Design Decisions and Trade-offs in Apache Accumulo

Aaron Cordova
CTO Koverse Inc.
33rd International Conference Massive Storage Systems and Technology

KOVERSE

“Design is not just what it looks like and feels like. Design is how it works.”

–Steve Jobs

“There ain’t no such thing as a free lunch.”

–Milton Friedman, others

2003 Mountain View California

WebSearch for a Planet: the Google Cluster Architecture

"... the most important factors that influence its design: energy efficiency and price-performance ratio."

"… we provide reliability in software rather than in server-class hardware, so we can use commodity PCs to build a high-end computing cluster at a low-end price."

WebSearch for a Planet: the Google Cluster Architecture

Different queries run on different processors

Partitioned Index

A single query uses multiple processors

More than 15,000 commodity-class PCs

Fault-tolerance built into software

Superior performance at a fraction of the cost of a system built from fewer, but more expensive high-end servers

2005 Jeffrey Dean, University of Washington

BigTable

Motivation

lots of semi-structured data behind google apps

multiple versions of crawled web pages

user information

satellite imagery

geographical data

100s of millions of users, many queries per second


“scale is too large for most
commercial databases”

“even if scale were not too large, the cost would be very high …”

“… requiring high-end hardware that doesn’t match well with infrastructure”

“building internally means system can be applied across many projects for low incremental cost”

“low-level storage optimizations help performance significantly”

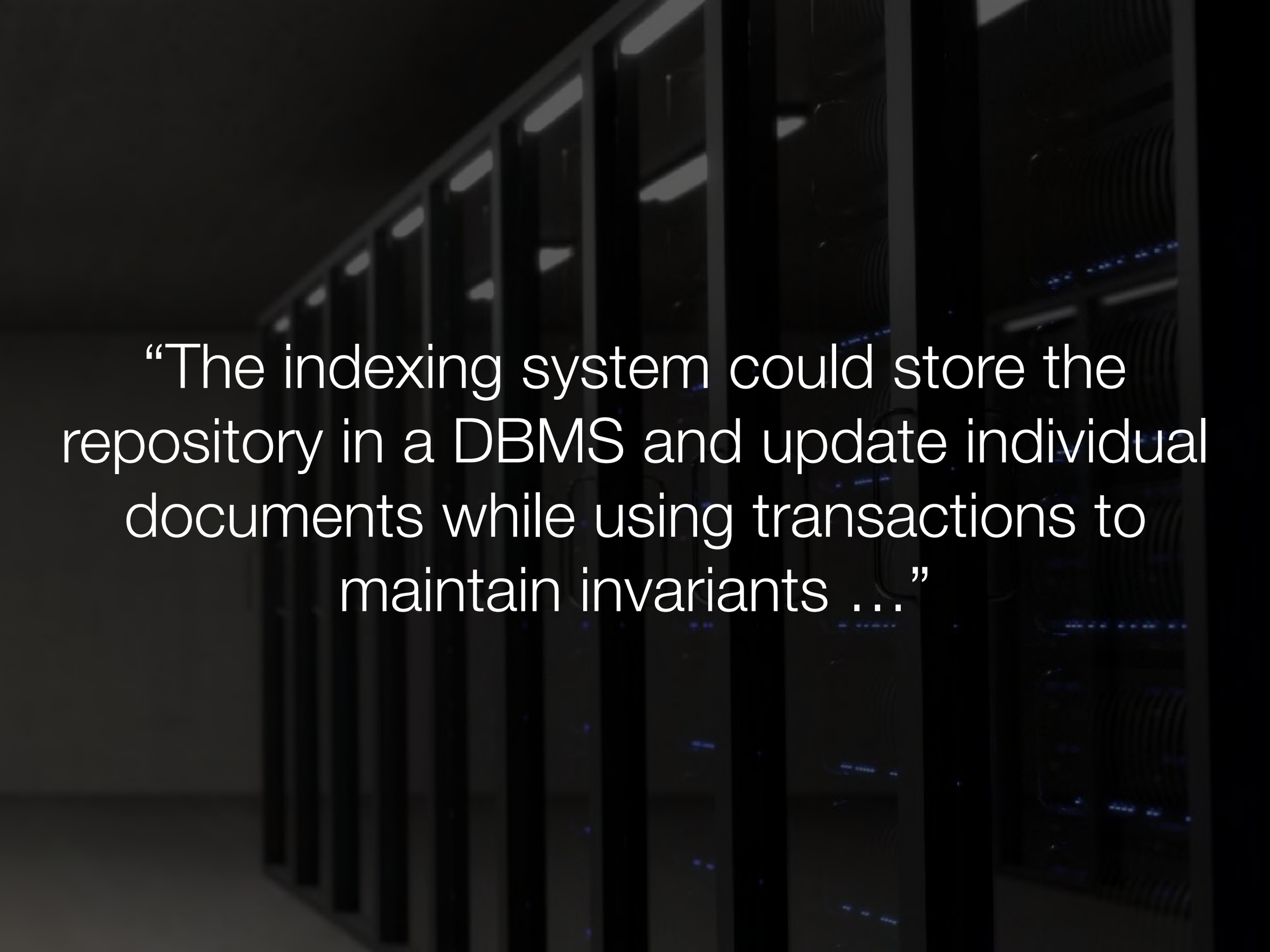
“because we're able to develop code at all levels, can take advantage of storage and network transfer optimizations, much harder to do when running on top of a database layer”


“also fun and challenging to
build large scale systems”

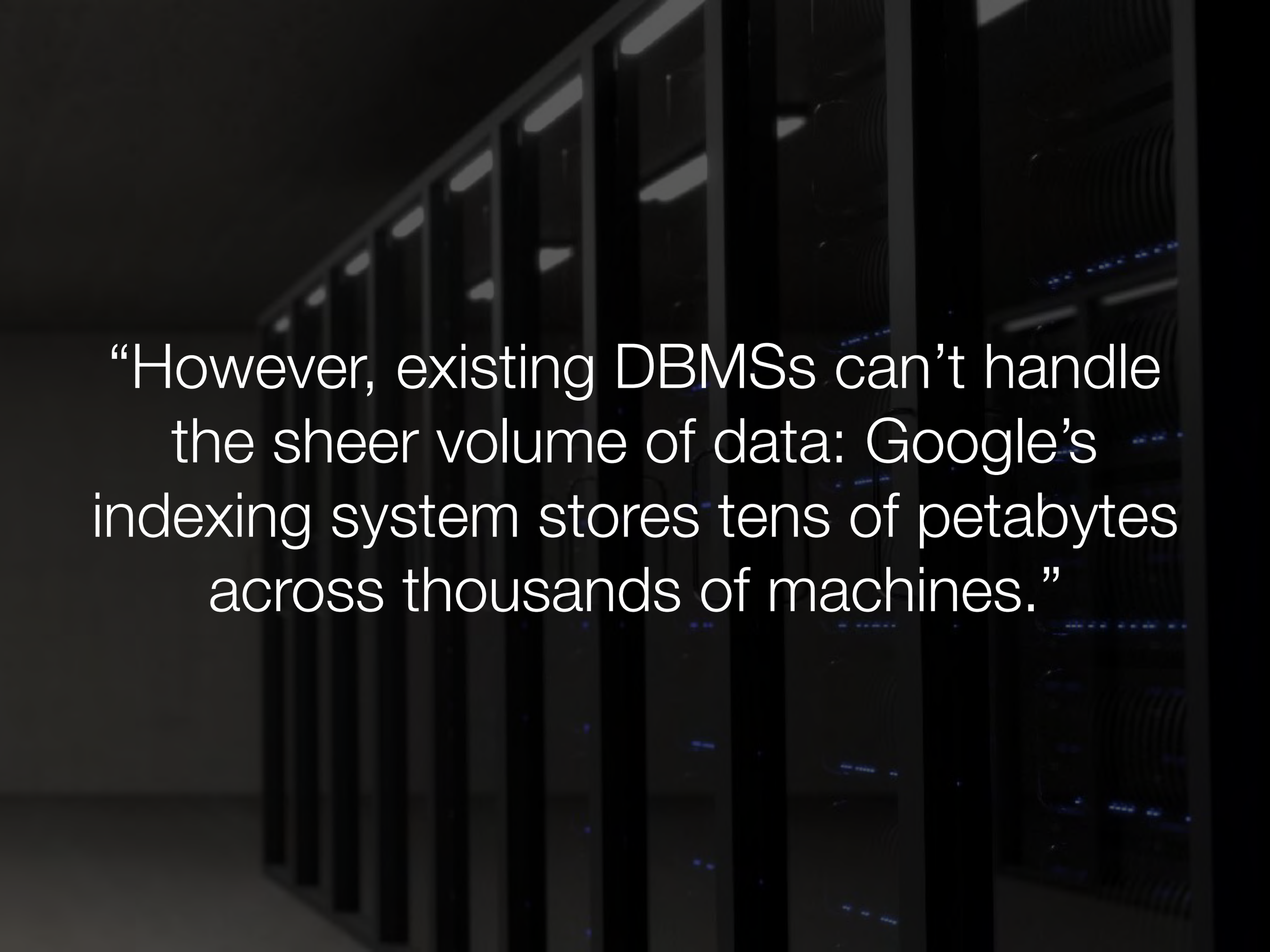
Large-scale Incremental Processing Using Distributed Transactions and Notifications

2010 OSDI

Percolator, a system for incrementally processing updates to a large data set, is used to produce Google's websearch index, persisted in BigTable



"The indexing system could store the repository in a DBMS and update individual documents while using transactions to maintain invariants …"



"However, existing DBMSs can't handle the sheer volume of data: Google's indexing system stores tens of petabytes across thousands of machines."

BigTable Design Objectives

BigTable Design Objectives

want a lot of asynchronous processes to continuously update and read from their part of the global state

want access to most current data at any time

need high read/write rates

efficient scans over all or interesting subsets of data

efficient joins of large one-to-one and one to many data sets

want to examine data changes over time

BigTable Design Decisions

Highly consistent, not eventually consistent

Designed for a single data center, not geographically distributed data centers

Keys organized via sorting, partitioned into ranges, not hashing

Service of each range is decoupled from storage, reassignment doesn’t require data movement

Support for single-row transactions

BigTable Features

Distributed multi-level map - interesting data model

Fault tolerant, persistent

Scalable

- Thousands of servers
- Terabytes of in memory data
- Petabytes of disk based data
- Millions of reads and writes per second, efficient scans

Self managing

- Servers can be added / removed dynamically
- Servers adjust to load imbalance

BigTable Data Model

Key	Value
(sorted)	(not sorted)

BigTable Data Model

Key			
row ID	**Column**	**Timestamp**	Value

Key consists of three main components

BigTable Data Model

columns

	age	phone	sneakers	hat
bill	49	555-1212	$100	-
george	38	-	$80	$30
george	37	-	$80	$30

rows

time

BigTable Data Model

row	column	time	value
bill	age	Jun 2010	49
bill	phone	Jun 2010	555-1212
bill	sneakers	Apr 2010	$100
george	age	Oct 2009	38
george	sneakers	Nov 2009	$80
george	hat	Dec 2009	$30

BigTable Data Model

A read request for a single key or a range of keys is routed to one server, and is designed to involve a minimal amount of seeks on a cheap spinning disk, read data sequentially, and return, typically in less than a second.

BigTable loads key value pairs into memory in blocks, and caches recently read blocks, enabling applications to exploit temporal and spatial locality in access patterns

BigTable Data Model

Key				Value
row ID	Column		Timestamp	
	Family	**Qualifier**		

Column is split into two additional components

BigTable Data Model

Column families can be assigned to locality groups which are stored together on disk.

This allows scanning columns within a locality group without reading other columns from disk.

Locality groups can be marked as being served from memory, loaded lazily.

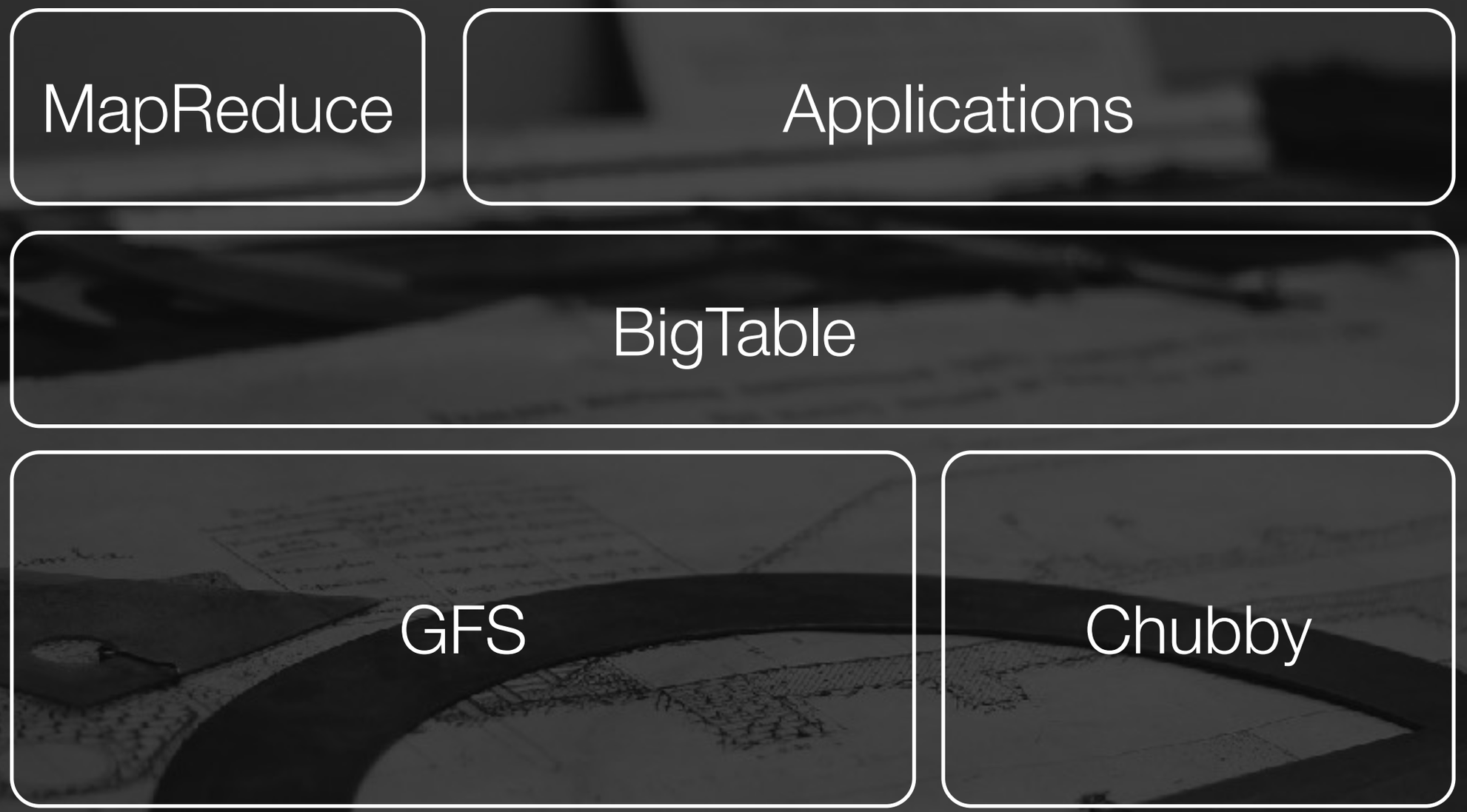
BigTable Data Model

Column families must be declared before hand, but column qualifiers do not, can be dynamically created during ingest.

