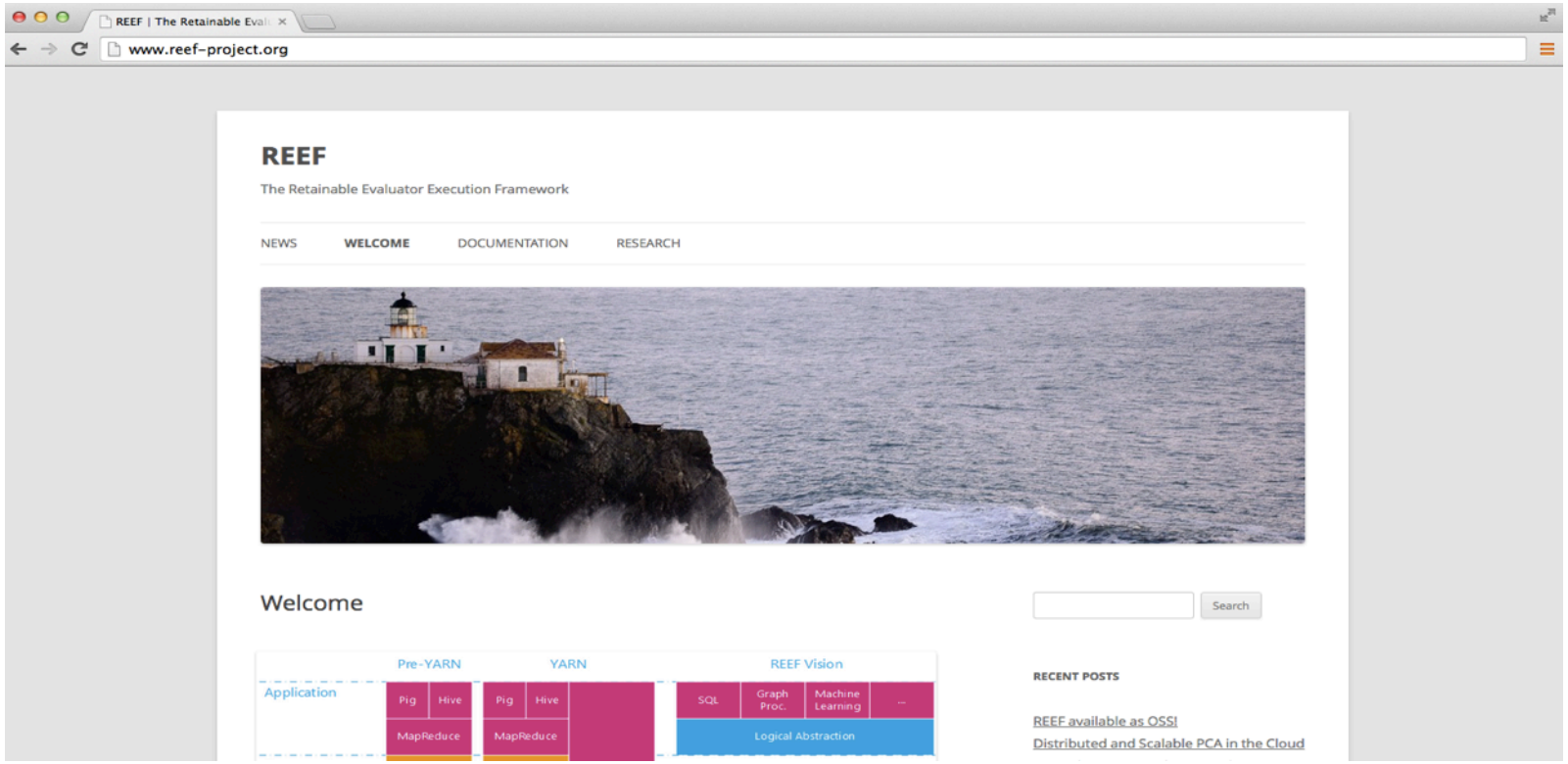
REEF Tutorial

Cloud & Mobile Systems Lab @ SNU
(http://cmslab.snu.ac.kr)
Brian Cho
Yunseong Lee
Taegeon Um

Welcome to the REEF Tutorial

Open-sourced under the Apache License 2 in Jan. 2014

http://www.reef-project.org/
https://github.com/Microsoft-CISL/REEF



Agenda

• Coding Practices	10:00 – 10:30
• Environment Setup	10:30 – 11:00
• REEF Overview	11:00 – 12:00
• Lunch	12:00 – 13:00
• Coding Tutorials & Lab	13:00 – 17:00

Coding Practices

Brian Cho

REEF's Impact Potential

chosun.com 사회

책 읽기 문화 캠페인 책, 세상을 열다

뉴스 오피니언 경제 스포츠 연예 라이프

사회

사람들

전병곤 마이크로소프트 학술상

기사 100자평(0)

입력 : 2014.06.19 03:02

전병곤 서울대 컴퓨터공학부 교수가 한국인 첫 마이크로소프트 연구소 학술상 수상자로 선정됐다. 컴퓨터 공학계를 선도하는 젊은 교수를 대상으로 한 상이다. 10만달러 상금과 함께 마이크로소프트 연구소와의 공동 연구 기회를 갖는다.

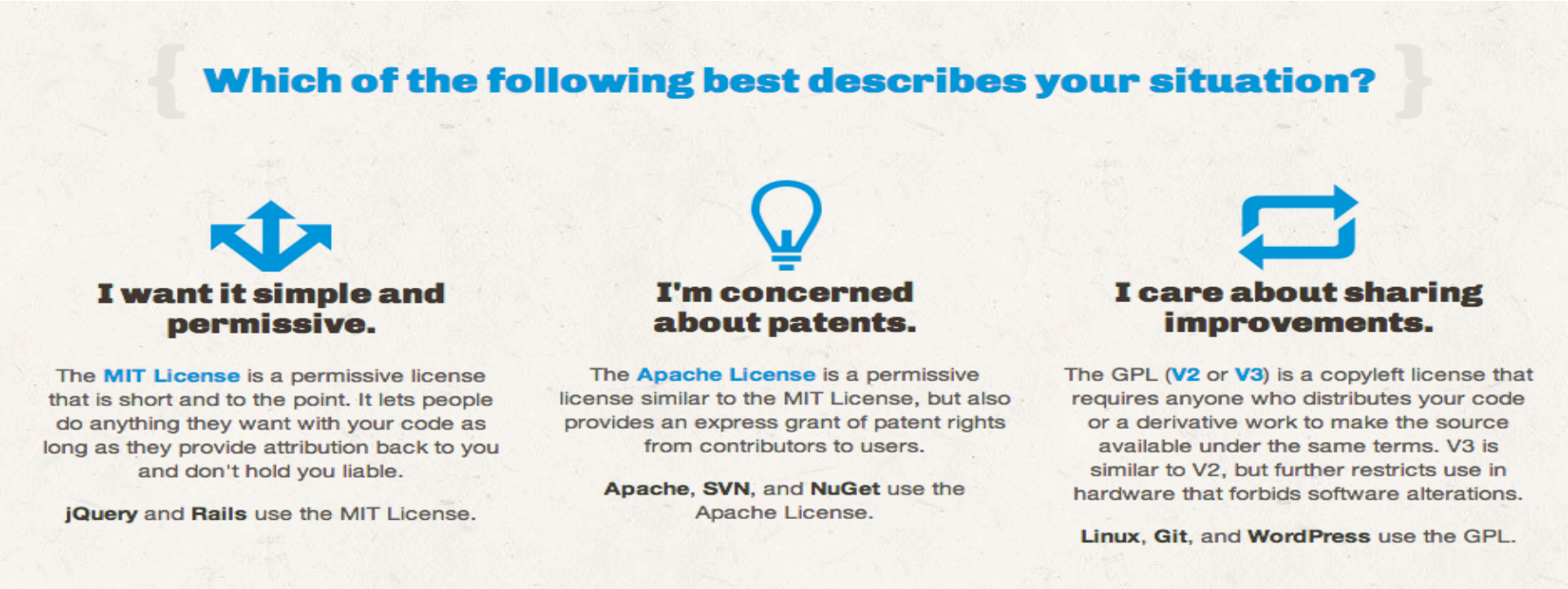
Byung-Gon Chun
Assistant Professor
Department of Computer Science and Engineering
Seoul National University

Byung-Gon Chun is interested in creating new platforms for operating and distributed systems. He is currently developing a big data platform that makes it easy to implement large-scale, fault-tolerant, heterogeneous data processing applications. He has also built systems that seamlessly integrate cloud computing with mobile devices for improved performance, reliability, and security. Chun received his PhD in Computer Science from the University of California, Berkeley. Prior to joining Seoul National University, Chun was a principal scientist at Microsoft, a research scientist at Yahoo! Research and Intel Research, and a postdoctoral researcher at ICSI.

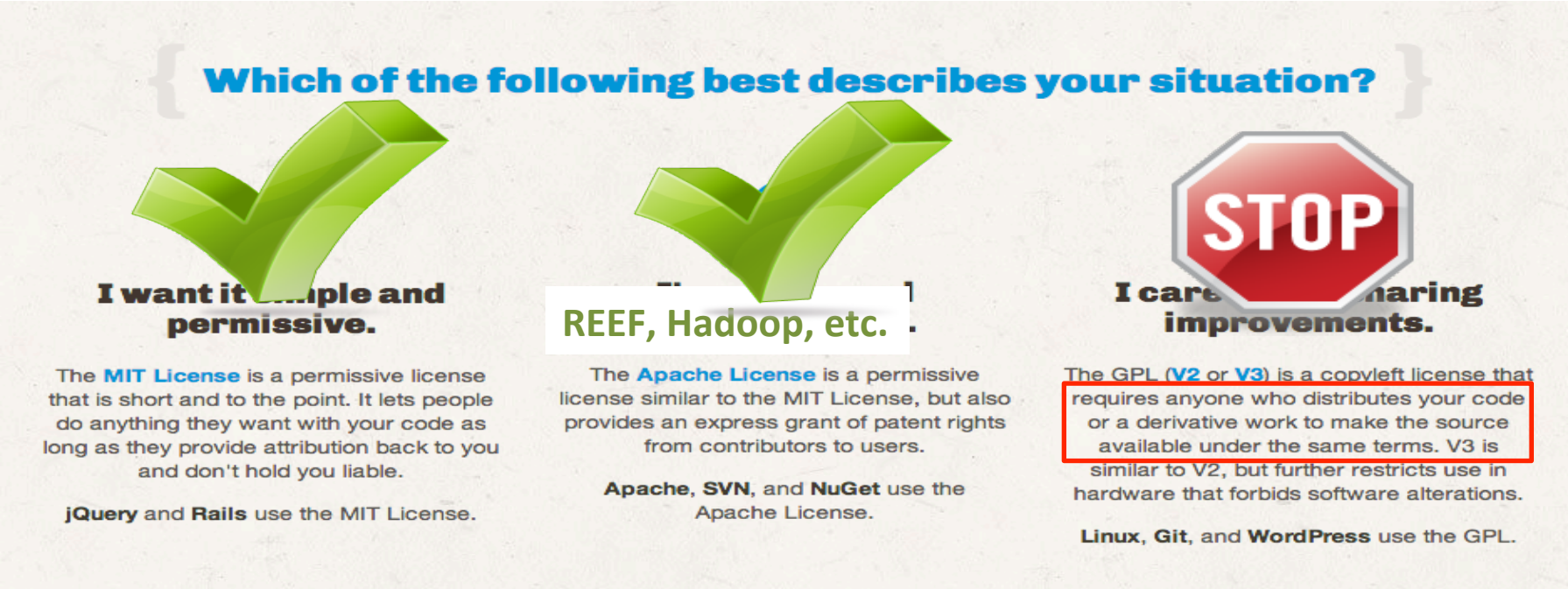
Coding Practices

- Open source
- Agile development
  - Overview
  - Test Driven Development (TDD)
  - Sprint
- GitHub and Git
  - GitHub process: commit logs, pull requests, issues
  - Git commands
- Coding style

Open source licenses



Open source licenses used by CMSLab

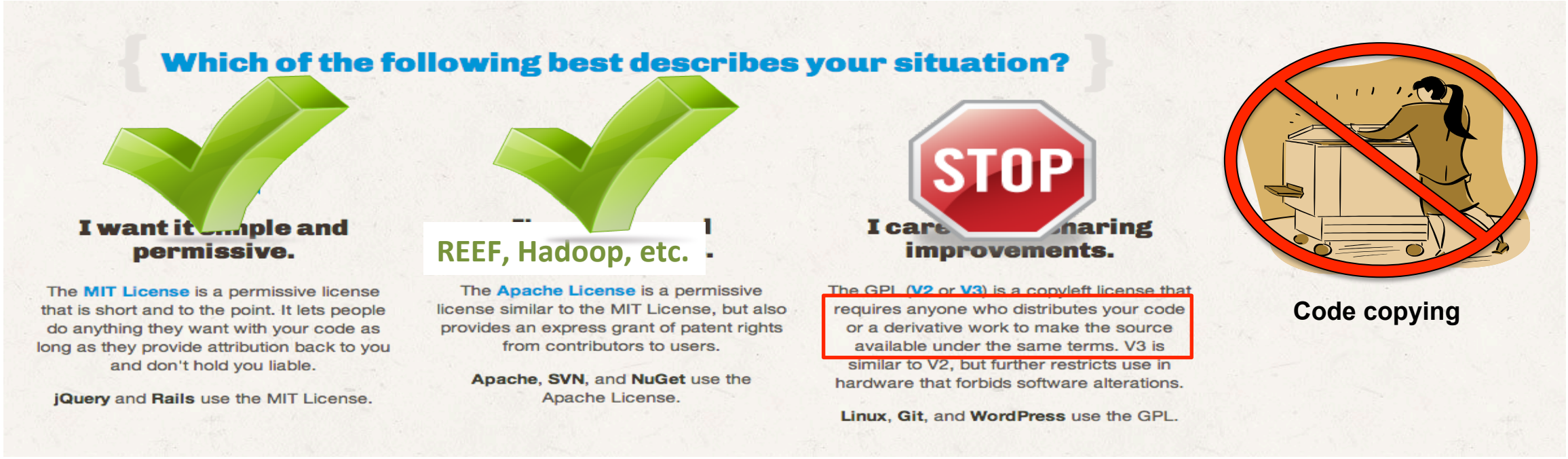


- MIT licensed libraries are OK

- REEF is Apache licensed
- Apache licensed libraries are OK

- Do not use GPL licensed libraries

Say NO to Code Copying



- MIT licensed libraries are fine

- REEF is Apache licensed
- Can use Apache licensed libraries

- Do not use GPL licensed libraries

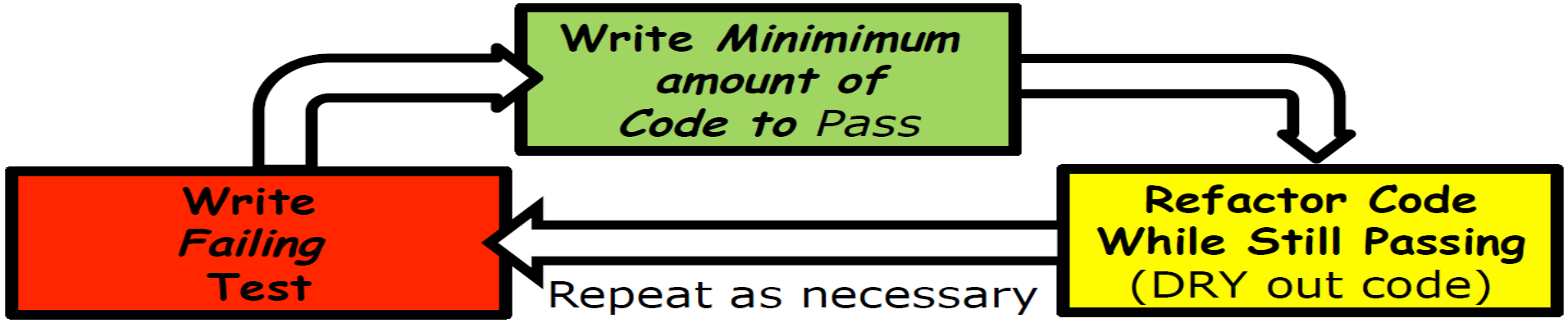
- Never copy open source code
- Import as a library

Agile Development

- Iterative
- Incremental
- Collaborative

- ➢ Continuous
- ➢ Quick
- ➢ Momentum

Test-Driven Development (TDD)



- Test-driven development (TDD) is a software development process that relies o n the repetition of a very short development cycle:
  - First the developer writes an (initially failing) automated test case that define s a desired improvement or new function,
  - Then produces the minimum amount of code to pass that test, and
  - Finally refactors the new code to acceptable standards.
- Key thing – *Tests* come before *Code*

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

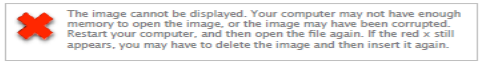
Test-First development

- Think about one thing the code *should* do
- Capture that thought in a test, which fails
- Write the simplest possible code that lets the test pass
- Refactor: DRY out commonality w/other tests
- Continue with next thing code should do

Red – Green – Refactor

Aim for "always have working code"

"working" to "working"

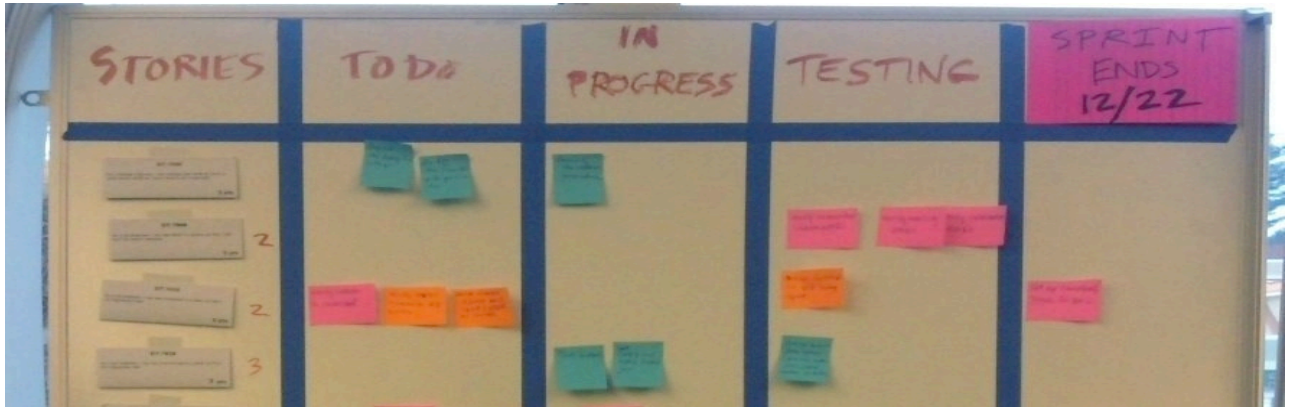


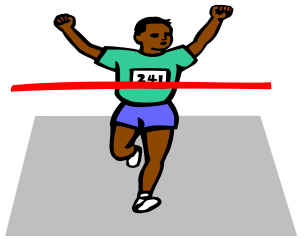
JUnit

- Unit tests run after every Maven build
  - New features must be thoroughly tested
  - Regression testing: make changes with confidence
- Use mocks (Mockito)
  - Objects that are not under test should be mocked
  - If mocking is difficult, you may be violating the Single Responsibility Principle (SRP) in your classes

Sprint

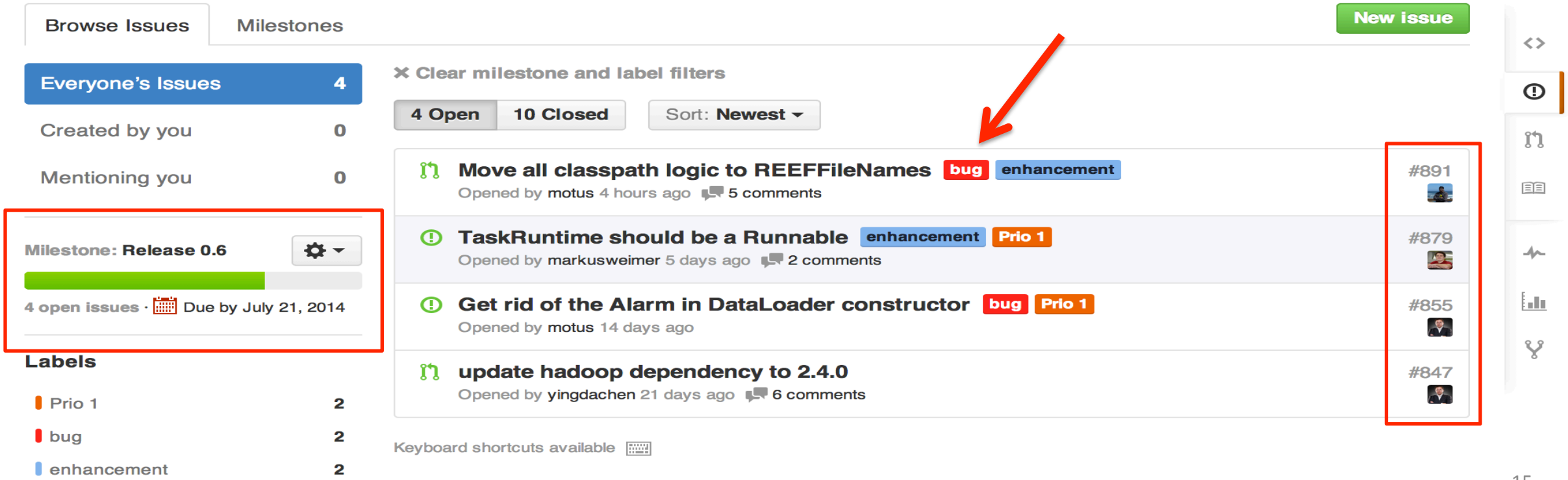
- Basic unit of development
- One week – one month
- Development tasks
  - Defined at planning meeting
  - Each developer signs up for tasks on task board





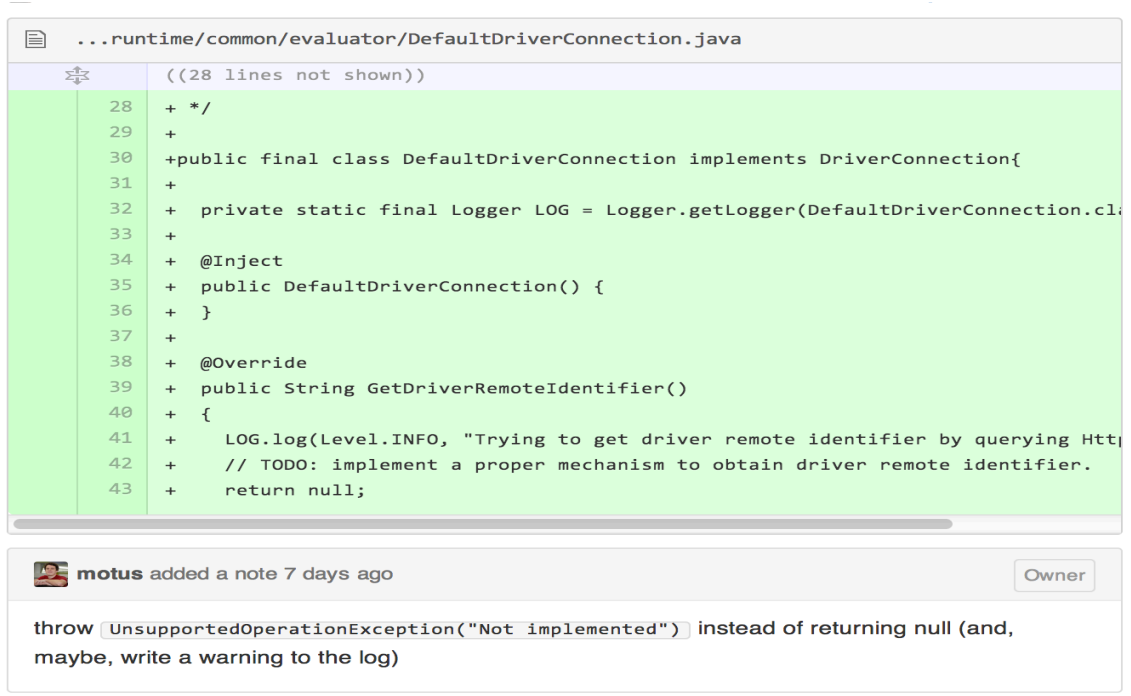
GitHub: Milestones and Issues

- Sprint → Milestone
- Task Board → Issues
- Developer sign-up → Assignment



GitHub: Pull Requests (1/2)

- Code review
  - Improve quality of code
  - Improve developer skills



GitHub: Pull Requests (2/2)

- Code is to be submitted only via pull requests!
- Process
  - Create branch
  - Make local commits
    - Commit often
  - Push branch to repo
    - REEF: work on forked repo
  - Create new pull request
    - Double check code changes before submitting
  - Pull request is merged by reviewer
- Commit / Pull Request messages
  - First line of message: concise description
  - Long and meaningful message
  - Pull Request messages should reference Issues: "… closes #611" (the reviewer will close the Issue)

Remove usage of Java Serialization #612

Merged **motus** merged 3 commits into `master` from `weimer_611` 7 months ago

Conversation 4 · Commits 3 · Files changed 7

markusweimer commented on Nov 21, 2013 — Owner

This closes #611:

1. Added a new protocol message that will be used as an envelope for all REEF control flow messages
2. Added a codec for it.
3. Bound that codec implementation in `Launcher` and `LaunchClass` for processes started by REEF (Driver, Evaluator) and in `LocalRuntimeConfiguration` and `YarnClientConfiguration` for the client.

