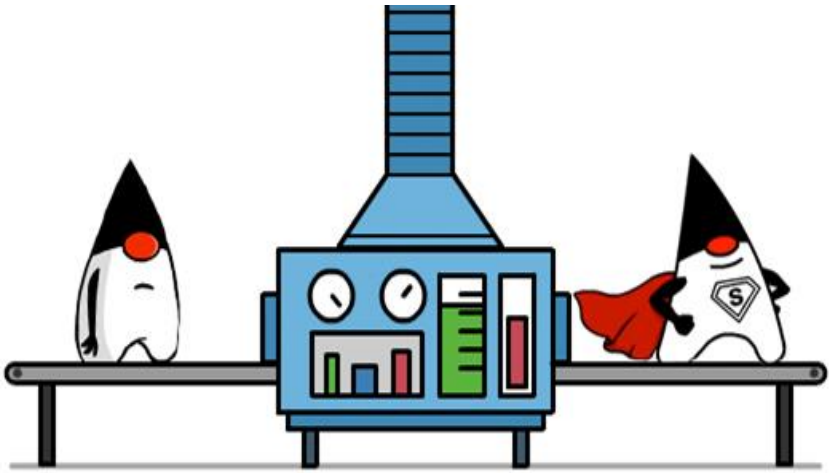
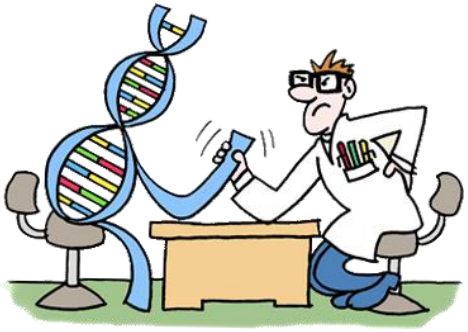
Darwinian Data Structure Selection

