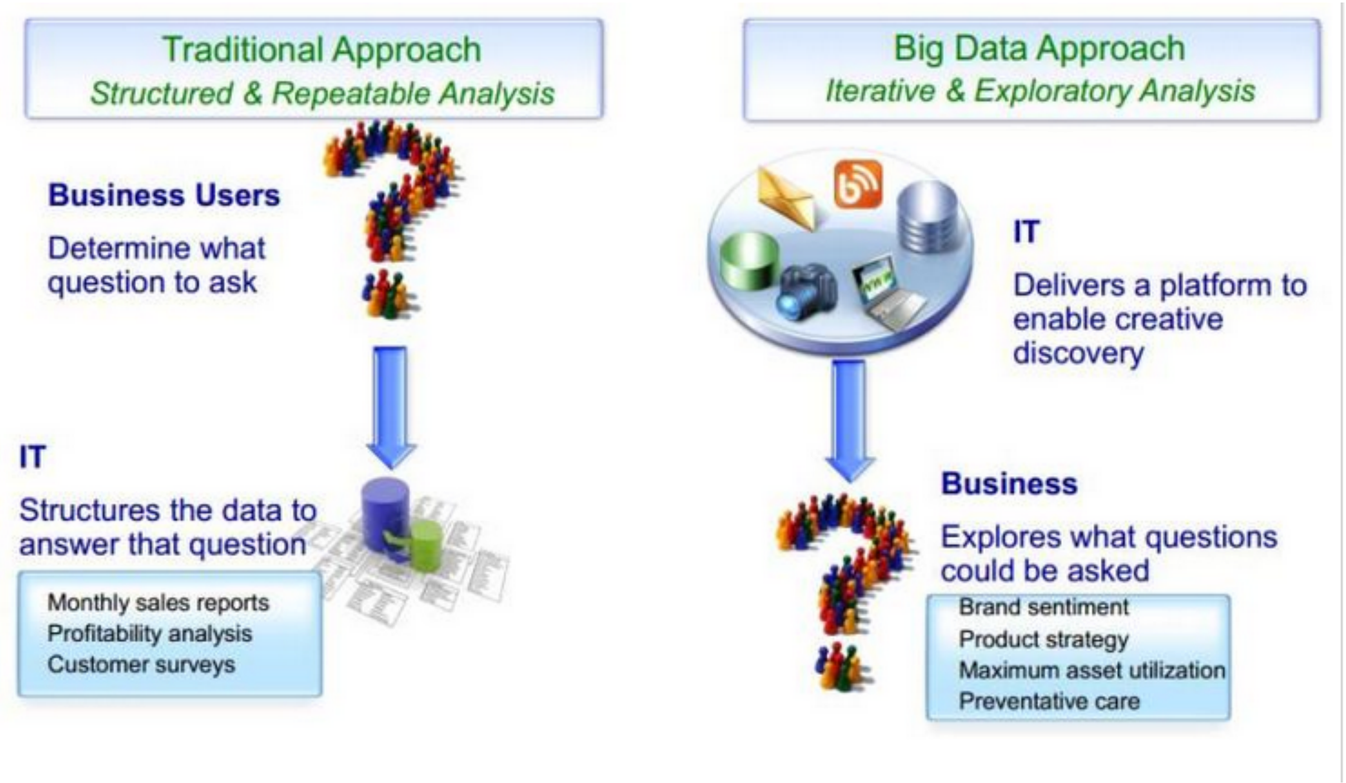
Module 4 Review

Data Lakes: Storage and Maintenance

Goals of Module 4

1. Understand high level architectural approaches.
2. Understand how to think about data models and how they map to high level architecture.
3. Understand how data is stored in HDFS and NOSQL stores.
4. Understand how data is stored in row and columnar oriented databases.
5. Understand what a software defined storage system is.
6. Understand the main concerns of data governance and privacy.