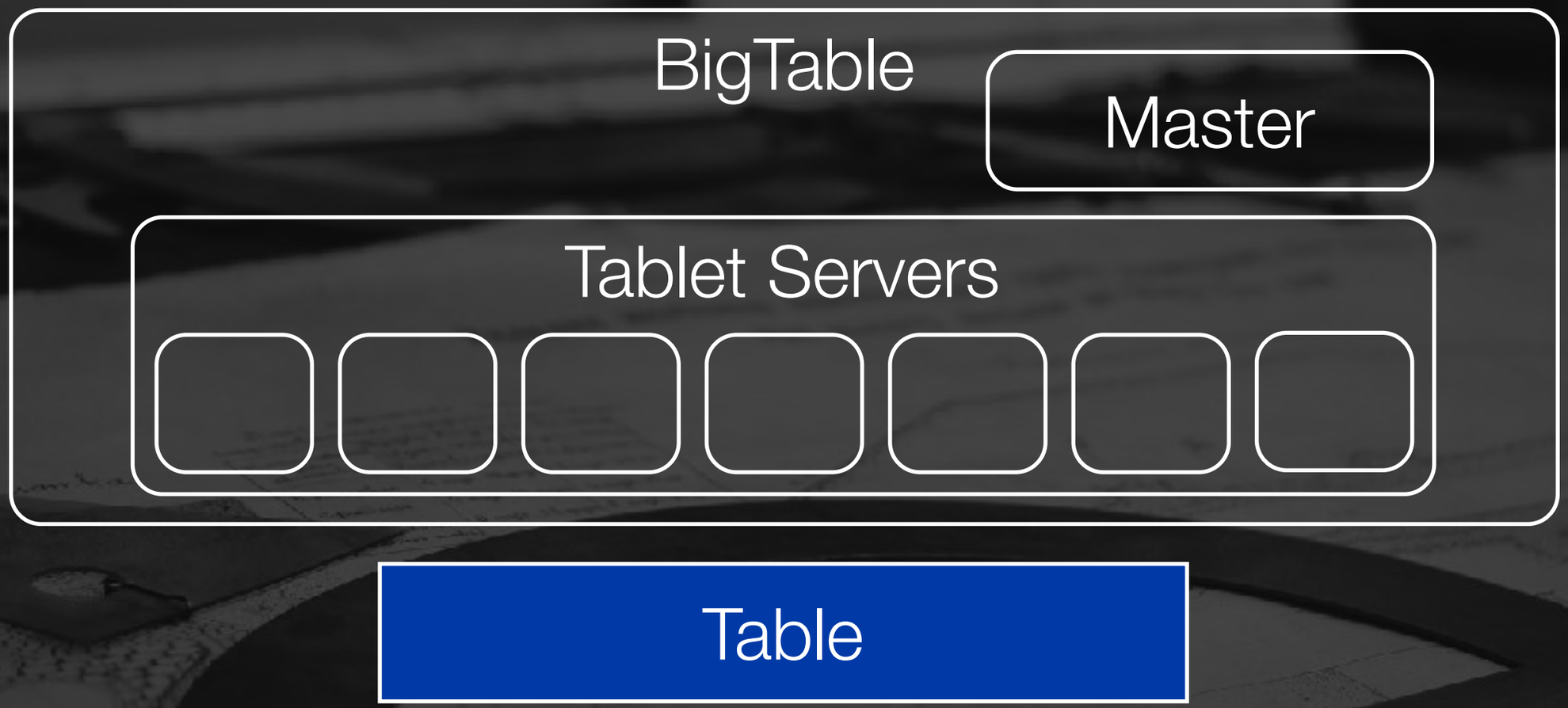
Rows can be very large, millions of columns or more.

Rows within a table need not all have the same set of columns. No penalty for highly sparse and dynamic data.

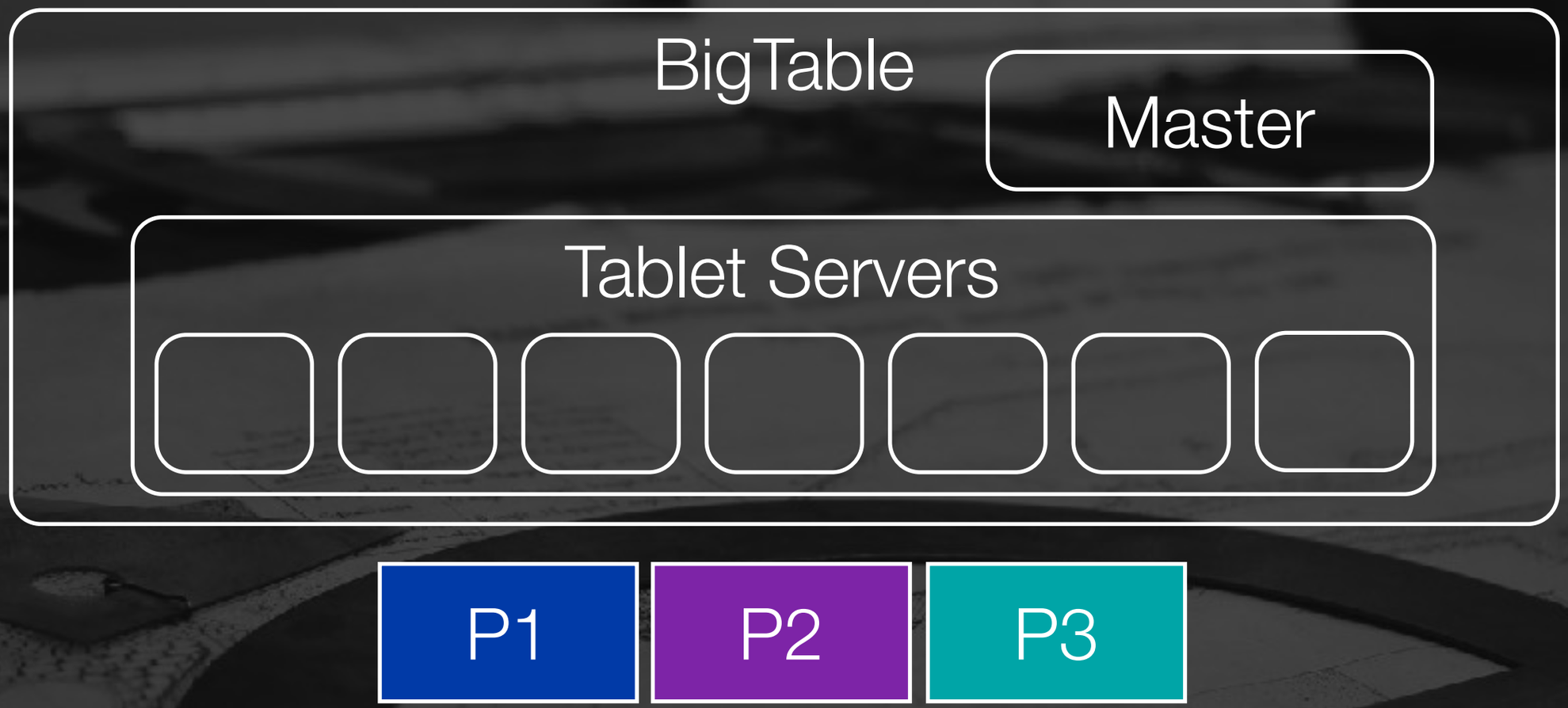
BigTable Architecture



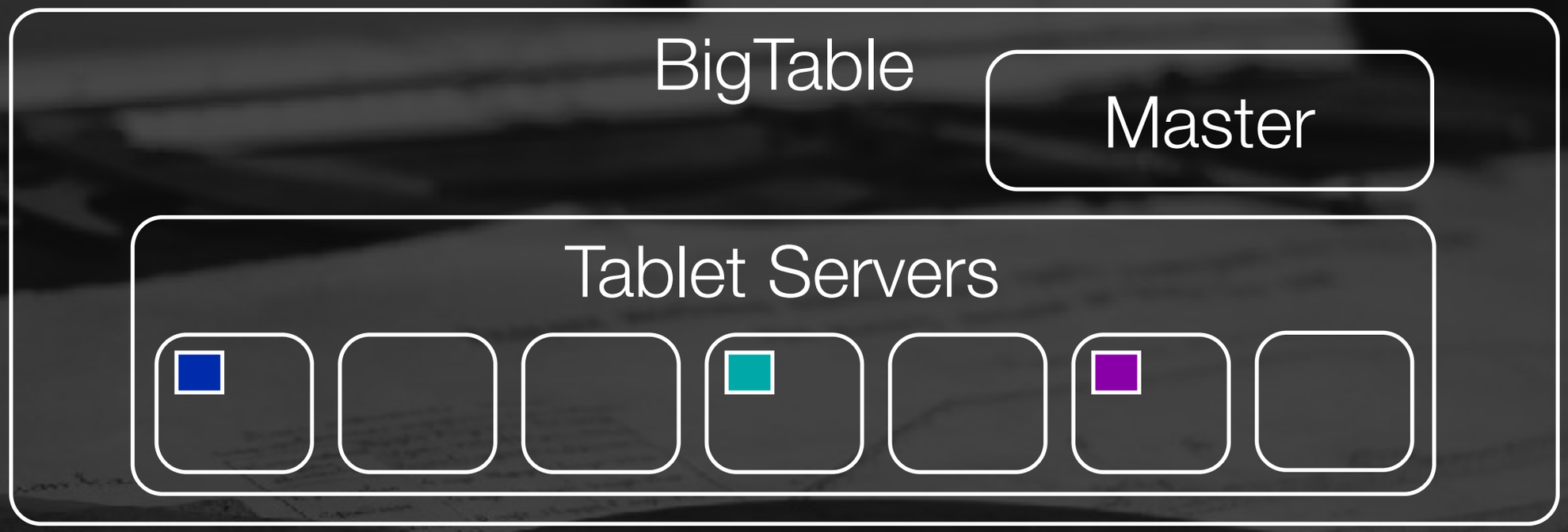
Architecture: Tables



Architecture: Tables



Architecture: Tables

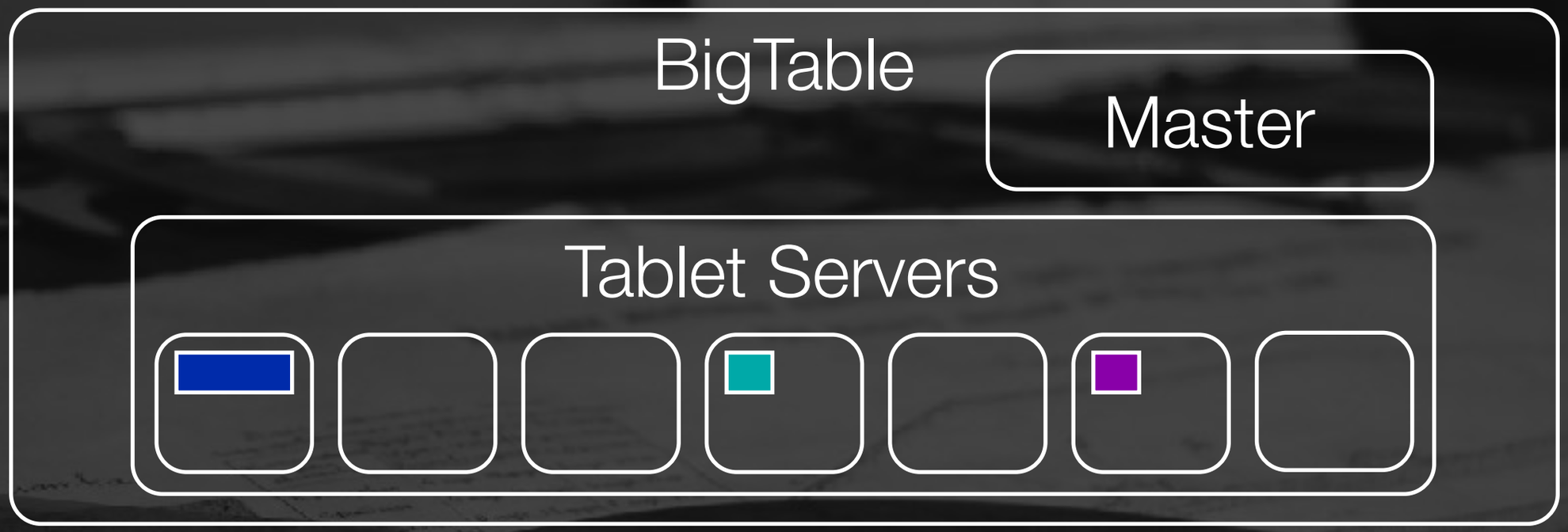


Architecture: Splits

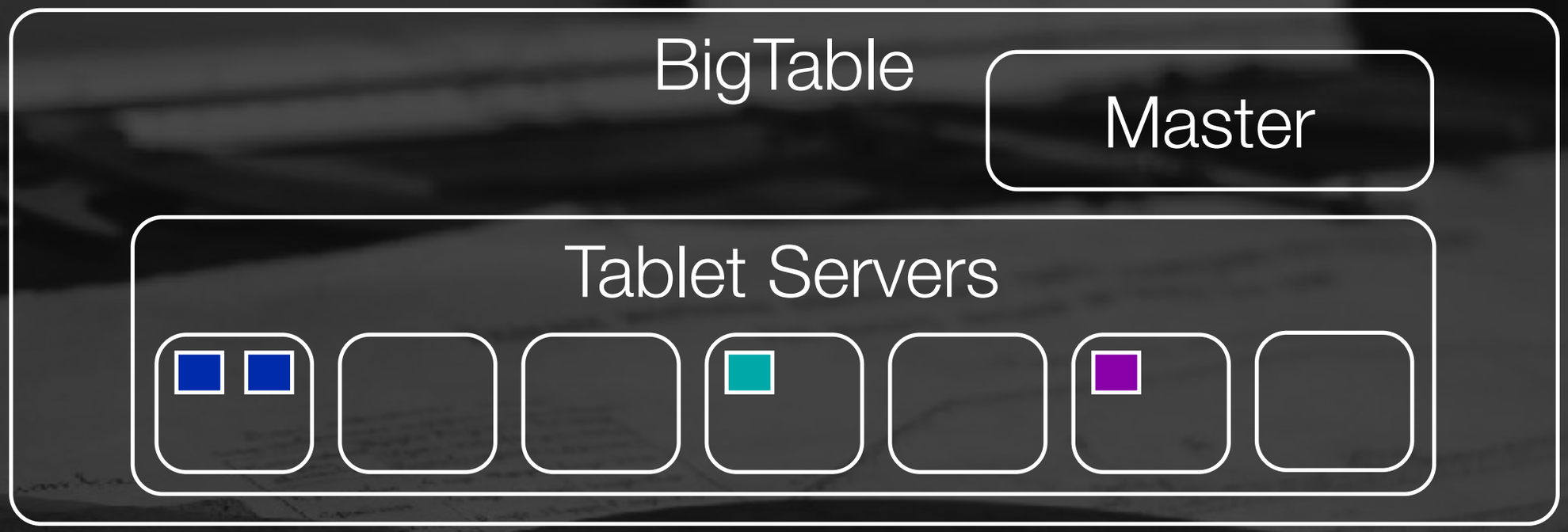
row	col fam	col qual	time	value
bill	attribute	age	Jun 2010	49
bill	attribute	phone	Jun 2010	555-1212
bill	purchases	sneakers	Apr 2010	$100

george	attribute	age	Oct 2009	38
george	purchases	sneakers	Nov 2009	$80
george	returns	hat	Dec 2009	$30

