Automated Software Transplantation

QRS 2016

Talk by Mark Harman

PhD work by Alexandru Marginean

Collaborators

Earl Barr, Yue Jia, Justyna Petke

CREST, University College London

UCL

Automated Software Transplantation

QRS 2016

Talk by Mark Harman

PhD work by Alexandru Marginean

Collaborators

Earl Barr, Yue Jia, Justyna Petke

CREST, University College London

UCL

Why A ansplantation?

~100 players

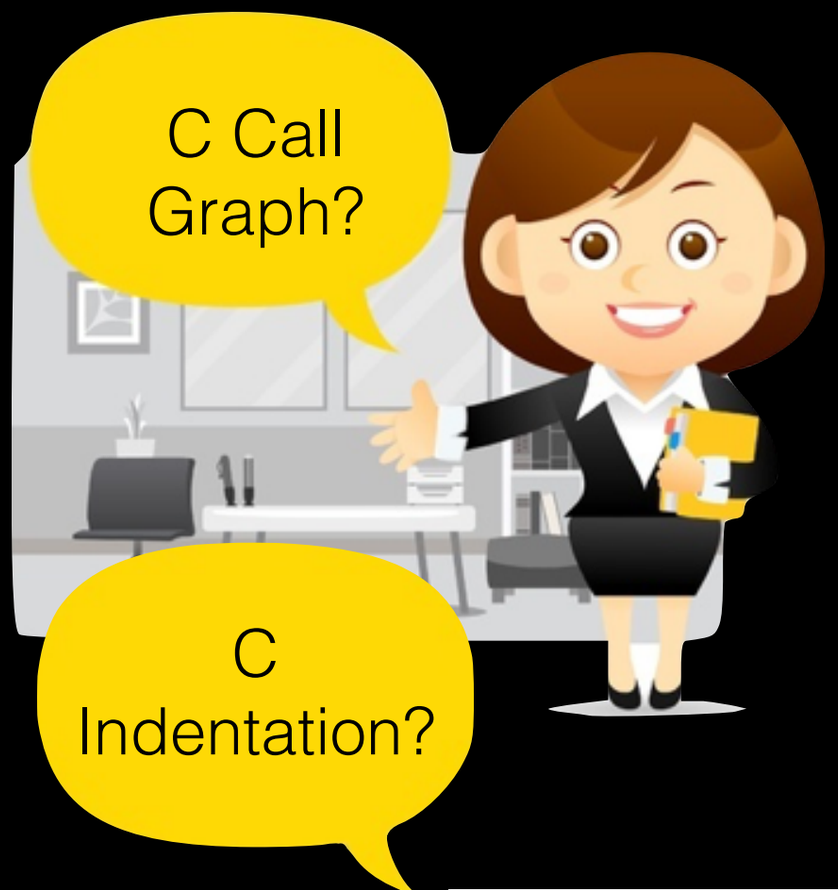
Video Player

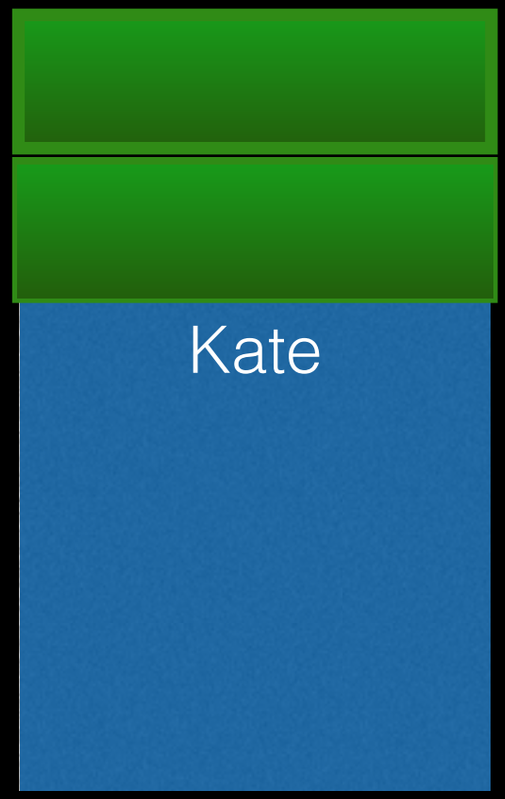
Check open source repositories

Start from scratch

UCL

Why Autotransplantation?





