The core of this idea comes from the first 2 paragraphs ofAlexey Akhunov's ReGenesis proposal:

Lessons from Cosmos Hub

If you observed how Cosmos Hub performed their upgrade from version 1 to version 2, and then from version 2 to version 3, you would know that it was essentially done via re-starting the blockchain from a fresh genesis. Upon the upgrade, node operators had to shut down their nodes, then generate a snapshot of the state of Cosmos Hub, and then effectively use that snapshot as a genesis for launching a new blockchain, from block 1.

Now anyone who wants to join Cosmos Hub, needs to acquire genesis for CosmosHub-3, download all the blocks of CosmosHub-3 (but not CosmosHub-1 or CosmosHub-2) and replay them.

Can we "relaunch" Ethereum 1?

Lets look at a hypothetical application of such method in Ethereum, where we have a really large blockchain (150-160Gb), as well as fairly large state (40-100Gb depending on how you store it). The obvious gain of such "relaunch" would be that the new joiner nodes will need to start with the 40Gb genesis state and not with 150 Gb worth of blocks. But downloading 40 Gb genesis is still not a great experience.

(ReGenesis - resetting Ethereum to reduce the burden of large blockchain and state)

The rest of Alexey's proposal is good but still has research problems that suggest it could not be realistically deployed to mainnet (in my opinion) in under 2+ years. This also applies to the related proposal that Vitalik posted. (Alternative bounded-state-friendly address scheme)

While these proposals may be the long term solution to state grown on Ethereum I don't believe that they could be deployed quickly™. This proposal is attempting to define a quicker shorter term solution that will make further steps easier to deploy later on.

I propose that ETH1 clients repackage ETH1 state on some time period (per Alexey's suggestion maybe every 6 months, although I would hope it would be closer to 3*). We keep all of the current client state (but not transitions) and package it into a new regenesis block.

Existing ETH1 clients would build this "Regenesis" block incrementally and in a deterministic and binary identical way between clients. Since we want to do this Regenesis process on a regular basis we want to reduce the downtime of the network. If the regenesis block is incrementally built that should allow the network to have no downtime at all. I admit I'm fuzzy on many of the specific issues that would need to be addressed to make this happen, but perhaps there's a way to design the regenesis block to continue incrementing the block counter forward and not reset it to 0 at every Regenesis? I'm not sure if this makes the change easier or harder to implement.

The first regenesis would be the most intensive since it would contain years of information. But again it would be created incrementally by each ETH1 client.

Negatives:

1. Since ETH1 clients would no longer have state transitions from pre-regenesis, ETH1 clients would have to know that they cannot directly lookup any state transition information that is pre-regenesis, without some additional tooling.
2. The regenesis block data format would need to be rigorously defined and implemented by all ETH1 clients. This would probably be something new unless someone can propose an existing option to use. Preferably something already in use by ETH1 clients but which would still allow for fast ingest and quick distribution in chunks similar to P2P protocols like Bittorrent.

Positives:

1. This approach could cut down on the disk storage needed by ETH1 clients and allow more ETH1 clients to be the regenesis version of "archive" nodes.
2. It reduces the total network data transfer needed to synch a new node.
3. Makes it easier for smaller organizations or individuals to run nodes.
4. Once this pattern of regenesis is happening regularly on mainnet it would hopefully be easier for further refinements via (e.g. Alexey's or Vitalik's) proposals as those develop further in the future.
5. Hopefully the "complexity of Ethereum implementations will go down." (per Alexey's proposal).
6. This requires no comparatively larger changes like adding state rent fees or using witnesses.
7. Regenesis blocks could be an ideal place to perform network upgrades (HFs) in the future.

How feasible is this?

Are there changes that could be made to this proposal to reduce the amount of new code needed by ETH1 client teams to implement this?

I am perhaps foolishly optimistic that this would be possible to implement in testnets this year (2021) and then deployed next year.

(*I suggest a 3 month regenesis period because I believe we would not need to store historical state transition information for longer than we would need to protect against a hard fork. But there may be other considerations that would point to a longer time frame.)