ARTEMIS



Michail Basios
Lingbo Li
Fan Wu
Leslie Kanthan
Earl T. Barr



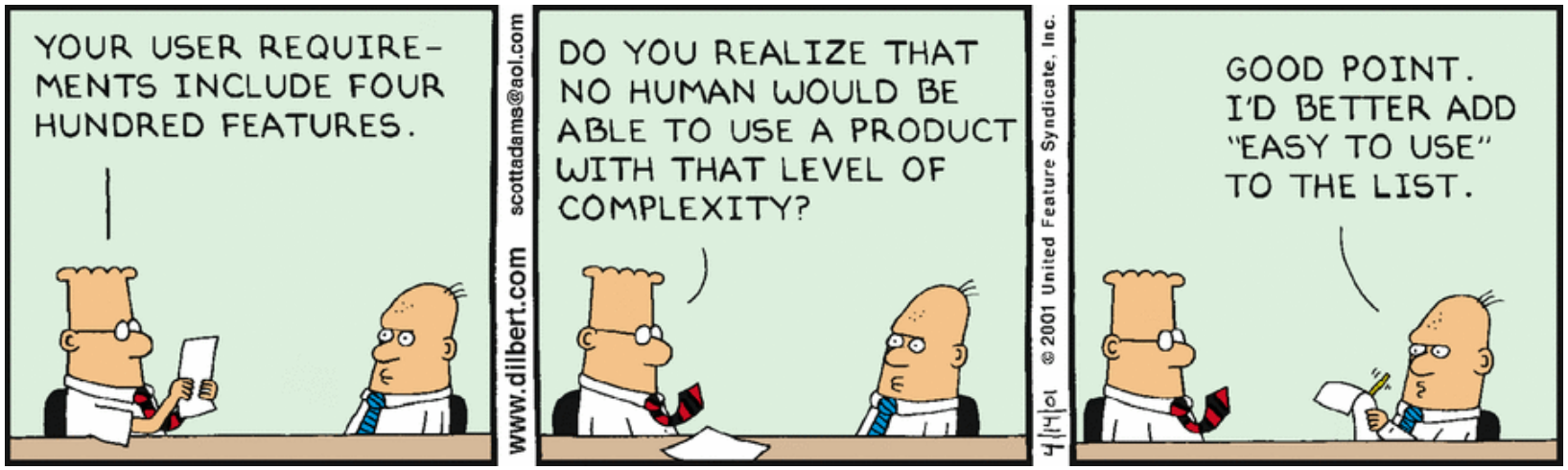
CREST

UCL

Donald Knuth's Advice

> We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**
>
> — Donald Knuth

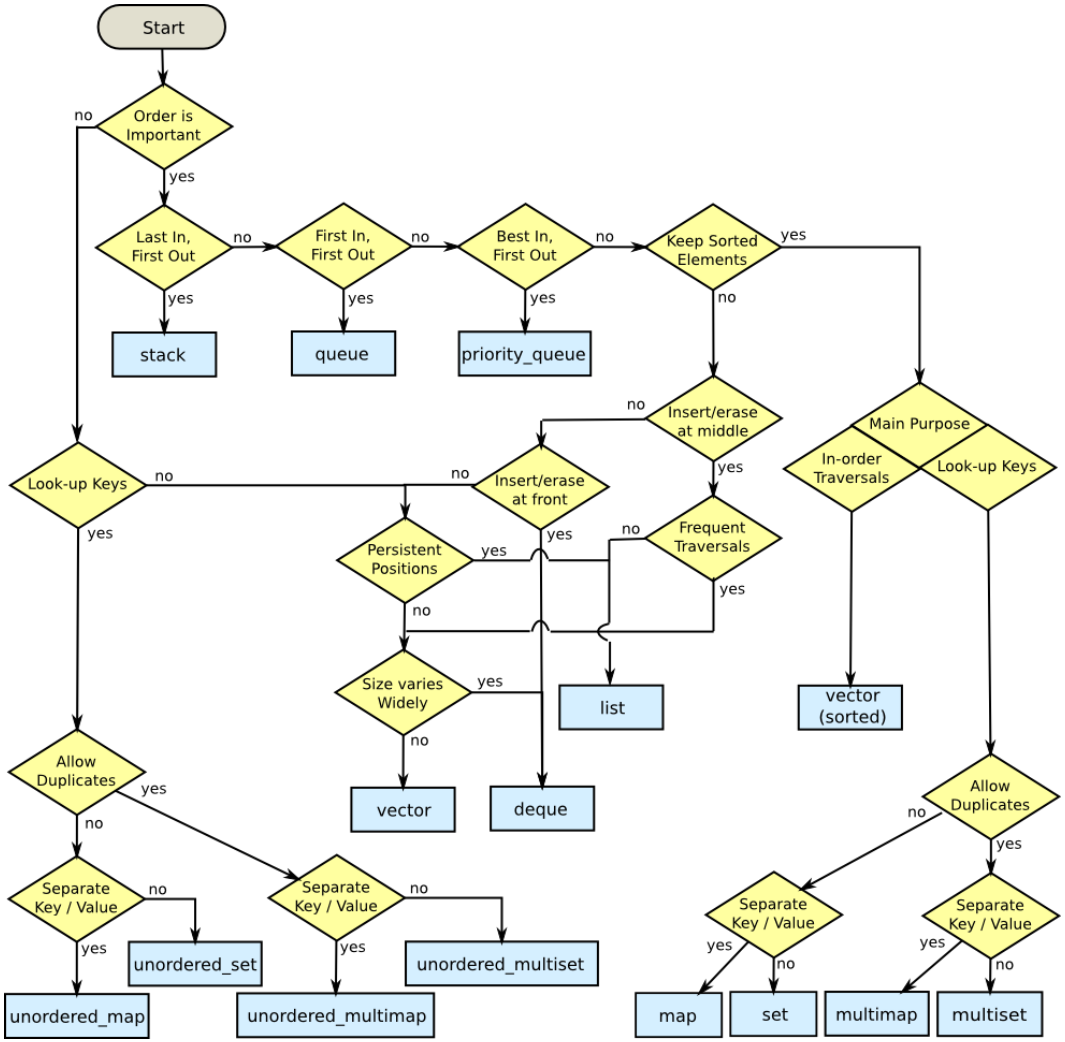
Focus Is on User Requirements



It’s Not Just Pre-optimization

Optimisation Can Be Beneficial

Data Structure Selection/Optimisation Process



