Merkle-tree-based integrity verification protocol for geo-distributed storage systems

Santo Cariotti

Alma Mater Studiorum
Università di Bologna

October 30, 2025

Content

1. What kind of problem are we facing?

Content

1. What kind of problem are we facing?
2. How did we manage to solve this?

Content

1. What kind of problem are we facing?
2. How did we manage to solve this?
3. How about scalability?

Problem

Given a filesystem, how can I be sure that every file keeps integrity?

```
root@node:/root# ls -lh
total 200G
-rw-r--r-- 1 root   root    21M Jan  1 00:00 photo1.png
-rw-r--r-- 1 root   root   4.4K Jan 10 08:30 photo2.png
-rw-r--r-- 1 root   root     2G Jan  7 09:50 photo3.png
-rw-r--r-- 1 root   root   7.1M Mar  9 11:00 photo4.png
-rw-r--r-- 1 root   root    24K Mar  9 11:01 photo5.png
-rw-r--r-- 1 root   root   130M Jun 25 18:25 photo6.png
...
```

Problem

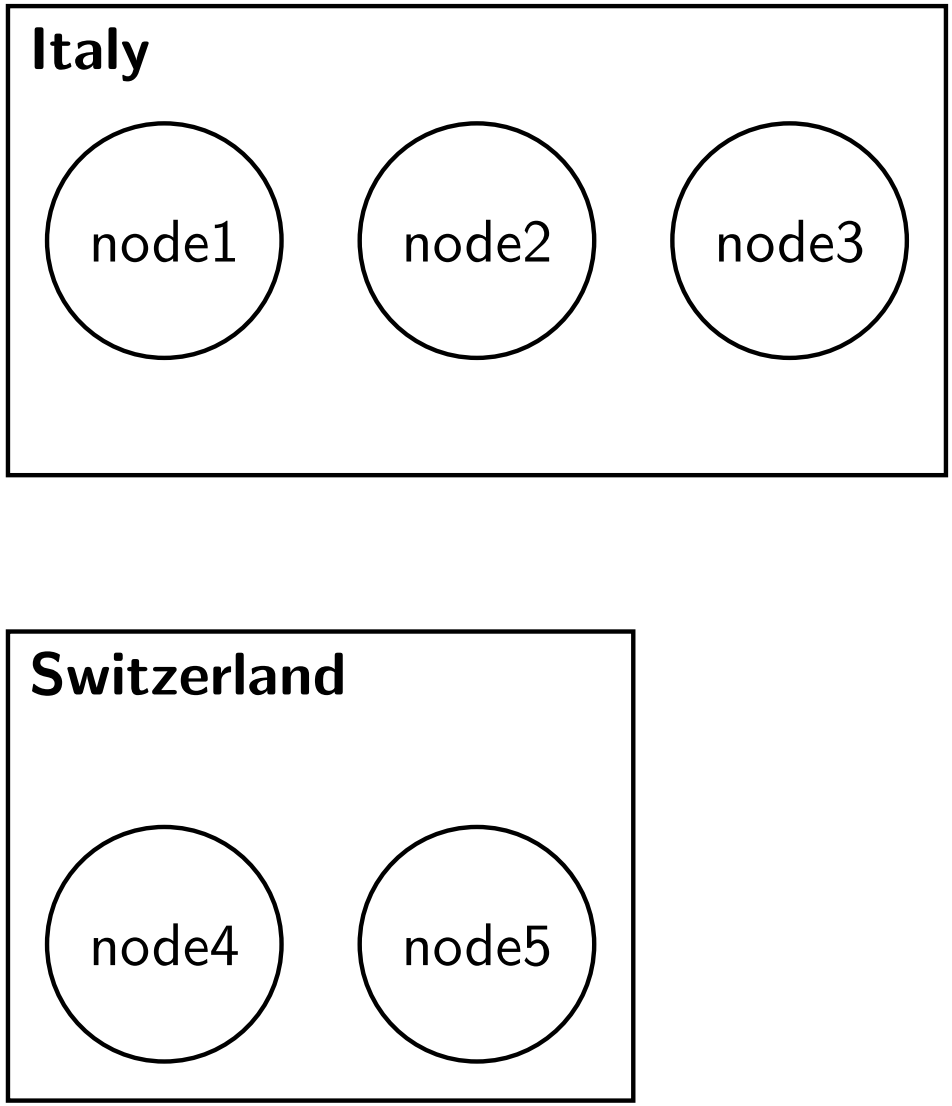
Given a filesystem, how can I be sure that every file keeps integrity?

```
root@node:/root# ls -lh
total 200G
-rw-r--r-- 1 root   root    21M Jan  1 00:00 photo1.png
-rw-r--r-- 1 root   root   4.4K Jan 10 08:30 photo2.png
-rw-r--r-- 1 root   root     2G Jan  7 09:50 photo3.png
-rw-r--r-- 1 root   root   7.1M Mar  9 11:00 photo4.png
-rw-r--r-- 1 root   root    24K Mar  9 11:01 photo5.png
-rw-r--r-- 1 root   root   130M Jun 25 18:25 photo6.png
...
```

We could use a checksum system to find out that the bordered file is corrupted.

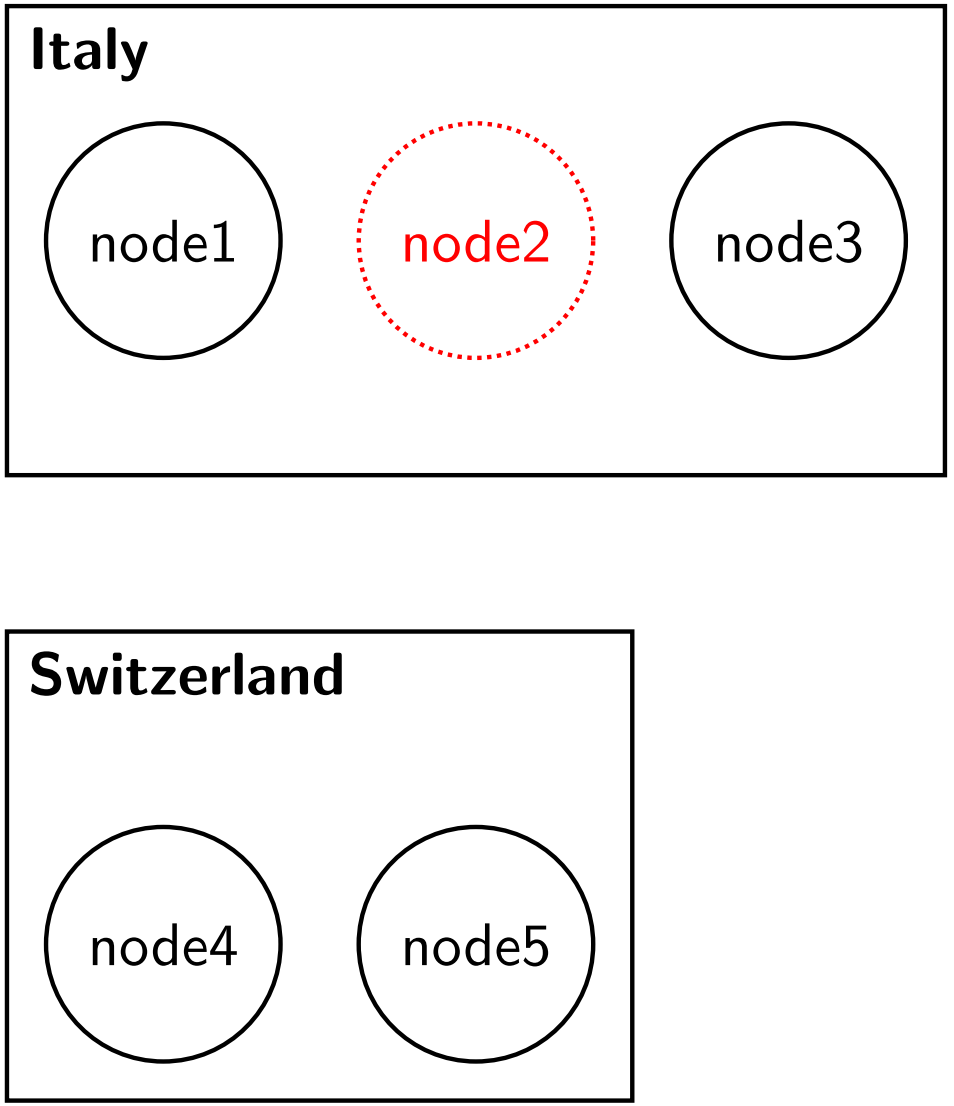
Problem

But, if we have more than one filesystem distributed across different regions?



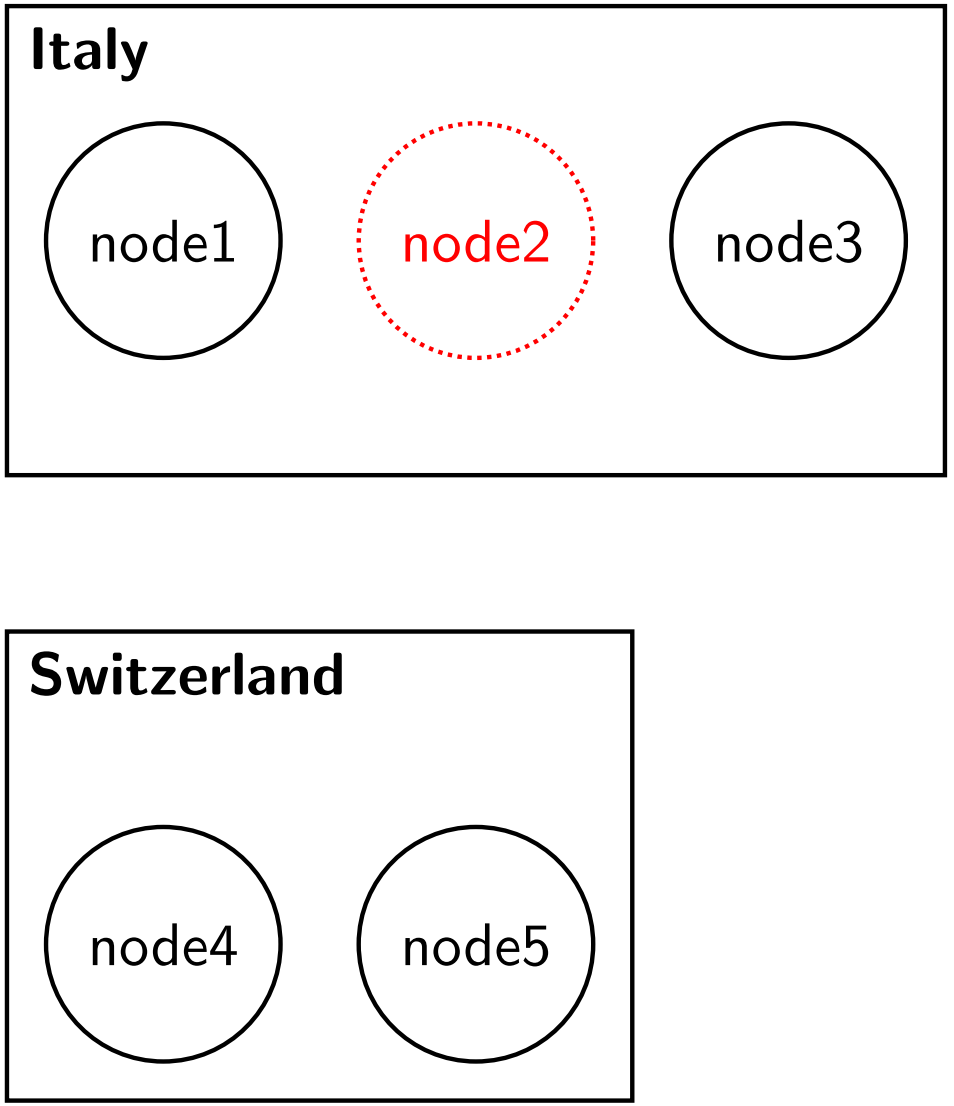
Problem

But, if we have more than one filesystem distributed across different regions? We should make a checksum check for each node.



