This lecture:

- Go memory model
- Concurrency primitives
- Concurrency patterns
- Debugging

Go memory model

- why did we assign this reading? gives examples of correct ways to write threaded code.
- "if you need to read this, you're being too clever"
- why goroutines/concurrency?
 - expressivity
 - performance (parallelism)
 - for lab: care about expressivity but not performance
 - only care that code is correct

Goroutines, closures

- goroutines are lightweight threads, run concurrently with each other
 - program terminates when main goroutine exits
- closures: identifier capture, binding gotchas
 - closure.go: go has first-class functions, combine nicely with goroutines. has access to variables in the enclosing scope.
 - loop.go: pattern to spawn goroutines in a loop
 - bad.go: illustrates a common bug: i references outer i, which has been mutated (run multiple times, see different results)

Time

- sleep.go: time.Now, time.Sleep
- sleep-cancel.go: don't write infinite loops; in labs, rf.killed()

Mutexes

- bad.go
- basic.go: basic usage
 - defer, semantics
- per-field.go: locks protect invariants, not "locks protect access to shared data"
 - critical section: temporarily break invariants, restore them before unlock, so nobody observes in-progress updates
 - lock: ensure atomicity of block of code
 - racing audits and transfers
- bank.go

Condition variables

- use case; wait, signal, broadcast
- vote-count-1.go ... vote-count-4.go
- cond.txt, how to avoid bugs
 - lock around use
 - check condition in loop

Channels

- queue-like synchronization primitive
- producer-consumer.go
- wait for N things
- not good for
 - kicking another goroutine, that may or may not be waiting
- unbuffered.go: default, synchronous!
- deadlock.go
- I personally avoid channels for the most part and program with shared memory (mutexes and condvars) instead

Debugging

- Use DPrintf in util.go
 - Can put in test_test.go to see the phase of the test â"€â"€â"€â"€
 - \hat{a} ", // if the leader disconnects, a new one should be elected.
 - a", cfg.disconnect(leader1)

- â", DPrintf("TESTACTION: leader disconnects")
 â", cfg.checkOneLeader()
- â""â"€â"€â"€â"€
- use -race to help detect data races
 - go test -race -run 2A
 - not a proof
 - there are other types of race conditions this can't detect
- SIGOUIT
 - the default SIGQUIT handler for Go prints stack traces for all goroutines (and then exits)
 - Ctrl+\ will send SIGQUIT to current process
- Dealing with leaking goroutines
 - use `ps` to see the running processes
 - send SIGQUIT or SIGKILL using `kill -QUIT pid` or `kill -KILL pid`
- parallel
 - running tests in parallel makes it easier to find concurrency bugs
 - [bash script] by a previous TA https://gist.github.com/jonhoo/f686cacb4b9fe716d5aa

General tools

- use `man toolname`, to lookup the manual of the tool
 - q to quit
 - / to search, n for next, N for prev
- or use `tldr toolname` for quick reference of the tool
 - requires installation <https://tldr.sh/#installation>
- Redirect
 - stdout for normal output
 - stderr for errors
 - **>**, &>
 - > redirect stdout to file but errors are still shown
 - &> redirect both stdout and stderr
 - tee, pipe |, |&
 - tee if you want to redirect to file but still show output
 - echo "example" | tee FILE
 - use |& to redirect both stdout and stderr
- Read the file
 - use your favorite editor
 - quick tools
 - head, tail, less
 - grep
 - -i case insensitive
 - -n line number
 - grep -in searchstring file
 - ripgrep <https://github.com/BurntSushi/ripgrep#installation>
 - like grep but entire directory
 - -i case insensitive
 - rg searchstring