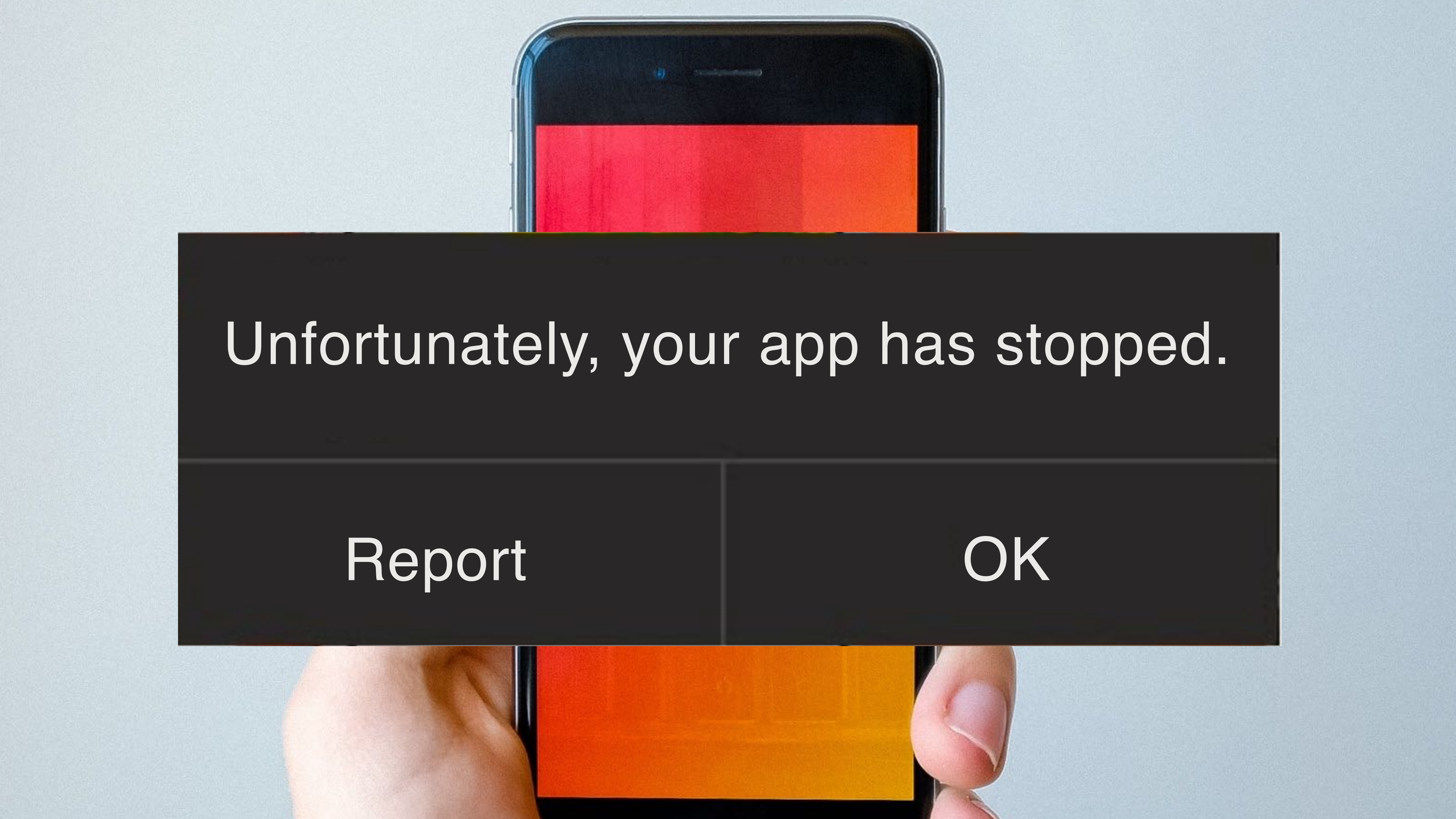
facebook

Automated Search Based Test Case Design at Facebook Scale

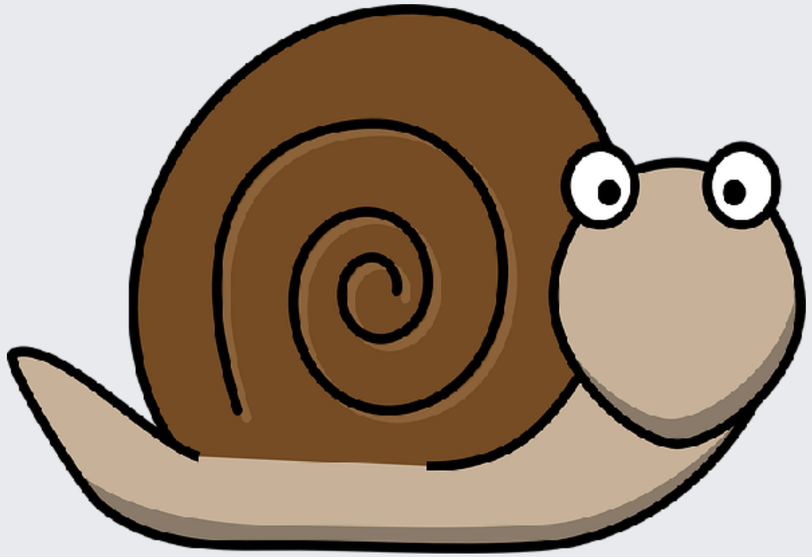
Mark Harman
Joint work with the Sapienz Team
Software Engineering Manager, Sapienz Team
November, 2017


Unfortunately, your app has stopped.
Report
OK

so we test

think of testing

think of testing



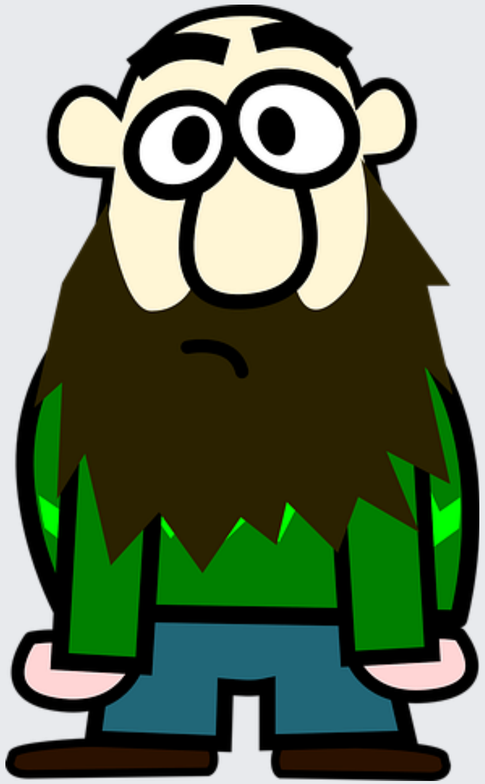
slow

think of testing

slow

painful

think of testing



slow

painful

boring

think of testing

slow

painful

boring

low impact

think of testing

slow

painful

boring

low impact

necessary

State of Practice we seek to advance

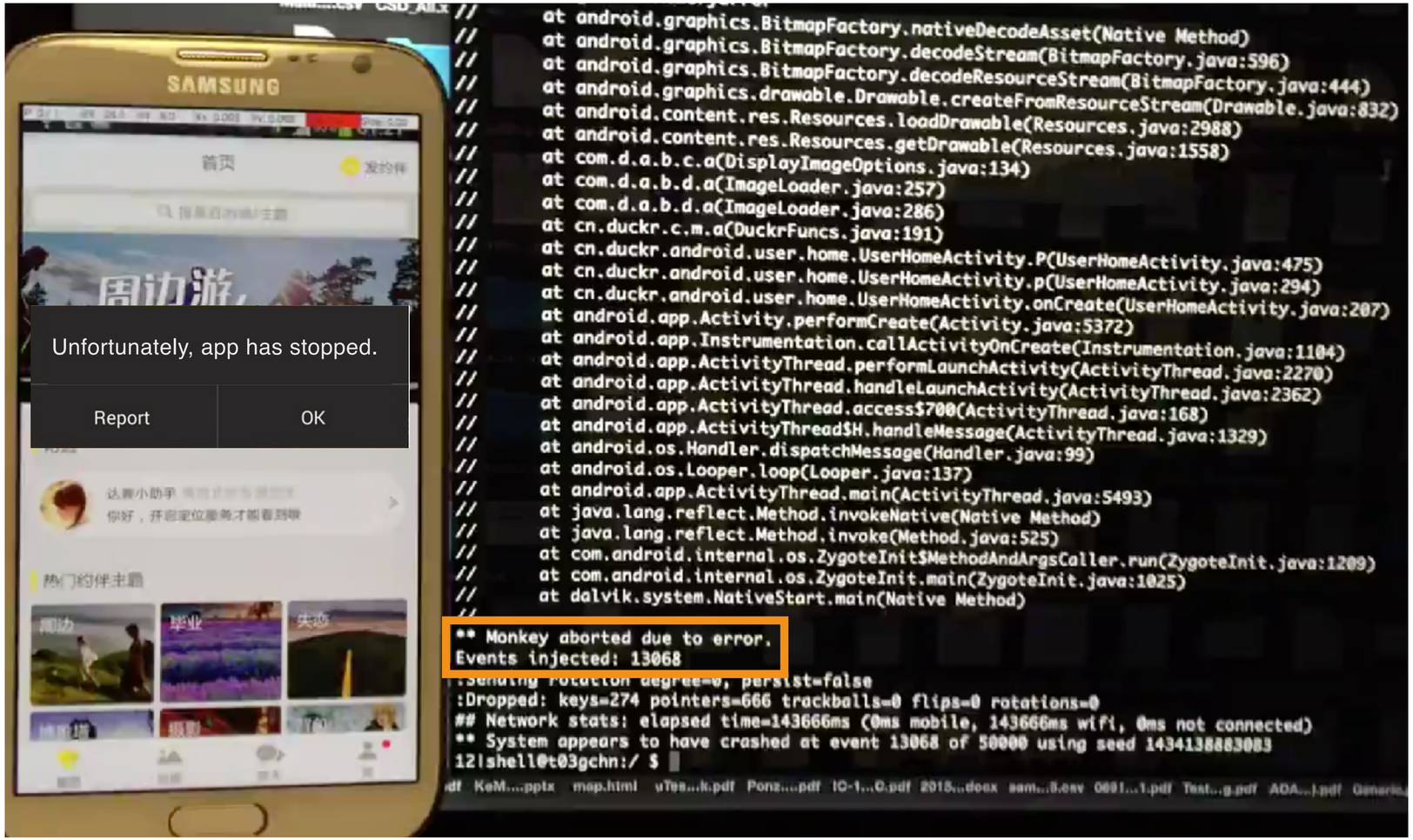
Mobile App Testing

1. **Manual**: heavily rely on human [1]

2. **Semi-automated**: automated execution, manually generated
- Appium, Robotium, Selendroid, etc.

3. **Fully-automated:** Monkey testing

Monkey Testing



The State of the Art in 2016

fully-automated Android testing

Technique	Venue	Publicly Available	Box	Approach	Crash Report	Replay Scripts	Emulator / Real Device
Monkey	N/A	Yes	Black	Random-based	Text	No	Both
AndroidRipper	ASE'12	Yes	Black	Model-based	Text	No	Emulator
ACTEve	FSE'12	Yes	White	Program analysis	N/A	Yes	Emulator
A^3E	OOPSLA'13	Partially	Grey	Model-based	N/A	Yes	Real device
SwiftHand	OOPSLA'13	Yes	Black	Model-based	N/A	No	Both
ORBIT	FASE'13	No	Grey	Model-based	N/A	No	Emulator
Dynodroid	FSE'13	Yes	Black	Random-based	Text, Image	Yes	Emulator
PUMA	MobiSys'14	Yes	Black	Model-based	Text	Yes	Both
EvoDroid	FSE'14	No	White	Search-based	N/A	No	Emulator
SPAG-C	TSE'15	No	Black	Record-replay	N/A	Yes	Real device
MonkeyLab	MSR'15	No	Black	Trace mining	N/A	Yes	Both
Thor	ISSTA'15	Yes	Black	Adverse conditions	Text, Image	Yes	Emulator
TrimDroid	ICSE'16	Yes	White	Program analysis	Text	Yes	Both
CrashScope	ICST'16	No	Black	Combination	Text, Image	Yes	Both
SAPIENZ	ISSTA'16	Yes	Grey	Search-based	Text, Video	Yes	Both

— — —

K. Mao, M. Harman, and Y. Jia. Sapienz: Multi-objective automated testing for Android applications. In ISSTA'16, 2016.

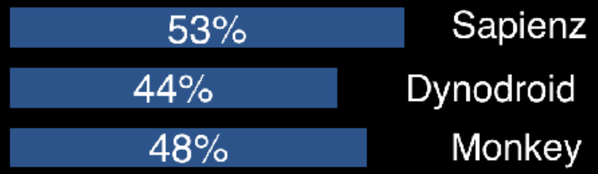
Ke Mao, Mark Harman, and Yue Jia, “Sapienz: Multi-objective automated testing for Android applications,”

