

# Apache Nemo: A Framework for Building Distributed Dataflow Optimization Policies

Youngseok Yang<sup>1</sup> Jeongyoon Eo<sup>1</sup> Geon-Woo Kim<sup>2</sup> Joo Yeon Kim<sup>3</sup>  
Sanha Lee<sup>4</sup> Jangho Seo<sup>1</sup> Won Wook Song<sup>1</sup> Byung-Gon Chun<sup>1</sup>

<sup>1</sup>Seoul National University   <sup>2</sup>Viva Republica   <sup>3</sup>Samsung Electronics   <sup>4</sup>Naver Corp.

# Execution of Distributed WordCount

Application

```
dataSet.map(word => (word, 1))  
        .reduceByKey((l, r) => l + r)
```

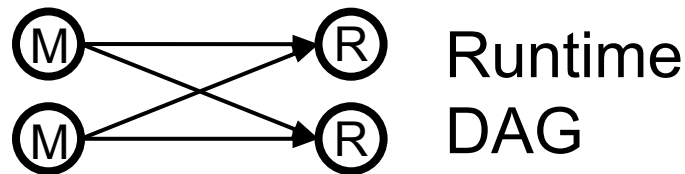
# Execution of Distributed WordCount

Application

```
dataSet.map(word => (word, 1))  
        .reduceByKey((l, r) => l + r)
```



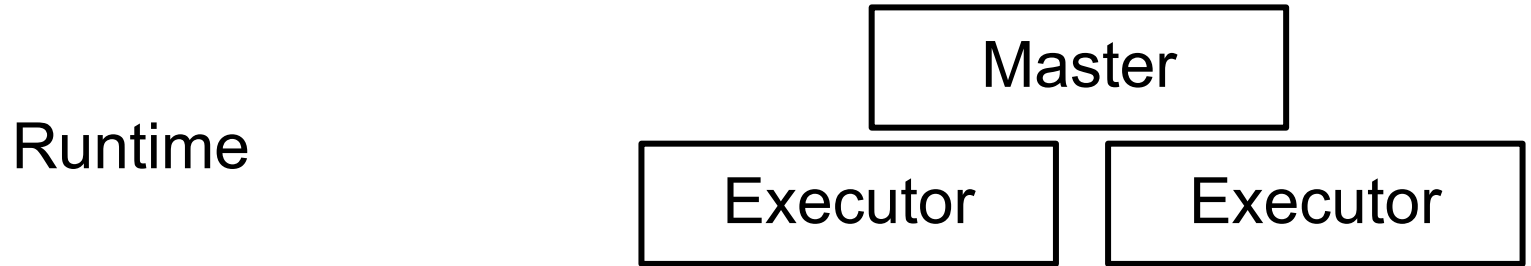
Compiler



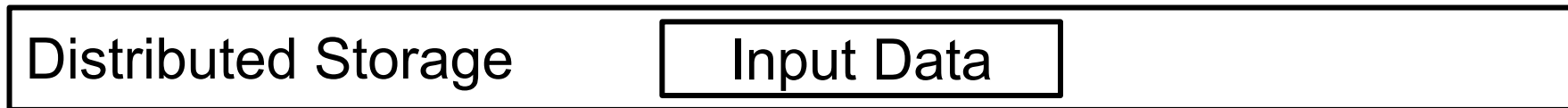
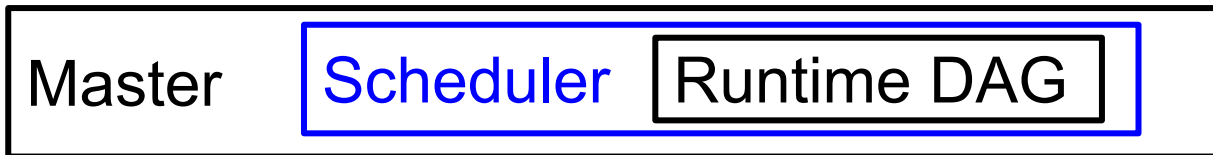
# Execution of Distributed WordCount

Application

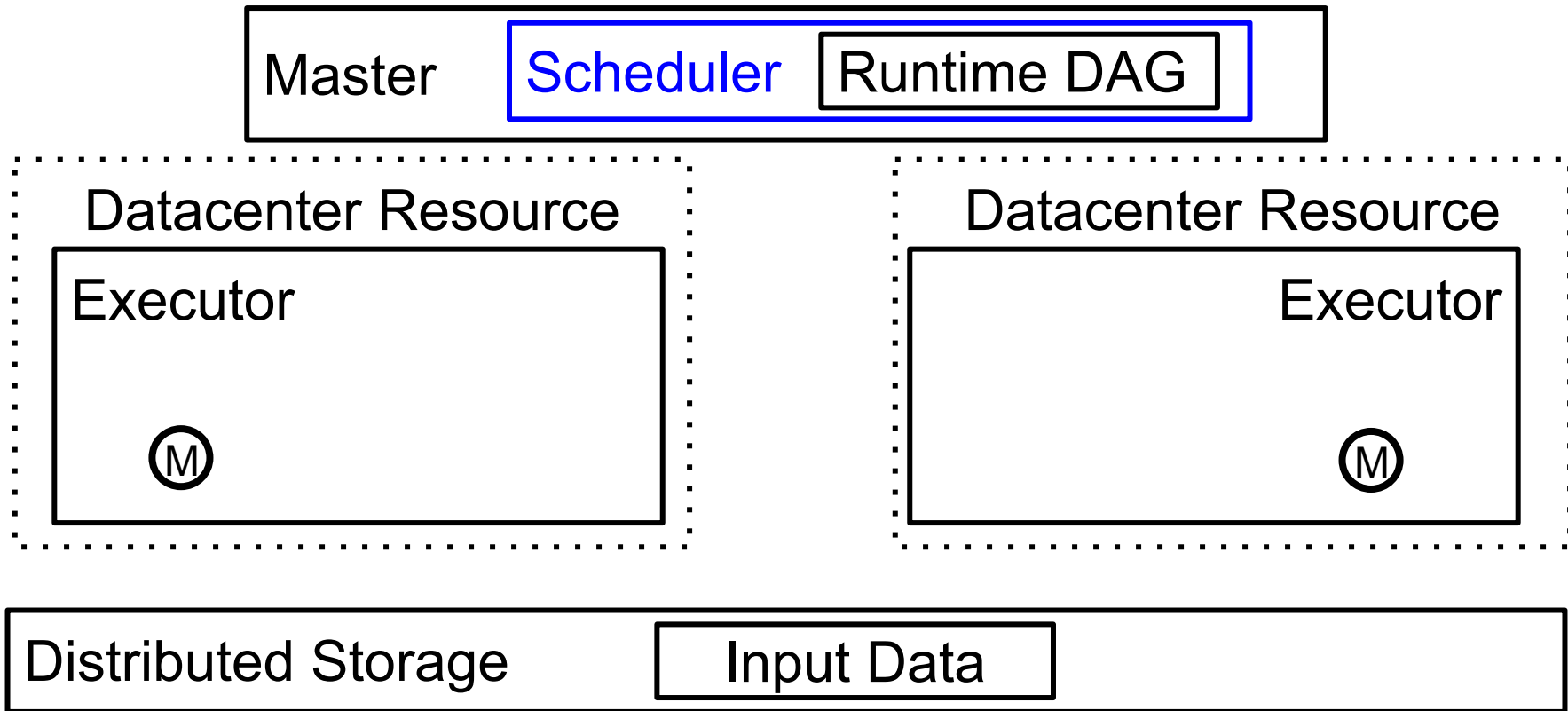
```
dataSet.map(word => (word, 1))  
        .reduceByKey((l, r) => l + r)
```



