

# Lecture 1: Introduction to **Spark** and HPC

Shuo Zhou

COM6012: Scalable ML

# Week 1 Contents / Objectives

- The Big Data Problem: Why Spark?
- What is Spark?: The Essentials
- An Example of Spark: Log Mining
- How to Use Spark: PySpark, HPC, Resources

# Week 1 Contents / Objectives

- **The Big Data Problem: Why Spark?**
- What is Spark?: The Essentials
- An Example of Spark: Log Mining
- How to Use Spark: PySpark, HPC, Resources



# Where Does Big Data Come From?

- All happening **online**, e.g. tracking of:
  - Clicks
  - Billing events
  - Server requests
  - Transactions
  - Network messages
  - Faults
  - ...

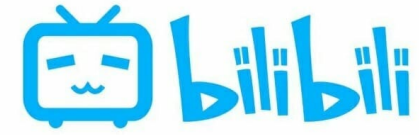


# Where Does Big Data Come From?

- User generated content: web + mobile



YouTube



TikTok

amazon

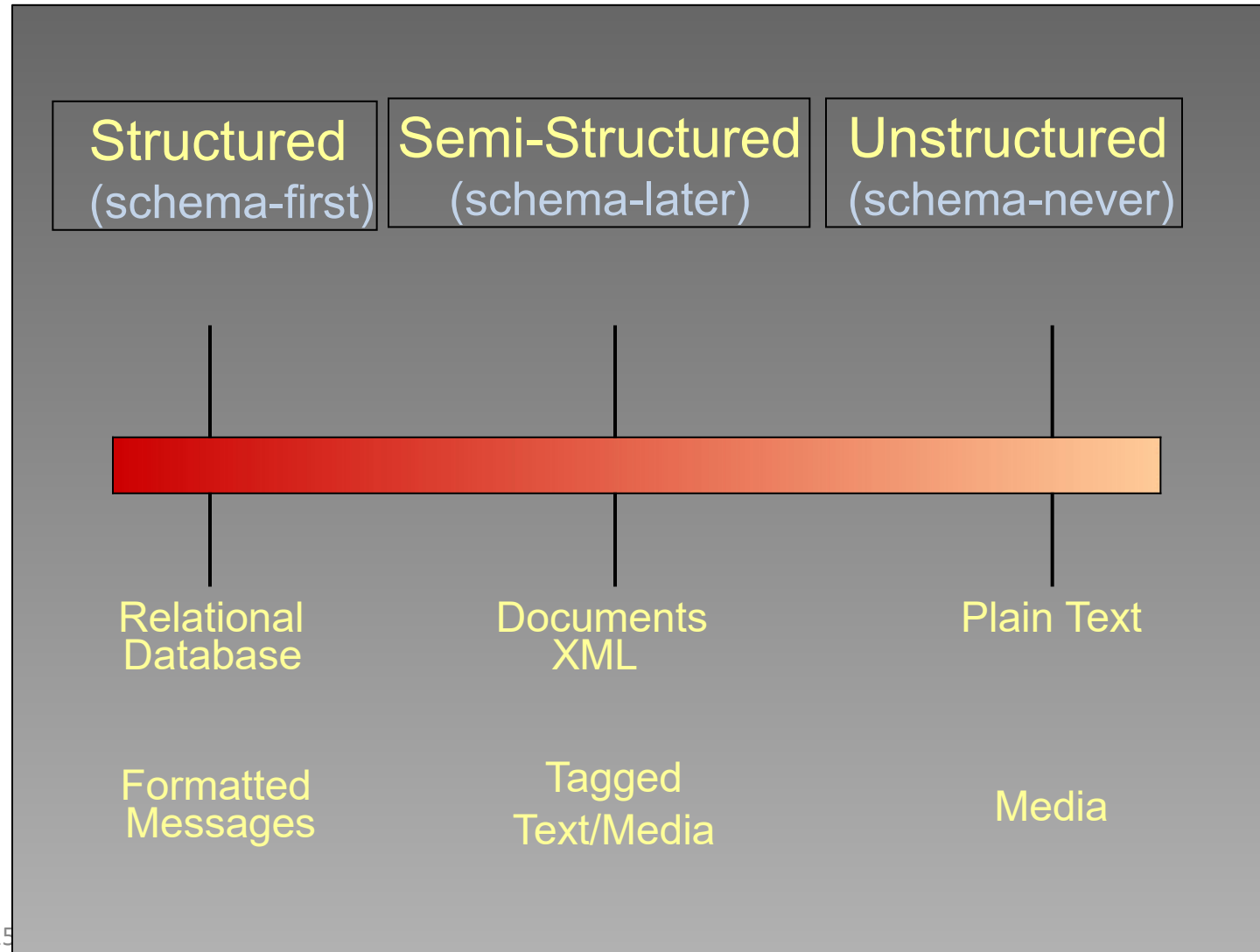


Instagram



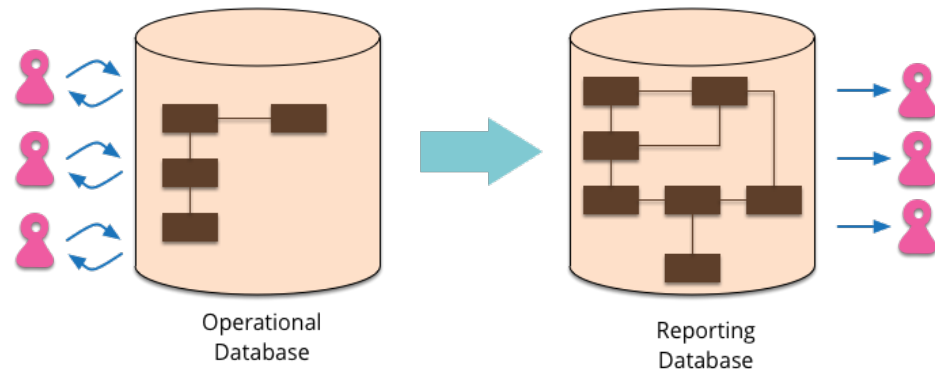
weibo

# Data Structure Spectrum



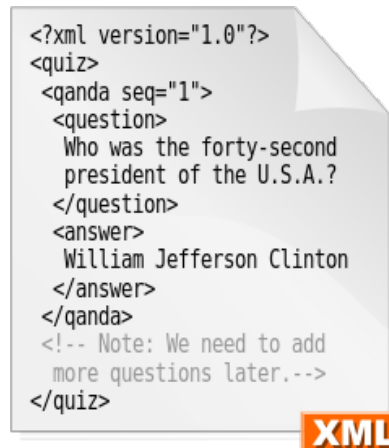
# Structured Data

- **Database:** relational data model → how a database is structured and used
- **Schema:** the organisation of data as a blueprint of how the database is constructed
  - The programmer **must statically specify** the schema
  - Decreasing ← consumer/media app, enterprise search
- **SQL:** Structured Query Language