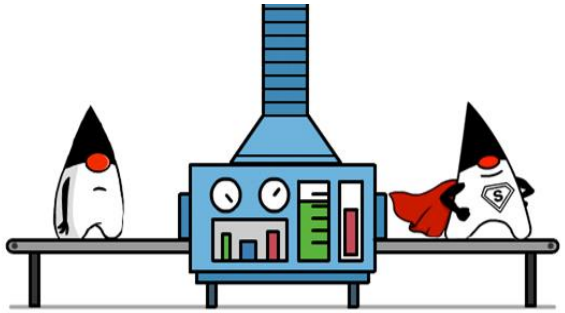
Hmmm, maybe just use defaults that work in most cases.

Darwinian Data Structure Selection

Corpus: 43 projects

4.8% median execution time improvement
10.2% median less memory consumption
5.1% median less CPU usage

ARTEMIS

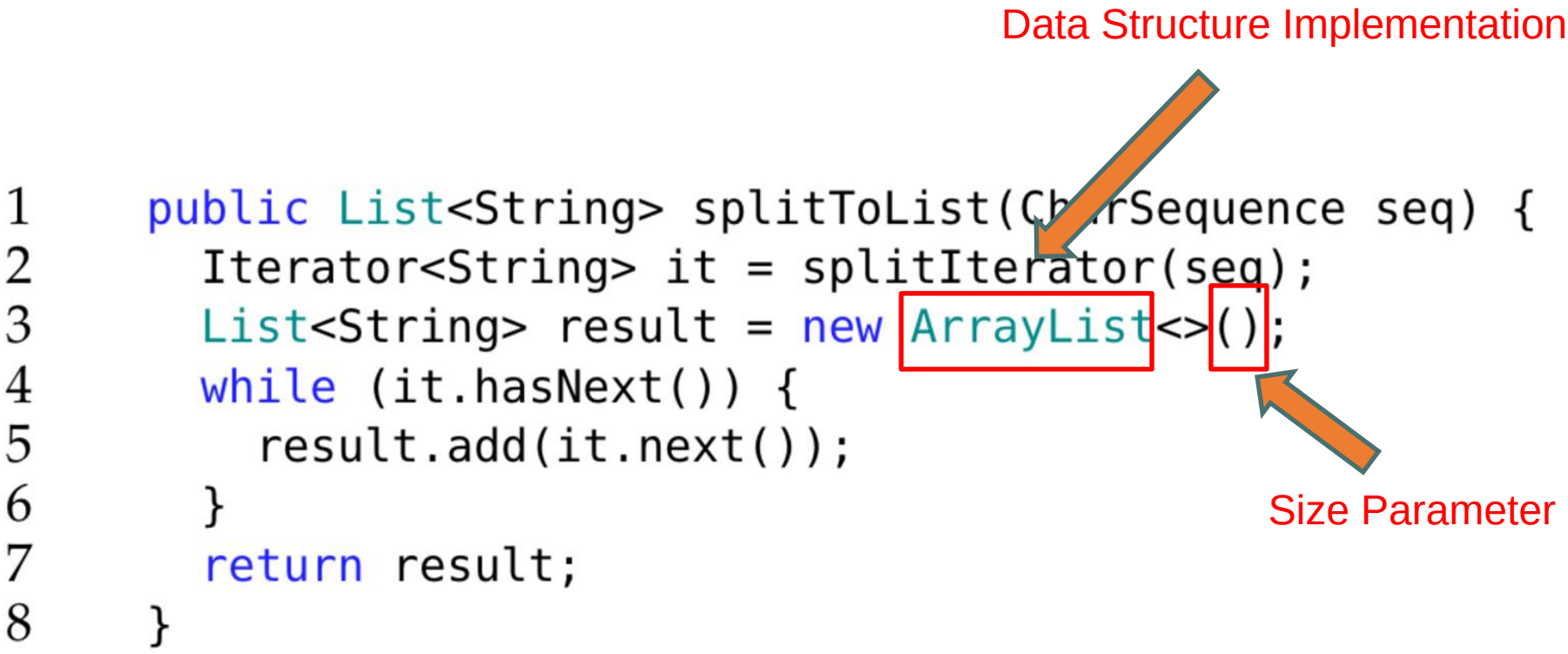


- 13% execution time improvement
- 9% less memory consumption
- 4% less CPU usage

