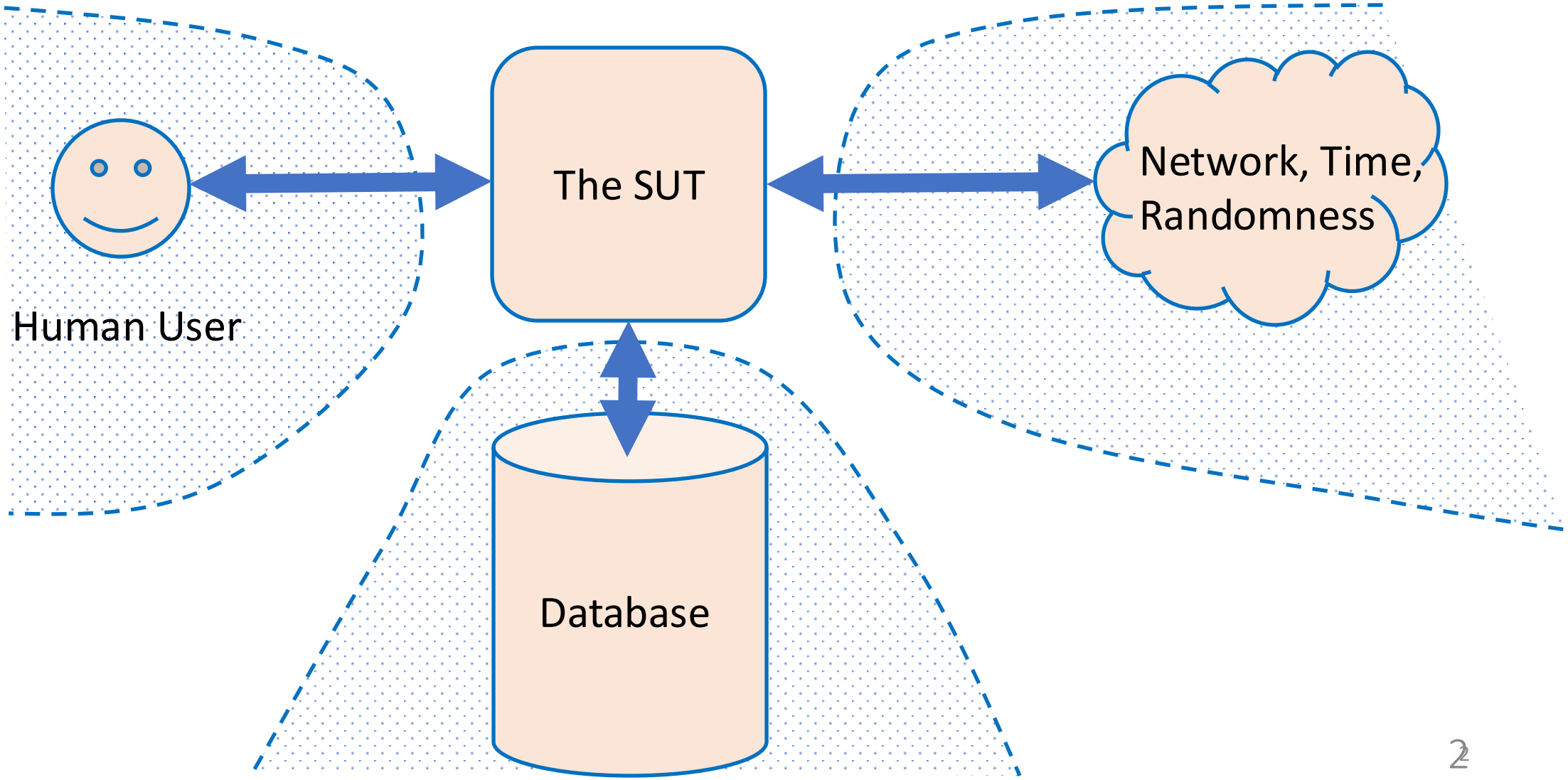
CS 4530: Fundamentals of Software Engineering
Lesson 2.5
Testing Integrated Software Systems

