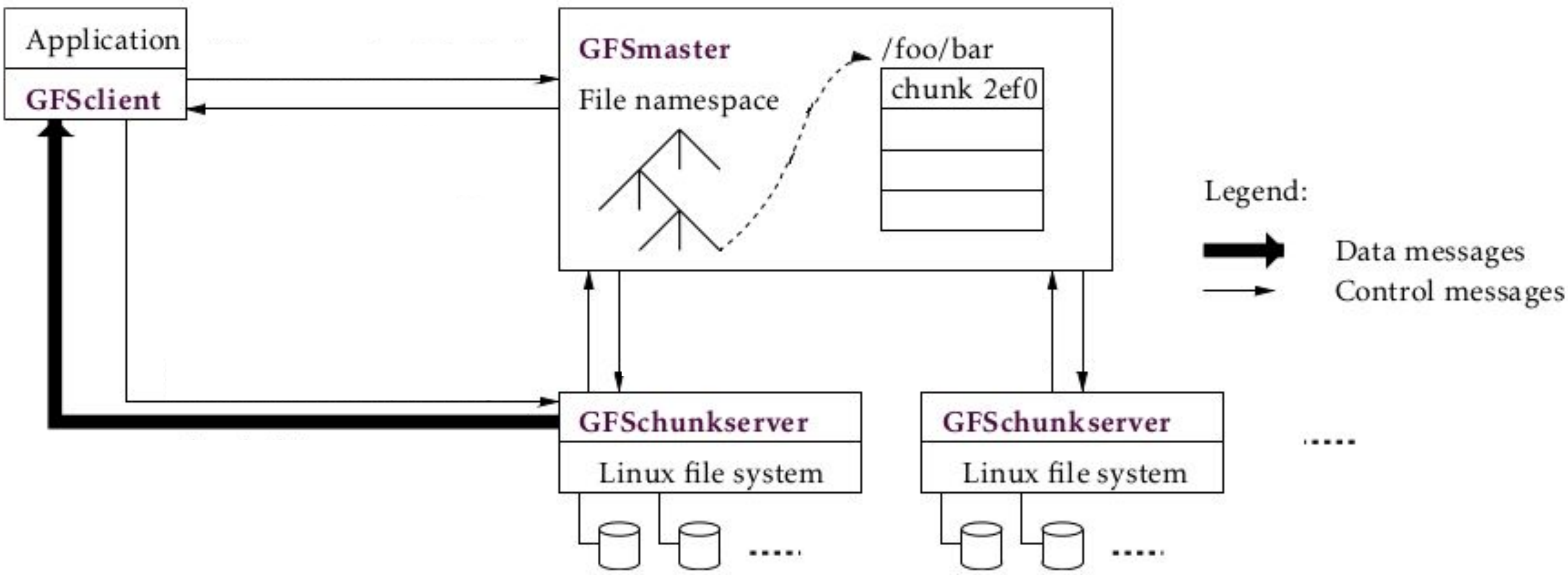
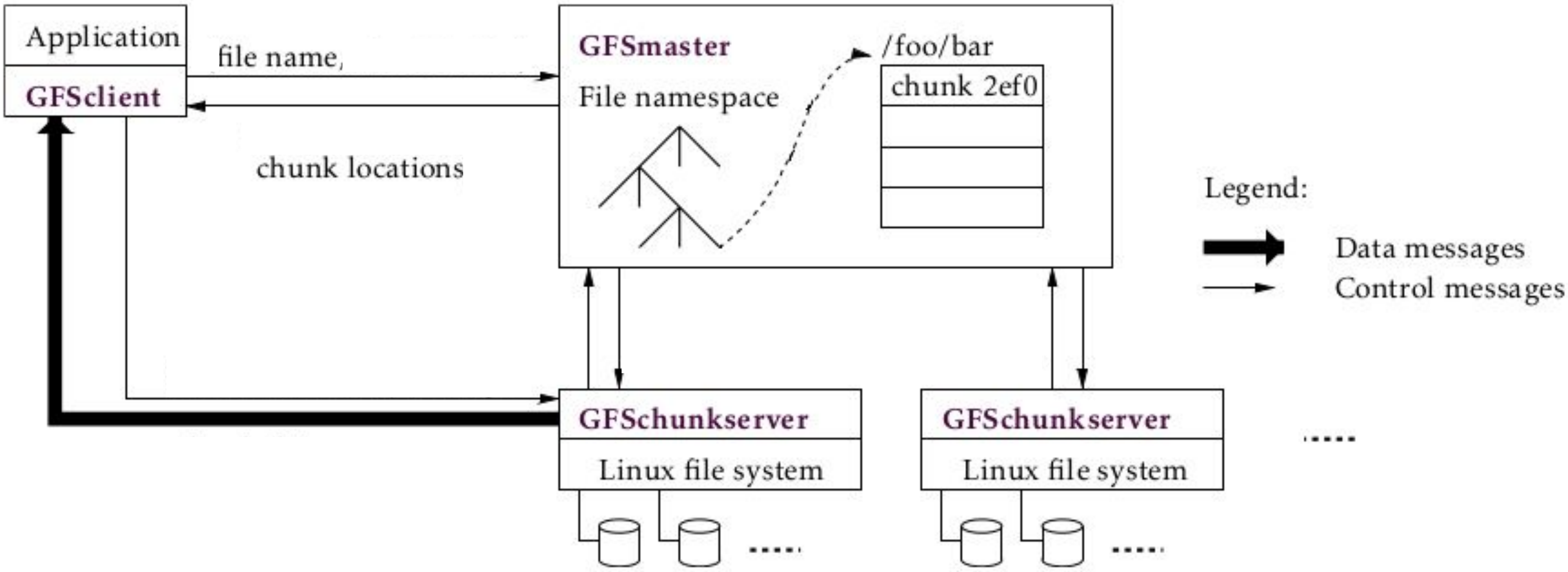
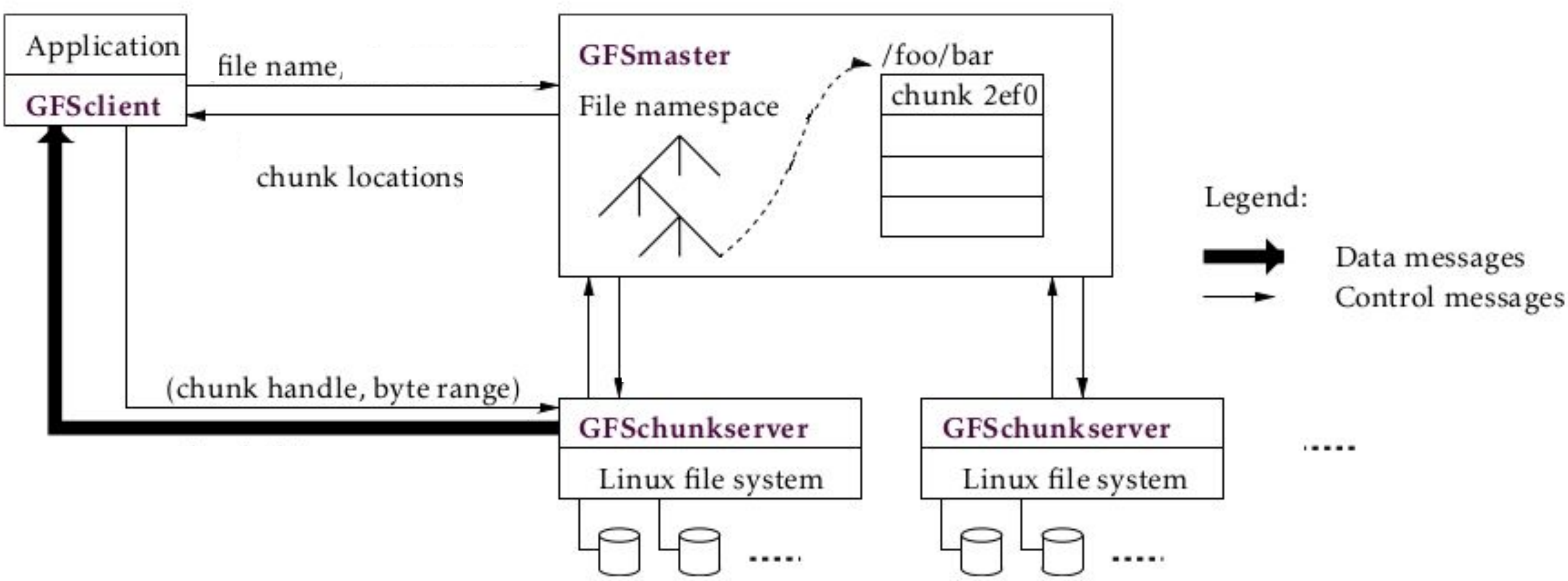
GovFS

a scalable control plane using groups of metadata nodes

By David Kleingeld
Supervised by Alexandru Uta and Kristian Rietveld


Application
GFSclient
GFSmaster
File namespace
/foo/bar
chunk 2ef0
Legend:
Data messages
Control messages
GFSchunkserver
Linux file system
GFSchunkserver
Linux file system


