UNIVERSITY of WASHINGTON

Week 9: Additional Database Systems

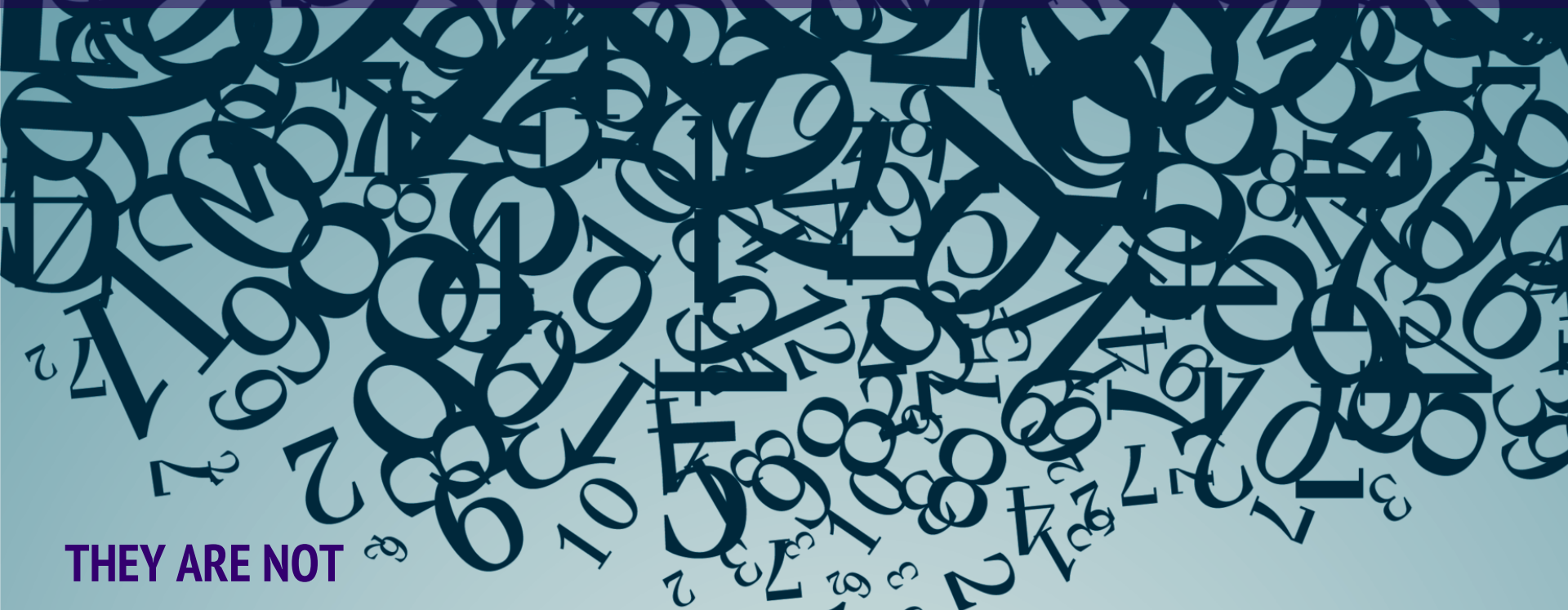
AGENDA

- Semi Structured Data
  - XML
  - JSON
- NoSQL
- Project Time

UNIVERSITY *of* WASHINGTON

Semi-Structured Data

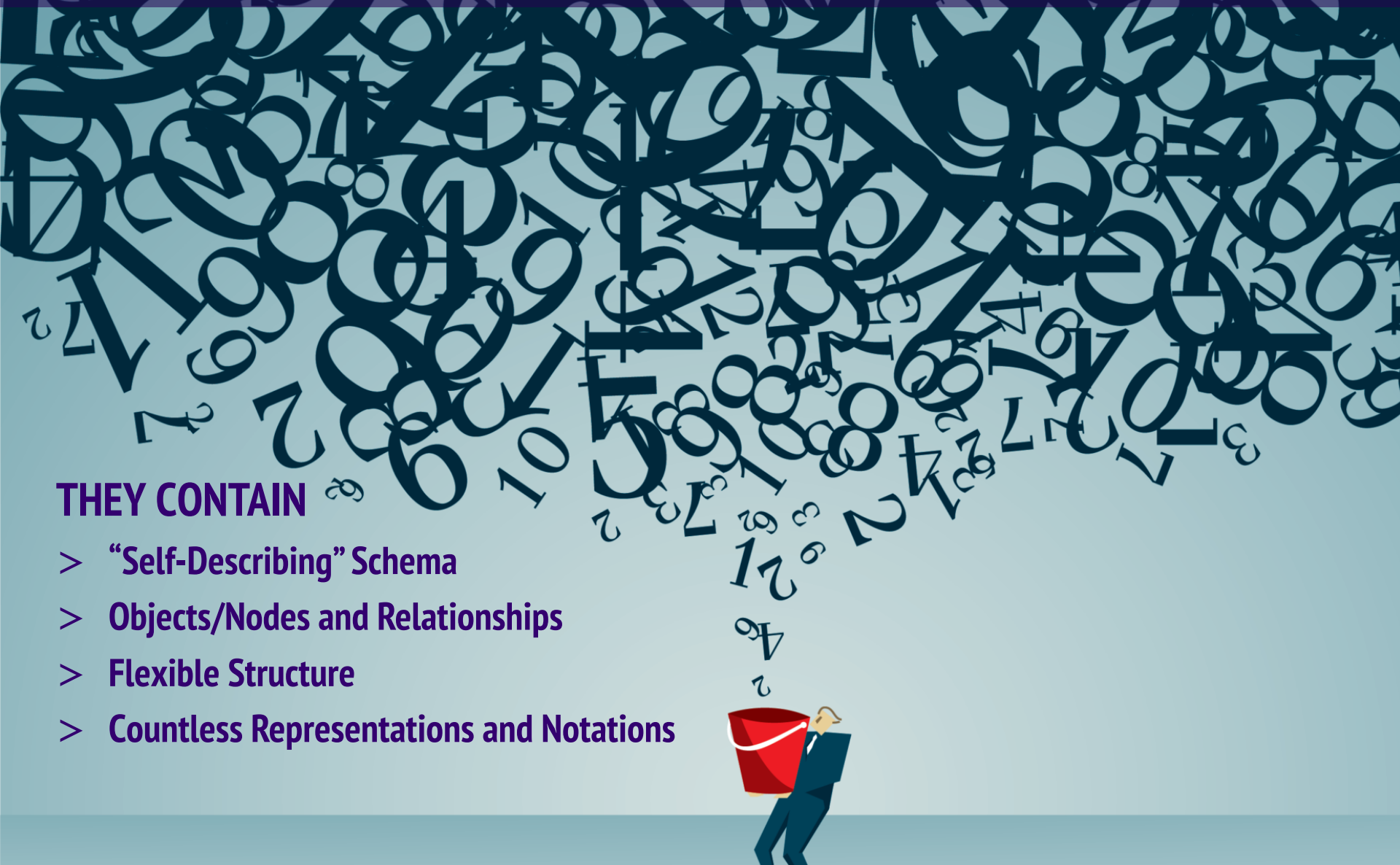
WHAT ARE SEMI-STRUCTURED DATA



THEY ARE NOT

- Random unintelligible bits
- RAW Image, Movie or Text files (though could _contain_ them)
- ”Unstructured” data
- Completely Schema-less

WHAT ARE SEMI-STRUCTURED DATA



THEY CONTAIN

- "Self-Describing" Schema
- Objects/Nodes and Relationships
- Flexible Structure
- Countless Representations and Notations

ADVANTAGES | DISADVANTAGES

ADVANTAGES

Fast and effective for integration

Flexible and 'easy' to implement

Can describe higher-complexity

DISADVANTAGES

Slow to query or modify

Offers little guarantees

Anomalous / inconsistent data

XML: Extensible Markup Language

- Born (on the shoulders SGML) in 1997
  - SGML: Standard Generalized Markup Language (1986)
- Allows for Schema or No Schema definition
- Hierarchical in nature but supports non-hierarchical structures
- If you are familiar with HTML, XML will “click”

