GopherChina2018

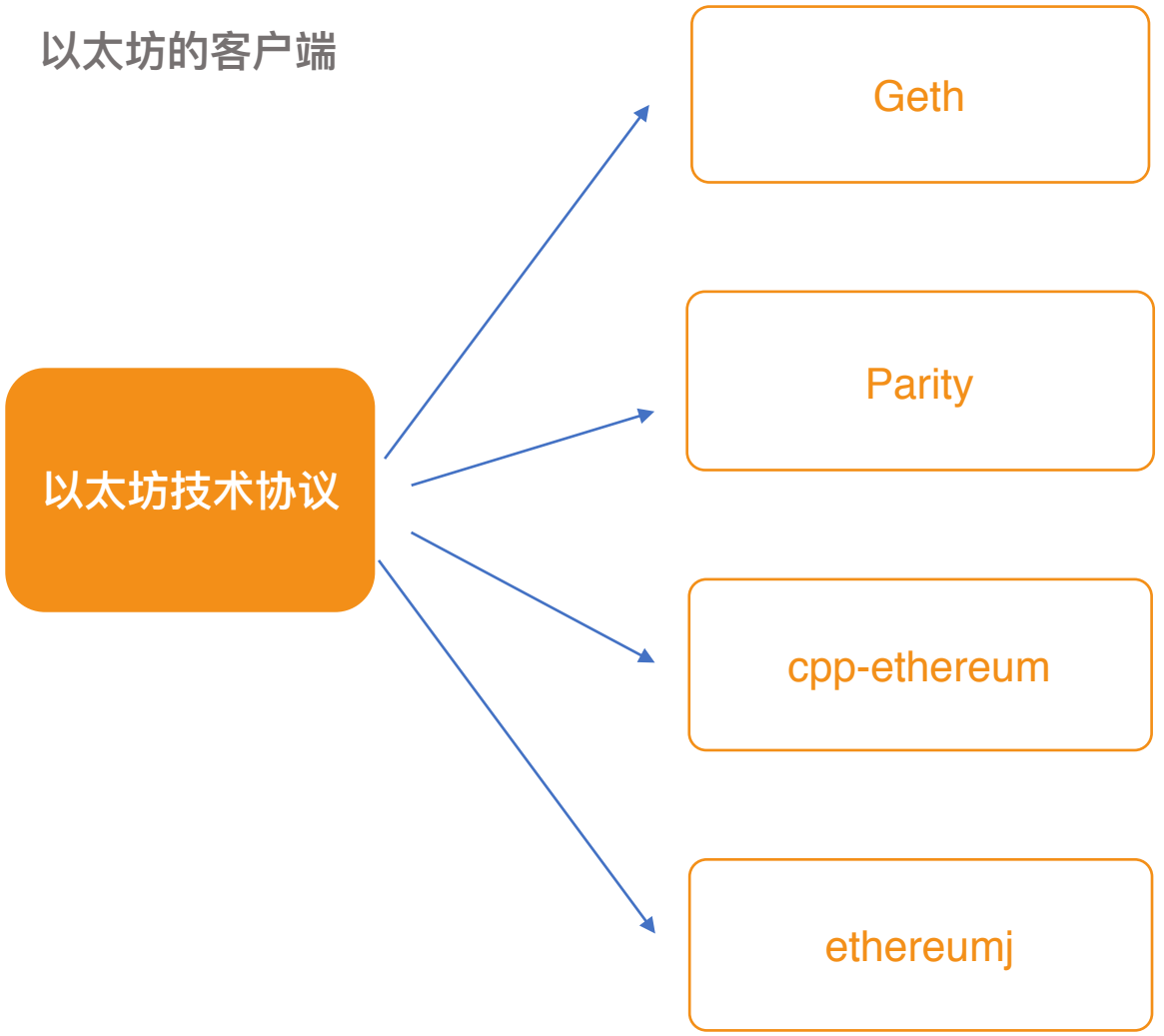
基于Go-Ethereum构建DPOS机制下的区块链

恺英网络—朱崇文

目录

Go版本以太坊

以太坊的客户端



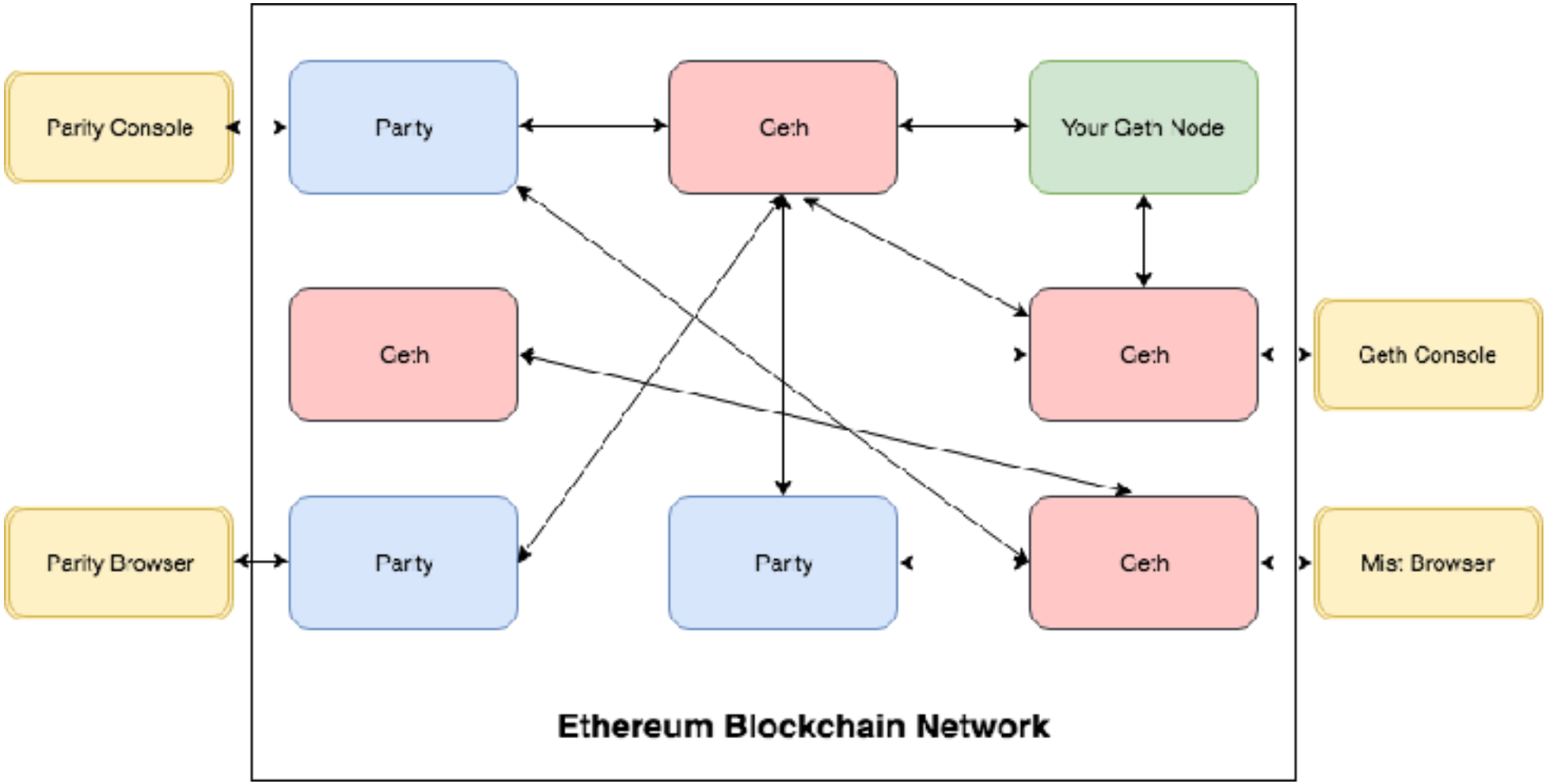
- **Go、官方**版本实现
- Rust的实现，第二大客户端

Go版本以太坊

以太坊的工具组

Go版本以太坊

以太坊公链网络拓扑



为何选择DPOS机制

共识机制对比

POW

- 消耗计算力
- 出块速度慢，确认慢
- TPS极低 10~20
- 确认 1分钟+

DPOS

- 代理人模式
- 出块速度快，确认快
- TPS 700~1000 (实现)
- 平均确认1~3秒

为何选择DPOS机制

DPOS机制的优势

系统可靠性

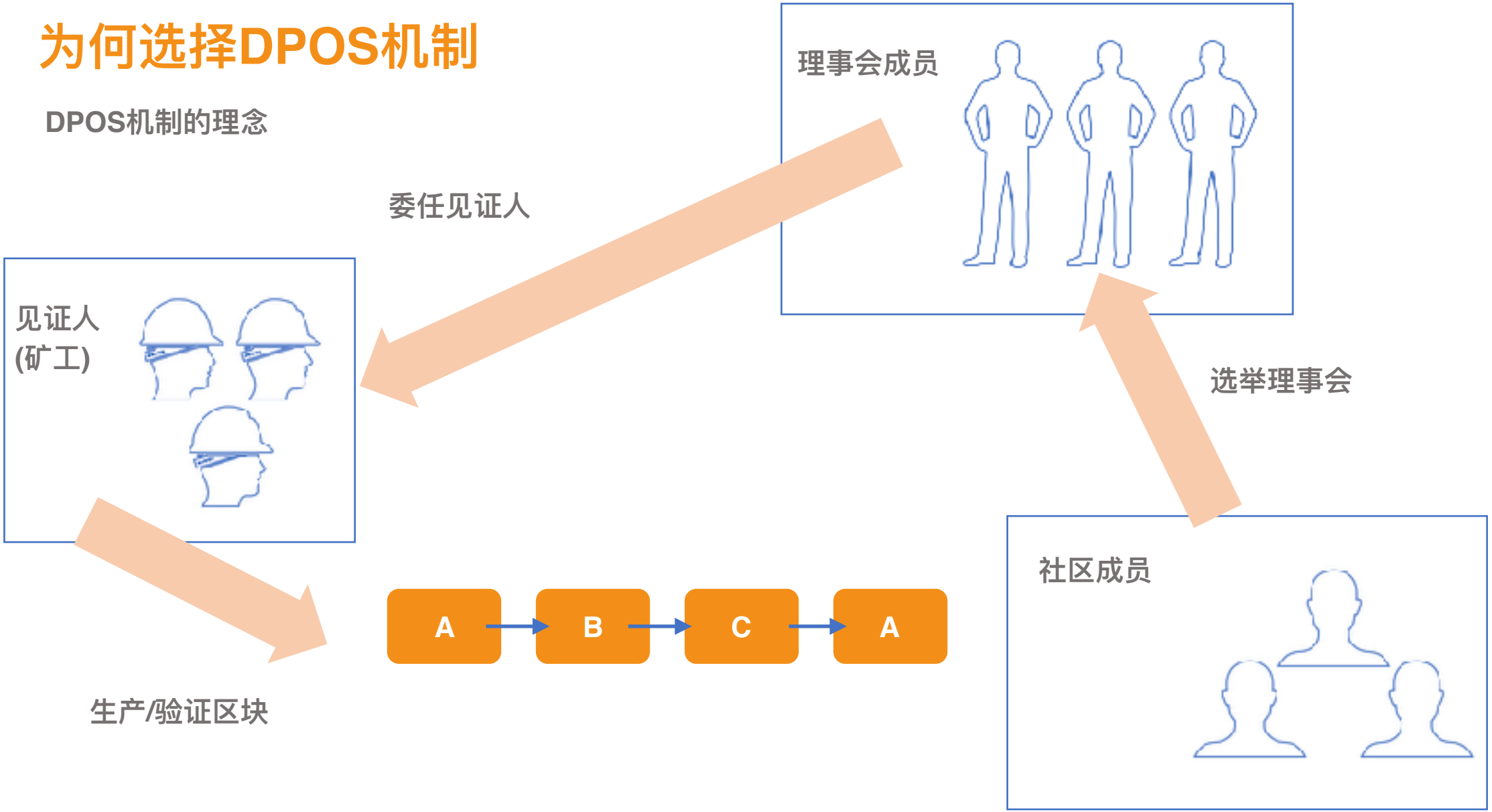
- 在商业场景下，网络性能可控
- 对异常情况能快速处理并恢复
- 对TPS/QPS，以及确认性能有一定要求

