Turbo-Geth/Silkworm architecture plans

Modularity that falls out naturally from solving problems

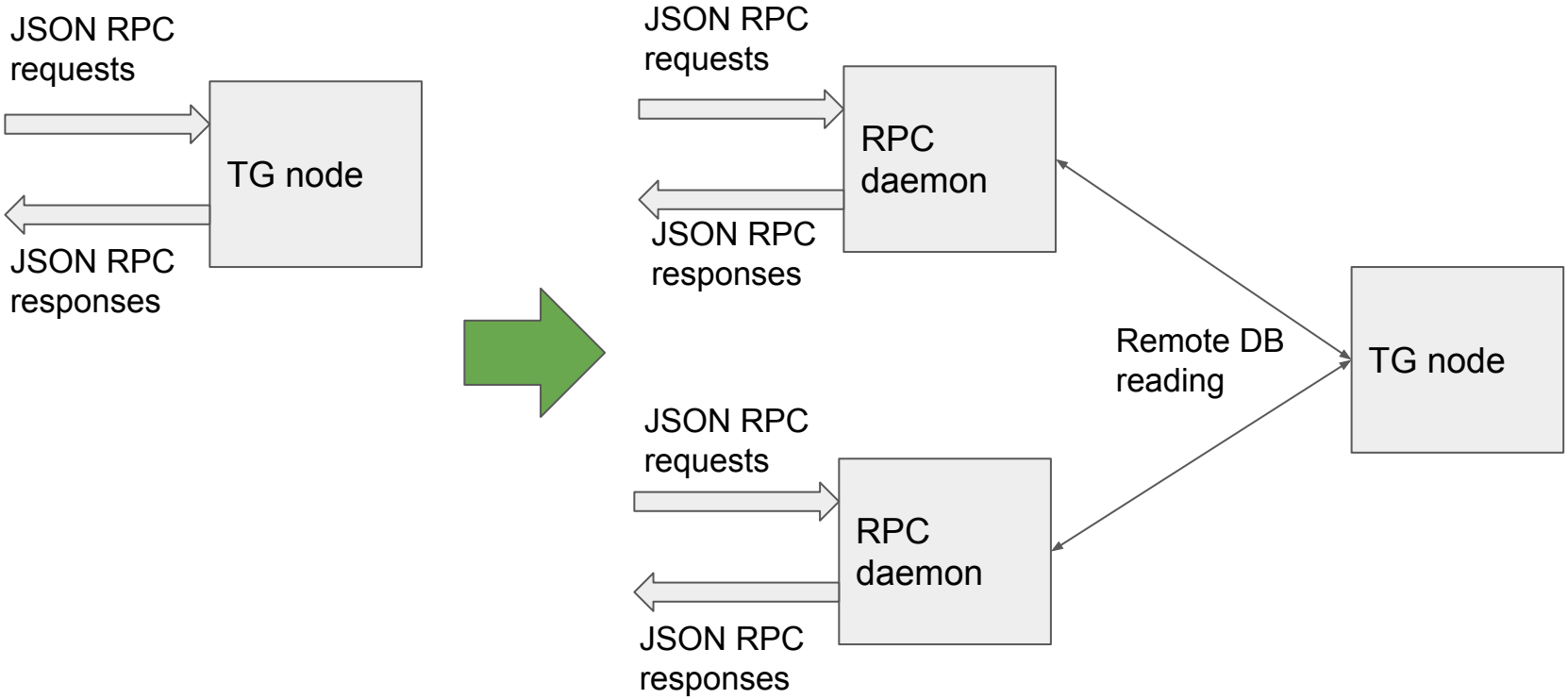
Start with monolith, then split it up

Many projects start out as “modular”, or “more modular” implementations of something.

However, in the beginning, it is often not clear what modules and components one would need.

Therefore, it is often best to actually start with a monolithic implementation, make it work, and then make modules where they start naturally bring value.