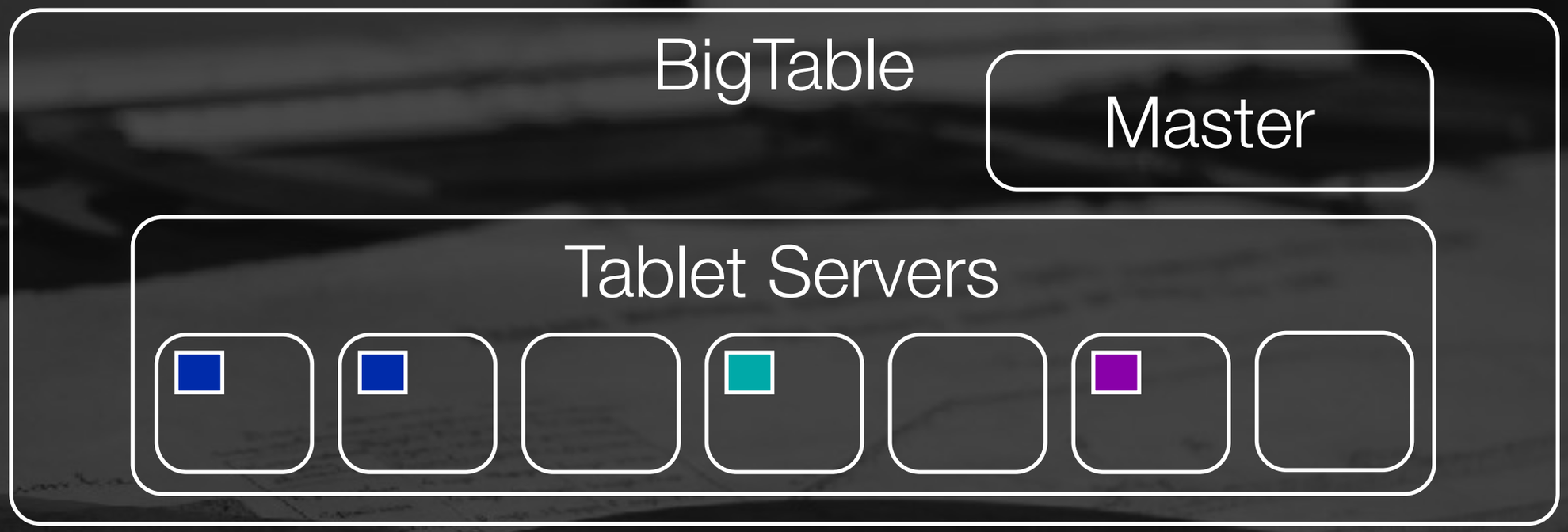
Architecture: Splits



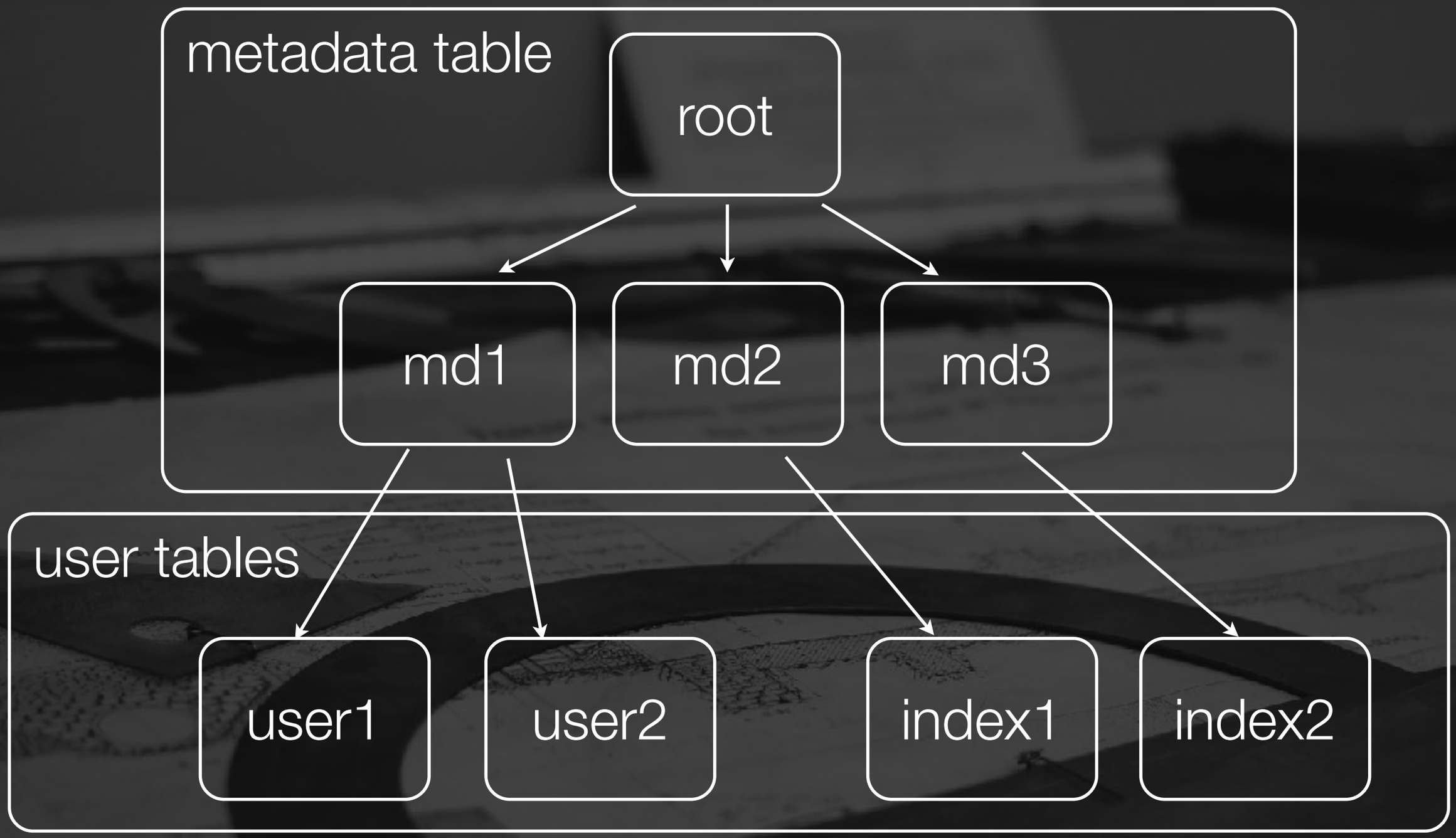
Architecture: Splits



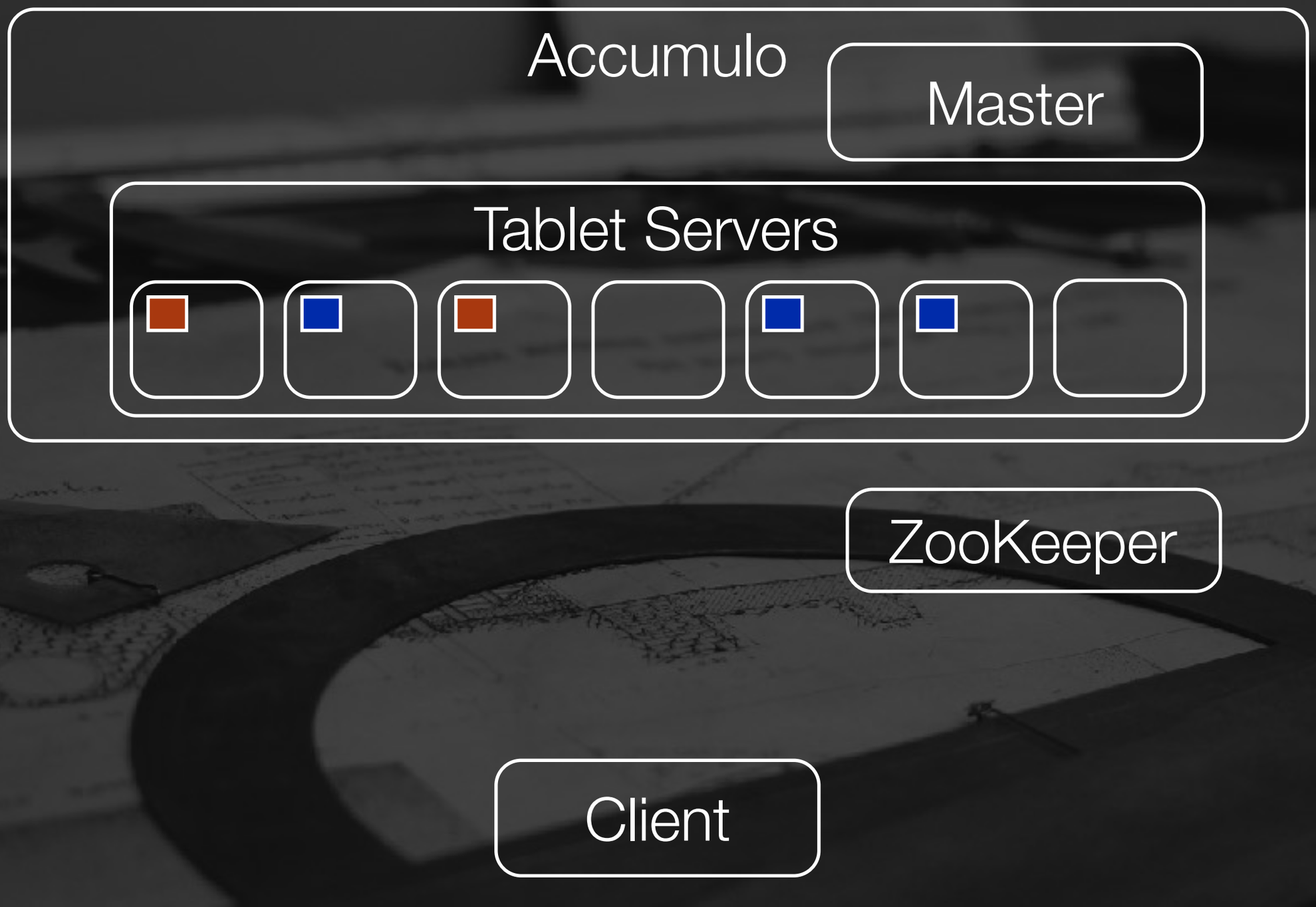
Architecture: Splits



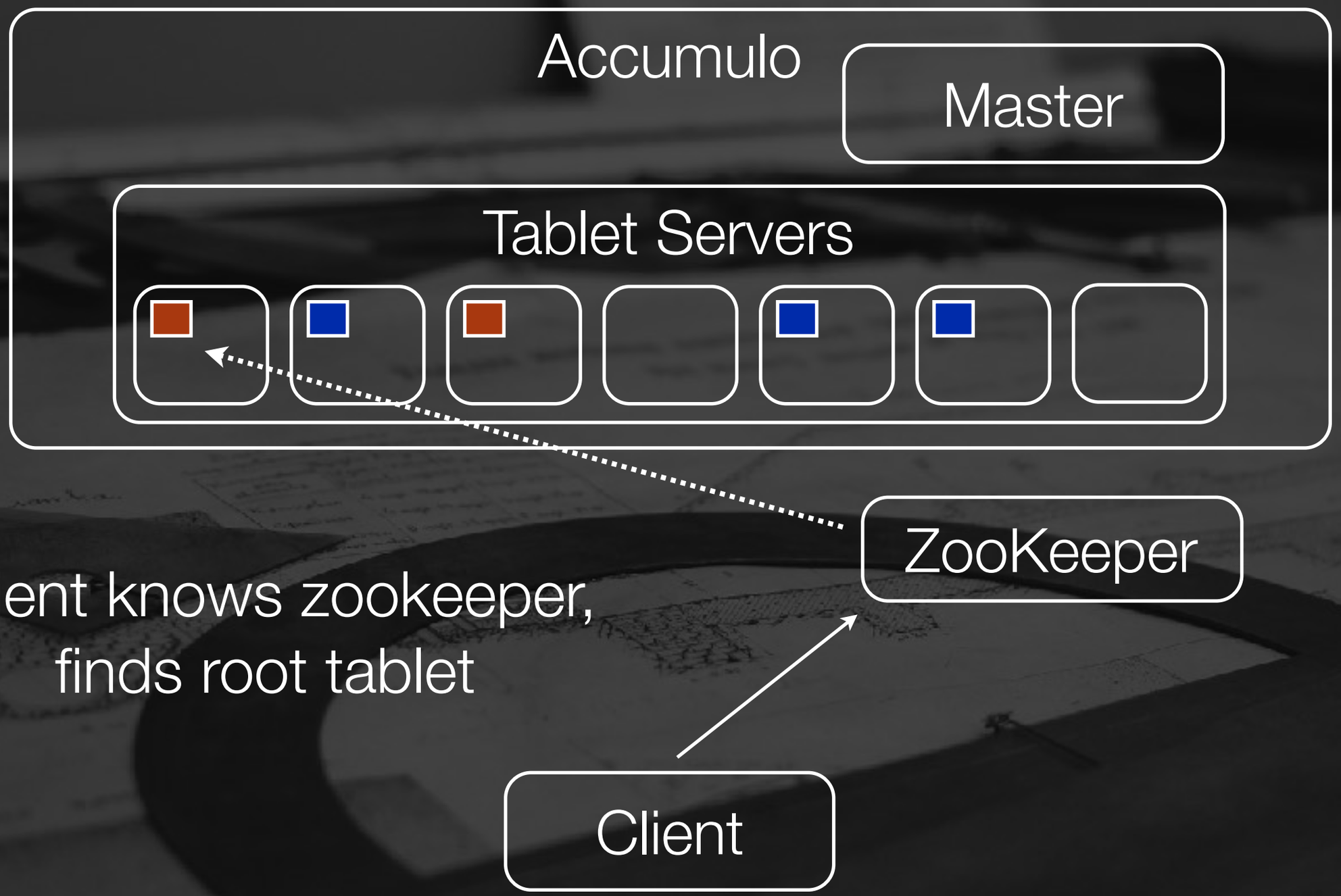
Metadata Hierarchy



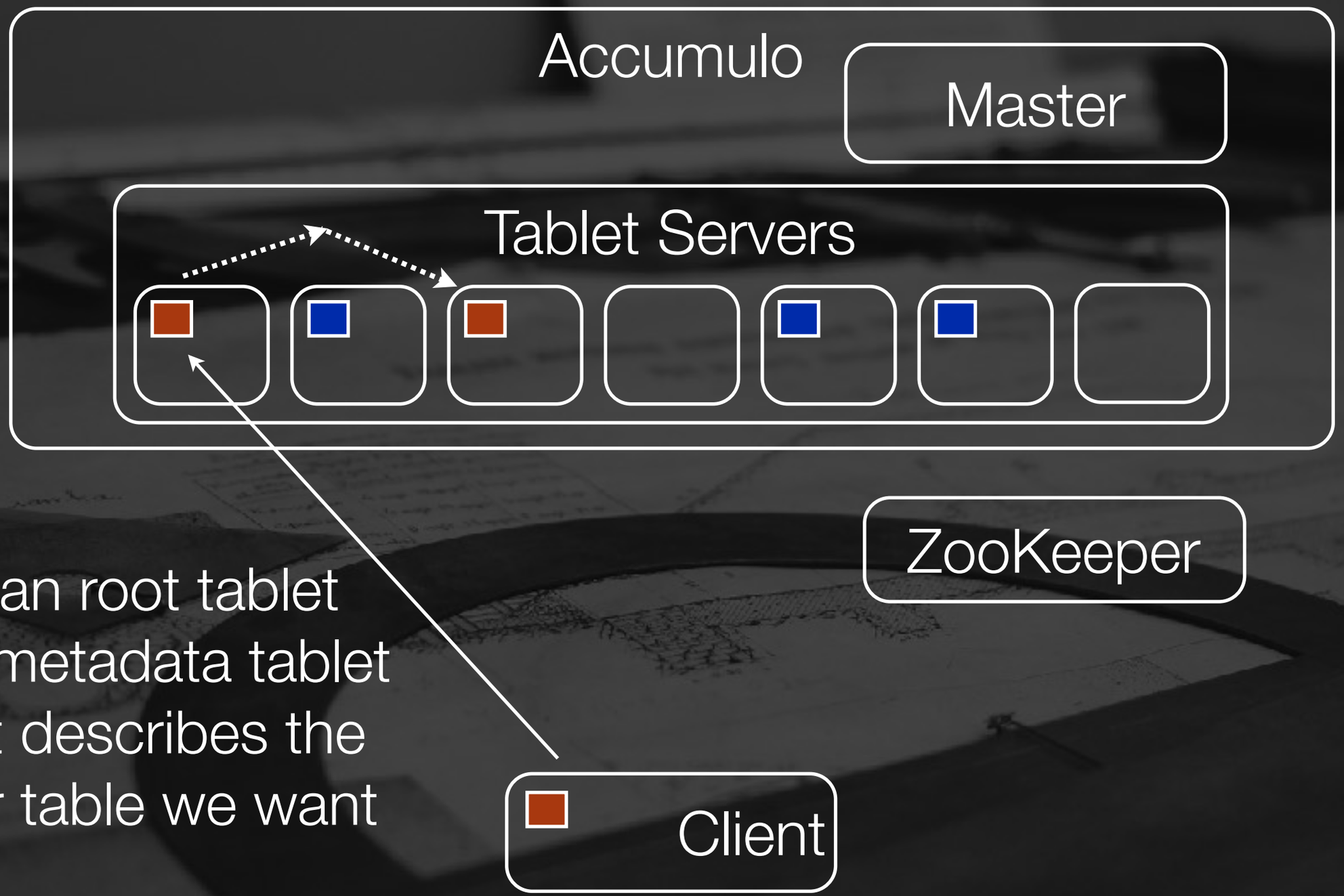
