# Project Voldemort

## Distributed Key-Value Storage


Roshan Sumbaly

# What has changed?


- No joins
- Making data access APIs cacheable
- Frequent schema changes
- Rise of huge datasets - storing relationships
  - Batch computed offline, serve in near real time – People you may know (#in), Who to follow (#twitter)

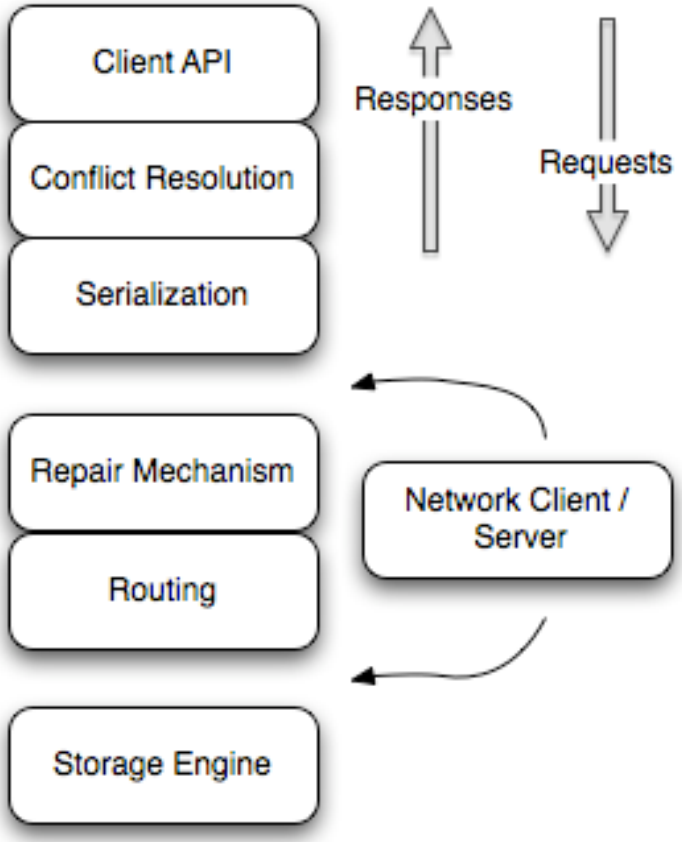
# So, what should our system do?


- Growing dataset – Horizontal Scalability
  - Partition the data
  - Make it transparent to the application
- High availability and durability
  - Replicate the data
- Fast per-node performance
- Simple API with predictable performance
- No single point of failure

# What inspired you?

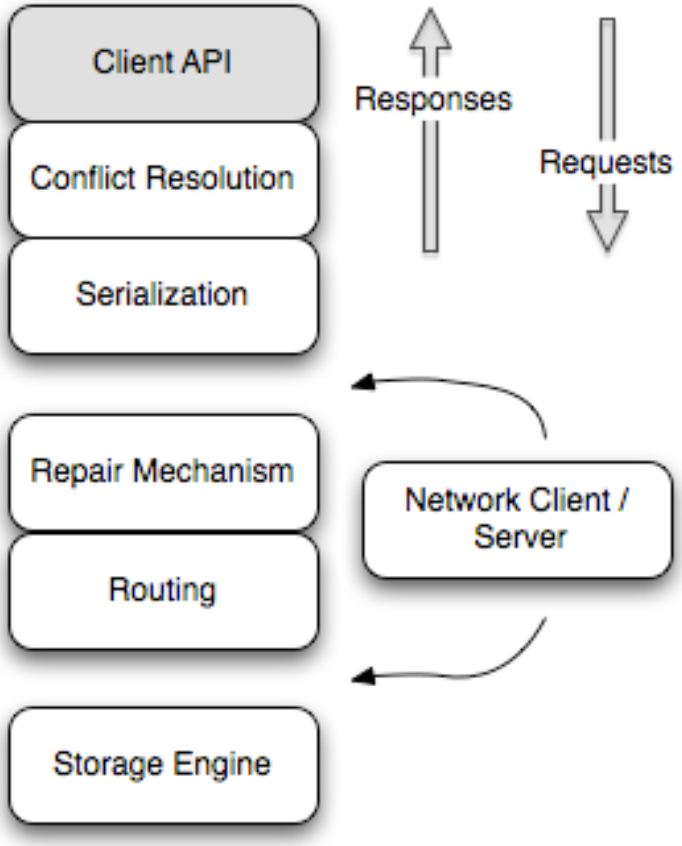

- Amazon Dynamo
  - Highly Available, Horizontal Scalable system
  - Key/Value model
  - Replication
  - All nodes are peers
  - Commodity Hardware
  - Simple to build
- Things to remember
  - Replication gives high availability but causes inconsistencies
  - Failures are fairly common in distributed systems
  - User must be isolated from these problems

