Big NoSQL Data, Apache AsterixDB, and Beyond

Michael J. Carey

Computer Science Department

University of California, Irvine

mjcarey@ics.uci.edu

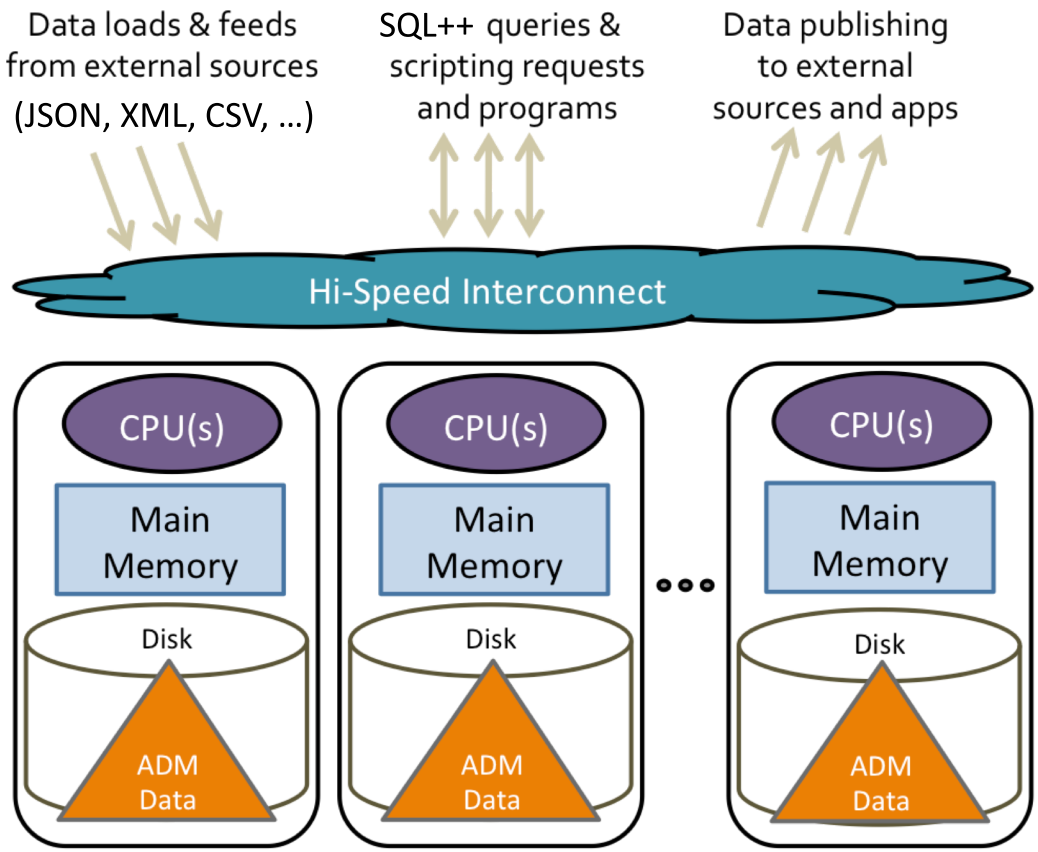
UCIRVINE

(Joint work with ***UC Riverside***
and contributions from ***UC San Diego***)

Today’s Keynote Forecast

Partly cloudy with a 100% chance of data

Apache AsterixDB



Data loads & feeds from external sources (JSON, XML, CSV, …)

SQL++ queries & scripting requests and programs

Data publishing to external sources and apps

Hi-Speed Interconnect

CPU(s)

Main Memory

Disk

ADM Data

CPU(s)

Main Memory

Disk

ADM Data

…

CPU(s)

Main Memory

Disk

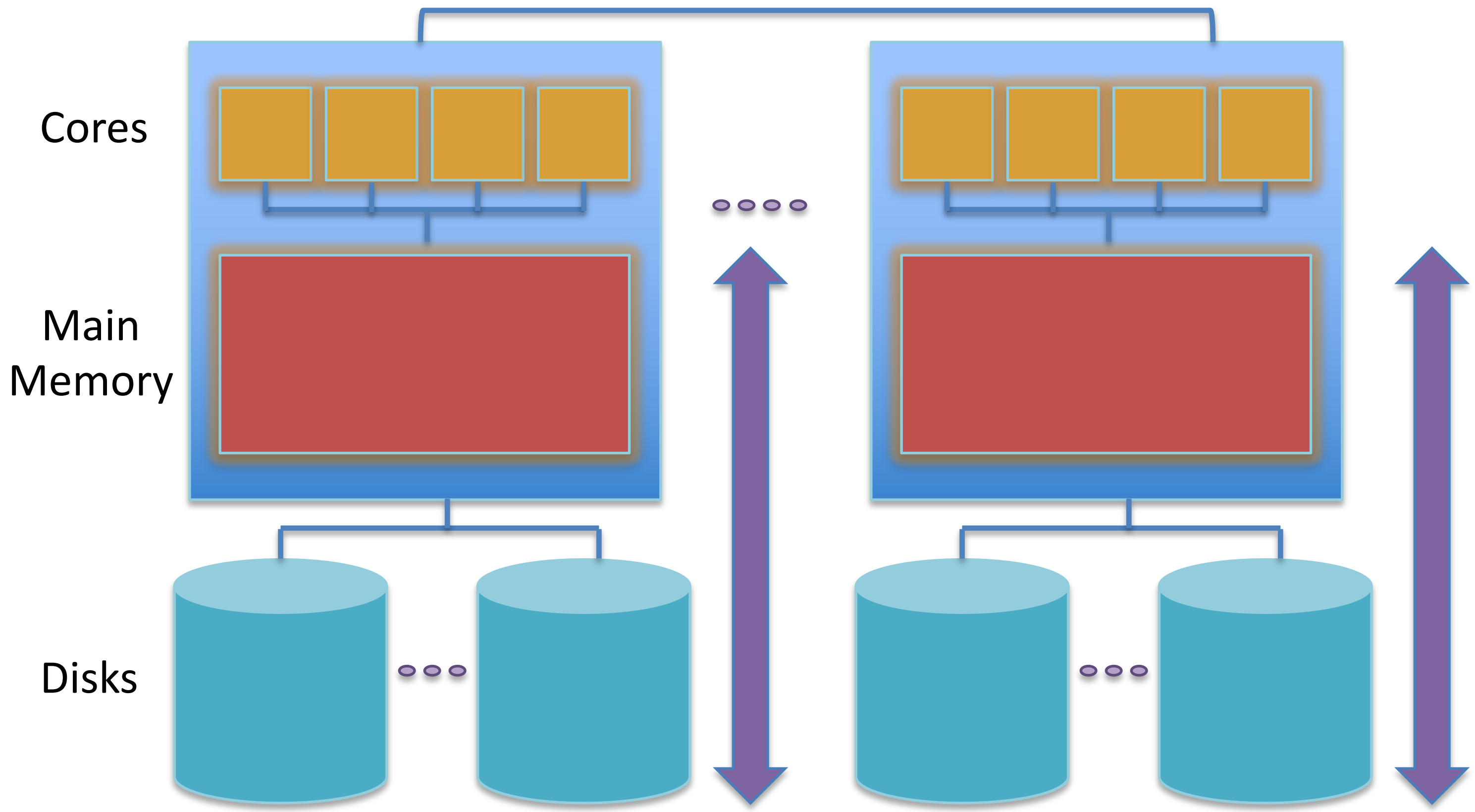
ADM Data

ASTERIX Goal:
To *ingest*, *digest*, *persist*, *index*, *manage*, *query*, *analyze*, and *publish* massive quantities of semistructured information…

http://asterixdb.apache.org/

Asterix * DB ™

Just How **Big** is "Big Data"?

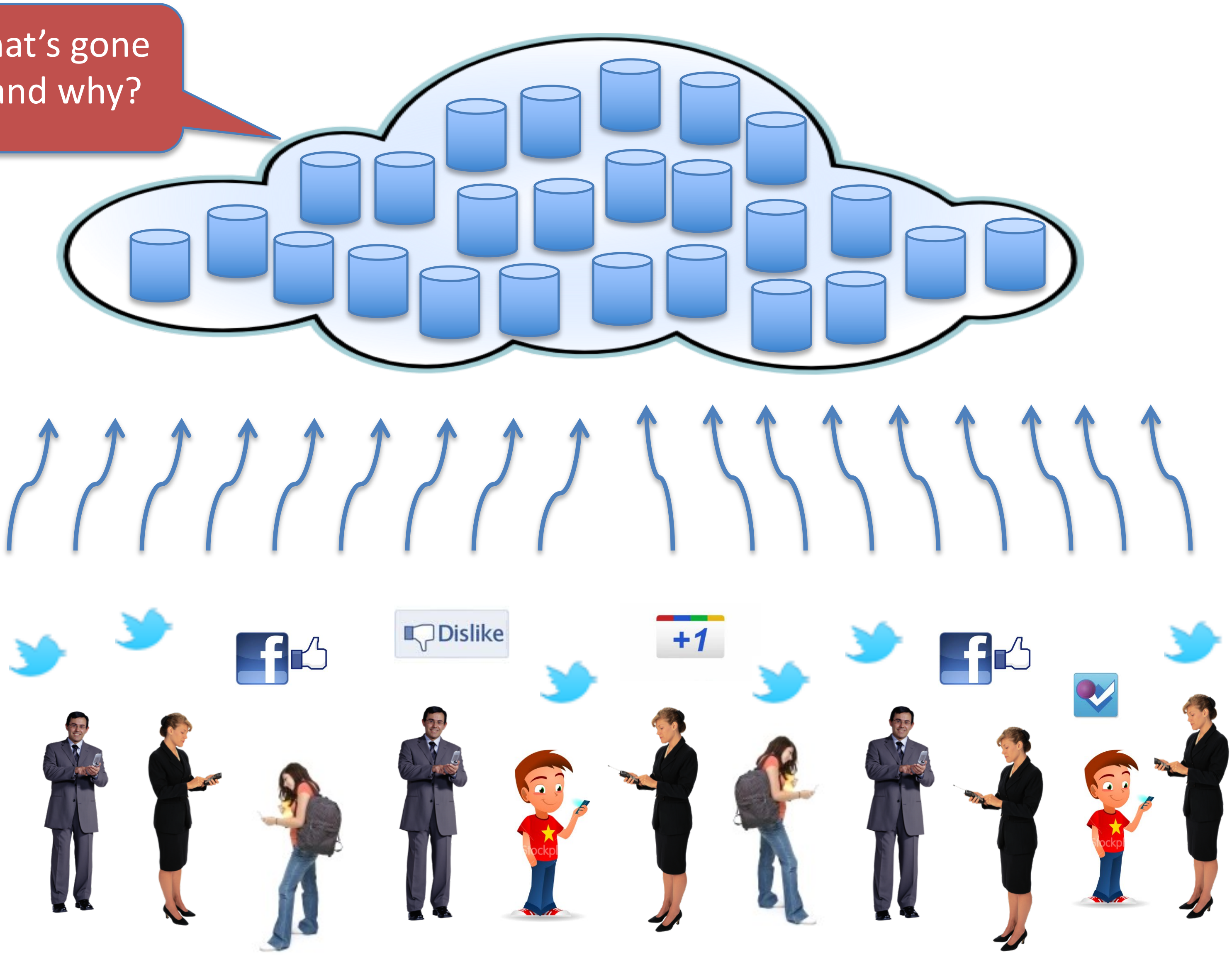


This

is big data!