JSON: JavaScript Object Notation

- Has nothing to do with Java
- Born from JavaScript applications but…
  - Is language independent
- Originally Specified in the 2000s

Typical XML Document Structure

```
<declaration>
<rootnode>
…
</rootnode>
```

```
<? xml version="1.0" encoding = "utf-8” ?>
<my_cool_root_node>
    <people>
        <person>Bill</person>
        <person>Donald</person>
        <person>George</person>
    </people>
</my_cool_root_node>
```

Typical XML Document Structure

<declaration>
<rootnode>
…
</rootnode>

```
<? xml version="1.0" encoding = "utf-8” ?>
<root>
  <element attribute="value" />
  <element>Data</element>

  <example name="hello" age=100>
    Here is some information!
  </example>
</root>
```

An Example of XML and JSON

```
<? xml version="1.0" encoding = "utf-8” ?>
<Users>
  <user id=1><name>Howdy</name></user>
  <user id=2><name>Doody</name></user>
  <user id=3 />
  <user id=4 job='Web Developer'></user>
  <user job='Web Developer' />
</Users>
```

```
{ 'Users':
    [ { 'id':1,'Name':'Howdy'}
    , { 'id':2,'Name':'Doody'}
    , { 'id':3 }
    , { 'id':4, 'job':'Web Developer' }
    , { 'job':'Web Developer' } ]
}
```

Document Type Definition (DTD)

```
<!DOCTYPE root-tag [
  <!ELEMENT name (components)>
    more elements
]>
```

```
<!DOCTYPE Users [
  <!ELEMENT Users (User+)>
  <!ELEMENT User (name?, job*)>
    <!ATTLIST User
        id ID #REQUIRED
       job CDATA #IMPLIED
     >
  <!ELEMENT name (#PCDATA)>
]
```

+ One or more
** Zero or More*
? Zero or One
No markup = One

Document Type Definition (DTD)

USERS.DTD

```
<!DOCTYPE Users [
  <!ELEMENT Users (User+)>
  <!ELEMENT User (name?, job*)>
    <!ATTLIST User
        id ID #REQUIRED
       job CDATA #IMPLIED
    >
  <!ELEMENT name (#PCDATA)>
]>
```

USER_DATA.XML

```
<? xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Users SYSTEM "users.dtd">
<Users>
  <user id=1><name>Howdy</name></user>
  <user id=2><name>Doody</name></user>
  <user id=3 />
  <user id=4 job='Web Developer'></user>
  <user job='Web Developer' />
</Users>
```

Document Type Definition (DTD)

USERS.DTD

```
<!DOCTYPE Users [
  <!ELEMENT Users (User+)>
  <!ELEMENT User (name?, job*)>
    <!ATTLIST User
        id ID #REQUIRED
       job CDATA #IMPLIED
    >
  <!ELEMENT name (#PCDATA)>
]>
```

USER_DATA.XML

```
<? xml version="1.0" encoding="utf-8" standalone="no" ?>
<!DOCTYPE Users SYSTEM "users.dtd">
<Users>
  <user id=1><name>Howdy</name></user>
  <user id=2><name>Doody</name></user>
  <user id=3 />
  <user id=4 job='Web Developer'></user>
  <user job='Web Developer' />
</Users>
```

Requires ID.
This XML document is not "Well Formed"

XML SCHEMA DEFINITION (XSD)

Positives:

> More expressive

> More features (datatypes!)

Negatives:

> Not as terse

> Not immediately compatible with SGML

XML SCHEMA DEFINITION (XSD)

```
<? xml version="1.0" encoding="utf-8" ?>

<xs:schema xlmns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="User">
    <xs:attribute name="id" type="xs:integer" use="required">
    <xs:attribute name="job" type="xs:string" use="optional">
    <xs:element name="name" type="xs:string" />
  </xs:complexType>
</xs:schema>
```

