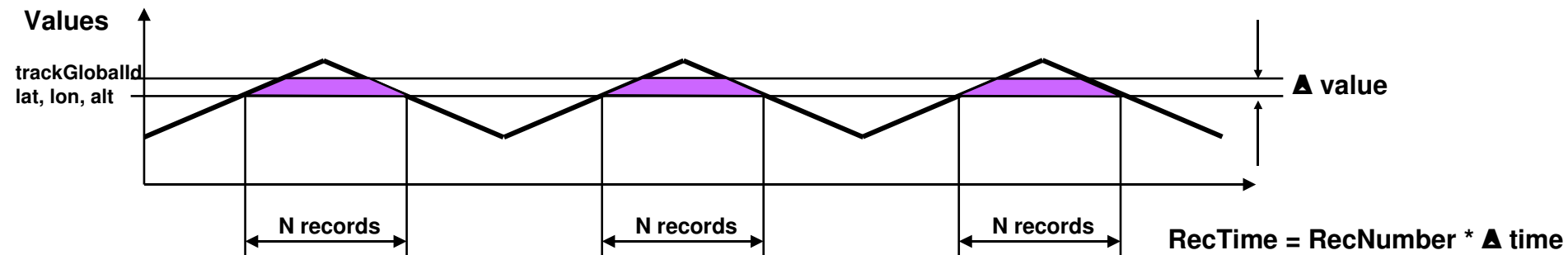
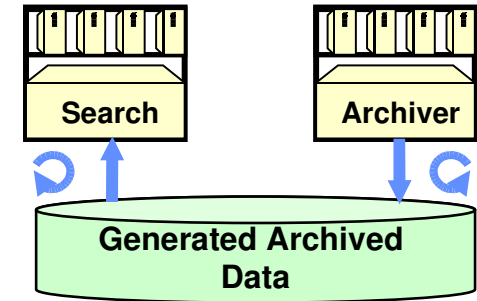
- Introduction
- Hardware Selection
- Database Server Selection
- Schema Design
- Optimizing Search Operations
-  • Gauging Concurrent Insert and Search Operations
  - Insertion/Search Operations Benchmark
  - Indexed vs Non-indexed Searches
  - Measuring Insertion Slowdown
- Summary

# Insertion/Search Operations Benchmark

## Generating representative insertion data

### Data properties for benchmarking:

- Outcome known in advance (natural data are not suitable)
- Selected values should be distributed all over the data set
- Six search parameters: trackGlobalId, lat, lon, alt, trackType, trackTime
- Values of selected parameters are not important

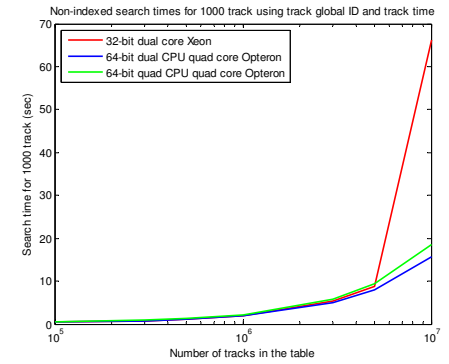
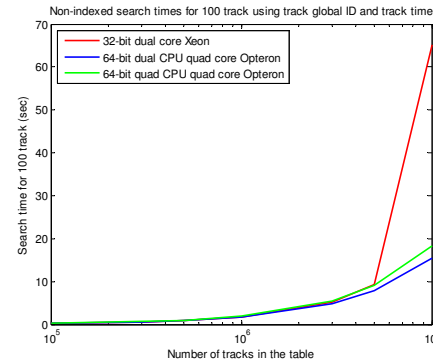
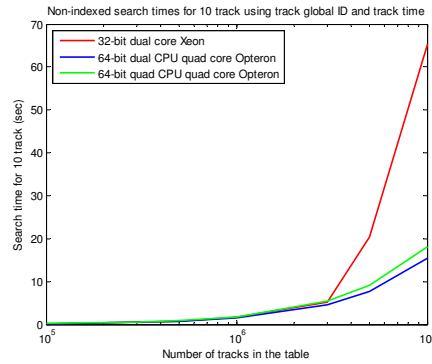
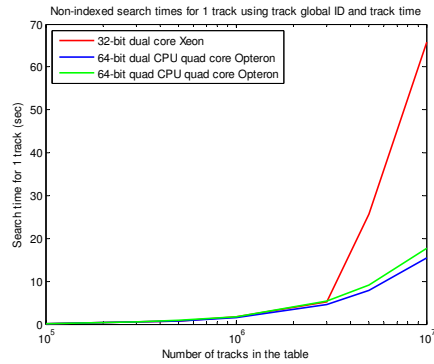


