Each 6.1800 lecture will come with an outline. You can fill this in during lecture, after lecture, or not at all — it's entirely up to you how you use it. The goal of these outlines is to help you understand the main points that you should be taking away from each lecture. In some cases we will also include examples of things you should be able to do after each lecture.

*In the past, these outlines have proved to be an effective tool for studying for the exams. Note that the outlines are **not exhaustive**; there will be topics and nuances in lecture that aren't captured by the outline.*

Lecture 04: Bounded Buffers + Locks

- What is virtualization? *(Same first question as last time! It's important)*
- What is a bounded buffer? What does an operating system use it to do?
- What is a race condition?
- What do locks do? For example, what is the difference between this code snippet:
  ```
  bb.buf[bb.in mod N] <- message
  bb.in <- bb.in + 1
  ```
  And this code snippet:
  ```
  acquire(bb.lock)
  bb.buf[bb.in mod N] <- message
  bb.in <- bb.in + 1
  release(bb.lock)
  ```
- Why does the final send/receive code look the way it does? Specifically:
  - What is the purpose of the outer acquire/release?
  - What is the purpose of the *inner* acquire/release (inside of the while loop)?
  - *Note: a good way to test that you fully understand the final send/receive code is to make sure you understand why all of the previous versions didn't work. And a good way to do <u>that</u> is to try working through the incorrect code in the same way we did in lecture, and see what goes wrong. Come to office hours if you have questions!*
- What do we mean by "inconsistent state" in the context of locks and atomic actions?
- What is deadlock?
- Why do we need an atomic exchange operation to implement locks?
- At this point in 6.1800, it doesn't seem like locks perform well, or are a particularly good design; why? (There are multiple answers)