Enterprise Programmering 2

Lesson 06: Mocking, Circuit Breakers and Test Generation

Prof. Andrea Arcuri

Goals

- Learn how to test services relying on other external services, by *mocking* those external services

- Understand how and why *Circuit Breakers* are used when dealing with external services

- A look at *advanced research topics* like *Automated Test Generation*

Service Mocking

External Dependencies

- A client calls our REST, and we need to call an external service to compute our response
- What if the external service is currently down?
- What if the external service has a temporary bug?
- Test cases have to be *deterministic*
- External service poses a major challenge for testing

Mocking

Client ⟷ HTTP ⟷ REST ✕ External
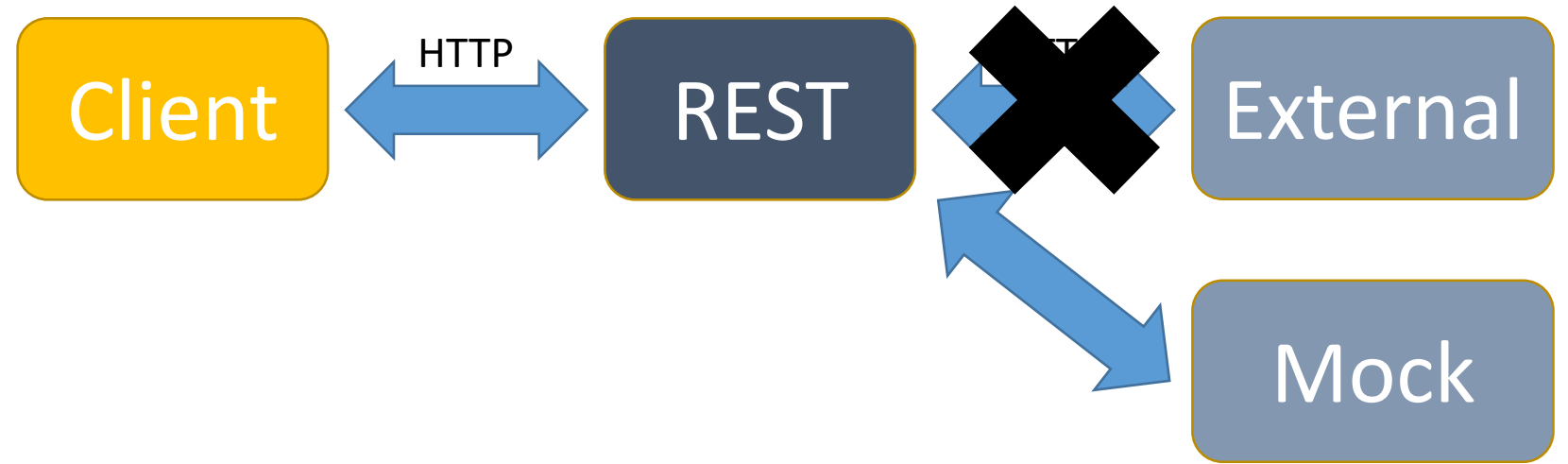
REST ⟷ Mock

- When running tests, run a Mock process listening to TCP connections
- Change in REST the IP address of External to point to Mock
- From the tests, specify what Mock should return when receiving HTTP messages
- REST does not know that it is speaking with Mock instead of the real External

Benefits

- Tests become *deterministic*
  - do not need to worry of network and External's state (eg its database)
- Can easily test scenarios which would be hard to configure for External
  - eg, special rare responses
- Can implement and test our REST even if External is not working/implemented yet
  - Example: 2 students writing 2 REST APIs. First student can implement and test X which depends on Y, even if other student is not done with Y yet

Downsides

- We are not testing how REST would behave in a real context, but in what is our *expectation* of how External interacts with it
- If lots of interactions, might need to write a lot of mocked responses
- If External does change often, then *maintaining* the mocks becomes expensive
- Still need some *live* tests with real External anyways
  - but those will be handled specially

Circuit Breaker

Circuit Breaker

- If too many connections to a server fail, stop ALL future attempts at connecting
- Can use a library to wrap each call to external services
- Once the circuit breaker is on after several failures, it will periodically check if the server comes up again. If so, all communications are restored

Why?

