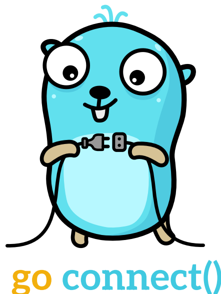
CS3211 Tutorial 7

Classic concurrency problems in C++ and Go

(AY 25/26 Semester 2)

March 26, 2026

(Compiled by Benson, thanks to all past and current TAs!)



Admin Info

- No tutorials next week! (NUS Well-Being Day)
  - You *may* attend another tutorial class.
  - Not necessary unless you still need attendance points.
- Week 12 Tutorial: I plan to cover Tutorial 8+9 (we should be able to finish).
- Week 13 Tutorial: We will do semester recap!
  - There will be some important information on the final exam (and how to approach the code writing questions...).

Contents

H2O Problem

- (C++) Using a barrier
- (Go) Using a daemon goroutine
- (Go) Using oxygen atoms as leader goroutines

FIFO Semaphore

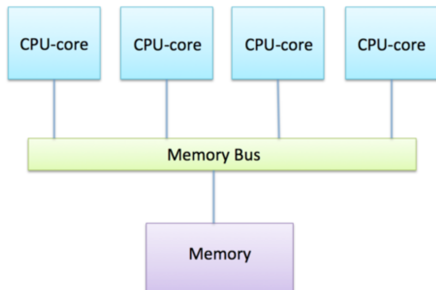
- (C++) Using a ticket queue
- (C++) Using a queue of semaphores
- (Go) Using a buffered channel
- (Go) Using a daemon goroutine

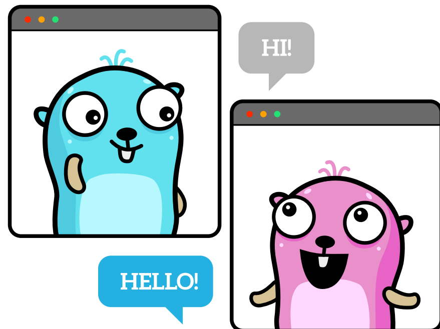
Why this tutorial?

Understand how to approach a problem from two perspectives.

C++: Shared memory.

Go: Message passing.



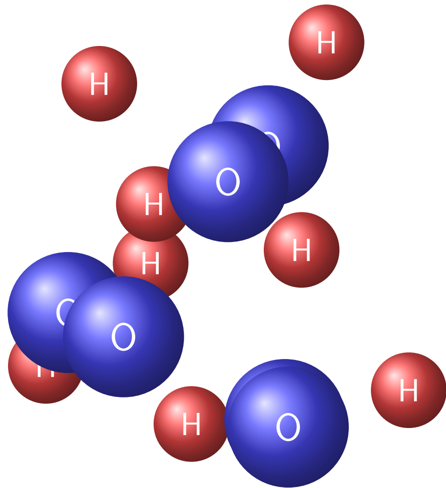


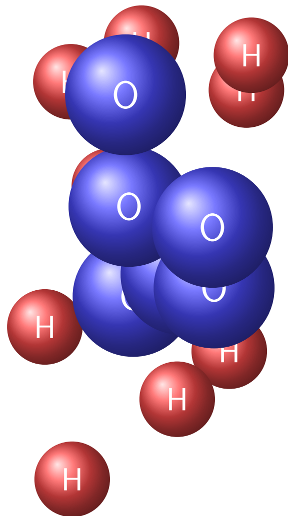
Contents

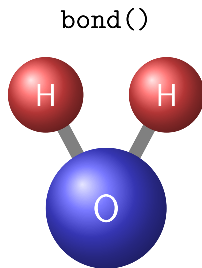
H2O Problem
- (C++) Using a barrier
- (Go) Using a daemon goroutine
- (Go) Using oxygen atoms as leader goroutines

FIFO Semaphore
- (C++) Using a ticket queue
- (C++) Using a queue of semaphores
- (Go) Using a buffered channel
- (Go) Using a daemon goroutine

H2O Problem





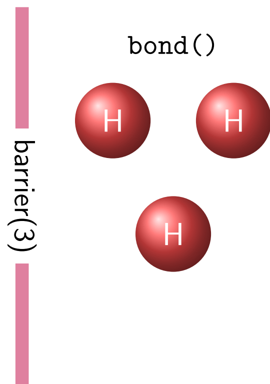


(C++) Using a barrier

First Attempt

```
std::barrier<> barrier{3};

void oxygen(void (*bond)()) {
  barrier.arrive_and_wait();
  bond();
}

void hydrogen(void (*bond)()) {
  barrier.arrive_and_wait();
  bond();
}
```

