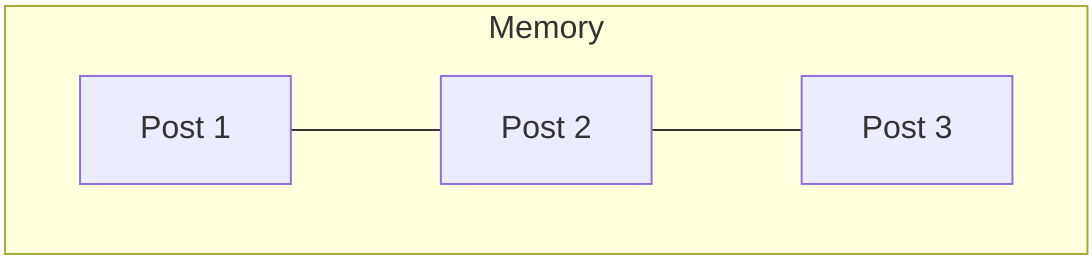
`avl.tree` vs `map` in Gno

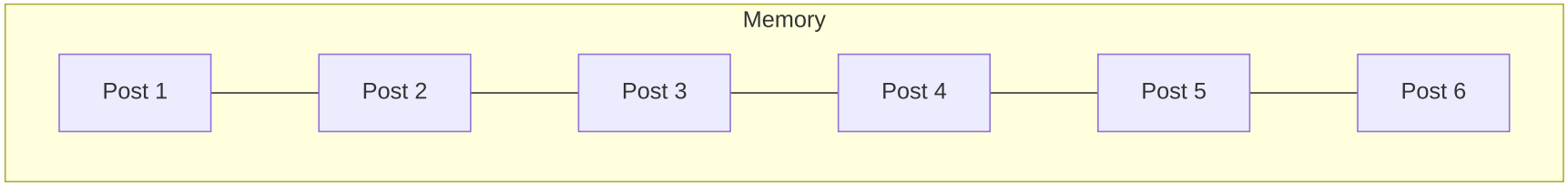
Why `map` is not suitable for efficient dynamic on-chain storage
How Gno's `avl.Tree` powers an efficient alternative

Let’s say we have a blog

📚 Use Case: Blog Posts with `map`

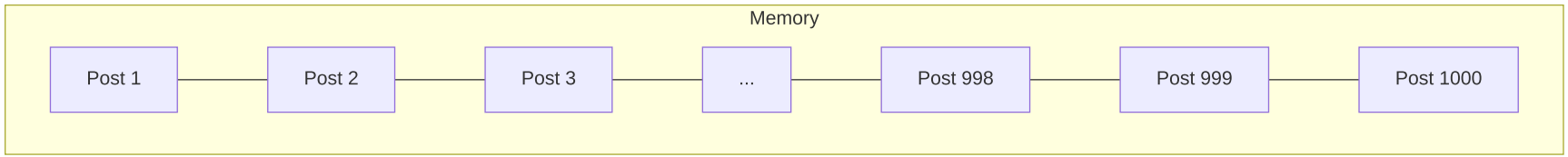


📚 Use Case: Blog Posts with `map`



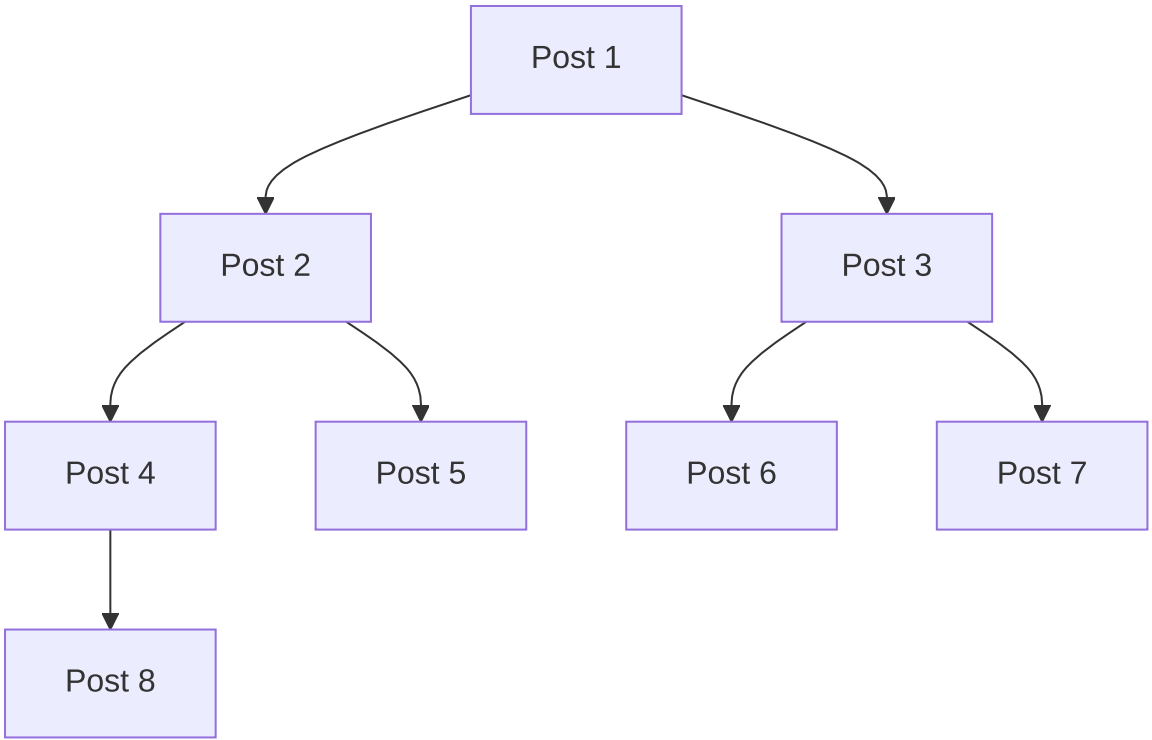
⚠️ More memory usage = More gas usage !

