Let's Build a Distributed File System

Sanket Patel

Site Reliability Engineer

Agenda

Explore:

- **Typical File Systems**
  - Reads
  - Writes
- **Distributed File Systems**
  - Architecture
  - Reads
  - Writes
- **Demo**

Won’t Explore:

- Individual File System (like ext3)
- Answer to the Ultimate Question of Life, the Universe, and Everything

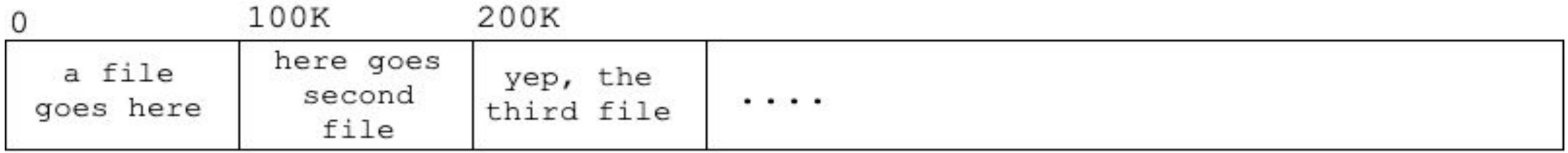
File Systems

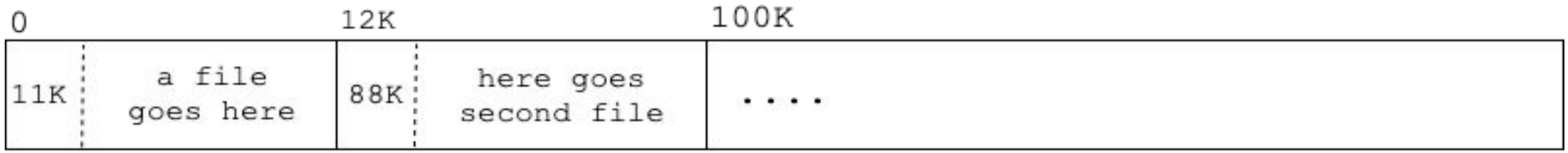
What is it?

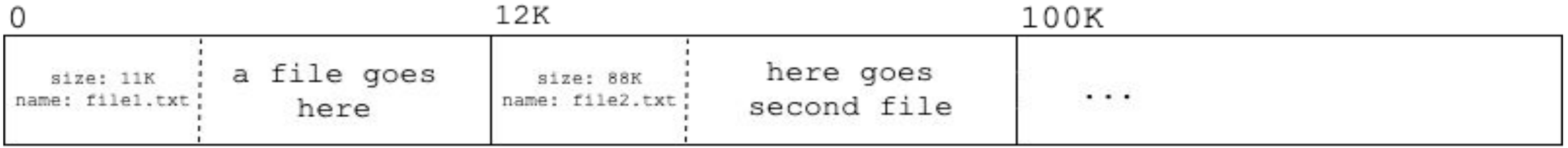
"A structure defined over raw storage space"

