# ReadWrite Lock

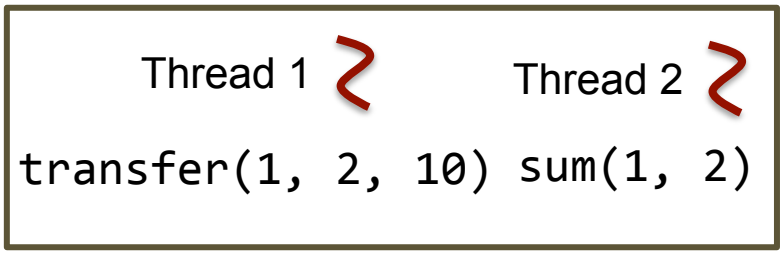
Zhaoguo Wang

# Review Previous Example

```
account *accounts[10];
pthread_mutex_t mus[10];

void transfer(int x, int y, int amount)
{

    pthread_mutex_lock(&mus[x]);
    pthread_mutex_lock(&mus[y]);
    accounts[x]->val -= amount;
    accounts[y]->val += amount;
    pthread_mutex_unlock(&mus[x]);
    pthread_mutex_unlock(&mus[y]);
}

int sum(int x, int y)
{
    pthread_mutex_lock(&mus[x]);
    pthread_mutex_lock(&mus[y]);
    int xv = accounts[x]->val;
    int yv = accounts[y]->val;
    pthread_mutex_unlock(&mus[x]);
    pthread_mutex_unlock(&mus[y]);
    return xv + yv;
}
```

```
typedef struct {
    char *name;
    int val;
} account;
```

No thread is able to observe the middle state of the transfer.

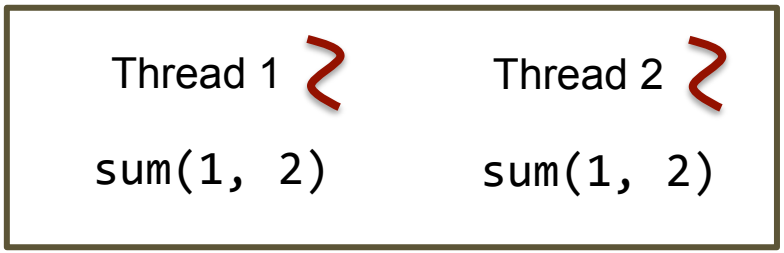


# Review Previous Example

```
account *accounts[10];
pthread_mutex_t mus[10];

void transfer(int x, int y, int amount)
{

    pthread_mutex_lock(&mus[x]);
    pthread_mutex_lock(&mus[y]);
    accounts[x]->val -= amount;
    accounts[y]->val += amount;
    pthread_mutex_unlock(&mus[x]);
    pthread_mutex_unlock(&mus[y]);
}

int sum(int x, int y)
{
    pthread_mutex_lock(&mus[x]);
    pthread_mutex_lock(&mus[y]);
    int xv = accounts[x]->val;
    int yv = accounts[y]->val;
    pthread_mutex_unlock(&mus[x]);
    pthread_mutex_unlock(&mus[y]);
    return xv + yv;
}
```

```
typedef struct {
    char *name;
    int val;
} account;
```

No thread is able to observe the middle state of the transfer.



# Review Previous Example

Thread 1

```
sum(1, 2)

pthread_mutex_lock(&mus[1]);
pthread_mutex_lock(&mus[2]);
int xv = accounts[1]->val;
int yv = accounts[2]->val;
pthread_mutex_unlock(&mus[1]);
pthread_mutex_unlock(&mus[2]);
```

Thread 2

```
sum(1, 2)

pthread_mutex_lock(&mus[1]);
```

wait for thread 2 to release mus[1]

```
pthread_mutex_lock(&mus[2]);
int xv = accounts[1]->val;
int yv = accounts[2]->val;
pthread_mutex_unlock(&mus[1]);
pthread_mutex_unlock(&mus[2]);
```

# Review Previous Example

Thread 1

```
sum(1, 2)
```

```
pthread_mutex_lock(&mus[1]);
pthread_mutex_lock(&mus[2]);
int xv = accounts[1]->val;
int yv = accounts[2]->val;
pthread_mutex_unlock(&mus[1]);
pthread_mutex_unlock(&mus[2]);
```

Thread 2

```
sum(1, 2)
```

```
pthread_mutex_lock(&mus[1]);
```

wait for thread 2 to release mus[1]

```
pthread_mutex_lock(&mus[2]);
int xv = accounts[1]->val;
int yv = accounts[2]->val;
pthread_mutex_unlock(&mus[1]);
pthread_mutex_unlock(&mus[2]);
```

Is it necessary ?

