DEFANG
TECHNOLOGY
德方智链

Fabric工作机制及数据存储剖析

德方智链 袁运亮 2018.8.11

- **个人简介**

德方智链 技术总监

曾任职于美的金融，阿里巴巴从事技术与架构的工作，短暂做过两个学期的高校老师。

主要研究领域：区块链技术与应用，分布式技术及架构，微服务，安全和密码学等。

大纲

01

Fabric架构

02

Fabric交易流程

03

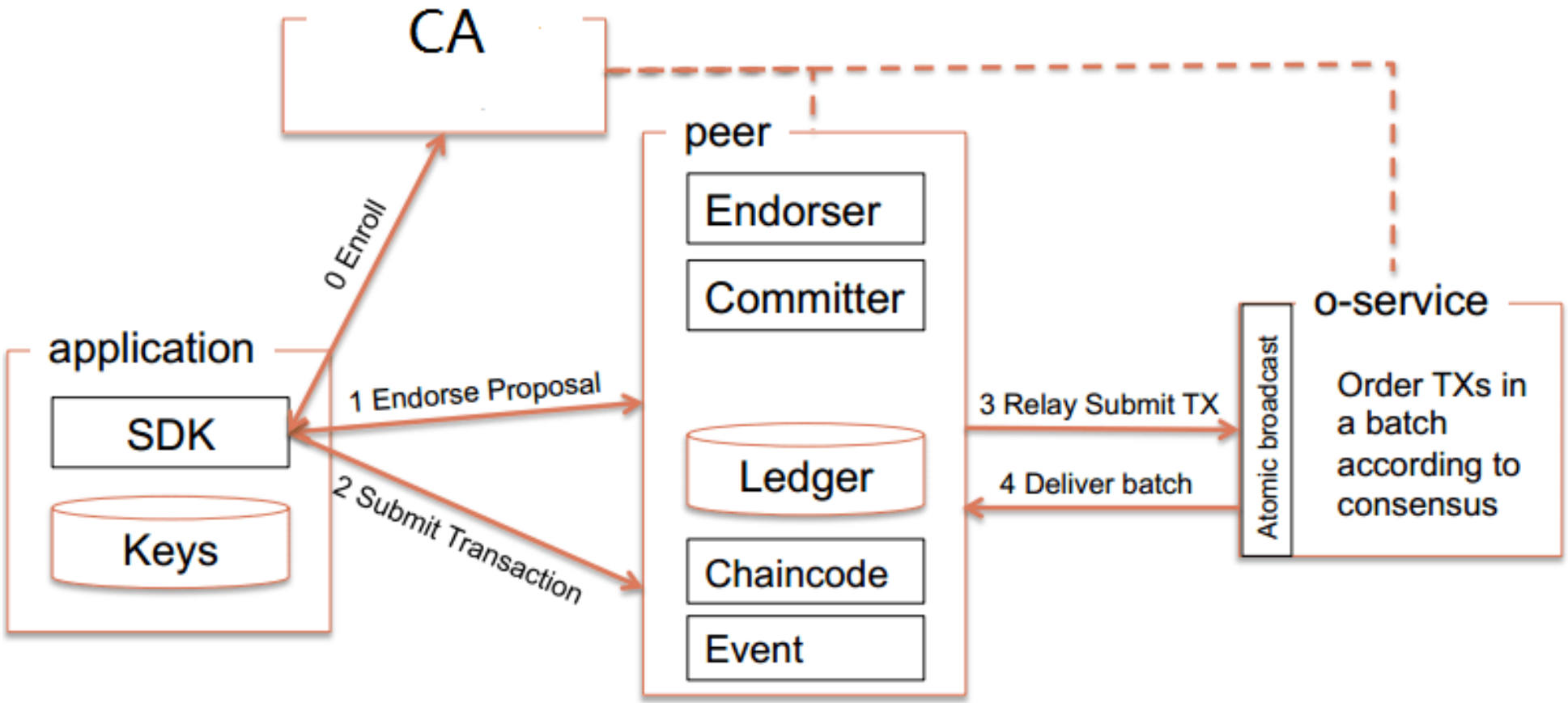
链式结构/区块结构/交易结构

04

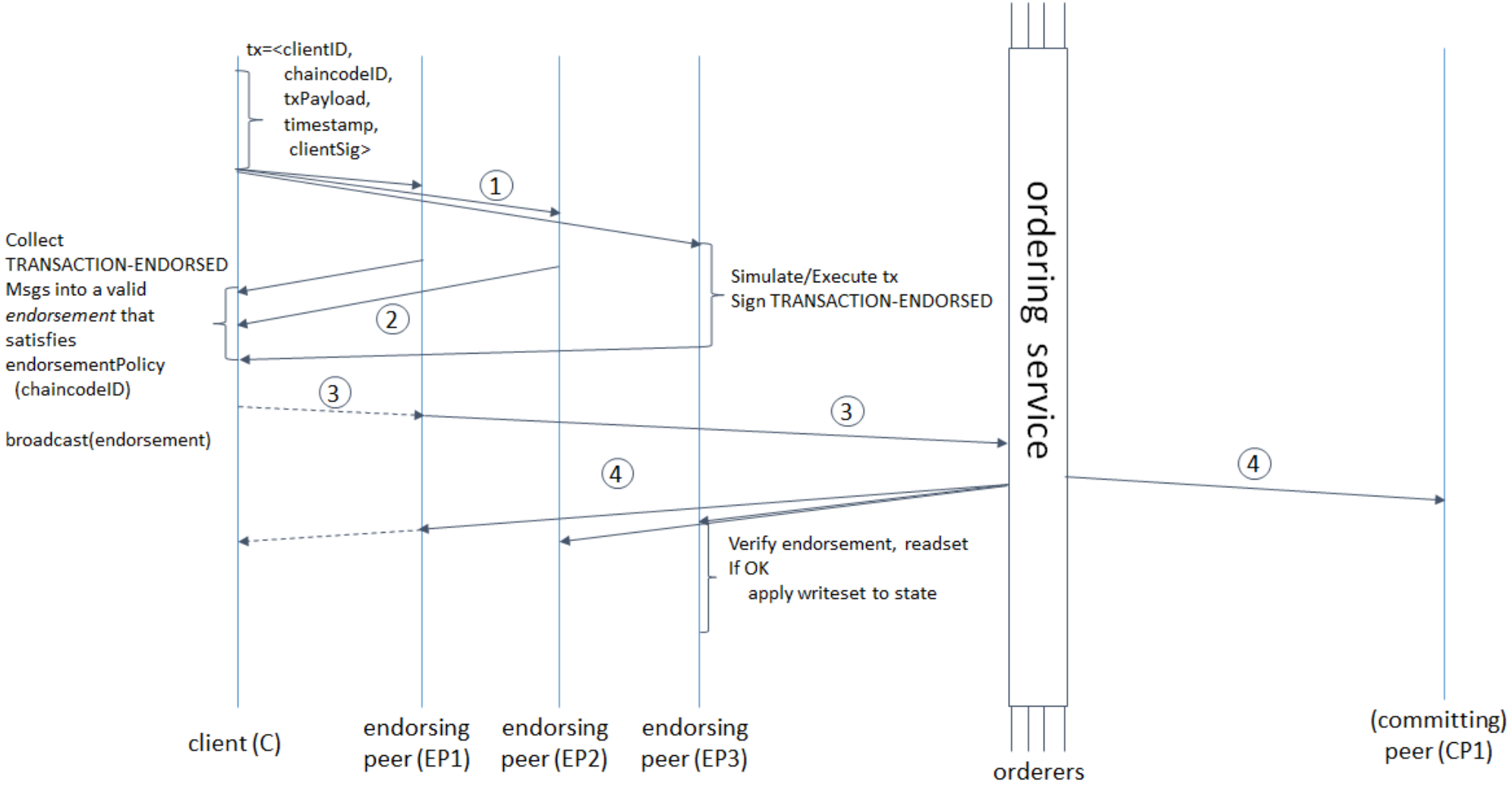
数据存储

1) Blockfile
2) stateDB
3) indexDB

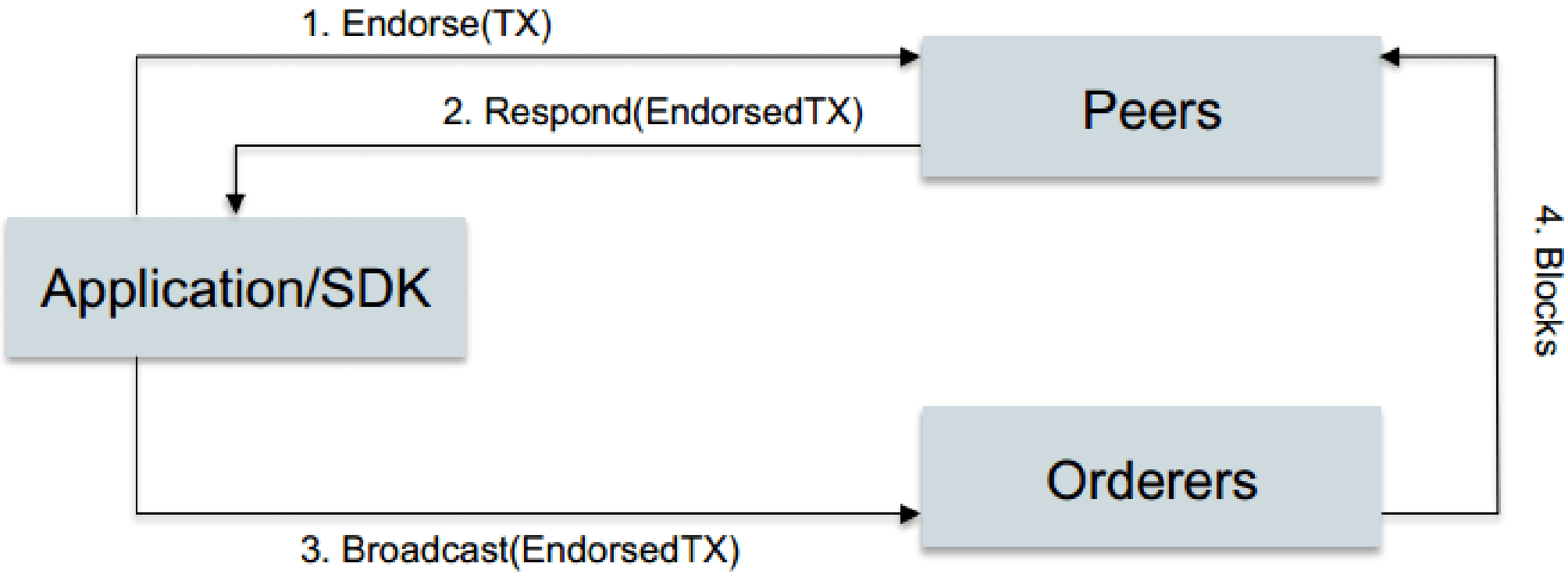
Fabric 运行架构 (V1.0及之后)



