

GIAC | BEIJING
Dec.12.16-17

架構
ARCHNOTES
高可用架構

技术架构未来

thegiacy.com

360 redis 生态圈

陈宗志

Introduce

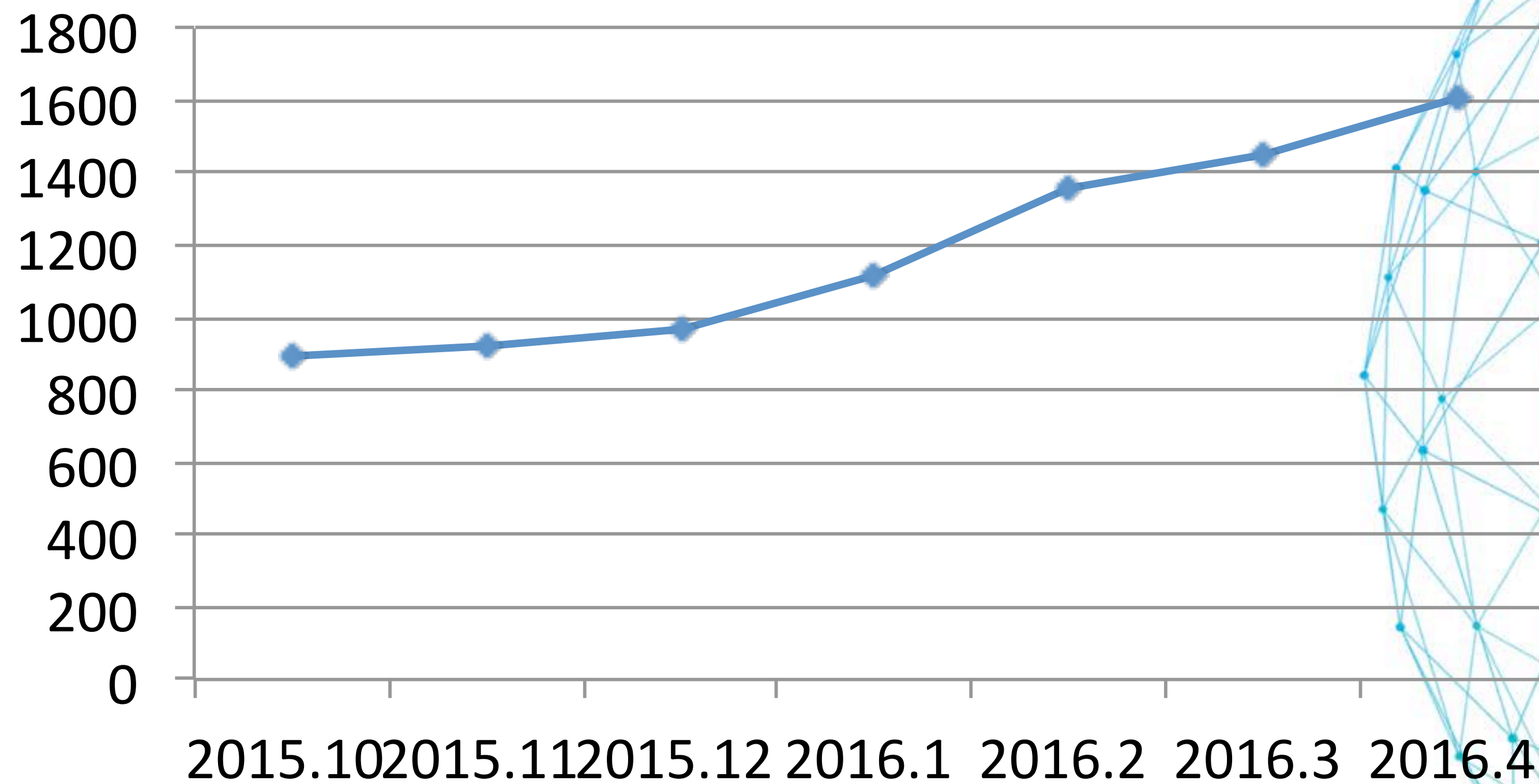
- 13年入职 360 基础架构组
 - Bada
 - Pika
 - Zeppelin
 - Mario, Pink, slash, floyd
- <https://github.com/Qihoo360>

简介

- redis
- pika
- redis-cluster

单位：亿

Redis Hit

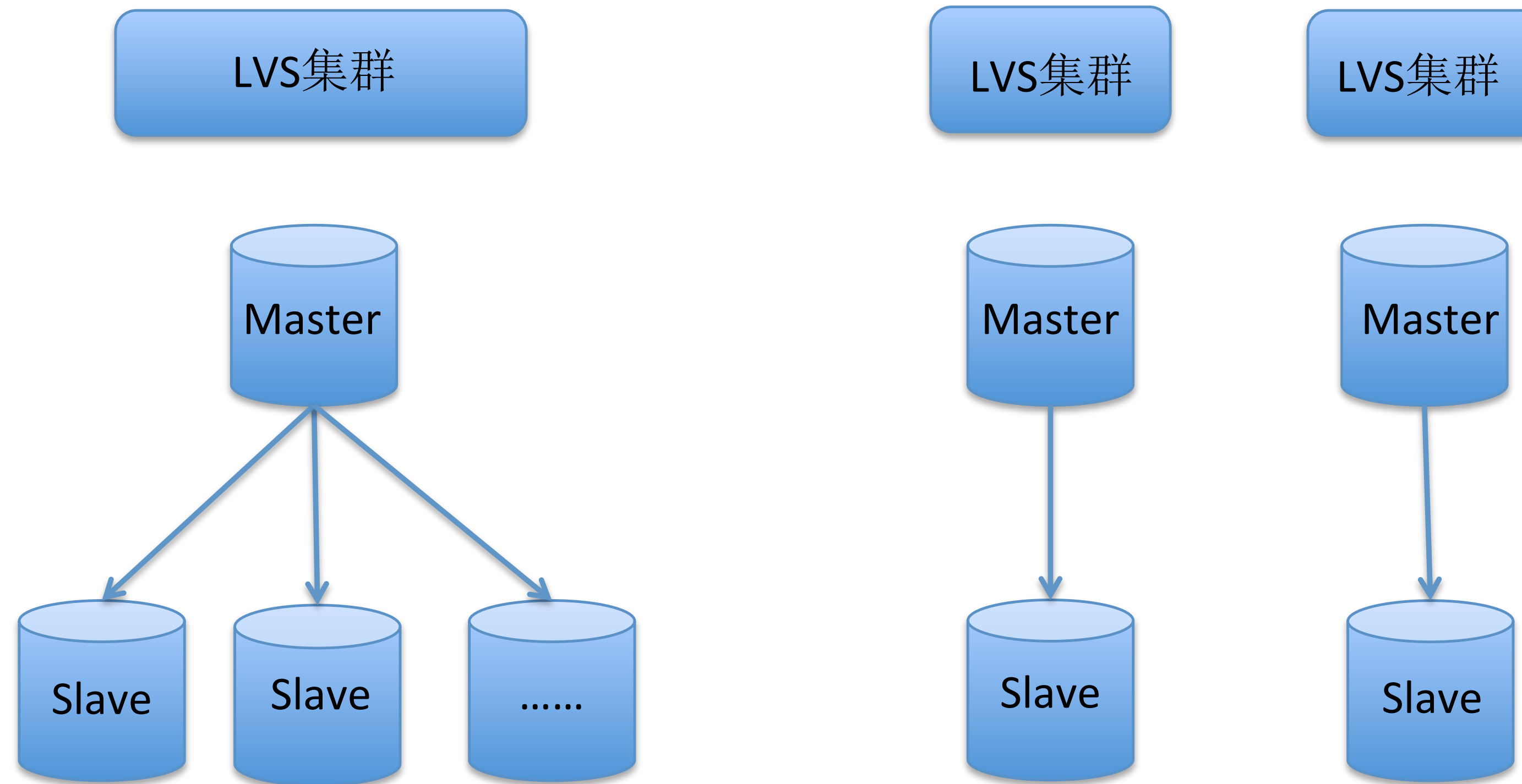


redis

redis

- 高性能读写
 - 单线程读写 qps 10w/s
- 多数据结构
 - 支持string, hash, list, set, zset, HyperLogLog, Geo 等等
- 持久化
 - RDB, AOF
- 高可用
 - Master-slave(sentinel)
- 事务支持
- 自动过期