Evaluation on Top 1,000 Google Play Apps

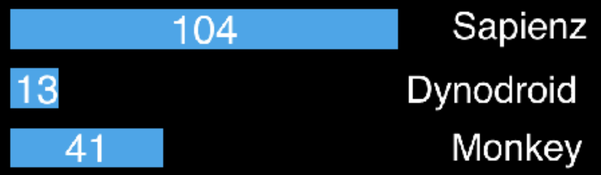
Sapienz Highlights

68 Benchmark Apps

Mean Coverage



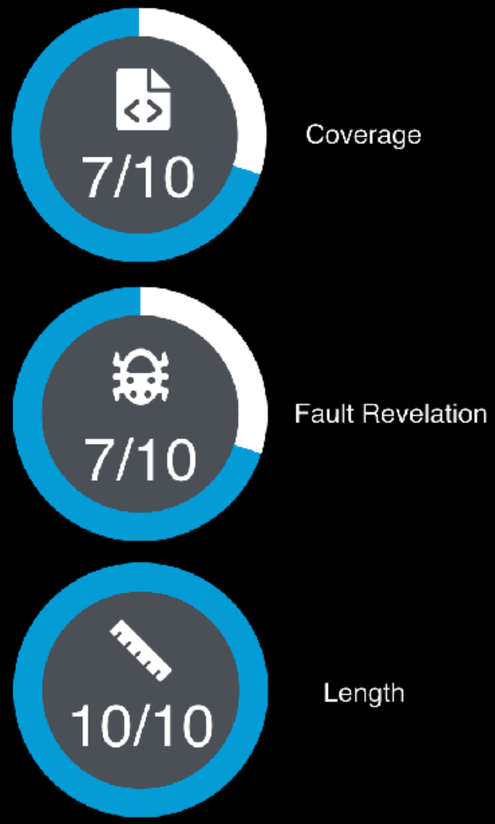
Mean Crashes



Mean Length

10 F-Droid Apps

Statistically outperformed Monkey and Dynodroid



1000 GooglePlay Apps

Revealed Failures

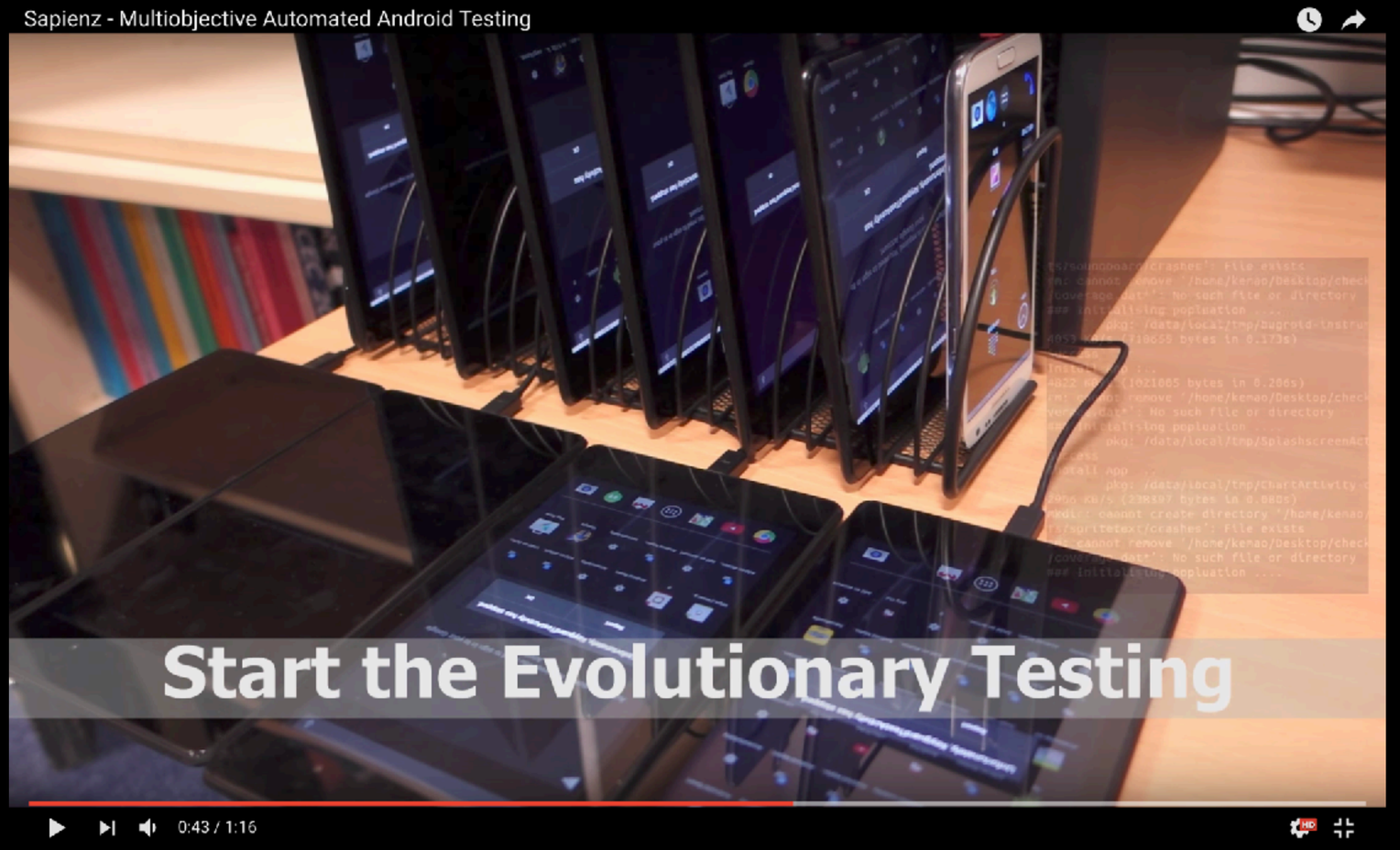
558

developer confirmed bugs


SAPIENZ

Sapienz Prototype

ISSTA'16 Research Demo



https://www.youtube.com/watch?v=j3eV8NiWLg4

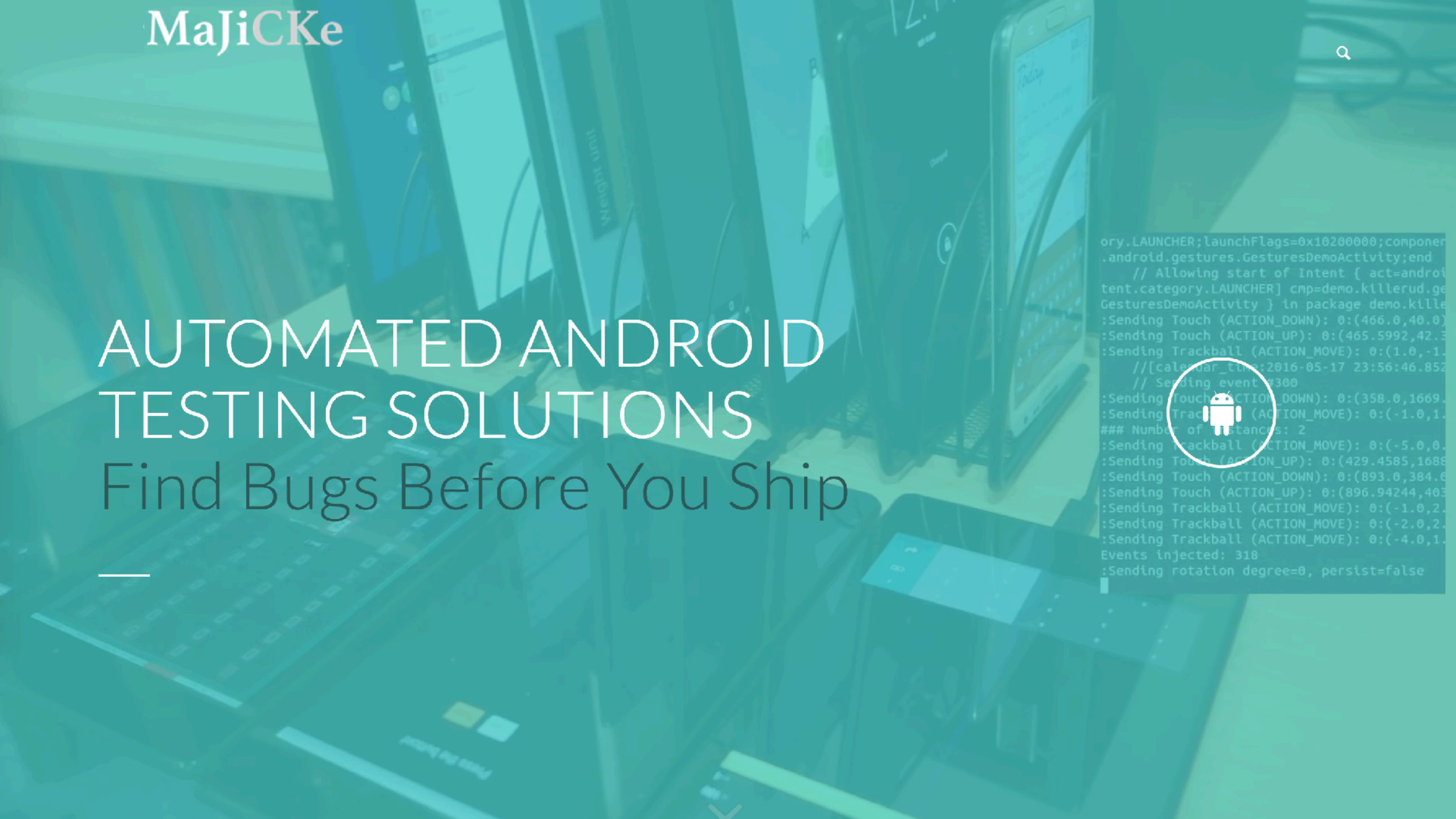
The State of the Practice

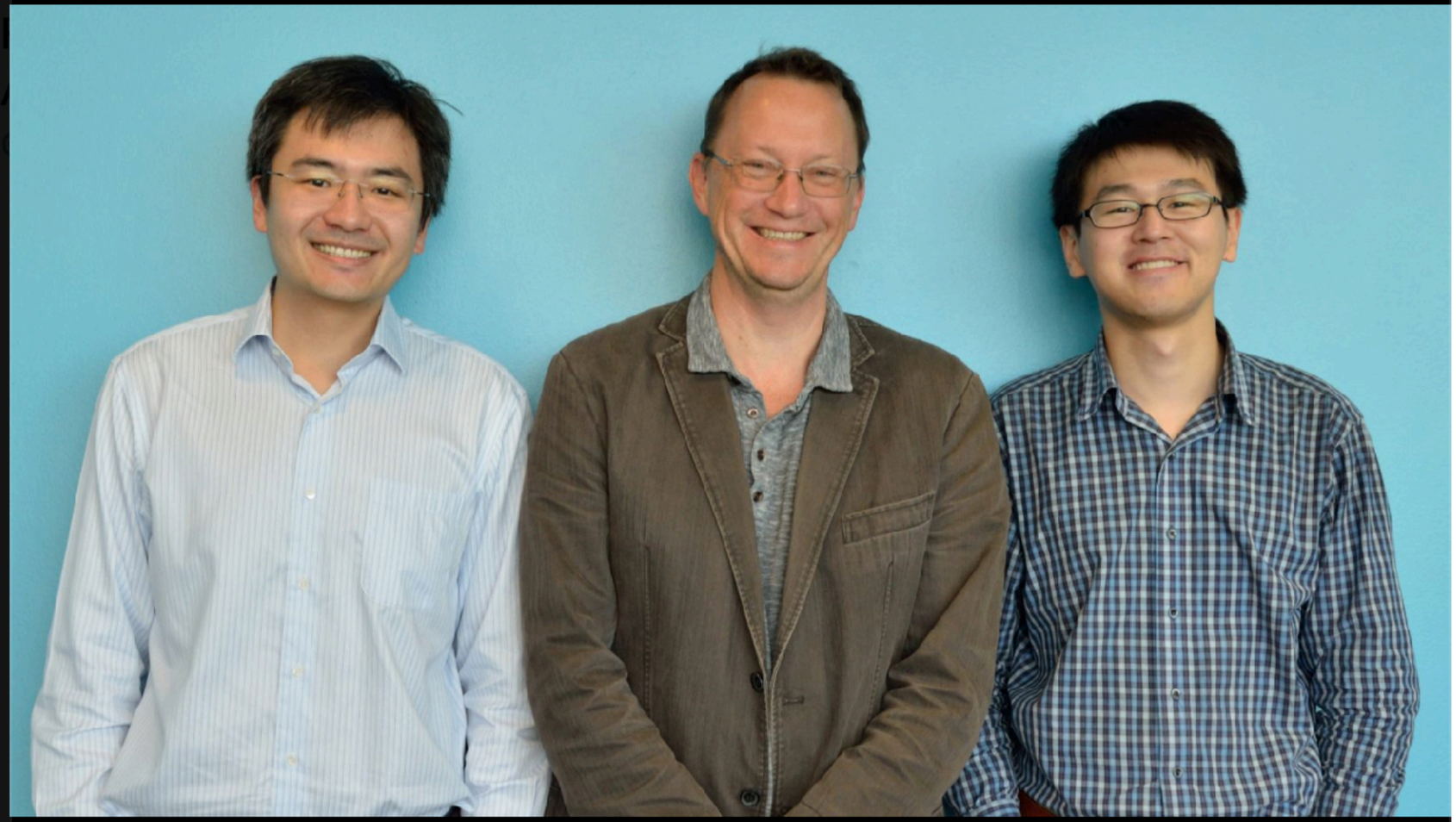
The State of the Art

MaJiCKe

AUTOMATED ANDROID TESTING SOLUTIONS

Find Bugs Before You Ship





Facebook Academics
Like This Page · 10 January ·

We're excited to announce that the team behind MaJiCKe will be joining us at Facebook in London. MaJiCKe has developed software that uses Search Based Software Engineering (SBSE) to help engineers find bugs while reducing the inefficiencies of writing test code. Their key product, Sapienz, is a multi-objective end-to-end testing system that automatically generates test sequences using SBSE to find crashes using the shortest path it can find.

The company's three co-founders Mark Harman (Scientific Advisor), Yue Jia (CEO), and Ke Mao (CTO) are researchers at University College London (UCL), currently funded, in part, by the UK's Engineering and Physical Sciences Research Council (EPSRC). They are all leaders in the field of computational search intelligence and will be joining an existing roster of strong engineering talent in our London office that is critical to building Facebook. We can't wait for the team to get started and to help us move faster towards our goal of connecting the world.

computational search intelligence and will be joining an existing roster of

What's the biggest challenge that cuts across *all* software engineering ?

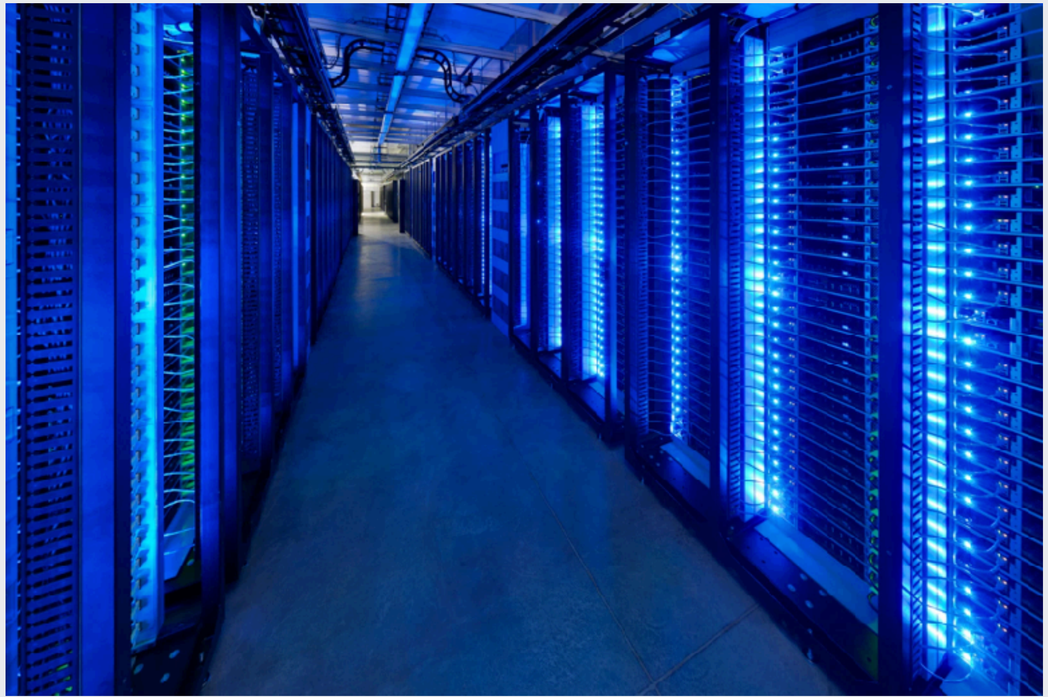
in one word?

scalability

If it doesn't scale then it's bound to fail

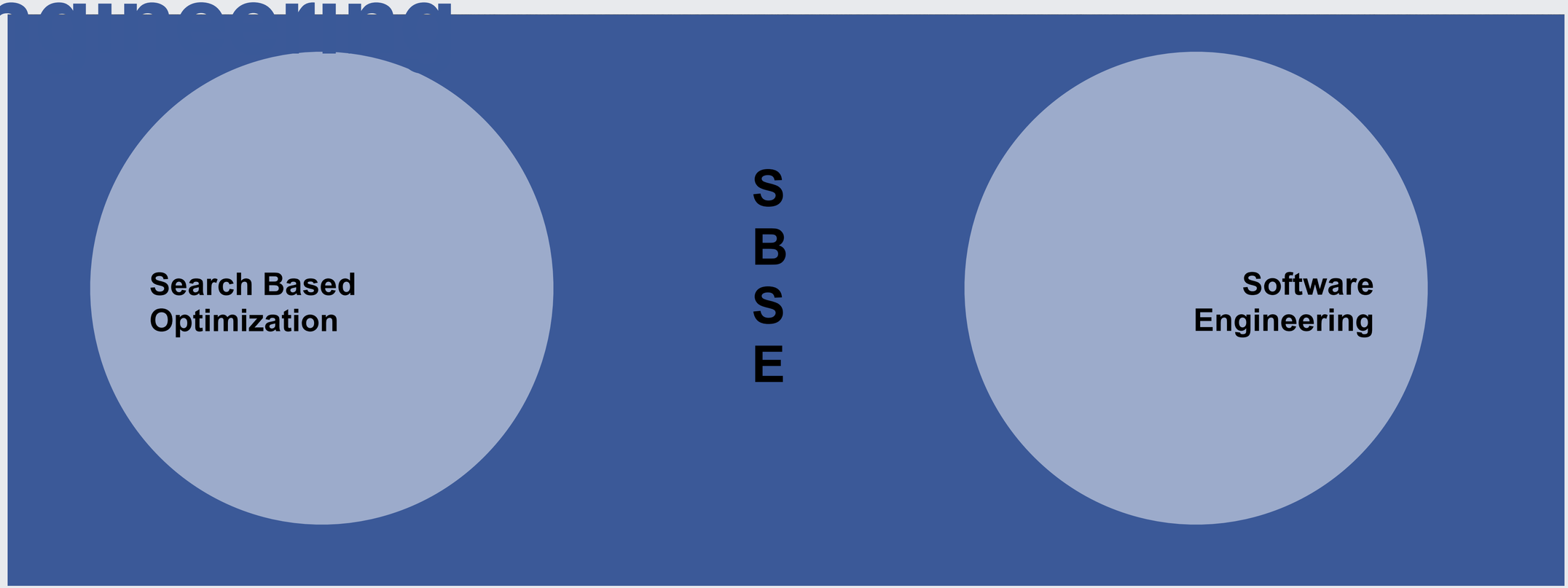
Engineers Design

Machines Execute



Design & Execution Automate Both

We use Search Based Software Engineering

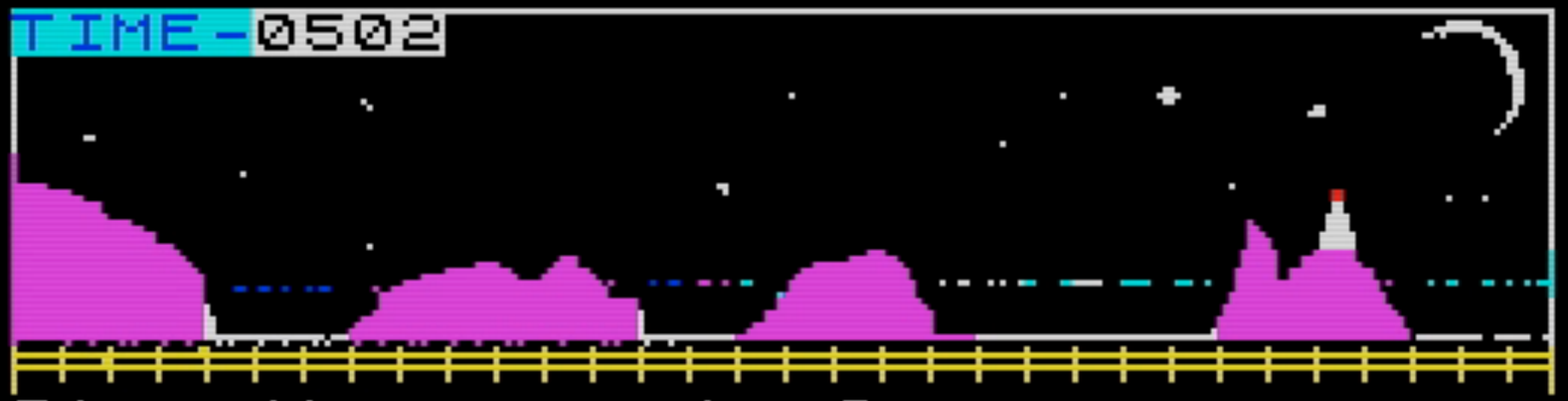


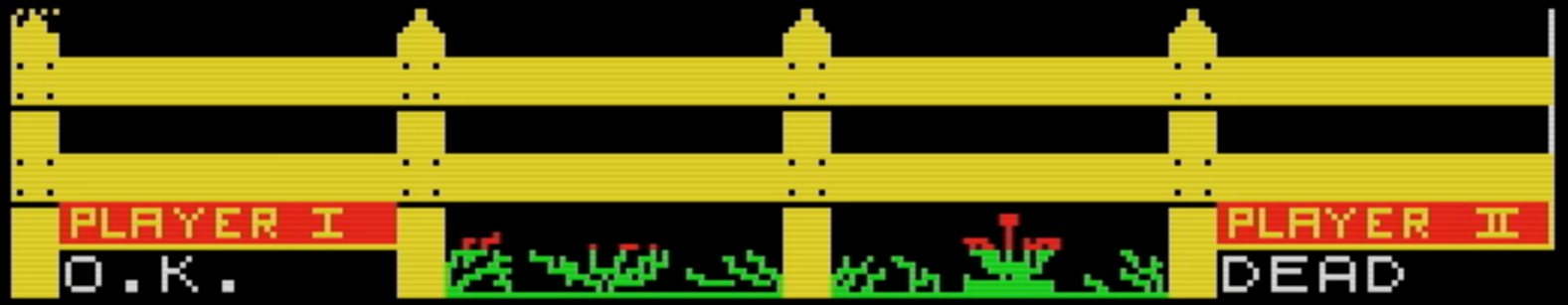
widely studied in academia; now starting to reach into industry

My coding life...