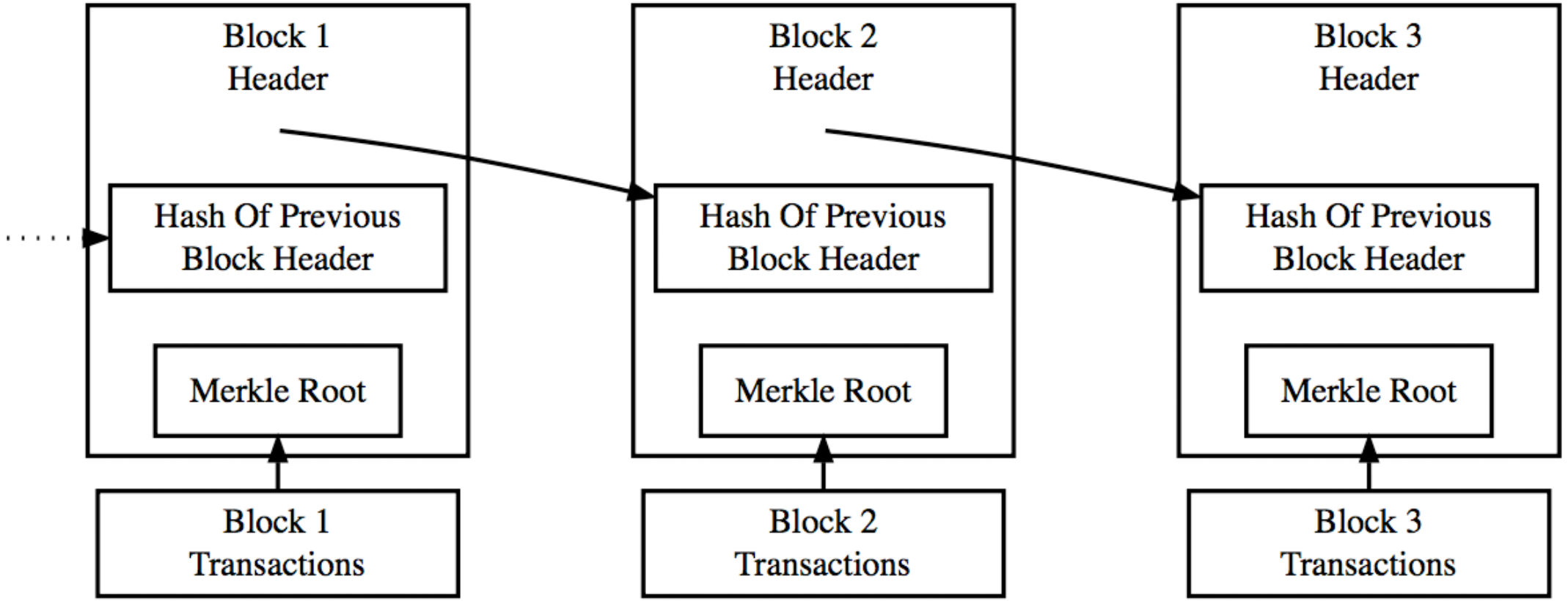
交易流程



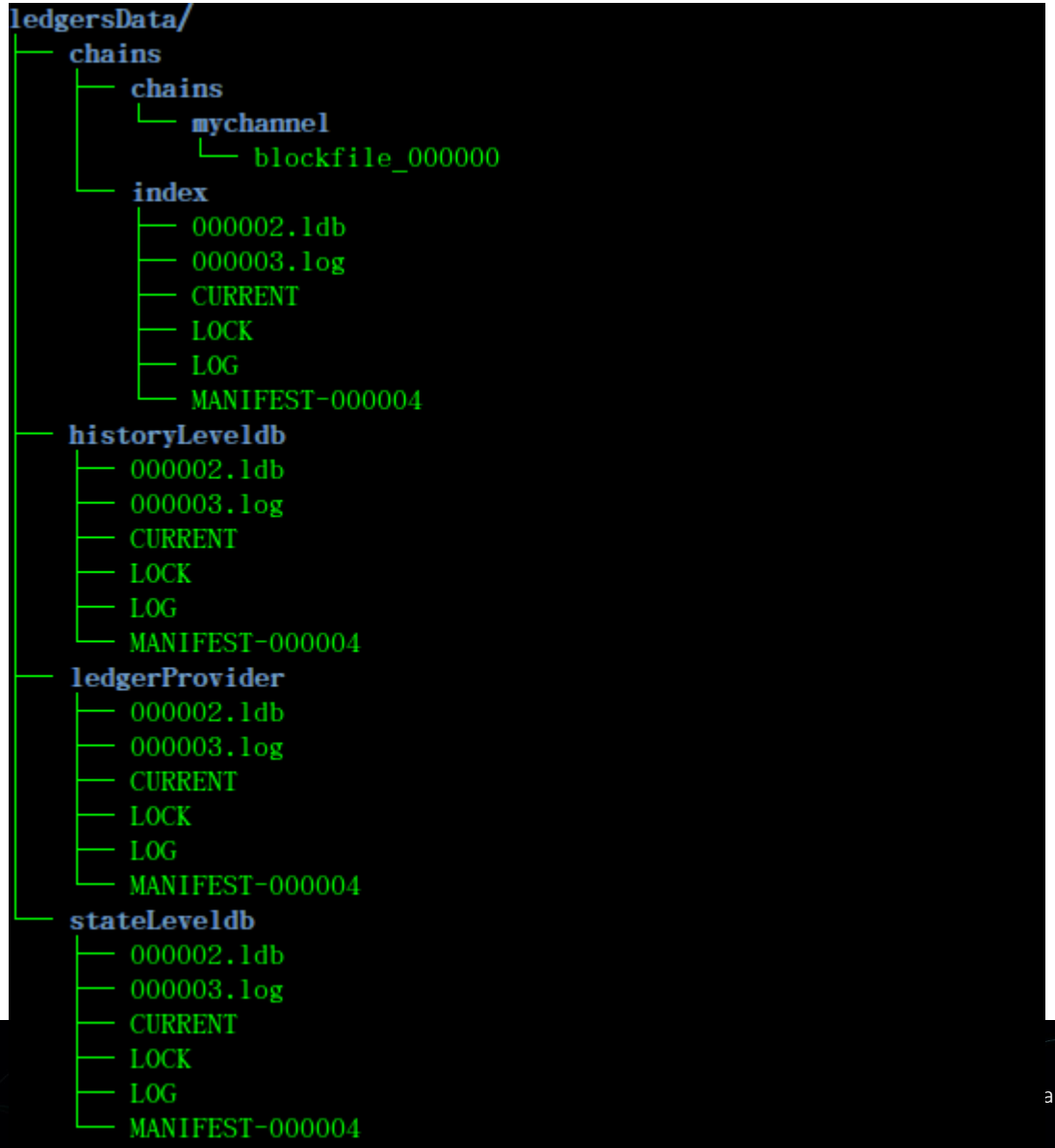
简化流程



链式结构



账本整体结构



chains：chains/chains下包含的mychannel是对应的channel的名称，存储的是每个channel都有自己的账本。chains/index下面包含的是levelDB数据库文件，在Fabric中默认所有的数据库都是LevelDB，DB中存储的就是区块索引部分。chains/chains和chains/index就是上面所说的File System和indexDB;

