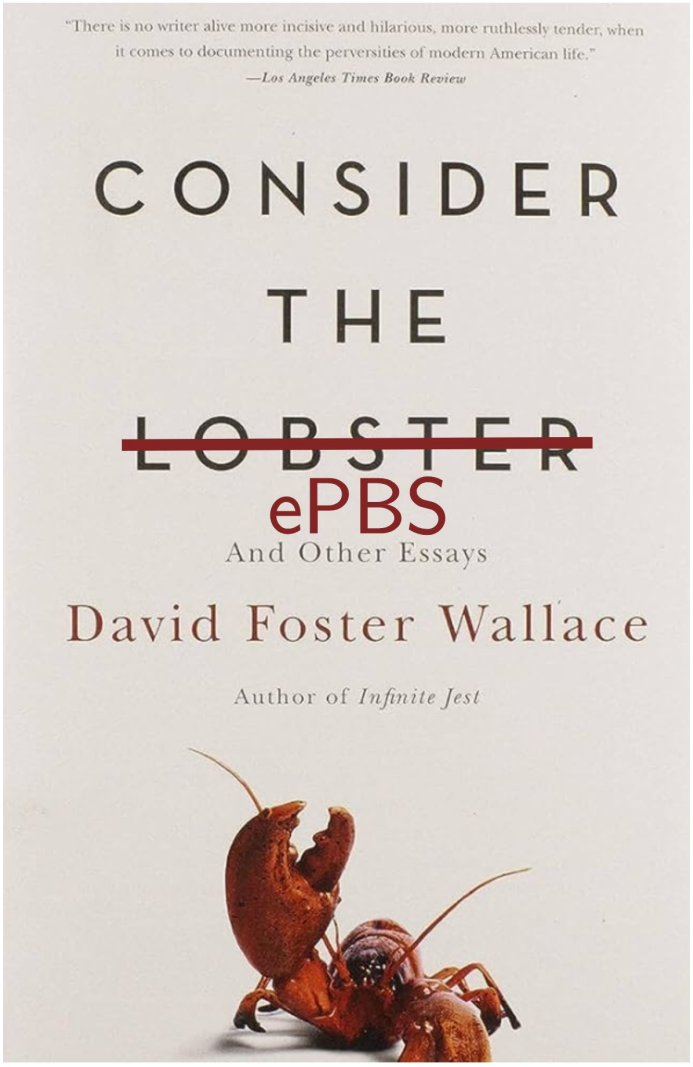
Consider the ePBS



$\cdot$

by mike – february 7, 2024

$\cdot$

tl;dr; *– Yesterday, after nine missed slots due to a bug in an optimistic relay, there was significant confusion around the role of relays and how ePBS would have changed the situation. This doc aims to reduce that uncertainty. The three important points to highlight:*

1. ***ePBS does not solve for builder bugs (the slots would've been missed regardless),***
2. ***best case ePBS – the proposer payment is faster and trustless, and***
3. ***worst case ePBS – bypassability reverts us to the same trust model as today.***

It follows that the only reason yesterday's event should change your mind about ePBS is if you think the proposer refund was too slow, and even then, you must acknowledge that in the worst case, the same refund process would have to occur in ePBS.

$\cdot$

related work

- *optimistic relays*
  - *Optimistic relay proposal*
  - *Towards enshrined PBS – An optimistic relay roadmap*

