# Tachyon (20:30准时开课)

--讲师：Yasaka

12月26日周末班 2月26日全日制班 3月19日 在线班 欢迎您的到来！

需要代码、PPT、视频等资料请加以下几位老师QQ：

贾老师：1786418286

何老师：1926106490

詹老师：2805048645

讨论技术可以加入以下QQ群： 172599077 ， 156927834

热烈庆祝1221班爆满开班！！！

16.1.1日之后学费上调，考虑春节后培训的学生可以提前报名，预订座位，不管以后何时过来学习，费用都是以报名进的费用为主！

# 内容大概

- Tachyon在大数据生态的位置
- Tachyon的发展
- Tachyon是必然
- Tachyon解决的问题
- Tachyon的架构和原理
- Tachyon的使用
- Tachyon的公司案例
- Tachyon的一些新特性

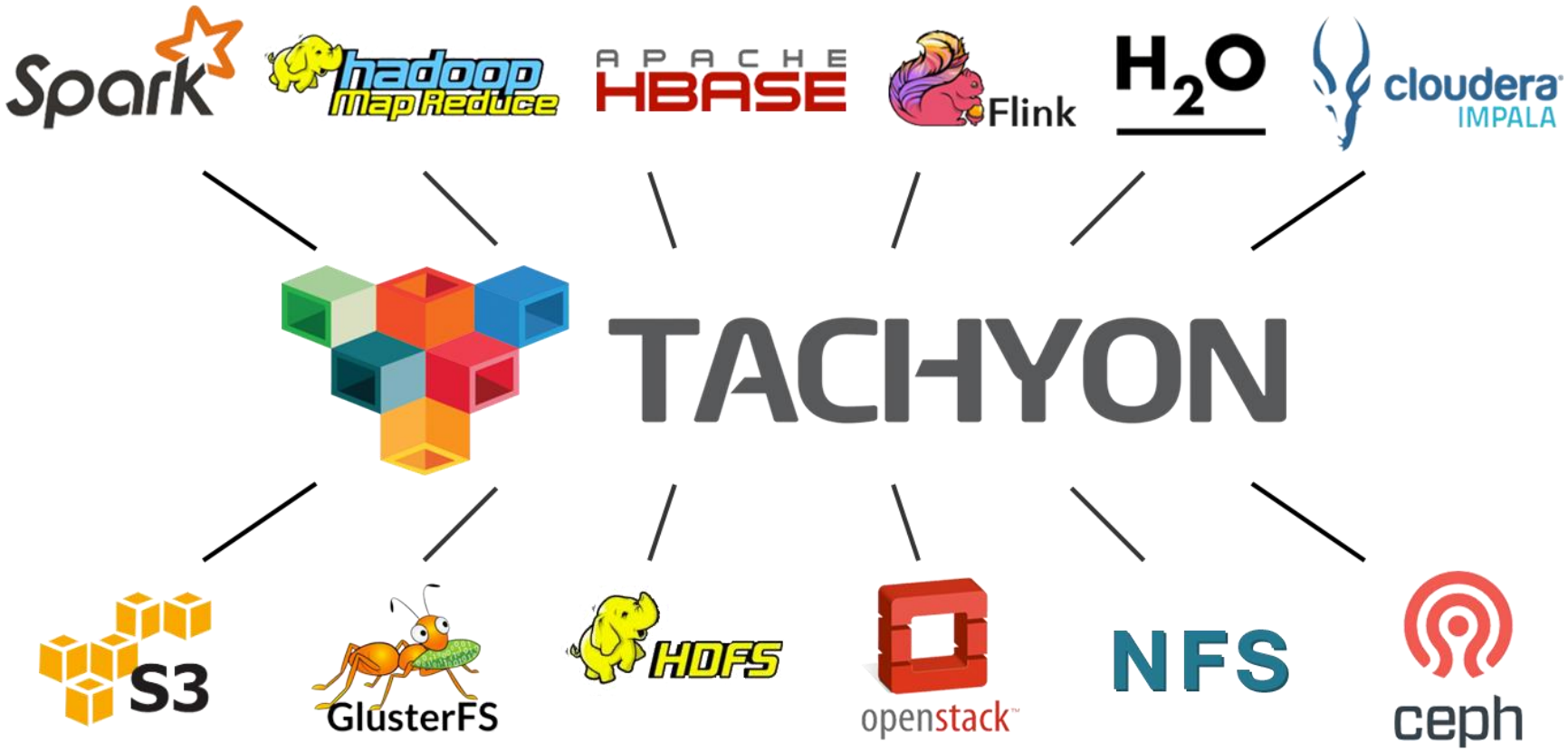
# 一种假想的超光速粒子

TACHYON
NEXUS
www.tachyonnexus.com

Tachyon项目的创始人及核心开发人员

A轮融资： Andreessen Horowitz，750万美元

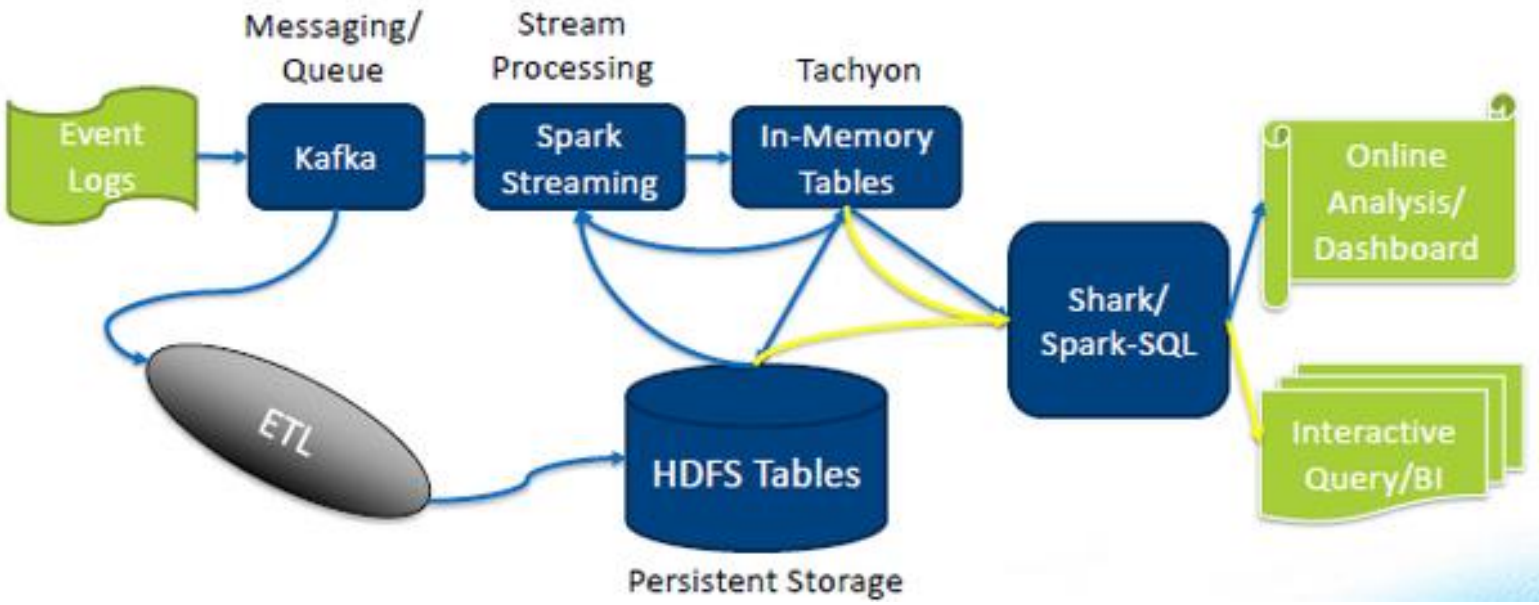
致力于Tachyon开源项目


Spark
hadoop MapReduce
APACHE HBASE
Flink
H2O
cloudera IMPALA
TACHYON
S3
GlusterFS
HDFS
openstack
NFS
ceph

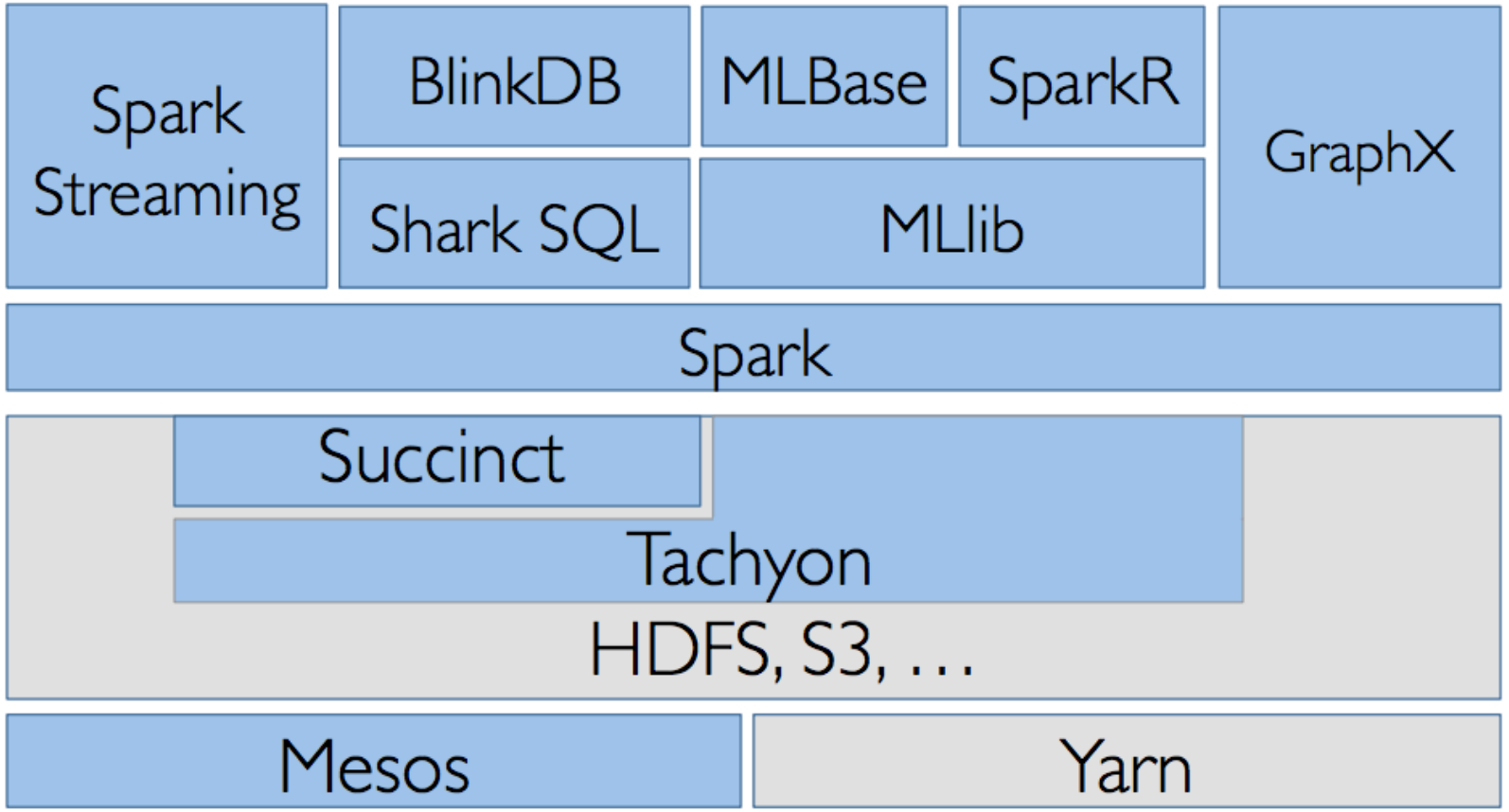
# 介于磁盘存储和计算框架的一个中间件

- 用于做分布式的内存文件读写等操作
  - 分布式的机器可以访问Tachyon
  - 从Tachyon中可以读取数据
  - 分布式集群共享数据
    - Executor
    - 磁盘

