NoSQL, Big Data, and all that

Alternatives to the relational model – past, present and future

Grant Allen – fuzz@google.com

Technology Program Manager, Principal Architect, Google

University of Cambridge, February 2014

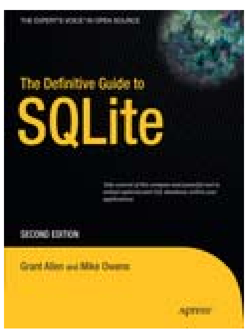
Agenda

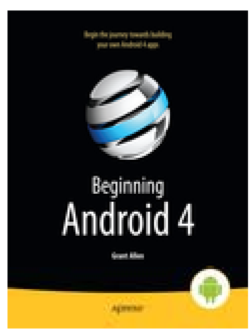
- Ubiquitous Intro
- The Genesis of NoSQL
- Architecture of NoSQL Databases
- Technical Implementation Details
- NoSQL Implications
- The Future of NoSQL

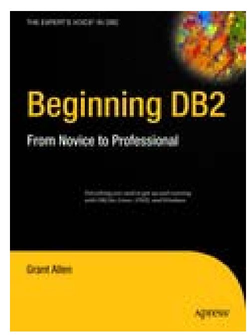
Intro - Who is Grant?

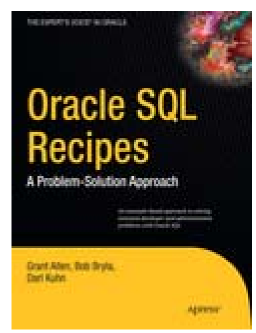
Grant Allen – @fuzzytwtr – fuzz@google.com

- Technology Program Manager & Data Architect at **Google**
- Works with all manner of databases
  - (Oracle, MySQL, Postgres, SQL Server, DB2, Informix, Sybase, SQLite, BigTable and successors, HBase, MongoDB, Cassandra, etc. etc.)
- Talks regularly at conferences, universities, etc.
- Writes about all sorts of things









The Genesis of NoSQL

The Genesis of NoSQL and “Big Data”

Google

- The relational model is very successful
  - Strong theoretical foundations – relational calculus and algebra (Codd, Date, etc.)
  - The relational model is infinitely scalable in theory (models are nice like that).

    Not concerned with:
    - Storage capacity
    - Processing capacity
    - Communication capacity
  - Desirable qualities, ACID, etc.

The Genesis of NoSQL and “Big Data”

- A model and its implementation are not the same, e.g.
  - Computing resources are finite in various ways, and imperfect
  - Relational calculus/algebra does not cover all interesting cases
    - SQL != Relational calculus/algebra, and is also not “complete”
  - Should one pay the price for qualities/attributes that are unimportant or unused?
    - What if I don’t care about consistency, isolation, etc.?
    - Why bother with concurrency control for logically isolated work?
- Putting it another way, we’re looking at the difference between computer science vs computer (software) engineering

The Genesis of NoSQL and "Big Data"

- Some examples
  - Current approximate size of the internet (2013): **60* trillion pages**
  - Internet use (2011): **3+ billion people**
  - Typical query: **funniest cat video**
- The relational approach can answer the query, in theory
- But consider
  - Annual component failure rate in typical hardware (2007,2012): **>0.5%**
  - Time to read 60 trillion 1MB pages (60 Exabytes!!): **??**
  - While implementing concurrency: **millions of locks, more time?**
- Can you answer the query ***before your hardware dies***?

The Genesis of NoSQL and “Big Data”

- 60EB of data
    - 20 million 3TB disks
    - “*Failure Trends in Large Disk Drive Populations*” (2007)
    - 18000 disks will fail every day
    - 18000TB (18PB) of data will be lost (not just to your query)
- Not even beginning to worry about:
    - Can I build a machine with 20 million disks?
    - And power it?
    - Etc.

The Genesis of NoSQL and “Big Data”

- Solutions
  - Distribute the data to more realistic hardware
  - Compensate for imperfect hardware with software fault tolerance
  - Use software to bridge the distributed nature of the data
  - Sacrifice/remove unneeded (or little needed) qualities

The Genesis of NoSQL and "Big Data"

- Solutions
  - Distribute the data to more realistic hardware
  - Compensate for imperfect hardware with software fault tolerance
  - Use software to bridge the distributed nature of the data
  - Sacrifice/remove unneeded (or little needed) qualities
- Interesting consequences
  - Fault tolerant software can work with all kinds of hardware
    => commodity hardware
  - The software model can encompass more than just the "database"
    => bespoke filesystems, abandon generic platforms
  - Distributed data challenges monolithic software engineering
    => massively parallel software matched to massively distributed data

Architecture of NoSQL Databases

Initial premise of NoSQL

- Solve the issues of very large data on imperfect machines

- Size of the data exceeds the technical limitations of relational databases
    - Not just one's appetite for licencing costs

- Desirable properties of traditional databases hinder scaling
    - What useful things can we achieve with tradeoffs?

