Chapter 8
Data-Intensive Computing
MapReduce Programming

Mastering Cloud Computing
Paul Talaga

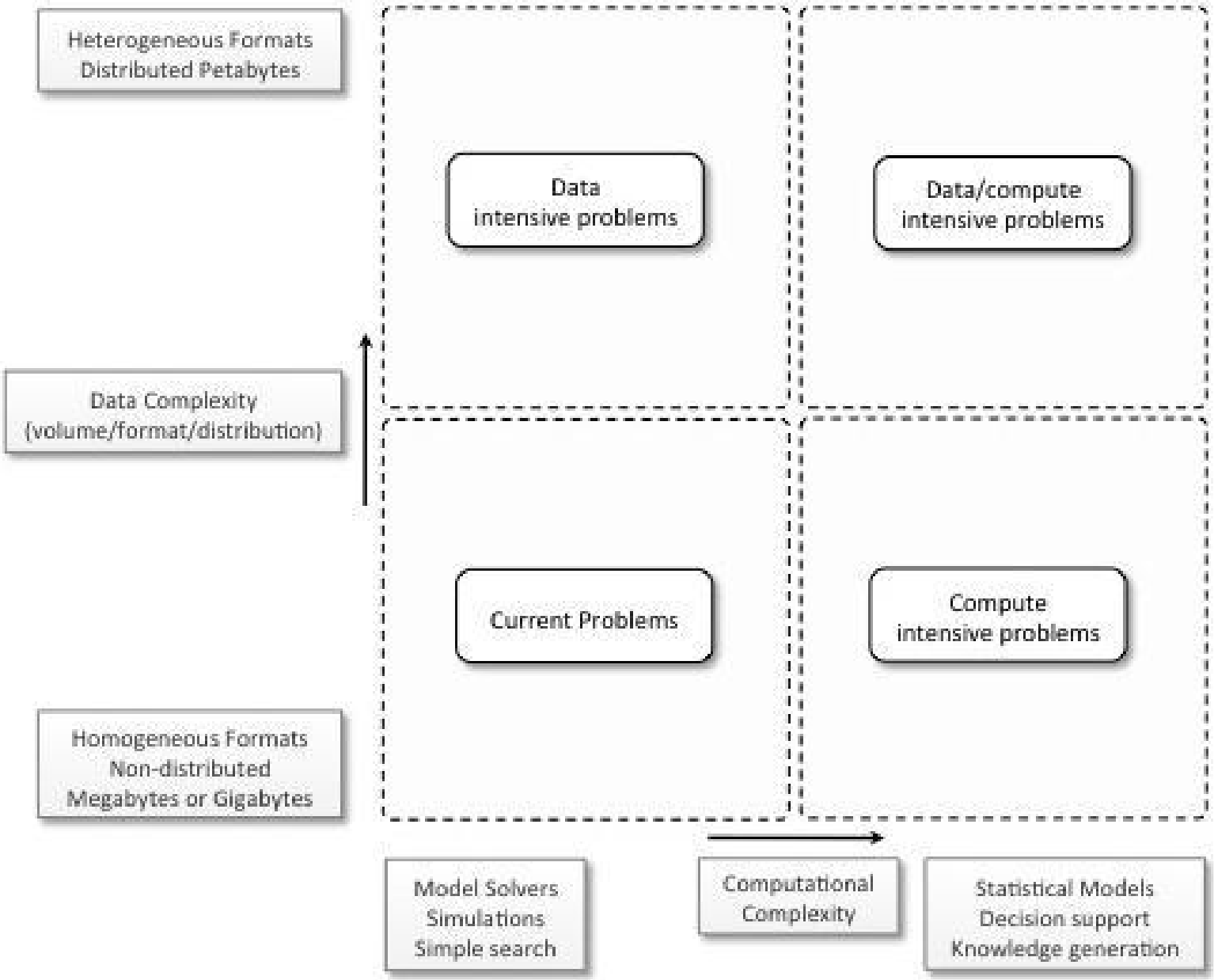
UNIVERSITY OF Cincinnati

What is data-intensive computing?

- Production, manipulation, and analysis of 100 MB -> 100 PB and beyond.
- Where does the data come from?
  - computational science - telescopes, bioinformatics, high-energy physics (colliders)
  - IT/commercial - cell phone info, buying habits, browsing habits, social media
    - Cell phone data (US) 8PB/month (2012) 327 by 2015
    - Google 24PB/day processing
    - Facebook 36PB
    - Zynga (Farmville & Frontierville) 1PB per day, +1000 servers per week to store data

UNIVERSITY OF Cincinnati

Computation Landscape



Challenges

- Data doesn’t fit on one machine.
- Data locality!
- Data partitioning
- Scalable algorithms
- Metadata management
- Large in-memory data structures
- High performance distributed file systems
- Methods to move computation to data
- Streaming data
- Software integration over diff platforms

UNIVERSITY OF Cincinnati

Some History...

- Began with TCP/IP in 1991 - remote visualization of data
- Wide Area Large Data Object (WALDO) in 1998 - auto metadata, catalog, real-time processing, and storage
- MAGIC project (DARPA) - distributed parallel storage system - TerraVision
- Clipper - research institution to build shared data computing