Problem

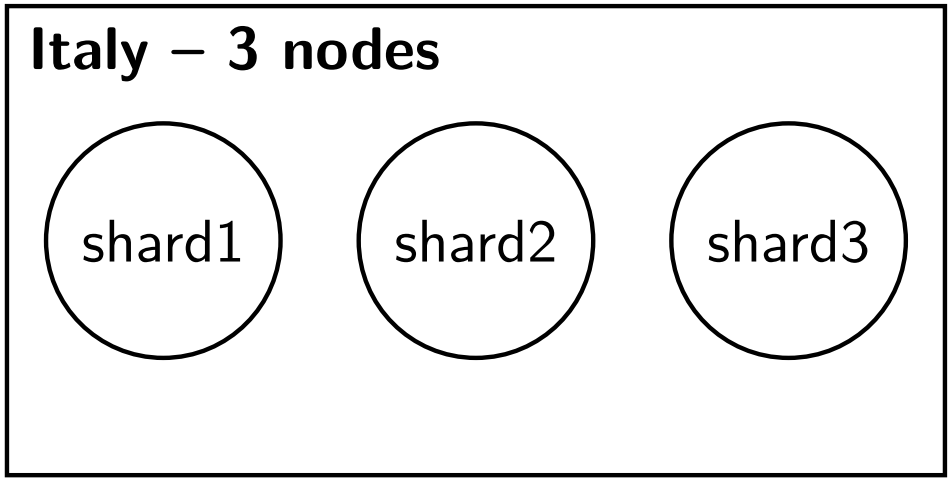
But, if we have more than one filesystem distributed across different regions? We should make a checksum check for each node. **Too complex.**

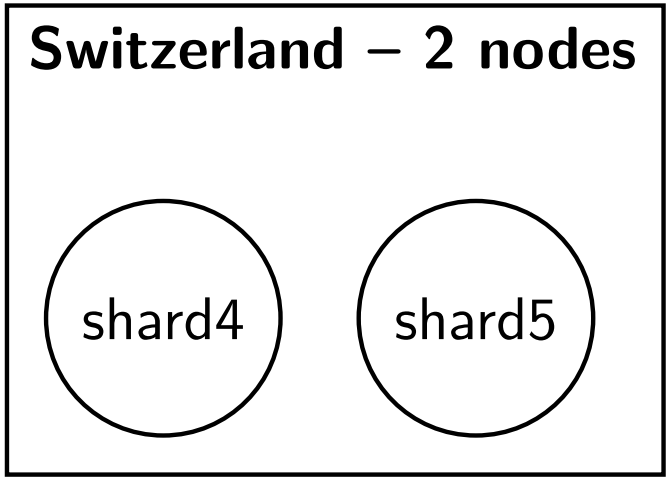


Cubbit, a geo-distributed storage system.

Cubbit – How it works

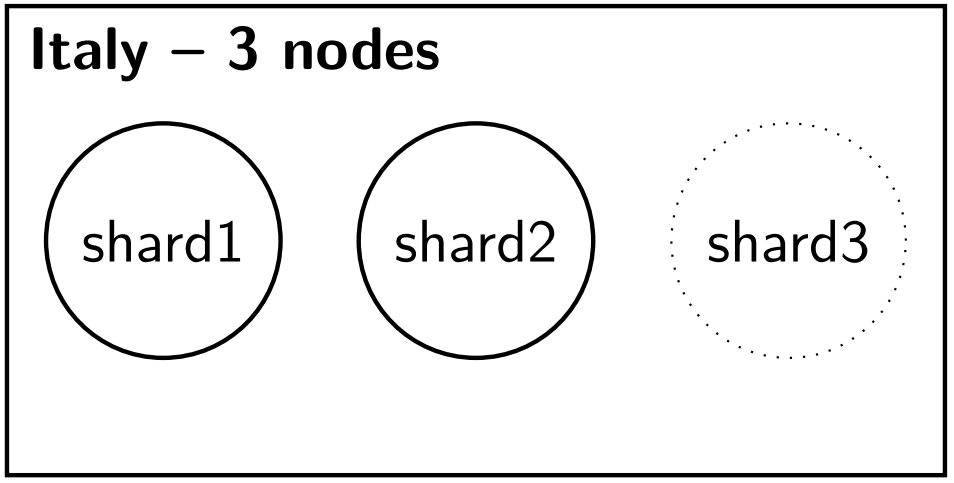
A file is split in $n + k$ shards.
We need at least n shards to reconstruct the entire file, k shards are used as redundancy.

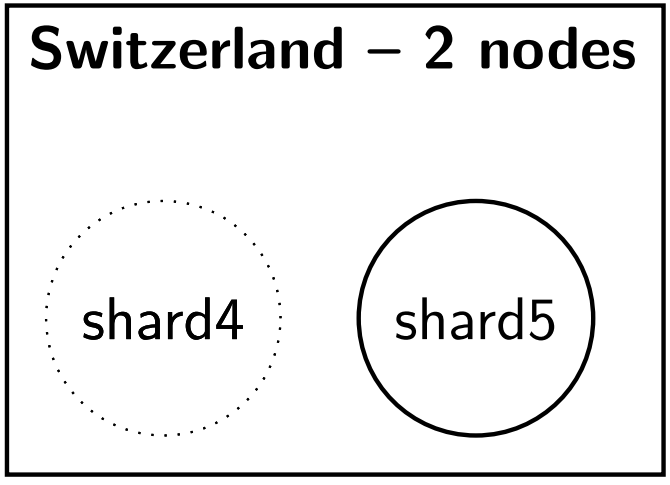




Cubbit – Problems using checksum

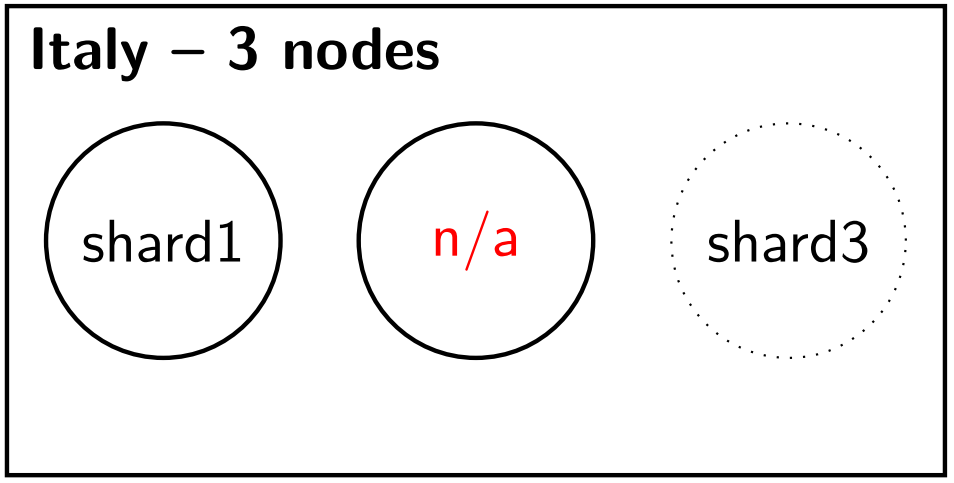
1. If nodes are offline, can't check all shards for a file.

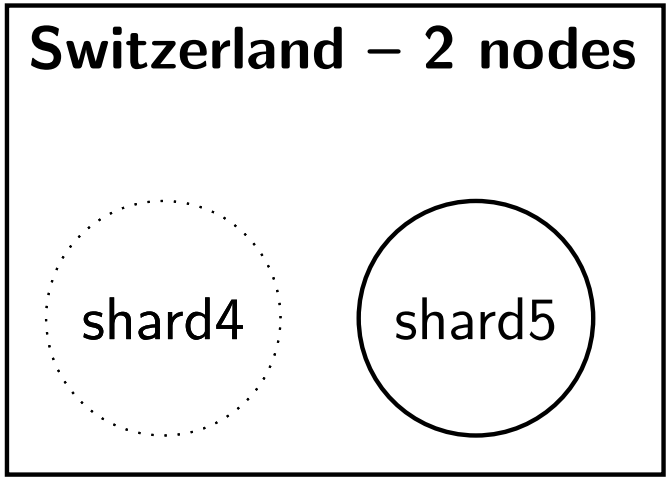




Cubbit – Problems using checksum

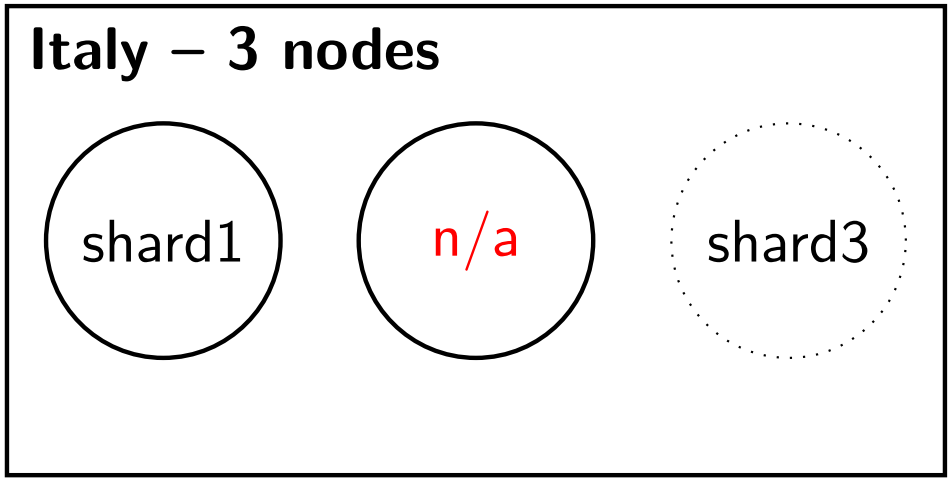
1. If nodes are offline, can't check all shards for a file.
2. During an upload some agents can be offline, but they could be online during the check.