All tests pass on my box, with the following exceptions from REEF-IO:

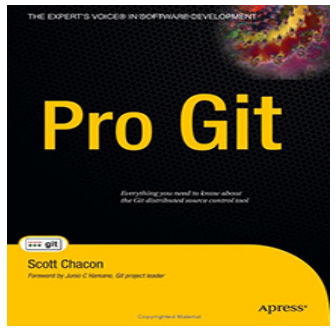
```
Tests in error:
  ReceiverTest.testReceiveListOfList:218->sendMsg:143 ¶  test timed out after 10...
  ReceiverTest.testReceiveList:179->sendMsg:143 ¶  test timed out after 1000 mil...
  ReceiverTest.testReceiveIdentifierIdentifierType:131->sendMsg:143 ¶  test time...
  ReceiverTest.testReceiveListOfQextendsIdentifierIdentifierType:256->sendMsg:143 ¶
  SenderTest.testSendIdentifierIdentifierTType:131 ¶  test timed out after 1000 ...
  SenderTest.testSendListOfList:210 ¶ Network com.microsoft.reef.io.network.nami...
  SenderTest.testSendIdentifierListOfQextendsIdentifierListOfTListOfIntegerType:258 ¶
  SenderTest.testSendIdentifierIdentifierListOfTType:170 ¶  test timed out after...
  NameClientTest.testClose:66 ¶  test timed out after 1000 milliseconds
  NameClientTest.testLookup:88 ¶  test timed out after 1000 milliseconds
```

Those failures also occur on current master on my box, and cannot be reproduced on the build servers.

Git Commands

- Create branch
  - `git checkout -b branch_name`
- Make local commits
  - See changes:
    - `git status / git diff`
  - Stage changes:
    - `git add file_name`
  - Make commit:
    - `git commit`
- Push branch to repo
  - `git push origin branch_name`

- References
  - Pro Git (free book): http://git-scm.com/book
  - Learn Git Branching: http://pcottle.github.io/learnGitBranching



Java Coding Style

- We use 2 spaces for indents, no tabs
- Lines end at 120 characters
- Everything that can conceivably carry a 'final' keyword should.
- To the extend possible, all our API objects (and much of the implementation) can be instantiated via Tang.
- The reef-annotations submodule contains a few annotations we found useful.
- For thread-safety, please use the "Java Concurrency in Practice" annotations.

Agile Development Recap

- Iterative + Incremental
  - 1 week – 1 month Sprints defined as GitHub Milestones
  - Small developer tasks defined as GitHub Issues
  - TDD+JUnit guides changes and prevents regressions
- Collaborative
  - Discussion on GitHub Issues
  - Code reviews using GitHub Pull Requests
  - Stand-up meeting
  - Face-to-face discussions
  - Improve code and developer skills

Setup the Environment

Yunseong Lee

Setup the environment

- You can refer to http://yunseong.github.io/reef-tutorial/commands.html
- On linux, there exists an excellent shell script to install Hadoop and REEF made by Taegeon
- But the script was designed to deploy Hadoop and REEF into a blank EC2 instance. It may cause a problem if some programs already have been installed.
- It is recommended to build the following directions

Setup the environment

- YARN (in Hadoop v2)
  - We will see what it is in the next section
  - Setup order
    - Install Java 7 (If you do not have)
    - Configure SSH
    - Download hadoop & Install
    - Execute HDFS & YARN deamons
    - Test with an example

Setup the environment

- Check whether you have JDK 7(Java Development Kit)

  $ java –version

  - If the version is not 7(1.7), download and install
  - http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html
  - Make sure you download jdk, not jre only
  - Set $JAVA_HOME properly

    $ export JAVA_HOME=$(/usr/libexec/java_home -v 1.7)
  - If you use OSX, then link /usr/bin/java to /bin/java

    $ sudo ln -s /bin/java /usr/bin/java

Setup the environment

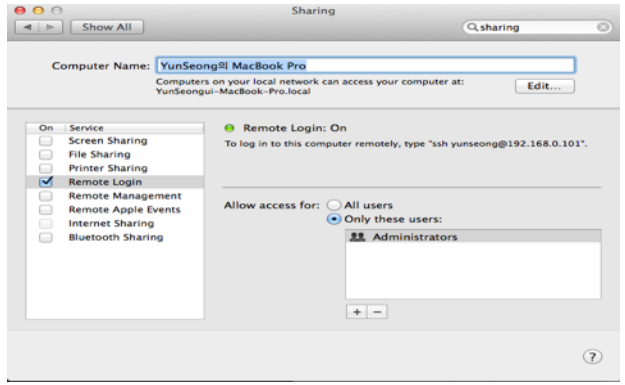
- Secure Shell (SSH)
  - Hadoop uses SSH protocol
  - To use Hadoop on your machine, ssh to localhost must be enabled

    $ ssh-keygen –t rsa –P ‘’ –f ~/.ssh/id_rsa

    $ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

    $ ssh localhost # see if it works

    $ exit
  - In OSX, you can turn on remote login in ‘System Preferences -> Sharing -> remote Login’

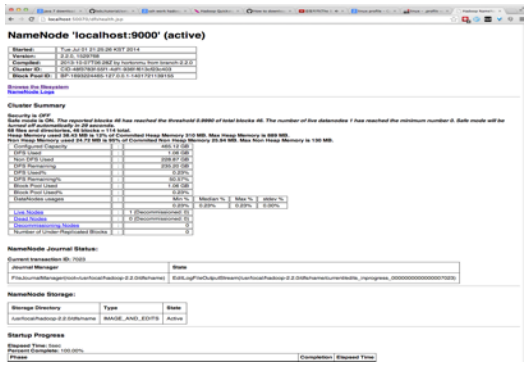


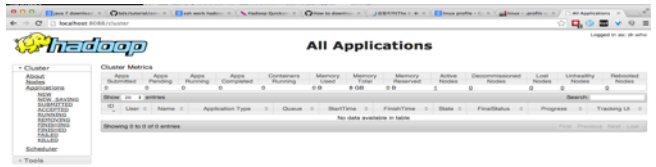
Setup the environment

- Download hadoop & Install
  $ wget http://mirror.apache-kr.org/hadoop/core/hadoop-2.2.0/hadoop-2.2.0.tar.gz
  $ cd {dir_to_install} (your home is OK)
  $ tar xzvf hadoop-2.2.0.tar.gz
  - Set path variables in your profile (~/.profile)
    $ export HADOOP_HOME = {dir_you_installed}
    $ export YARN_HOME = {dir_you_installed}
  - For configurations, you can refer to https://github.com/yunseong/yunseong.github.io/blob/master/reef-tutorial/hadoop_conf.md

Setup the environment

- Check hadoop successfully installed
  - $ HADOOP_HOME/sbin/start-dfs.sh
  - $ HADOOP_HOME/sbin/start-yarn.sh
- You can see it working via web interface
  - Namenode – http://localhost:50070
  - YARN – http://localhost:8088
- Run an example application
  - $ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar pi 20 5





Setup the environment

- To build REEF, you need
  - Java
  - Git
  - Maven
  - Protocol buffer
- IDEs commonly used
  - Eclipse
  - IntelliJ IDEA

Setup the environment

For those who use Mac, using homebrew is recommended (http://brew.sh)

- Java
  - We already covered this
- Git
  - (Linux) sudo apt-get install git
  - (OSX) brew install git
  - $ git --version
- Maven
  - (Linux) sudo apt-get install maven
  - (OSX) brew install maven
  - $ mvn -version # Should be v3.x
- Protocol buffer
  - (Linux) https://code.google.com/p/protobuf/
  - (OSX) brew install protobuf
  - $ protoc --version # Should be v2.5.0

Import REEF Project(Eclipse)

- Build the Project in this order
  - TANG -> WAKE -> REEF
    $ git clone https://github.com/Microsoft-CISL/TANG.git
    $ cd TANG
    $ (Eclipse) mvn -T1C -q -fae -DskipTests clean install eclipse:clean eclipse:eclipse package
    $ (IntelliJ) mvn -T1C -q -fae -DskipTests clean install
  - Repeat with Wake and REEF
  - Set $REEF_HOME as the dir REEF installed
- Import in Eclipse
  - File > Import > Existing project into Workspace
- Import in IntelliJ
  - File > Import Project > import as a Maven Project

Hadoop & REEF Overview

Brian Cho

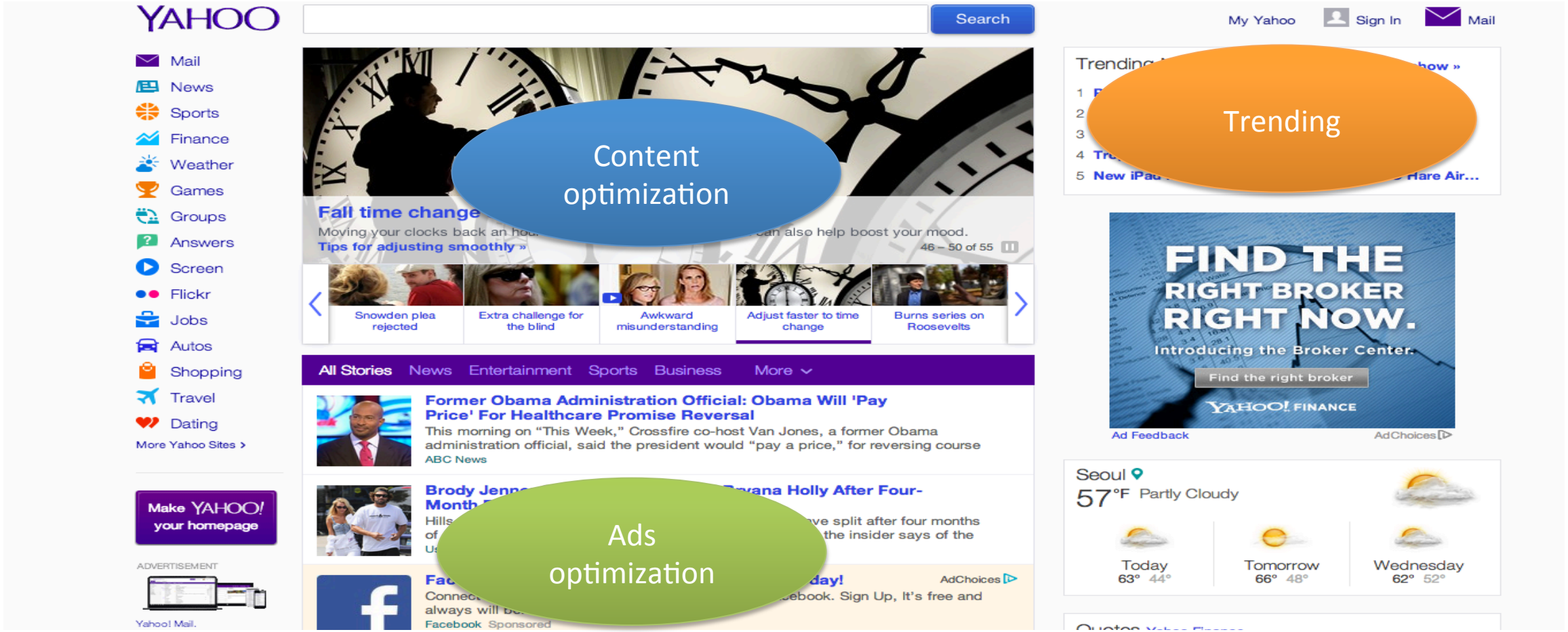
TOWARDS AN OPERATING SYSTEM FOR BIG DATA

March 31, 2014

Byung-Gon Chun
CMS Lab, CSE Dept., SNU

Collaboration with Microsoft CISL and UCLA
[Credit: many slides from Microsoft CISL]

Powered by Big Data Analytics



Y!

in

ebay

Microsoft

NETFLIX

PANDORA

amazon.com

Google

Pinterest

hadoop

mahout

HIVE

hadoop

APACHE HBASE

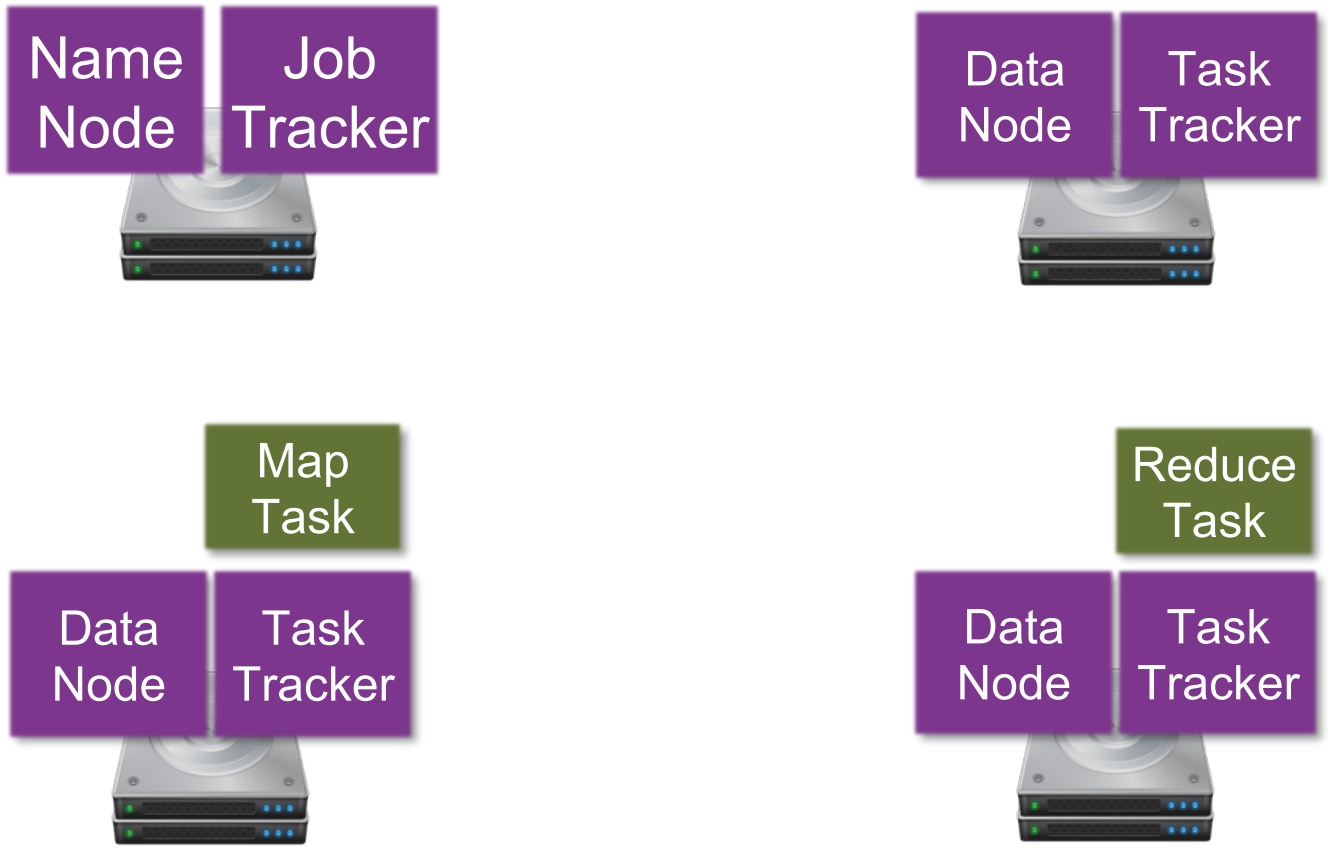
GIRAPH

Storm

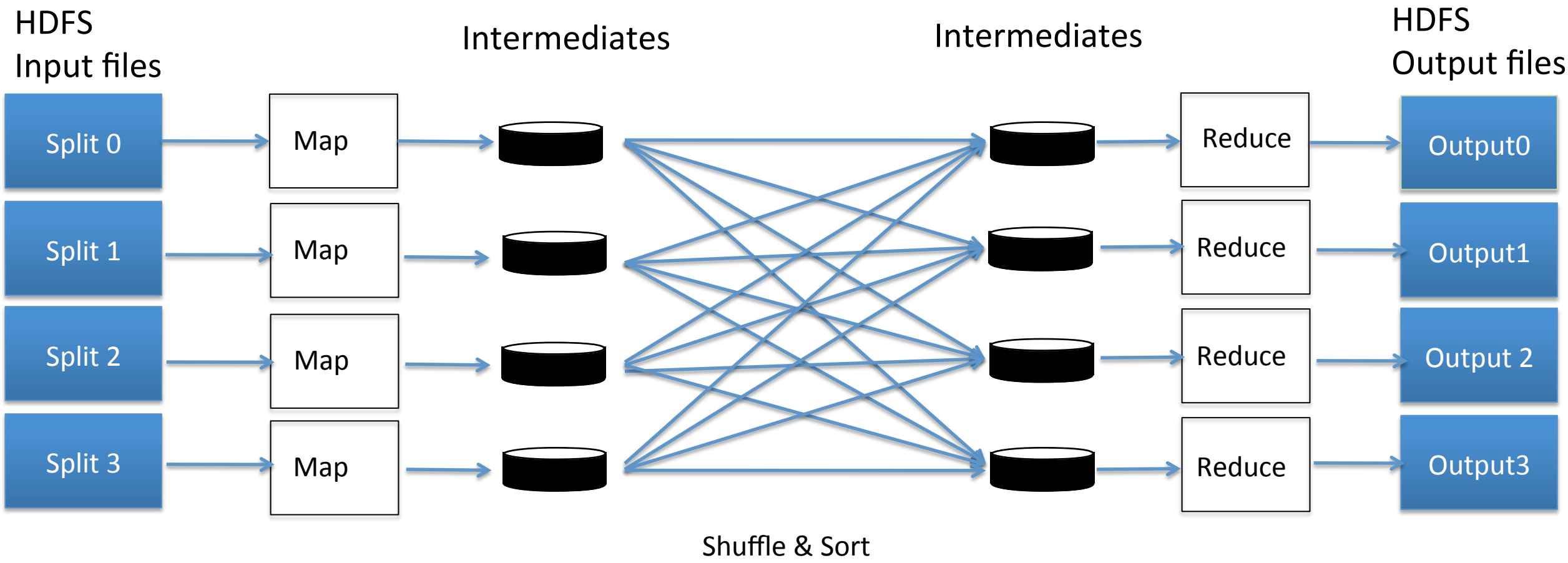
Impala

OOZIE

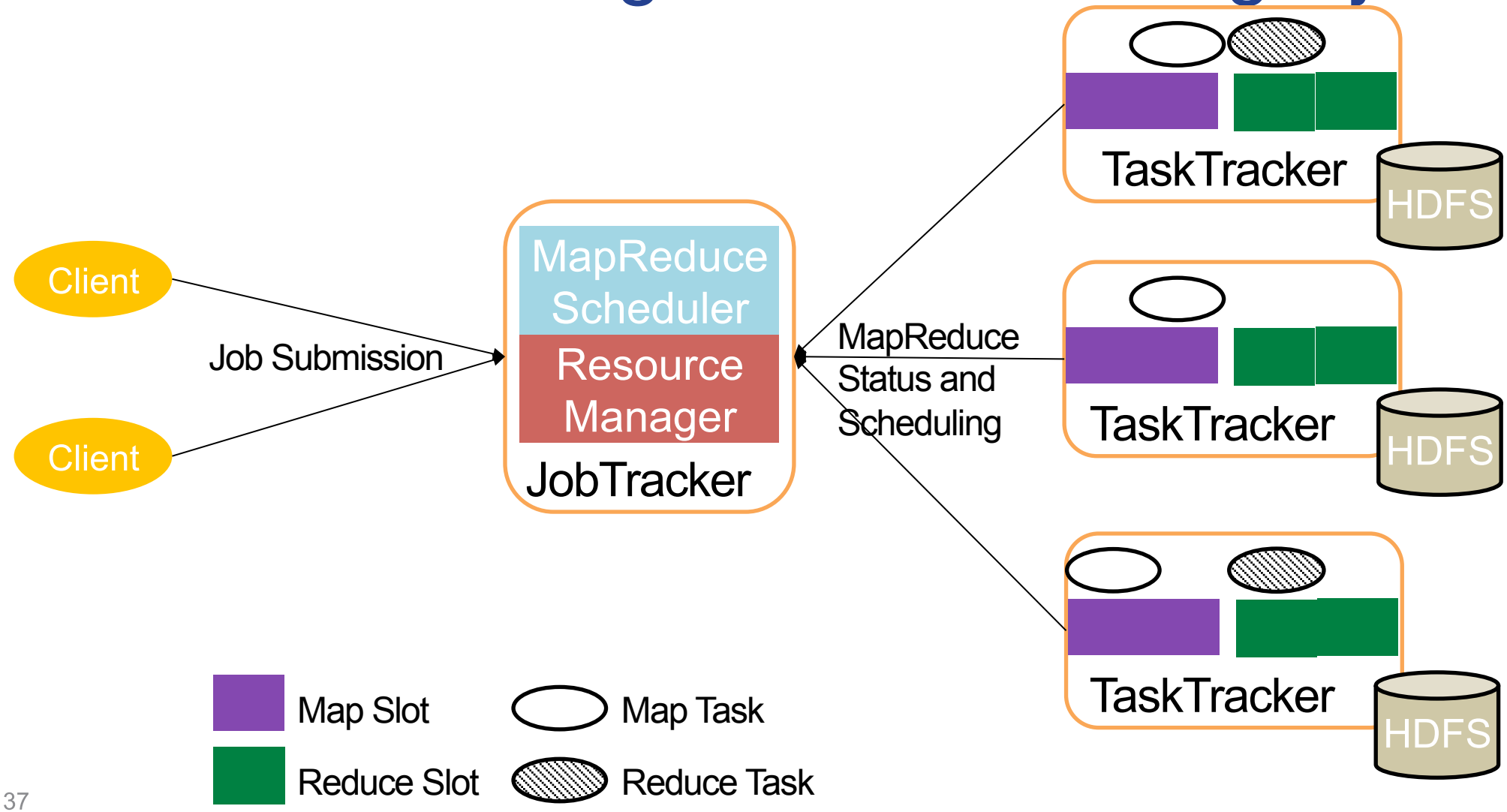
Hadoop v1


Name Node
Job Tracker
Data Node
Task Tracker
Map Task
Data Node
Task Tracker
Reduce Task
Data Node
Task Tracker

