Genetic Improvement

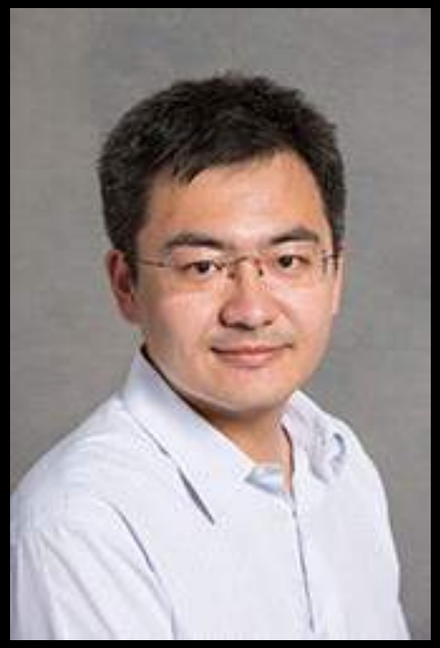
Justyna Petke

Centre for Research in Evolution, Search and Testing
University College London

CREST

Thank you

Mark Harman



Yue Jia

Alexandru Marginean

What does the word "Computer" mean?

Oxford Dictionary

"a person who makes calculations,
especially with a calculating machine."

Wikipedia

"The term "computer", in use from the mid 17th century, meant "one who computes": a person performing mathematical calculations."

What does the word "Computer" mean?

Oxford Dictionary

"a person who makes calculations, especially with a calculating machine."

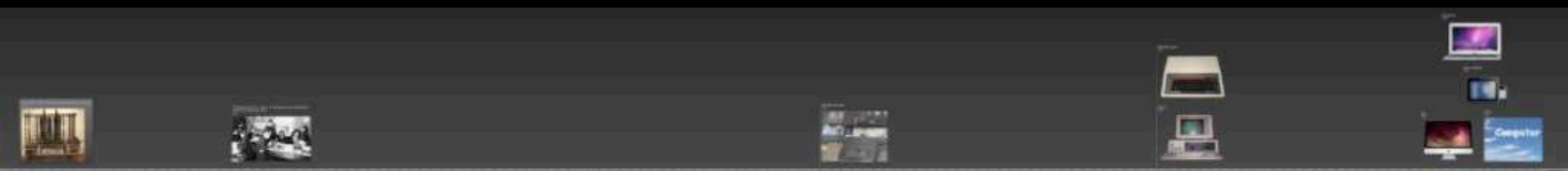
Wikipedia

"The term "computer", in use from the mid 17th century, meant "one who computes": a person performing mathematical calculations."

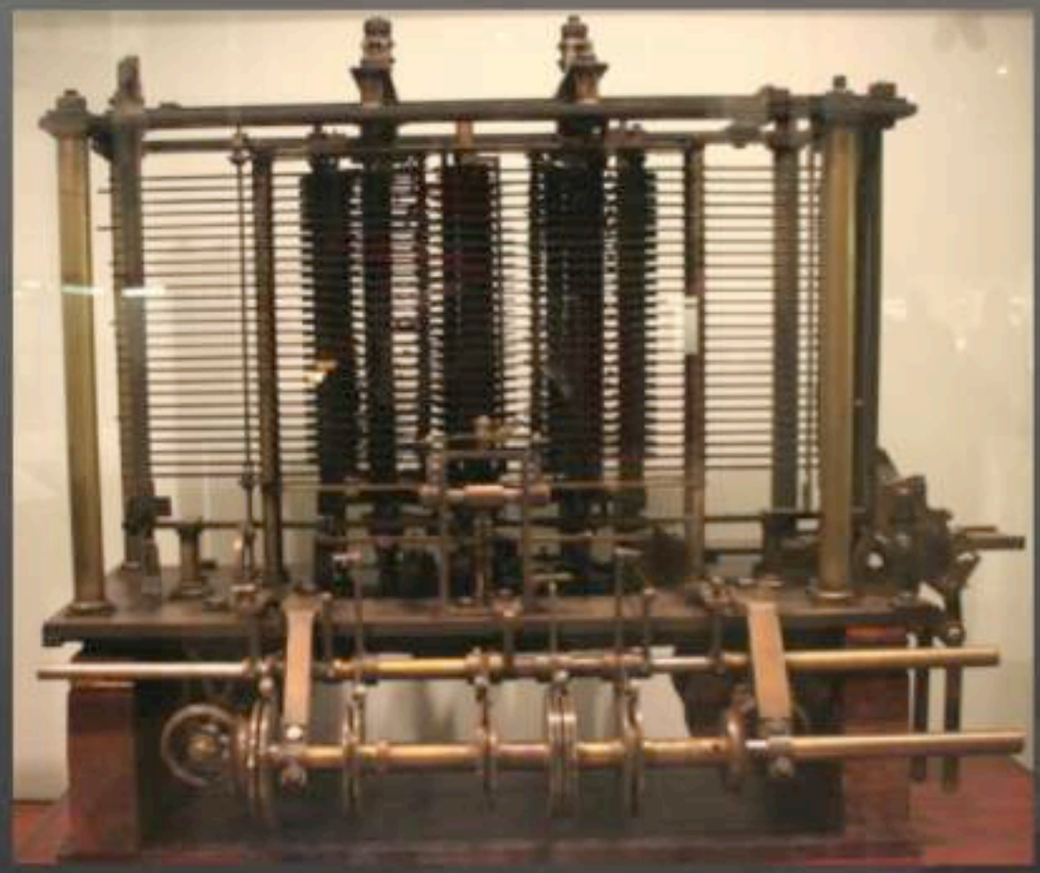
in the beginning ...

The First Computer?

Different People have different opinions



Trial model of a part of the Analytical Engine, built by Babbage.
1873



1872 1873 1874 1875 1876 1877 1878 1879

"Pickering's Harem," so-called, for the group of women computers at the Harvard College Observatory
1893

1890 1891 1892 1893 1894 1895 1896 1897 1898 1899

IBM 026 Card Punch
1949

945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955

BBC Micro Series
1981

IBM PC
1981


IBM PC
1981
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987

Iphone and Ipad
2010

iMac
2006

2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

Macbook Air
2008

2010

Cloud

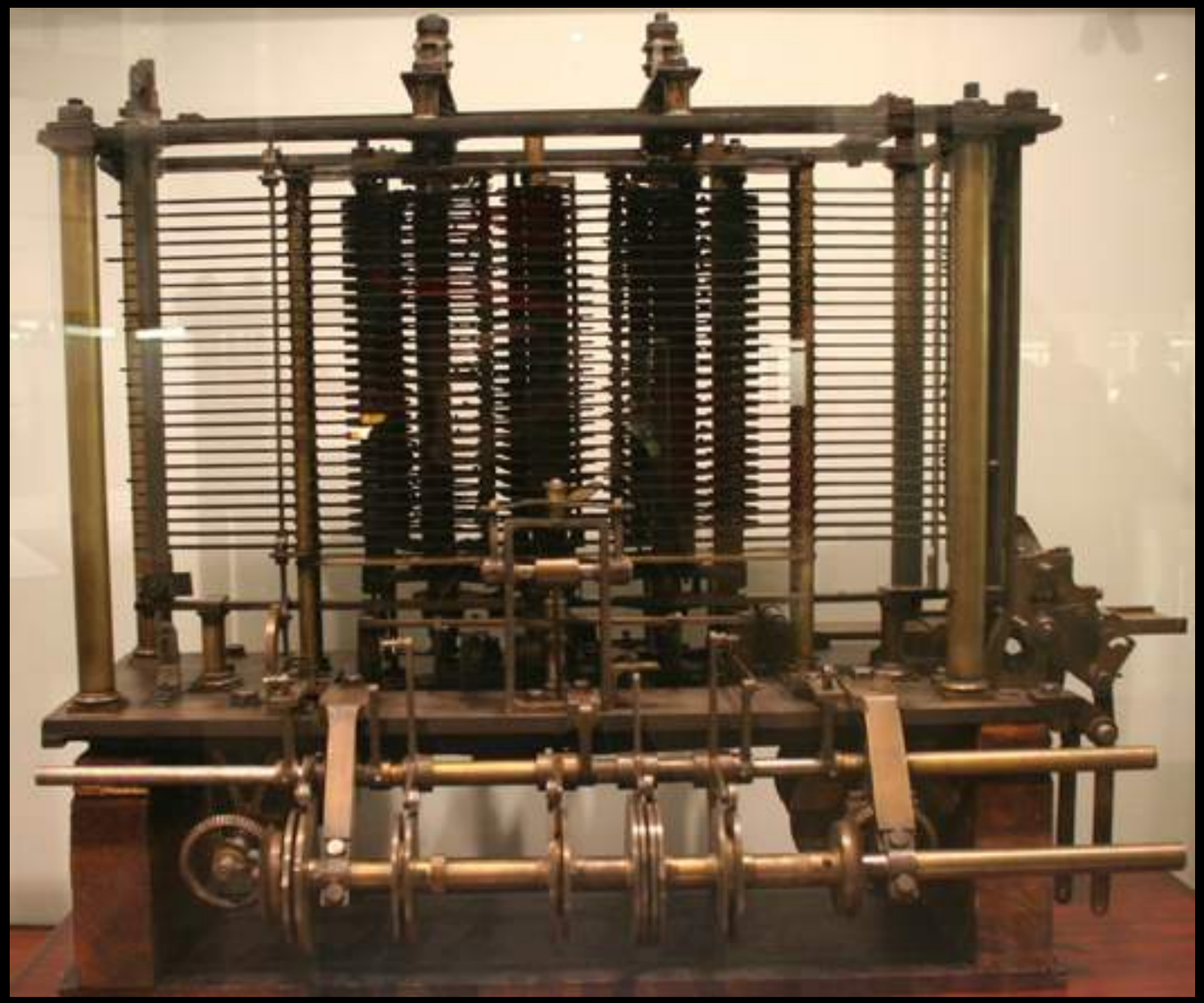
Iphone and Ipad
2010

Cloud
2012

iMac
2006

Who are the programers

Who are the programers

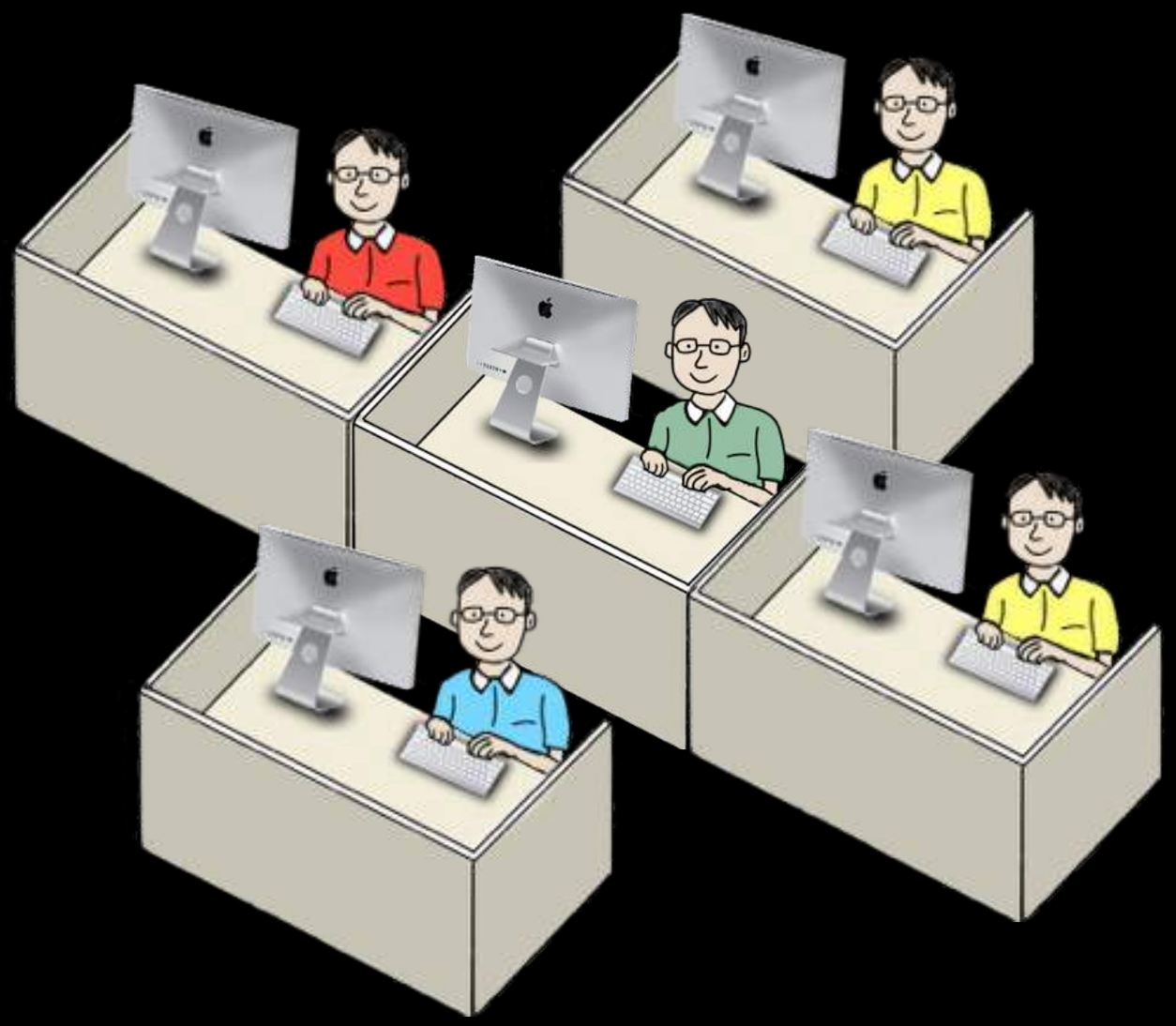


Who are the programers

Who are the programers

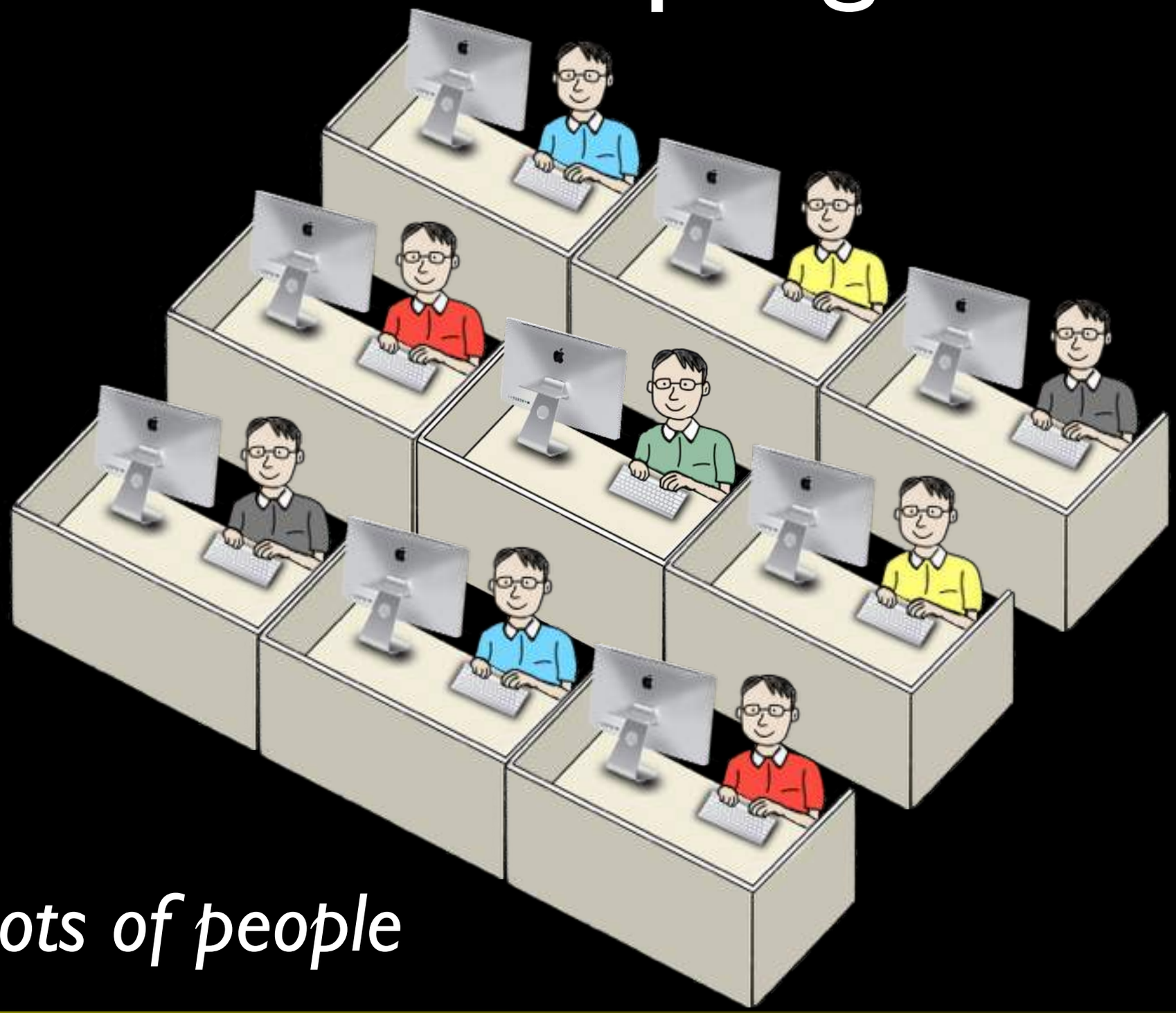
it's always been people

Who are the programers



it's always been people

Who are the programers



lots of people

why people?

human computers seem quaint today

will human programmers seem quaint tomorrow ?