stateLeveldb： 同样是levelDB数据库，存储的是链码putstate写入的数据;

ledgerProvider： 数据库内存储的是当前节点所包含channel的信息

historyLeveldb： 数据库内存储的是链码中写入的key的历史记录的索引地址。

区块结构

区块结构体定义

```
type Block struct {
    Header   *BlockHeader
    Data     *BlockData
    Metadata *BlockMetadata
}
```

区块头：唯一区块号、前一个区块头Hash和DataHash(它是当前区块中的BlockData部分 SHA256 后的hash值)

区块数据：包含多条交易数据

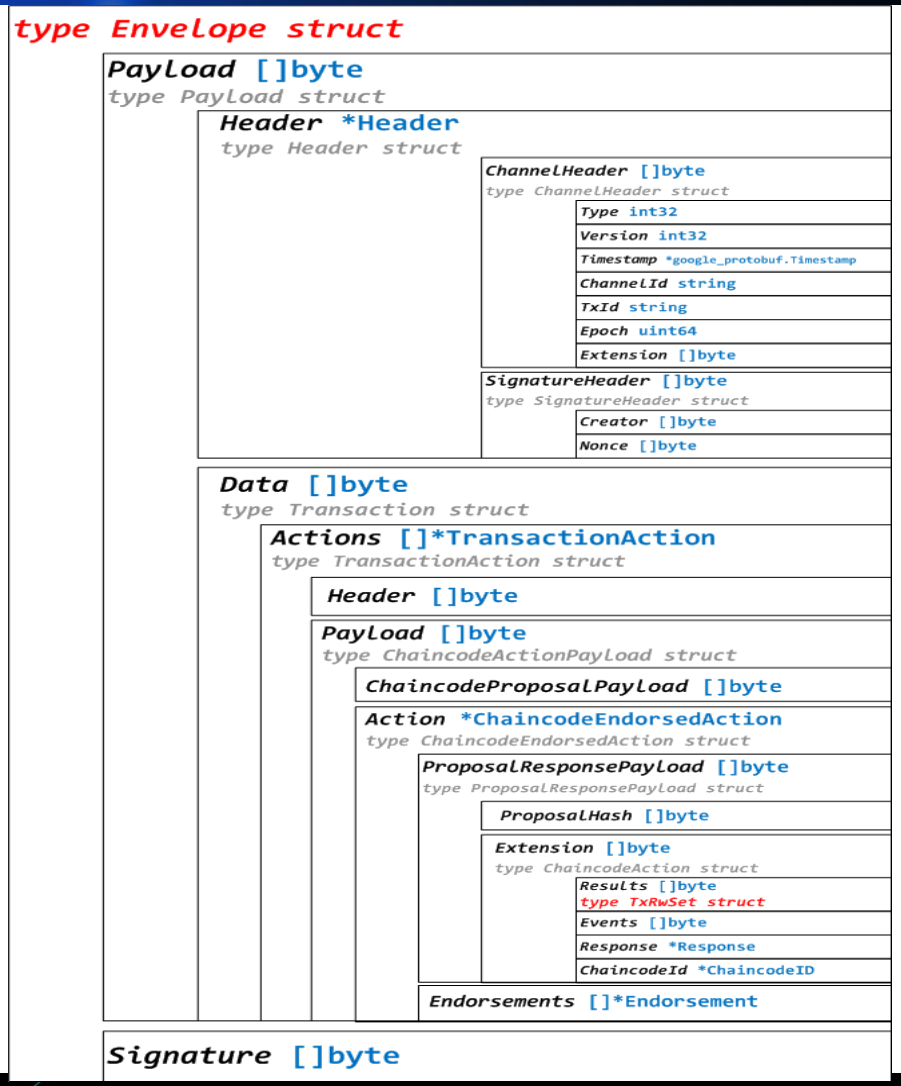
区块元数据：包含4种元数据，SIGNATURES、LAST_CONFIG、ORDERER和TRANSACTIONS_FILTER，前三个是Order service添加的，最后一个是Committer添加的。

交易（Transation）结构

交易的封装结构体Envelope:
Envelope结构体封装Payload和Signature。

type Envelope struct { //用签名包装Payload，以便对信息做身份验证
    Payload []byte //Payload序列化
    Signature []byte //Payload header中指定的创建者签名
}

Header结构体:
type Header struct {
    ChannelHeader []byte
    SignatureHeader []byte
}



ChannelHeader结构体:

```
type ChannelHeader struct {
    Type int32
    Version int32 //消息协议版本
    Timestamp *google_protobuf.Timestamp //创建消息时的本地时间
    ChannelId string //消息绑定的ChannelId
    TxId string //TxId
    Epoch uint64 //纪元
    Extension []byte //可附加的扩展
}
```

SignatureHeader结构体:

```
type SignatureHeader struct {
    Creator []byte //消息的创建者, 指定为证书链
    Nonce []byte //可能只使用一次的任意数字，可用于检测重播攻击
}
```

Transaction结构体：

```
type Transaction struct {
    //Payload.Data是个TransactionAction数组，容纳每个交易
    Actions []*TransactionAction
}
```

TransactionAction结构体：

```
type TransactionAction struct {
    Header []byte
    Payload []byte
}
```

