计算技术研究所

优秀博士生论坛

演讲人：陆杰

指导老师：李炼（研究员，百人计划）

大纲

个人介绍

小组介绍

研究

问答

大纲

个人介绍

小组介绍

研究

问答

个人介绍

- 2014级直博生，计算所体系结构国重点实验室
- 研究领域：程序分析，分布式系统，日志分析

第一作者

SOSP 19，FSE 18

SANER 19

2 CCF-A，1 CCF-B

- 所长特别奖（夏培肃奖）
- 三好学生标兵 2019
- 国家奖学金 2019

大纲

- 个人介绍
- **小组介绍**
- 研究
- 问答

小组介绍

程序分析小组

李炼

新南威尔士大学

甲骨文

中科院百人计划

小组研究方向

通过程序分析技术来帮助提高软件系统的可靠性和安全性(缺陷和漏洞检测)

小组研究方向

静态&动态分析检测系统：Wukong

小组研究方向

静态&动态分析检测系统：Wukong

深度缺陷和漏洞

小组研究方向

静态&动态分析检测系统：Wukong

深度缺陷和漏洞

C/C++,Java, Go
Android, 分布式

小组研究方向

静态&动态分析检测系统：Wukong

深度缺陷和漏洞

C/C++,Java, Go
Android, 分布式

SOSP， FSE,
ASE,CGO等

Best Paper

大纲

个人介绍

小组介绍

研究

问答

计算技术研究所

分布式系统崩溃恢复缺陷检测

演讲人：陆杰

指导老师：李炼（研究员，百人计划）

计算技术研究所

分布式系统 崩溃 恢复 缺陷 检测

演讲人：陆杰

指导老师：李炼（研究员，百人计划）

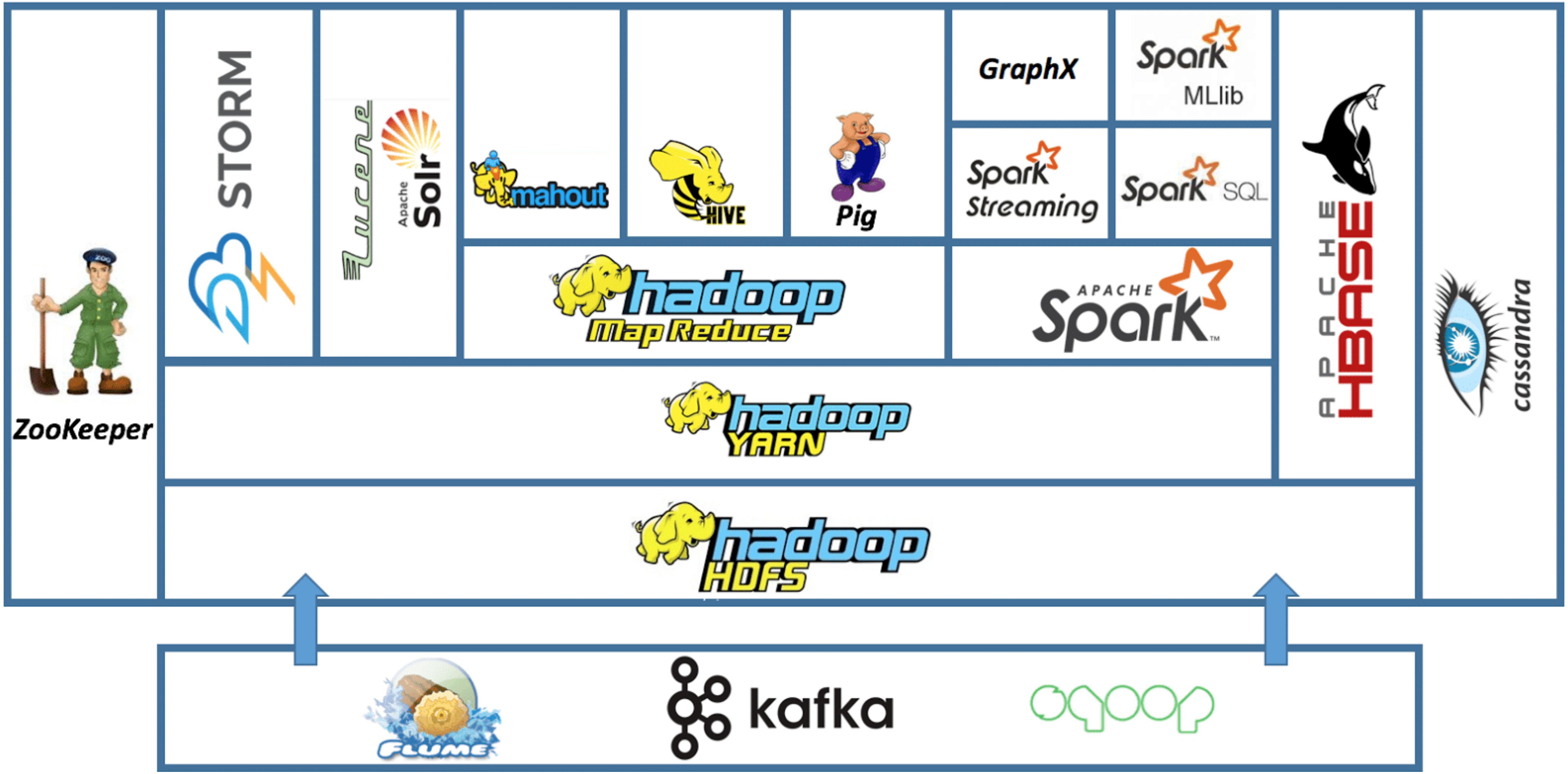
计算技术研究所

分布式系统 崩溃 恢复 缺陷 检测

演讲人：陆杰

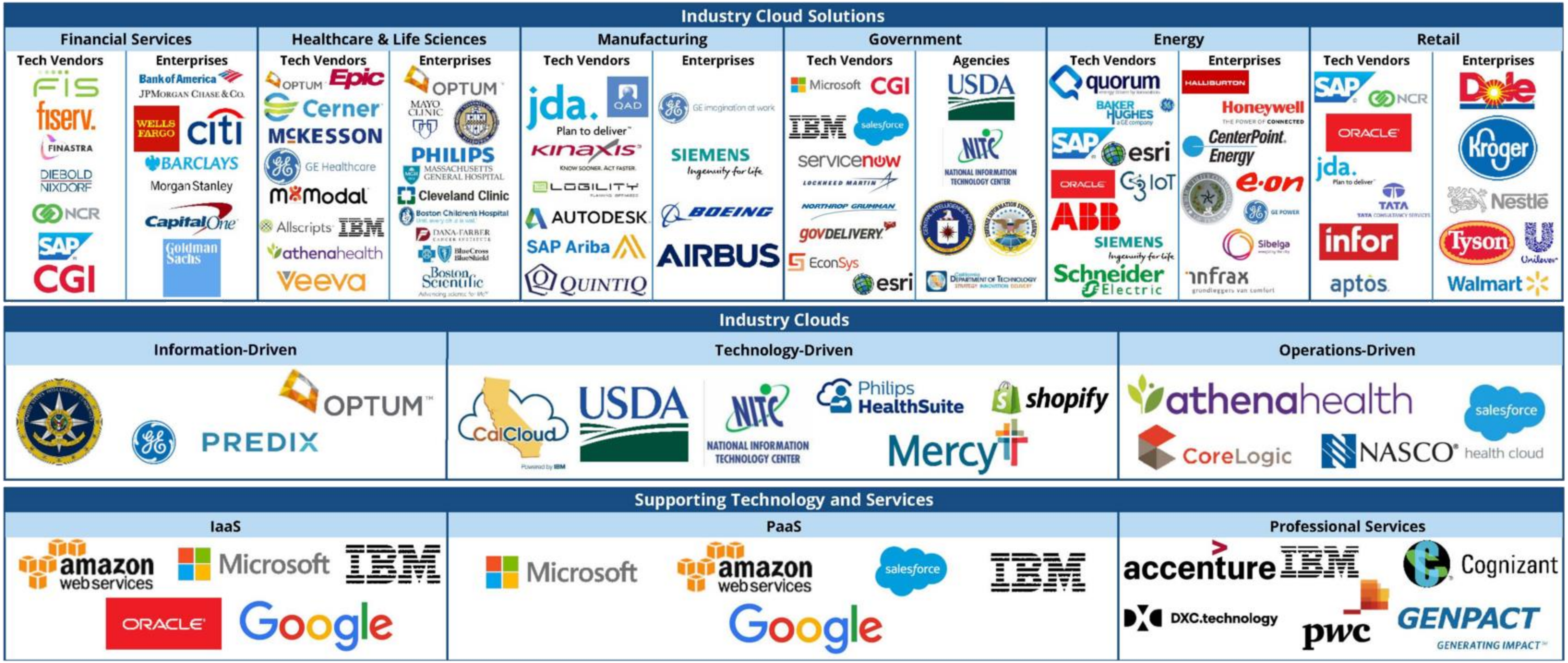
指导老师：李炼（研究员，百人计划）

分布式系统



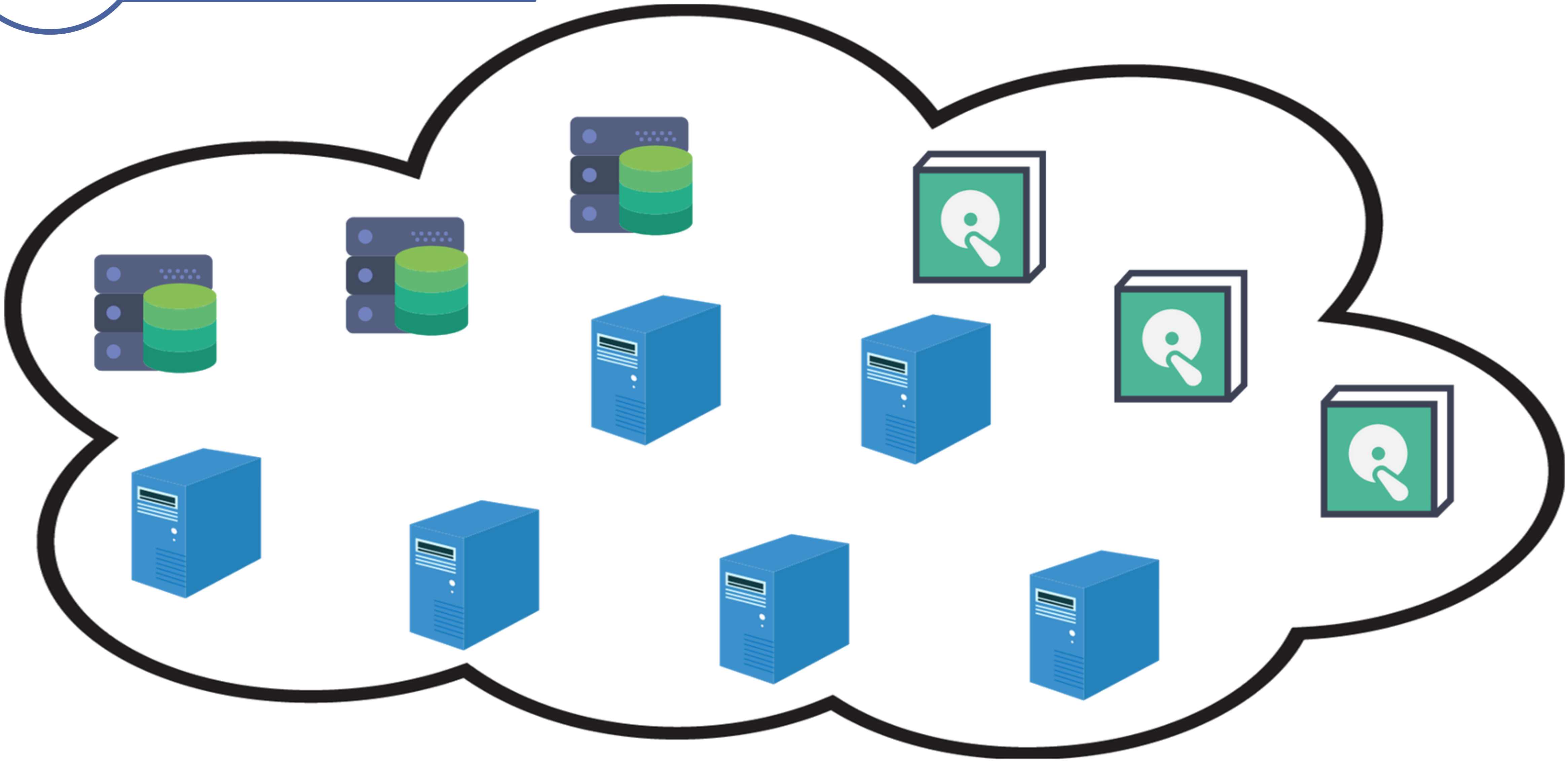
ZooKeeper
STORM
Lucene
Apache Solr
mahout
HIVE
Pig
GraphX
Spark MLlib
Spark Streaming
Spark SQL
hadoop Map Reduce
APACHE Spark
APACHE HBASE
cassandra
hadoop YARN
hadoop HDFS
FLUME
kafka
sqoop

分布式系统

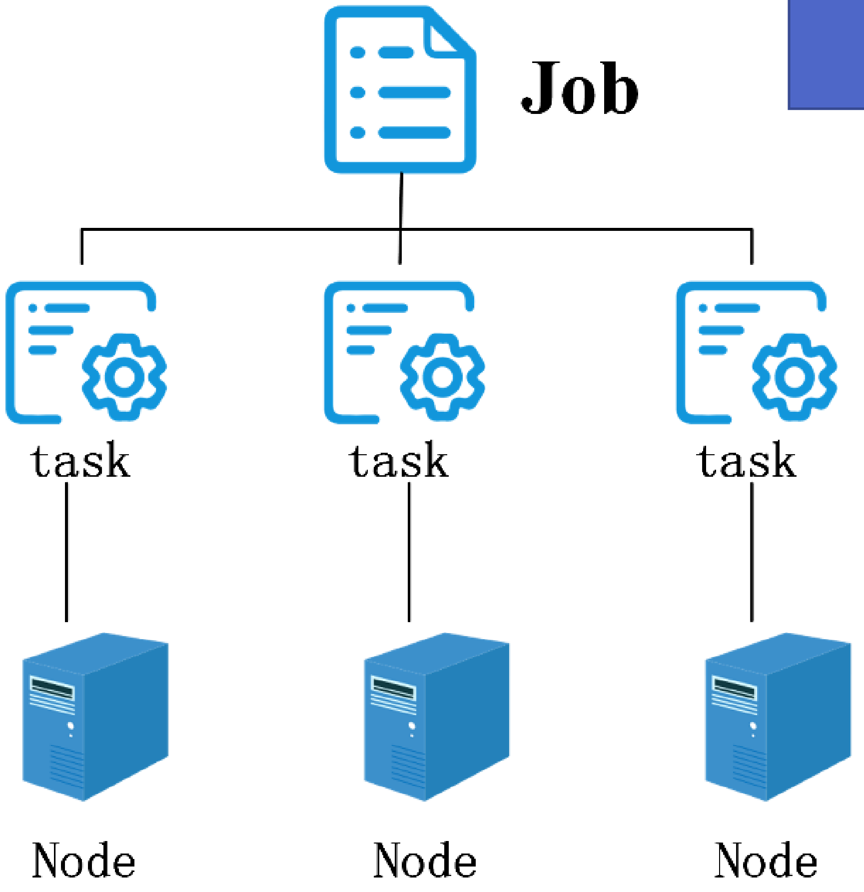


Industry Cloud Solutions
Financial Services
Healthcare & Life Sciences
Manufacturing
Government
Energy
Retail
Tech Vendors
Enterprises
Agencies
Industry Clouds
Information-Driven
Technology-Driven
Operations-Driven
Supporting Technology and Services
IaaS
PaaS
Professional Services

