Big Data, Deep Learning and Other Allegories: Scalability and Fault-tolerance of Parallel and Distributed Infrastructures

Divy Agrawal

Professor of Computer Science
UC Santa Barbara

Visiting Scientist, Ads Data Infrastructure
Google Inc.

Research Director, Data Analytics
Qatar Computing Research Institute

With: Sanjay Chawla et al. (QCRI), Amr El Abbadi et al. (UCSB), &
Shiv Venkataraman et al. (Google)

Motivation

- Availability of vast amounts of data:
  - Hundreds of billions of text documents
  - Billions of images/videos with descriptive annotations
  - Tens of trillions of log records capturing human activity

- Machine Learning + Big Data transforming fiction into reality:
  - Self-driven automobiles
  - Automated image understanding
  - And most recently, deep learning to simulate a human brain

Big Data Challenges

	X_1	X_2	…	X_d
x_1	x_{11}	x_{12}	…	x_{1d}
x_2	x_{21}	x_{22}	…	x_{2d}
⋮	⋮	⋮	⋱	⋮
x_n	x_{n1}	x_{n2}	…	x_{nd}

Statistical Hardness

$$f : \{C\} \rightarrow 2^{\{C\}}$$

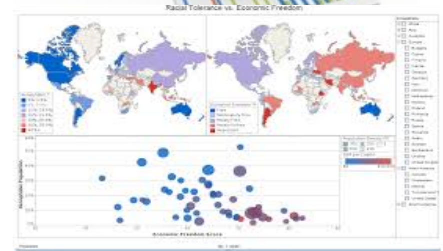
Computational Complexity

$$T : S \times S \rightarrow_{dup} \{0,1\}$$

Data Analytics, Data Mining, and Machine Learning

- **Data:** “The apple of my eye is hooked on Apple’s smart phone and loves apple and yogurt.”
- **Database Query:** how many times does **apple** appear in the data?
- **Data Mining Query**: what are the most frequent items that appear together in the data?
- **Machine Learning:** how many time does the **fruit:<apple>** appear in the data?

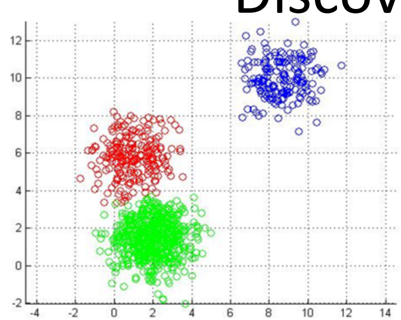
Analysis
Reporting

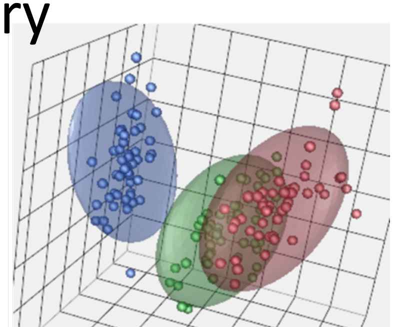


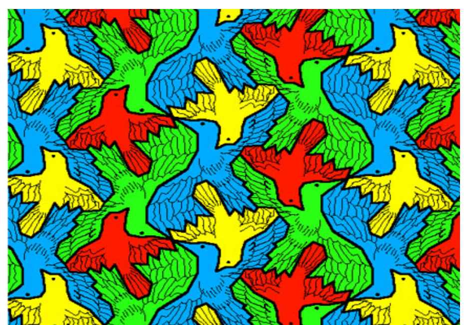
Learning

	X_1	X_2	…	X_d
x_1	x_{11}	x_{12}	…	x_{1d}
x_2	x_{21}	x_{22}	…	x_{2d}
⋮	⋮	⋮	⋱	⋮
x_n	x_{n1}	x_{n2}	…	x_{nd}

Discovery



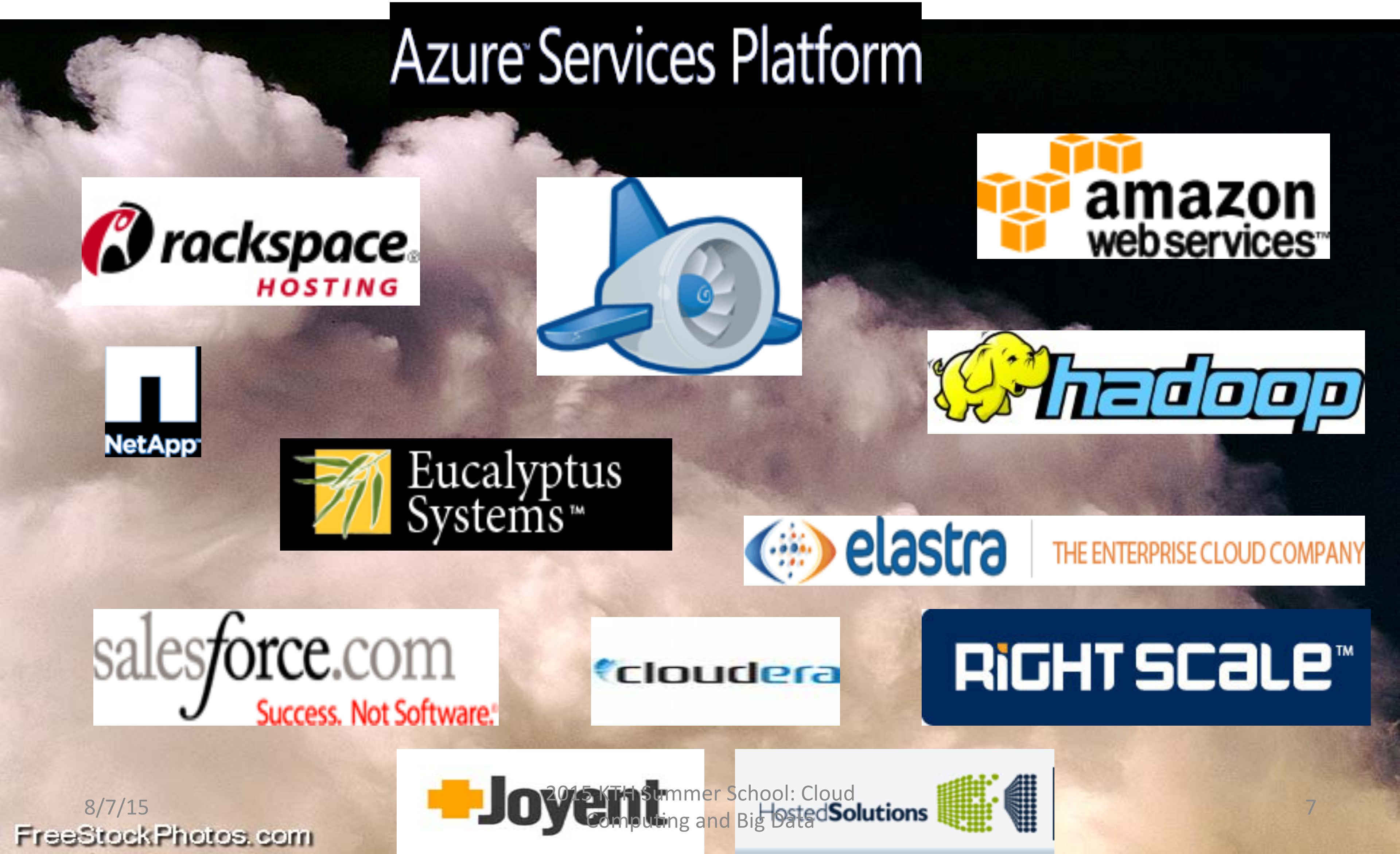




	X_1	X_2	…	X_d
x_1	x_{11}	x_{12}	…	x_{1d}
x_2	x_{21}	x_{22}	…	x_{2d}
⋮	⋮	⋮	⋱	⋮
x_n	x_{n1}	x_{n2}	…	x_{nd}

BIG DATA MANAGEMENT (UCSB)

Paradigm Shift in Computing

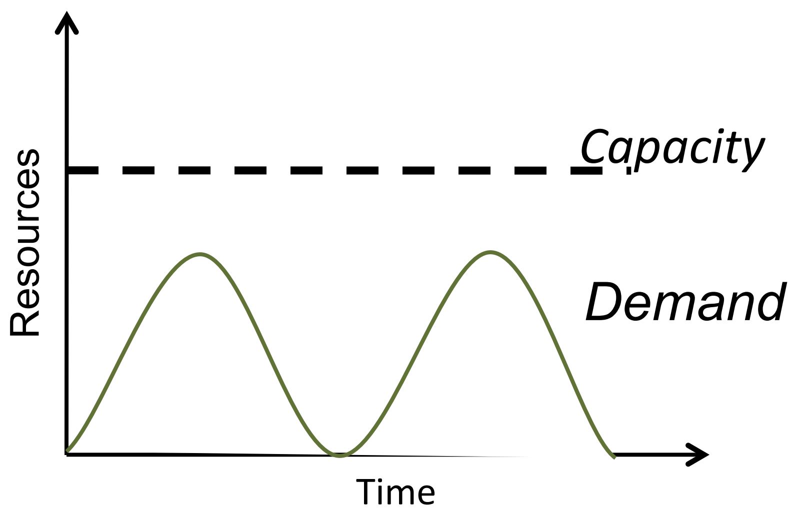


Cloud Computing: Why?

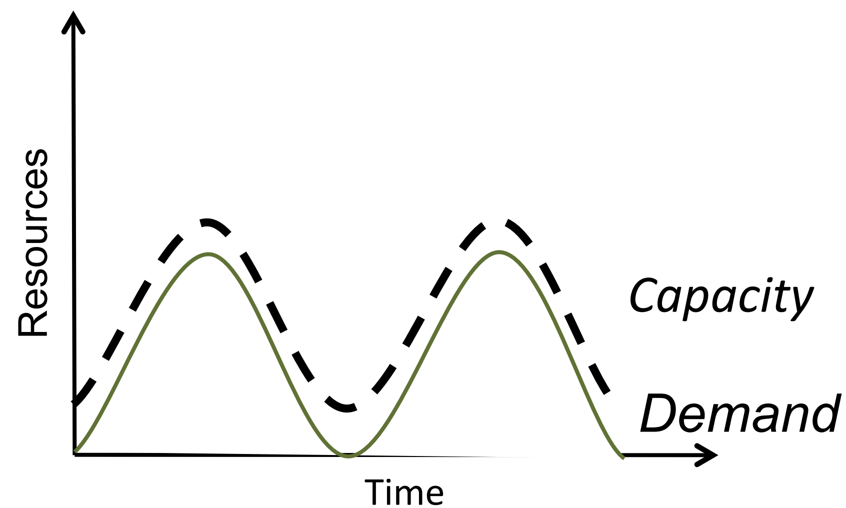
- Experience with very large datacenters
  - Unprecedented economies of scale
  - Transfer of risk
- Technology factors
  - Pervasive broadband Internet
  - Maturity in Virtualization Technology
- Business factors
  - Minimal capital expenditure
  - Pay-as-you-go billing model