Application
GFSclient
file name,
chunk locations
GFSmaster
File namespace
/foo/bar
chunk 2ef0
Legend:
Data messages
Control messages
GFSchunkserver
Linux file system
GFSchunkserver
Linux file system

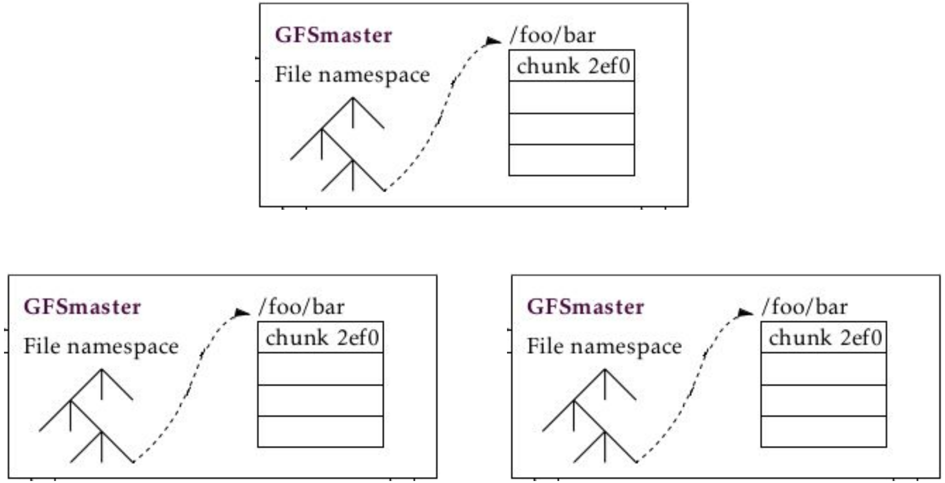

Application
GFSclient
file name,
chunk locations
GFSmaster
File namespace
/foo/bar
chunk 2ef0
Legend:
Data messages
Control messages
(chunk handle, byte range)
GFSchunkserver
Linux file system
GFSchunkserver
Linux file system

Distributed master

Nodes need to agree on each decision

How do we replace a node that goes down?

What if it comes up again?



Consensus

The truth is defined by the majority

Vote over every decision

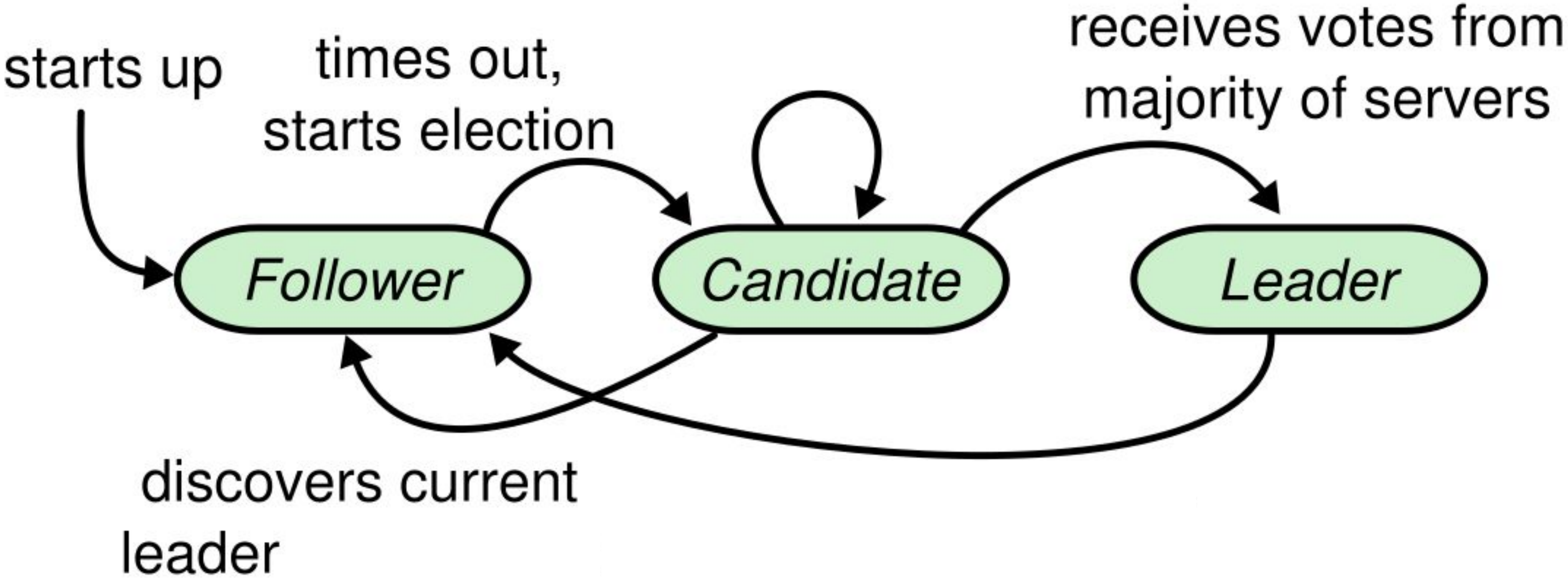
Consensus

The truth is defined by the majority

Vote over every decision

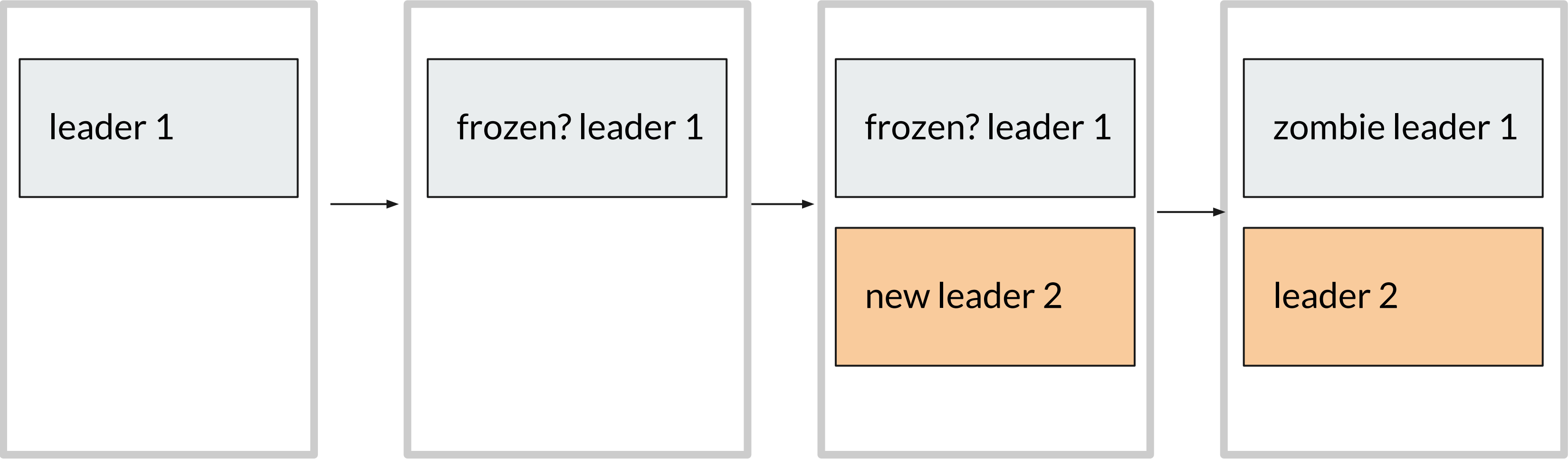
If we contact every node for each decision why have multiple nodes?