# Review Previous Example

Thread 1

```
sum(1, 2)

pthread_mutex_lock(&mus[1]);
pthread_mutex_lock(&mus[2]);
int xv = accounts[1]->val;
int yv = accounts[2]->val;
pthread_mutex_unlock(&mus[1]);
pthread_mutex_unlock(&mus[2]);
```

Thread 2

```
sum(1, 2)

pthread_mutex_lock(&mus[1]);
pthread_mutex_lock(&mus[2]);
int xv = accounts[1]->val;
int yv = accounts[2]->val;
pthread_mutex_unlock(&mus[1]);
pthread_mutex_unlock(&mus[2]);
```

## Is it necessary ?

NO! If all operations are
read only, there is no need to use lock.

# ReadWrite Lock

Conflicting operations

- Two operations are conflicting, if they access the same resource (memory), and at least one is write.

Only use lock to synchronize conflicting operations

- ReadWrite Lock

# ReadWrite Lock

Allow concurrent accesses for read-only operations, while write operations require exclusive access

- At any time, permit any number of readers to hold lock concurrently,  but only a single writer can hold the lock exclusively.

# pthread API

Type
- pthread_rwlock_t

Apply a read lock to ask for read permit
- int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock)

Apply a write lock to ask for write permit
- int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock)

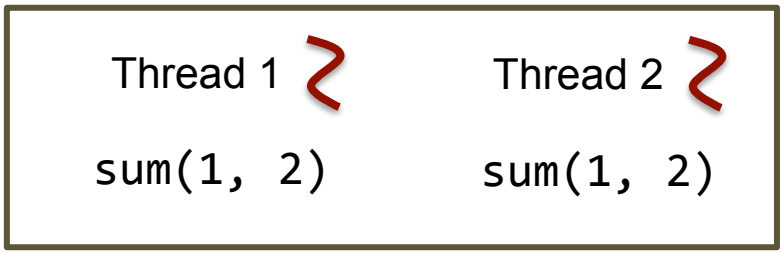
Release lock
- int pthread_rwlock_unlock(pthread_rwlock_t *rwlock)

# Review Previous Example

```
account *accounts[10];
pthread_rwlock_t rwm[10];

void transfer(int x, int y, int amount)
{

    pthread_rwlock_wrlock(&rwm[x]);
    pthread_rwlock_wrlock(&rwm[y]);
    accounts[x]->val -= amount;
    accounts[y]->val += amount;
    pthread_rwlock_unlock(&rwm[x]);
    pthread_rwlock_unlock(&rwm[y]);
}

int sum(int x, int y)
{
    pthread_rwlock_rdlock(&rwm[x]);
    pthread_rwlock_rdlock(&rwm[y]);
    int xv = accounts[x]->val;
    int yv = accounts[y]->val;
    pthread_rwlock_unlock(&rwm[x]);
    pthread_rwlock_unlock(&rwm[y]);
    return xv + yv;
}
```

```
typedef struct {
    char *name;
    int val;
} account;
```

No thread is able to observe the middle state of the transfer.



Thread 1

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
```

Thread 2

```
sum(1, 2)
```

Thread 3

```
transfer(1, 2)
```

Thread 1

```
sum(1, 2)
```

```
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
```

Thread 2

```
sum(1, 2)
```

Thread 3

```
transfer(1, 2)
```

Thread 1

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
```

Thread 2

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
```

Thread 3

```
transfer(1, 2)
```


```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
```


```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
```


```
transfer(1, 2)
rwlock(&rwm[1]);
```

Thread 1

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

Thread 2

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = counts[1]->val
yv = accounts[2]->val
```

Thread 3

```
transfer(1, 2)
rwlock(&rwm[1]);
```

*wait and block*

Thread 1

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

Thread 2

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = counts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

Thread 3

```
transfer(1, 2)
rwlock(&rwm[1]);
```


```
rwlock(&rwm[2]);
```

Thread 1

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

```
sum(1, 2)
rdlock(&rwm[1]);
```

wait and block

Thread 2

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = counts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

```
sum(1, 2)
rdlock(&rwm[1]);
```

wait and block

Thread 3

```
transfer(1, 2)
rwlock(&rwm[1]);
```

*wait and block*

```
rwlock(&rwm[2]);
counts[1]->val -= 10
accounts[2]->val += 10
```

Thread 1

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

```
sum(1, 2)
rdlock(&rwm[1]);
```


```
rdlock(&rwm[2]);
```