XML SCHEMA DEFINITION (XSD)

```
<? xml version="1.0" encoding="utf-8" ?>
<xs:schema xlmns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="movieType">
    <xs:sequence>
      <xs:element name="Title" type="xs:string" />
      <xs:element name="Year" type="xs:integer" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name = "Movies">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Movie" type="movieType"
                    minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Practice

- Get into your project group.
- Write an example XSD for your database
- Write a small XML file for a few of your records.

UNIVERSITY *of* WASHINGTON

NOSQL

Before the What... Let's talk Why?

Before the What... Let's talk Why?

SCALE

WHY NOSQL

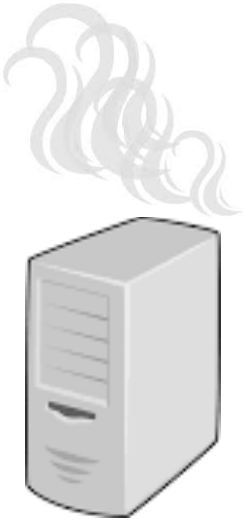
My really cool website.net.org

Thousands of Users

WHY NOSQL

SOLVING SCALE PROBLEMS:
WE HAVE TWO OPTIONS!

My really cool website.net.org



Millions of Users

WHY NOSQL

SCALE UP / VERTICALLY:

Get a bigger machine!

No need to change your system

Expensive, and does have limits

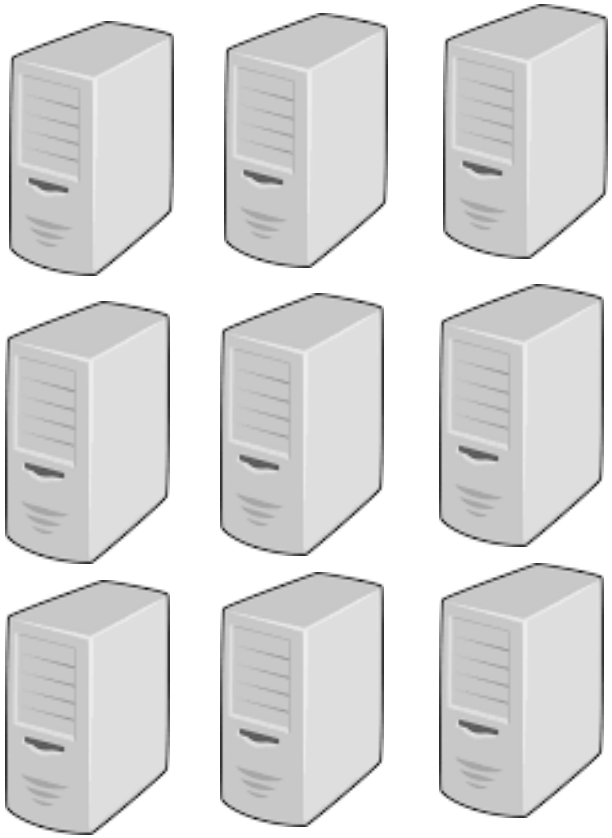
Millions of Users

WHY NOSQL

SCALE OUT / HORIZONTALLY

Cheap hardware

Much higher cap on performance



Millions of Users

WHAT IS NOSQL

- "Not SQL, Not Only SQL, Non-Relational SQL"
  - *It was really just a hashtag to advertise a meetup and it caught on.*

- No "Webster's" Definition
  - Kind of like answering "What is 'Big Data'?"

WHAT IS NOSQL

- Martin Fowler's Characteristics of NoSQL:
  - Non-Relational
  - Open-Source
  - Cluster-Friendly
  - 21st Century Web
  - "Schema-less"

Major Players

