Testing Strategies

Software Engineering 2017
Alessio Gambi - Saarland University

Generic Testing Strategies

- Automate as much as possible
- Manual not necessary bad, automatic not necessary good
- Automate the “right” part of testing
  input generation, execution, checking, …

Automated Unit Test Generation

- Generate the code of the test, the input data, and possibly the behavior of the environment at once
  - Directed random generation
    each test build incrementally, Randoop
  - Evolutionary method
    whole test suite generation, Evosuite
- Combine coverage and other metrics (e.g., branch distance) to define a fitness function
- False alarms and “impossible” behaviors

Test Carving

- From the execution of system tests, generate a number of unit tests which execute only a subset of the observed method invocations
- Unit tests capture the same set of behaviors which are observed during the system tests

Fuzz Testing or Fuzzing

- Generate a large number of random inputs
- Mostly focused on security testing

  *Does system handle **unforeseen** inputs or crashes?*
- Generative or metamorphic
- Structured inputs might be problematic

Deep Fuzzing

- Generate a large number of random but **syntactically valid** inputs to reach deeper parts of the code
- Grammar based fuzzing: use a description of the input structure to guide the input generation

  Who provides the grammar? Can we learn it automatically?

Capture and Replay

- Record the (manual) interactions between the system and its environment
- Replay interactions during testing

Capture and Replay

- Record the (manual) interactions between the system and its environment
- Record the (manual) interactions between users and the GUI
- Replay interactions during regression testing

Dealing with environmental and other dependencies

Dealing with environmental and other dependencies

- The SUT interacts with external entities
  file system, database, remote services, …

- The code under test requires other code to be in place
  DB-bridge library, service API, …

- Avoid to directly interact with the actual entities
  brittle, dependent, and non-isolated tests

Dealing with environmental and other dependencies

How to achieve?

- Isolation
- Repeatability
- Reuse
- Separation of concerns
  setup/teardown vs test execution vs verification

Text Fixtures

Fixed states used as a baseline for running tests to ensure well known and fixed environment are setup up before the test executions

Text Fixtures

Fixed states used as a baseline for running tests to ensure well known and fixed environment are setup up before the test executions

Text Fixtures

Fixed states used as a baseline for running tests to ensure well known and fixed environment are setup up before the test executions

- Loading DB with known data
- Copying files to specific location
- Setup of test drivers, mocks, stubs, etc.

Testing using Doubles

To make a single unit work, you often need other units

- **Test Double.** Generic term for any kind of pretend object used in place of a real object for testing purposes

Gerard Meszaros

Testing using Doubles

To make a single unit work, you often need other units

- **Dummy objects**
  Object passed around but never used.
- **Fake objects**
  Working implementations with shortcuts.
- **Stubs**
  Provide canned answers to calls during tests.
- **Mocks**
  Objects pre-programmed with expectations on calls.

Martin Fowler

State and Behavior Verification

- State verification examines the **state** of the SUT and its collaborators **after** the method was exercised.
- Behavior verification examines the **interactions** of the SUT with its collaborators **during** the method execution.

Mocks vs Stubs

- Mocks always use **behavior verification**, Stubs can use also **state verification**.
- Setting up stubs includes defining canned responses.
- Setting up mocks also includes expressing **expectations**.

Test with a Database

Stub	Mock
In-memory structure for storing records	Define data which will be written in the DB but not the logic to do it
SUT reads and writes records to the stub DB	SUT invokes mock, but no data are stored
Check state of DB and behavior of objects not related to the DB	Check what was written to DB and how values ended up there

The Test Oracle Problem

Given an input for a system, the challenge of distinguishing the corresponding desired, correct behavior from potentially incorrect behavior

The Test Oracle Problem

Given an input for a system, the challenge of distinguishing the corresponding desired, correct behavior from potentially incorrect behavior

- Test preconditions
  specify the state before the test

- Test execution
  specifies what to do during the test

- Test postconditions (oracle)
  specify the expected results after the test

The Test Oracle Problem

Given an input for a system, the challenge of distinguishing the corresponding desired, correct behavior from potentially incorrect behavior

- Oracles are **necessarily incomplete**
  observable behavior, miss and false alarms

- Oracles are **heuristics** help testers to decide
  might point to wrong decision

Software Test Oracles

- Not always possible to express the oracles
  non-testable programs

- Not always can be/need to be precise
  float/doubles, asynchronous interaction, …

- Not always possible to capture all the facets of "correct" behavior
  result is correct but it took hours to compute

Types of Oracles

- Constraint oracle
  test values or relationships
- Regression oracle
  check results of current tests vs results of previous tests
- Self-verifying data as an oracle
  embed the correct answer in the test data
- Calculation oracles
  check the calculations of a program using other programs
- Inverse oracle
  Apply the inverse function, check if result is same (neg)

Model as Oracles

- Physical model
  test a software simulation of a physical process

- Statistical model
  unlikely (yet formally correct) behaviors

- State model
  what the program does for each input in each state

- Interaction model
  how the SUT and the other programs behave in response to each other

Model as Oracles

- Physical model
  test a software simulation of a physical process

- Statistical model
  unlikely (yet formally correct) behaviors

- State model
  what the program does for each input in each state

- Interaction model
  how the SUT and the other programs behave in response to each other

Do not forget that models may be wrong !

The Test Oracle Problem

- Test with **several** oracles at once
- No matter what combination is used, some errors will escape anyway

Anything else to test for ?

What's beyond functional correctness and coverage?

Anything else to test for ?

What's beyond functional correctness and coverage?

- Load testing
- Performance testing
- Endurance testing
- Stress testing
- User Experience testing

Anything else to test for ?

What's beyond functional correctness and coverage?

- Load testing
- Performance testing
- Endurance testing
- Stress testing
- User Experience testing
- Acceptance testing
- Security testing
- Penetration testing
- A/B testing
- …

Specific Strategies. Why ?

- Testing different properties requires different approaches to testing

  adequacy criteria, test generation and execution, scaffolding, …

Specific Strategies. Why ?

- Testing different properties requires different approaches to testing
  adequacy criteria, test generation and execution, scaffolding, …

- Different systems work in different contexts and have peculiar properties that must be taken into account during testing
  environment, distribution, limited resources, interactivity, …

Testing
Mobile Applications

Challenges

- Limited memory/computational resources
- Hardware and software fragmentation
- Multiple network connectivity and carriers
- Unforgiving users
- Native, web-based, hybrid applications

Main (Many) Strategies

- Functional Testing

  functionalities of application as per requirement specification

- Performance Testing

  client application, server, and network performance

- Memory Testing/Load Testing

  optimized (limited) memory usage by an application

- Security Testing

  secure connections, encrypted data, data leakage

Main (Many) Strategies

- Usability Testing, User eXperience Testing, and User Acceptance Testing
- Interruption Testing

  incoming call, SMS, low memory/battery warning, …
- Installation Testing

  installation process including update and uninstall
- Carrier and Network Testing

  WiFI, 3G, 4G how switching network impacts on the application

Functional Testing

- Check the flow of actions
- Mostly at the level of user interface
- Tests can be local or instrumented
  developer runtime, emulator, real devices, cloud
- Combination of manual and automated tests
  capture&replay, monkey testing, …

Memory/Load Testing

Devices have limited memory and OS stop applications when they consume too much memory

- Memory leakage testing
  performance slow down while using the app

- Find the breaking point of the application
  how many items in the cart make my app unresponsive?