UCL

Recognition for Our Tool μScalpel

BBC World Service Interview
WIRED article
Many shares on Social Media
ACM Distinguished Paper Award at ISSTA '15

Automated Transplantation of Call Graph and Layout Features into Kate

Alexandru Marginean, Earl T. Barr, Mark Harman, Yue Jia

UCL, Department of Computer Science, CREST Centre

Abstract. We report the automated transplantation of two features currently missing from Kate: call graph generation and automatic layout for C programs, which have been requested by users on the Kate development forum. Our approach uses a lightweight annotation system with Search Based techniques augmented by static analysis for automated transplanting. The results are promising: on average, our tool requires 101 minutes of standard desktop machine time to transplant the call graph feature, and 31 minutes to transplant the layout feature. We repeated each experiment 20 times and validated the resulting transplants using unit, regression and acceptance test suites. In 34 of 40 experiments conducted our search-based autotransplantation tool, μSCALPEL, was able to successfully transplant the new functionality, passing all tests.

Automated Software Transplantation

Earl T. Barr Mark Harman Yue Jia Alexandru Marginean Justyna Petke

CREST, University College London, Malet Place, London, WC1E 6BT, UK
{e.barr,m.harman,yue.jia,alexandru.marginean.13,j.petke}@ucl.ac.uk

ABSTRACT

Automated transplantation would open many exciting avenues for software development: suppose we could autotransplant code from one system into another, entirely unrelated, system. This paper introduces a theory, an algorithm, and a tool that achieve this. Leveraging lightweight annotation, program analysis identifies an organ (interesting behavior to transplant); testing validates that the organ exhibits the desired behavior during its extraction and after its implantation into a host. While we do not claim automated transplantation is now a solved problem, our results are encouraging: we report that in 12 of 15 experiments, involving 5 donors and 3 hosts (all popular real-world systems), we successfully autotransplanted new functionality and passed all regression tests. Autotransplantation is also already useful: in 26 hours computation time we successfully autotransplanted the H.264 video encoding functionality from the x264 system to the VLC media player; compare this to upgrading x264 within VLC, a task that we estimate, from VLC's version history, took human programmers an average of 20 days of elapsed, as opposed to dedicated, time.

