Computing Platforms, Storage Systems, Data Definition
Big Data Technology & Analytics

Data Ingestion
ETL, Distcp, Kafka, OpenRefine, ...

Storage Systems
HDFS, RDBMS, Column Stores, Graph Databases

Stream Processing Platforms
Storm, Spark, ...

Query & Exploration
SQL, Search, Cypher, ...

Batch Processing Platforms
MapReduce, SparkSQL, BigQuery, Hive, Cypher, ...

Data Definition
SQL DDL, Avro, Protobuf, CSV

Computing Platforms
Distributed Commodity, Clustered High-Performance, Single Node

Data Serving
BI, Cubes, RDBMS, Key-value Stores, Tableau, ...
Big Data Technology & Analytics

Computing Platforms
Distributed Commodity, Clustered High-Performance, Single Node

Data Ingestion
ETL, Distcp, Kafka, OpenRefine, ...

Query & Exploration
SQL, Search, Cypher, ...

Stream Processing Platforms
Storm, Spark, ..

Batch Processing Platforms
MapReduce, SparkSQL, BigQuery, Hive, Cypher, ...

Data Definition
SQL DDL, Avro, Protobuf, CSV

Storage Systems
HDFS, RDBMS, Column Stores, Graph Databases

Data Serving
BI, Cubes, RDBMS, Key-value Stores, Tableau, ...

Query & Exploration
SQL, Search, Cypher, ...

Stream Processing Platforms
Storm, Spark, ..

Batch Processing Platforms
MapReduce, SparkSQL, BigQuery, Hive, Cypher, ...

Data Definition
SQL DDL, Avro, Protobuf, CSV

Storage Systems
HDFS, RDBMS, Column Stores, Graph Databases

Data Serving
BI, Cubes, RDBMS, Key-value Stores, Tableau, ...
Computing Platforms

Computing

- Single Node Computing
  - CPU
  - GPU
- Parallel Computing
- Distributed Computing
  - Grid Computing
  - Cluster Computing
Single Node Computing

- A single node (usually multiple cores)
- Attached to a data store (Disc, SSD, …)
- One process with potentially multiple threads
**Single Node Example**

R: All processing is done on one computer

BidMat: All processing is done on one computer with specialized HW
Single Node
In memory
Retrieve/Stores from Disc

Pros
- Simple to program and debug

Cons
- Can only scale-up
- Does not deal with large data sets
Single Node solution for large scale exploratory analysis
Specialized HW and SW for efficient Matrix operations

Elements:
- Data engine software for optimized operations
- HW design pattern for balancing Storage, CPU and GPU computing
- Optimized machine learning package
- Advanced communication patterns
Parallel Computing

Common Data Store
Many Processors
Parallel execution of tasks
Processor communication
**R Snow: Parallel R**

**R** is a single thread computing application  
**R Snow** enable multi threading/distribution

**Pros**
- Distributed/parallel
- Commonly known tool and model

**Cons**
- Each node requires access to all data
Connected processors collaborate to achieve a common goal

Requires:
- Message passing
- Coordination
- Scheduling
- Tolerate failures on individual nodes
Cluster Computing

- Uniform nodes
- Data shards in a distributed storage
Cluster Computing Example

- Hadoop
- BigQuery
- Pregel
- Spark
- ...

Dremel/BigQuery – General Model

Dremel Architecture
- Partial Reduction
- Diskless data flow
- Long lived shared serving tree
- Columnar Storage

Diagram:
- Distributed Storage
- Mixer 0
- Mixer 1
- Leaf

Chart:
- Node connectivity and data flow
Each node does computation
Each node can be distributed
Information is passed between nodes
Execution is coordinated amongst nodes
Grid Computing

- Distributed Nodes
- Heterogeneous and Physically Separate Nodes
Grid Computing Example

- SETI@home (SETI Institute)
- Large Hadron Collider Computing Grid (CERN)
- NFCR Centre for Comp. Drug Discovery (Oxford Univ)
- Globus Toolkit (Globus Alliance)
- ...