Thread 2

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = counts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

```
sum(1, 2)
rdlock(&rwm[1]);
```


```
rdlock(&rwm[2]);
```

Thread 3

```
transfer(1, 2)
rwlock(&rwm[1]);
```


```
rwlock(&rwm[2]);
counts[1]->val -= 10
accounts[2]->val += 10
unlock(&rwm[1]);
unlock(&rwm[2]);
```

Thread 1

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

```
sum(1, 2)
rdlock(&rwm[1]);
```


```
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

Thread 2

```
sum(1, 2)
rdlock(&rwm[1]);
rdlock(&rwm[2]);
xv = counts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

```
sum(1, 2)
rdlock(&rwm[1]);
```


```
rdlock(&rwm[2]);
xv = accounts[1]->val
yv = accounts[2]->val
unlock(&rwm[1]);
unlock(&rwm[2]);
```

Thread 3

```
transfer(1, 2)
rwlock(&rwm[1]);
```


```
rwlock(&rwm[2]);
counts[1]->val -= 10
accounts[2]->val += 10
unlock(&rwm[1]);
unlock(&rwm[2]);
```

# Question

Can you implement your own RW lock with mutex and condition variable?

To make it simple, we have four interfaces:

```
write_lock(rwlock_t *)/ write_unlock(rwlock_t *)
read_lock(rwlock_t *)/ read_unlock(rwlock_t *)
```

# Implementation I

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers; // > 0: read mode, number of readers
  int writer;  // 0: no writer 1: in write mode
} rwlock_t;
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

---

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

---

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

---

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->readers > 0)
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}

void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 1st Implementation

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

# Example Read/Write Counter

```
int global;
rwlock_t rwlock;


void* get(void* arg) {
  read_lock(&rwlock);
  int v = global;
  read_unlock(&rwlock);
}


void* update(void* arg) {
  write_lock(&rwlock);
  global++;
  write_unlock(&rwlock);
}
```

Multiple threads invoke `get` and `update` functions concurrently.

rwlock < readers: 0, writer: 0 >

rwlock < readers: 1, writer: 0 >


Thread 1

get()

read_lock(&rwlock);


Thread 2

get()


Thread 3

update()

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 2, writer: 0 >


Thread 1

get()

```
read_lock(&rwlock);
```


Thread 2

get()

```
read_lock(&rwlock);
```


Thread 3

update()

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 2, writer: 0 >

Thread 1

get()

```
read_lock(&rwlock);
int v = global;
```

Thread 2

get()

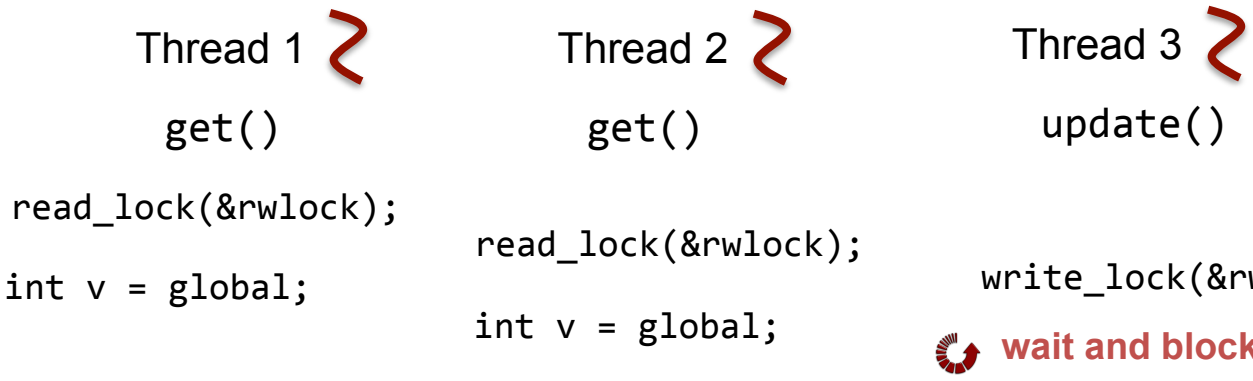
```
read_lock(&rwlock);
```

Thread 3

update()

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 2, writer: 0 >




```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 1, writer: 0 >

Thread 1

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 2

get()

```
read_lock(&rwlock);

int v = global;
```

Thread 3

update()

```
write_lock(&rwlock);
```

wait and block

wait and block

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 0 >

Thread 1

```
get()

read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 2

```
get()

read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 3