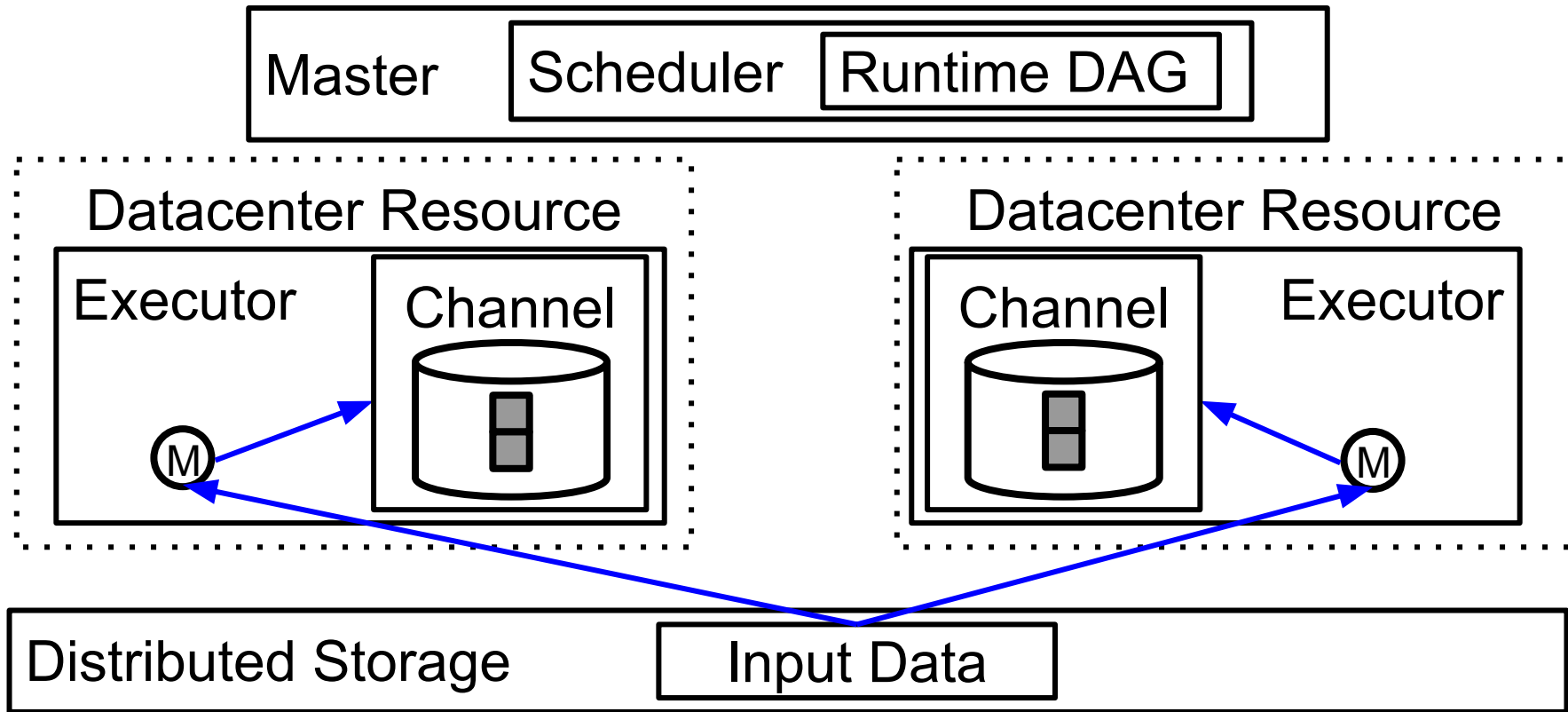
# Execution of Distributed WordCount



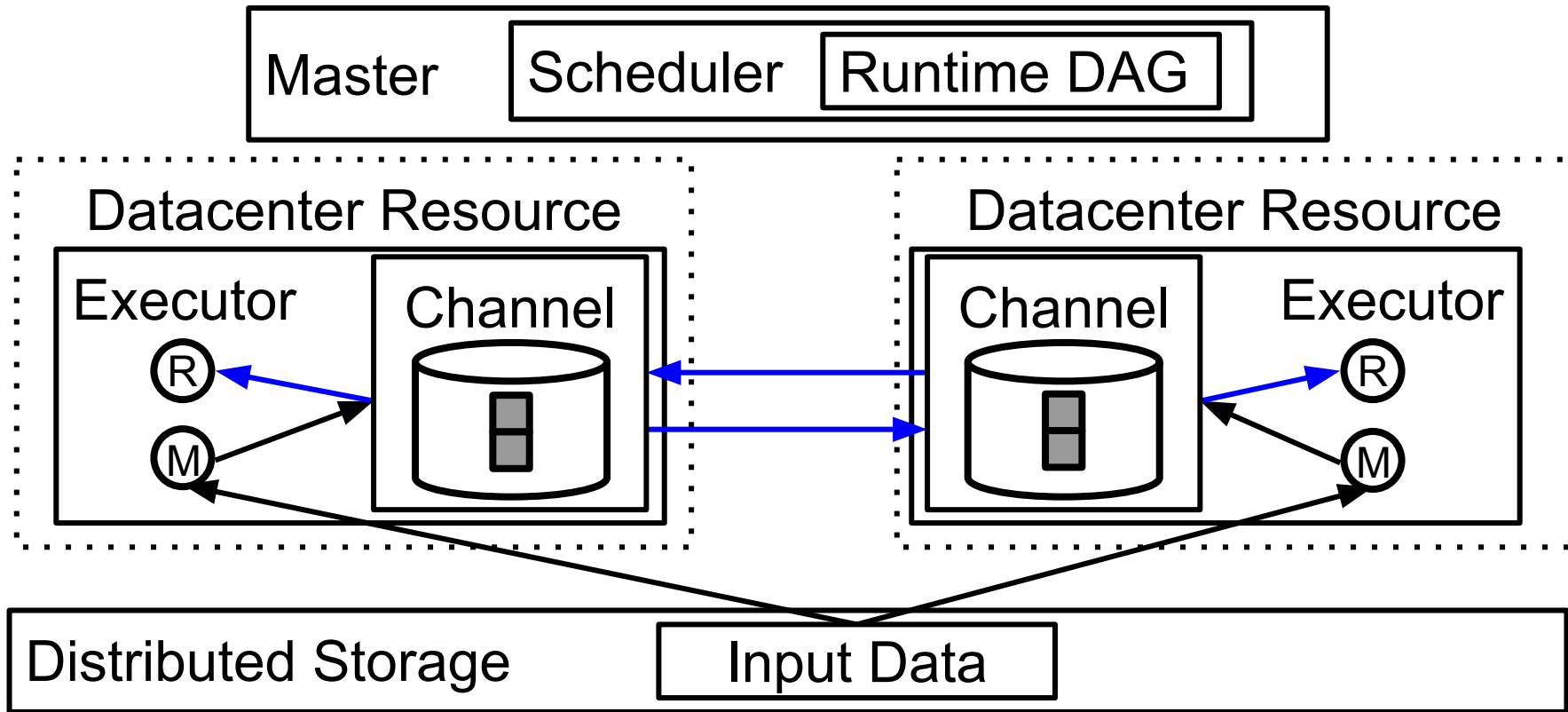
# Execution of Distributed WordCount



# Execution of Distributed WordCount

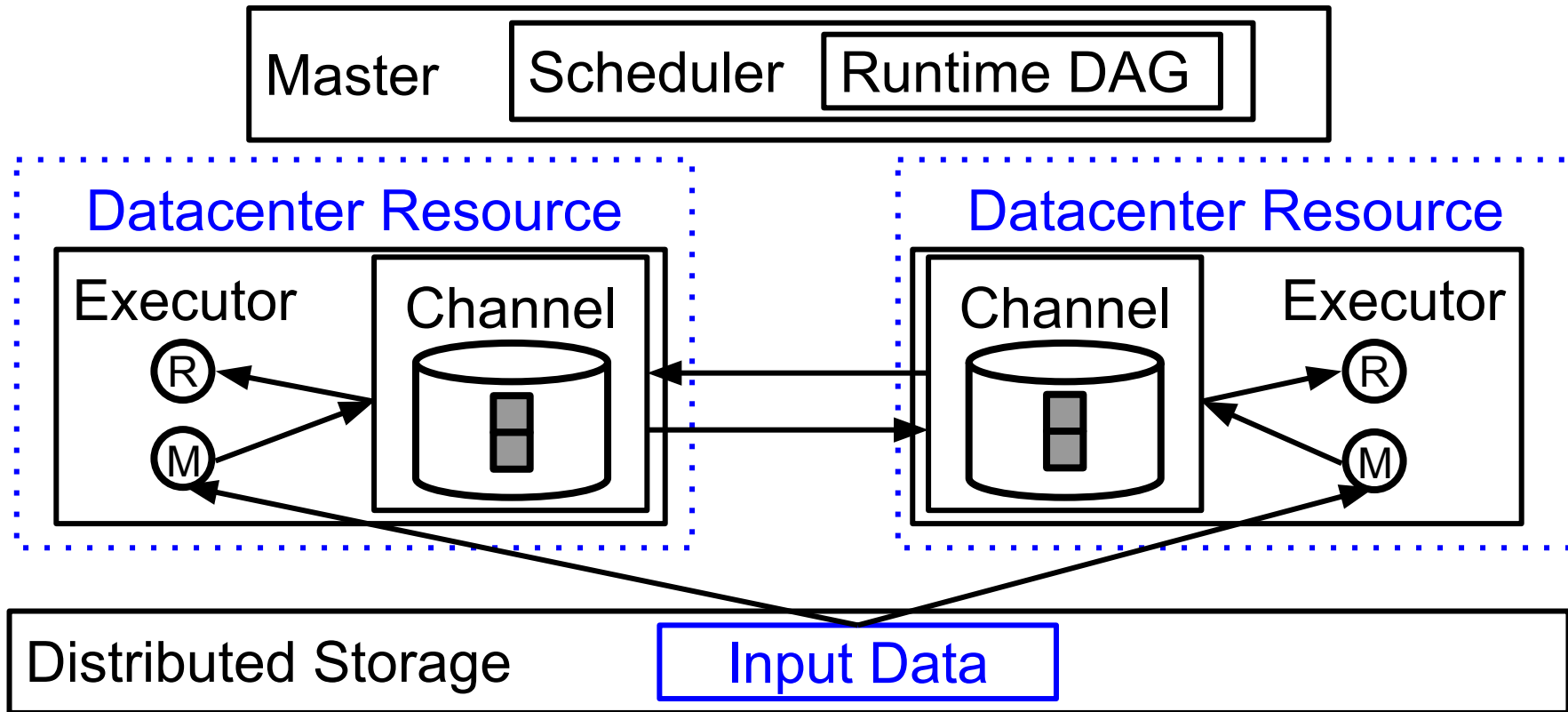


# Execution of Distributed WordCount





# Our Observations: Resources & Data



# Trend: Diverse Characteristics

## **Datacenter Resources**

Geographically  
-distributed

Cheap transient

## **Input Data**

Large-scale

Skewed

# In This Talk (See Paper for Others)

## **Datacenter Resources**

Geographically  
-distributed

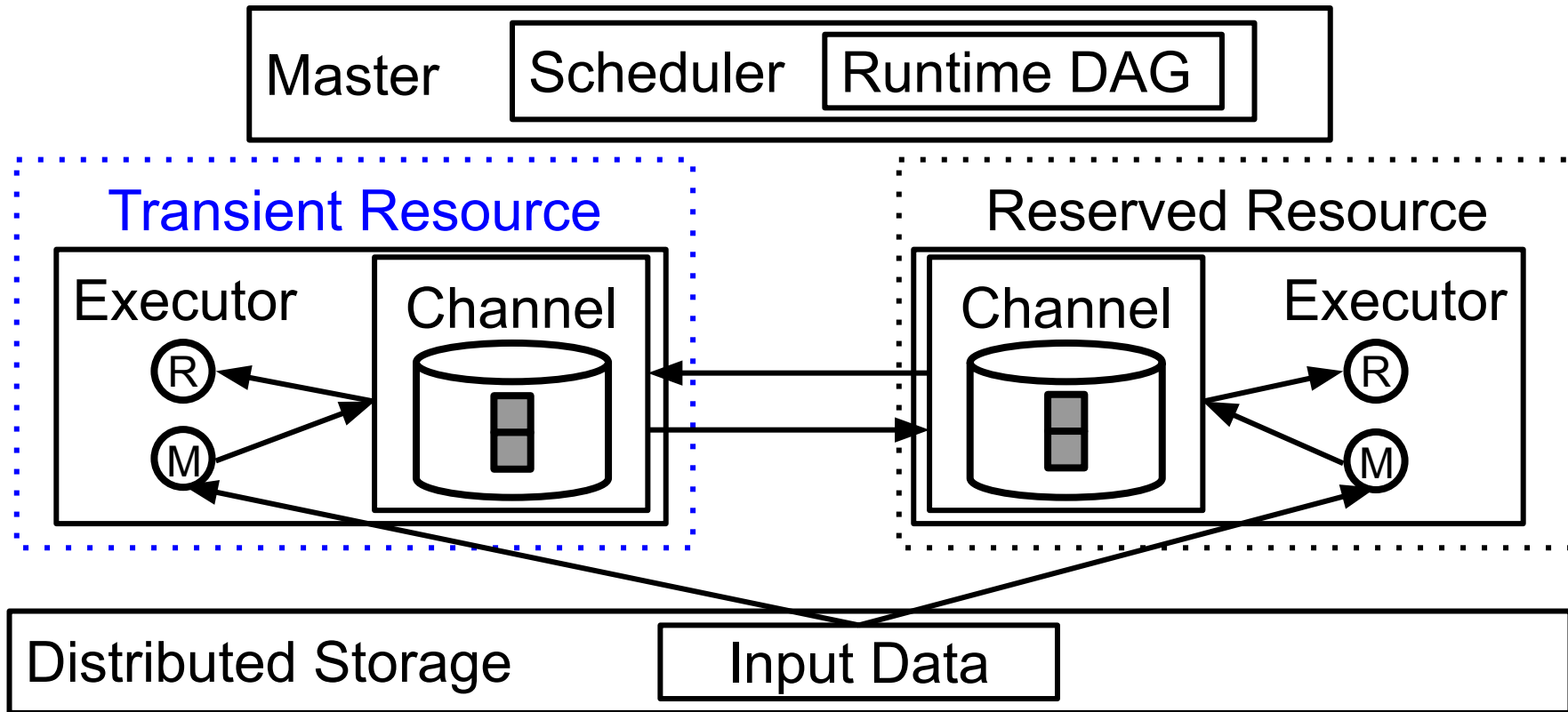
**(1) Cheap transient**

## **Input Data**

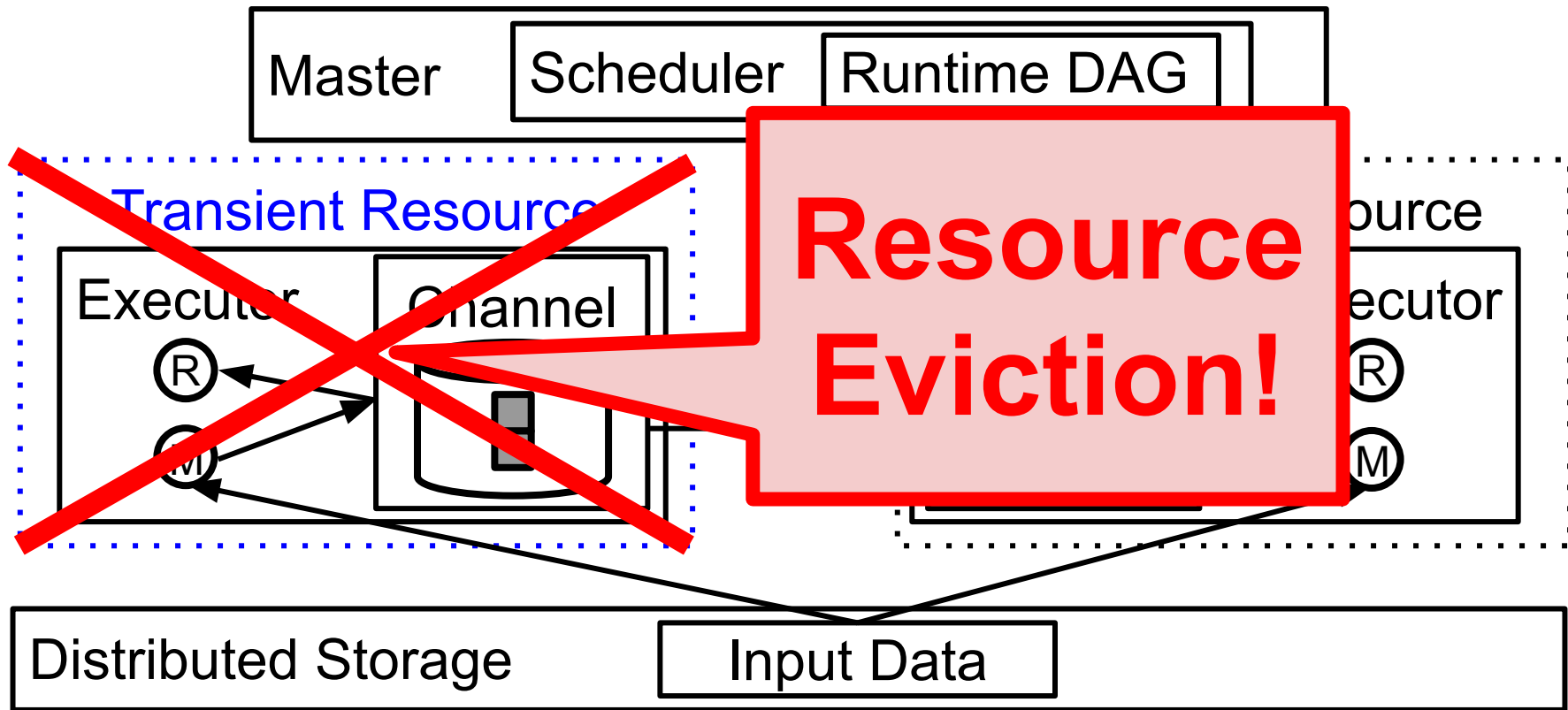
**(2) Large-scale**

Skewed

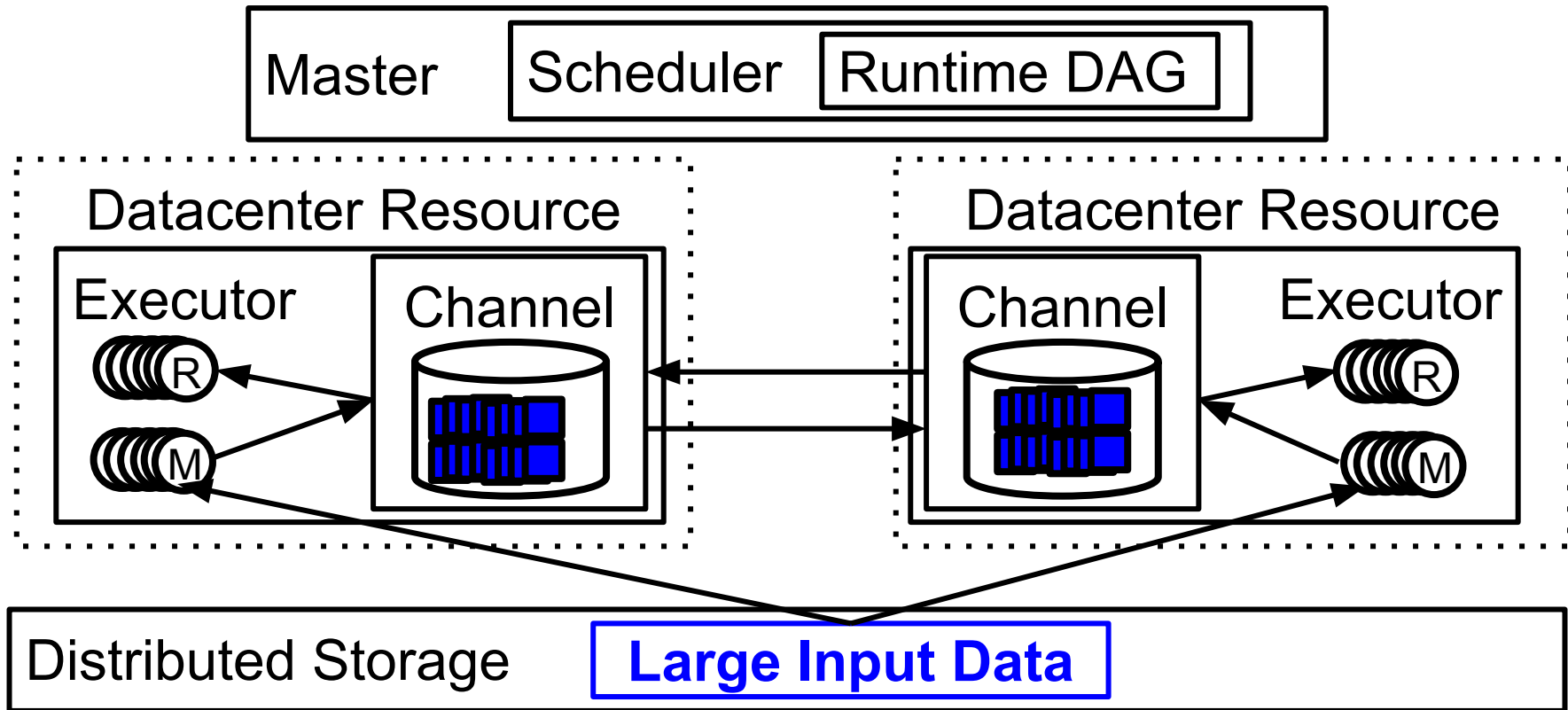
# (1) Cheap Transient Resources



# (1) Cheap Transient Resources

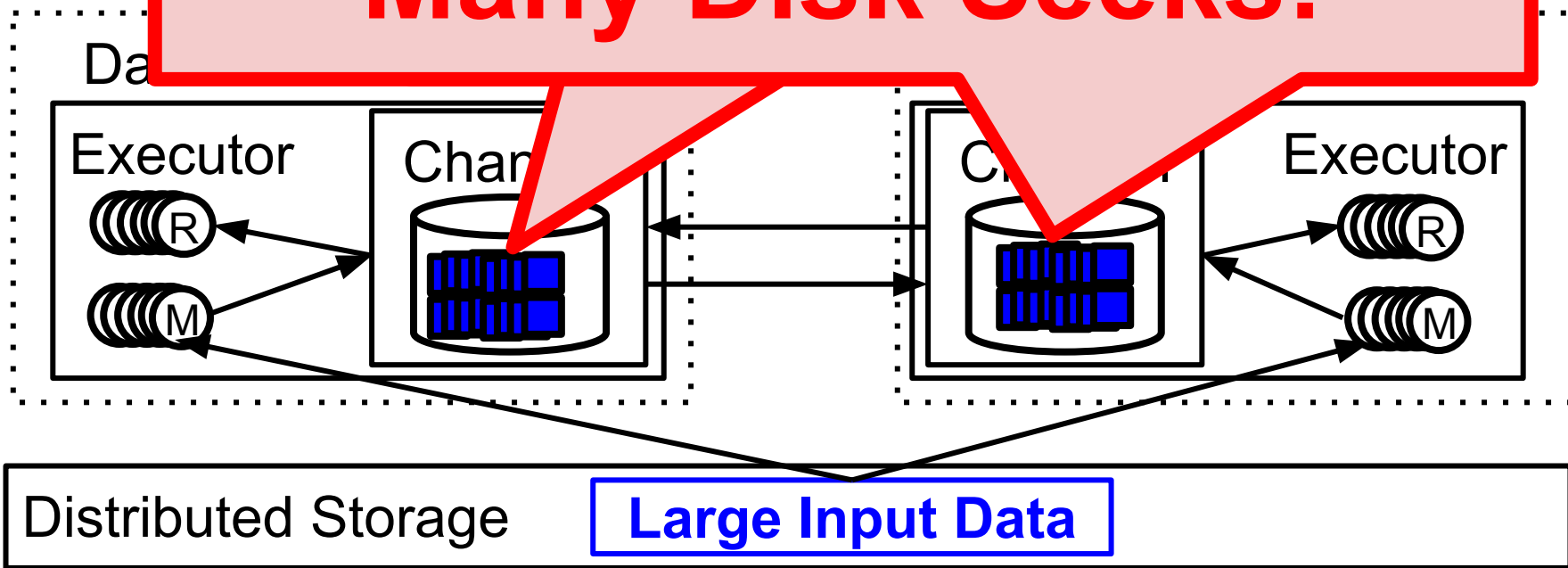


## (2) Large-scale Data Shuffle



## (2) Large-scale Data Shuffle

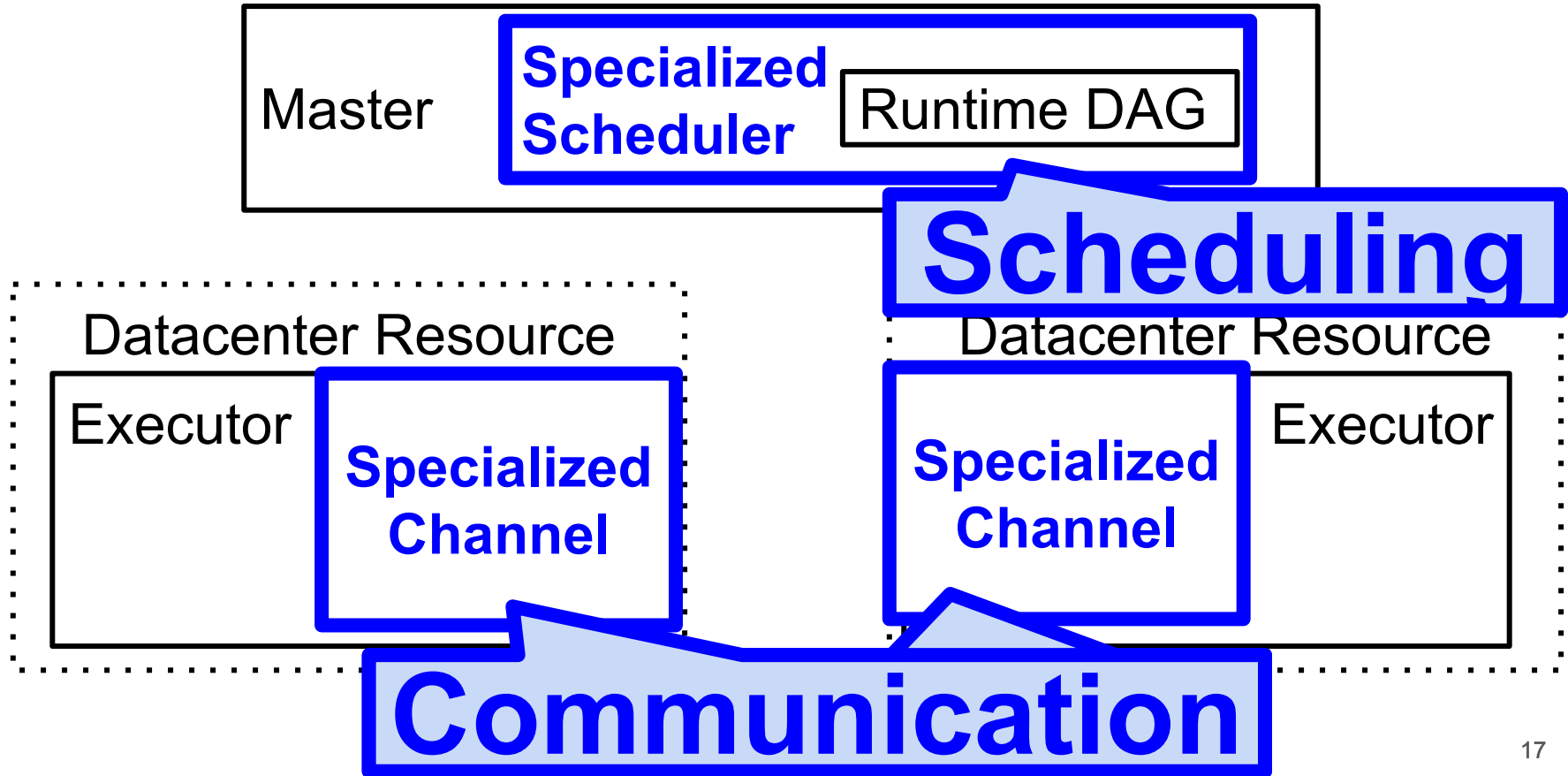
**Many Disk Seeks!**



How to optimize  
distributed execution?



# Existing Approach: Direct Specialization



# Direct Specialization: **Hard** to Ensure...

## **(1) Correctness**

Optimized execution produces the same results

## **(2) Reusability**

Single specialization across different applications

## **(3) Composability**

Combine multiple specialized optimizations

Our goal:  
Make it *easy* to optimize  
distributed execution

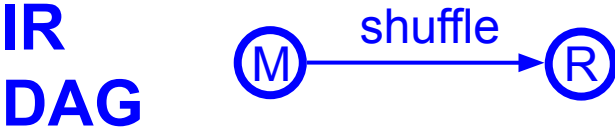
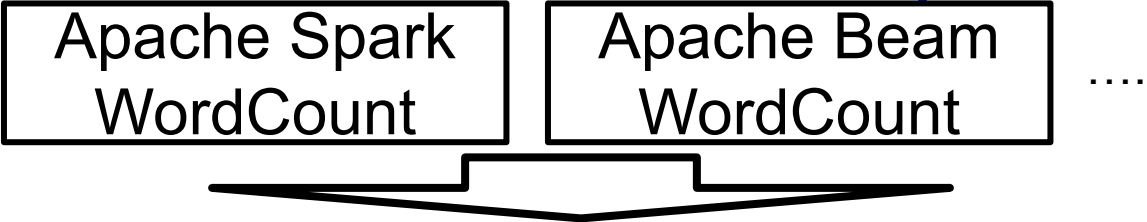