Economics of Cloud Computing

- Pay by use instead of provisioning for peak

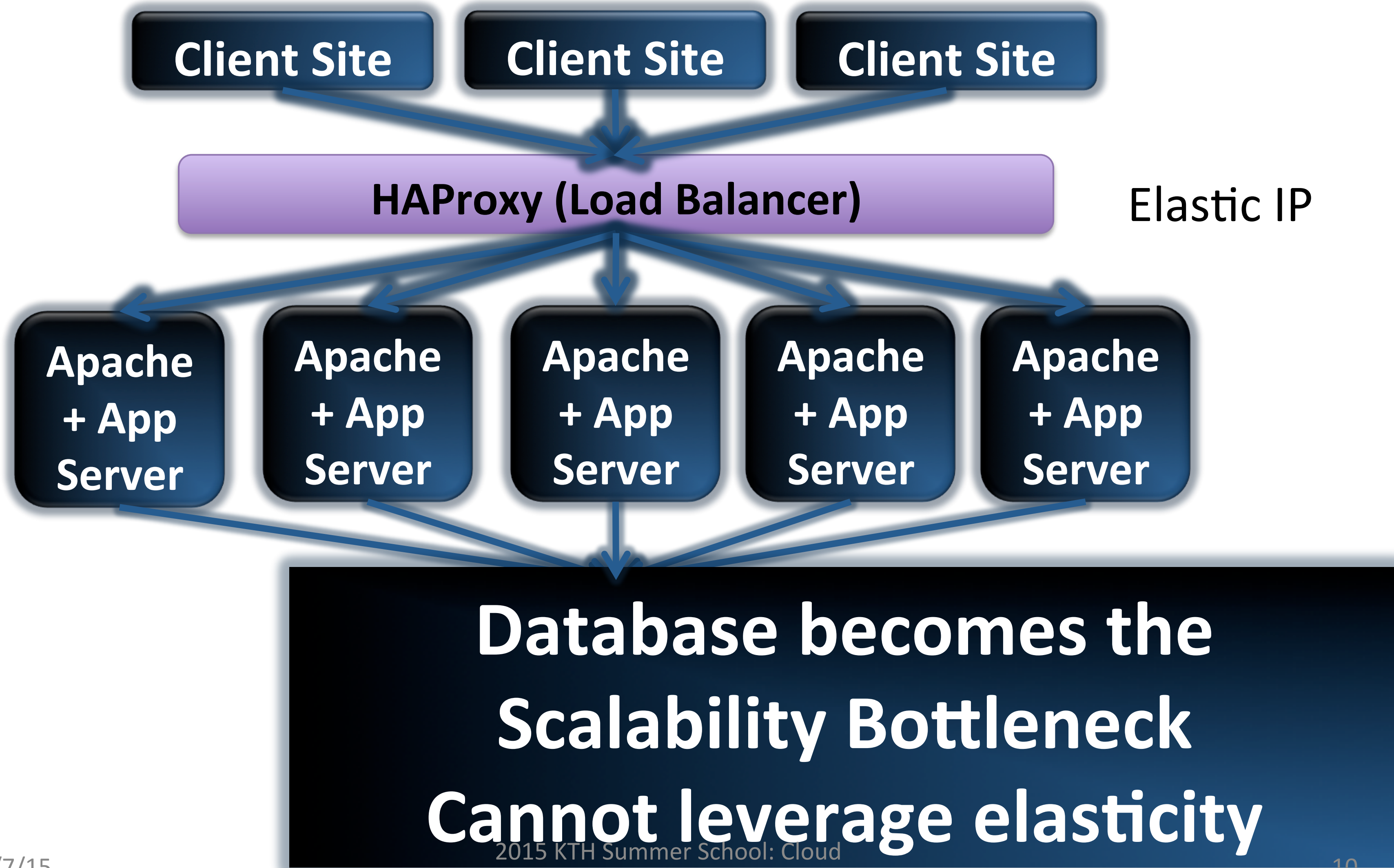


Static data center

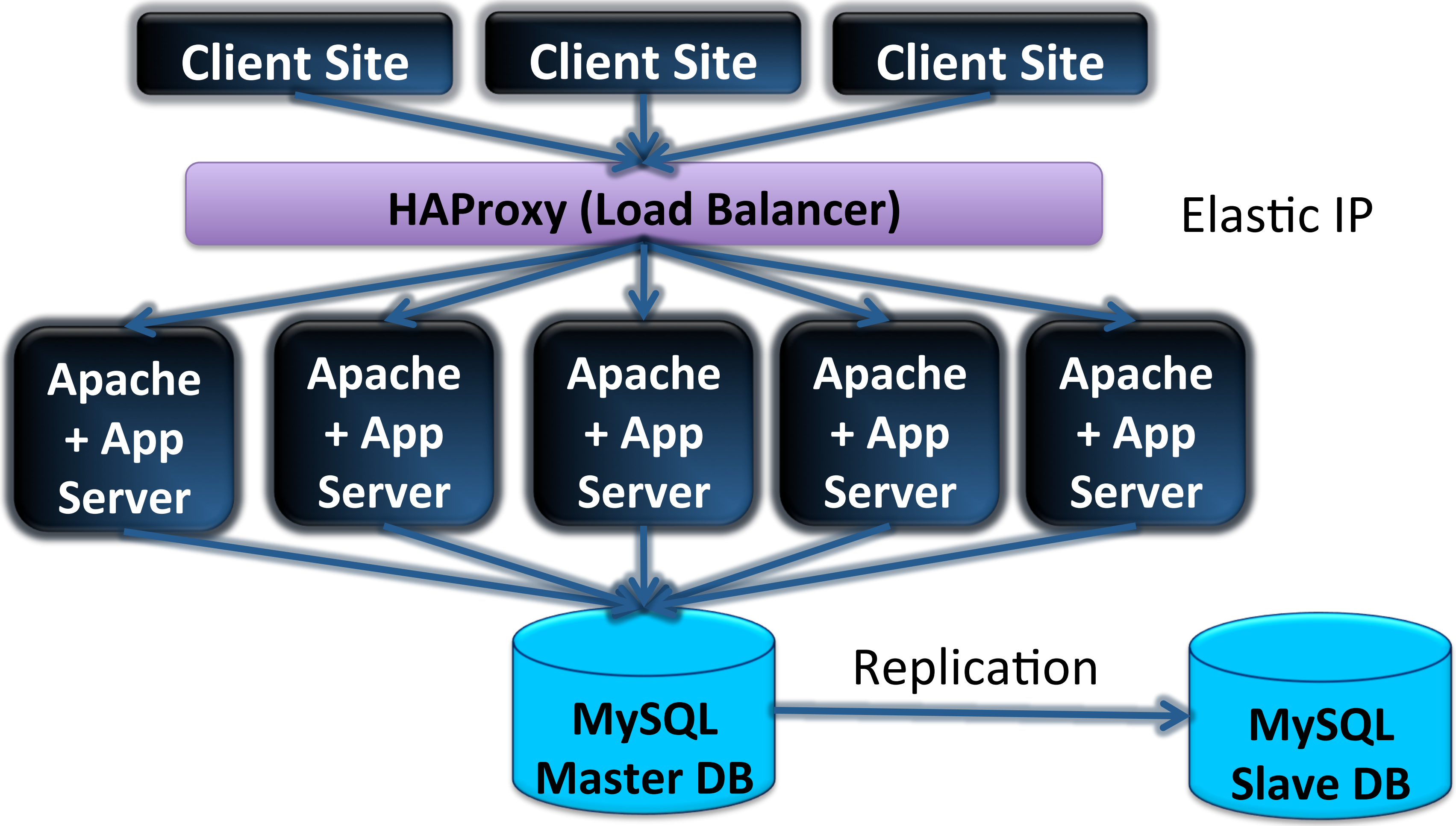


Data center in the cloud

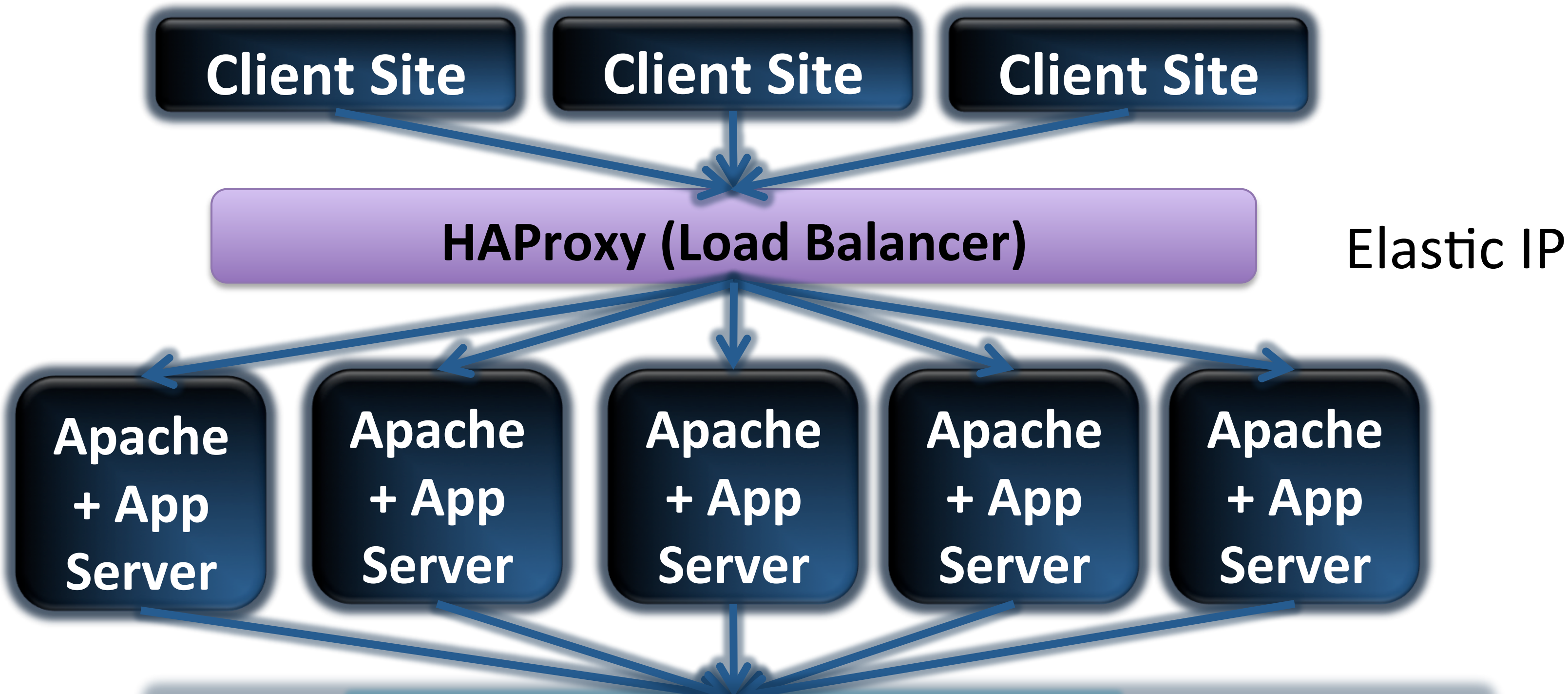
Scaling in the Cloud



Scaling in the Cloud



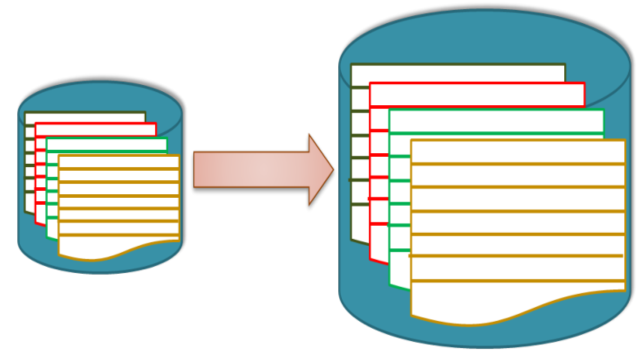
Scaling in the Cloud

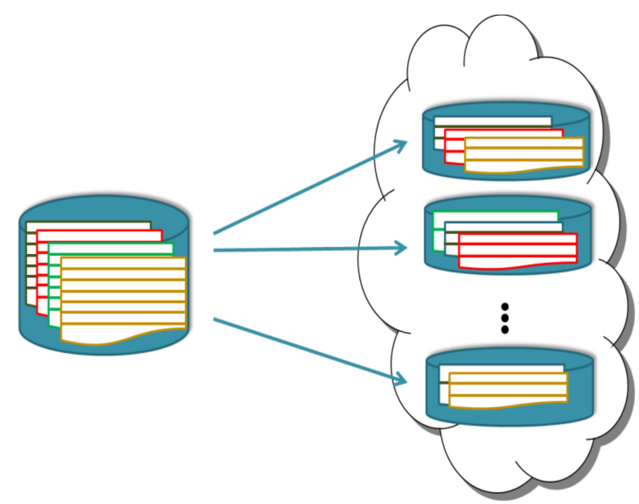


Scalable and Elastic
But limited consistency and operational flexibility

Two approaches to scalability

- **Scale-up**
  - **Classical enterprise** setting (RDBMS)
  - Flexible **ACID transactions**
  - Transactions in a single node
- **Scale-out**
  - **Cloud friendly** (Key value stores)
  - Execution at a single server
    - Limited functionality & guarantees
  - No **multi-row** or **multi-step** transactions

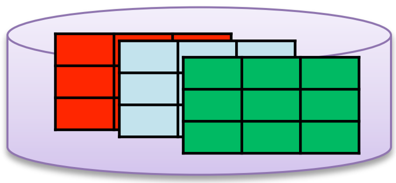




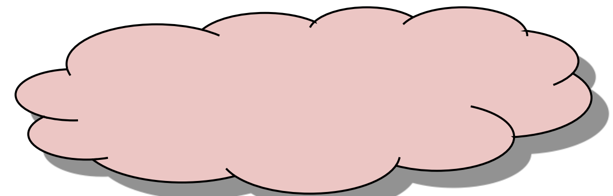
Key-value Stores: Design Principles

- **Separate System and Application State**
- **Limit Application interactions to a single node**
- **Decouple Ownership from Data Storage**
- **Limited distributed synchronization is practical**

Scalable Data Managementin the Cloud



RDBMS



Key Value Stores

Fission

Fusion

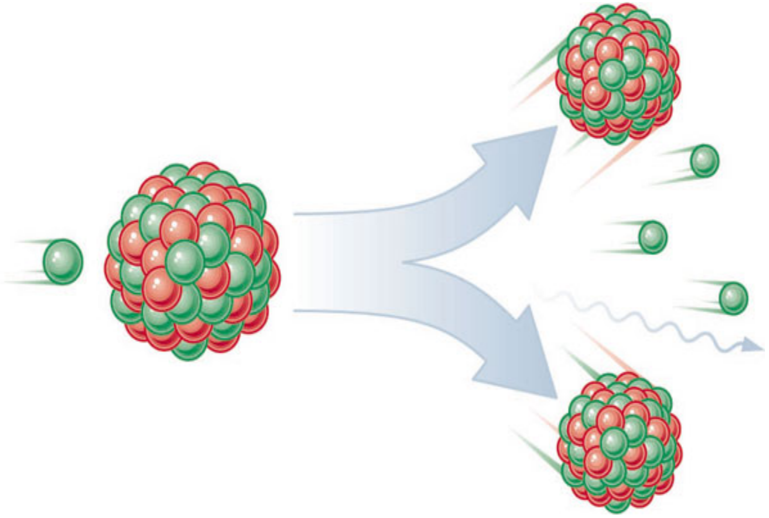
ElasTraS [HotCloud '09,TODS'13]
Cloud SQL Server [ICDE '11]
RelationalCloud [CIDR '11]
Google F1 (SIGMOD'12, VLDB'13)

G-Store [SoCC '10]
MegaStore [CIDR '11]
ecStore [VLDB '10]

Data Fission

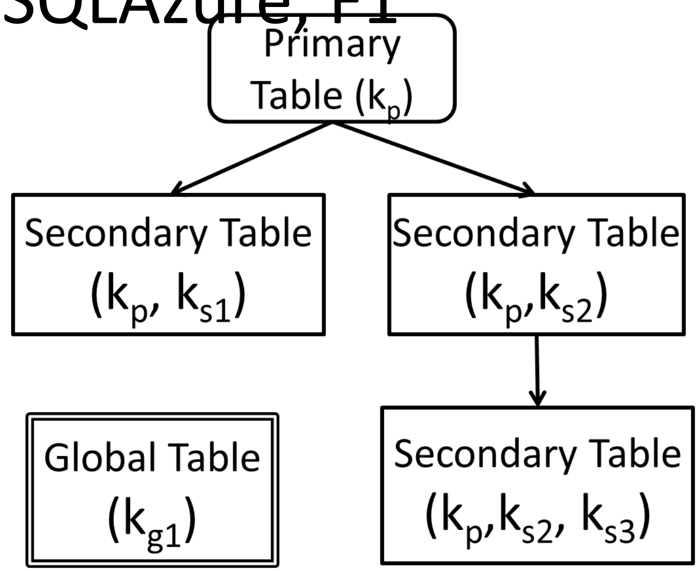
- Basic building-block:
  - Data Partitioning (Table level → Distributed Transactions)

- Three Example Systems
  - ElasTraS (UCSB)
  - SQL Azure (MSR)
  - Relational Cloud (MIT)

