Improving Hadoop MapReduce Performance on Supercomputers with JVM Reuse

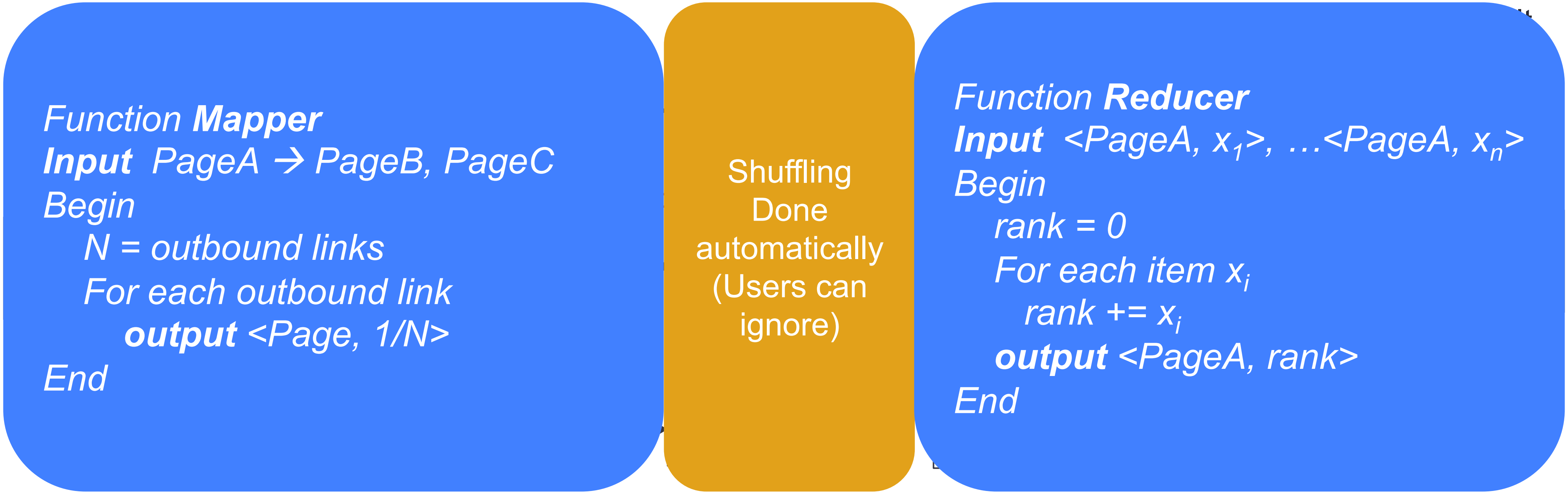
Thanh-Chung Dao and Shigeru Chiba
The University of Tokyo

Supercomputers

- Expensive clusters
    - Multi-core processors
    - Large capacity of main memory
    - High-speed network
- Focus mainly on compute-intensive applications
- Data-intensive workloads are emerging as supercomputing problems
    - Graph processing
    - Pre-processing of simulation data

MapReduce

- Simple parallel paradigm to process large datasets
- Hidden parallelization & communication
- PageRank example



```
Function Mapper
Input  PageA → PageB, PageC
Begin
    N = outbound links
    For each outbound link
        output <Page, 1/N>
End
```

Shuffling Done automatically (Users can ignore)

```
Function Reducer
Input  <PageA, $x_1$>, …<PageA, $x_n$>
Begin
    rank = 0
    For each item $x_i$
        rank += $x_i$
    output <PageA, rank>
End
```

Hadoop MapReduce

- Standard of MapReduce implementation
- Provide easy-to-use MapReduce APIs
- TCP/IP-based communication
- Designed to run on commodity clusters
  - Lab clusters, or Amazon EC2
- Scalability (32,000 nodes at Yahoo) & Resilience
- Written in Java

Improving Hadoop MapReduce Performance on Supercomputers

- Hadoop MapReduce is good choice on supercomputers
  - Maturity
  - Productivity

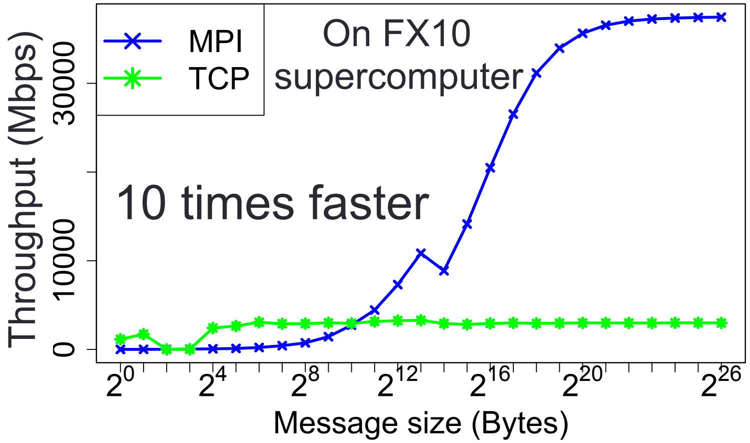
	Supercomputer	Hadoop
Resource allocation at runtime (# of processes, memory, CPU)	Static	Dynamic
Communication	MPI	TCP/IP
Workload	Compute-intensive	Data-intensive

Our Approach

- JVM Reuse
  - Statically create JVM processes and dynamically allocate to Hadoop tasks
    - Enable efficient MPI communication by Hadoop tasks
      - Statically created processes can exploit efficient MPI
      - Dynamic allocation enables to use the original Hadoop implementation
    - Shorten start-up time of processes
- Technique
  - Process pool is used to implement JVM Reuse
  - Minimize changes of the original Hadoop engine

Why MPI is required for Hadoop

- The de facto high-speed communication on supercomputers
  - Improve slow MapReduce shuffling



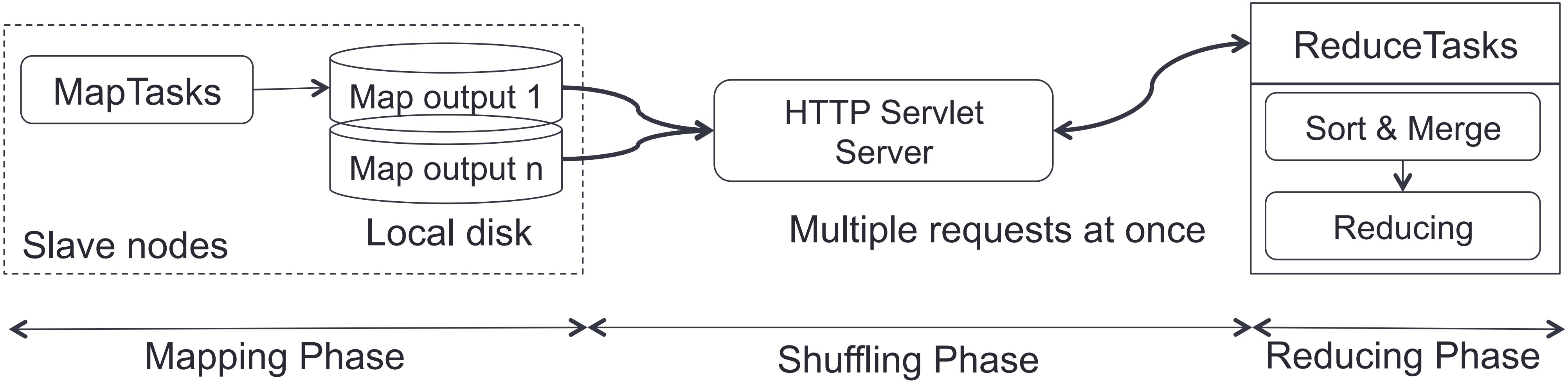
On FX10 supercomputer

10 times faster

- Enable Hadoop to co-host traditional MPI applications
  - Combine MPI and MapReduce models
  - Rich data analysis workflow
    - Efficient data sharing between MPI and MapReduce models
      - E.g. MPI can access data located at Hadoop file system (HDFS)

Slow MapReduce shuffling on Hadoop

- TCP/IP-based communication
  - JVM-Bypass (Wang et al., IPDPS 2013)



Dynamic Process Creation on MPI

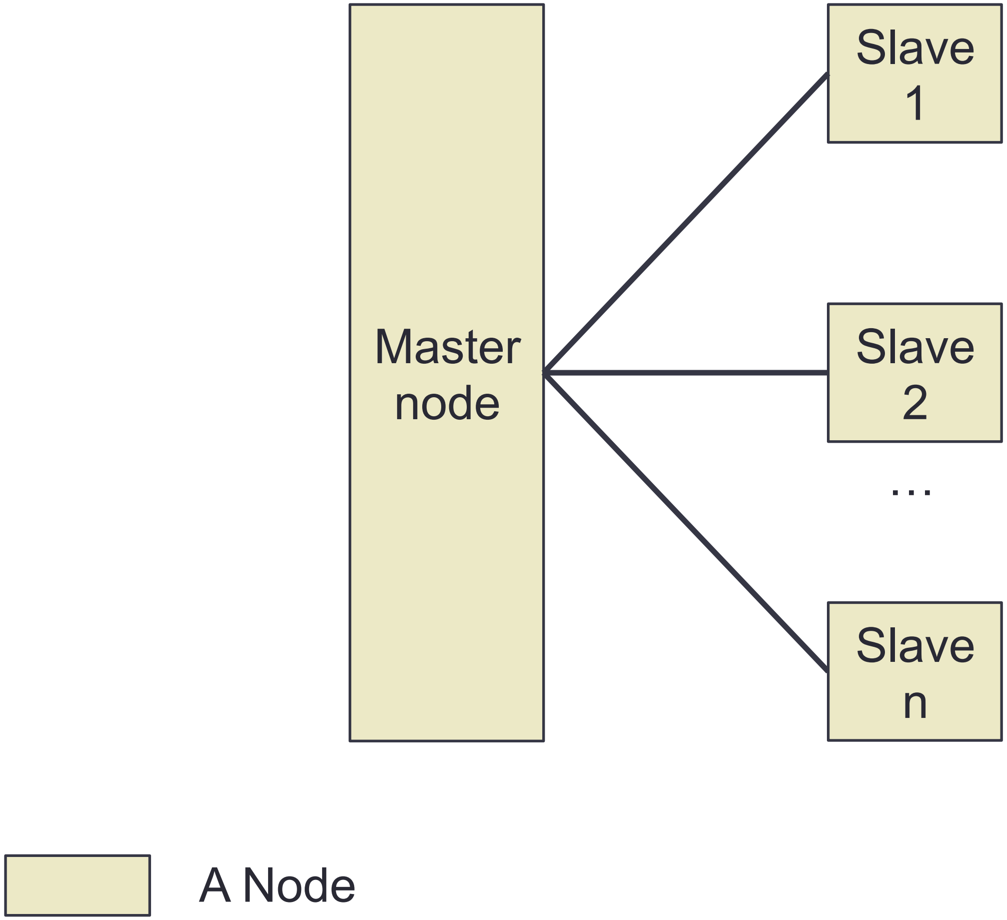
- Discouraged on supercomputers
  - Reasons of performance
    - Collective mechanism (MPISpawn)
    - Gang scheduling (error-prone if not enough resource)
  - Gerbil (Xu el al., CCGrid 2015)
    - Co-hosting MPI applications on Hadoop
    - Creating dynamically processes
    - Its experiments showed significant overhead
- Resources should be specified before running MPI applications
  - Number of processes is known (static)
  - Memory and CPU cores

