GIAC BEIJING Dec.12.16-17

白山云科技 BAISHANCLOUD

架構 ARCHNOTES 高可用架构

分布式对象存储面临的挑战

陈闯 白山云科技

技术架构未来

thegiac.com

白山云科技
BAISHANCLOUD

目 录

一、海量小文件如何存储 (haystack)

二、如何节省存储成本 (erasure code)

三、如何实现数据的自动恢复 (paxos)

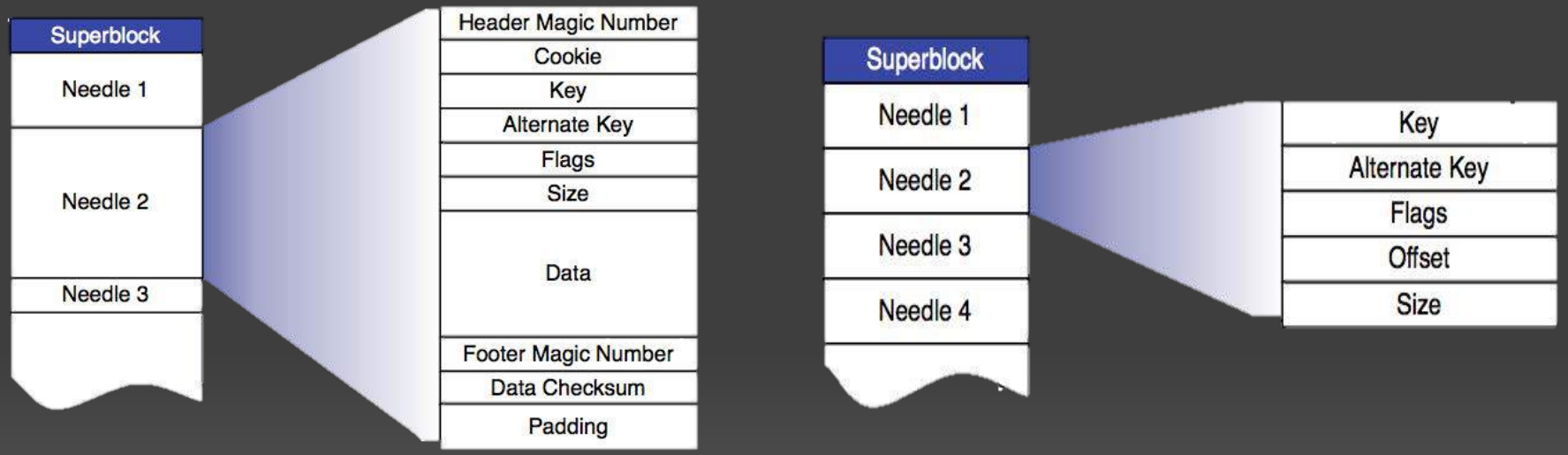
白山云科技
BAISHANCLOUD

挑战一：Too Much 小文件

图片、文档、缩略图等小文件，占据了半壁江山

小文件的问题：元数据管理、访问性能

Facebook Haystack 巨人的肩膀



将小文件合并、生成索引文件，通过查找索引定位小文件在数据文件中的位置

索引占用的字节

白山云科技
BAISHANCLOUD

存储端使用文件的SHA1作为key

Index(32byte) = key(20byte) + offset(8byte) + size(4byte)

索引消耗内存严重

白山云科技
BAISHANCLOUD

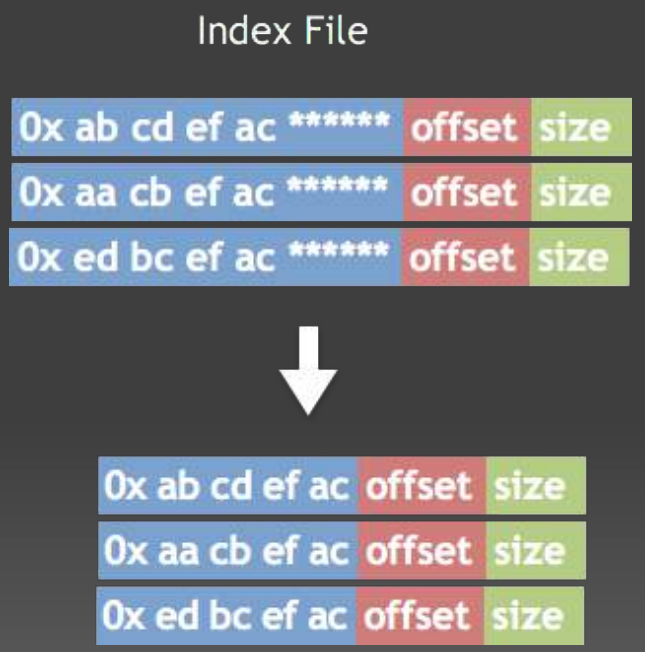
栗子：

- 服务器：12*6T盘、32G内存
- 场景：存放10亿个10KB文件。
- 使用：存储占用 9TB、内存占用29GB

磁盘使用12%，内存使用90%

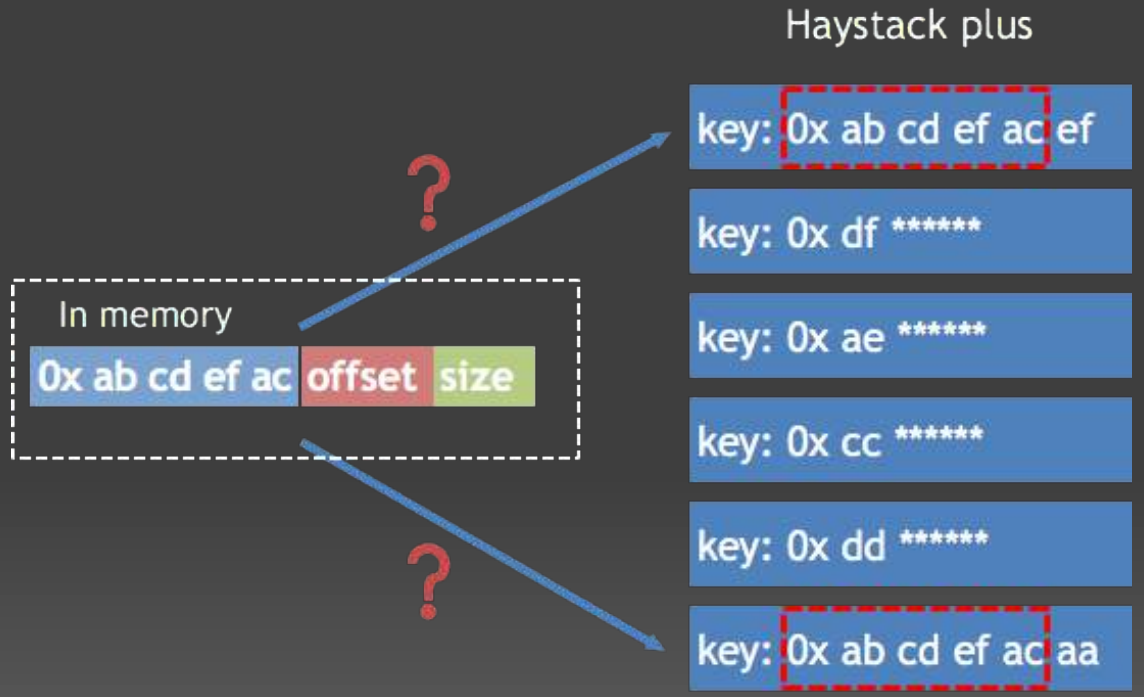
索引文件存部分key

白山云科技
BAISHANCLOUD



只存前4字节的key, 每个索引省16字节

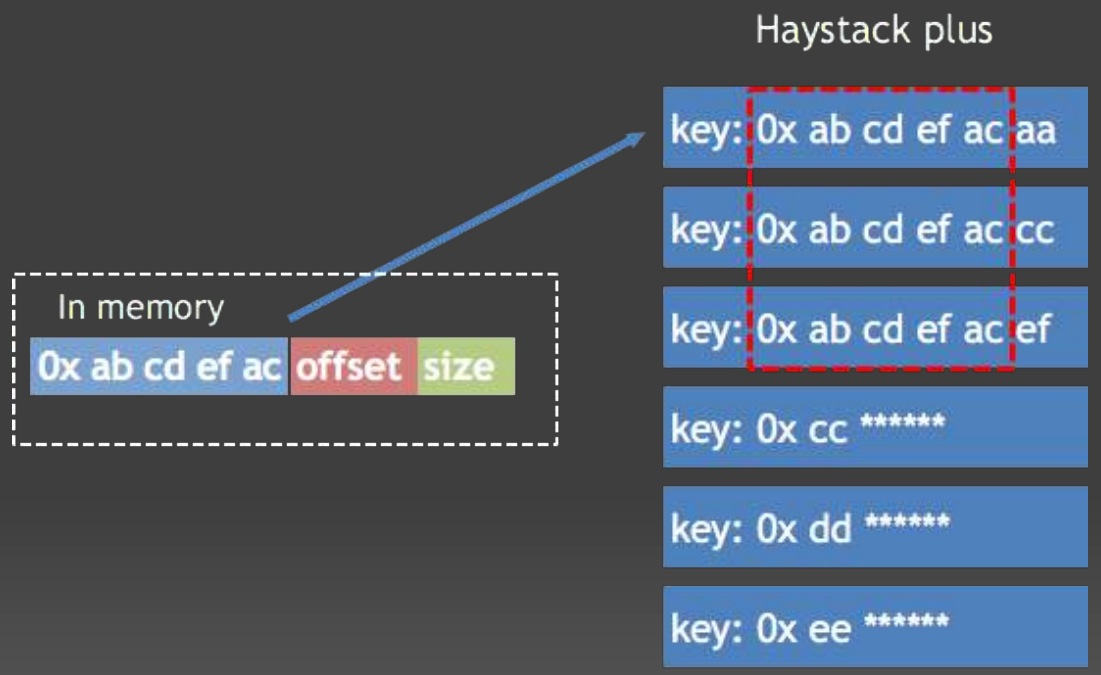
部分索引带来的问题



无法准确定位完整key对应的文件位置

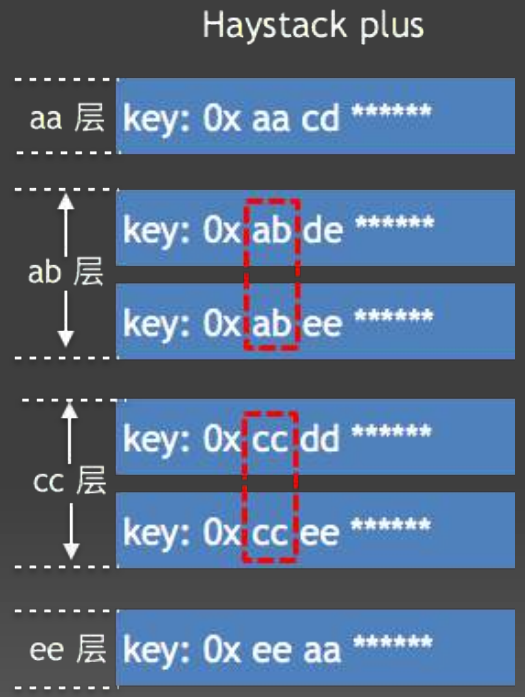
needle顺序存放

白山云科技
BAISHANCLOUD



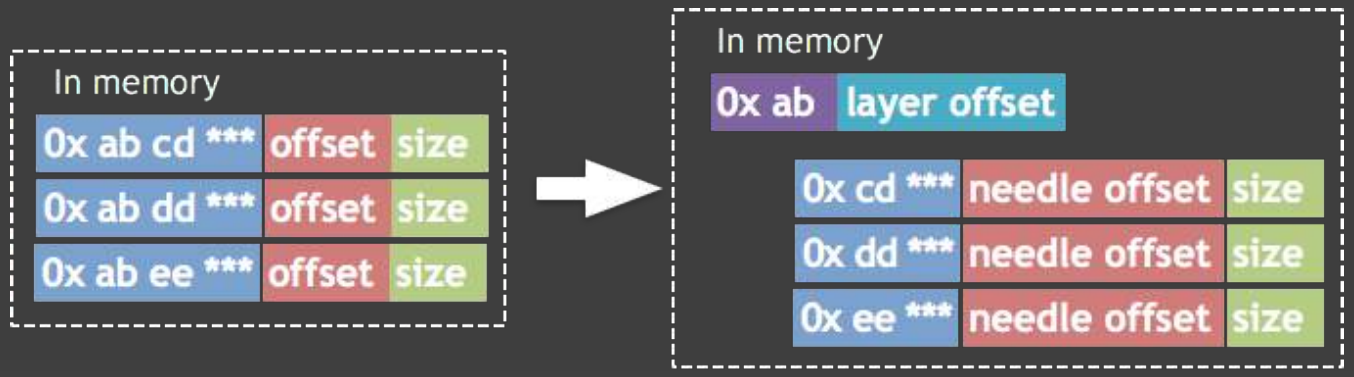
Needle按照key的顺序存放， 当key前缀相同，
通过顺序查找定位needle