# Our Idea: Intermediate Representation (IR)

Apache Spark  
WordCount

Apache Beam  
WordCount

....

# Our Idea: Intermediate Representation (IR)



# Our Idea: Intermediate Representation (IR)

Apache Spark  
WordCount

Apache Beam  
WordCount

....

IR  
DAG

(M)

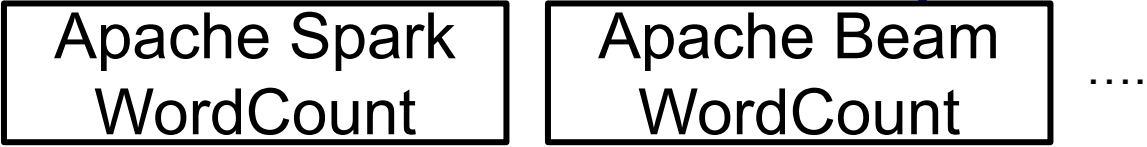
shuffle

(R)

Optimization Pass  
f: irdag  $\rightarrow$  irdag'

**Easy! (Think Functions)**

# Our Idea: Intermediate Representation (IR)

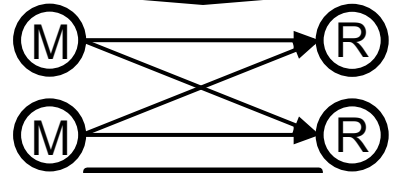


**IR  
DAG**



**Optimization Pass**  
f: irdag → irdag'

**Runtime  
DAG**



**Optimized**



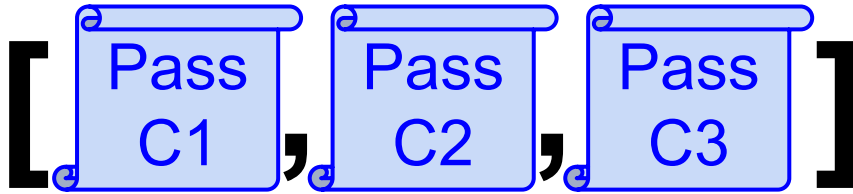
# Overall Workflow of Apache Nemo



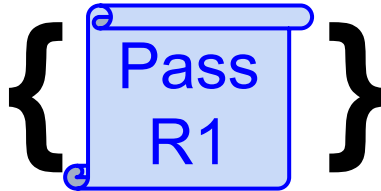
# Nemo User Job Submission (Easy!)

Application

e.g., Spark/Beam Application



**Compile-time Passes  
(List)**



**Run-time Passes  
(Set)**

# Nemo Applies Compile-time Passes

Application

---

**Nemo Compiler**

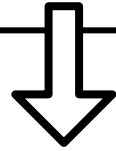
---

# Nemo Applies Compile-time Passes

Application

```
graph TD; Application[Application] --> irdag[irdag]; NemoCompiler[Nemo Compiler];
```

The diagram illustrates the compilation process. A box labeled 'Application' is positioned above a horizontal line. A large downward-pointing arrow indicates the flow of data from the application to the 'irdag' (Intermediate Representation Directed Acyclic Graph) below the line. The 'Nemo Compiler' is positioned to the right of the arrow, indicating its role in this process. A second horizontal line is located below the 'irdag' label.