Difference in Approach

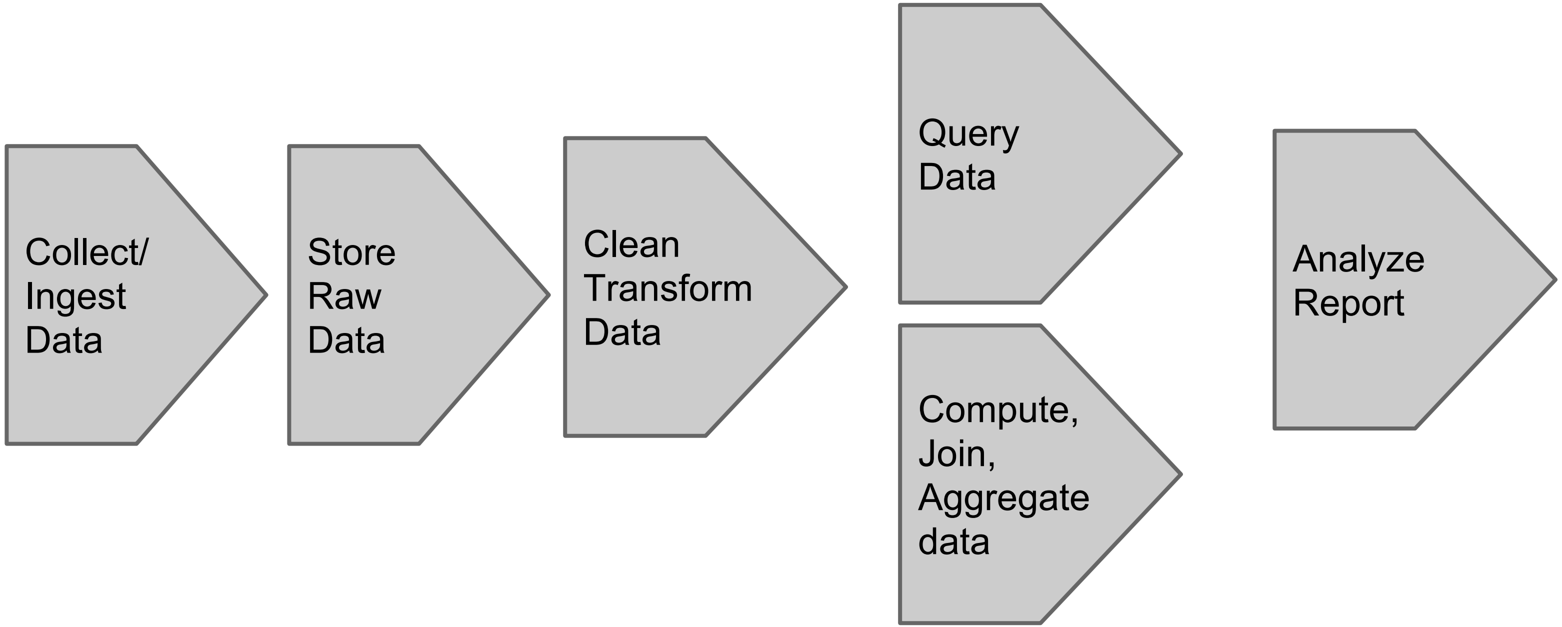


Data Warehouse v.s Data Lake

- Data Transformed to defined schema.
- Loaded when usage identified.
- Allows for quick response of defined queries.

- Many data sources.
- Retain all data.
- Allows for exploration.
- Apply transform as needed.
- Apply schema as needed.

