LazyBase: Trading freshness and performance in a scalable database (EuroSys 2012)

Jim Cipar, Greg Ganger,

*Kimberly Keeton, *Craig A. N. Soules,

*Brad Morrey, *Alistair Veitch

PARALLEL DATA LABORATORY

Carnegie Mellon University

* HP Labs

Carnegie Mellon
Parallel Data Laboratory

(very) High-level overview

LazyBase is…

- Distributed data base
- High-throughput, rapidly changing data sets
- Efficient queries over consistent snapshots
- Tradeoff between query latency and freshness

Query freshness

- There is delay when loading data into database
  - Data may not be visible to read() immediately
  - In some cases delay could be minutes or hours

- **Freshness** is an indication of this delay
  - "These results contain all data as of 15 seconds ago"

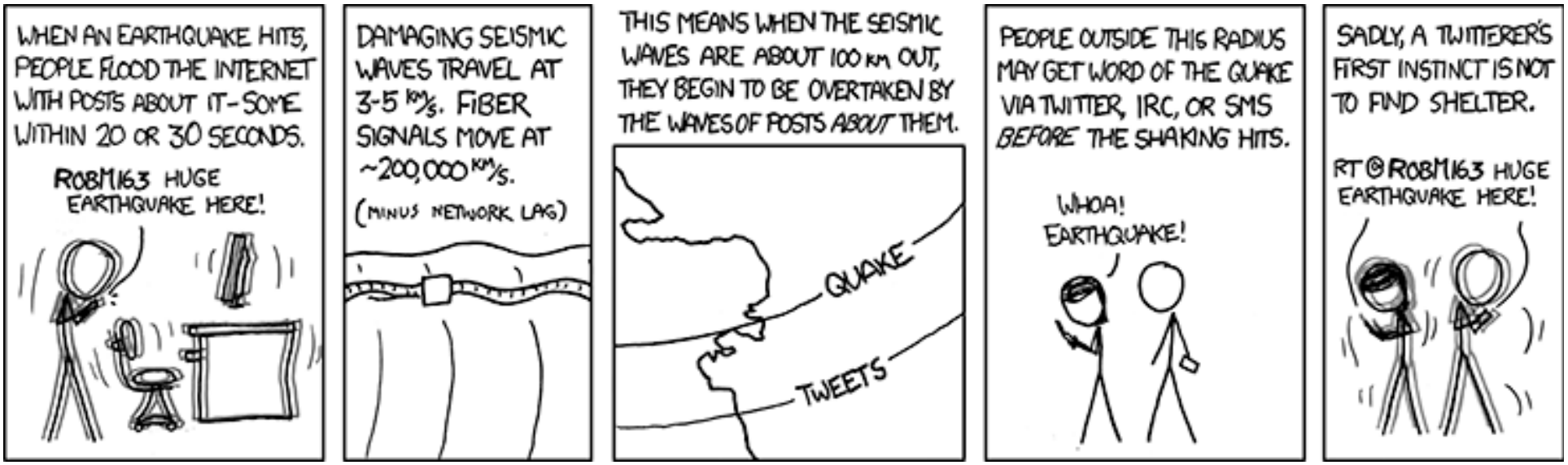
Query latency

Time between issuing a query,
and receiving results

- Can range from milliseconds to hours depending on complexity of query and amount of data involved

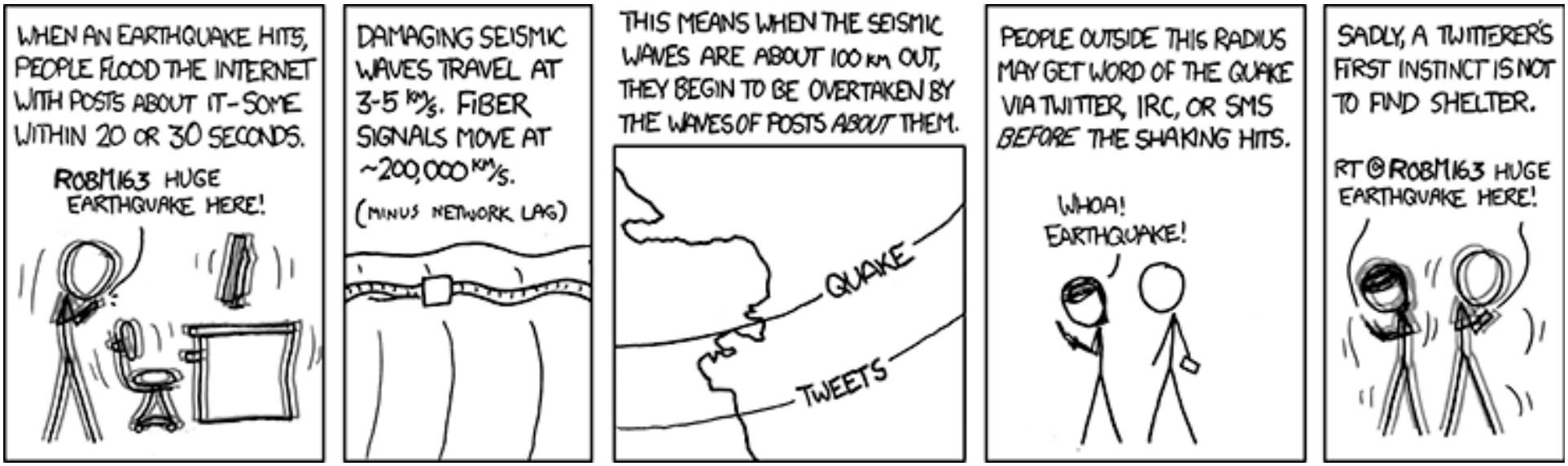
- LazyBase allows programmers to choose a tradeoff between query freshness and latency

Example application



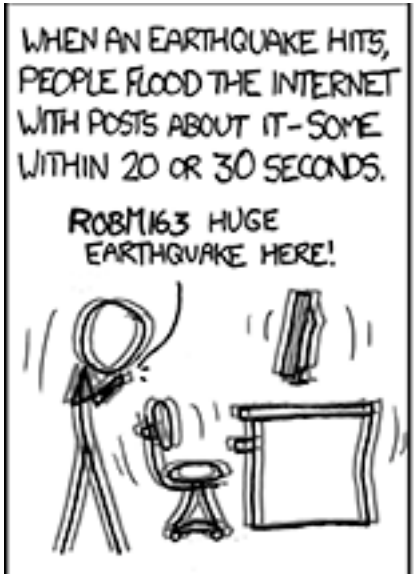
Example application

- High bandwidth stream of Tweets
  - Many thousands per second
  - 200 million per day



Example application

- High bandwidth stream of Tweets
  - Many thousands per second
  - 200 million per day



- Queries accept different freshness levels
  - Freshest: USGS Twitter Earthquake Detector
  - Fresh: Hot news in last 10 minutes
  - Stale: social network graph analysis
- Consistency is important
  - Tweets can refer to earlier tweets
  - Some apps look for cause-effect relationships

Class of analytical applications

- **Performance**
  - Continuous high-throughput updates
  - "Big" analytical queries (scan large parts of data set)
- **Freshness**
  - Varying freshness requirements for queries
  - Freshness varies by query, not data set
- **Consistency**
  - There are many types, more on this later...

Applications and freshness

Freshness / Domain	Seconds	Minutes	Hours+
Retail	Real-time coupons, targeted ads	Just-in-time inventory	Product search, earnings reports
Enterprise information management	Infected machine identification	File-based policy validation	E-discovery requests, search
Transportation	Emergency response	Real-time traffic maps	Traffic engineering, route planning

Current solutions

- Online transaction processing databases (OLTP)
- Data warehouse systems
- “NoSQL” databases

Online transaction processing

- Typical databases supporting SQL language
- Focus on consistency and reliability
- Typically used for short transactions
  - Highly selective query with index or small inserts

✔ Support continuous ingest and querying

✖ Performance often not sufficient for data mining
  - [Chaudhuri ‘97], [Plattner ‘09]

Data warehouse systems

- Sometimes called Online Analytical Processing
- Used for analyzing large data sets
  - Efficient scans and aggregations

✔ Designed for data mining: fast efficient queries

✖ Data loaded by batch jobs:
  - Extract-transform-load pipeline (ETL)
  - At any given time data may be many hours old

NoSQL databases

- Many diverse architectures
  - Cassandra, HBase, Hypertable, MongoDB …
- Designed for scalability on large clusters