Data Storage: Data Warehouse vs Data Lakes
Data Warehouse
- Data Transformed to defined schema
- Loaded when usage identified
- Allows for quick response of defined queries

Data Lake
- Many data sources
- Retain all data
- Allows for exploration
- Apply transform as needed
- Apply schema as needed
Key points:
- Extract needed data
- Map to schema
- Prepare for defined use cases
Data Lake

Key points:
- Store all data
- Transform as needed
- Apply schema as needed

Query
Processing
Cleaning, Transformation
Schema (Avro, Thrift, Protobuf)

Data Ingest
Data Storage
Data Characteristics

- Understanding of data models and schemas in traditional BW
- Understanding of data models and schemas proposed for big data systems
New transactions and facts

Transaction data and Fact based data model

Analytics Data
Online Transaction Processing (OLTP) vs Online Analytics Processing (OLAP)

OLTP: Systems and approach that facilitate and manage transaction-oriented applications, typically for data entry and retrieval transaction processing.

OLAP: Systems and approach to answering multi-dimensional analytical queries quickly.
OLTP vs OLAP

OLTP
CRUD Transactions (Create, Read, Update, and Delete)
Frequent Updates

Example query:
UPDATE Employees
SET Salary='100,000'
WHERE EmployeeName='Alfred Nobel';
SELECT Salary FROM Employees
WHERE Salary ='Alfred Nobel';

OLAP
Aggregations
Drill-downs,
Roll-ups

Example query:
select o.customerid, o.orderid,
o.orderdate, p.price, sum
(p.price) over (partition by
o.customerid order by o.orderdate)
running_total
**Master Data**: The source of truth in your system, cannot withstand corruption

**Transaction Data**: Keeps relationships (e.g., who bought what)

**Analytics Data**: Aggregated/processed transaction data
Data Warehouse Model

Master Data
(Dimension Tables)

Transaction Data
(Fact table)

Analytics Data
(Cuboid)
Captures concepts and relationships that participate in transactions

*object, relationship, object attributes*
## Transaction Data (Fact Table)