分布式系统



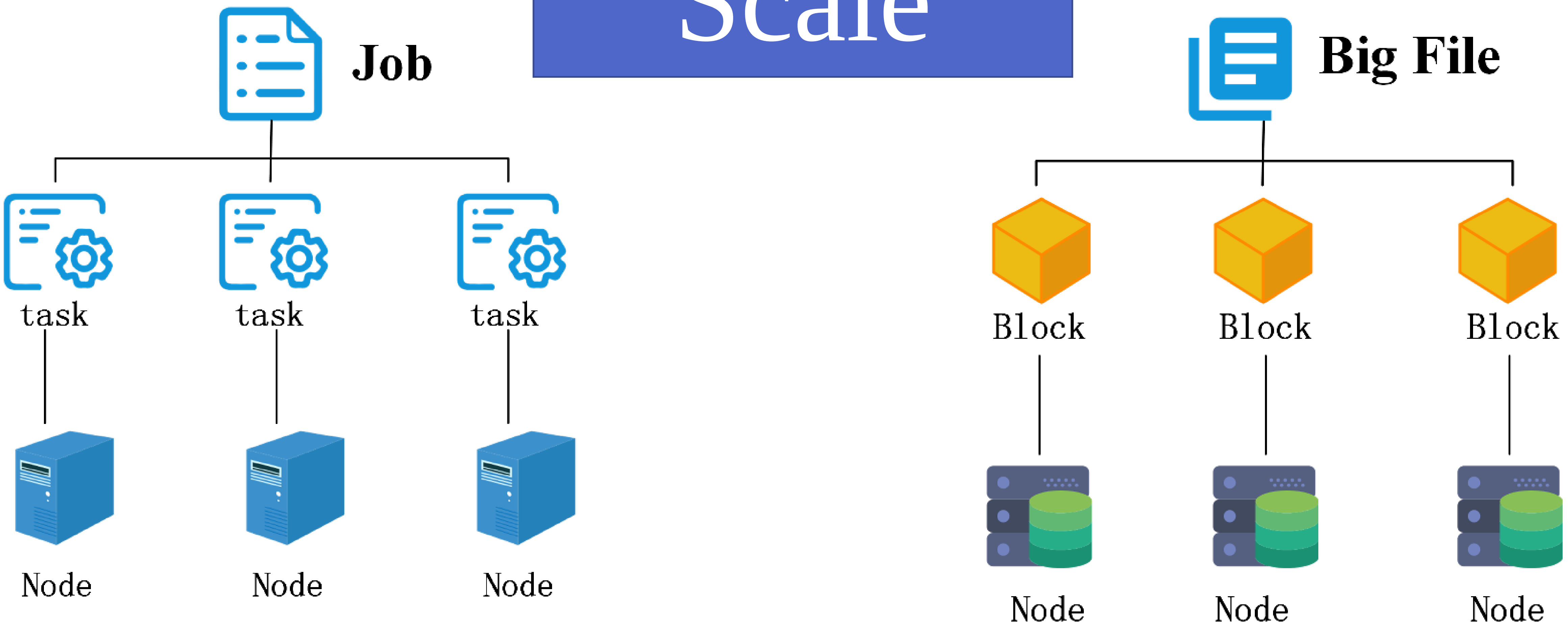
分布式系统



Scale

分布式系统

Scale


Job
task
task
task
Node
Node
Node
Big File
Block
Block
Block
Node
Node
Node

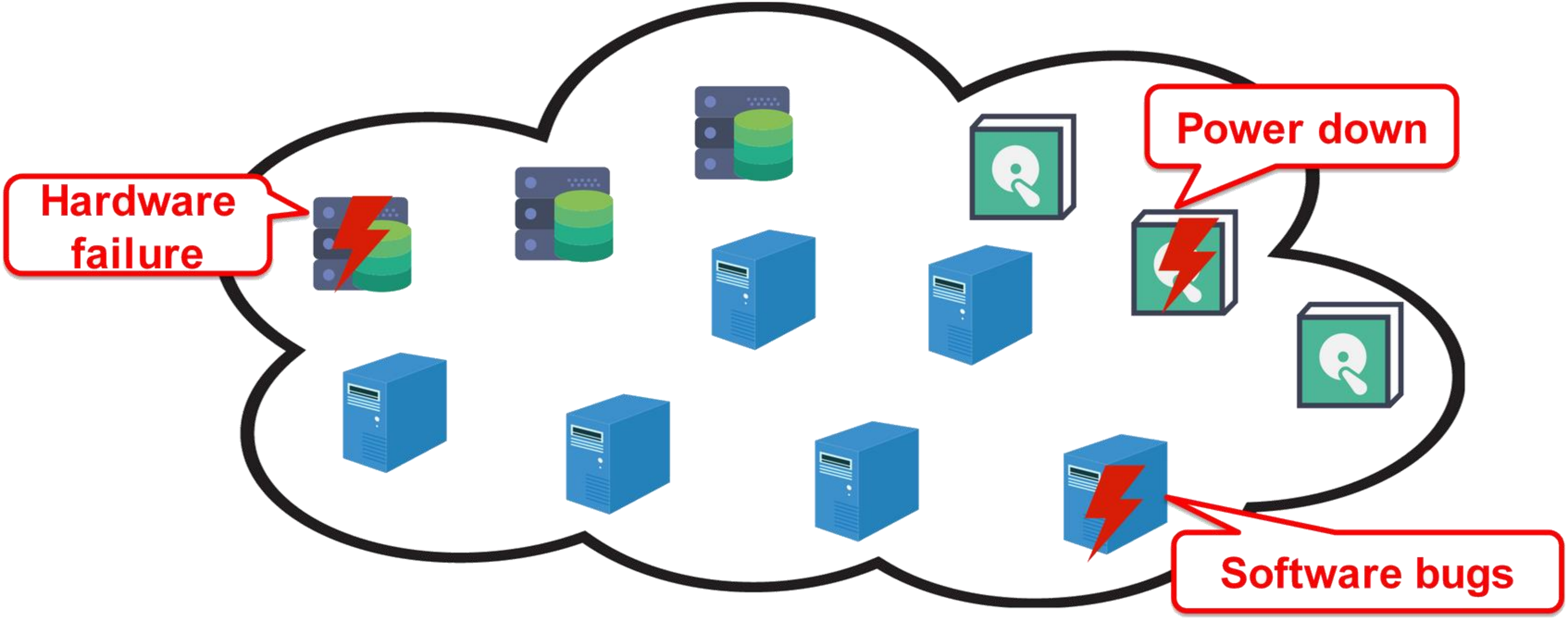
计算技术研究所

分布式系统 崩溃 恢复 缺陷 检测

演讲人：陆杰

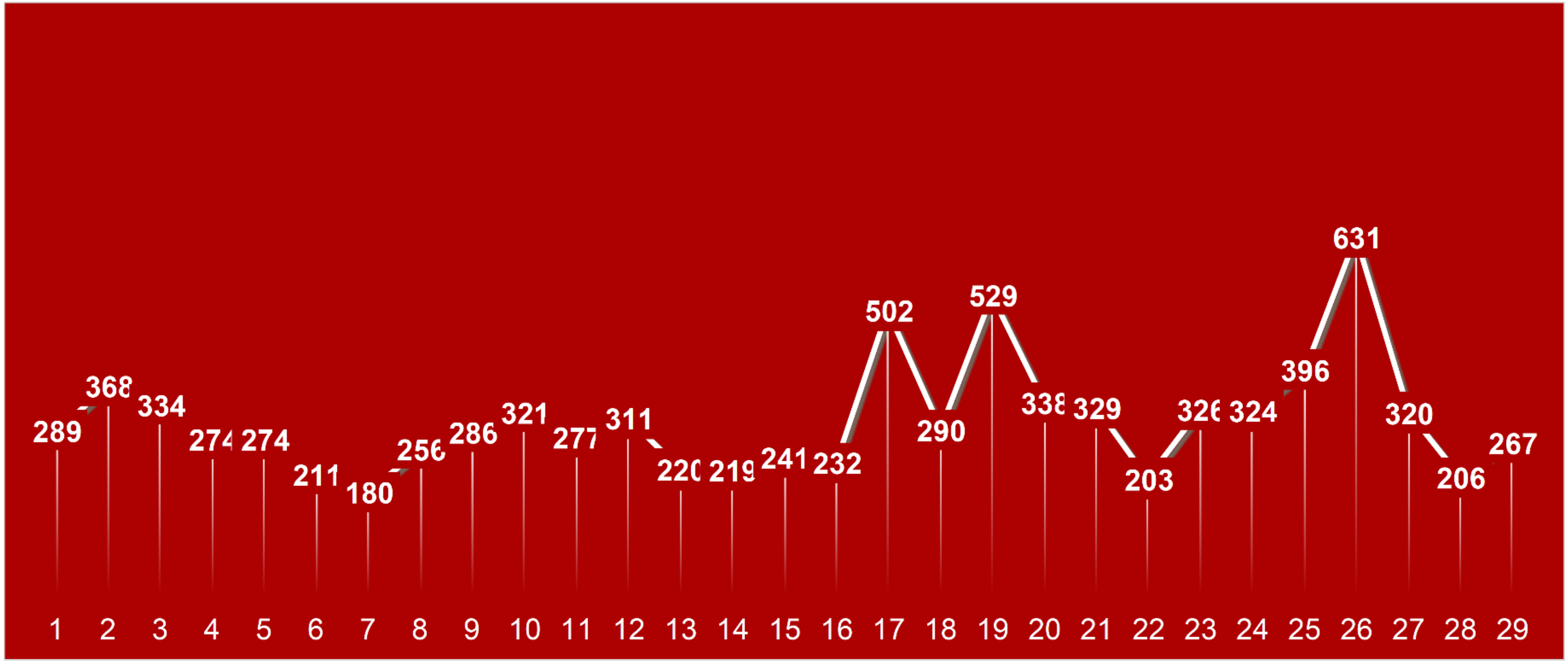
指导老师：李炼（研究员，百人计划）

崩溃(Crash)



崩溃(crash)

每天至少180节点崩溃（谷歌）



计算技术研究所

分布式系统 崩溃 恢复 缺陷 检测

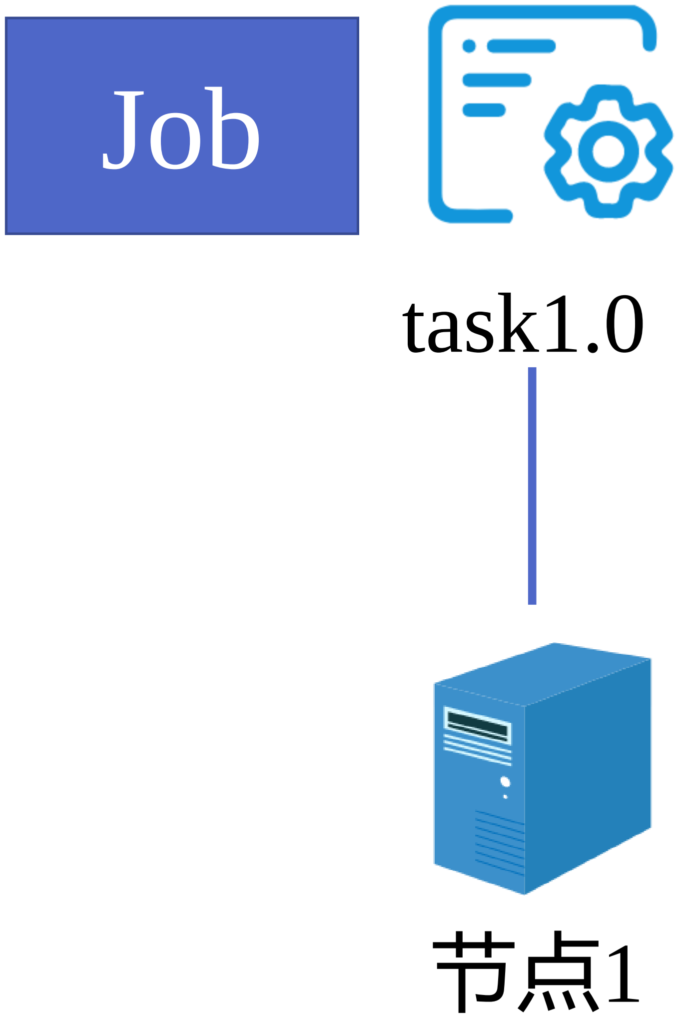
演讲人：陆杰

指导老师：李炼（研究员，百人计划）

恢复(recovery)

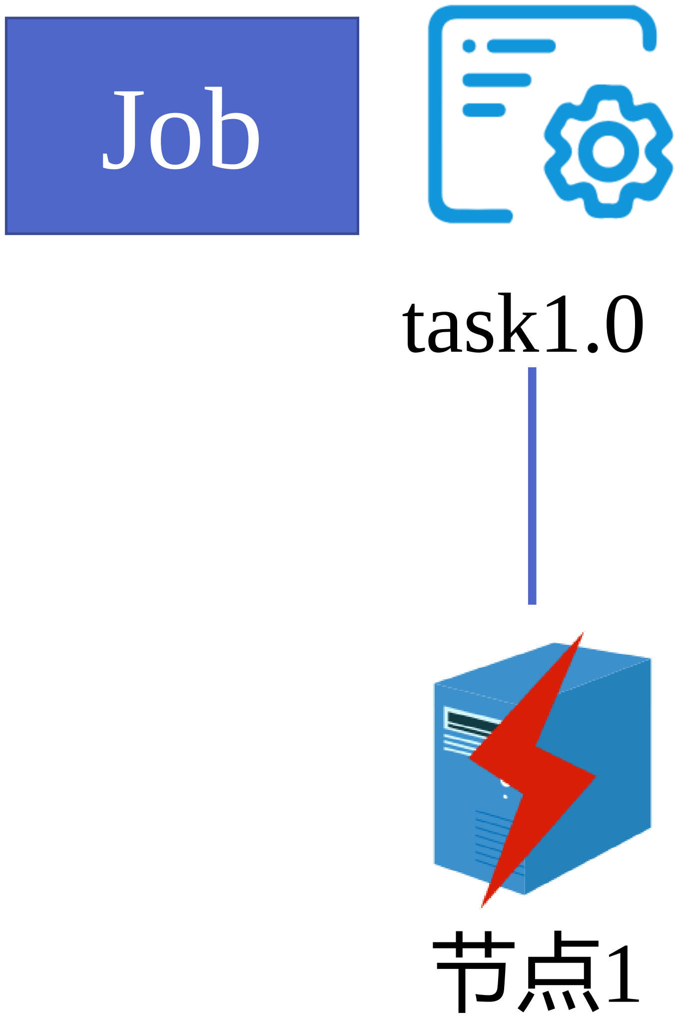
恢复是分布式系统的首要操作

恢复(recovery)



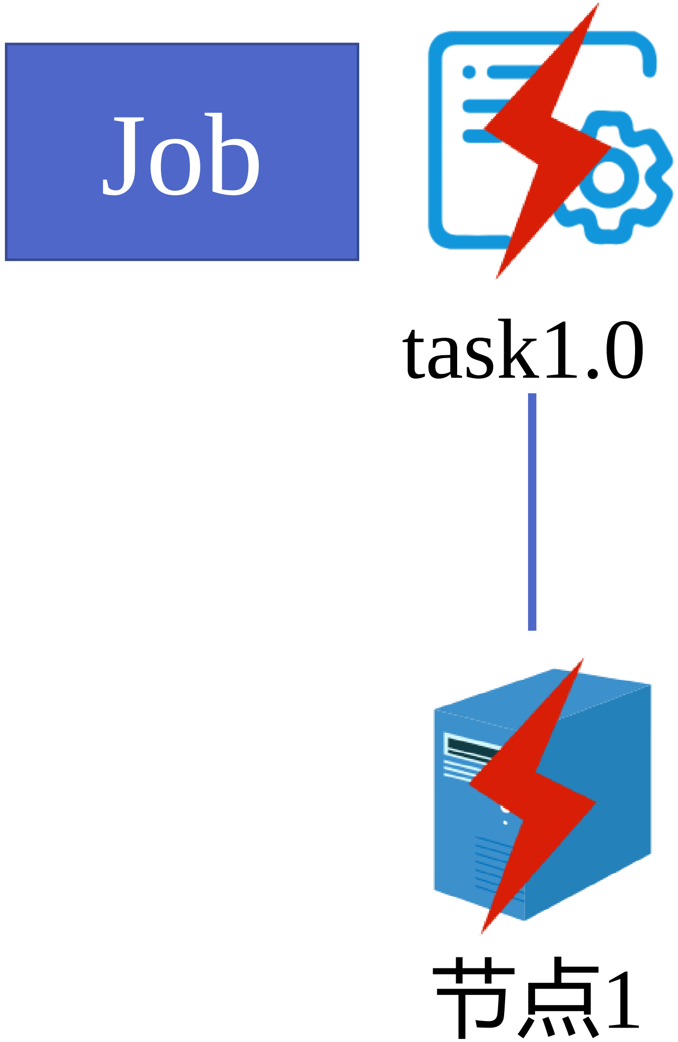
容错

恢复(recovery)



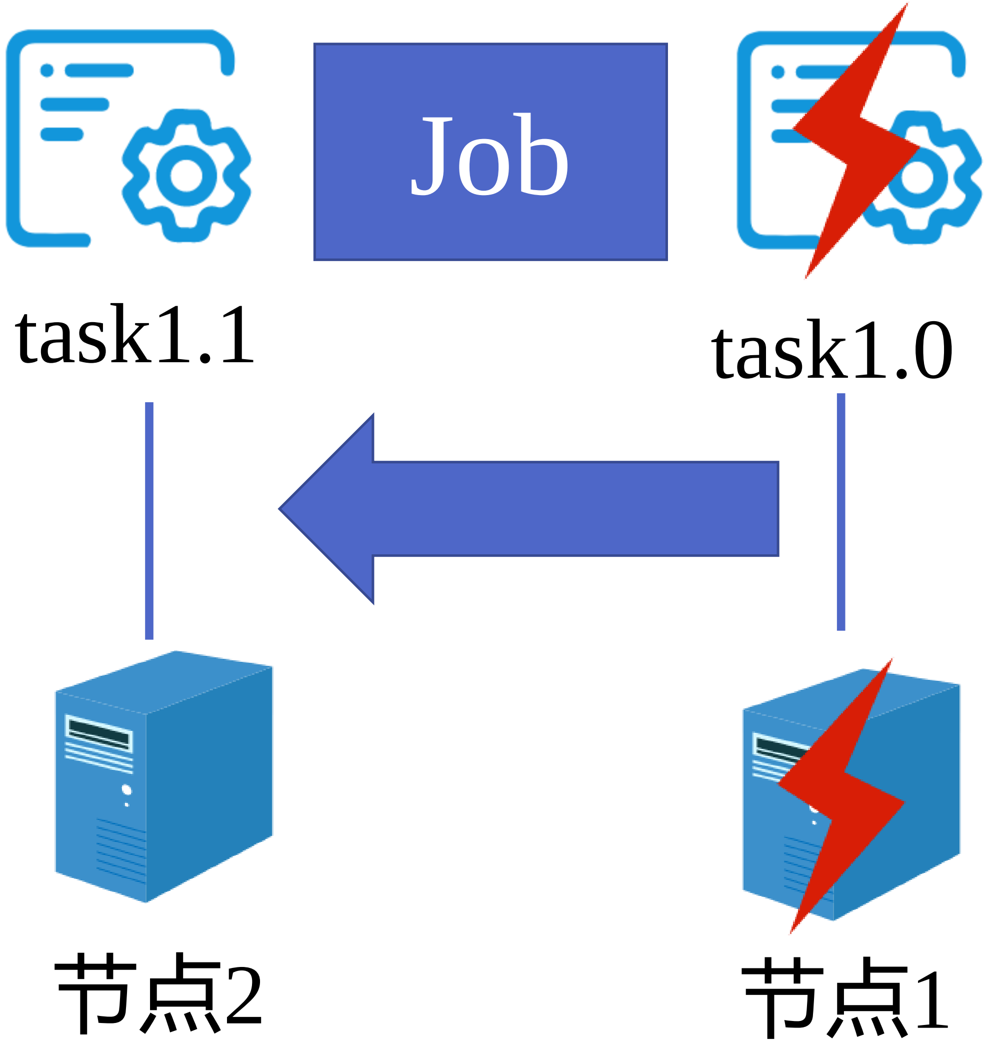
容错

恢复(recovery)