- 23.5% less memory consumption (without affecting other measures)

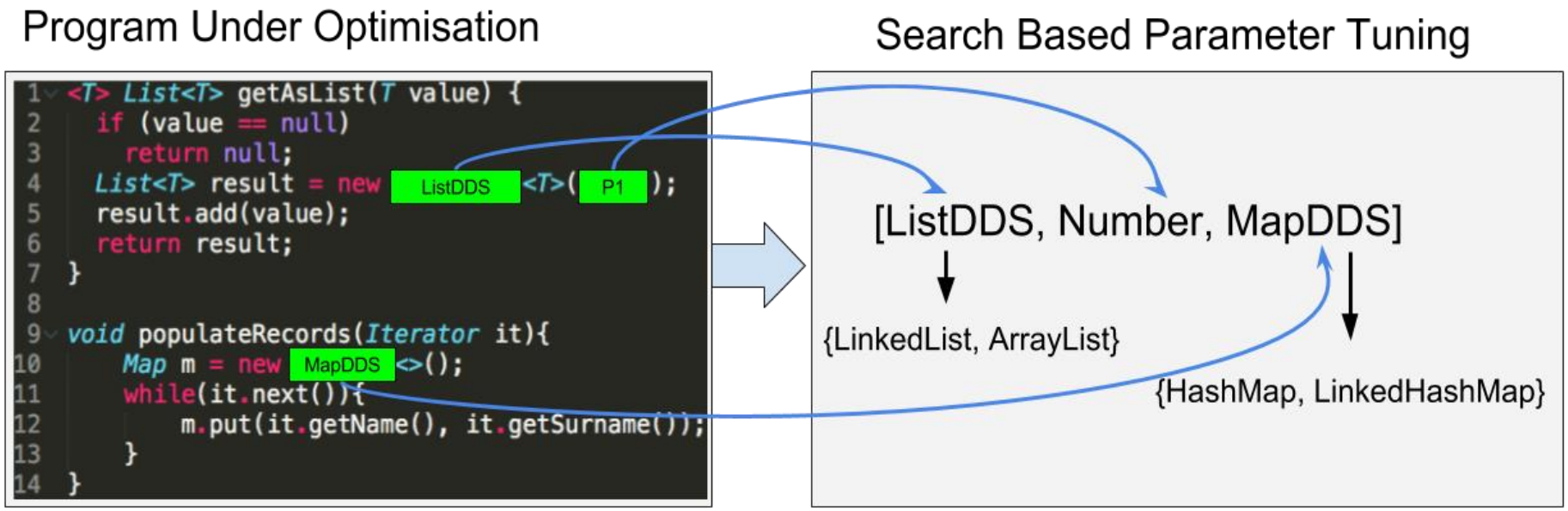
- 16.5% less memory consumption (without affecting other measures)

