Locks and Condition Variables

Mendel Rosenblum

Locks and Conditional Variables

Optional readings:

Operating Systems: Principles and Practice: Sections 5.2-5.4.

Too-much-milk solution is too complicated

Needed: higher-level synchronization mechanism that provides

- **Mutual exclusion**: easy to create critical sections
- **Blocking**: delay a thread until some desired event occurs

Naming aside: Lock

- The word **lock** is both a noun and a verb
  - Bike lock, Combo lock, Pad lock, Deadbolt lock.
  - Lock the door. Lock the bike. Lock the car.
- Lock the lock?
- C++ language designers disliked lock.lock(); …; lock.unlock()
  - Want something to create **critical sections**
    - Need a mutual exclusion mechanism (`mutex`)

Locks (C++ class `std::mutex`)

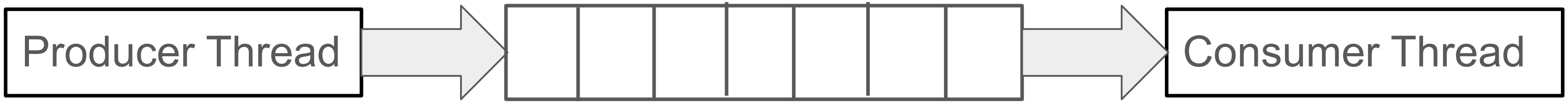
- `lock()`
  - Block until the lock is free
  - Mark the lock as held by the calling thread
- `unlock()`
  - Mark the lock as free
  - Unblock a waiting thread
- try_lock() - Like `lock()` except return error rather than blocking if lock held

Too much milk solution with locks

Both threads:

```
std::mutex mutex;

...

mutex.lock();
if (milk == 0) {
  buy_milk();
  milk = 1;
}
mutex.unlock();
```

Producer/Consumer (a pipe)



Producer/Consumer, v1 broken

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

Initializer list

State variables

```
char Pipe::get() {
  char c;
  mutex.lock();
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v1 (broken)

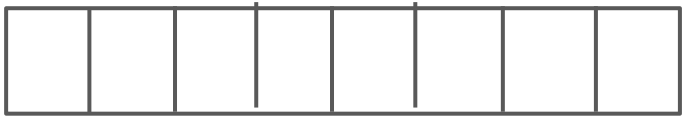
```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  mutex.unlock();
  return c;
}
```

Where are the critical sections?

Execute put('A')

```
mutex unlocked

count:   0

nextPut: 0

nextGet: 0

SIZE:    8

buffer:
```



```
void Pipe::put(char c) {
  mutex.lock();
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

Execute put('A')

```
mutex unlocked

count:   0

nextPut: 0

nextGet: 0

SIZE:    8

buffer:
```


```
mutex unlocked

count:   1

nextPut: 1

nextGet: 0

SIZE:    8

buffer:
```

A							

Execute put('B'), put('C')

```
mutex unlocked

count:   1

nextPut: 1

nextGet: 0

SIZE:    8

buffer:
```

A							

```
void Pipe::put(char c) {
  mutex.lock();
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

Execute put('B'), put('C')

```
mutex unlocked

count:   1

nextPut: 1

nextGet: 0

SIZE:    8

buffer:
```

A							

```
mutex unlocked

count:   3

nextPut: 3

nextGet: 0

SIZE:    8

buffer:
```

A	B	C					

Execute get()

```
mutex unlocked

count:   3

nextPut: 3

nextGet: 0

SIZE:    8

buffer:
```

A	B	C					

```
char Pipe::get() {
  char c;
  mutex.lock();
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  mutex.unlock();
  return c;
}
```

Execute get()

```
mutex unlocked

count:   3

nextPut: 3

nextGet: 0

SIZE:    8

buffer:
```

A	B	C					

```
mutex unlocked

count:   2

nextPut: 3

nextGet: 1

SIZE:    8    c:  A

buffer:
```

A	B	C					

Execute put('D'), put('E'), put('F'), put('G')

```
mutex unlocked

count:    2

nextPut: 3

nextGet: 1

SIZE:     8

buffer:
```

A	B	C					

```
void Pipe::put(char c) {
  mutex.lock();
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

Execute put('D'), put('E'), put('F'), put('G')

```
mutex unlocked

count:   2

nextPut: 3

nextGet: 1

SIZE:    8

buffer:
```

A	B	C					

```
mutex unlocked

count:   6

nextPut: 7

nextGet: 1

SIZE:    8

buffer:
```

A	B	C	D	E	F	G	

Execute put('H'), put('I')

```
mutex unlocked

count:   6

nextPut: 7

nextGet: 1

SIZE:    8

buffer:
```

A	B	C	D	E	F	G	

```
void Pipe::put(char c) {
  mutex.lock();
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

Execute put('H'), put('I')

```
mutex unlocked

count:    6

nextPut: 7

nextGet: 1

SIZE:     8

buffer:
```

A	B	C	D	E	F	G	

```
mutex unlocked

count:    8

nextPut: 1

nextGet: 1

SIZE:     8

buffer:
```

I	B	C	D	E	F	G	H

Execute put('J')