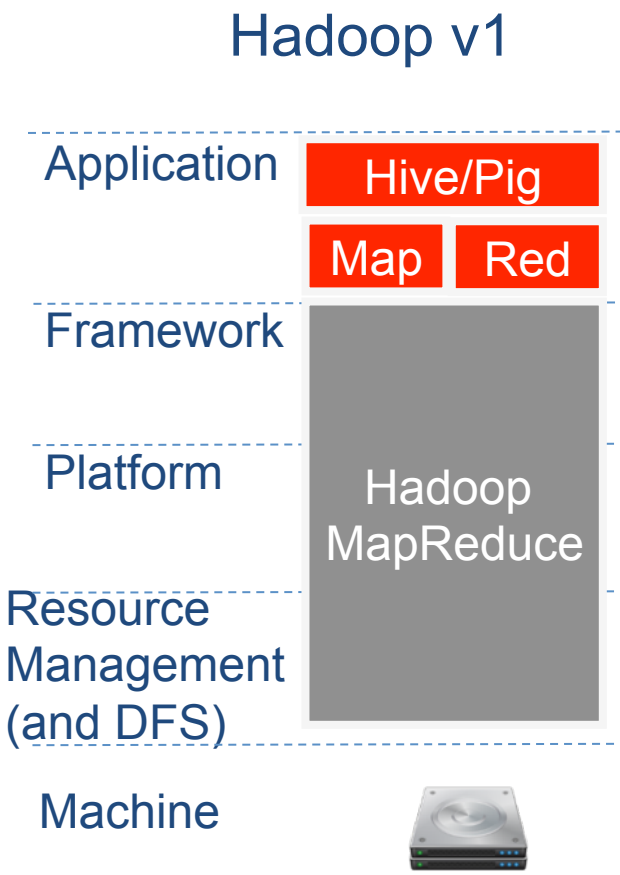
Hadoop MapReduce



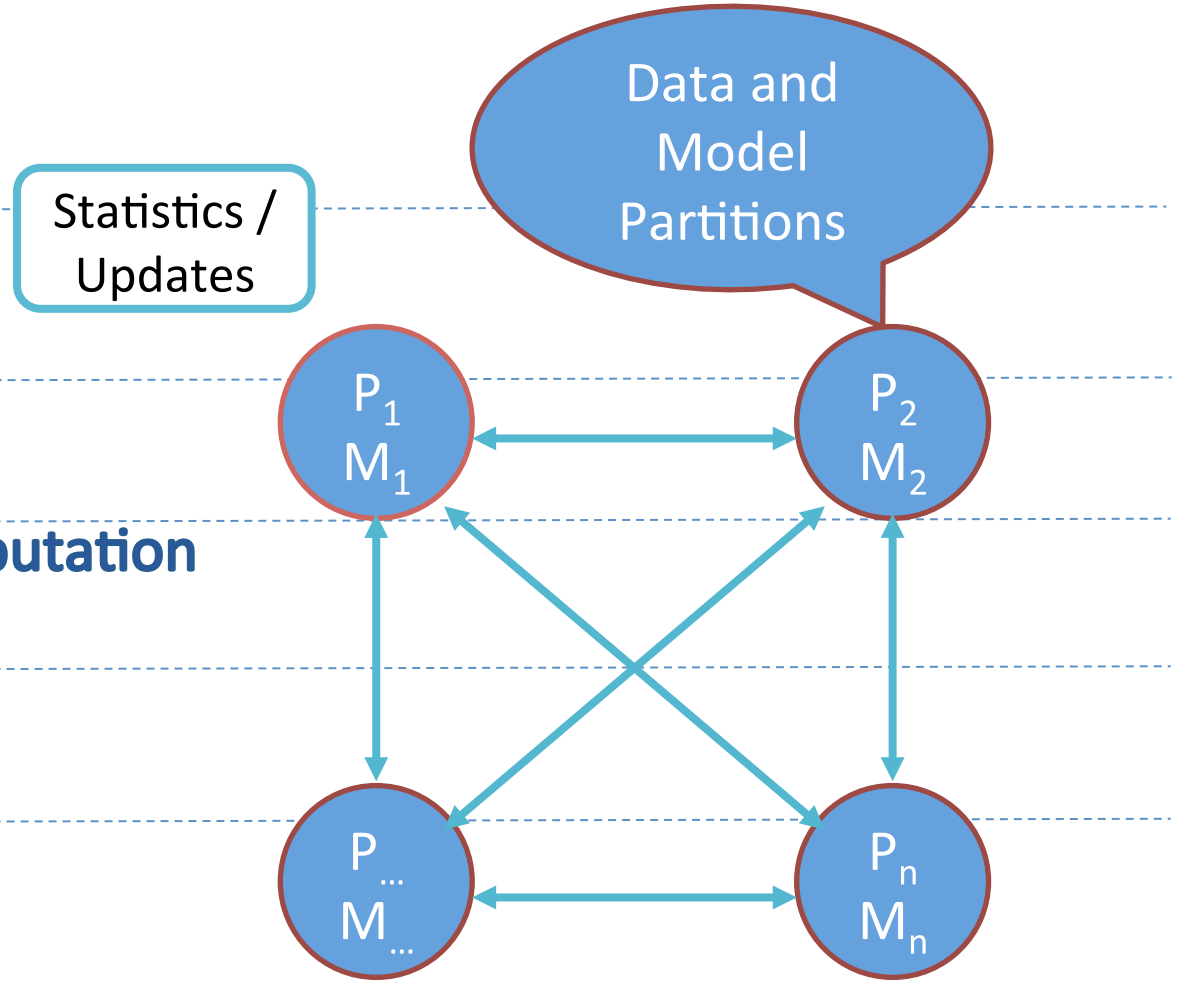
Hadoop v1
Monolithic Big Data Processing System



Hadoop v1



Hadoop v1 Abuse
- **Inefficient computation**
- **Map-only jobs**



Problems with Hadoop v1

Fault Tolerance Mismatch

MapReduce fault tolerance actually gets in the way

Resource Model Mismatch

MapReduce's resource selection often suboptimal for the job at hand (data local vs. communication)

Cumbersome integration with M/R

Every Abuser has to implement …

Networking

Cluster Membership

Bulk data transfers

Abusers Violate MapReduce assumptions

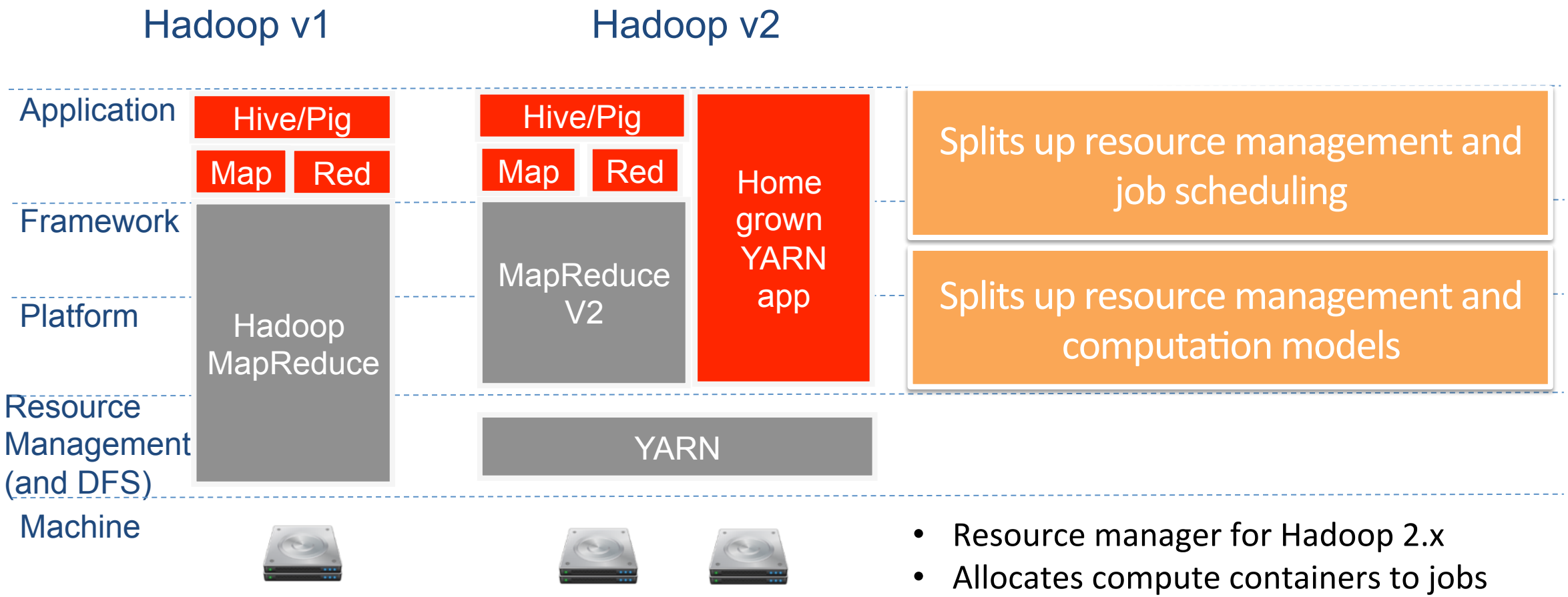
Network usage bursts in (All)Reduce

Local disk use in VW

Inefficient resource utilization

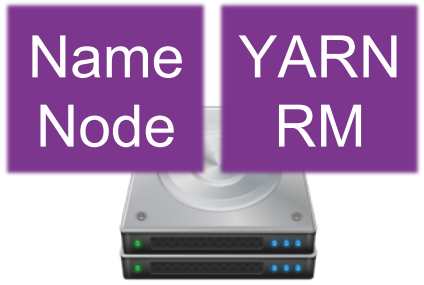
Job tracker scaling concerns

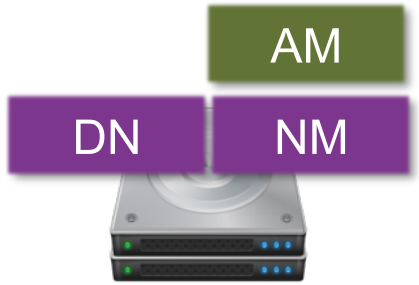
YARN: A Step Towards Refactoring Big Data Processing Systems

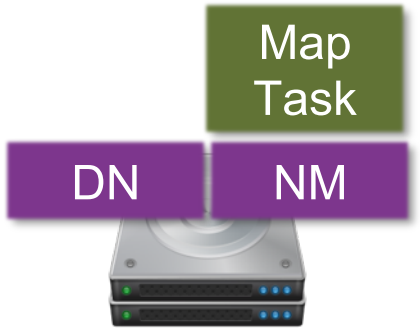


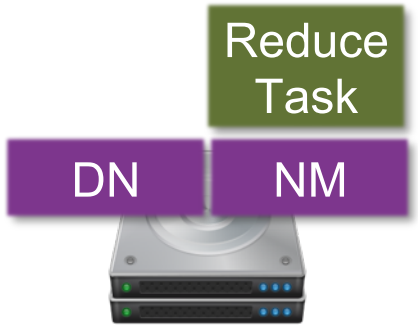
- Resource manager for Hadoop 2.x
- Allocates compute containers to jobs
- Other resource managers: Mesos, Omega

Background: YARN/HDFS





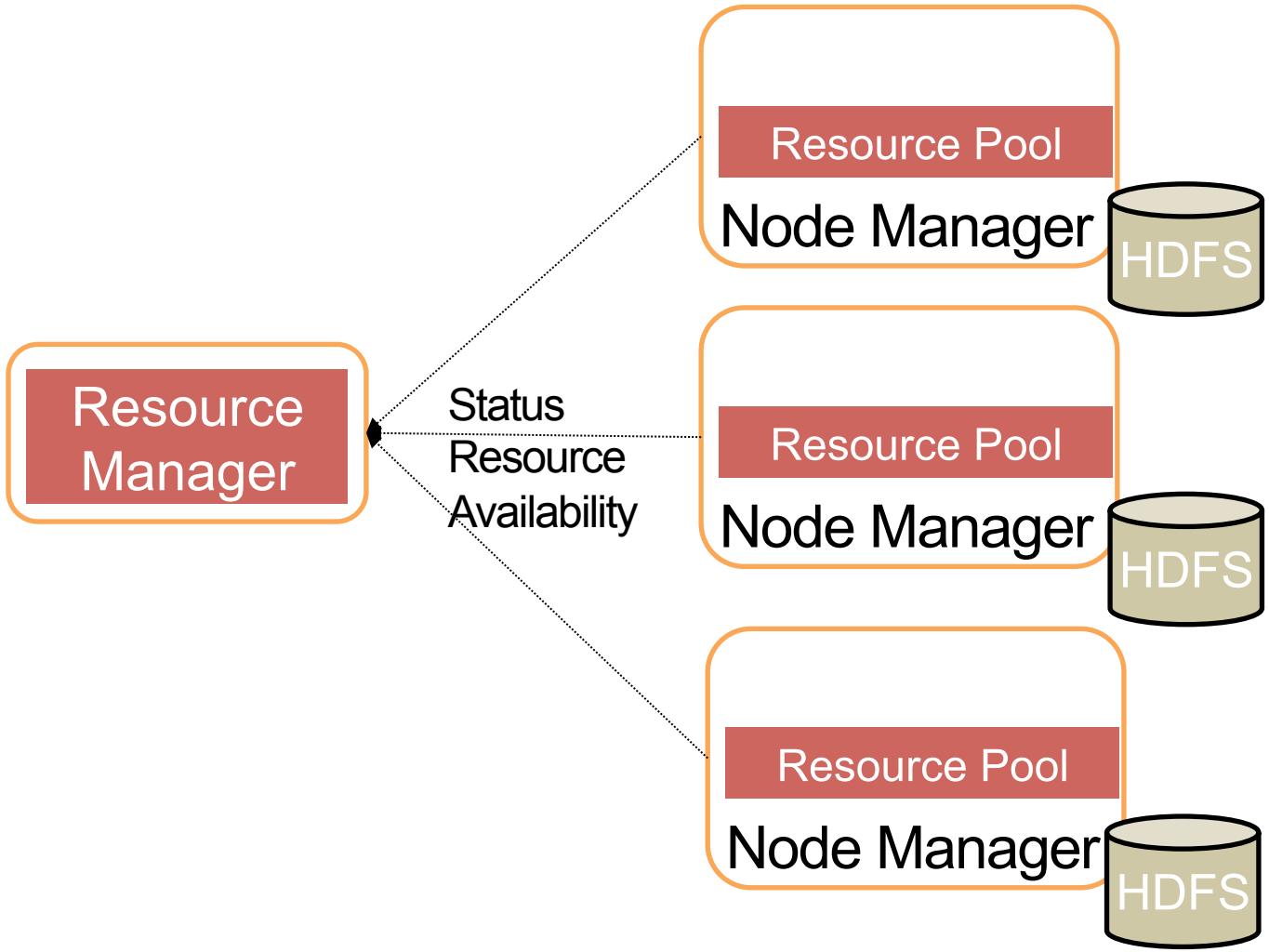




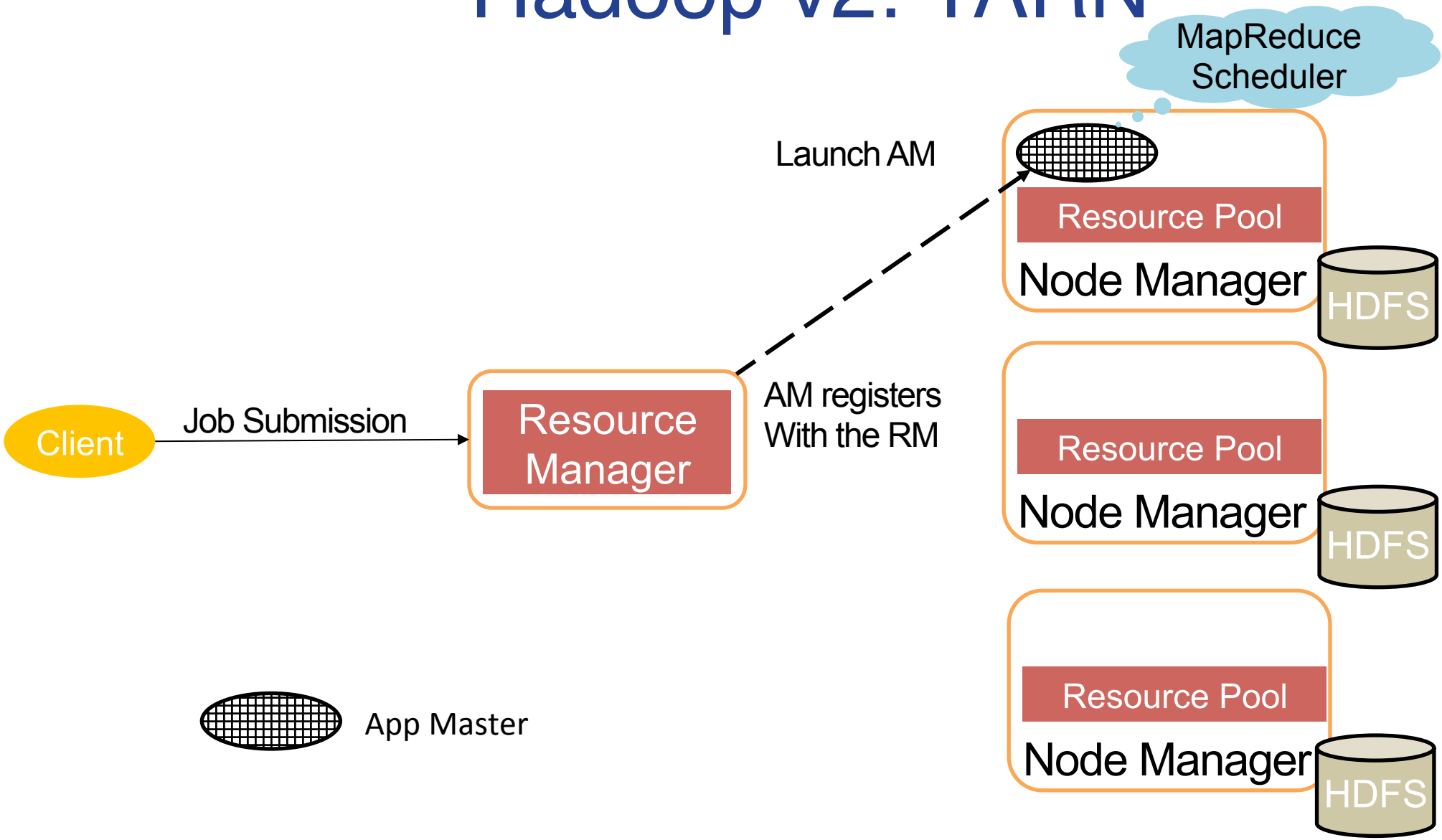
Application Execution in YARN

1. Application submission
2. Boostraping the ApplicationMaster (AM) instance for the application
3. Application execution managed by the AM instance

