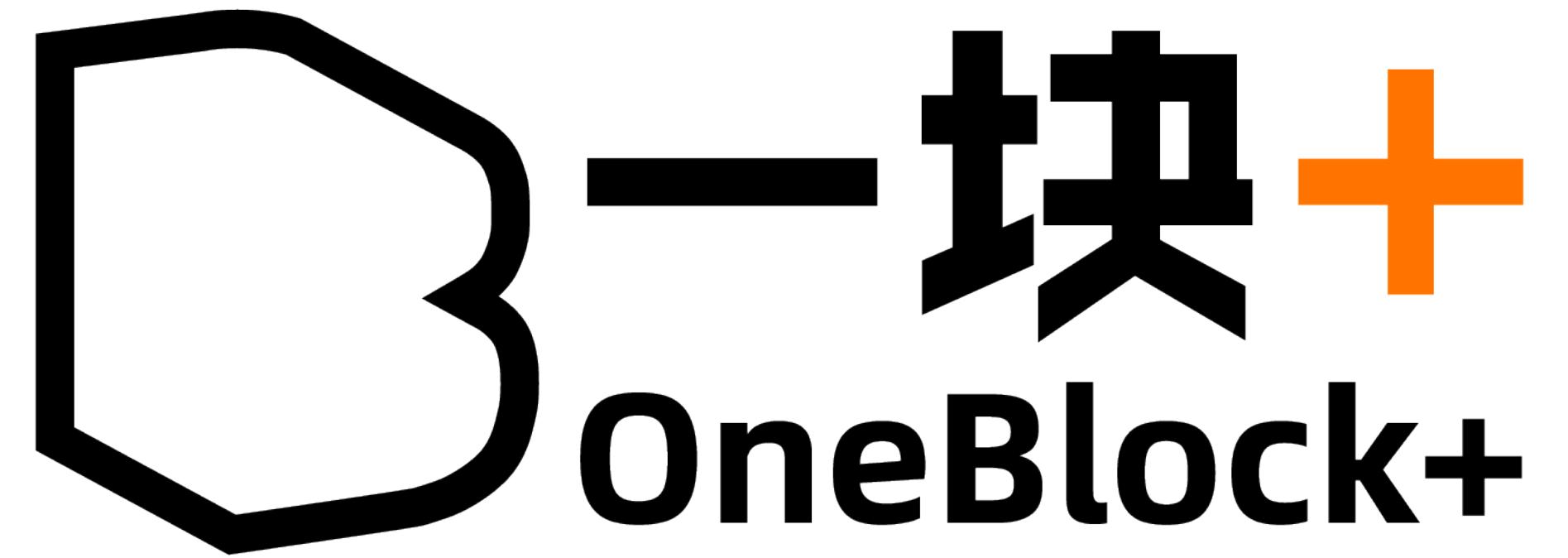


波卡跨链核心技术介绍

Alex 徐杨 | 2020-07-26

感谢



关于作者

- 徐杨
- 拓链科技CTO, HashKey Custody CTO
- <https://github.com/alexxyuyang/substrate-dex>
- <https://github.com/alexxyuyang/RSA-rust>



Alex 徐杨

上海 黄浦



扫一扫上面的二维码图案，加我微信

说明

本文中所有带下划线的文字，都可以链接到代码库，
方便大家结合代码，再去深入理解其中的逻辑

文中所有的内容，均为作者自己的理解，可能有偏差
或错误，欢迎大家沟通并指正

版本说明

- 2020-06-25
- cumulus@05a83e8bdbcf4c216e8632431aa49dd996ce4c3f
- polkadot@91f3e68f5750dff7bddedeb846ff55447e8cf8f8
- substrate@740115371b096af4ca2c3b2c860d95ba801cffcb

为什么要了解波卡跨链机制

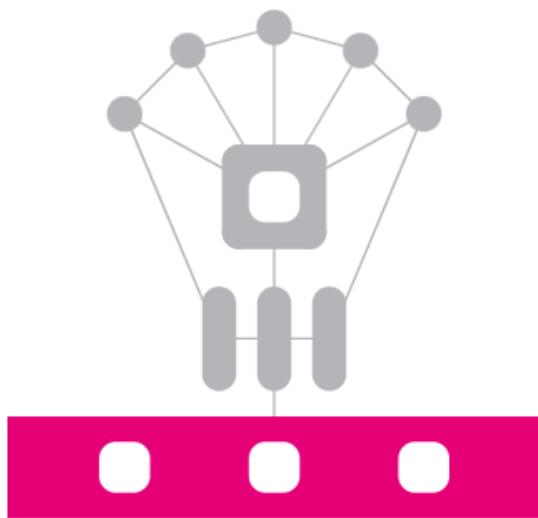
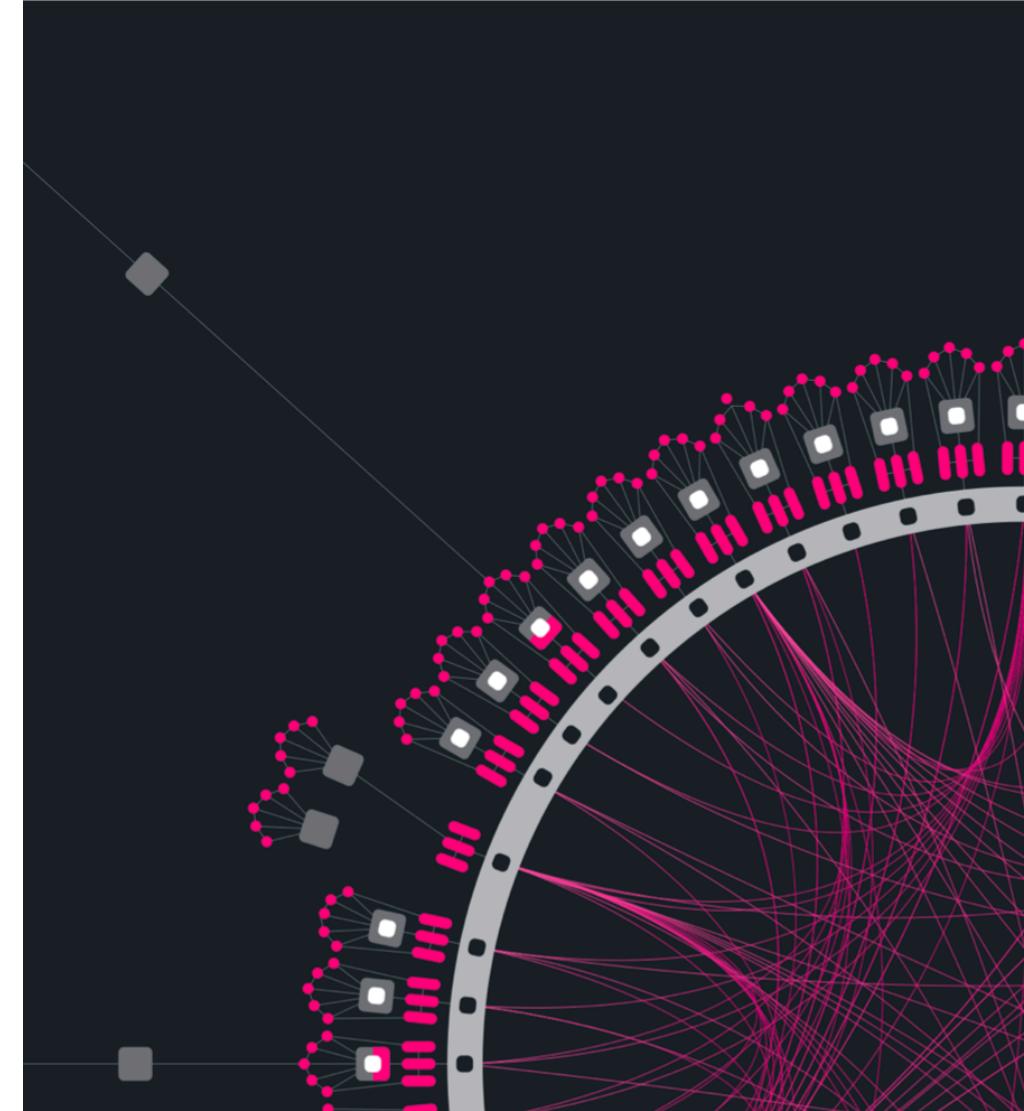
- 对跨链协议本身感兴趣
- 帮助更好的将parachain通过cumulus接入relay chain
- polkadot是学习如何深入使用substrate最好的参考

我们在理解波卡的跨链概念时，会遇到的常见问题是

- 收集人 (collator) , 到底在干些什么? 它和parachain、relay chain的关系是什么?
- 跨链, 到底是什么东西在从parachain跨到relay chain?
- 平均每个链十个validators, 为什么保障安全性?
- 所谓的pooled security, 到底在讲什么?
- 与跨链共识相关的各种数据, 那些上链了? 那些没有上链? 设计的依据是什么?
- 波卡基于substrate而建, 哪些模块适合放到波卡? 哪些适合放到substrate?
- 会什么需要使用纠删码、VRF这些技术?
- 波卡整个生态的tps, 能达到多少? 为什么?

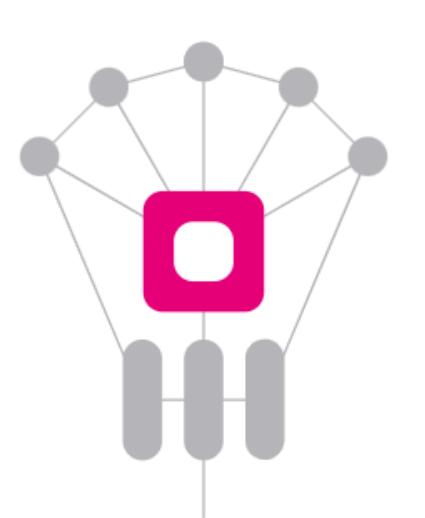


波卡架构



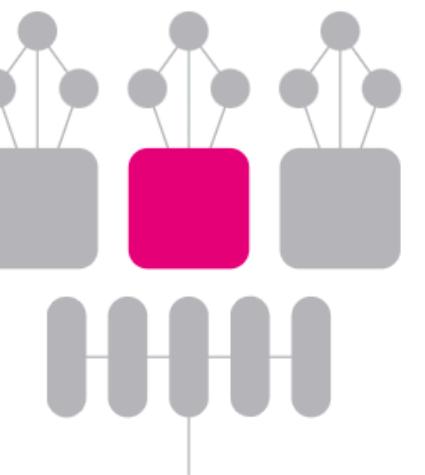
Relay Chain

Polkadot的核心，负责网络的安全性、共识和跨链互操作性。



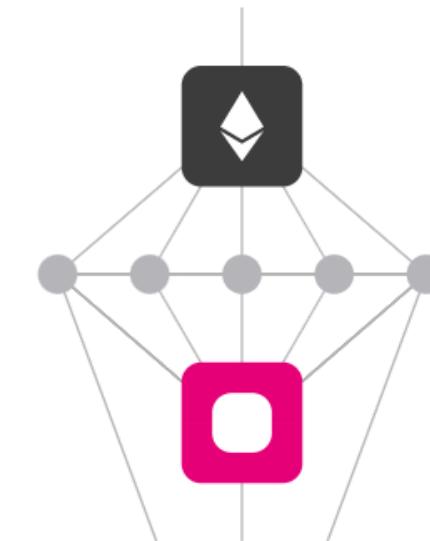
Parachains

主权区块链，可以拥有自己的代币，并针对特定场景优化其功能。



Parathreads

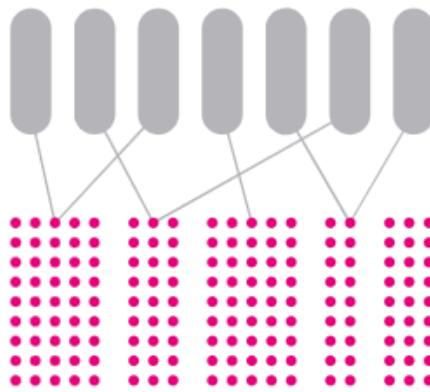
类似于parachains，但采用即用即付的收费模式。对于不需要持续连接网络的区块链来说更加经济。



Bridges

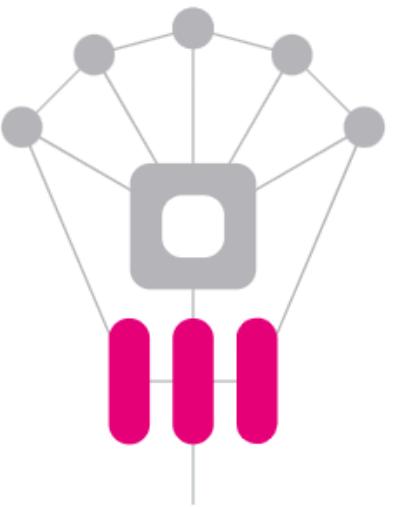
允许parachains和parathreads连接Ethereum和Bitcoin等外部网络并与之通信。