```
mutex unlocked

count:   8

nextPut: 1

nextGet: 1

SIZE:    8

buffer:
```

A	B	C	D	E	F	G	H

```
void Pipe::put(char c) {
  mutex.lock();
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

Execute put('J')

```
mutex unlocked

count:   8

nextPut: 1

nextGet: 1

SIZE:    8

buffer:
```

A	B	C	D	E	F	G	H

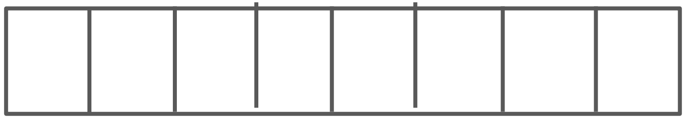
```
mutex unlocked

count:   9

nextPut: 2

nextGet: 1

SIZE:    8

buffer:
```

I	J	C	D	E	F	G	H

What was the error?

Execute get() when empty

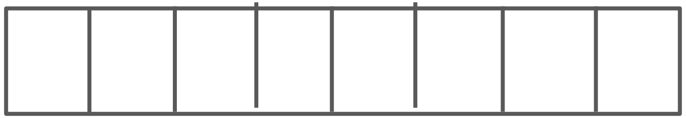
```
mutex unlocked

count:    0

nextPut:  0

nextGet:  0

SIZE:     8

buffer:
```



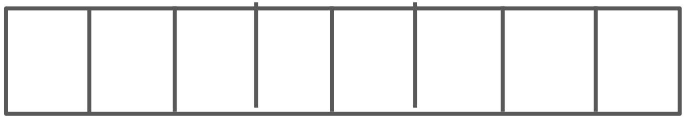
```
char Pipe::get() {
  char c;
  mutex.lock();
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  mutex.unlock();
  return c;
}
```

Execute get() when empty

```
mutex unlocked

count:   0

nextPut: 0

nextGet: 0

SIZE:    8

buffer:
```



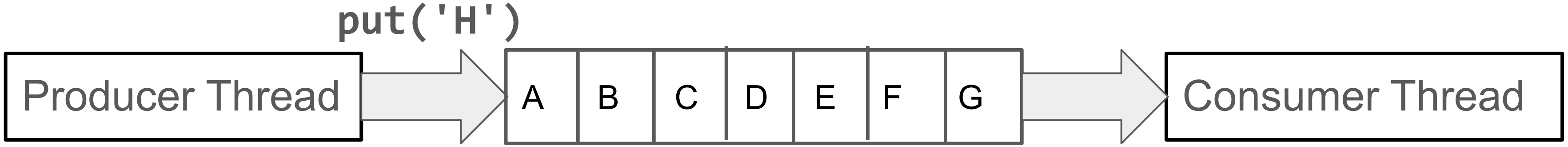
```
mutex unlocked

count:  -1

nextPut: 0

nextGet: 1

SIZE:    8    c:  undefined

buffer:
```



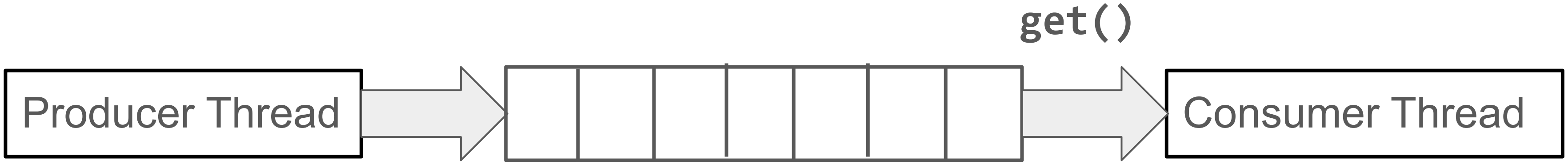
What was the error?

Producer/Consumer flow control

- The Producer produces faster than Consumer and the buffer overflows



- The Consumer consumes faster than the Producer and no char available



Need to be able to wait in producer/consumer

- Consumer
  - No characters in buffer (when count == 0)
- Producer
  - Buffer is full (when count == SIZE)

Producer/Consumer, v2 (broken)

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  while (count == 0) {
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  mutex.unlock();
  return c;
}
```

What goes wrong?

Producer/Consumer, v2.5 (broken)

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  while (count == SIZE) {
  }
  mutex.lock();
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  while (count == 0) {
  }
  mutex.lock();
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  mutex.unlock();
  return c;
}
```

What goes wrong?

