Hadoop
YARN - Under the Hood

Sharad Agarwal
sharad@apache.org

About me

@sharad_ag

Hadoop Committer & Head, Platforms @InMobi

Bangalore India · http://in.linkedin.com/in/sharadagarwal

Bangalore Hadoop Meetups

Change photo

Organizer:
Sharad Agarwal

Bangalore, India
Founded Nov 2, 2011

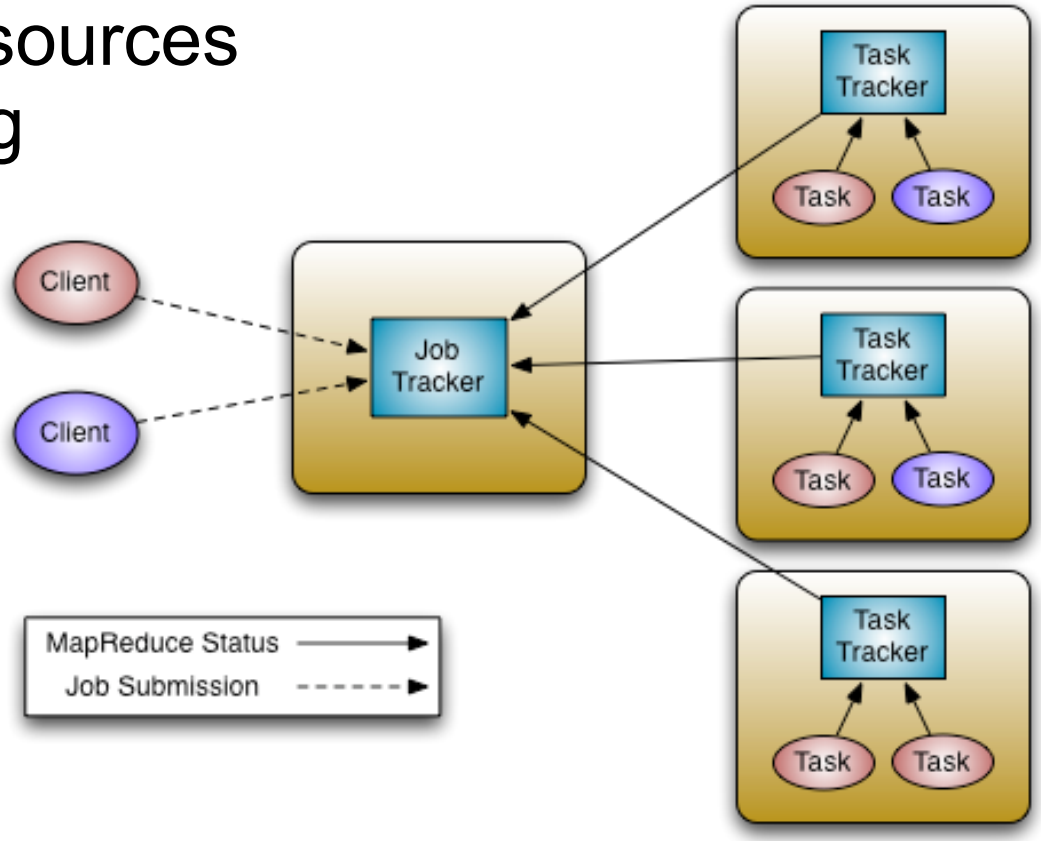
Recap: Hadoop 1.0 Map-Reduce

JobTracker

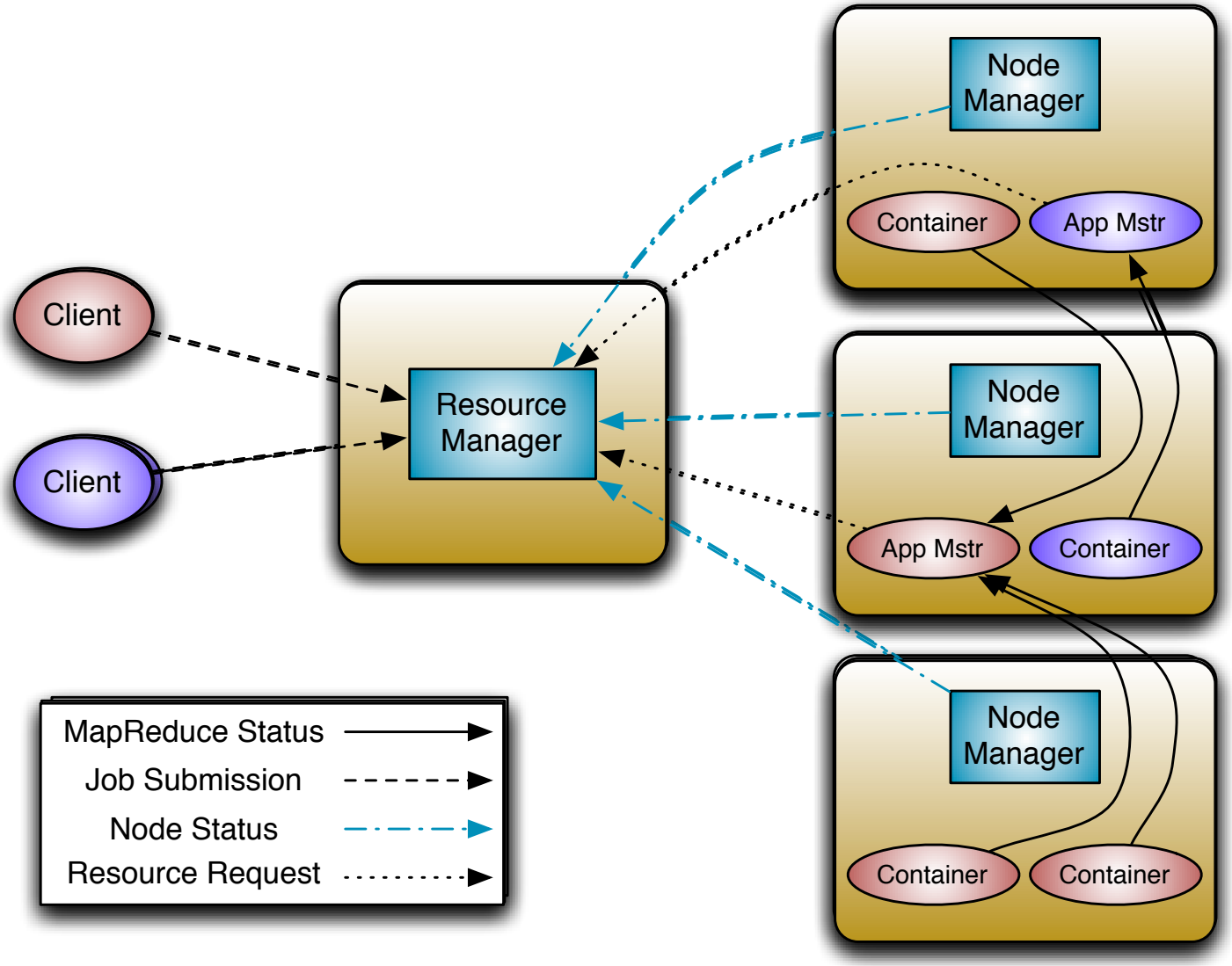