programming is changing

Functional Requirements

Non-Functional Requirements

Requirements

Functional Requirements

functionality of the Program

Non-Functional Requirements

Execution Time

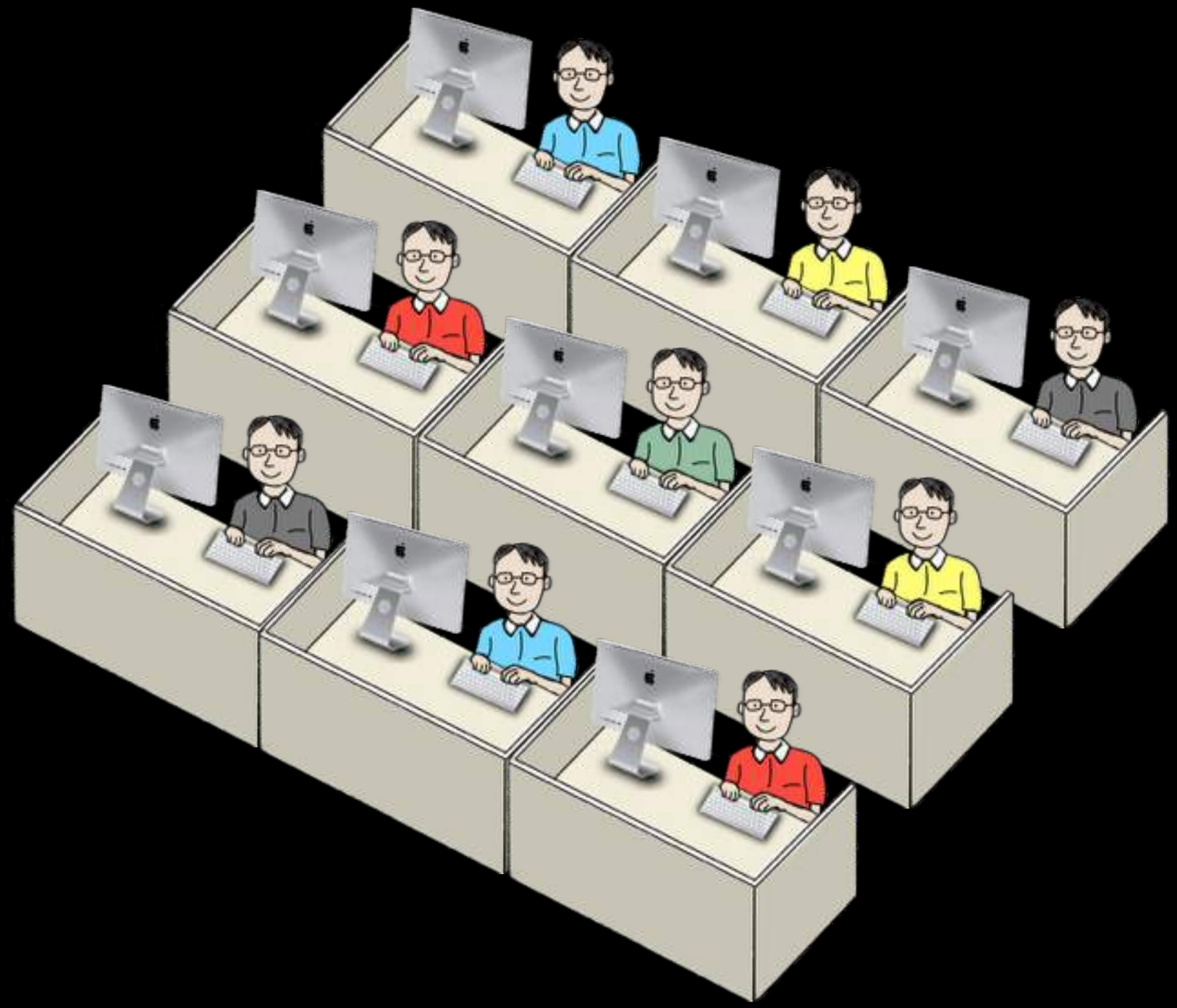
Memory

Bandwidth

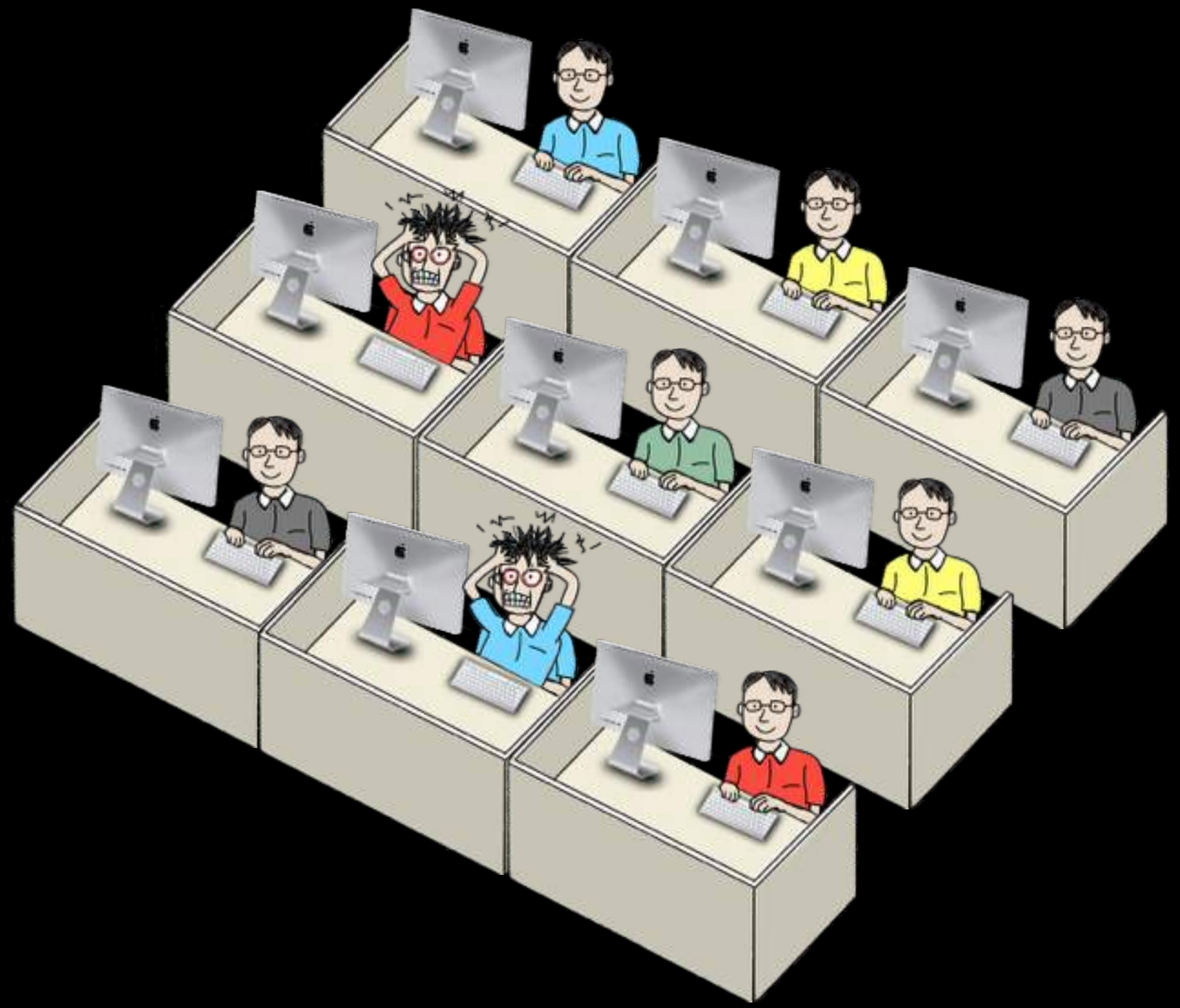
Battery

Size

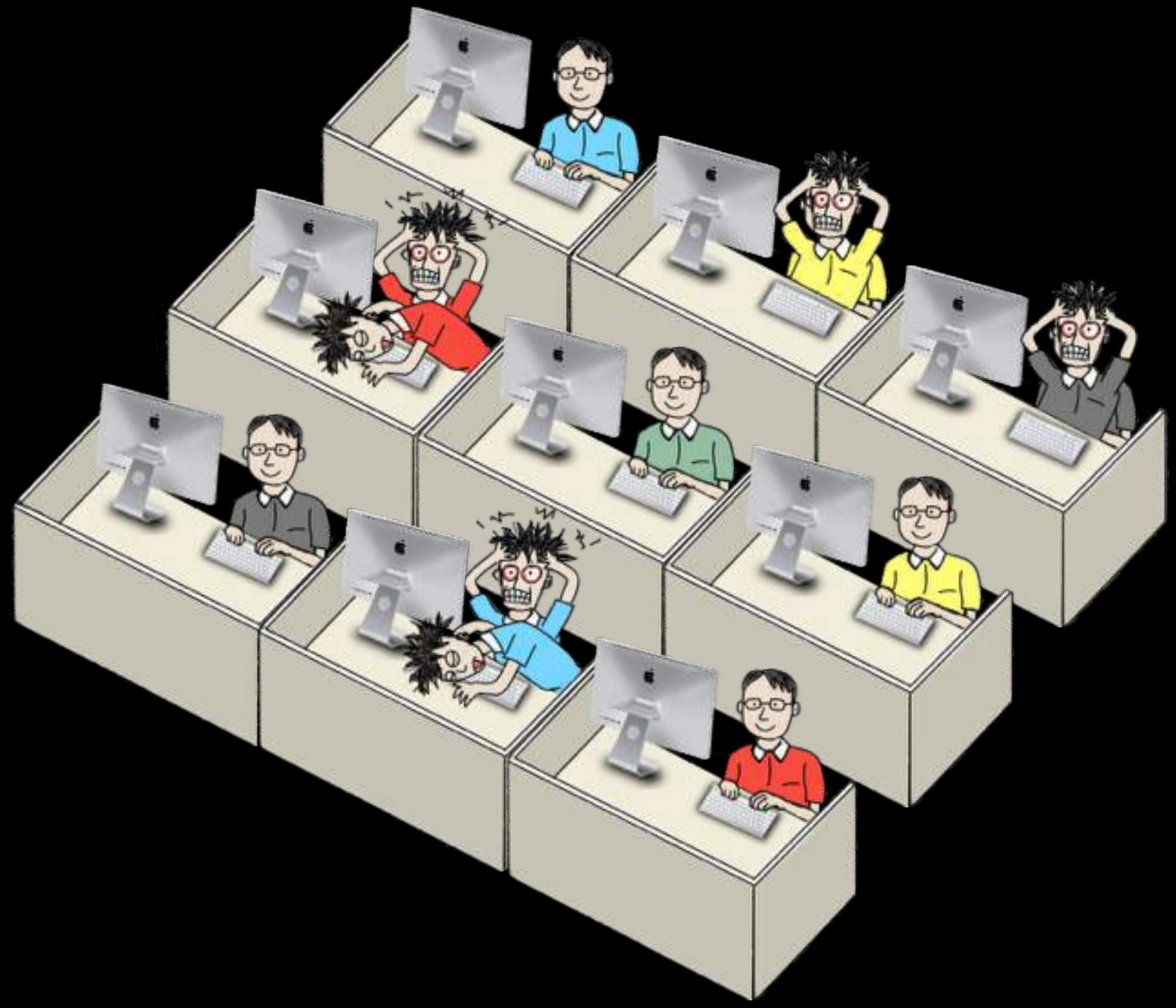
Software Design Process



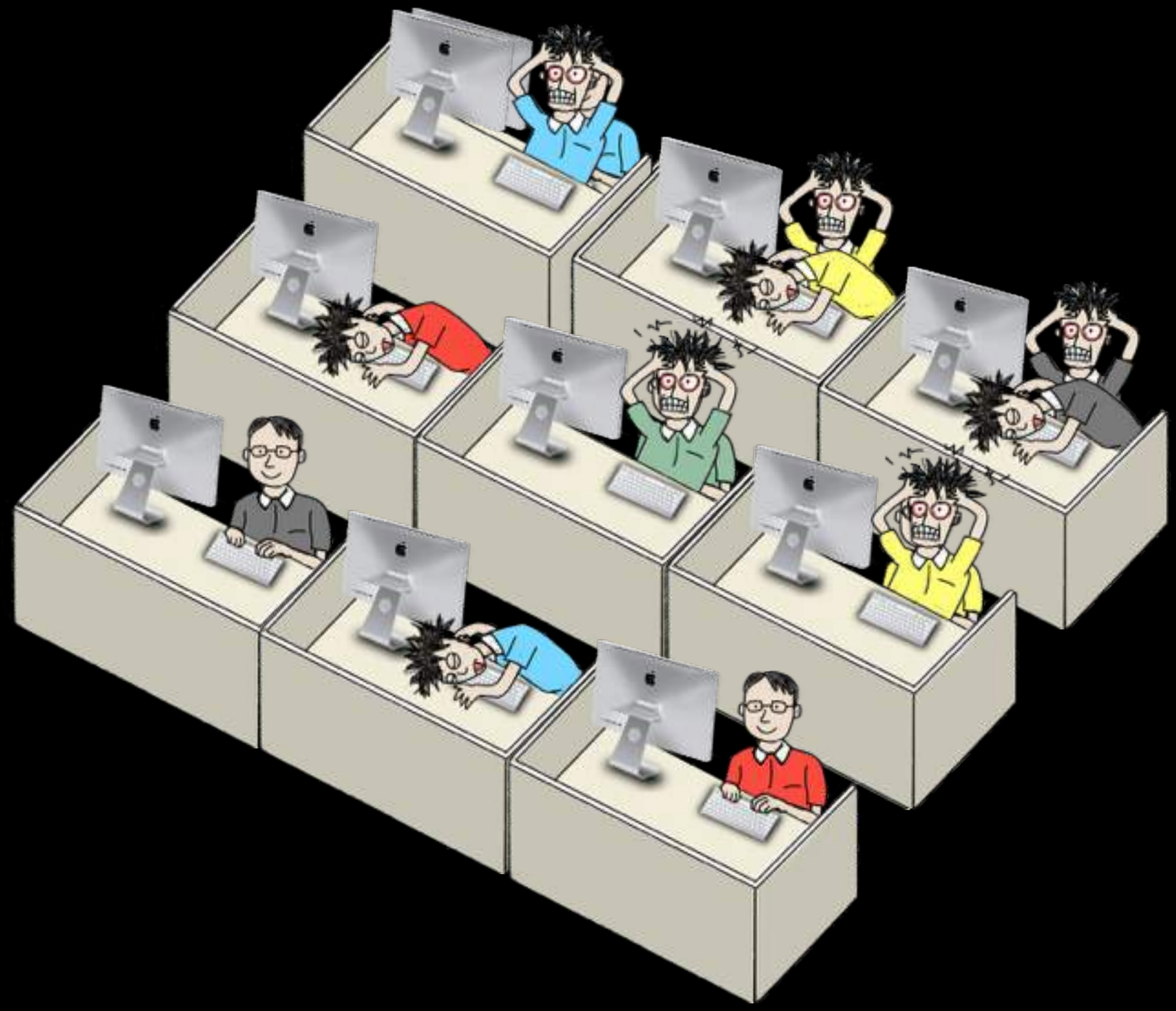
Software Design Process



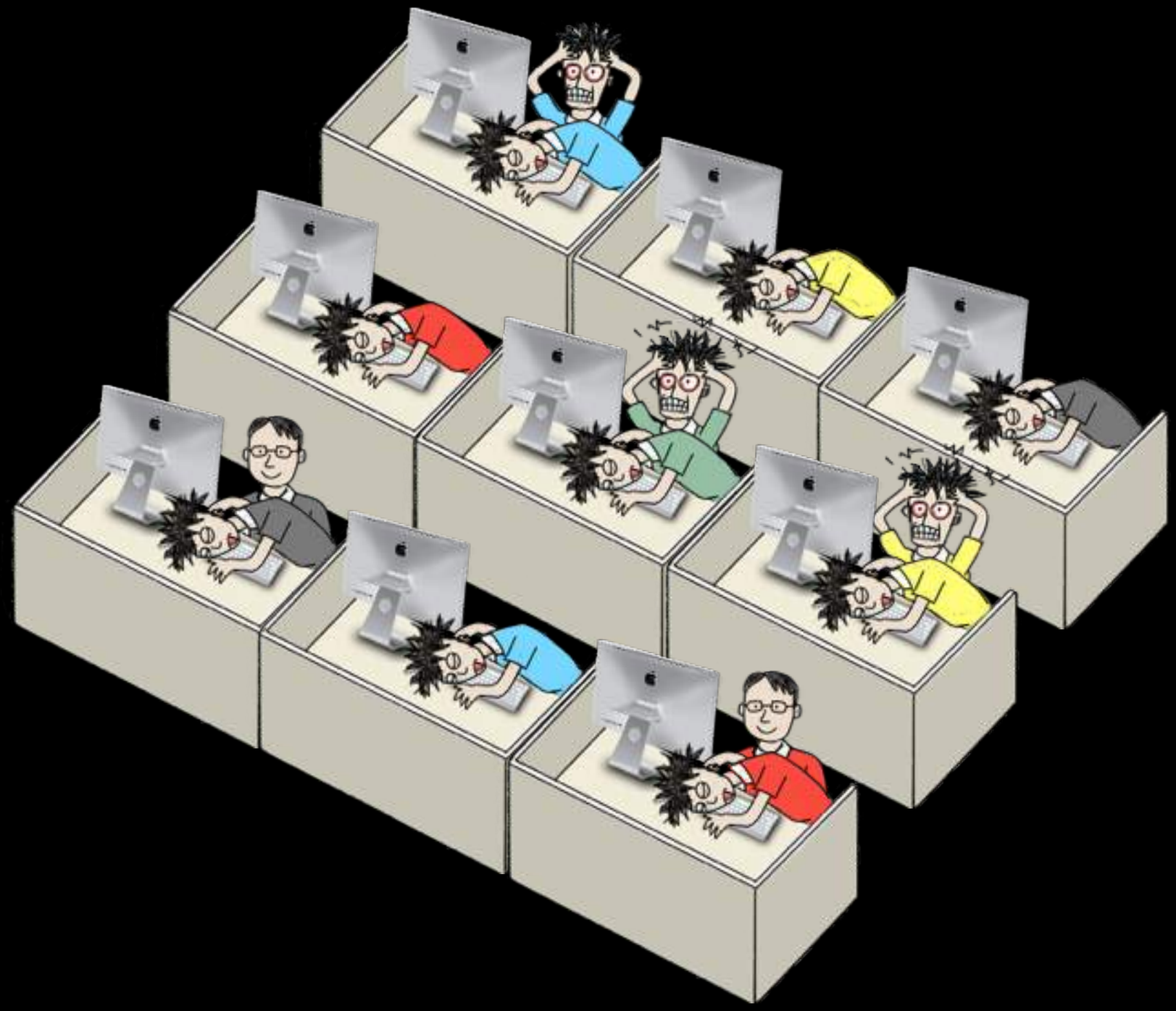
Software Design Process



Software Design Process



Software Design Process



Multiplicity

Which requirements must be human coded ?

Functional Requirements

humans have to define these

Non-Functional Requirements

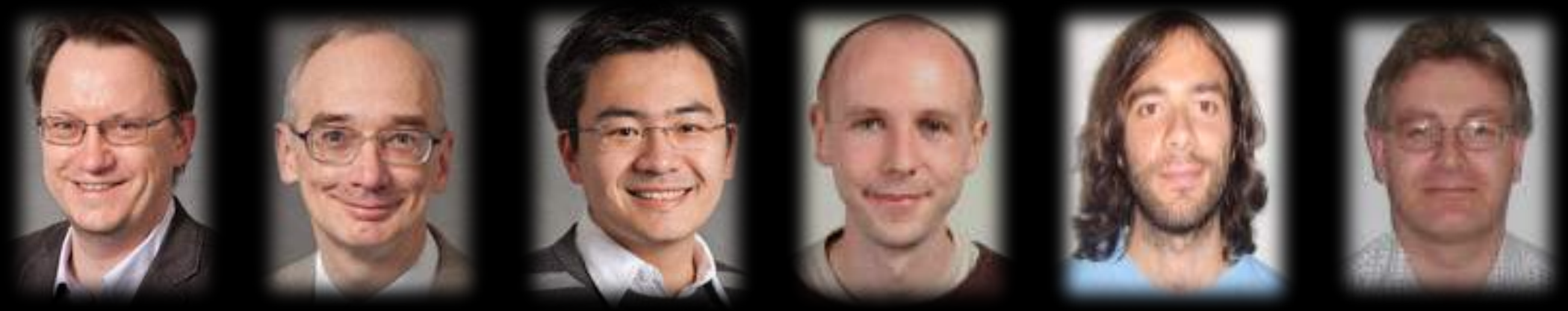
Which requirements are essential to human ?

Functional Requirements

humans have to define these

Non-Functional Requirements

we can optimise these



The GISMOE challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs·

Mark Harman[1], William B. Langdon[1], Yue Jia[1], David R. White[2], Andrea Arcuri[3], John A. Clark[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
[3]Simula Research Laboratory, P. O. Box 134, 1325 Lysaker, Norway.
[4]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, UK.

ABSTRACT

Optimising programs for non-functional properties such as speed, size, throughput, power consumption and bandwidth can be demanding; pity the poor programmer who is asked to cater for them all at once! We set out an alternate vision for a new kind of software development environment inspired by recent results from Search Based Software Engineering (SBSE). Given an input program that satisfies the functional requirements, the proposed programming environment will automatically generate a set of candidate program implementations, all of which share functionality, but each of which differ in their non-functional trade offs. The software designer navigates this diverse Pareto surface of candidate implementations, gaining insight into the trade offs and selecting solutions for different platforms and environments, thereby stretching beyond the reach of current compile[illegible]

Keywords

SBSE, Search Based Optimization, Compilation, Non-functional Properties, Genetic Programming, Pareto Surface.

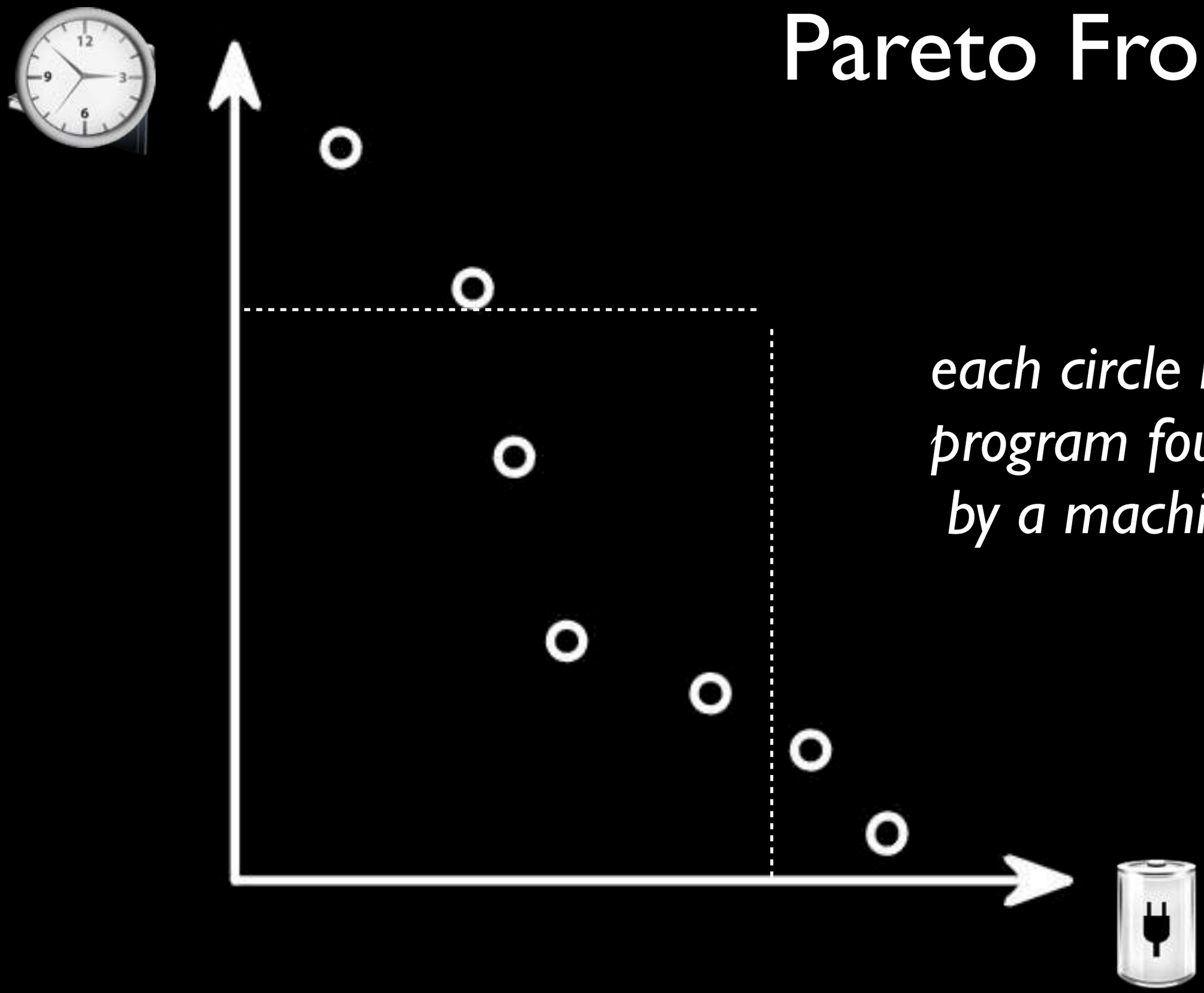
1. INTRODUCTION

Humans find it hard to develop systems that balance many competing and conflicting non-functional objectives. Even meeting a single objective, such as execution time, requires automated support in the form of compiler optimisation. However, though most compilers can optimise compiled code for both speed and size, the programmer may find themselves making arbitrary choices when such objective are in conflict with one another.