对索引进行分层



按照相同key的前缀，进行分层

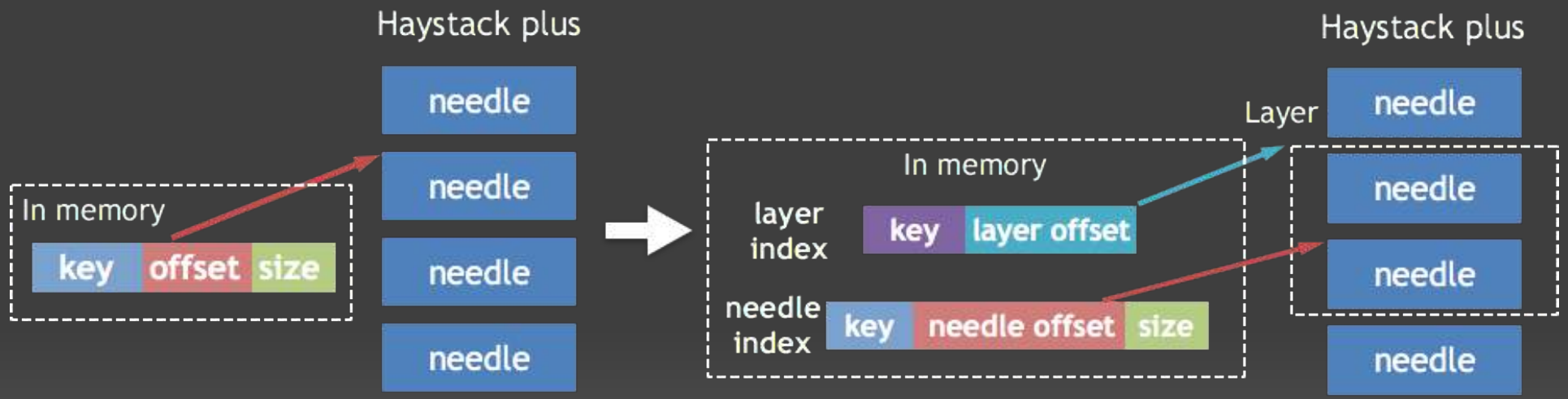
分层减少key的字节



将key的公共前缀从needle index中提取出来

分层减少offset的字节

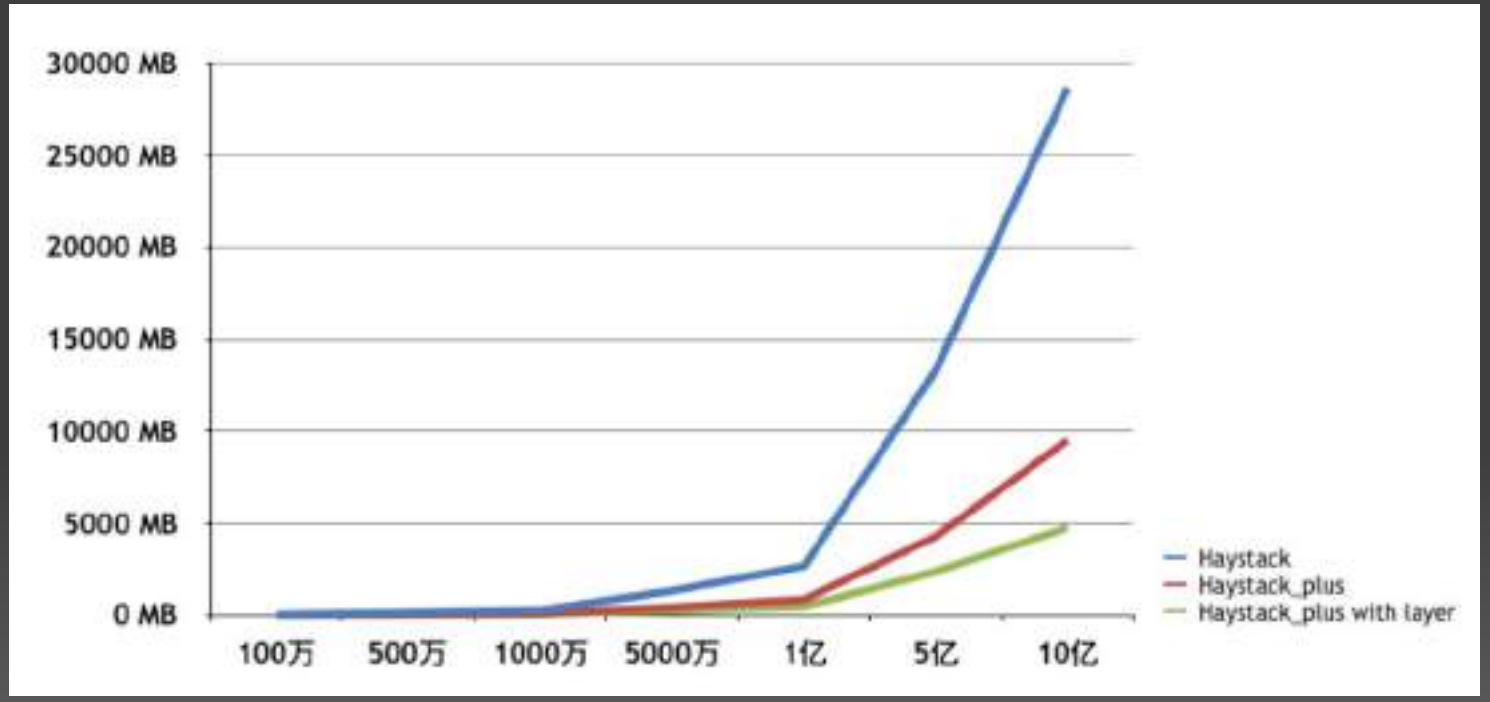
白山云科技
BAISHANCLOUD



offset只需做层内的地址偏移，
offset占用的字节只需要满足层内的地址空间

再也不担心内存了

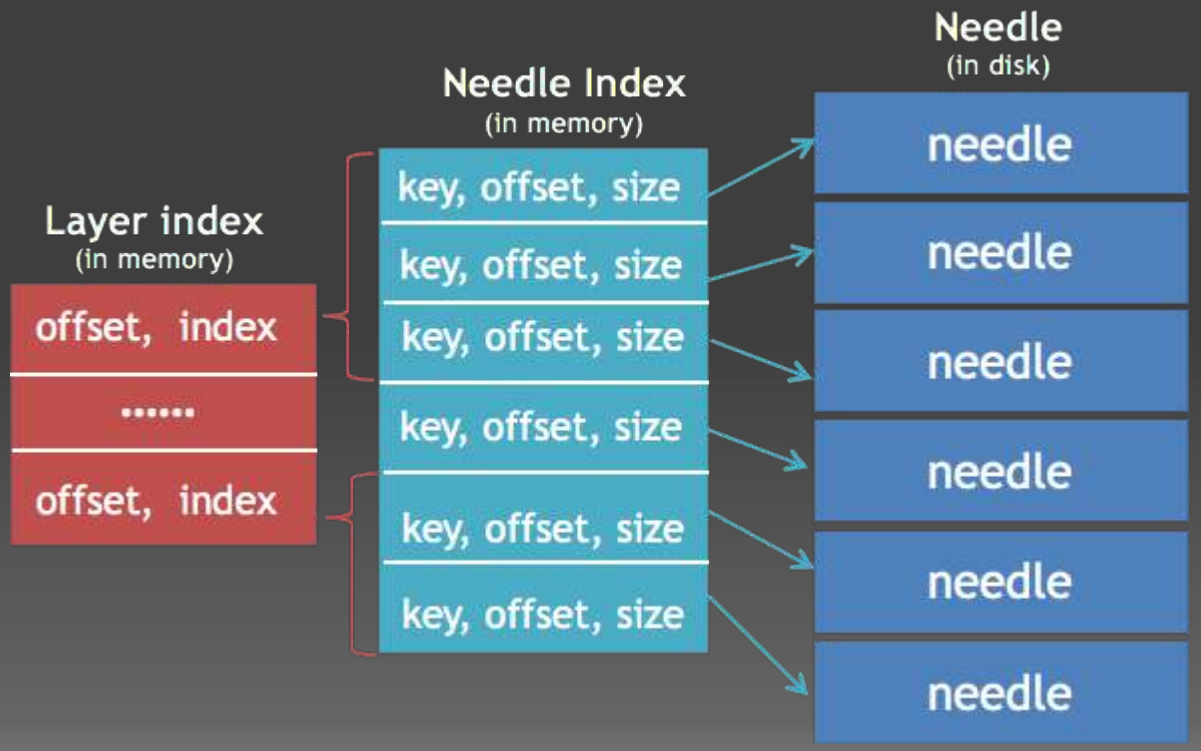
白山云科技
BAISHANCLOUD



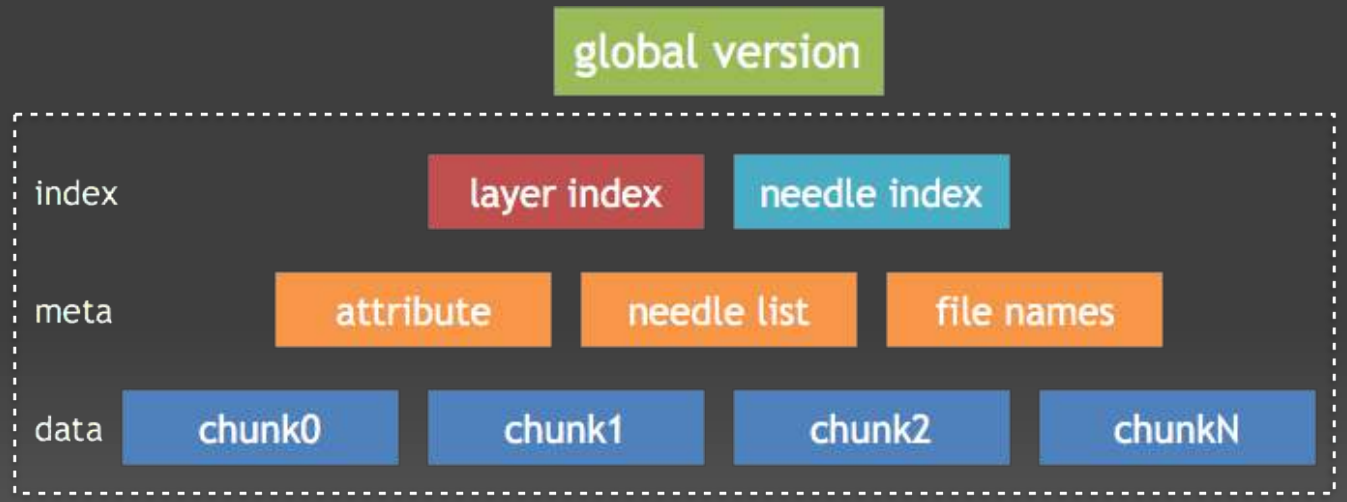
- 通过压缩索引，内存占用减少60%
- 再通过索引分层，内存占用减少80%

我们称之为haystack_plus

白山云科技
BAISHANCLOUD



Haystack_plus 文件组织形式



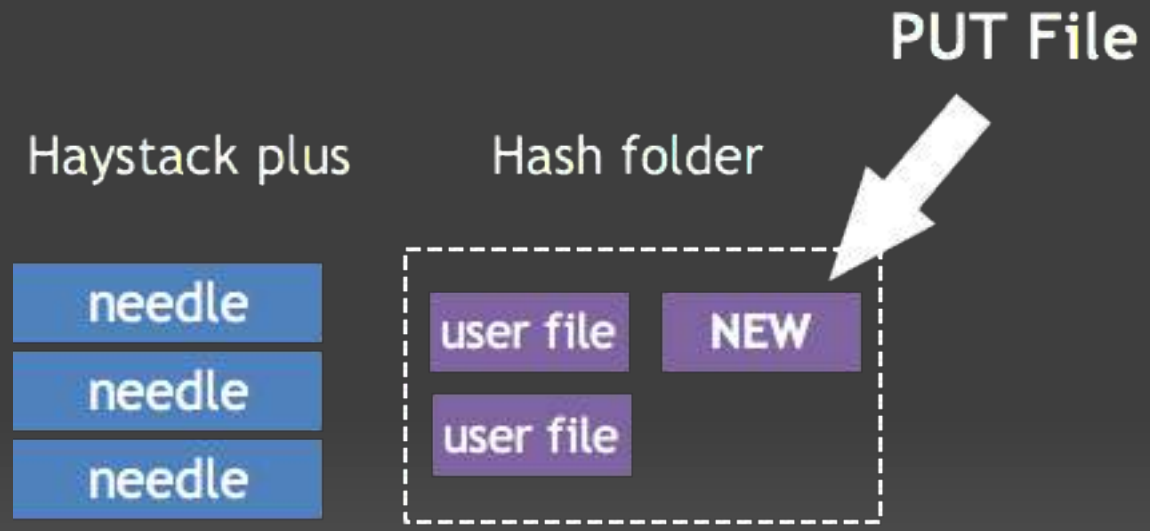
数据文件被拆分成多个chunk，减少数据恢复的时间

白山云科技
BAISHANCLOUD

Haystack_plus 如何保存新上传的文件

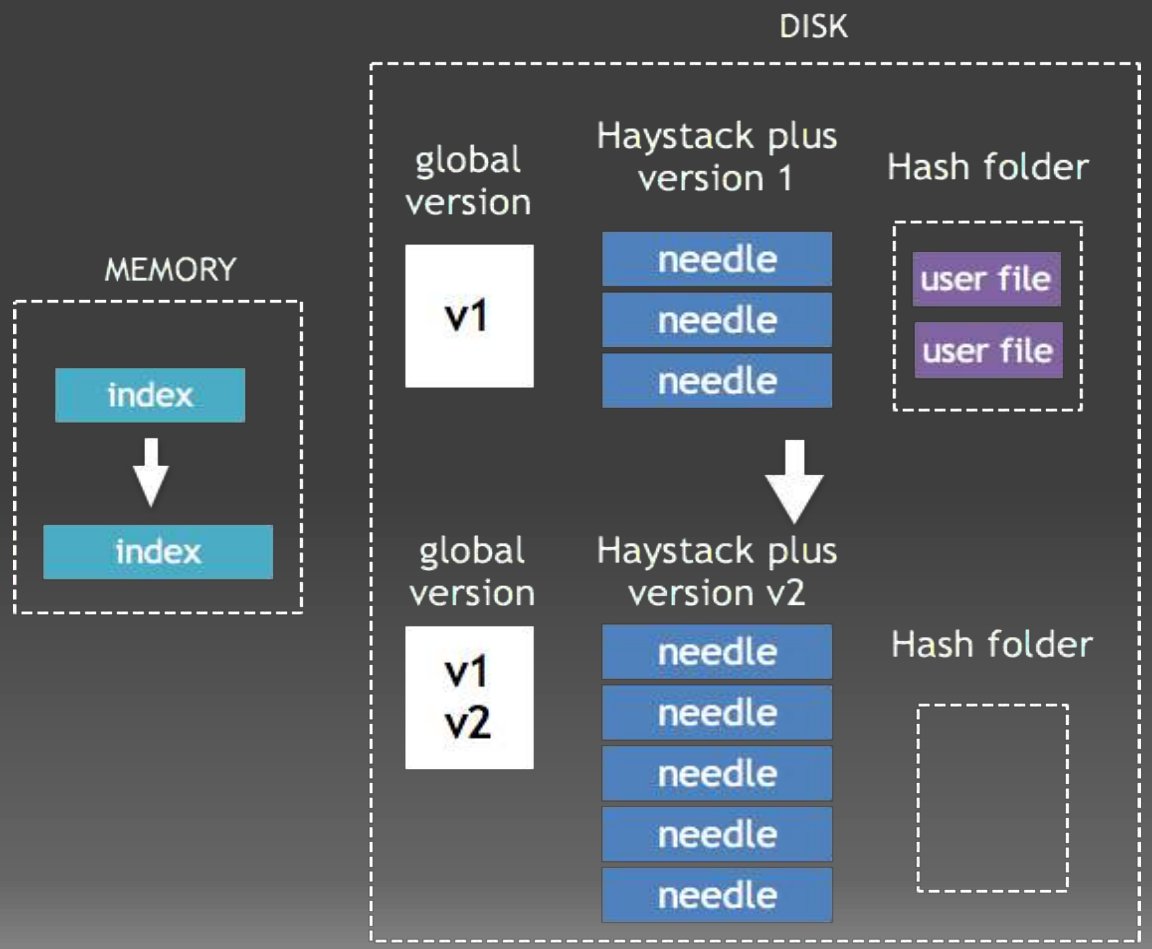
Hash 目录

白山云科技
BAISHANCLOUD



新上传的文件会保存到hash目录下，临时存放