- **Made of:** metadata + data

My Own File System

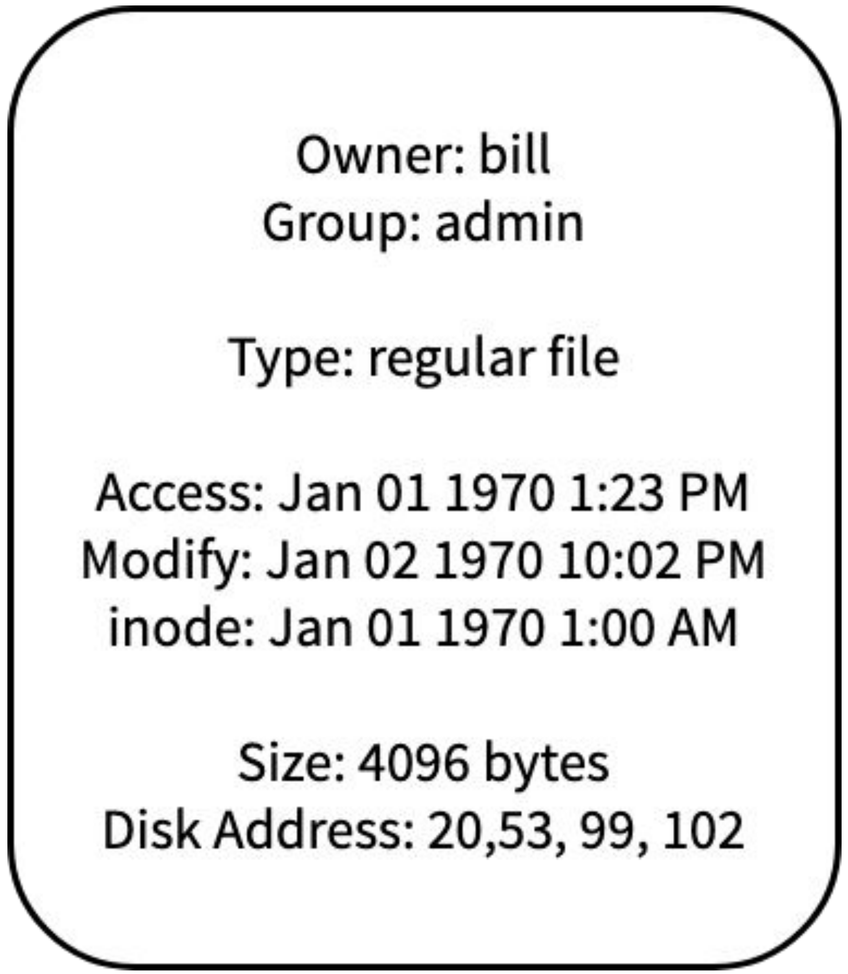






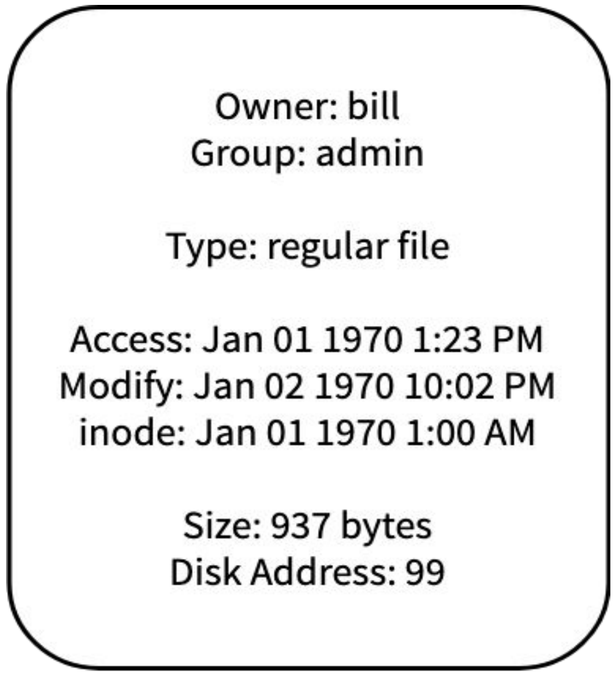
An Inode

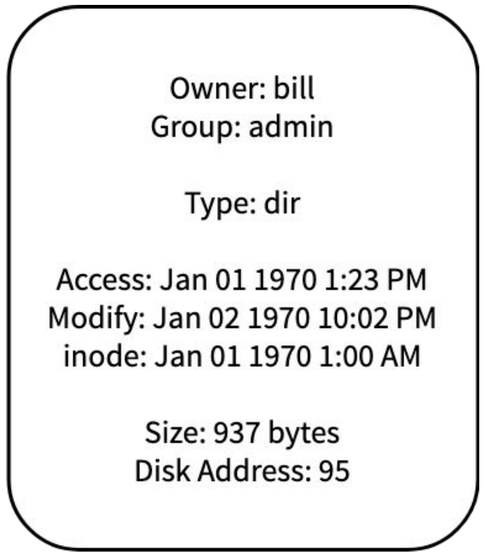
ring any bells?