```
update()

write_lock(&rwlock);
```

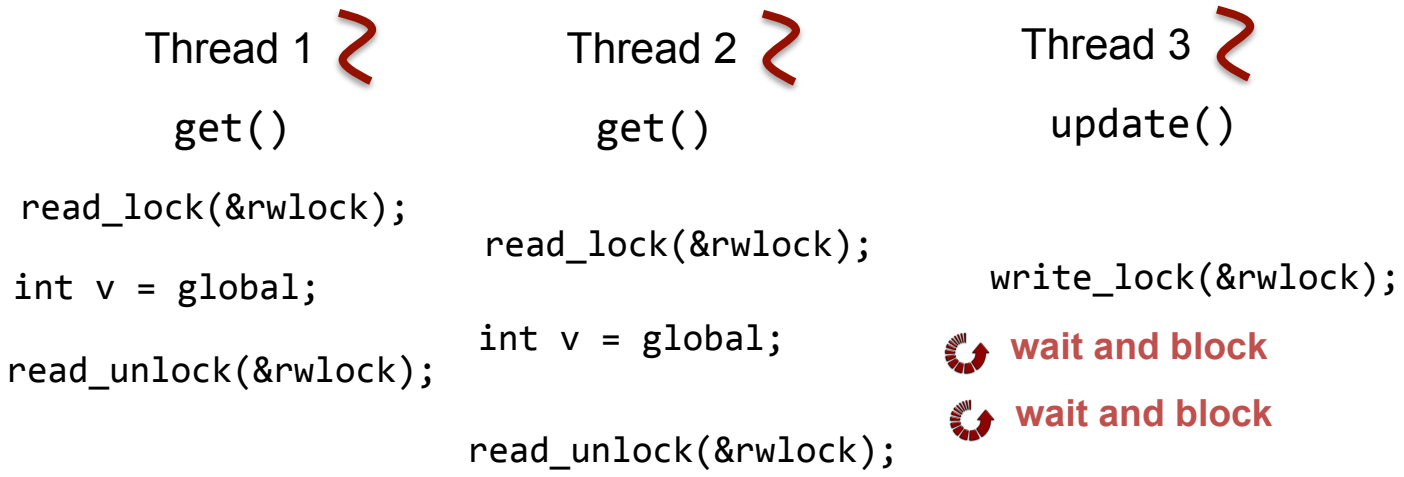
wait and block

wait and block

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 1 >




```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 1 >

Thread 1

```
get()

read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 2

```
get()

read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 3

```
update()

write_lock(&rwlock);
```

wait and block

wait and block

```
global++;
```

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 1 >


Thread 1

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```


Thread 2

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```


Thread 3

update()

```
write_lock(&rwlock);
```


wait and block


wait and block

```
global++;
```


Thread 4

get()

```
read_lock(&rwlock);
```


wait and block

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 0 >


Thread 1

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```


Thread 2

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```


Thread 3

update()

```
write_lock(&rwlock);
```


wait and block


wait and block

```
global++;

write_unlock(&rwlock);
```


Thread 4

get()

```
read_lock(&rwlock);
```


wait and block

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 1, writer: 0 >


Thread 1

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```


Thread 2

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```


Thread 3

update()

```
write_lock(&rwlock);
```


wait and block


wait and block

```
global++;

write_unlock(&rwlock);
```


Thread 4

get()

```
read_lock(&rwlock);
```


wait and block

```
int v = global;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 0 >

| Thread 1 | Thread 2 | Thread 3 | Thread 4 |
|---|---|---|---|
| get() | get() | update() | get() |
| read_lock(&rwlock); | read_lock(&rwlock); | write_lock(&rwlock); | |
| int v = global; | int v = global; | wait and block | |
| read_unlock(&rwlock); | read_unlock(&rwlock); | wait and block | |
| | | global++; | read_lock(&rwlock); |
| | | write_unlock(&rwlock); | wait and block |
| | | | int v = global; |
| | | | read_unlock(&rwlock); |

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 1st Implementation

Readers starve writers

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->reader++;
  pthread_mutex_unlock(&rwl->mutex);
}

void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}

void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 0 >

rwlock < readers: 1, writer: 0 >


Thread 1

get()

read_lock(&rwlock);


Thread 2

get()


Thread 3

update()

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 2, writer: 0 >


Thread 1

get()

```
read_lock(&rwlock);
```


Thread 2

get()

```
read_lock(&rwlock);
```


Thread 3

update()

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 2, writer: 0 >

Thread 1

get()