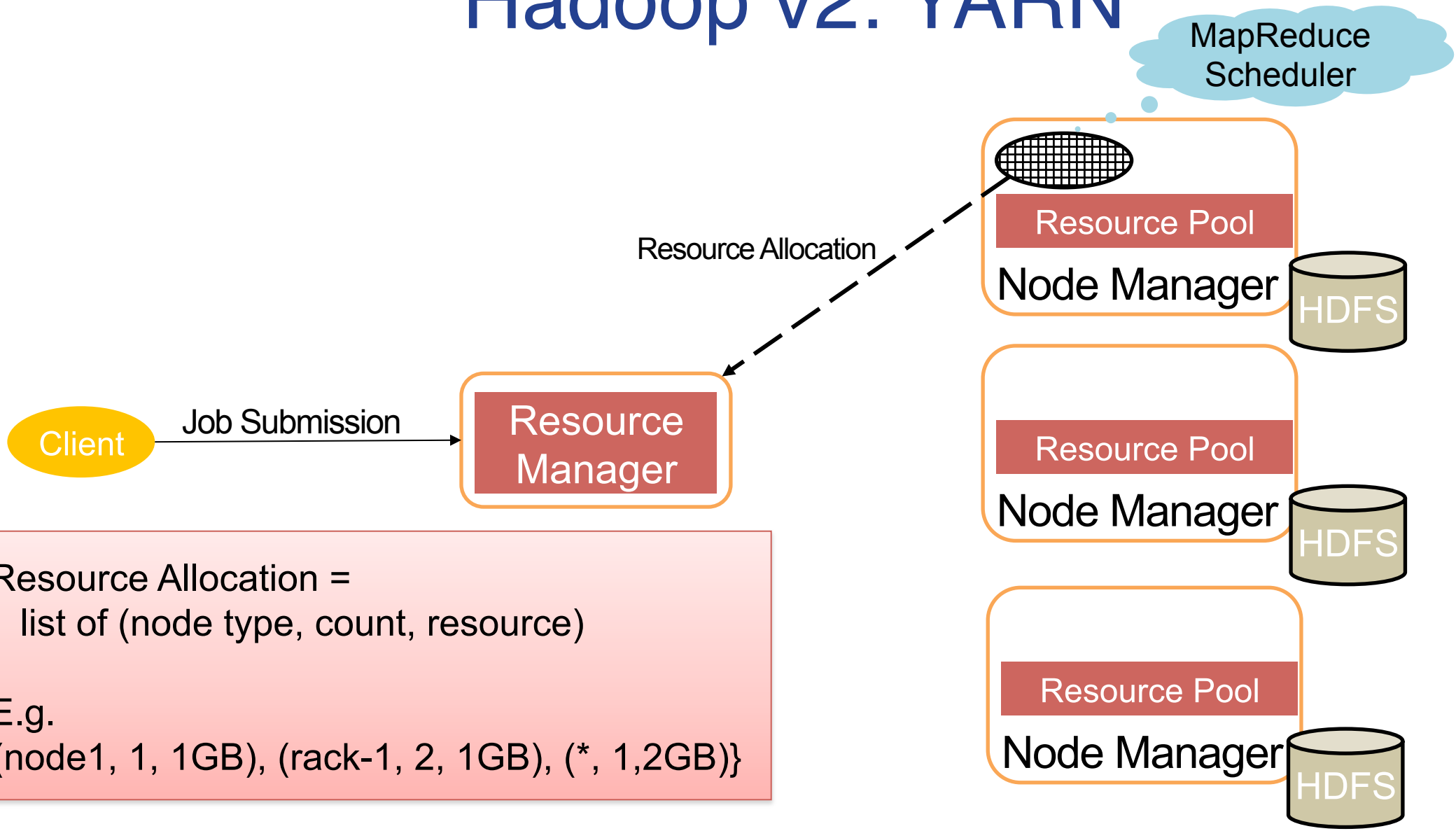
Hadoop v2: YARN



Hadoop v2: YARN



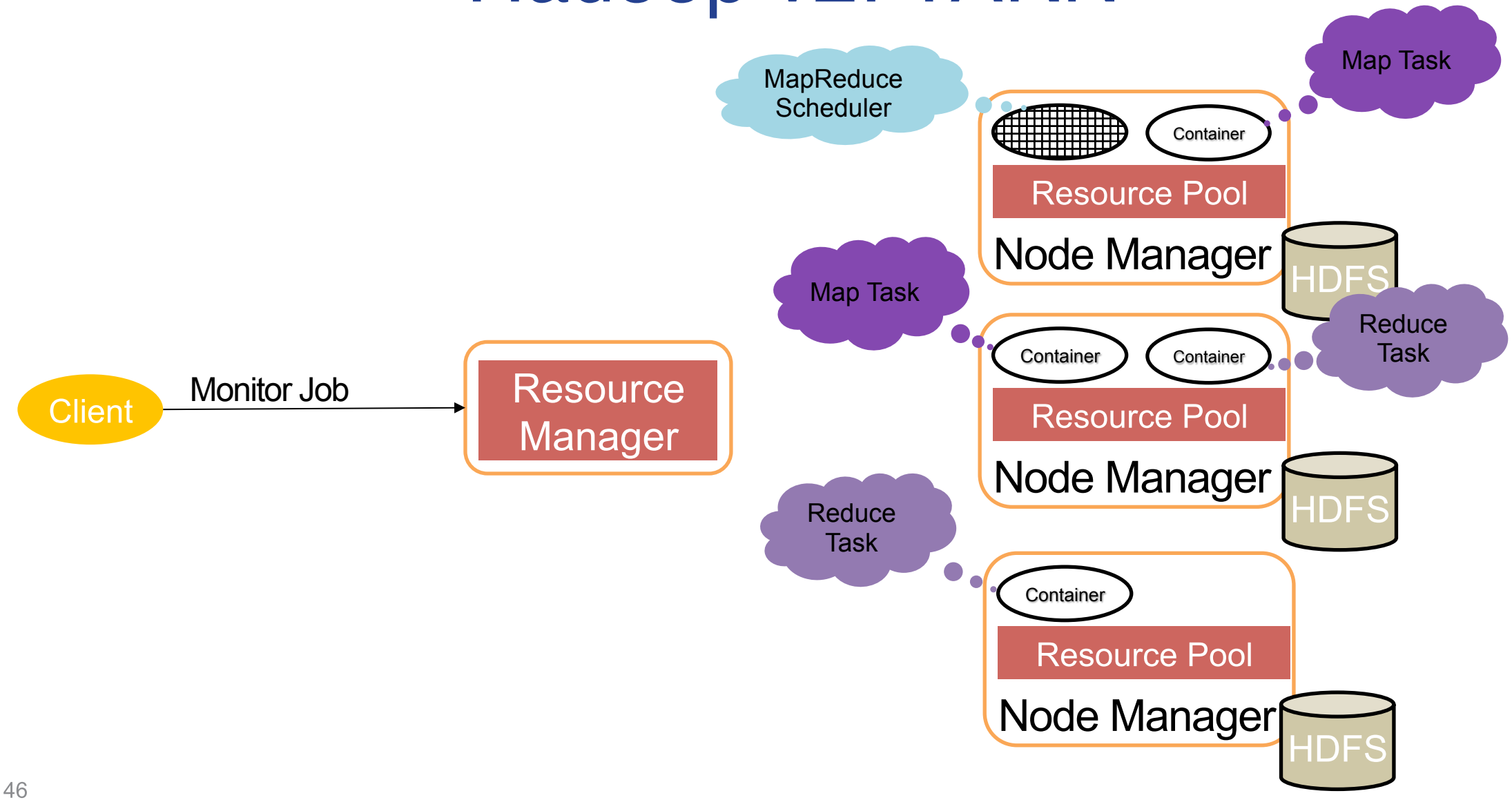
Hadoop v2: YARN



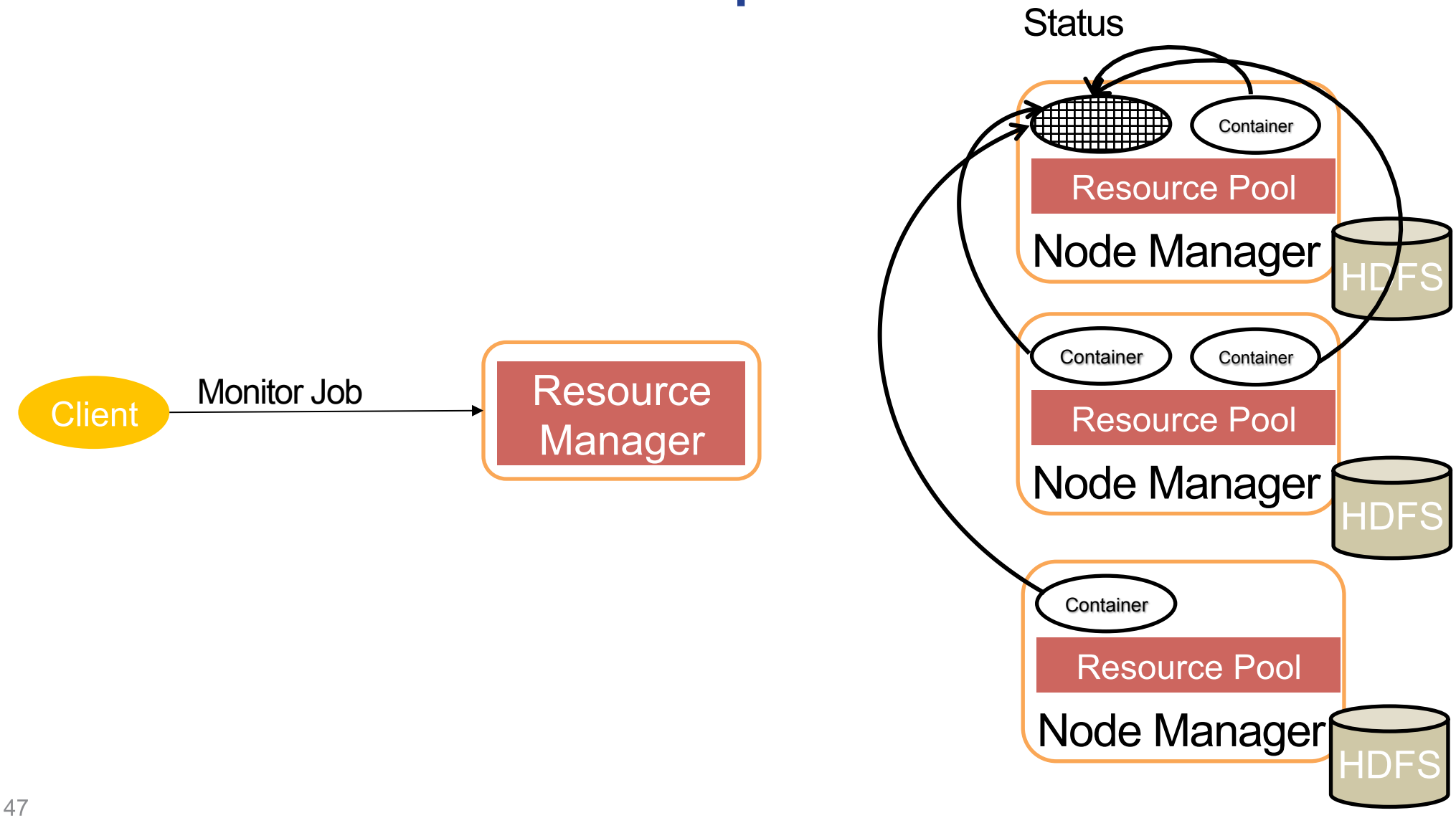
Resource Allocation =
list of (node type, count, resource)

E.g.
{(node1, 1, 1GB), (rack-1, 2, 1GB), (*, 1,2GB)}

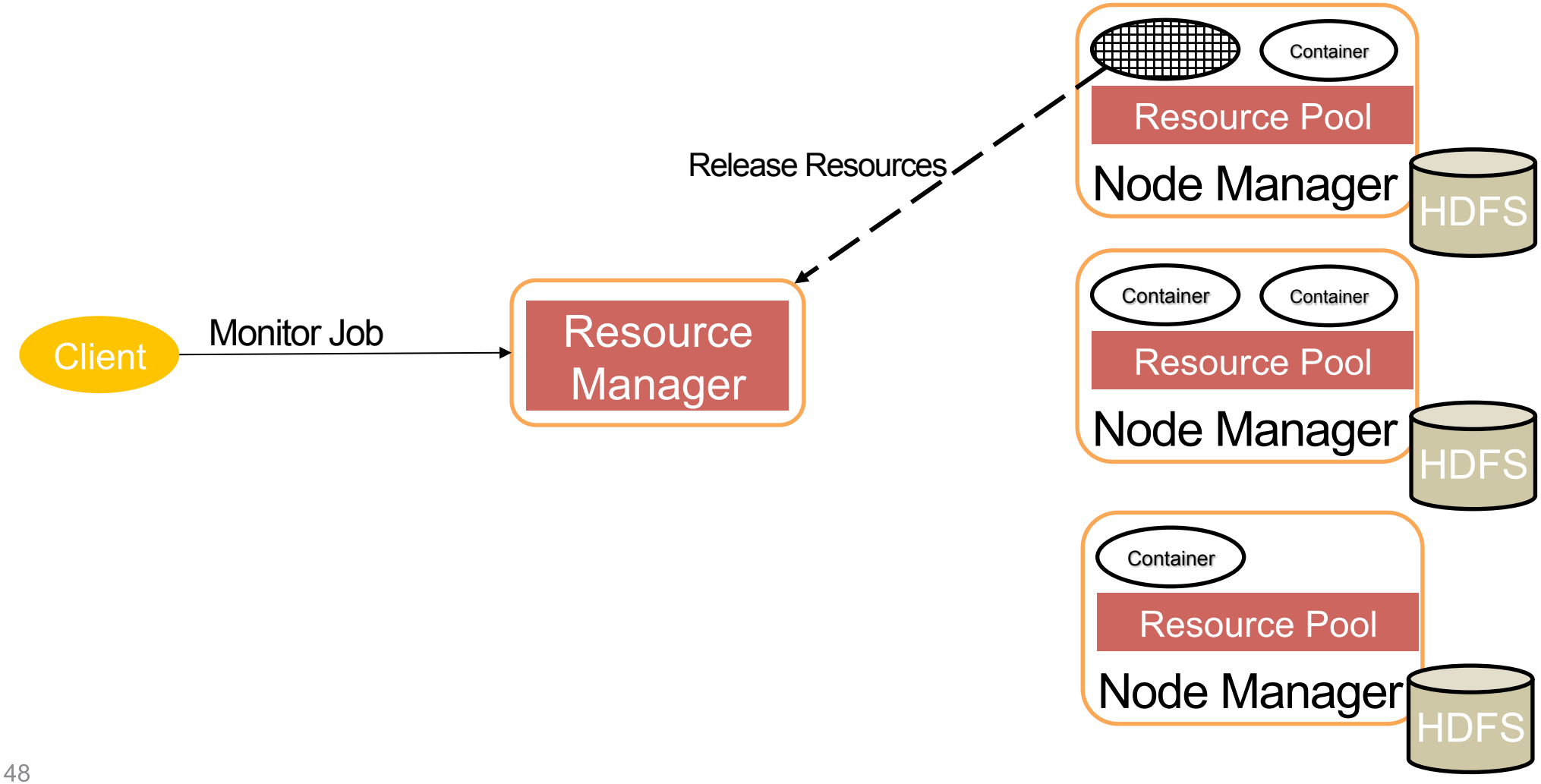
Hadoop v2: YARN



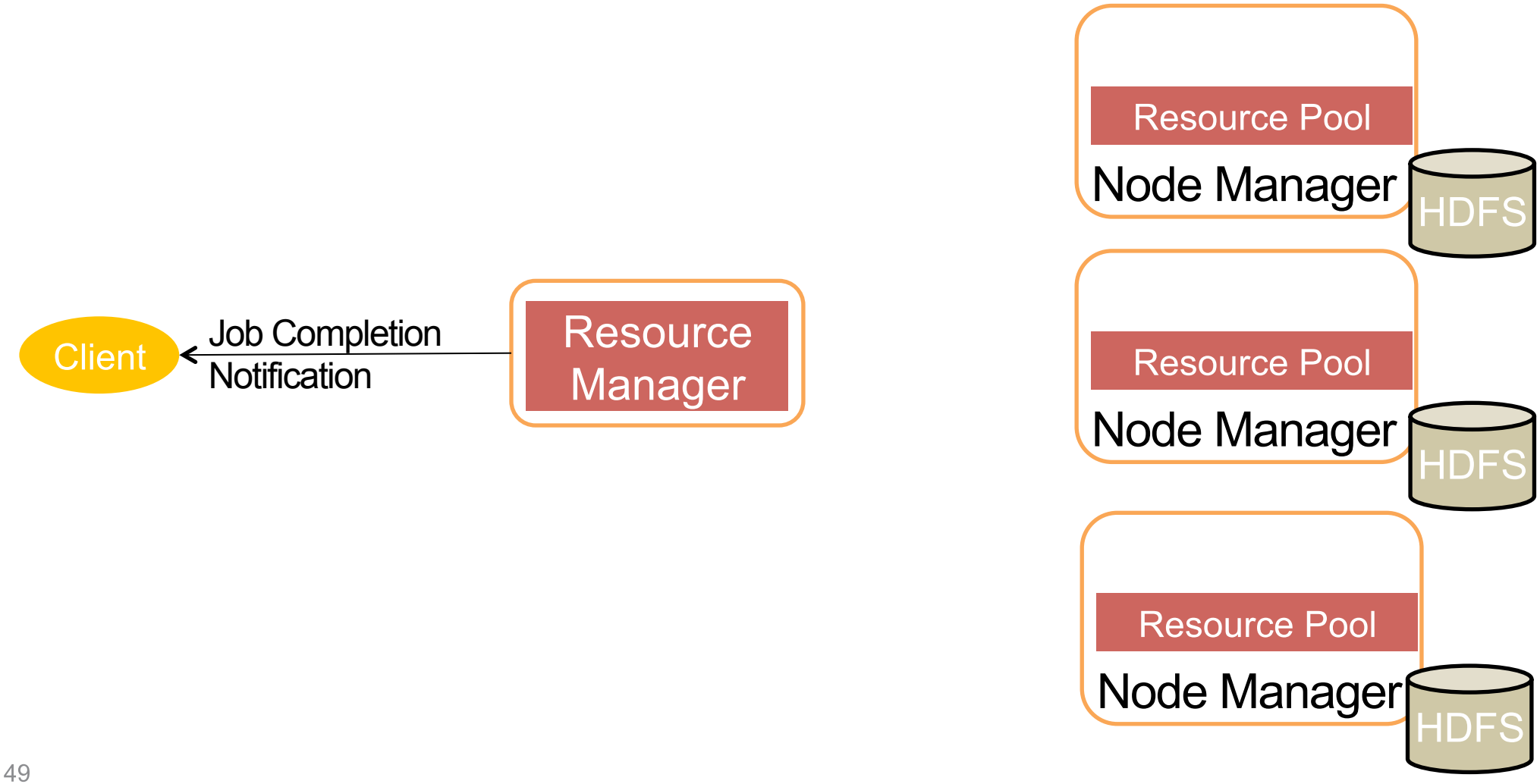
Hadoop v2: YARN



Hadoop v2: YARN



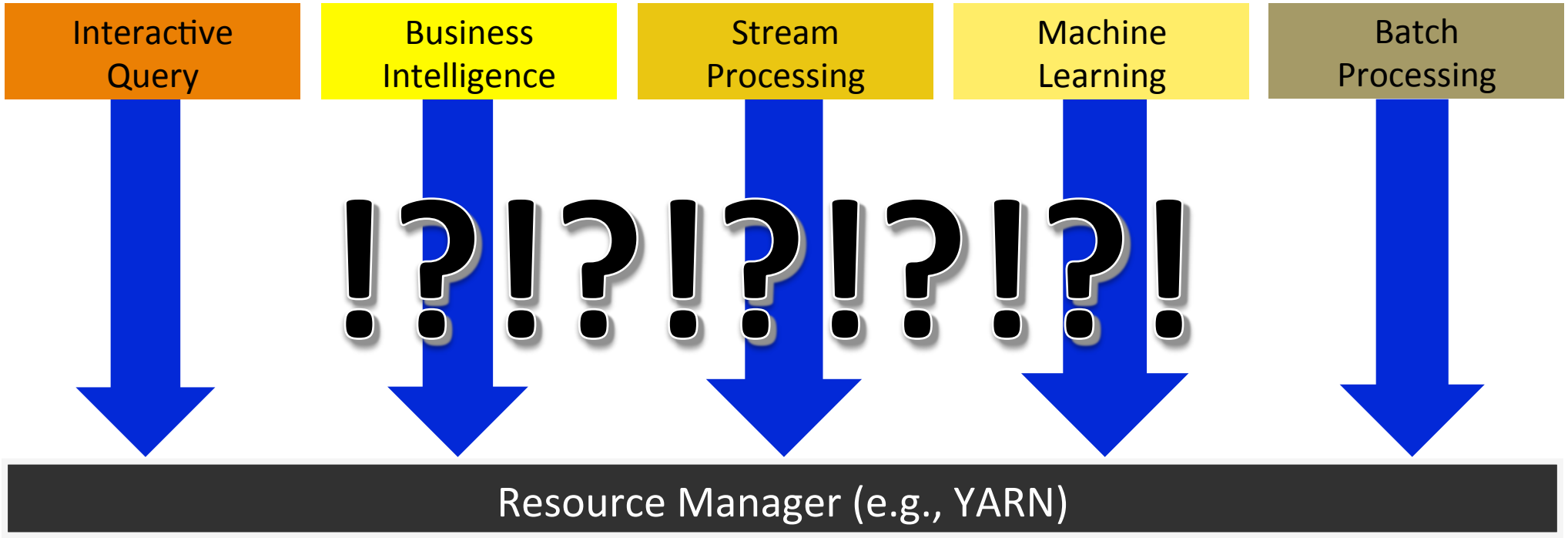
Hadoop v2: YARN



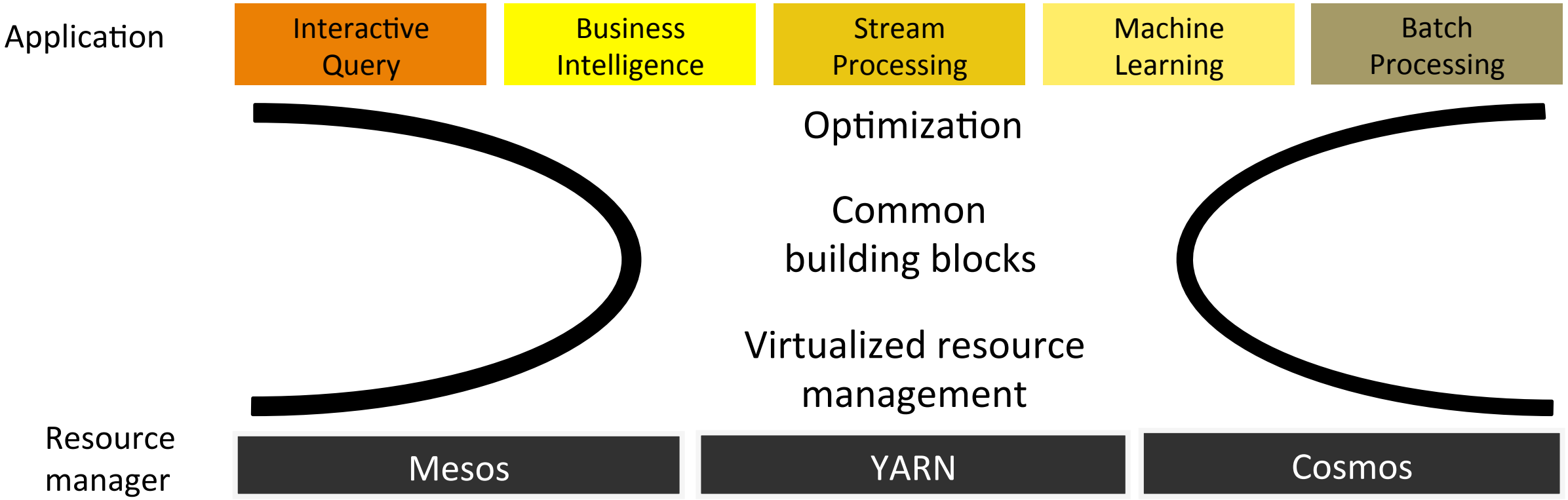
YARN

- Industry
  - Active deployment
    - E.g., Yahoo! Data analytics cluster: 100% YARN
  - Active development: “JIRA issues”
- Academia
  - Active research on resource management
- Interesting topics: API that YARN exposes, High Availability, etc.

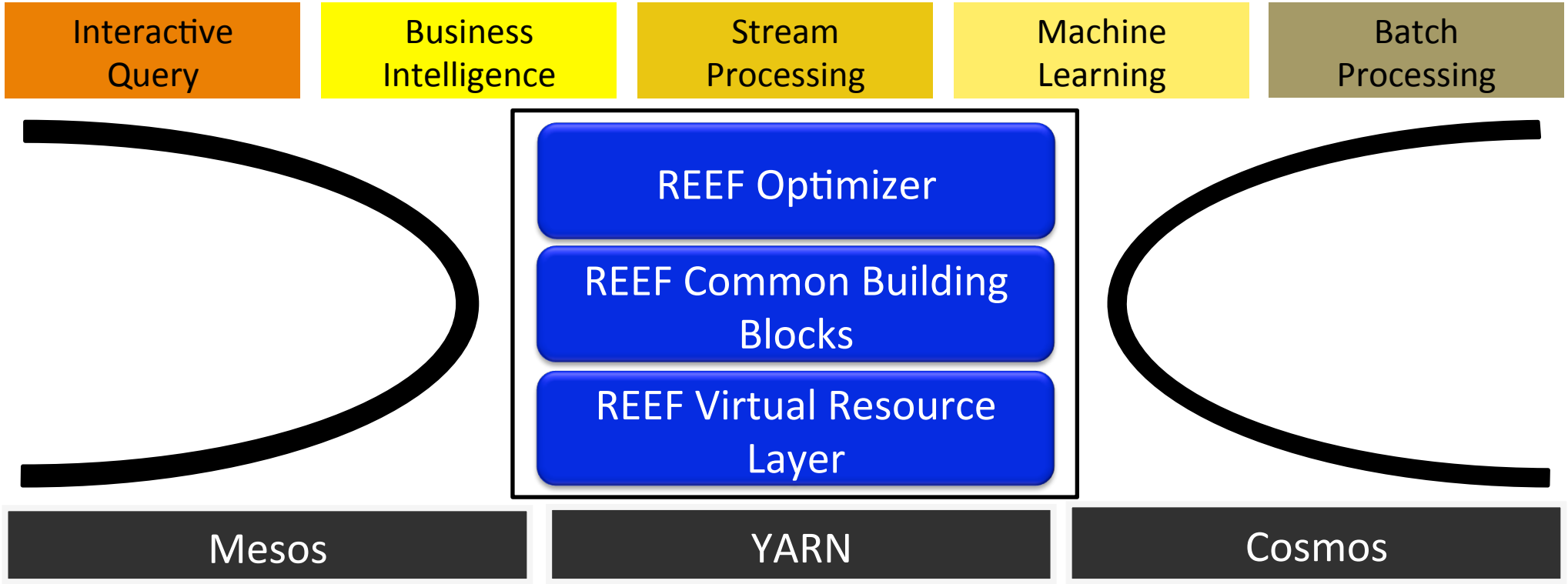
Challenges


Interactive Query
Business Intelligence
Stream Processing
Machine Learning
Batch Processing
!?!?!?!?!?!
Resource Manager (e.g., YARN)

My Research:
An Operating System for Big Data

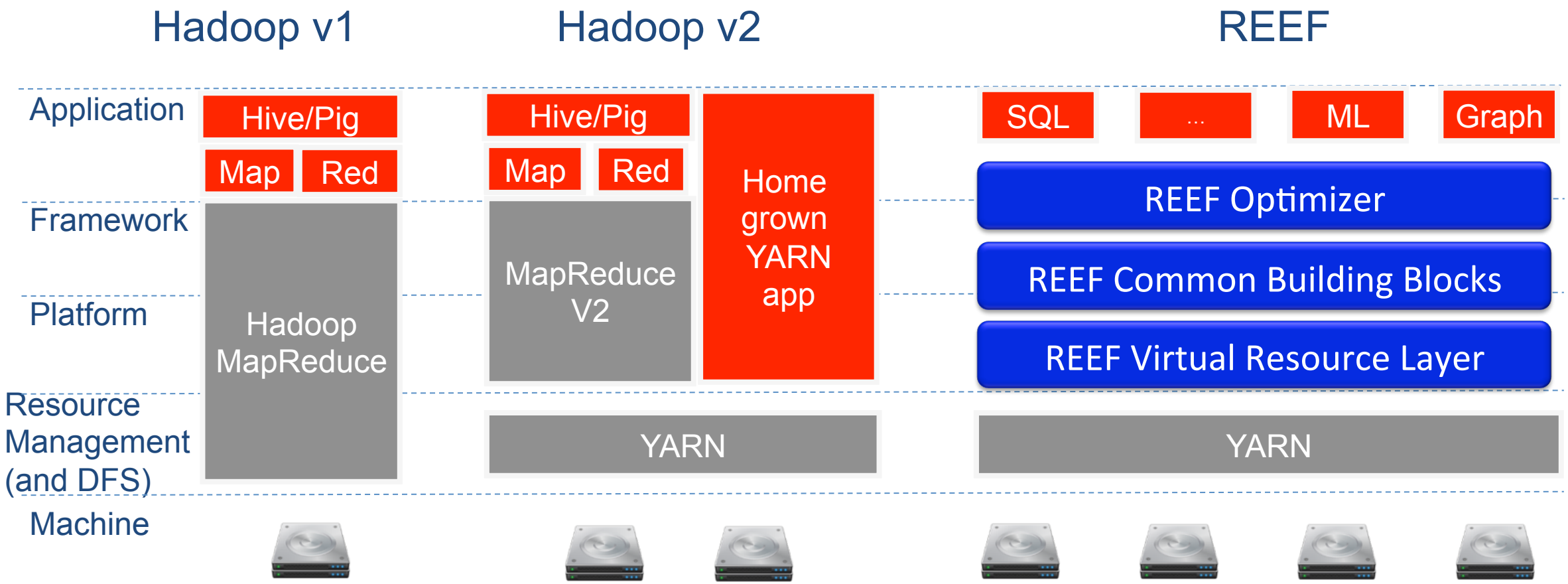


REEF: Retainable Evaluator Execution Framework



- Eases the development of Big Data applications
- Reuses containers across tasks and retains state in a container
- Simplifies configuration management and event handling
- Open source: www.reef-project.org; Users: a microsoft product team, a Korean company

REEF: Retainable Evaluator Execution Framework



REEF Value Proposition

Ease development of Resource Manager (e.g., YARN) applications

- Cluster membership: Heartbeats, Failure notification, etc.
- Networking: Naming, Message Passing, Group Communications, etc.
- State management: Checkpointing, Storage, etc.

Enable to retain state in a container and reuse containers across tasks

- Hand-over state and resource between different operators in a pipeline
- Retain state across iterations in iterative or recursive distributed programs
- Enable pipelines of different computations
  E.g. Map/Reduce followed by MPI followed by stream processing

Configuration management and event handling

Cross-language framework: Java and .NET

REEF: Control and Data Plane

Extensible Control Flow

Driver — **User code** that implements the resource allocation and task scheduling components.

Evaluator — Execution Environment on a YARN container that can retain state within **Contexts** and execute **Tasks**.

Context — A stackable state management environment within an **Evaluator.** **Context** is accessible to any **Task** hosted by that **Evaluator.**

Task — **User code** executed within an **Evaluator**.