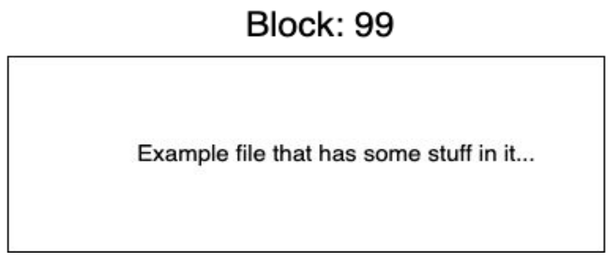


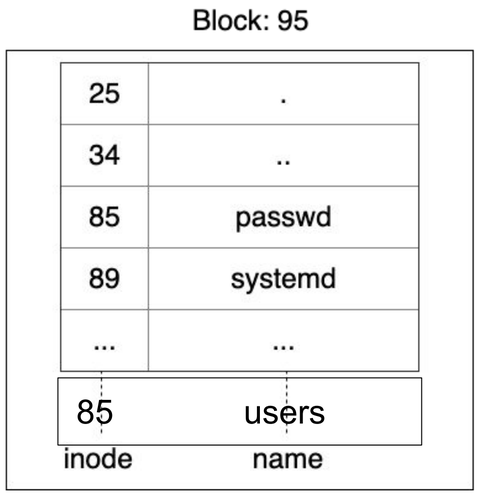
Inode For a File And Dir

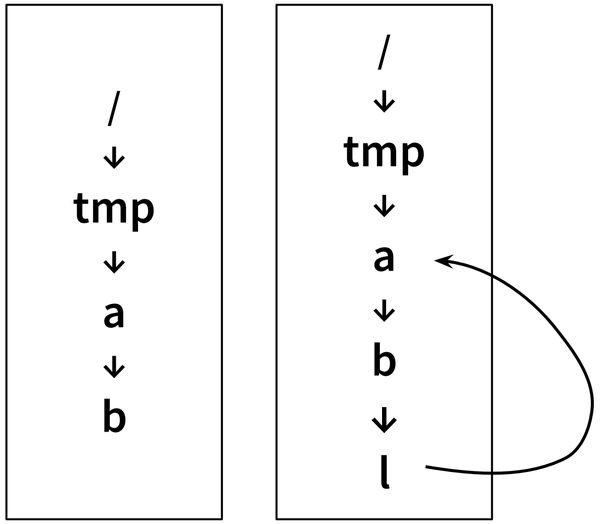
and what does the data looks like







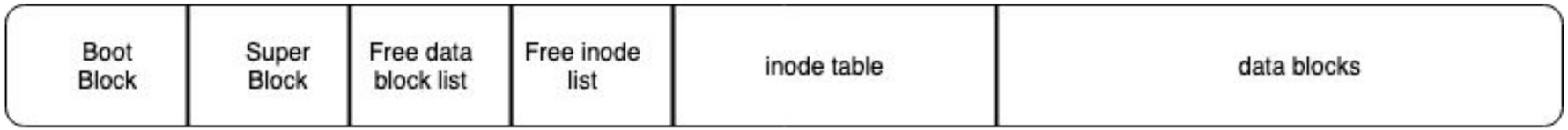




Hard linking Folders

The Raw Storage

and how it looks

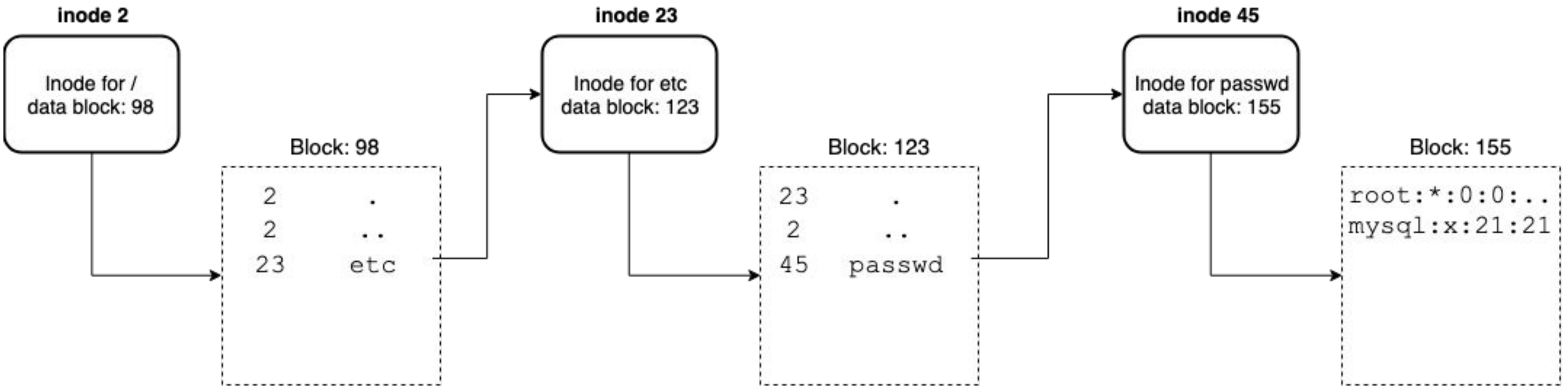


Total data segment size = 1,000,000 bytes
1 block size = 1000 bytes
Total number of blocks = 1000 blocks