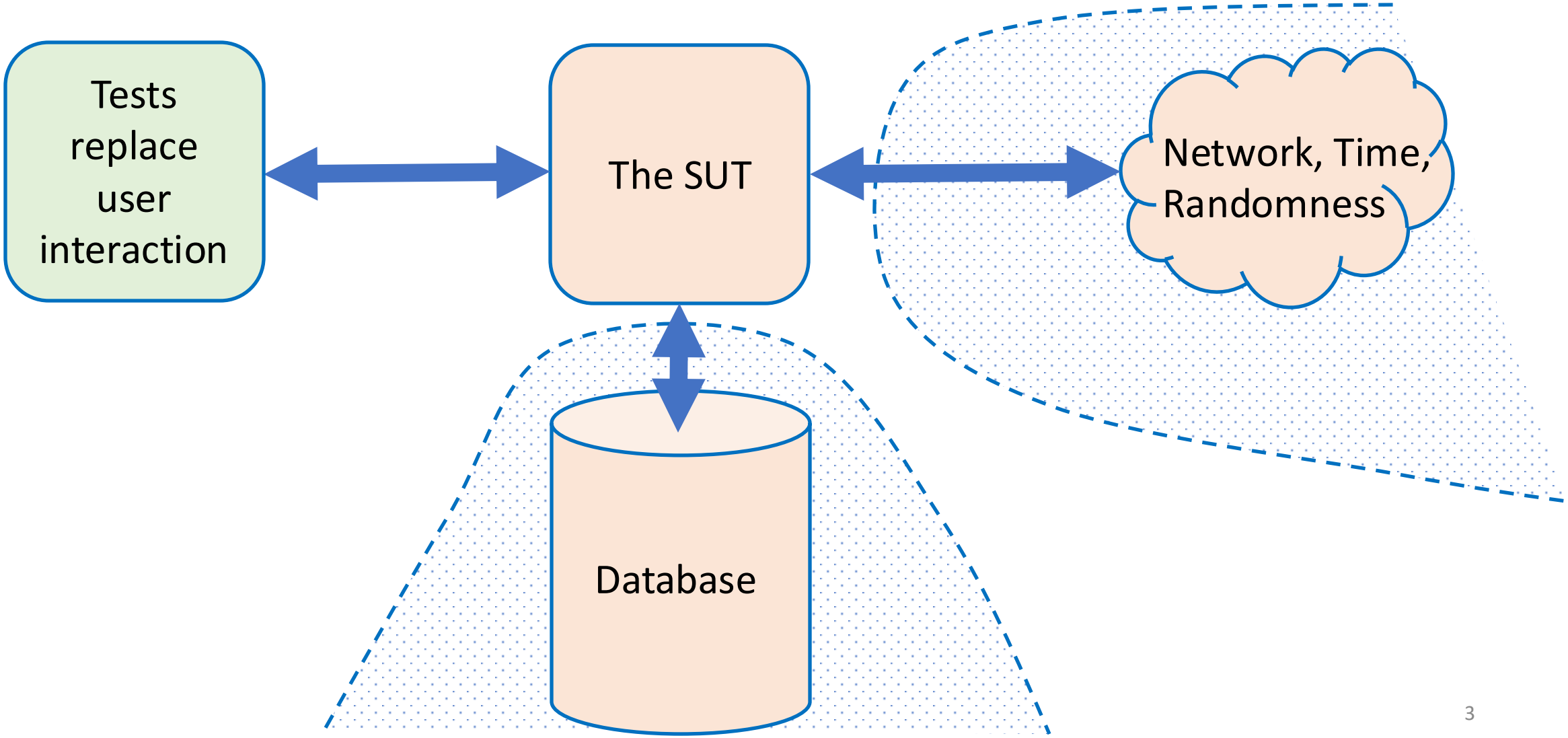
Rob Simmons

Khoury College of Computer Sciences

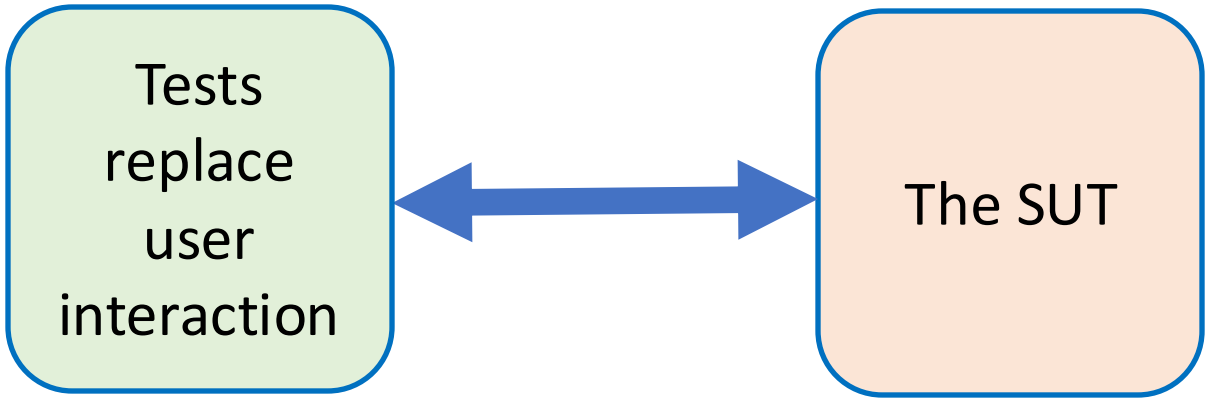
Software interacts with an environment



Remove unnecessary parts of environment



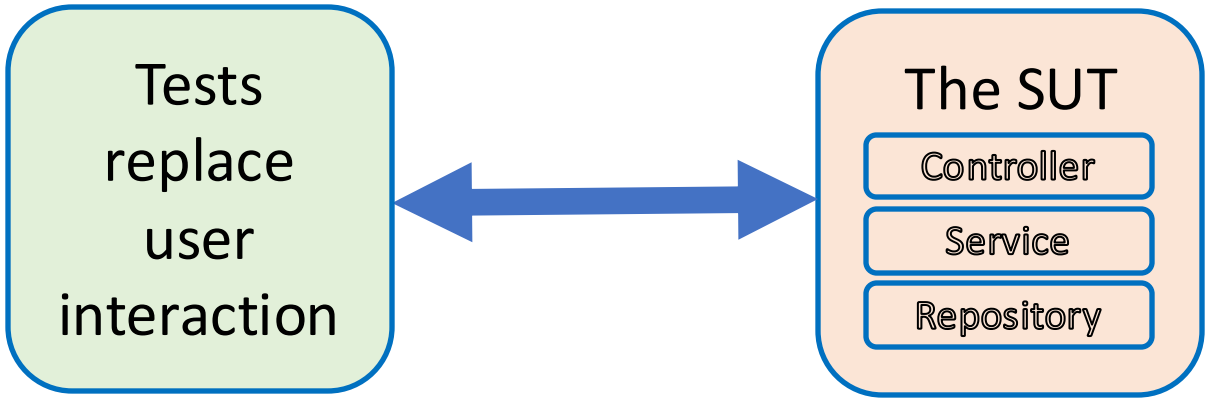
Test the appropriate connection points




```
describe('GET /api/user/:id', () => {
  it('should 404 for nonexistent users', async () => {
    response = await supertest(app).get(`/api/user/${randomUUID().toString()}`);
    expect(response.status).toBe(404);
    expect(response.body).toStrictEqual({ error: 'User not found' });
  });

  it('should return existing users', async () => {
    response = await supertest(app).get(`/api/user/user1`);
    expect(response.status).toBe(200);
    expect(response.body).toStrictEqual({ ...user1, createdAt: expect.anything() });
```

