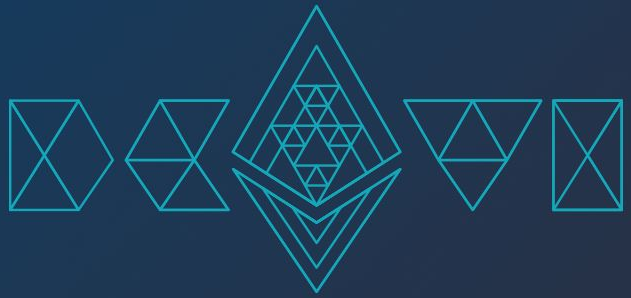


Hybrid PBS in Consensus Layer

Terence Tsao

Core dev @ Offchain Labs

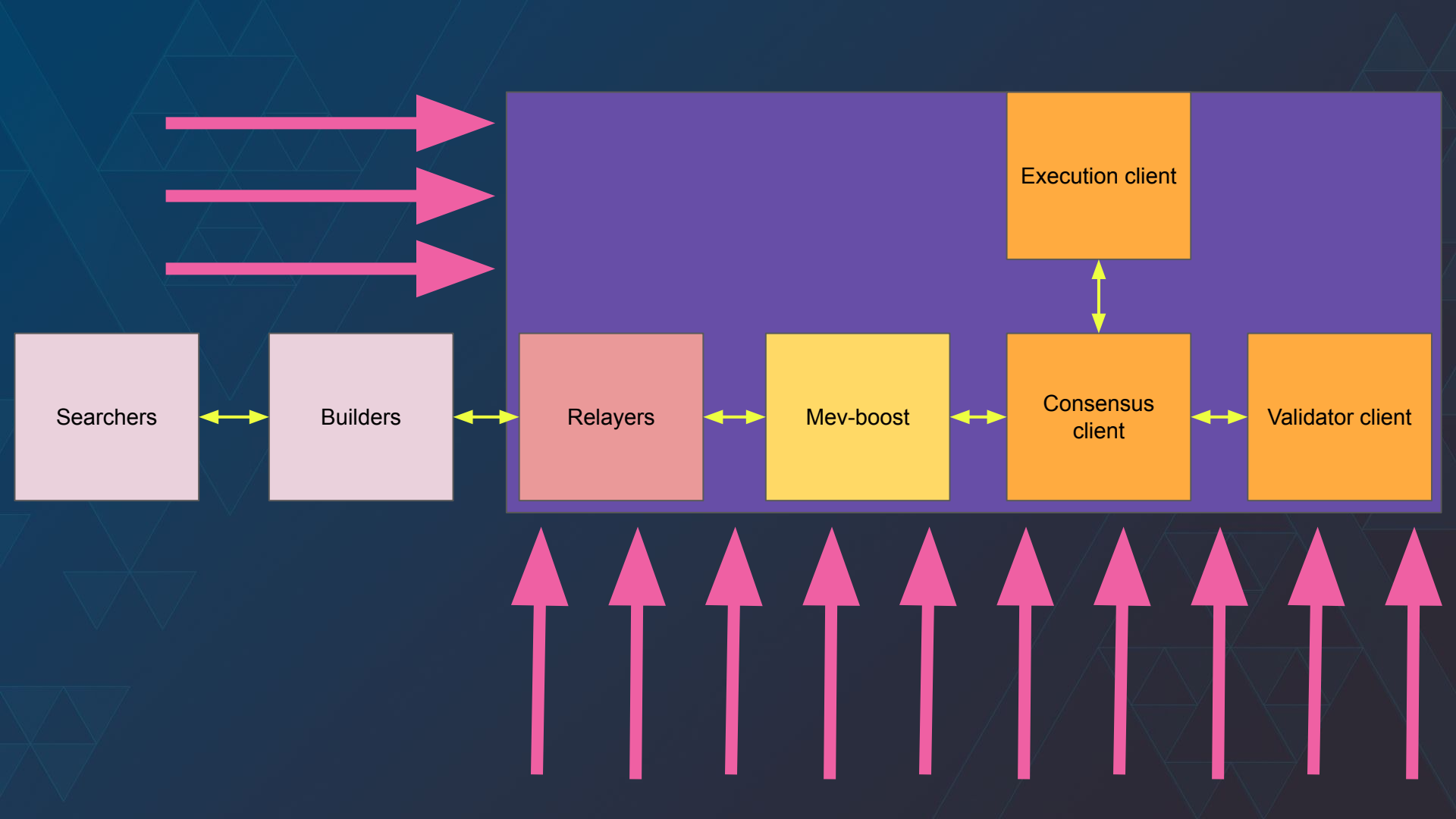
Before we start...

What this talk is:

- **Consensus layer** interface
- Hybrid PBS risks (today)
  - Latency
  - Fault
  - Censorship
- Mitigations

What this talk is **not**:

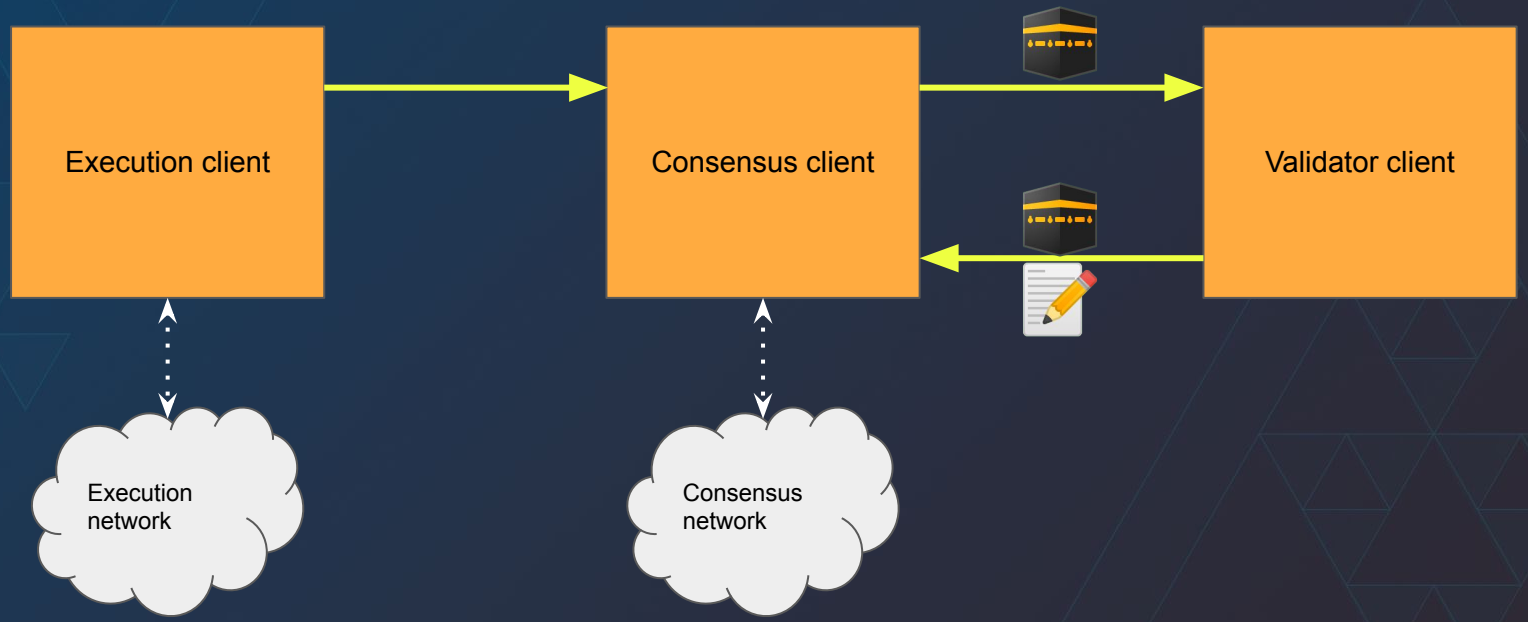
- **Searcher & builder** interface
- Your typical MEV talks...


Execution client
Searchers
Builders
Relayers
Mev-boost
Consensus client
Validator client

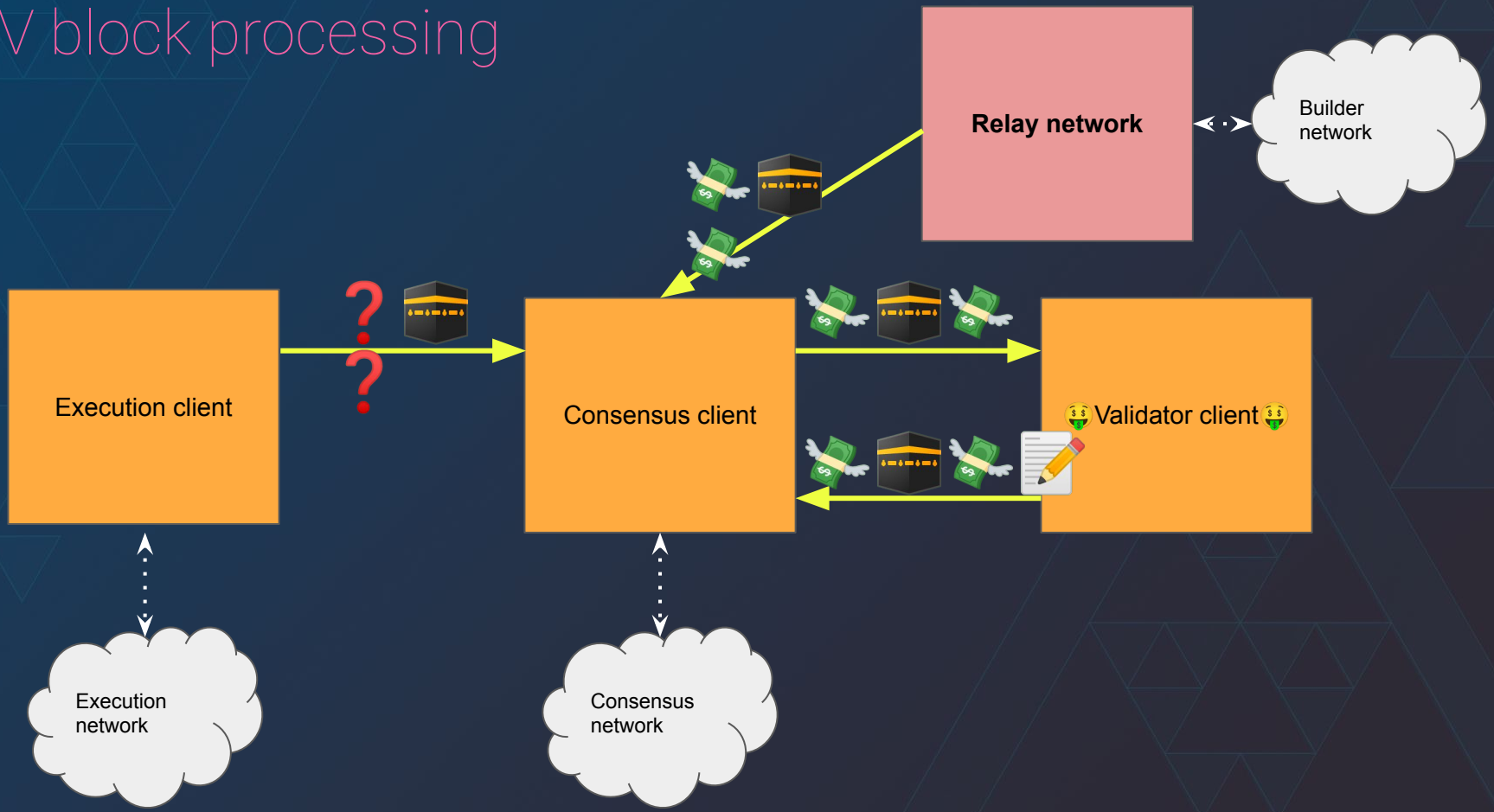
Section 0

Background

Normal block processing



MEV block processing



Today's number

mevboost.org

Tracking MEV-Boost relays and block builders. A quick hack by Anish. Design inspired by file.app. API documentation.