- Manages cluster resources and job scheduling

TaskTracker

- Per-node agent
- Manage tasks



YARN Architecture



What the new Architecture gets us?

Scale
Compute Platform

Scale for a compute platform

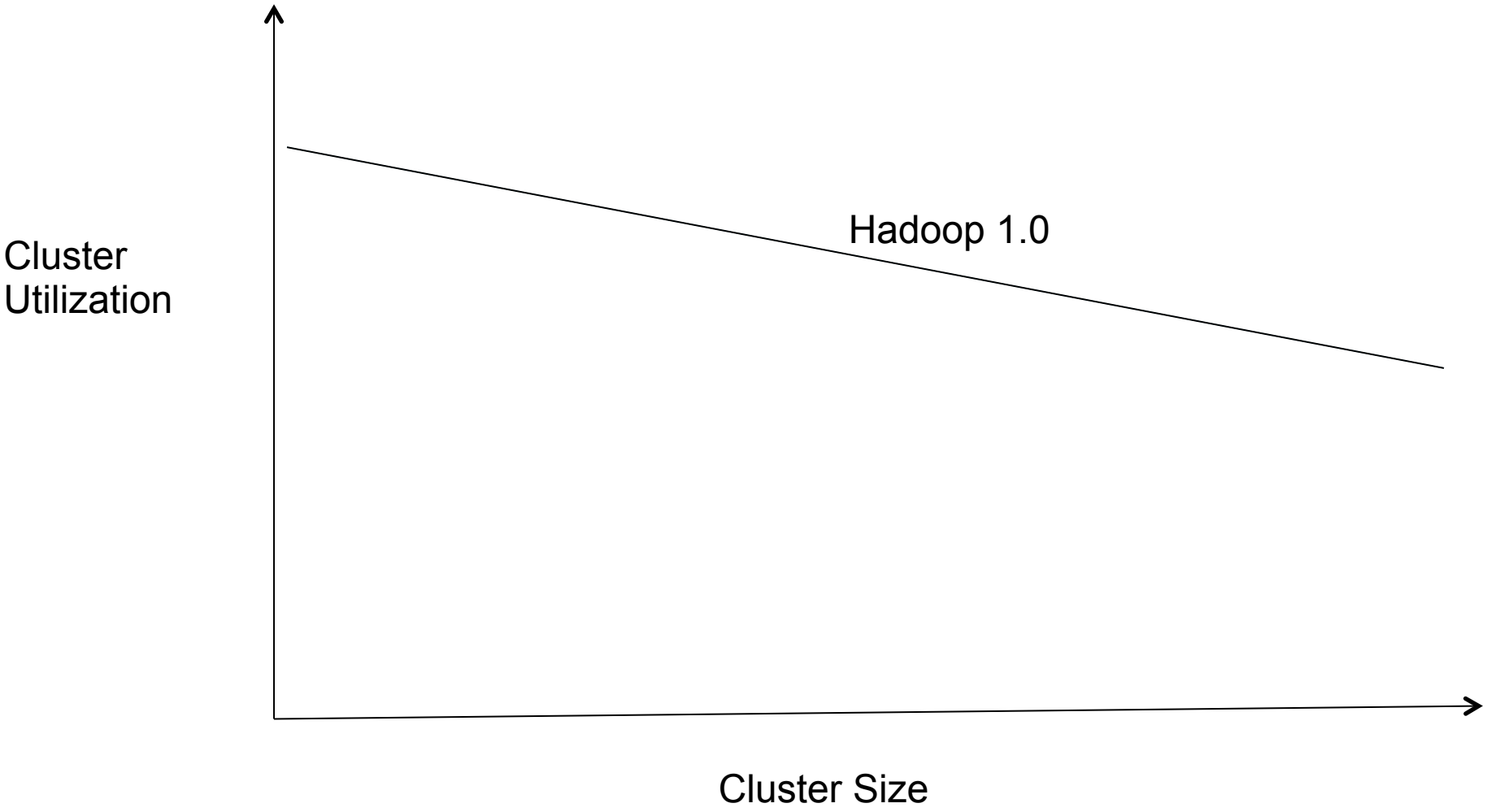
- Application Size
  - No of sub-tasks
  - Application level state
    - eg. Counters
- Number of Concurrent Tasks in a single cluster