Big Data / Web Warehousing

So what's gone on – and why?

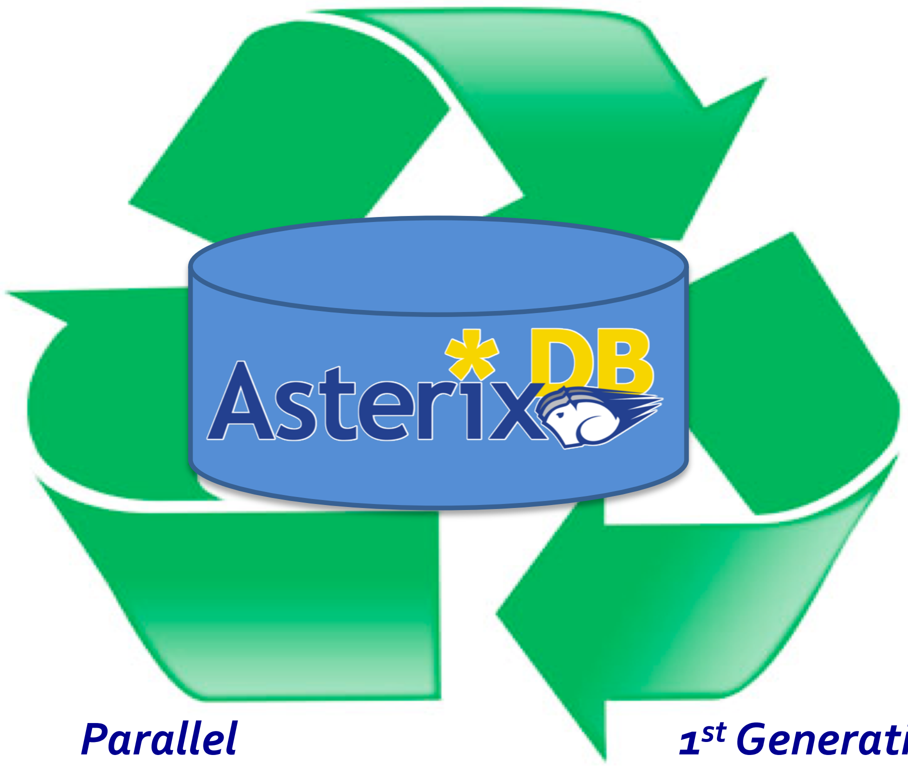


Also: Today's Big Data Tangle

Spark SQL
(Pig)
HIVE
ongoDB
Cassandra
ly scalable, eventually consistent, distributed, structured key-value store.
hadoop
MySQL®
H·BASE

AsterixDB: “One Size Fits a Bunch”

Semistructured Data Management



Parallel Database Systems

1^{st} Generation “Big Data” Systems

BDMS Desiderata:

- Able to **manage** data
- **Flexible** data model
- Full **query** capability
- Continuous data **ingestion**
- Efficient and robust **parallel** runtime
- Cost **proportional** to task at hand
- Support “**Big Data** data types”

⋮

ASTERIX Data Model (ADM)

```
CREATE DATAVERSE TinySocial;
USE TinySocial;

CREATE TYPE GleambookUserType AS {
   id: int,
   alias: string,
   name: string,
   userSince: datetime,
   friendIds: {{ int }},
   employment: [EmploymentType]
};
```

```
CREATE TYPE EmploymentType AS {
   organizationName: string,
   startDate: date,
   endDate: date?
};

CREATE DATASET GleambookUsers
                        (GleambookUserType)
PRIMARY KEY id;
```

Highlights include:

- JSON++ based data model
- Rich type support (spatial, temporal, …)
- Records, lists, bags
- ***Open vs. closed types***

ASTERIX Data Model (ADM)

```
CREATE DATAVERSE TinySocial;
USE TinySocial;

CREATE TYPE GleambookUserType AS {
   id: int
};
```

```
CREATE TYPE EmploymentType AS {
   organizationName: string,
   startDate: date,
   endDate: date?
};

CREATE DATASET GleambookUsers
                        (GleambookUserType)
PRIMARY KEY id;
```

Highlights include:

- JSON++ based data model
- Rich type support (spatial, temporal, …)
- Records, lists, bags
- ***Open vs. closed types***

ASTERIX Data Model (ADM)

```
CREATE DATAVERSE TinySocial;
USE TinySocial;

CREATE TYPE GleambookUserType AS {
   id: int
};

CREATE TYPE GleambookMessageType AS {
      messageId: int,
      authorId: int,
      inResponseTo: int?,
      senderLocation: point?,
      message: string
   };
```

```
CREATE TYPE EmploymentType AS {
   organizationName: string,
   startDate: date,
   endDate: date?
};

CREATE DATASET GleambookUsers
                        (GleambookUserType)
PRIMARY KEY id;

CREATE DATASET GleambookMessages
                        (GleambookMessageType)
PRIMARY KEY messageId;
```

Highlights include:

- JSON++ based data model
- Rich type support (spatial, temporal, …)
- Records, lists, bags
- ***Open vs. closed types***

Ex: GleambookUsers Data

```
{"id”:1, "alias":"Margarita", "name":"MargaritaStoddard", "nickname":"Mags”,
  "userSince":datetime("2012-08-20T10:10:00"),  "friendIds":{{2,3,6,10}},
  "employment": [ {"organizationName":"Codetechno”, "startDate":date("2006-08-06")},
                  {"organizationName":"geomedia" ,    "startDate":date("2010-06-17"),
                                                      "endDate":date("2010-01-26")} ],
  "gender":"F”
},

{"id":2, "alias":"Isbel”, "name":"IsbelDull", "nickname":"Izzy",
  "userSince":datetime("2011-01-22T10:10:00"),  "friendIds":{{1,4}},
  "employment": [ {"organizationName":"Hexviafind",  "startDate":date("2010-04-27")} ]
},

{"id":3, "alias":"Emory", "name":"EmoryUnk”,
  "userSince":datetime("2012-07-10T10:10:00"), "friendIds":{{1,5,8,9}},
  "employment": [ {"organizationName":"geomedia”,  "startDate":date("2010-06-17"),
                                                    "endDate":date("2010-01-26")} ]
},
          . . . . .
```

Other DDL Features

```
CREATE INDEX gbUserSinceIdx ON GleambookUsers(userSince);
CREATE INDEX gbAuthorIdx ON GleambookMessages(authorId) TYPE BTREE;
CREATE INDEX gbSenderLocIndex ON GleambookMessages(senderLocation) TYPE RTREE;
CREATE INDEX gbMessageIdx ONGleambookMessages(message) TYPE KEYWORD;
//-------------------- and also ---------------------------------------------------------------------------
CREATE TYPE AccessLogType AS CLOSED
   { ip: string, time: string, user: string, verb: string, `path`: string,  stat: int32, size: int32 };
CREATE EXTERNAL DATASET AccessLog(AccessLogType) USING localfs
   (("path"="localhost:///Users/mikejcarey 1/extdemo/accesses.txt"),
     ("format"="delimited-text"), ("delimiter"="|"));

CREATE FEED myMsgFeed USING socket_adapter
    (("sockets"="127.0.0.1:10001"), ("address-type"="IP"),
     ("type-name"="GleambookMessageType"), ("format"="adm"));
CONNECT FEED myMsgFeed TO DATASET GleambookMessages;
START FEED myMsgFeed;
```

External data highlights:

- Equal opportunity access
- Feeds to “keep everything!”
- Ingestion, *not* streams

ASTERIX Queries (SQL++ or AQL)

- *Q1:* List the user names and messages sent by Gleambook social network users with less than 3 friends:

```
SELECT user.name AS uname,
            (SELECT VALUE msg.message
             FROM GleambookMessages msg
             WHERE msg.authorId = user.id) AS messages
FROM GleambookUsers user
WHERE COLL_COUNT(user.friendIds) < 3;
```

```
{ "uname": "NilaMilliron", "messages": [  ] }
{ "uname": "WoodrowNehling", "messages": [ " love acast its 3G is good:)" ] }
{ "uname": "IsbelDull", "messages": [ " like product-y the plan is amazing", " like
  product-z its platform is mind-blowing" ] }
. . .
```

SQL++ *(cont.)*

- *Q2:* Identify active users (last 30 days) and group and count them by their numbers of friends:

```
WITH endTime AS current_datetime(),
        startTime AS endTime - duration("P30D")
SELECT nf AS numFriends, COUNT(user) AS activeUsers
FROM GleambookUsers user
LET nf = COLL_COUNT(user.friendIds)
WHERE SOME logrec IN AccessLog SAT
        user.alias = logrec.user
        AND datetime(logrec.time) >= startTime
        AND datetime(logrec.time) <= endTime
GROUP BY nf;
```

```
{ "numFriends": 2, "activeUsers": 1 }
{ "numFriends": 4, "activeUsers": 2 }
 . . .
```

***SQL++ highlights*:**

- UCSD (Papakonstantiou)
- Many features (see docs)
- Spatial & text predicates
- Set-similarity matching

