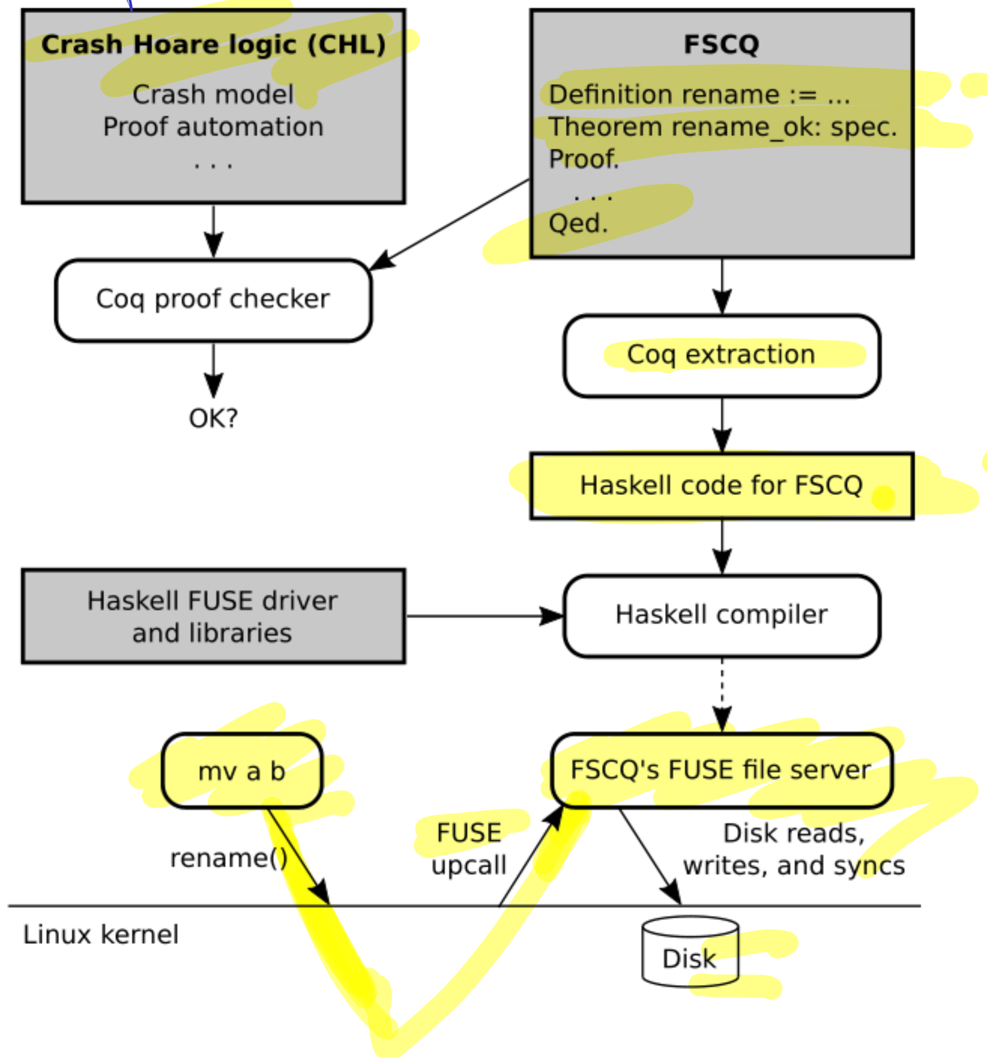
FSCQ

- Sys. verification
  - Importent code.
  - Stable specs.
  - Bugs
- Crashes
  - Logging
  - Lab 3, 4.
- Non-determinism
  - Async disk

- Academic papers
  - Prototype: show idea.

exec
specs . . .



Crash Hoare logic (CHL)
Crash model
Proof automation
. . .
FSCQ
Definition rename := ...
Theorem rename_ok: spec.
Proof.
. . .
Qed.
Coq proof checker
OK?
Coq extraction
Haskell code for FSCQ
Haskell FUSE driver and libraries
Haskell compiler
mv a b
rename()
FUSE upcall
FSCQ's FUSE file server
Disk reads, writes, and syncs
Linux kernel
Disk

