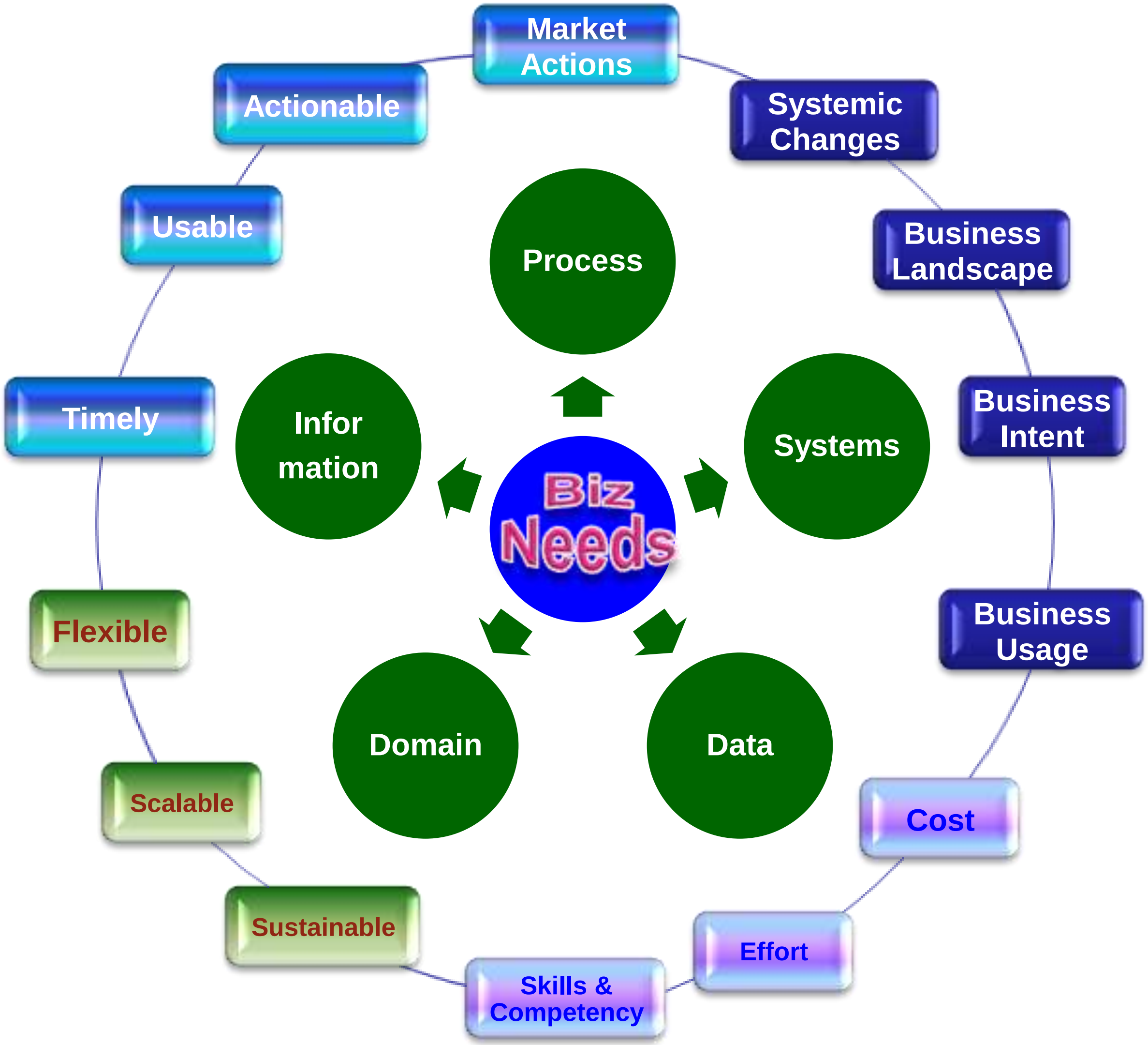
Process, Data, Domain driven Business Excellence

Hadoop and No SQL

Nagaraj Kulkarni

Nagaraj.Kulkarni@compegence.com

Process, Data and Domain Integrated Approach



Decision Excellence

Competitive Advantage lies in the exploitation of:

–More detailed and specific information

–More comprehensive external data & dependencies

–Fuller integration

–More in depth analysis

–More insightful plans and strategies

–More rapid response to business events

–More precise and apt response to customer events

Ram Charan's Book: What The CEO Wants You To Know: How Your Company Really Works

Touching Upon

Context For Big Data Challenges

Data Base Systems – Pre Hadoop Strengths and Limitations

What is Scale, Why No SQL

Think Hadoop, Hadoop Eco system

Think Map Reduce

Nail Down Map Reduce

Think GRID (Distributed Architecture)