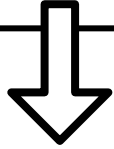


**Nemo Compiler**

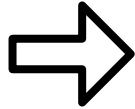
irdag

# Nemo Applies Compile-time Passes

Application



irdag



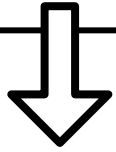
Pass  
C1

Nemo Compiler

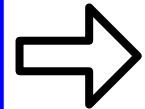
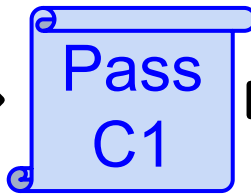
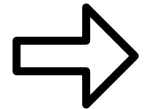
**Check correctness of  
the output IR DAG**

# Nemo Applies Compile-time Passes

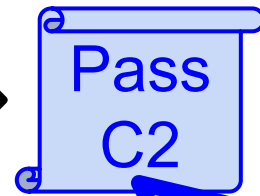
Application



irdag



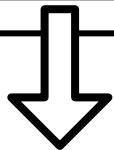
Nemo Compiler



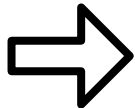
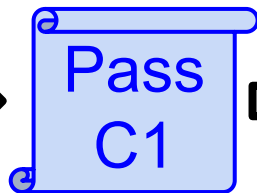
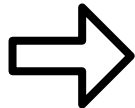
**Check correctness &  
Check conflict with C1**

# Nemo Applies Compile-time Passes

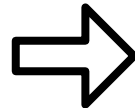
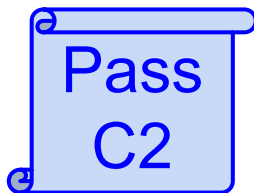
Application



irdag



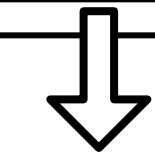
**Nemo Compiler**



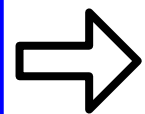
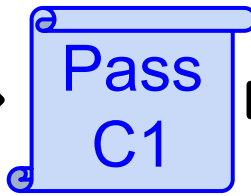
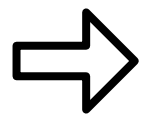
**Check correctness &  
Check conflict with C1+C2**

# Nemo Applies Compile-time Passes

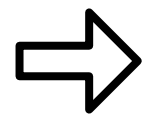
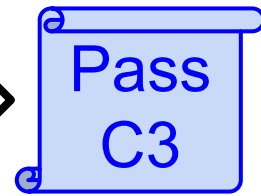
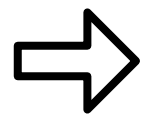
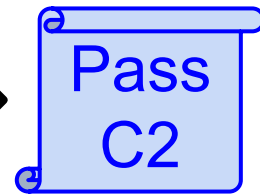
Application



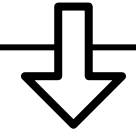
irdag



**Nemo Compiler**



irdag'  
(optimized)



**Nemo Runtime**

Runtime  
DAG'  
(optimized)

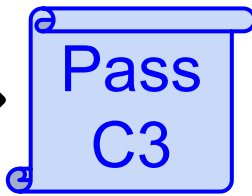
**If all checks pass**

# Nemo Applies Compile-time Passes

Application

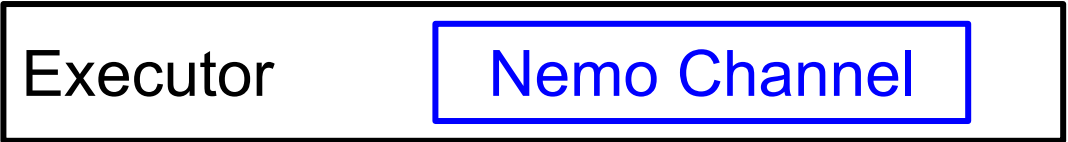
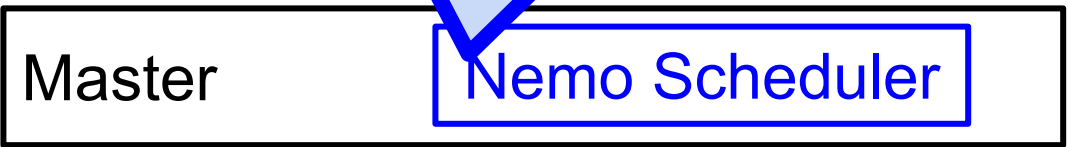
**Reflects the optimizations**

Compiler



irdag'  
(optimized)

Nemo Runtime



Runtime DAG'  
(optimized)



# Nemo Applies Run-time Passes

Application

Nemo Core

**During job execution**

Message

Nemo Runtime

Master

Nemo Scheduler

Executor

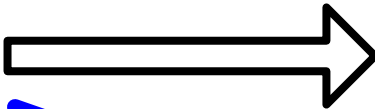
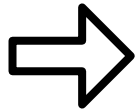
Nemo Channel

# Nemo Applies Run-time Passes

Application

Nemo Compiler

irdag'  
(optimized)



irdag''  
(optimized more)

Message

Master

Executor

Ne

Nemo Compiler

**Correctness &  
Conflict checks**

# Nemo Applies Run-time Passes

Application

**Updates lazily  
for correctness**

Message

Nemo Run

Master

Nemo Scheduler

Executor

Nemo Channel

irdag"  
(optimized more)

Runtime  
DAG"

(optimized more)

# Example Apache Nemo Optimization Passes

# What A Pass Does

While traversing the input IR DAG,

(1) Inserts **Utility Vertices**

(2) Annotates **Execution Properties**

# What A Pass Does

**Applies a specific function**

with the input DAG,

- (1) Inserts **Utility Vertices**
- (2) Annotates **Execution Properties**

# What A Pass Does

While traversing the input IR DAG,

(1) Inserts **Utility Vertices**

(2) Annotates **Execution Properties**



**Scheduling/Communication**

# Passes We Implemented & Evaluated

GeoDistResourcePass

LargeShufflePass

TransientResourcePass

SkewCTPass

SkewRTPass

SkewSamplingPass



# In This Talk (See Paper for Others)

GeoDistResourcePass

**(1) LargeShufflePass**

**(2) TransientResourcePass**

SkewCTPa

SkewRTPa

SkewSampling

**Both are  
compile  
time passes**

# (1) LargeShufflePass: Goal

## **Avoid on-disk data shuffle!**

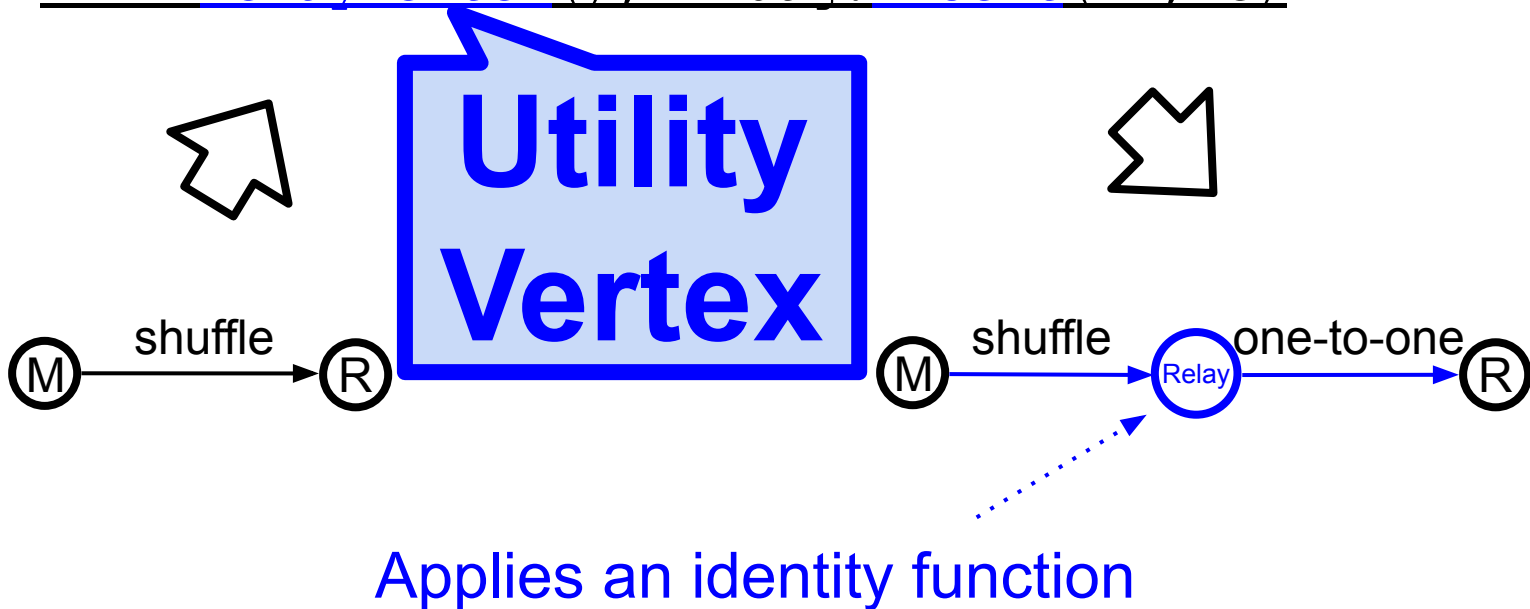
- Shuffle data in memory
- Write shuffled data to disks
- Read from disks sequentially

Related Work: Riffle (EuroSys18)

# (1) LargeShufflePass: Algorithm

for each shuffle edge  $e$  in irdag:

$rv = \text{RelayVertex}(), \text{irdag.insert}(rv, e)$



# (1) LargeShufflePass: Algorithm

## Execution Properties

```
for each s
```

```
  rv = Rela
```

```
  rv.inEdge.set(DataFlow.Push,
```

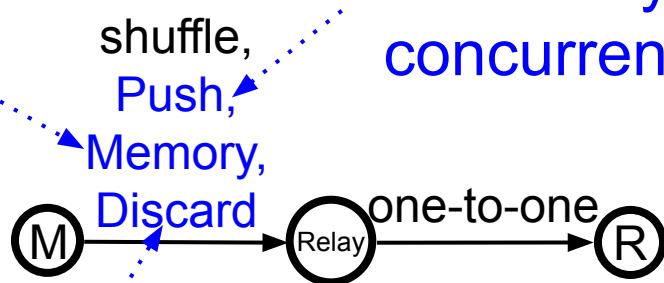
```
    DataStore.Memory, Persistence.Discard)
```



In-memory  
shuffle



Execute M  
and Relay  
concurrently



Do not persist data in memory

# (1) LargeShufflePass: Algorithm

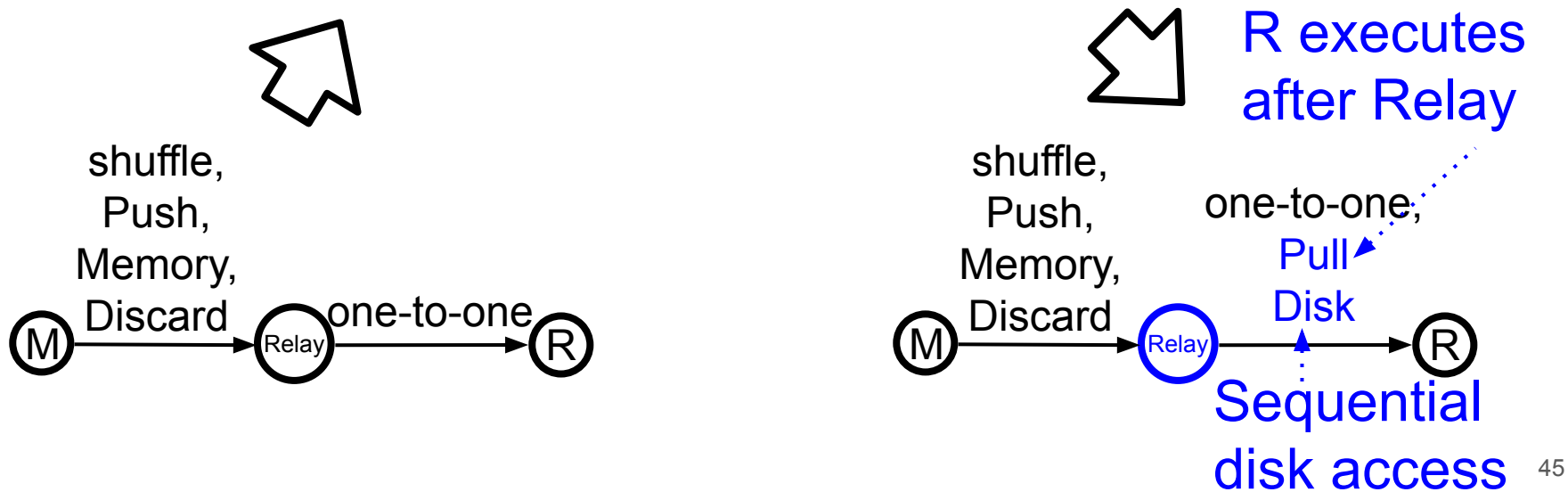
for each shuffle edge  $e$  in irdag:

```
rv = RelayVertex(), irdag.insert(rv, e)
```