共识参与者



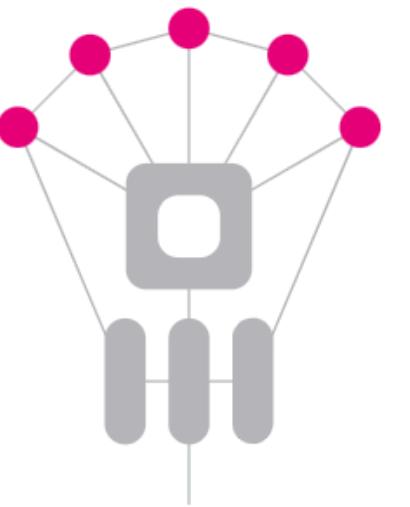
Nominators

提名人，通过选择可信赖的 validators，并抵押dots来保护 Relay Chain。



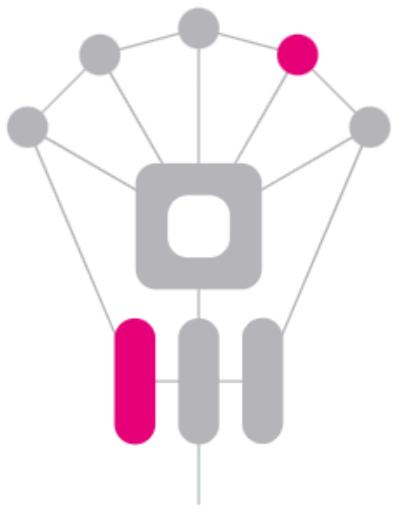
Validators

见证人，通过抵押dots，验证 collators的证据，并与其他 validators达成共识来保护 Relay Chain。



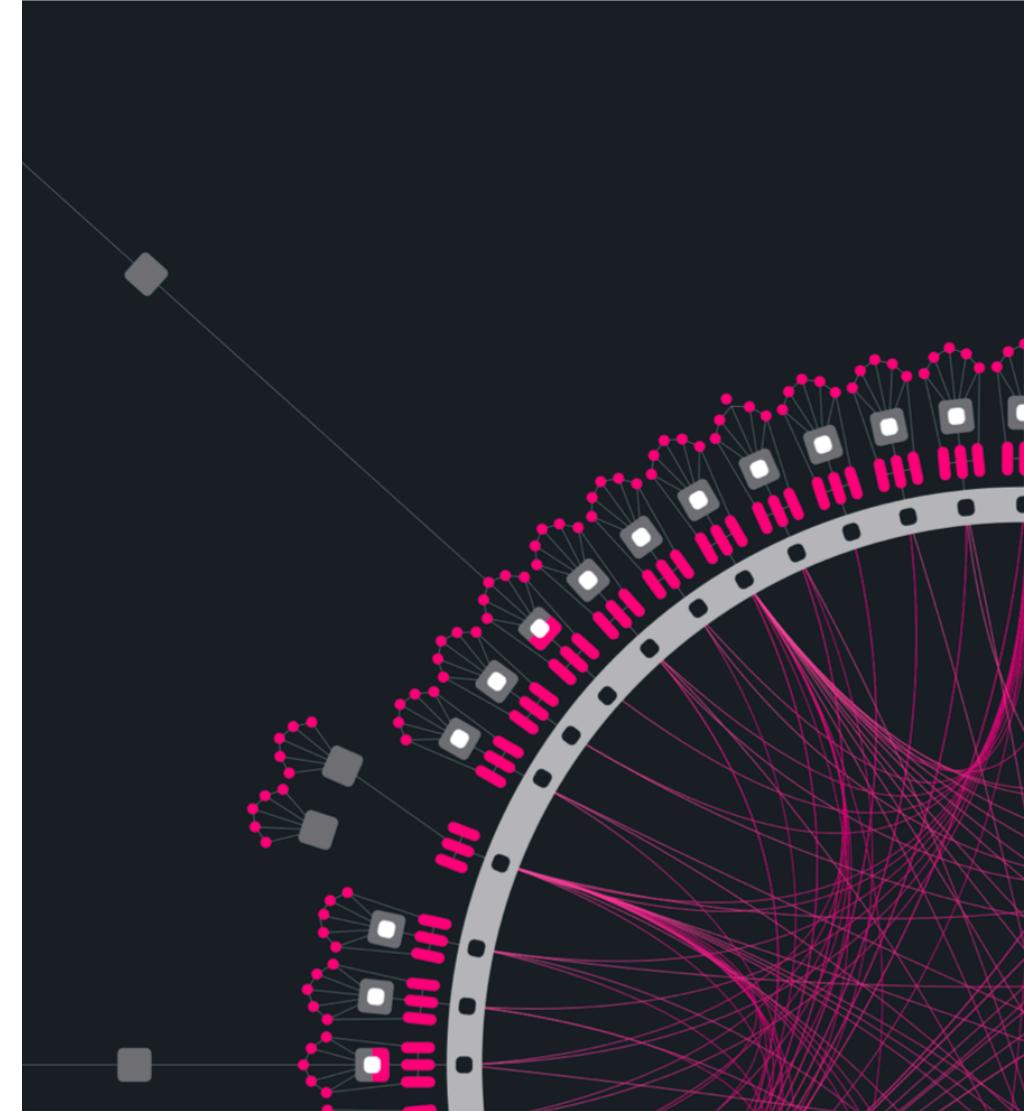
Collators

收集人，通过收集用户的分片交易，并为validator提供证明来维护分片。



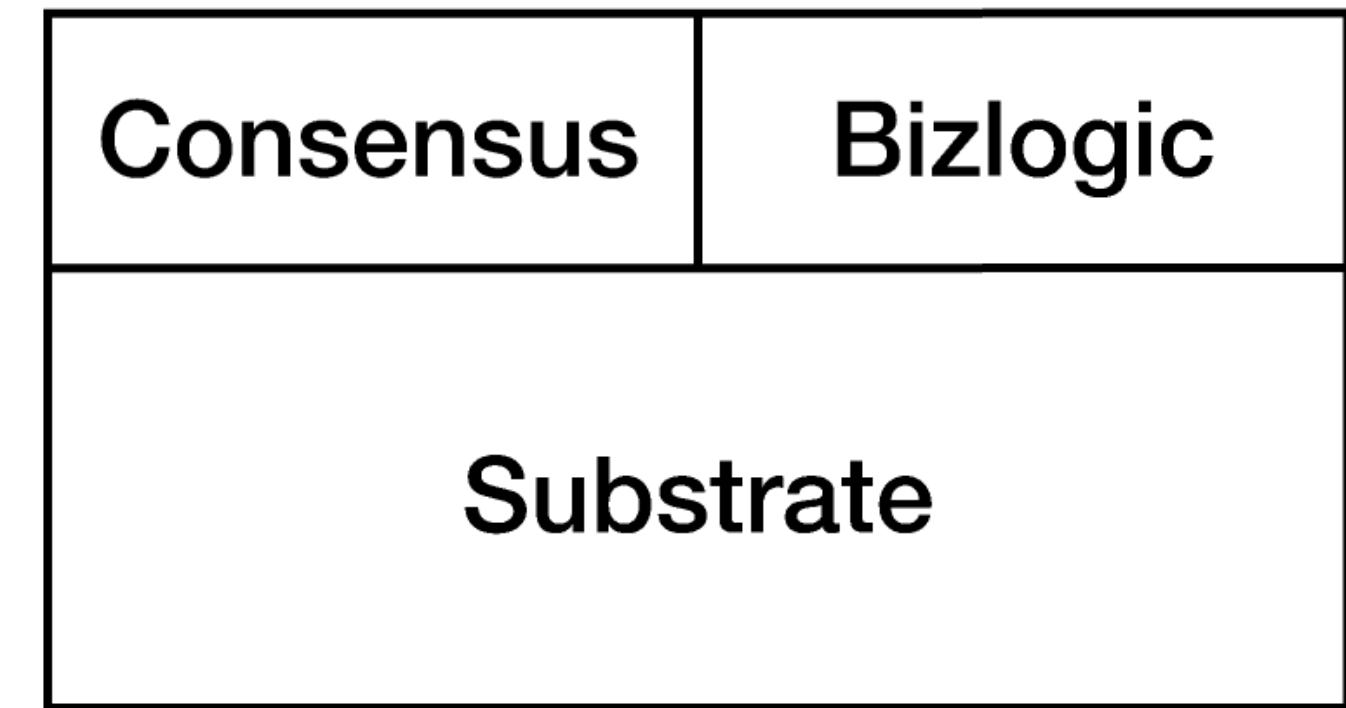
Fishermen

监控网络并将不良行为报告给 validators。Collators和任何 parachain全节点都可以扮演 fisherman的角色。



Polkadot与Substrate的关系

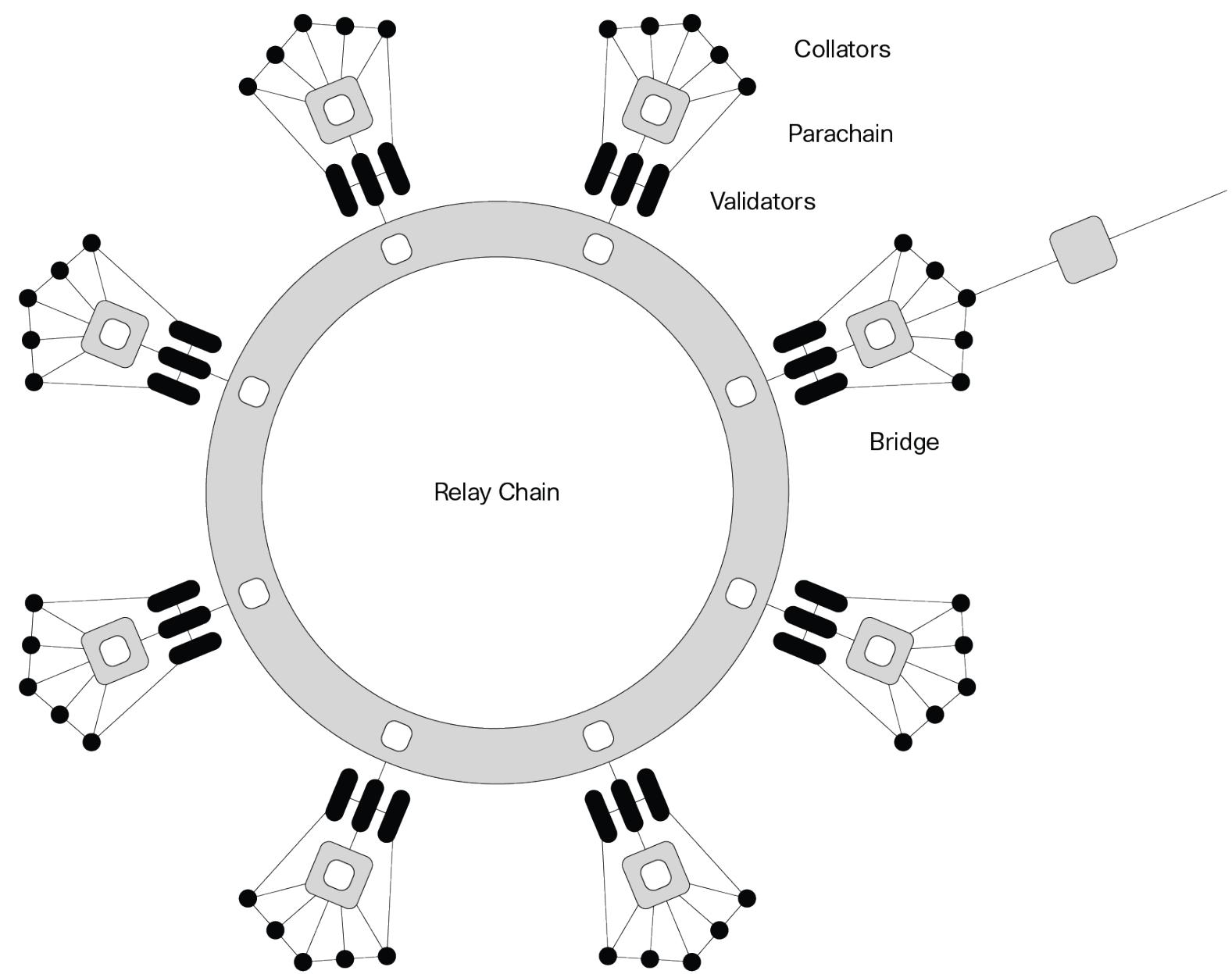
- 为了让生态伙伴快速搭建parachain, parity将polkadot中generic的逻辑抽象出来，形成了substrate
- 基于substrate之上，再去构建polkadot
- 在波卡的代码库中，会有两大定制化的逻辑代码
- 一是共识：collation、collator、candidate receipt、commitment、executor、chunks、gossip、network protocol、availability store等
- 二是业务逻辑：parachains生命周期管理、执行管理、募资管理、众筹等



= Polkadot

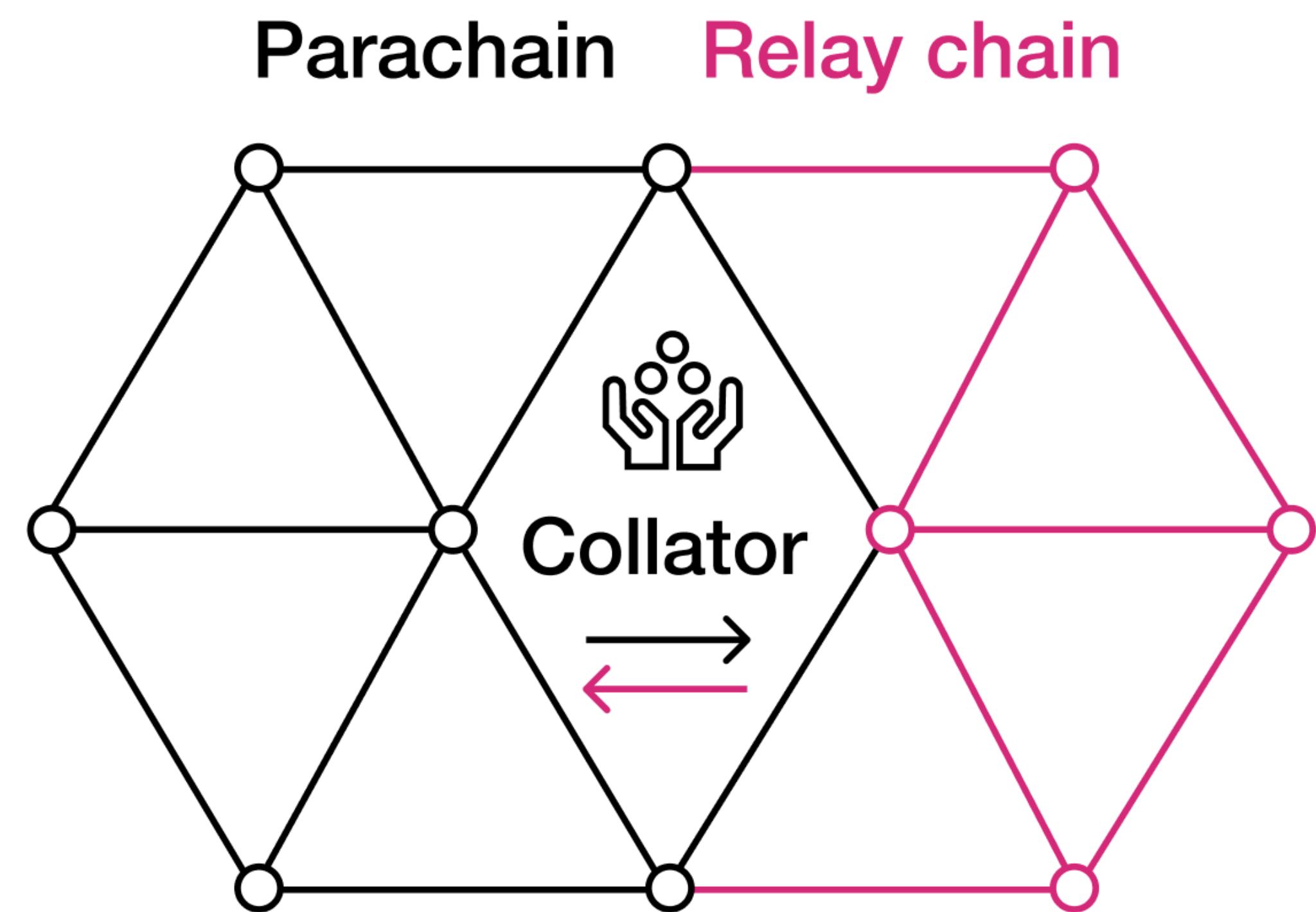
跨链协议流程 Availability and Validity

- 1. 平行链 (parachains) 阶段
- 2. 中继链 (relay-chain) 提交阶段
- 3. 可用性子协议
- 4. 在 GRANDPA 机制中执行二次检查
- 5. 钓鱼人 (fisherman) 的异议程序
- 6. 调用拜占庭容错的确定性小工具 (finality gadget) 让链最终确定