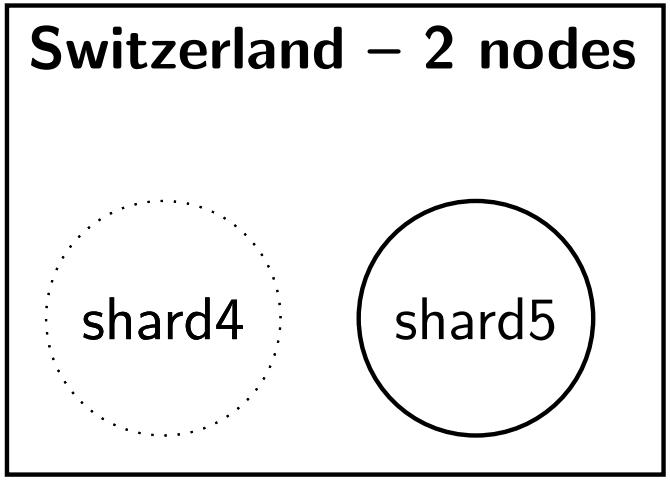




Cubbit – Problems using checksum

1. If nodes are offline, can't check all shards for a file.
2. During an upload some agents can be offline, but they could be online during the check.
3. Check for each reconstructed file or for each shard?

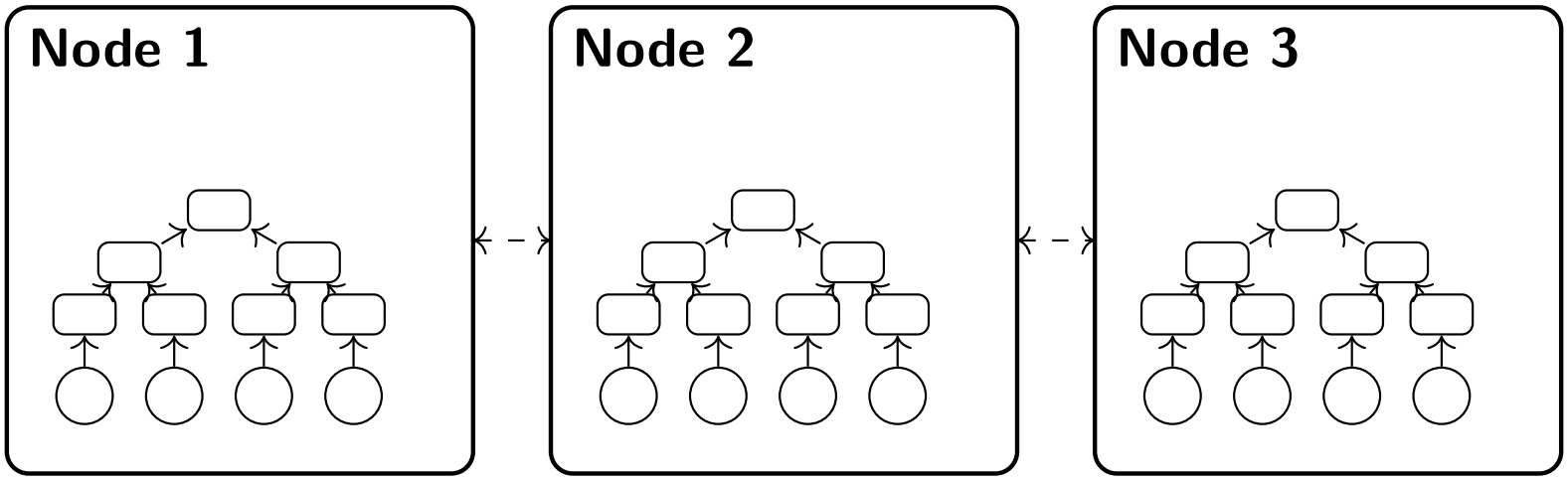




Solution

Solution

Each node uses a Merkle-tree-structure to organize shards during the integrity verification. Every node agree on what file is corrupted thanks to Raft. Data are organized using Reed-Solomon codes.

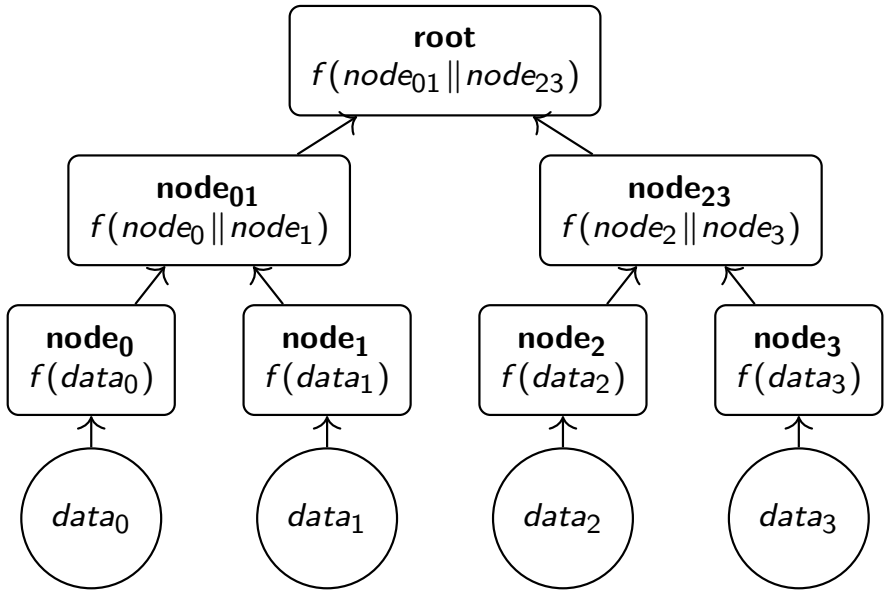


Some background

Merkle trees

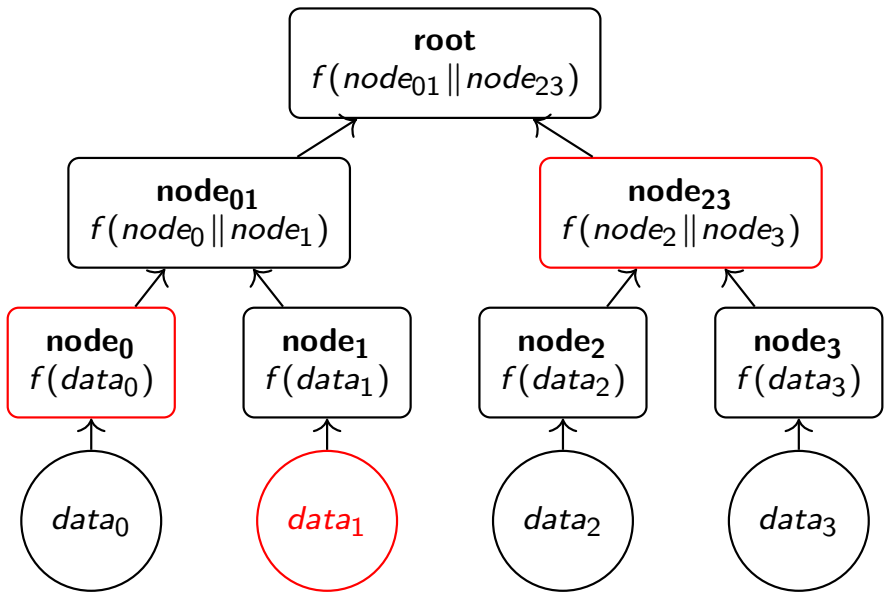
It is a binary tree T of height H with 2^H leaves and $2^H - 1$ internal nodes. Each leaf stores the cryptographic hash of the underlying data, rather than the raw data itself. The same cryptographic hash function is applied recursively at internal nodes, which store the hash of the concatenation of their two children.

$$n_{parent} = f(n_{left} || n_{right})$$



Merkle trees – Proof

Merkle trees has the ability to prove that a given piece of data is part of a larger set, without revealing or recomputing the entire dataset. For $data_1$ we have $\pi = \{node_0, node_{23}\}$.

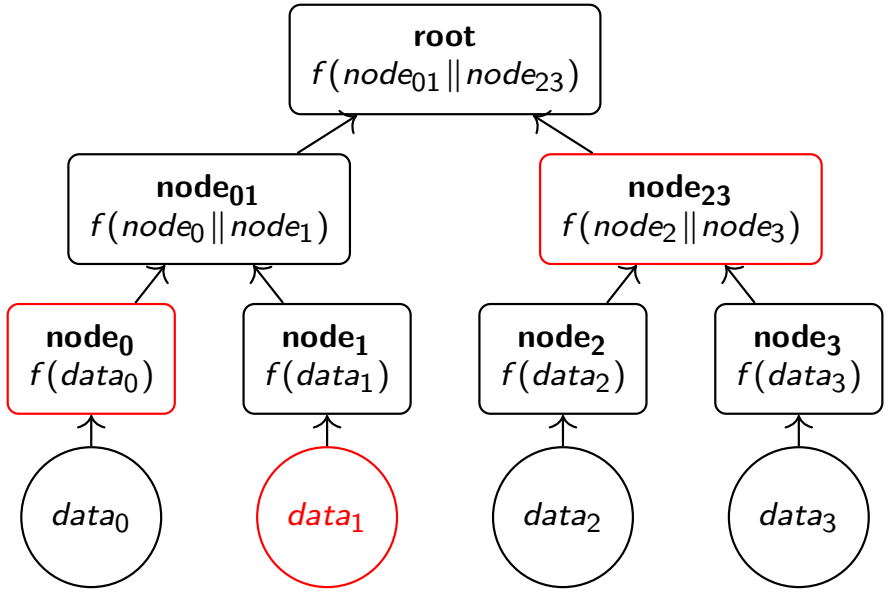


Merkle trees – Proof

Given a path π we can make a proof verification comparing the result with the presumed root in $O(\log n)$.

Input: Data d, proof π, expected root R, hash function f
Output: `true` if valid, `false` otherwise

```
1 h ← f(d)
2 foreach (sibling, position) in π do
3 |   if position = Left then
4 |   |   h ← f(sibling||h)
5 |   else
6 |   |   h ← f(h||sibling)
7 |   end
8 end
9 return h = R
```



Merkle trees – Lib

A dedicated library written in Rust to retreive Merkle tree root hash fast.

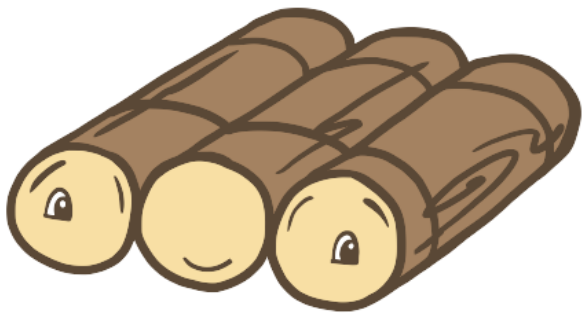
Merkle trees – Lib

A dedicated library written in Rust to retreive Merkle tree root hash fast.