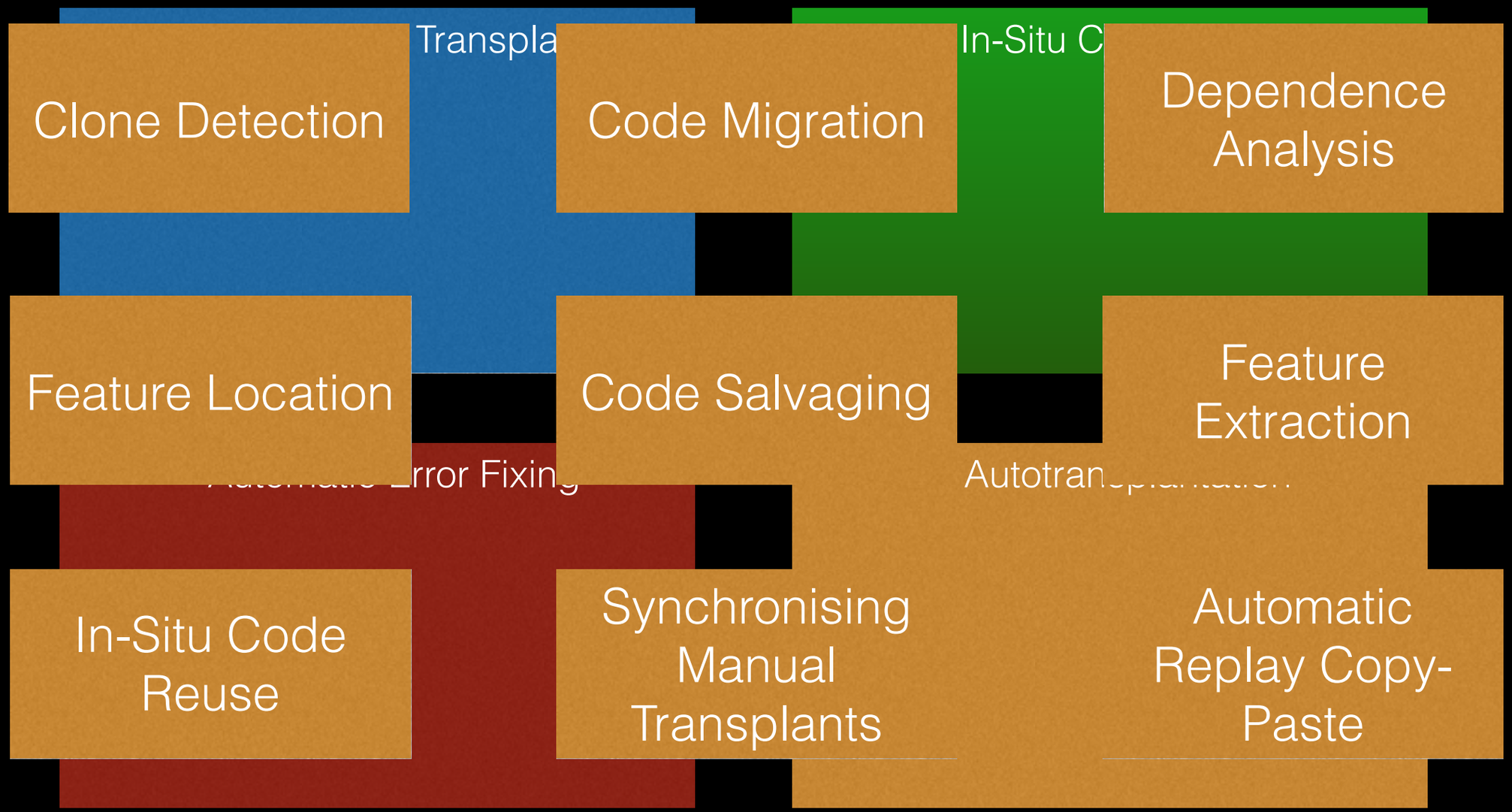
dependence analysis [3, 22, 29, 30] and feature extraction techniques [24, 33, 44]. However, the overall process remains largely unautomated, particularly the critical transplantation of code that implements useful functionality from a *donor* into a target system, which we call the *host*.

What if we could automate the process of extracting functionality from one system and transplanting it into another? This is the goal we set ourselves in this paper. That is, we are the first to develop and evaluate techniques to implement automated software transplantation from one system to another, which one of the authors proposed (hitherto unimplemented and unevaluated) in the keynote of the 2013 WCRE [28].

A programmer must first identify the entry point of code that implements a feature of interest. Given an entry point in the donor and a target implantation point in the host program, the goal of automated transplantation is to identify and extract an organ, all code associated with the feature of interest, then transform it to be compatible with the name space and context of its target site in the host. The programmer also supplies test suites that guide the search for donor code modifications required to make it fully executable (and pass all test cases) when deployed in the host.