✔ Scalable ingest and query

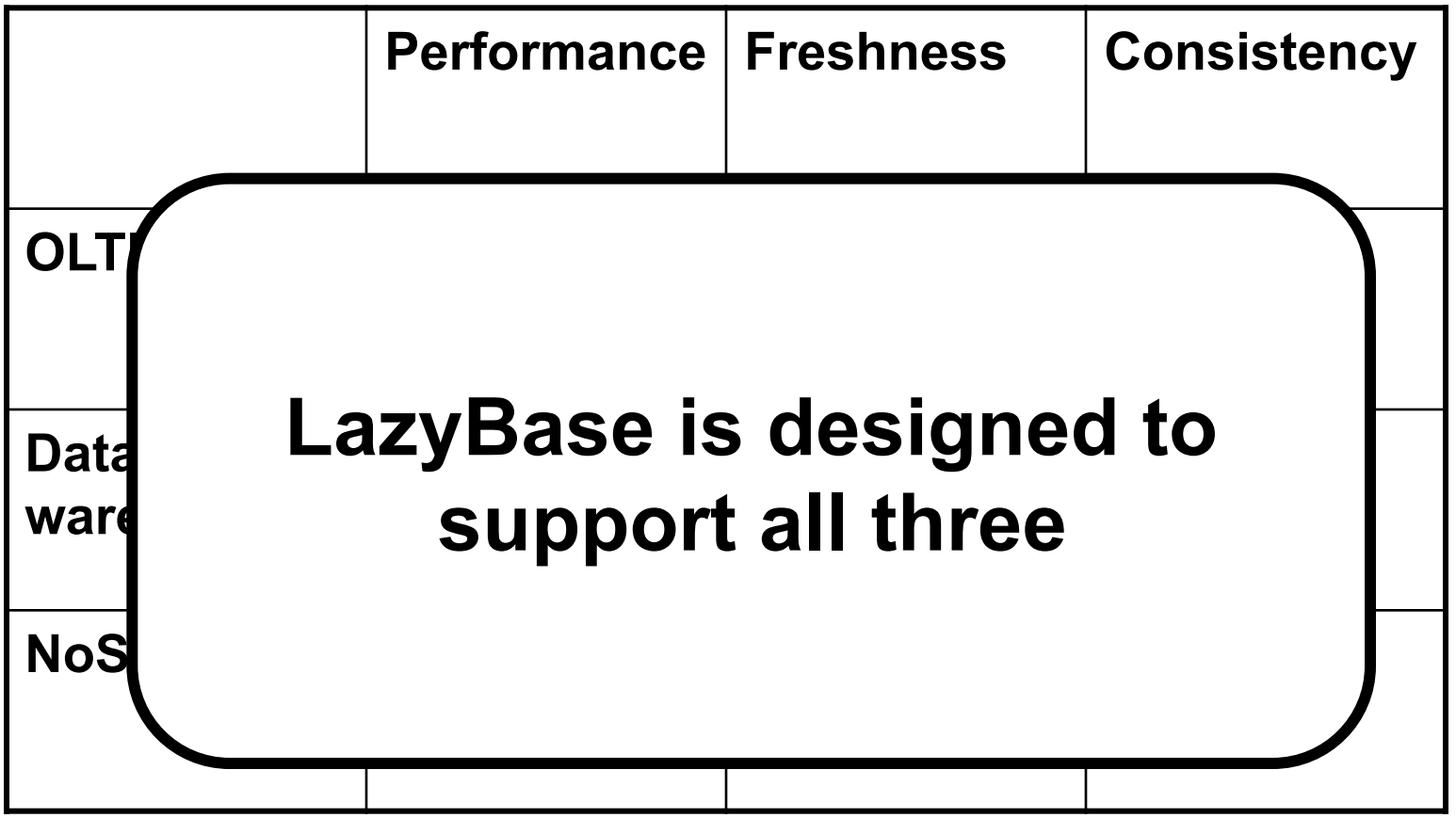
✖ Sacrifice consistency for scalability

Cassandra

Current Solutions

	Performance	Freshness	Consistency
OLTP/ SQL DBs	✗	✓	✓
Data warehouse	✓	✗	✓
NoSQL	✓	✓	✗

Current Solutions



Key ideas

- **Separate concepts of consistency and freshness**
  - Batching to improve throughput, provide consistency
- **Trade latency for freshness**
  - Can choose on a per-query basis
    - Fast queries over stale data
    - Slower queries over fresh data

Consistency ≠ freshness

Separate concepts of consistency and freshness

- Query results **may be stale**: missing recent writes
- Query results **must be consistent**

Query (or read) transactions

- A query transaction is a group of queries
- Each query asks for some subset of the data
  - "All rows where the user name is 'jcipar'"
- Possibly aggregated in different ways
  - "The average size of tweets from user 'jcipar'"

Write transactions

- Group of inserts, updates and deletes
- Insert – “add a new row with these values”
- Update – “modify row with id X”
- Delete – “remove row with id X”

Snapshot isolation

- Commonly used in OLTP databases
- Simple case: read-only/write-only transactions
- Write transactions applied atomically to most recent version of database
- Read transactions act on single version in database
  - Every query in the transaction accesses same snapshot

Consistency in LazyBase

- Atomic **multi-row updates**
- **Monotonicity**:
  - If a query sees update A, all subsequent queries will see update A
- **Consistent prefix**:
  - Total ordering of updates
  - If a query sees update number X, it will also see updates 1…(X-1)

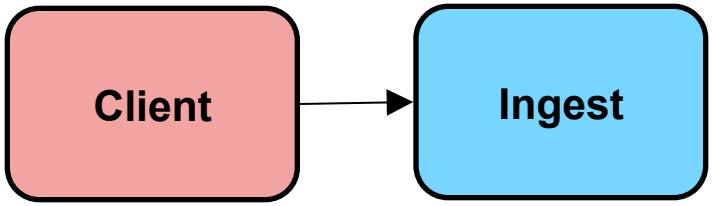
LazyBase limitations

- Only supports **observational data**
  - Transactions are read-only or write-only
  - No read-modify-write
  - Not online transaction processing
- Not (currently) targeting really huge scale
  - 10s of servers, not 1000s
  - Not everyone is a Google (or a Facebook…)

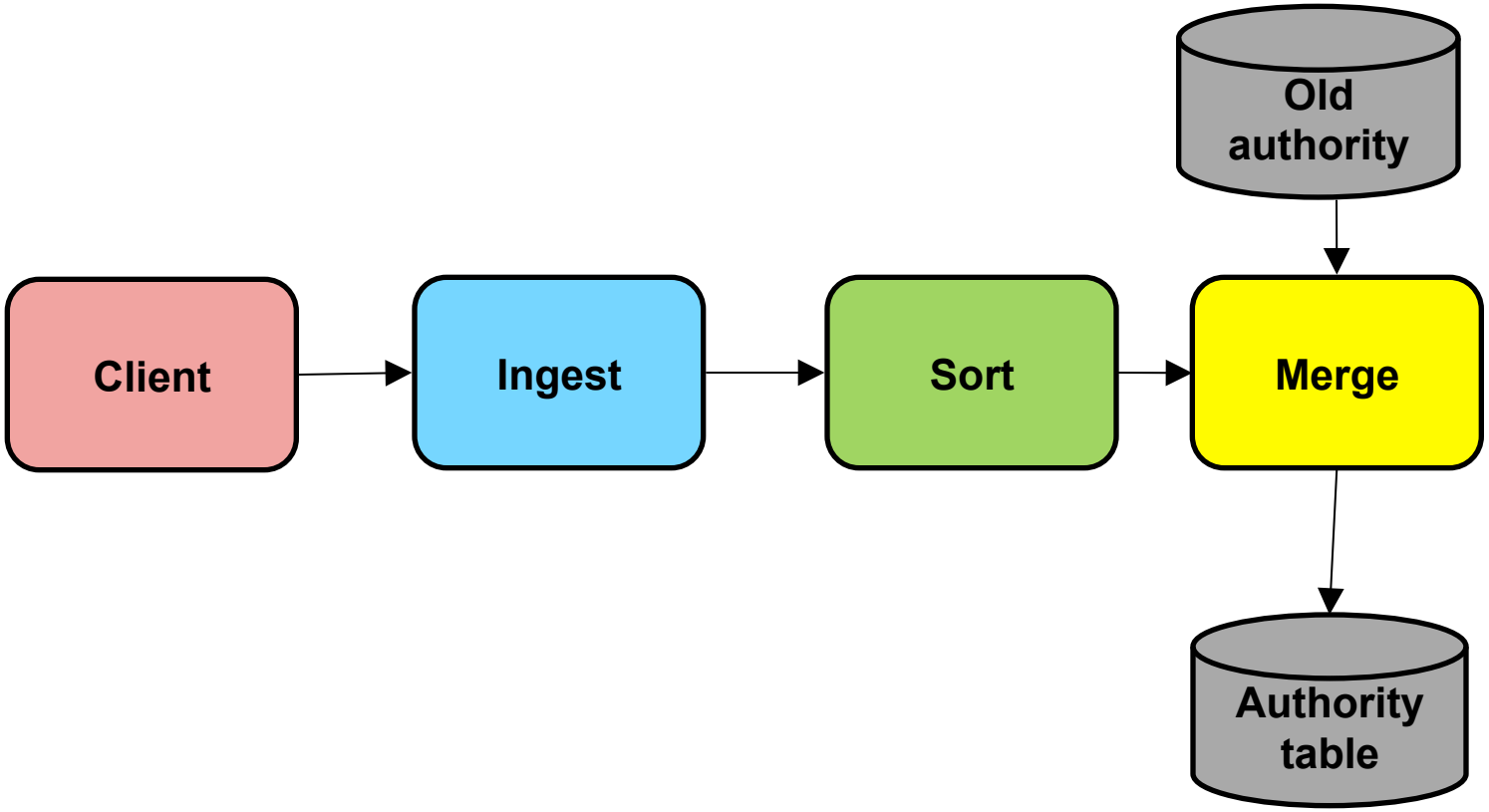
LazyBase design

- LazyBase is a distributed database
  - Commodity servers (e.g. 8 CPU cores, 16 GB RAM)
  - Can use direct attached storage
- Each server runs:
  - General purpose worker process
  - Ingest server that accepts client requests
  - Query agent that processes query requests
- Logically LazyBase is a pipeline
  - Each pipeline stage can be run on any worker
  - Single stage may be parallelized on multiple workers