Google
BigTable

DynamoDB

amazon
web services

UNIVERSITY *of* WASHINGTON

Cassandra

What is Cassandra

- Began at Facebook
- Became Top Level Apache Project in 2010
  - Currently at Version 3.11 (Released Oct 10)
- Distributed, non-relational database
- Highly Fault Tolerant
- Extremely Fast Writes!
  - Used for collecting millions of transactions a second

Knowing Your Roots

DynamoDB

amazon web services

Google BigTable

Intelligent and Fault Tolerant Distribution

Rich Data Model

How data is written (super fast):

Data stored to a commit log, then stored in a Memtable

Once in Memtable, acknowledgement is sent to the client

Memtables flush to disk using sequential write (very fast IO)

Every write is timestamped to later processes know which is the old record.

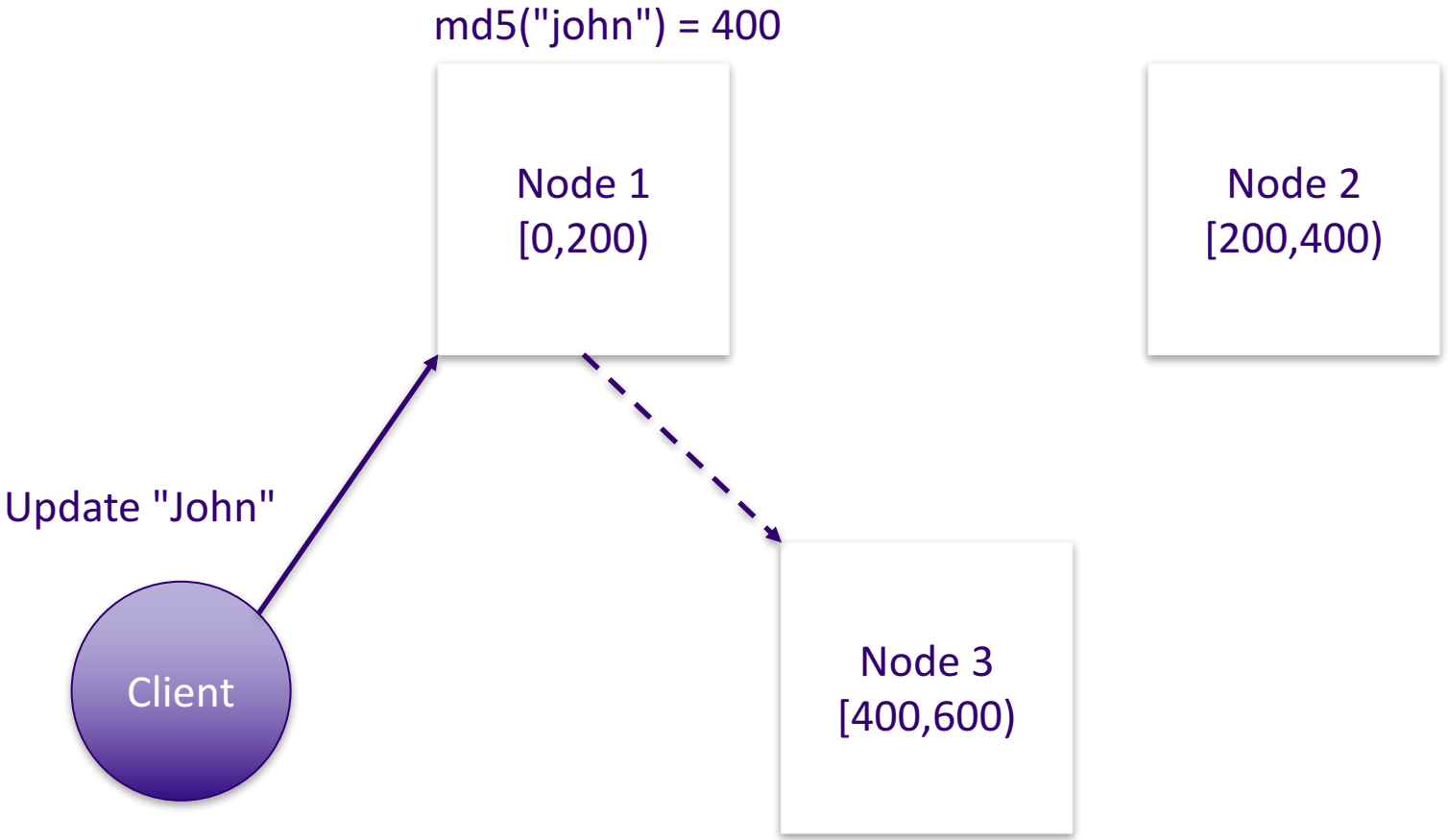
Compaction Process

Sequential writes introduce an inefficiency – duplication!

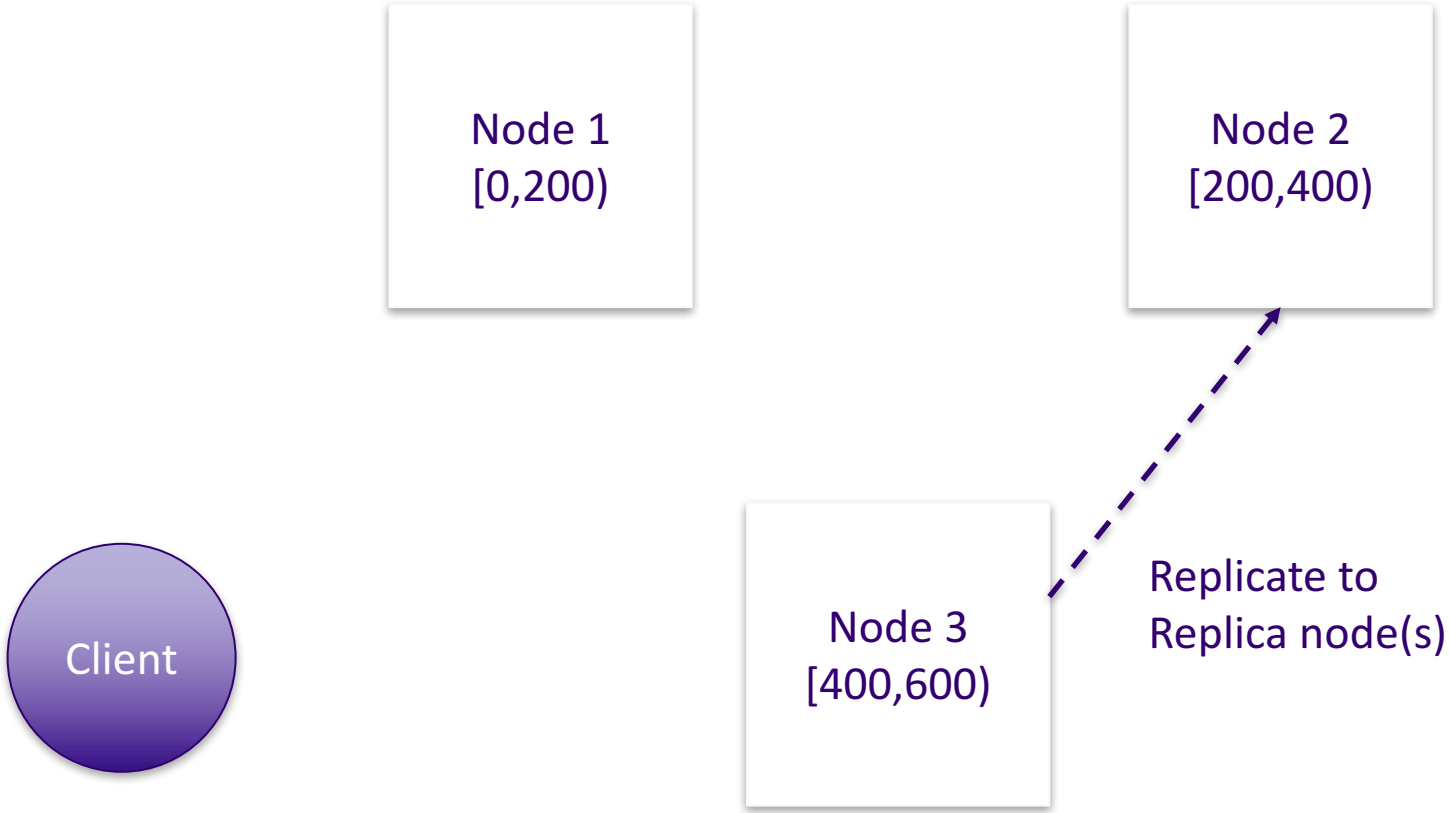
Compaction does sequential reads, and deletes duplicates.

Cassandra at the local server level is all about minimizing (or deferring) IO!

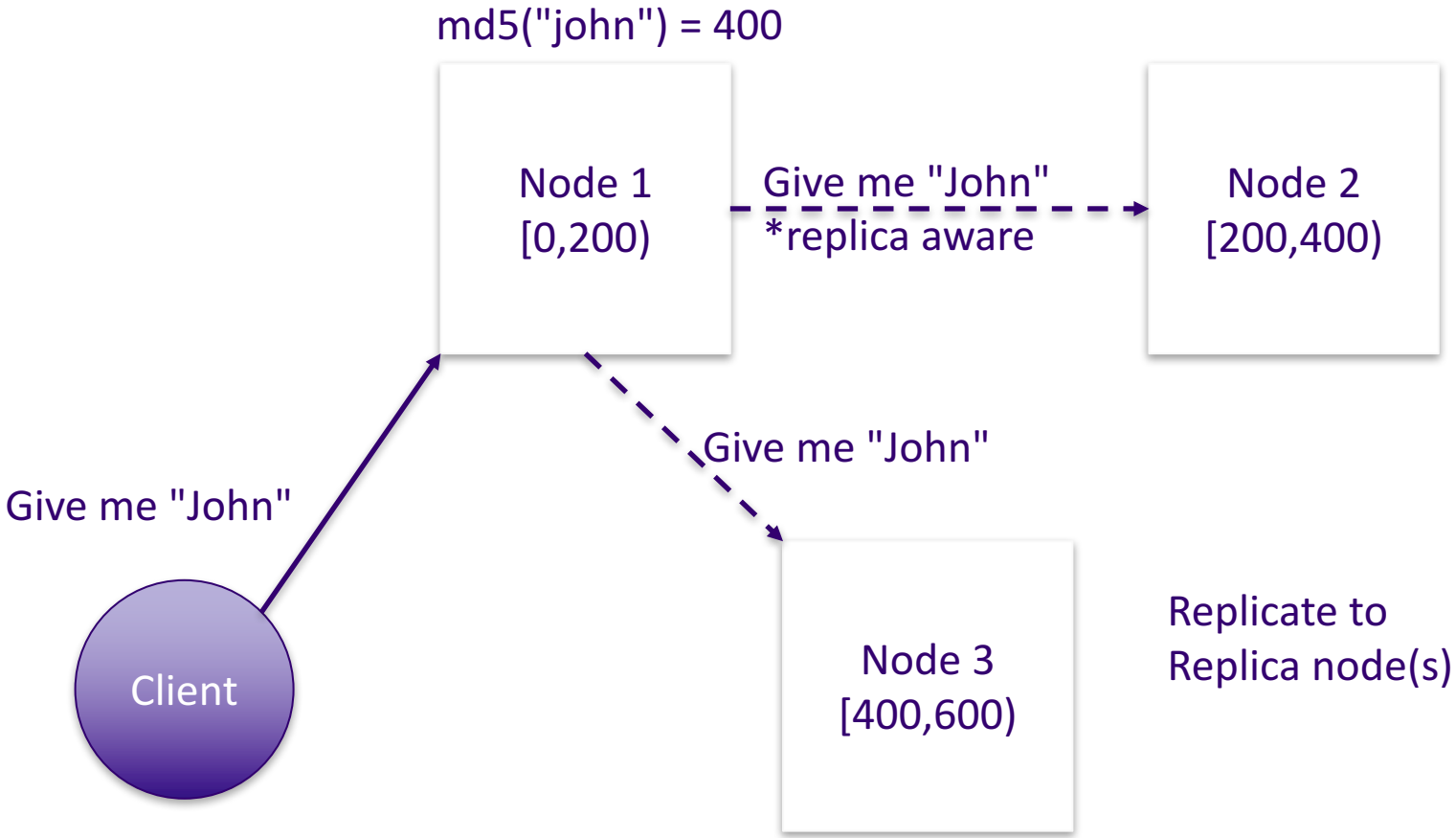
Distributed/Consistent Hashing



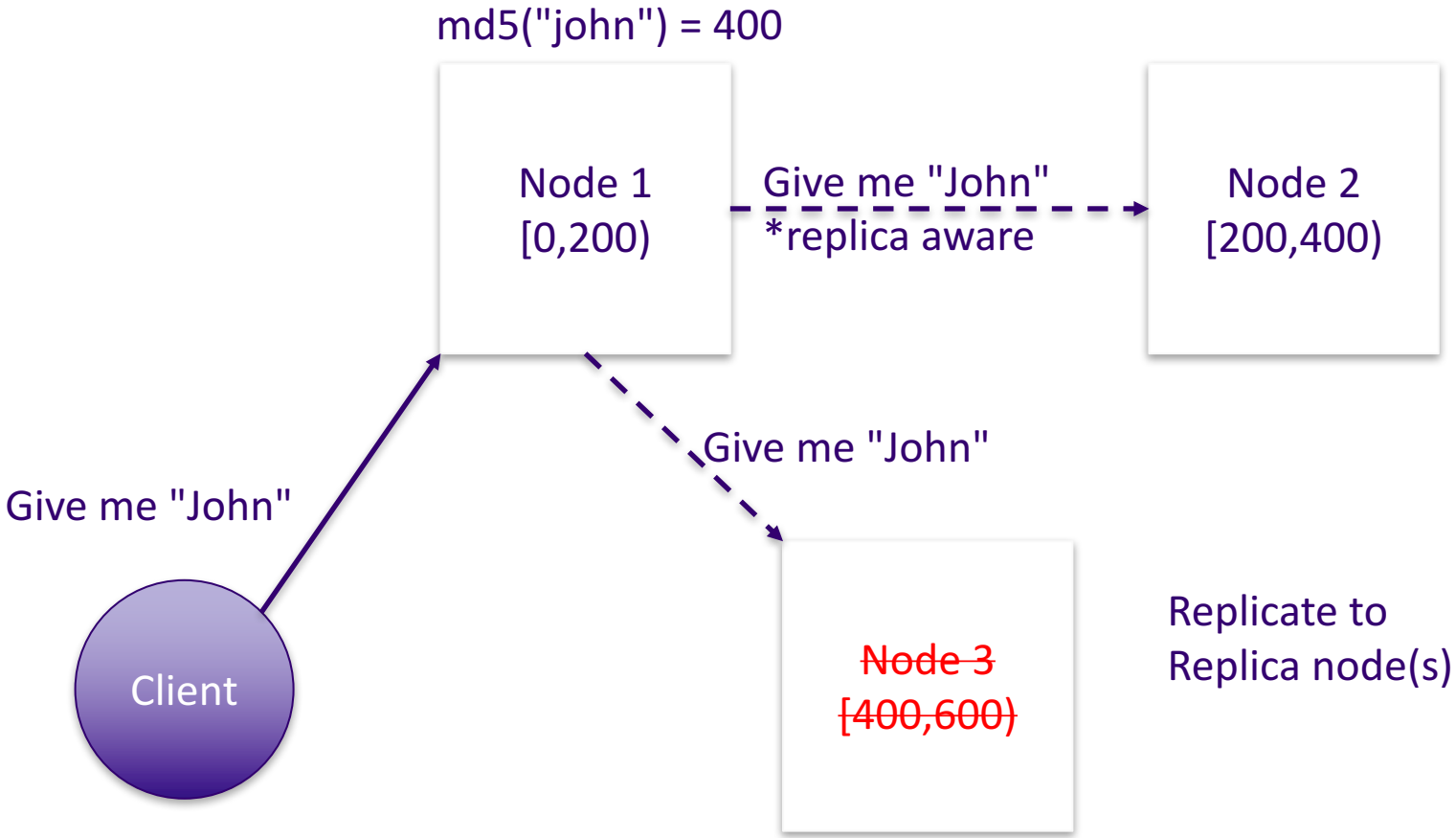
Distributed/Consistent Hashing



Reading


md5("john") = 400
Node 1
[0,200)
Give me "John"
*replica aware
Node 2
[200,400)
Give me "John"
Give me "John"
Client
Node 3
[400,600)
Replicate to
Replica node(s)

Reading


md5("john") = 400
Node 1
[0,200)
Give me "John"
*replica aware
Node 2
[200,400)
Give me "John"
Give me "John"
Client
~~Node 3~~
~~[400,600)~~
Replicate to
Replica node(s)

Cassandra Query Language (CQL)