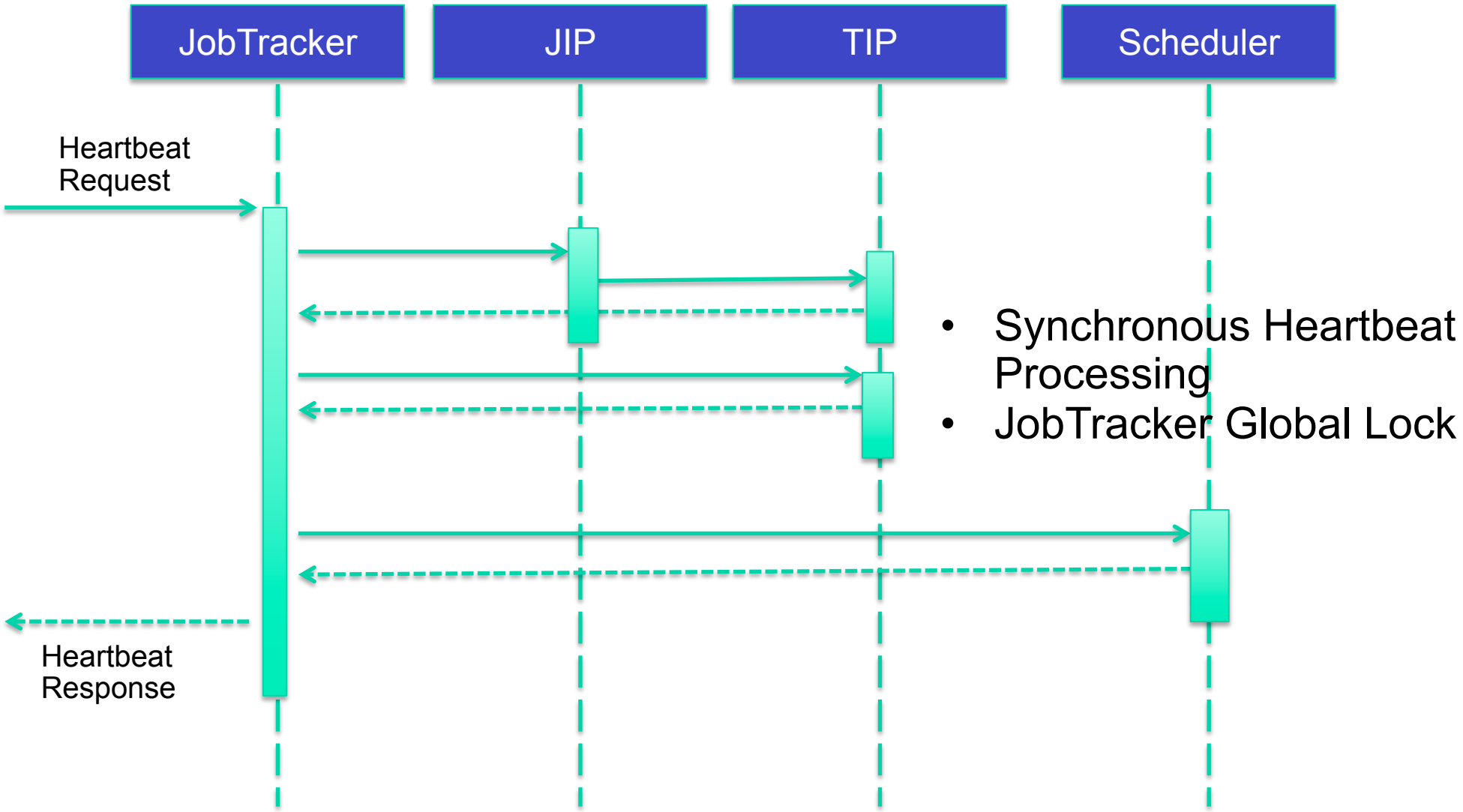
Application size scaling in Hadoop 1.0

$$JTHeap \propto TotalTasks, Nodes, JobCounters$$

Application size scaling in YARN is by Architecture