区块链可信

- 以公有链为基础，可对外开放
- 任何人都可以参与，设立理事会和见证人角色
- 理事会管理区块链网络
- 见证人生产并验证区块

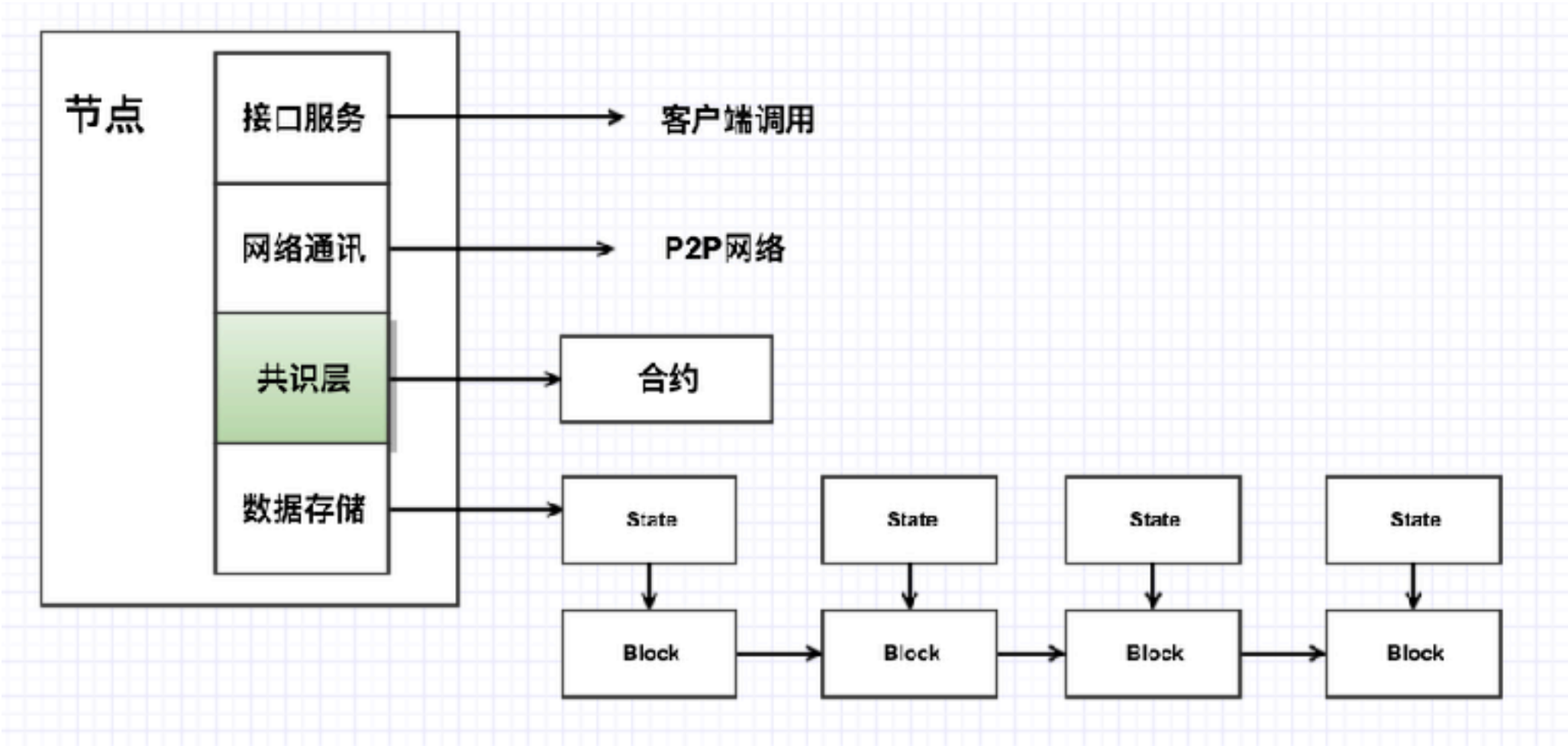
为何选择DPOS机制

DPOS机制的理念



拓展共识改造实战

共识框架引擎—改造共识层逻辑



拓展共识改造实战

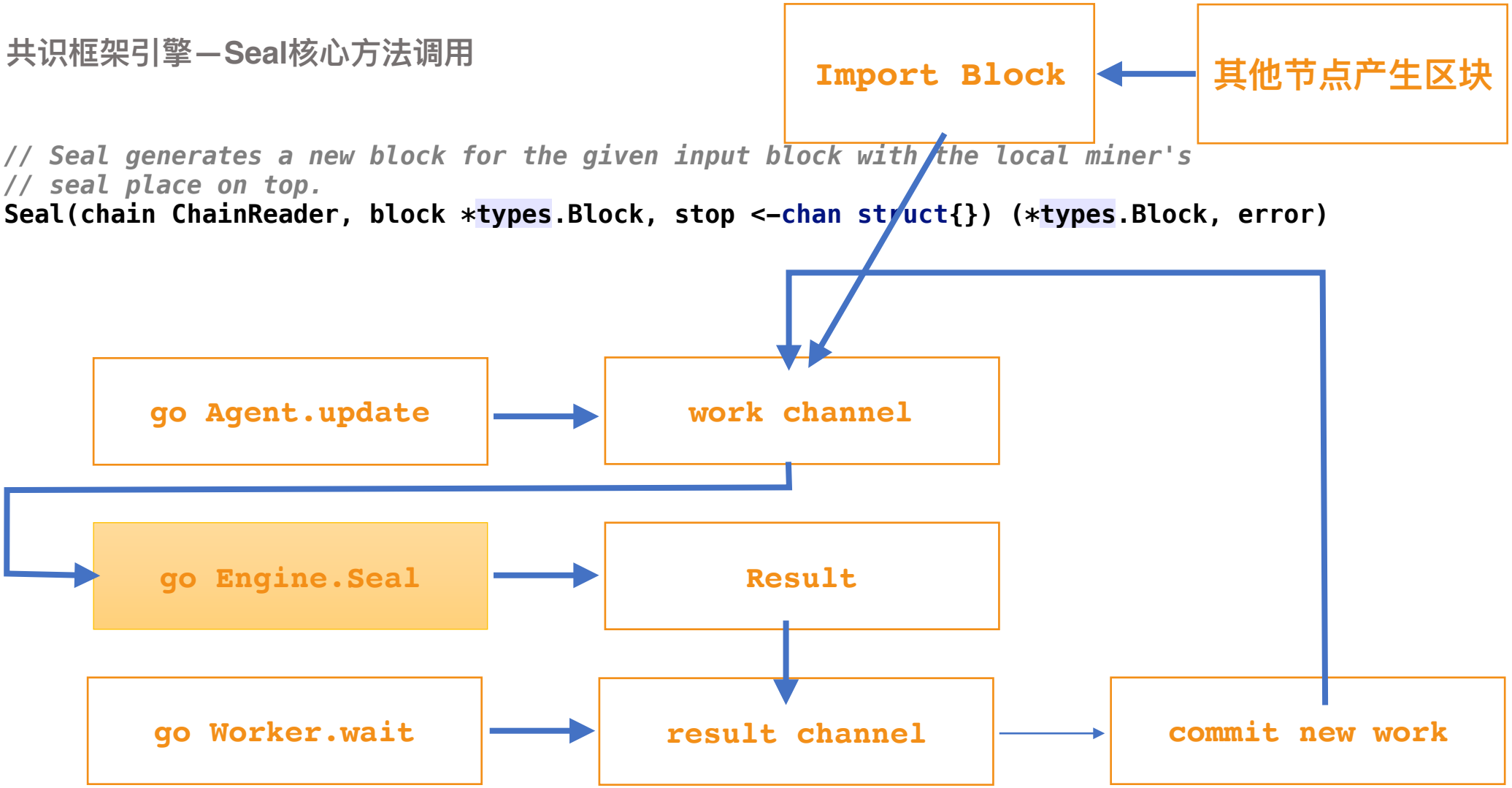
共识框架引擎—官方实现引擎: Ethash / Clique

```
// Engine is an algorithm agnostic consensus engine.
type Engine interface {

    Author(header *types.Header) (common.Address, error)

    VerifyHeader(chain ChainReader, header *types.Header, seal bool) error

    VerifyHeaders(chain ChainReader, headers []*types.Header, seals []bool) (chan<-
struct{}, <-chan error)

    VerifyUncles(chain ChainReader, block *types.Block) error

    VerifySeal(chain ChainReader, header *types.Header) error

    Prepare(chain ChainReader, header *types.Header) error

    Finalize(chain ChainReader, header *types.Header, state *state.StateDB, txs