Architecture: Lookup



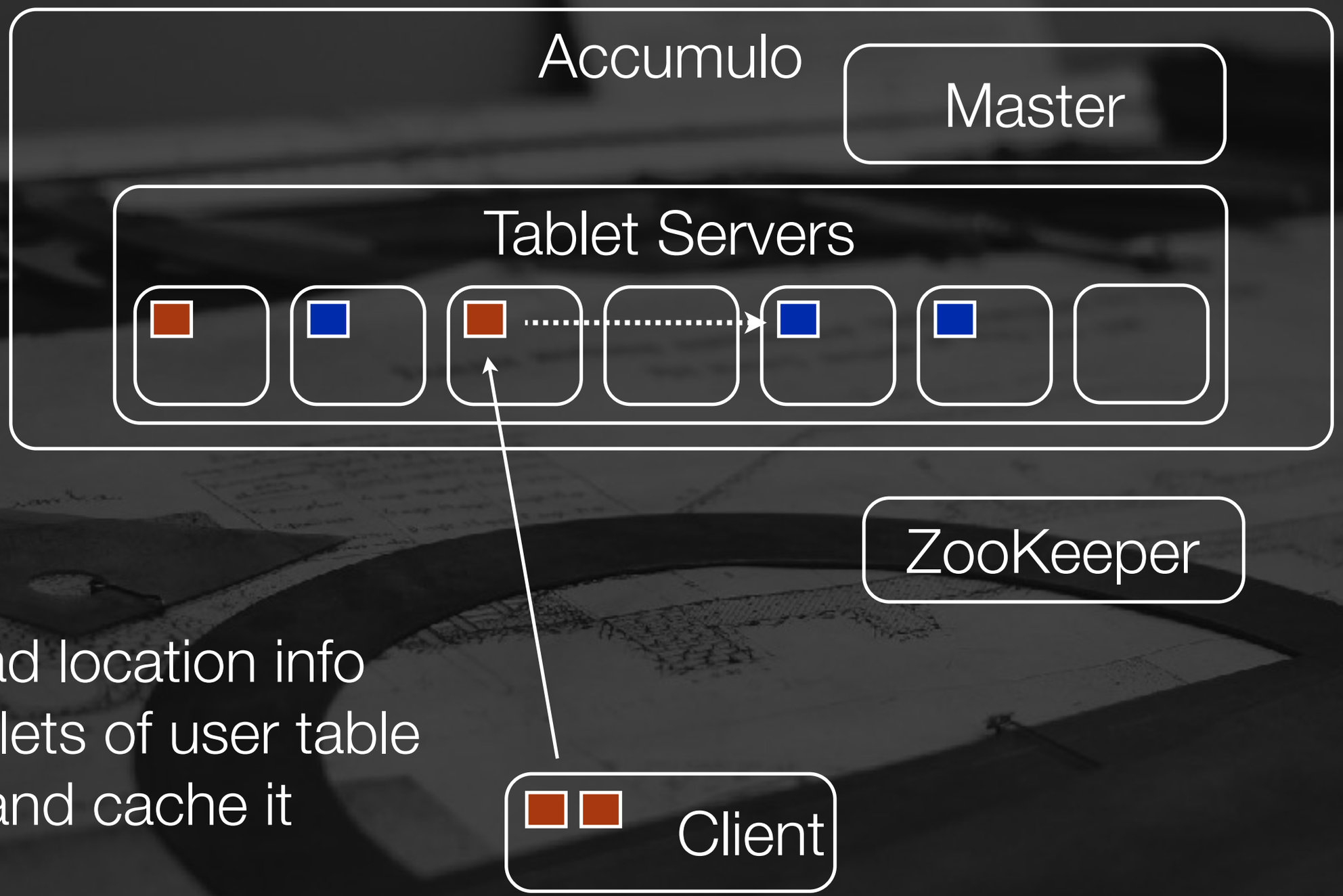
Architecture: Lookup



Architecture: Lookup

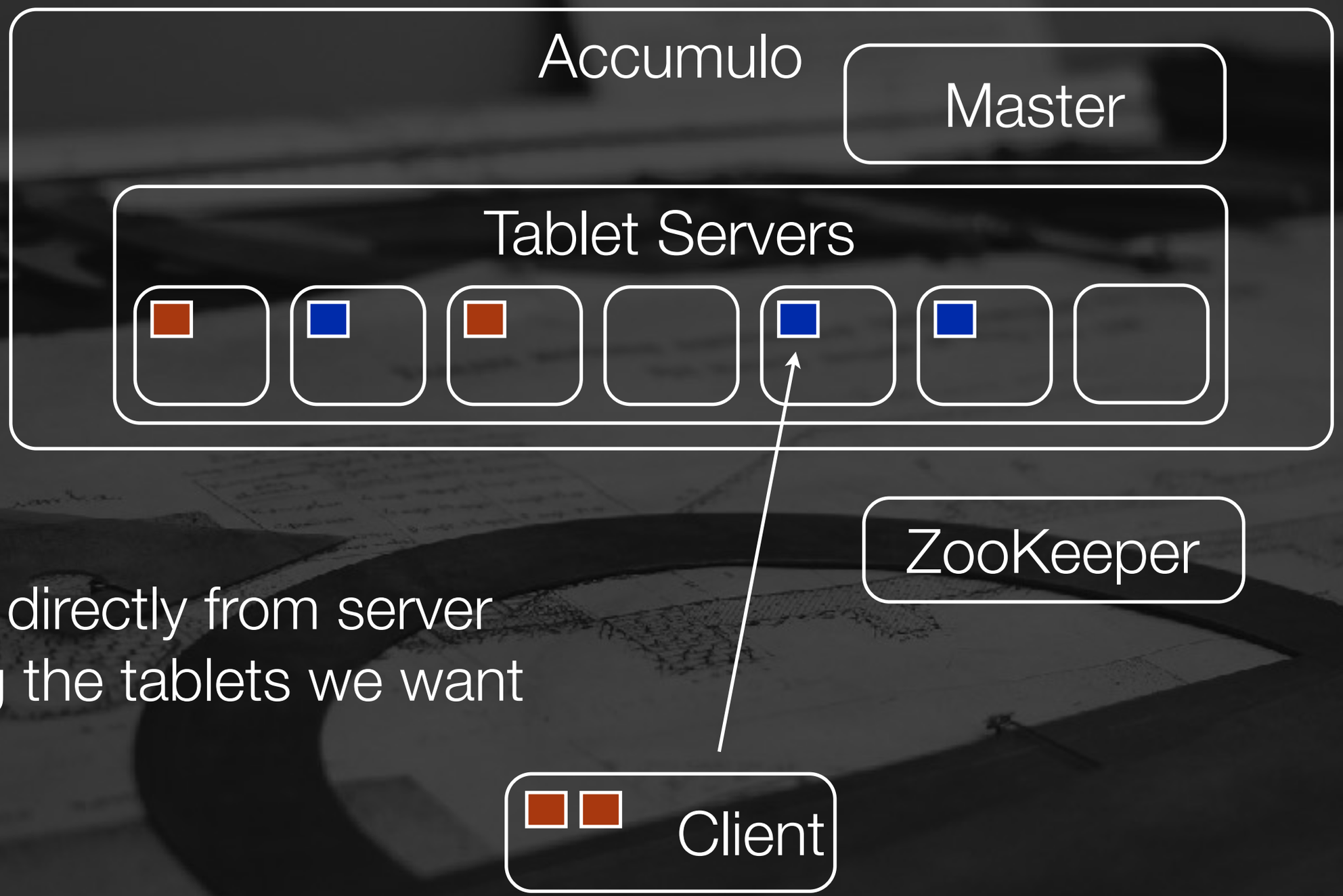


Architecture: Lookup



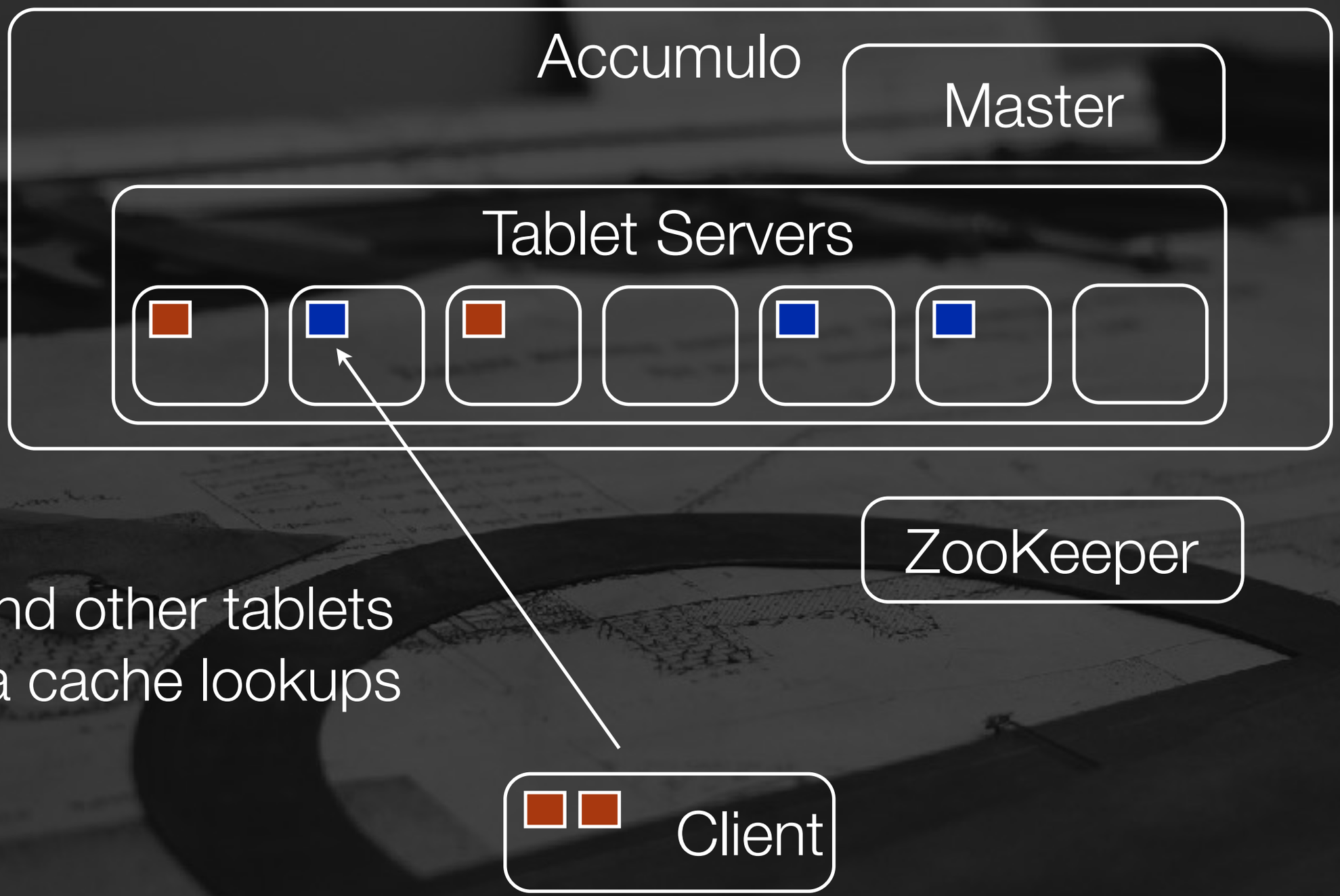
Read location info
of tablets of user table
and cache it