RPC daemon split (done)



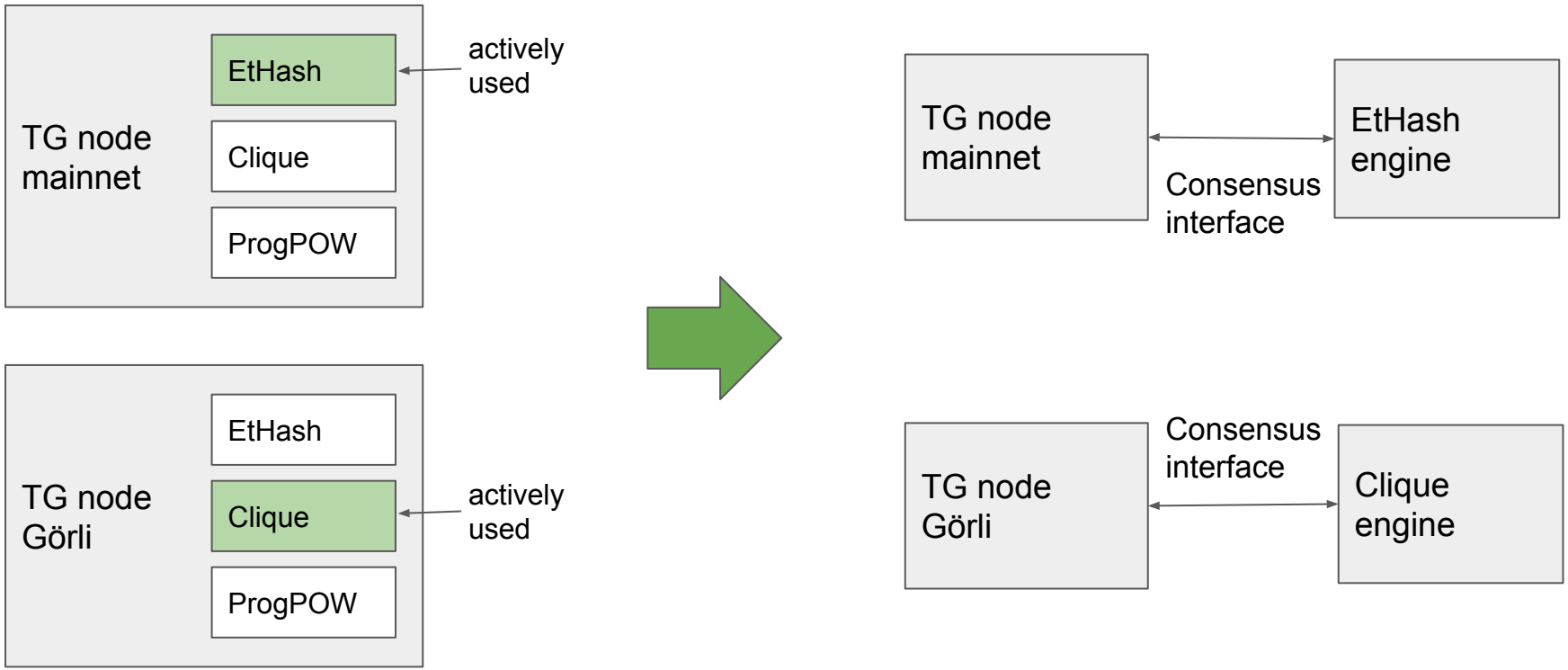
RPC daemon split - reasons

Performance/Scalability. Processing of most JSON RPC request is CPU bound. It is easier to add capacity if RPC processing can be done on a different machines from where the node is