- TCP/HTTP communications are expensive
- If an external service is down, want to avoid wasting resources in trying to connect to it
- Performance gain: can return response immediately instead of trying to connect to external services which are down
- When external service will come up again, don't want to bombard it immediately with all clients resending all the messages that failed before, all at the same time (which might congest the external service again)

Automated Test Generation

Goals

- See how **test case generation** can be modelled as an optimization problem, eg using Evolutionary Algorithms
- Introduction to **EvoMaster** *research* tool, aimed at system testing of REST APIs
  - *Disclaimer*: this is based on my research work, and not something widespread in industry

Example: Triangle Classification (TC)

Input

Output

A

B

C

SUT

Equilateral

Isosceles

Scalene

Not a triangle
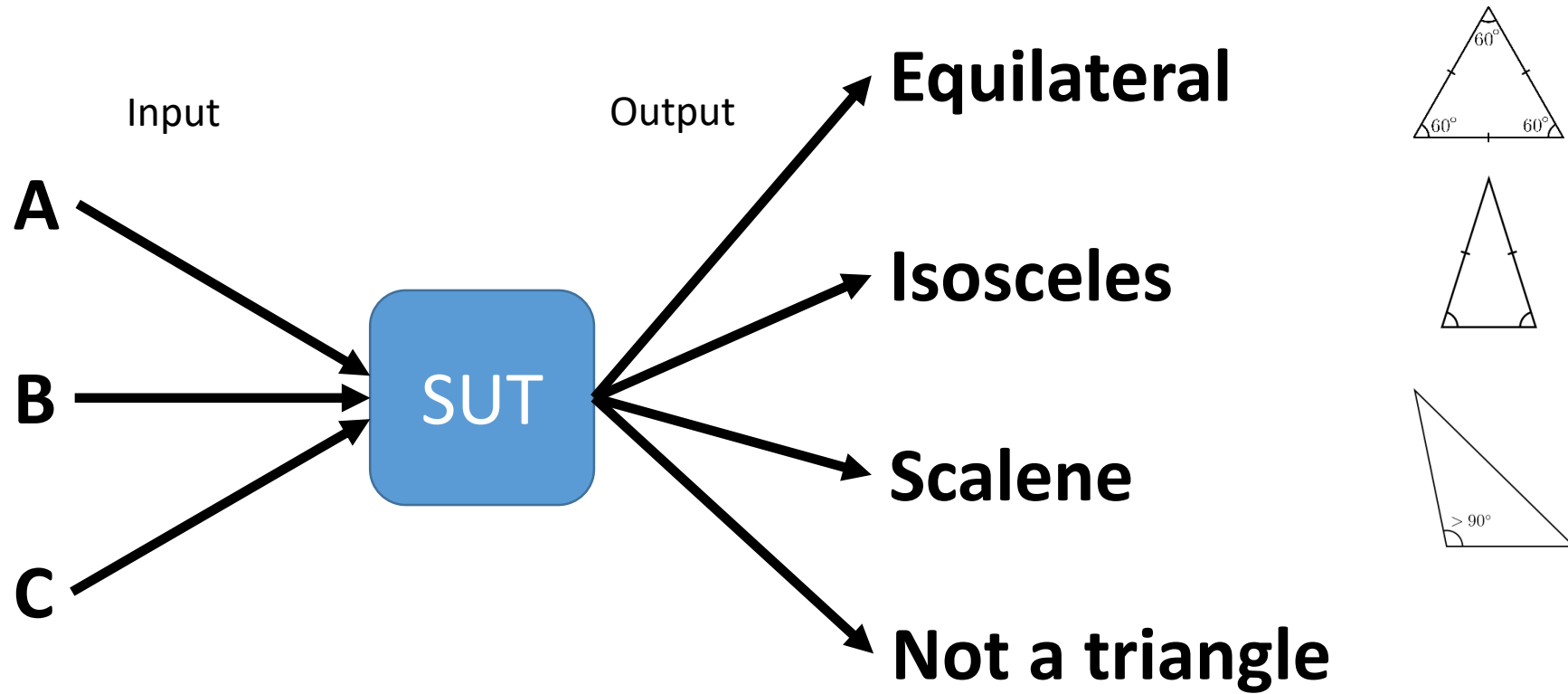
- 3 integer numbers (A, B and C) as input representing the length of the *edges*
- 4 possible outcomes
- Does the *system under test* (SUT) give the right answer?