Goal: crash safety

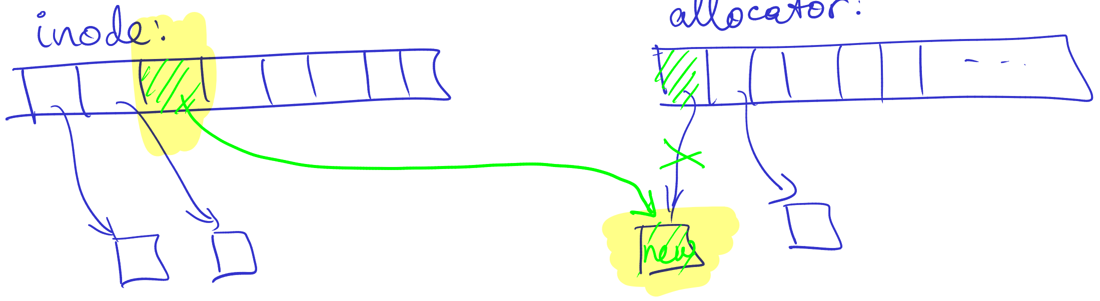
OS boots ✓

FS intact ✓

User data present

Challenge: crash any time

append to file: write(f, new data)



Logging

Never update in-place

Write to log, apply log.

Recovery: apply log.

Verify logging

Spec?

Model?

- Disk
- Crash
- Recover

Proof?

- No crash: func. correctness
  SibylFS.
- Crashes: ...

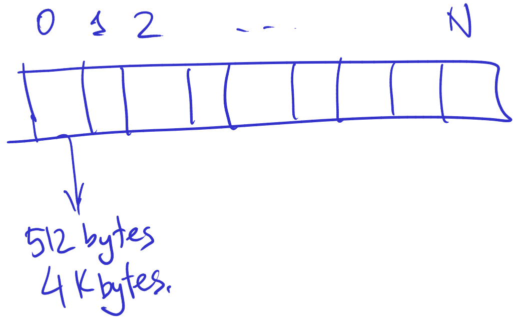
Spec: transactions

Atomicity: all-or-none updates.

Durability: persist after crash.

Concurrency: sequential.

Disk model



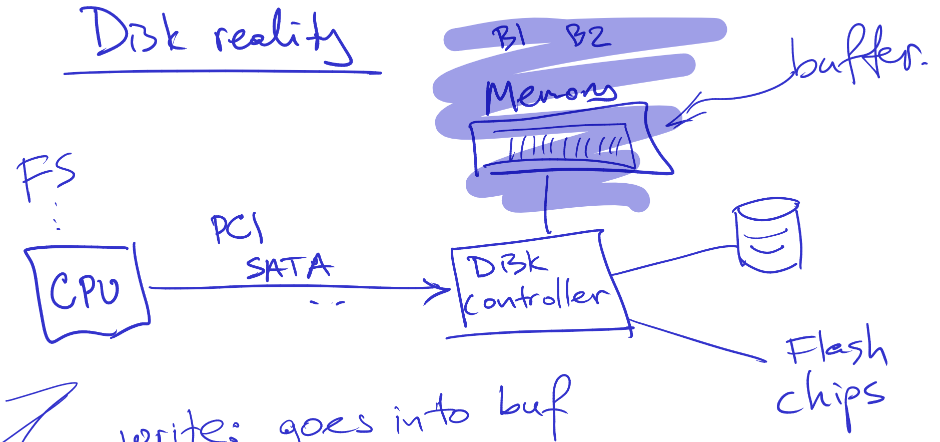
labs [read(a)
write(a,v)]

sync()

$\{a \mapsto v_0\}$ write(a,v) $\{a \mapsto v\}$

SYNC
DISK.

Disk reality



write: goes into buf
read: buf, or durable
sync: flush buf to durable
crash: lose mem / buf.
background: buf → durable, without sync.

Q: FSCQ model?

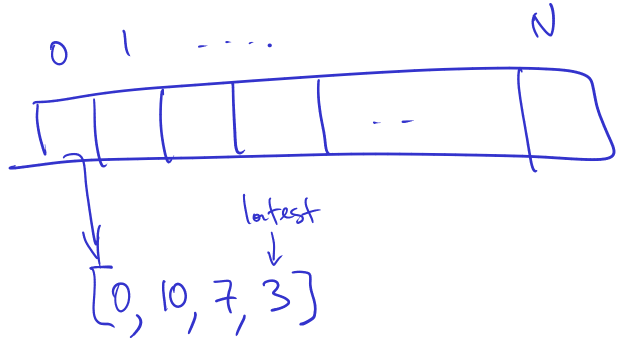
write(a, B1)

write (a, B2)

crash

read (a) → B0
B2
B1

Async disk model



$\{ a \mapsto [0] \}$ $\longrightarrow$ $a \mapsto 0.$

write(a, 10);