Pipelined data flow



Pipelined data flow



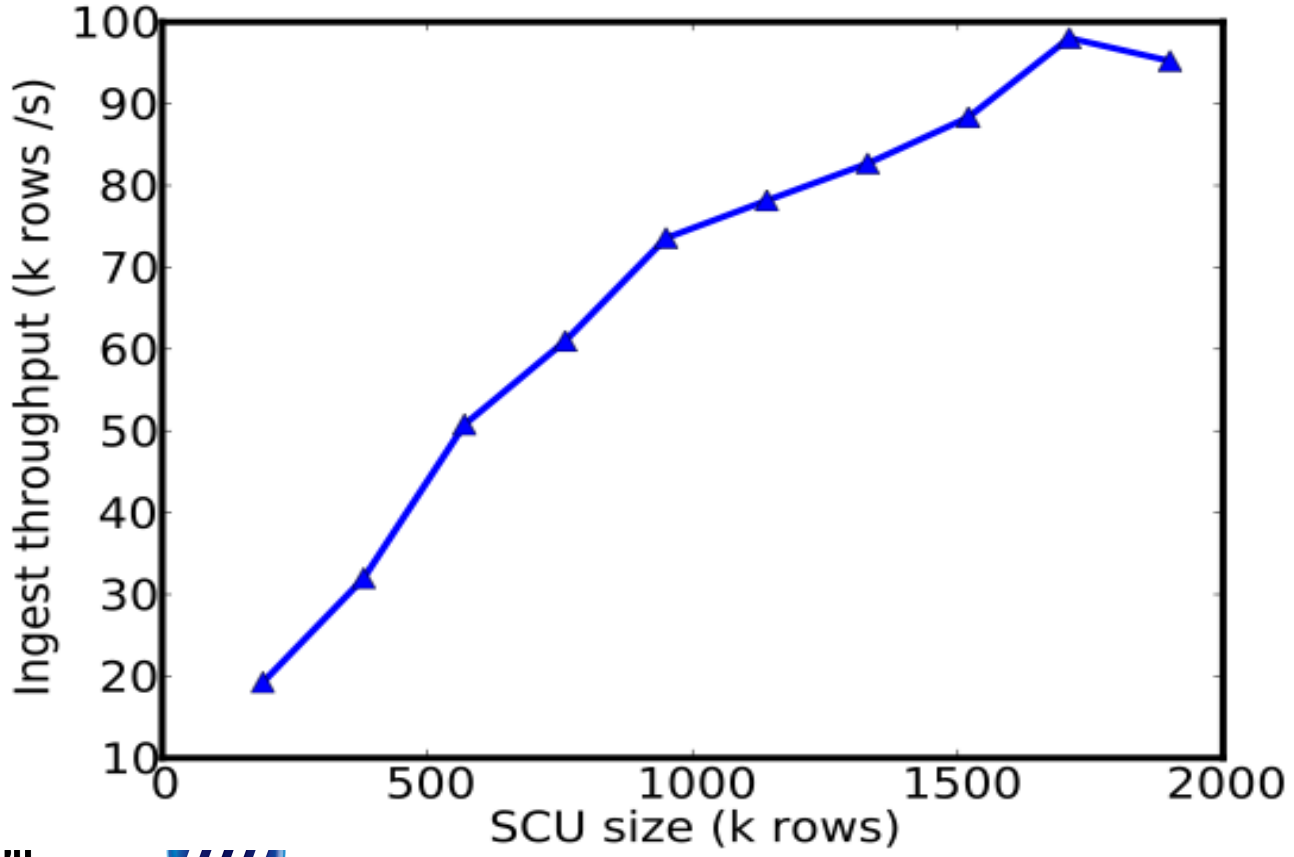
Batching updates

- **Batching for performance and atomicity**
- Common technique for throughput
  - E.g. bulk loading of data in data warehouse ETL

- Also provides basis for atomic multi-row operations
  - Batches are applied atomically and in-order
  - Called SCU (self-consistent update file)
  - SCUs contain inserts, updates, deletes

Batching for performance

Large batches of updates increase throughput



Problem with batching: latency

- Batching trades update latency for throughput
  - Large batches → database is very stale
  - Very large batches/busy system → could be hours old