How to test TC?

- If numbers are 32 bit integers, there are $2^{32} * 2^{32} * 2^{32} = 2^{96} = 79,228,162,514,264,337,592,626,226,666$ possible combinations
  - ie, 79 Octillion possible combinations of edge lengths
- Cannot test all of them
- Need to define some *test criteria* to decide a good enough test suite which is:
  1. good at finding bugs
  2. small enough to be manageable

1 Test per Output

- **t0:(A=42, B=42, C=42) => EQUILATERAL**
- **t1:(A=42, B=42, C=5) => ISOSCELES**
- **t2:(A=42, B=43, C=44) => SCALENE**
- **t3:(A=42, B=42, C=12345) = NOT A TRIANGLE**
- *Would such 4 test cases be enough?*
- What if the EQUILATERAL case is implemented with just something as naïve as "**if A==B and B==C then EQUILATERAL**"?
  - (**A=-3, B=-3, C=-3**) would wrongly return EQUILATERAL instead of NOT A TRIANGLE
  - Just checking basic scenarios is not enough

White-Box Testing

- Code can have bugs
- *To trigger a bug, the code must be executed*
- But code can have very complex control flow
- Some rare "paths" in the code might be executed only in very complex scenarios
- *Goal: in a test suite, have each single line and branch be executed at least once*

```java
public Classification classify(
                    int a, int b, int c){

    if(a <=0 || b<= 0 || c<= 0){
        return Classification.NOT_A_TRIANGLE;
    }

    if(a==b && b==c){
        return Classification.EQUILATERAL;
    }

    int max = Math.max(a, Math.max(b,c));

    if( (max == a && max -b -c >= 0 ) ||
            (max == b && max -a -c >= 0 ) ||
            (max == c && max -a -b >= 0 ) ){
        return Classification.NOT_A_TRIANGLE;
    }

    if(a==b || b==c || a==c){
        return Classification.ISOSCELES;
    } else {
        return Classification.SCALENE;
    }

}
```

Example

- **if((max == a && max -b -c >= 0) ||**
  (max == b && max -a -c >= 0) ||
  (max == c && max -a -b >= 0))

- In this disjunction of 3 different clauses, if in your test suite the first clause is always true, the other 2 would never be executed
  - so if wrong, you would not know

- This is a TRIVIAL example... real industrial software can be way more complex...

- Writing tests for each path is not only tedious, but can be quite hard as well

Automated Test Case Generation

- **Automatically generate test cases**
- Model software testing as an **optimization problem**
  - Maximize code coverage
  - Find bugs
  - Etc.
- Use optimization algorithms
- Benefits: *cheaper and more effective than manual testing*
- *Hard problem to automate*
  - given a non-linear constraint, there is no guaranteed algorithm that can solve it in polynomial time

# Search-Based Software Testing (SBST)

- Biology meets Software Engineering (SE)
- Casting SE problems into *Optimization Problems*
- *Genetic Algorithms*: one of most famous optimization algorithm, based on theory of evolution
- *Evolve* test cases
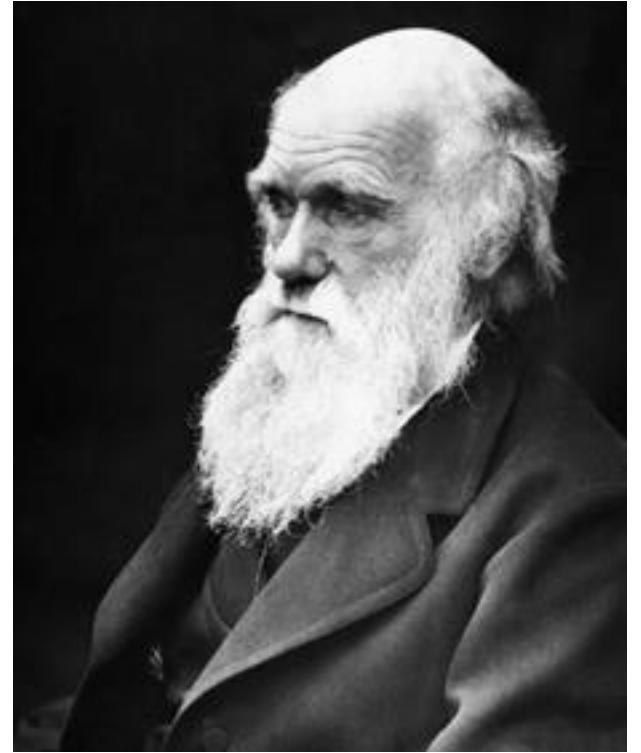
Success Stories: Facebook