- For some subset of applications, perfection is not necessarily a goal

NoSQL-Style implementations

- Starting with Google, and encompassing others
    - Google BigTable, GFS, (MapReduce) => Spanner, successors
    - Hadoop HBase, HDFS
    - Cassandra
    - MongoDB
    - CouchDB
    - Others

High-level conceptual design of NoSQL Databases

- High-level "database" layer
  - Sparse row-column store (BigTable/Spanner, HBase)
  - Key-value store (Cassandra)
  - Document store (MongoDB)
  - Others such as graph stores
  - Good for various "read" workloads, simple discrete "write" workloads.
  - Poor for complex/large write workloads

- Low-level filesystem/storage layer
  - Bespoke filesystem (GFS, HDFS, S3)
  - POSIX-style filesystem (EXT*n*, XFS, JFS, NTFS etc.)

- "Bring/build your own query tools" – SQL-like tools absent or nascent
  - MapReduce, Dremmel

Technical Implementation Details

The Anatomy of a NoSQL Database

- Using Google’s BigTable, GFS and MapReduce as an example
- Concepts applicable to almost all NoSQL databases
  - 1:1 equivalence for Hadoop, etc.
  - Distributed storage
  - Replication
  - Faults *always* happen

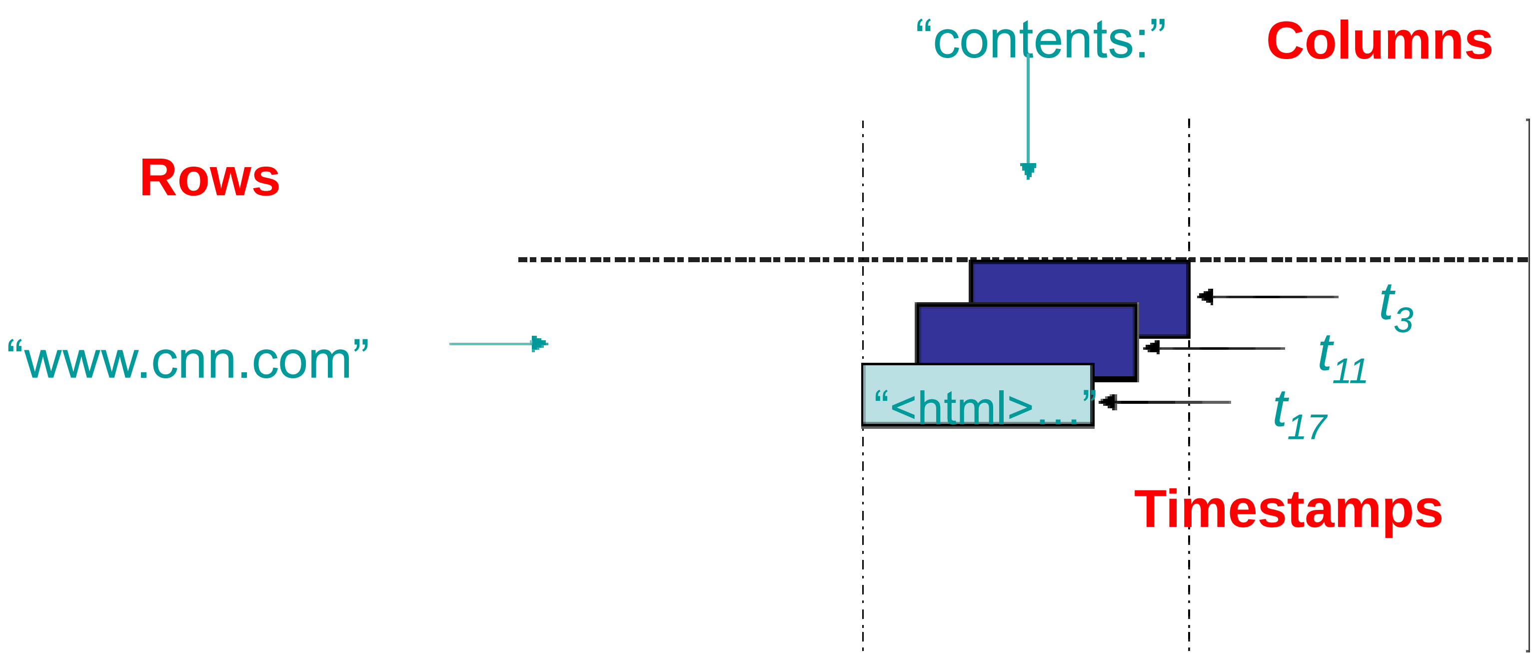
BigTable

- Higher level API than a raw file system
  - o Somewhat like a database, but not as full-featured
- Useful for structured/semi-structured data
  - o URLs:
    - Contents, crawl metadata, links, anchors, pagerank, …
  - o Per-user data:
    - User preference settings, recent queries/search results, …
  - o Geographic data:
    - Physical entities, roads, satellite imagery, annotations, …
- Scales to large amounts of data
  - o trillions of URLs, many versions/page (~20K/version)
  - o billions of users, millions of q/sec
  - o PetaBytes+ of satellite image data