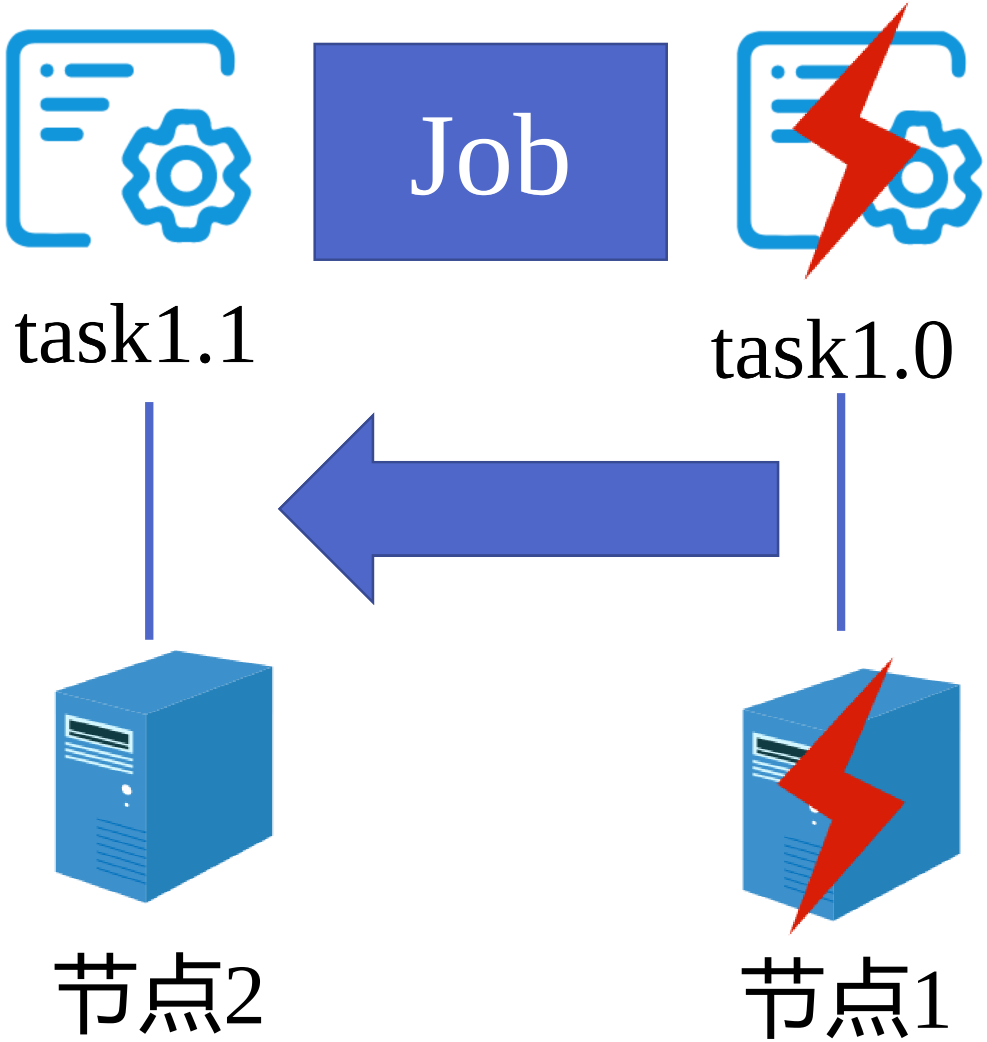
容错

恢复(recovery)



容错

恢复(recovery)



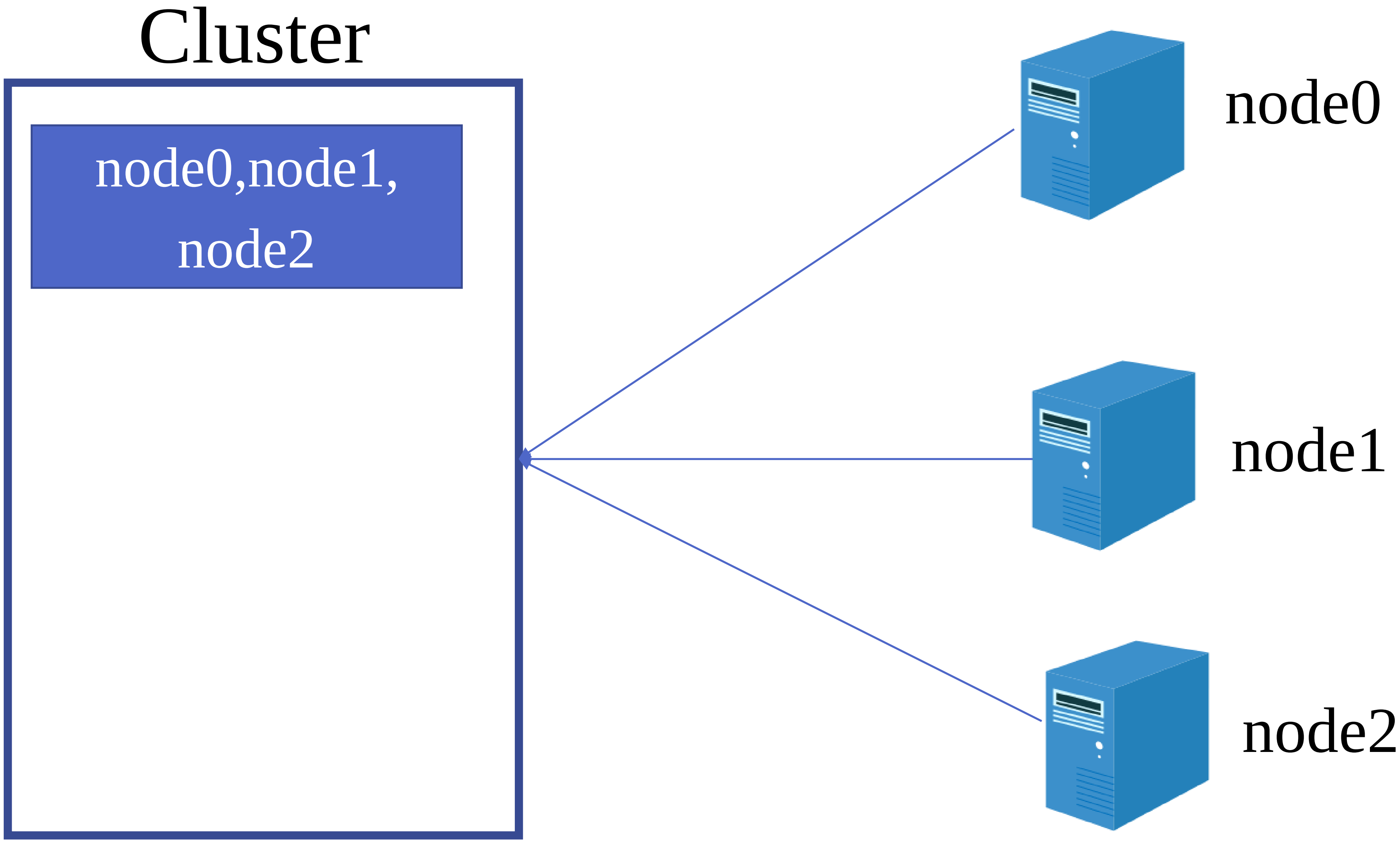
更新

元信息变量

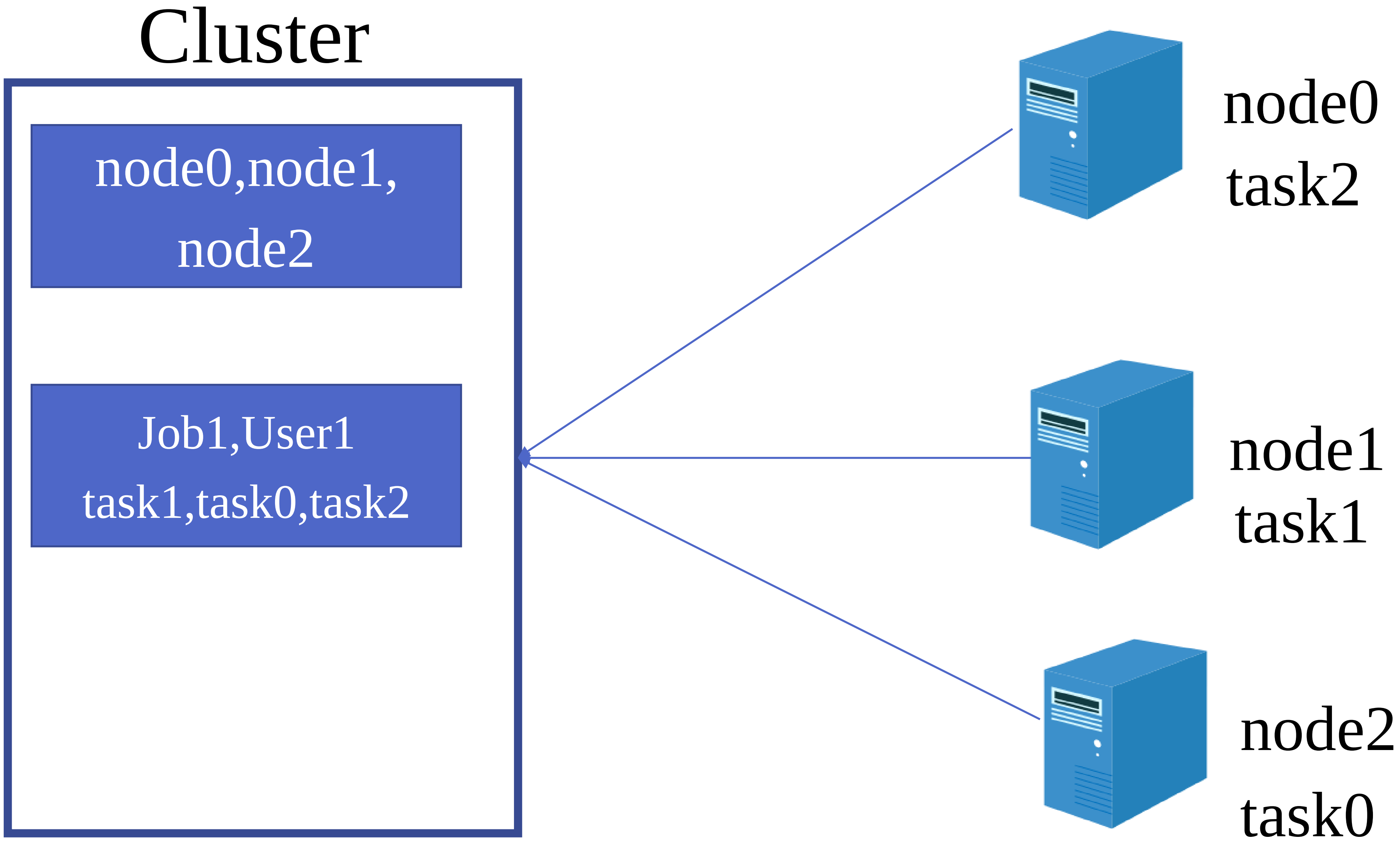
元信息变量

Cluster

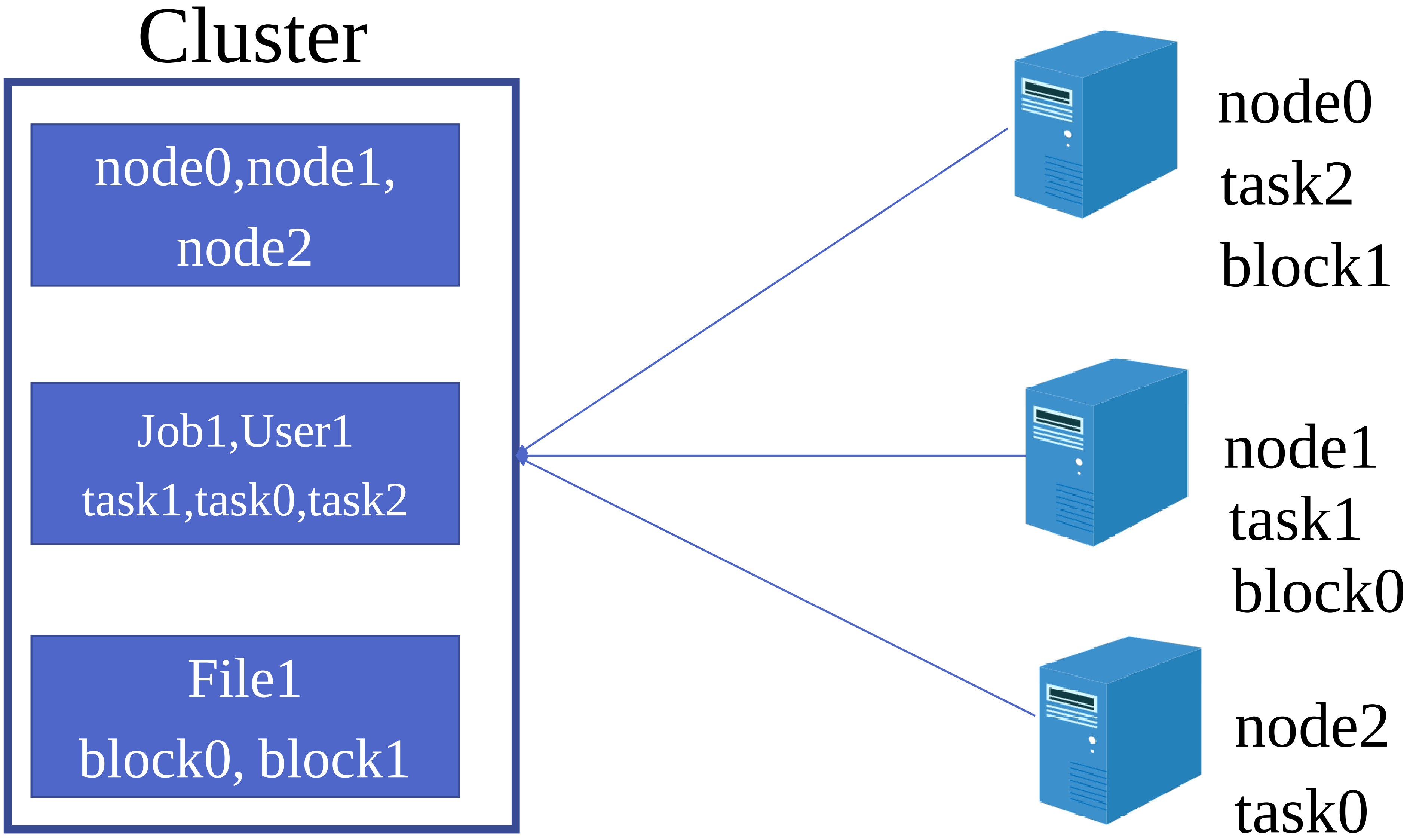
元信息变量



元信息变量

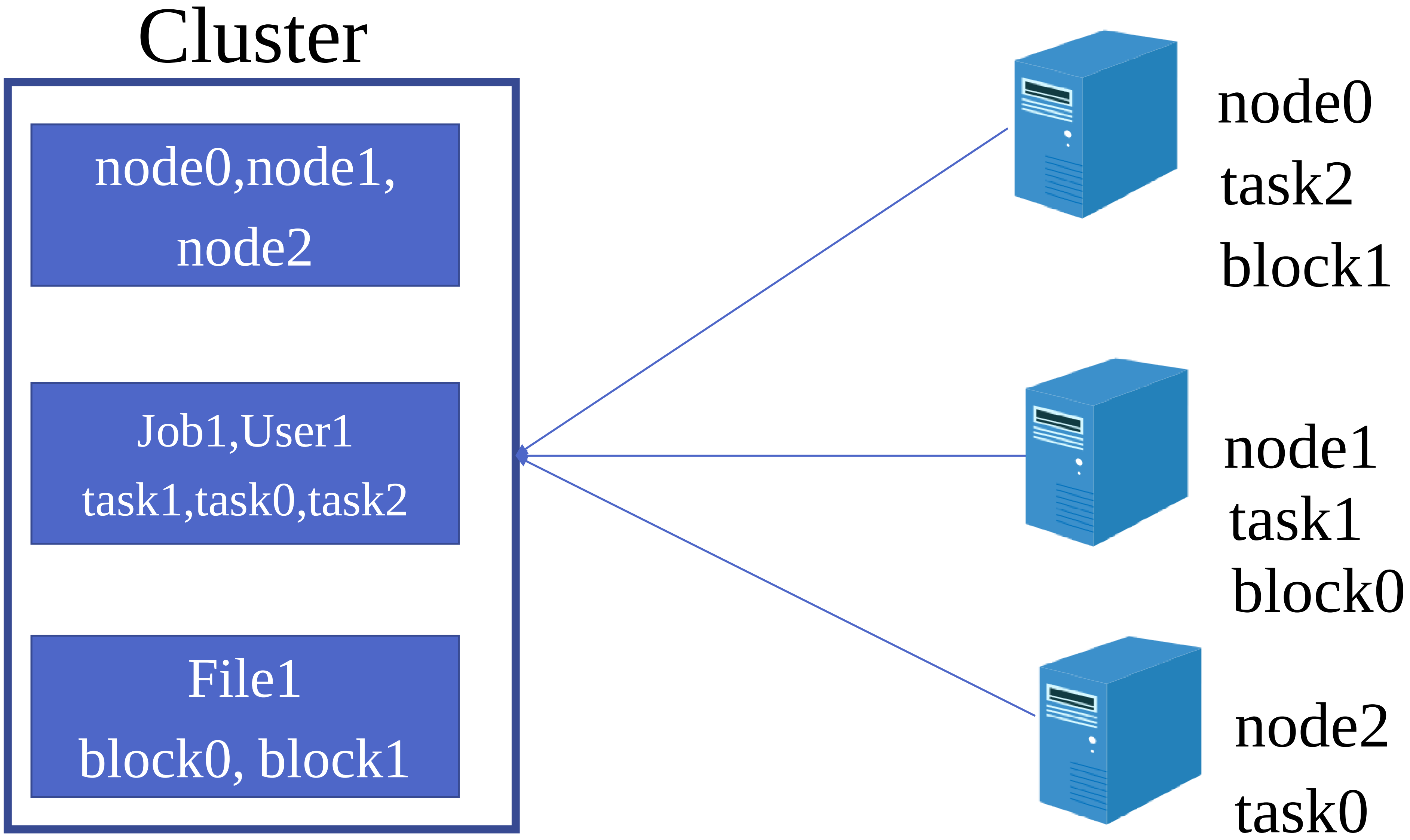

Cluster
node0,node1,
node2
Job1,User1
task1,task0,task2
node0
task2
node1
task1
node2
task0

元信息变量


Cluster
node0,node1,
node2
Job1,User1
task1,task0,task2
File1
block0, block1
node0
task2
block1
node1
task1
block0
node2
task0