[]*types.Transaction,
        uncles []*types.Header, receipts []*types.Receipt) (*types.Block, error)

    Seal(chain ChainReader, block *types.Block, stop <-chan struct{}) (*types.Block,
error)

    CalcDifficulty(chain ChainReader, time uint64, parent *types.Header) *big.Int

    APIs(chain ChainReader) []rpc.API
}
```

拓展共识改造实战

共识框架引擎—Seal核心方法调用

```
// Seal generates a new block for the given input block with the local miner's
// seal place on top.
Seal(chain ChainReader, block *types.Block, stop <-chan struct{}) (*types.Block, error)
```

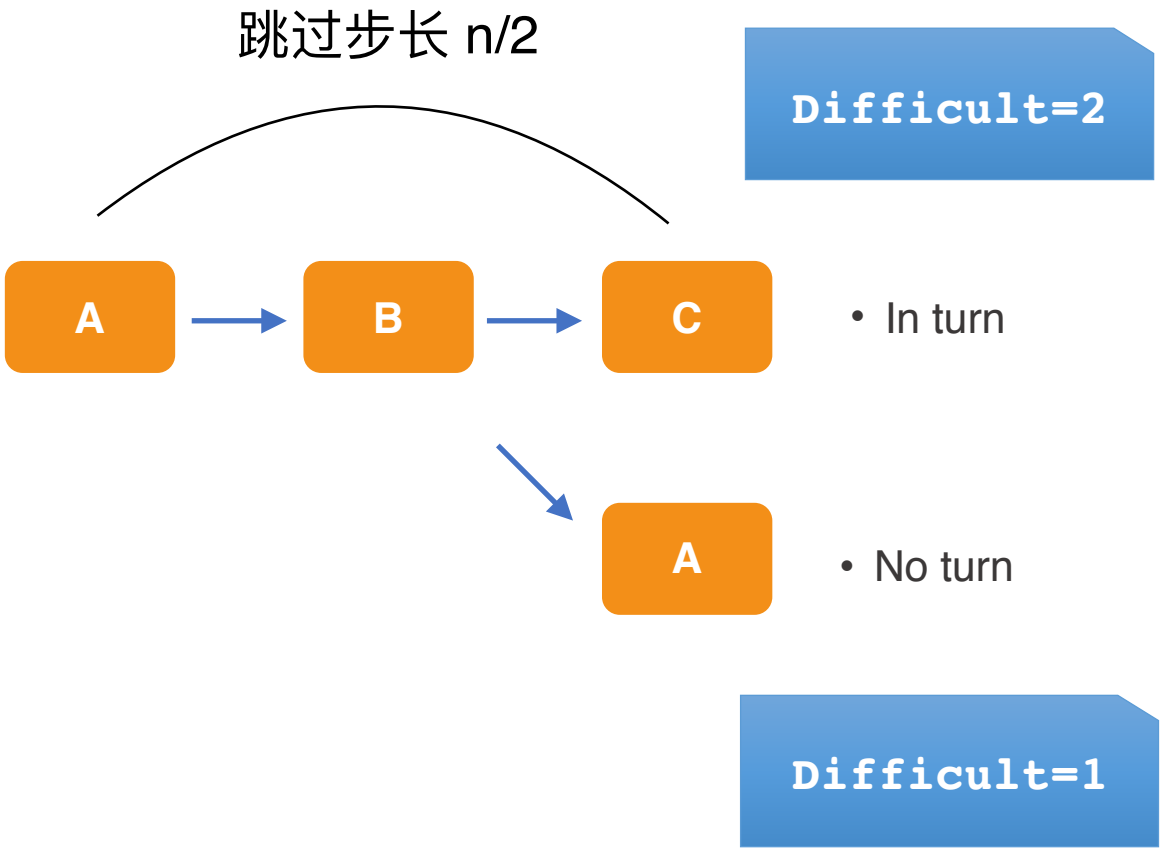


拓展共识改造实战

借鉴Clique(POA)的实现

Clique

- Go-Ethereum 实现的机制，
  用以公共测试链
- 整个网络由Signer节点出块
- Signer节点可以投票选择其他Signer节点
- 节点之间可以相互竞争出块
- 存活节点数 > (n/2) + 1
- Signer 节点的选举记录在Extra Data中