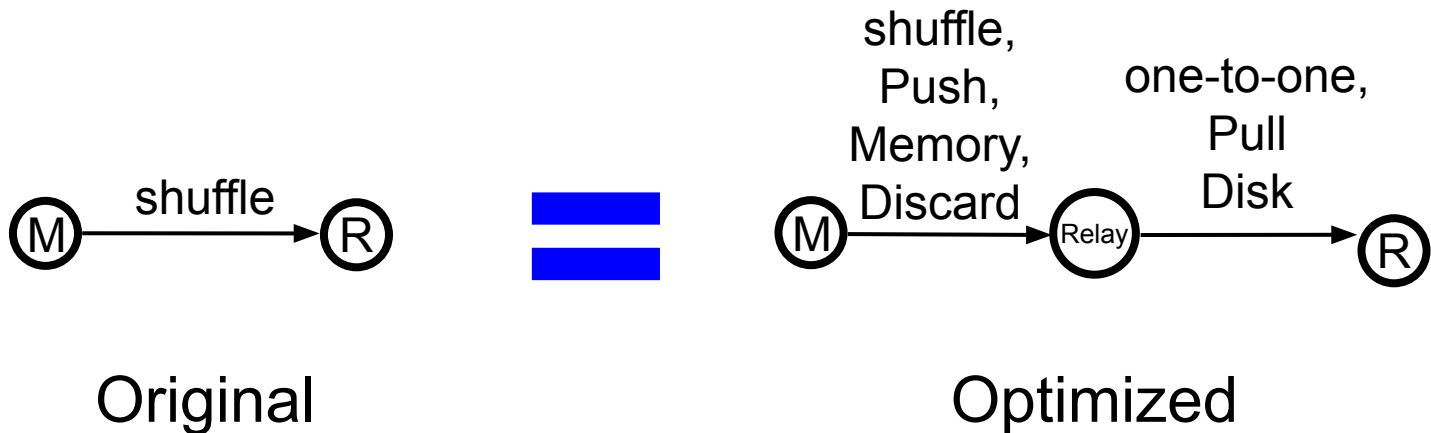
```
rv.inEdge.set(DataFlow.Push,
```

```
  DataStore.Memory, Persistence.Discard)
```

```
rv.outEdge.set(DataFlow.Pull, DataStore.Disk)
```

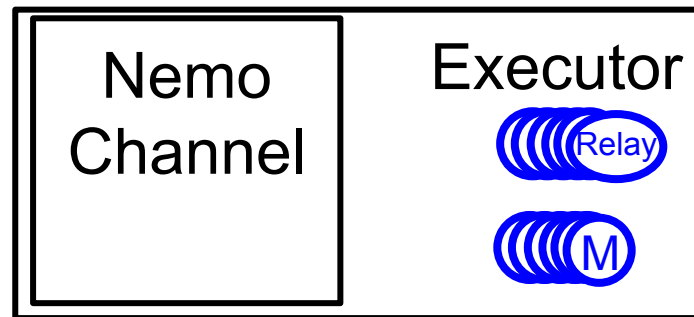
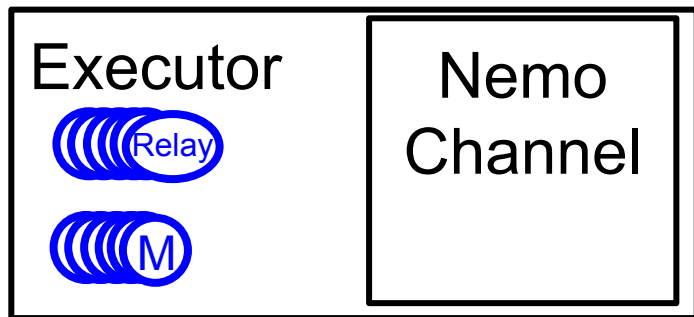
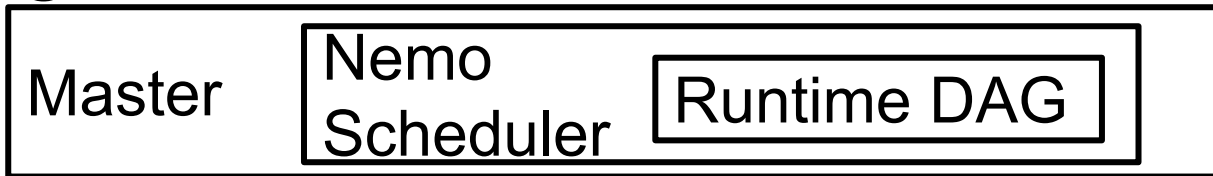


# (1) LargeShufflePass: Correctness



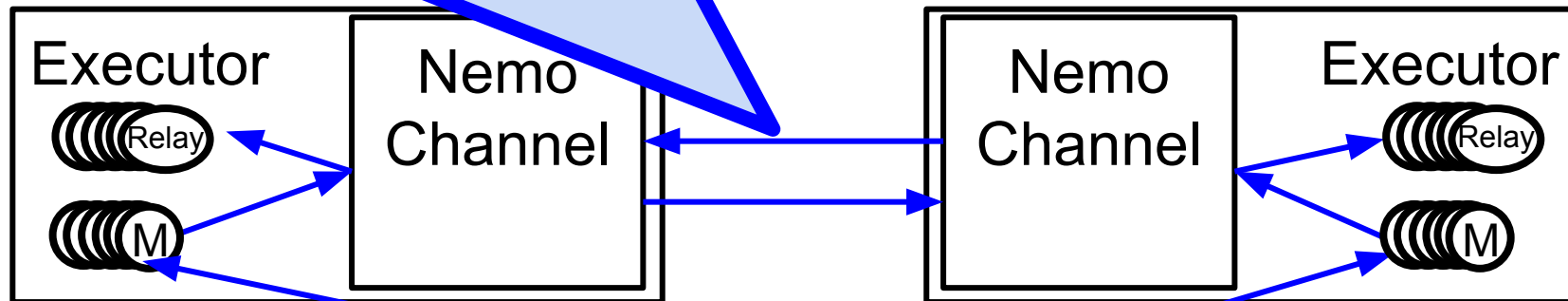
**Equivalent final outputs!**

# (1) LargeShufflePass: Runtime Execution



# (1) LargeShufflePass: Runtime Execution

## Memory+Discard Shuffle

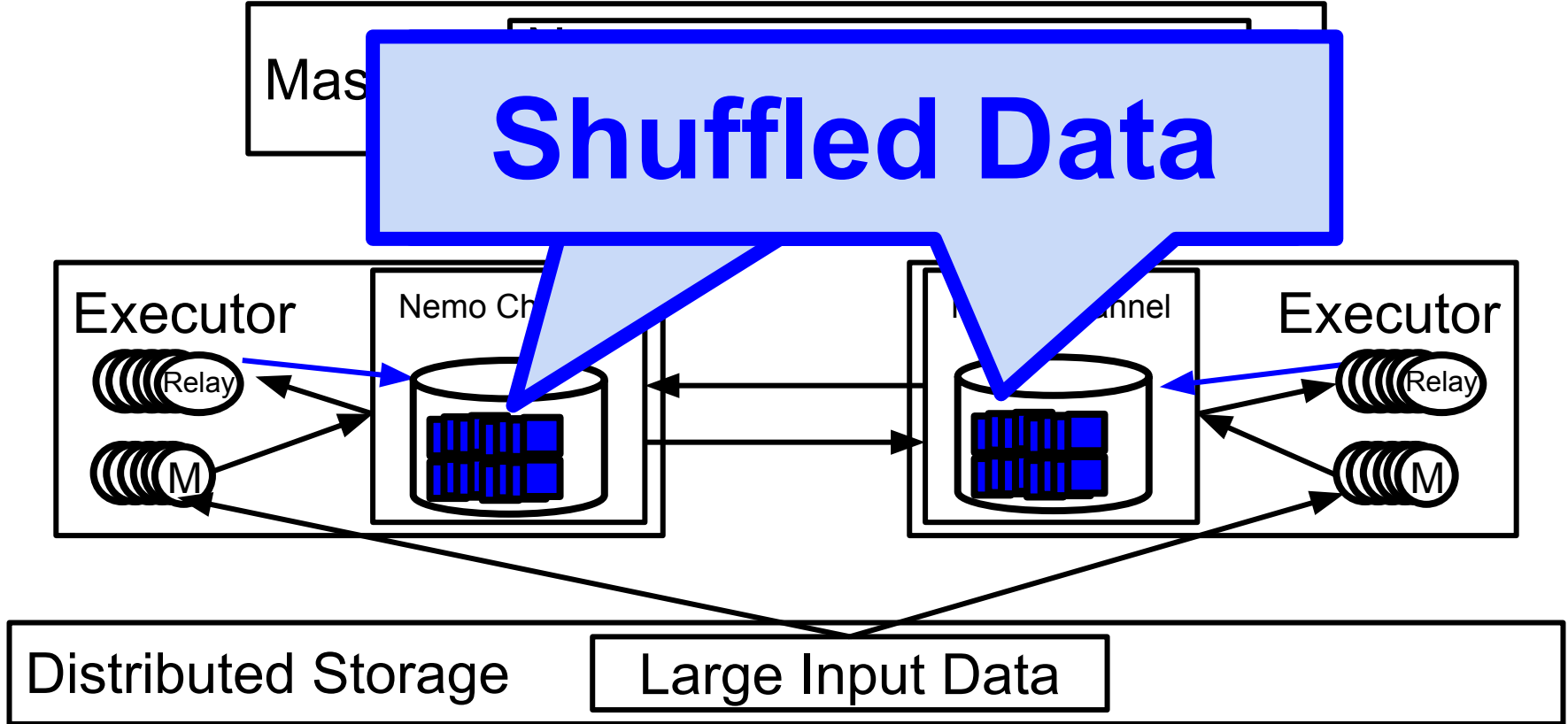


Distributed Storage

Large Input Data

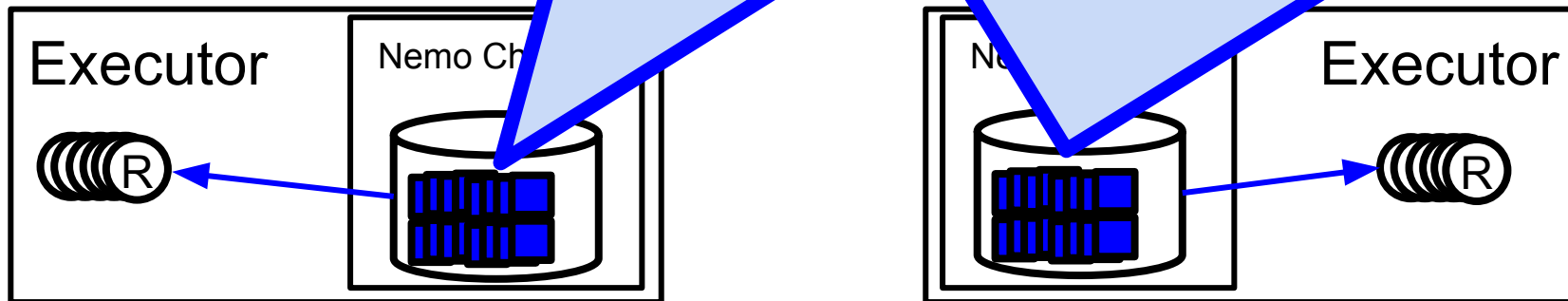


# (1) LargeShufflePass: Runtime Execution



# (1) LargeShufflePass: Runtime Execution

## Sequential Disk Access



Distributed Storage

Large Input Data

## (2) TransientResourcePass: Goal

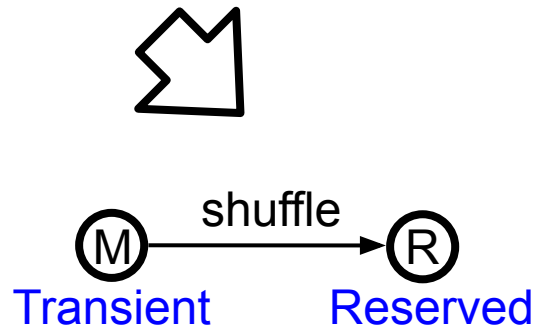
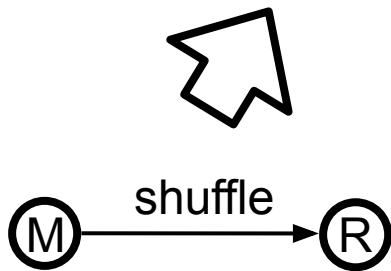
### **Minimize recomputations!**

- Place on Transient/Reserved judiciously
- Push data from Transient to Reserved

Related Work: Pado (EuroSys17)

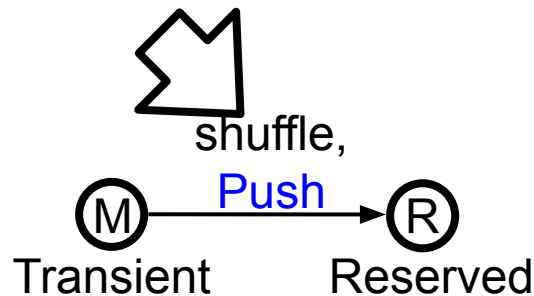
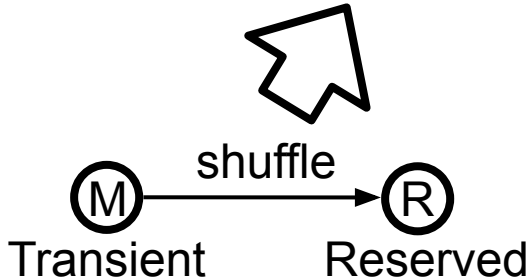
## (2) TransientResourcePass: Algorithm

```
for each vertex v in topologicalSort(irdag):  
  if (containsShuffle(v.inEdges) || ...):  
    v.set(ResourcePriority.Reserved)  
  else:  
    v.set(ResourcePriority.Transient)
```