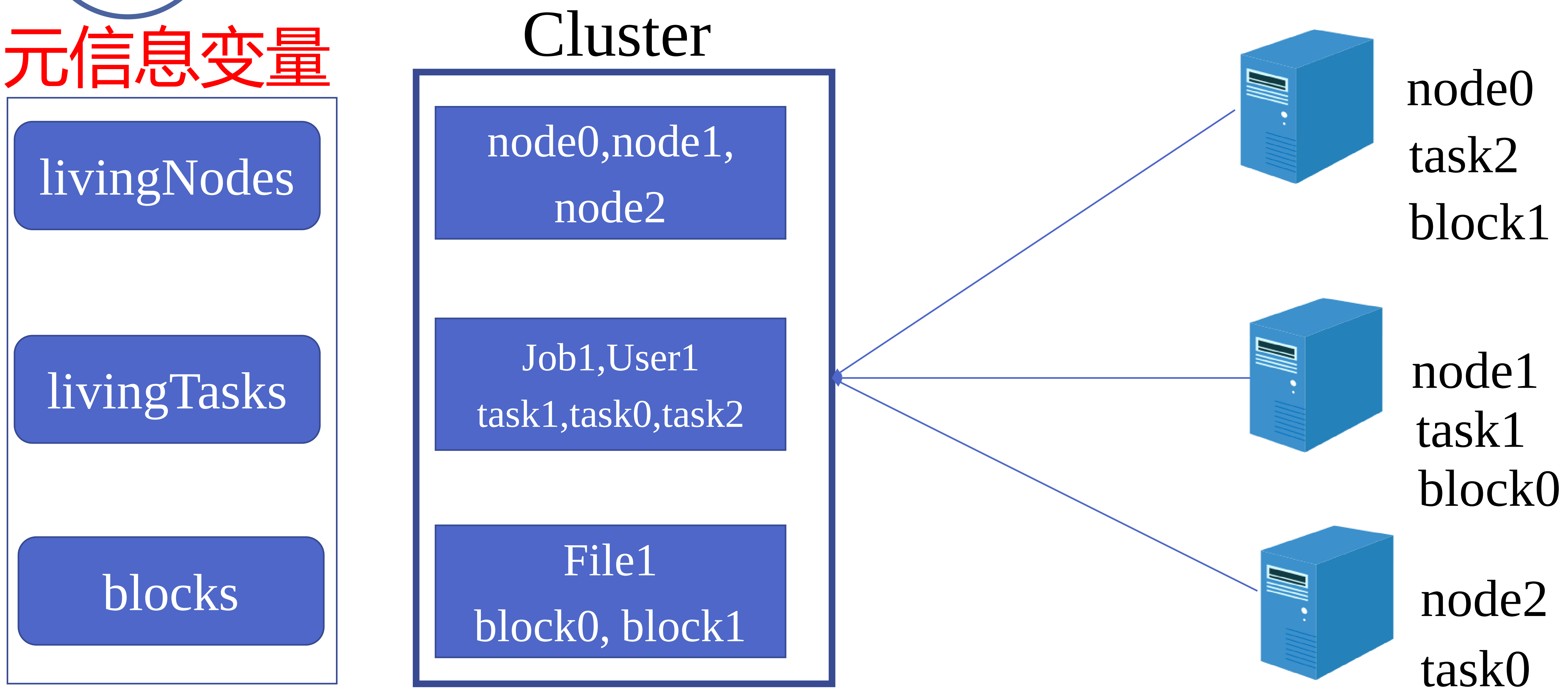
元信息变量

元信息:nodes, tasks,job,block等



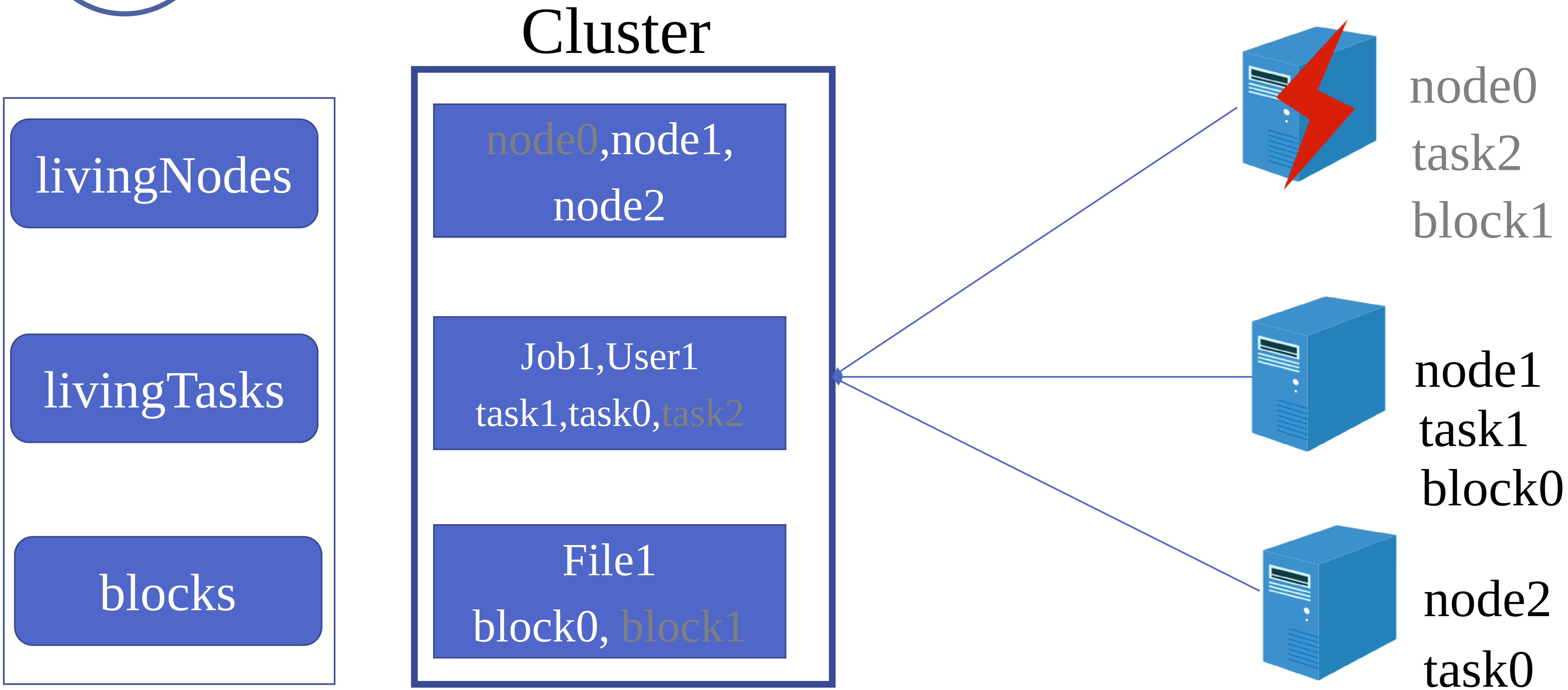
元信息变量

元信息:nodes, tasks,job,block等



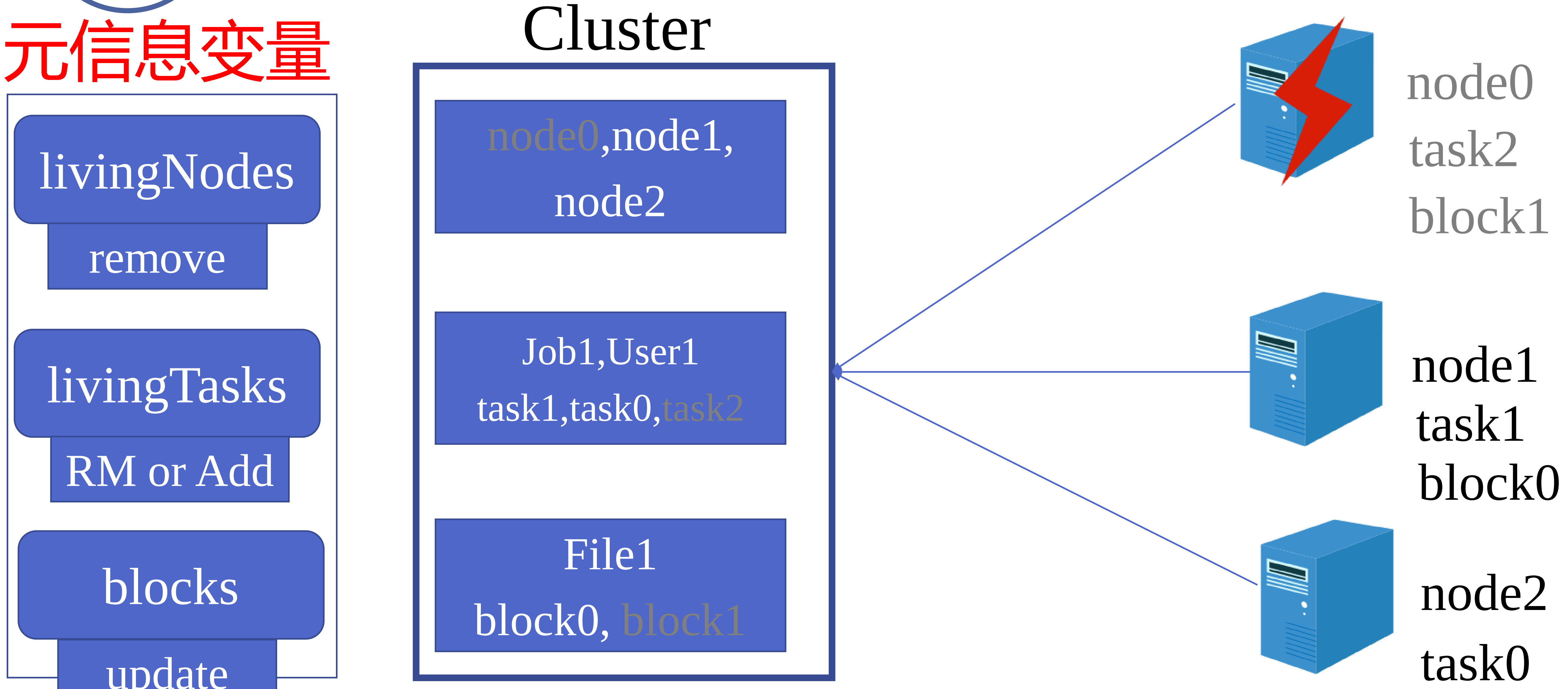
元信息变量

元信息:nodes, tasks,job,block等

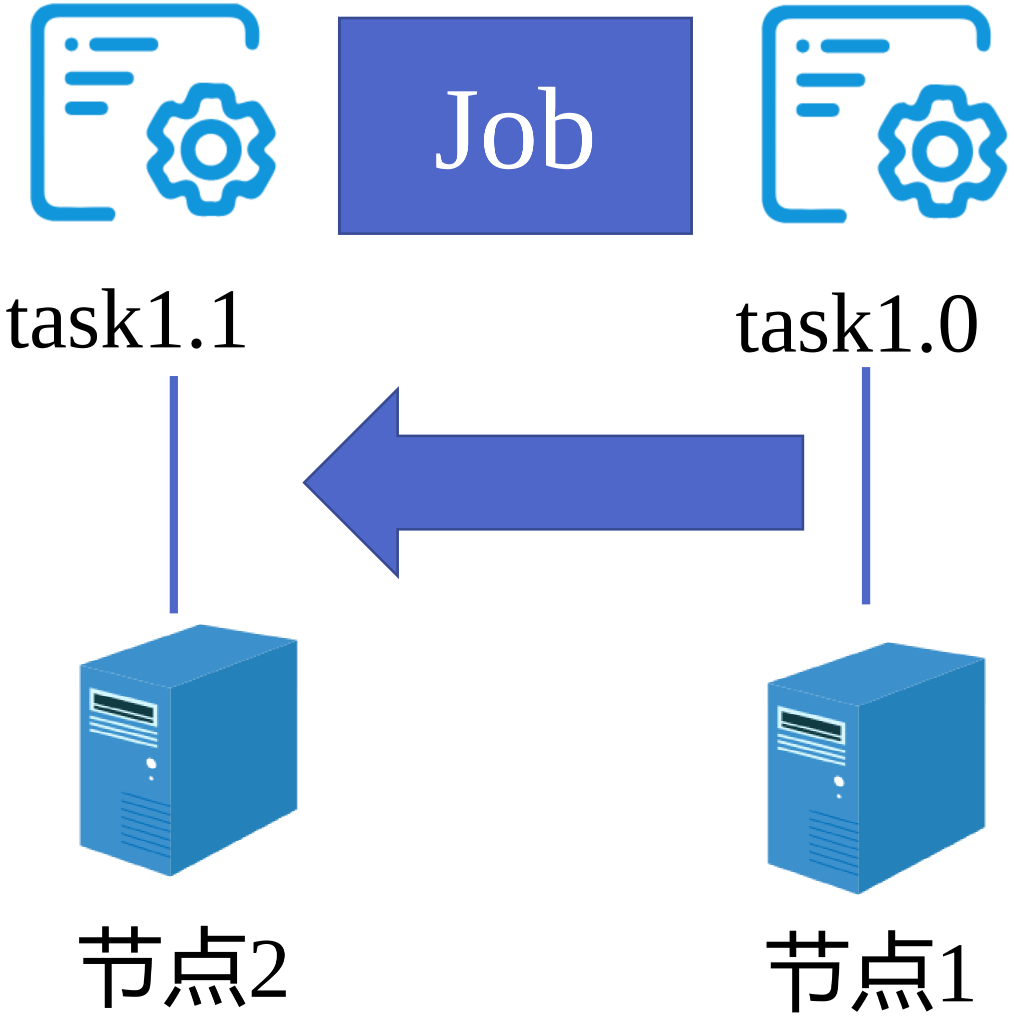


元信息变量

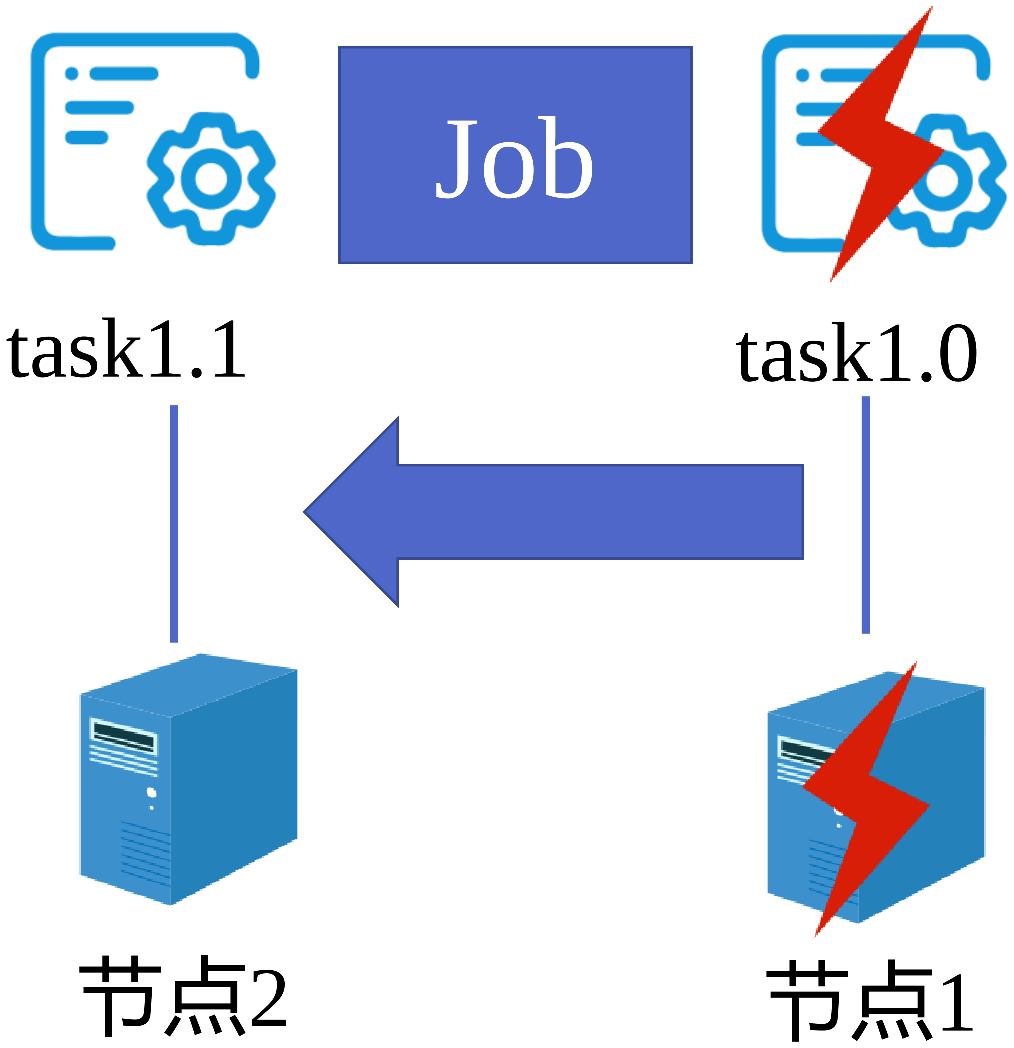
元信息:nodes, tasks,job,block等



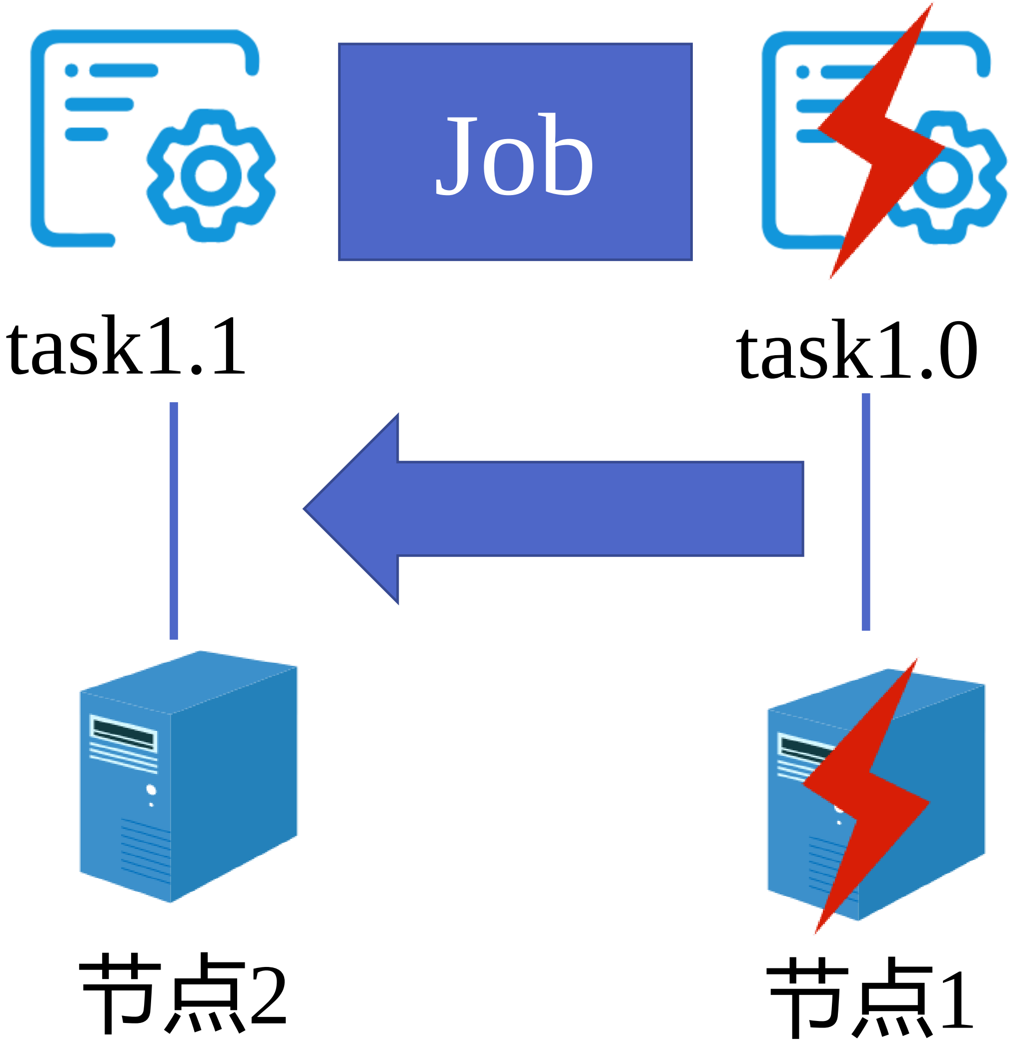
小结



小结

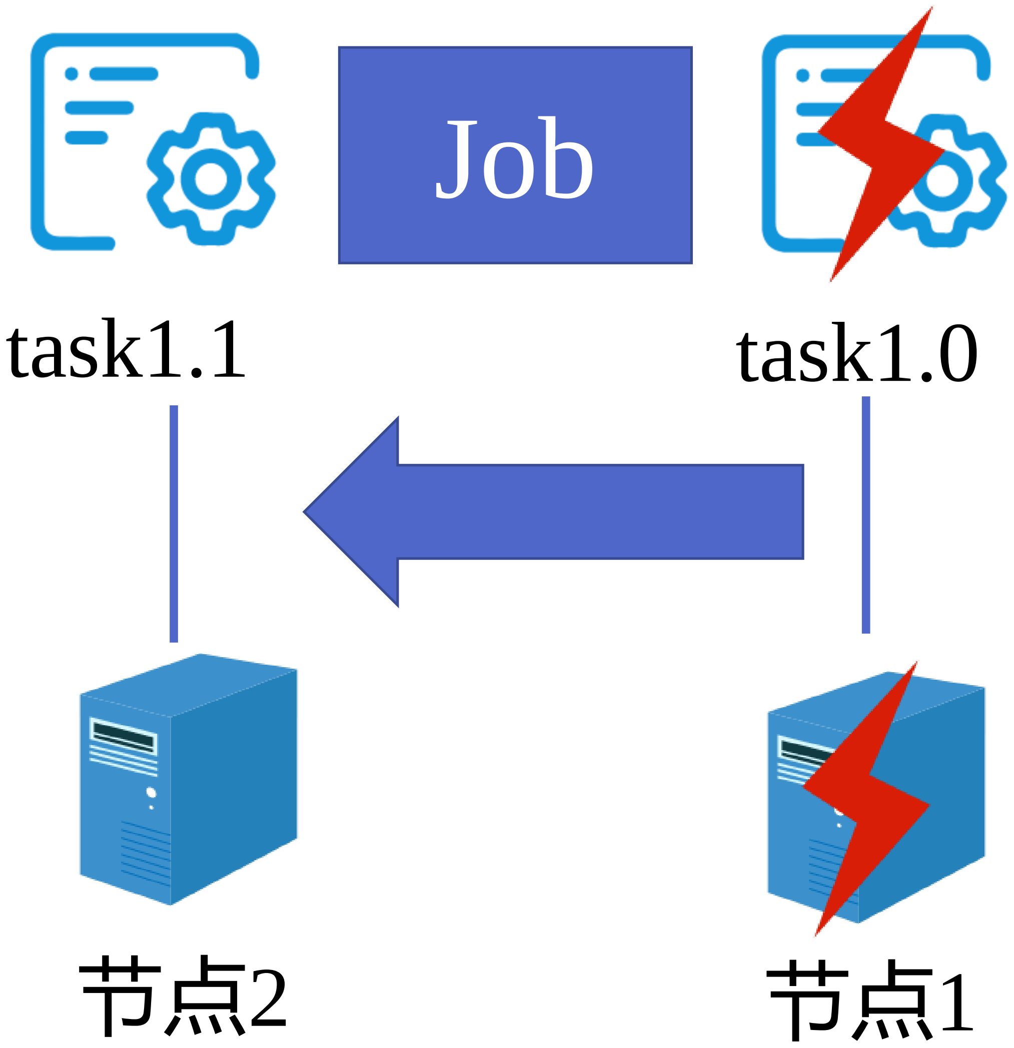


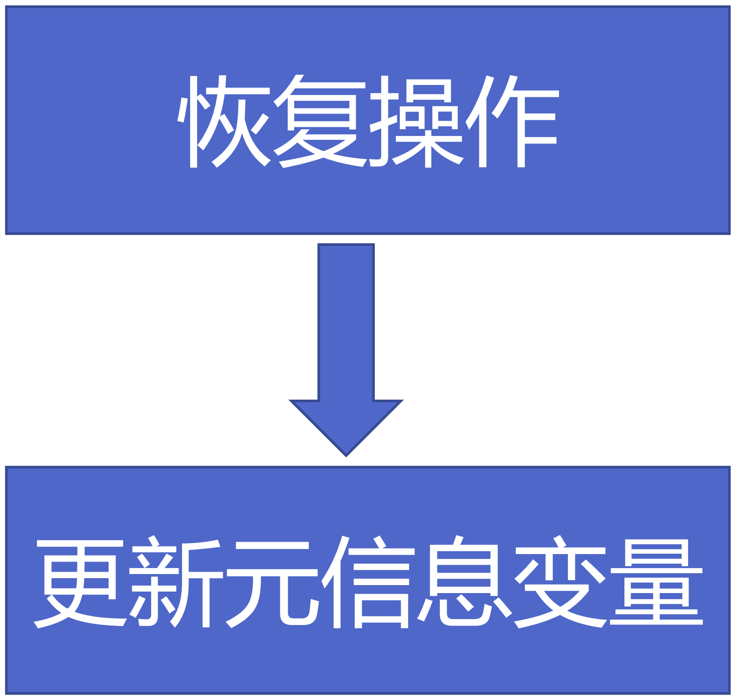
小结



恢复操作

小结





计算技术研究所

分布式系统 崩溃 恢复

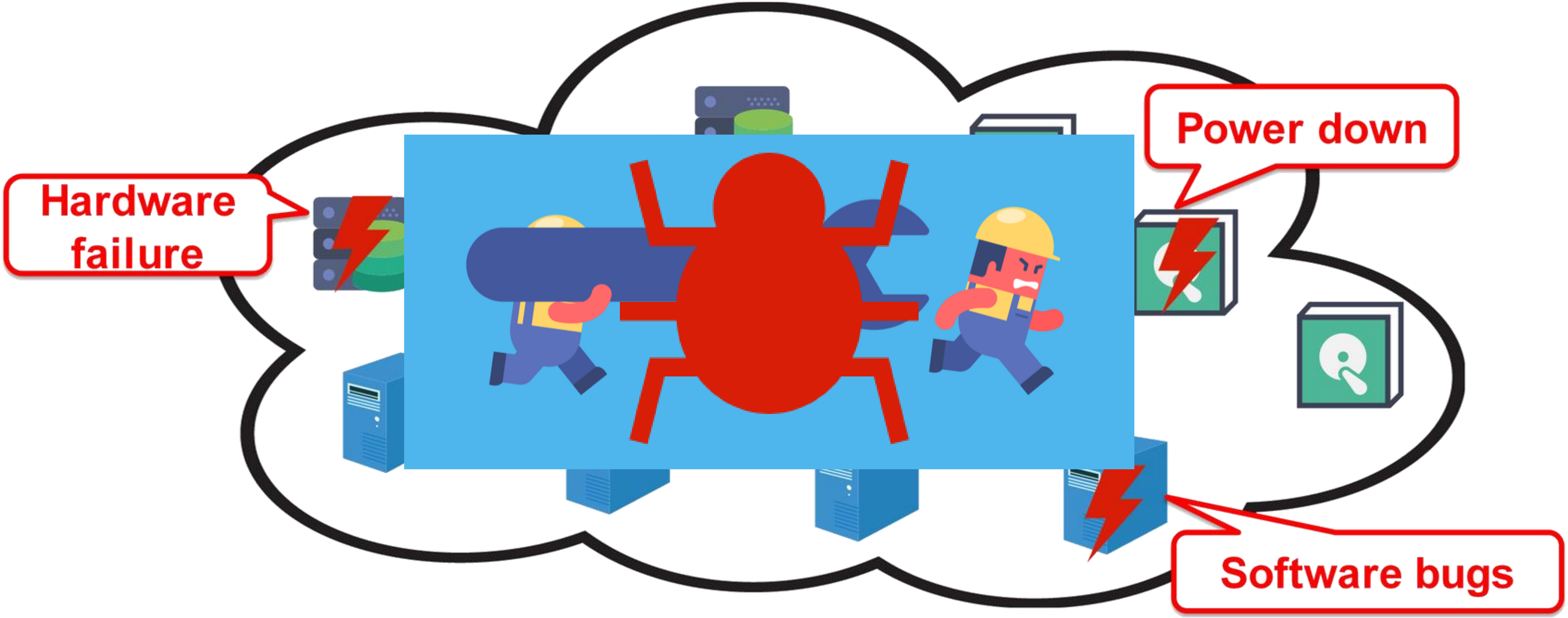
缺陷 检测

演讲人：陆杰

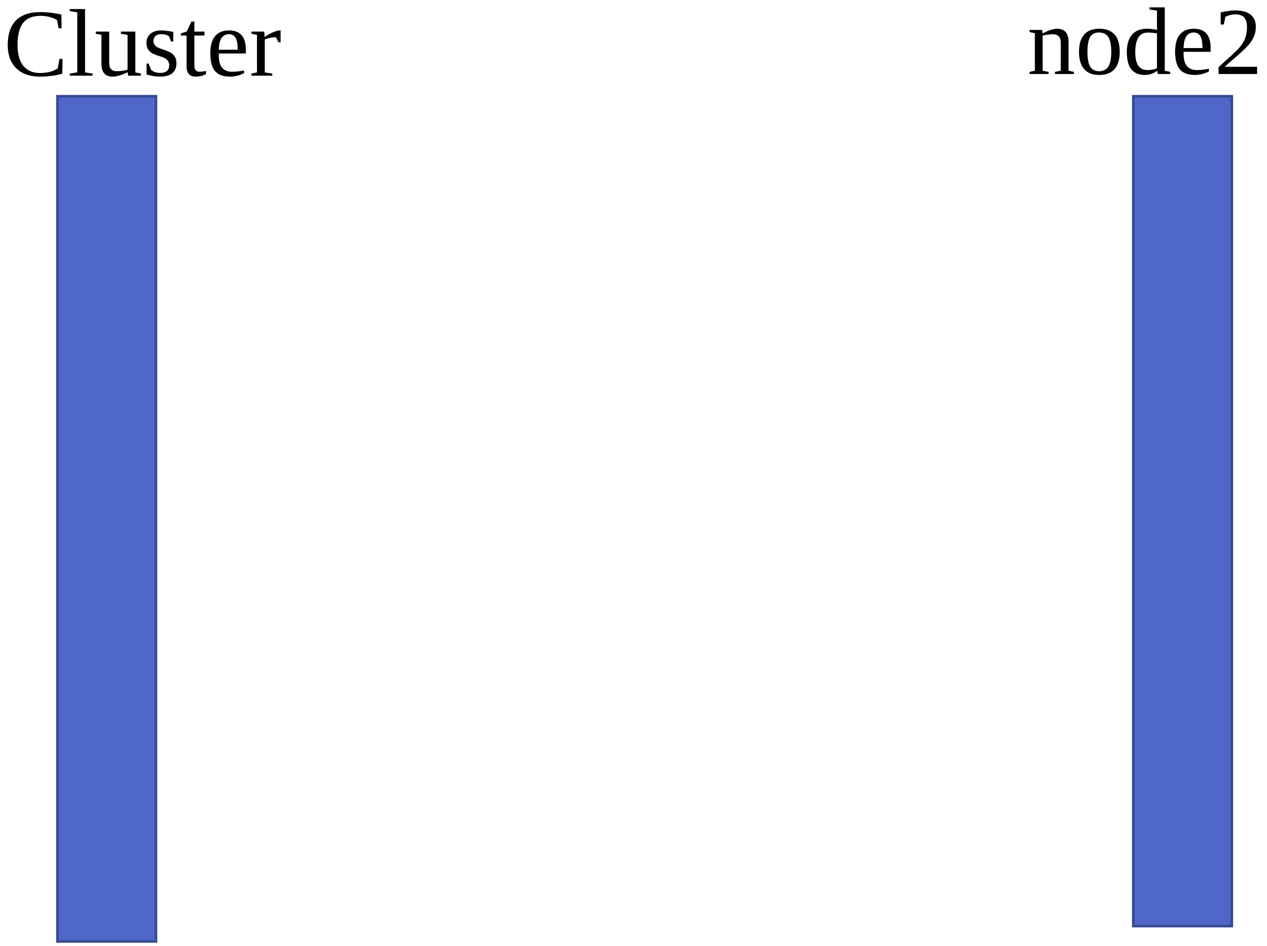
指导老师：李炼（研究员，百人计划）

崩溃恢复缺陷

崩溃恢复缺陷

