Google™ 11 IO

More 9’s Please:
Under the Covers of the
High Replication Datastore

Matt Wilder
Alfred Fuller
2011-05-11

Hashtags: #io2011 #AppEngine

Feedback: http://goo.gl/l3ojJ

Who?

Alfred Fuller

- Software Engineer
- App Engine – Datastore

Matt Wilder

- Site Reliability Engineer
- Distributed Storage

Past Datastore talks (on YouTube)

2010 - Next Gen Queries

2009 - Building Scalable, Complex Apps on App Engine

2008 - Under the Covers of the Google App Engine Datastore

Outline

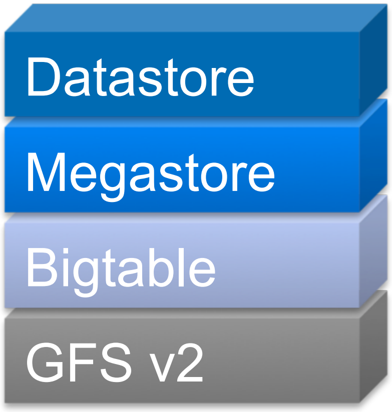
- Datastore Overview
- Datastore in Production
  - Common Case
  - Planned Maintenance
  - Unplanned Events
- Lessons Learned
- High Replication Tips

Datastore Types

	Master / Slave	High Replication
Released	April 2008	January 2011
Replication	Asynchronous	Synchronous
Replicas	2	>2 . . .
Master	Single	None

Datastore Software Stack

- App Engine Datastore
  - Schema-less storage
  - Advanced query engine
- Megastore
  - Multi-row transactions
    - Across machines
    - Entity Groups
  - Simple indexes/queries
  - Strict schema
- Bigtable
  - Distributed key/value store
- Next gen distributed file system



Datastore

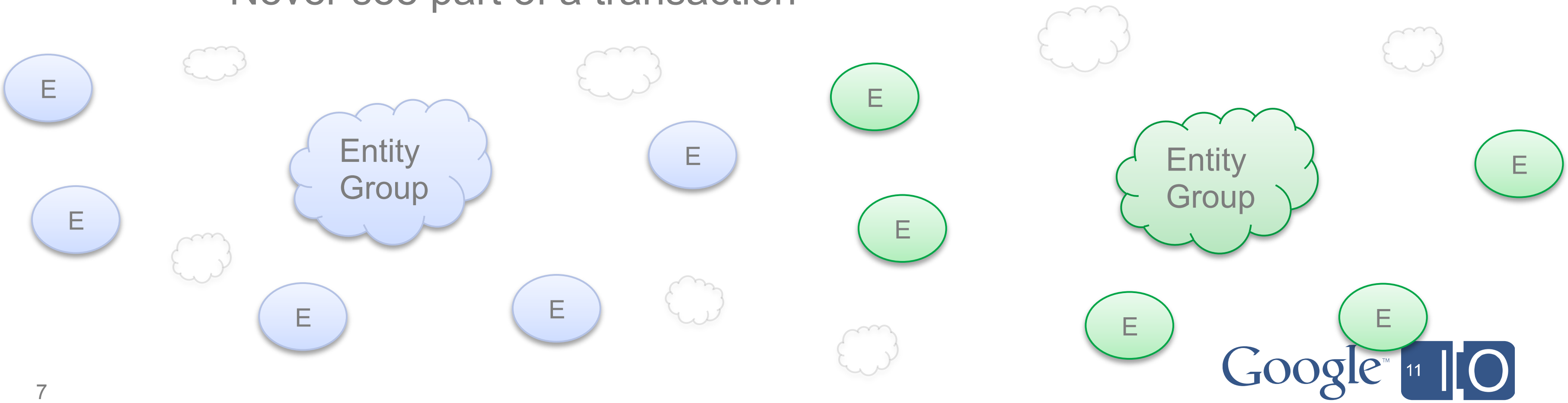
Megastore

Bigtable

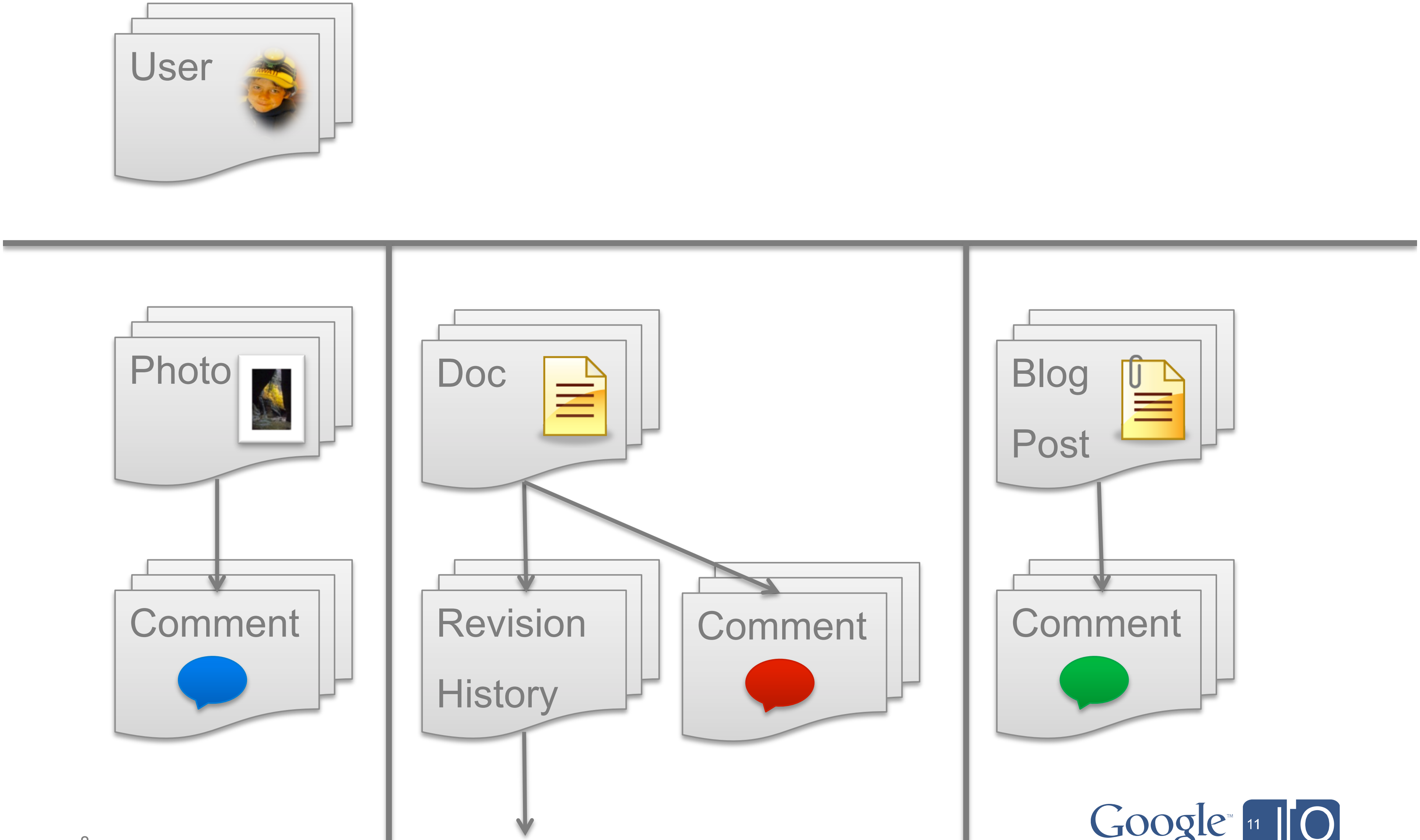
GFS v2

Entity Group?

