BONSAI archive + state proofs

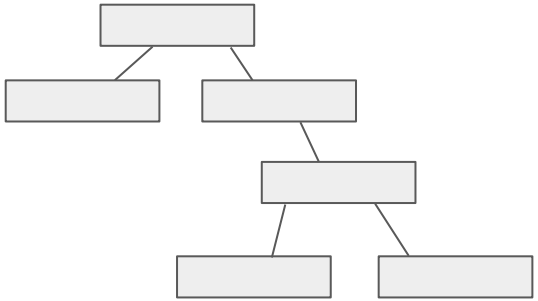
www.kaleido.io

kaleido

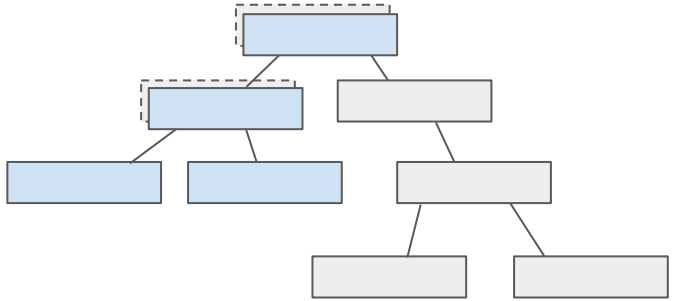
Bonsai archive state proofs

Recap of Ethereum state trie

Block 1



Block 2



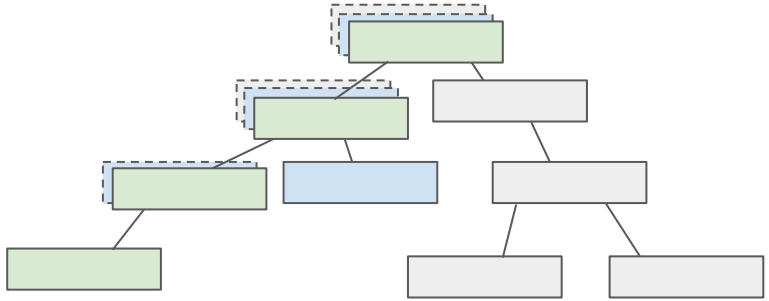
As blocks are mined, the state trie is modified

kaleido

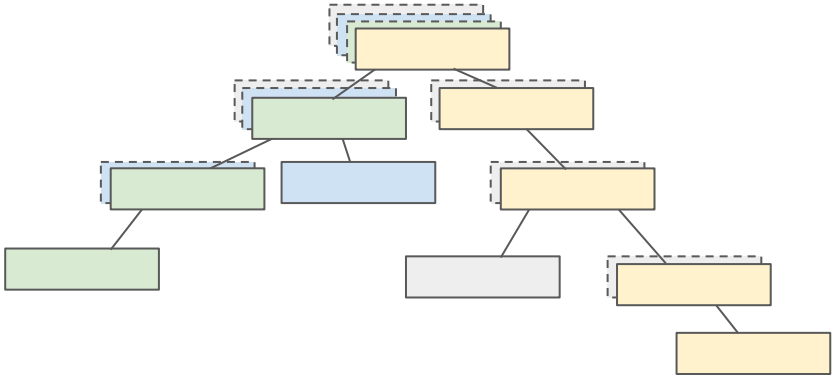
Bonsai archive state proofs

Recap of Ethereum state trie

Block 3



Block 4

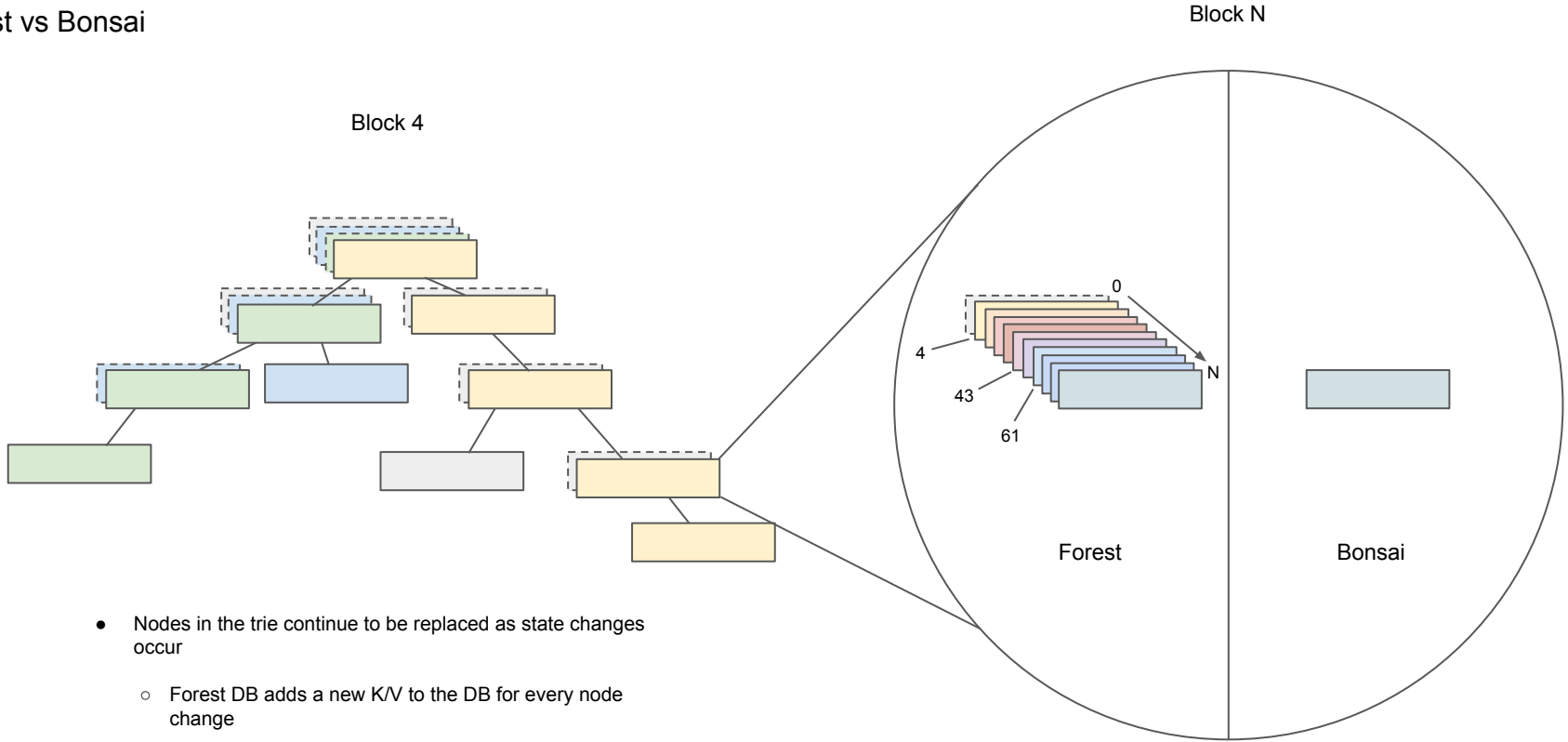


A block may modify just a subset of the state trie

kaleido

Bonsai archive state proofs

Forest vs Bonsai

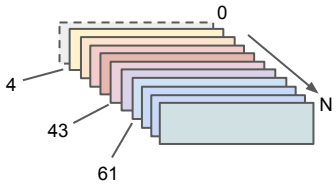


- Nodes in the trie continue to be replaced as state changes occur
  - Forest DB adds a new K/V to the DB for every node change
  - Bonsai DB replaces the old K/V pair, keeping only the latest version of the node

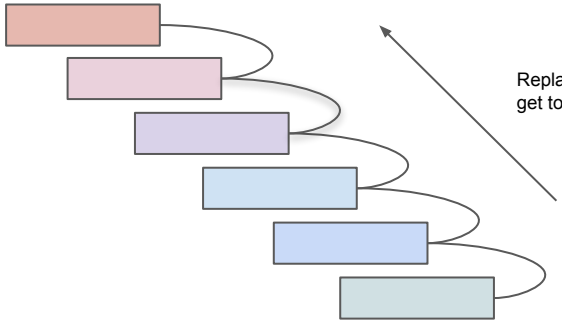
kaleido

Bonsai archive state proofs

How does Bonsai recreate state for e.g. 10 blocks ago?