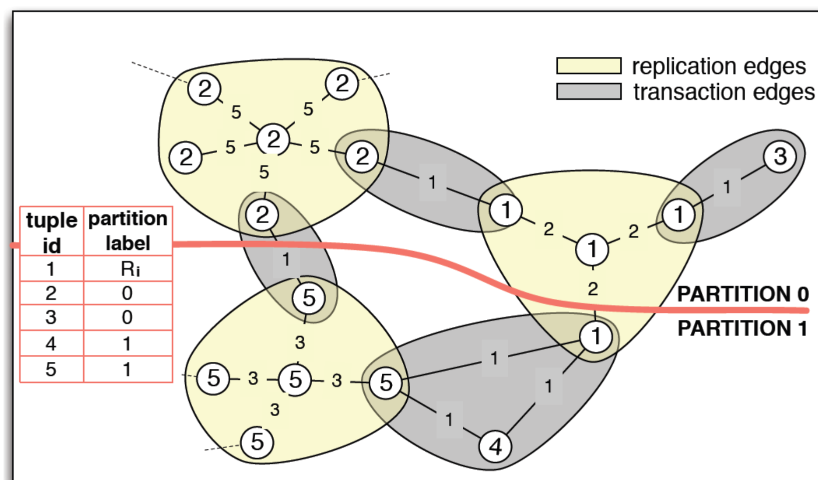


Schema Level Partitioning

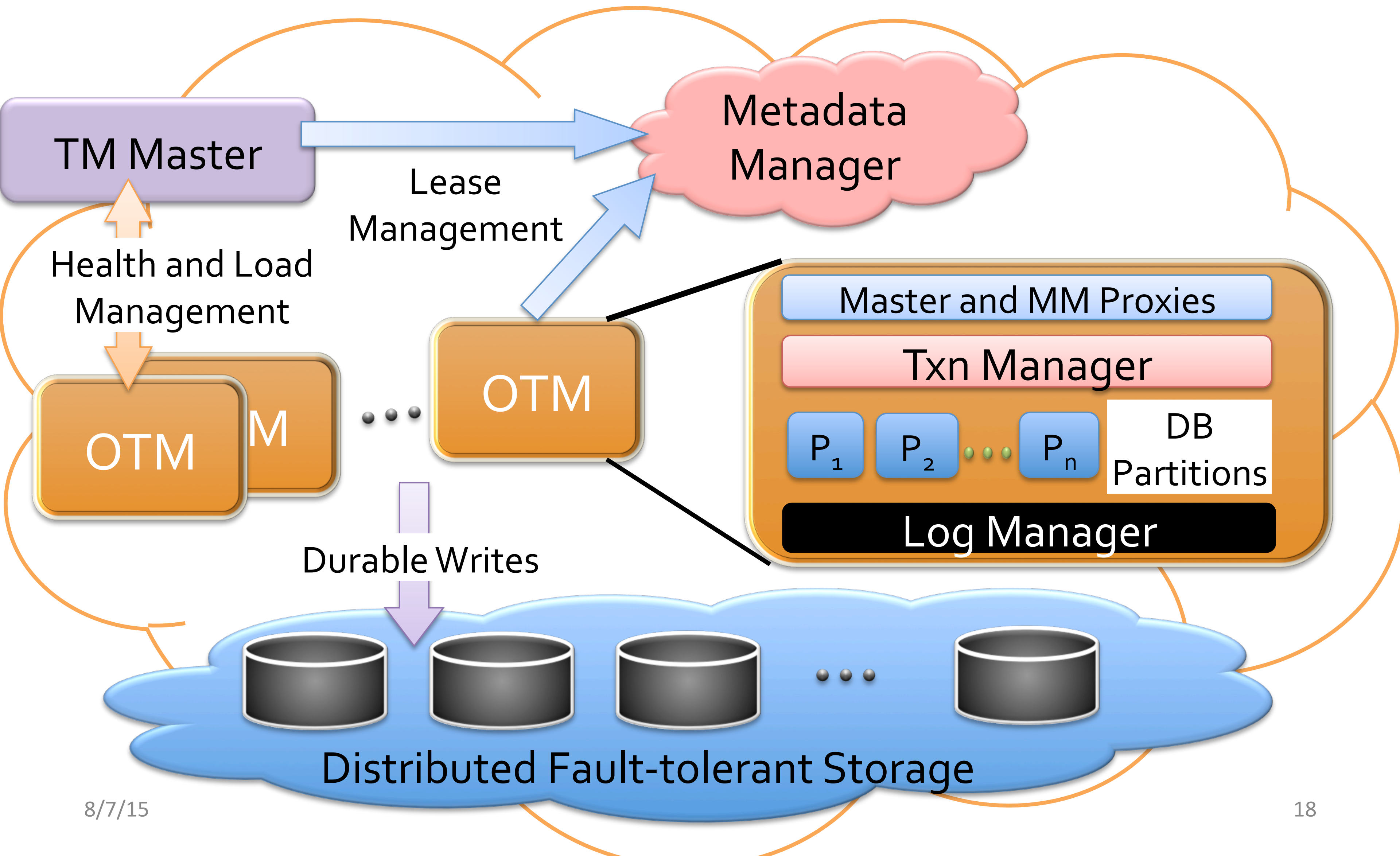
- **Pre-defined** partitioning scheme
  – e.g.: Tree schema
  – ElasTras, SQLAzure, F1



- **Workload driven** partitioning scheme
  – e.g.: Schism in RelationalCloud

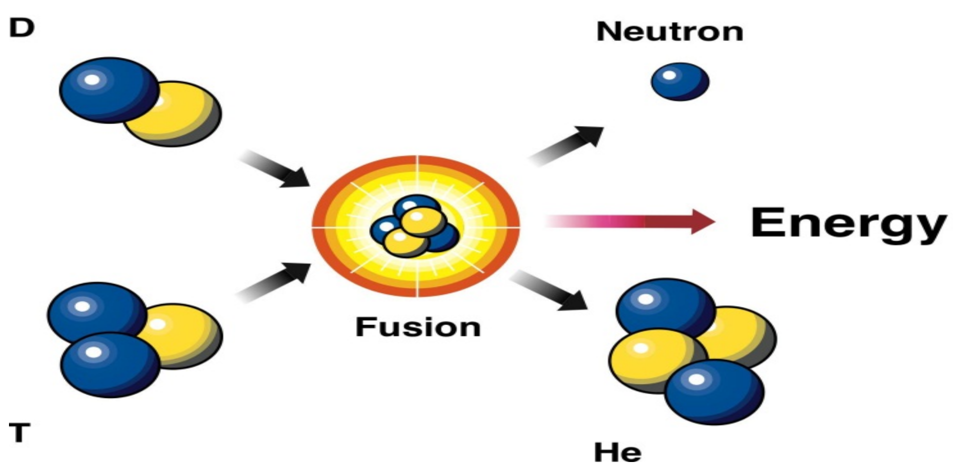


ElasTraS Architecture



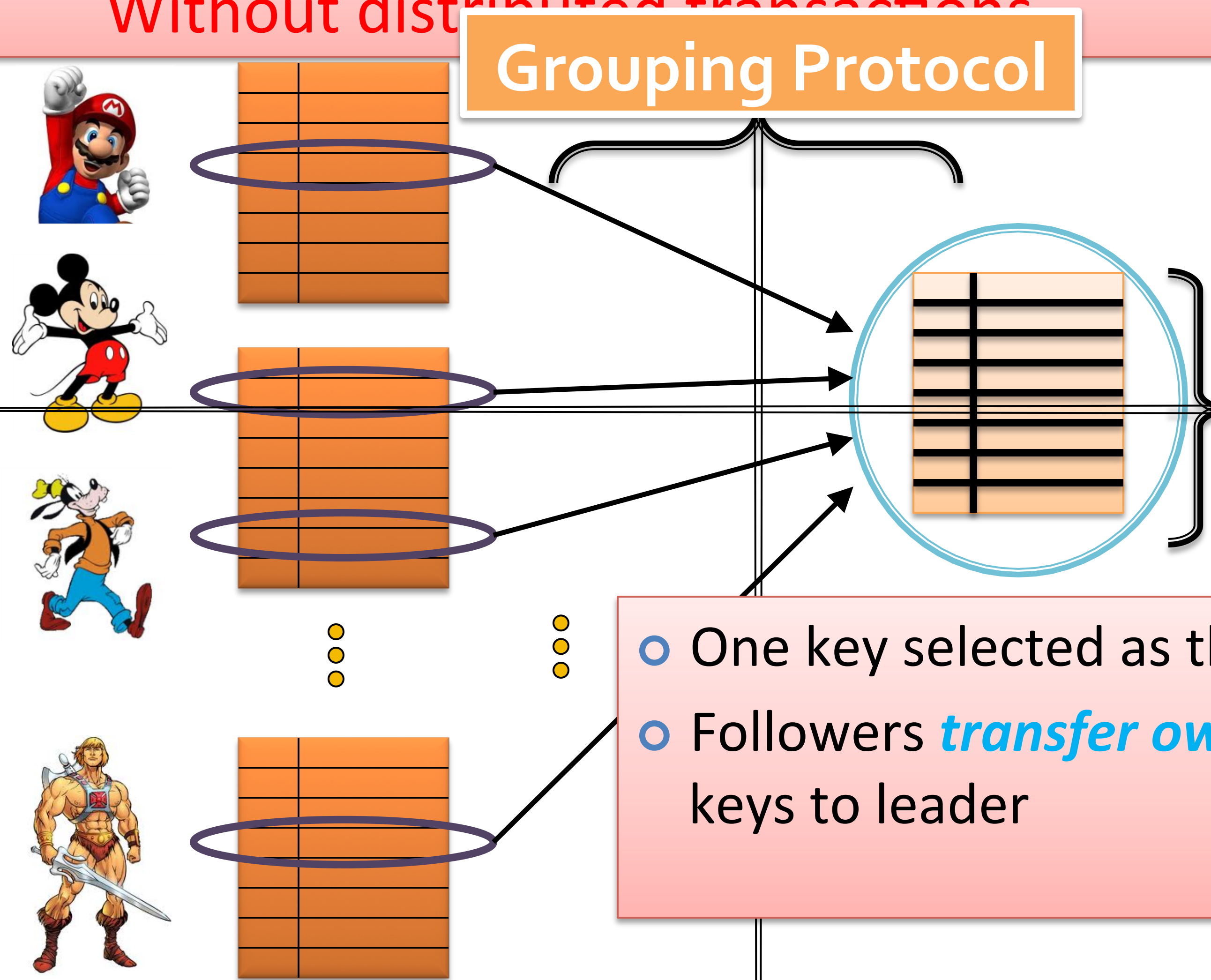
Data Fusion

- Key value: Atomicity guarantee on single keys
- Combining the individual key-value pairs into larger granules of transactional access
- Megastore: Statically defined



G-Store: Transactions on Groups

Without distributed transactions



Ownership of keys at a single node

- One key selected as the **leader**
- Followers ***transfer ownership*** of keys to leader

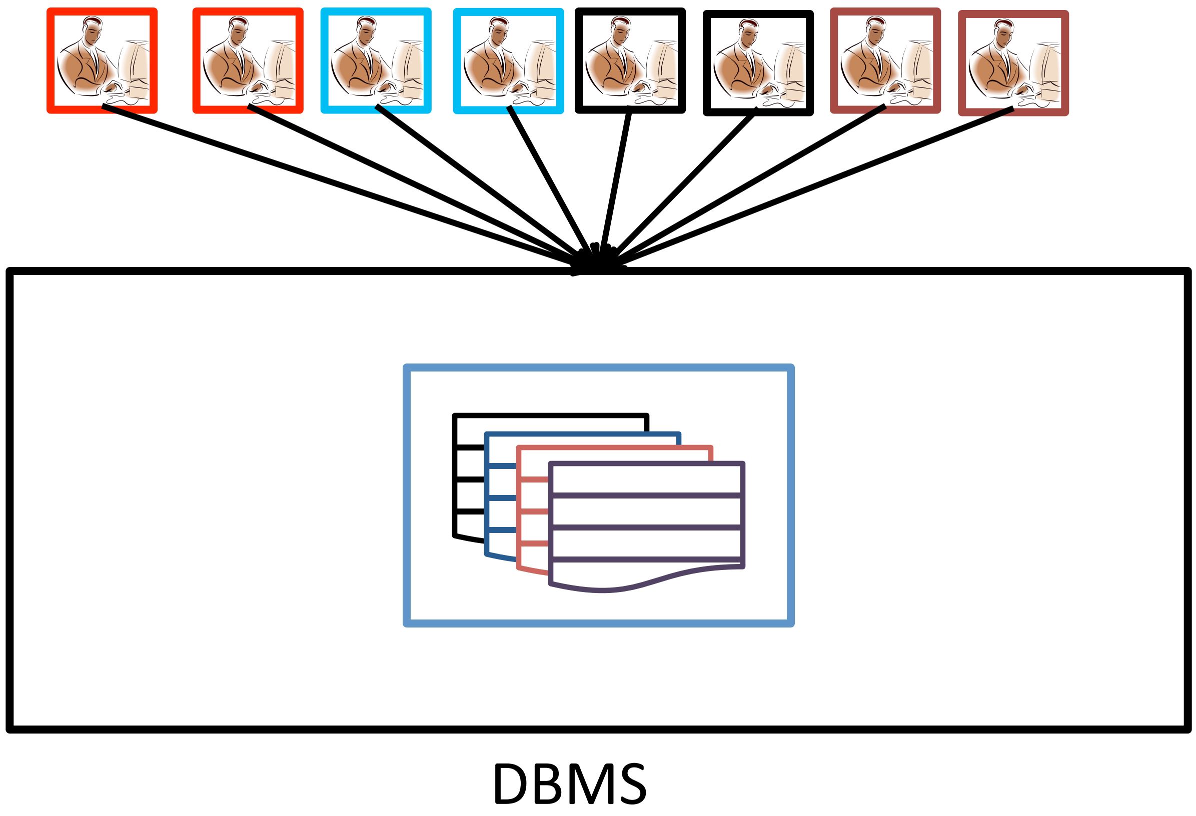
Elasticity

- A database system built over a pay-per-use infrastructure
  – Infrastructure as a Service for instance