Network participation (24h)

55.31%

% of MEV-Boost blocks relayed in last 24h.

Flashbots dominance

81.36%

% of MEV-Boost blocks relayed by Flashbots.

Active relays

7

Relays that relayed at least one block (Flashbots, BloXroute Max Profit, Blocknative, BloXroute Ethical, Manifold, BloXroute Regulated, Eden).

Top relays

Relays sorted by number of relayed blocks.

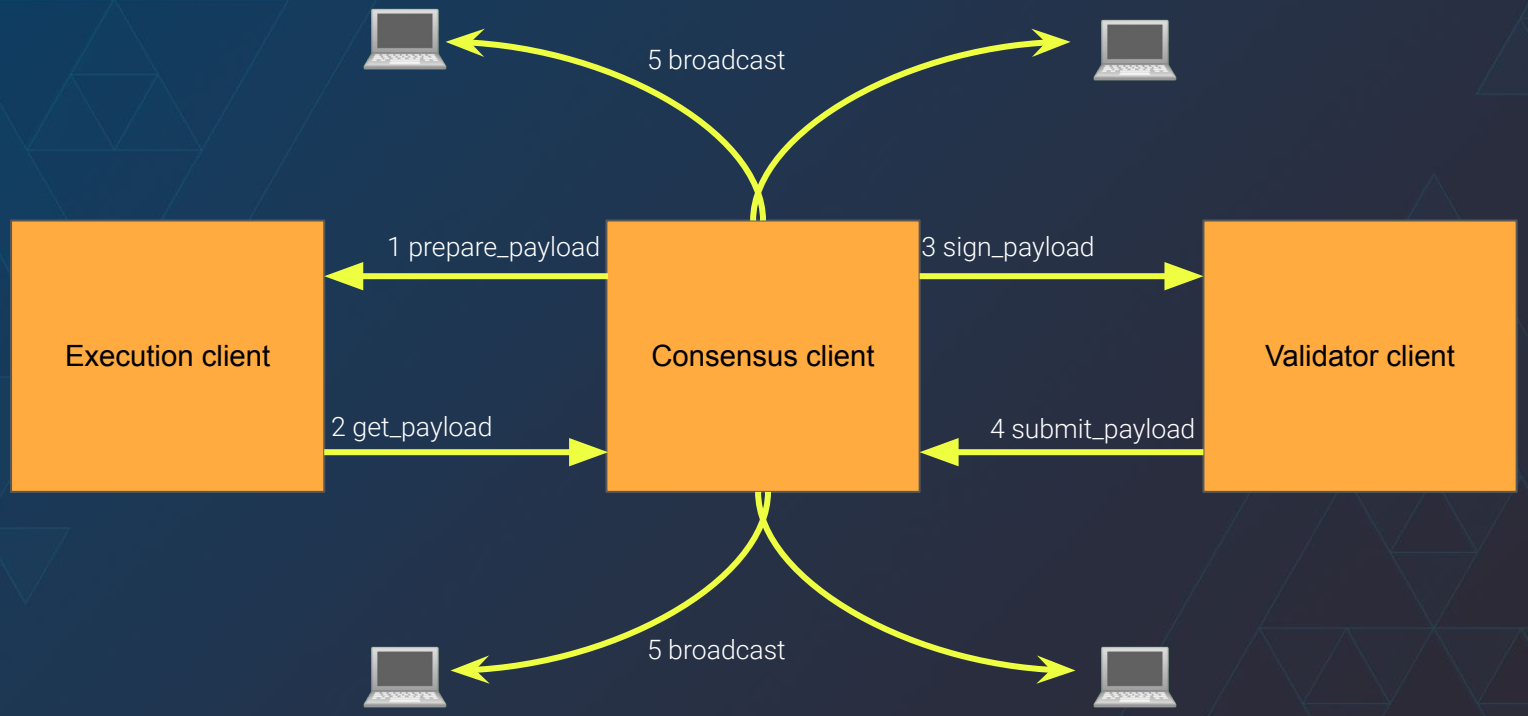
RELAY	# BLOCKS	TOTAL VALUE (ETH)	AVERAGE BLOCK VALUE (ETH)
Flashbots	64,033	9,418.484	0.147
BloXroute Max Profit	6,212	779.929	0.126
Blocknative	2,254	164.847	0.073
BloXroute Ethical	2,101	185.542	0.088
Manifold	1,477	140.422	0.095
BloXroute Regulated	1,359	189.957	0.14
Eden	1,271	172.03	0.135