=> elect a **leader**, let the leader decide the rest


starts up
times out,
starts election
receives votes from
majority of servers
Follower
Candidate
Leader
discovers current
leader

Sharing the leaders decisions

Problem 1: zombie leaders



Sharing the leaders decisions

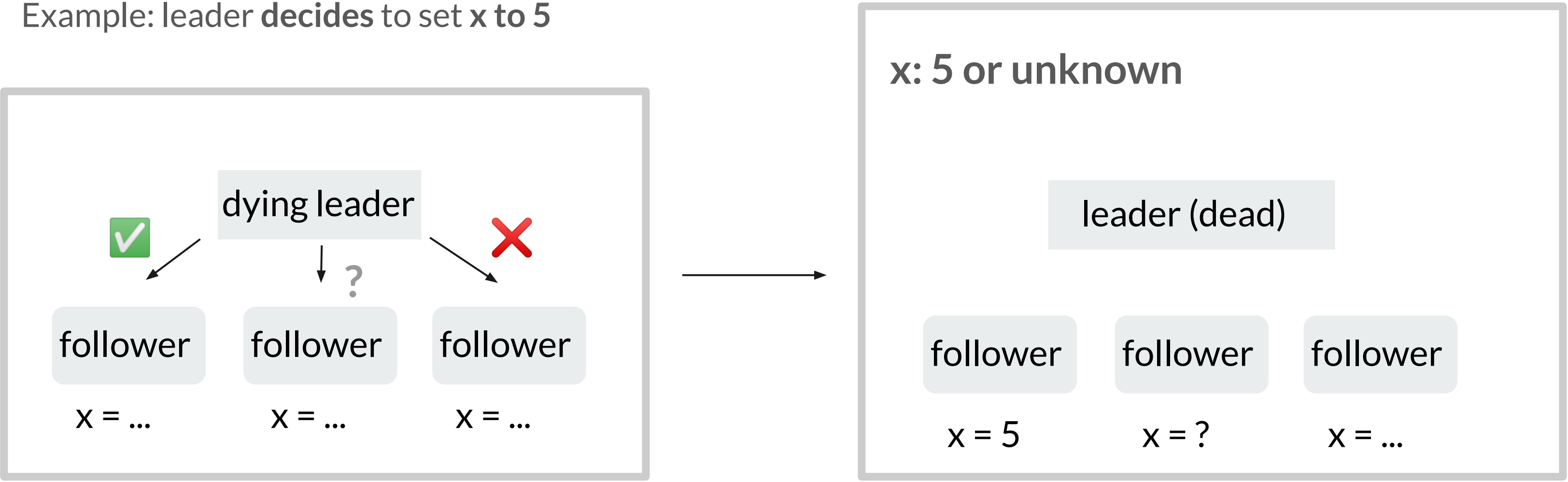
Problem 1: zombie leaders

Solution: mark each **decision** with its **leaders term**

Sharing the leaders decisions

Problem 2: leader fails while sharing

Example: leader **decides** to set **x to 5**



dying leader

✅ ? ❌

follower follower follower

x = ... x = ... x = ...

x: 5 or unknown

leader (dead)

follower follower follower

x = 5 x = ? x = ...

Sharing the leaders decisions

Problem 2: leader fails while sharing

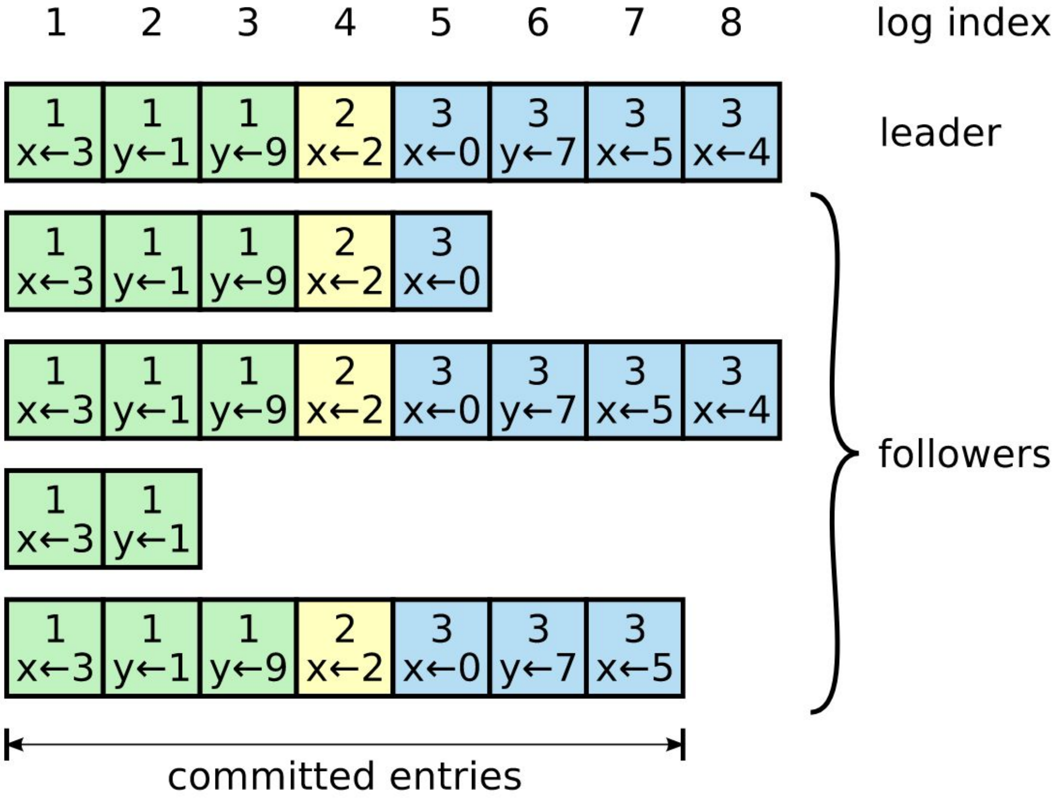
Solution: assign each decision a **number**. The number **increases** with each decision and is therefore **unique**

The leader informs the followers of the **highest number replicated** to a majority of followers

Sharing the leaders decisions

Problem 2: leader fails while sharing

Solution: assign each decision a **number**. The number **increases** with each decision and is therefore **unique**



log index	1	2	3	4	5	6	7	8
leader	1 x←3	1 y←1	1 y←9	2 x←2	3 x←0	3 y←7	3 x←5	3 x←4
followers	1 x←3	1 y←1	1 y←9	2 x←2	3 x←0			
	1 x←3	1 y←1	1 y←9	2 x←2	3 x←0	3 y←7	3 x←5	3 x←4
	1 x←3	1 y←1						
	1 x←3	1 y←1	1 y←9	2 x←2	3 x←0	3 y←7	3 x←5	