定期版本更替

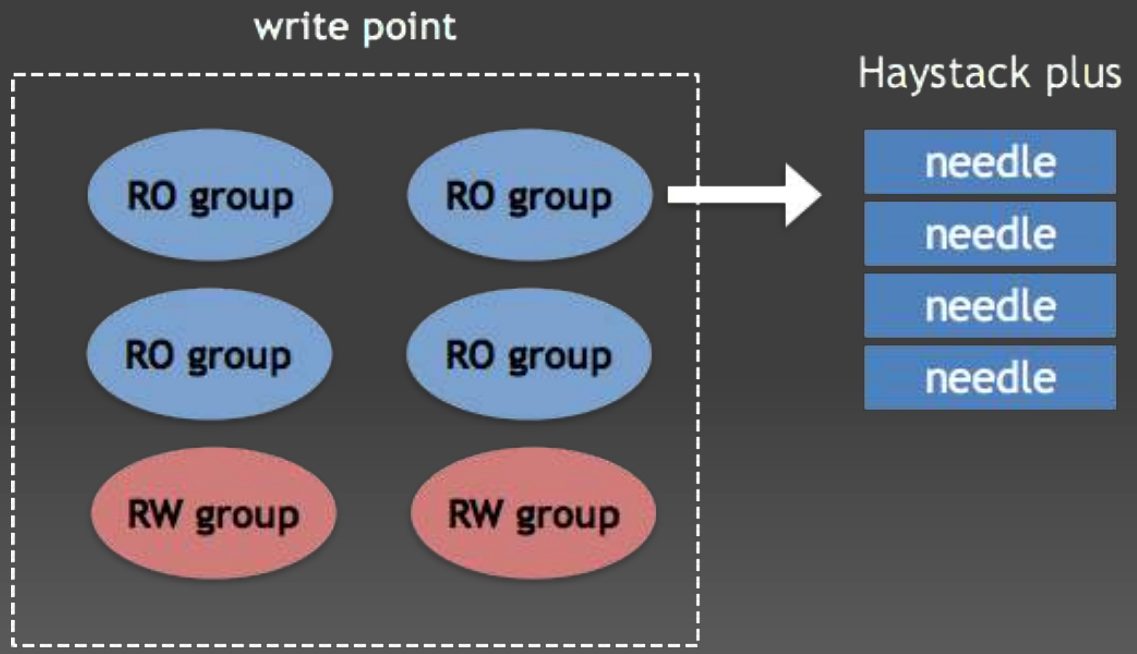


- 定期将Hash目录下的文件重新合并，生成新版本haystack_plus。
- 版本名为所有文件名的sha1

上传频率少

白山云科技 BAISHANCLOUD

每个group都是集群的上传点，达到一定阀值，会置成readonly，不接受上传写入。

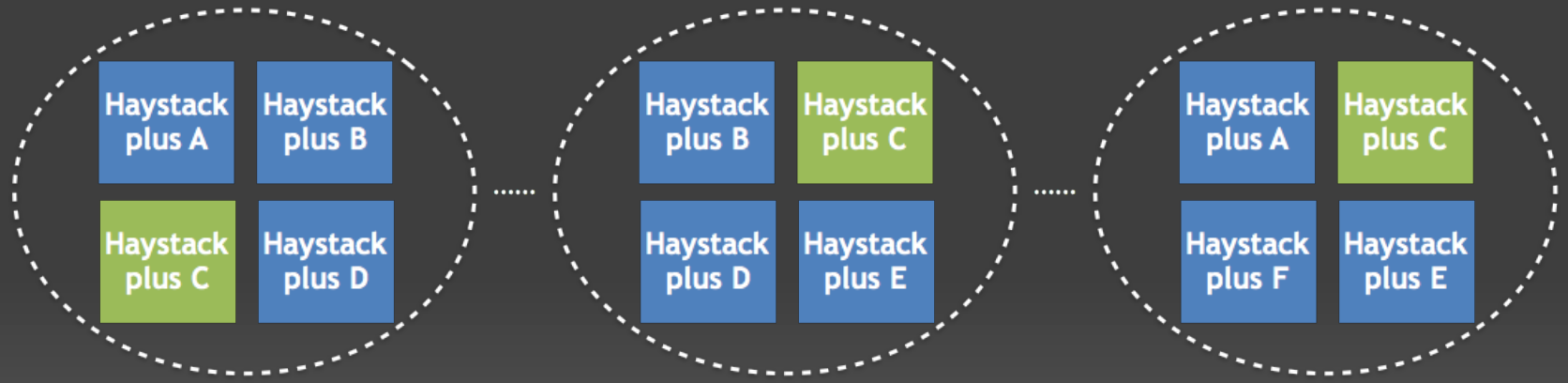


将readonly的group做成haystack_plus，避免多次合并haystack_plus造成的IO消耗。

白山云科技
BAISHANCLOUD

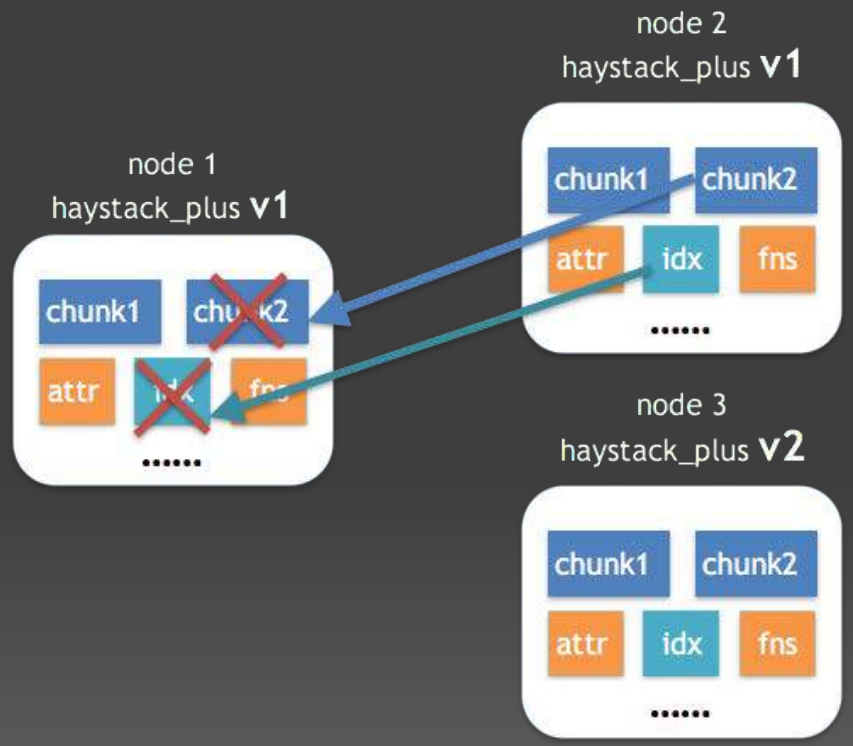
haystack_plus如何保证高可靠性

Haystack_plus 以三副本形式存在



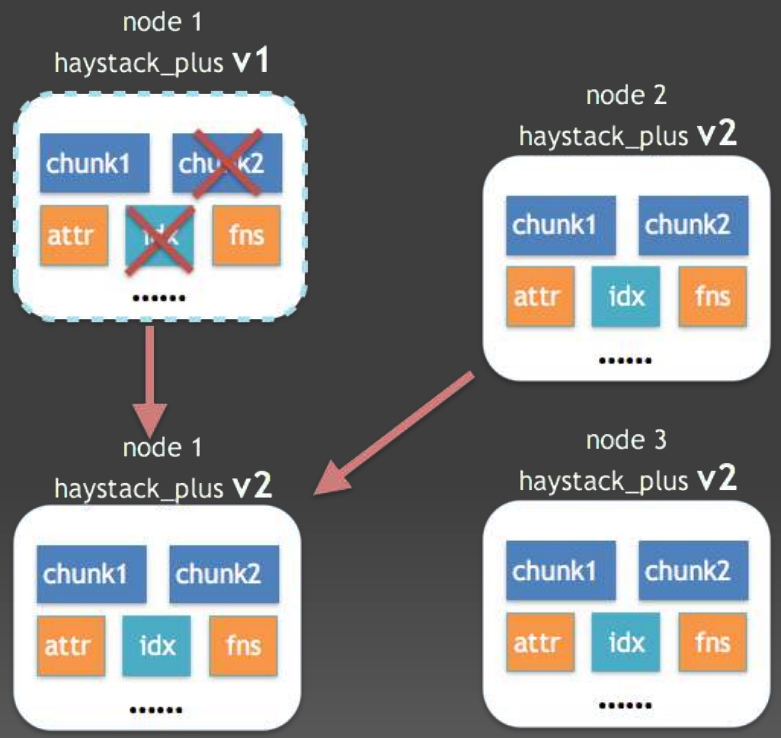
一组文件会做成haystack_plus，
并且存放在三台机器上，增加可靠性

恢复场景一：有相同版本的haystack_plus



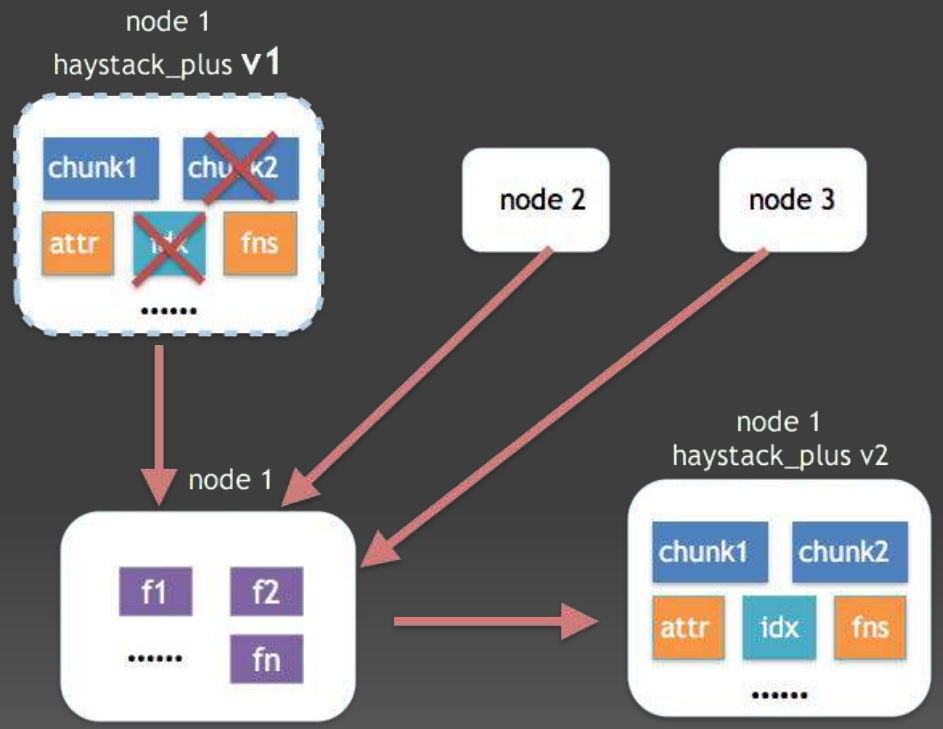
直接从副本机器上，下载相同版本的文件

恢复场景二：没有相同版本haystack_plus



直接从副本机器上，下载高版本的haystack_plus，
同时将缺失的文件同步过来