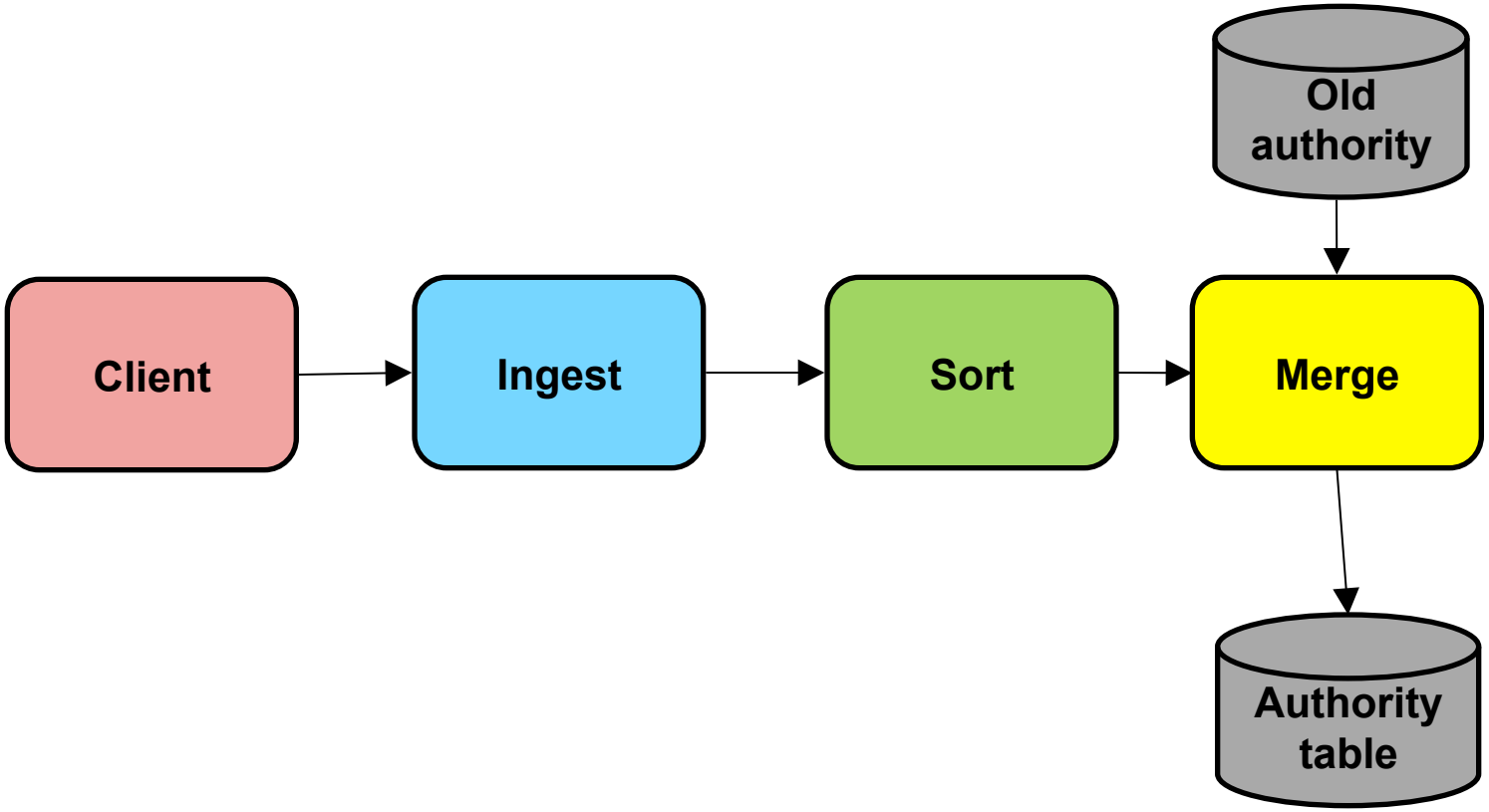
- OK for some queries, bad for others

Put the “lazy” in LazyBase

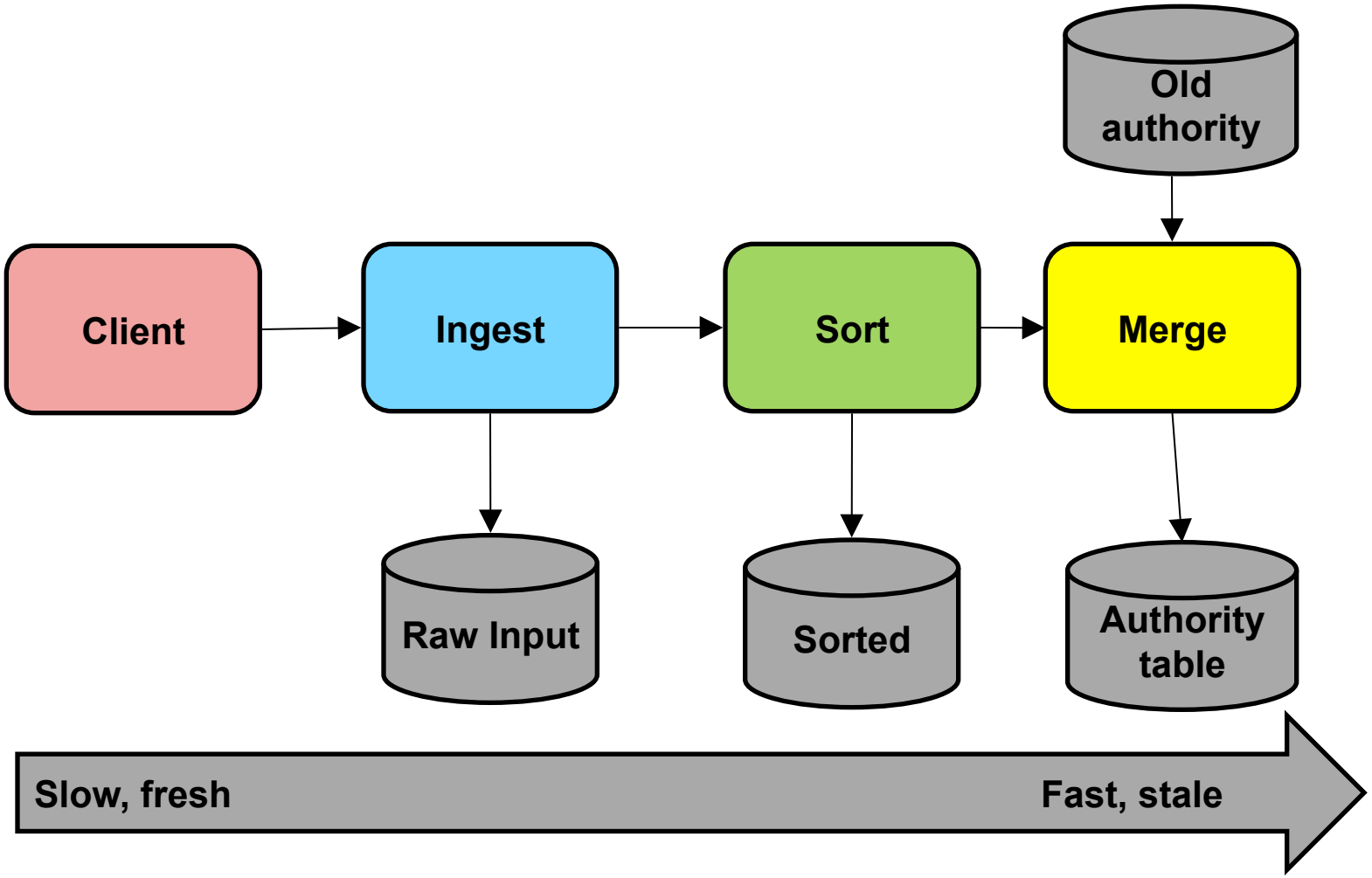
As updates are processed through pipeline, they become progressively “easier” to query.

We can use this to trade query latency for freshness.