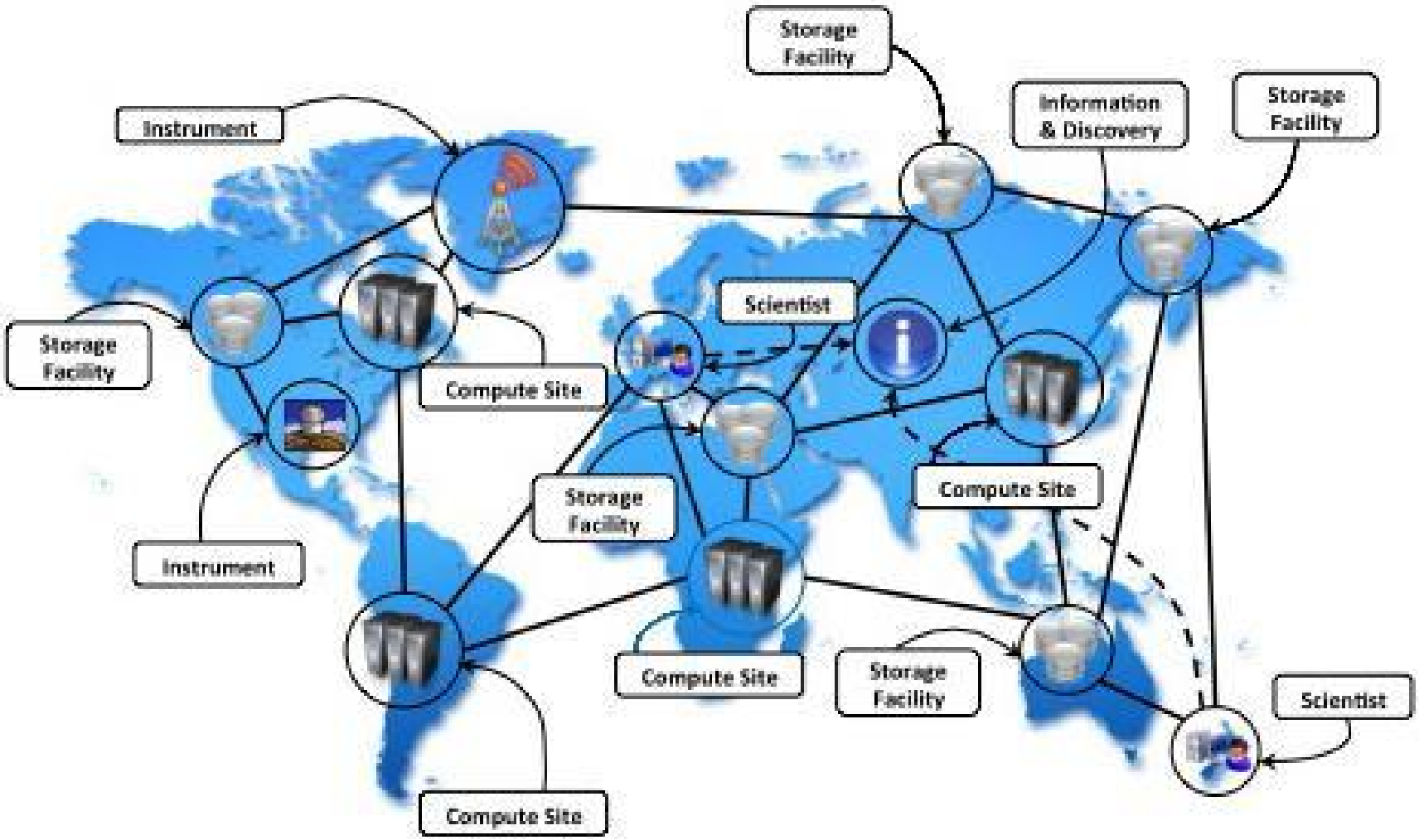
UNIVERSITY OF Cincinnati

Then… Data Grids

- Shared, heterogeneous resources, for scientific research
- High performance data-transfer
- Scalable replicas
- Characteristics:
  - Massive datasets - LHC - 40GB/min
  - Shared data collections - read/write
  - Unified namespace
  - Access restrictions - authentication

UNIVERSITY OF Cincinnati

Data Grid Architecture



UNIVERSITY OF Cincinnati

Big Data

- Data Grids…. for business.
- Log analysis popular
- Big Data - Computations on data that do not fit on a single machine/RDBMS or using typical analysis tools.
- Ex: Weblogs, sensor networks, search indexing, call records, surveillance, medical records, etc……
- Storage of time may be important

UNIVERSITY OF Cincinnati

How Can Cloud Computing Help?

- On-demand compute instances
- Flexible, non-structured data storage
- APIs to build scalable software

So… build a Big Data system on a cloud!

- MapReduce (Google) - Google File system
- Hadoop - HDFS
- Sector - SDFS
- Greenplum

UNIVERSITY OF Cincinnati

Why not Relational Databases?

- Mature technology….
- Distributed databases exist..
- Very robust, transactions, query optimization, efficient resource management….

Data must be in relational model.
ACID compliance limits ability to scale.
Limited tunability.

UNIVERSITY OF Cincinnati

Large Data Storage is Key!

- Shift away from RDBMS
- Why?
  - Popularity of Big Data
  - Importance of Data Analytics in business
  - Unstructured data
  - On-demand/burst loads
  - New technologies
  - Re-evaluation of one-size-fits-all RDBMS model
  - Re-evaluation of existing computing paradigms