Dynamic Process Creation on Hadoop

- Required
  - Resources are allocated on demand to run MapReduce applications
  - Number of processes is unknown (dynamic)

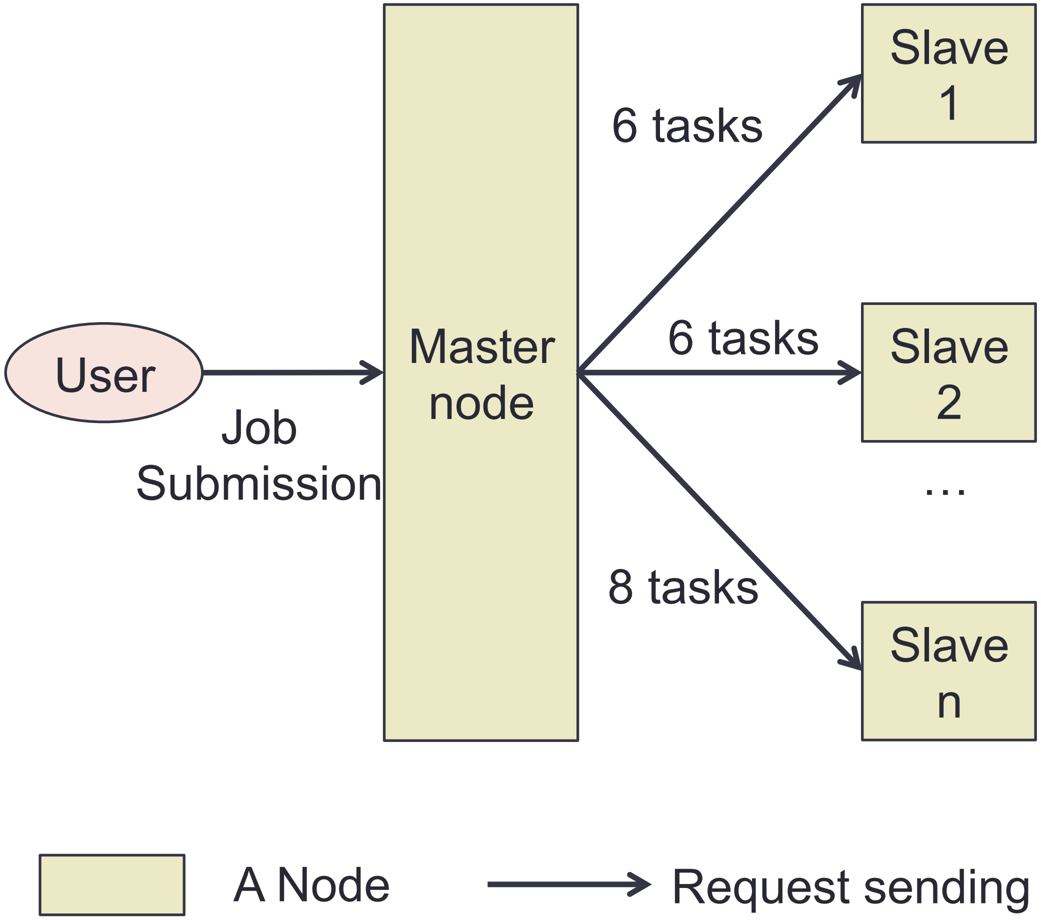
Dynamic Process Creation on Hadoop

- Required
  - Resources are allocated on demand to run MapReduce applications
  - Number of processes is unknown (dynamic)



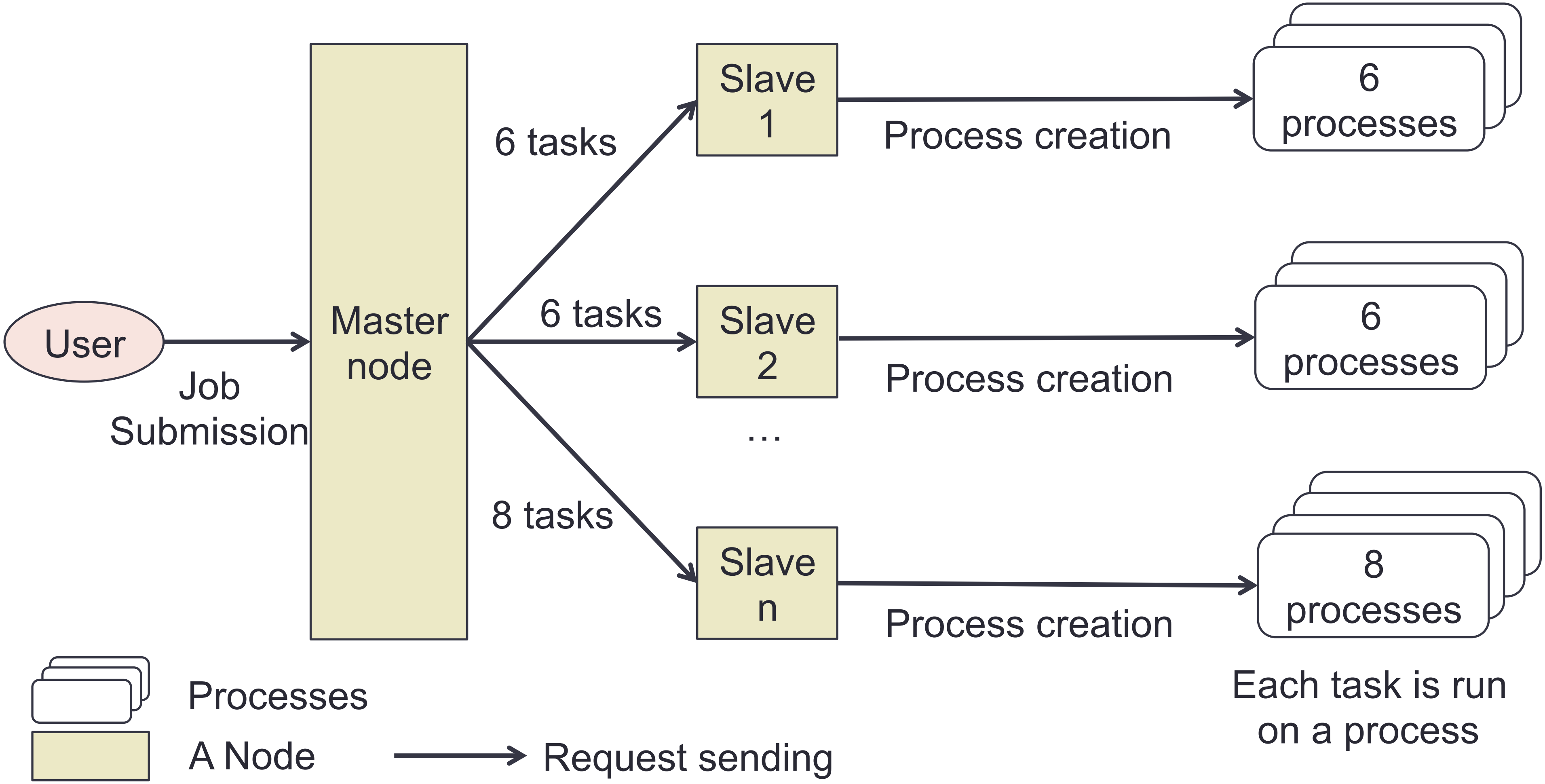
Dynamic Process Creation on Hadoop

- Required
  - Resources are allocated on demand to run MapReduce applications
  - Number of processes is unknown (dynamic)



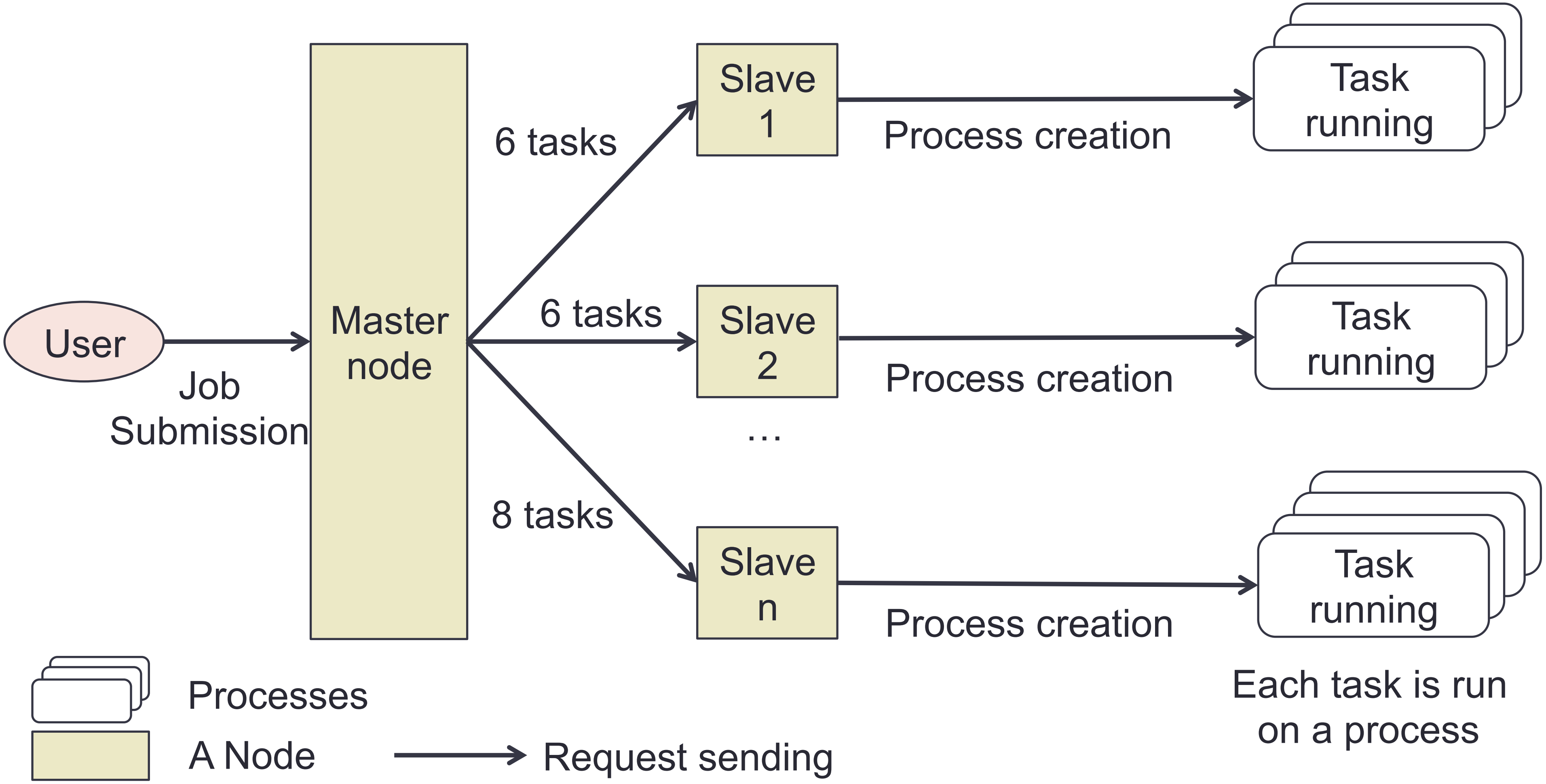
Dynamic Process Creation on Hadoop

- Required
  - Resources are allocated on demand to run MapReduce applications
  - Number of processes is unknown (dynamic)