Hash function	**5 MB**	**10 MB**	**15 MB**
SHA-256	89.901 ms	178.42 ms	268.53 ms
Keccak-256	521.49 ms	1.1334 s	1.3438 s
BLAKE3	73.091 ms	154.68 ms	219.79 ms

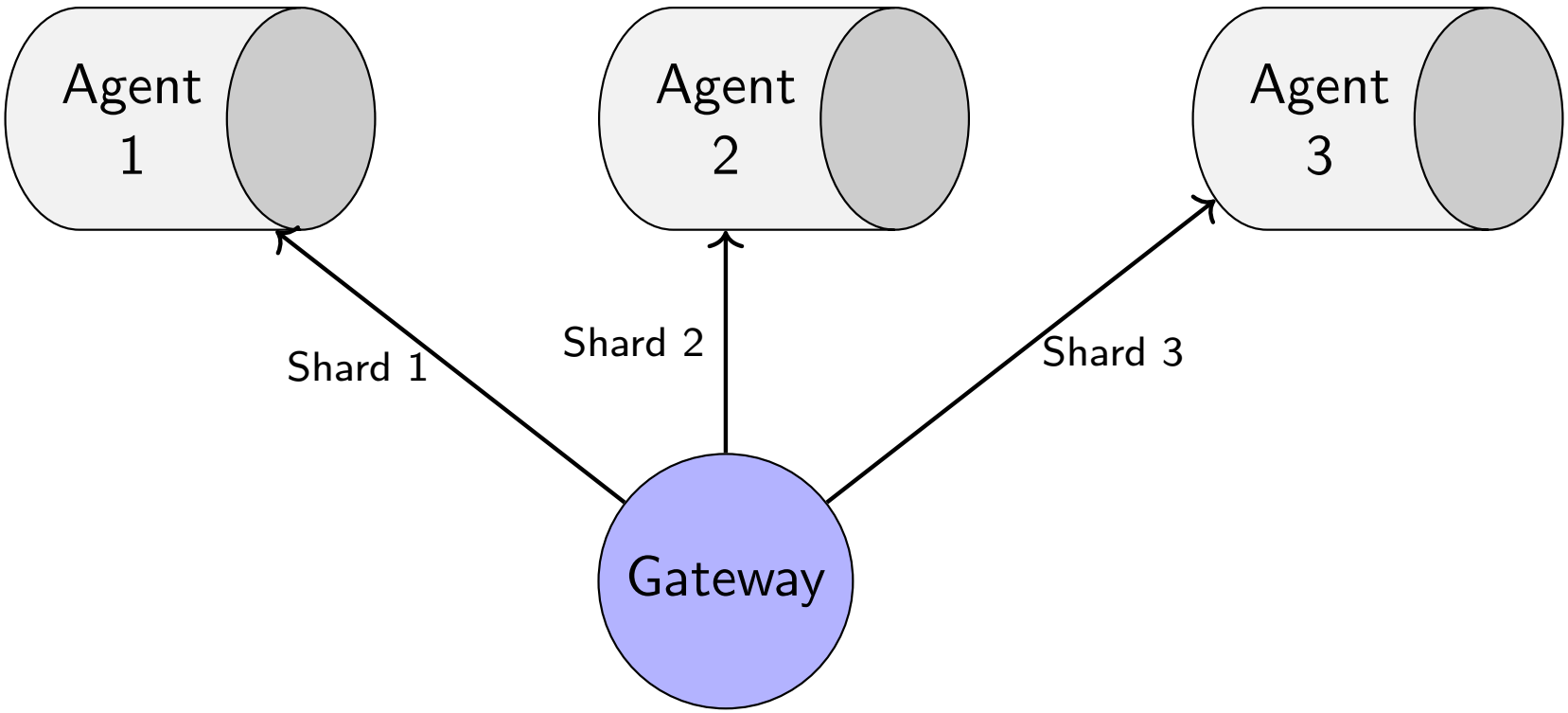
Raft

A consensus protocol, where each server on a cluster is a follower, a candidate or a leader. There is only leader and it is responsible to send messages to other servers via a log.



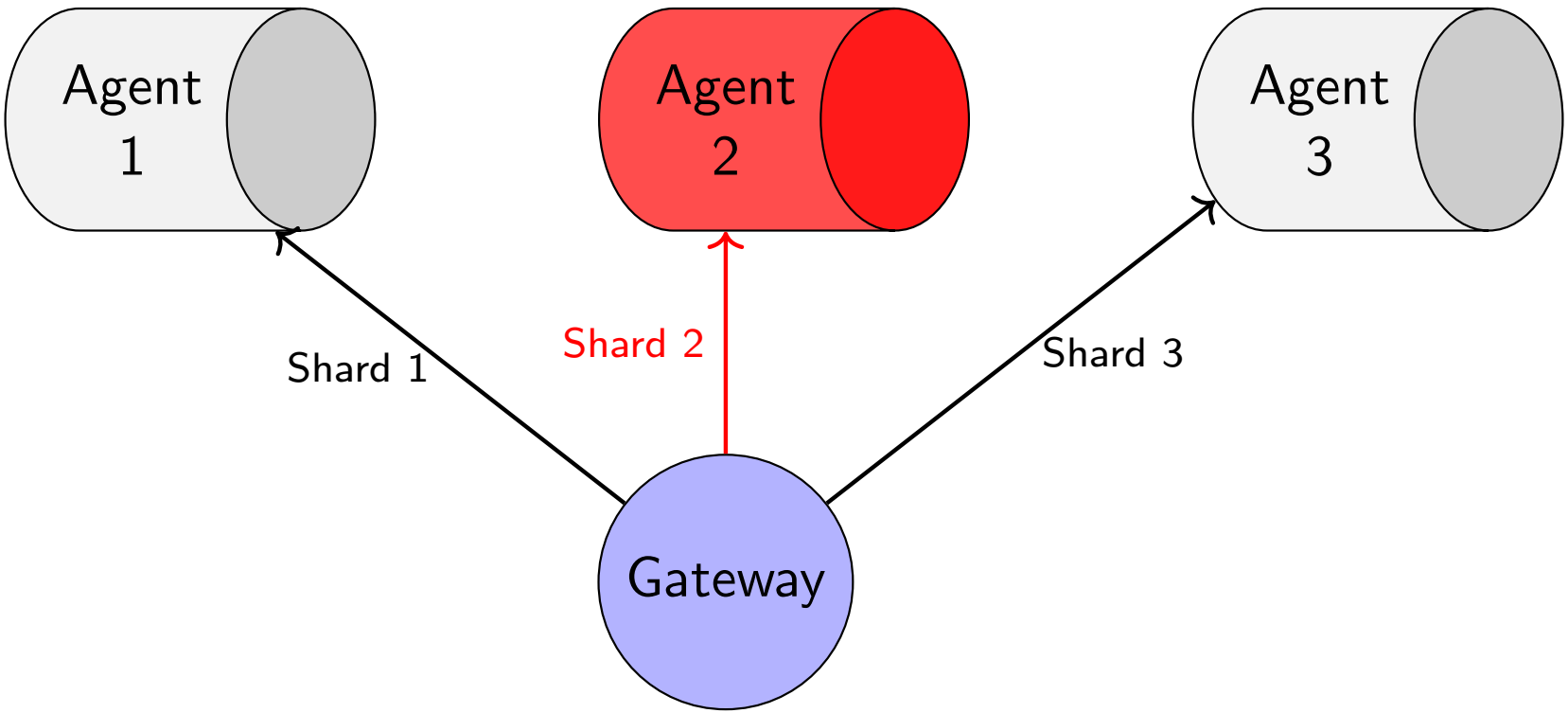
Reed-Solomon

During an upload, a file is split in $n + k$ shards and send each shard to a different node.



Reed-Solomon

Up to k agents could be offline, and the upload/download of files still works. We should make a recoverage of the missing shard when the agent comes back online.



Let's put all together

Upload flow

Let's save our Christmas holidays family picture on Cubbit.

Upload flow

Let's save our Christmas holidays family picture on Cubbit.

- Select a file. (e.g., xmas2024.png)

Upload flow

Let's save our Christmas holidays family picture on Cubbit.

- Select a file. (e.g., xmas2024.png)
- Rename the file using a random string. (e.g., ff4c4b3)

Upload flow

Let's save our Christmas holidays family picture on Cubbit.

- Select a file. (e.g., xmas2024.png)
- Rename the file using a random string. (e.g., ff4c4b3)
- Use Reed-Solomon and send each shard to a different node. (e.g., ff4c4b3.1 to Agent 1)

Upload flow

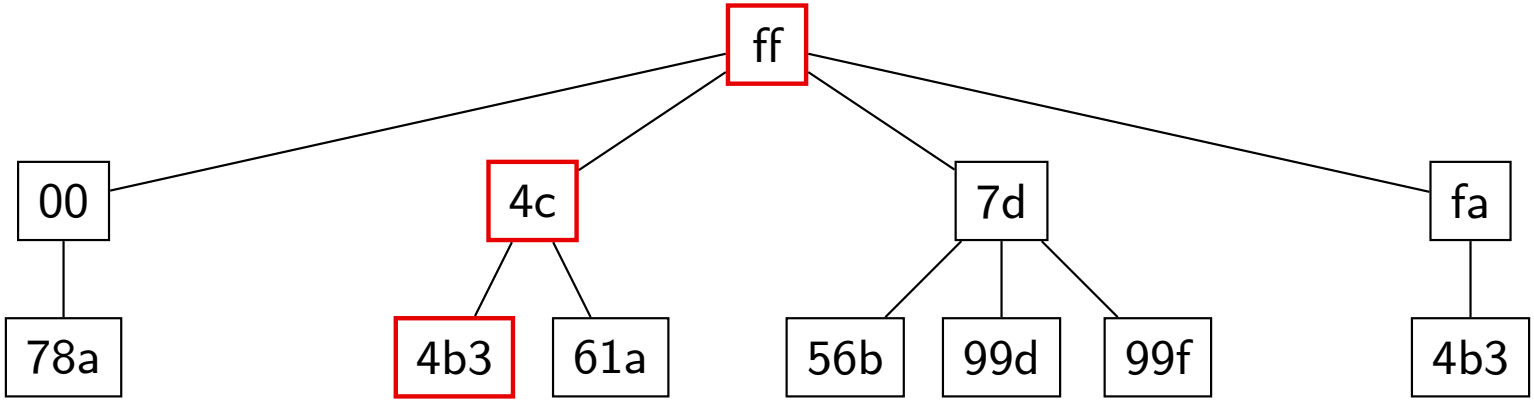
Let's save our Christmas holidays family picture on Cubbit.