恢复场景三：无法同步完整的haystack_plus



从副本机器上，同步所有用户文件，
生成新的haystack_plus

haystack_plus的好处

白山云科技
BAISHANCLOUD

- 碎片降低80%：优化最小存储单元
- IO减少50%：去除inode查询
- 对比facebook，索引内存降低80%

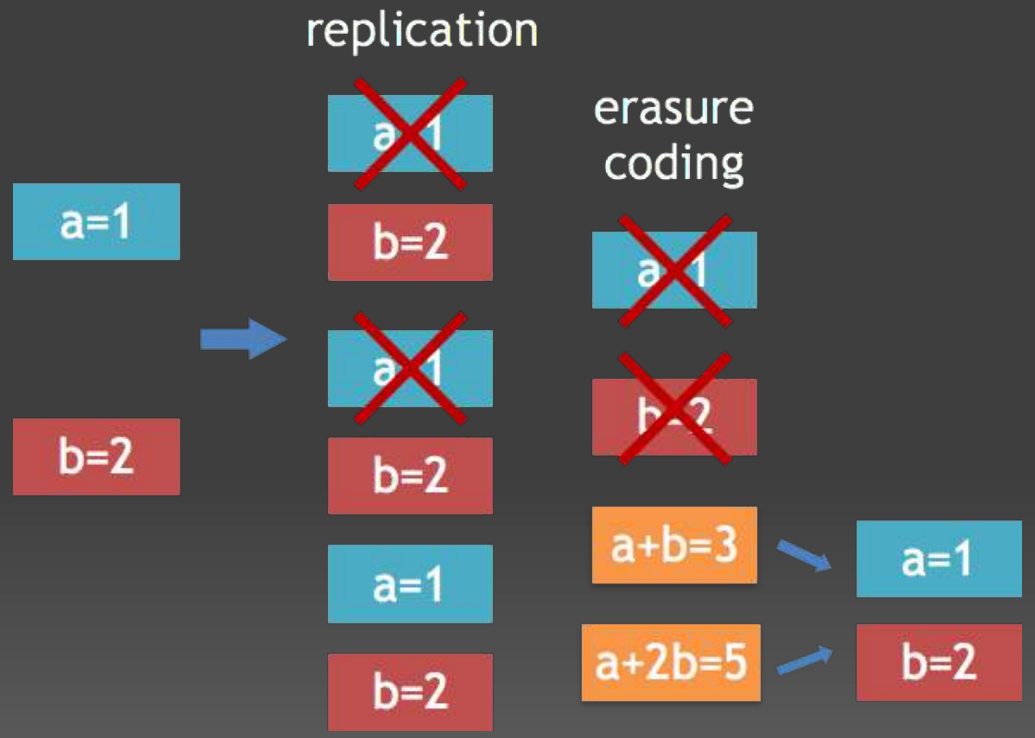
白山云科技
BAISHANCLOUD

挑战二：如何解决存储成本

每个文件都以三副本的形式存在，钱啊

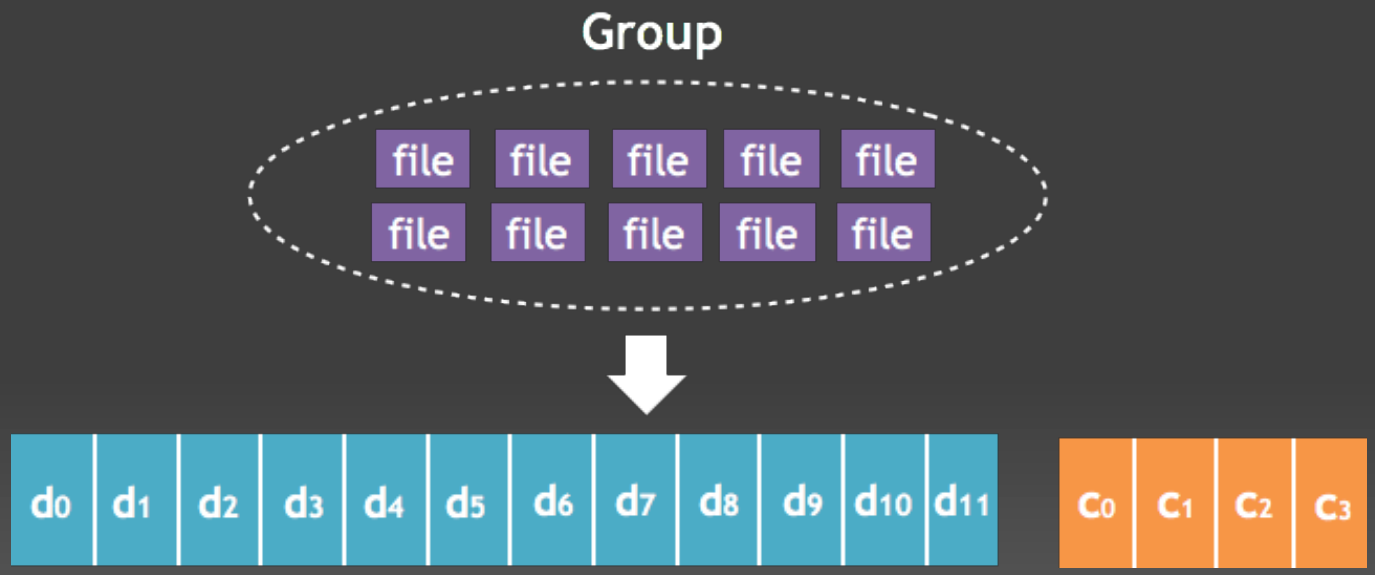
erasure code是什么？

白山云科技
BAISHANCLOUD



都可以在损坏两块数据的情况下，
保证数据不丢失，但ec节省更多成本

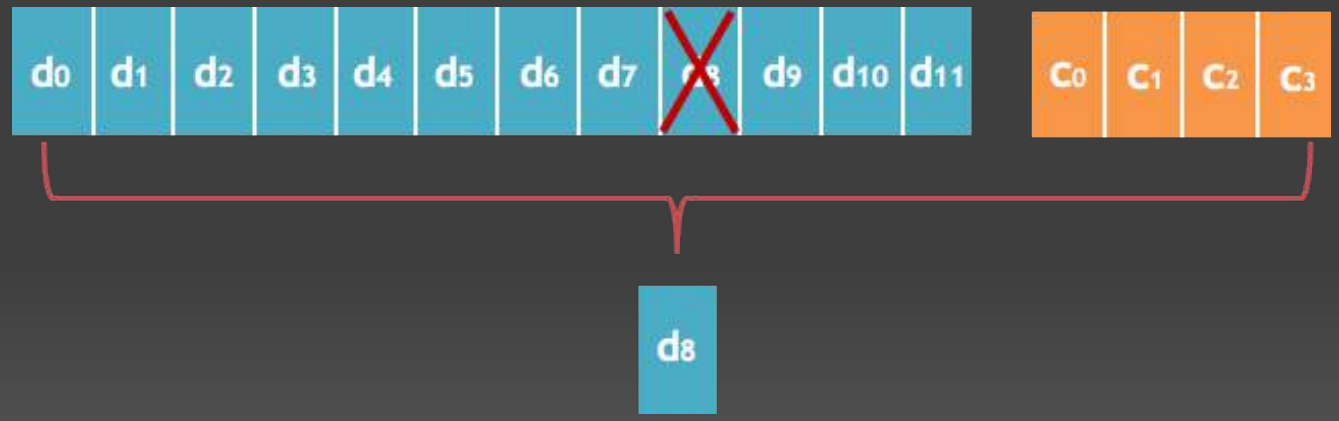
通过erasure code，节约存储成本



- ec 成本消耗: (12+4)/12 = 1.33
- 三副本成本消耗: (12*3)/12 = 3

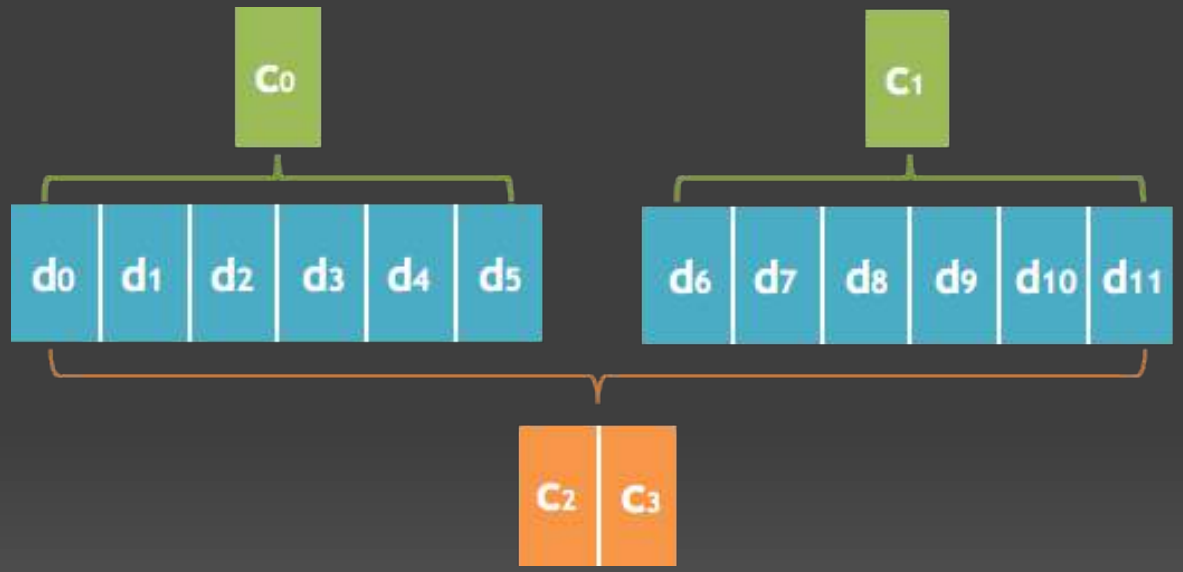
恢复速度是个问题

白山云科技
BAISHANCLOUD



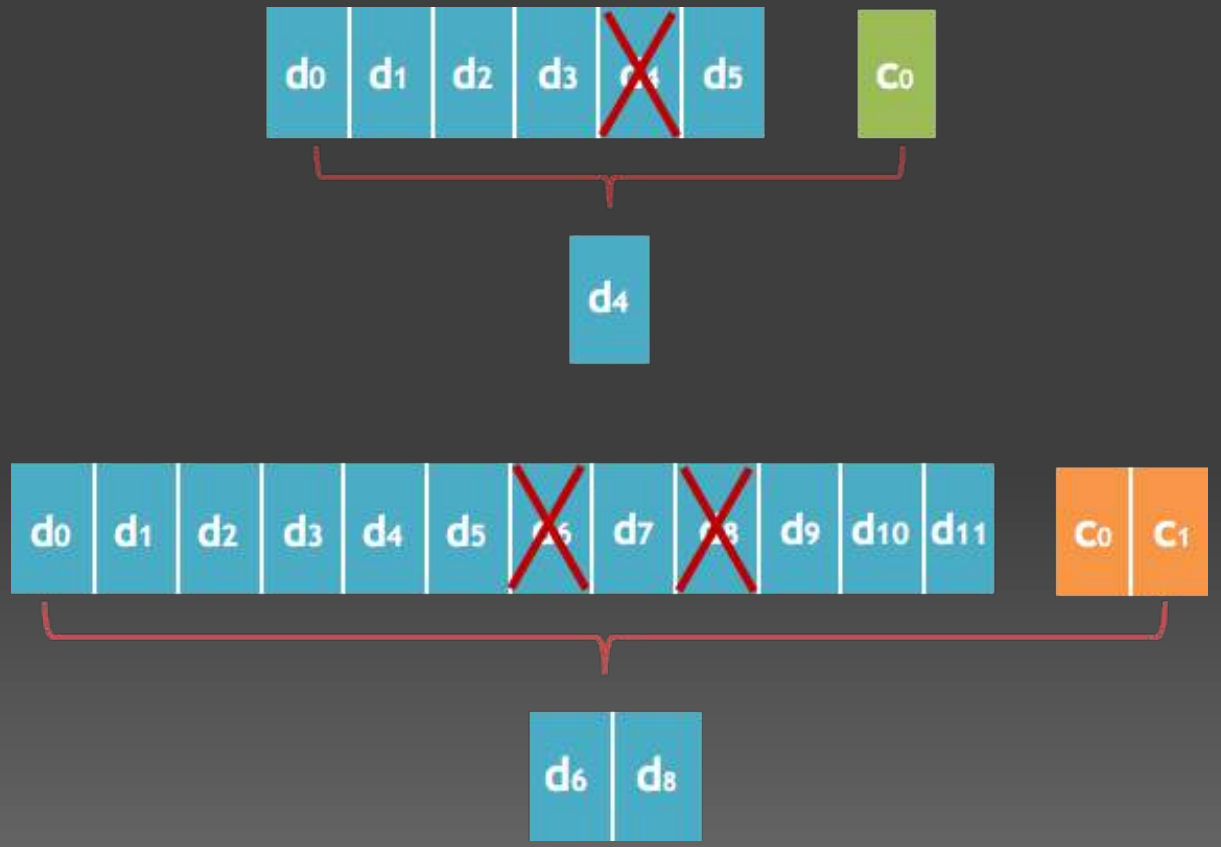
一个数据块损坏，需要得到所有数据，再进行恢复

LRC (Local Reconstruction Code)