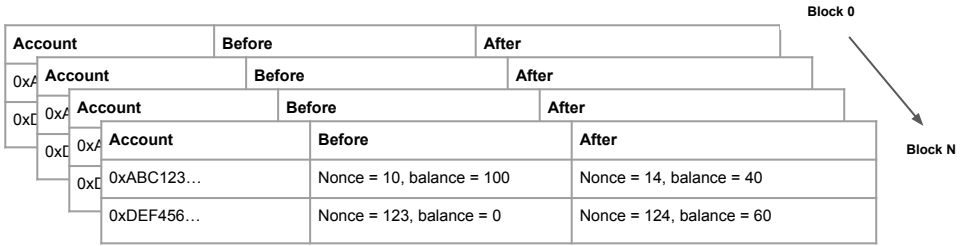


Forest



Bonsai

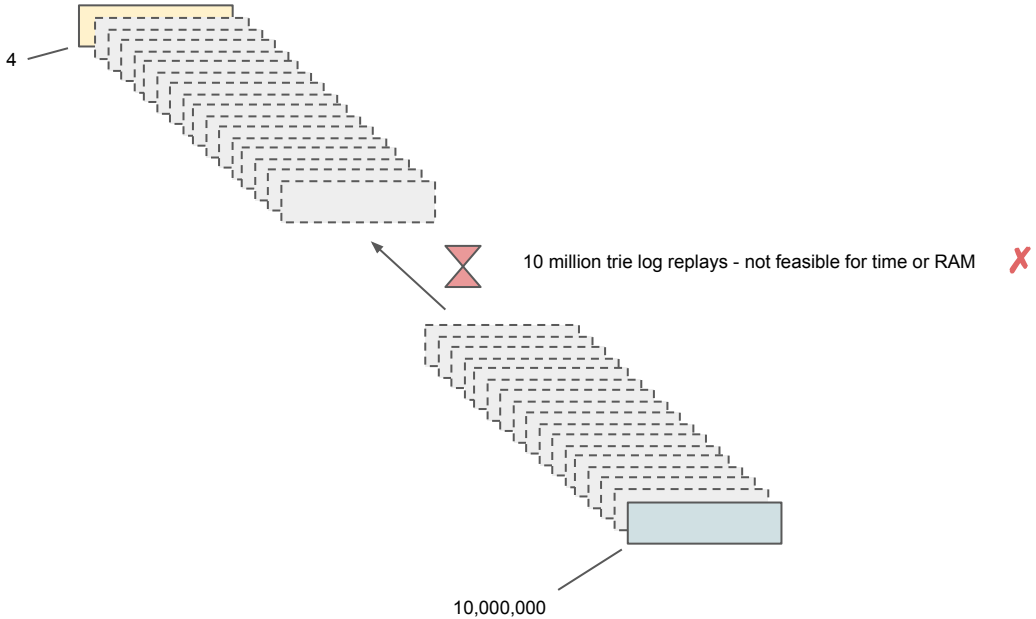
- A separate DB table records “trie logs”. Every state change is recorded as a before & after diff
- By taking the current state and applying the reverse of every trie log, we can build every trie node back to a specific block
- This is fine for 10s or 100s of blocks



kaleido

Bonsai archive state proofs

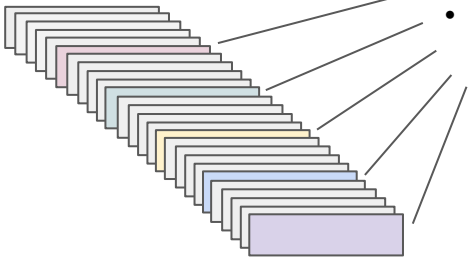
How does Bonsai recreate state for **10,000,000** blocks ago!? It can't 😕



kaleido

Bonsai archive state proofs

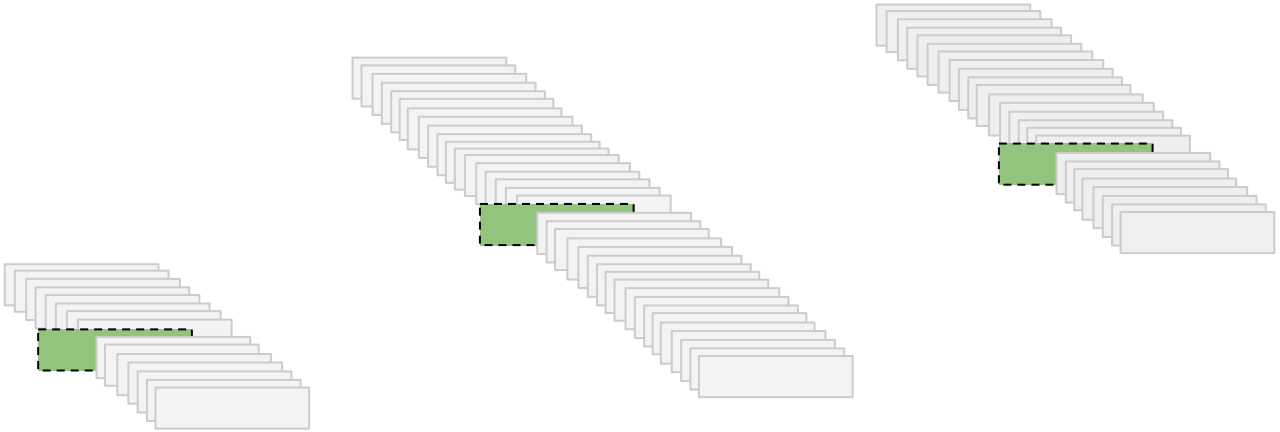
Bonsai state proof design



- Every N blocks, keep the latest trie node state up to that point
- Grey nodes are discarded and will need re-calculating when required