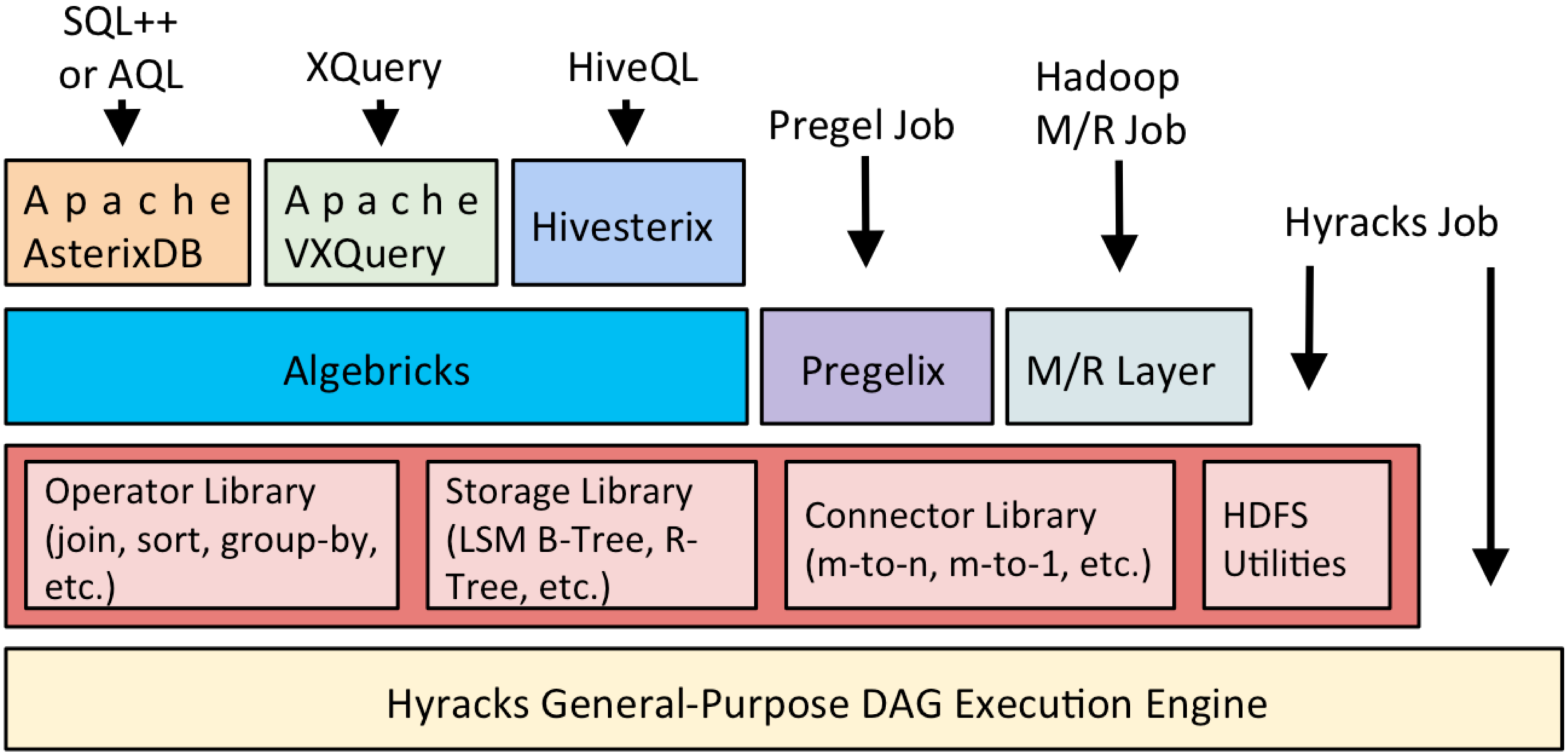
Updates and Transactions

- *Q3:* Add a new user to Gleambook.com:

```
UPSERT INTO GleambookUsers (
{"id":667,"alias":”dfrump",
 "name":"DonaldFrump",
  "nickname":"Frumpkin",
  "userSince":datetime("2017-01-01T00:00:00"),
  "friendIds":{{ }},
  "employment":[{"organizationName":"USA",
  "startDate":date("2017-01-20")}],
  "gender":"M"}
);
```

- **Insert**, **delete**, and **upsert** ops
- Key-value store-like transactions (w/record-level atomicity)
- Index-consistent

Software Stack



SQL++ or AQL

XQuery

HiveQL

Pregel Job

Hadoop M/R Job

Hyracks Job

Apache AsterixDB

Apache VXQuery

Hivesterix

Algebricks

Pregelix

M/R Layer

Operator Library (join, sort, group-by, etc.)

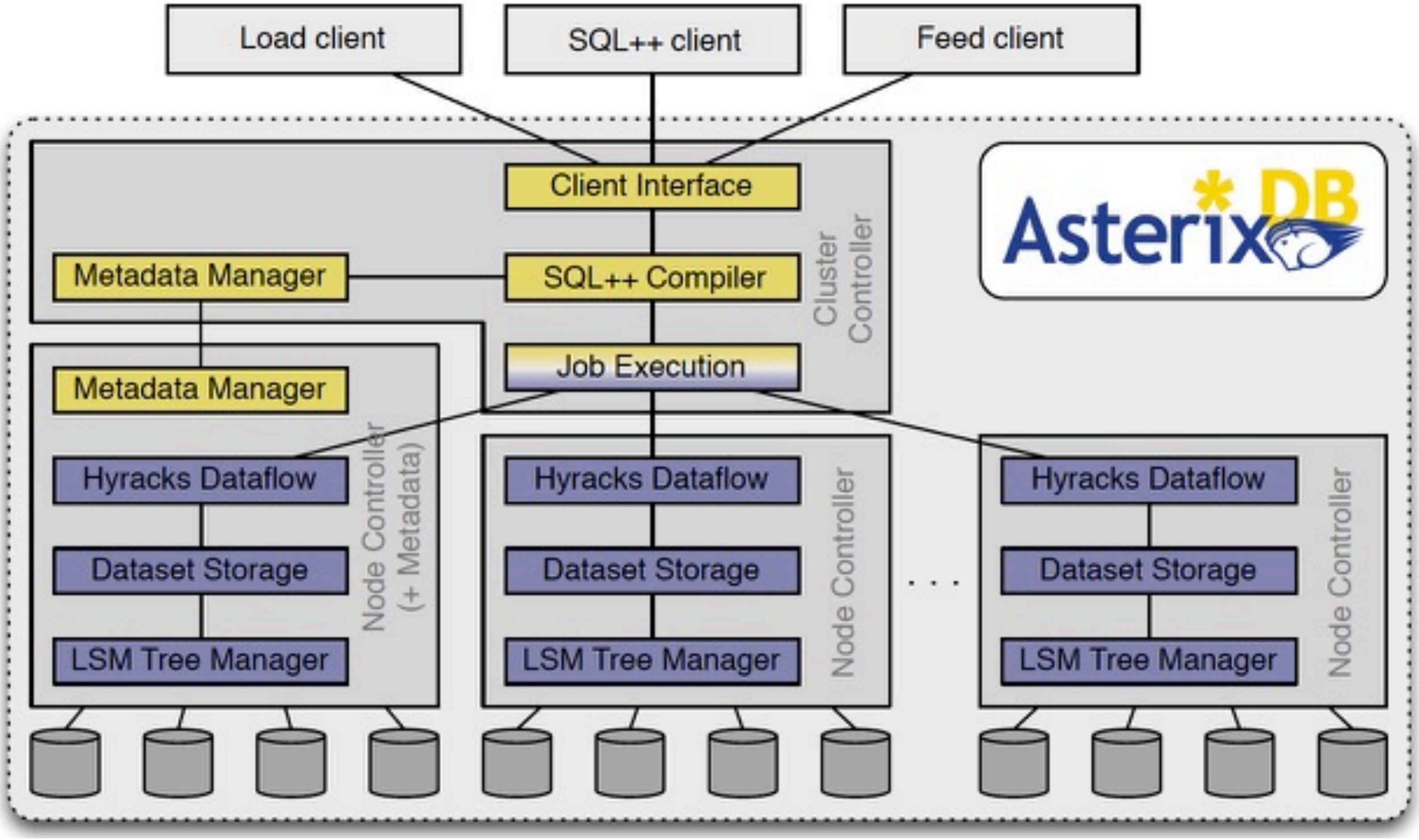
Storage Library (LSM B-Tree, R-Tree, etc.)

Connector Library (m-to-n, m-to-1, etc.)

HDFS Utilities

Hyracks General-Purpose DAG Execution Engine

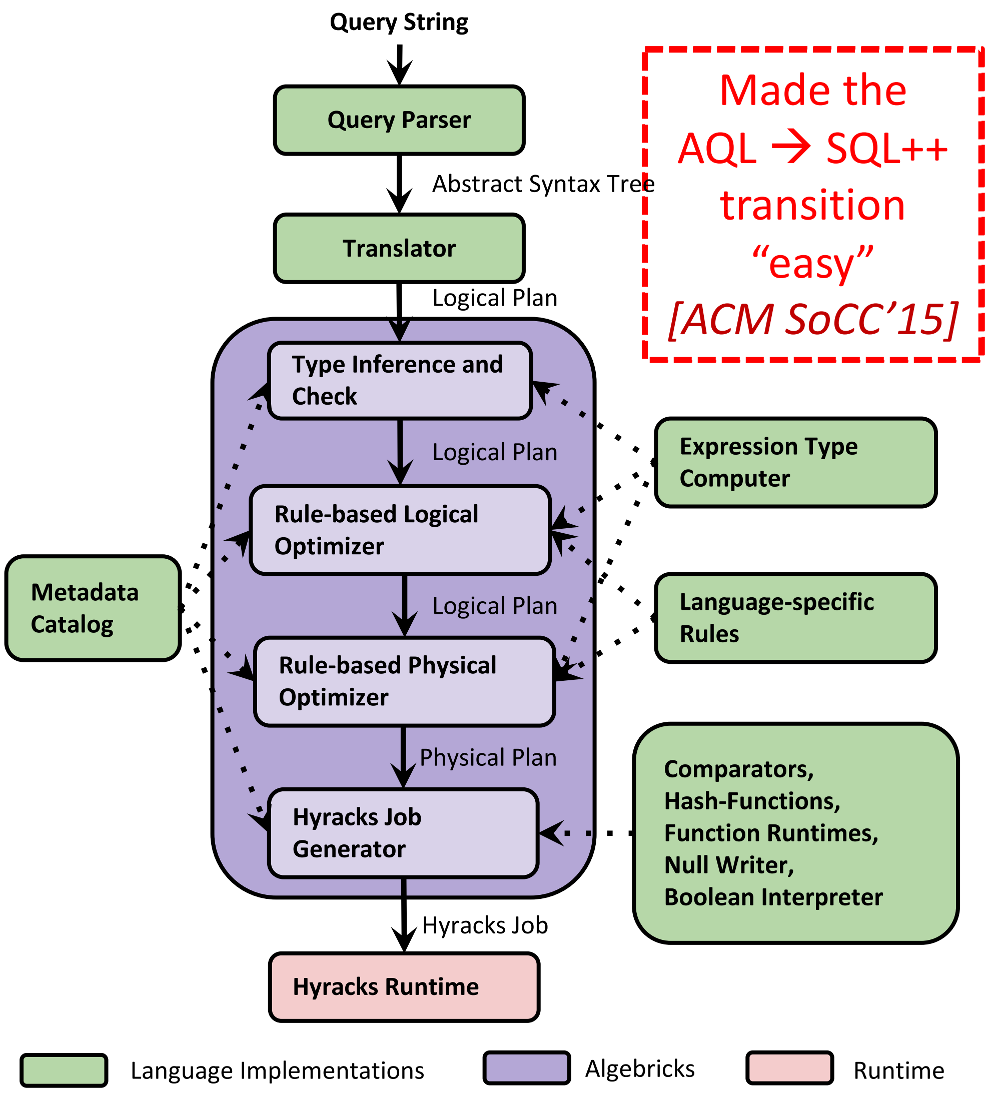
AsterixDB System Overview



Hyracks Dataflow Runtime

- Partitioned-parallel platform for data-intensive computing
- Job = dataflow DAG of operators and connectors
  - Operators consume and produce ***partitions*** of data
  - Connectors ***route*** (repartition) data between operators
- Hyracks *vs.* the "competition"
  - Based on time-tested parallel database principles
  - *vs.* Hadoop MR: More flexible model and less "pessimistic"
  - *vs.* SQL-on-Hadoop runtimes (e.g., Spark): Emphasis on out-of-core execution and adherence to memory budgets
  - Fast job activation, data pipelining, binary format, state-of-the-art DB style operators (hash-based, indexed, ...)
- Early tests at Yahoo! Labs on 180 nodes (1440 cores, 720 disks)