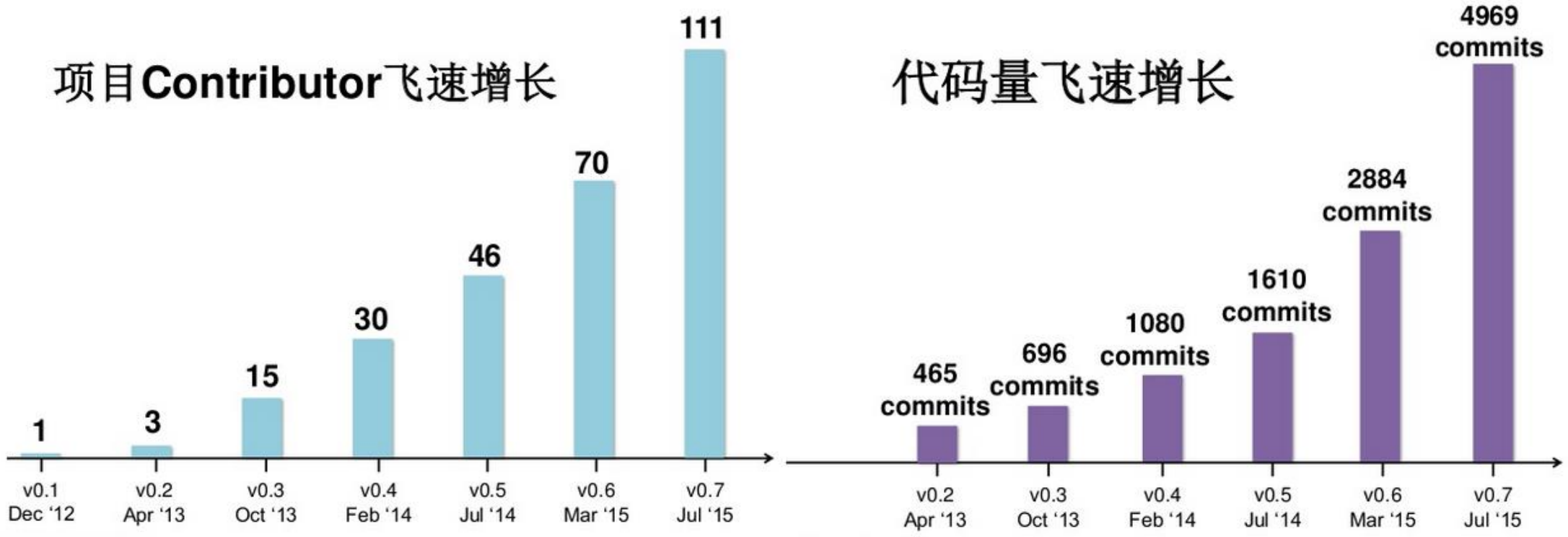
## Tachyon应用实例-数据共享



# Berkeley Data Analytics Stack



# UC Berkeley Amplab 李浩源


项目Contributor飞速增长
1
3
15
30
46
70
111
v0.1 Dec '12
v0.2 Apr '13
v0.3 Oct '13
v0.4 Feb '14
v0.5 Jul '14
v0.6 Mar '15
v0.7 Jul '15
代码量飞速增长
465 commits
696 commits
1080 commits
1610 commits
2884 commits
4969 commits
v0.2 Apr '13
v0.3 Oct '13
v0.4 Feb '14
v0.5 Jul '14
v0.6 Mar '15
v0.7 Jul '15

## Contributors Growth

> 170 Contributors (V0.8)
(3x increment over the last AMPCamp)

> 50 Organizations

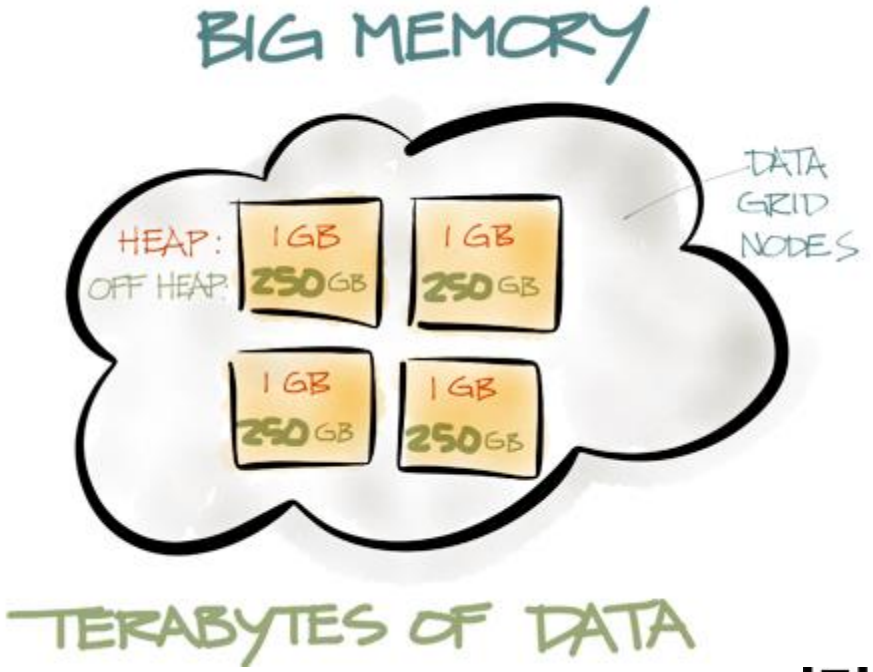
## Thanks to Contributors and Users!

http://tachyon-project.org/community/

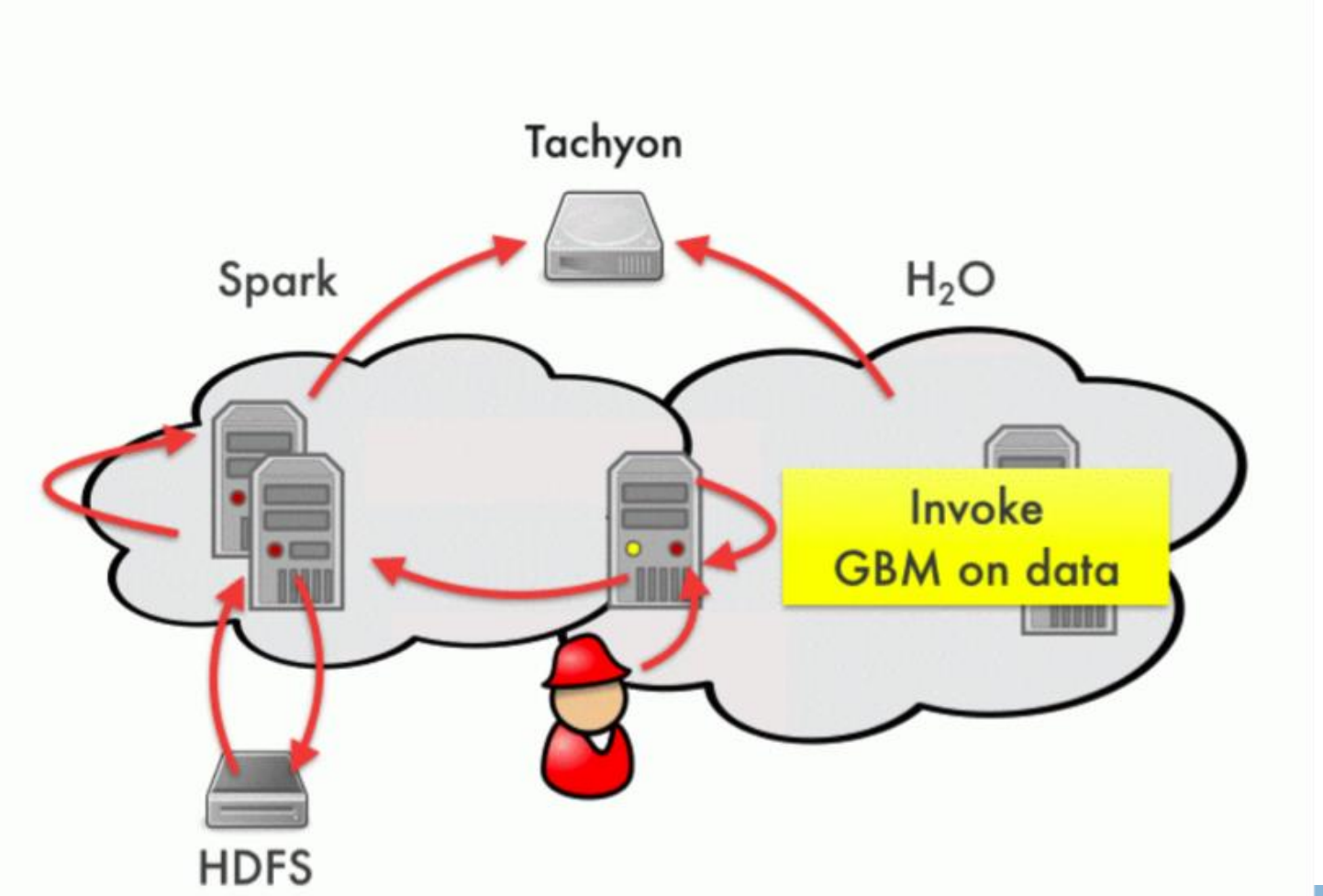
YAHOO! intel IBM Adobe
HUAWEI Baidu 百度 NOKIA databricks
redhat Berkeley CRAY THE SUPERCOMPUTER COMPANY
Atigeo TACHYON NEXUS Mellanox TECHNOLOGIES Pivotal Palantir technologies ClearStory DATA

- Default Tachyon !!!