Link to godbolt

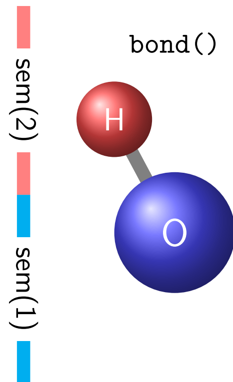


(C++) Using a barrier

Second Attempt

```
1  std::counting_semaphore<> oxygenSem{1};
2  std::counting_semaphore<> hydrogenSem{2};
3
4  void oxygen(void (*bond)()) {
5    oxygenSem.acquire();
6    bond();
7    oxygenSem.release();
8  }
9
10 void hydrogen(void (*bond)()) {
11   hydrogenSem.acquire();
12   bond();
13   hydrogenSem.release();
14 }
```

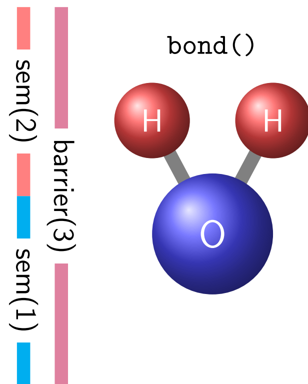
Link to godbolt



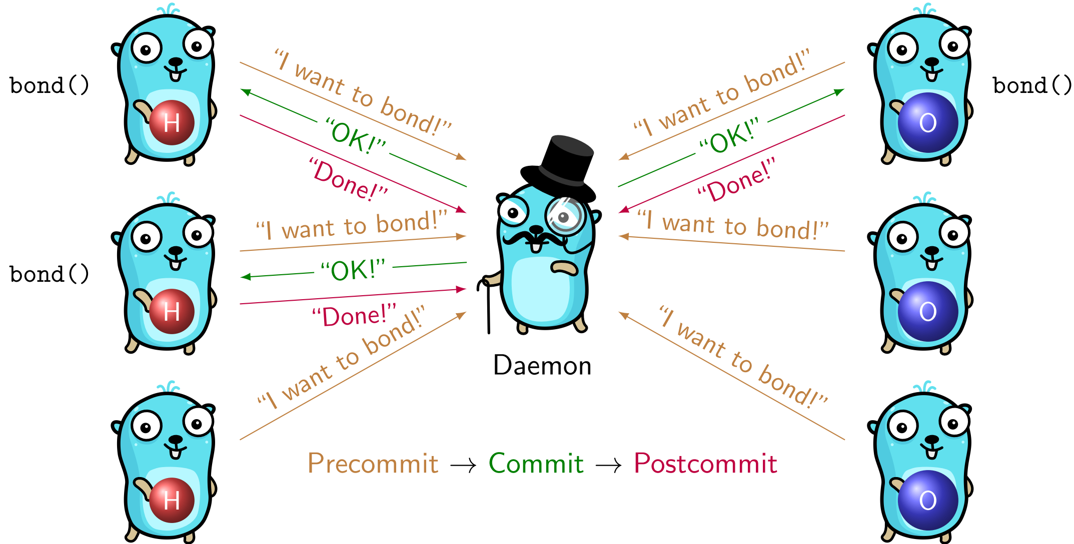
(C++) Using a barrier

```cpp
std::counting_semaphore<> oxygenSem{1};
std::counting_semaphore<> hydrogenSem{2};
std::barrier<> barrier{3};

void oxygen(void (*bond)()) {
  oxygenSem.acquire();
  barrier.arrive_and_wait();
  bond();
  oxygenSem.release();
}

void hydrogen(void (*bond)()) {
  hydrogenSem.acquire();
  barrier.arrive_and_wait();
  bond();
  hydrogenSem.release();
}
```

Link to godbolt



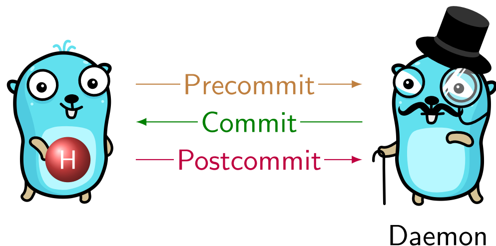
(Go) Using a daemon goroutine



(Go) Using a daemon goroutine

```go
type WaterFactoryWithDaemon struct {
  // Channels for atoms to send
  // their arrival requests
  precomH chan chan struct{}
  precomO chan chan struct{}
}

// Daemon (example with 1 H)
for {
  h1 := <-wfd.precomH // precommit
  h1 <- struct{}{}    // commit
  <-h1                // postcommit
}

// Hydrogen
commit := make(chan struct{})
wfd.precomH <- commit // precommit
<-commit              // commit
bond()
commit <- struct{}{}  // postcommit
```