Architecture: Lookup

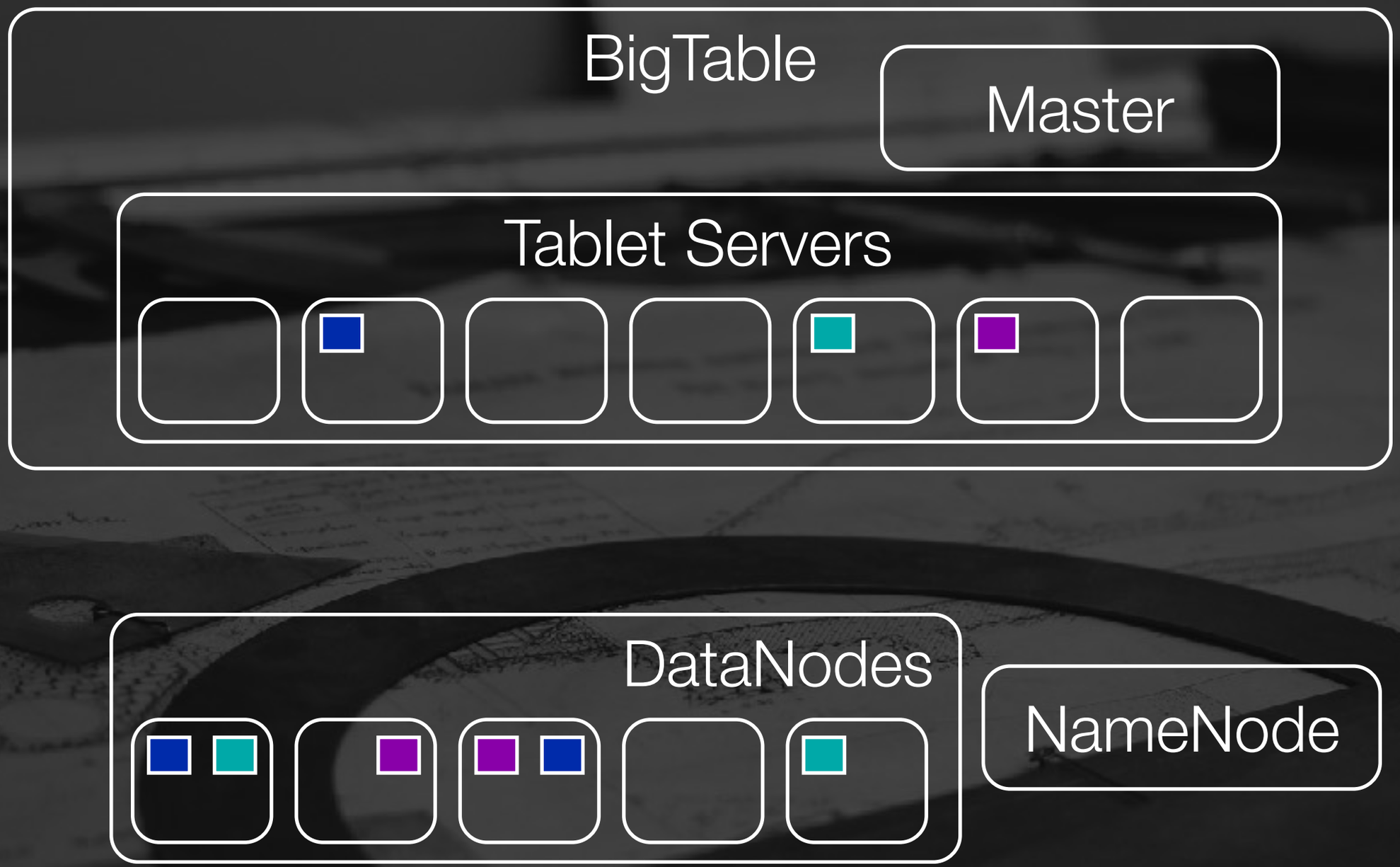


Read directly from server
holding the tablets we want

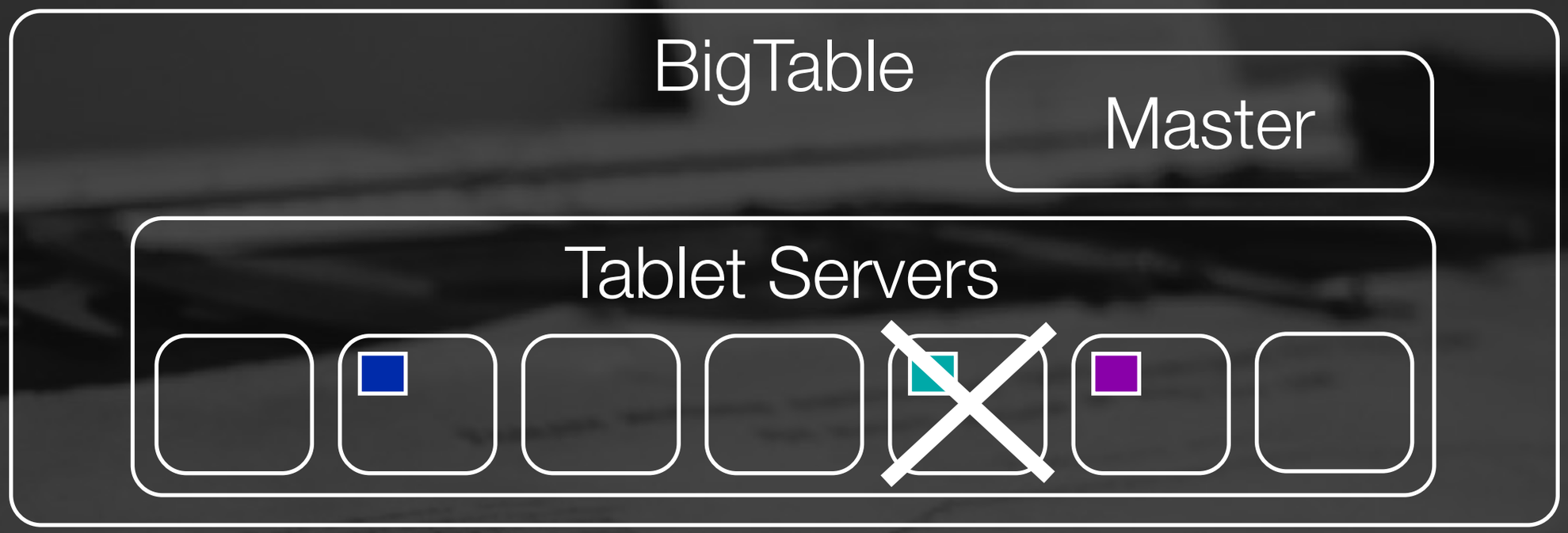
Architecture: Lookup



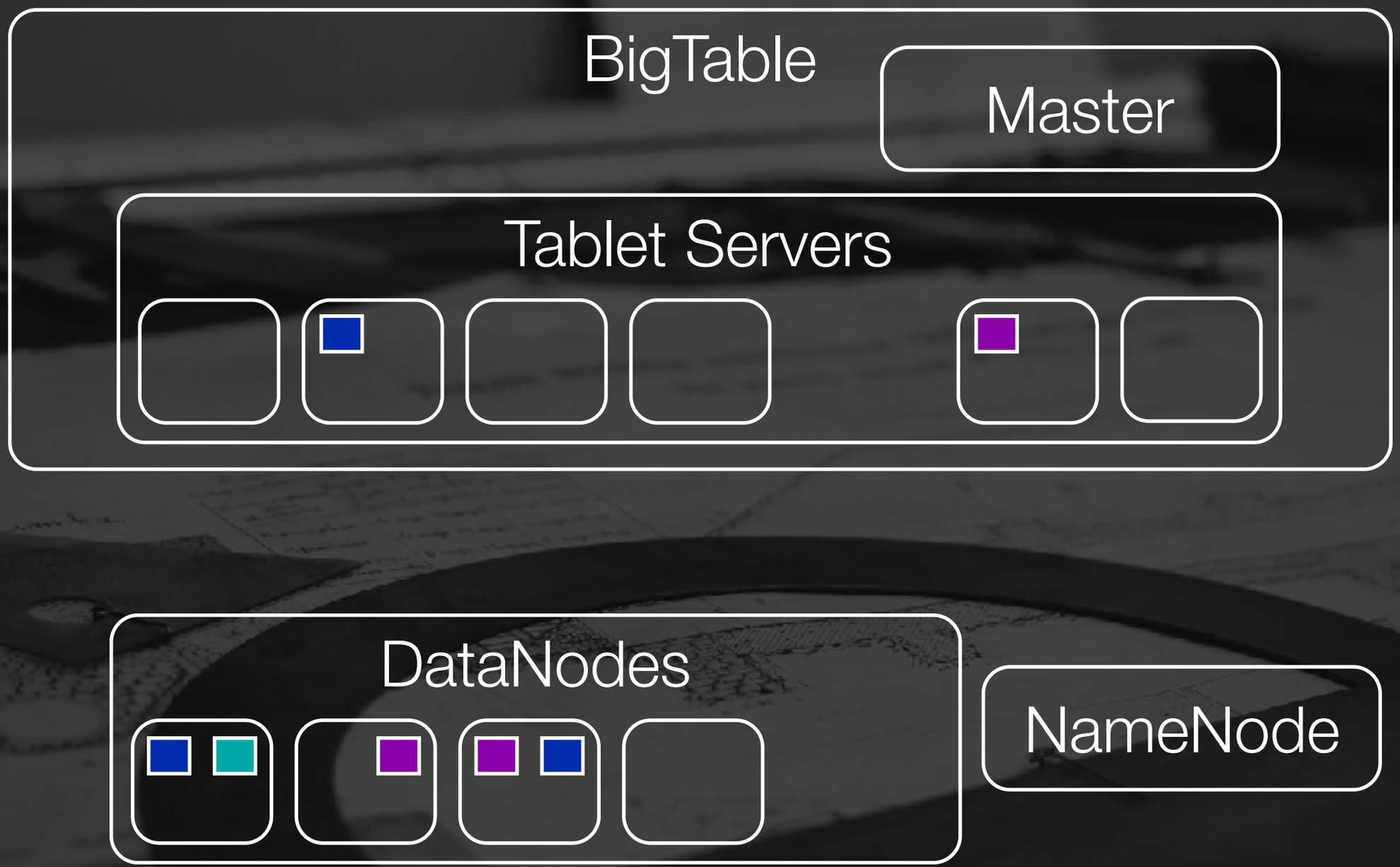
Architecture: Recovery



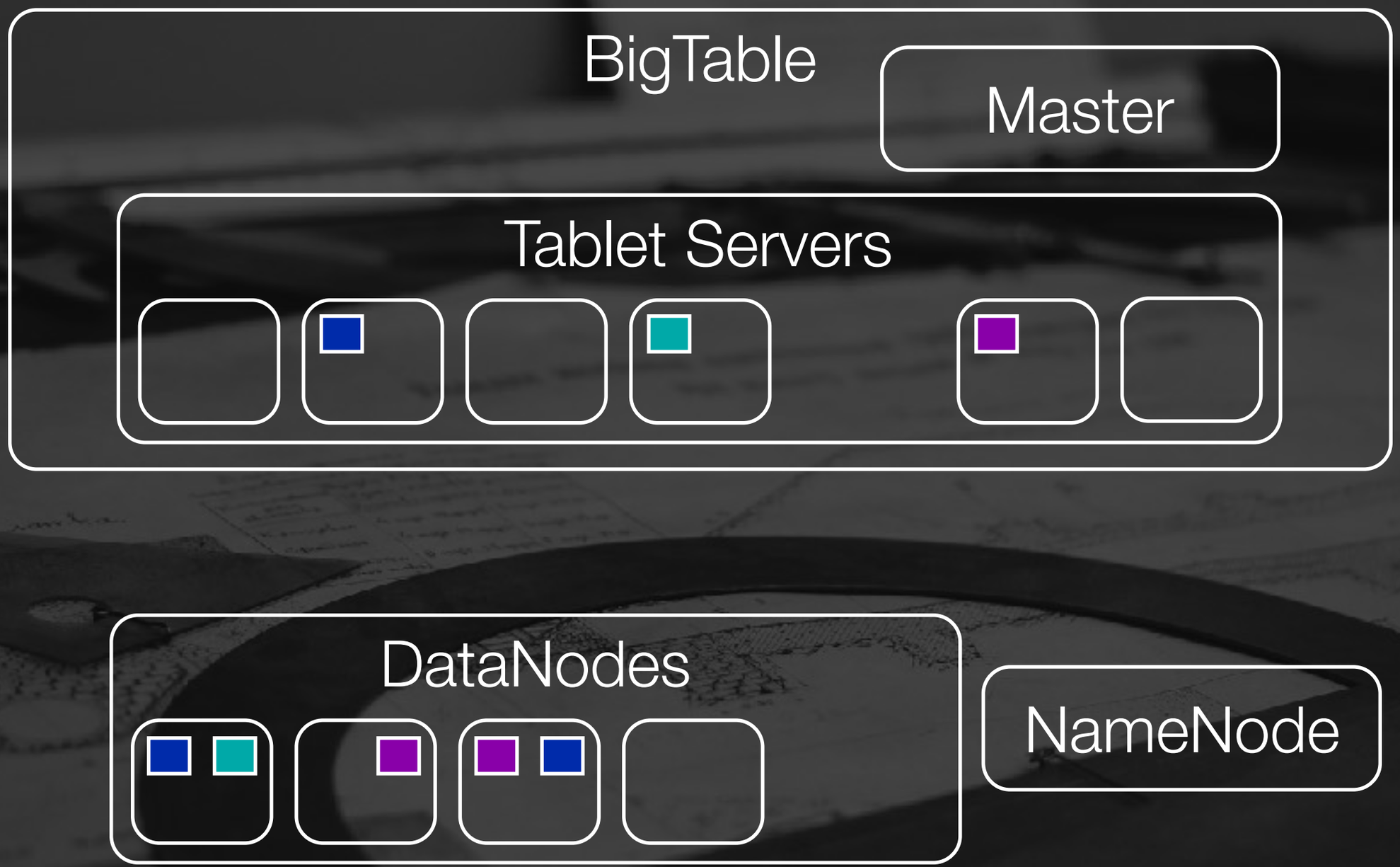
Architecture: Recovery



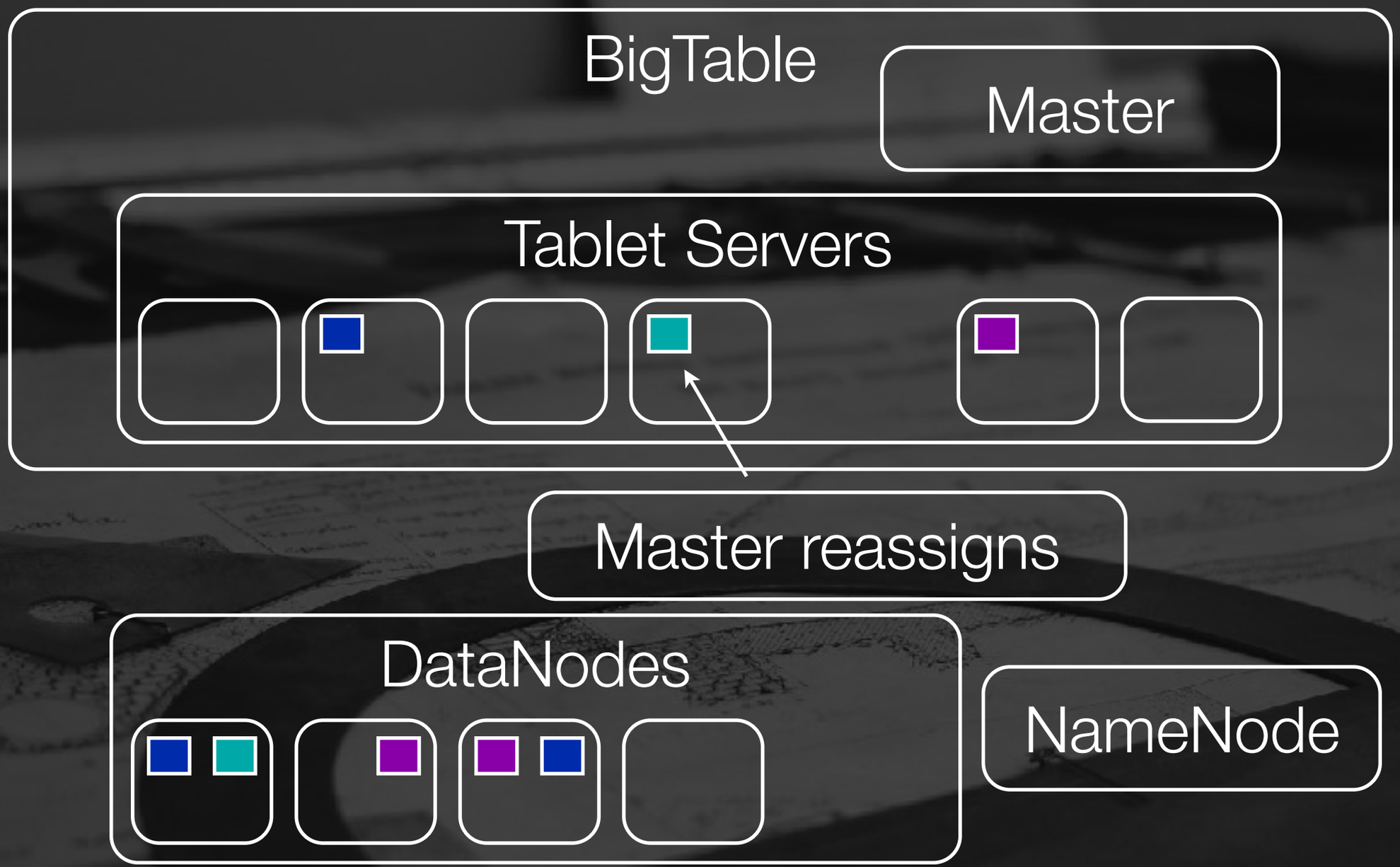
Architecture: Recovery



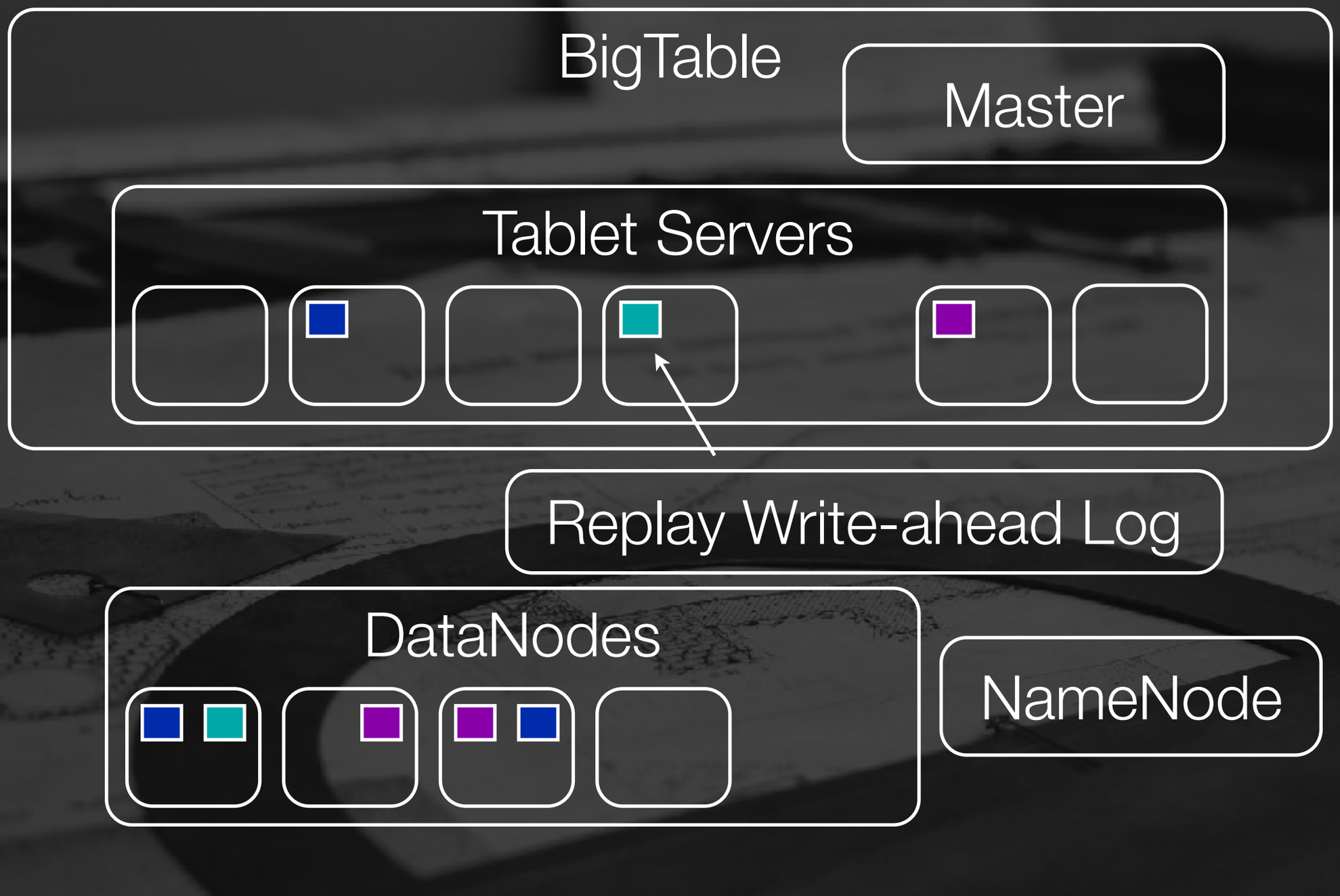
Architecture: Recovery



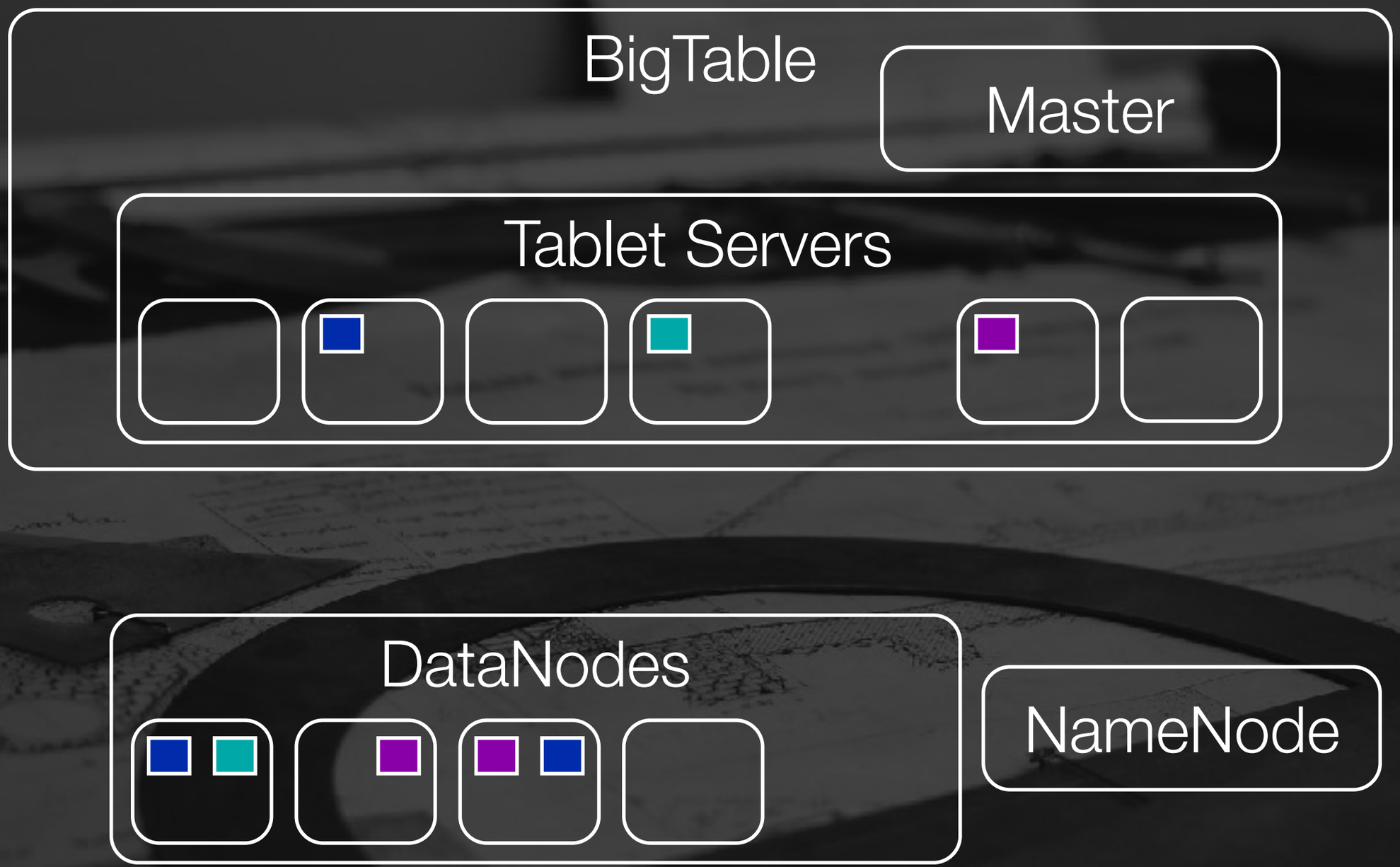
Architecture: Recovery



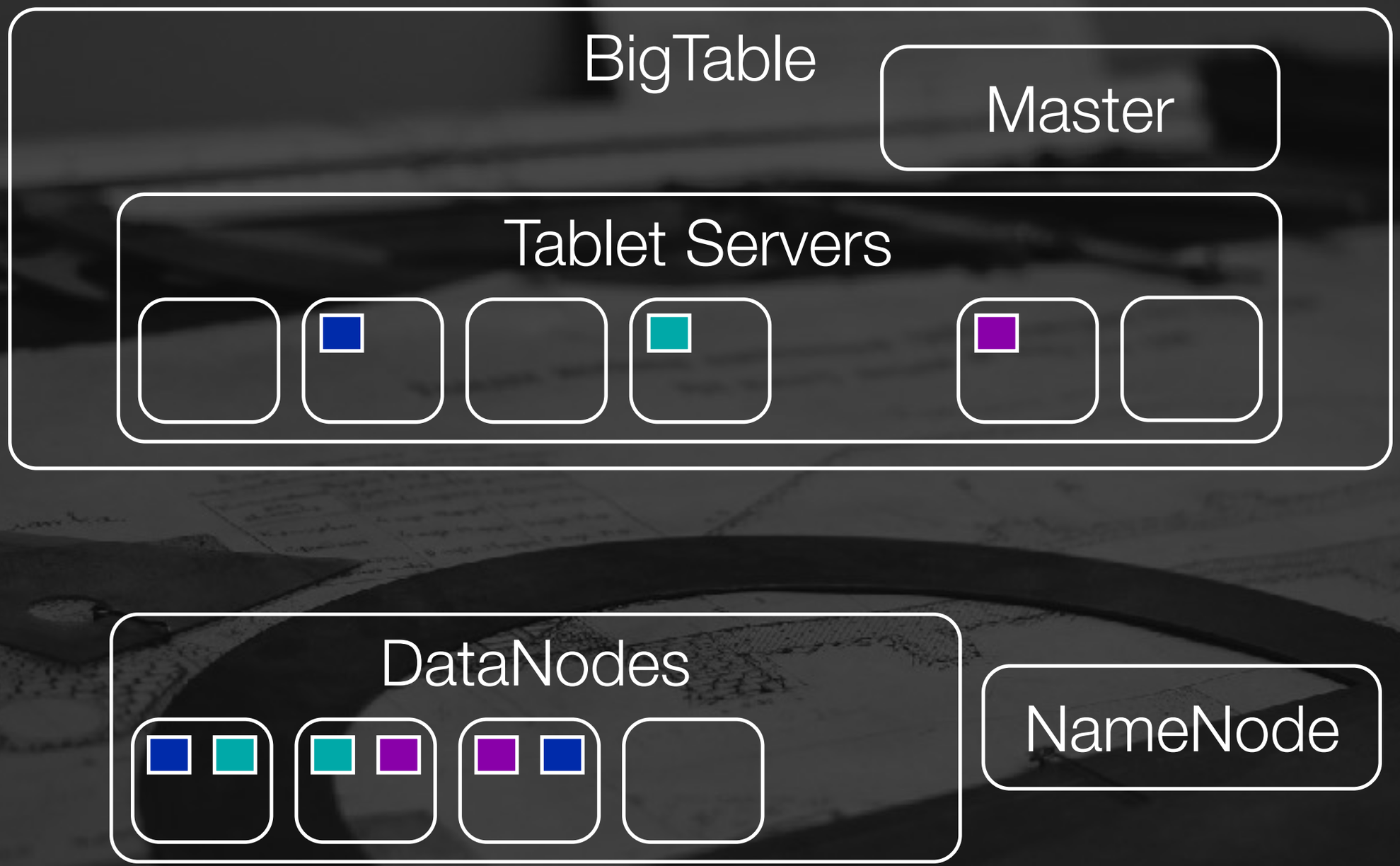
Architecture: Recovery

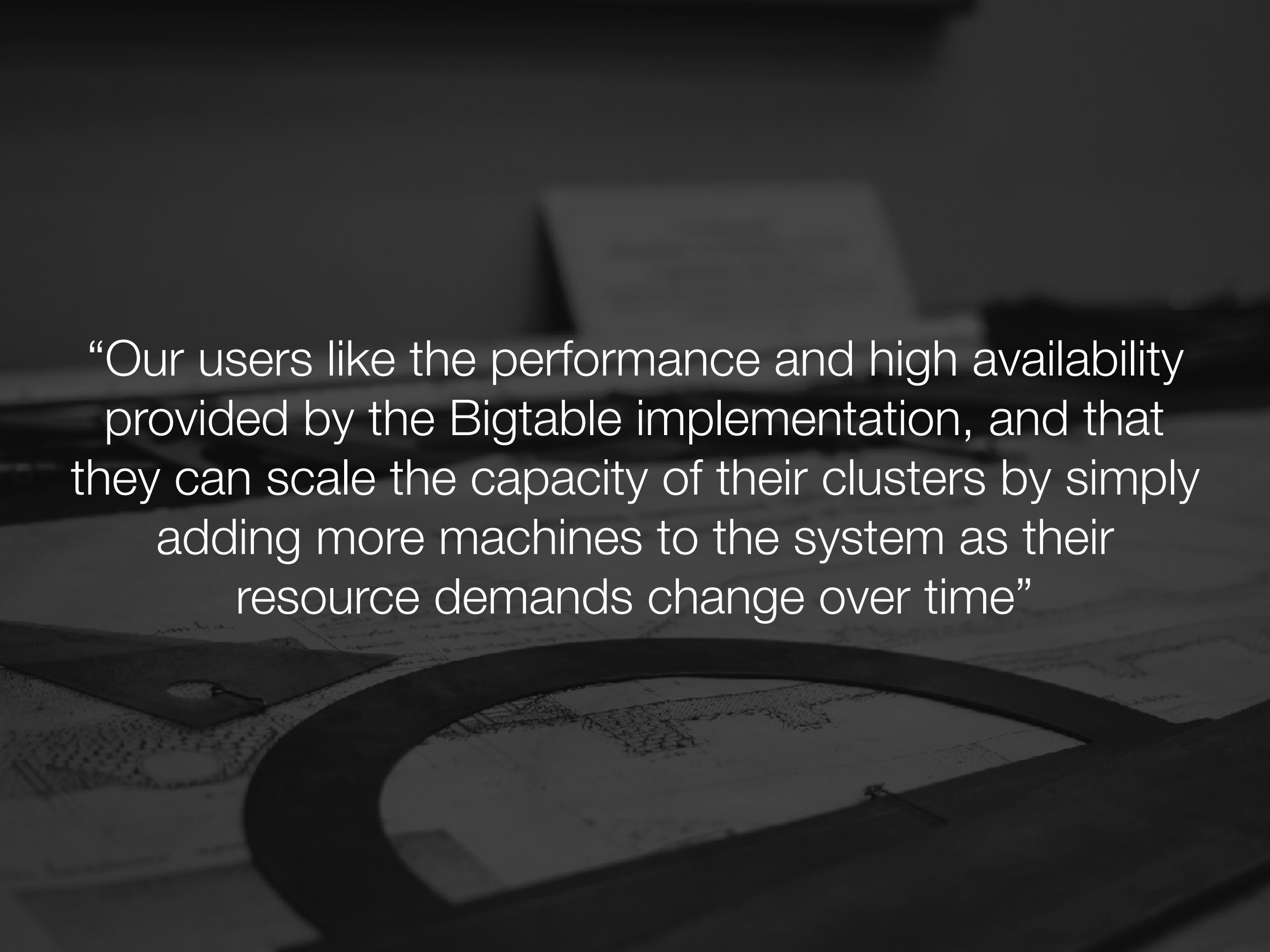


Architecture: Recovery



Architecture: Recovery





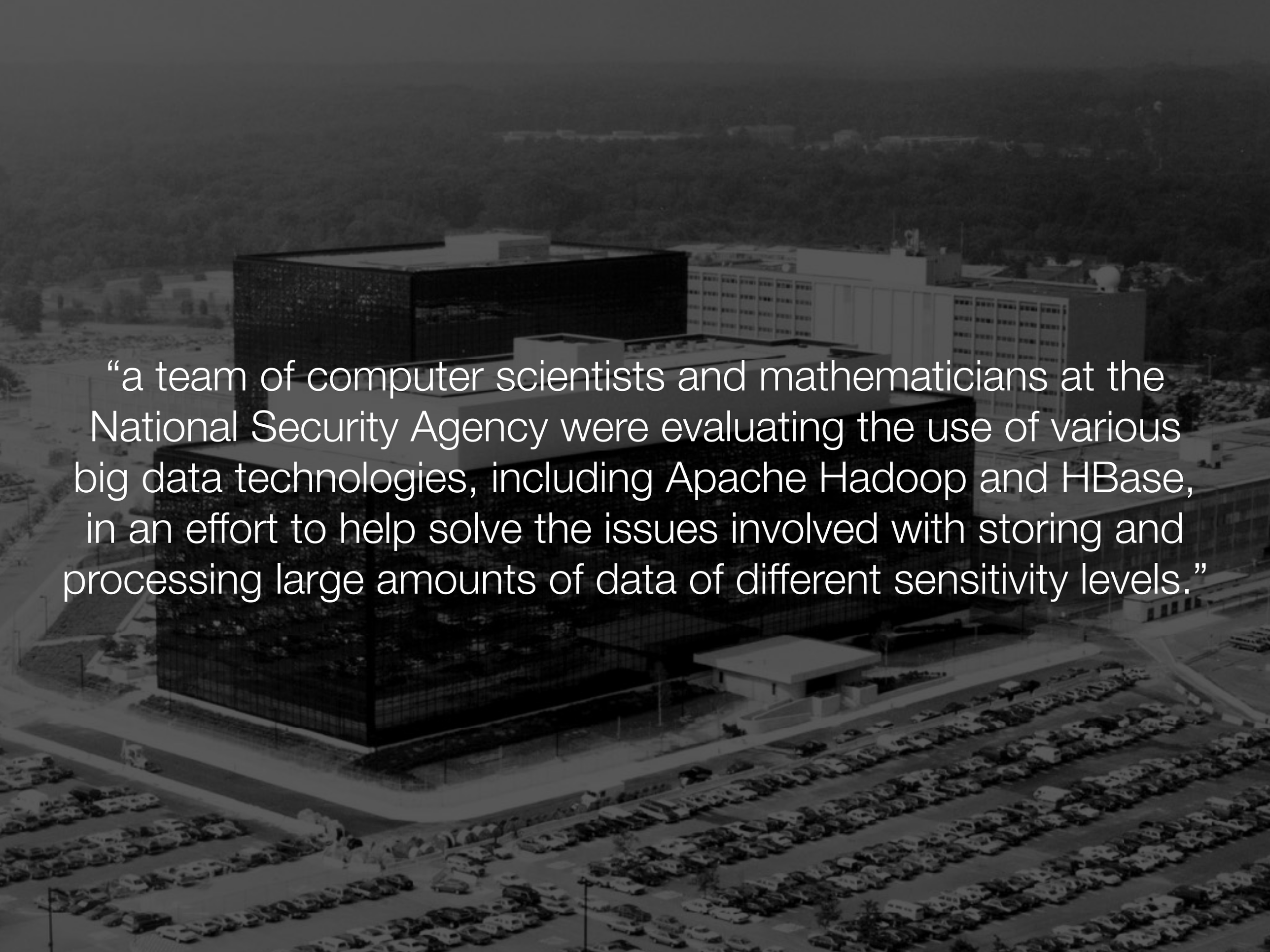
“Our users like the performance and high availability provided by the Bigtable implementation, and that they can scale the capacity of their clusters by simply adding more machines to the system as their resource demands change over time”

“New users are sometimes uncertain of how to best use the Bigtable interface, particularly if they are accustomed to using relational databases that support general-purpose transactions”

“We have gotten a substantial amount of flexibility from designing our own data model for Bigtable.”