# Semi-Structured Data

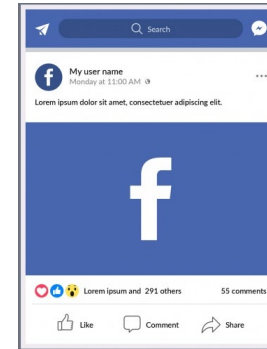
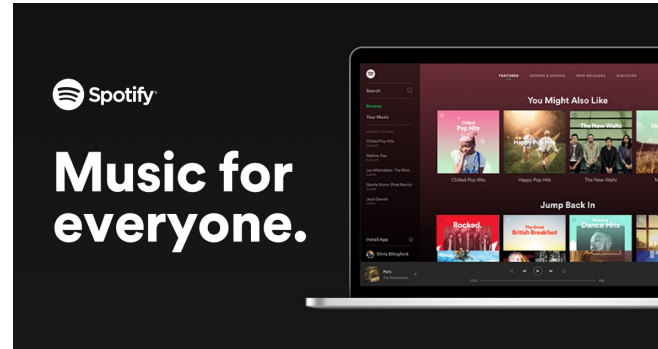
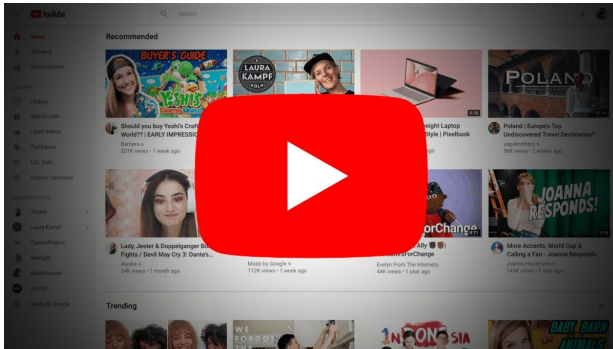
- **Self-describing** rather than formal structures, tags/markers to separate semantic elements
- The column types → the **schema** for the data
  - Spark dynamically infers the schema while reading each row
  - Programmer statically specifies the schema
- Examples:





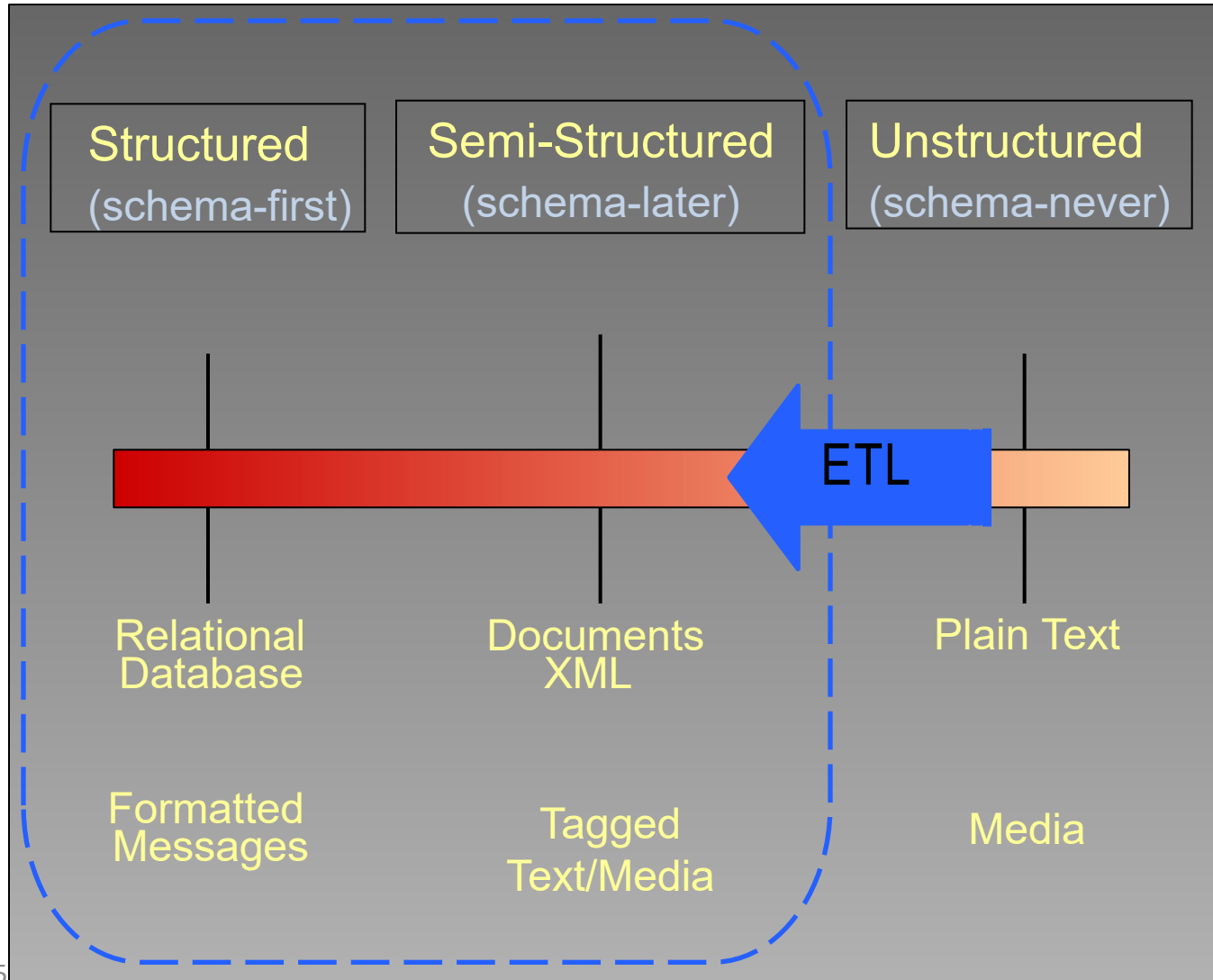
# Unstructured Data

- Only one column with string or binary type
- Examples



- Note: File formats  $\neq$  data structure

# Traverse the Data Structure Spectrum



- Impose structure on unstructured data
  - [Extract](#)
  - [Transform](#)
  - [Load](#)

# Traditional Analysis Tools

- Unix shell commands (awk, grep, ...)

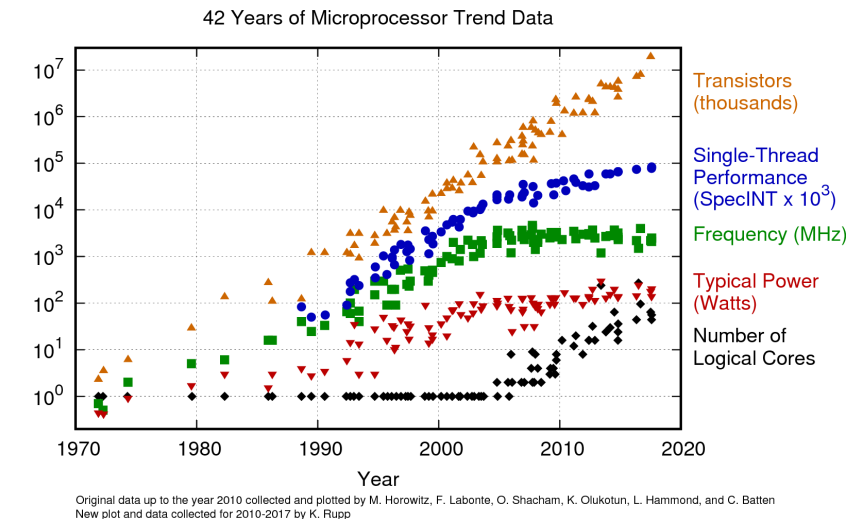
```
root@nginx:~# awk ' {print $0}' file.txt
Item      Model    Country    Cost
1         BMW      Germany    $25000
2         Volvo    Sweden     $15000
3         Subaru   Japan      $2500
4         Ferrari  Italy      $2000000
5         SAAB     USA        $3000
```

```
vulphere@arifuretaarch:~|⇒ grep root /etc/passwd
root:x:0:0:root:/root:/bin/zsh
vulphere@arifuretaarch:~|⇒ grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/zsh
vulphere@arifuretaarch:~|⇒ grep -c false /etc/passwd
3
vulphere@arifuretaarch:~|⇒ _
```

All run on a single machine!