1980s

1980s

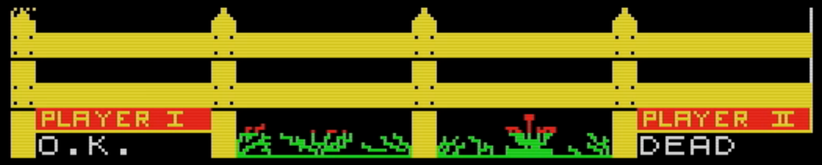


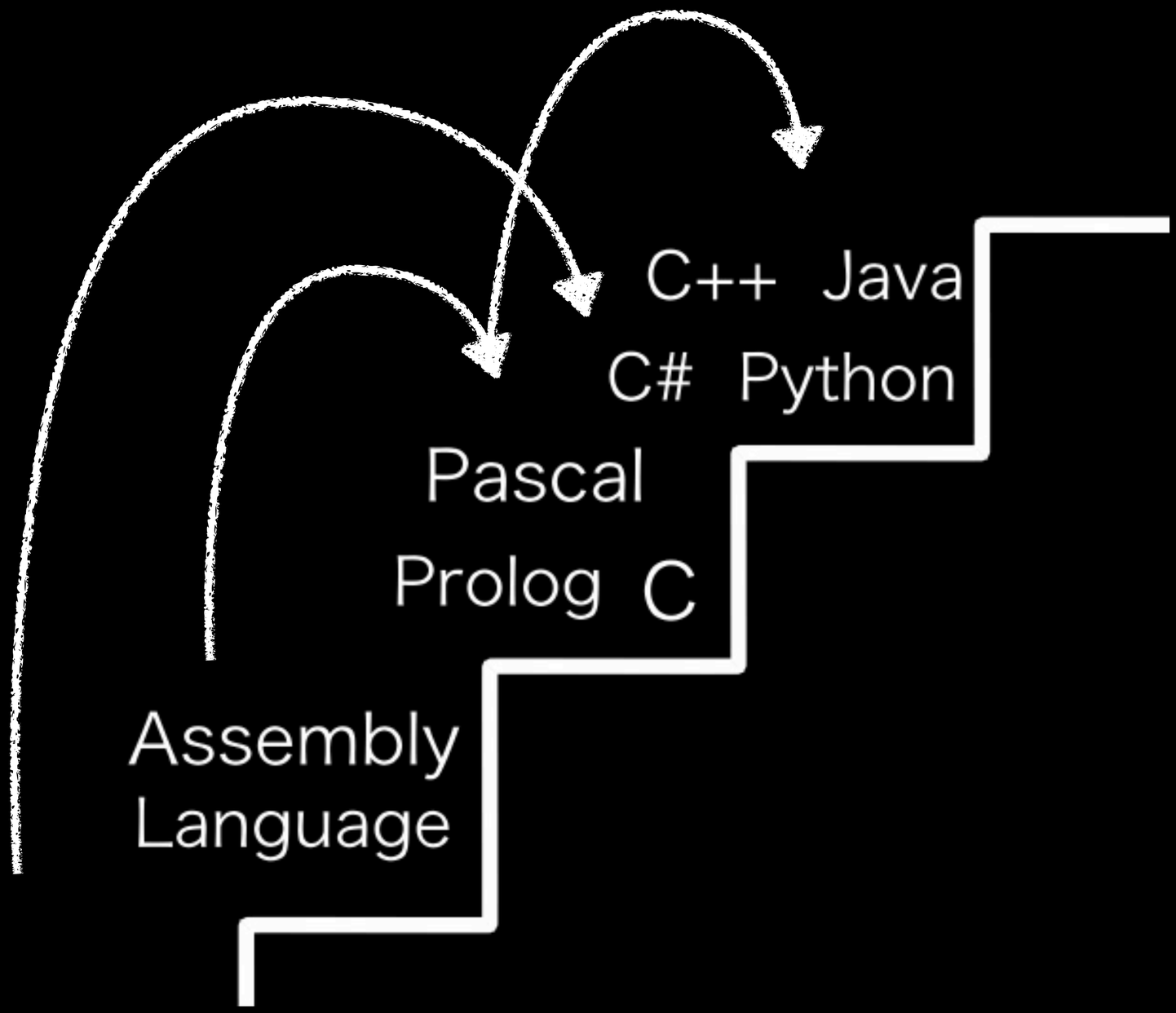


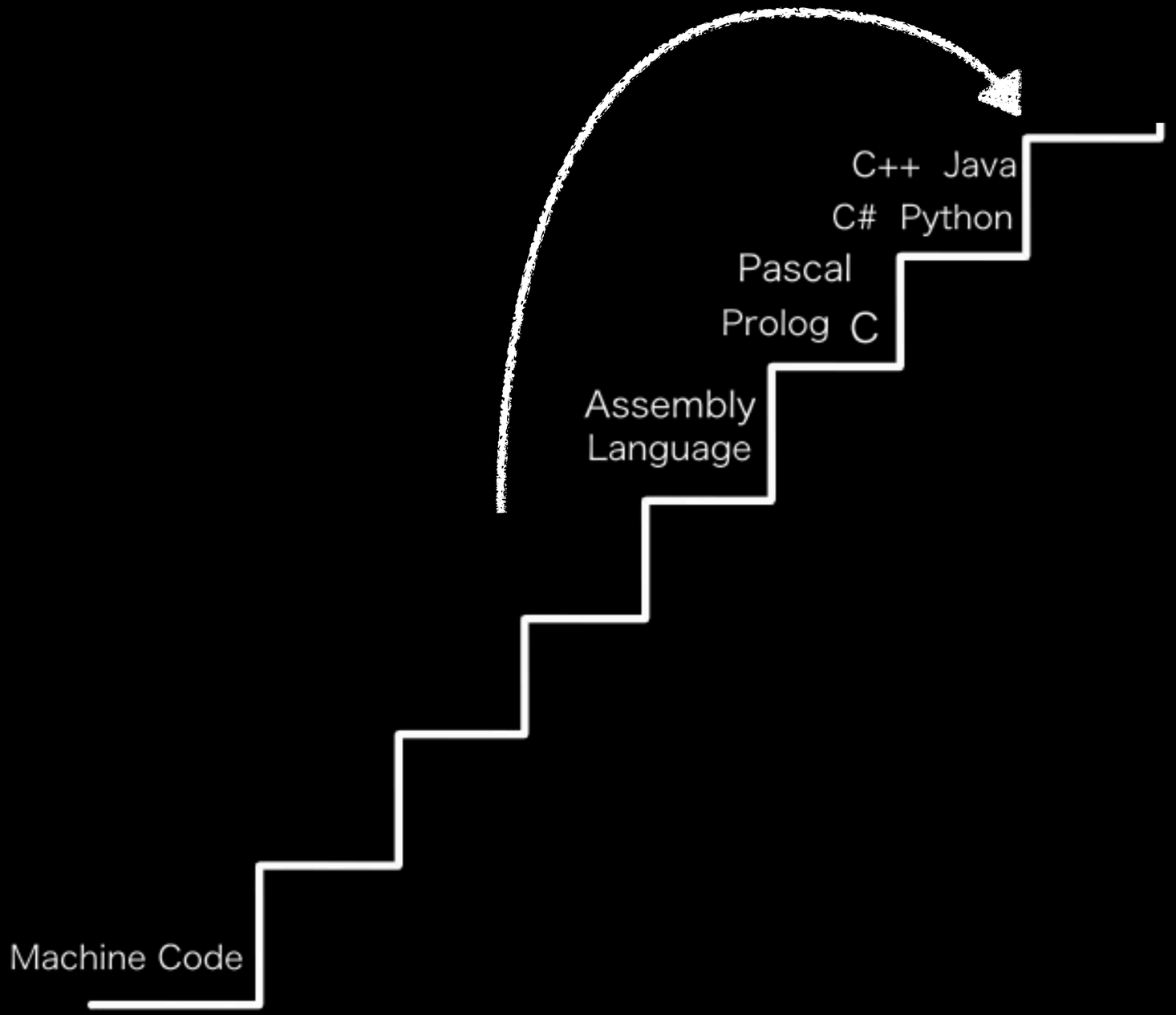
1980s

```
7000  3A 08 5C FE. 79 CA AE 70
7008  FE 6E CA 0F. 70 18 F1 CD
```

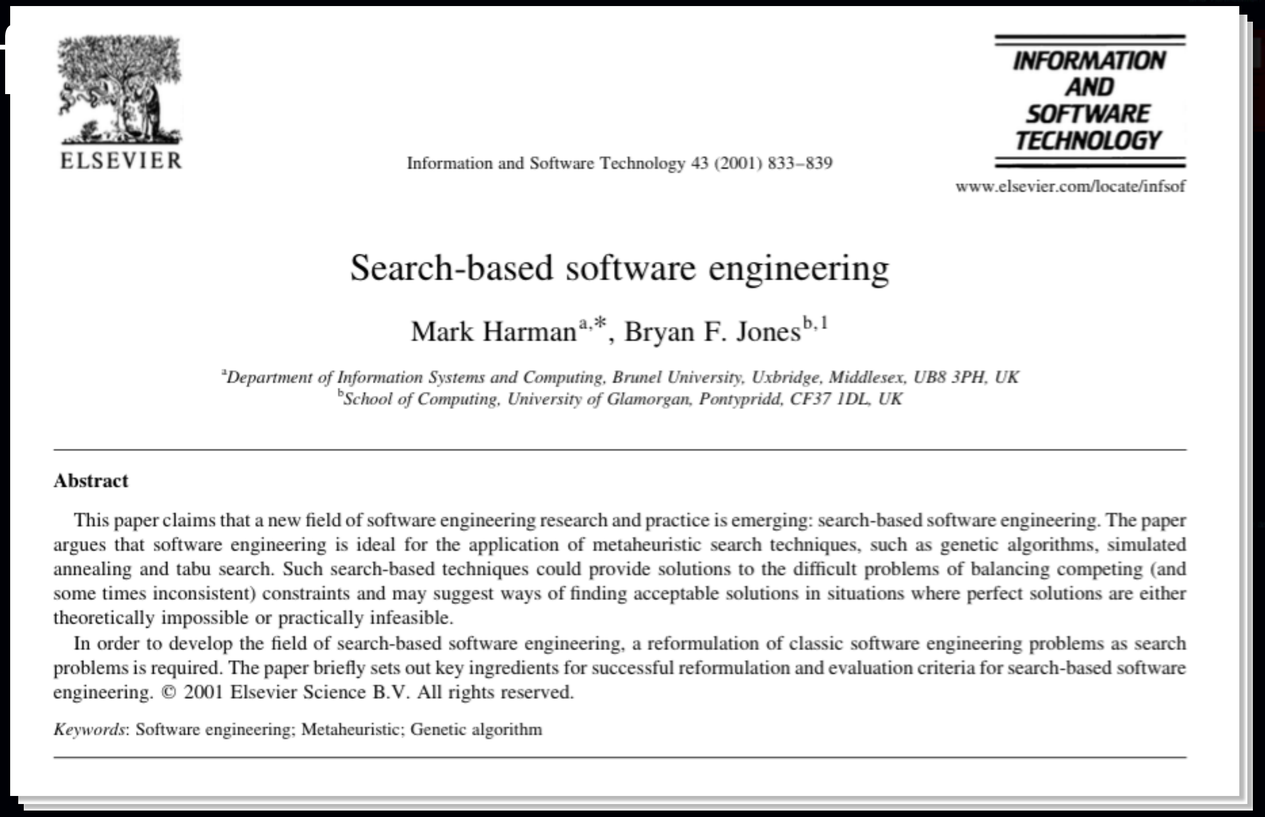
```
7000  ld   a,($5C08)
7003  cp   a,121
7005  jp   z,$70AE
7008  cp   a,110
700A  jp   z,$700F
700D  jr   $7000
```




C++ Java
C# Python
Pascal
Prolog C
Assembly
Language


C++ Java
C# Python
Pascal
Prolog C
Assembly
Language
Machine Code

History of



ELSEVIER

Information and Software Technology 43 (2001) 833–839

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Search-based software engineering

Mark Harman[a,*], Bryan F. Jones[b,1]

[a]*Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK*
[b]*School of Computing, University of Glamorgan, Pontypridd, CF37 1DL, UK*

Abstract

This paper claims that a new field of software engineering research and practice is emerging: search-based software engineering. The paper argues that software engineering is ideal for the application of metaheuristic search techniques, such as genetic algorithms, simulated annealing and tabu search. Such search-based techniques could provide solutions to the difficult problems of balancing competing (and some times inconsistent) constraints and may suggest ways of finding acceptable solutions in situations where perfect solutions are either theoretically impossible or practically infeasible.

In order to develop the field of search-based software engineering, a reformulation of classic software engineering problems as search problems is required. The paper briefly sets out key ingredients for successful reformulation and evaluation criteria for search-based software engineering.

Keywords: Software engineering; Metaheuristic; Genetic algorithm

1842

2001

The first use of ptimisation techniques in software testing and verification
1969

"In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them for the purposes of a Calculating Engine. **One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation**."

Extract from 'Note D'.

The first mention of software optimisation
1842

1845 1850 1855 1860 1865 1870 1875 1880 1885 1890 1895 1900 1905 1910

The first mention of software optimisation

1842

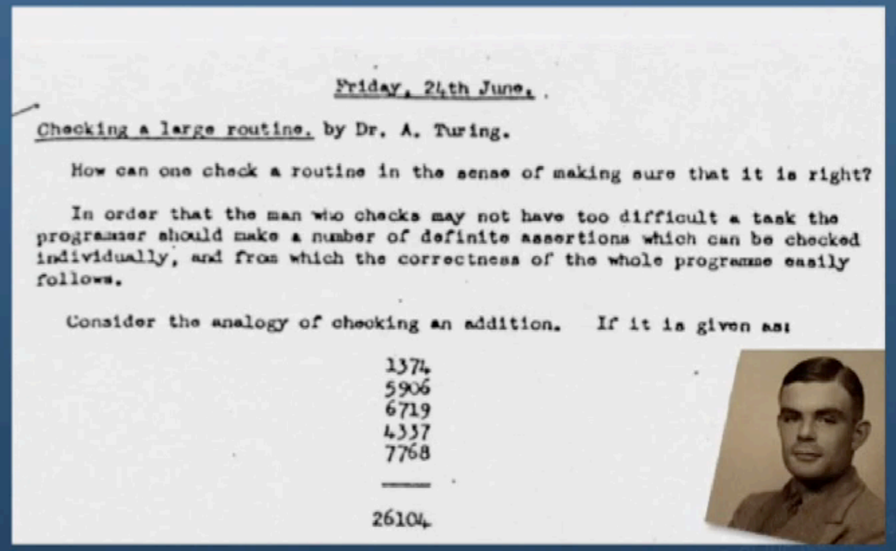
1845 1850 1855 1860 1865

Checking a large routine
by Dr. A. Turing

In this shot paper, Turing suggested the use of manually constructed assertions and we can find the origins of **both software testing and software verification**.

The introduction of the idea of software testing and verification

1949



Friday, 24th June.

Checking a large routine. by Dr. A. Turing.

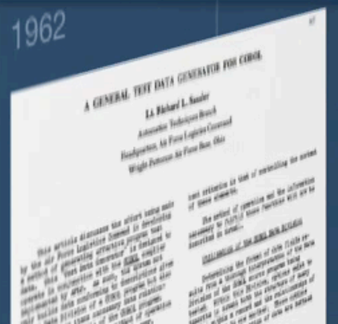
How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

1374
5906
6719
4337
7768
———
26104

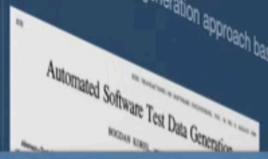
1962



A GENERAL TEST DATA GENERATOR FOR COBOL

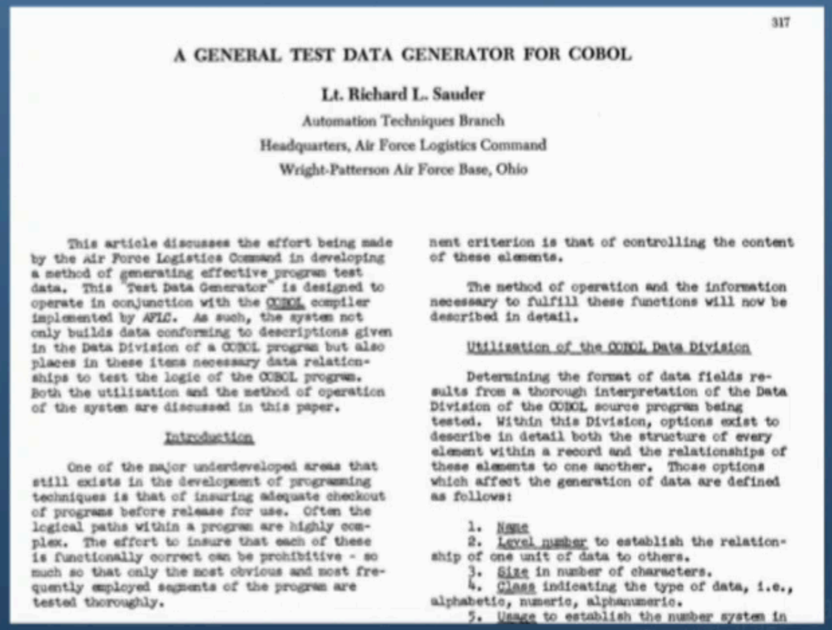
Sauder formulates the test generation problem as one of finding test inputs from a search space, though the search algorithm is random search, making this likely to be the first paper on **Random Test Data Generation**.

1990



Formulation of the test input space as a search space

1962



317

A GENERAL TEST DATA GENERATOR FOR COBOL

Lt. Richard L. Sauder
Automation Techniques Branch
Headquarters, Air Force Logistics Command
Wright-Patterson Air Force Base, Ohio

This article discusses the effort being made by the Air Force Logistics Command in developing a method of generating effective program test data. This "Test Data Generator" is designed to operate in conjunction with the COBOL compiler implemented by AFLC. As such, the system not only builds data conforming to descriptions given in the Data Division of a COBOL program but also places in these items necessary data relationships to test the logic of the COBOL program. Both the utilization and the method of operation of the system are discussed in this paper.

Introduction

One of the major underdeveloped areas that still exists in the development of programming techniques is that of insuring adequate checkout of programs before release for use. Often the logical paths within a program are highly complex. The effort to insure that each of these is functionally correct can be prohibitive - so much so that only the most obvious and most frequently employed segments of the program are tested thoroughly.

nent criterion is that of controlling the content of these elements.

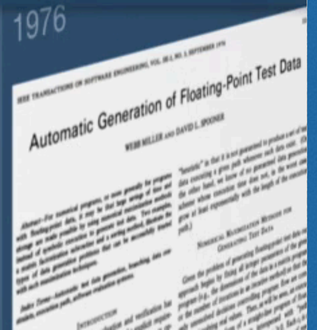
The method of operation and the information necessary to fulfill these functions will now be described in detail.

Utilization of the COBOL Data Division

Determining the format of data fields results from a thorough interpretation of the Data Division of the COBOL source program being tested. Within this Division, options exist to describe in detail both the structure of every element within a record and the relationships of these elements to one another. Those options which affect the generation of data are defined as follows:

1. Name
2. Level number to establish the relationship of one unit of data to others.
3. Size in number of characters.
4. Class indicating the type of data, i.e., alphabetic, numeric, alphanumeric.
5. Usage to establish the number system in

1976

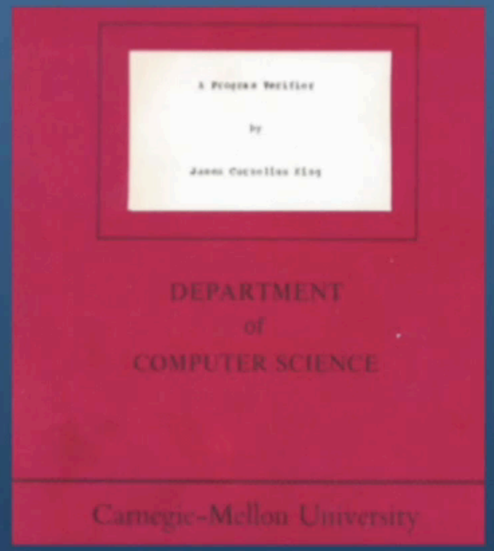


The seminal PhD thesis
by James King

James King used automated symbolic execution to capture path conditions, solved using **linear programming**

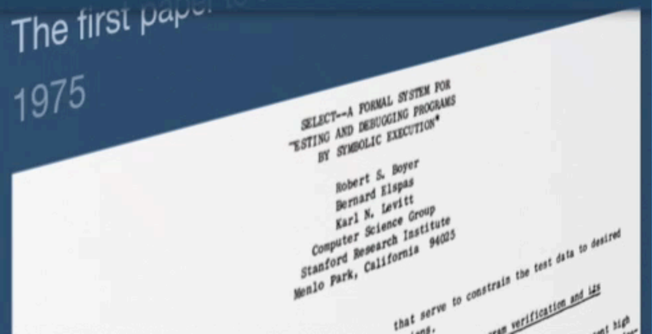
The first use of optimisation techniques in software testing and verification

1969



The first paper to use a meta-heuristic search technique

1975



“We therefore considered various alternatives that would not be subject to this limitation. The most promising of these alternatives appears to be a conjugate gradient algorithm (‘**hill climbing**’ program) that seeks to minimise a potential function constructed from the inequalities.”

The first paper to use a meta-heuristic search technique

1975

SELECT--A FORMAL SYSTEM FOR
TESTING AND DEBUGGING PROGRAMS
BY SYMBOLIC EXECUTION*

Robert S. Boyer
Bernard Elspas
Karl N. Levitt
Computer Science Group
Stanford Research Institute
Menlo Park, California 94025

Keywords: Program Testing; Test Data Generation; Symbolic Execution; Program Verification; Program Debugging; Solution of Systems of Inequalities.

Abstract

SELECT is an experimental system for assisting in the formal systematic debugging of programs. It is intended to be a compromise between an automated program proving system and the current ad hoc de-

that serve to co
regions.

A. Mechanical p
shortcomings

Much attention
cost of producin
quency of latent
effort has been

1990

Automated Software Test Data Ge

BOGDAN KOREL, MEMBER, IEEE

At about the same time, Miller and Spooner were also experimenting with **optimisation-based approaches for generating test data** (which they refer to as ‘test selection’ in the sense that they ‘select’ from the input space, which, in the more recent literature we would refer to as ‘test data generation’).

Application of genetic algorithms to software testing

Authors S Xanthakis, C Ellis, C Skourlas, A Le Gall, S Katsikas, K Karapoulios

Publication date 1992/12/7

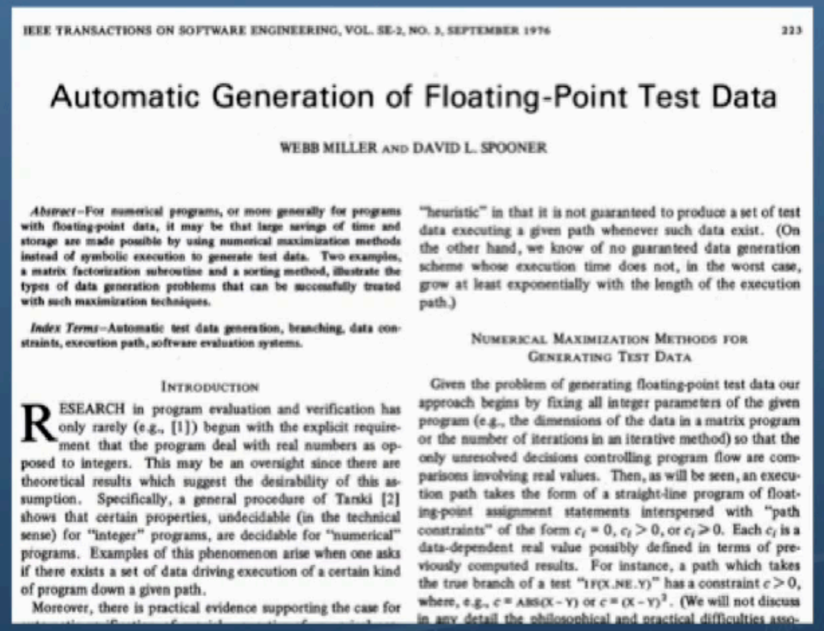
Journal Proceedings of the 5th International Conference on Software

Pages 625-636

Total citations

The use of optimisation-based approach for 'test selection'

1976



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-2, NO. 3, SEPTEMBER 1976 223

Automatic Generation of Floating-Point Test Data

WEBB MILLER AND DAVID L. SPOONER

Abstract—For numerical programs, or more generally for programs with floating-point data, it may be that large savings of time and storage are made possible by using numerical maximization methods instead of symbolic execution to generate test data. Two examples, a matrix factorization subroutine and a sorting method, illustrate the types of data generation problems that can be successfully treated with such maximization techniques.