Total number of files?

Let’s Read

and see how it goes



Let’s Write

1. Convert name to inode
    - If file does not exist: alloc and init an inode
2. Get block address from offset
    - Load the block in memory
        - Block does not exist: allocate one
3. Modify block
    - Not necessarily put it back on disk

[Distributed] File Systems

What?

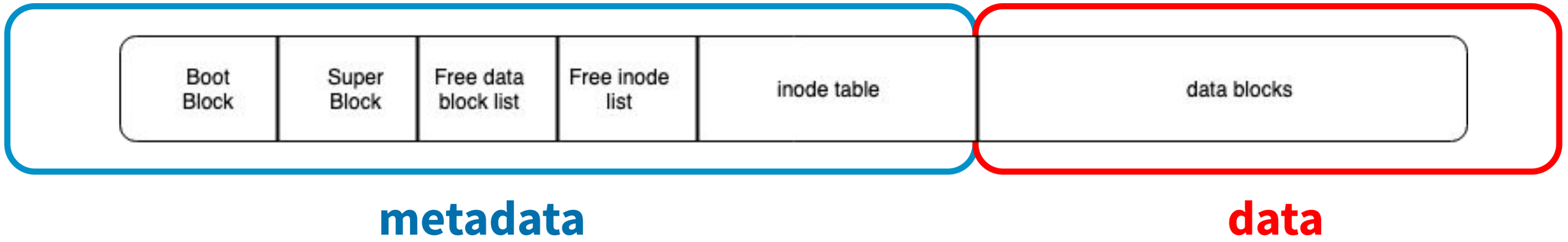
"A file system, but distributed!"

Why Distributed?