# Start with the design



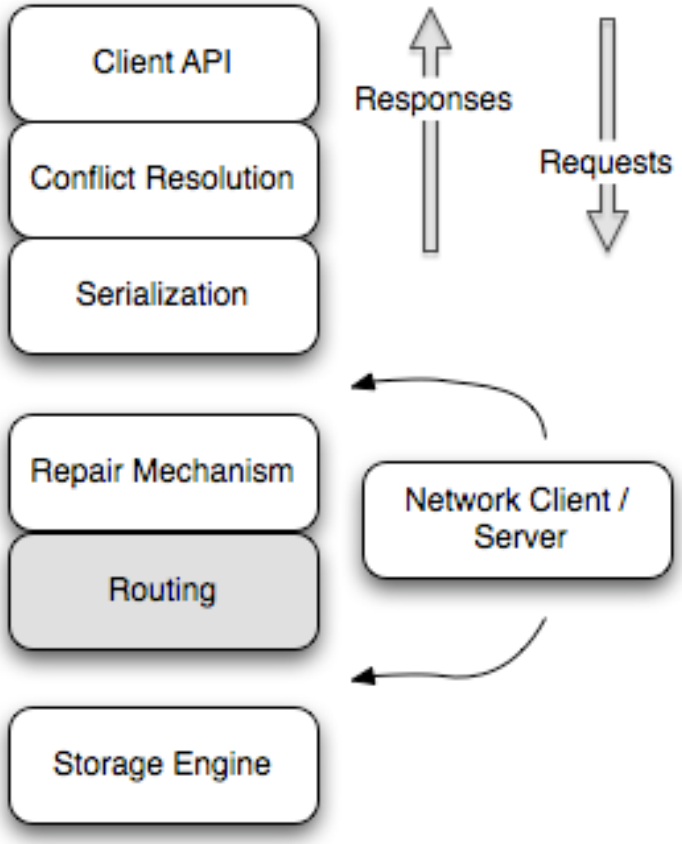

- Single interface for all components
  - get, put, getAll, delete
- Easy to test

# How do I talk to Voldemort?



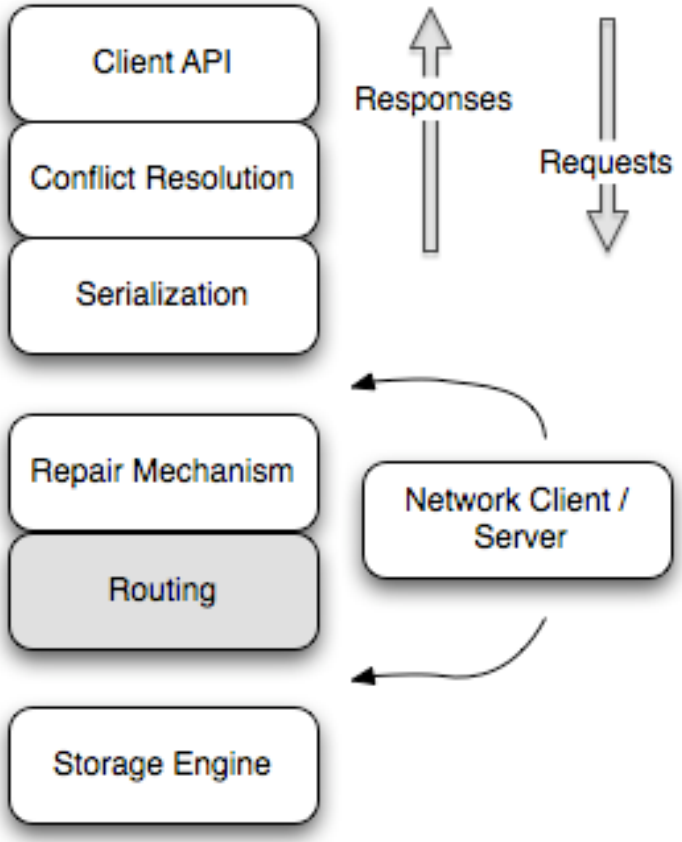

- DB Tables ~ Stores
- Key unique to a Store
- Operations
  - *GET*
  - *PUT*
  - *GETALL*
  - *DELETE*
  - *APPLYUPDATE* – Optimistic Locking