- Select a file. (e.g., xmas2024.png)
- Rename the file using a random string. (e.g., ff4c4b3)
- Use Reed-Solomon and send each shard to a different node. (e.g., ff4c4b3.1 to Agent 1)
- Store the file using a two-level of folders. (e.g. ff/4c/4b3.1 for Agent 1)

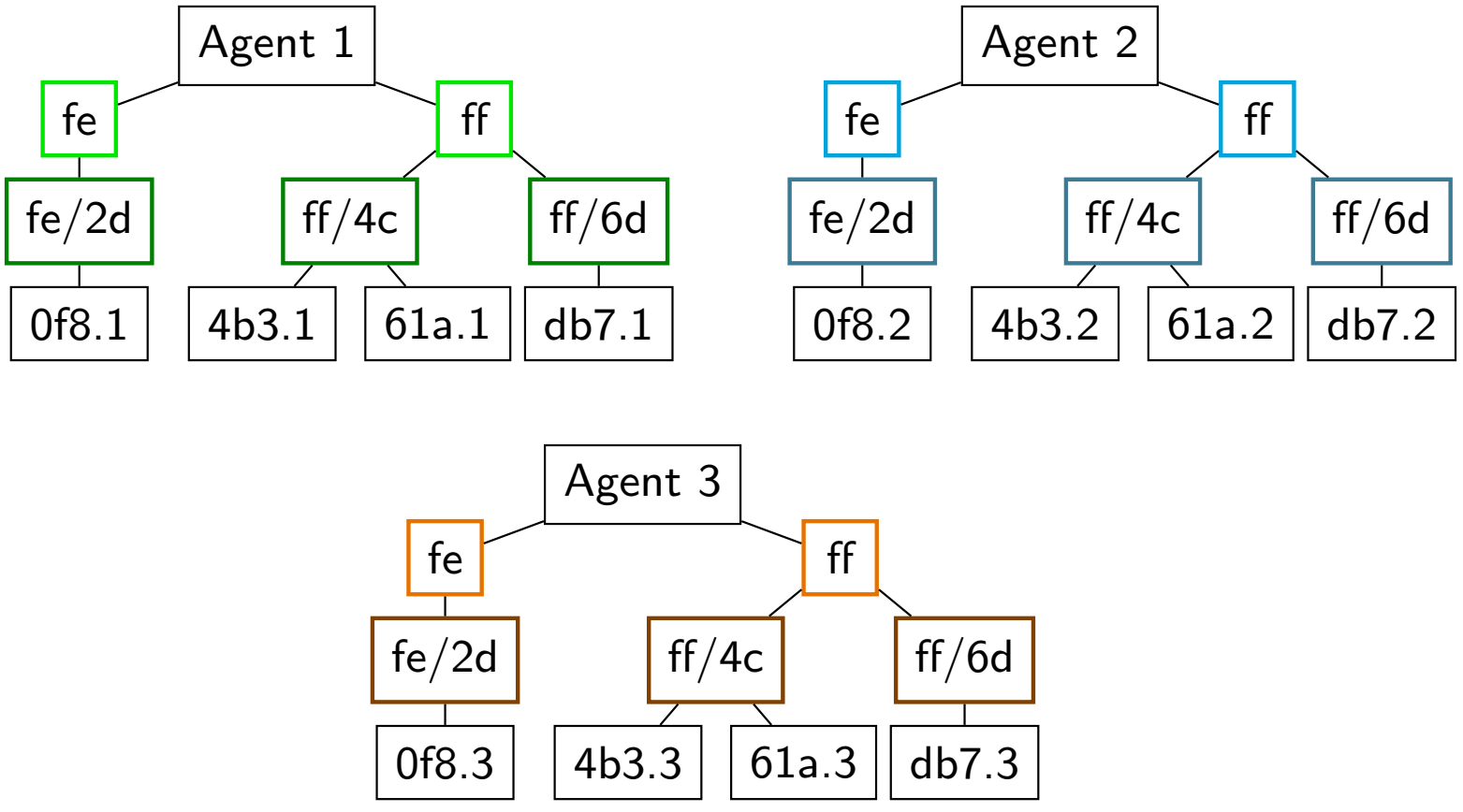
Upload flow – Directory tree

Let's save our Christmas holidays family picture on Cubbit.

- Select a file. (e.g., xmas2024.png)
- Rename the file using a random string. (e.g., ff4c4b3)
- Use Reed-Solomon and send each shard to a different node. (e.g., ff4c4b3.1 to Agent 1)
- Store the file using a two-level of folders. (e.g. ff/4c/4b3.1 for Agent 1)

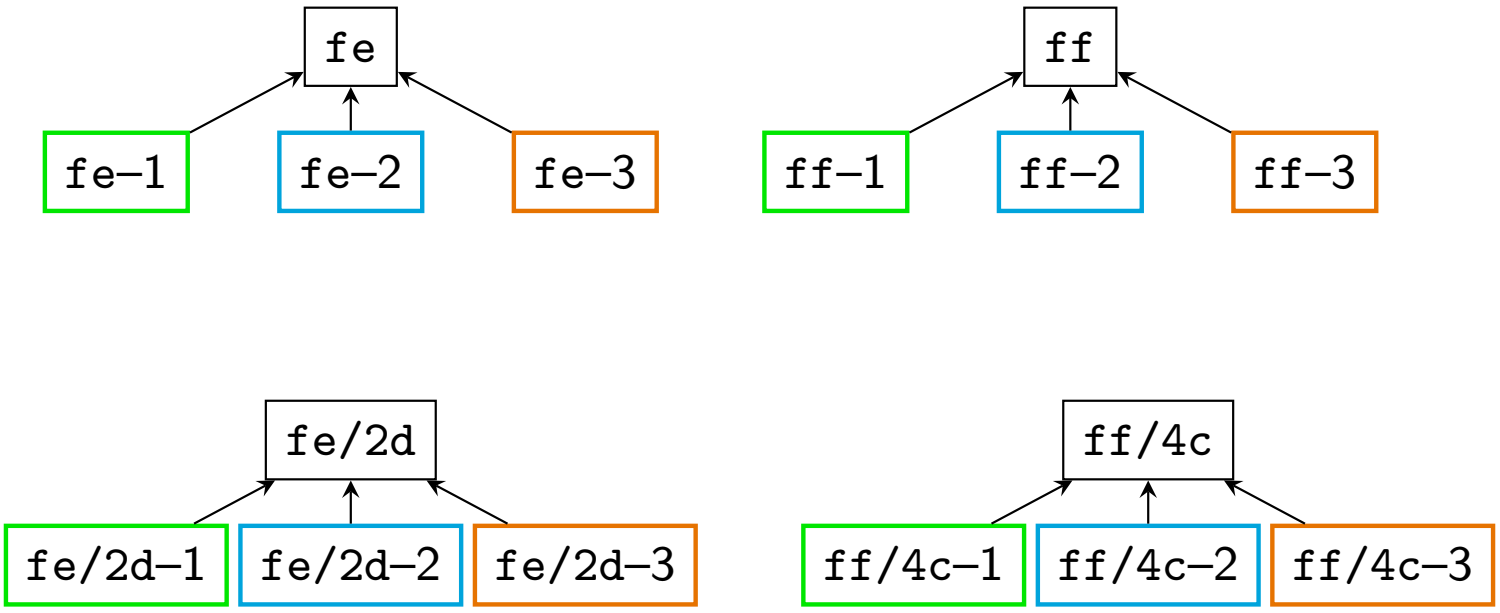


Directory trees as Merkle forest



Every Agent sends Merkle tree root hashes for top-level and second-level folders to the Agent leader.

Aggregated roots



The Agent leader computes other Merkle trees for each folder using the Merkle tree hashes for each Agent.

Map of hashes

```
roots = {
    "fe":    "new root hash of fe",
    "ff":    "new root hash of ff",
    "ff/4c":"new root hash of ff/4c",
    # ...
}

agent_roots = {
    "fe": [
        "Agent 1 root hash of fe",
        "Agent 2 root hash of fe",
        "Agent 3 root hash of fe"
    ],
    # ...
    "ff/4c": [
        "Agent 1 root hash of ff/4c",
        "Agent 2 root hash of ff/4c",
        "Agent 3 root hash of ff/4c"
    ],
}
```

Map of hashes

This map allows us to:

- Use `agent_roots[<some_folder>][i-th-agent]` when the i-th Agent is offline.
- If an Agent is offline during the upload, the `roots[<some_folder>]` value is still computed using the current status of every Agent.

