ALY6110

Data Management and Big Data

Instructor: Valeriy Shevchenko

Northeastern

Lecture 3 (Week 3)

Methods Used in Big Data Analysis

Learning Objectives

This session will explain:

- What is Data Analytics and Data Science
- How Big Data technology is defined
- What is MapReduce
- Details of Lambda Architecture
- The key properties of data
- Advantages of the fact-based model for the master dataset

Big Data

- In the first years of the 20th century, the term **big data** has appeared

Big Data and Data Science

- Big Data, a technology for data processing, is now defined by the ***"four Vs":***
  - **V**olume
  - **V**ariety
  - **V**elocity
  - **V**eracity

- **Volume:** how to store big data (data repositories for large amounts of data)?
- **Variety:** how to put together data from different sources?
- **Velocity**: how to deal with data arriving very fast, in streams known as *data streams*?
- **Veracity:** how to deal with noise and abnormality in data? Is the data that is being stored, and mined meaningful to the problem being analyzed and current to make the right decision?

Big Data and Data Science

- Another term used as a synonym for big data is **data science**
- “Big data are data sets that are too large to be managed by conventional data-processing technologies, requiring the development of new techniques and tools for data storage, processing and transmission.”
- These tools include, for example:
  - MapReduce.
  - Hadoop.
  - Spark.
  - Storm.

Big Data and Data Science

- The Data Volume is not the only characterization of big data.
- The word “big” can refer to:
  - The number of data sources
  - The importance of the data
  - The need for new processing techniques
  - How fast data arrive
  - The combination of different sets of data so they can be analyzed in real time
- Big data major concern is the technology.
- The technology provides a computing environment, not only for analytics, but also for other data processing tasks.

Big Data and Data Science

- Data science is concerned with the creation of models able to extract patterns from complex data and the use of these models in real-life problems
- Data science extracts meaningful and useful knowledge from data, with the support of suitable technologies.
- Data science has a close relationship to analytics and data mining.
- Data science goes beyond data mining by providing a knowledge extraction framework, including statistics and visualization.
- Therefore big data collects and data science discovers.

Big Data Distribution Technique

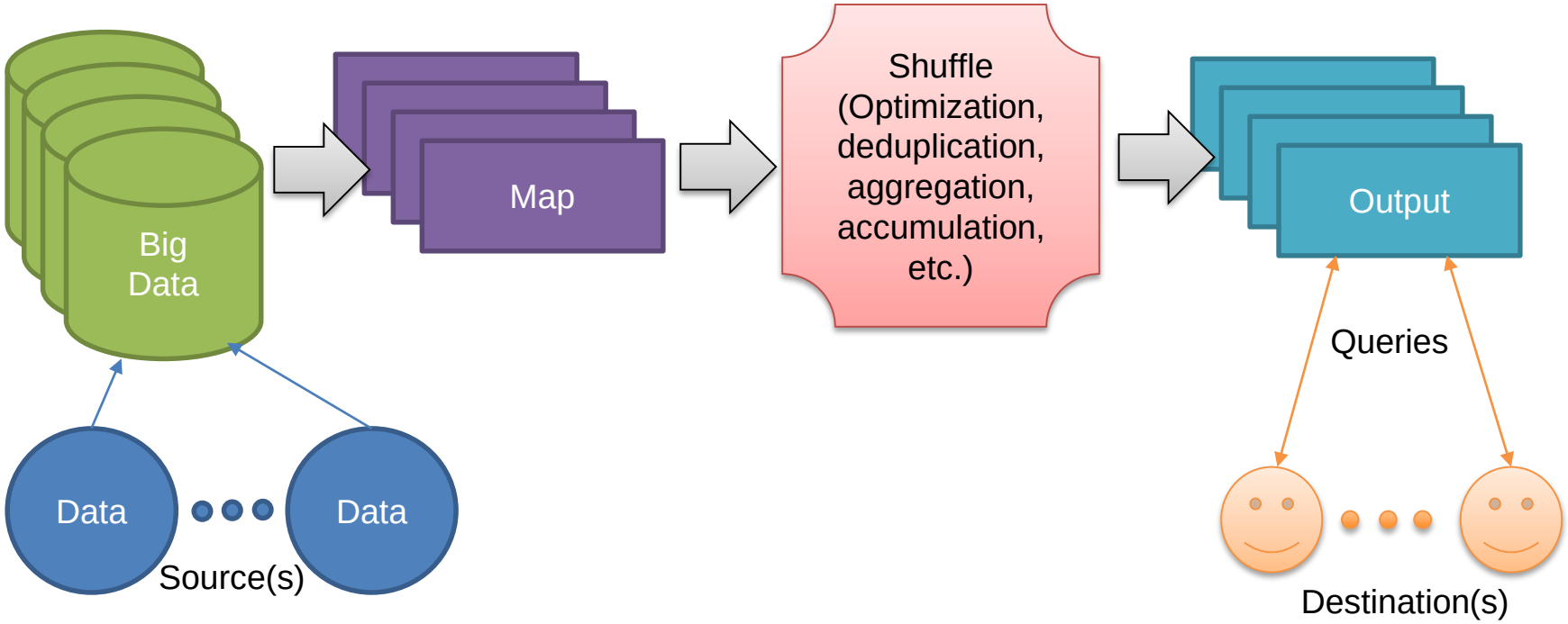
- As data increase in size, velocity and variety, new computer technologies become necessary.
- These new technologies, which include hardware and software, must be easily expanded as more data are processed.
- This property is known as scalability.
- One way to obtain scalability is by distributing the data processing tasks into several computers, which can be combined into clusters of computers.
- Clusters produced by analytics techniques (not computer clusters) is where a data set is partitioned in groups within it

MapReduce

- Even if processing power is expanded by combining several computers in a cluster, creating a distributed system, conventional software for distributed systems cannot keep up with big data.
- One of the limitations is the efficient distribution of data among the different processing and storage units.
- To deal with these requirements, new software tools and techniques have been developed.
- One of the first techniques developed for big data processing using clusters was **MapReduce**.
- MapReduce is a programming model that consists of two steps: map and reduce.
- The most famous implementation of MapReduce is called Hadoop.

MapReduce

- MapReduce is a programming model that has two steps: map and reduce.
- The most famous implementation of MapReduce is called Hadoop.



MapReduce

What is the other term describing the similar process of “transforming” the data?

Anyone?

MapReduce

Data Warehousing

Big Data System Requirements

- To efficiently solve a big data problem, a distributed system must attend the following requirements:

Make sure that no chunk of data is lost and the whole task is concluded.
(If one or more computers has a failure, their tasks, and the corresponding data chunk, must be assumed by another computer in the cluster.)

Repeat the same task, and corresponding data chunk, in more than one cluster computer (redundancy).
(If one or more computer fails, the redundant computer carries on with the task.)

Failed computers can return to the cluster when they are fixed.

Computers can be easily removed from the cluster or extra ones included in it as the processing demand changes.

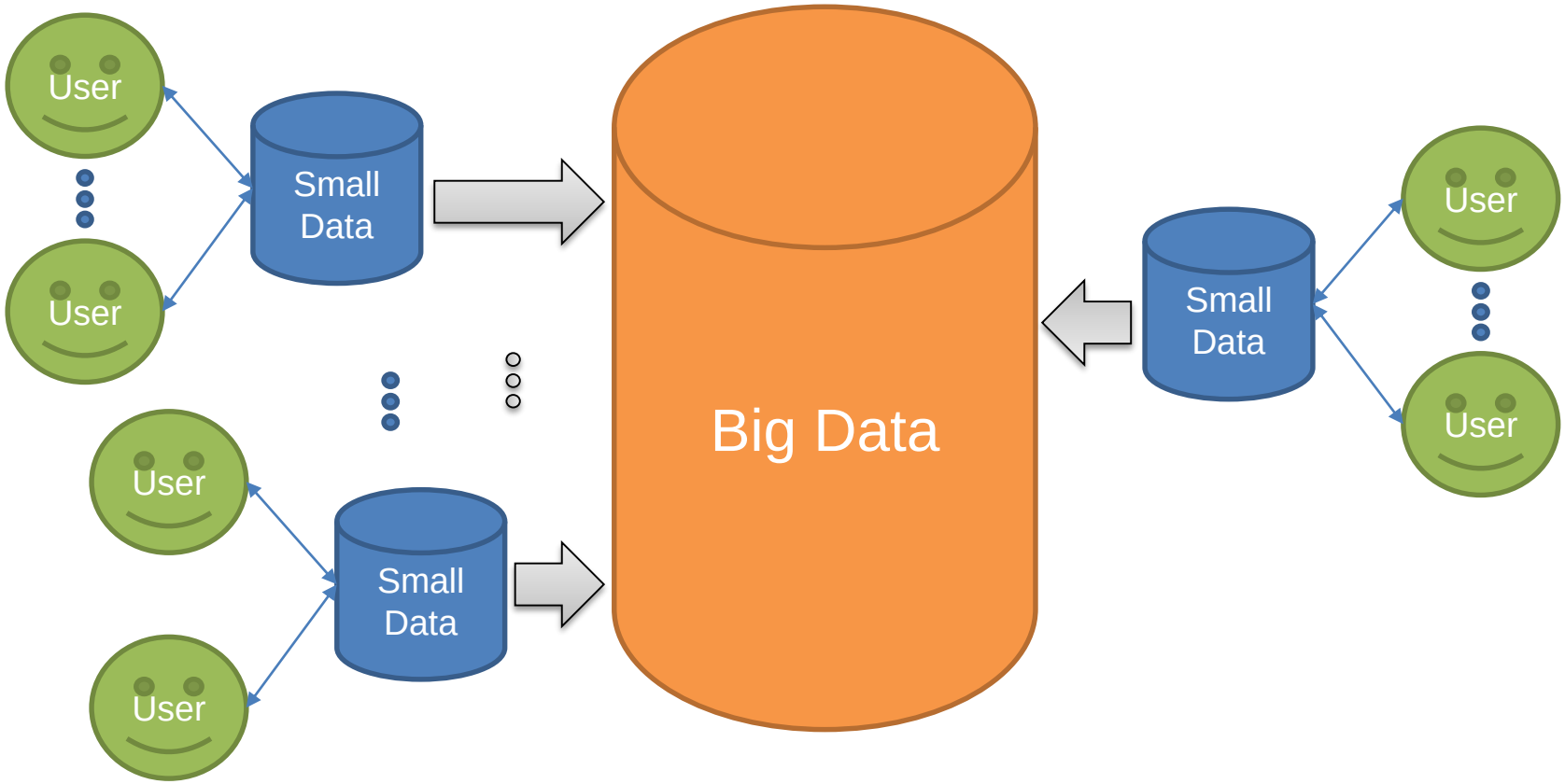
MapReduce Example

- MapReduce divides the data set into parts (chunks) and stores in the memory of each cluster computer the chunk of the data set needed by this computer to accomplish its processing task.