# Where does my data go?

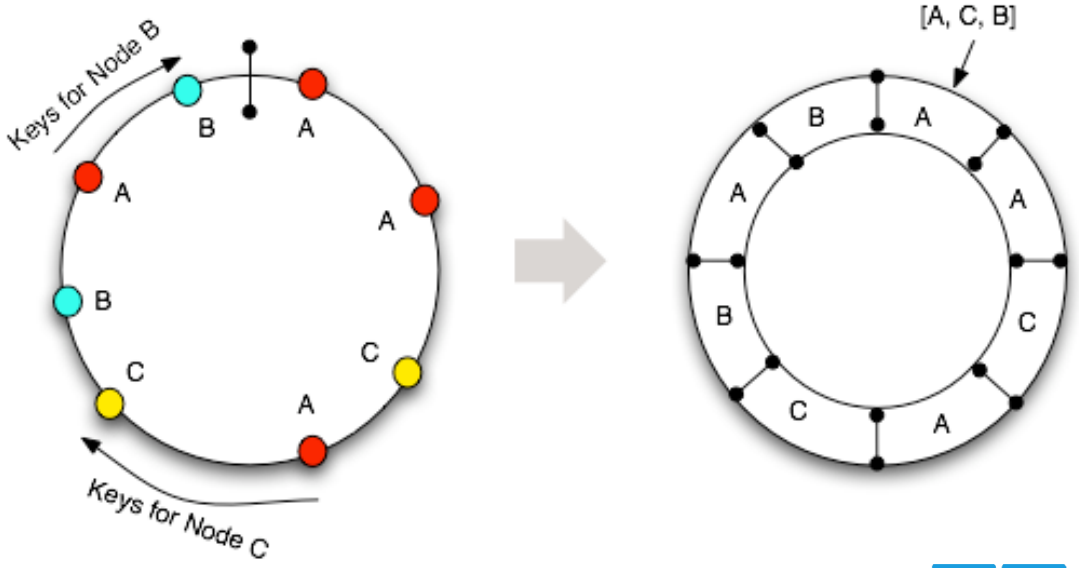



- Client or Server side
- Convert single *GET*, *PUT*, *DELETE* ops to multiple parallel ops
- Pluggable Routing strategy
  - Consistent Hashing
  - Zone aware Routing

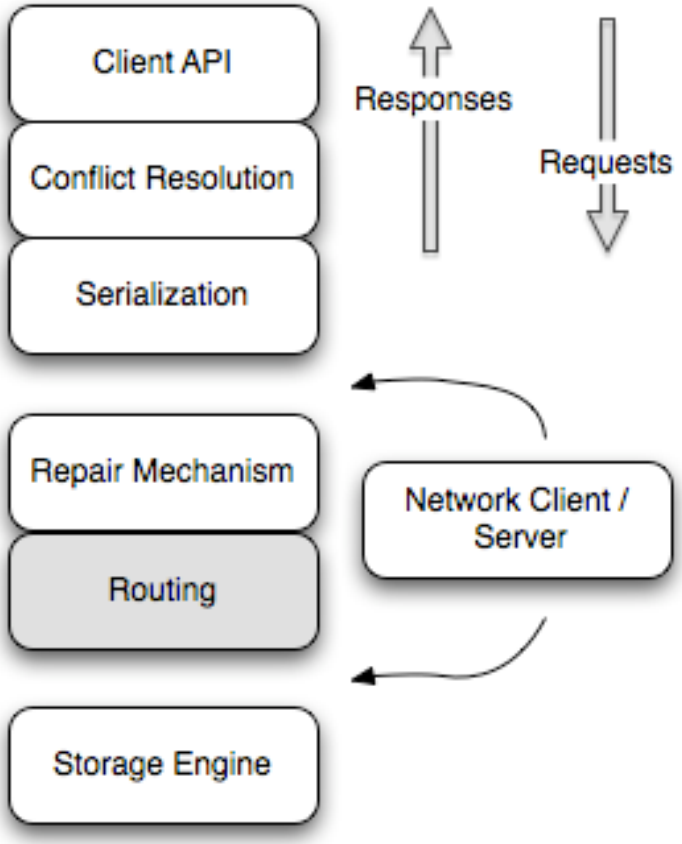
# What is Consistent hashing?




