0. Introduction
- Now: how to systematically deal with failures, or build "fault-tolerant" systems
- We'll allow more complicated failures and also try to recover from failures
- Thinking about large, distributed systems. 100s, 1000s, even more machines, potentially located across the globe.
- Will also have to think about what these applications are doing, what they need

1. Building fault-tolerant systems
- General approach:
  1. Identify possible faults (software, hardware, design, operation, environment, ...)
  2. Detect and contain
  3. Handle the fault
- Caveats
  - Components are always unreliable
  - Reliability comes at a cost; always a tradeoff
  - All of this is tricky; we iterate.
  - We'll have to rely on *some* code to work correctly

2. Quantifying reliability
- Goal: increase availability
- Metrics:
  - MTTF = mean time to failure
  - MTTR = mean time to repair
  - MTBF = mean time between failures = MTTF + MTTR
  - availability = MTTF / (MTTF + MTTR)
- This is one way to think about reliability! Different systems will get more specific

3. Reliability via Replication
- To improve availability, add redundancy
- One way to add redundancy: replication
- Today: replication within a single machine to deal with disk failures
- Tomorrow in recitation: replication across machines to deal with machine failures.

4. Dealing with disk failures
- If the disk fails, that's very bad (data is gone!)
- Probability of *some* disk in a large system failing is high

5. Whole-disk failures
- RAID 1: Mirror data across 2 disks.
- RAID 4: With N disks, add an additional parity disk. Sector i on the parity disk is the XOR of all of the sector i's from the data disk.
- Handles single-disk failures with only one additional disk
- RAID 5: Same as RAID 4, except intersperse the parity sectors amongst all N+1 disks to load balance writes.
- RAID 5 used in practice, but falling out in favor of RAID 6, which uses the same techniques but provides protection against two disks failing at the same time.

6. Your future
- RAID doesn't solve everything.
  - E.g., what about failures that aren't independent?
  - We are going to work to develop abstractions that allow us to be more systematic about handling failures.
  - Lectures will tackle these abstractions on larger and larger systems.