Why a limitation on cluster size ?

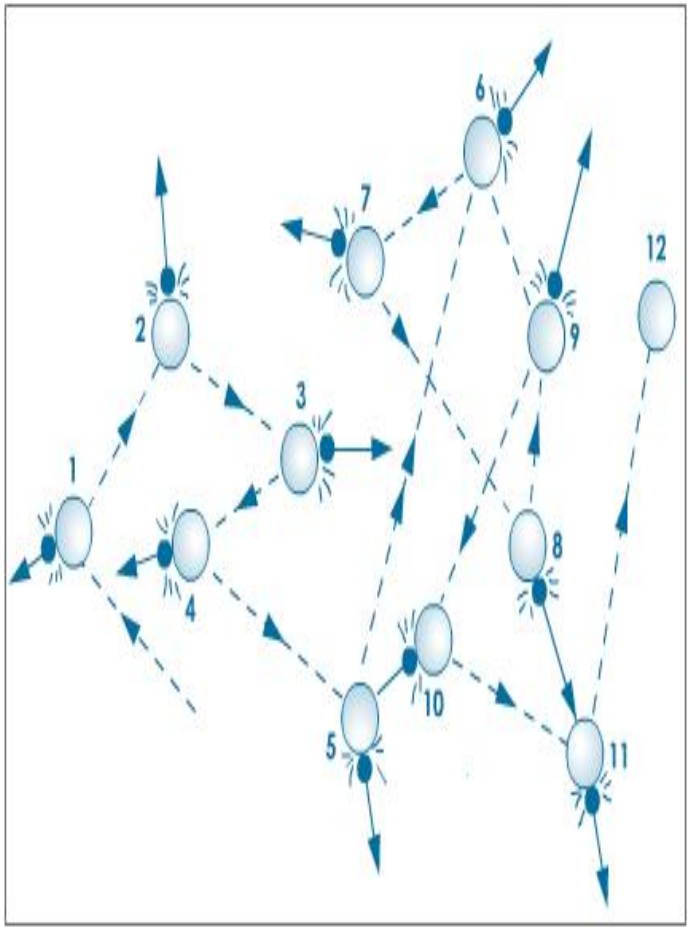




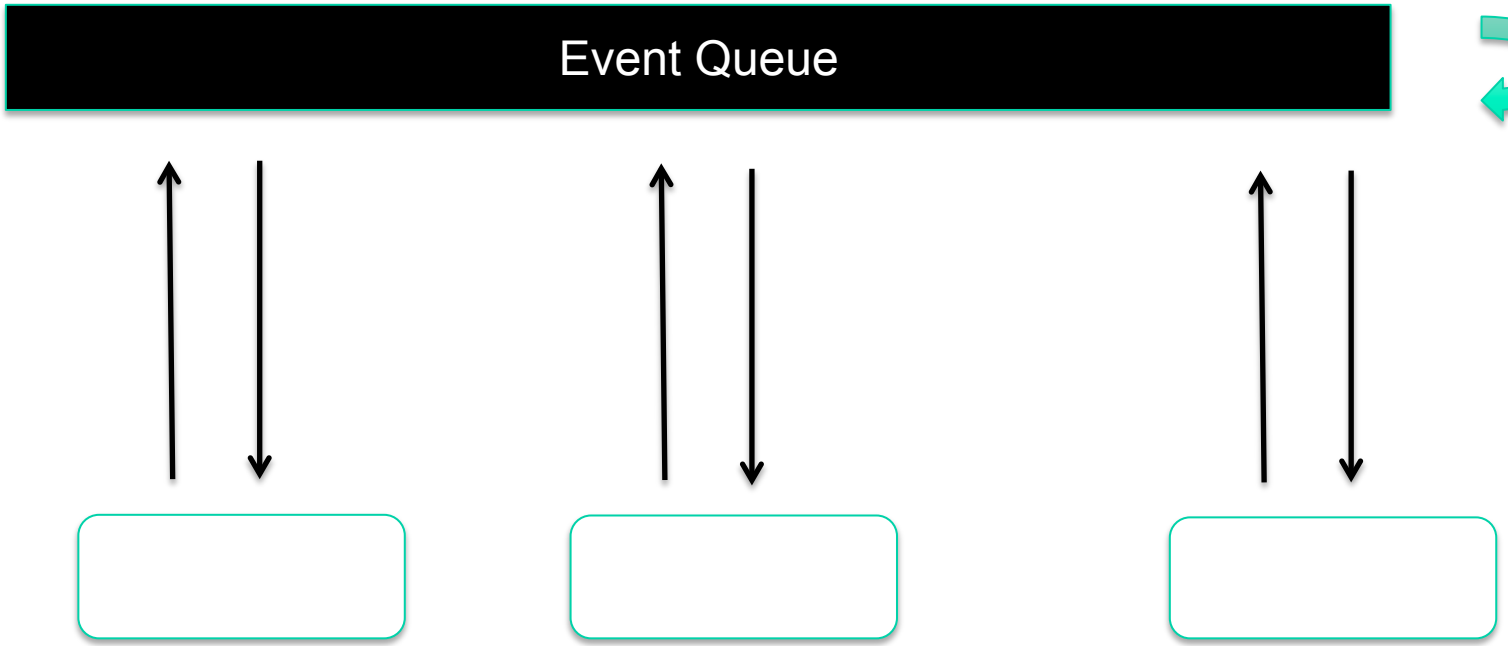
- Synchronous Heartbeat Processing
- JobTracker Global Lock