Deployment Options

Map Reduce Not and Map Reduce Usages

Nail Down HDFS and GRID Architecture

Objective: Understand MapReduce Paradigm

Systemic Changes

Connected Globe

Boundary less ness

Best Sourcing

Interlinked Culture

Customer Centric

Demand Side Focus

Bottom Up Innovation

Empowered employees

Agility and

Response Time

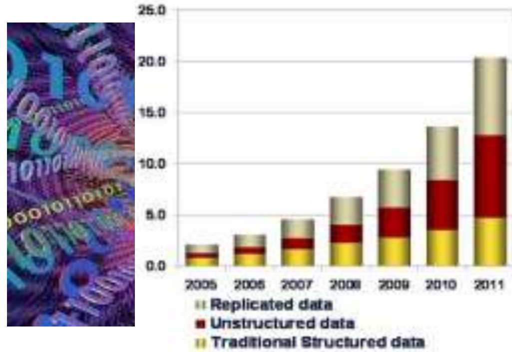
Leading Trends

Responsiveness

Speed, Agility, Flexibility

Landscape To Address

Data Explosion



- Manageability
- Scalability
- Performance

Information Overload

- Agility
- Decision Making
- Time to Action

Interlinked Processes & Systems

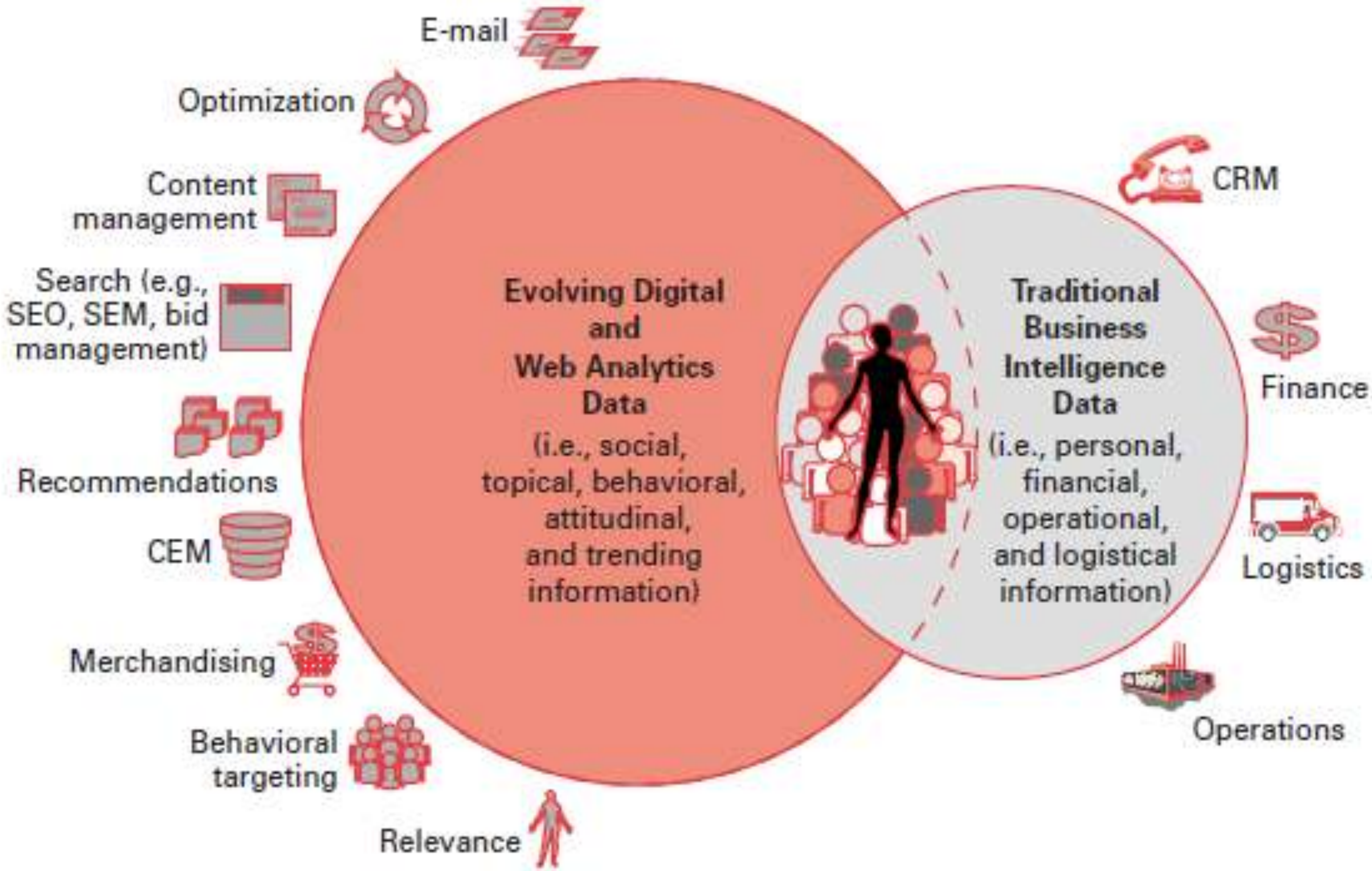
- Boundaryless
- Systemic Understanding
- Collaborate and Synergize
- Simplify and Scale

Information Overload

A wealth of information creates a poverty of attention.

Herbert Simon,
Nobel Laureate Economist

More Touch points, More Channels



Source: JupiterResearch (7/08)

How do we scale

Traditional System - How they achieve Scalability

- Multi Threading
- Multiple CPU – Parallel Processing
- Distributed Programming – SMP & MPP
- ETL Load Distribution – Assigning jobs to different nodes
- Improved Throughput

COMPEGENCE
Process, Data, Domain driven Business Excellence

Scale – What is it about?