https://www.mevboost.org/ Oct 13, 2022

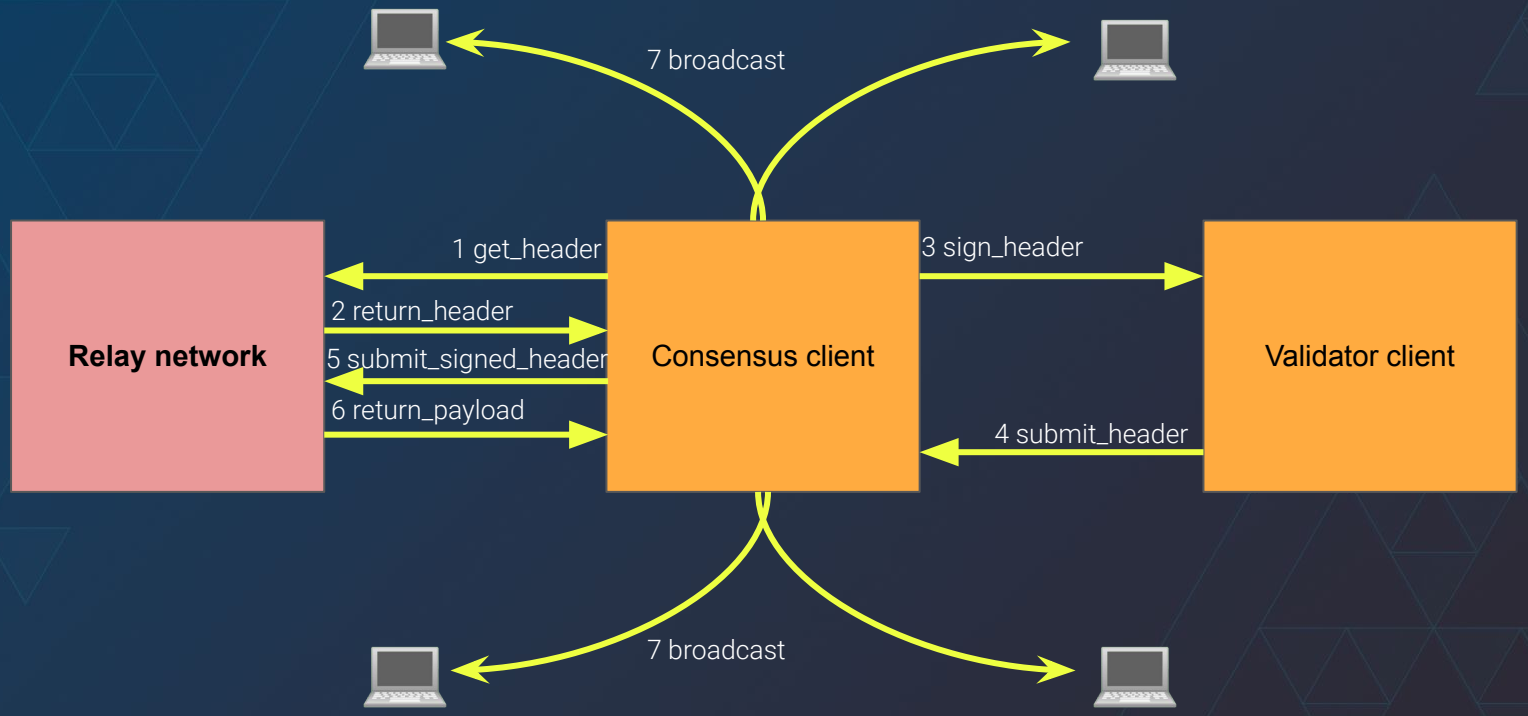
Section 1

Risk: Latency

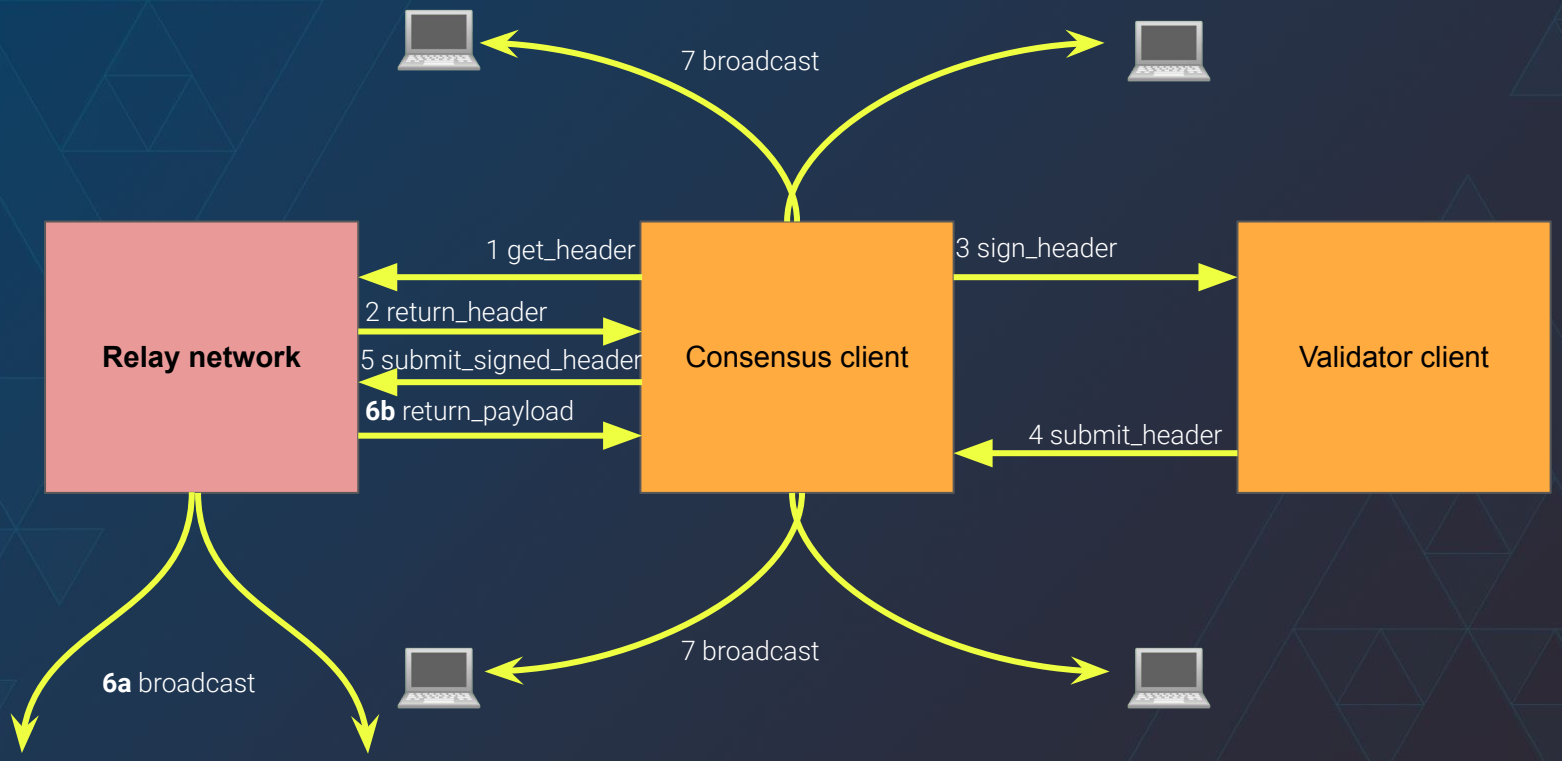
Normal block processing



MEV block processing



MEV block processing improved



publish blocks via beacon node #103

Merged metachris merged 1 commit into `main` from `bn-publish-block` 18 days ago

Conversation 4 | Commits 1 | Checks 4 | Files changed 9

metachris commented 19 days ago • edited — Collaborator

📝 Summary

Publish the signed beacon block to the network

closes #100

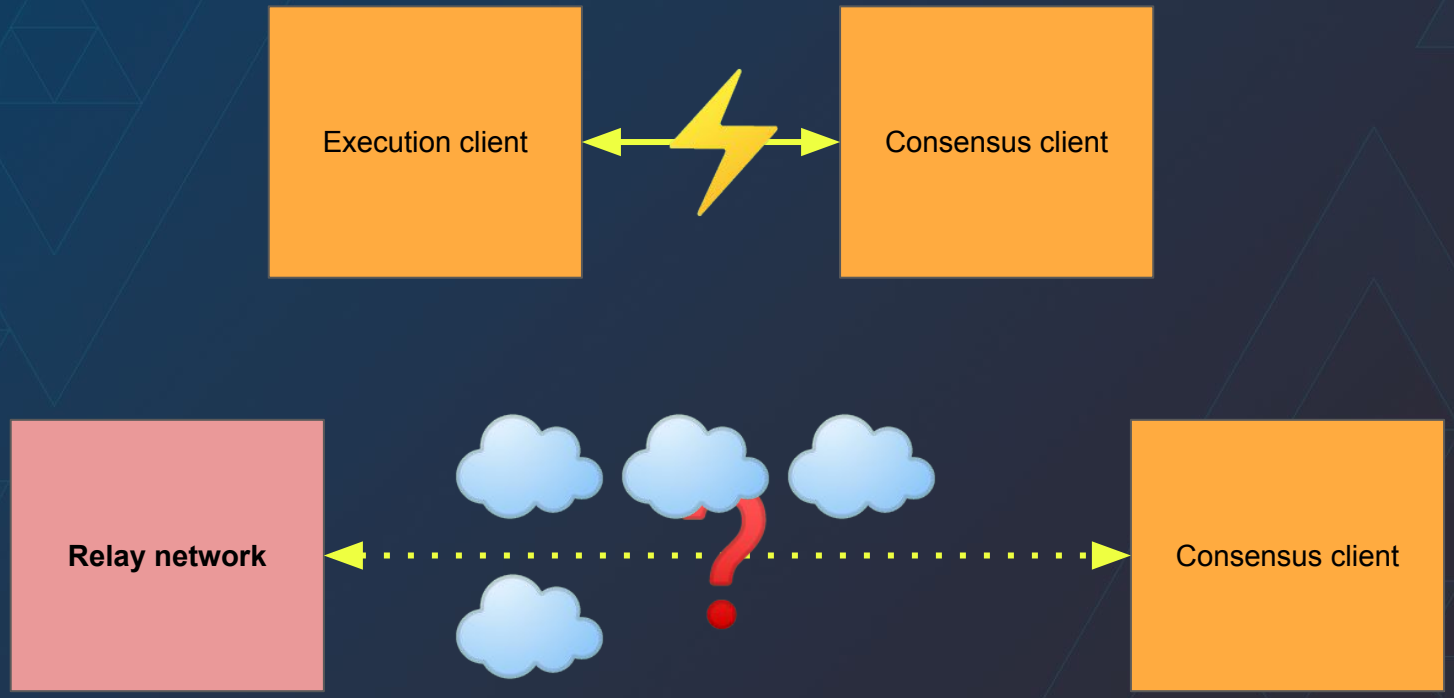
deployed on https://builder-relay-goerli.flashbots.net/

✅ I have run these commands

- [x] `make lint`
- [x] `make test-race`
- [x] `go mod tidy`
- [x] I have seen and agree to `CONTRIBUTING.md`

https://github.com/flashbots/mev-boost-relay/pull/103

Do the additional round trips + latencies matter?

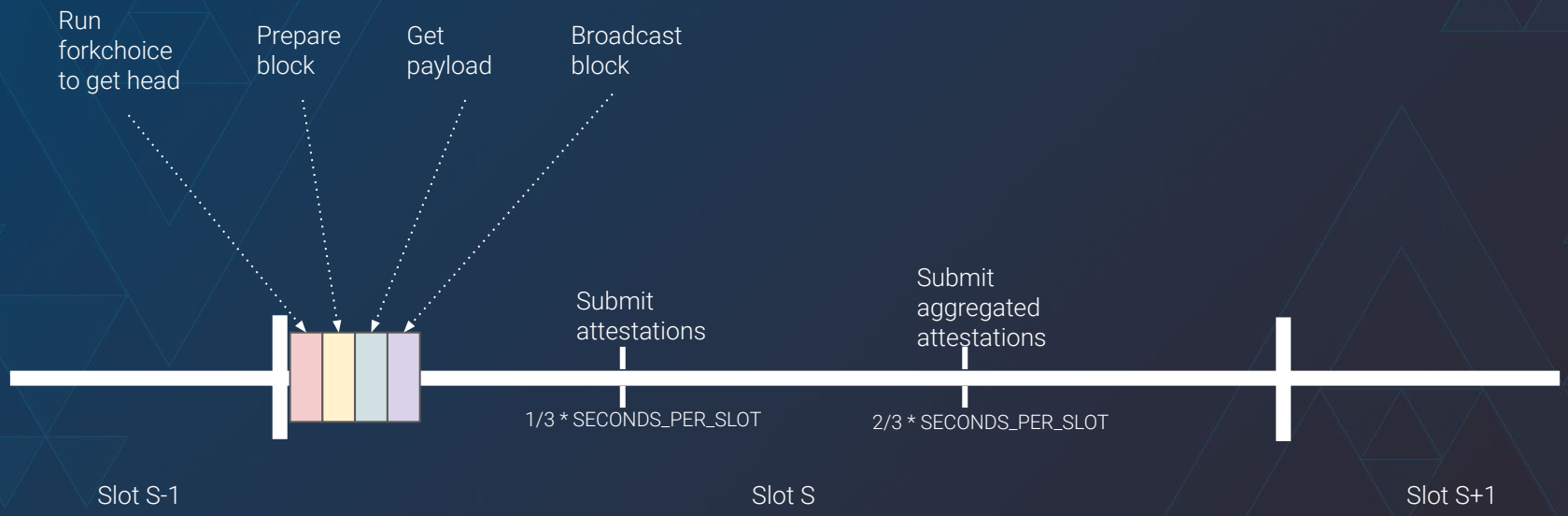


Time to propose block differences (without mev-boost)

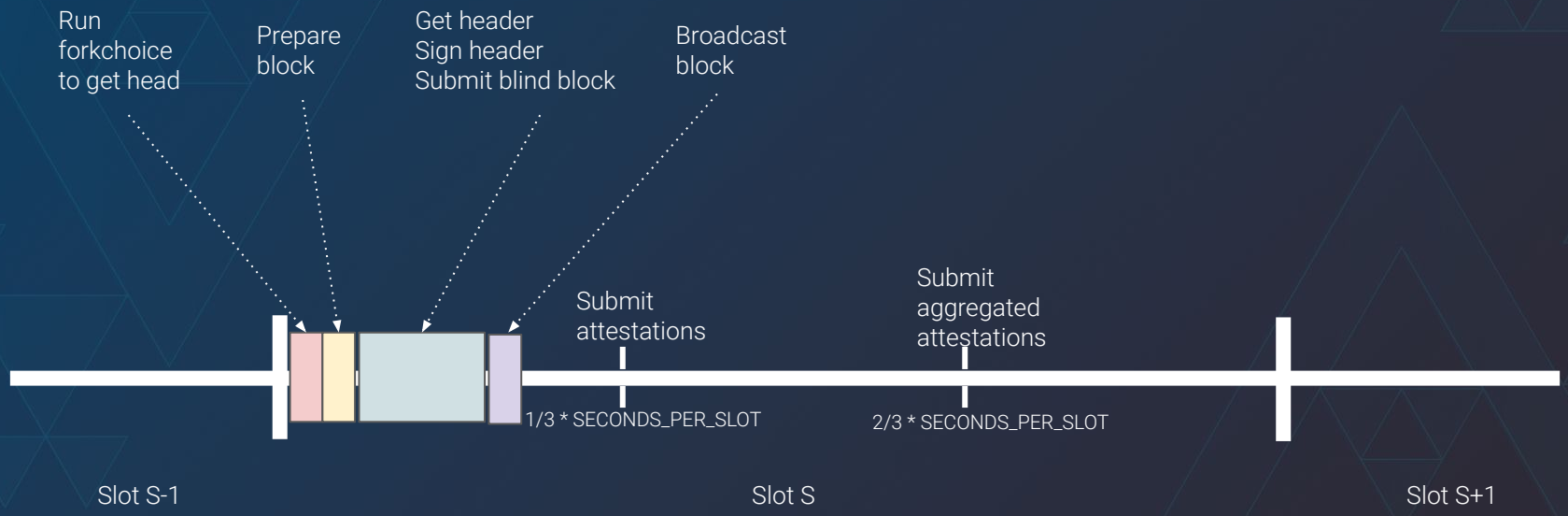
Types	Time to propose (since start of the slot)	Samples
Normal block	310 ms	20
MEV block	1098 ms	20

Network: Goerli (1000 validators). Captured on Macbook Pro.
2.6 GHz 6-Core Intel Core i7. 16 GB 2667 MHz DDR4
2022-10-01 - 2022-10-02
https://github.com/prysmaticlabs/prysm/tree/devcon

