Summary of Big Data Modeling and Management

After this video you will be able to..

- Recall why big data modeling and management is essential in preparing to gain insights from your data
- Summarize different kinds of data models
- Describe streaming data and the different challenges it presents
- Explain the differences between a DBMS and a BDMS

Big Data Modeling and Management

- Data modeling tells you
  - How your data is structured
  - What operations can be done on the data
  - What constraints apply to the data
- Database Management Systems
  - Typically handle many low-level details of data storage, manipulation, retrieval, transactional updates, failure and security
  - Relieves a user to focus on higher level operations like querying and analysis

Different Data Models

- Relational Data
  - Where data look like tables
- Semi-structured Data
  - Document data, XML and JSON
- Graph Data
  - Social Networks, email networks
- Text Data
  - Articles, reports

Streaming Data

- An infinite flow of data coming from a data source
  - Sensor data from instruments
  - Stock price data
- Data rates vary – can be too fast and too large to store
- Often processed in memory
- May need to be processed immediately
  - Inform whenever 3 tech stocks go up by 3% within a 30 second span
  - Used for event detection and prediction

DBMS and BDMS

- BDMS
  - Designed for parallel and distributed processing
    - Data-partitioned parallelism
  - May not always guarantee consistency for every update
    - More likely to guarantee eventual consistency
  - Often built-on Hadoop
    - Offer Map-reduce style computation
    - Utilizes replication natively offered by HDFS

Why is Big Data Processing Different?

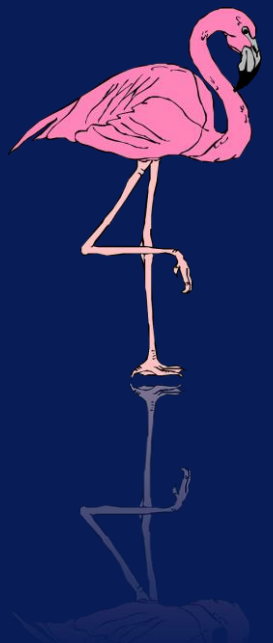
After this video you will be able to..

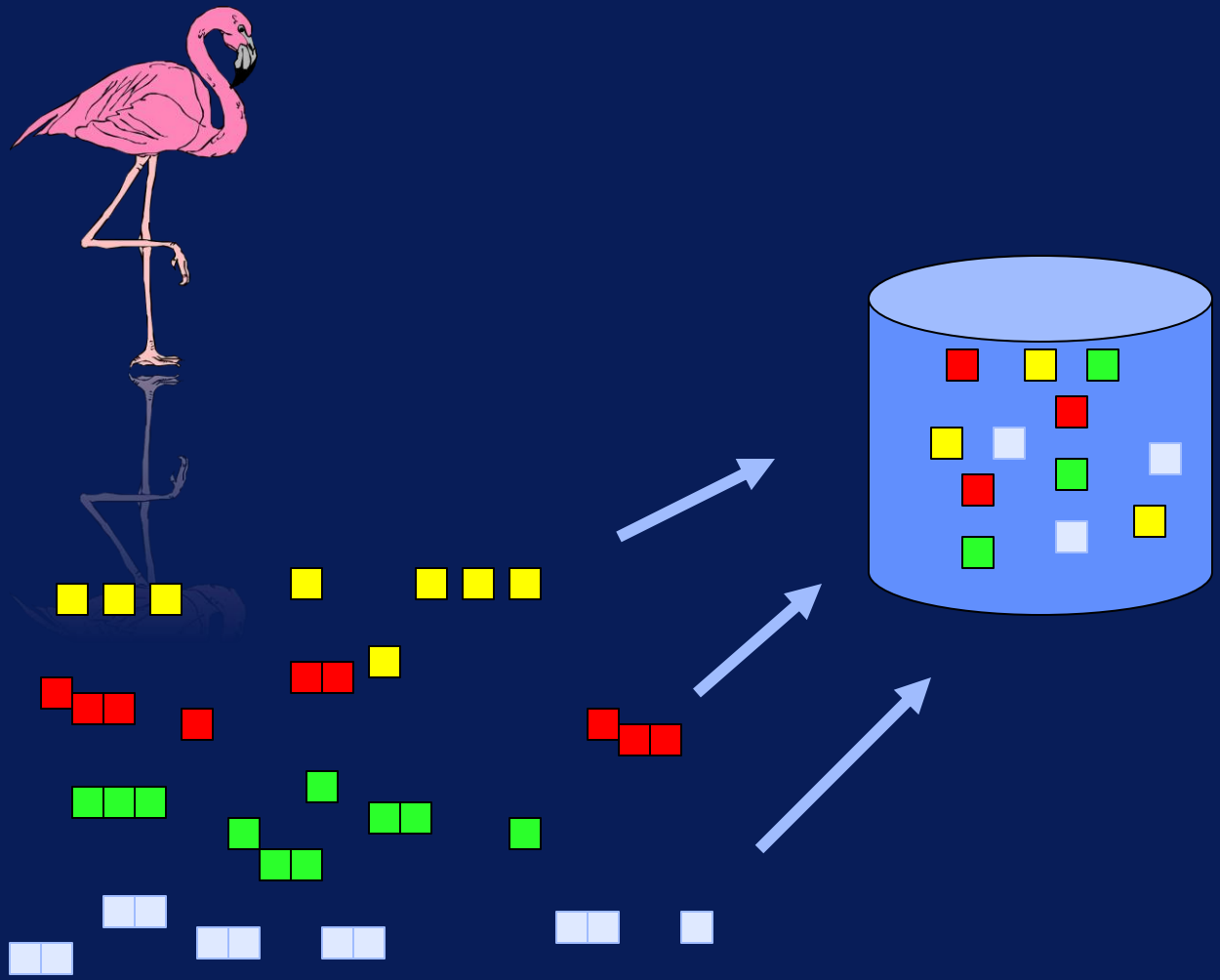
- Summarize the requirements of programming models for big data and why you should care about them

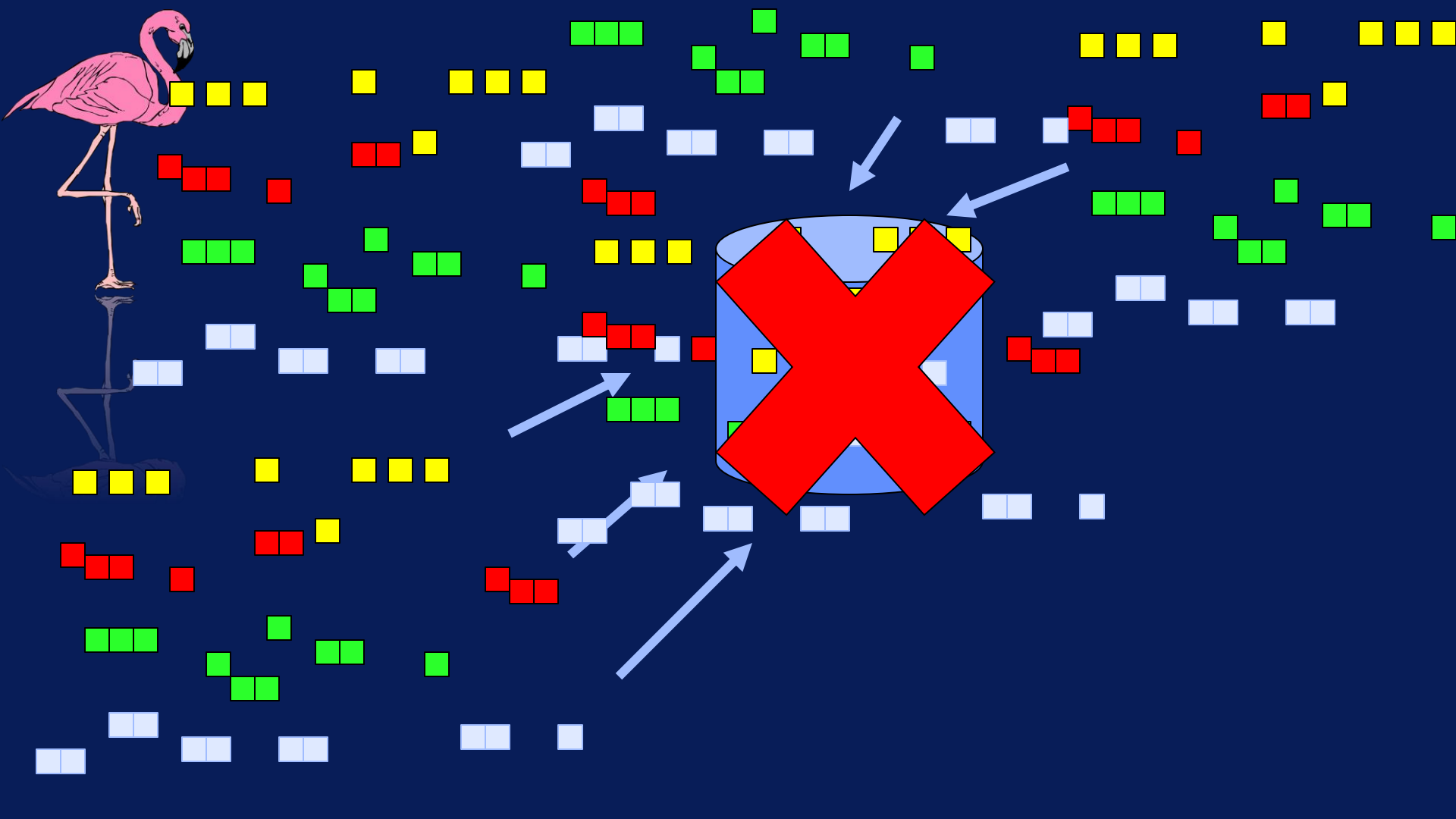
- Explain how the challenges of big data related to its variety, volume and velocity affects its processing