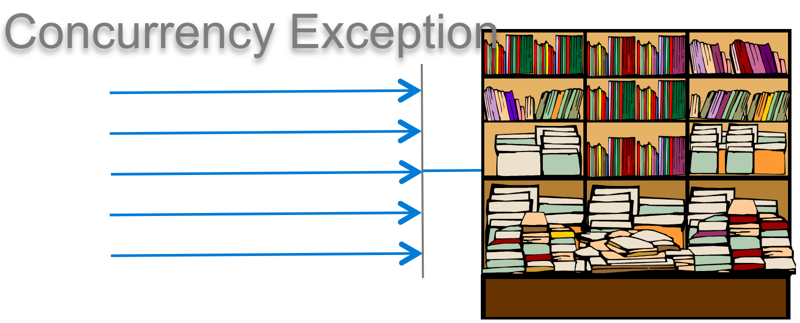
- Logical grouping of entities
  - Parent/child key relationship
- Unit of Transactionality
  - Transactions can only read/write entities in a single group
- Unit of Consistency
  - Strong serial consistency
    - Will always Get an entity once Put
    - Never see part of a transaction



Entity Group Example

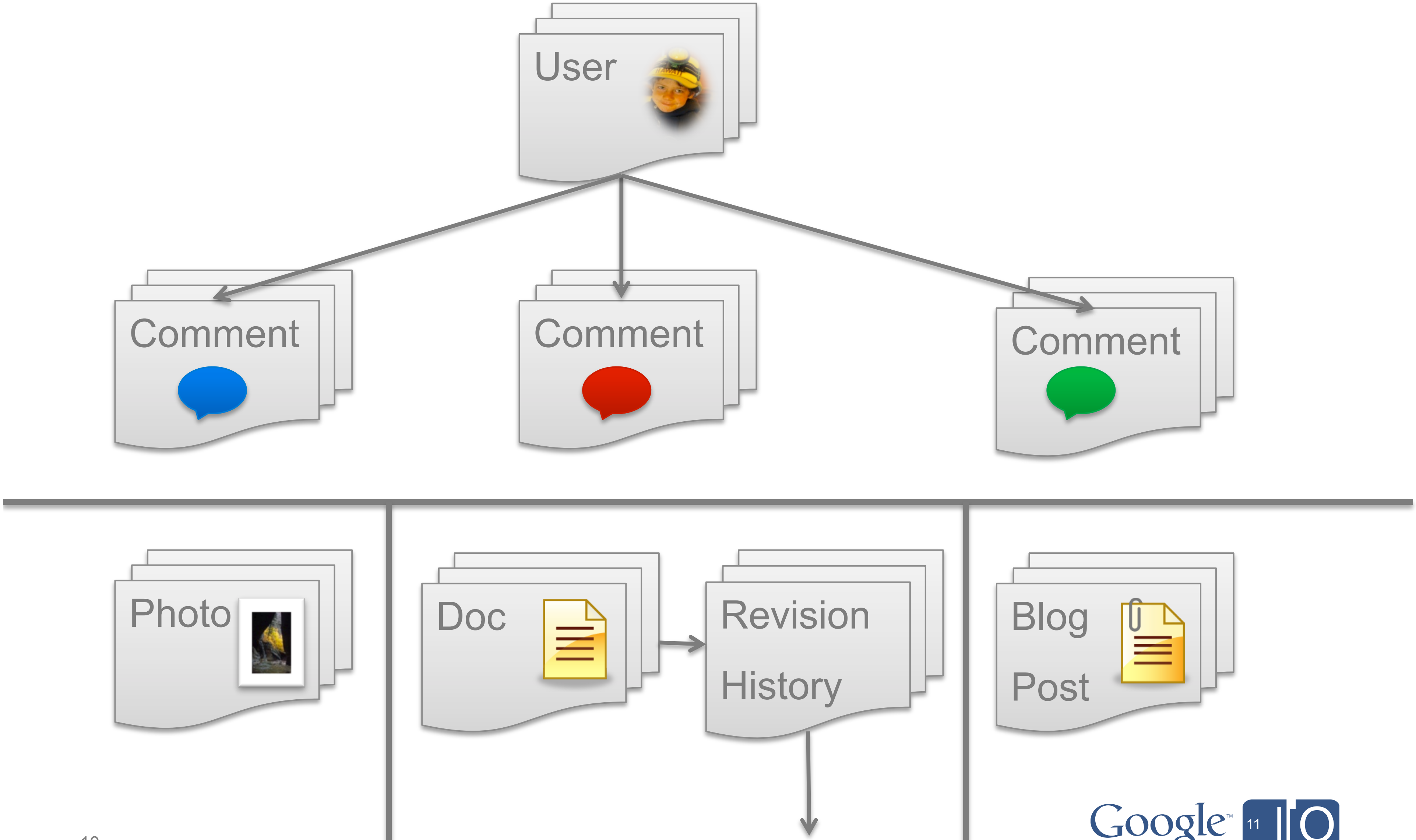


Entity Group – Limitations

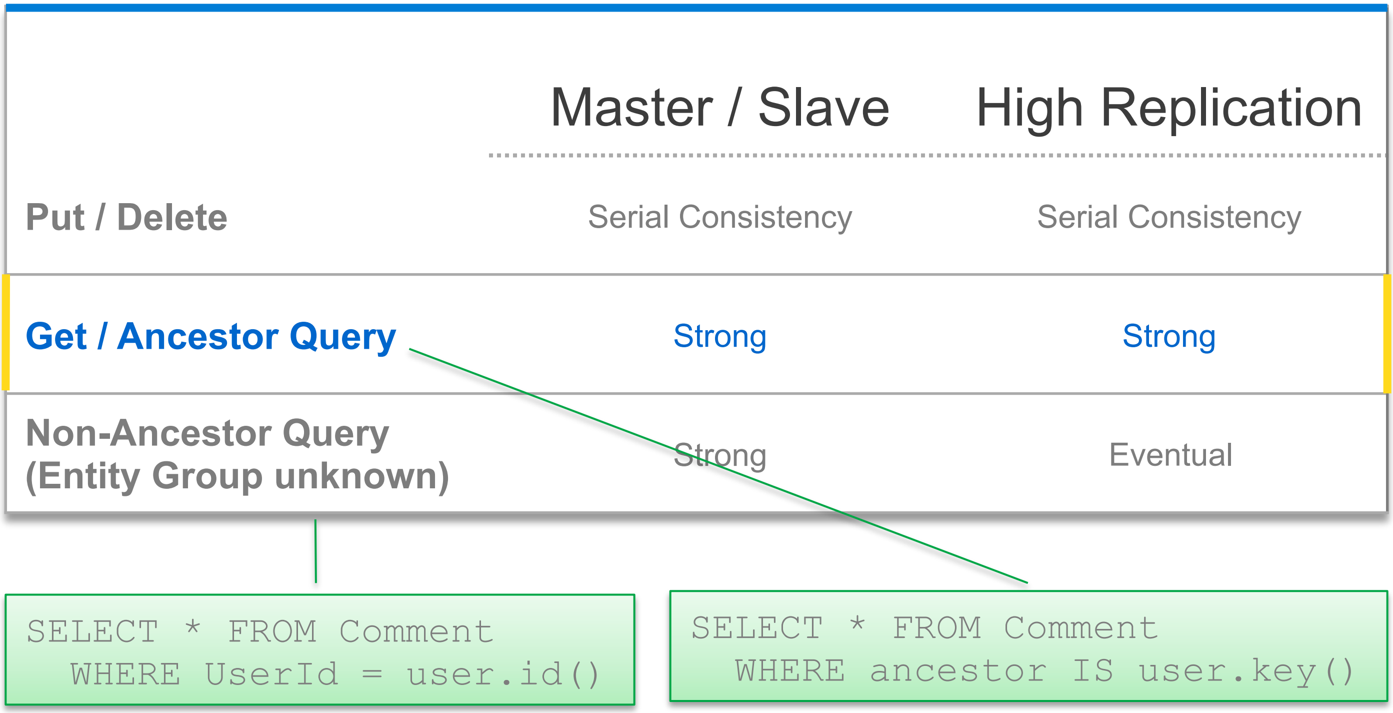


- Throughput limited
  - At least 1 write / second
  - 5-10 in practice
- Write / Sec != Entity / Sec
  - Batch puts / transactions count as 1 write!
- Arbitrary Size
  - 10’s of Millions of entities