JT transaction rate limit:
200 heartbeats/sec

Highly Concurrent Systems

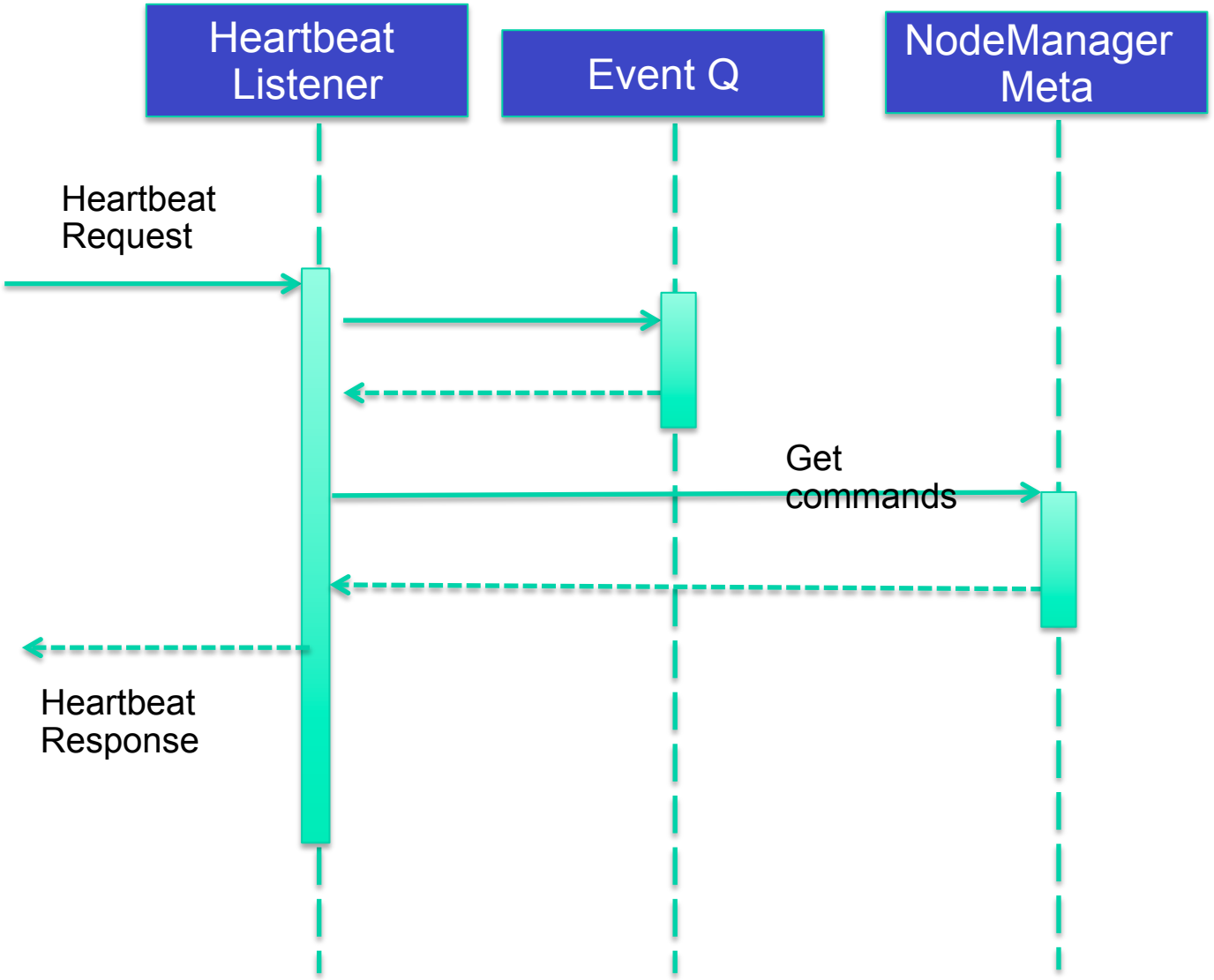


- scales much better (if done right)
- makes effective use of multi-core hardware
- managing eventual consistency of states hard
- need for a systemic framework to manage this