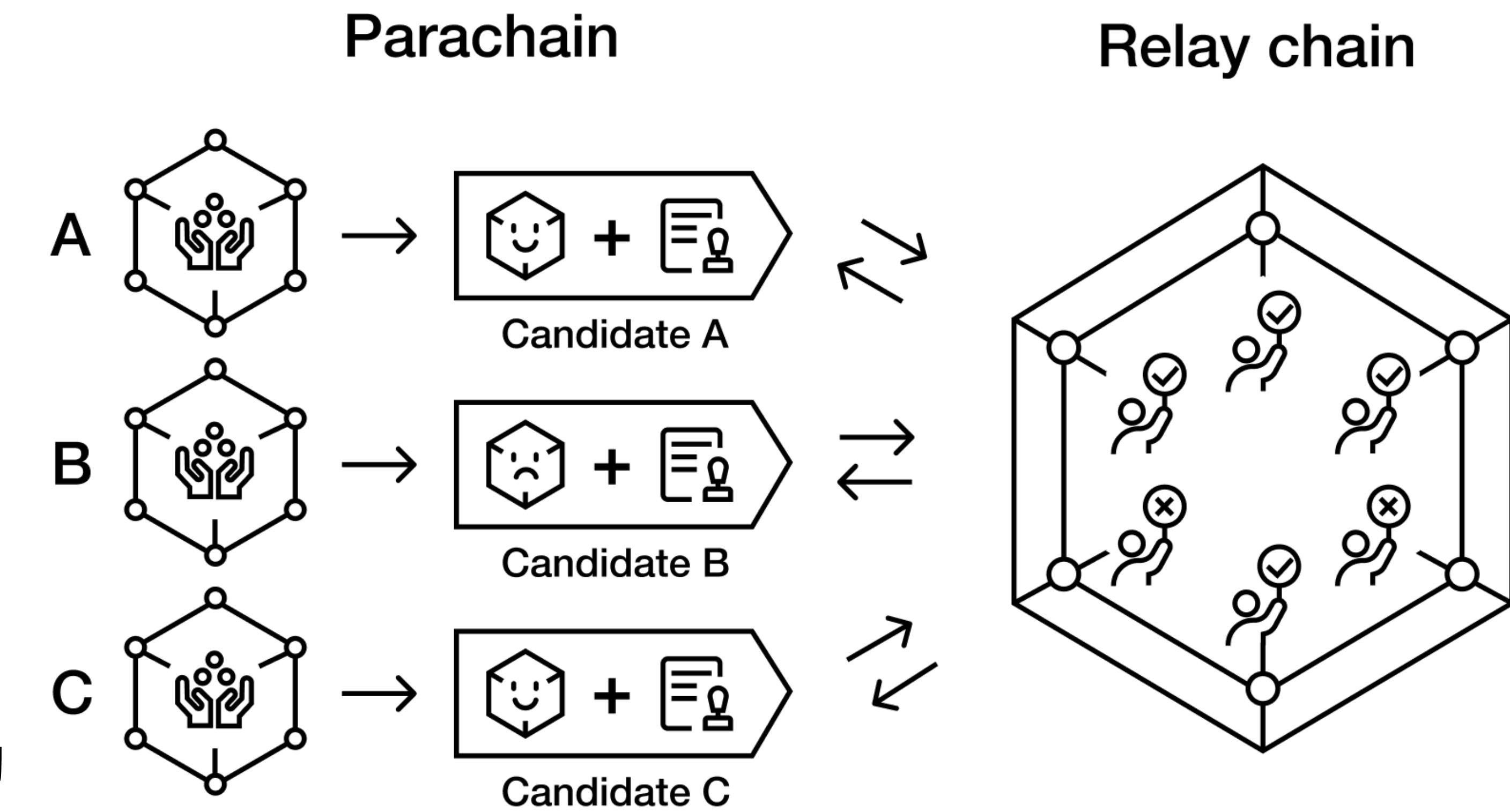
1. 平行链阶段 - 网络架构

- 收集人是一个独立的程序，它进入了两个链的p2p网络，一是parachain，二是relay chain
- 收集人是将信息从parachain摆渡到relay chain的中间节点



1. 平行链阶段 - 整体流程

- 平行链阶段是指平行链的收集人 (collator) 向当前分配给该平行链的一组验证人 (validators) 提出一个候选区块
- 收集人节点活跃在一个独特的平行链上，并提交状态转换的建议以及其有效性证明 validity proof
- 候选区块是指平行链的收集人 (collator) 提出的新区块，它可能是有效的，也可能是无效的
- 在被纳入中继链之前必须经过relay chain的 validators的有效性检查



1. 平行链阶段 - collator激励

平行链只需要一个诚实的收集人来提交区块

从relay chain的角度看，不需要收集人抵押dot

collator做为收集人没有收益

对收集人的经济奖励（如果有），需要在parachain上去实现

collator也可以做fisherman，这样也可以有相应收益

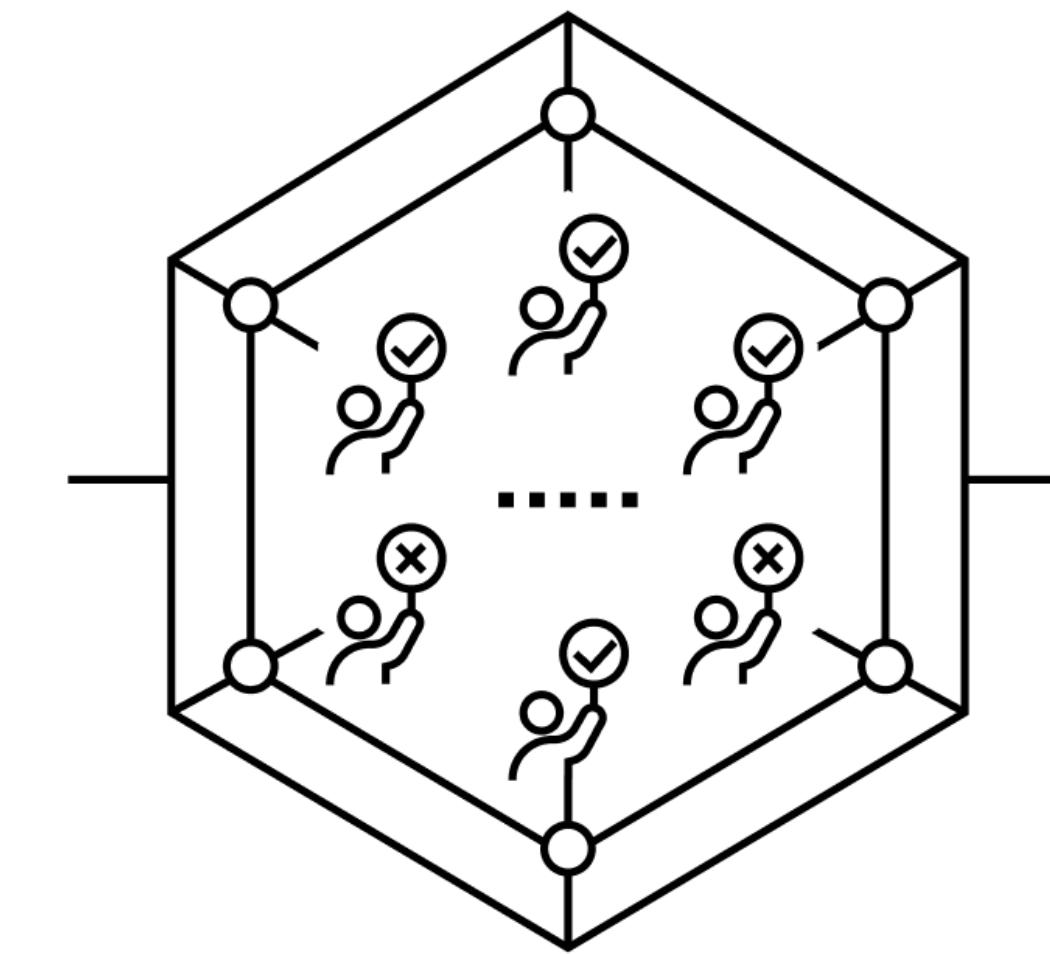
1. 平行链阶段 - 测试方法

在polkadot的代码仓库中，有一个adder模块，用来模拟一个最简单的parachain

在实际使用过程中，parachain需要参考cumulus代码库，将其接入relay chain

2. 中继链阶段 - slot

- collator生成了候选区块后，通过p2p网络，将其传递给relay chain的validators
- 在relay chain的每个slot中，会对每个parachain分配若干个validators
- validators per parachain = (validator count - 1) / parachain count
- 它们会负责验证候选区块 (parablock)



2. 中继链阶段 - duty_roaster

- 每个parachain分配的validators
- 基于链上的一个随机数（基于之前81个区块的数据生成）
- 再按照右侧的这个shuffle算法
- 进行乱序分配的，每个slot分配的 validators都是随机的
- 这里并没有使用VRF，和[这个文章](#)的描述是有出入的

```
1058     let mut roles_val : Vec<Chain> = (0..validator_count).map(|i| match i {  
1059         i if i < parachain_count * validators_per_parachain => {  
1060             let idx : usize = i / validators_per_parachain;  
1061             Chain::Parachain(parachains[idx].clone())  
1062         }  
1063         _ => Chain::Relay,  
1064     }).collect::<Vec<_>>();  
  
1086     // shuffle  
1087     for i : i32 in 0..(validator_count.saturating_sub(1)) {  
1088         // 4 bytes of entropy used per cycle, 32 bytes entropy per hash  
1089         let offset : usize = (i * 4 % 32) as usize;  
1090  
1091         // number of roles remaining to select from.  
1092         let remaining : usize = sp_std::cmp::max(v1: 1, v2: (validator_count - i) as usize);  
1093  
1094         // 8 32-bit ints per 256-bit seed.  
1095         let val_index : usize = u32::decode(input: &mut &seed[offset..offset + 4])  
1096             .expect(msg: "using 4 bytes for a 32-bit quantity") as usize % remaining;  
1097  
1098         if offset == 28 {  
1099             // into the last 4 bytes - rehash to gather new entropy  
1100             seed = BlakeTwo56::hash(s: seed.as_ref());  
1101         }  
1102  
1103         // exchange last item with randomly chosen first.  
1104         roles_val.swap(a: remaining - 1, b: val_index);  
1105     }
```

2. 中继链阶段 - VRF

- VRF (Verifiable Random Function), babe中确定出块validator的算法
- hash函数: $\text{result} = \text{hash}(\text{info})$
- $\text{result} = \text{hash}(\text{SK}, \text{info})$
- $\text{result} = \text{vrf_hash}(\text{SK}, \text{info})$
- $\text{proof} = \text{vrf_proof}(\text{SK}, \text{info})$
- $\text{vrf_verify}(\text{PK}, \text{info}, \text{result}, \text{proof}) \rightarrow \text{T/F}$
- 其中, SK/PK是validator的key pair
- 这样, 既可以验证info与result的关联关系, 也可以知道是哪个validator签署的