Entity Group Example – User Centric



Datastore – Consistency



	Master / Slave	High Replication
Put / Delete	Serial Consistency	Serial Consistency
Get / Ancestor Query	Strong	Strong
Non-Ancestor Query (Entity Group unknown)	Strong	Eventual

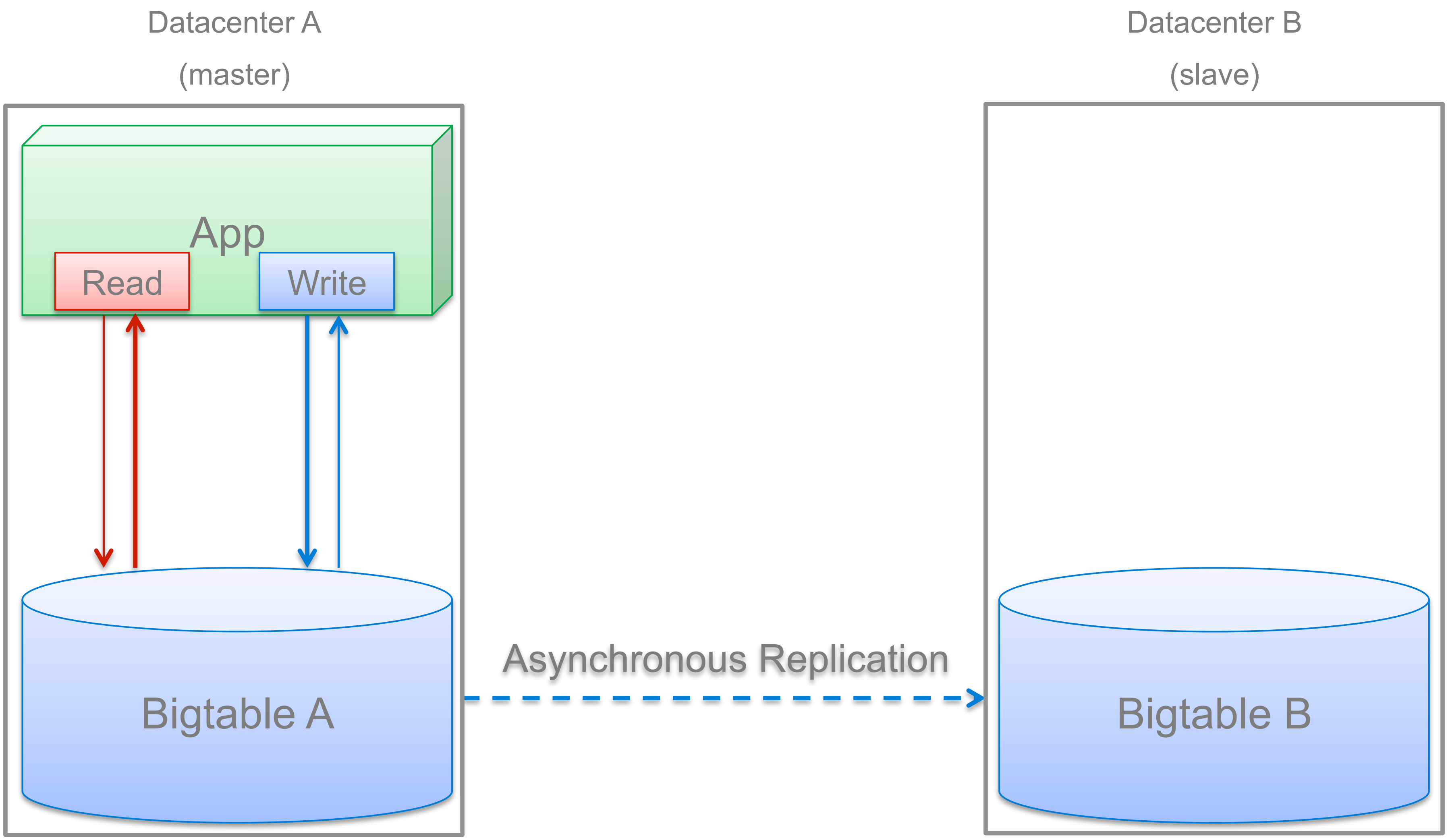
```
SELECT * FROM Comment
  WHERE UserId = user.id()
```

```
SELECT * FROM Comment
  WHERE ancestor IS user.key()
```

Common Case - Master / Slave

- Write
  - Write local (master)
  - Asynchronous replication
- Read
  - Read local (master)