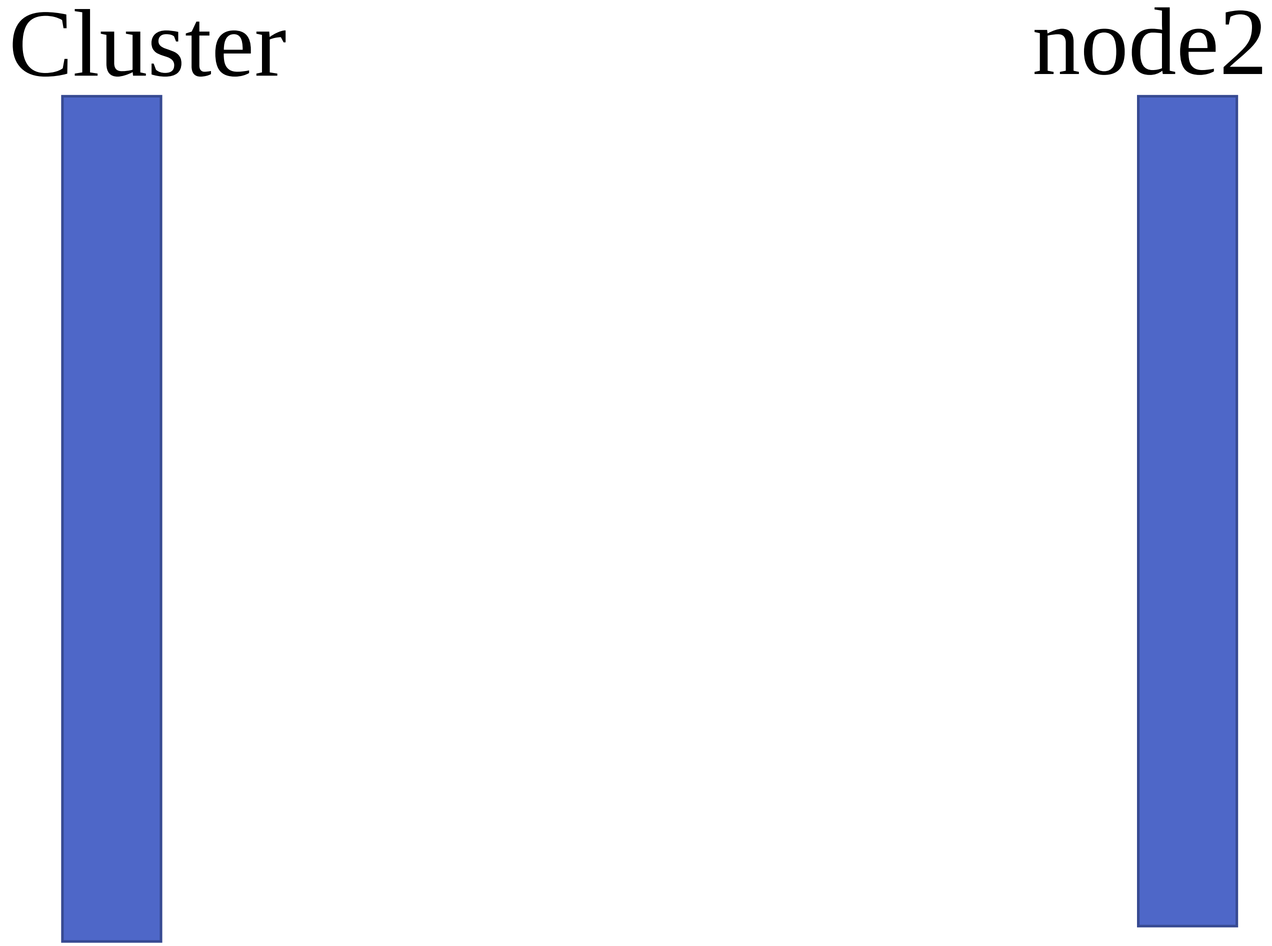


崩溃恢复缺陷

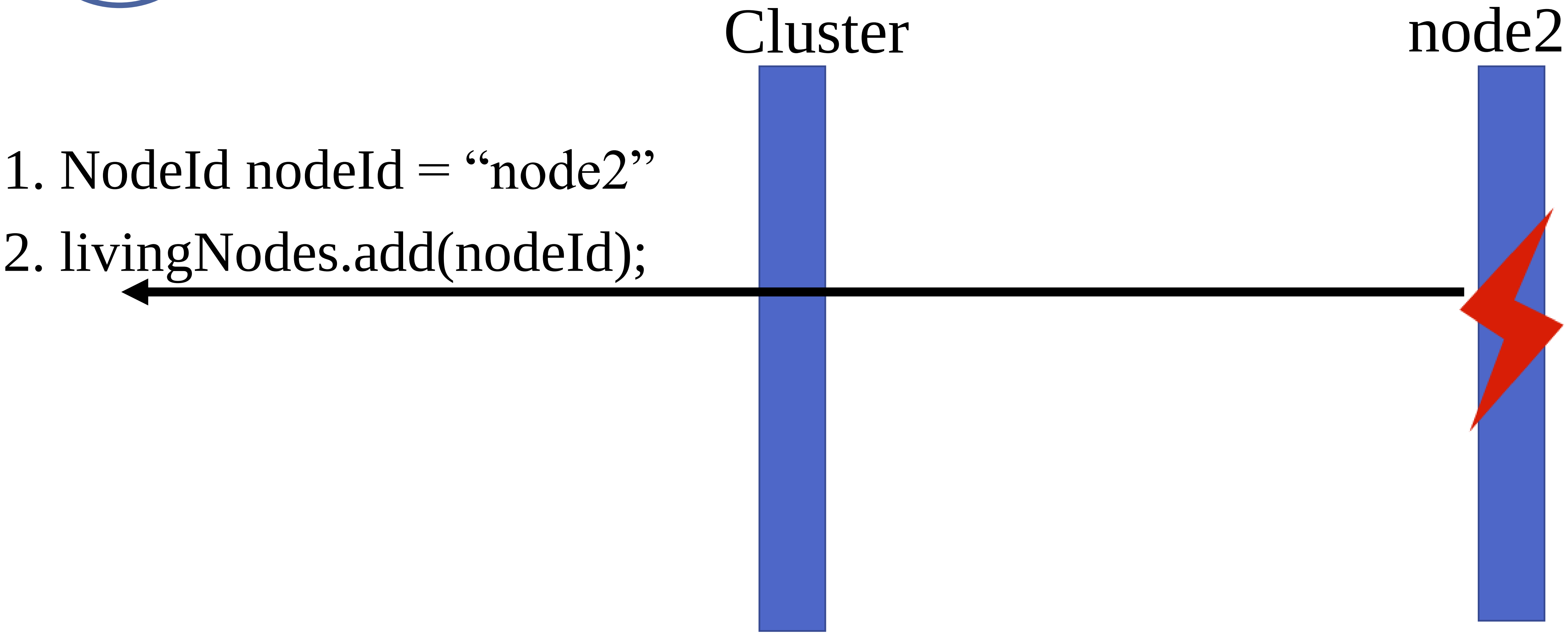


崩溃恢复缺陷

1. NodeId nodeId = “node2”
2. livingNodes.add(nodeId);

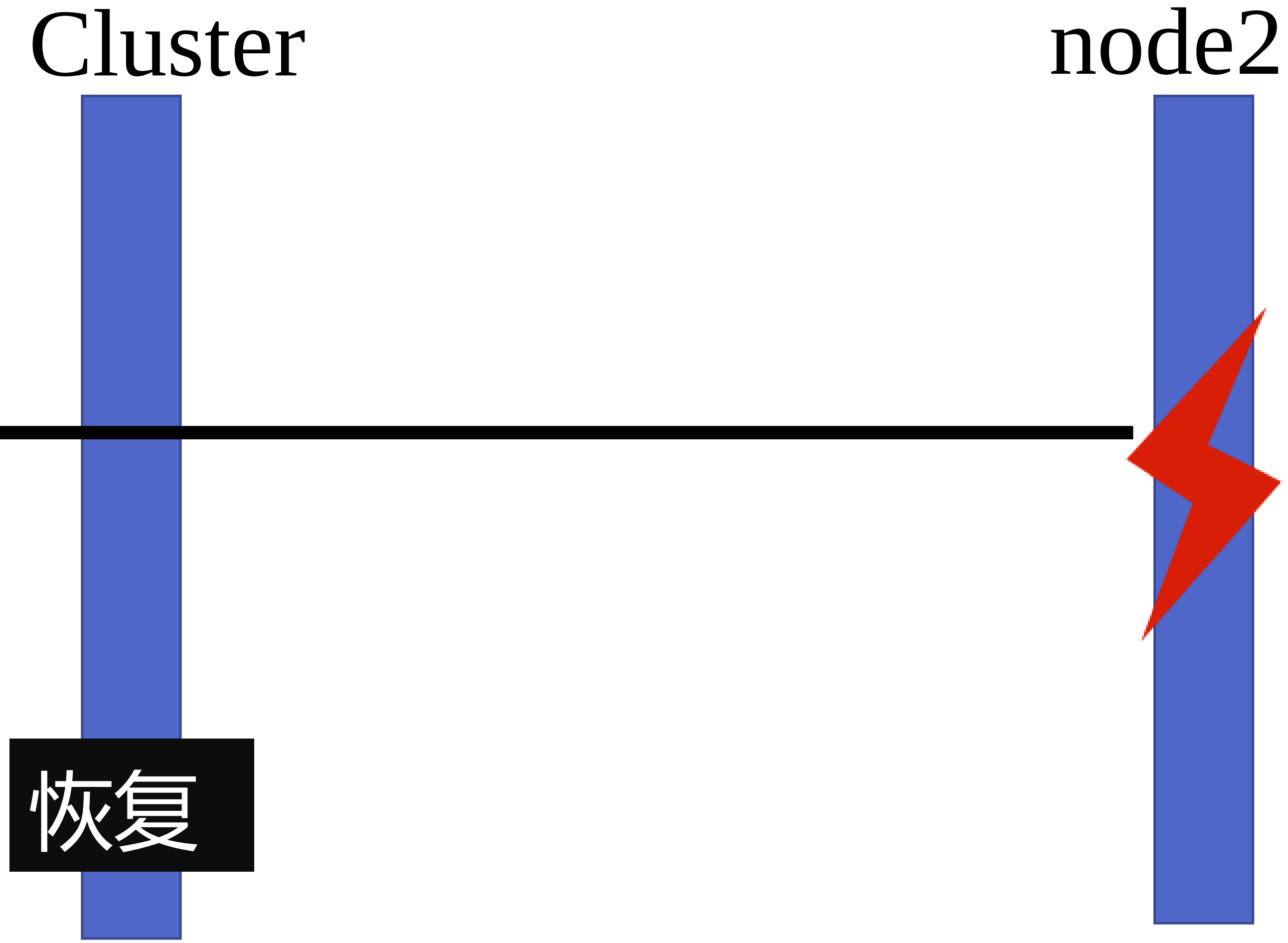


崩溃恢复缺陷


Cluster
node2
1. NodeId nodeId = “node2”
2. livingNodes.add(nodeId);

崩溃恢复缺陷

1. NodeId nodeId = “node2”
2. livingNodes.add(nodeId);



livingNodes.remove(nodeId);

崩溃恢复缺陷

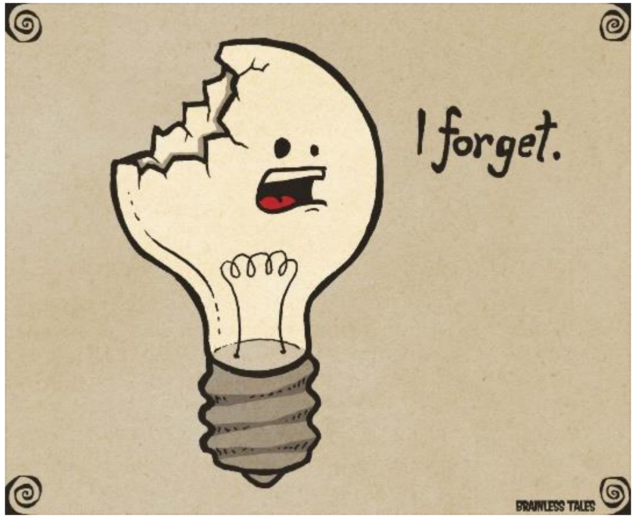
Cluster

node2

1. NodeId nodeId = “node2”
2. livingNodes.add(nodeId);

livingNodes.remove(nodeId);

恢复



崩溃恢复缺陷

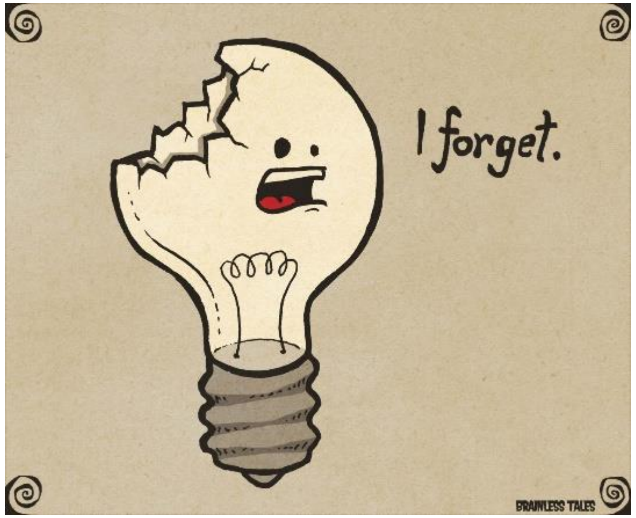
Cluster

node2

1. NodeId nodeId = “node2”
2. livingNodes.add(nodeId);

```
while(livingNodes.has(“node2”)) {