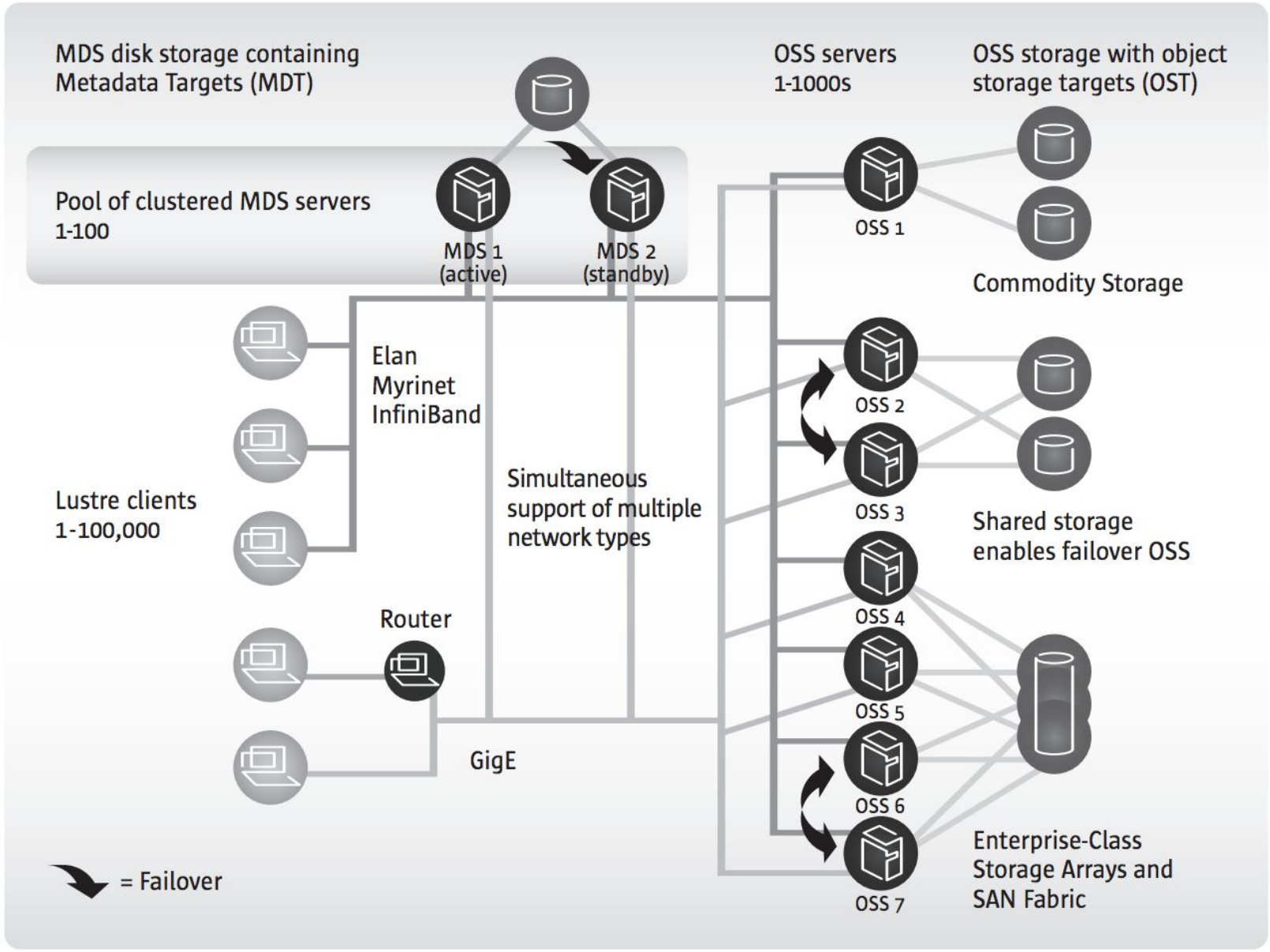
UNIVERSITY OF Cincinnati

Distributed File Systems

- File-level access
- Vary in features… access control, concurrency, latency, max size, etc…
- Ex:
  - Lustre (Linux + Cluster) - From 1999
    - POSIX compliant
    - Scales to many PB
    - Object Servers & Metadata Servers
    - Parallel I/O
    - Many of top 100 supercomputers