Motivating Example

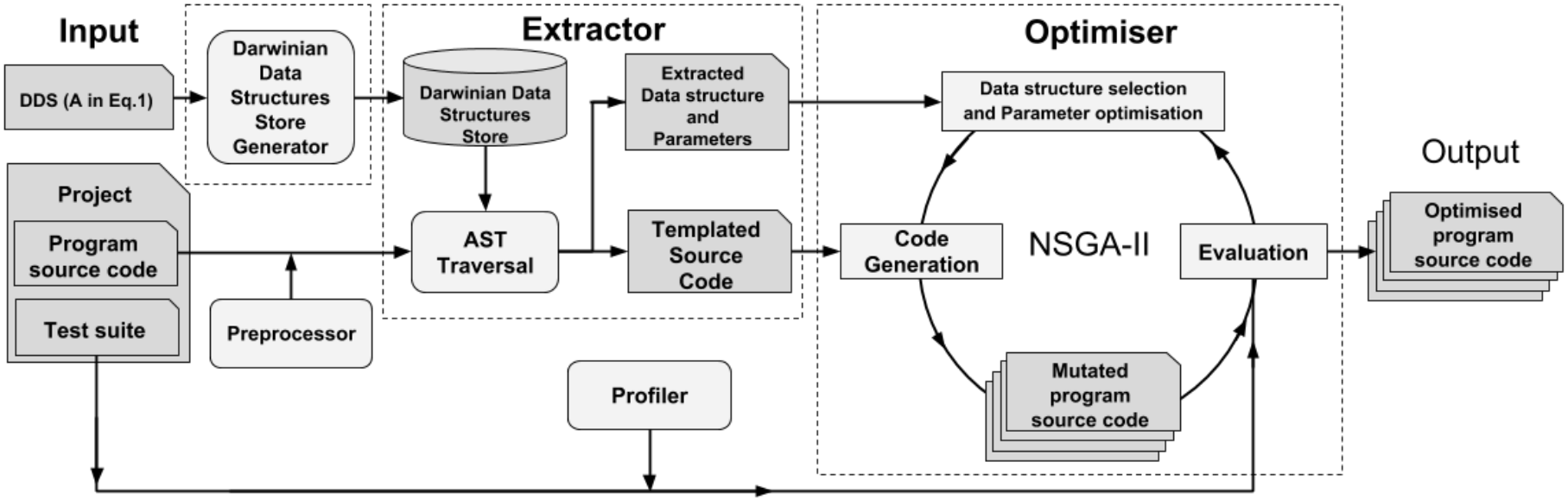


A function from the Splitter class of Guava Library

Artemis searches a program's space of **Darwinian Data Structure** holes for optimal implementations and configurations.

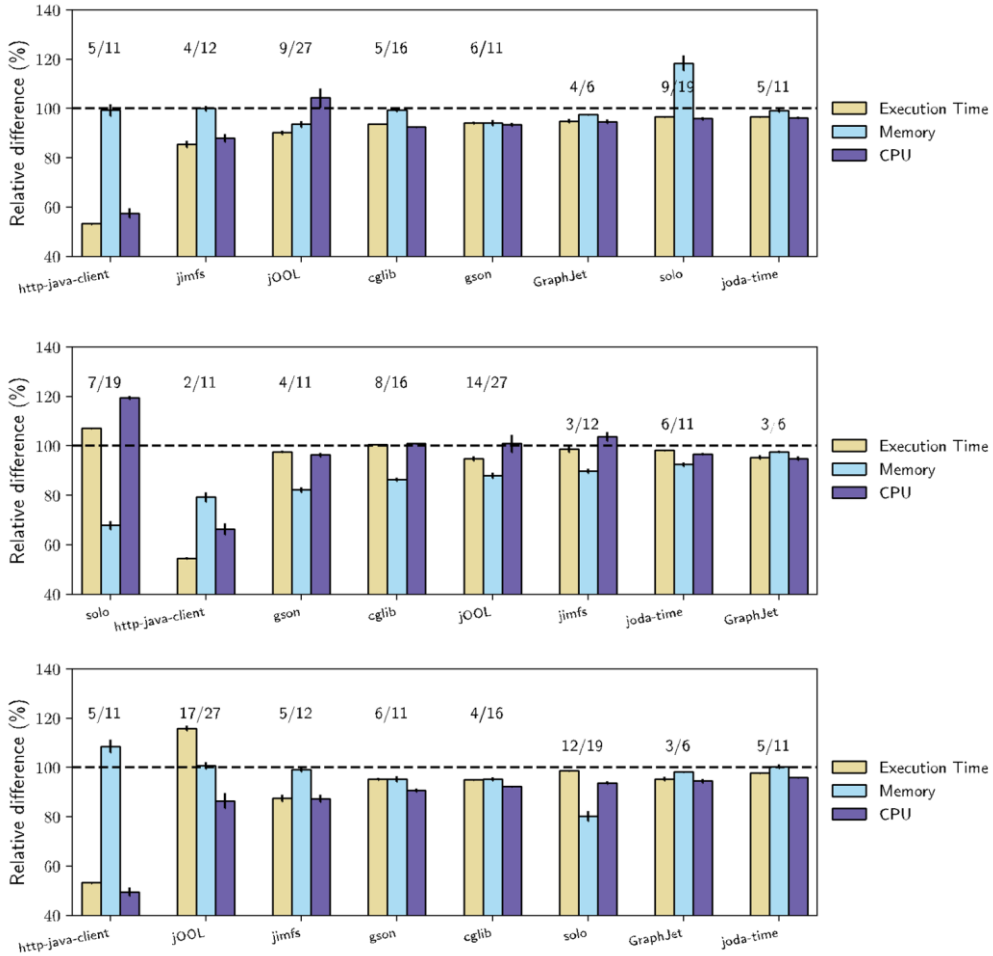


Architecture of Artemis



Source to Source Transformation

What is the improvement that ARTEMIS provides for each measure?