将数据块前后分成两组，每组各生成一个校验块。
再使用所有的数据块，生成两个检验块。

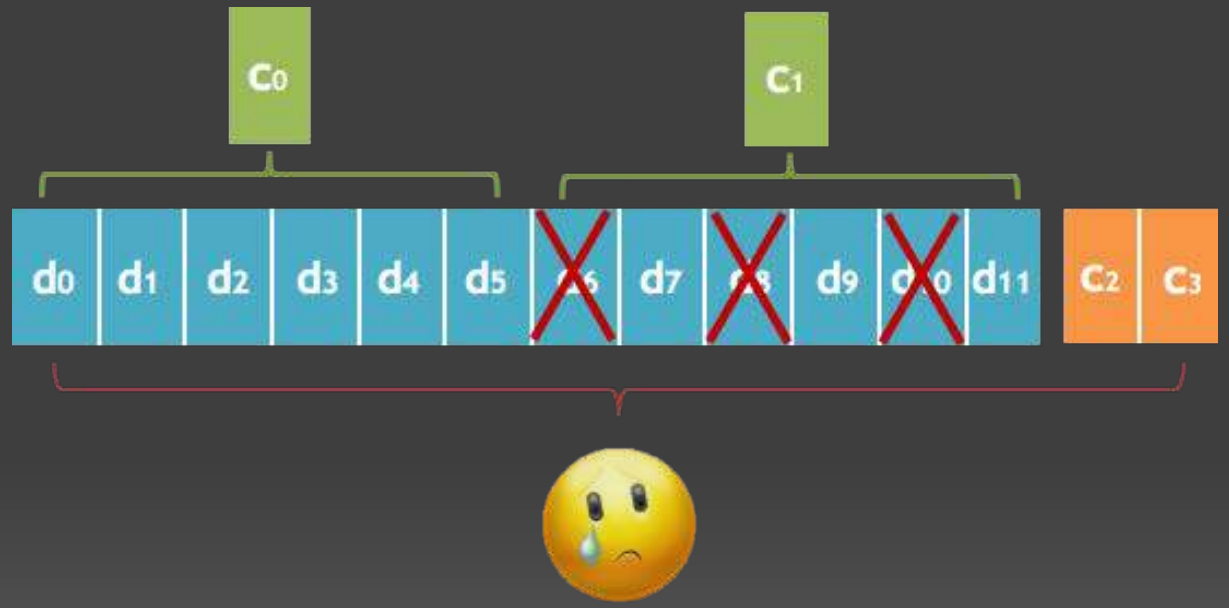
一个数据块损坏只需要局部恢复



一块数据损坏，只需要读取一半的数据进行恢复

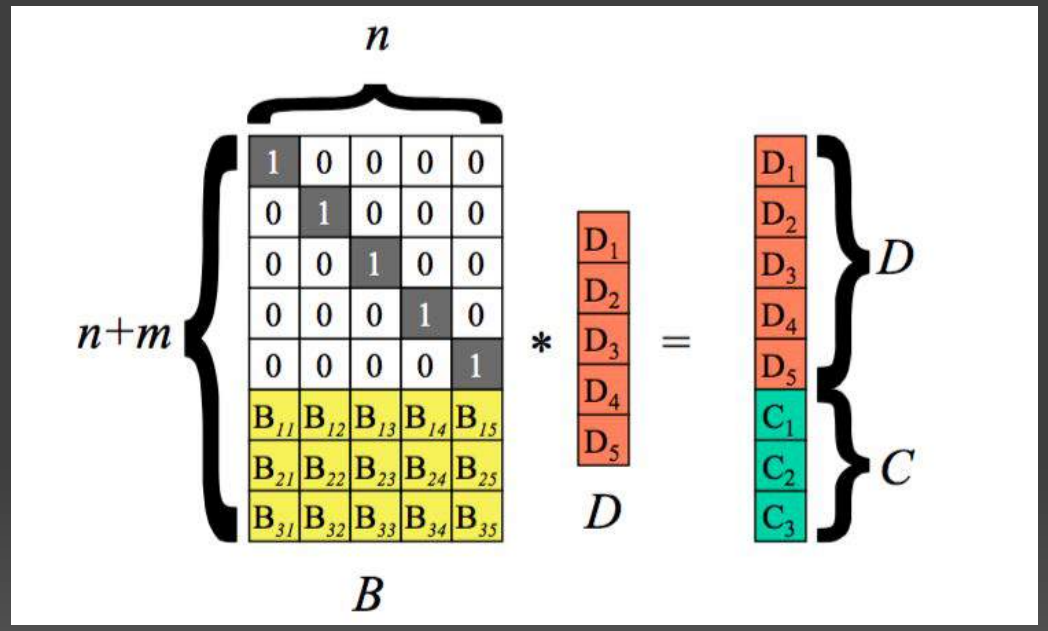
LRC 不爽的地方

白山云科技
BAISHANCLOUD



虽然LRC能减少恢复过程中读取的数据，
但是数据的可靠性降低了

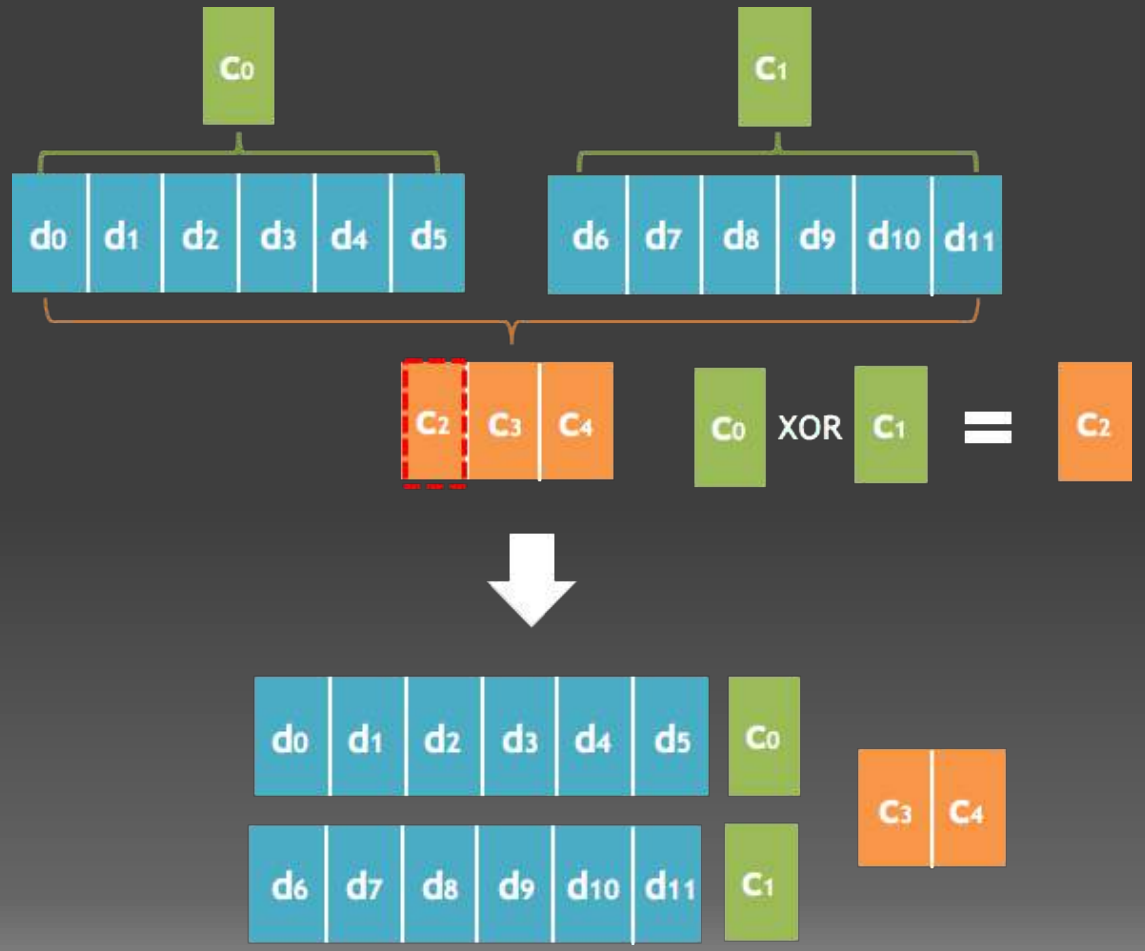
Reed–Solomon 算法 和 Vandermonde 矩阵



$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{bmatrix}$$

$C_1 = D_1+D_2+D_3+D_4+D_5$,
在伽罗华域中，加法等同对应位的异或。
实际运算：$C_1 = D_1 \hat{} D_2 \hat{} D_3 \hat{} D_4 \hat{} D_5$

提高可靠性



$C_0 = d_0 \hat{} d_1 \hat{} d_2 \hat{} d_3 \hat{} d_4 \hat{} d_5$

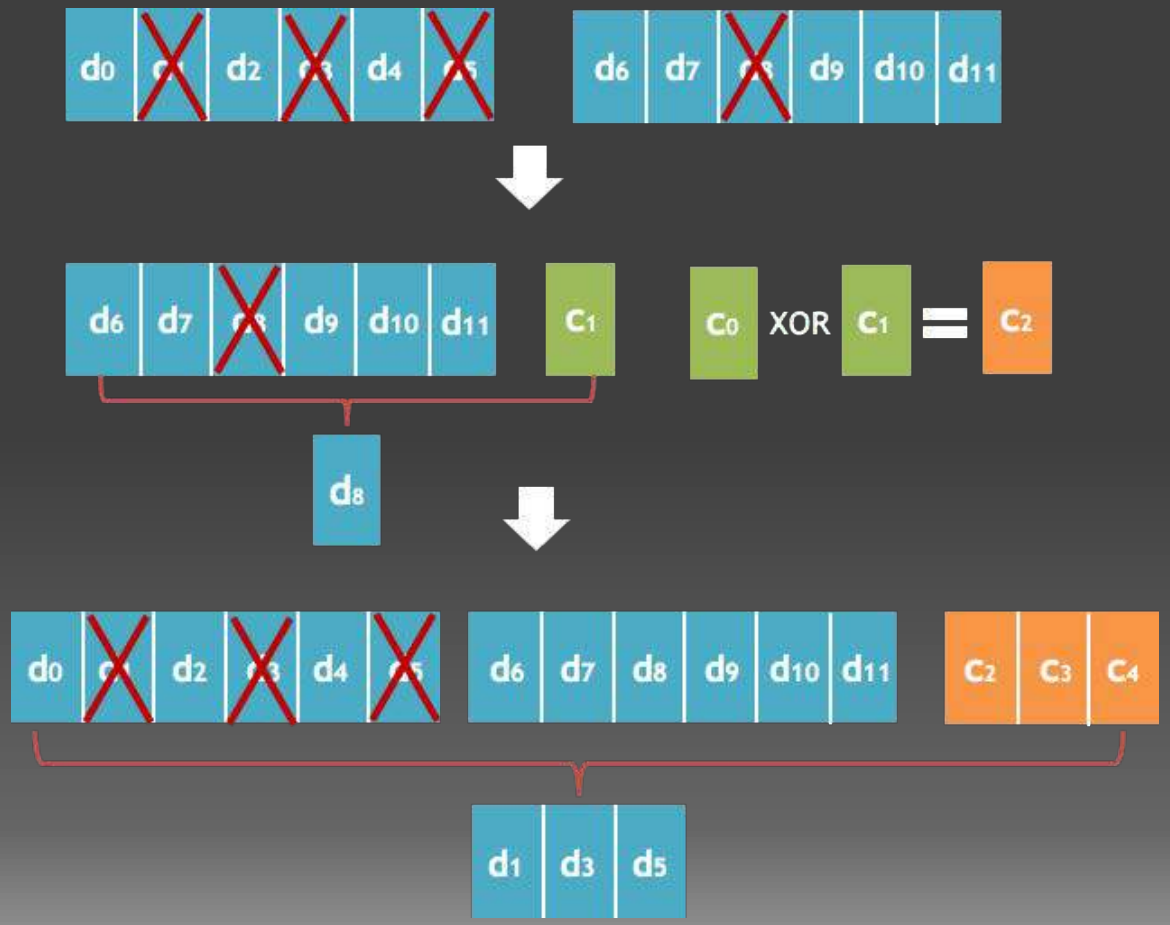
$C_1 = d_6 \hat{} d_7 \hat{} d_8 \hat{} d_9 \hat{} d_{10} \hat{} d_{11}$

$C_2 = d_0 \hat{} d_1 \hat{} d_2 \hat{} d_3 \hat{} d_4 \hat{} d_5 \hat{} d_6 \hat{} d_7 \hat{} d_8 \hat{} d_9 \hat{} d_{10} \hat{} d_{11} = C_0 \hat{} C_1$