Facebook
500 Million Active Users per Month

500 Billion+ Page Views per month

25 Billion+ Content per month

15 TB New Data / day
1200 m/cs, 21 PB Cluster

eBay
90 Million Active Users

10 Billion Requests per day

220 million+ items on sale

40 TB + / day
40 PB of Data

Yahoo
82 PB of Data
25000+ nodes

Twitter
1 TB plus / day
80 + nodes

1.73 Billion Internet Users

247 Billion emails per day

126 Million Blogs

5 Billion Facebook Content per week

50 Million Tweets per day

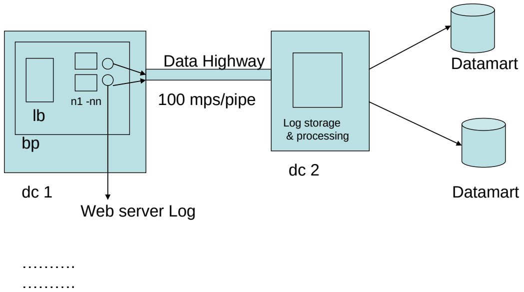
80% of this data is **unstructured**

Estimated 800 GB of data per user (million Petabyte!)

How do we scale – Think Numbers

Thinking of Scale - Need for Grid

Think Numbers



1000 Nodes / DC
10 DC
1K byte webserver log record
1 second / row

In one day

1000 * 10 * 1K * 60 * 60 * 24 = 864 GB

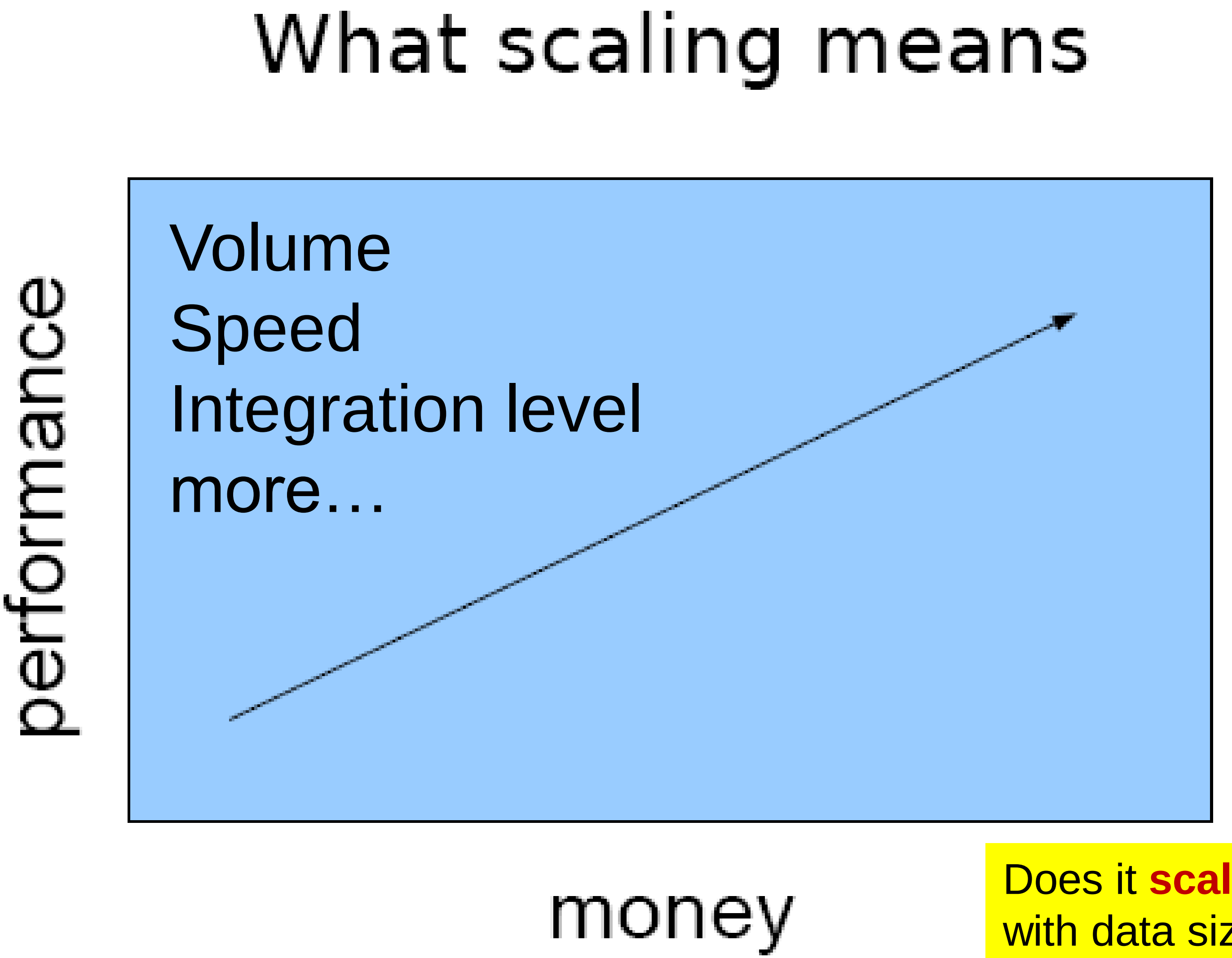
Storage for a year

864 GB * 365 = 315 TB

To store 1 PB – 40K * 1000 = Millions $
To process 1 TB = 1000 minutes ~ 17 hrs

Think Agility and Flexibility

Scale – What is it about?