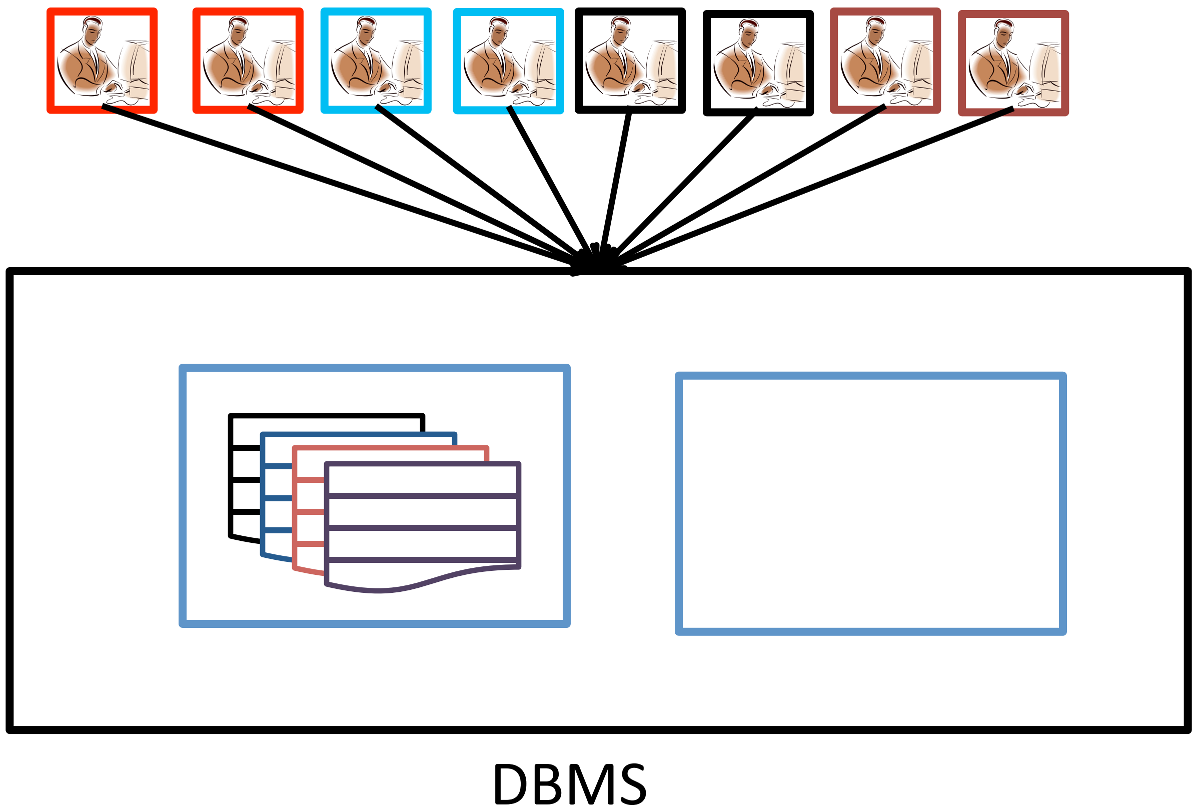
- Scale up and down system size on demand
  – Utilize peaks and troughs in load

- Minimize operating cost while ensuring good performance

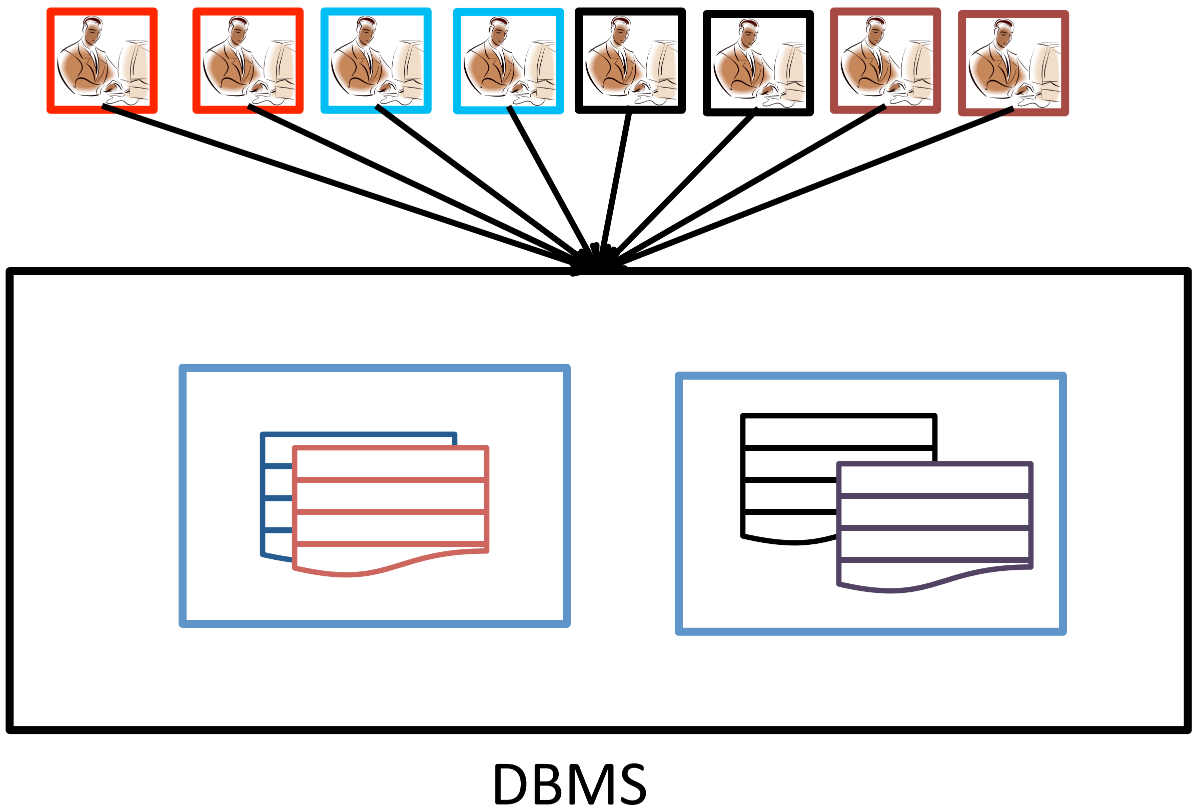
Elasticity in the Database Layer



Elasticity in the Database Layer



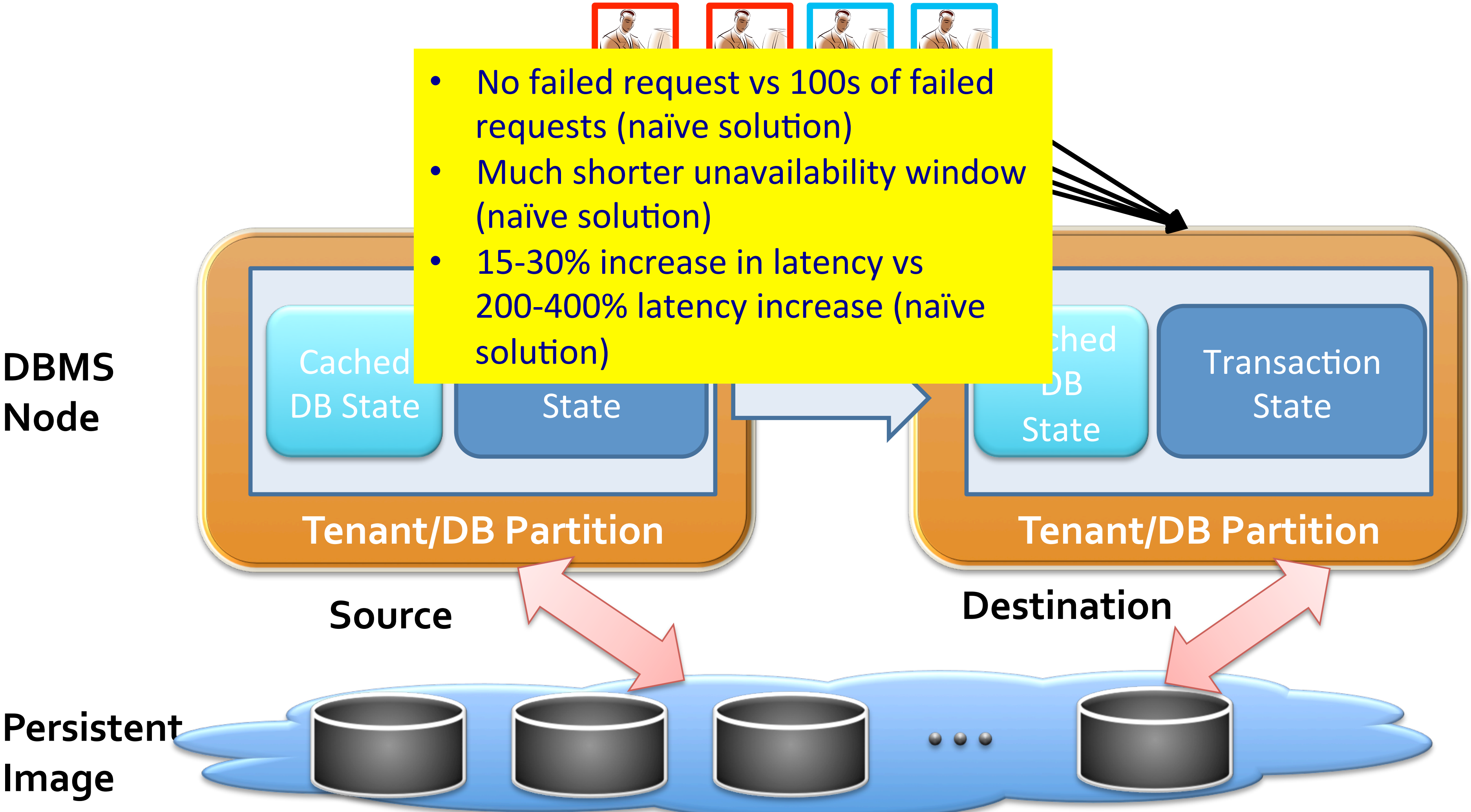
Elasticity in the Database Layer



Live Database Migration

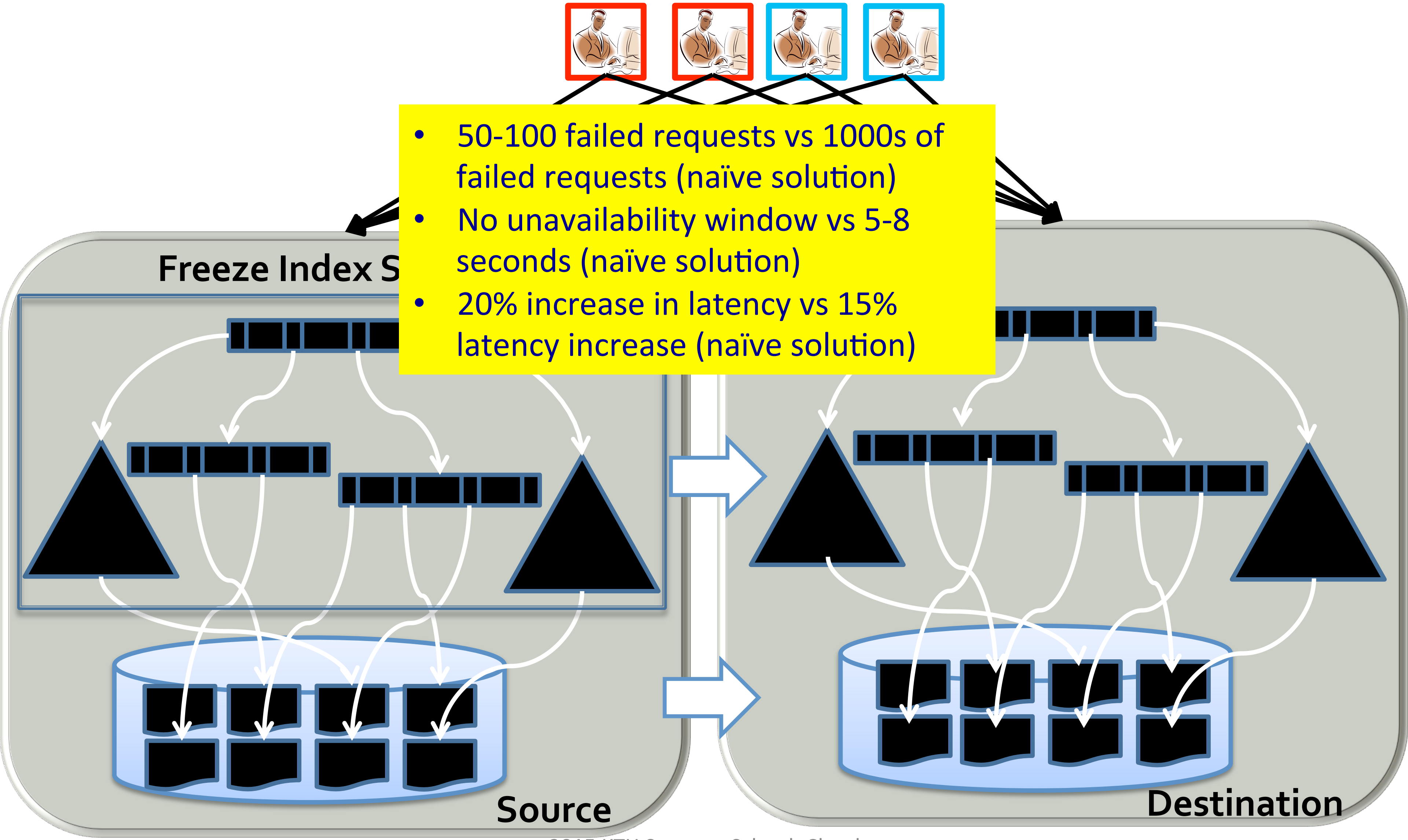
- No prior work on **Database migration**
- State-of-the-art – use VM migration
  - [Clark et al., NSDI 2005], [Liu et al., HPDC 2009]
- Requires executing DB-in-VM
  - High performance overhead
  - Poor performance and consolidation ratio [Curino et al., CIDR 2011]