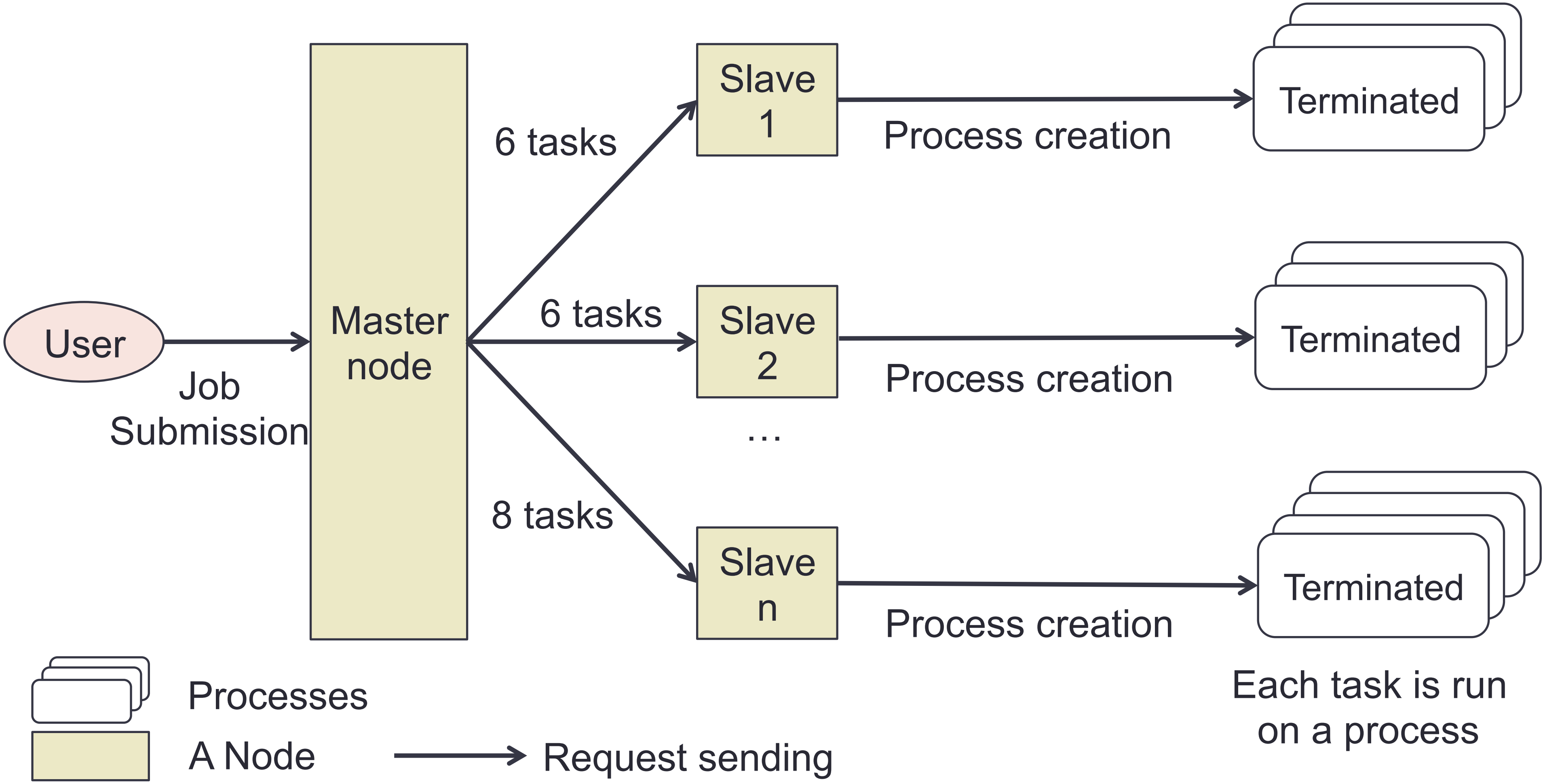
Dynamic Process Creation on Hadoop

- Required
  - Resources are allocated on demand to run MapReduce applications
  - Number of processes is unknown (dynamic)



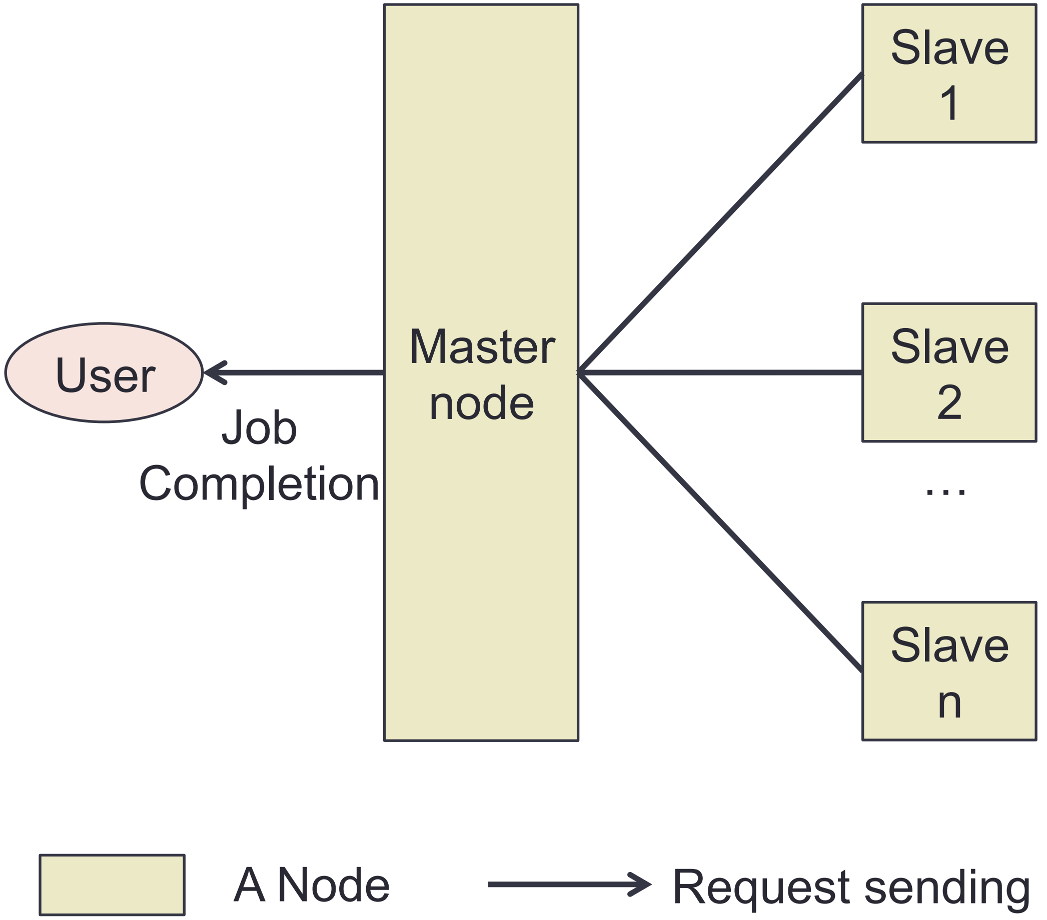
Dynamic Process Creation on Hadoop

- Required
  - Resources are allocated on demand to run MapReduce applications
  - Number of processes is unknown (dynamic)