Producer/Consumer, v2.9

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    mutex.unlock();
    mutex.lock();
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  while (count == 0) {
    mutex.unlock();
    mutex.lock();
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  mutex.unlock();
  return c;
}
```

What is the problem?

Excess **busy-waiting**

Announcements

Need a way of waiting/blocking inside of a critical section

- **Condition Variable**
  - Atomically release lock and go to sleep waiting for a condition to change
- Useful mental model: Special kind of list a thread can go to sleep on

Condition Variables in C++

- `std::condition_variable`
  - `wait(lock):`
    - Release lock and puts threads to sleep (block)
    - Re-acquire lock went awoken
  - `notify_one():`
    - Wake up one thread if there is one sleeping
  - `notify_all():`
    - Wake up all the threads that are sleeping

Note: Notified thread might not re-acquire lock right away

Condition might not be valid when awoken thread executes again

Producer/Consumer, v3

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  while (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  while (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Why `while` rather than `if`?

Producer/Consumer, v3 (broken)

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Describe what can go wrong.

T1 T2 T3

Producer/Consumer, v3 (broken)

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T1 T2 T3

Producer/Consumer, v3 (broken) - T1 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T1

T2 T3

Producer/Consumer, v3 (broken) - T1 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T1

T2 T3

Producer/Consumer, v3 (broken) - T1 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

T1

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T2 T3

Producer/Consumer, v3 (broken) - T2 put('A')

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T1

T2

T3

Producer/Consumer, v3 (broken) - T2 put('A')

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T2

T1

T3

Producer/Consumer, v3 (broken) - T2 put('A')

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

T2

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T1

T3

Producer/Consumer, v3 (broken) - T2 put('A')

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T2 put('A')

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

T2

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T1

T3

Producer/Consumer, v3 (broken) - T3 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
T2 if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
T3 mutex.lock();
  if (count == 0) {
T1  charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T2 put('A')

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
T2 charAdded.notify_one();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
T3 mutex.lock();
  if (count == 0) {
T1  charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T2 put('A')

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
T2  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
T3  mutex.lock();
  if (count == 0) {
T1    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T3 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
T2 mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
T3 if (count == 0) {
T1   charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T3 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
T2 mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
T1  charAdded.wait(mutex);
  }
T3 count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T3 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
T2 mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
T1  charAdded.wait(mutex);
  }
  count--;
T3 c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T3 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
T2 mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
T1  charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
T3 nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T3 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
T2 mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
T1  charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
T3 if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T3 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
T2 mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
T1  charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
T3 charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Producer/Consumer, v3 (broken) - T3 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

T2

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);      // T1
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;                     // T3
}
```

Producer/Consumer, v3 (broken) - T1 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

T2

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
T1 ->
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
T3 ->
  return c;
}
```

Producer/Consumer, v3 (broken) - T1 get()

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

T2

```
char Pipe::get() {
  char c;
  mutex.lock();
  if (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

T1

T3

Error: c is undefined

Producer/Consumer, v3 notify_all

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_all();
  mutex.unlock();
}
```

```
char Pipe::get() {
  char c;
  mutex.lock();
  while (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_all();
  mutex.unlock();
  return c;
}
```

Will this work?

How many locks should you use?

How many locks should you use?

- More locks
  - May permit more concurrency (less lock contention)
- Less locks:
  - More locks lead to complexity, potential for problems like races & deadlock (Monday's lecture)
  - Lock acquisition is relatively expensive
    - Fewer locks may result in less overhead
- Terminology: **course-grained** locking vs **fine-grained** locking
- Examples: Linux and Python

Best approach

- Few locks as possible, while providing an acceptable level of lock contention
- Good general approach:
  - Associate a lock with a collection of related variables
  - Example: **monitors**

Monitor style locking

- Monitors contain:
  - A shared data structure
  - A collection of methods
  - A lock
- Lock must be held whenever accessing the shared data
  - Typically each procedure acquires the lock at beginning and releases before returning
- One or more **condition variables** used for waiting
- Language support: Example: Java `synchronized` keyword

Producer/Consumer, v3 - **monitor example**

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  mutex.lock();
  while (count == SIZE) {
    charRemoved.wait(mutex);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
  mutex.unlock();
}
```

Single lock

Shared state

```
char Pipe::get() {
  char c;
  mutex.lock();
  while (count == 0) {
    charAdded.wait(mutex);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  mutex.unlock();
  return c;
}
```

Methods that operate on state holding the lock

Producer/Consumer, v4 (correct)

```
class Pipe {
  Pipe() : count(0), nextPut(0), nextGet(0) {}
  void put(char c);
  char get();

  std::mutex mutex;
  std::condition_variable charAdded, charRemoved;
  char buffer[SIZE];
  int count;
  int nextPut;
  int nextGet;
};

void Pipe::put(char c) {
  std::unique_lock<std::mutex> lock(mutex);
  while (count == SIZE) {
    charRemoved.wait(lock);
  }
  count++;
  buffer[nextPut] = c;
  nextPut++;
  if (nextPut == SIZE) {
    nextPut = 0;
  }
  charAdded.notify_one();
}
```

```
char Pipe::get() {
  char c;
  std::unique_lock<std::mutex> lock(mutex);
  while (count == 0) {
    charAdded.wait(lock);
  }
  count--;
  c = buffer[nextGet];
  nextGet++;
  if (nextGet == SIZE) {
    nextGet = 0;
  }
  charRemoved.notify_one();
  return c;
}
```

`std:unique_lock<std:mutex>` - hold lock for scope