拓展共识改造实战

借鉴Clique(POA)的实现

- 满足条件，竞争出块
- 非自己的轮次，小小的延迟

```
// Sweet, the protocol permits us to sign the block, wait for our time
delay := time.Unix(header.Time.Int64(), 0).Sub(time.Now()) // nolint: gosimple
if header.Difficulty.Cmp(diffNoTurn) == 0 {
    // It's not our turn explicitly to sign, delay it a bit
    wiggle := time.Duration(len(snap.Signers)/2+1) * wiggleTime
    delay += time.Duration(rand.Int63n(int64(wiggle)))

select {
case <-stop:
    return nil, nil
case <-time.After(delay):
}
```

竞争出块的节点有一点延迟，避免过多分叉

拓展共识改造实战

扩展区块头结构—增加见证人列表

- 框架程度较高

  很容易扩展字段

- 地址按字典排序

区块头结构

```
// Header represents a block head      the Ethereum blockchain.
type Header struct {
    ParentHash  common.Hash    `json:"parentHash"       gencodec:"required"`
    UncleHash   common.Hash    `json:"sha3Uncles"       gencodec:"required"`
    Coinbase    common.Address `json:"miner"            gencodec:"required"`
    Root        common.Hash    `json:"stateRoot"        gencodec:"required"`
                               ------------
                                              ----------------
    WitnessVotes WitnessVoteSlice  `json:"witnessVotes"   gencodec:"required"`
}
```

数据案例

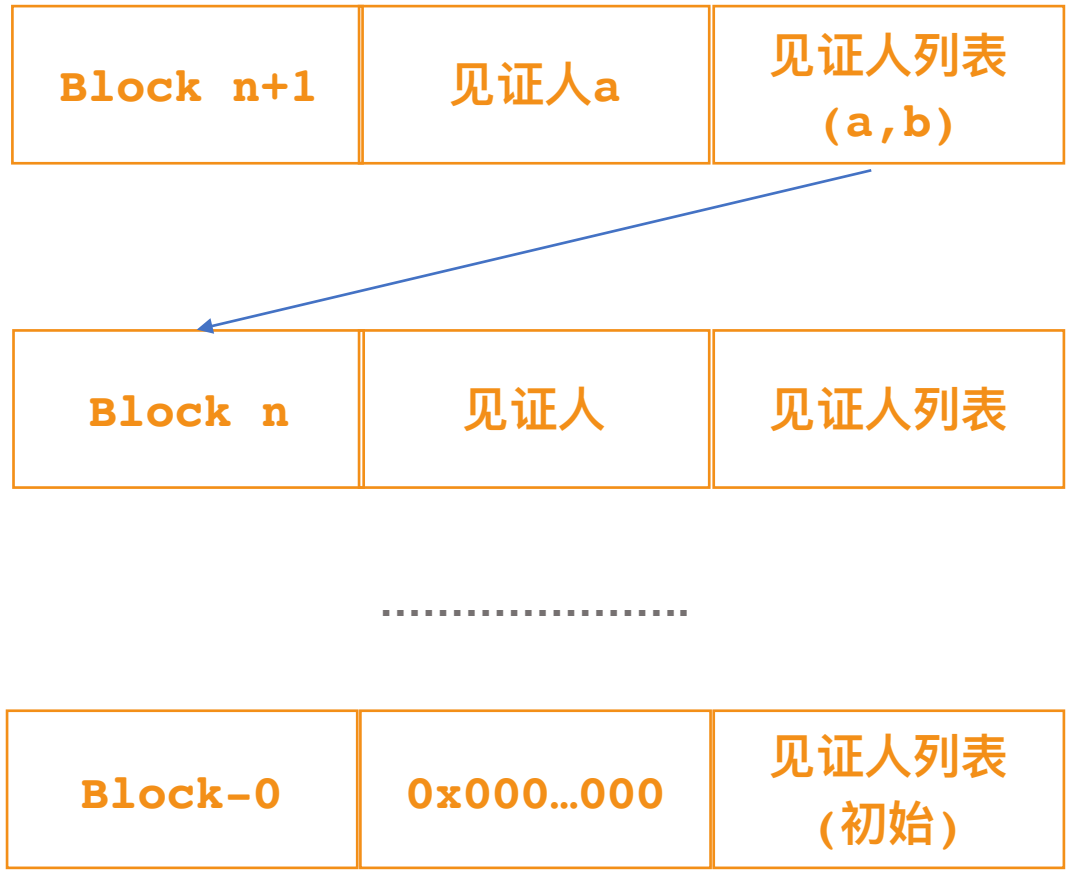
```
witnessVotes: [{
    address: "0x36114a04a4f621da0f19b8453f414cf1f0c40757"
}, {
    address: "0x45db4d996660a6cf312ccf2a6e42fb723bab1be5"
}, {
    address: "0x6e2059b7aabc5179a01a5b54523747b15620b4e3"
}, {
    address: "0xa2477cadf6b3531072d11dab58e76729346baa69"
}, {
    address: "0xe0263a75d6b7f5ebdcce6546bed9e7b6102a2d8d"
}]
```

拓展共识改造实战

扩展区块头结构—见证人列表生成规则

- 区块N的见证人列表为区块(N-1)

  所达成共识的见证人



拓展共识改造实战

轮流生产者的实现 — 判断当前轮次是否需要产块

`is_step_proposer`

- 当前时间戳
- 当前区块的产块人
- Parent区块的见证人
- Parent区块的时间戳
- 产块周期

拓展共识改造实战

轮流生产者的实现 — 场景分析

- 产块周期3秒，当前时间T+3
- 三个见证人节点判断该时间点是否应该出块

Parent	Block n	见证人 a	见证人列表 (a,b,c)	Timestamp T	
节点A	Block n+1	见证人 a	见证人列表 (a,b,c)	Timestamp T+3	
节点B	Block n+1	见证人 b	见证人列表 (a,b,c)	Timestamp T+3	
节点C	Block n+1	见证人 c	见证人列表 (a,b,c)	Timestamp T+3	

拓展共识改造实战

轮流生产者的实现 – 判断当前轮次–代码实现

```
// parent的signer存在于合法签发列表中
if parentOk {
    delta = int64(math.Abs(float64(currentIndex) - float64(parentIndex)))
    // 上一个签发的是自己，则等待len(Signers)周期
    if delta == 0 {
        delta = int64(len(snap.Signers))
    } else {
        // 如果 p 在 c 后面，从P开始计算差
        if (parentIndex > currentIndex) {
            delta = signsLength - delta
        }
    }
} else {
    // parent不在当前 singer list中， 则默认从当前的index + 1开始计算delta
    delta = int64(currentIndex + 1)
}
                        -----------
                            ----------------
                        -----------
                                ----------------

if (parentTimestamp + delta * period ) <= currentHeaderTime {
   return true
}
```