- Split hash ring into partitions
- Assign partitions to nodes
- Replicas = Next partition on different node

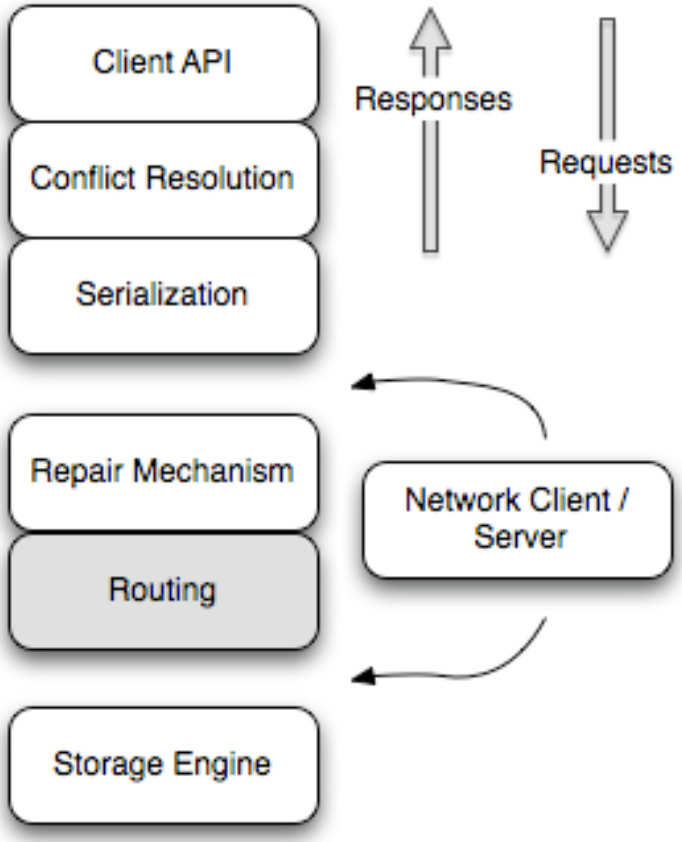


# What about routing parameters?



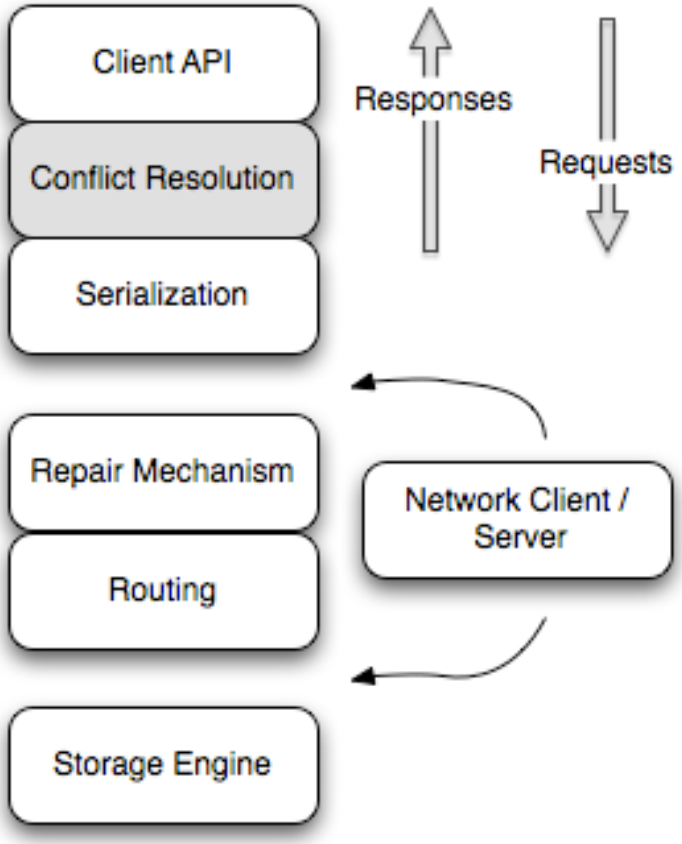

- Per store routing parameters
  - N - The replication factor (how many copies of each key-value pair we store)
  - R - The number of reads required
  - W - The number of writes we block for
- If R+W > N then we get to read our own writes