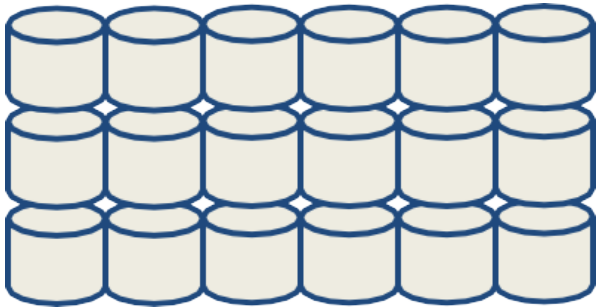
# The Big Data Problem

- Data growing faster than computation speeds
- Growing data sources
  - Web, mobile, scientific, ...
- Storage getting cheaper
- But, stalling CPU speeds and storage bottlenecks

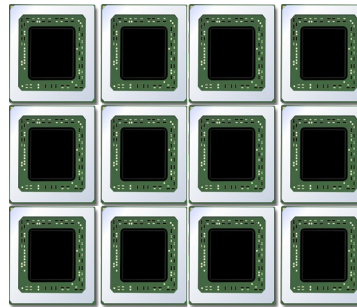


# Solution for the Big Data Problem

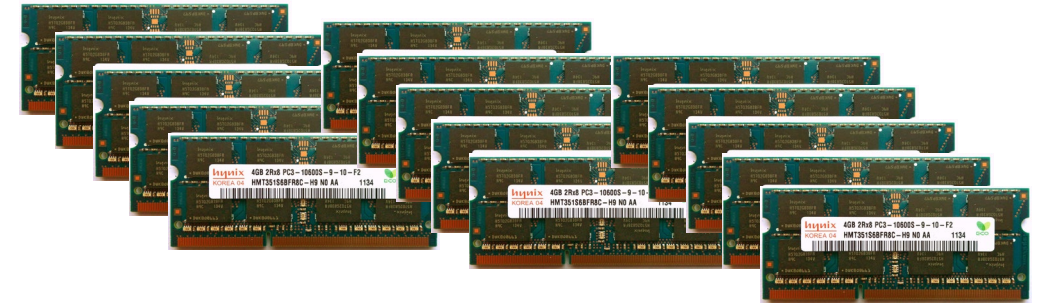
- **One machine** cannot process or *even store* all the data!
- Solution: **distribute** data over a **cluster** of machines



Lots of hard drives



... and CPUs



... and memory!



# Specifically in this module

# Week 1 Contents / Objectives

- The Big Data Problem: Why Spark?
- **What is Spark?: The Essentials**
- An Example of Spark: Log Mining
- How to Use Spark: PySpark, HPC, Resources

# Apache Spark

- Fast and general **cluster** computing system
- Interoperable with



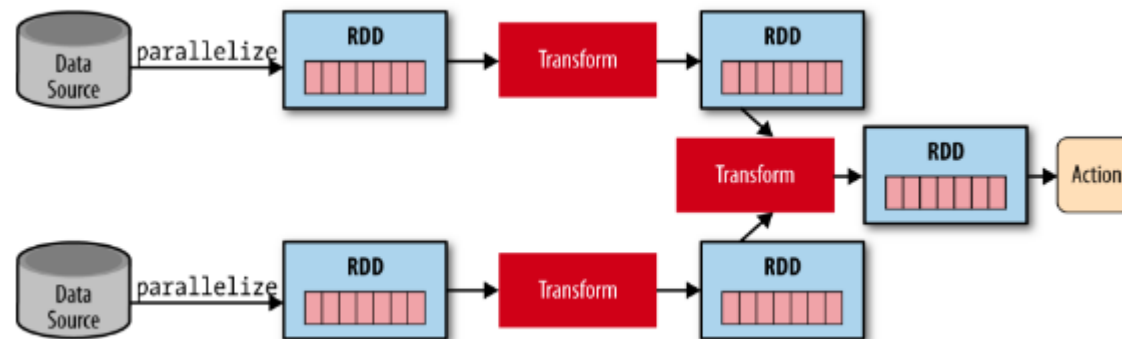
- Improves efficiency through:
  - **In-memory** computing primitives
  - General **computation graphs**
- Improves usability through:
  - Rich APIs in Scala, Java, **Python**
  - **Interactive shell**

→ Up to 100× faster  
(2-10× on disk)

→ 2-5× less code

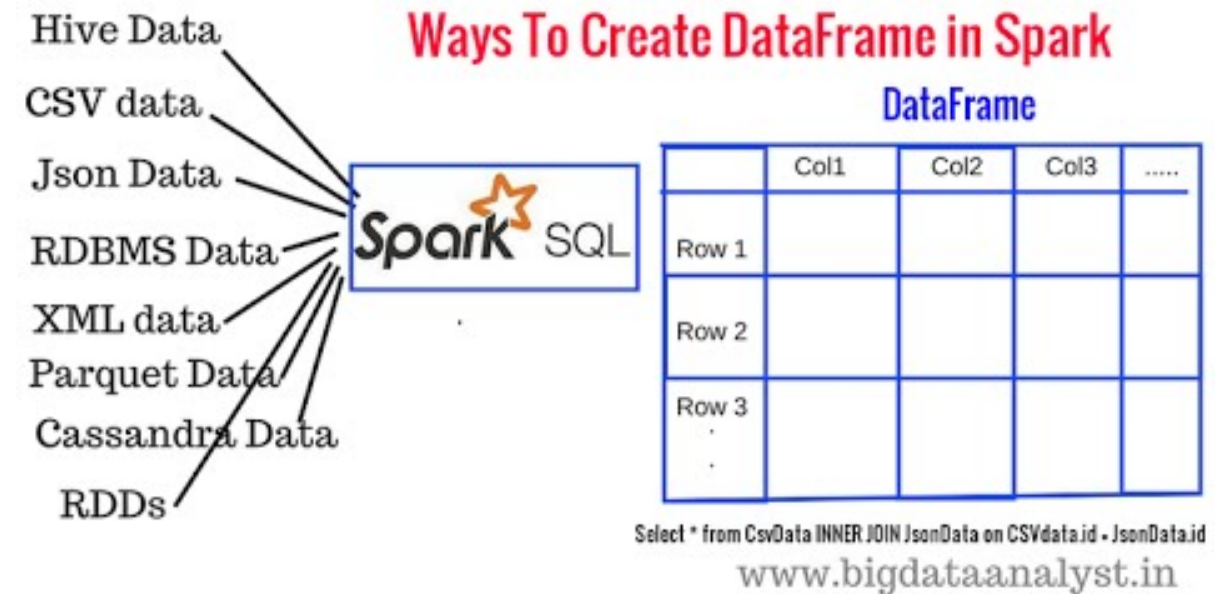
# Spark Model

- Write programs in terms of **transformations** on **distributed** datasets
- Resilient Distributed Datasets (RDDs)
  - **Collections** of objects that can be stored in memory or disk across a cluster
  - **Parallel** functional transformations (map, filter, ...)
  - Automatically rebuilt on **failure**



# Spark for Data Science

- DataFrames
  - Structured data (SQL)
  - Familiar API based on R/Python Pandas
  - Distributed, optimised implementation
- Machine learning pipelines
  - Simple construction and tuning of ML workflows