UCL

Related Work

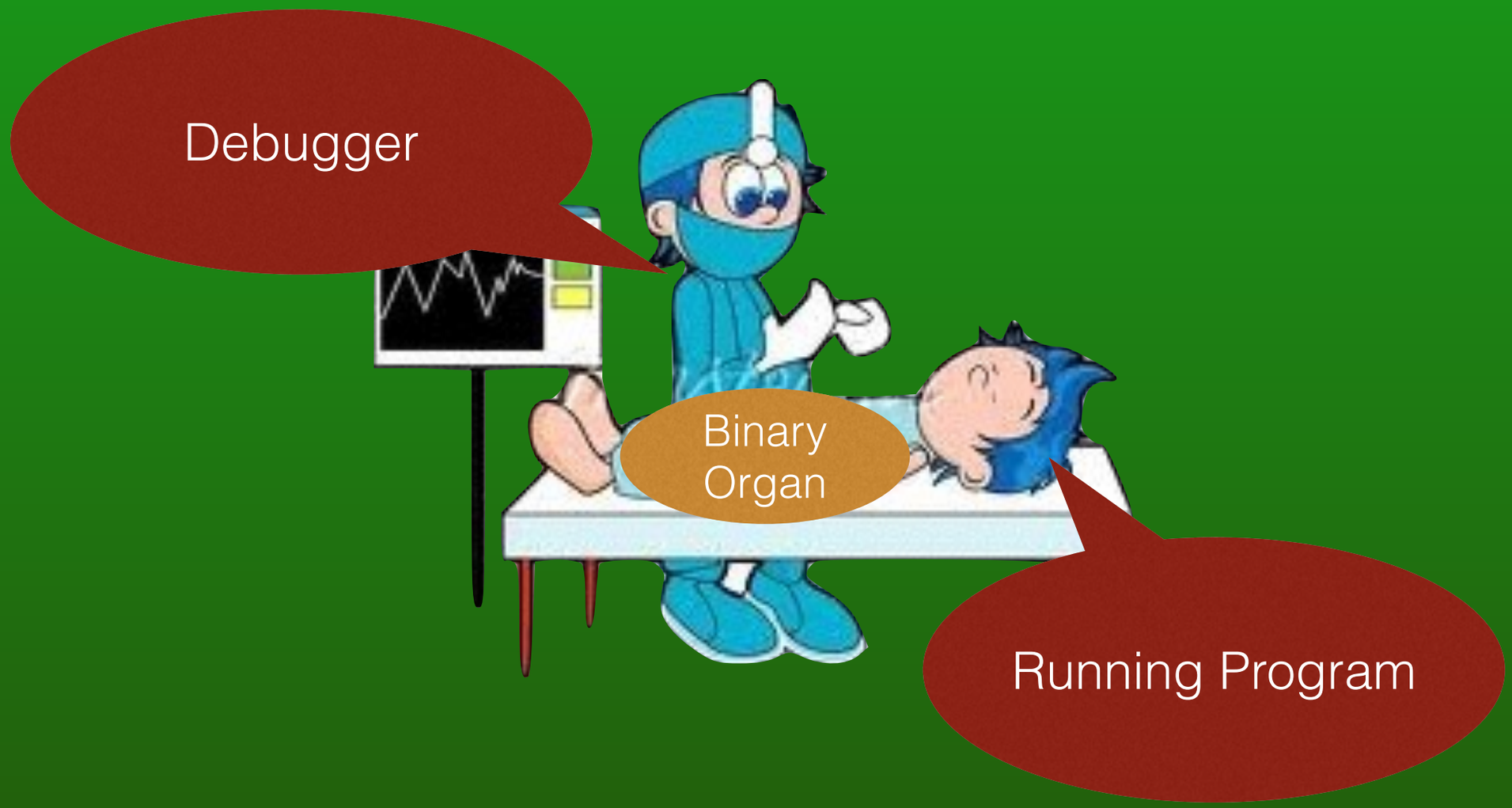


UCL

Related Work

Miles *et al.:* In situ reuse of logically extracted functional components