$\{ a \mapsto [0, 10] \}$

read(a);

$\{ r = \text{last}([0,10]) \ast a \mapsto [0,10] \}$

write(a, 11);

$\{ a \mapsto [0, 10, 11] \}$ $\xrightarrow{\text{crash?}}$ $a \mapsto [0]$, $a \mapsto [10]$, $a \mapsto [11]$

sync()

$\{ a \mapsto [11] \}$ $\xrightarrow{\text{crash}}$ $a \mapsto [11]$

Crashes in code

1. Where could we crash?
2. How to recovery?

1 = crash condition.

$\{ a \mapsto [0] \}$

write(a, 10)

$\{ a \mapsto [0, 10] \}$

CRASH

$\{ a \mapsto [0] \lor \underline{a \mapsto [0, 10]} \}$

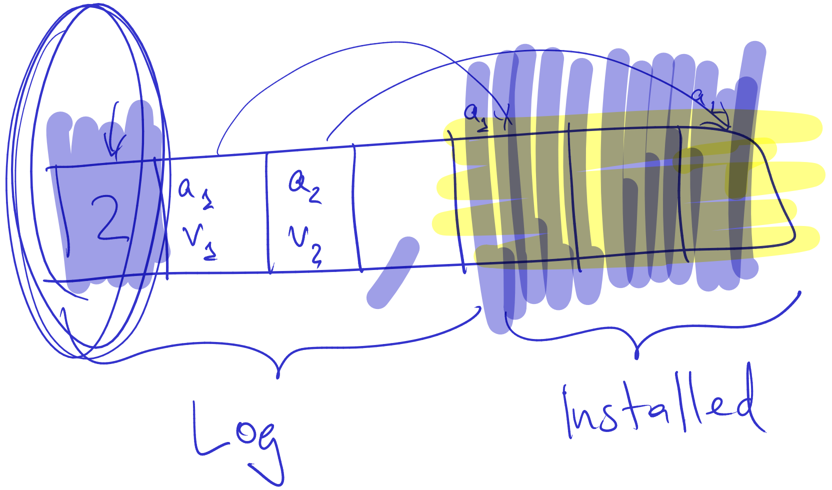
Atomic sector write.

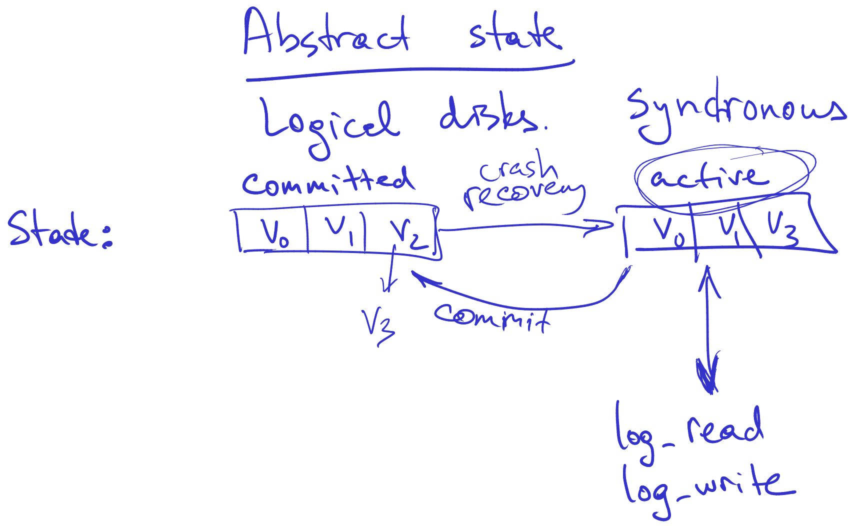
2 = recovery.

reboot: choose one block val for each addr.