Furthermore, speed and size are but two of many objectives that the next generation of software systems will have [illegible]

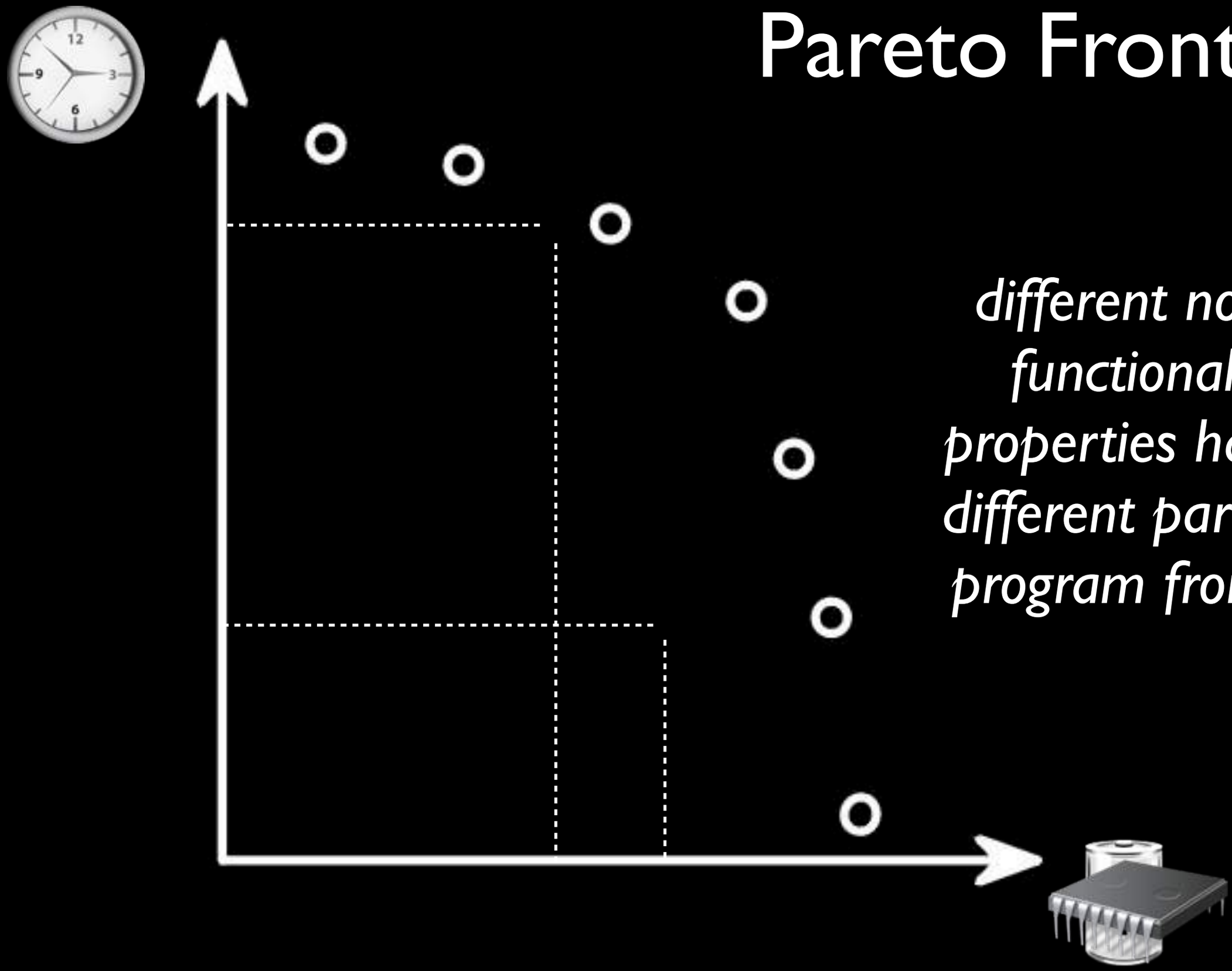
Pareto Front

Pareto Front



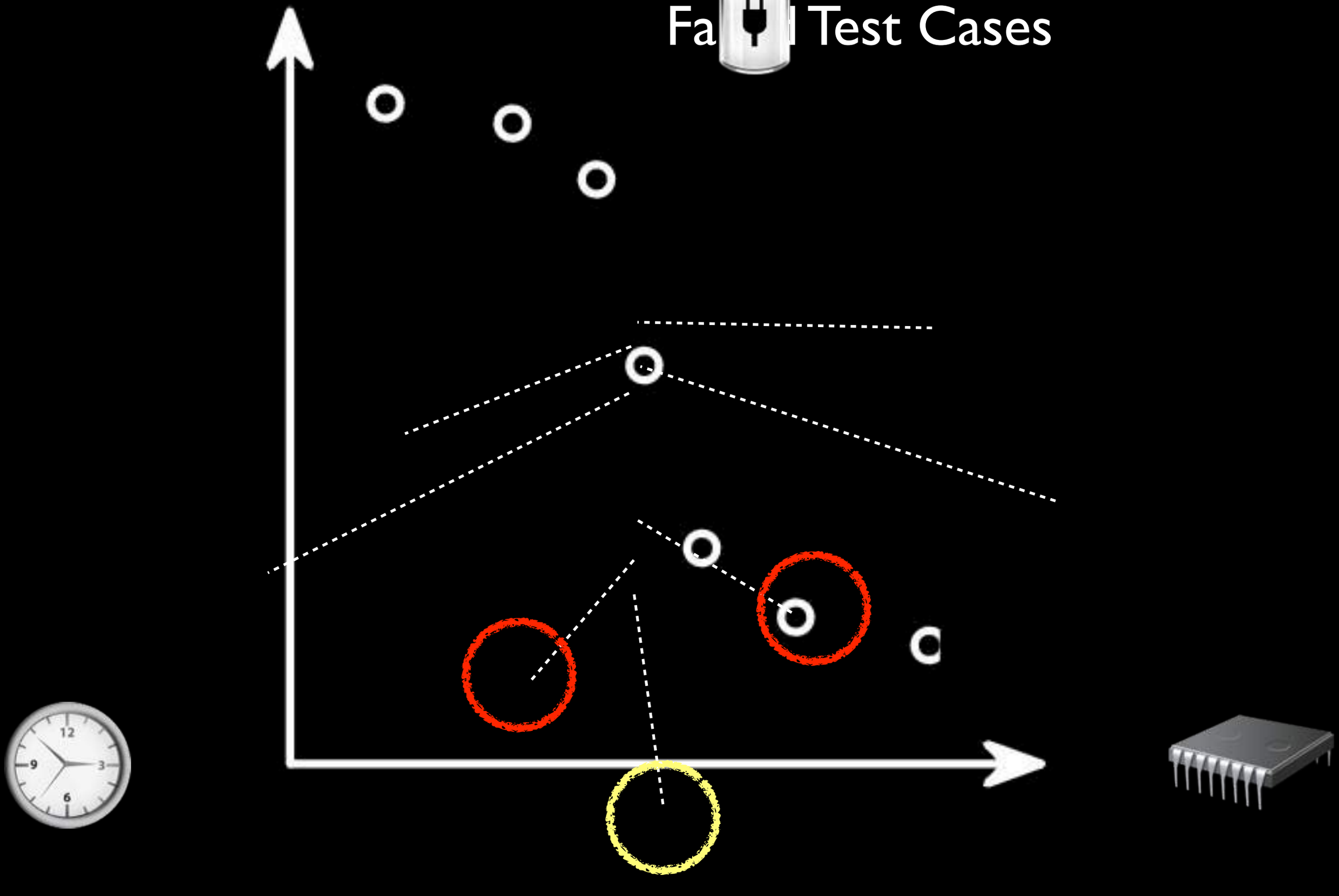
each circle is a program found by a machine

Pareto Front

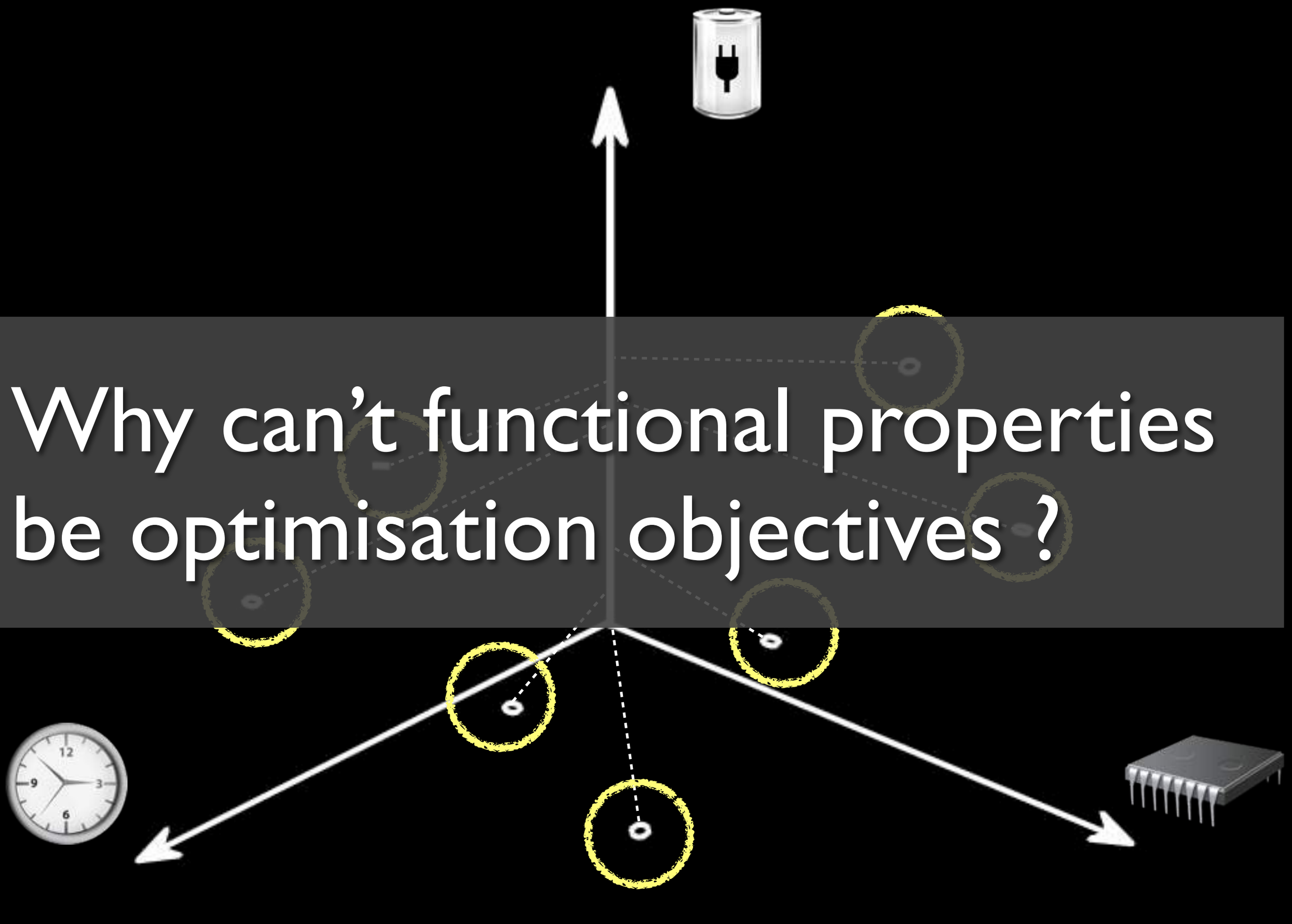


different non functional properties have different pareto program fronts

Fa l Test Cases

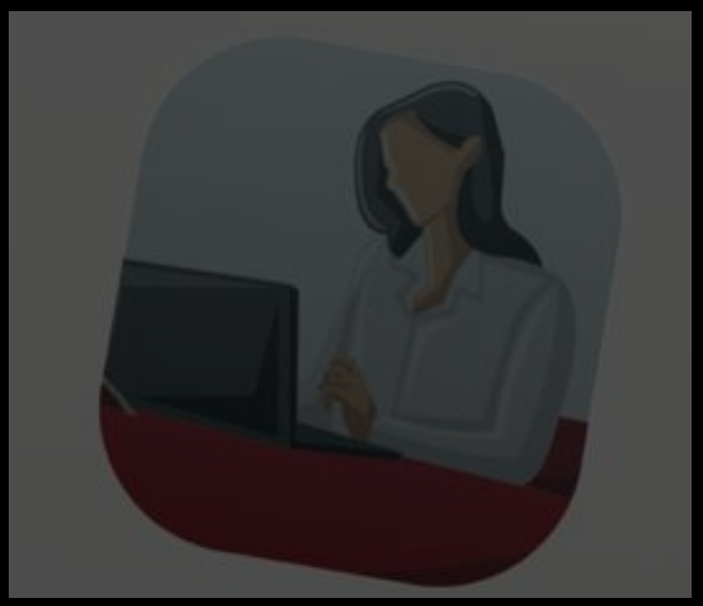


Why can’t functional properties be optimisation objectives ?

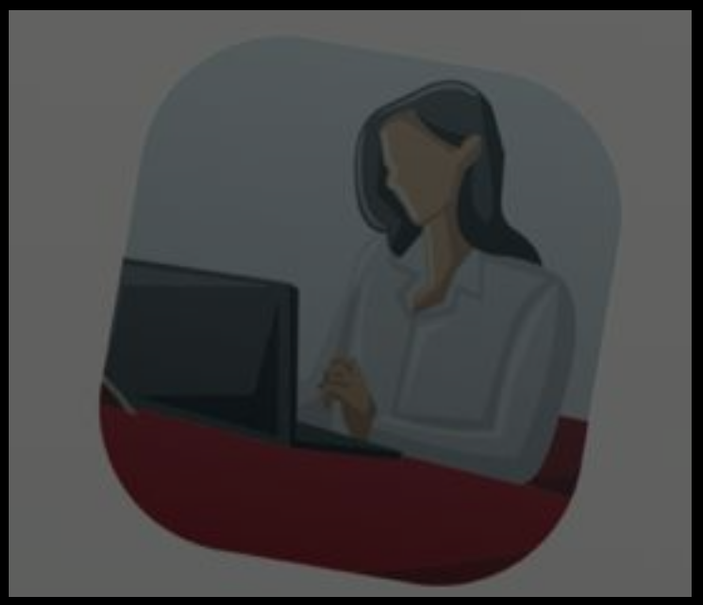


Optimisation

Optimisation



2.5 times faster but
failed 1 test case?



Optimisation

double the battery life
but failed 2 test cases?

Conformant GI

VS.

Heretical GI

functional correctness is king

conforms to traditional views

Conformant GI	VS.	Heretical GI
functional correctness is king		correctness means having sufficient resources for computation
conforms to traditional views		conforms to a compelling new orthodoxy

Conformant GI	VS.	Heretical GI
functional correctness is king		correctness means having sufficient resources for computation
conforms to traditional views		conforms to a compelling new orthodoxy

there's nothing correct about a flat battery

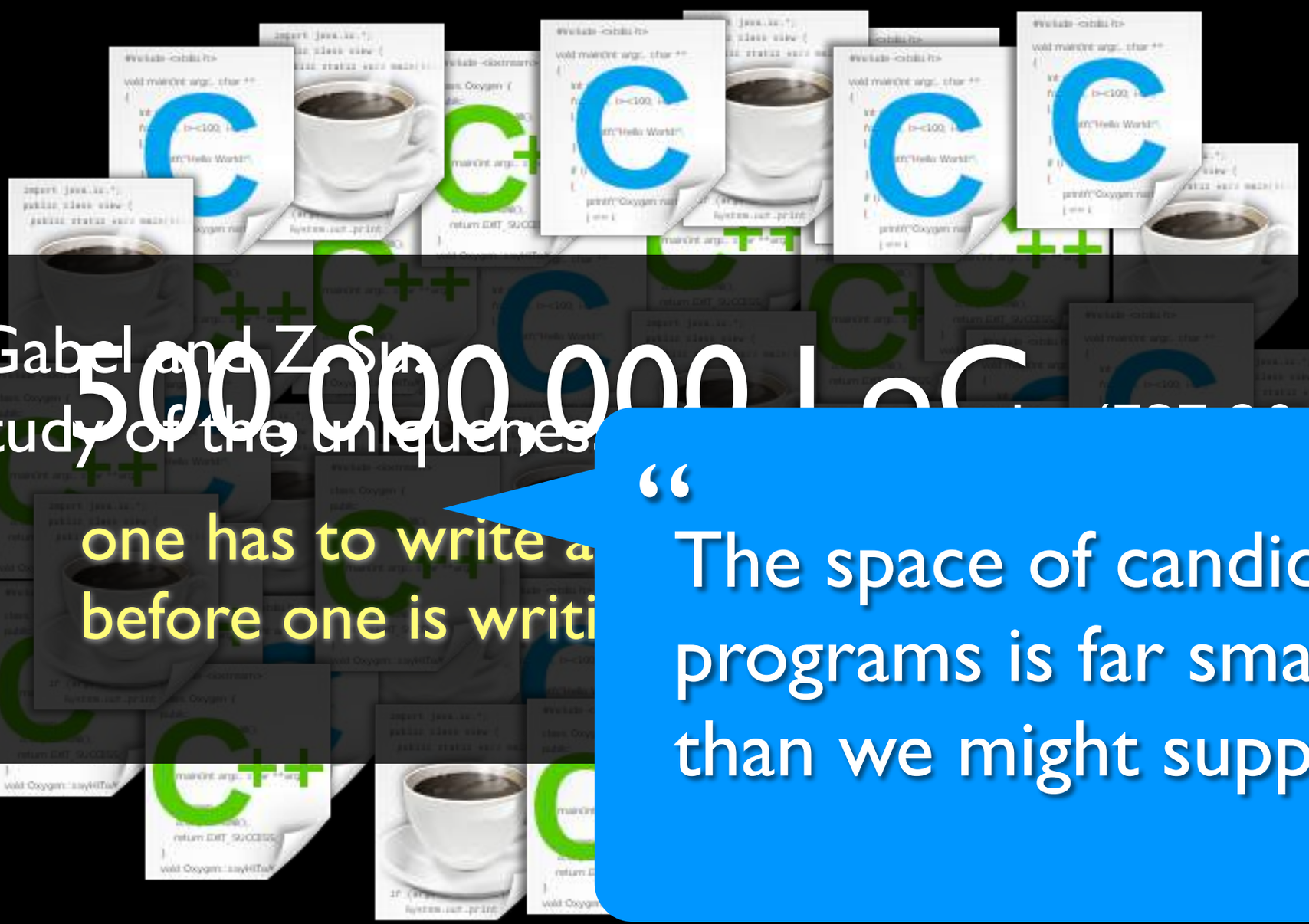
can it work ?

Software Uniqueness

500,000,000 LoC

one has to write approximately 6 statements
before one is writing unique code

Software Uniqueness



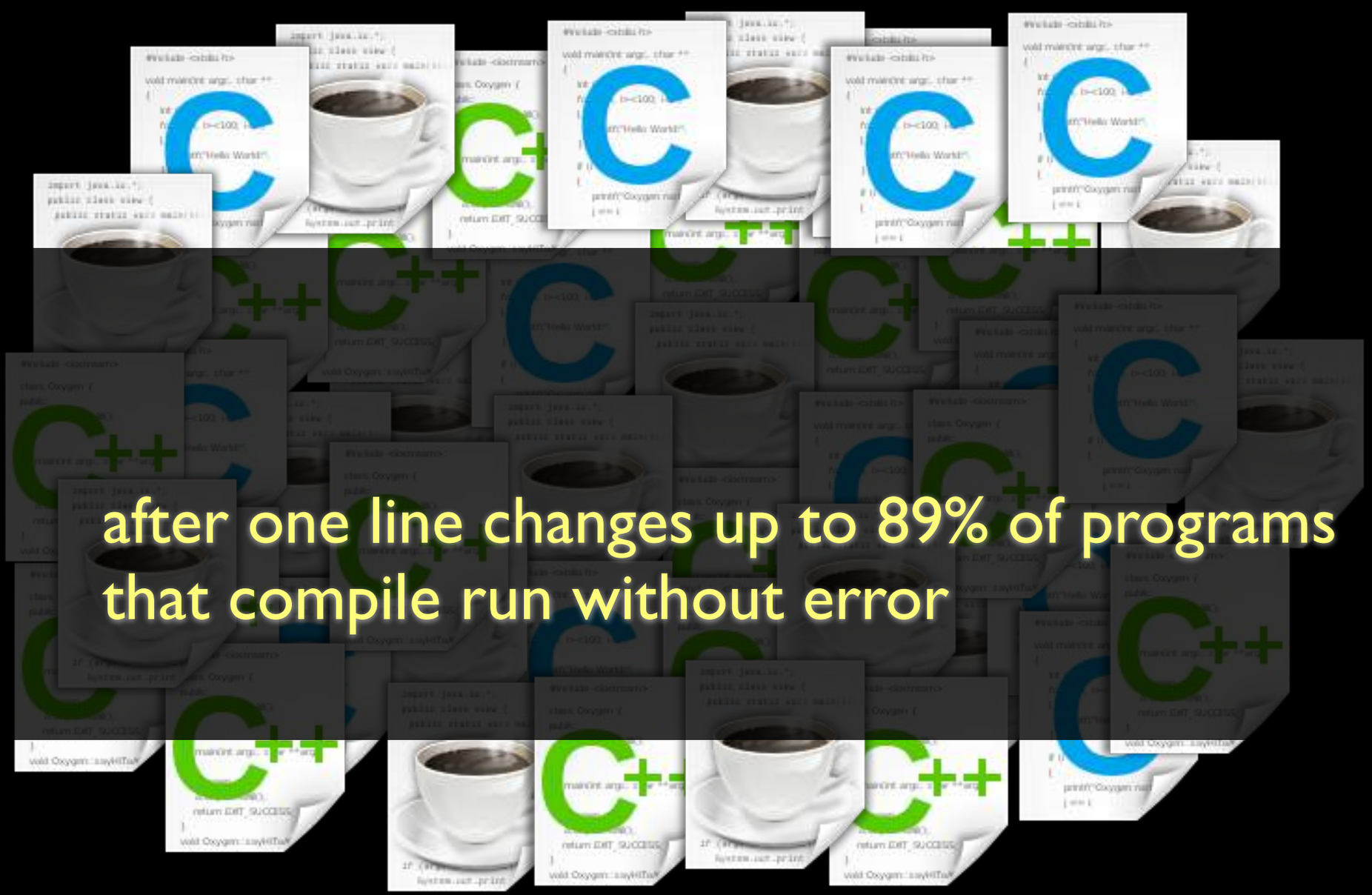
M. Gabel and Z. Su:
A study of the uniqueness

500 000 000 LoC

one has to write a
before one is writi

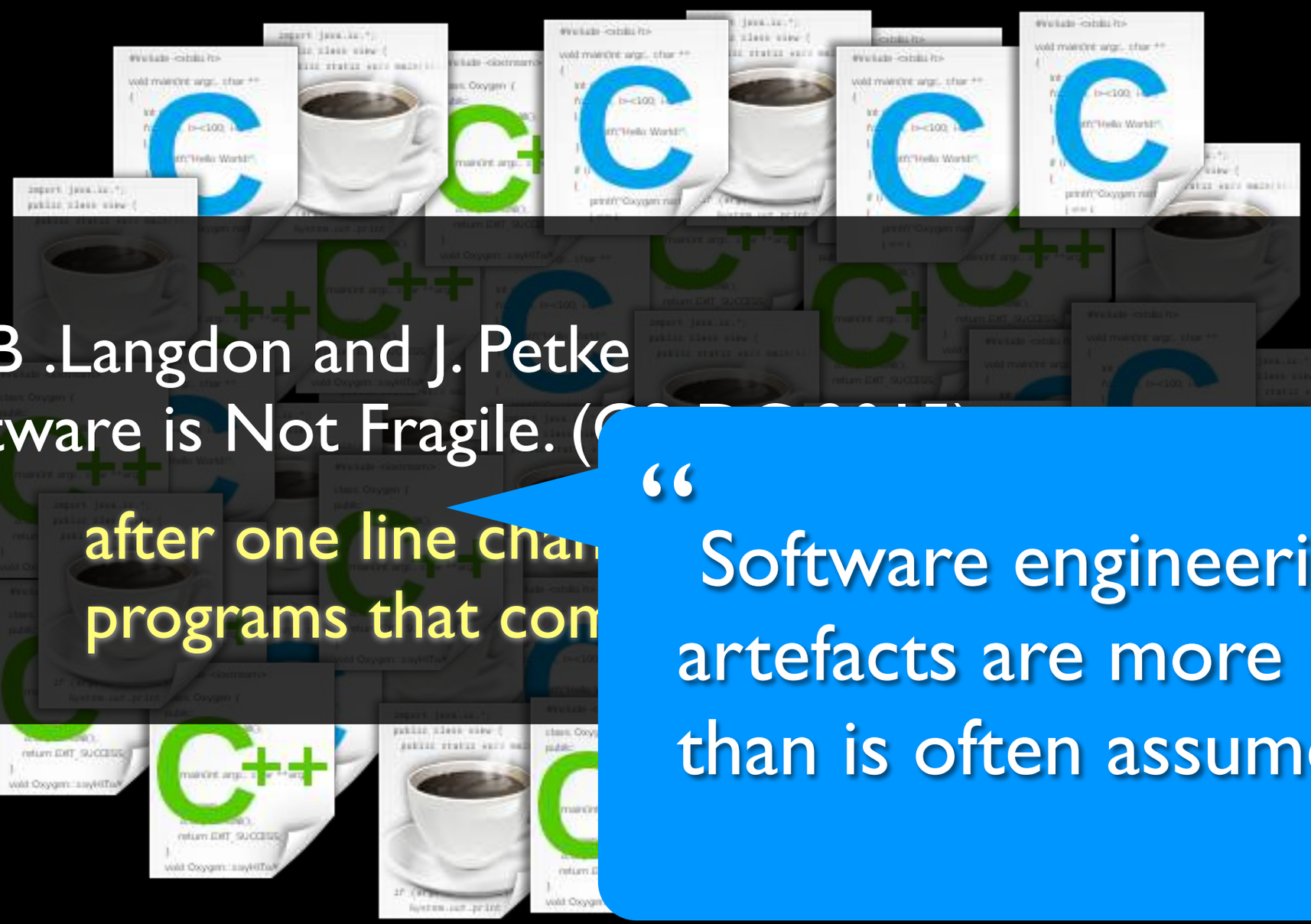
“The space of candidate programs is far smaller than we might suppose.”