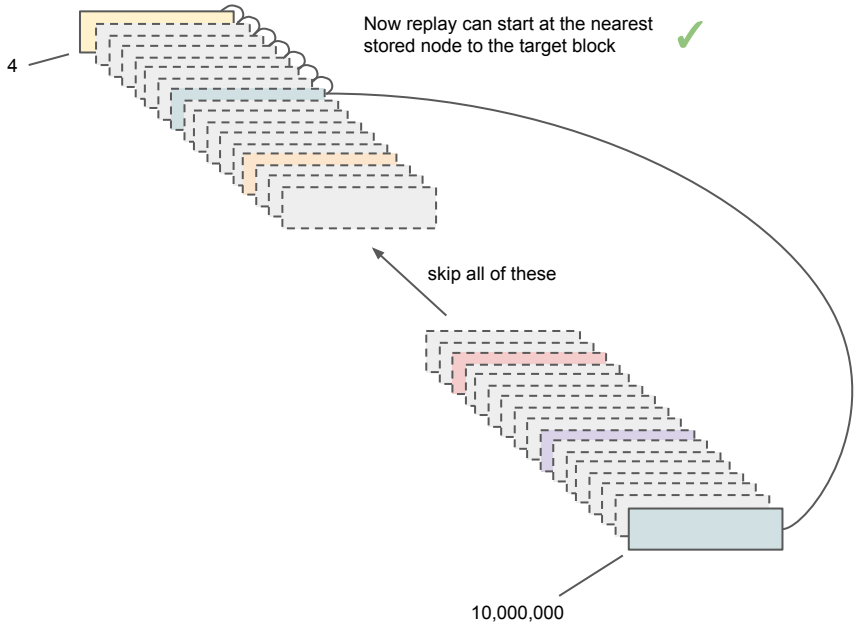
Bonsai archive state proofs

Bonsai state proof design



Bonsai archive state proofs

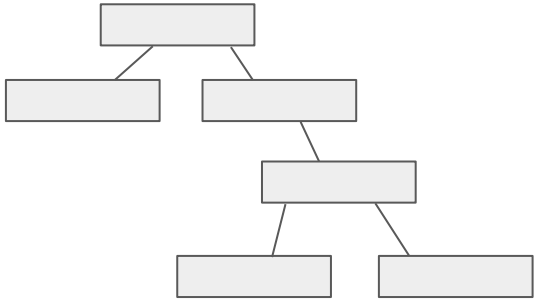
Bonsai state proof design



kaleido

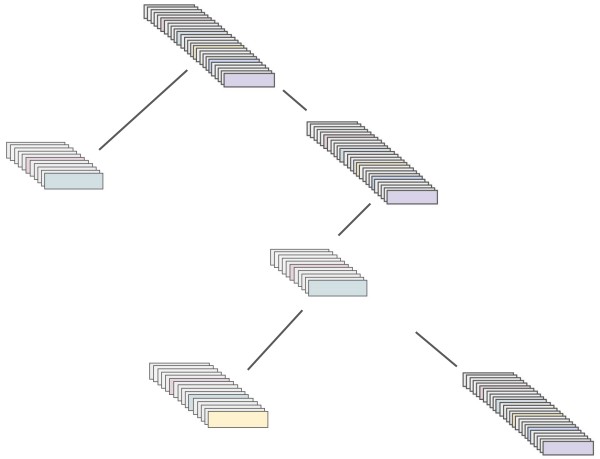
Bonsai archive state proofs

Bonsai state proof design



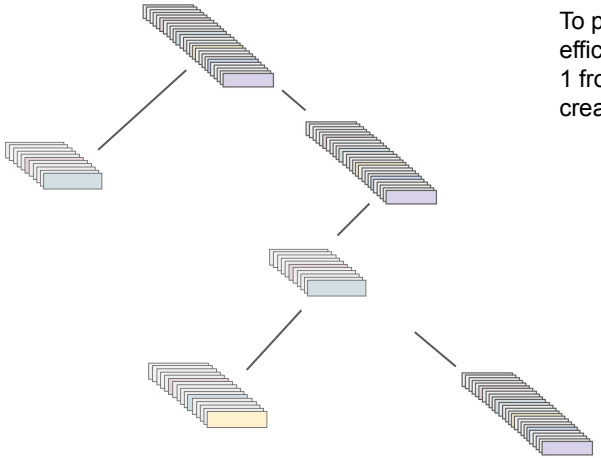
The state trie
looks like this…

…but the DB
looks like this…

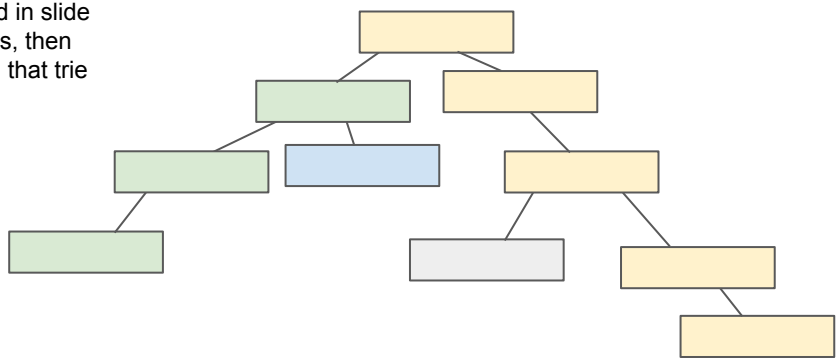


Bonsai archive state proofs

Bonsai state proof design

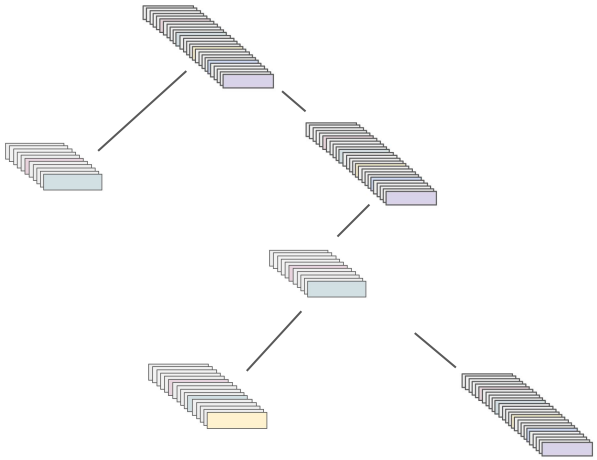


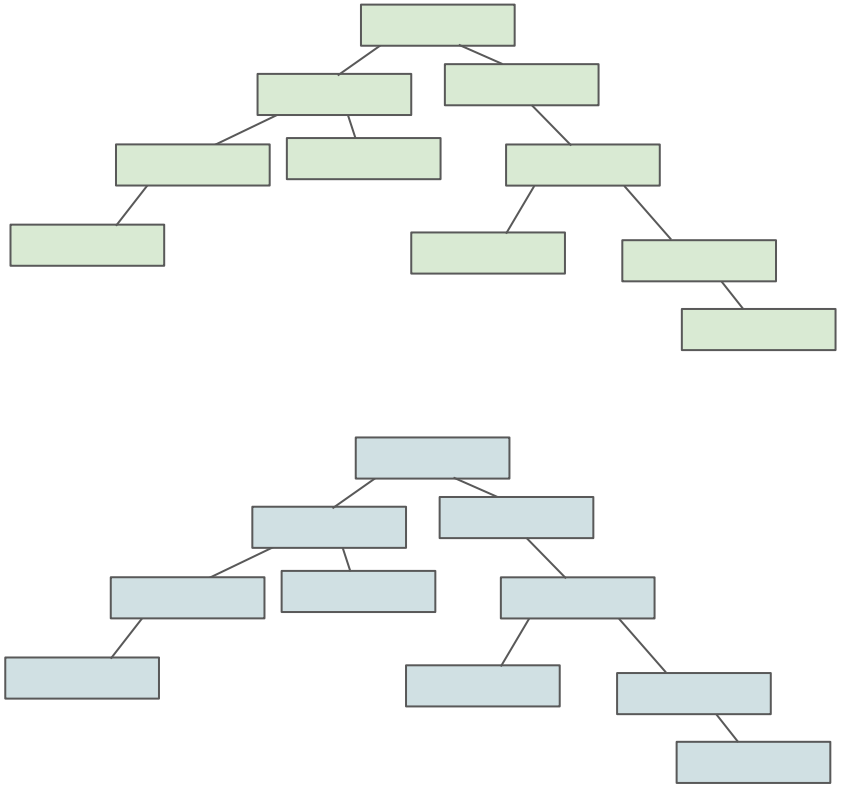
To produce a proof for block 4 we can efficiently build the state we had in slide 1 from our checkpoint trie nodes, then create the state proof based on that trie