- As an example: suppose that you need to calculate the average salary of 1 billion people and you have a cluster with 1000 computers, each with a processing unit and a storage memory.
- The people can be divided into 1000 chunks (subsets) with data from 1 million people each.
- Each chunk can be processed independently by one of the computers.
- The results (1000) produced by each these computers (the average salary of 1 million people) can be averaged, returning the final salary average.

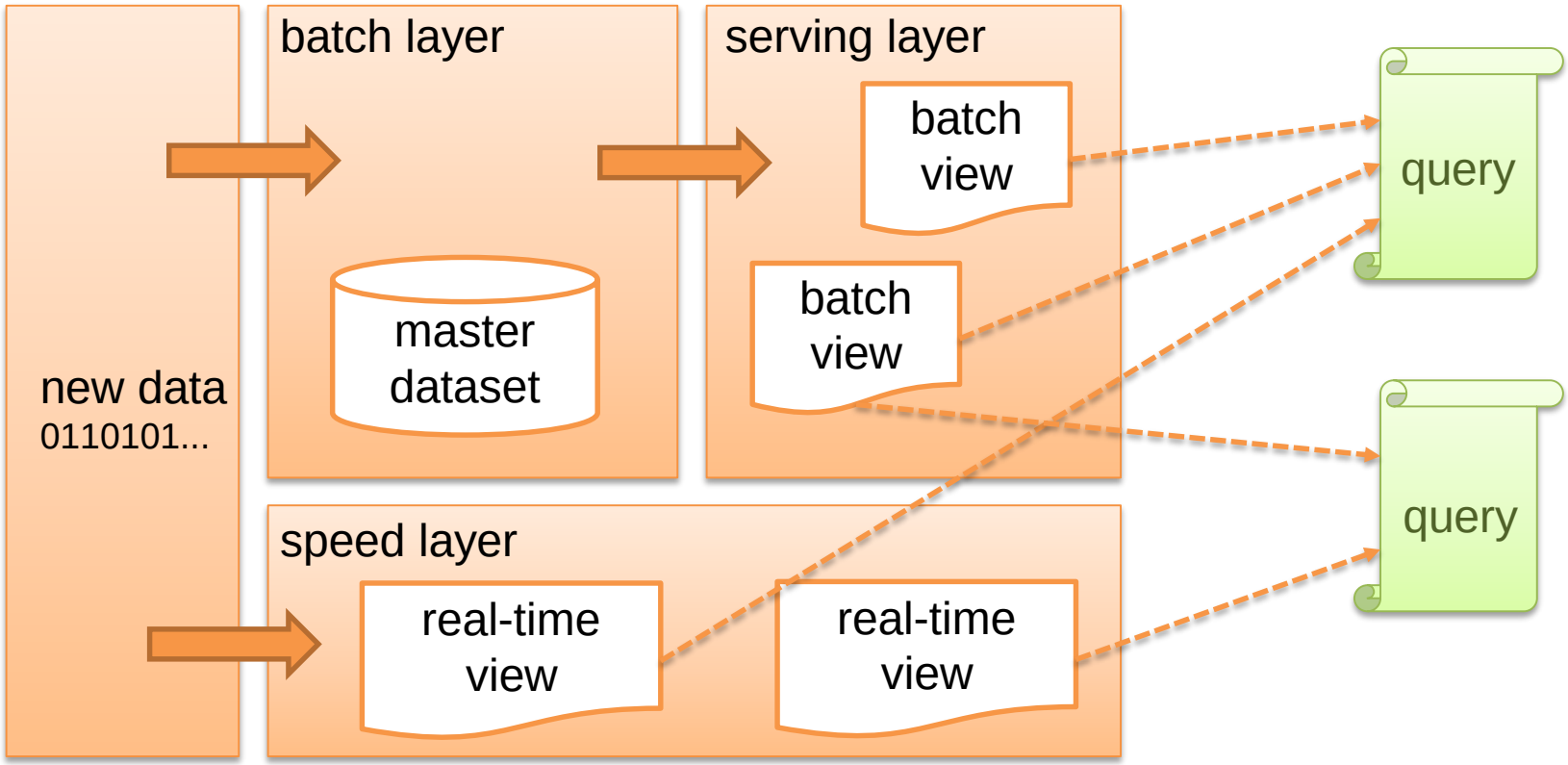
Small Data

To be characterized as small data, a data set must have a size that allows its full understanding by an user. When these data are collected to be stored and processed in large data servers they become big data.