Facebook uses SBST for automatically testing their software, especially their mobile apps

- eg, tools like *Sapienz* and *SapFix*
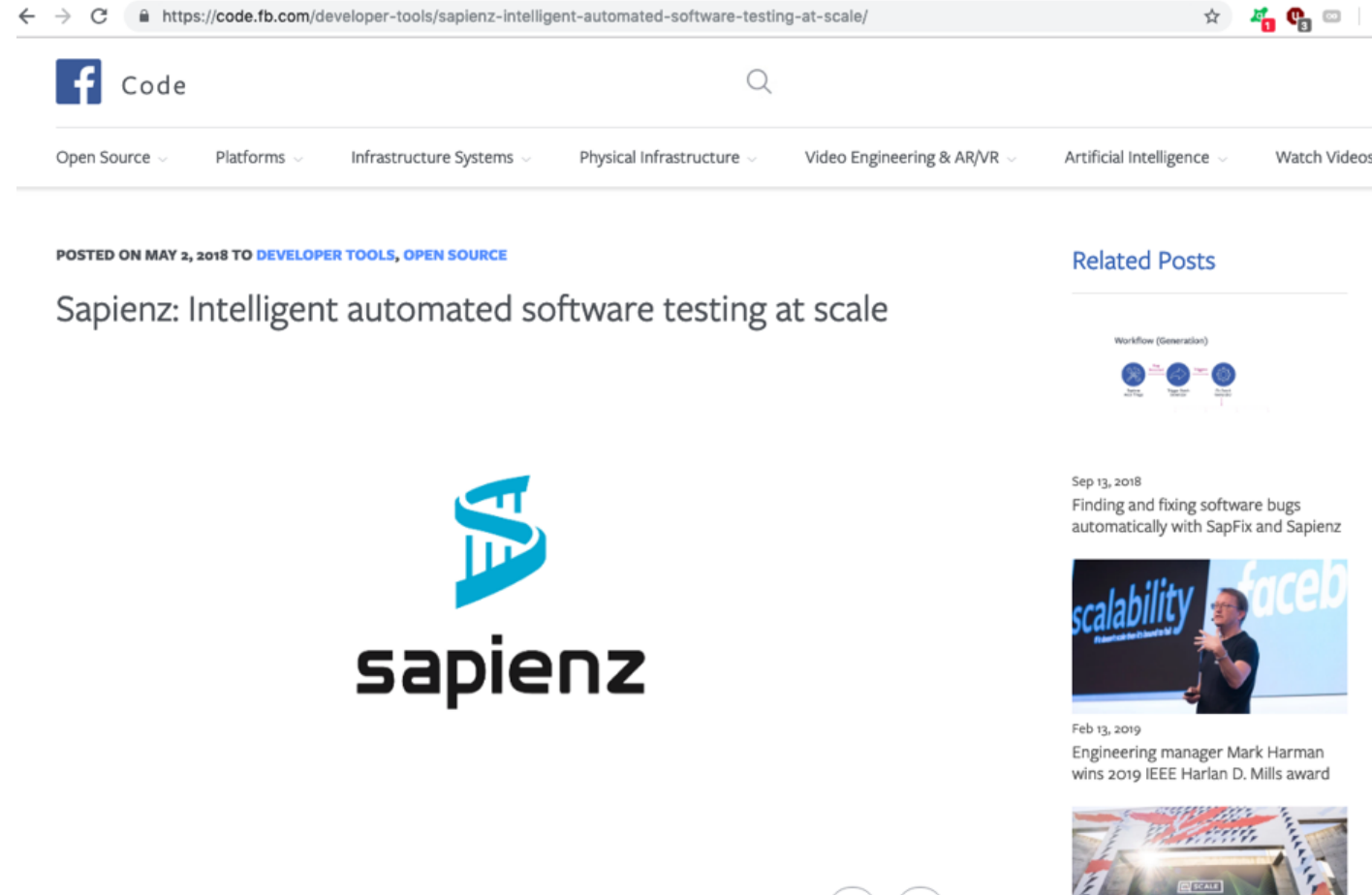
Search Space