redis



redis

- 场景
 - 高性能
 - 小数据量
 - 缓存, 持久化

redis

- 业务端
 - 读写分离
 - hash 分片

Pika

pika

- pika 是 DBA 和 Bada 团队一起设计开发的大容量 Redis 的解决方案
- 完全兼容 redis 协议, 用户不需要修改任何代码进行迁移

Pika

- 大容量redis 问题
- Pika 架构
- Pika 实现

Pika

- 恢复时间长
 - 50G redis 恢复时间70分钟
 - 同时开启aof 和 rdb

Pika

- 一主多从, 主从切换代价大
- 主库挂掉后升级从库, 所有的从库全部重传数据

Pika

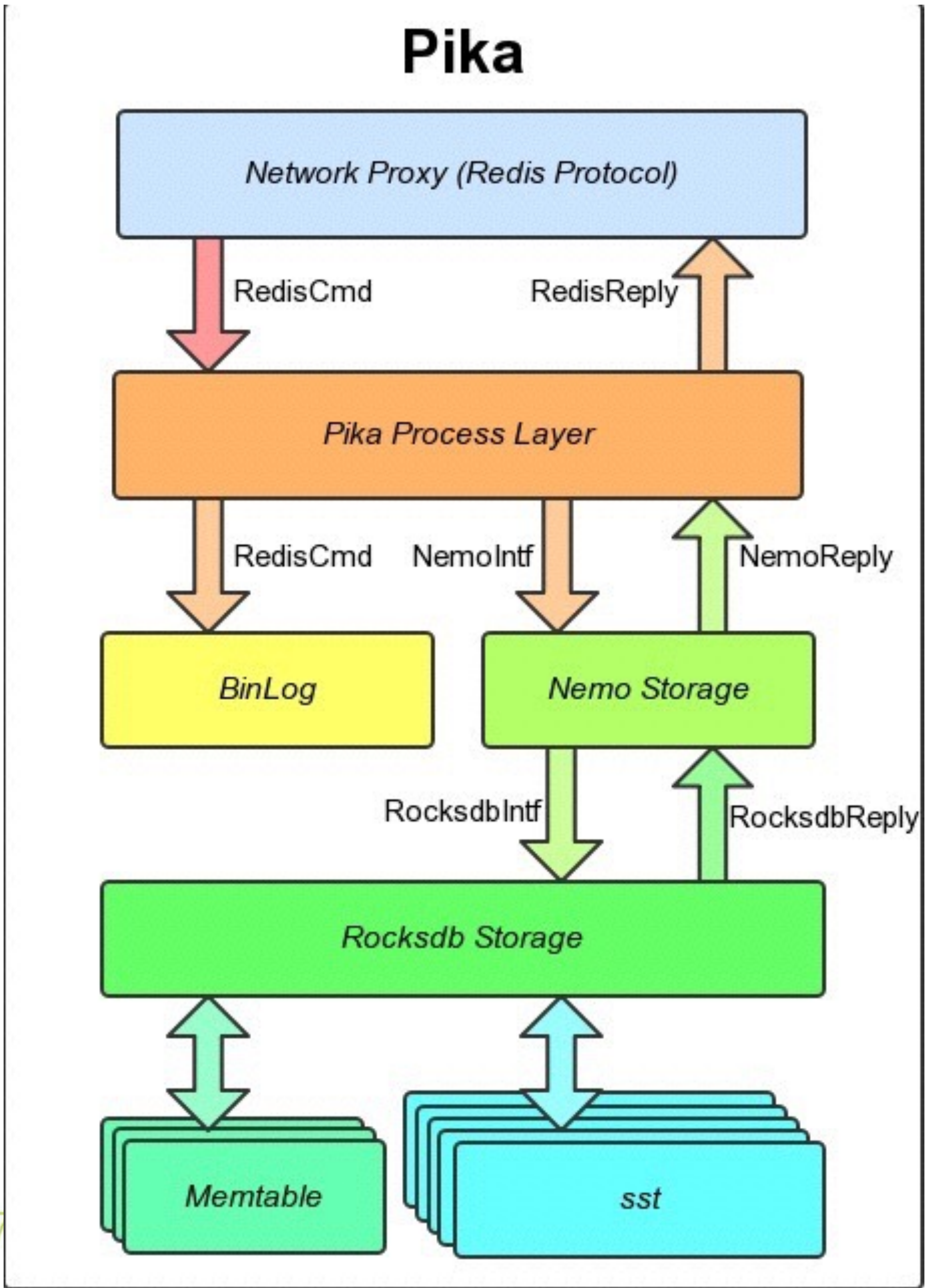
- 缓冲区写满问题
 - 内存是昂贵资源, 缓冲区一般设置2G
 - 网络原因很容易将数据堵死, 那么就会发生大量数据重传

Pika

- 内存价格昂贵
 - 我们一般线上使用的 redis 机器是 64G, 96G. 我们只会使用80% 的空间.
 - 如果一个 redis 的实例是50G, 那么基本一台机器只能运行一个 redis 实例. 因此特别的浪费资源

Pika 架构

Pika



Pika

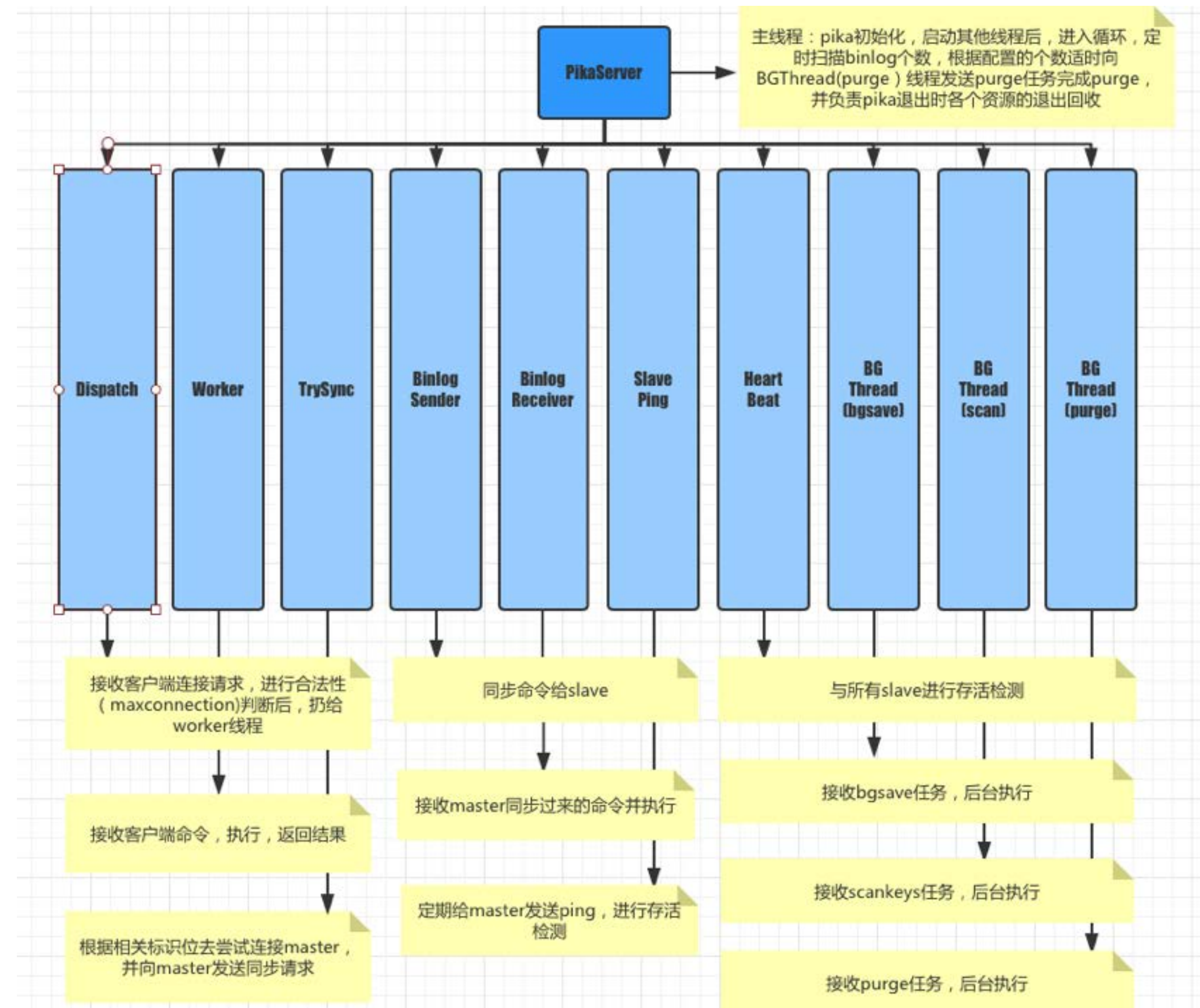
- pink
 - 基础架构团队开发网络编程库, 支持pb, redis等协议. 对网络编程的封装, 用户实现一个高性能的server 只需要实现对应的DealMessage() 函数即可
 - 抽象各种不同类型线程
 - DispatchThread
 - WorkThread
 - BGThread

Pink

```
class MyPbConn : public pink::PbConn {
public:
    MyPbConn(int fd, std::string ip_port, pink::Thread* self_thread_ptr = NULL) : pink::PbConn(fd, ip_port) {
        res_ = dynamic_cast<google::protobuf::Message*>(&message_);
    }
    ~MyPbConn() {}
    int DealMessage() {
        message_.ParseFromArray(rbuf_ + cur_pos_ - header_len_, header_len_);
        message_.set_name("hello " + message_.name());
        uint32_t u = htonl( message_.ByteSize());
        memcpy(static_cast<void*>(wbuf_), static_cast<void*>(&u), COMMAND_HEADER_LENGTH);
        message_.SerializeToArray(wbuf_ + COMMAND_HEADER_LENGTH, PB_MAX_MESSAGE);
        set_is_reply(true);
    }
}
```