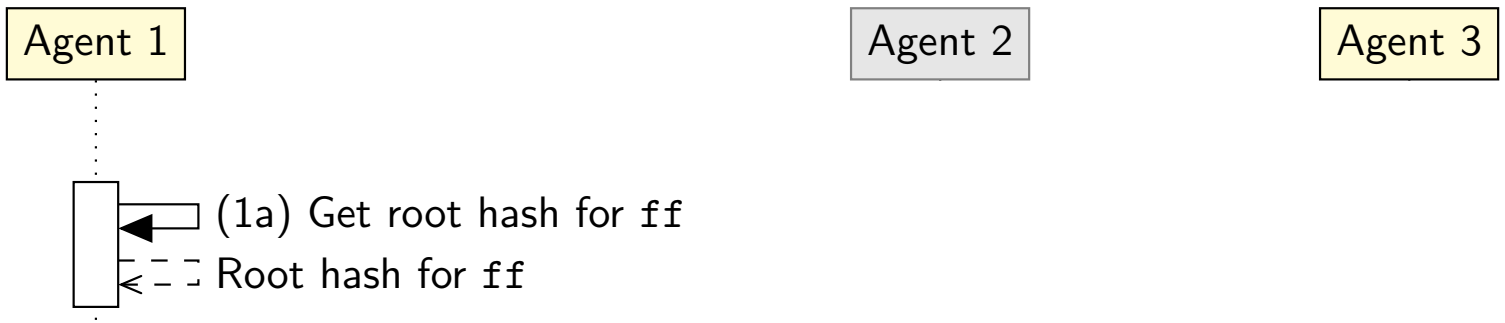
Check corruptions

Agent 1

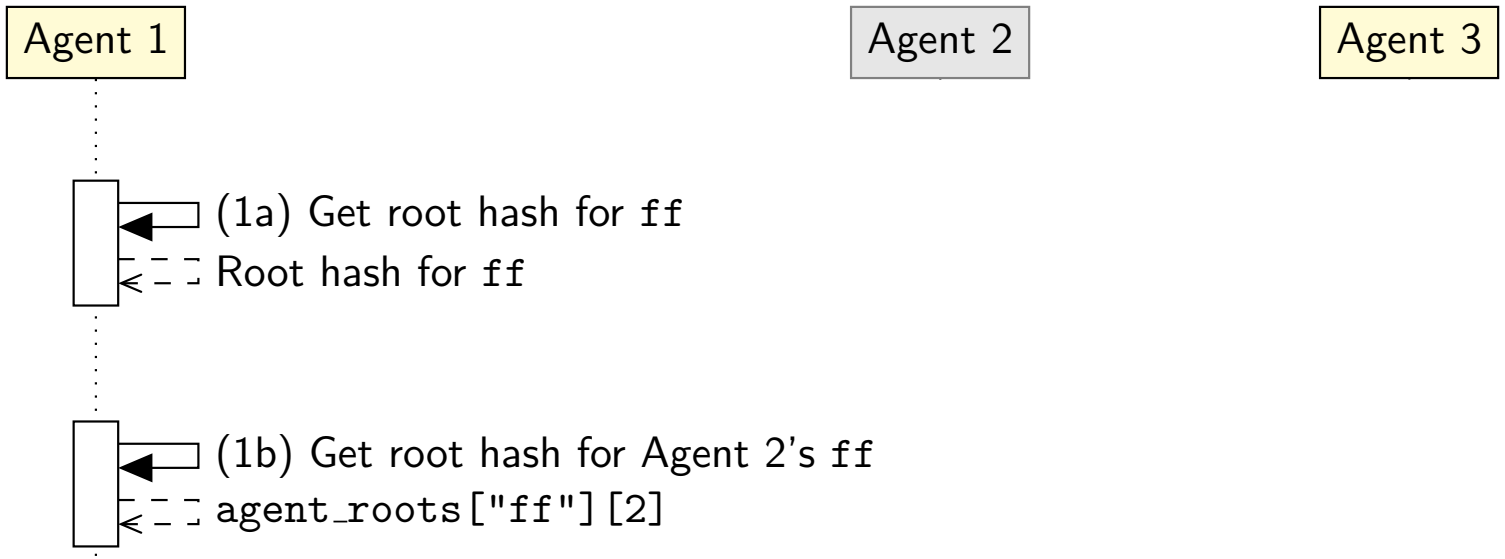
Agent 2

Agent 3

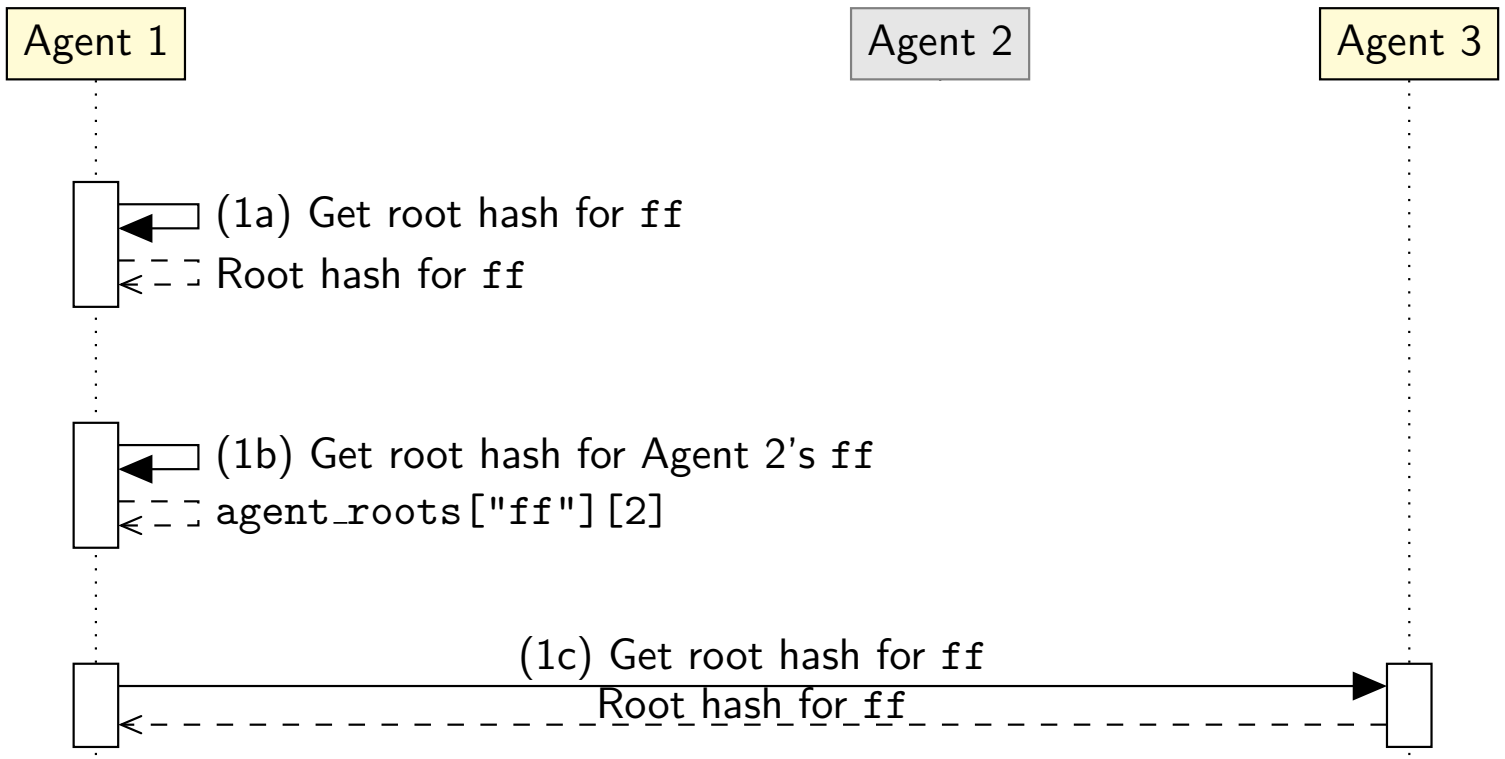
Check corruptions



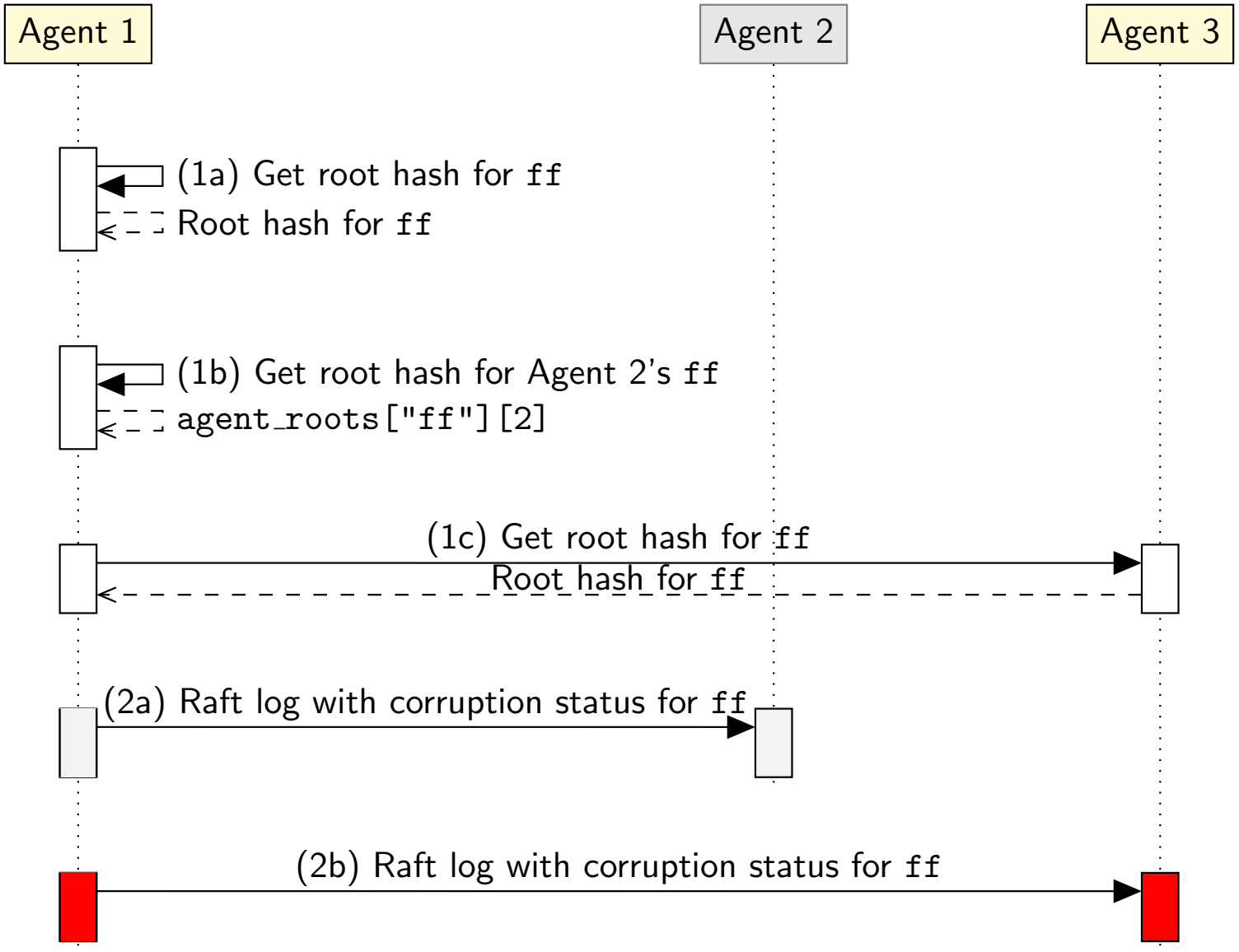
Check corruptions



Check corruptions



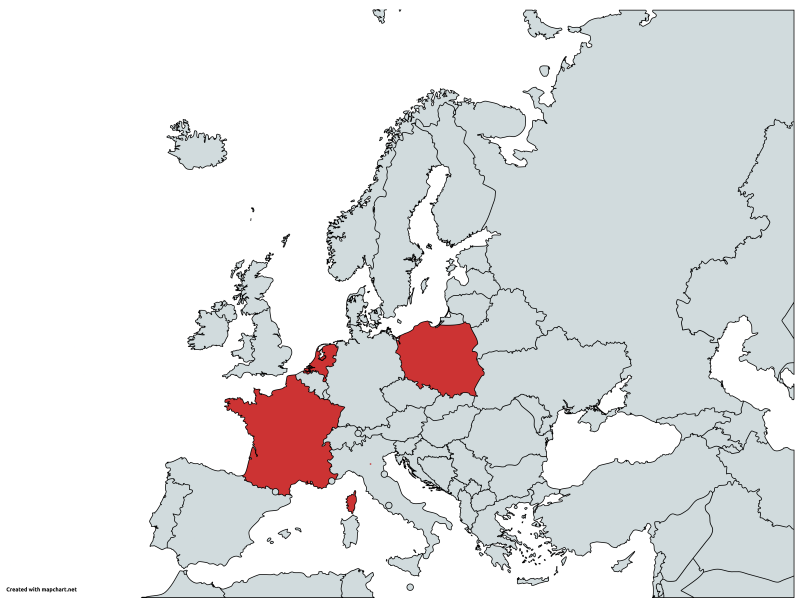
Check corruptions



Tests