Data Management Services

Storage
- Abstractions: Map and Spool
- Local and Remote

Network
- Message passing
- Bulk Transfers
- Collective Communications

State Management
- Fault Tolerance, Preemption
- Checkpointing

Service

REEF Control Plane

Control Flow is centralized in the Driver

- Evaluator allocation & configuration
- Task configuration & submission

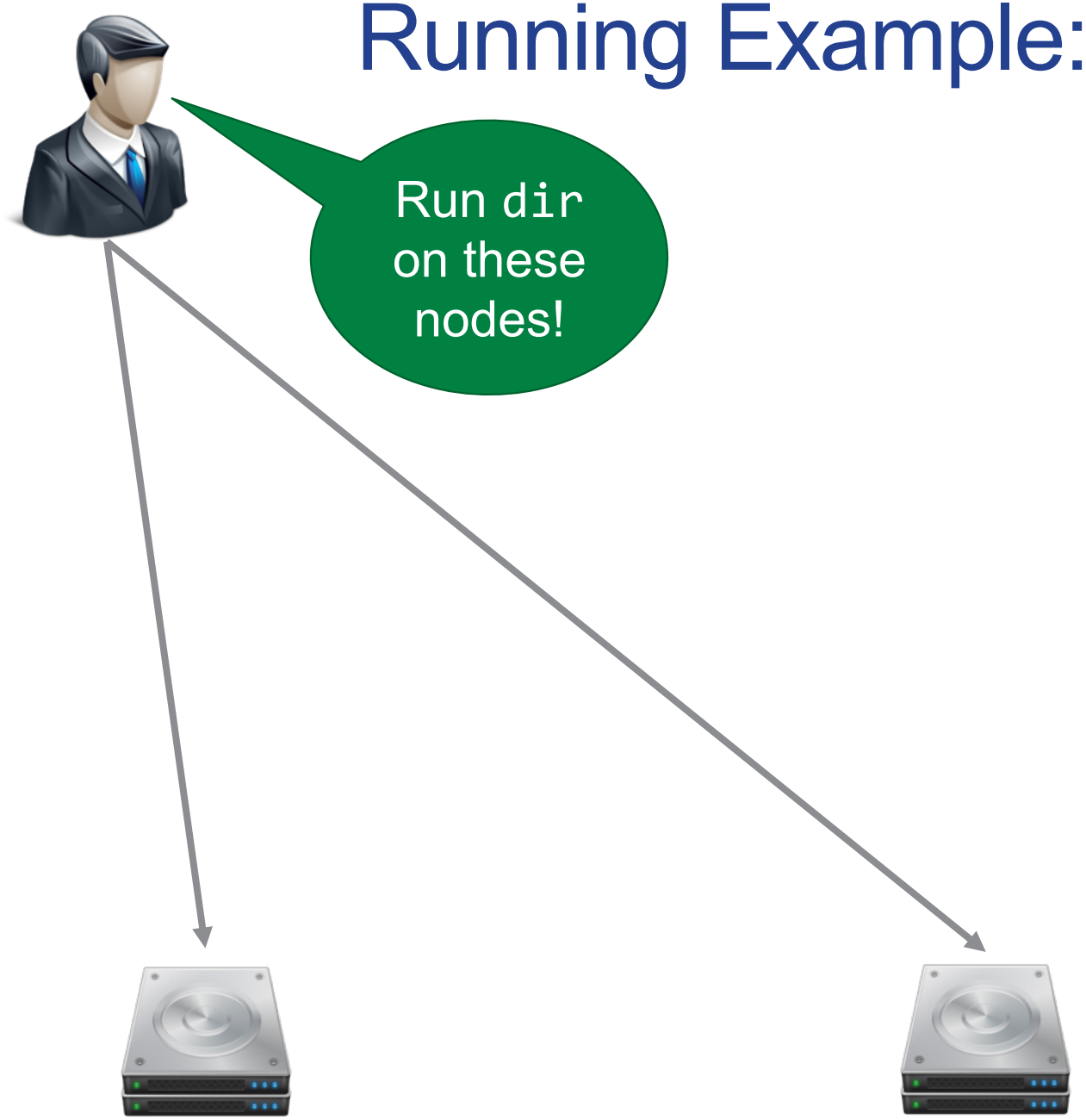
Error Handling is centralized in the Driver

- When a Task throws an Exception, we ship & throw it at the Driver
- When an Evaluator dies, we throw an Exception at the Driver

All APIs are Rx-Style asynchronous APIs (Wake)

- Driver files requests via non-blocking API calls
- REEF fires events at user-provided event Observers (e.g. Evaluator availability, Exceptions, …)
- Majority of the state keeping (e.g. work queues) is maintained by the Driver.

Running Example: Distributed Shell



REEF Control Flow: Job Life Cycle

Client

submit job

Name Node

YARN RM

launch container

REEFRuntime

DN

NM

DN

NM

DN

NM

```
public class DistributedShell {
  ...
  public static void main(String[] args){
    ...
    Injector i = new Injector(yarnConfiguration);
    ...
    REEF reef = i.getInstance(REEF.class);
    ...
    reef.submit(driverConf);
  }
}
```

REEF Control Flow

Client

submit job

Name Node

YARN RM

launch container

Driver

REEFRuntime

DN

NM

DN

NM

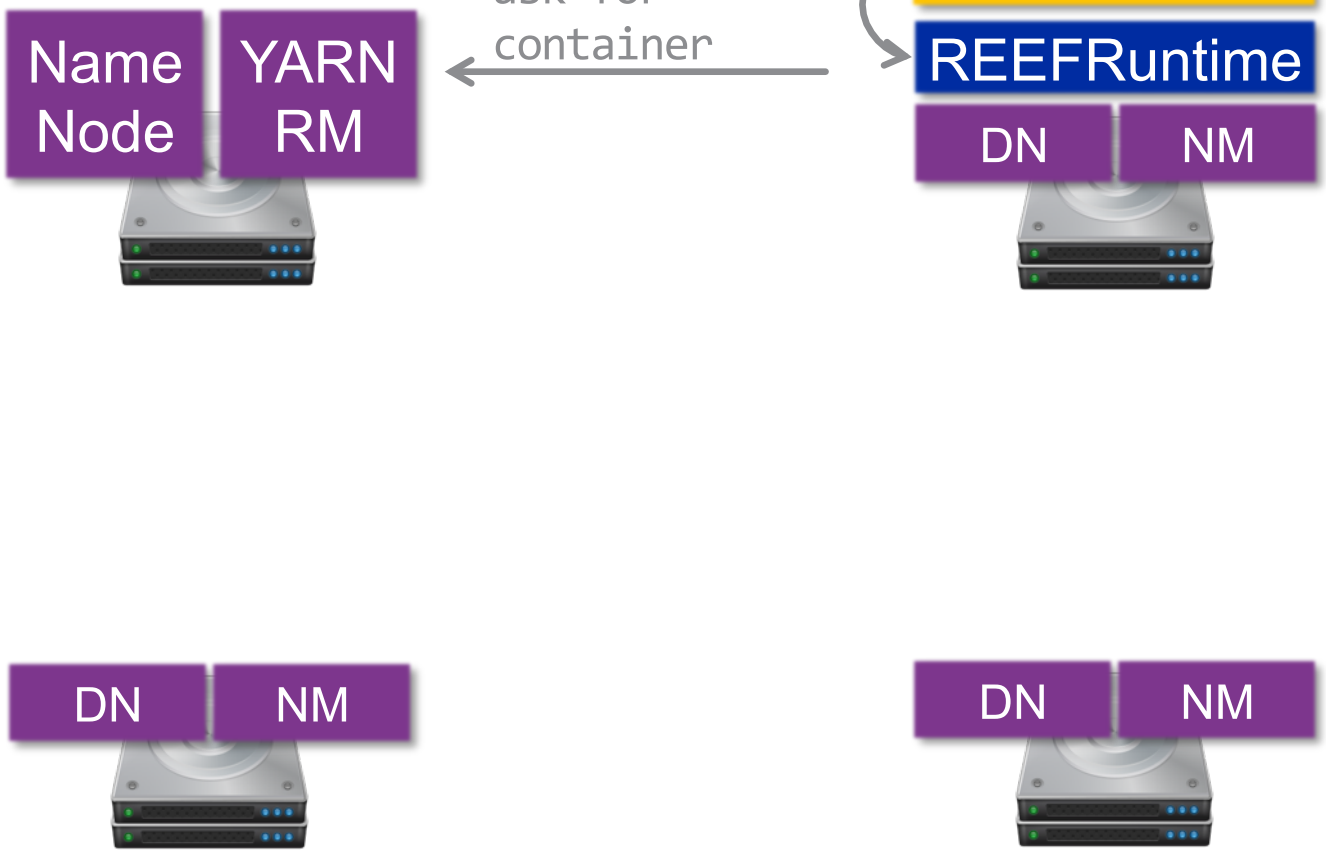
DN

NM

```
public class DistributedShell {
  ...
  public static void main(String[] args){
    ...
    Injector i = new Injector(yarnConfiguration);
    ...
    REEF reef = i.getInstance(REEF.class);
    ...
    reef.submit(driverConf);
  }
}
```

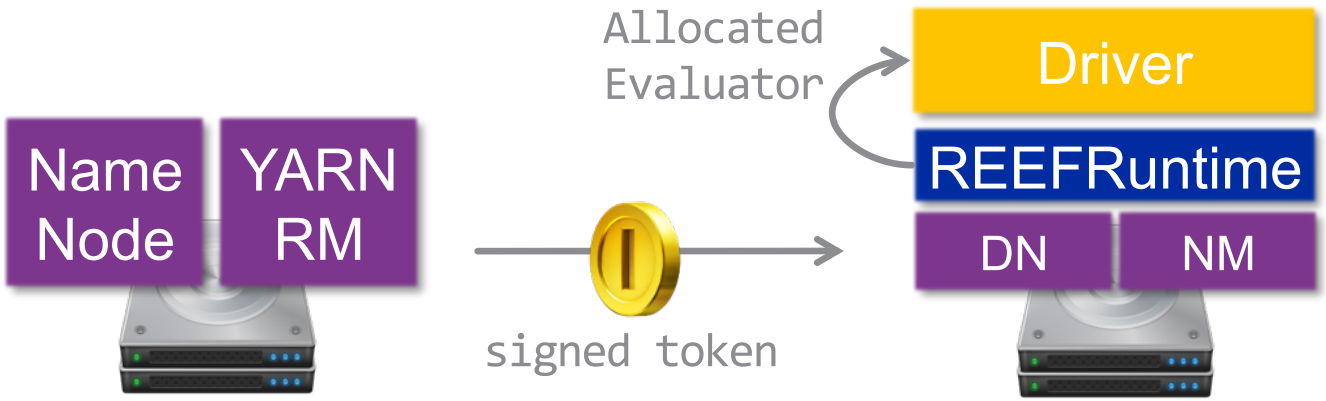
REEF Control Flow

Client



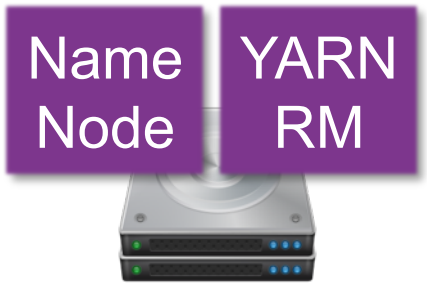

```java
public class DistributedShellJobDriver {
  private final EvaluatorRequestor requestor;
  ...

  public void onNext(StartTime time) {

    requestor.submit(EvaluatorRequest.Builder()
                     .setSize(SMALL).setNumber(2)
                     .build()
                     );

  }

  ...
}
```

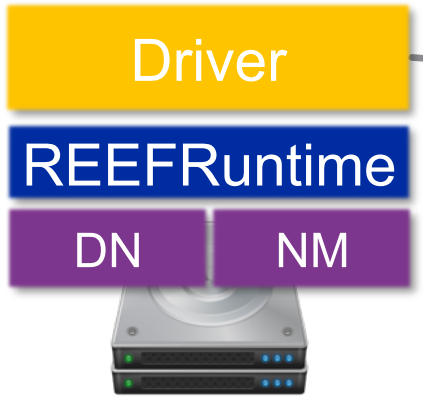
REEF Control Flow

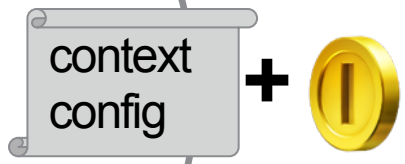


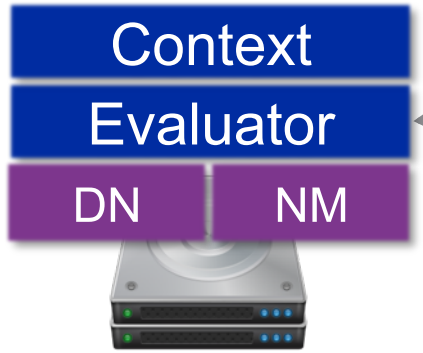

```
public class DistributedShellJobDriver {
  private final EvaluatorRequestor requestor;
  ...

  public void onNext(AllocatedEvaluator eval) {
    Configuration contextConf = ...;
    eval.submitContext(contextConf)
  }

  ...
}
```

REEF Control Flow

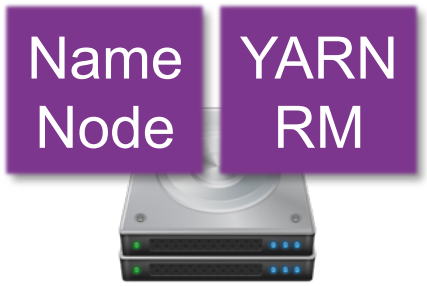


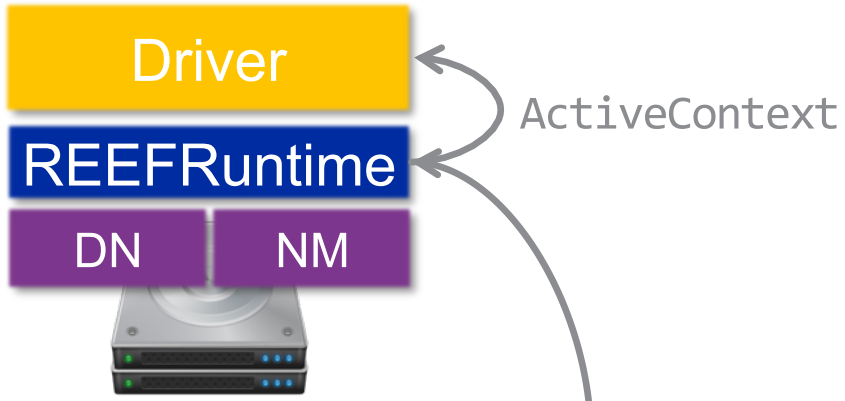


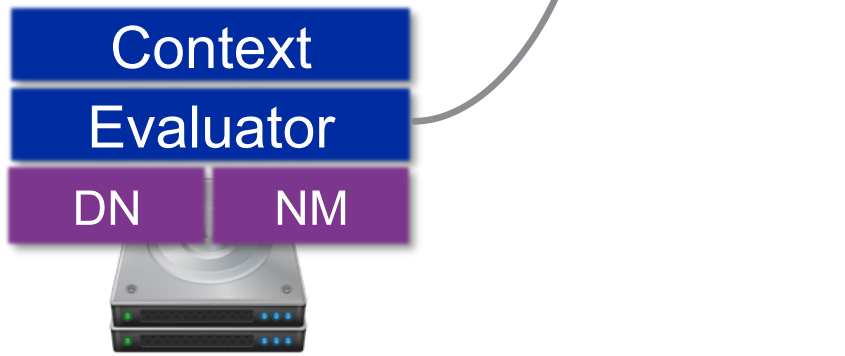




REEF Control Flow

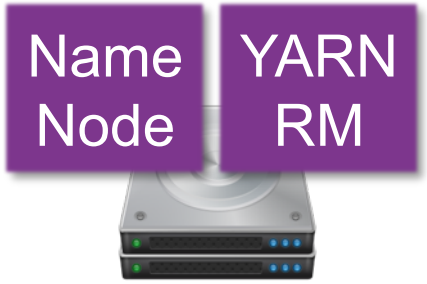


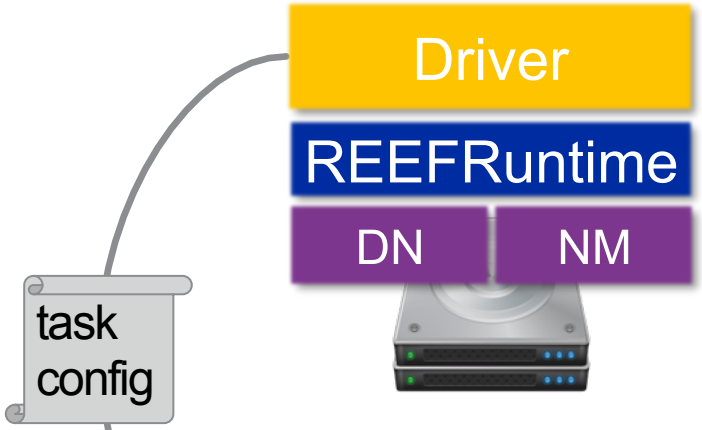




REEF Control Flow

Client






```
public class DistributedShellJobDriver {
  private final String cmd = “dir”;

  [...]

  public void onNext(ActiveContext ctx) {
    final String taskId = [...];

    Configuration taskConf = Task.CONF
            .set(IDENTIFIER, "ShellTask")
            .set(TASK, ShellTask.class)
            .set(COMMAND, this.cmd)
            .build();

    ctx.submitTask(taskConf);
  }

  [...]

}
```

REEF Control Flow

Client

Name Node | YARN RM

DN | NM

Driver

REEFRuntime

DN | NM

Task

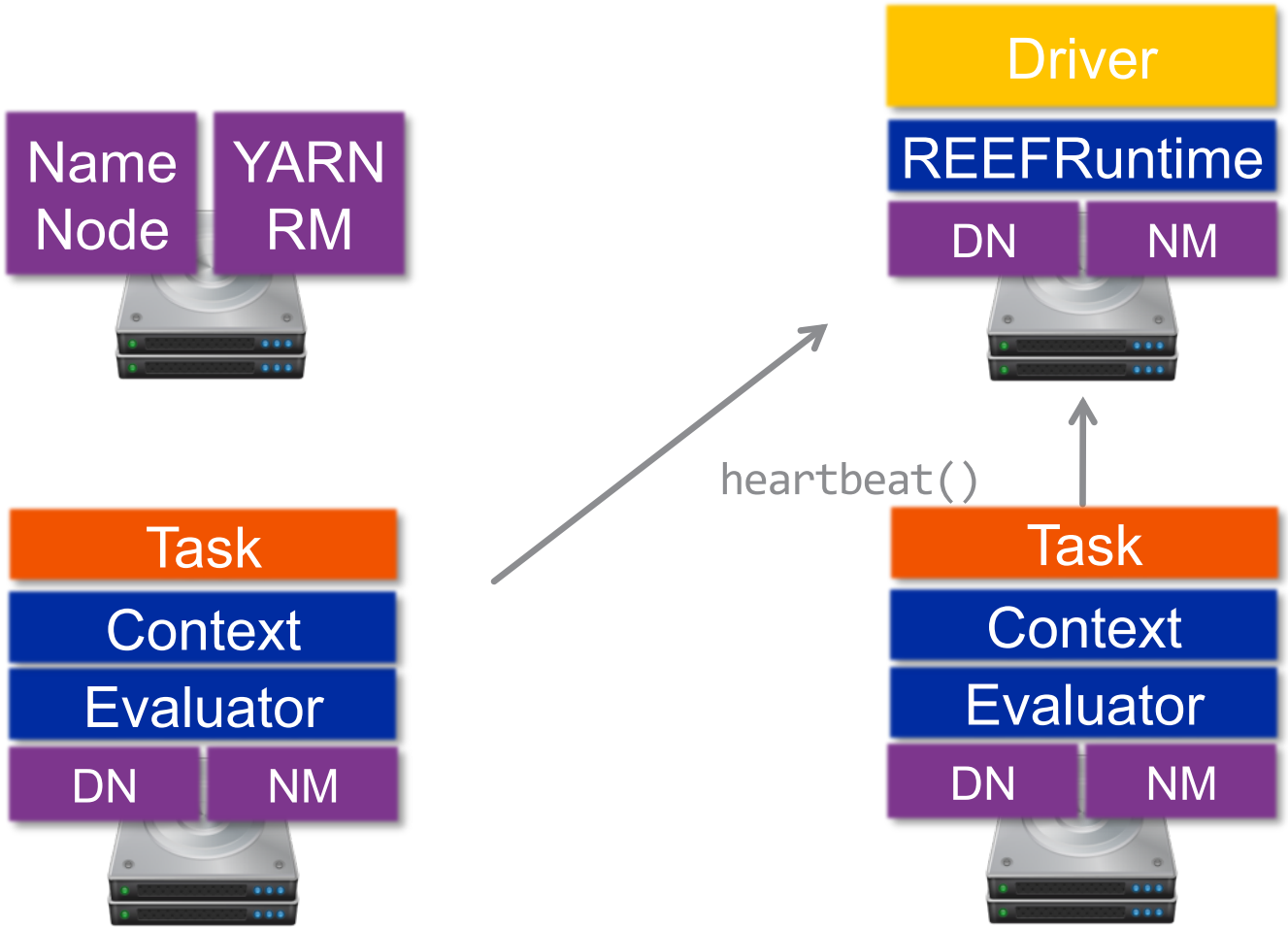
Context

Evaluator

DN | NM

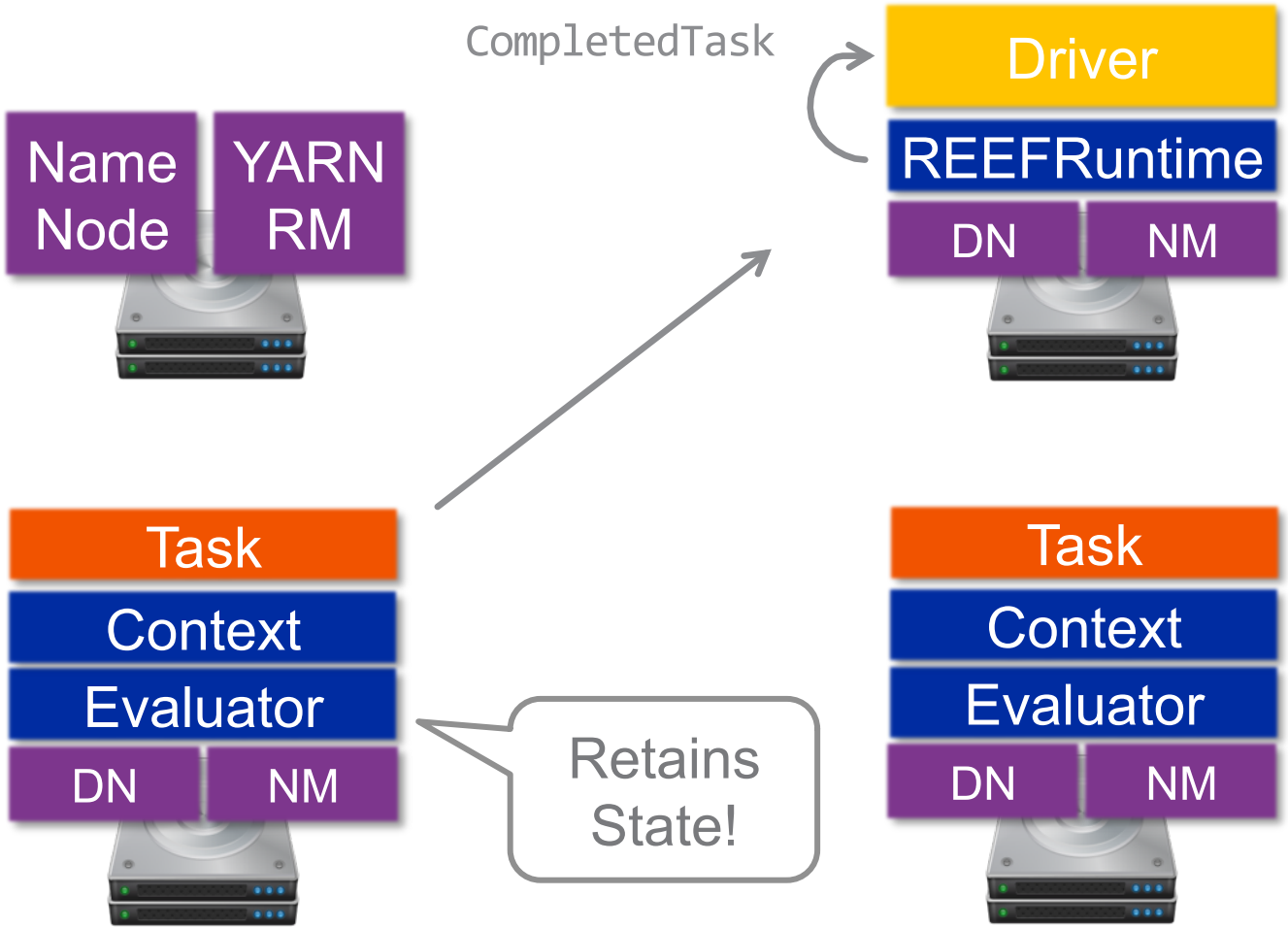
```
class ShellTask implements Task {

  private final String command;

  @Inject
  ShellTask(@Parameter(Command.class) String c) {
    this.command = c;
  }

  private String exec(final String command){
    ...
  }

  @Override
  public byte[] call(byte[] memento) {
    String s = exec(this.cmd);
    return s.getBytes();
  }

}
```