Software Robustness



after one line changes up to 89% of programs that compile run without error

Software Robustness



W. B .Langdon and J. Petke
Software is Not Fragile. (

after one line cha
programs that con

> “Software engineering artefacts are more robust than is often assumed.”

Genetic Improvement for Software Specialisation

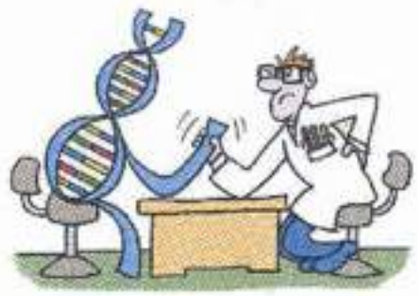
Genetic and Evolutionary Computation Conference

July 12-16, 2014
Vancouver, British Columbia

Winners of the
2014 Humies Silver Award:

Justyna Petke, Mark Harman,
William B. Langdon, Westley Weimer

Using Genetic Improvement and Code Transplants to Specialize a C++ Program to a Problem Class



Association for Computing Machinery

sigevo

Question

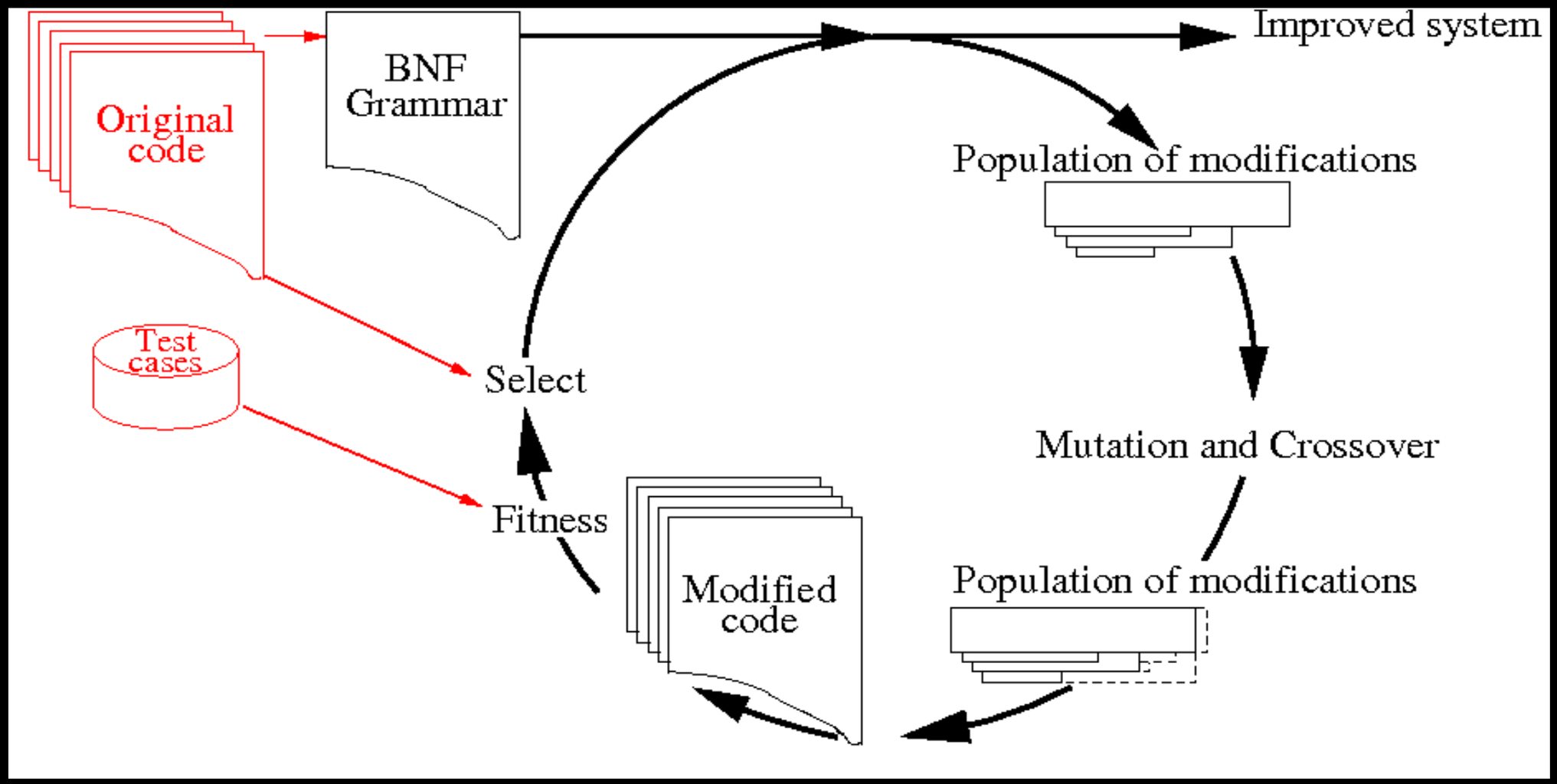
Can we improve the efficiency of an already highly-optimised piece of software using genetic programming?

Contributions

Introduction of multi-donor software transplantation

Use of genetic improvement as means to specialise software

Genetic Improvement



Program Representation

Changes at the level of lines of source code

Each individual is composed of a list of changes

Specialised grammar used to preserve syntax

Example

```
 <Solver_135>    ::=     " if" <IF_Solver_135> "  return false;\n"
<IF_Solver_135> ::=     "(!ok)"
<Solver_138>    ::=     "" <_Solver_138> "{Log_count64++;/*138*/}\n"
<_Solver_138>   ::=     "sort(ps);"
<Solver_139>    ::=     "Lit p; int i, j;\n"
<Solver_140>   ::=    "for(" <for1_Solver_140> ";" <for2_Solver_140> ";" <for3_Solver_140> ") {\n"
<for1_Solver_140>       ::=     "i = j = , p = lit_Undef"
<for2_Solver_140>       ::=     "i < ps.size()"
<for3_Solver_140>       ::=     "i++"
```

Code Transplants

GP has access to both:

- the *host* program to be evolved
- the *donor* program(s)

Mutation

Addition of one of the following operations:

delete

copy

replace

Example

```
<_Solver_135>

<_Solver_138>+<_Solver_140>

<for3_Solver_140><for3_Solver_836>
```

Crossover

Concatenation of two individuals

by appending two lists of mutations

```
<_Solver_135>

<_Solver_138>+<_Solver_140>

-------------------------------------

 <_Solver_135>  <_Solver_138>+<_Solver_140>
```

Fitness

Based on solution quality and

Efficiency in terms of lines of source code

Avoids environmental bias

Fitness

Test cases are sorted into groups

One test case is sampled uniformly from each group

Avoids overfitting

Selection

Fixed number of generations

Fixed population size

Initial population contains single-mutation individuals

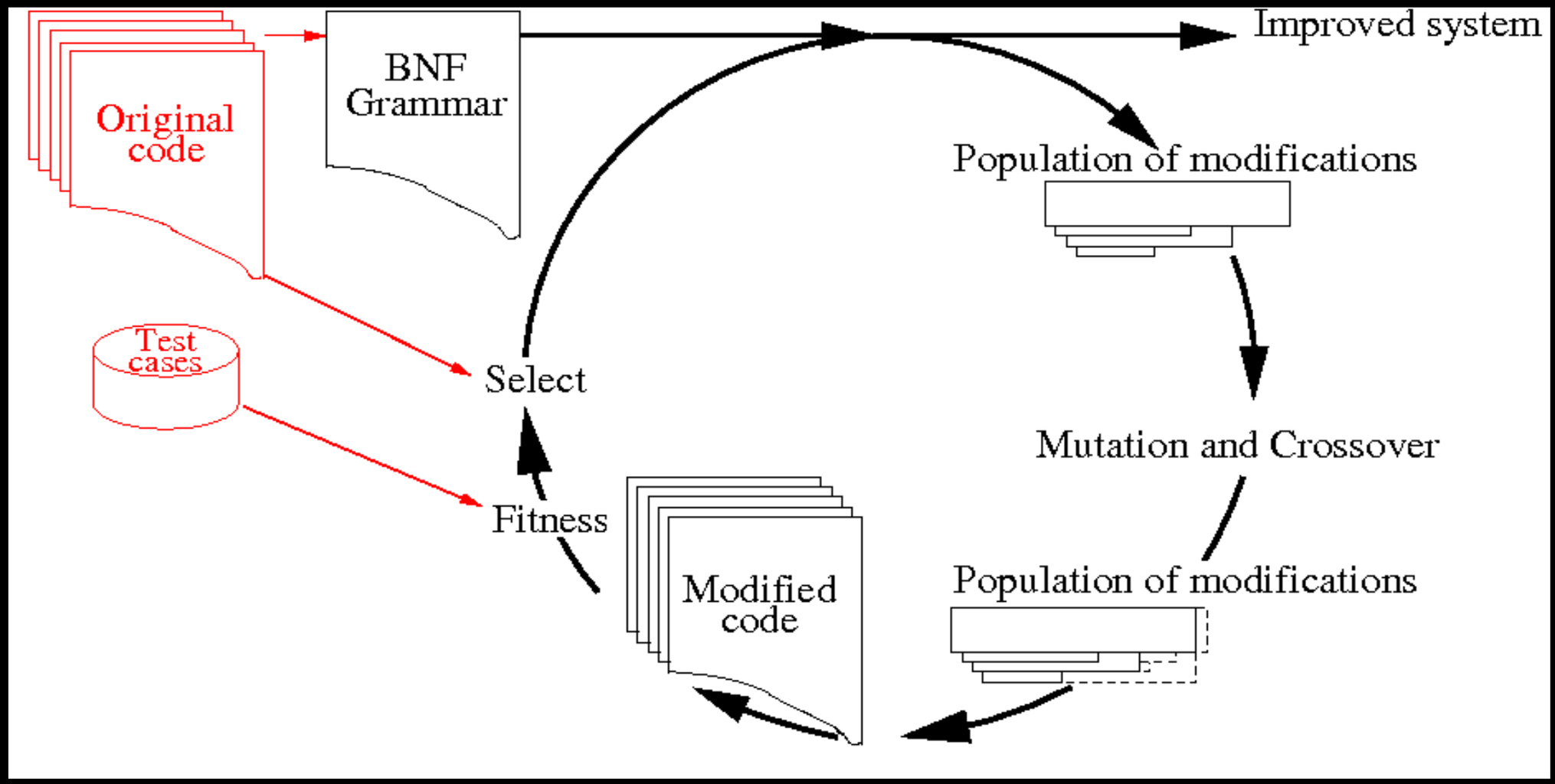
Selection

Top-half of population selected

Based on a threshold fitness value

Mutation and Crossover applied

Genetic Improvement



Filtering

Mutations in best individuals are often independent

Greedy approach used to combine best individuals

Question

Can we improve the efficiency of an already highly-optimised piece of software using genetic programming?

Motivation for choosing a SAT solver

Boolean satisfiability (SAT) example:

$$x_1 \vee x_2 \vee \neg x_4$$

$$\neg x_2 \vee \neg x_3$$

- x_i : a Boolean variable
- x_i, $\neg x_i$: a literal
- $\neg x_2 \vee \neg x_3$: a clause

Motivation for choosing a SAT solver

Bounded Model Checking

Planning

Software Verification

Automatic Test Pattern Generation

Combinational Equivalence Checking

Combinatorial Interaction Testing

and many other applications..

Motivation for choosing a SAT solver

MiniSAT-hack track in SAT solver competitions

Question

Can we evolve a version of the MiniSAT solver that is faster than any of the human-improved versions of the solver?

Experiments: Setup

Solvers used:

MiniSAT2-070721

Test cases used:

$\sim$ 2.5% improvement for general benchmarks (SSBSE'13)

Motivation for choosing a SAT solver

MiniSAT-hack track in SAT solver competitions

- good source for software transplants

Question

Can we evolve a version of the MiniSAT solver that is faster than any of the human-improved versions of the solver for a particular problem class?

Experiments: Setup

Solvers used:

MiniSAT2-070721

Test cases used:

from Combinatorial Interaction Testing field

Combinatorial Interaction Testing

Use of SAT-solvers limited due to poor scalability

SAT benchmarks containing millions of clauses

It takes hours to days to generate a CIT test suite using SAT

Experiments: Setup

Host program:

MiniSAT2-070721 (478 lines in main algorithm)

Donor programs:

MiniSAT-best09 (winner of ’09 MiniSAT-hack competition)

MiniSAT-bestCIT (best for CIT from ’09 competition)

- total of 104 new lines

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63

Question

How much runtime improvement can we achieve?

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63
MiniSAT-gp	best09	0.93	0.95

Results

Donor: best09

13 delete, 9 replace, 1 copy

Among changes:

3 assertions removed

1 deletion on variable used for statistics

Results

Mainly if and for statements switched off

Decreased iteration count in for loops

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63
MiniSAT-gp	best09	0.93	0.95
MiniSAT-gp	bestCIT	0.72	0.87

Results

Donor: bestCIT

1 delete, 1 replace

Among changes:

1 assertion deletion

1 replace operation triggers 95% of donor code

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63
MiniSAT-gp	best09	0.93	0.95
MiniSAT-gp	bestCIT	0.72	0.87
MiniSAT-gp	best09+bestCIT	0.94	0.96

Results

Donor: best09+bestCIT

50 delete, 20 replace, 5 copy

Among changes:

5 assertions removed

4 semantically equivalent replacements

3 operations used for statistics removed

$\sim$ half of the mutations remove dead code

Results

Solver	Donor	Lines	Seconds
MiniSAT (original)	—	1.00	1.00
MiniSAT-best09	—	1.46	1.76
MiniSAT-bestCIT	—	0.72	0.87
MiniSAT-best09+bestCIT	—	1.26	1.63
MiniSAT-gp	best09	0.93	0.95
MiniSAT-gp	bestCIT	0.72	0.87
MiniSAT-gp	best09+bestCIT	0.94	0.96
MiniSAT-gp-combined	best09+bestCIT	**0.54**	**0.83**

Results

Combining results:

37 delete, 15 replace, 4 copy

56 out of 100 mutations used

Among changes:

8 assertion removed

95% of the bestCIT donor code executed

Conclusions

Introduced multi-donor software transplantation

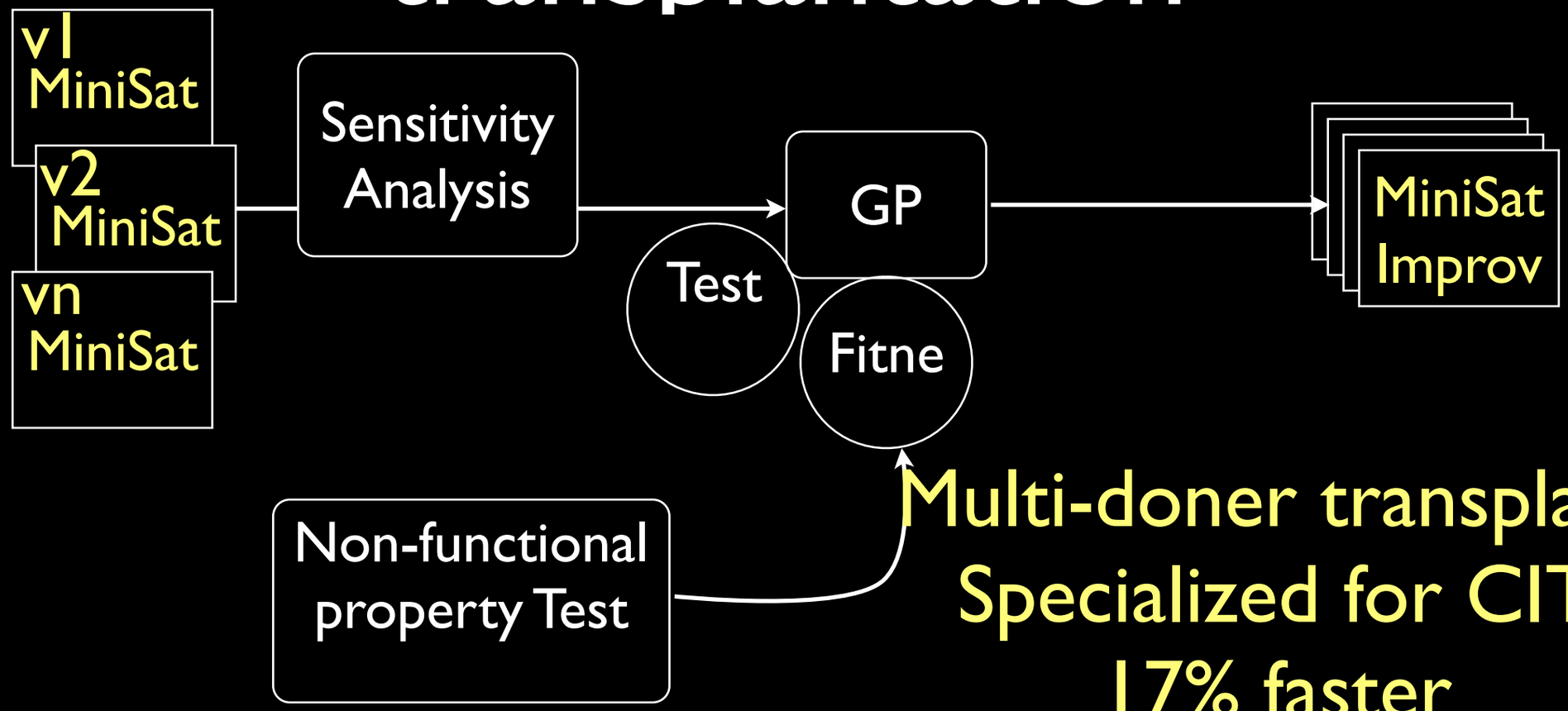
Used genetic improvement as means to specialise software