```
read_lock(&rwlock);
int v = global;
```

Thread 2

get()

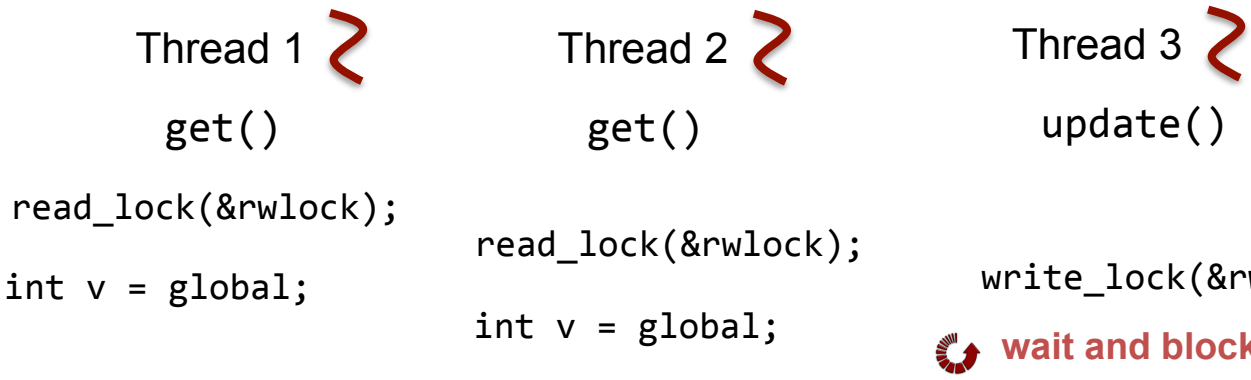
```
read_lock(&rwlock);
```

Thread 3

update()

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

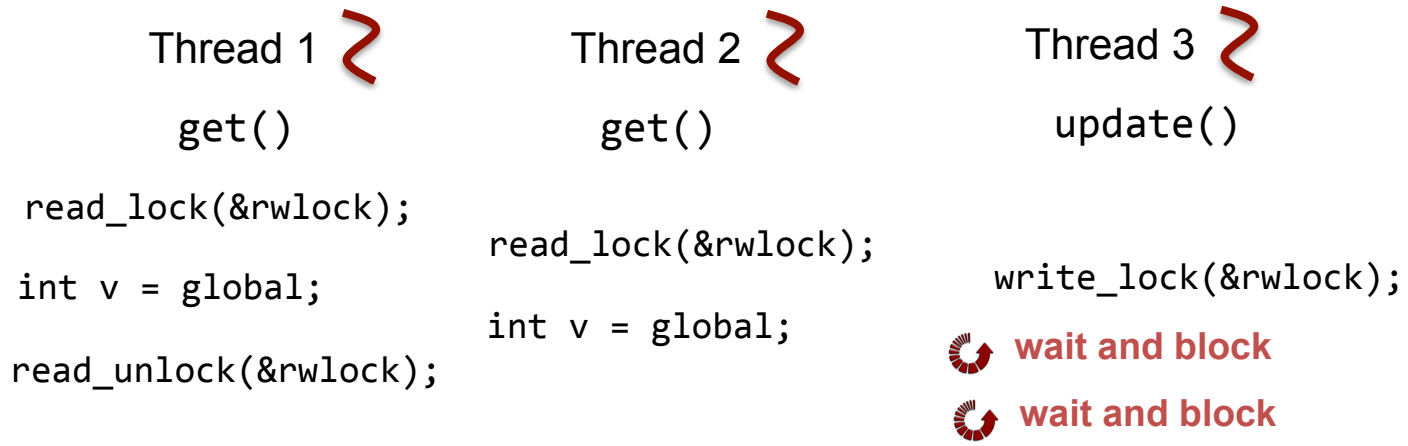
rwlock < readers: 2, writer: 0 >