### Search queries and their indexed parameters

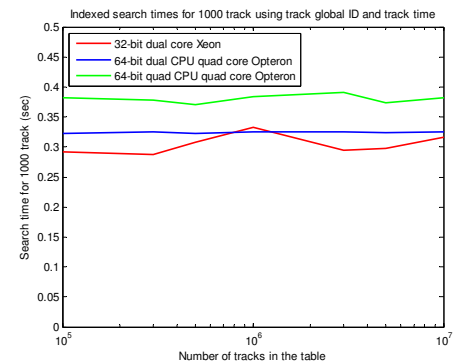
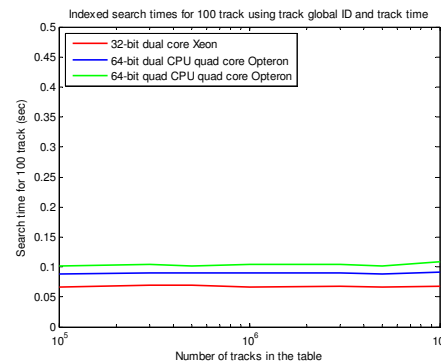
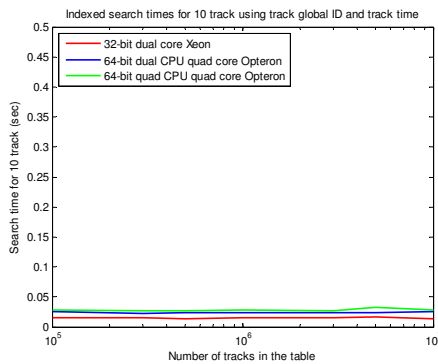
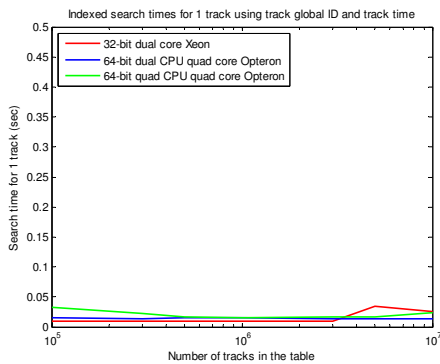
Content of the request	trackGlobalId (VARCHAR)	trackTime (BIGINT)	trackType	lat	lon	alt
Search for all tracks (contacts) in the specified area	—	X	X	X	X	—
Search for all contacts with specified ID	X	X	X	—	—	—
Search for all contacts near specified position	—	X	X	X	X	X