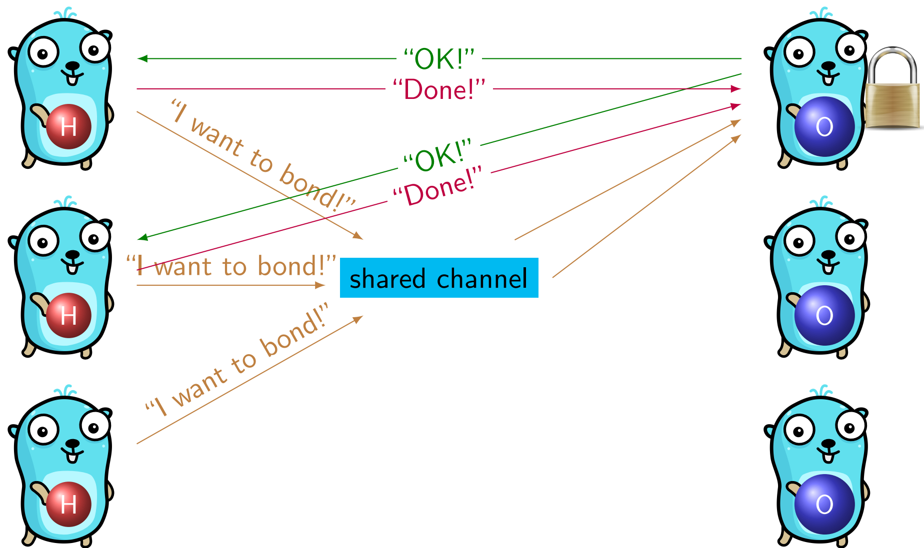
Link to godbolt



(Go) Using a daemon goroutine

- **Performance bottleneck**
  - All bonds have to go through the daemon.
- **Memory leak**
  - Daemon stays alive forever or user has to manage it manually.

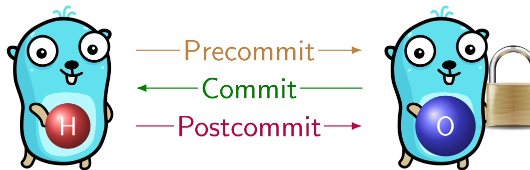
(Go) Using oxygen atoms as leader goroutines



(Go) Using oxygen atoms as leader goroutines

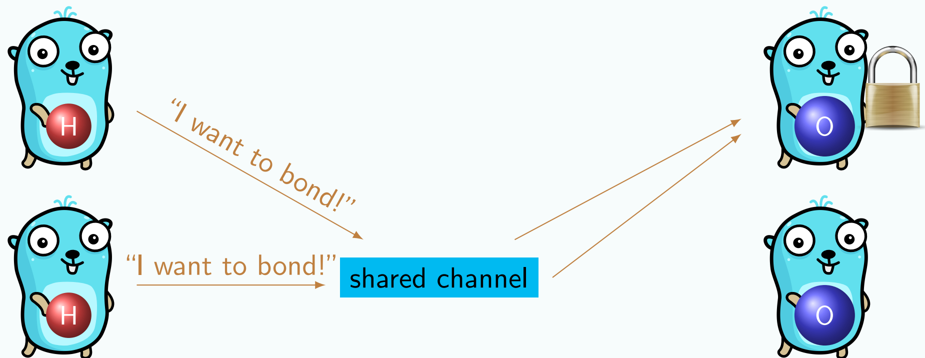
```
1  type WaterFactoryWithLeader struct {
2    oxygenMutex chan struct{}
3    precomH     chan chan struct{}
4  }
5
6  // Oxygen, using 1 H as example
7  <-wf.oxygenMutex
8  h1 := <-wf.precomH    // precommit
9  h1 <- struct{}{}      // commit
10 bond()
11 <-h1                  // postcommit
12 wf.oxygenMutex <- struct{}{}
13
14 // Hydrogen
15 commit := make(chan struct{})
16 wf.precomH <- commit  // precommit
17 <-commit              // commit
18 bond()
19 commit <- struct{}{}  // postcommit
```

Link to godbolt



Extra. H2O2 Problem

Homework: What if we want to bond 2 hydrogen atoms and 2 oxygen atoms at once? (*Hint*: Just add one more step.)