Pink

```
class MyWorkerThread : public pink::WorkerThread<MyPbConn> {  
public:  
    MyWorkerThread(int cron_interval = 0) : pink::WorkerThread<MyPbConn>(cron_interval) {}  
    ~MyWorkerThread() {}  
private:  
    MyWorkerThread& operator=(MyWorkerThread&);  
    MyWorkerThread(MyWorkerThread&);  
};  
  
MyDispatchThread *dispatch_thread = new MyDispatchThread(my_port, my_worker_threads_num,  
my_worker_threads_ptr, dispatch_cron_interval);
```



Pika

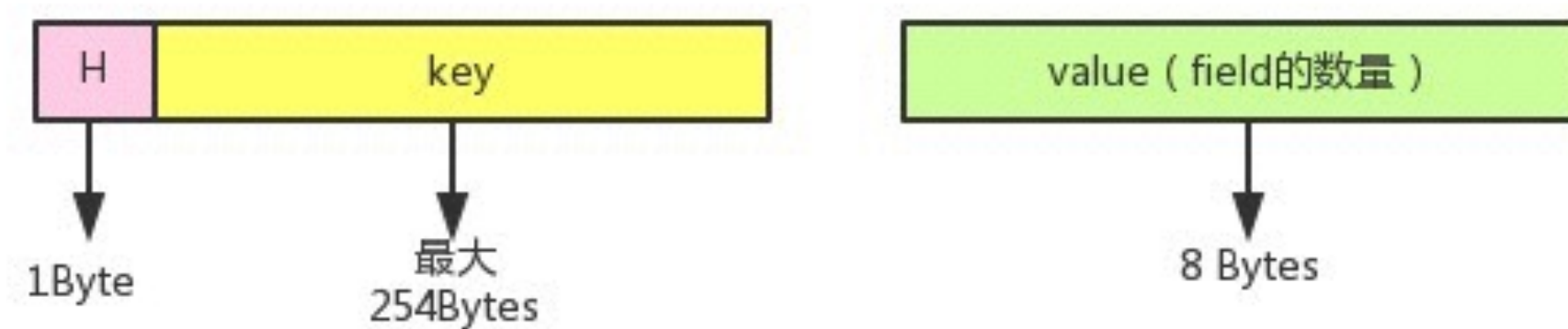
- nemo
 - pika 的存储引擎, 基于 rocksdb 实现. 实现了 hash, list, set 等数据结构
 - rocksdb 启动只需要加载 log 文件
 - rocksdb 大量使用的本地磁盘
 - <https://github.com/Qihoo360/nemo>

Hash

a. 每个hash键、field、value到落盘kv的映射转换



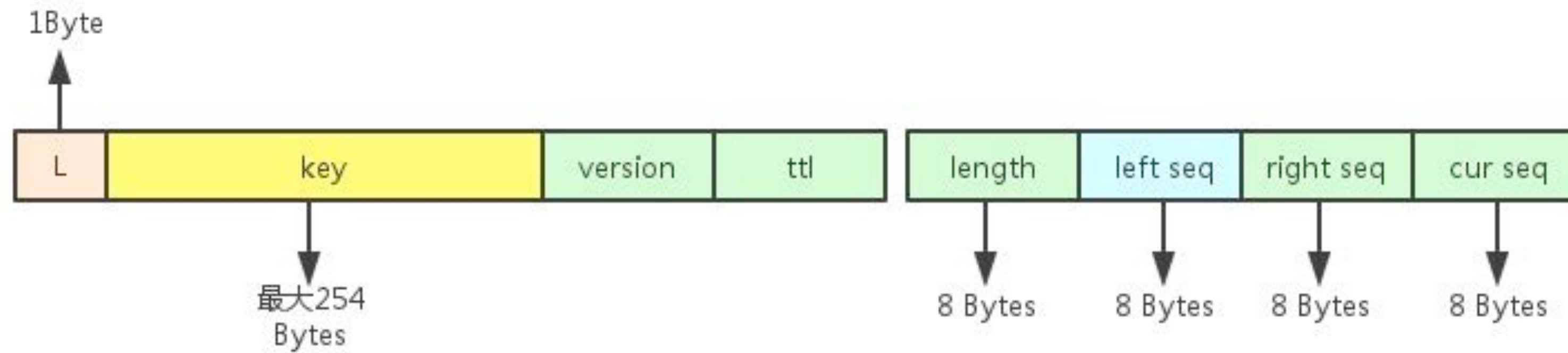
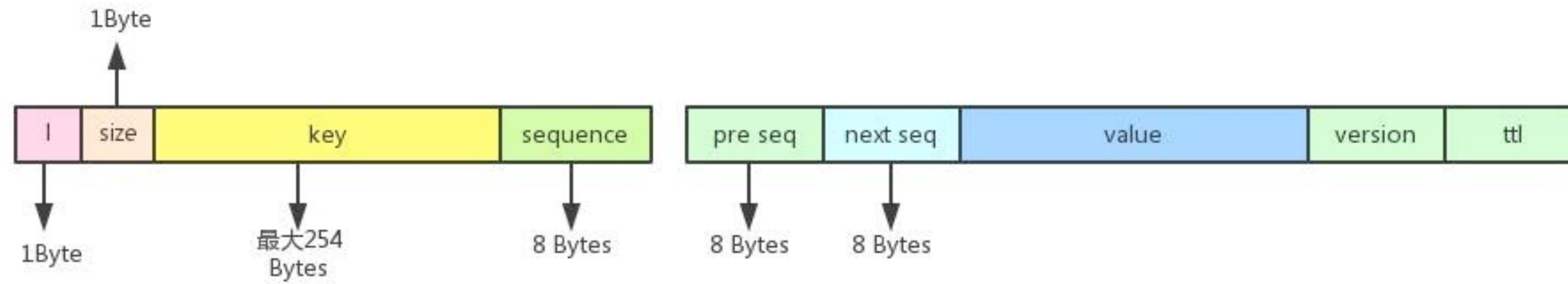
b. 每个hash键的元信息的落盘kv的存储格式



Hash

- HSET myhash field1 "Hello"
- DB->Put(wop, h6myhashfield1, Hello01477671118)
- DB->Put(wop, Hmyhash11477671118, 6)

List



List

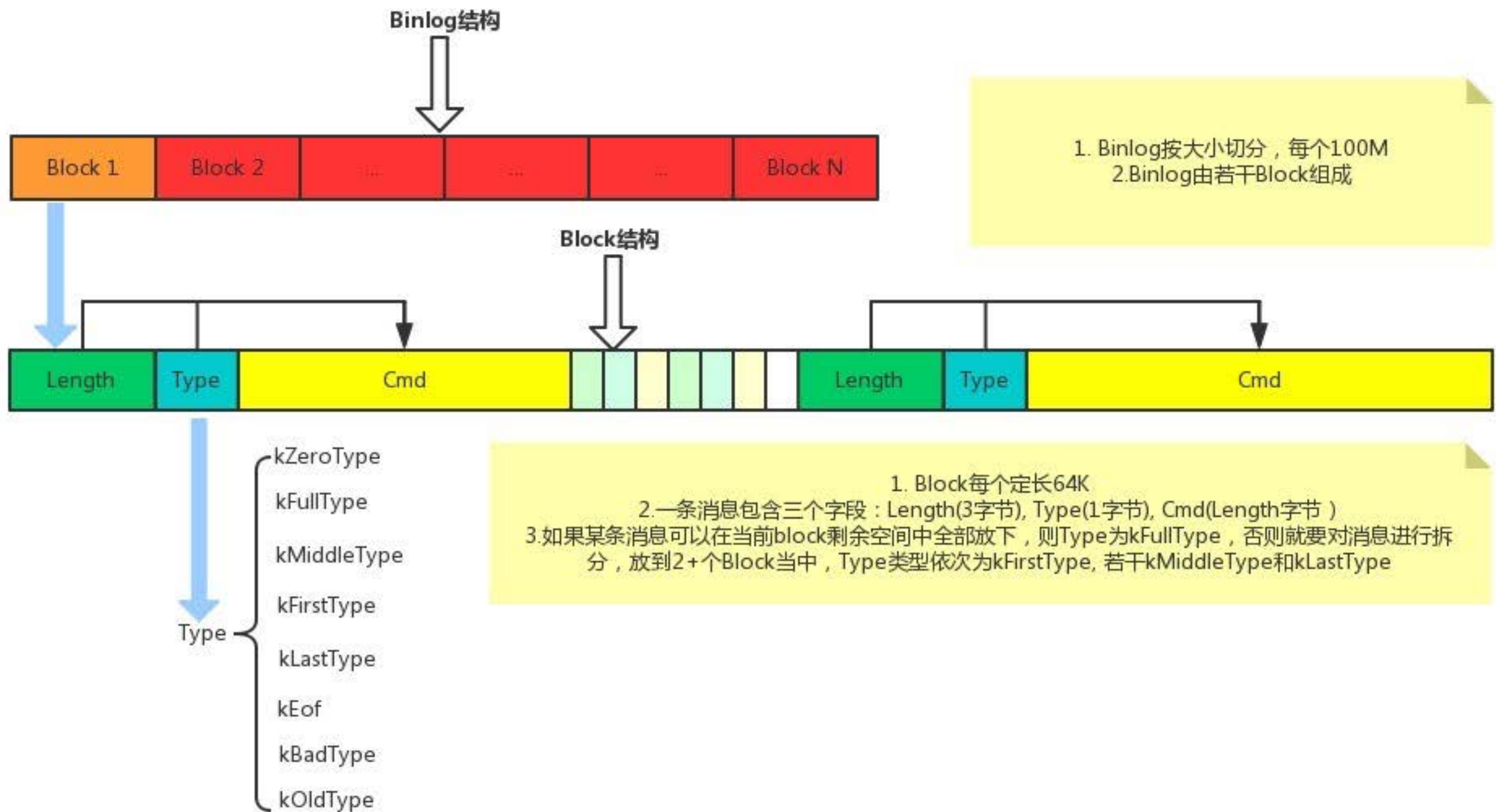
- LPUSH mylist "world"
- DB->Put(wop, l6mylist6, 57world01477671118)
- DB->Put(wop, Lmyhash11477671118, 6071)