Group Raft

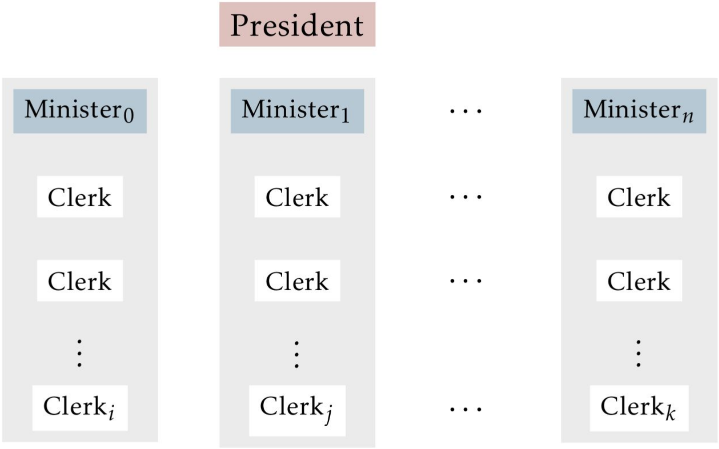
Multiple **groups**
(ministries)

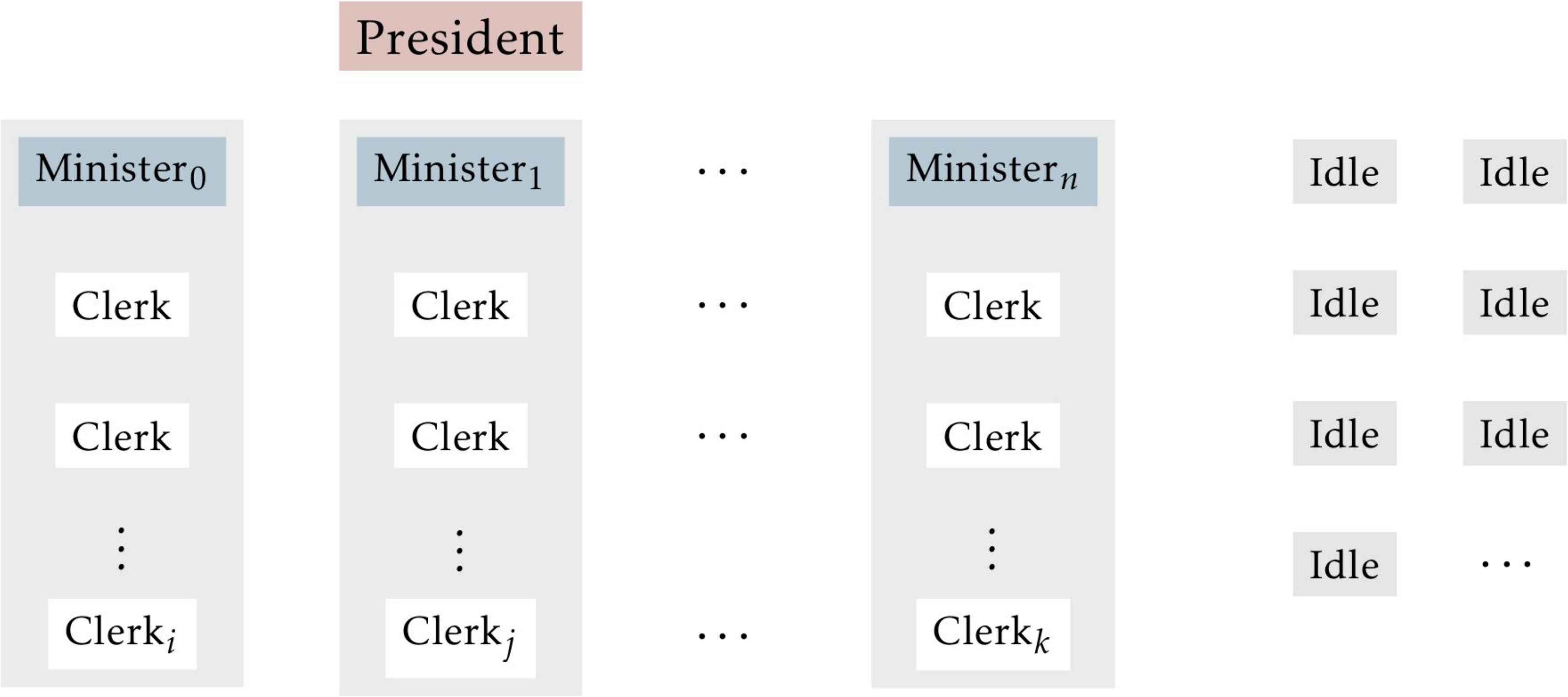
Cluster **elects** one leader
(president)

Elected leader **appoints** each group its own leader
(minister)

Elected leader **assigns** groups their **members**
(clerks)

Appointed leader has a **unique term**





President

Minister_0 Minister_1 $\cdots$ Minister_n Idle Idle

Clerk Clerk $\cdots$ Clerk Idle Idle

Clerk Clerk $\cdots$ Clerk Idle Idle

$\vdots$ $\vdots$ $\vdots$ Idle $\cdots$

Clerk_i Clerk_j $\cdots$ Clerk_k

File leases

Clients can only operate on files with a lease

A lease must be **maintained**

File leases

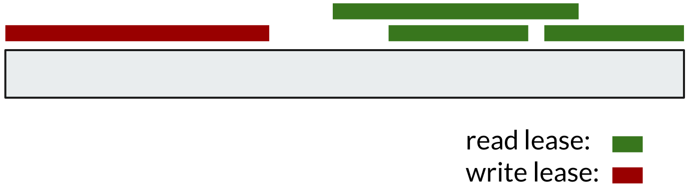
Clients can only operate on files with a lease