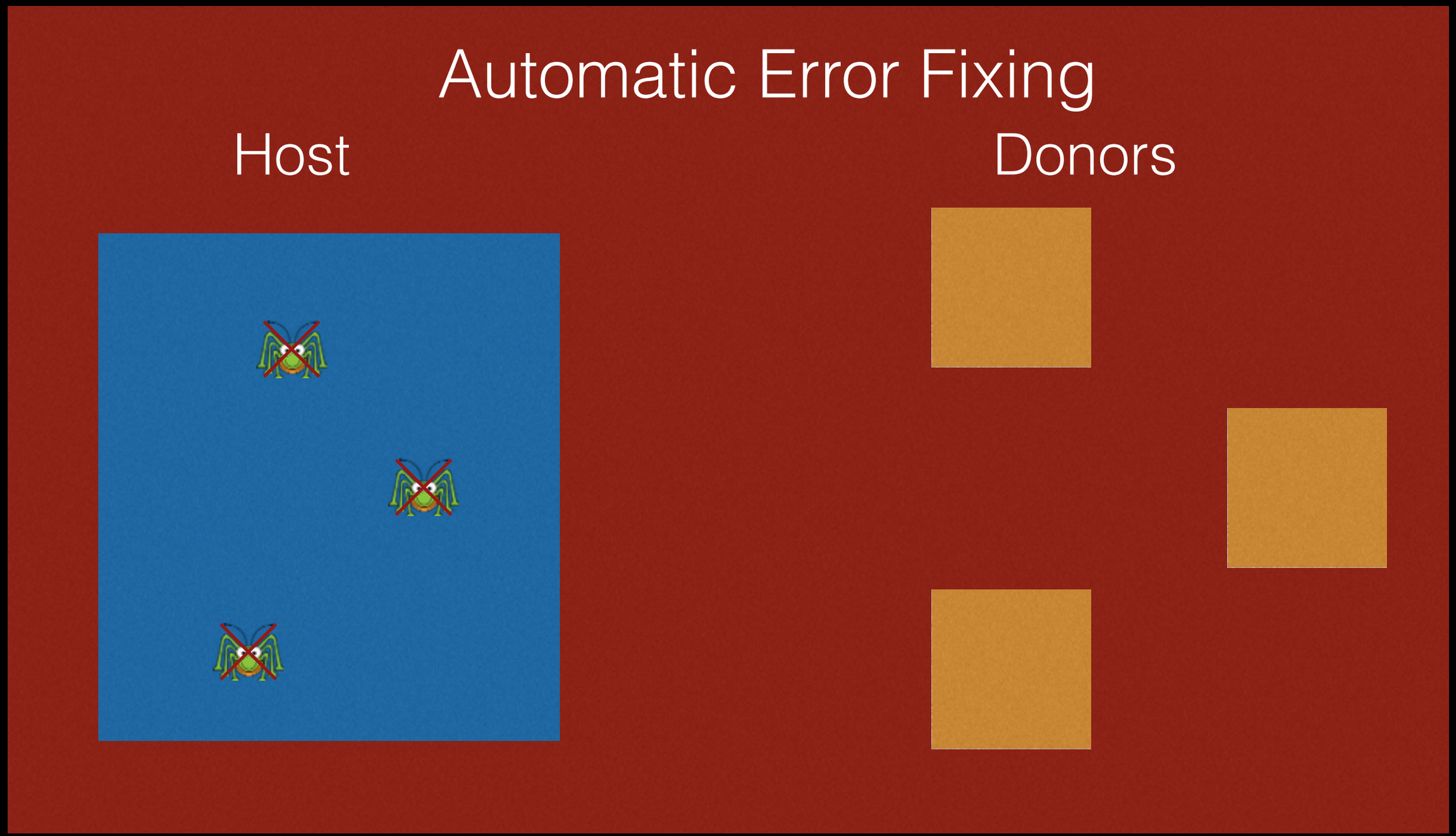
In-Situ Code Reusal



UCL

Related Work

Sidiroglou-Douskos *et al.:* Automatic Error Elimination by Multi-Application Code Transfer



