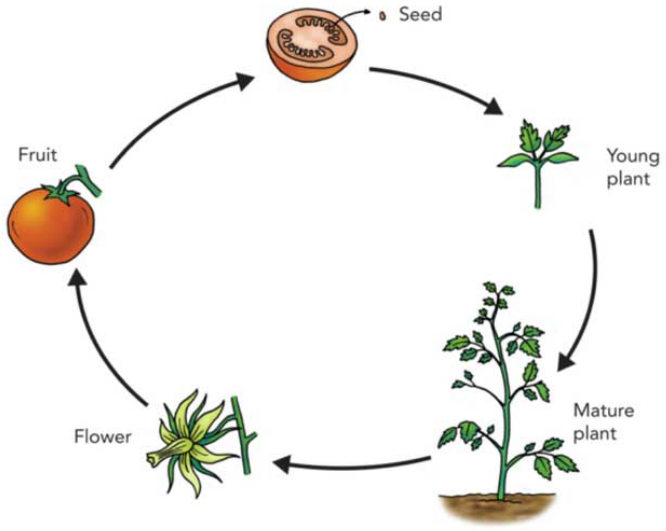
Inclusion lists: execution, consensus, & engine spec overview



$\cdot$

by mike based on discussions with Potuz, Terence, Francesco, Hsaio-Wei, Danny, Tim, Roman, Dan, Alexey, Lion, Mikhail, & Matt.

$\cdot$

february 7, 2024

$\cdot$

***tl;dr;** -- This document presents a minimal view of what a potential spec forEIP-7547 could look like. We outline changes to the execution specs, consensus specs, and execution APIs to clarify how the pieces fit together. These changes are in draft mode; the goal of this document is to*

*(i) convey the relative simplicity of the changes,

(ii) highlight the key design considerations that came up in discussion.**

The main non-goal of the document is to present the "final-form specification," because the design is continuing to evolve. Rather this exercise aims to be maximally simple \~from the specification\~ perspective and highlight the portion of the specs where changes will be. There may be engineering reasons to add/modify/remove some of what is presented below, and some of these discussions are laid out in the "key design considerations" section.

Note that inclusion lists are mainly useful in the context of PBS, where the proposer wants to externalize their block-building while retaining some agency over the transactions in the canonical chain. If the proposer is building their block, they can instead choose to insert the inclusion list transactions directly into their block.

$\cdot$

related work

Specs

| repo | draft pr link |

| -- | -- |

| execution-specs | https://github.com/michaelneuder/execution-specs/pull/1 |

| consensus-specs | https://github.com/hwwhww/consensus-specs/pull/2 |

| execution-apis | https://github.com/michaelneuder/execution-apis/pull/1 |

Execution specs – [link to draft PR]

We update the `state_transition` function (called when validating a block) to remove entries from the summary that are satisfied in the parent block:

```python=
def state_transition(chain: BlockChain, block: Block) -> None:
...
parent_transactions_addresses = [decode_transaction(tx).address for tx in parent.transactions]
inclusion_list_summary = [entry for entry in block.inclusion_list_summary
                          if entry.address not in parent_transactions_addresses]
...
```