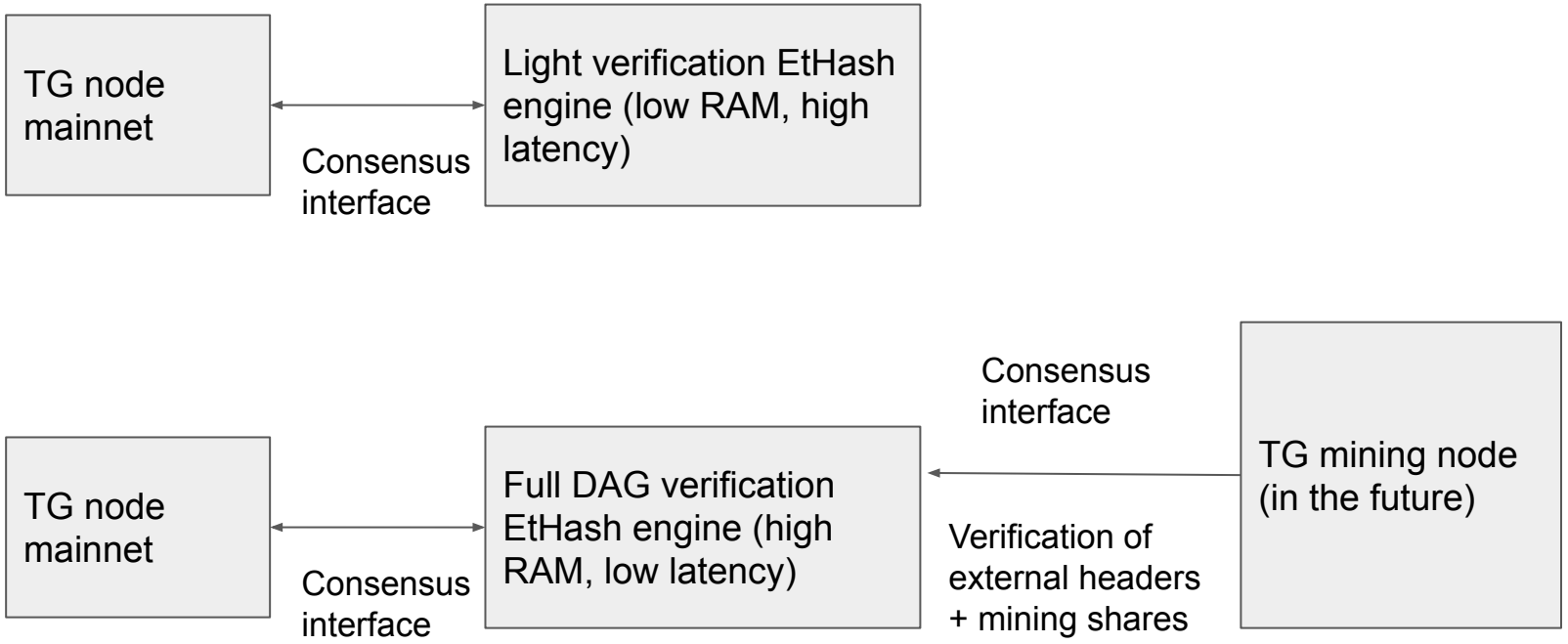
Permissionless development. Remote DB access is quite generic and allows writing code for most of the JSON RPC requests without any code changes on the TG node. It means that third party developers can create their own RPC methods, standards, and implement them. Only rarely they need to coordinate with TG node developers.

Diversity. It is easier to implement RPC daemons with different compatibilities (to geth, open ethereum, etc.)

Consensus engine split (in progress)