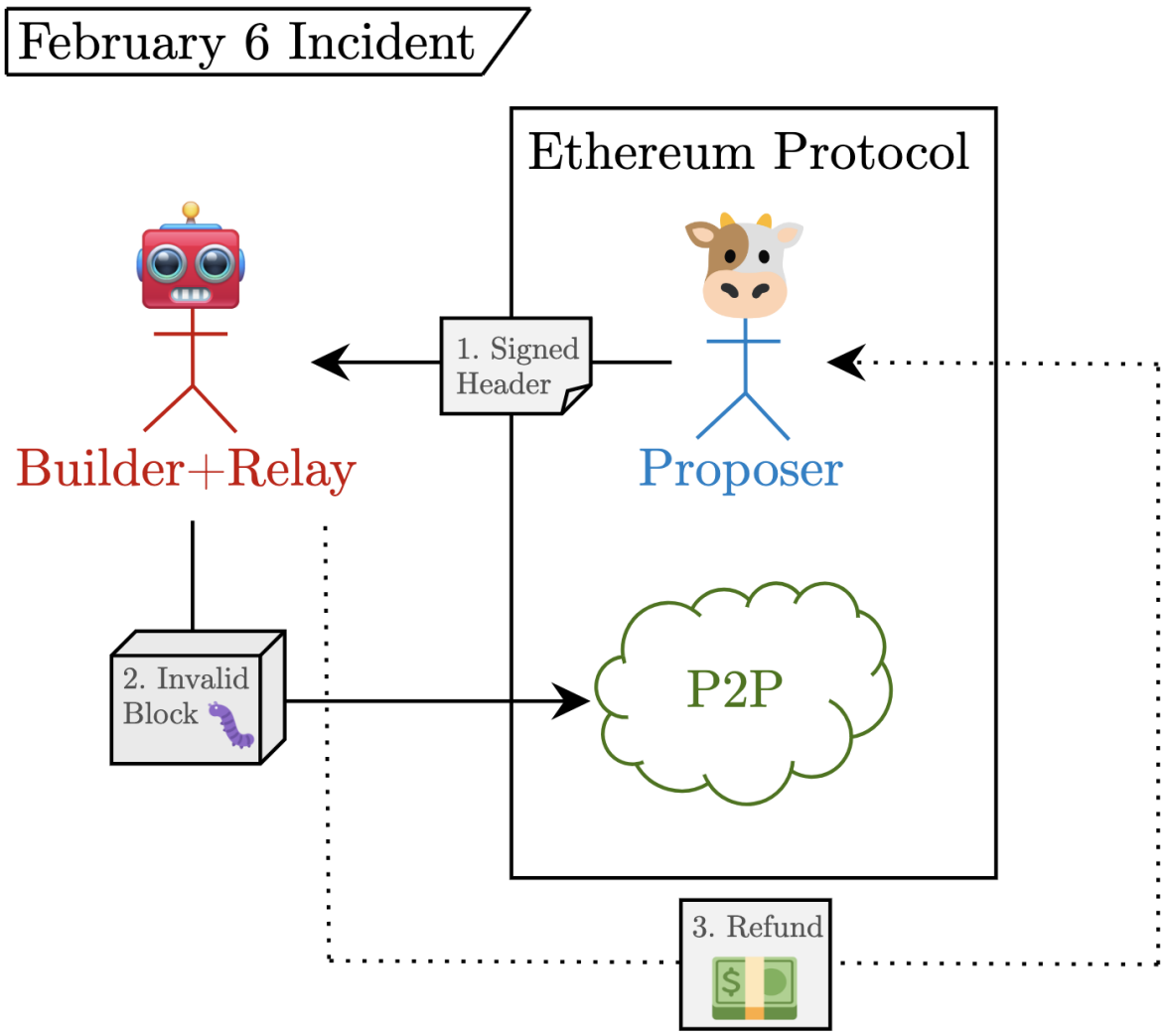
- *An optimistic weekend*
- *Optimistic relays and where to find them*

- *ePBS*
  - *Why enshrine proposer-builder separation*
  - *Payload-timeliness committee – a new ePBS design*
  - *Relays in a post-ePBS world*
  - *ePBS – the infinite buffet* (contains many additional links)

$\cdot$

February 6th Incident

Yesterday, a bug in the bloXroute max profit relay caused nine missed slots. The relay ran in "optimistic" mode and received invalid blocks from the rsync builder. Because the relay had an error, the invalid blocks were broadcast over the network (in the non-buggy optimistic case, only a single invalid block would have occurred). The Ethereum network reacted accordingly, with the nodes rejecting the invalid blocks, resulting in empty/missed slots. Optimistic relaying has been covered ad nauseam (see [1, 2, 3, 4]), so we won't discuss the specifics here. Instead, we are concerned with how ePBS fits into the picture. The figure below captures the incident in its most distilled form.