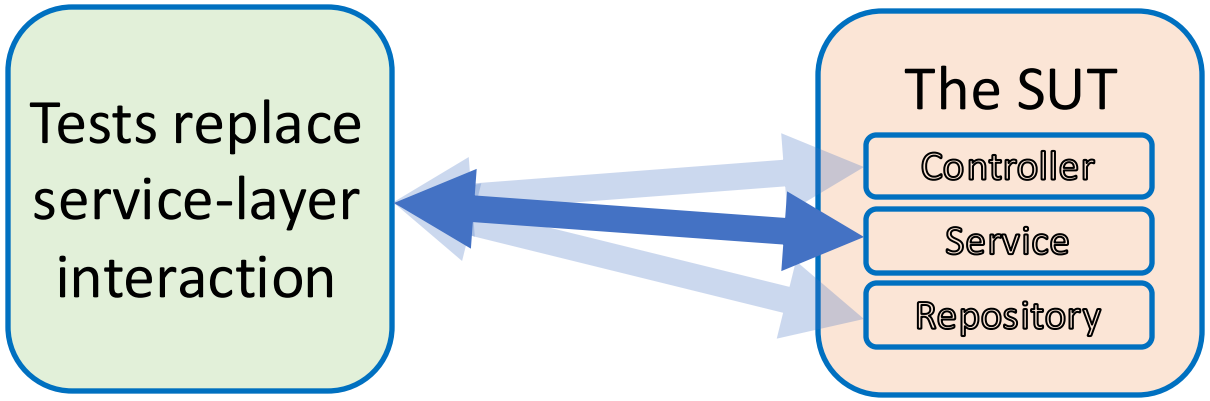
Test the appropriate connection points




```
describe('GET /api/user/:id', () => {
  it('should 404 for nonexistent users', async () => {
    response = await supertest(app).get(`/api/user/${randomUUID().toString()}`);
    expect(response.status).toBe(404);
    expect(response.body).toStrictEqual({ error: 'User not found' });
  });

  it('should return existing users', async () => {
    response = await supertest(app).get(`/api/user/user1`);
    expect(response.status).toBe(200);
    expect(response.body).toStrictEqual({ ...user1, createdAt: expect.anything() });
```