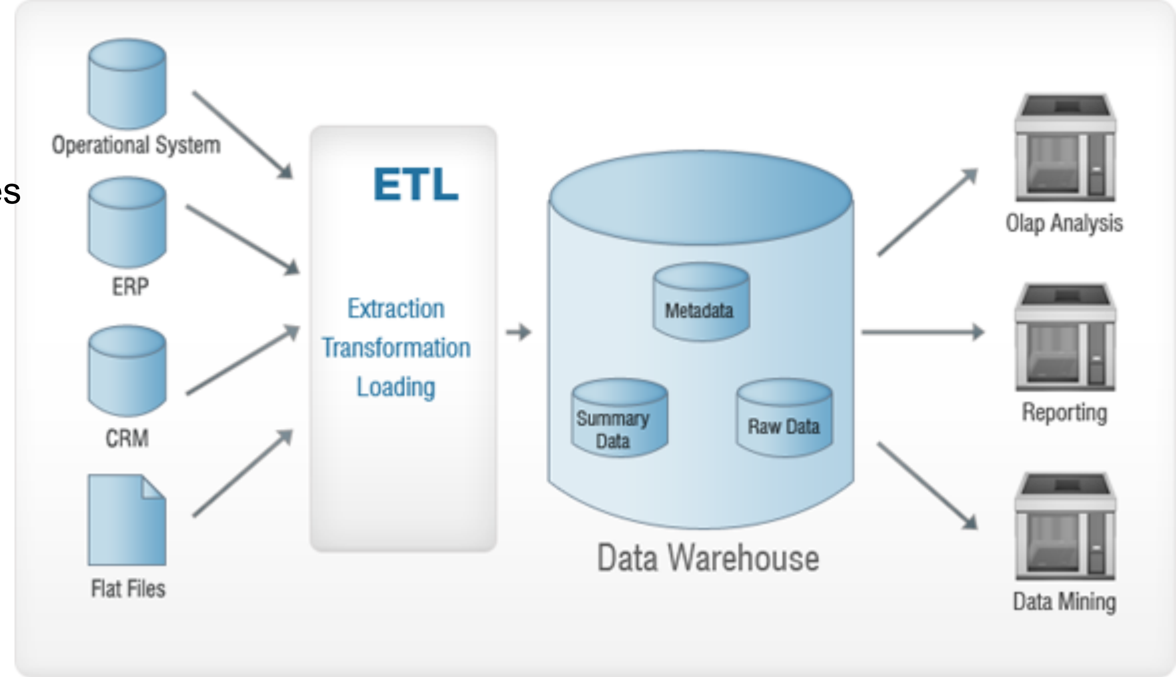
Elements of an Analytics Architecture



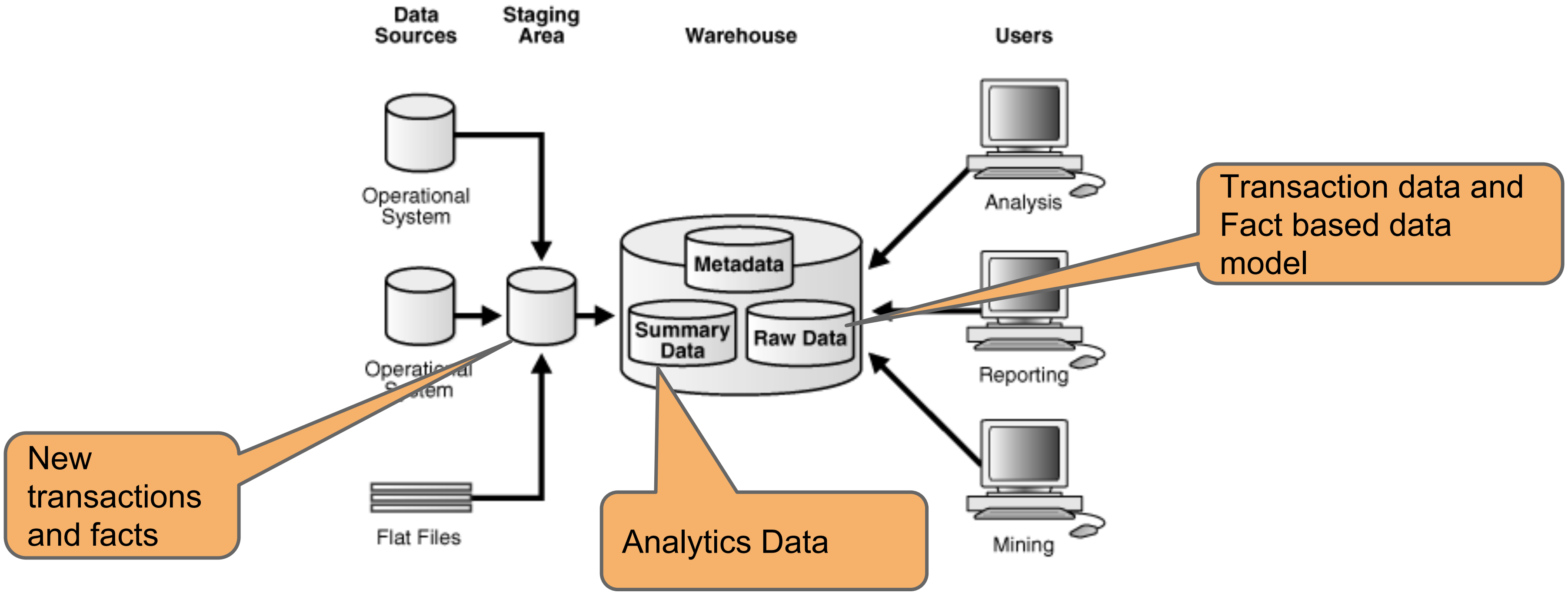
Data warehouse architectures

Key points:

- Extract needed data.
- Map to schema
- Prepare for defined use cases



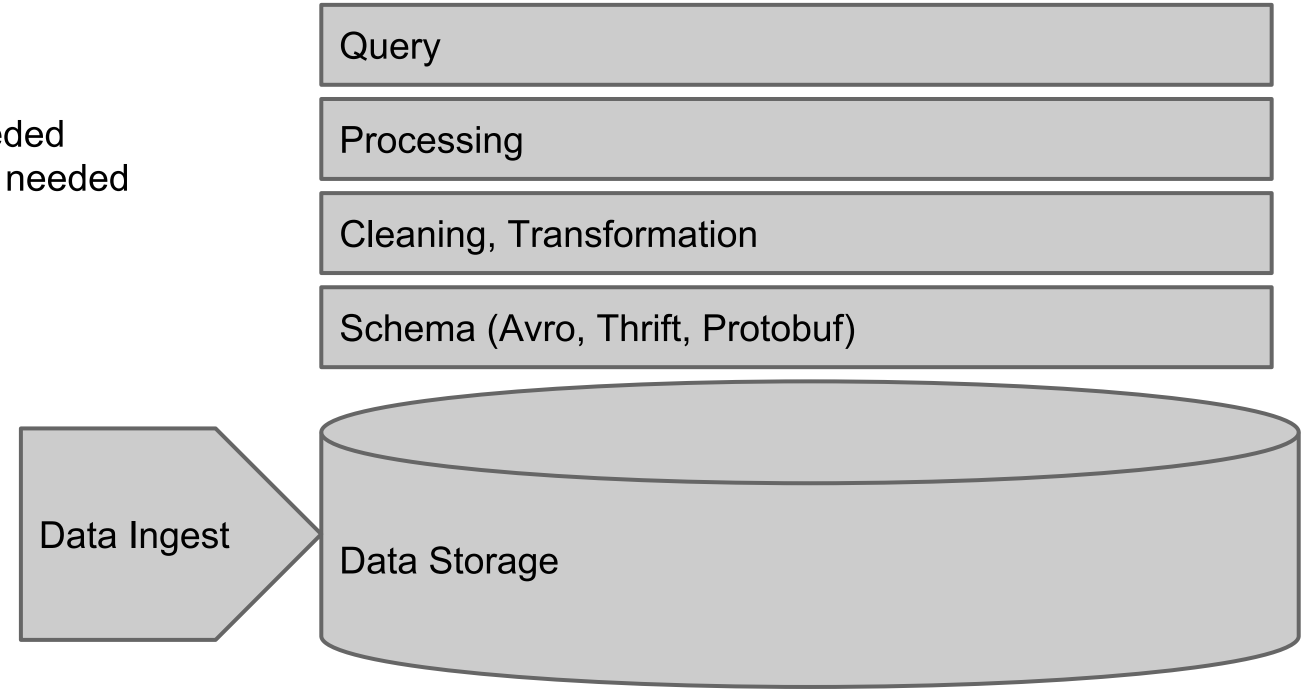
Traditional Business Warehouse



Data Lake

Key points:

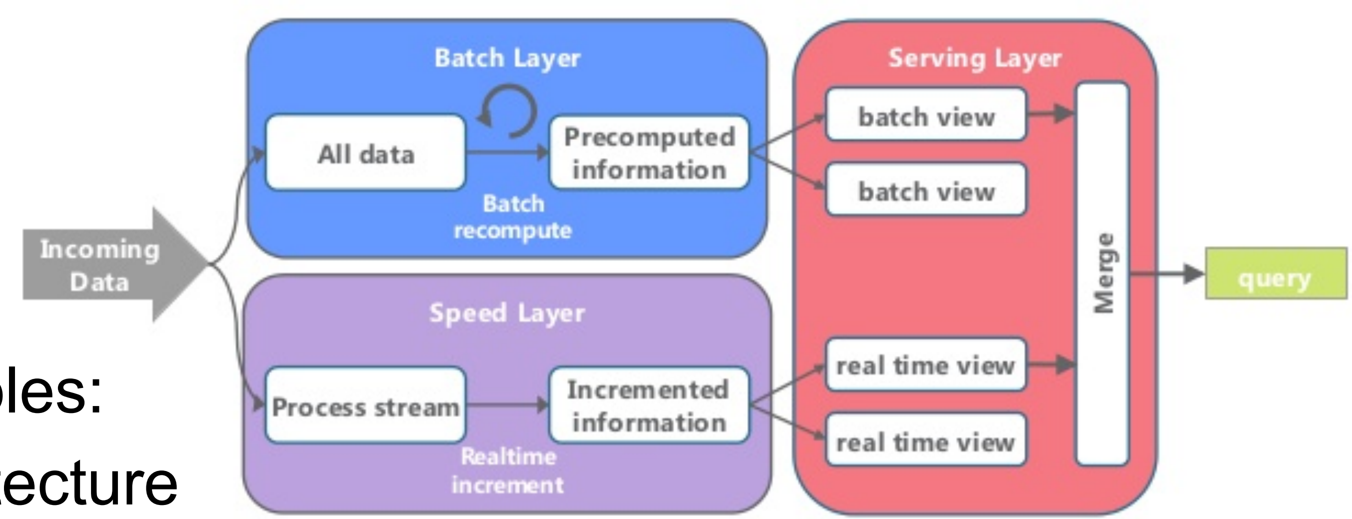
- Store all data
- Transform as needed
- Apply schema as needed



Big Data Analytics Architecture

Example:

Lambda Architecture

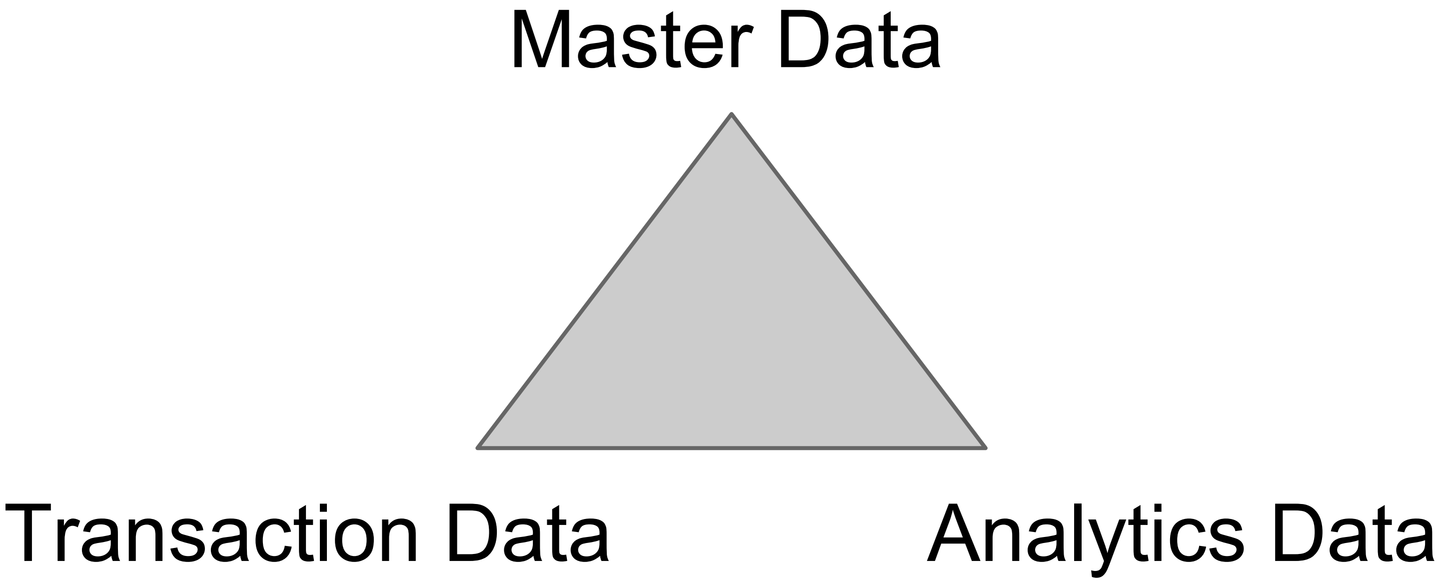


Other examples:

Kappa Architecture

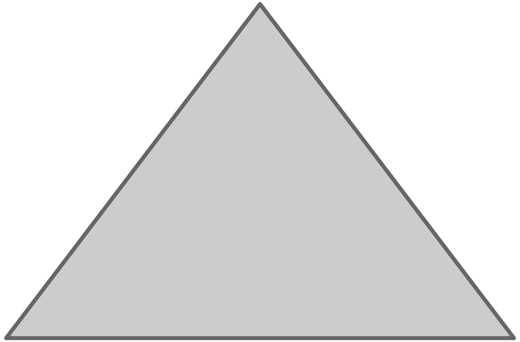
Netflix Architecture

The Data



The Data: Data warehouse model

Master Data
(Dimension Table)

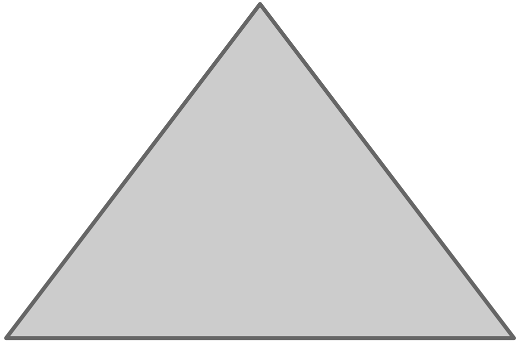


Transaction Data
(Fact table)

Analytics Data
(Cuboid)

The Data: Big Data Architecture

Master Data
(fact based, Immutable, Dimensions)



Transaction Data
(Log items)

Analytics Data
(Aggregates, Roll-ups)

Mutable table

User information					
id	name	age	gender	employer	location
1	Alice	25	female	Apple	Atlanta, GA
2	Bob	36	male	SAS	Chicago, IL
3	Tom	28	male	Google	San Francisco, CA
4	Charlie	25	male	Microsoft	Washington, DC
...	...	...	...	...	...

Should Tom move to
a different city, this value
would be owerwritten.

Fact Based, Immutable

Name data		
user id	name	timestamp
1	Alice	2012/03/29 08:12:24
2	Bob	2012/04/12 14:47:51
3	Tom	2012/04/04 18:31:24
4	Charlie	2012/04/09 11:52:30
...	...	...

Age data		
user id	age	timestamp
1	25	2012/03/29 08:12:24
2	36	2012/04/12 14:47:51
3	28	2012/04/04 18:31:24
4	25	2012/04/09 11:52:30
...	...	...

1 Each field of user information is kept separately.

Location data		
user id	location	timestamp
1	Atlanta, GA	2012/03/29 08:12:24
2	Chicago, IL	2012/04/12 14:47:51
3	San Francisco, CA	2012/04/04 18:31:24
4	Washington, DC	2012/04/09 11:52:30
...	...	...