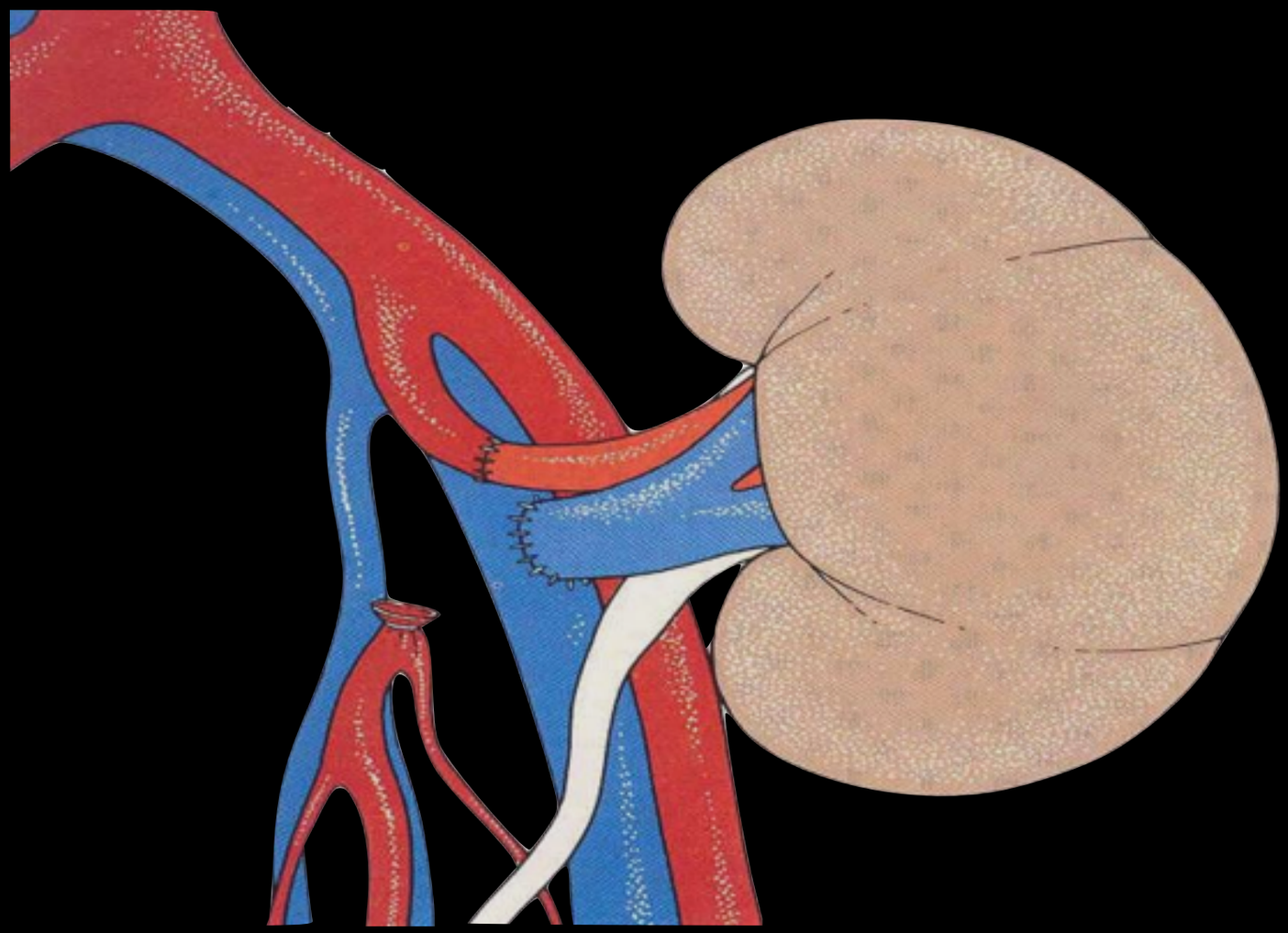
UCL

Related Work

Autotransplantation