UNIVERSITY OF Cincinnati

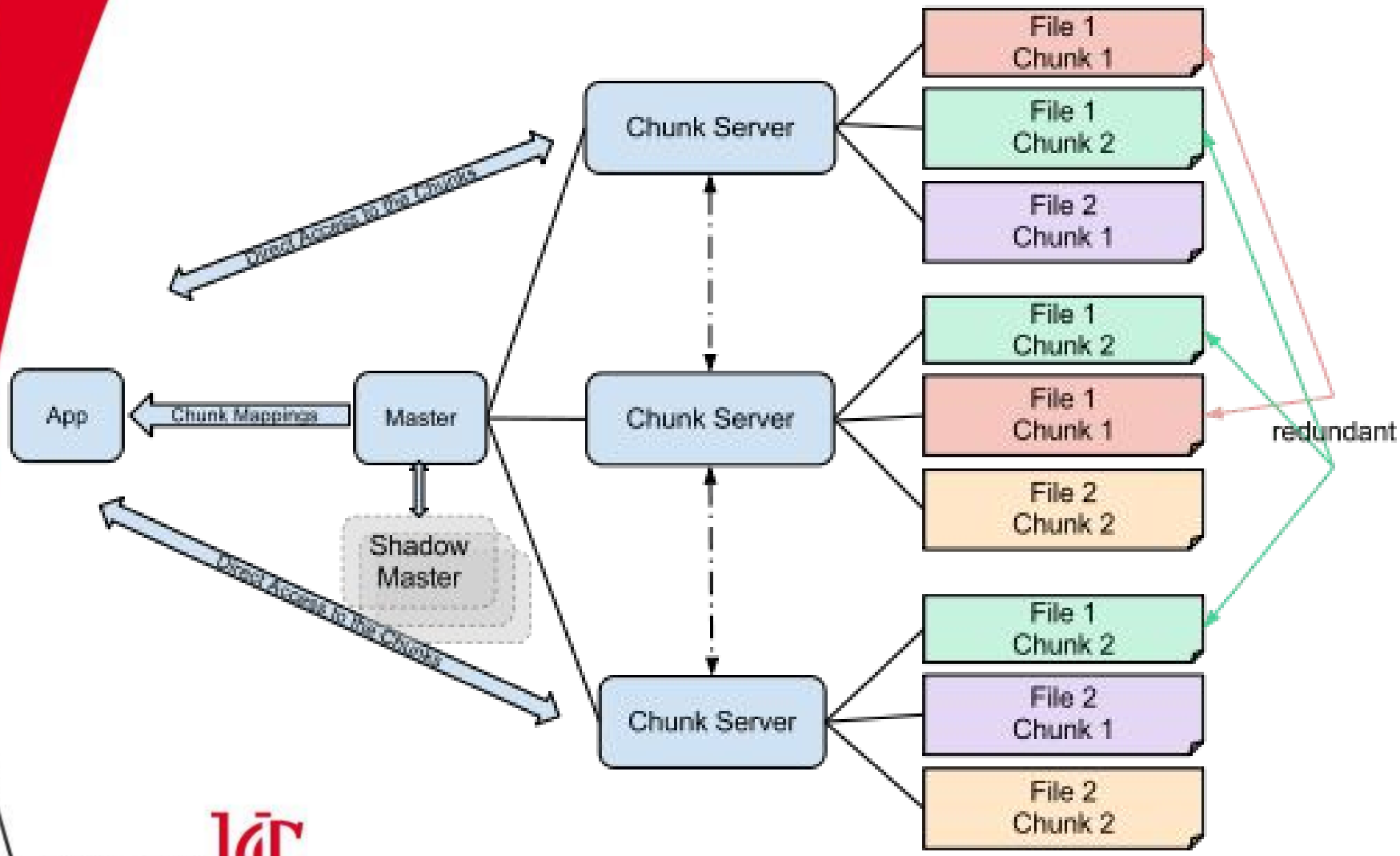
Lustre Architecture



Distributed File Systems

- Ex (cont):
  - IBM General Parallel File System (GPFS)
    - Shared disk idea (big RAID)
    - Shared metadata
    - No single point of failure
  - Google File System (GFS) - specialized
    - Fault-tolerant, highly available
    - Commodity hardware
    - Designed for multi-GB files
    - Bandwidth over latency
    - Optimized for append
    - Redundant chunks, snapshots

GFS



UNIVERSITY OF Cincinnati

Distributed File Systems

- Ex (cont):
  - Sector - for Sphere framework
    - Optimized for WAN
    - Replicates whole files (customizable)
    - UDT connections (UDP)
  - Amazon Simple Storage Service (S3)
    - Highly available, reliable, scalable, low latency, infinite storage(?!?)
    - REST interface
    - POSIX library available
    - (hidden architecture)
  - HDFS - Hadoop Distributed File System

DFS Commonalities

- Metadata/master nodes - filenames, permissions, location of chunks
- Data nodes - store data

Clients talk directly to data nodes!

Scalable read/writes

Metadata changes not scalable

Other Large Storage Systems.. NoSQL!

- Related to DFS, but more general
- (See Databases at Scale slides)
- Ex:
  - CouchDB, MongoDB
  - Amazon Dynamo (DynamoDB)
  - Apache Cassandra
  - Hadoop HBase
  - Many many more.

UNIVERSITY OF Cincinnati

Data Storage Summary

- RDBMS work well for some datasets
  - Unstructured data difficult
  - Scalability issues
  - Easy to use - developer friendly
- Distributed File Systems / NoSQL
  - Loose relational features
  - Gain scalability & reliability
  - Lower hardware requirements (commodity)
  - Resilient to hardware failures

UNIVERSITY OF Cincinnati

Programming Platforms

- *Many* older systems require explicit data movement and computation (tasks)
- *MapReduce* abstracts data location, failures, and tasks away.
  - Use an old idea from Functional Programming