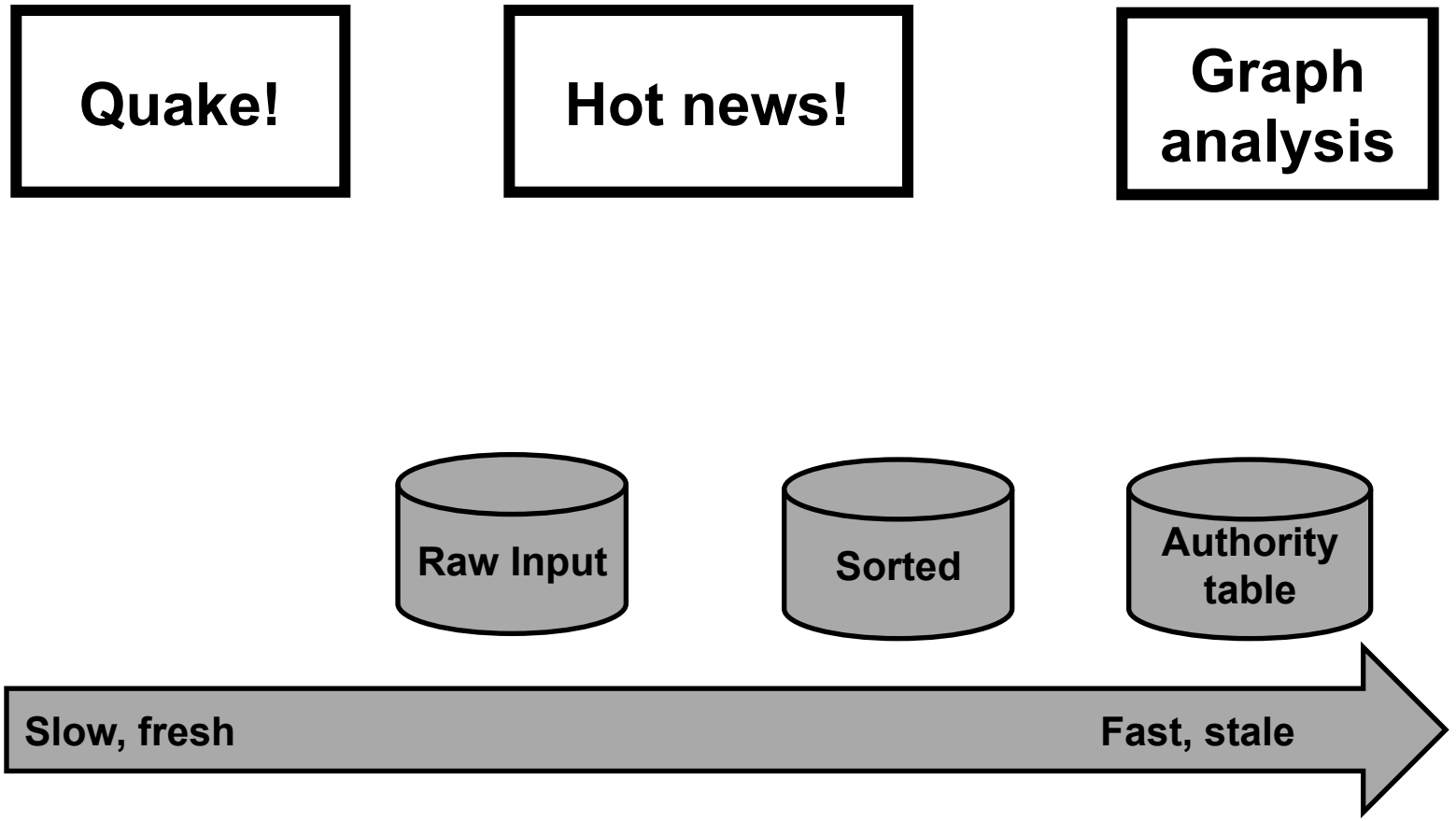
Query freshness



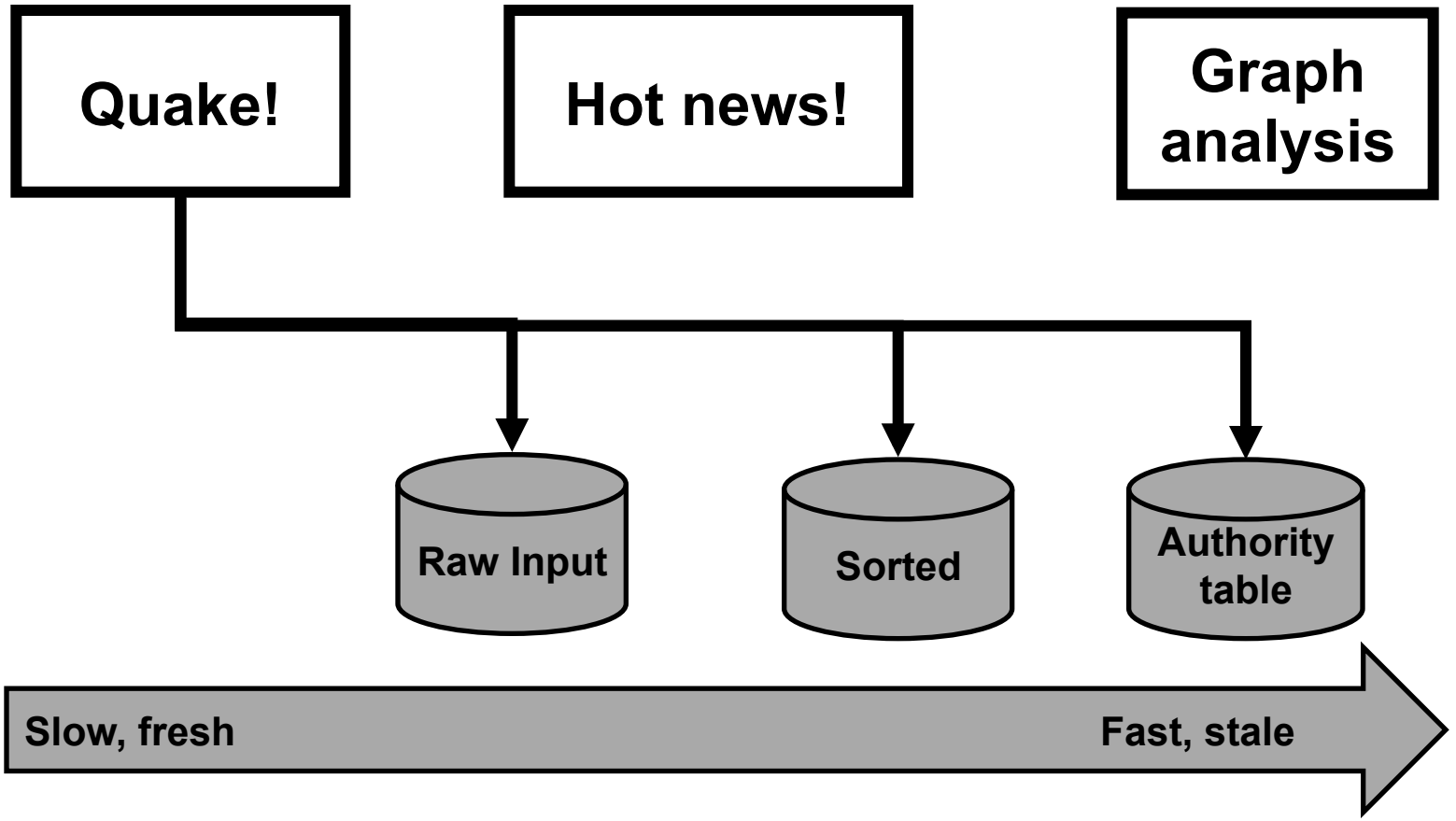
Query freshness



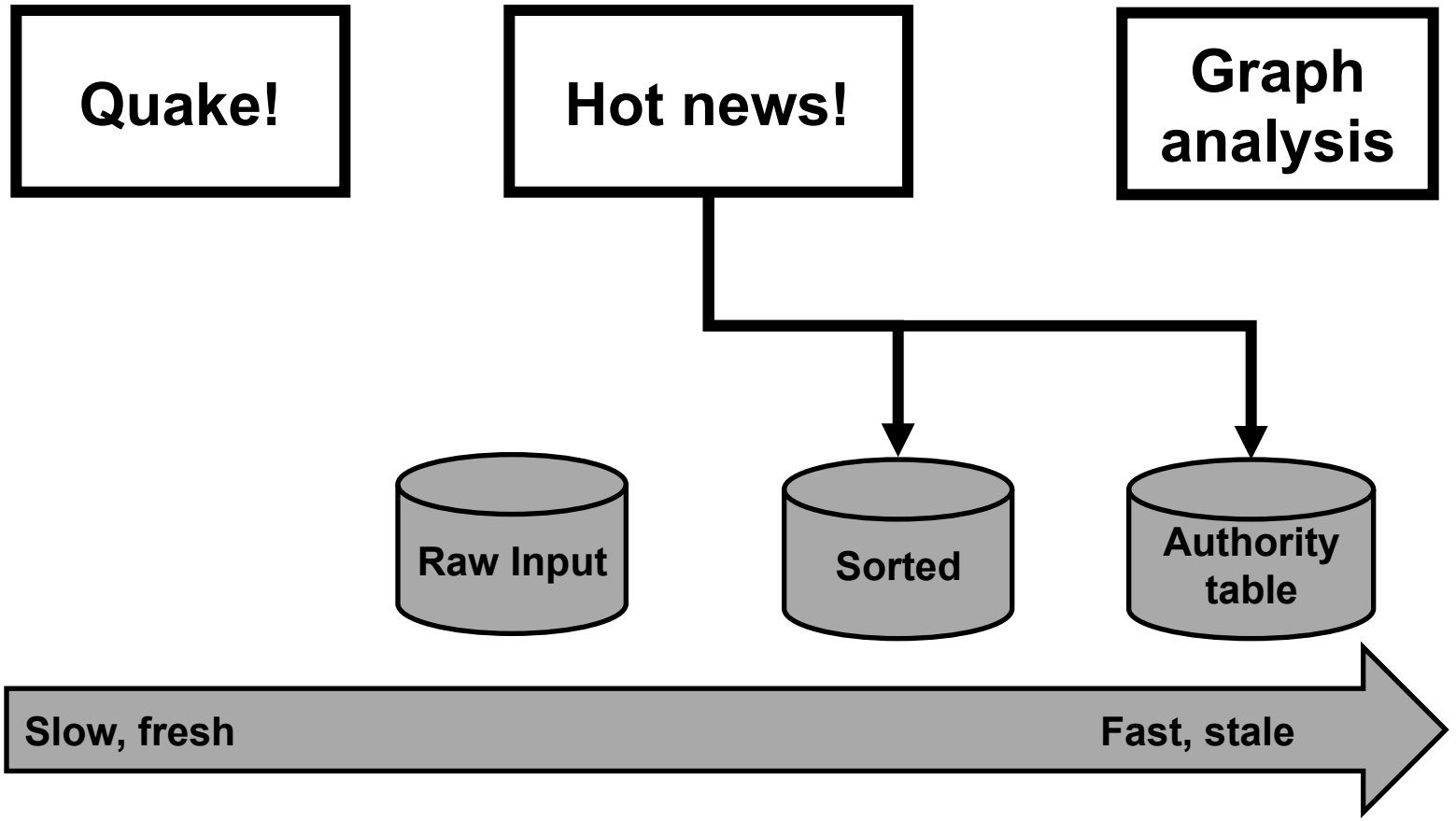
Query freshness



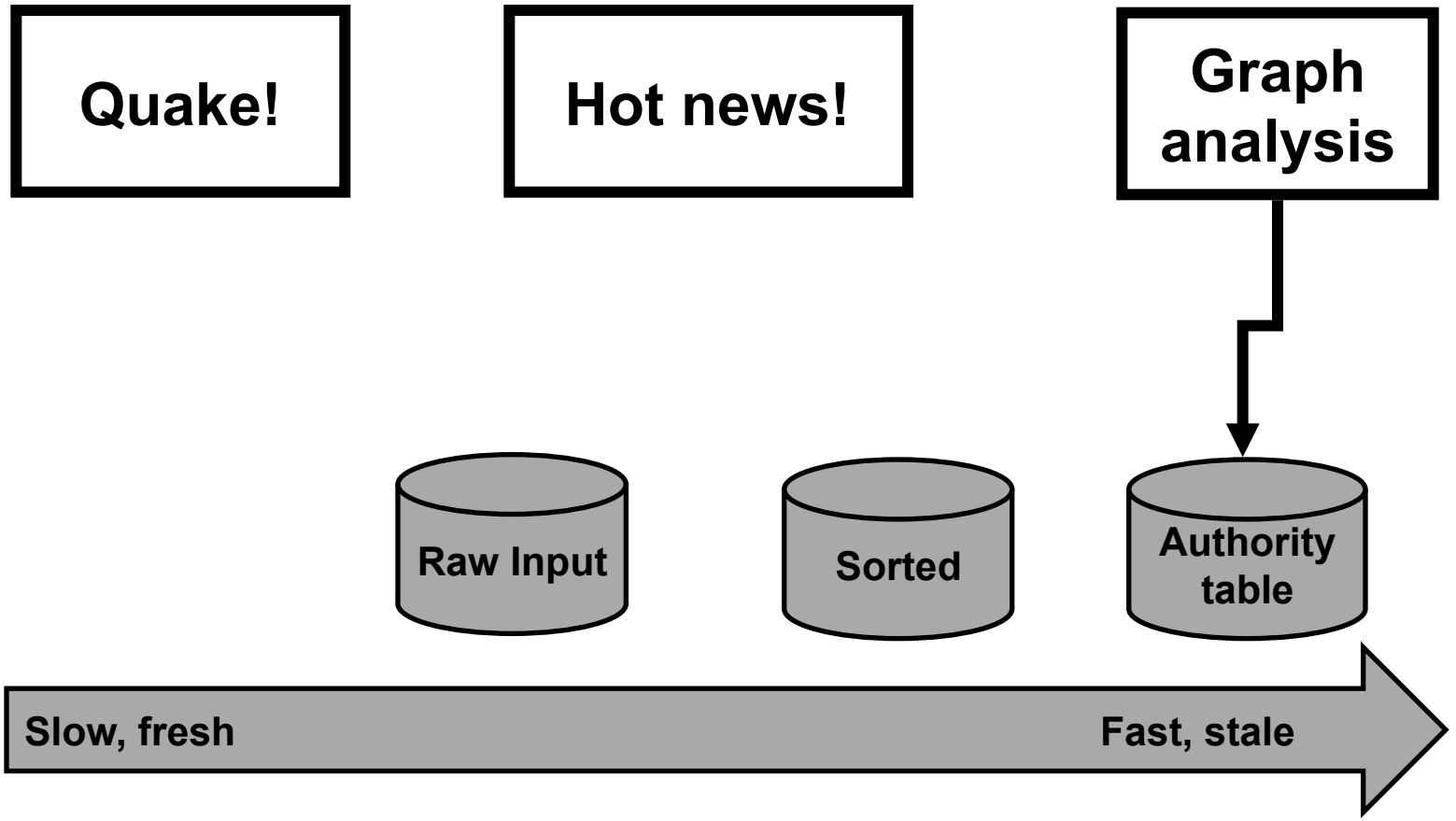
Query freshness



Query freshness



Query freshness



Query interface

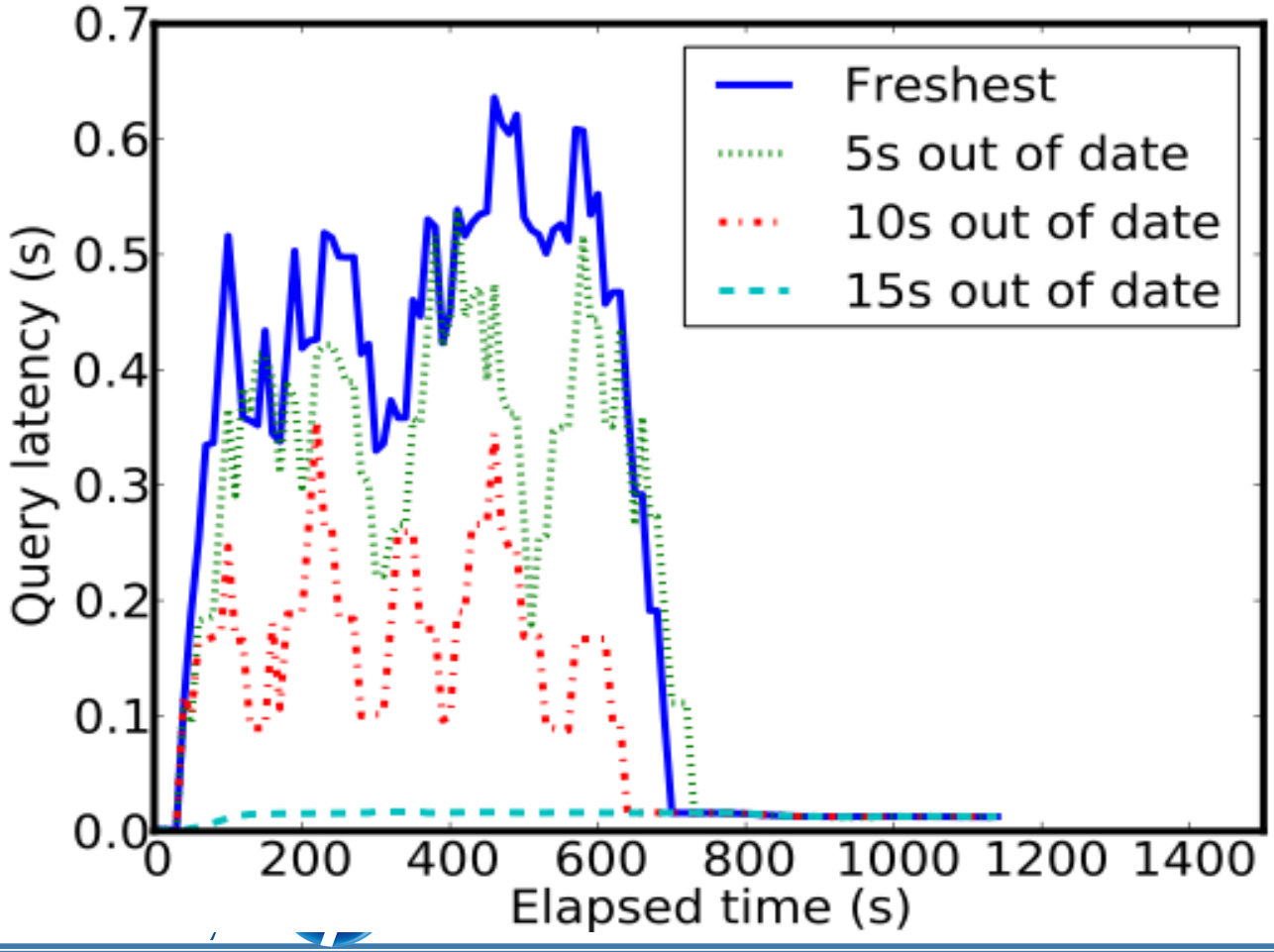
- User issues high-level queries
  - Programatically or like a limited subset of SQL
  - **Specifies freshness**

```
SELECT COUNT(*) FROM tweets
WHERE user = "jcipar"
FRESHNESS 30;
```

- Client library handles all the "dirty work"

Query latency/freshness

Queries allowing staler results return faster



Experiments to show…

- Importance of batching
- Freshness/performance tradeoff

Experiments to show…

- Importance of batching