Consensus engine split - possibilities



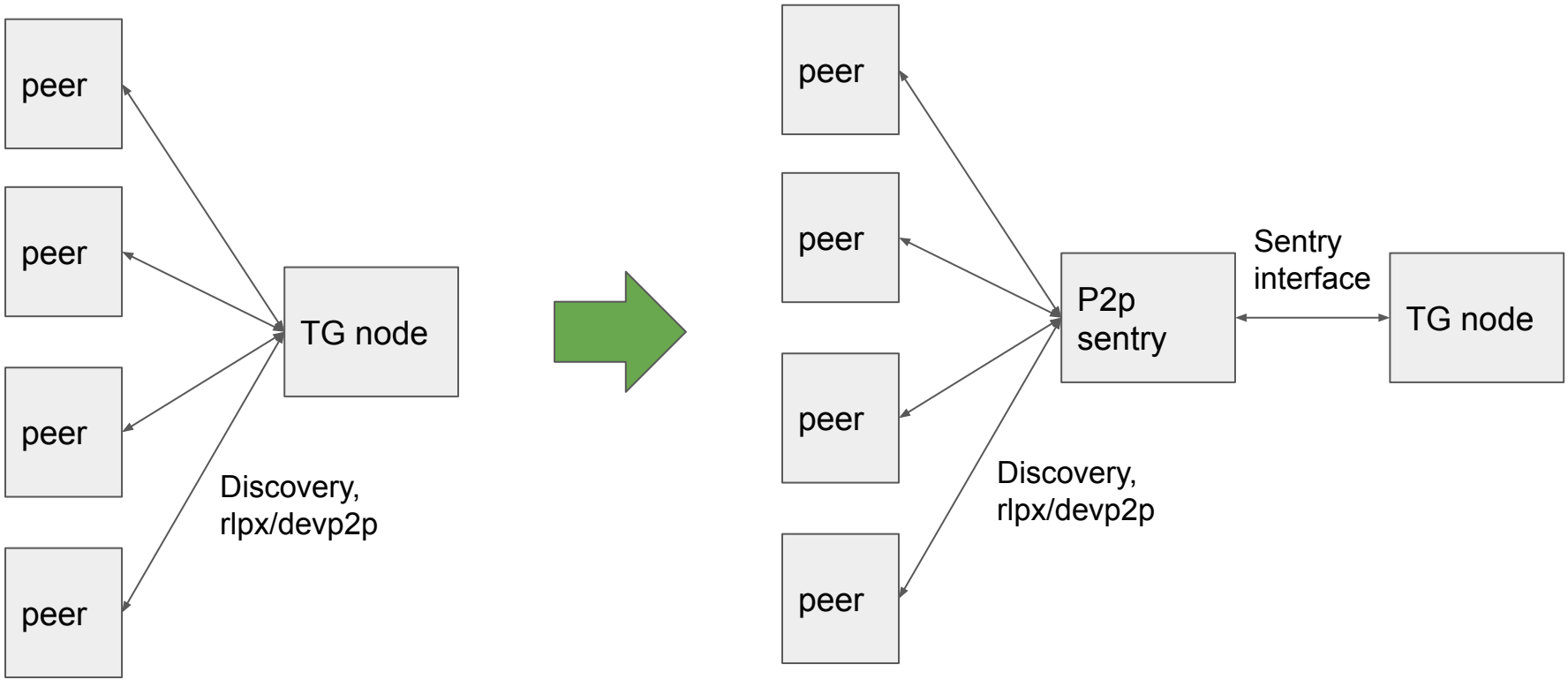
Consensus engine split - reasons

More efficient development. After the split, somebody can fully concentrate on making the best engine possible for given consensus algorithm.

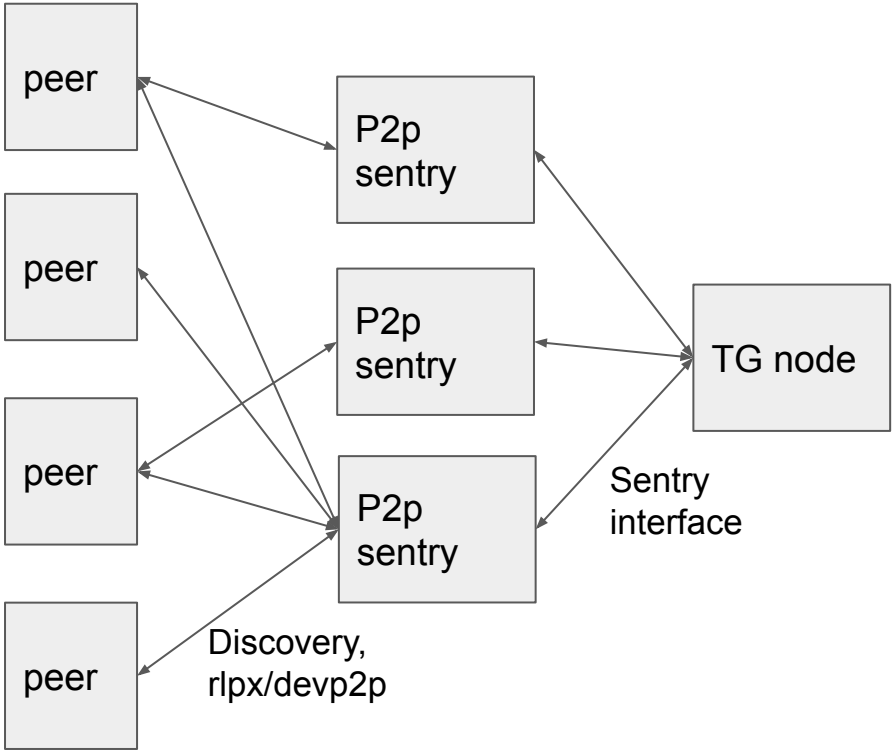
Performance/Scalability. Enables running full DAG verification engine on a separate machine, can also connect multiple TG nodes to the same Consensus engine. Mining nodes can use it to also verify mining shares.

Permissionless development/Diversity. As long as the “consensus interface” fits, anyone can develop alternative Consensus engines for TG without having to coordinate with TG developers.