BigTable

- Distributed multi-dimensional sparse map

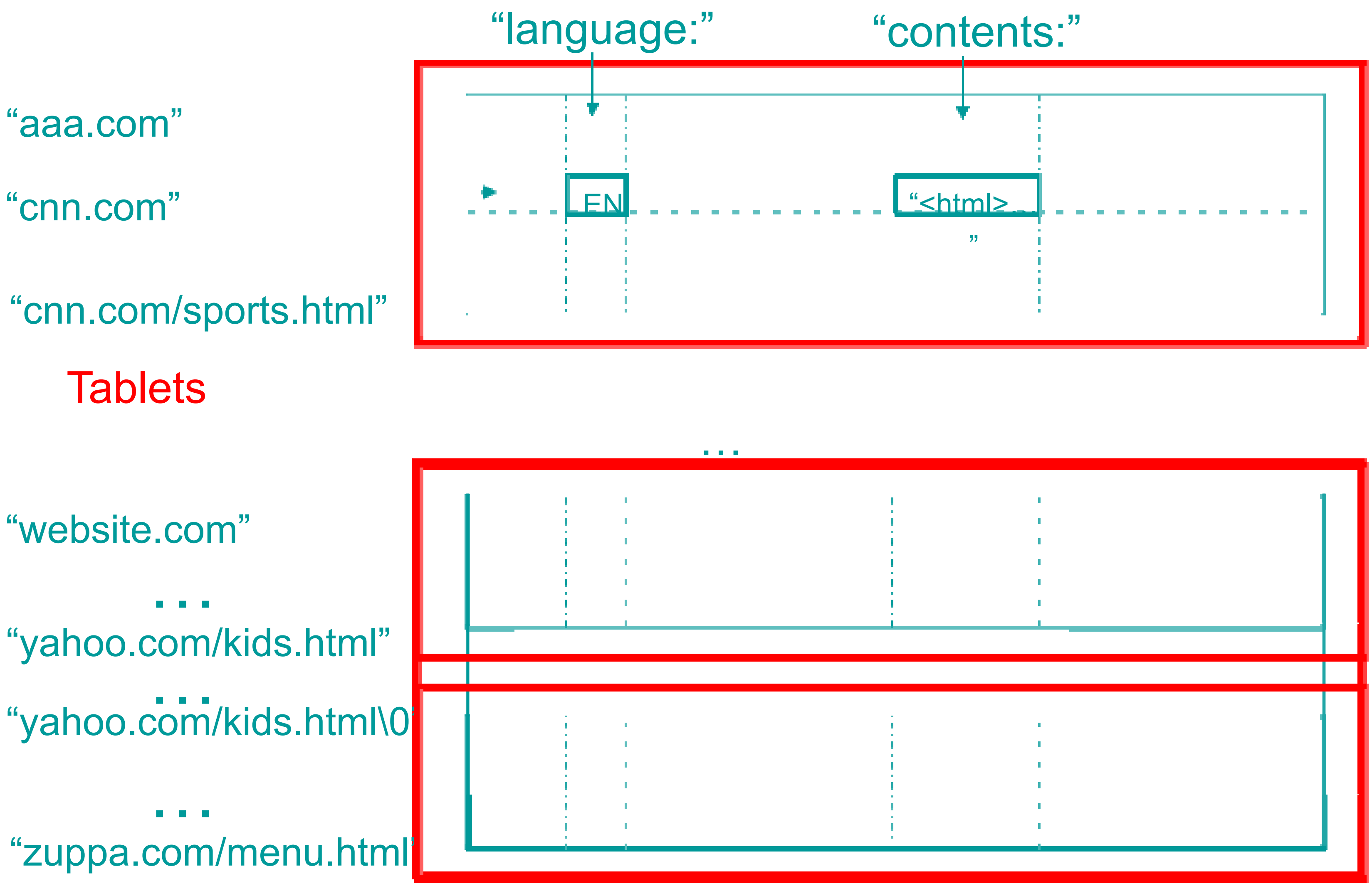
  (row, column, timestamp) >>> cell contents



BigTable - Tablets

- Large tables broken into tablets at row boundaries
  - Tablet holds contiguous range of rows
    - Clients can often choose row keys to achieve locality
  - Aim for ~100MB to 200MB of data per tablet
- Serving machine responsible for ~100 tablets
  - Fast recovery:
    - 100 machines each pick up 1 tablet from failed machine
  - Fine-grained load balancing:
    - Migrate tablets away from overloaded machine
    - Master makes load-balancing decisions