Bonsai archive state proofs

Bonsai state proof design

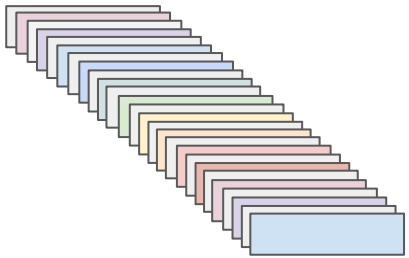




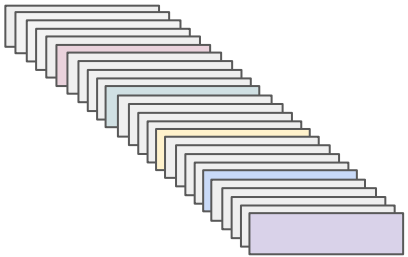
Bonsai archive state proofs

Bonsai state proof design

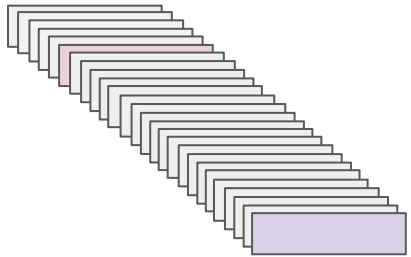
We can configure how often to take “checkpoint” nodes



Every 10 blocks? High disk usage.



Every 10,000 blocks? Slow query time.



Every 1,000,000 blocks? Query might never complete.

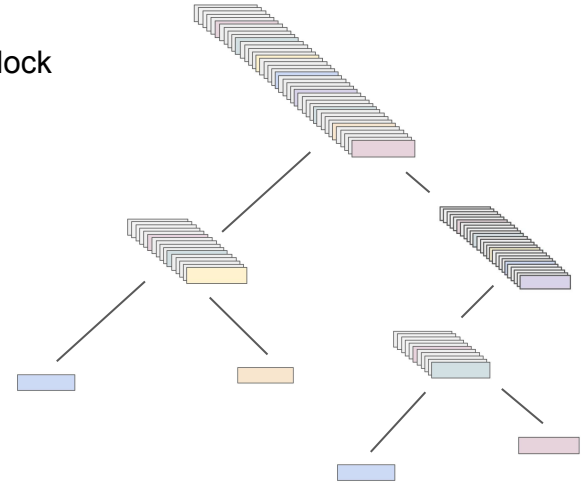
Testing and experimentation will hopefully lead to optimum configuration that gives the best tradeoff between disk use and query time.

kaleido

Bonsai archive state proofs

Is it just taking a snapshot of the entire trie every N blocks? **No**

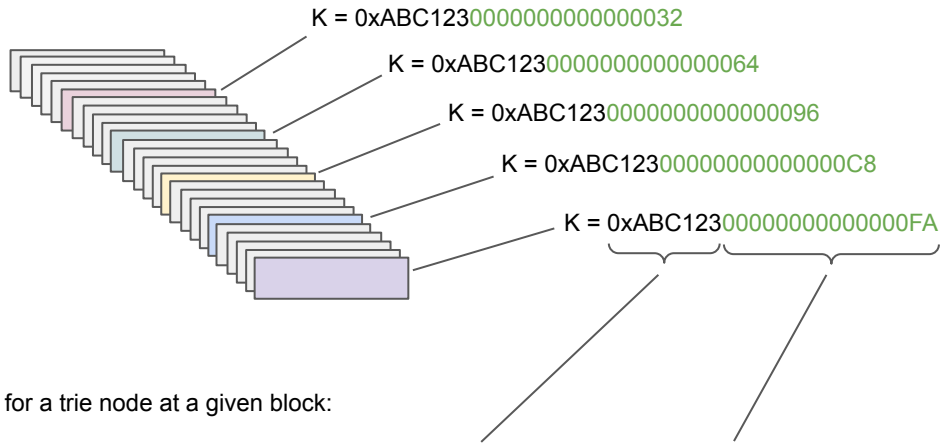
- Checkpoint nodes are created when **writing an individual node** to the DB
  - If a trie node is written once and never touched again, there will be a single entry in the DB for that node
  - If a trie node is written once at block 0, then once more at block 50,000,000, there will be 2 DB entries for that node
  - The DB is likely to look something like this…



Bonsai archive state proofs - flat DB implementation

The design uses a similar “bonsai context” design to that used for the initial Bonsai Archive work.

- There is already a Flat DB provider for Bonsai Archive that adds block-specific suffixes to Rocks DB K/V keys, both for PUTting and GETting from the DB
- I’ve used a similar approach for storing trie nodes, with a few differences
  - Because we’re going to use trie logs to roll back trie nodes, a new K/V pair isn’t written every time there is a change. It is written every N blocks.
  - Unlike accounts, trie keys vary in length e.g. 0x00, 0xFF22, 0x00AB2334. The DB query logic is therefore stricter than account lookup logic and only looks for keys of the expected length (i.e. original key + suffix length)
  - E.g. for state trie node 0xABC123 with checkpoint blocks every 50 blocks…

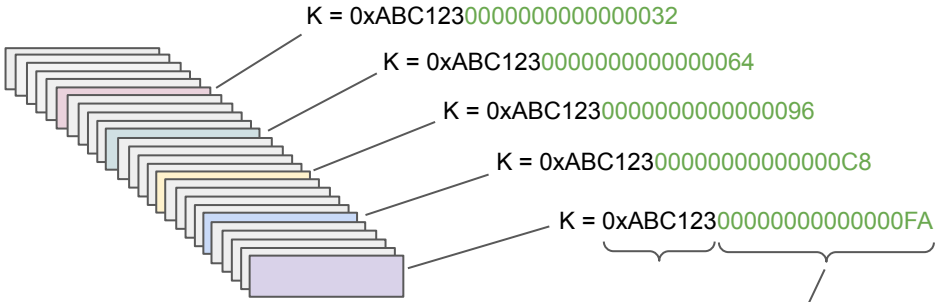


To read an entry for a trie node at a given block:

- `storage.get(TRIE_BRANCH_STORAGE, concat(TRIE_PATH, WORLD_BLOCK_NUMBER_KEY));`

kaleido

Bonsai archive state proofs - flat DB implementation



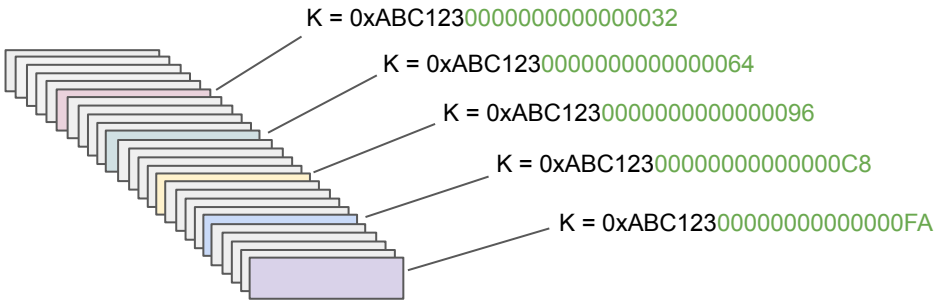
Reading the trie node for a block that is between checkpoint blocks requires a search:

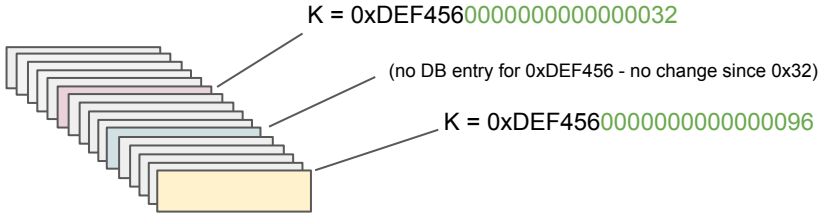
- `storage.get(TRIE_BRANCH_STORAGE, concat(0xABC123, 0x00000000000000FD));`

Since 0x00000000000000FD doesn't exist in the DB (it hasn't need a write since block 0xFA) we search for the nearest entry **before** the key we need:

- `storage.get(TRIE_BRANCH_STORAGE, concat(0xABC123, 0x00000000000000FD)).key() == 0xABC1230000000000000000FA;`

Bonsai archive state proofs - flat DB implementation





Checkpoint nodes are only written on PUT to the DB

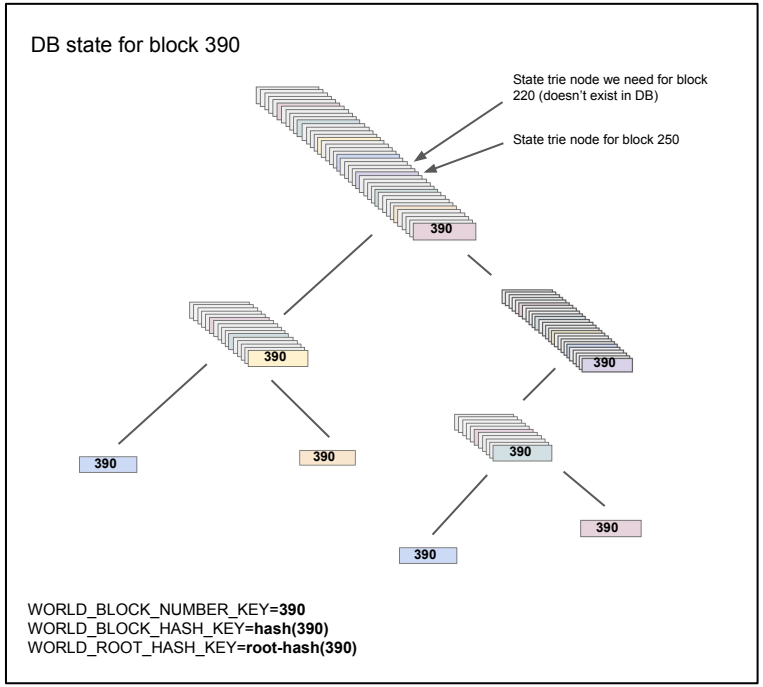
- If there are no writes to a state trie node, no DB entries exist for intermediate checkpoint blocks

kaleido

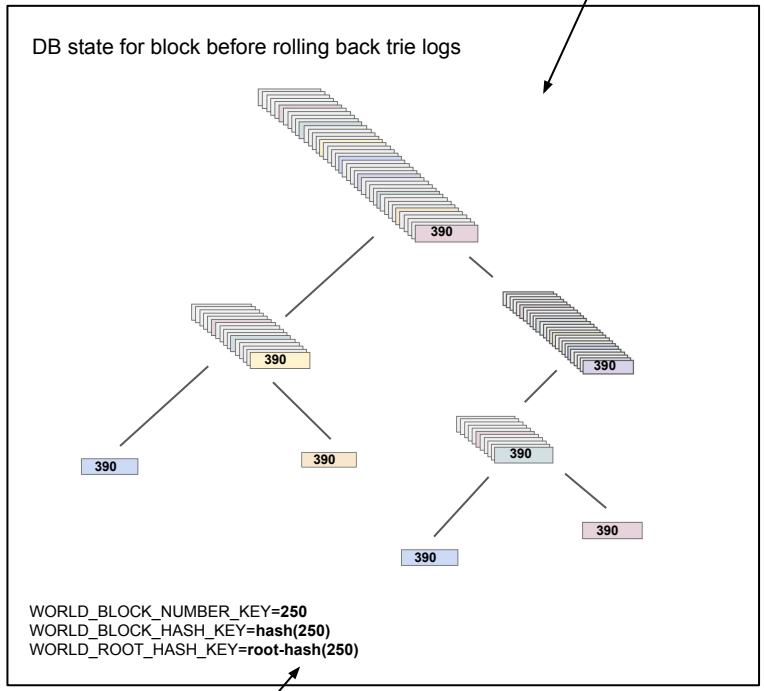
Bonsai archive state proofs - rolling world state to a specific block

Example with some code specifics:

- Chain head: 390
- State proof requested for block: 220
- Checkpoint interval: 50



Step 1



Assert that we are currently at the nearest checkpoint block to the target block

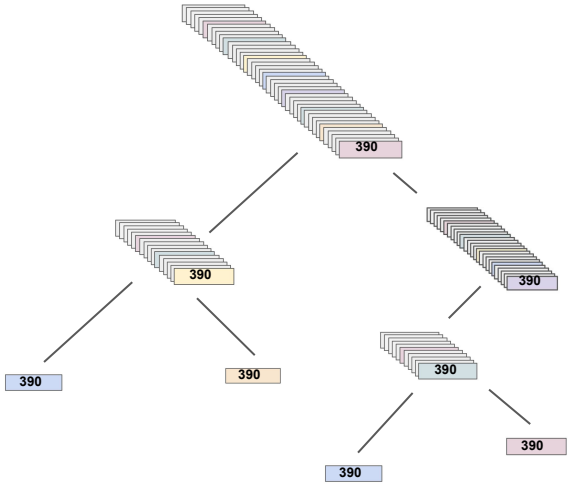
kaleido

Bonsai archive state proofs - rolling world state to a specific block

Example with some code specifics:

- Chain head: 390
- State proof requested for block: 220
- Checkpoint interval: 50

DB state for block before rolling back trie logs



WORLD_BLOCK_NUMBER_KEY=**250**
WORLD_BLOCK_HASH_KEY=**hash(250)**
WORLD_ROOT_HASH_KEY=**root-hash(250)**

Step 2

Construct list of trie-logs from block 250 -> 220:

```
while (persistedHeader.getNumber() > targetHeader.getNumber()) {
  LOG.debug("Rollback {}", persistedBlockHash);
  rollBacks.add(trieLogManager.getTrieLogLayer(persistedBlockHash).get());
  persistedHeader = blockchain.getBlockHeaderSafe(persistedHeader.getParentHash()).get();
  persistedBlockHash = persistedHeader.getBlockHash();
}
```