```
CREATE TABLE web_log (
    username varchar,
    ip_address varchar,
    action_taken varchar,
    time_stamp timeuuid
    PRIMARY KEY (username, time_stamp)
);

SELECT username, action_taken FROM web_log WHERE time_stamp >
'2017-01-01';

INSERT INTO web_log VALUES ('jdoe',10.1.1.2,
'click:http://blah.com','2017-01-01 12:00:01.0912')
```

UNIVERSITY *of* WASHINGTON

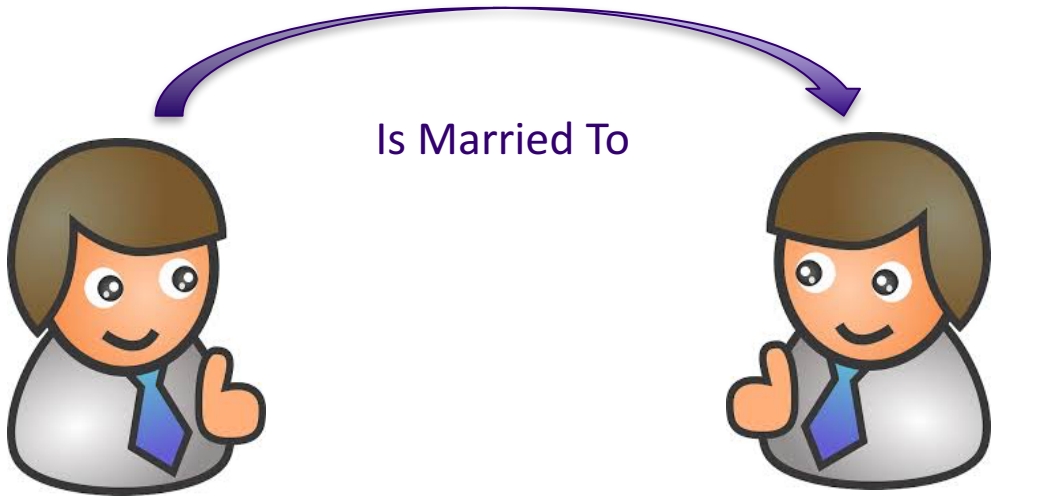
NEO4J: Graph Database

NEO4J

- Open Source
- 1.0 Released in 2010
- Graph Database
  - (ACID Compliant!)
- Written in Java (so not the fastest tool in the barn)
- Implements Relationships between Objects

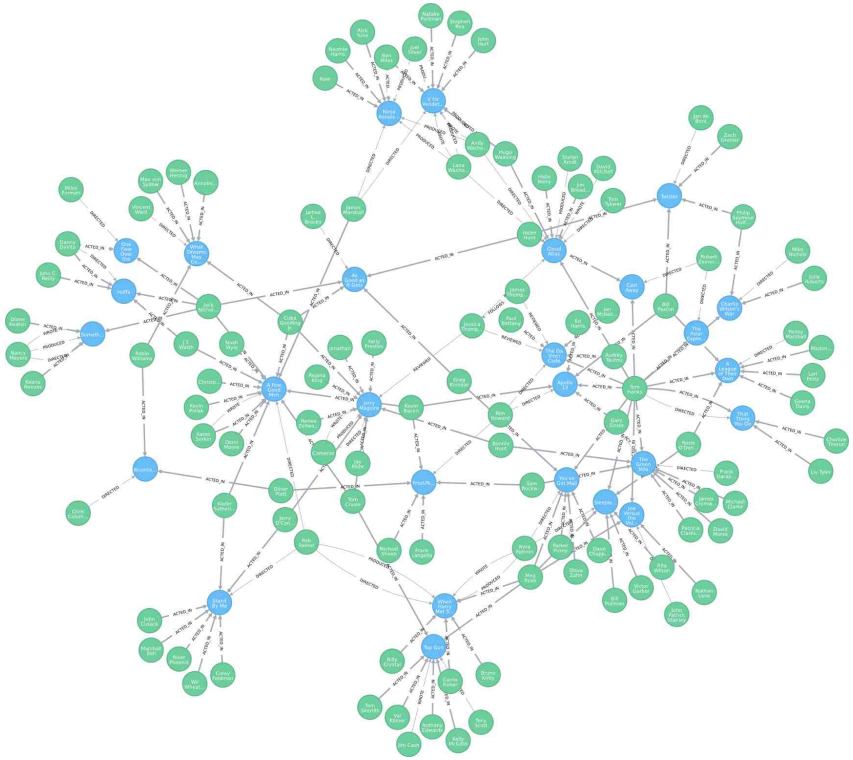
No Tables, No Relations, Just Nodes and Edges

UserName	Date of Birth	Married To
John Doe	1981-02-12	Jane Doe
Jane Doe	1977-06-12	John Doe



Really, it's just a Property Graph

- Nodes, and Edges (Called Relationships)
- All Relationships are Directed!



Let's use our Movie Example

INDIANA JONES

STAR WARS

Let's use our Movie Example

:MovieStar

:MovieStar

:MovieStar

:Movie

:Movie

Let's use our Movie Example

:MovieStar
f_name: "Harrison"
l_name: "Ford"

:MovieStar
f_name: "Carrie"
l_name: "Fisher"

:MovieStar
f_name: "Mark"
l_name: "Hammil"

:Movie
title: "Indiana Jones"
released: 1981

:Movie
title:"Star Wars"
released: 1977

Let's use our Movie Example

Stars In

Stars In

Stars In

Stars In

Let's use our Movie Example

Stars In
Role: Lead

Stars In
Role: Han Solo

Stars In
Role: Princess Leia

Stars In
Role: Lead, Luke Skywalker

Query language called Cipher (VERY MUCH NOT SQL)

```
CREATE (n:Star { name:"Harrison Ford"}) RETURN
CREATE (n:Star { f_name:"Carrie",l_name:"Fisher"} )

MATCH (n:Star {name:"Harrison Ford"})
   SET n.birth_date = '1942-07-13'

MATCH (a:Star {name: "Harrison Ford"})
, (b:Movie {title="Star Wars"})  MERGE (a)-[r:StarsIn]->(b)