# And zone routing?



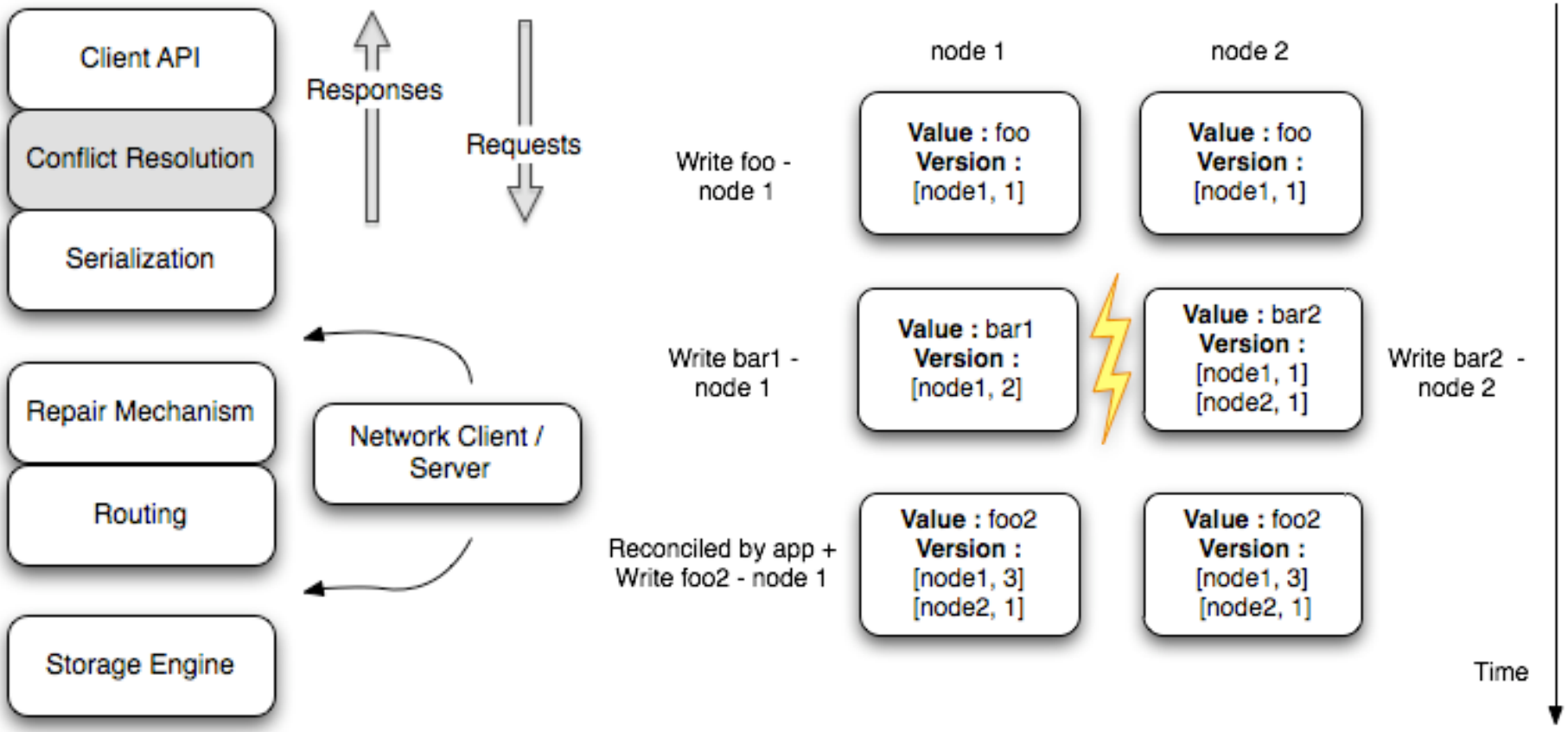

- Map nodes to zones (zone ~ datacenter, rack)
- Also provided is proximity list of zones
- In addition to N,R,W :
  - $Z_R$ , $Z_W$ - Zones to block