Normal block proposal timeline



MEV block proposal timeline

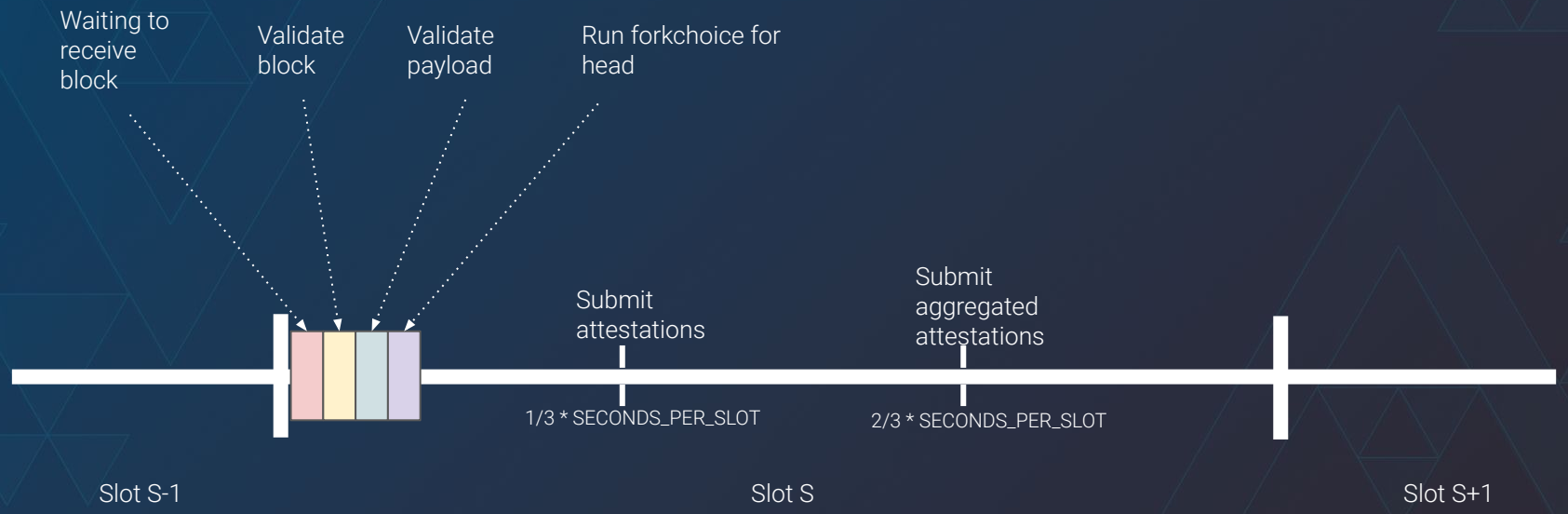


Block arrival latency differences

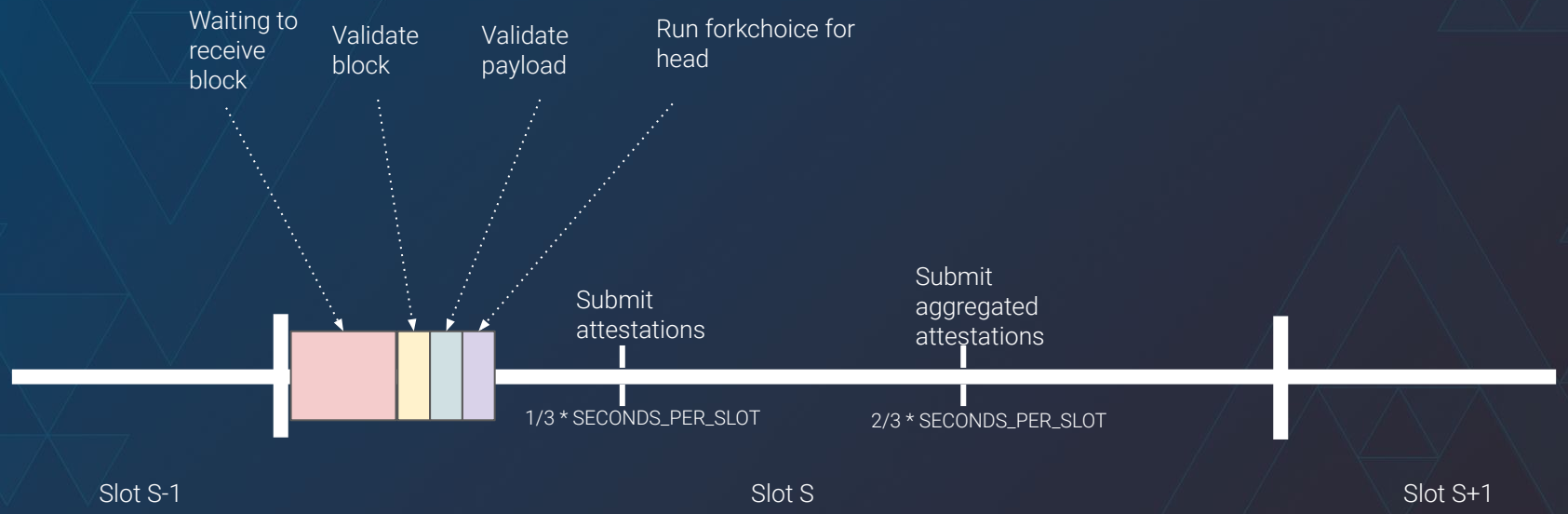
Types	Time to arrive (since start of the slot)	Samples	Extra Data
Normal block	1158ms	20817	
MEV block	1624ms	16125	"Illuminate Dmocratize Dstribute" "Powered by bloXroute" "@builder0x69"

Network: Mainnet. Captured on my NUC at home, From Sep,26-Oct,1
300Mb bandwidth.
https://github.com/prysmaticlabs/prysm/tree/devcon