📚 Use Case: Blog Posts with `map`

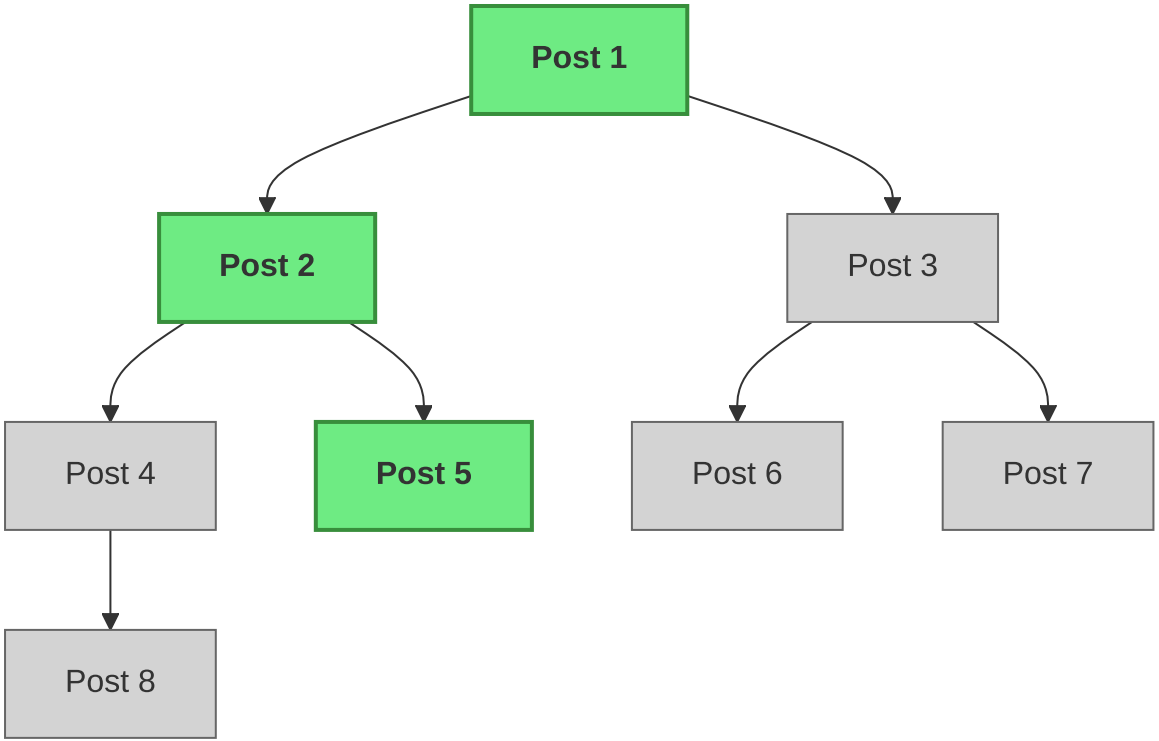


🤯 Unlimited gas usage

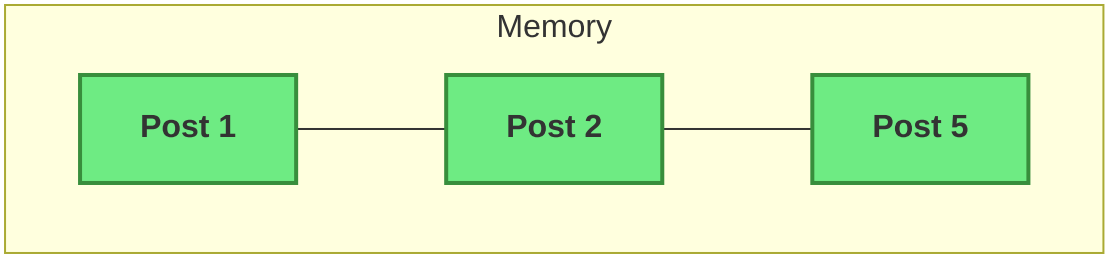
📚 Use Case: Blog Posts with `avl.tree`



📚 Use Case: Blog Posts with `avl.tree`



📚 Use Case: Blog Posts with `avl.tree`



🗺️ Maps

In-memory key/value structure

✅ Pros

- Good for small, fixed-size data
- Simple syntax

❌ Cons

- Not scalable: All data is kept in memory

```
var data = make(map[string]string)
data["key"] = "value"
```

🌲 AVL Trees

Self-balancing binary search tree

✅ Pros

- Efficient memory usage
- Suitable for large datasets

❌ Cons

- O(log n) access times

```
import "gno.land/p/demo/avl"

var tree avl.Tree
tree.Set("key", "value")
value := tree.Get("key")
```

⚖️ Comparison: `Map` vs `avl.Tree`

Operation	**Map** (Small Data)	**AVL Tree** (Large Data)
Lookup	O(1)	O(log n)
Insert	O(1)	O(log n)
Delete	O(1)	O(log n)
Scalability	Poor	Excellent

🧠 When to Use What?

- 🗺️ **Use Maps when:**
  - Working with small constant datasets
- 🌲 **Use AVL Trees when:**
  - Dealing with dynamic datasets
  - Scalability and efficiency are required

📚 Further Reading

- Why use AVL Trees in Gno
- Gno AVL Tree Documentation
- Effective Gno