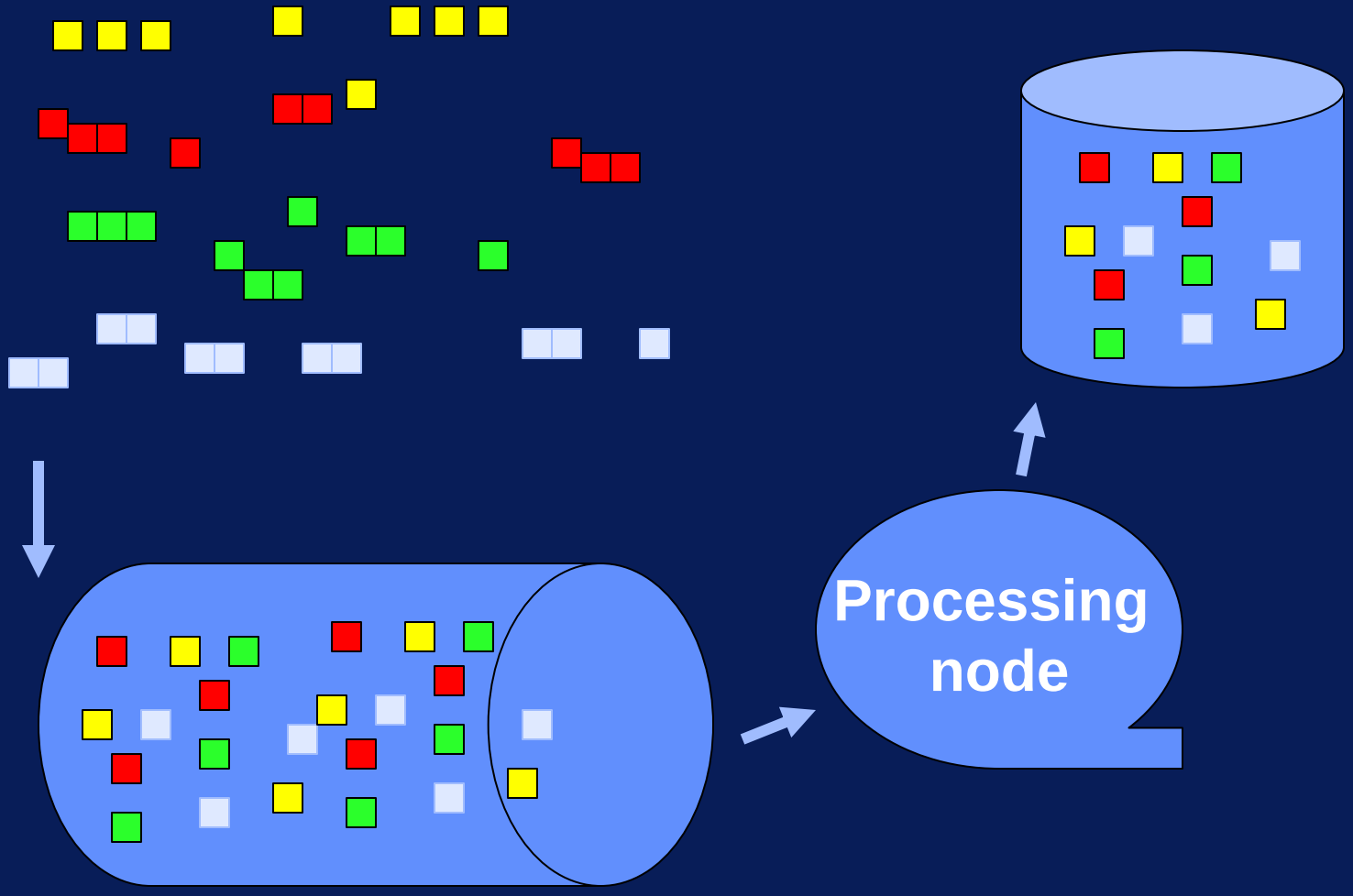
Requirements for Big Data Systems

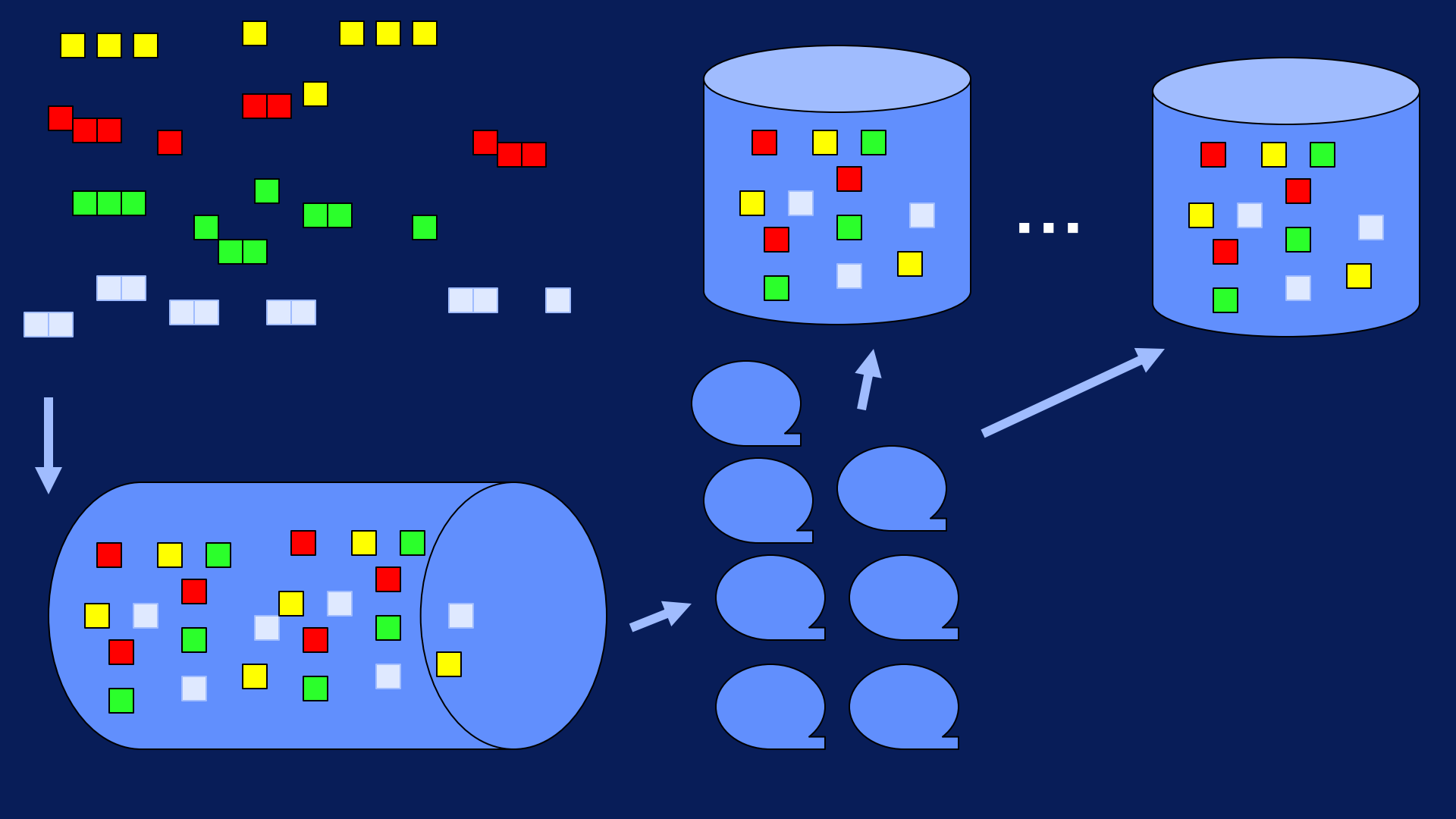
A Big Data System for an Online Game

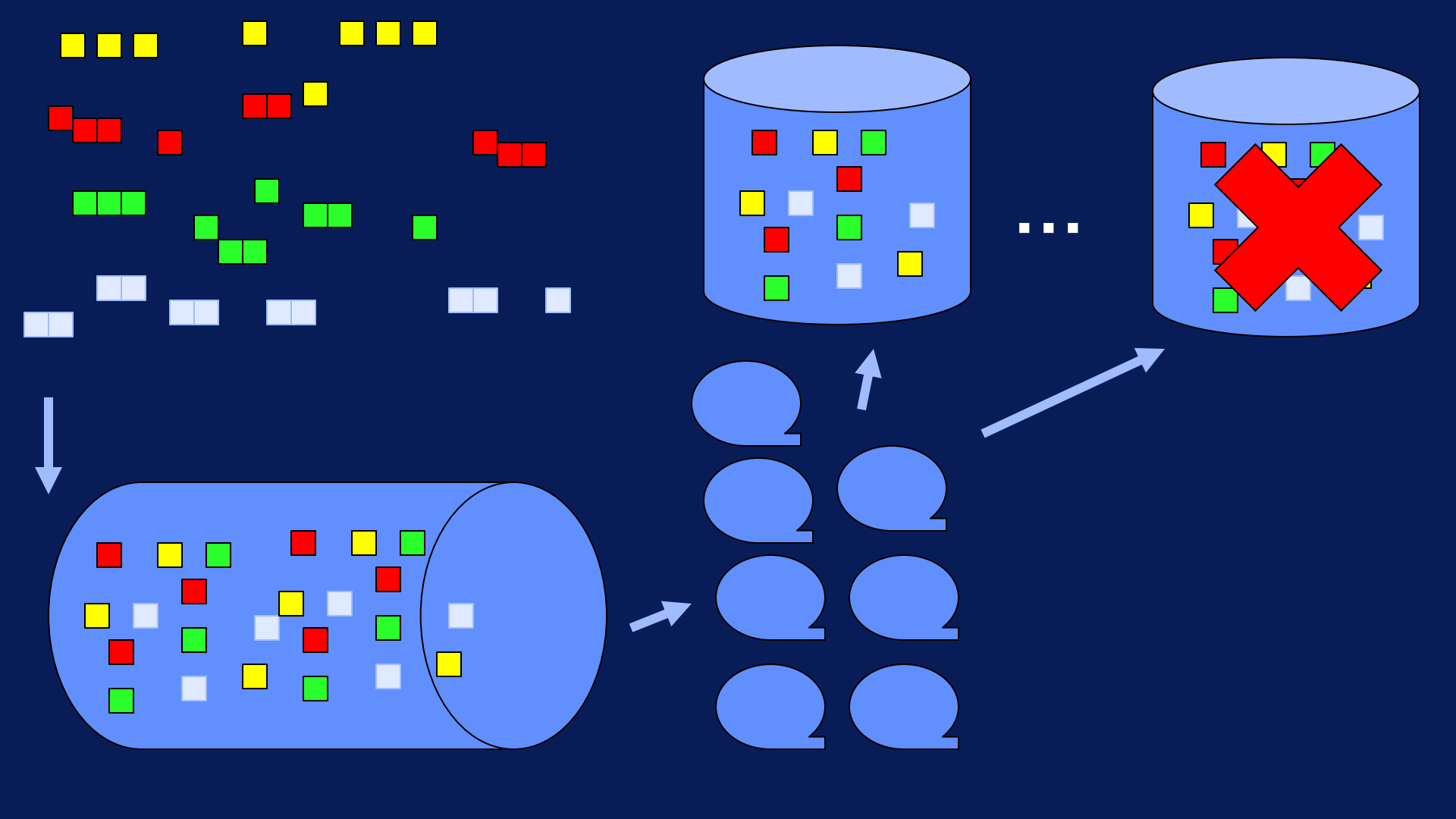


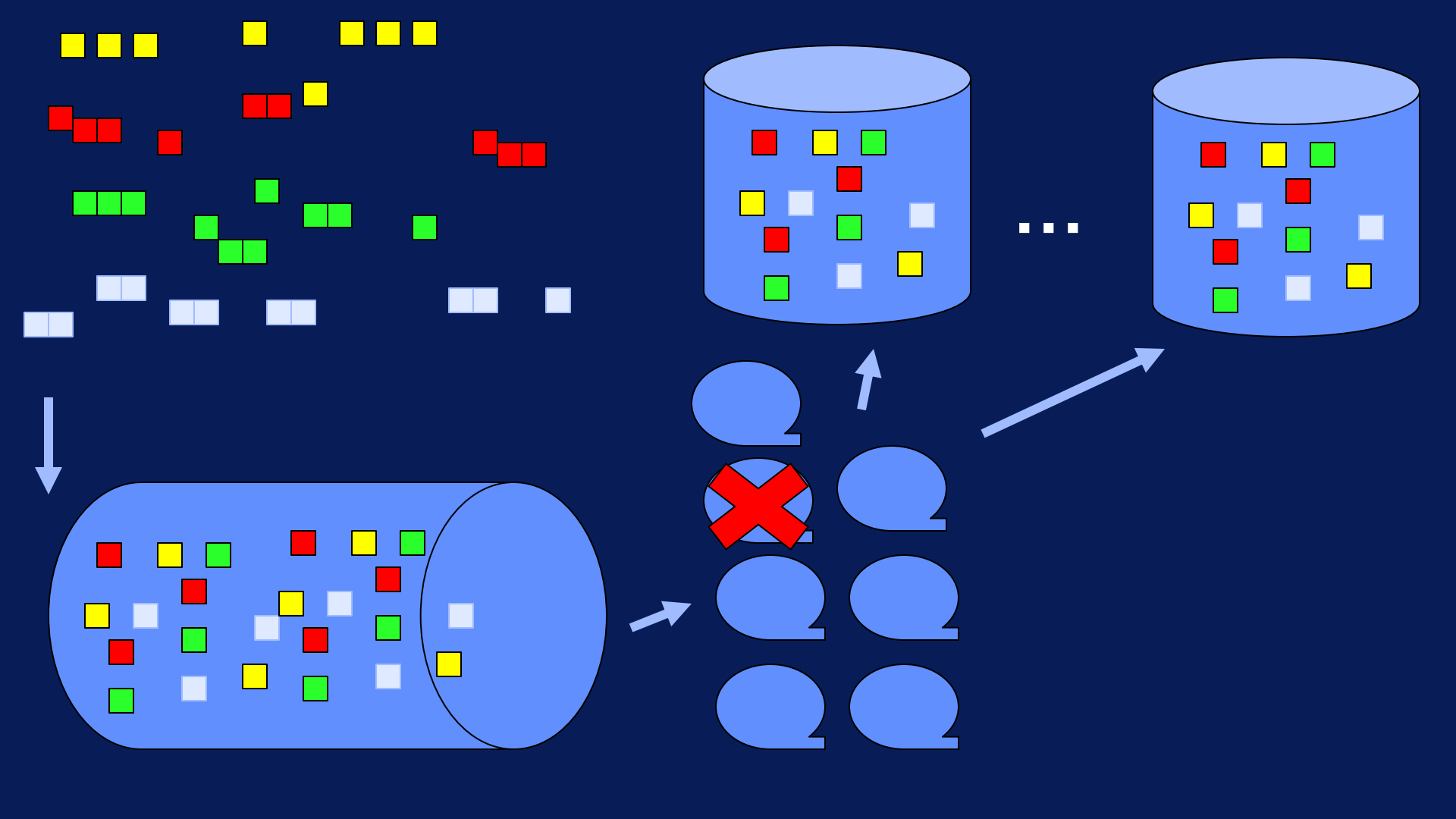


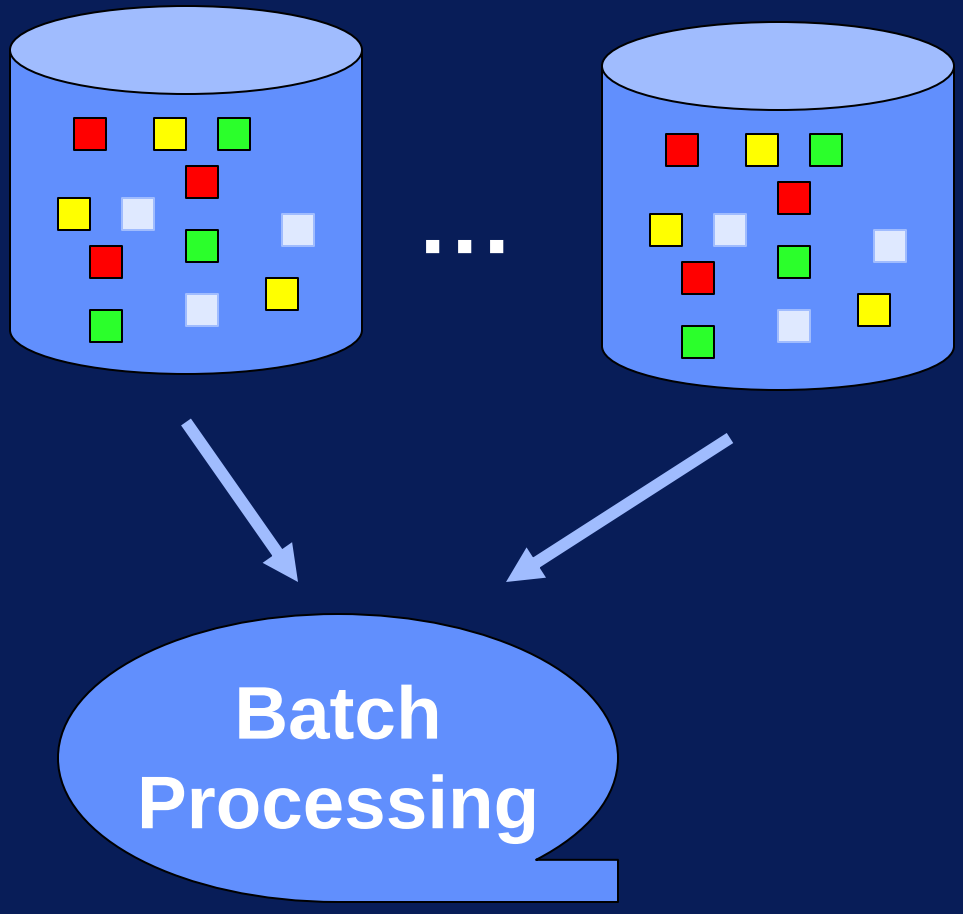



Processing node

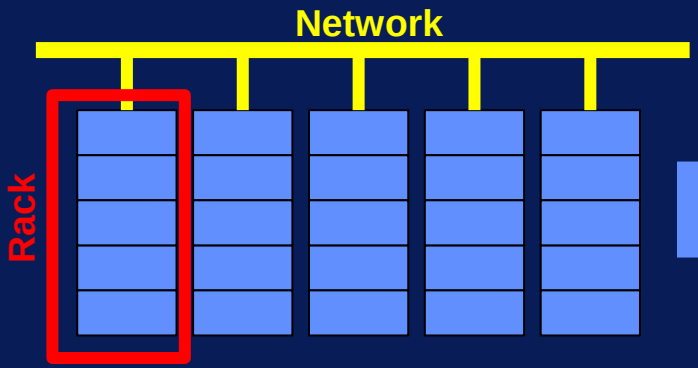




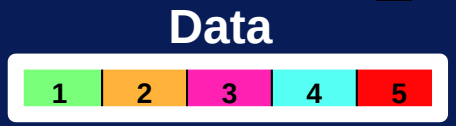


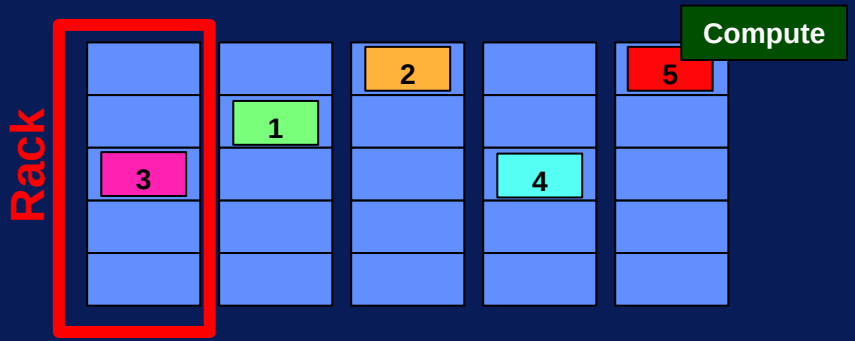

...
Batch
Processing

Scalability
Complexity



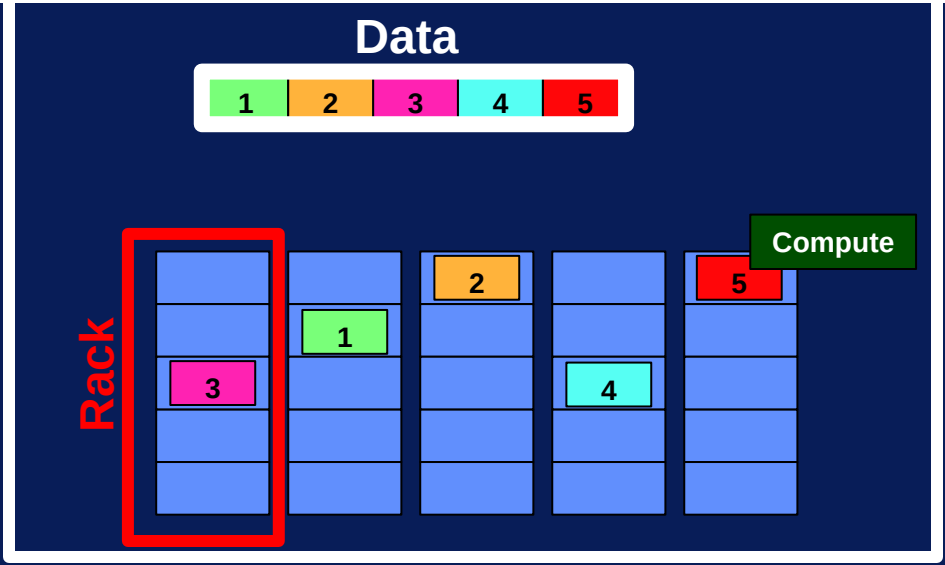
Data-parallel scalability





Programming Model = abstractions

Runtime Libraries + Programming Languages



Requirements for Big Data Systems

1. Support Big Data Operations

Split volumes of data

1. Support Big Data Operations

Split volumes of data

Access data fast

1. Support Big Data Operations

Split volumes of data

Access data fast

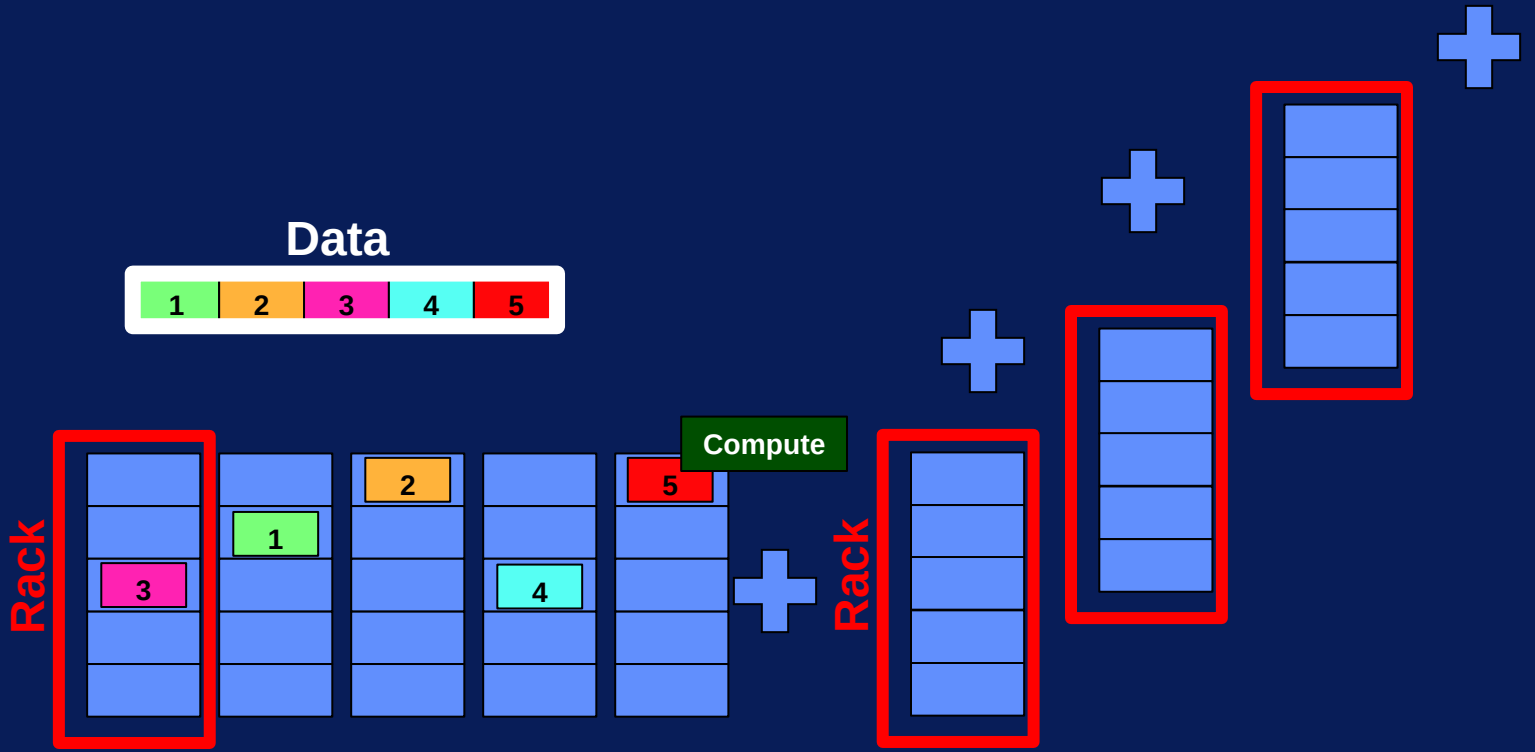
Distribute computations to nodes

2. Handle Fault Tolerance

Replicate data partitions

Recover files when needed

3. Enable Adding More Racks



4. Optimized and extensible for many data types

Document

Table

Graph

Key-value

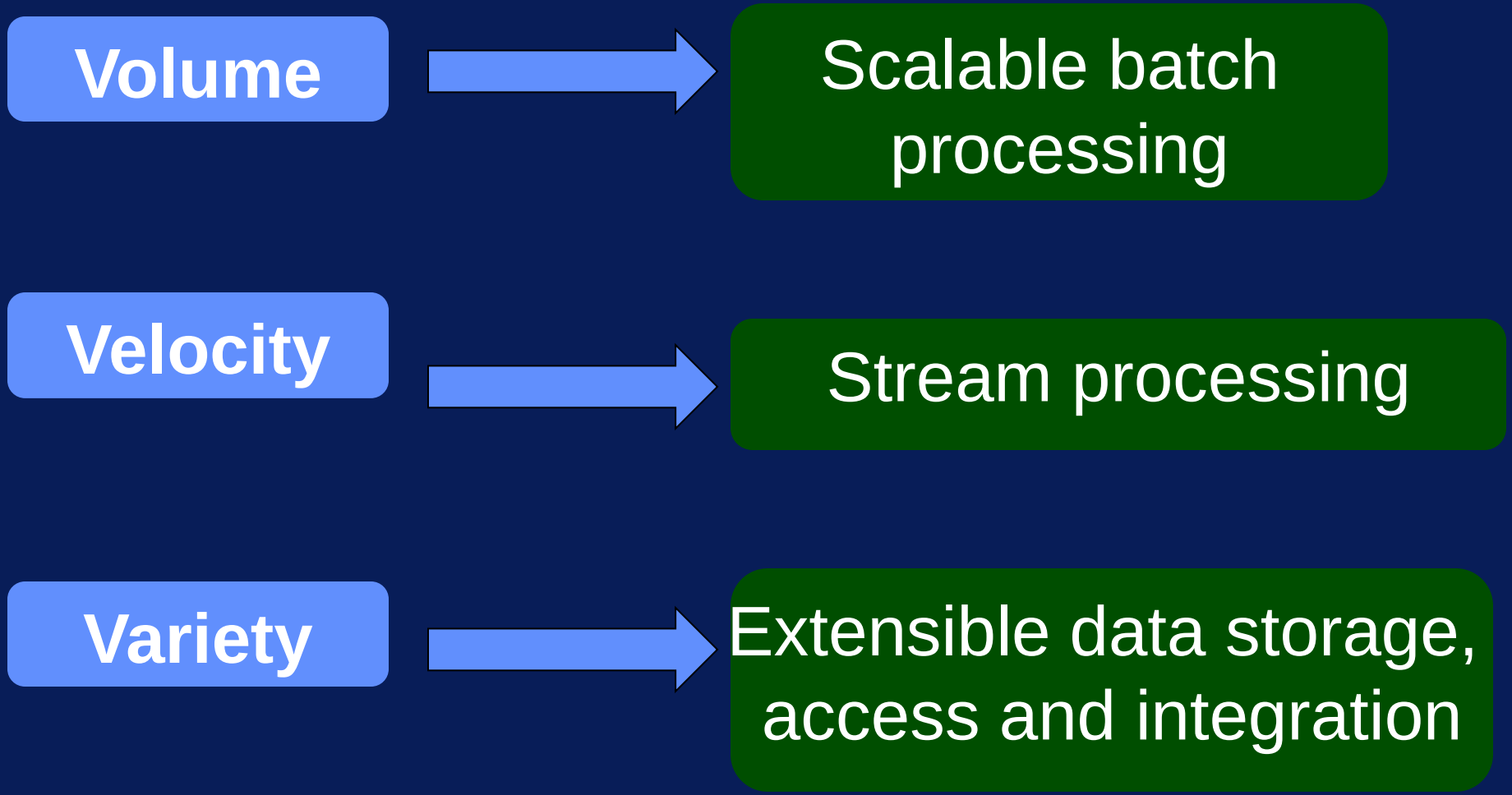
Multimedia

Stream

5. Enable both streaming and batch processing

Low latency processing of streaming data

Accurate processing of all available data


Volume
Scalable batch processing
Velocity
Stream processing
Variety
Extensible data storage, access and integration

What is Data Retrieval?

SDSC SAN DIEGO SUPERCOMPUTER CENTER

What is Data Retrieval?

- **Data retrieval**
  - The way in which the desired data is specified and retrieved from a data store
- **Our focus**
  - How to specify a data request
    - For static and streaming data
  - The internal mechanism of data retrieval
    - For large and streaming data

What is a Query Language?

- **A language to specify the data items you need**
- **A query language is declarative**
  - Specify what you need rather than how to obtain it
  - SQL (Structured Query Language)
- **Database programming language**
  - Procedural programming language
  - Embeds query operations

SQL

- **The standard for structured data**
  - Oracle's SQL to Spark SQL
- **Example Database Schema**

  Bars(name, addr, license)
  Beers(name, manf)
  Sells(bar, beer, price)
  Drinkers(name, addr, phone)
  Frequents(drinker, bar)
  Likes(drinker, beer)

name	addr	license
Great American Bar	363 Main St., SD, CA 92390	41-437844098
Beer Paradise	6450 Mango Drive, SD, CA 92130	41-973428319
Have a Good Time	8236 Adams Avenue, SD, CA 92116	32-032263401

SELECT-FROM-WHERE

- **Which beers are made by Heineken?**

```
SELECT name
FROM Beers
WHERE manf = 'Heineken'
```

Output attribute(s)

Table(s) to use

The condition(s) to satisfy

Strings like 'Heineken' are case-sensitive and are put in quotes

$Select_{manf='Heineken'}(Beers)$

↓

Project(name)

name
Heineken Lager Beer
Amstel Lager
Amstel Light
…

More Example Queries

- **Find expensive beer**
  - SELECT DISTINCT beer, price
  - FROM Sells
  - WHERE price > 15
- **Which businesses have a Temporary License (starts with 32) in San Diego?**
  - SELECT name
  - FROM Bars
  - WHERE addr LIKE '%SD%' **AND** license LIKE '32%' LIMIT 5

name	addr	license
Great American Bar	363 Main St., SD, CA 92390	41-437844098
Beer Paradise	6450 Mango Drive, SD, CA 92130	41-973428319
Have a Good Time	8236 Adams Avenue, SD, CA 92116	32-032263401

Select-Project Queries in the Large

- **Large Tables can be partitioned**
  - Many partitioning schemes
    - Range partitioning on primary key

name	manf
A…	Gambrinus
A…	Heineken
…	
B…	Anheuser-Busch

Machine 1

name	manf
C…	MillerCoors
C…	MillerCoors
…	
D…	Duvel Moortgat

Machine 2

…

name	manf
H…	Heineken
H…	Pabst
…	
H…	Anheuser-Busch

Machine 5

…

Select-Project Queries in the Large

name	manf
A…	Gambrinus
A…	Heineken
…	
B…	Anheuser-Busch

Machine 1

name	manf
C…	MillerCoors
C…	MillerCoors
…	
D…	Duvel Moortgat

Machine 2

name	manf
H…	Heineken
H…	Pabst
…	
H…	Anheuser-Busch

Machine 5

```
SELECT *
FROM Beers
WHERE name like 'Am%'
```

pattern

```
SELECT name
FROM Beers
WHERE manf = 'Heineken'
```

- **Two queries**
  - Find records for beers whose name starts with 'Am'
  - Which beers are made by Heineken?

Evaluating SP Queries for Large Data

name	manf
A…	Gambrinus
A…	Heineken
…	
B…	Anheuser-Busch

Machine 1

name	manf
C…	MillerCoors
C…	MillerCoors
…	
D…	Duvel Moortgat

Machine 2

…

name	manf
H…	Heineken
H…	Pabst
…	
H…	Anheuser-Busch

Machine 5

…

```
SELECT *
FROM Beers
WHERE name like 'Am%'
```

- **A query processing trick**
  - Use the partitioning information
    - Just use partition 1!!

Evaluating SP Queries for Large Data

name	manf
A…	Gambrinus
A…	Heineken
…	
B…	Anheuser-Busch

Machine 1

name	manf
C…	MillerCoors
C…	MillerCoors
…	
D…	Duvel Moortgat

Machine 2

…

name	manf
H…	Heineken
H…	Pabst
…	
H…	Anheuser-Busch

Machine 5

…

```
SELECT name
FROM Beers
WHERE manf = 'Heineken'
```

Broadcast query

In each machine in parallel:

Select $_{manf='Heineken'}$ *(Beers)*

Project(name)

Gather Partial Results

Union

Return

Local and Global Indexing

- **What if a machine does not have any data for the query attributes?**
- **Index structures**
  - Given value, return records
  - Several solutions
    - Use local index on each machine
    - Use a machine index for each value
    - Use a combined index in a global index server

manf	RecordIDs
…	…
MillerCoors	34, 35, 87, 129, …
Duvel Moortgat	5, 298, 943, 994,…
Heineken	631, 683, 882,…
…	…

manf	machineIDs
…	…
MillerCoors	10
Duvel Moortgat	3, 4
Heineken	1, 3, 5
…	…

Pause

Querying Two Relations

- **Often we need to combine two relations for queries**
  - Find the beers liked by drinkers who frequent The Great American Bar
- **In SQL**

```
SELECT DISTINCT beer
FROM Likes L, Frequents F
WHERE bar = The Great American Bar' AND
      F.drinker = L.drinker
```

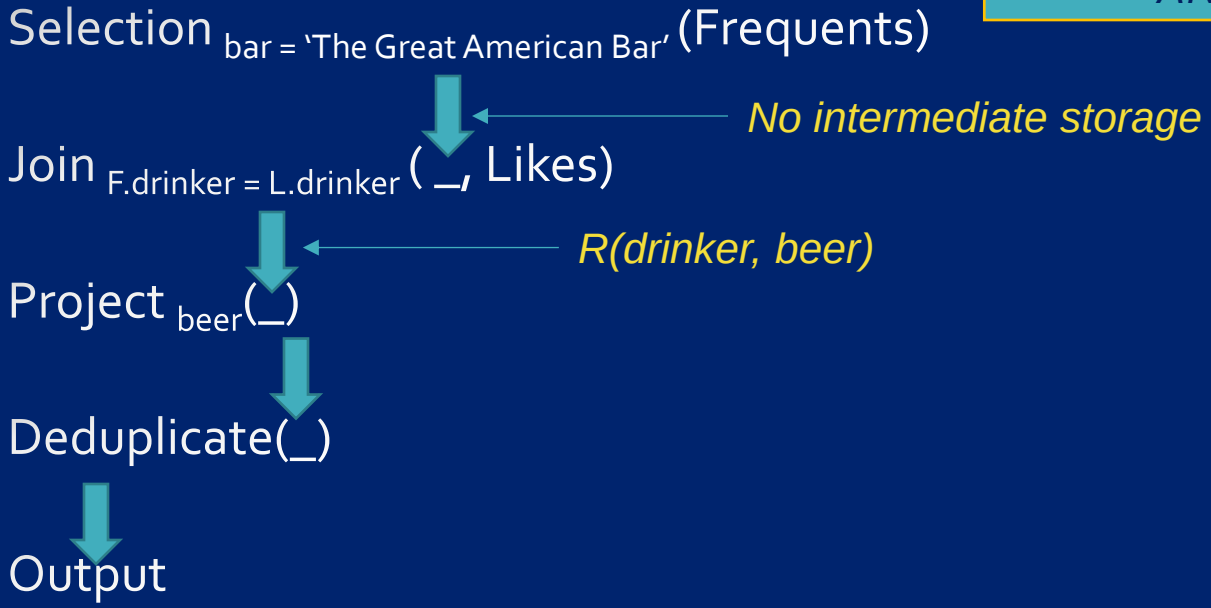
Frequents(drinker, bar)
Likes(drinker, beer)

SPJ Queries

Frequents(drinker, bar)
Likes(drinker, beer)

```
SELECT DISTINCT beer
FROM Likes L, Frequents F
WHERE bar = 'The Great American Bar'
      AND F.drinker = L.drinker
```

- **Steps**

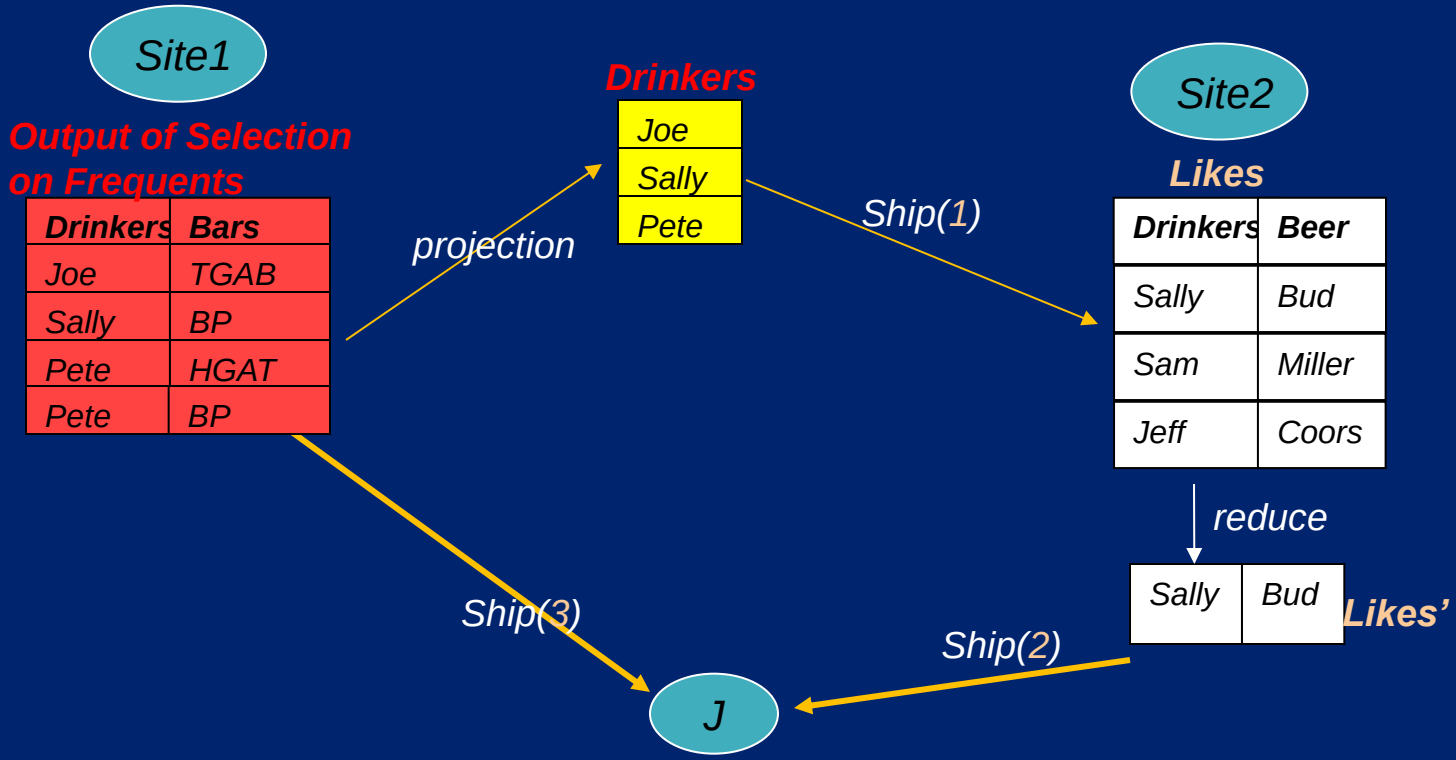


Join in a Distributed Setting

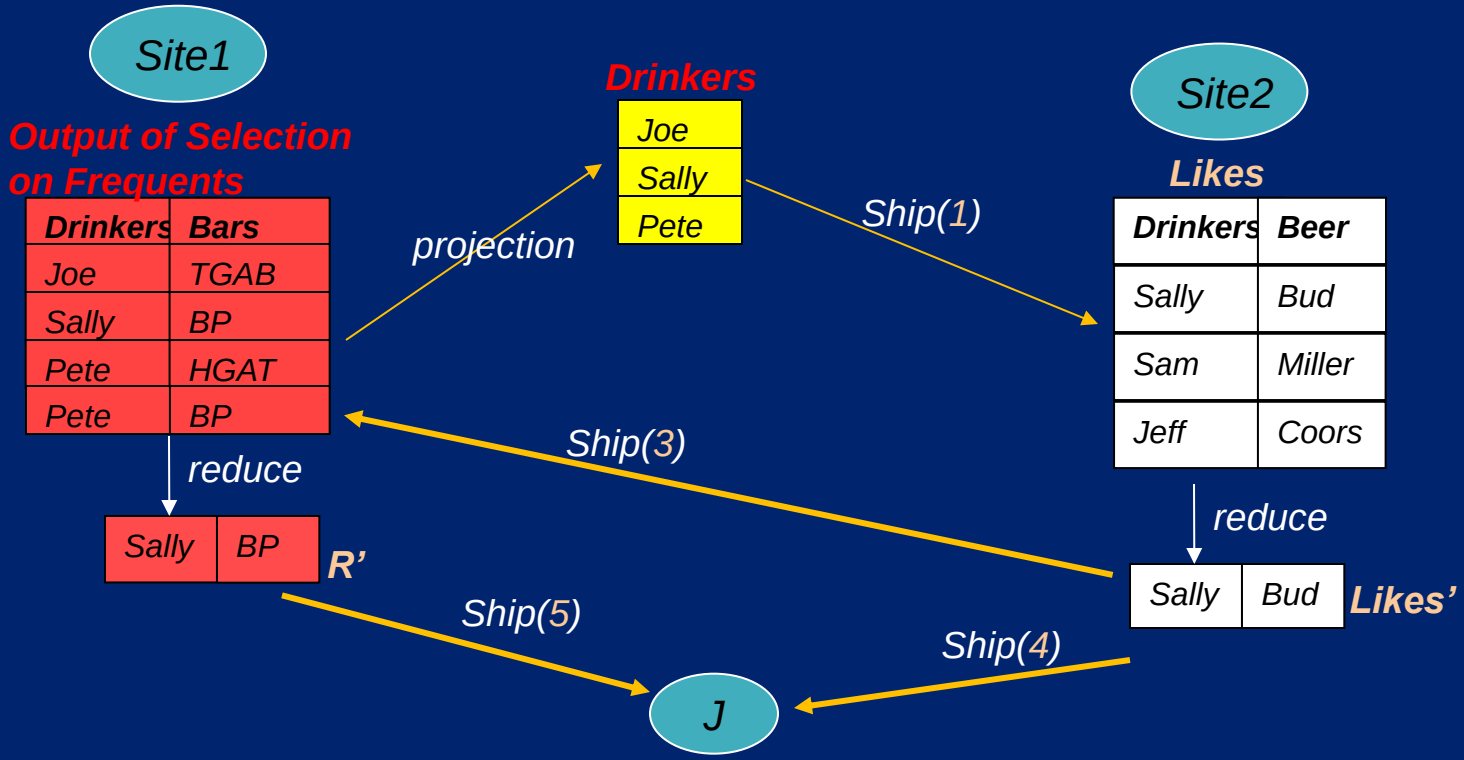
Frequents(drinker, bar)
Likes(drinker, beer)

- **Semijoin**
  - A semijoin from R to S on attribute is used to reduce the data transmission cost
  - Computing steps:
    - **Project** R on attribute A and call it (R[A]) – the Drinkers column
    - **Ship** this projection (a semijoin projection) from the site of R to the site of S
    - **Reduce** S to S' by eliminating tuples where attribute A are not matching any value in R[A]

Semijoin s: Frequents—Drinkers→Likes



Semijoin s: Frequents—Drinkers→Likes



Pause

Subqueries

- A slightly complex query
- Find the bars that serve Miller for the same or less price than what TGAB charges for Bud
- We may break it into two queries:
  1. Find the price TGAB charges for Bud
  2. Find the bars that serve Miller at that price

Subqueries in SQL

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
    price <= (SELECT price
              FROM Sells
              WHERE bar = 'TGAB'
              AND beer = 'Bud');
```

The price at which TGAB sells Bud

Subqueries with IN

- **Find the name and manufacturer of each beer that Fred does not like**
- **Query**

Beers(name, manf)
Likes(drinker, beer)

```
SELECT *
FROM Beers
WHERE name NOT IN
                ( SELECT beer
                        FROM Likes
                        WHERE drinker = 'Fred');
```

Correlated Subqueries

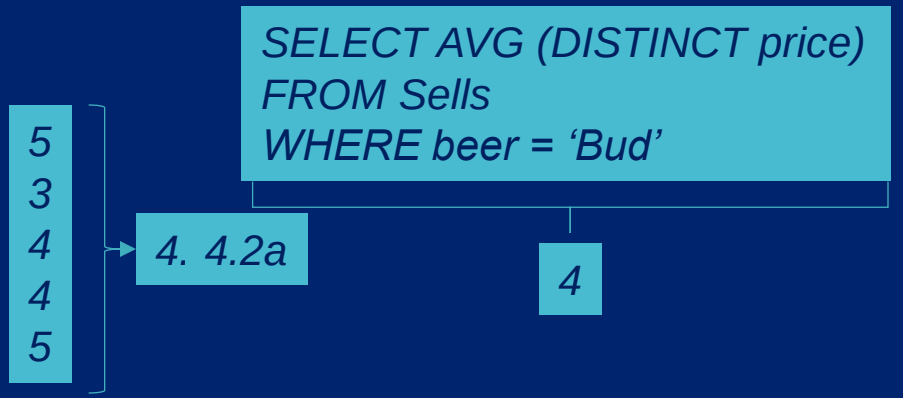
- **Find the name and price of each beer that is more expensive than the average price of beers sold in the bar**

```
SELECT beer, price
FROM  Sells s1
WHERE price >
    (SELECT AVG(price)
     FROM Sells s2
     WHERE s1.bar = s2.bar)
```

Bar	Beer	Price
HGAT	Bud	5
BP	Michelob	4
TGAB	Heineken	6
HGAT	Guinness	10

Aggregate Queries

- **Example**
  - Find the average price of Bud:
  - SELECT AVG(price)
  - FROM Sells
  - WHERE beer = 'Bud';
- **Other aggregate functions**
  - SUM, MIN, MAX, COUNT, ...



GROUP BY Queries

- Find for each drinker the average price of Bud at the bars they frequent

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
GROUP BY drinker;
```

Drinker	Bar	Price
Pete	HGAT	5
Pete	BP	4
Joe	TGAB	6
Joe	HGAT	5

Drinker	Price
Pete	4.5
Joe	5.5

Grouping Aggregates over Partitioned Data

Drinker	Bar	Price
Pete	HGAT	5
Pete	BP	4
Joe	TGAB	6
John	HGAT	5

Drinker	Bar	Price
Pete	BO	6
John	BP	4
Sally	TGAB	6
Sally	HGAT	5

Drinker	Bar	Price
Pete	HGAT	5
Pete	BP	4
Pete	BO	6
Joe	TGAB	6

Drinker	Bar	Price
John	HGAT	5
John	BP	4
Sally	TGAB	6
Sally	HGAT	5

Drinker	Price
Pete	5
Joe	6

Drinker	Price
John	4.5
Sally	5.5

From Structured to Semistructured

```
[
  {
    _id: 1,
    name: "sue",
    age: 19,
    type: 1,
    status: "P",
    favorites: { artist: "Picasso", food: "pizza" },
    finished: [ 17, 3 ],
    badges: [ "blue", "black" ],
    points: [
      { points: 85, bonus: 20 },
      { points: 75, bonus: 10 }
    ]
  },
  {
    _id: 2,
    name: “john”,
    age: 21
  }
]
```

Key-value pair
(key → value)

Named Tuple
(Tuple-key → tuple)
(Tuple-key, attrib-key→attrib-value)

Named Array
(Array-key→array)
(Array-key, position → array-element)
(Array-key, value-list → matching-values)

Named Array of unnamed Tuples

SQL SELECT and MongoDB find()

- MongoDB is a collection of documents
- The basic query primitive

How many results to return etc.

db.collection.find(<query filter>, <projection>).<cursor modifier>

Like FROM clause, specifies the collection to use

Like WHERE clause, specifies which documents to return

Projection variables in SELECT clause

Some Simple Queries

- **Query 1**
  - SQL
    - SELECT * FROM Beers
  - MongoDB
    - db.Beers.find()
- **Query 2**
  - SQL
    - SELECT beer, price FROM Sells
  - MongoDB
    - db.Sells.find(
    - { },
    - { beer: 1, price: 1}
    -)

{ beer: 1, price: 1, _id: 0}

Adding Query Conditions

- **Query 3**
  - SQL
    - SELECT manf FROM Beers WHERE name = 'Heineken'
  - MongoDB
    - db.Beers.find({ name: "Heineken" }, { manf: 1, _id: 0 })
- **Query 4**
  - SQL
    - SELECT DISTINCT beer, price FROM Sells WHERE price > 15
  - MongoDB
    - db.Sells.distinct({price: {$gt: 15} }, {beer:1, price:1, _id:0})

Some Operators of MongoDB

Symbol	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.
$or	Joins query clauses with a logical OR.
$and	Joins query clauses with a logical AND.
$not	Inverts the effect of a query expression.
$nor	Joins query clauses with a logical NOR.

URL For MongoDB operators
https://docs.mongodb.com/manual/reference/operator/query/

Regular Expressions

- **Query 5**
  - Count the number of manufacturers whose names have the partial string “am” in it – must be case insensitive
    - db.Beers.find(name: {$regex: /am/i}).count()
- **Query 6**
  - Same, but name starts with “Am”
    - db.Beers.find(name: {$regex: /^Am/}).count()
  - Starts with “Am” ends with “corp”
    - db.Beers.count(name: {$regex: /^Am.*corp$/})

Array Operations

- **Find items which are tagged as "popular" or "organic"**
  - db.inventory.find({tags: {$in: ["popular", "organic"]})
- **Find items which are *not* tagged as "popular" nor "organic"**
  - db.inventory.find({tags: {$nin: ["popular", "organic"]})
- **Find the 2nd and 3rd elements of tags**
  - db.inventory.find({}, { tags: { $slice: [1, 2] } })

  - db.inventory.find({}, tags: {$slice: -2})
- **Find a document whose 2nd element in tags is "summer"**
  - db.inventory.find(tags.1: "summer")

{ _id: 1,
item: "bud",
qty: 10,
tags: ["popular", "summer", "Japanese"],
rating: "good" }

Compound Statements

```
db.inventory.find( {
    $and : [
        { $or : [ { price : 3.99 }, { price : 4.99 } ] },
        { $or : [ { rating : good }, { qty : { $lt : 20 } } ] }
        {item: {$ne: "Coors"}}
    ]
} )
```

{ _id: 1,
item: "bud",
qty: 10,
tags: ["popular", "summer", "Japanese"],
rating: "good",
price: 3.99 }

*SELECT * FROM inventory*
WHERE ((price = 3.99) OR (price=4.99)) AND
((rating = "good") OR (qty < 20)) AND
item != "Coors"

Queries over Nested Elements

```
_id: 1,
      points: [
         { points: 96, bonus: 20 },
         { points: 35, bonus: 10 }
      ]
```

```
_id: 2,
      points: [
         { points: 53, bonus: 20 },
         { points: 64, bonus: 12 }
      ]
```

```
_id: 3,
      points: [
         { points: 81, bonus: 8 },
         { points: 95, bonus: 20}
      ]
```

- db.users.find({ 'points.0.points': { $lte: 80 } })
- db.users.find({ 'points.points': { $lte: 80 } })
- db.users.find({ "points.points": { $lte: 81 }, "points.bonus": 20 })

MongoDB does not have adequate support to perform recursive queries over nested substructures

Information Integration

SDSC SAN DIEGO SUPERCOMPUTER CENTER

After this video you will be able to

- Explain the data integration problem
- Define integrated views and schema mapping
- Describe the impact of increasing the number of data sources
- Appreciate the need to use data compression
- Describe Record Linking, Data Exchange and Data Fusion Tasks

A Business Case (from IBM)

"Mergers and acquisitions in the past decade have increased our customer base by 200 percent. Having a single view of the customer, we're more accurately able to target and cross-sell across our brands."

Suncorp is a diversified financial services group that offers general insurance, banking, life insurance and wealth management services. With operations in Australia and New Zealand, Suncorp has over AU$95 billion in assets, more than 16,000 employees and relationships with over nine million customers. The financial services organization maintains five operating divisions, managing 14 market brands, and is supported by corporate and shared services divisions.

Suncorp-Metway Ltd wanted a single, integrated view of its customers to ensure its marketing campaigns didn't encourage internal conflict between the brands and duplication of efforts, both of which had a negative effect on the bottom line.