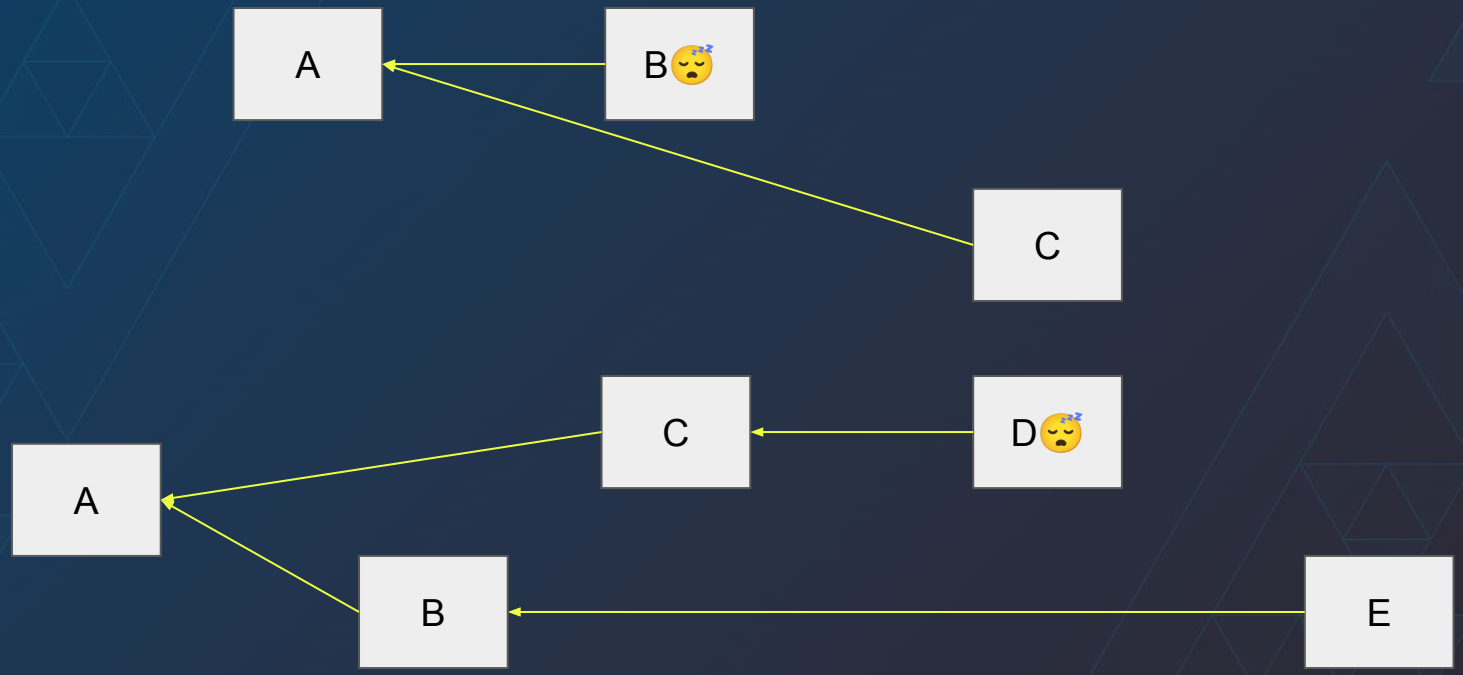
Submit attestation timeline



Submit attestation timeline if block takes longer



Late blocks getting orphaned

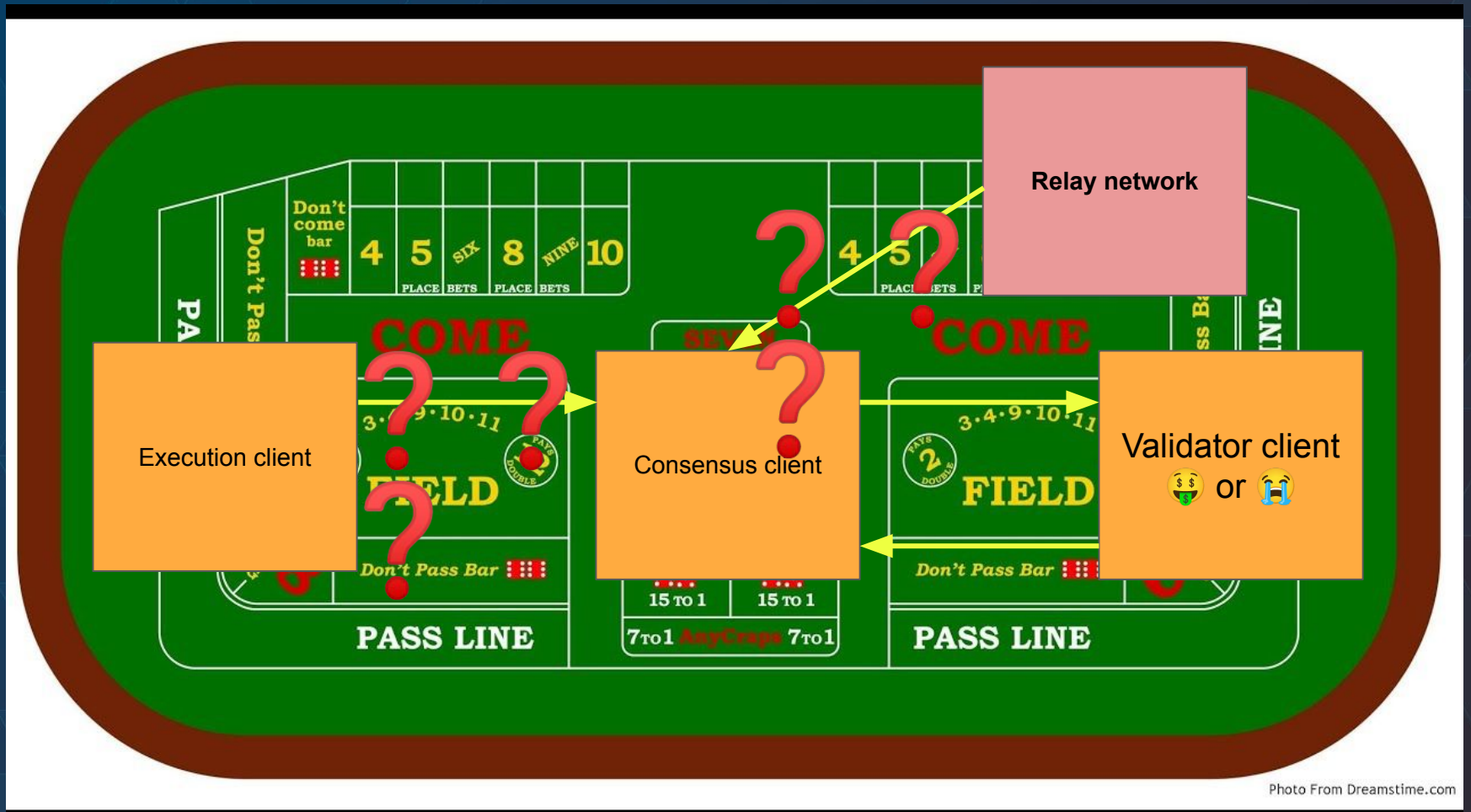


50% of the orphaned blocks came from relayers

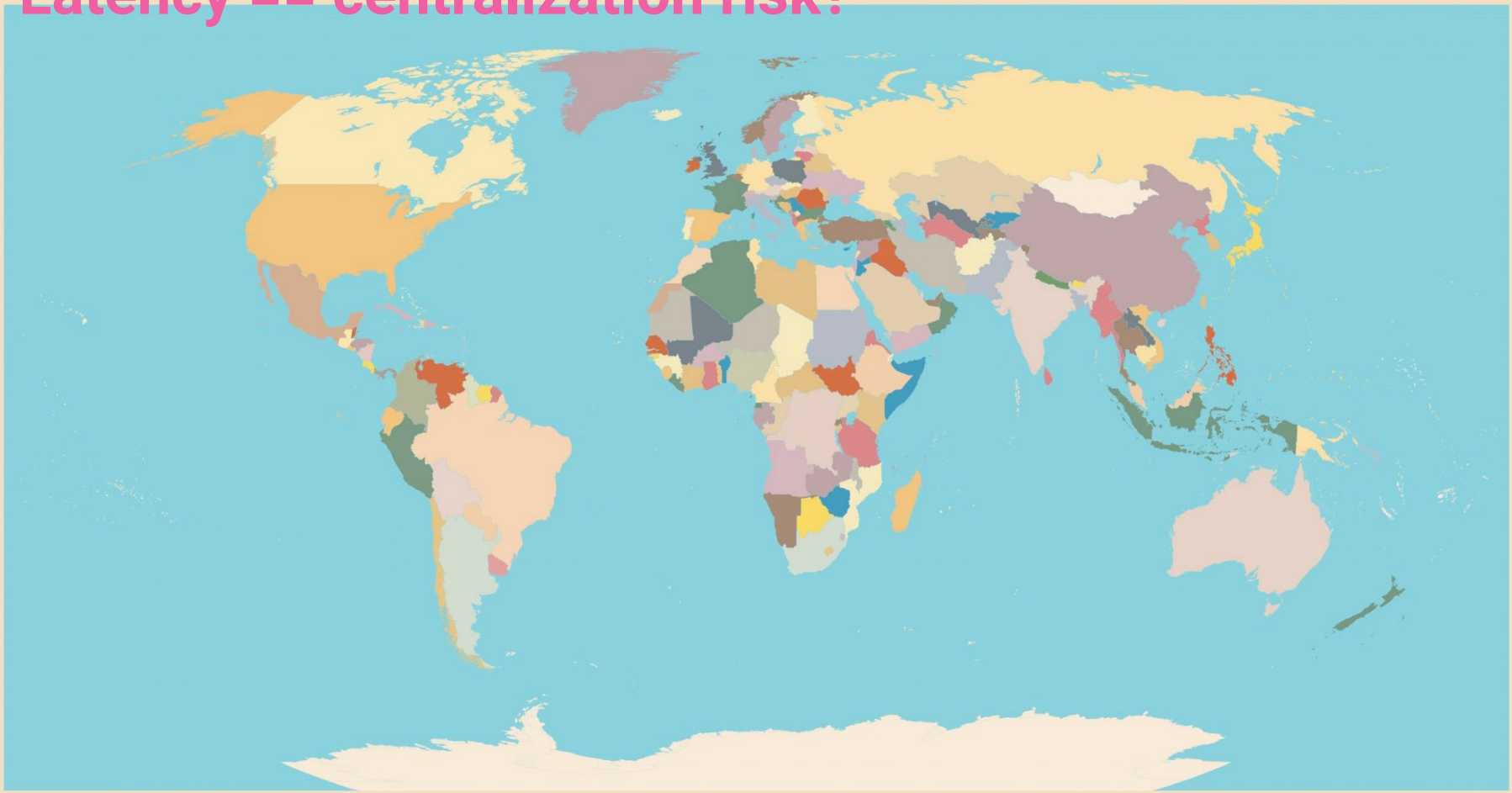
(Sep 17 - Sep 27)

Orphaned block	Relay	Validator ID	Entity
4788790	No Relay	405190	
4775168	Flashbots	50483	Stakefish
4773300	Flashbots	333970	Rocketpool
4750605	Flashbots	293666	Rocketpool
4745786	BR, max profit	127406	
4734533	No Relay	371425	
4733889	No Relay	323484	
4726812	No Relay	355127	
4721112	No Relay	138183	
4716243	BR, ethical	13278	

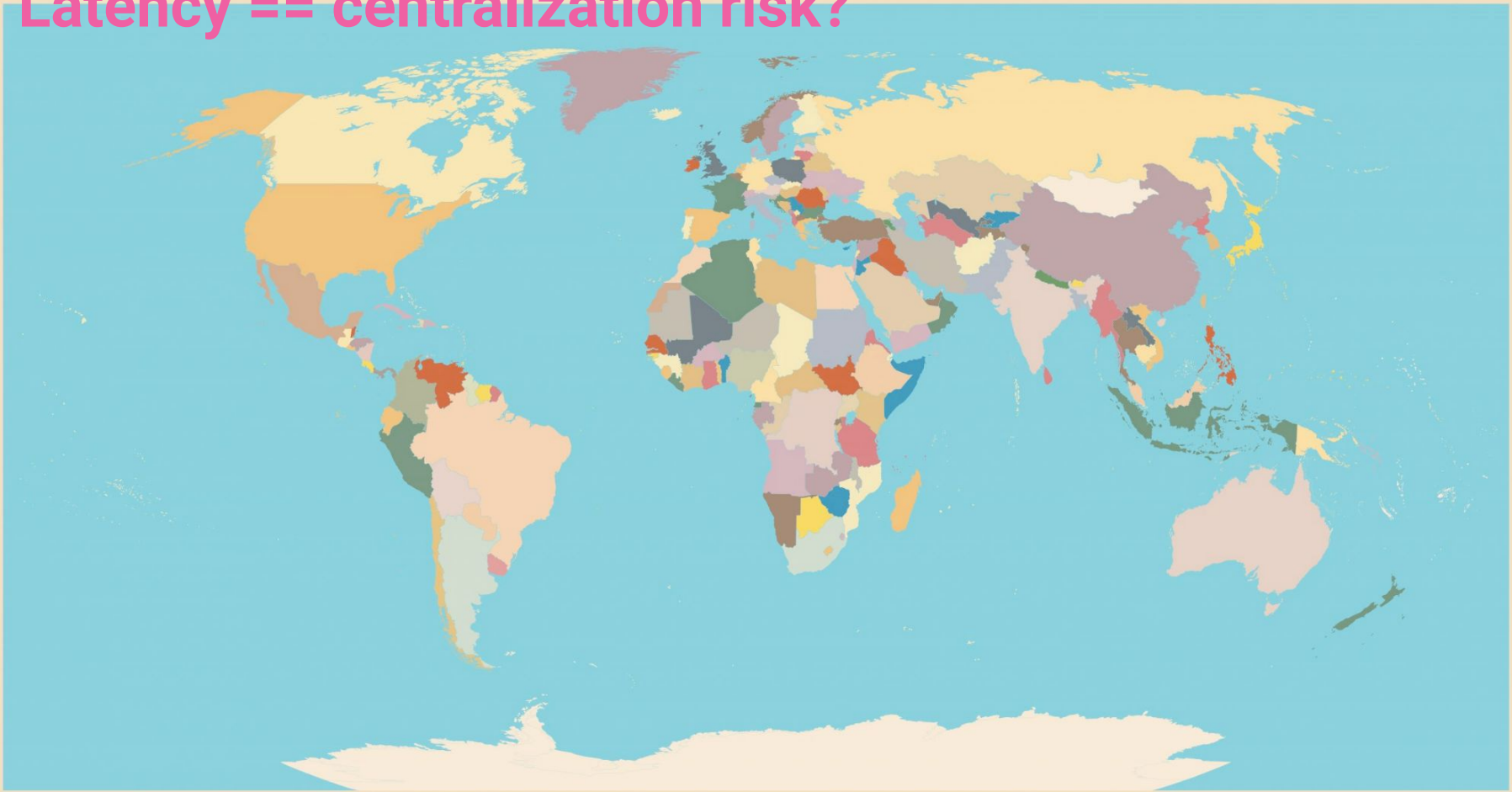
slots: 4715000 - 4790000, **dates**: sep/17/2022, 8:40:23 AM - sep/27/2022, 6:40:23 PM


Relay network
Execution client
Consensus client
Validator client
🤑 or 😭
Don't come bar
4
5
SIX
8
NINE
10
PLACE BETS
PLACE BETS
COME
COME
FIELD
FIELD
Don't Pass Bar
Don't Pass Bar
15 TO 1
15 TO 1
7 TO 1
7 TO 1
PASS LINE
PASS LINE
Photo From Dreamstime.com

Latency == centralization risk?



Latency == centralization risk?



