Python高级编程

Part-4
分布式与并行计算

Outline

并行编程

- 并行编程模型
- 基于线程的并行
- 基于进程的并行
- 同步
- 死锁问题

分布式

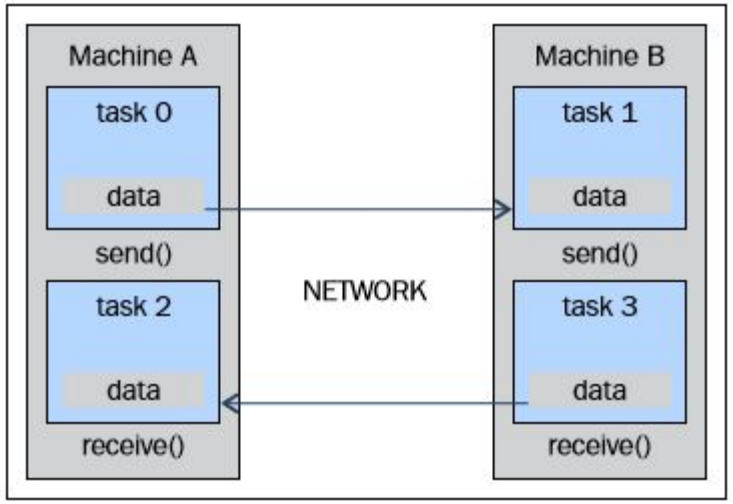
- 分布式理论
- Celery/Ray分布式框架
- 分布式实战

并行编程模型

- 多线程模型
- 多进程模型

并行编程模型

- 消息传递模型

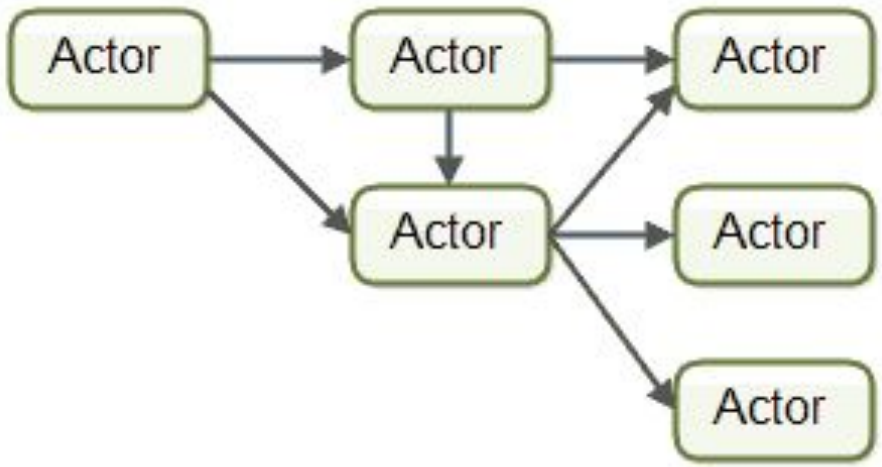


The message passing paradigm model

并行编程模型

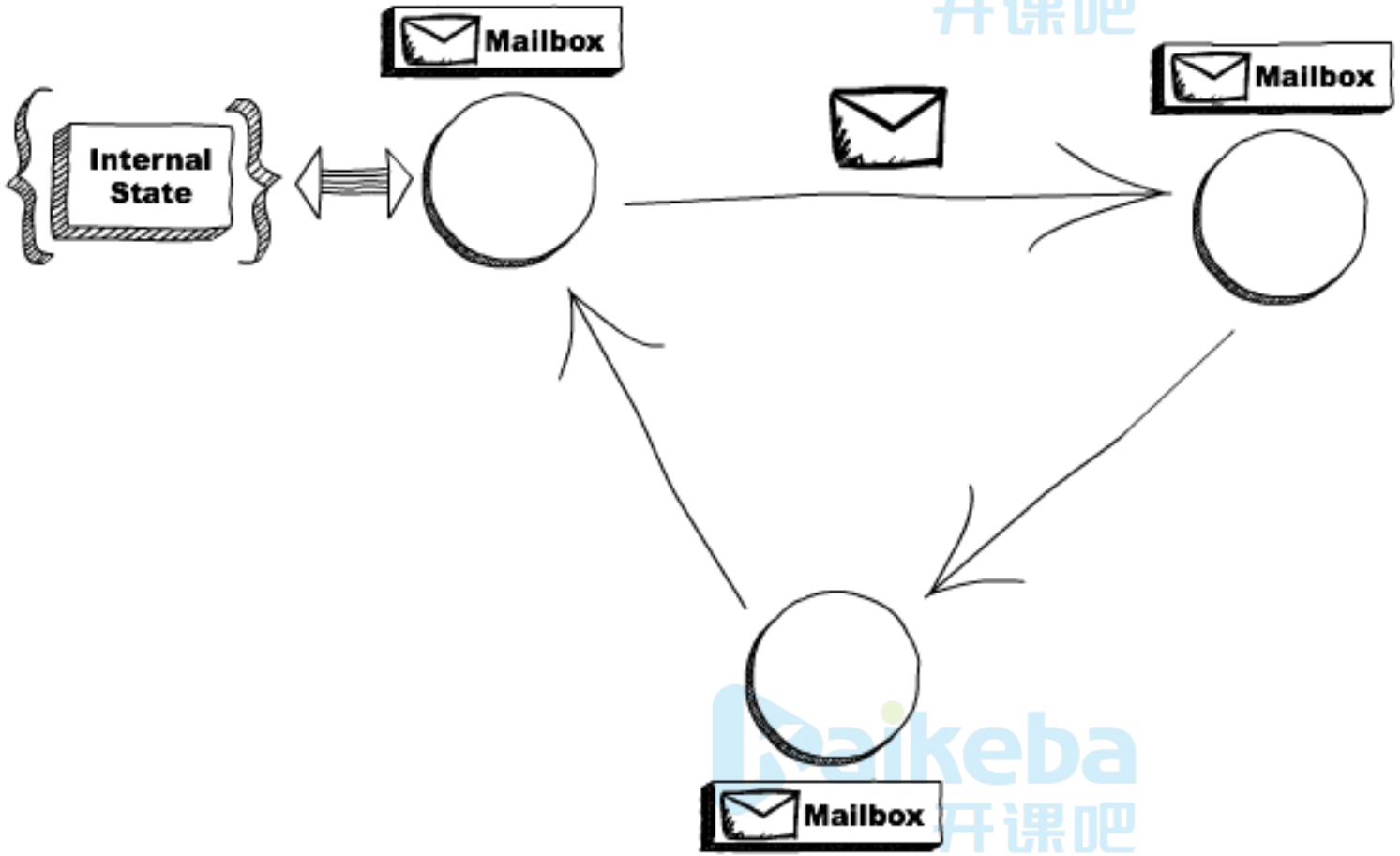