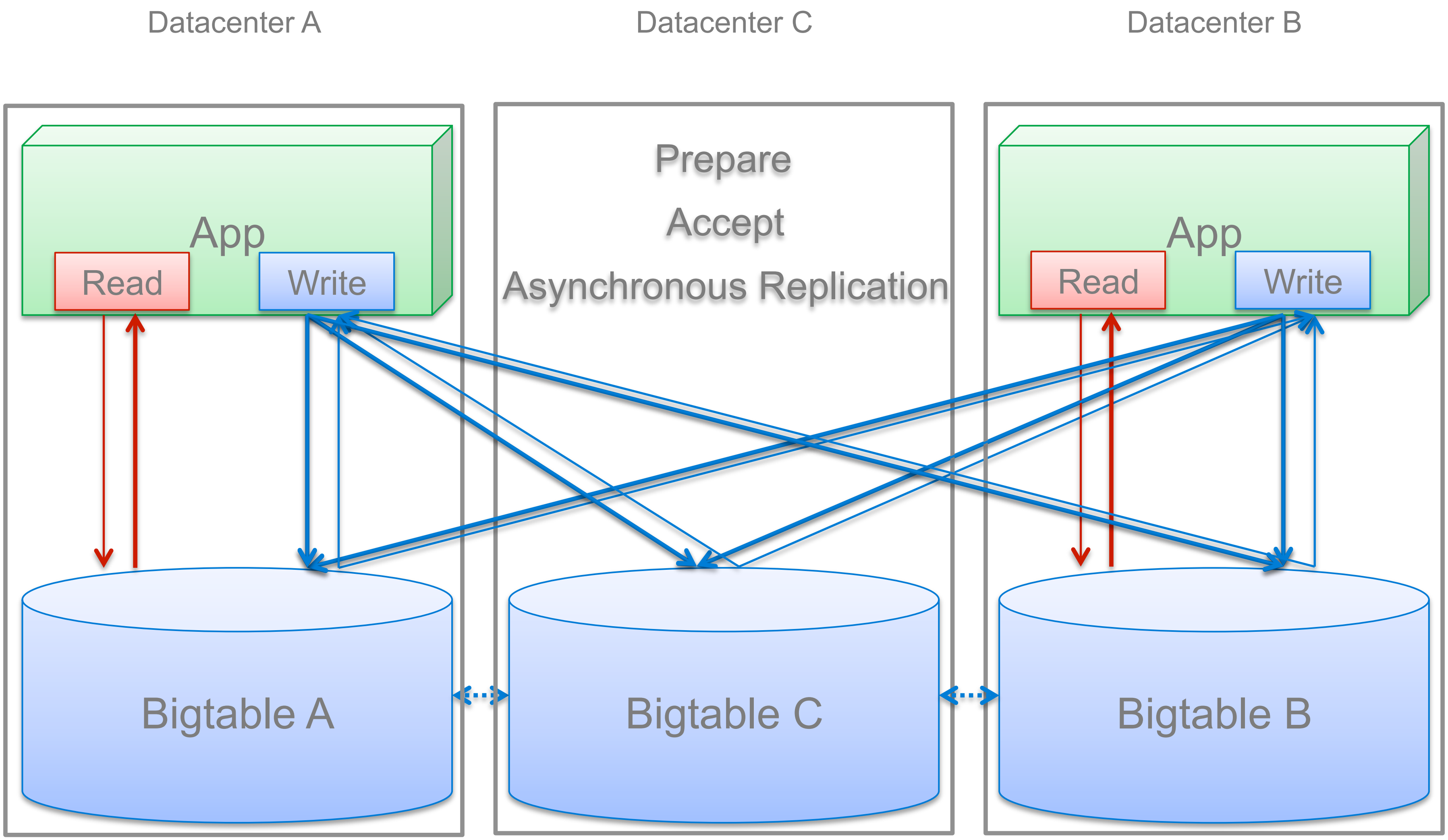
Master Slave



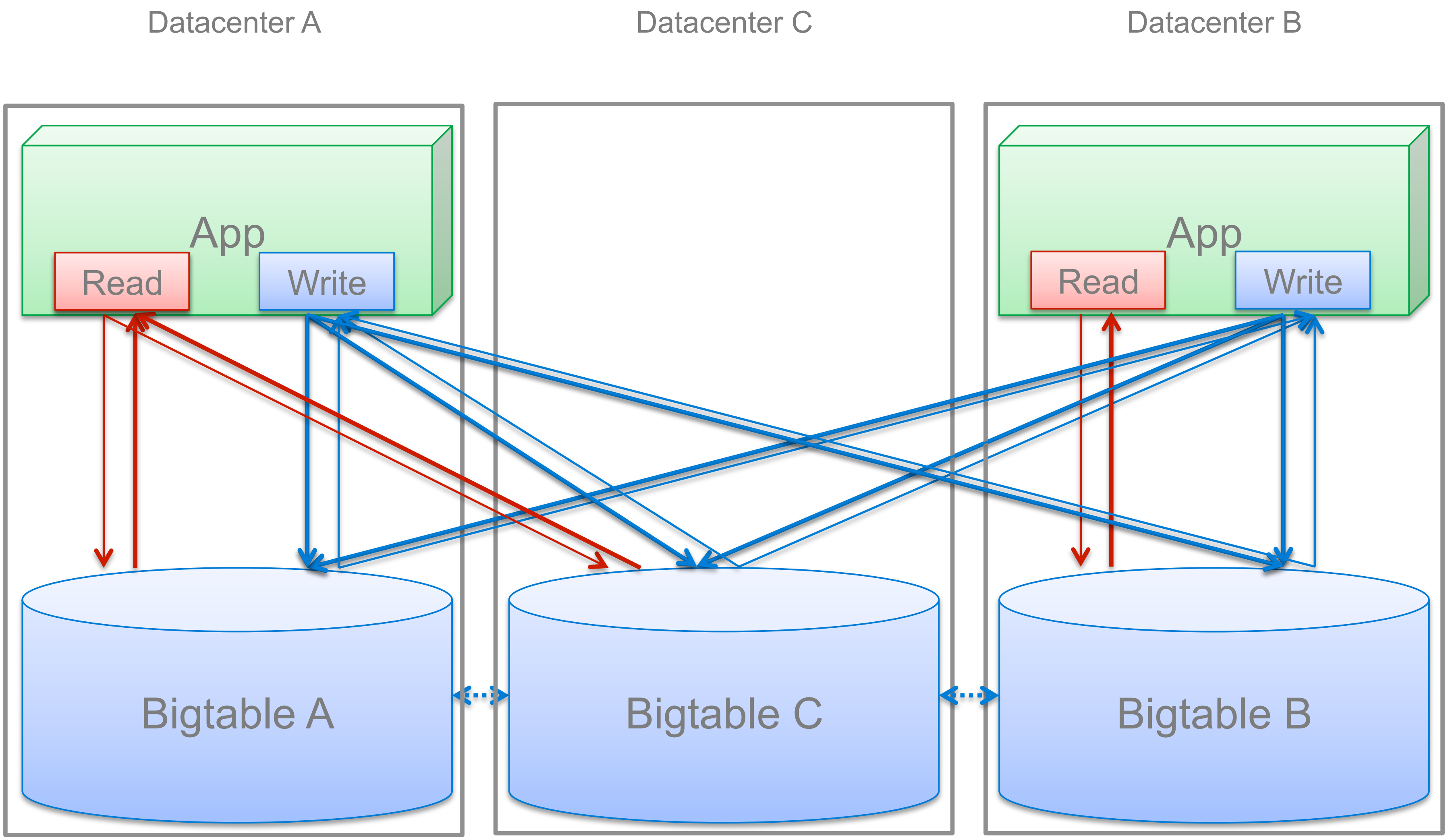
Common Case – High Replication

- Write
  - Write to at least a majority
    - Two phase
    - Minority may not get write synchronously
  - Asynchronous replication
  - On demand replication
- Read
  - Read
    - Fastest (usually local)
    - Catch up on demand

High Replication



High Replication



Datastore – Performance

NOTE: These numbers are approximate

		Master / Slave	High Replication
Average Latency	**Read**	15 ms	15 ms
	Write	20 ms	45 ms
Average Error Rate	**Read**	.1%+	.001%
	Write	.1%+	.001%