In python….

```
map(fn, list) -> list
reduce(fn, list) -> value
```

UNIVERSITY OF Cincinnati

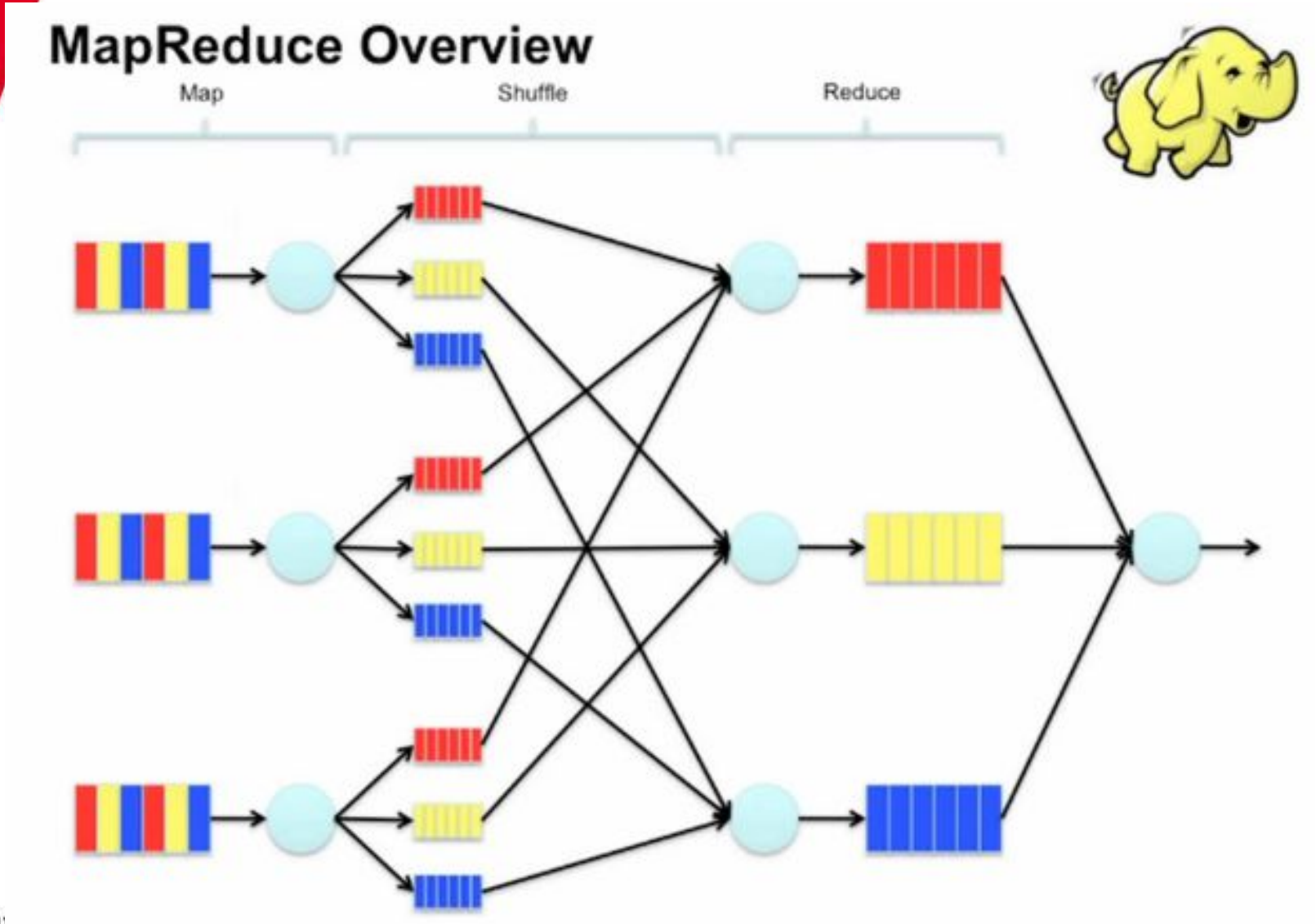
Programming Platforms

- In Hadoop, we use key/value pairs

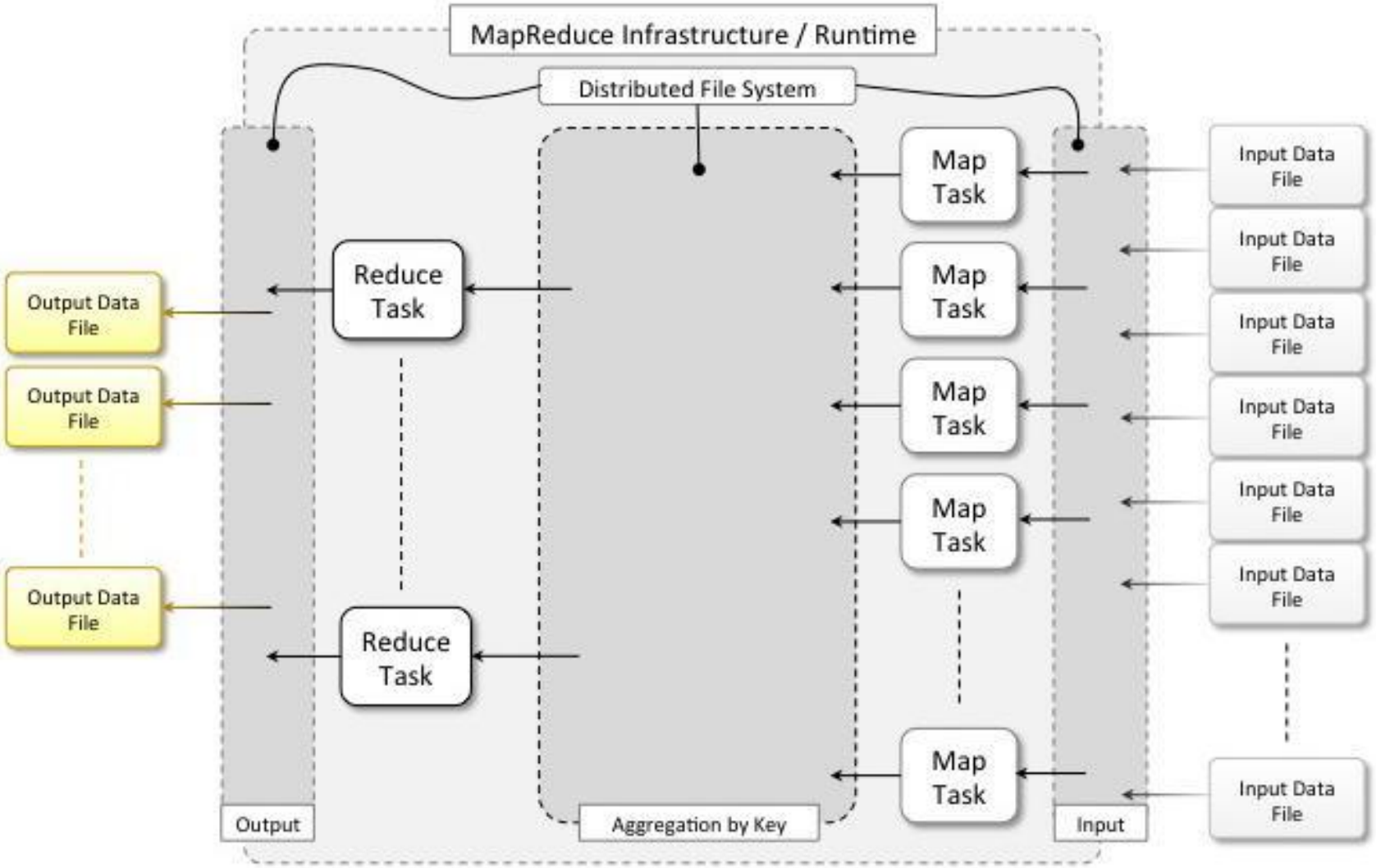
```
map(k1, v1) -> list(k2, v2)
reduce(k2, list(v2)) -> list(v3)
```

UNIVERSITY OF Cincinnati

Pictorially - (Hadoop)



Pictorially 2



Examples of MapReduce Problems

- Distributed grep
- URL-access frequency (log analysis)
- Reverse web-link graph
- Term vector (word frequency)
- Inverted index (word -> doc lookup)
- Distributed Sort
- Machine Learning - SVN, NB, NN
- Heavy computation (calculate Pi)

What type of problems can use MapReduce?