Security Testing

- Data flow vulnerability
  - Check data are encrypted and channels are secure
  - Do not save clear private data on client’s side
- Data leakage
  - Check log files

Usability, User eXperience and User Acceptance Testing

- Entirely manual
- Alpha Testing: select a focused group of user
- Beta Testing: select a larger group of users with different devices/connections/location
- Crowd-sourcing: very large sets of unknown, and possibly untrustworthy, users

Interruption Testing

Incoming notifications, text and calls, and memory/ battery warning interrupt the operation of the app

- Go smoothly in the suspended state and restart afterwards
- Check that after the interruption the application does not end up in "frustrating" the users
  game not saved is lost, apps resume on home screen

Carriers & Network Connectivity Testing

- Do not test only on WiFi connection
- Different carriers have different performances and features/standards
- Test for lost connectivity
- Test for connectivity downgrade
  4G to 1G, 1G to Edge, …

Tools

there are many, many more

- Monkey, DroidMATE: generate pseudo-random streams of user events and system-level events
- UIAutomator: JavaScript to define tests which consists of complex user actions
- Robotium, Appium: support native and hybrid automated black-box tests
- MonkeyTalk: record and playback
- Screenfly: emulators to test different screen sizes and device profiles

Testing Web Applications

Challenges

- Client/Server architecture
  - Part of the app runs in the browser (e.g., JS)
  - Part of the app runs in the server (CGI, DB, business logic)
- Huge variability in browsers, versions, connection speeds, standard protocols, devices
- Open World (intranet vs internet audience)

Challenges

- Fast rollout and updates
- Standard or specific requirements for appearance, graphics, or user interface
- Links between the pages, navigation
- Page content, consistent layout

Main (Many) Strategies

- Functionality testing
- Interface testing
- Database testing
- Performance testing

Main (Many) Strategies

- Functionality testing
- Interface testing
- Database testing
- Performance testing
- Security testing
- Usability testing
- Cookies testing
- Compatibility testing
- ...

Functionality Testing

- Check all the **links**
  - Outgoing and internal links must be valid (non broken)
  - Orphan pages or dead-end navigation
  - Back-button
- Test all the **forms**
  - Fields validation, default values, wrong inputs
  - Optional elements, and modification to form

Interface Testing

- Check the interactions between, client and Web server, between Web server and application server, and between application server and database

- Handle errors gracefully and display proper messages to users

- Reset connections and interrupts running transactions

Database Testing

- Check data consistency, integrity and errors for every DB related functionality

  edit, delete, modify

- Check all queries to database are sanitized and correctly executed

- Check that data is retrieved and updated correctly

Performance/Load Testing

- Subject the application to heavy load
  - Define user model and runs as many concurrent users as possible
  - Use small and large input data
  - Target specific pages (much like DoS)
- Stress testing: go beyond know system limits and break it; then, check if the system recovers.

Cookies Testing

Cookies are small files stored on the user machine which enable server side user session and might contains private data

- Test if the application crashes if cookies are enabled/disabled
- Check if data are encrypted and cookies expire after sessions end
- “Fuzz” cookies (alter their content) or delete them

Compatibility Testing

- Cross-platform, cross-browser, and mobile
- CSS Style and AJAX/JS
  layout and interactions
- Security and validation
- Some operations (graphics) might not be available to OS

Tools and Services

- Many, diverse, and with tons of features

  Load and Performance Test, Page Speed, Mobile Web/App, Link Checkers, HTML Validators, Web Accessibility, Web Services, Cross-Browser, Web Functional/Regression Test, Web Site Security, …

- Start with the most widely known: Selenium, JMeter

- Focus on the essential and what you need

- Outsource to the cloud (compatibility)

http://www.softwaretestinghelp.com/most-popular-web-application-testing-tools/

Testing
Distributed Applications

Challenges

- Think of the problems and issues of testing normal applications and …

Challenges

- Think of the problems and issues of testing normal applications and …

- … multiply them by multiple processes written in multiple languages running on multiple boxes that could potentially all be on different OS

End-to-End Testing

In a distributed application, data propagate to multiple parts. Hence, testing only a single system at the time it not enough.

- Order and timing of data arrival can cause bugs
  write after read, read after write, write over write
- Huge number of possible interleaving of operations
  difficult to predict and test

Simple Distributed Systems

- The number of components is small
- Run all the components locally
  bare metal or inside virtual machines
- Better control over component lifecycle
- No network related issues and delays

Complex Distributed Systems

- Many components, impossible to test all the configurations

- Test using a “good approximation” of production system
  vary the number of nodes

- Mimic any system that is not available during testing
  simulators, stubs, or doubles

- Generate positive and negative scenarios
  hard to enumerate and foreseen

On Simulators/Stubs

- Valuable tools for development and debugging, but …
  - are time consuming to develop
  - do not behave like the actual systems

Distributed Data Systems

Guarantee eventual consistency by means of data replication and distribution

- Asynchronous data delivery
- Nodes failure and recovery

Asynchronous Data Delivery

- Data is delivered to any the entry node which replicates that data asynchronously to all other nodes.

- The entry node does not notify that the data has been successfully replicated.

- Testing needs to verify that data reaches the entry node and all the other nodes. Therefore, tests need to pull the data from each node where the data supposedly lives.

Tests Asynchronous Data Delivery

- Change the SUT and introduce a notification when data is propagated.
  - Guarantees on data delivery but changes the implementation
  - Tests might not be representative

Tests Asynchronous Data Delivery (2)

- Introduce artificial delays
    - No guarantees on data delivery but highly changes it will complete
    - Brittle tests: outcome of the tests is not predictable and depends on the environment the tests are executed (load, network speed, …)

Tests Asynchronous Data Delivery (3)

- Introduce **more** and **longer** artificial delays
    - Overall execution become exponentially longer
    - It worsens as more tests are defined

Tests Asynchronous Data Delivery (4)

- Introduce **more** and **longer** artificial delays **but** parallelize the tests execution
  - Shorter time, but most likely more problems
  - Problems appear in the test suite itself

Asynchronous Data Delivery

- No ideal and “one-fits-all” solutions for testing
- Tailor testing to the specific requirements of the SUT
- Go for an hybrid solution that combines the different delays, small/large tests, and parallelization

Node Failure

- A component that fail must not cause the entire system to fail, and ideally it should be invisible to the users.

- For example, data should be retrieved by a replica if the entry node is down. Data should propagate if a replica node is down and when it comes back synchronized.

Test Node Failure

- Setup and run the system

  store data in the system

- Kill a node (take the node offline)

  which node? check that it is actually down

- Check data can be correctly retrieved

  is the data up-to-date or the “best version” (all the data)
  retrieving data from the broken node must fail (not hang)

- Restore the node (take the node online)

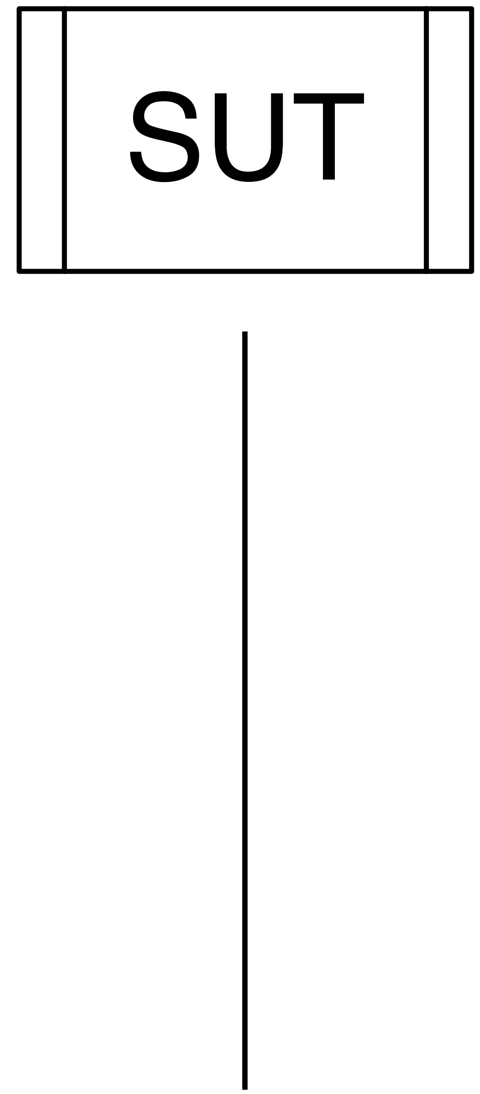
- Check data are synchronized (reconstruct the node)