MATCH (a:Movie{title:"Star Wars"})<-[:StarsIn]-(b:Star)
  RETURN b
```

UNIVERSITY *of* WASHINGTON

MongoDB

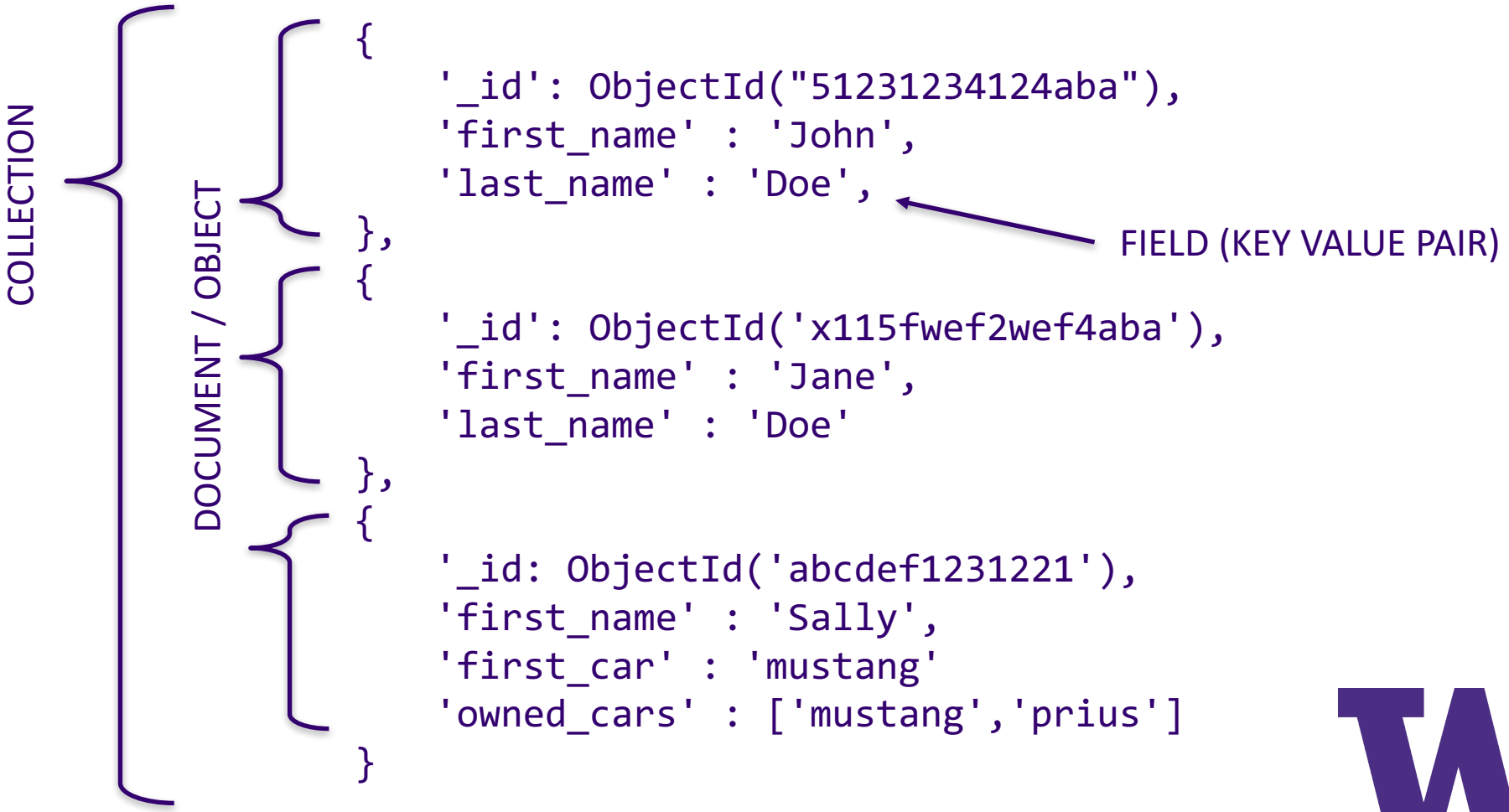
MongoDB

- Document Oriented Database
- Horizontally Scalable
- High Performance on Large Datasets

Mongo DB Pros and Cons

Pros	Cons
High Performance on large datasets	No joins
Horizontally Scalable	No ACID Transactions
No Rigid Schema	*No Rigid Schema*
Supports JavaScript inside Queries	No constraints / integrity
“Fault Tolerant”	Sometimes loses your data

How records are stored in MongoDB



How records are stored in MongoDB

```
{
    '_id': ObjectId('51231234124aba'),
    'first_name' : 'John',
    'last_name' : 'Doe',
    'addresses' : [
            {
                'street' : '1234 Road Street',
                'city' : 'Tacoma',
                'state’ : 'WA'
                'ZIP' : 98405
            },
            {
                'street' : '4567 Street Avenue'
                'city' : 'Seattle'
            }
        ]
}
```

Information Retrieval

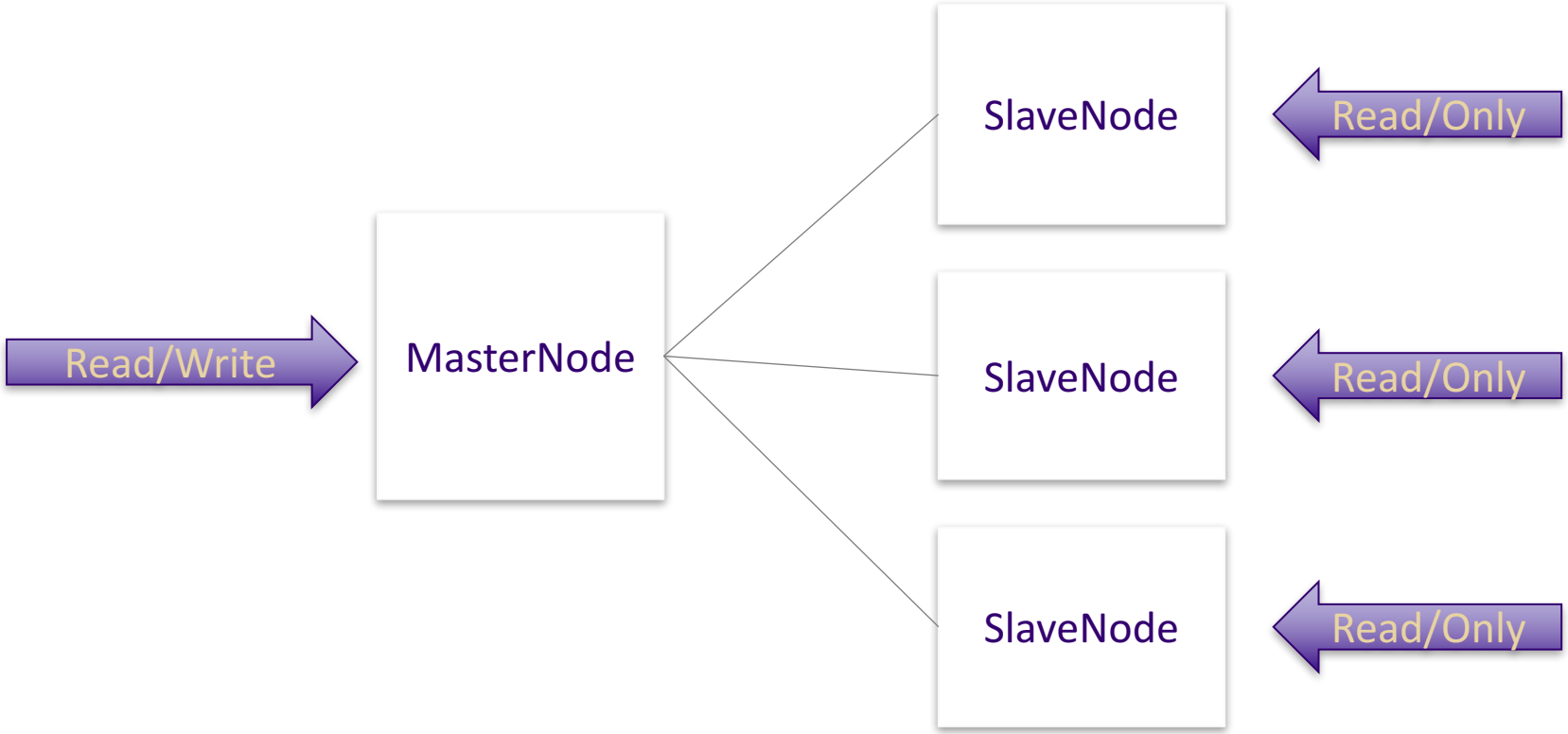
```
db.collectionname.find( { _id: 12345 } );