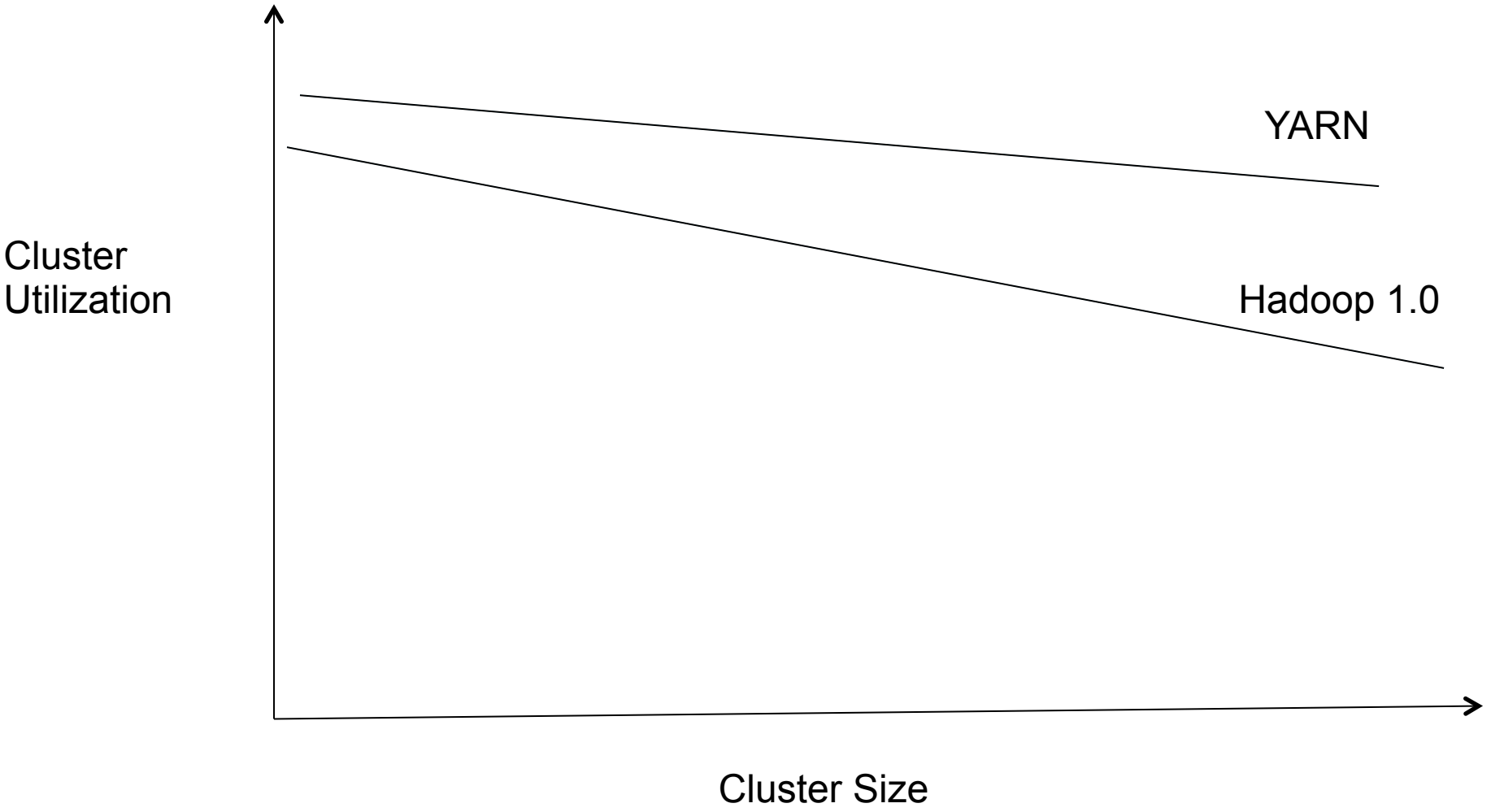
- Mutations only via events
- Components only expose Read APIs
- Use Re-entrant locks
- Components follow clear lifecycle