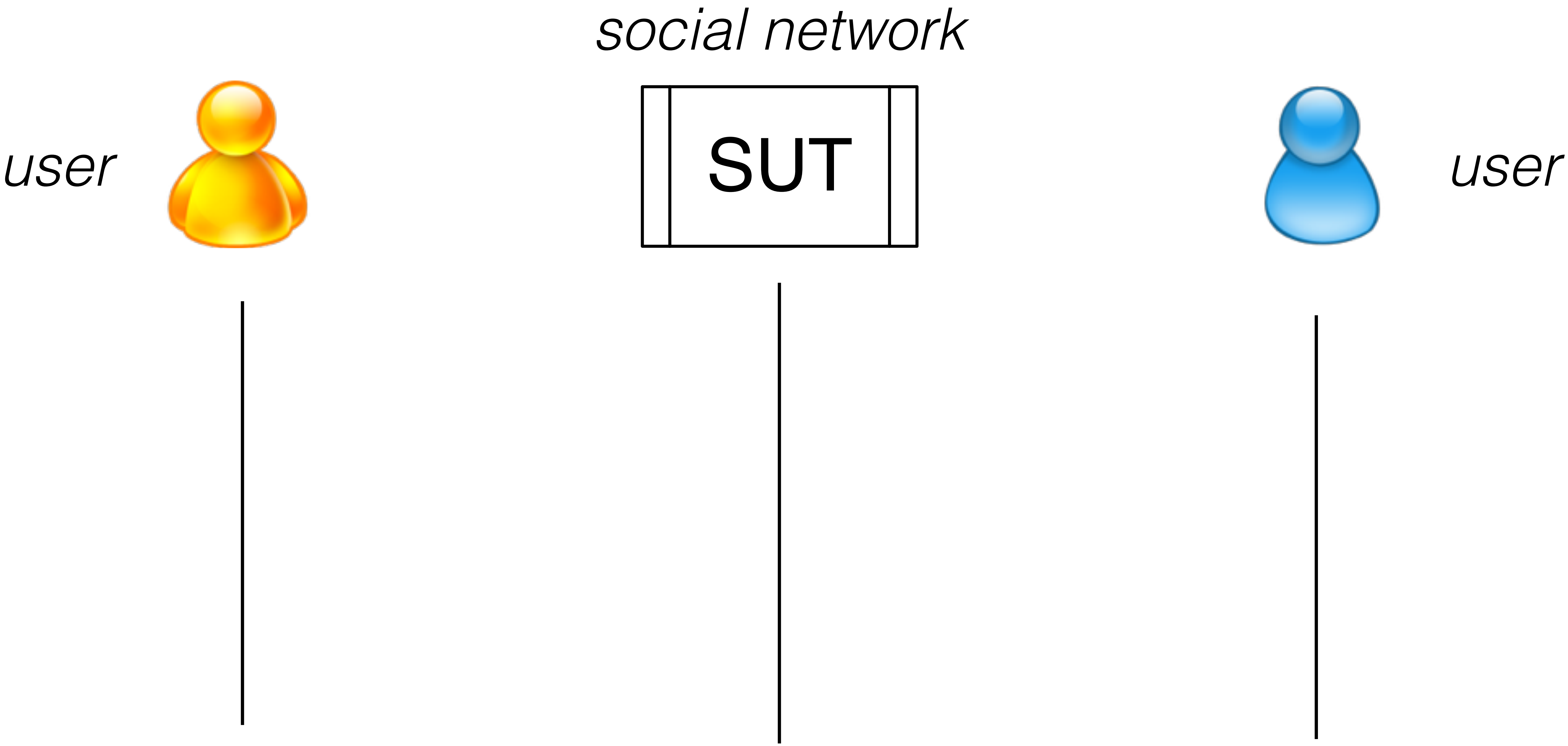
  requires the ability to pull data from that particular node