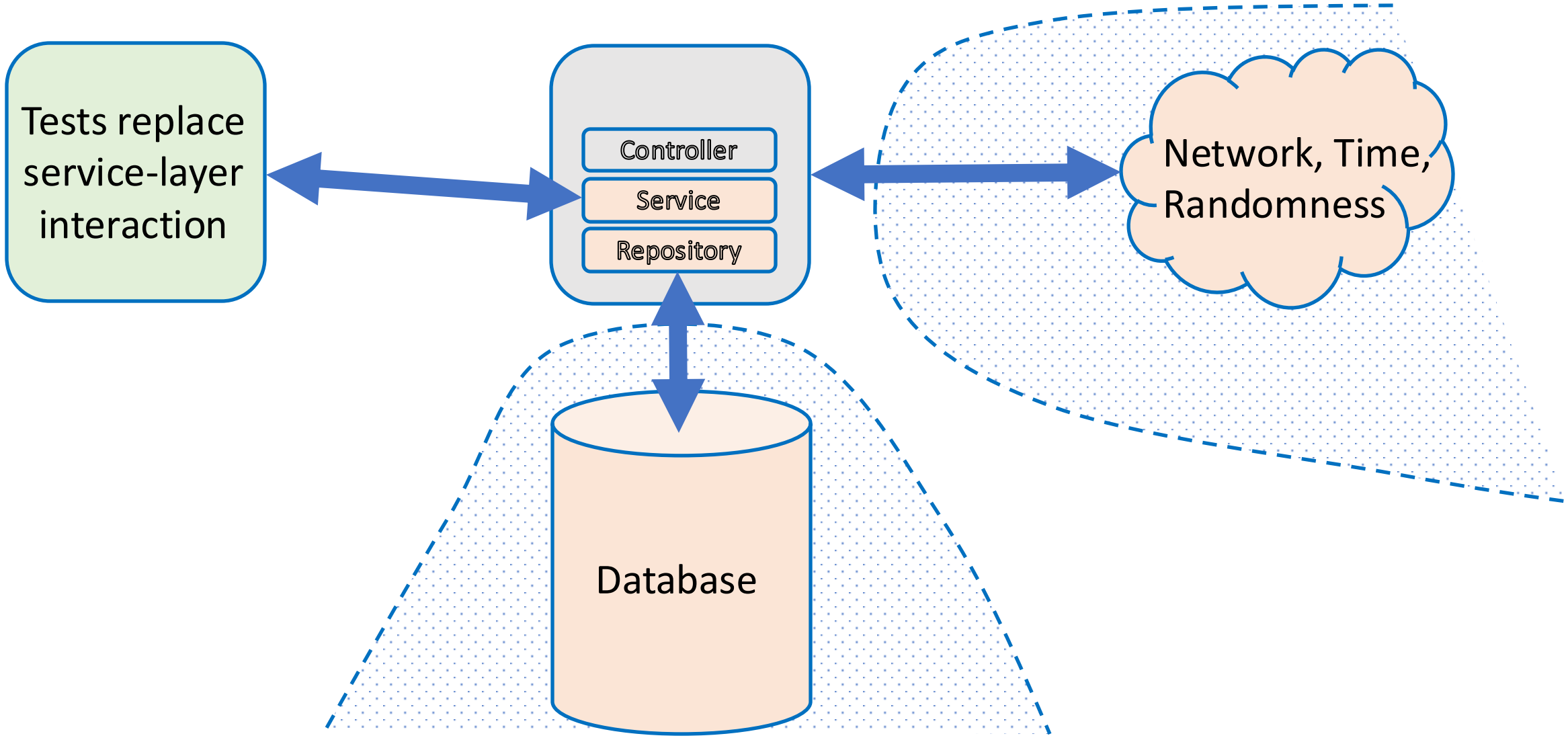
Test the appropriate connection points



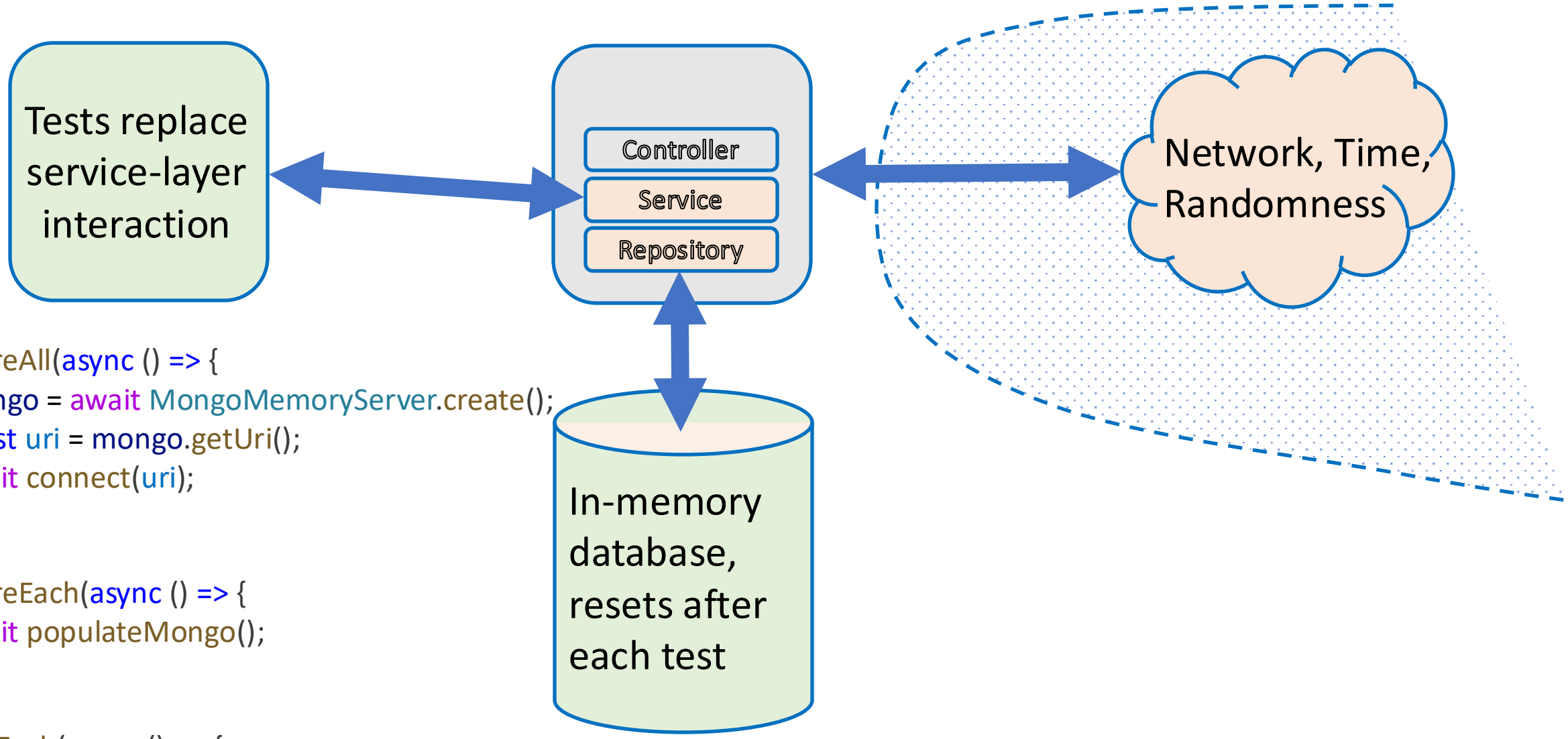

```
describe('enforceAuth', () => {
 it('should return a user and id on good auth', async () => {
  const user = await enforceAuth({ username: 'user1', password: 'pwd1' });
  expect(user).toStrictEqual({
   _id: expect.any(Types.ObjectId),
   username: 'user1',
  });
 });

 it('should raise on bad auth', async () => {
  await expect(
```

Test the appropriate connection points



Control easy-to-control parts of environment

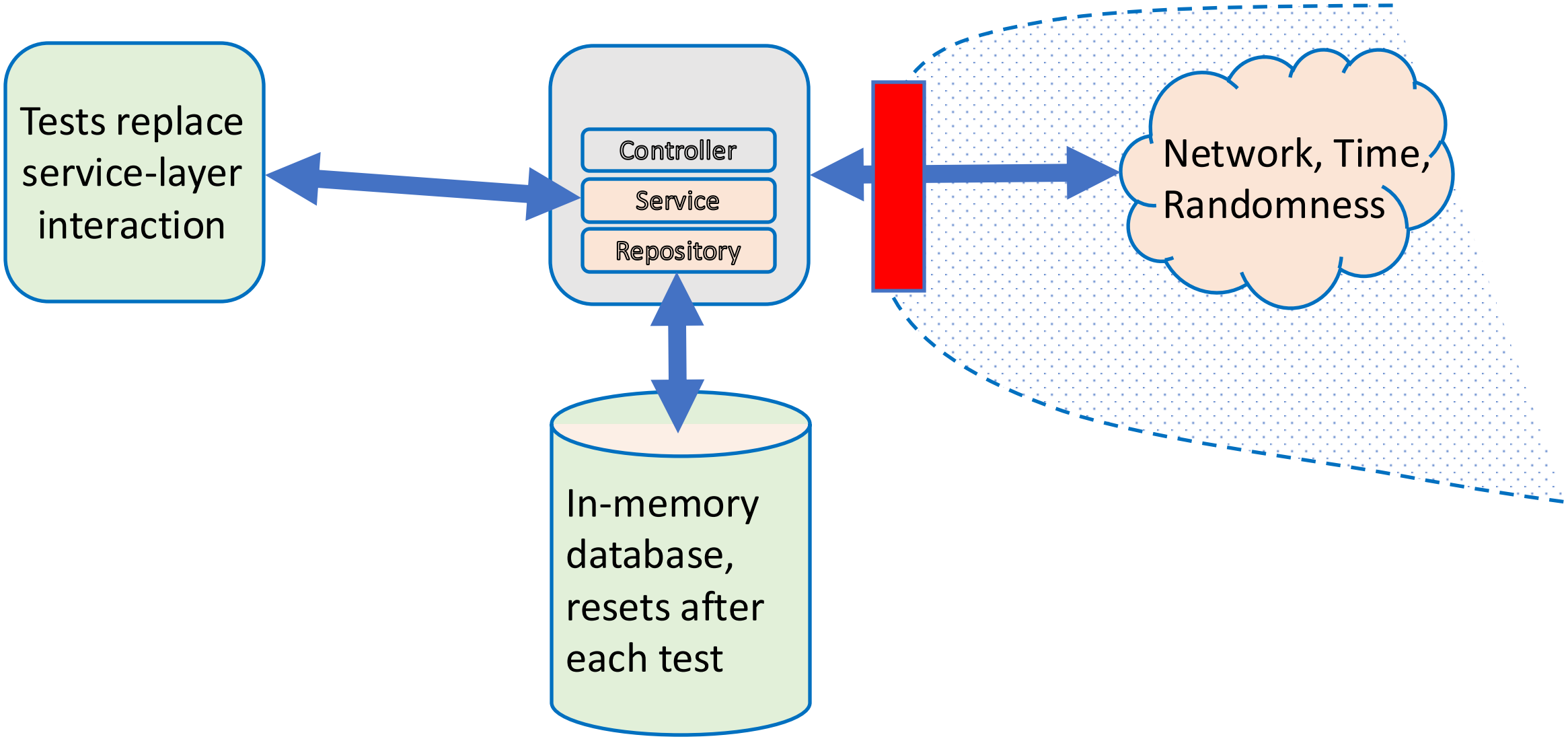



```
beforeAll(async () => {
  mongo = await MongoMemoryServer.create();
  const uri = mongo.getUri();
  await connect(uri);
});

beforeEach(async () => {
  await populateMongo();
});

afterEach(async () => {
```