# Versioning the data

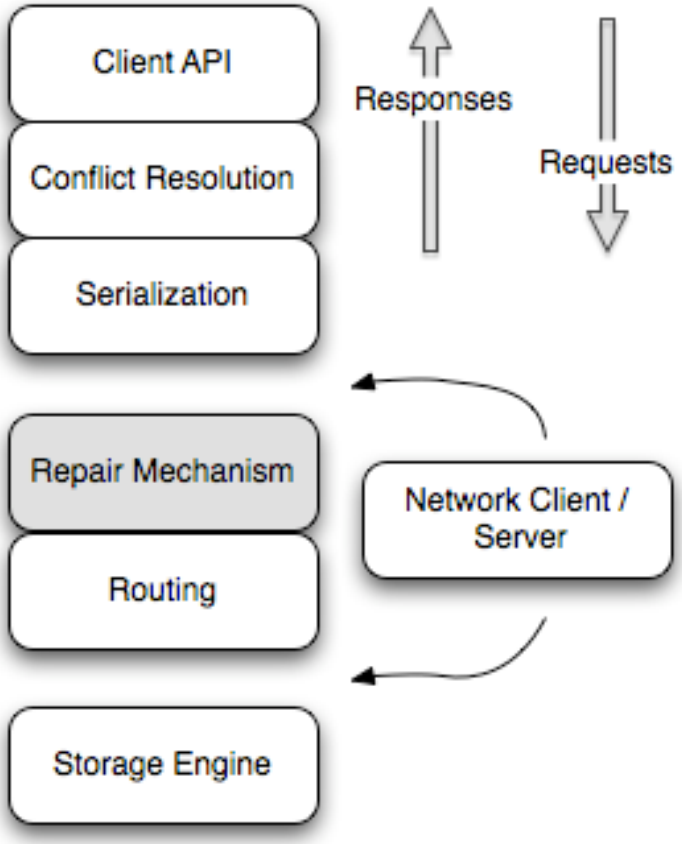



- Vector Clocks – Map<Node, Integer>
- Version every key/value
- What about concurrent writes?
  - Store all conflicting versions during writes
  - Client resolves them during reads
  - Pluggable resolver

