# Nearly-Optimal Merkle Trie (NOMT) Database

The NOMT Database is a merklized database optimized for SSDs. It supports efficient merkle multiproofs of reads and updates. Below I am listing down a summary of the key features in detail which help achieve this.

## Efficiency with the van Emde Boas Data Structure

Merkle Patricia Tries store each node as an individual database entry, whereas the NOMT database uses van Emde Boas data structure which groups data each node stores rootless tree

- Data Grouping
    - Nodes are grouped with a size of √n where n is the total number of items in the state
- Successor and Predecessor Queries in O(log(log n)) Complexity
    - When searching for the next key, vEB efficiently identifies the successor with minimal computational overhead.
    - This drastically improves the performance of state lookups and iteration over the trie compared to O(log n) complexity.
- Less Storage Overhead
- Binary Trie
    - Nomt's bitbox storage has binary trie structure
    - Node types
        - Branch
        - Leaf
        - Terminator

### Example: Storage Differences

For a state with the following keys and values:

```
| Key (Hex) | Value         |
|-----------|---------------|
| a711355   | 45.0 ETH      |
| a77d337   | 1.00 WEI      |
| a7f9365   | 1.1 ETH       |
| a77d397   | 0.12 ETH      |
```

**Merkle Patricia Trie**:

```
Key: a7
   -> Branch Node [0, 1, 2, 3, ..., f]
          -> Leaf Node 1 mapped from 1 in Branch Node with kv pair
(1355, 45.0 ETH)
          -> Leaf Node 2 mapped from f in Branch Node with kv pair
(9365, 1.1 ETH)
          -> Branch Node mapped from 7 in Branch Node
                 -> Leaf Node 1 mapped from 3 with kv pair (7, 1.00 WEI)
                 -> Leaf Node 2 mapped from 9 with kv pair (7, 0.12 ETH)
```

**NOMT**

```
Key: 10100111
Value: {
    branch: [
        0 -> Subtree (000): Terminator (Key: 00010001001101010101,
Value: 45.0 ETH),
        1 -> Subtree (01111101): {
            0 -> Terminator (Key: 01111101001100110111, Value: 1.00
WEI),
            1 -> Terminator (Key: 01111101001110010111, Value: 0.12 ETH)
        },
        1 -> Subtree (1111): Terminator (Key: 11111001001101100101,
Value: 1.1 ETH)
    ]
}
```

## Optimized for Modern SSDs

NOMT is specifically designed to take advantage of modern SSD architectures, where read and write operations are most efficient when done in **pages** (typically 4 KB per page).

- **Efficient I/O**:
    - Grouped nodes are written as contiguous pages, minimizing random access overhead and reducing I/O latency.
    - Instead of batching with size root n, NOMT batches according to SSD page size for efficiency

## Reduced Roundtrips and Random Access

By grouping nodes, NOMT minimizes the number of **random access operations** required to traverse or update the trie.

- **Fewer Database Lookups**
  - In traditional MPTs, deep trie traversal involves multiple disk I/O operations to fetch each intermediate node.
  - NOMT fetches grouped nodes in a single operation, significantly reducing roundtrips.
- **Improved Cache Efficiency**
  - Grouped nodes are more cache-friendly, as related nodes are fetched and stored together, reducing latency for repeated access.

## Bitbox and Beattree

NOMT has two databases

- **Bitbox**:
  - Hash table
  - Uses the **van Emde Boas** data structure for faster updates and proof generation.
- **Beattree**:
  - Key value store
  - Uses a **B-tree** structure for faster reads.
  - Serves state reads in 1 I/O

## Applications in Substrate-Based Blockchains

Substrate, the blockchain framework underlying Polkadot and other networks, heavily relies on trie-based storage for managing state.

- **Improved Runtime Performance**
  - Substrate's state updates and reads (e.g., for smart contracts, account balances) benefit from the reduced storage and access overhead in NOMT.
- **Designed Better for State Snapshots**
  - Data grouping and compact storage enable faster creation of state snapshots and efficient pruning of historical states, which are critical for maintaining blockchain performance as the state grows.
- **Lower Storage Costs**
  - By minimizing redundancy and improving write efficiency, NOMT reduces the storage footprint of Substrate-based nodes.

## Impact on Light Clients

Light clients do not store the entire blockchain state, they rely on efficient proof generation and verification for operations such as account queries and transaction validation by validating merkle proofs

- **Faster Verification**
  - With fewer nodes to traverse and verify, light clients can validate proofs more quickly, improving responsiveness.
- **Better Syncing Performance**
  - Light clients downloading only necessary trie fragments benefit from the grouped storage model, as fewer database lookups are required to reconstruct relevant state segments.

## Conclusion

The NOMT database addresses the key bottlenecks of traditional MPT implementations by using the van Emde Boas data structure, compact data grouping, and reduced roundtrips. For Substrate-based blockchains, NOMT yields improved performance, lower costs of storage, and effective pruning while providing light clients with the advantages of faster proof generation and validation.

## References

- NOMT Sub0 reset 2024
- https://www.rob.tech/blog/introducing-nomt/
- https://sovereign.mirror.xyz/jfx_cJ_15saejG9ZuQWjnGnG-NfahbazQH98i1J3NN8
- VAN Emde Boas Tree
- Quantum fusion NOMT
- Github