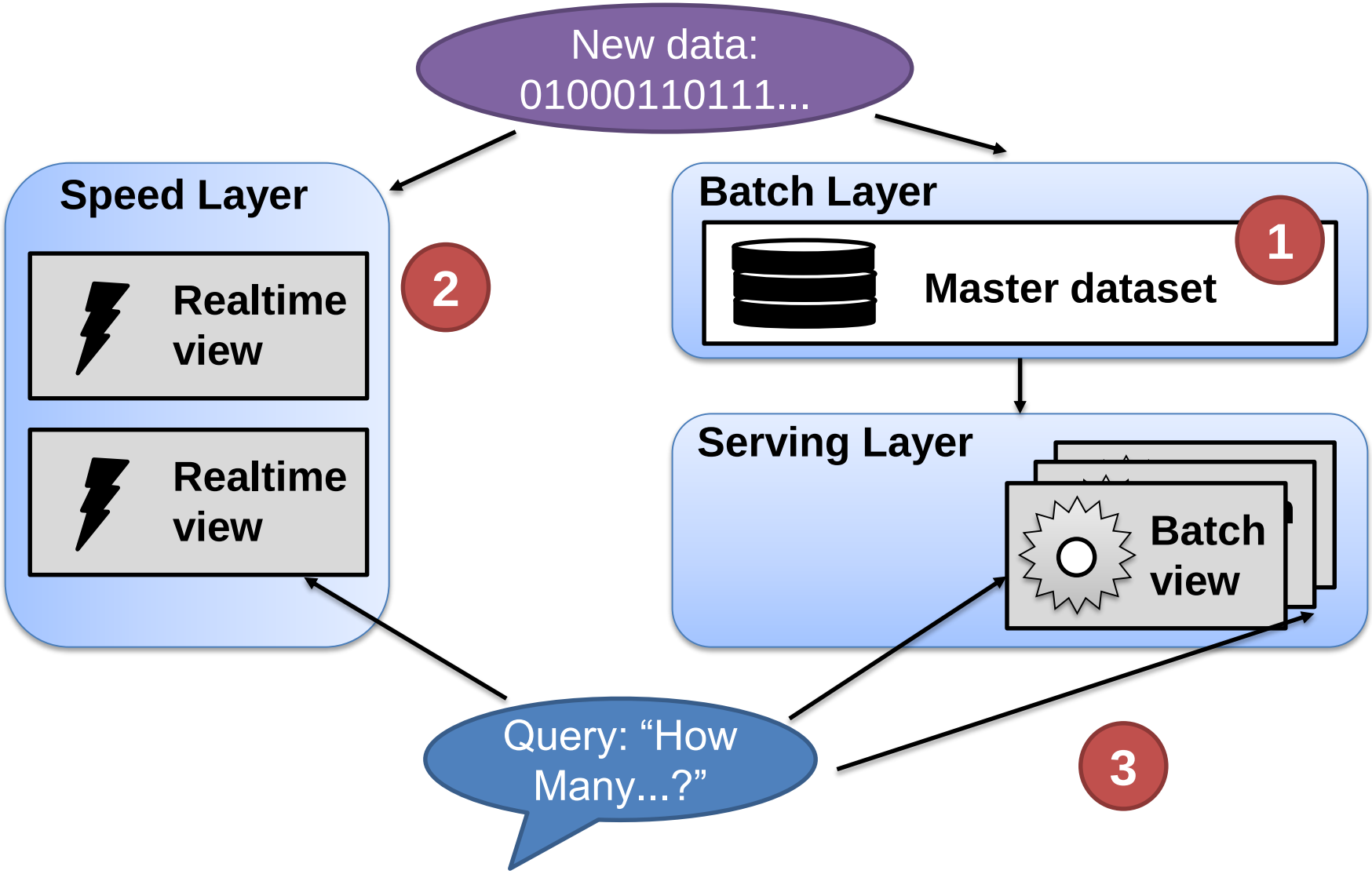


Lambda Architecture

- Lambda Architecture designed to build Big Data systems using series of layers



Data Model for Big Data



Data Model for Big Data

1. The master dataset is the source of truth in Big Data system and cannot withstand corruption.

2. The data in the speed layer realtime views has a high turnover rate, so any errors are quickly expelled.

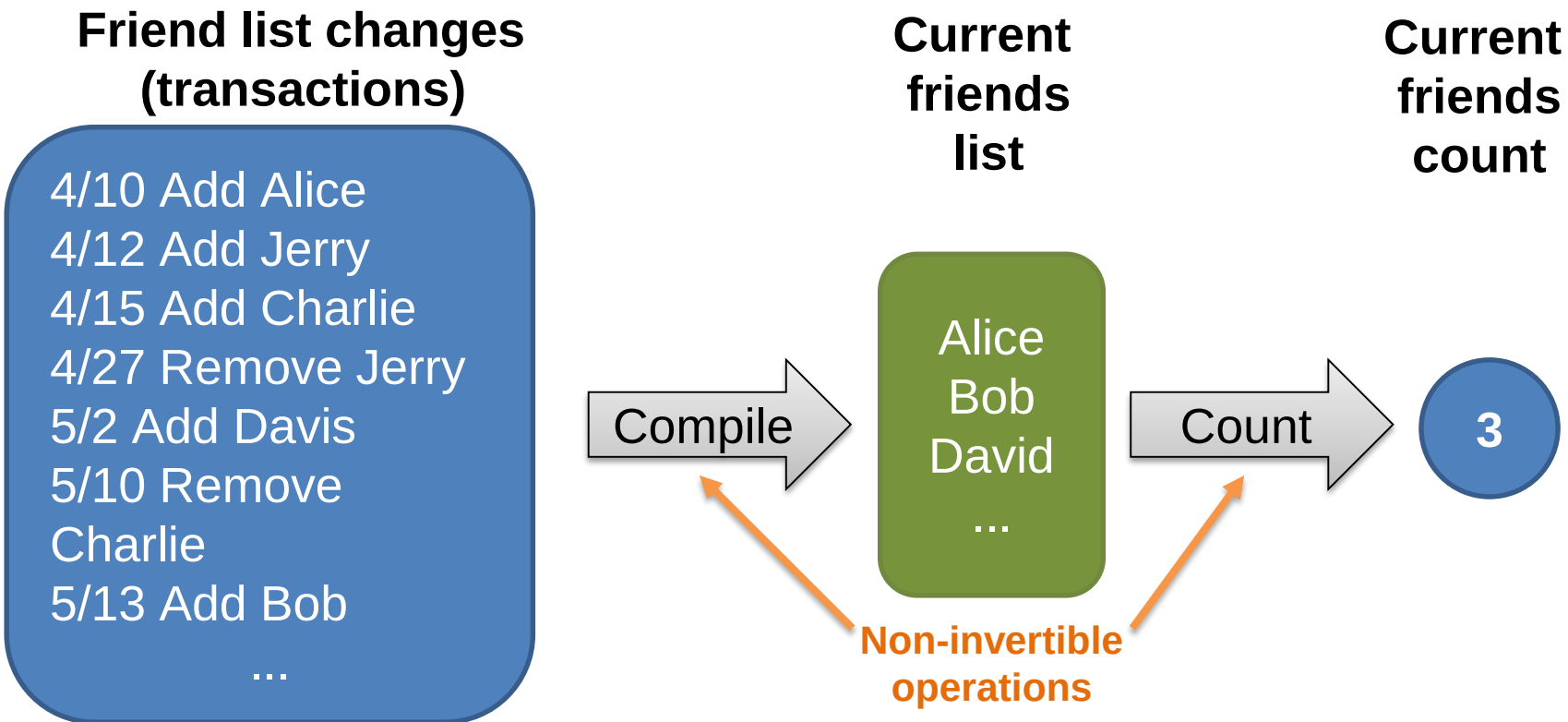
3. Any errors introduced into the serving layer batch views are overwritten because they are continually rebuilt from the master dataset.

The Properties of Big Data

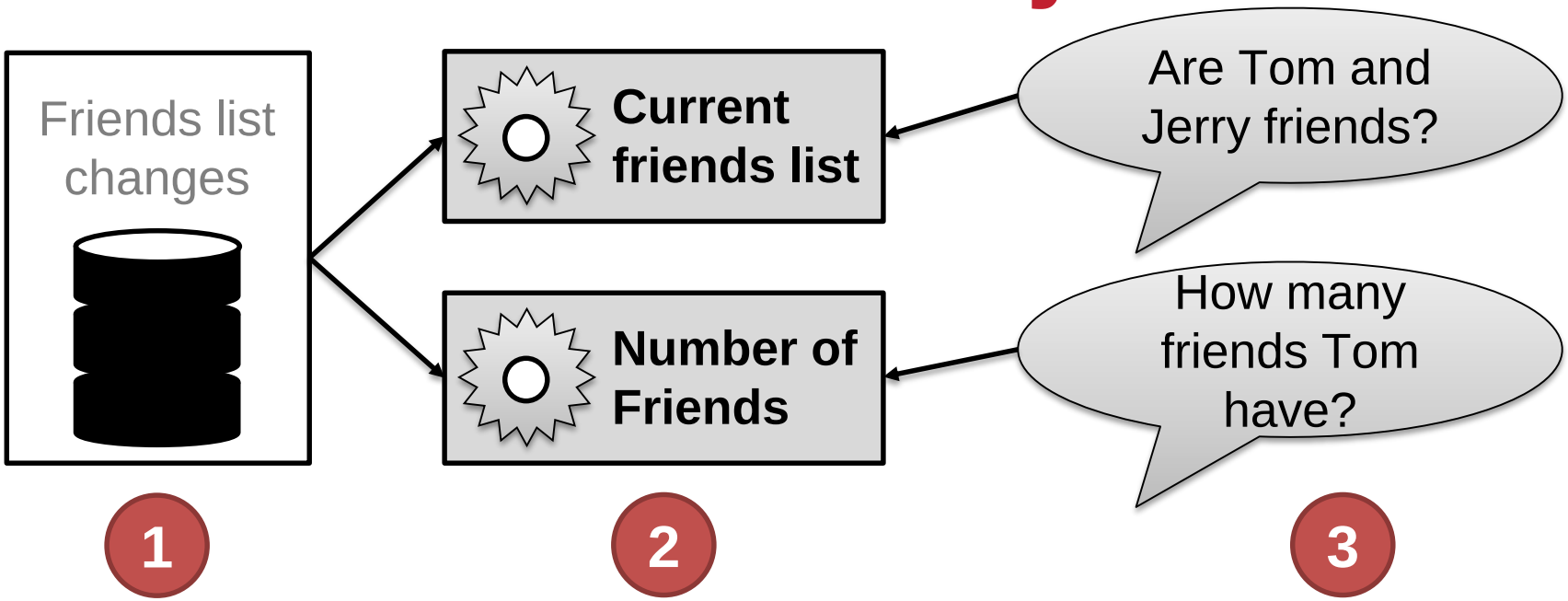
- ***Information*** is the general collection of knowledge relevant to your Big Data system.
- ***Data*** refers to the information that can't be derived from anything else. Data serves as the axioms from which everything else derives.
- ***Queries*** are questions you ask of your data. (query financial transaction history to determine your current bank account balance).
- ***Views*** are information that has been derived from your base data. They are built to assist with answering specific types of queries.

Social Network System

- Example of FaceData social network system (your own FaceBook)
- There are three possible options for storing friendship information for Social Network System.
- Each option can be derived from the one to its left, but it's a one-way process.



Social Network System



The Data is information that cannot be derived from anything else.

The views are computed from the data to help answer queries.

The queries access the information stored in the views.

Why Do We Need Perfomance?

- Prior to Big Data era the term Data Warehousing was widely used.
- The key difference between the two is that Data Warehousing just one part of the Big Data System, which primary responsibility is performance.
- The idea is to pre-calculate the data to serve the most common queries faster – create ***views*** in advance
- The native human behavior is to expect most current data immediately. There is no patience to wait for what is here as of Today.
- At the same time natively people would understand that if historical data needed to be extracted, it should take more time and there is more patience there.

Stock Status Report Example

- Most common example from the business side of Big Data Counting and Management is a Stock Status report.
- Lets say company buys and sells small number of parts (part numbers) but in large volumes
- Each transaction is stored for each part for with the date
- Receiving transaction recorded with "+" quantity amount
- Selling transaction recorded will "-" quantity amount

Transactions History

Date	Part Number	Qty
01/05/2016	Part A	+1,500
02/09/2016	Part A	-350
02/09/2016	Part B	+25,000
02/10/2016	Part C	+15,000
02/11/2016	Part B	-5,000
03/01/2016	Part C	+12,000
03/01/2016	Part B	-5,500
03/02/2016	Part C	-10,000
03/03/2016	Part A	+12,000
03/04/2016	Part B	- 1,000
...		