    retryConnet(“node2”);
}
livingNodes.remove(nodeId);
```

恢复



崩溃恢复缺陷

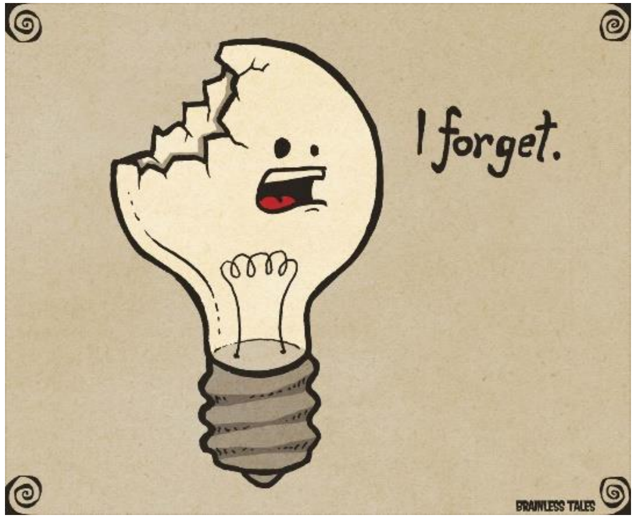
Cluster

node2

1. NodeId nodeId = “node2”
2. livingNodes.add(nodeId);

```
while(livingNodes.has(“node2”))  {
    retryConnet(“node2”);
}
livingNodes.remove(nodeId);
```

恢复



崩溃恢复缺陷

写之后(post-write)

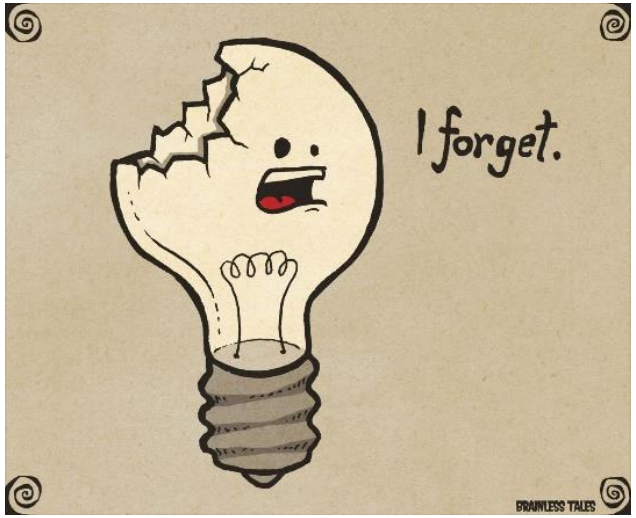
Cluster

node2

1. NodeId nodeId = “node2”
2. livingNodes.add(nodeId);

while(livingNodes.has(“node2”)) {
    retryConnet(“node2”);
}
livingNodes.remove(nodeId);

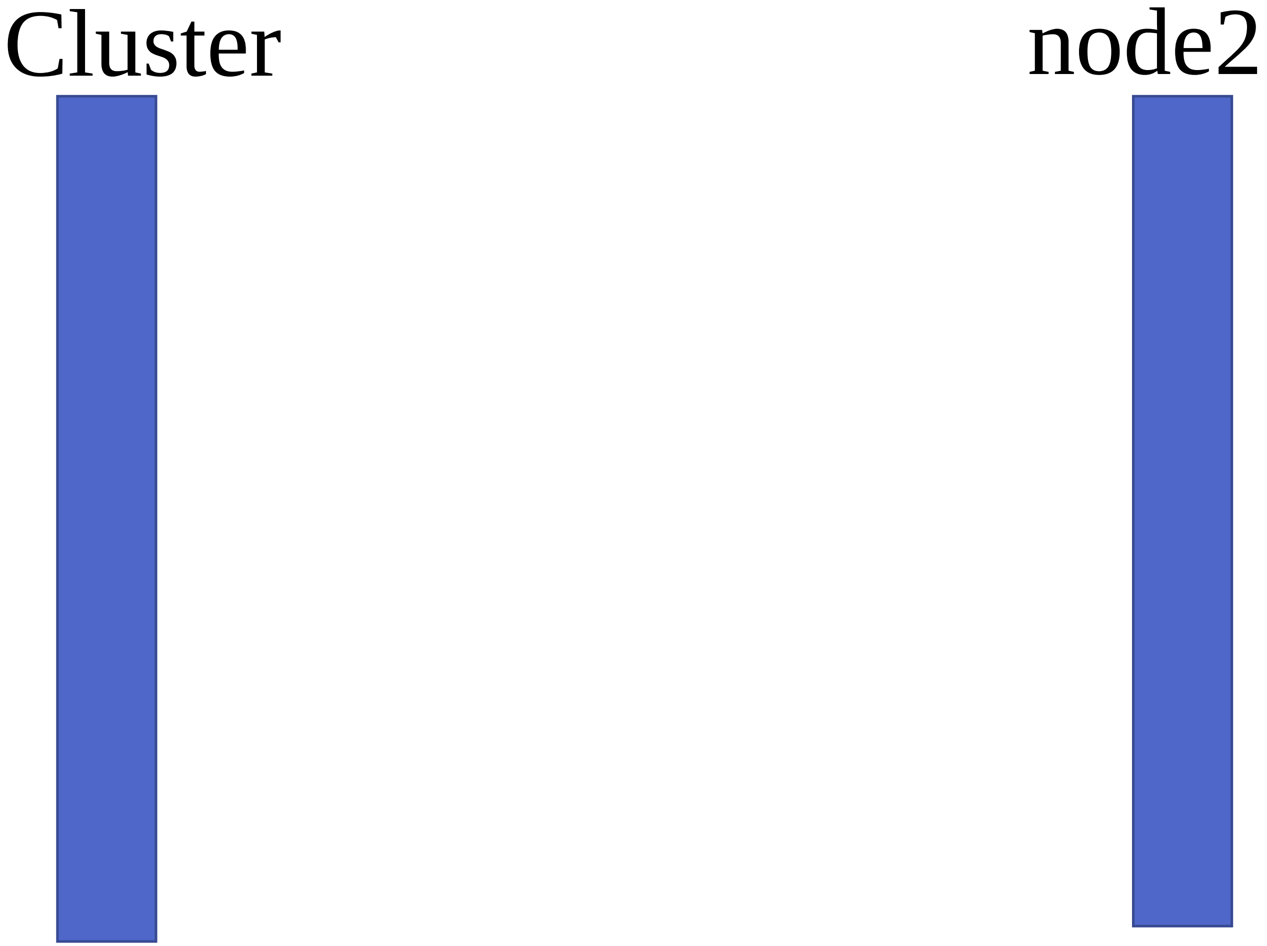
恢复



崩溃恢复缺陷

int length = livingNodes.size();
(资源调度)

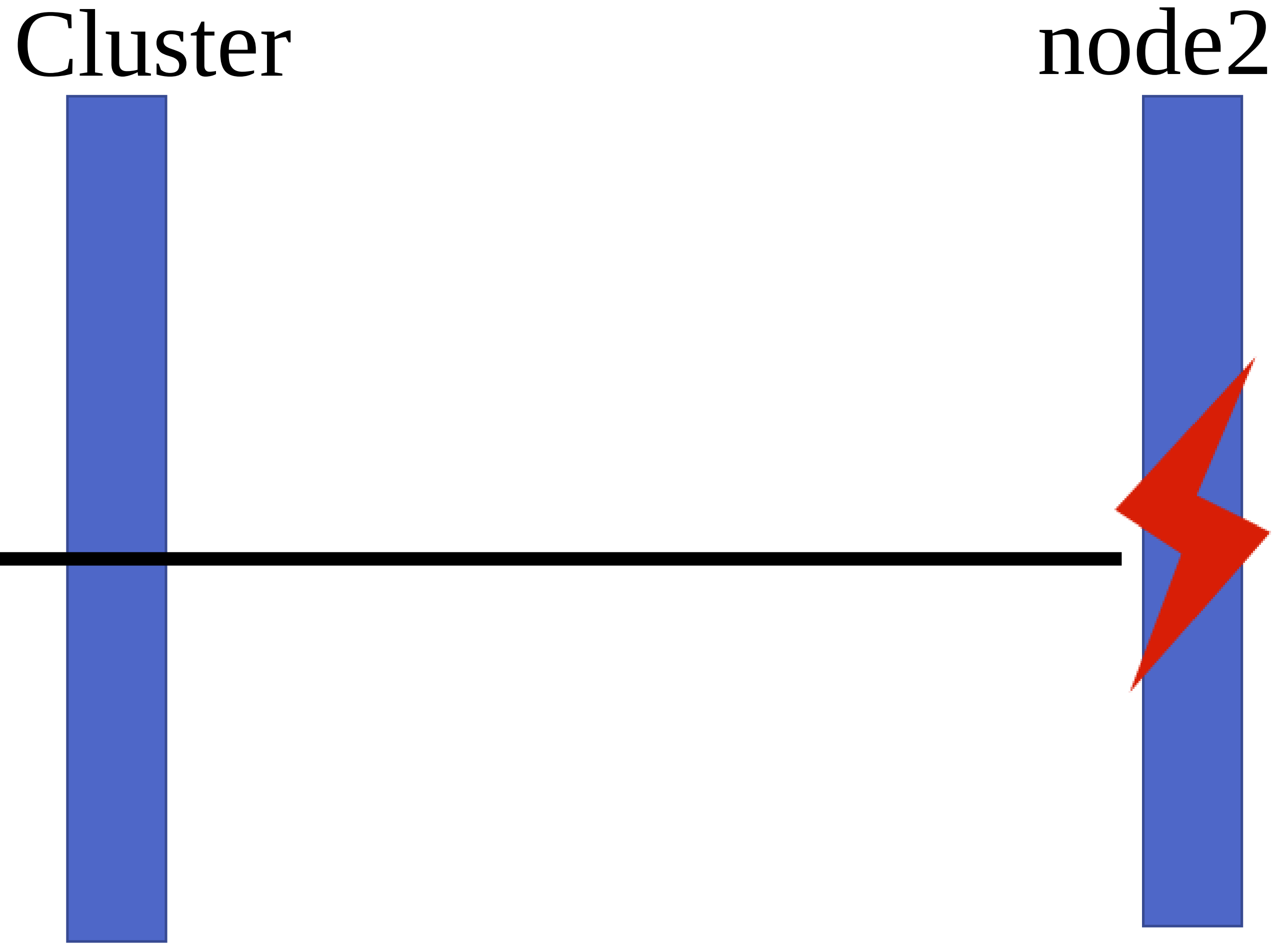
livingNodes.get(length-1);