BigTable – Tablets and Splitting

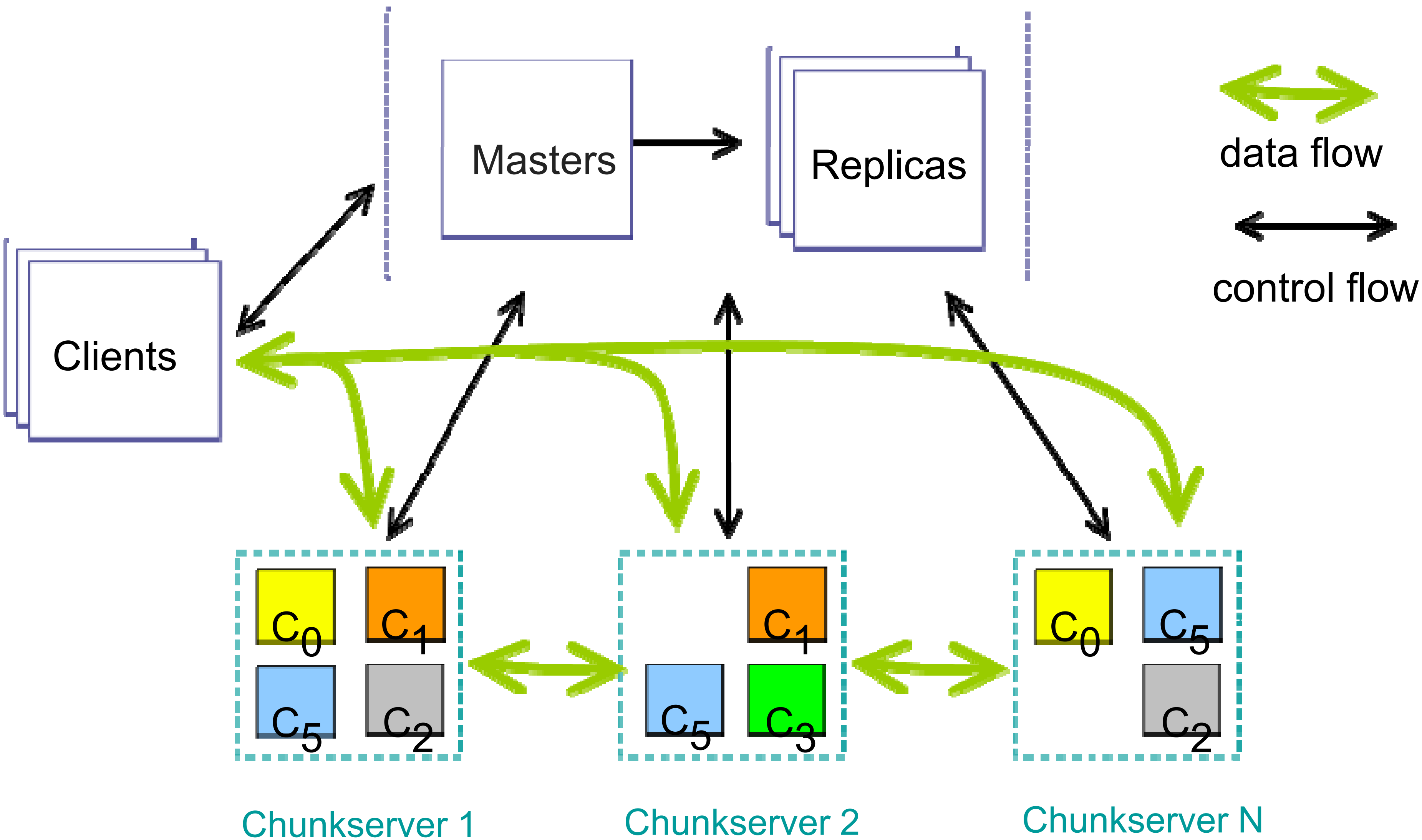


GFS – Google File System

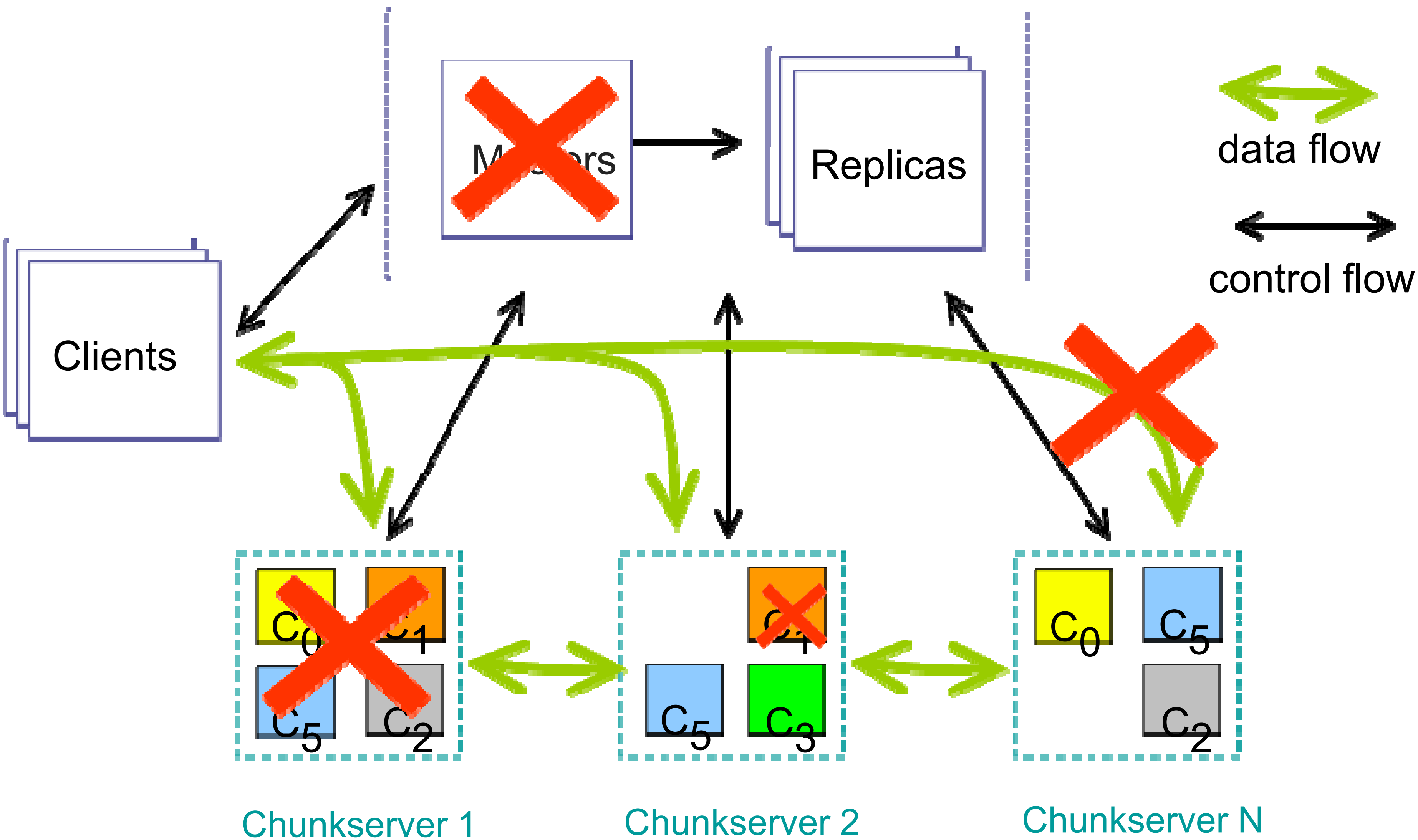
- Why develop a bespoke file system?

- Working with large data has unique FS requirements
  - Huge read/write bandwidth
  - Reliability over thousands of nodes
  - Mostly operating on large data blocks
  - Need efficient distributed operations
  - Scale - Unprecedented!!!

GFS – Architecture



GFS – Failure Scenarios



GFS – Software fault tolerance

- Typical forms of frequent failure (as already mentioned)
  - Disks, servers, networks, software bugs
- Basic strategies
  - Checksum everything
  - Replication to allow recovery
- Chunks are replicated on different chunkservers
  - Default is 3x, but configurable

MapReduce

There is no standard query language!

A simple programming model that applies to many large-scale computing problems

Evolving MapReduce libraries incorporate:

- automatic parallelisation
- load balancing
- network and disk transfer optimization
- handling of machine and task failures

Typical Problems Solved by MapReduce

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
- Write the results

Outline stays the same,
map and reduce change to fit the problem

MapReduce More Formally

Programmer specifies two primary methods:

- map(k, v) → <k', v'>*
- reduce(k', <v'>*) → <k', v'>*

All v' with same k' are reduced together, in order.