2 Each record is timestamped when it is stored.

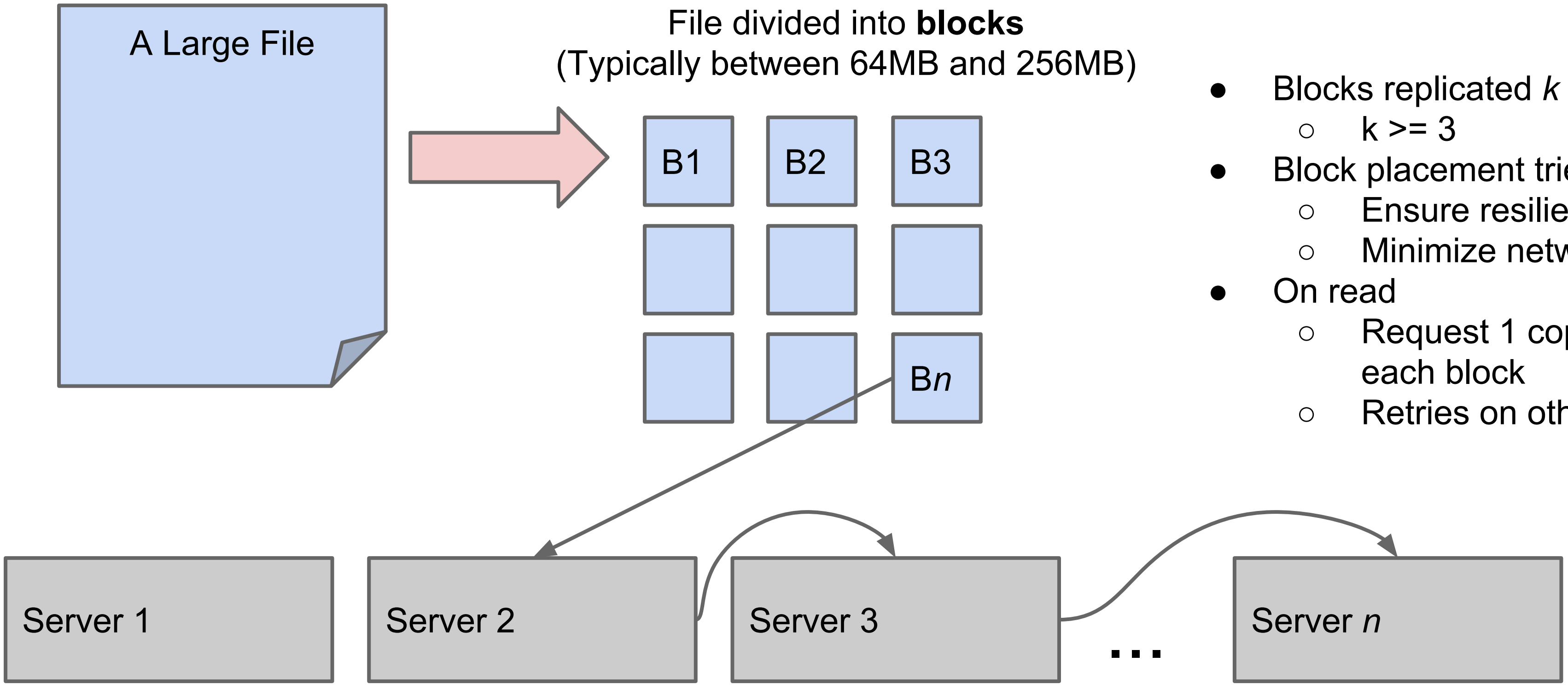
NOSQL and HDFS

Two distributed approaches to data storage

- HDFS (Hadoop Distributed File System)
  - Presents like a local filesystem
  - Distribution mechanics handled automatically

- NoSQL Databases
  - Typically store records as “key-value pairs”
  - Distribution mechanics tied to record keys

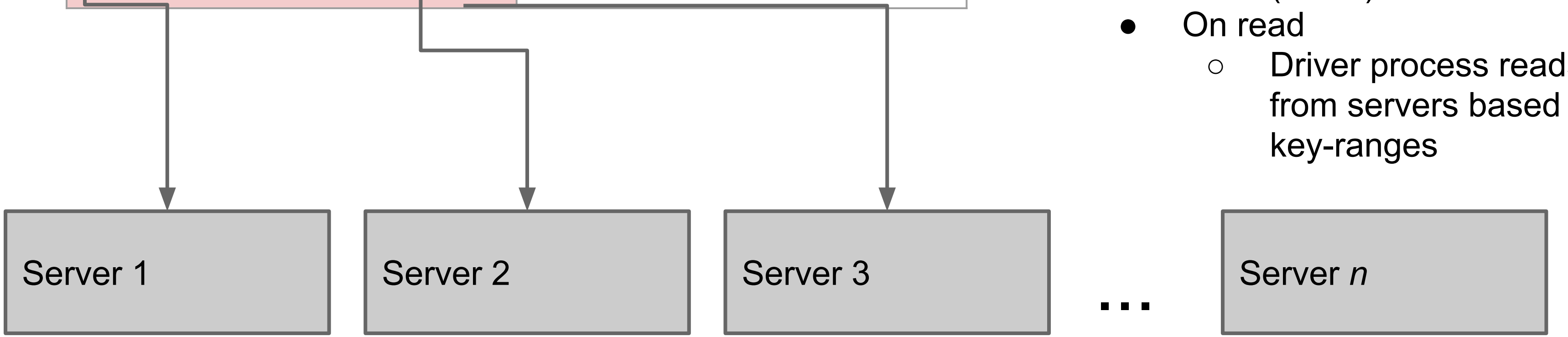
How Does HDFS Work?