消息传递模型

- Actor



并行编程模型

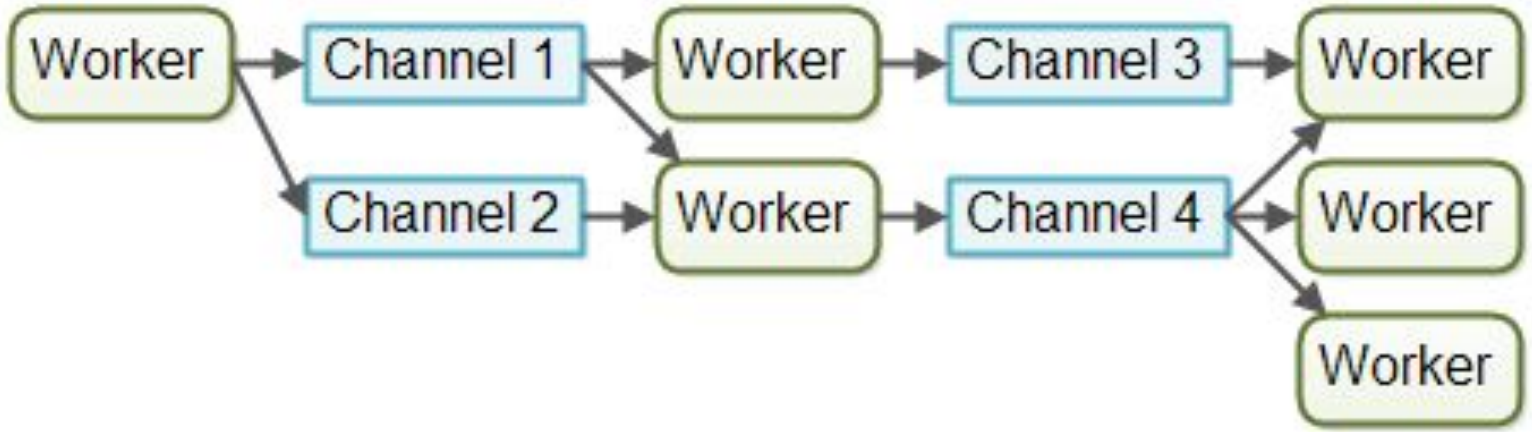
Actors Mailbox



并行编程模型

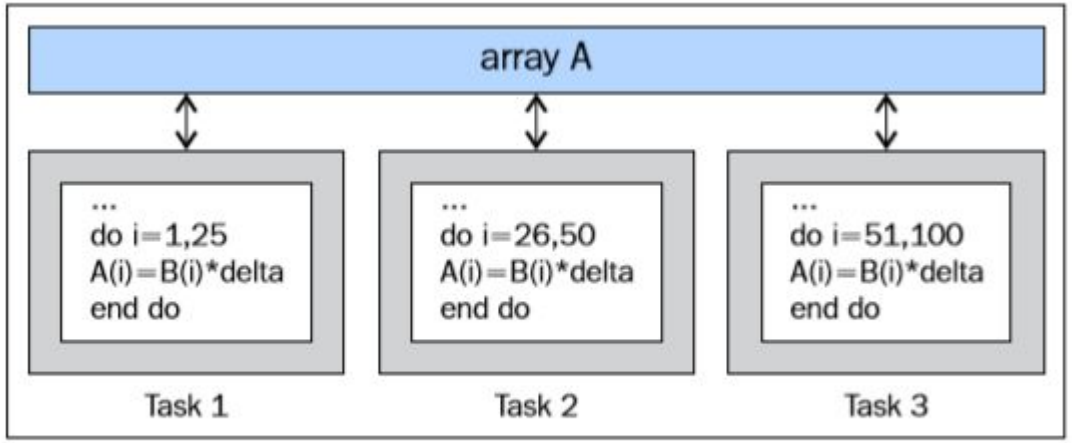
消息传递模型

- CSP



并行编程模型

- 数据并行模型

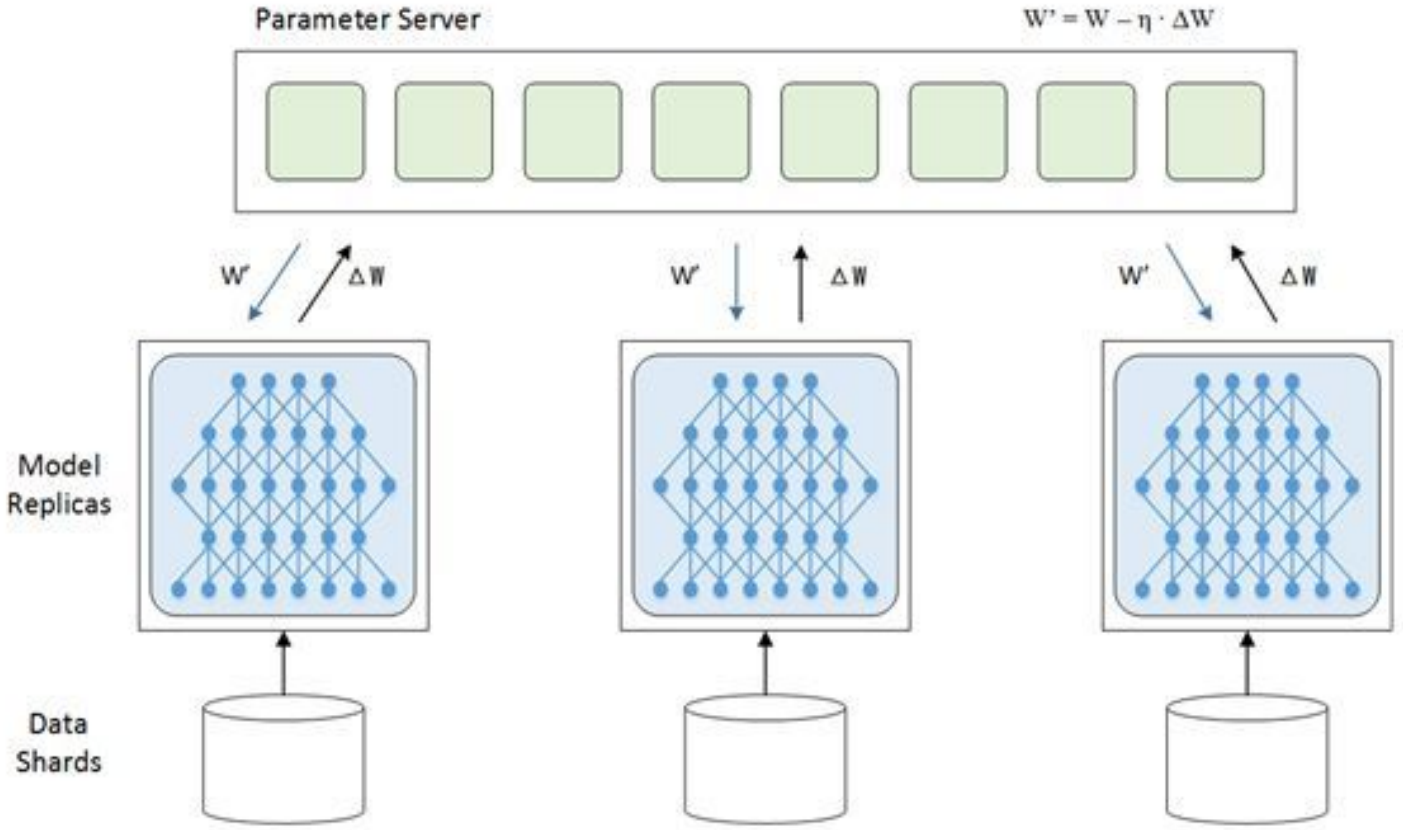


The data parallel paradigm model

并行编程模型

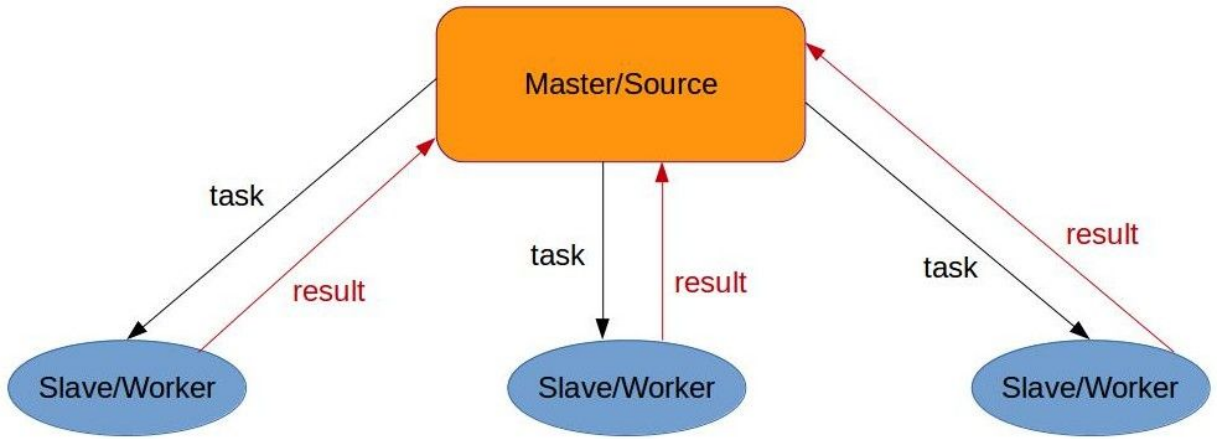
数据并行模型

- 每个节点都有一份模型