读写集 TxRwSet

```
<TxReadWriteSet>
  <NsReadWriteSet name="chaincode1">
    <read-set>
      <read key="K1", version="1">
      <read key="K2", version="1">
    </read-set>
    <write-set>
      <write key="K1", value="V1"
      <write key="K3", value="V2"
      <write key="K4", isDelete="true"
    </write-set>
  </NsReadWriteSet>
<TxReadWriteSet>
```

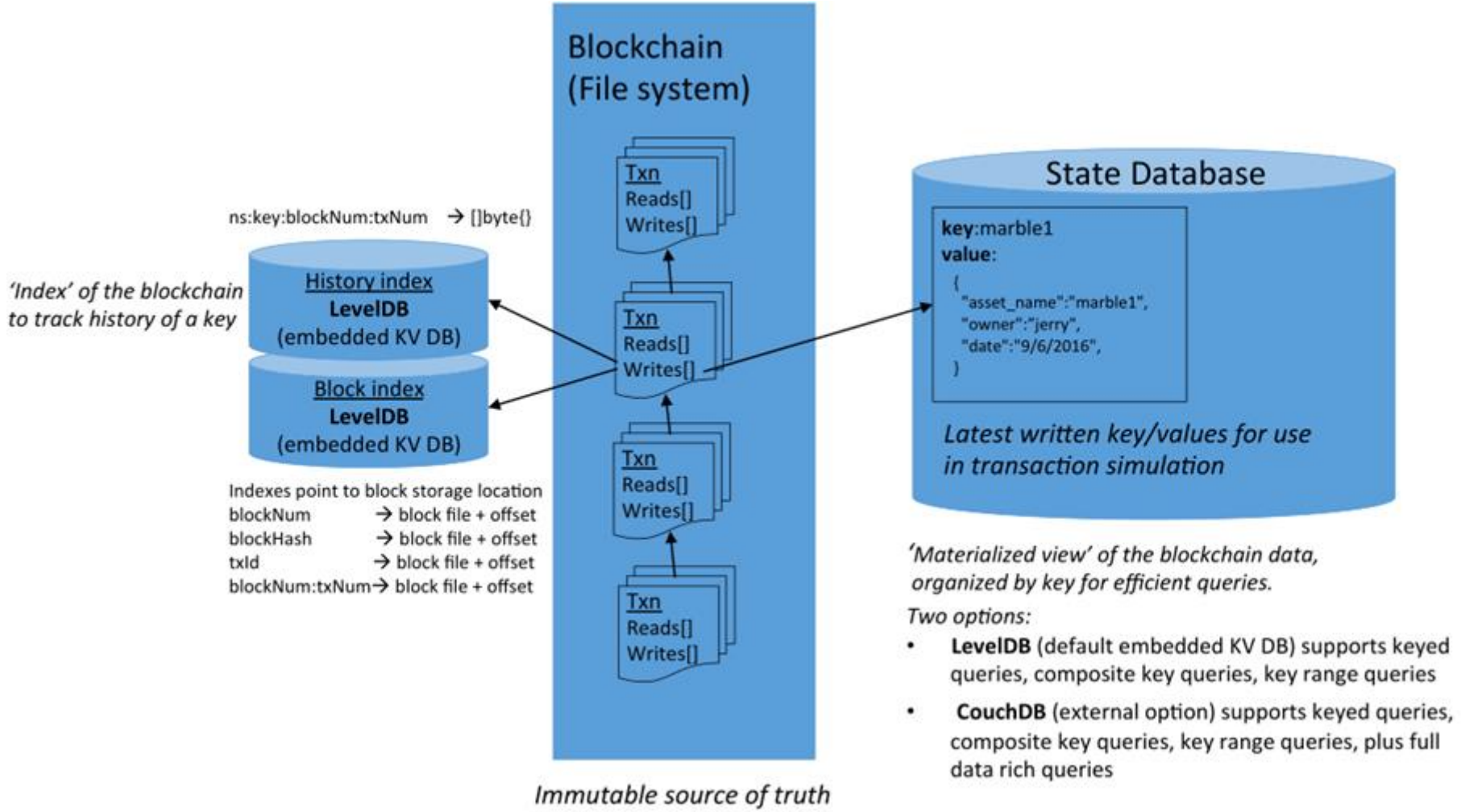
理解读写集，键k在世界状态中被表示为（k,ver,val），其中ver是最新版本，val是它的值。
现在假设有T1,T2,T3,T4和T5共5笔交易，所有的模拟都基于同一个世界状态的快照。下面的片段展示了世界状态的快照以及交易模拟和每个交易模拟的一系列读写操作:

```
World state: (k1,1,v1), (k2,1,v2), (k3,1,v3), (k4,1,v4), (k5,1,v5)
T1 -> Write(k1, v1'), Write(k2, v2')
T2 -> Read(k1), Write(k3, v3')
T3 -> Write(k2, v2'')
T4 -> Write(k2, v2'''), read(k2)
T5 -> Write(k6, v6'), read(k5)
```