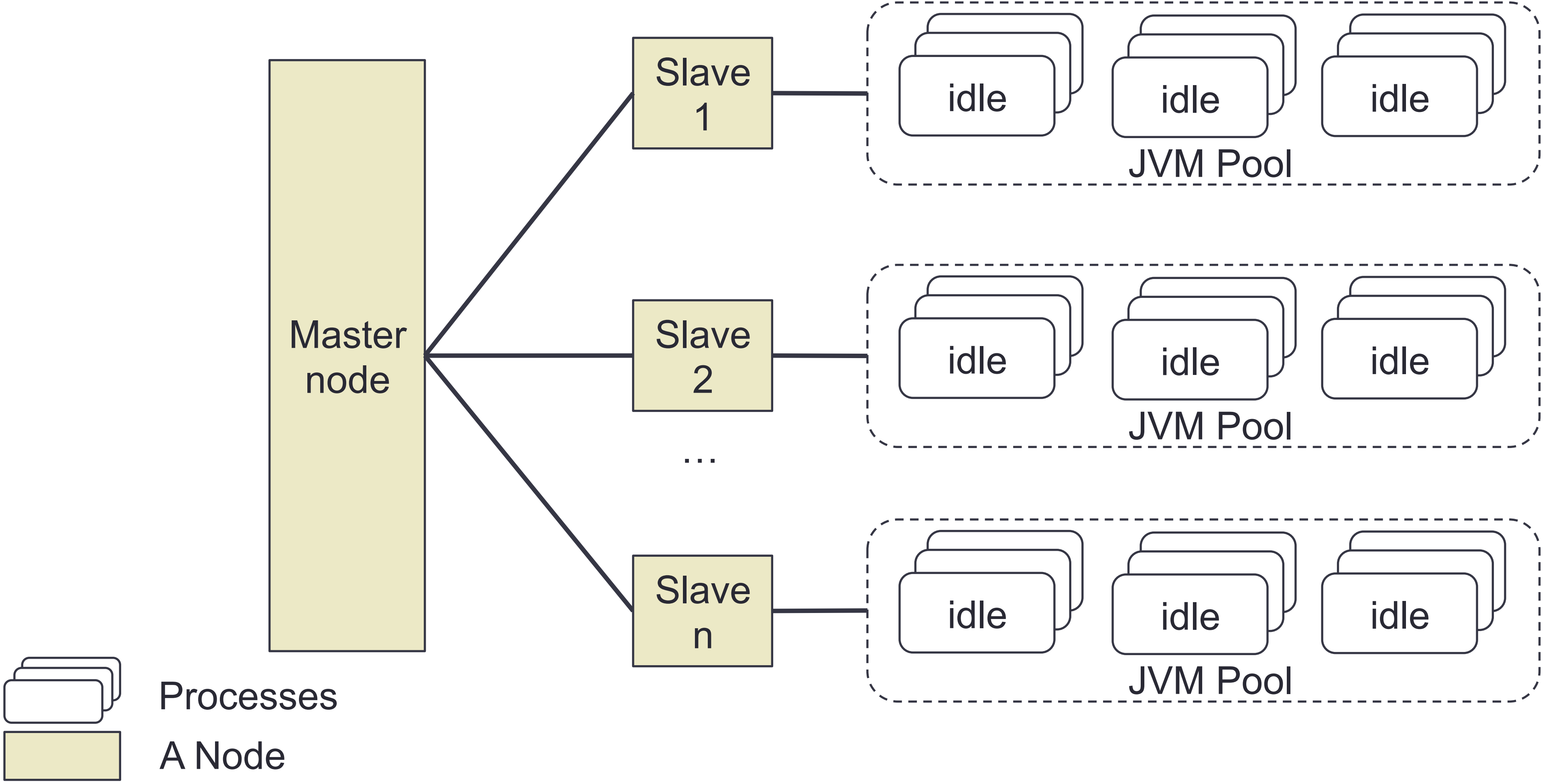
Dynamic Process Creation on Hadoop

- Required
  - Resources are allocated on demand to run MapReduce applications
  - Number of processes is unknown (dynamic)



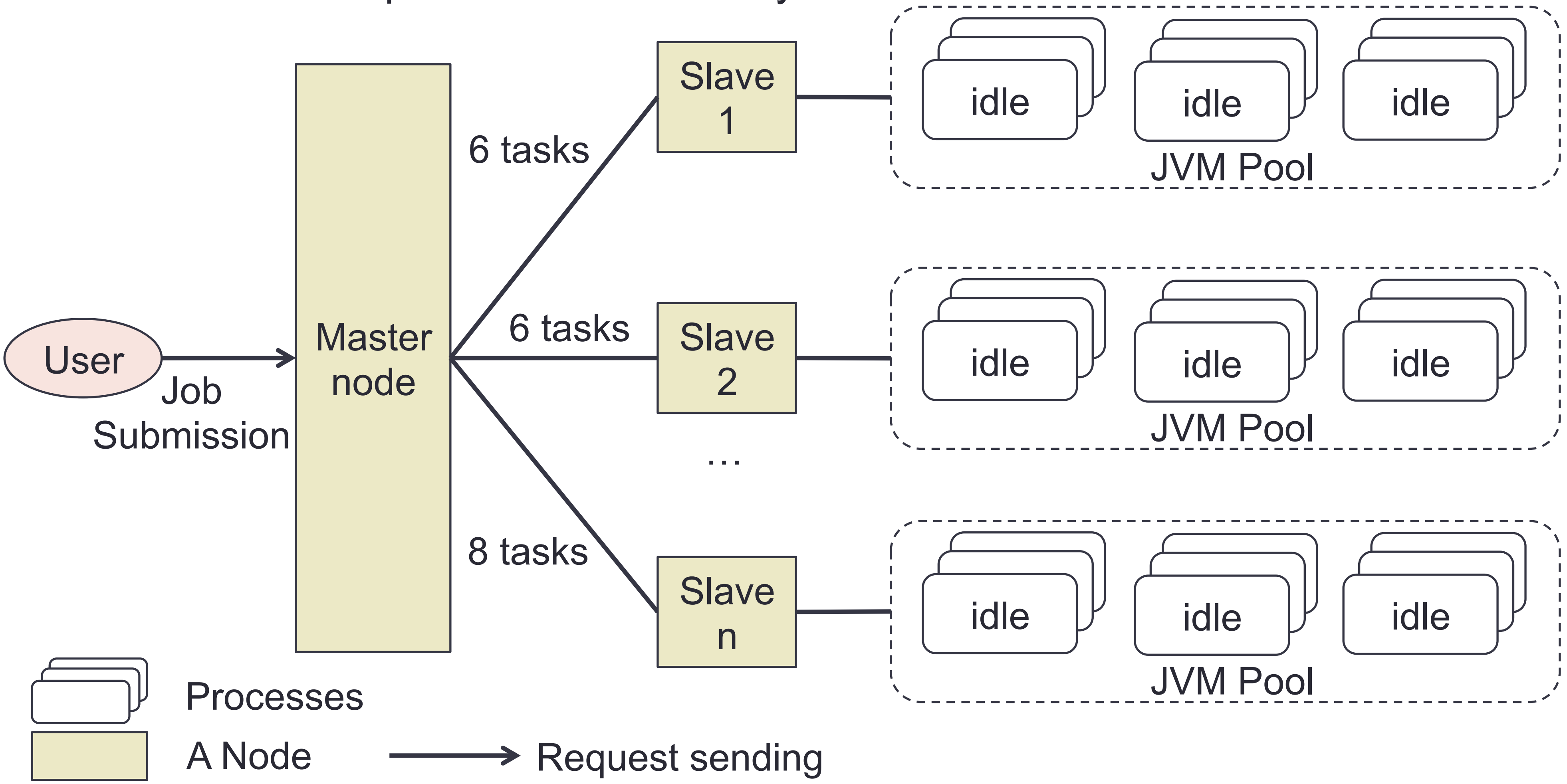
Idea of Reusing

- JVM Pool added
  - Idle JVM processes
  - Number of processes is statically fixed



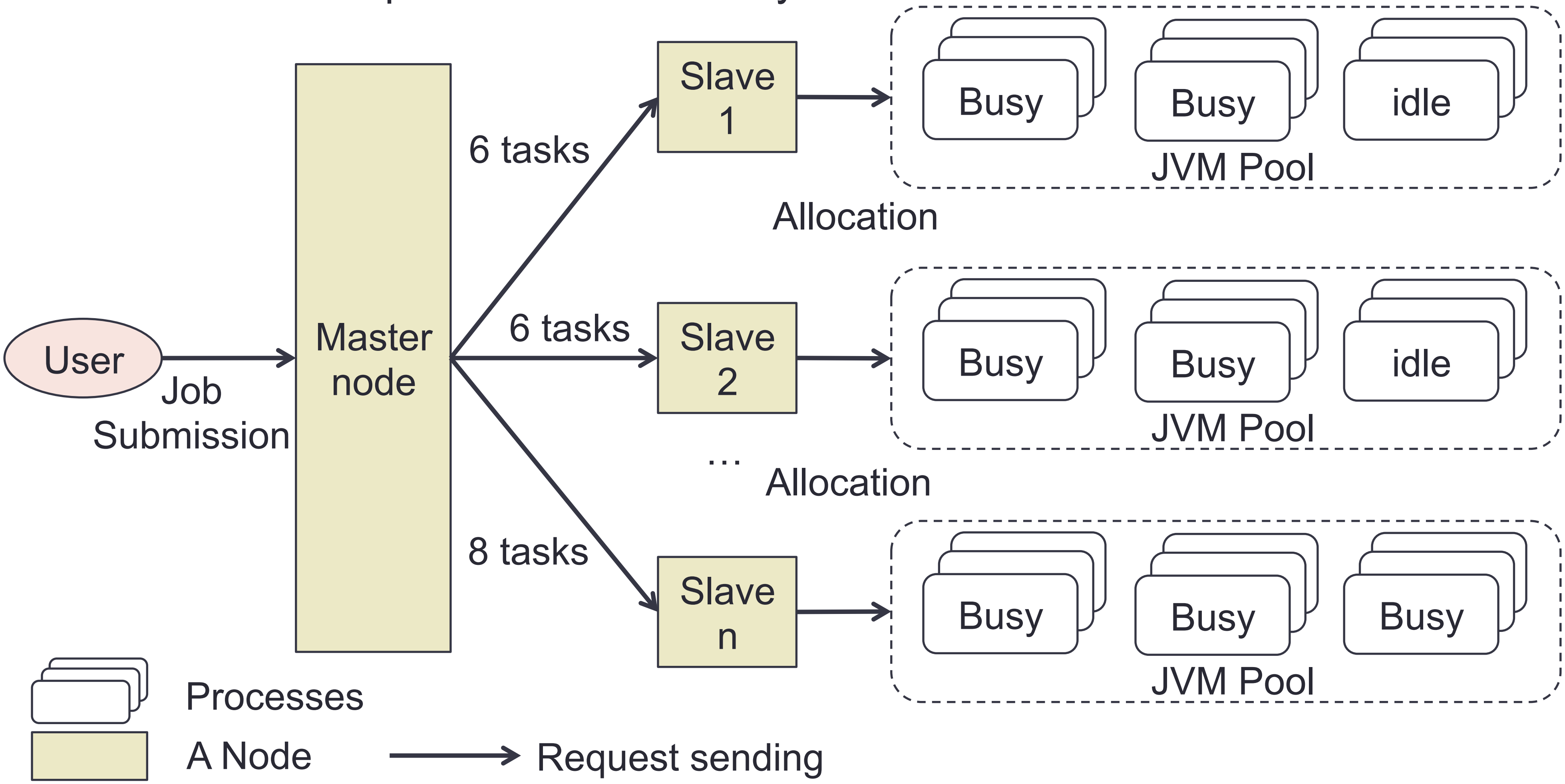
Idea of Reusing

- JVM Pool added
  - Idle JVM processes
  - Number of processes is statically fixed