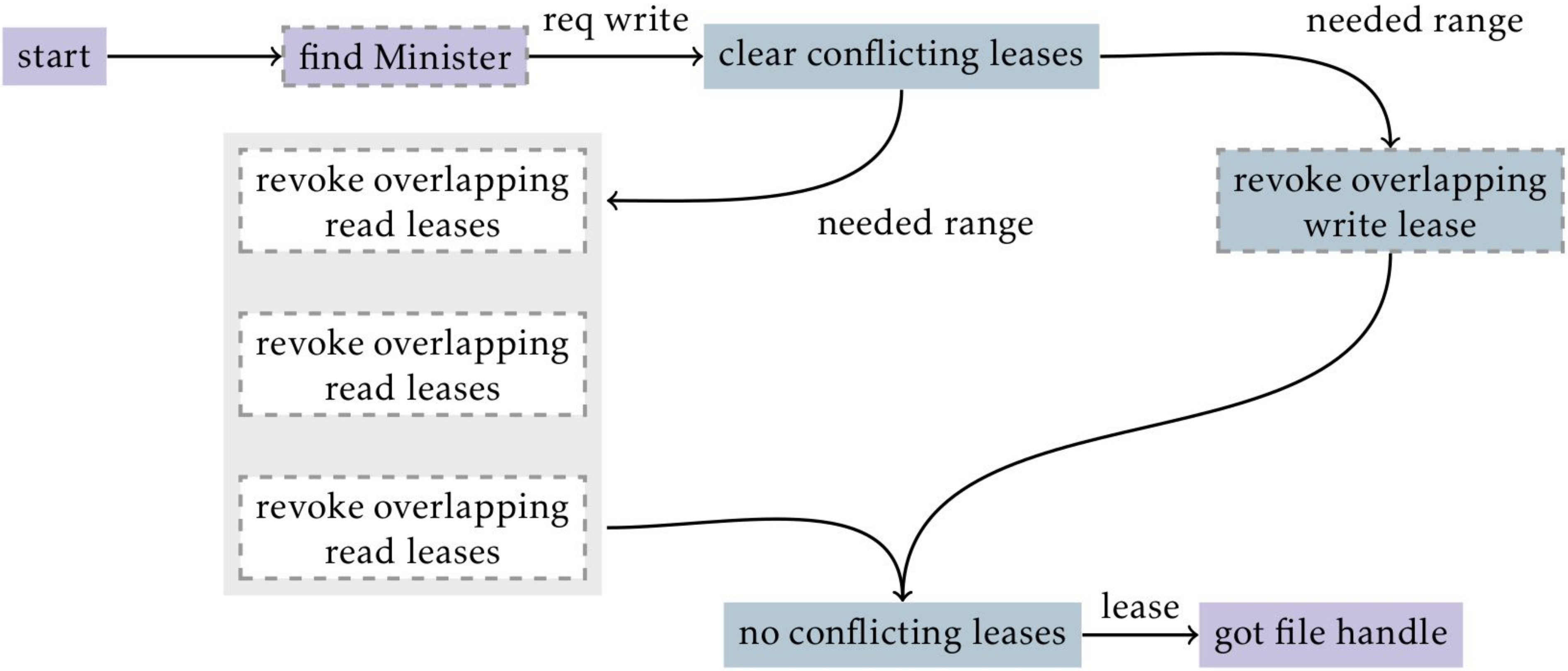
A lease must be **maintained**

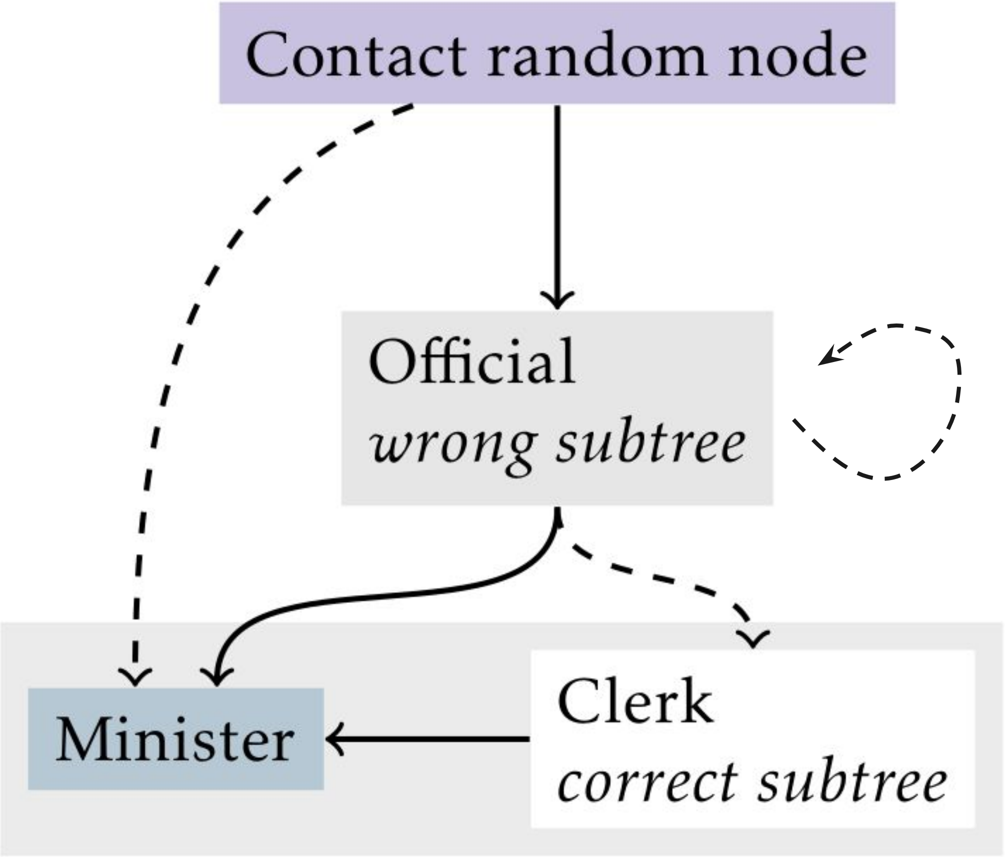
Ministers give out **write leases**

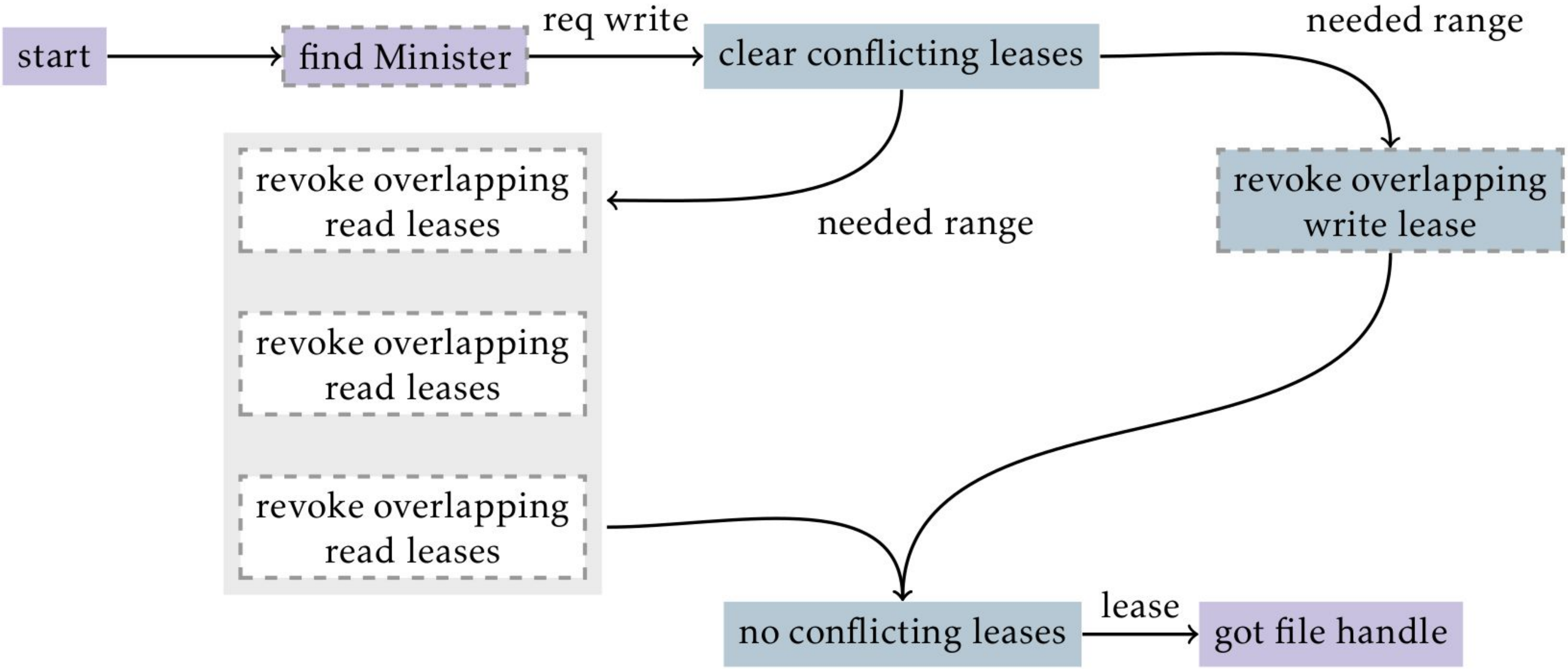
Clerks give out **read leases**

Read leases may **overlap** Read leases:




start
find Minister
req write
clear conflicting leases
needed range
revoke overlapping read leases
needed range
revoke overlapping write lease
revoke overlapping read leases
revoke overlapping read leases
no conflicting leases
lease
got file handle