Achieved 17% runtime improvement on MiniSAT

for the Combinatorial Interaction Testing domain

by combining best individuals

Inter version transplantation

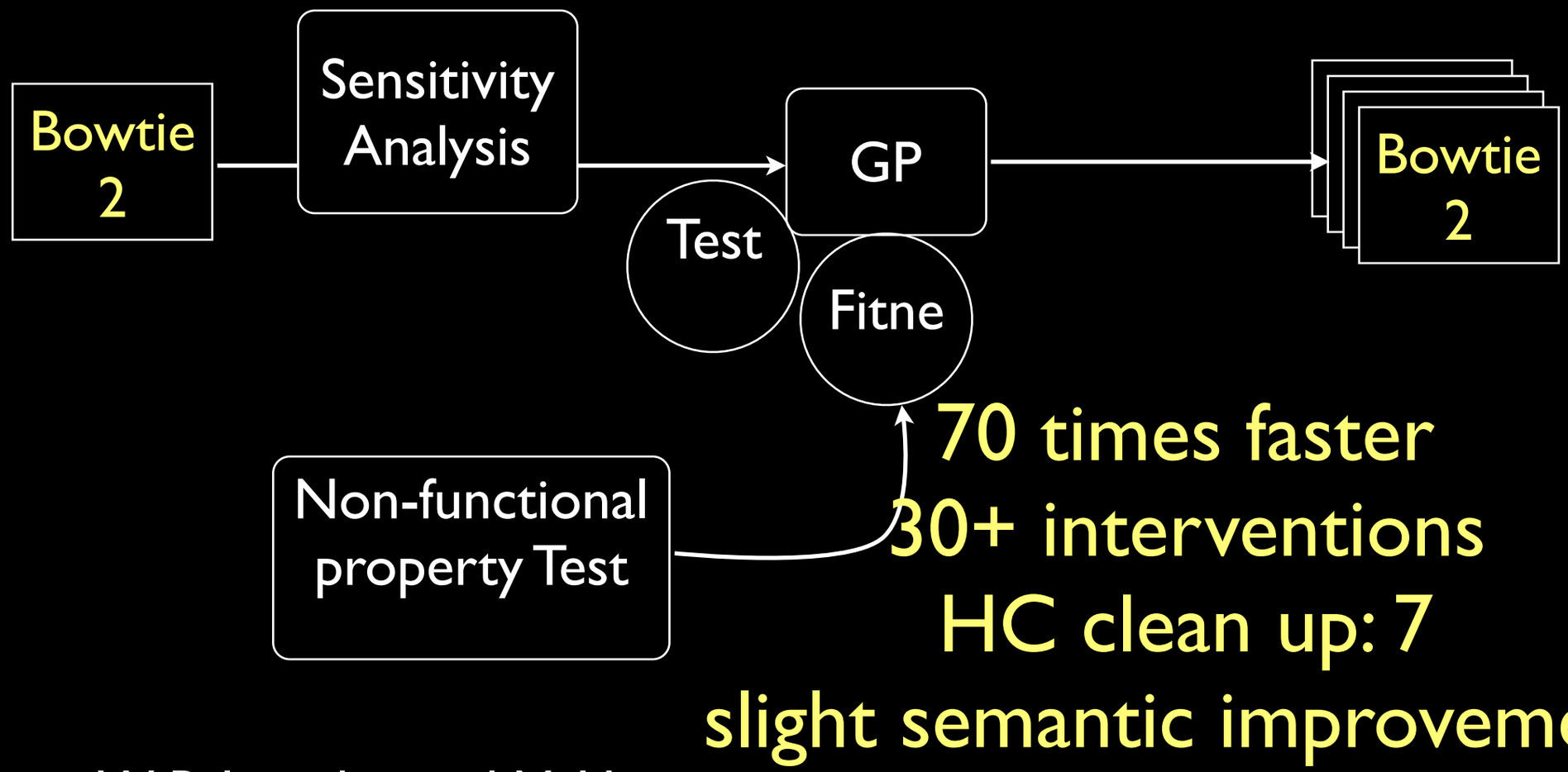


Multi-doner transplant
Specialized for CIT
17% faster

Justyna Petke, Mark Harman, William B. Langdon and Westley Weimer
Using Genetic Improvement & Code Transplants to Specialise a C++ program
to a Problem Class (EuroGP'14)

GECCO Humie silver medal

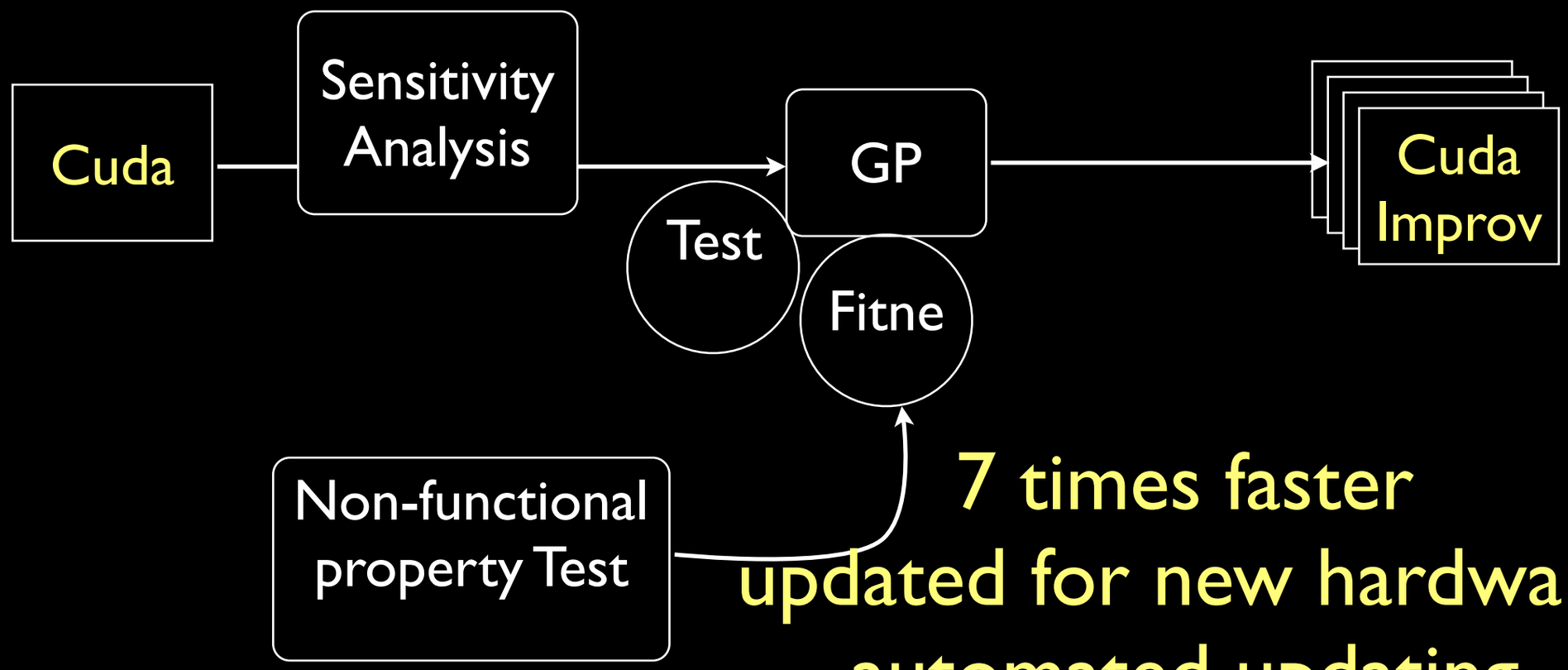
Genetic Improvement of Programs



70 times faster
30+ interventions
HC clean up: 7
slight semantic improvement

W. B. Langdon and M. Harman
Optimising Existing Software with Genetic Programming. TEC 2015

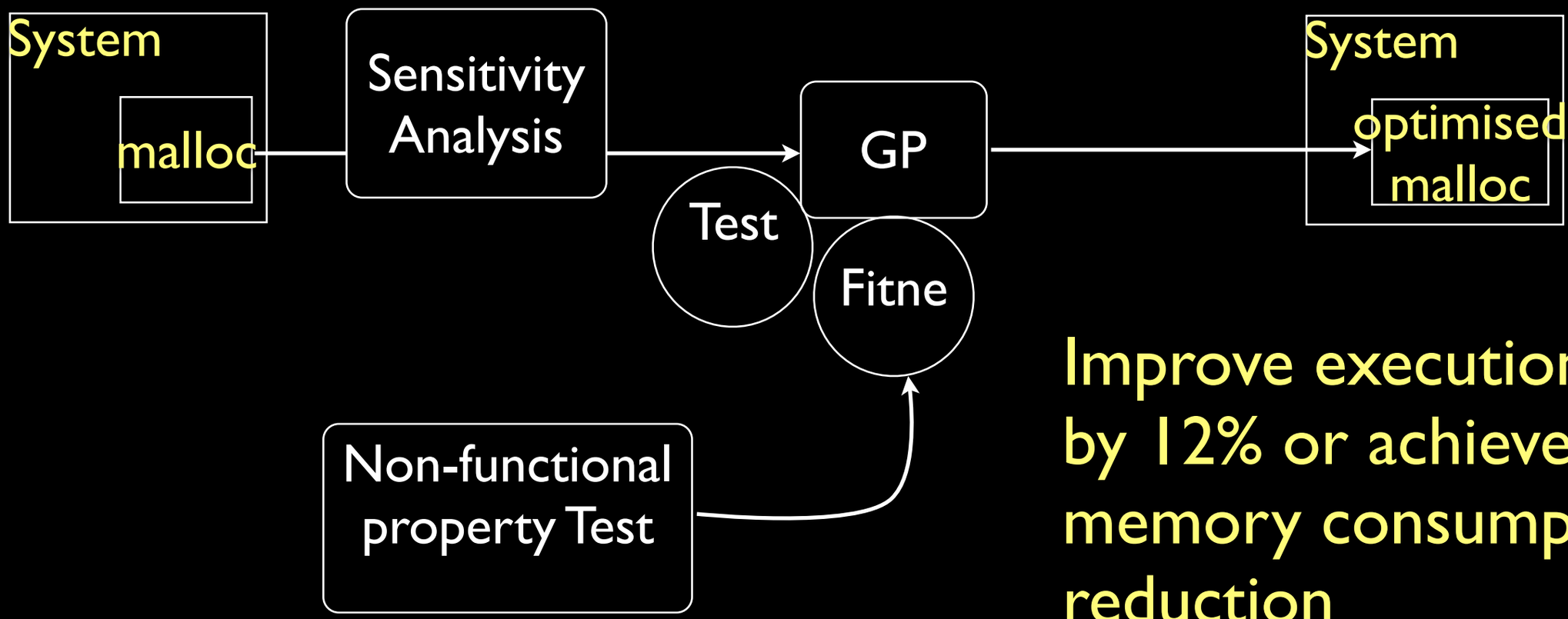
Genetic Improvement of Programs



7 times faster
updated for new hardware
automated updating

W. B. Langdon and M. Harman
Genetically Improved CUDA C++ Software, EuroGP 2014

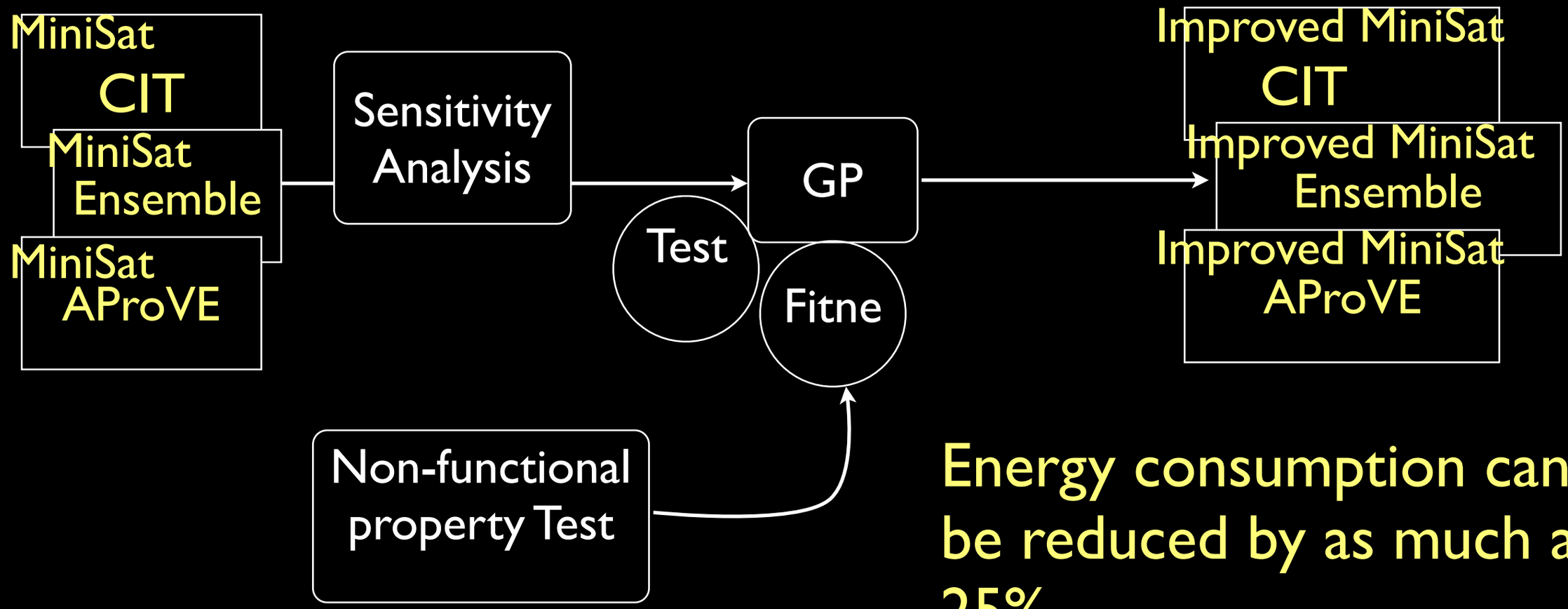
Memory speed trade offs



Improve execution time by 12% or achieve a 21% memory consumption reduction

Fan Wu, Westley Weimer, Mark Harman, Yue Jia and Jens Krinke
Deep Parameter Optimisation
Conference on Genetic and Evolutionary Computation (GECCO 2015)

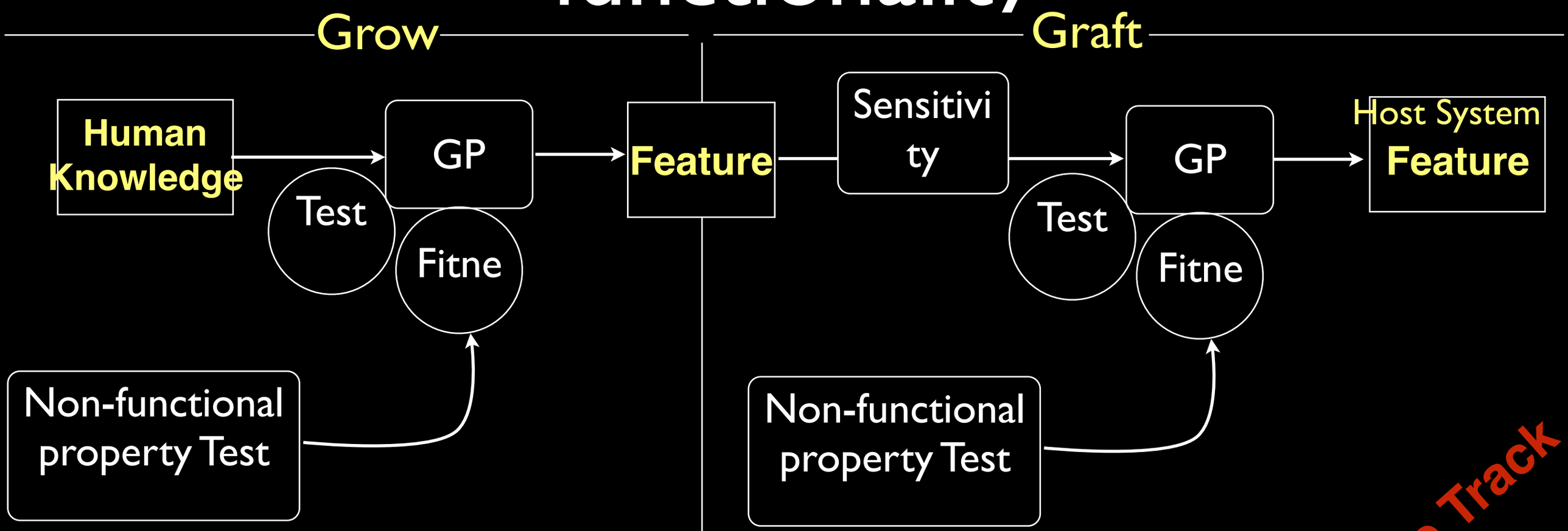
Reducing energy consumption



Energy consumption can be reduced by as much as 25%

Bobby R. Bruce Justyna Petke Mark Harman
Reducing Energy Consumption Using Genetic Improvement
Conference on Genetic and Evolutionary Computation (GECCO 2015)

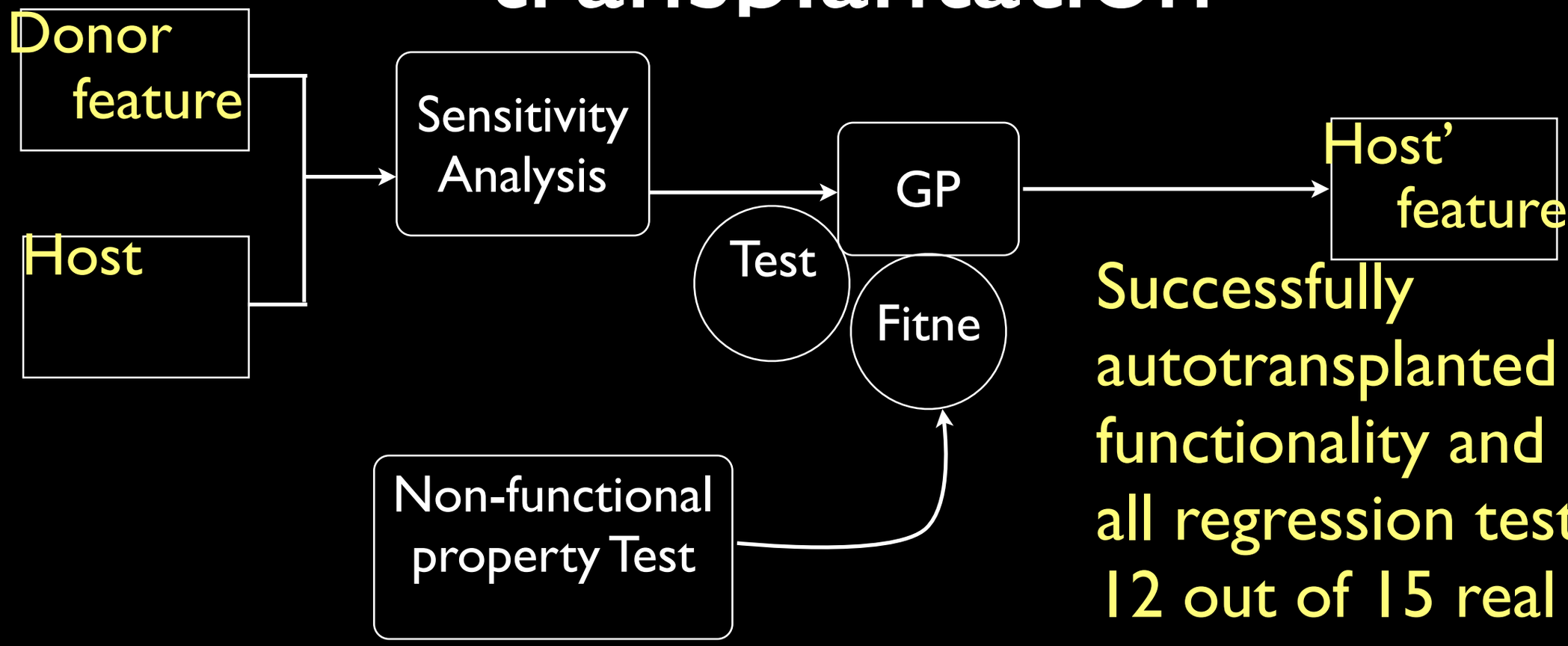
Grow and graft new functionality



Mark Harman, Yue Jia and Bill Langdon,
Babel Pidgin: SBSE can grow and graft entirely new functionality into a real world system
Symposium on Search-Based Software Engineering SSBSE 2014. (Challenge track)

Challenge Track Award

Real world cross system transplantation



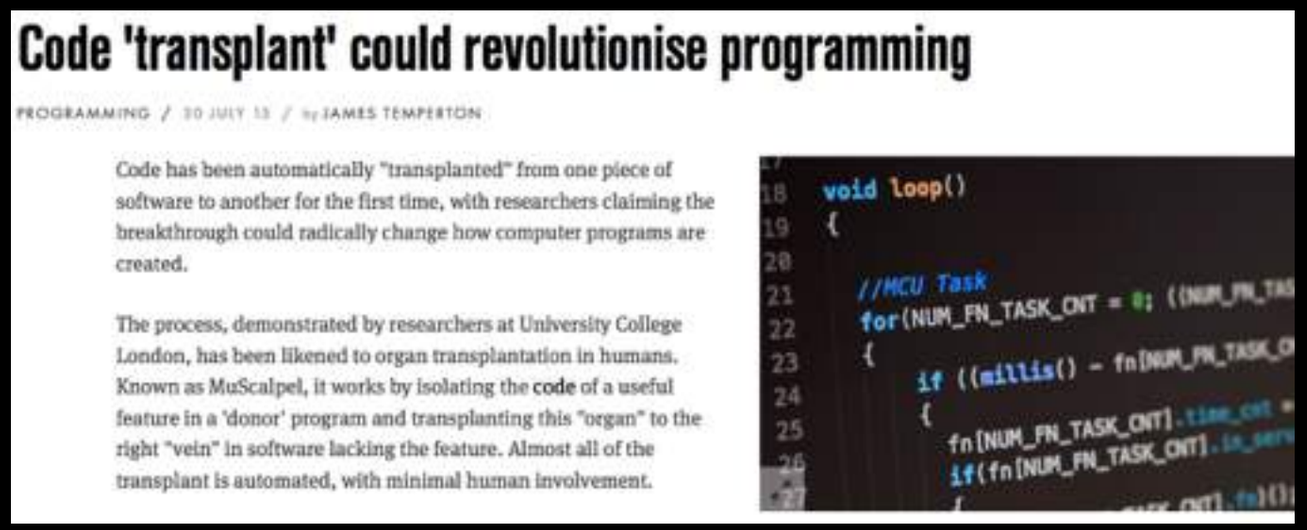
Successfully autotransplanted new functionality and passed all regression tests for 12 out of 15 real world systems

Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke
Automated Software Transplantation (ISSTA 2015)

Automated Software Transplantation

E.T. Barr, M. Harman, Y. Jia, A. Marginean & J. Petke

ACM Distinguished Paper Award at ISSTA 2015



2647 shares of

article in WIRED.CO.UK

coverage in BBC WORLD SERVICE Click

Why A ansplantation?