Does it **scale linearly** with data size and analysis complexity

We would not have no issues…

If the following assumptions Hold Good:

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

New Paradigm: Go Back to Basics

- ✓Divide and Conquer (Divide and Delegate and Get Done)
- ✓Move Work or Workers ?
- ✓Relax Constraints (Pre defined data models)
- ✓Expect and Plan for Failures (avoid n address failures)
- ✓Community backup
- ✓Assembly Line Processing
  - ✓(Scale, Speed, Efficiency, Commodity Worker)
- ✓The “For loop”
- ✓Parallelization (trivially parallelizable)
- ✓Infrastructure and Supervision (Grid Architecture)
- ✓Manage Dependencies
- ✓Ignore the Trivia (Trivia is relative!)

Charlie Munger’s Mental Models

Joel Spolsky
http://www.joelonsoftware.com/items/2006/08/01.html

New Paradigm: Go Back to Basics

Map Reduce Paradigm	**Grid Architecture**
✓Divide and Conquer ✓The “for loop” ✓Sort and Shuffle ✓Parallelization (trivially parallelizable) ✓Relax Data Constraints ✓Assembly Line Processing Scale, Speed, Efficiency, Commodity Worker)	✓Split and Delegate ✓Move Work or Workers ✓Expect and Plan for Failures ✓Assembly Line Processing (Scale, Speed, Efficiency, Commodity Worker) ✓Manage Dependencies and Failures ✓Ignore the Trivia (Trivia is relative!)

Map Reduce History

- Lisp
- Unix
- Google FS

Replication, Redundancy,
Heart Beat Check, Cluster rebalancing,
Fault Tolerance, Task Restart,
Chaining of jobs (Dependencies),
Graceful Restart,
Look Ahead or Speculative execution,

No SQL Options

Hbase/Cassandra for huge data volumes- PBs.

•Hbase fits in well where Hadoop is already being used.

•Cassandra less cumbersome to install/manage

MongoDB/CouchDB

Document oriented databases for easy use and GB-TB volumes. Might be problematic at PB scales

Neo4j like graph databases

for managing relationship oriented applications- nodes and edges

Riak, redis, membase like Simple key-value databases

for huge distributed in-memory hash maps

Let us Think Hadoop

Hadoop Ecosystem Map

COMPEGENCE
Process, Data, Domain driven Business Excellence

RDBMS and Hadoop

	RDBMS	**MapReduce**
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Structure	Fixed schema	Unstructured schema
Language	SQL	Procedural (Java, C++, Ruby, etc)
Integrity	High	Low
Scaling	Nonlinear	Linear
Updates	Read and write	Write once, read many times
Latency	Low	High

Apache Hadoop Ecosystem

Hadoop Common: The common utilities that support the other Hadoop subprojects.

HDFS: A distributed file system that provides high throughput access to application data.

MapReduce: A software framework for distributed processing of large data sets on compute clusters.

Pig: A high-level data-flow language and execution framework for parallel computation.

HBase / Flume / Scribe: A scalable, distributed database that supports structured data storage for large tables.

Hive: A data warehouse infrastructure that provides data summarization and ad hoc querying.

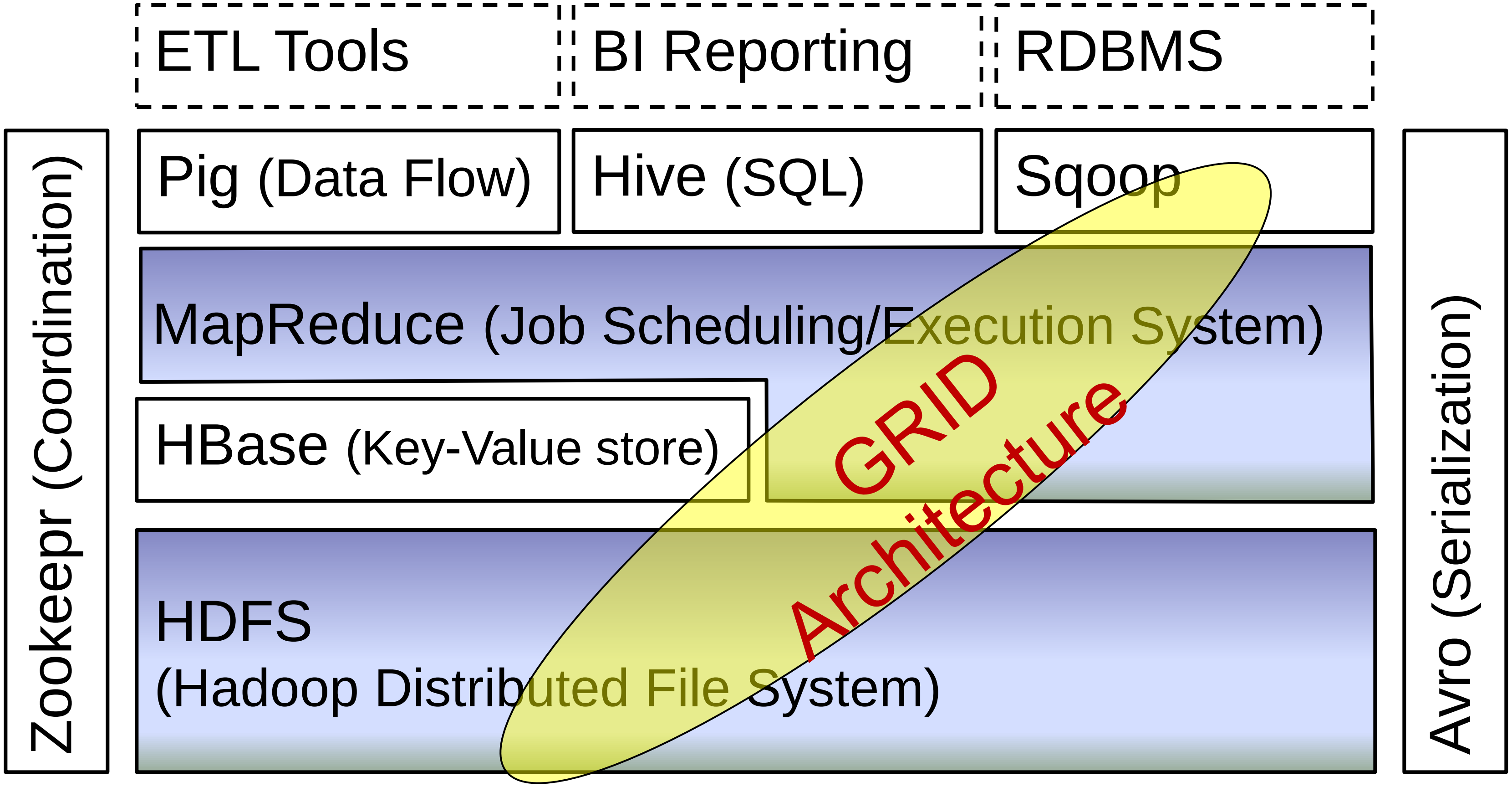
ZooKeeper: A high-performance coordination service for distributed applications.