# Spark Computing Framework

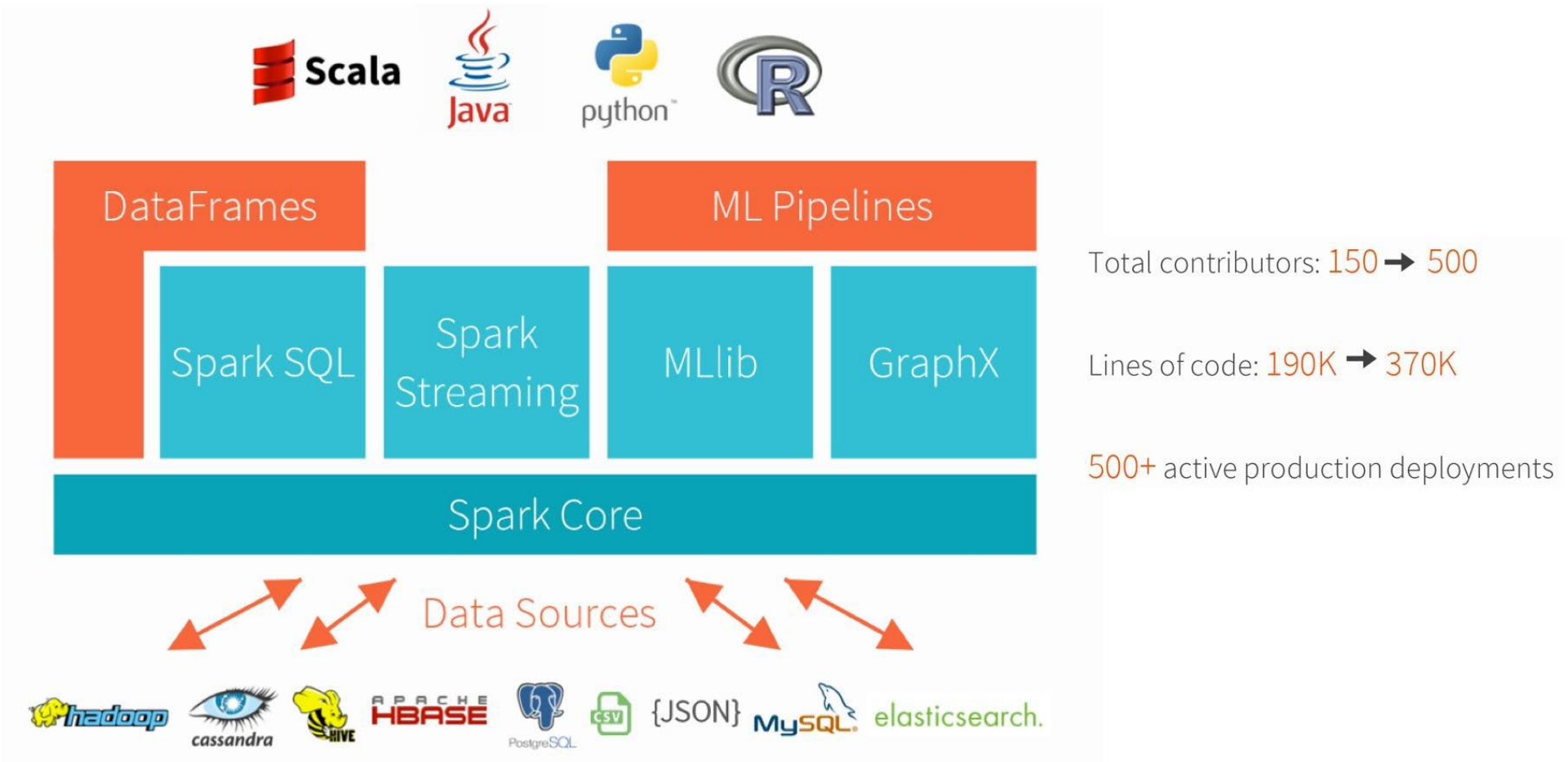
- Programming abstraction and parallel runtime to hide complexities of **fault-tolerance** and **slow** machines

“Here’s an operation, run it on all of the data”

**JUST DO IT.**

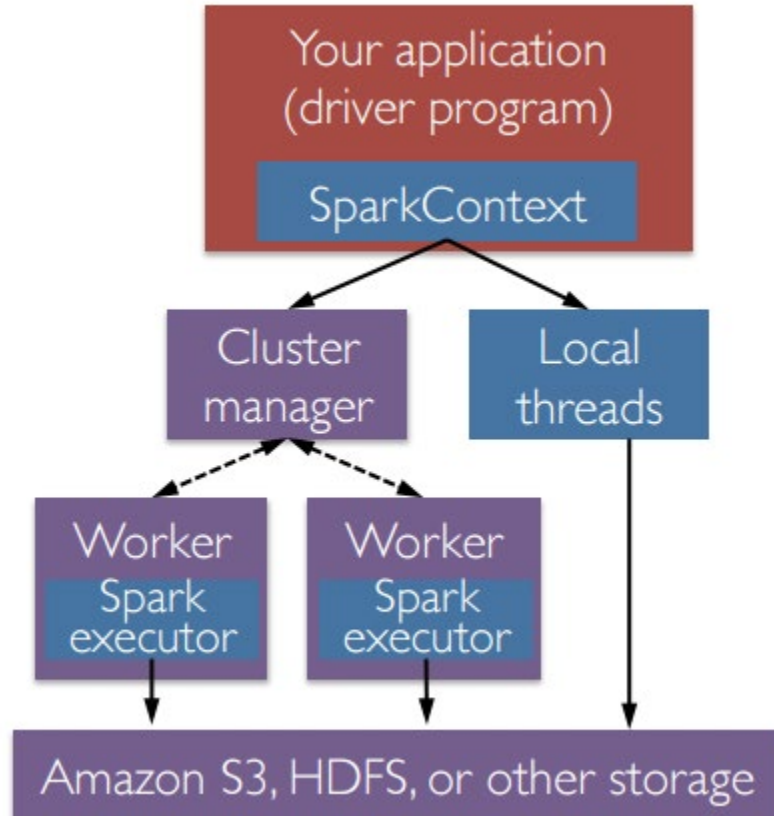
- Don’t care where it runs (you schedule that)
- In fact, feel free to run it twice on different nodes (e.g. when it fails)

# Apache Spark Ecosystem



<https://i.pining.com/originals/e7/f3/2d/e7f32d041846a5938a09e192bdf3885d.jpg>

# Spark Components



- A Spark program first creates a **SparkSession** object as the driver (including **SparkContext**)
  - Tells Spark how/where to access a cluster
  - Connect to cluster managers
- Cluster managers
  - Allocate resources across applications
- Spark executor (worker):
  - Run computations
  - Access data storage

# SparkSession and SparkContext

- SparkSession

- Entry point for DataFrame API, create **DataFrames**
- PySpark shell automatically create SparkSession as **spark**
- Programs: must create a new SparkSession first (see lab)

- SparkContext

- Entry point for Spark functionality, create **RDDs**
- Connect to a Spark cluster
- Associated with a SparkSession
- PySpark shell automatically create SparkContext as **sc**
- Programs: **sc = spark.sparkContext**

# The 'Master' Parameter for a SparkSession

- Determines cluster type and size

| Master Parameter  | Description   |
|-------------------|---|
| local             | run Spark locally with one worker thread (no parallelism)                       |
| local[K]          | run Spark locally with K worker threads (ideally set to number of cores)        |
| spark://HOST:PORT | connect to a Spark standalone cluster; PORT depends on config (7077 by default) |
| mesos://HOST:PORT | connect to a Mesos cluster; PORT depends on config (5050 by default)            |