Popular GitHub projects

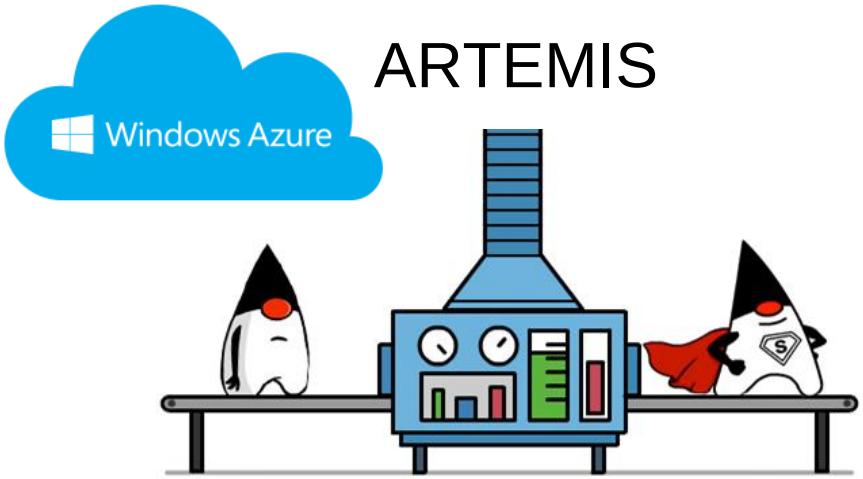
Best improvements per measure

Execution Time: 4.8 %
Memory Consumption: 10.2%
CPU usage: 5.1%

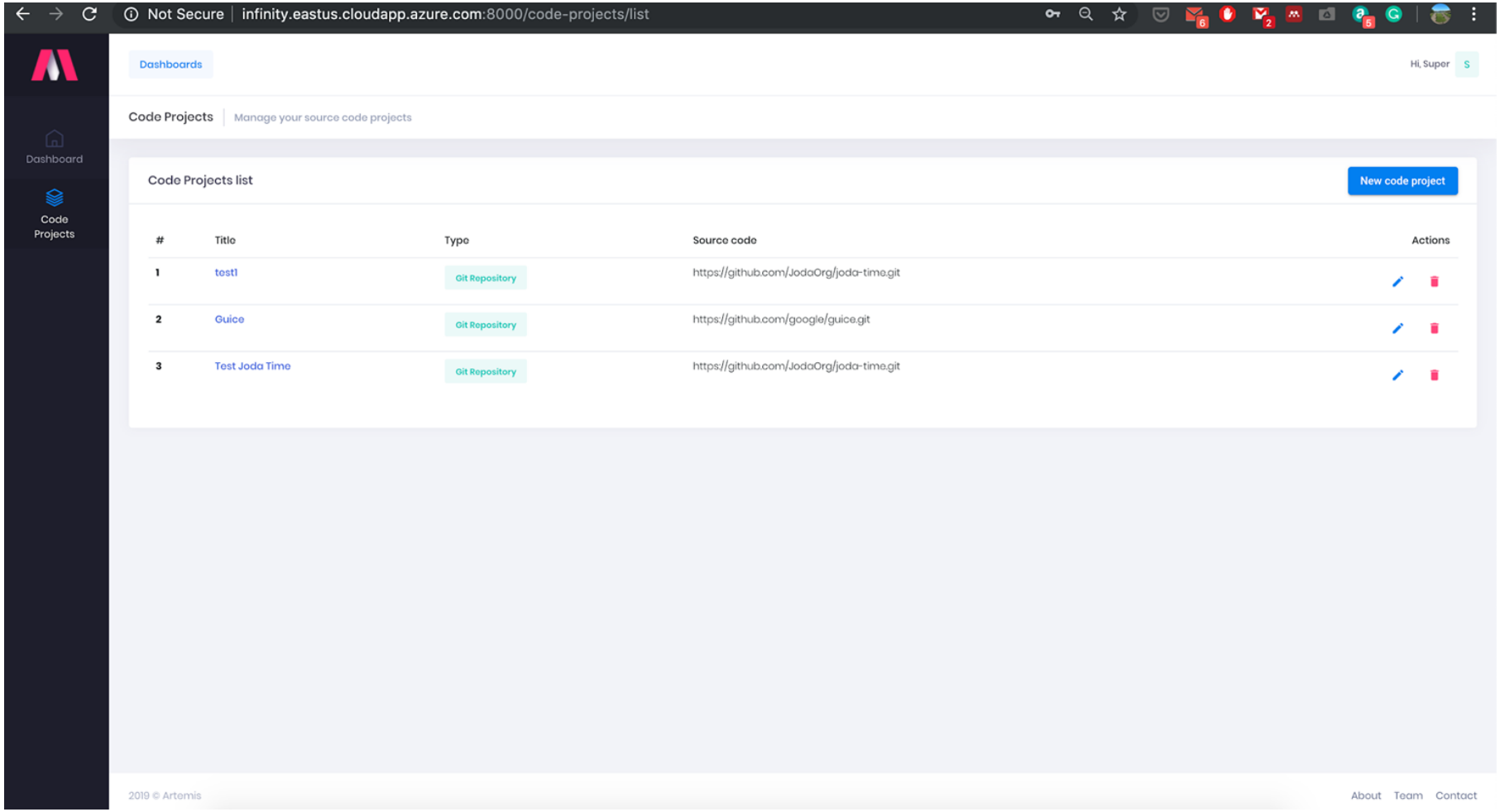
Cost of Artemis

Average optimisation process -> 3.05 hours
Azure cost -> £ 0.41 per hour

Artemis cost ≪ Software Engineer Cost

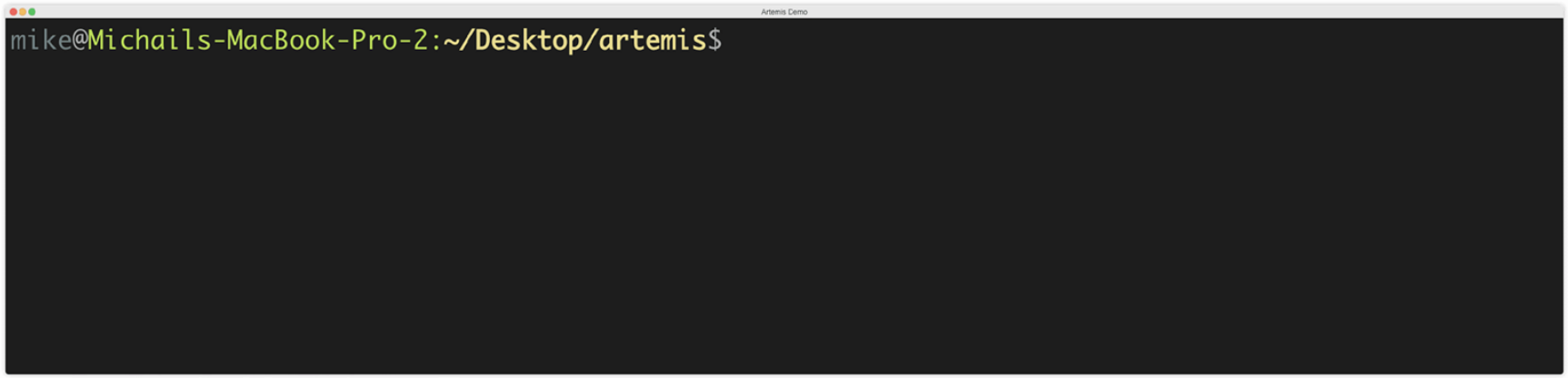


Artemis as a Web Service



Artemis Web-Based Dashboard

Artemis as a Command Line Tool




```
usage: Data Structure Optimiser
 -a,--all-methods <arg>        1 if enabled, 0 if disabled
 -d,--ds <arg>                 File with list of data structures
 -f,--file <arg>               input folder
 -h,--help                     shows this message
 -i,--ids <arg>                File with initial list of data structures
 -m,--mode <arg>               executor mode: artemis, strategy, dacapo,
                               cpp, solution
 -p,--popular-methods <arg>    number of popular methods
 -s,--solution <arg>           solution folder
```

What makes Artemis cool?