Hijack hard-to-control parts of environment



Break for live coding

Test Doubles

- The in-memory Mongo database is a test double for a production database that doesn't die when you restart the server
- Supertest is a test double for HTTP server architecture
- Pre-determined coin flip is the test double for the random coin flip
- The spied-on coin flip is the test double for the not-spied-on coin flip, kinda?

- Terminology (mocks, spies, stubs, fakes, dummies) is an inconsistent mess, though many individuals have described consistent and useful categorizations

Test Doubles Have Weaknesses

- Some failures may occur purely at the integration between components:
  - The test may assume wrong behavior (wrongly encoded by mock)
  - Higher fidelity mocks can help, but still just a snapshot of the real world
- Test doubles can be brittle:
  - Spies expect a particular usage of the test double;
  - The test is "brittle" because it depends on internal behavior of SUT;
- Potential maintenance burden: as SUT evolves, mocks must evolve.

Did we correctly model the behavior of httpbin?

Not just its IO behavior, but also its dependencies

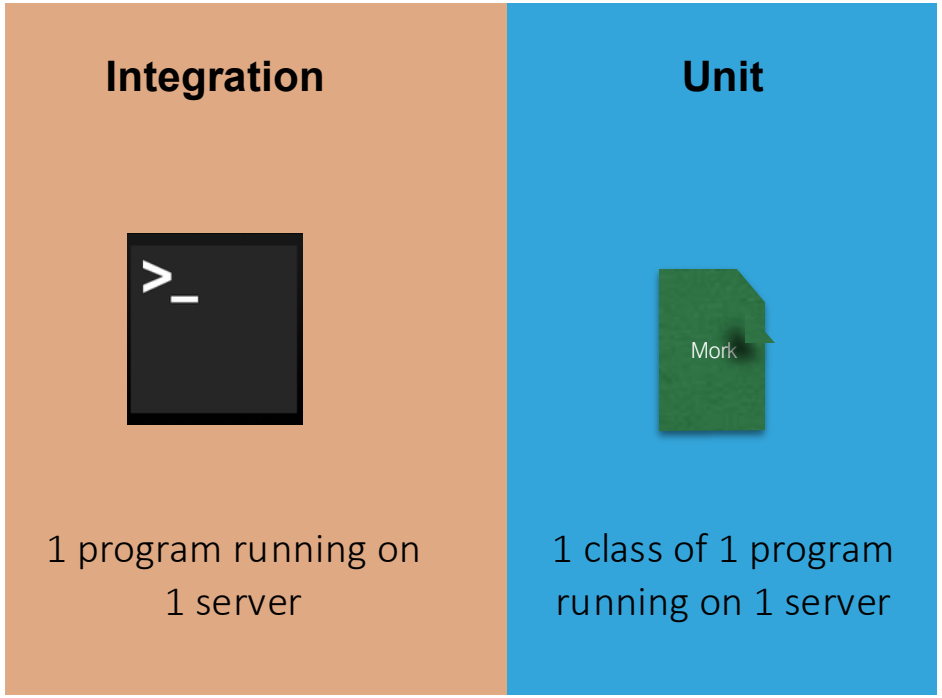
Break for more live coding

What’s the endgame here?

- We want to be able to get the system under test as small as possible
  - Fast (to write and to run and to understand)
  - Independent from other parts of the system (unit test failures pinpoint where the error is)
  - Can help improve coverage (but beware the code that’s only ever run in tests…)
- The endgame is *unit testing*