```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 1, writer: 0 >




```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

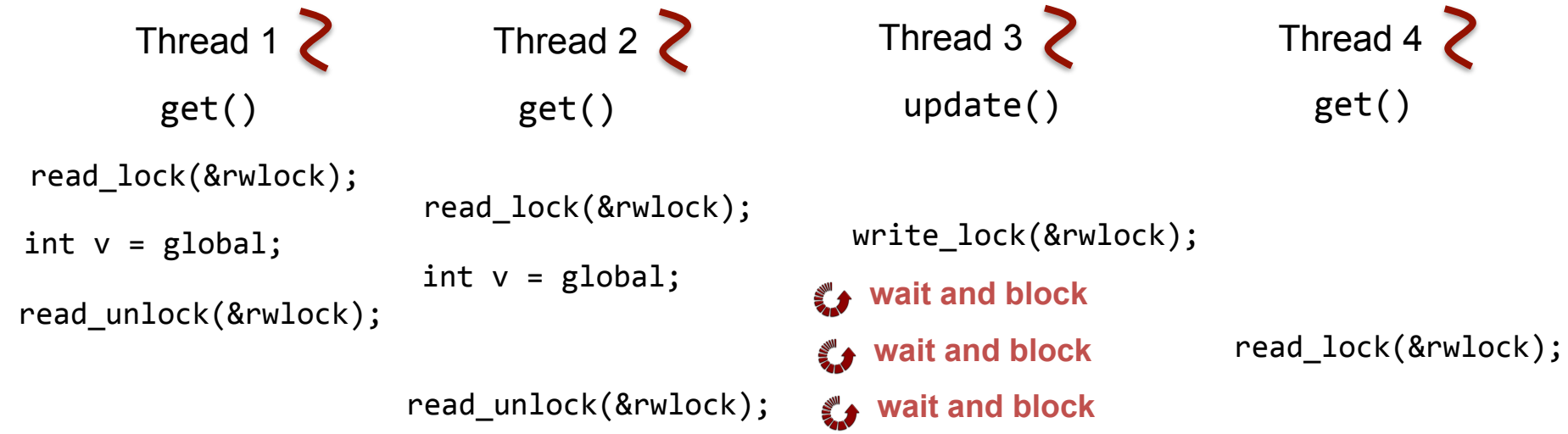
rwlock < readers: 2, writer: 0 >

| Thread 1 | Thread 2 | Thread 3 | Thread 4 |
|---|---|---|---|
| get() | get() | update() | get() |
| read_lock(&rwlock); | read_lock(&rwlock); | write_lock(&rwlock); | |
| int v = global; | int v = global; | wait and block | |
| read_unlock(&rwlock); | | wait and block | read_lock(&rwlock); |

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 1, writer: 0 >




```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 1, writer: 0 >

Thread 1

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 2

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 3

update()

```
write_lock(&rwlock);
```

wait and block

wait and block

wait and block

wait and block

Thread 4

get()

```
read_lock(&rwlock);

int v = global;
```

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 1 >

Thread 1

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 2

get()

```
read_lock(&rwlock);

int v = global;


read_unlock(&rwlock);
```

Thread 3

update()

```
write_lock(&rwlock);
```

wait and block

wait and block

wait and block

wait and block

Thread 4

get()

```
read_lock(&rwlock);

 int v = global;

read_unlock(&rwlock);
```

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 1 >

Thread 1

```
get()

read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 2

```
get()

read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 3

```
update()

write_lock(&rwlock);
```

wait and block

wait and block

wait and block

wait and block

```
global++;

write_unlock(&rwlock);
```

Thread 4

```
get()

read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 1st Implementation

Readers starve writers

*Blocked readers always preempt blocked writers.*

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writers != 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  reader++;
  pthread_mutex_unlock(&rwl->mutex);
}

void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  reader--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  writer++;
  pthread_mutex_unlock(&rwl->mutex);
}

void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
  int w_waiters;
} rwlock_t;
```

## 2nd Implementation

Favor writer over reader.

---

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
```

## 2nd Implementation

Favor writer over reader.

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
  int w_waiters;
} rwlock_t;
```

---

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

## 2nd Implementation

Favor writer over reader.

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
  int w_waiters;
} rwlock_t;
```

---

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
```

## 2nd Implementation

Favor writer over reader.

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
  int w_waiters;
} rwlock_t;
```

---

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
```

## 2nd Implementation

Favor writer over reader.

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
  int w_waiters;
} rwlock_t;
```

---

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  rwl->w_waiters--;
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 2nd Implementation

Favor writer over reader.

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
  int w_waiters;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}

void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  rwl->w_waiters--;
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 2nd Implementation

Favor writer over reader.

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
  int w_waiters;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}

void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  rwl->w_waiters--;
  pthread_mutex_unlock(&rwl->mutex);
}

void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 0, w_waiters: 0 >

rwlock < readers: 1, writer: 0, w_waiters: 0 >

Thread 1 ≀ Thread 2 ≀ Thread 3 ≀

get() get() update()

```
read_lock(&rwlock);
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

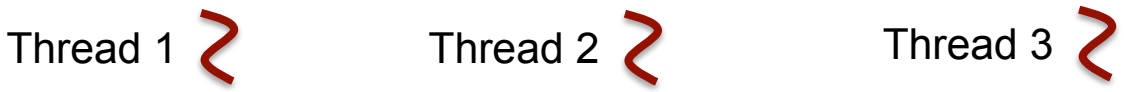
rwlock < readers: 2, writer: 0, w_waiters: 0 >


```
read_lock(&rwlock);
```

```
read_lock(&rwlock);
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 2, writer: 0, w_waiters: 0 >