只存放C_3和C_4,
C_2由C_0和C_1计算出来
最多可恢复出4个数据块损坏,

如何恢复4个数据块损坏

白山云科技
BAISHANCLOUD



接近4个校验块的可靠性

白山云科技
BAISHANCLOUD

3个数据块损坏，恢复概率100%

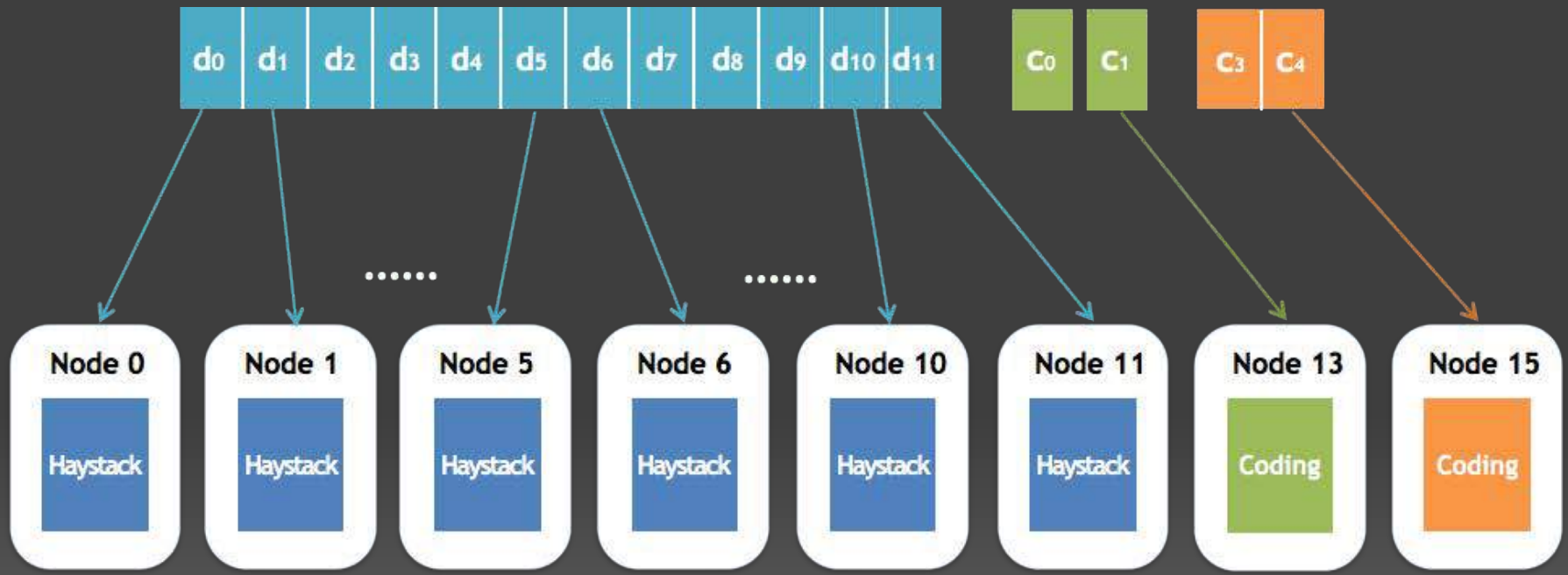
4个数据块损坏，恢复概率86%

白山云科技
BAISHANCLOUD

EC集群，怎么做数据恢复

EC的组织形式

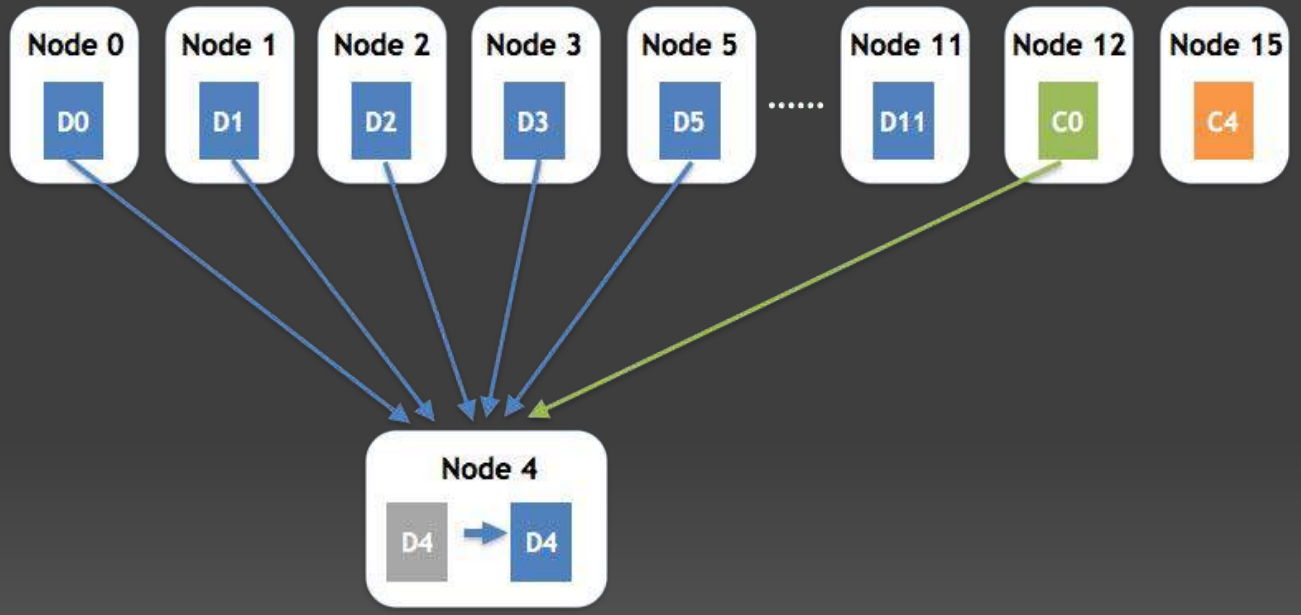
白山云科技
BAISHANCLOUD



每个数据chunk都是一个Haystack,
各个chunk分配在不同的机器上,
每个机器记录chunk序号

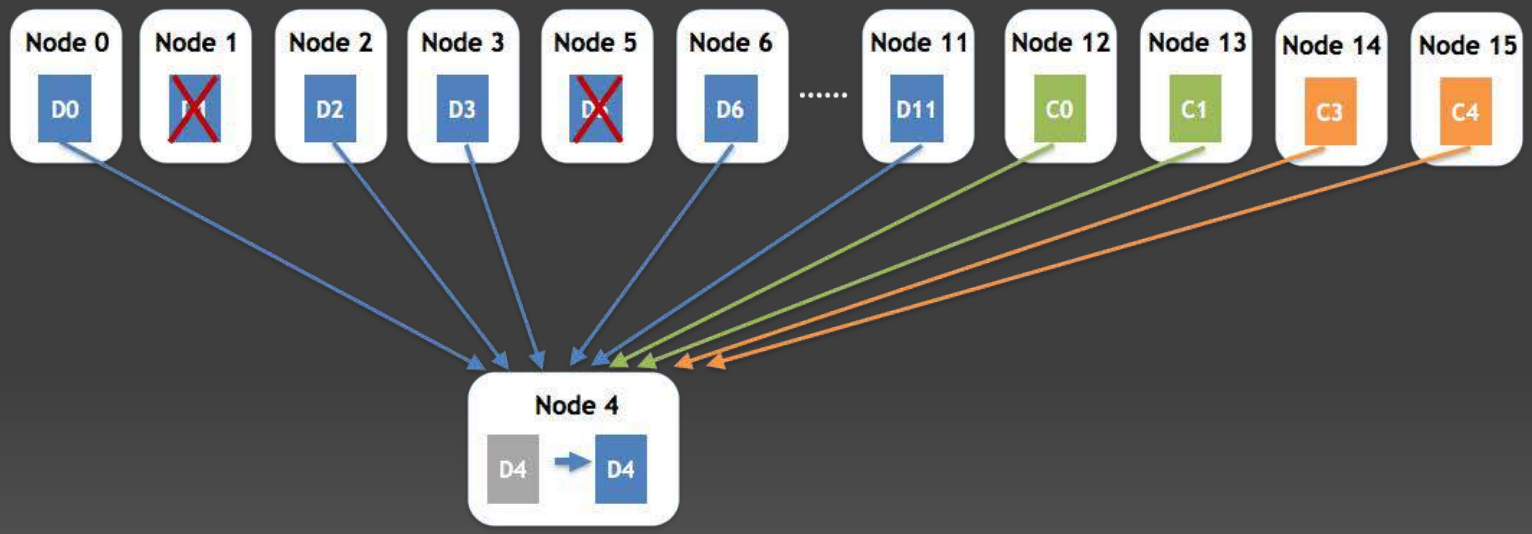
恢复场景一

白山云科技
BAISHANCLOUD



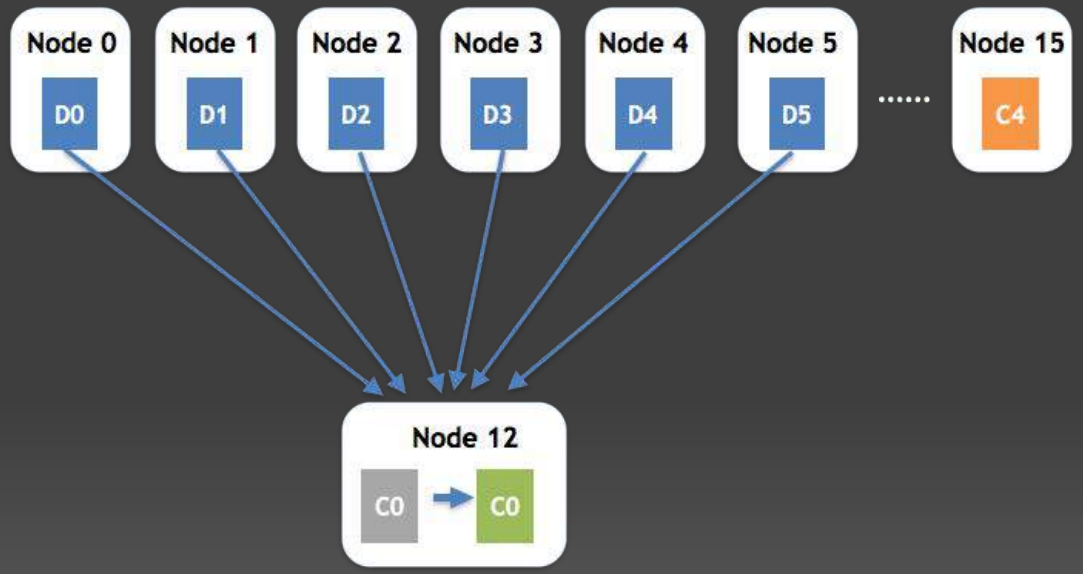
前6个数据块只有D4损坏，
直接下载另外五个数据块和对应的一个校验块，
使用EC进行数据恢复