Each Record Captures a Transaction

*(id, object, amount, time, ...)*

<table>
<thead>
<tr>
<th>Complaint ID</th>
<th>Product</th>
<th>Sub-product</th>
<th>Issue</th>
<th>Sub-issue</th>
<th>State</th>
<th>ZIP code</th>
<th>Submitted via</th>
<th>Date received</th>
<th>Date sent to.cc</th>
<th>Date sent to IIB</th>
<th>Company</th>
<th>Company resp</th>
<th>Timely response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1351460</td>
<td>Debt collection</td>
<td></td>
<td>Corrupt attempts collect debt not owed</td>
<td>Debt is not mine</td>
<td>OH</td>
<td>44077</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>In progress</td>
<td>Expert Global Solutions, Inc.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Student loan</td>
<td></td>
<td>Dealing with my lender or servicer</td>
<td></td>
<td>NJ</td>
<td>86070</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>In progress</td>
<td>Transworld Systems Inc.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Credit reporting</td>
<td></td>
<td>Incorrect information on credit report</td>
<td>Account status</td>
<td>IL</td>
<td>60618</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>In progress</td>
<td>PNC (PNC Financial Services, Inc.)</td>
<td>Closed with explanation</td>
<td>Yes</td>
</tr>
<tr>
<td>1351460</td>
<td>Other/phone, health club, etc.</td>
<td>Checking account</td>
<td>Disclosure verification of debt</td>
<td>Right to dispute notice not received</td>
<td>WA</td>
<td>98130</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>Closed with explanation</td>
<td>Stellar Recovery Inc.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Account opening, closing, or management</td>
<td>Checking account</td>
<td>Problems caused by my funds being low</td>
<td></td>
<td>AL</td>
<td>35127</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>Closed with explanation</td>
<td>Wells Fargo</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Account opening, closing, or management</td>
<td>Checking account</td>
<td>Account opening, closing, or management</td>
<td></td>
<td>TX</td>
<td>78577</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>Closed with explanation</td>
<td>Ally Financial Inc.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>FL</td>
<td>34077</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>In progress</td>
<td>HSBC</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>NV</td>
<td>89143</td>
<td>Web</td>
<td>04/29/2015</td>
<td>04/29/2015</td>
<td>Closed with explanation</td>
<td>Nevada Creditco, Inc.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>FL</td>
<td>38252</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>In progress</td>
<td>Transworld Systems Inc.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>AZ</td>
<td>83004</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>In progress</td>
<td>Patents &amp; Files AIC</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>NC</td>
<td>27034</td>
<td>Web</td>
<td>04/29/2015</td>
<td>04/29/2015</td>
<td>In progress</td>
<td>RIS Recovery Solutions</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>CA</td>
<td>90044</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>In progress</td>
<td>Wells Fargo</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>TX</td>
<td>77440</td>
<td>Web</td>
<td>04/29/2015</td>
<td>04/29/2015</td>
<td>Closed with explanation</td>
<td>Commonwealth Financial Systems, Inc.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>TX</td>
<td>75287</td>
<td>Web</td>
<td>04/29/2015</td>
<td>04/29/2015</td>
<td>Closed with explanation</td>
<td>Wells Fargo</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>GA</td>
<td>35114</td>
<td>Web</td>
<td>04/30/2015</td>
<td>04/30/2015</td>
<td>Closed with non-monetary relief</td>
<td>United Security Financial Corp.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>ID</td>
<td>83076</td>
<td>Web</td>
<td>04/29/2015</td>
<td>04/29/2015</td>
<td>Closed with non-monetary relief</td>
<td>Wells Fargo</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>AR</td>
<td>72712</td>
<td>Web</td>
<td>04/29/2015</td>
<td>04/29/2015</td>
<td>In progress</td>
<td>MoneyGram</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1351460</td>
<td>Checking account</td>
<td></td>
<td>Account opening, closing, or management</td>
<td></td>
<td>FL</td>
<td>32068</td>
<td>Web</td>
<td>04/29/2015</td>
<td>04/29/2015</td>
<td>In progress</td>
<td>Global Loan Servicing, LLC</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Each record captures a transaction with details such as id, object, amount, time, and various issues related to the transaction.
## Analytics Data

Roll-ups, drill downs, summaries

<table>
<thead>
<tr>
<th>1. Total Family Households, All Races</th>
<th>Total Age of Householder Mean age</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>3. ALL FAMILIES</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>4. TENURE</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>5. Own/Buying</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>6. Rent</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>7. No Cash Rent /3</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>8. SIZE OF FAMILY</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>9. Two members</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>10. Three members</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>11. Four members</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>12. Five members</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>13. Six members</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>14. Seven members or more</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>15. AGE OF OWN CHILDREN</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>16. Without own children, any age</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>17. With own children, any age</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>18. With own children under 25 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>19. With own children under 25/4</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>20. Without own children under 18 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>21. With own children under 18 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>22. Without own children under 12 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>23. With own children under 12 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>24. Without own children under 8 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>25. With own children under 8 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>26. Without own children under 5 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>27. With own children under 5 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>28. Without own children under 3 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>29. With own children under 3 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>30. Without own children under 1 year</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>31. With own children under 1 year</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>32. Without own children 3-6 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>33. With own children 3-5 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>34. With own children 8-11 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>35. With own children 6-11 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>36. With own children 12-17 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>37. FAMILIES WITH OWN CHILDREN, SPECIFIC AGE/BY</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>38. No own children under 25</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>39. Own children, more than one age group</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>40. Own children 18-24 only</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>41. Own children 12-17 only</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>42. Own children 8-11 only</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>43. Own children 5-7 only</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>44. Own children 3-5 only</td>
<td>Total Age of Householder Mean age</td>
</tr>
<tr>
<td>45. Own children under 3 years</td>
<td>Total Age of Householder Mean age</td>
</tr>
</tbody>
</table>
Big Data Architecture Model