~100 players

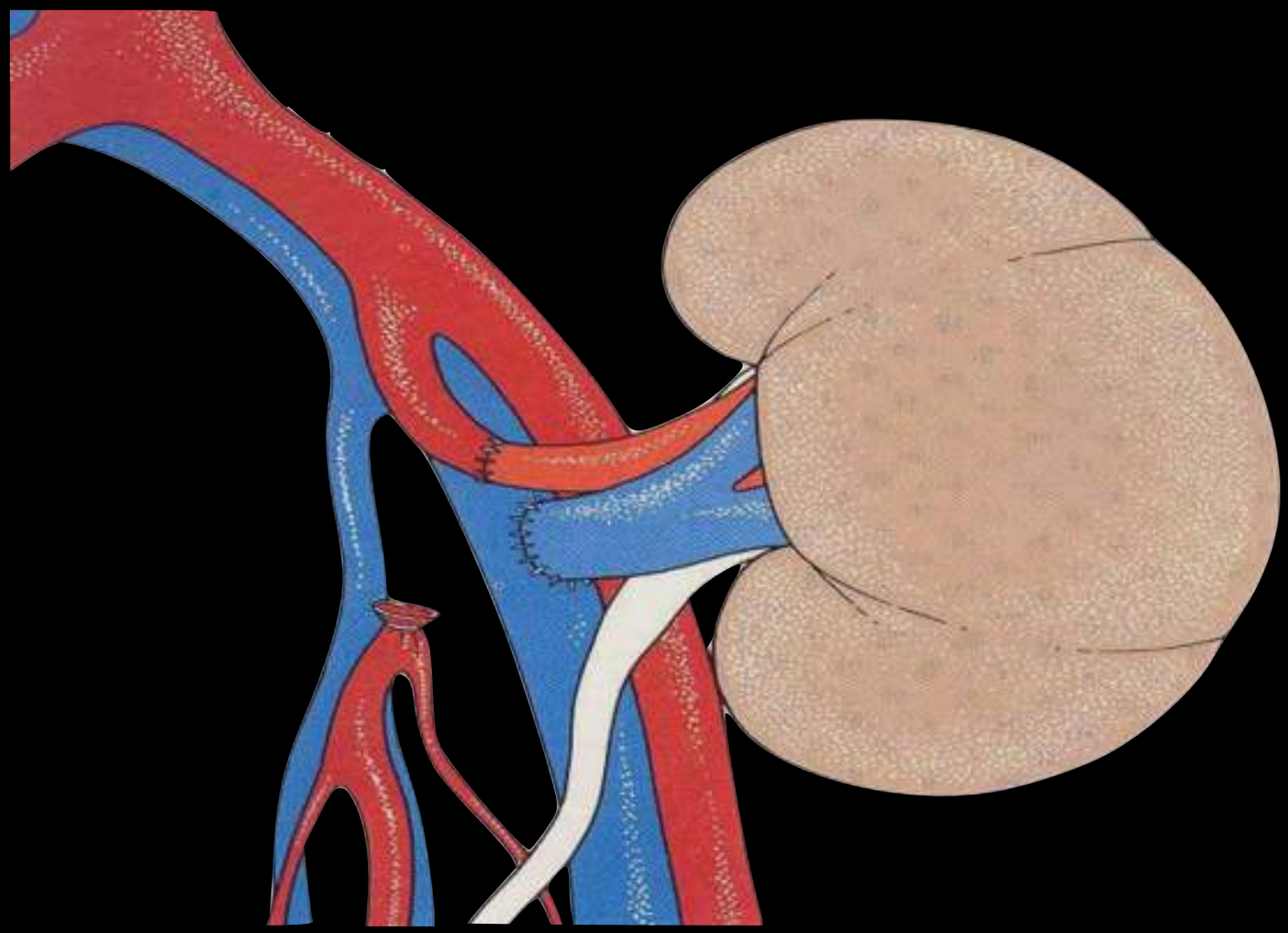
Video Player

Why not handle H.264?

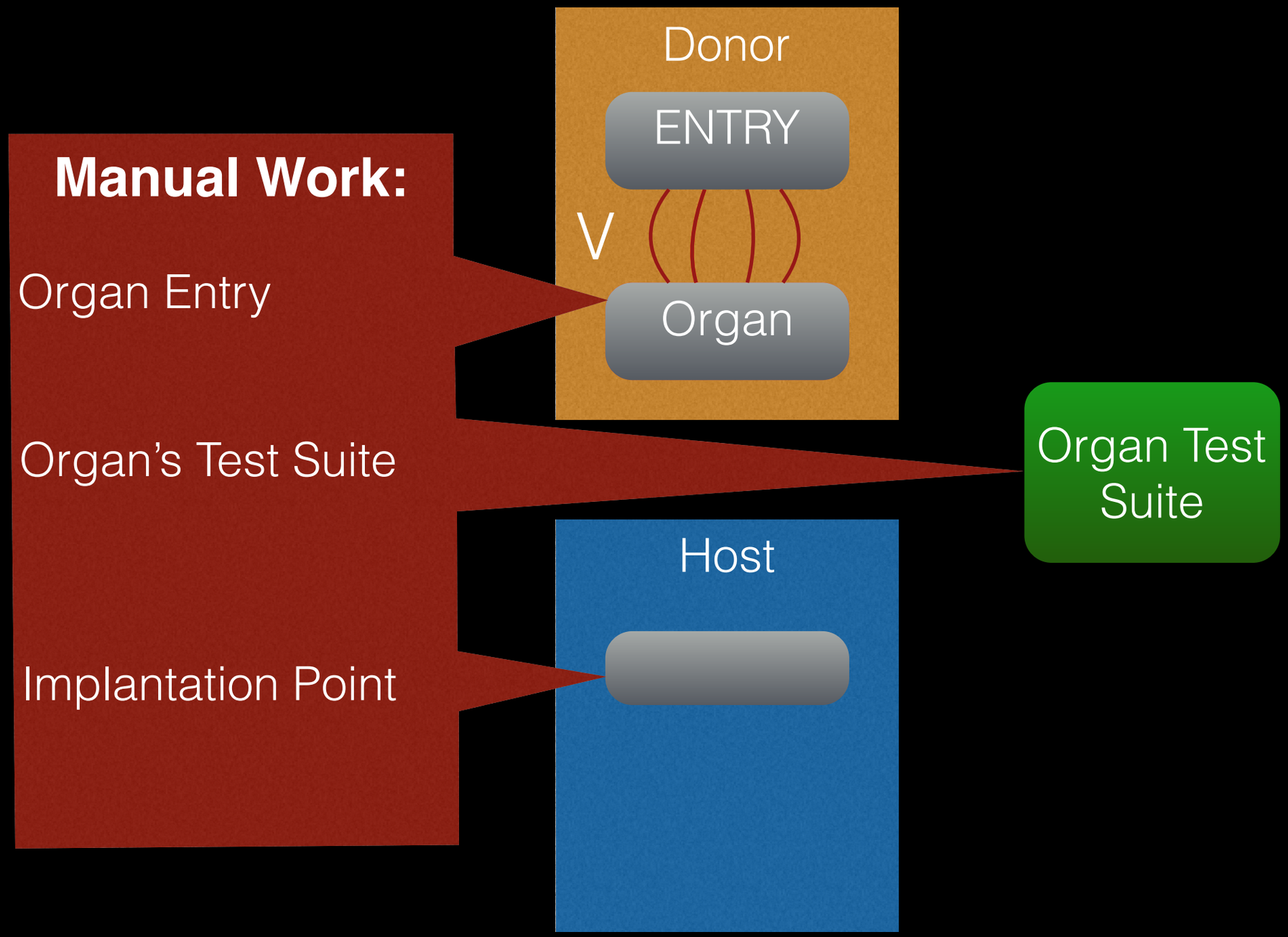
Check open source repositories

Start from scratch

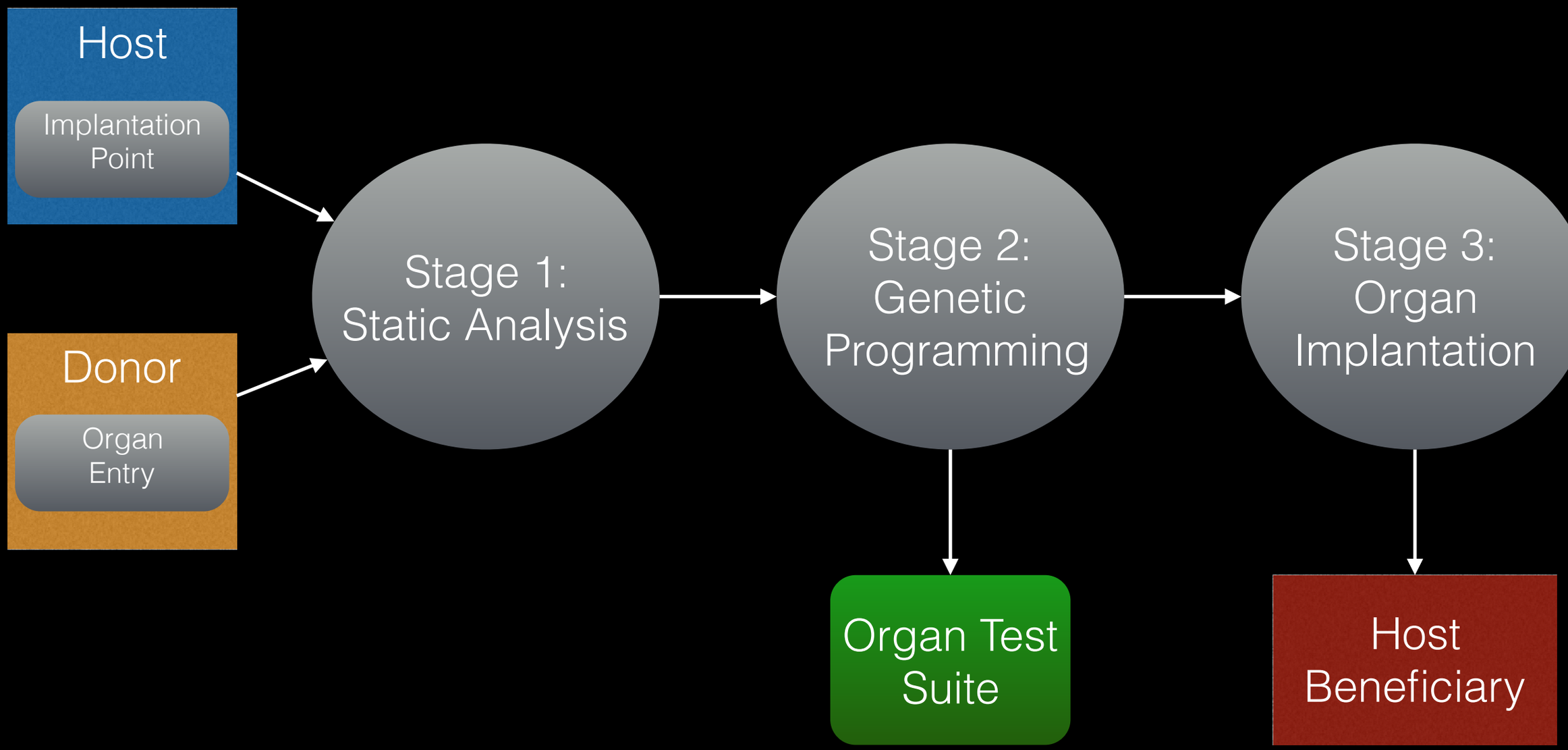
Human Organ Transplantation



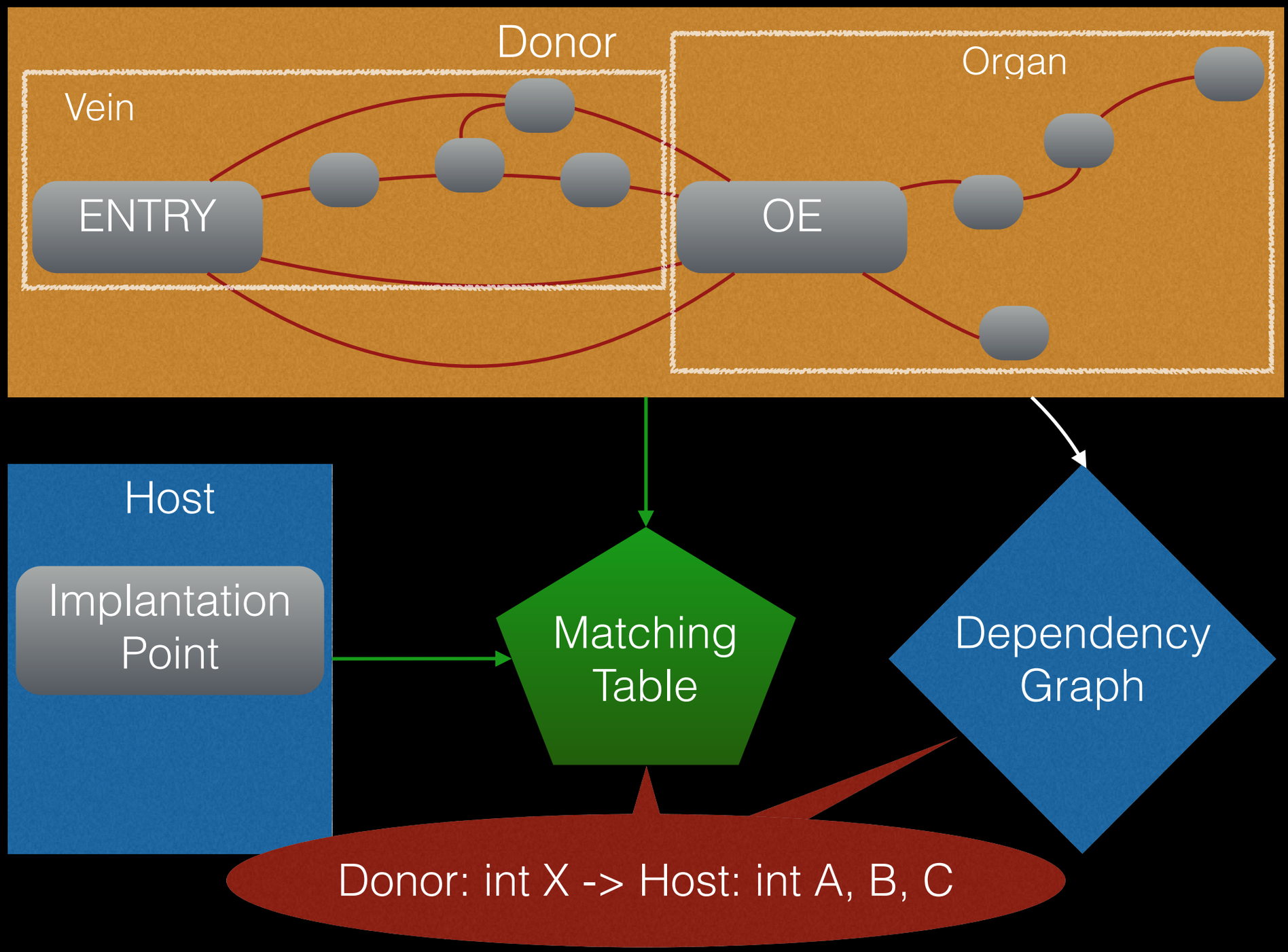
Automated Software Transplantation



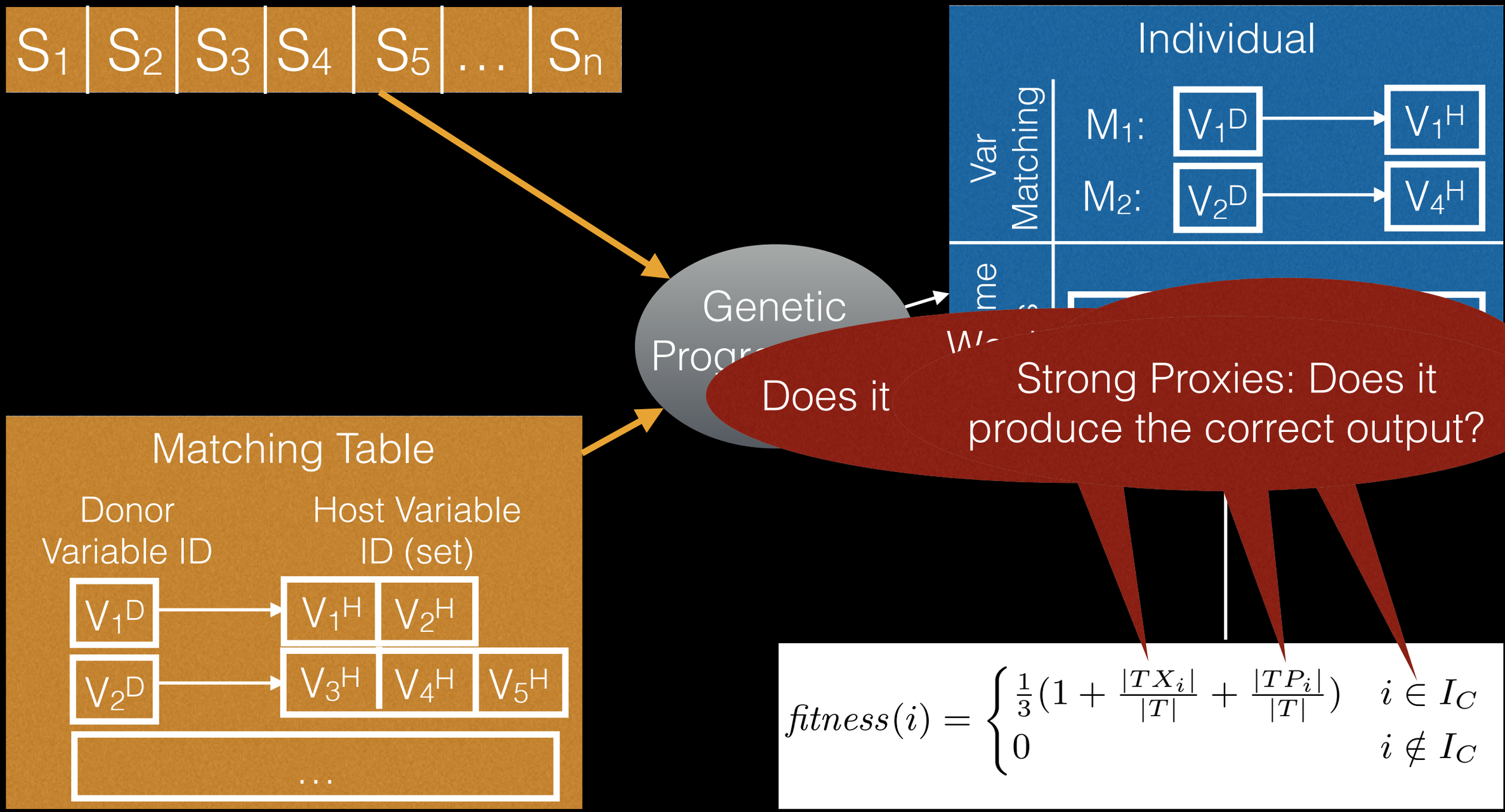
μTrans


Host
Implantation Point
Donor
Organ Entry
Stage 1: Static Analysis
Stage 2: Genetic Programming
Stage 3: Organ Implantation
Organ Test Suite
Host Beneficiary

Stage 1 — Static Analysis



Stage 2 — GP



$$fitness(i) = \begin{cases} \frac{1}{3}(1 + \frac{|TX_i|}{|T|} + \frac{|TP_i|}{|T|}) & i \in I_C \\ 0 & i \notin I_C \end{cases}$$

Research Questions

Acceptance Tests

Host

Donor

Is autotransplantation useful?

ch Qu

Do we break the initial functionality?

Have we really added new functionality?

Empirical Study

15 Transplantations
300 Runs
5 Donors
3 Hosts

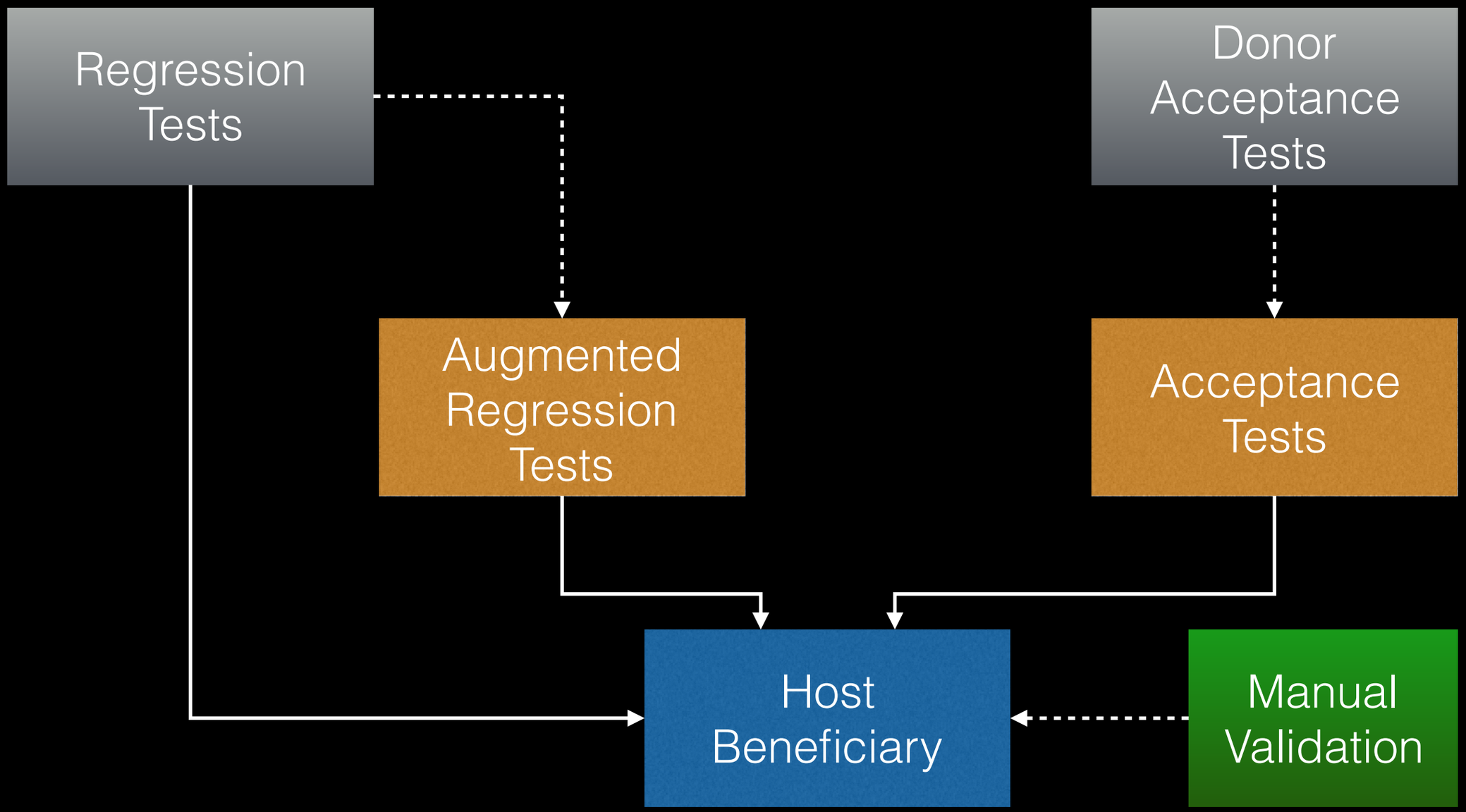
Case Study:

H.264 Encoding Transplantation

How about the computational effort?