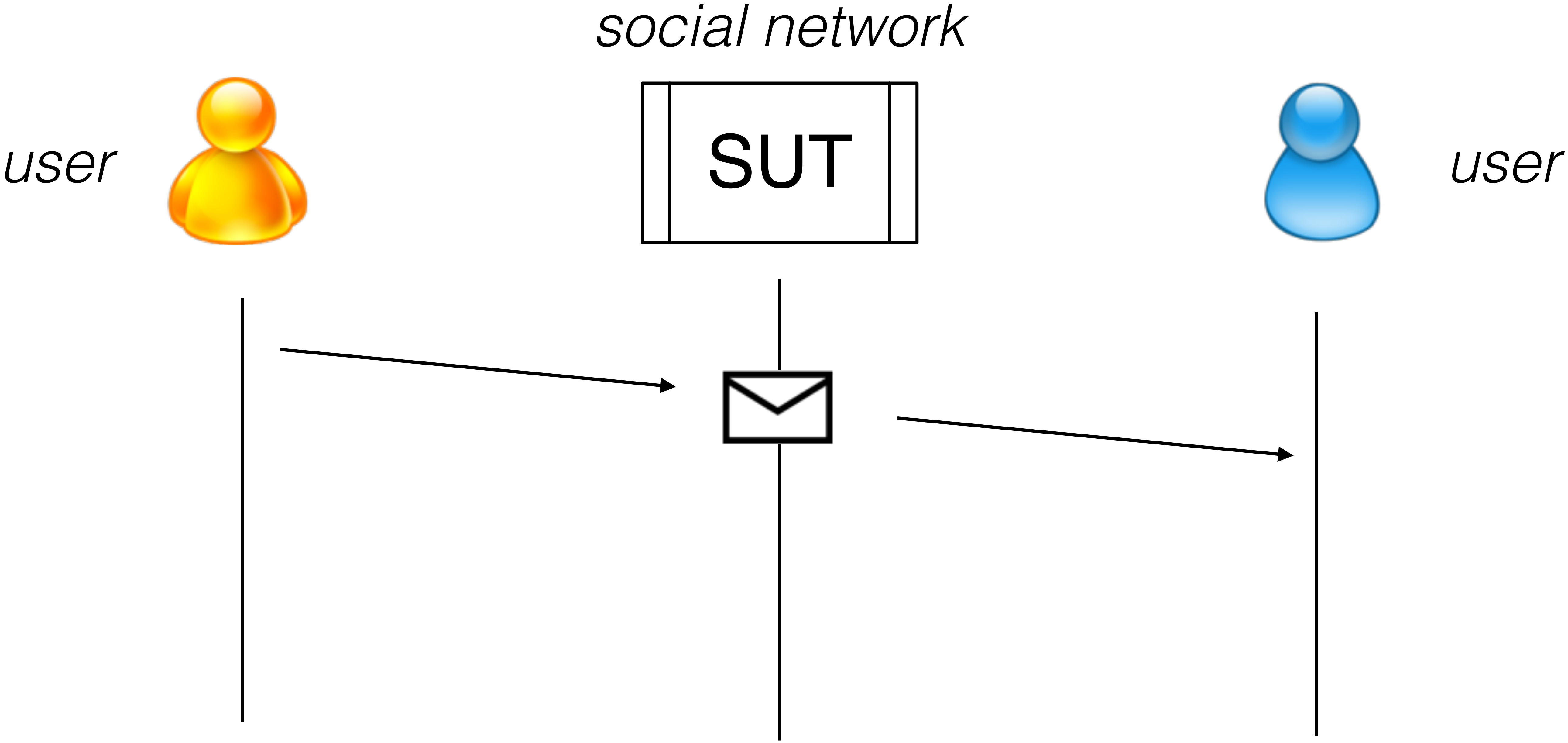
Model-Based Testing

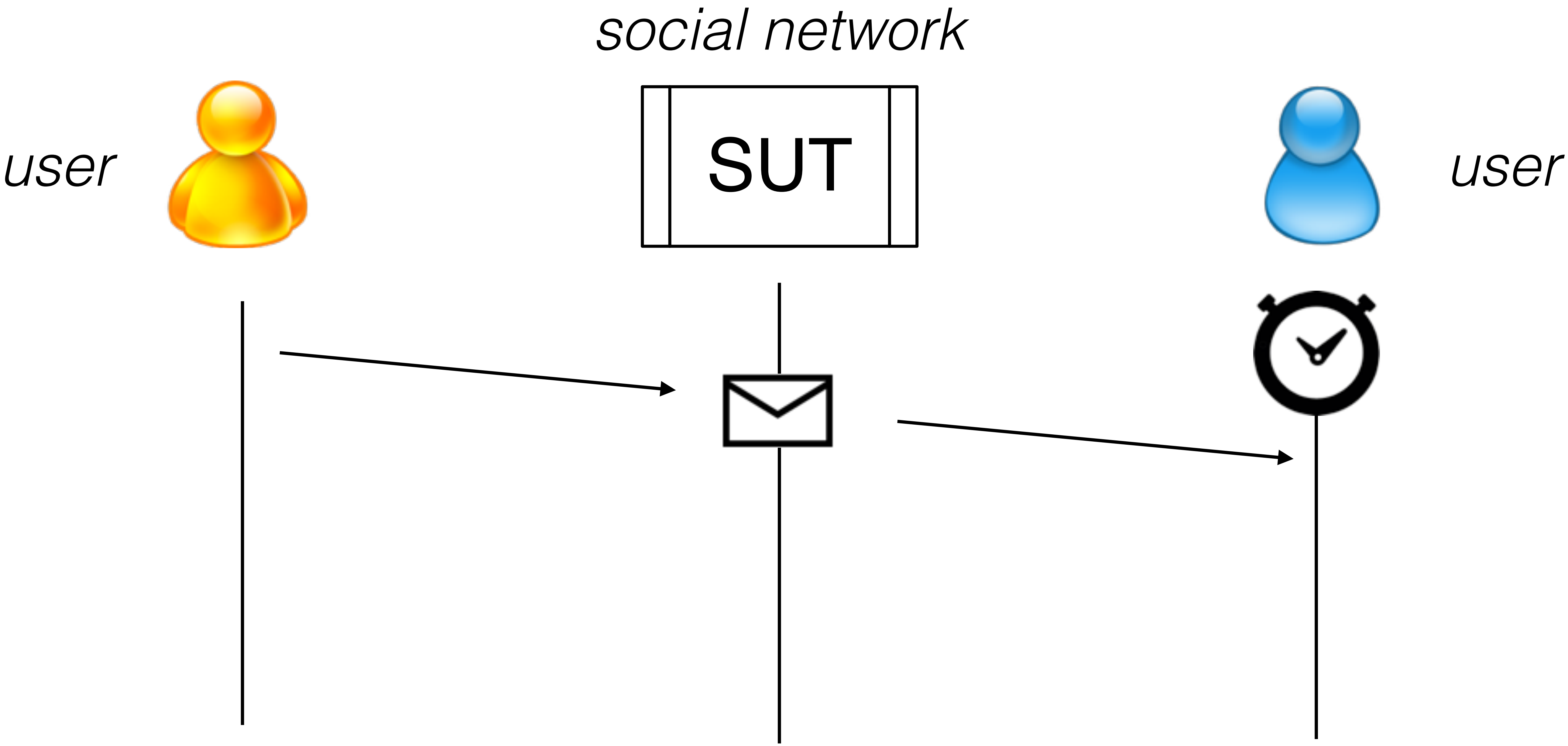
- Design the code to run in simulated environments and deterministic execution order
- Add the ability to programmatically inject faults and perform sanity checks
- Encapsulate the correct behavior in a model and design tests on the model
  state searches, pseudo-random activity, and timing variations