# Week 1 Contents / Objectives

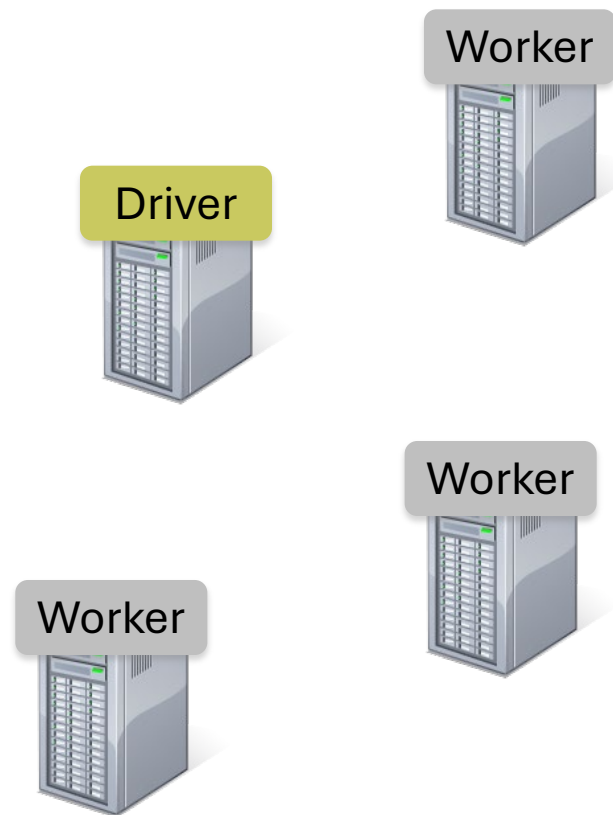
- The Big Data Problem: Why Spark?
- What is Spark?: The Essentials
- **An Example of Spark: Log Mining**
- How to Use Spark: PySpark, HPC, Resources

# Spark Example: Log Mining (w/t RDD)

Load error messages from a log into memory, then interactively search for various patterns

# Spark Example: Log Mining

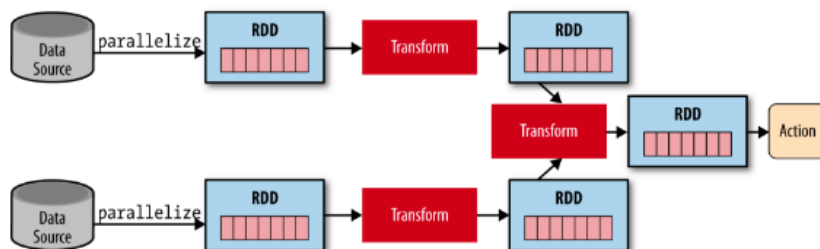
Load error messages from a log into memory, then interactively search for various patterns



# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
```



Driver

Worker

Worker

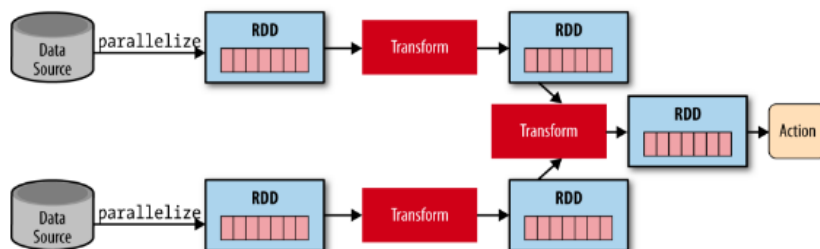
Worker

# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Base RDD

```
lines = spark.textFile("hdfs://...")
```



Driver

Worker

Worker

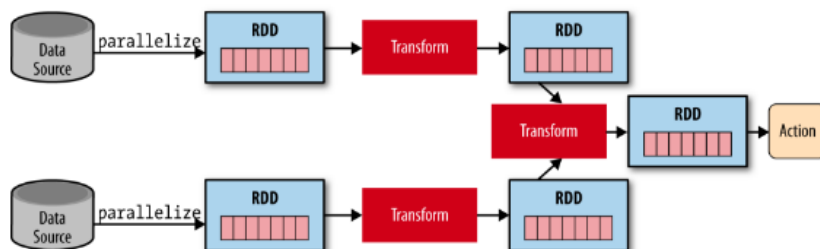
Worker



# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))
```