Event Model

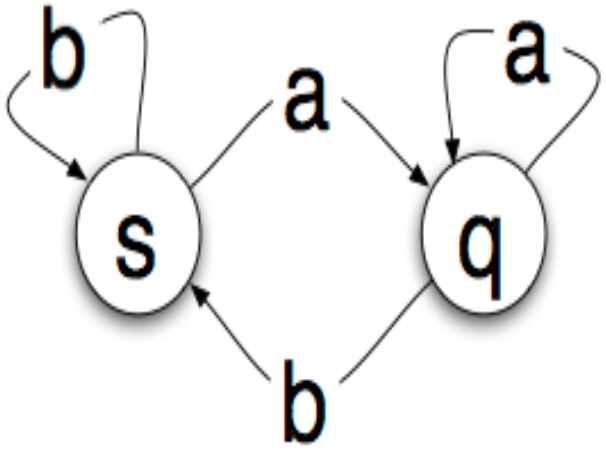


Asynchronous Heartbeat Handling

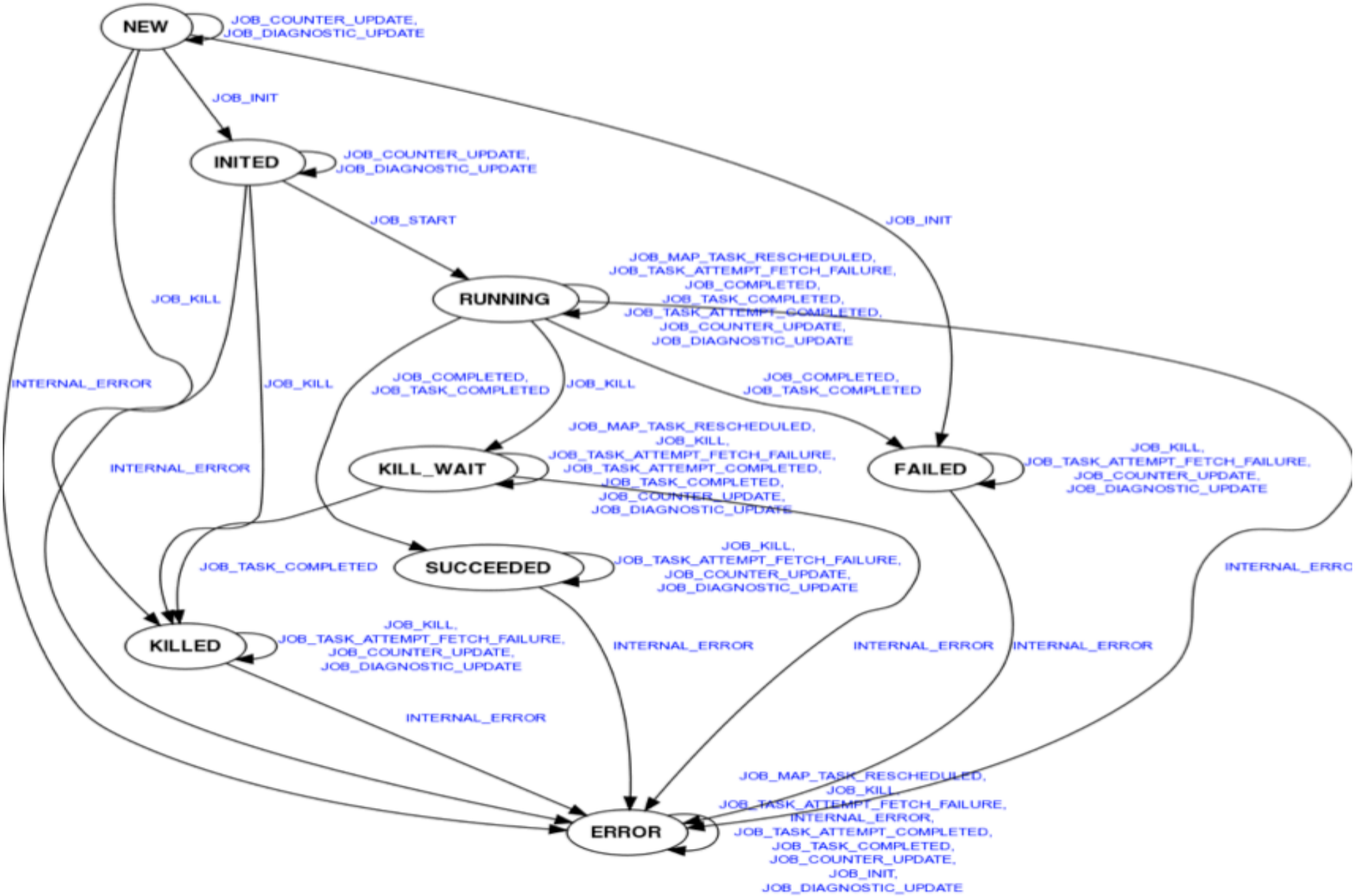
YARN: Better utilization bigger cluster



State Management



Job


NEW
JOB_COUNTER_UPDATE,
JOB_DIAGNOSTIC_UPDATE
JOB_INIT
INITED
JOB_COUNTER_UPDATE,
JOB_DIAGNOSTIC_UPDATE
JOB_START
JOB_INIT
JOB_KILL
RUNNING
JOB_MAP_TASK_RESCHEDULED,
JOB_TASK_ATTEMPT_FETCH_FAILURE,
JOB_COMPLETED,
JOB_TASK_COMPLETED,
JOB_TASK_ATTEMPT_COMPLETED,
JOB_COUNTER_UPDATE,
JOB_DIAGNOSTIC_UPDATE
INTERNAL_ERROR
JOB_KILL
JOB_COMPLETED,
JOB_TASK_COMPLETED
JOB_KILL
JOB_COMPLETED,
JOB_TASK_COMPLETED
INTERNAL_ERROR
KILL_WAIT
JOB_MAP_TASK_RESCHEDULED,
JOB_KILL,
JOB_TASK_ATTEMPT_FETCH_FAILURE,
JOB_TASK_ATTEMPT_COMPLETED,
JOB_TASK_COMPLETED,
JOB_COUNTER_UPDATE,
JOB_DIAGNOSTIC_UPDATE
FAILED
JOB_KILL,
JOB_TASK_ATTEMPT_FETCH_FAILURE,
JOB_COUNTER_UPDATE,
JOB_DIAGNOSTIC_UPDATE
JOB_TASK_COMPLETED
SUCCEEDED
JOB_KILL,
JOB_TASK_ATTEMPT_FETCH_FAILURE,
JOB_COUNTER_UPDATE,
JOB_DIAGNOSTIC_UPDATE
INTERNAL_ERRO
KILLED
JOB_KILL,
JOB_TASK_ATTEMPT_FETCH_FAILURE,
JOB_COUNTER_UPDATE,
JOB_DIAGNOSTIC_UPDATE
INTERNAL_ERROR
INTERNAL_ERROR
INTERNAL_ERROR
INTERNAL_ERROR
ERROR
JOB_MAP_TASK_RESCHEDULED,
JOB_KILL,
JOB_TASK_ATTEMPT_FETCH_FAILURE,
INTERNAL_ERROR,
JOB_TASK_ATTEMPT_COMPLETED,
JOB_TASK_COMPLETED,
JOB_COUNTER_UPDATE,
JOB_INIT,
JOB_DIAGNOSTIC_UPDATE

State management in JT

Very Hard to Maintain
Debugging even harder

```
void updateTaskStatus(..) {
  .....
  // If the job is complete and a task has just reported its
  // state as FAILED_UNCLEAN/KILLED_UNCLEAN,
  // make the task's state FAILED/KILLED without launching
     cleanup attempt.
  // Note that if task is already a cleanup attempt,
  // we don't change the state to make sure the task gets a
     killTaskAction
  if ((this.isComplete() || jobFailed || jobKilled) &&
        !tip.isCleanupAttempt(taskid)) {