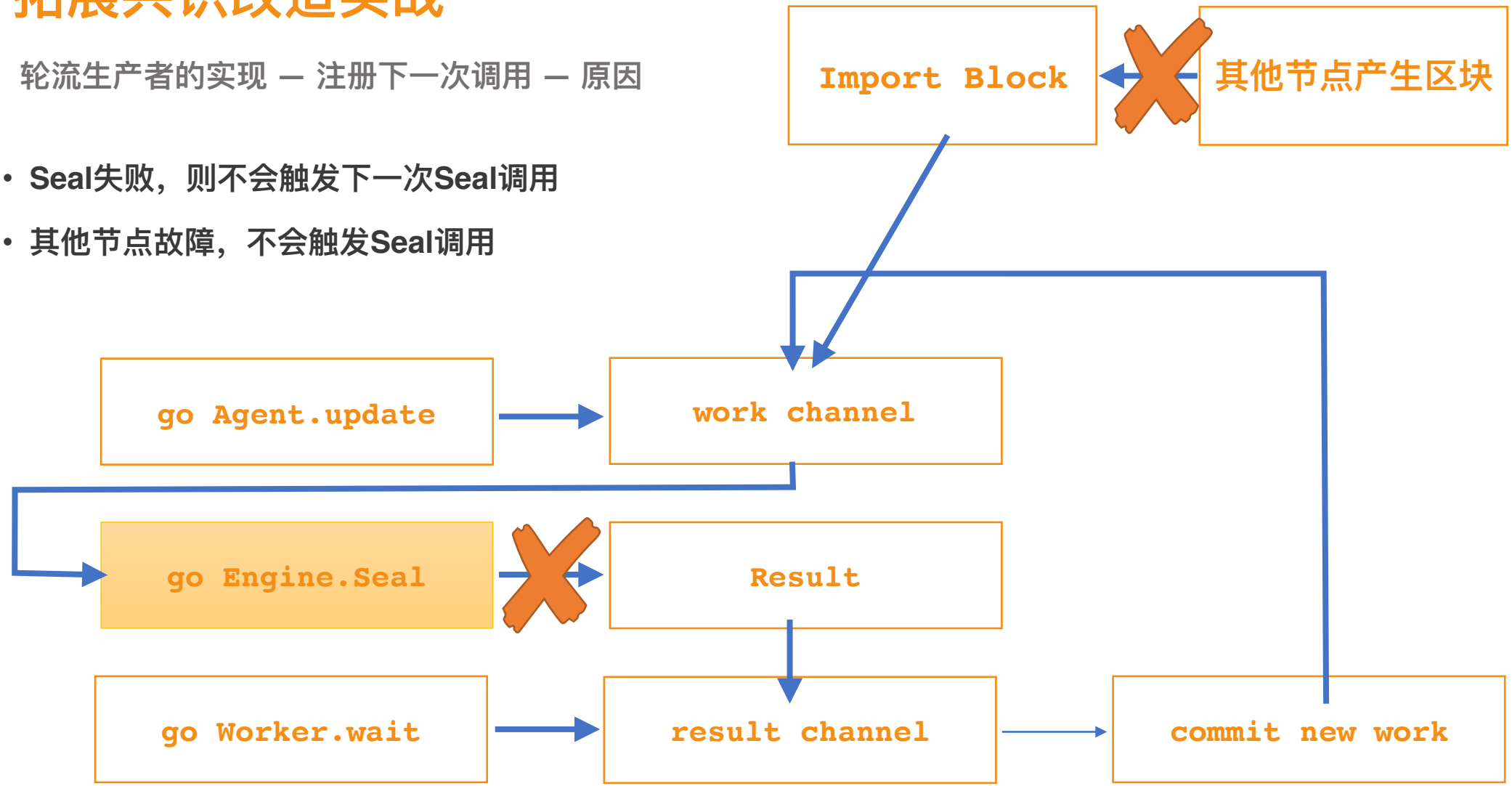
计算与Parent块之间的轮次差值

和当前出块时间比较

拓展共识改造实战

轮流生产者的实现 — 注册下一次调用 — 原因

- Seal失败，则不会触发下一次Seal调用
- 其他节点故障，不会触发Seal调用



拓展共识改造实战

轮流生产者的实现 — 注册下一次调用 — 场景分析

- 注册下一次delay时间，最大值是3秒，防止陷入side fork
- 最小值 = (3-块打包消耗时间)

拓展共识改造实战

轮流生产者的实现 — 注册下一次调用 — 代码实现

- Seal方法中，通过defer来注册下一次调用

```
// 注册返回函数，执行下一次mine
defer func() {
    var duration = time.Duration(c.config.Period) * time.Second
    var currentTimeStamp = time.Now().Unix()
    // 没有赋值，等待一个默认周期
    if nextTimeStamp == 0 {
        go c.registerNextDurationTimer(duration)
    } else {
        var interval int64
        // 如果进度滞后，duration = 0
        if currentTimeStamp >= nextTimeStamp {
            interval = 0
        } else {
            // duration时间不超过一个周期，防止陷入 side fork
            interval = min(nextTimeStamp - currentTimeStamp, int64(c.config.Period))
        }
        duration := time.Duration(interval) * time.Second
        go c.registerNextDurationTimer(duration)
    }
}()
```

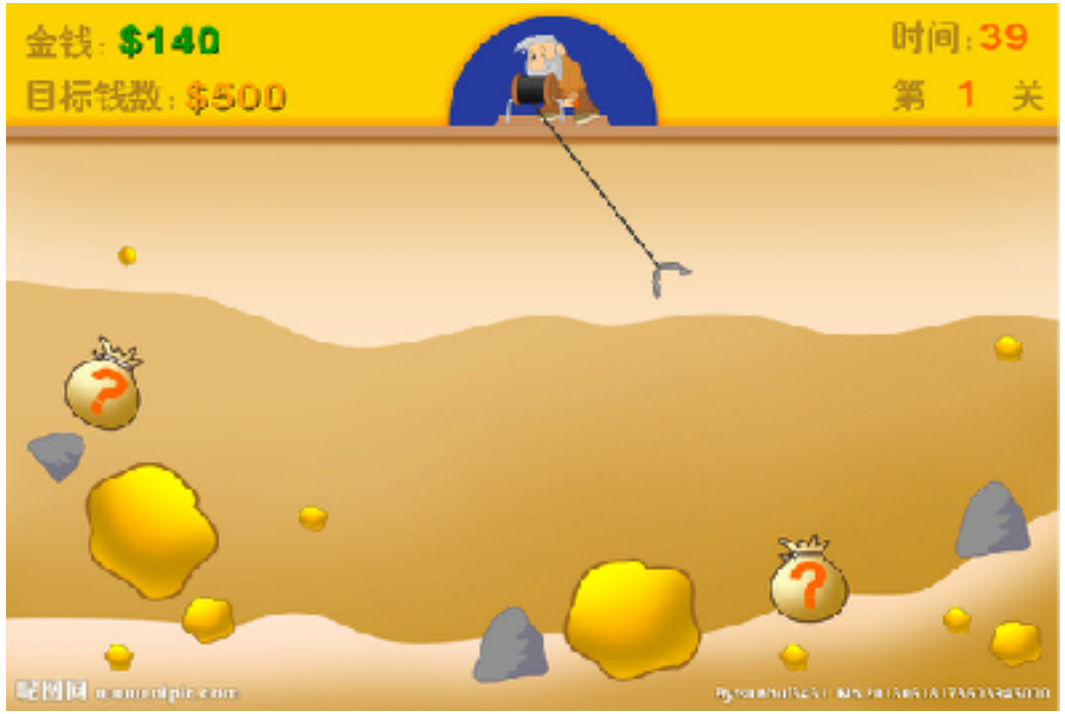
计算delay时间

注册函数

拓展共识改造实战

轮流生产者的实现 — 自定义奖励规则

- 抽象奖励逻辑
- 实现多种奖励策略



拓展共识改造实战

轮流生产者的实现 — 自定义奖励规则

```
"config": {
   "board": {
     "period": 3,
     "reward":"default_reward"
   }
 },
```