Shared Disk Architecture: Albatross



- No failed request vs 100s of failed requests (naïve solution)
- Much shorter unavailability window (naïve solution)
- 15-30% increase in latency vs 200-400% latency increase (naïve solution)

Shared-Nothing Architecture: Zephyr

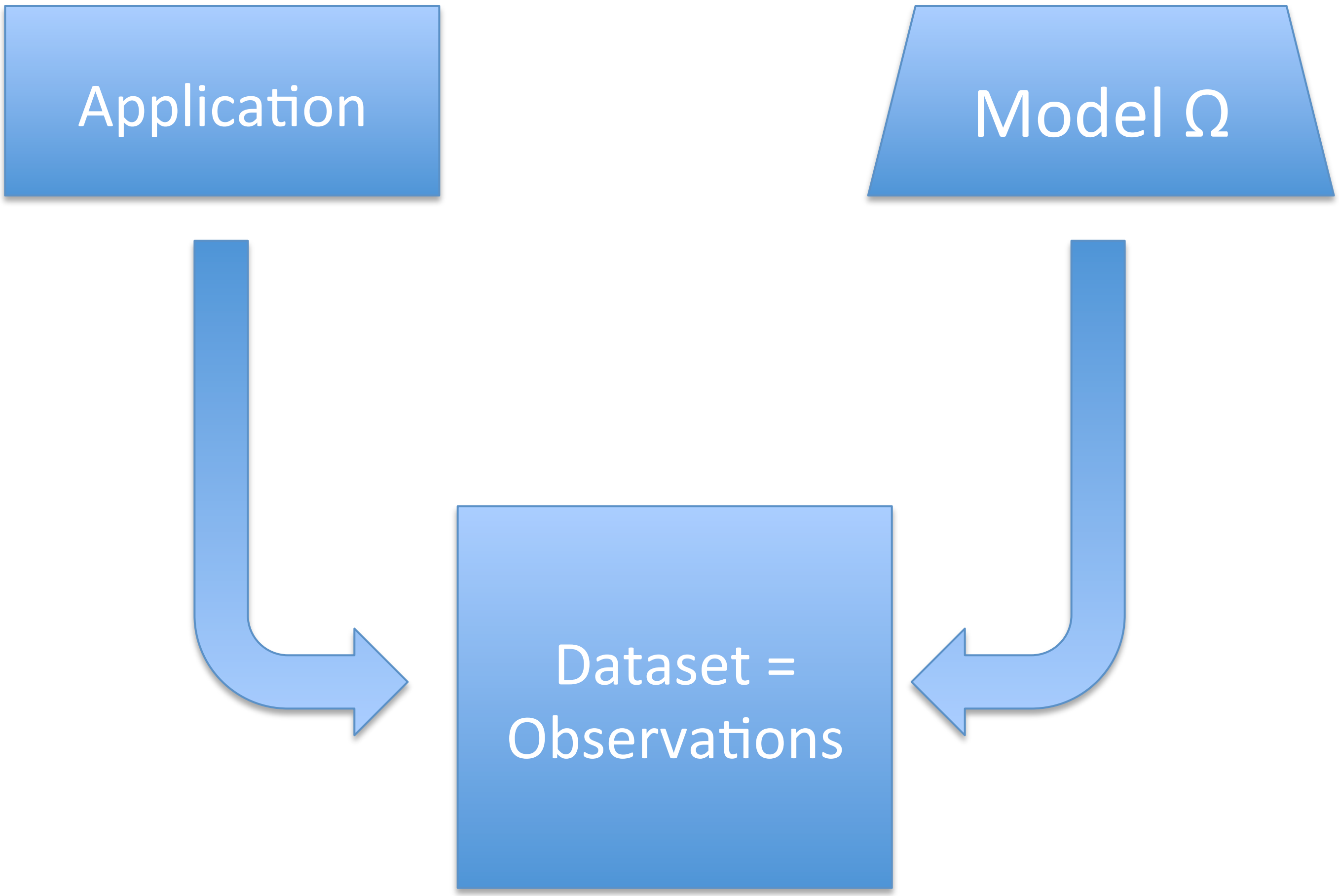


- 50-100 failed requests vs 1000s of failed requests (naïve solution)
- No unavailability window vs 5-8 seconds (naïve solution)
- 20% increase in latency vs 15% latency increase (naïve solution)

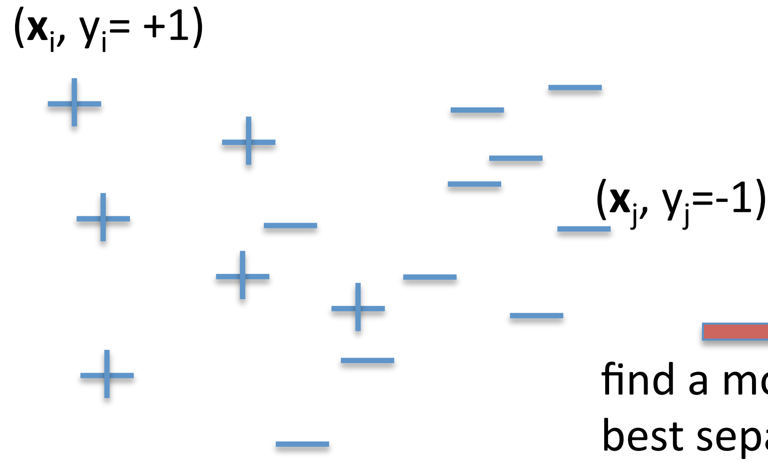
	X_1	X_2	…	X_d
x_1	x_{11}	x_{12}	…	x_{1d}
x_2	x_{21}	x_{22}	…	x_{2d}
⋮	⋮	⋮	⋱	⋮
x_n	x_{n1}	x_{n2}	…	x_{nd}

BIG DATA ANALYTICS (QCRI): LEARNING

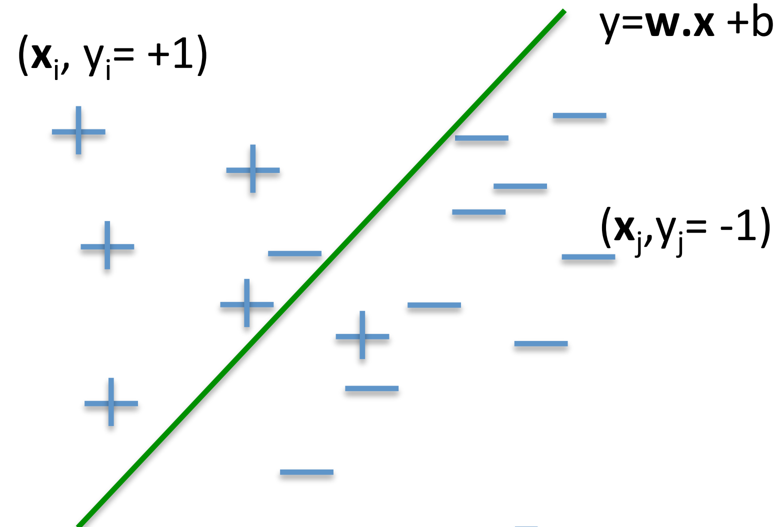
Learning: Model Fitting



Machine Learning in One Slide



find a model that
best separates:
+ and -



Big Data Setting

$$w^* = \min_w \sum_{i=1}^{n=10^{12}} \ell(x_i, y_i, w) + \Gamma(w)$$

dimension 10^9

$$w^* = \min_w \sum_{i=1}^{n} \ell(x_i, y_i, w) + \Gamma(w)$$

Loss function

Regularization

Linear Regression Example

- For a given training data with features x_1, *and* x_2, we model the dependent variable y as a hypothesis function:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- With a training data of size m, minimize a cost function:

$$\mathrm{J}(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

- Iteratively, compute the gradient and update θ_j

$$\theta_j \leftarrow \theta_j - \alpha\frac{\partial}{\partial\theta_j}\mathrm{J}(\theta)$$

Gradient Descent: Sequential Computation

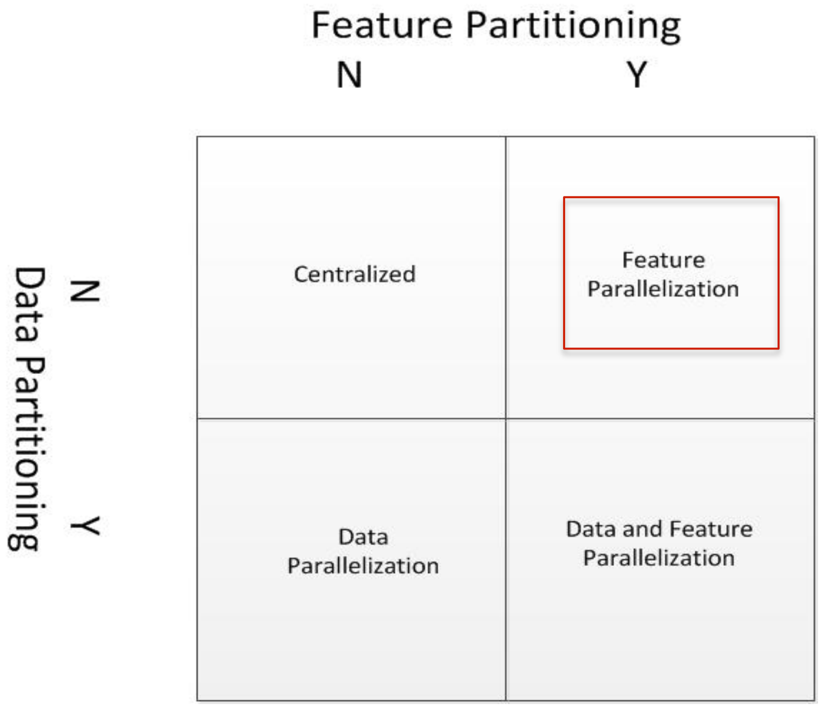
Repeat until convergence
{
    read ($\theta_0, \theta_1, \ldots, \theta_m$);

    Compute gradient;

    write ($\theta_0', \theta_1', \ldots, \theta_m'$);
}

Scaling Machine Learning Algorithms

- Leverage data and feature partitioning to parallelize the computations.



Parallel Version

π_1 π_2 .... π_d

Worker i (at iteration α):

read synchronization

read (π_1, π_2, …, π_d);

Compute gradient projection;

write synchronization

write (π_i');

Worker j (at iteration α):

read synchronization

read (π_1, π_2, …, π_d);

Compute gradient projection;

write synchronization

write (π_j');

Parallel Version

π_1 π_2 …. π_d

Worker i (at iteration α):

$\forall i,j,k \;\; wk[\pi k][\alpha-1] < ri[\pi j][\alpha]$

read (π_1, π_2, …, π_d);

Compute gradient projection;

$\forall i,j,k \;\; rk[\pi j][\alpha] < wi[\pi i][\alpha]$

write (π_i');

Worker j (at iteration α):

$\forall i,j,k \;\; wk[\pi k][\alpha-1] < ri[\pi j][\alpha]$

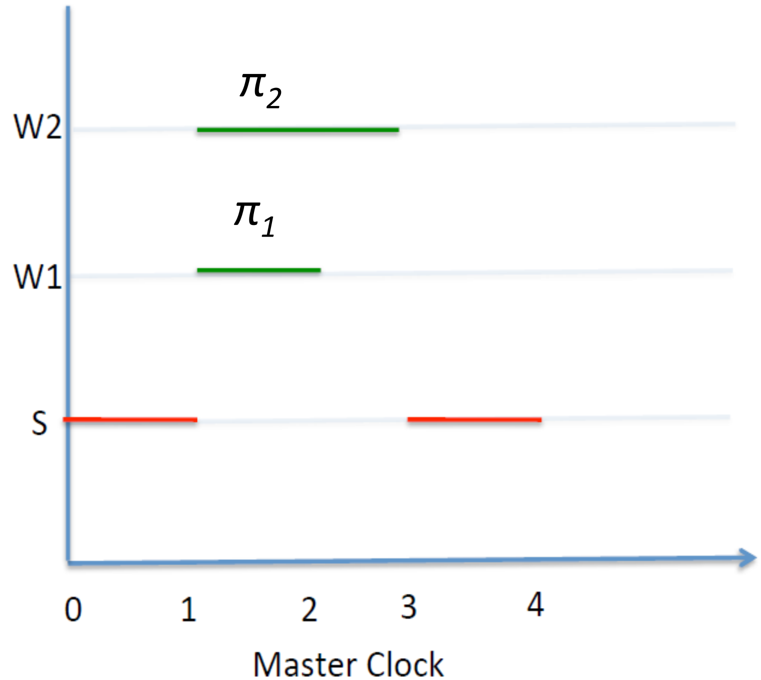
read (π_1, π_2, …, π_d);

Compute gradient projection;

$\forall i,j,k \;\; rk[\pi j][\alpha] < wi[\pi i][\alpha]$

write (π_j');

Straggler or Last Reducer Problem



(a) The Straggler Problem

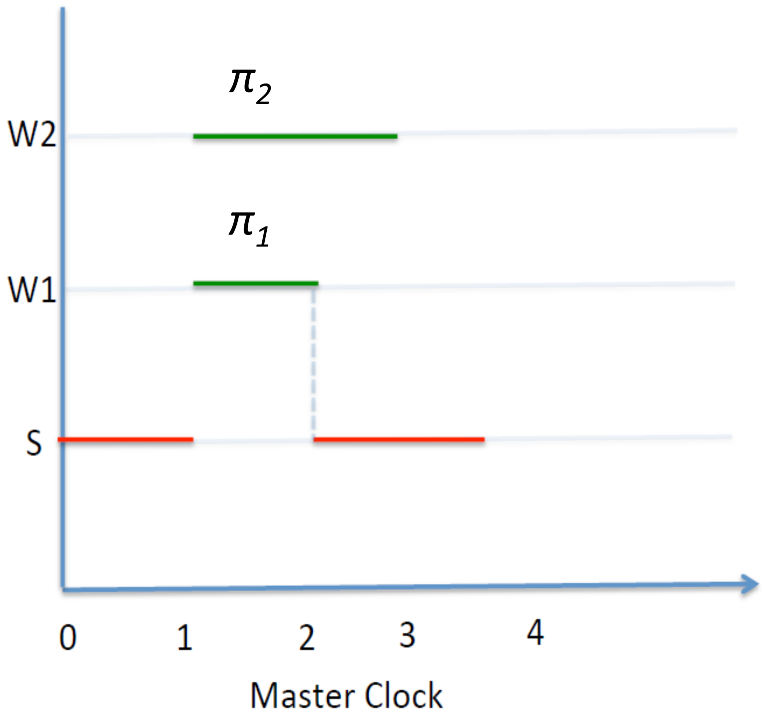
Scaling ML Algorithms

- Current Approaches [e.g., Parameter Server]:
  - Allow synchronization violations albeit bounded
  - Not equivalent to semantics of sequential executions
  - Use function-centric arguments to demonstrate convergence
  - Higher tolerance to imprecision (nature of ML)

- Process-based synchronization ➔ data-centric approaches:
  - Model read and write of parameter variables as database actions
  - 2-phase locking during each iteration for fine-grained concurrency

- Unfortunately, does not work:
  - Need a new framework for data-centric synchronization!