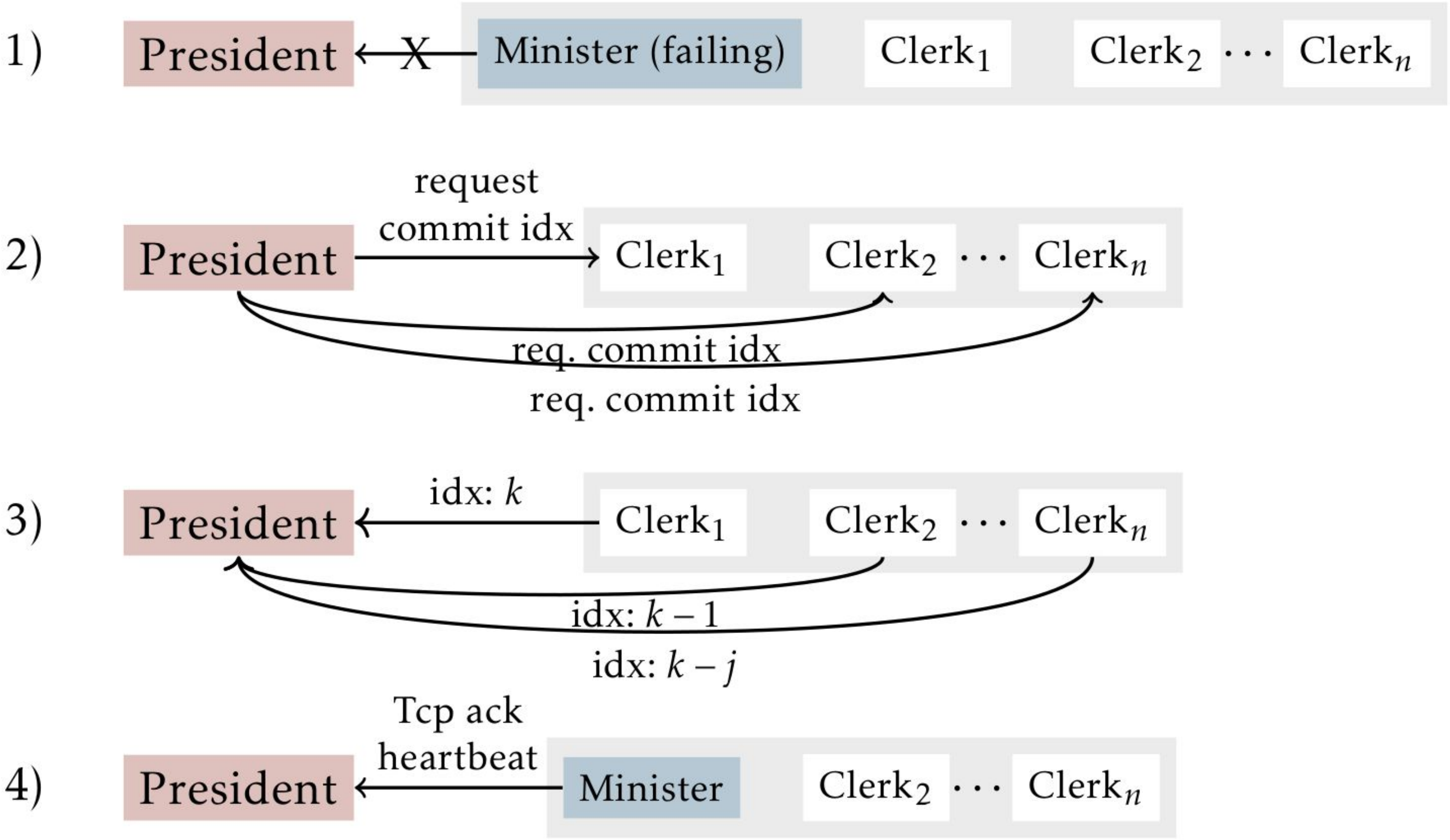

Contact random node
Official
wrong subtree
Minister
Clerk
correct subtree


start
find Minister
req write
clear conflicting leases
needed range
revoke overlapping read leases
revoke overlapping read leases
revoke overlapping read leases
needed range
revoke overlapping write lease
no conflicting leases
lease
got file handle

Minister failure

Lease are tracked **only** by the **node** that **issued** them

Valid for **less time** then it takes to **replace** the minister


1)
President
X
Minister (failing)
Clerk_1
Clerk_2 · · · Clerk_n
2)
President
request
commit idx
Clerk_1
Clerk_2 · · · Clerk_n
req. commit idx
req. commit idx
3)
President
idx: k
Clerk_1
Clerk_2 · · · Clerk_n
idx: $k - 1$
idx: $k - j$
4)
President
Tcp ack
heartbeat
Minister
Clerk_2 · · · Clerk_n