Index Terms—Automatic test data generation, branching, data constraints, execution path, software evaluation systems.

Introduction

RESEARCH in program evaluation and verification has only rarely (e.g., [1]) begun with the explicit requirement that the program deal with real numbers as opposed to integers. This may be an oversight since there are theoretical results which suggest the desirability of this assumption. Specifically, a general procedure of Tarski [2] shows that certain properties, undecidable (in the technical sense) for "integer" programs, are decidable for "numerical" programs. Examples of this phenomenon arise when one asks if there exists a set of data driving execution of a certain kind of program down a given path.

Moreover, there is practical evidence supporting the case for "heuristic" in that it is not guaranteed to produce a set of test data executing a given path whenever such data exist. (On the other hand, we know of no guaranteed data generation scheme whose execution time does not, in the worst case, grow at least exponentially with the length of the execution path.)

Numerical Maximization Methods for Generating Test Data

Given the problem of generating floating-point test data our approach begins by fixing all integer parameters of the given program (e.g., the dimensions of the data in a matrix program or the number of iterations in an iterative method) so that the only unresolved decisions controlling program flow are comparisons involving real values. Then, as will be seen, an execution path takes the form of a straight-line program of floating-point assignment statements interspersed with "path constraints" of the form $c_i = 0$, $c_i > 0$, or $c_i \geq 0$. Each c_i is a data-dependent real value possibly defined in terms of previously computed results. For instance, a path which takes the true branch of a test "IF(X.NE.Y)" has a constraint $c > 0$, where, e.g., $c = \text{ABS}(X - Y)$ or $c = (X - Y)^2$. (We will not discuss in any detail the philosophical and practical difficulties asso-

1970 1975 1980 1985 1990 2000 2006

It appears that SBST research lay dormant for at approximately a decade until the work of Korel, which introduced a practical test data generation approach, the **Alternating Variable Method** (AVM), based on hill climbing.

has been paid to the present high
producing reliable software, and to the fre-
quency of latent bugs even after a great deal of
effort has been expended in testing and debugging.

A practical test data generation approach based on AVM

1990

870 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 16, NO. 8, AUGUST 1990

Automated Software Test Data Generation

BOGDAN KOREL, MEMBER, IEEE

Abstract—Test data generation in program testing is the process of identifying a set of test data which satisfies given testing criterion. Most of the existing test data generators [6], [8], [10], [16], [30] use symbolic evaluation to derive test data. However, in practical programs this technique frequently requires complex algebraic manipulations, especially in the presence of arrays. In this paper we present an alternative approach of test data generation which is based on actual execution of the program under test, function minimization methods, and dynamic data flow analysis. Test data are developed for the program using actual values of input variables. When the program is executed, the program execution flow is monitored. If during program execution an undesirable execution flow is observed (e.g., the "actual" path does not correspond to the selected control path) then function minimization search algorithms are used to automatically locate the values of input variables for which the selected path is traversed. In addition, dynamic data flow analysis is used to determine those input variables responsible for the undesirable program behavior, leading to significant speed-up of the search process. The approach of generating test data is then extended to programs with dynamic data structures, and a search

tomatically generate test data that meet the selected criterion. The basic operation of the pathwise generator consists of the following steps: program control flow graph construction, path selection, and test data generation. The path selector automatically identifies a set of paths (e.g., near-minimal set of paths) to satisfy selected testing criterion. Once a set of test paths is determined, then for every path in this set the test generator derives input data that results in the execution of the selected path.

Most of the pathwise test data generators [6], [8], [10], [16], [30] use symbolic evaluation to derive input data. Symbolic evaluation involves executing a program using symbolic values of variables instead of actual values. Once a path is selected, symbolic evaluation is used to generate a path constraint, which consists of a set of equalities and inequalities on the program's input vari-

The first use of genetic algorithm to develop whole test suites

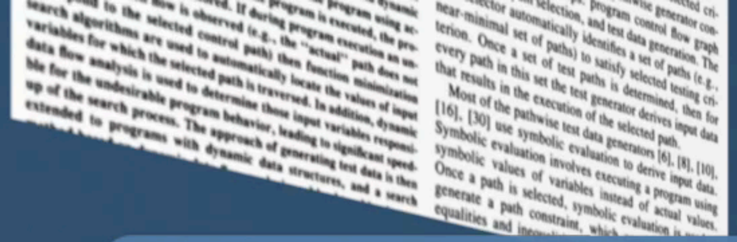
1992

Application of genetic algorithms to software testing

I Xanthakis, C Ellis, C Skourlas, A Le Gall, S Katsikas, K Karapoulios

Proceedings of the 5th International Conference on Software Engineering and its Applications

The **first use of genetic algorithms** for software engineering problems is usually attributed also to the field of SBST, with the work of Xanthakis et al., who introduced a genetic algorithm to develop whole test suites.

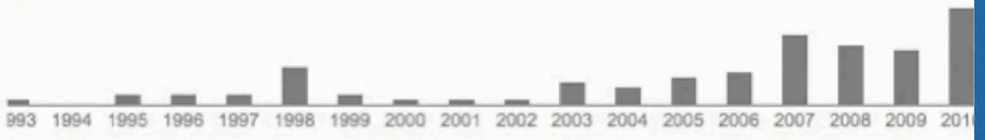


The first use of genetic algorithm to develop whole test suites

1992

Application of genetic algorithms to software testing

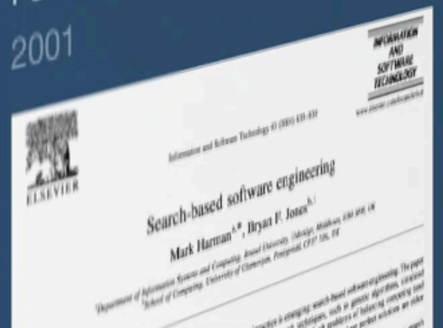
Authors	S Xanthakis, C Ellis, C Skourlas, A Le Gall, S Katsikas, K Karapoulios
Publication date	1992/12/7
Journal	Proceedings of the 5th International Conference on Software Engineering and its Applications
Pages	625-636
Total citations	Cited by 156



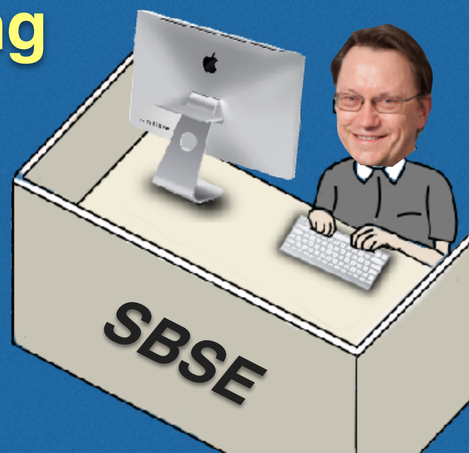
d approach for 'test selection'

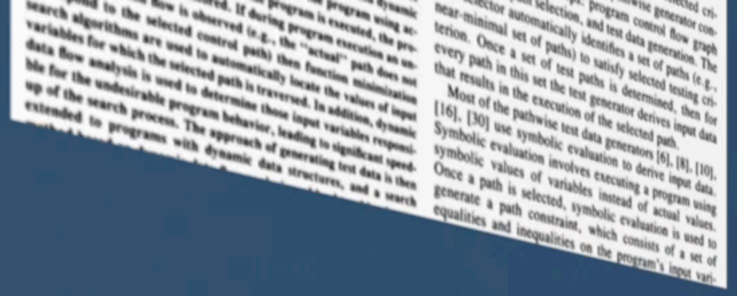
Formally established as a field of study

2001



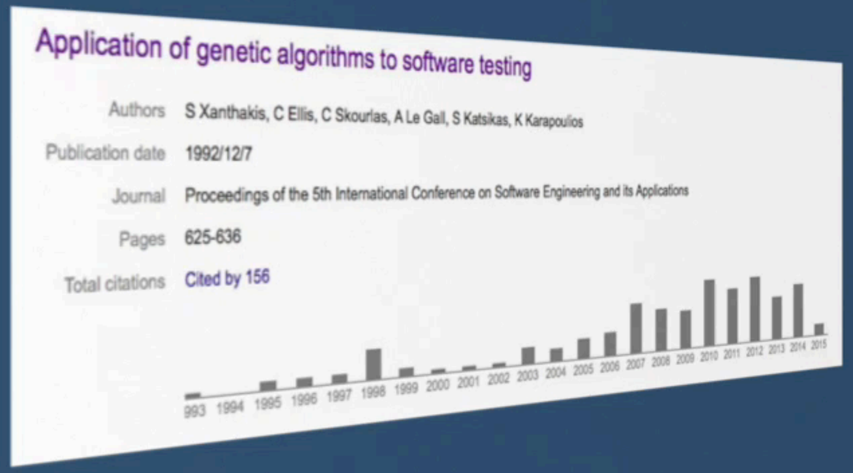
The first suggestion of **search as a universal approach to Software Engineering**





The first use of genetic algorithm to develop whole test suites

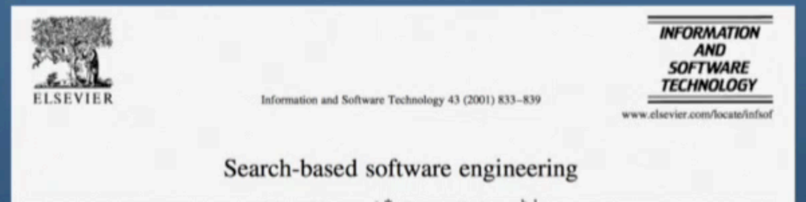
1992



d approach for 'test selection'

Formally established as a filed of study with SBSE

2001



SBSE Repository

Tests live in a search space

... covers history of SBST up to 2015

Achievements, open problems and challenges for search based software testing

Mark Harman, Yue Jia and Yuanyuan Zhang
University College London, CREST Centre, London, UK

***Abstract*—Search Based Software Testing (SBST) formulates testing as an optimisation problem, which can be attacked using computational search techniques from the field of Search Based Software Engineering (SBSE). We present an analysis of the SBST research agenda[1], focusing on the open problems and challenges of testing non-functional properties, in particular a topic we call 'Search Based Energy Testing' (SBET), Multi-objective SBST and SBST for Test Strategy Identification. We conclude with a vision of FIFIVERIFY tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FIFIVERIFY tools constitute an exciting challenge for the SBSE community that already could be within its reach.**

I. INTRODUCTION

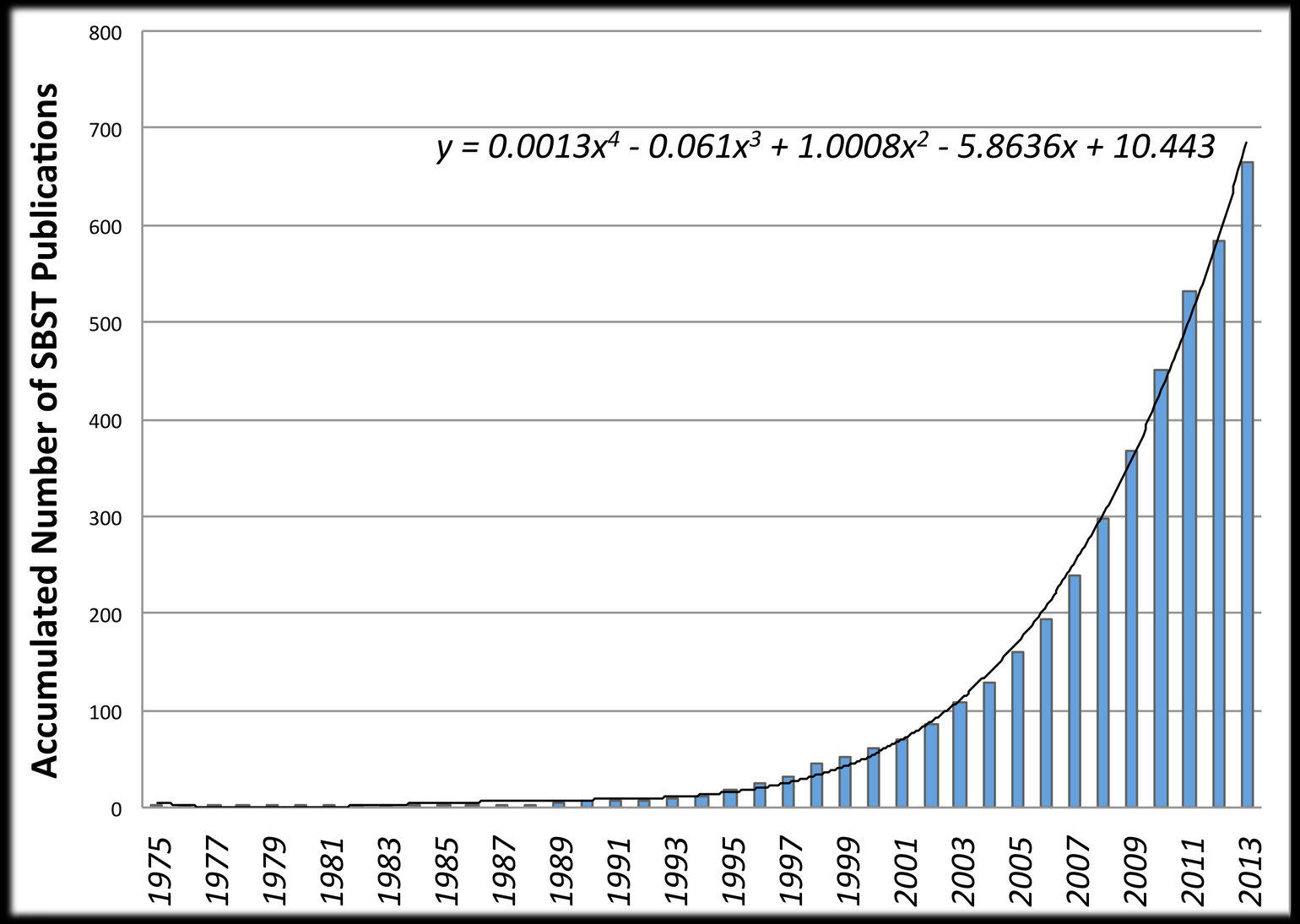
Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering (SBSE) concerned with software testing [2], [85]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typified by large complex search spaces [58]. Test objectives find natural counterparts as the fitness functions used by SBSE to guide automated

II. A BRIEF HISTORY OF SBST

Since the first paper on SBST is also likely to be the first paper on SBSE, the early history of SBST is also the early history of SBSE. SBSE is a sub-area of software engineering with origins stretching back to the 1970s but not formally established as a field of study in its own right until 2001 [51], and which only achieved more widespread acceptance and uptake many years later [38], [43], [100].

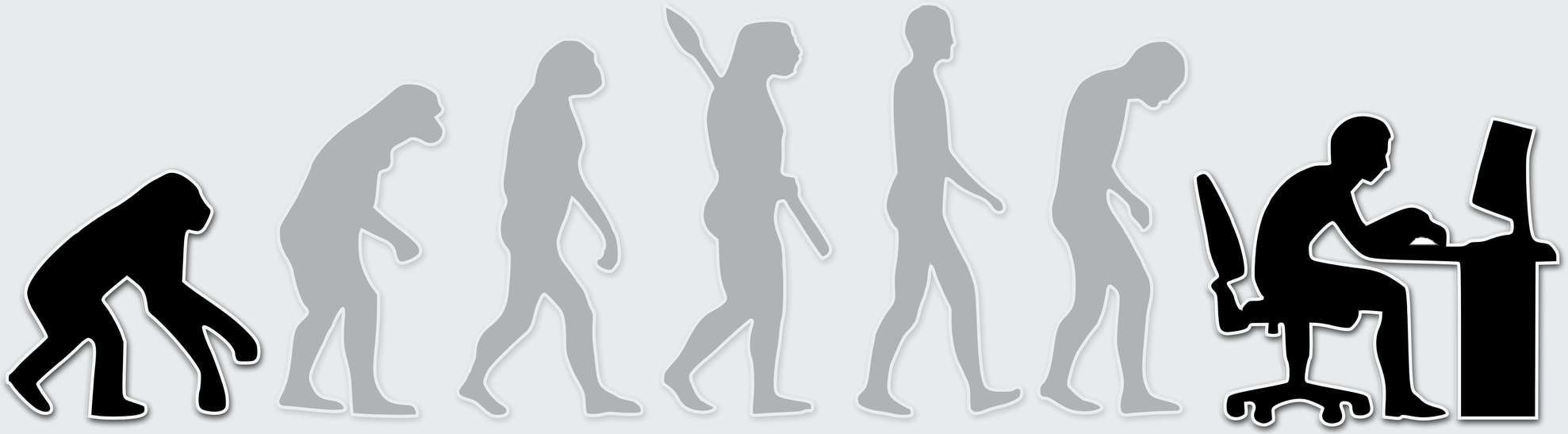
The first mention of *software optimisation* (of any kind) is almost certainly due to Ada Augusta Lovelace in 1842. Her English language translation of the article (written in Italian by Menabrae), 'Sketch of the Analytical Engine Invented by Charles Babbage' includes seven entries, labelled 'Note A' to 'Note G' and initialed 'A.A.L'. Her notes constituted an article themselves (and occupied three quarters of the whole document). In these notes we can see perhaps the first recognition of the need for software optimisation and source code analysis and manipulation (a point argued in more detail elsewhere [44]):