Driver

Worker

Worker

Worker

# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Transformed RDD

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))
```

Driver

Worker

Worker

Worker

# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

Driver

Worker

Worker

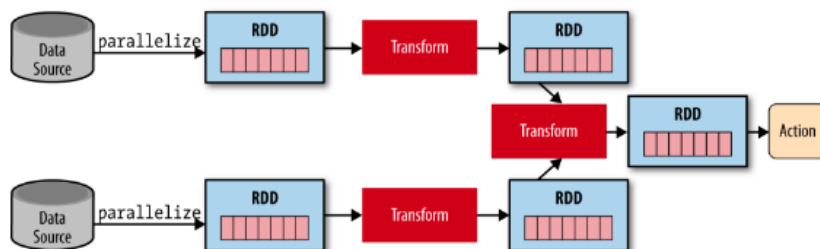
Worker

# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```



Driver

Action

Worker

Worker

Worker

# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

Driver

A server rack icon representing the Spark Driver node.

Worker

Block 1

A server rack icon representing a Spark Worker node.

Worker

Block 2

A server rack icon representing a Spark Worker node.

Worker

Block 3

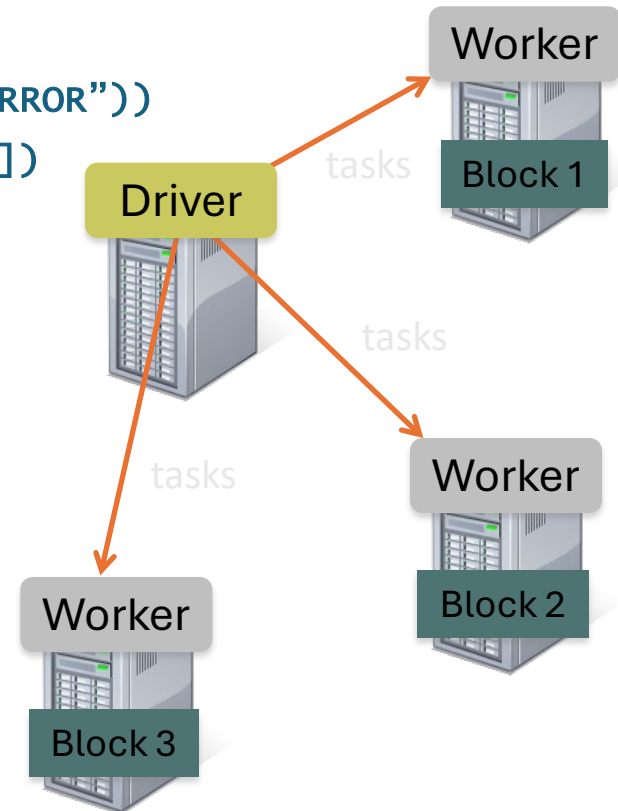
A server rack icon representing a Spark Worker node.

# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

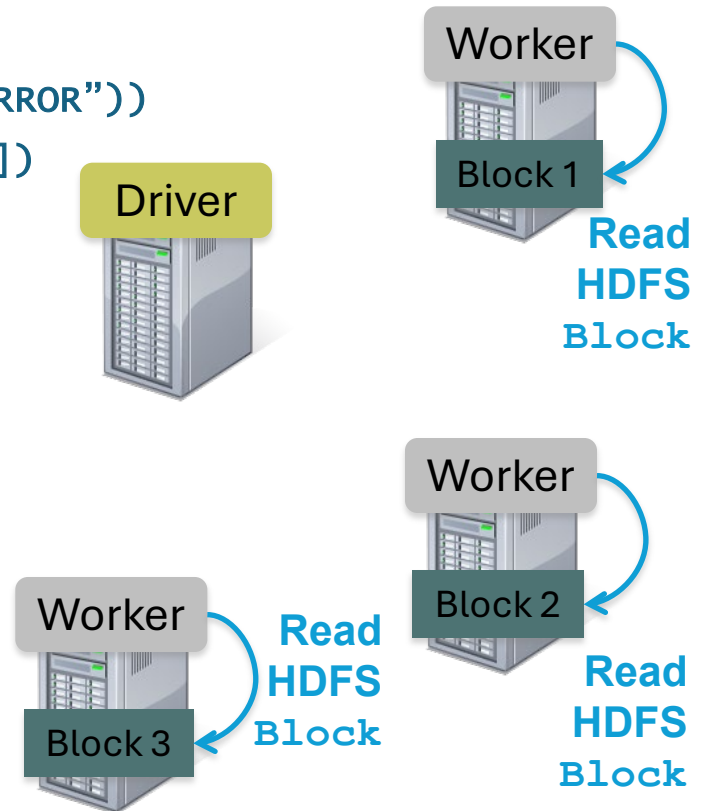


# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

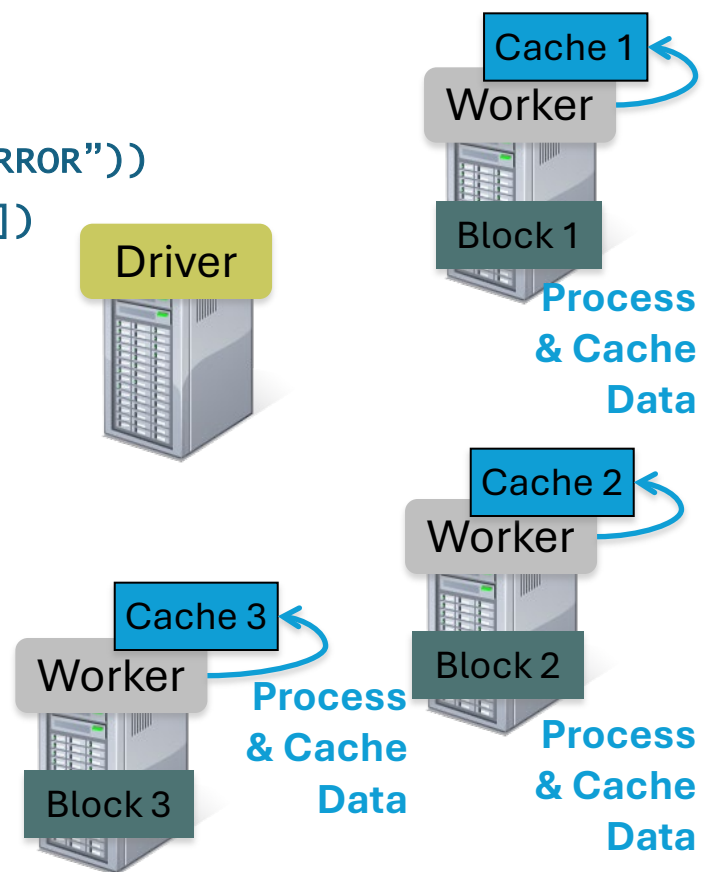


# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```



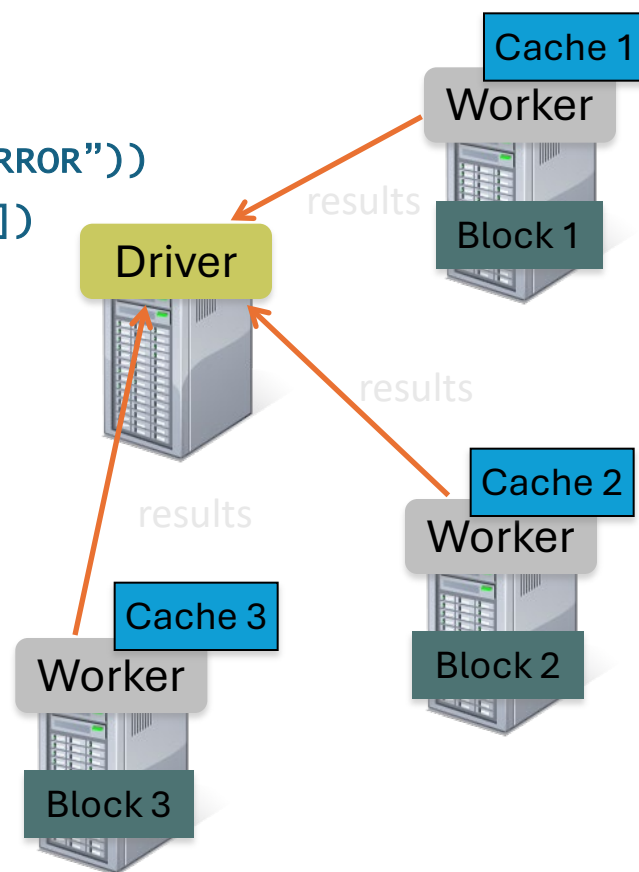


# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

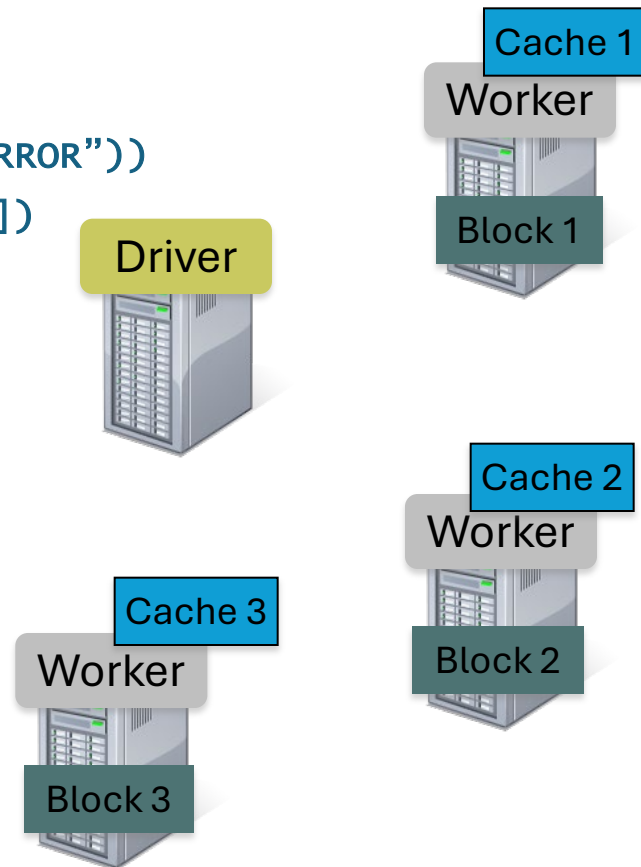


# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

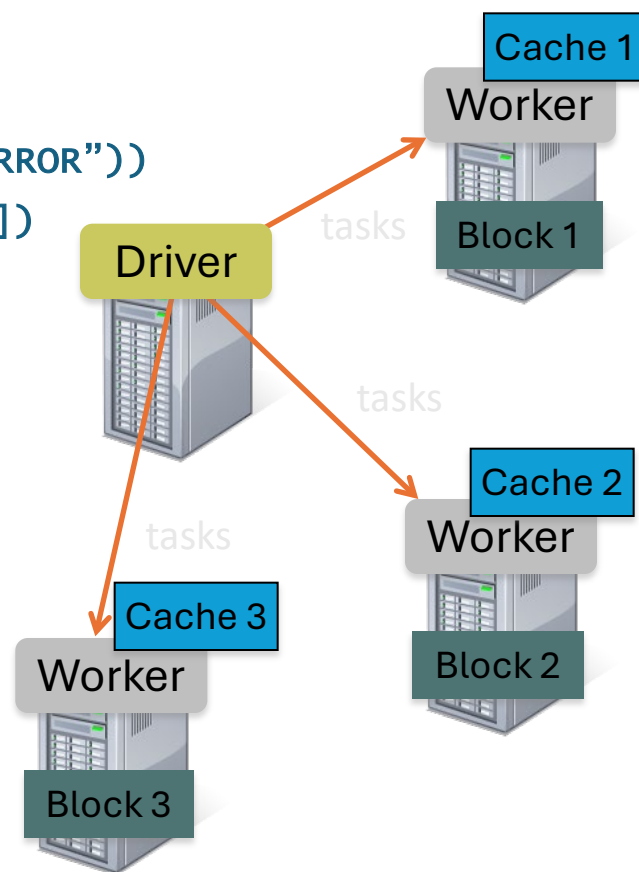


# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

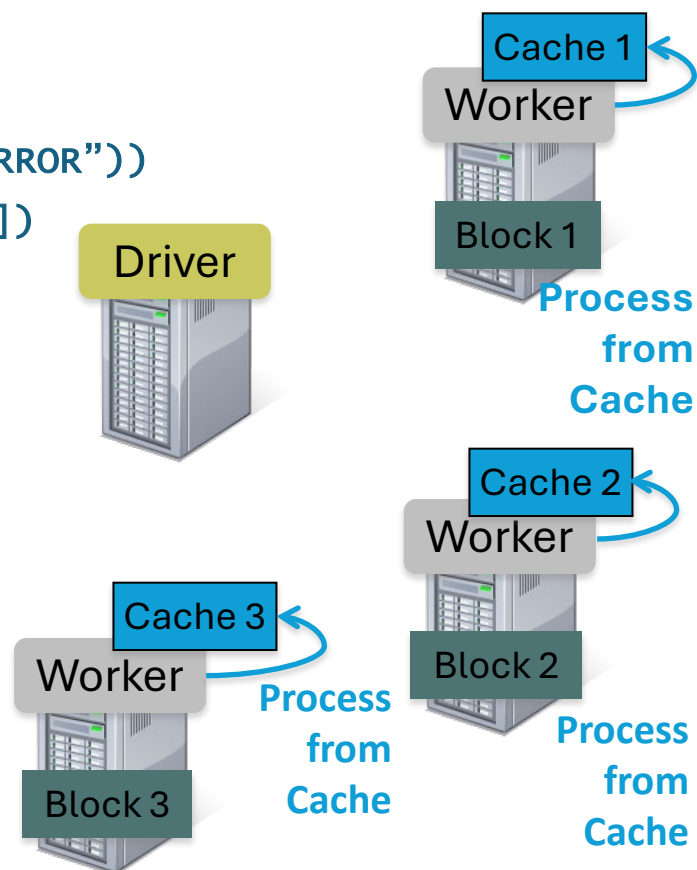


# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

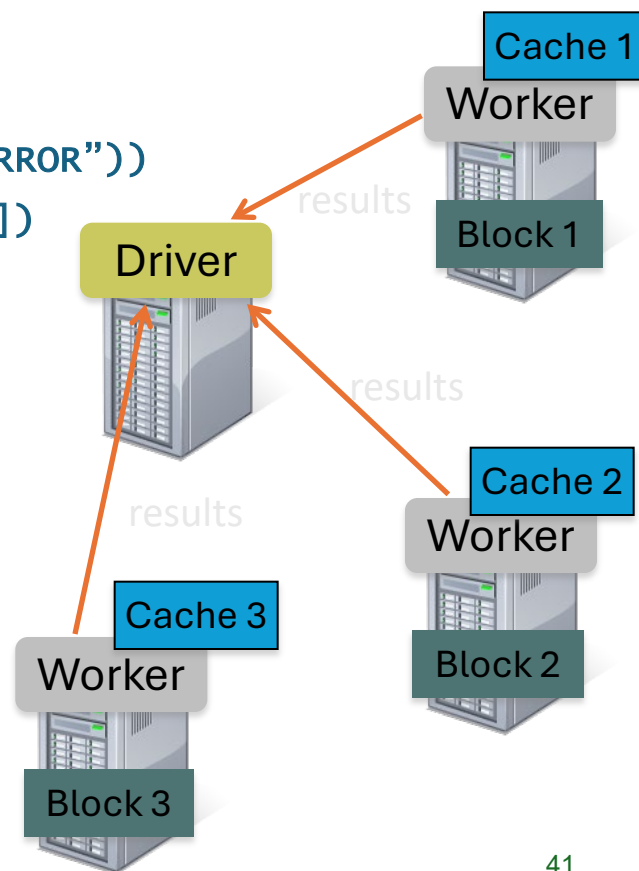


# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()  
messages.filter(lambda s: "php" in s).count()
```



# Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

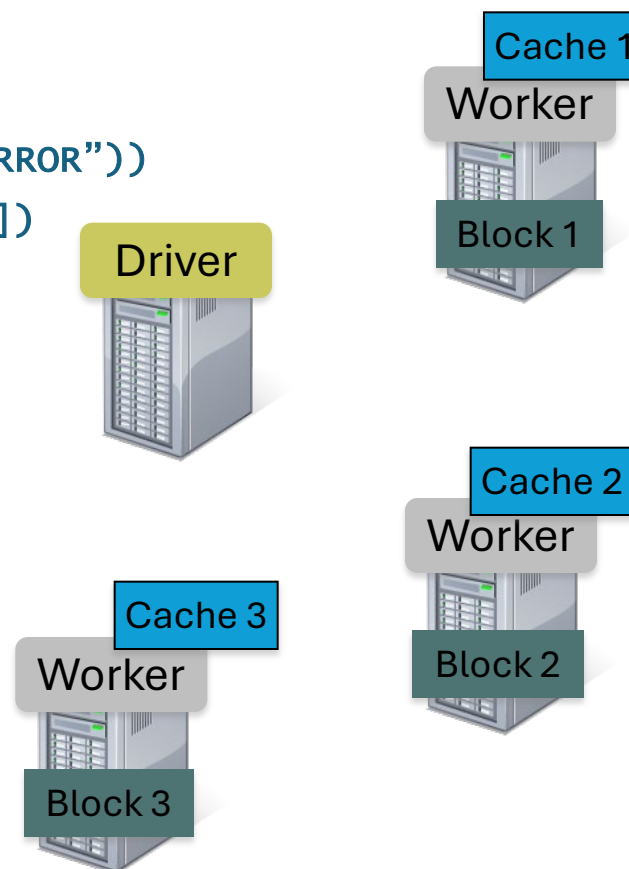
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

**Cache** your data → Faster results

*Full-text search of Wikipedia*

- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk



# Week 1 Contents / Objectives

- The Big Data Problem: Why Spark?
- What is Spark?: The Essentials
- An Example of Spark: Log Mining
- **How to Use Spark: PySpark, HPC, Resources**

# Spark Program Lifecycle

- Create DataFrames from external data or [createDataFrame](#) from a collection in a driver program
- Lazily transform them into new DataFrames
- `cache()` some DataFrames for reuse
- Perform actions to execute parallel computation and produce results

Use Spark Transformations and Actions wherever possible: Search [DataFrame reference API](#)