Master Data
(fact based, Immutable, Dimensions)

Transaction Data
(Log items)

Analytics Data
(Aggregates, Roll-ups)
## Mutable table

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>age</th>
<th>gender</th>
<th>employer</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice</td>
<td>25</td>
<td>female</td>
<td>Apple</td>
<td>Atlanta, GA</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>36</td>
<td>male</td>
<td>SAS</td>
<td>Chicago, IL</td>
</tr>
<tr>
<td>3</td>
<td>Tom</td>
<td>28</td>
<td>male</td>
<td>Google</td>
<td>San Francisco, CA</td>
</tr>
<tr>
<td>4</td>
<td>Charlie</td>
<td>25</td>
<td>male</td>
<td>Microsoft</td>
<td>Washington, DC</td>
</tr>
</tbody>
</table>

Should Tom move to a different city, this value would be overwritten.
In the fact-based model, you deconstruct the data into fundamental units called facts.
## Benefits of a fact based model

- Supports queries about any time in history
- Enables complex queries (that a predefined schema might have ignored)

<table>
<thead>
<tr>
<th>user id</th>
<th>location</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Atlanta, GA</td>
<td>2012/03/29 08:12:24</td>
</tr>
<tr>
<td>2</td>
<td>Chicago, IL</td>
<td>2012/04/12 14:47:51</td>
</tr>
<tr>
<td>3</td>
<td>San Francisco, CA</td>
<td>2012/04/04 18:31:24</td>
</tr>
<tr>
<td>4</td>
<td>Washington, DC</td>
<td>2012/04/09 11:52:30</td>
</tr>
<tr>
<td>3</td>
<td>Los Angeles, CA</td>
<td>2012/06/17 20:09:48</td>
</tr>
</tbody>
</table>

1. The initial information provided by Tom (user id 3), timestamped when he first joined FaceSpace.

2. When Tom later moves to a new location, you add an additional record timestamped by when you received the new data.
**Benefits of a fact based model**

More easily correctable (remove erroneous facts)

<table>
<thead>
<tr>
<th>user id</th>
<th>location</th>
<th>timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Atlanta, GA</td>
<td>2012/03/29 08:12:24</td>
</tr>
<tr>
<td>2</td>
<td>Chicago, IL</td>
<td>2012/04/12 14:47:51</td>
</tr>
<tr>
<td>3</td>
<td>San Francisco, CA</td>
<td>2012/04/04 18:31:24</td>
</tr>
<tr>
<td>4</td>
<td>Washington, DC</td>
<td>2012/04/09 11:52:30</td>
</tr>
<tr>
<td>3</td>
<td>Los Angeles, CA</td>
<td>2012/06/17 20:09:48</td>
</tr>
</tbody>
</table>

Human faults can easily be corrected by simply deleting erroneous facts. The record is automatically reset by using earlier timestamps.
Storage Systems
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?

- 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

### Logical Schema

<table>
<thead>
<tr>
<th>&lt;RowID&gt;</th>
<th>EmployeeID</th>
<th>FirstName</th>
<th>LastName</th>
<th>Dept</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>12</td>
<td>John</td>
<td>Smith</td>
<td>Marketing</td>
<td>90</td>
</tr>
<tr>
<td>002</td>
<td>24</td>
<td>Sue</td>
<td>Richards</td>
<td>Engine</td>
<td>130</td>
</tr>
<tr>
<td>003</td>
<td>35</td>
<td>Maggie</td>
<td>Smith</td>
<td>DataSci</td>
<td>120</td>
</tr>
<tr>
<td>004</td>
<td>44</td>
<td>Bobby</td>
<td>Jones</td>
<td>DataSci</td>
<td>120</td>
</tr>
</tbody>
</table>

### Physical Schema

001:12,John,Smith,Marketing,90\n002:24,Sue,Richards,Engine,130\n003:35,Maggie,Smith,DataSci,120\n004:44,Bobby,Jones,DataSci,120\n
- **Row-Oriented**
  - Very fast to insert rows
  - Very fast to retrieve **whole** rows
  - Slower to retrieve whole columns
How Do Columnar DBs Work?