恢复场景二



前6个数据块中超过一块损坏，
只能下载全部未损坏的数据块和校验块
使用EC进行数据恢复

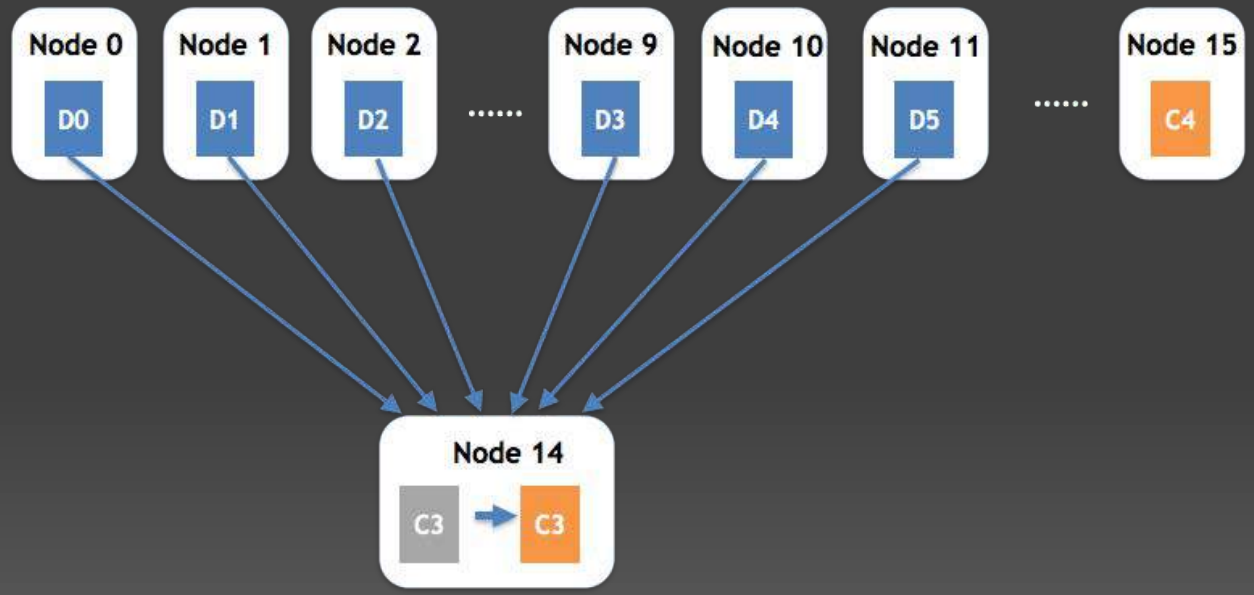
恢复场景三



如果C0或C1损坏，只需要下载对应的6个数据块，
用EC重新生成校验块

恢复场景四

白山云科技
BAISHANCLOUD



如果C3或C4损坏，需要下载全部数据块，
用EC重新生成校验块

EC的好处

白山云科技
BAISHANCLOUD

- 降低存储的成本
- 增加数据的可靠性
- 使用改进的LRC算法，节省一个校验块

白山云科技
BAISHANCLOUD

挑战三：如何自动进行数据恢复

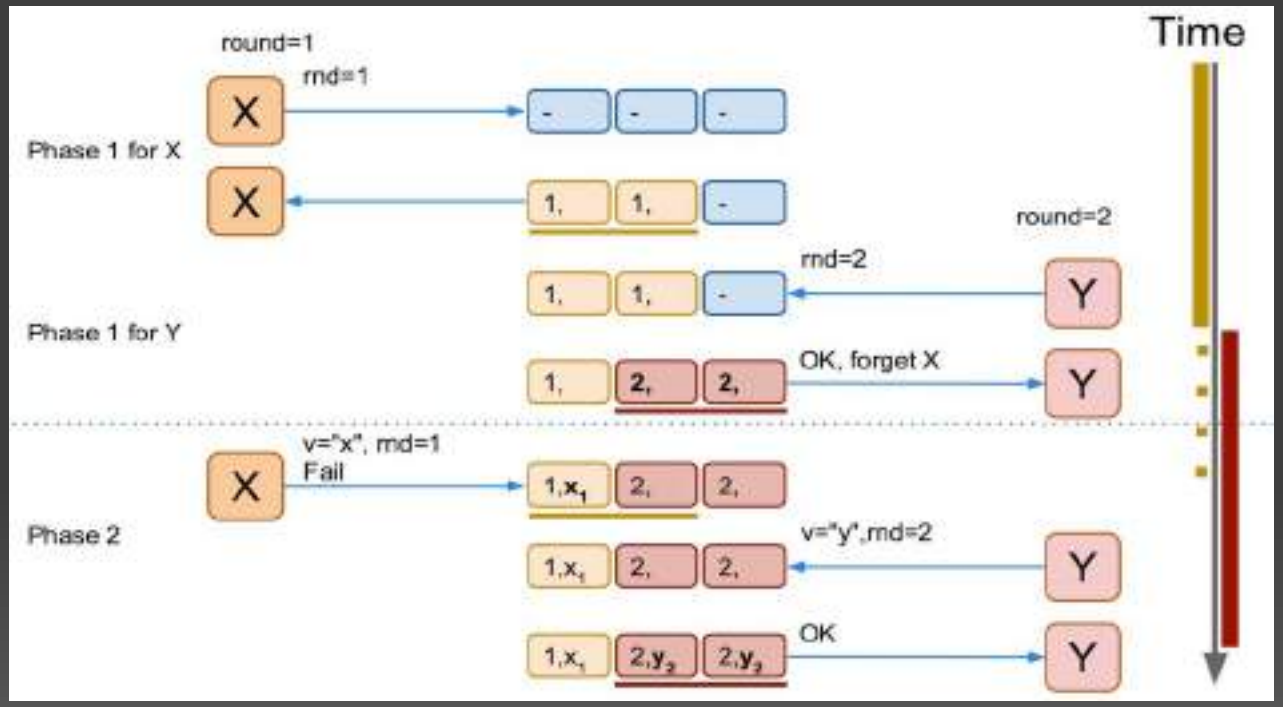
每个EC都组成一个自我恢复的小集群，有几十万个，
如何不用人为干预、自动管理呢?

各个节点如何保存一致的信息

EC集群中各个节点需要保存共同的信息
包括集群中成员列表、集群中的leader等，
需要通过这些信息进行集群管理、数据恢复等操作。

集群有可能出现网络异常、宕机、磁盘损坏等各种情况。
如何保证各个节点之间保存的信息都是一致的呢?

Paxos、 Paxos、 Paxos，重要的事情说三遍



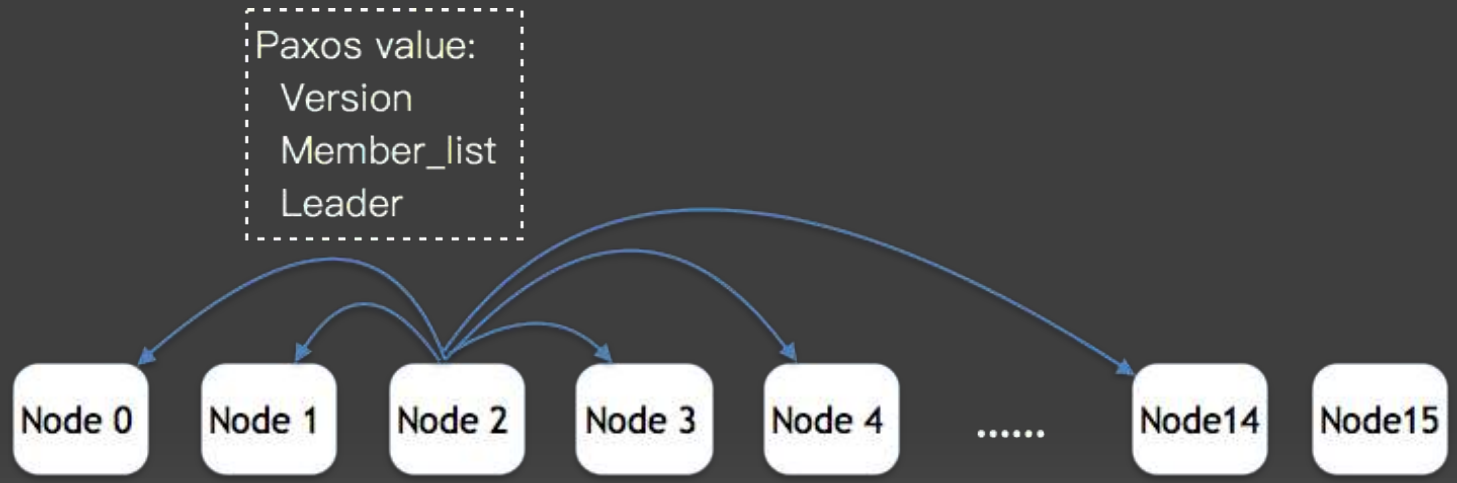
Google Chubby的作者Mike Burrows说过：“这个世界上只有一种一致性算法，那就是Paxos，其它的算法都是残次品。”

有了paxos 再恶劣的情况也不用担心了

分布式系统，不可避免的会发生以下错误：
进程可能会慢、被杀死或者重启，消息可能会延迟、丢失、重复

Paxos算法解决的问题是在一个可能发生上述异常的分布式系统中如何就某个值达成一致，保证不论发生以上任何异常，都不会破坏决议的一致性。

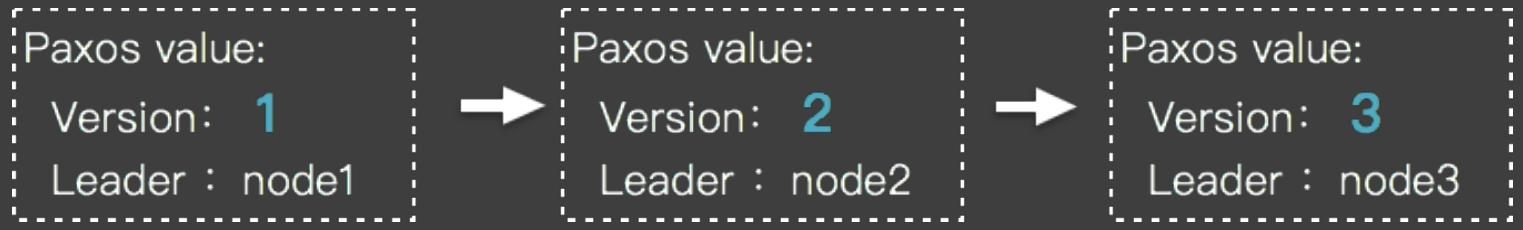
通过paxos在集群中，确定某个值



phase 1: 确定accepter超过多数派
phase 2: 将确定的paxos value, 发送到各个accepter
phase 3: 将paxos value提交给所有的Node, 保存到本地

成员列表、leader信息一段时间都会进行变更，
如何保证本次变更不受上次变更的影响?