- No single point of failure
- Avoid bottleneck
- Scalable storage space

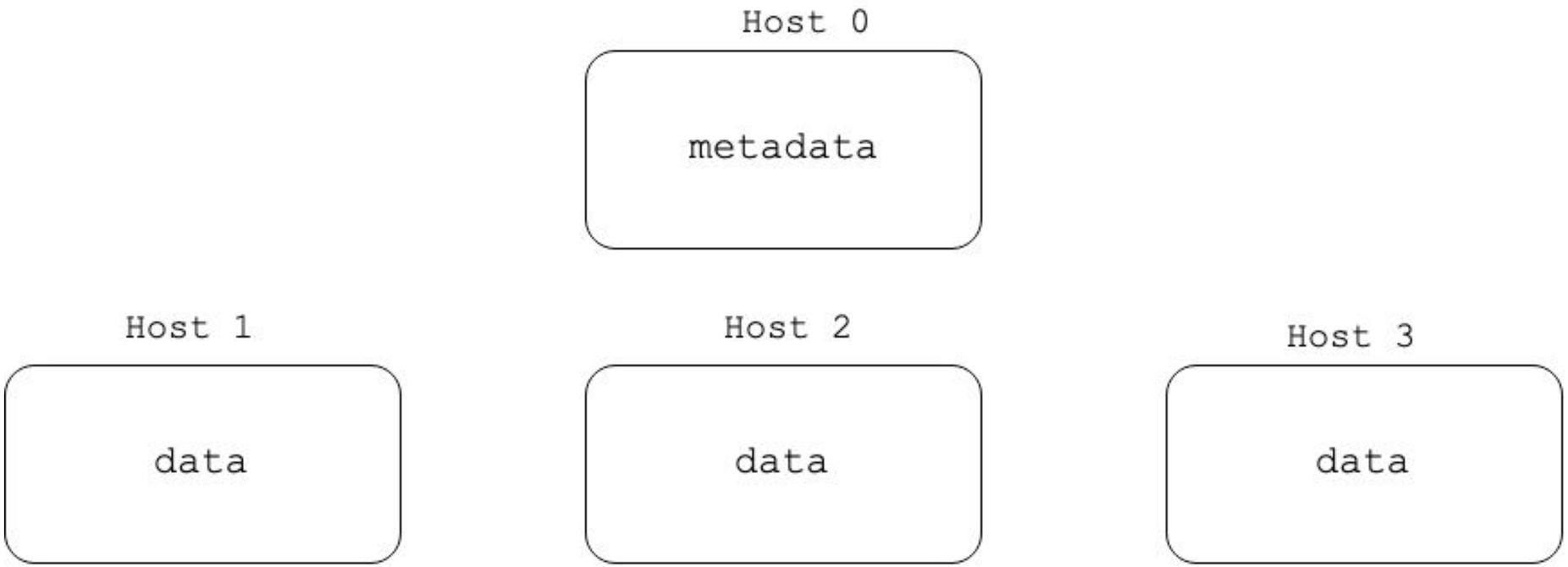
Looking Back...