Tachyon is the
Default Off-Heap Storage
Solution for Spark



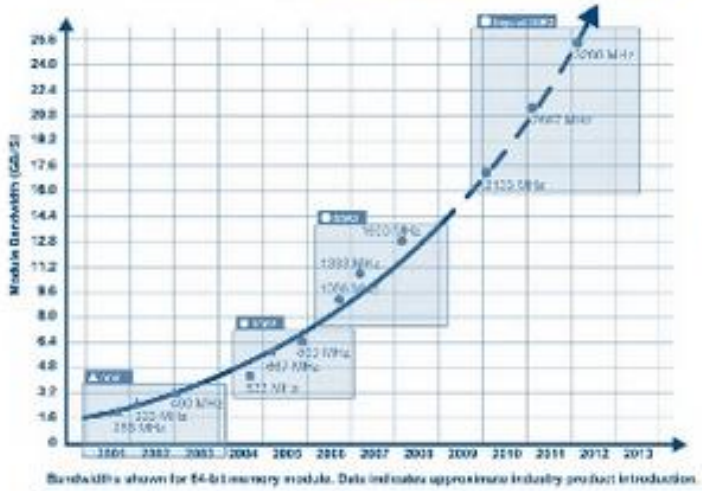
# 不同框架间的协调合作上下文环境



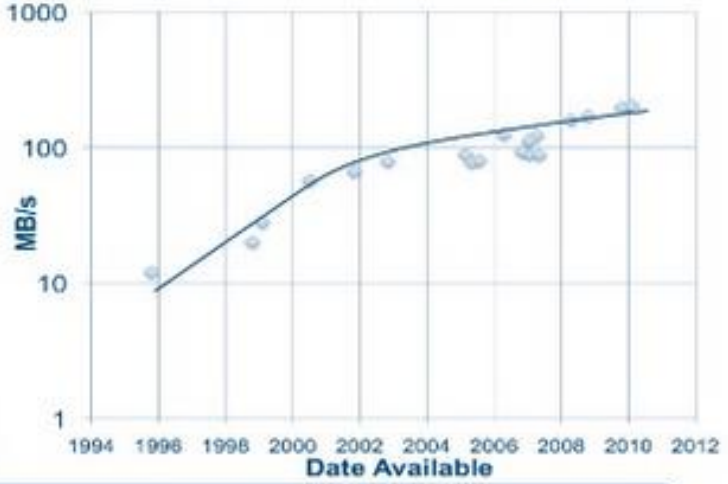
- 数据移动
- Shuffle

## Performance Trend: Memory is Fast

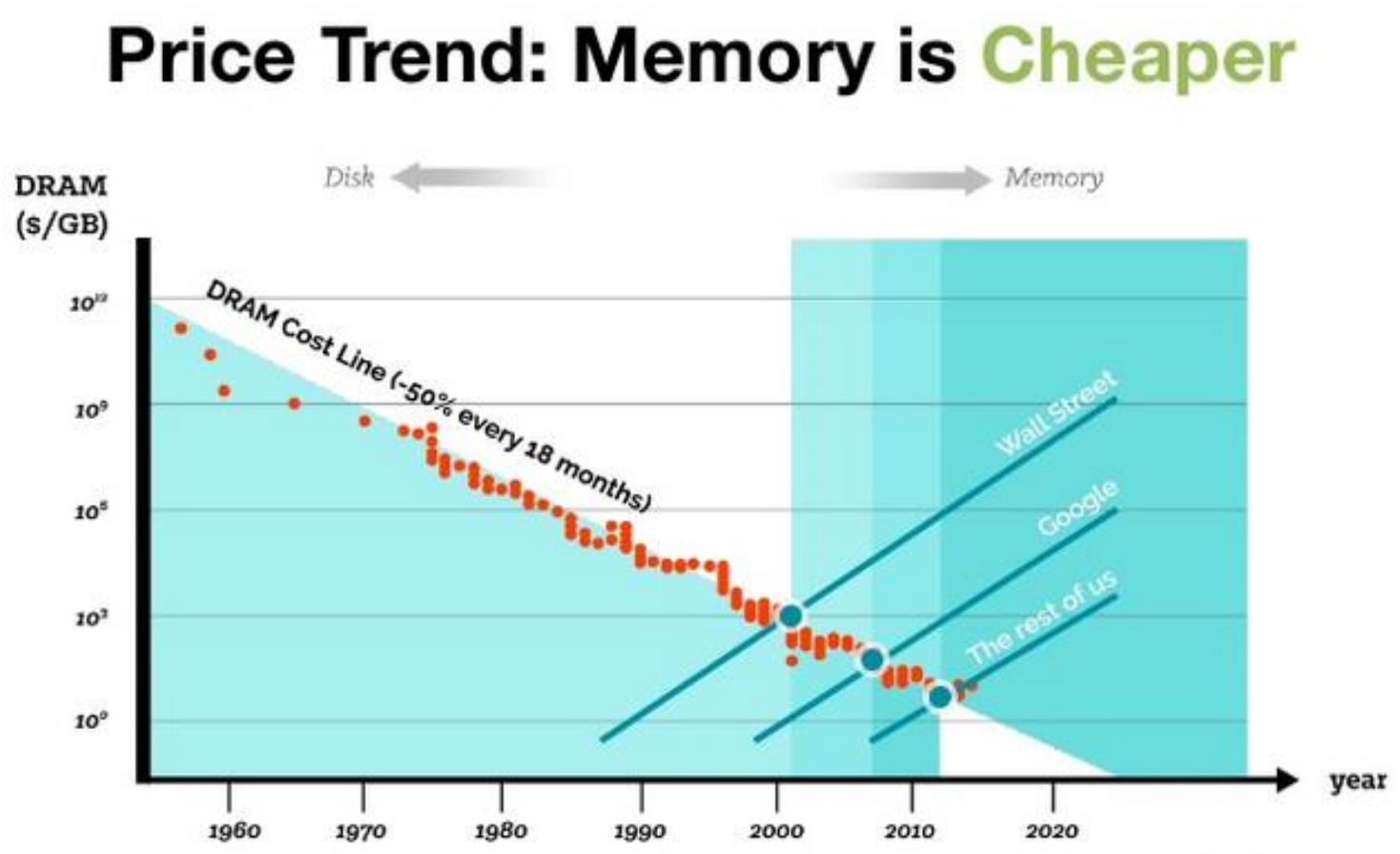
- RAM throughput increasing exponentially



- Disk throughput increasing slowly

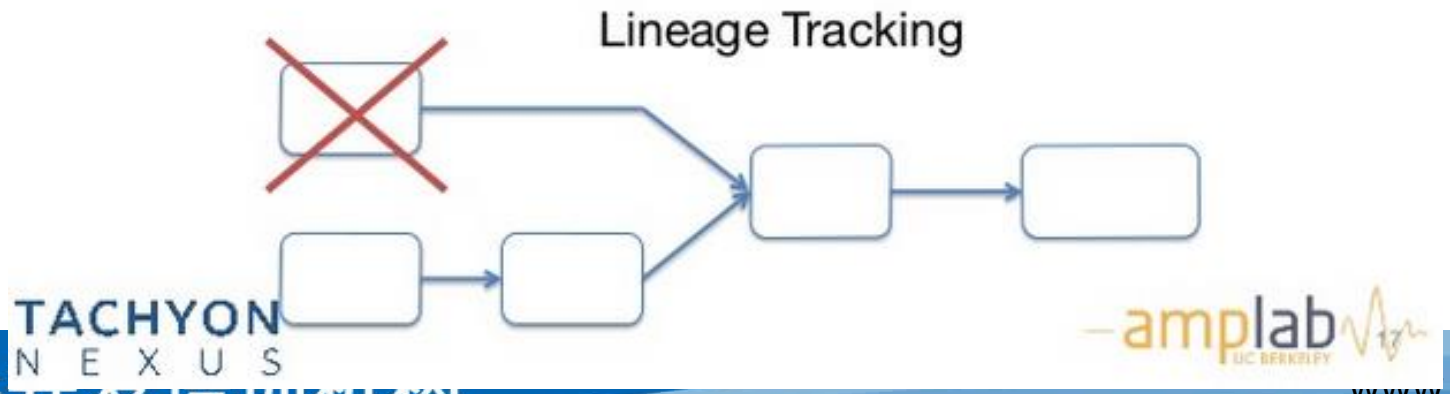



Memory-locality key to interactive response times

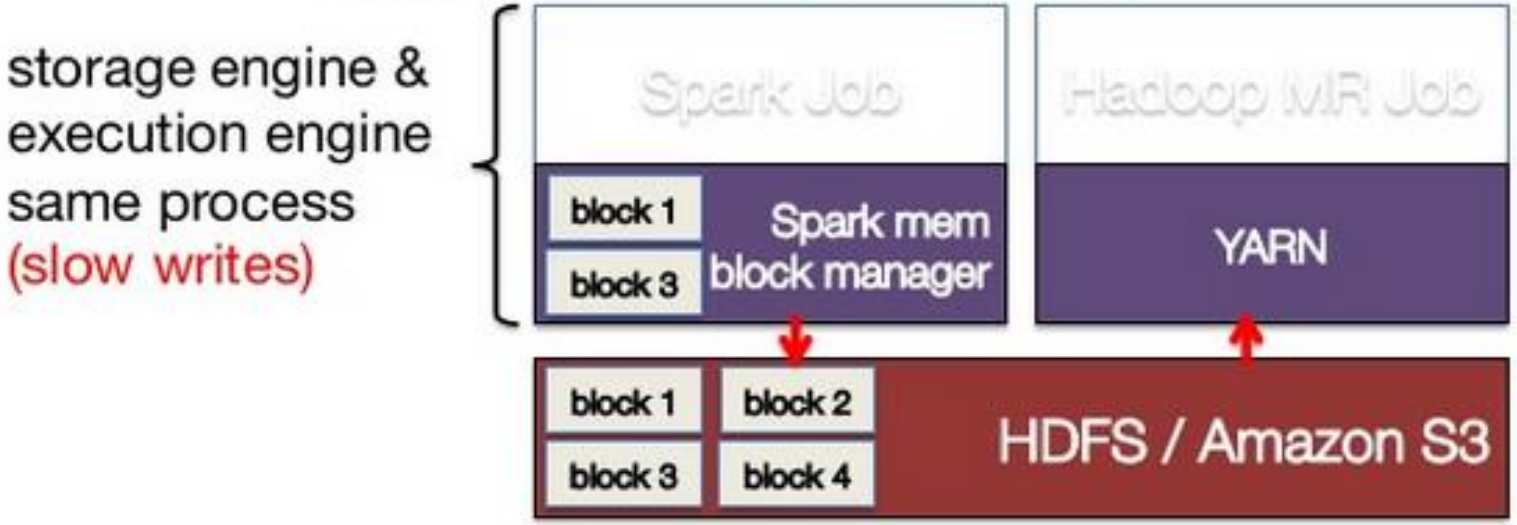

Price Trend: Memory is Cheaper
Disk
Memory
DRAM
(s/GB)
DRAM Cost Line (-50% every 18 months)
Wall Street
Google
The rest of us
year
1960
1970
1980
1990
2000
2010
2020