Takeaways

- Latency **matters** for both hybrid and in-protocol PBS
- Latency can lead to **centralization** risks
- **Hard** to optimize network latency
  - Beacon client optimization
  - Mev boost optimization
  - Relay optimization
  - Network configs

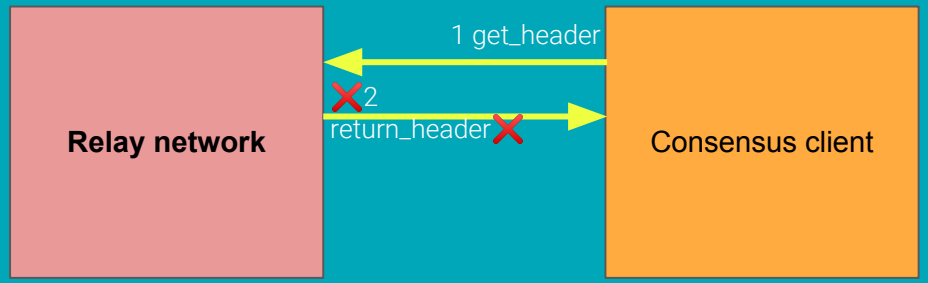
Section 3

Risk: Faults

Where relay faults can happen?

Get header (commit) faults

- Malformed header
- Consensus invalid header
- Payment invalid header
- Non conforming header



Malformed header

- Syntatically invalid
- Invalid structure
- Invalid signature

Can consensus layer client detect? 🟢

```
class SignedBuilderBid(Container):
    message: BuilderBid ✅
    signature: BLSSignature ✅

class SignedBuilderBid(Container):
    message: BlockHeader ❌
    signature: BLSSignature ✅

class SignedBuilderBid(Container):
    message: BuilderBid ✅
    signature: BadBLSSignature ❌
```

Consensus invalid header

- Invalid with respect to consensus rule
- Invalid block number
- Invalid block hash
- Invalid transaction

Can consensus layer client detect?

🟡

```
class ExecutionPayloadHeader(Container):
    parent_hash: Hash32
    fee_recipient: ExecutionAddress
    state_root: Bytes32 ❌
    receipts_root: Bytes32
    logs_bloom: ByteVector[BYTES_PER_LOGS_BLOOM]
    prev_randao: Bytes32
    block_number: uint64 ❌
    gas_limit: uint64
    gas_used: uint64
    timestamp: uint64 ❌
    extra_data: ByteList[MAX_EXTRA_DATA_BYTES]
    base_fee_per_gas: uint256
    block_hash: Hash32  # Hash of execution block
    transactions_root: Root
```

Payment invalid header

- Payment doesn't fulfill the value delivered to the proposer
- Require Execution API support

Can consensus layer client detect?

```
class SignedBuilderBid(Container):
    message: BuilderBid
    signature: BLSSignature
```

0x00cf0...b3a6b	Transfer	0xdafea...98bc5	0xffee0...1143b	0.10633 Ether	0.00015 Ether	7.45274 GWei

Overview

Transaction Hash:	0x00cf034107e7737be80c878e76d0bdd943043e306a5162121c18e5c3eacb3a6b
Status:	Success
Block:	15635025
Timestamp:	19 min. ago
From:	0xdafea4...c98bc5
Interacted With:	0xffee08...b1143b Contract
Value:	0.1063314 Ether
Transaction Fee:	0.00015675 Ether
Effective Gas Price:	7.45274174 GWei

Advanced Info

Non conforming header

- Incorrect gas limit
- Incorrect timestamp
- Incorrect parent hash

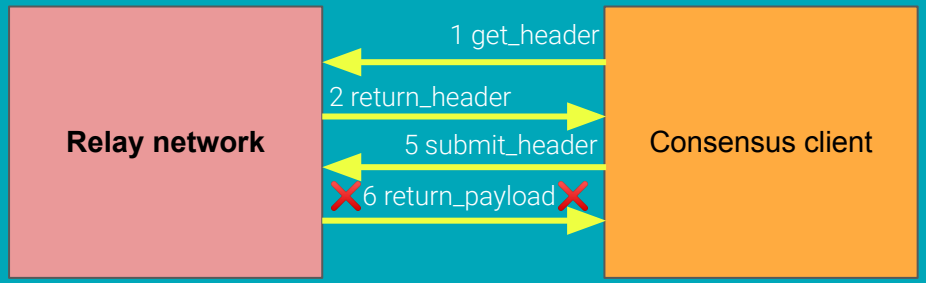
Can consensus layer client detect? 🟢

```
class ValidatorRegistrationV1(Container):
    fee_recipient: ExecutionAddress
    gas_limit: uint64 ❌
    timestamp: uint64
    pubkey: BLSPubkey
```

Get payload (reveal) faults

- Malformed payload
- Consensus invalid payload
- Unavailable payload

(There's no fall back for these)



Malformed payload

- Execution payload is syntactially invalid
- Full payload does not match committed header

Can consensus client validate? 🟢

(but it's too late....) 💀

```
class ExecutionPayloadHeader(Container):
    block_number: uint64

class ExecutionPayload(Container):
    block_number: uint64
```

Consensus invalid payload

- Payload contains invalid txs

Can consensus client validate? 🟢

(but it's too late....) 💀

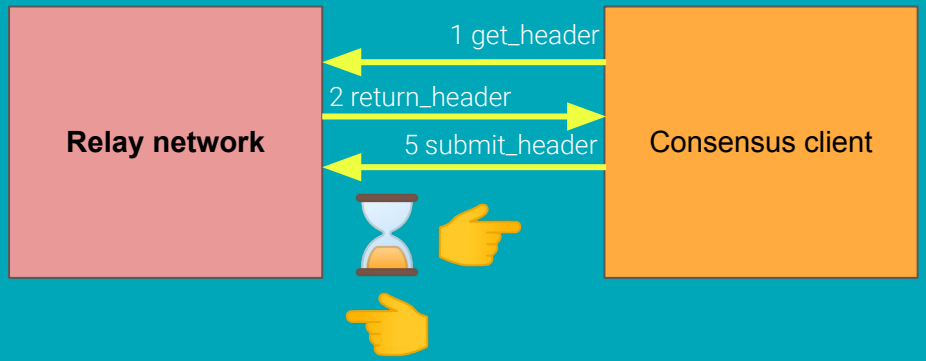
```
class ExecutionPayloadHeader(Container):
    transactions_root: Root

class ExecutionPayload(Container):
    transactions: List[Transaction,
MAX_TRANSACTIONS_PER_PAYLOAD]
```

Unavailable payload

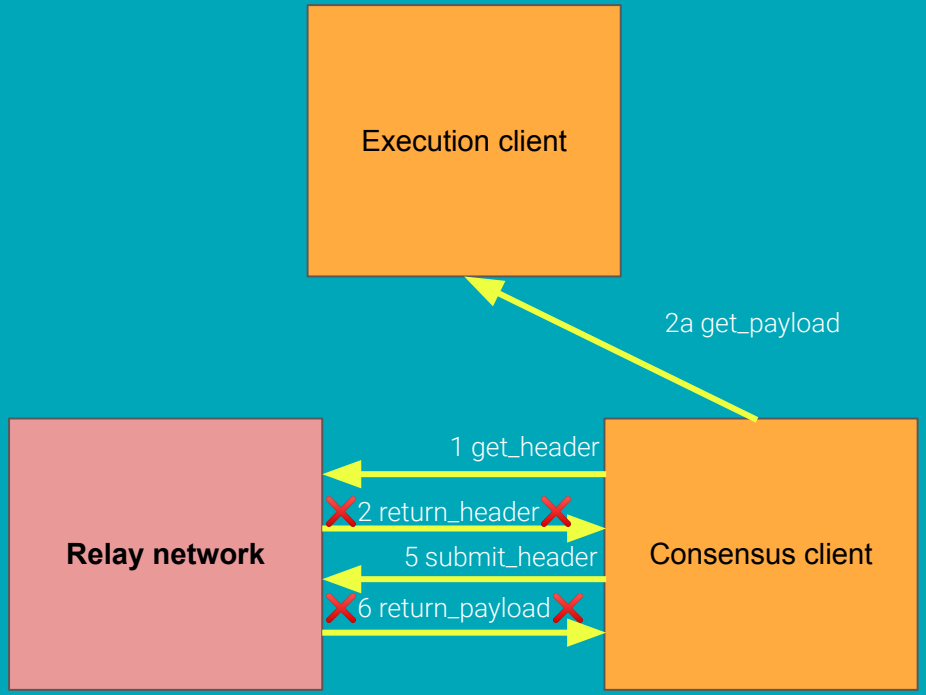
- Relay did not make the payload available
- Relay did not fulfill the commitment

Can consensus layer client detect?



Fallback to execution client

- If get header fails*
  - Produce with local execution client ✌️
- If get payload fails
  - Can't produce with local execution client
  - Can't double sign 🔪
  - Ops!! 💀



- return_header can fail **two** ways. Faults or timeout. Faults are better than timeout

Mainnet incident #1, Sep 16, Flashbots Relay

- Get payload fault
- Malformed payload
- Damage: **3** blocks missed

Post-mortem on the mev-boost getPayload bug when deposits were included (fixed in mev-boost v1.3.1 on Friday 16.9.)

The News Flashbots Status Updates post-mortem

metachris mate 4 6d

Summary & Impact

mev-boost had a bug in decoding the `getPayload` request body, which would fail if a deposit is included in the `SignedBlindedBeaconBlock` sent to the relay. We are aware of three instances where this bug caused a missed slot.

The issue was resolved in mev-boost `v1.3.1` 6.

Timeline

Friday, 16.9.

- Flashbots was notified at 9am UTC by Enrico del Fante 1 (from the Teku team) about a report of a failed `getPayload` call to mev-boost.

```
msg="could not decode payload (signed blinded beacon block)" error="json: unknown field "proof""
```

- The Flashbots team, together with Ethereum core dev community, began investigating and identified the issue shortly after.
- The cause of the error was that the `Deposit` type 1, which can be included in the `BeaconBlockBody` and `BlindedBeaconBlockBody` types, was missing the container structure with the `proof` field.
- The issue was initially fixed in go-boost-utils PR #38 4 and mev-boost v1.3.0 was released at 1pm UTC.
- A follow-up issue in the proof decoding was discovered and reported at 5pm UTC by Stefan Bratanov 4 (from the Teku team).
- The final solution was implemented in go-boost-utils PR #40 1 with the correct proof decoding. A big help was the inspiration by github.com/attestantio/go-eth2-client 1 (thanks @jgm 1) and the Prysm codebase to double-check and verify the implementation.
- mev-boost v1.3.1 6 was released with the final solution at 8pm UTC

Corrective and Preventative Measures

- We've improved automatic testing on both go-boost-utils and mev-boost, by using additional static analysis and linting tools as well as additional test vectors.
- We've compared the implementation, encoding, and hashing algorithms from go-boost-utils with those of github.com/attestantio/go-eth2-client 1 and Prysm, and can confirm that they are compatible.
- We've incorporated test-vectors from ethereum/consensus-spec-tests to both go-boost-utils and mev-boost. These test vectors include all possible fields (including deposits, slashings, etc.). These tests are run on every commit and pull request, which would catch any regressions if they were to happen.
- We've started an additional release checklist with steps taken before each release of mev-boost and go-boost-utils to provide additional security.
- We're continuing to invest into our security processes, both internally and with the help of external security advice.
- All mev-boost releases go through a signoff process with multiple parties, including node operators testing the latest release candidate on test networks.
- We're continuing to work closely with the core dev community on hardening mev-boost.