Pika

- 主从结构
- slaveof
- binlog

Binlog

- 顺序写文件, 通过Index + offset 进行同步点检查
- 解决了缓冲区小的问题
- 支持全同步 + 增量同步



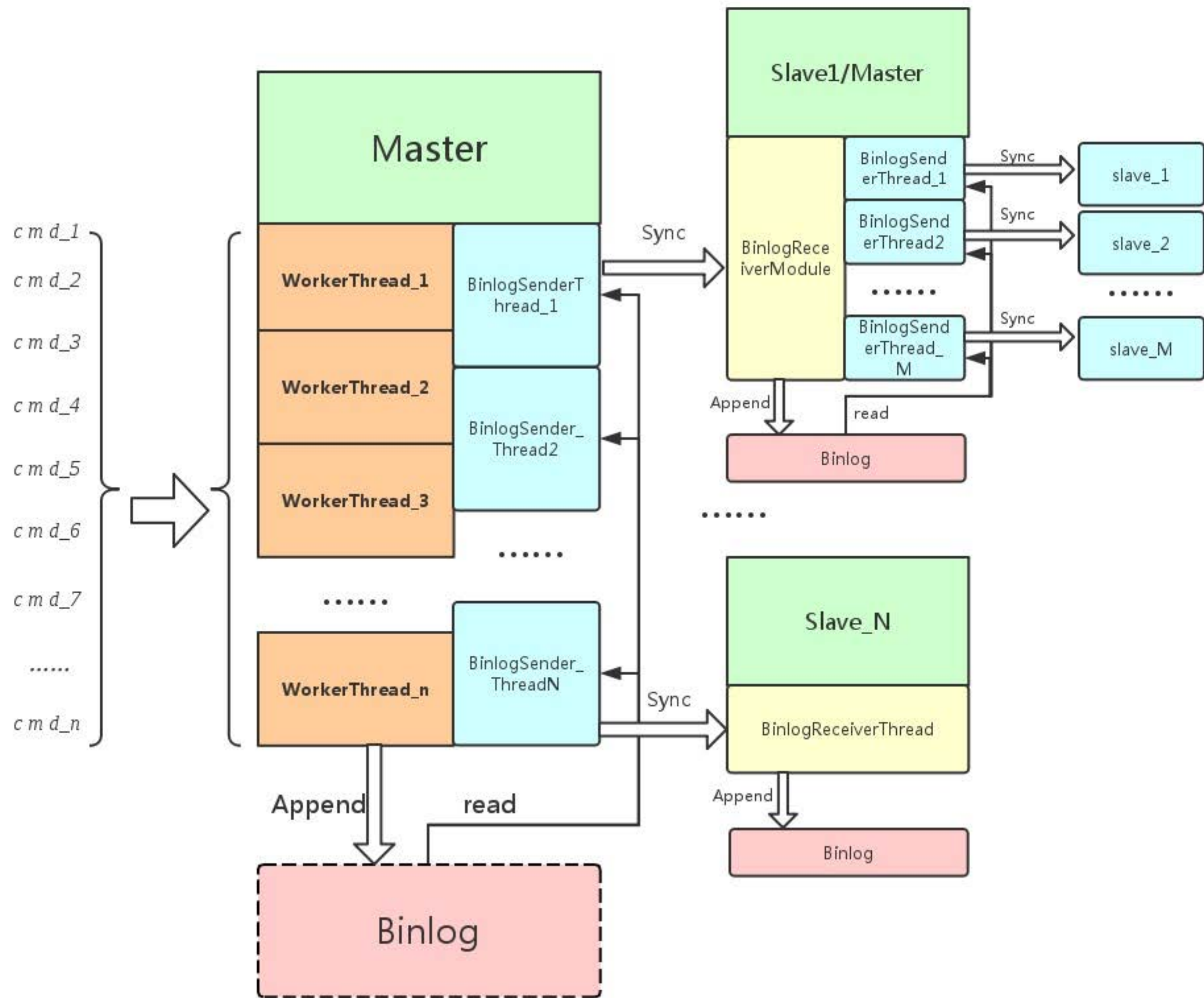
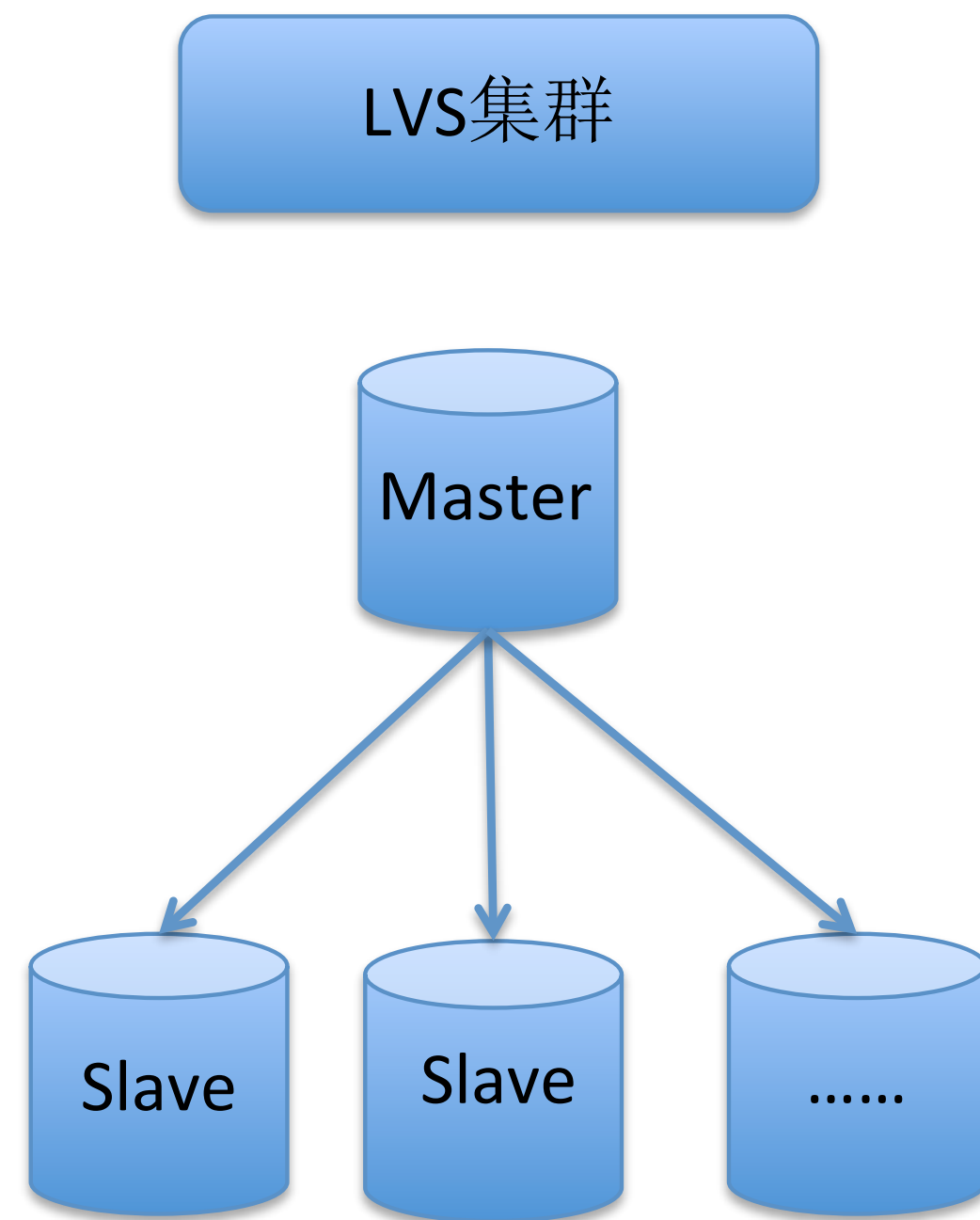