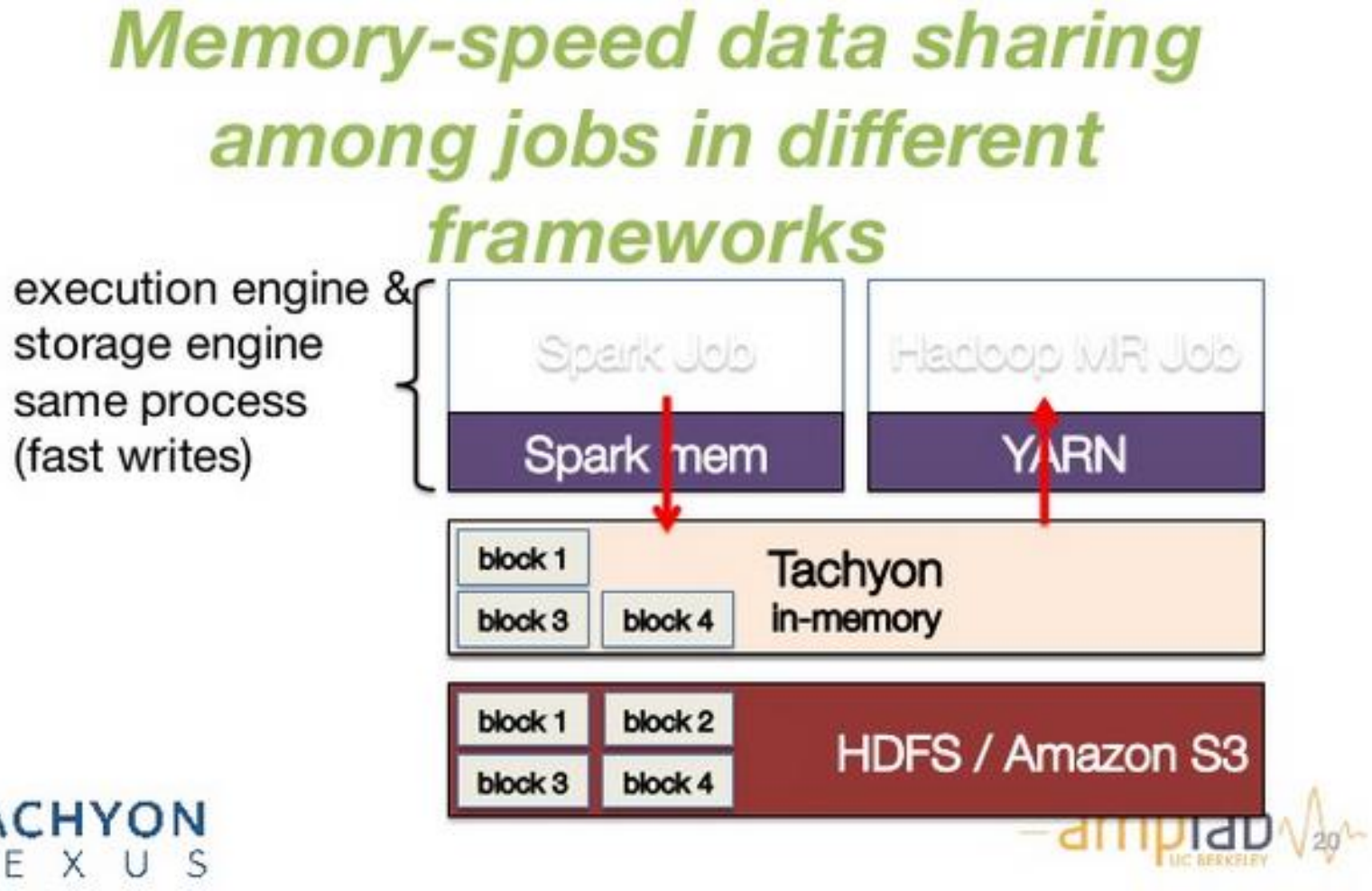
## A Use Case Example with Spark

- Fast, in-memory data processing framework
  - Keep one in-memory copy inside JVM
  - Track lineage of operations used to derive data
  - Upon failure, use lineage to recompute data