REEF Control Flow

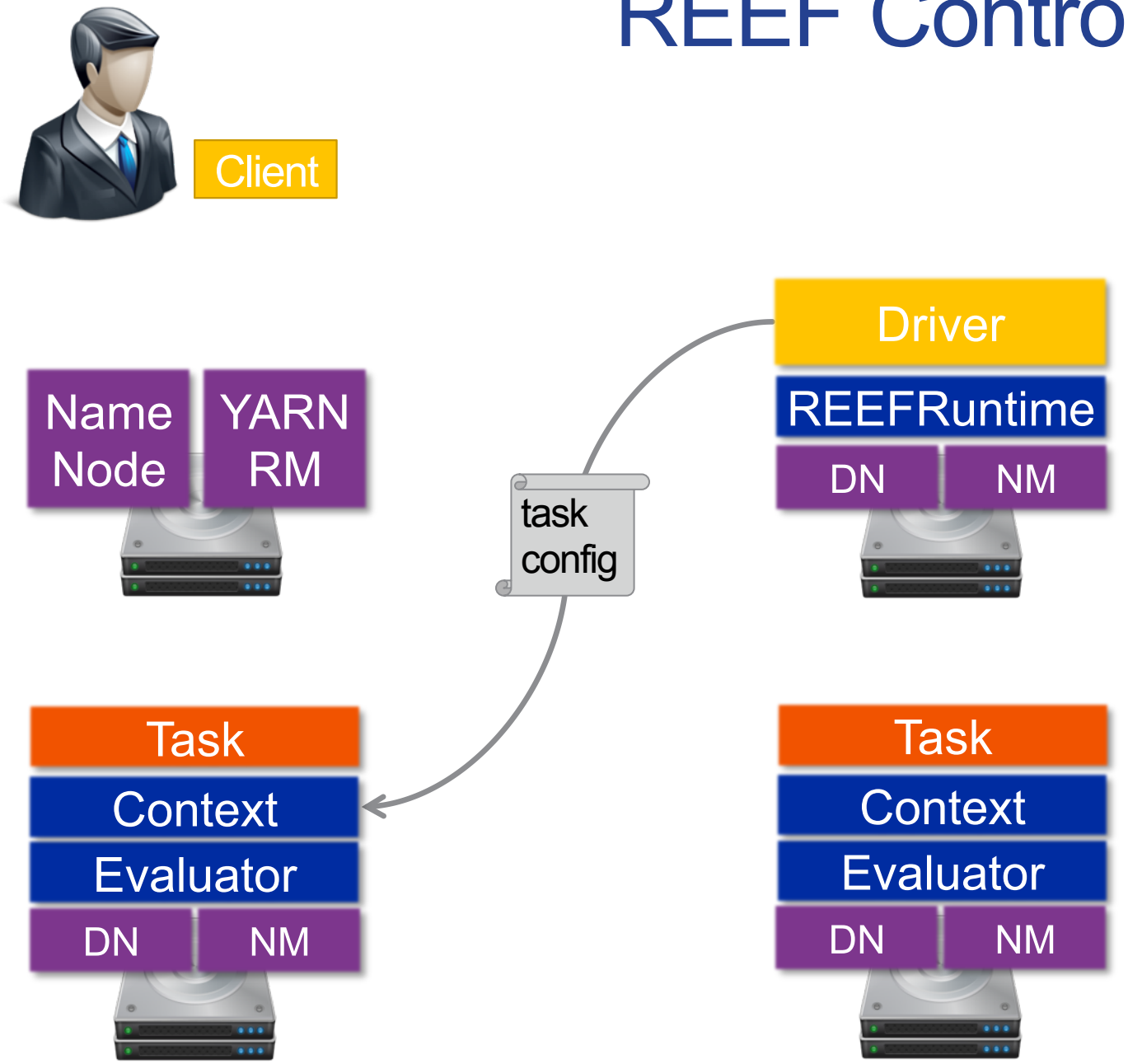


REEF Control Flow

CompletedTask

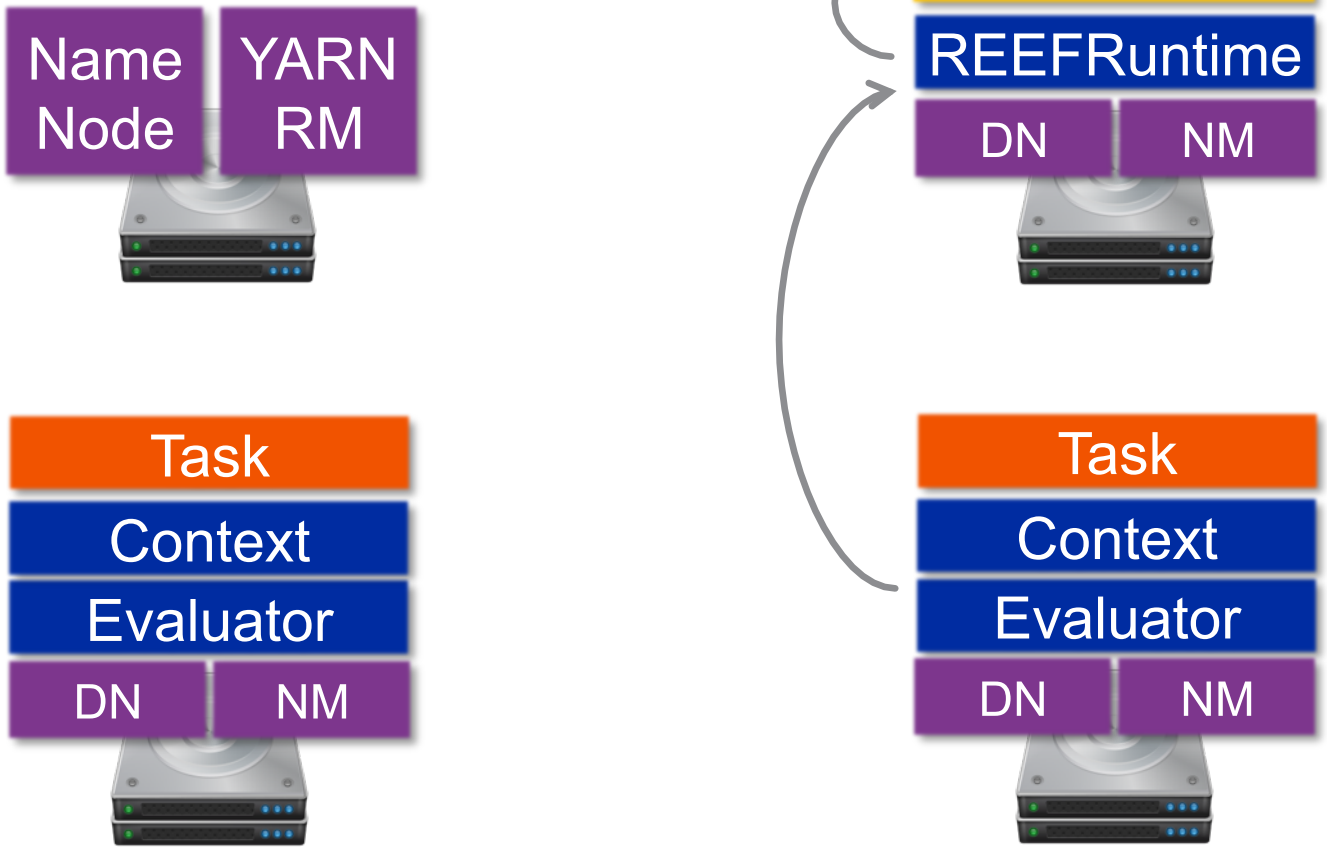


REEF Control Flow



REEF Control Flow

CompletedTask



REEF Control Flow

Context.
close()

Driver

CompletedEvaluator

REEFRuntime

DN

NM

Name
Node

YARN
RM

Task

Context

Evaluator

DN

NM

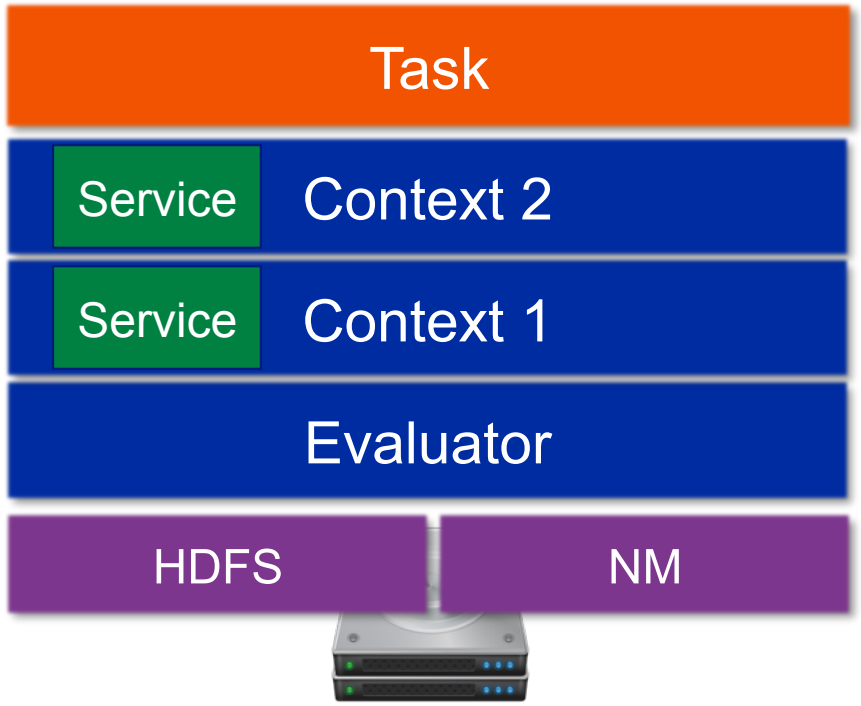
Context

Evaluator

DN

NM

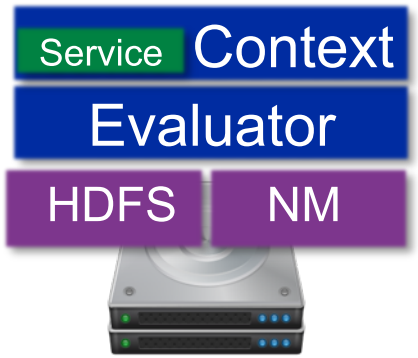
REEF Control Flow: Context



REEF Data Management Services

Evaluator Side

- Services live in the scope of the Context
- They outlive Tasks
- Tasks get them injected at construction time



Driver Side

- Services provide help with creating their Evaluator-Side Configuration
- Typically, via a declarative specification

REEF Data Management Services

Storage

Evaluator Side	Driver Side

• Two abstractions: Map and Spool
• Many different implementations
  • Local vs. Remote
  • Tiered Storage
  • Access Patterns
	• User declares the abstraction and (expected) access pattern
	• Provides configurations that bind the abstract interfaces to optimal implementations

REEF Data Management Services

Network

Evaluator Side	Driver Side

• Name-based Inter-Task Messaging
• Group communications
  • E.g., Reduce, Broadcast
	• Name Service
	  • Tracks the physical location of Tasks
	• Wire-up for group communications

REEF Data Management Services

State management

Evaluator Side	Driver Side

• Checkpoint channel that returns Memento
• Access to state when presented with a Memento
	• User select check pointing constraints
	  • Durability level: Memory, Disk, HDFS, WAS, …
	  • Lifetime of checkpoints
	• Service generates configurations satisfying these requirements

Simplify building REEF

Tang

Wake

Cross-language framework: Java and .NET

Tang 

- Configuration management using dependency injection
  - Bind an implementation to an interface
  - Bind a value to a configuration parameter
  - Automated checks *before* spinning up (remote) machines

- Tang properties
  - “Configuration” are just data
  - Configuration options can be set at most once
  - A large subset of Tang’s public API is commutative
  - Interfaces and configuration parameters are encouraged to specify defaults
  - Static analysis and documentation tools

Wake 

- Event-driven Framework
  - EventHandler / Observer / Observable but with static wire-up
  - Stages: synchronous and asynchronous
  - Thread management
  - Network and storage I/O
  - Remoting
    - Multi-codec support
    - Multiplexing communication between two end points
    - Message ordering guarantee

What Have Been Built atop REEF?

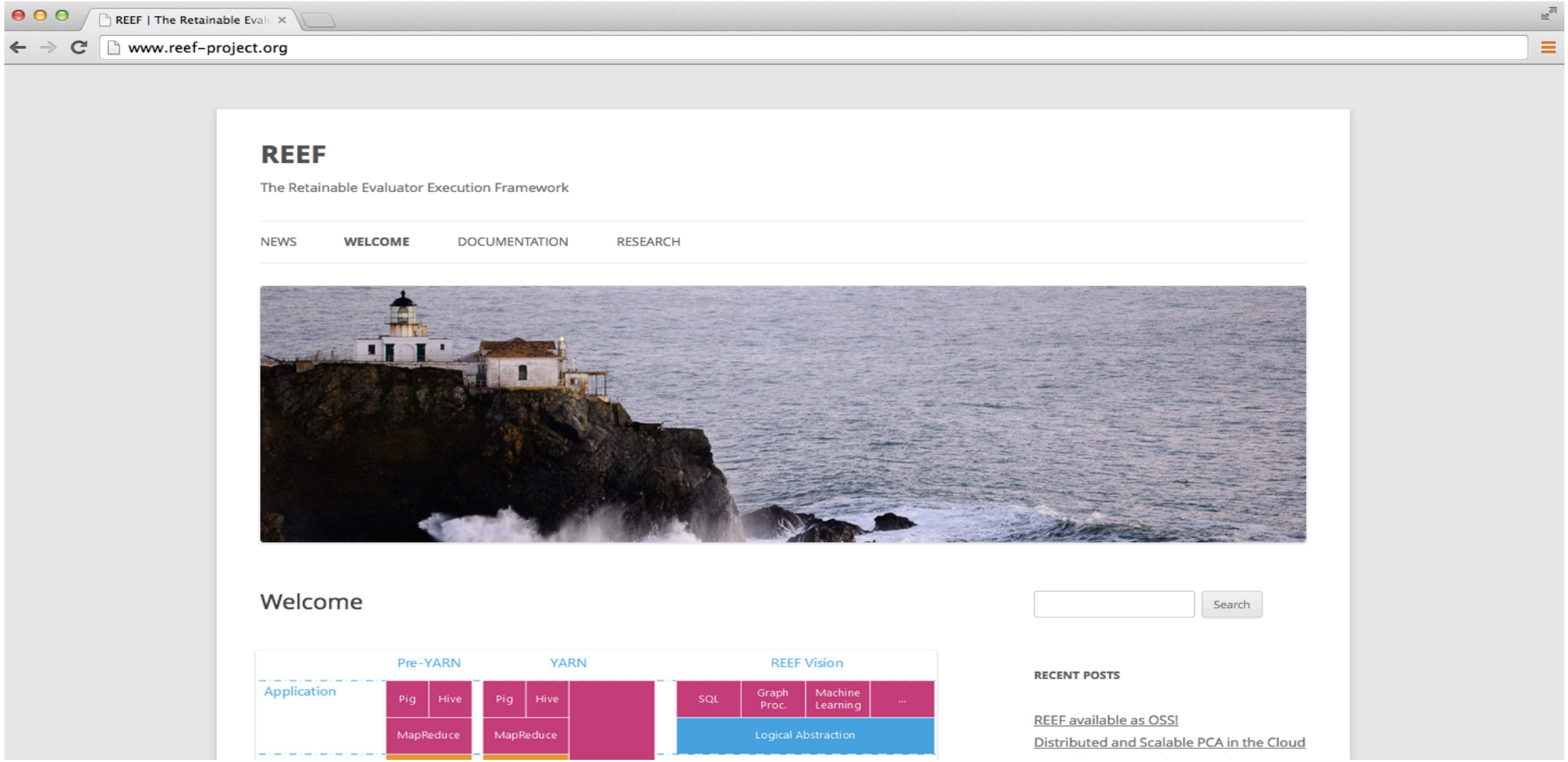
- Batch processing
  - DAG scheduling (a generalized version of MapReduce)
- “Fault-aware” Machine learning
- “Elastic” Graph processing

Current Status

- Ongoing work: data plane common building blocks and optimizers

- Development on top of REEF
  - A Microsoft product team
  - A Korean company

Open-sourced under the Apache License 2 in Jan. 2014 (**http://www.reef-project.org/**)



REEF Examples (Part I)

Yunseong Lee

REEF examples

- You can also see these tutorials in the REEF Github repo (https://github.com/Microsoft-CISL/REEF/wiki)
- HelloREEF
  - Basic version
  - On Yarn environment
  - With no Client
  - Http
- Retained Evaluators
- State Passing
- Group communication

HelloREEF

- Description
  - The most basic application
  - Execute a driver and allocate an Evaluator
  - Start Task and print a log message
- How to execute?

  $ cd $REEF_HOME/reef-examples

  $ mvn -PHelloREEF
- Where is the Result?

  $ cd REEF_LOCAL_RUNTIME/

HelloREEF

- The topics to be covered
  - The main components in REEF Applications
  - Job Client and the Launcher
  - Job Driver
  - Task

HelloREEF

- The main components in REEF Applications
  - **Client:** The client runs on the gateway host; it is usually started from the main() method in Java. Client starts the Driver on YARN. For simple tasks, like HelloREEF, we use a higher level class DriverLauncher that implements most of the generic Client functionality.
  - **Driver:** runs on the YARN container, and can request multiple Evaluators to run Tasks on. The driver controls the tasks and communicates with the job Client.
  - **Evaluator:** reusable environment to run Tasks on. It can be viewed as REEF abstraction around the YARN Container. Evaluator can retain the data shared across the Tasks that run on that evaluator.
  - **Task:** is a unit of work on REEF, submitted by job Driver to the Evaluators.

HelloREEF

- Job Client and the Launcher
  - HelloREEF.java
    - In the main() method, create either the local configuration
    - Configure the job driver
      - specify events the Driver should handle
      - which JARs have to be copied to the remote
    - supply runtimeConfiguration and driverConfiguration to the DriverLauncher
      - DriverLauncher implements the basic Client functionality

HelloREEF

- Job Driver
  - HelloDriver.java
    - Constructor
      - @Inject annotation
      - HelloDriver object is instantiated automatically by REEF
      - The framework also injects the constructor parameters
    - HelloREEF Driver responds to two events
      - ON_DRIVER_STARTED
      - ON_EVALUATOR_ALLOCATED
      - The handlers implement EventHandler interface, which has only one method, .onNext()

HelloREEF

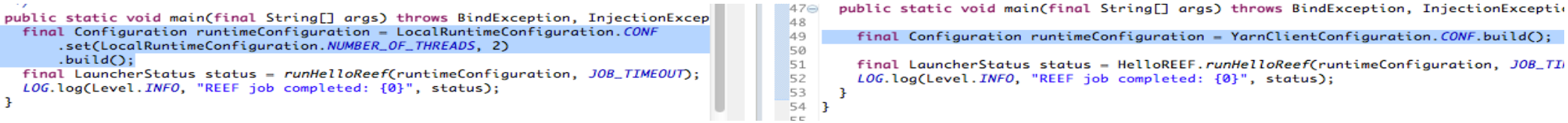
- Job Driver
  - HelloDriver.java
    - @Unit annotation
      - It tells TANG (dependency injection framework used in REEF) to instantiate HelloDriver and its inner classes as a single unit
    - StartHandler
      - Called when the driver started
      - requests one small instance of the Evaluator to run the Task on
    - EvaluatorAllocatedHandler
      - Called when an evaluator successfully allocated
      - submit a new Task to the corresponding Evaluator
      - Two kind of Configuration : Task and Context

HelloREEF

- Task
  - HelloTask.java
    - Main Task method is call()
    - Task *must* have a constructor annotated with @Inject
    - Task can accept and return a value, encoded as byte array
      - .call() method argument (memento) usually serves as a restart point for tasks that can be suspended, restarted, and/or moved from one Evaluator to another
      - Return value (also a byte array), is delivered by REEF to the Driver as part of the CompletedTask event.
      - To pass parameters to the Task, to add them to the Task or Context configuration is preferred to using memento

HelloREEFYarn

- Run HelloREEF on YARN runtime environment
  - Use the most part of HelloREEF except runtime configuration
  - In c.m.r.e.h.HelloReefYarn.java
    - LocalRuntimeConfiguration -> YarnClientConfiguration



```
public static void main(final String[] args) throws BindException, InjectionExcep
  final Configuration runtimeConfiguration = LocalRuntimeConfiguration.CONF
      .set(LocalRuntimeConfiguration.NUMBER_OF_THREADS, 2)
      .build();
  final LauncherStatus status = runHelloReef(runtimeConfiguration, JOB_TIMEOUT);
  LOG.log(Level.INFO, "REEF job completed: {0}", status);
}
```

```
47  public static void main(final String[] args) throws BindException, InjectionExceptio
48
49    final Configuration runtimeConfiguration = YarnClientConfiguration.CONF.build();
50
51    final LauncherStatus status = HelloREEF.runHelloReef(runtimeConfiguration, JOB_TI
52    LOG.log(Level.INFO, "REEF job completed: {0}", status);
53  }
54 }
```

HelloREEFNoClient

- A main() for running hello REEF without a persistent client connection

```
public static LauncherStatus runHelloReef(final Configuration runtimeConf, final in
    throws BindException, InjectionException {
  final Configuration driverConf = getDriverConfiguration();
  return DriverLauncher.getLauncher(runtimeConf).run(driverConf, timeOut);
}

/**
 * Start Hello REEF job. Runs method runHelloReef().
 *
 * @param args command line parameters.
 * @throws BindException      configuration error.
 * @throws InjectionException configuration error.
 */
public static void main(final String[] args) throws BindException, InjectionExcepti
  final Configuration runtimeConfiguration = LocalRuntimeConfiguration.CONF
```

```
37
38  public static void runHelloReefWithoutClient(
39      final Configuration runtimeConf) throws InjectionException {
40
41    final REEF reef = Tang.Factory.getTang().newInjector(runtimeConf).getInstance(REEFImplementation.class);
42
43    final Configuration driverConf =
44        EnvironmentUtils.addClasspath(DriverConfiguration.CONF, DriverConfiguration.GLOBAL_LIBRARIES)
45            .set(DriverConfiguration.DRIVER_IDENTIFIER, "HelloREEF")
46            .set(DriverConfiguration.ON_DRIVER_STARTED, HelloDriver.StartHandler.class)
47            .set(DriverConfiguration.ON_EVALUATOR_ALLOCATED, HelloDriver.EvaluatorAllocatedHandler.class)
48            .build();
49
50    reef.submit(driverConf);
51  }
52
```

```
/**
 * Instantiate a launcher for the given Configuration.
 *
 * @param runtimeConfiguration the resourcemanager configuration to be used
 * @return a DriverLauncher based on the given resourcemanager configuration
 * @throws BindException      on configuration errors
 * @throws InjectionException on configuration errors
 */
public static DriverLauncher getLauncher(
    final Configuration runtimeConfiguration) throws BindException, InjectionExcep

  final Configuration clientConfiguration = ClientConfiguration.CONF
      .set(ClientConfiguration.ON_JOB_RUNNING, RunningJobHandler.class)
      .set(ClientConfiguration.ON_JOB_COMPLETED, CompletedJobHandler.class)
      .set(ClientConfiguration.ON_JOB_FAILED, FailedJobHandler.class)
      .set(ClientConfiguration.ON_RUNTIME_ERROR, RuntimeErrorHandler.class)
      .build();

  return Tang.Factory.getTang()
      .newInjector(runtimeConfiguration, clientConfiguration)
      .getInstance(DriverLauncher.class);
}
```

```
public LauncherStatus run(final Configuration driverConfig, final long timeOut) {
  final long endTime = System.currentTimeMillis() + timeOut;
  this.reef.submit(driverConfig);
  synchronized (this) {
    while (!this.status.isDone()) {
      try {
        final long waitTime = endTime - System.currentTimeMillis();
        if (waitTime <= 0) {
          break;
        }
        LOG.log(Level.FINE, "Wait for {0} milliSeconds", waitTime);
        this.wait(waitTime);
      } catch (final InterruptedException ex) {
        LOG.log(Level.FINE, "Interrupted: {0}", ex);
      }
    }
    if (System.currentTimeMillis() >= endTime) {
      LOG.log(Level.WARNING, "The Job timed out.");
      this.status = LauncherStatus.FORCE_CLOSED;
    }
  }

  this.reef.close();
  return this.status;
}
```

HelloREEFHttp

- REEF supports Handling Http requests
- This example implements *DistributedShell* using Http requests
- I apologize I had no time to prepare this example

RetainedEvaluators

- Description
  - Retainability : One of the key features of REEF(Retainable Evaluator Execution Framework)
  - Similar to *DistributedShell* example of Hadoop
  - Re-use allocated evaluators more than once
- How to execute?
  - mvn -PRetainedEval
  - After loading done, you can type shell commands
- Result?
  - It used to work, but it seems not working currently

RetainedEvaluators

- The topics to be covered
  - How to inject configuration via TANG
  - Parsing the command line args
  - How to reuse evaluators
  - Communication between Driver and Evaluator
  - Communication between Driver and Client

RetainedEvaluators

- How to inject configuration via TANG
  - Launch.java
    - NamedParameters

```
/**
 * Command line parameter: a command to run. e.g. "echo Hello REEF"
 */
@NamedParameter(doc = "The shell command", short_name = "cmd", default_value = "*INTERACTIVE*")
public static final class Command implements Name<String> {
}
```

    - Parse command line

```
private static Configuration parseCommandLine(final String[] args)
    throws BindException, IOException {
  final JavaConfigurationBuilder confBuilder = Tang.Factory.getTang().newConfigurationBuilder();
  final CommandLine cl = new CommandLine(confBuilder);
  cl.registerShortNameOfClass(Local.class);
  cl.registerShortNameOfClass(Command.class);
  cl.registerShortNameOfClass(NumRuns.class);
  cl.registerShortNameOfClass(NumEval.class);
  cl.processCommandLine(args);
  return confBuilder.build();
}
```

RetainedEvaluators

- How to inject configuration via TANG
  - Launch.java
    - Client Configuration
      - getClientConfiguration()
      - merge CommandLineConfig + RuntimeConfig + ClientConfig
    - run the application
      - run()
      - Client launch the driver via client.submit()

```
public static String run(final Configuration config) throws InjectionException {
  final Injector injector = Tang.Factory.getTang().newInjector(config);
  final JobClient client = injector.getInstance(JobClient.class);
  client.submit();
  return client.waitForCompletion();
}
```

RetainedEvaluators

- Job Client launches driver and Communicate with the driver with an interaction with user
  - JobClient.java
    - Takes the parameters passed via TANG in Launch
    - Build Driver Configuration (same as HelloREEF)
    - Commands are submitted by submitTask() method
      - Later we will see where the Driver receives this event.