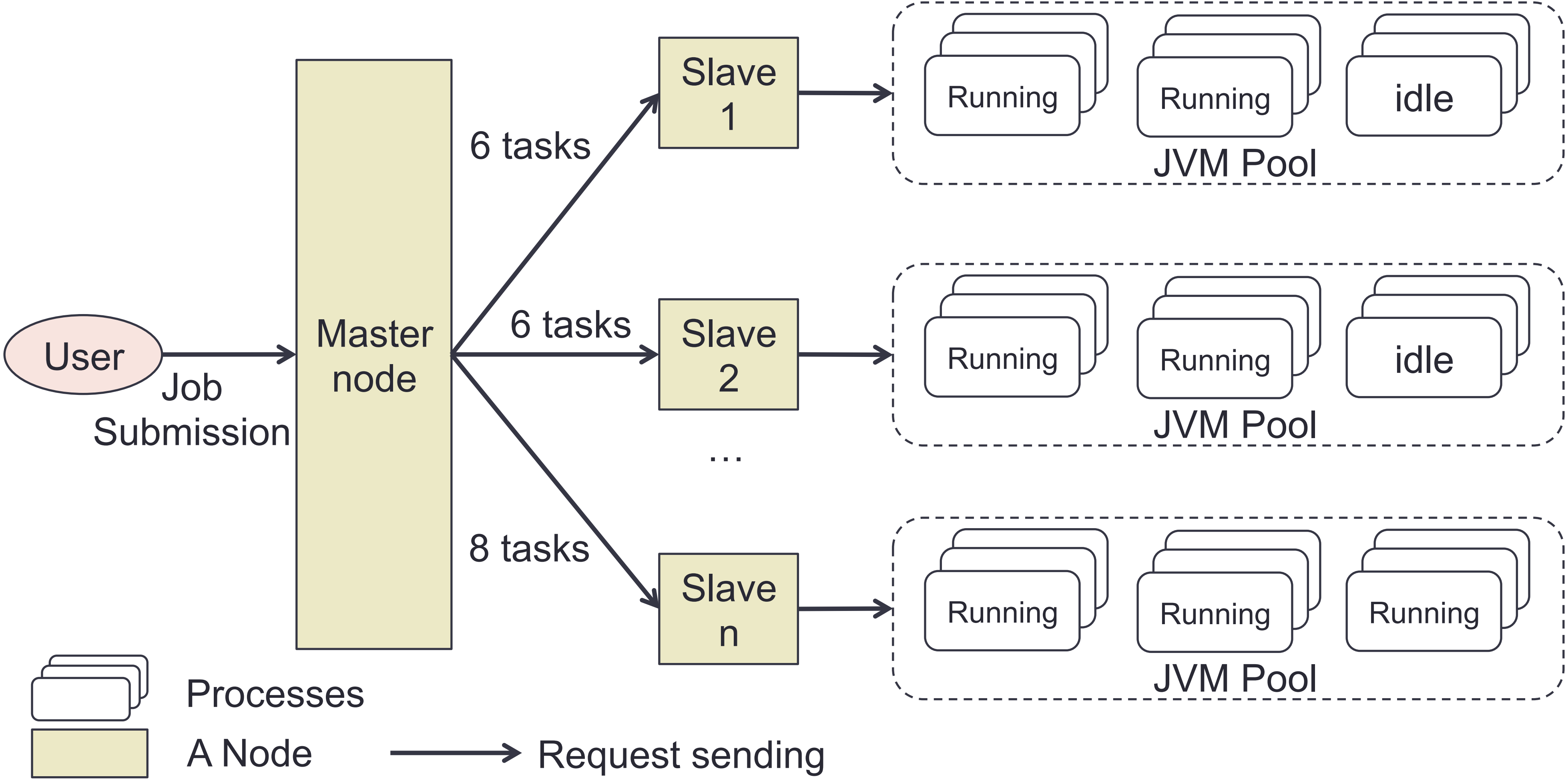
Idea of Reusing

- JVM Pool added
  - Idle JVM processes
  - Number of processes is statically fixed



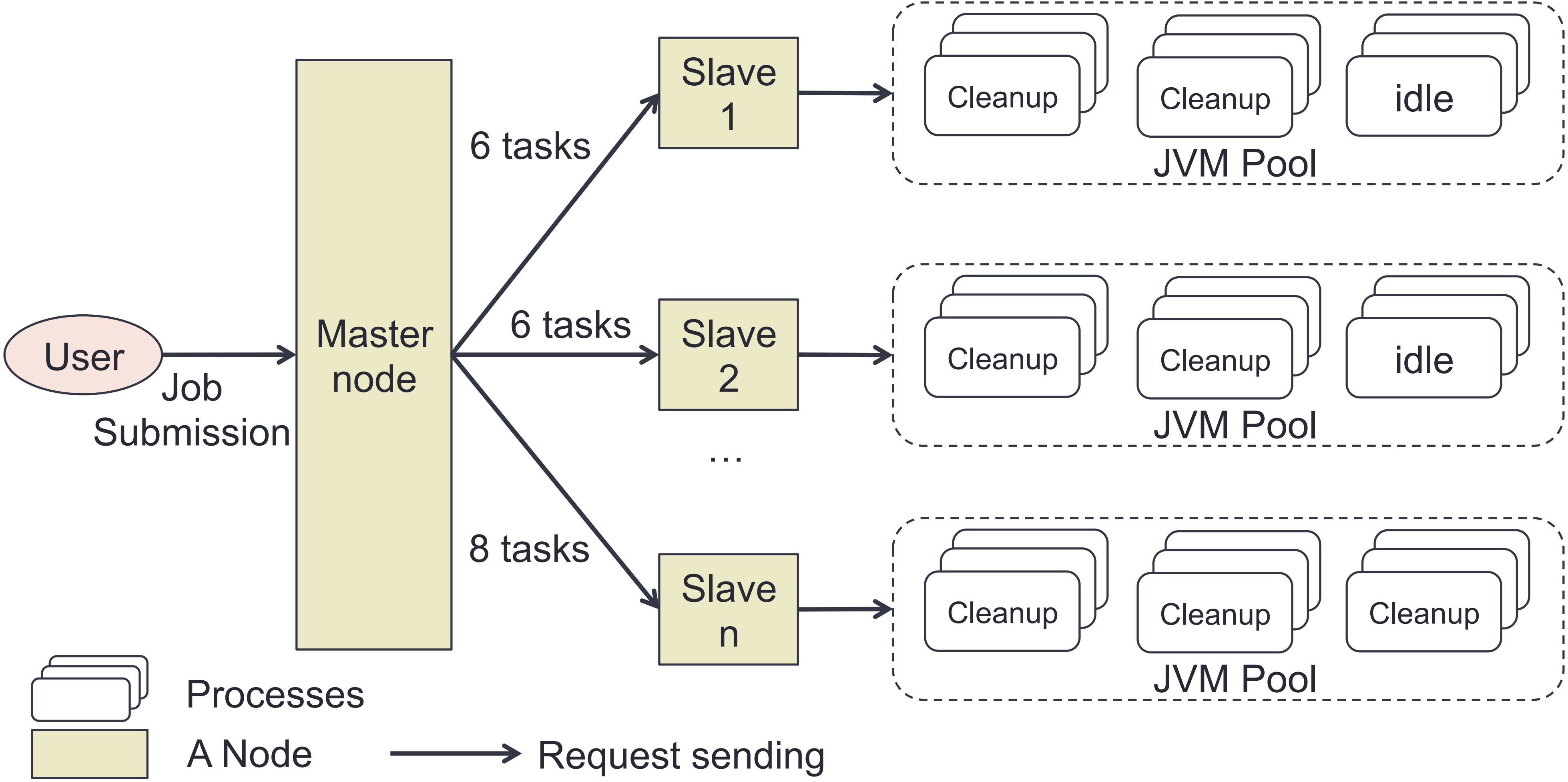
Idea of Reusing

- JVM Pool added
  - Idle JVM processes
  - Number of processes is statically fixed



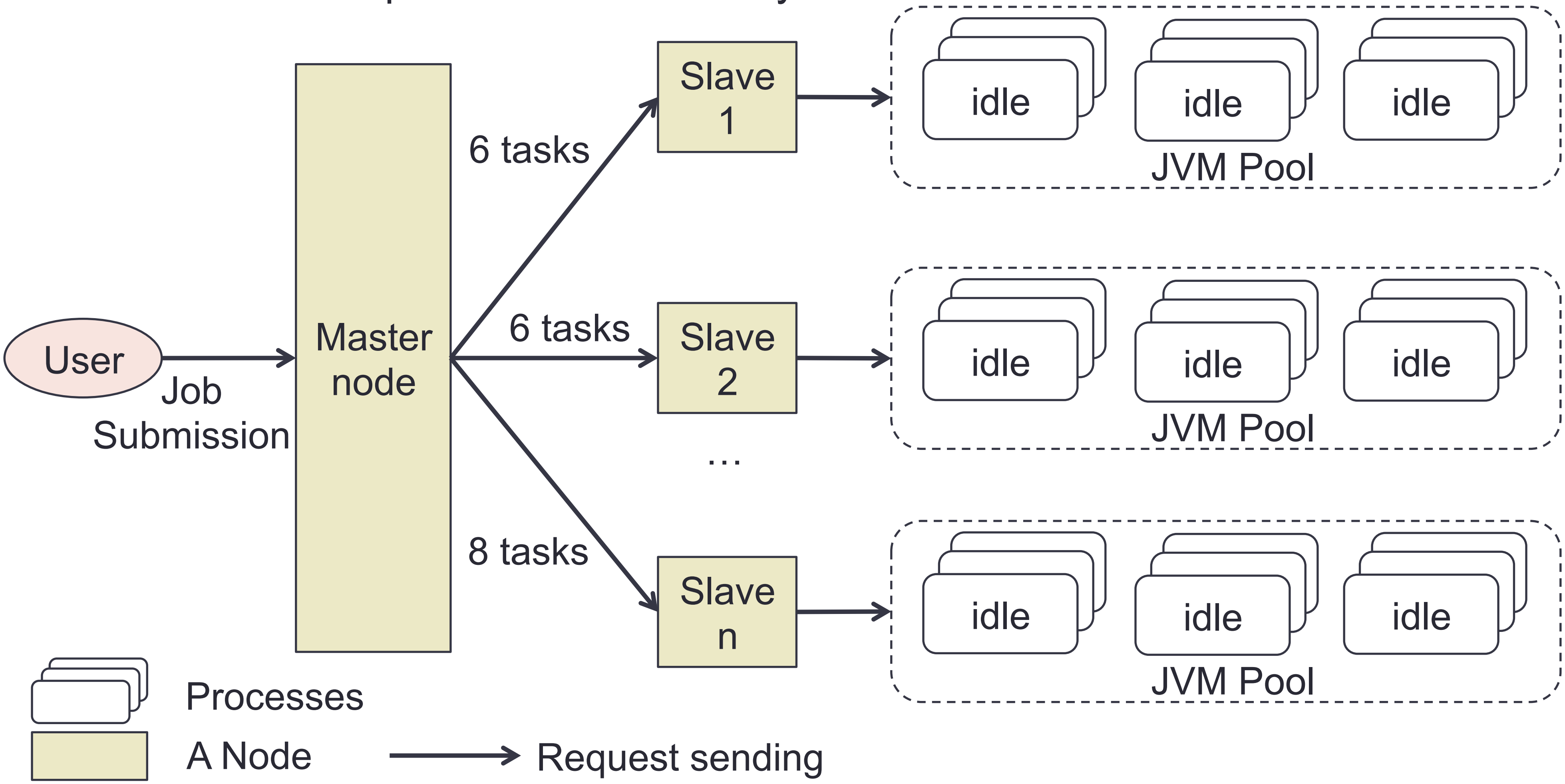
Idea of Reusing

- JVM Pool added
  - Idle JVM processes
  - Number of processes is statically fixed