We also modify the `apply_body` function (called from `state_transition`) to calculate the added gas from the inclusion list and assert that the inclusion list entries are satisfied.

```python=
def apply_body(
...
inclusion_list_summary: Tuple[InclusionListSummaryEntry, ...],
):
...

# Calculate the additional gas added by the inclusion list and construct
# a map of addresses is required.
inclusion_list_gas = sum(entry.gas_limit for entry in inclusion_list_summary)
inclusion_list_addresses = {entry.address: False for entry in inclusion_list_summary}
gas_available = block_gas_limit + inclusion_list_gas

...

if sender_address in inclusion_list_addresses:
    # Mark this address as satisfied
    inclusion_list_addresses[sender_address] = True

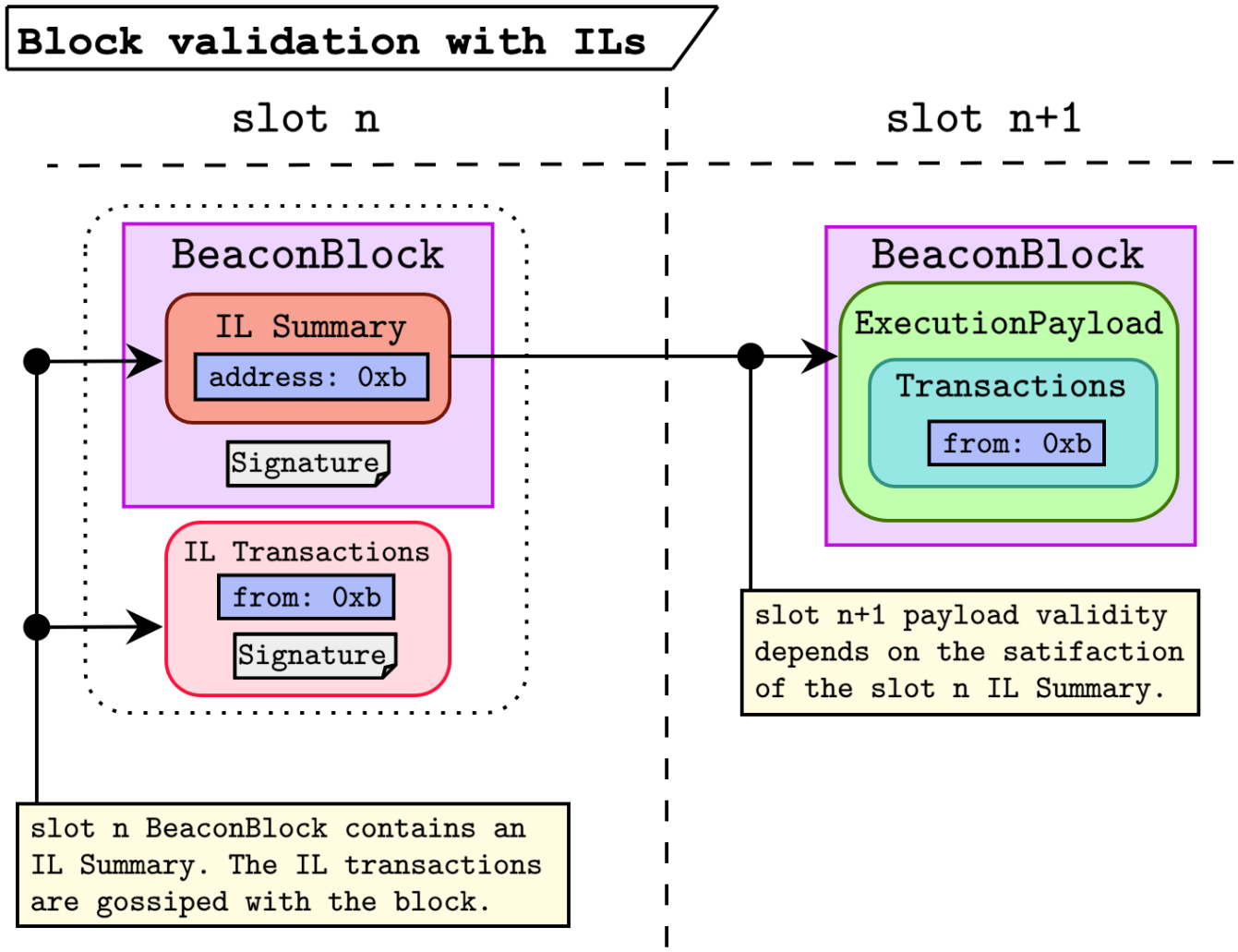
...

# If any inclusion list addresses are not satisfied, the block is invalid.
for entry in inclusion_list_addresses:
    if inclusion_list_addresses[entry] is False:
        raise InvalidBlock

...
```

```

The figure below encapsulates the full block validation flow:




### Consensus specs – [link to draft PR]

The main beacon chain spec (specs/_features/eip7547/beacon-chain.md) has extremely minimal changes.

- Define the InclusionListSummaryEntry and InclusionListSummary