P2p sentry split (in progress)



P2p sentry split - possibilities



Better connectivity for cheaper (sentries are lightweight)

Cheaper DOS, de-anonymisation, and eclipsing protection

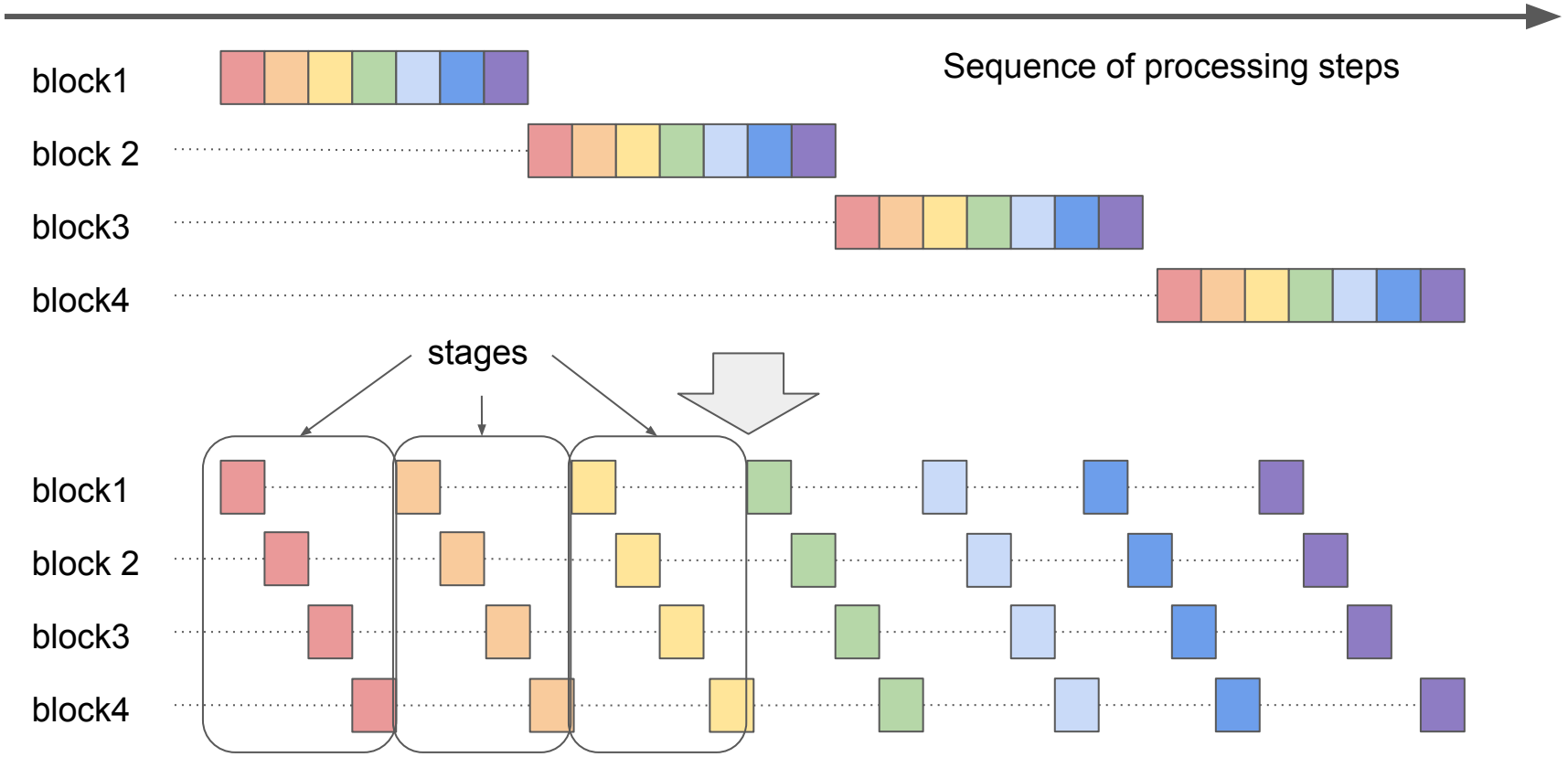
P2p sentry split - reasons

More efficient development. Peer discovery, peer management, RLPX and managing of sub-protocols becomes the job of the sentry. Someone can fully concentrate on making the best sentry possible.

Connectivity/Protection. Connecting TG node to multiple sentries allows cheaper way of obscuring the location of the node, maintaining better connectivity (which is crucial for mining nodes).