Algebricks Query Compiler Framework



Made the AQL → SQL++ transition "easy" *[ACM SoCC'15]*

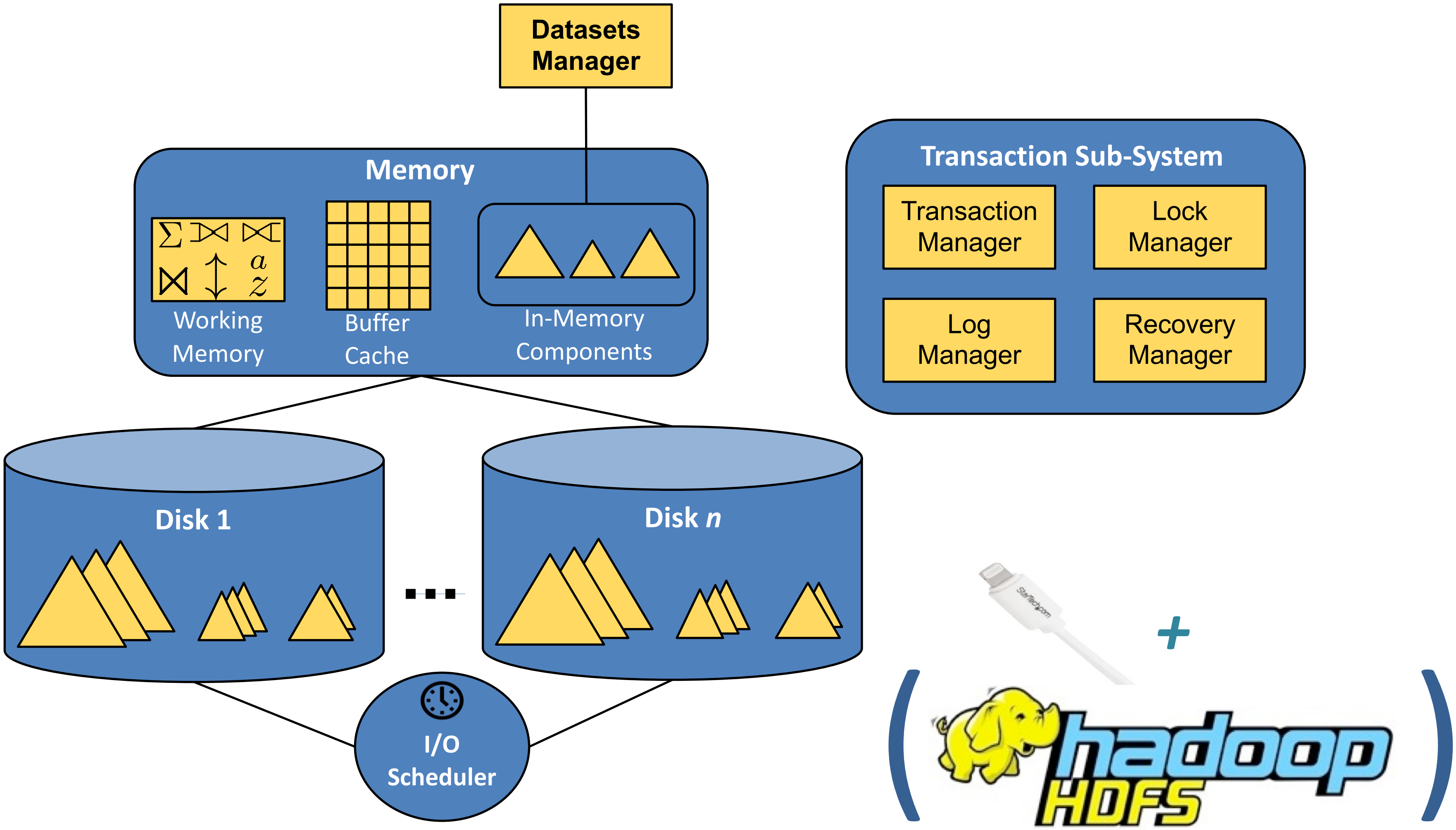
Algebricks

- Logical Operators
- Logical Expressions
- Metadata Interface
- Model-Neutral Logical Rewrite Rules
- Physical Operators
- Model-Neutral Physical Rewrite Rules
- Hyracks Job Generator

Target Query Language

- Query Parser (AST)
- AST Translator
- Metadata Catalog
- Expression Type Computer
- Logical Rewrite Rules
- Physical Rewrite Rules
- Language Specifics

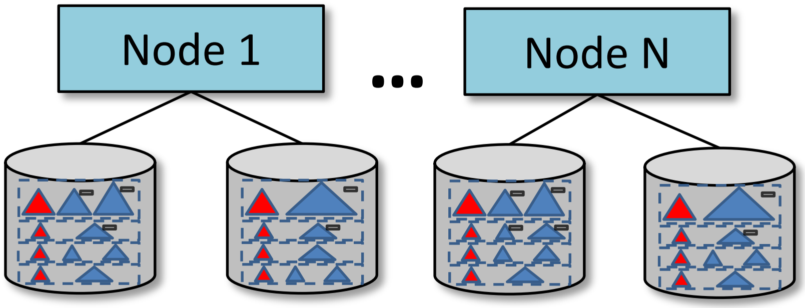
Native Storage Management

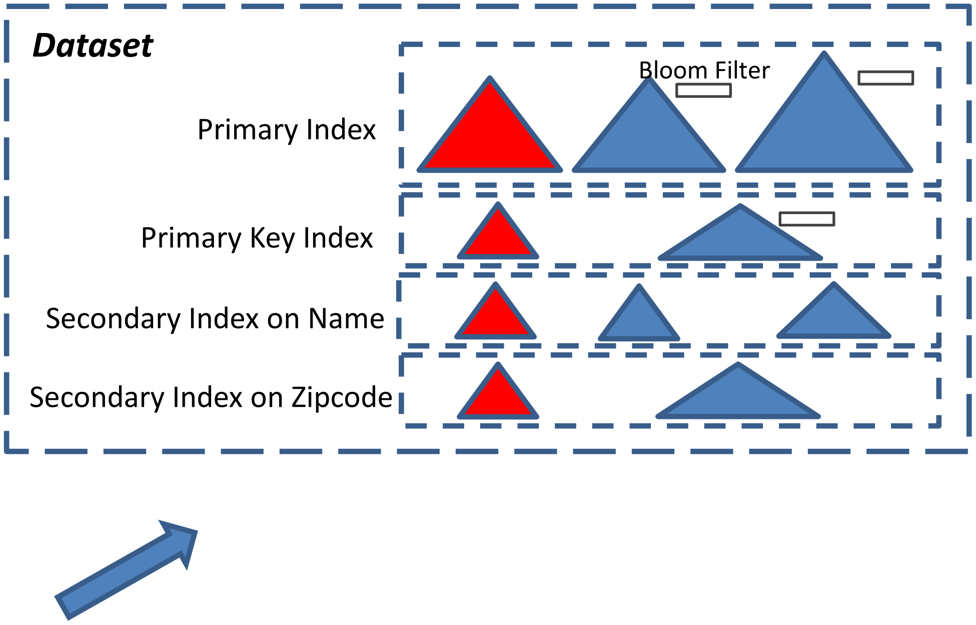


An Indexed Dataset

Partitioned local storage approach

- Hashed on primary key (PK)
- Primary index w/ PK + record
- Secondary index(es) with SK + PK
- Record updates are always local

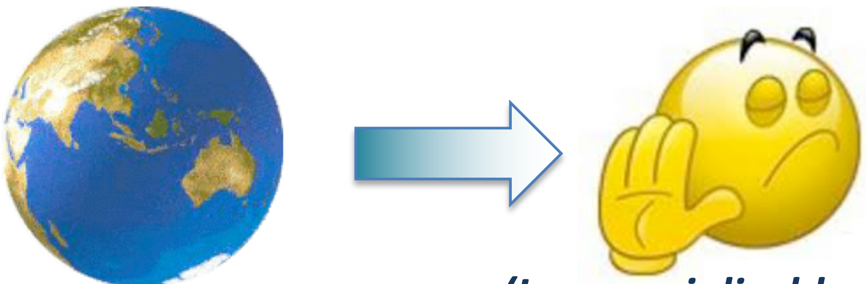




Transaction Support

- Key-value store-like transaction semantics
  - Entity-level transactions (by key) within “transactors”
  - Atomic insert, delete, and upsert (including indexing)
  - Concurrency control (based on entity-level locking)
  - Crash recovery (based on no-steal logging + shadowing)

- Expected use of AsterixDB is to model, capture, and ***track*** the “state of the world” (not to ***be*** it)...



SELECT ... FROM Weather W...
  // return current conditions by city

(Long serializable reads)

Example AsterixDB Use Cases

- Potential use case areas include
  - Behavioral science
  - Cell phone event analytics
  - Social data analytics
  - Public health
  - Cluster management log analytics
  - Power usage monitoring
  - IoT data storage and querying
  - ....

Current Status

- 4 year initial NSF project (250+ KLOC), started 2009
- Now available as *Apache AsterixDB*
  - Semistructured “NoSQL” style data model
  - Declarative queries, inserts, deletes, upserts (SQL++)
  - Scalable parallel query execution
  - Data storage/indexing (primary & secondary, LSM-based)
  - Internal and external datasets both supported
  - Rich set of data types (including text, time, location)
  - Fuzzy and spatial query processing
  - NoSQL-like transactions (for inserts/deletes)
  - Data feeds and indexes for external datasets
  - ....