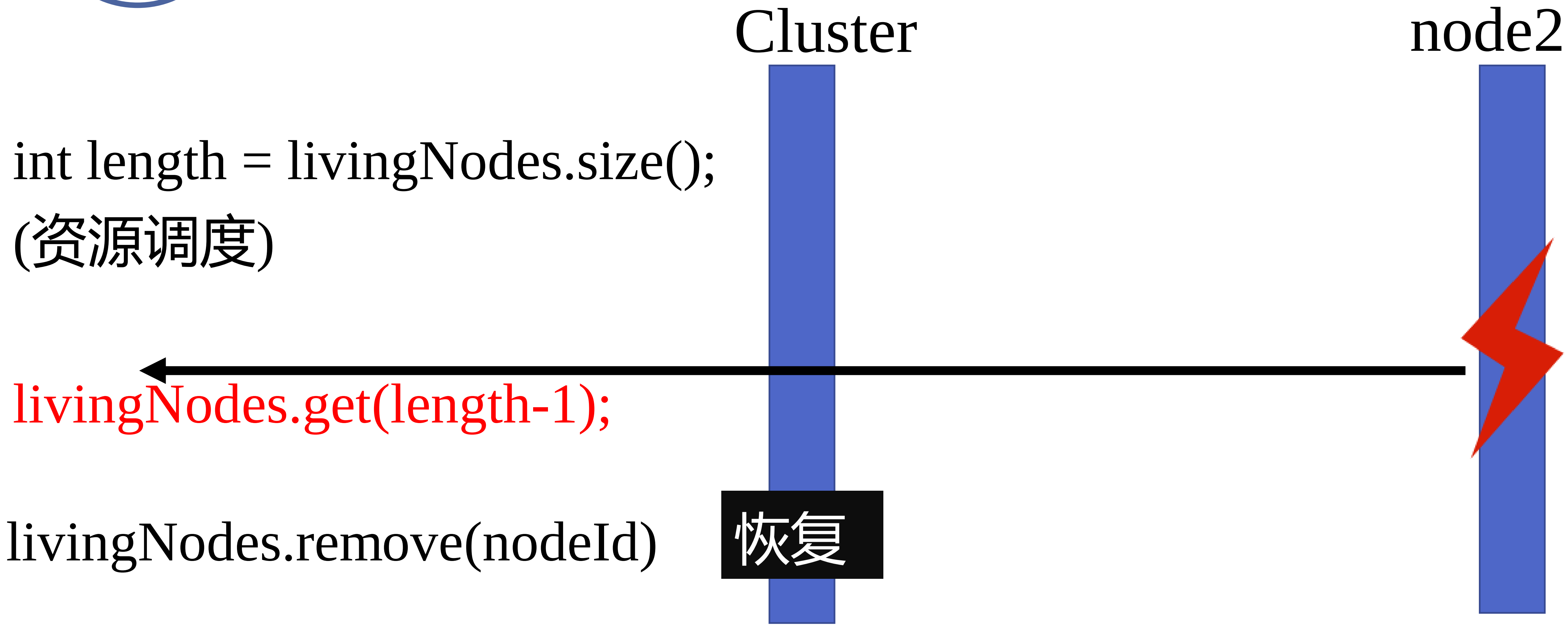
崩溃恢复缺陷

int length = livingNodes.size();
(资源调度)

livingNodes.get(length-1);



崩溃恢复缺陷



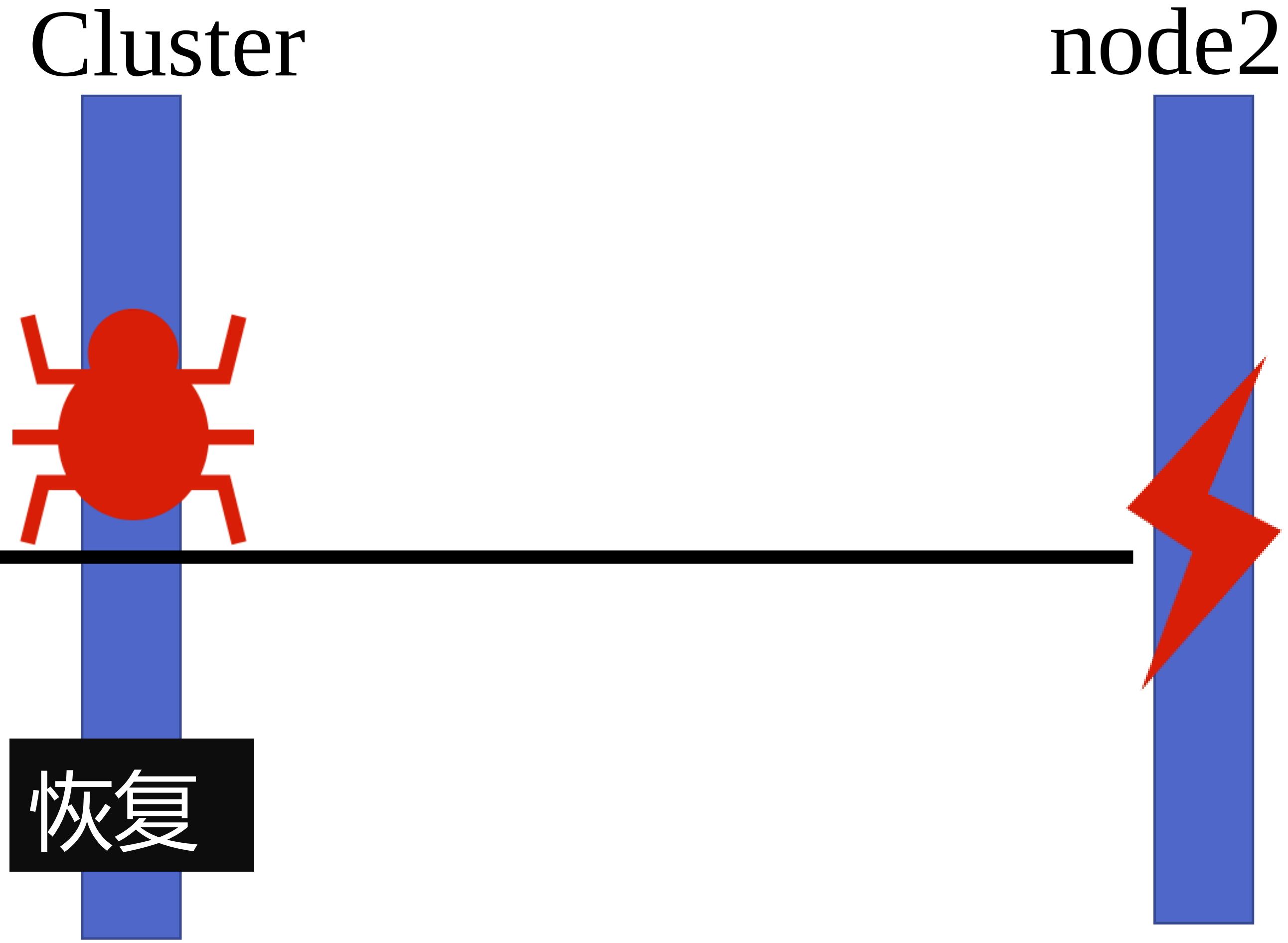
Cluster
node2
int length = livingNodes.size();
(资源调度)
livingNodes.get(length-1);
livingNodes.remove(nodeId)
恢复

崩溃恢复缺陷

int length = livingNodes.size();
(资源调度)

livingNodes.get(length-1);

livingNodes.remove(nodeId)



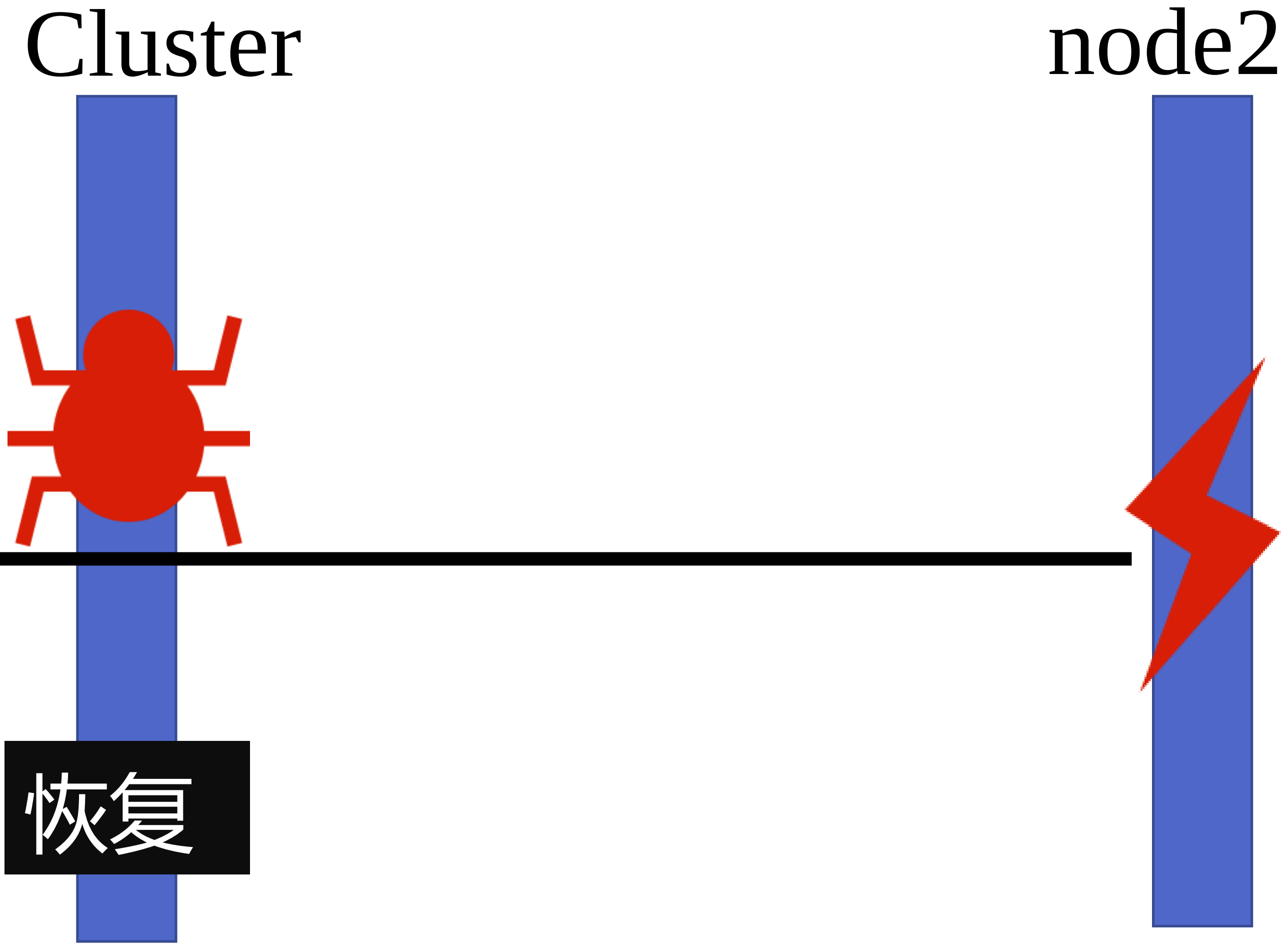
崩溃恢复缺陷

读之前(pre-read)

int length = livingNodes.size();
(资源调度)

livingNodes.get(length-1);

livingNodes.remove(nodeId)



崩溃恢复缺陷

崩溃恢复缺陷两种模式：
元信息变量的
读之前，写之后

计算技术研究所

分布式系统 崩溃 恢复 缺陷 检测

演讲人：陆杰

指导老师：李炼（研究员，百人计划）

检测设计

1.识别所有的元信息变量

检测设计

1.识别所有的元信息变量

元信息变量

1. NodeId nodeId = “node2”
2. livingList.add(nodeId);

写或读

node1

2.崩溃点

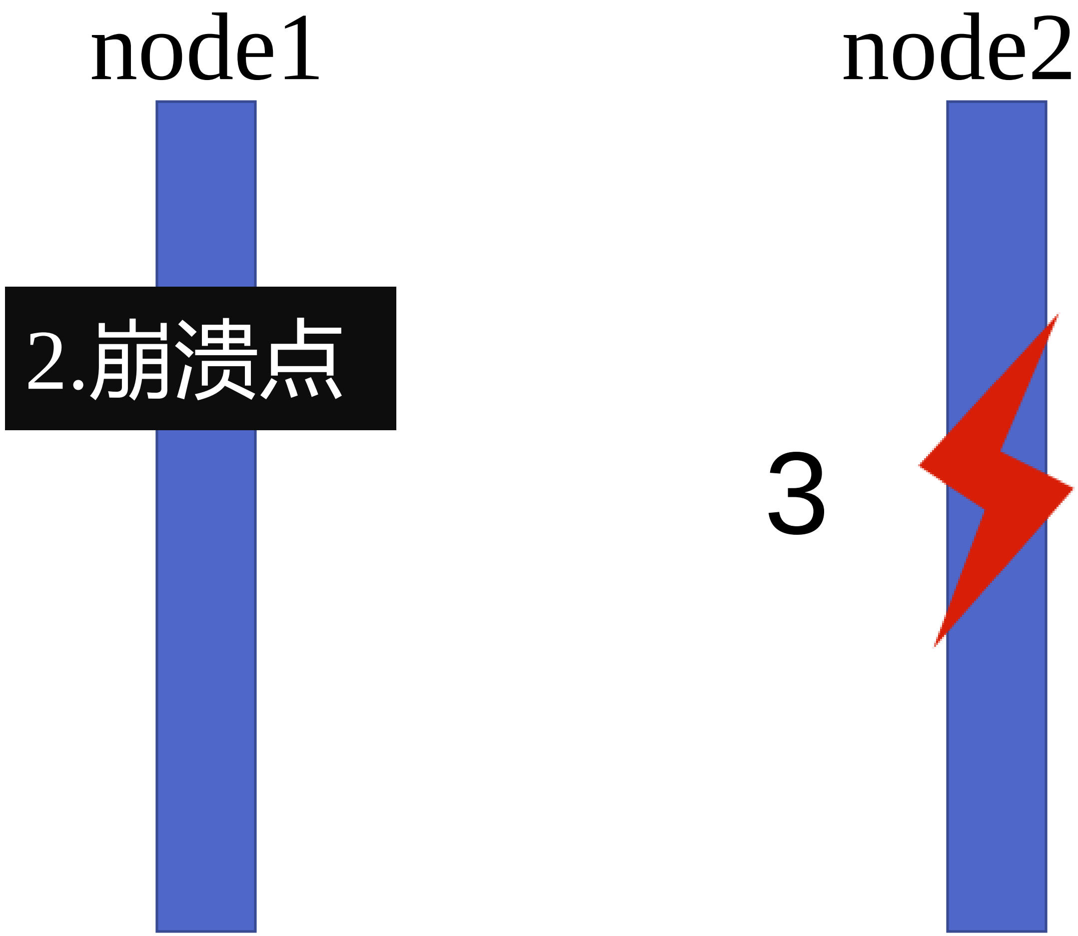
检测设计

1.识别所有的元信息变量

元信息变量

1. NodeId nodeId = “node2”
2. livingList.add(nodeId);

写或读



检测设计

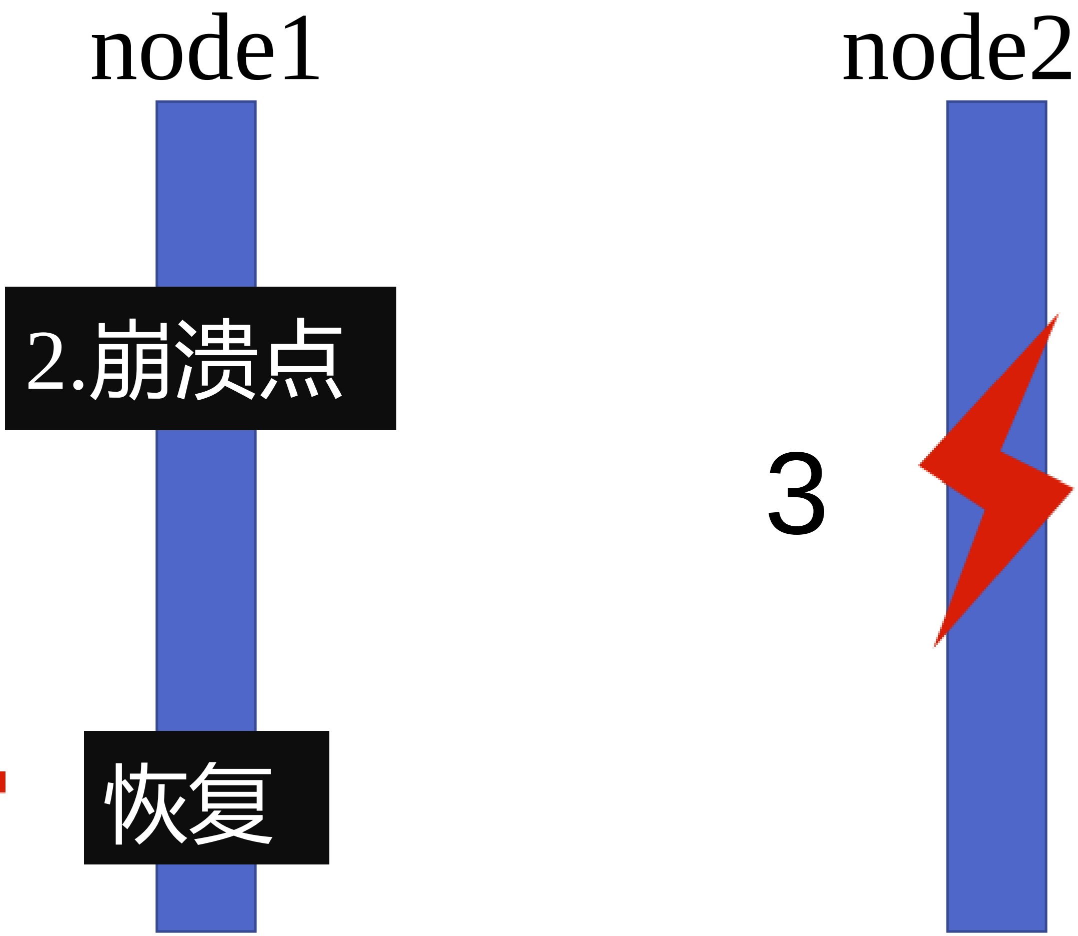
1.识别所有的元信息变量

元信息变量

1. NodeId nodeId = “node2”
2. livingList.add(nodeId);

写或读

……………………



识别元信息变量

节点变量是最基础的元信息变量

```
public void registerNode(NodeId nodeId) {
    LOG.info("registering node " + nodeId);
}
```