"*In almost every computation a great variety of*



Polynomial yearly rise in the number of papers Search Based Software Testing

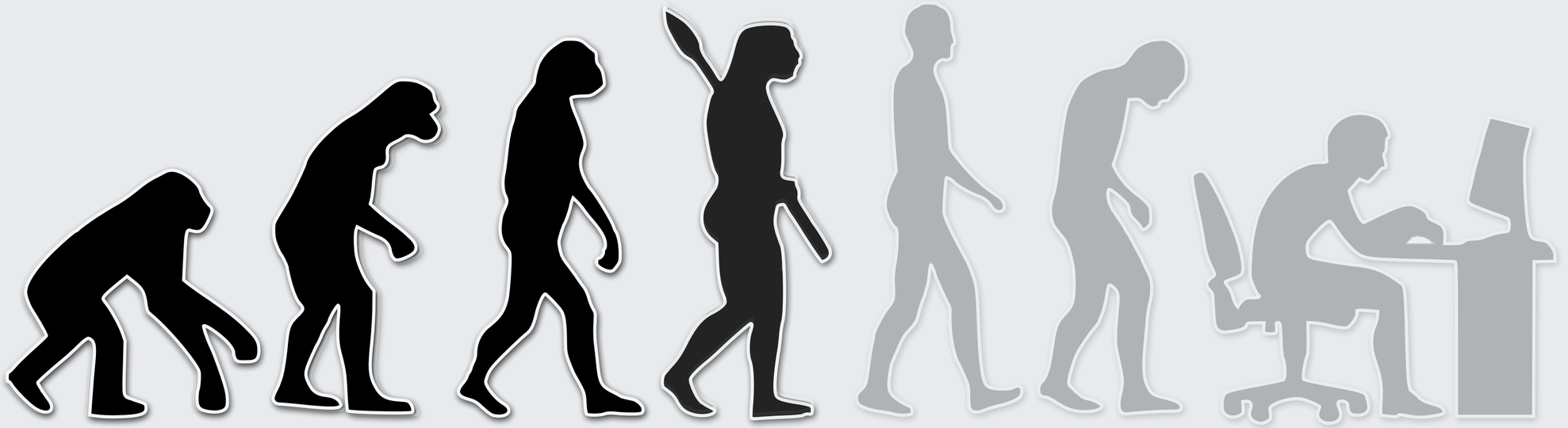
E2E System Level Testing



Random
Fuzzer

Human
Testers

E2E System Level Testing



Random
Fuzzer

sapienz

Human
Testers

Sapienz

Motif Genes

Guided Search

Multi Objective

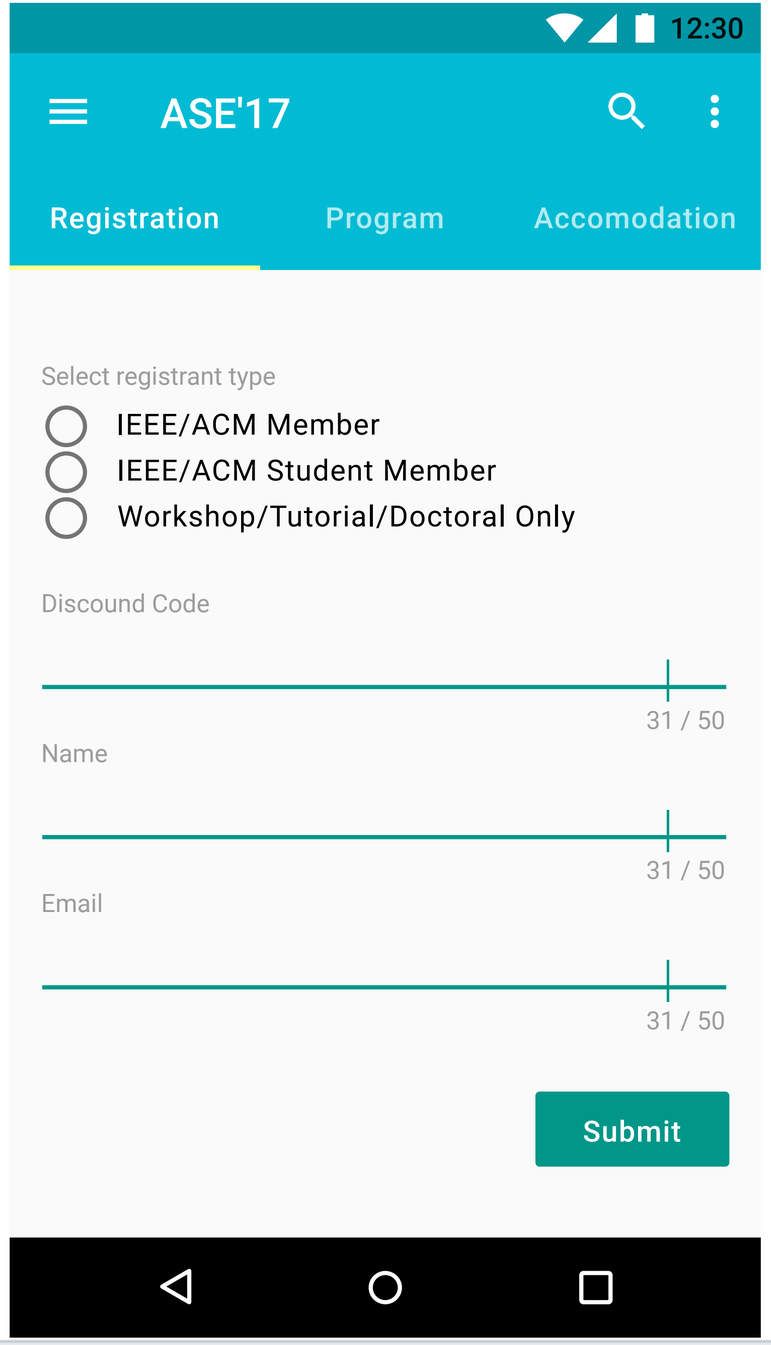
Sapienz: Smarter Fuzzing

Motif Genes

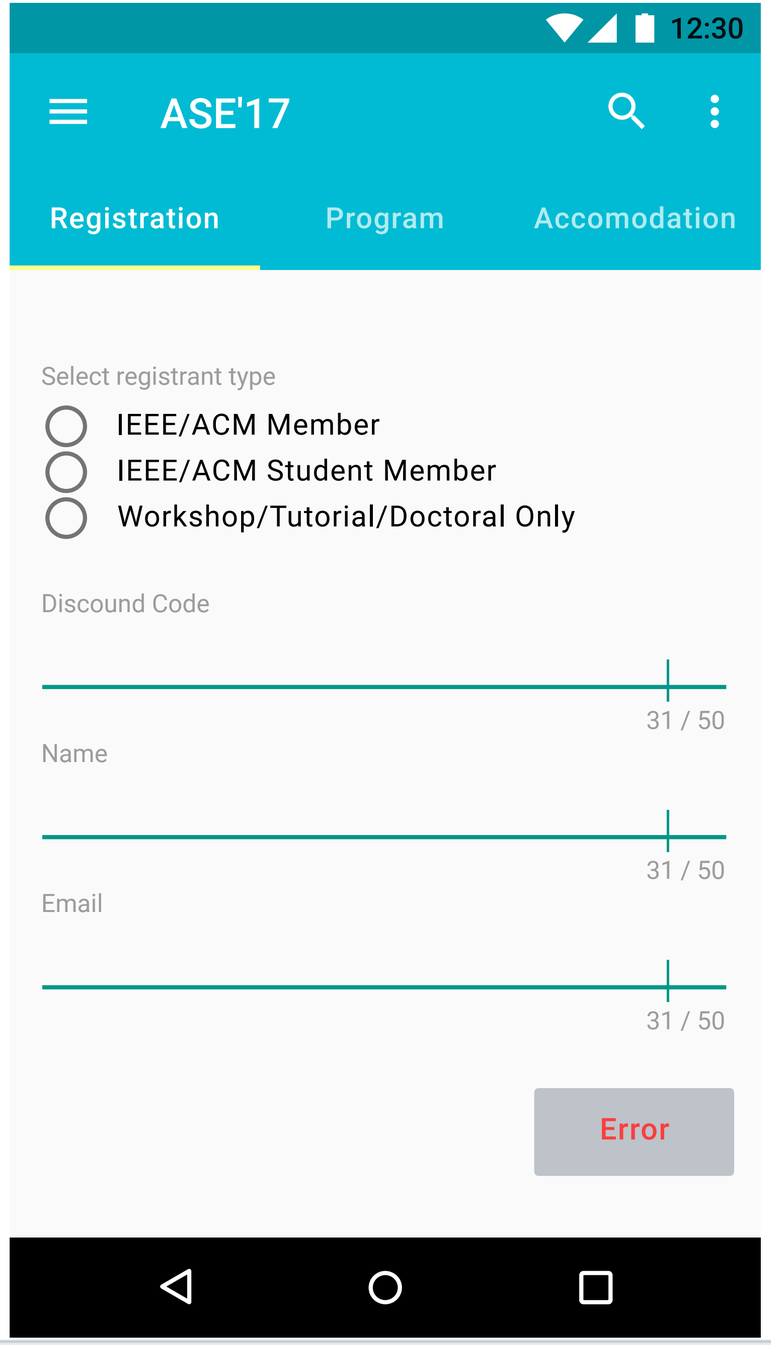
Guided Search

Multi Objective

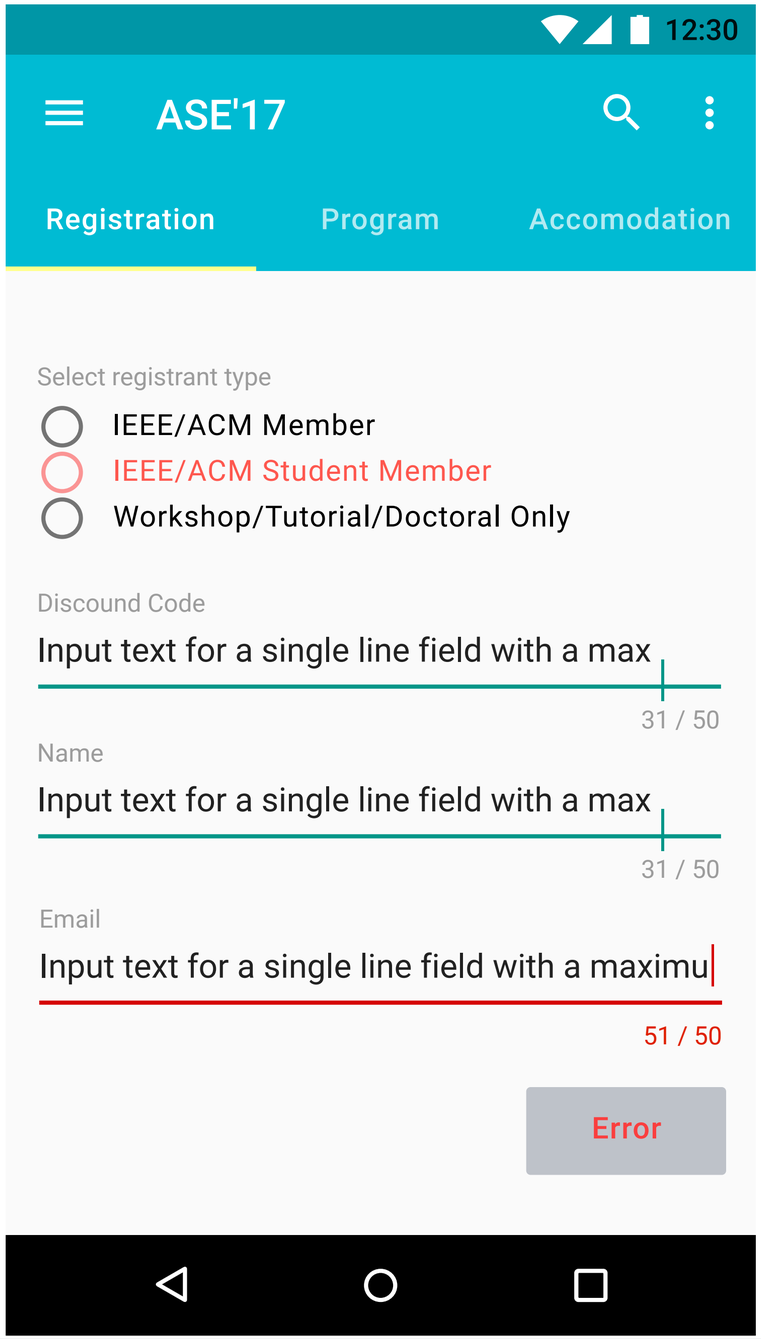
Learning motif events from human


12:30
ASE'17
Registration
Program
Accomodation
Select registrant type
IEEE/ACM Member
IEEE/ACM Student Member
Workshop/Tutorial/Doctoral Only
Discound Code
31 / 50
Name
31 / 50
Email
31 / 50
Submit

Sample app

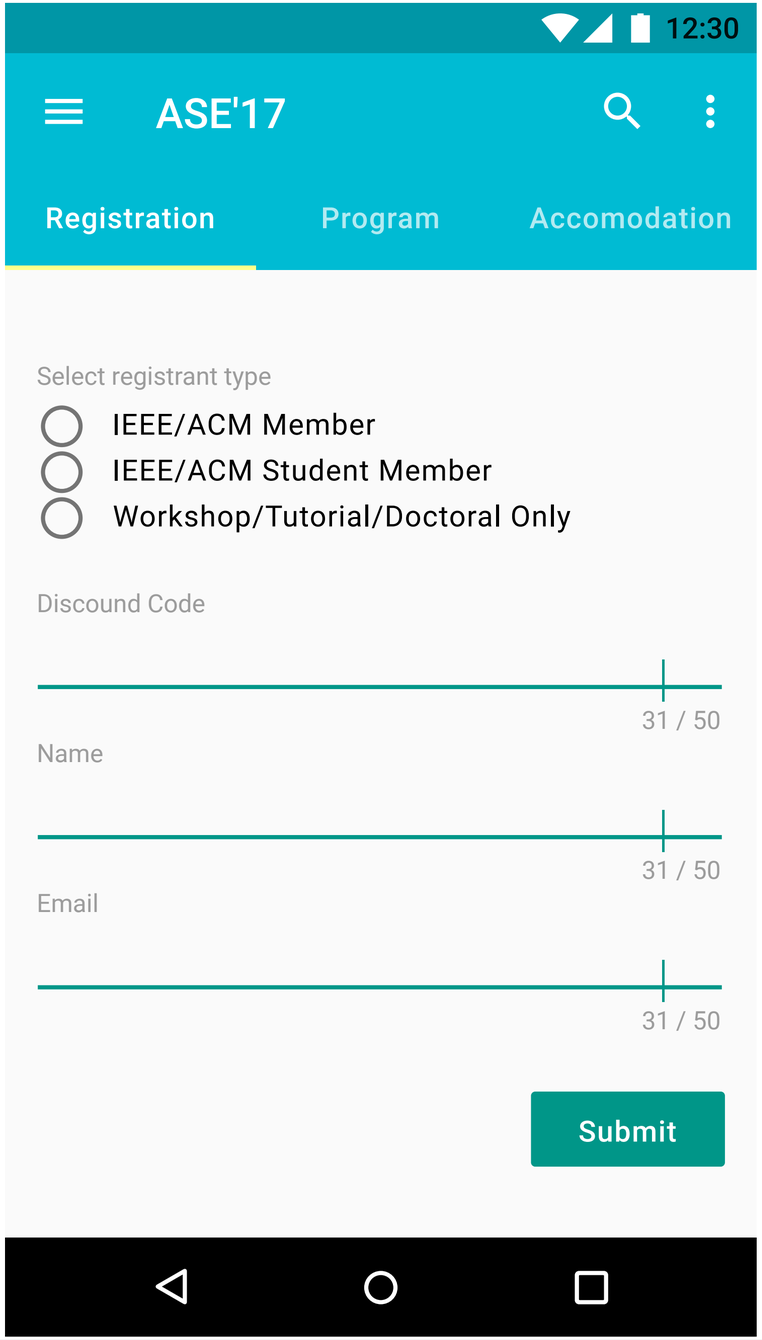

12:30
ASE'17
Registration
Program
Accomodation
Select registrant type
IEEE/ACM Member
IEEE/ACM Student Member
Workshop/Tutorial/Doctoral Only
Discound Code
31 / 50
Name
31 / 50
Email
31 / 50
Error

Sample app


12:30
ASE'17
Registration
Program
Accomodation
Select registrant type
IEEE/ACM Member
IEEE/ACM Student Member
Workshop/Tutorial/Doctoral Only
Discound Code
Input text for a single line field with a max
31 / 50
Name
Input text for a single line field with a max
31 / 50
Email
Input text for a single line field with a maximu
51 / 50
Error