RetainedEvaluators

- Shell Task executes the shell command given by user
  - ShellTask.java

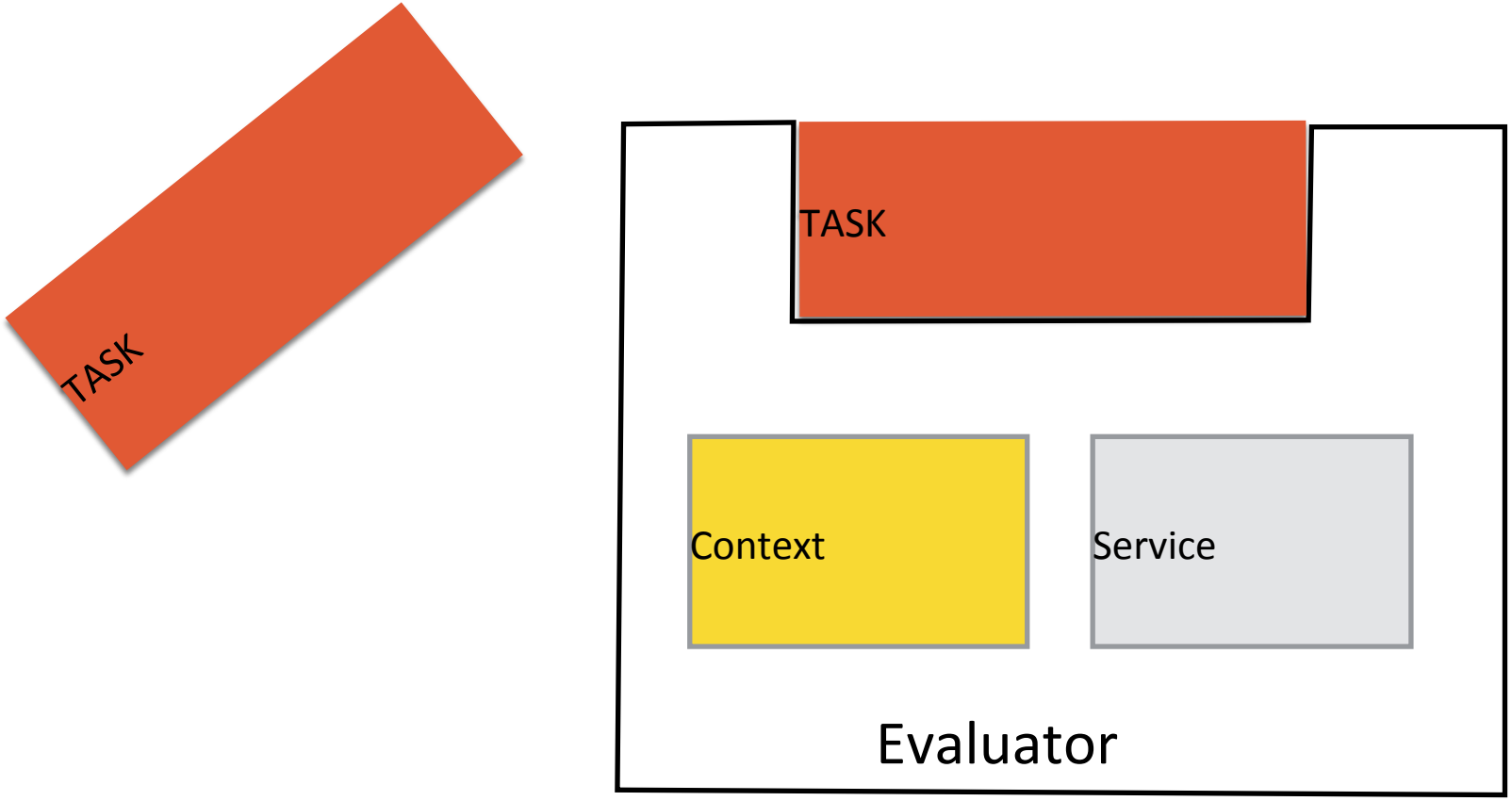
- Driver retains and reuses the evaluator
  - JobDriver.java

REEF Examples (Part II)

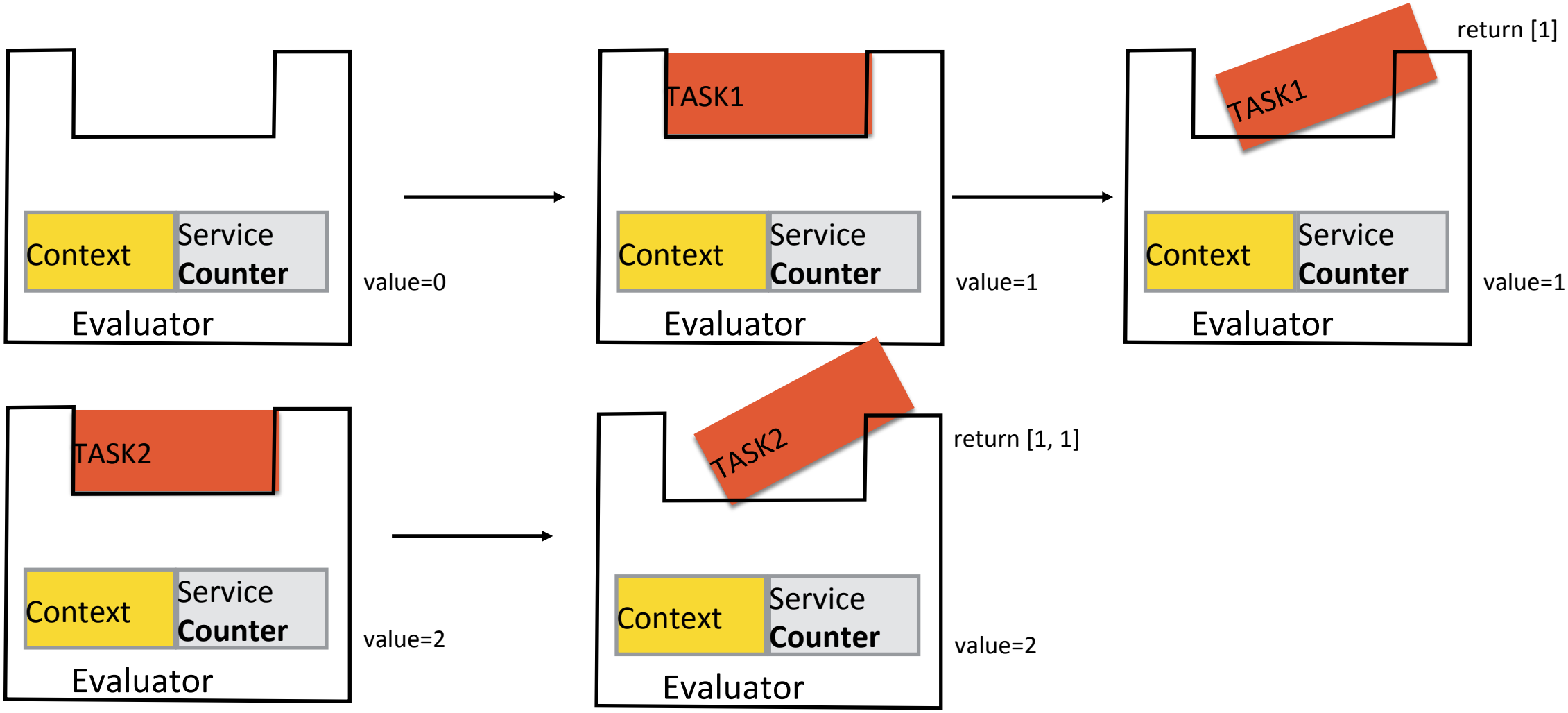
Taegeon Um

StatePassing

- Pass a state to another Task via Service



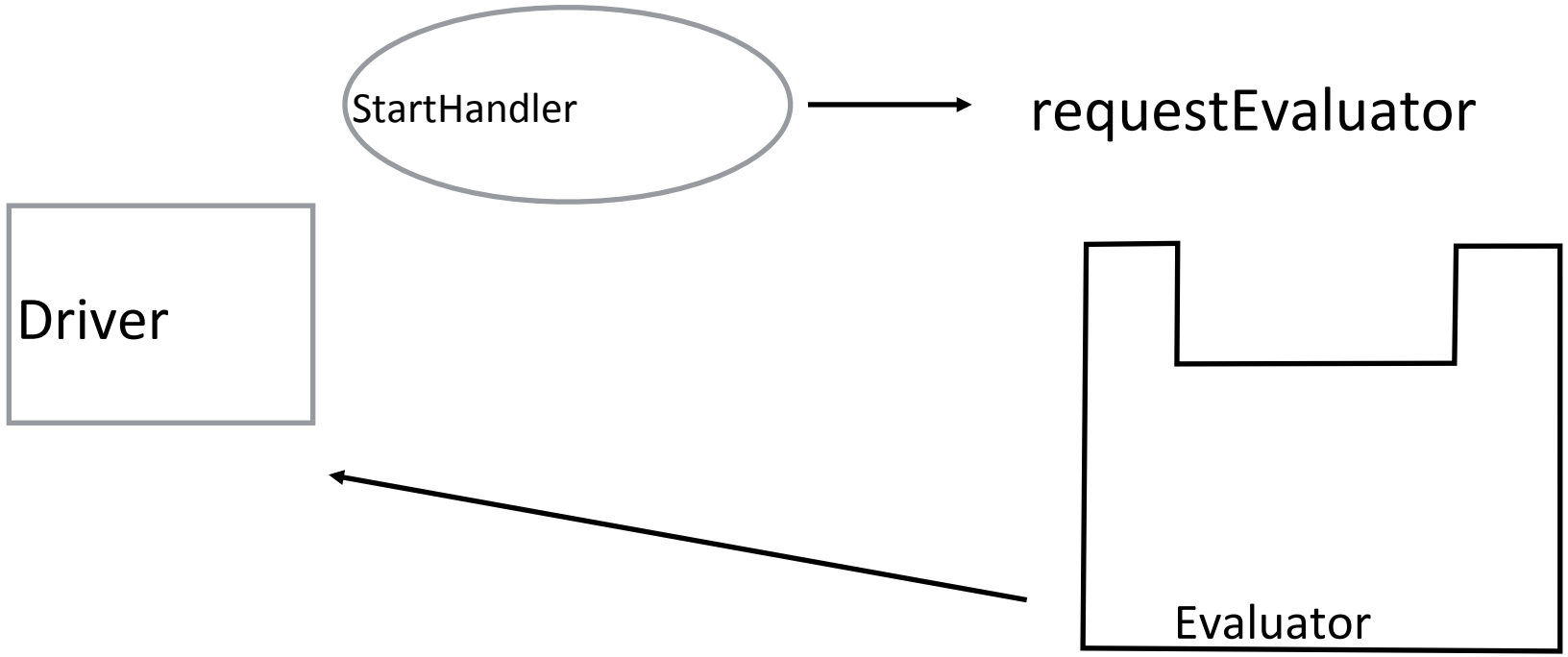
StatePassing



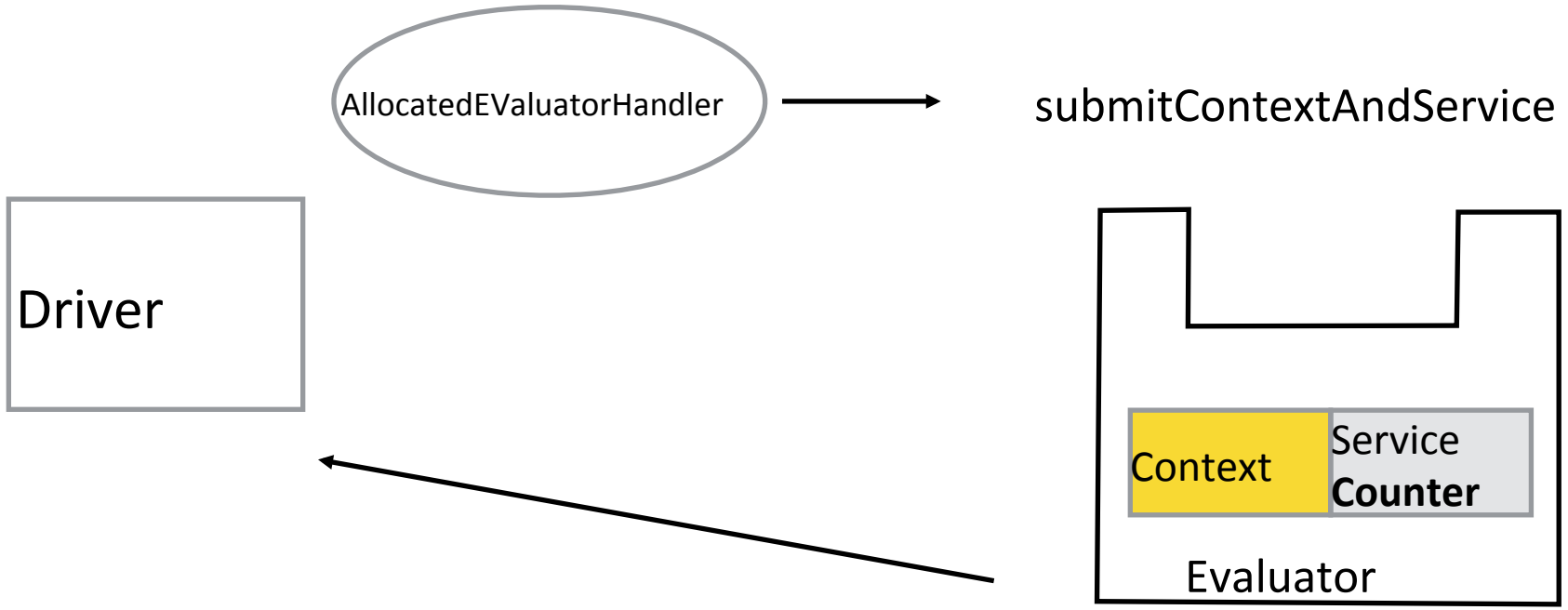
Flow

- Launch driver
- StartHandler - request 1 Evaluator
- AllocatedEvaluator - submit context and **service(Counter)**
- ActiveContextHandler - submit Task
- Run task - increment the count and return bytes array
- CompletedTaskHandler - submit Task again

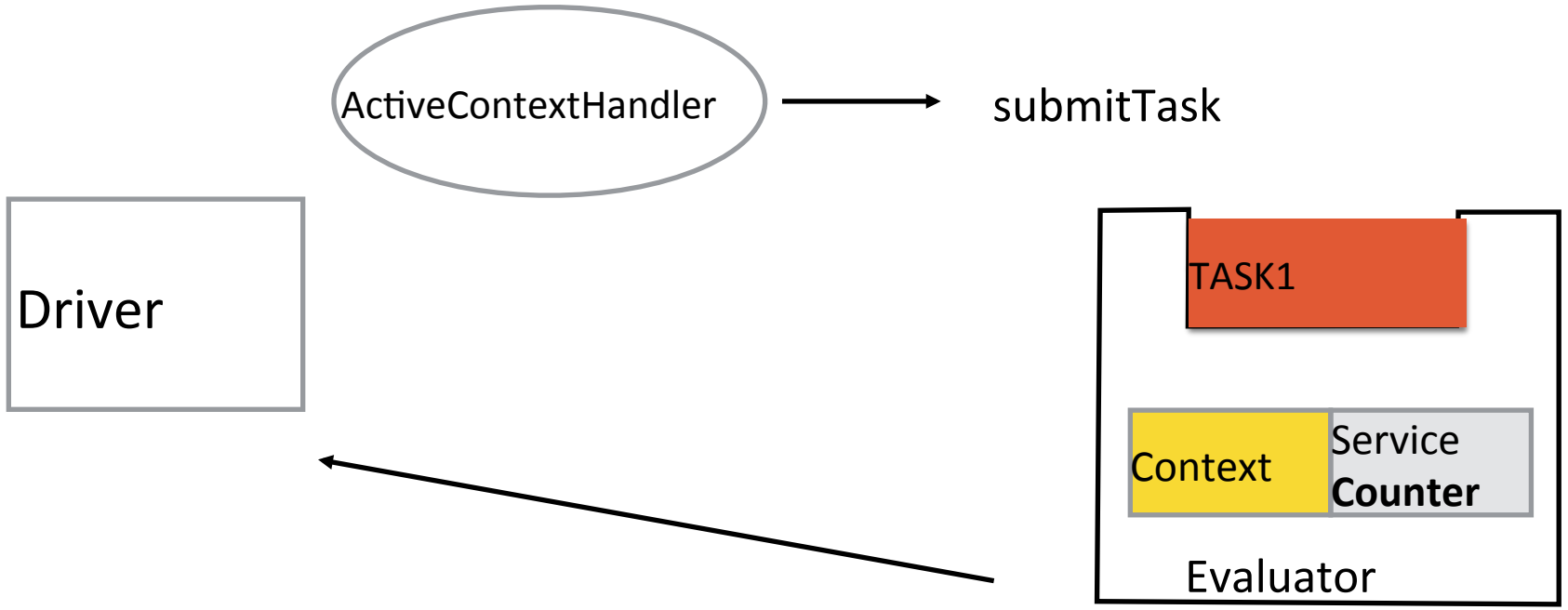
Flow



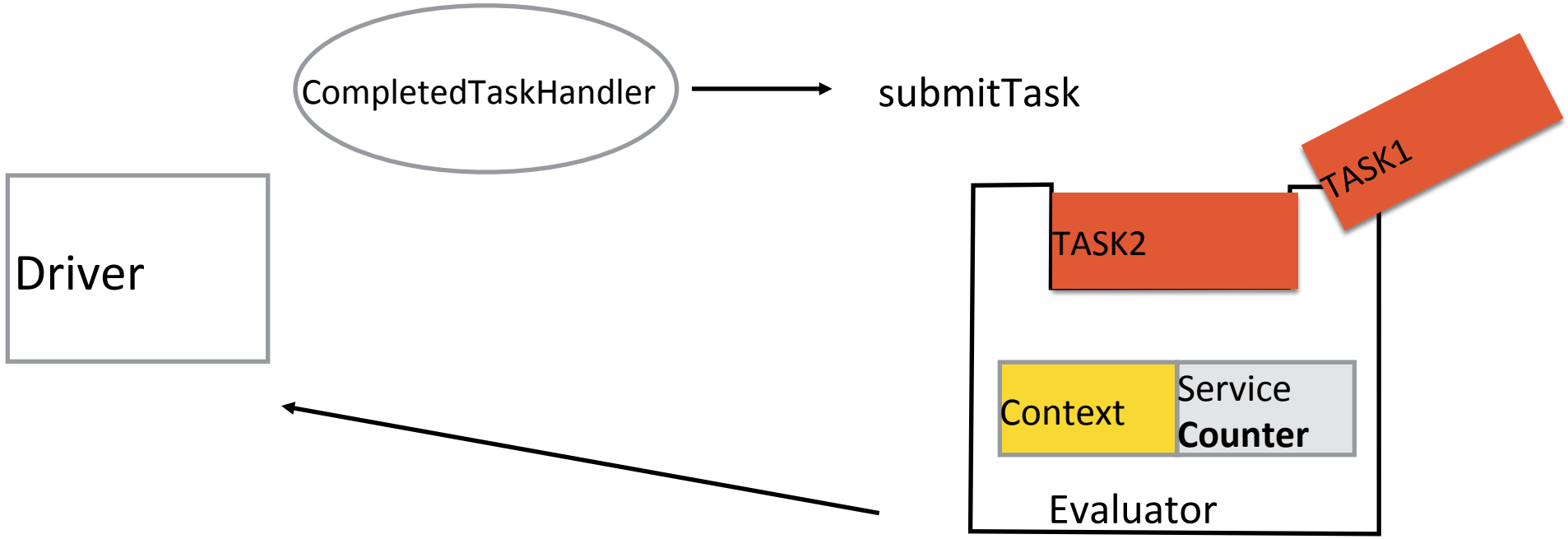
Flow


AllocatedEValuatorHandler
submitContextAndService
Driver
Context
Service
Counter
Evaluator

Flow

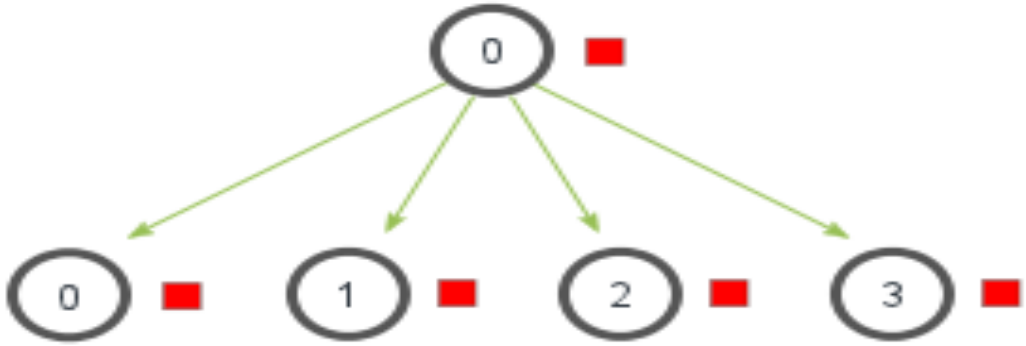

ActiveContextHandler
submitTask
Driver
TASK1
Context
Service
Counter
Evaluator

Flow

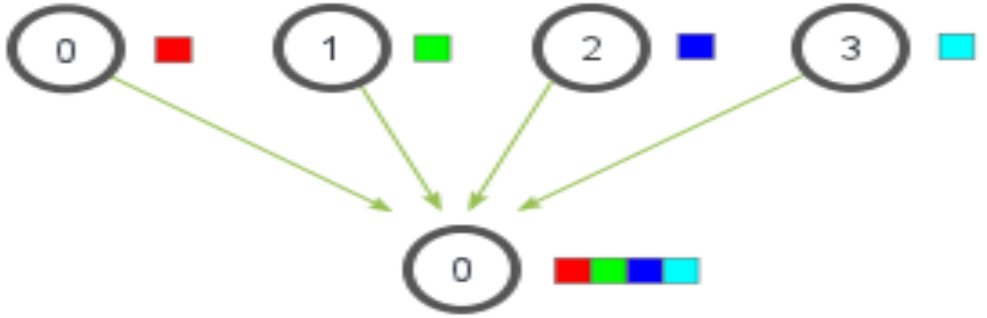

CompletedTaskHandler
submitTask
Driver
TASK1
TASK2
Context
Service
Counter
Evaluator