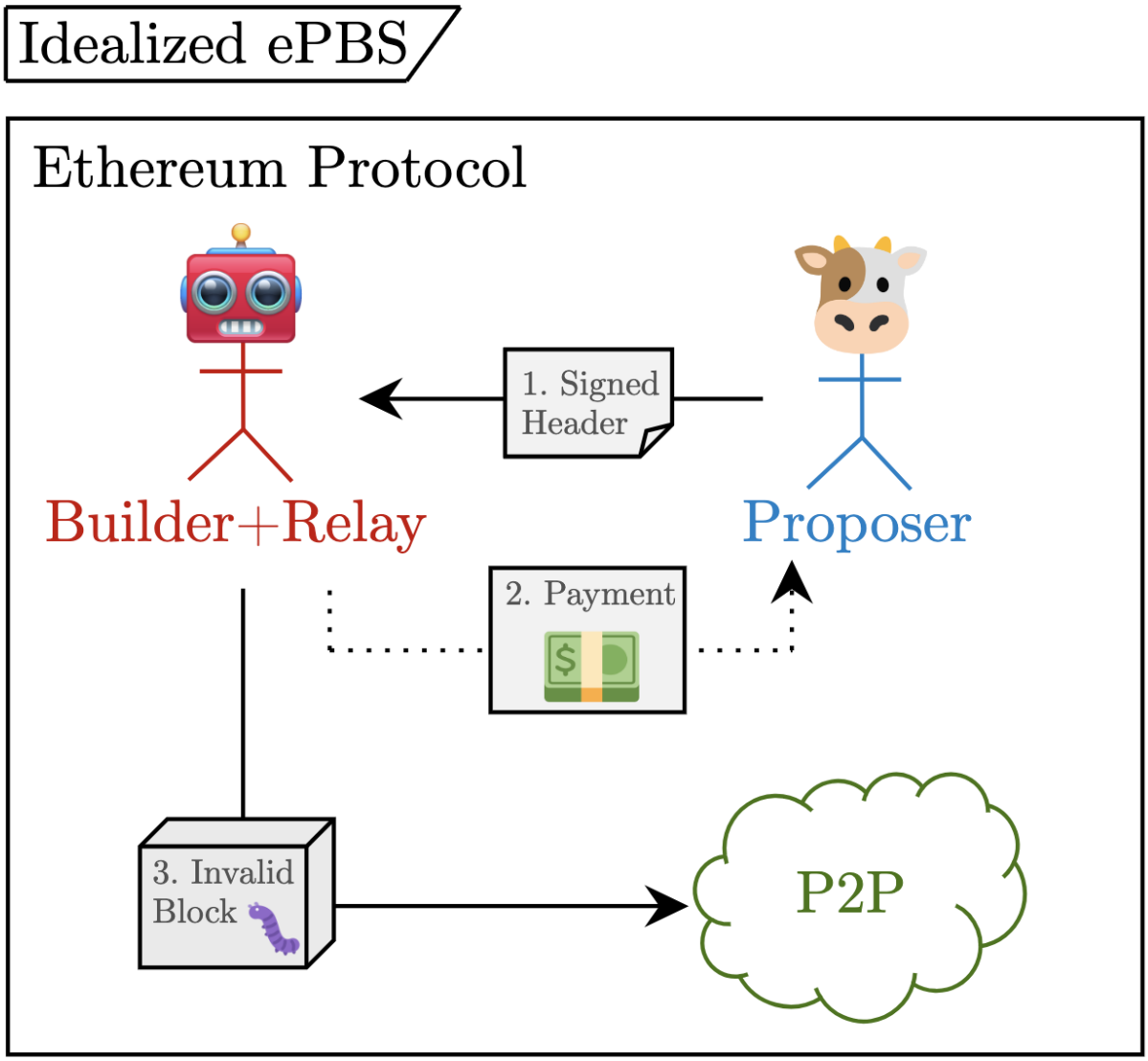


1. *The Proposer signs the header sent to them from the Builder+Relay.* This is done using mev-boost. Note that for this document, we treat the builder and relay as a single entity.
2. *The Builder+Relay publishes the invalid block to the network.*
3. *The Builder+Relay refunds the Proposer for the missed slot.*

Note that Step 3 here happens "out-of-band". In this case, it was a few hours before the proposers were refunded, but this procedure could be automated if there was a strong demand for immediate refunds.