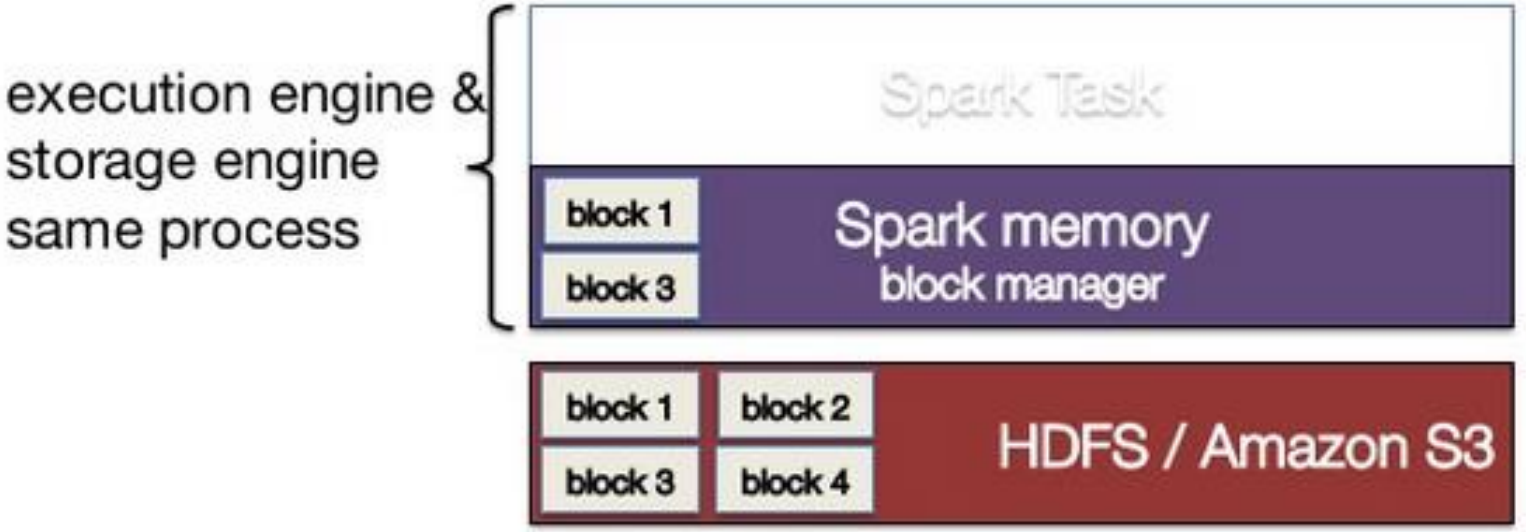


## Data Sharing is the bottleneck in analytics pipeline: Slow writes to disk

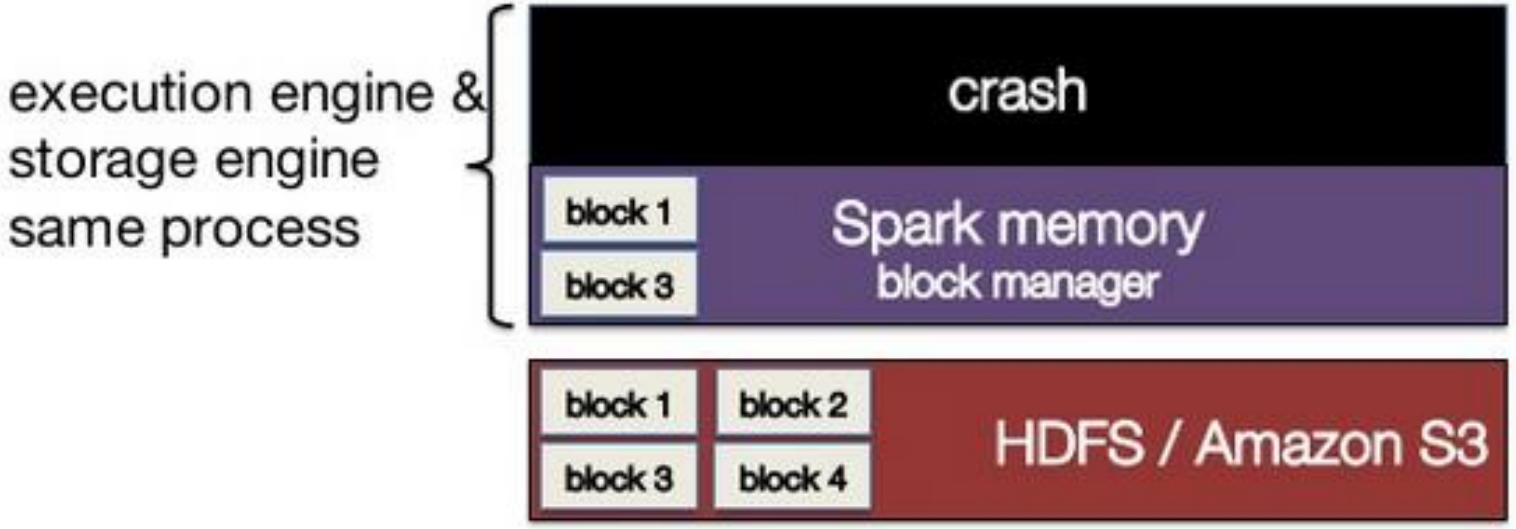




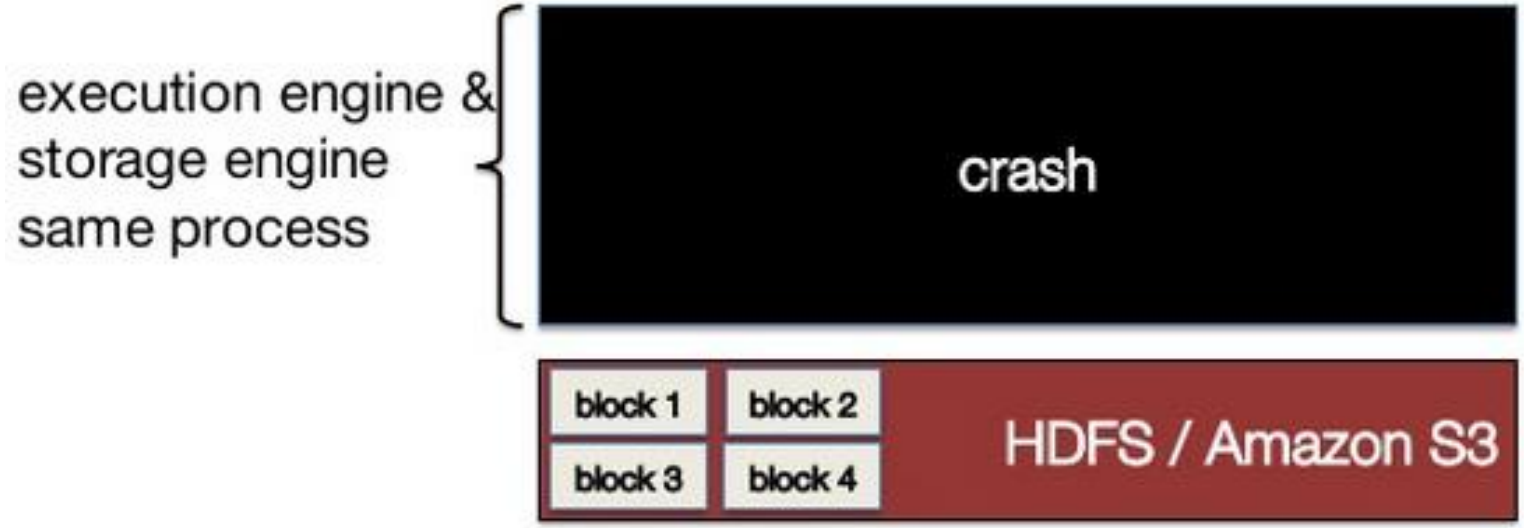
# Cache loss when process crashes



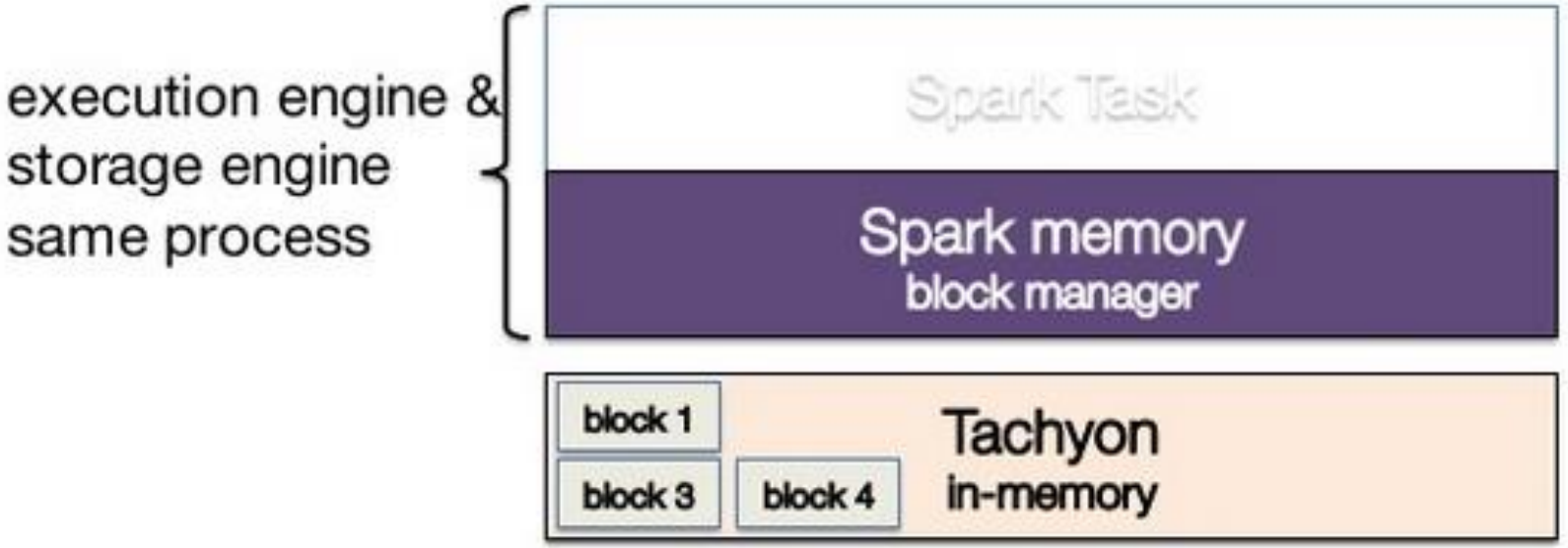
## Cache loss when process crashes



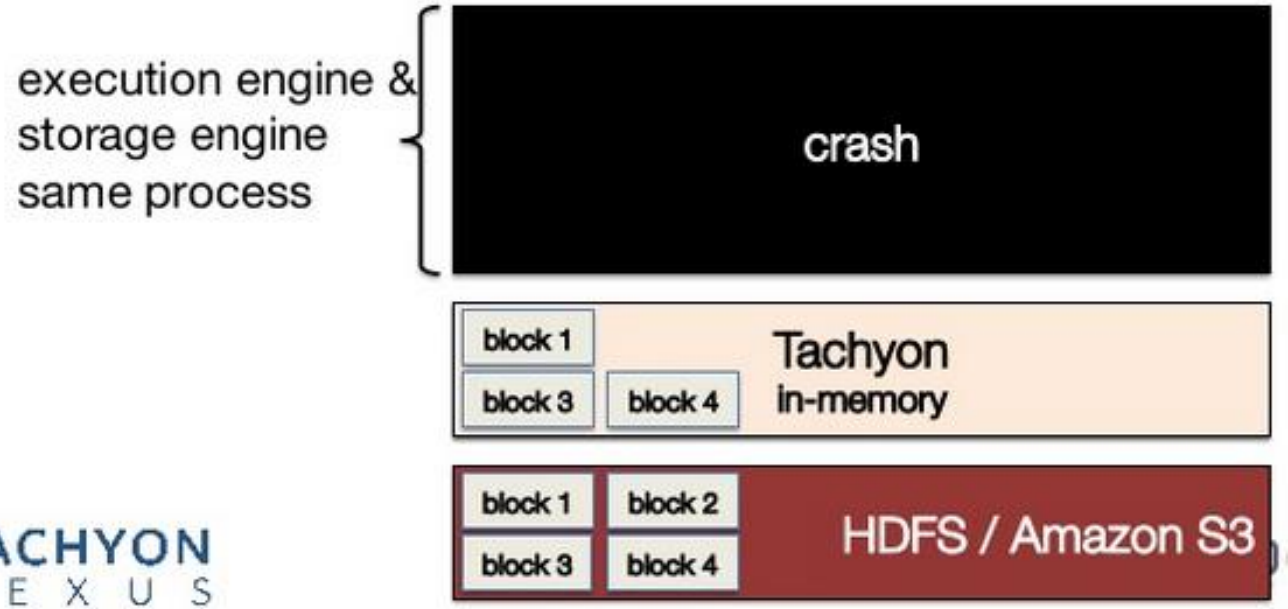
## Cache loss when process crashes



*Keep in-memory data safe, even when a job crashes.*

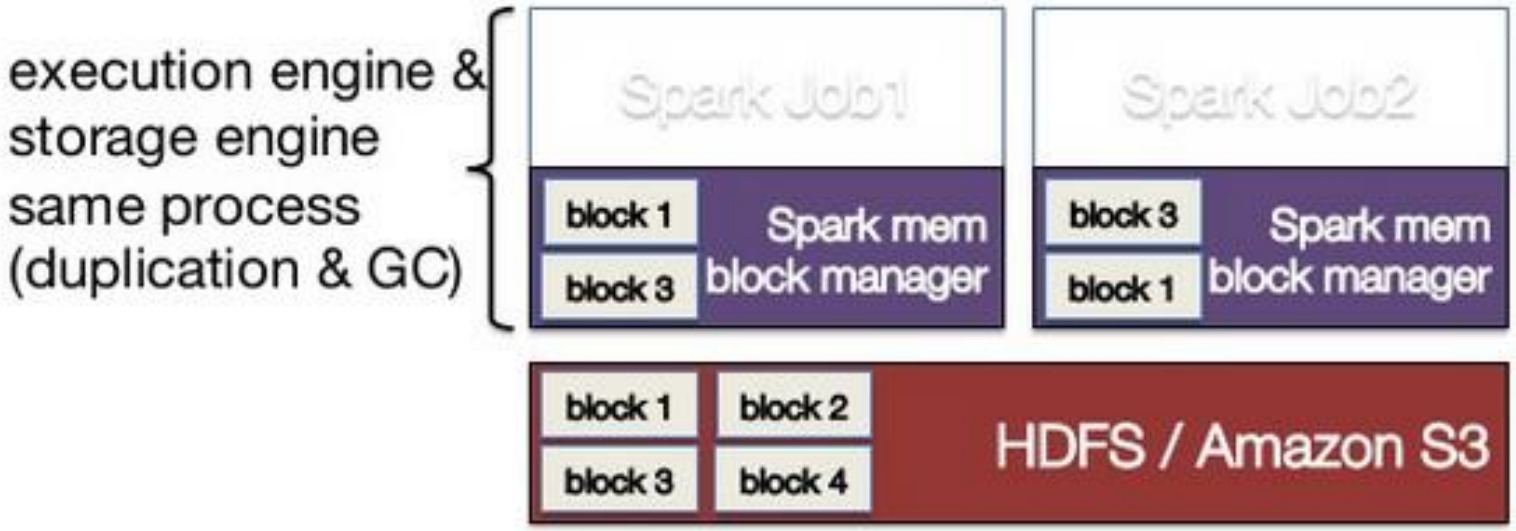


*Keep in-memory data safe, even when a job crashes.*

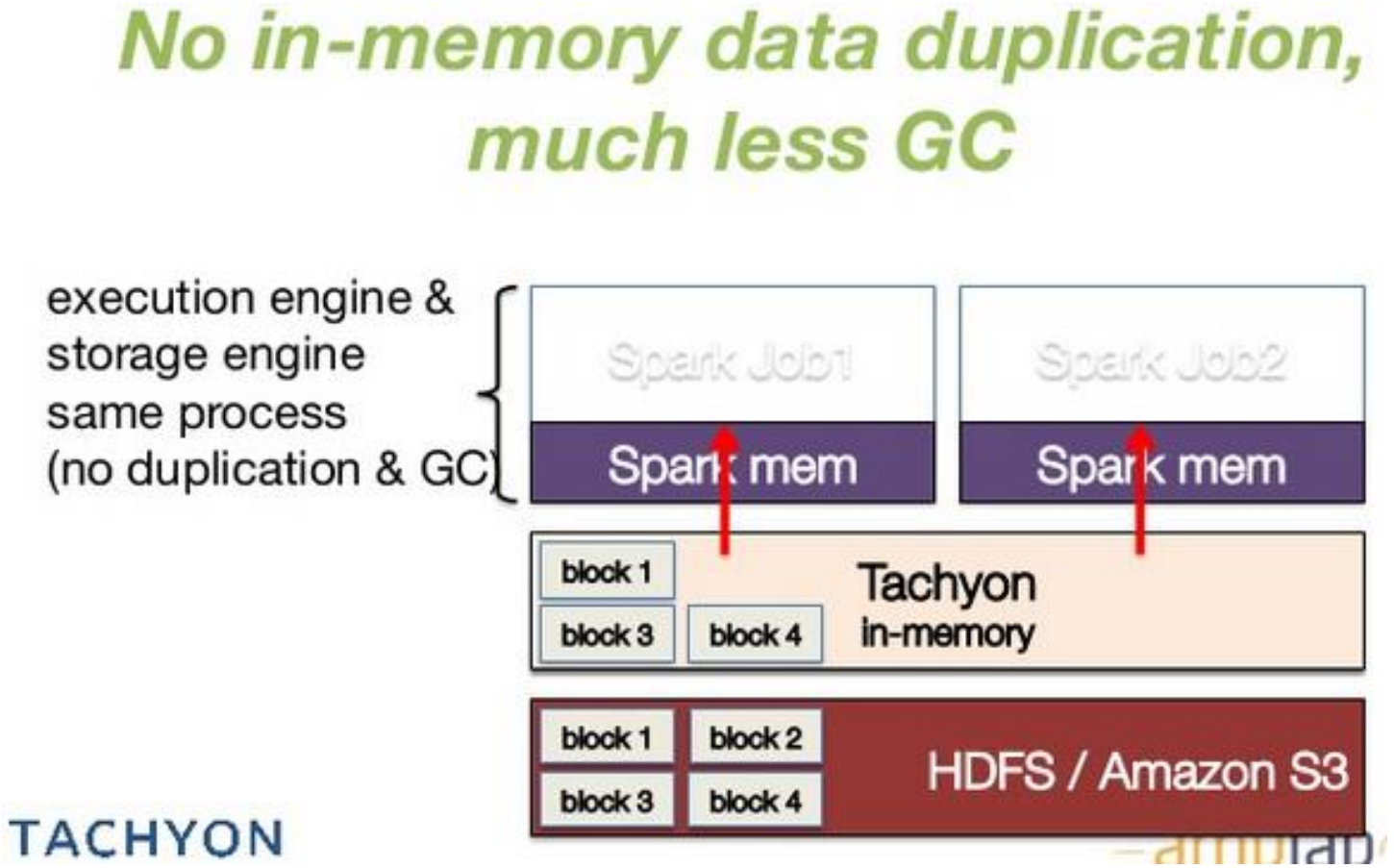



TACHYON
NEXUS

