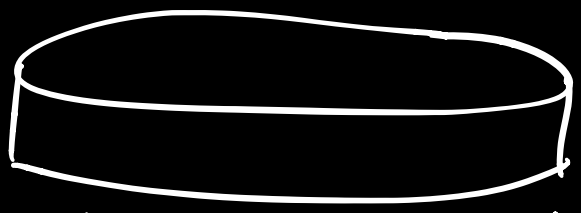
Petal C_1 ... C_n

disk =>



more b/w, more reliability

limits

=> scale => reconfig

=> usability => block-based

File System : Frangipani

Layering nets

HTTP
TCP
IP
ether

does work for storage? (benefits? costs?)

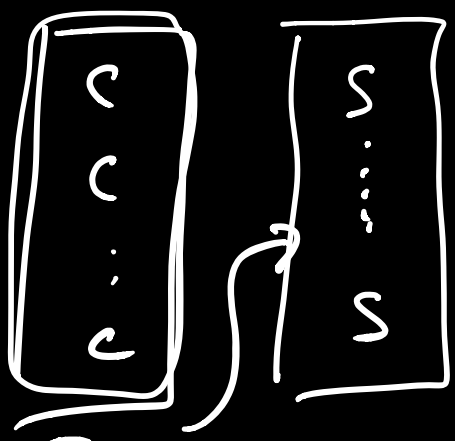
Frangipani

"cluster" FS

classic

NFS / AFS

Cluster



shared FS

client / Frang. "Servers" → Lock service

client-side: memory (cache)
(consistency)

crash recovery: logging (WAL)

[how to build a dist FS on a shared virt disk?]

C
Local FS ← Linux ext4
⇒ ↓
Petal

works ⇒
not a dist fs

Layout [on-disk Data Structures + Access Methods]

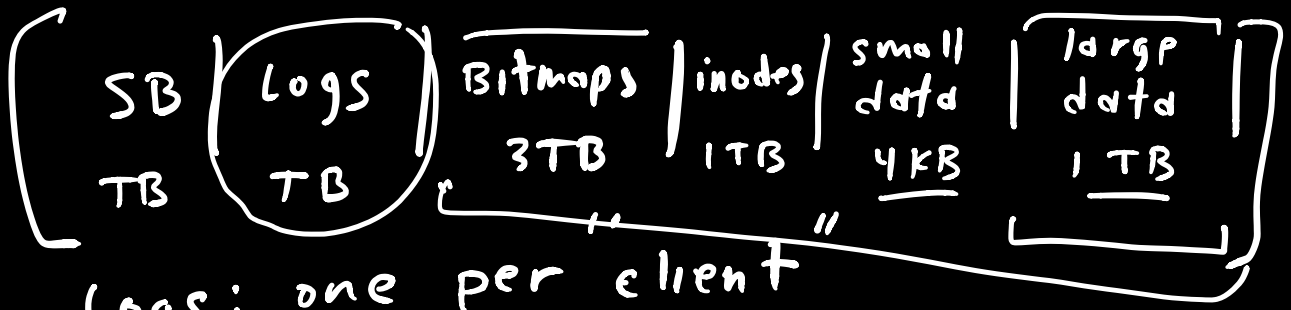
Superblock: config info

1 TB alloc, only 3 KB used

tennis court → 32-bit

64-bit Europe

key aspect of Petal?
sparse, lazily allocated
alloc: 64 KB



Logs: one per client

=) crash recovery =) many logs

=) log is in Petal (not local to clients)

(others can access, useful for recovery?)

file: up to 16 4KB blocks (small) + one additional 1 TB block

Logging / Recovery:

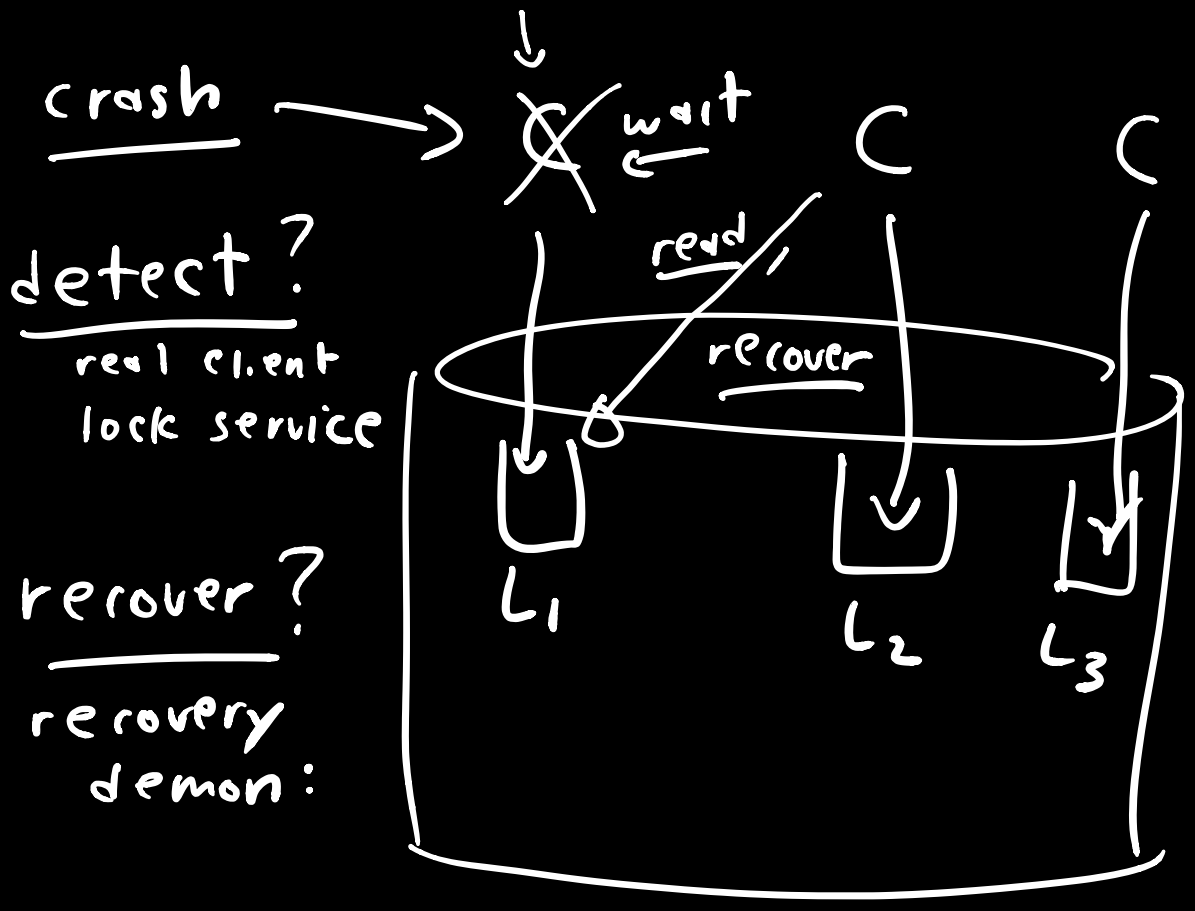
=) each client -) own log (exclusive)