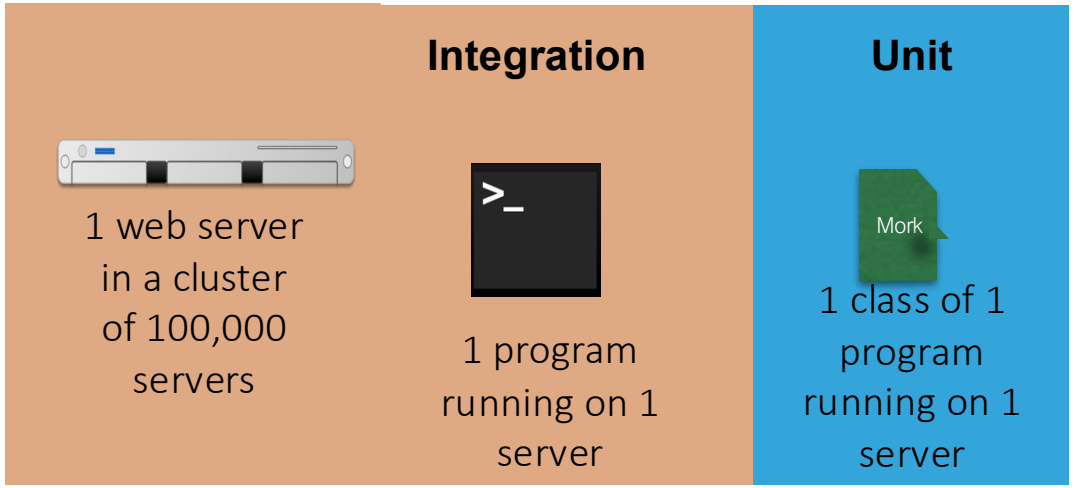
But some bugs are observable only when multiple components interact.

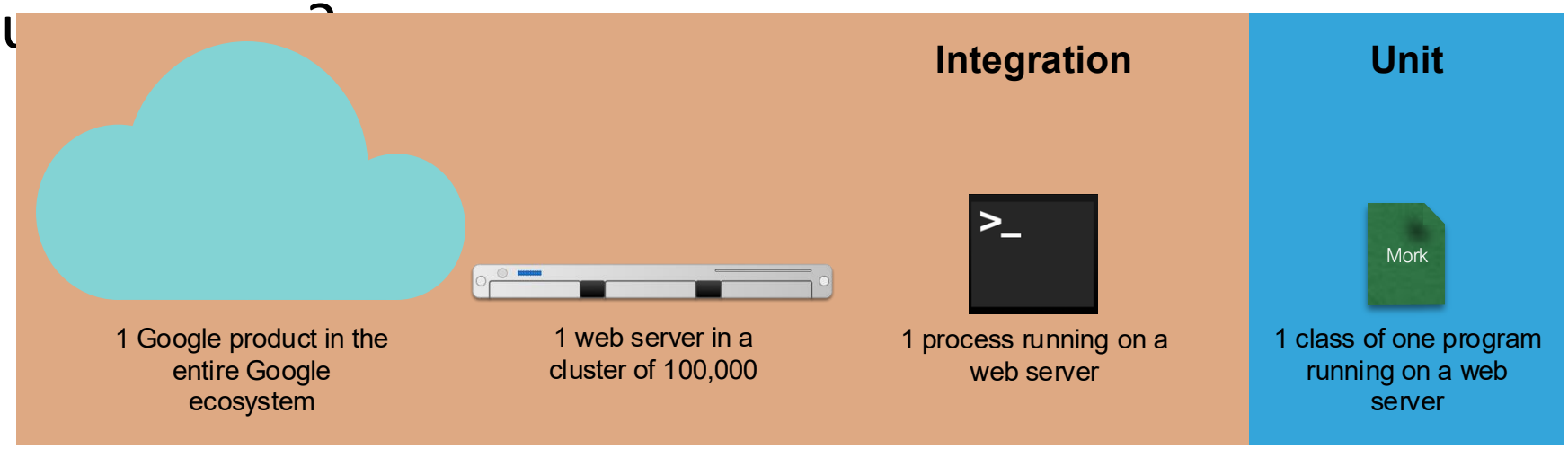
- These are usually because one module has made incorrect assumptions about some other module
- Unit tests won't reveal such bugs
- Mocks won't help, either (since they may incorporate our incorrect assumptions)
- So you really need *integration tests*



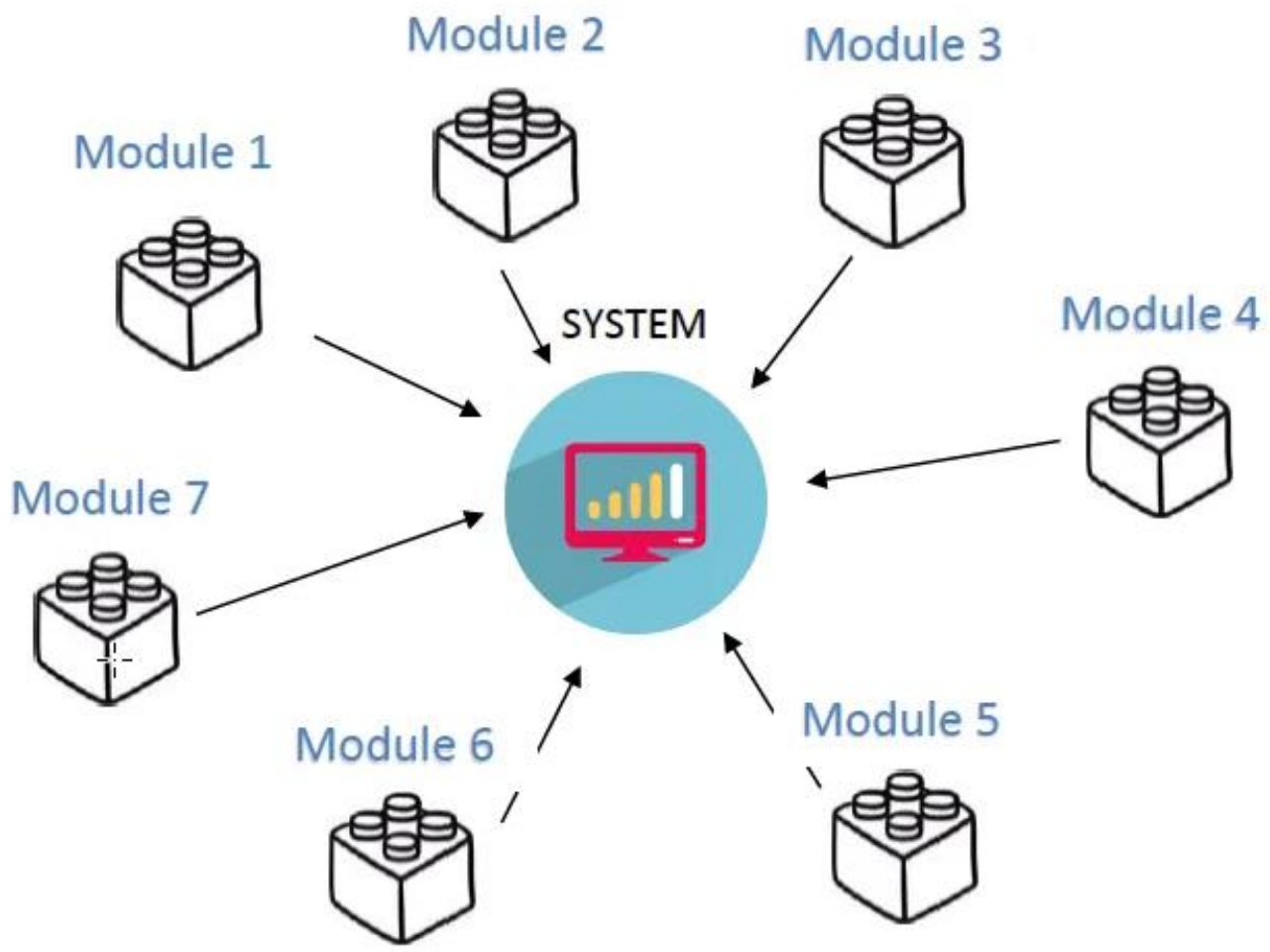
Integration tests may be larger, even enormous