db.people.find( { name: 'John' } );
```

```
db.people.find( { name: 'John' } ).sort( { last_name:1 } );
```

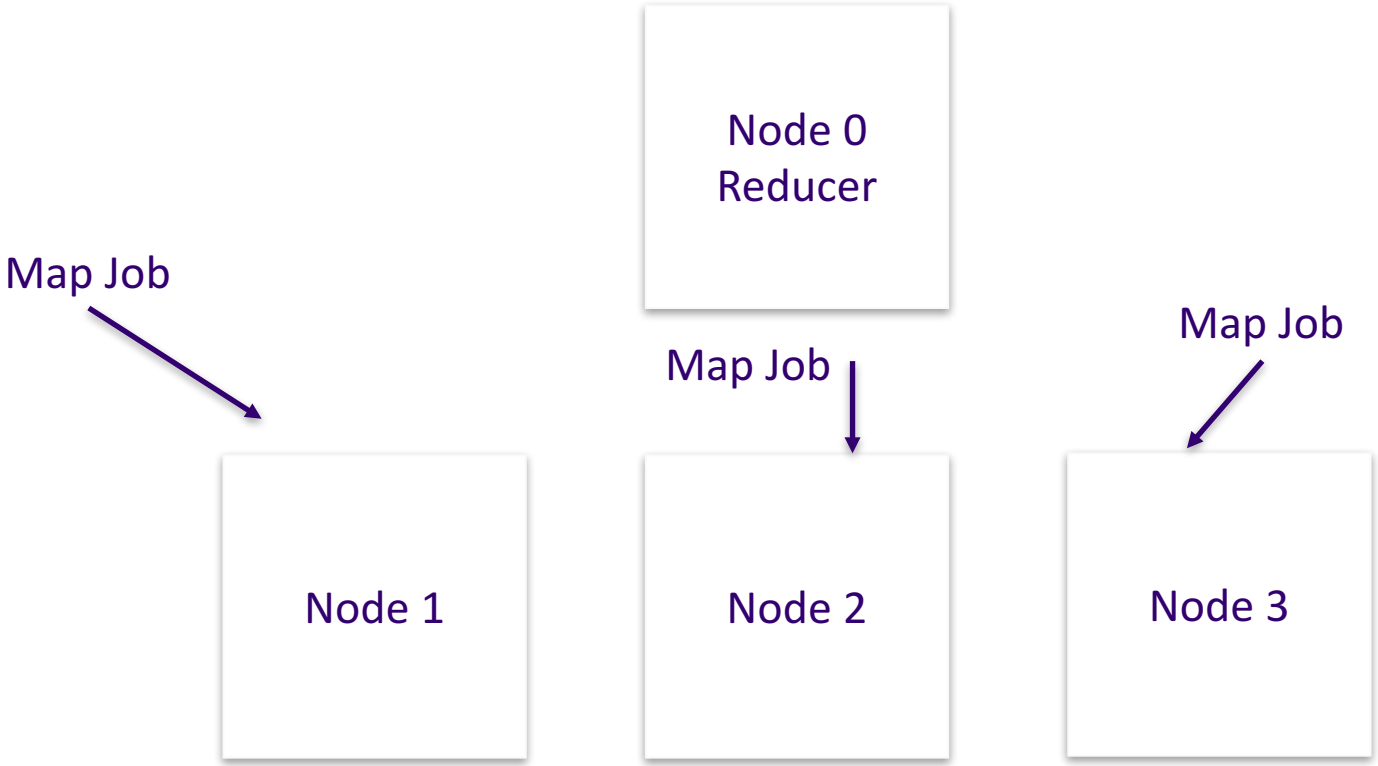
Information Retrieval