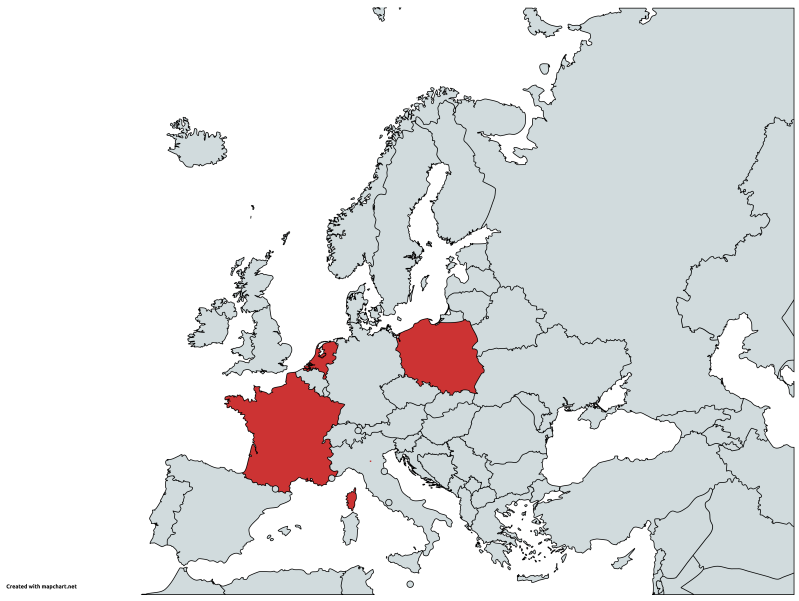
Set up

8 Agents with 4 CPU(s), 8-16 GB of RAM, and 45 GB of Disk.



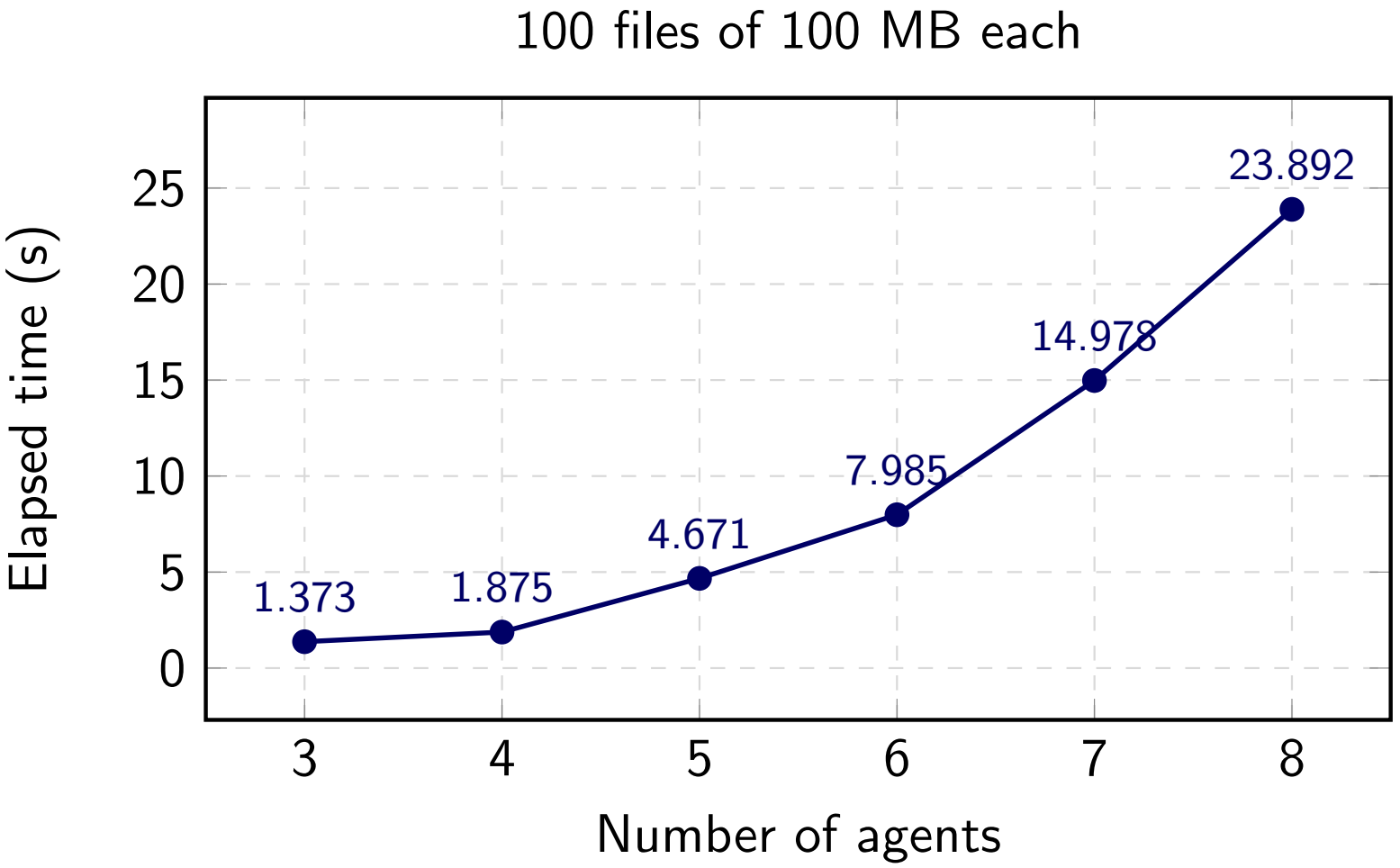
Set up

8 Agents with 4 CPU(s), 8-16 GB of RAM, and 45 GB of Disk.

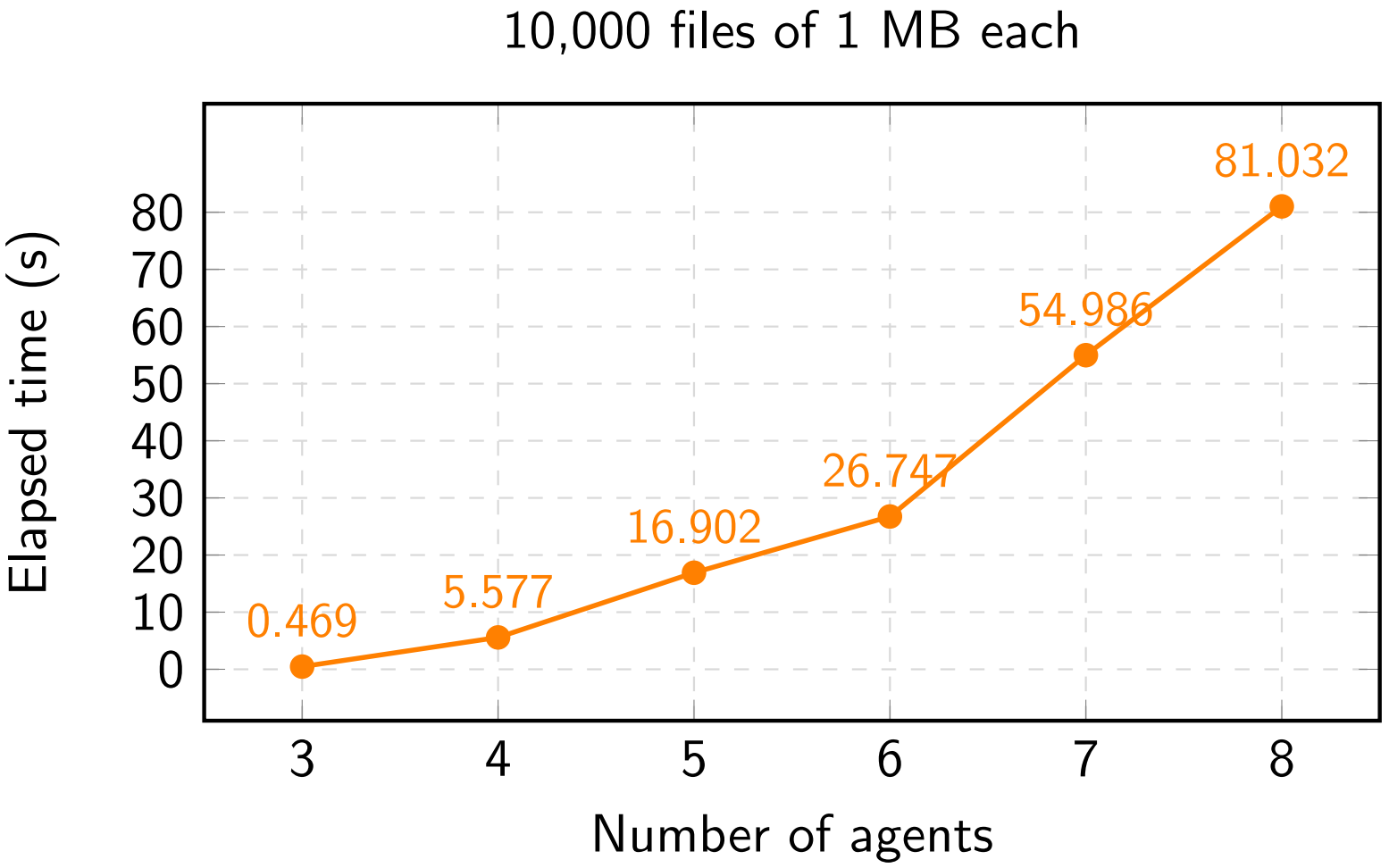


Per each test let's consider: number of online/offline Agents and file sizes and counts.