- Freshness/performance tradeoff
- Throughput and scalability of updates
- Performance for queries
  - Both “small” and “big” queries
- Consistency relative to Cassandra
- Freshness relative to Cassandra

Experimental setup

- Ran on OpenCirrus cluster in DCO
  - 6 dedicated cores, 12GB RAM, per server
  - Local storage
- Data set was ~38 million tweets
  - 50 GB uncompressed
- Compared to Cassandra
  - Reputation as “write-optimized” NoSQL database

Experiments to show…

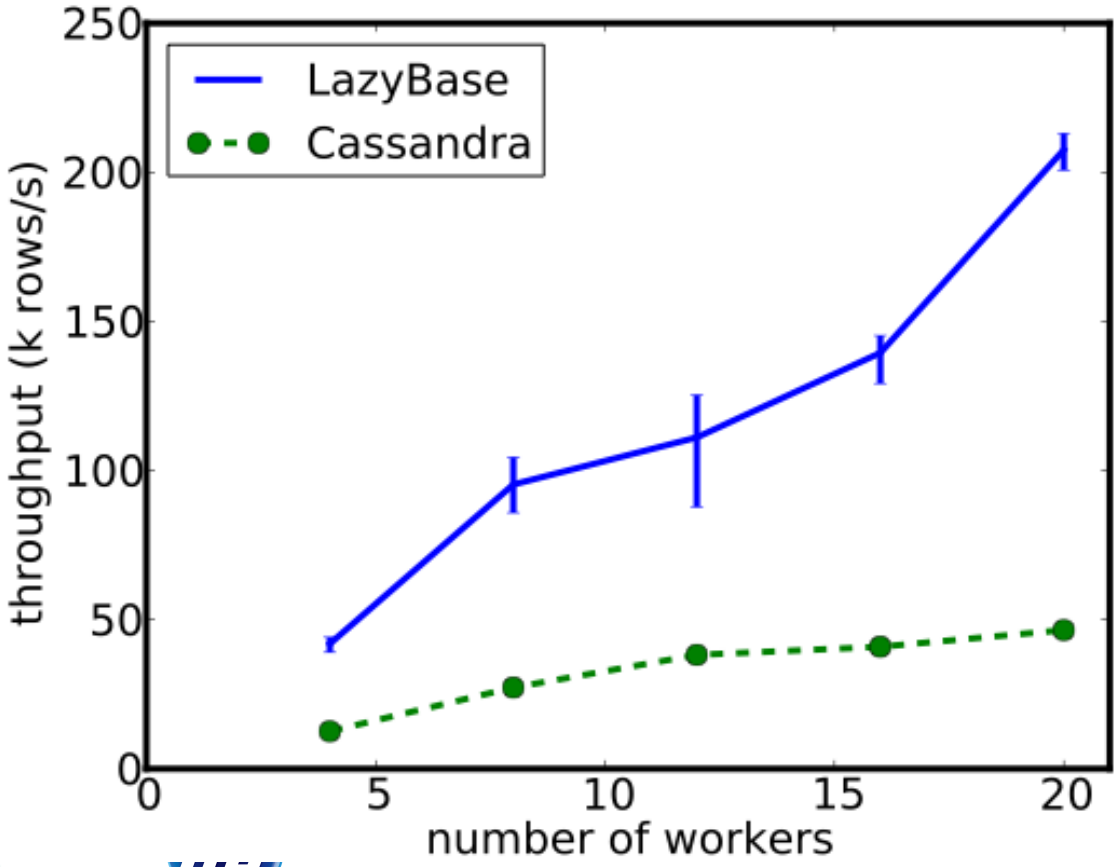
- Importance of batching
- Freshness/performance tradeoff
- **Throughput and scalability of ingest**
- Performance for queries
  - Both “small” and “big” queries
- Consistency relative to Cassandra
- Freshness relative to Cassandra