2. 中继链阶段 - VRF在波卡中的应用

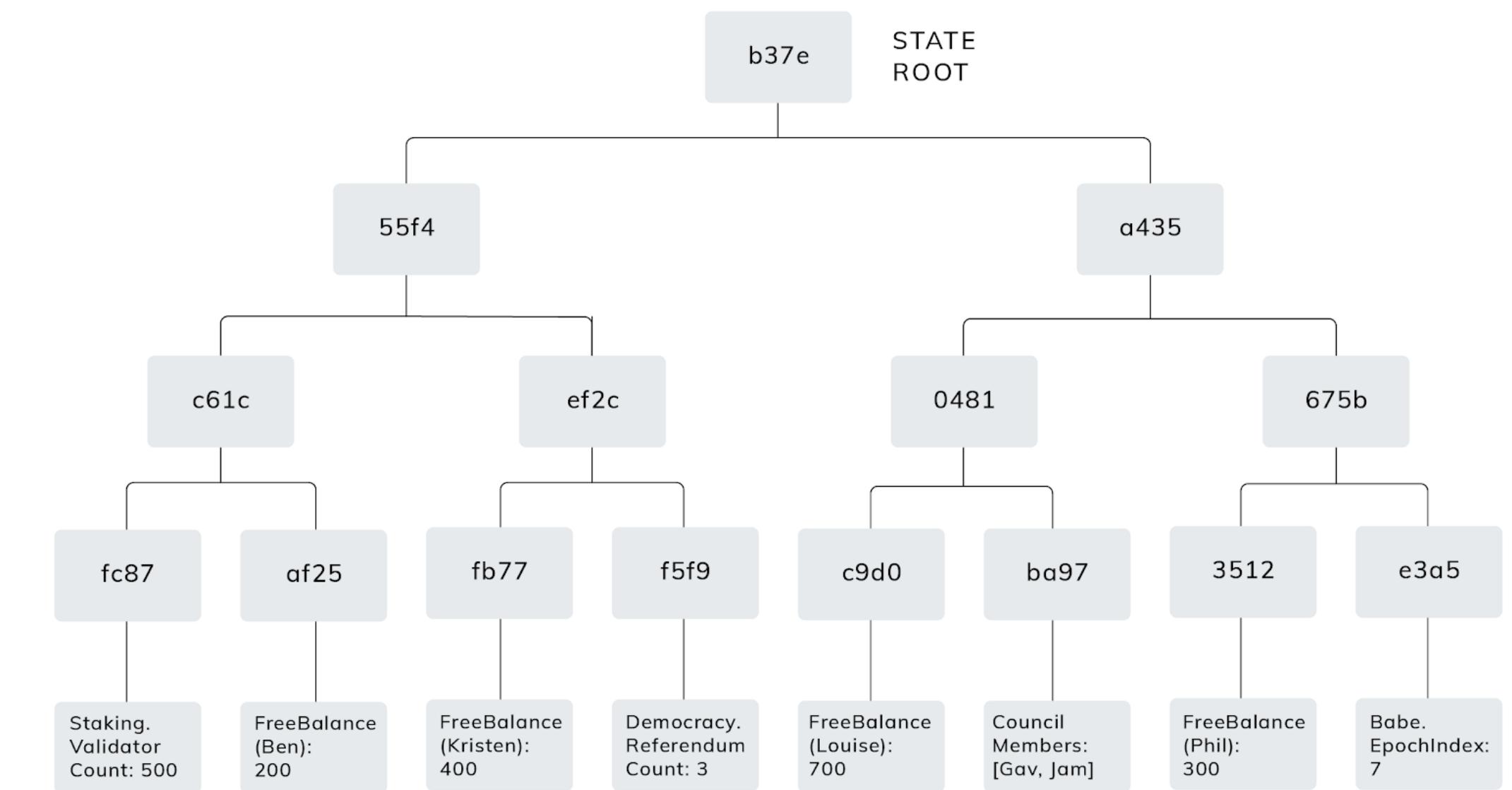
- `result = hash(SK, info)`
- `result = vrf_hash(SK, info)`
- `proof = vrf_proof(SK, info)`
- `vrf_verify(PK, info, result, proof) -> T/F`
- `result`是经过vrf计算后的结果，在polkadot中，`result`需要小于一个阈值才算抽中签，参考[代码1](#)、[代码2](#)
- `info`的值，在每个slot中，是一个全网固定的值
- 在polkadot中，`info`称作[transcript](#)，是将：一个随机数（每个epoch中固定，将N-2个epoch前的随机数拼接的Hash，[代码1](#)、[代码2](#)）、slot number、epoch index，这三个元素拼在一起
- 因为validator是否选中来出块，是根据vrf计算而定的，所以一个slot，会有一个、多个、或零个节点在vrf过程中胜出
 - 一个：正常情况（primary slot leaders）
 - 多个：多个validators出块，最后，需要grandpa来确定哪个分叉为最终链
 - 零个：使用round robin确定一个出块节点（secondary slot leaders）

2. 中继链阶段

- 1 接收candidate block
- 2 使用STF(State Transition Function)验证状态转移是否正确
- 3 广播验证结果commitment & block给其它validators
- 4 一旦超过一半的validators认可，便准备candidate receipt
- 5 使用纠删码技术保存block数据
- 6 发送交易到节点交易池
- 7 打包节点将candidate receipt打包进入区块

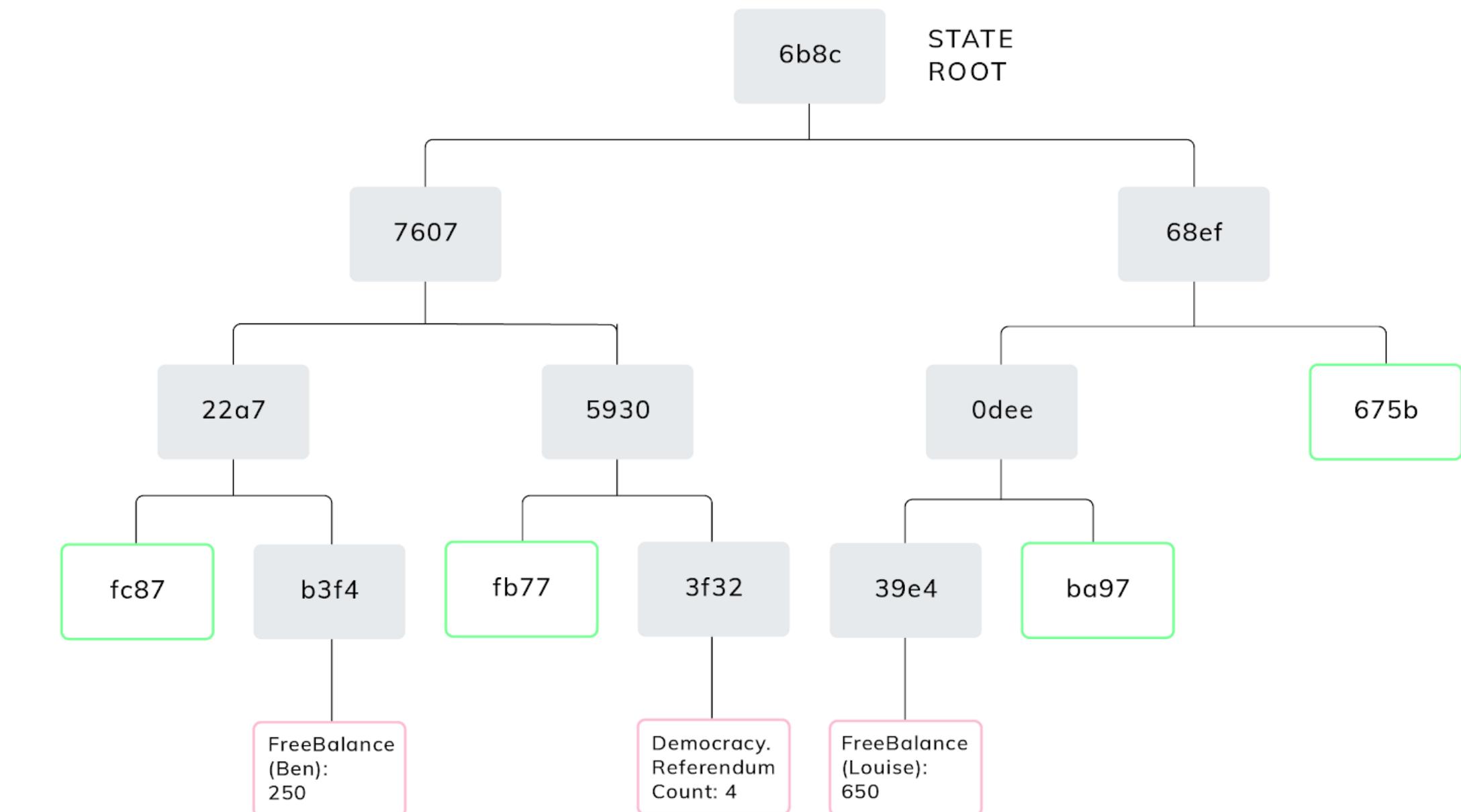
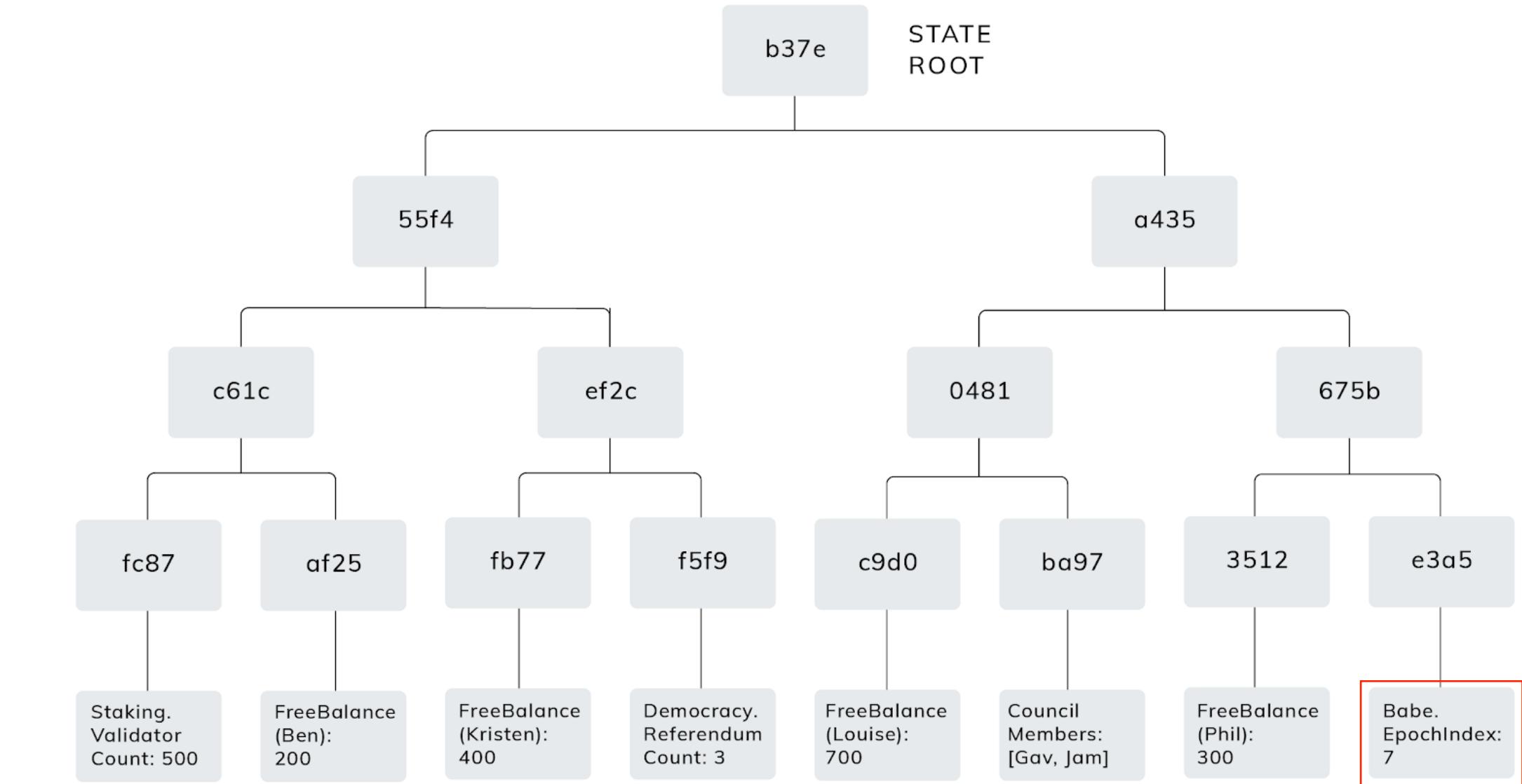
2.2 STF - 默克尔树

- 平行链的状态保存在一个称为默克尔树的结构中
- 所有的状态都是该树的叶子
- 例如， Phil - FreeBalance: 300
- 相邻的两个节点取其hash值，以此递归往上
- 直到状态根 (state root) - b37e