10.5.1.11
节点的IP

识别元信息变量

和元信息变量相关的变量也是元信息变量

```
public void assignTask(NodeId nodeId,
    ContainerId containerId) {
    LOG.info("Using container " + containerId +
                                "on node" + nodeId);
}
```

元信息变量

元信息变量

识别元信息变量

元信息

```
public void registerNode(NodeId nodeId) {
     Log.info("registering node" + nodeId);
     RPCfun(nodeId);
}
public void RPCfun(NodeId nodeId) {
     livingNodes.add(nodeId);
}
```

识别元信息变量

元信息

```
public void registerNode(NodeId nodeId) {
    Log.info("registering node" + nodeId);
    RPCfun(nodeId);
}
public void RPCfun(NodeId nodeId) {
    livingNodes.add(nodeId);
}
```

识别元信息变量

元信息

```
public void registerNode(NodeId nodeId) {
    Log.info("registering node" + nodeId);
    RPCfun(nodeId);
}
public void RPCfun(NodeId nodeId) {
    livingNodes.add(nodeId);
}
```

指针分析

识别元信息变量

元信息

```
public void registerNode(NodeId nodeId) {
    Log.info("registering node" + nodeId);
    RPCfun(nodeId);
}
public void RPCfun(NodeId nodeId) {
    livingNodes.add(nodeId);
}
```

指针分析

元信息变量

识别元信息变量

元信息

```
public void registerNode(NodeId nodeId) {
      Log.info("registering node" + nodeId);
      RPCfun(nodeId);
}
public void RPCfun(NodeId nodeId) {
      livingNodes.add(nodeId);
}
```

元信息变量

识别元信息变量

元信息

```java
public void registerNode(NodeId nodeId) {
    Log.info("registering node" + nodeId);
    RPCfun(nodeId);
}
public void RPCfun(NodeId nodeId) {
    livingNodes.add(nodeId);
}
```

类型分析

元信息变量

```java
List<NodeId> livingNodes = null;
```

细节

SOSP 2019

CrashTuner: Detecting Crash-Recovery Bugs in Cloud Systems via Meta-Info Analysis

Jie Lu
lujie@ict.ac.cn
SKL Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences, China

Chen Liu
liuchen17z@ict.ac.cn
SKL Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences, China

Lian Li*†
lianli@ict.ac.cn
SKL Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences, China

Xiaobing Feng
fxb@ict.ac.cn
SKL Computer Architecture, ICT, CAS
University of Chinese Academy of Sciences, China

Feng Tan
Jun Yang
Liang You
{tanfeng.tf,muzhuo.yj,youliang.yl}@alibaba.com
Alibaba Group

Abstract

Crash-recovery bugs (bugs in crash-recovery-related mechanisms) are among the most severe bugs in cloud systems and can easily cause system failures. It is notoriously difficult to detect crash-recovery bugs since these bugs can only be exposed when nodes crash under special timing conditions. This paper presents CrashTuner, a novel fault-injection testing approach to combat crash-recovery bugs. The novelty of CrashTuner lies in how we identify fault-injection points (crash points) that are likely to expose errors. We observe that if a node crashes while accessing *meta-info* variables, i.e., variables referencing high-level system state information (e.g., an instance of node or task), it often triggers crash-recovery bugs. Hence, we identify crash points by automatically inferring meta-info variables via a log-based static program analysis. Our approach is automatic and no manual specification is required.

We have applied CrashTuner to five representative distributed systems: Hadoop2/Yarn, HBase, HDFS, ZooKeeper, and Cassandra. CrashTuner can finish testing each system in 17.39 hours, and reports **21** new bugs that have never been found before. All new bugs are confirmed by the original developers and **16** of them have already been fixed (14 with our patches). These new bugs can cause severe damages such as cluster down or start-up failures.

*corresponding author: lianli@ict.ac.cn
†Also with TianQi Soft Inc., China.

SOSP '19, October 27–30, 2019, Huntsville, ON, Canada

ACM ISBN 978-1-4503-6873-5/19/10…$15.00
https://doi.org/10.1145/3341301.3359645

CCS Concepts • **Software and its engineering → Software testing and debugging; Cloud computing.**

Keywords Crash Recovery Bugs;Fault Tolerance;Distributed Systems;Bug Detection;Fault Injection;Cloud Computing

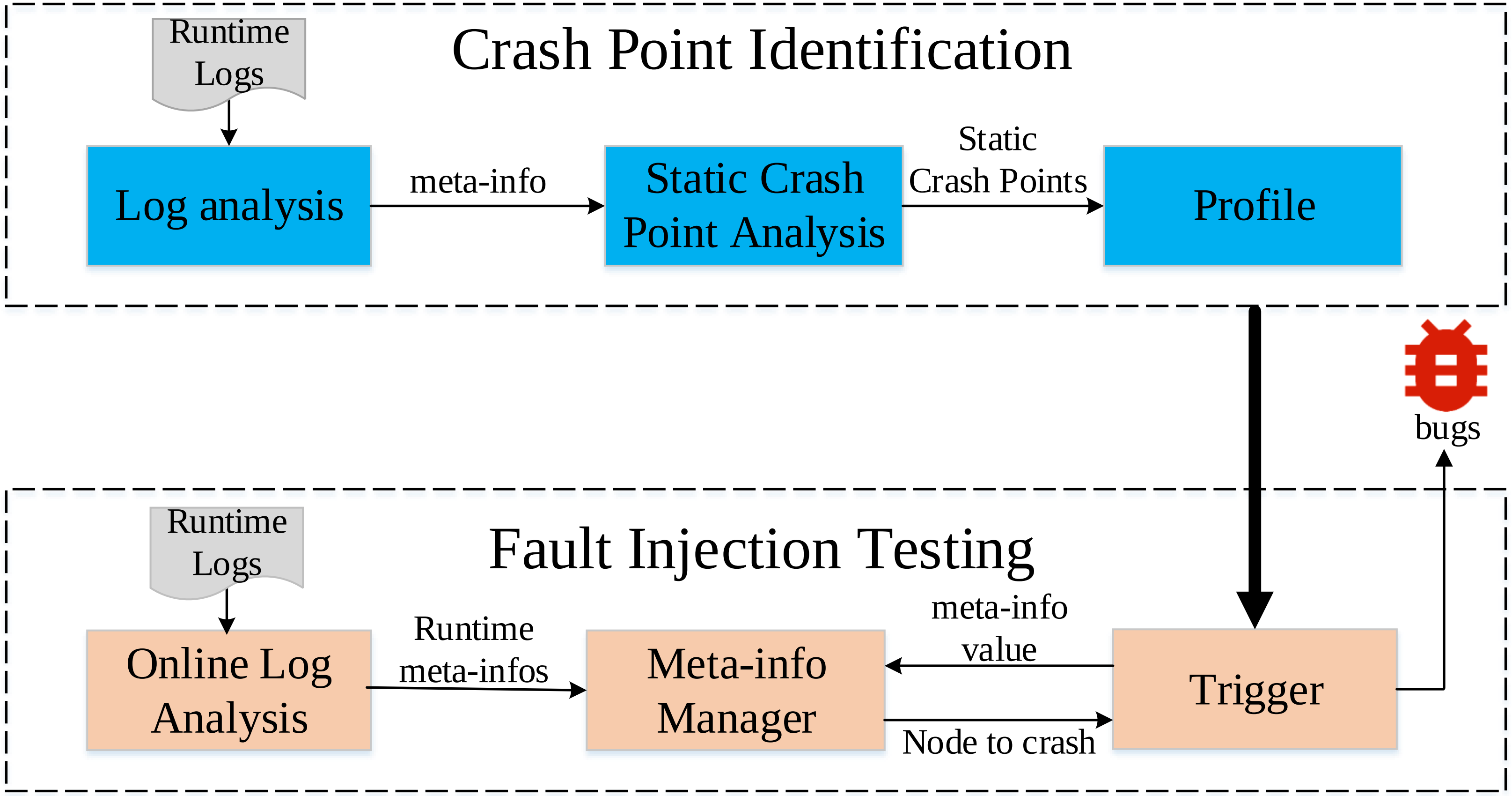
ACM Reference Format:
Jie Lu, Chen Liu, Lian Li, Xiaobing Feng, Feng Tan, Jun Yang, and Liang You. 2019. CrashTuner: Detecting Crash-Recovery Bugs in Cloud Systems via Meta-Info Analysis. In *ACM SIGOPS 27th Symposium on Operating Systems Principles (SOSP '19), October 27–30, 2019, Huntsville, ON, Canada.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3341301.3359645

1 Introduction

Distributed systems have become the backbone of computing in the cloud era. More and more applications are built on top of large-scale distributed systems (such as scalable computing frameworks [20, 57] and distributed storage systems [22, 36]), to provide online services to users. High availability of those systems is crucial: failures of the underlying distributed systems can lead to cloud outage, easily costing service providers millions of dollars [2, 9].

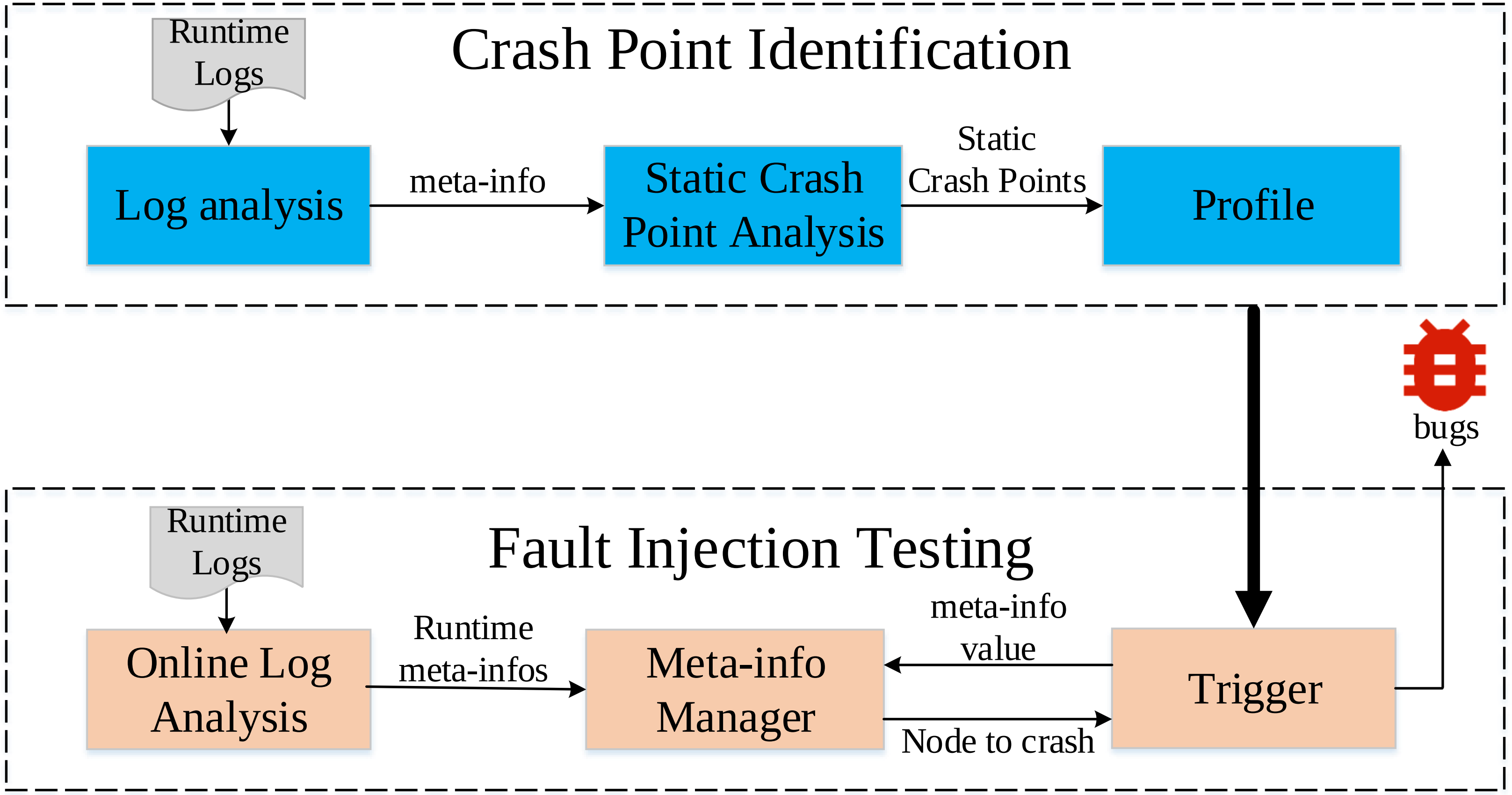
High availability of distributed systems largely hinges on how well these systems tolerate node crashes (failures). Large-scale distributed systems are often comprised of thousands of nodes (machines) [55], and it is common that a node may fail due to hardware or software faults [49]. Although various sophisticated crash-recovery mechanisms [4, 13, 16] have been adopted in distributed systems, it is still challenging to handle node crashes correctly. It is very difficult, if not impossible, for developers to anticipate all possible crash scenarios and correctly implement corresponding recovery mechanisms. In this paper, we refer to bugs in crash-recovery-related mechanisms as *crash-recovery bugs*.

CrashTuner


Crash Point Identification
Runtime Logs
Log analysis
meta-info
Static Crash Point Analysis
Static Crash Points
Profile
bugs
Fault Injection Testing
Runtime Logs
Online Log Analysis
Runtime meta-infos
Meta-info Manager
meta-info value
Node to crash
Trigger

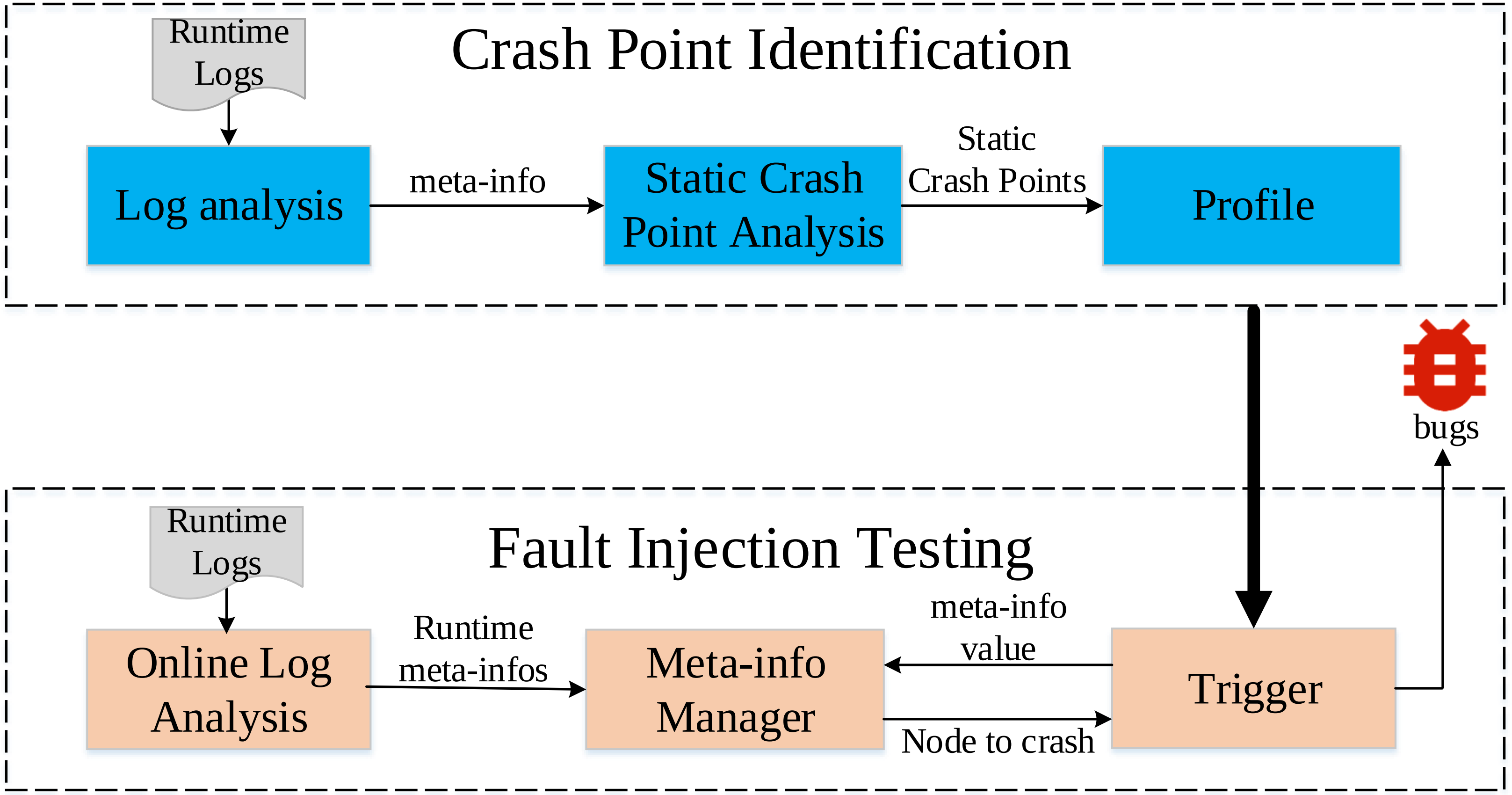
CrashTuner

YARN, HDFS, Cassandra, Hbase, Zookeeper



CrashTuner

YARN, HDFS, Cassandra, Hbase, Zookeeper



17个小时内

21个缺陷

10个critical

16被修复

展示完毕 感谢各位聆听

答辩人：陆杰　　　　指导教师：李炼