=) metadata only WAL

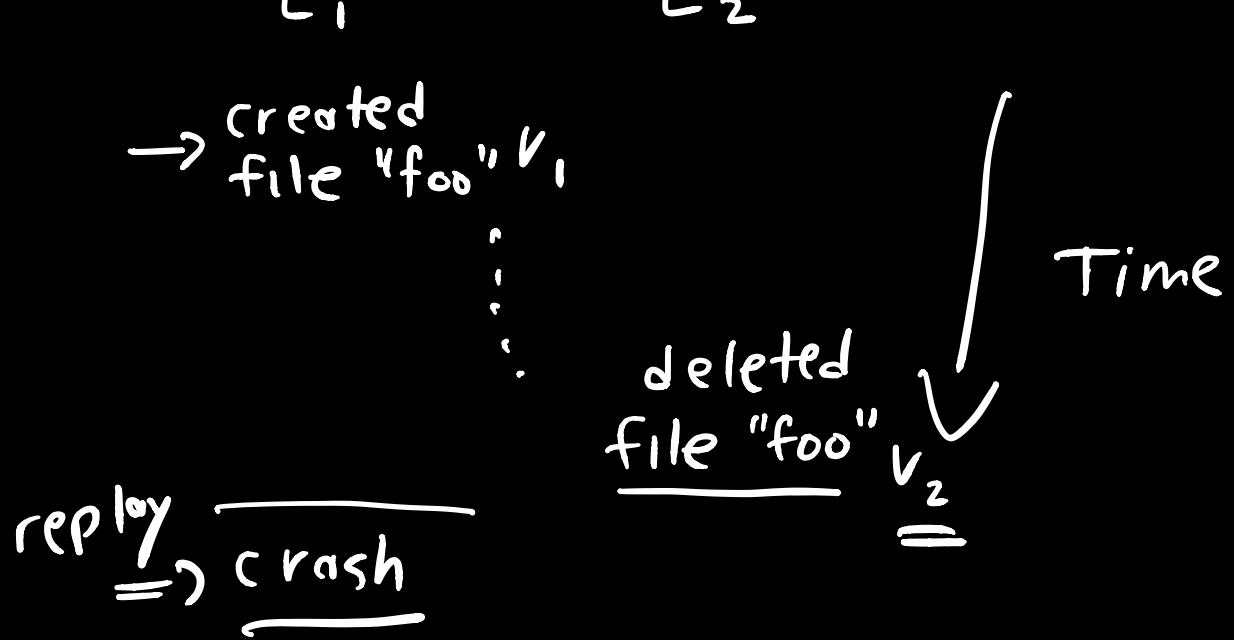
writes: => buffer updates in client mem

=> fills: write in order to Petal Log

=> after: update metadata in place



Tricky: serialization of updates



replay ⇒ crash

Solution: version #s

⇒ in log

⇒ w/ metadata in FS

one other problem:
{ have to be careful when reusing metadata }

Monday: send email

Exam solution: email

Decide on last 2 weeks

(=) ? until weekend

4th ... 11th x □

Final? -> Yes / No ? ?

Synch + Cache Coherence

multi-reader, single writer locks

read: read lock

allows: cache block in mem, read

invalidate: => flush cache

write : write lock
allows: cache, buffer
release: → downgrade to read

→ lose all together
=> flush dirty => Retain

Workload patterns?
(supported)
=> may be read sharing, not much write sharing

Granularity: logical "segments"
1 lock/segment

=> log
=> bitmap split into N segments
each client → one

=) each file/dir
(inode + data blocks)
one lock each

Problem : Deadlock

operations that acquire 71 lock

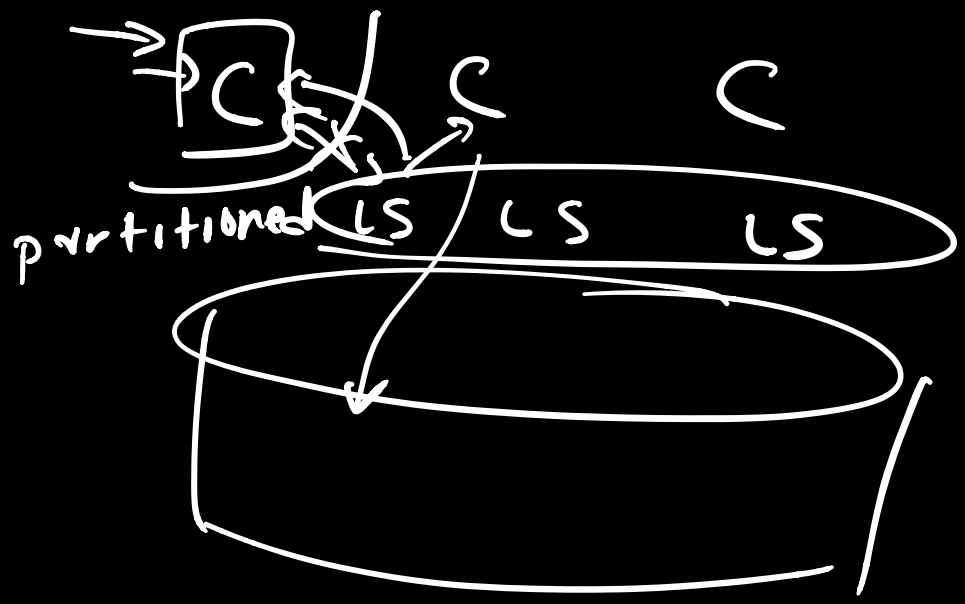
=) figure out which locks needed

=) acquire in inode address order

Lock Service : (Leases)

each client → one lease (all locks)

expires after 30 secs



Many implementations of lock service

⇒ distributed, replicated (Paxos)

Lease expiration:

⇒ C_1 has lease (but slow)

LS → C_1 is dead

writes: (and will be for x secs)

check lease is still valid

do write

missing : wnte fence in Petal

Performance

✓

Layering

=> Simpler to build

=> Costs

=> wnte fence

=> read ahead

Storage

where layering is costly

(= [=> double logging]