- 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

### Logical Schema

<table>
<thead>
<tr>
<th>&lt;RowID&gt;</th>
<th>EmployeeID</th>
<th>FirstName</th>
<th>LastName</th>
<th>Dept</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>12</td>
<td>John</td>
<td>Smith</td>
<td>Marketing</td>
<td>90</td>
</tr>
<tr>
<td>002</td>
<td>24</td>
<td>Sue</td>
<td>Richards</td>
<td>Engine</td>
<td>130</td>
</tr>
<tr>
<td>003</td>
<td>35</td>
<td>Maggie</td>
<td>Smith</td>
<td>DataSci</td>
<td>120</td>
</tr>
<tr>
<td>004</td>
<td>44</td>
<td>Bobby</td>
<td>Jones</td>
<td>DataSci</td>
<td>120</td>
</tr>
</tbody>
</table>

### Physical Schema

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

- **EmployeeID**
- **Dept**
- **Salary**

- **Logical Schema**
- **Physical Schema**

<table>
<thead>
<tr>
<th>Logical Schema</th>
<th>Physical Schema</th>
</tr>
</thead>
</table>
| 12:001;24:002;35:003;44:004
John:001;Sue:002;Maggie:003;Bobby:004
**Smith:**001,003;Richards:002;Jones:004
**Marketing:**001;**Engine:**002;**DataSci:**003,004
90:001;130:002;120:003,004 |
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
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 (Key/Value Stores)**
  - Typically store records as “key-value pairs”
  - Distribution mechanics tied to record keys
How Does HDFS Work?

File divided into blocks
(Typically between 64MB and 256MB)

- Blocks replicated $k$ times
  - Typically: $k \geq 3$
- 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?

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Some text like this</td>
</tr>
<tr>
<td>2345</td>
<td>{name: “A JSON Document&quot;, number: 7}</td>
</tr>
<tr>
<td>3456</td>
<td>&lt;image data&gt;</td>
</tr>
</tbody>
</table>

- 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

Server 1  Server 2  Server 3  ...  Server n
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
Two Concepts

- **Object Storage (OS)**: as a new abstraction for storing data
- **Software Defined Storage (SDS)**: An architecture that enables cost effective, scalable, highly available (HA) storage systems

Combining OS and SDS provides an efficient solution for certain data applications
What is it?

Object Storage

- Manage data as one (large) logical object
- Consists of meta data and data
- Unique identifier across system
- Data is an uninterpreted set of bytes

Software Defined Storage

- Data Replication for resilience and HA
- Store anything
- Runs on commodity hardware
- Manages data automatically, blocks hidden

Distributed Storage Manager/Files System

Network

Node

Disc

File

Node

Disc

File
What Problem Does it Solve?

- Simplified model for managing data growth
- Lowers management cost
- Lowers hardware cost
- Meet Resiliency and HA needs and lower cost
Simple API with an Object Abstraction
Data is partitioned, partitions are replicated for resiliency and HA
Software layer
- manages replication and data placement
- automatically re-distributes data in case of expansion, contraction or failure
Examples

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Storage Service</td>
<td>OpenStack SWIFT based Storage. HA, scalable storage with eventual consistency</td>
</tr>
<tr>
<td>Amazon S3</td>
<td>HA, scalable storage with eventual consistency</td>
</tr>
<tr>
<td>Google Cloud Storage</td>
<td>HA, scalable storage with strong consistency</td>
</tr>
<tr>
<td>Windows Azur Storage</td>
<td>HA, scalable storage with strong consistency</td>
</tr>
<tr>
<td>Rackspace Files</td>
<td>OpenStack SWIFT based Storage. HA, scalable storage with eventual consistency</td>
</tr>
</tbody>
</table>