Flume: Message Que Processing

Mahout: scalable Machine Learning algorithms using Hadoop

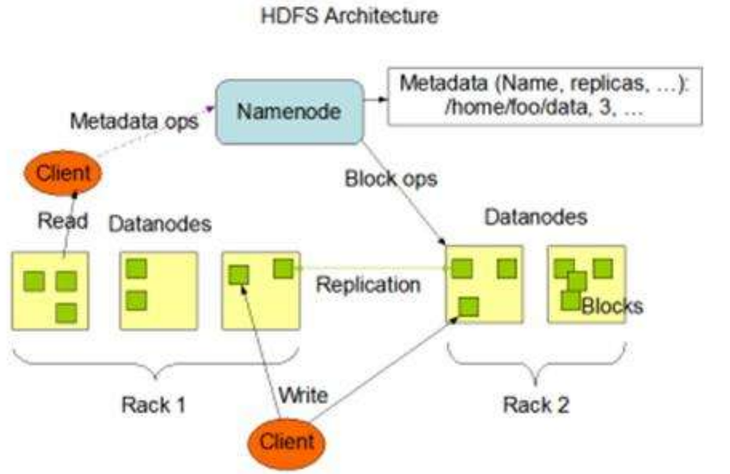
Chukwa: A data collection system for managing large distributed systems.