创世块配置中设置奖励规则

定义奖励规则接口

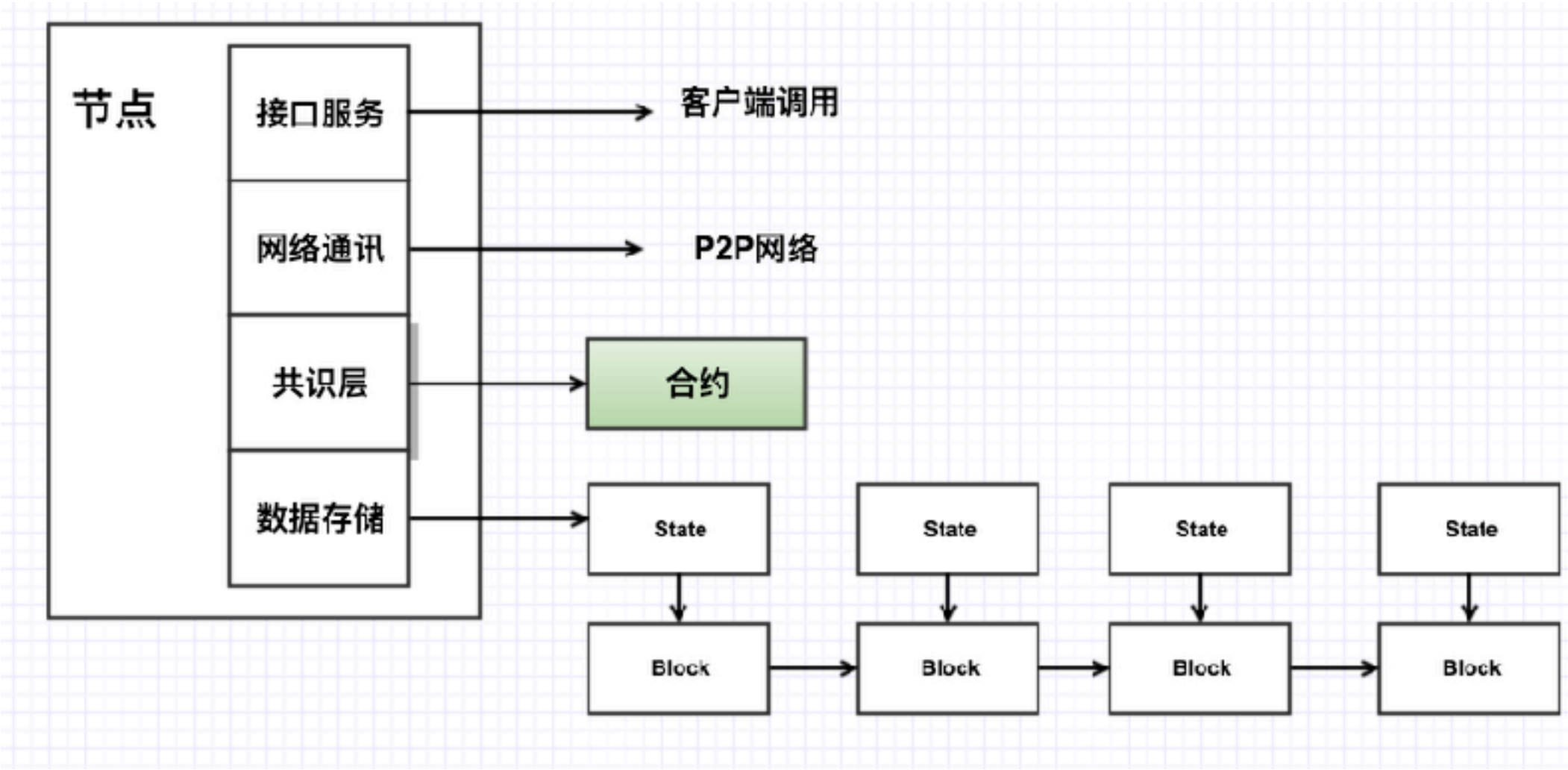
```
type Reward interface

AccumulateRewards(config *params.ChainConfig, state *state.StateDB, header
*types.Header)
```

```
var BoardBlockReward    *big.Int = big.NewInt(5e+18) // 每一个witness产块的奖励
//默认产币规则，每一个block奖励5个coin给予见证者
func (c *DefaultReward)  AccumulateRewards(config *params.ChainConfig, state
*state.StateDB, header *types.Header) {
    reward := new(big.Int).Set(BoardBlockReward)
    state.AddBalance(header.Coinbase, reward)
}
```