Stock Status Report Example

- To calculate the current stock status report, we use a simple formula:

  For each Part Number, calculate the summary or quantity

- As a result we have a table were each part only shown once

- The quantity field shows the total quantity of this part we currently have in stock

- It runs every day as the decision has t be made to spend more money buying more parts or not

Stock Status Report as of Now

Part Number	Current Qty
Part A	13,150
Part B	13,500
Part C	17,000

Stock Status Report Example

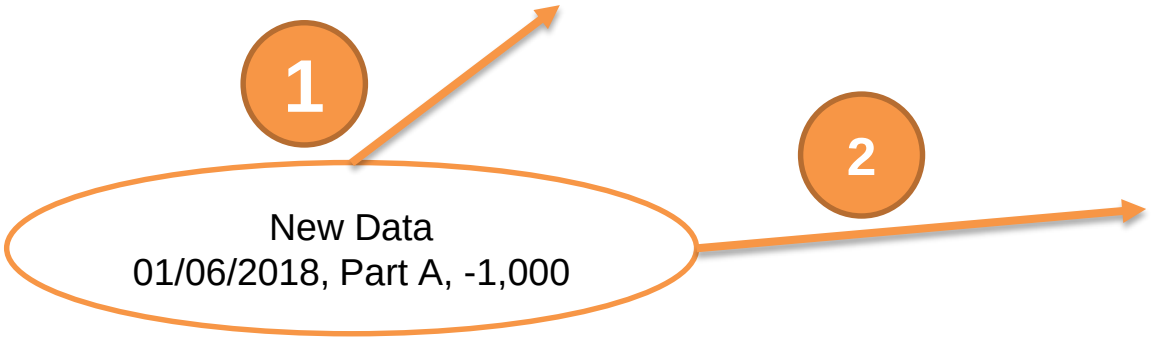
- The system designed to calculate this report on demand using all transactions designed to fail.
- We can keep adding more hardware resources: RAM, CPUs, etc but after some period of time we would need more to keep up with the performance and deliver this report on demand.
- No one wants to wait just to see what is going on now.
- Complains from all departments start to accumulate saying: it is faster to go to the warehouse and manually count it, rather then wait for system to show the data
- This is where batch layer or data warehouse changes the game

Stock Status Report Example

Transactions History

Date	Part Num	Qty
01/05/2016	Part A	+1,500
02/09/2016	Part A	-350
02/09/2016	Part B	+25,000
02/10/2016	Part C	+15,000
02/11/2016	Part B	-5,000
03/01/2016	Part C	+12,000
...		

1. Batch: New transaction added for Part A“as-is”
2. Batch: Part A quantity updated by subtracting 1,000



Stock Status

Part Number	Current Qty
Part A	13,150
Part B	13,500
Part C	16,000

Stock Status Report Example

- Stock Status report (query) now runs from Stock Status table (view) and does not require millions of calculations to be performed each time report is requested
- Stock Status and Transaction History tables (views) are updated instantly with each new data coming in
- New record is always added to the Transaction History
- New record is added to the Stock Status table if there is no matching Part Number exists
- Existing record is updated in the Stock Status table for the part that matches the transaction part

Stock Status Report Example

- We've covered a task of having Todays stock status data available instantly instead of waiting for processing of millions transactions recorder over the years
- How about Yesterday Stok Status?
- Should we still wait all this time and process all recorded transactions to see what was the status as of Yesterday?
- Well, yes, we need to calculate the "in" and "out" quantities to show accurate results.
- We also might to make a decision to have a Stock Status Table for each day. So in 5 years we will have at least 1825 tables (views)

Stock Status Report Example

- We still can find a compromise between storing “everything” and calculating on demand
- One of the most popular solutions is instead calculating parts quantities “forward” from the beginning of the transactions history, is to calculate them “backward”.
- We start from the current stock status table (view) and “reverse” all transactions happened between Today and Yesterday
- The stock status as of beginning of Yesterday will only include transaction for two days: Today and Yesterday, which is significantly smaller number then number of transactions for the last 5 years

Stock Status Report Example

- There are hybrid solution can be implemented as well
- For example creating stock status table for the beginning and for the end of each year.
- We will have only 10 tables for 5 years period
- When Stock Status report request, the Stock Status table (view) representing the Year of the request will be used
- If the date is closer to the beginning of the year, we use the table for the beginning of this year and “add” all transaction between first date of the year and requested stock status report date
- If the date is closer to the end of the year, we use the table for the end of the year and “reverse” transaction within this range

Stock Exchange Example

- Stock market trading is a fountain of information, with millions of shares and billions of dollars changing hands on a daily basis.
- With so many trades taking place, stock prices are historically recorded daily as an opening price, high price, low price, and closing price.
- But those bits of data often don't provide the big picture and can potentially skew your perception of what happened.
- For instance, lets look at data that show the price data for Google, Apple, and Amazon stocks on a day when Google announced new products targeted at their competitors.

Stock Exchange Example

- This data suggests that Amazon may not have been affected by Google's announcement, as its stock price moved only slightly.

- The data also suggest that the announcement had either no effect on Apple, or a positive effect.

Company	Symbol	Previous	Open	High	Low	Close	Net
Google	GOOG	564.68	567.70	573.99	566.02	569.30	+4.62
Apple	AAPL	572.02	575.00	576.74	571.92	574.50	+2.48
Amazon	AMZN	225.61	225.01	227.50	223.30	225.62	+0.01

- Financial reporting promotes daily net change in closing prices.

- What conclusion would you draw about the impact of Googke's announcements?

Stock Exchange Example

But if you have access to data stored at a finer time granularity, you can get a clearer picture of the events on that day and probe further into potential cause and effect relationships.

Stock Exchange Example

1. Apple held steady throughout the day
2. Amazon's stock dipped in late day trading
3. Google's stock price had a slight boost on the day of the announcement

- The minute-by-minute relative changes in the stock prices of all three companies suggests that both Amazon and Apple were indeed affected by the announcement, Amazon more so than Apple.

- Additional data can suggest new ideas you may not have considered when examining the original daily stock price summary.

Stock Exchange Example

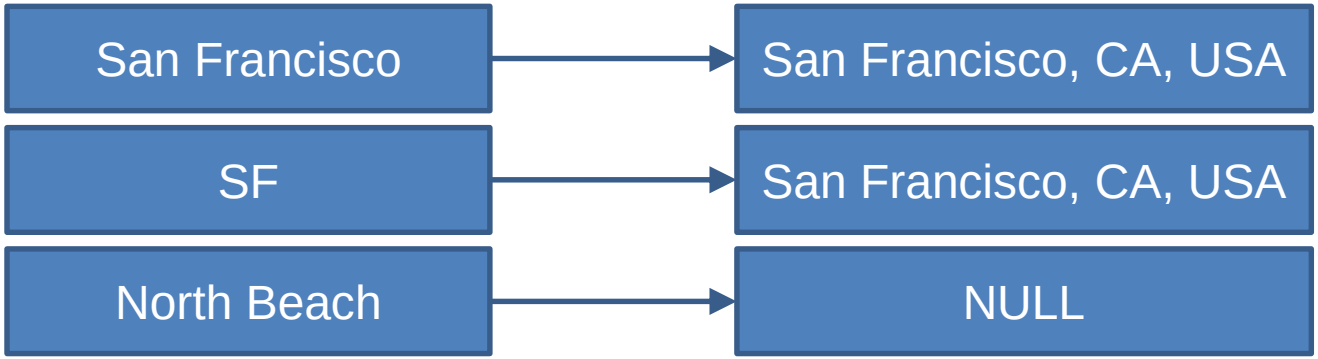
- Storing raw data is hugely valuable because you rarely know in advance all the questions you want answered.
- By keeping the rawest data possible, you maximize your ability to obtain new insights, whereas summarizing, overwriting, or deleting information limits what your data can tell you.
- Big Data technologies are designed to manage petabytes and exabytes of data.
- Specifically, they manage the storage of your data in a distributed, scalable manner while supporting the ability to directly query the data.

The Data is raw

- A data system answers questions about information you've acquired in the past.
- When designing your Big Data system, you want to be able to answer as many questions as possible.
- The master data set is more valuable then views because view can be regenerated from the master data set data but not the other way around.
- We call this property *rawness* of data
- Although the concept is straightforward, it's not always clear what information you should store as your raw data.

Unstructured vs Normalized

- The unstructured data is rawer than normalized data.
- When deciding what raw data to store, a common hazy area is the line between parsing and semantic normalization.
- Semantic normalization is the process of reshaping freeform information into a structured form of data.



- Semantic normalization of unstructured location responses to city, state, and country.

Unstructured vs Normalized

- A simple algorithm will normalize “North Beach” to NULL if it doesn’t recognize it as a San Francisco neighborhood.
- If you come across a form of data such as an unstructured location string, should you store the unstructured string or the semantically normalized form?
- If you store the unstructured string, you can renormalize that data at a later time when you have improved your algorithms.
- If your algorithm for extracting the data is simple and accurate you should store the results of that algorithm.
- If the algorithm is subject to change, due to improvements or broadening the requirements, store the unstructured form of the data.

Unstructured vs Normalized

- More information does not necessary mean rawer data
- Let's say that Tom is a blogger, and he wants to add his posts to his FaceSpace profile.
- What exactly should you store once Tom provides the URL of his blog?
- Storing the pure text of the blog entries is certainly a possibility. But any phrases in italics, boldface, or large font were deliberately emphasized by Tom and could prove useful in text analysis.
- Store the full HTML of Tom's blog as your data will require more space: color schemes, stylesheets, JavaScripts, etc.

Data is immutable

- Immutable data may seem like a strange concept if you're well versed in relational databases.
- In the relational database world *update* is one of the fundamental operations.
- For immutability you don't update or delete data, you only add more.
- By using an immutable schema for Big Data systems, you gain two vital advantages:
  - Human-fault tolerance
  - Simplicity

Data is immutable

Human-fault tolerance

- Human-fault tolerance is an essential property of data systems.
- People will make mistakes, and you must limit the impact of such mistakes and have mechanisms for recovering from them.
- With a mutable data model, a mistake can cause data to be lost, because values are actually overridden in the database.
- With an immutable data model, *no data can be lost*.
- If “bad” data is written - earlier (good) data units still exist.
- Fixing the data system is just a matter of deleting the bad data units and re-computing the views built from the master dataset.