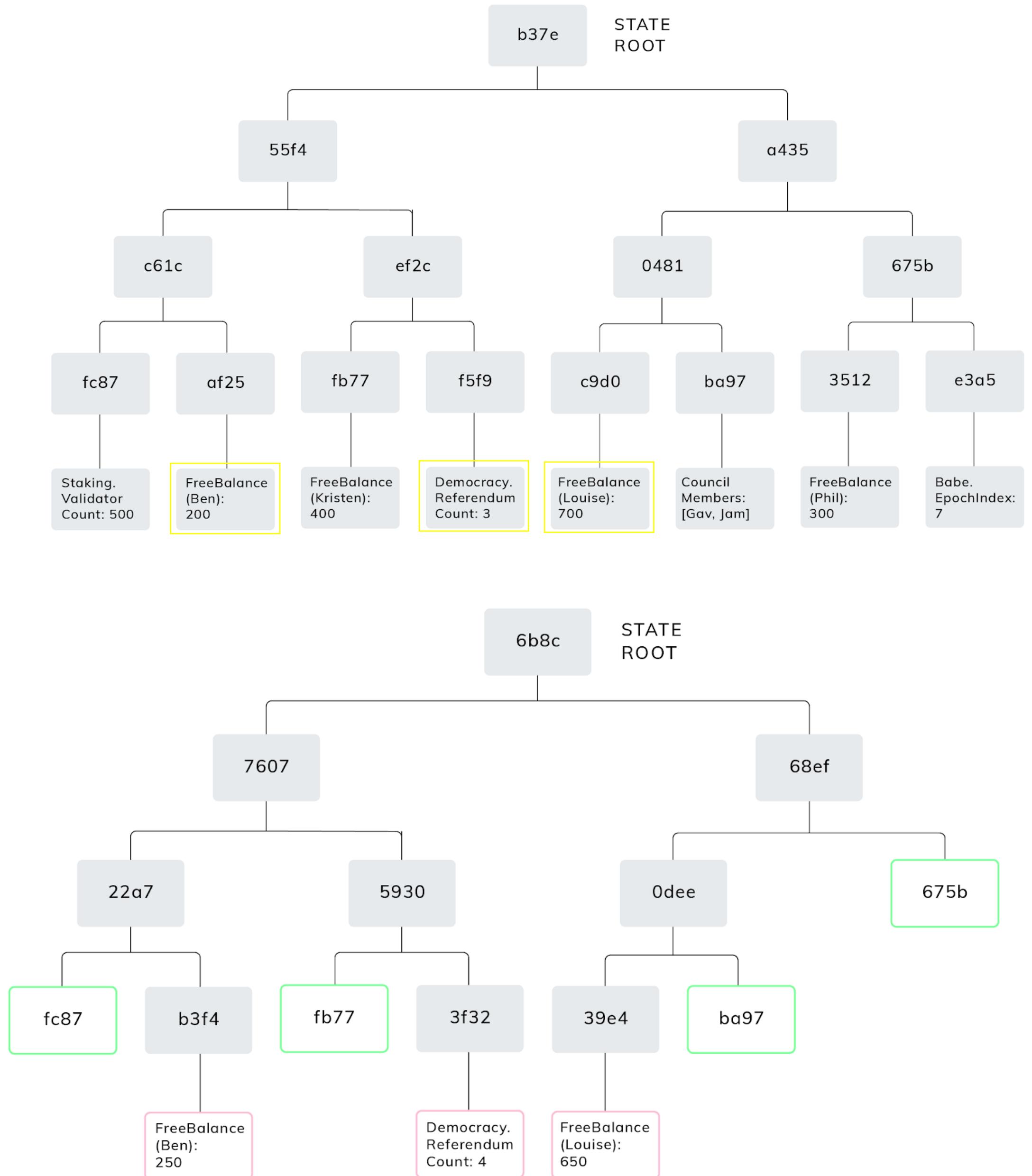
2.2 STF - 验证

- 默克尔树具有一种方便的属性：如果某些值发生更改，则仅查看新修改的值，以及该树中未受影响数据根Hash，即可验证更改
- $\text{Hash}(A_1, A_2 \dots, A_8) \rightarrow \text{Hash}(A_1, A_2 \dots, A_7, B_8)$
- A_8 变为 B_8 , 其它不变，则不需要7个hash, 而是3个, $\log(n)$
- 基于此属性，验证程序可以验证状态转换，而无需访问整个状态。它只需要：
 - 该区块所修改的平行链数据库中的值
 - Merkle树中未受影响数据根哈希
- 这套信息构成了有效性的证明，哈希值具有固定的长度
- 未修改的值有多大都没有关系，哈希足以表示它们



2.2 STF - 验证过程

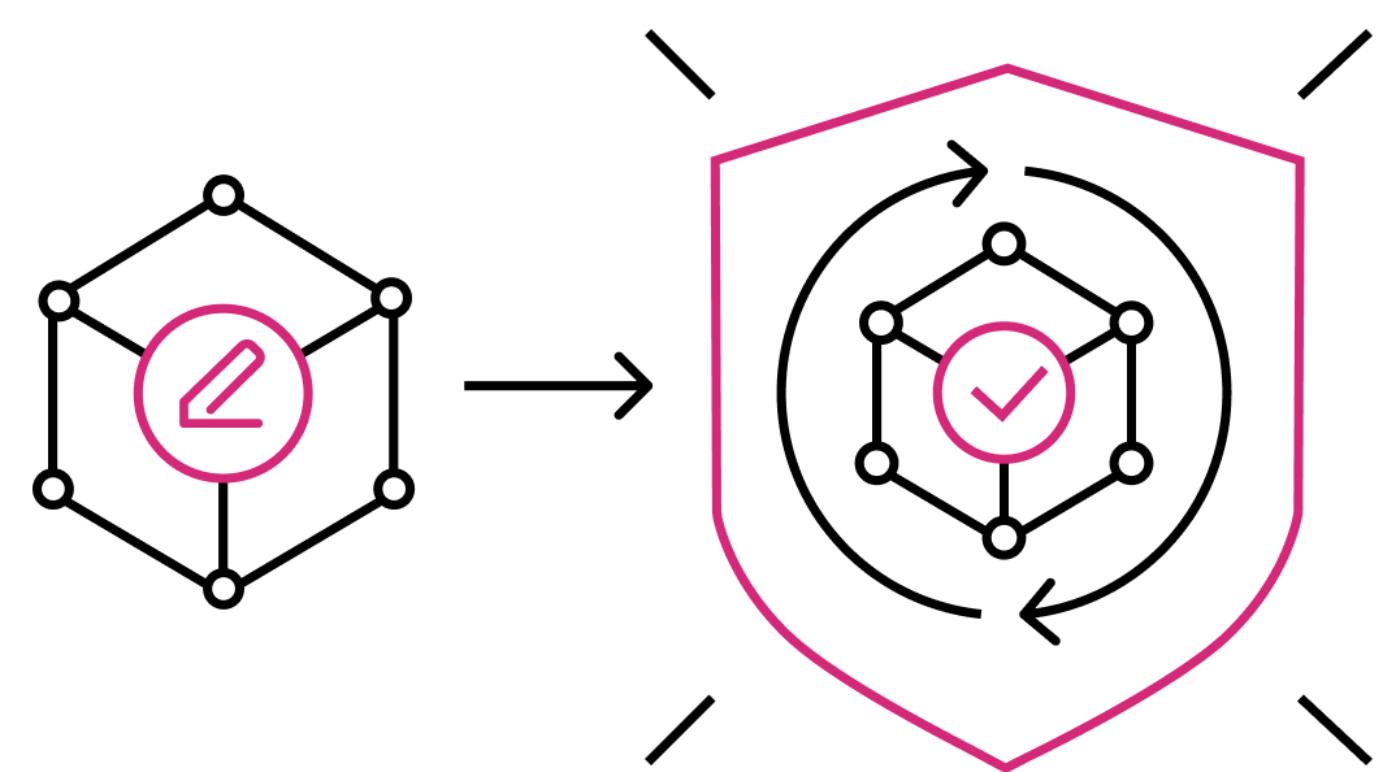
- 我们来看看relay chain的验证的具体过程
- collator提供以下数据：
 - 1 区块（状态转换列表）
 - 2 该区块所修改的平行链数据库中的值，黄框的内容
 - 3 Merkle树中未受影响数据根哈希，绿框的内容
- 其中1是candidate_block, 2+3是validity proof
- relay chain的validator，基于2、3，可以计算出parent block的状态根 S1
- relay chain的validator，使用parachain的STF wasm镜像，基于修改的状态数据 - 2，执行区块交易 - 1，保证只修改了“2”中的内容，同时得到新的值，红框的内容 - 4
- 基于3、4计算当前block的状态根 S2
- 将 (S1, S2) 与collator传送过来的数据进行对比，相同则表示验证成功



2.2 trust free shared security

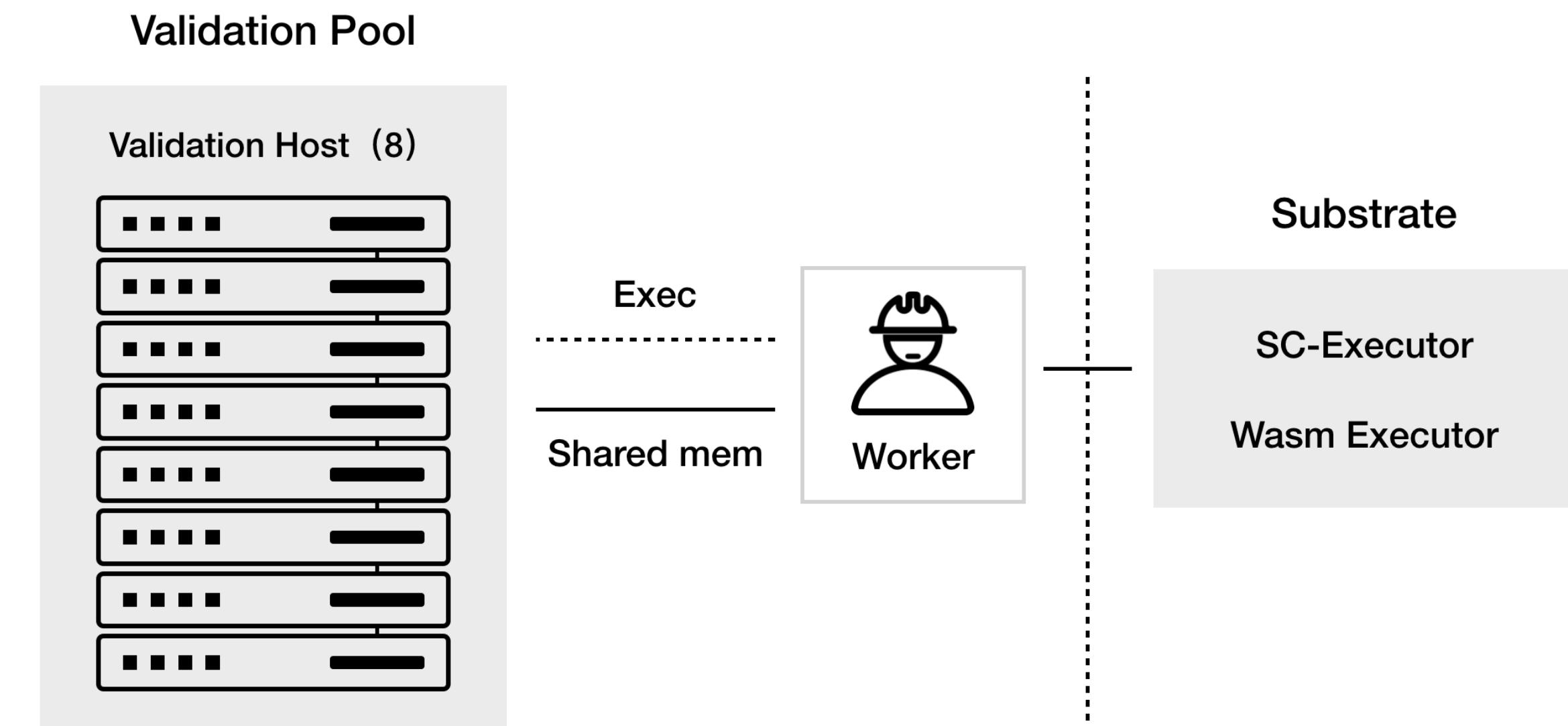
- Polkadot验证程序不会检查平行链状态中的每个值
- 通过执行区块，会检查已修改的值，以确保修改有效
- 如果一条链以有效状态加入Polkadot网络并在Polkadot安全性的保护下执行其所有转换，则它将具有有效状态
- Polkadot不保证有效状态，它保证了有效的状态转换
- Parathreads
- 无信任
- 共享安全

Polkadot



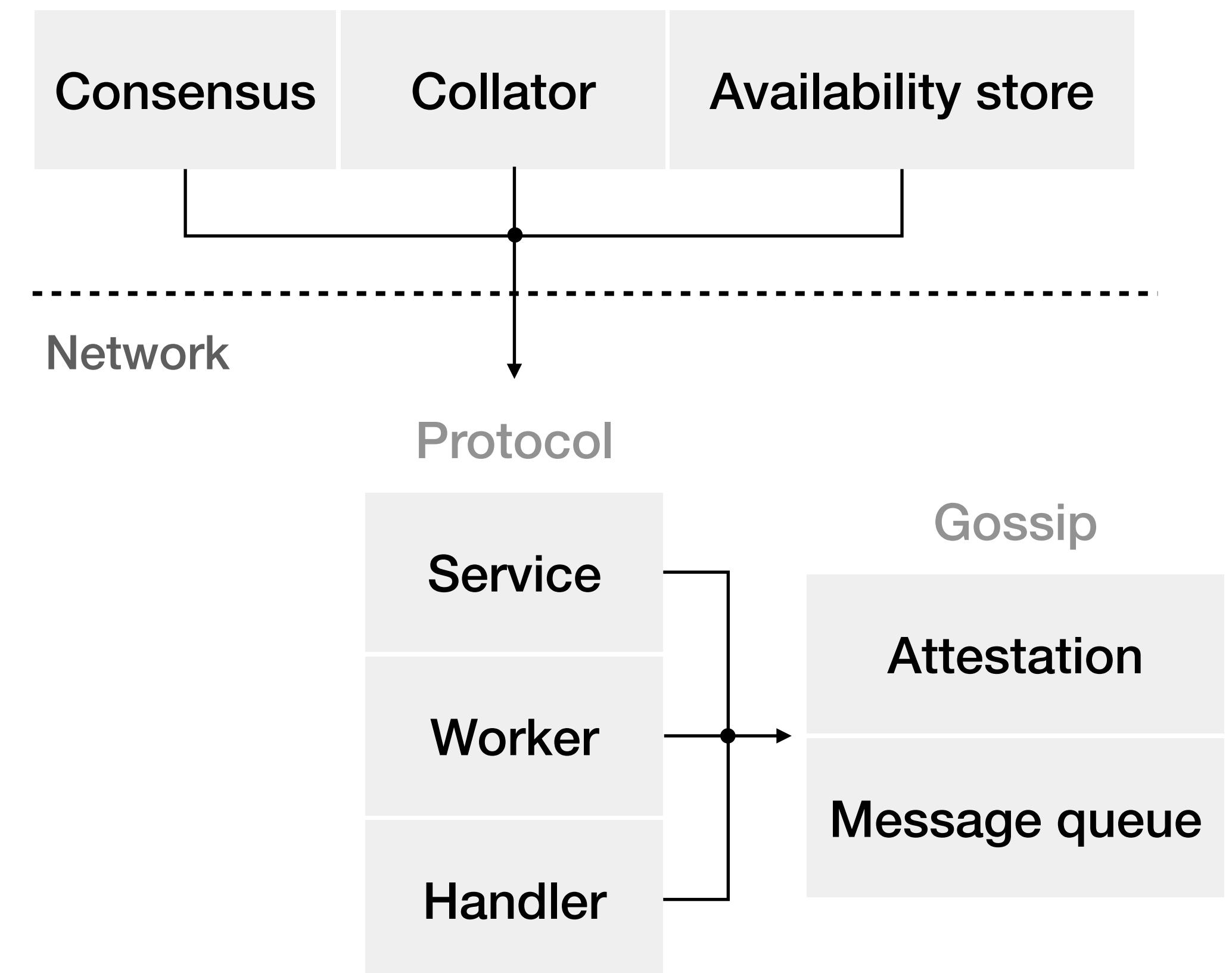
2.2 STF - executor

- validator最终使用executor验证器，来做STF验证
- validator上有一个validation pool
- pool中最多启动八个validation host
- 在有验证工作时，host会通过exec创建单独进程worker，
args: validation_worker
- host与worker之间通过shared memory通信
- worker最终调用了substrate->sc-executor->wasm_executor
来做stf的验证