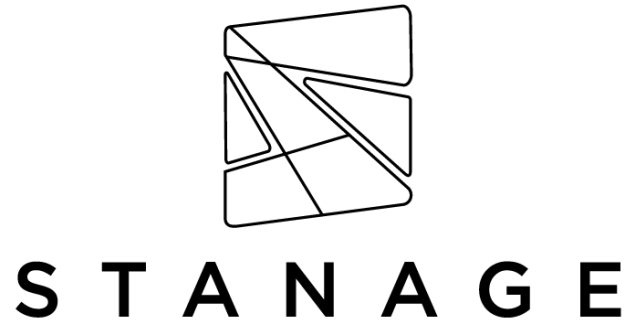


# PySpark 3.5.4

- Need: **Java**, Python, Spark
- See lab 1 on how to install on HPC
- To install on Windows (optional)
  - [Lab 1 instructions](#): Install Java JRE, Python, Spark
  - Or pip install pyspark==3.5.4
- To install on Linux/Mac (optional): see lab references

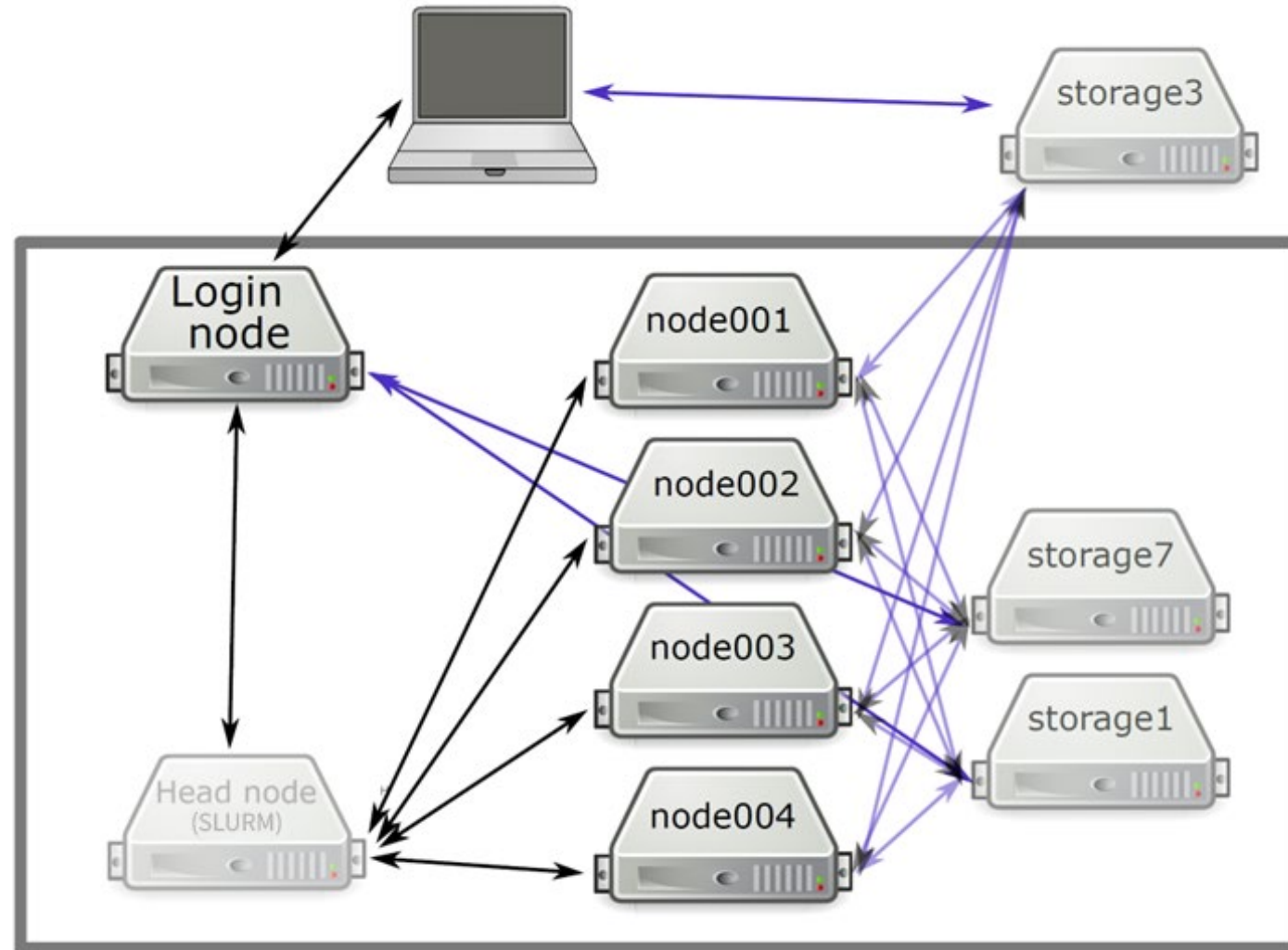


# University of Sheffield's HPC



- VPN: a **MUST** for the first time
- Account created for you already!
- Training (due 13<sup>th</sup> Feb Thursday-AS0): HPC Driving License test
- SSH access via [stanage.sheffield.ac.uk](https://stanage.sheffield.ac.uk)
  - Windows: MobaXTerm
  - Linux/MAC OS: terminal (command line)

# HPC Cluster Structure



# Storage

| Location         | Quota | Speed | Suitable for?                 |
|------------------|-------|-------|-------------------------------|
| /users/\$USER    | 50GB  | >     | Personal data                 |
| /mnt/parscratch/ | -     | >>>   | Temporary large files         |
| /tmp             | -     | >>>   | Temporary lots of small files |



# Interactive Session

```
(myspark) pyspark
Python 3.11.7 (main, Dec 15 2023, 18:12:31) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
24/01/30 16:47:47 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicab
le
Welcome to

      /_/_/_/_/_\
     /  V  _V_  X  \
    /_/_/_/_/_\_/_/_\ version 3.5.0
   /_/_/_/_/_\_/_/_\

Using Python version 3.11.7 (main, Dec 15 2023 18:12:31)
Spark context Web UI available at http://node001.pri.stanage.alces.network:4040
Spark context available as 'sc' (master = local[*], app id = local-1706633268587).
SparkSession available as 'spark'.
>>>
```

# Batch Session – Shell Script xx.sh

Create a file `Lab1_SubmitBatch.sh`

```
#!/bin/bash
#SBATCH --nodes=1 # Specify a number of nodes
#SBATCH --mem=5G # Request 5 gigabytes of real memory (mem)
#SBATCH --output=../Output/COM6012_Lab1.txt # This is where your output and errors are logged
#SBATCH --mail-user=username@sheffield.ac.uk # Request job update email notifications

module load Java/17.0.4

module load Anaconda3/2022.10

source activate myspark

spark-submit ../Code/LogMiningBig.py
```

# Batch Session: Submit & Relax

- `sbatch` your job (can run at the login node): see Lab 1
- Then?
  - Close the terminal and leave
  - Wait for pre-set email notification
  - Check status: `squeue`
  - Cancel job: `scancel`
- How much resources to request
  1. Run `short` test jobs
  2. View resource utilisation
  3. Extrapolate
  4. Submit larger jobs



# Spark Resources

- [Apache Spark Documentation](#)
- [PySpark tutorial](#)
- [Spark videos on YouTube](#)
- [Open-source code](#)
- Suggested reading in labs

## Suggested reading:

- [Spark Overview](#)
- [Spark Quick Start](#) (Choose **Python** rather than the default *Scala*)
- Chapters 2 to 4 of [PySpark tutorial](#) (several sections in Chapter 3 can be safely skipped)
- Reference: [PySpark documentation](#)
- Reference: [PySpark source code](#)



# Acknowledgements

- Some slides (sec. 1) are modified from the “[Introduction to Apache Spark](#)” course by Prof. A. D. Joseph, University of California, Berkeley.
- This module benefits from many open resources. See the acknowledgement on our [GitHub page](#).
- There are many other resources that I have consulted but may somehow lost track of the origins.



Thank-you-word-cloud.jpg (967x522) (wikimedia.org)