```
db.people.insert( {  name: 'John',
                     state: 'WA'}
                  , {  name: 'Jane',
                     state: 'PA'} );
```

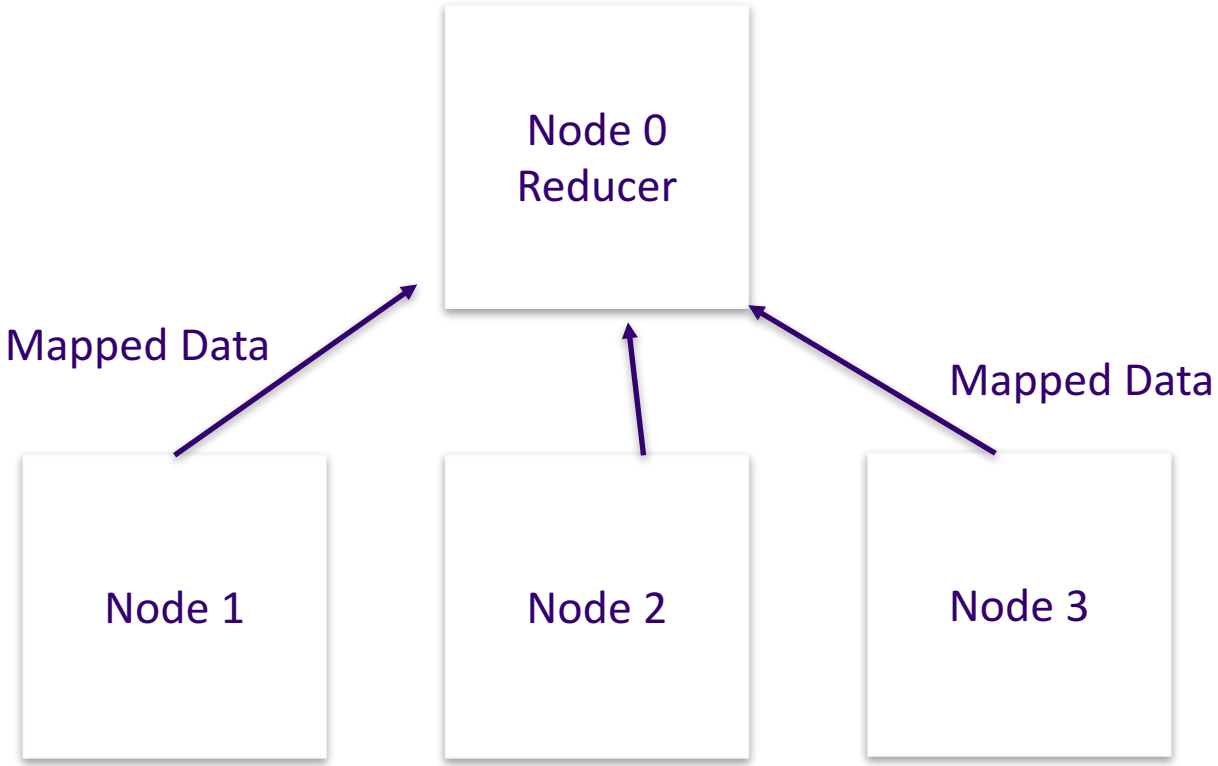
Master Slave Duplication



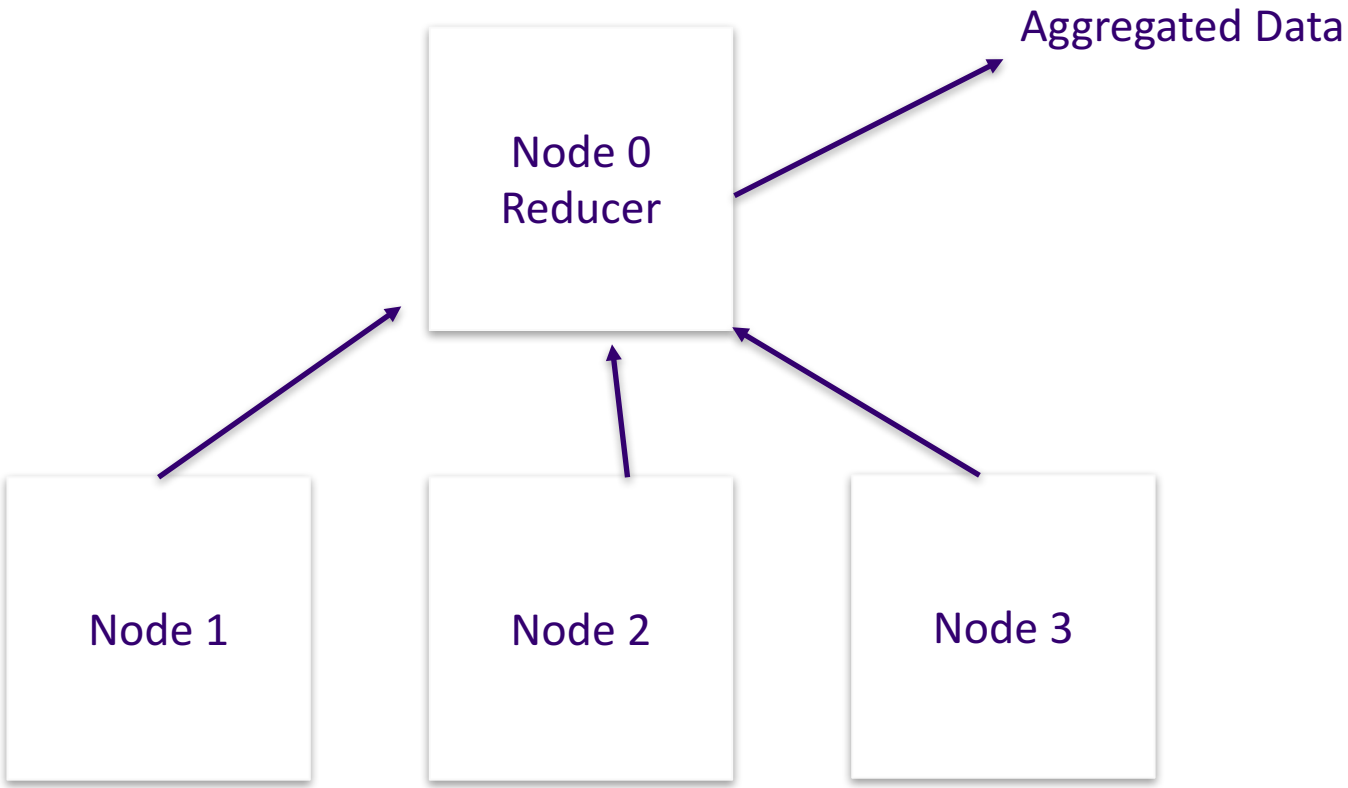
Aggregation: MapReduce



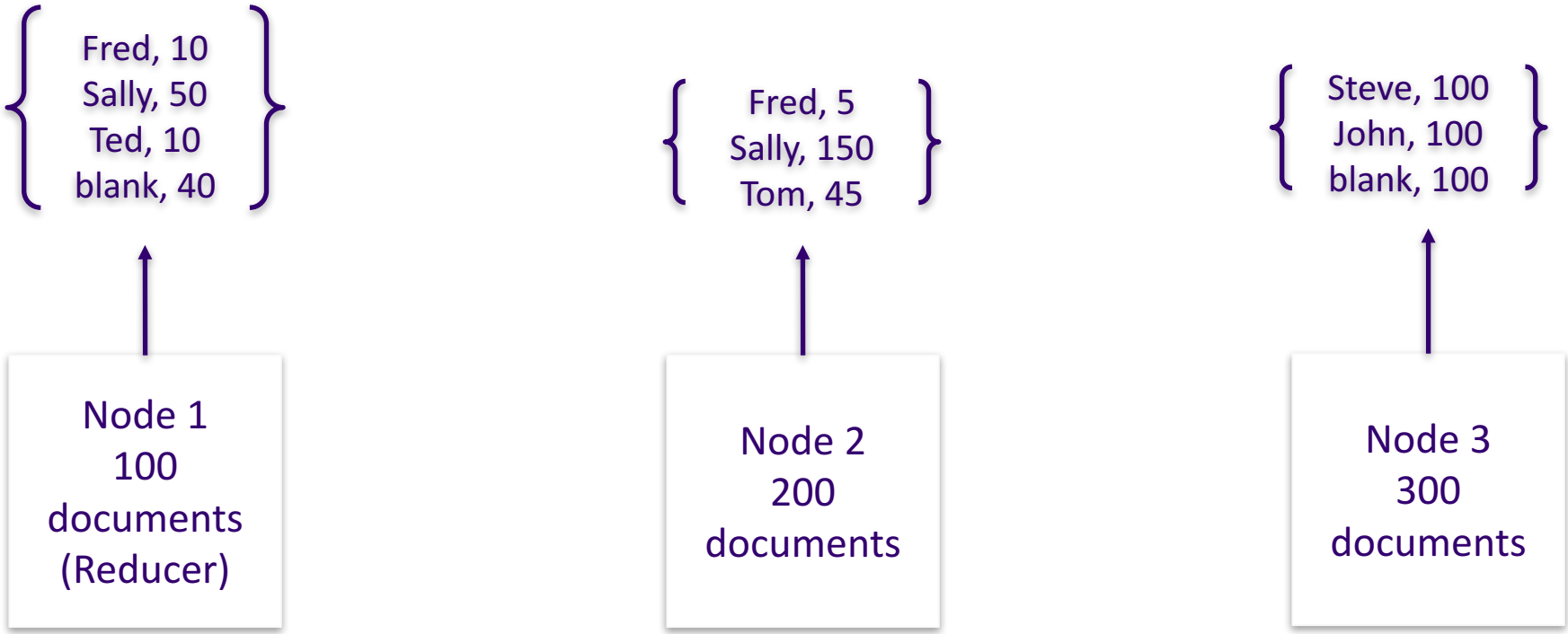
Aggregation: MapReduce



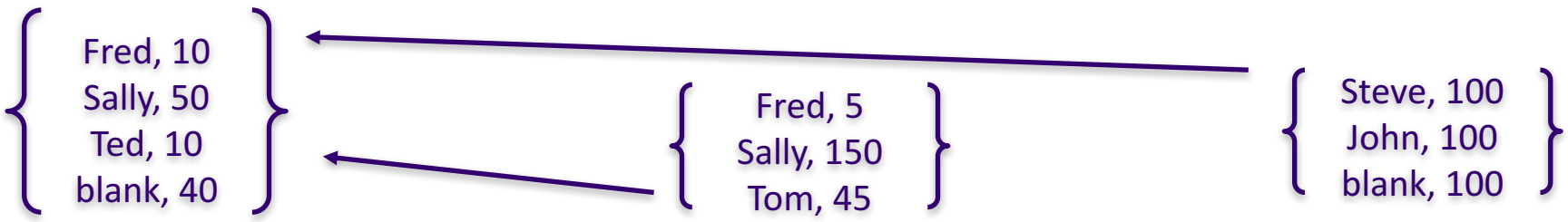
Aggregation: MapReduce



Blog Count By Name Example



Blog Count By Name Example

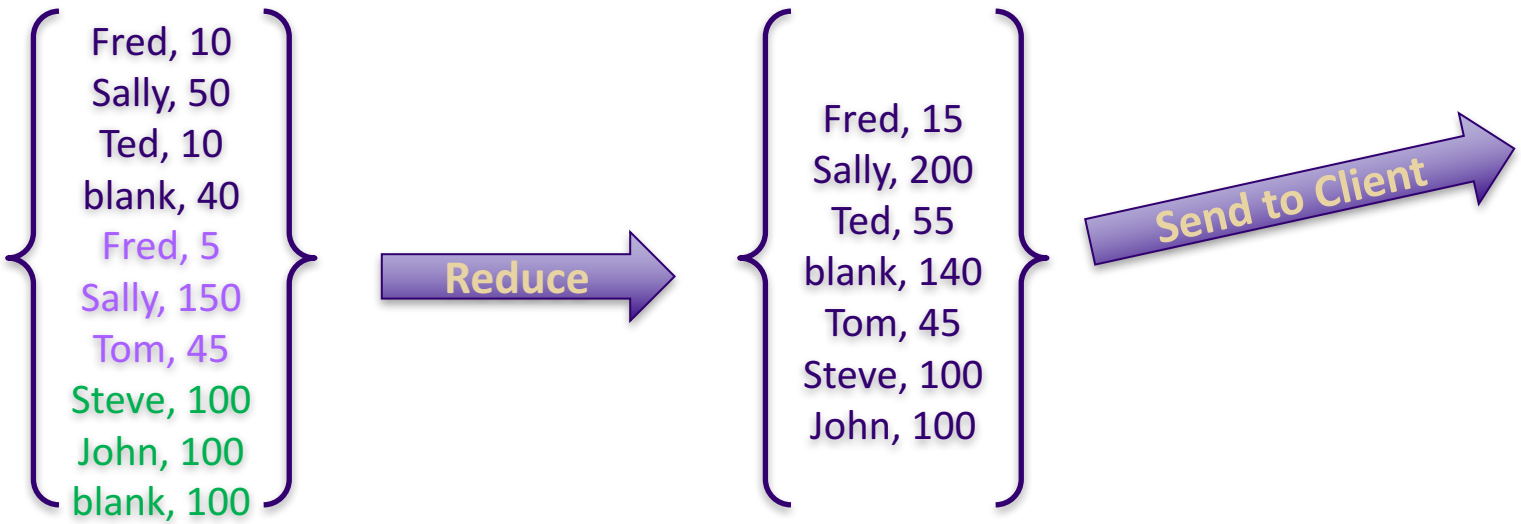


Node 1
100
documents
(Reducer)

Node 2
200
documents

Node 3
300
documents

Blog Count By Name Example



Node 1
100 documents
(Reducer)

Node 2
200
documents

Node 3
300
documents