Contents

H2O Problem

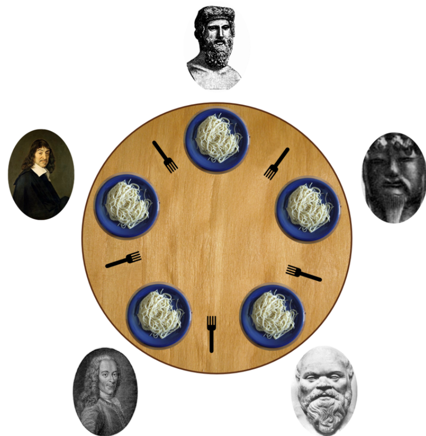
- (C++) Using a barrier
- (Go) Using a daemon goroutine
- (Go) Using oxygen atoms as leader goroutines

FIFO Semaphore

- (C++) Using a ticket queue
- (C++) Using a queue of semaphores
- (Go) Using a buffered channel
- (Go) Using a daemon goroutine

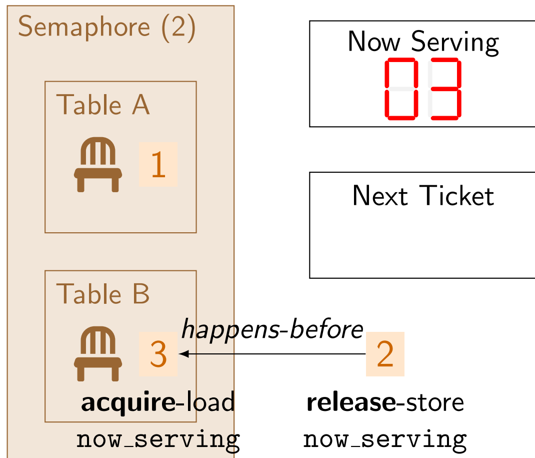
FIFO Semaphore

- Semaphores are not FIFO!
- FIFO Semaphore $\Rightarrow$ **Starvation-freedom**.



(C++) Using a ticket queue

(C++) Using a ticket queue



(C++) Using a ticket queue

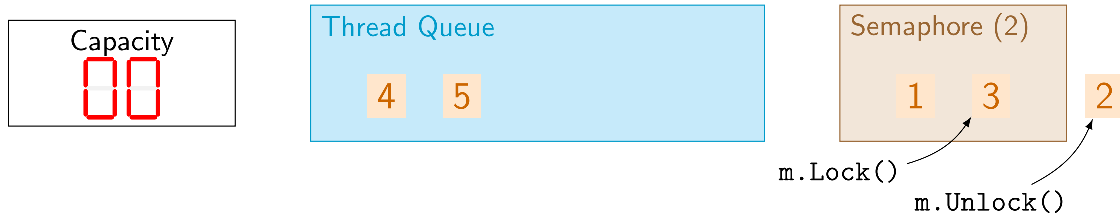
```
1   void acquire() {
2     std::size_t my_ticket =
3           next_ticket.fetch_add(1);
4
5     while (now_serving.load() < my_ticket) {}
6   }
7
8   void release() {
9     now_serving.fetch_add(1);
10  }
```

Link to godbolt

Bonus: Can we avoid the busy loop?

- Is `notify_one` sufficient?

(C++) Using a queue of semaphores



std::mutex::unlock

```
void unlock();    (since C++11)
```

Unlocks the mutex. The mutex must be locked by the current thread of execution, otherwise, the behavior is undefined.

This operation *synchronizes-with* (as defined in `std::memory_order`) any subsequent lock operation that obtains ownership of the same mutex.

(C++) Using a ticket queue

```
struct Waiter {
  std::binary_semaphore sem{0};
};
```

```
void acquire() {
  Waiter waiter;
  {
    std::scoped_lock lock{mut};
    if (count > 0) {
      count--;
      return;
    }
    waiters.push(&waiter);
  }
  waiter.sem.acquire();
}
```

```
void release() {
  Waiter* waiter;
  {
    std::scoped_lock lock{mut};
    if (waiters.empty()) {
      count++;
      return;
    }
    waiter = waiters.front();
    waiters.pop();
  }
  waiter->sem.release(); // safe?
}
```