- Does the presence of other jobs on our server change the behavior of our program?
- Does the presence of the other servers change the behavior of ou





Integration tests can be done in many ways



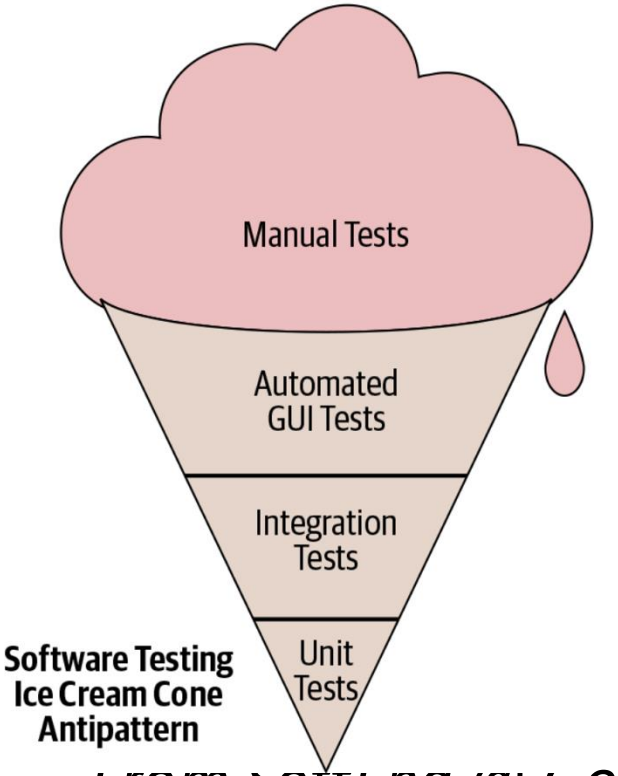
- All at once ("Big Bang")
- Top-down
- Bottom-up
- Middle-out
- Top-Bottom-Middle
- etc., etc., etc.

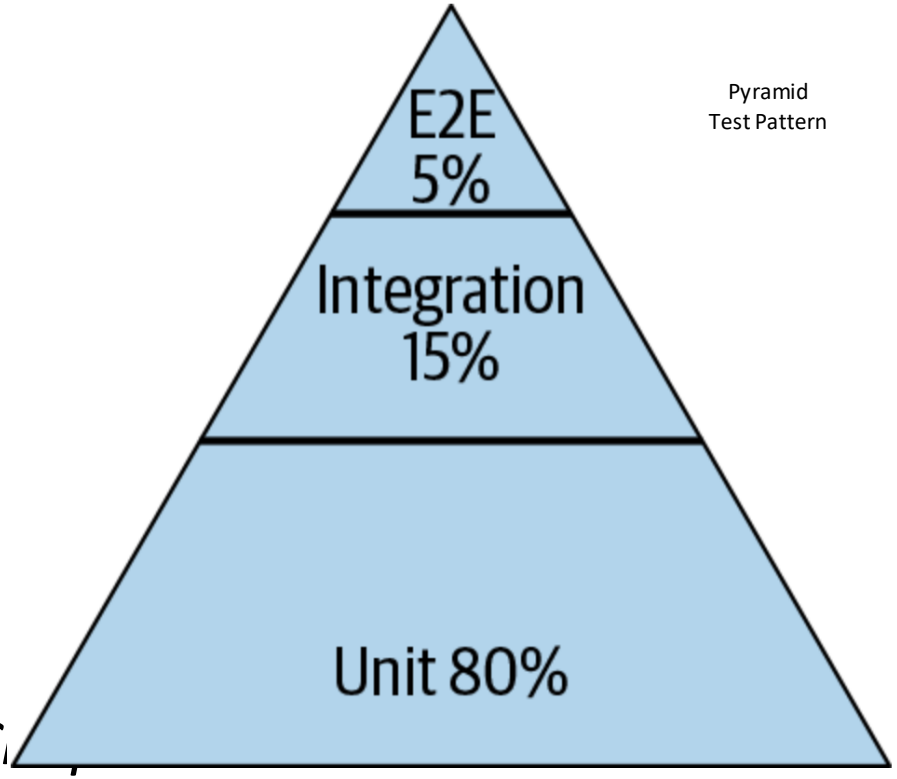
How big is my test? Google's Classification

- Small: run in a single thread, can't sleep, perform I/O or make blocking calls
- Medium: run on single computer, can use processes/threads, perform I/O, but only contact localhost
- Large: Everything else

"Software Engineering at Google: Lessons Learned from Programming Over Time," Wright, Winters and Manshreck, 2020 (O'Reilly)

Testing Distribution (How much of each kind of testing we should do?)