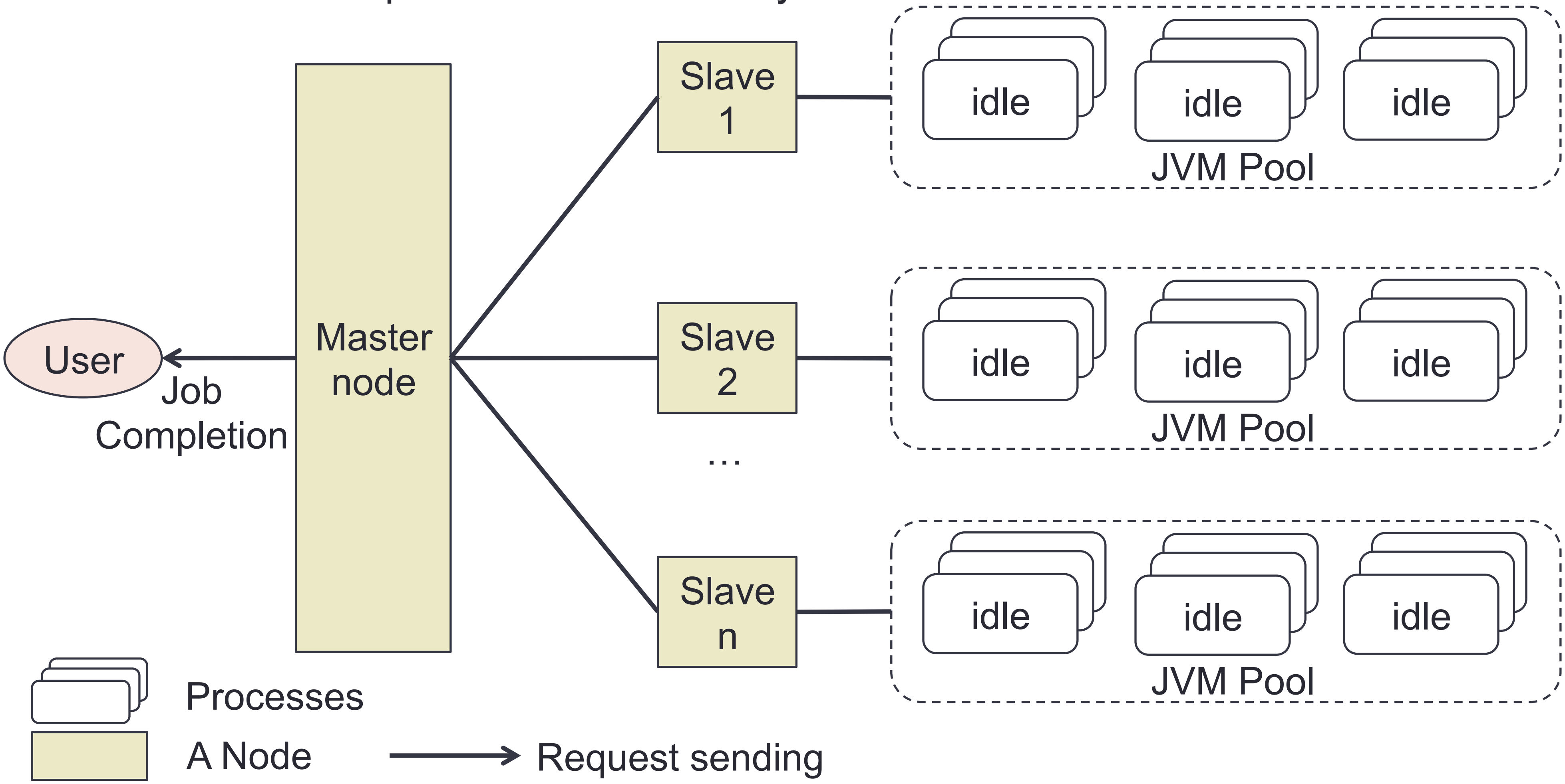
Idea of Reusing

- JVM Pool added
  - Idle JVM processes
  - Number of processes is statically fixed



Idea of Reusing

- JVM Pool added
  - Idle JVM processes
  - Number of processes is statically fixed

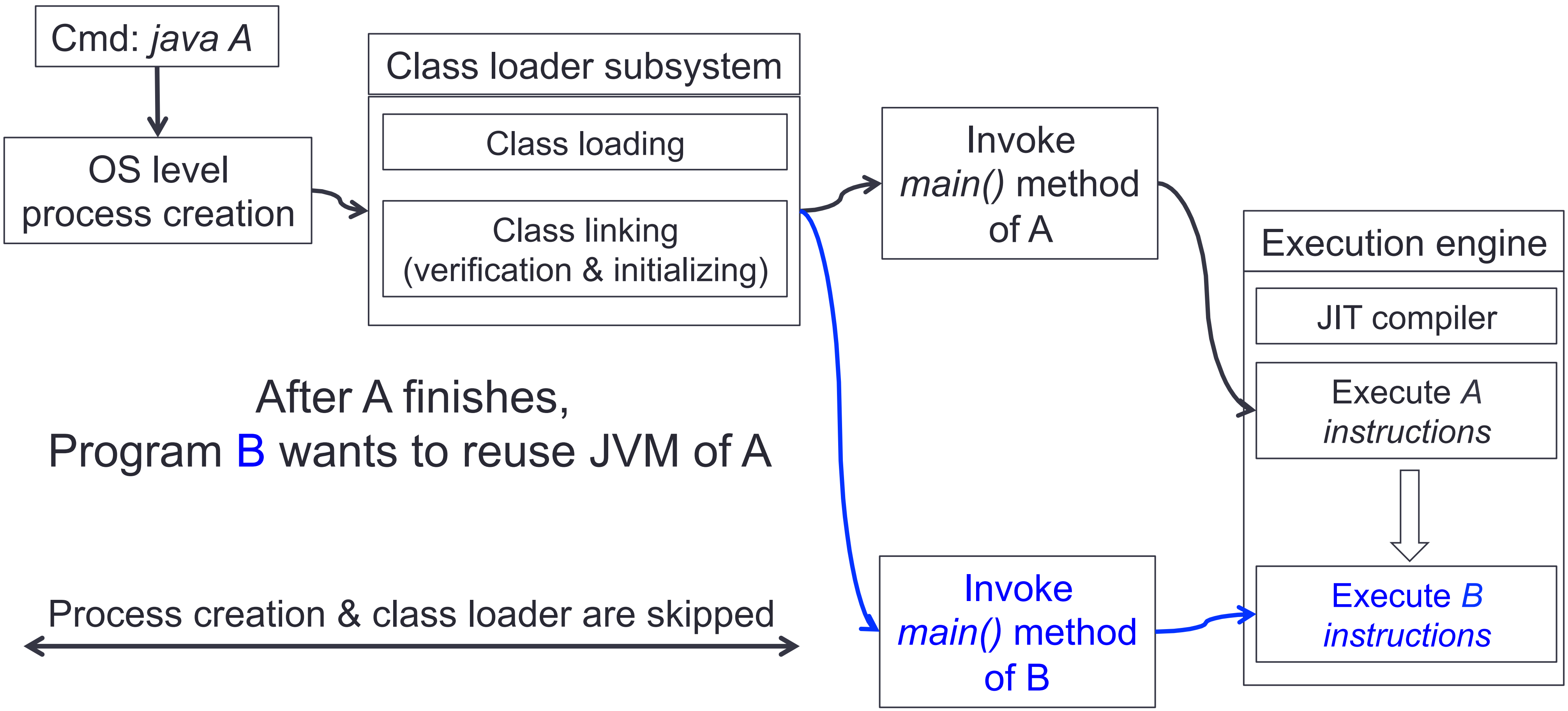


JVM Reuse enables MPI communication

- MPI communication is established at the beginning
- JVM Reuse keeps processes running
  - MPI connection is always available

JVM Reuse shortens start-up time

JVM start-up flow of Program A



Cmd: *java A*

OS level process creation

Class loader subsystem

Class loading

Class linking (verification & initializing)

Invoke *main()* method of A

Execution engine

JIT compiler

Execute *A instructions*

Execute *B instructions*

Invoke *main()* method of B

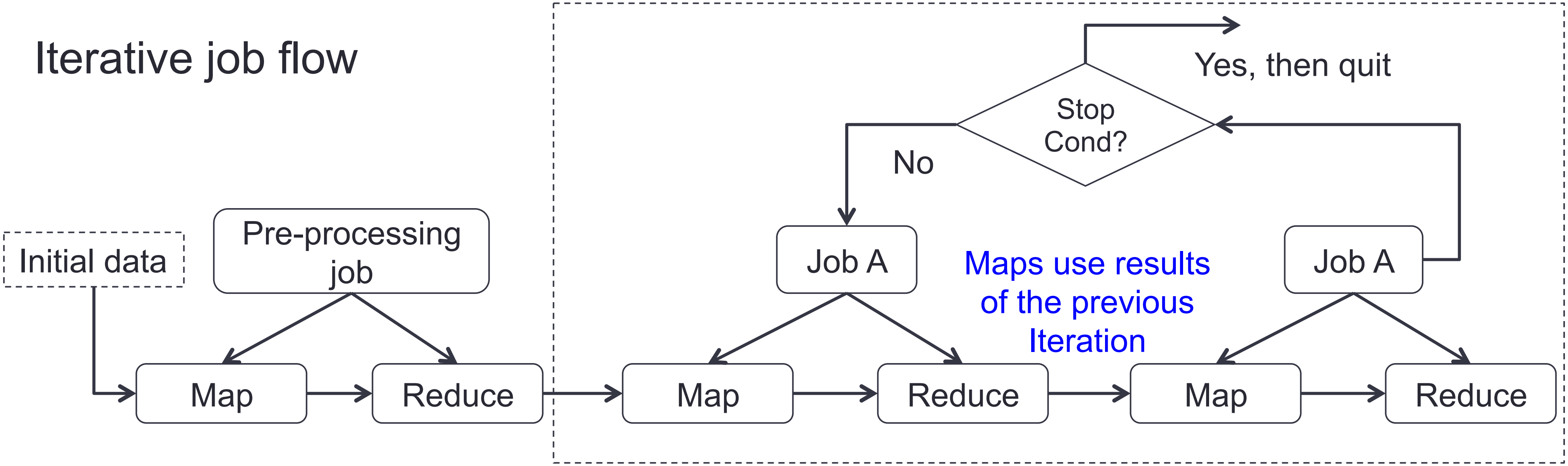
After A finishes,
Program B wants to reuse JVM of A