… a file system is



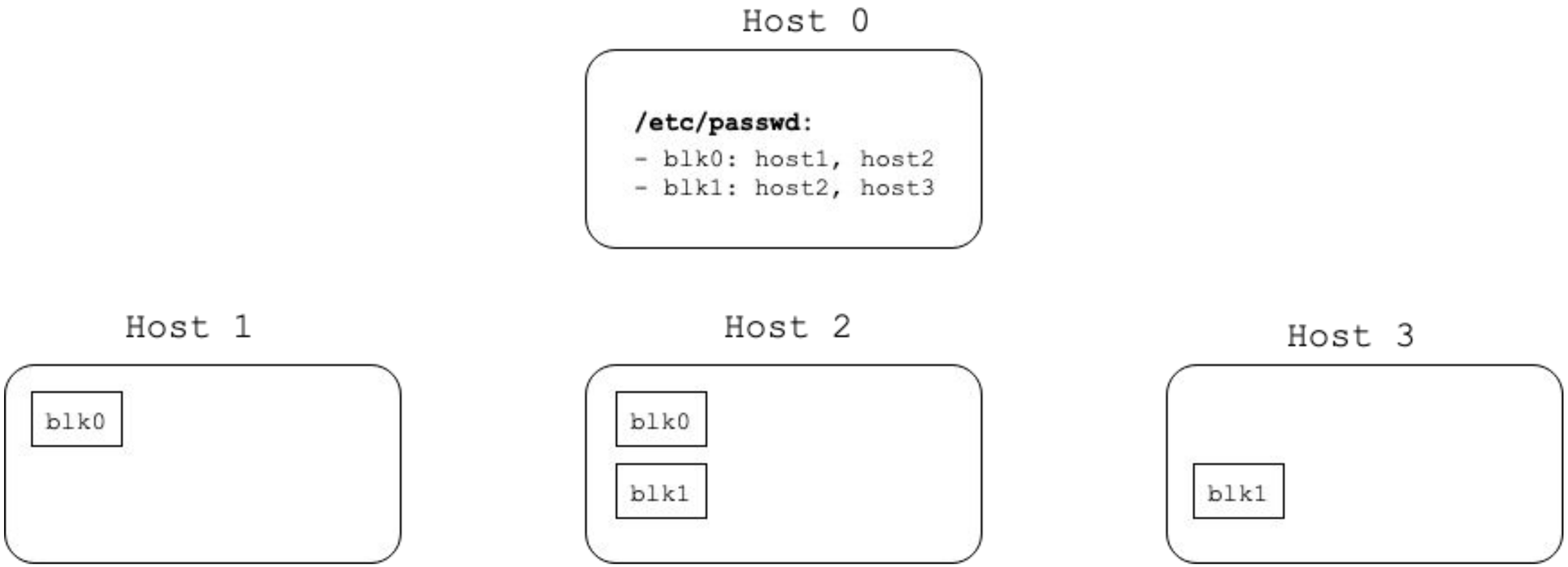
Making it Distributed

would look something like this



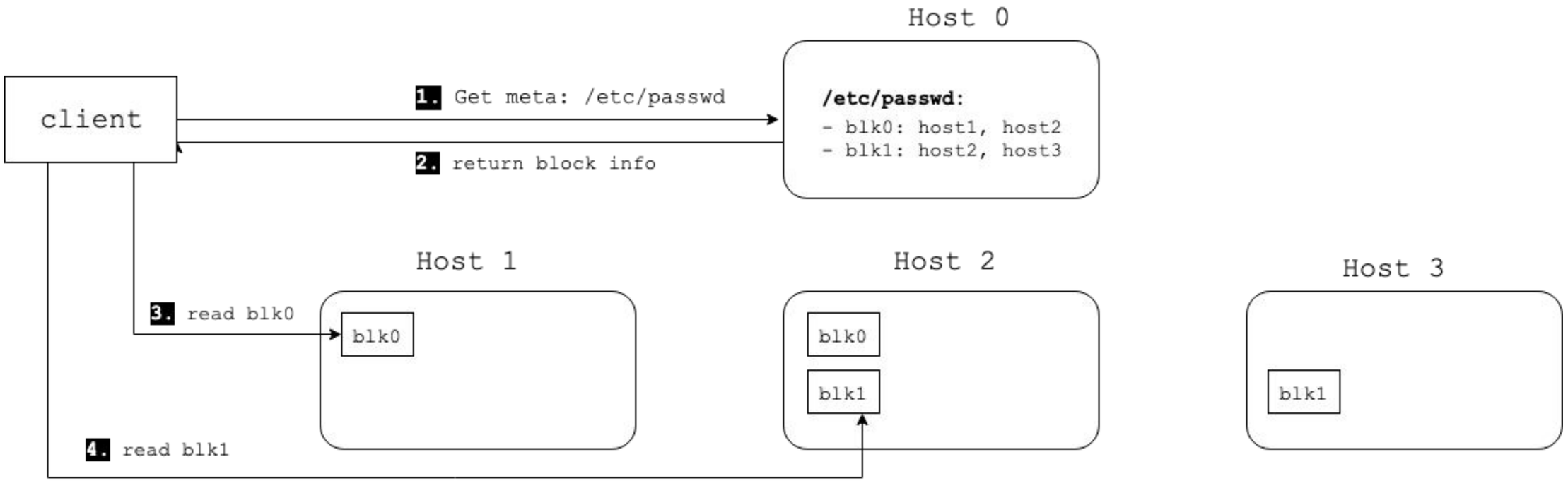
Structure

would look something like this



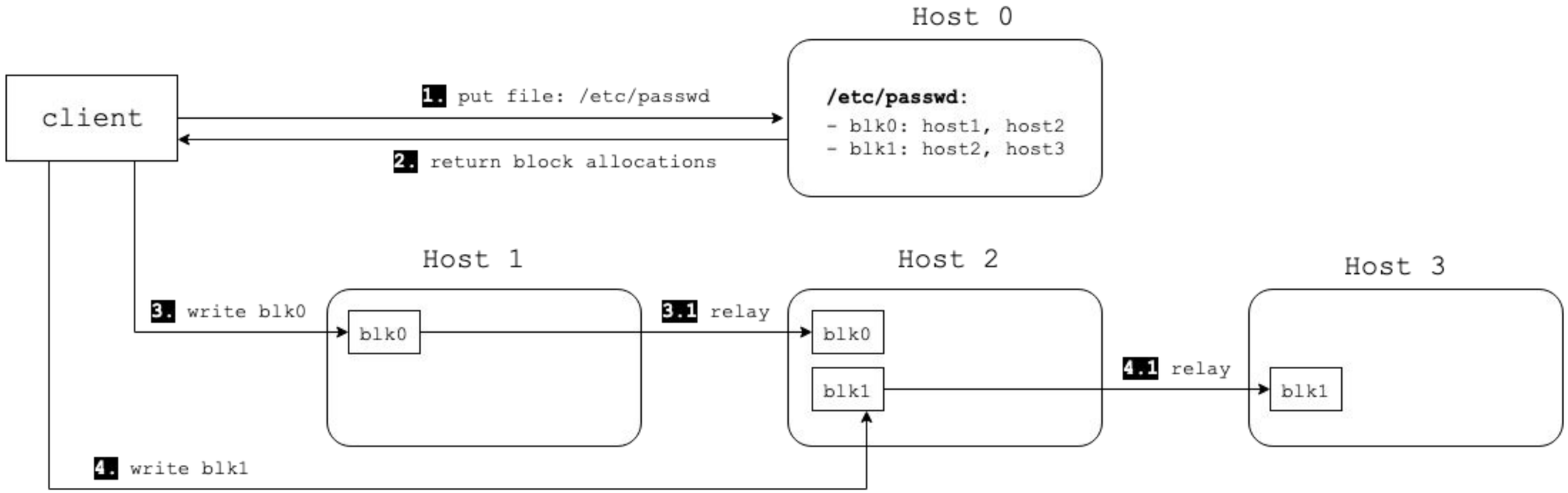
So When You Read...

… it would go like



And When You Write...

… the following happens



Let’s see it in action

https://github.com/sanketplus/PyDFS/tree/srecon

PyDFS

The Greatest DFS Alive

1. Master:
   - metadata storage
2. Minion
   - stores blocks
3. Client
   - to interact with above guys

About Data

… and how it will look

```
example file: /etc/passwd

    file_block = {"/etc/passwd": ["block0", "block1"]}
    block_minion = {"block0": [minion1 ,minion2],
                    "block1": [minion2, minion3]}
    minions = {
      "minion1": (host1, portX),
      "minion2": (host2, portY),
      "minion3": (host3, portZ)
    }
```

1. `replication_factor`: how many copies to make of a block
2. `block_size`: what should be size of each block
3. block placement strategy: random