99.9% = Three 9’s

= 8.7 hours/year

99.999% = Five 9’s

= 5 minutes/year

Planned Maintenance

Cause

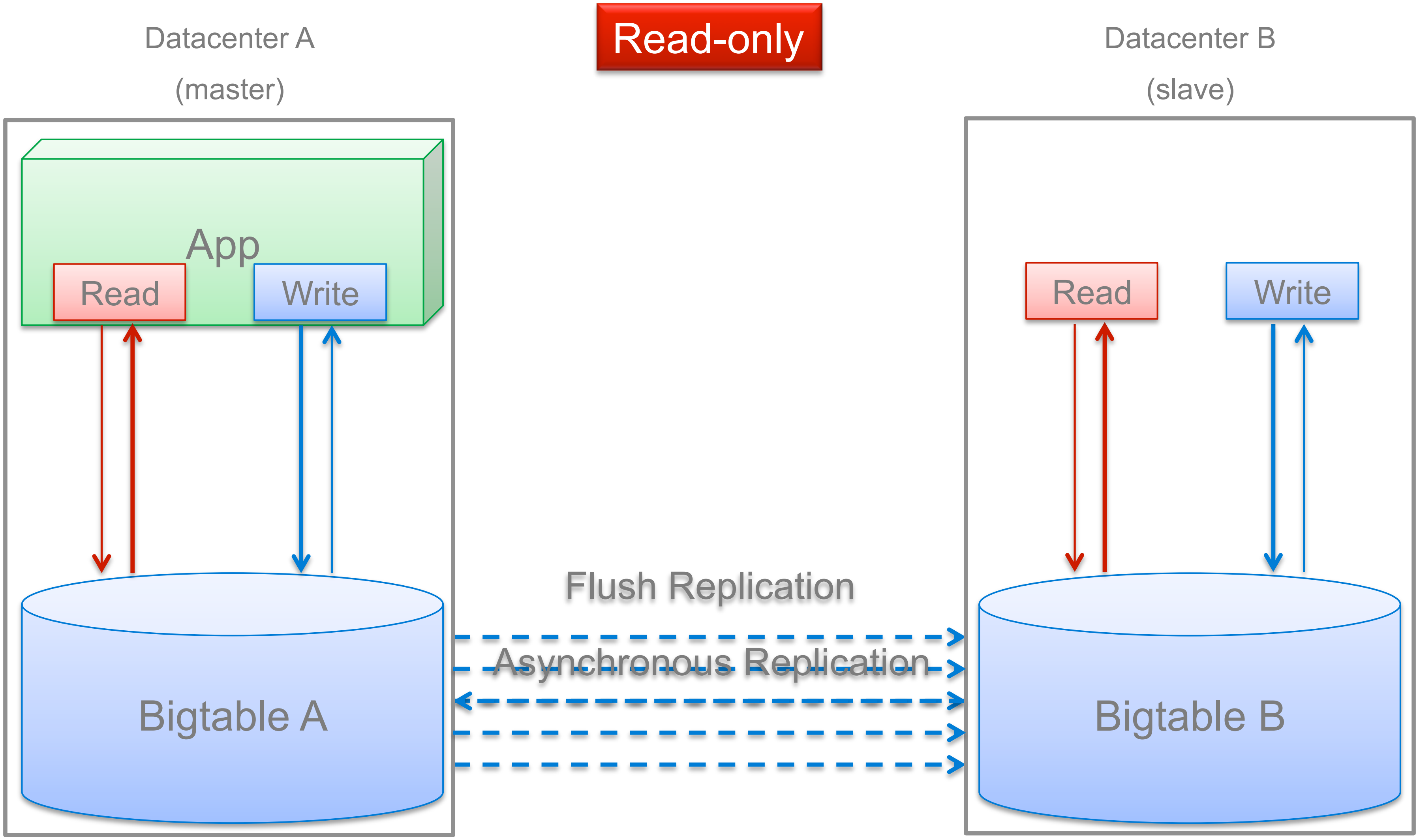
- Common infrastructure updates
  - Network
  - Power/Cooling
  - Distributed Storage
- Why?
  - Not all services support in place upgrades
  - Architectural services (power, cooling) must be taken offline

Planned Maintenance

Effect: Master/Slave

- Maintenance Period
  - Switch Masters
  - 1 hour of read-only datastore
    - Semi-automated procedure (requires engineer)
    - Maintenance windows

Master Slave – Planned Maintenance

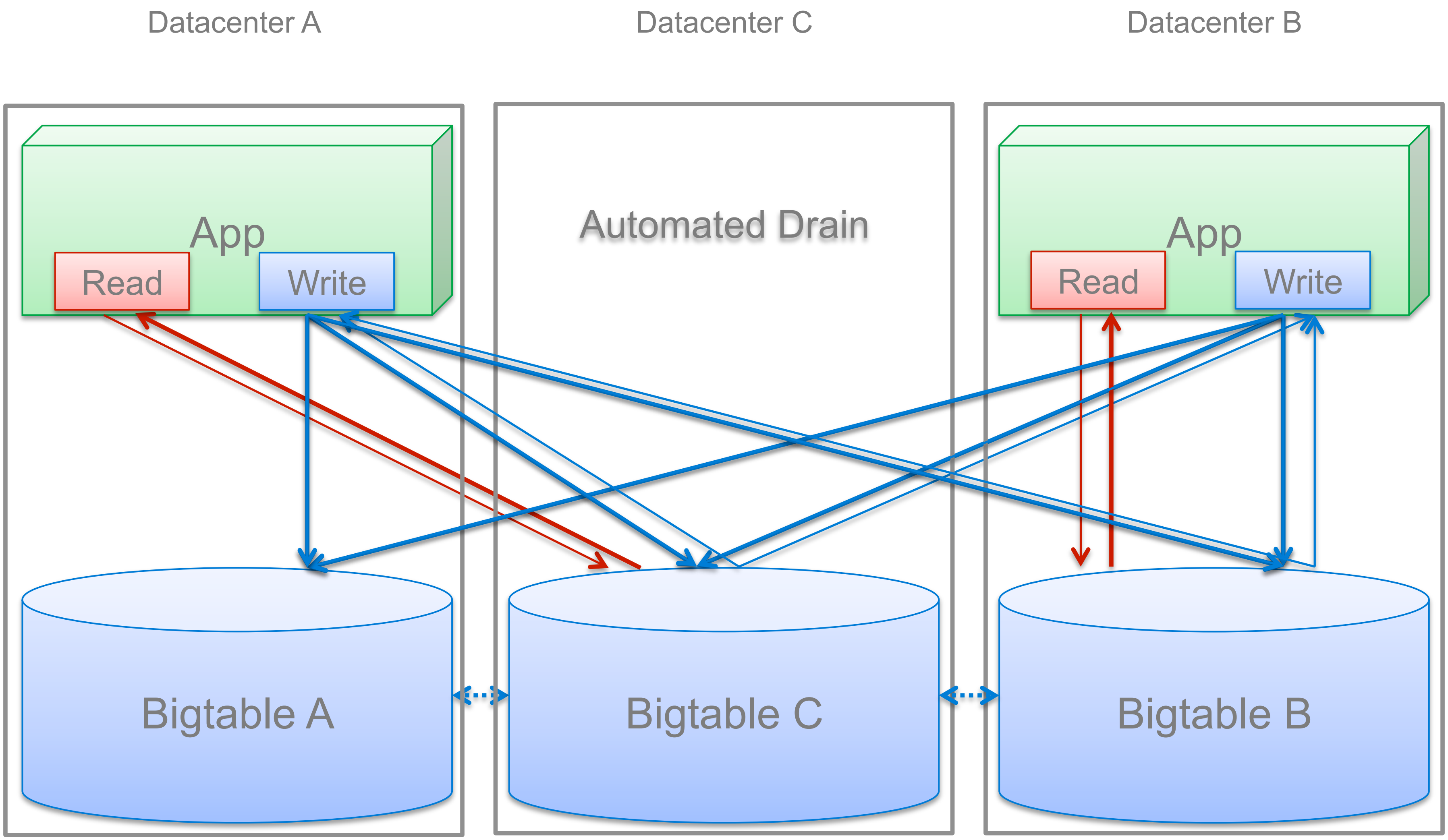


Planned Maintenance

Effect: High Replication

- Seamless Migration
  - Applications serve primarily in 1 datacenter
  - Switching is *almost* transparent
    - Memcache flush + 1 min no-caching