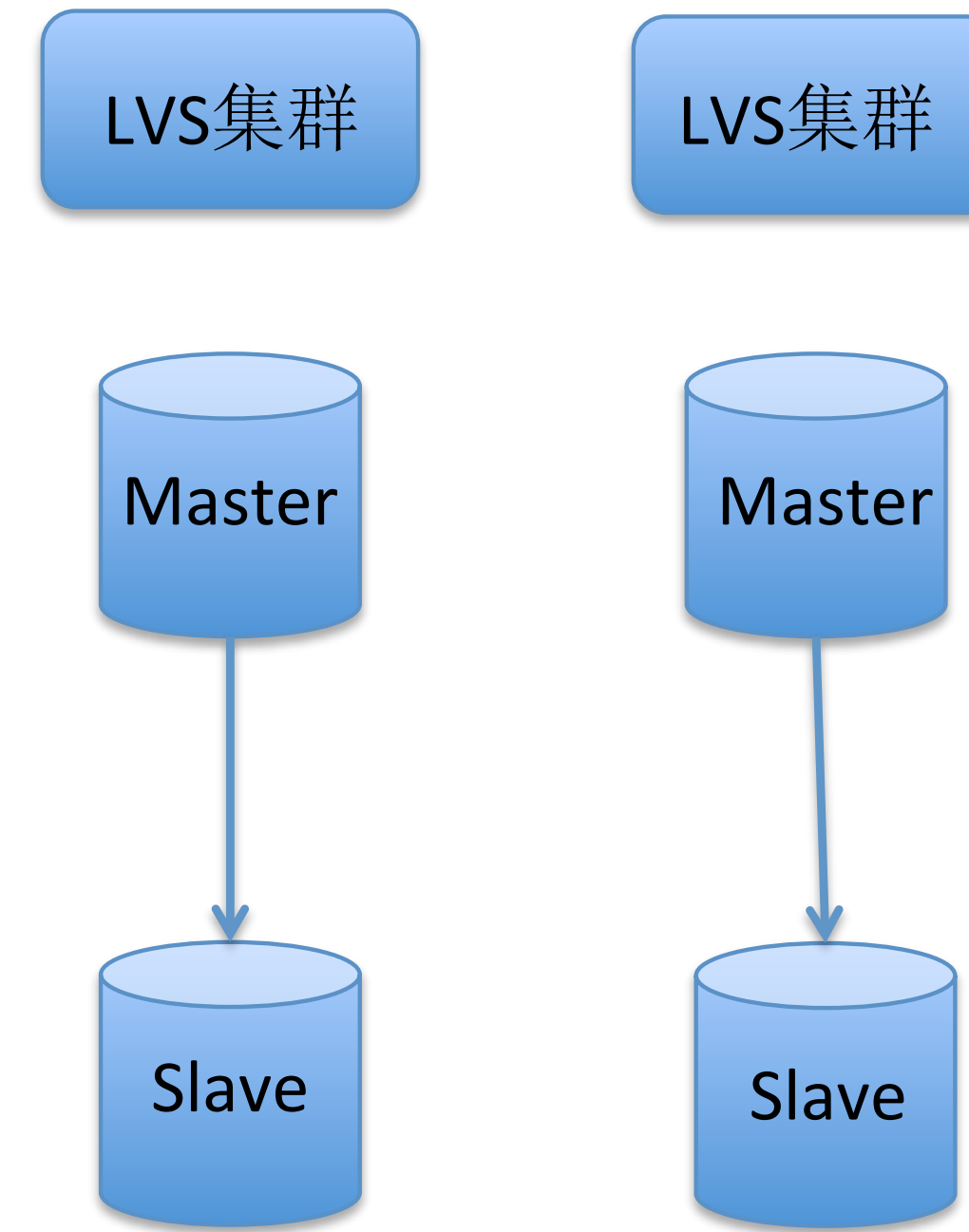
图1 Pika的主从命令同步框架图

Pika

一主多从



多机房自治



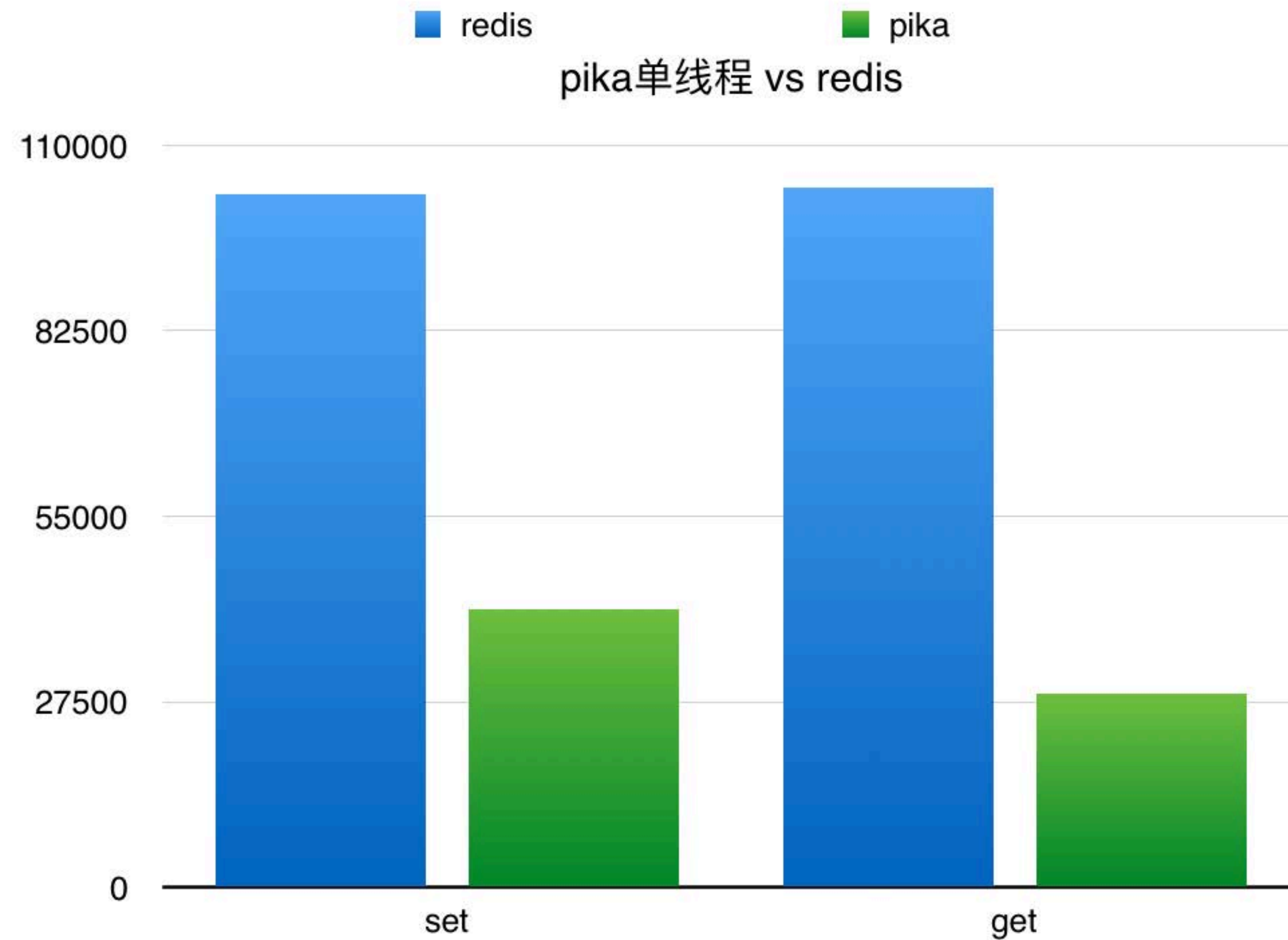
Pika

- 高性能
 - 多线程结构, 单节点重复利用多核. 单进程8w qps
- 低成本
 - 持久化存储数据在磁盘上, 基于Rocksdb 引擎, 性价比高
- 支持全同步, 半同步
 - 对网络容忍度更高
- 启动速度快