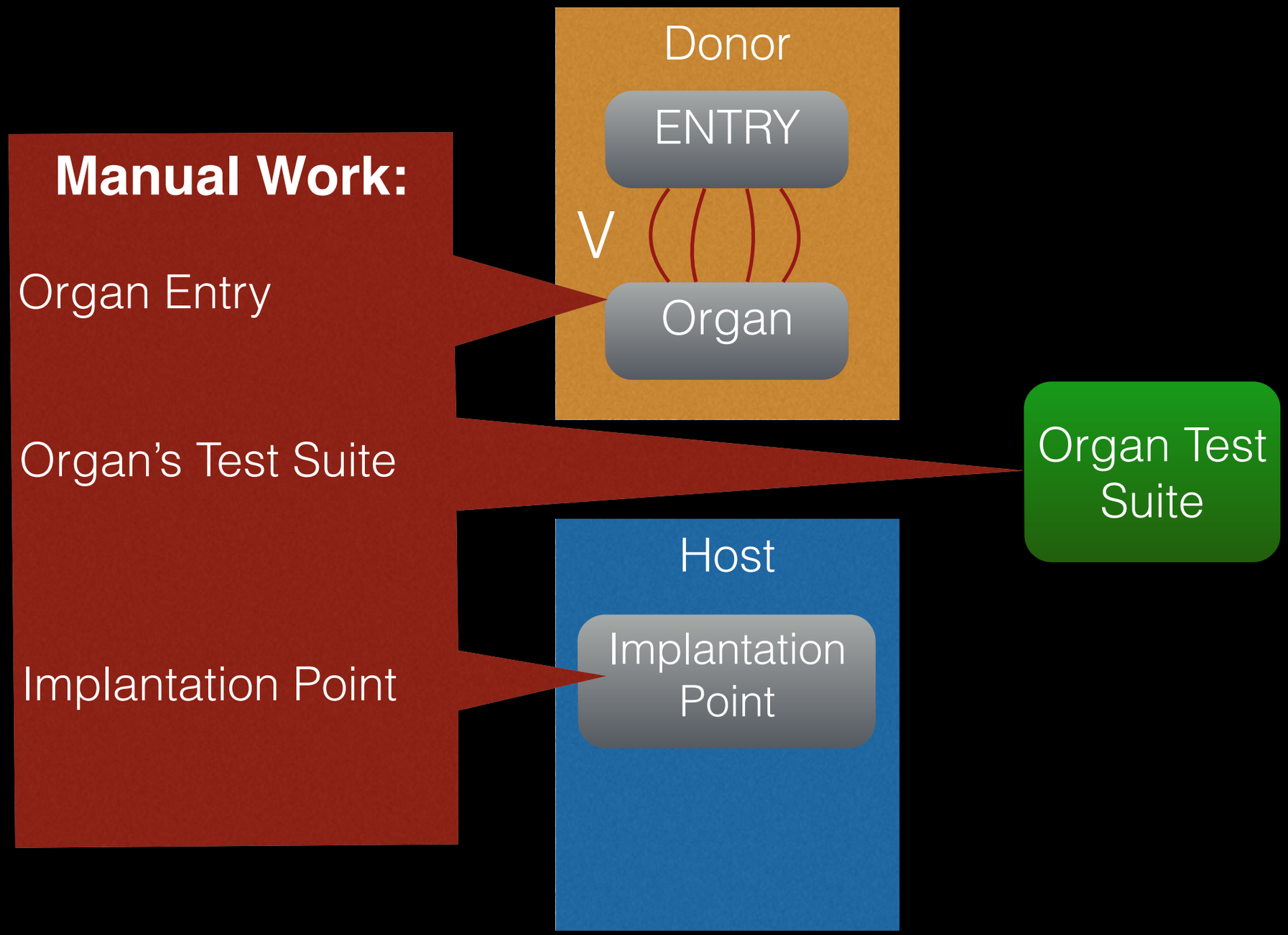
UCL

Human Organ Transplantation