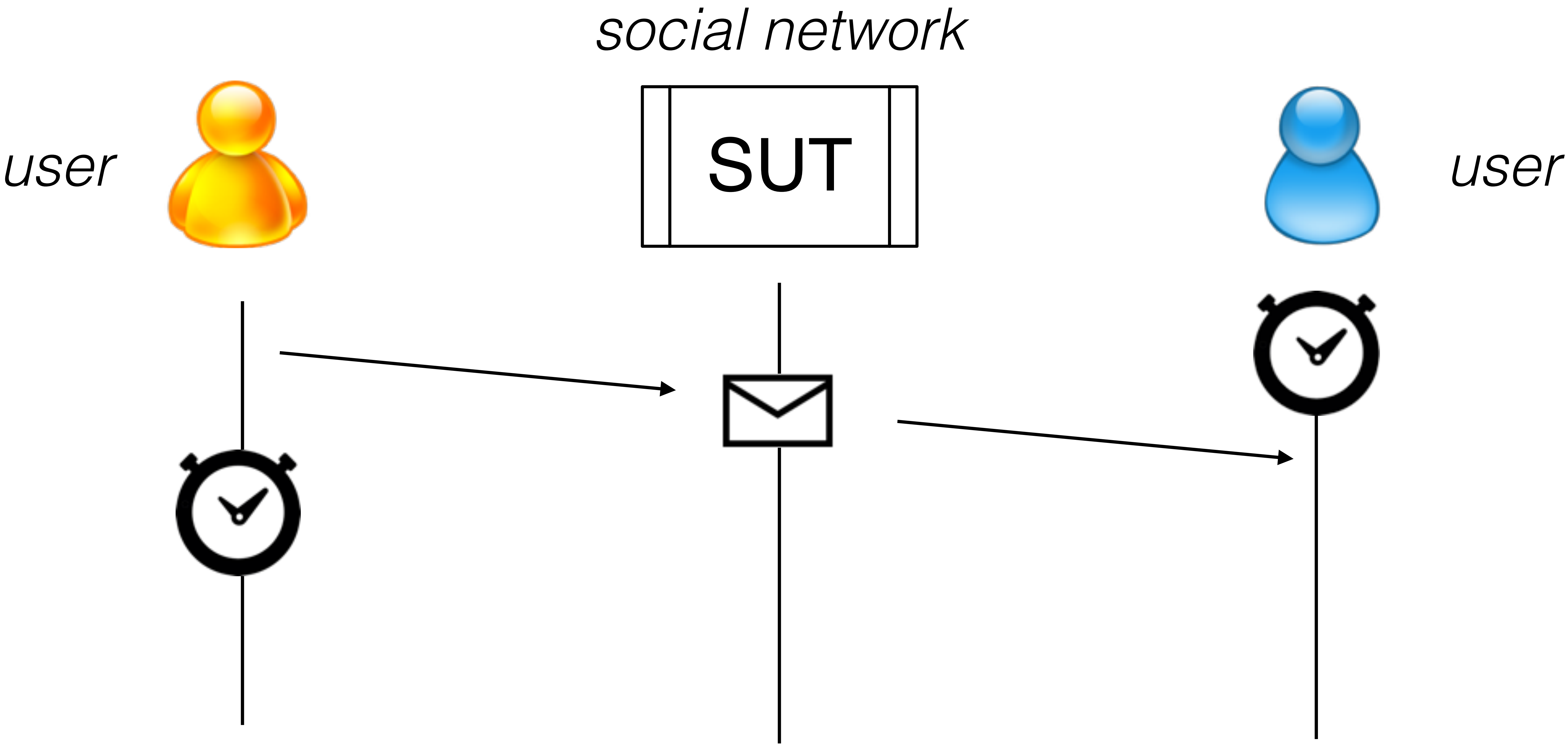
social network

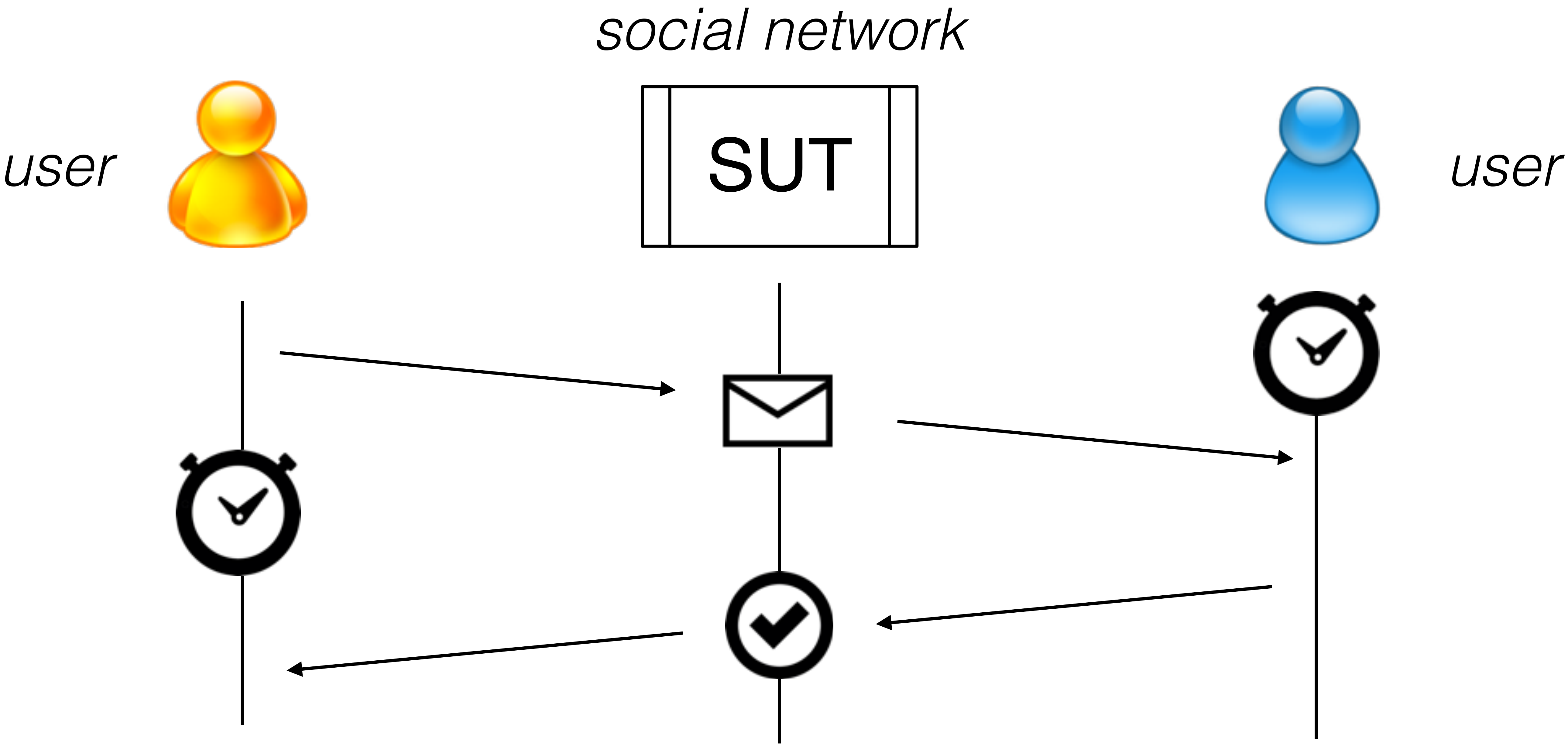



social network
user
SUT
user

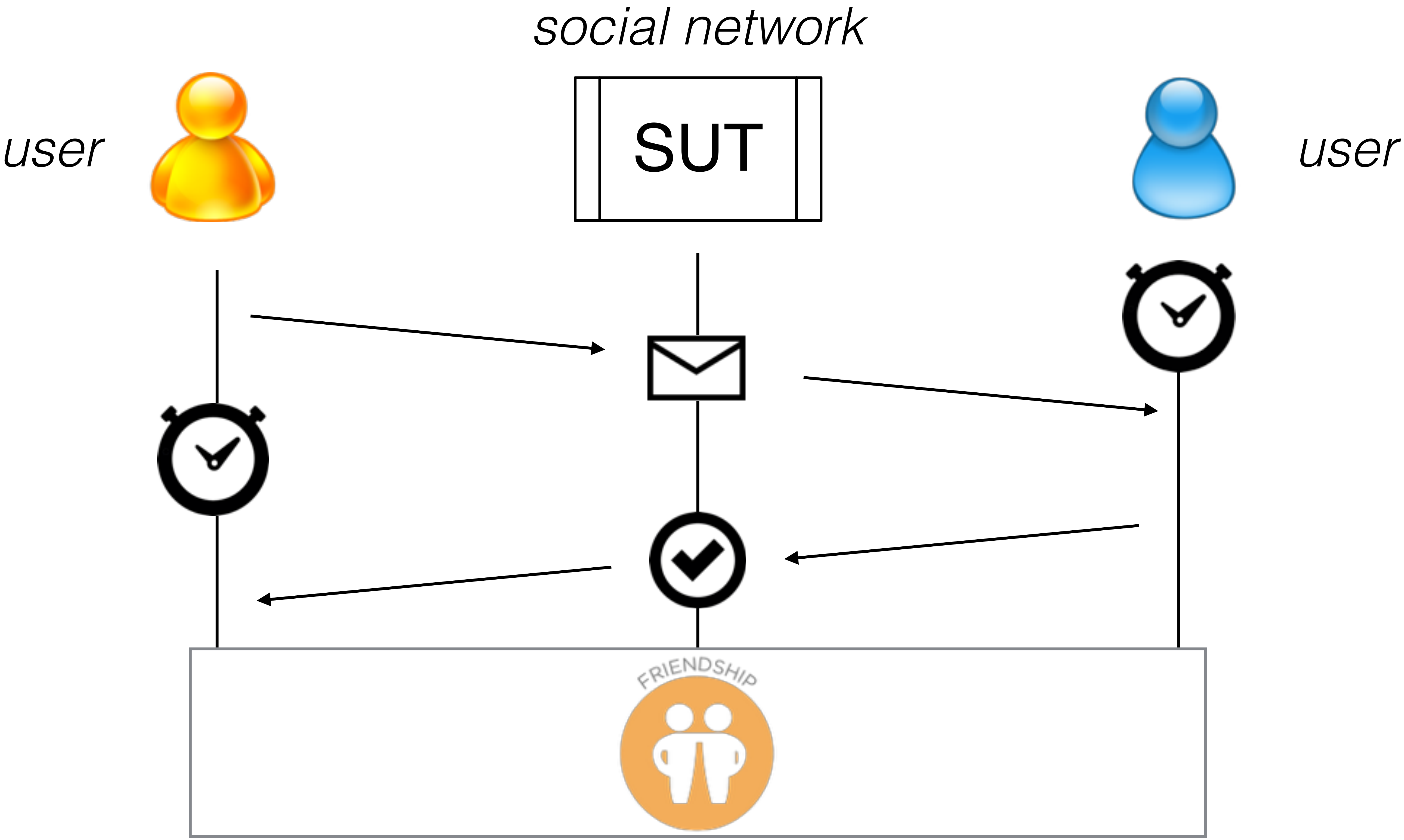

social network
user
SUT
user


social network
user
SUT
user

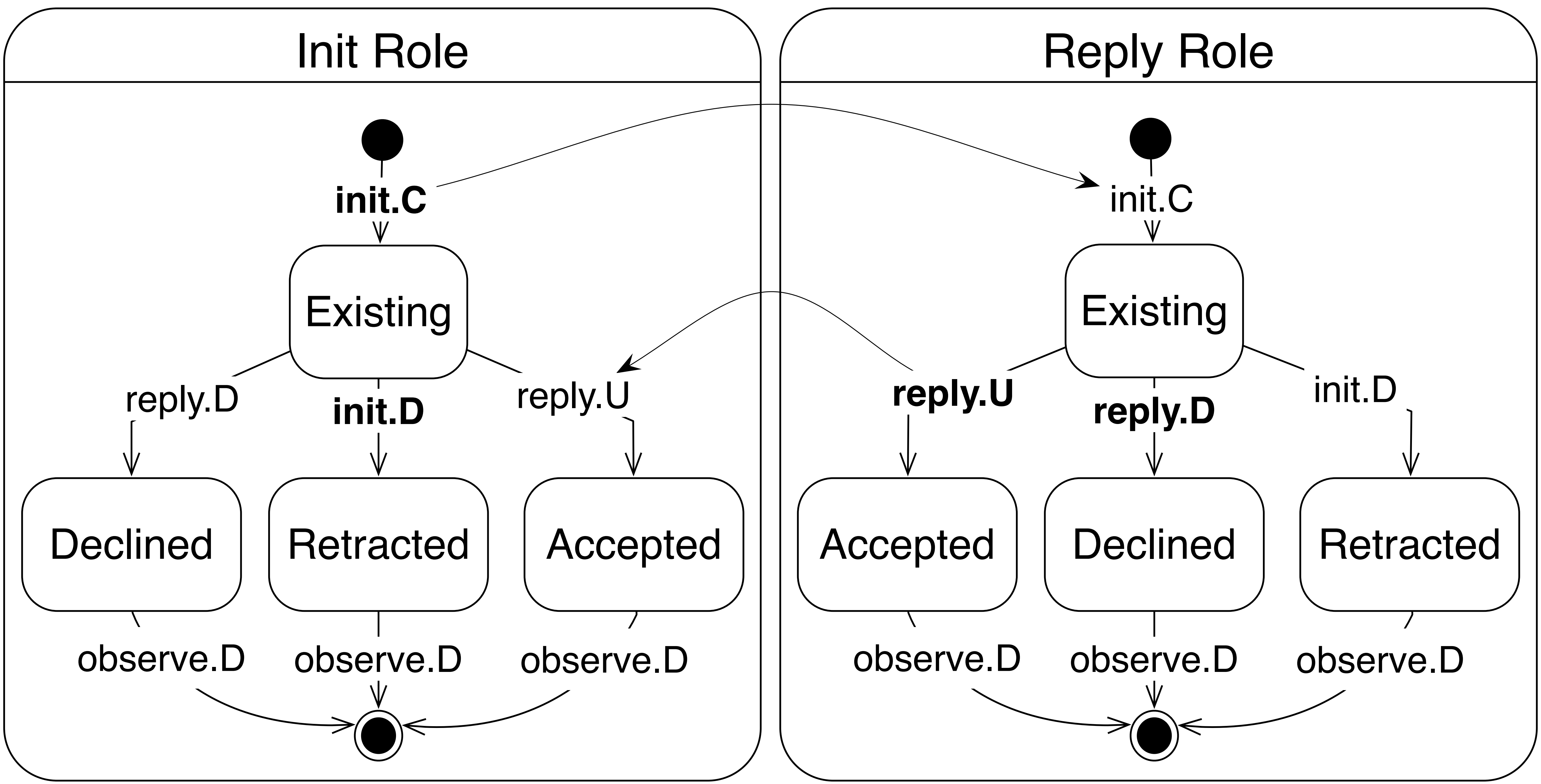

social network
user
SUT
user



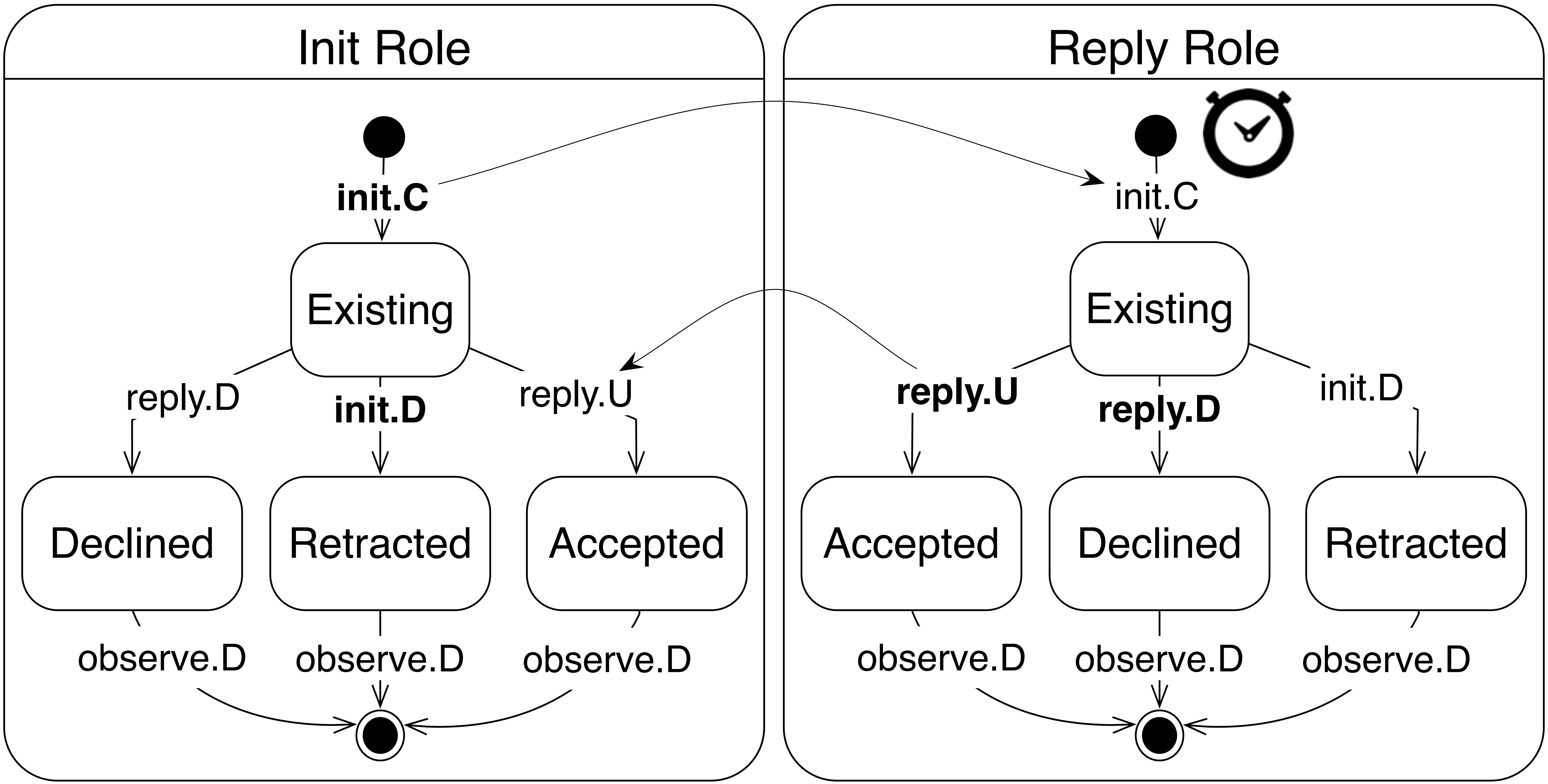
social network
user
SUT
user


social network
SUT
user
user
FRIENDSHIP

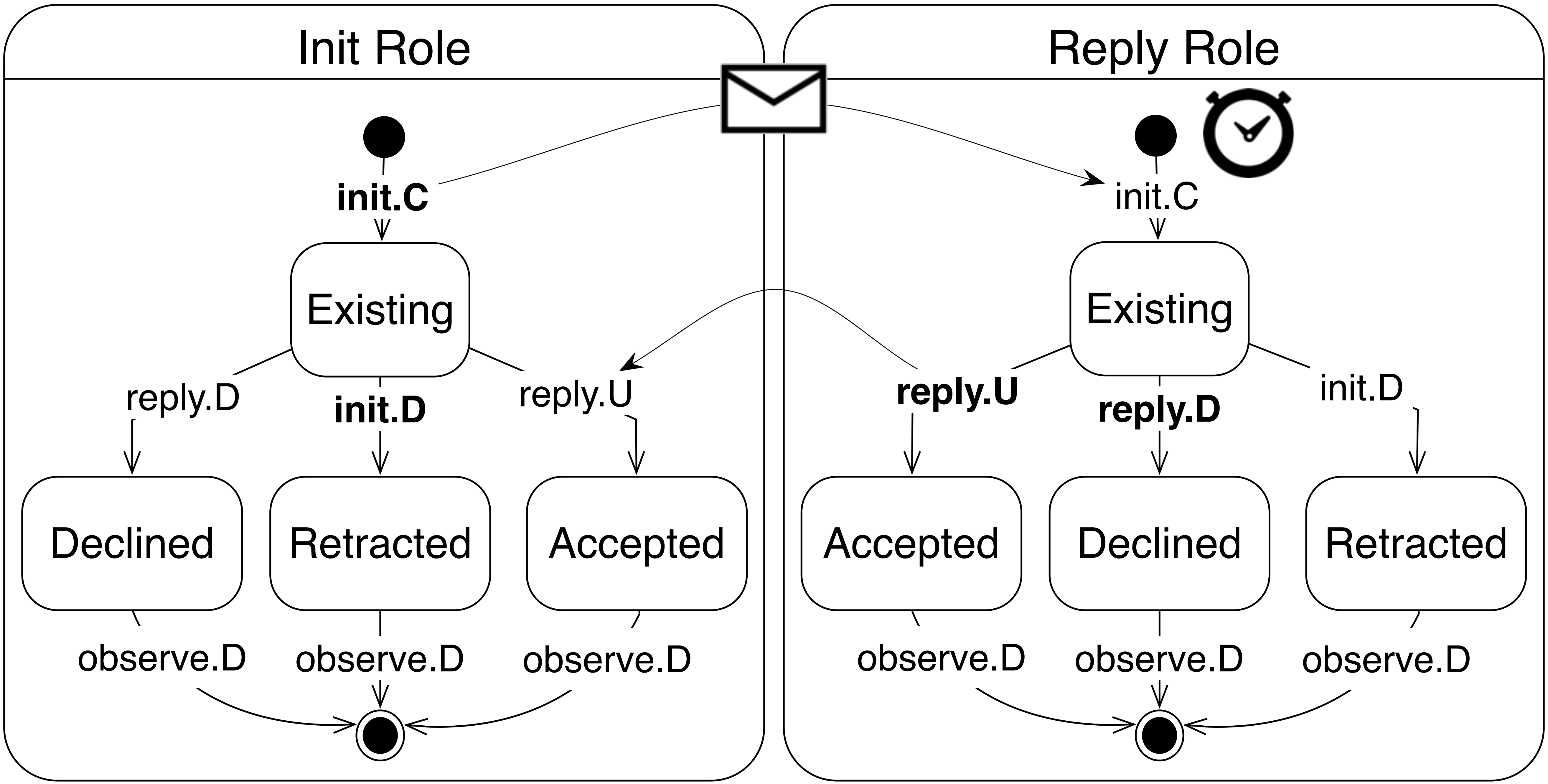
Synchronized FSMs



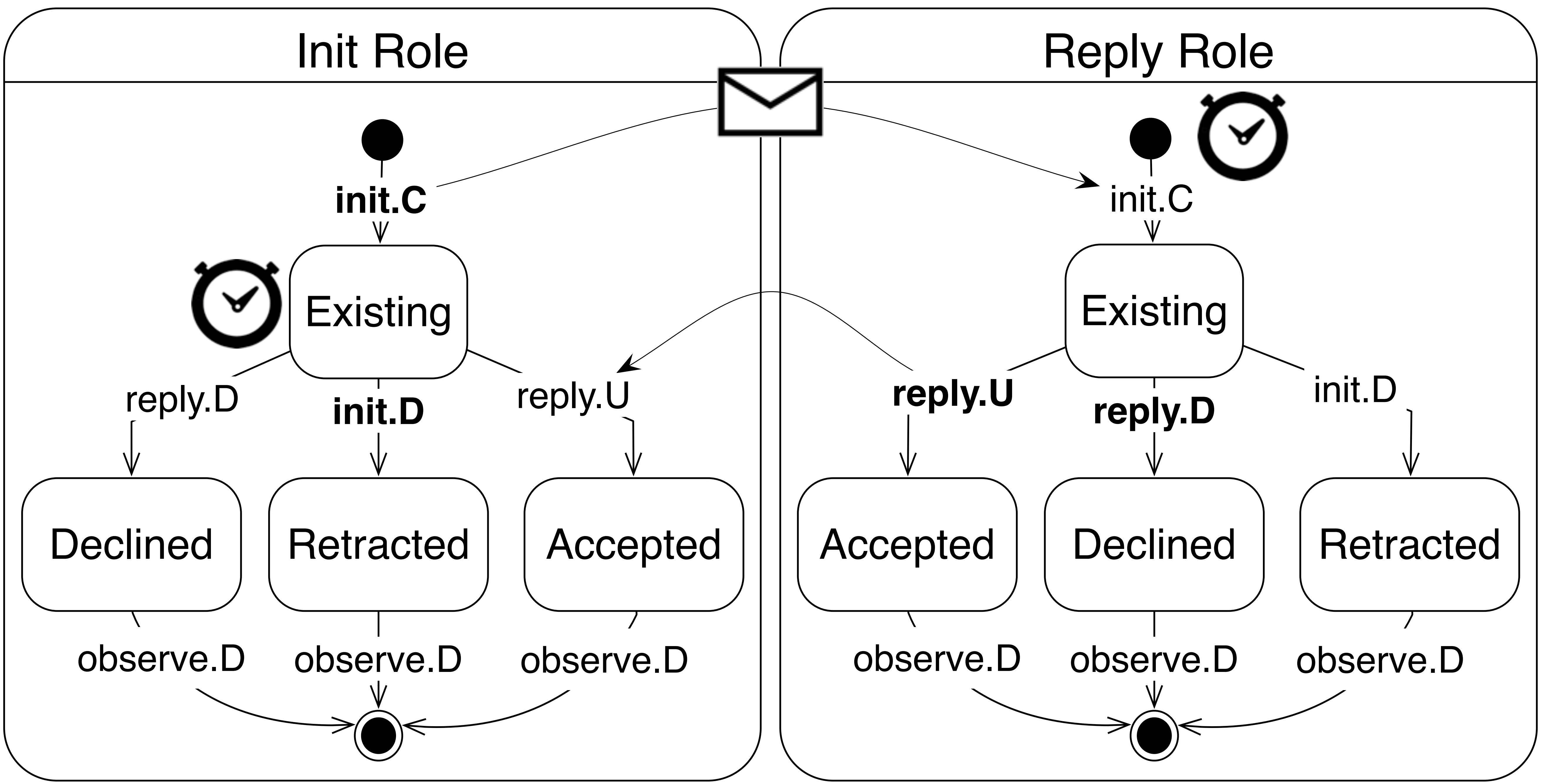
Synchronized FSMs



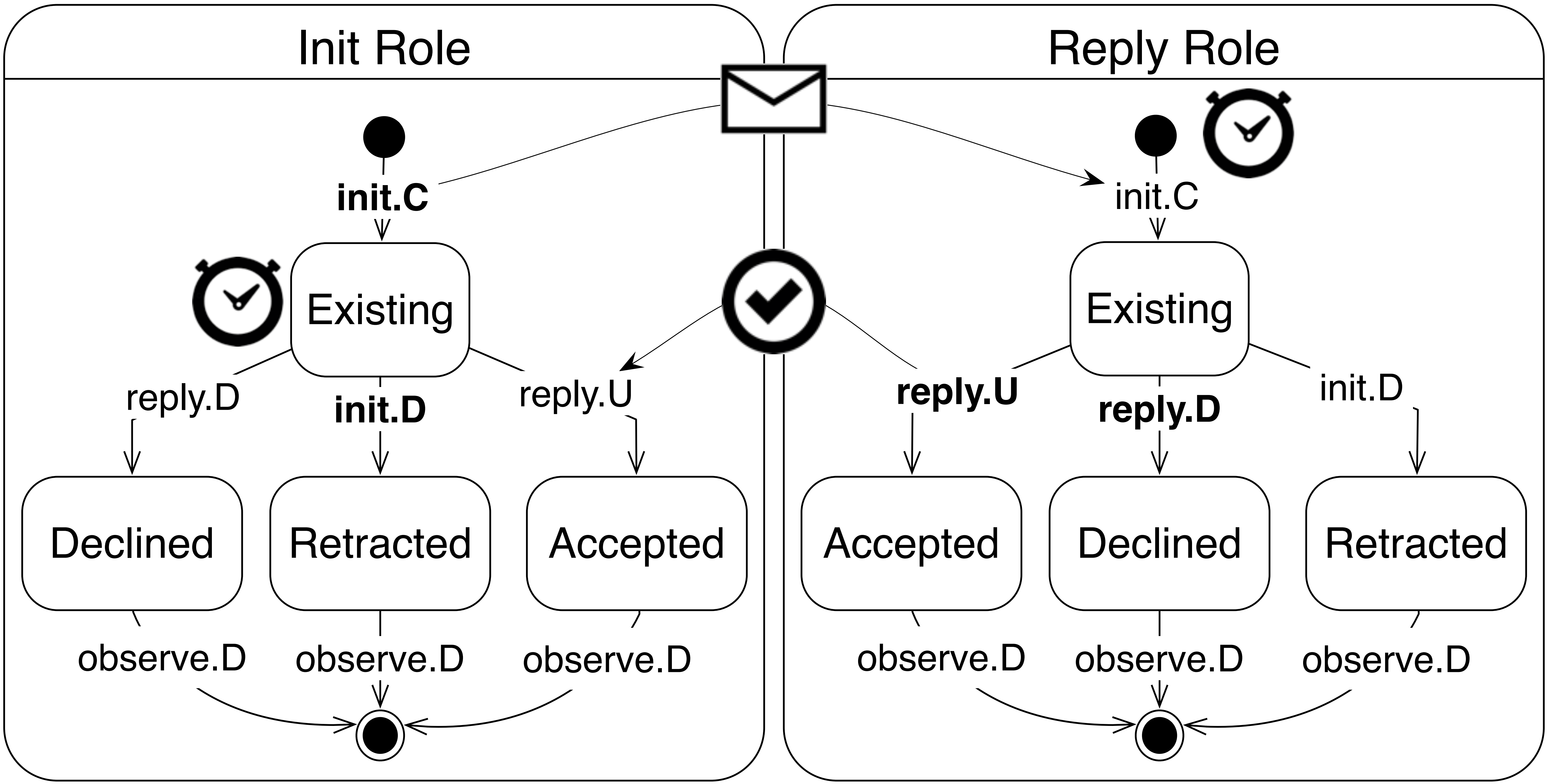
Synchronized FSMs



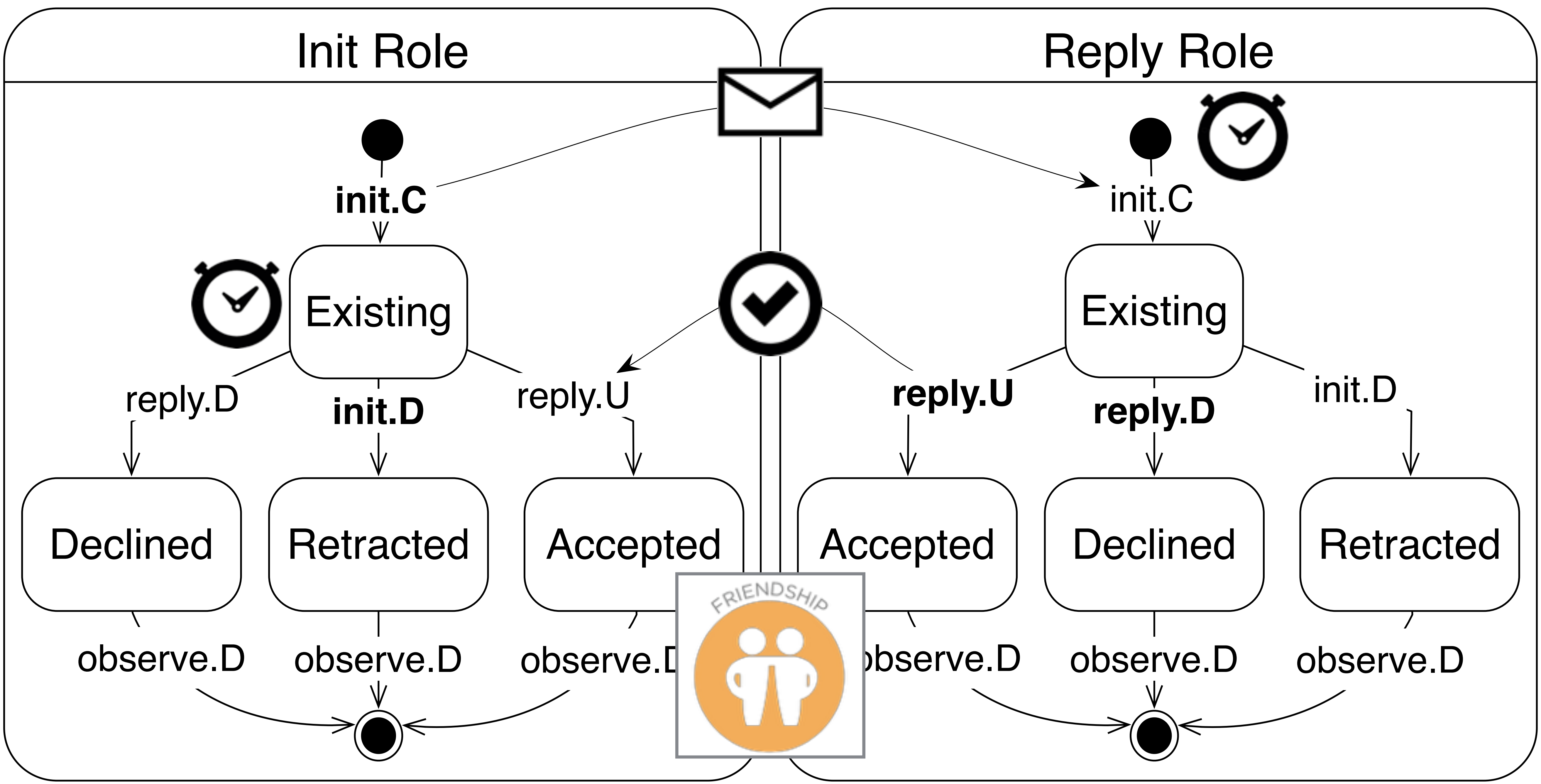
Synchronized FSMs



Synchronized FSMs



Synchronized FSMs



Additional Considerations

to build testable distributed applications

- Ideal systems can be deterministically validated
  not always possible!
- Design the system to run locally
  do not use global state
- Make sure you have **dynamic** logging
  timestamp, node ID, log levels, multiple sources