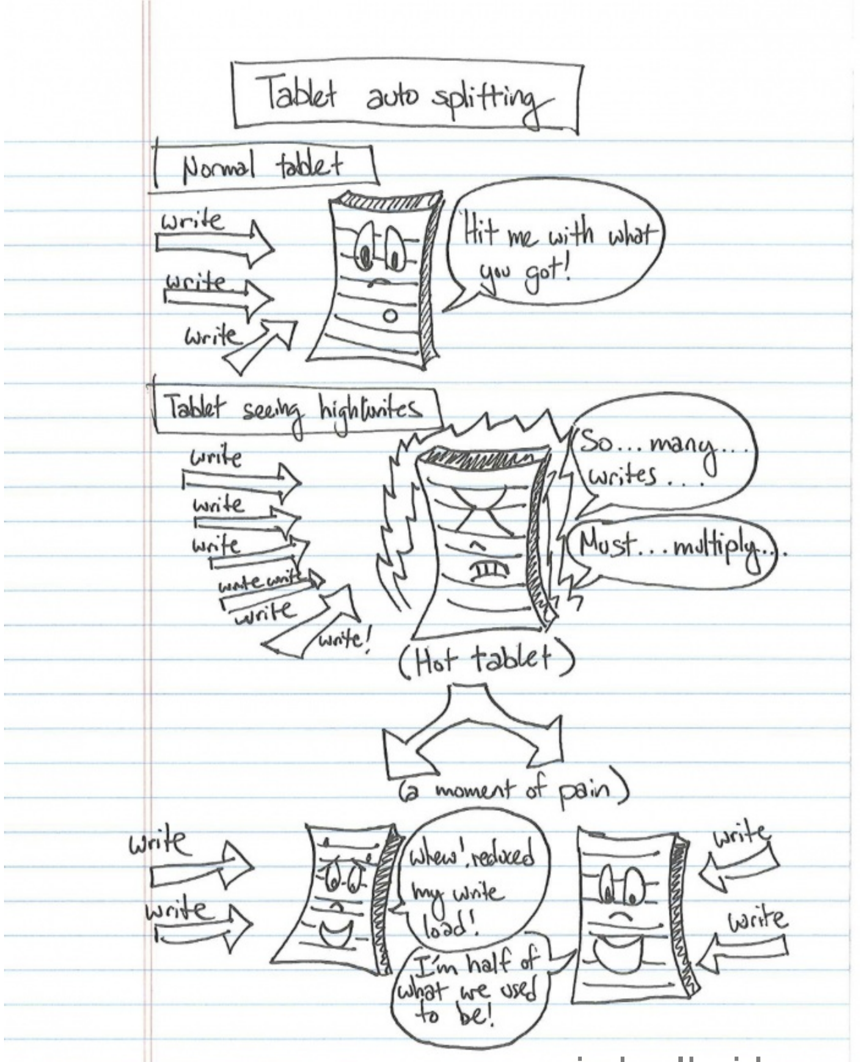
High Replication – Planned Maintenance



Unplanned Issues – Local Failures

Cause

- Expected
  - Tablet split
  - Tablet migration
- Unexpected
  - Inconsistent Bigtable Performance
    - Sick tablet server
    - Shared storage
    - Isolation



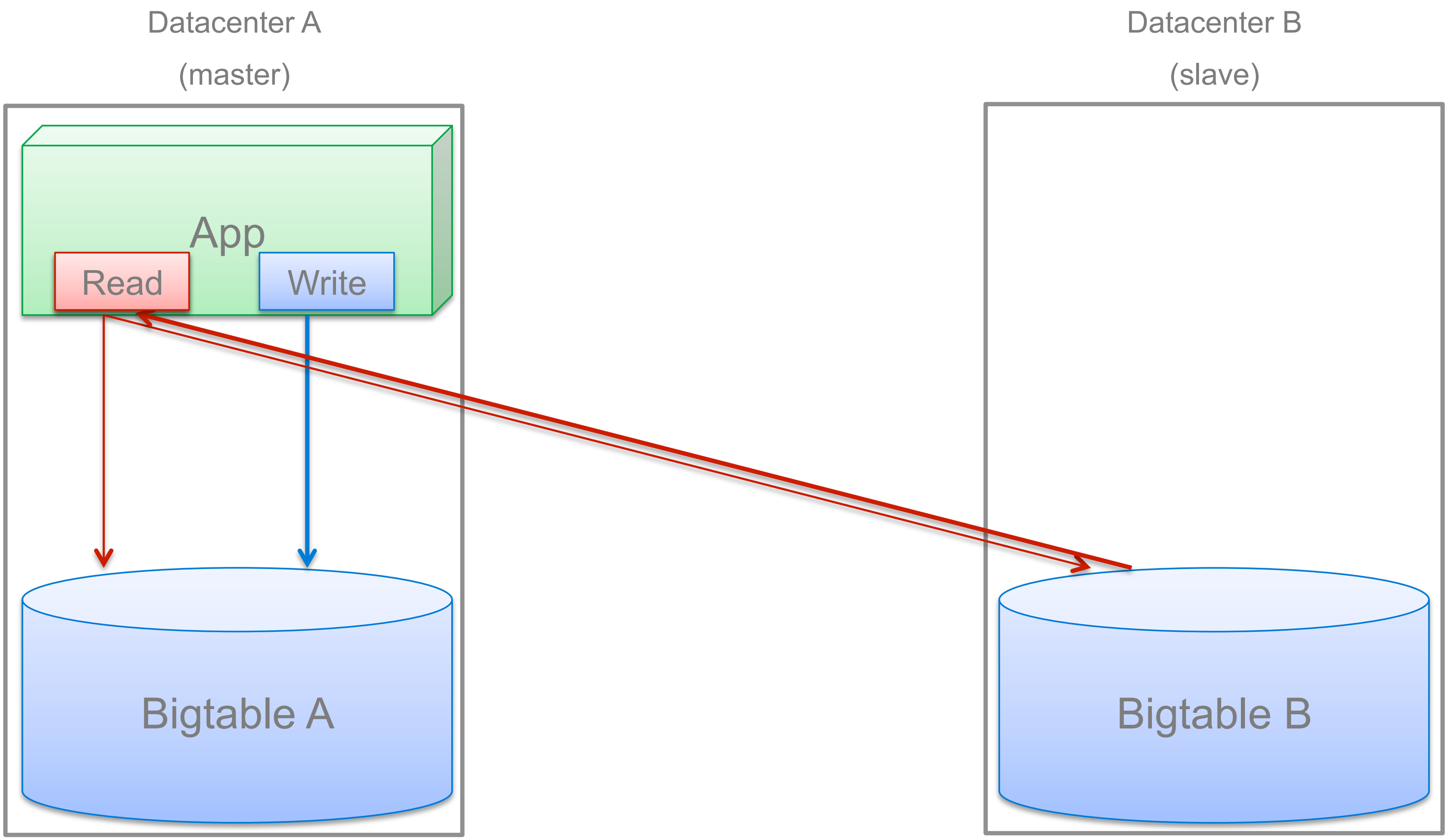
a comic by Ikai Lan

Unplanned Issues – Local Failures

Effect: Master/Slave

- Local Unavailability
  - App data unavailable
  - DeadlineExceeded
  - Request queue can back up
  - Clustered in space and time
  - Status site still green
    - http://code.google.com/status/appengine