Is autotransplantation useful?

Validation


Regression Tests
Donor Acceptance Tests
Augmented Regression Tests
Acceptance Tests
Host Beneficiary
Manual Validation

Subjects

Subjects	Type	Size KLOC
Idct	Donor	2.3
Mytar	Donor	0.4
Cflow	Donor	25
Webserver	Donor	1.7
TuxCrypt	Donor	2.7
Pidgin	Host	363
Cflow	Host	25
SoX	Host	43
Case Study		
x264	Donor	63
VLC	Host	422

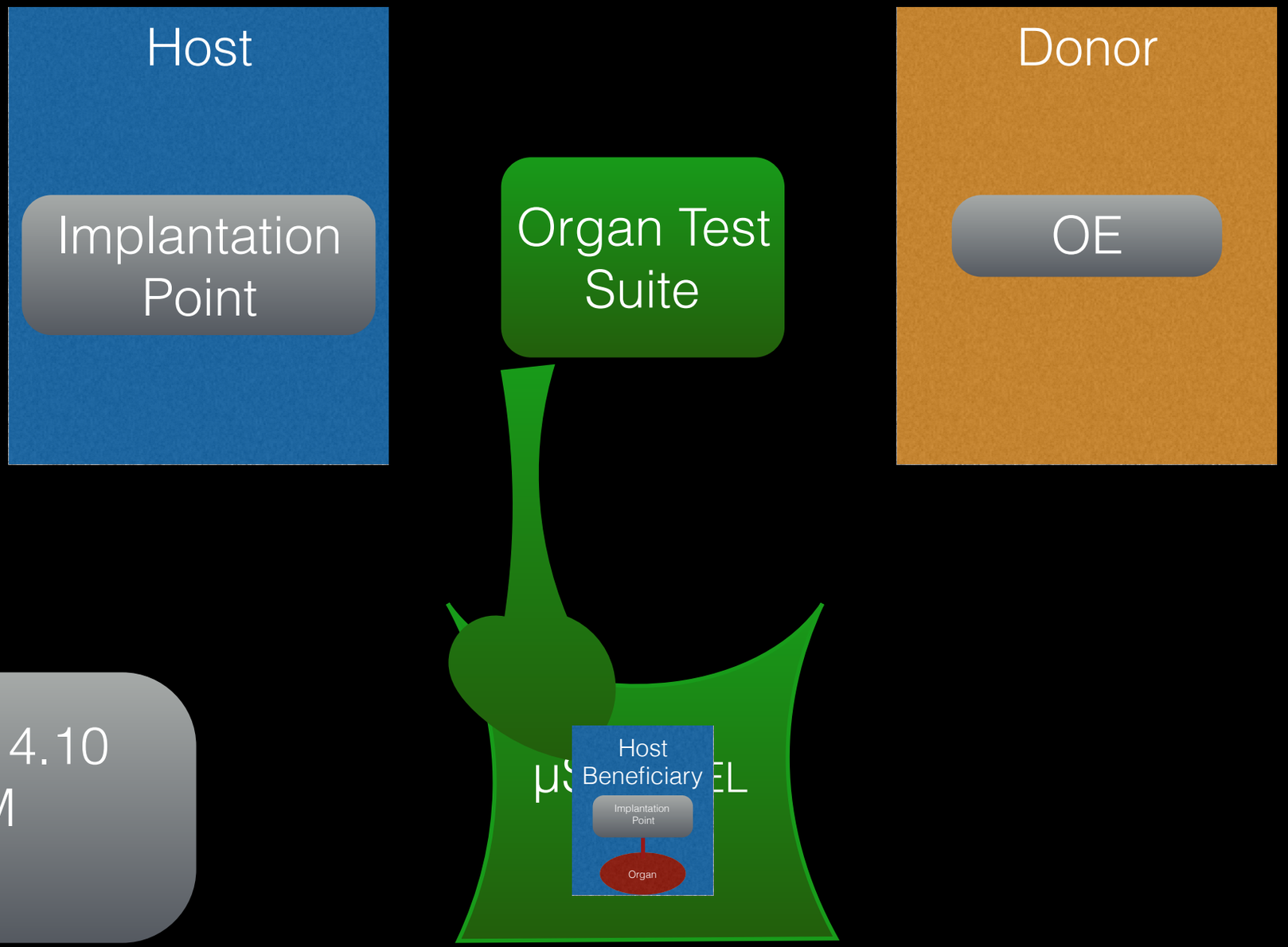
Minimal size: 0.4k

Max size: 422k

Average Donor:16k

Average Host: 213k

Experimental Methodology and Setup



64 bit Ubuntu 14.10
16 GB RAM
8 threads

Empirical Study

Donor	Host	All Passed	Regression	Regression++	Acceptance
Idct	Pidgin	16	20	17	16
Mytar	Pidgin	16	20	18	20
Web	Pidgi			0	18
Cflow	Pidg			5	16
Tux	Pidg			7	16
Idct	Cflo			6	16
Mytar	Cflo			7	20
Web	Cflo			0	17
Cflow	Cflo			0	20
Tux	Cflo			4	16
Idct	SoX			17	16
Mytar	SoX	17	17	17	20
Web	SoX	0	0	0	17
Cflow	SoX	14	16	15	14
Tux	SoX	13	13	13	14
TOTAL		188/300	233/300	196/300	256/300

in 12 out of 15 experiments we successfully autotransplanted new functionality

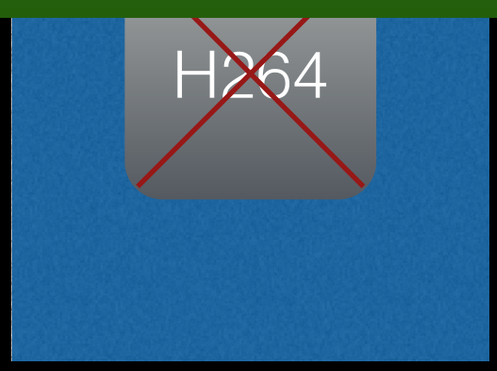
Empirical Study

		Execution Time (minutes)		
Donor	Host	Average	Std. Dev.	Total
Idct	Pidgin	**5**	**7**	**97**
Mytar	Pidgin	**3**	**1**	**65**
Web	Pidgin	**8**	**5**	**160**
Cflow	Pidgin	**58**	**16**	**1151**
Tux	Pidgin	**29**	**10**	**574**
Idct	Cflow	**3**	**5**	**59**
Mytar	Cflow	**3**	**1**	**53**
Web	Cflow	**5**	**2**	**102**
Cflow	Cflow	**44**	**9**	**872**
Tux	Cflow	**31**	**11**	**623**
Idct	SoX	**12**	**17**	**233**
Mytar	SoX	**3**	**1**	**60**
Web	SoX	**7**	**3**	**132**
Cflow	SoX	**89**	**53**	**74**
Tux	SoX	**34**	**13**	**94**
Total		334 (min)	10 (Average)	72 (hours)

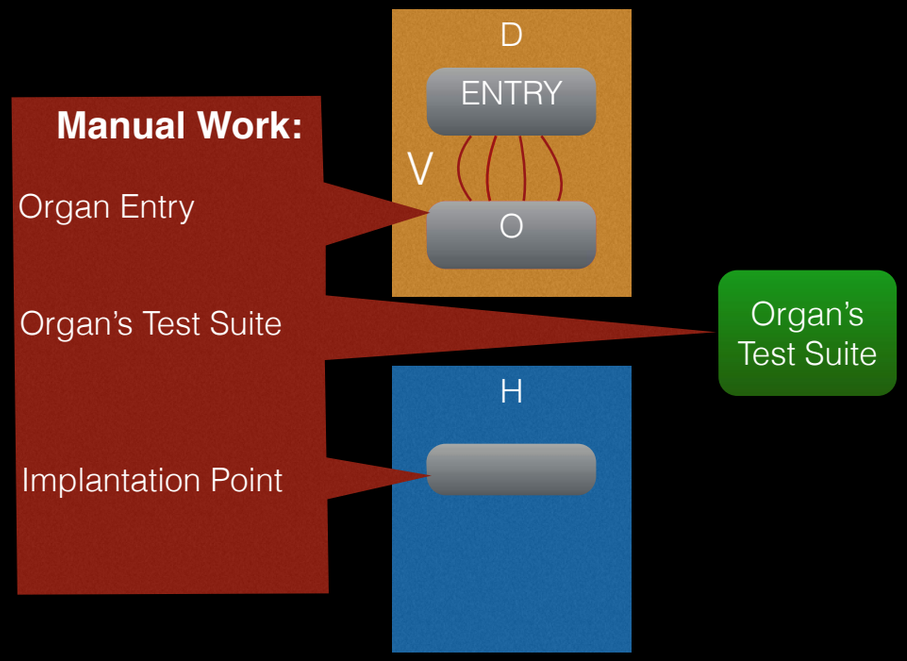
Case Study

		+	Acceptance
H.264		%	100%

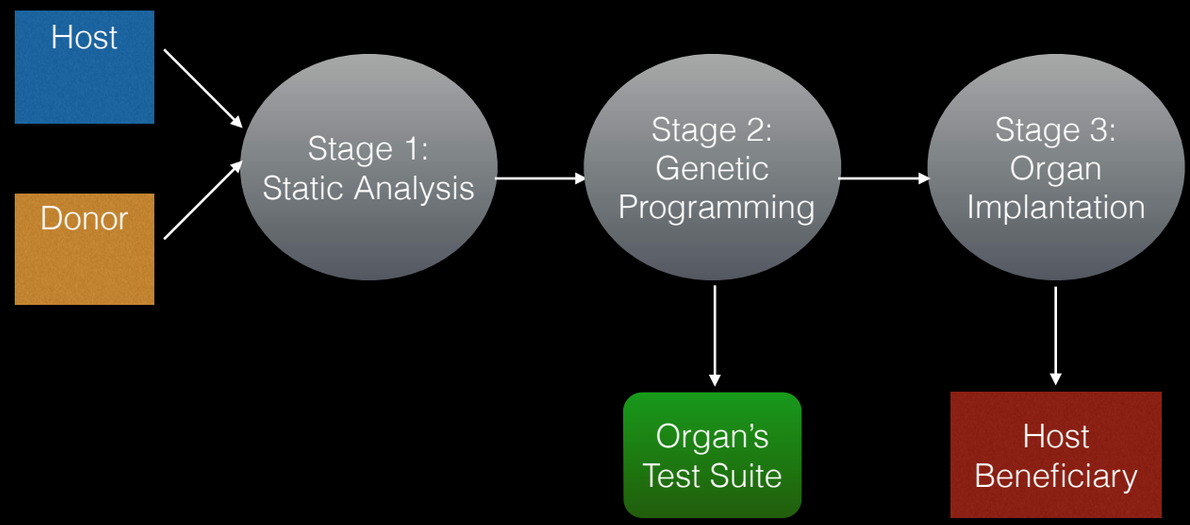
within 26 hours performed a task
that took developers
an avg of 20 days of elapsed time



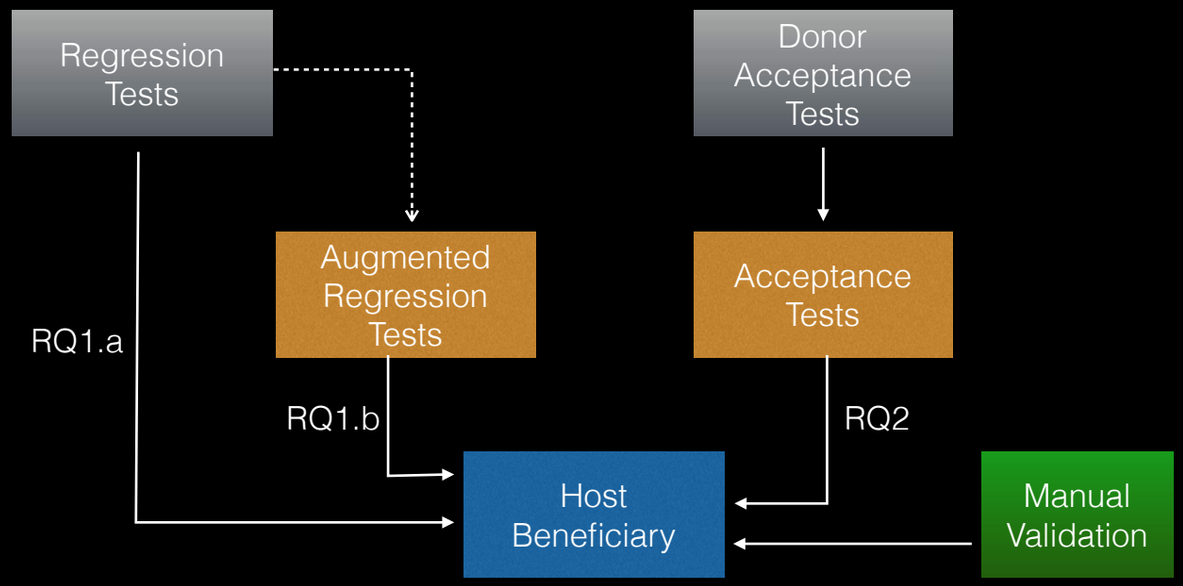
Automated Software Transplantation



μTrans



Validation



Subjects

Subjects	Type	Size KLOC
Idct	Donor	2.3
Mytar	Donor	0.4
Cflow	Donor	25
Webserver	Donor	1.7
TuxCrypt	Donor	2.7
Pidgin	Host	363
Cflow	Host	25
SoX	Host	43
Case Study		
x264	Donor	63
VLC	Host	422

Minimal size: 0.4k

Max size: 422k

Average Donor:16k

Average Host: 213k

Appeared at ISSTA 2015

- Overview
- Preprint
- **MuScalpel**
- Subjects
- Test Suites
- Artifact Evaluation
- SSBSE 2015

muScalpel

Implemented in TXL and C, muScalpel realizes µTrans and comprises 28k SLoCs, of which 16k is TXL, and 12k is C. muScalpel implements a custom version of GP. Unlike conventional GP, which creates an initial population from individuals that contain multiple statements, muScalpel generates an initial population of individuals with just 1 statement, uniformly selected. muScalpel's underlying assumption is that our organs need very few of the statements in their donor. Starting from one LOC gives muScalpel the possibility to find small solutions quickly. muScalpel focuses on evolving the organ's vein. muScalpel also inherits the limitations of TXL, such as its stack limit which precludes parsing large programs and its default C grammar's inability to properly handle preprocessor directives.

As we all know, software is often difficult to build and run, due to dependencies on its development environment and target platform. muScalpel is no exception. Please keep in mind that we built and ran muScalpel only on 64-bit Ubuntu 14.04 LTS machine, with 16 GB RAM, SSD and 8 physical cores, with its TXL v10.6a-64 (14.7.13), gcc-4.8, cflow (GNU cflow) 1.4 installed. Any other configurations may have affect on the results of the replication of our experiments.

This website contains the source for muScalpel, muScalpel in binary form, and the data sets, including test suites, that underlie our experiments. To facilitate replicating our results, we have written a sequence of **scripts** that run a *single* run of each of our experiments. The name of the script identifies the experiment. We have worked hard to make each script bullet-proof and have it thoroughly check your environment for its dependencies and tell you what, if anything, is missing. Despite our best efforts, you may still encounter problems. If that happens, please contact us so we can work with you to resolve them.

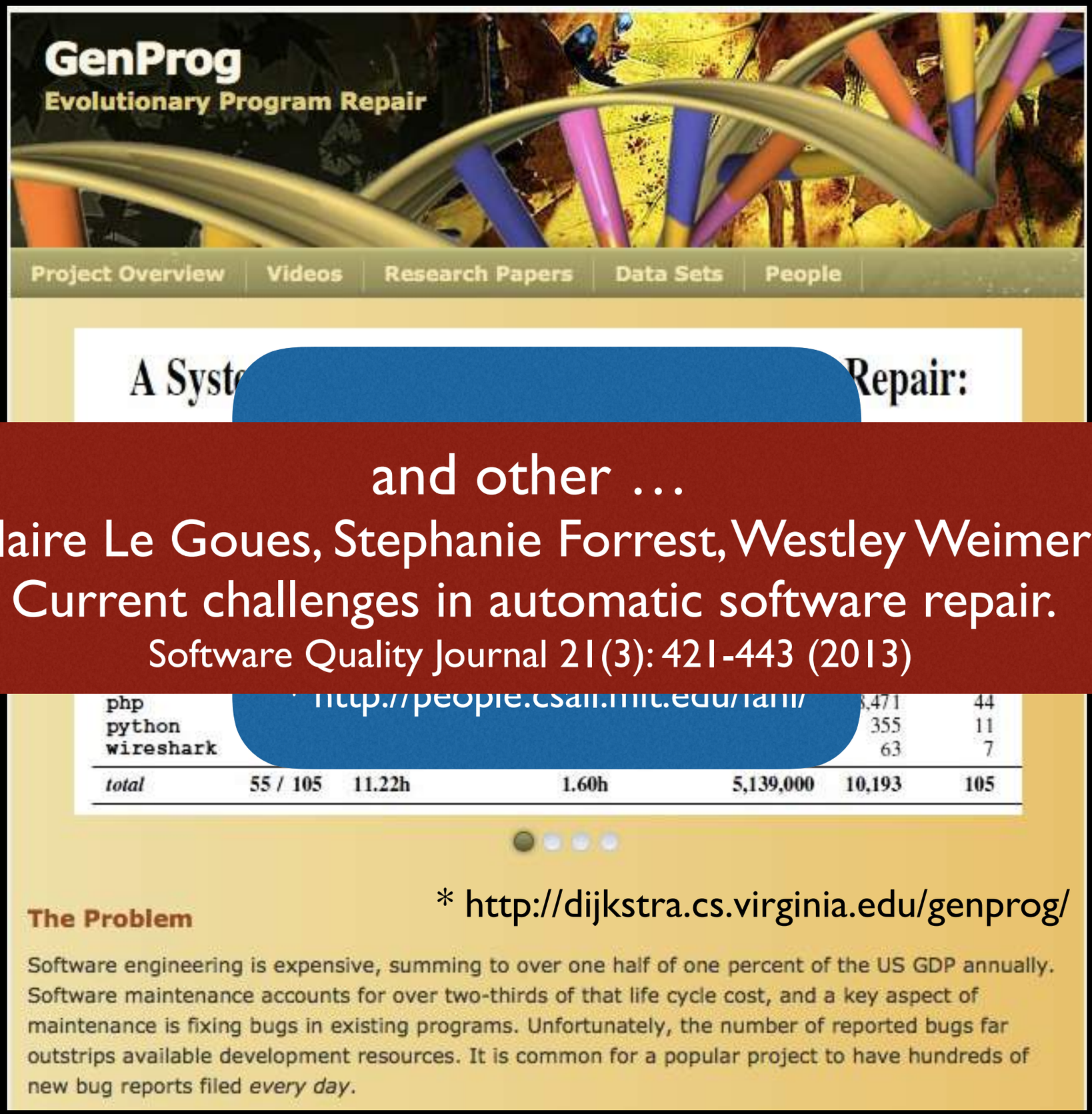
Experiment Scripts

- Link to a script that runs all our experiments, as submitted to ISSTA 2015 artifact evaluation track. Here we also provide a dockerized version of our experiments.
  - **All experiments:** : Download

* http://crest.cs.ucl.ac.uk/autotransplantation/MuScalpel.html

GI Applications

Bug fixing



and other …

Claire Le Goues, Stephanie Forrest, Westley Weimer:
Current challenges in automatic software repair.
Software Quality Journal 21(3): 421-443 (2013)

* http://dijkstra.cs.virginia.edu/genprog/

GI Applications

Bug fixing

Improving energy consumption

Porting old code to new hardware

Grafting new functionality into an existing system

Specialising software for a particular problem class

Other

What if fitness is expensive to compute ?

GI4GI: Improving Genetic Improvement Fitness Functions
Mark Harman & Justyna Petke
(Genetic Improvement Workshop 2015)

GI4GI: Energy Optimisation Example

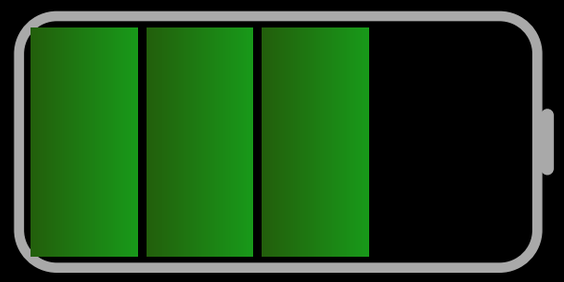
many factors affecting energy consumption, including:

screen behaviour

memory access

device communications

CPU utilisation



GI4GI: Energy Optimisation Example

a hardware-dependent linear energy model for GI:

$$power = C_{const} + C_{ins}\frac{ins}{cycle} + C_{flops}\frac{flops}{cycle} + C_{tca}\frac{tca}{cycle} + C_{mem}\frac{mem}{cycle}$$

$$energy = seconds \times power$$

Coefficient	Description	Intel (4-core)	AMD (48-core)
C_{const}	constant power draw	31.530	394.74
C_{ins}	instructions	20.490	-83.68
C_{flops}	floating point ops.	9.838	60.23
C_{tca}	cache accesses	-4.102	-16.38
C_{mem}	cache misses	2962.678	-4209.09

Table 2. Power model coefficients.

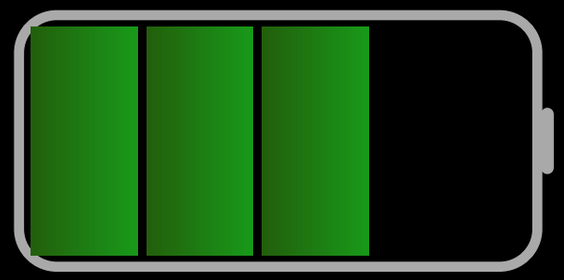
Post-compiler software optimization for reducing energy (ASPLOS'14) Schulte et al.