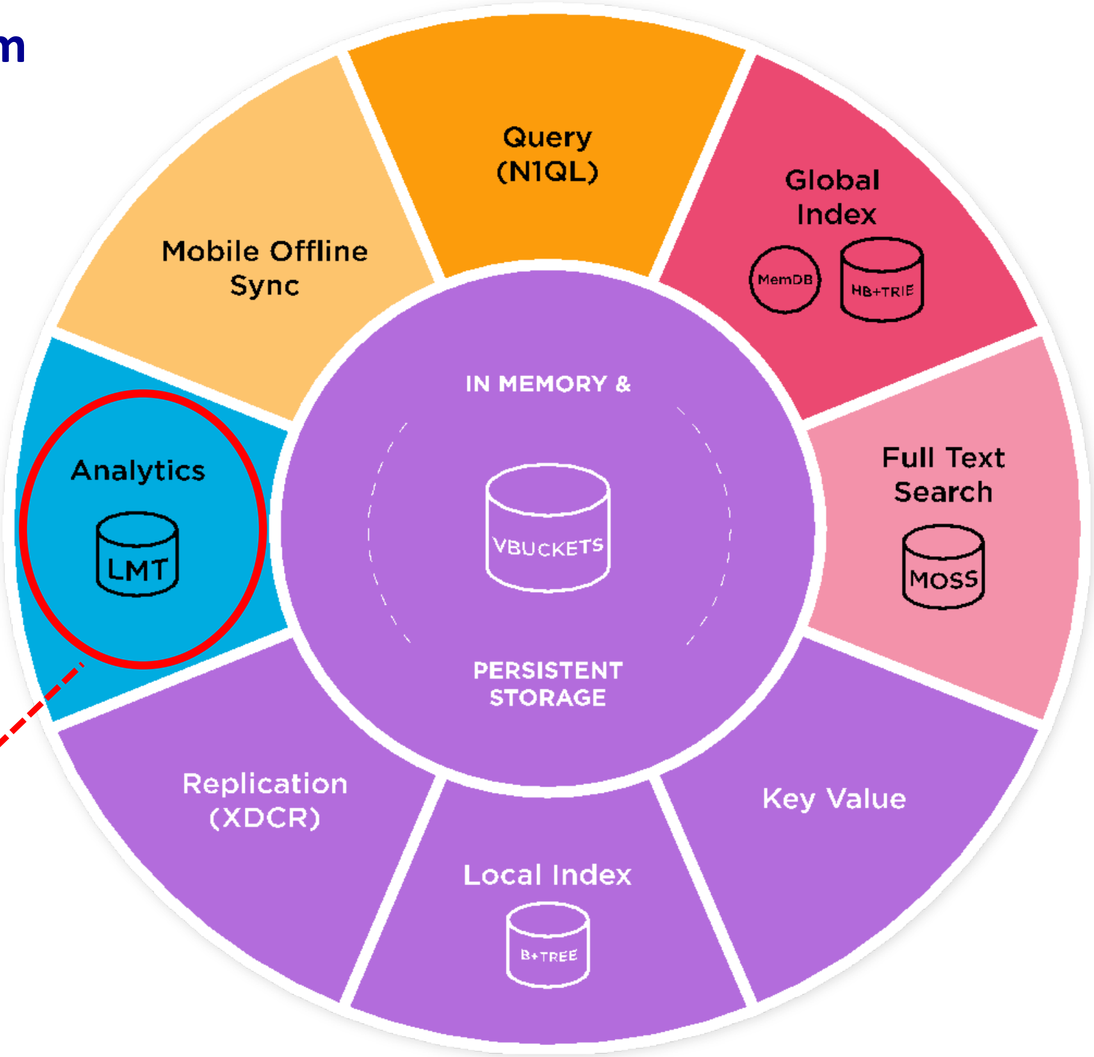
Research Roadmap: Big NoSQL Data

- Big NoSQL query processing on large shared clusters
  - Memory management (long term and short term)
- General-purpose LSM-based storage management
  - Primary and (multiple) secondary indexes and queries
  - Mutation and component management policies
  - Lifecycle exploitation (e.g., for incremental statistics)
- Generalized data “compression” and restructuring
  - Schema-like efficiency in a schema-free world
  - Column-like storage in a column-free (semistructured!) world
- Transactions revisited
  - What exactly were we thinking in the 70’s? (Hmmm….)
- AsterixDB meets ML and (social) Data Science (→)
- Big Data visualization (→ →)

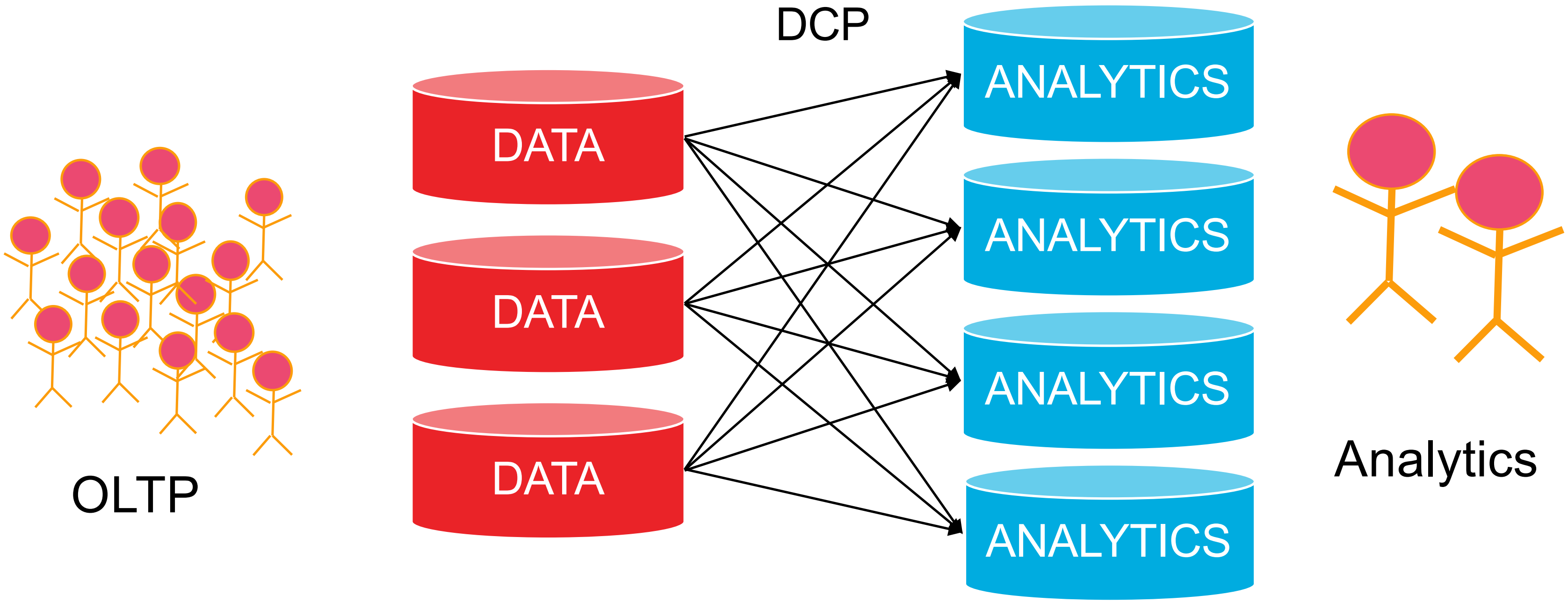
Commercial Use: NoSQL Analytics

Couchbase Data Platform

- ✓ Service-Centric Clustered Data System
  - ✓ Multi-process Architecture
  - ✓ Dynamic Distribution of Facilities
  - ✓ Cluster Map Distribution
  - ✓ Automatic Failover
  - ✓ Enterprise Monitoring/Management
  - ✓ Security
- ✓ Offline Mobile Data Integration
- ✓ Streaming REST API
- ✓ SQL-like Query Engine for JSON
- ✓ Clustered* Global Indexes
- ✓ Lowest Latency Key-Value API
- ✓ Active-Active Inter-DC Replication
- ✓ Local Aggregate Indexes
- ✓ Full-Text Search*
- ✓ **Operational Analytics***



Couchbase Analytics Service



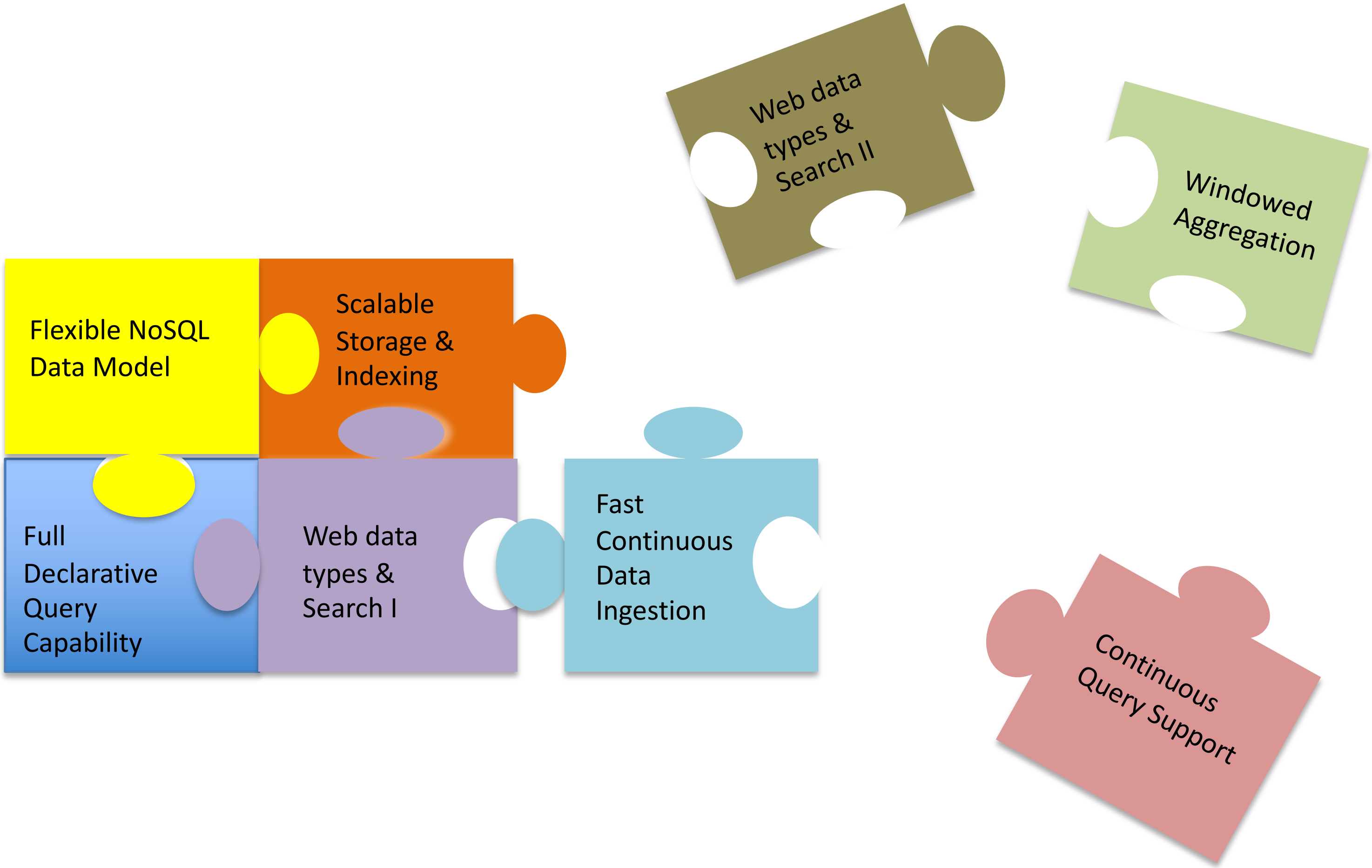
- Separate services, separate nodes
  - Performance isolation (HTAP-like)
  - Separate scale-out based on needs
  - Parallel (M:N) connectivity for performance