# In-memory Data Duplication & Java Garbage Collection



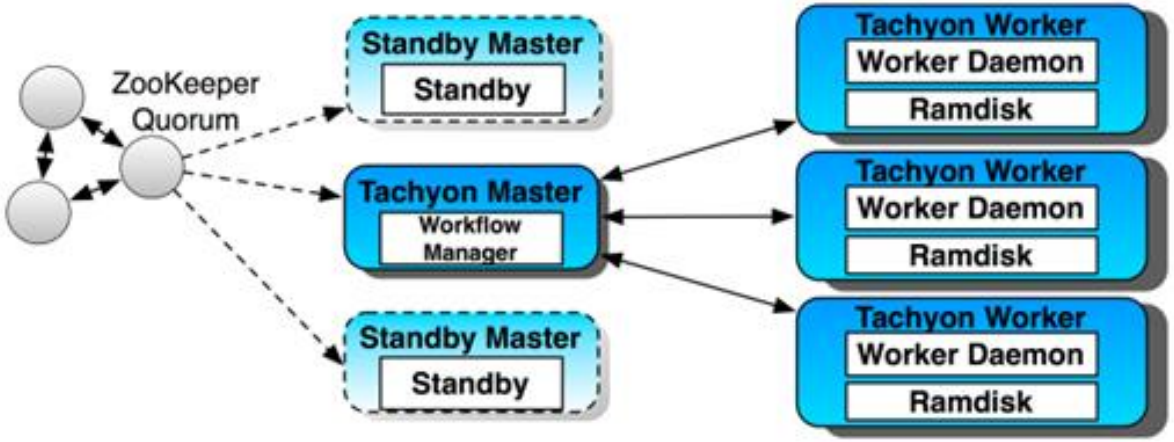
# 不同任务使用一个数据重复加载

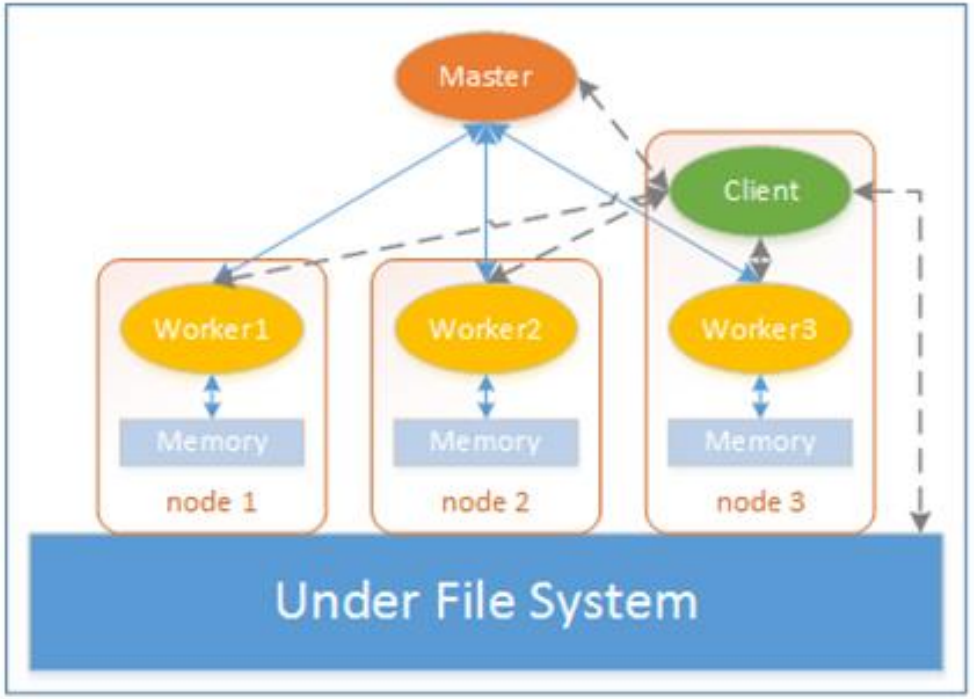


- A memory-centric storage architecture

- Push lineage down to storage layer

- 数据在work的内存中

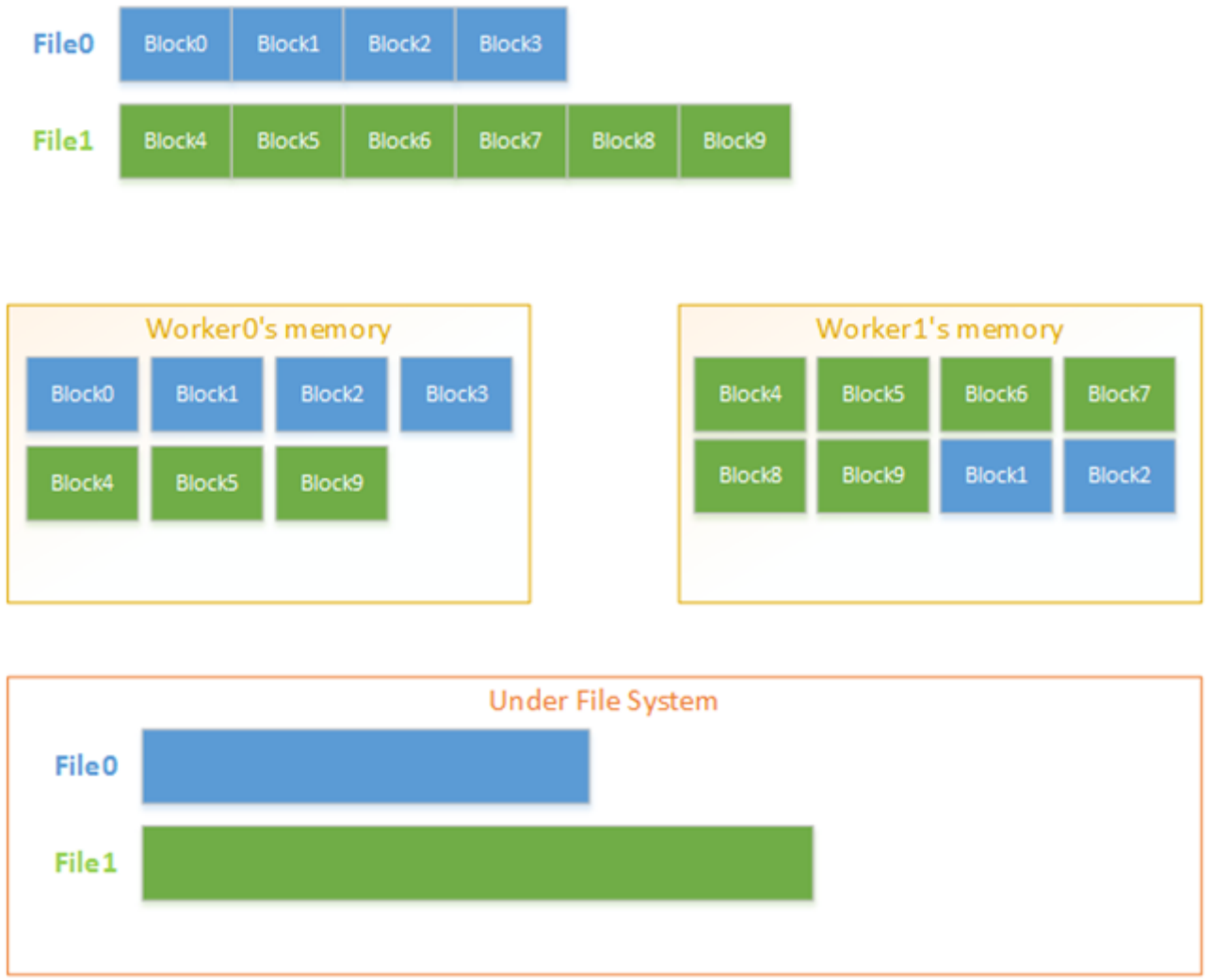


# 心跳机制

- 在Tachyon中，心跳（HeartBeat）用于两个方面：Master, Worker, Client之间的定期通信；Master, Worker自身的定期状态自检。具体地：
- Client向Master发送心跳信号：表示Client仍处于连接中，Client释放连接后重新连接会获得新的UserId
- Client向Worker发送心跳信号：表示Client仍处于连接中，释放连接后Worker会回收该Client的用户空间
- Worker自检，同时向Master发送心跳信号：Worker将自己的存储空间信息更新给Master（容量，移除的块信息），同时清理超时的用户，回收用户空间
- Master自检：检查所有Worker的状态，若有Worker失效，会统计丢失的文件并尝试重启该Worker