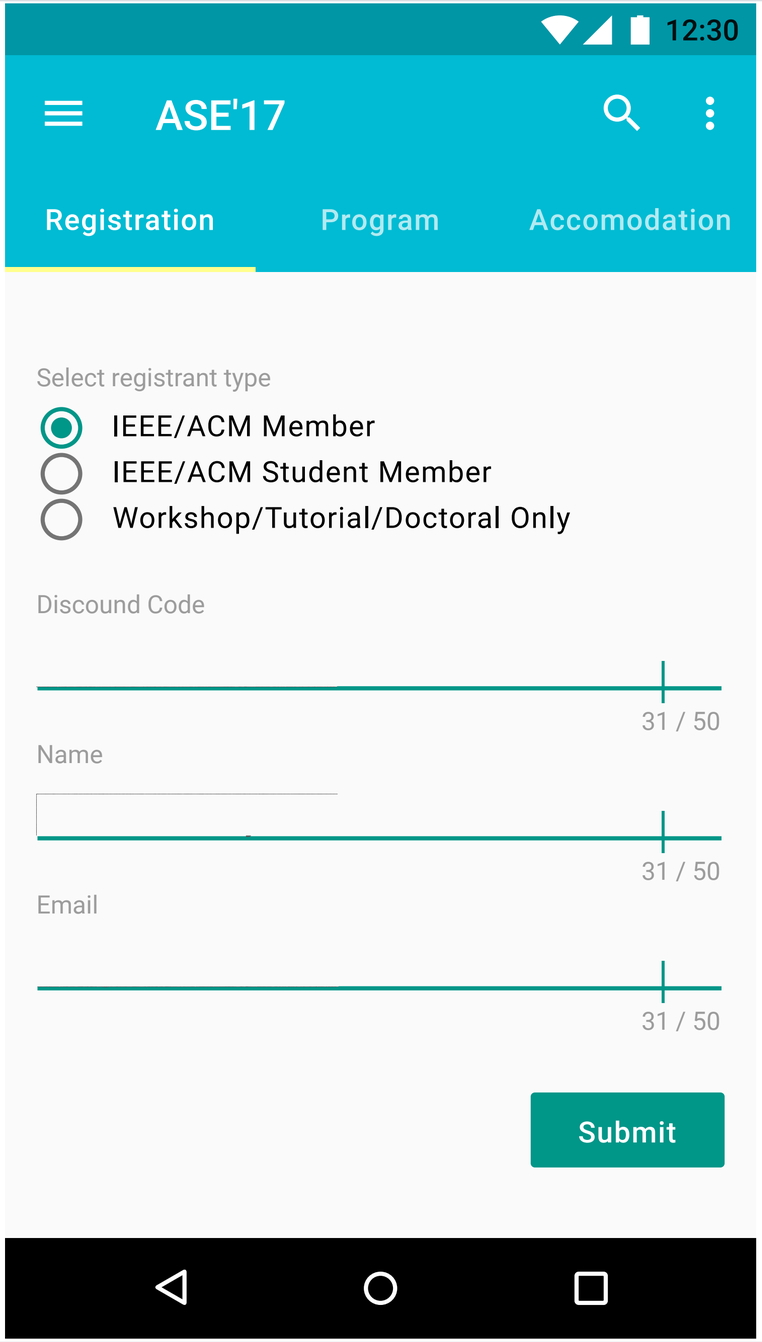
Sample app

Motif Event



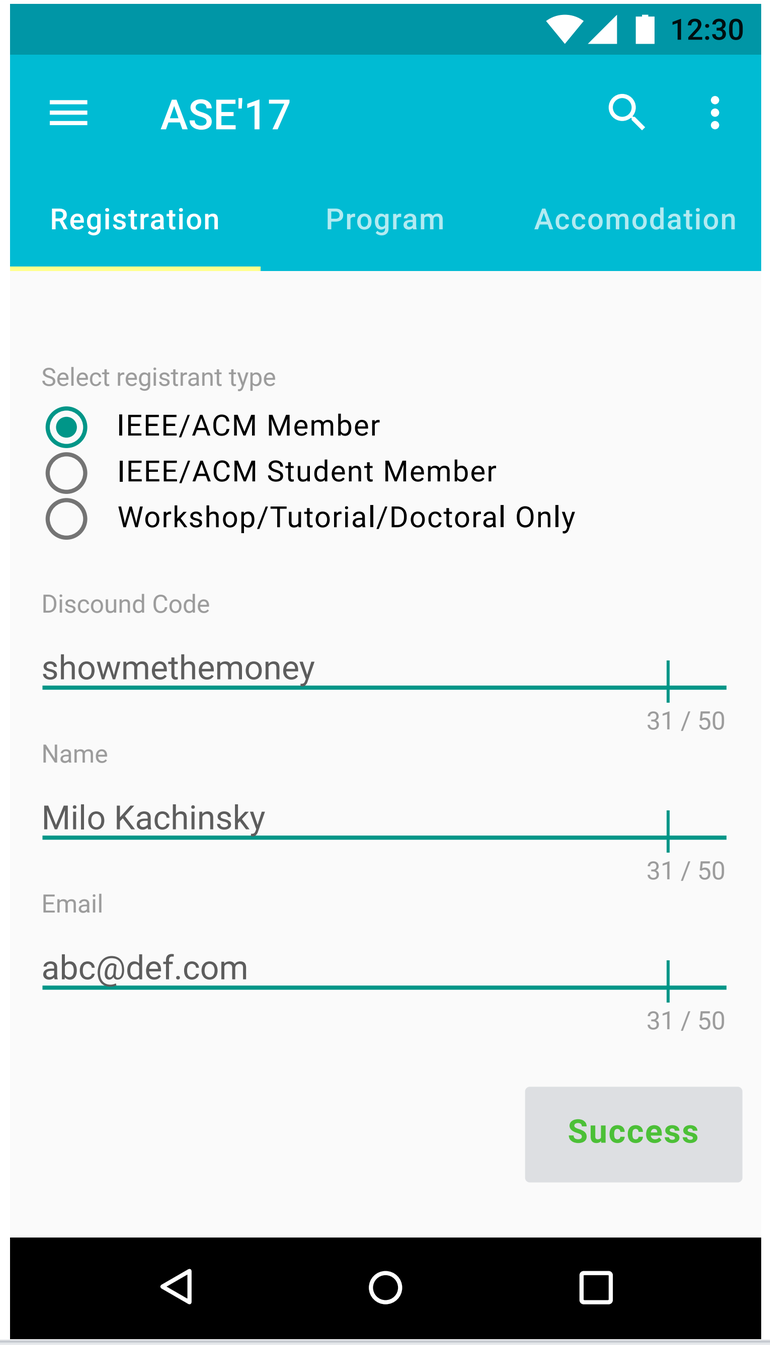
Sample app

Motif Event



Sample app

Motif Event



Sample app

Sapienz

Motif Genes

Guided Search

Multi Objective

Sapienz: Smarter Fuzzing

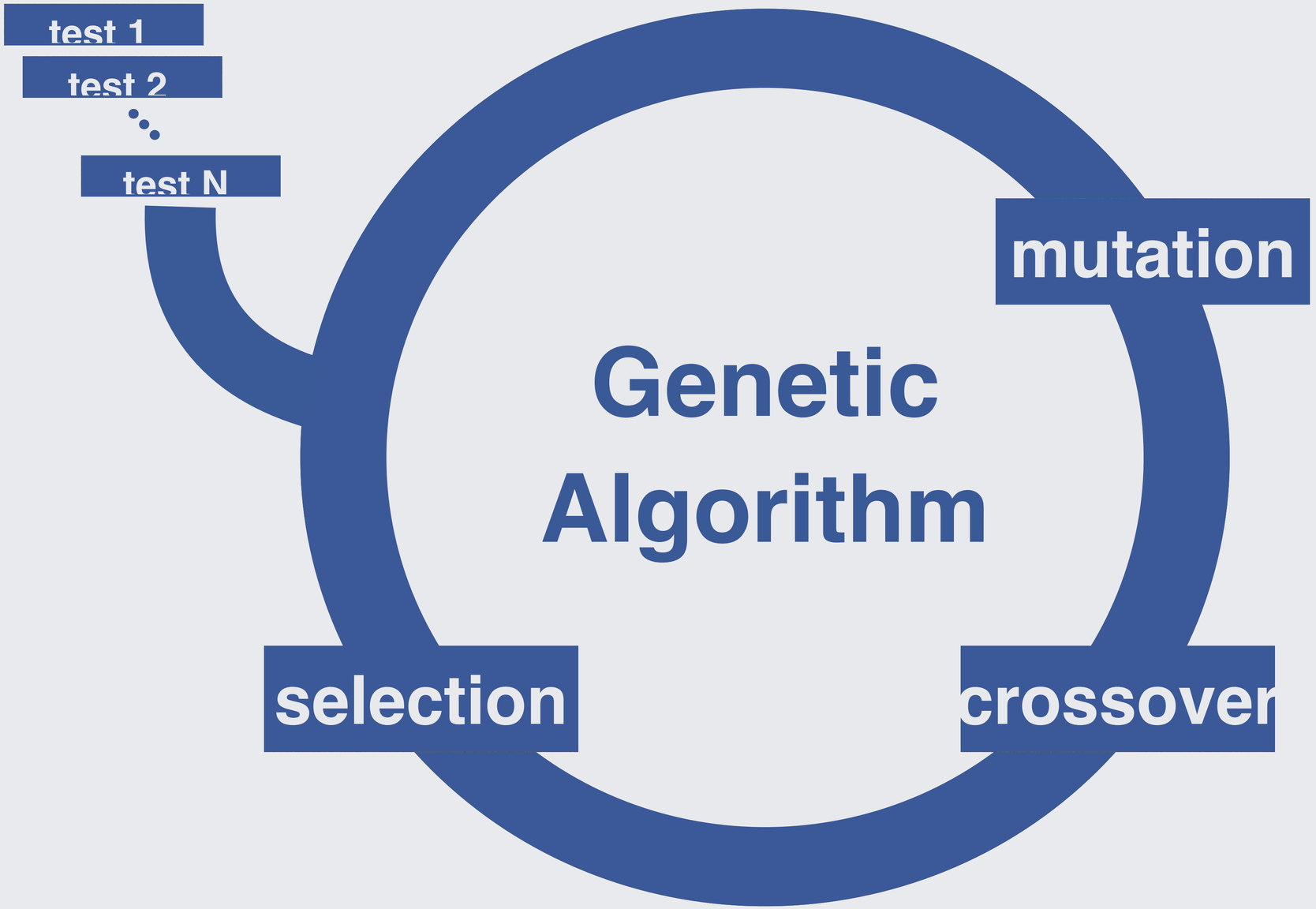
Motif Genes

Guided Search

Multi Objective

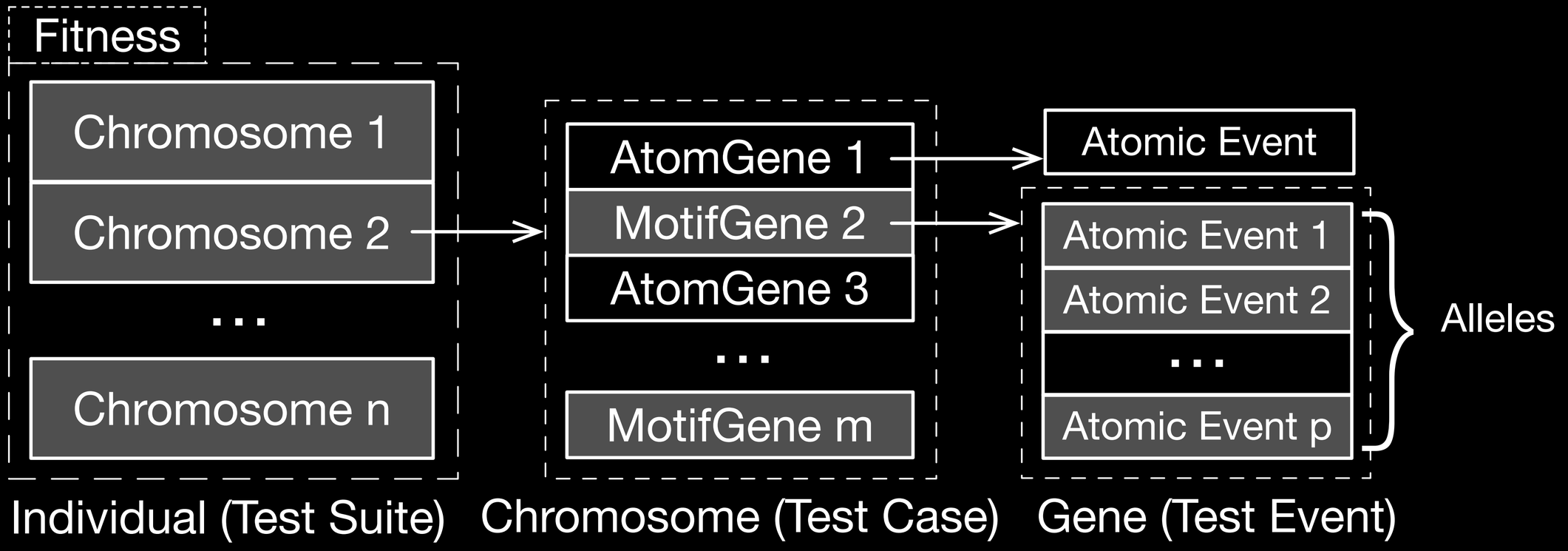
Sapienz: Smarter Fuzzing

Guided Search



Individual Representation

whole suite evolution



Sapienz: Smarter Fuzzing

Motif Genes

Guided Search

Multi Objective

Sapienz: Smarter Fuzzing

Multi Objective

Multi Objective

Motif Genes

Guided Search

Coverage

Fault Revelation

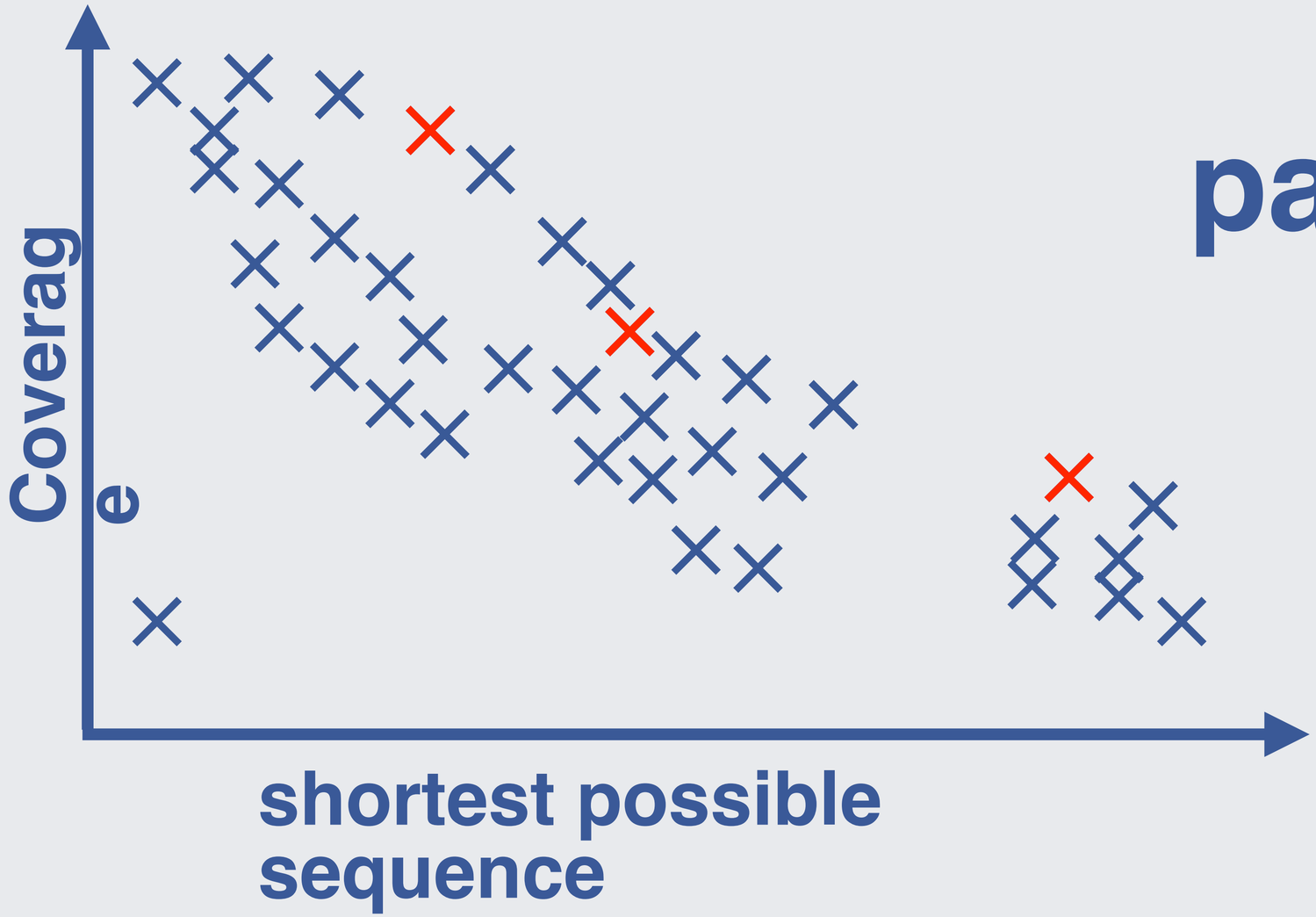
Sequence Length

Sapienz: Smarter Fuzzing

Multi Objective

coverage length

pareto front



Sapienz @ Facebook scale


facebook

Facebook Scale

1,230,000,000 *daily* active users

1,000,000+ source control commands per day

100,000+ commits per week

sapienz

Started work February 2017

Deployed in September 2017

Fully integrated into CI system

Auto Localises

Auto comments in code review

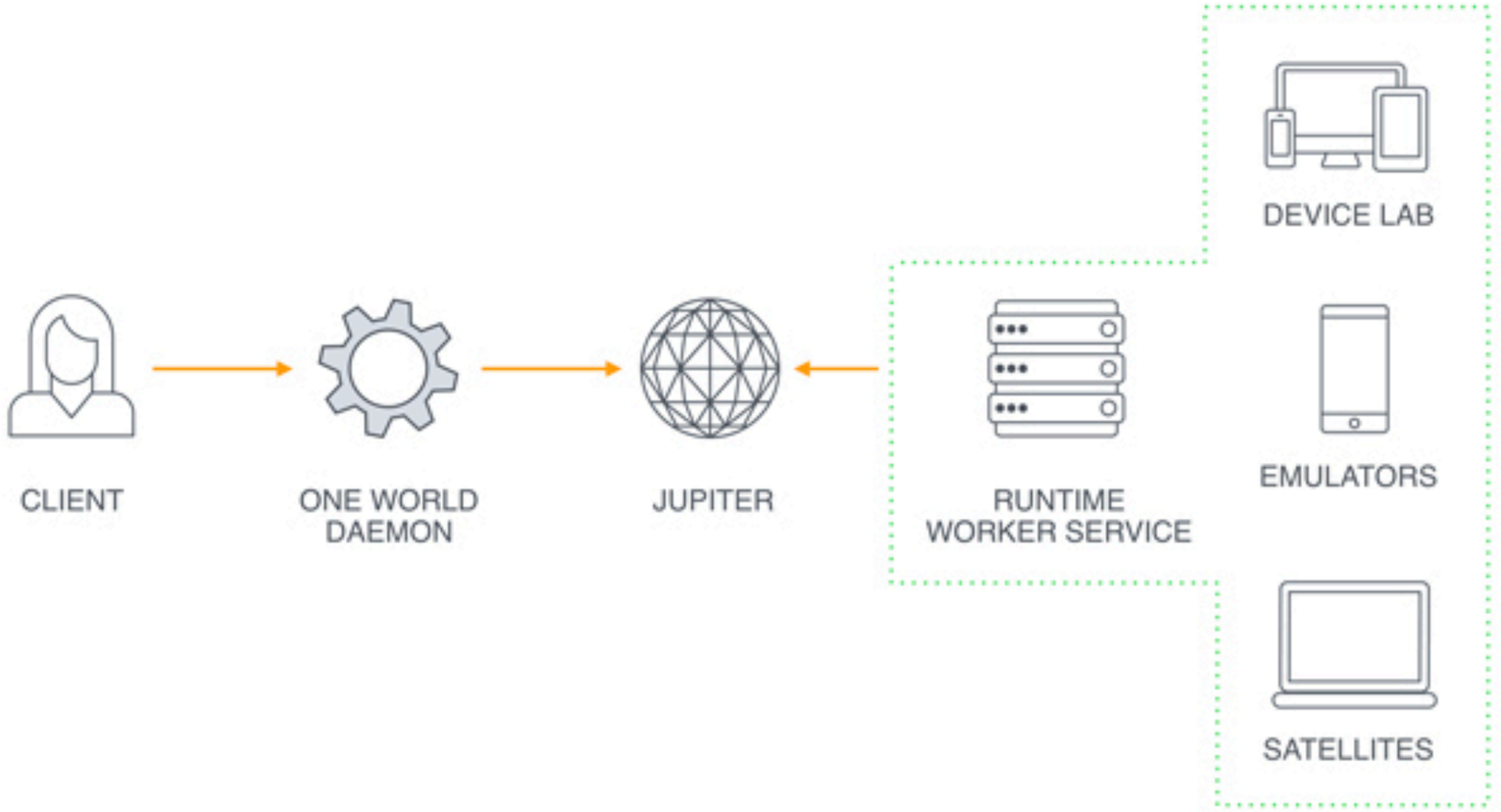
Auto detects fixes

Helps Facebook to move faster

Infra for CI

OneWorld

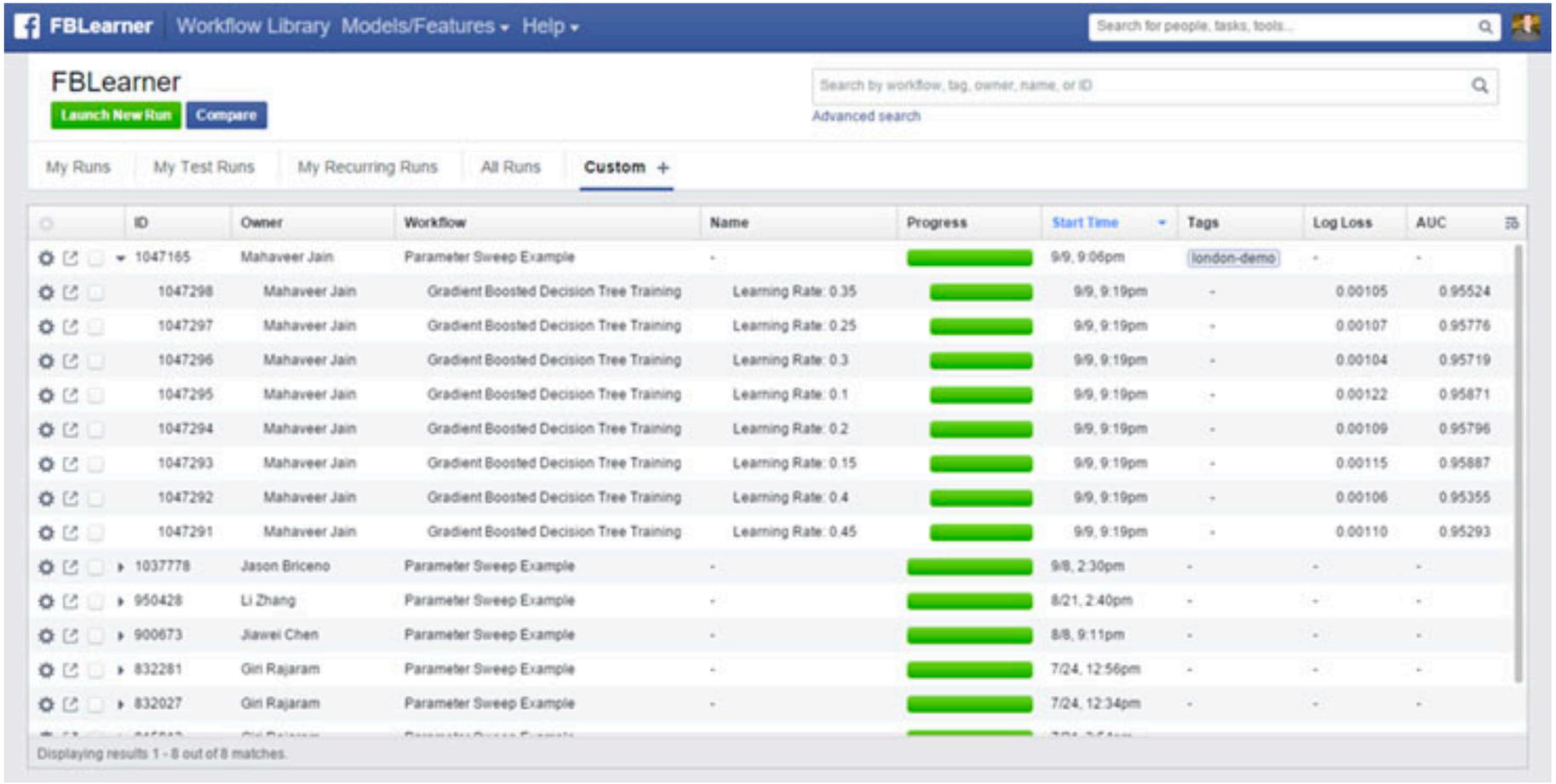
Managing resources for large-scale testing