- 各个节点取不同的数据，batch_size
- 各个节点完成梯度下降
- 更新参数



并行编程模型

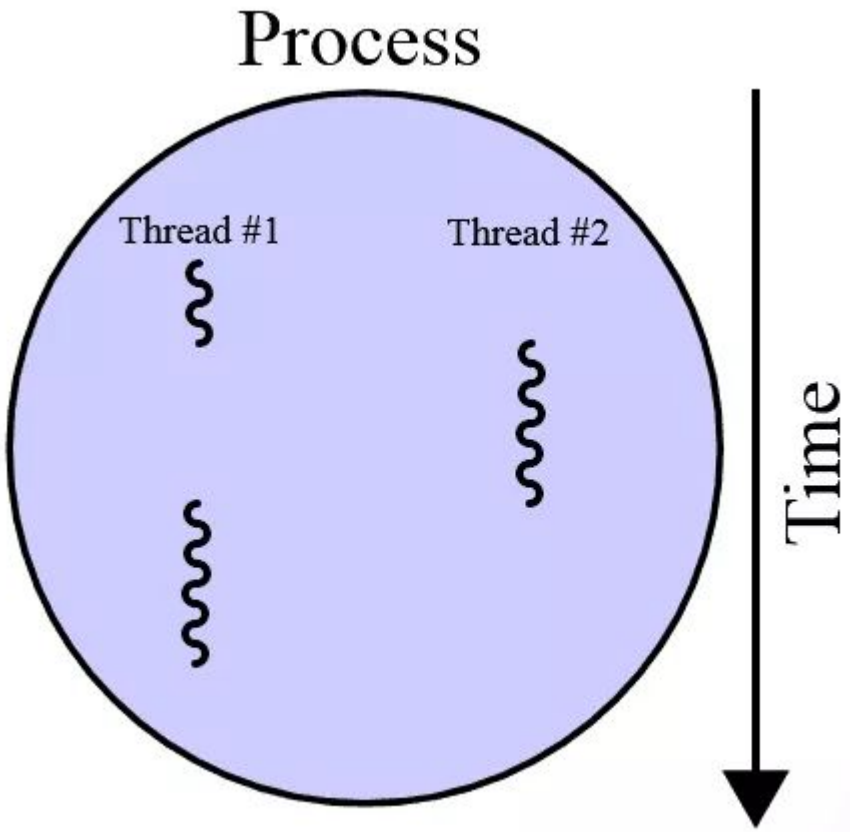
- Task farming(Master-Worker)



Get a message with the task, process the task and send the result to the master

基于线程的并行

- 线程是进程的组件，一个进程中可以有多个线程
- 多个线程共享父进程的内存空间
- 由OS调度
- 例子：浏览器，播放器



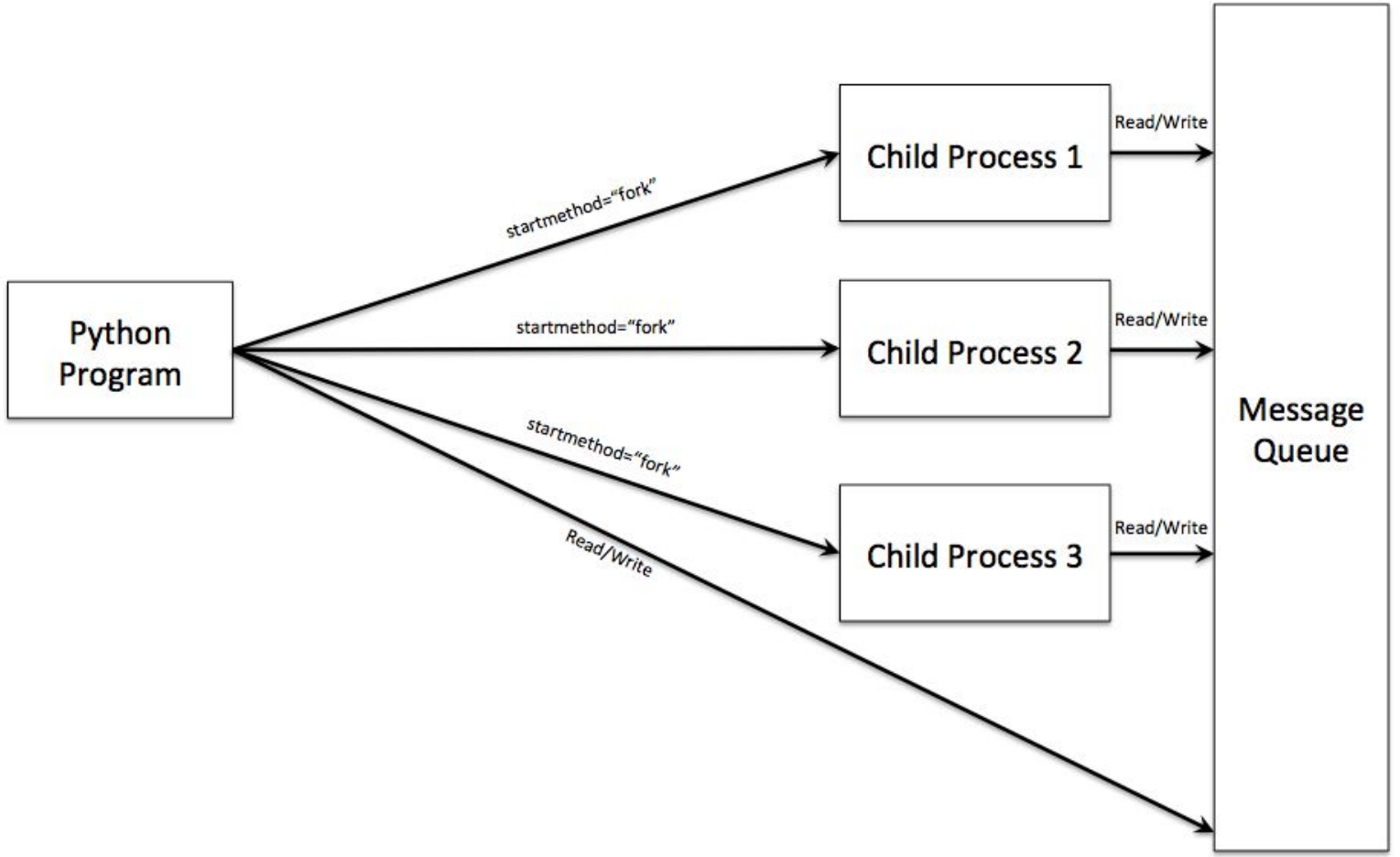
基于线程的并行

“假多线程”