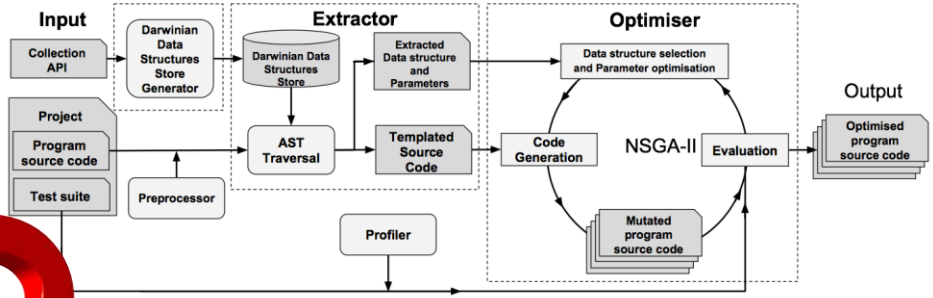
- Easy to use as it **automatically** identifies and tunes the data structures and corresponding parameters of a program
- Easy adaptable (String Buffer vs String Builder)
- Multiple languages (Java, C++, Scala, etc)
- Multi Objective optimisation on programs used by thousands of users
- It can scale on the cloud and provide fast optimisations
- Potential to be a very nice cloud service connected with Git

ARTEMIS	HUMANS
Can automatically and fast scan thousands of lines of code and tune data structures	Will take significant amount of time to identify and change potential under-optimised data structures
Very cheap and easy to use	Can be very costly and time consuming
Found optimisation opportunities and improved highly optimised code by humans (open-source programs such as Guava)	Had neglected optimisation opportunities
Applicable on other domains (i.e., algorithmic trading)	Need domain knowledge

Darwinian Data Structure Selection

> We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**
>
> — Donald Knuth

Architecture of Artemis

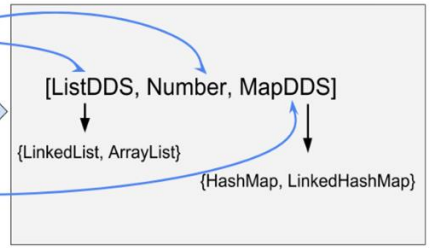


ARTEMIS searches a program's space of **DDS** holes for optimal implementations and configurations.

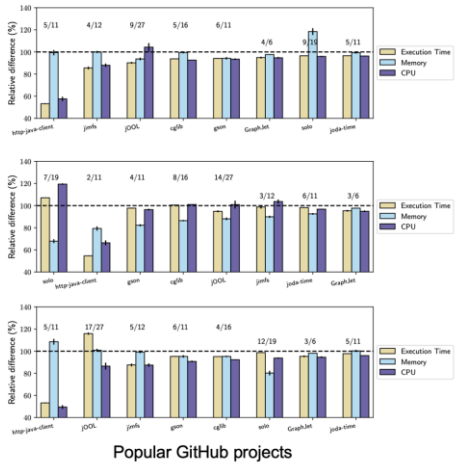
Program Under Optimisation

```
1  <T> List<T> getAsList(T value) {
2    if (value == null)
3      return null;
4    List<T> result = new ListDDS <T>( P1 );
5    result.add(value);
6    return result;
7  }
8
9  void populateRecords(Iterator it){
10   Map m = new MapDDS <>();
11   while(it.next()){
12       m.put(it.getName(), it.getSurname());
13   }
14 }
```

Search Based Parameter Tuning



What is the improvement that ARTEMIS provides for each measure?



Best improvements per measure

Execution Time: 4.8 %
Memory Consumption: 10.2%
CPU usage: 5.1%