“our control over Bigtable’s implementation, and the other Google infrastructure upon which Bigtable depends, means that we can remove bottlenecks and inefficiencies as they arise”

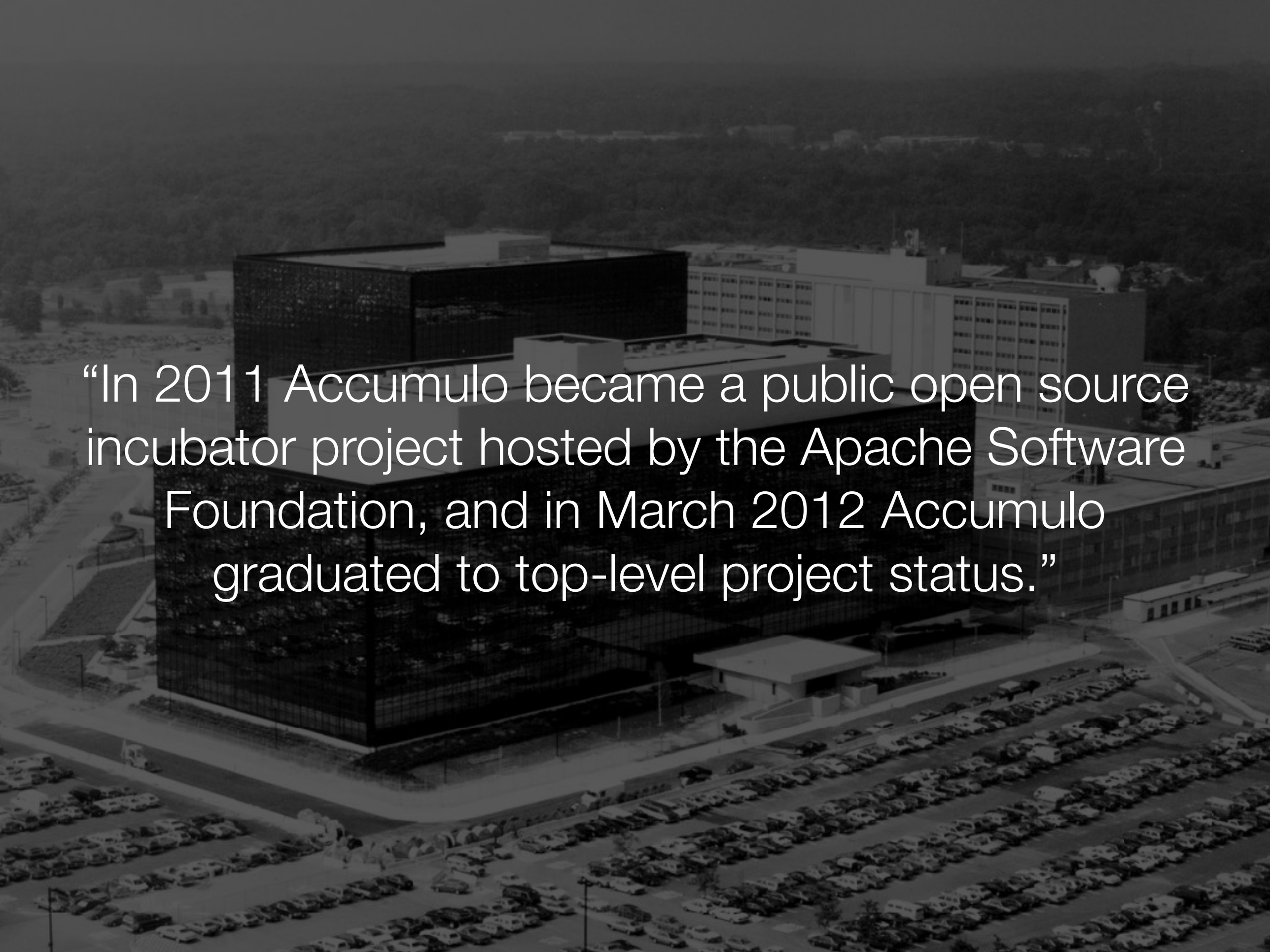
2008 National Security Agency



“a team of computer scientists and mathematicians at the National Security Agency were evaluating the use of various big data technologies, including Apache Hadoop and HBase, in an effort to help solve the issues involved with storing and processing large amounts of data of different sensitivity levels.”

"After reviewing existing solutions and comparing the stated objectives of existing open source projects to the agency's goals, the team began a new implementation of Google's BigTable."

“the team extended the BigTable design with additional features that included a method for labeling each key-value pair with its own access information, called *Column Visibilities*, and a mechanism for performing additional server-side functionality, called *Iterators*.”



"In 2011 Accumulo became a public open source incubator project hosted by the Apache Software Foundation, and in March 2012 Accumulo graduated to top-level project status."

Accumulo Data Model

Key					
row ID	Column			Timestamp	Value
	Family	Qualifier	**Visibility**		

Accumulo introduces an additional column component

Accumulo Data Model

column family — column family — column family

column qualifiers

	attribute: age	attribute: phone	purchases: sneakers	returns:hat
bill	49	555-1212	$100	-
george	38	-	$80	$30
george	37	-	$80	$30

rows

time

private

public

Accumulo Data Model

row	col fam	col qual	col vis	time	value
bill	attribute	age	public	Jun 2010	49
bill	attribute	phone	private	Jun 2010	555-1212
bill	purchases	sneakers	public	Apr 2010	$100
george	attribute	age	private	Oct 2009	38
george	purchases	sneakers	public	Nov 2009	$80
george	returns	hat	public	Dec 2009	$30

Accumulo API

Column families can be created dynamically

Introduced batch scanners, maintain support for large rows, both of which enable building tables that serve as secondary indexes

‘Archaeologist’s Approach’ to Data

Allow the data to inform you about its schema

Avoid making assumptions, changing data as long as possible

Store, protect, index data to allow exploration and discovery

Use bulk processing like Spark to create clean, summarized derivatives of data. Preserve original in case assumptions prove to be false and reprocessing is required.