Barrier Relaxation



(b) Barrier Relaxation

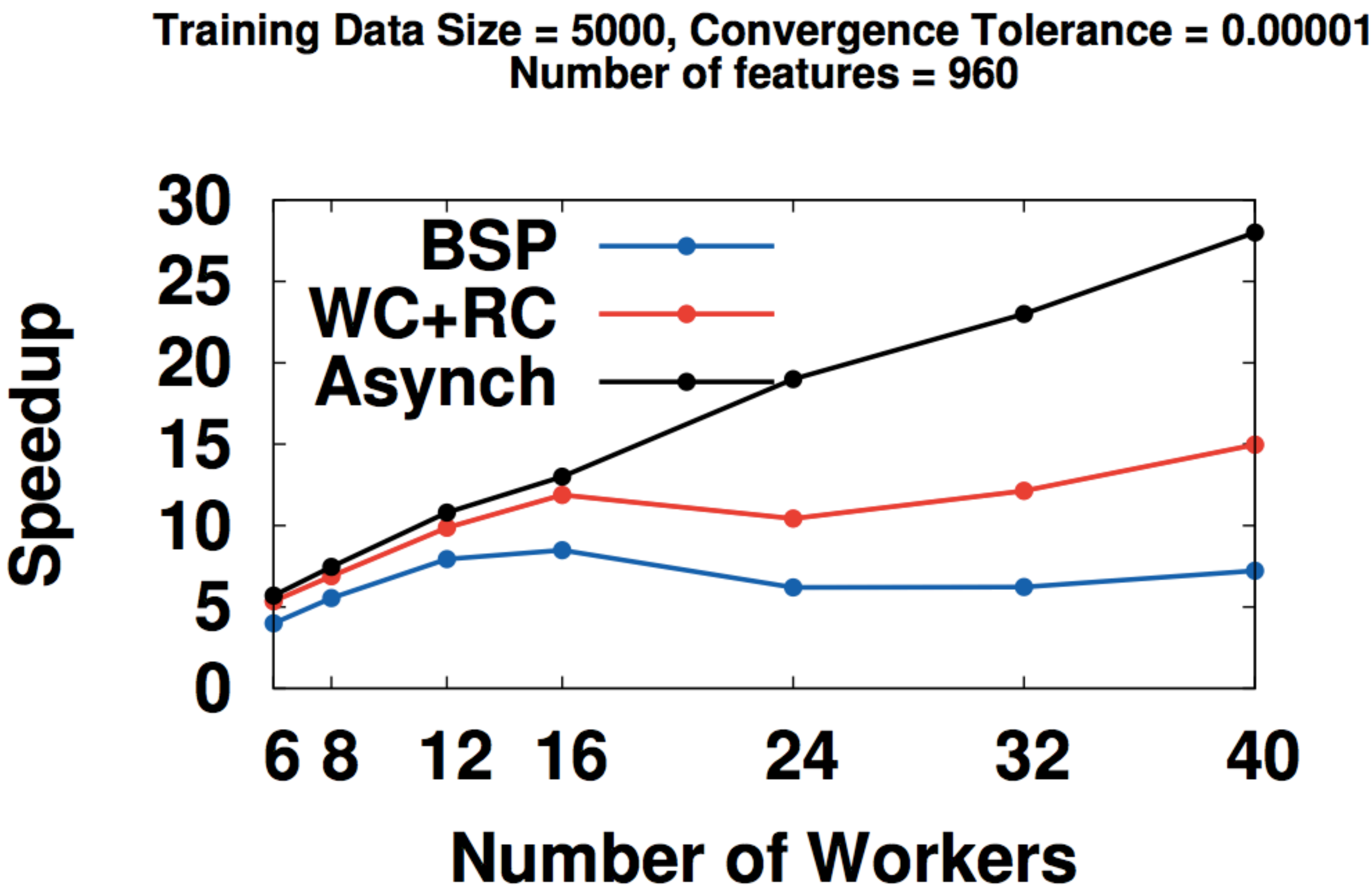
Data-centric View of Synchronization

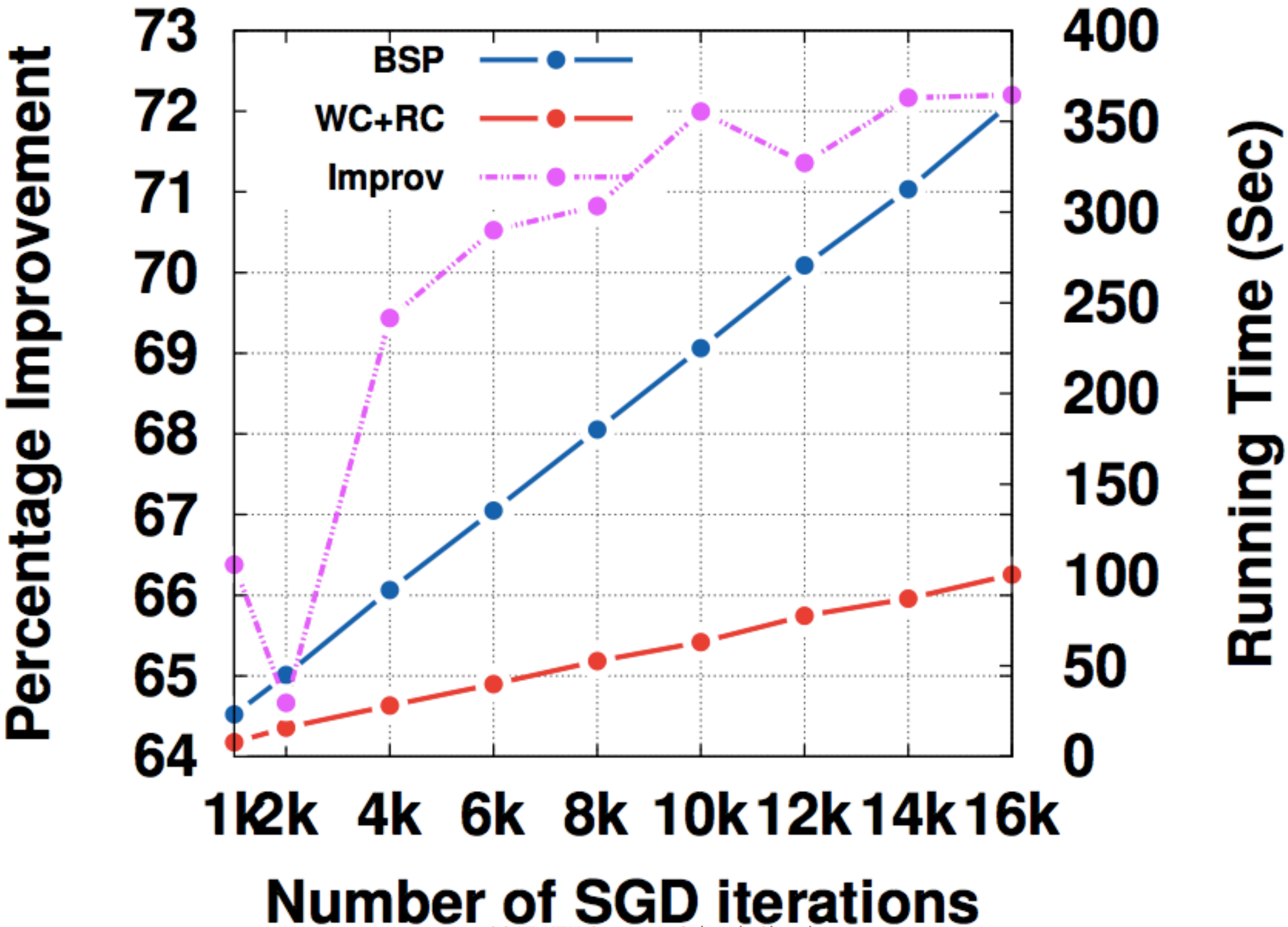
- Read Constraint: *Worker i* can read π_j in iteration α only after worker j has completed writes of iteration $\alpha - 1$.

$$\forall i,j \;\; wj[\pi j][\alpha-1] < ri[\pi j][\alpha]$$

- Write Constraint: *Worker j* can write π_j in iteration α only after all the workers have read it.

$$\forall i,j \;\; ri[\pi j][\alpha] < wj[\pi j][\alpha]$$

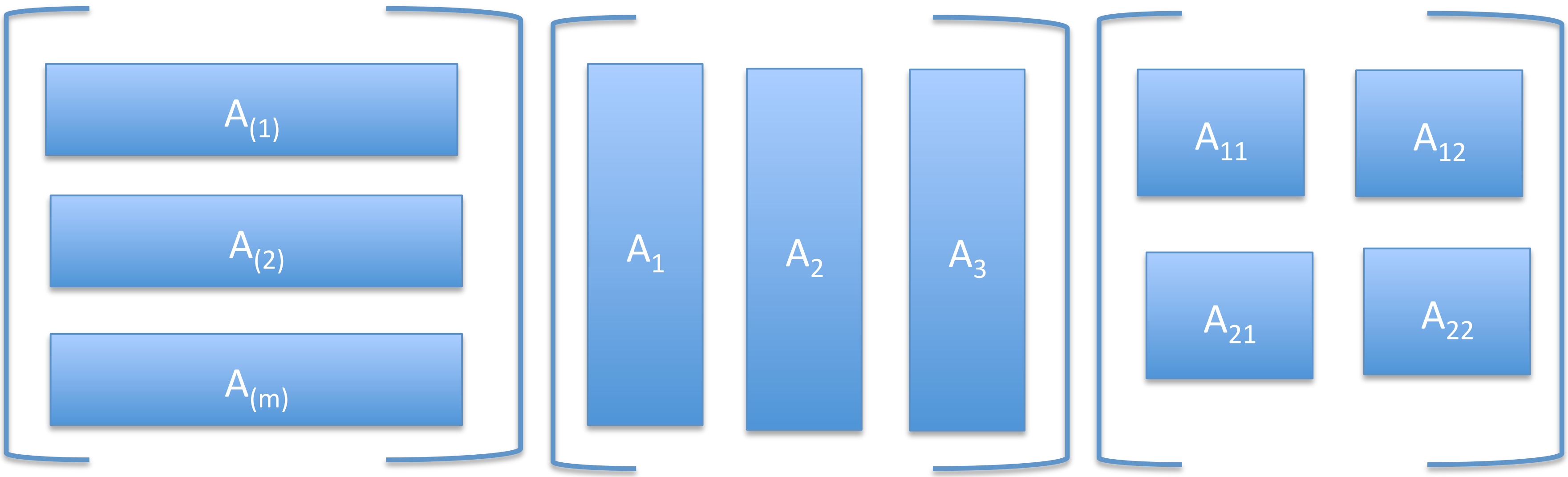

Training Data Size = 5000, Convergence Tolerance = 0.00001
Number of features = 960
BSP
WC+RC
Asynch
Speedup
30
25
20
15
10
5
0
6
8
12
16
24
32
40
Number of Workers


Percentage Improvement
73
72
71
70
69
68
67
66
65
64
BSP
WC+RC
Improv
400
350
300
250
200
150
100
50
0
Running Time (Sec)
1k
2k
4k
6k
8k
10k
12k
14k
16k
Number of SGD iterations

Data Partitioning

Feature Partitioning

Fully Distributed



Row-Oriented

Column-Oriented

Mixed

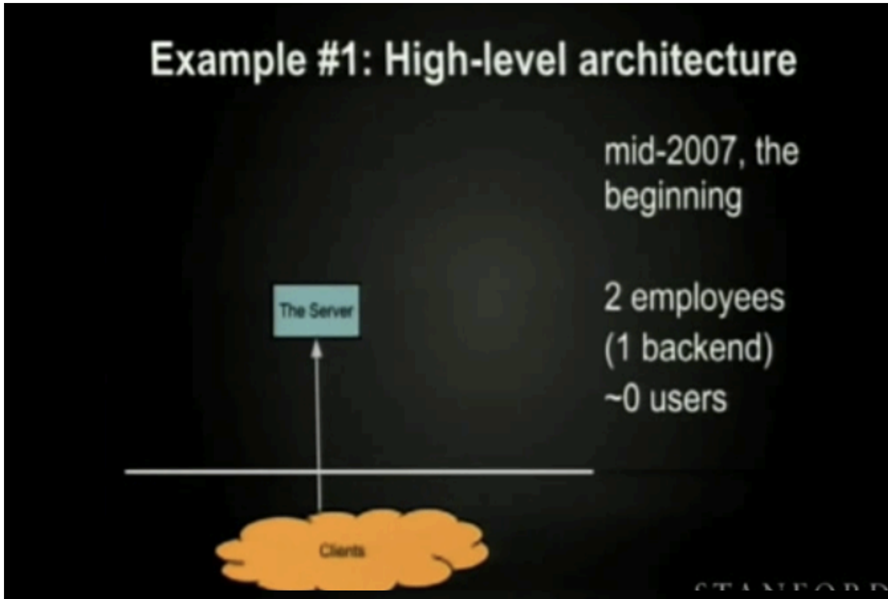
Work-in-progress

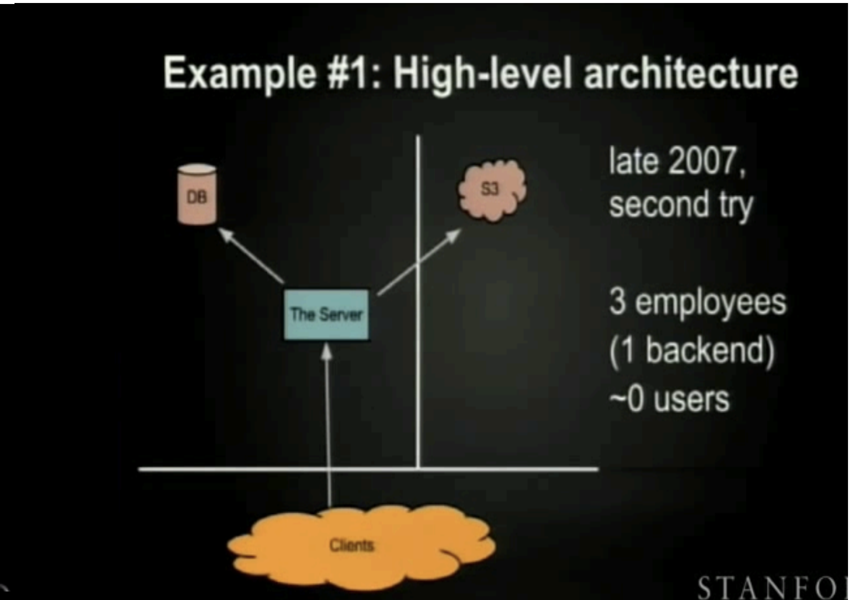
1. Characterization of sequentially consistent executions.
2. Data-centric constraints ➔ sequentially consistent
3. Process-centric synchronization ➔ sequentially consistent
4. Qualitative analysis of different classes of executions
5. Develop protocols that enforce data-centric constraints
6. Experimental evaluation