- Depends on testing problem
- Unit testing of TC? space of all possible combination of 3 integer inputs
- OO software? sequence of function calls, with their heterogenous inputs
- REST APIs? sequence of HTTP calls, setting strings like query parameters, and JSON/XML body payloads, etc.

Search Operators

- Depend on the representation of the genes
- Usually, small changes on the genes
- Integer inputs? add +-1 on them
- Strings? add/remove/change single chars in them
- Object? modify 1 of their fields
- etc.

Fitness Guidance

- Just using code coverage as fitness function is not enough
  - binary: either we cover or we do not cover a target
- Need guidance to be able to solve constraints in code predicates
  - *"if(x == 123456 && complexPredicate(y))"*
- Would be extremely difficult to get the right *x* and *y* to solve that predicate at random

SBST Heuristics: *Branch Distance*

- Standard technique in the SBST literature to solve constraints
- Example: *if(x==100)*
  - both 5 and 90 do not solve the constraint, but 90 is *heuristically* closer
- Distance to minimize: if 0, then predicate is solved
- Not just for integers, but also all other types, eg strings
- Need to *instrument* the code to calculate those branch distances
  - *Bytecode manipulation*: need it *fully automatically,* eg with class loaders and Java Agents
  - Lot of technical details on how to achieve it efficiently

Branch Distance Examples

- $d(x == y) = |x-y|$
- $d(x >= 0) = 0$ if $x>=0$, $-x$ otherwise
- $d(A \&\& B) = d(A) + d(B)$
- $d(A || B) = \min(d(A), d(B))$
- etc.

EvoMaster

What about **Automated Test Generation** for RESTful APIs?

- Would be very useful for enterprises

- No tool available (AFAIK)

- In the past, quite a lot of work on **SOAP** web services
    - (which are not so common any more)