Usually also specify:

- partition(k’, total partitions) -> partition for k’
  - often a simple hash of the key
  - allows reduce operations for different k’ to be parallelised

Typical Problems Solved by MapReduce

Input

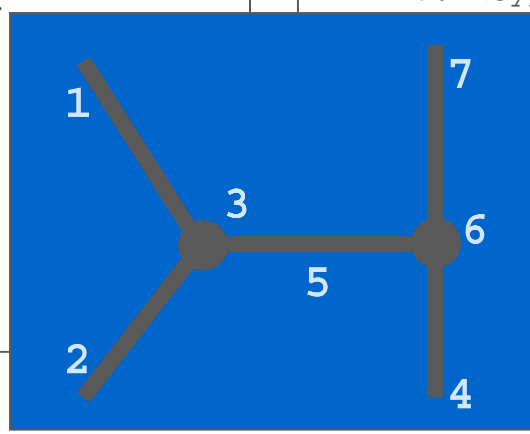
Feature List

```
1: <type=Road>, <intersections=(3)>, <geom>, …
2: <type=Road>, <intersections=(3)>, <geom>, …
3: <type=Intersection>, <roads=(1,2,5)>, …
4: <type=Road>, <intersections=(6)>, <geom>,
5: <type=Road>, <intersections=(3,6)>, <geom>, …
6: <type=Intersection>, <roads=(5,6,7)>, …
7: <type=Road>, <intersections=(6)>, <geom>, …
8: <type=Border>, <name>, <geom>, …
.
.
.
```

Output

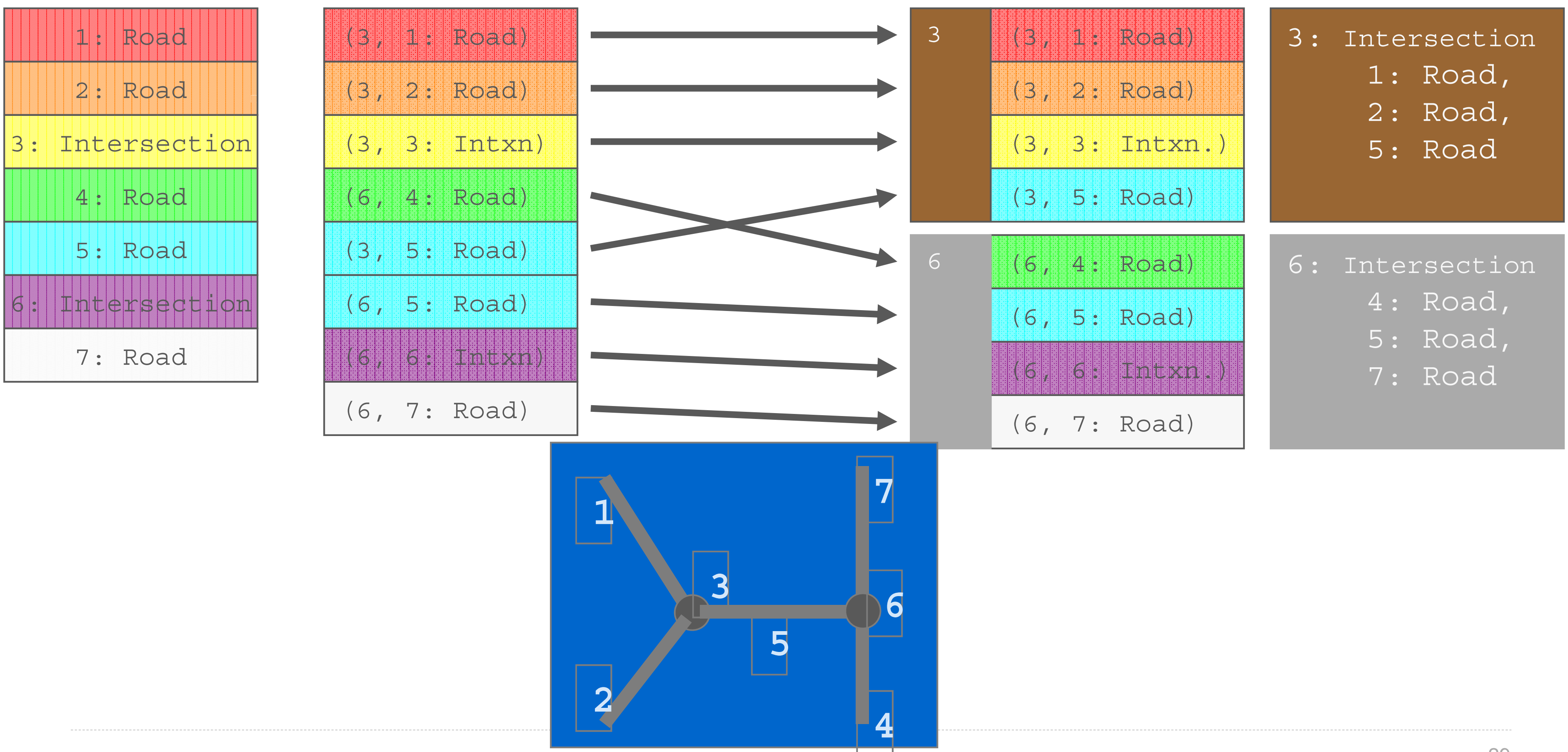
Intersection List

```
3: <type=Intersection>, <roads=(
     1: <type=Road>, <geom>, <name>, …
     2: <type=Road>, <geom>, <name>, …
     5: <type=Road>, <geom>, <name>, …)>, …
6: <type=Intersection>, <roads=(
     4: <type=Road>, <geom>, <name>, … >
     5: <type=Road>, <geom>, <name>, … >
     7: <type=Road>, <geom>, <name>, …)>, …
.
.
.
```