Load balancing

by the President

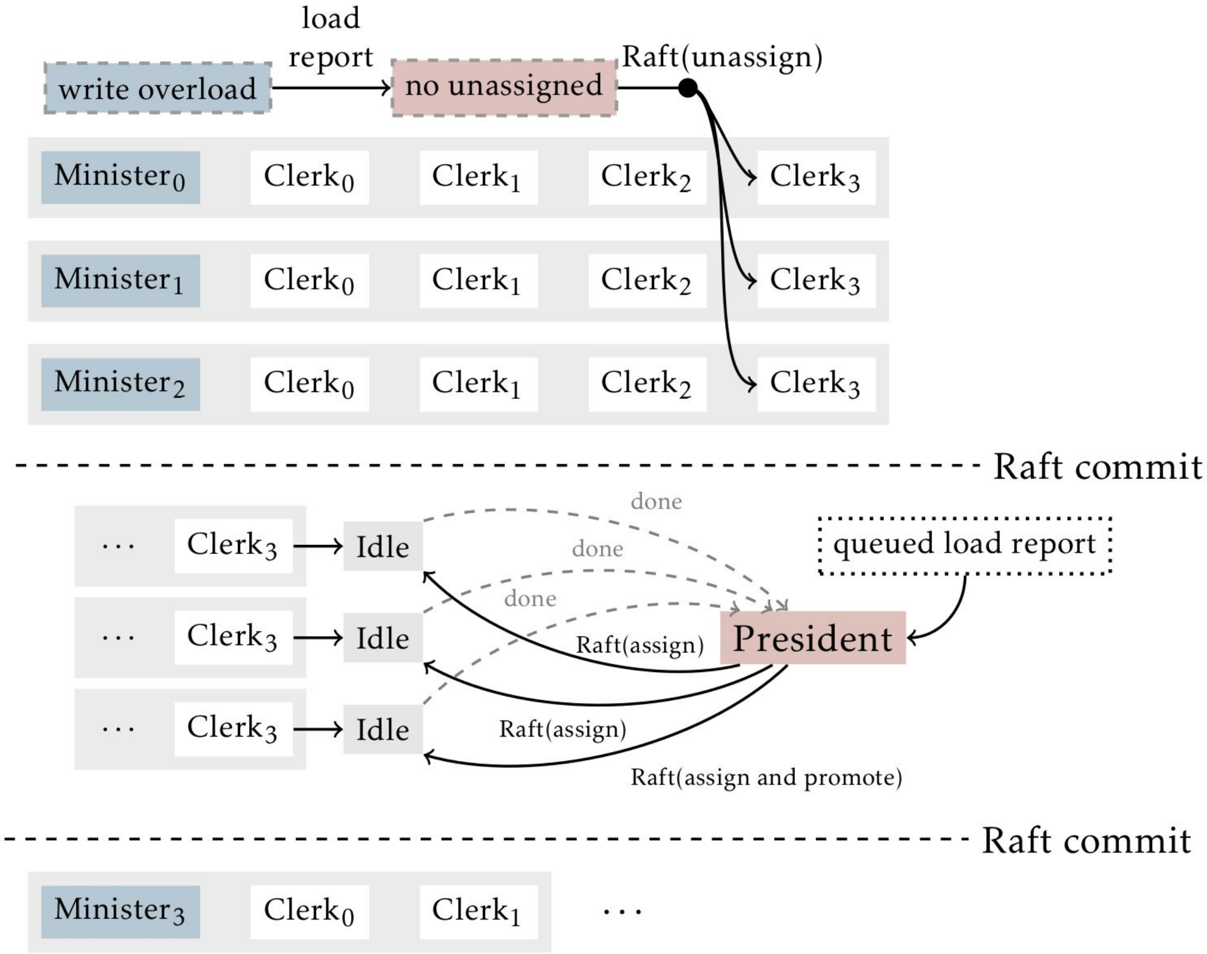
More or fewer ministries

Larger or smaller ministries

Load balancing

More or fewer ministries

Larger or smaller ministries



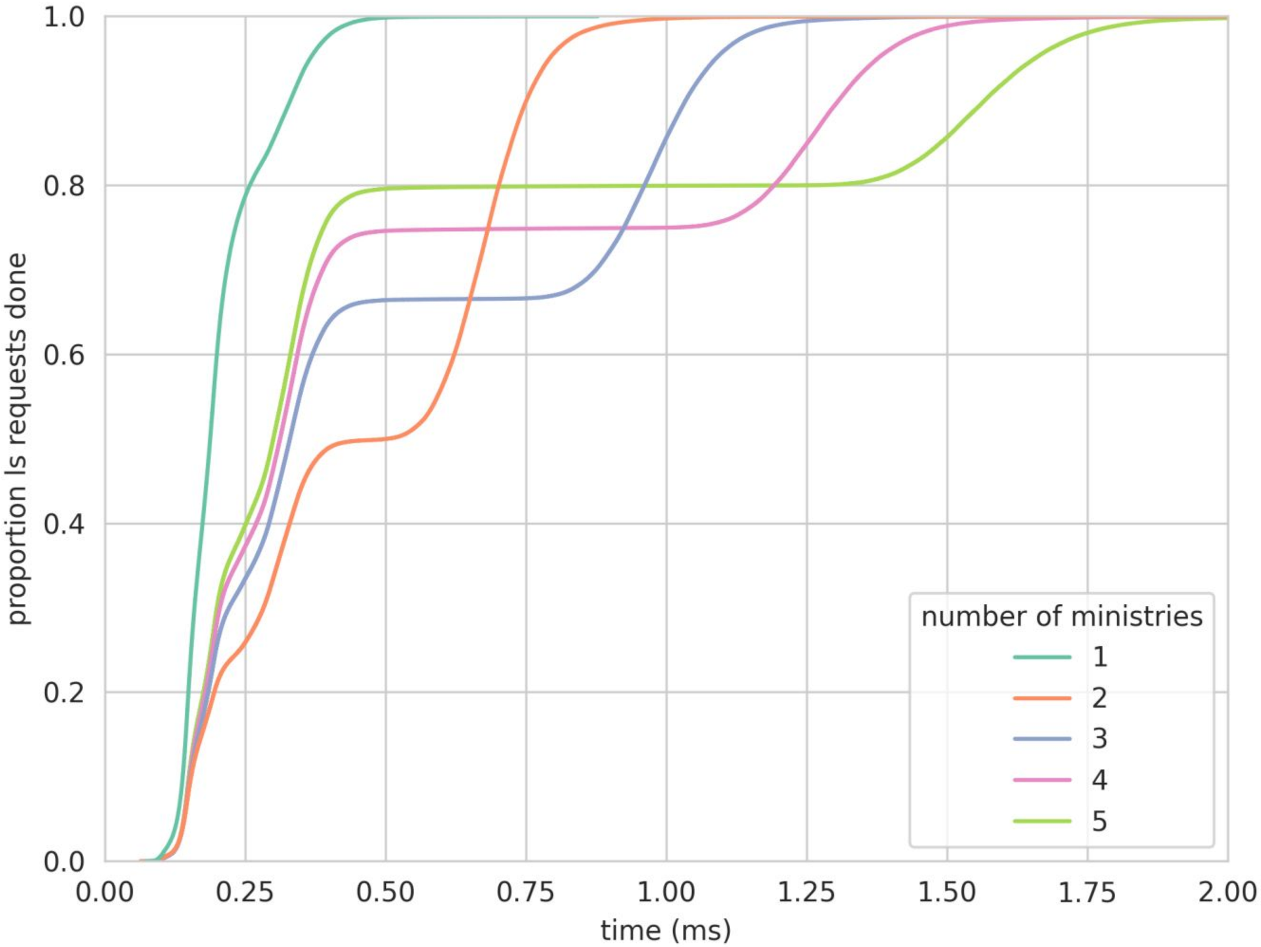
List 60k Directories

(using 30 clients)

Proportion completed

vs

time in milliseconds



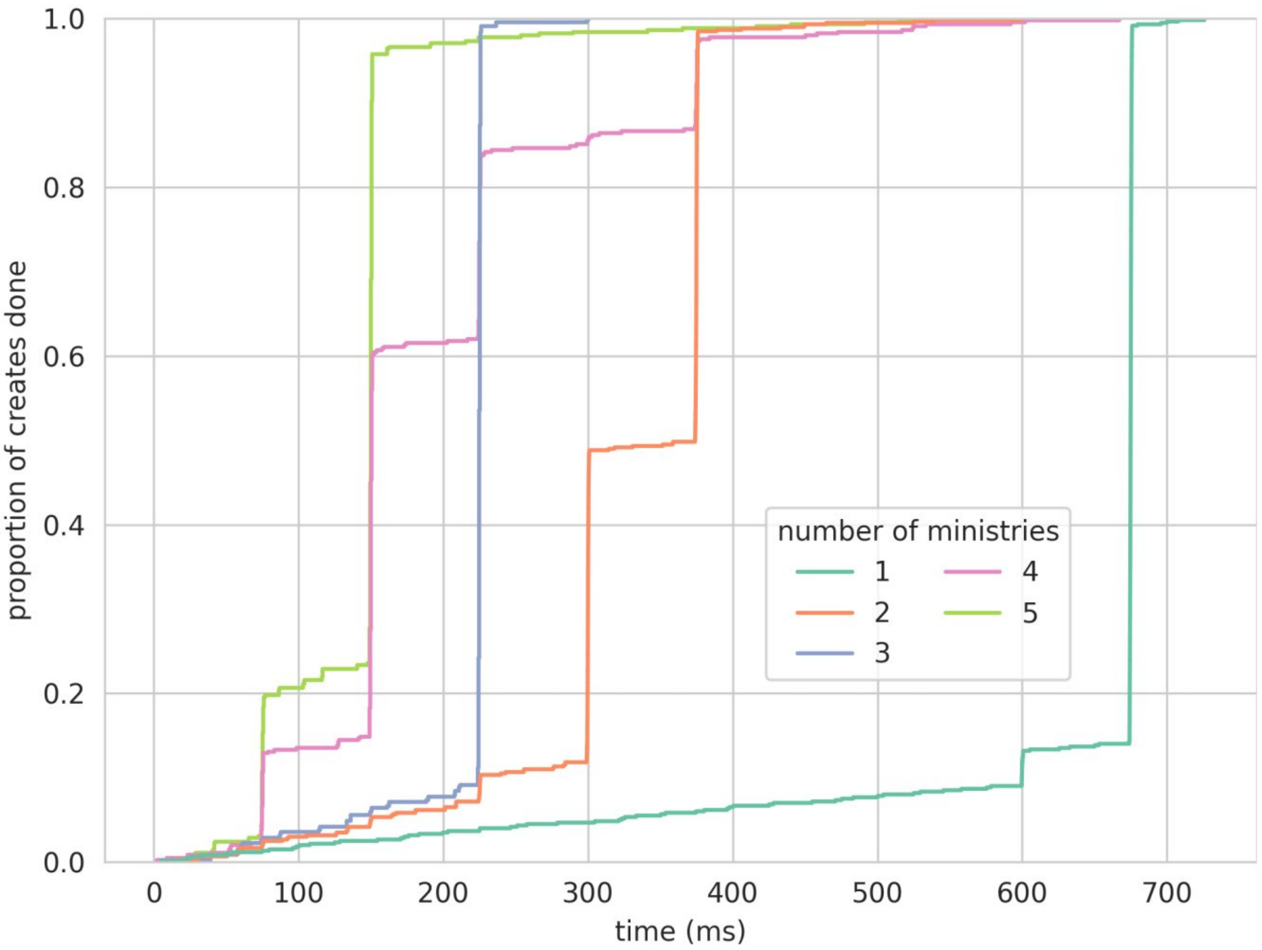
Create 90 files

(using 9 clients)