Deconstructing the Case – Hypothetically

Insurance Company's Partial Schema

Policies(PolicyKey, PolicyTypeKey, Agent, Conditions)
PolicySales(PolicyKey, PolicyholderKey, StartDate, TransactKey,Premium,CoveragePeriod, CoverageLimit)
Transactions(TransactKey, Date, Time, Amount, Balance)
Policyholders(PolicyHolderKey, Name, Address, City, State, ZIP)
Claims(PolicyKey, ClaimKey, TransactKey, ClaimAmount)
ClaimDescription(ClaimKey, TypeKey, ClaimantKey, ProcCode, Description)
Claimants(ClaimantKey, Name, Address, City, State, ZIP)
ClaimTypes(TypeKey, Description)
PolicyTypes(PolicyTypeKey, Name, Description)

Bank's Partial Schema

Accounts(AcctNumber, AcctType, MemberID, MemberType, TypeID, StartDate, EndDate, InterestRate, CreditLimit)
Individuals(MemberID, FName, MI, LName, SSN, Nationality, DoB, LegalStatus, FullAddress, Phone, PhoneType, Email)
Corporations(MemberID, Name, RegisteredAddress, CorporationType, Signatory1, Signatory2, DNBNumber, Phone, Email)
Transactions(TrID, AcctNum, Date, Time, TransactionType, Description, TransactionAmount, Debit/Credit, Balance, Payoff)
AccountType(TypeID, Name, Description)
TransactionTypes(Ttype, Name, Description)
Disputes(AccntNumber, DisputeID, TrID, Date, DisputeAmt, Explanation, Valid, ValidatorID)

Integrated Views

PolicySales(*PolicyKey, PolicyholderKey, StartDate, TransactKey, Premium, CoveragePeriod, CoverageLimit)*
Policyholders(*PolicyHolderKey, Name, Address, City, State, ZIP)*

Accounts(*AcctNumber, AcctType, MemberID, MemberType, TypeID, StartDate, EndDate, InterestRate, CreditLimit)*
Individuals(*MemberID, FName, MI, LName, SSN, Nationality, DoB, LegalStatus, FullAddress, Phone, PhoneType, Email)*

- **Find current customers of the insurance company who are also customers of the bank, and create this integrated view**

discountCandidates(*custID, address, policyKey, AcctNumber)*

Schema Mapping

Policyholders(*PolicyHolderKey, Name, Address, City, State, ZIP*)

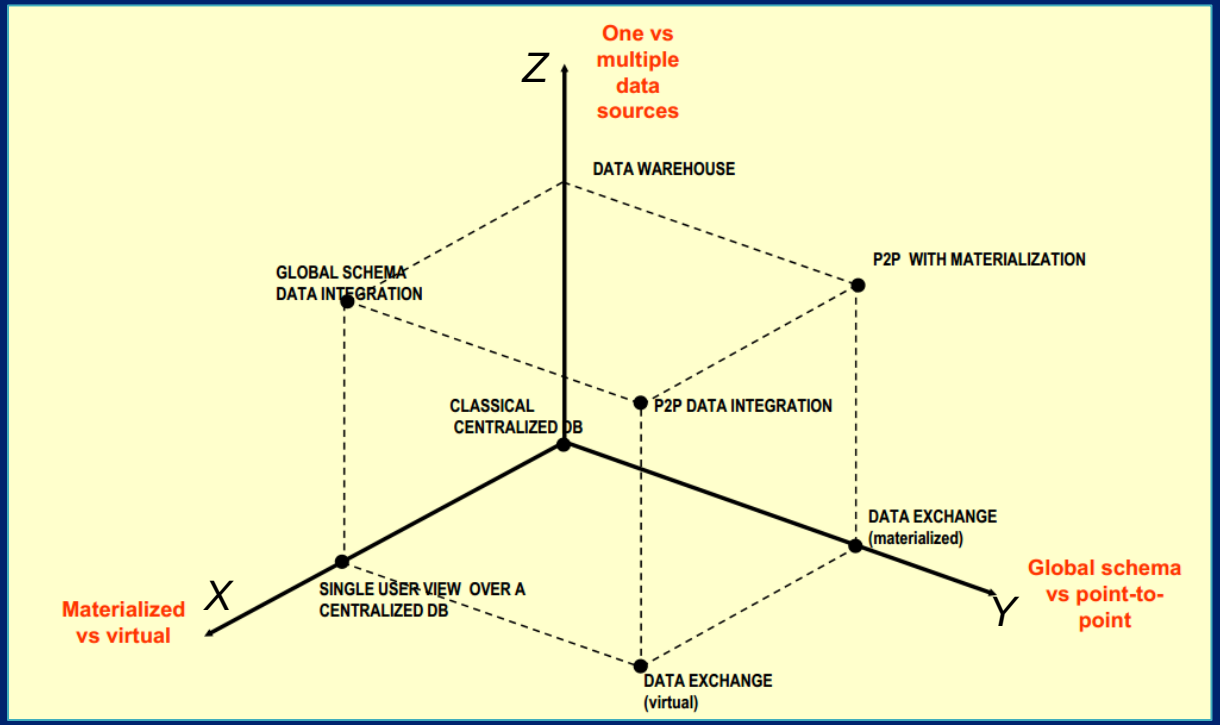
Individuals(*MemberID, FName, MI, LName, SSN, Nationality, DoB, LegalStatus, FullAddress, Phone, PhoneType, Email*)

discountCandidates(*custID, address, policyKey, AcctNumber*)

PolicySales(*PolicyKey, PolicyholderKey, StartDate, TransactKey, Premium, CoveragePeriod, CoverageLimit*)

Accounts(*AcctNumber, AcctType, MemberID, MemberType, TypeID, StartDate, EndDate, InterestRate, CreditLimit*)

Querying Integrated Data



- **Find the bank account number of a person whose policyKey is known**
  - SELECT AcctNumber FROM discountCandidates where policyKey = '4-937528734'

Record Linkage

Policyholders(*PolicyHolderKey, Name, Address, City, State, ZIP*)

Individuals(*MemberID, FName, MI, LName, SSN, Nationality, DoB, LegalStatus, FullAddress, Phone, PhoneType, Email*)

Individuals(*101, Stephen, C., Jones, 123-45-6789, US, 10/02/1983, citizen, "231 Cedar St. LA, CA 90005", 661-266-9374, landline, scjones@gmail.com*)

Individuals(*102, Elizabeth, , McFarlane, 123-54-6789, US, 06/18/1978, citizen, "4157 Elm St. LA, CA 90005", 213-266-9374, mobile, emlane@gmail.com*)

Individuals(*103, Liz, P., McFarlane-Gray, 123-92-2318, US, 06/18/1978, citizen, "231 Cedar St. LA, CA 90005", 213-702-4343, landline, emlane@gmail.com*)

Individuals(*104, Lisa, M., Brady, 423-45-6209, US, 08/09/1975, foreign-student, "231 Cedar St. LA, CA 90005", 302-266-9374, landline, scjones@gmail.com*)

Policyholders(*3-764528104, Liz, P., McFarlane-Gray, 4157 Elm St. LA, CA, 90005*)

The "Big Data" Problem

- **Many sources**
  - Hundreds of tables
  - Schema mapping problem is a combinatorial challenge
- **Pay-as-you-go model**
  - Only integrate sources that are needed when needed
- **Probabilistic Schema Mapping**

Designing Mediated Schema

- **Customers – an integrated table**
- **Candidate designs**
  - Create the customer table to include individuals and corporations – use a flag called customer_type to distinguish. In the mediated schema
    - Individual.(FN+MI+LN), PolicyHolder.Name, Corporations.Name map to Customer_Name
    - Names of two types of customers become two different attributes
    - Ind.FullAddress, Corp.RegisteredAddress, PH.(Address+City+State+Zip) map to Customer_Address
  - Issues
    - Should the DOB, Nationality, Legal Status be included in this table?

Attribute Grouping

- **How to evaluate if two attributes should go together?**
  - How similar are the attributes
    - Individual names vs. policyholder names?
    - Individual names vs. Corporation names?
  - How likely is it that two attributes would co-occur?
    - Should the DOB put in the same schema as the individual name?
    - How about DOB and the corporation name?

Customer Transactions

BankTransactions(TransactionID (TID), TransactionBeginTime(TBT), TransactionEndTime(TET), TransactionAmount(TA), Credit-Debit(CD), TransactionParty(TP), Transaction Description(TD), Balance(B), Payoff(P))

InsuranceTransactions(TransactionID (TID), TransactionDateTime (TDT), TransactionType(TT), Amount(A), TransactionDetails(TDT))

Med1({TID}, {TBT, TET, TDT} {TA+CD, A}, {TP, TD, TDT}, {TT}, {B}, {P})
Med2({TID}, {TBT}, {TET}, {TDT} {TA+CD, A}, {TP}, {TD}, {TDT}, {TT}, {B}, {P})
Med3({TID}, {TBT, TDT}, {TET, TDT} {TA+CD, A}, {TP}, {TD, TDT}, {TT}, {B}, {P})
…

Compute pairwise attribute similarity and using a threshold plus/minus an error, put similar attributes in the same cluster

For every subset of uncertain pairs create a mediated schema

Probabilistic Mediated Schema

- **Find source schemas that are consistent with a mediated schema**
  - A source schema is consistent with a mediated schema if two different attributes of the source schema do not occur in a cluster
    - Med3({TID}, {TBT, TDT}, {TET, TDT} {TA+CD, A}, {TP}, {TD, TDT}, {TT}, {B}, {P}) is better than
    - Med1({TID}, {TBT, TET, TDT} {TA+CD, A}, {TP, TD, TDT}, {TT}, {B}, {P}) with respect to BankTransactions
- **Choose the k best mediated schema**

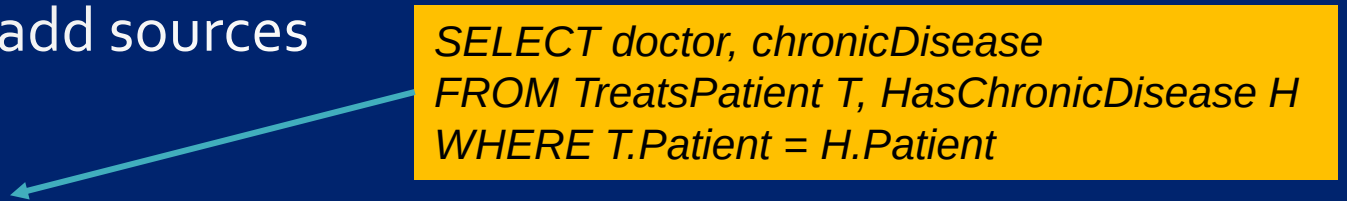
Pause

A Data Integration Scenario

- **4 data sources each with one relation**
  - S1: Treats(Doctor, ChronicDisease)
  - S2: Discharges(Doctor, Patient, Clinic)
  - S3: Treats(Doctor, ChronicDisease)
  - S4: Surgeons(SurgeonName)
- **Target schema**
  - TreatsPatient(Doctor, Patient)
  - HasChronicDisease(Patient, ChronicDisease)
  - DischargesPatientsFromClinic(Doctor, Patient, Clinic)
  - Doctors(DoctorName)
  - Surgeons(SurgeonName)

Example Schema Mapping

- **Local-as-View (LAV) mapping**
  - Mapping source schemas to target schema
  - Easier to add sources



S1.Treats(d, s)→TreatsPatient(d, p) AND HasChronicDisease(p,s)
S2.Discharges(d, p, c)→DischargesPatientFromClinic(d, p, c)
S3.Treats(d,s)→TreatsPatient(d,p) AND HasChronicDisease(p,s) AND Doctors(d)
S4.Surgeons(d)→Surgeons(d)

Query Answering

- **Query**
  - Which doctors are responsible for discharging patients?
  - SELECT DoctorName
  - FROM Doctors D1, DischargesPatientsFromClinic D2
  - WHERE D1.DoctorName = D2.DoctorName
- **Query reformulation**
  - Automatically transform query against the target schema to the simplest query against source schemas
- **Ideal Answer**
  - SELECT Doctor
  - FROM S3.Treats T, S2.Discharges D
  - WHERE T.Doctor = D.Doctor

Integration of Public Health Infrastructure

Washington DC Disease Surveillance System (WADDS)

CATEGORY	SOLUTION	DESCRIPTION
Data Exchange	Integration hub with HL7 messaging	All internal and external data moves through commercial integration hub that transforms HL7 V2 data into a consistent HL7 V3 representation.
Terminology	SNOMED LOINC	Implemented standard concept terminologies SNOMED and LOINC for coding of clinical and lab data.
Conceptual	RIM-based integrated data repository	A centralized, commercial data repository was natively designed on the HL7 RIM to normalize clinical data from disparate sources. Implemented a data quality algorithm to manage patient matching and identify duplicate records.
Architecture	PHIN architecture	Developed architecture consistent with the CDC's Public Health Information Network requirements.

*Used to enable interoperability between existing hospital and lab systems and WADSS.

Reference Information Model

Data Exchange

*Health Level-7 or **HL7** refers to a set of international standards for transfer of clinical and administrative data between software applications used by various healthcare providers.*

```
<!DOCTYPE ADT_A03 SYSTEM "hl7_v231.dtd">
<ADT_A03>
<MSH>
        <MSH.1>|</MSH.1>
        <MSH.2>^~\&amp;</MSH.2>
        <MSH.3><HD.1>LAB</HD.1></MSH.3>
        <MSH.4><HD.1>767543</HD.1></MSH.4>
        <MSH.5><HD.1>ADT</HD.1></MSH.5>
        <MSH.6><HD.1>767543</HD.1></MSH.6>
        <MSH.7>19900314130405</MSH.7>
        <MSH.9>
                <CM_MSG_TYPE.1>ADT</CM_MSG_TYPE.1>
                <CM_MSG_TYPE.2>A04</CM_MSG_TYPE.2>
        </MSH.9>
        <MSH.10>XX3657</MSH.10>
        <MSH.11><PT.1>P</PT.1></MSH.11>
        <MSH.12><VID.1>2.3.1</VID.1></MSH.12>
</MSH>
<EVN>
        <EVN.1>A01</EVN.1>
        <EVN.2>19980327101314</EVN.2>
        <EVN.3>19980327095000</EVN.3>
        <EVN.4>I</EVN.4>
        <EVN.6>19980327095000</EVN.6>
</EVN>
<PID>
        <PID.1>1</PID.1>
        <PID.3.LST>
                <PID.3><CX.1>123456789ABCDEF</CX.1></PID.3>
```

HL-7

```
MSH|^~\&|LAB|767543|ADT|767543|19900314130405||ADT^A04|XX3657|P|2.3.1<CR>
EVN|A01|19980327101314|19980327095000|I||19980327095000<CR>
PID|1||123456789ABCDEF|123456789ABCDEF|PATIENT^BOB^S||19590520|M||
    612345 MAIN STREET^^ANYTOWN^CA^91234||714-555-1212|
    714-555-1212||||123456789ABCDEF|||U<CR>
PD1||||WELBY<CR>
PV1|1|I|0||NEW|||SPOCK<CR>
```

- Find all prescriptions and lab reports of patient #19590520 containing serum protein, along with age-specific normal values between 1/1/2012 and 9/1/2015
  - The patient went to three different clinics and four different labs in this period
  - The doctor's own office uses a relational database for EHR

Data Exchange

- Given a source database with a finite number of relations, a set of schema mappings, and a set of constraints that the target schema must satisfy, the data exchange problem is to find a finite target database such that both the schema mappings and the target constraints are satisfied.

Using Codebooks

- Logical Observation Identifiers Names and Codes (LOINC) is a database and universal standard for identifying medical laboratory observations.

2. COMPONENT	Text	255	First major axis-component or analyte
3. PROPERTY	Text	30	Second major axis-property observed (e.g., mass vs. substance)
4. TIME_ASPCT	Text	15	Third major axis-timing of the measurement (e.g., point in time vs 24 hours)
5. SYSTEM	Text	100	Fourth major axis-type of specimen or system (e.g., serum vs urine)
6. SCALE_TYP	Text	30	Fifth major axis-scale of measurement (e.g., qualitative vs. quantitative)
7. METHOD_TYP	Text	50	Sixth major axis-method of measurement
8. CLASS	Text	20	An arbitrary classification of the terms for grouping related observations together. The current classifications are listed in Table 32. We present the database sorted by the class field within class type (see field 23). Users of the database should feel free to re-sort the database in any way they find useful, and/or to add their own classifying fields to the database. The content of the laboratory test subclasses should be obvious from the subclass name.

BP.ATOM	Blood pressure atomic
BP.CENT.MOLEC	Blood pressure central molecular
BP.MOLEC	Blood pressure molecular
BP.PSTN.MOLEC	Blood pressure positional molecular
BP.TIMED.MOLEC	Blood pressure timed molecular
BP.VENOUS.MOLEC	Blood pressure venous molecular
CARD.RISK	Cardiac Risk Scales Framingham
CARD.US	Cardiac ultrasound (was US.ECHO)
CARDIO-PULM	Cardiopulmonary
CLIN	Clinical NEC (not elsewhere classified)

MOLPATH.DELDUP	Gene deletions or duplications
MOLPATH.INV	Gene inversion
MOLPATH.MISC	Gene miscellaneous
MOLPATH.MUT	Gene mutation
MOLPATH.REARRANGE	Gene rearrangement
MOLPATH.TRINUC	Gene trinucleotide repeats
MOLPATH.TRISOMY	Gene chromosome trisomy
MOLPATH.TRNLOC	Gene translocation

Using Compressed Data

- **Compression**
  - Encoded representation of data so that it uses less space
  - Dictionary encoding

Record #	Patient ID	Date	Test Code	Test Result
1	100	1/1/2012	SE-AC	14.5
2	502	1/1/2012	BP-S	123
3	301	1/2/2012	HAC	5.8
4	502	1/1/2012	BP-D	91
…	…	…	…	…
…	…	…	…	…
10M	1274	7/20/2016	SE-AC	13.8

Using Compressed Data

- **Compression**
  - Encoded representation of data so that it uses less space
  - Dictionary encoding

Record #	Patient ID	Date	Test Code	Test Result
1	100	1/1/2012	32	14.5
2	502	1/1/2012	125	123
3	301	1/2/2012	174	5.8
4	502	1/1/2012	126	91
…	…	…	…	…
…	…	…	…	…
10M	1274	7/20/2016	32	13.8

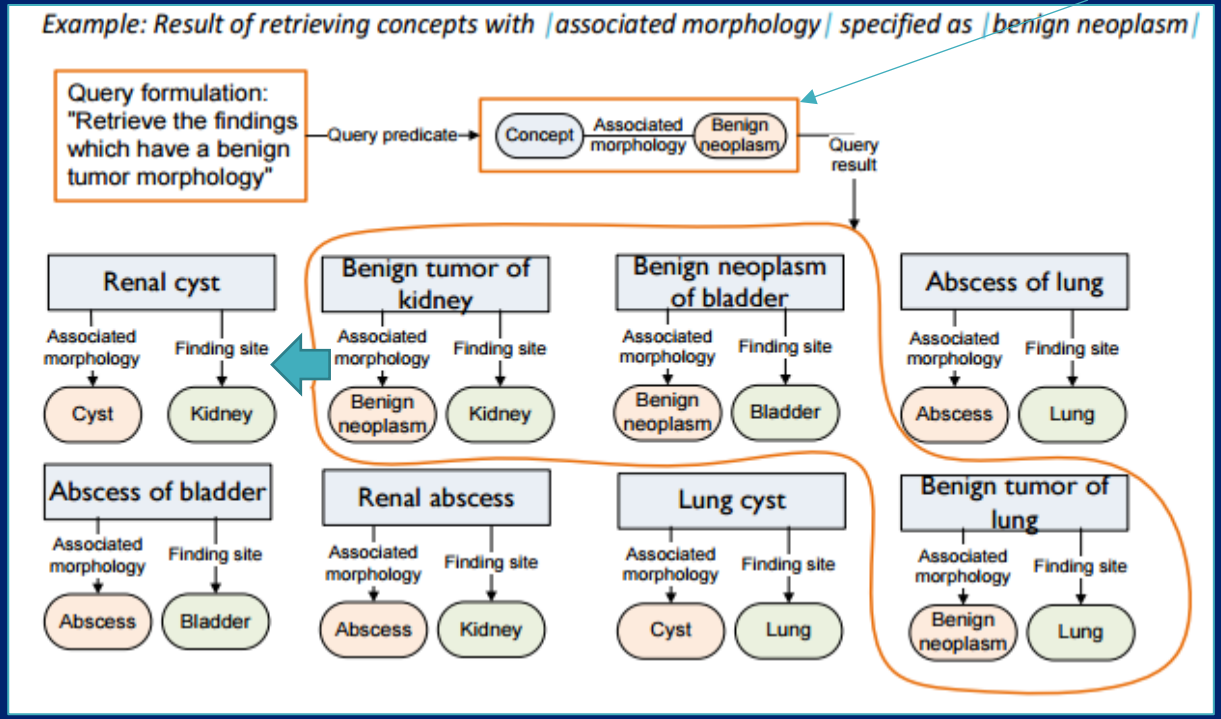
Dictionary

Orig. Test Code	Encoded Test Code
SE-AC	32
BP-S	125
HAC	174
BP-D	126
…	…
…	…
SE-AC	32

Data compression is an important technology for big data.

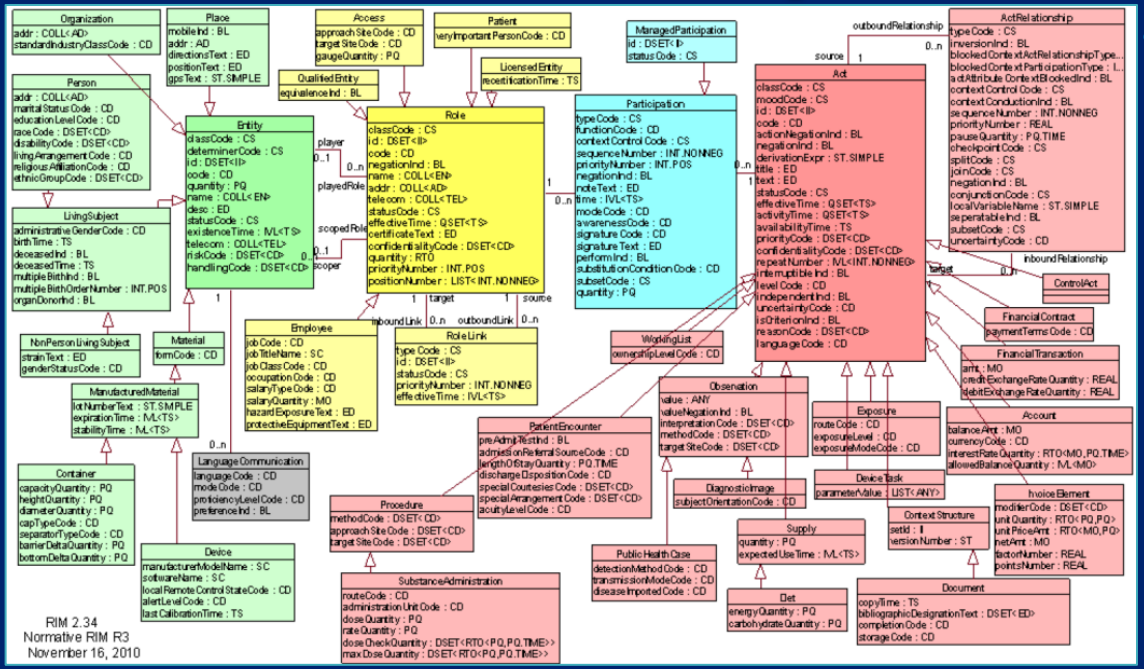
Ontological Data

Ontology queries are graph queries



- Ontology
  - A set of terms of a domain
  - Relationships between the terms
- SNOMED
  - A medical ontology used for clinical data

The Takeaway Points



- **Integration across multiple data models**
  - Global Schema – RIM
  - Data Exchange
    - Format conversions
    - Constraints
    - Compressed Data
    - Model transformation
    - Query transformation

Pause

Integration for Multichannel Customer Analytics

- **Customer analytics**
  - processes and technologies that give organizations the customer insight necessary to deliver offers that are anticipated, relevant and timely
- **Questions one would like to ask**
  - Is our product launch going well?
  - Is there an emerging product issue?
  - Where should the product team focus its development dollars?
  - Are there more effective methods for positioning current products?
  - Which services have the best chance of surviving a turbulent market?
  - Is there a product defect in the market?

Data Fusion

- **Data sources**
- **Data Items**
  - A product
  - A part of a product
  - A feature of a product
  - The utility of a product feature
  - …

value

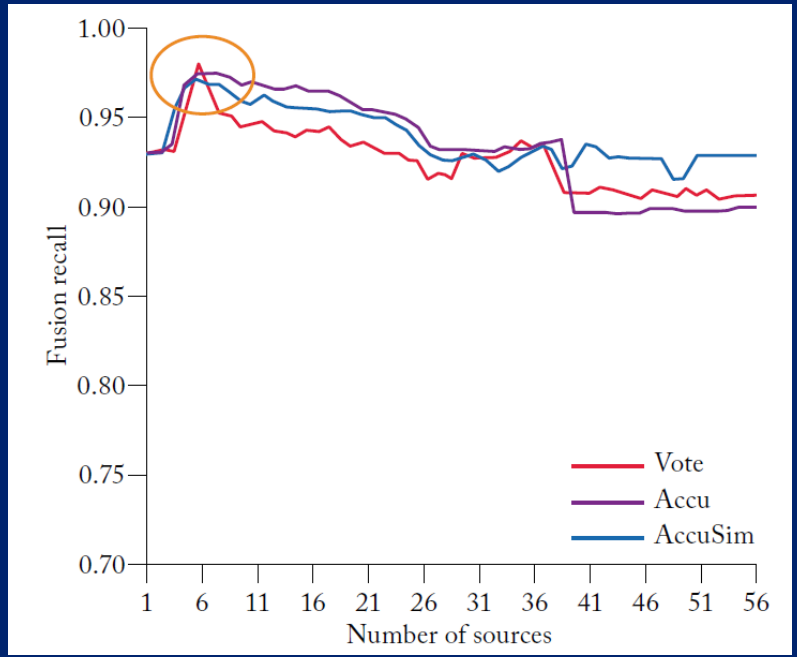
- **Using data from a subset of sources find the true value or a true value distribution of a data item**
- **Assemble all such values for the real-world entity represented by the data items**

Too Many Sources

- **Too many sources = too many values**
- **Voting to select the "right" value**
  - Simple voting can be problematic – Veracity problem
    - Source Reliability
    - Copy Detection
  - Statistical techniques to estimate
    - Trustworthiness of sources
    - Bias introduced by copies
    - True distribution of values for data items

Source Selection

- **The problem**
  - Choose only useful sources
  - Adding sources first improves integration accuracy then reduces it
- **The solution**
  - Order candidate source based on a measure of "goodness"
  - Add sources until the marginal benefit is less than the marginal cost
  - Current techniques scale well



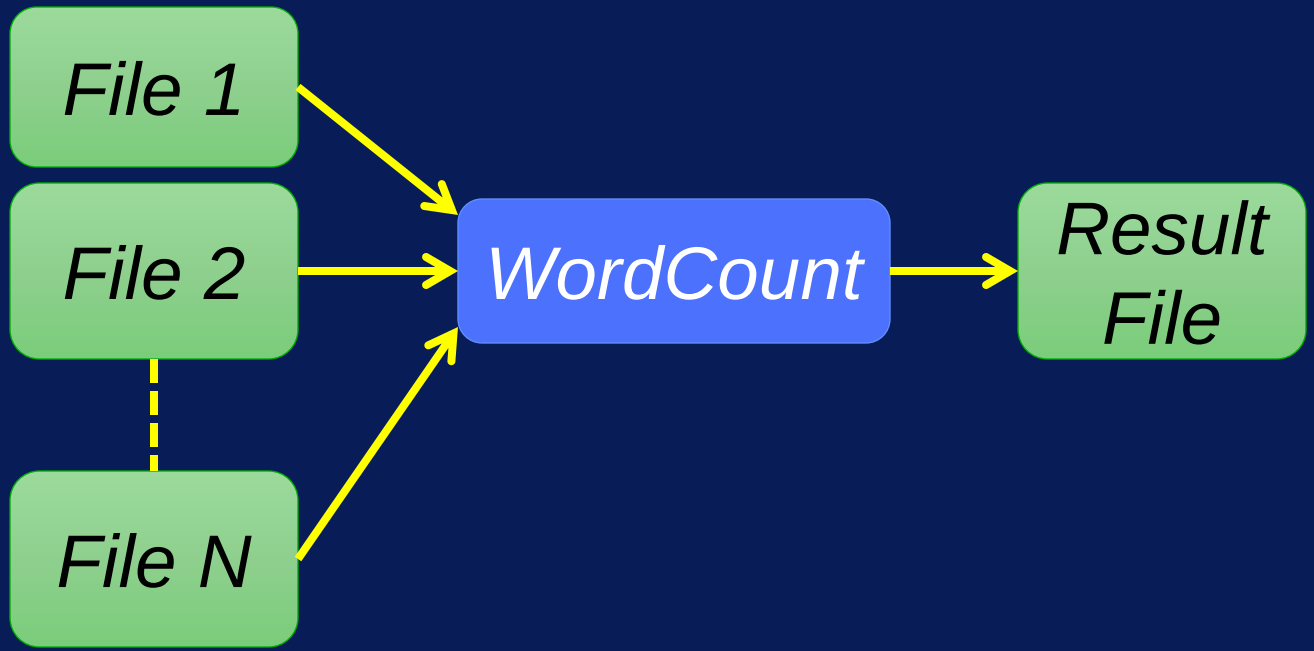
Big Data Processing Pipelines:

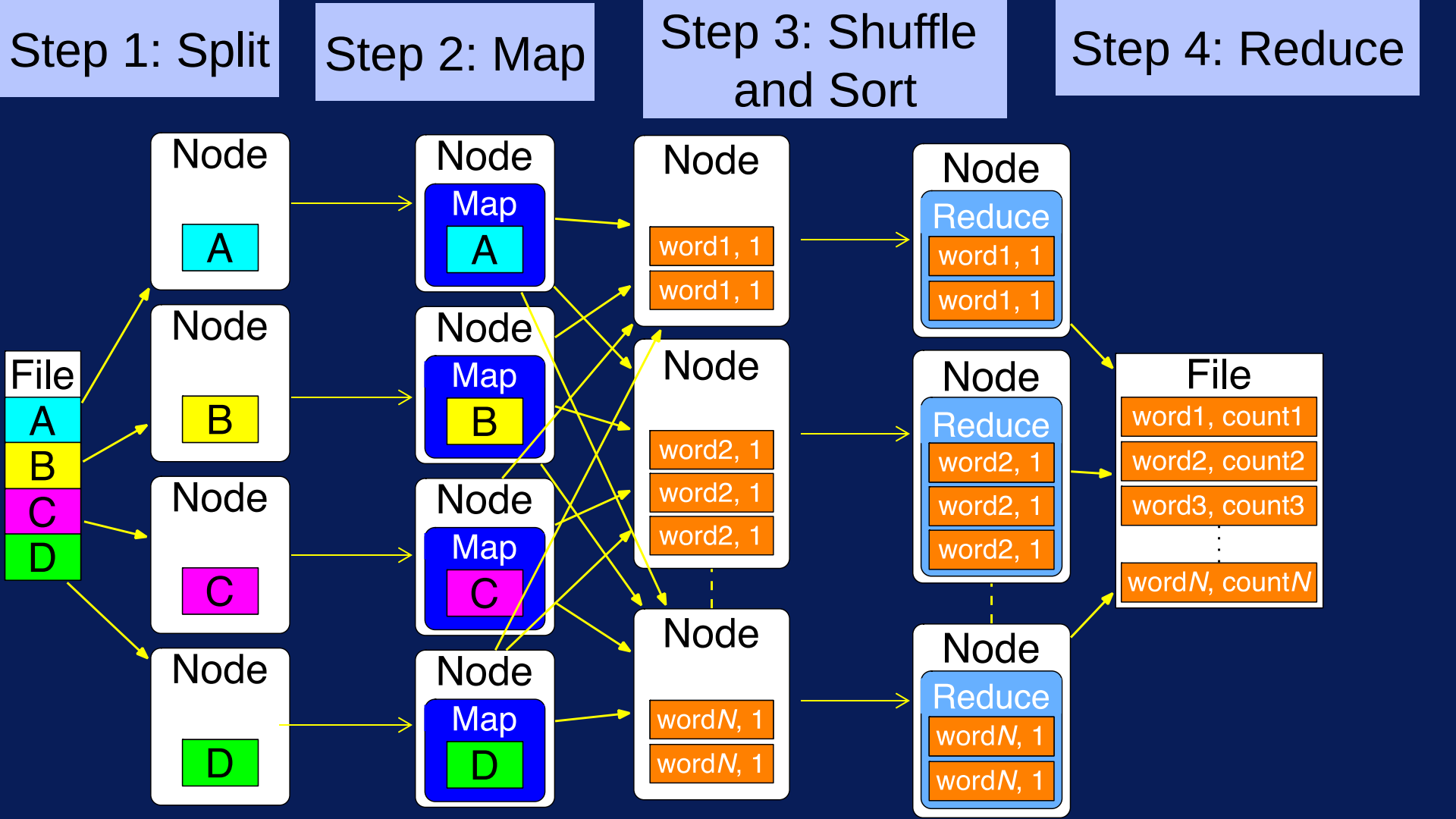
A Dataflow Approach

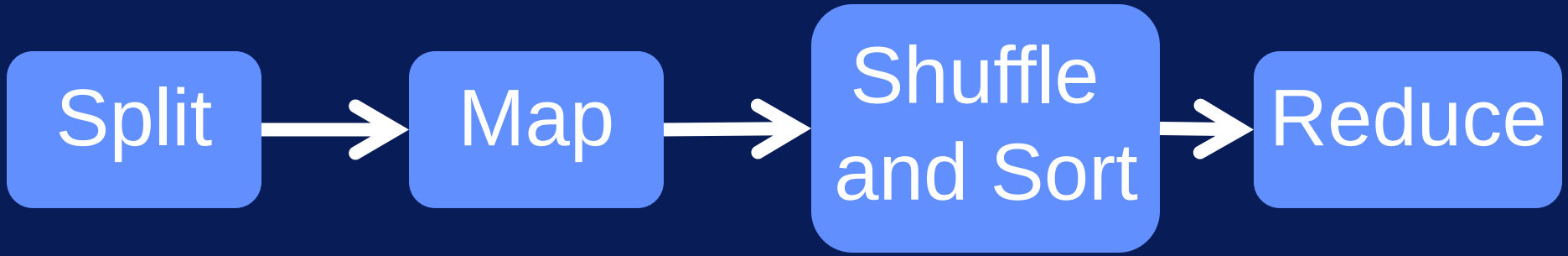
After this video you will be able to..

- Summarize what dataflow means and its role in data science
- Explain “split->do->merge” big data pipeline with examples
- Define the terms data parallel

Example MapReduce Application: WordCount

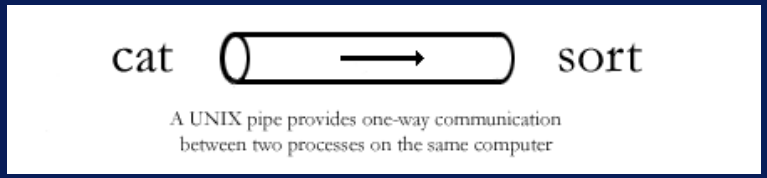



Step 1: Split
Step 2: Map
Step 3: Shuffle and Sort
Step 4: Reduce
File
A
B
C
D
Node
A
Node
B
Node
C
Node
D
Node
Map
A
Node
Map
B
Node
Map
C
Node
Map
D
Node
word1, 1
word1, 1
Node
word2, 1
word2, 1
word2, 1
Node
wordN, 1
wordN, 1
Node
Reduce
word1, 1
word1, 1
Node
Reduce
word2, 1
word2, 1
word2, 1
Node
Reduce
wordN, 1
wordN, 1
File
word1, count1
word2, count2
word3, count3
wordN, countN


Split
Map
Shuffle and Sort
Reduce

Represents a large number of applications.