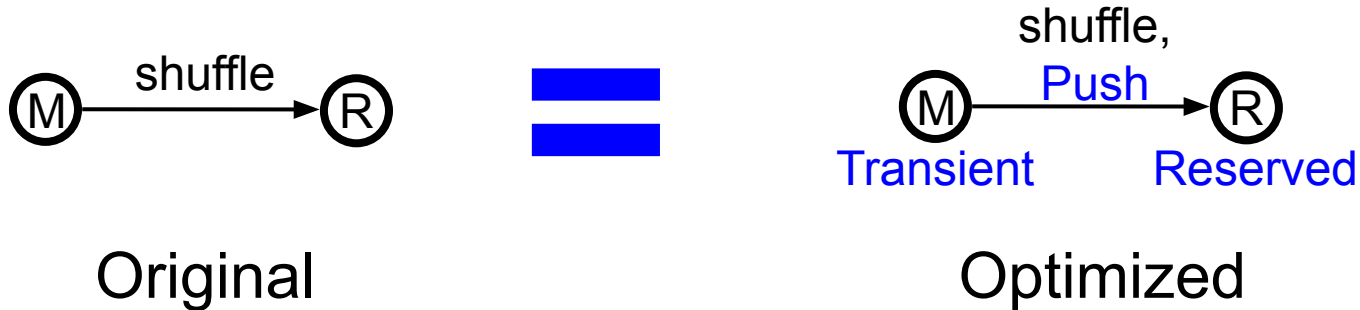


## (2) TransientResourcePass: Algorithm

```
for each vertex v in topologicalSort(irdag):  
  if (containsShuffle(v.inEdges) || ...):  
    v.set(ResourcePriority.Reserved)  
  else:  
    v.set(ResourcePriority.Transient)  
for e in v.inEdges:  
  if fromTransientToReserved(e.src, e.dst):  
    e.set(DataFlow.Push)
```