Ingest scalability experiment

- Measured time to ingest entire data set
- Uploaded in parallel from 20 servers
- Varied number of worker processes

Ingest scalability results

LazyBase scales effectively up to 20 servers
Efficiency is ~4x better than Cassandra



Experiments to show…

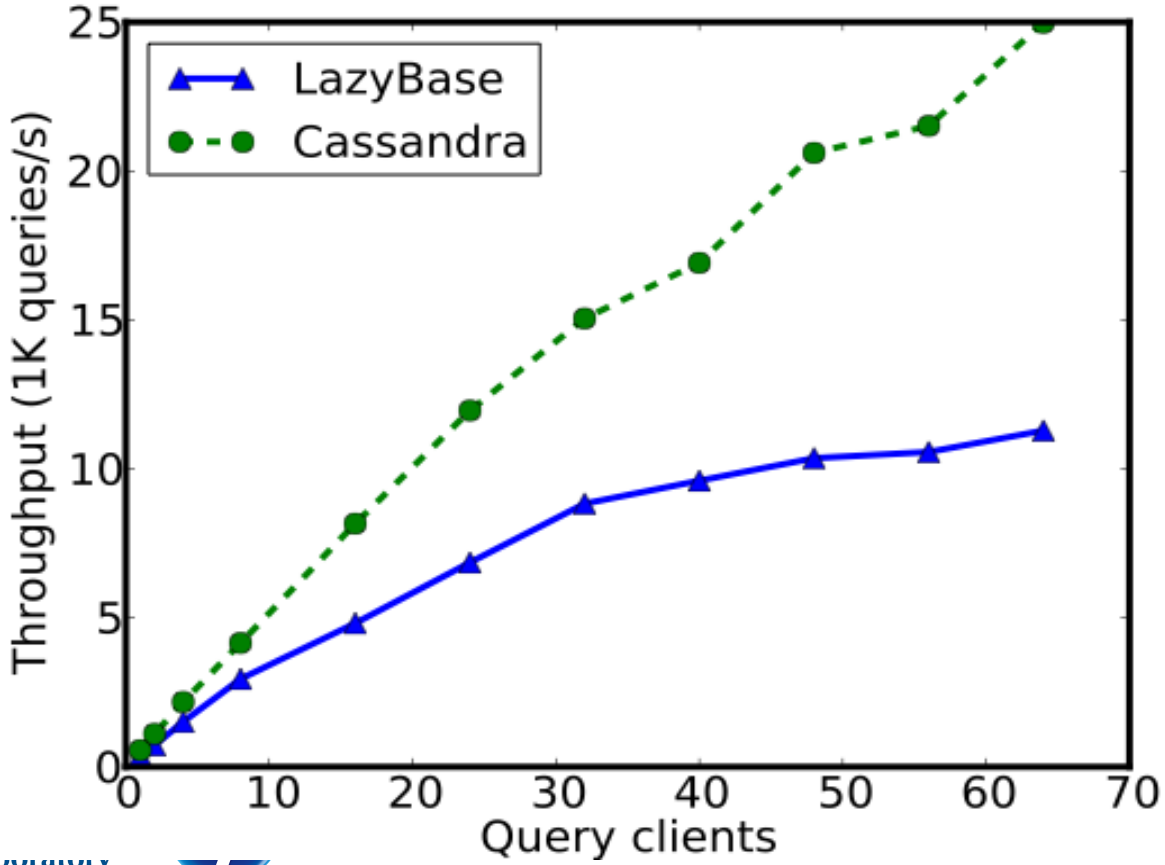
- Importance of batching
- Freshness/performance tradeoff
- Throughput and scalability of ingest
- **Performance for queries**
  - **Both "small" and "big" queries**
- Consistency relative to Cassandra
- Freshness relative to Cassandra

Query experiments

- Test performance of fastest queries
  - Access only authority table
- Two types of queries: point and range
  - Point queries get single tweet by ID
  - Range queries get list of valid tweet IDs in range
    - Range size chosen to return ~0.1% of all IDs
- Cassandra range queries used get_slice
  - Actual range queries discouraged

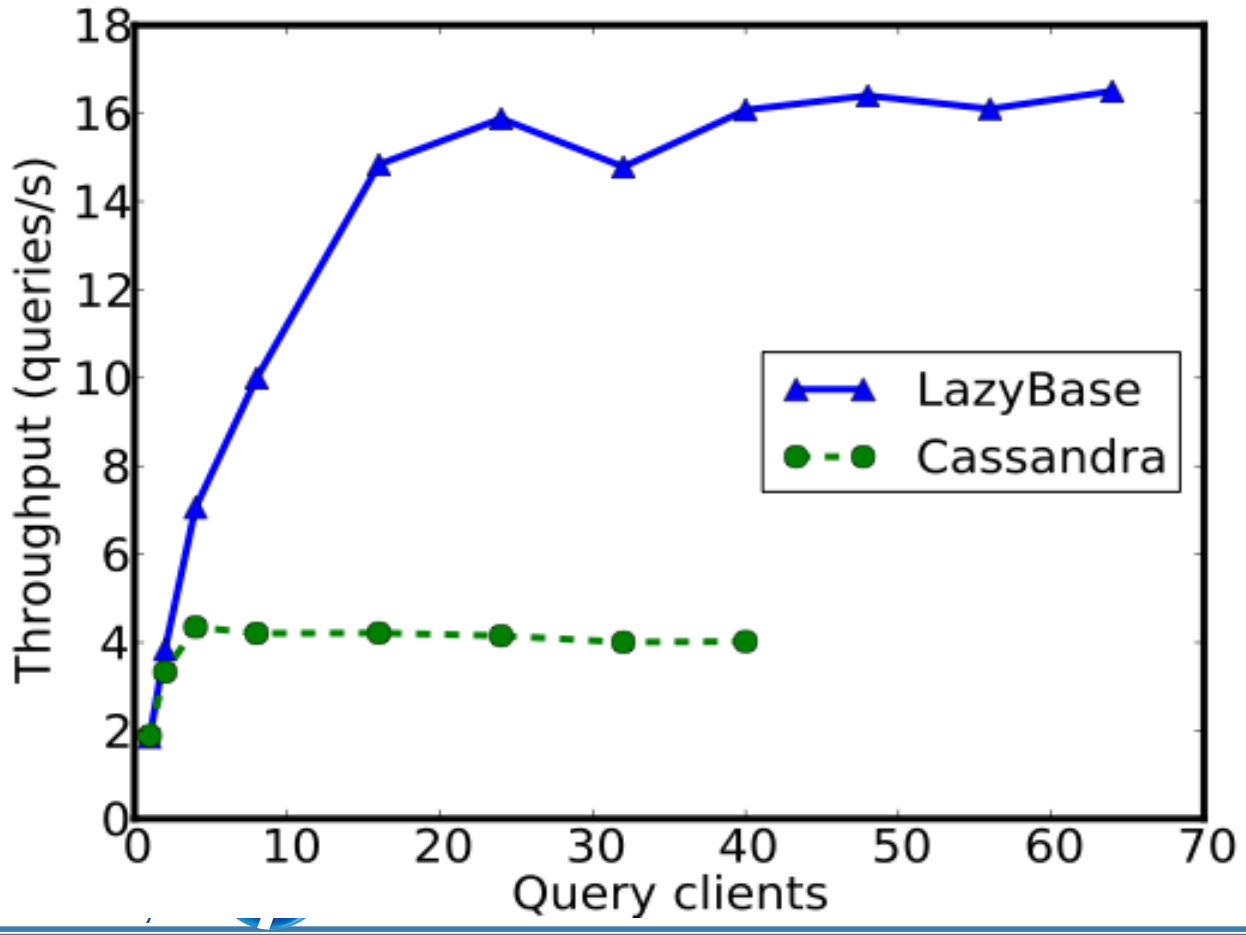
Point query throughput

Queries scale to multiple clients
Raw performance suffers due to on-disk format



Range query throughput

Range query performance ~4x Cassandra



Experiments to show…

- Importance of batching
- Freshness/performance tradeoff
- Throughput and scalability of ingest
- Performance for queries
  - Both “small” and “big” queries
- **Consistency relative to Cassandra**
- Freshness relative to Cassandra