Data is immutable

Simplicity

- Mutable data models imply that the data must be indexed in some way so that specific data objects can be retrieved and updated.
- In contrast, with an immutable data model you only need the ability to append new data units to the master dataset.
- This doesn't require an index for your data, which is a huge simplification.
- Storing a master dataset is as simple as using flat files.

Data is immutable

- A mutable schema for Social Network user information.
- The advantages of keeping your data immutable become evident when comparing with a mutable schema.

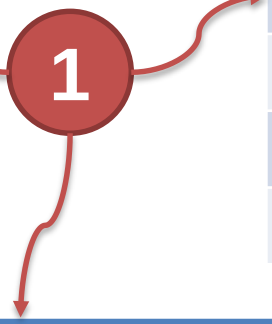
id	Name	Age	Gender	Employer	Location
1	Alice	25	Female	Apple	Atlanta, GA
2	Bob	36	Male	SAS	Chicago, IL
3	Tom	28	Male	Google	San Francisco, CA
4	Charlie	25	Male	Microsoft	Washington, DC
...	...	...	...	...	...

- When details change - say, Tom moves to Los Angeles - previous values are overwritten and lost.

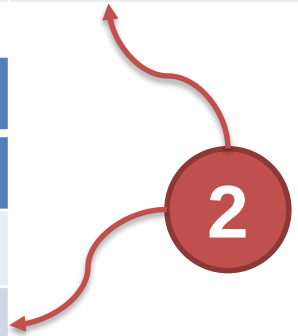
Data is immutable

Name Data		
id	Name	Timestamp
1	Alice	09/04/2018 07:30:01
2	Bob	10/11/2017 12:04:13
3	Tom	05/08/2018 18:01:15
4	Charlie	09/12/2016 01:12:01
...	...	...

Age Data		
id	Age	Timestamp
1	25	09/04/2018 07:30:01
2	36	10/11/2017 12:04:13
3	28	05/08/2018 18:01:15
4	25	09/12/2016 01:12:01
...	...	...



Location Data		
id	Name	Timestamp
1	Atlanta, GA	09/04/2018 07:30:01
2	Chicago, IL	10/11/2017 12:04:13
3	San Francisco, CA	05/08/2018 18:01:15
4	Washington, DC	09/12/2016 01:12:01
...	...	...



Data is immutable

- An immutable schema for Social Network user information.

1. Each field of user information is kept separately.
2. Each record is timestamped when it is stored.

- Each field is tracked in a separate table, and each row has a timestamp for when it's known to be true.

- Rather than storing a current snapshot of the world, as done by the mutable schema, immutable one creates a separate record every time a user's information evolves.

Data is immutable

- Accomplishing this requires two changes.
  - Each field of user information tracked in a separate table.
  - Each unit of data is tied to a moment in time when the information is known to be true.
- When Tom moves to Los Angeles on June 17, 2018, new record added to the location table, timestamped by when he changed his profile
- Now there are two location records for Tom (user ID #3), and because the data units are tied to particular times, they can both be true.
- Tom's *current location* involves a simple query on the data: look at all the locations, and pick the one with the most recent timestamp.

Data is immutable

- By keeping each field in a separate table, you only record the information that changed.
- This requires less space for storage and guarantees that each record is new information and is not simply carried over from the last record.

Location Data		
id	**Name**	**Timestamp**
1	Atlanta, GA	09/04/2018 07:30:01
2	Chicago, IL	10/11/2017 12:04:13
3	San Francisco, CA	05/08/2018 18:01:15
4	Washington, DC	09/12/2016 01:12:01
3	Los Angeles	06/17/2018 15:30:05
...	...	...

Initial Data about Tom location

Additional record about new Tom location

Data is eternally true

- The key consequence of immutability is that each piece of data is true in perpetuity.
- Each piece of data, once true, must always be true.
- Immutability wouldn't make sense without this property, this is why each piece of data tagged with a timestamp as a practical way to make data eternally true.
- In general, your master dataset consistently grows by adding new immutable and eternally true pieces of data.

Data is eternally true

- There are some special cases, though, in which data deleted and these cases are not incompatible with data being eternally true:

- *Garbage collection:*
  - Implemented to delete all data units that have low value
  - Used to implement data retention policies that control the growth of the master dataset. (keep only one location per person per year)

- *Regulations:*
  - Government regulations may require to purge data from databases under certain conditions.

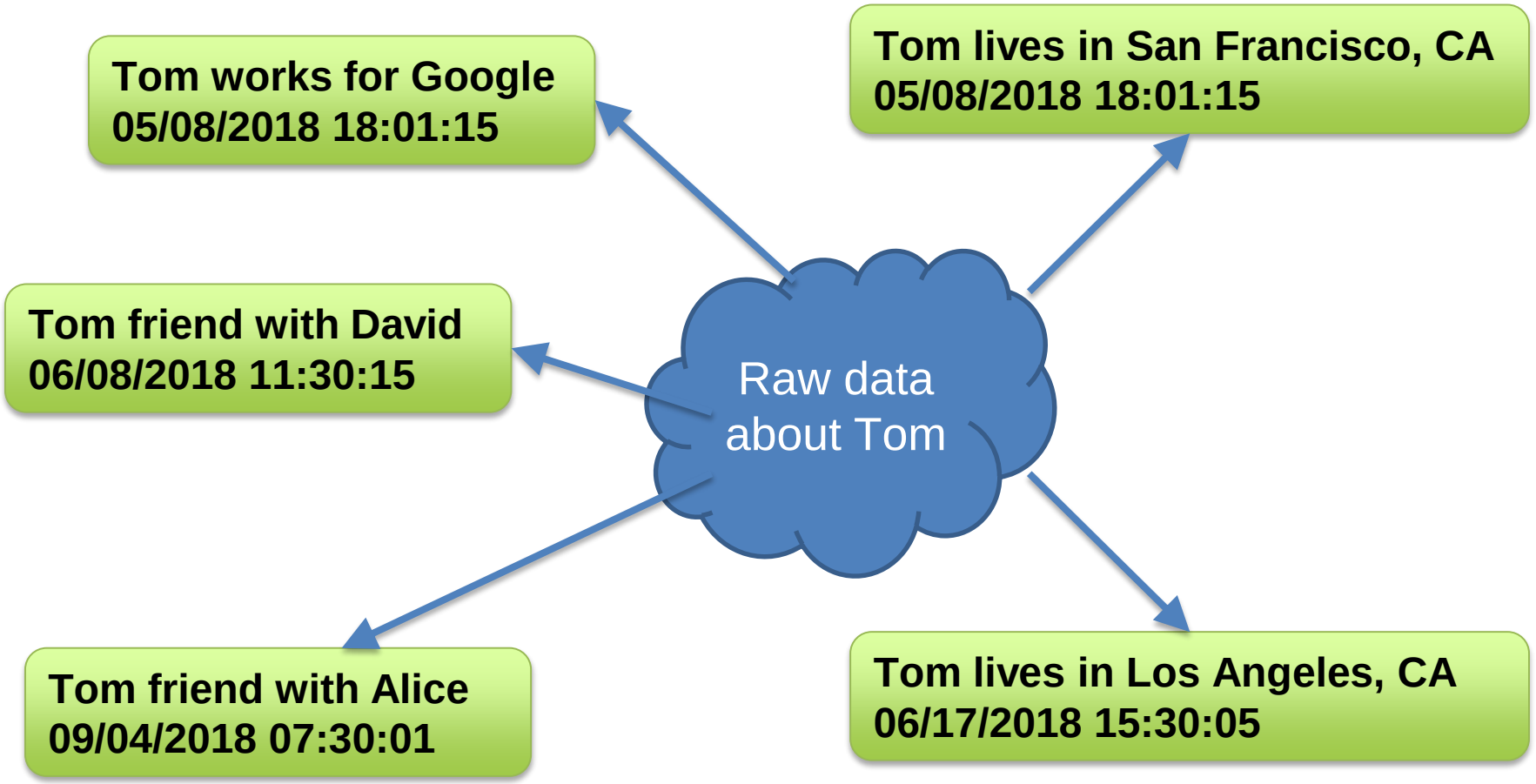
- In both cases, deleting the data is not a statement about the truthfulness of the data but instead, it's a statement about the value of the data.

The fact-based model

- Data is the set of information that can't be derived from anything else, but there are many ways you could choose to represent it within the master dataset.
- Besides traditional relational tables, structured XML and semi structured JSON documents there are other possibilities for storing data.
- In the fact-based model, you deconstruct the data into fundamental units called (unsurprisingly) *facts*.
- Facts are atomic because they can't be subdivided further into meaningful components.

The fact-based model

- Collective data, such as Tom's friend list in the figure, are represented as multiple, independent facts.



The fact-based model

- *identifiability*- facts should be associated with a uniquely identifiable piece of data.
- If facts comes for the same time we might encounter the same exact data record
- If we encounter two identical pageview records, there's no way to tell whether they refer to two distinct events or if a duplicate entry was accidentally introduced into our dataset.
- To distinguish different pageviews, you can add a *nonce* to your schema - a 64-bit number randomly generated for each pageview

The fact-based model

- The addition of the nonce makes it possible to distinguish pageview events from each other, and if two pageview data units are identical (all fields, including the nonce), you know they refer to the exact same event.