# 文件组织



# 读写类型



读类型： CACHE – 读取数据并缓存在本地内存

NO_CACHE – 读取数据但不缓存在本地内存

写类型： MUST_CACHE – 只写本地内存，空间不足时报ERROR

TRY_CACHE – 只写本地内存，空间不足时报WARNING

THROUGH – 只写UFS

CACHE_THROUGH – 同时写本地内存和UFS（TRY_CACHE + THROUGH）

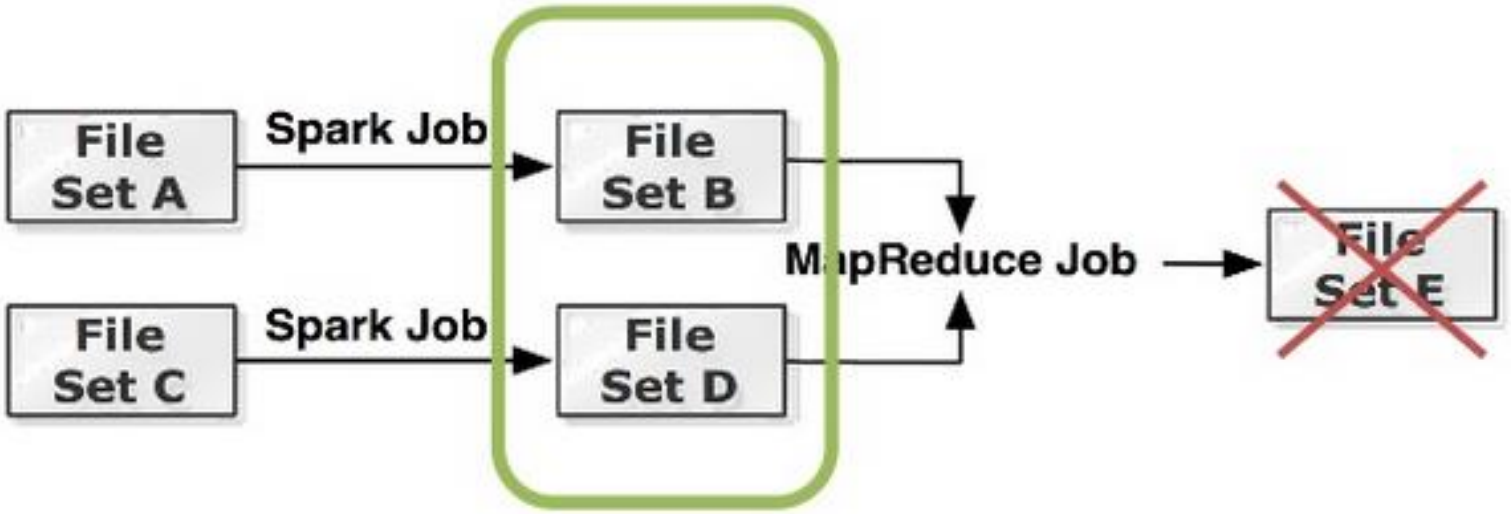
ASYNC_THROUGH– 先写本地内存，异步备份到UFS

# 容错机制



- 作为分布式文件系统，Tachyon具有良好的容错机制，Master和Worker都有自己的容错方式。
- 从之前的系统架构图中也可看出，Master支持使用ZooKeeper进行容错。同 时，Master中保存的元数据使用Journal进行容错，具体包括Editlog——记录所有对元数据的操作，以及Image——持久化元数据信息。 此外，Master还对各个Worker的状态进行监控，发现Worker失效时会自动重启对应的Worker。
- 对于具体的文件数据，使用血统关系（Lineage）进行容错。文件元数据中记录了文件之间的依赖关系，当文件丢失时，能够根据依赖关系进行重计算来恢复文件数据。

- 文件之间血统关系，默认情况下，Lineage没有打开，可以设置tachyon.user.lineage.enabled属性为true在配置文件中
- Lineage Client API (alpha)
  - 提供接口管理和获取lineage信息



# Spark/MapReduce/Shark without Tachyon

## Spark

```
scala> val file = sc.textFile("hdfs://ip:port/path")
```

## Hadoop MapReduce

```
$ hadoop jar hadoop-examples-1.0.4.jar wordcount
hdfs://localhost:19998/input
hdfs://localhost:19998/output
```

## Shark

```
CREATE TABLE orders_cached AS SELECT * FROM orders;
```

# Spark/MapReduce/Shark with Tachyon

## Spark

```
scala> val file = sc.textFile("tachyon://ip:port/path")
```

## Hadoop MapReduce

```
$ hadoop jar hadoop-examples-1.0.4.jar wordcount
tachyon://localhost:19998/input
tachyon://localhost:19998/output
```