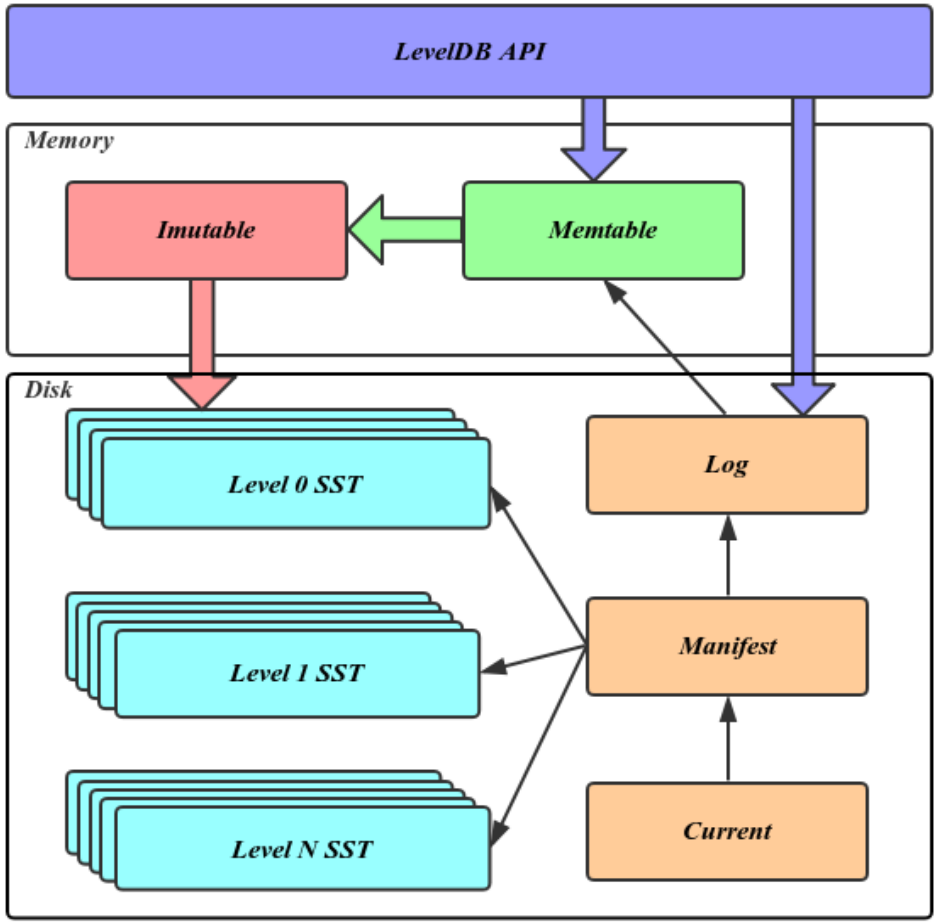
假设交易被排序为T1到T5依次执行：

1、T1通过验证，因为它并没有任何读操作。并且世界状态中k1和k2键的值别更新为(k1,2,v1'), (k2,2,v2')。
2、T2不能通过验证，因为它读取了k1键的值，该值在T1中被修改了。
3、T3通过验证，因为它并没有任何读操作。并且世界状态中k2的值被更新为(k2,3,v2")。
4、T4不能通过验证，因为它读取了k2键的值，该值在T1中被修改了。
5、T5通过验证，虽然读取了k5的值，但是k5的值之前交易并未修改过。

数据存储

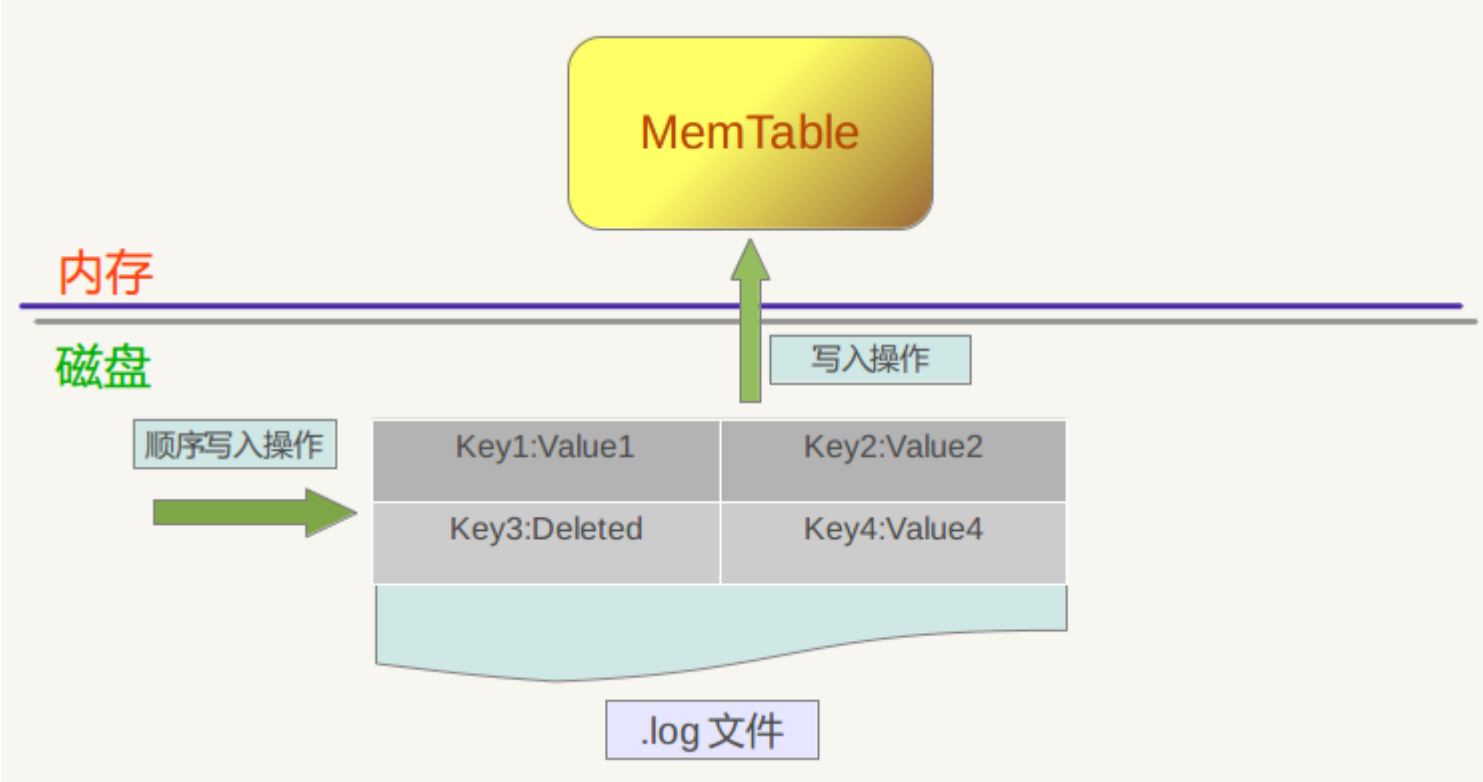


LevelDB整体结构



LevelDB的数据是持久存储在磁盘上的，采用LSM-Tree（Log-Structured Merge Tree）的结构实现。LSM-Tree将磁盘的随机写转化为顺序写，从而大大提高了写速度。