# Versioning the data

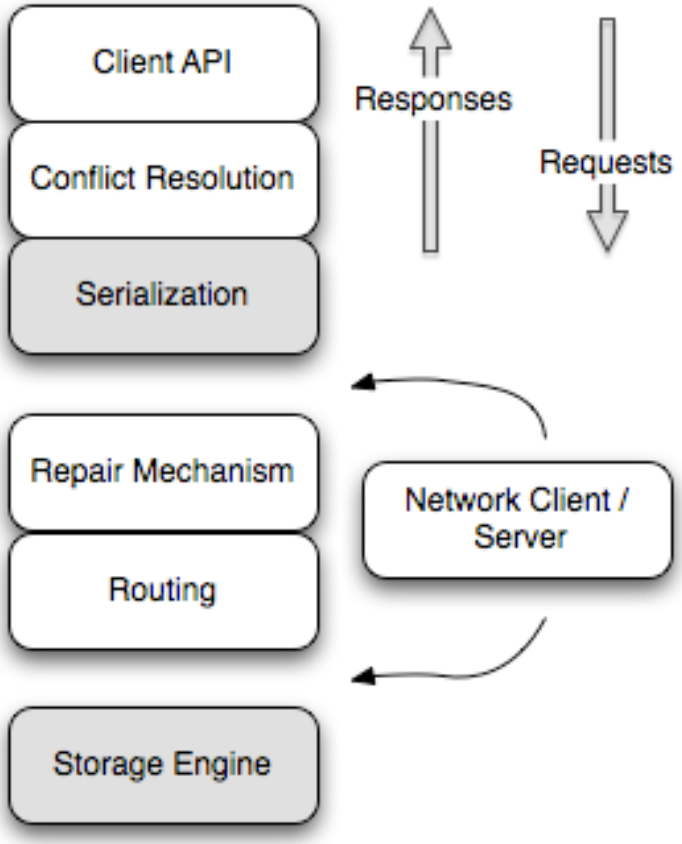


# How do we repair conflicting version?



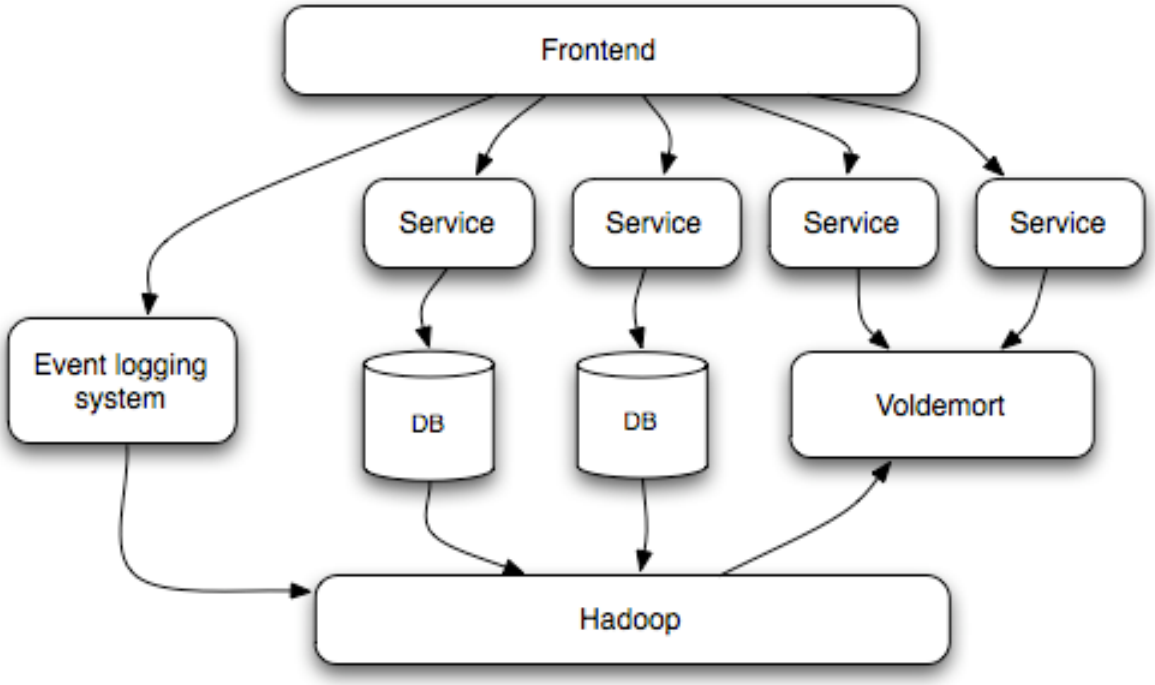

- Read Repair
  - Find inconsistent versions at read time
  - Asynchronously send back correct version
  - Max R network roundtrips

# Serialization & Storage



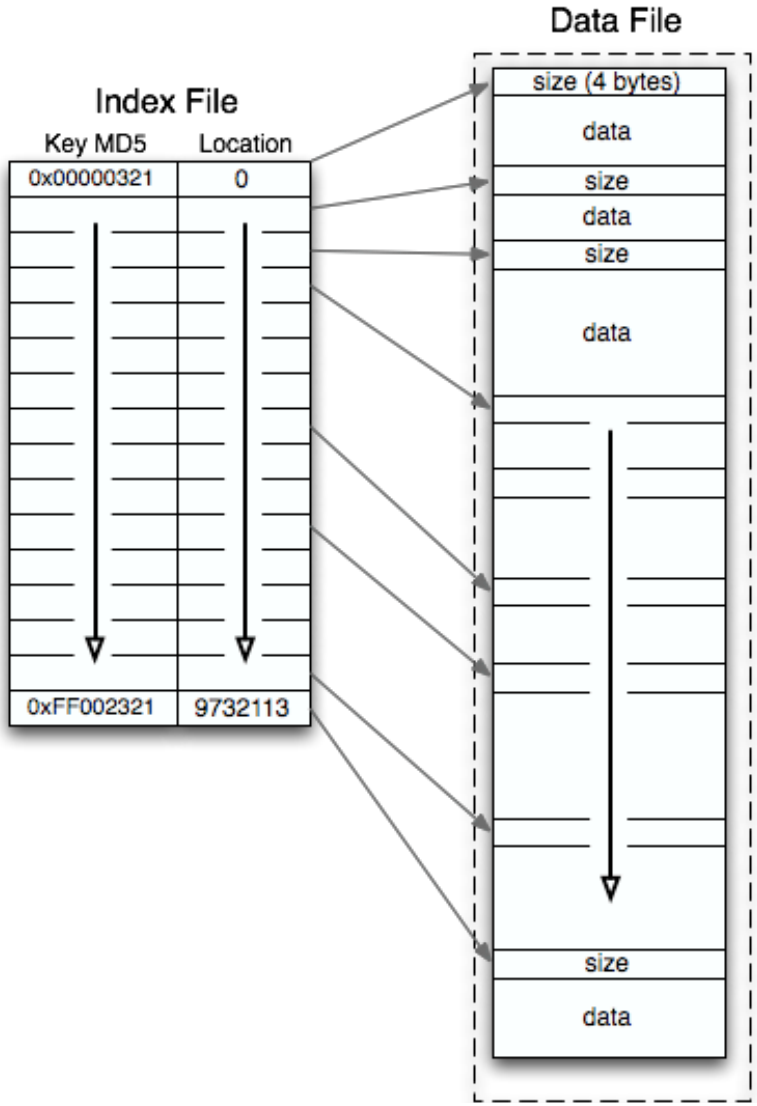

- Pluggable Serialization
  - Custom Binary JSON, Thrift, Protocol Buffers, Avro, Java serialization
- Pluggable Storage Engine
  - ConcurrentHashMap (great for testing), MySQL, BDB JE, Krati, Read-only