全局解释器锁（GIL）

基于进程的并行

- 拥有独立的内存空间
- 充分利用多核
- 开销比线程大

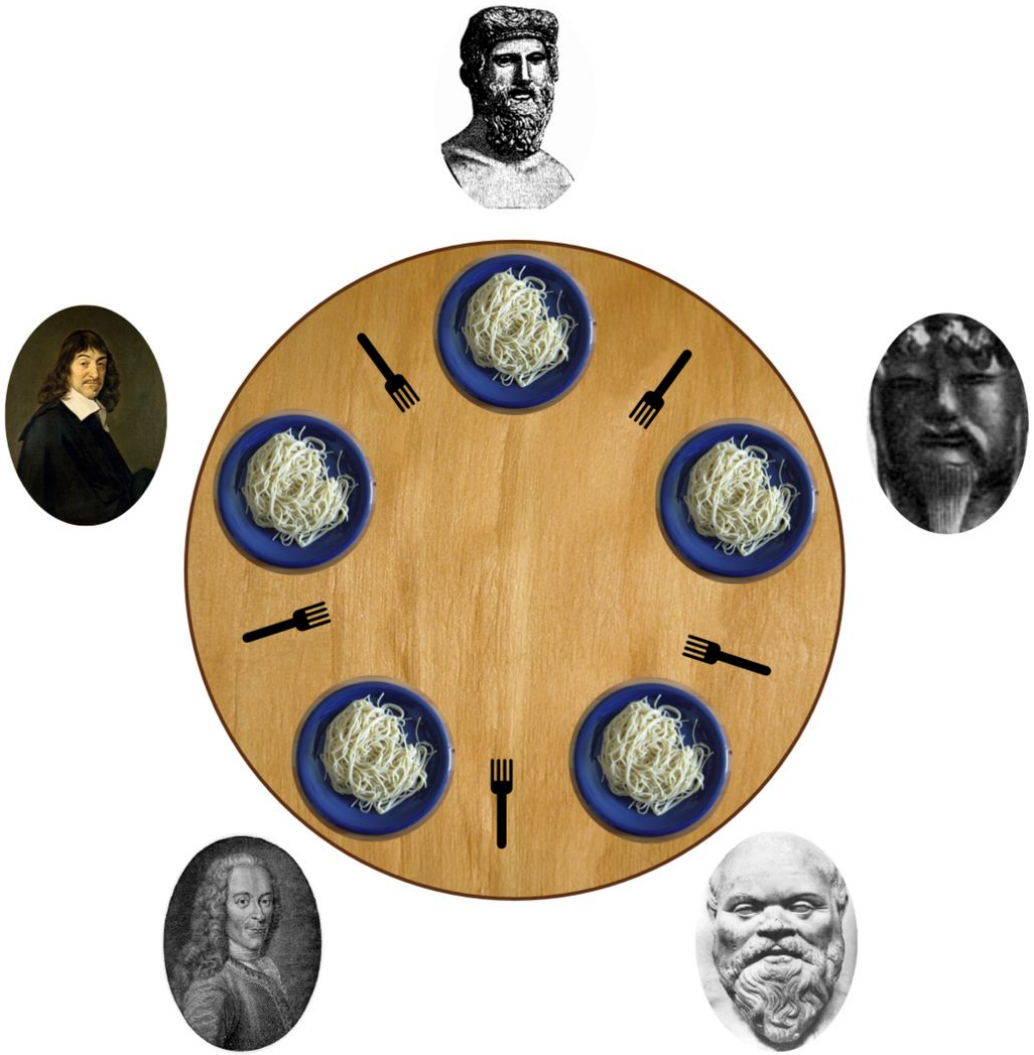


同步问题

- 不管是线程还是进程，都涉及到资源抢占的问题
- 例子：银行账户

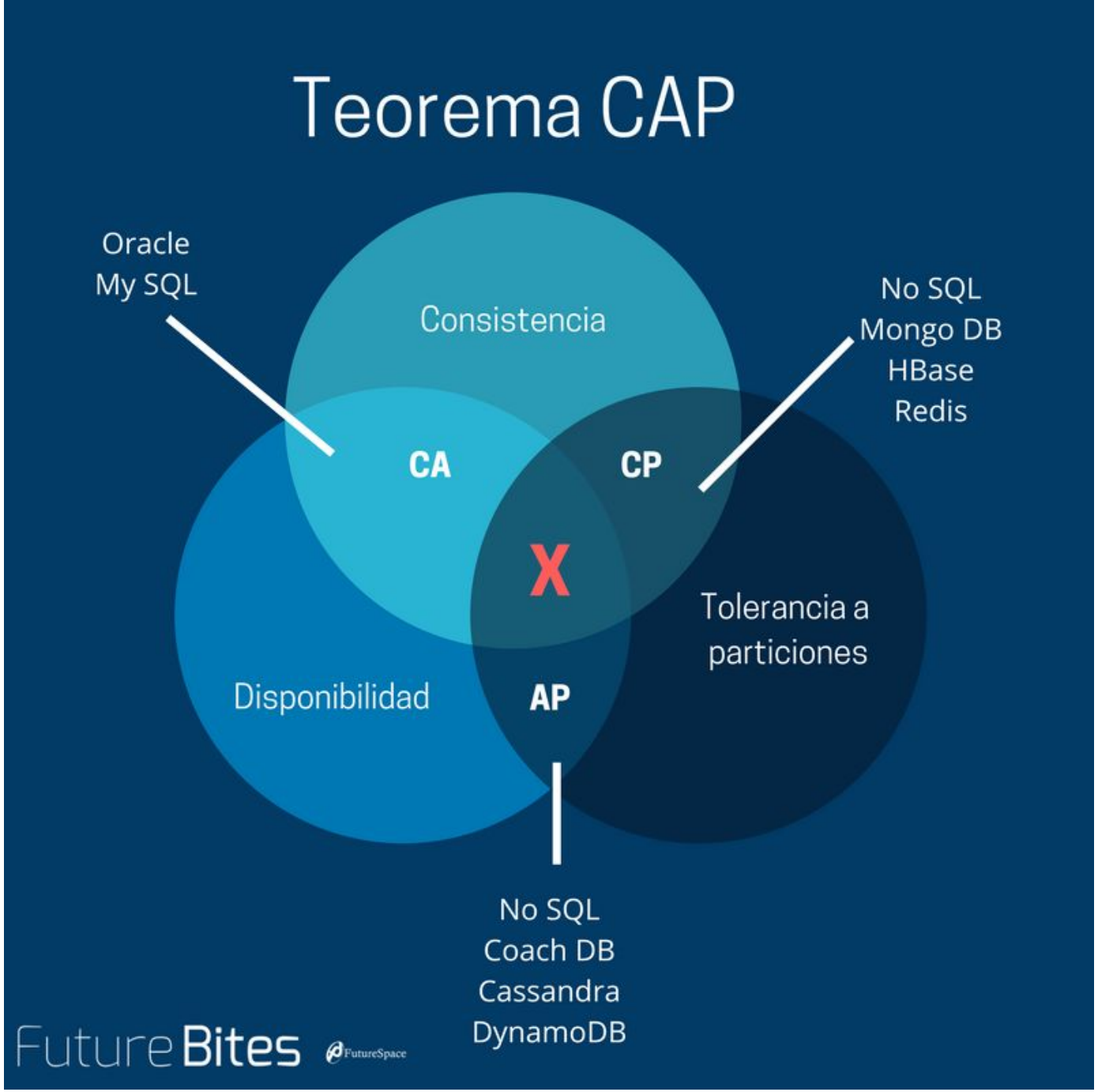
死锁

- 哲学家就餐问题



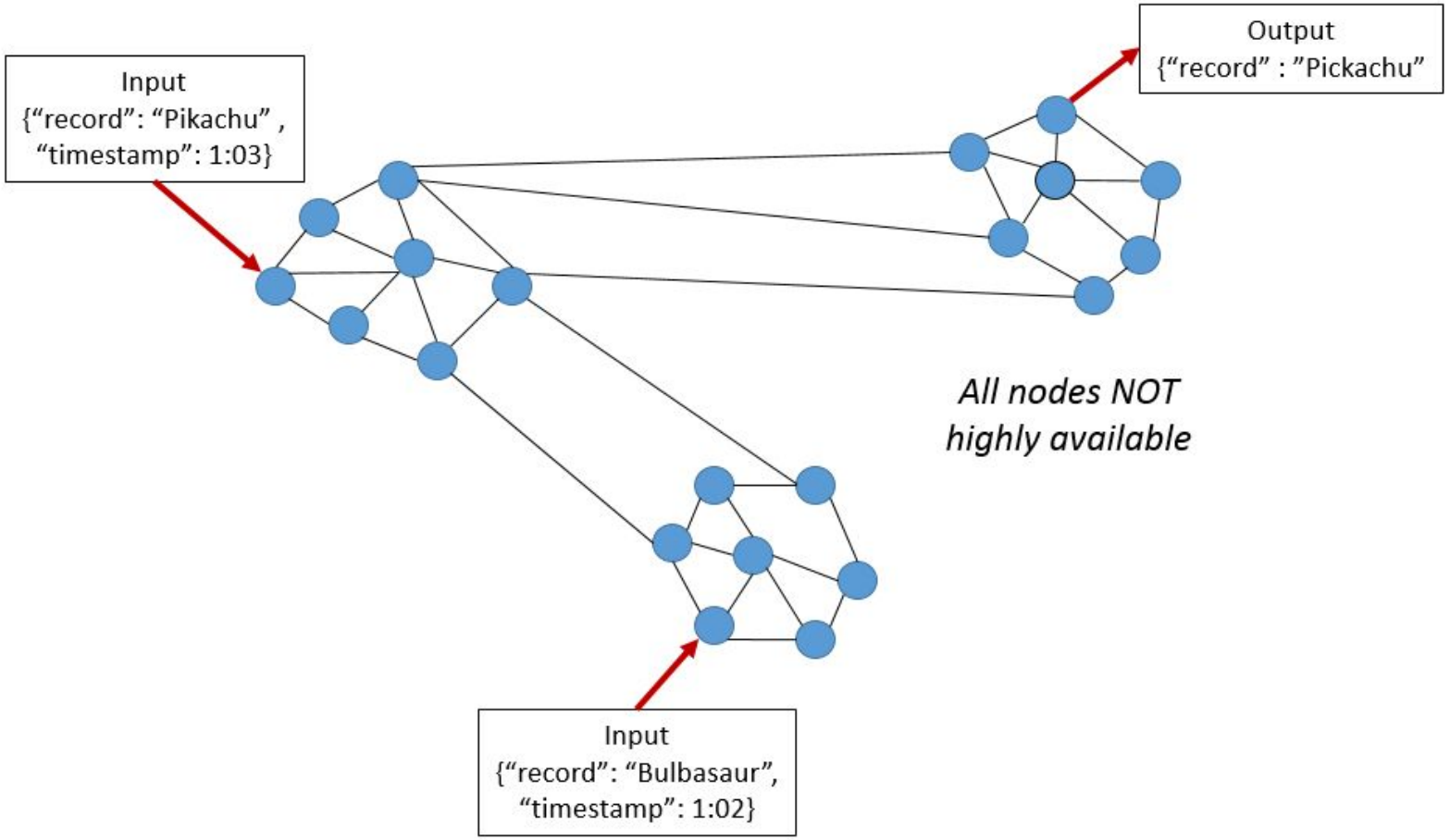