Mainnet incident #2, Sep 21, BloXroute Relay

- Get payload fault
- Consensus invalid payload
- Damage: **88** blocks missed

September 21, 2022

7:03 AM Eyal|bloXroute We would like to provide a short update as well as an apology in regards to the stability of our MEV services. There was a version of our relay released last night that included a bug in how we check blocks provided by our builders. In turn, this resulted in missed bids for select validators.

This version was deployed and tested in our staging environment as well as on chain testnets. The bug was not present in either environment and thus deployed to mainnet. We have since rolled back to a previous version and stability/performance is back to normal. Moving forward we will be changing the method of releasing new versions that will improve our abilities to catch bugs before they reach mainnet.

We are still investigating the root cause of the issue and will post again once this is clear.
Thank you for your ongoing support

3

Eyal 阿勇
@eyalmarkov

1/ Our MEV services were down for several hours due to a bug that was not identified in testing. We fully take the blame, and I would like personally to apologize to our validators who missed a slot because of it. We will learn from it and improve our services.

8:58 AM · Sep 21, 2022 · Twitter Web App

15 Retweets 8 Quote Tweets 123 Likes

Tweet your reply

Reply

Eyal 阿勇 @eyalmarkov · Sep 21
Replying to @eyalmarkov
2/ MEV boost is based on trust between validators and relays. Our validators trusted us, and we will honor all bids received from our relays. All missed slots will be paid today to the validators.

1 2 24

Eyal 阿勇 @eyalmarkov · Sep 21
3/ There were a total of 100 missed slots during the time of the downgraded performance and 88 of them were missed because of our service. We will pay the validators soon.

3 33

Mainnet incident #3, Sep 28, BloXroute Relay

- Get payload fault
- Consensus invalid payload
- Damage: 15 blocks missed

9:21 AM Benjamin | bloXroute @here An overdue update regarding the issues last week.

BloXroute Relays Delivering Invalid Blocks

Over the course of the day on 09/27 the bloXroute ethical relay was receiving and returning bad blocks to validators from one of our block builders running an experimental build strategy that resulted in missed slots.

Overview

This builder was improperly committing transactions and caused improperly formatted blocks. Because the relay was not simulating blocks from known builders these blocks were getting sent to validators causing them to miss slots due to proposing blocks with improper receipts or log blooms. A total of 15 slots were missed due to this issue.

Contributing Factors

BloXroute's relay was not simulating blocks from internal builders and was instead relying on these builders to submit valid blocks.

Resolution

This was resolved as quickly as we were able to identify where the issue was located.

The experimental builder was taken offline and ensured that it was no longer submitting blocks to any of our relays.

All of the validators who missed slot's were paid for the bids they received from the bloXroute relay.

Additionally:

On 09/30 all of the relays now simulate all block submissions regardless of source.

Timeline

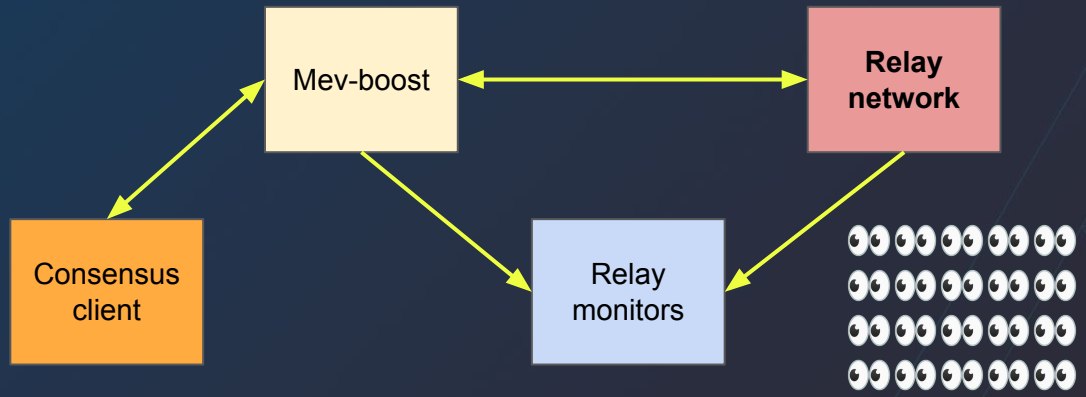
Slot first seen: 4790116
Slot last seen: 4791561

Mitigation: circuit breaker

- Beacon client to detect “liveness failure”
- Triggered by missing slots consecutively or period of time
  - Ex: missing 3 slots in a row or 8 out of 32 slots
- If triggered, default to local execution client
- Protect against dominant relay/builder malicious or offline

Mitigation: relay monitor

- Score relays based on **behavior** and **performance**
- **Behavior**: relay follows safety and liveness
- **Performance**: relay has good latency
- Relay monitor exports ratings API (i.e. scorecard)



Feature: bid filtering

- Beacon node to filter out bid amount
  - ex: --builder-profit-threshold <wei value>
- If bid is below threshold then default to engine client
- Unfortunately, the local value can't be retrieved at this moment until further engine-API support

Takeaways

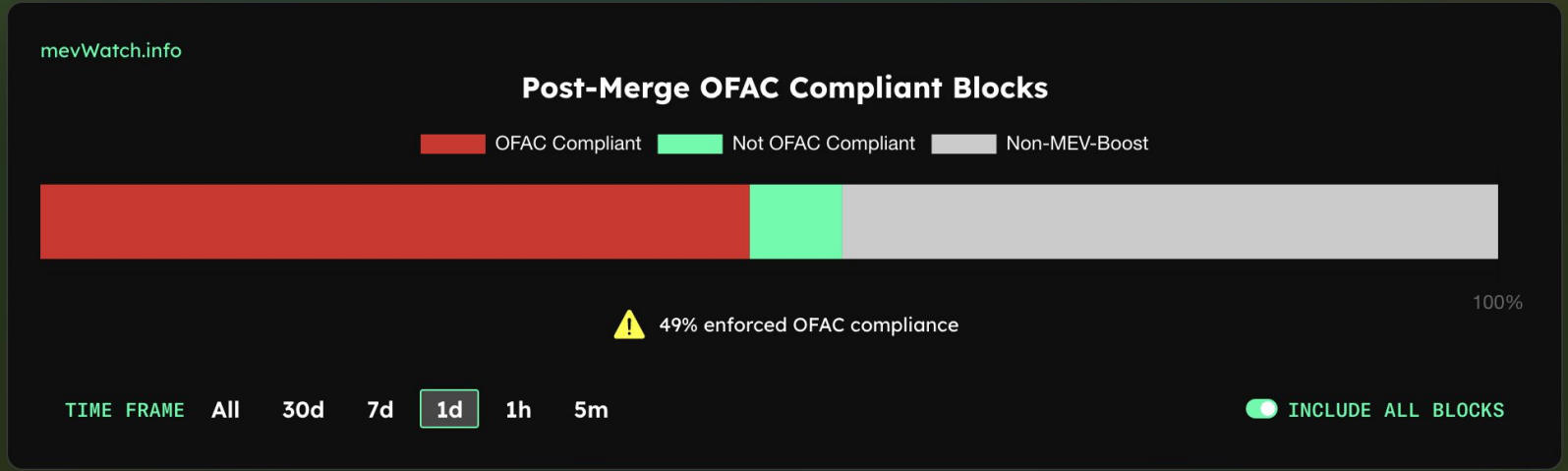
- Stil earlym but we need more robust relays
- We need to hold the relays accountable
  - **Idea**: Monitor missing or orphaning slots, poll relay APIs, tweet if relay misses a slot
- We need infra to monitor relays
- getHeader faults > getHeader timeout > getPayload faults
- Relay quality will improve overtime