```
read_lock(&rwlock);
int v = global;
```

```
read_lock(&rwlock);
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 2, writer: 0, w_waiters: 1 >

Thread 1

get()

```
read_lock(&rwlock);

int v = global;
```

Thread 2

get()

```
read_lock(&rwlock);

int v = global;
```

Thread 3

update()

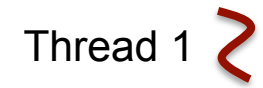
```
write_lock(&rwlock);
```

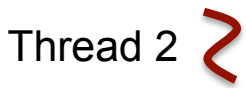
wait and block

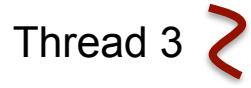
```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  rwl->w_waiters--;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 1, writer: 0, w_waiters: 1 >




get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

get()

```
read_lock(&rwlock);

int v = global;
```

update()

```
write_lock(&rwlock);
```

wait and block

wait and block

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  rwl->w_waiters--;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 1, writer: 0, w_waiters: 1 >

Thread 1

```
get()

read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 2

```
get()

read_lock(&rwlock);

int v = global;
```

Thread 3

```
update()

write_lock(&rwlock);
```

wait and block

Thread 4

```
get()
```

wait and block

```
read_lock(&rwlock);
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  rwl->w_waiters--;
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 0, writer: 0, w_waiters: 1 >

Thread 1

get()

```
read_lock(&rwlock);

int v = global;

read_unlock(&rwlock);
```

Thread 2

get()

```
read_lock(&rwlock);

int v = global;


read_unlock(&rwlock);
```

Thread 3

update()

```
write_lock(&rwlock);
```

wait and block

wait and block

wait and block

Thread 4

get()

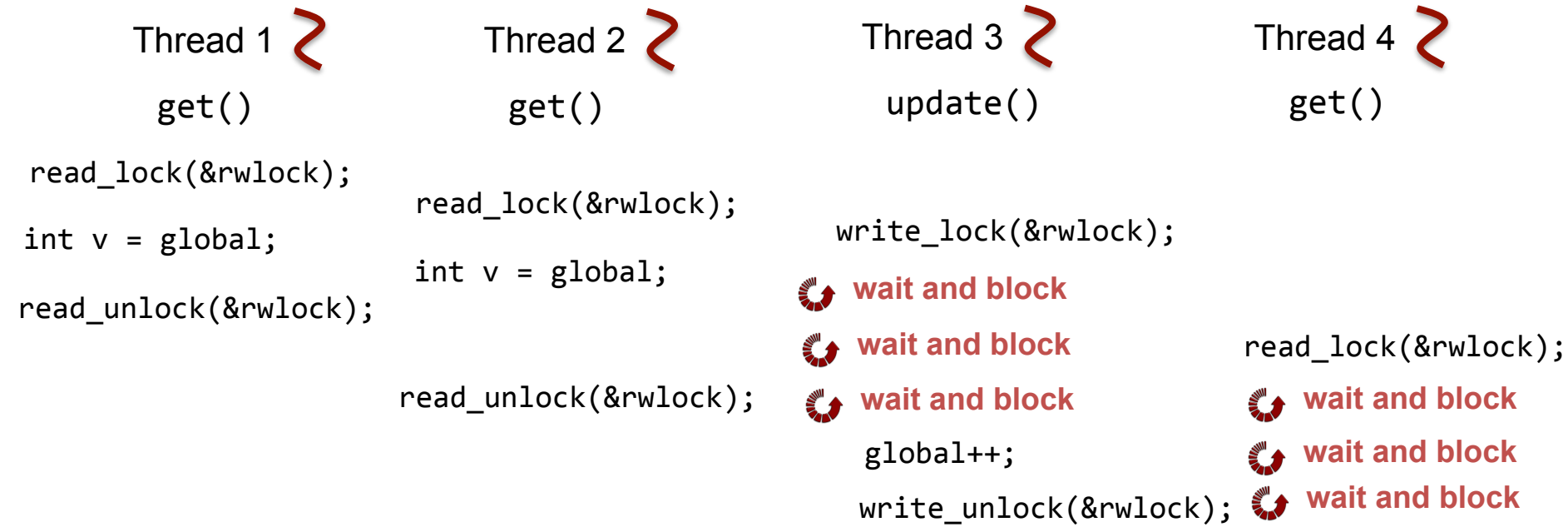
```
read_lock(&rwlock);
```

wait and block