实现默认规则

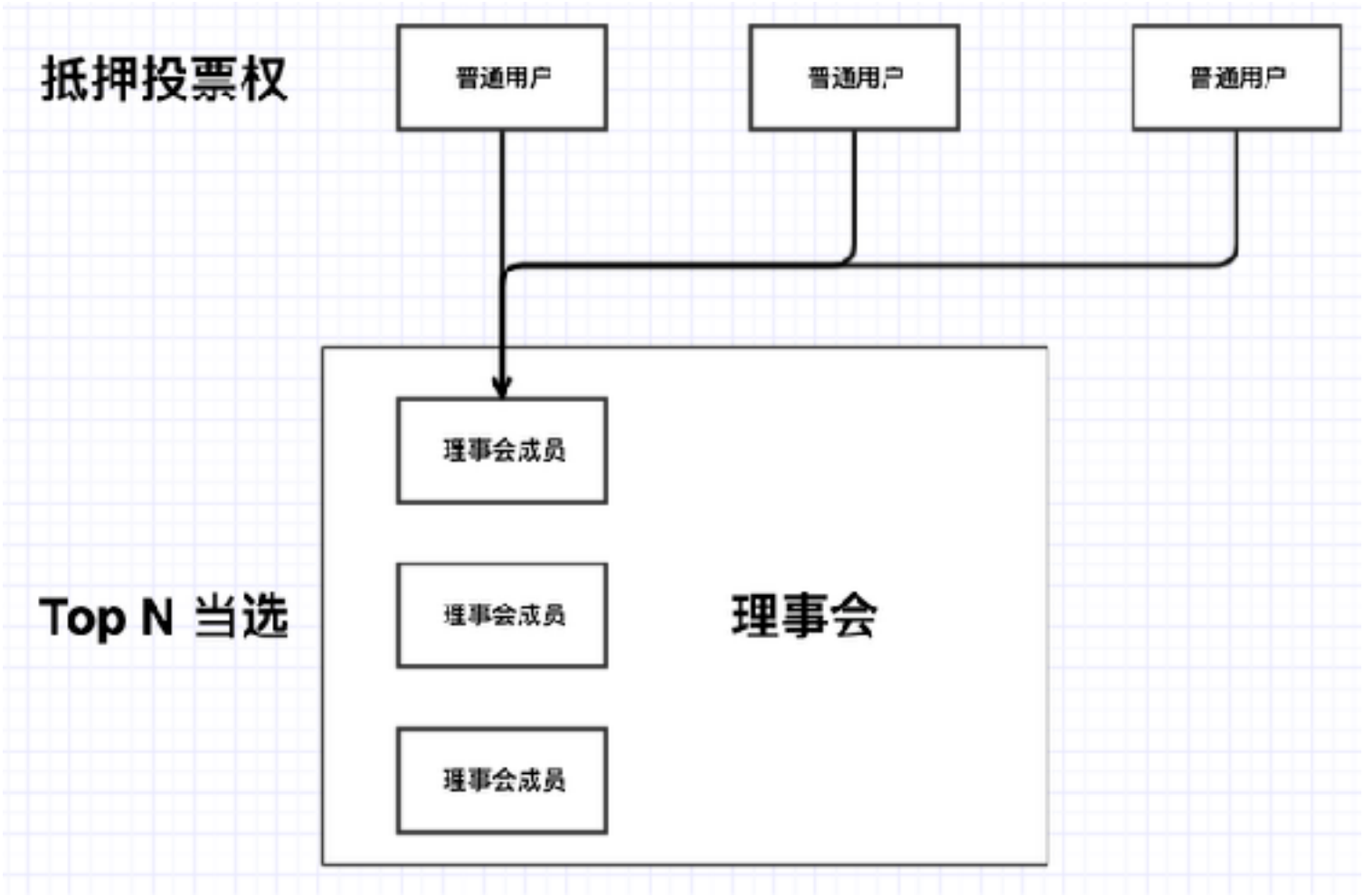
智能合约的实践

合约语言Solidity



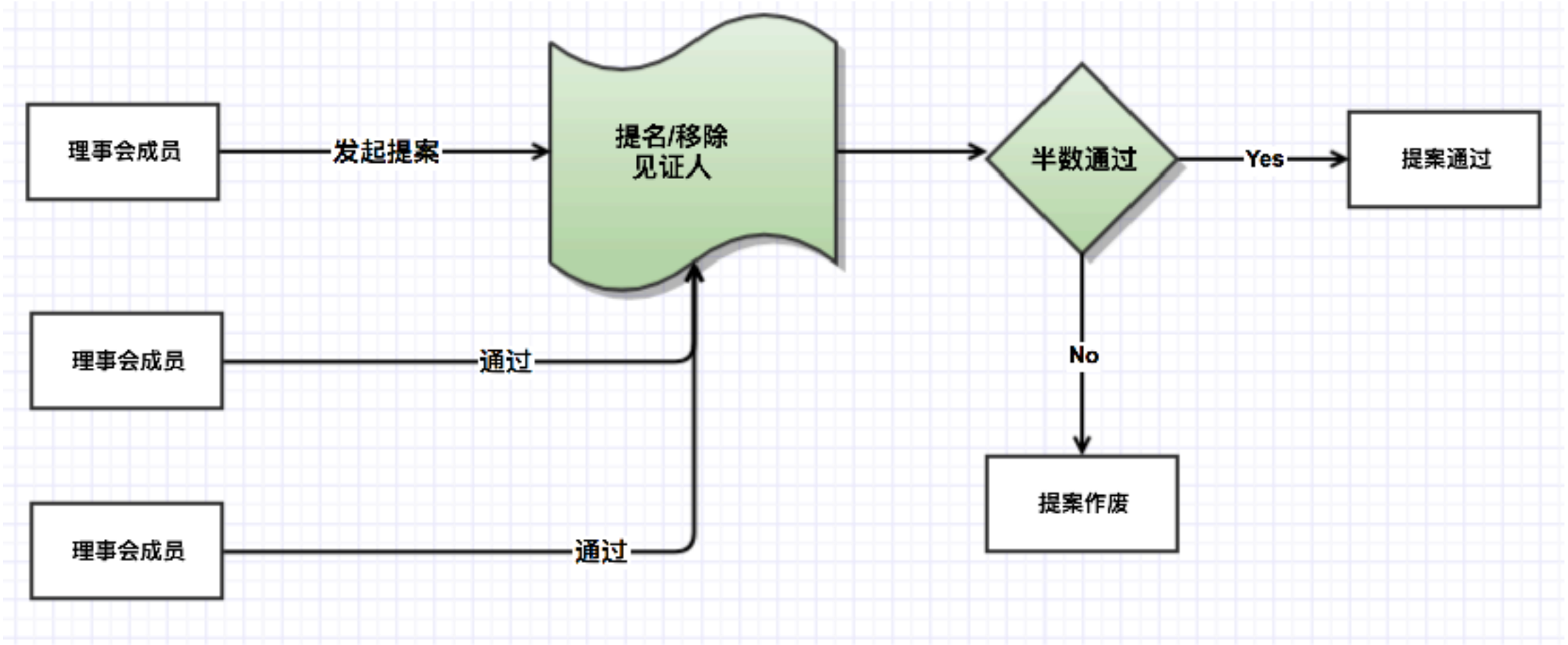
智能合约的实践

实现投票合约 — 理事会



智能合约的实践

实现投票合约 — 见证人



智能合约的实践

智能合约设计模式

- 合约控制器模式设计
- 逻辑层和存储层分离

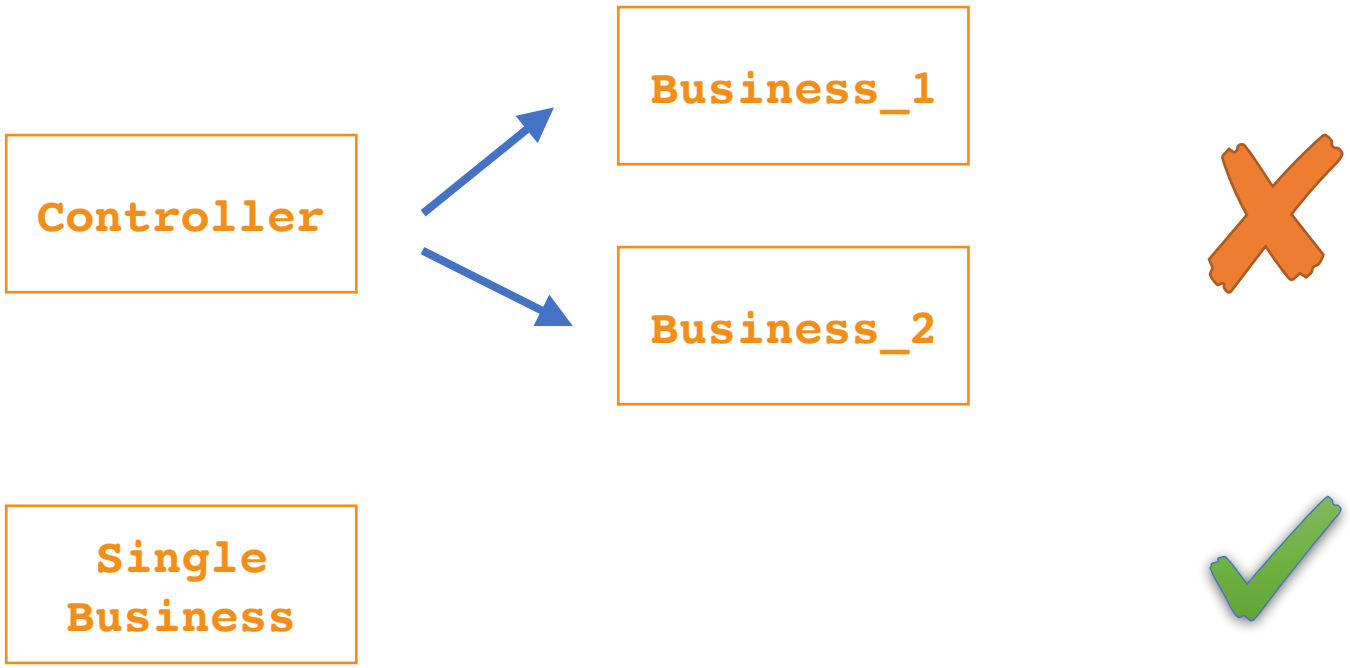
智能合约的实践

智能合约并不智能，反而有太多坑

1. 函数参数不能超过7个
2. 函数参数不能是任何变长对象的数组，例如string[]，uint[][6]
3. 合约之间调用不能传递变长数组，例如uint[],只能传递定长数组，例如uint[6]
4. 二维数组的定义方式与大多数语言相反，例如：uint[][6]表示一个长度为5的uint数组，每个元素又是一个变长数组
5. 只有internal和private的function才能使用结构类型的参数
6. mapping不可遍历，一个可以遍历的版本https://github.com/ethereum/dapp-bin/blob/master/library/iterable_mapping.sol
7. delete不同对象有不同的效果，主要为重置为初始值，而非真正删除该对象，详见https://baijiahao.baidu.com/s?id=1566265348199485&wfr=spider&for=pc
8. 不要在view方法中使用event，这会导致失去view限制，消耗gas，并导致用户调用方式的改变
9. 合约创建的大小不能过大,24k