- Unstructured data
- Embarrassingly Parallel - no communication/synchronization between compute units
- 2 Stage Problems:
  - Analysis - embarrassingly parallel (map)
  - Aggregation (reduce)
- Multi-stage/multi-pass solutions can exist, but will be slower (the future!)

Easy MapReduce via `mincemeatpy`

- Python framework to do MapReduce
  - Uses server/client (worker) architecture
  - Original github repository
  - My forked repository
  -

```
def mapFn(k, v):
    yield ('sum', sum(v))

def reduceFn(k, vs):
    result = sum(vs)
    return result
```

Sets a key/value pair!
You can have as many yields as you want, with numerous identical keys.

One reduce function will be called per unique key value, with vs holding all the values (in a list) for each of the key/value pairs with that key.

UNIVERSITY OF Cincinnati

`mincemeatpy` Features & Limitations

- Python 2.7 only
- Resilient to client failures
- map and reduce are not in global scope
- Doesn't require Hadoop infrastructure
- Can create as many workers(clients) as you like)
- No distributed FS! Slow IO - all over net

UNIVERSITY OF Cincinnati

Hadoop!

- Created in 2005 at Yahoo - inspired by 2004 Google MapReduce paper
- Now an Apache project
- Named after developer's elephant toy.
- Modules
  - Common - utilities
  - HDFS - Distributed storage
  - YARN - Yet another resource negotiator - Job scheduling
  - MapReduce - Parallel processing
- MANY other projects use the Hadoop base: HBase, Hive, Mahout, Pig, Spark, ZooKeeper, Sqoop, Oozie

UNIVERSITY OF Cincinnati

Hadoop (details)

- Open Source -> Many developers!
- Versions….

Feature	1.x	0.22	2.x
Secure authentication	Yes	No	Yes
Old configuration names	Yes	Deprecated	Deprecated
New configuration names	No	Yes	Yes
Old MapReduce API	Yes	Yes	Yes
New MapReduce API	Yes (with some missing libraries)	Yes	Yes
MapReduce 1 runtime (Classic)	Yes	Yes	No
MapReduce 2 runtime (YARN)	No	No	Yes
HDFS federation	No	No	Yes
HDFS high-availability	No	No	Yes

UNIVERSITY OF Cincinnati

From Hadoop - The Definitive Guide - Tom White

Running Hadoop

- 3 Running Modes - p619
  - Standalone (local) Run using local filesystem and local resources
    - No HDFS
  - Pseudodistributed
    - Simulate cluster
    - Local resources, but job scheduler
    - Use local HDFS with no replication
  - Fully Distributed - Default in our images!
    - Submit job to jobtracker server
    - Use remote HDFS with full (3) replication
    - Uses HDFS for all input/output

Can change mode via config files!

UNIVERSITY OF Cincinnati

Installing Hadoop

- See instructions on: http://cscloud.ceas.uc.edu
- Already fully installed in image:
  - Hadoop 2.6
  - Spark 1.6.1
  - R

Using Hadoop (no extra bits)

- With Java program
  - Extreme flexibility
- With *Streaming* via Unix streams
  - Uses Java code to run whatever you tell it
  - We can use any language!
  - Take from `stdin` and output to `stdout`
  - Each line (`\n` delimited) is key/value pair
  - Key/value delimited by tab (`\t`)
  - Input to reduce will be sorted by key: may see more than 1 key in reduce instance!

Streaming example

- Test via command line:
  - `cat <input> | map.py | sort | reduce.py`

Running on a Cluster

- Make sure to send files via `-file`
- Run on cluster!
  - `hadoop jar $HADOOP_INSTALL/<streaming jar location> -input <somewhere on hdfs> -output output -files *.py -mapper map.py -reducer reduce.py`
- Details

UNIVERSITY OF Cincinnati

Hadoop Extras

- Sqoop - Move data from RDBMS to Hadoop and back
- Pig - System for processing pipelines -> map-reduce jobs in Hadoop
- Hive - SQL-like manipulation of data in Hadoop -> map-reduce jobs in Hadoop
- HBase - NoSQL Column Store on top of HDFS, like Google BigTable

Apache Hive

- Developed at Facebook and Netflix
- Compile commands to map/reduce jobs
- Allows SQL users to manipulate large datasets using `HiveQL`
- Builds tables as files/folders in HDFS
- Requires `metastore` to store schemas and DB state (RDBMS system)
- HiveQL:
  - No insert, update or delete, but has **join!**
  - NOT ACID compliant!
  - Little indexing….. slow, but scalable queries!

UNIVERSITY OF Cincinnati

ref1 - ref2 - ref3 - See Ch 12 in *Hadoop - The Definitive Guide*

Hive Example (Tweets)

```
CREATE TABLE my_table (data STRING)
LOAD DATA INPATH '2015-02-01.txt' OVERWRITE INTO TABLE my_table;
# You must have write access to the file as it will get moved.

select * from my_table limit 10;

select count(*) from my_table;

# Now extract out the useful bits
CREATE TABLE tweets AS
SELECT get_json_object(my_table.data, '$.created at') AS created_at,   # split is a fn!
        get_json_object(my_table.data, '$.text') AS text,
        get_json_object(my_table.data, '$.user.id_str') AS userid
FROM my_table;

select * from tweets where userid == 486794524;
```