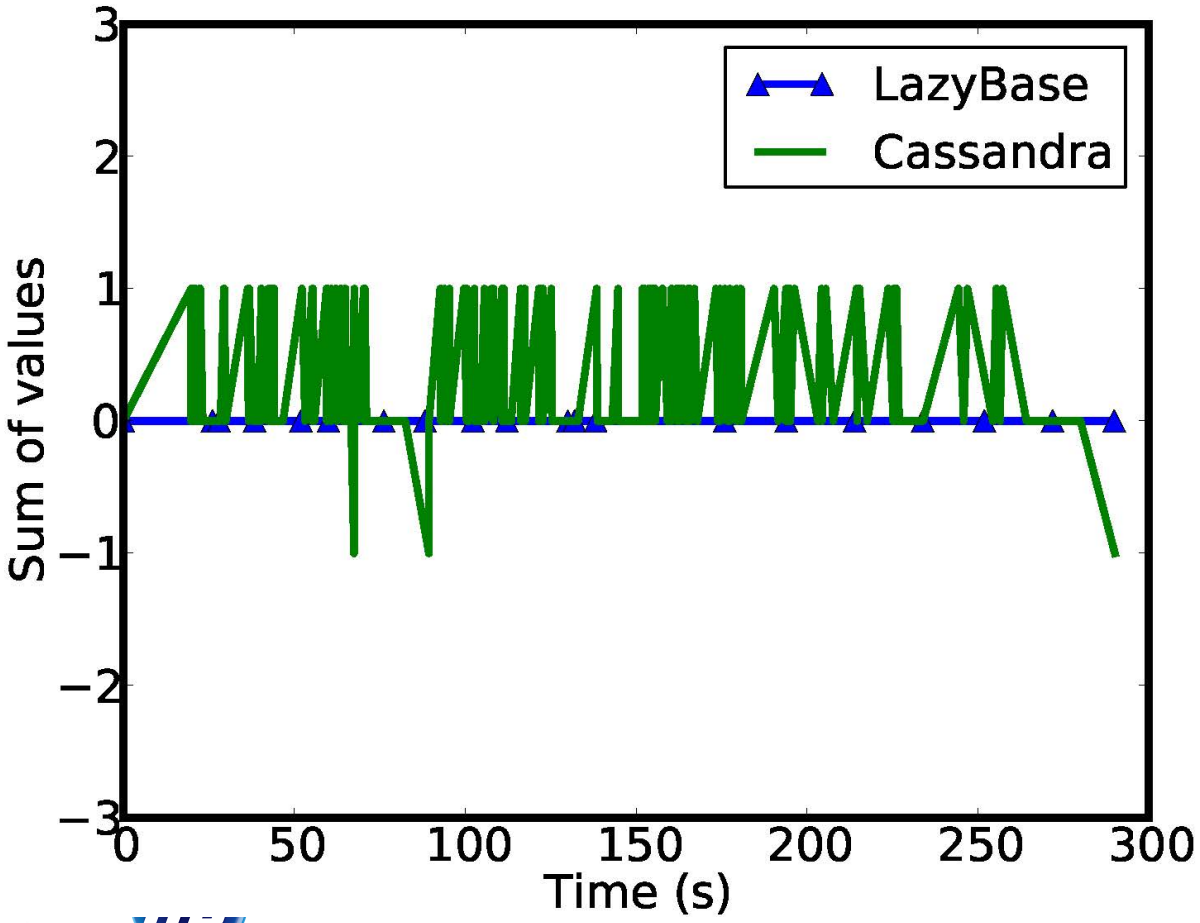
Consistency experiments

- Goal: test atomicity of updates

- Table with 2 rows, A and B
- Each row stores an integer
- Write transactions simultaneously increment A and decrement B
- A+B should always be 0

Sum = A+B

LazyBase maintains inter-row consistency



Experiments to show…

- Importance of batching
- Freshness/performance tradeoff
- Throughput and scalability of ingest
- Performance for queries
  - Both “small” and “big” queries
- Consistency relative to Cassandra
- **Freshness relative to Cassandra**

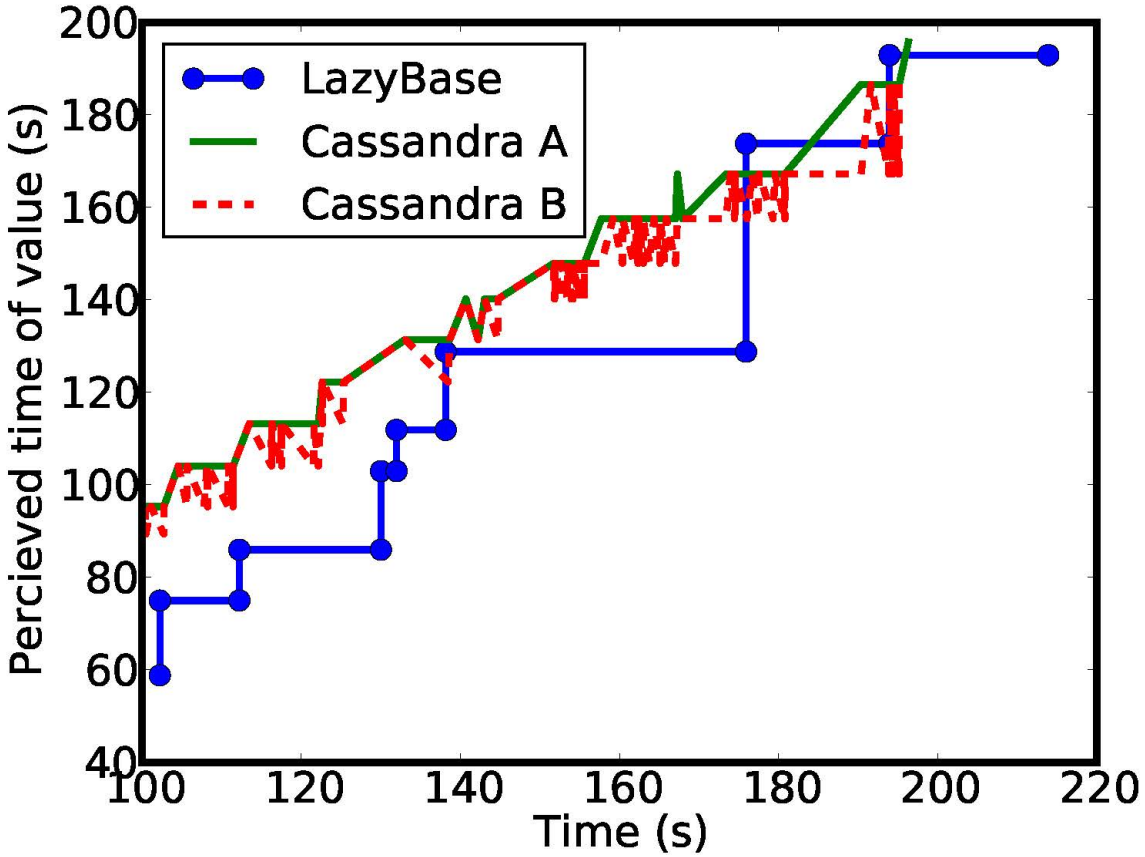
Freshness experiment

- Goal: test freshness in consistency experiment

- Same 2 rows, A and B, but add timestamps
- Timestamp shows age of data in database
- A and B should have the same timestamp

Freshness

LazyBase results may be stale, timestamps are nondecreasing



Summary

- Provide performance, consistency and freshness
  - Batching improves update throughput, hurts latency
- Separate ideas of consistency and freshness
  - Tradeoff between freshness and latency
- Use pipelined database to meet these needs
  - Allow queries to access intermediate pipeline data

Future work

Soft causality constraints

- LazyBase assumes updates have total ordering
- In reality it is often useful to reorder updates
  - Shortest Job First policy for improved update latency
  - Process “high priority” updates quickly
- Many systems use **causal consistency**

Causal consistency

- An update can be caused by previous updates
  - A tweet may depend on a user's previous tweets
  - A Tweet can refer to a previous tweet explicitly

- Allow updates to be reordered
  - Must respect causalality
  - If A caused B, then update A must happen before B

Causal consistency is hard

- Need to track causal relationships:
- Must maintain information about each update
  - Can become a performance problem
  - Often infeasible
- May be (practically) impossible
  - Tweets refer to other tweets
  - It is difficult to determine which update a Tweet was in

Estimating causality

- Causality may not be *that* important…
  - Some messages out of order
  - Roll back and re-execute operations

- Can we estimate causal relationships using already available data?
  - Relative time stamps
  - User ID
  - User popularity

“Soft” causal consistency

- Estimating causal relationships means we might occasionally violate them
- Instead of treating them as hard constraints, consider them another objective
- Trade off between causality violations and…
  - Latency
  - Parallelism
  - Intermediate results

Thanks!