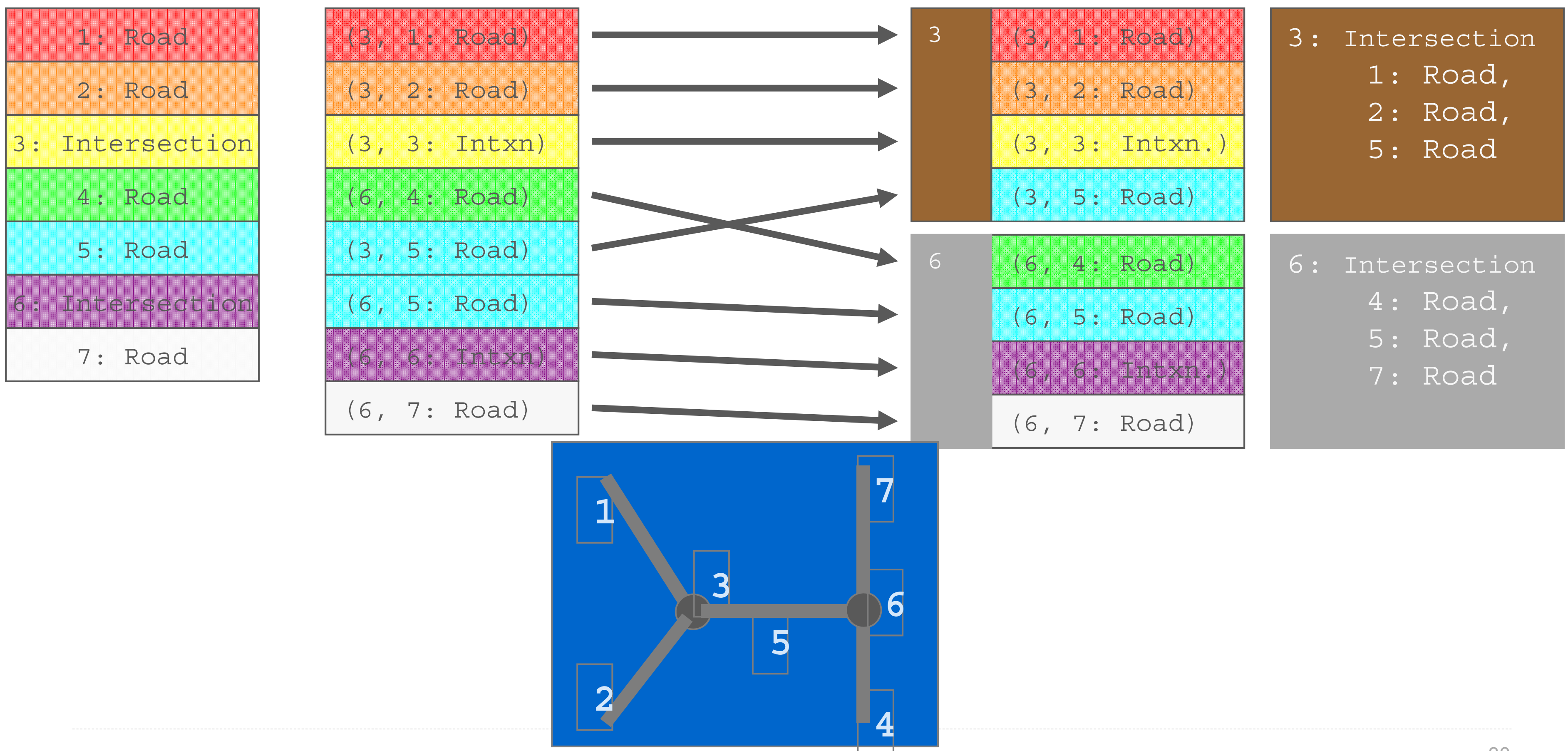
Typical Problems Solved by MapReduce

Input	**Map**	Shuffle	**Reduce**	Output
List of items	Apply **map()** to each; emit (key,val) pairs	Sort by key	Apply **reduce()** to list of pairs with same key	New list of items