("NoETL for NoSQL")

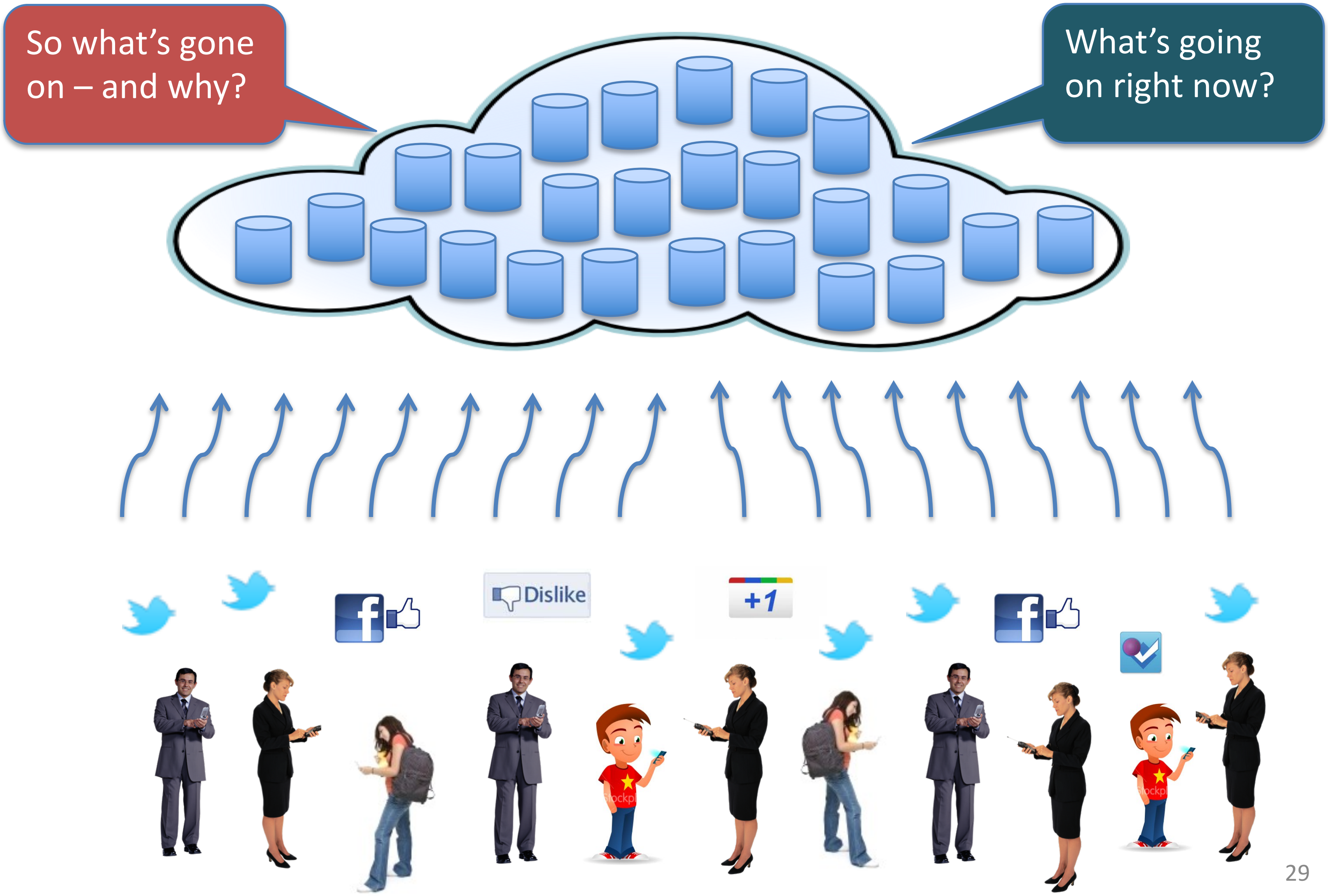
For More Information

- Asterix project UCI/UCR research home
  - http://asterix.ics.uci.edu/
- Apache AsterixDB home
  - http://asterixdb.apache.org/
- SQL++ Primer (to get started)
  - http://asterixdb.apache.org/docs/0.9.4/index.html
- SQL++ Tutorial
  - D. Chamberlin, *SQL++ for SQL Users* (see Couchbase website or Apache AsterixDB site)

Research Roadmap: Big Active Data


Flexible NoSQL Data Model
Scalable Storage & Indexing
Full Declarative Query Capability
Web data types & Search I
Fast Continuous Data Ingestion
Web data types & Search II
Windowed Aggregation
Continuous Query Support

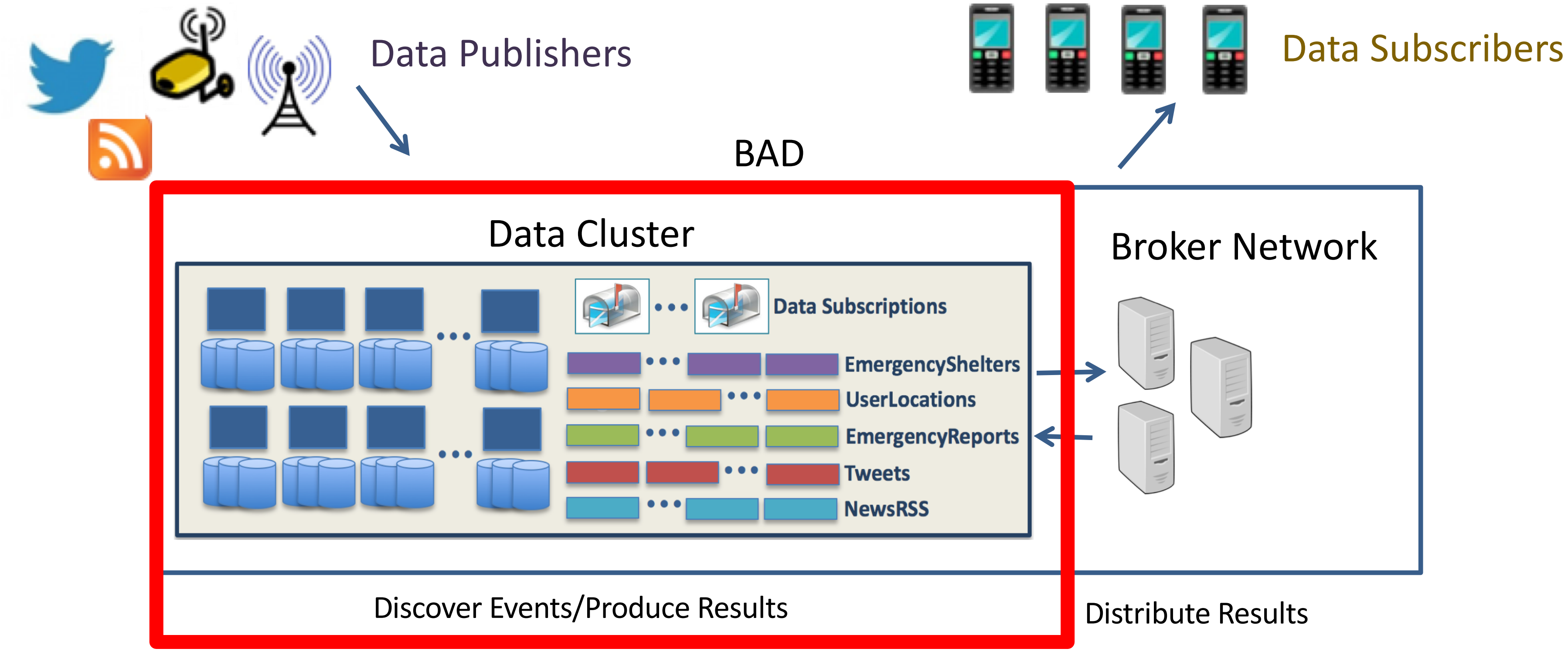
Our Original Motivation



So what's gone on – and why?

What's going on right now?

Big Active Data (BAD) from 10K Feet



Example BAD User Query

- “***Whenever*** I am in the impact zone of some emergency, notify me with the message for the emergency and all of the nearby emergency shelters.”
- This continuous query joins three data sources:
  - Emergency Report Data
  - User Location Data
  - Emergency Shelter Data

What Constitutes An “Event”?

- There are **three** ways our example might yield new results:

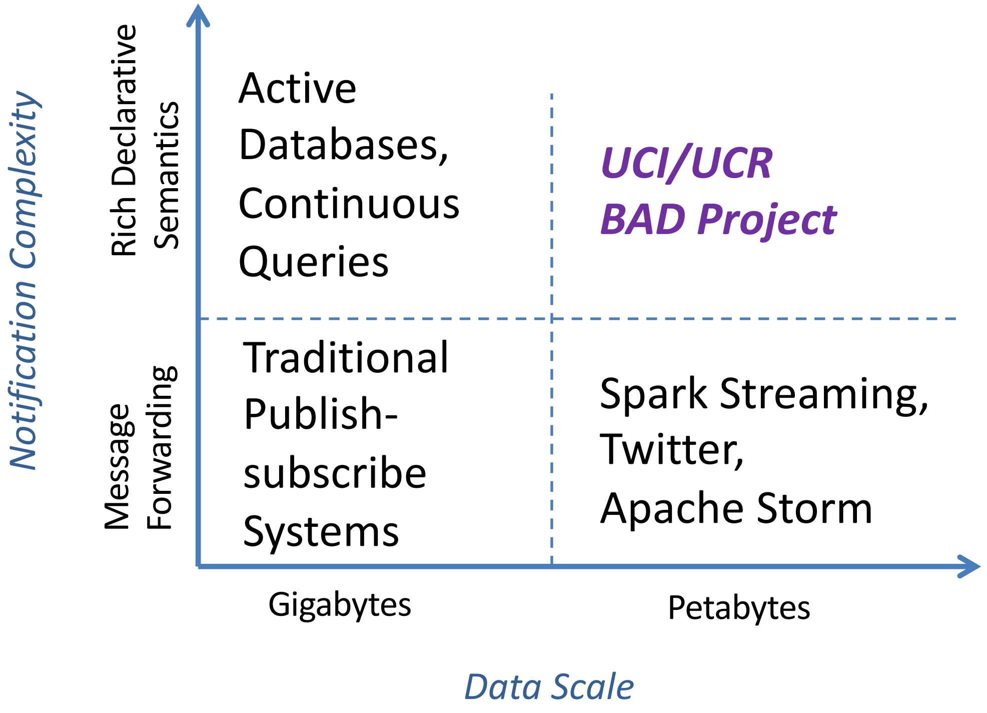
1. A user enters the impact zone of an active emergency
2. An emergency arises at a user’s current location
3. An ad-hoc triage center is set up for an active emergency

(More complex than content-based routing or windowed CQ!)