Big Data Pipelines



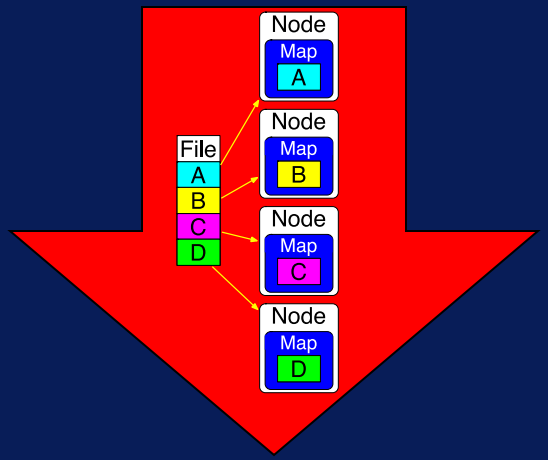
A UNIX pipe provides one-way communication
between two processes on the same computer


Map
Shuffle and Sort
Reduce

Map

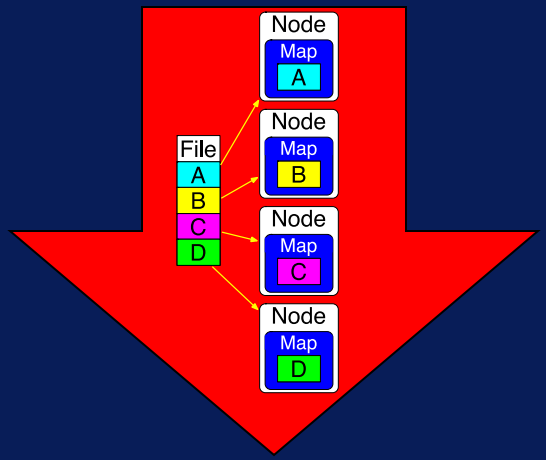
Shuffle and Sort

Reduce

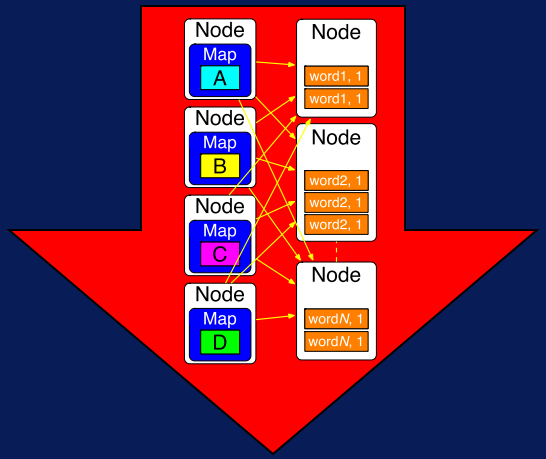


Parallelization over the input

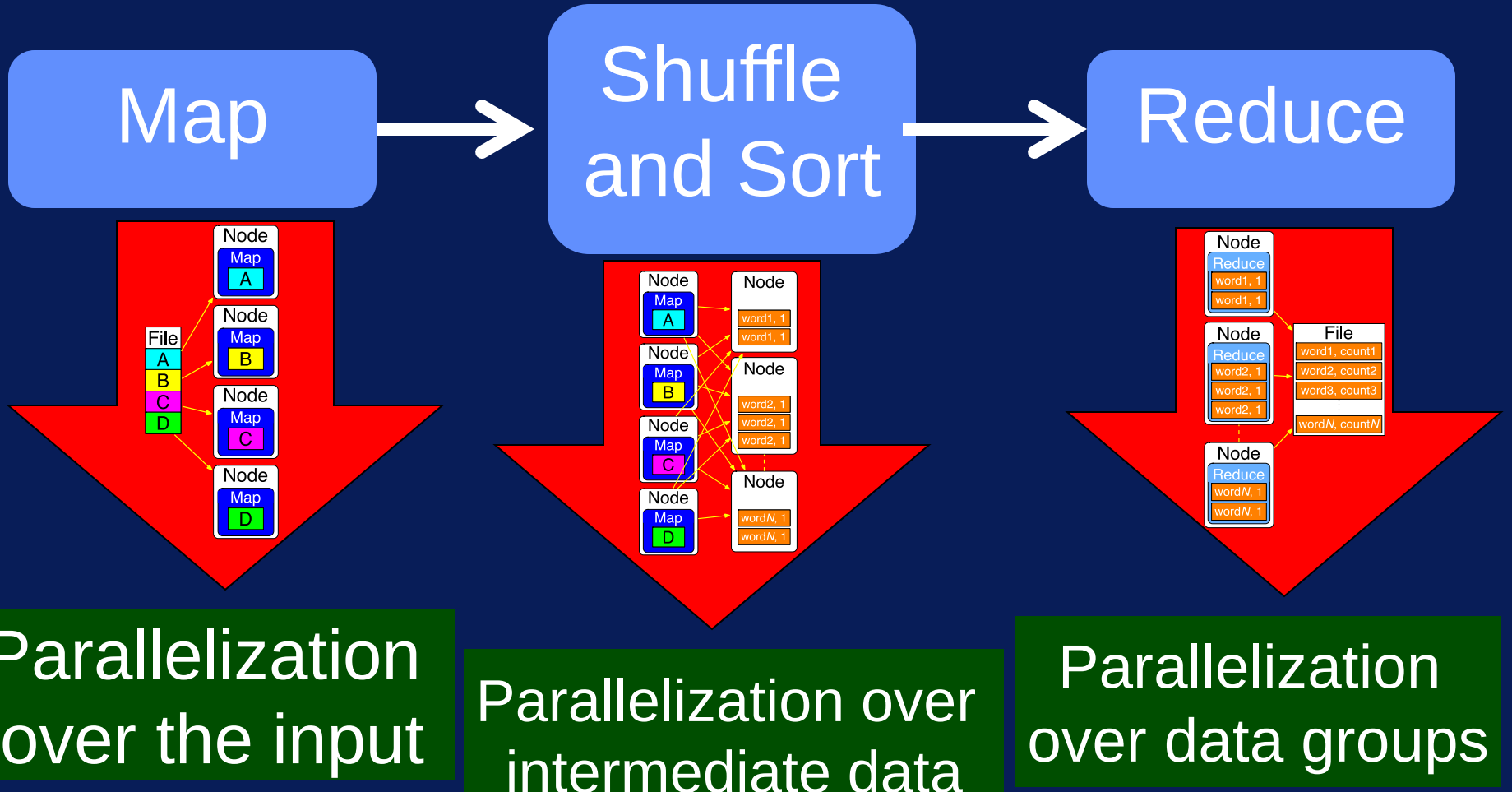
Map → Shuffle and Sort → Reduce

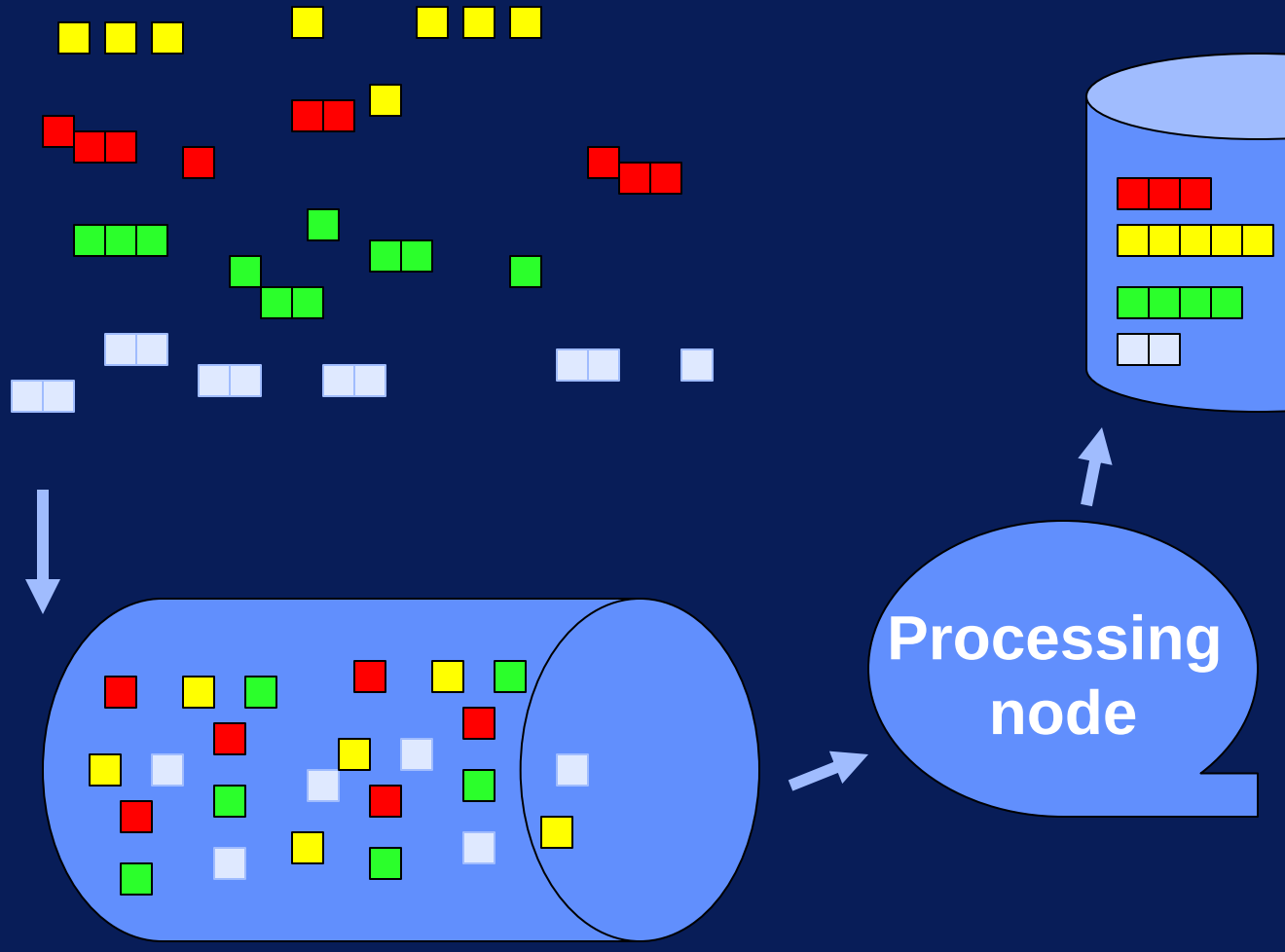


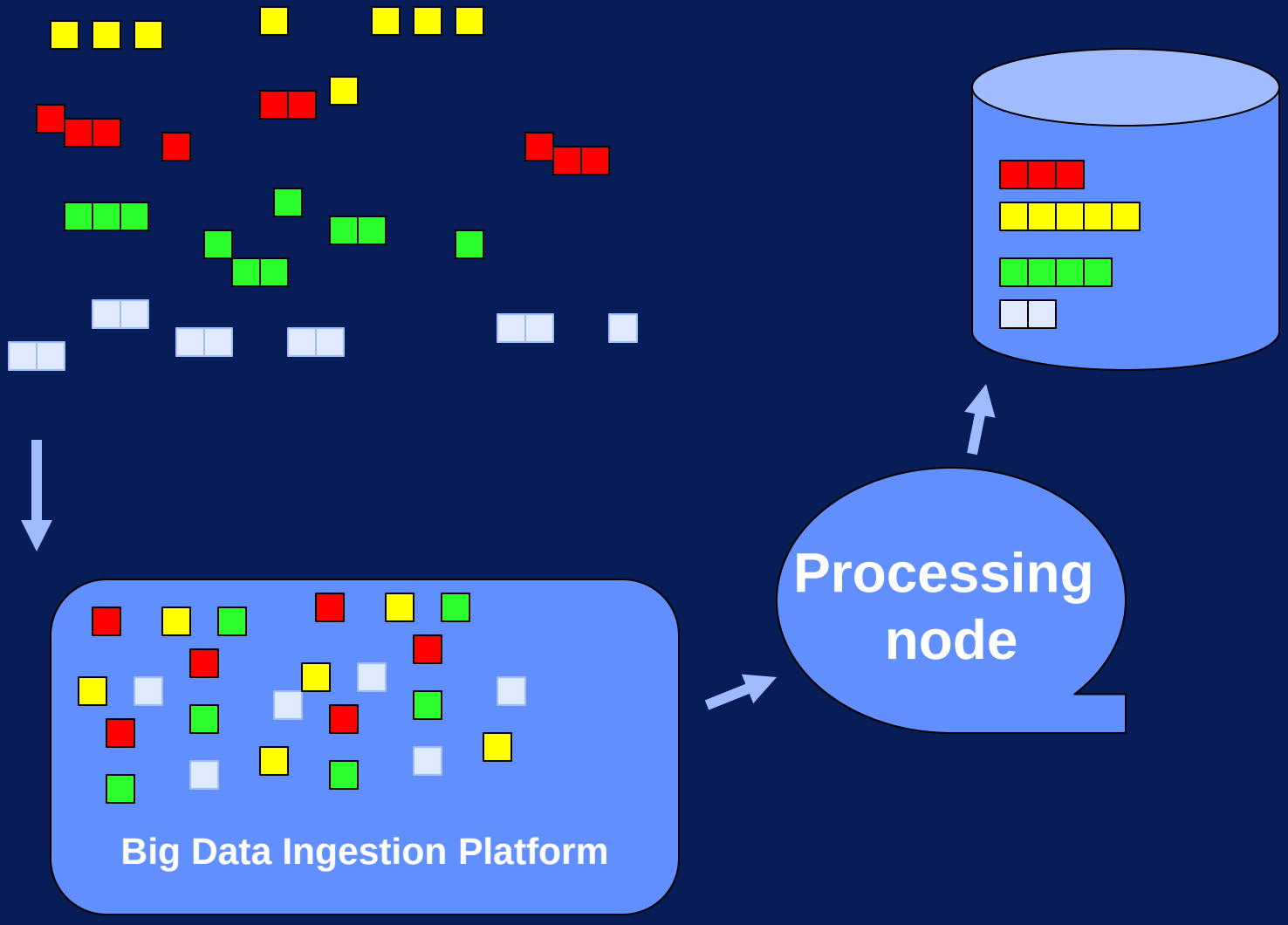
Parallelization over the input

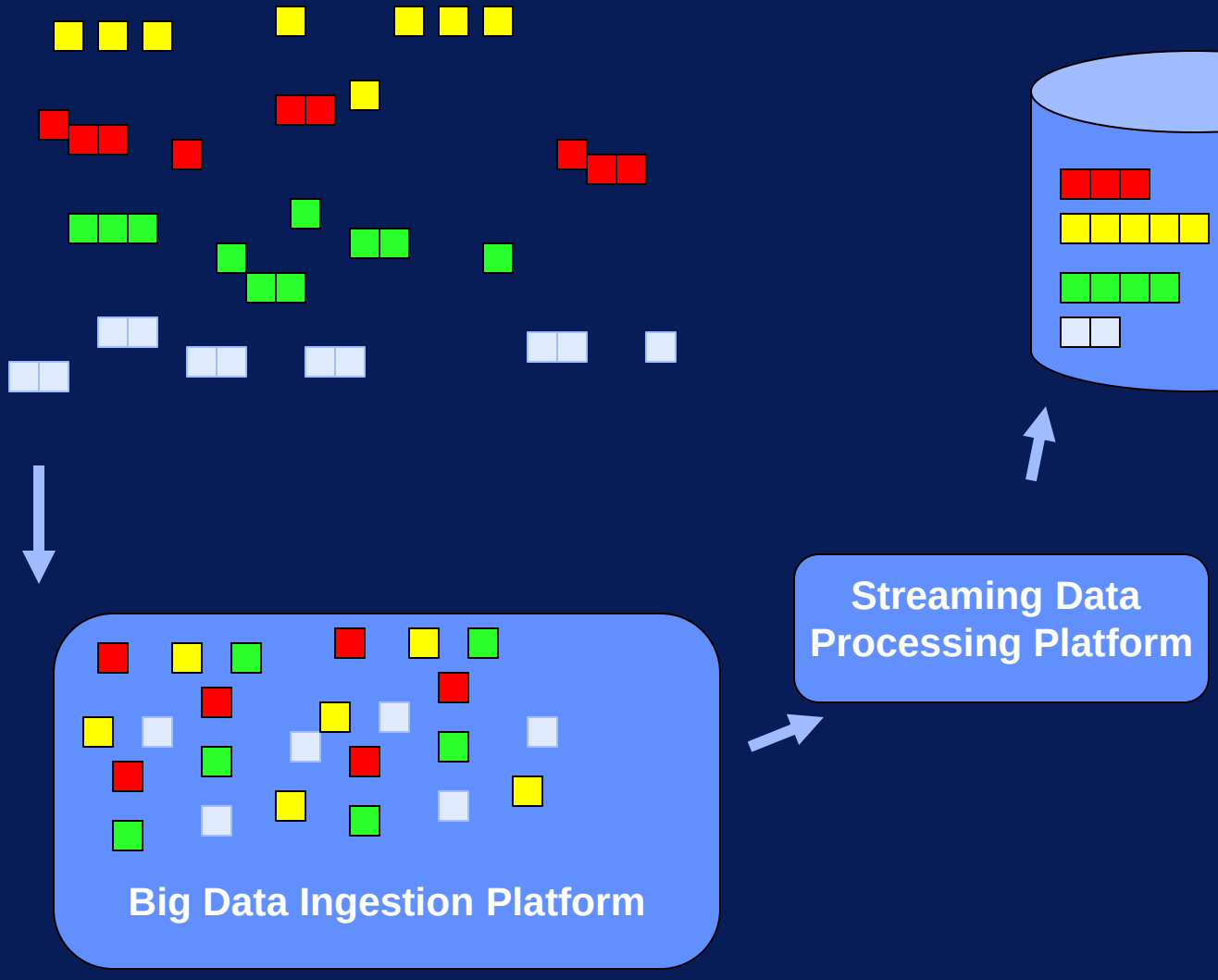


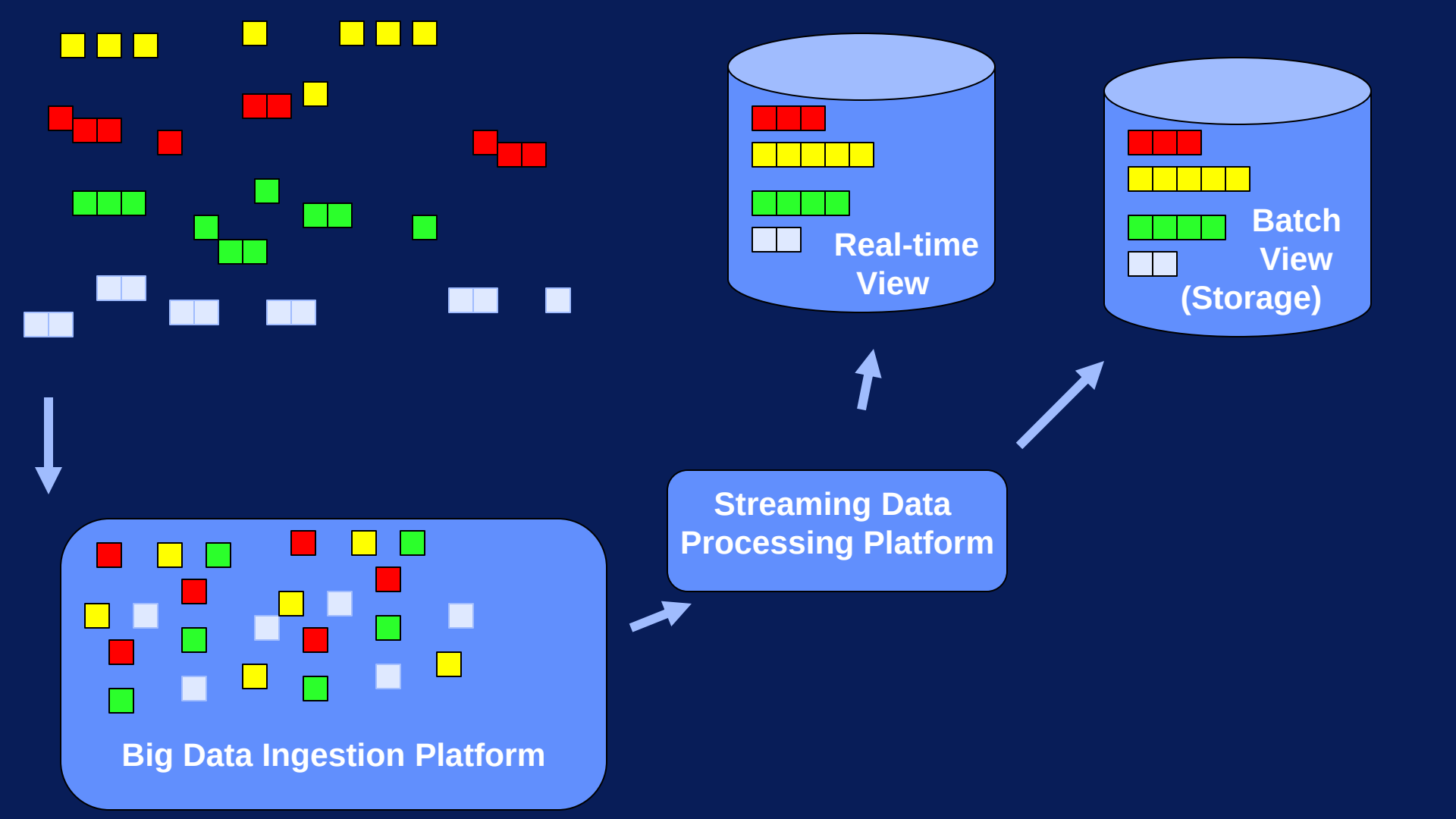
Parallelization data sorting


Map
Shuffle and Sort
Reduce
File
A
B
C
D
Node
Map
A
Node
Map
B
Node
Map
C
Node
Map
D
Node
Map
A
Node
Map
B
Node
Map
C
Node
Map
D
Node
word1, 1
word1, 1
Node
word2, 1
word2, 1
word2, 1
Node
wordN, 1
wordN, 1
Node
Reduce
word1, 1
word1, 1
Node
Reduce
word2, 1
word2, 1
word2, 1
Node
Reduce
wordN, 1
wordN, 1
File
word1, count1
word2, count2
word3, count3
wordN, countN
Parallelization over the input
Parallelization over intermediate data
Parallelization over data groups


Processing
node


Processing node
Big Data Ingestion Platform


Streaming Data Processing Platform
Big Data Ingestion Platform


Real-time
View
Batch
View
(Storage)
Streaming Data
Processing Platform
Big Data Ingestion Platform

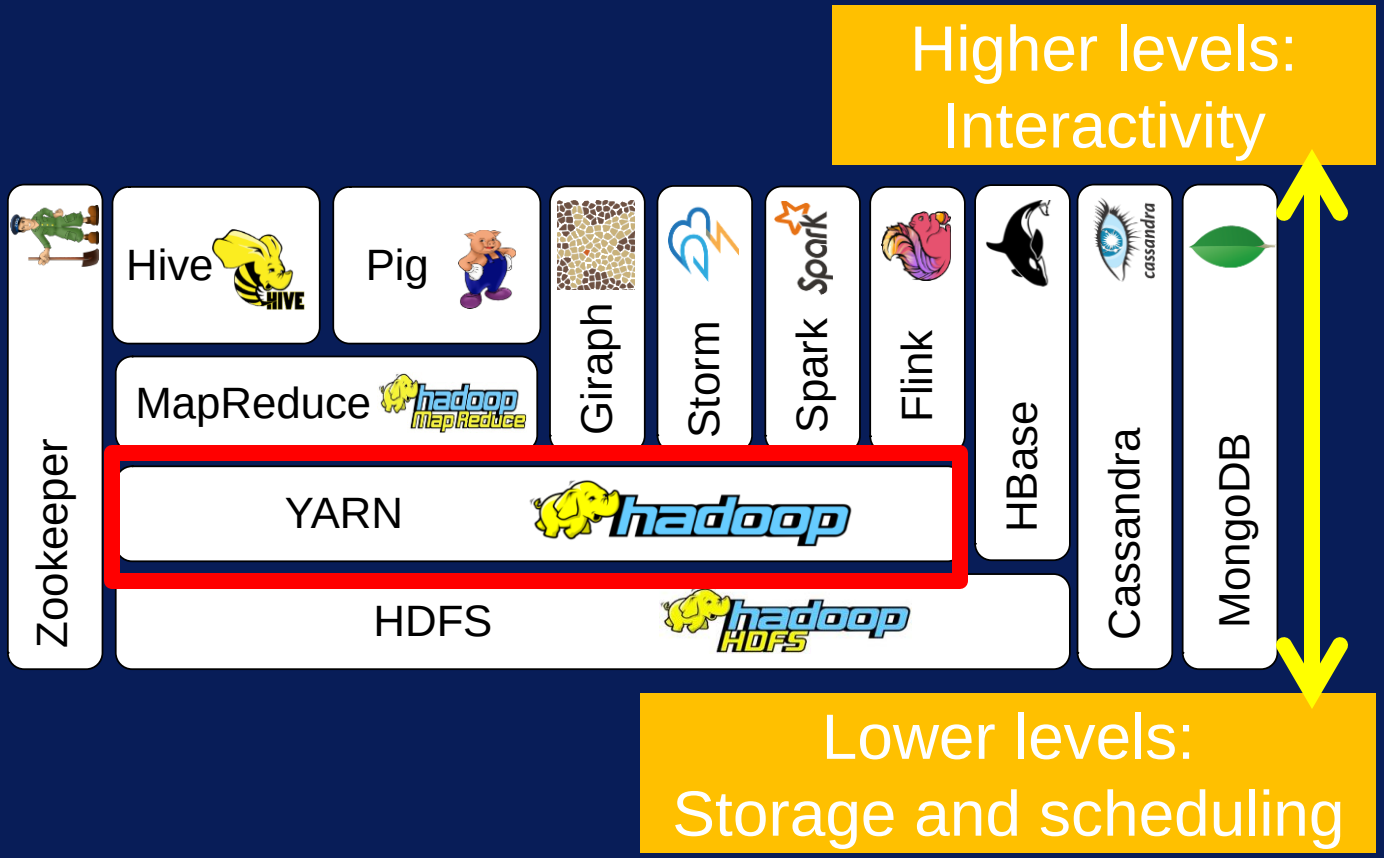

Split
Do Something
Merge

Overview of Big Data Processing Systems

After this video you will be able to..

- Recall the Hadoop Ecosystem
- Draw a layer diagram with three layers for data storage, data processing and workflow management
- Summarize an evaluation criteria for big data processing systems
- Explain the properties of Hadoop, Spark, Flink, Beam and Storm

One possible layer diagram for Hadoop tools



Another way to look at the Hadoop Ecosystem

COORDINATION AND WORKFLOW MANAGEMENT

DATA INTEGRATION AND PROCESSING

DATA MANAGEMENT AND STORAGE

Another way to look at the Hadoop Ecosystem

COORDINATION AND WORKFLOW MANAGEMENT

DATA INTEGRATION AND PROCESSING

DATA MANAGEMENT AND STORAGE

DATA MANAGEMENT AND STORAGE

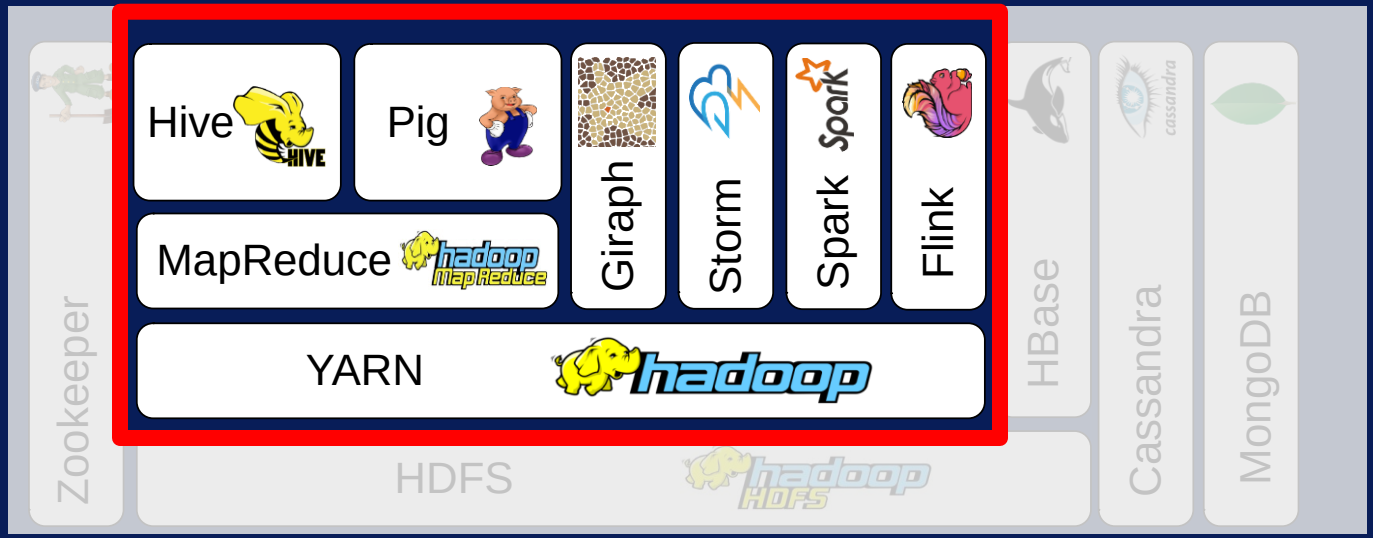
Another way to look at the Hadoop Ecosystem

COORDINATION AND WORKFLOW MANAGEMENT

DATA INTEGRATION AND PROCESSING

DATA MANAGEMENT AND STORAGE

DATA INTEGRATION AND PROCESSING



Another way to look at the Hadoop Ecosystem

COORDINATION AND WORKFLOW MANAGEMENT

DATA INTEGRATION AND PROCESSING

DATA MANAGEMENT AND STORAGE

COORDINATION AND WORKFLOW MANAGEMENT

ACQUIRE → PREPARE → ANALYZE → REPORT → ACT

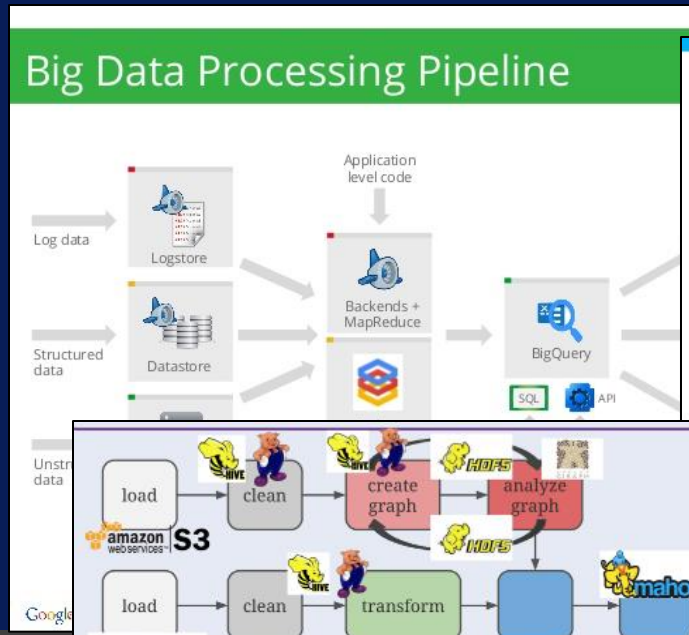
Another way to look at the Hadoop Ecosystem

