Taiga shielded state transitions are computed prior to ordering (typically client-side or offline) and then verified (via SNARKs) post-ordering. This is primarily because shielded state is not publicly visible, but also this state transition model can provide some scalability and security benefits since the state transition only needs to be verified and not re-computed.

However, at least sometimes it is not possible to compute the state transition before ordering. Some example applications might include AMMs or bridges. The primary problems happen when:

1. The state transition depends on the result of ordering, and/or
2. The state transition updates globally addressable shared state

For example, if we have a public global counter which shielded state updates can increment or decrement, then each shielded execution cannot individually update this state because the prior state (from a previous block or previous transaction within the same block) would not have been available at execution time (pre-ordering).

So the orderer must be able to compute a post-ordering execution with input state determined only once ordering is finalized, and apply the final state once computed. The input and output state and the computation must be public, since it is not known in advance who the orderer might be. The requirement to update globally addressable shared state is also different than the shielded execution's state model. Finally, the last requirement is rollup efficiency, requiring that the post-ordering execution be circuit friendly.

There are several potential options for the post-ordering execution model, where the orderer runs a computation and:

1. replicates across other {validators, full nodes} in a transparent execution model that can read/write addressable state
2. proves its validity, in a transparent (circuit) execution model that can read/write addressable state
3. proves its validity in exactly the shielded execution model, but with special shielded state representation of global addresses

I think (1) is undesirable because of the complexity of adding an entire VM (or similar execution environment) solely to compute post-ordering execution.

In some ways (3) is theoretically ideal, since it would depart the least from the existing shielded execution model and thus require the fewest modifications and additions. However, it may have some practical problems: shielded predicates take 2 inputs (which may be insufficient) and also the latency of proving in the shielded environment may be more than acceptable for an orderer.

The best option might be (2), where a slightly modified execution model (but still substantially similar) can be used. Predicates would have a different API signature (perhaps more inputs) and the Action circuit can be drastically simplified (no merkle tree, no VP commitments, etc) which should help latency substantially.