GI4GI: Energy Optimisation Example

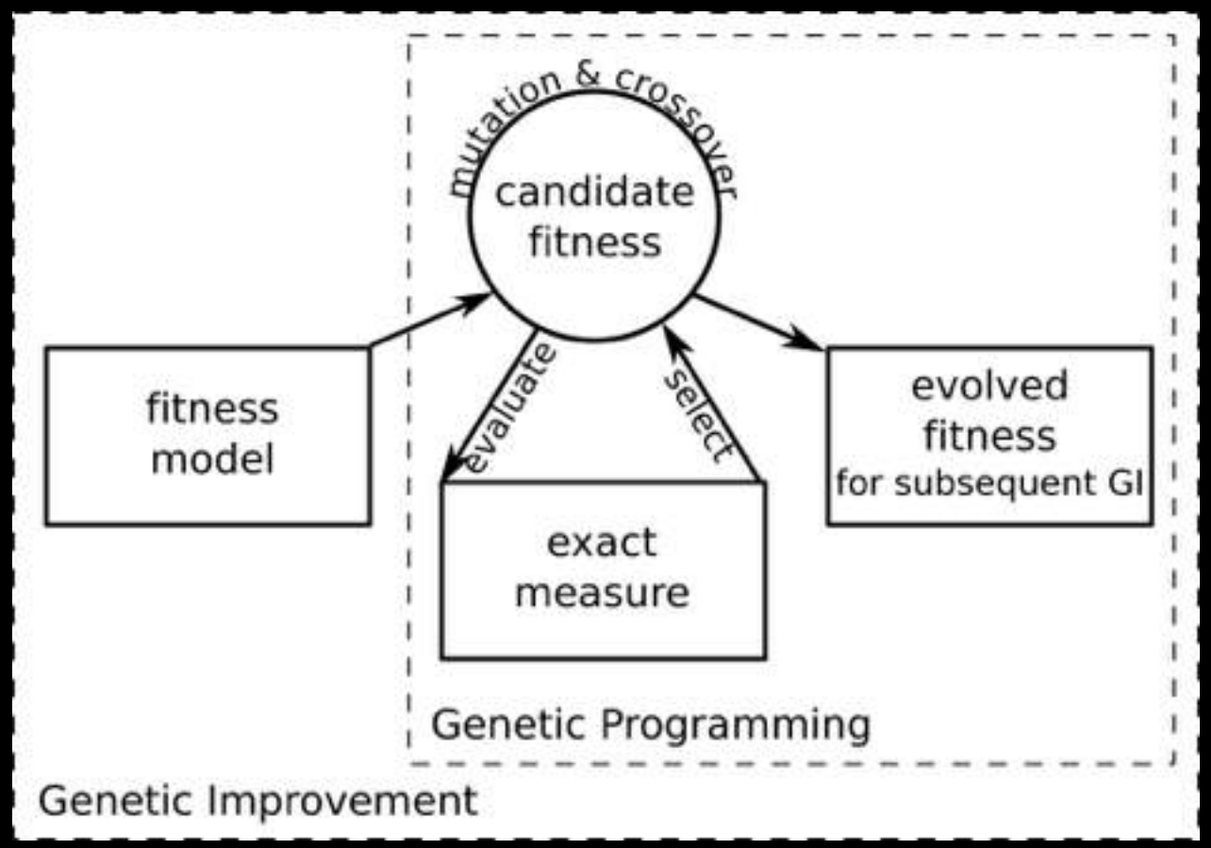
Idea:

Use GI to evolve a fitness function f for energy consumption.

Use f to improve energy consumption of software.



GI4GI


mutation & crossover
candidate
fitness
fitness
model
evaluate
select
evolved
fitness
for subsequent GI
exact
measure
Genetic Programming
Genetic Improvement

GI Growth

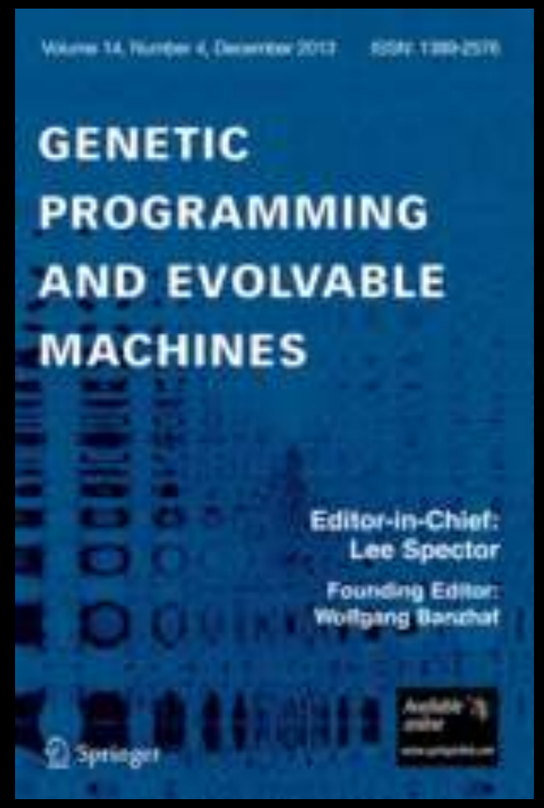
1st International Genetic Improvement Workshop

at GECCO 2015, Madrid, Spain

www.geneticimprovementofsoftware.com

GI Growth

Special Session on GI *http://www.wcci2016.org/



Special Issue on GI

Conclusions

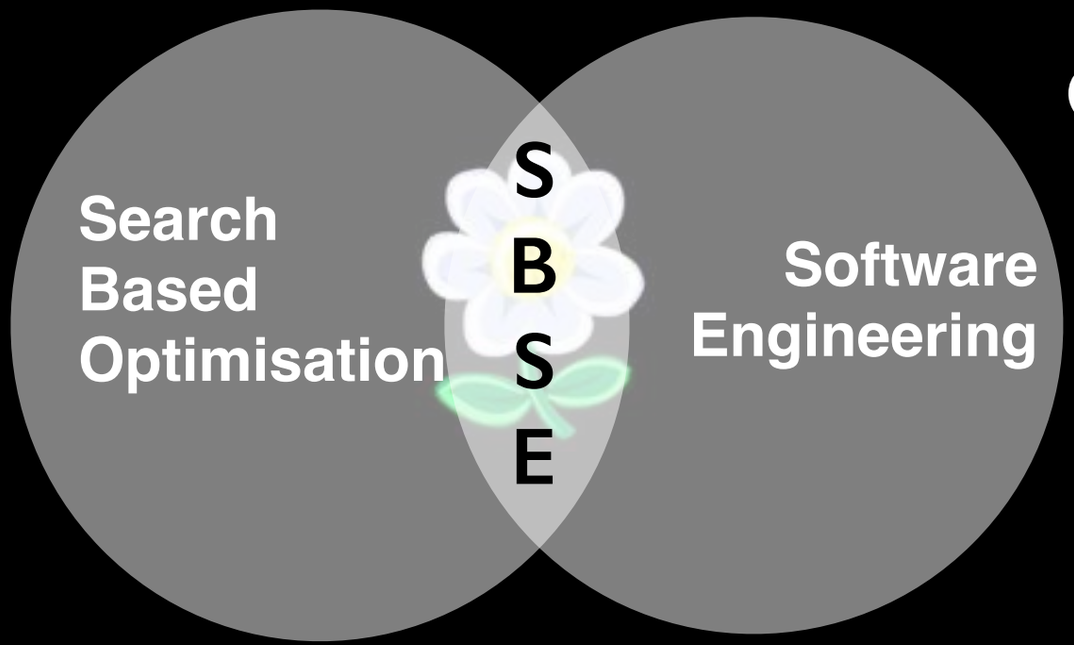
Functional Requirements

humans have to define these

Non-Functional Requirements

we can optimise these

Summary



Combinatorial Interaction Testing

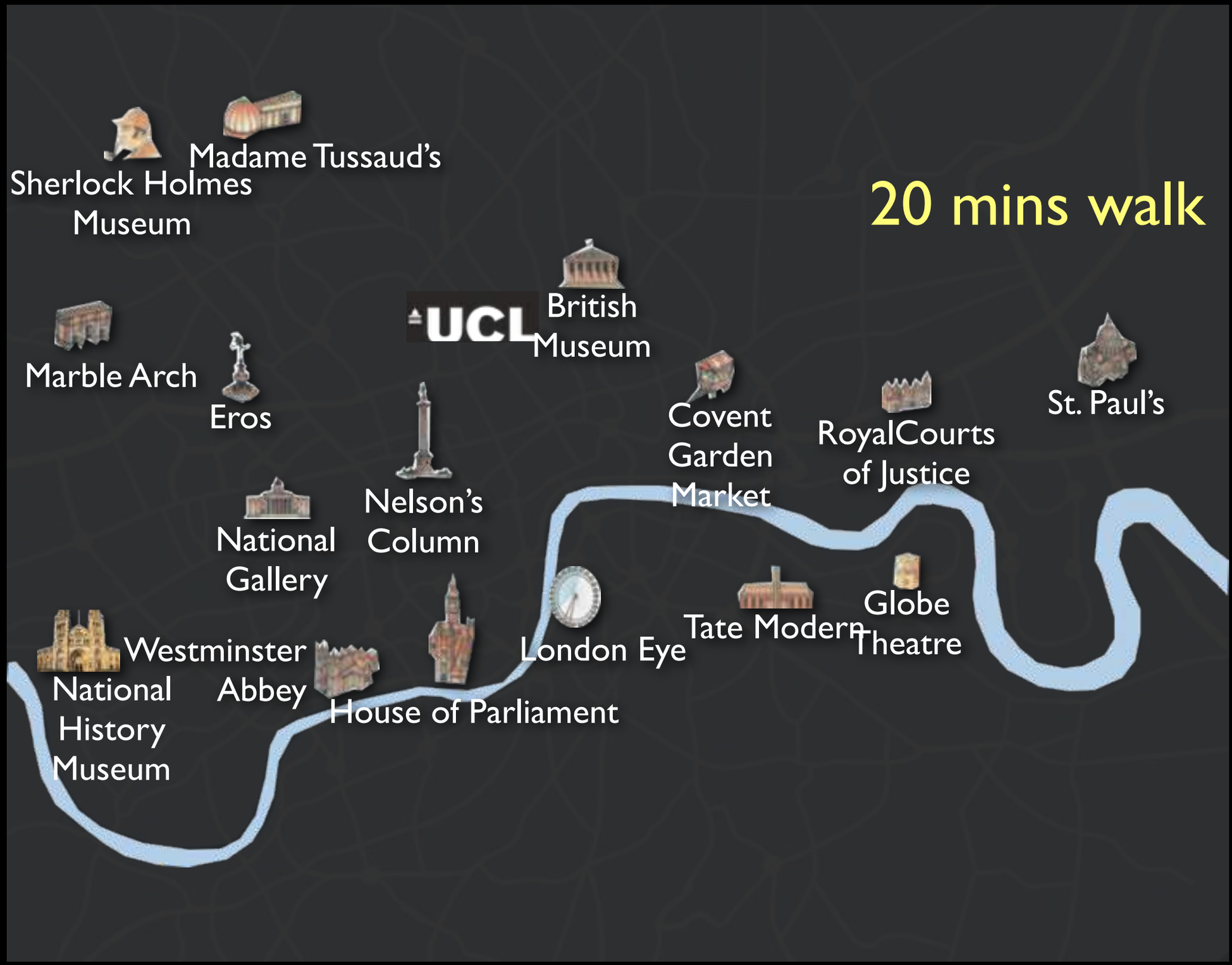
Genetic Improvement

Research Opportunities

Contact me if you want to visit CREST:

j.petke at ucl.ac.uk

Centre for Research on Evolution, Search and Testing
University College London


Sherlock Holmes Museum
Madame Tussaud's
20 mins walk
Marble Arch
Eros
UCL
British Museum
Covent Garden Market
RoyalCourts of Justice
St. Paul's
National Gallery
Nelson's Column
Tate Modern
Globe Theatre
London Eye
Westminster Abbey
National History Museum
House of Parliament

COWs

CREST Open Workshop

Roughly one per month

Discussion based

Recorded and archived

http://crest.cs.ucl.ac.uk/cow/

COWs

http://crest.cs.ucl.ac.uk/cow/

COWs

#Total Registrations 1512
#Unique Attendees 667
#Unique Institutions 244
#Countries 43
#Talks 421

(Last updated on November 4, 2015)

http://crest.cs.ucl.ac.uk/cow/

CREST Open Workshop (COW)

The 41st CREST Open Workshop

Software Engineering And Computer Science Using Information

Date: 27 -28 April 2015

Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 40th CREST Open Workshop

SSBSE 2015 Challenge: Collaborative Jam Session

Date: 30 - 31 March 2015

Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 39th CREST Open Workshop

Measuring, Testing and Optimising Computational Energy Consumption

Date: 23 - 24 February 2015

Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 38th CREST Open Workshop

Working Tutorial on Statistical Methods in Experimental Software Engineering

Date: 26 - 27 January 2015

Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 37th CREST Open Workshop

Working Tutorial on Empirical Software Engineering Methods

Date: 24 - 25 November 2014

Venue: Engineering Front Executive Suite, Roberts Building, UCL

The 36th CREST Open Workshop

App Store Analysis

Date: 27 - 28 October 2014

45th COW

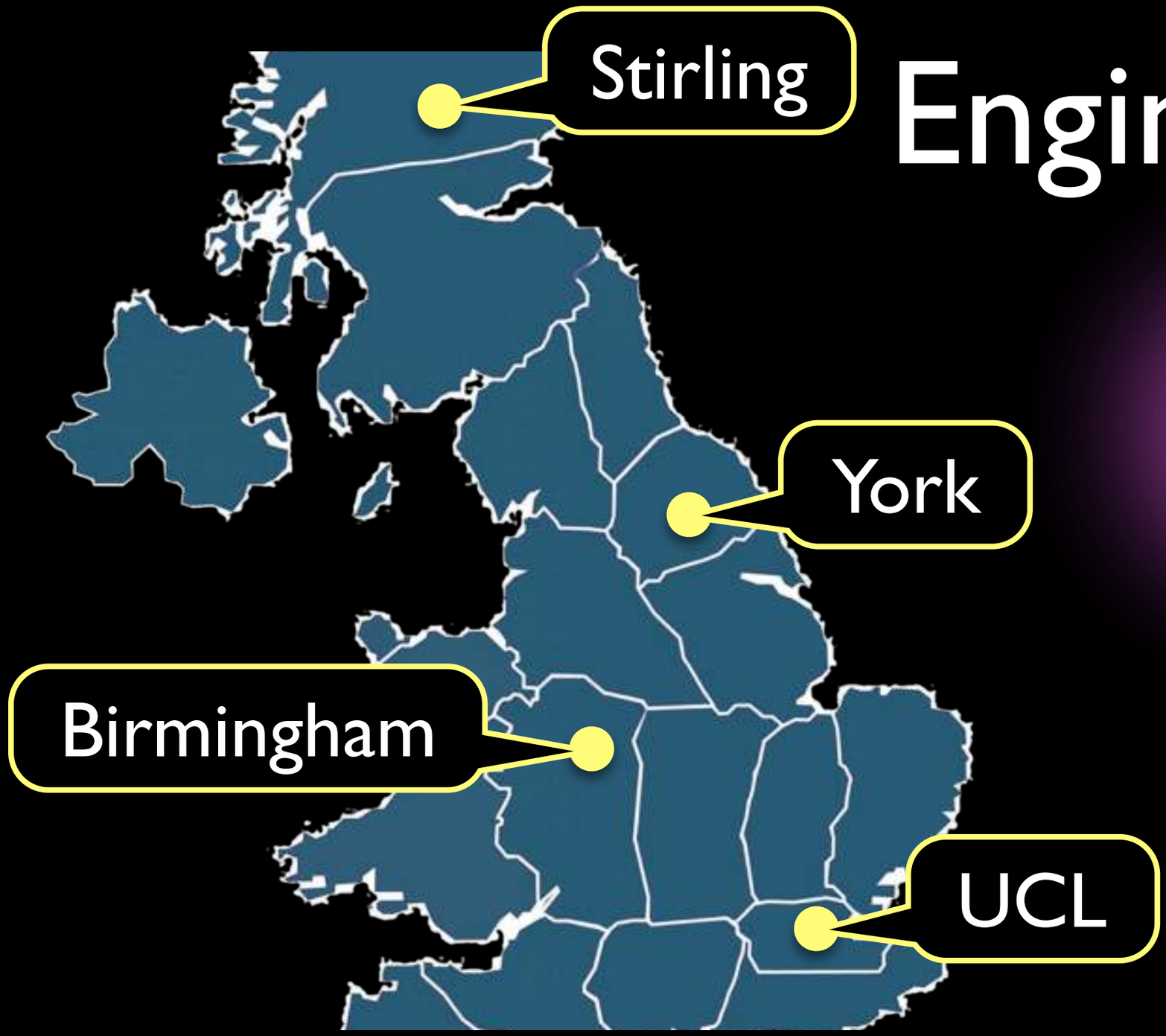
Genetic Improvement

25-26 January 2016

http://crest.cs.ucl.ac.uk/cow/

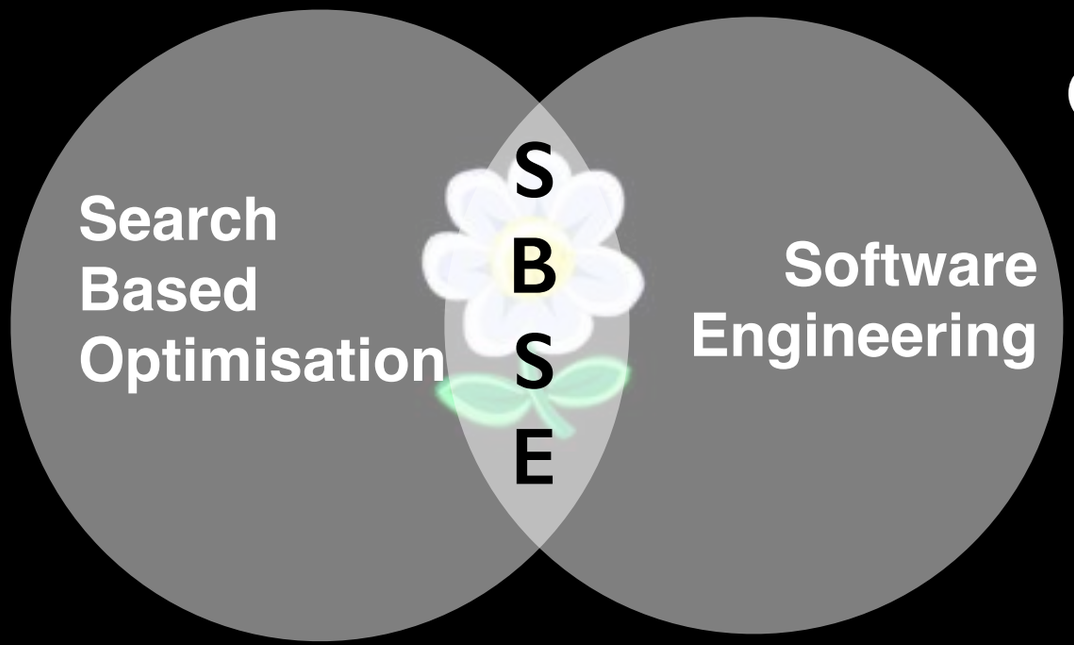
EPSRC Grant

Dynamic Adaptive Search Based Software Engineering



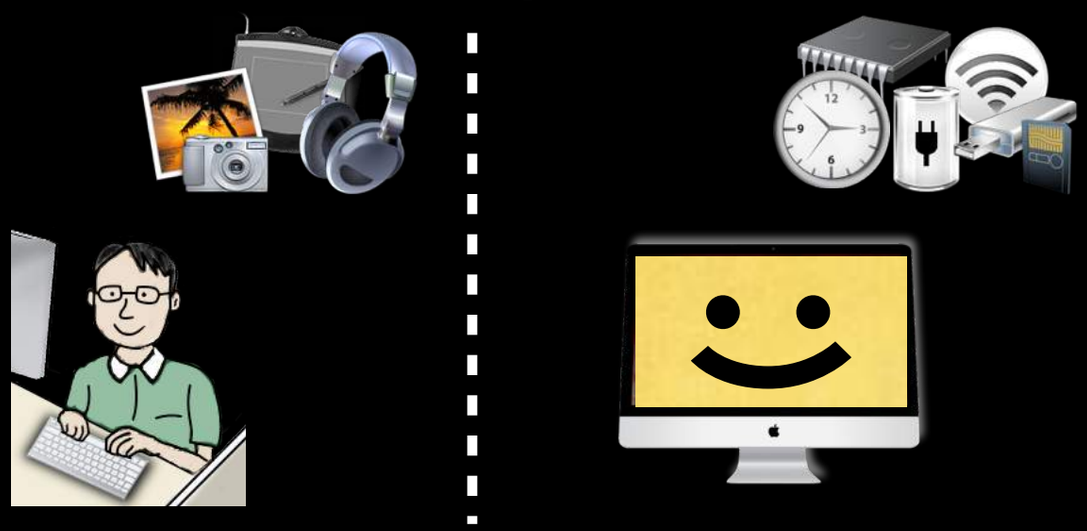
DAASE

Summary



Combinatorial Interaction Testing

Genetic Improvement



COWs
Visitor Scheme
Open positions

Pictures used with thanks from these sources

Stonehenge: By Yuanyuan Zhang [All right reserved] via Flickr

Pickering's Harem: [Public domain], via Wikimedia Commons

IBM 026 Card Punch: By Ben Franske (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0-2.5-2.0-1.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

BBC_Micro: [Public domain], via Wikimedia Commons

IBM PC: By Boffy B (Own work) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0-2.5-2.0-1.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

IMac: By Matthieu Riegler, Wikimedia Commons [CC-BY-3.0 (http://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons

Ada Lovelace: By Alfred Edward Chalon [Public domain], via Wikimedia Commons

Programmer: undesarchiv, B 145 Bild-F031434-0006 / Gathmann, Jens / CC-BY-SA [CC-BY-SA-3.0-de (http://creativecommons.org/licenses/by-sa/3.0/de/deed.en)], via Wikimedia Commons

Bath Abbey: By Yuanyuan Zhang [All right reserved] via Flickr