Section 3

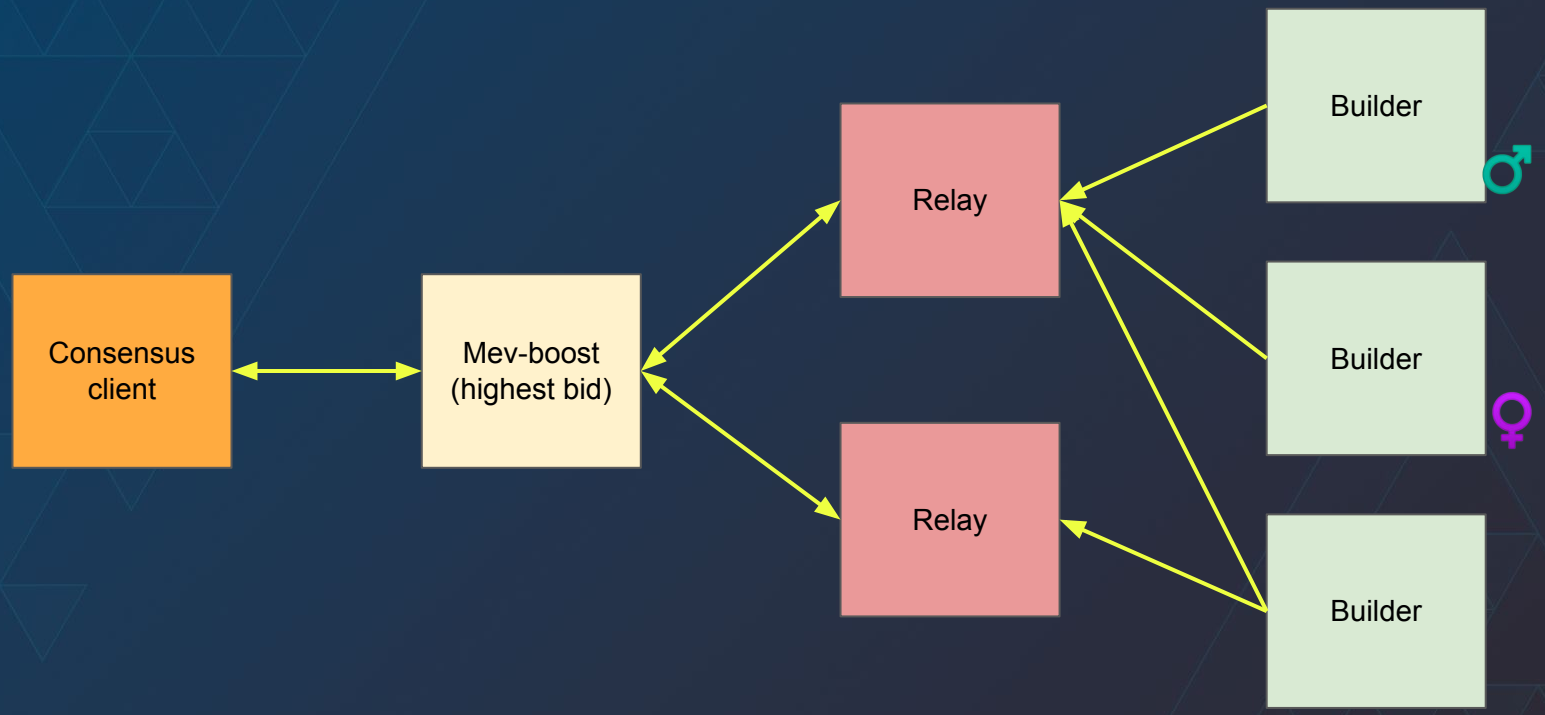
Risk: Censorship

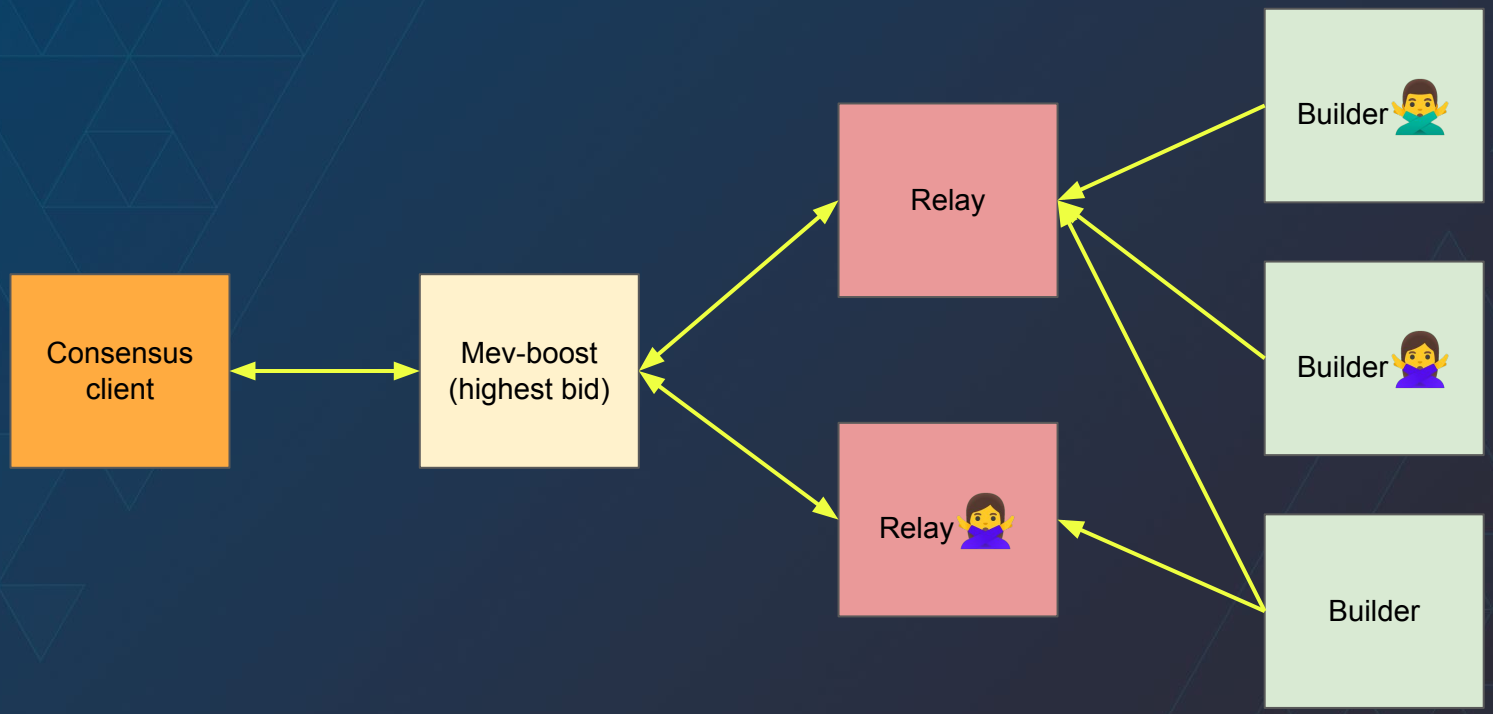
MEV Watch

Some MEV-Boost relays are regulated under OFAC and will censor certain transactions. Use this tool to observe the effect it's having on Ethereum blocks.



13 Oct 2022, https://www.mevwatch.info/


Consensus client
Mev-boost (highest bid)
Relay
Relay
Builder
Builder
Builder


Consensus client
Mev-boost (highest bid)
Relay
Relay
Builder
Builder
Builder

What are the problems?

- Mev-boost (relay proxy) is **neutral**
- The UX between mev-boost and consensus client is still **early**
- **Hard** for consensus client to filter out censored relay
- **Harder** to determine which relay is censoring at a given time

Potential solutions

- Active inclusion (mev-boost crList)
- Censorship filtering
- Censorship oracle

Harder

Easier

- **Warning**: these are experimental ideas

Active inclusion

- Proposer expresses intent to relays to force txs into payload
- Relayer presents constraint to block builder
- Proposer only accepts payload which txs are included or full block
- Requires multi-proof for included txs from relayer or builder
- Validation can be done on CL client or mev-boost or relay

Downsides: timing, latency, complexity, CL client implementation(mem pool, state), additional engine API requirement, trusting relay... etc

Censorship filtering

- Proposer monitors mem pool for top N gas txs at time M
- Relayer replied back header with proofs that top N gas txs are included or all the txs had higher gas
- If top N gas txs are not included, then CL client defaults to local engine

Downsides: CL client has to access mempool. How to synchronize M. Loses inclusion control

Censorship oracle

- Introduce a new actor to police censorship
- Ex: relay monitor
- Mev-boost receiving headers but also query oracle to filter out censored headers
- No change on CL client

Downsides: Putting trust to a new actor

Research #1

How much can we constrain builders without bringing back heavy burdens to proposers?

Proof-of-Stake

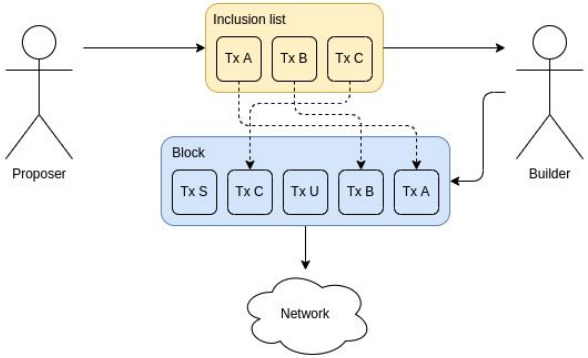
vbuterin 9h

One natural response to the risks of builder centralization (mainly censorship, but also various forms of economic exploitation) is to try to constrain the power that builders have. Instead of builders having full rein to construct the *entire* block if they win an auction, builders would have a more limited amount of power. This power should still be enough to capture almost all MEV that could be captured, and it should ideally still be enough to capture other benefits of PBS, but it should be weakened to limit opportunities for abuse.

This idea is sometimes called **partial block auctions**: instead of auctioning off the right to decide everything in a block, auction off the right to decide *some things*, where those "some things" could be much more nuanced than eg. "the builder chooses the first half of the block and not the second": you could give the builder the right to reorder, prepend, append, and you could even constrain the proposer. This post gets into some possible ways of doing this, and some of the tradeoffs that result.

Inclusion lists

In the inclusion list paradigm, a proposer provides an *inclusion list*, a list of transactions that they demand must be included in the block, unless the builder can fill a block *completely* with other transactions.



https://ethresear.ch/t/how-much-can-we-constrain-builders-without-bringing-back-heavy-burdens-to-proposers/13808

Research #2

Unbundling PBS: Towards protocol-enforced proposer commitments (PEPC)

Economics proposer-builder-separation

barnabe 9h

Many thanks to @fradamt @casparschwa @vbuterin @ralexstokes @nikete for discussions and comments related to the following post. Personal opinions are expressed in the first-person-singular.

Protecting the proposer and ensuring liveness of the chain are a big part of why PBS is considered to be moved into the Ethereum protocol. Ideally, when the proposer utilises the services of a builder, there is a contract between parties for the delivery of some goods (valuable blockspace), and the contract is honoured atomically:

- Either the contract fails to be made and the goods are not delivered/block content is not published, or
- The contract is successfully made and payment always succeeds, no matter what the party committed to supply the goods does.

This stands in contrast to MEV-boost [1], where a proposer could enter into a commitment with a relay, by signing a block header, after which the relay could fail to publish the block in time, and the proposer is not trustlessly compensated while missing the opportunity to make a block.

But with our version of in-protocol PBS (IP-PBS), we bind ourselves to a very specific mechanism for making these contracts, where there is trustless infrastructure for the proposer to sell off *entirely* their right of making the block. Amendments exist, such as inclusion lists [3], or increasing proposer agency by letting them build part of the block [4]. Still, few results exist showing that a proposer can be fairly unsophisticated and achieve most of the value their position confers upon them.

As an example, what if there is economic value for the proposer in selling the rights to make their block in advance [2], say 10 slots before? Under IP-PBS, a cartel of builders must honour an out-of-protocol market, where the winner of the blockspace future (perhaps auctioned at slot $n-10$) trusts the winner of the slot n IP-PBS auction to let them make the block. Yet the notion of an IP-PBS "winner" is semantically violated, and the value cannot be achieved by an untrusted proposer. Builder colocation with trusted proposers could also increase the delta between what IP-PBS returns to an unsophisticated proposer and what trusted proposers can achieve, beyond simple latency improvements.

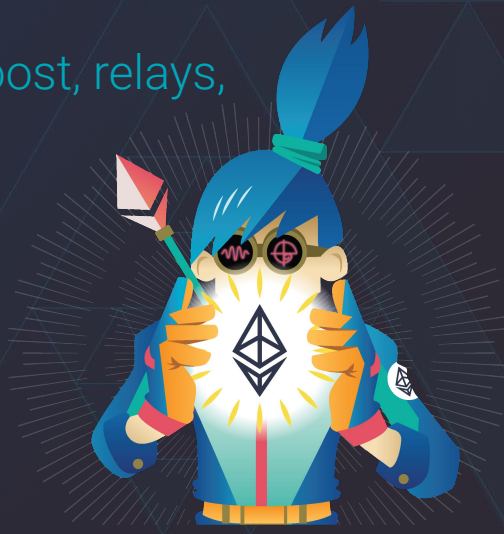
https://ethresear.ch/t/unbundling-pbs-towards-protocol-enforced-proposer-commitments-pepc/13879

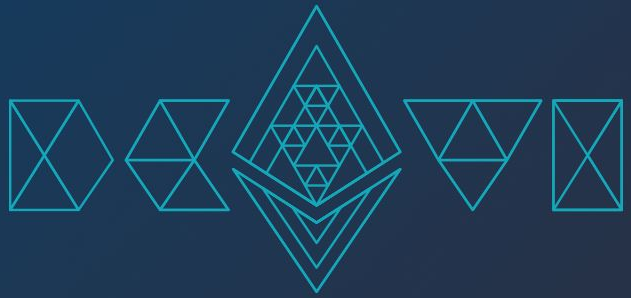
Takeaways

- **Who** can censor?
- **Who** can filter out censored txs?
- Use the **Builder API** and provide more ways for out-of-protocol markets to organise. Ex: proposer specifies inclusion list, block prefix, etc.
- **Spectrum** of solutions. **Simpler** solutions have **more trusted** assumptions

Final thoughts

- Censorship resistance is the **highest** priority asides from scaling and withdrawal
- Hybid-PBS is our best toolboxes, it allows iterations and we need more experiments before in-protocol PBS
- **Shoutout** to all the teams working on hybrid-PBS (mev-boost, relays, builders...etc)





Thank you

@terencechain