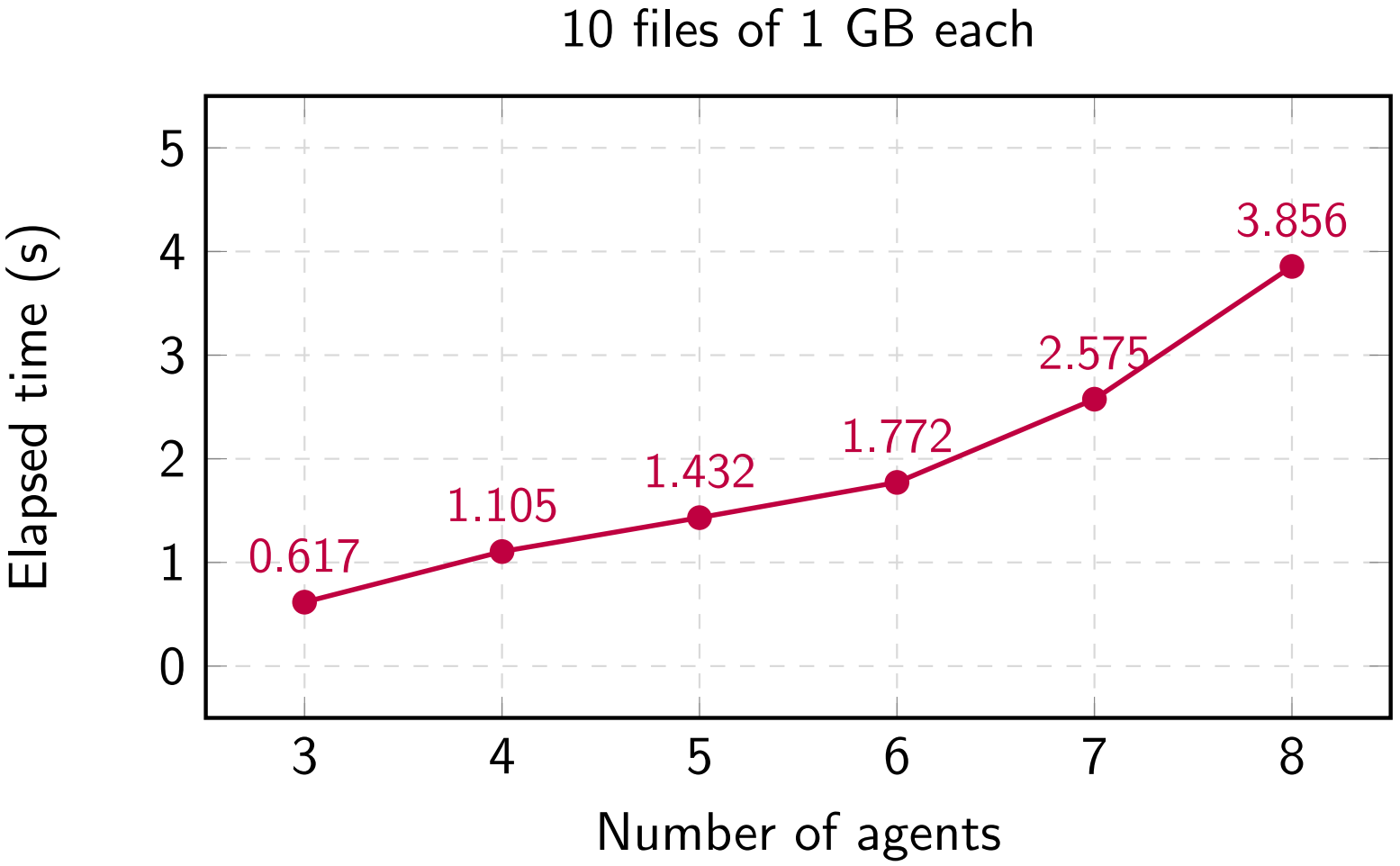
Same dataset but different file size and count – 1



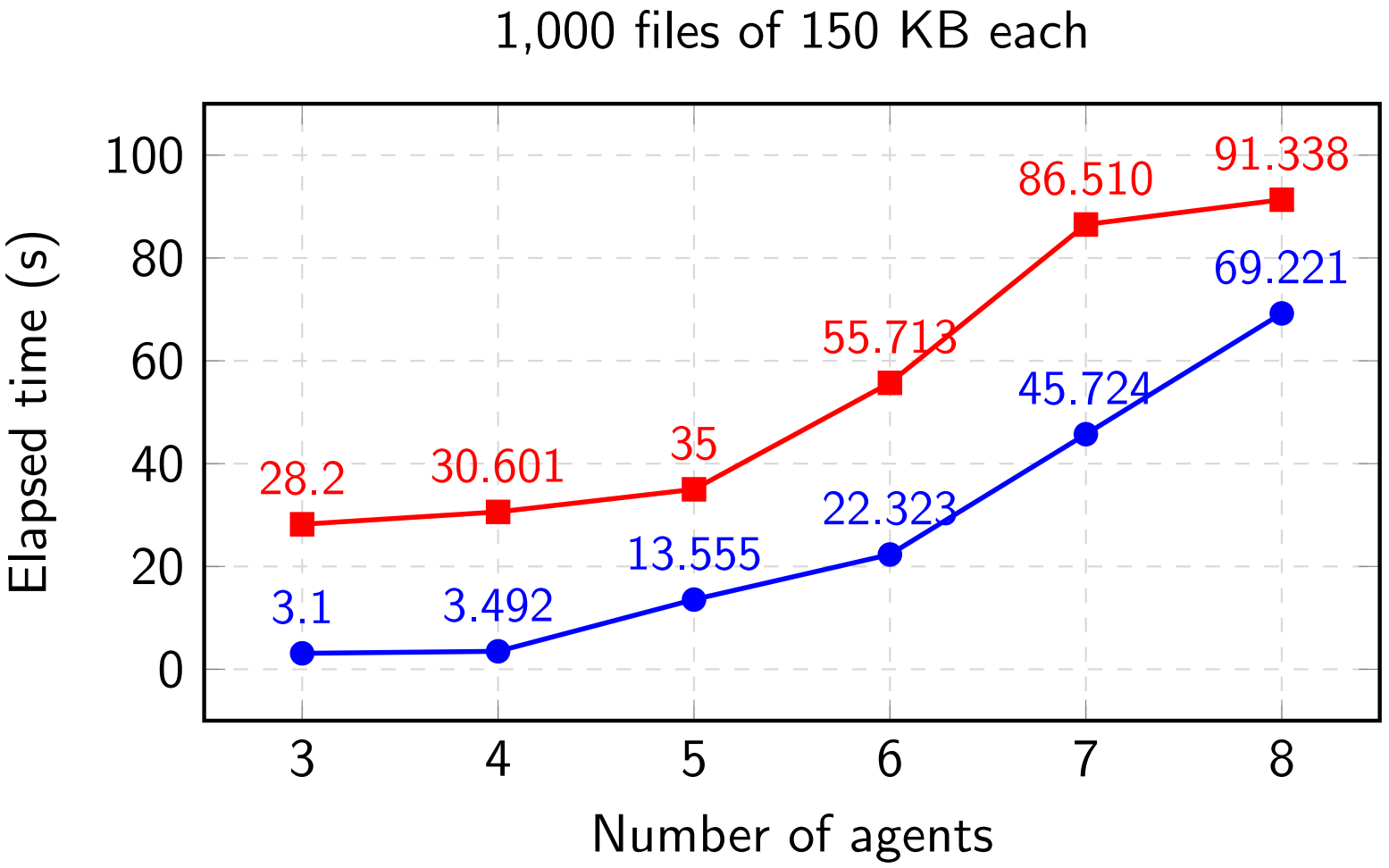
Same dataset but different file size and count – 2



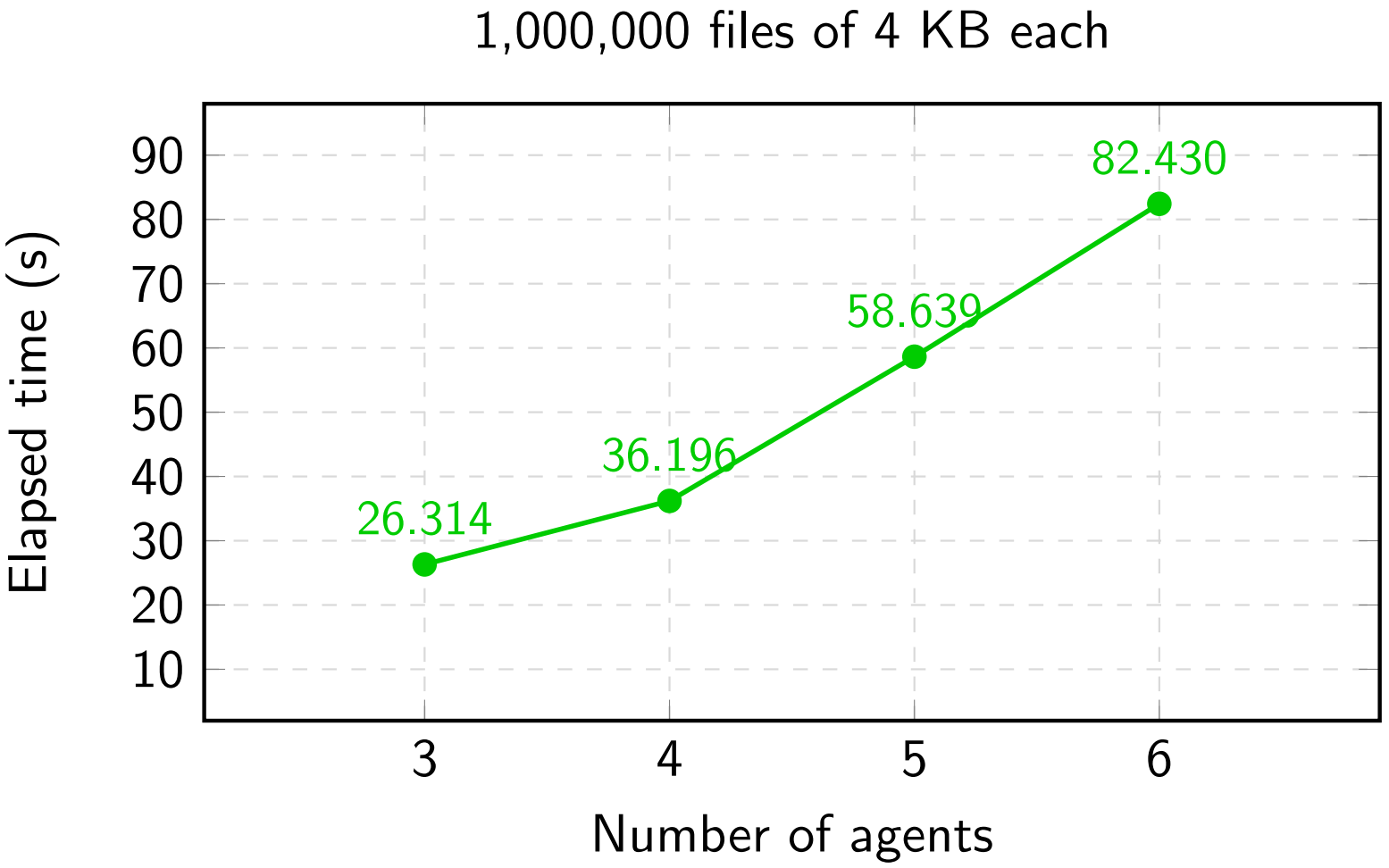
Same dataset but different file size and count – 3



Test with offline agents



Test with very large number of tiny files



Results

- Scenarios with many small files amplify the cost of synchronization and consensus.
- Verification time increases with the number of agents and files.
- Confirmed the correctness and resilience of the approach under different scenarios and network conditions.

Conclusion

- Raft ensures that integrity metadata is synchronized cluster-wide, while Merkle trees allow hierarchical and efficient integrity checks at the folder or subfolder level.

Conclusion

- Raft ensures that integrity metadata is synchronized cluster-wide, while Merkle trees allow hierarchical and efficient integrity checks at the folder or subfolder level.
- It is possible to maintain a consistent and verifiable integrity state across distributed agents without relying on full file scans or constant node availability.

Thank you for the attention.