Process creation & class loader are skipped

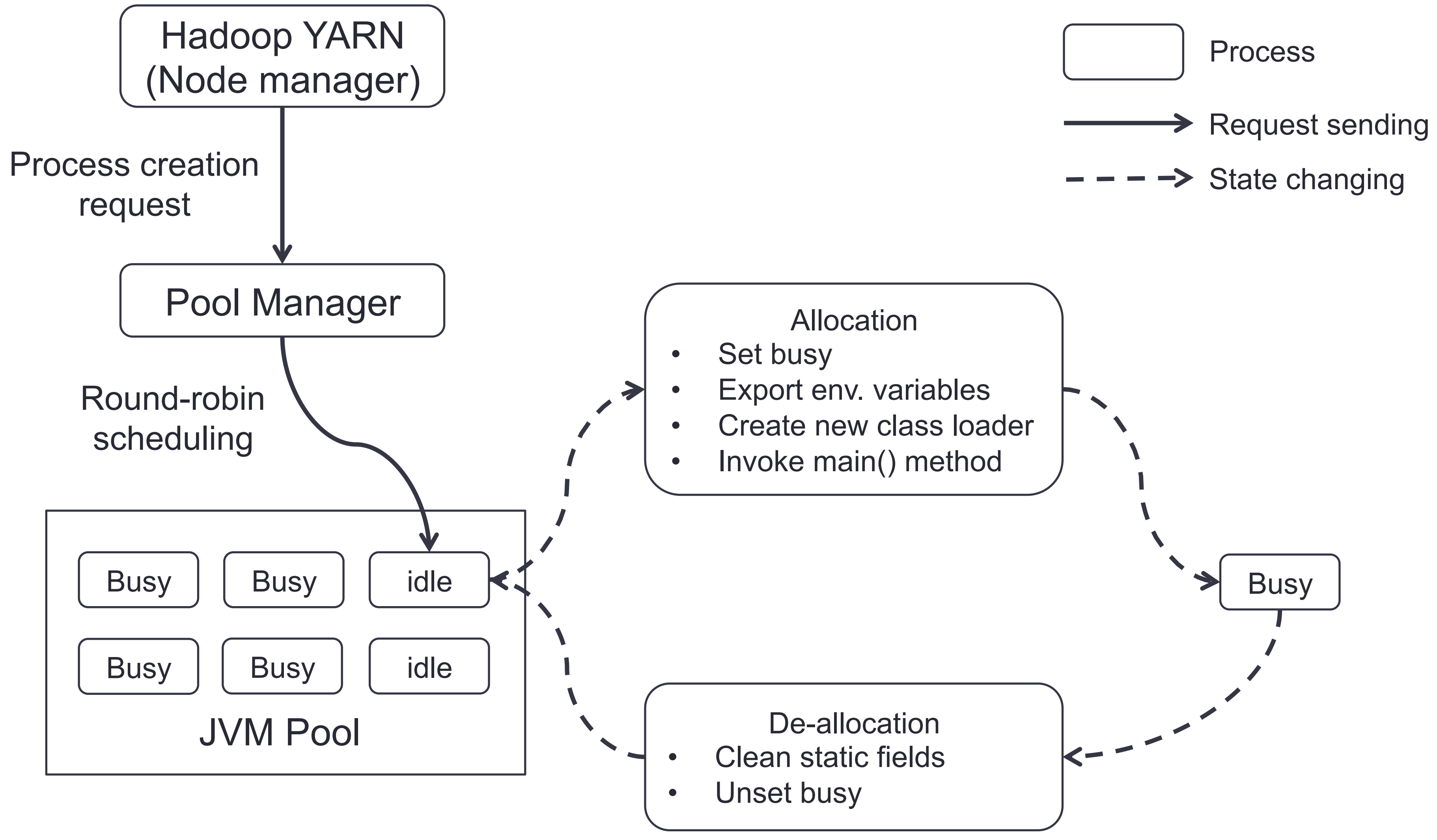
Iterative jobs benefit from JVM Reuse

- Iterative jobs
  - Many short running JVM processes
  - PageRank is an example

Iterative job flow

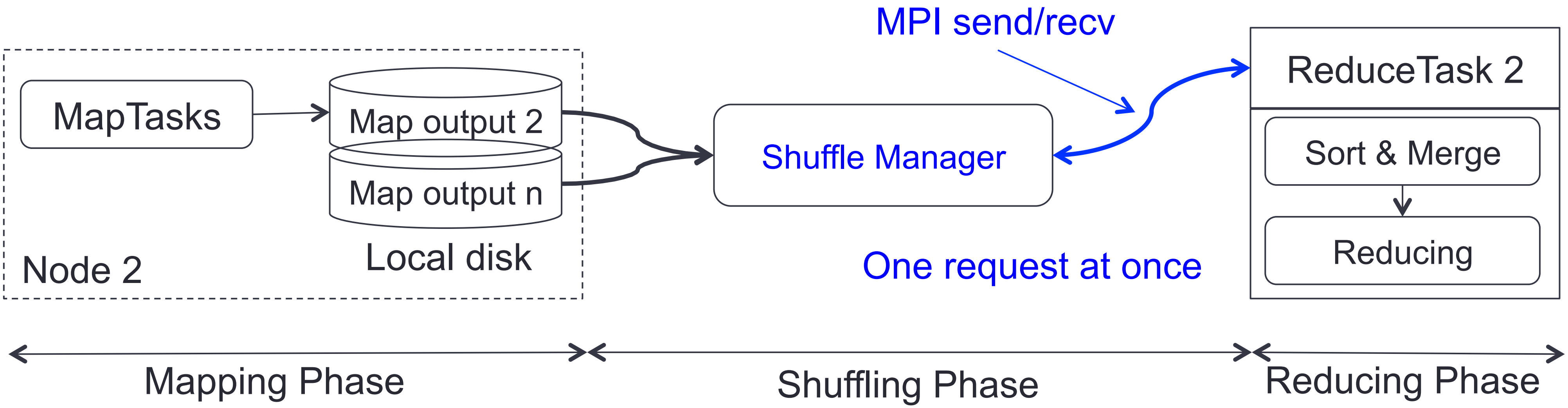


Implementation: Process Pool



Process Pool on each node

Our MPI shuffling design



Reuse's Technical Issues

- Loading user's classes
  - The original flow exports CLASSPATH before running
  - Reflection
    - Load user's classes at runtime
    - Create a new class loader for each user
      - Avoid class confliction
- Clean-up
  - Static fields
    - Security problem
      - e.g. UserGroup static field
    - Must be reset
  - Current design
    - Reset user information and job conf. static fields

Other Technical Issues

- Enable Hadoop YARN to host traditional MPI applications
  - YARN is a resource manager
  - Work in progress
    - MPI AppMaster
      - Monitor MPI ranks
    - MPI Container
      - Host a rank
- Avoid gang scheduling of MPI
  - Work in progress

Evaluation

- Hadoop version
  - v2.2.0
- Changes in our implementation
  - Line of code / total of Hadoop: ~1000 / 1,851,473
  - Number of classes / total of Hadoop: 9 / 35142

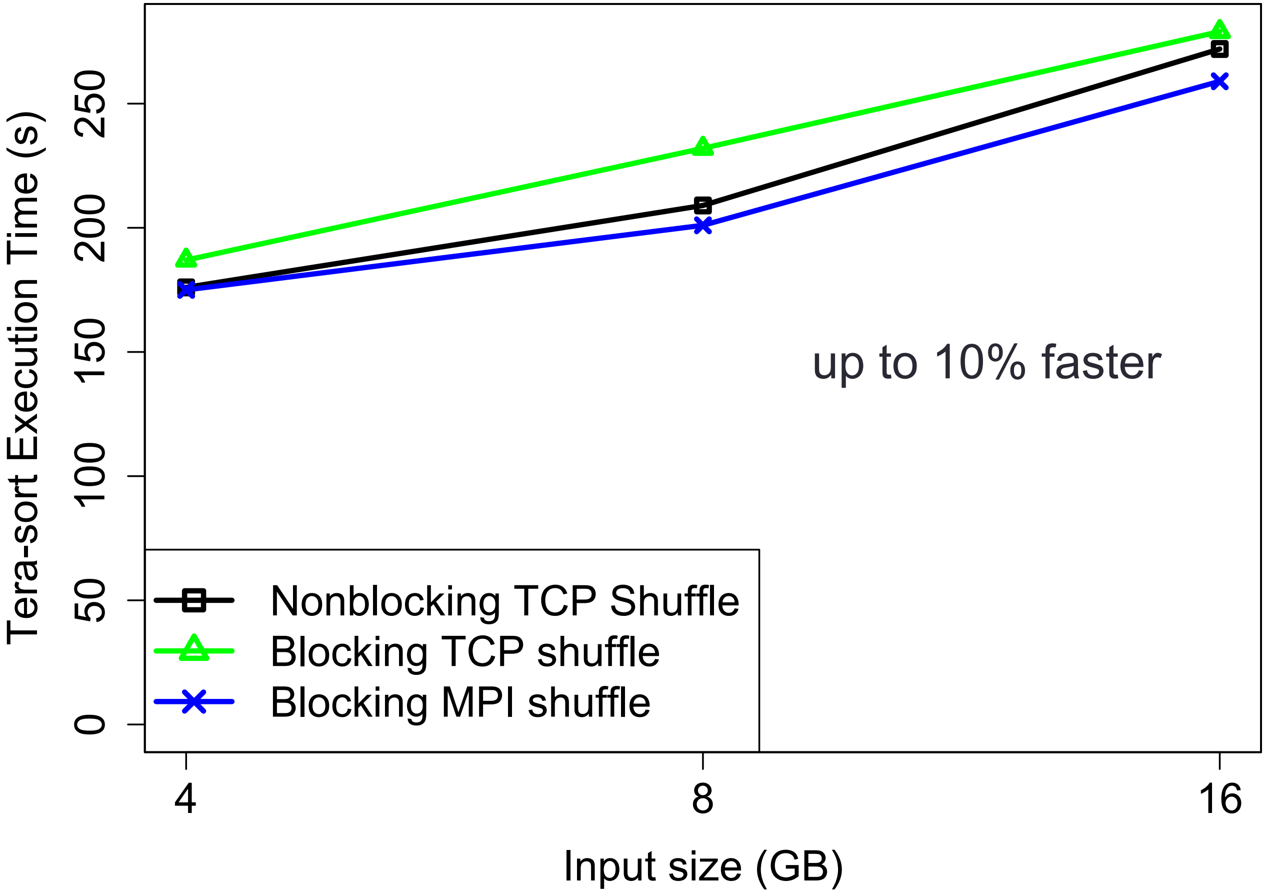
Cluster setup

- FX10 supercomputer
  - Sparc64 Ixfx 1.848 GHz (16 cores) & 32GB RAM
  - MPI over Tofu interconnection (5GB/s)
  - Central storage
- Hadoop setup
  - One master and many slaves
  - OpenJDK 7
  - HDFS is run on the central storage
- OpenMPI 1.6
  - Java MPI binding (Vega-Gisbert et al.)
  - MCA parameter: plm_ple_cpu_affinity = 0