Master Slave – Local Failures

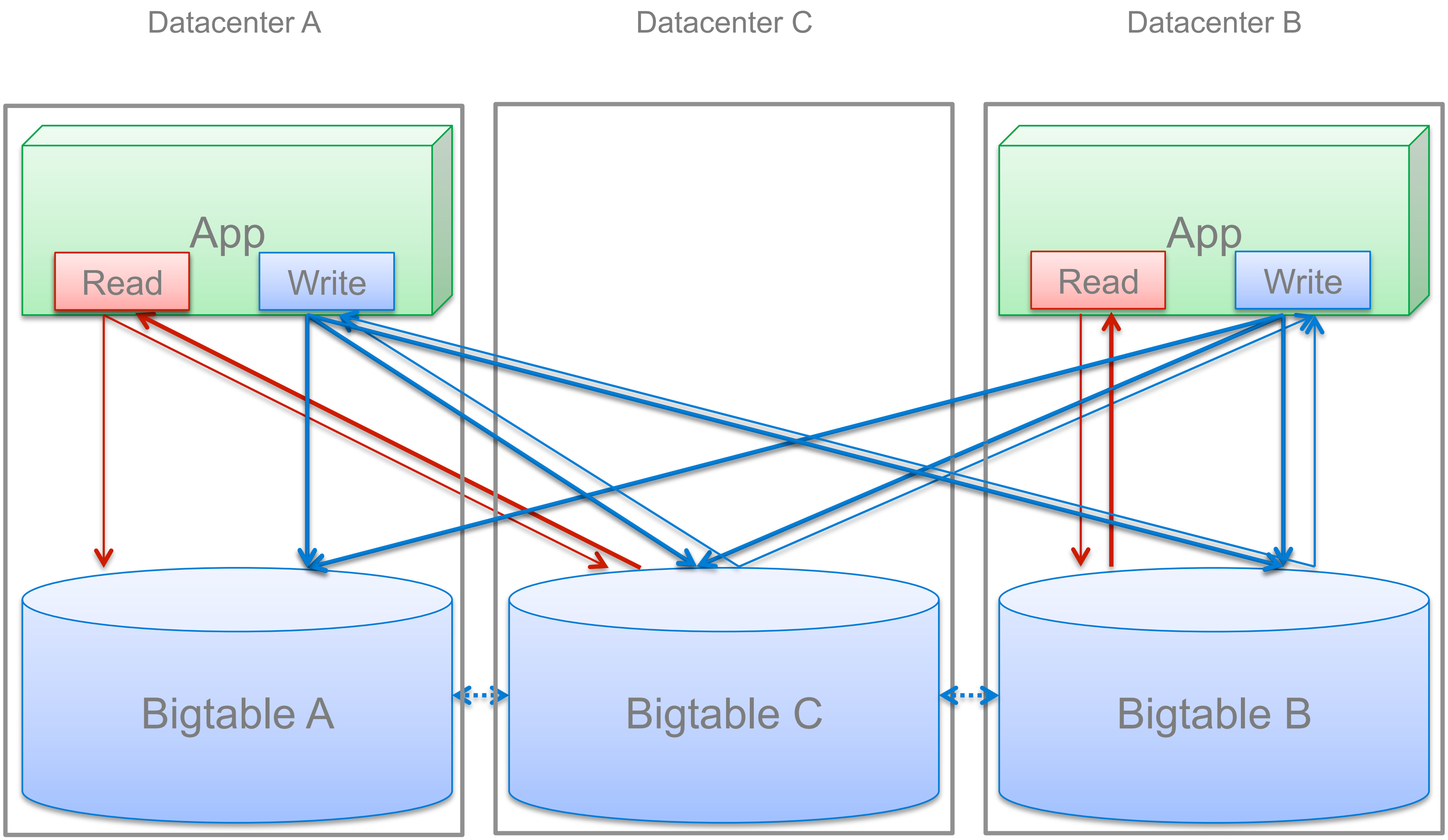


Unplanned Issues – Local Failures

Effect: High Replication ☀

No impact on performance!

High Replication – Local Failures



Unplanned Issues – Global Failures

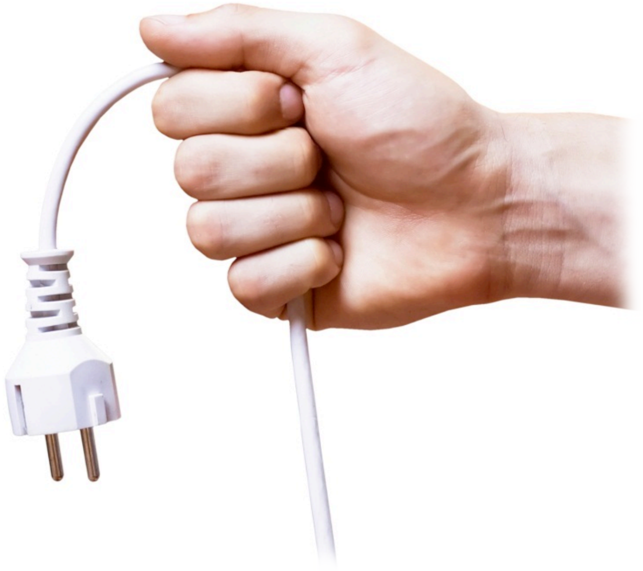
Cause

- Network
- Power
- Shared Infrastructure
  - Bigtable
  - Distributed Storage
  - Cluster Management

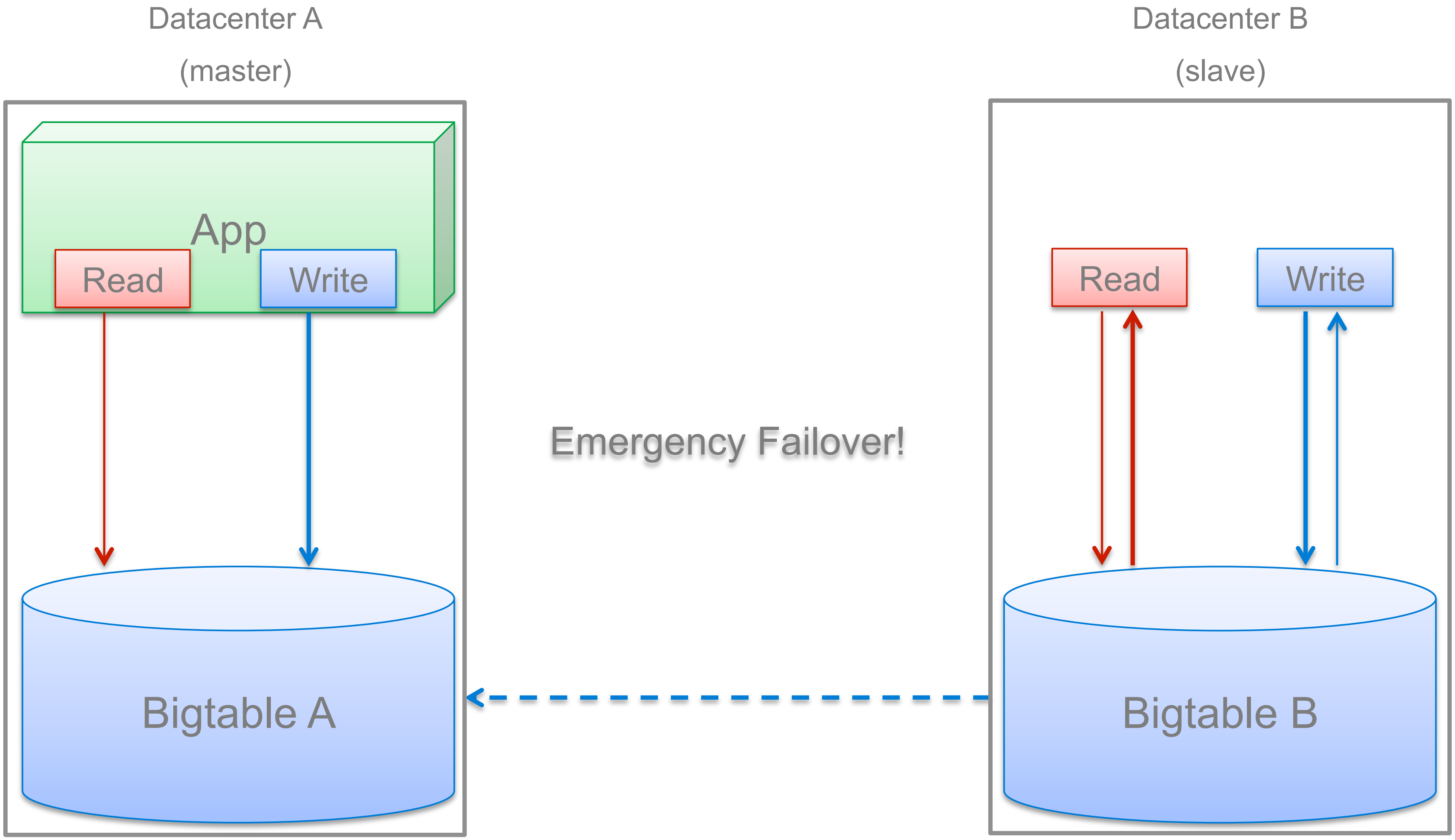
Unplanned Issues – Global Failures

Effect: Master/Slave

- Complete Unavailability
  - Not just the Datastore
- Emergency Failover
  - Temporary data loss
    - Unreplicated data
    - Partially replicated data