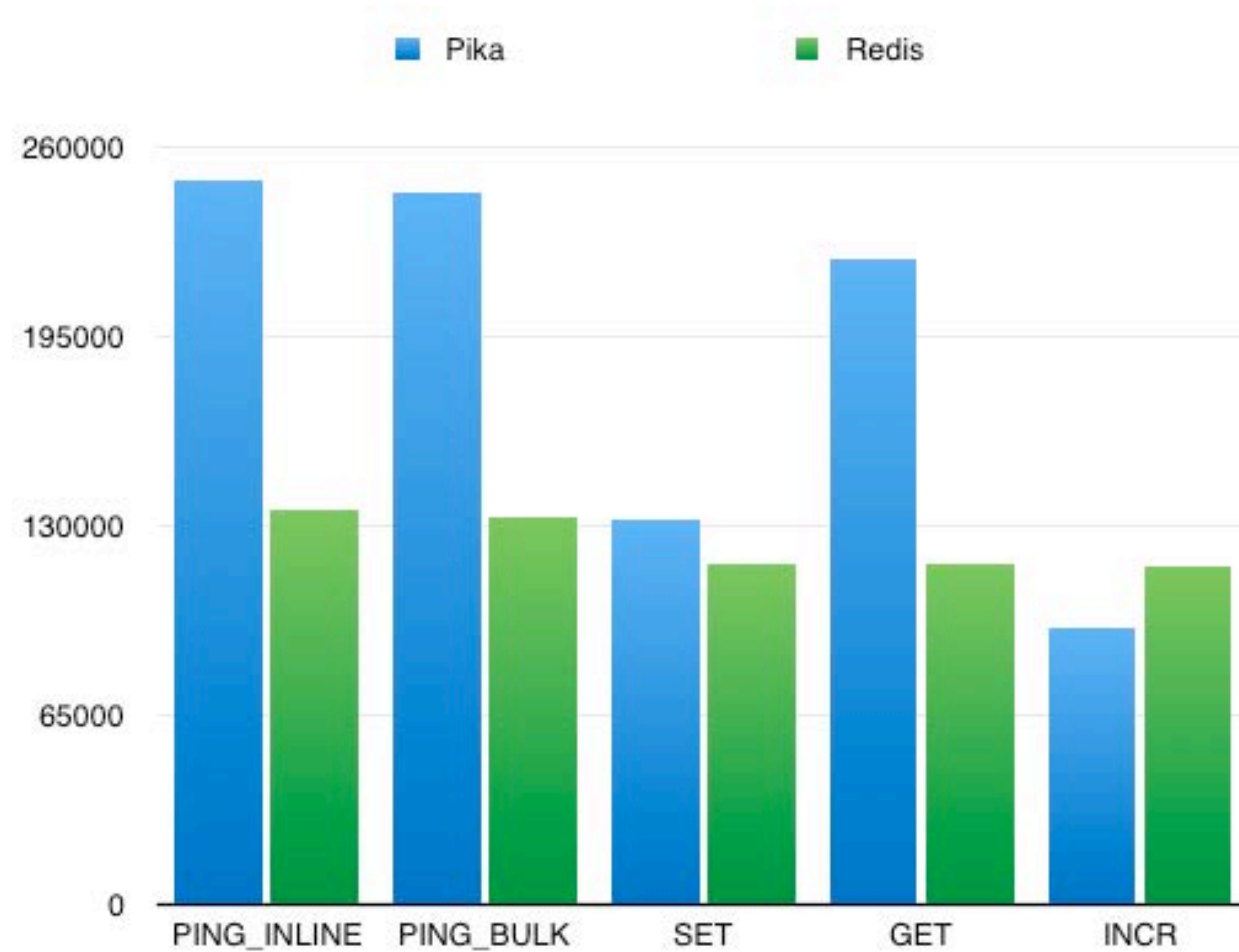
Pika

- 场景
 - 多数据结构
 - 持久化数据存储
 - 大容量

pika vs redis



pika vs redis



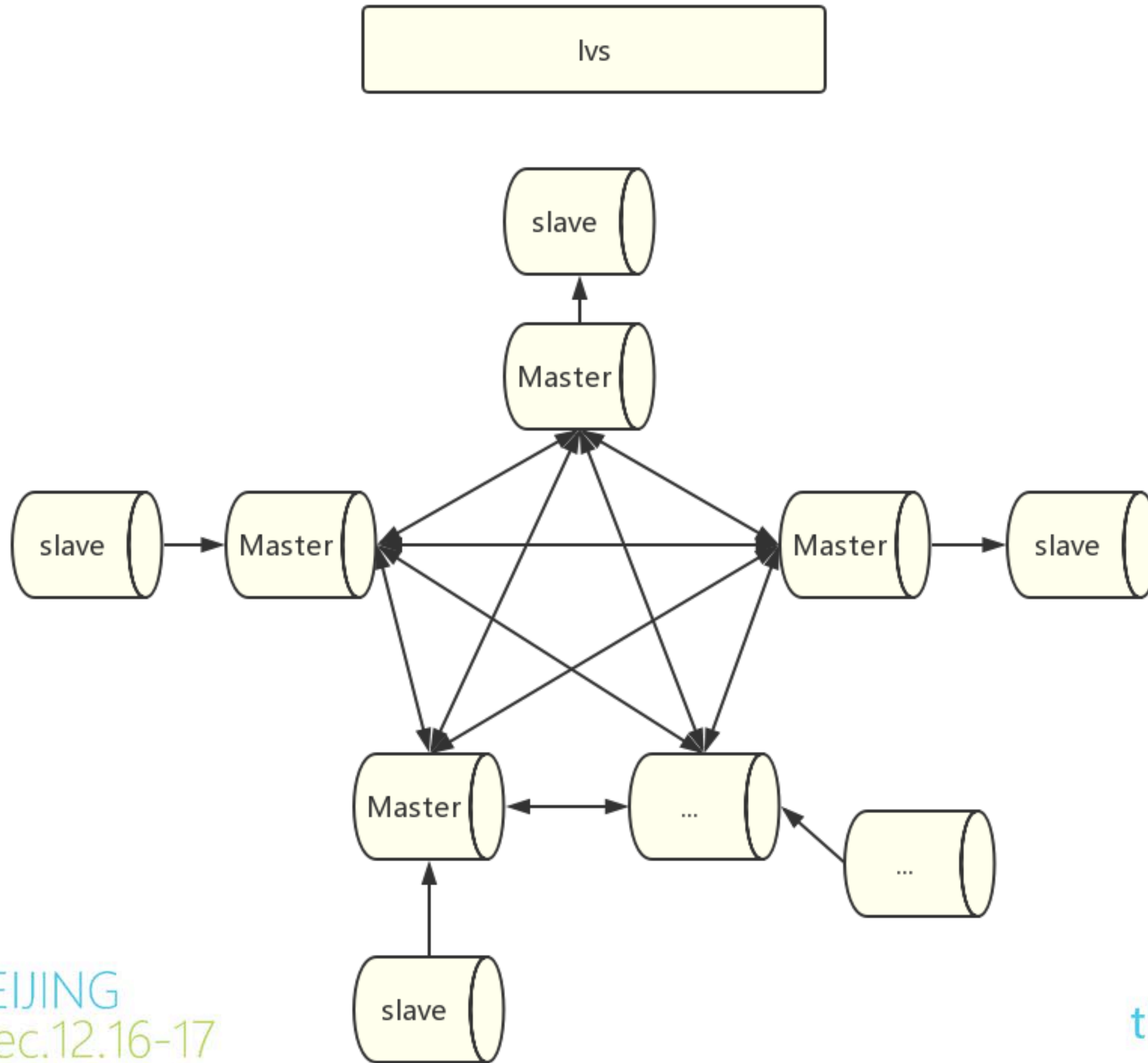
wiki

- <https://github.com/Qihoo360/pika/wiki>

redis-cluster

redis-cluster

- 无中心节点
- 数据分布 Hash slot
- 主从副本 最终一致性
- failover gossip



Hash slot

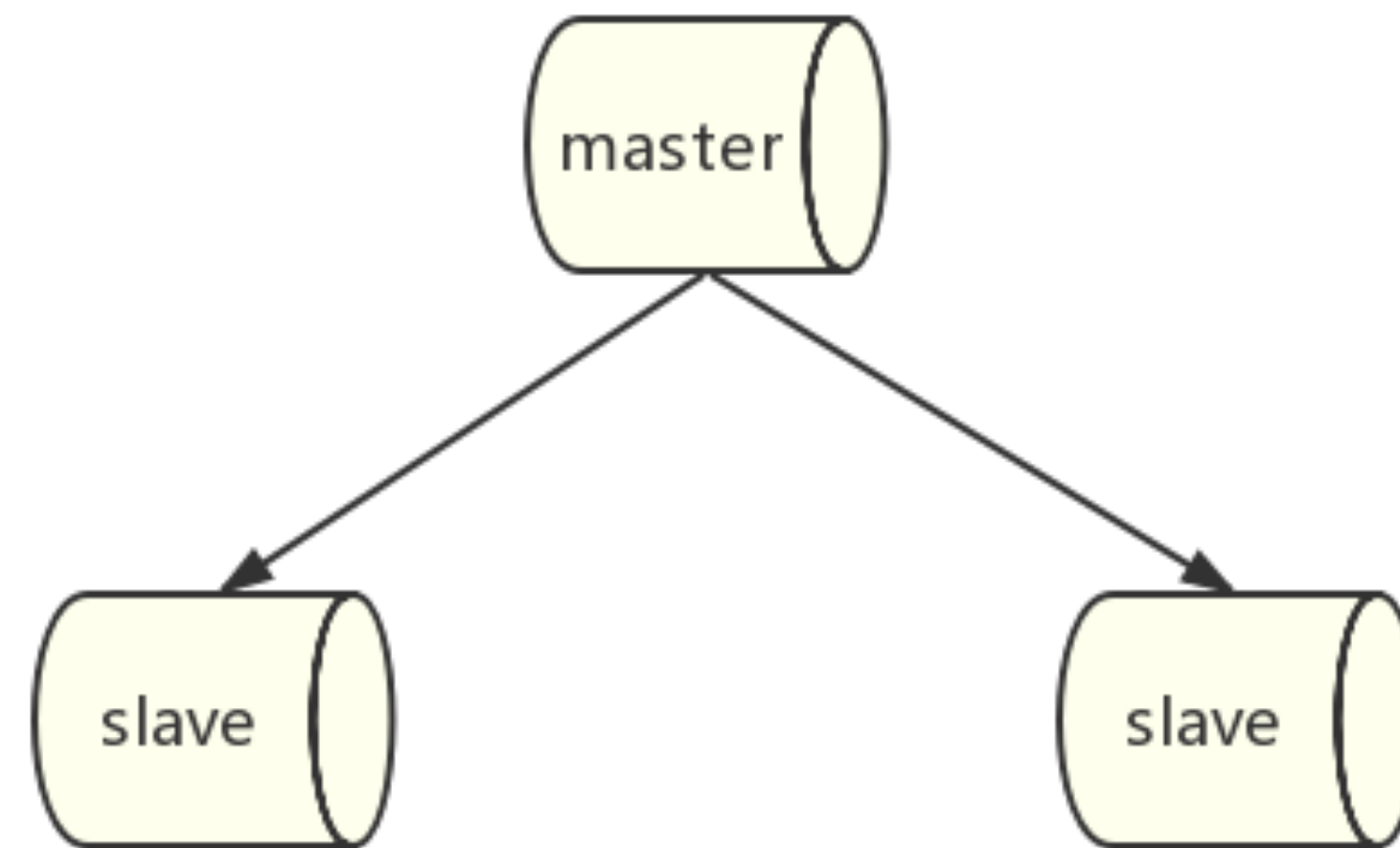
- Node A contains hash slots from 0 to 5500.
- Node B contains hash slots from 5501 to 11000.
- Node C contains hash slots from 11001 to 16383.