2.3 广播验证结果 - network设计

- 在substrate的network协议之上，波卡基于自身的共识逻辑，构建了一套定制化的网络协议
- 主要由两个组件组成，protocol 和 gossip
- protocol的核心是worker，是所有消息的核心路由
- handler是大多逻辑的具体实现层，供protocol使用
- service包装了protocol，并对外提供各种trait实现
- gossip使用了substrate的gossip engine来发送消息
- gossip实现了两个主要功能
- attestation 和 message queue

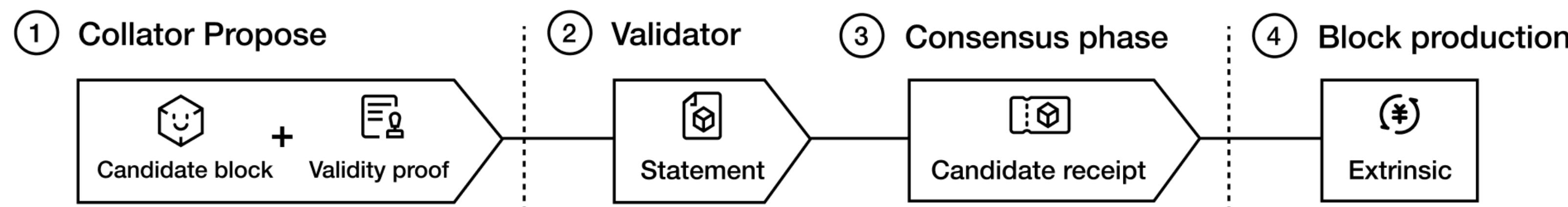


2.4 候选回执(candidate receipt)

- 对于每个parachain的一个区块，最终在relay chain上链的是这个候选回执
- candidate receipt是大小恒定的数据
- 每个参数都是ID、Hash、签名
- 这样在parachain扩展的时候，是一个线性的过程
- 区块数据、状态数据、纠删码数据等，都保存在本地的一个key-value数据库中，availability-store

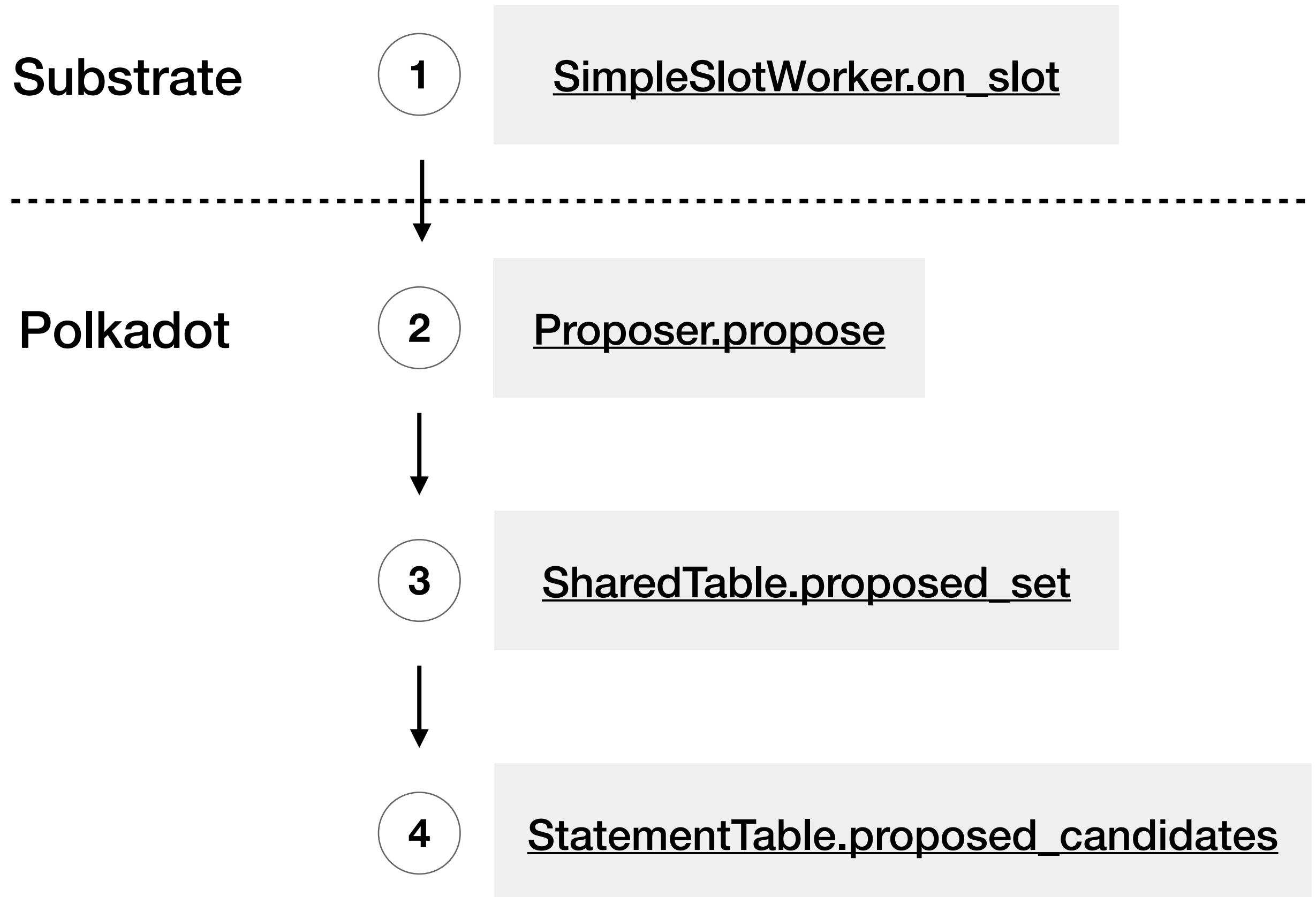
Candidate Receipt

- parachain的ID
- collator的ID 和 签名
- 父区块的candidate receipt的Hash
- 纠删码的默克尔根
- 外发消息的默克尔根
- 区块Hash
- parachain在执行该区块前的状态根
- parachain在执行该区块后的状态根



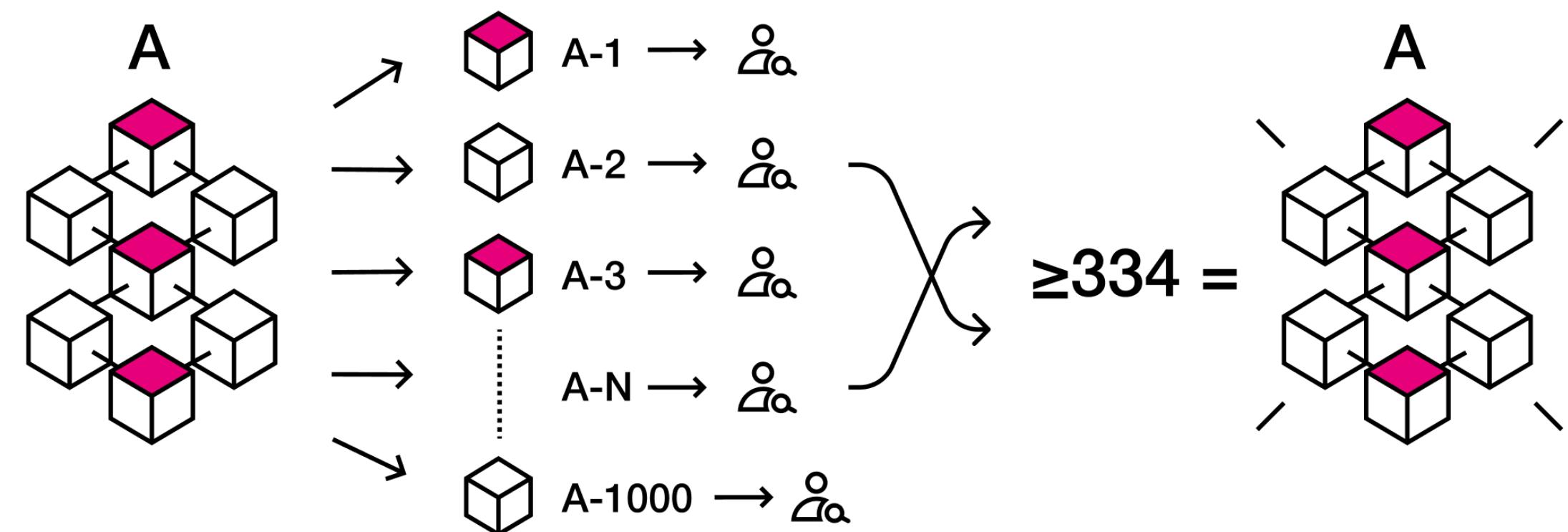
2.7 打包区块

- 一旦超过一半的validators认可，便准备candidate receipt
- substrate的每个slot，babe回调slots中的on_slot函数
- 波卡验证组件中的block_production
- SharedTable保存了一轮共识中，所有的过程数据validated、承诺数据StatementTable、candidate receipt的进展status、availability_store引用
- StatementTable保存了一轮共识中，validator的签名数据、不当行为misbehavior数据、candidate的投票数据；所有数据都是hash、sig、ID
- 区块数据、状态数据、纠删码数据等，都保存在本地的一个key-value数据库中，availability-store
- 对于babe中赢得slot的validator A而言，它的交易池中会有很多candidate receipt，需要保证：
 - candidate receipt的parent candidate已经收录在之前的区块
 - A的本地availability_store中已经保存了该candidate的纠删码chunk



3. 可用性子协议 - 纠删码

- 在candidate receipt生成后、进入交易池之前，validator需要保存区块(block) + 有效性证明(proof of validity) 数据
- 生成candidate receipt的validator，同时生成纠删码数据chunks
- 纠删码会将一个数据A分散成n份分片数据，A1...A10。当超过某个阈值(4)的分片组合在一起时，就可以还原出原始数据A
- 多项式插值算法，可以用在这样的场景：知道两个点可以确定一条线，知道三个点可以确定一条抛物线，知道四个点可以确定一条三次曲线
- 在polkadot中，假设总共有1000个validators，纠删码chunks也发送到了这1000个节点上，那么最终需要超过1/3的节点 - 334个节点的chunks就可以恢复出原始数据
- 收到candidate receipt以及纠删码的验证者，会将candidate receipt包括在中继链交易池中，block author可以再将其包含在一个块中



4-6. 更多的安全保障

所有的验证，都是交叉的

见证人验证了，fisherman、收集人也要验

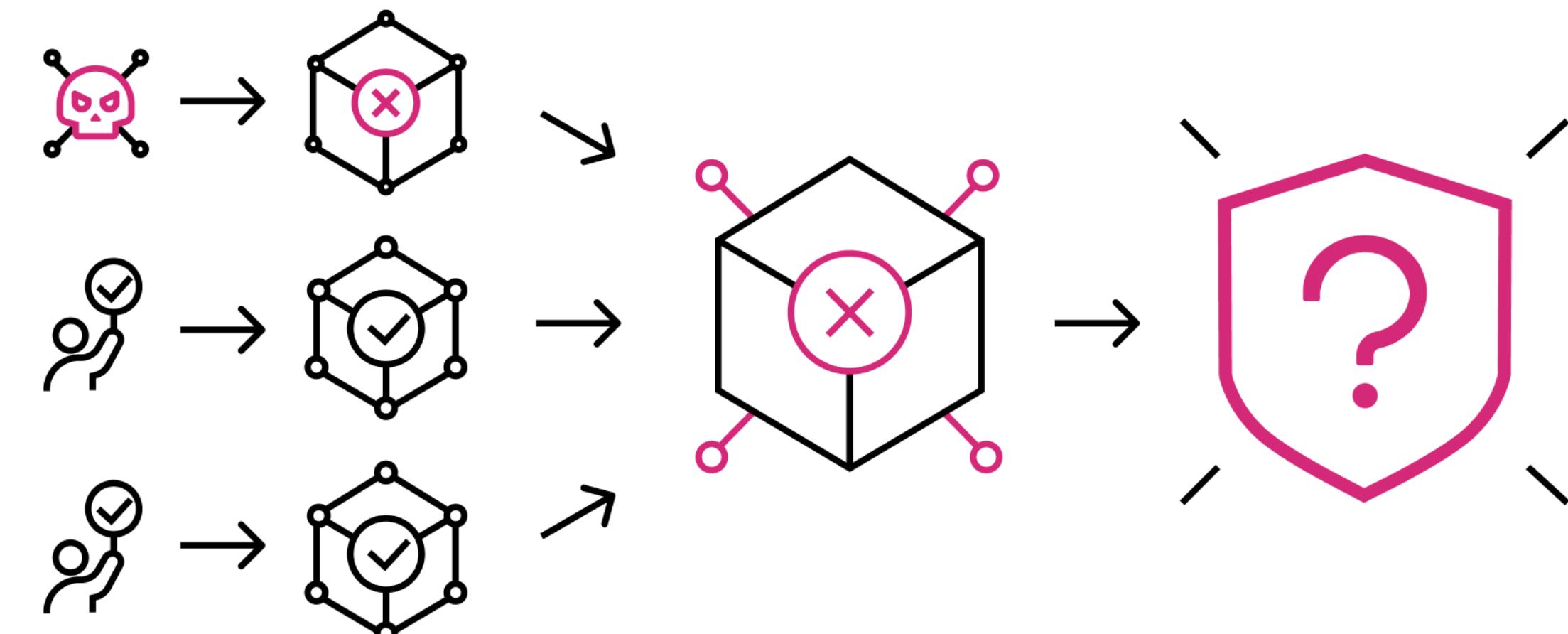
从下、从上分别发起验证

如何处理合谋做坏事

如何处理多分叉下的奖惩信息同步

4-6. 更多的安全保障

- 由于波卡只会分配~10个validators给一个parachain
- 超过1/2的确认便会将某parachain的candidate receipt打包
- 所以这里还是会小概率的存在validators合谋做坏事的情况
- 在将candidate receipt打包到区块后
- 还需要更多的安全机制设计，来保证安全
 - 二次验证：将出块（babe）与最终确认（grandpa）分开，提供一个可以做二次验证的时机
 - 监督机制：fisherman