Logging spec

```
def atomic_two_write(a1, v1, a2, v2):
    log_begin()
    log_write(a1, v1)
    log_write(a2, v2)
    log_commit()
```





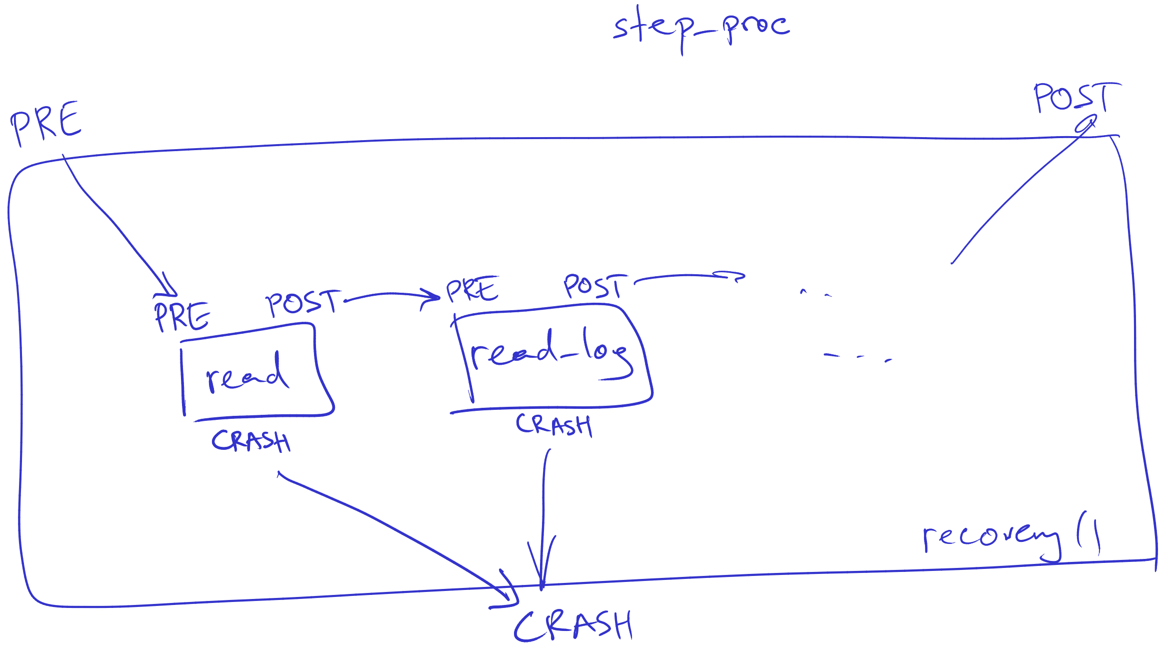
Type of state

Abs. state

$$\begin{aligned} &\texttt{log\_rep}(\text{ActiveTxn}, \mathit{start\_state}, \mathit{cur\_state}) := \\ &\quad \text{COMMITBLOCK} \mapsto \langle 0, \varnothing \rangle \star (\forall a,\ \mathit{start\_state}[a] = v \ \rightarrow\ a \mapsto \langle v, \varnothing \rangle) \\ &\quad \land\ \texttt{replay}(\mathit{start\_state}, \texttt{inMemoryLog}) = \mathit{cur\_state} \end{aligned}$$

full disk (async)

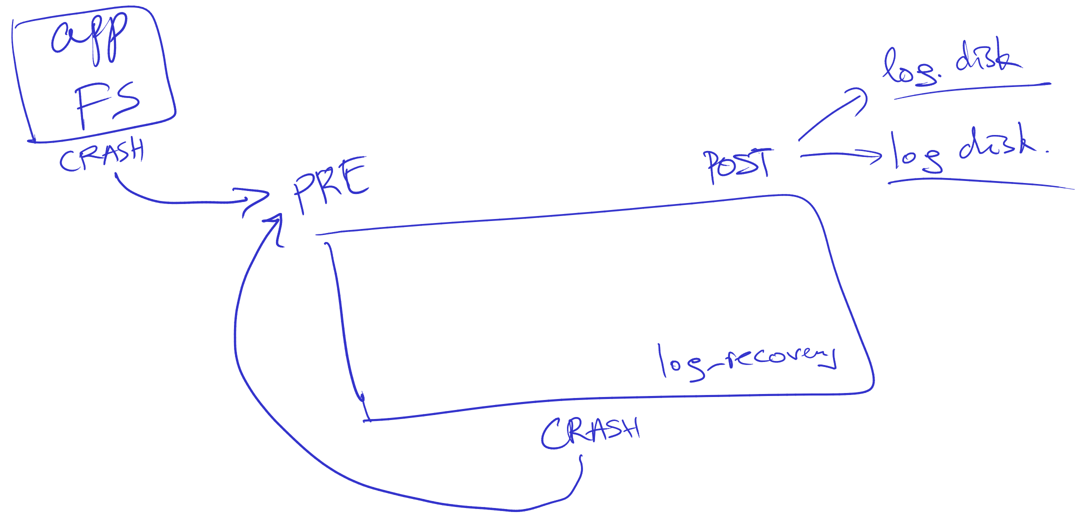
```
def recovery():
  r = read(LOG_HDR)
  if r != 0:
    l = read_log()
    for a, v in l:
      write(a, v)
    sync()
    write(LOG_HDR, 0)
```