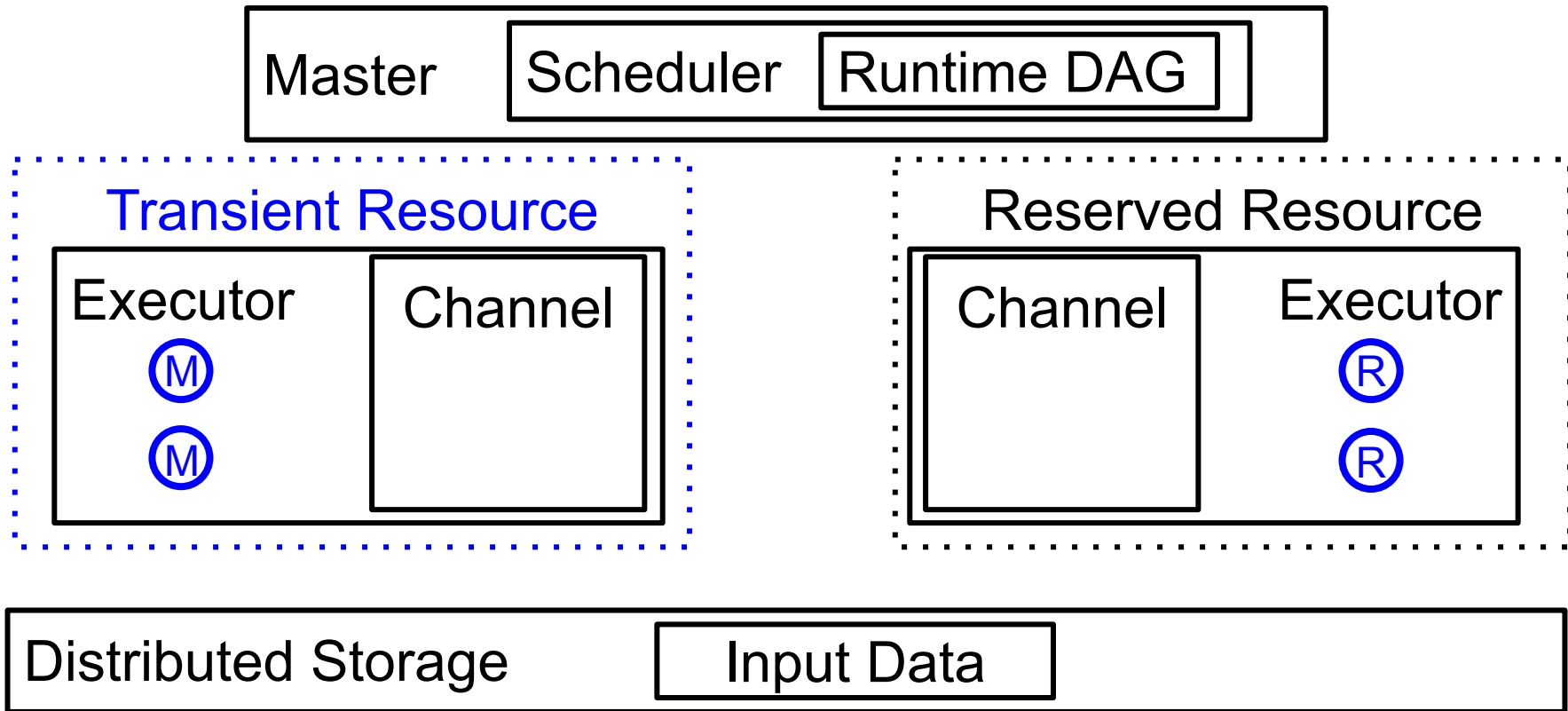


## (2) TransientResourcePass: Corectness

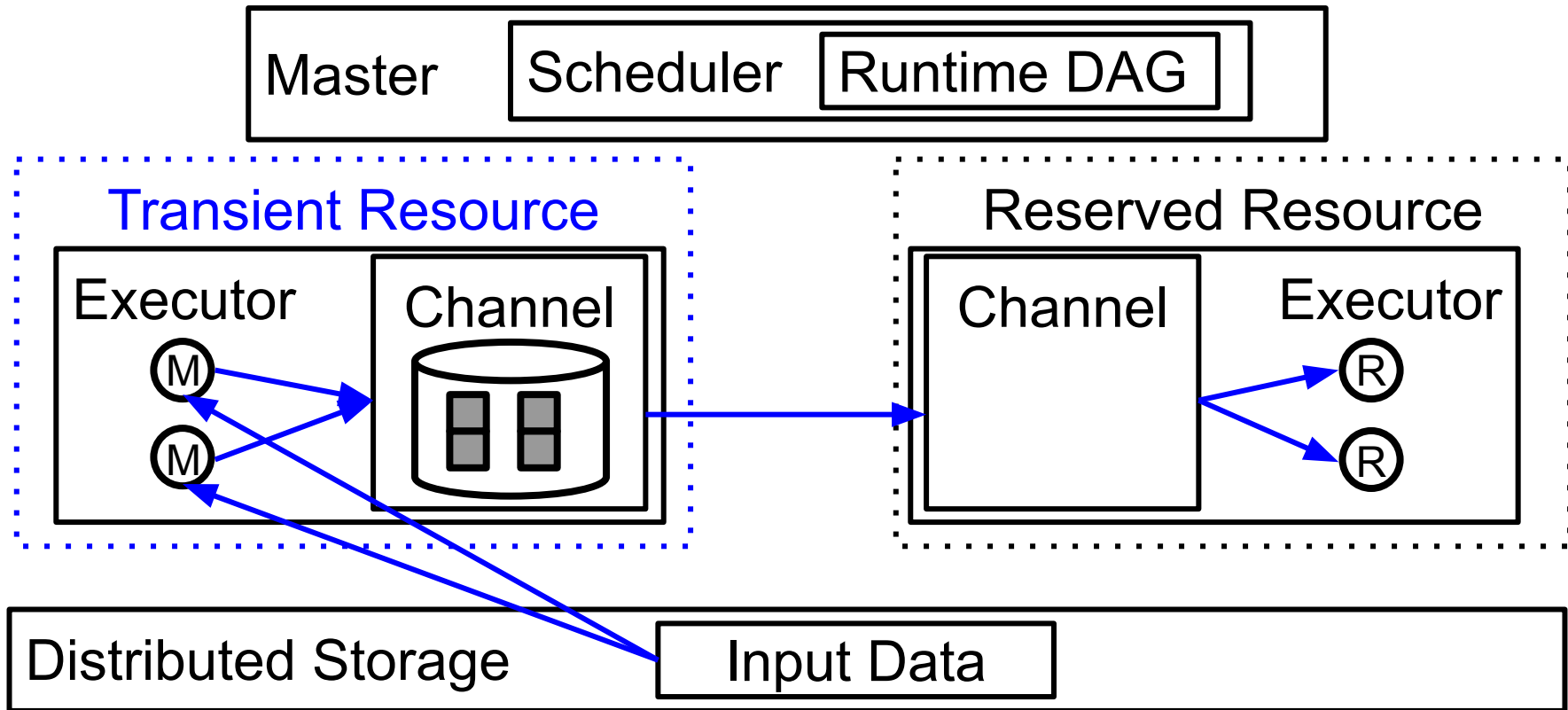


**Equivalent final outputs!**

## (2) TransientResourcePass: Runtime



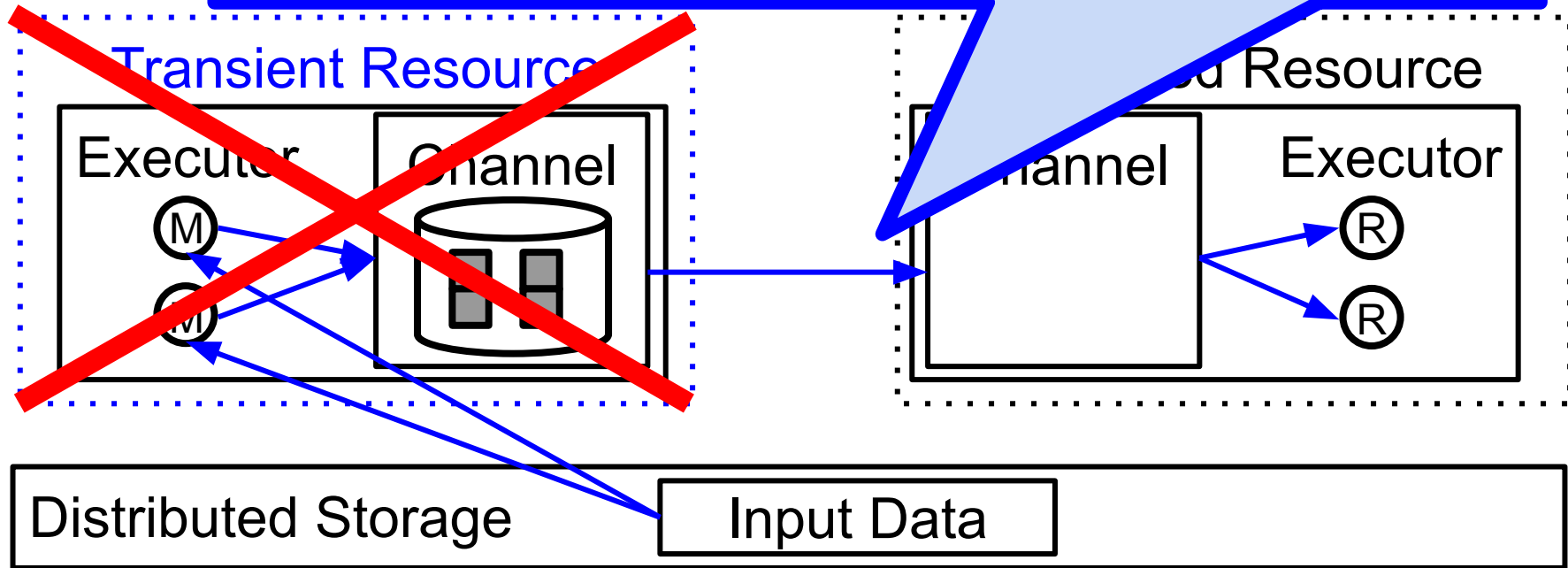
## (2) TransientResourcePass: Runtime



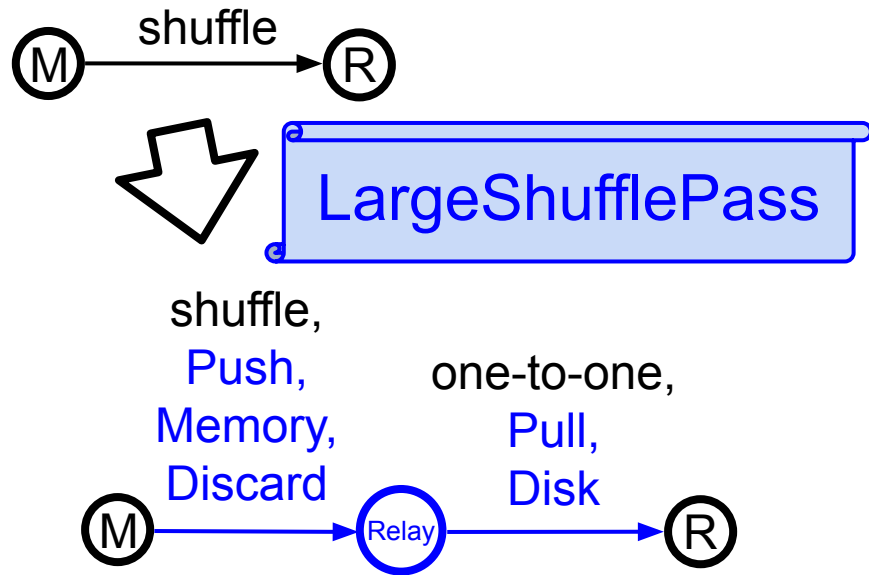


## (2) TransientResourcePass: Runtime

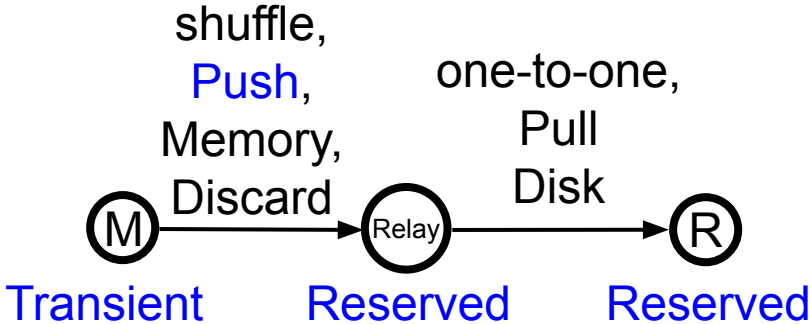
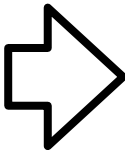
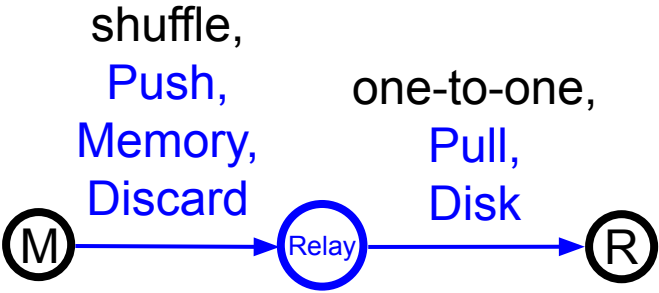
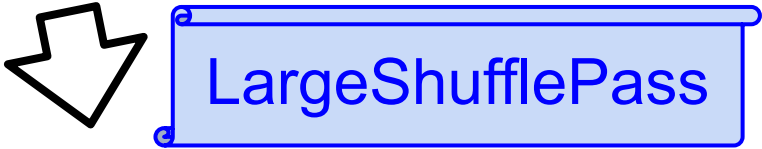
**Moves data out quickly**



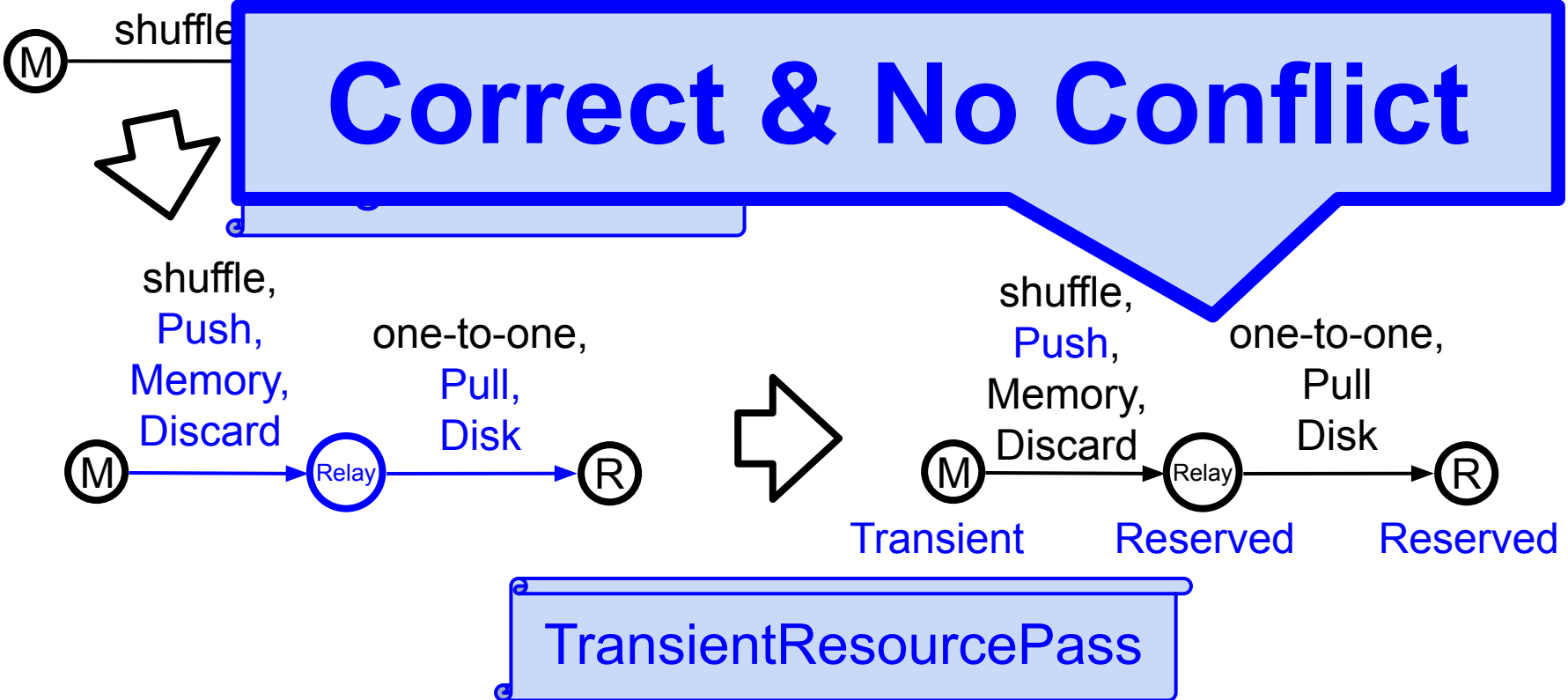
# LargeShufflePass+TransientResourcePass



# LargeShufflePass+TransientResourcePass



# LargeShufflePass+TransientResourcePass



# Implementation & Evaluation

# Nemo Implementation

- Open source (<https://nemo.apache.org>)
- 32K lines of Java code, including its own runtime
- Good integration with other Apache Big Data projects
  - Supported applications



- Supported cluster resource managers



(thanks to  REEF )

# What We Evaluated: Scenarios

Large Data Shuffle

Transient Resources

Geo-distributed Resources

Skewed Data

Large Shuffle on Transient Resources

Skewed Data on Geo-distributed Resources

Large Shuffle with Skewed Data

# In This Talk (See Paper for Others)

**Large Data Shuffle**

**Transient Resources**

Geo-distributed Resources

Skewed Data

**Large Shuffle on Transient Resources**

Skewed Data on Geo-distributed Resources

Large Shuffle with Skewed Data



# What We Evaluated: Systems

Apache Nemo

Apache Spark: A state-of-the-art runtime

Pado (EuroSys17): Specialized for transient resources

Hurricane (EuroSys18): Specialized for data skew

Iridium (SIGCOMM15): Specialized for geo analytics

# In This Talk (See Paper for Others)

**Apache Nemo**

**Apache Spark: A state-of-the-art runtime**

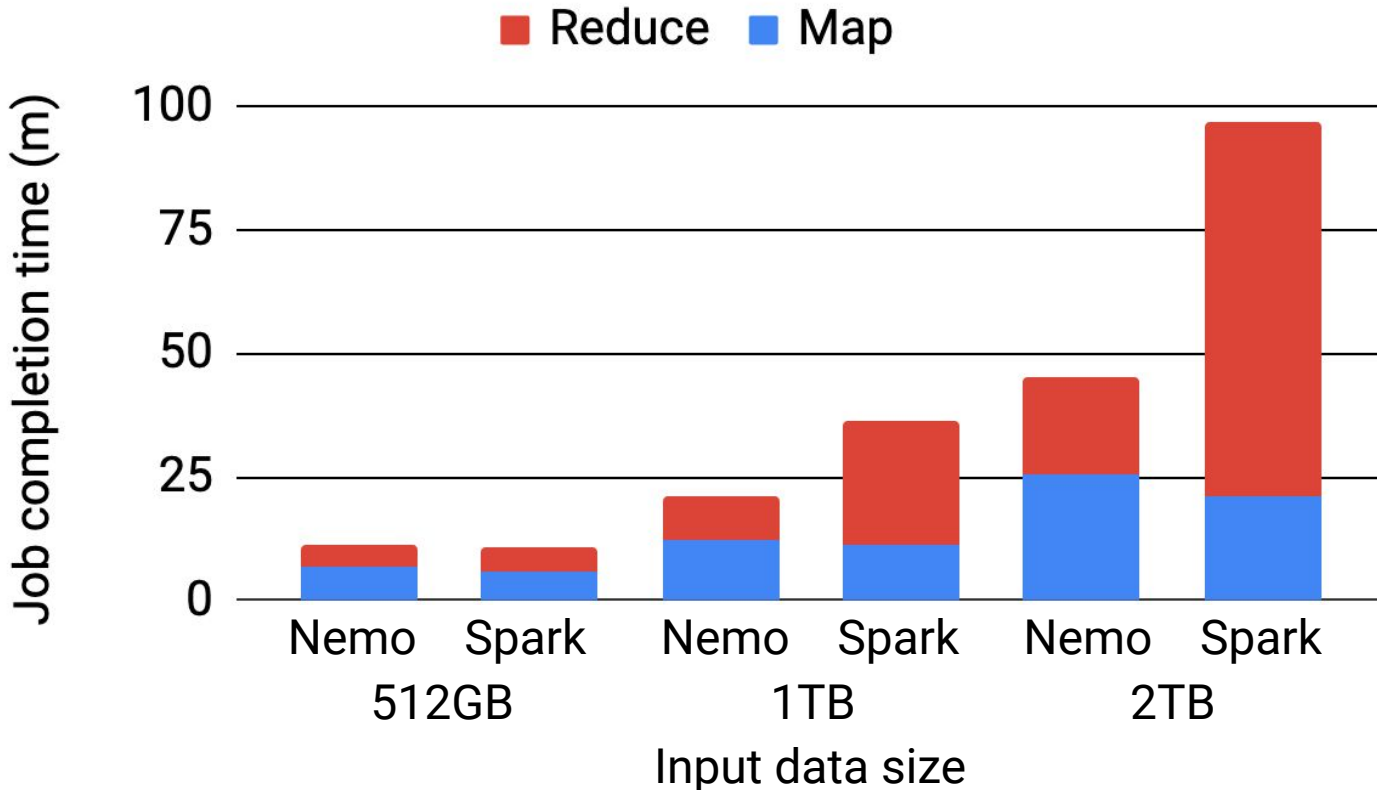
**Pado (EuroSys17): Specialized for transient resources**

Hurricane (EuroSys18): Specialized for data skew

Iridium (SIGCOMM15): Specialized for geo analytics

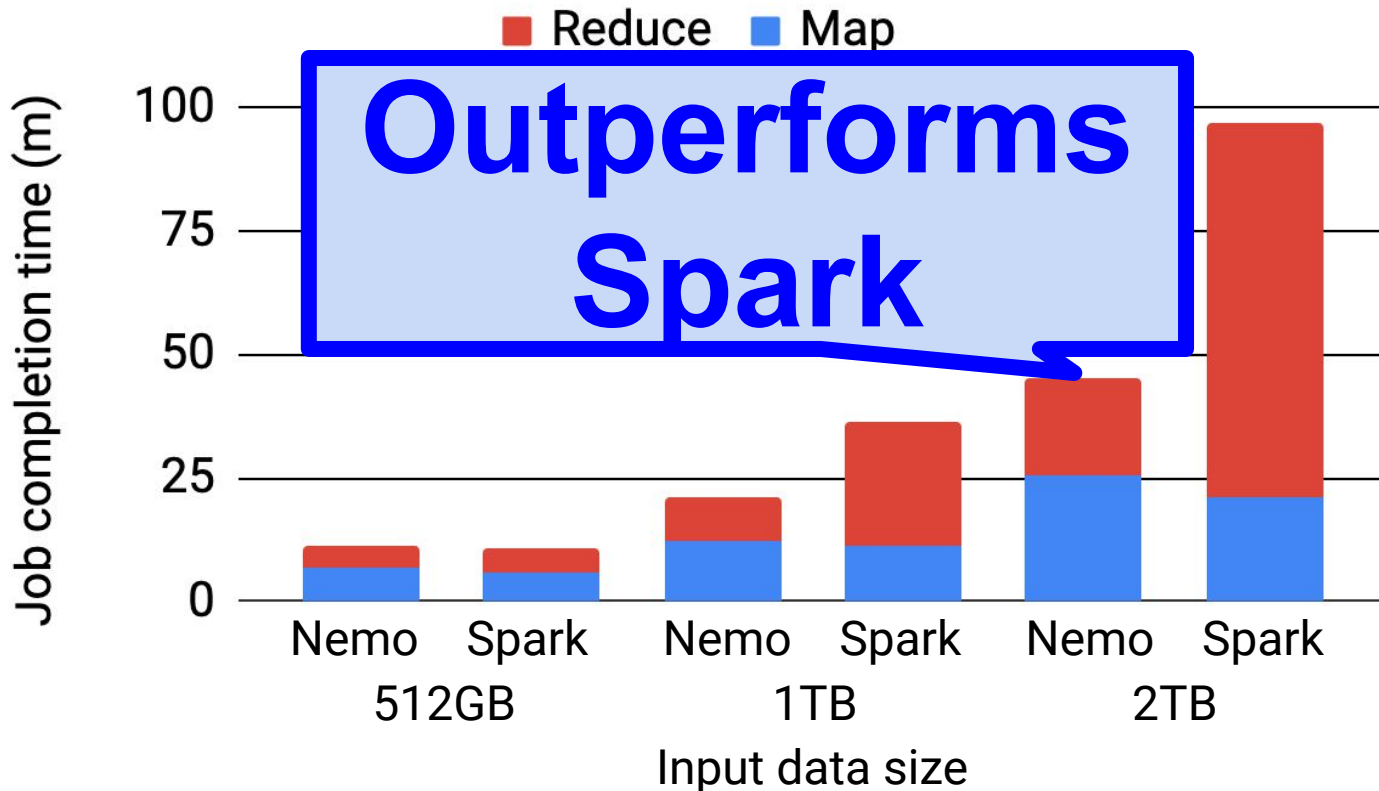
# Large Shuffle (Lower is Better)