```python=

class InclusionListSummaryEntry(Container):

address: ExecutionAddress

gas_limit: uint64

class InclusionListSummary(Container):

slot: Slot

proposer_index: ValidatorIndex

summary: List[InclusionListSummaryEntry, MAX_TRANSACTIONS_PER_INCLUSION_LIST]

```

- Add InclusionListSummary to the BeaconBlockBody

```python=

class BeaconBlockBody(Container):

...

inclusion_list_summary: InclusionListSummary  # [New in EIP7547]

```

The fork-choice spec (specs/_features/eip7547/fork-choice.md) only updates the PayloadAttributes type to include the inclusion list transactions that are passed from the CL to the EL when block construction is initiated. Other engine APIs use the updates ExecutionPayload type (covered in more detail in Execution API below).

```python=

class PayloadAttributes(object):

...

inclusion_list_transactions: List[Transaction, MAX_TRANSACTIONS_PER_INCLUSION_LIST]  # [New in EIP7547]

```

The p2p spec (specs/_features/eip7547/p2p-networking.md) creates the signed transactions list type.

```python=

class SignedInclusionListTransactions(Container):

transactions: transactions: List[Transaction, MAX_TRANSACTIONS_PER_PAYLOAD]

signature: BLSSignature

```

It also updates the type of the gossip object for the beacon_block topic.

```python=

class SignedBeaconBlockAndInclusionList(Container):

signed_block: SignedBeaconBlock

signed_transactions: SignedInclusionListTransactions

```

The block is transmitted with the transactions because block validation requires the existence of an inclusion list.

The validator spec (specs/_features/eip7547/validator.md) updates the proposer behavior to construct and broadcast the inclusion list transactions alongside the block.