分布式理论

CAP

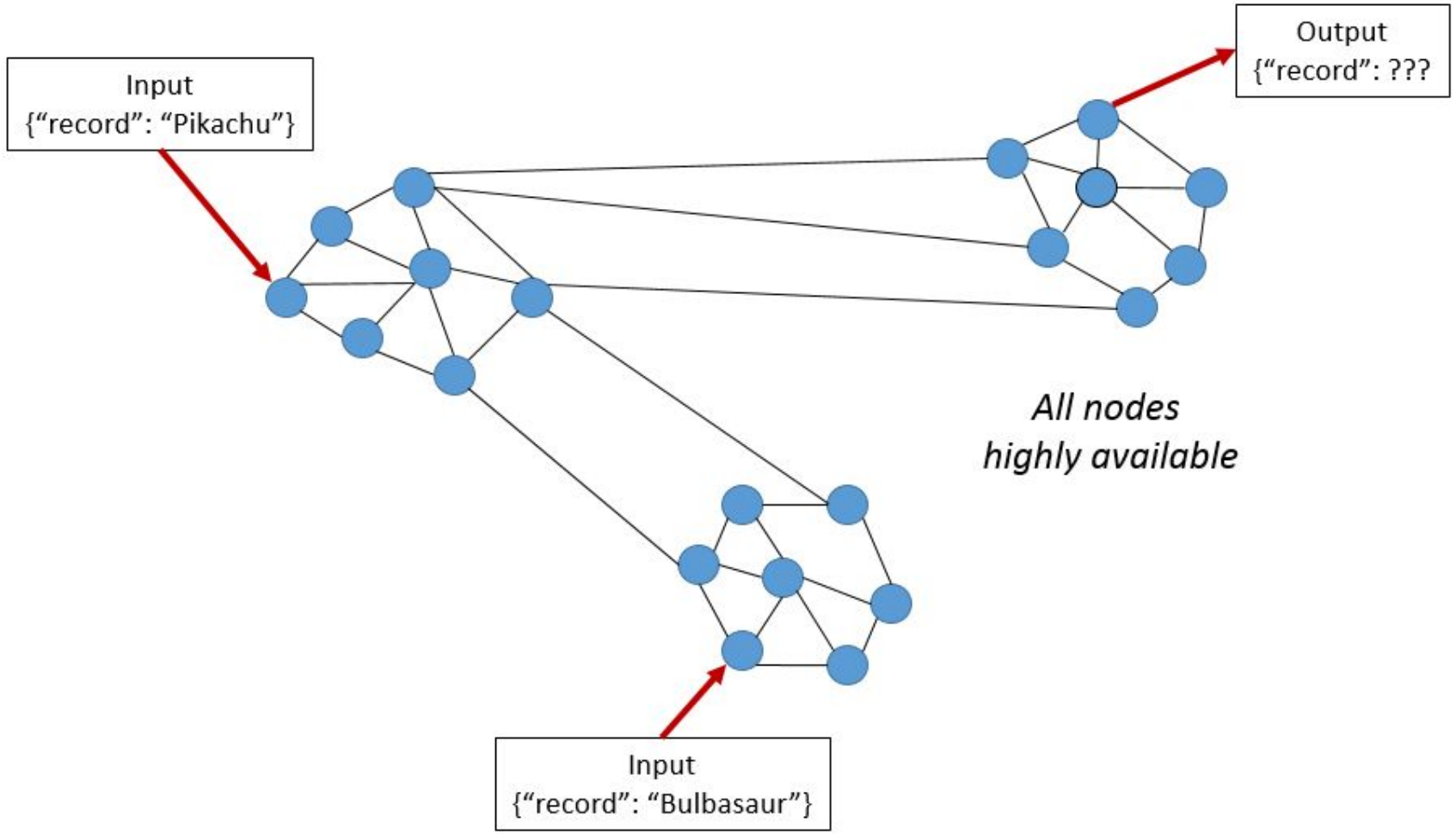

Teorema CAP
Oracle
My SQL
Consistencia
No SQL
Mongo DB
HBase
Redis
CA
CP
X
Tolerancia a
particiones
Disponibilidad
AP
No SQL
Coach DB
Cassandra
DynamoDB
FutureBites
FutureSpace