```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  rwl->w_waiters--;
  pthread_mutex_unlock(&rwl->mutex);
}
```

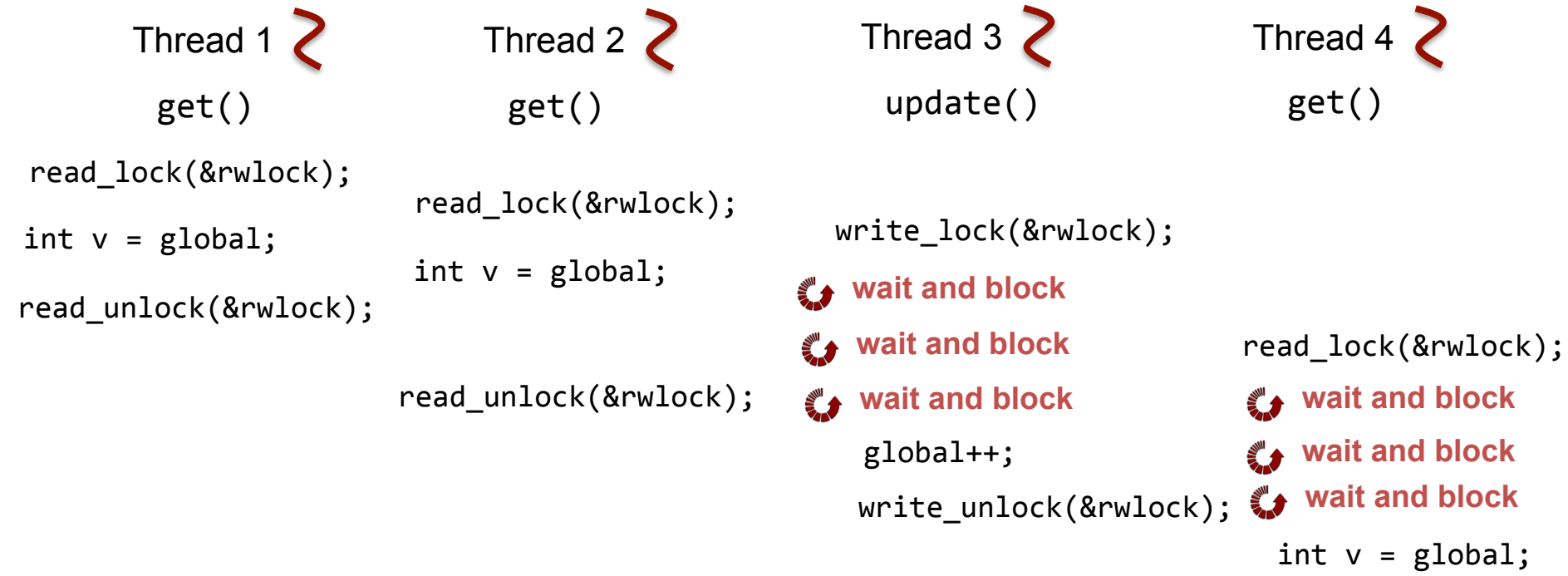
rwlock < readers: 0, writer: 0, w_waiters: 0 >




```
void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

rwlock < readers: 1, writer: 0, w_waiters: 0 >




```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

# 2nd Implementation

Favor writer over reader.

*Blocked writers always preempt blocked readers*

```
typedef struct {
  pthread_mutex_t mutex;
  pthread_cond_t cond;
  int readers;
  int writer;
  int w_waiters;
} rwlock_t;
```

```
void read_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  while(rwl->writer != 0
        || rwl->w_waiters != 0 ) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->readers++;
  pthread_mutex_unlock(&rwl->mutex);
}

void read_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->readers--;
  if(reader == 0)
    pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

```
void write_lock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->w_waiters++;
  while(rwl->writer != 0
        || rwl->readers > 0) {
    pthread_cond_wait(&rwl->cond,
                      &rwl->mutex);
  }
  rwl->writer++;
  rwl->w_waiters--;
  pthread_mutex_unlock(&rwl->mutex);
}

void write_unlock(rwlock_t* rwl) {
  pthread_mutex_lock(&rwl->mutex);
  rwl->writer--;
  pthread_cond_broadcast(&rwl->cond);
  pthread_mutex_unlock(&rwl->mutex);
}
```

# Advanced Topics

Randomly wakeup either all readers or one writer

- Use separate addition variable

Locks are dealt out in the order they are requested

- Create one conditional variable for each waiter, and signal all readers or a single writer, both at the head of the queue