Roll back every trie log on the accumulator:

```
for (final TrieLog rollBack : rollBacks) {
  LOG.info("Attempting Rollback of {}", rollBack.getBlockHash());
  diffBasedUpdater.rollBack(rollBack);
}
diffBasedUpdater.commit();
```

WORLD_BLOCK_NUMBER_KEY=**250**
WORLD_BLOCK_HASH_KEY=**hash(250)**
WORLD_ROOT_HASH_KEY=**root-hash(250)**

World state DB still appears to be at the nearest checkpoint block

kaleido

Bonsai archive state proofs - rolling world state to a specific block

Example with some code specifics:

- Chain head: 390
- State proof requested for block: 220
- Checkpoint interval: 50

Construct list of trie-logs from block 250 -> 220:

```
while (persistedHeader.getNumber() > targetHeader.getNumber()) {
  LOG.debug("Rollback {}", persistedBlockHash);
  rollBacks.add(trieLogManager.getTrieLogLayer(persistedBlockHash).get());
  persistedHeader = blockchain.getBlockHeaderSafe(persistedHeader.getParentHash()).get();
  persistedBlockHash = persistedHeader.getBlockHash();
}
```

Roll back every trie log on the accumulator:

```
for (final TrieLog rollBack : rollBacks) {
  LOG.info("Attempting Rollback of {}", rollBack.getBlockHash());
  diffBasedUpdater.rollBack(rollBack);
}
diffBasedUpdater.commit();
```

WORLD_BLOCK_NUMBER_KEY=**250**
WORLD_BLOCK_HASH_KEY=**hash(250)**
WORLD_ROOT_HASH_KEY=**root-hash(250)**

Step 3

Persist the mutable world state. This involves validating the root hash which requires us to continue **reading** from world state as if we are at block 250 (because we haven't applied the trie log changes yet)

Therefore WORLD_BLOCK_NUMBER_KEY must still be asserted to be 250.

However, we need to **write** new entries to the state trie as if we were at block 220. So we introduce a new DB key:

ARCHIVE_PROOF_BLOCK_NUMBER_KEY=**220**

1. All reads during the mutable world state persist are done at block 250
2. All writes during the mutable world state persist are done at block 220

```
SegmentedKeyValueStorageTransaction tx =
    mutableState.getWorldStateStorage().getComposedWorldStateStorage().startTransaction();
tx.put(
    TRIE_BRANCH_STORAGE,
    ARCHIVE_PROOF_BLOCK_NUMBER_KEY,
    Bytes.ofUnsignedLong(blockchain.getBlockHeaderSafe(targetBlockHash).get().getNumber())
        .toArrayUnsafe());
tx.commit();

mutableState.persist(blockchain.getBlockHeaderSafe(targetBlockHash).get());
```

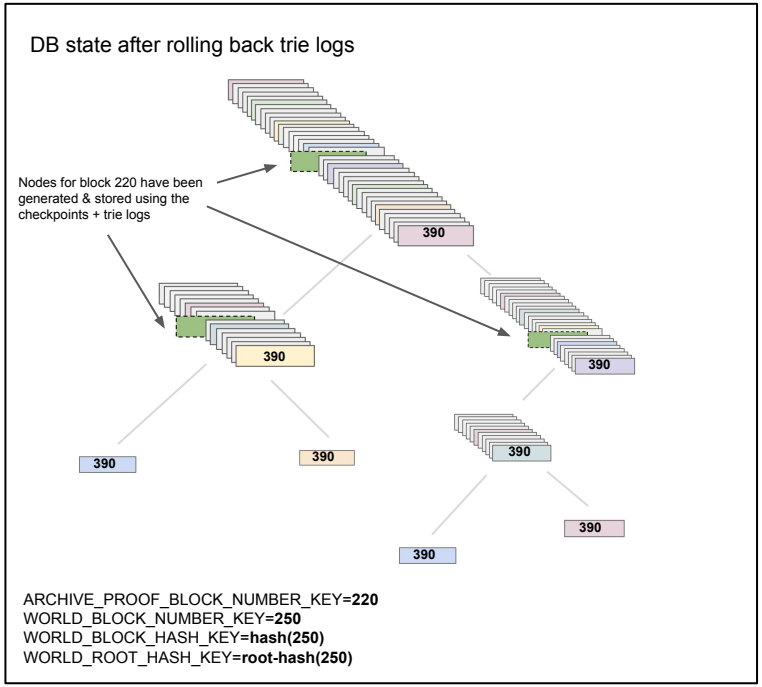
ARCHIVE_PROOF_BLOCK_NUMBER_KEY=**220**
WORLD_BLOCK_NUMBER_KEY=**250**
WORLD_BLOCK_HASH_KEY=**hash(250)**
WORLD_ROOT_HASH_KEY=**root-hash(250)**

kaleido

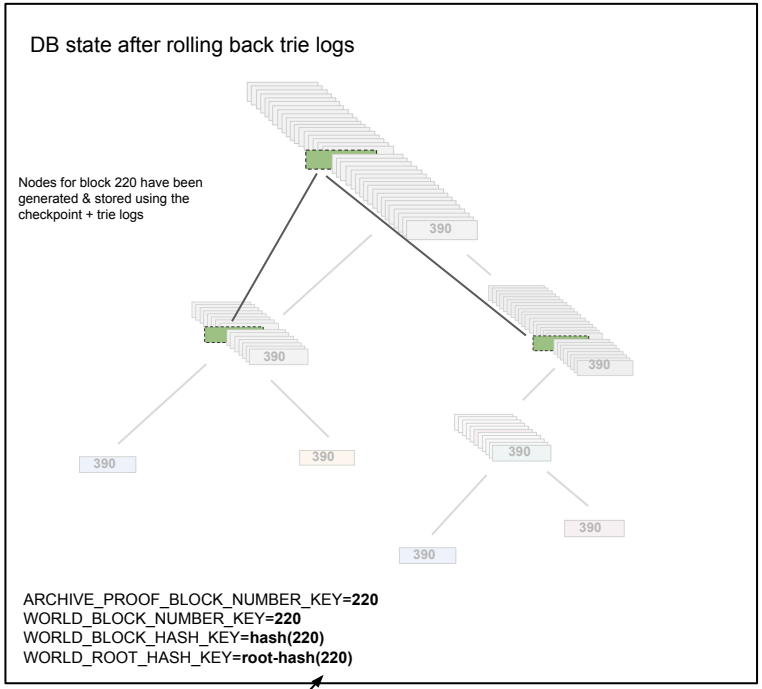
Bonsai archive state proofs - rolling world state to a specific block

Example with some code specifics:

- Chain head: 390
- State proof requested for block: 220
- Checkpoint interval: 50