Proportion completed

vs

time in milliseconds



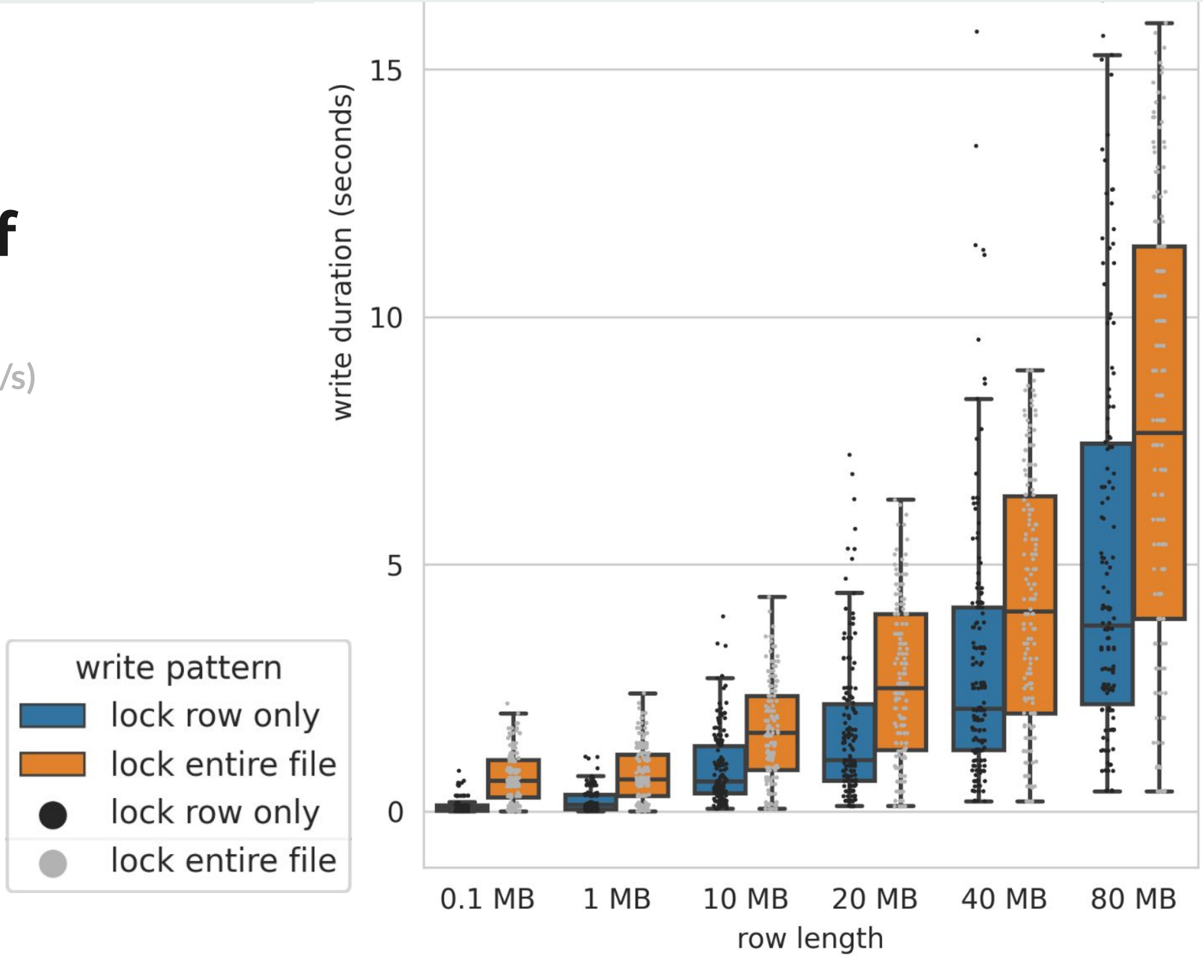
Write part of a file

(simulated io as 200MB/s)
(using 6 clients)

time in seconds

vs

row length in megabyte (MB)



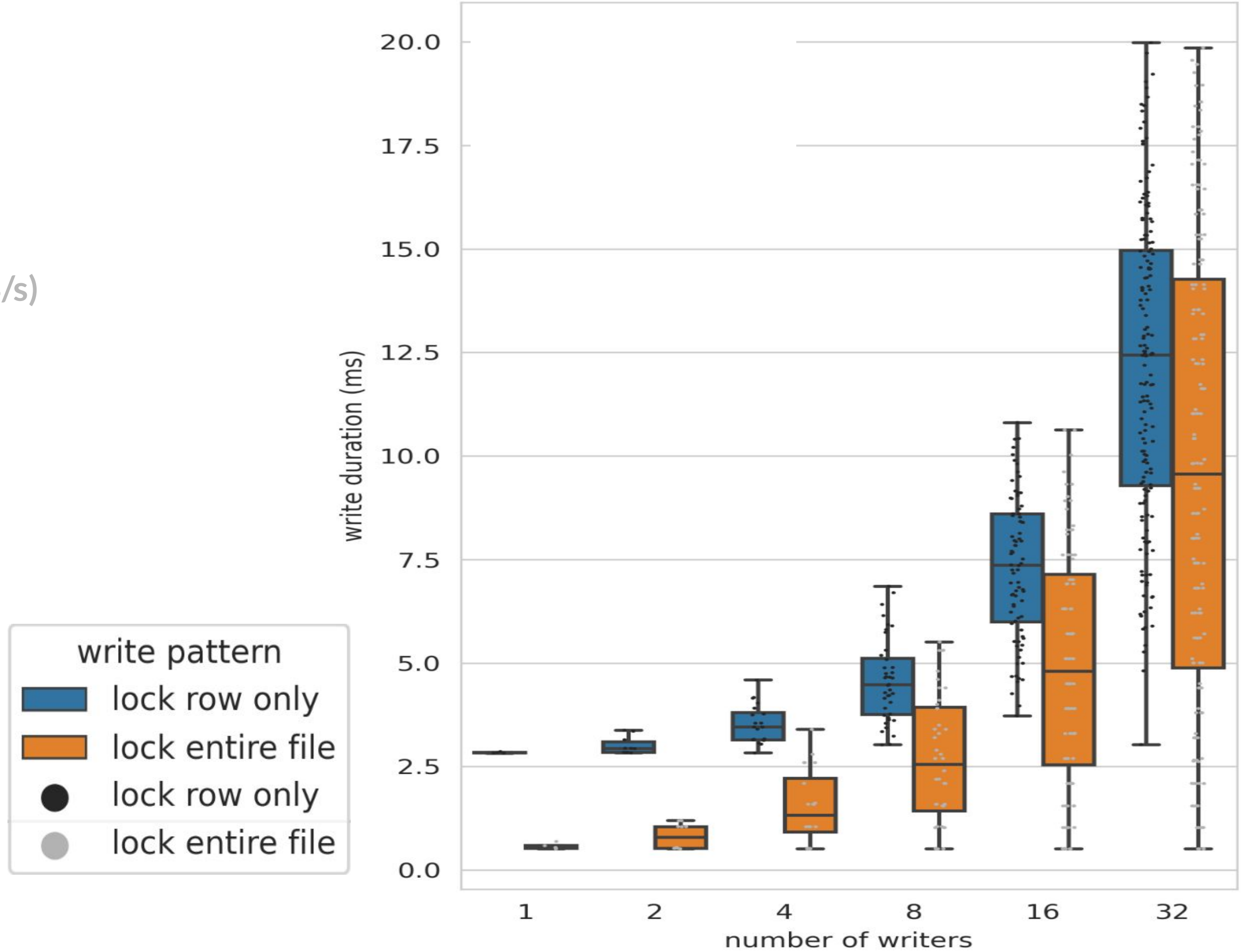
Write a file

(simulated io as 200MB/s)
(using 6 clients)

time in seconds

vs

row length in megabyte (MB)



Conclusion

Ranged locking useful addition

Linear scaling when creating files

More implementation effort needed