智能合约的实践

智能合约设计模式, 使用单一合约



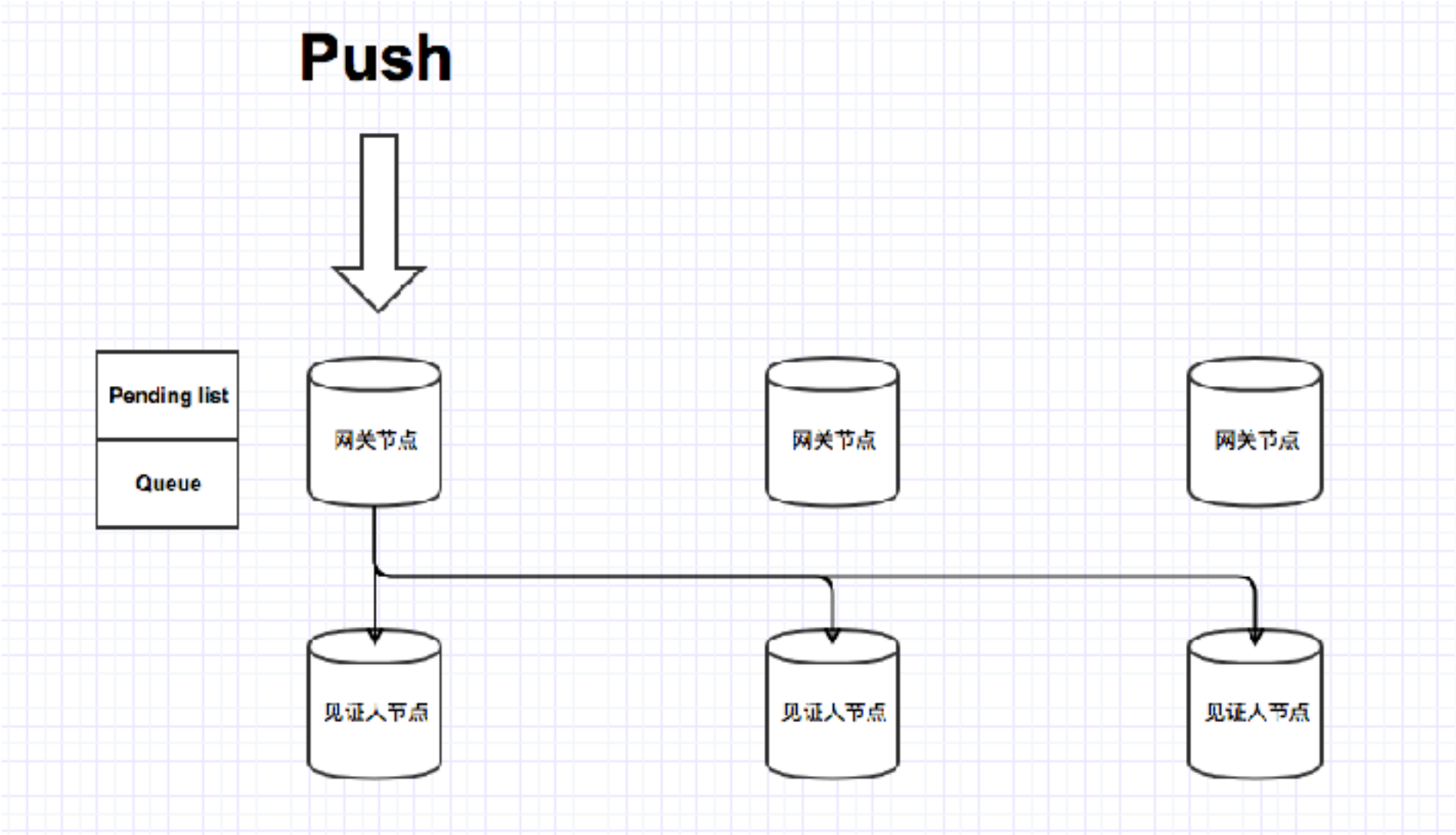
压力测试下暴露的问题

以太坊公链并不会有压力测试的场景，需要大量的优化和测试

问题	原因	解决办法
节点压挂，发生雪崩	节点压力过大，处理变慢，pending池和队列中的数据堆积、内存、cpu资源耗尽，进程kill	增加流量控制、优化网络拓扑、增加pending重发机制
经常出现分叉	数据同步延时，如果没有理事会介入、分叉会持续很久	
节点启动加载慢	堆积了大量交易在transactions.rlp文件，启动时加载没有批量处理	控制文件大小，启动批量加载
大量同步区块数据，内存消耗殆尽	同步的数据块会加入到内存，state状态的更新也会加载到内存，但没有有效的控制	增加物理内存或通过调整Swap区

压力测试下暴露的问题

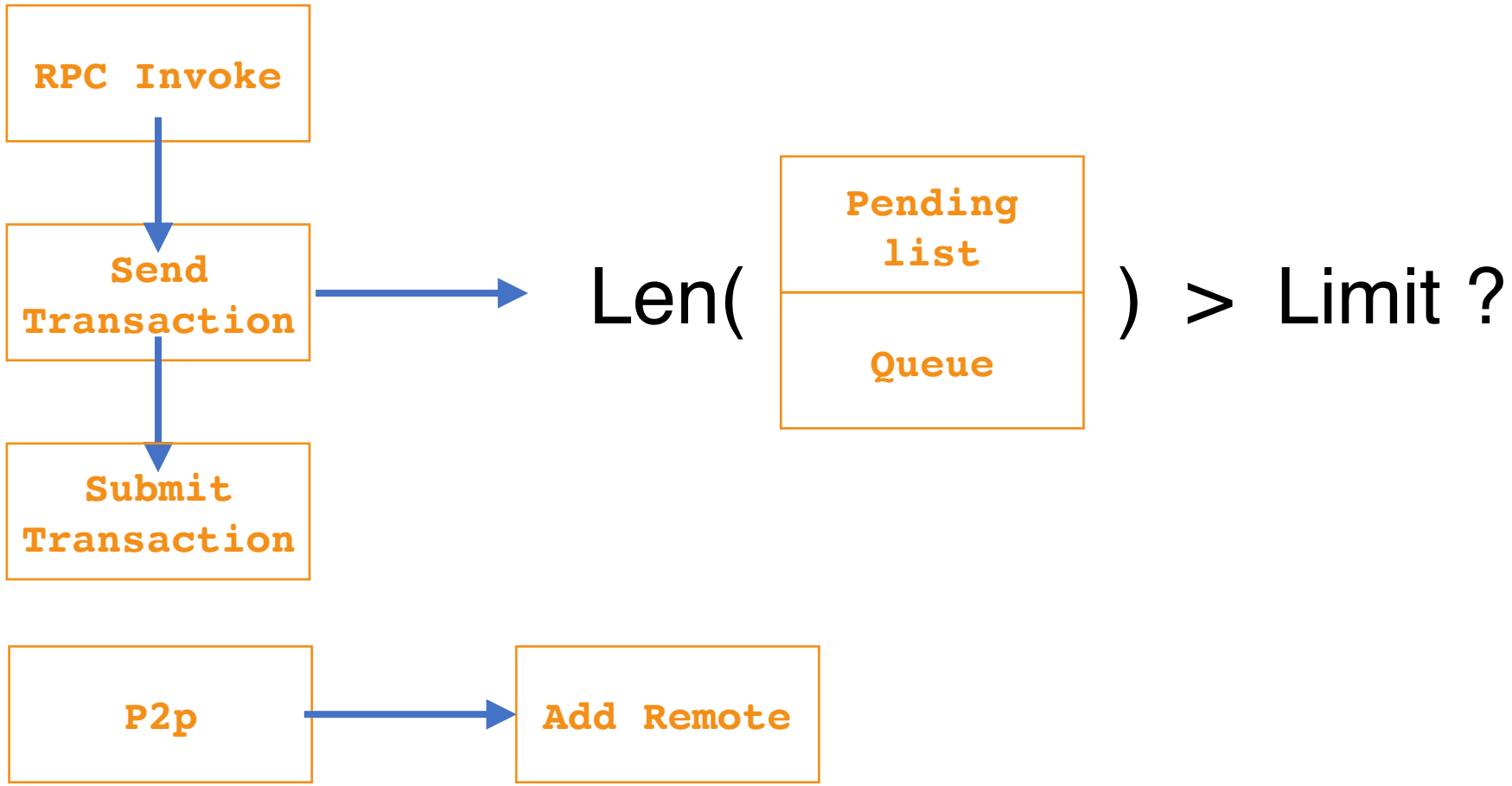
流量控制/重发机制



压力测试下暴露的问题

流量控制—对交易请求进行检查，是否达到阈值上限

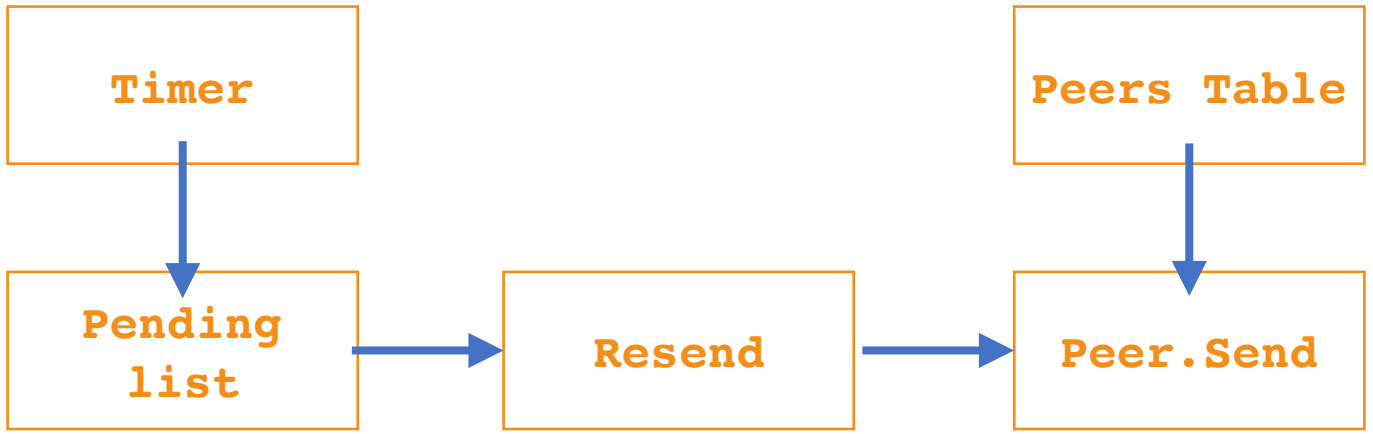
- 只对rpc请求做限制
- 不对p2p广播做限制



压力测试下暴露的问题

重发机制，防止P2P网络交易广播失败

- 广播不会check返回状态
- 节点数量不够压力过大导致交易丢失



GopherChina2018

Q & A