COORDINATION AND WORKFLOW MANAGEMENT

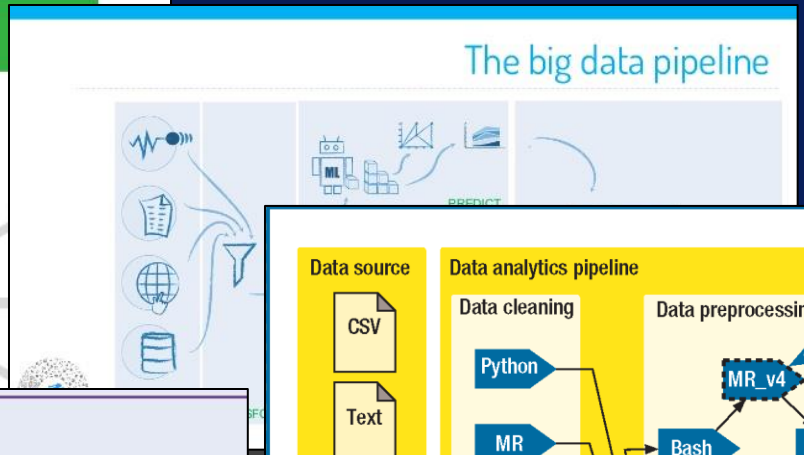
DATA INTEGRATION AND PROCESSING

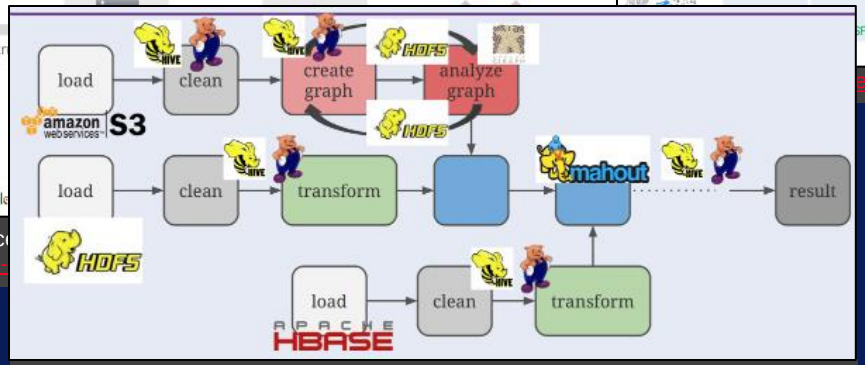
DATA MANAGEMENT AND STORAGE

Example Big Data Processing Pipelines

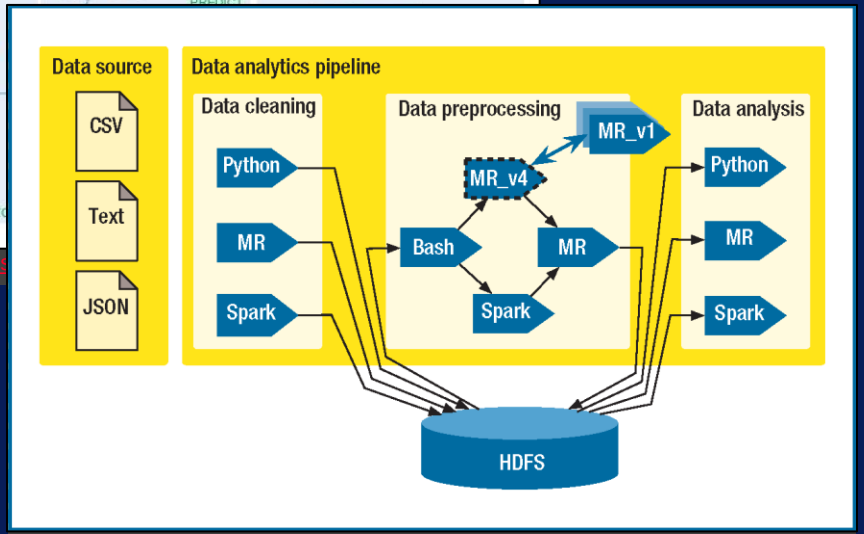


Source: cloud-



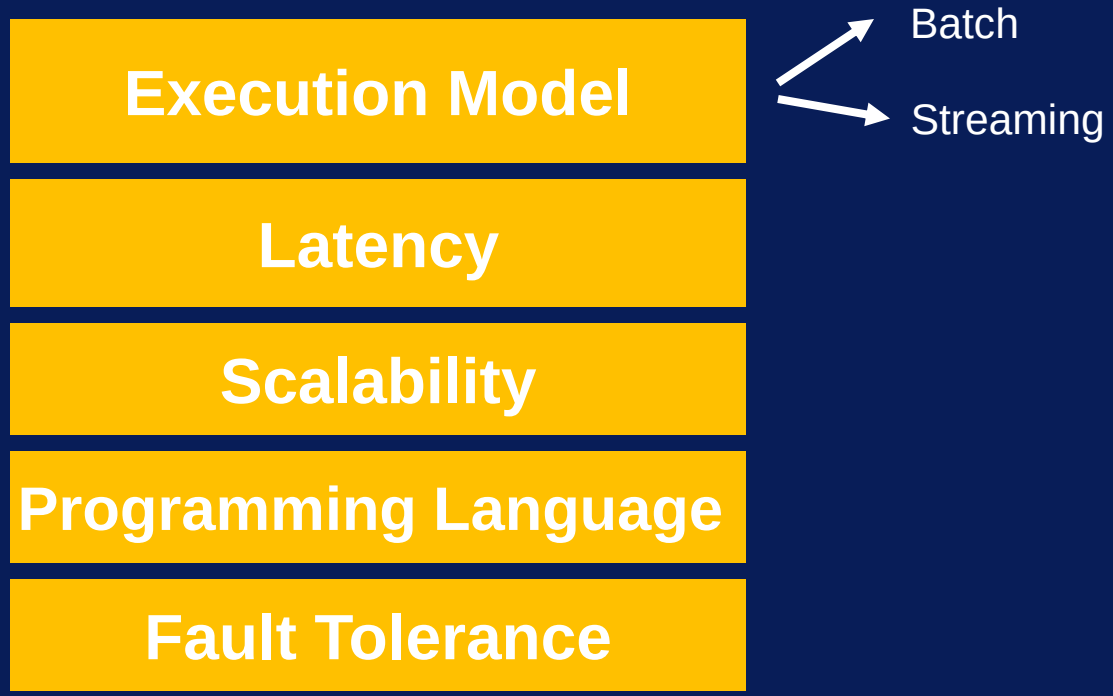


Source: https://www.mapr.com/blog/distributed-stream-and-graph-processing-apache-flink



Source: https://www.computer.org/csdl/mags/so/2016/02/mso2016020060.html

Categorization of Big Data Processing Systems



Big Data Processing Systems

MapReduce

Execution Model	Batch processing using disk storage
Latency	High-latency
Scalability	
Programming Language	Java
Fault Tolerance	Replication

Spark

Execution Model	Batch and stream processing using disk or memory storage
Latency	Low-latency for small micro-batch size
Scalability	
Programming Language	Scala, Python, Java, R
Fault Tolerance	

Flink

Execution Model	Batch and stream processing using disk or memory storage
Latency	Low-latency
Scalability	
Programming Language	Java and Scala
Fault Tolerance	

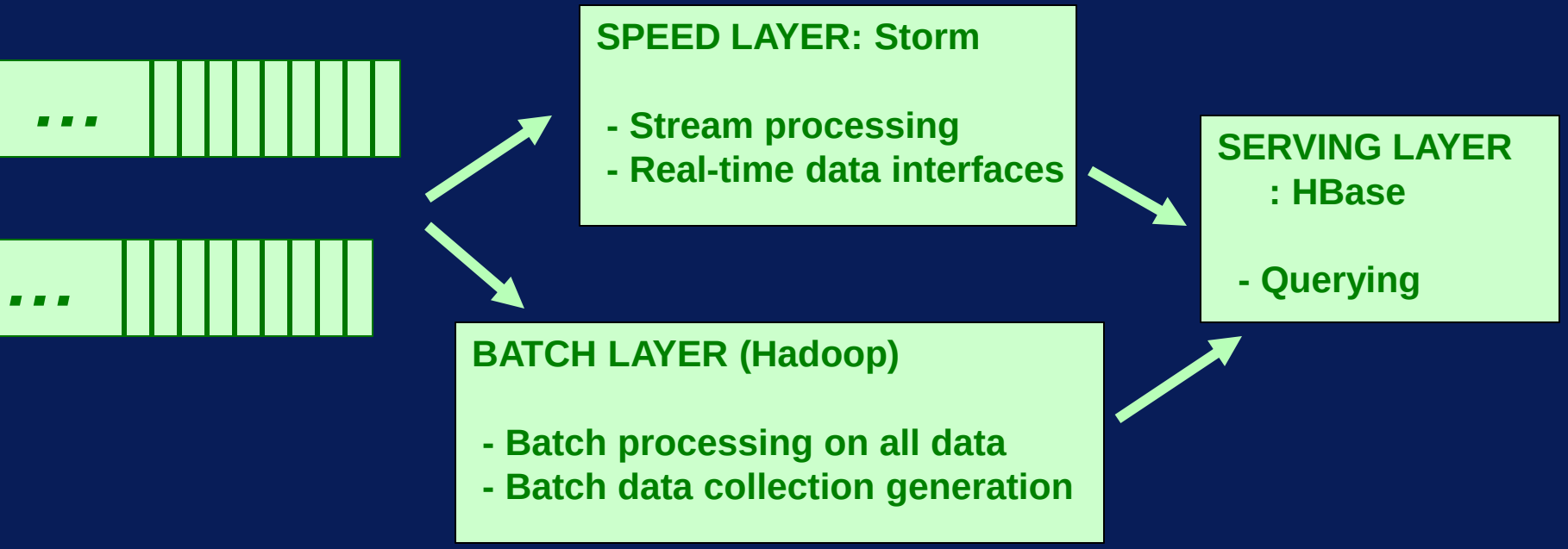
Beam

Execution Model	Batch and stream processing
Latency	Low-latency
Scalability	
Programming Language	Java and Scala
Fault Tolerance	

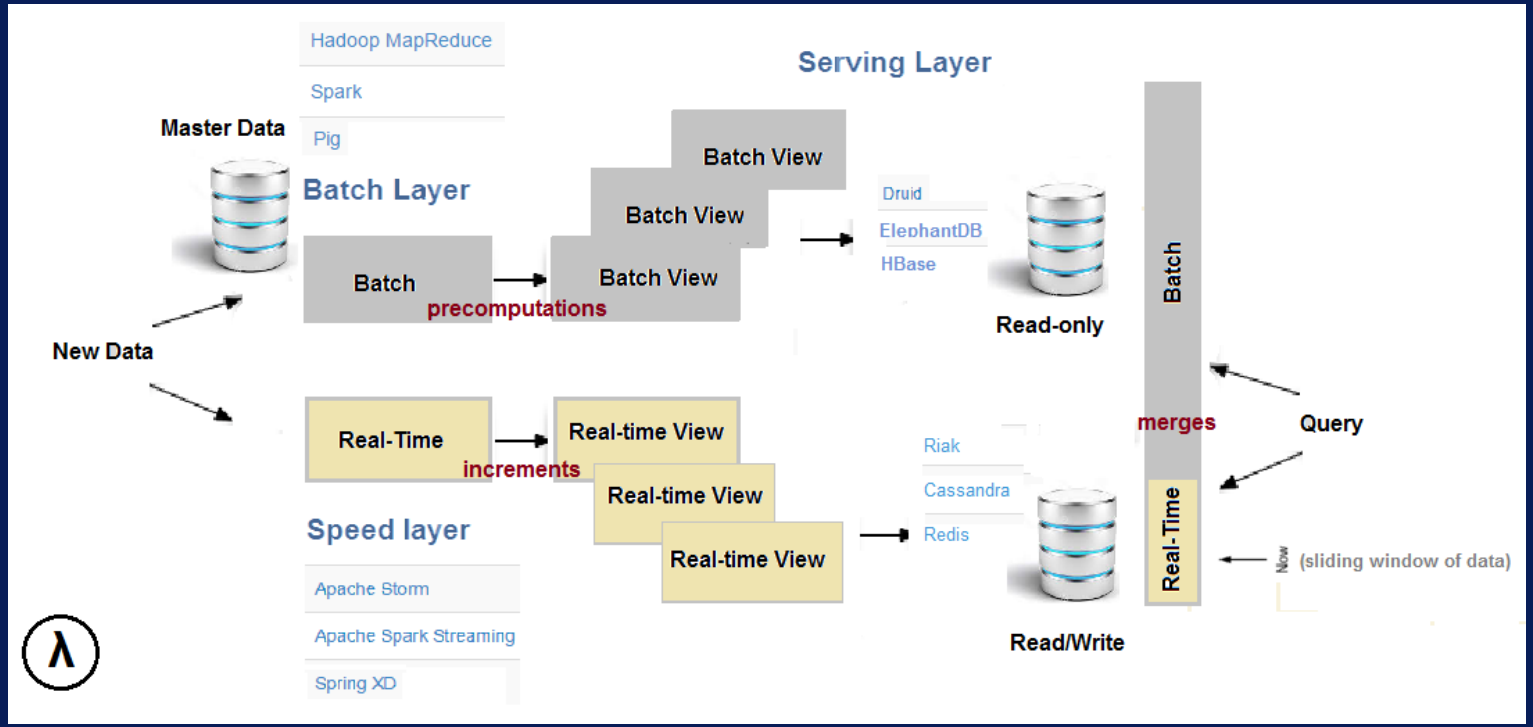
Storm

Execution Model	Stream processing
Latency	Very low-latency
Scalability	
Programming Language	Many programming languages
Fault Tolerance	

Lambda Architecture:
A Hybrid Data Processing Architecture



Lambda Architecture:
A Hybrid Data Processing Architecture



Introduction to Apache Spark

After this video you will be able to..

- List the main motivations for the development of Spark
- Draw the Spark stack as a layer diagram
- Explain the functionality of the components in the Spark stack

Why Spark?

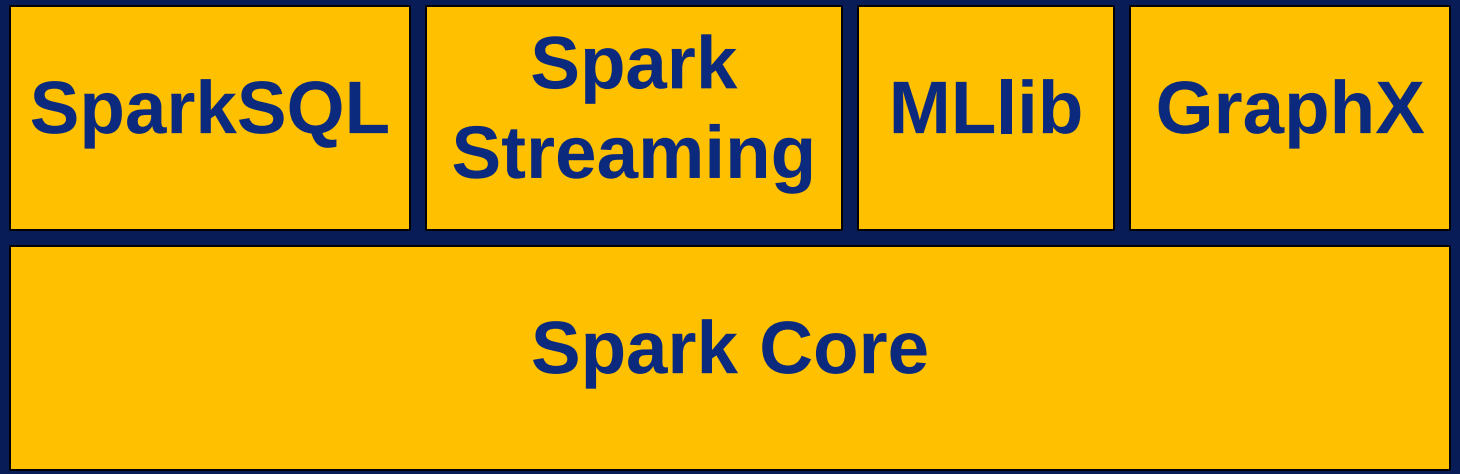
Hadoop MapReduce Shortcomings

- Only for Map and Reduce based computations
- Relies on reading data from HDFS
- Native support for Java only
- No interactive shell support
- No support for streaming

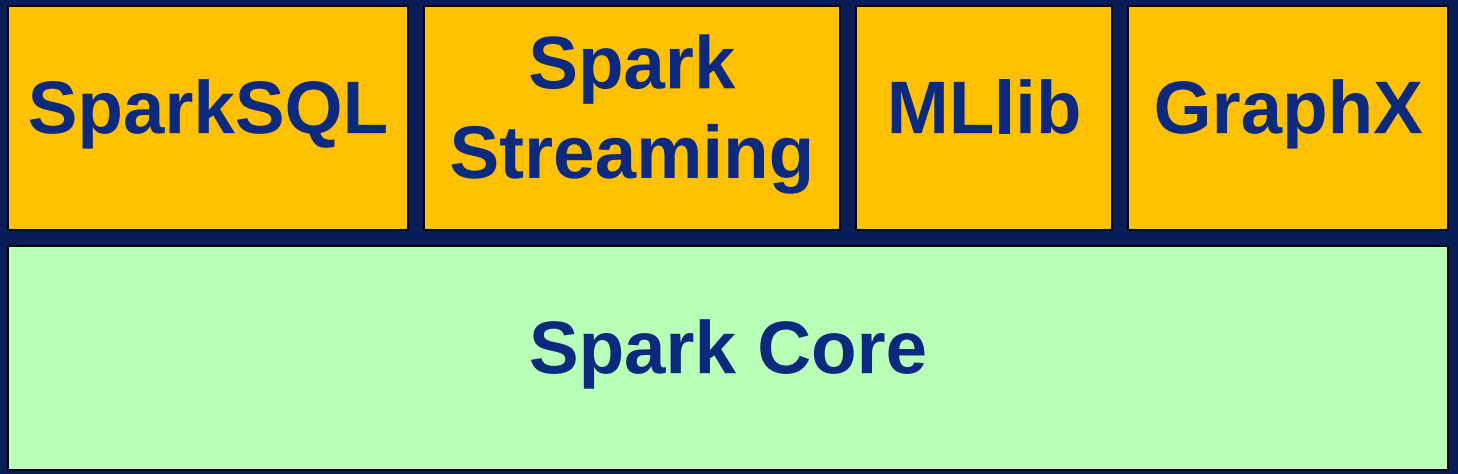
Basics of Data Analysis with Spark

- Expressive programing model
- In-memory processing
- Support for diverse workloads
- Interactive shell

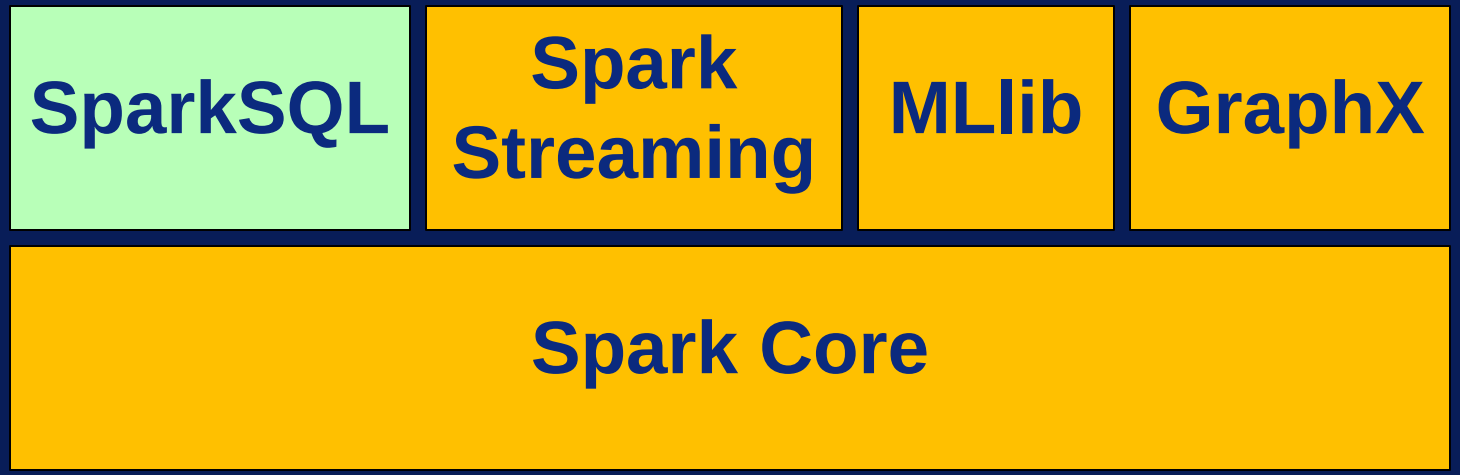
The Spark Stack



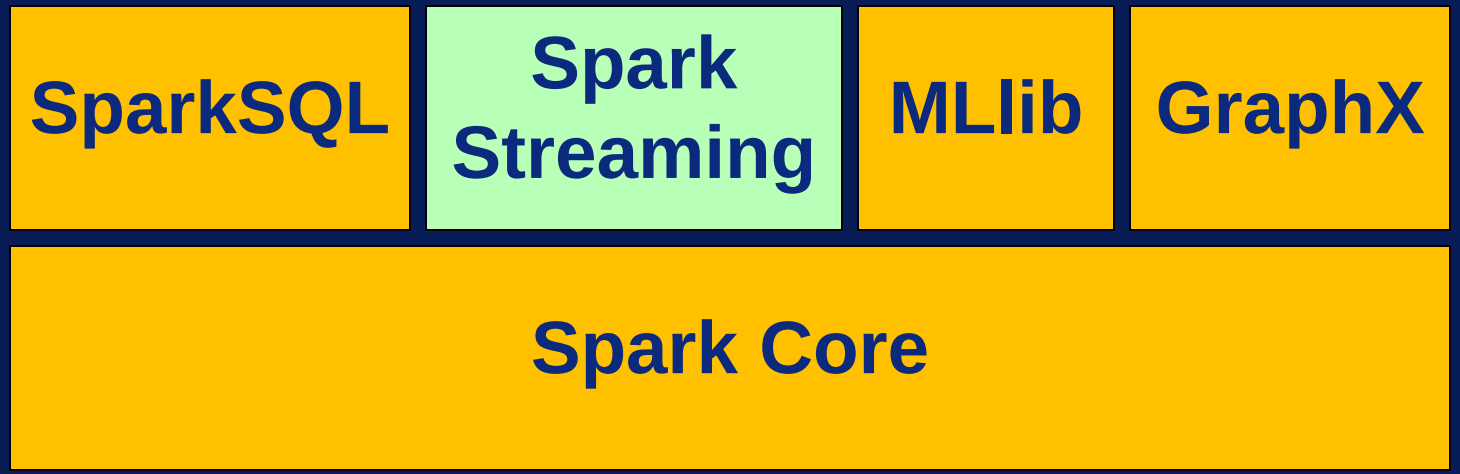
The Spark Stack



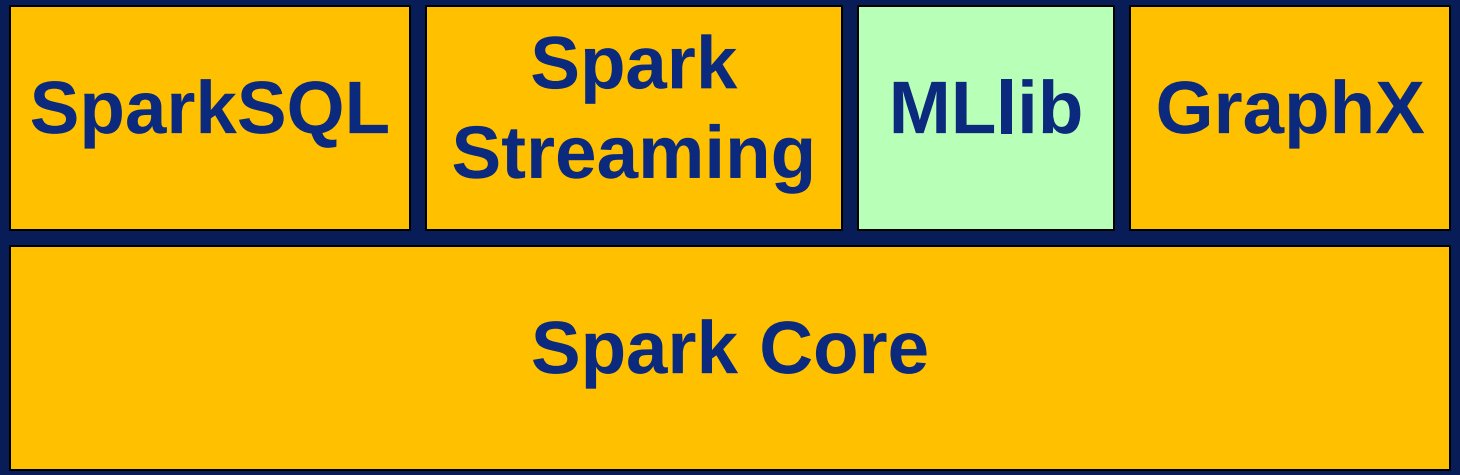
The Spark Stack



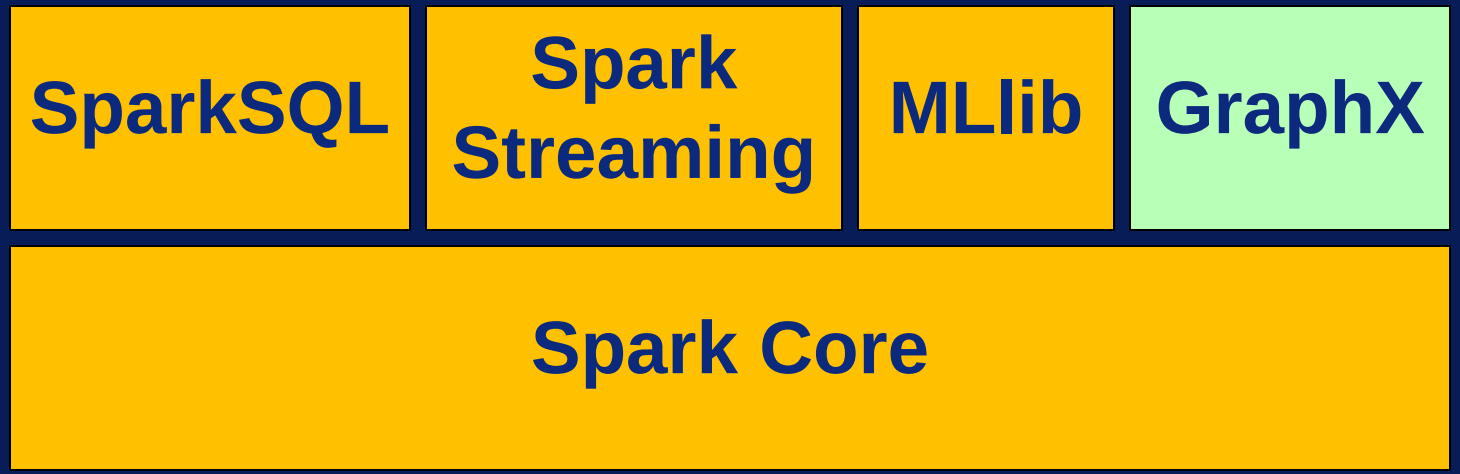
The Spark Stack

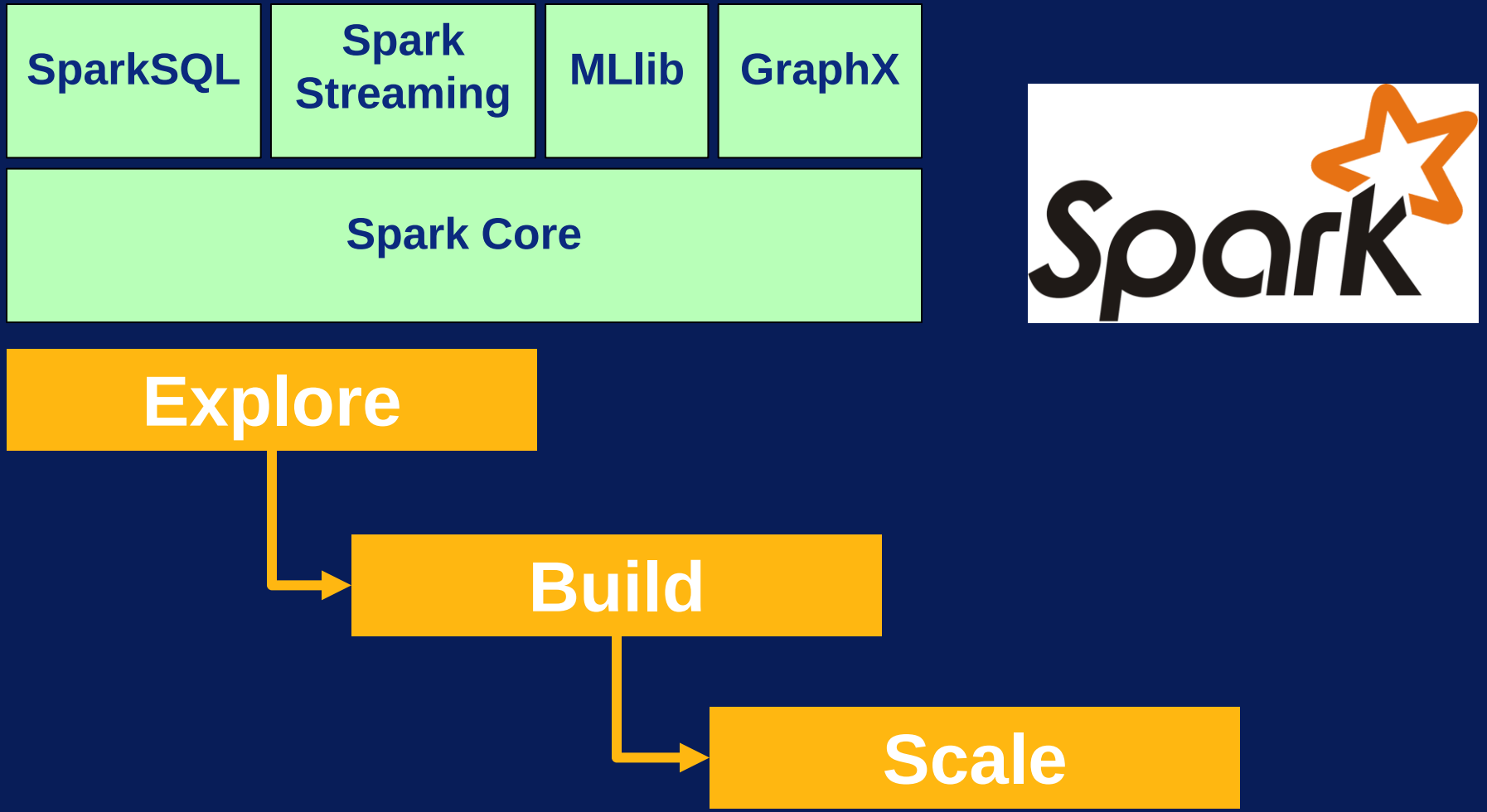


The Spark Stack



The Spark Stack




SparkSQL
Spark Streaming
MLlib
GraphX
Spark Core
Spark
Explore
Build
Scale

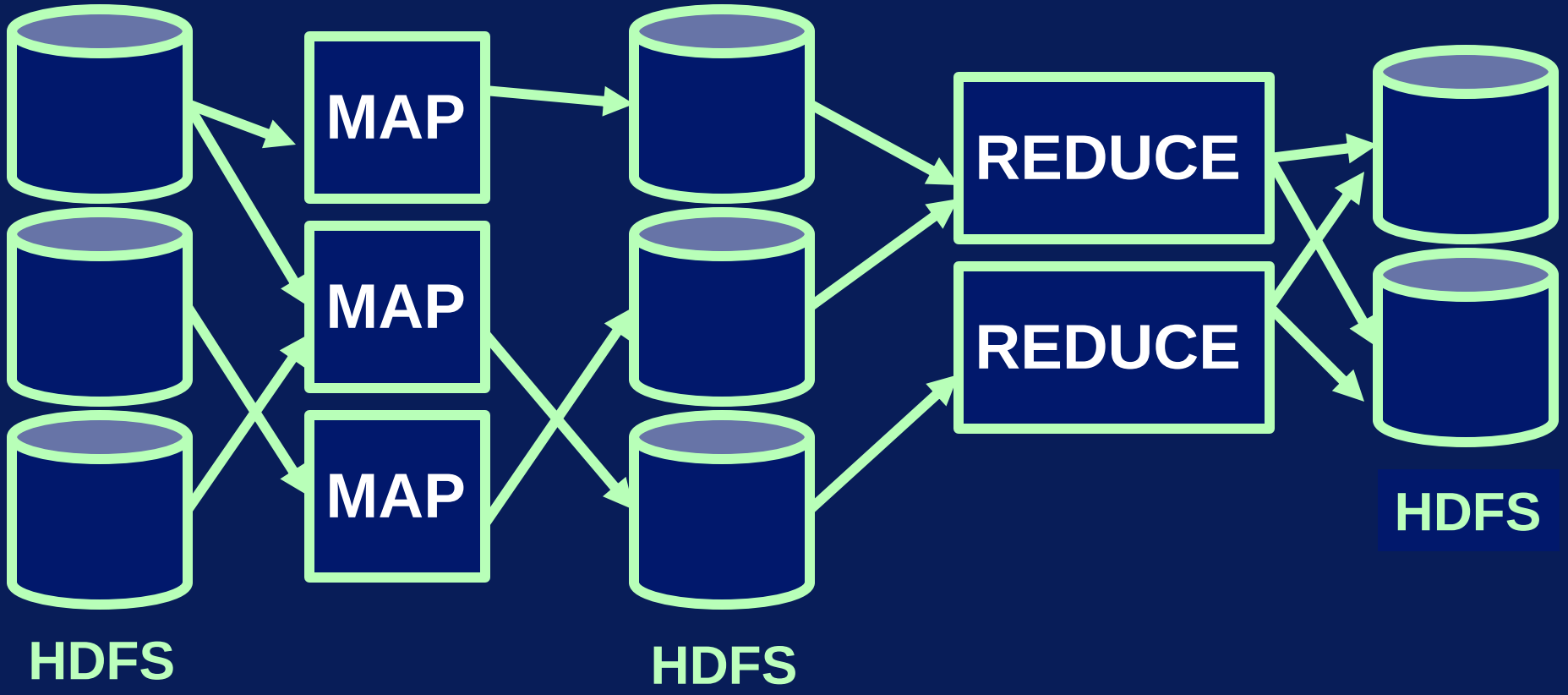
Getting Started with Spark: The Architecture and Basic Concepts

After this video you will be able to..