- Very few research papers on testing REST

- Most techniques are **black box**

EvoMaster: Evolutionary Multi-Context Automated System Testing

- Tool to automatically generate tests for REST APIs
- **White box**
  - can exploit structural and runtime information of the SUT
- Search-based testing technique (**SBST**)
- Fully automated
- Open-source prototype: *www.evomaster.org*
- Currently targeting JVM languages (eg Java and Kotlin)
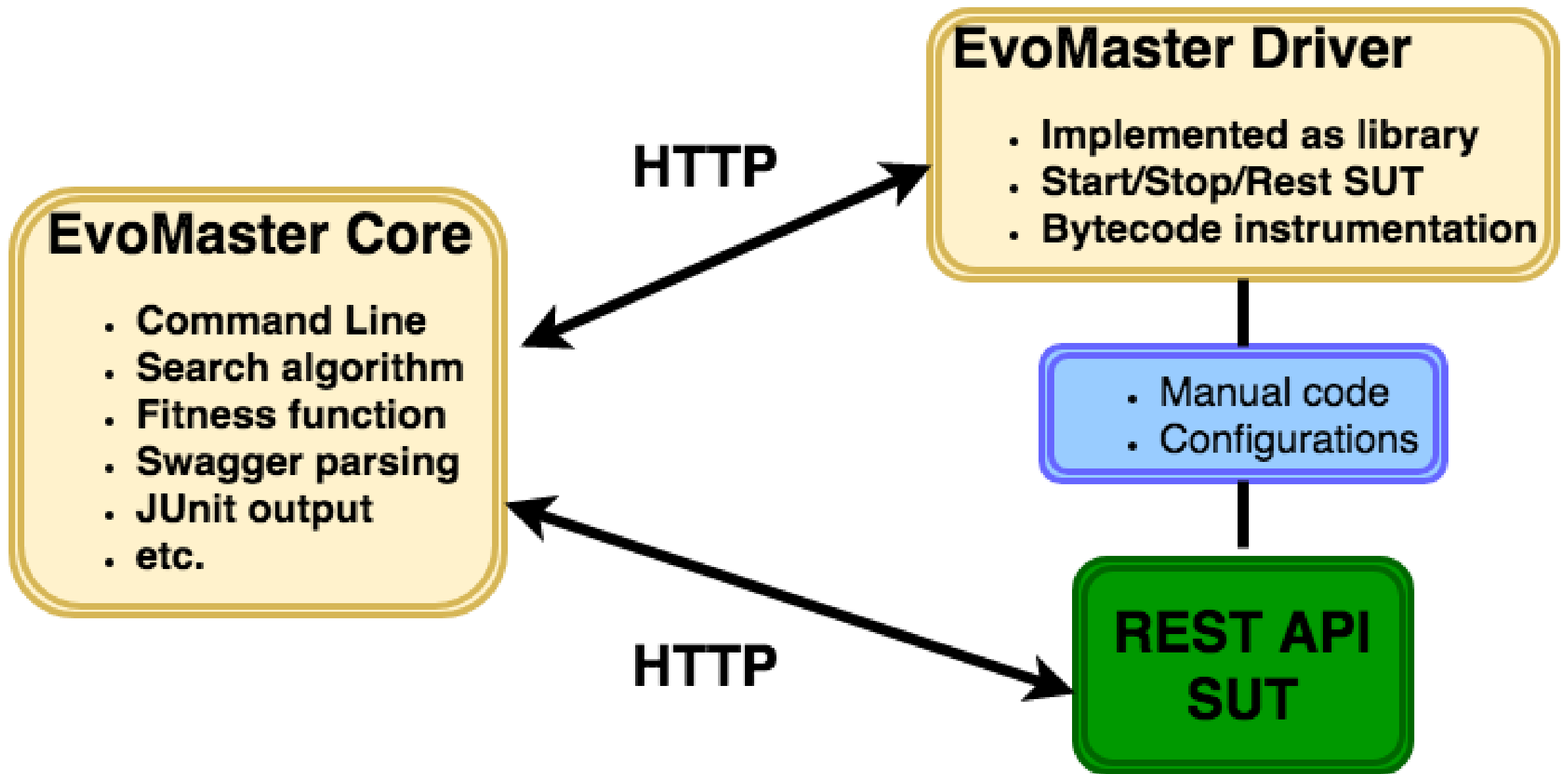
OpenAPI/Swagger