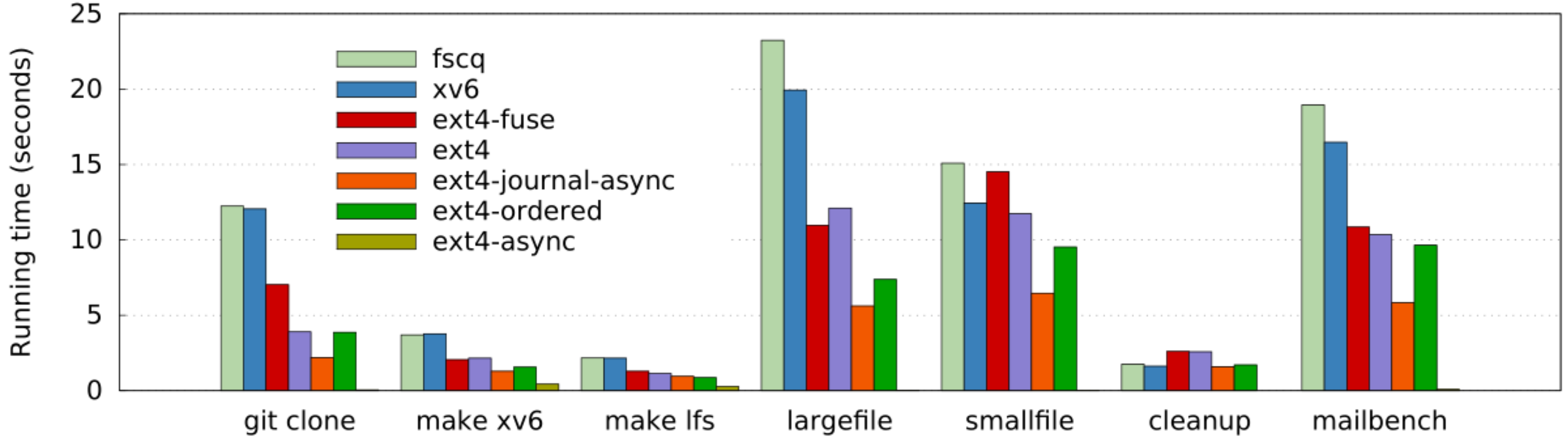
CHL

Idempotence

SPEC	log_recover()
PRE	**disk**: log_intact(*last_state*, *committed_state*)
POST	**disk**: log_rep(NoTxn, *last_state*) ∨
	log_rep(NoTxn, *committed_state*)
CRASH	**disk**: log_intact(*last_state*, *committed_state*)



```
SPEC   rename($cwd\_ino$, $oldpath$, $newpath$) $\gg$ fs_recover
PRE    disk: log_rep(NoTxn, $start\_state$)
       start_state: tree_rep($old\_tree$) $\wedge$
           find_subtree($old\_tree$, $cwd$) = $cwd\_tree$ $\wedge$
           tree_inum($cwd\_tree$) = $cwd\_ino$
POST   disk: (($ret$ = (COMPLETED, NoErr) $\vee$ $ret$ = RECOVERED) $\wedge$
                 log_rep(NoTxn, $new\_state$)) $\vee$
               (($ret$ = (COMPLETED, Error) $\vee$ $ret$ = RECOVERED) $\wedge$
                 log_rep(NoTxn, $start\_state$))
       new_state: tree_rep($new\_tree$) $\wedge$
           $mover$ = find_subtree($cwd\_tree$, $oldpath$) $\wedge$
           $pruned$ = tree_prune($cwd\_tree$, $oldpath$) $\wedge$
           $grafted$ = tree_graft($pruned$, $newpath$, $mover$) $\wedge$
           $new\_tree$ = update_subtree($old\_tree$, $cwd$, $grafted$)
```


Running time (seconds)
25
20
15
10
5
0
fscq
xv6
ext4-fuse
ext4
ext4-journal-async
ext4-ordered
ext4-async
git clone
make xv6
make lfs
largefile
smallfile
cleanup
mailbench