4-6. 更多的安全保障

1000个节点中，10个一组，分出100组，有六个节点是坏节点，
他们被分到一组的概率是多少？

$$C(4/994) / C(10/1000) = 1.53e-13$$

有十个节点是坏节点，至少六个被分到一组的概率是多少？

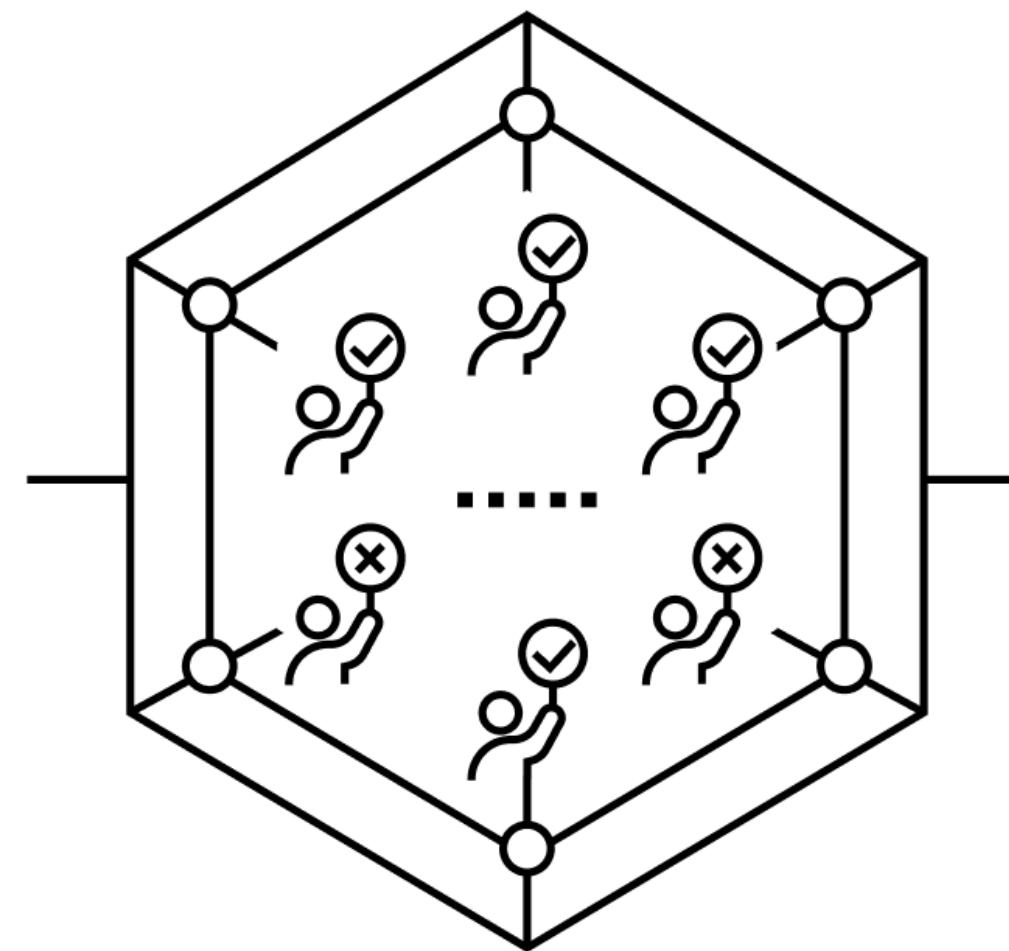
$$(C(6/10) * C(4/994) + C(7/10) * C(3/994) + C(8/10) * C(2/994) + C(9/10) * C(1/994) + C(10/10)) / C(10/1000) = 3.23e-11$$

6秒一个区块，大概要6000年

坏节点数翻个倍，概率提升2个数量级

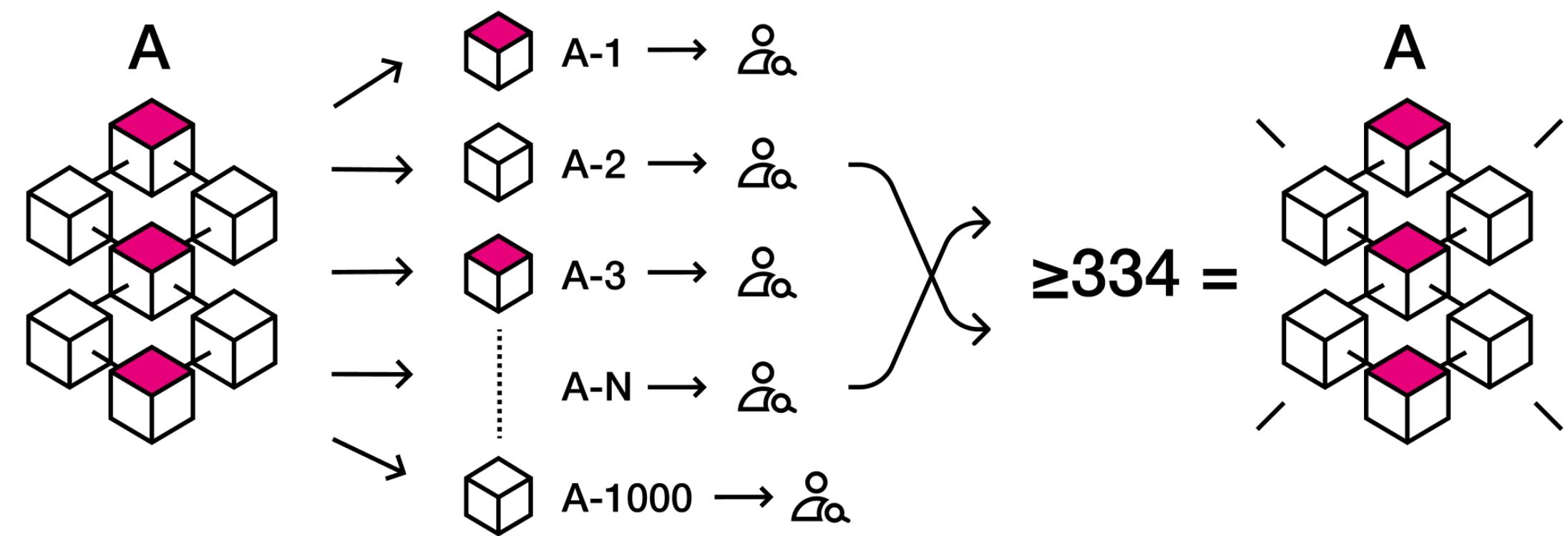
4-6. 更多的安全保障 - 二次验证 - STF

- 当relay chain的babe，生成了一个新区块后，会启动一个二次验证的会话
- relay chain会随机挑选validator对区块中所有的candidate receipt进行验证
- 验证过程包括：
 - 通过纠删码chunks恢复出block data、validity proof, availability
 - 按照前文STF验证的过程，做一次重新的验证，validity
 - 最终比对是否可以生成相同的candidate receipt



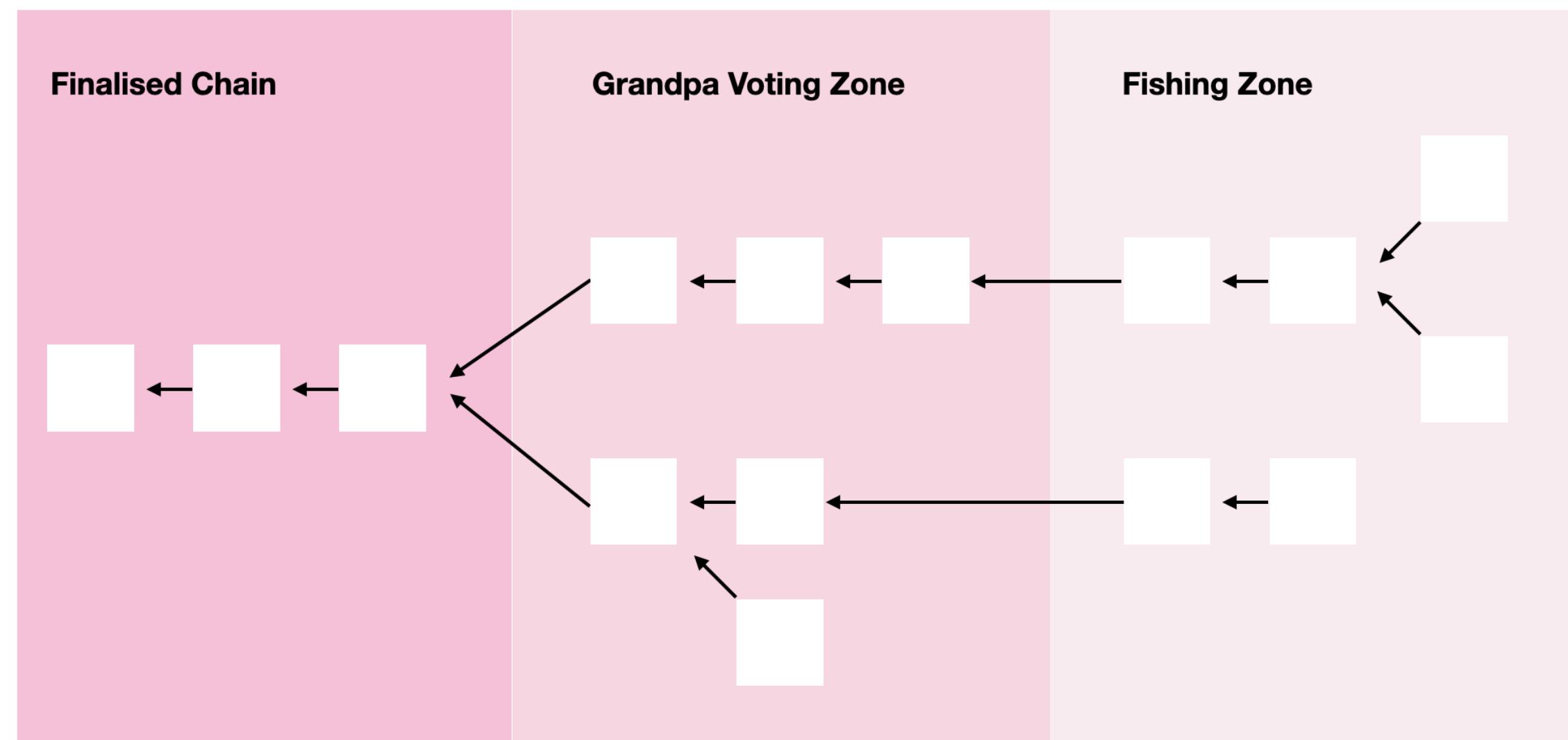
4-6. 更多的安全保障 - 纠删码有效性验证

- 当relay chain生成了一个新区块后，会将block广播给所有的validators
- 每个validators接收到新区块后，会执行一个import block的操作
- 该操作中，会对区块中的每个candidate receipt，查看其对应的纠删码chunk是否保存在本地
- 如果没有保存在本地，则在p2p网络中发出一个warning信息，表明该receipt对应的chunk发生了遗漏
- 如果超过1/3的validators，都对某个receipt发出了warning信息，则该区块立即作废



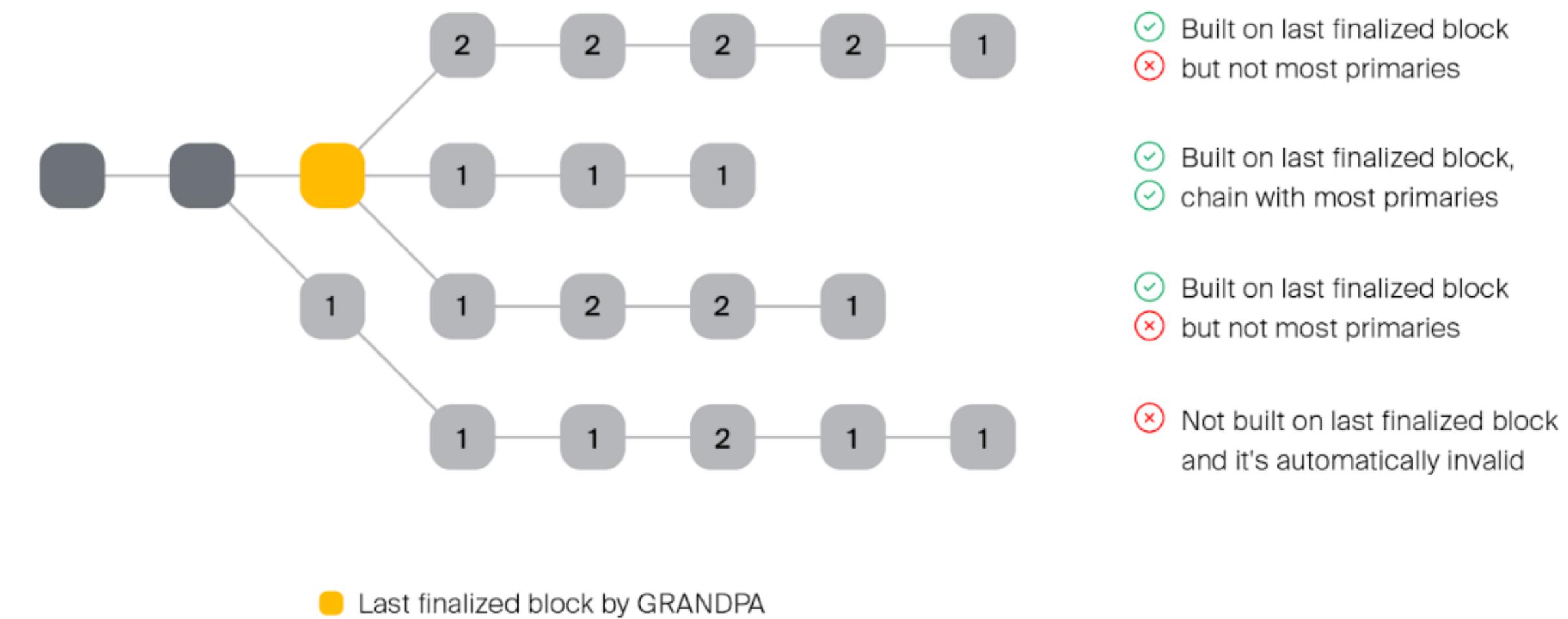
4-6. 更多的安全保障 - 二次验证 - Fishing Zone

- 接下来，collators和fisherman，会执行更多的验证工作
- fisherman是parachain 或者 relay chain的节点
- 成为fisherman，首先需要抵押一定数量的dot
- fisherman负责验证区块中所有candidate receipt的有效性 (validity)
- collator负责验证纠删码 (chunks) 的可用性 (availability)
- 一旦发现问题，fisherman便会发出invalidity (抵押dot) 、collator发出unavailability的消息
- 随着关于某个candidate receipt的invalidity、unavailability消息越来越多，需要更多的validators参与验证
- 最终，如果对于某个candidate receipt，超过1/3的validators都验证是invalidity或者unavailability，则该block作废