```python=

Proposer must construct and broadcast SignedInclusionListTransactions alongside SignedBeaconBlock.

```

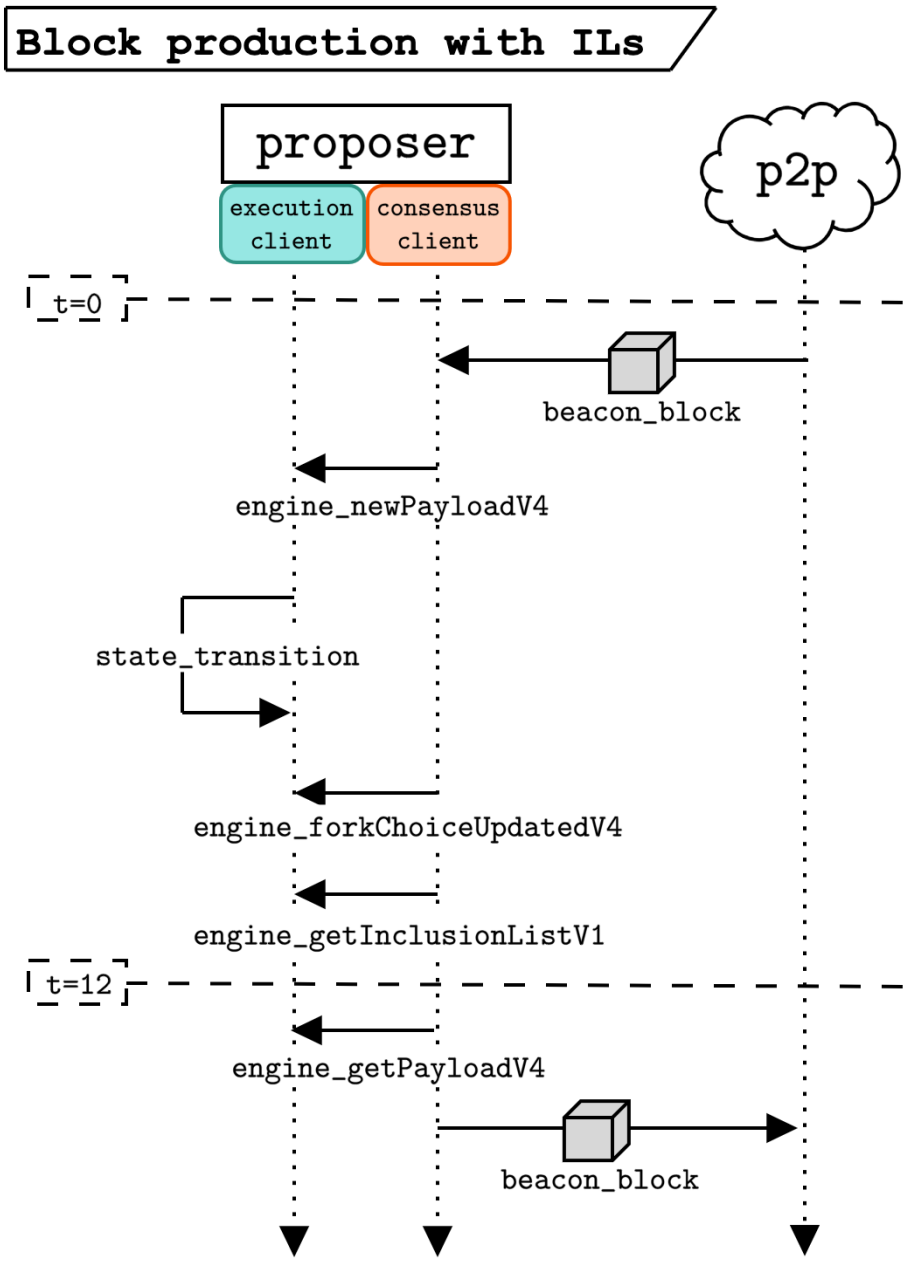
Execution API – link to draft PR

We modify the engine API with a new API and three modified APIs. Recall that the point of inclusion lists is to allow validators to flag transactions for inclusion even when outsourcing their block production. In mev-boost (out-of-protocol PBS), the proposer may need to communicate their inclusion list details to the builder through the relay.

- engine_newPayloadV4 (updated API)
    - CL requests validation of a block from the EL. Updated with inclusion list types and validation requirements.
- engine_forkChoiceUpdatedV4 (updated API)
    - CL notifies the EL of a new head. If the payload attributes are supplied, the EL begins building a block. The payload attributes type is updated to include the transactions.
- engine_getInclusionListV1 (new API)
    - CL requests the construction of an inclusion list from the EL.
- engine_getPayloadV4 (updated API)
```

- CL requests the current best block from the EL. Updated with inclusion list types.

The sequence diagram below shows the block production flow, using each of the above APIs.



Builder & Relay API

- Inclusion lists will likely change the builder API and the relay API. Leaving this out for now until the exact specifics are ironed out.
  - https://ethereum.github.io/builder-specs/#/Builder
  - https://flashbots.github.io/relay-specs/#/Data

Key design considerations

- ***Conditional vs unconditional enforcement of inclusion lists.***
  - See unconditional ILs for tradeoff discussion.
  - In the case of unconditional enforcement, should the IL transactions extend the gas limit beyond 30mm? Or require the transaction inclusion within the 30mm gas?
- ***Top-of-block vs anywhere-in-block***
  - Where in the $\text{slot } n+1$ block should the $\text{slot } n$ IL transactions go?

    - Top-of-block is more MEV risk (slot-specific MEV is capturable by getting in the inclusion list). There exist differing views on how much risk this induces.

- ***Gossip inclusion list with the block or not?***
    - By gossiping them together over the `beacon_block` topic (as described in this document), there is no chance of a validator seeing the block but no inclusion list. It is also simpler from the specification perspective.
    - Engineering considerations may make separating them over different gossip channels preferable.
        - We already have a lot of infra for sidecars with 4844.
        - Rejecting duplicate blocks over the wire is easier with them separate given there may be multiple ILs.
- ***Gas limit & transaction limit to constrain the inclusion lists?***
    - The IL needs to be limited in some way (if it is unconditional). Right now we have both a limit on the number of transactions in the IL (16) and the amount of gas in the IL (2mm). We should consider these numbers and the who/when/where of their enforcement.