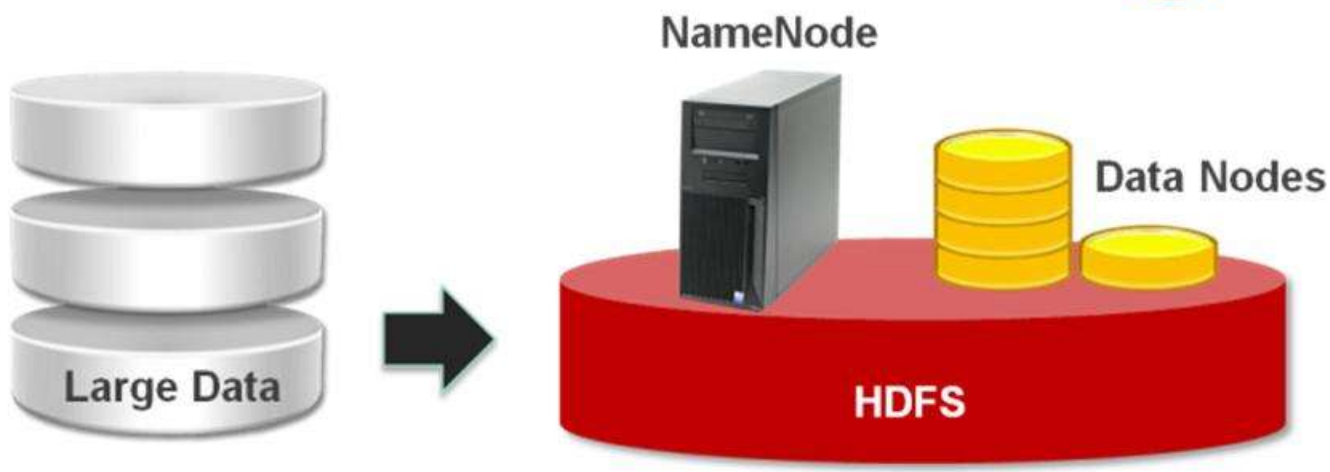
Apache Hadoop Ecosystem



HDFS – The BackBone

Hadoop Distributed File System

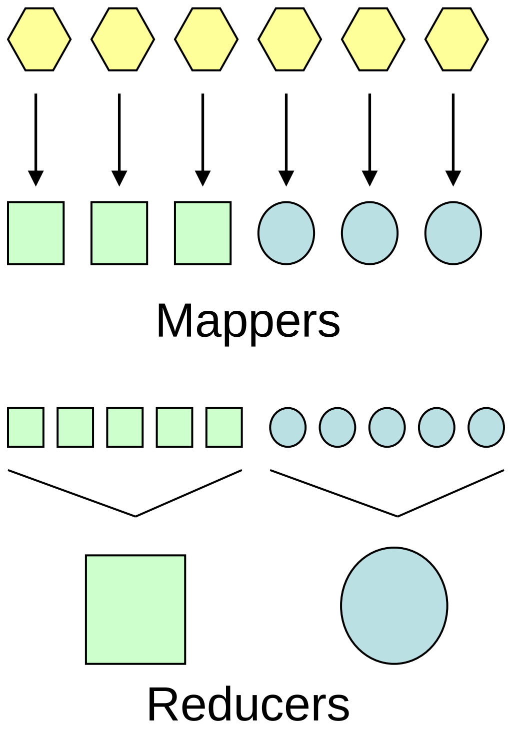




Transforming Large Data

MapReduce Basics

- Functional Programming
- List Processing
- Mapping Lists



PIG – Help the Business User Query

Pig: **Data-aggregation functions over semi-structured data** (log files).