    if (status.getRunState() == TaskStatus.State.FAILED_UNCLEAN)
          status.setRunState(TaskStatus.State.FAILED);
    else if (status.getRunState() ==
                TaskStatus.State.KILLED_UNCLEAN)
      status.setRunState(TaskStatus.State.KILLED);
    }
  .....
}
```

Complex State Management

- Light weight State Machines Library
- Declarative way of specifying the state Transitions
- Invalid transitions are handled automatically
- Fits nicely with the event model
- Debug-ability is drastically improved. Lineage of object states can easily be determined
- Handy while recovering the state

Declarative State Machine

```
stateMachineFactory
     = new StateMachineFactory<JobImpl, JobState, JobEventType,
                   JobEvent> (JobState.NEW)
.....
// Transitions from INITED state
.addTransition(JobState.INITED, JobState.RUNNING,
               JobEventType.JOB_START,
               new StartTransition())
           .addTransition(JobState.INITED, JobState.KILLED,
               JobEventType.JOB_KILL,
               new KillInitedJobTransition())
.....
```

High Availability

MR Application Master Recovery

- Hadoop 1.0
  - Application need to resubmit Job
  - All completed tasks are lost

- YARN
  - Application execution state check pointed in HDFS
  - Rebuilds the state by replaying the events

Resource Manager HA

- Based on Zookeeper
- Coming Soon
  - YARN-128

YARN: New Possibilities

- Open MPI - MR-2911
- Master-Worker – MR-3315
- Distributed Shell
- Graph processing – Giraph-13
- BSP – HAMA-431
- CEP
  - S4 – S4-25
  - Storm - https://github.com/nathanmarz/storm/issues/74
- Iterative processing - Spark
  https://github.com/mesos/spark-yarn/

YARN - a solid foundation to take Hadoop to next level

on

Scale, High Availability, Utilization
And
Alternate Compute Paradigms

Thank You

@twitter: sharad_ag