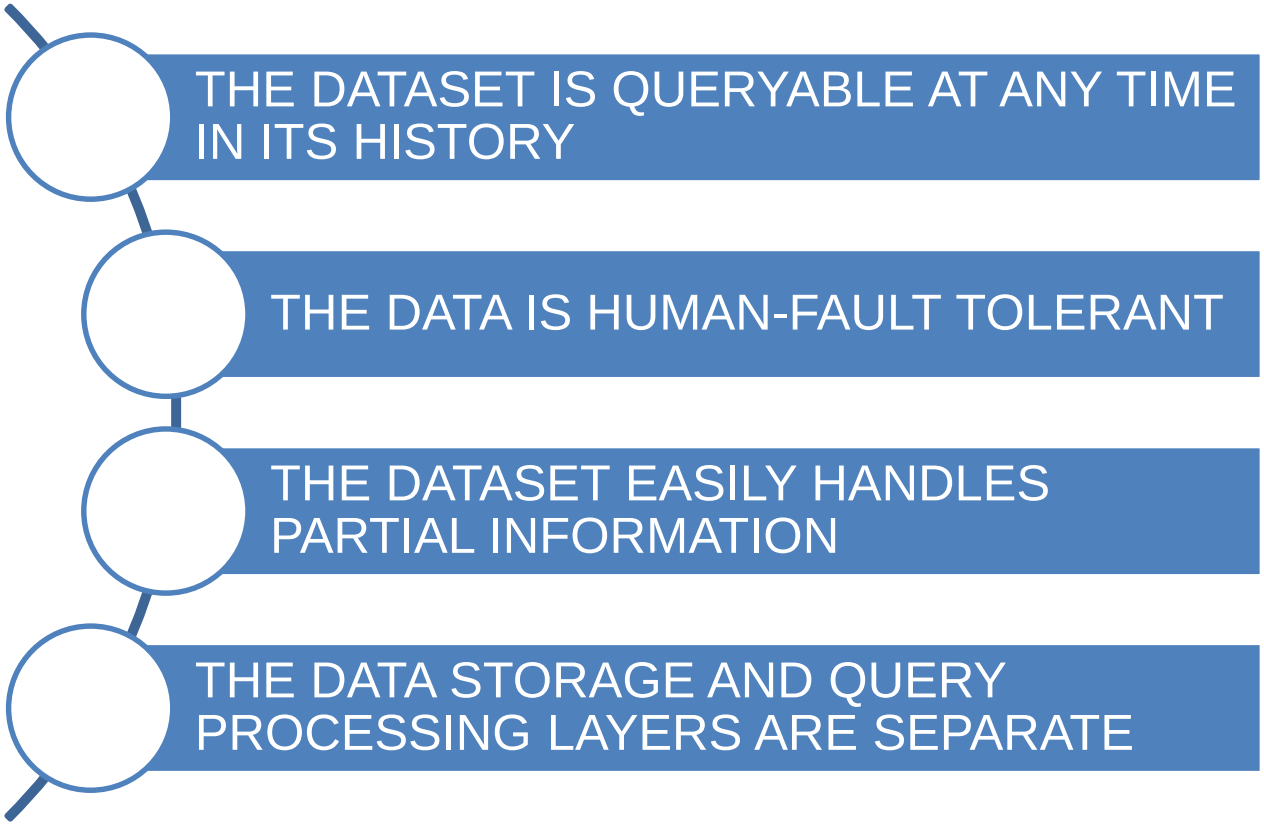
- Making facts identifiable means that we can write the same fact to the master dataset multiple times without changing the semantics of the master dataset.

- Queries can filter out the duplicate facts when doing their computations.

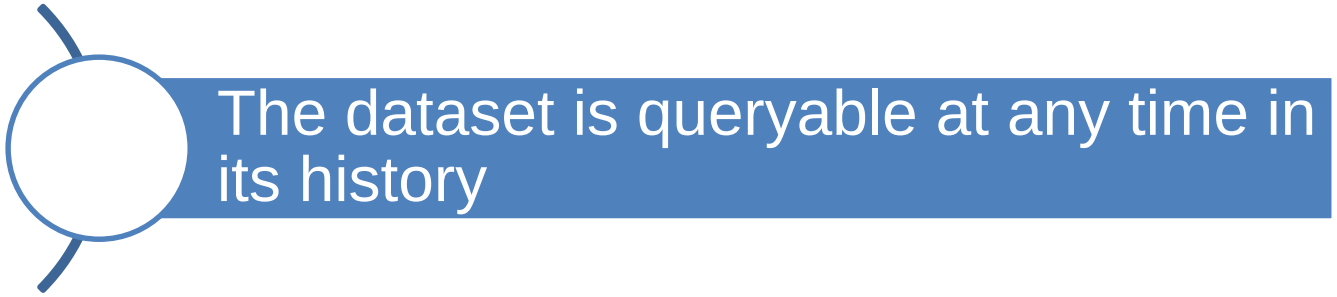
- Having distinguishable facts makes implementing the rest of the Lambda Architecture much easier.

The fact-based model

Benefits of the fact-based model

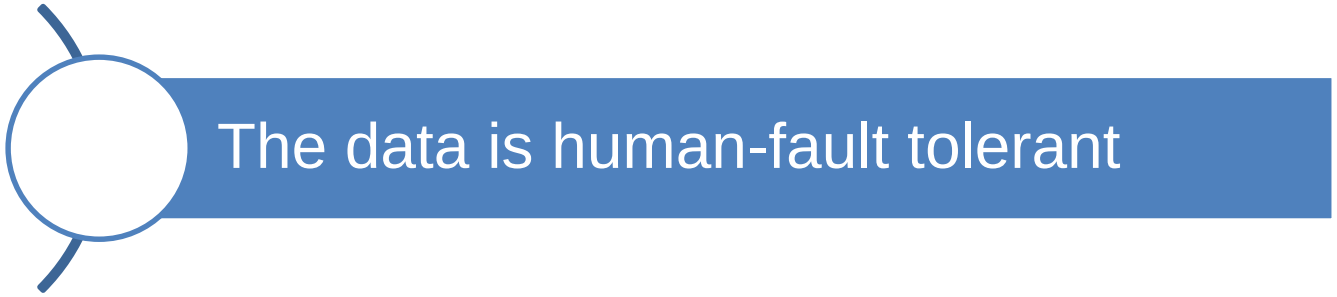


The fact-based model



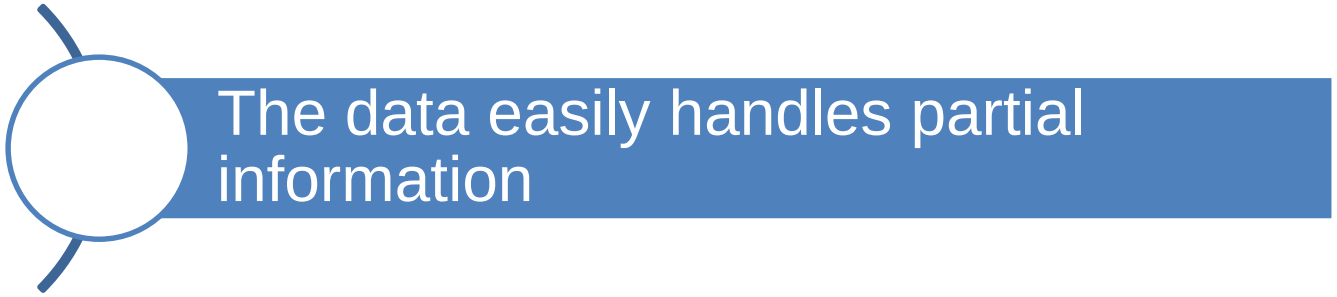
- Instead of storing only the current state of the world, we have the ability to query the data for any time covered by the dataset.
- This is a direct consequence of facts being timestamped and immutable.
- “Updates” and “Deletes” are performed by adding new facts with more recent timestamps, but because no data is actually removed, we can reconstruct the state of the world at the time specified by the query.

The fact-based model



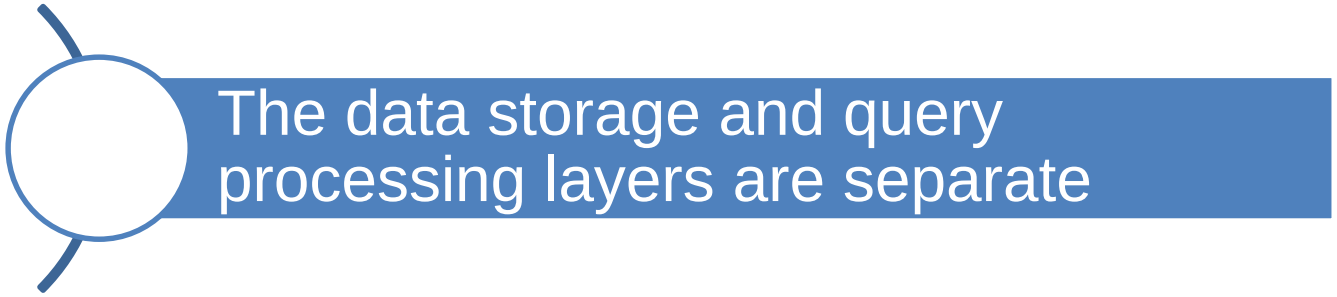
- Human-fault tolerance is achieved by simply deleting any erroneous facts.
- Suppose you mistakenly stored that Tom moved from San Francisco to Los Angeles.
- By removing the Los Angeles fact, Tom's location is automatically "reset" because the San Francisco fact becomes the most recent information.

The fact-based model



- Storing one fact per record makes it easy to handle partial information about an entity without introducing NULL values into your dataset.
- Suppose Tom provided his age and gender but not his location or profession.
- Dataset would only have facts for the known information - any “absent” fact would be logically equivalent to NULL.
- Additional information that Tom provides at a later time would naturally be introduced via new facts.