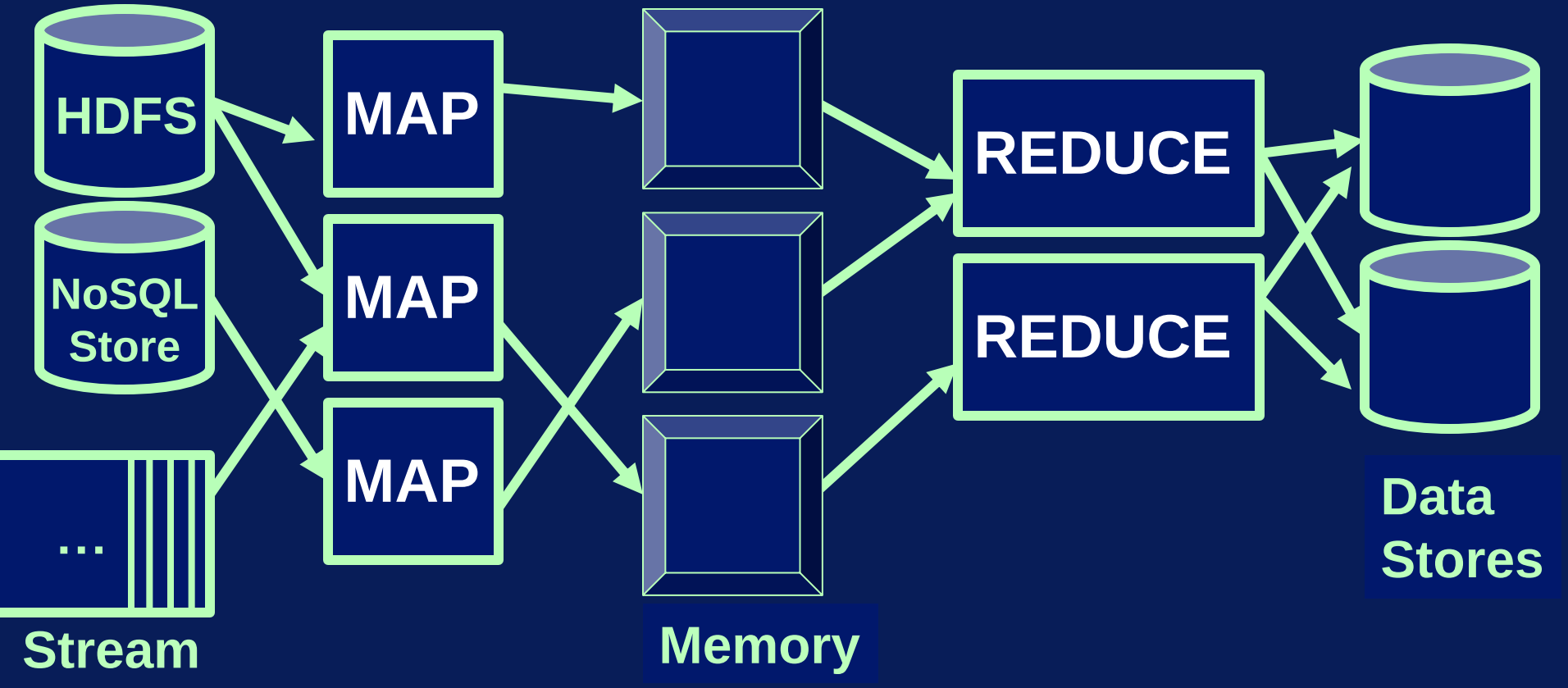
- Describe how Spark does in-memory processing using the RDD abstraction
- Explain the inner workings of the Spark architecture
- Summarize how Spark manages and executes code on Clusters

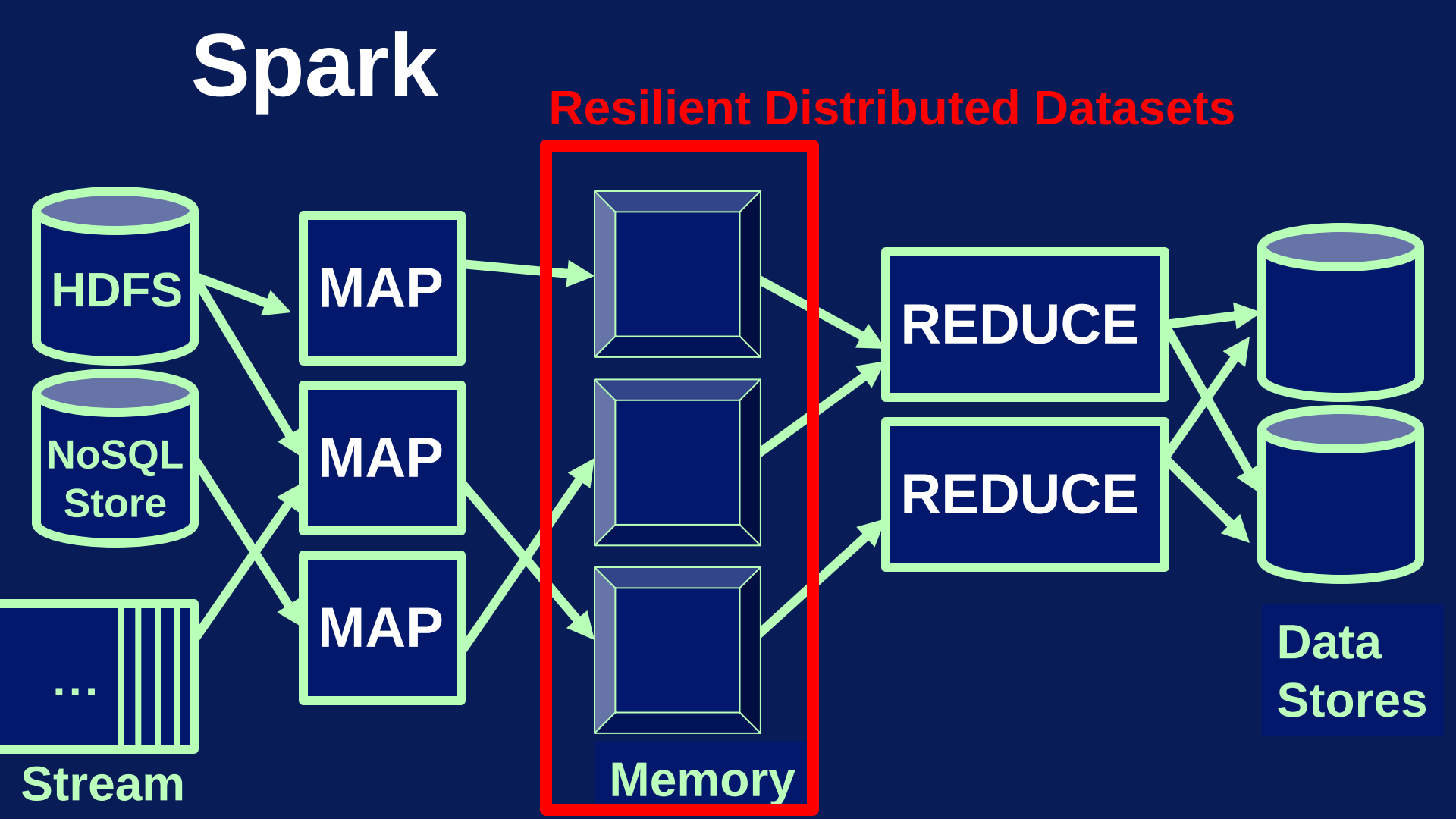
What does in memory processing mean?

MapReduce



Spark


HDFS
NoSQL Store
...
Stream
MAP
MAP
MAP
Memory
REDUCE
REDUCE
Data Stores


Spark
Resilient Distributed Datasets
HDFS
NoSQL Store
...
Stream
MAP
MAP
MAP
Memory
REDUCE
REDUCE
Data Stores

Resilient Distributed Datasets

Dataset

Data storage created from: HDFS, S3, HBase, JSON, text, Local hierarchy of folders

Or created transforming another RDD

Resilient Distributed Datasets

Distributed

Distributed across the cluster of machines

Divided in partitions, atomic chunks of data

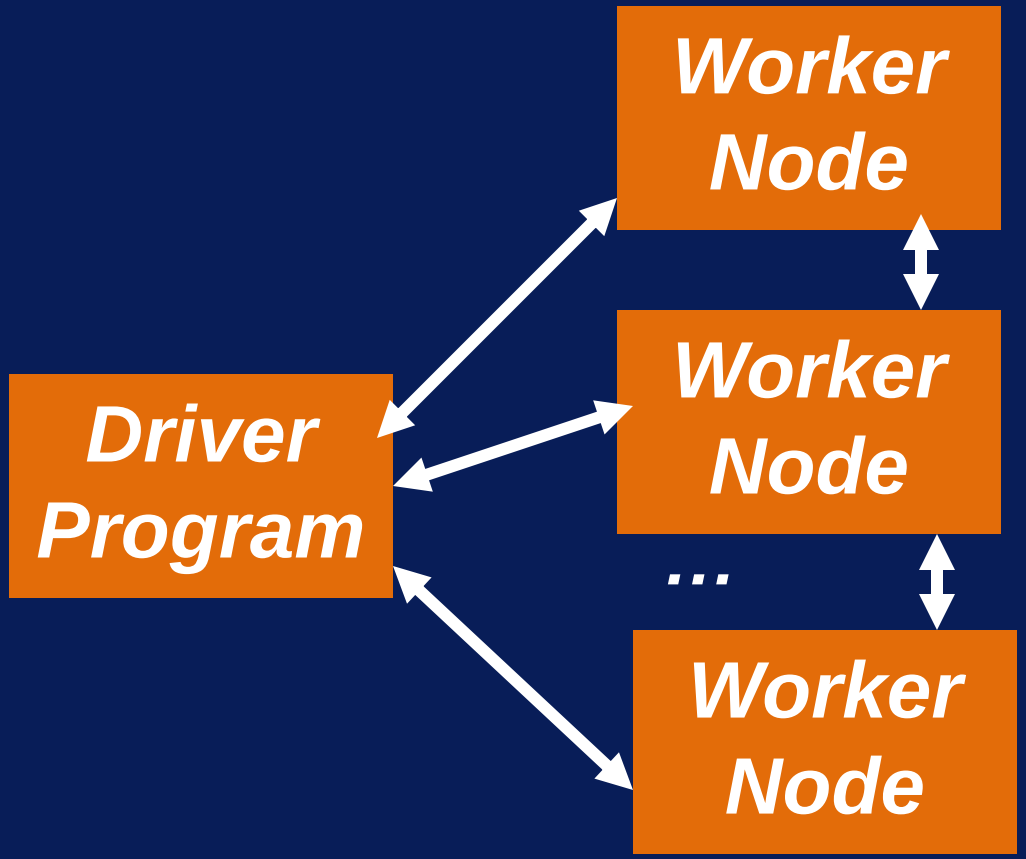
Resilient Distributed Datasets

Resilient

Recover from errors, e.g. node failure, slow processes

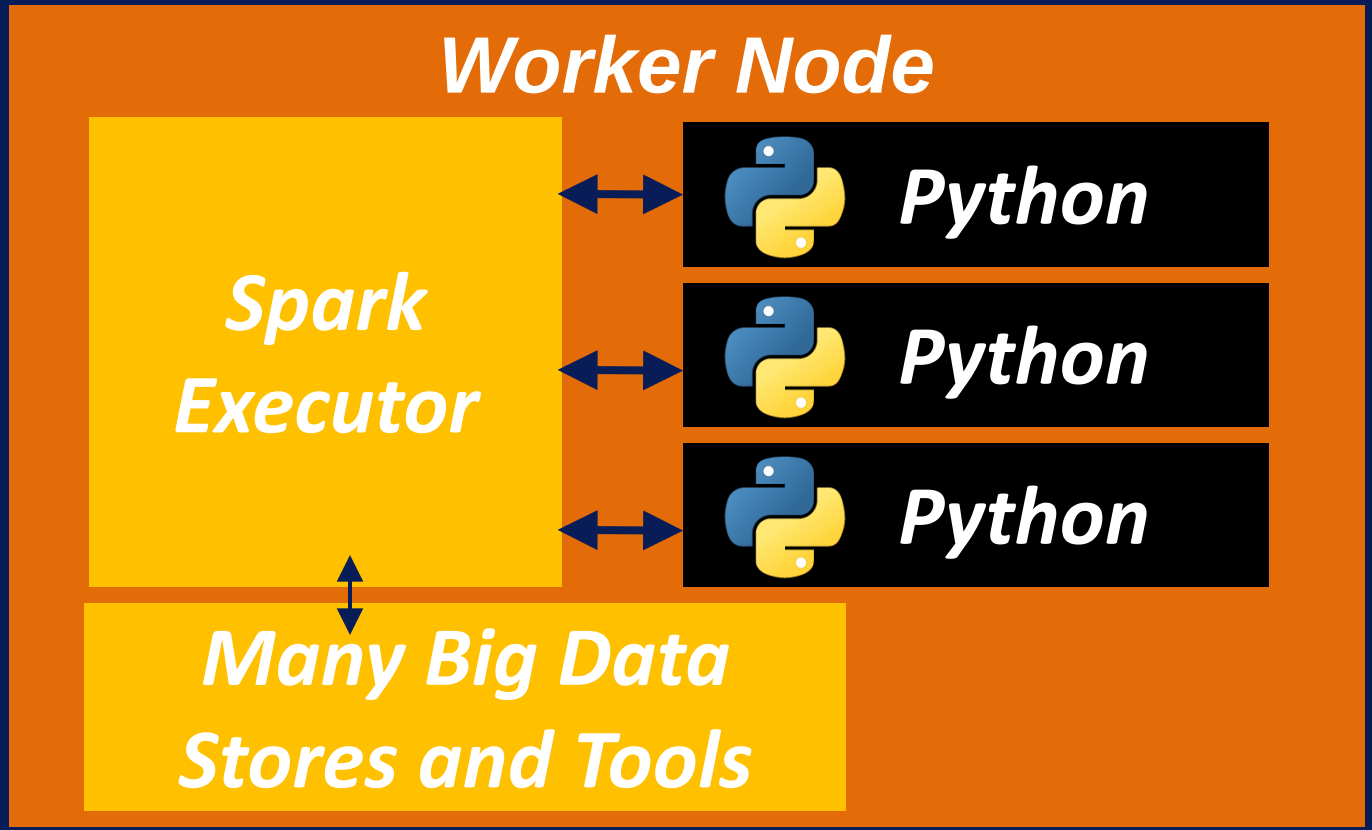
Track history of each partition, re-run

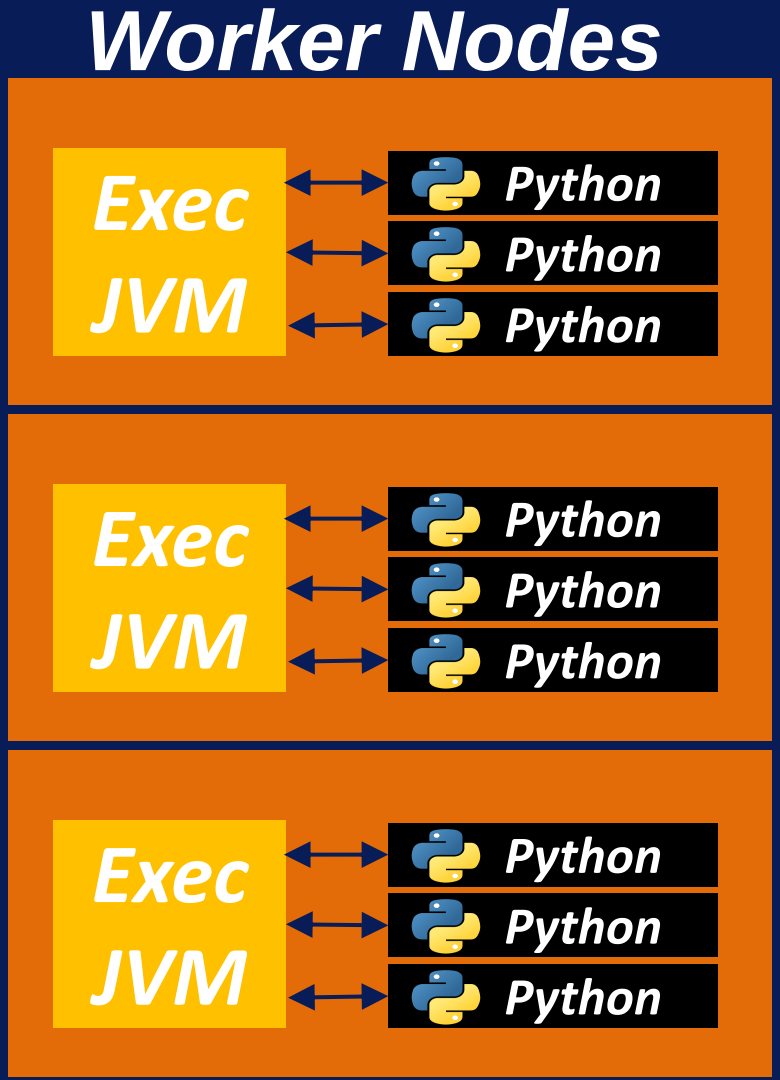
Spark Architecture

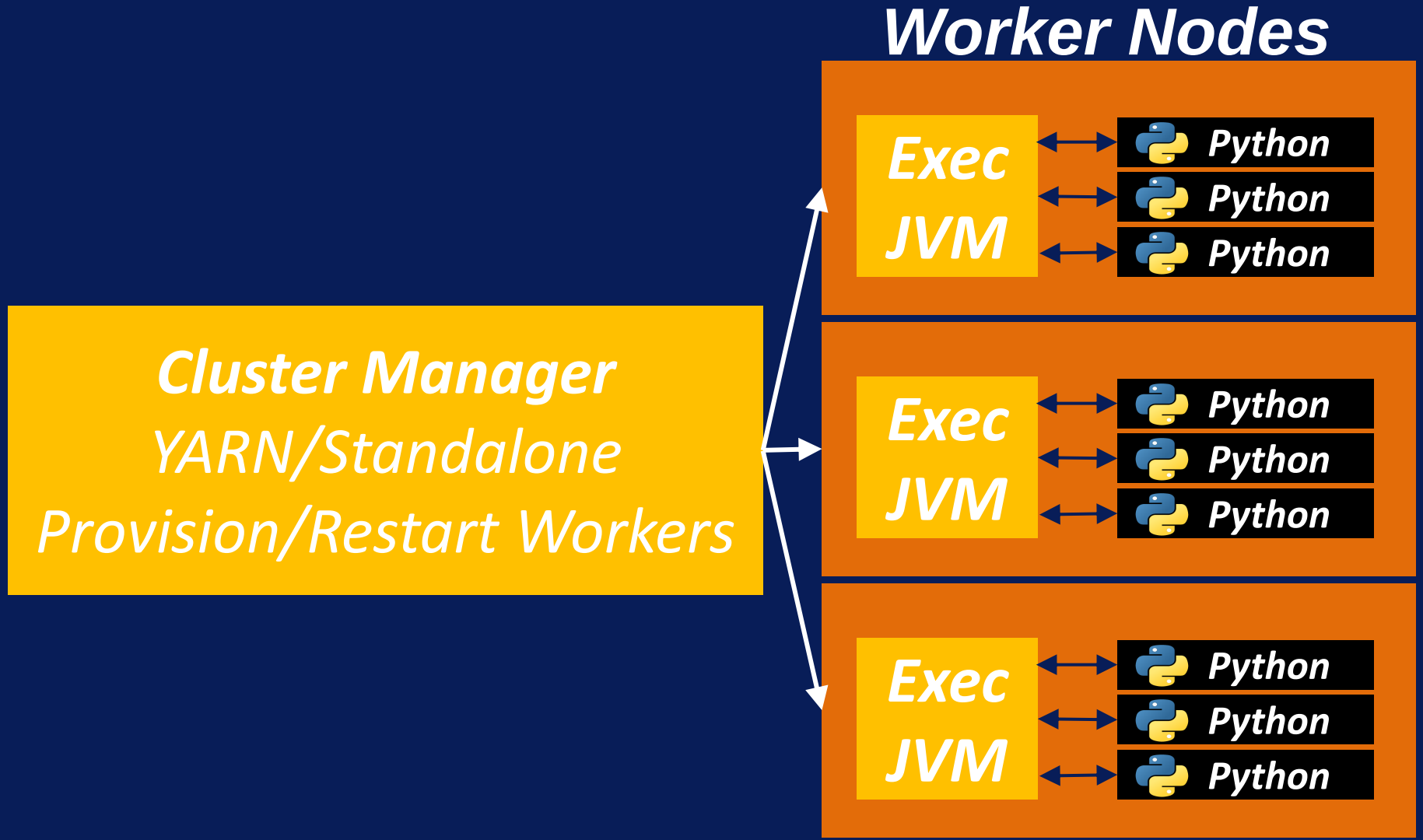

Worker Node
Worker Node
...
Worker Node
Driver Program

Driver Program

```
In [1]: lines = sc.textFile("hdfs:/user/cloudera/words.txt")
```

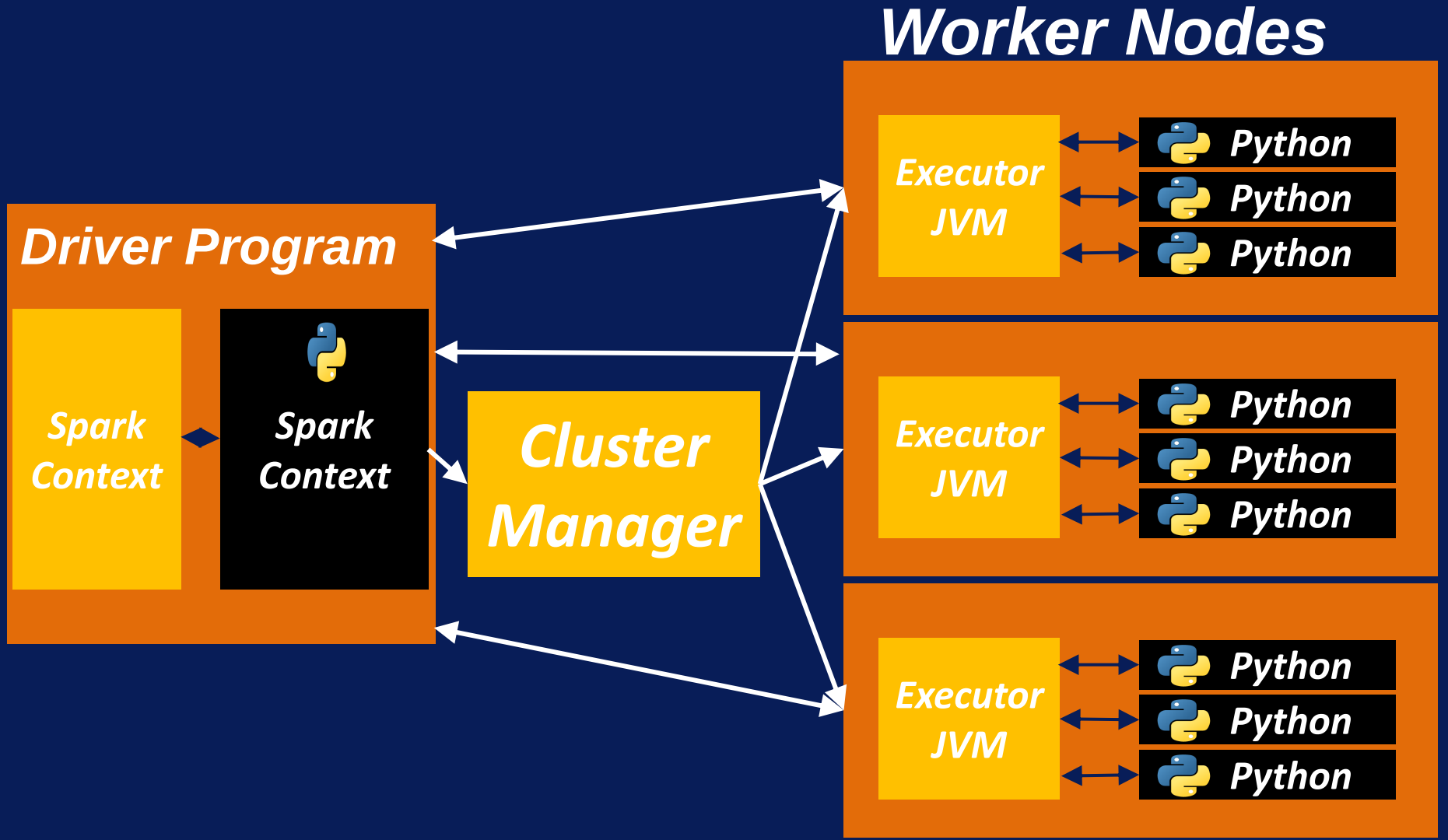

Worker Node
Spark Executor
Python
Python
Python
Many Big Data Stores and Tools


Worker Nodes
Exec
JVM
Python
Python
Python
Exec
JVM
Python
Python
Python
Exec
JVM
Python
Python
Python

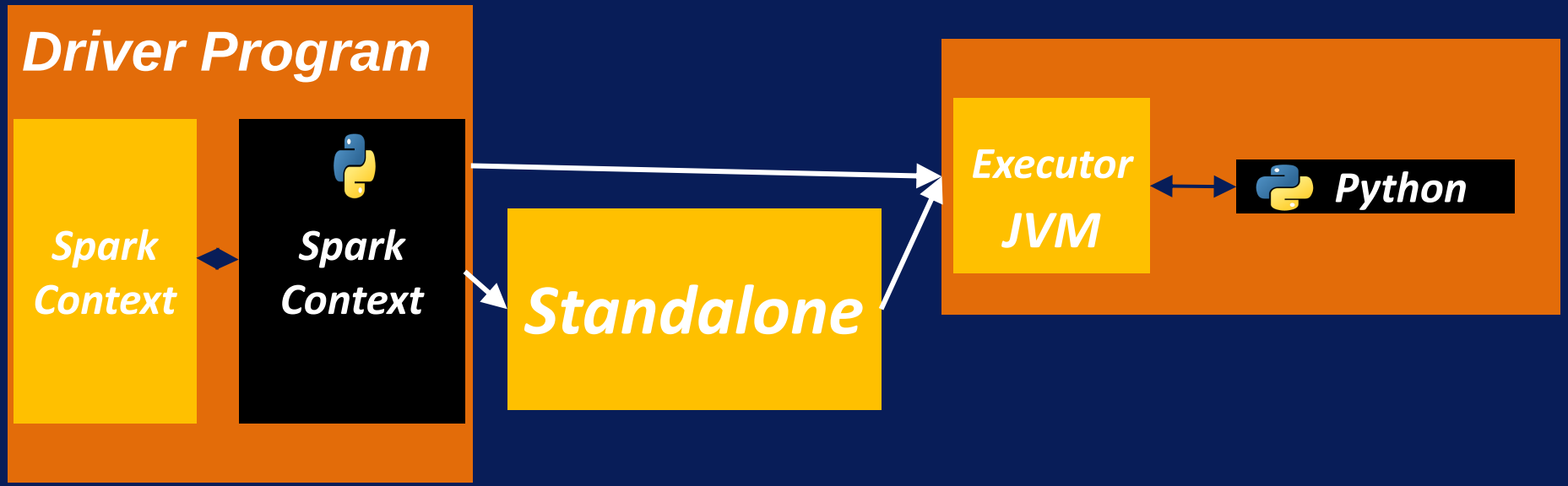

Worker Nodes
Cluster Manager
YARN/Standalone
Provision/Restart Workers
Exec JVM
Python
Python
Python
Exec JVM
Python
Python
Python
Exec JVM
Python
Python
Python

Which cluster manager?

http://www.agildata.com/apache-spark-cluster-managers-yarn-mesos-or-standalone/


Worker Nodes
Driver Program
Spark Context
Spark Context
Cluster Manager
Executor JVM
Python
Python
Python
Executor JVM
Python
Python
Python
Executor JVM
Python
Python
Python

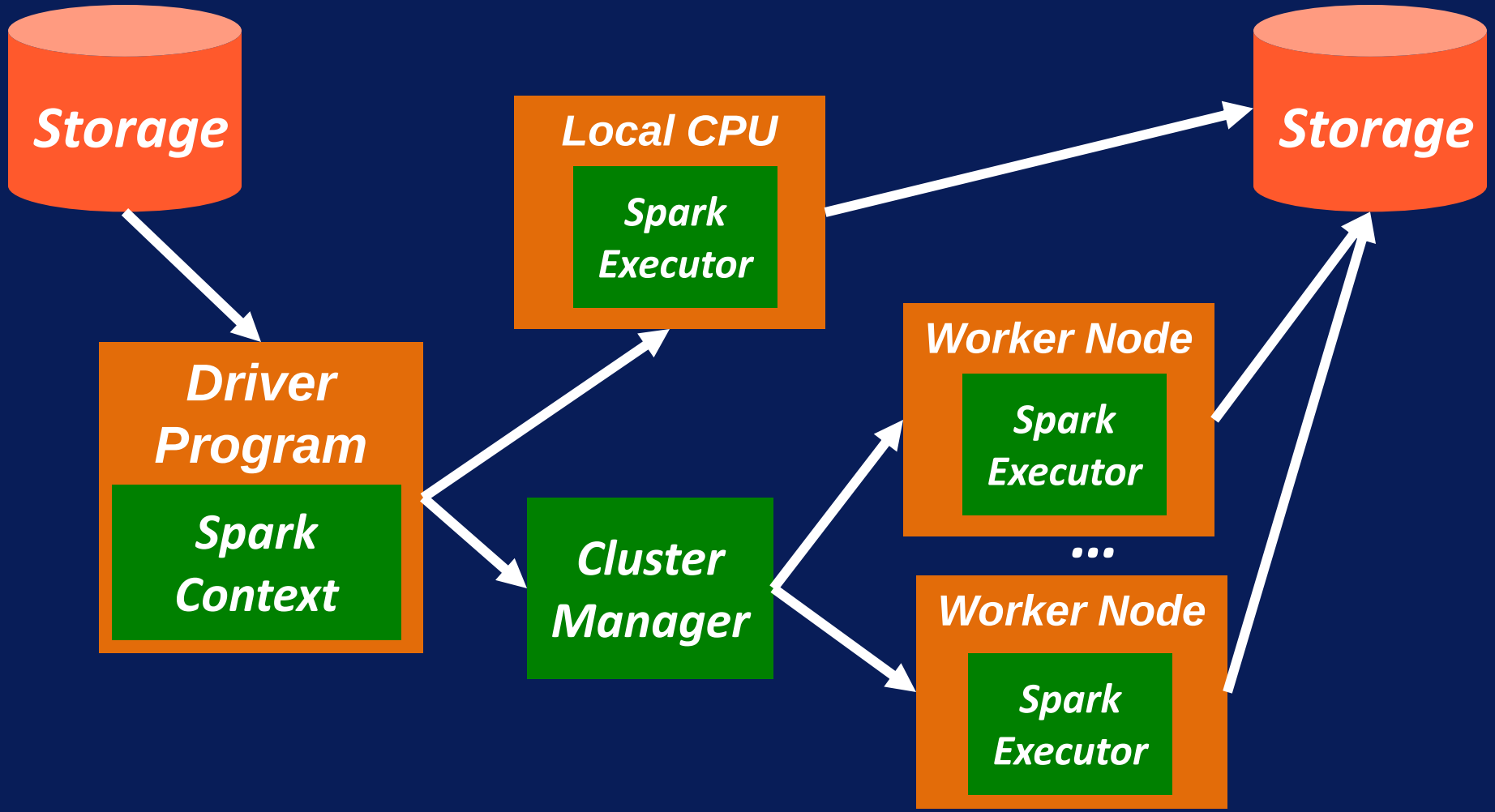
Cloudera VM


Driver Program
Spark Context
Spark Context
Standalone
Executor JVM
Python

Spark Core:
Programming In Spark

After this video you will be able to..