What's Needed for *Big Active* Data?

- Needs **unmet** by Pub/Sub or traditional CQ (streaming):
  - Data *in context*
    - Incoming data may be important due to **relationships** with other existing data, including historical and static data
  - *Actionable* notifications
    - User notifications may need to be **enriched** based on other existing data
  - *Retrospective* Big Data analytics
    - Need "in the moment" processing plus **later queries/analyses** on the **collected data**
- "From Petabytes to Megafolks in Milliseconds"
  - *Goal:* Big Data backend (*petabytes*) for population-scale applications (*megafolks*), enabling individualized continuous queries, delivering results as fast as possible (*milliseconds*)

Related Work in a Nutshell



Notification Complexity

Rich Declarative Semantics

Message Forwarding

Active Databases, Continuous Queries

UCI/UCR BAD Project

Traditional Publish-subscribe Systems

Spark Streaming, Twitter, Apache Storm

Gigabytes

Petabytes

Data Scale

Particularly BAD Inspirations

- Two particularly relevant prior projects/systems...
  - NiagaraCQ
  - Spatial Alarms
- Each advanced the idea of turning queries into **stored data**
  - Rather than creating specialized data flows, process many continuous queries simply by joining data with queries (a **data-centric** approach)
  - Able to scale well, but both had (very) limited query languages

Remninder: Queries in AsterixDB

```
select e.message, e.impactZone,
   (select value s from EmergencyShelters s
    where spatial_intersect (e.impactZone, s.location))
    as shelters
from EmergencyReports e;
```

Data Ingestion in AsterixDB

- Use Asterix Feeds to rapidly ingest new data on a continuous basis
- We can create a data feed for EmergencyReports so that they can be rapidly ingested as they are being produced by data publishers

```
create feed EmergencyFeed using EmergencyFeedAdapter (...);
connect feed EmergencyFeed to dataset EmergencyReports;
start feed EmergencyFeed;
```

(User location observations are another natural data feed use ... →)

Tracking User Locations

```
create type UserLocation as {
    id: uuid,
    userId: int ,
    location: point,
    timestamp: datetime
};
create dataset UserLocations(UserLocation) primary key id;

create feed UserLocationsFeed
          using UserLocationsFeedAdapter (...);
connect feed UserLocationsFeed to dataset UserLocations;
start feed EmergencyFeed;
```

Example Application Data

Moderately dynamic data

EmergencyReports

timestamp	emergencyType	state	message	expirationTime	ImpactZone	…
2015-11-25 09:00:00	tornado	KS	Please proceed to the nearest shelter	2015-11-25 09:30:00	circle("300,20 12.0")	…
2015-11-25 09:02:00	tornado	IA	Please proceed to the nearest shelter	2015-11-25 09:32:00	circle("100,5 10.0")	…
2015-11-25 09:04:00	tornado	KS	Please proceed to the nearest shelter	2015-11-25 10:04:00	circle("300,10 5.0")	…
2015-11-25 09:05:00	flood	IA	Shelters will provide drinkable water	2015-11-25 09:10:00	circle("105,15 50.6")	…
…	…	…	…	…	…	…

EmergencyShelters

shelterName	location
Downtown Evacuation Center	point("100,10")
Public Shelter 152	point("100,20")
Public Shelter 148	point("100,100")
…	…

Relatively static data

UserLocations

timestamp	userId	location
2015-11-25 09:01:00	1	point("101,12")
2015-11-25 09:09:00	2	point("105,22")
2015-11-25 09:15:00	3	point("113,115")
…	…	…

Highly dynamic data

New: Channels in BAD Asterix

- The Channel Model
  - A Channel is a parameterized version of a query that will continue to execute over time
  - Users subscribe with individualized parameters
  - *Goal*: Lots of Channels, each with lots of subscriptions
- Type 1: *Repetitive Channels*
  - “data cron job”
  - Executes periodically (e.g., every five minutes)
  - Notifications include the **full result** at each execution
- Type 2: *Continuous Channels*
  - Executes on data changes
  - Checks whether these changes contribute new results
  - Notifications include just the **differential result**

An Overall Example

- Suppose we have two sample channels running:
  - ***Repetitive***: “Select the message and impact zone for tornados occurring within the last hour in my state”
  - ***Continuous***: “Whenever I’m in the impact zone of some emergency, notify me with the message for the emergency, its impact zone, and all emergency shelters that are within that impact zone.”
- Let’s look at the channel DDL and the (internal) workings of the system in this scenario…

DDL for Repetitive Channel

```
create function TornadoesInState  (state) {
(select r as reports from
(select * from EmergencyReports r
                where r.timestamp > current_datetime() - day_time_duration("PT1H") ) r
 where r.emergencyType = "tornado"
     and r.state = state)
};
```

Notice the state parameter

Reports within current datetime minus 1 hour

```
create repetitive channel TornadoesInStateChannel
    using TornadoesInState@1 period duration("P1H");
```

Every hour find tornados in the last hour

```
subscribe to TornadoesInStateChannel("IA");
subscribe to TornadoesInStateChannel("KS");
```

Subscriptions with parameter values (states)

DDL for Continuous Channel

```
create function EmergenciesNearUser(userId) {
(select e.message, e.impactZone,
  (select value s from EmergencyShelters s
    where spatial_intersect(e.impactZone, s.location)) as shelters
from EmergencyReports e, UserLocations u
where u.userId = userId
    and spatial_intersect(e.impactZone, u.location)
     and u.timestamp >= e.timestamp
     and u.timestamp <= e.expirationTime)
};
create continuous channel EmergenciesNearUserChannel
    using EmergenciesNearUser@1;

subscribe to EmergenciesNearUserChannel ("12345”) on Broker3;
```

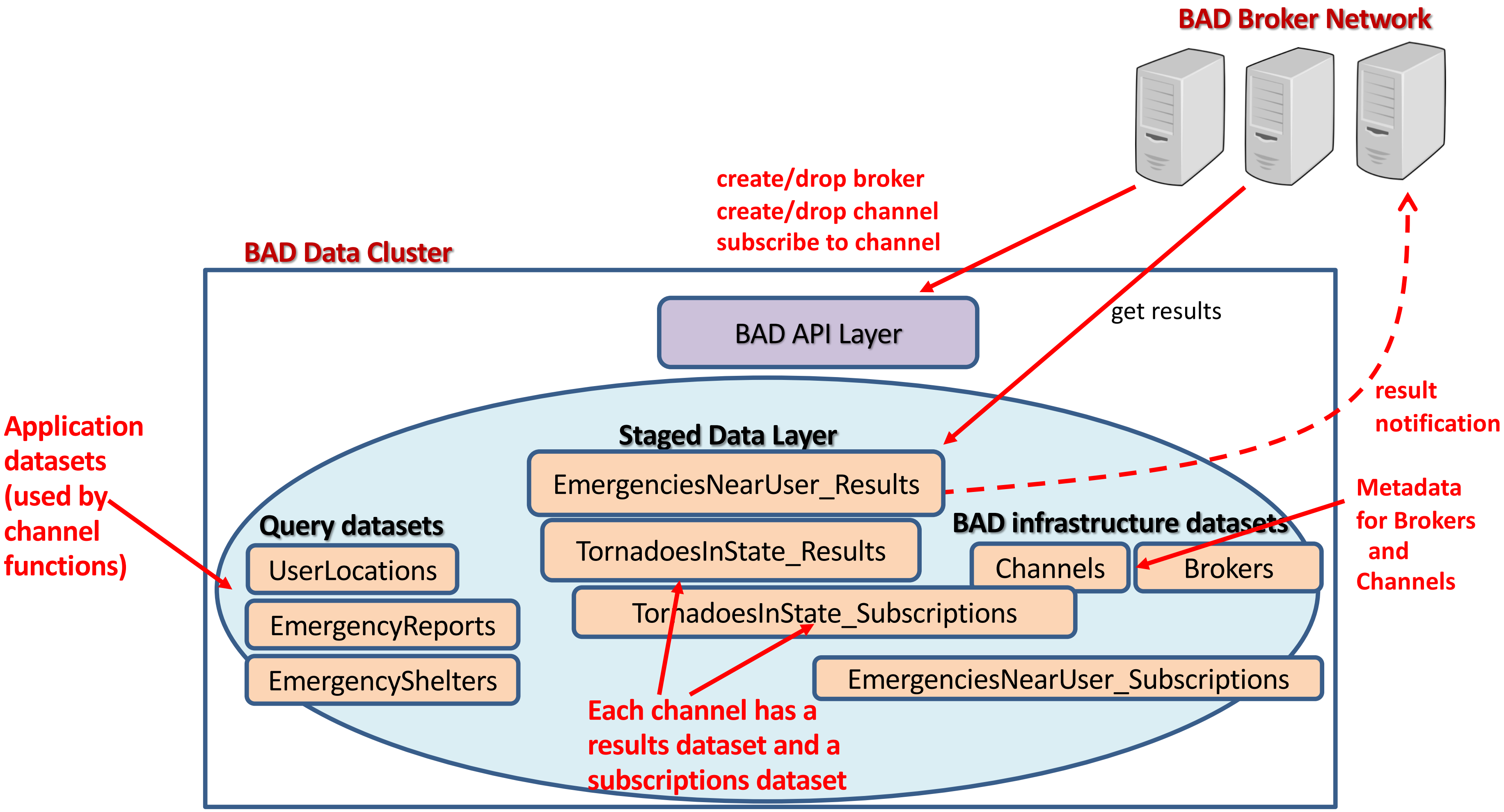
Notice the user parameter

Enrichment of results (nested query)

Spatiotemporal Join

Subscription with parameter value (userId)

Broker/Cluster Interaction



BAD Broker Network

create/drop broker
create/drop channel
subscribe to channel

BAD Data Cluster

get results

BAD API Layer

result notification

Application datasets (used by channel functions)

Staged Data Layer

EmergenciesNearUser_Results

Query datasets

UserLocations

EmergencyReports

EmergencyShelters

TornadoesInState_Results

BAD infrastructure datasets

Channels

Brokers

Metadata for Brokers and Channels

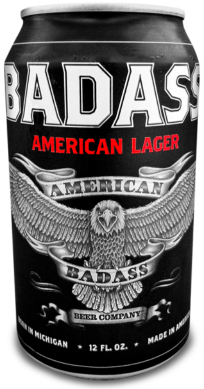
TornadoesInState_Subscriptions

EmergenciesNearUser_Subscriptions

Each channel has a results dataset and a subscriptions dataset

Aiming to be a BAD Asterix

BAD system implementation progress so far:



1. AsterixDB itself (including feeds)
2. Broker creation
3. Repetitive channel creation
4. Subscription creation
5. Result retrieval by brokers
6. Removal of subscriptions, channels, and brokers
7. Optimization of repetitive channels
8. Initial performance and scalability testing

Lots of BAD Plans Ahead

- Implementation of (batch) continuous channels
- More performance and scalability testing
- Non-monotonic queries and data
- Scalable distributed broker network (in progress)
- Framework and tools for building BAD applications
- .....