Master Slave – Global Failure

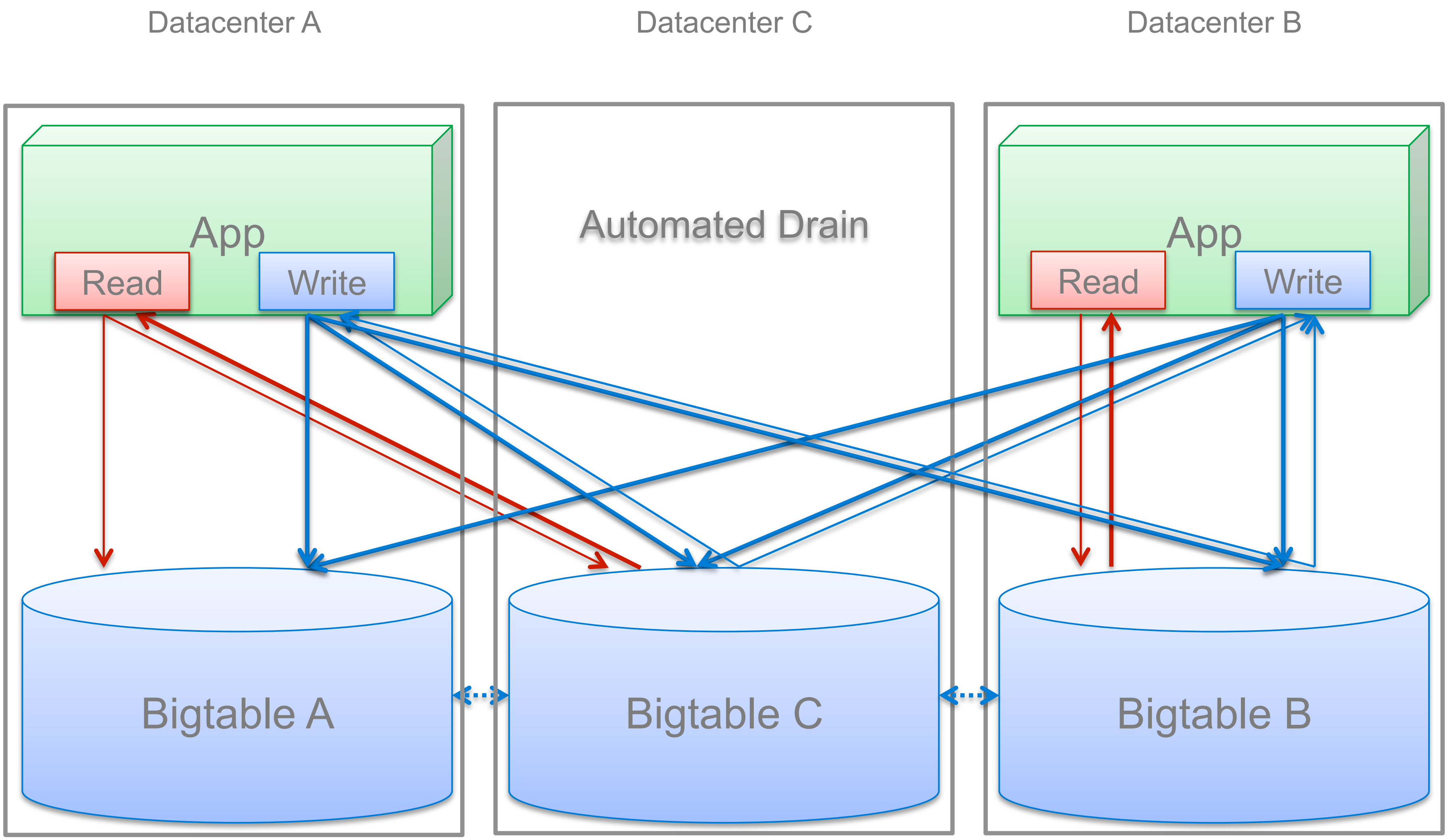


Unplanned Issues – Global Failures

Effect: High Replication ☀

- Brief Unavailability
  - On the order of minutes
  - Automatic infrastructure drain
- Data Integrity Maintained
- Redundancy Maintained
  - Can lose multiple datacenters

High Replication – Global Failure



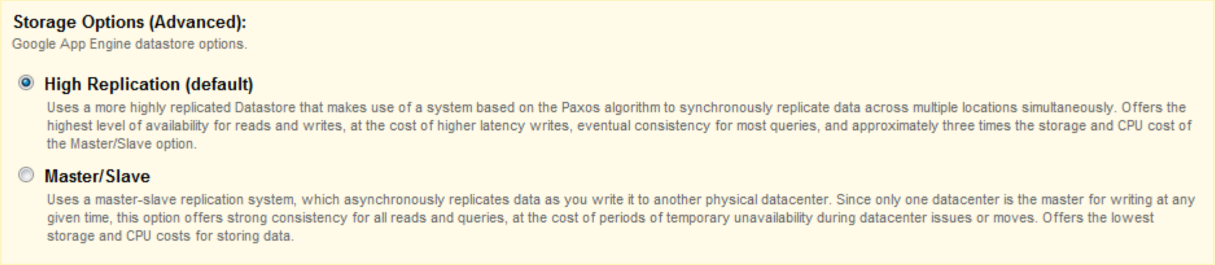
Lessons Learned

- Expect the unexpected!
  - Global Failures are never expected
  - The improbable is probable at scale
- Consistent performance > low latency
  - Low latency + inconsistent performance != low latency
  - Developers can program around slower if expected
- Fully-automatic failure handling means less downtime
  - Faster reaction time
  - Better fault recovery
- Unavailability is never good
  - Small percentage at Google’s scale has a big impact

High Replication Recap

- Slightly higher write latency
- Slightly less global consistency
- Fault Tolerant to a Fault
  - Geographically distributed
  - Resilient to catastrophic failure
  - Many more 9's!

- Reduced price
- Default ON!



Storage Options (Advanced):

Google App Engine datastore options.

● **High Replication (default)**

Uses a more highly replicated Datastore that makes use of a system based on the Paxos algorithm to synchronously replicate data across multiple locations simultaneously. Offers the highest level of availability for reads and writes, at the cost of higher latency writes, eventual consistency for most queries, and approximately three times the storage and CPU cost of the Master/Slave option.

○ **Master/Slave**

Uses a master-slave replication system, which asynchronously replicates data as you write it to another physical datacenter. Since only one datacenter is the master for writing at any given time, this option offers strong consistency for all reads and queries, at the cost of periods of temporary unavailability during datacenter issues or moves. Offers the lowest storage and CPU costs for storing data.

- What's next?
  - Improved migration tools

Dealing with Eventual Consistency

- Code audit to find global queries
  - Everything else is strongly consistent
- Accept it
  - A lot of global queries don’t need strong consistency
- Avoid it
  - Use larger entity groups + batch writes
- Work around it
  - Mix datastore results
    - Ancestor Query + Global Query
    - Memcache
      - Session Cache (keep track of recent writes for a user)

Questions?

Hashtags: #io2011 #AppEngine

Feedback: http://goo.gl/l3ojJ

Google™ 11 I O