Step 4



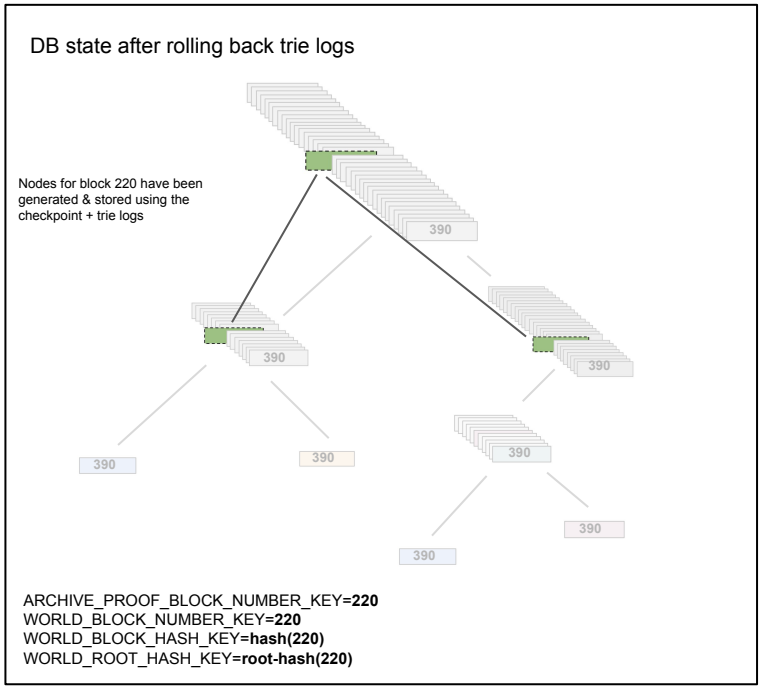
World state DB is now asserted to be at block 220 and we can use it to make any state queries

kaleido

Bonsai archive state proofs - rolling world state to a specific block

Example with some code specifics:

- Chain head: 390
- State proof requested for block: 220
- Checkpoint interval: 50



From earlier slide:

To read an entry for a trie node at a given block:

- `storage.get(TRIE_BRANCH_STORAGE, concat(TRIE_PATH, WORLD_BLOCK_NUMBER_KEY));`

So all queries against the rolled archive world state now return nodes from the correct block in history i.e. block 220

kaleido

Bonsai archive state proofs

Orange = PMT node path/location
Blue = archive block suffix
Black = PMT node value

Checkpoint blocks are updated with new values until a new checkpoint is passed

- For example, at block 100 entries are written to the DB with suffix 0x0000000000000064
- If a node is updated at block 105, the entry 0x0000000000000064 is updated with a new value and so on.
- When we reach block 199 the last update to 0x0000000000000064 is written (if there is one), and then new updates for block 200+ are made with suffix 0x00000000000000C8
- 0x00000000000000C8 will be updated with new values until block 300, and so on

Here are some example entries from a real DB at block 205 in a new chain. Checkpoints are every 100 blocks.

```
0x050000000000000000 ==> 0xF8518080808080A0253E6…
0x050000000000000064 ==> 0xF8518080808080A0253E6…
0x0505000000000000000 ==> 0xF869A0208AA36520B83F9…
0x050C000000000000000 ==> 0xF86EA020D9D8A442C611…
0x050C000000000000064 ==> 0xF86EA020D9D8A442C611…
0x050C00000000000000C8 ==> 0xF86EA020D9D8A442C611…
0x060000000000000000 ==> 0xF883A03C276B7180DCF065…
0x060000000000000064 ==> 0xF883A03C276B7180DCF065…
0x0C000000000000000064 ==> 0xF869A0365617775BABD6B8…
0x0D4455509DB54AD570526524416A469F52C01D371599BC31062539E8871373BD0000000000000000 ==> 0xE6A120290DECD9548B62A8D60…
```

kaleido

Bonsai archive state proofs

Orange = PMT node path/location
Blue = archive block suffix
Black = PMT node value

```
0x0500000000000000000 ==> 0xF8518080808080A0253E6…
0x0500000000000000064 ==> 0xF8518080808080A0253E6…
0x0505000000000000000000 ==> 0xF869A0208AA36520B83F9…
0x050C000000000000000000 ==> 0xF86EA020D9D8A442C611…
0x050C000000000000000064 ==> 0xF86EA020D9D8A442C611…
0x050C0000000000000000C8 ==> 0xF86EA020D9D8A442C611…
0x060000000000000000000 ==> 0xF883A03C276B7180DCF065…
0x060000000000000000064 ==> 0xF883A03C276B7180DCF065…
0x0C000000000000000064 ==> 0xF869A0365617775BABD6B8…
0x0D4455509DB54AD570526524416A469F52C01D371599BC31062539E8871373BD0000000000000000 ==> 0xE6A120290DECD9548B62A8D60…
```

This entry is the checkpoint block for 0x05, block 100. It will remain unchanged from now on because we have passed block 200.

Several updates to 0x05 could have been made during blocks 100-199. We can't tell from this entry how many.

kaleido

Bonsai archive state proofs

Orange = PMT node path/location
Blue = archive block suffix
Black = PMT node value

```
0x050000000000000000 ==> 0xF8518080808080A0253E6…
0x050000000000000064 ==> 0xF8518080808080A0253E6…
0x0505000000000000000 ==> 0xF869A0208AA36520B83F9…
0x050C000000000000000 ==> 0xF86EA020D9D8A442C611…
0x050C000000000000064 ==> 0xF86EA020D9D8A442C611…
0x050C0000000000000C8 ==> 0xF86EA020D9D8A442C611…
0x060000000000000000 ==> 0xF883A03C276B7180DCF065…
0x060000000000000064 ==> 0xF883A03C276B7180DCF065…
0x0C000000000000000064 ==> 0xF869A0365617775BABD6B8…
0x0D4455509DB54AD570526524416A469F52C01D371599BC31062539E8871373BD0000000000000000 ==> 0xE6A120290DECD9548B62A8D60…
```

This entry is the checkpoint block for 0x0505

We know it hasn't been updated since block 100 because there is no 0x0505000000000000000064 or 0x05050000000000000C8 entry

kaleido

Bonsai archive state proofs

Orange = PMT node path/location
Blue = archive block suffix
Black = PMT node value

```
0x050000000000000000 ==> 0xF8518080808080A0253E6…
0x050000000000000064 ==> 0xF8518080808080A0253E6…
0x0505000000000000000000 ==> 0xF869A0208AA36520B83F9…
0x050C000000000000000000 ==> 0xF86EA020D9D8A442C611…
0x050C000000000000000064 ==> 0xF86EA020D9D8A442C611…
0x050C0000000000000000C8 ==> 0xF86EA020D9D8A442C611…
0x060000000000000000 ==> 0xF883A03C276B7180DCF065…
0x060000000000000064 ==> 0xF883A03C276B7180DCF065…
0x0C000000000000000064 ==> 0xF869A0365617775BABD6B8…
0x0D4455509DB54AD570526524416A469F52C01D371599BC31062539E8871373BD0000000000000000 ==> 0xE6A120290DECD9548B62A8D60…
```

This entry is the checkpoint block for leaf node 0x0D4455509DB54AD570526524416A469F52C01D371599BC31062539E8871373BD

We know it hasn't been updated in blocks 100-199 because there is no **0x**0D4455509DB54AD570526524416A469F52C01D371599BC31062539E8871373BD0000000000000064 block

kaleido

Bonsai archive state proofs

Orange = PMT node path/location
Blue = archive block suffix
Black = PMT node value