Accumulo Architecture

Accumulo Proof Points

AWS benchmark

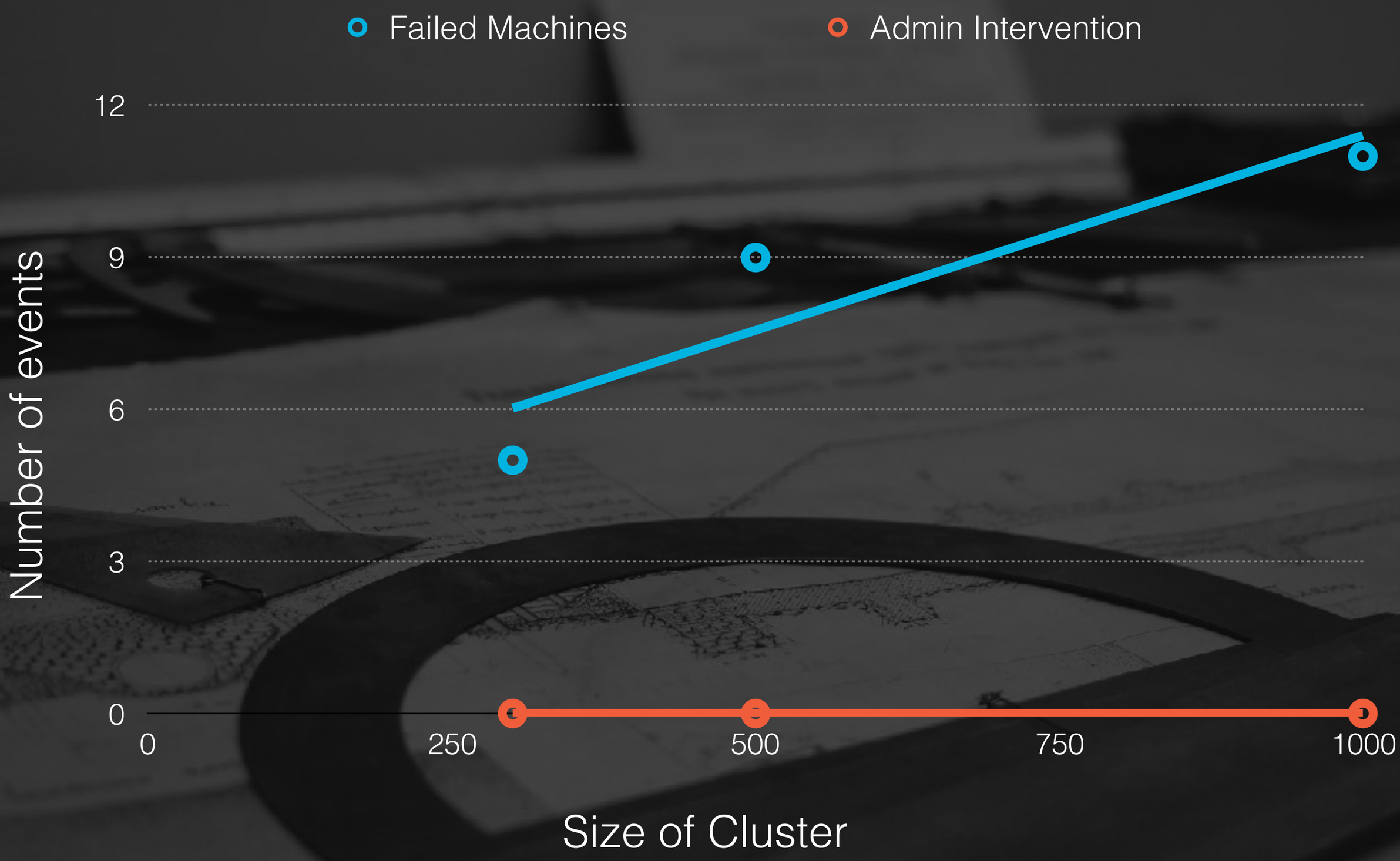
- Tested at 300, 500, and 1000 machines
- 100 million entries written per second
- 408 terabytes
- 7.56 trillion total entries
- Several hardware failures, zero interruptions

https://accumulo.apache.org/papers/accumulo-benchmarking-2.1.pdf

Ingest Benchmark

Scan Latency

Administrative Overhead



Accumulo Proof Points

Graph processing benchmark

- 1200 machines
- 4.4 trillion vertices
- 70.4 trillion edges
- 1 petabyte processed
- 149 million edges traversed per second

http://www.pdl.cmu.edu/SDI/2013/slides/big_graph_nsa_rd_2013_56002v1.pdf

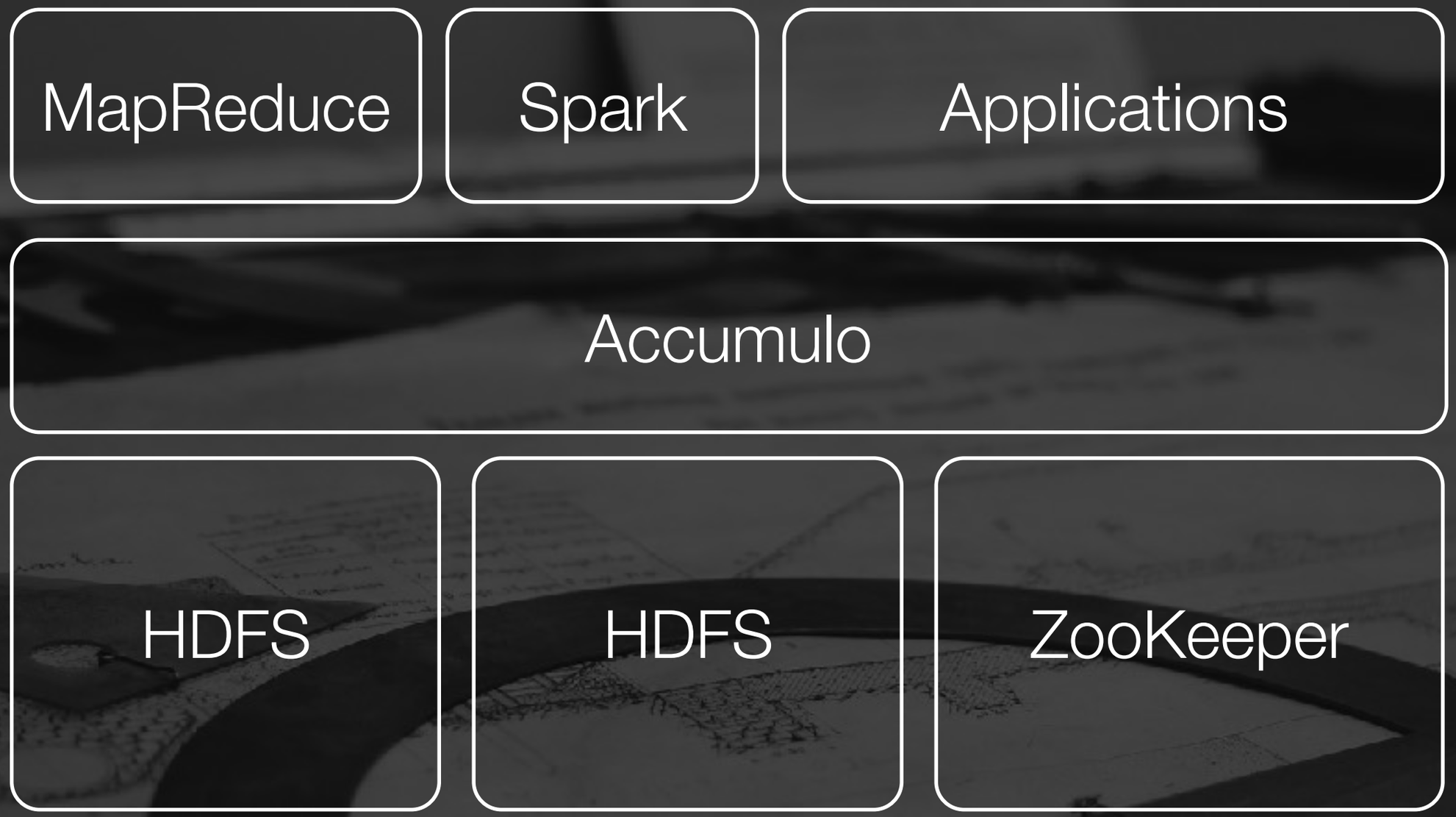
Accumulo Proof Points

D4M benchmark

- D4M is a data model integrating Accumulo with pMatlab
- 216 machines
- 115 million inserts per second
- Used checkpointing instead of write-ahead log

http://www.ieee-hpec.org/2014/CD/index_htm_files/FinalPapers/31.pdf

Accumulo Architecture



Accumulo in the Enterprise

Accumulo in the Enterprise

Besides the Intelligence community, Accumulo receives special interest from highly regulated industries such as finance and healthcare, often because of its strong security features and scalability.

Accumulo is supported by all Hadoop vendors and several companies have built commercial products on Accumulo.

Enterprises have so much data in so many different systems that the ‘archaeologist’s approach’ is warranted

Bringing data together physically in Accumulo and protecting it logically is a major enabler to data science initiatives

Accumulo in the Enterprise

Three strengths make it attractive as a place for gathering data:

1. Flexible schema handling, columns created dynamically, making it possible to load data without fully characterizing it first, and to handle inconsistent or changing data

2. Highly scalable

3. Fine-grained access control, avoiding creating a security problem just because there are multiple levels of data sensitivity

Accumulo in the Enterprise

Accumulo has good support for *secondary indexing*, making it possible to query data on values in any field.

Support for analytical frameworks like MapReduce and Spark make it possible to process data *in situ* and serves as a good place to host and serve up analytical results for interactive consumption by users, services, or applications

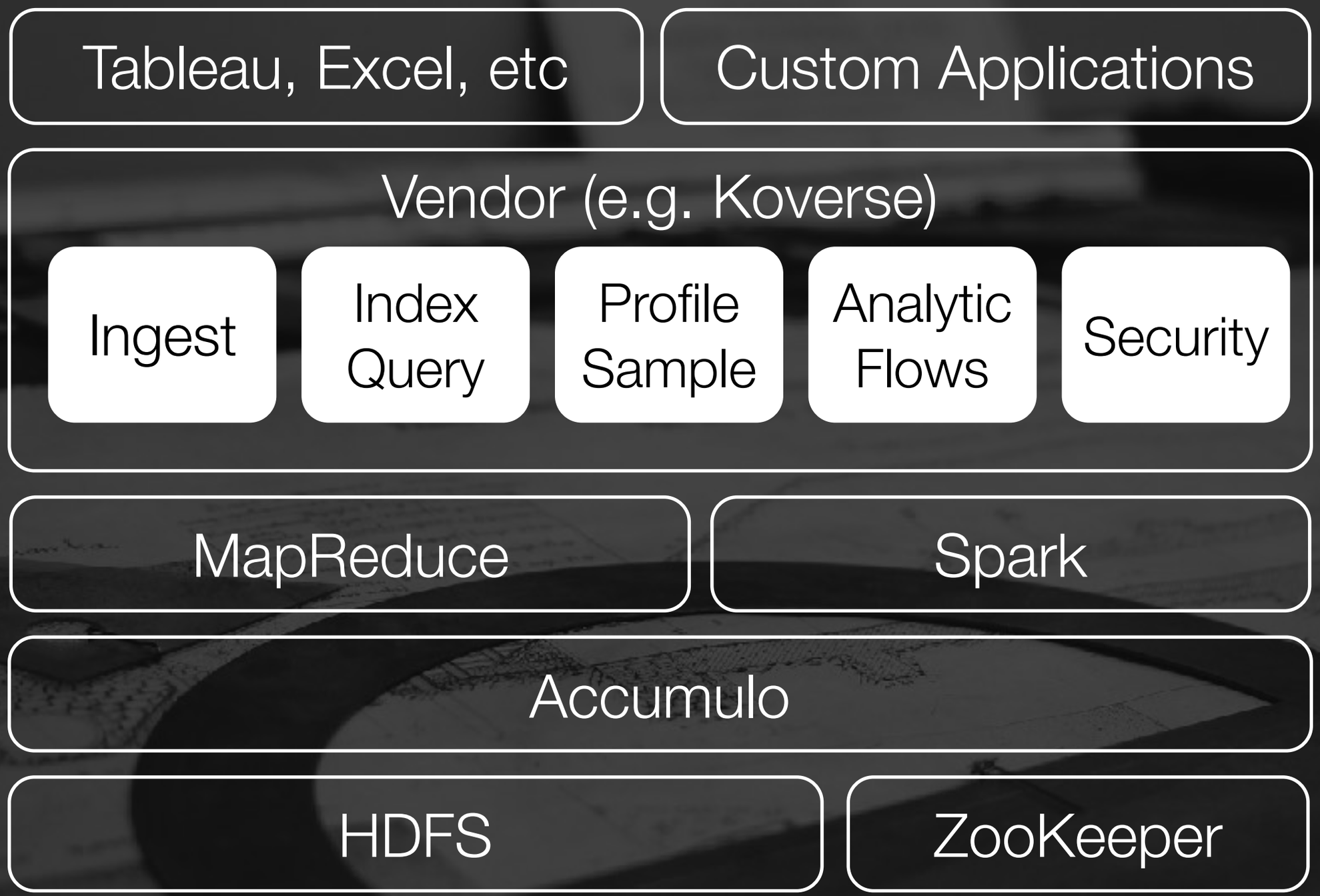
Challenges with Accumulo in the Enterprise

Like BigTable new users are sometimes uncertain of how to best use the Accumulo interface and data model.

While open source components allow organizations some control over the entire storage stack, many organizations lack the expertise to modify these components.

Mapping organizational security policies to Accumulo column visibilities remains an exercise left to the reader

Typical Architecture

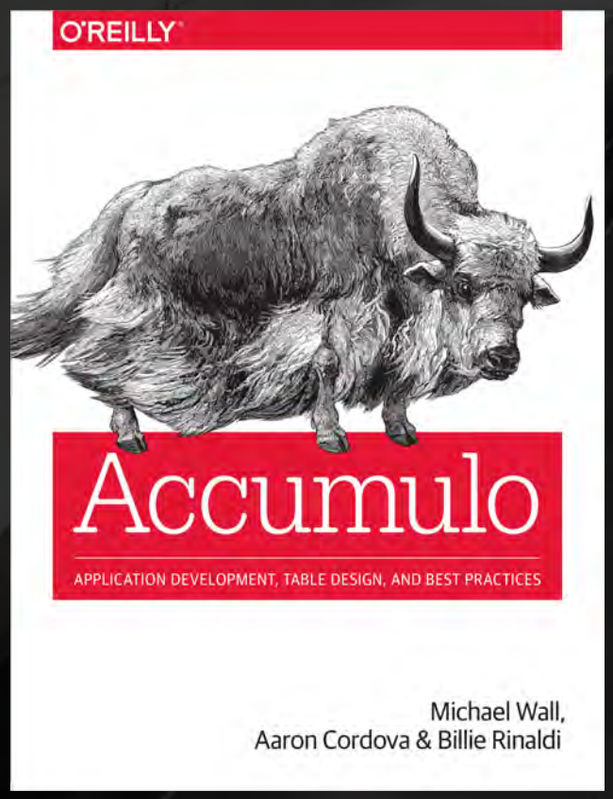


Resources

accumulo.apache.org

@ApacheAccumulo on Twitter

#accumulo on FreeNode IRC



Aaron Cordova

@aaroncordova
www.koverse.com

KOVERSE