- Use two methods to create RDDs in Spark
- Explain what immutable means
- Interpret a Spark program as a pipeline of transformations and actions
- List the steps to create a Spark program


Storage
Local CPU
Spark Executor
Storage
Driver Program
Spark Context
Cluster Manager
Worker Node
Spark Executor
...
Worker Node
Spark Executor

Creating RDDs

Driver Program

```
In [1]: lines = sc.textFile("hdfs:/user/cloudera/words.txt")
```

```
lines = sc.parallelize(["big", "data"])
```

```
numbers = sc.parallelize(range(10), 3)
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

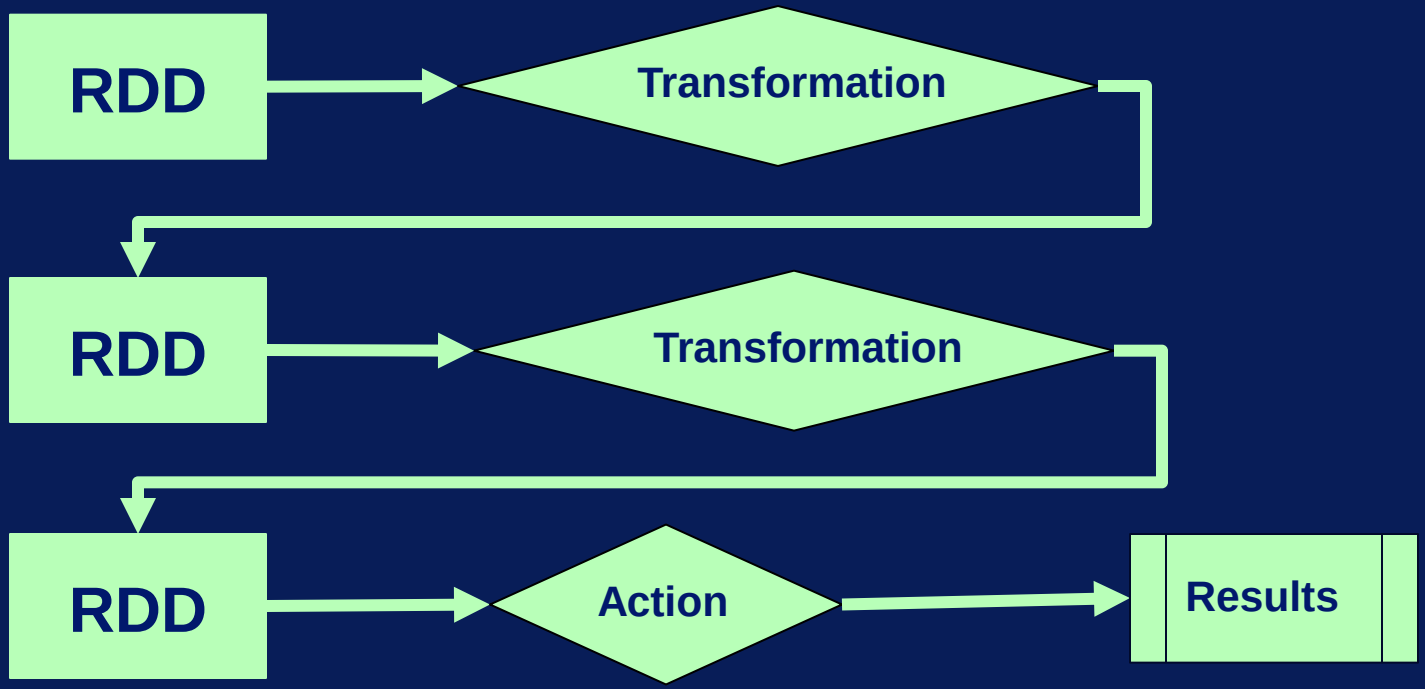
Parallelize range output into 3 partitions

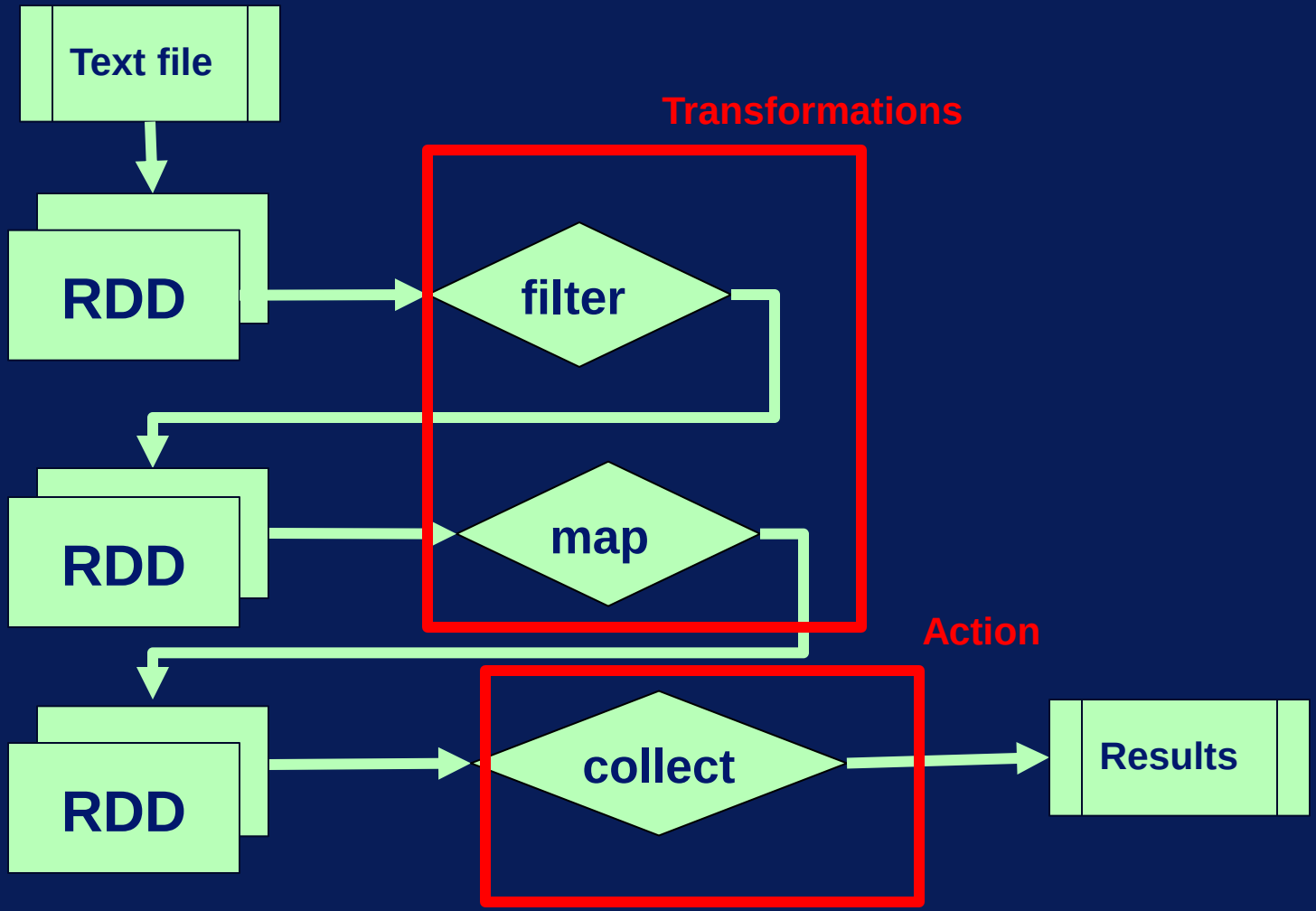
[0, 1, 2], [3, 4, 5], [6, 7, 8, 9]

```
numbers.collect()
```

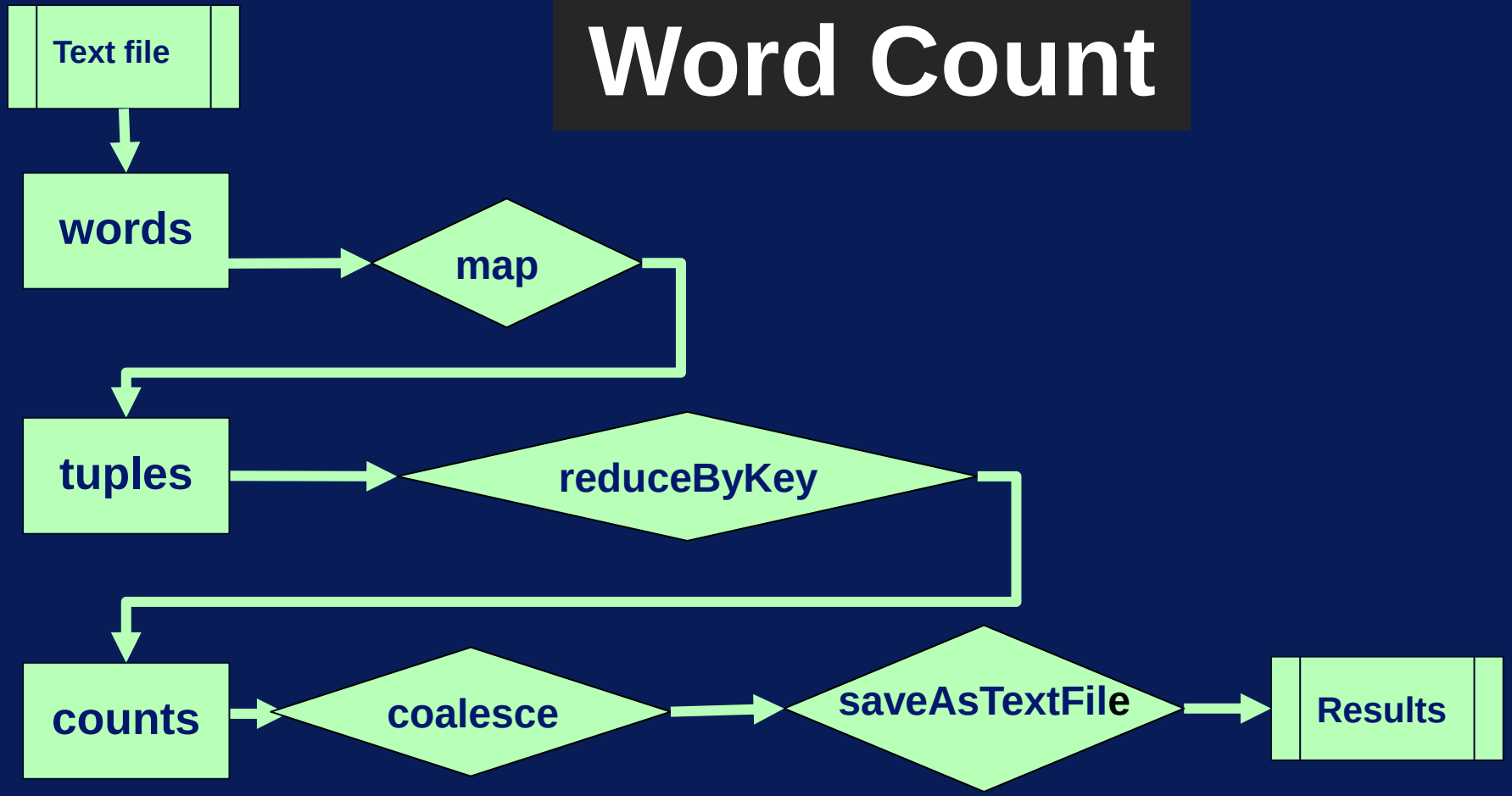
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Processing RDDs




Text file
Transformations
RDD
filter
RDD
map
Action
RDD
collect
Results

Word Count


Text file
words
map
tuples
reduceByKey
counts
coalesce
saveAsTextFile
Results

Programming in Spark

Create RDDs

Apply transformations

Perform actions

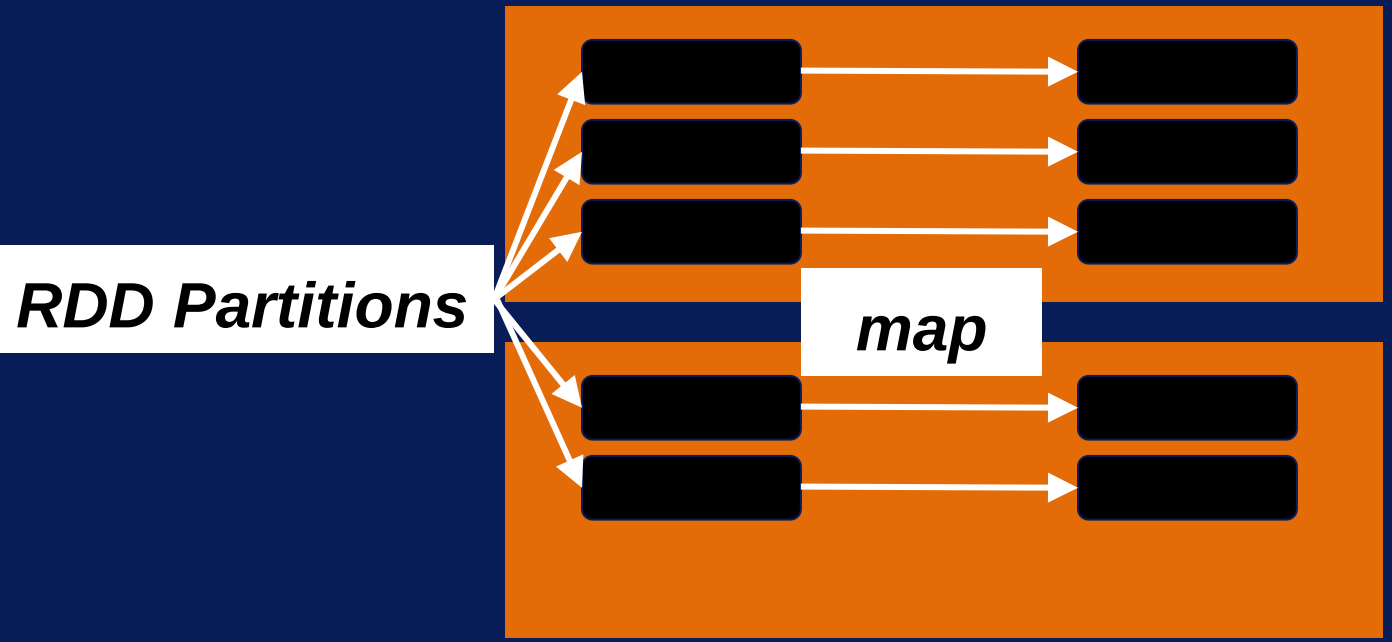
Spark Core: Transformations

After this video you will be able to..

- Explain the difference between a narrow transformation and wide transformation
- Describe map, flatmap, filter and coalesce as narrow transformations
- List two wide transformations

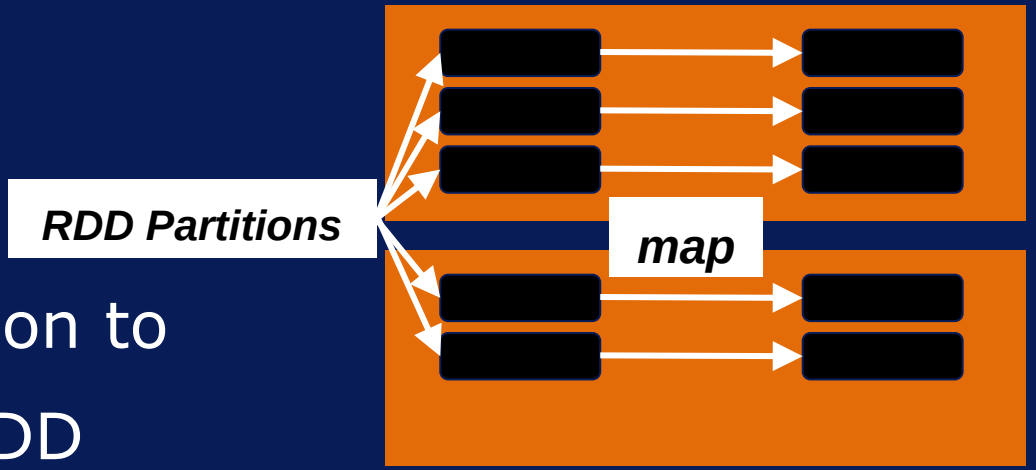
map

map : apply function to each element of RDD



map

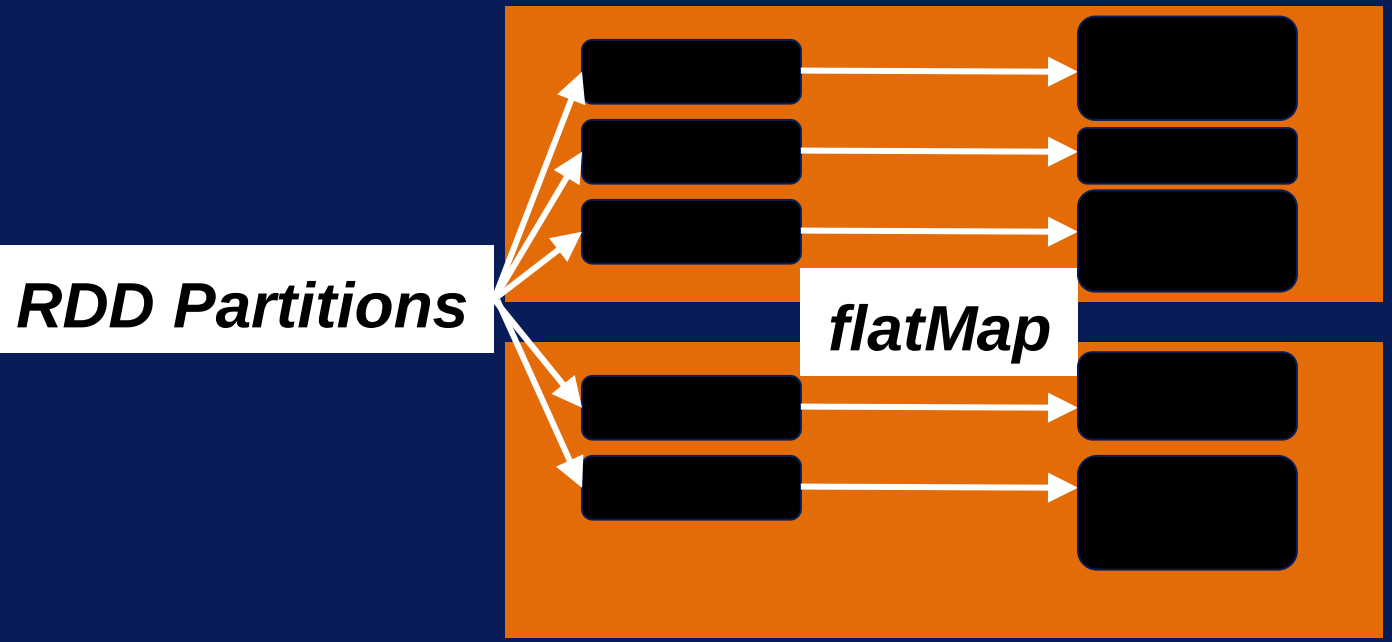
map : apply function to each element of RDD




```
def lower(line):
    return line.lower()
lower_text_RDD = text_RDD.map(lower)
```

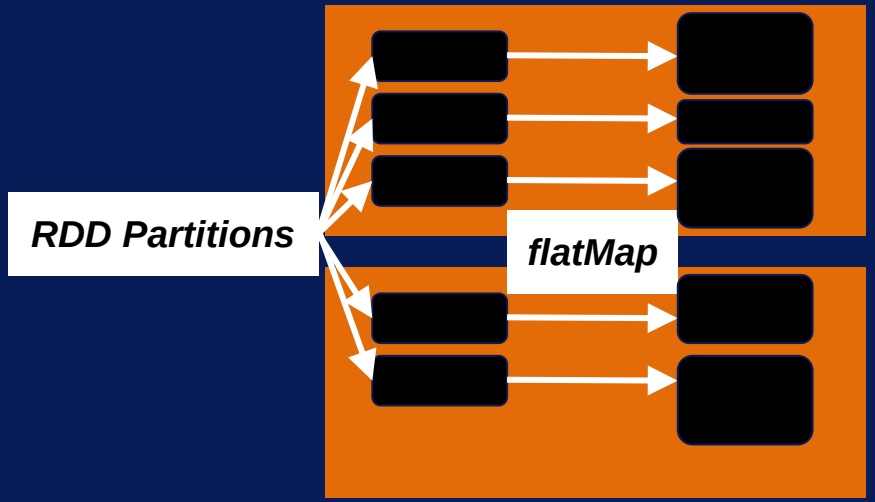
flatMap

flatMap : map then flatten output



flatMap

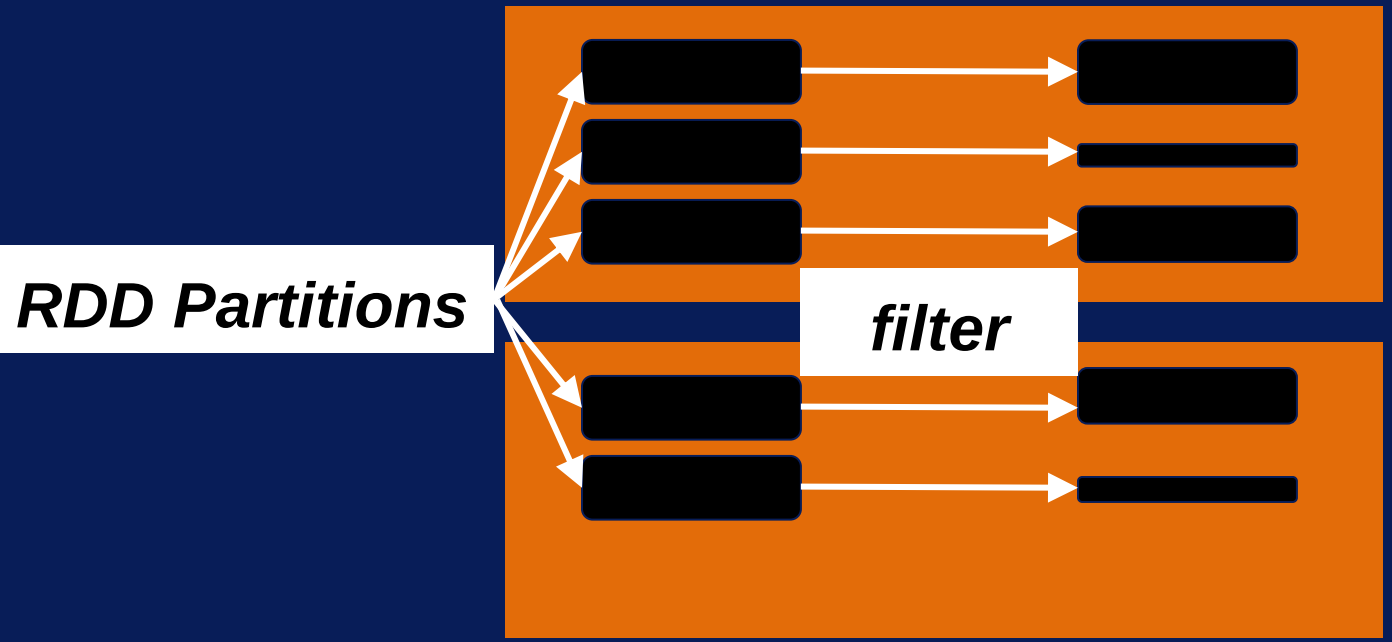
flatMap : map then flatten output



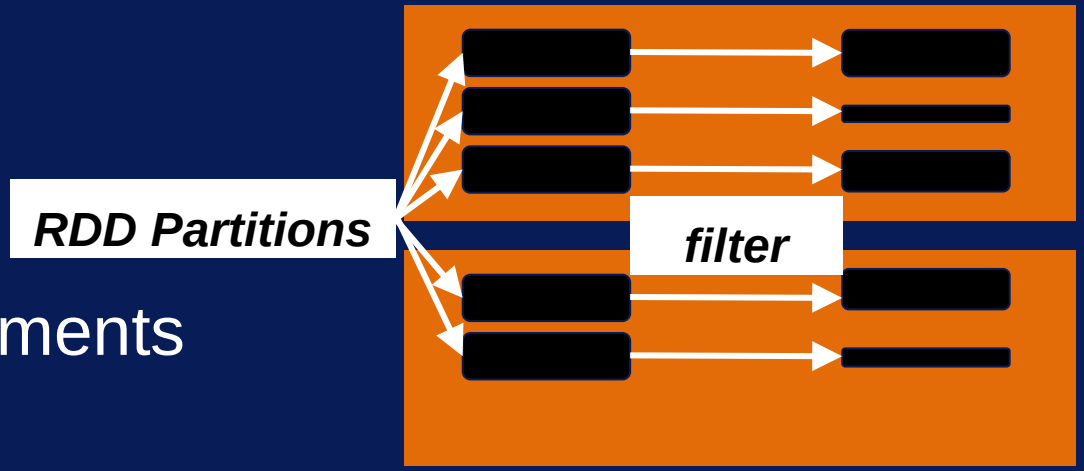

```
def split_words(line):
    return line.split()

words_RDD = text_RDD.flatMap(split_words)
words_RDD.collect()
```

filter

filter : keep only elements where function is true



filter



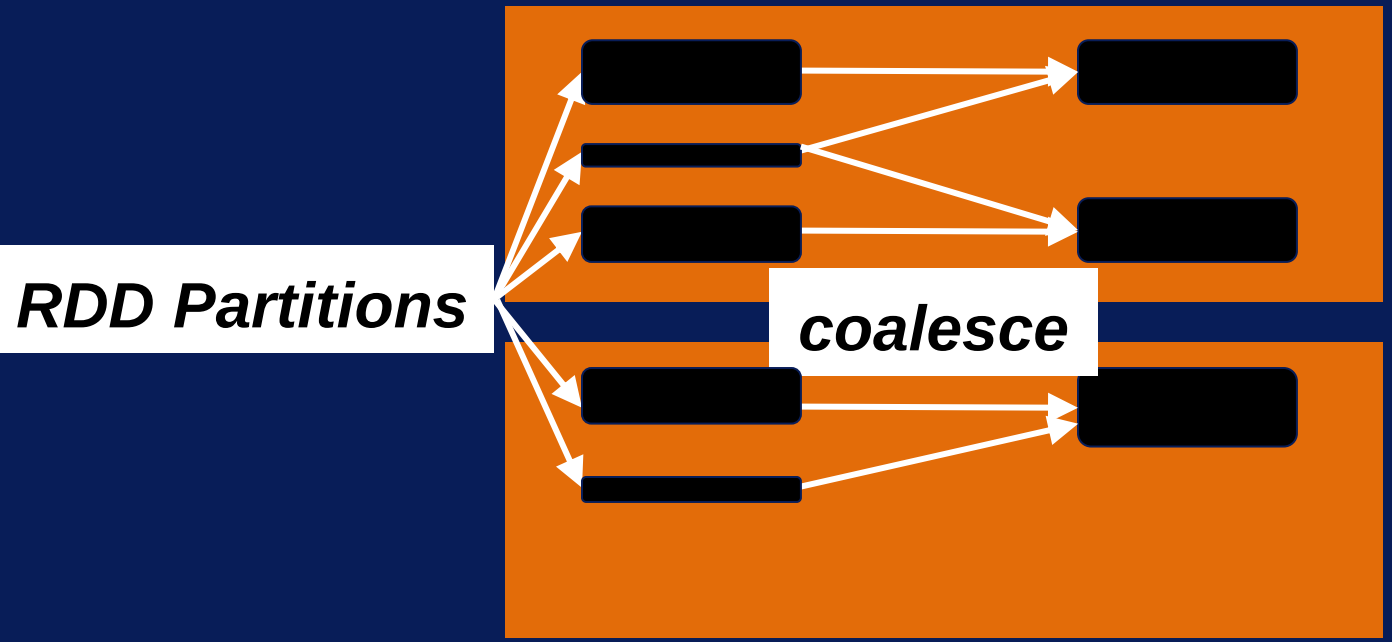
filter : keep only elements

where function is true