- REST is not a protocol
- Need to know what endpoints are available, and their parameters
- Schema defining the APIs
- Swagger is the most popular one
- Defined as JSON file, or YAML
- Many REST frameworks can automatically generate Swagger schemas from code
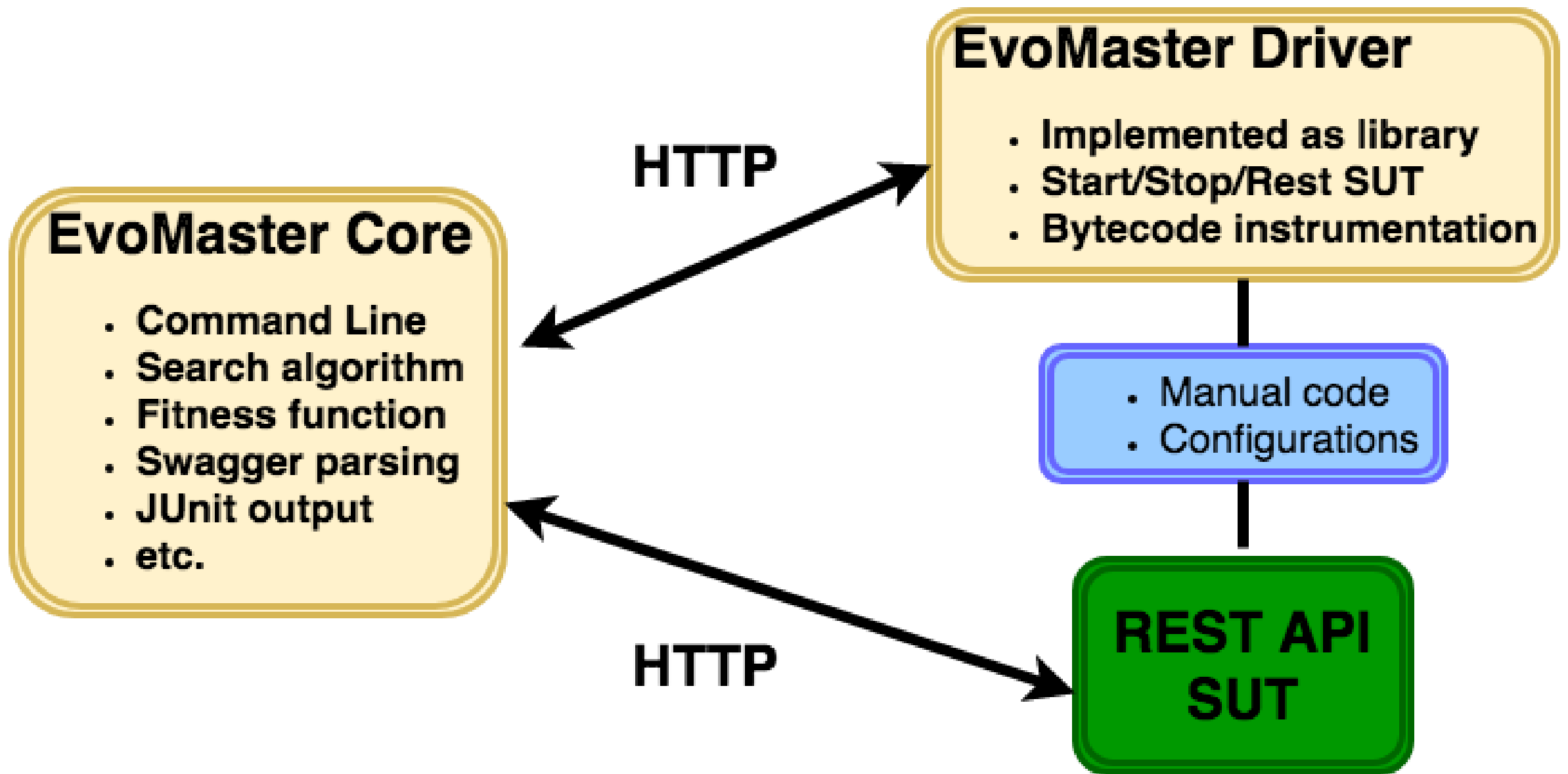
EvoMaster Core

- From Swagger schema, defines set of endpoints that can be called
- Test case structure:
  1. setup initializing data in DB with SQL INSERTs
  2. sequence of HTTP calls toward such endpoints
- HTTP call has many components:
  - Verb (GET, POST, DELETE, etc.)
  - Headers
  - Query parameters
  - Body payload (JSON, XML, etc.)
- Evolutionary algorithm to evolve such sequences and their inputs
- Output: *self-contained* JUnit tests

Fitness Function

- Needed to drive the evolution

- Reward *code coverage* and *fault detection*

- HTTP return statuses as *automated oracles*:
  - Eg 2xx if OK, 4xx are user errors, but **5xx** are server errors (often due to bugs)

EM Driver: SBST Heuristics as a Service

- Core and Driver are running on different processes
- Code coverage and branch distances sent over the net, in JSON format
- Cannot send all data: too inefficient if per test execution
  - different techniques to determine only what is necessary
- *EM Driver is itself a RESTful API*
- **Why?** Because so we can use Driver for other languages (eg C# and JS) without the need to touch EM Core

Using EvoMaster

- Need to add dependency library

- Need to write a driver class extending *EmbeddedSutController*

- Once driver is up and running, should run "*evomaster.jar*" from command-line
  - once downloaded from GitHub

- More info at *www.evomaster.org*

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **advanced/rest/wiremock**
- **advanced/rest/circuit-breaker**
- **advanced/rest/evomaster**