# Indexed vs Non-indexed Searches

## Non-indexed search

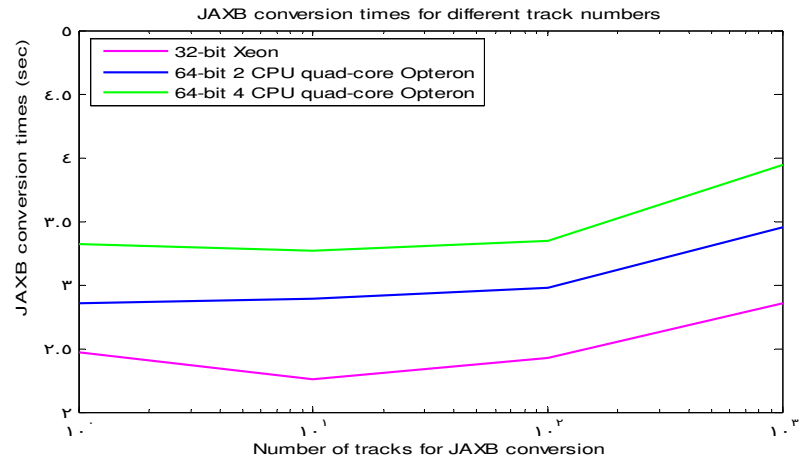
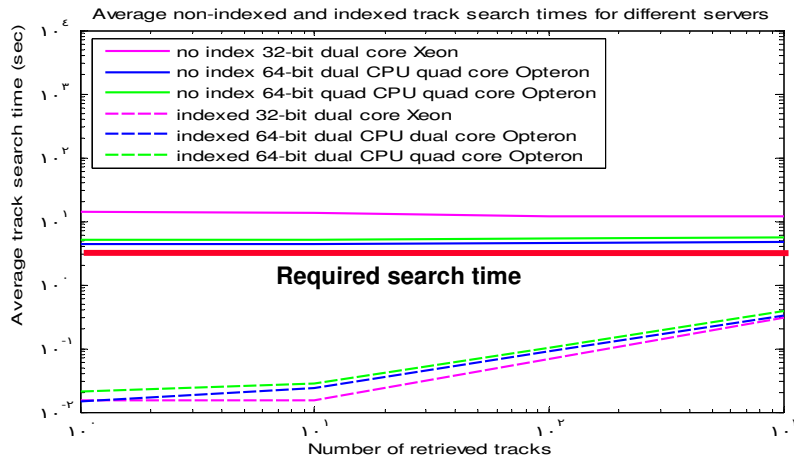
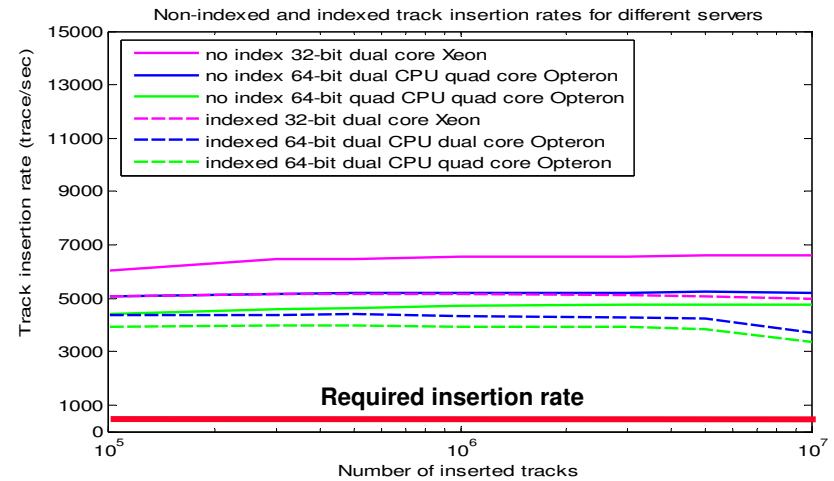
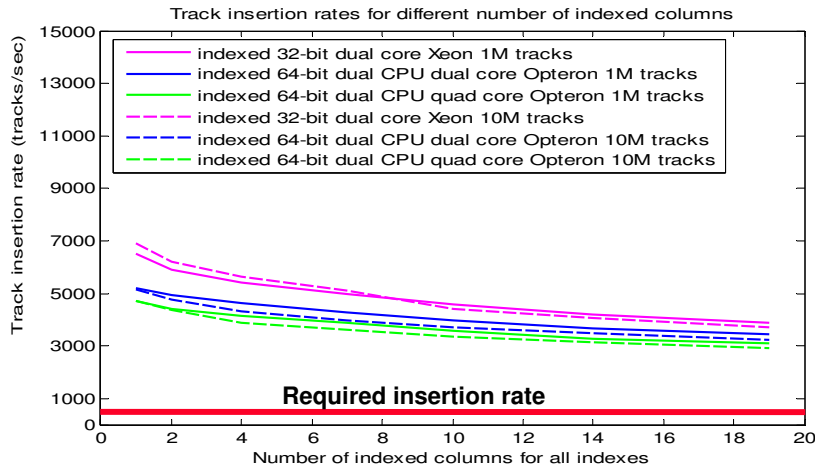


## Indexed search



**Indexes increase search performance by orders of magnitude!**

# Measuring Insertion Slowdown



**Indexes do slow down insertion, but the insertion rate remains very high**

# Outline

---

- **Introduction**
- **Hardware Selection**
- **Database Server Selection**
- **Schema Design**
- **Optimizing Search Operations**
- **Gauging Concurrent Insert and Search Operations**
- **Summary**



# Summary

- Benchmarked and selected the hardware for Archive and Search Operations
- Demonstrated that Opteron-based systems deliver better I/O than Xeon-based with comparable CPU power
- Compared different database servers and selected MySQL database server based on insertion rate and throughput
- Designed different database schemata and selected the simplest one with sufficient insertion performance
- Investigated search operation performance with constant flow of Insert queries and demonstrated its unsatisfactory performance without indexes
- Designed search query indexes and measured the data retrieval acceleration
- Demonstrated that the longer insertion times due to the indexation are still sufficient for successful archive operations