⇒ MapReduce on 20 AWS EC2 h1.4xlarge instances



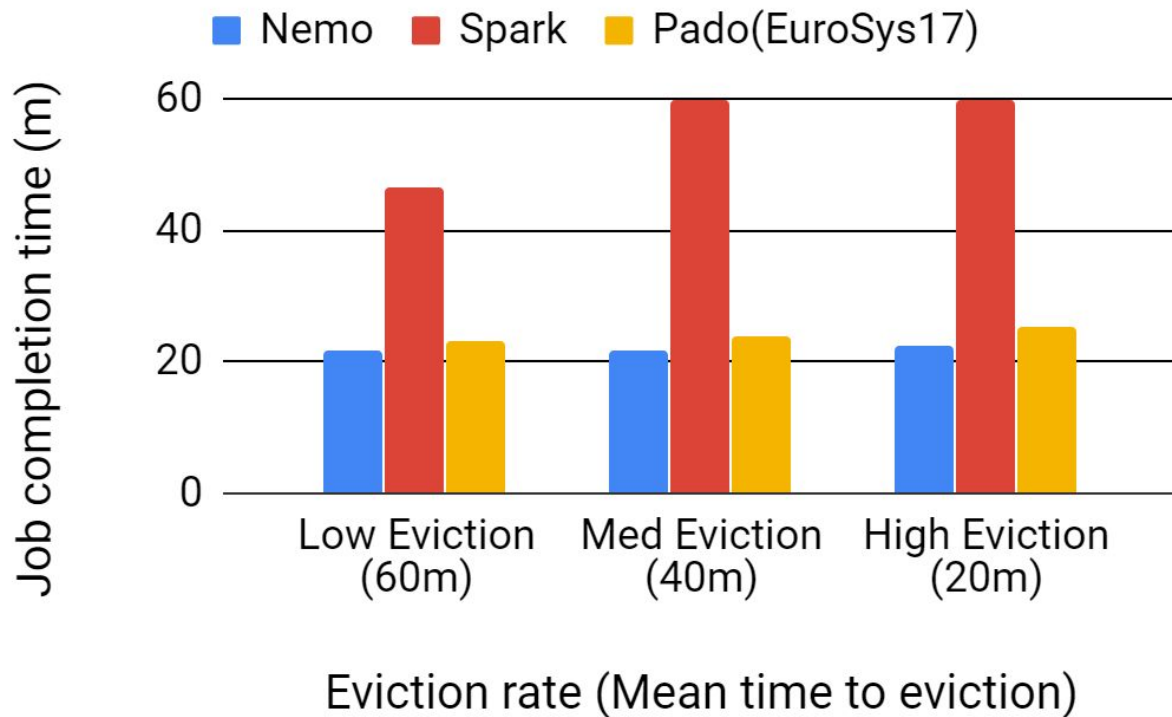
# Large Shuffle (Lower is Better)

⇒ MapReduce on 20 AWS EC2 h1.4xlarge instances



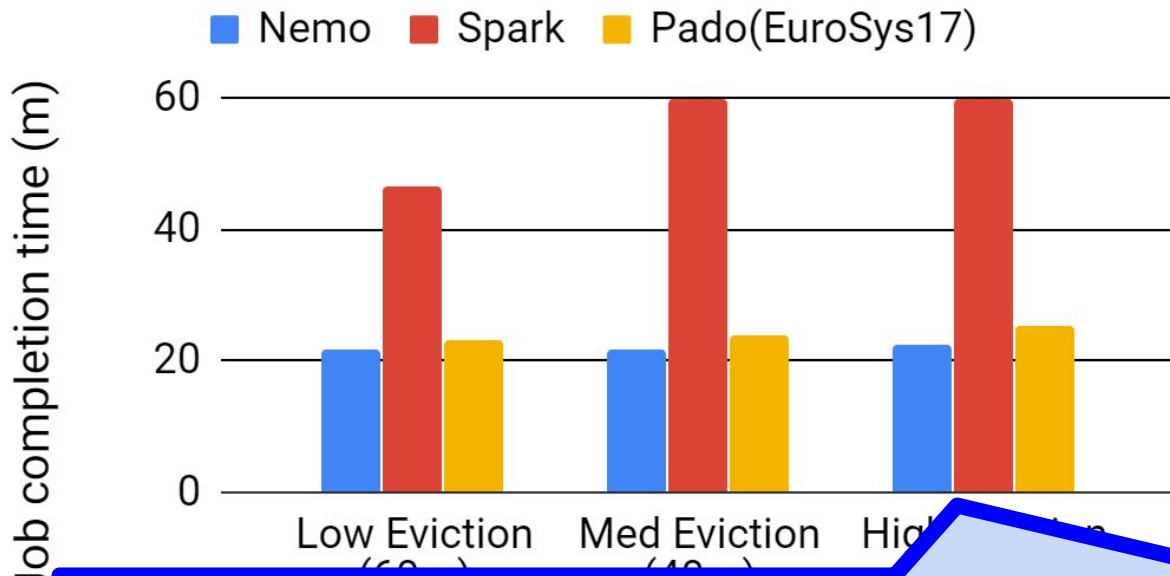
# Transient Resources (Lower is Better)

⇒ ALS on 10 transient + 2 reserved EC2 instances



# Transient Resources (Lower is Better)

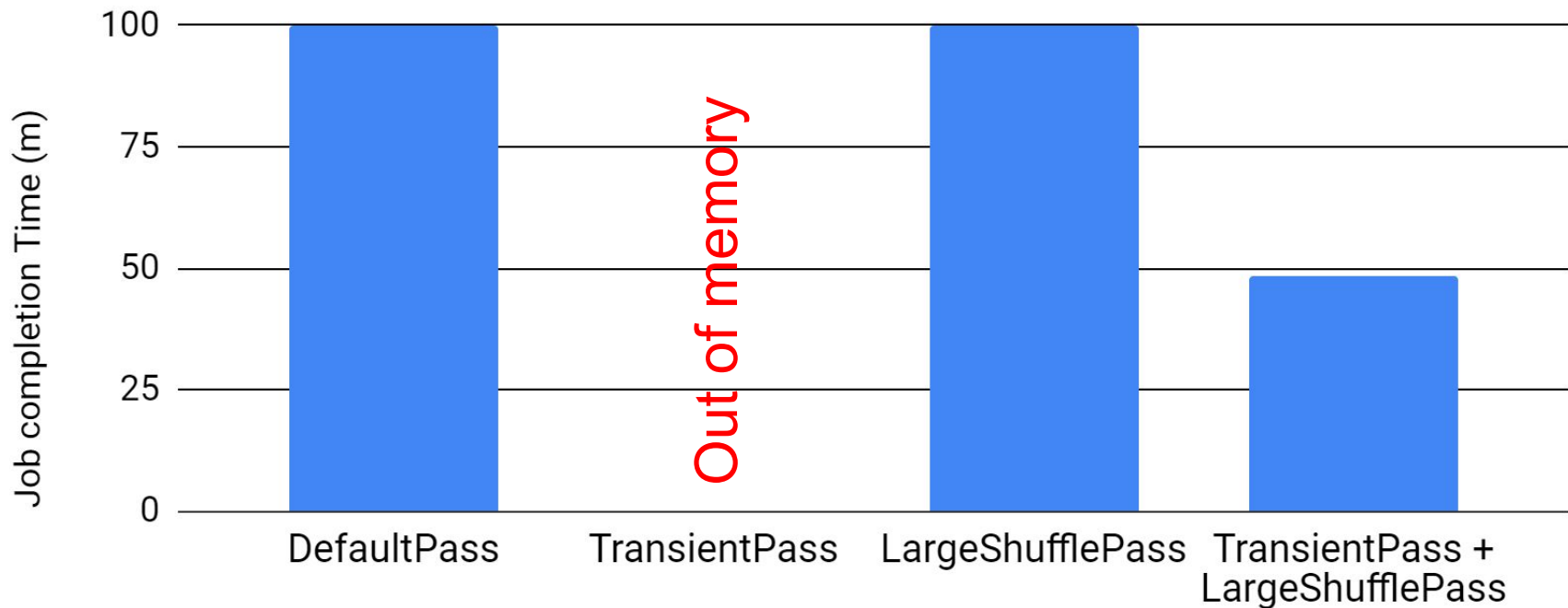
⇒ ALS on 10 transient + 2 reserved EC2 instances



**On par with Pado**

# Large Shuffle on Transient Resources

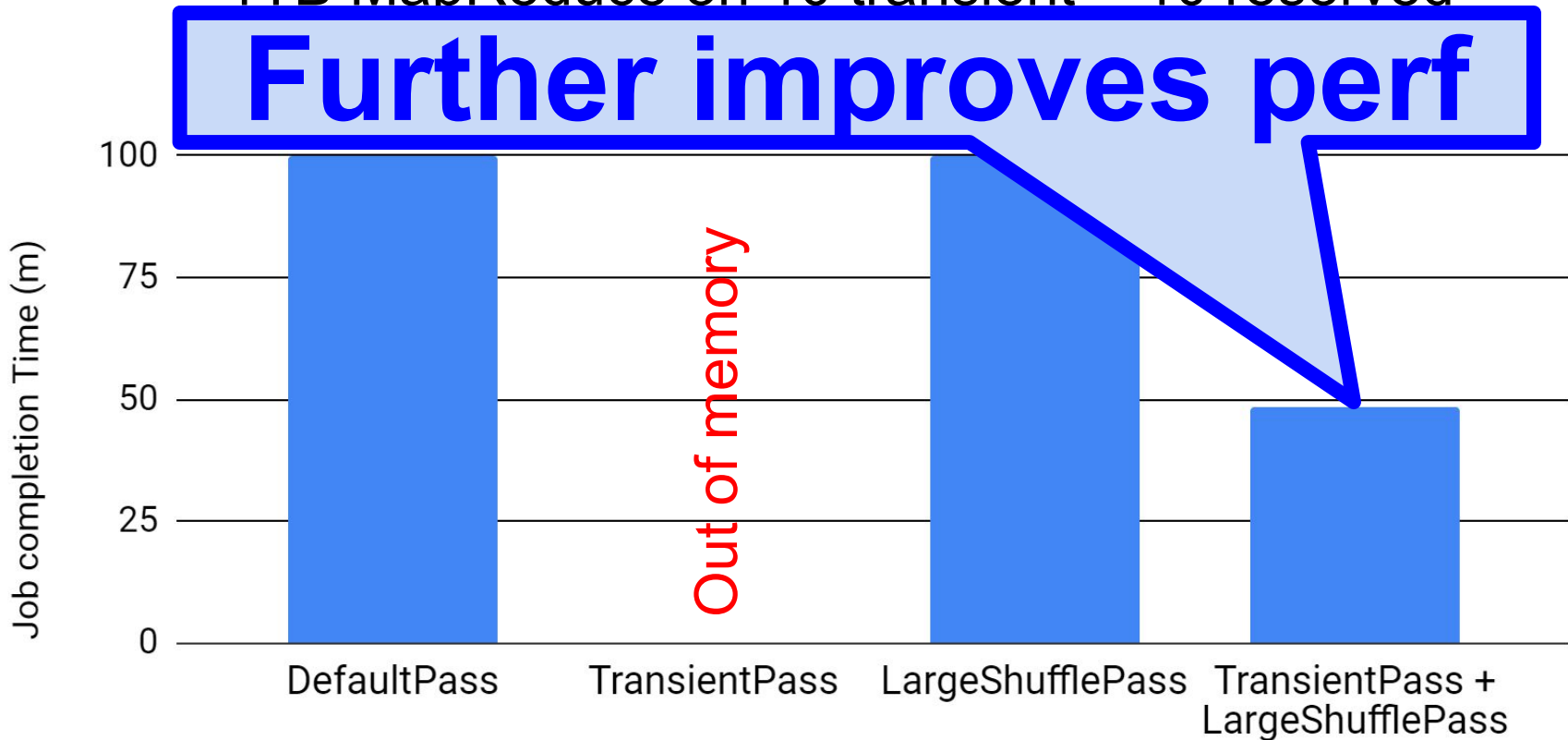
⇒ 1TB MapReduce on 10 transient + 10 reserved



# Large Shuffle on Transient Resources

⇒ 1TB MapReduce on 10 transient + 10 reserved

**Further improves perf**





# Summary: Apache Nemo

- Problem: Make it easy to optimize distributed dataflows
- Solution: **Optimization passes** that transform an intermediate representation (IR) DAG
- Result
  - Nemo outperforms a state-of-the-art Apache Spark with clean and simple **optimization passes**
  - Nemo is on par with specialized runtimes
  - Nemo further improves performance for scenarios with combined resource and data characteristics



**<https://nemo.apache.org>**

**<https://github.com/apache/incubator-nemo>**

**Build Your Own Passes,  
For Your Dataflow Research!**