Hash-slot

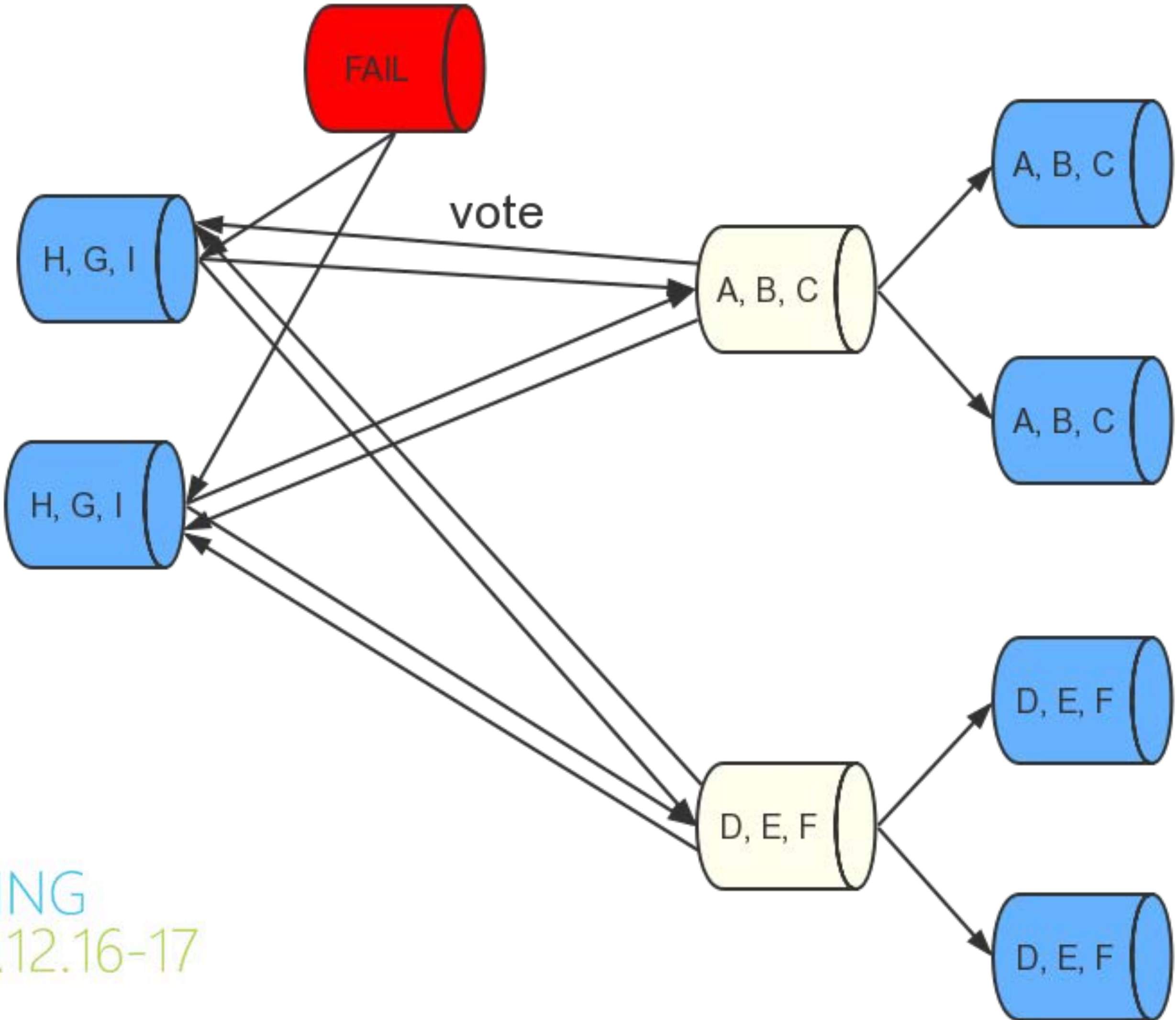
```
cluster nodes
$6865
c4a4f66771d2ed0a10960963c28988b7194e51e0 16999 master - 0 1477560407533 74 connected 2048-2347
949eddf054b7364c95bfd2f79f64aee5f01c821d 16999 master - 0 1477560417533 63 connected 1000-1341 3070-3071 14336-14675
f253d5f3ae0ccc735cab424a343ca2bf578f5d9a 16999 master - 0 1477560410534 61 connected 3072-3413 15360-15701
29021cb07c9508747cb4e22068ffc187258ebcac :16999 master - 0 1477560435534 41 connected 14676-15357
6e9aaff11ef70d483a362acb665a2ea5e8bbf9c9 :16999 master - 0 1477560392527 39 connected 8532-9213
4b780db69f56f366a924f67ee2da0cec12c42601 :16999 slave 091614c3461f58c32c40bade4251acceff4b5e24d 0 1477560402532 20 connected
2a3e0197f2cf923b274dab7109411db2eb64c54a :16999 slave 43a68252e445fd630831c035d07236e3cbee9d43 0 1477560425534 42 connected
90f7c96ded0733d8bc0237cab7928db062020d54 :16999 slave c4a4f66771d2ed0a10960963c28988b7194e51e0 0 1477560405534 74 connected
682a77c2179efe31cf42d9180adf11286c77dc08 :16999 master - 0 1477560436534 60 connected 5120-5461 11264-11605
aa8c87002f66742d20ee312b087f295da9af0c0a :16999 slave 9de4e976c82fa76f3939eb725b0bcd639d3605ce 0 1477560431534 37 connected
e2d90136f1aacf1b3c044bd5bcb674f859a67ba8 :16999 slave 5f2a19e5c8ffe5ee73a4556c5b3ef6716cd4f12c 0 1477560415533 11 connected
a28c8abf26e299ca889e4dff3d06b92598465ec8 :16999 slave eb94a47331be34481db5c6ab403be8d6d2f1508b 0 1477560421533 43 connected
5dd5958c20fda085ef004b84dccf8e36f32ecef5 16999 slave f79ea0890cc21b5bcc5f95468b34a0625319cb42 0 1477560420533 67 connected
02aa432b5909b6bf3ff1b44c4a0616f8df245e2e 16999 master - 0 1477560417534 64 connected 4394-4437 10240-10579
b7e2d6dd82837a5d6ee2122b3e5c32322fee6c5c 16999 slave a7b77c5cbbef296b4807432a251c7663cd9956ff 0 1477560418533 62 connected
2ff4a742549b489d39627b31f5245590984a8003 :16999 slave a6afef602a10e45e931a13abd21671e2a4b7079b 0 1477560396529 18 connected
2f6b6e904ff77c178e8033b14decccc64a516bad 16999 slave 7905a7ab10d14674c2d8df8a59682f6f0581c7ef 0 1477560404534 69 connected
43a68252e445fd630831c035d07236e3cbee9d43 :16999 master - 0 1477560424534 42 connected 642-999 13290-13311 15358-15359
ad28b24686f7605a3c8a1615c798ed05dff4d3a2 16999 slave 02aa432b5909b6bf3ff1b44c4a0616f8df245e2e 0 1477560402532 64 connected
6722b98702ad6398bbef3bf6f56c68d299d3406d 16999 slave c55c9482c83d462936c44277a238631ace58a8c0 0 1477560403532 73 connected
7905a7ab10d14674c2d8df8a59682f6f0581c7ef 16999 master - 0 1477560430534 69 connected 2023-2024 4096-4393
87f3a4c2f5a5b6232358e0f4000f74ce0b7ac0c9 :16999 slave 64b917470d686ec32cae3d26c73f33d03629aa11 0 1477560412533 19 connected
1979a5c341e5fb27fd3daf9aea152eeb1cd79d7f :16999 slave 85f94f62955d37e6dbe4a9a137eb8b0ce89db1 0 1477560438535 1 connected
a7b77c5cbbef296b4807432a251c7663cd9956ff 16999 master - 0 1477560441536 62 connected 7169-7510 9216-9557
091614c3461f58c32c40bade4251acceff4b5e24d :16999 master - 0 1477560443539 20 connected 13654-14335
f79ea0890cc21b5bcc5f95468b34a0625319cb42 16999 master - 0 1477560414533 67 connected 2348-2389 5118-5119 8192-8531
85f94f62955d37e6dbe4a9a137eb8b0ce89db1 :16999 master - 0 1477560440535 1 connected 3414-4095
89db9f2e08fc9a2c2734ada63dee227ff3d8266f :16999 slave 1af7ddc8b6b5eaf3e6d3160e65501246af1479d2 0 1477560418533 13 connected
969e9b323995a99db0fe573097a91b1beb49cfde :16999 master - 0 1477560427534 36 connected 1342-2022 7168
feb092298bfcc295f595f5f758600d6a3d172333 :16999 slave 969e9b323995a99db0fe573097a91b1beb49cfde 0 1477560406533 36 connected
cc6e96ef996f05c0bb3313f0f02c82a317917e10 :16999 slave 6e9aaff11ef70d483a362acb665a2ea5e8bbf9c9 0 1477560415533 39 connected
50a9aee9e8d62395033d639a8a66eed41d9adfc :16999 slave cb4e5ef2e1b1f51bb0159b72177ecd694d2b415d 0 1477560423533 44 connected
cb4e5ef2e1b1f51bb0159b72177ecd694d2b415d :16999 master - 0 1477560398529 44 connected 2390-3069 9214-9215
72cc3e59a88138b94628830ffe993a3d8592090f 16999 slave 92456c06e5feb93f33f6eefe2642770ad15c4365 0 1477560428534 66 connected
eb94a47331be34481db5c6ab403be8d6d2f1508b 16999 master - 0 1477560395529 43 connected 4438-5117 11262-11263
```