- Use APIs for messaging, time, timed events, and scheduling
  dependency injections during testing

Summary

- Testing is not only functional
- Automation is good but not always, manual testing is not always bad
- Focus the test effort on the requirements and the type of SUT
- Many tools… choose widely !

References and Additional Readings

- The Art of Software Testing, Glenn Myers
- Testing Computer Software, Cem Kaner
- Test-Driven Development by Example, Kent Beck
- How Google Tests Software, James Whittaker, Jason Arbon and Jeff Carollo
- **Mock, Stubs, and Test Double:**
    - https://stackoverflow.com/questions/2665812/what-is-mocking
    - http://cdn2.hubspot.net/hubfs/582328/GrammaTech-Img/mock-diagram.png
    - https://martinfowler.com/articles/mocksArentStubs.html
- **Test Oracle Problem**
    - http://www0.cs.ucl.ac.uk/staff/M.Harman/tse-oracle.pdf
    - http://kaner.com/?p=190
- **Test Mobile Applications:**
    - https://ymedialabs.com/17-strategies-for-end-to-end-mobile-testing-on-both-ios-and-android/
    - http://www.softwaretestingclass.com/mobile-testing-tutorial-1-mobile-application-testing-strategy/
    - http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2015/08/Mobile-Testing-Tools1.jpg
    - https://ymedialabs.com/17-strategies-for-end-to-end-mobile-testing-on-both-ios-and-android/

- **Test Mobile Applications:**
  - https://developer.android.com/training/testing/unit-testing/local-unit-tests.html
  - https://developer.android.com/training/testing/unit-testing/instrumented-unit-tests.html
  - http://www.optimusinfo.com/top-10-mobile-testing-tools/
  - https://www.ranorex.com/mobile-automation-testing/android-test-automation.html?gclid=CNrrjte2sNQCFekW0wodOroJTQ
  - http://www.softwaretestinghelp.com/best-mobile-testing-tools/
  - http://www.testingtools.com/mobile-testing/
  - http://www.softwaretestinghelp.com/5-best-automation-tools-for-testing-android-applications/
  - http://www.softwaretestingclass.com/overview-of-appium-mobile-automation-testing-tool/
  - http://www.softwaretestingclass.com/overview-of-selendroid-mobile-automation-testing-tool/
  - https://stackoverflow.com/questions/3337505/mocking-library-framework-that-works-best-in-android

- **Test Web Applications:**
  - http://www.softwaretestinghelp.com/how-can-a-web-site-be-tested/
  - Web Testing: A Complete guide about testing web applications. http://www.softwaretestinghelp.com/web-application-testing/
  - http://www.softwaretestinghelp.com/website-cookie-testing-test-cases/
  - http://www.softwaretestinghelp.com/best-cross-browser-testing-tools-to-ease-your-browser-compatibility-testing-efforts/
  - http://www.softwareqatest.com/qatweb1.html
  - https://dzone.com/articles/top-10-automated-software-testing-tools
  - http://www.softwaretestinghelp.com/most-popular-web-application-testing-tools/
- **Testing Distributed Applications**:
  - http://queue.acm.org/detail.cfm?id=2800697
  - https://www.quora.com/How-do-I-test-a-distributed-system
  - https://www.quora.com/How-do-you-design-a-distributed-system-so-that-it-can-be-deterministically-validated-by-tests