**Strong consistency** offers **up-to-date** data but at the cost of **high latency**
**Eventual consistency** offers **low latency** but may reply to read requests with **stale data** since all nodes of the database may not have the updated data.
It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

**Consistency**: Every read receives the most recent write or an error.

**Availability**: Every request receives a (non-error) response – without guarantee that it contains the most recent write.

**Partition Tolerance**: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.
Moving Large Data

- If data is distributed how can you leverage parallelism?
- What kind of source and sink is involved?
- How do you use network bandwidth efficiently?
- How to handle different formats and structures?
- Large files take a long-time, how are failures handled?

As a data scientist you need to understand how to think about data transfer and movement
Performance Measures

- **Bandwidth** measured in bits/sec is the maximum transfer rate
- **Throughput** is the actual rate that information is transferred
- **Latency** the delay between the sender and the receiver, this is a function of the calls/signals travel time and intermediate processing time
- **Jitter** variation in the time of arrival at the receiver of the information
- **Error rate** # of corrupted bits as a percentage or fraction of the total sent
Limiting Factors:
- Available Network Bandwidth
- Read/Write performance
- Error rates

Some info (for Disk storage we use base 10)
- 1 byte = 8 bits
- 1000 Bytes = 1 Kilobyte
- 1000 Kilobytes = 1 Megabyte
- 1000 Megabytes = 1 Gigabyte
- 1000 Gigabytes = 1 Terabyte

Example
How long would it take to transfer 2 TB file over a 100 Mbit/sec connection?

Answer
2TB = 2*1000*1000*1000*1000*8 = 1.6E+13 bit
100 Mbit = 100*1000*1000 = 100000000
seconds = 1.6E+13 / 100000000 = 160000
hours = 160000 / 60 / 60 = 44 hours
## Example Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sqoop</td>
<td>RDBMS, BDW to Hadoop</td>
</tr>
<tr>
<td>distcp2</td>
<td>HDFS to HDFS copy</td>
</tr>
<tr>
<td>Rsync</td>
<td>FS to FS copy, FS to FS synchronization</td>
</tr>
</tbody>
</table>
## Tools Comparison

<table>
<thead>
<tr>
<th>Type(s) of source and sink</th>
<th>Sqoop</th>
<th>rsync</th>
<th>Distcp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RDBMS to HDFS</td>
<td>A unix / linux system inter transfer tool</td>
<td>HDFS to HDFS Storage Service (S3) to HDFS</td>
</tr>
<tr>
<td>Network usage</td>
<td>High</td>
<td>Smart use of bandwidth for sync</td>
<td>Uses available bandwidth</td>
</tr>
<tr>
<td>Level of parallelism</td>
<td>Low</td>
<td>No built-in parallelism</td>
<td>High. Configurable</td>
</tr>
<tr>
<td>Structures and formats</td>
<td>Per table or free form SQL</td>
<td>Agnostic to file content</td>
<td>Agnostic to structure in files</td>
</tr>
<tr>
<td>Resilience to failures</td>
<td>If task fails it will be rolled back and result in a partial export</td>
<td>Need to be reinitiated/restarted</td>
<td>High, will retry per job. If one job fails it will be restarted</td>
</tr>
</tbody>
</table>
Summary: Moving Large Data

- Understand the bottleneck and your use case
- Define your time constraints, can you move all the data or do you need to segment
- Pick the right tool
SQL DDL, Avro, Protobuf, CSV

Data Definition
Schemas represent the logical view of data
We can apply them
  ▪ When data is written
  ▪ When data is read
The application of schema comes with trade-offs
Schema-on-Read