Infra for CI

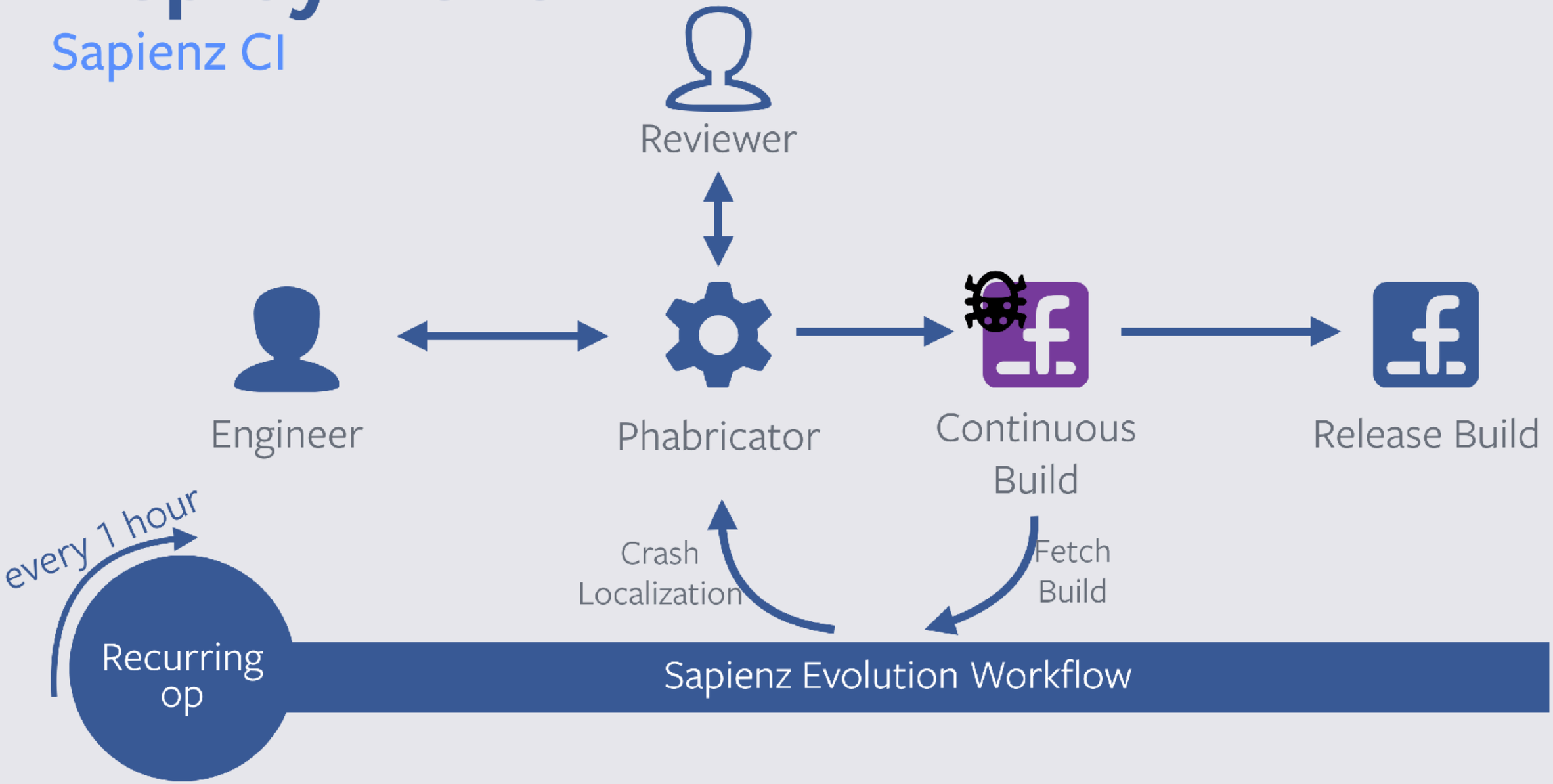
FBLearner

Introducing FBLearner Flow: Facebook's AI backbone

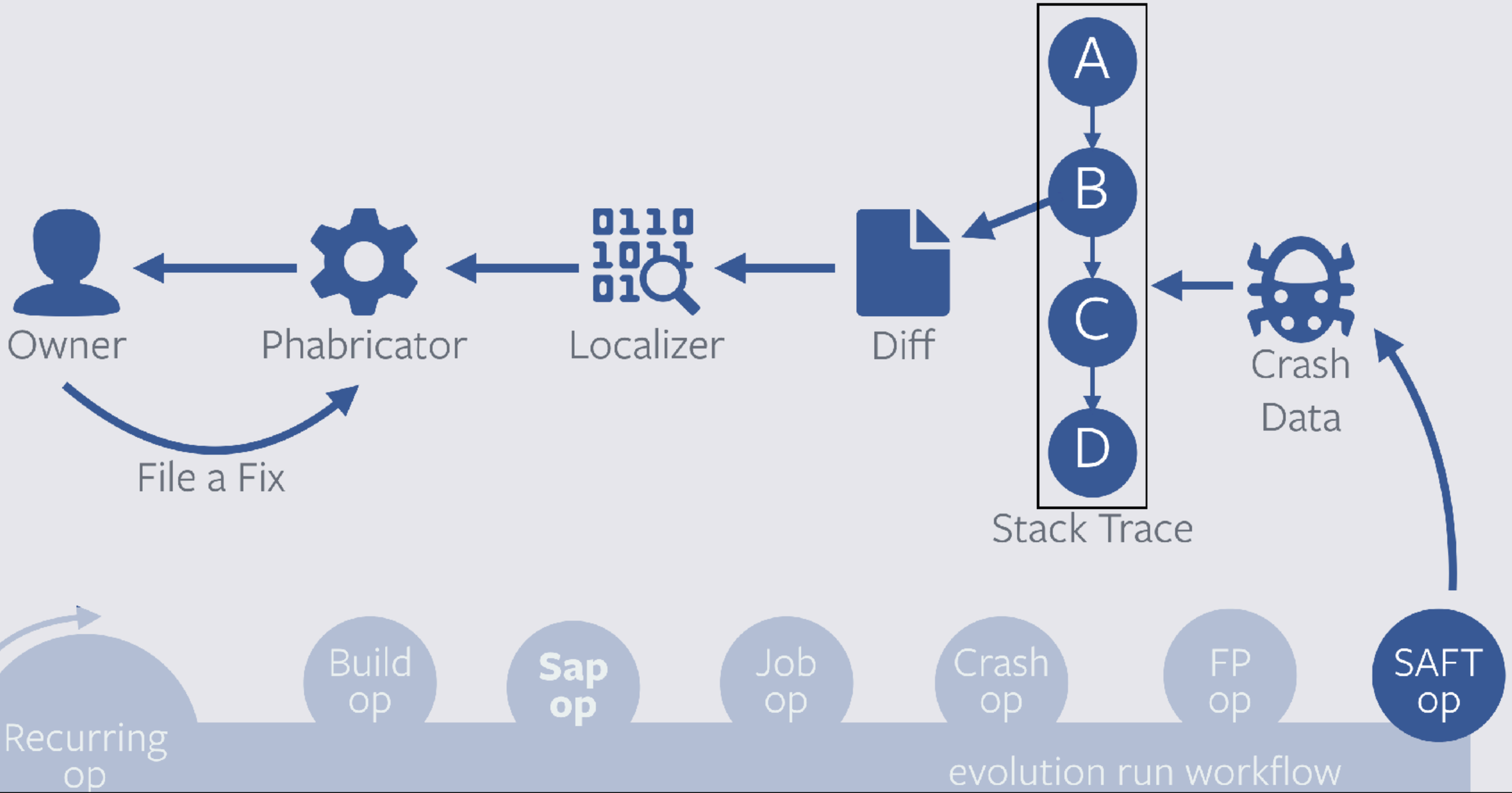


Deployment

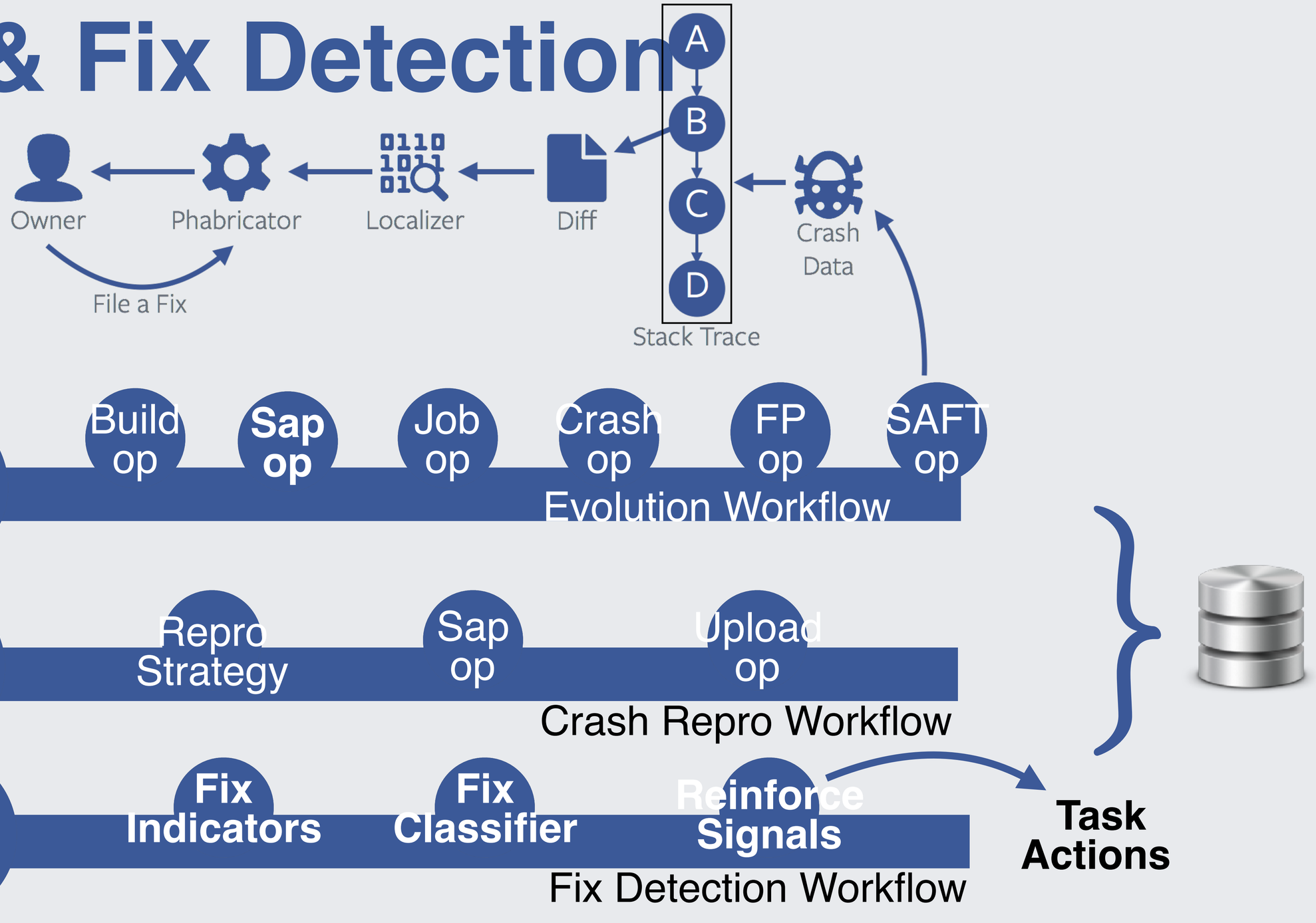
Sapienz CI



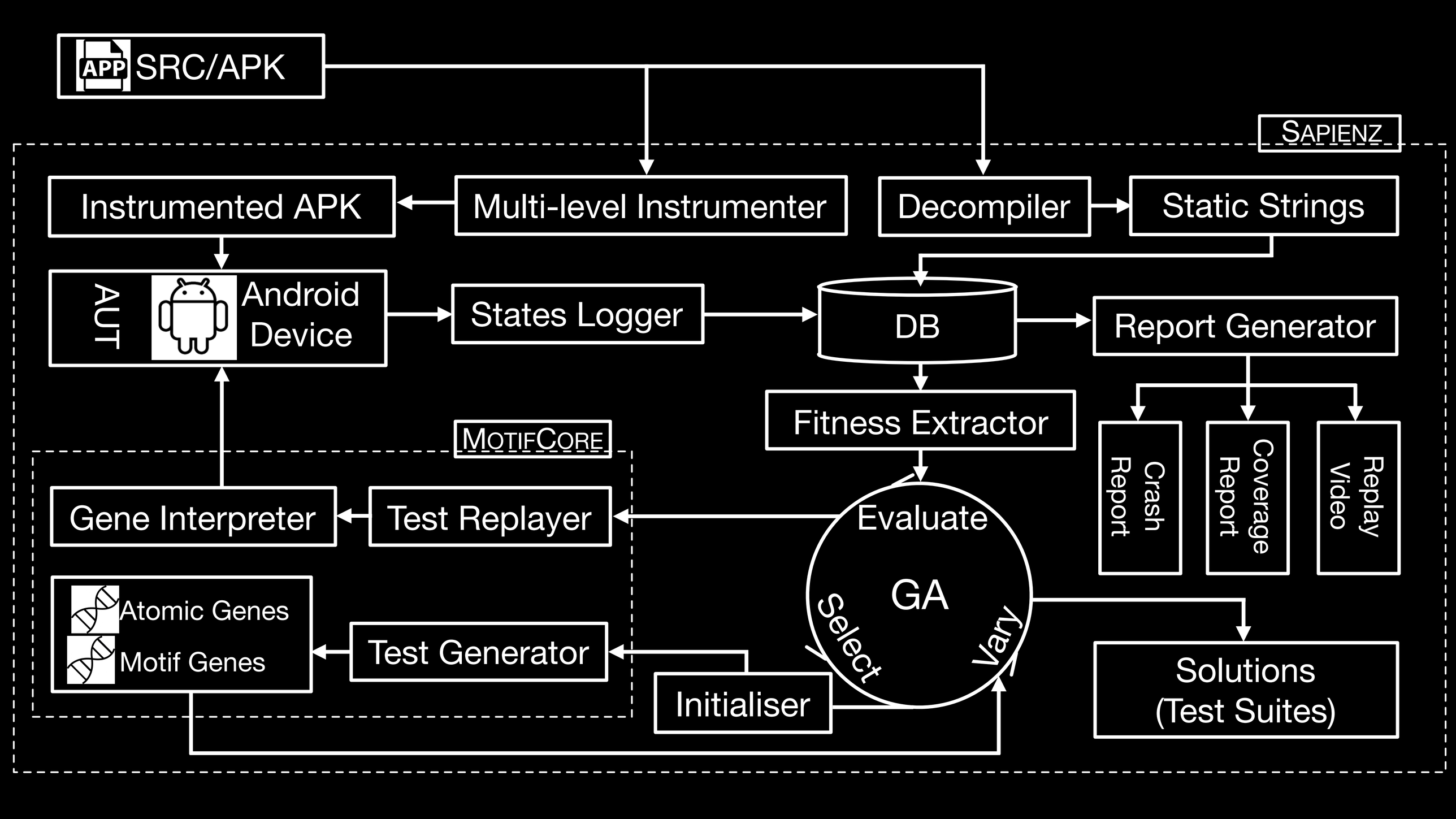
SAFT Triage Process



Repro & Fix Detection

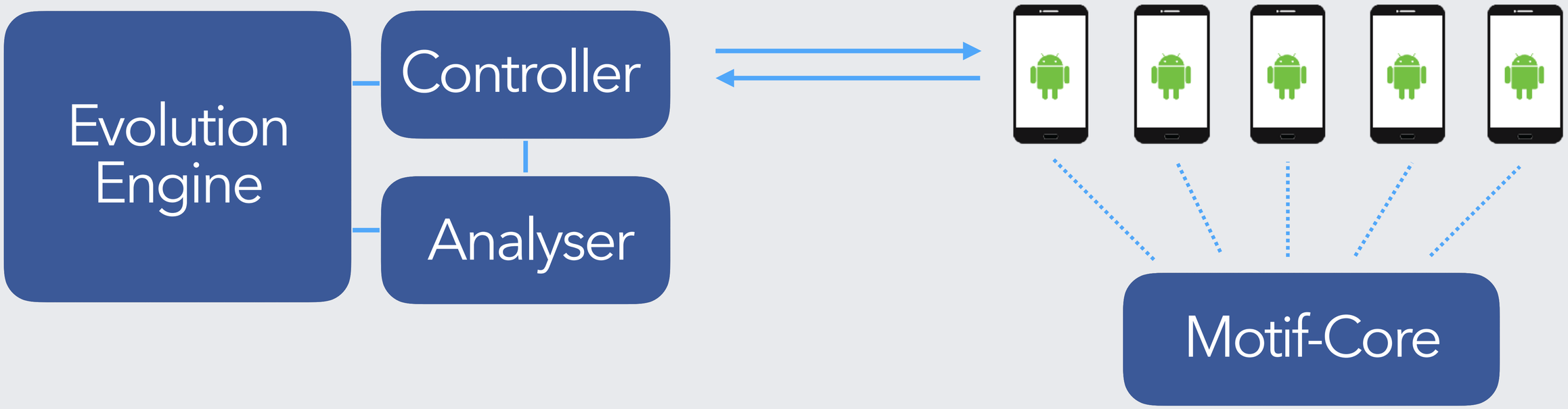


Sapienz Breakdown

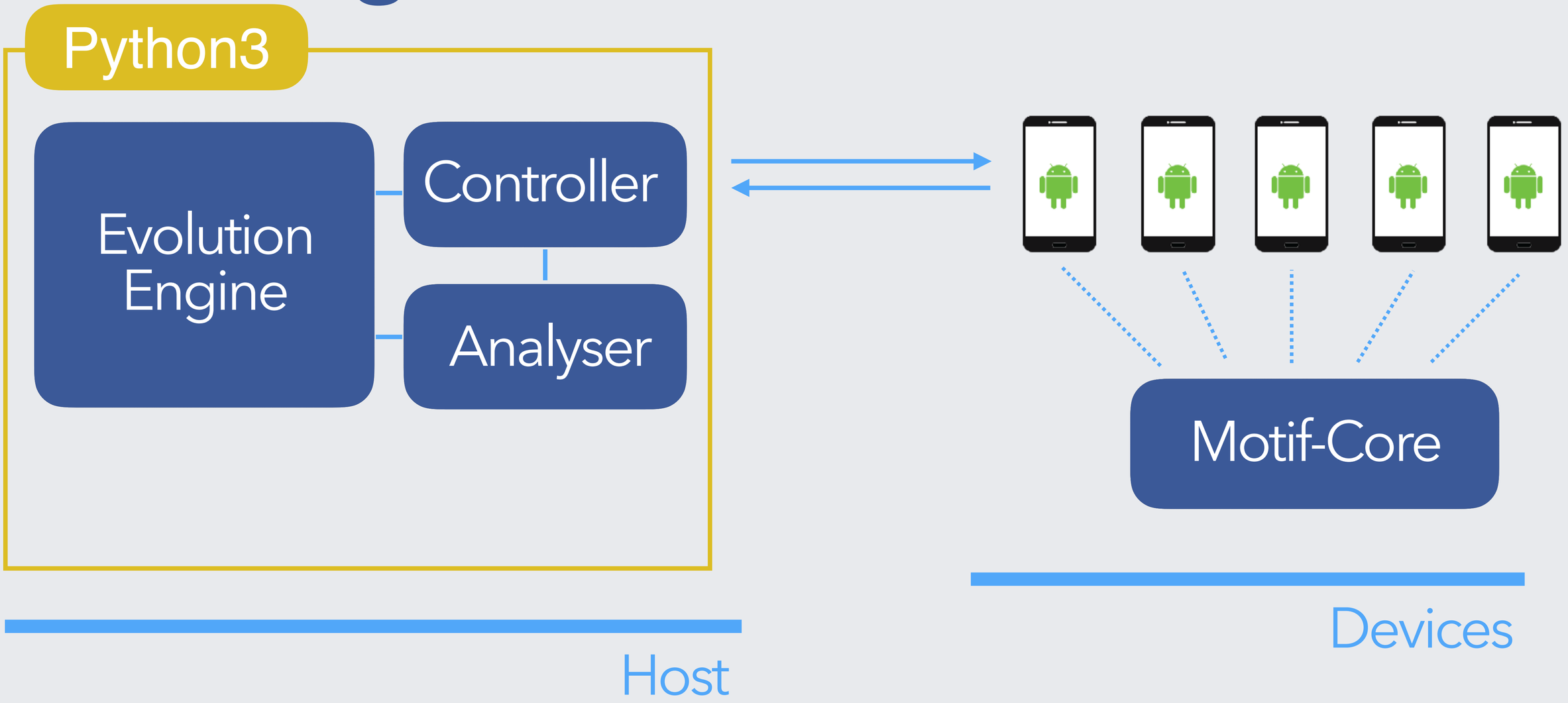

APP
SRC/APK
SAPIENZ
Instrumented APK
Multi-level Instrumenter
Decompiler
Static Strings
AUT
Android Device
States Logger
DB
Report Generator
Fitness Extractor
MOTIFCORE
Crash Report
Coverage Report
Replay Video
Gene Interpreter
Test Replayer
Evaluate
GA
Select
Vary
Atomic Genes
Motif Genes
Test Generator
Initialiser
Solutions (Test Suites)