```
def starts_with_a(word):
    return word.lower().startswith("a")
words_RDD.filter(starts_with_a).collect()
```

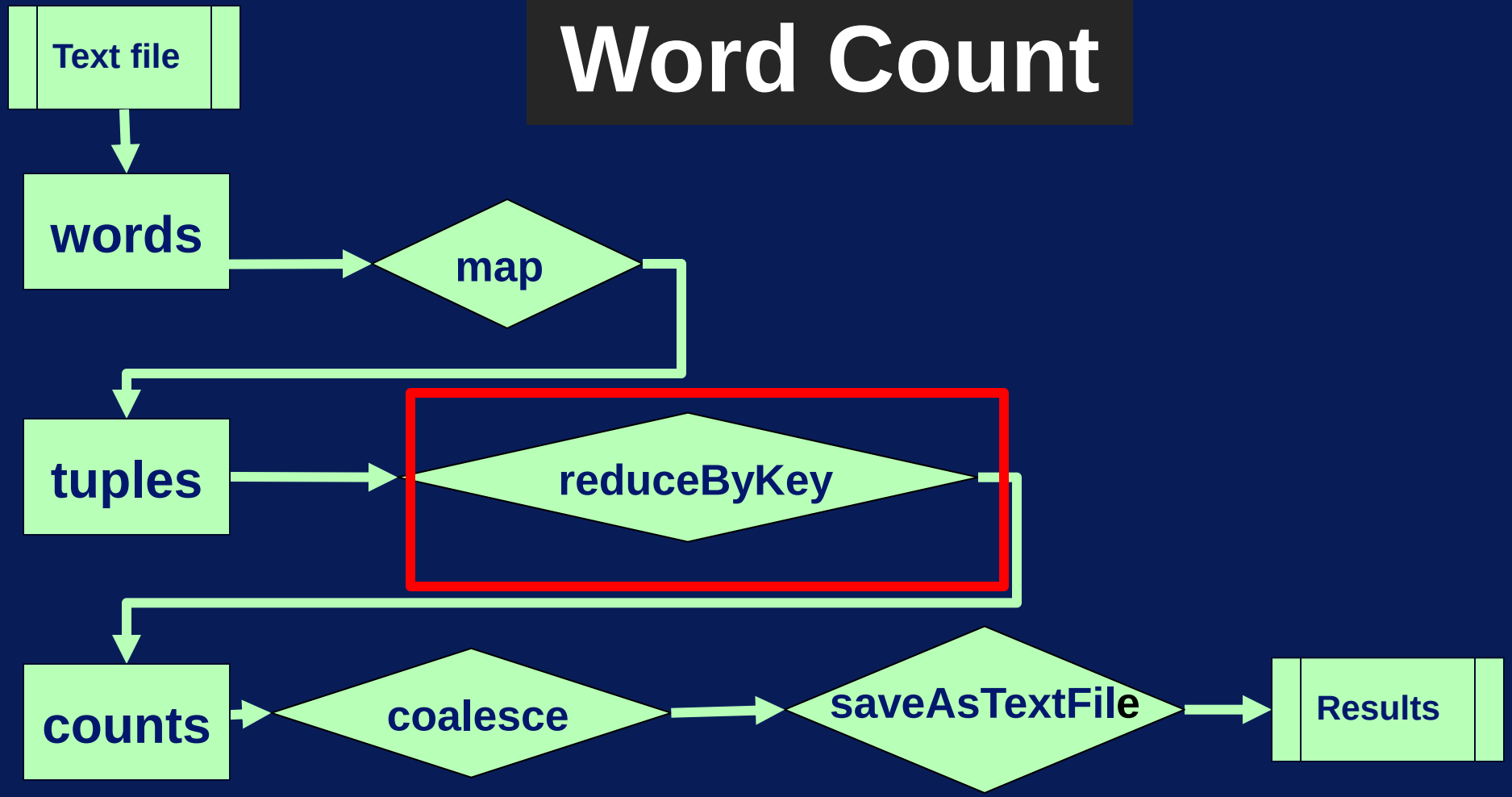
coalesce

coalesce : reduce the number of partitions

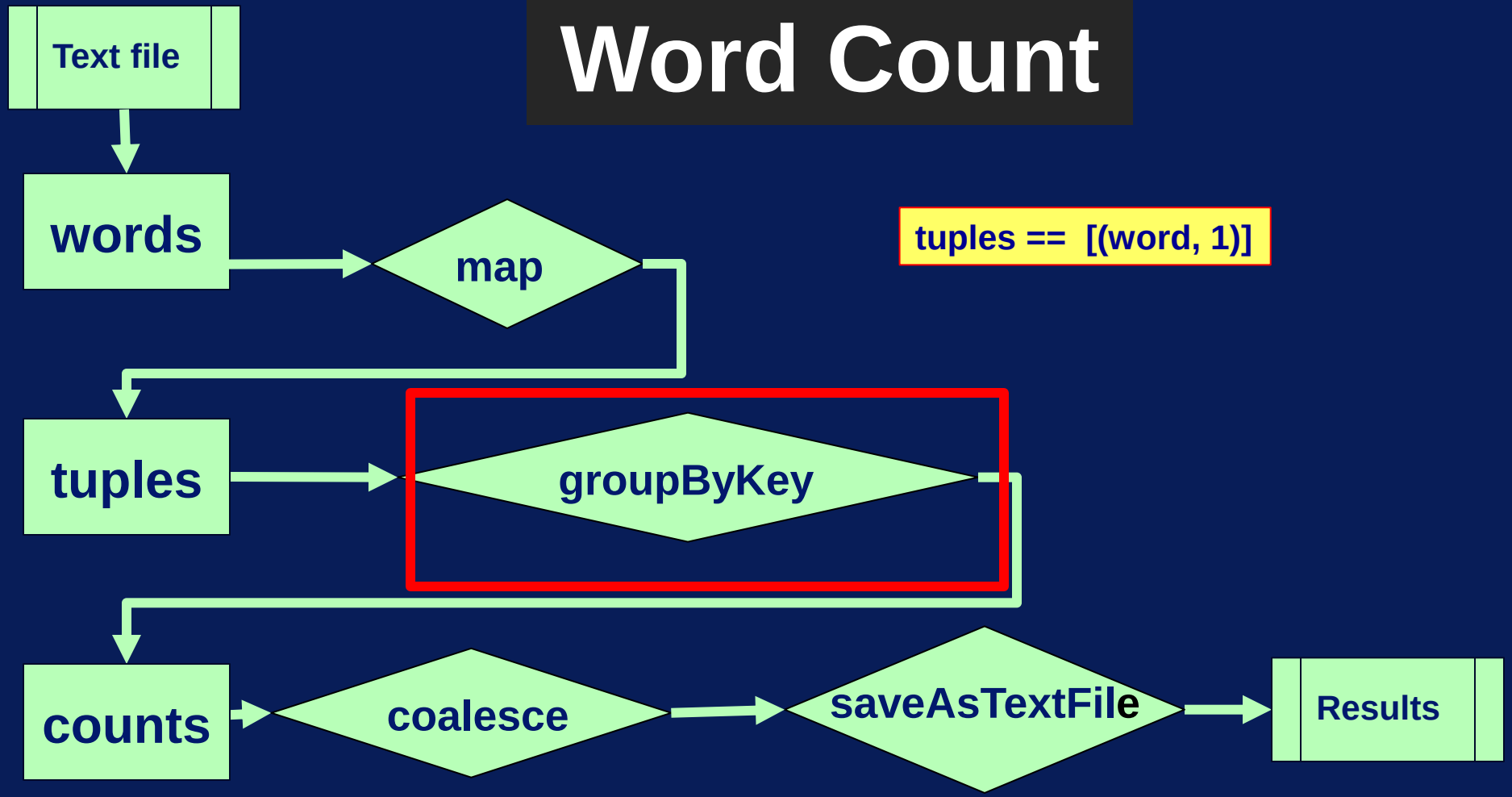


Wide Transformations

Word Count

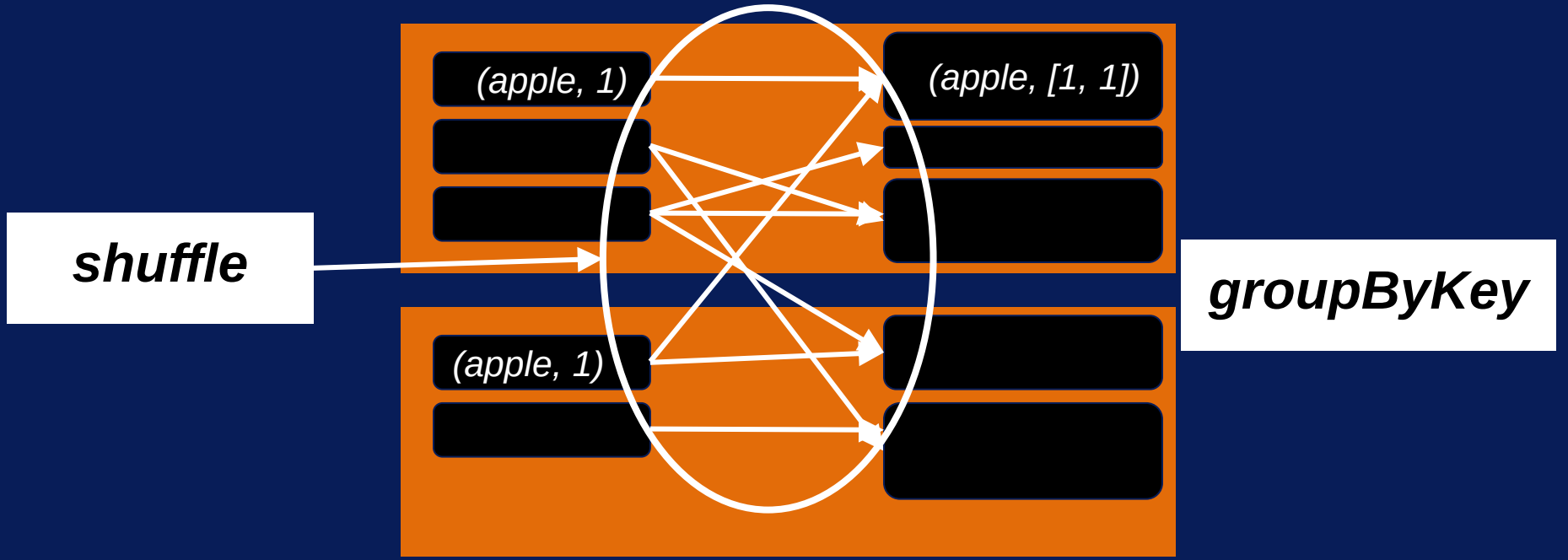

Text file
words
map
tuples
reduceByKey
counts
coalesce
saveAsTextFile
Results

Word Count

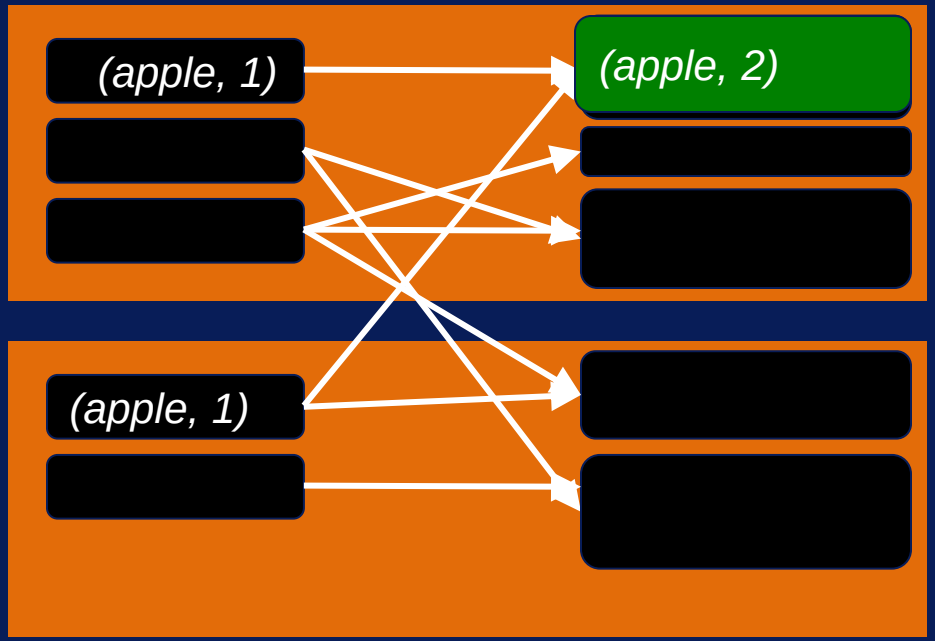

Text file
words
map
tuples == [(word, 1)]
tuples
groupByKey
counts
coalesce
saveAsTextFile
Results

groupByKey

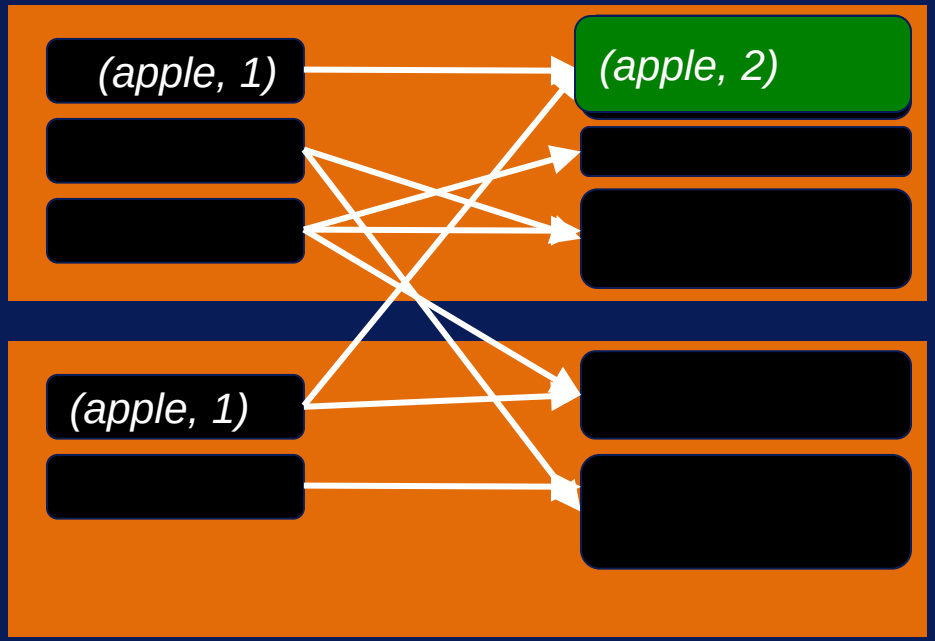
groupByKey : (K, V) pairs => (K, list of all V)



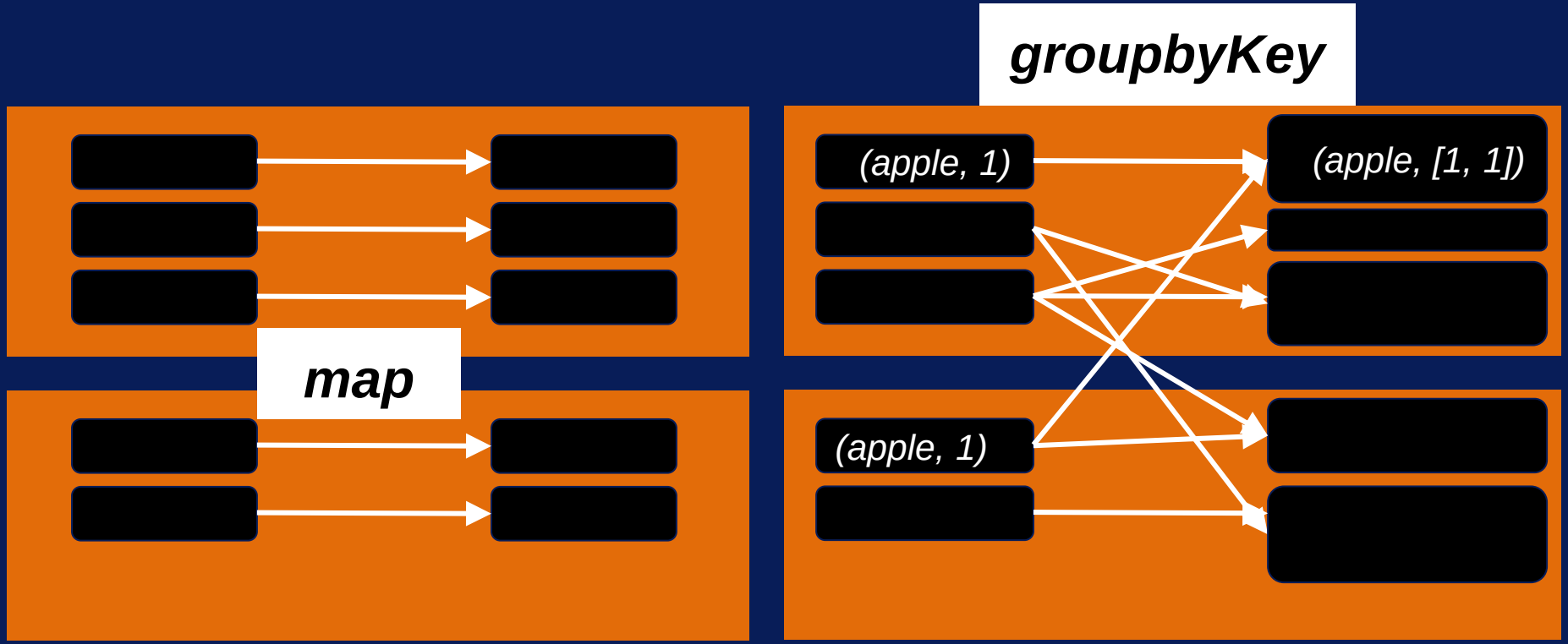
groupByKey + reduce



reduceByKey



Narrow vs Wide



Many more transformations…

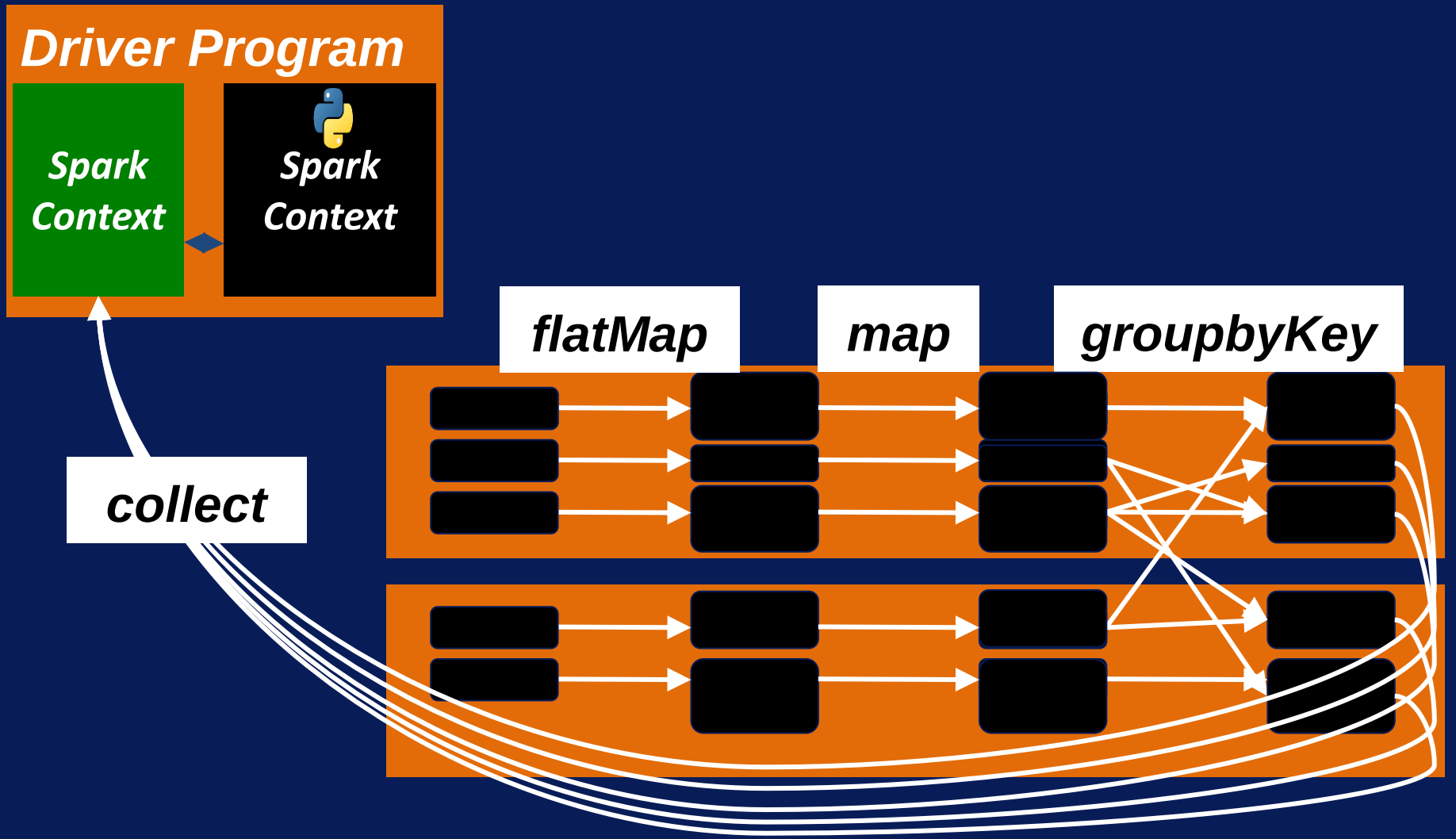
Full list of transformations at:

https://spark.apache.org/docs/1.2.0/programming-guide.html#transformations

Spark Core: Actions

After this video you will be able to..

- Explain the steps of a Spark pipeline ending with a collect action
- List four common action operations in Spark


Driver Program
Spark Context
Spark Context
flatMap
map
groupbyKey
collect

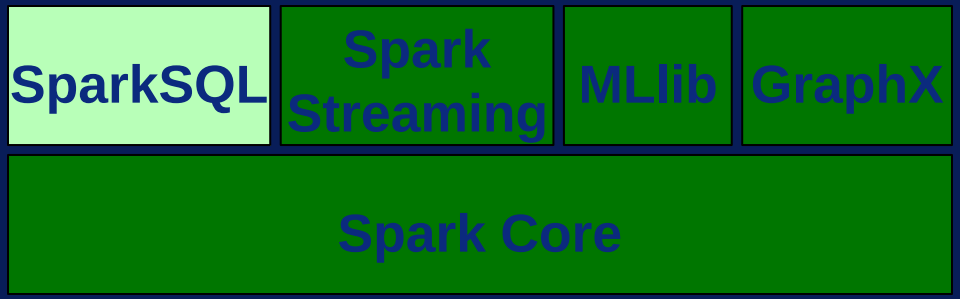
Some Common Actions

Action	Usage
collect()	Copy all elements to the driver
take(n)	Copy first n elements
reduce(func)	Aggregate elements with func (takes 2 elements, returns 1)
[redacted]	Save to local file or HDFS

Spark SQL

After this video you will be able to..

- Process structured data using Spark's SQL module
- Explain the numerous benefits of Spark SQL



Spark SQL

- Enables querying structured and unstructured data through Spark
- Provides a common query language
- Has APIs for Scala, Java and Python to convert results into RDDs

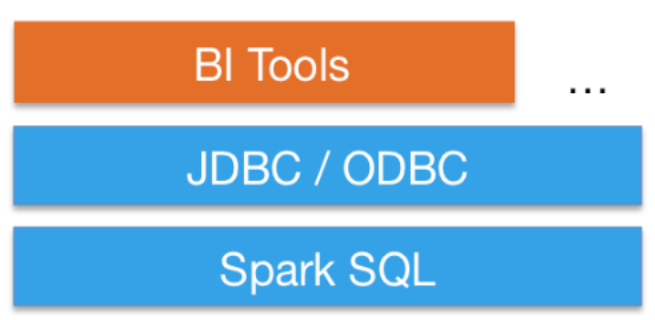
Relational Operations

Perform Relational Processing such as Declarative Queries

Embed SQL queries inside Spark Programs

Business Intelligence Tools

Spark SQL connects to all BI tools that support JDBC or ODBC standard



http://spark.apache.org/

DataFrames

Distributed Data organized as named columns

Look just like a table in relational databases

How to go Relational in Spark ?

Step 1: Create a SQLContext

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

How to go Relational in Spark ?

Create a DataFrame from

- an existing RDD
- a Hive table
- data sources

JSON → DataFrame

```
# Read
df = sqlContext.read.json("/filename.json")

# Display
df.show()
```

RDD of Row objects → DataFrame

```
# Read
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

# Load a text file and convert each line to a Row.
lines = sc.textFile("filename.txt")
cols = lines.map(lambda l: l.split(","))
data = cols.map(lambda p: Row(name=p[0], zip=int(p[1])))

# Create DataFrame
df = sqlContext.createDataFrame(data)

# Register the DataFrame as a table
df.registerTempTable(”table")

# Run SQL
Output = sqlContext.sql("SELECT * FROM table WHERE …")
```

DataFrames are just like tables

http://spark.apache.org/

```
# Show the content of the DataFrame
df.show()

# Print the schema
df.printSchema()

# Select only the "X" column
df.select("X").show()

# Select everybody, but increment the discount by 5%
df.select(df["name"], df["discount"] + 5).show()

# Select people height greater than 4.0 ft
df.filter(df["height"] > 4.0).show()

# Count people by zip
df.groupBy("zip").count().show()
```

Spark SQL

Relational on Spark

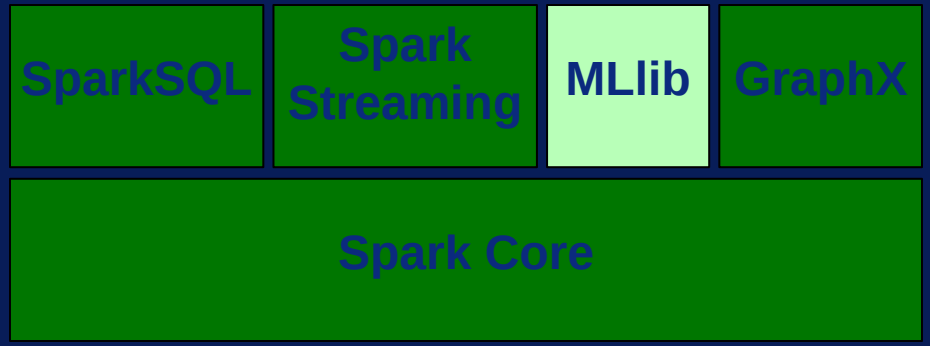
Connect to variety of databases

Deploy business intelligence tools over Spark

Spark MLlib

After this video you will be able to..

- Describe what MLlib is
- List main categories of techniques available in MLlib.
- Explain code segments containing MLlib algorithms.



Spark MLlib

- Scalable machine learning library
- Provides distributed implementations of common machine learning algorithms and utilities
- Has APIs for Scala, Java, Python, and R

MLlib Algorithms & Techniques

- Machine Learning
  - Classification, regression, clustering, etc.
  - Evaluation metrics
- Statistics
  - Summary statistics, sampling, etc.
- Utilities
  - Dimensionality reduction, transformation, etc.

MlLib Example – Summary Statistics

- Compute column summary statistics

```
from pyspark.mllib.stat import Statistics

# Data as RDD of Vectors
dataMatrix = sc.parallelize([ [1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12] ])

# Compute column summary statistics.
summary = Statistics.colStats(dataMatrix)
print(summary.mean())
print(summary.variance())
print(summary.numNonzeros())
```

MLlib Example – Classification

- Build decision tree model for classification

```
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

# Read and parse data
data = sc.textFile(“data.txt”)

# Decision tree for classification
model = DecisionTree.trainClassifier
            (parsedData, numClasses=2)
print(model.toDebugString())
model.save(sc, “decisionTreeModel”)
```

MLlib Example – Clustering

- Build k-means model for clustering

```
from pyspark.mllib.clustering import KMeans, KMeansModel
from numpy import array

# Read and parse data
data = sc.textFile("data.txt")
parsedData = data.map(lambda line:
        array([float(x) for x in line.split(' ')]))

# k-means model for clustering
clusters = Kmeans.train (parsedData, k=3)

print(clusters.centers)
```

1

2

3

4

5

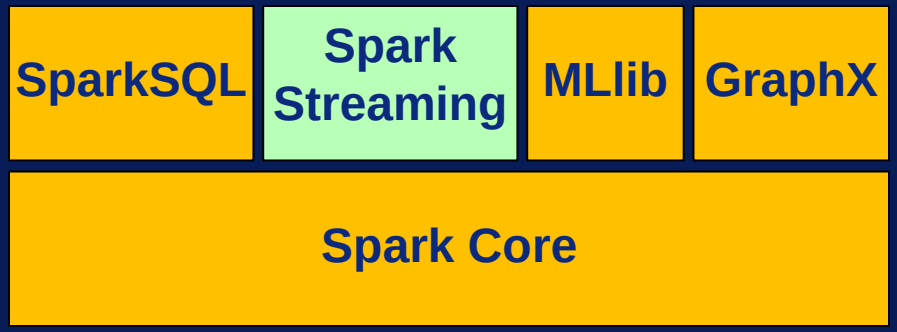
Main Take-Aways

- MLlib is Spark's machine learning library.
  - Distributed implementations
- Main categories of algorithms and techniques:
  - Machine learning
  - Statistics
  - Utility for ML pipeline

Spark Streaming

After this video you will be able to..

- Summarize how Spark reads streaming data
- List several sources of streaming data supported by Spark
- Describe Spark’s sliding windows



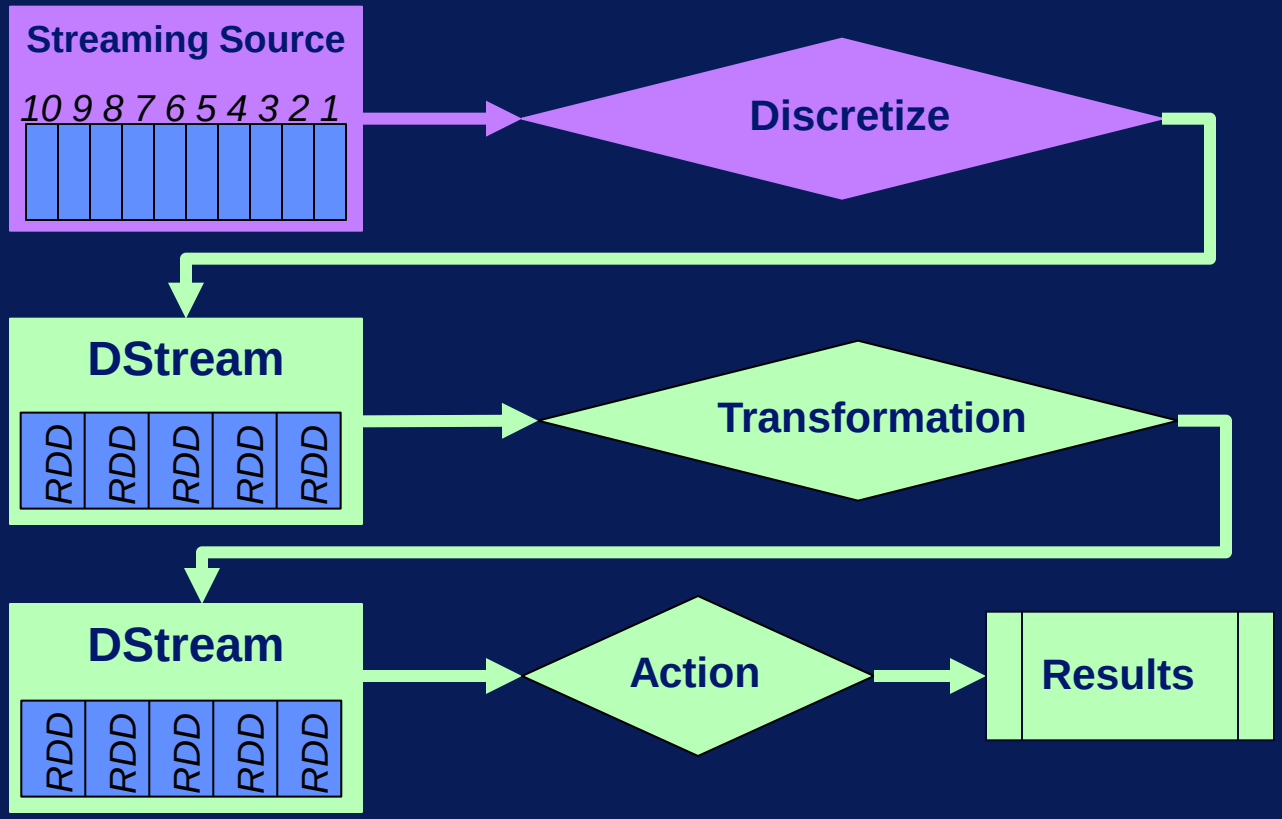
Spark Streaming

- Scalable processing for real-time analytics
- Data streams converted to discrete RDDs
- Has APIs for Scala, Java, and Python

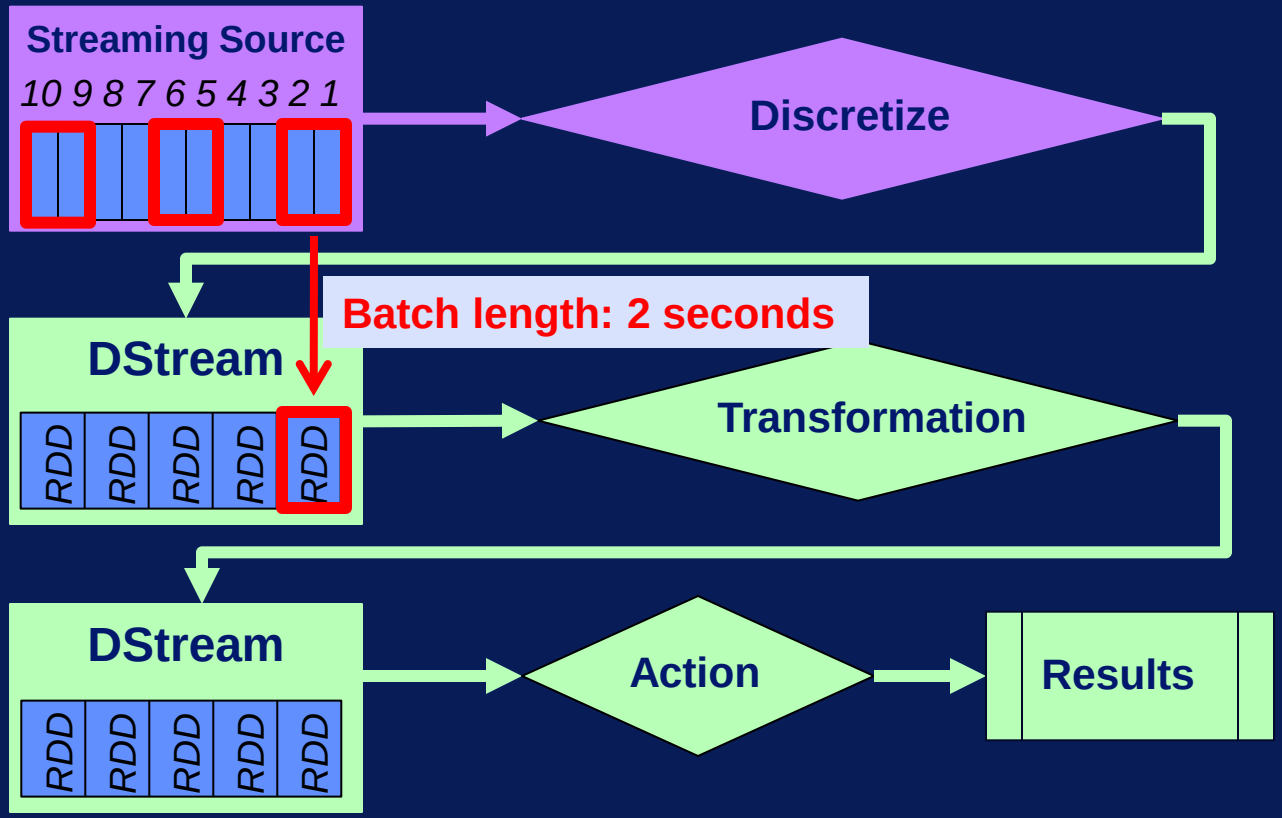
Spark Streaming Sources

- Kafka
- Flume
- HDFS
- S3
- Twitter
- Socket
- …etc.

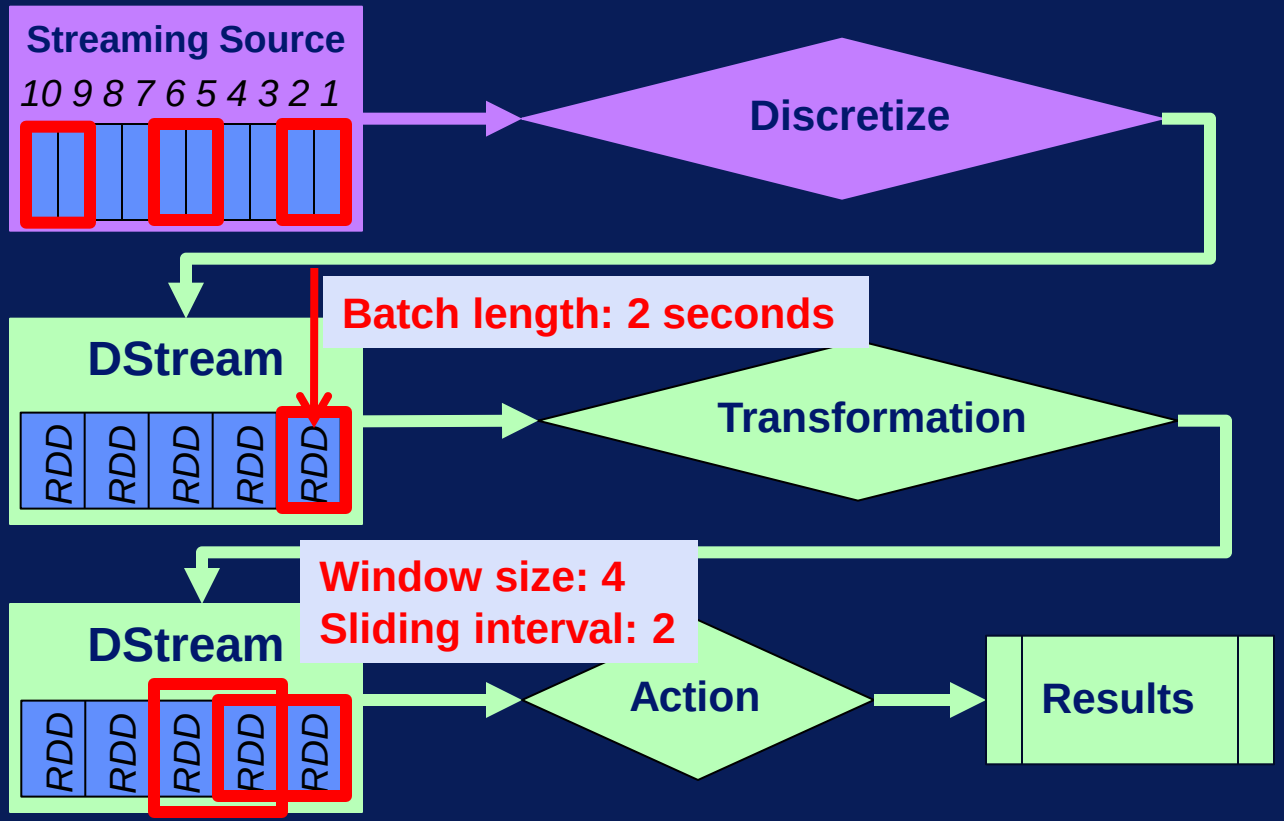
Creating and Processing DStreams



Creating and Processing DStreams



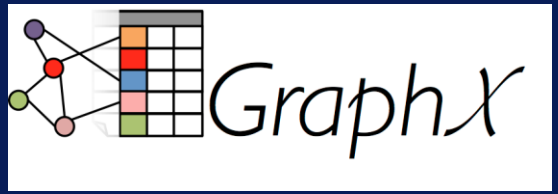
Creating and Processing DStreams



Main Take-Aways

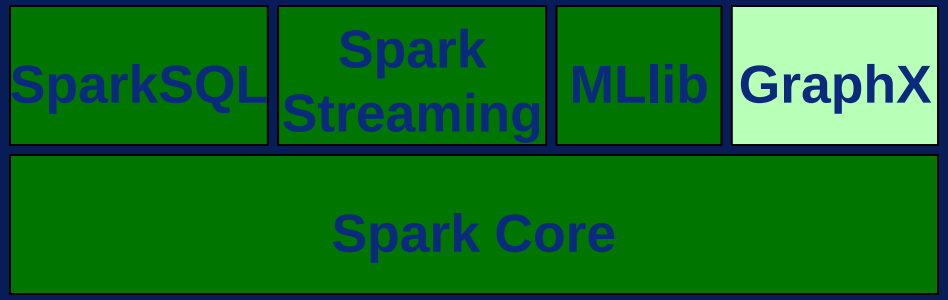
- Spark uses DStreams to make discrete RDDs from streaming data.
  - Same transformations and calculations applied to batch RDDs can be applied
- DStreams can create a sliding window to perform calculations on a window of time.

Spark GraphX



After this video you will be able to..

- Describe what GraphX is
- Explain how Vertices and Edges are stored
- Describe how Pregel works at a high level



Spark GraphX

GraphX is Apache Spark's API for graphs and graph-parallel computation.

GraphX uses a property graph model.

Both Nodes and Edges can have attributes and values

Properties → Tables

Vertex Table

Node properties

Edge Table

Edge properties

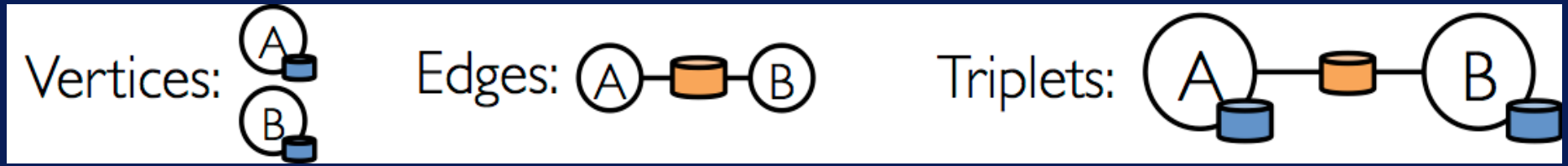
GraphX uses special RDDs

VertexRDD[A] extends RDD[(VertexID, A)]

EdgeRDD[ED, VD] extends RDD[Edge[ED]]

Triplets

The triplet view logically joins the vertex and edge properties.



http://spark.apache.org/docs/latest/img/triplet.png

Pregel API

Bulk-synchronous parallel messaging mechanism

Constrained to the topology of the graph

GraphX

Graph Parallel Computations

Special RDDs for storing Vertex and Edge information

Pregel operator works in a series of super steps