Evaluation of JVM Reuse

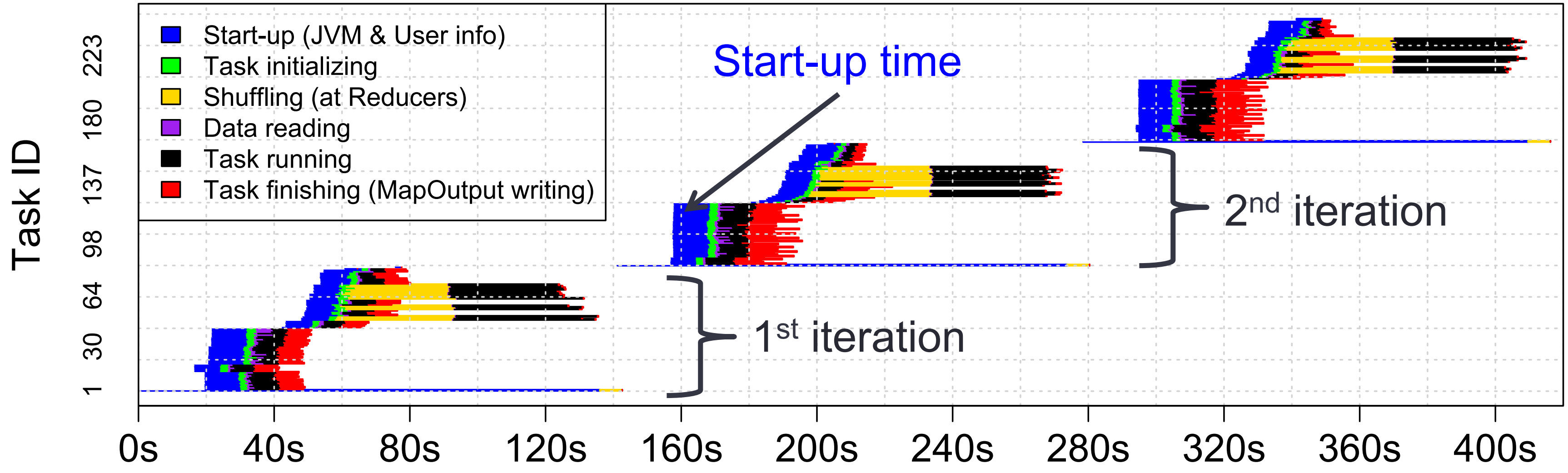
- MPI benefit
  - MPI vs. TCP/IP shuffling
  - Tera-sort job
    - Run on 32 FX10 nodes
    - 4-slot pool & -Xmx4096
- Start-up time
  - JVM Reuse vs. the original
  - PageRank iterative job
    - 400 GB wikipedia data
    - Run on 8 FX10 nodes
    - 6-slot pool & -Xmx4096

MPI vs. TCP/IP shuffling



Nonblocking : accept multi-connection at once
Blocking : accept one connection at once

Shorten start-up time (PageRank)

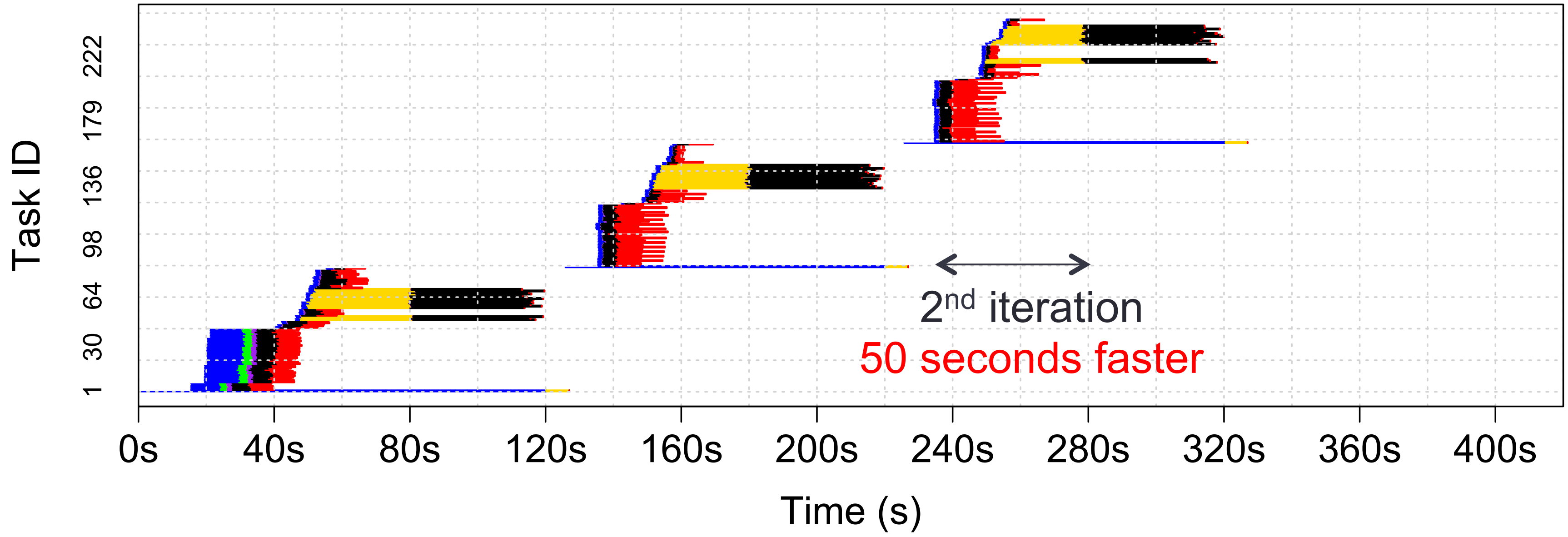


Start-up time

1st iteration

2nd iteration

Original Hadoop

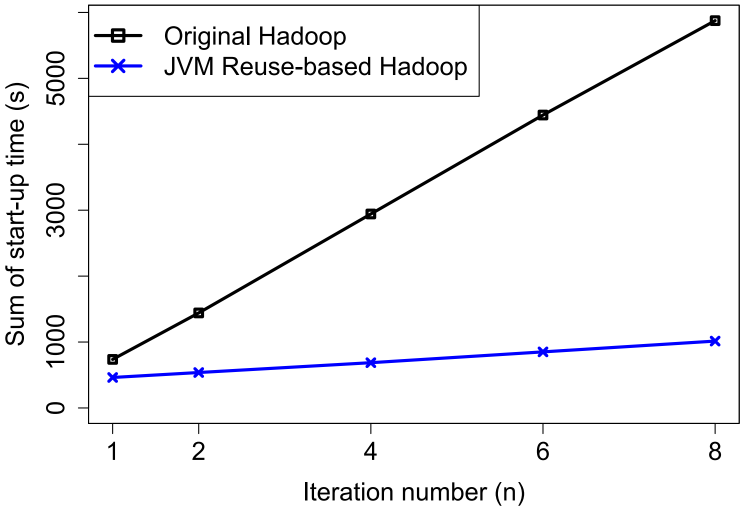


2nd iteration

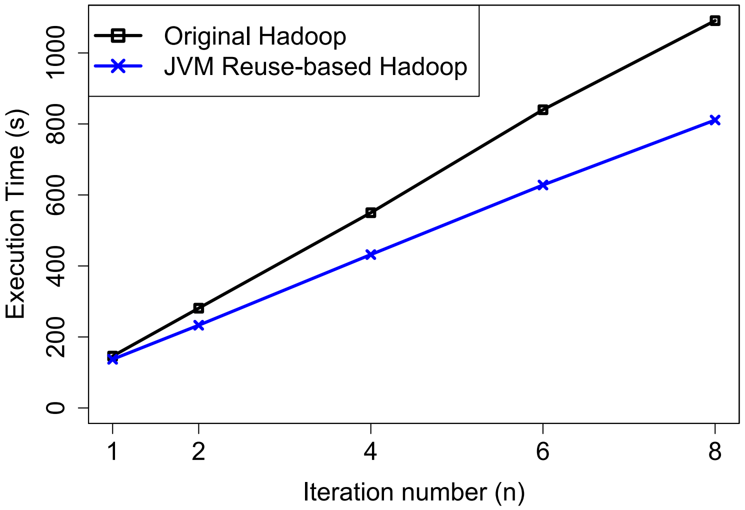
50 seconds faster

JVM Reuse Hadoop

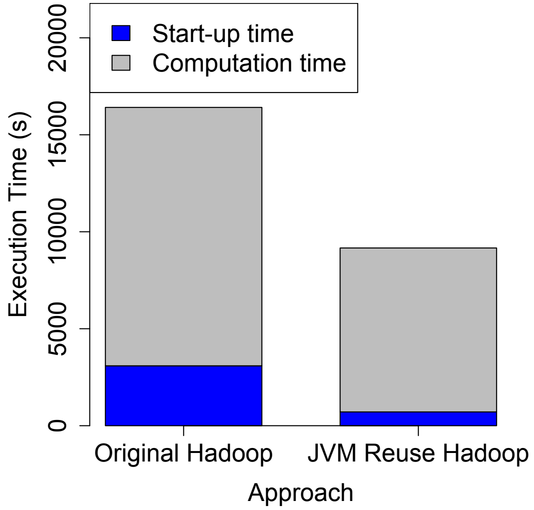
More iterations



Sum of start-up time



Execution time



Ratio

Related work

- M3R (VLDB 2012)
  - Also apply JVM Reuse to enable in-memory MapReduce
    - Not providing any optimization of JVM Reuse and its evaluation
  - Hadoop MapReduce (HMR) engine is written in X10
    - We keep the original HMR engine with minimum changes
- JVM Reuse in Hadoop v1 (2012)
  - Only for a single job
    - JVM processes are terminated after their job is completed
- Gerbil: MPI + YARN (CCGrid 2015)
  - Hadoop Yarn co-hosts MPI applications
  - Long start-up time and significant overhead
- DataMPI (IPDPS 2014)
  - Hadoop-like MapReduce implementation using MPI & C
- JVM-Bypass (IPDPS 2013)
  - C-based shuffling engine & RDMA supported
  - We focus on using MPI over Hadoop processes

Summary

- Improving Hadoop MapReduce performance on supercomputers

- Approach: JVM Reuse
  - Statically create JVM processes and dynamically allocate to Hadoop tasks
    - Enable efficient MPI communication on Hadoop
    - Shorten start-up time
  - Minimum changes of the original Hadoop

- Future work
  - JVM Reuse drawback
    - Affect CPU-bound tasks
  - Co-host MPI applications more efficiently
  - Full cleanup