分布式理论

Consistency



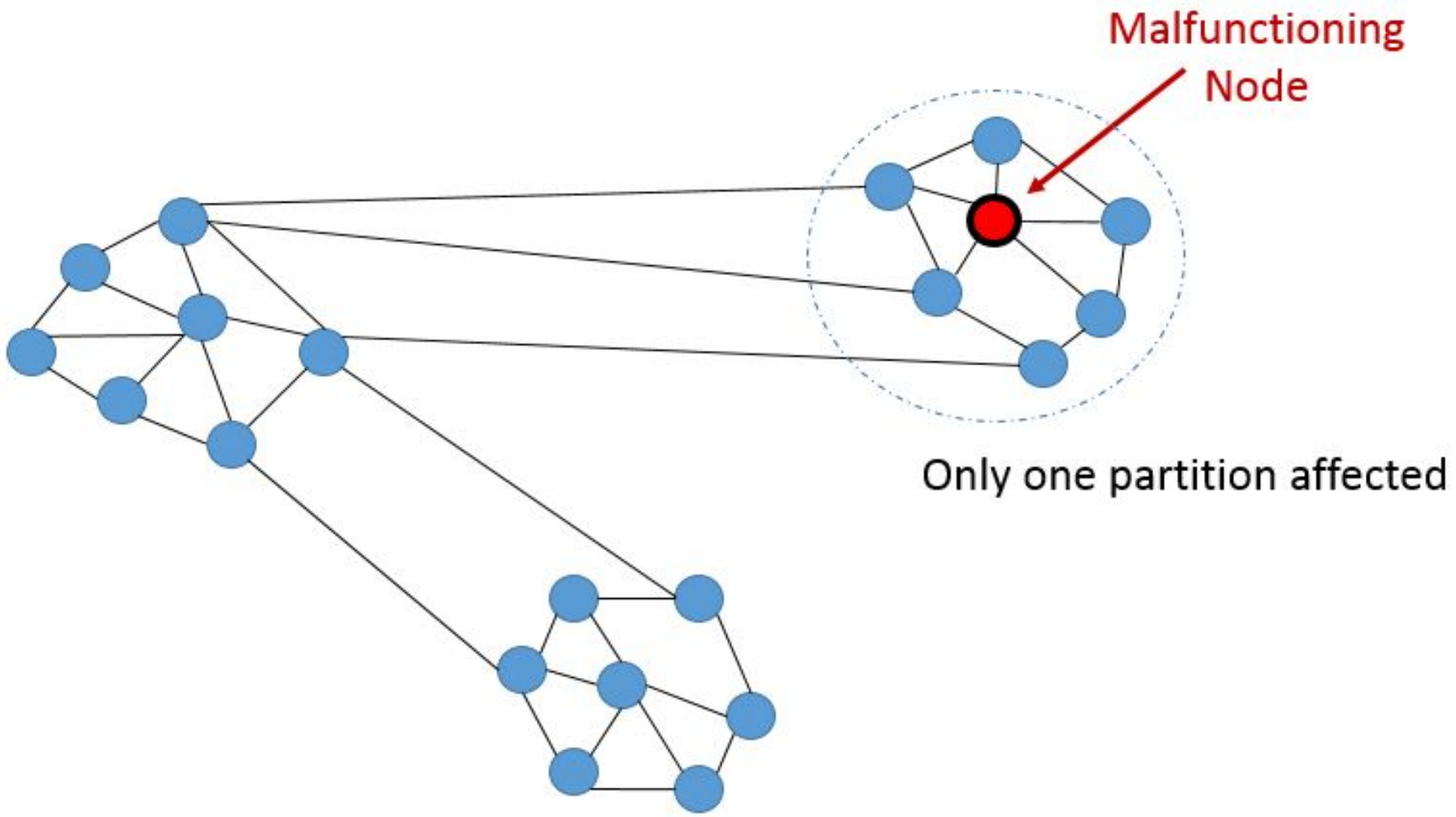
分布式理论



Availability

分布式理论

Partition-Tolerance

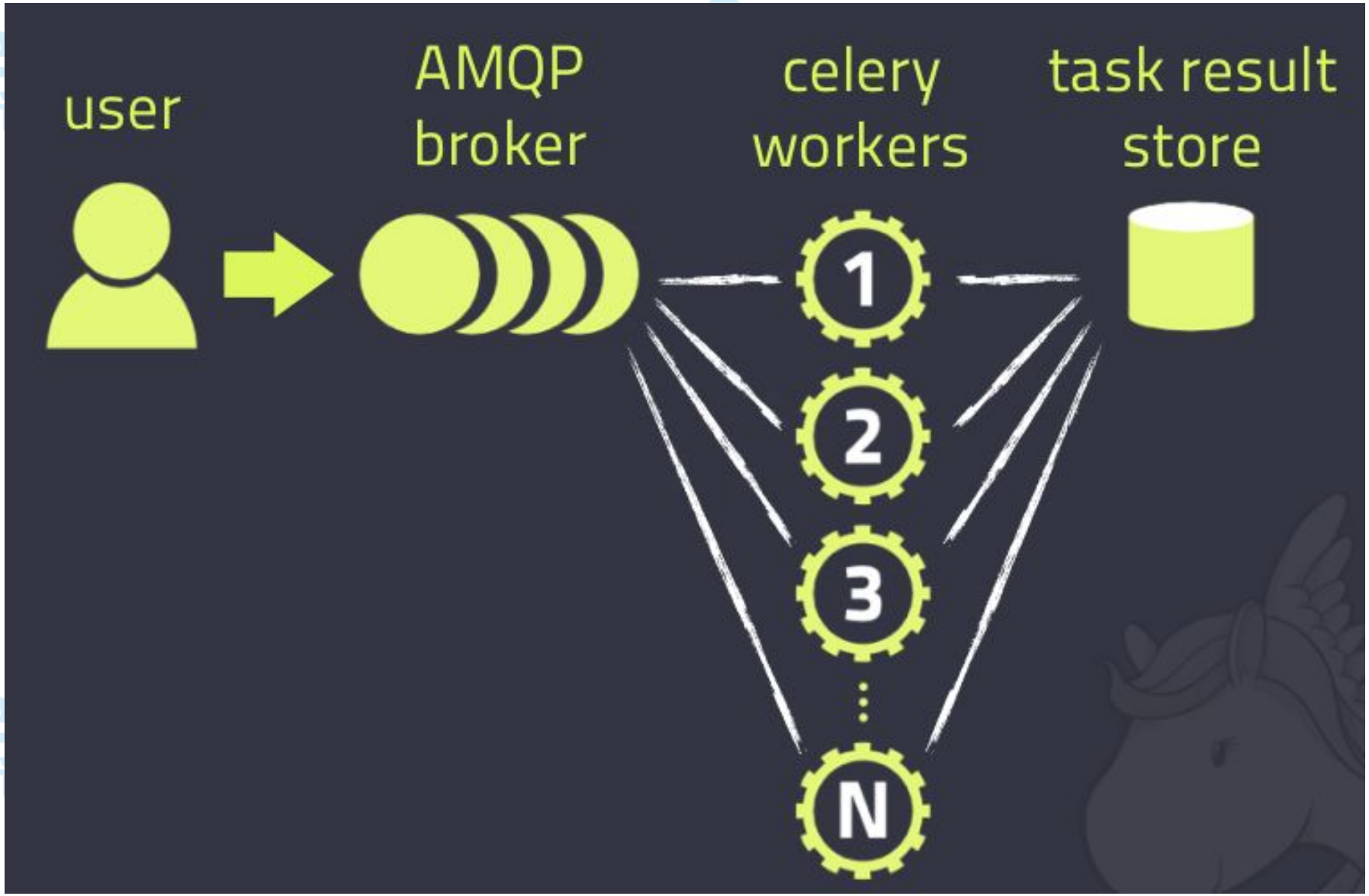


分布式理论

可用性和一致性的矛盾

分布式框架介绍

Celery

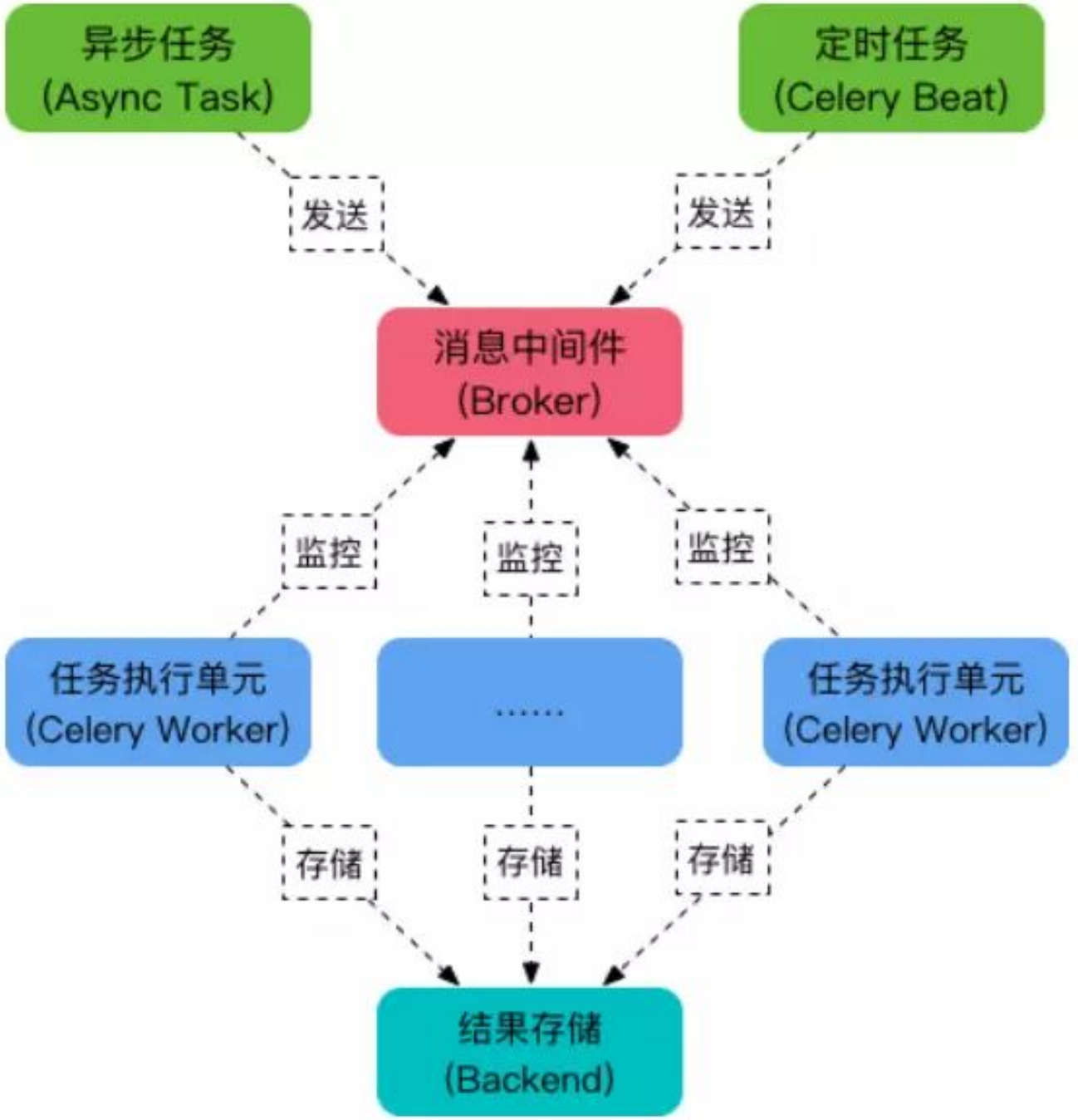


分布式框架介绍

消息中间件(message broker): 本身不提供消息服务，可以和第三方消息中间件集成，常用的有 redis mongodb rabbitMQ

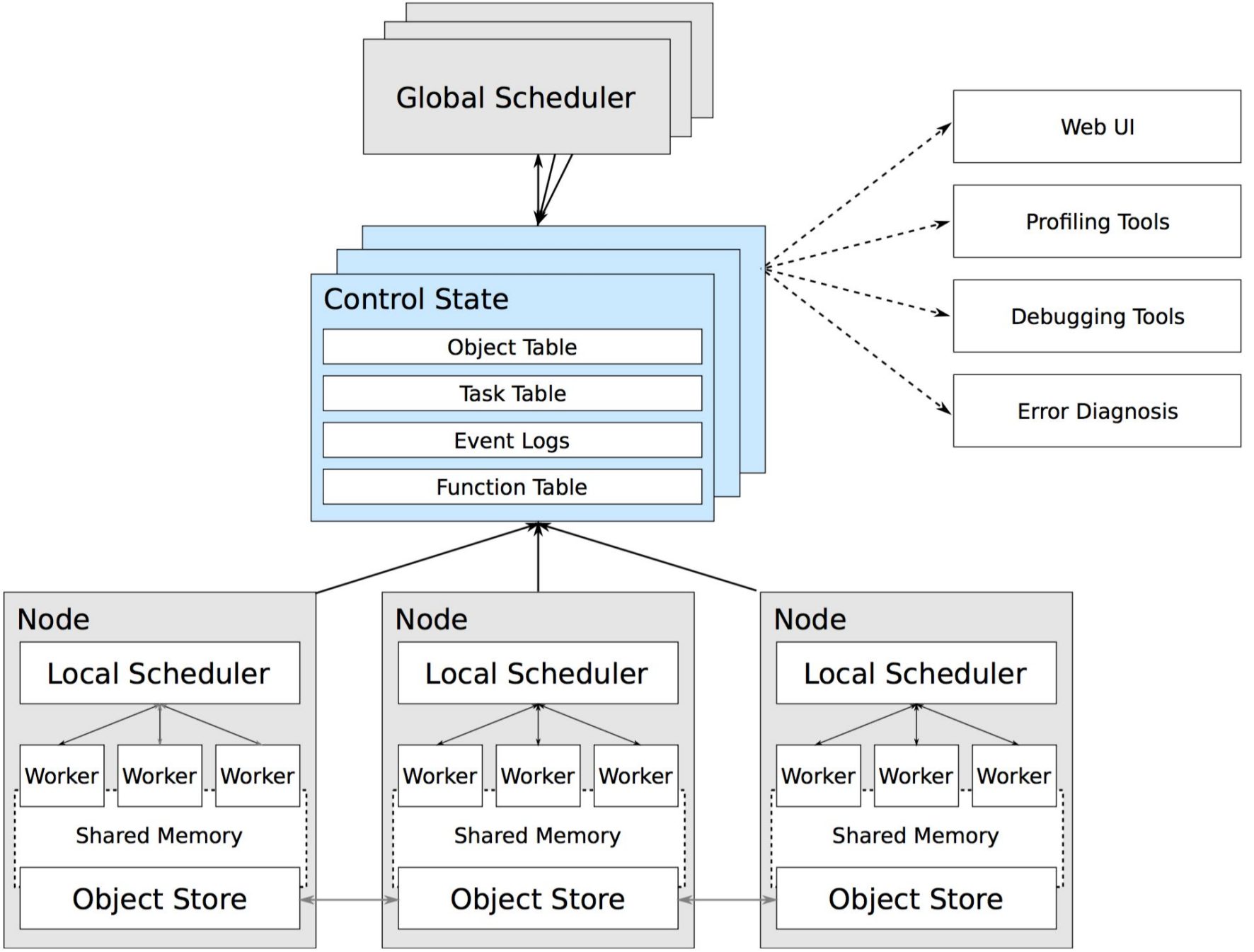
任务执行单元(worker): 是Celery提供的任务执行单元，worker并发的运行在分布式的系统节点中

任务执行结果存储(task result store): 用来存储Worker执行的任务的结果，Celery支持以不同方式存储任务的结果，包括
Redis，MongoDB，AMQP等



分布式框架介绍

Ray


Global Scheduler
Web UI
Profiling Tools
Debugging Tools
Error Diagnosis
Control State
Object Table
Task Table
Event Logs
Function Table
Node
Local Scheduler
Worker
Worker
Worker
Shared Memory
Object Store
Node
Local Scheduler
Worker
Worker
Worker
Shared Memory
Object Store
Node
Local Scheduler
Worker
Worker
Worker
Shared Memory
Object Store