0x050000000000000000 ==> 0xF8518080808080A0253E6…
0x050000000000000064 ==> 0xF8518080808080A0253E6…
0x0505000000000000000000 ==> 0xF869A0208AA36520B83F9…
0x050C000000000000000000 ==> 0xF86EA020D9D8A442C611…
0x050C000000000000000064 ==> 0xF86EA020D9D8A442C611…
0x050C0000000000000000C8 ==> 0xF86EA020D9D8A442C611…
0x060000000000000000 ==> 0xF883A03C276B7180DCF065…
0x060000000000000064 ==> 0xF883A03C276B7180DCF065…
0x0C000000000000000064 ==> 0xF869A0365617775BABD6B8…
0x0D4455509DB54AD570526524416A469F52C01D371599BC31062539E8871373BD0000000000000000 ==> 0xE6A120290DECD9548B62A8D60…

If an update is needed for trie node 0x050C at block 205 this key will be updated with the new value

If another update is needed at block 210 the same value entry will be updated with a new value

kaleido

Bonsai archive state proofs

Orange = PMT node path/location
Blue = archive block suffix
Black = PMT node value

Some real DB entries (checkpoint every 100 blocks, these entries captured at block 205)

```
0x050000000000000000 ==> 0xF8518080808080A0253E6…
0x050000000000000064 ==> 0xF8518080808080A0253E6…
0x0505000000000000000 ==> 0xF869A0208AA36520B83F9…
0x050C000000000000000 ==> 0xF86EA020D9D8A442C611…
0x050C000000000000064 ==> 0xF86EA020D9D8A442C611…
0x050C0000000000000C8 ==> 0xF86EA020D9D8A442C611…
0x060000000000000000 ==> 0xF883A03C276B7180DCF065…
0x060000000000000064 ==> 0xF883A03C276B7180DCF065…
0x0C000000000000064 ==> 0xF869A0365617775BABD6B8…
0x0D4455509DB54AD570526524416A469F52C01D371599BC31062539E8871373BD0000000000000000 ==> 0xE6A120290DECD9548B62A8D60…
```

Requesting a proof for an account at block 105 will result in RocksDB queries for 0x<path>0000000000000069

These will perform find-nearest queries to arrive at 0x<path>0000000000000000 or 0x<path>0000000000000064 depending on when the node was updated last

Trie-log rolling can then begin from there to arrive at what the actual block entry should be e.g. 0x<path>0000000000000037 for a node that was last updated at block 55.

kaleido

Bonsai archive state proofs demo

Bonsai

(port 8545)

Current state & proofs	Historic state		Historic proofs	
	<= 512 blocks old	> 512 blocks old	<= 512 blocks old	> 512 blocks old
✓	✓	✗	✓	✗

Bonsai archive

(port 8546)

Current state & proofs	Historic state		Historic proofs	
	<= 512 blocks old	> 512 blocks old	<= 512 blocks old	> 512 blocks old
✓	✓	✓	✓	✗

Bonsai archive + state proofs

(port 8547)

Current state & proofs	Historic state		Historic proofs	
	<= 512 blocks old	> 512 blocks old	<= 512 blocks old	> 512 blocks old
✓	✓	✓	✓	✓

Forest

(port 8548)

Current state & proofs	Historic state		Historic proofs	
	<= 512 blocks old	> 512 blocks old	<= 512 blocks old	> 512 blocks old
✓	✓	✓	✓	✓

kaleido

Bonsai archive state proofs demo

Bonsai

(port 8545)

Current state & proofs	Historic state		Historic proofs	
	<= 512 blocks old	> 512 blocks old	<= 512 blocks old	> 512 blocks old
✓	✓	✗	✓	✗

Bonsai archive

(port 8546)

Current state & proofs	Historic state		Historic proofs	
	<= 512 blocks old	> 512 blocks old	<= 512 blocks old	> 512 blocks old
✓	✓	✓	✓	✗

Bonsai archive + state proofs

(port 8547)

Current state & proofs	Historic state		Historic proofs	
	<= 512 blocks old	> 512 blocks old	<= 512 blocks old	> 512 blocks old
✓	✓	✓	✓	✓

Why go to all the effort to implement this…

Forest

(port 8548)

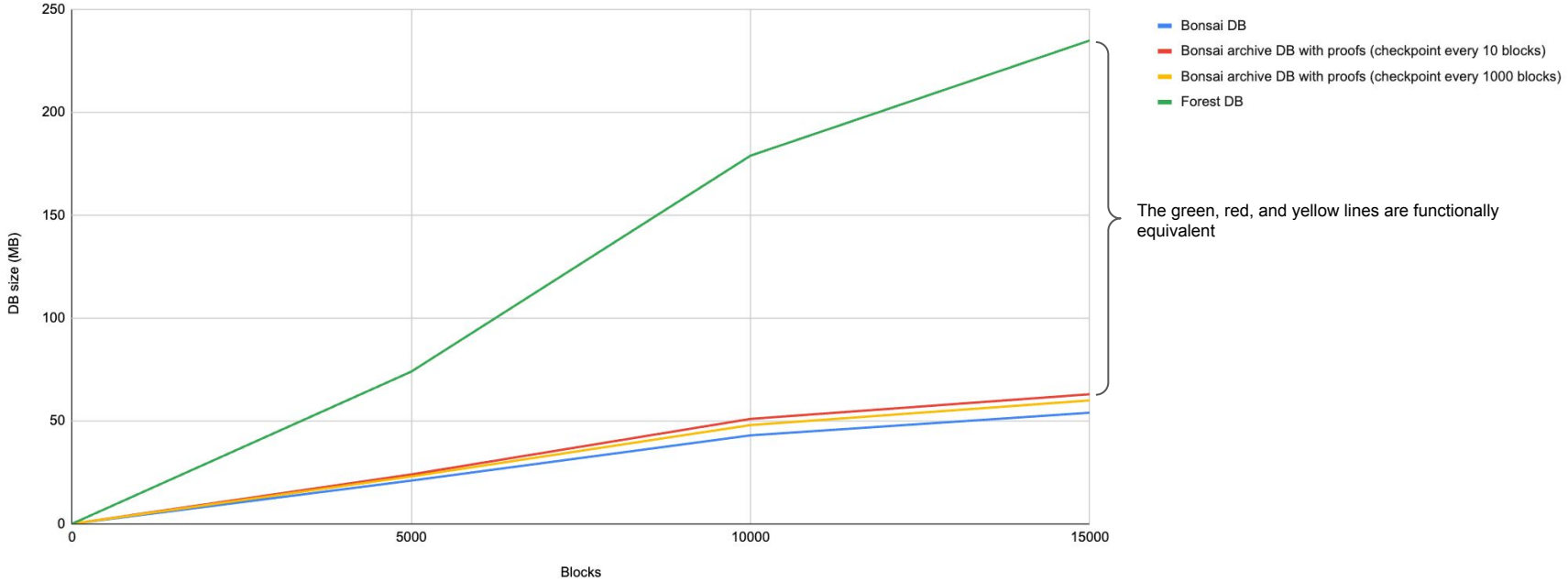
Current state & proofs	Historic state		Historic proofs	
	<= 512 blocks old	> 512 blocks old	<= 512 blocks old	> 512 blocks old
✓	✓	✓	✓	✓

…if Besu already had this?

kaleido

Bonsai archive state proofs - DB size comparison

- ~10 TPS simple storage contract with Bonsai archive checkpoints every 1000 blocks
- Bonsai proofs require up to 999 blocks of trie logs to be rolled back



kaleido