Paxos value多版本

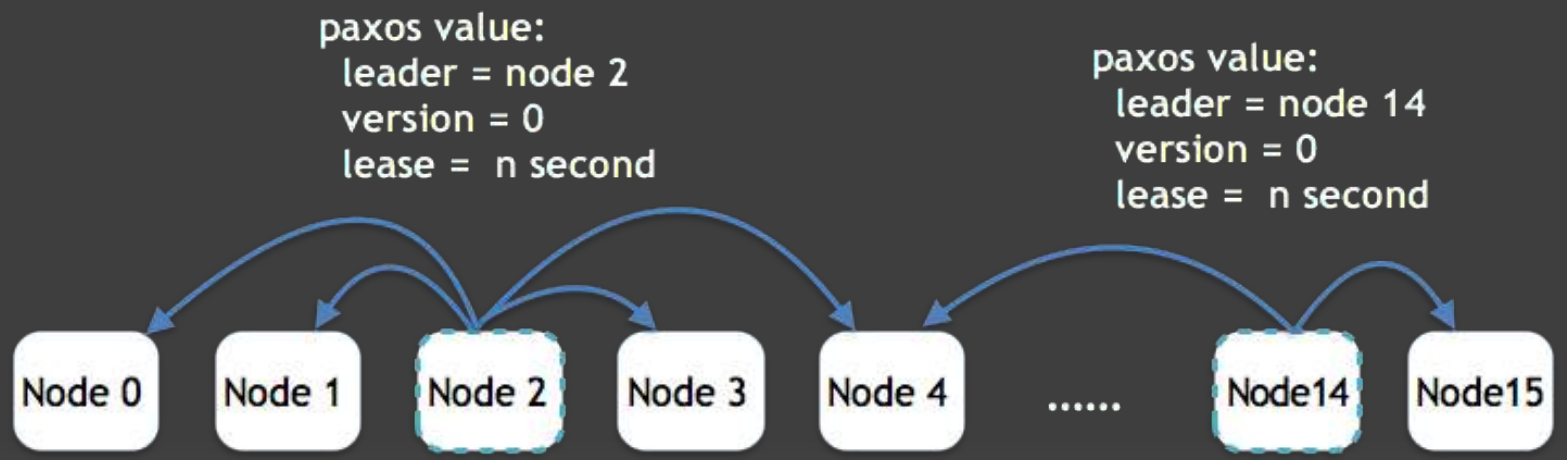


多版本的概念，
每次运行paxos只确定一个版本的值，当该版本的值提交之后.
下次value修改，运行paxos，确定的是新版本的值。

白山云科技
BAISHANCLOUD

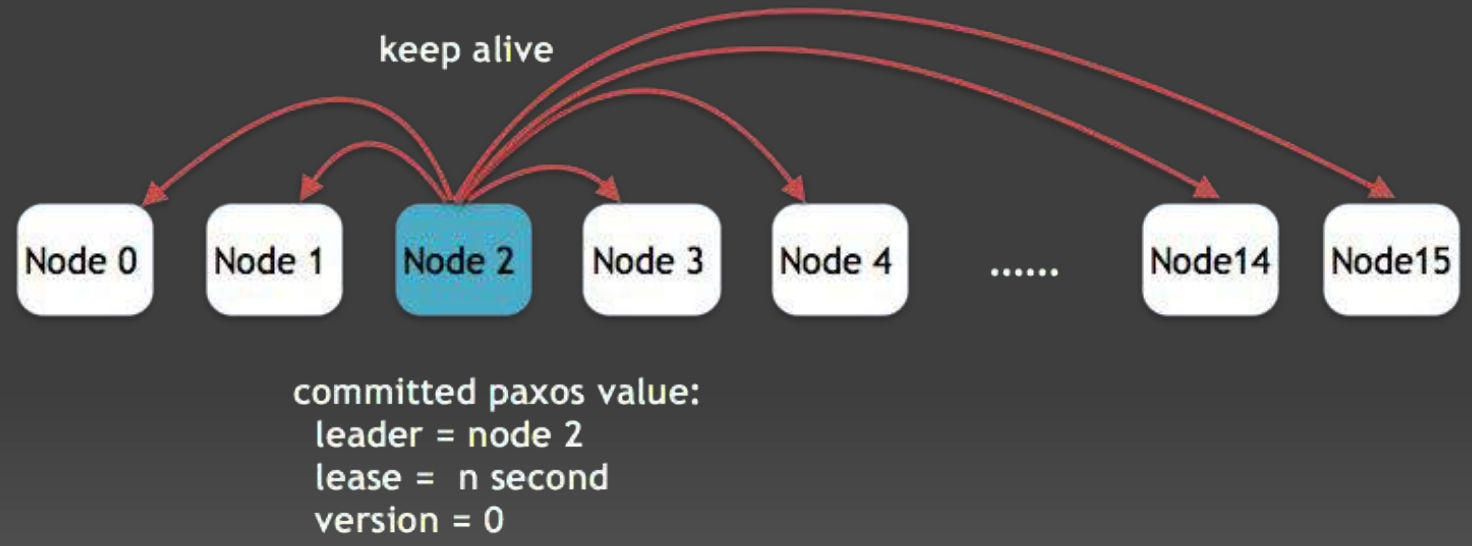
聊一个网络异常导致leader被隔离的场景

通过paxos 选择leader



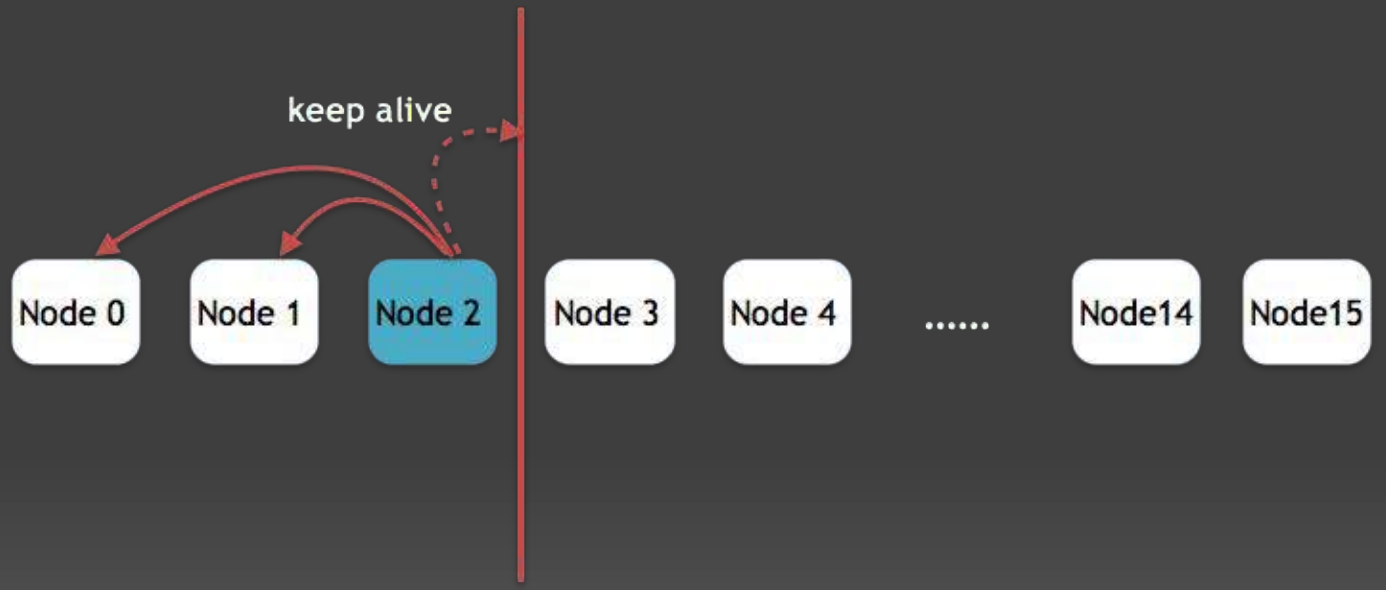
没有leader时，每台机器都选择自己当leader，
通过paxos最后确定出一个leader

Leader检查每台机器



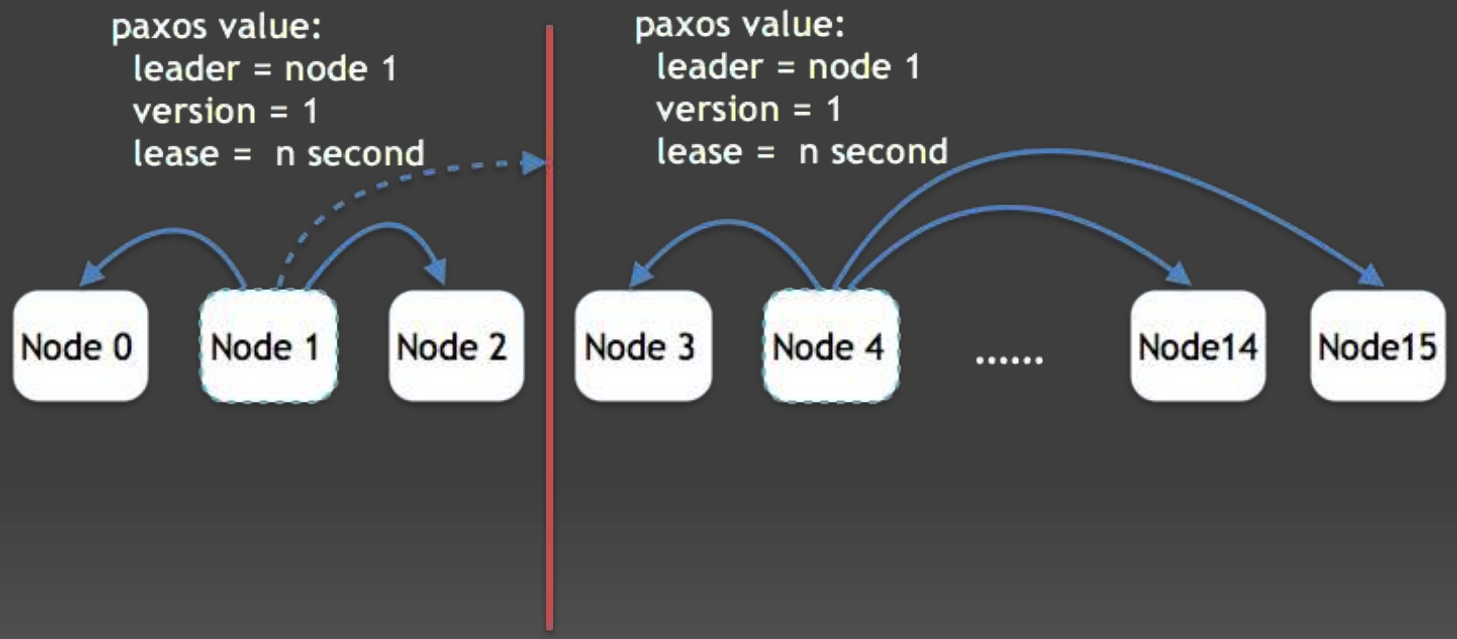
Leader负责监听，每个机器是否存活、磁盘是否有故障等

网络出现异常



网络异常 Leader联系不到 node3到node15

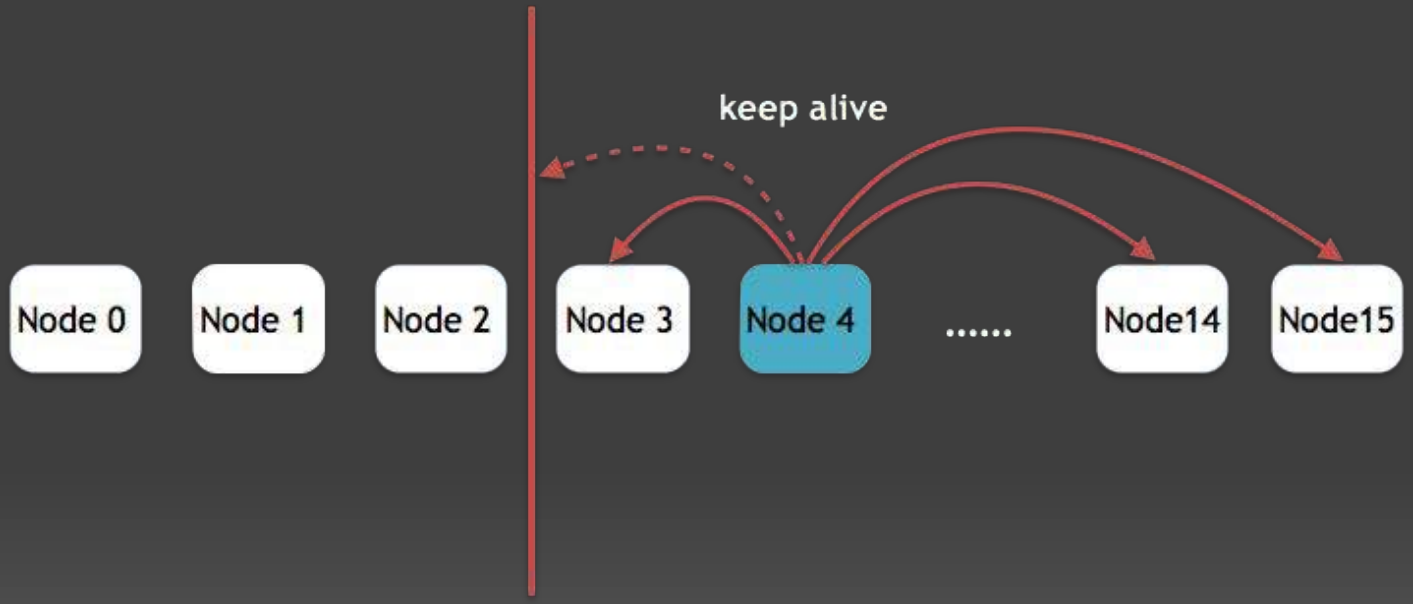
重新选主



Node 2作为leader的租约到期，大家重新选主
Node 4能联系到多数节点，成功运行paxos成为leader

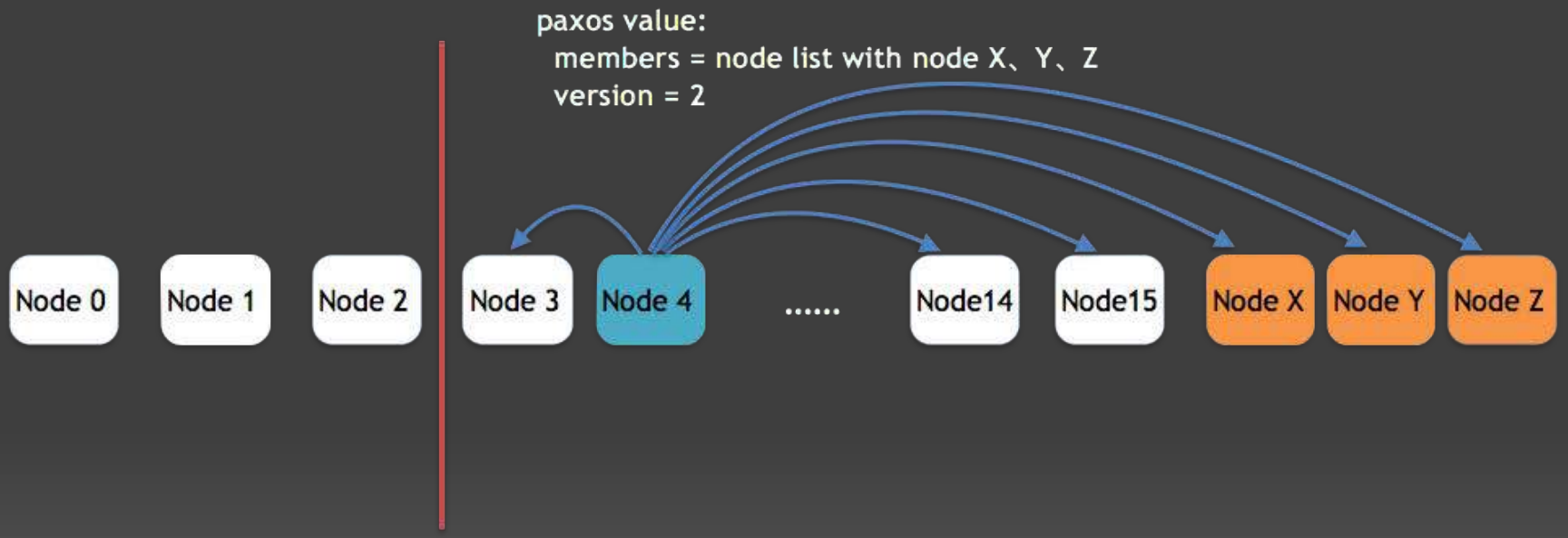
新主探测节点存活

白山云科技
BAISHANCLOUD



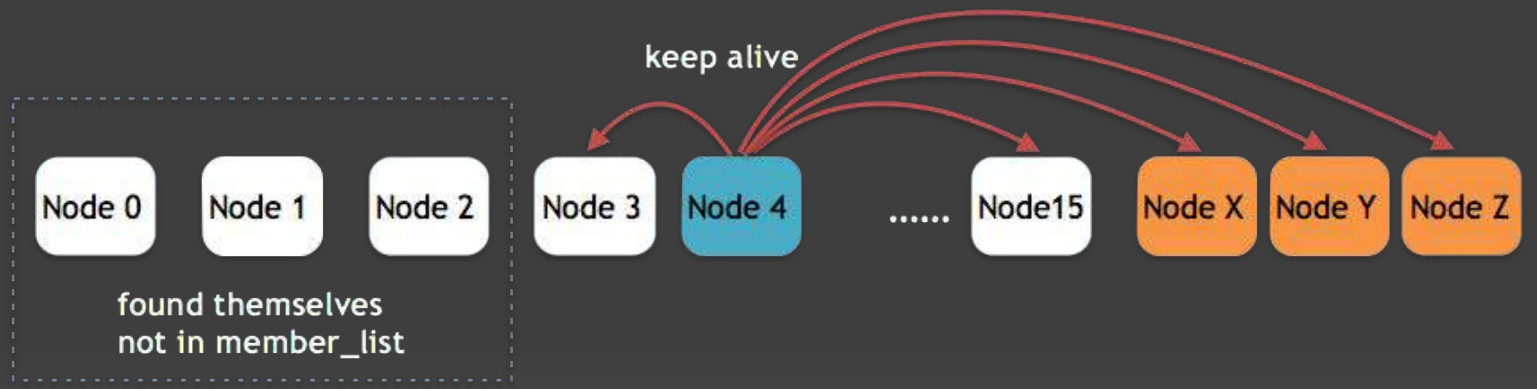
Node 4 探测每个节点的存活，
发现Node 0、1、2一段时间内无法联系到

更换新的成员



选择三个新的node替换node0、1、2
所有node的成员列表都将node0、1、2
更新为nodeX、Y、Z

网络异常又恢复了



网络恢复后 Node0、1、2同步paxos value，
发现自己已经不在EC集群的成员列表里面了，
删除相关EC的数据，进行自我毁灭。。。