Permissionless development/Diversity. As long as the sentry interface fits, alternative sentries can be developed without coordination with TG developers.

Staged sync (done)



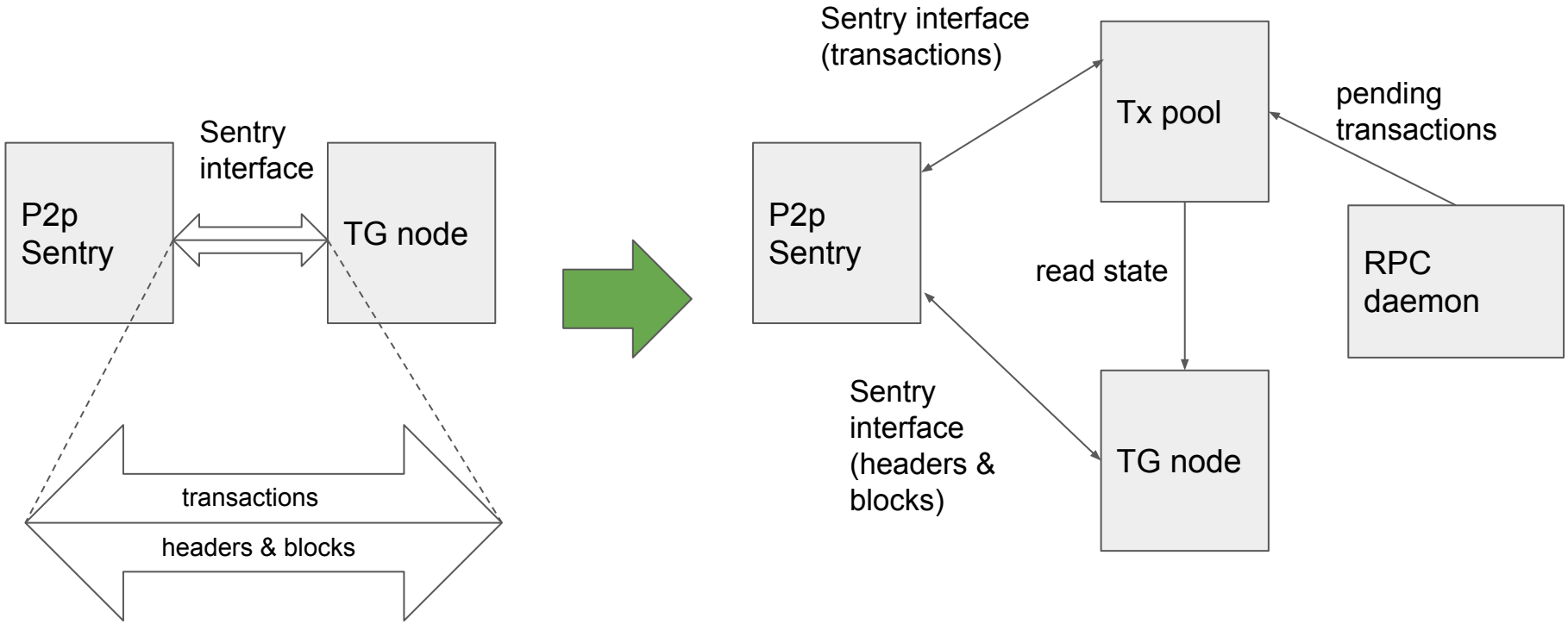
Staged sync - reasons

More efficient development. Stages are easier to profile and optimise, because they perform more homogeneous work. Also, they provide a natural way to split work among developers and let them specialise.

Performance. Due to database design, it is much faster to insert data into the database in bulk in pre-sorted order. Faster sync creates positive feedback loop on speed of testing and further development.

Permissionless development/Diversity. Individual stages can be replaced with alternative implementations, new staged can be added, without very little boilerplate and coordination with TG developers.

Transaction pool split (not started)



Transaction pool split - reasons

More efficient development. Transaction pool code can be worked on by someone to make the best possible implementation.

Performance. Transaction pool component may run on a computer with large memory with multiple CPUs to be able to process transactions quicker and form a priority queue to be consumed by a mining node.

Experimentation. There might be more options for different persistence of transactions, for later analysis.

Diversity. Because all interfaces are language-agnostic, it would be possible to have transaction pool components written in different languages