Link to godbolt

(Go) Using a buffered channel

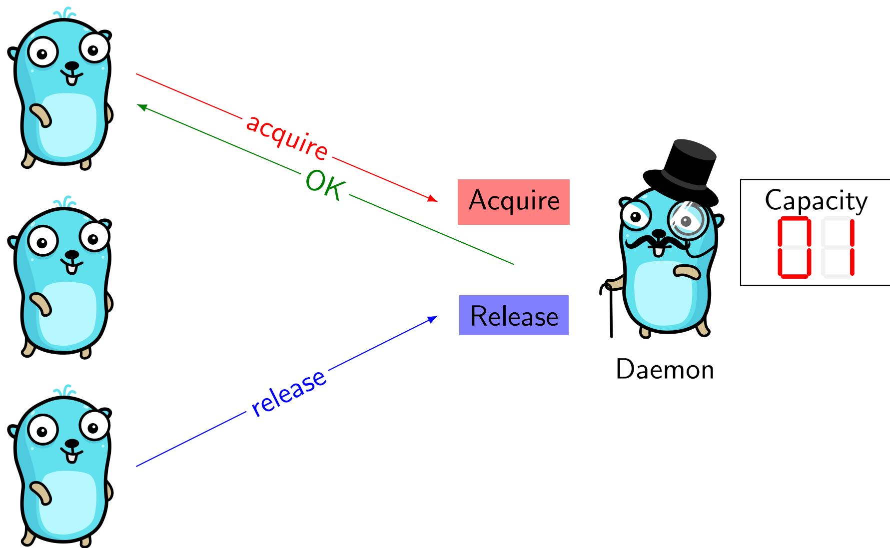
```go
ch := make(chan struct{}, capacity)
for i := 0; i < initValue; i++ {
  ch <- struct{}{}
}
```

```go
func acquire() {
  <-ch
}
```

```go
func release() {
  ch <- struct{}{}
}
```

- Is this FIFO?
  - In practice, yes (as of Go 1.24) – Go runtime implements channels as a lock-protected link-list queue of waiters.
  - However, **not guaranteed by Go specification**.
- Another Issue: Possible for `release()` to block.

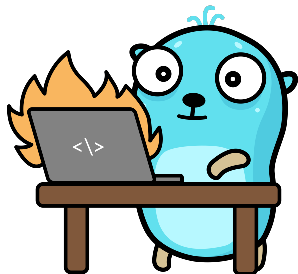
(Go) Using a daemon goroutine



(Go) Using a daemon goroutine

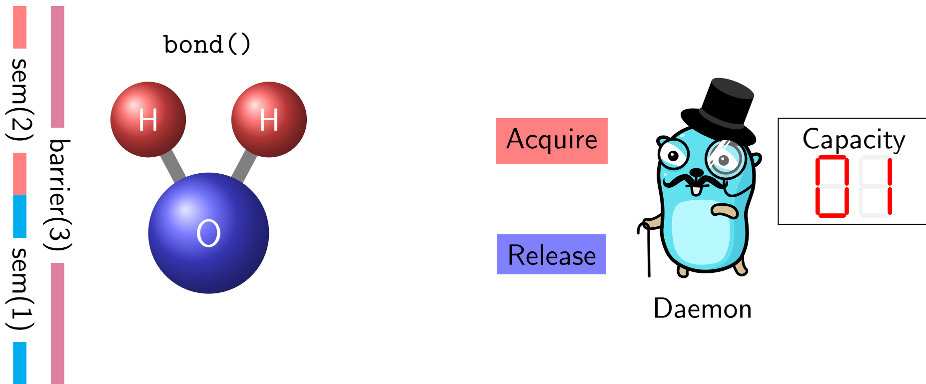
```go
for { // for...select loop in daemon
  select {
  case <-sem.releaseCh:
    if waiters.Len() > 0 {
      ch := waiters.Pop()
      ch <- struct{}{}
    } else {
      count++
    }
  case ch := <-sem.acquireCh:
    if count > 0 {
      count--
      ch <- struct{}{}
    } else {
      waiters.PushBack(ch)
    }
  }
}
```

Link to godbolt

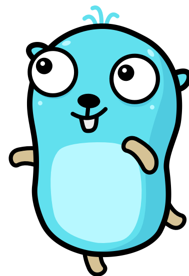


Summary

- We studied various interesting solutions to concurrency problems in C++ and Go.



Attendance



Session ID: 1940159
CS3211 Tutorial 7
26 March 2026 14:00-16:00
BIZ2-02-02 - SEMINAR ROOM 2-2

That's it!

Anonymous Feedback (throughout the semester):

https://forms.gle/6a9T4t88wNwYxG4G8

The link to the tutorial slides will be posted in telegram.