The fact-based model



- Another key advantage of the fact-based model that is in part due to the structure of the Lambda Architecture itself.
- By storing the information at both the batch and serving layers, we have the benefit of keeping your data in both normalized and de-normalized forms and reaping the benefits of both.
- Data normalization is completely unrelated to the semantic normalization term, it refers to storing data in a structured manner to minimize redundancy and promote consistency.

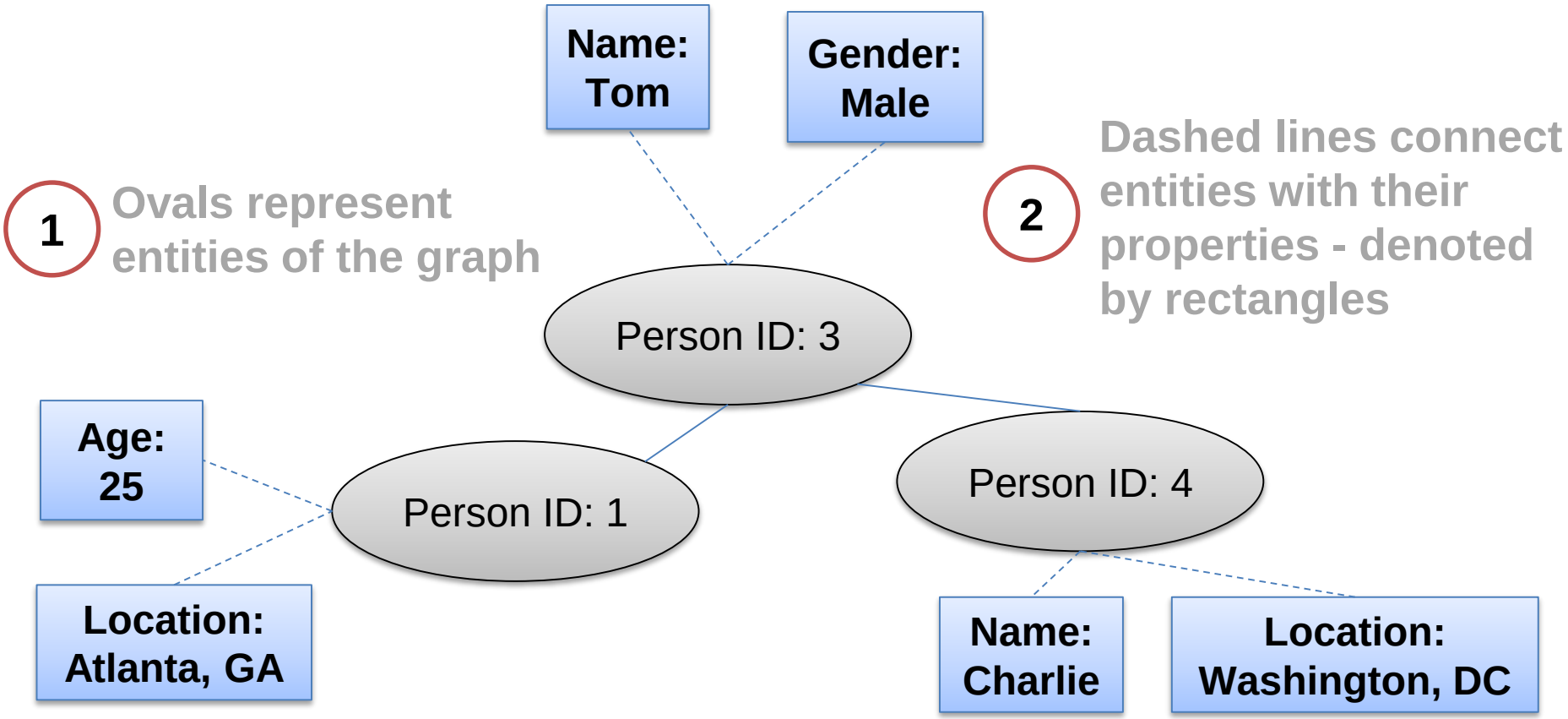
The fact-based model

- The mutually exclusive choice between normalized and de-normalized schemas is necessary because, for relational databases, queries are performed directly on the data at the storage level.
- The objectives of query processing and data storage are cleanly separated in the Lambda Architecture.
- In the Lambda Architecture, the master dataset is fully normalized and no data is stored redundantly.
- Updates are easily handled because adding a new fact with a current timestamp “overrides” any previous related facts.

Graph schemas

- Each fact within a fact-based model captures a single piece of information.
- The facts alone don’t convey the structure behind the data.
- There is no description of the types of facts contained in the dataset, nor any explanation of the relationships between them.
- Graph schemas capture the structure of a dataset stored using the fact-based model.
- Graph schema representing the relationships between the facts.
- Graph provides a useful visualization existing facts and relationship between them.

Graph schemas



Graph schemas

- There are three core components of a graph schema:
  - *Nodes*
  - *Edges*
  - *Properties*

- *Nodes* are the entities in the system.

- *Edges* are relationships between nodes.

- *Properties* are information about entities.

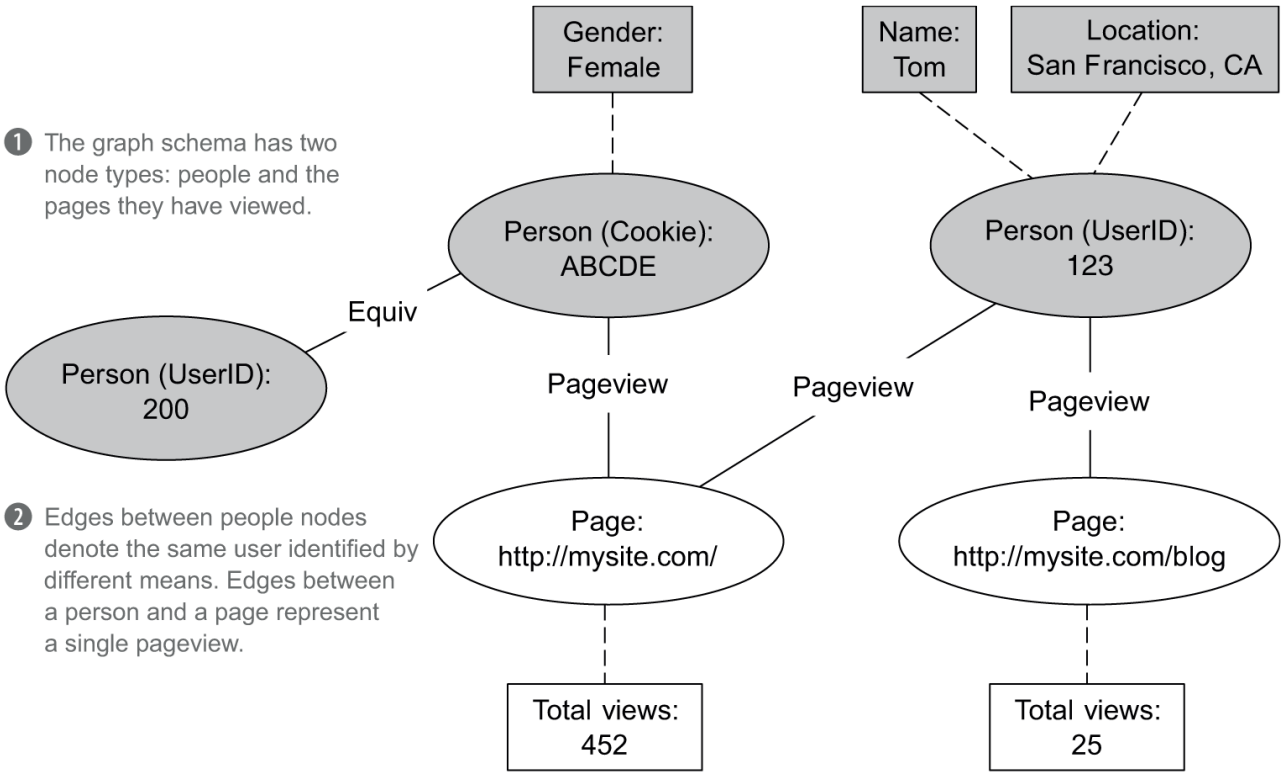
- Edges are strictly between nodes. Even though properties and nodes are visually connected, these lines are not edges. They are present only to help illustrate the association between entity and its information.

Big Data Model Illustration

- We'll implement the SuperWebAnalytics.com data model using Apache Thrift, a serialization framework.
- Even in a task as straightforward as writing a schema, there is friction between the idealized theory and what you can achieve in practice.
- Serialization frameworks are an easy approach to making an enforceable schema.
- Serialization frameworks generate code for whatever languages you wish to use for reading, writing, and validating objects that match your schema.
- Serialization frameworks are limited when it comes to achieving a fully rigorous schema.

Big Data Model Illustration

- The graph schema for SuperWebAnalytics.com.
- There are two node types: people and pages.
- People nodes and their properties are slightly shaded to distinguish the two.



Big Data Model Illustration

- Apache Thrift is a tool that can be used to define statically typed, enforceable schemas.
- It provides an interface definition language to describe the schema in terms of generic data types, and this description can later be used to automatically generate the actual implementation in multiple programming languages.
- Thrift was initially developed at Facebook for building cross-language services.
- It can be used for many purposes, but we'll limit our use to its usage as a serialization framework.

Big Data Model Illustration

- The workhorses of Thrift are the *struct* and *union* type definitions.
- They're composed of other fields:
  - Primitive data types (strings, integers, longs, and doubles)
  - Collections of other types (lists, maps, and sets)
  - Other structs and unions
- Unions are useful for representing nodes
- Structs are natural representations of edges
- Properties use a combination of both.