Master API

… and how it will serve you

```
def read(file)
 returns: [
            {"block_id": "block1", "block_addr": [(host1,portX),...]},
            {"block_id": "block2", "block_addr: [(host2,portY),...]"}
          ]


def write(file, size)
 returns: [
            {"block_id": "block1", "block_addr": [(host1,portX),...]},
            {"block_id": "block2", "block_addr: [(host2,portY),...]"}
           ]
```

Minion API

… and how it will obey

```
def put(block_id, data, minions)
=> writes the block on local disk and forward to minions

def get(block_id)
=> reads the block and returns the contents

def forward(block_id, data, minions)

=> calls put() on next minion with remaining minions as
forward list
```

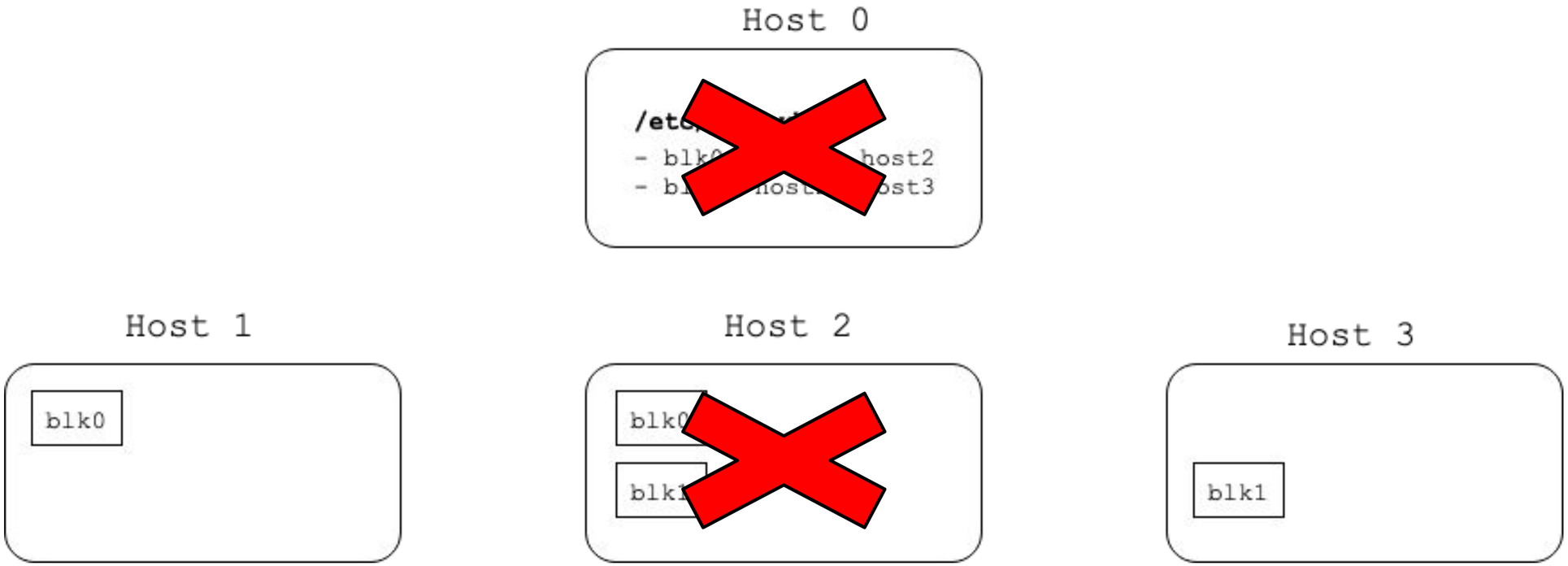
Onwards

Check these out

- HDFS
- HopsFS
- Perkeep

Problems

because, why not?



Limitations

of course

1. Latency
2. Operational Complexity
3. Small Files
4. Usage Patterns

That’s all for today...

My home is @ https://sanket.plus