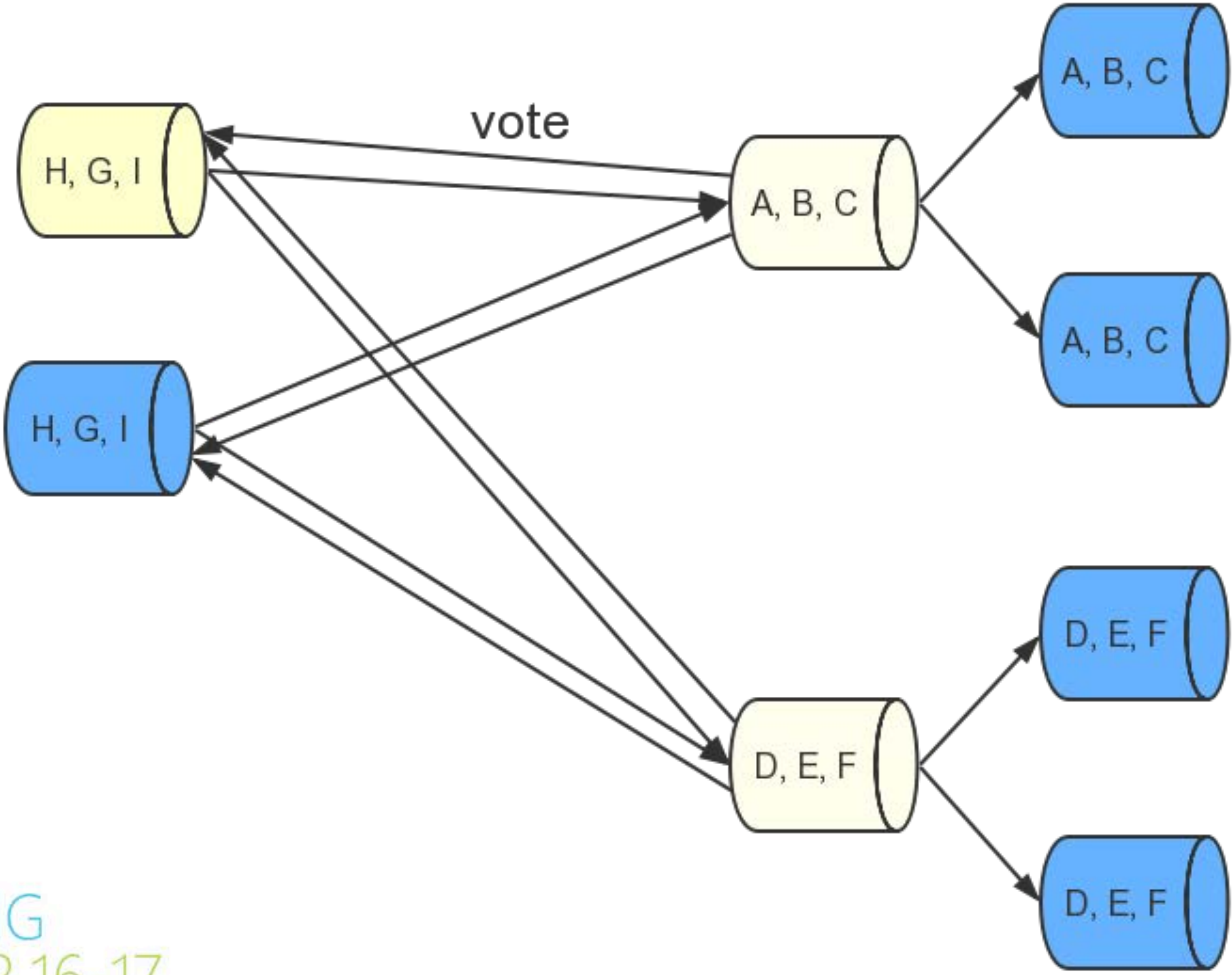
redis-cluster



failover



failover



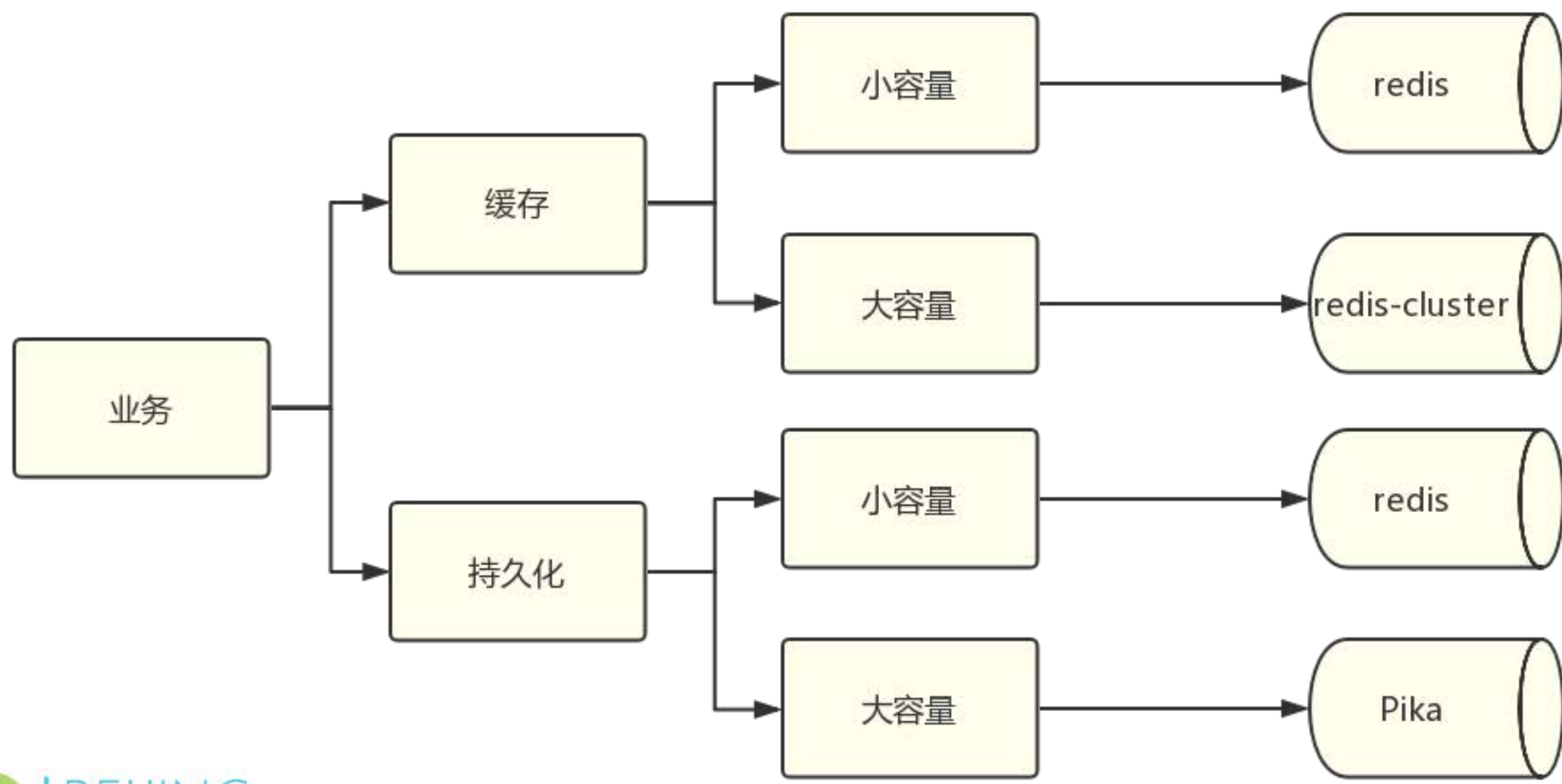
redis-cluster

- 场景
 - 纯缓存
 - 大容量
 - 高并发
 - 高增长

redis vs pika vs redis-cluster

	redis	Pika	redis-cluster
线程数	单线程	多线程	单线程
性能	10w+ qps	8w+ qps	线性增长
存储方式	缓存,持久化	持久化	纯缓存
架构	主从	分布式+HA	主从
场景	小容量, 缓存/持久化	纯缓存, 高性能	大容量, 持久化

summary





GIAC | BEIJING
Dec.12.16-17

技术架构未来

架構
ARCHNOTES
高 可 用 架 构



Thank you!