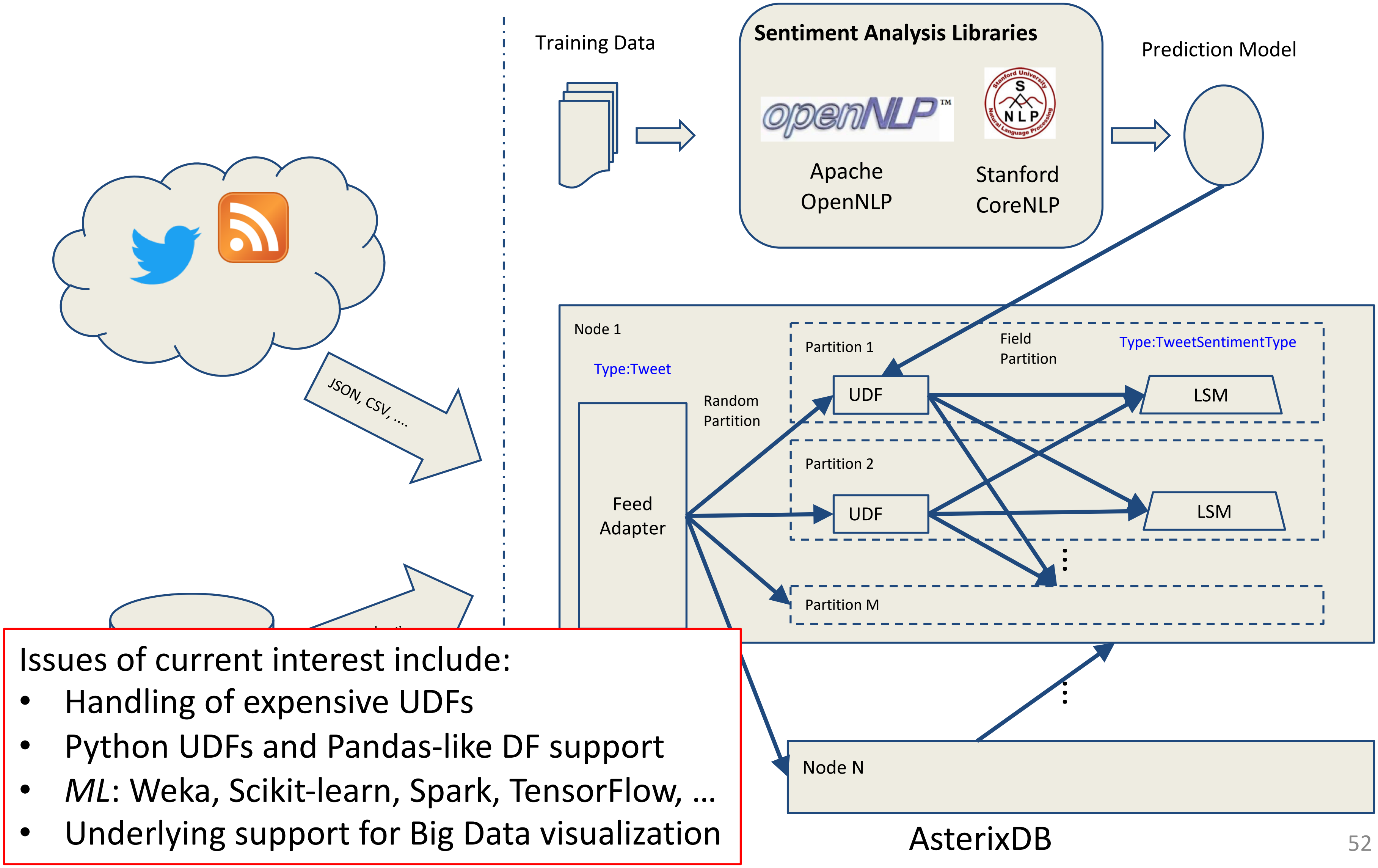
Some BAD Memories For You

- Distinguishing characteristics of a truly BAD platform
  - *Data in context*
    - Incoming data may be important due to its **relationships** with existing data
  - *Actionable notifications*
    - User notifications may need to be **enriched** based on other existing data
  - *Big Data analytics*
    - Able to do **retrospective** queries (and other **analyses**) on the data as a **whole**
- BAD Cluster: BAD extensions to Apache AsterixDB
  - Brokers, channels, and subscriptions
  - Initial (data-centric) internals
- Bad Broker network: Work in progress by our middleware colleagues

Getting BAD Information

- Project overview paper (with a data focus)
  - M. Carey, S. Jacobs, and V. Tsotras, “Breaking BAD: A Data Serving Vision for Big Active Data”, *Proc. of the 10th ACM Int’l. Conf. on Distributed and Event-Based Systems (ACM DEBS)*, Irvine, CA, June 2016.
- Current project status (again with a data focus)
  - S. Jacobs, X. Wang, M. Carey, V. Tsotras, and Y. Uddin, "BAD to the Bone: Big Active Data at its Core", submitted for publication, July 2018.
- UCI/UCR BAD project website
  - http://asterix.ics.uci.edu/bigactivedata/

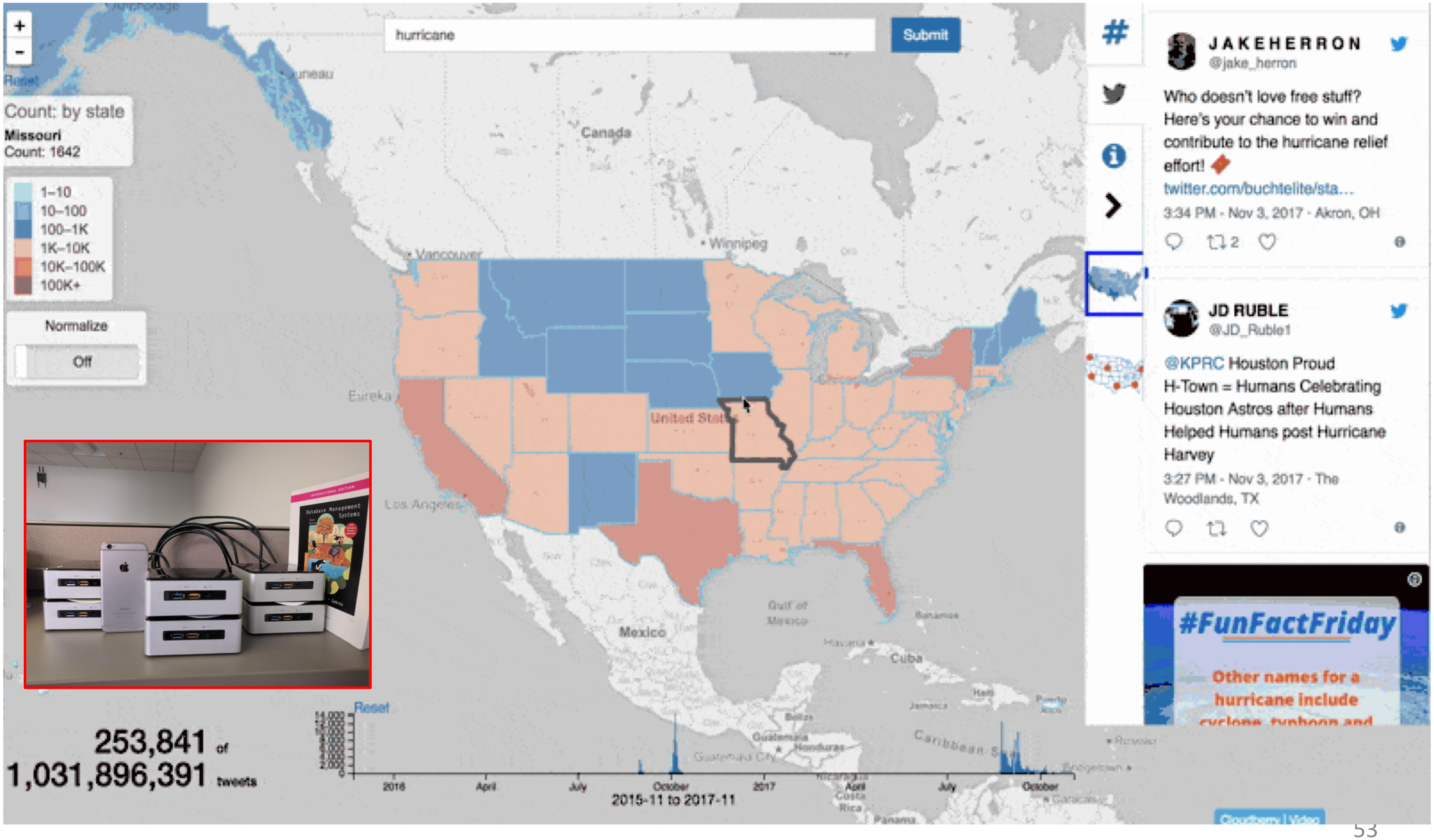
Briefly: AsterixDB Meets Data Science



Issues of current interest include:

- Handling of expensive UDFs
- Python UDFs and Pandas-like DF support
- *ML*: Weka, Scikit-learn, Spark, TensorFlow, …
- Underlying support for Big Data visualization

Briefly: Big Data Visualization



JAKEHERRON
@jake_herron

Who doesn't love free stuff? Here's your chance to win and contribute to the hurricane relief effort! 🔶 twitter.com/buchtelite/sta…

3:34 PM - Nov 3, 2017 - Akron, OH

2

JD RUBLE
@JD_Ruble1

@KPRC Houston Proud H-Town = Humans Celebrating Houston Astros after Humans Helped Humans post Hurricane Harvey

3:27 PM - Nov 3, 2017 - The Woodlands, TX

#FunFactFriday

Other names for a hurricane include cyclone, typhoon and

Questions…?