- Data is stored without constraint
  - Store data without validation
- Apply the plan to the data stream on every read
  - Extract the record from the data stream
  - Extract the details from the record
  - Validate the record
Examples of Schema-on-Read

- **File systems**
  - The local disk on your computer

- **“Big Data Frameworks”**
  - Apache Hadoop
  - Apache Spark

- **Some NoSQL Databases**
  - MongoDB
  - CouchDB

Rely on runtime logic to apply schema

Rely on self-describing data to apply schema
Consider this Avro schema expressed in JSON

```json
{
    "type": "record",
    "name": "Person",
    "fields": [
        {"name": "userName", "type": "string"},
        {"name": "favoriteNumber", "type": ["null", "long"]},
        {"name": "interests", "type": {"type": "array", "items": "string"}}
    ]
}
```
Other Schema-encoding Formats

- Google protobuf
- messagepack
- Cap’n Proto
- Parquet
  - Designed for large, column-oriented data
Trade-Offs in Schema-on-Read

**Pros**
- Data can be subject to varying interpretation
- Validation can be as strict as needed
- Schema can mutate over time

**Cons**
- Higher cost to read data
  - Assemble records
  - Validate records
  - Access details
- No implicit guarantees about data contents
Data is stored only if it fits constraints
  - Validate the schema before persisting
    - # and type of attributes
    - uniqueness, size, etc.

Extract data quickly, without re-validation
  - Data shape, size, and location is already known
Examples of Schema-on-Write

- Databases (Typically Relational or Object-Relational)
  - MySQL
  - PostgreSQL
  - Oracle Database
  - Microsoft SQL Server
  - and many more
Trade-Offs in Schema-on-Write

Pros
- Data characteristics are guaranteed
- Storage can be optimized to speed retrieval
  - Lower-cost reads

Cons
- Schema must be provided before data is written
- Schema modifications may cause writes to existing data
- Multiple interpretations require
  - Duplicate definitions
  - Sometimes duplicate materializations
BigTable: a Hybrid Approach

- **Column-Family Database**
  - Organize data into a hierarchy
    - Columns → record details
    - Column families → groups of columns
  - Column families are *schema-on-write*
  - Columns are *schema-on-read*
    - Can add columns, interpret bytes variably

- **Examples:**
  - Apache HBase
  - Apache Cassandra
Big Data Architectures
Elements of DAV Architecture

1. Collect/Ingest Data
2. Store Raw Data
3. Clean Transform Data
4. Query Data
5. Compute, Join, Aggregate data
6. Analyze Report
Difference in Approach

Traditional Approach
Structured & Repeatable Analysis

Business Users
Determine what question to ask

IT
Structures the data to answer that question
- Monthly sales reports
- Profitability analysis
- Customer surveys

Big Data Approach
Iterative & Exploratory Analysis

IT
Delivers a platform to enable creative discovery

Business
Explores what questions could be asked
- Brand sentiment
- Product strategy
- Maximum asset utilization
- Preventative care

Notice the difference!
Traditional Business Warehouse

New transactions and facts

Transaction data and Fact based data model
Example:
Lambda Architecture

Other examples:
Kappa Architecture
Netflix Architecture
Key points:
- Keep your data in Kafka and HDFS
- Low latency processing as a stream
- Re-process and batch processing in Hadoop
Kappa Architecture

Key points:
- Keep your data in Kafka
- Treat everything as a stream
- Re-process stream by resetting offset
- Advantage: simplified architecture, everything is a stream
Processing Platforms

MapReduce, Spark, BigQuery, ...
Processing Platforms

- **Batch Processing**
  - Google GFS/MapReduce (2003)
  - Apache Hadoop HDFS/MapReduce (2004)
- **SQL**
  - BigQuery (based on Google Dremel, 2010)
  - Apache Hive (HiveQL) (2012)
- **Streaming Data**
- **Unified Engine (Streaming, SQL, Batch, ML)**
  - Apache Spark (2012)