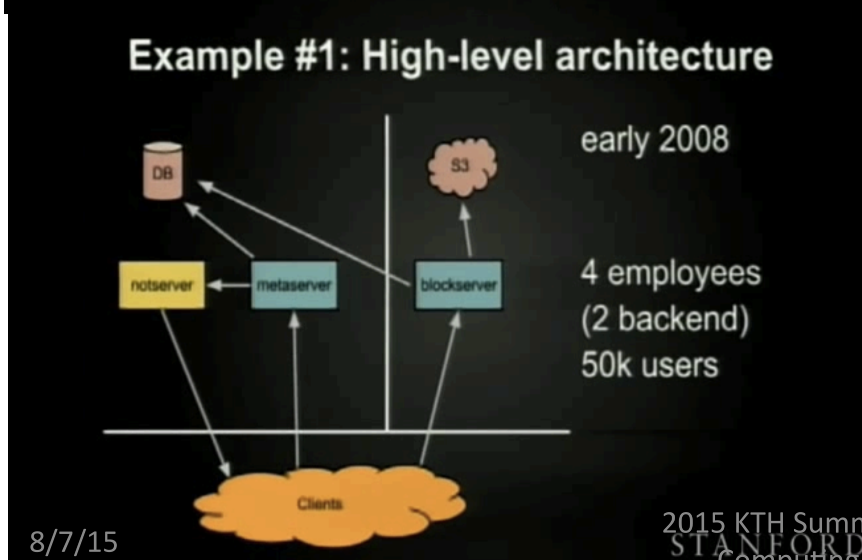
	X_1	X_2	…	X_d
x_1	x_{11}	x_{12}	…	x_{1d}
x_2	x_{21}	x_{22}	…	x_{2d}
⋮	⋮	⋮	⋱	⋮
x_n	x_{n1}	x_{n2}	…	x_{nd}

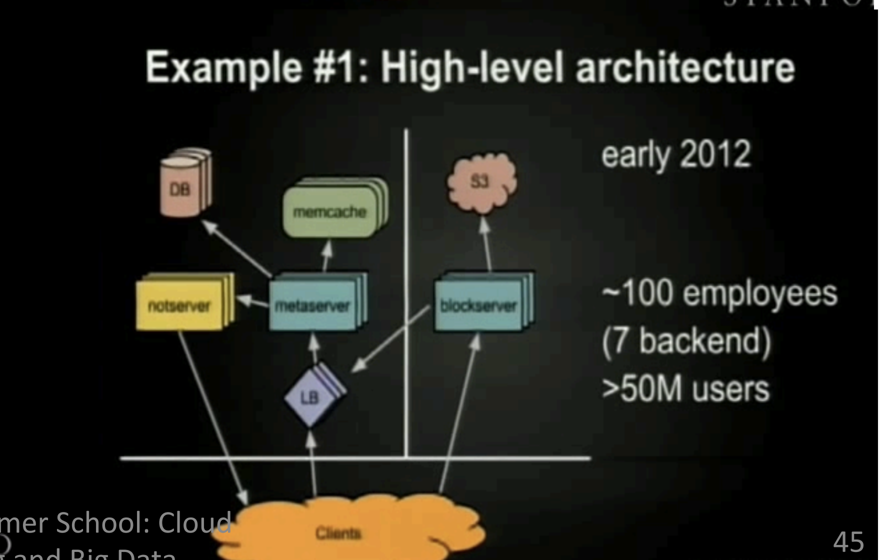
BIG DATA PRAGMATICS: DATA-CENTERS, DATA PIPELINES AND MULTI-HOMING (GOOGLE)

The Scalability Challenge: DropBox Case Study









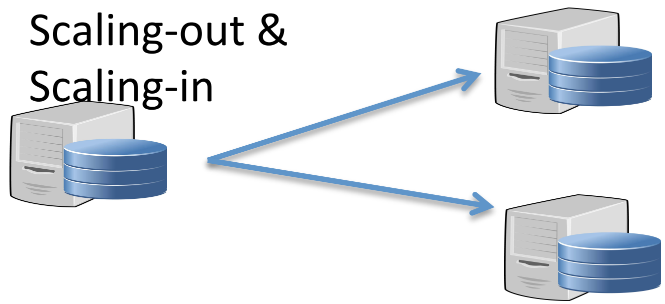
Datacenter is a new substrate: Why?

- Dis-aggregation (& virtualization) of resources:
  - Processing elements
  - Storage elements

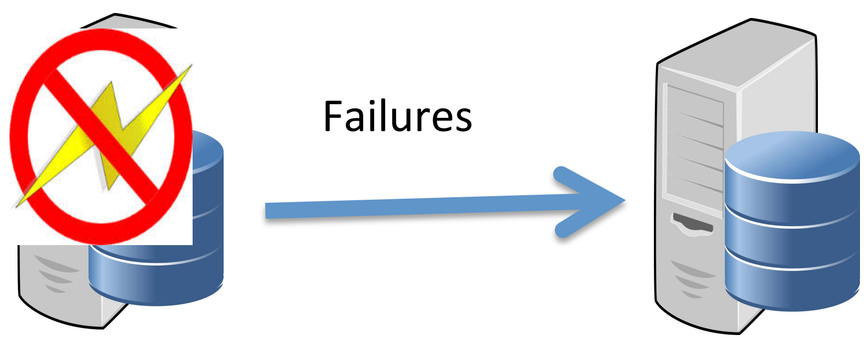
➔ The classical model of CPU + Disk is not tenable

Resource Dis-aggregation?

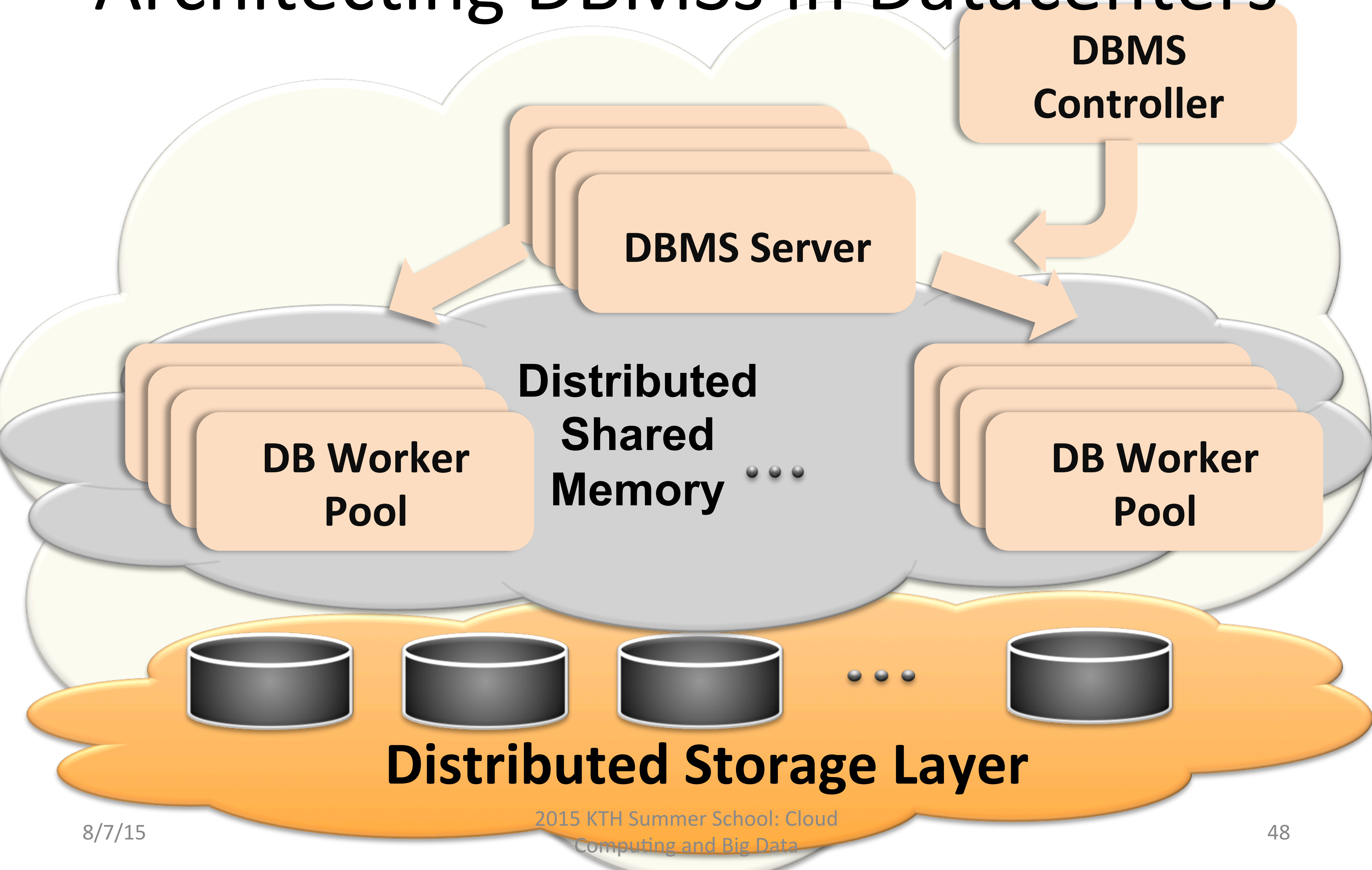
- At odds with the cloud computing model (scalability and elasticity)



- At odds with the utility model (fault tolerance)