Pig Latin Programs

Query Parser

Logical Plan

Semantic Checking

Logical Plan

Logical Optimizer

Optimized Logical Plan

Logical to Physical Translator

Physical Plan

Physical To M/R Translator

MapReduce Plan

Map Reduce Launcher

PIG Latin Example

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x > 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
    x, COUNT(B);
STORE D INTO 'output';
```

pig.jar:

- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

Execution Plan

Map:
Filter

Reduce:
Count

hadoop

Do you like the Flexibility?

HIVE – SQL Like

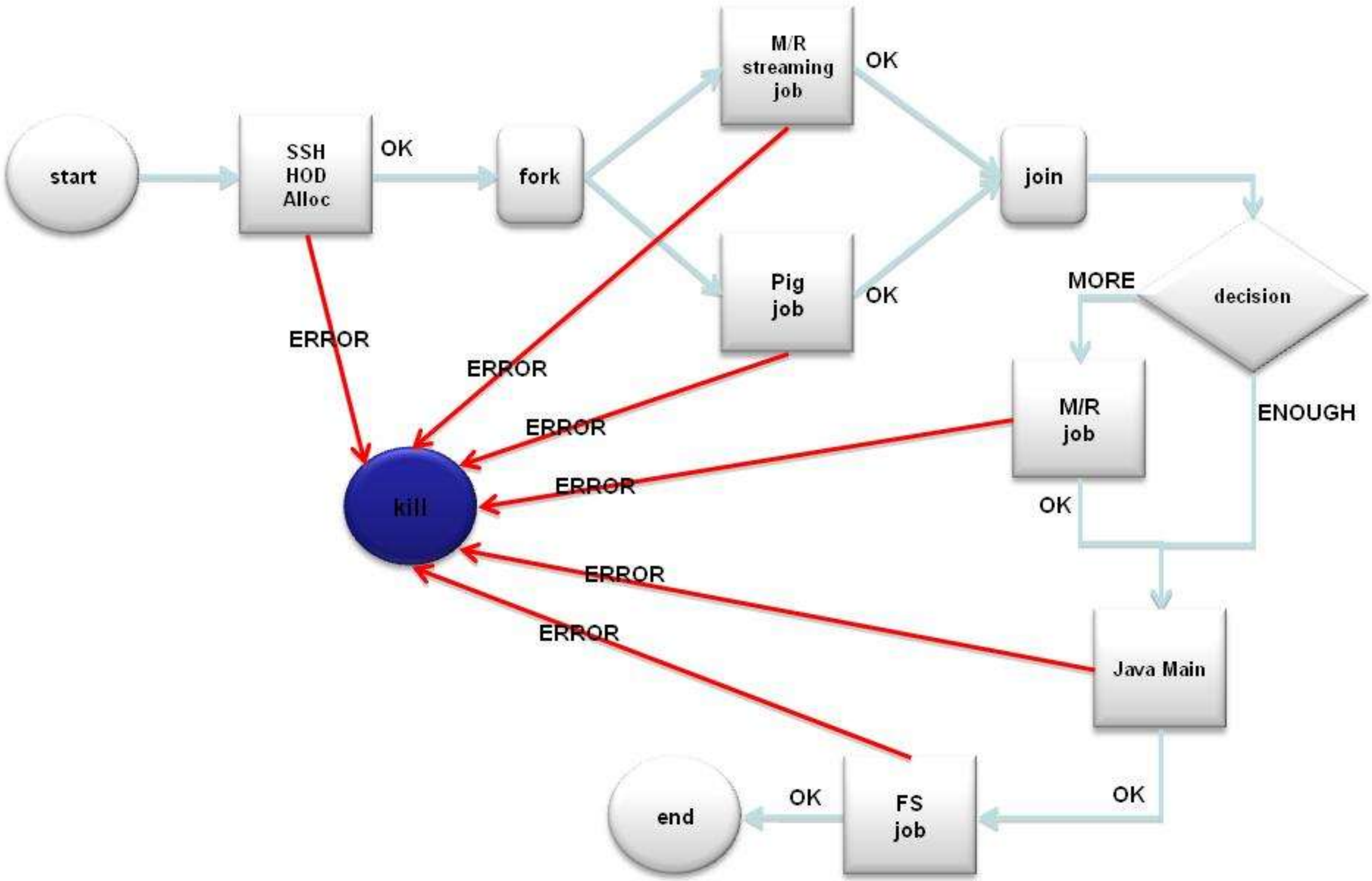
- A high level interface on Hadoop for managing and querying structured data
  - Interpreted as Map-Reduce jobs for execution
  - Uses HDFS for storage
  - Uses Metadata representation over hdfs files
- Key Building Principles:
  - Familiarity with SQL
  - Performance with help of built-in optimizers
  - Enable Extensibility – Types, Functions, Formats, Scripts

FLUME – Distributed Data Collection

- Distributed Data / Log Collection Service
- Scalable, Configurable, Extensible
- Centrally Manageable

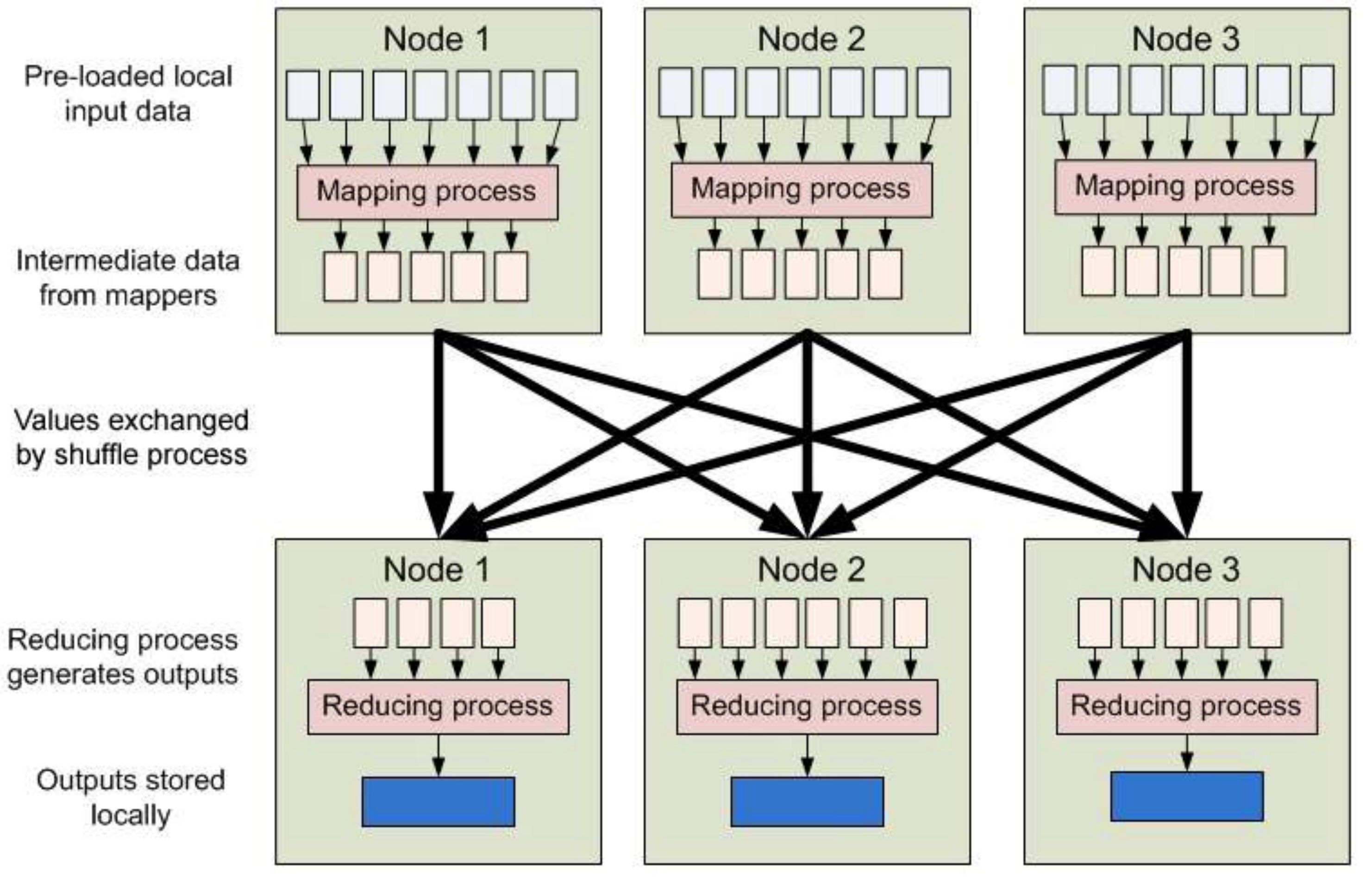
- Agents fetch data from apps, Collectors save it
- Abstrations: Source -> Decrator(s) -> Sink

An Oozie Workflow

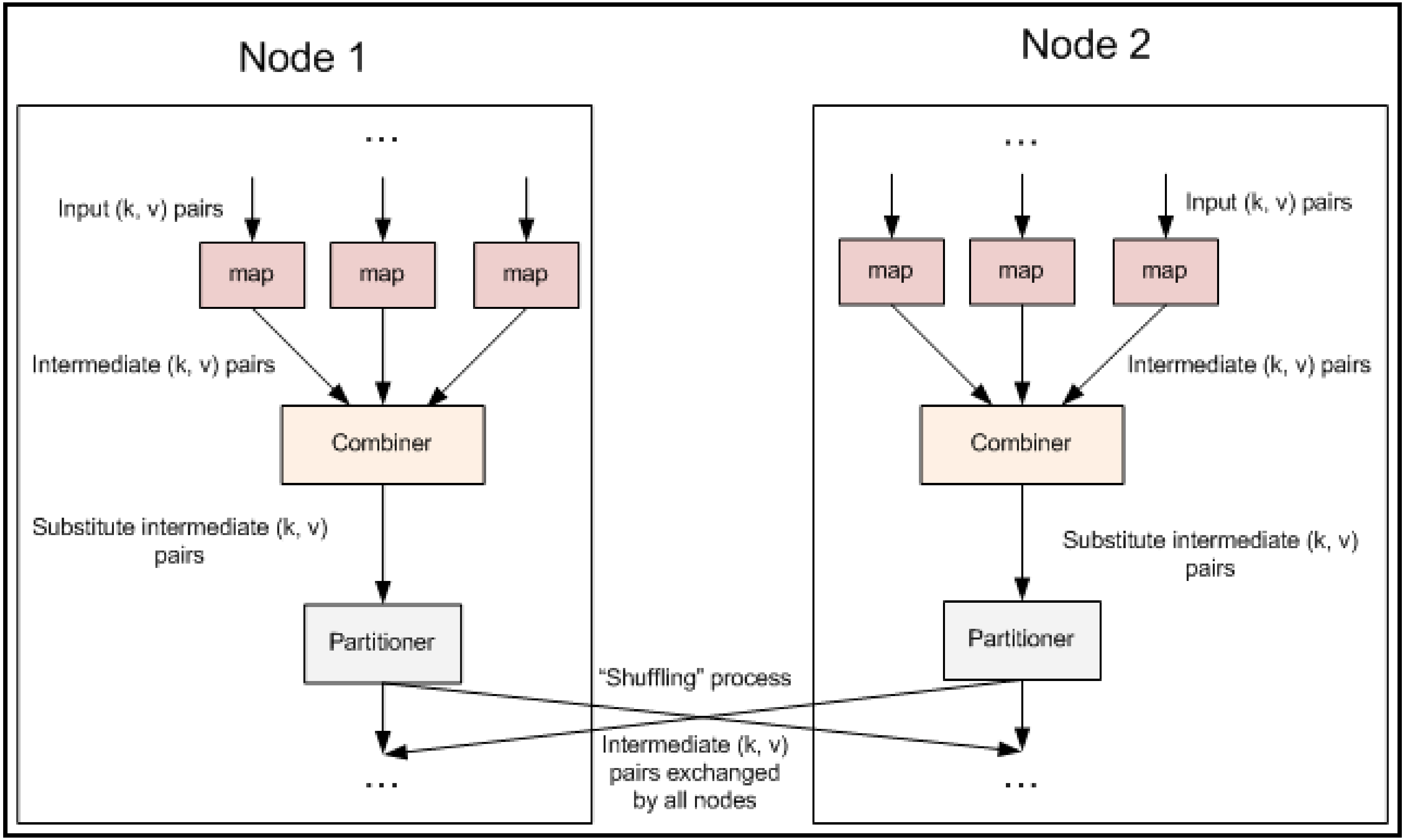


Understanding Map Reduce Paradigm

Logical Architecture



Understanding Map Reduce Paradigm



Map Reduce Paradigm

Job

Configure the Hadoop Job to run.

Mapper

map(LongWritable key, Text value, **Context context)**

Reducer

reduce(Text key, **Iterable<IntWritable> values, Context context)**

Programming model

Map –Reduce Definition

MapReduce is a

functional programming model and an

associated implementation model

for processing and generating large data sets.

Users specify a ***map*** function that processes a **key/value pair**
to generate a set of intermediate key/value pairs,

and

a ***reduce*** function that **merges** all intermediate values associated
with the same intermediate key.

Many real world tasks are expressible in this model.

CONCEPTS

Programming model