UNIVERSITY OF Cincinnati

Hive Example (Transactions)

```
# hadoop distcp <from> <to>   # Does distributed copy within HDFS
CREATE TABLE transraw (data STRING)
LOAD DATA INPATH `<path to folder>' OVERWRITE INTO TABLE transraw;
# You must have write access to the file as it will get moved.

select * from transraw limit 10;
select count(*) from transraw;

# Now extract out the useful bits
CREATE TABLE trans AS
SELECT split(my_table,',')[1] AS shop_date,   # split is a fn!
        float(split(my_table,',')[5]) AS spend,
        split(my_table,',')[6] AS prod_code,
        split(my_table,',')[11] AS customer,
FROM transraw;
```

UNIVERSITY OF Cincinnati

Another Hive Example (src)

```
DROP TABLE IF EXISTS records;

CREATE TABLE records (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;

SELECT year, MAX(temperature)
FROM records
WHERE temperature != 9999
  AND (quality = 0 OR quality = 1 OR quality = 4 OR quality = 5 OR quality = 9)
GROUP BY year;
```

```
1949  111
1950  22
```

UNIVERSITY OF Cincinnati

Hive Notes

- *Schema on read* not *on write/load*
  - No indexes (>0.7.0 has them somewhat)
  - But..
    - Loads data much faster
    - Can load/move data whenever
- Dropping an external table doesn't delete data!
- Can specify partition column (to files) and buckets
- Files can be binary: Sequence, Avro, RCFiles - faster access, stay compressed
- Faster on newer versions of Hadoop

Apache Pig

- Developed at Yahoo in 2006
- Higher abstraction than map/reduce
- Use `PigLatin` to manipulate and analyze data
- Generates map/reduce jobs in Hadoop
- ***Procedural*** programming
- Can store data whenever in pipeline
- Can run `grunt` (interactive) or a script
- `run` will run a pig script in `grunt` mode

UNIVERSITY OF Cincinnati

See https://developer.yahoo.com/blogs/hadoop/comparing-pig-latin-sql-constructing-data-processing-pipelines-444.html for comparison between Pig and SQL

Pig example for Twitter

```
tweet_raw = load '/data/twitter/2015-02-22.txt' as (raw: chararray);

-- JSON loading does not work, we use REGEX to extract information (live we did in Hive)
tweets =foreach tweet_raw generate REGEX_EXTRACT(raw, 'created_at":"([^"]+)"',1) as created_at,
REGEX_EXTRACT(raw, 'created_at":"([^ ]+) ',1) as wday,REGEX_EXTRACT(raw, ' (\\d\\d):\\d\\d:\\d\\d',1) as
hday, REGEX_EXTRACT(raw, '"text":"([^"]+)"',1) as text, REGEX_EXTRACT(raw, '"id_str":"(\\d+)","name"',1)
as id;

-- remove everything except the hour of the day
tweet_count = foreach tweets generate hday;
-- group by hour of day, and specify 24 reducers to do it
hour_count = group tweet_count by hday parallel 24;

-- Now go over all group and create a new tuple containing hour and tweet count
hour_count2 = foreach hour_count generate $0 as hour,COUNT(tweet_count) as count;

-- print to screen
dump hour_count2;
-- store to HDFS
STORE hour_count2 INTO 'hour_count';
```

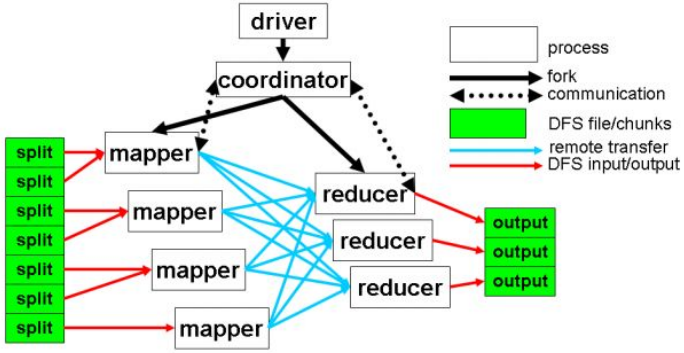
UNIVERSITY OF Cincinnati

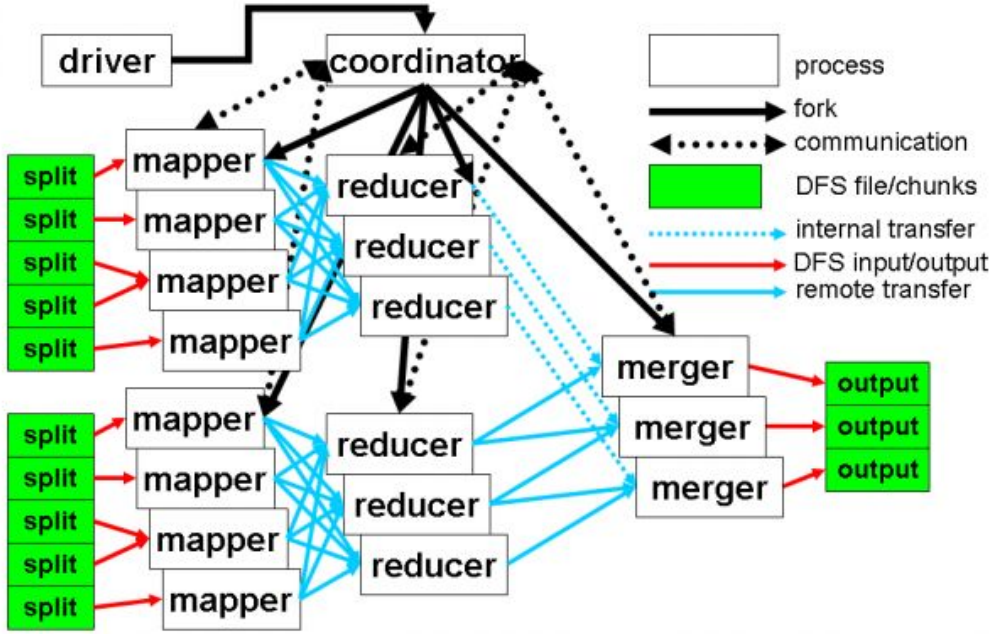
Extension to Map/Reduce

- Map-Reduce-Merge - Add merge step (ref)