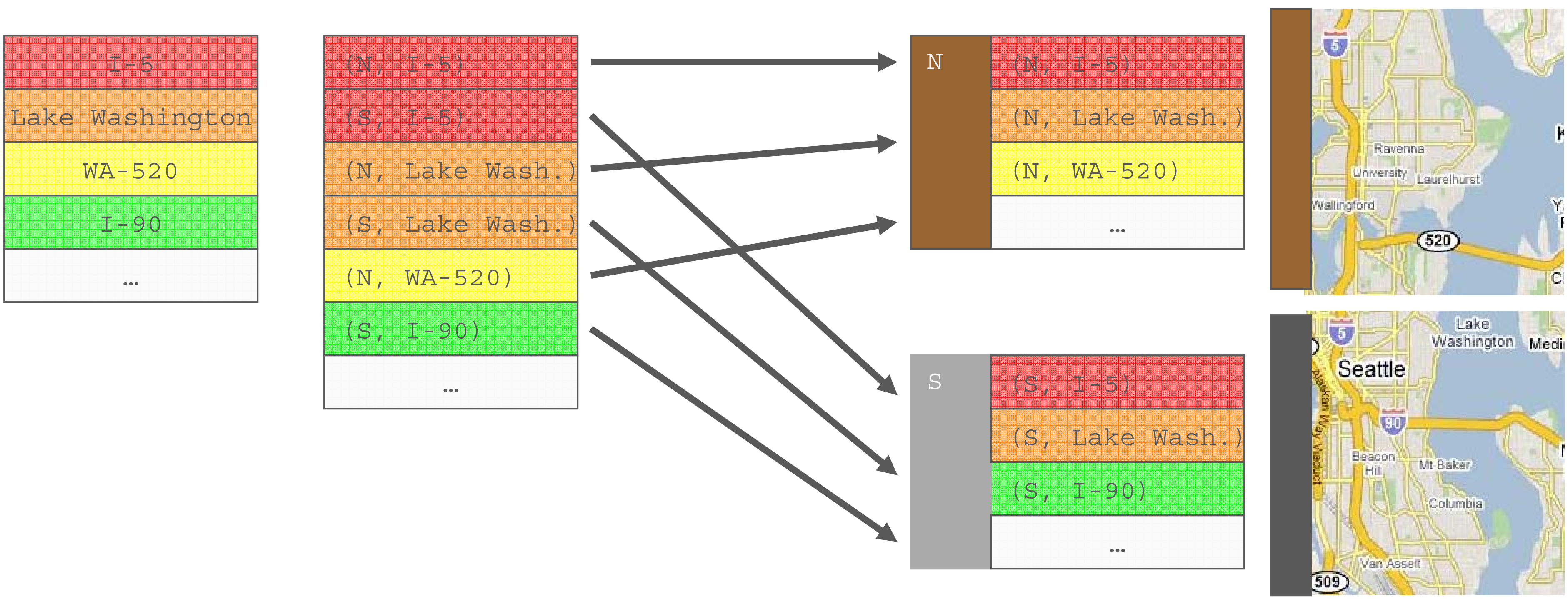
Typical Problems Solved by MapReduce

Input	Map	Shuffle	Reduce	Output
List of items	Apply **map()** to each; emit (key,val) pairs	Sort by key	Apply **reduce()** to list of pairs with same key	New list of items



Typical Problems Solved by MapReduce

Input	Map	Shuffle	Reduce	Output
Geographic feature list	Emit each to all overlapping latitude-longitude rectangles	Sort by key (key= Rect. Id)	Render tile using data for all enclosed features	Rendered tiles



NoSQL Implications

Software Fault Tolerance

- At the “database” tier (BigTable, HBase), high availability/fault tolerance is almost always the result of redundant replicas, and optionally idempotent tasks

- Replicas are not necessarily all at the same timestamp/transaction
  - What view of the data is considered “consistent”? => Eventual consistency
  - See also the previously mentioned timestamping of a given cell in BigTable, Hbase, etc.

- The loss of any one node or replica is considered normal
  - Rebuild or redistribute responsibility

- The failure of any one Map/Reduce task is considered normal
  - Restart / resume

ACID

ACID

Eventual Consistency

ACID

- Implications of Eventual Consistency
  - Conflict resolution
  - Data Ambiguity / Precision
  - Quorum-based certainty (e.g. CassandraDB)
  - Skews suitability to “information retrieval” workloads where loss of precision is tolerable =>
    Good for finding most funny cat videos,
    Bad for exactly managing bank accounts.

The Future of NoSQL

The Future of NoSQL

(You're Going to Need a Bigger Boat)

The Future of NoSQL

- Evolving standards
  - E.g. Dremel, a SQL-like declarative language and query client
- Traditional RDBMS vendors incorporating NoSQL concepts and capabilities
  - Akin to their encompassing of Java, XML and other supposed paradigm-shifters
- Maturing understanding that NoSQL suits only a limited number of use cases
  - The relational model will live on and grow.
- Maturing theoretical models for NoSQL
  - CAP Theorem – Consistency, Availability, (Partition) Tolerance, choose any two.
- Advanced NoSQL Implementations evolving back into Relational DBMS!
  - Google’s F1

Thank You

Q&A