Input & Output: each a set of key/value pairs

Programmer specifies two functions:

map (in_key, in_value) -> list(out_key, intermediate_value)

- Processes input key/value pair
- Produces set of intermediate pairs

reduce (out_key, list(intermediate_value)) -> list(out_value) Combines all intermediate values for a particular key

- Produces a set of merged output values (usually just one)
- Inspired by similar primitives in LISP and other languages

Map Reduce Paradigm

Word Count Example

A simple MapReduce program can be written to determine how many times different words appear in a set of files.

What does Mapper and Reducer do?

Pseudo Code:

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)
```

```
reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```

Programming model

Example: Count word occurrences

```
map(String input_key, String input_value):
        // input_key: document name
        // input_value: document contents
        for each word w in input_value:
                EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
        // output_key: a word
        // output_values: a list of counts
        int result = 0;
        for each v in intermediate_values:
                result += ParseInt(v);
        Emit(AsString(result));
```

Pseudocode: See appendix in paper for real code

Understanding Map Reduce Paradigm

Map – Reduce Execution Recap

RECAP

- Master-Slave architecture
- Master: JobTracker
  - Accepts MR jobs submitted by users
  - Assigns Map and Reduce tasks to TaskTrackers (slaves)
  - Monitors task and TaskTracker status, re-executes tasks upon failure
- Worker: TaskTrackers
  - Run Map and Reduce tasks upon instruction from the Jobtracker
  - Manage storage and transmission of intermediate output

Understanding Map Reduce Paradigm

Map – Reduce Paradigm Recap

Example of map functions –

Individual Count, Filter, Transformation, Sort, Pig load

Example of reduce functions –

Group Count, Sum, Aggregator

A job can have many map and reducers functions.

How are we doing on the Objective

Objective:

Understand

MapReduce

Paradigm