LevelDB写操作

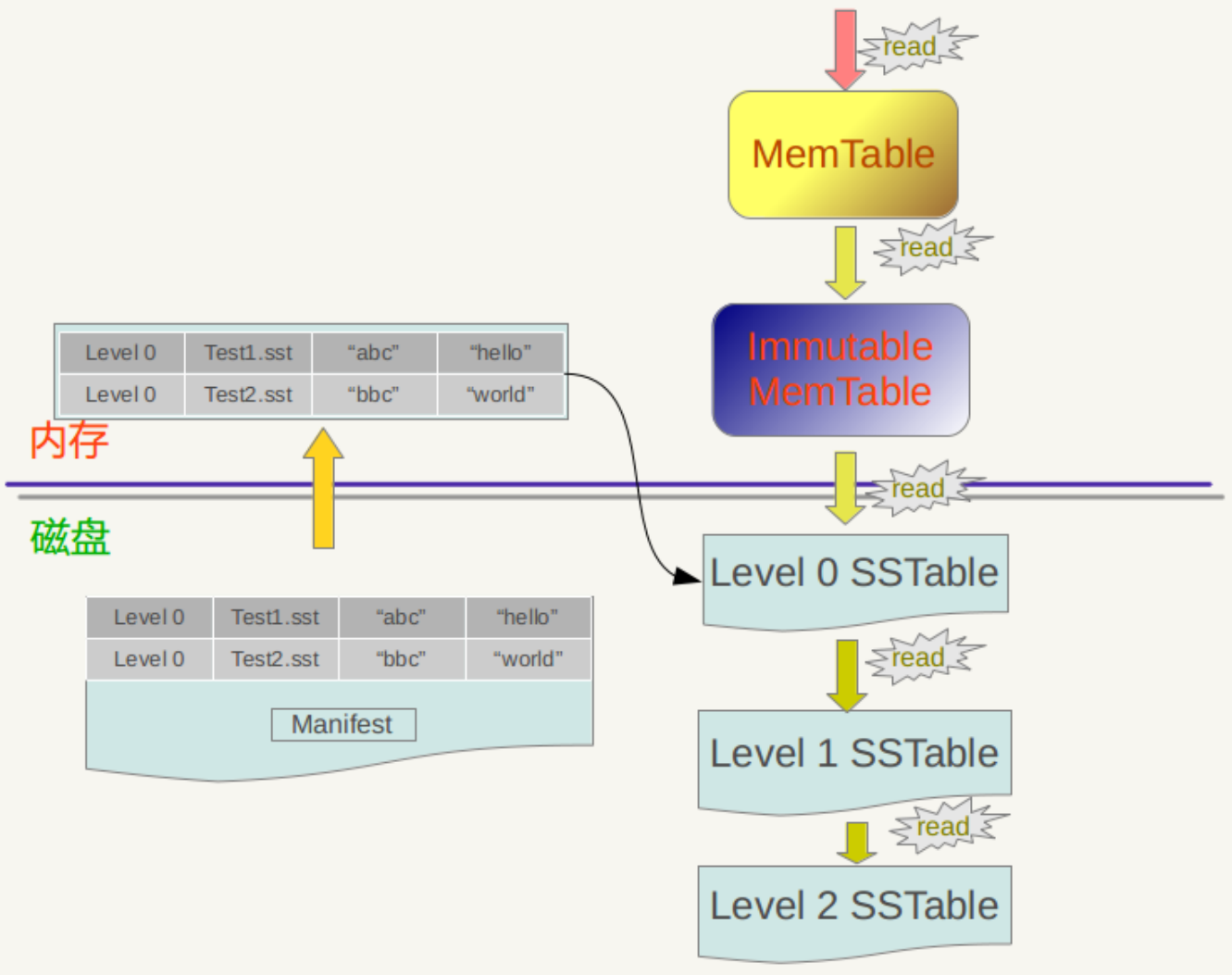


完成插入操作Put(Key,Value)包含两个具体步骤：

1）将这条KV记录追加到log文件末尾。

2）如果写入log文件成功，那么将这条KV记录插入内存中的Memtable中，插入一条新记录的过程很简单，即先查找合适的插入位置，然后修改相应的链接指针将新记录插入即可。完成这一步，写入记录就算完成了。

LevelDB读操作



用查询所用的Key，依次尝试从 Memtable，Immtable以及SST文件中读取，直到找到（或者查到最高level，查找失败，说明整个系统中不存在这个Key)。

数据存储-blockfile

基于文件系统的区块链数据库，存储了交易的读写集，不能更改，命名方式如下图
文件大小限制：**64M**

```
root@3579ad952b48:/var/hyperledger/production/ledgersData/chains/chains/mychannel# ls
blockfile_000000  blockfile_000040  blockfile_000080  blockfile_000120  blockfile_000160  blockfile_000200  blockfile_000240  blockfile_000280
blockfile_000001  blockfile_000041  blockfile_000081  blockfile_000121  blockfile_000161  blockfile_000201  blockfile_000241  blockfile_000281
blockfile_000002  blockfile_000042  blockfile_000082  blockfile_000122  blockfile_000162  blockfile_000202  blockfile_000242  blockfile_000282
blockfile_000003  blockfile_000043  blockfile_000083  blockfile_000123  blockfile_000163  blockfile_000203  blockfile_000243  blockfile_000283
blockfile_000004  blockfile_000044  blockfile_000084  blockfile_000124  blockfile_000164  blockfile_000204  blockfile_000244  blockfile_000284
blockfile_000005  blockfile_000045  blockfile_000085  blockfile_000125  blockfile_000165  blockfile_000205  blockfile_000245  blockfile_000285
blockfile_000006  blockfile_000046  blockfile_000086  blockfile_000126  blockfile_000166  blockfile_000206  blockfile_000246  blockfile_000286
blockfile_000007  blockfile_000047  blockfile_000087  blockfile_000127  blockfile_000167  blockfile_000207  blockfile_000247  blockfile_000287
blockfile_000008  blockfile_000048  blockfile_000088  blockfile_000128  blockfile_000168  blockfile_000208  blockfile_000248  blockfile_000288
blockfile_000009  blockfile_000049  blockfile_000089  blockfile_000129  blockfile_000169  blockfile_000209  blockfile_000249  blockfile_000289
blockfile_000010  blockfile_000050  blockfile_000090  blockfile_000130  blockfile_000170  blockfile_000210  blockfile_000250  blockfile_000290
blockfile_000011  blockfile_000051  blockfile_000091  blockfile_000131  blockfile_000171  blockfile_000211  blockfile_000251  blockfile_000291
```