## Shark

```
CREATE TABLE orders_tachyon AS SELECT * FROM orders;
```

# 使用

```
$ ./spark-shell
> val rdd = sc.textFile(inputPath)
> rdd.persist(StorageLevel.OFF_HEAP)
```

# 用例



## 用例一: Baidu

Framework: SparkSQL

Tachyon Storage: MEM + HDD

Under Storage: Baidu's File System

部署规模: 100+ 节点

管理存储容量: 1PB+

提升性能: 30x

## 用例二: SAAS公司

Framework: Impala

Tachyon Storage: MEM + SSD

Under Storage: S3

提升性能: 15x

# 用例



## 用例三: 石油公司

Framework: Spark

Tachyon Storage: MEM

Under Storage: GlusterFS

分析传统存储系统中的数据

## 用例四: SAAS公司

Framework: Spark

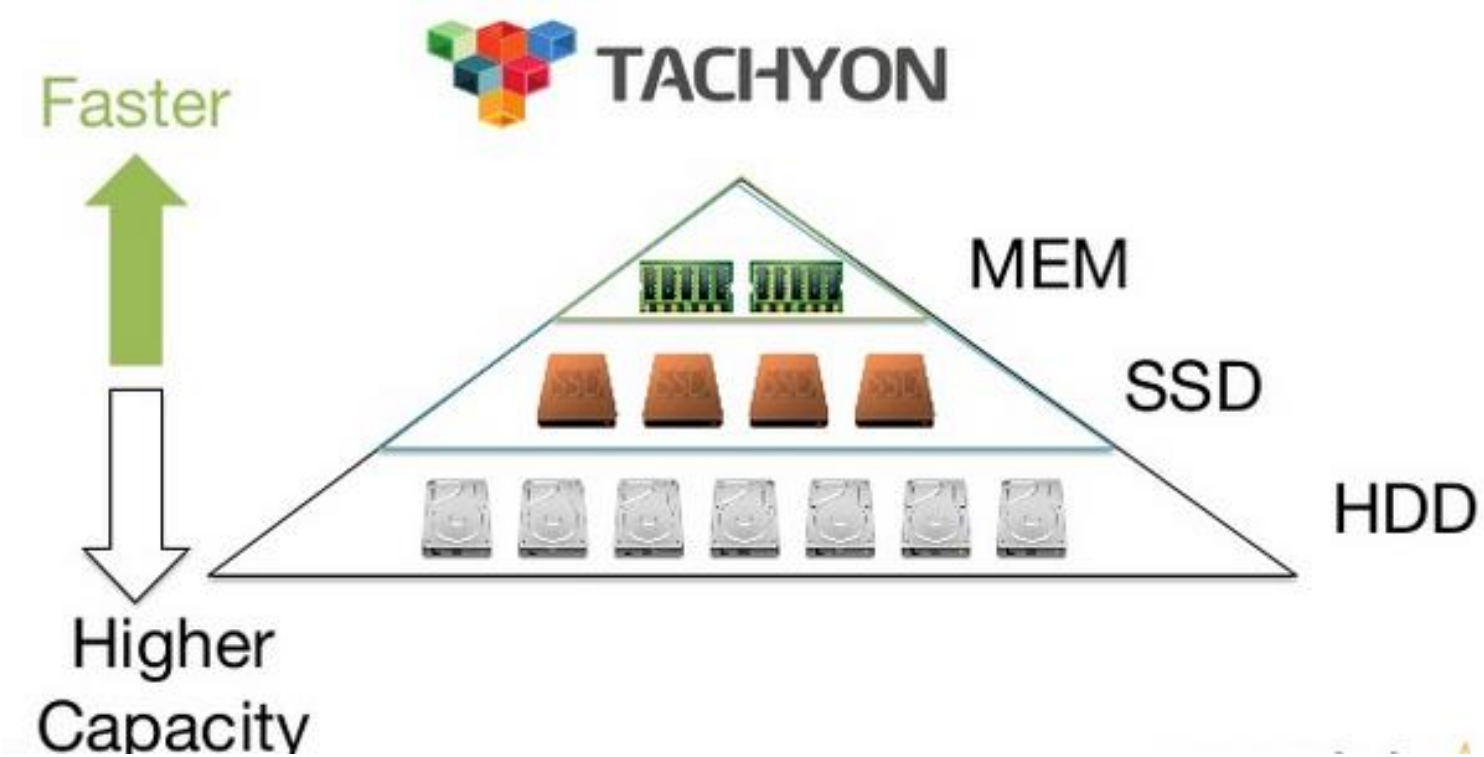
Tachyon Storage: SSD

Under Storage: S3

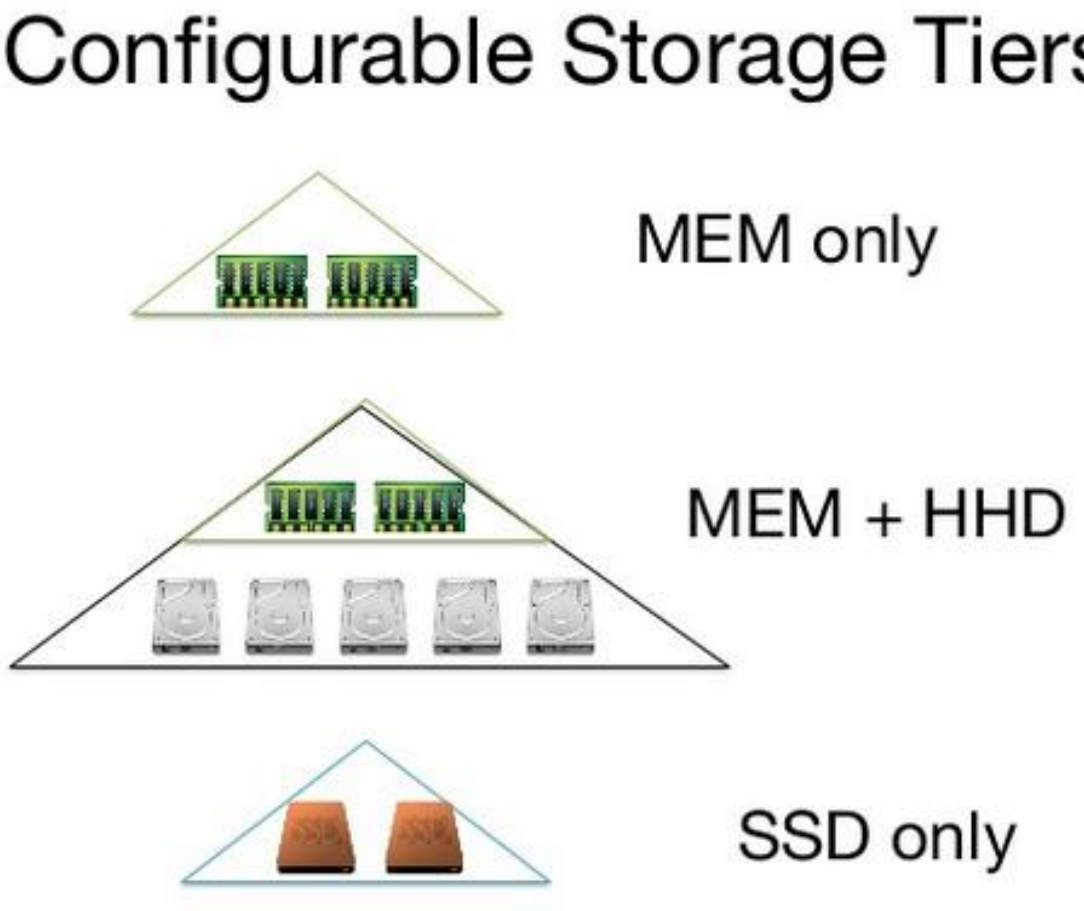
Elastic Tachyon deployment

# 如果数据超过内存大小

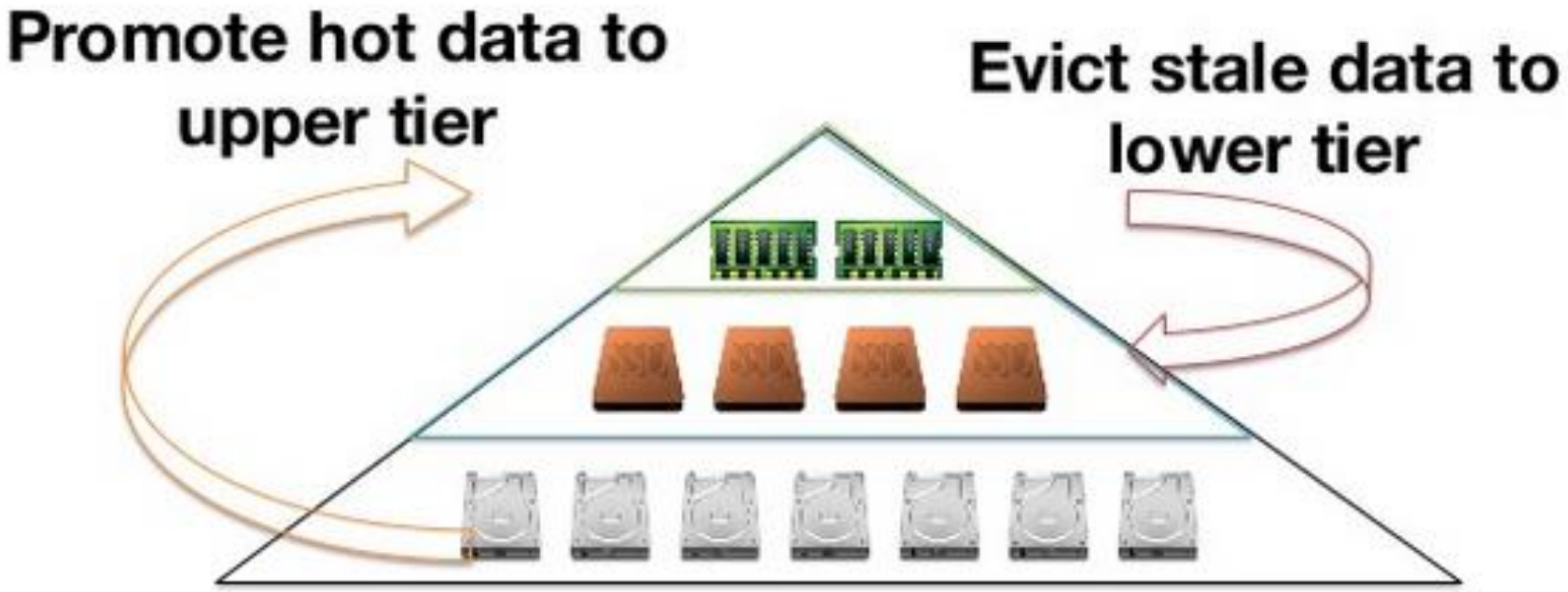
- Tiered Storage

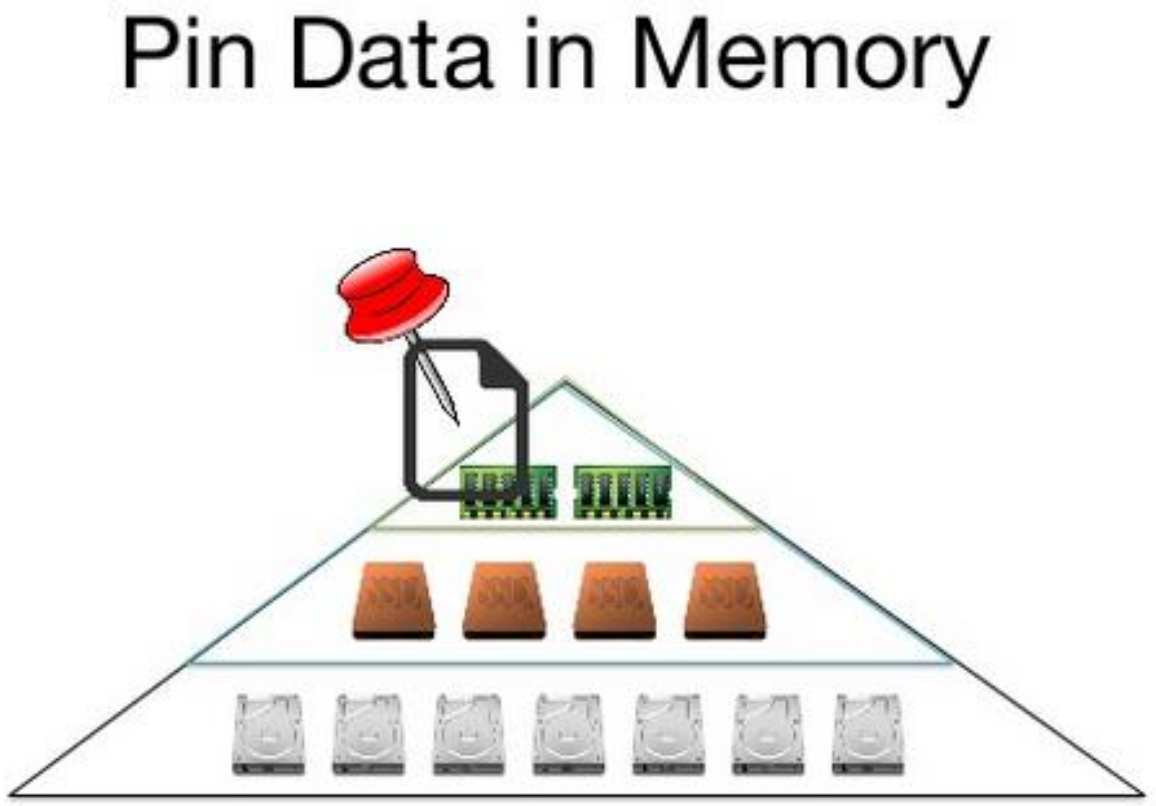


- 不仅DRAM



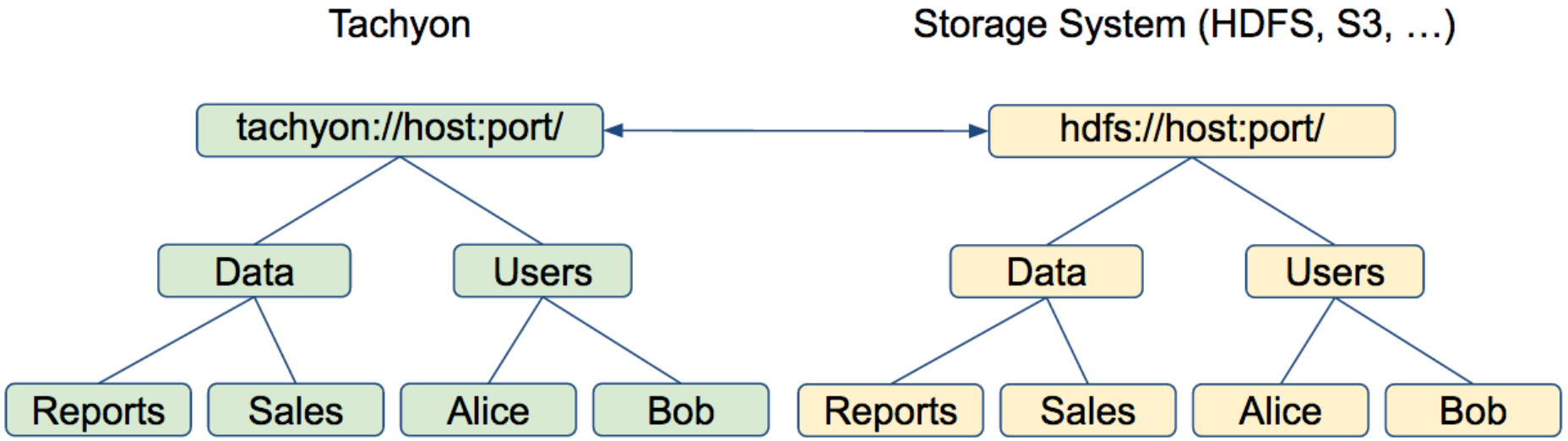
# 可插拔数据管理策略


Promote hot data to upper tier
Evict stale data to lower tier

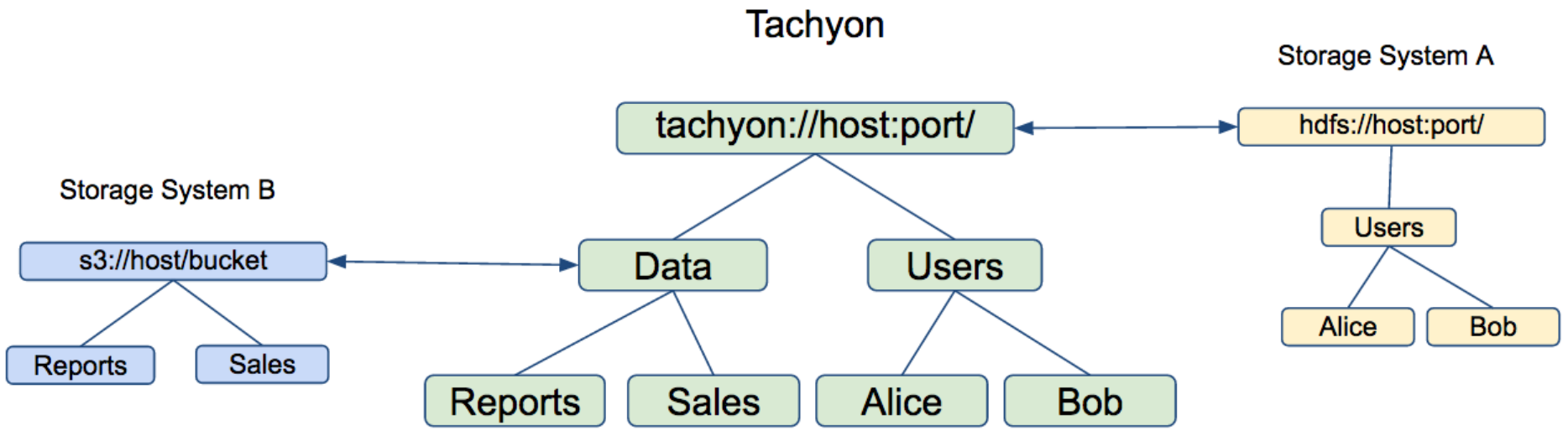


# 透明的命名

- 创建、重命名、删除对应存储层的持久化的Tachyon对象
- Tachyon的路径会被保留在存储层



- 统一的命名对于多数据源
- 跨存储系统分享数据
- mount / unmount



# 祝大家圣诞快乐！