Architecting DBMSs in Datacenters

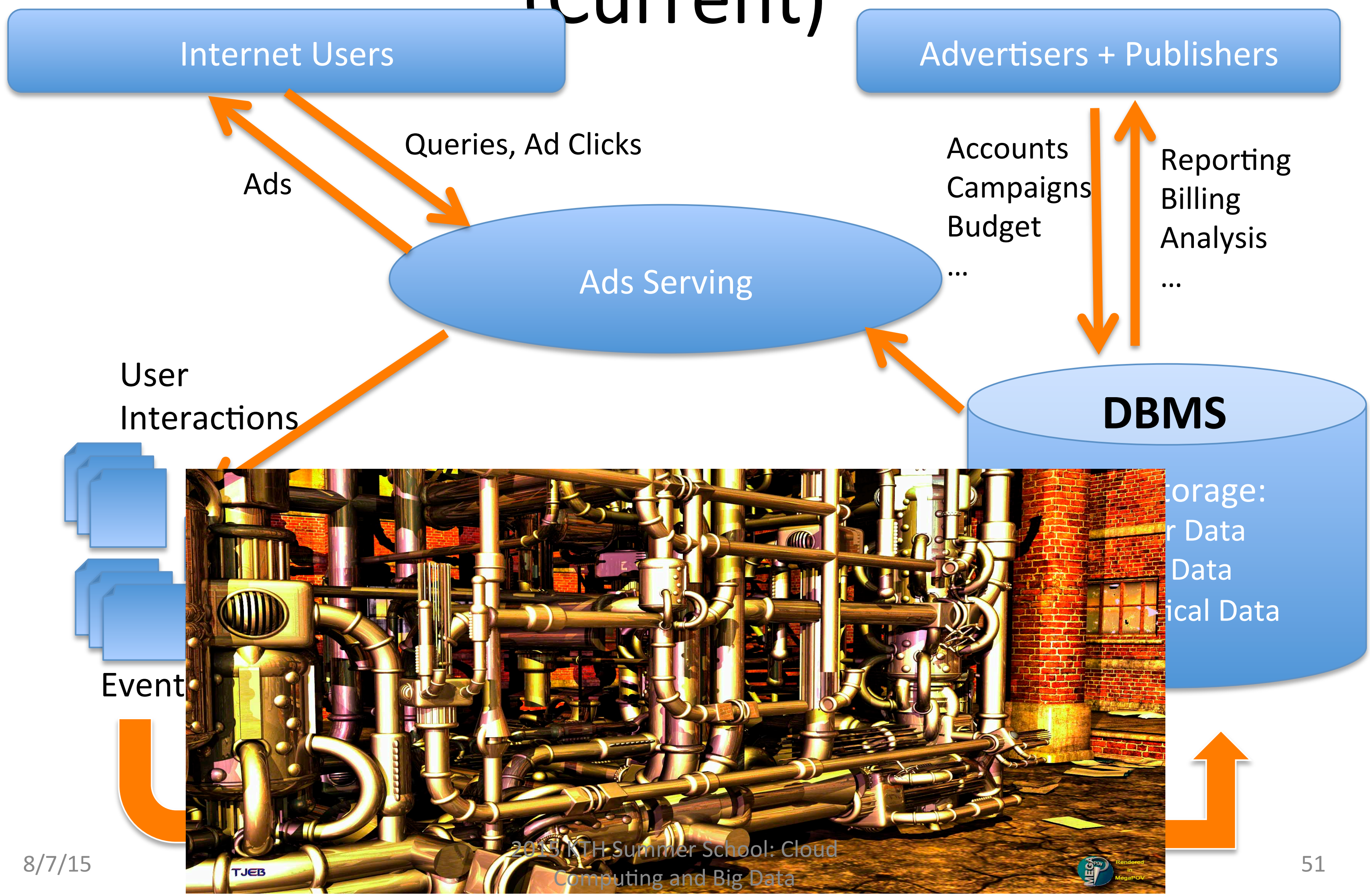


Notes on the Architecture

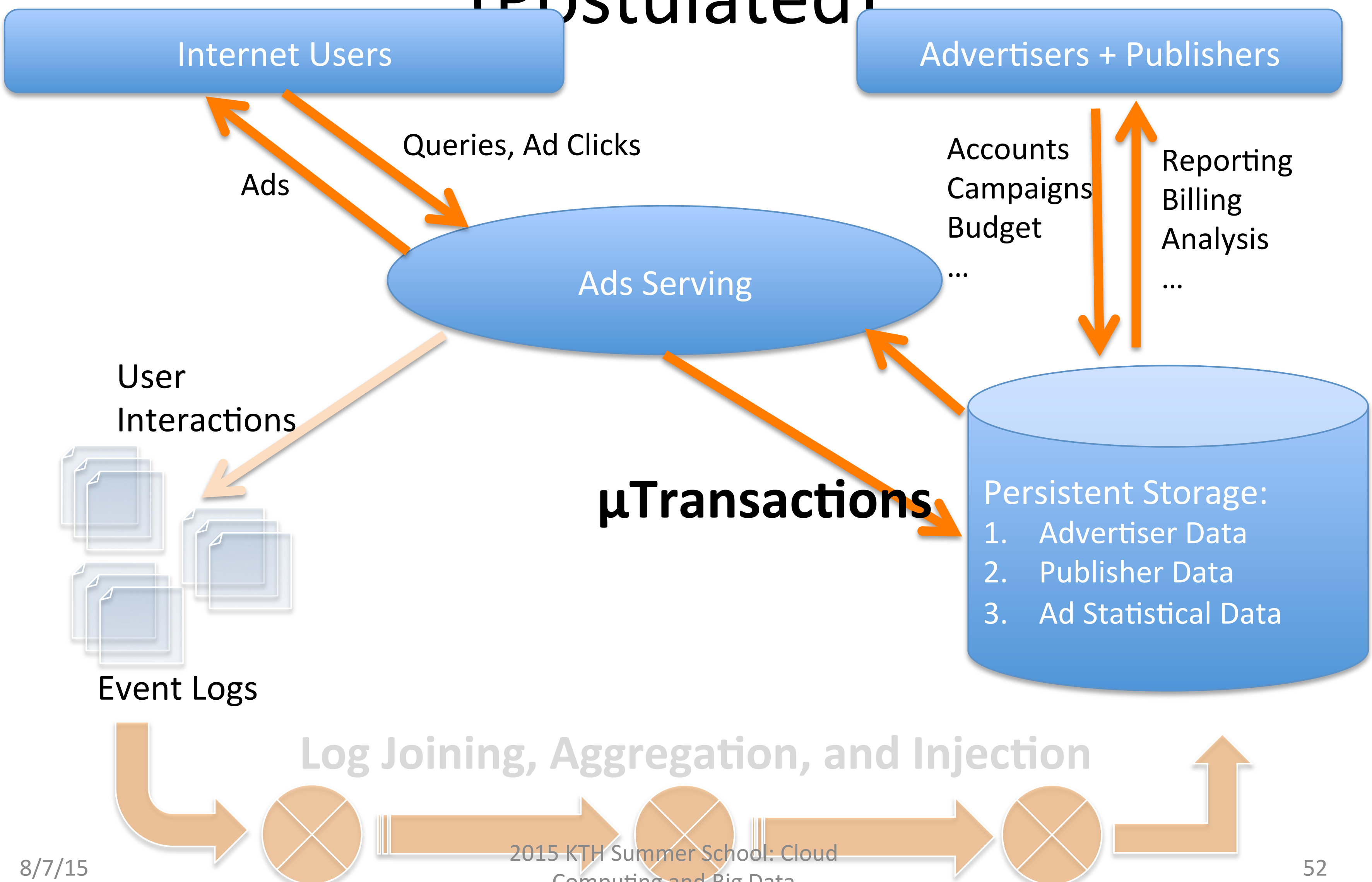
- Network and I/O latency mitigation:
  - Batching or pipelining of data accesses
  - Leverage parallelism at the distributed storage layer
- Query execution plans:
  - An additional degree-of-freedom (underlying resource platform is dynamic, e.g., 10 vs 100 machines)
- Data replication:
  - Block level vs DBMS level?

HIGH-VOLUME TRANSACTION PROCESSING

Internet Backend Architecture (Current)

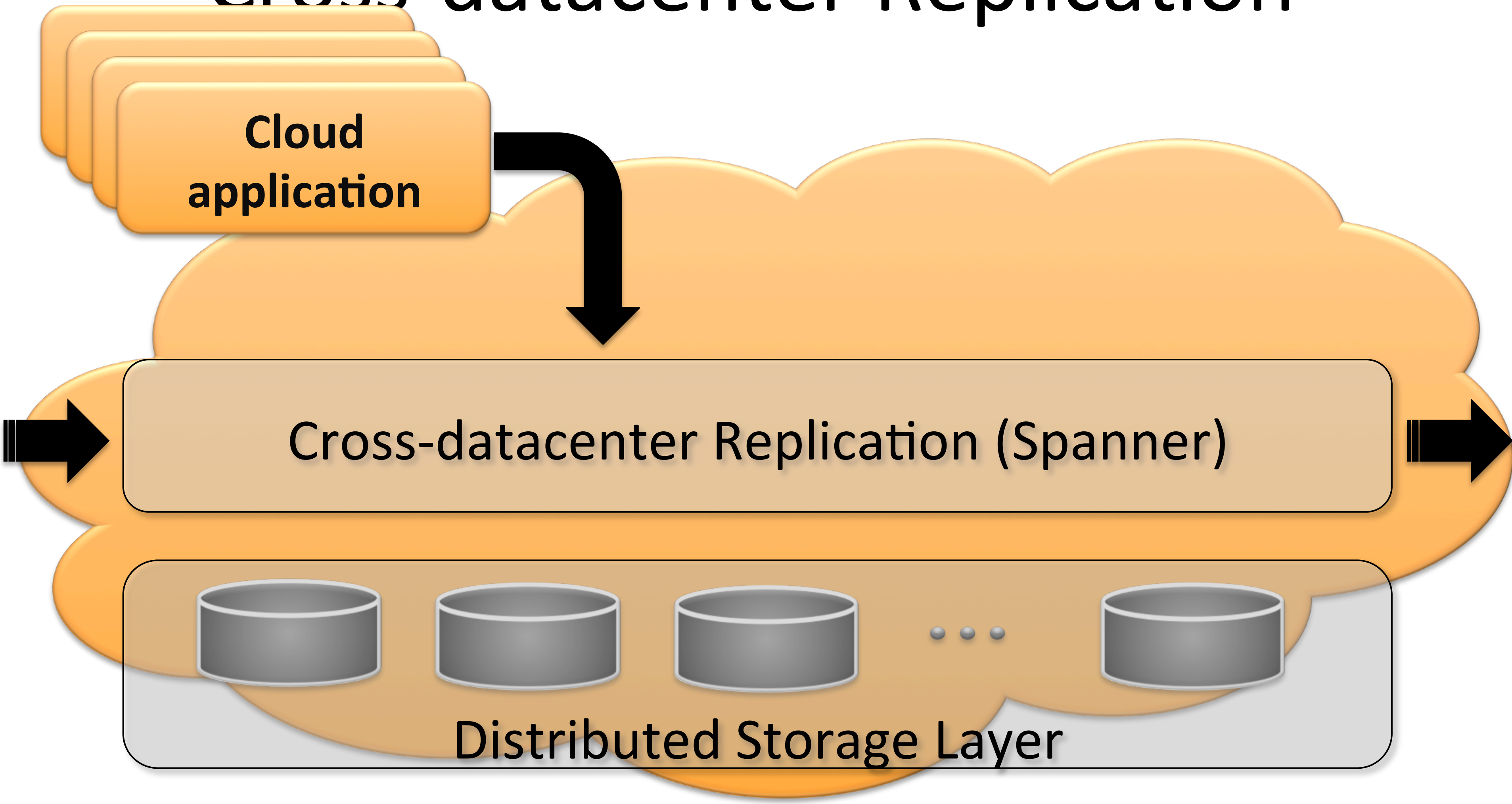


Internet Back-end Architecture (Postulated)



MULTI-HOMING (GEO-REPLICATION)

Cross-datacenter Replication

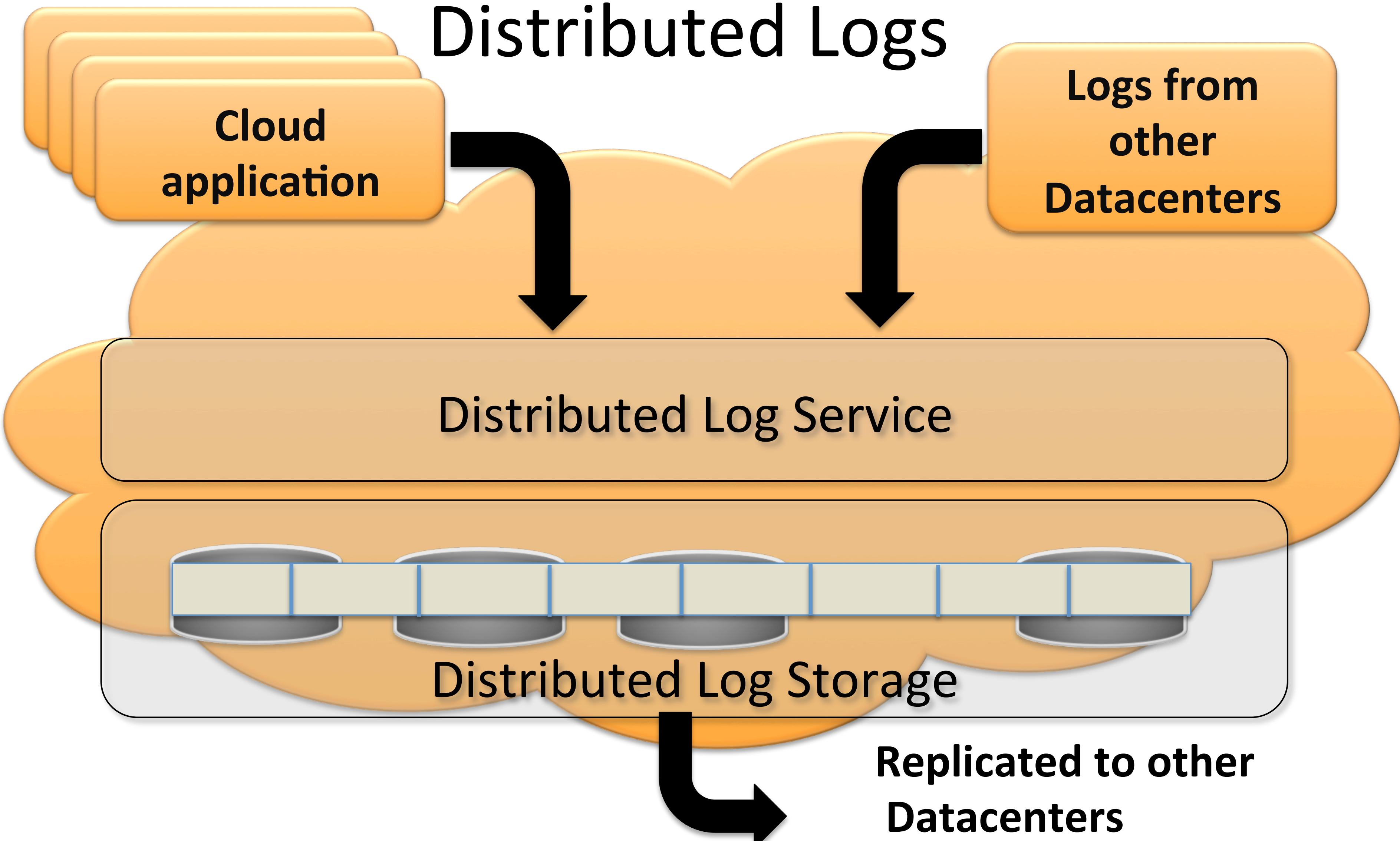


Cross-Datacenter Replication?

- Current state:
  - Google's technologies: MegaStore, Spanner, and PaxosDB
  - Typically, passive replication of ALL data
- Sustainable approach:
  - Critical data should be based on synchronous techniques
  - Most data, especially application data, should be updated using active replication (i.e., by executing operations redundantly at each datacenter)
- Why?
  - Fault-deterrence (by executing actions redundantly)
  - Operation latency not dependent on a single master (rather fastest quorum)

  → Cross-datacenter latencies: 100s of milliseconds

Cross-datacenter Replication: Distributed Logs



Big Data Pragmas

- Debunk Single Machine ➔ **Datacenter is a computer**
- Computation is already disaggregated
- Disaggregation of storage resources:
  - **Disk storage:** local disk assumption is seriously flawed
  - **Flash storage:** will be integral in the storage hierarchy
  - **Main memory:** likely to meet the same fate (local vs remote)
- Networking:
  - Intra-datacenter latencies < 0.5ms **(big opportunity)**
  - Inter-datacenter latency still remains a **challenge (need innovation)**

Concluding Remarks

- Cloud Computing Challenge:
  - Scalability, Reliability, and Elasticity
  - Re-architecting DBMS technology
- Big Data Analysis and Learning:
  - Scaling Iterative Computation over Big Data
  - DBMS-like platform for Machine Learning
- Big Data Pragmatics:
  - Complex Data Processing Pipelines
  - Multi-homing and Geo-replication