Big Data Model Illustration

- For our SuperWebAnalytics.com user **nodes**, an individual is identified either by a user ID or a browser cookie, but not both.
- This pattern is common for nodes, and it matches exactly with a union data type - a single value that may have any of several representations.
- In Thrift, unions are defined by listing all possible representations.
- The following code defines the SuperWebAnalytics.com nodes using Thrift unions:

```
union PersonID {
  1: string cookie;
  2: i64 user_id;
}
```

```
union PageID {
  1: string url;
}
```

Unions can also be used for nodes with a single representation. Unions allow the schema to evolve as the data evolves.

Big Data Model Illustration

- Each **edge** can be represented as a struct containing two nodes.
- The name of an edge struct indicates the relationship it represents, and the fields in the edge struct contain the entities involved in the relationship.
- The schema definition is very simple:

```
union EquivEdge {
  1: required PersonID id1;
  2: required PersonID id2;
}
```

```
union PageViewEdge {
  1: required PersonID person;
  2: required PageID page;
  3: required i64 nonce;
}
```

- The fields of a Thrift struct can be denoted as *required* or *optional*.
- If a field is defined as required, then a value for that field must be provided, or else Thrift will give an error upon serialization or deserialization.

Big Data Model Illustration

- A **property** contains a node and a value for the property.
- The value can be one of many types, so it's best represented using a union structure.
- Defining the schema for **page properties**:

```
union PagePropertyValue {
  1: i32 page_views;
}
```

```
union PageProperty {
  1: required PageID id;
  2: required PagePropertyValue property;
}
```

Big Data Model Illustration

- Defining the schema for **people and location** properties. Location property is more complex and requires another struct to be defined:

```
union Location {
  1: optional string city;
  2: optional string state;
  3: optional string country;
}
```

```
enum GenderType {
  1: MALE = 1,
  2: FEMALE = 2
}
```

```
union PersonPropertyValue {
  1: string full_name;
  2: GenderType gender;
  3: Location location;
}
```

```
struct PersonProperty {
  1: required PersonID id;
  2: required PersonPropertyValue
                property;
}
```

Big Data Model Illustration

- The edges and properties are defined as separate types.
- We would want to store all of the data together to provide a single interface to access the information.
- It also makes data easier to manage if it's stored in a single dataset.
- This is accomplished by wrapping every property and edge type into a *DataUnit* union
- Each DataUnit is paired with its metadata, which is kept in a *Pedigree* struct.
- The pedigree contains the timestamp for the information, but could also potentially contain debugging information or the source of the data.

Big Data Model Illustration

- The final Data struct corresponds to a fact from the fact-based model:

```
union Location {
  1: optional string city;
  2: optional string state;
  3: optional string country;
}

struct Pedigree {
  1: required i32 true_as_of_secs;
}

struct Data {
  1: required Pedigree pedigree;
  2: required DataUnit dataunit;
}
```

Big Data Model Illustration

- Thrift is designed so that schemas can evolve over time.
- This is a crucial property, because as business requirements might change new kinds of data can be added and we would want this process be as effortlessly as possible
- The key to evolving Thrift schemas is the numeric identifiers associated with each field.
- Those IDs are used to identify fields in their serialized form.

Big Data Model Illustration

- To change the schema but still be backward compatible with existing data, the following rules must be follwed:
  - Fields may be renamed. This is because the serialized form of an object uses the field IDs, not the names, to identify fields.
  - A field may be removed, but we must never reuse that field ID. When deserializing existing data, Thrift will ignore all fields with field IDs not included in the schema.
  - Only optional fields can be added to existing structs. We can't add required fields because existing data won't have those fields and thus won't be deserializable. *(Note that this doesn't apply to unions, because unions have no notion of required and optional fields.)*

Big Data Model Illustration

- Extending the SuperWebAnalytics.com schema
- For example: if we want to change the SuperWebAnalytics.com schema to store a person's age and the links between web pages, the following changes to Thrift definition needs to be done (changes in bold font).

```
union PersonPropertyValue {
  1: string full_name;
  2: GenderType gender;
  3: Location location;
  4: i16 age;
}
```

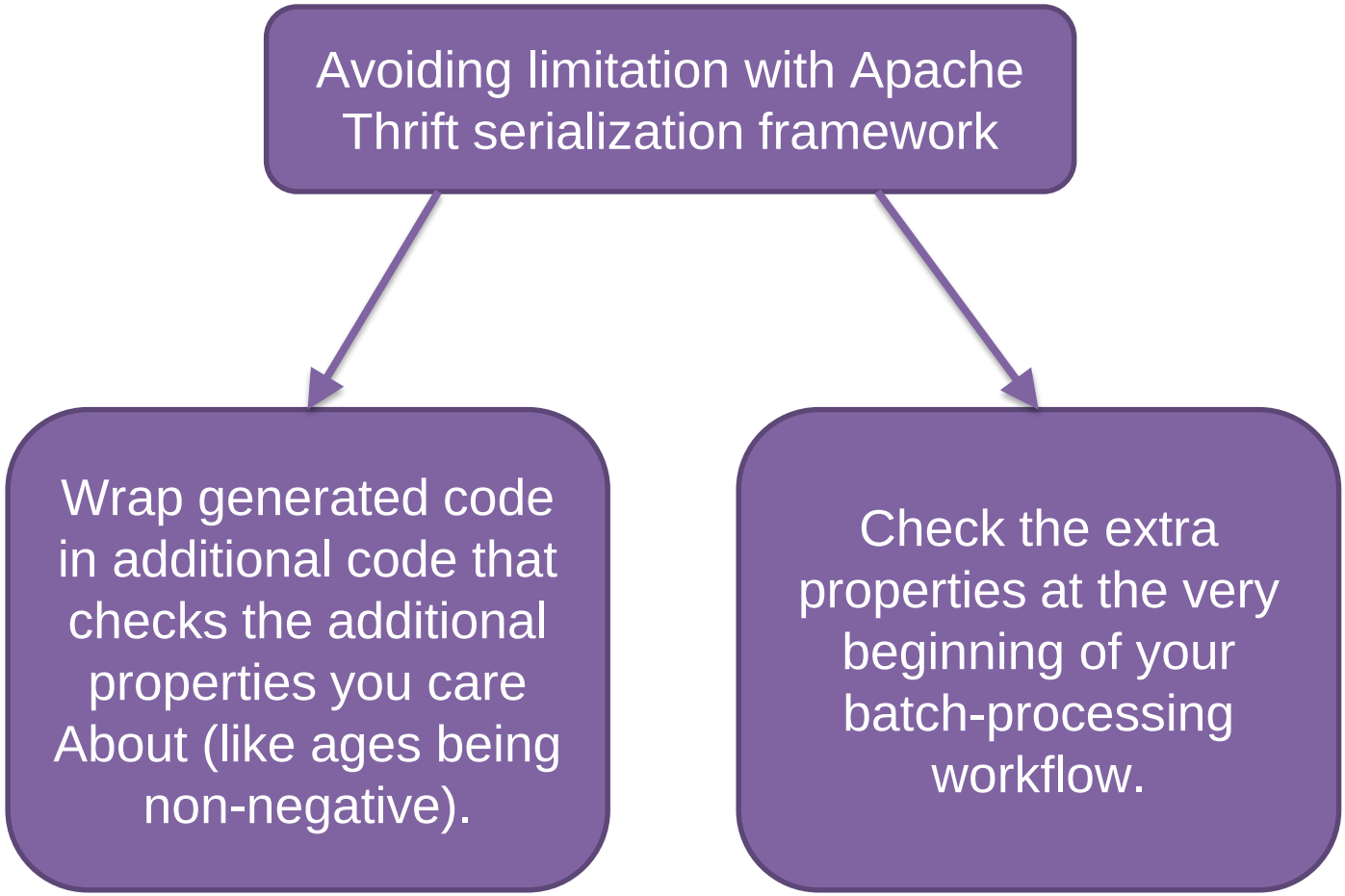
```
struct LinkedEdge {
  1: required PageID source;
  2: required PageID target;
}
```

```
union DataUnit {
  1: PersonProperty person_property;
  2: PageProperty page_property;
  3: EquivEdge equiv;
  4: PageViewEdge page_view;
  5: LinkedEdge page_link;
}
```

Big Data Model Illustration

- Limitations of serialization frameworks:
- Serialization frameworks only check that all required fields are present and are of the expected type.
- They're unable to check richer properties like "Ages should be nonnegative" or "true-as-of timestamps should not be in the future."
- Data not matching these properties would indicate a problem in your system, and you wouldn't want them written to your master dataset.
- The right way to think about a schema is as a function that takes in a piece of data and returns whether it's valid or not.

Big Data Model Illustration



Big Data Model Illustration

- For the most part, implementing the enforceable graph schema for SuperWebAnalytics.com was straightforward.
- The friction that appears when using a serialization framework for this purpose - namely, the inability to enforce every property you care about.
- The tooling will rarely capture your requirements perfectly, but it's important to know what would be possible with ideal tools.
- That way you're cognizant of the trade-offs you're making and can keep an eye out for better tools (or make your own).

Summary

In this session we've learned:

- What is Data Analytics and Data Science
- How Big Data technology is defined
- What is MapReduce
- Details of Lambda Architecture
- The key properties of data
- Advantages of the fact-based model for the master dataset

References

lambda-architecture.net

thrift.apache.org

Nathan Marz, James Warren

Big Data

Principles and Best Practices of Scalable Real-Time Data Systems

ALY6110

Data Management and Big Data

Q & A

Instructor: Valeriy Shevchenko
v.shevchenko@northeastern.edu