数据存储-stateDB

状态数据库用来记录当前的Fabric状态,存储了交易（transaction）中所有键的最新值，也称世界状态（world state）。可选择基于leveldb或couchdb实现。存储的是key-value键值对，交易中writeSet，按照key的顺序存储。

结构形式如下：

Key:car1

Value:

```
{
  asset_name: “car1”,
  owner:”jacky”
}
```

数据存储-indexDB

历史数据和区块链索引的leveldb数据库，不能更改。

HLF 区块索引信息格式在 kv 数据库中存储的最终的 LevelKey 值有前缀标志和区块 hash 组成，而 LevelValue 的值由区块高度，区块 hash，本地文件信息(文件名，文件偏移等信息)，每个交易在文件中的偏移列表和区块的 MetaData 组成，HLF 按照特定的编码方式将上述的信息拼接成 db 数据库中的 value 。

HLF交易索引信息格式在kv数据库中存储最终的LevelKey值由channel_name，chaincode_name和chaincode中的key值组合而成。而 LevelValue 的值由BlockNum 区块号，TxNum 交易在区块中的编号组成， HLF 通过将区块号和交易编号按照特定的方式编码，然后与 chaincode 中的 value 相互拼接最终生成 db 数据库中的 value 。

```
root@3579ad952b48:/var/hyperledger/production/ledgersData/chains/index# ls
000134.ldb  001358.ldb  002858.ldb  005555.ldb  005575.ldb  005588.ldb  005601.ldb  005614.ldb  005627.ldb  005640.ldb  005653.ldb  005666.ldb
000450.ldb  001709.ldb  004056.ldb  005556.ldb  005576.ldb  005589.ldb  005602.ldb  005615.ldb  005628.ldb  005641.ldb  005654.ldb  005667.ldb
001051.ldb  001757.ldb  005296.ldb  005557.ldb  005577.ldb  005590.ldb  005603.ldb  005616.ldb  005629.ldb  005642.ldb  005655.ldb  005668.ldb
001299.ldb  001758.ldb  005545.ldb  005558.ldb  005578.ldb  005591.ldb  005604.ldb  005617.ldb  005630.ldb  005643.ldb  005656.ldb  005669.ldb
001349.ldb  001759.ldb  005546.ldb  005559.ldb  005579.ldb  005592.ldb  005605.ldb  005618.ldb  005631.ldb  005644.ldb  005657.ldb  005670.ldb
001350.ldb  001760.ldb  005547.ldb  005560.ldb  005580.ldb  005593.ldb  005606.ldb  005619.ldb  005632.ldb  005645.ldb  005658.ldb  005671.log
001351.ldb  001761.ldb  005548.ldb  005561.ldb  005581.ldb  005594.ldb  005607.ldb  005620.ldb  005633.ldb  005646.ldb  005659.ldb  CURRENT
001352.ldb  001762.ldb  005549.ldb  005562.ldb  005582.ldb  005595.ldb  005608.ldb  005621.ldb  005634.ldb  005647.ldb  005660.ldb  LOCK
001353.ldb  001763.ldb  005550.ldb  005563.ldb  005583.ldb  005596.ldb  005609.ldb  005622.ldb  005635.ldb  005648.ldb  005661.ldb  LOG
001354.ldb  001764.ldb  005551.ldb  005564.ldb  005584.ldb  005597.ldb  005610.ldb  005623.ldb  005636.ldb  005649.ldb  005662.ldb  MANIFEST-005672
001355.ldb  001765.ldb  005552.ldb  005565.ldb  005585.ldb  005598.ldb  005611.ldb  005624.ldb  005637.ldb  005650.ldb  005663.ldb
001356.ldb  001766.ldb  005553.ldb  005566.ldb  005586.ldb  005599.ldb  005612.ldb  005625.ldb  005638.ldb  005651.ldb  005664.ldb
001357.ldb  002061.ldb  005554.ldb  005567.ldb  005587.ldb  005600.ldb  005613.ldb  005626.ldb  005639.ldb  005652.ldb  005665.ldb
root@3579ad952b48:/var/hyperledger/production/ledgersData/chains/index#
```

总结

1. Fabric是许可、隐私保护网络，提供可定制和可扩展的架构
2. 灵活、松耦合，可较自由组合、替换组件。可配置出块时间、区块大小及交易数上限等
3. 复杂性：平台架构较复杂；数据存储也比较复杂：包括存储交易信息的blockfile,存储世界状态的stateDB,存储历史信息的historyDB,以及存储索引的indexDB
4. 共识融入交易流程中，包括背书、排序和验证提交。

Join US!

- 让我们一起改变世界

区块链工程师/资深区块链工程师/区块链架构师

前端工程师/高级前端工程师

后端工程师/高级后端工程师/技术专家

项目经理（scrum master）

产品经理/产品总监

袁运亮
广东 广州

Mail: yunliang.yuan@defangchain.com


DEFANG
TECHNOLOGY
德方智链
Thank You