Idealized ePBS

Let's play out the same situation with ePBS. A core primitive of an ePBS design is an "unconditional payment" mechanism. Once the proposer signs a bid from a builder, they are guaranteed to be paid the amount in the bid, no matter what the builder does. The diagram below shows that the Builder+Relay is within the Ethereum protocol alongside the Proposer. This *does not* prevent the Builder+Relay from submitting invalid blocks to the network. The only difference is that instead of a "refund" for the missed slot, the Proposer receives an upfront "payment" for the sale of their slot.



1. *The Proposer signs the header sent to them from the Builder+Relay.* The communication could occur through mev-boost or some other fast networking layer to avoid latency from p2p gossip.
2. *The Builder+Relay pays the Proposer for their slot.*
3. *The Builder+Relay publishes the invalid block to the network.*

The Proposer can trustlessly ensure they will be paid in full because the unconditional payment mechanism is within the protocol; the payment happens "in real time" (during the slot).

NOTE: *From the protocol's perspective, the only difference between the two cases above is the latency between the missed slot and the payment/refund.*

Bypassing ePBS

My main critique of ePBS is the bypassability it admits; we don't have a way to enforce that the Proposer only uses the protocol to outsource block production. Generally, there are two ways that ePBS aims to provide the unconditional payments we discussed in the previous section:

- using a "just-in-time" payment mechanism so that a payment accompanies each bid (e.g.,Top-of-Block payments), or
- requiring in-protocol builder collateral (e.g., the builder is required to stake).

Let's return to bypassability as it pertains to each of these versions.