# Next problem, batch computed data



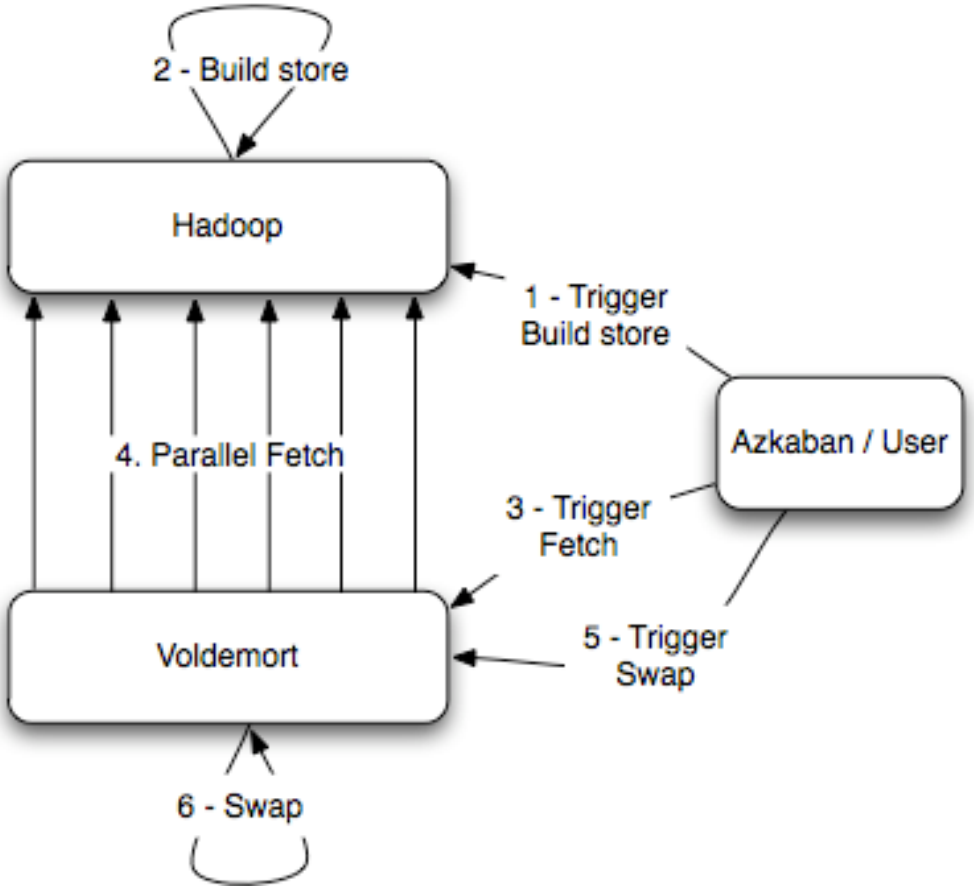

- Protect the live system
- Ability to rollback
- Failure tolerance
- Scalable – no bottleneck

# Read-only stores




- Build the Index offline
- Index structure
- Single Hadoop job
  - Input – any InputFormat
  - Output - Multiple "chunks" per partition (chunk ~ data + index file)
- Reads are fast
  - Cache warmness – Fetch the index files last
  - Memory map the .index files
  - Search – Binary / Interpolation

# Read-only stores




- No performance hit on the running DB
  - Store N different versions of data

```
store_name/
  version-0/
    0_0.data
    0_0.index
    <partition>_<chunk>.<data|index>
  version-1/
    ...
  version-2/
  latest->version-2/
```

  - Atomic Swap
  - Throttling
- Rollback - very quick!

# How does LinkedIn use this?


- Data dump to HDFS using Hadoop / Pig jobs
  - Binary JSON based OutputFormat
  - Custom Pig UDF which uses the above OutputFormat

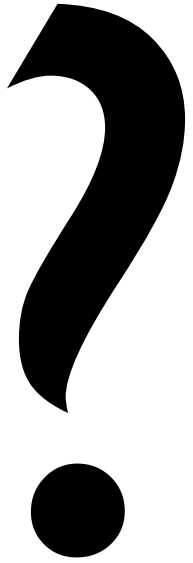
- Azkaban Job
  - Start store builder job on *input_data_path*
  - Trigger Fetch + Swap / Rollback on *voldemort_cluster_url*
  - Optional : Voldemort Sanity check (Sample gets)

# What else does Voldemort do?


- Monitoring stats via JMX
- Admin services
  - Allows adding, deleting stores without down-time
  - Retrieving, deleting, updating partitions
- Run Map Reduce on your data - ETL
- EC2 testing framework
- Server side transforms *
  - get(key, <function to run on server>)
- Rebalancing
  - Move a partition from one node to another
  - Add new nodes

# Future of Voldemort

- Publish Subscribe
- Other Repair mechanisms
- Incremental Pushes for Read-Only stores
- GUI



http://project-voldemort.com
http://github.com/voldemort/voldemort
http://sna-projects.com