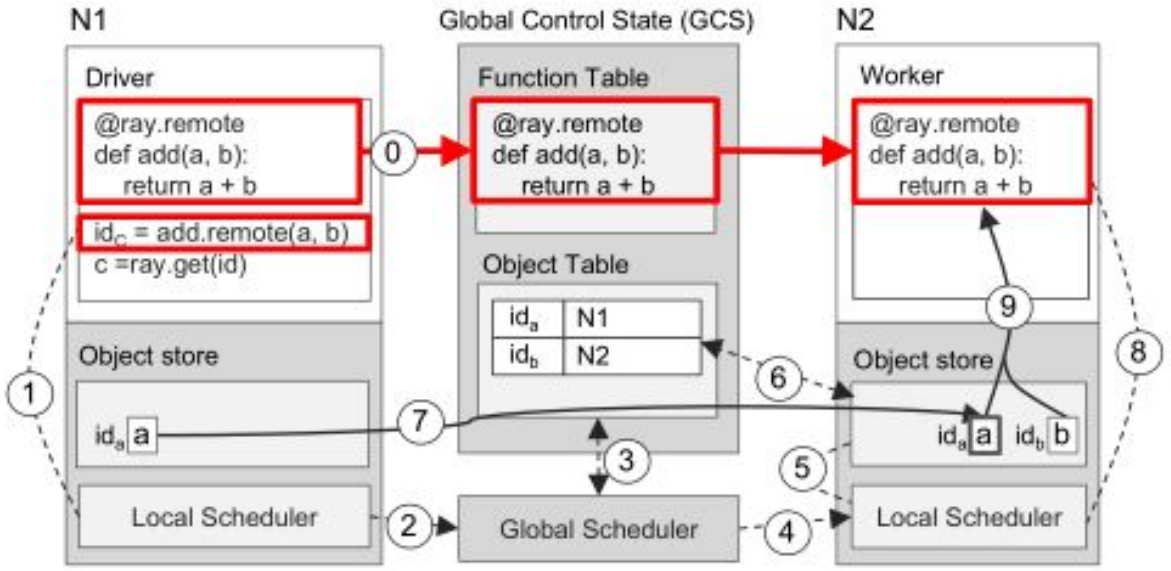
分布式框架介绍

Ray

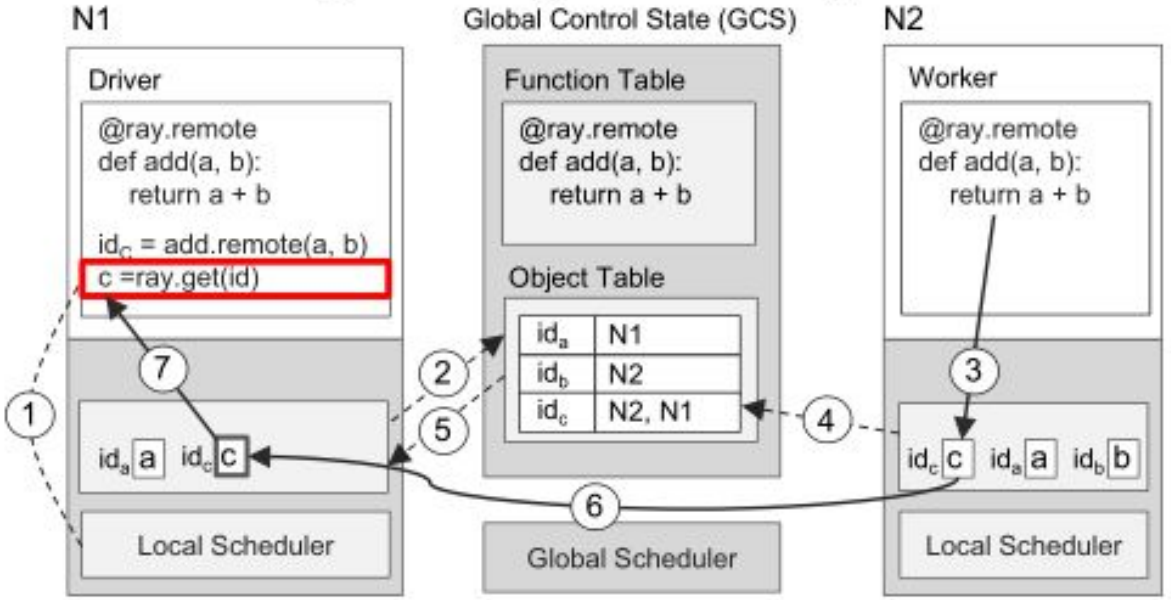
- **GlobalScheduler**：Master上启动了一个全局调度器，用于接收本地调度器提交的任务，并将任务分发给合适的本地任务调度器执行。
- **RedisServer**：Master上启动了一到多个RedisServer用于保存分布式任务的状态信息（ControlState），包括对象机器的映射、任务描述、任务debug信息等。
- **LocalScheduler**：每个Slave上启动了一个本地调度器，用于提交任务到全局调度器，以及分配任务给当前机器的Worker进程。
- **Worker**：每个Slave上可以启动多个Worker进程执行分布式任务，并将计算结果存储到ObjectStore。
- **ObjectStore**：每个Slave上启动了一个ObjectStore存储只读数据对象，Worker可以通过共享内存的方式访问这些对象数据，这样可以有效地减少内存拷贝和对象序列化成本。ObjectStore底层由Apache Arrow实现。
- **Plasma**：每个Slave上的ObjectStore都由一个名为Plasma的对象管理器进行管理，它可以在Worker访问本地ObjectStore上不存在的远程数据对象时，主动拉取其它Slave上的对象数据到当前机器。

分布式框架介绍

```
@ray.remote
def add(a,b):
    return a + b
a = 1
b = 2
id(c) = add.remote(a,b)
c = ray.get(id(c))
```



(a) Executing a task remotely

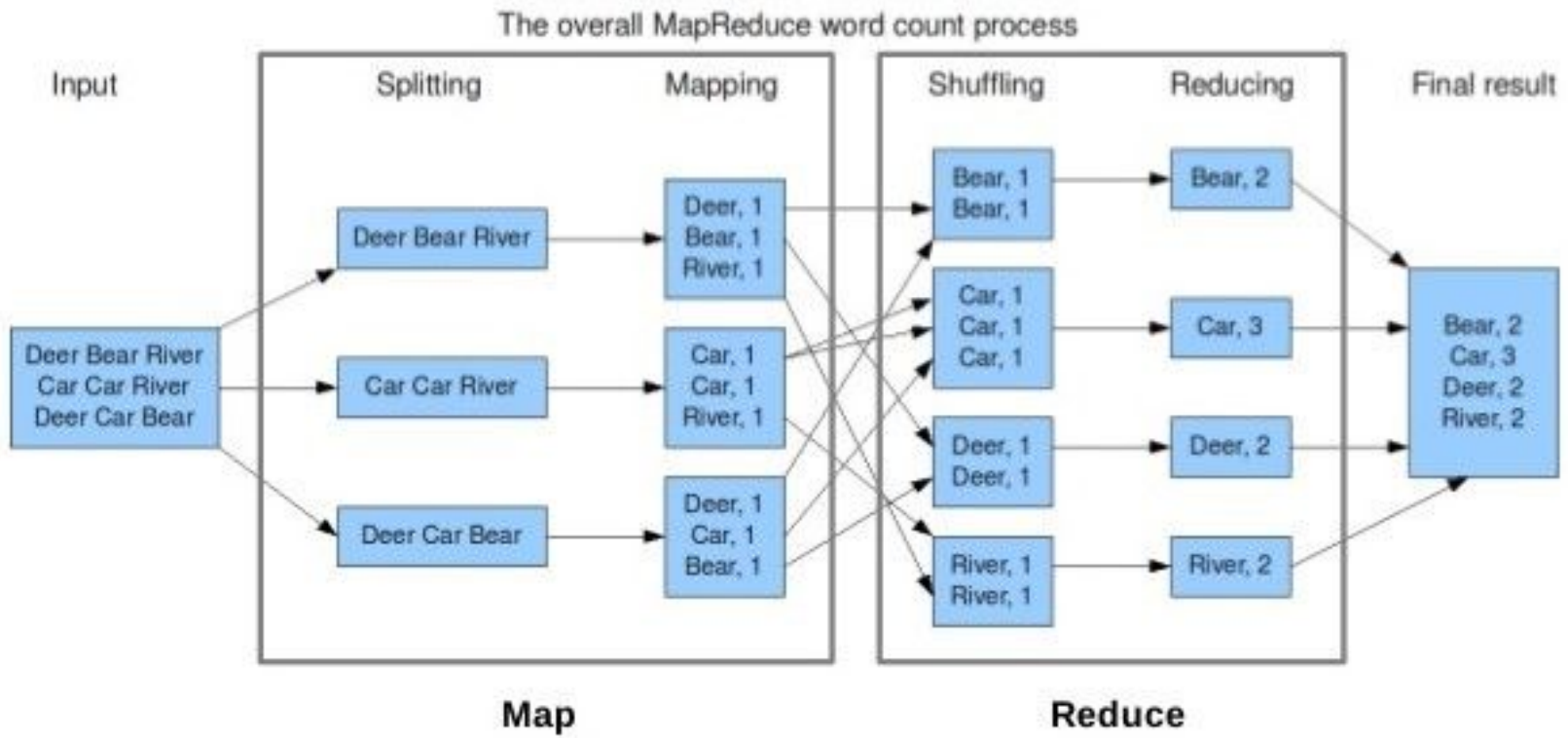


(b) Returning the result of a remote task

分布式实战

基于Ray实现MapReduce，用以实现分布式词频统计(WordCount)

分布式实战


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2
Map
Reduce

Assignment

- 分布式爬虫（不限制爬取内容）
- 基于Celery/Ray
- 多机部署