- Blocks replicated *k* times
  - k >= 3
- Block placement tries to
  - Ensure resiliency
  - Minimize network hops
- On read
  - Request 1 copy of each block
  - Retries on other copies

How Do Key-Value Stores Work?

Key	Value
1234	Some text like this
2345	{name: “A JSON Document”, number: 7}
3456	<image data>



- Records stored by **key**
  - Provides a simple index
- Record placement
  - Keys are used to *shard*
  - All keys in a certain range go to a certain (set of) servers
- On read
  - Driver process reads from servers based on key-ranges

When Are They Useful?

HDFS

- Popular bulk data store
- Many, large files
  - File size >= Block size
- Agnostic to file content
- Good as an immutable data store
- Good for parallel reads of lots of blocks
- Bad for small, specific reads
- Bad for fast writes

Key-Value Stores

- Many popular choices
  - Redis, Berkeley DB
  - MongoDB
  - Cassandra (column-families imposed on value)
- Good for fast, key-based access
- Good for fast writes
- Bad for off-key access
- Complicated for merging datasets

Relational and Columnar

Two kinds of *database management systems*

- Relational Databases
  - Presents via Declarative Query Languages
  - Organize underlying storage **row-wise**
    - Sometimes column-wise

- Columnar Databases
  - Presents via API and Declarative Query Languages
  - Organize underlying storage **column-wise**

How Do Relational DBs Work?

Logical Schema

<RowID>	EmployeeID	FirstName	LastName	Dept	Salary
001	12	John	Smith	Marketing	90
002	24	Sue	Richards	Engine	130
003	35	Maggie	Smith	DataSci	120
004	44	Bobby	Jones	DataSci	120

Physical Schema

```
001:12,John,Smith,Marketing,90\n
002:24,Sue,Richards,Engine,130\n
003:35,Maggie,Smith,DataSci,120\n
004:44,Bobby,Jones,DataSci,120\n
```

Row-Oriented

- Very fast to insert rows
- Very fast to retrieve **whole** rows
- Slower to retrieve whole columns

- Records stored as **tables**
  - Schema validated **on-write**
  - Typically **indexed**
- Records may be persisted
  - Row-wise
  - Column-wise
- Additional structures can be applied to enhance access
  - Secondary Indices
  - Materialized Views
  - Stored Procedures

How Do Columnar DBs Work?

Logical Schema

<RowID>	EmployeeID	FirstName	LastName	Dept	Salary
001	12	John	Smith	Marketing	90
002	24	Sue	Richards	Engine	130
003	35	Maggie	Smith	DataSci	120
004	44	Bobby	Jones	DataSci	120

Physical Schema

```
12:001;24:002;35:003;44:004\n
John:001;Sue:002;Maggie:003;Bobby:004\n
Smith:001,003;Richards:002;Jones:004\m
Marketing:001;Engine:002; DataSci:003,004\n
90:001;130:002; 120:003,004\n
```

- Records stored as **tables**
- Largely for analytical workloads
  - Read-mostly
  - Bulk-insertion
- Additional access structures
- **Presumption**
  - More likely to read all values of a column than all values of a row
  - Optimize storage for fast column retrieval

Column-Oriented

- Very fast to retrieve columns
- May save space for sparse data
- Slow to insert, slow for row reads

When are They Useful?

Relational

- Most common data storage and retrieval system
- Many drivers, declarative language (SQL)
- Good for fast inserts
- Good for (some) fast reads
- Good for sharing data among applications
- Limited schema-on-read support
- Can be costly or difficult to scale

Columnar

- Optimized for analytical workloads
- Maintains relational data model
- Good for analytical operations
- Good for horizontal scaling
- Bad for fast writes
- Bad for fast row-wise reads

Software Defined Object Storage

(1) Object Storage: as a new abstraction for storing data.

(2) Software Defined Storage: An architecture that enables cost effective, scalable, highly available storage systems.

Combining OS and SDS provided an efficient solution for certain data applications.

Examples

Oracle Storage Service	OpenStack SWIFT based Storage. HA, scalable with eventual consistency.
Amazon S3	HA, scalable storage with eventual consistency.
Google Cloud Storage	HA, scalable storage with strong consistency.
Windows Azur Storage	HA, scalable storage with strong consistency.
Rackspace Files	OpenStack SWIFT based Storage. HA, scalable with eventual consistency.

Data Management

Understand the concept of

- data lineage
- data provenance
- data governance