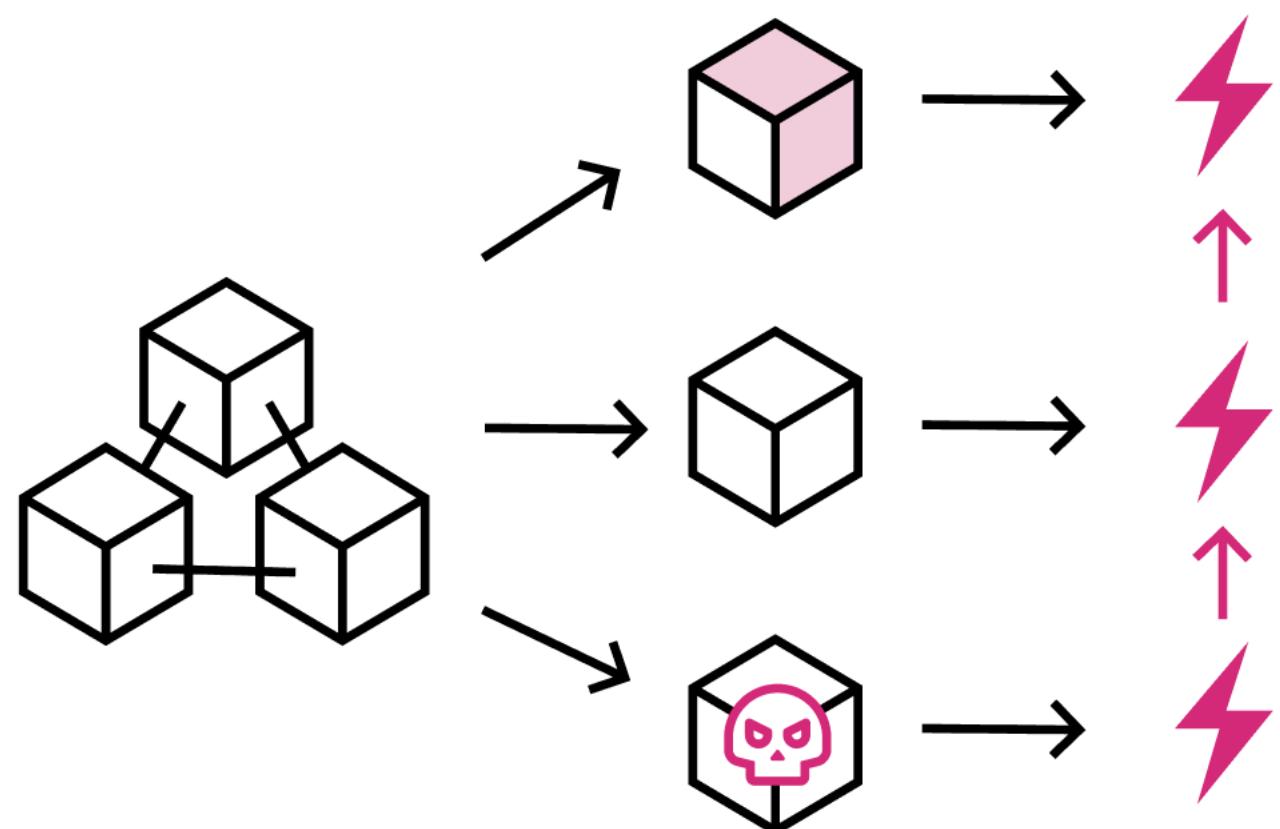
4-6. 更多的安全保障 - GRANDPA

- GRANDPA模块，负责区块的最终性确认
- GRANDPA做最终性确认时，确认的是链，而不是区块
- 该协议可传递地应用投票，并且GRANDPA算法找到具有足够数量的投票的最高区块号，以将其视为最终投票。此过程允许在一个回合中完成N个块的确认
- Polkadot中的所有parachain都遵循relay chain的终结性，未来的parablock必须始终以最终确认的、中继链中的candidate receipt为基础
- 一旦完成，该块将从共享的安全环境中受益，该安全环境允许链以无信任的方式彼此交互，而撤销该块意味着撤销relay chain的区块，那是非常难的



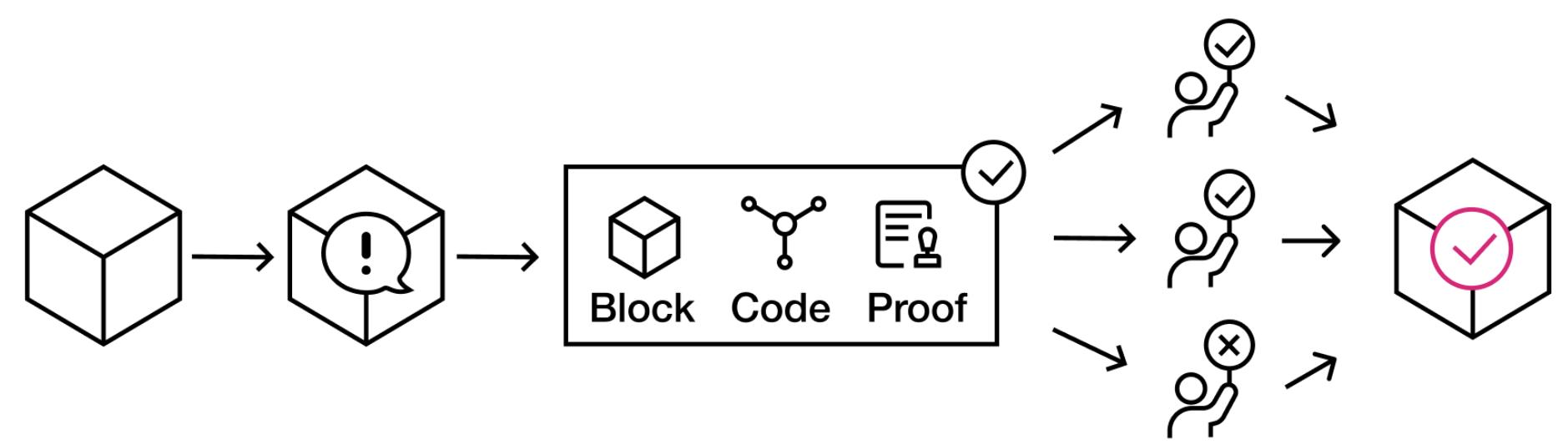
4-6. 更多的安全保障 - 分叉 (Fork)

- 前面讲过，如果超过1/3的validators在产块后，对某个candidate receipt验证时出错，则区块作废
- relay chain是使用babe出块的，它不停的在出块，所以“作废”操作，是在一个多分叉 (fork) 的环境下进行，这样事情又复杂了很多
- 在一个fork上有作恶行为的validator，要在所有的fork上进行惩罚
- 有争议的parablock没有出现在所有的fork中
- 有作恶行为的validator，不能创建fork以去掉自己的作恶行为
- 如果出块validator发现某fork上存在有争议的parablock，会导致该fork出现revert，则最好激励该validator在不包含该parablock的其它fork上构建区块



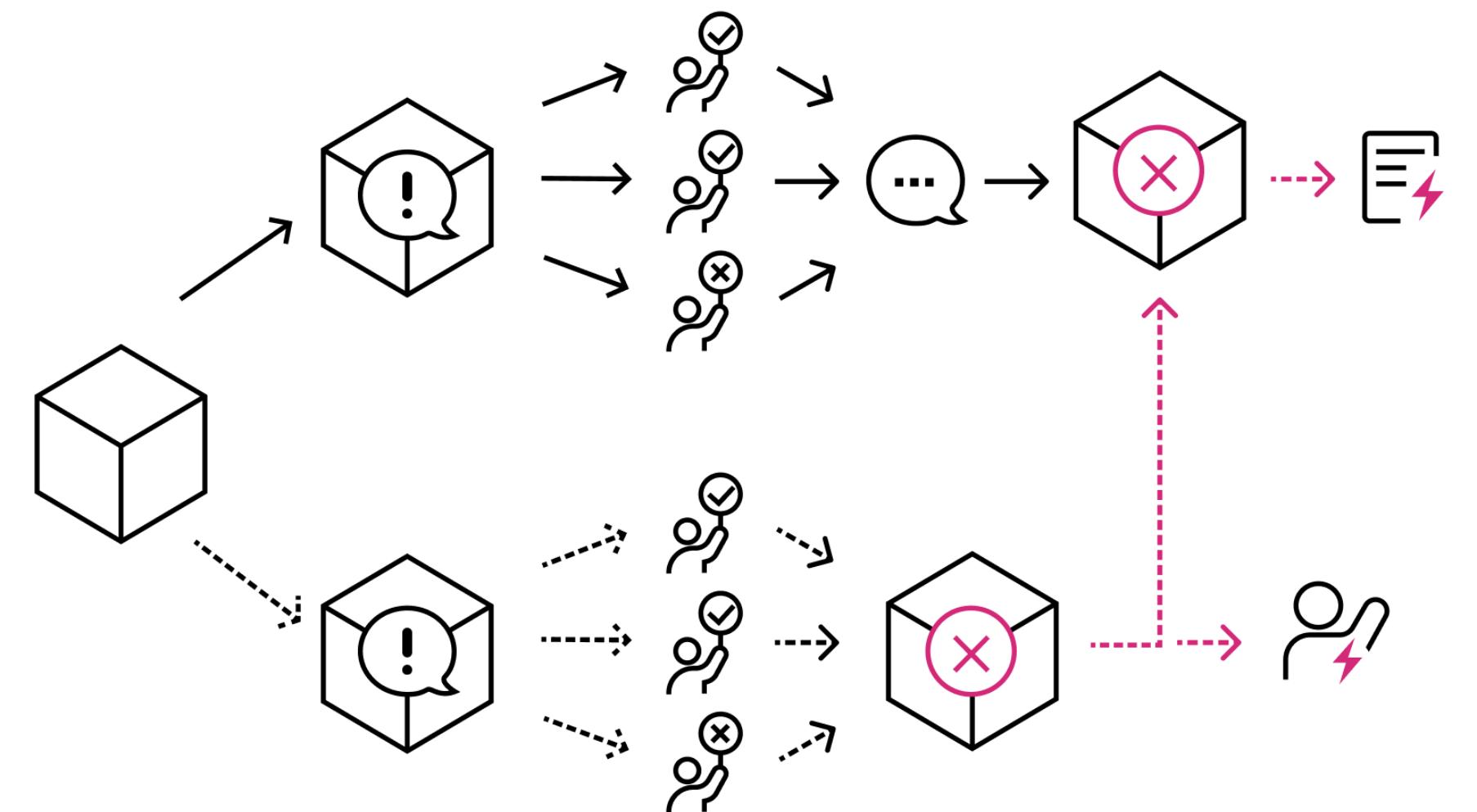
4-6. 更多的安全保障 - 本地争议 (Local Disputes)

- 争议：对于parablock，已经生candidate receipt、打包后，在后续验证过程中出现了问题的情况
- 本地争议：对于当前fork中存在的争议
- 已经打包、且在验证期的parablock，可以有本地争议
- 对于存在争议的parablock，需要保证其block、code、proof都可用
- 争议可能会存在于之前的session，需要上一轮的validators set进行争议解决；这说明validator duty延长了一个sesion
- 争议解决后，在错误那一方的validators会被slash
- 区块链会revert到争议前的位置；争议区块会进入黑名单



4-6. 更多的安全保障 - 远程争议 (Remote Disputes)

- 远程争议：已在另一fork部分或全部解决的争议，该争议的结论数据，需要同步到其它的fork
- 但是，这样会带来很多的挑战
- 与本地争议一样，parablock所在fork的validators set负责解决并确定candidate的可用性
- 如果可用性验证没有通过，则会惩罚不支持争议的那些validators；这个结论也会同步到所有其它分支
- 如果可用性验证通过，那么可以将该远程争议replay到所有其它fork中，需要保证在所有fork中选择validators set的VRF是相同的
- 确保每个分支上都replay了该远程争议
- 对于每个replay该争议的分支，按照上节执行本地争议的逻辑进行处理



关于波卡的TPS

- 每个parachain的tps是1000
- 总共有100个parachains
- relay chain有1000个validators，每个parachain分配10个validators
- 在每个relay chain的slot，每组validators只验证一个parachain
- relay chain 与 parachain的block time都是3秒
- $((1000 * 3) * 100) / 3 = 100,000 \text{ tps}$

参考文档链接

- [The Path of a Parachain Block](#)
- [Availability and Validity](#)
- [polkadot-path-of-a-parachain-block \(video\)](#)
- [Availability and Validity by web3](#)
- [Parachain Allocation](#)
- [Security of the network](#)
- [Polkadot Consensus Part 1: Introduction](#)
- [Polkadot Consensus Part 2: GRANDPA](#)
- [Polkadot Consensus Part 3: BABE](#)
- [Validity Module](#)
- [Consensus in Substrate](#)
- [Overview of Polkadot and its Design Considerations](#)

谢谢