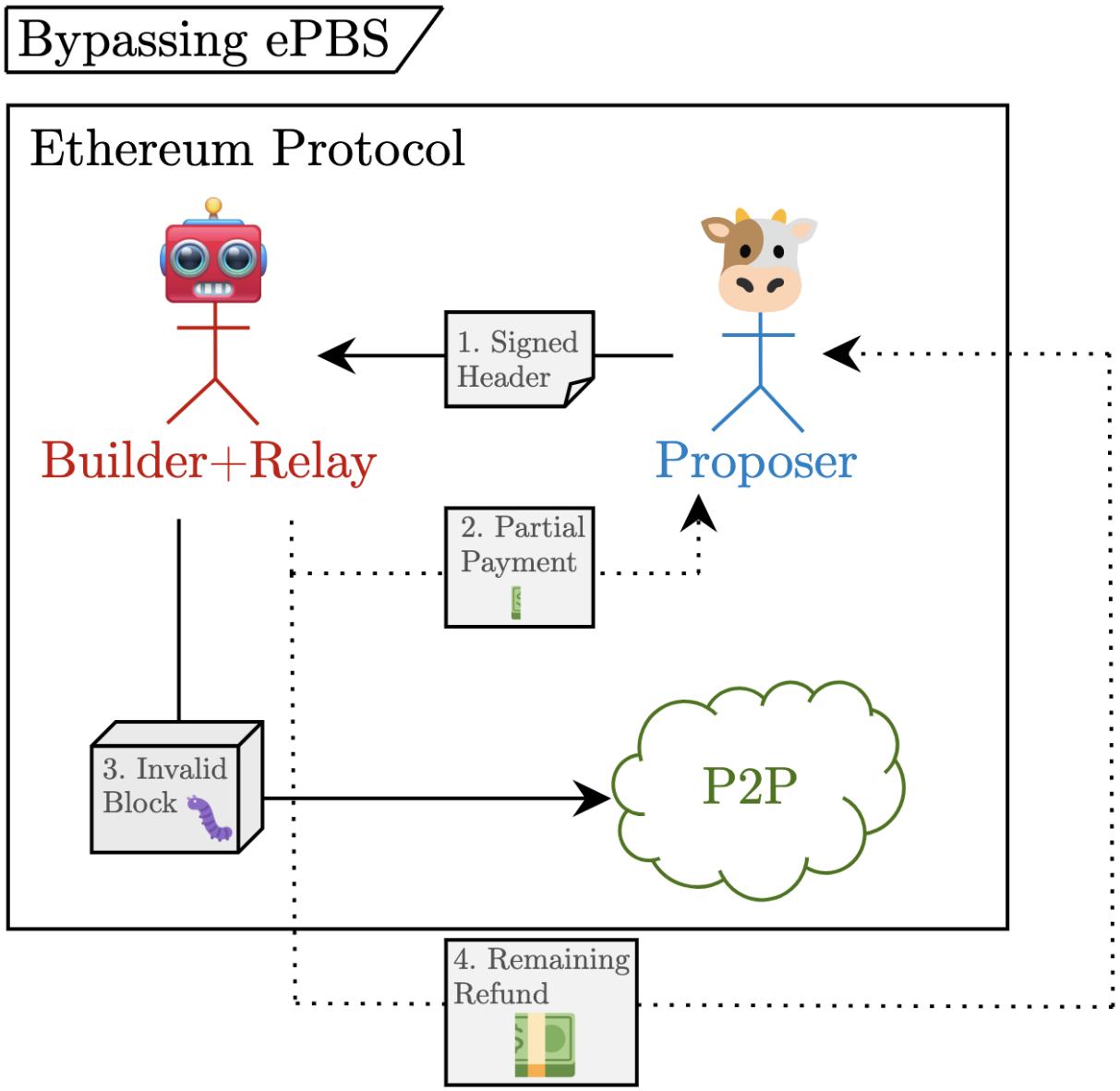
- *JIT payments* – Because these payments need to be valid at the top of the block, the builder must have liquidity up front to make this payment. Instead, the builder could continue to use mev-boost (and set the JIT payment to zero), leveraging the existing trust path to avoid paying at the top of the block.
- *In-protocol builder collateral* – The staked builder could continue usingmev-boost to relay bids to the proposer. While the proposer may choose to only listen to "real value bids", those backed by protocol collateral, the builders may send 0 value bids through the protocol and continue paying the validator in a trusted way.

  Aside: It may well be that proposers strongly prefer only committing to "trustless" bids resulting in wide adoption of the in-protocol builder collateral ePBS, but I have other issues with the builder collateral model. Namely,

  - collateral size choice (too high of a value is centralizing and capital inefficient, but too low of a value may not fully capture the bid values of high-MEV blocks),
  - top-up mechanics (if the proposer payments come from the builder collateral, regular new deposits are required to maintain the builder balance; this seems clunky and inefficient in terms of many beacon chain updates), and
  - no slashing conditions (the builder stake just kinda sits there and gets drained and refilled; there is no enforcement mechanism).

  I'll leave this discussion for another day.

Now returning to the pictures...



1. *The Proposer signs the header sent to them from the Builder+Relay.* The communication could occur through mev-boost or some other fast networking layer to avoid latency from p2p gossip.
2. *The Builder+Relay partially pays (first alliteration) the Proposer for their slot using the in-protocol unconditional payment.*
3. *The Builder+Relay publishes the invalid block to the network.*
4. *The Builder+Relay executes the remaining refund (second alliteration) out-of-protocol.*

Now we are back to where we started. Step 4 happens "out-of-band".

NOTE: *In either unconditional payment mechanism, bypassability could result in the reversion to the existing trust model of today, where proposers trust builders for their accurate payment.*

Conclusion

This is all you need to take away:

1. ***ePBS does not solve for builder bugs (the slots would've been missed regardless),***
2. ***best case ePBS – the proposer payment is faster and trustless, and***
3. ***worst case ePBS – bypassability reverts us to the same trust model as today.***