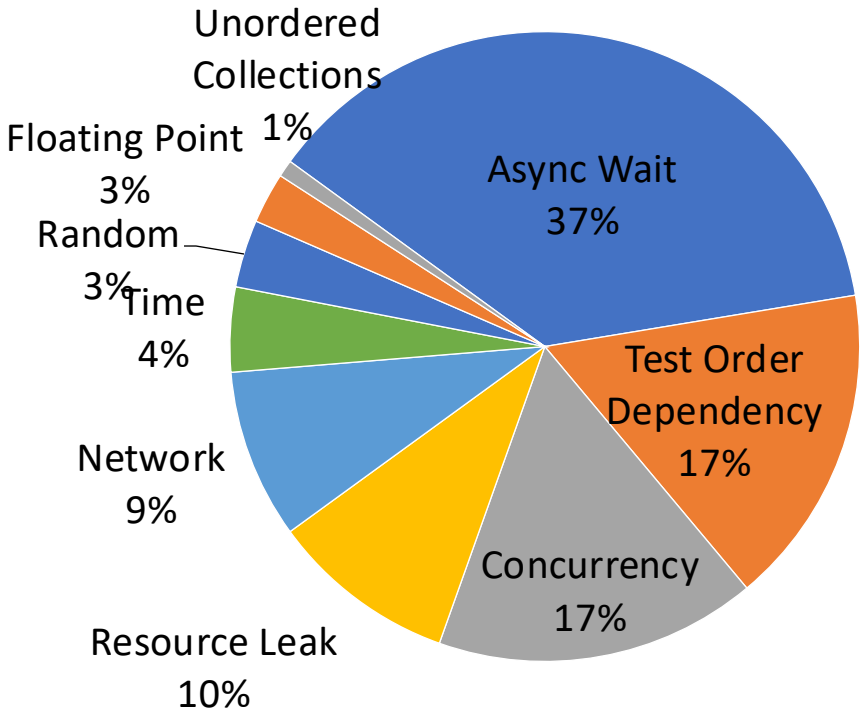


From SoftEng @ Google C

- https://learning.oreilly.com/library/view/software-engineering-at/9781492082781/ch11.html#testing_overview

Integration Tests can be Flaky

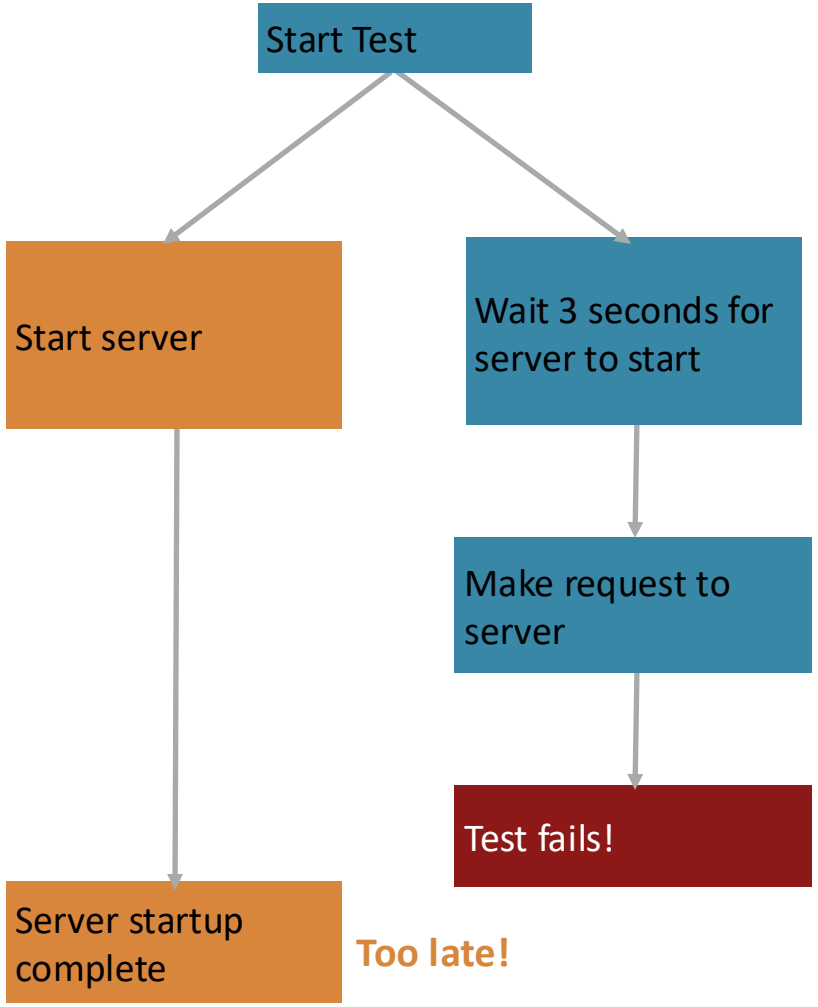
- Flaky test failures are false alarms
- Most common cause of flaky test failures: “async wait” - tests that expect some asynchronous action to occur within a timeout
- UI Testing is often flaky and slower
- Good tests avoid relying on timing
- Good tests avoid relying on the order in which the tests are run



[Luo et al, FSE 2014 “An empirical analysis of flaky tests”]

Flaky Test Example: Async/Wait

- Most common root cause of flakiness
- Difficult to avoid, but there are mitigations:
  - Have more “small” tests that don’t require concurrency
  - Ensure sufficient resources available for running tests
  - Embed reasonable error detection to classify test failures as likely to be “flaky” vs true failures



We make flaky tests anyway

```
- name: Test that the backend server starts
  run: |
    npm start -w=server & sleep 5
    echo "Checking if home page is served"
    curl --fail 'http://localhost:8000/' > /dev/null 2>&1

    echo "Checking if login page is served"
    curl --fail 'http://localhost:8000/login' > /dev/null 2>&1

    echo "Checking if api endpoint returns several threads"
    curl --fail 'http://localhost:8000/api/thread/list' 2> /dev/null | jq 'if length < 4 then error("Too few posts returned from ap
```