GroupCommunication

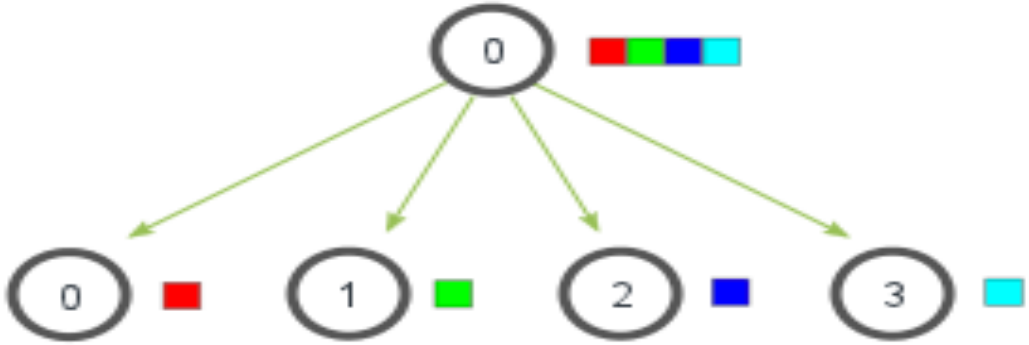
MPI_Bcast



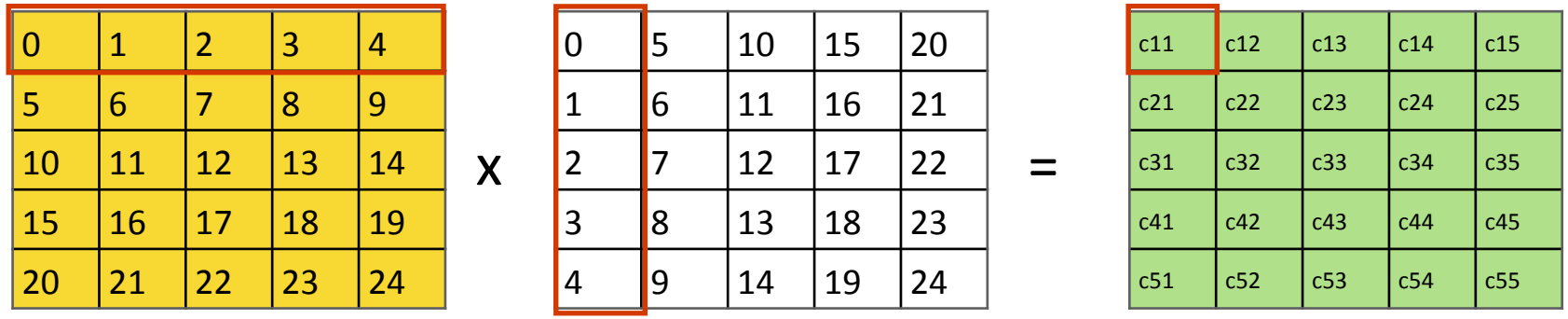
MPI_Gather



MPI_Scatter



GroupComm: MatMult



GroupComm: Scatter

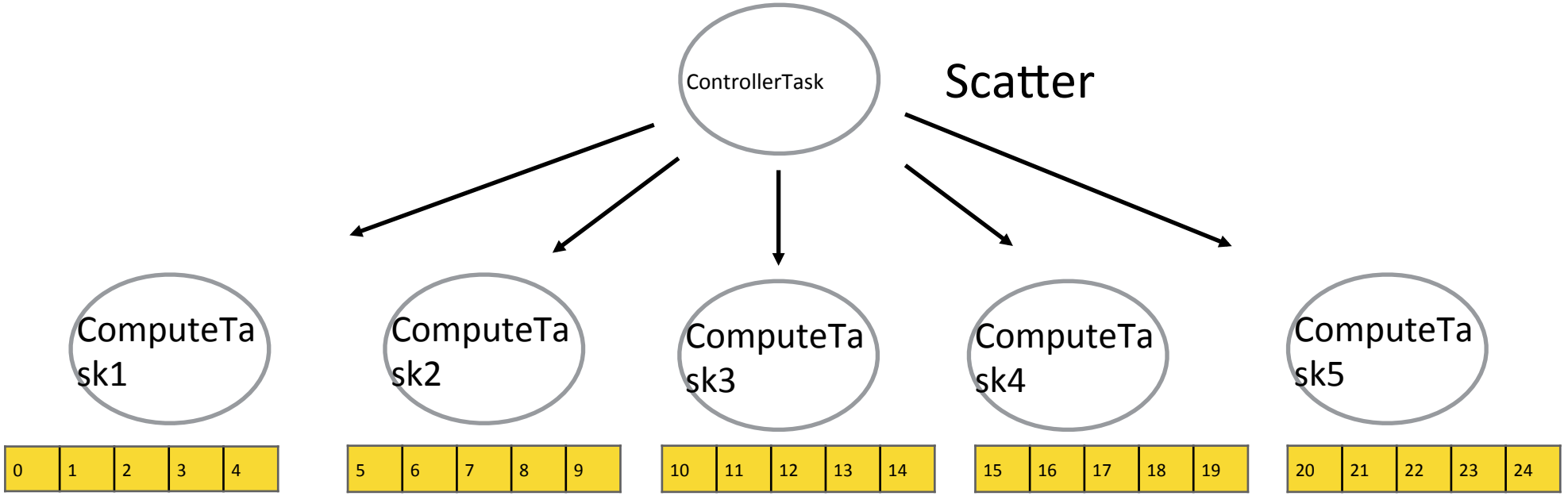
0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

x

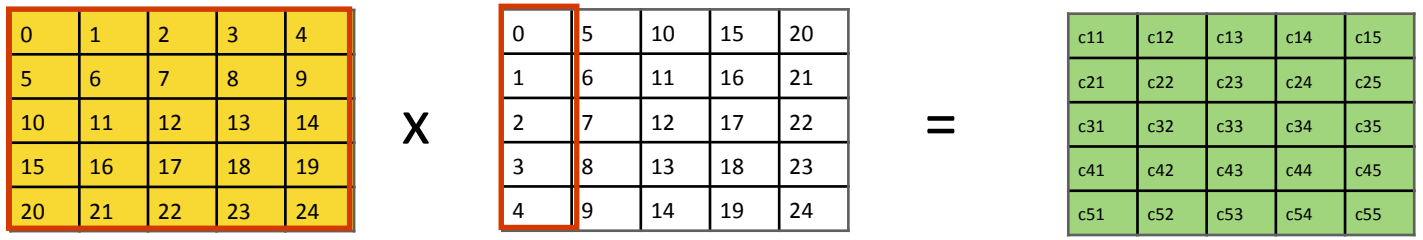
0	5	10	15	20
1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24

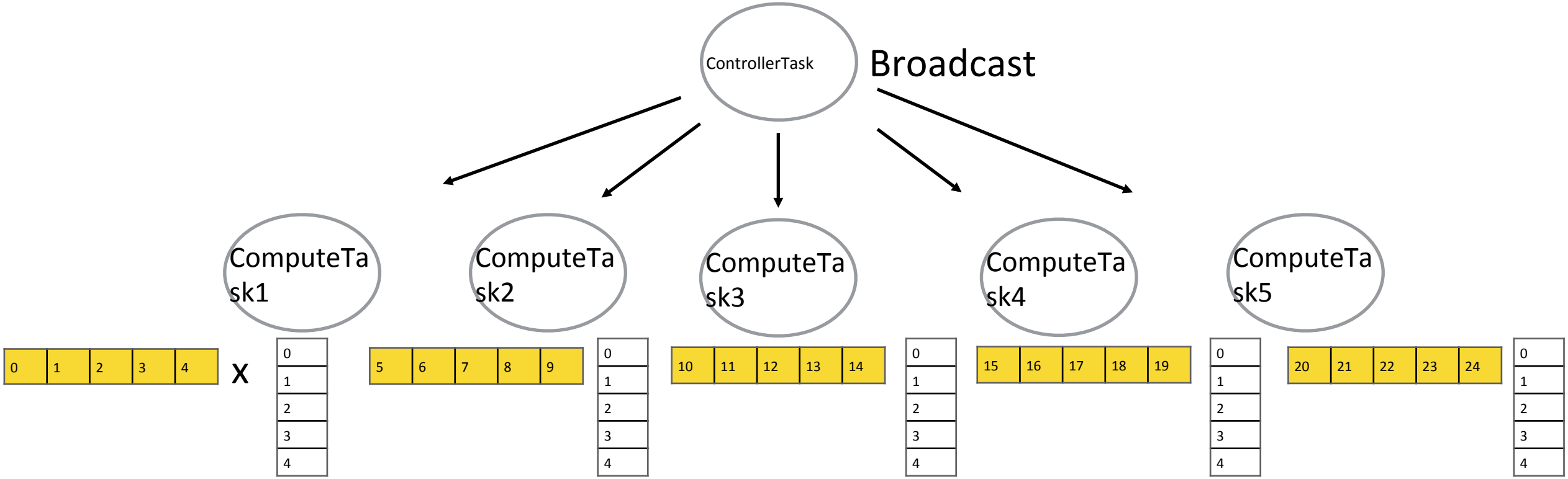
=

c11	c12	c13	c14	c15
c21	c22	c23	c24	c25
c31	c32	c33	c34	c35
c41	c42	c43	c44	c45
c51	c52	c53	c54	c55

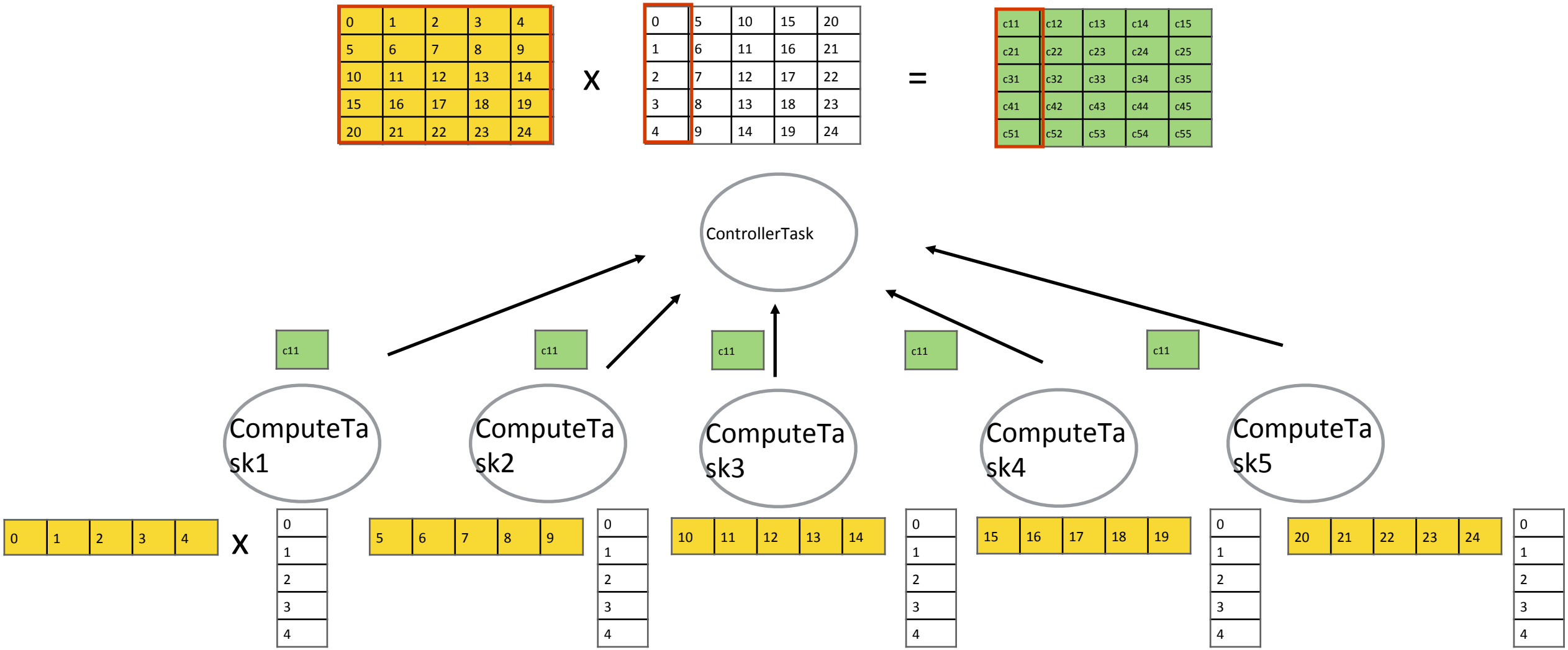


GroupComm: Broadcast





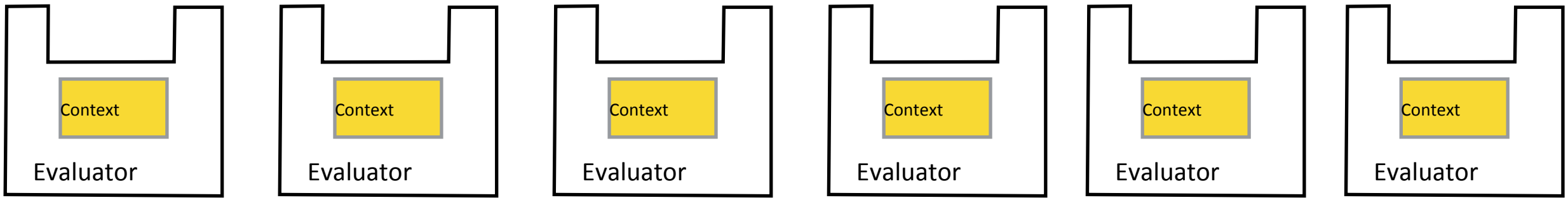
GroupComm: Reduce



Flow

- Launch **Driver**
- **StartEventHandler** - Request 6 Evaluators
- **AllocatedEvaluatorHandler** - submitContexts
- **ActiveContextHandler** - accumulate contexts
- Call taskSubmitter.onNext via contextAccumulator
- Submit computeTasks
- **RunningTaskHandler** - submit controller task after all the computeTasks is running
- ControllerTask - scatter and broadcast the matrix and reduce the results
- ComputeTask - receive the matrix and calculate the matrix multiplication and sends the results

Implementaiton

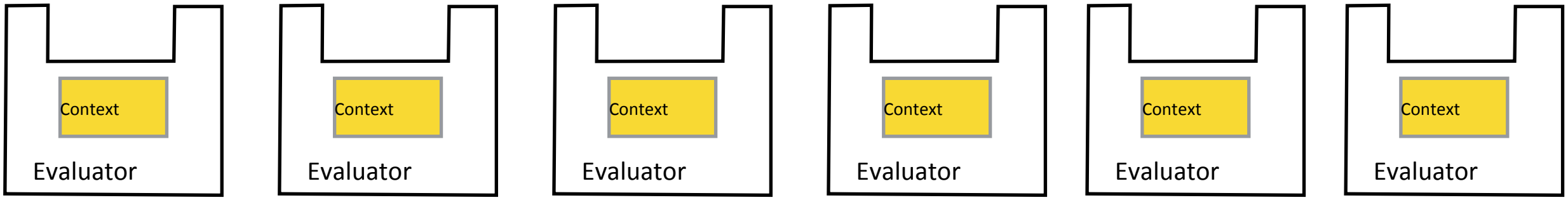


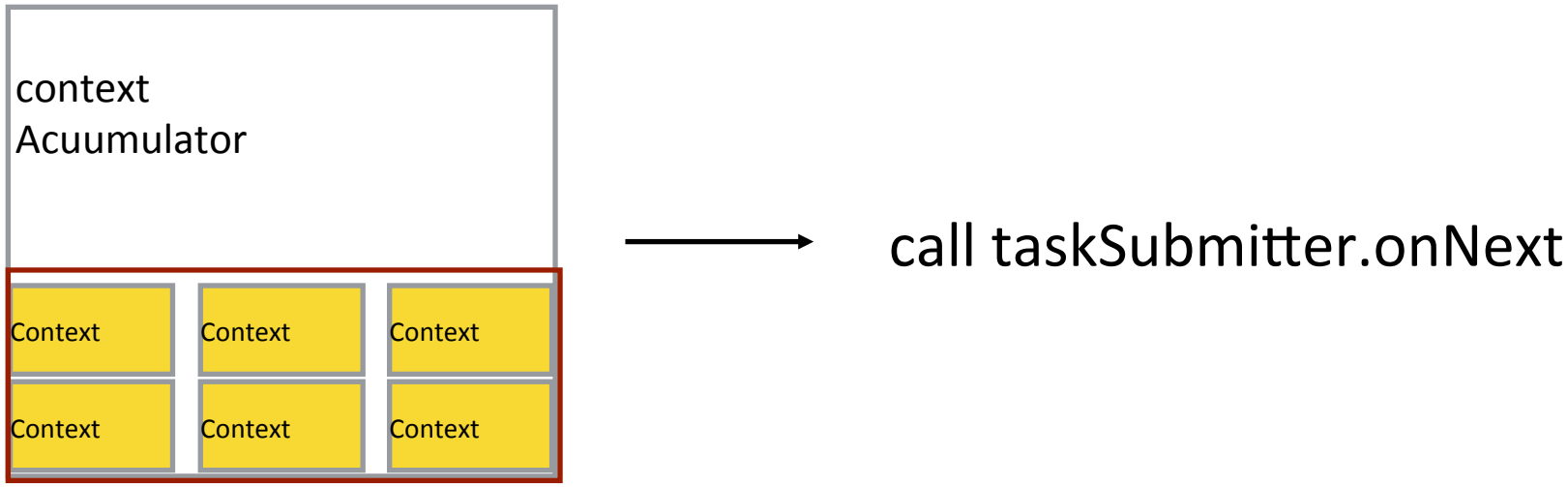

```
ActiveContextHandler implements EventHandler<ActiveContext> {
  @Override
  public void onNext(final ActiveContext activeContext) {
   contextAccumulator.onNext(activeContext);
  }
}
```

BlockingEventHandler

```
BlockingEventHandler<ActiveContext> contextAccumulator;
    this.taskSubmitter = new TaskSubmitter(this.computeTasks, nameServicePort);
    this.contextAccumulator = new BlockingEventHandler<>(
        this.computeTasks + this.controllerTasks, this.taskSubmitter);
```

BlockingEventHandler





TaskSubmitter

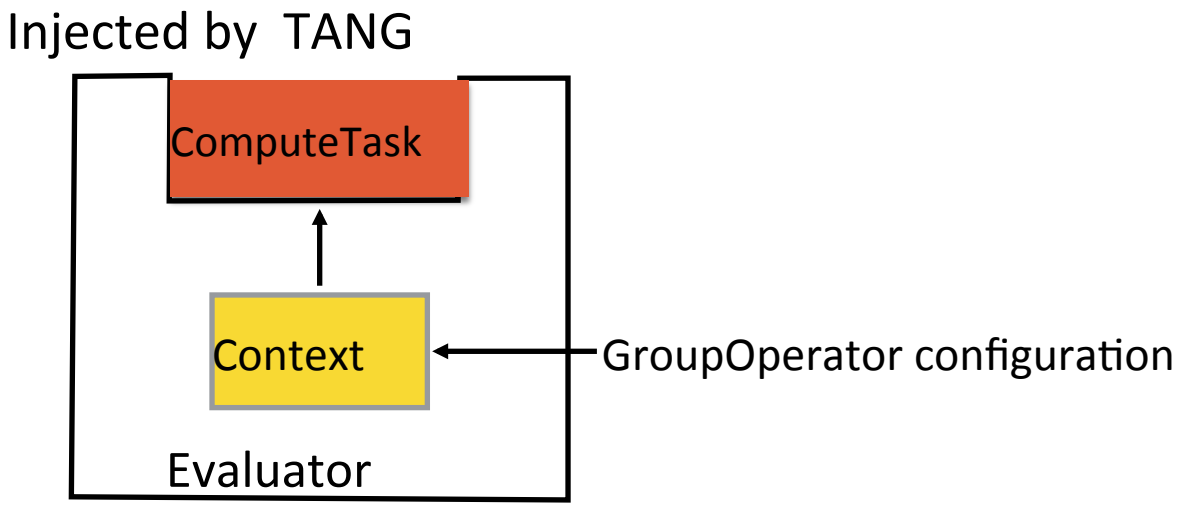
- Configure and submit controllerTask and computeTasks
- Configure GroupOperators
- Submit 5 computeTasks

GroupOperators

```
operators = new GroupOperators(VectorCodec.class, VectorConcat.class,
    nameServiceAddr, nameServicePort, id2port);

// Fluent syntax for adding the group communication
// operators that are needed for the job
operators.addScatter().setSender(controllerId)
    .setReceivers(computeTaskIds);

operators.addBroadCast().setSender(controllerId)
    .setReceivers(computeTaskIds);

// Each operator can override the default setting
// Here the reduce function though same put it in
// to illustrate
operators.addReduce().setReceiver(controllerId)
    .setSenders(computeTaskIds).setRedFuncClass(VectorConcat.class);
```

Configuration

```
private Configuration getComputeTaskConfig(final ComparableIdentifier compTaskId) {
  try {
   final JavaConfigurationBuilder b = Tang.Factory.getTang().newConfigurationBuilder();
    b.addConfiguration(operators.getConfig(compTaskId));
    b.addConfiguration(TaskConfiguration.CONF
        .set(TaskConfiguration.IDENTIFIER, compTaskId.toString())
        .set(TaskConfiguration.TASK, ComputeTask.class)
        .build());
    return b.build();
  } catch (BindException e) {
    logger.log(
        Level.SEVERE,
        "BindException while creating GroupCommunication operator configurations",
        e.getCause());
    throw new RuntimeException(e);
  }
}
```

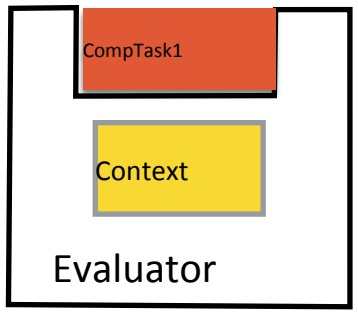

Injected by TANG
ComputeTask
Context
GroupOperator configuration
Evaluator

RunningTaskHandler

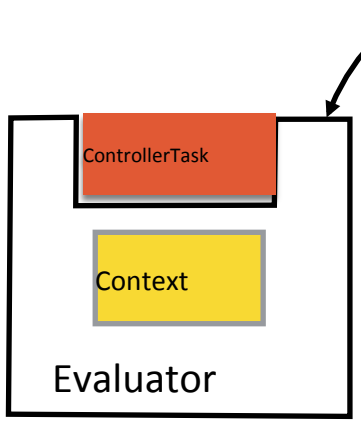
```
/**
 * Task is running. Track the compute tasks that are running.
 * Once all compute tasks are running submitTask the ControllerTask.
 */
final class RunningTaskHandler implements EventHandler<RunningTask> {
  @Override
  public final void onNext(final RunningTask task) {
   if (compTasksRunning.incrementAndGet() == computeTasks) {
      // All compute tasks are running - launch controller task
      taskSubmitter.submitControlTask();
    }
  }
}
```

GroupOperator Injection

Injection by TANG




```
@Inject
public ComputeTask(Scatter.Receiver<Vector> scatterReceiver,
                   Broadcast.Receiver<Vector> broadcastReceiver,
                   Reduce.Sender<Vector> reduceSender) {
    super();
    this.scatterReceiver = scatterReceiver;
    this.broadcastReceiver = broadcastReceiver;
    this.reduceSender = reduceSender;
}
```




```
@Inject
public ControllerTask(Scatter.Sender<Vector> scatterSender,
                      Broadcast.Sender<Vector> broadcastSender,
                      Reduce.Receiver<Vector> reduceReceiver) {
    super();
    this.scatterSender = scatterSender;
    this.broadcastSender = broadcastSender;
    this.reduceReceiver = reduceReceiver;
```