Verifying concurrent, crash-safe systems with Perennial

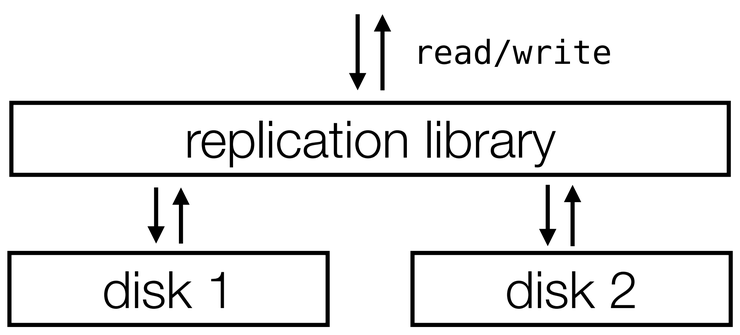
Tej Chajed, Joseph Tassarotti*, Frans Kaashoek, Nickolai Zeldovich

MIT CSAIL and *Boston College

Storage systems need concurrency for performance

Example: replicated disk

Replicates disk writes over two unreliable disks
Behaves like a single disk resilient to underlying failures



Despite simplicity, correctness is subtle

```
func write(a: addr, v: block) {
  lock_address(a)
  d1.write(a, v)
  d2.write(a, v)
  unlock_address(a)
}
```

what if system crashes here?

```
func read(a: addr): block {
  lock_address(a)
  v, ok := d1.read(a)
  if !ok {
    v, _ = d2.read(a)
  }
  unlock_address(a)
  return v
}
```

can this return a block that isn't durable yet?

```
func recover() {
  for a in … {
    // copy from d1 to d2
    v, ok := d1.read(a)
    if !ok { continue; }
    d2.write(a, v)
  }
}
```

what if disk 1 fails?

Perennial is the first framework for **verifying concurrent, crash-safe** systems

Perennial's techniques address challenges integrating crash safety into concurrency reasoning

```
func write(a: addr,
           v: block) {

  lock_address(a)
  d1.write(a, v)
```

crash

```
func recover() {
  …
  v, ok := d1.read(a)
  if !ok { continue; }
  d2.write(a, v)
  …
}
```

user's view:

tid: write(a, v) ⏳

Write was pending when system crashed

tid: write(a, v) ✓

Recovery *logically completes* the pending write
➡ use **recovery helping** to prove this is correct

other challenges and techniques:

Recovery interrupts critical sections
➡ **leases**

Crashes wipe in-memory state
➡ **memory versioning**

We wrote **Goose** to implement storage systems in **Go** and verify them in **Coq** with Perennial

Perennial's Go mail server was easier to verify

compared with CSPEC [OSDI '18]

	Perennial	CSPEC
mail server proof	3,400	4,000
time	2 weeks (**after** framework)	6 months (**with** framework)
code	159 (Go)	215 (Coq)

Perennial proof is both shorter and shows delivered mail is not lost

https://chajed.io/perennial