FB Integration

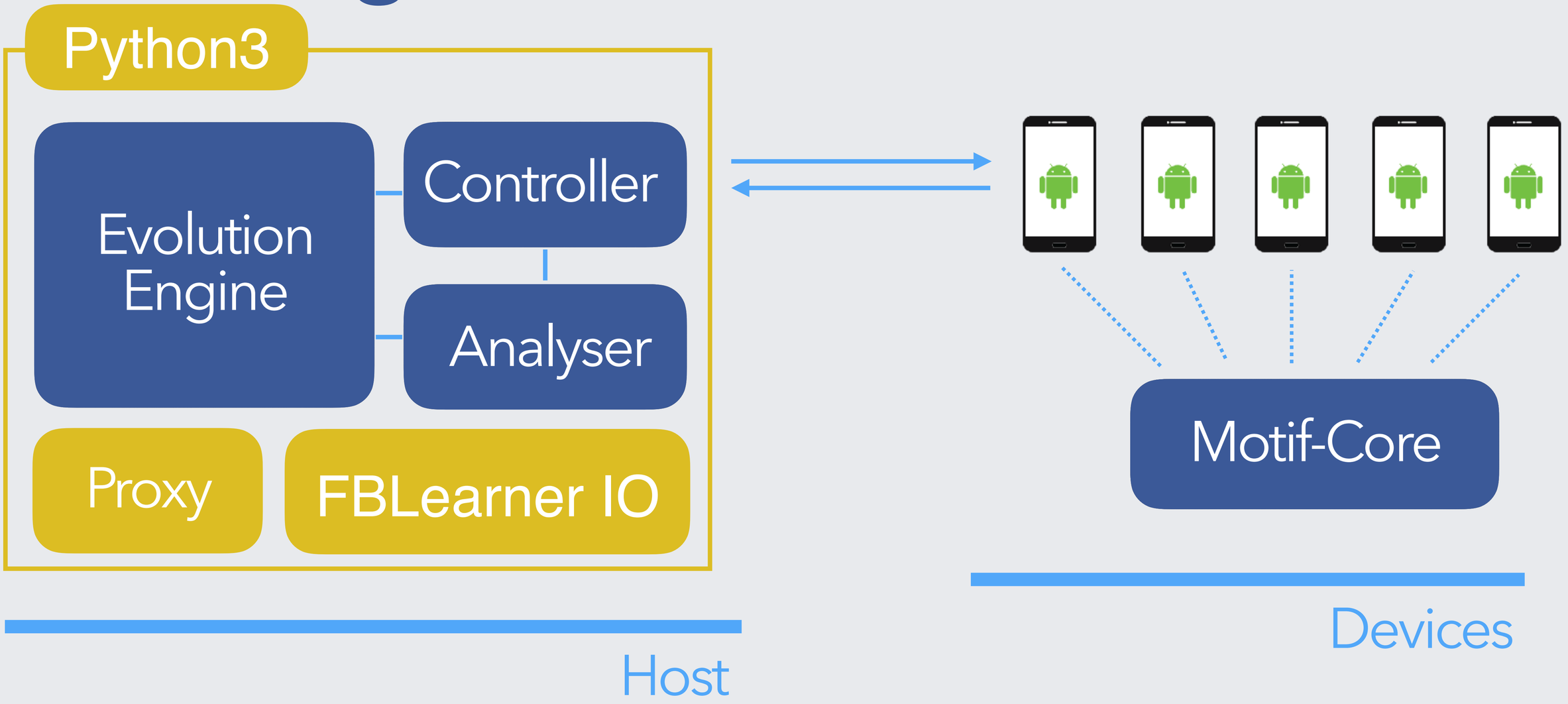
Original Sapienz architecture



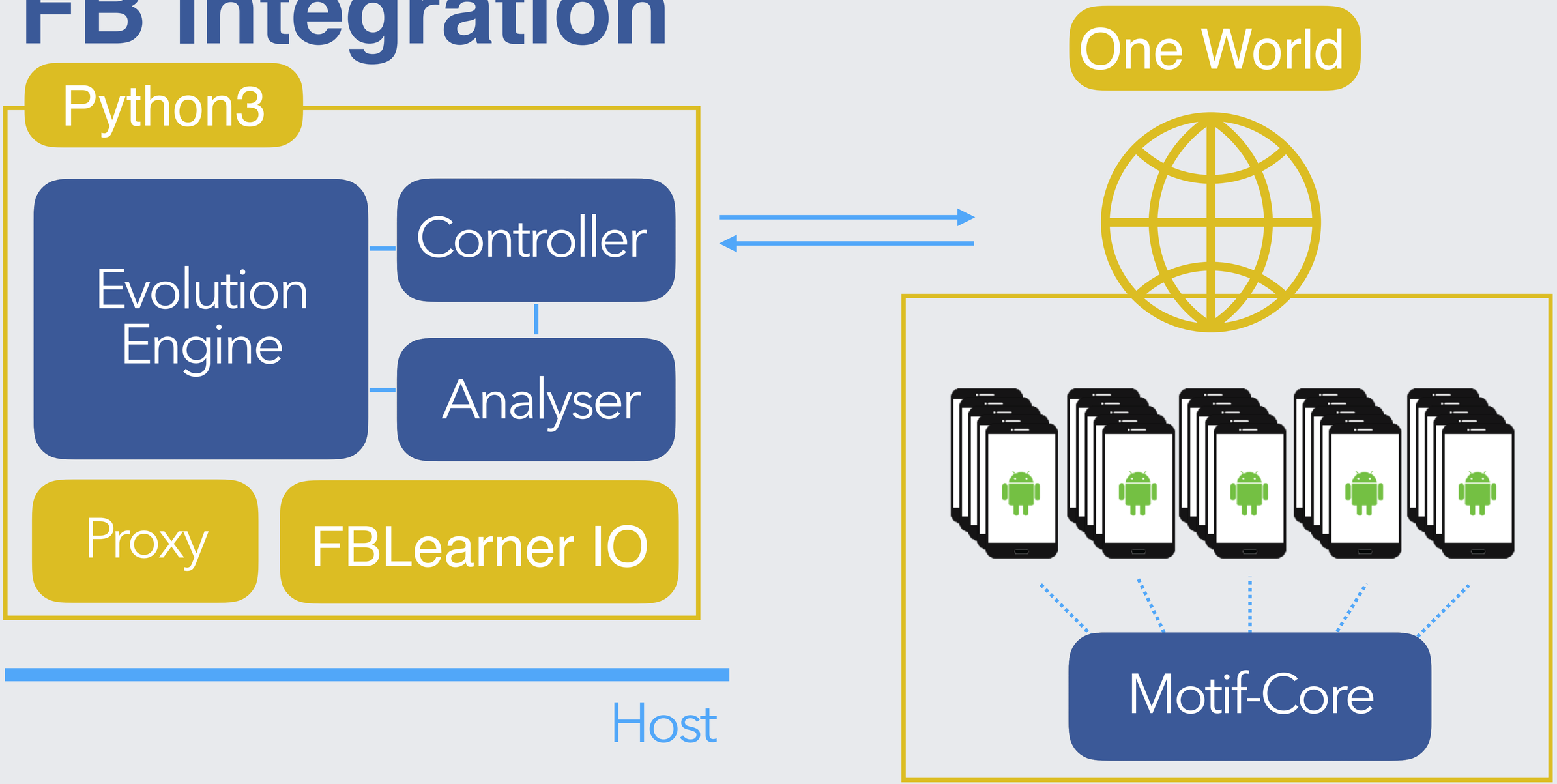
FB Integration



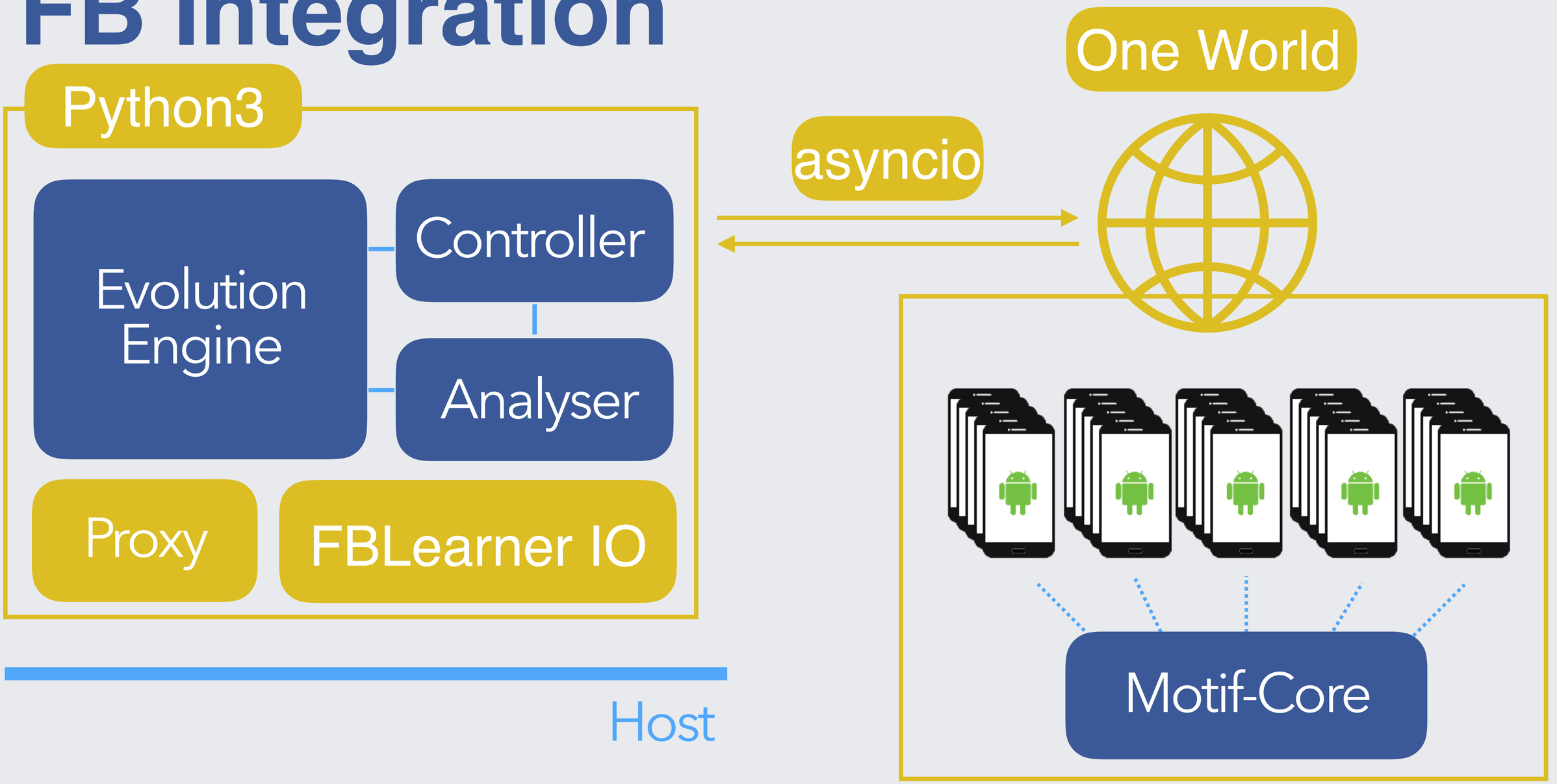
FB Integration

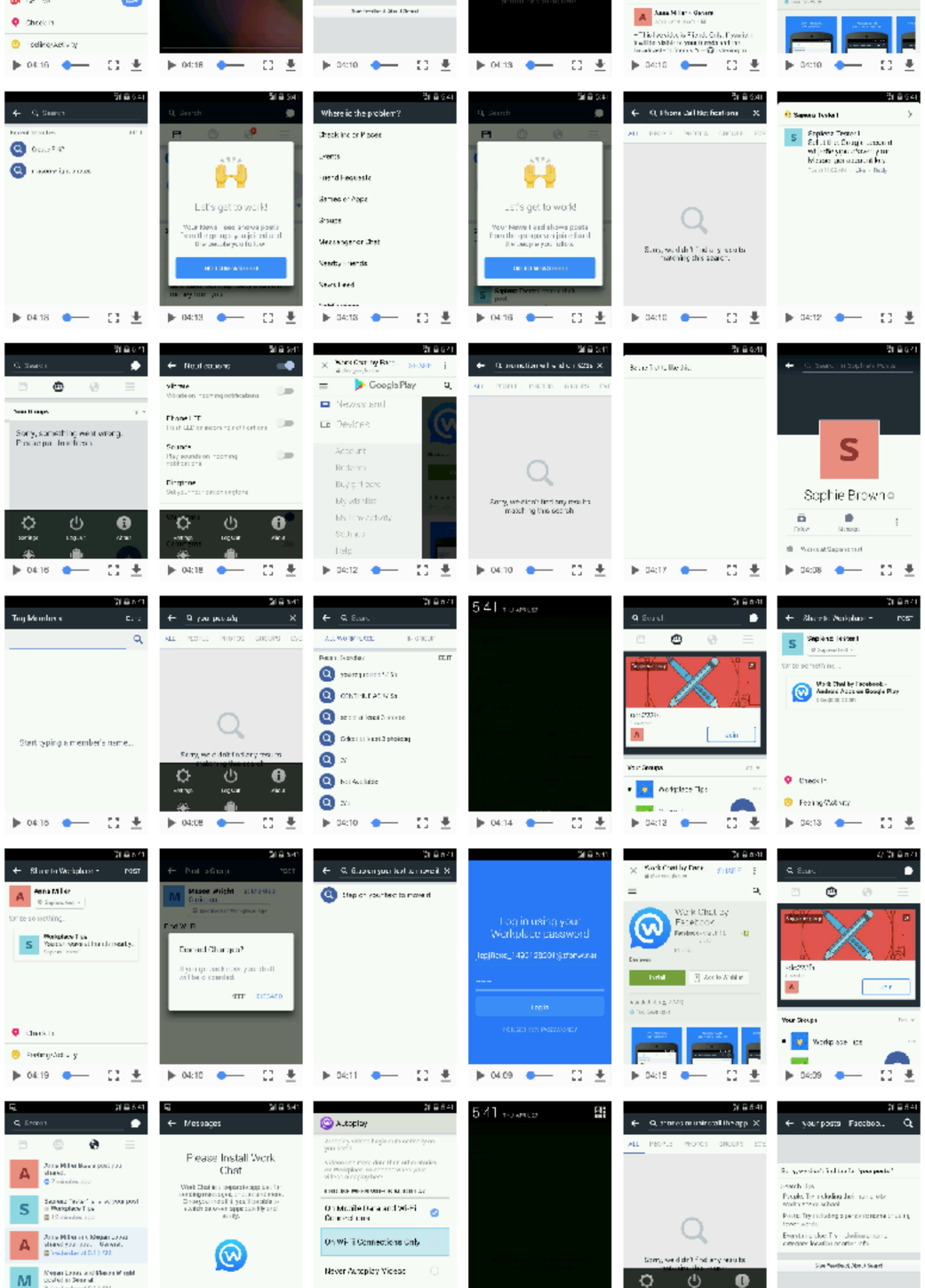


FB Integration



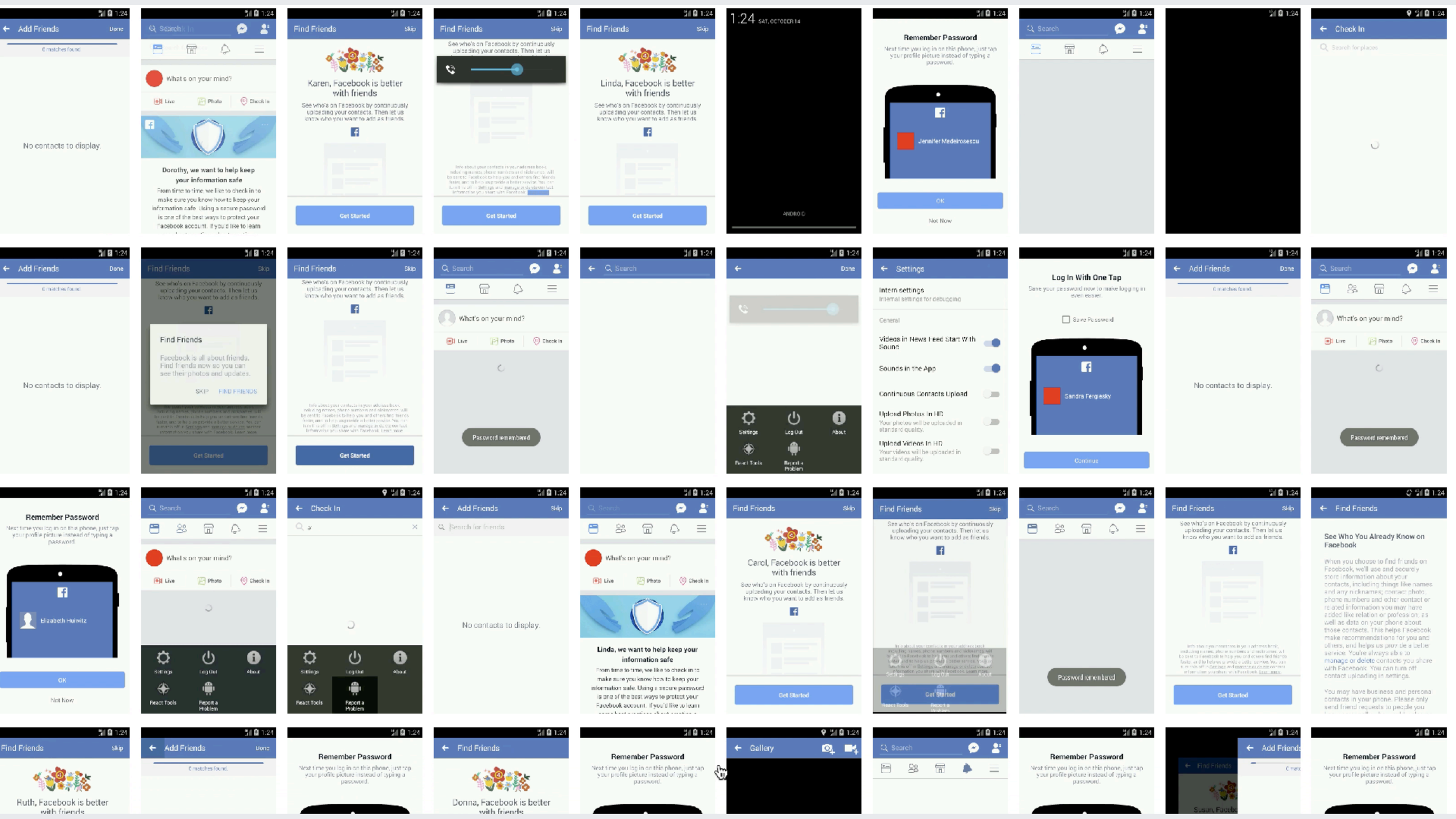
FB Integration





OneWorld allows Sapienz to run on arbitrarily many


0:06


Add Friends
Done
No contacts to display.
Search
What's on your mind?
Live
Photo
Check In
Dorothy, we want to help keep your information safe
From time to time, we like to check in to make sure you know how to keep your information safe. Using a secure password is one of the best ways to protect your Facebook account. If you'd like to learn
Find Friends
Skip
Karen, Facebook is better with friends
See who's on Facebook by continuously uploading your contacts. Then let us know who you want to add as friends.
Get Started
Linda, Facebook is better with friends
1:24 SAT, OCTOBER 14
ANDROID
Remember Password
Next time you log in on this phone, just tap your profile picture instead of typing a password.
Jennifer Medeiroseessu
OK
Not Now
Check In
Search for places
Find Friends
Facebook is all about friends. Find friends now so you can see their photos and updates.
SKIP
FIND FRIENDS
Password remembered
Settings
Log Out
About
React Tools
Report a Problem
Settings
Intern Settings
Internal settings for debugging
General
Videos in News Feed Start With Sound
Sounds in the App
Continuous Contacts Upload
Upload Photos in HD
Your photos will be uploaded in standard quality.
Upload Videos in HD
Your videos will be uploaded in standard quality.
Log In With One Tap
Save your password now to make logging in even easier.
Save Password
Sandra Fergasky
Continue
Remember Password
Next time you log in on this phone, just tap your profile picture instead of typing a password.
Elizabeth Huiwitz
OK
Not Now
Linda, we want to help keep your information safe
From time to time, we like to check in to make sure you know how to keep your information safe. Using a secure password is one of the best ways to protect your Facebook account. If you'd like to learn
Carol, Facebook is better with friends
See who's on Facebook by continuously uploading your contacts. Then let us know who you want to add as friends.
Get Started
Password remembered
See Who You Already Know on Facebook
When you choose to find friends on Facebook, we'll use and securely store information about your contacts, including things like names and any nicknames; contact photo; phone numbers and other contact or related information you may have added like relation or profession, as well as data on your phone about those contacts. This helps Facebook make recommendations for you and others, and helps us provide a better service. You're always able to manage or delete contacts you share with Facebook. You can turn off contact uploading in settings.
You may have business and persona contacts in your phone. Please only send friend requests to people you
Ruth, Facebook is better with friends
Donna, Facebook is better with friends
Gallery
Search
Remember Password
Next time you log in on this phone, just tap your profile picture instead of typing a password.
Add Friends
Done
0 matches found.

facebook

Automated Search Based Test Case Design at Facebook Scale

Mark Harman
Joint work with the Sapienz Team
Software Engineering Manager, Sapienz Team
November, 2017