$$\begin{aligned} \text{map:} &\ (k_1, v_1) \rightarrow [(k_2, v_2)] \\ \text{reduce:} &\ (k_2, [v_2]) \rightarrow [v_3] \end{aligned}$$

$$\begin{aligned} \text{map:} &\ (k_1, v_1)_\alpha \rightarrow [(k_2, v_2)]_\alpha \\ \text{reduce:} &\ (k_2, [v_2])_\alpha \rightarrow (k_2, [v_3])_\alpha \\ \text{merge:} &\ ((k_2, [v_3])_\alpha, (k_3, [v_4])_\beta) \rightarrow [(k_4, v_5)]_\gamma \end{aligned}$$

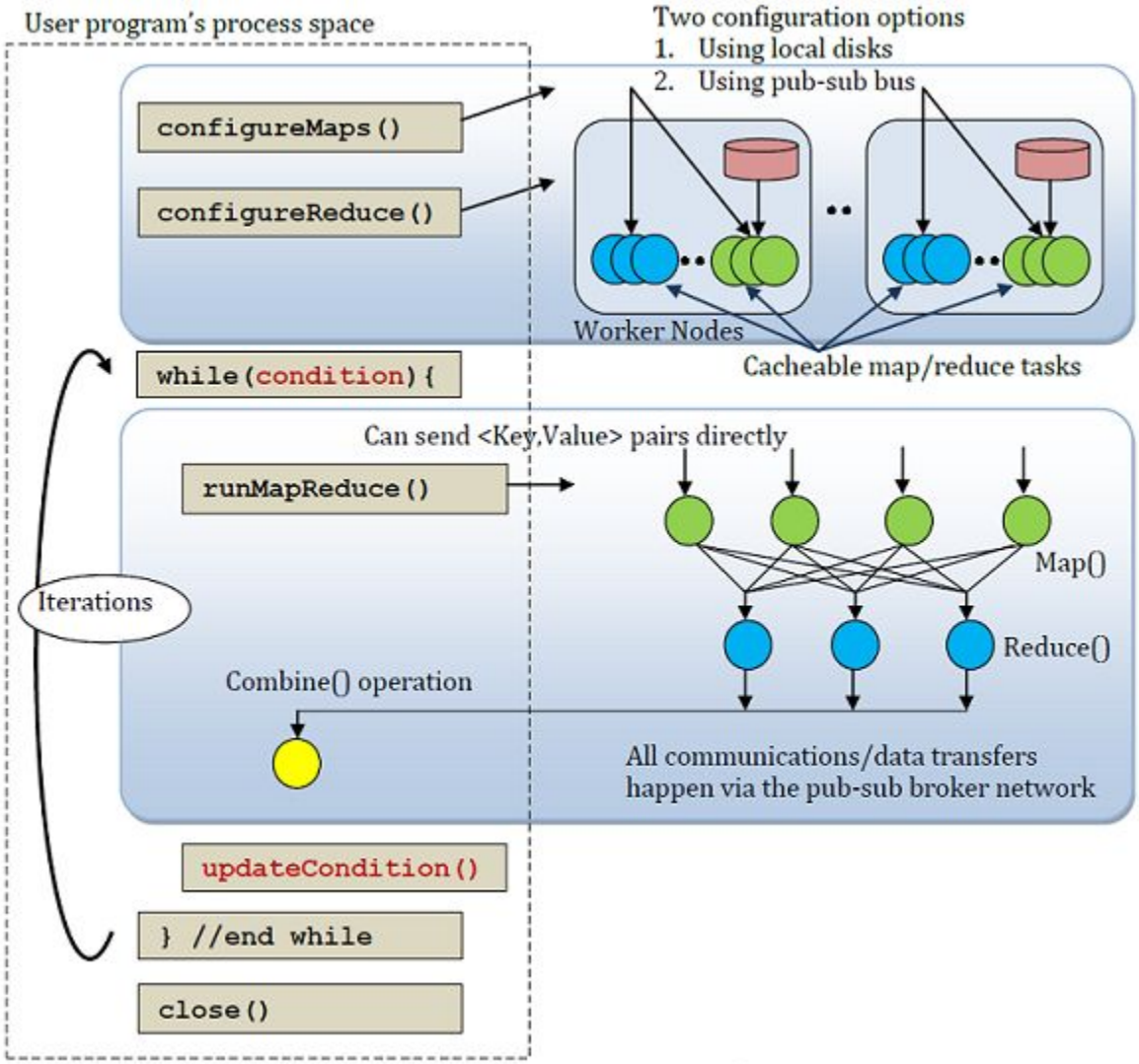




Cincinnati

Extension to Map/Reduce

- Twister (ref) - Loop map/reduce jobs



Alternatives to Map/Reduce

- Sphere - (ref) Uses sector DFS - WAN optimized - File based, not block - User Defined Functions - Better support legacy apps.
- All-Pairs - (ref) Efficiently do:

```
All-pairs(A:set, B:set, F:function) -> M:matrix
```

aka

```
for each $i in A:
    for each $j in B:
        Submit job F($i, $j)
```

Optimally move/assign data/jobs

UNIVERSITY OF Cincinnati

Alternatives to Map/Reduce

- DryadLINQ - (ref) - Microsoft Research - User defines a DAG: programs and channels. Superset of Map/Reduce.
- FlumeJava - (ref) - Google (2010) - Java library - Define dataflow using high-level primitives and mutators - deferred execution (optimization possible)

Alternatives to Map/Reduce