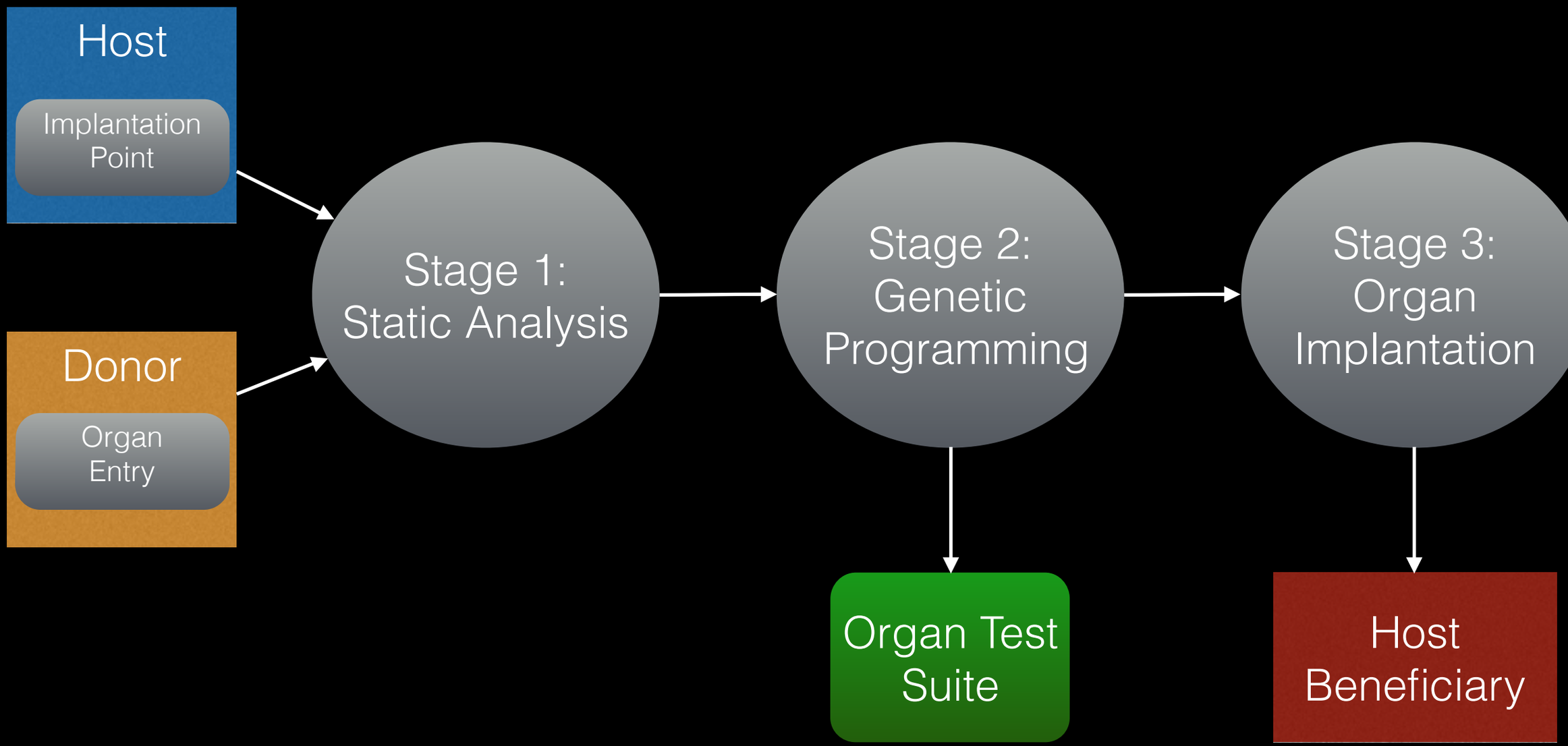
UCL

Automated Software Transplantation



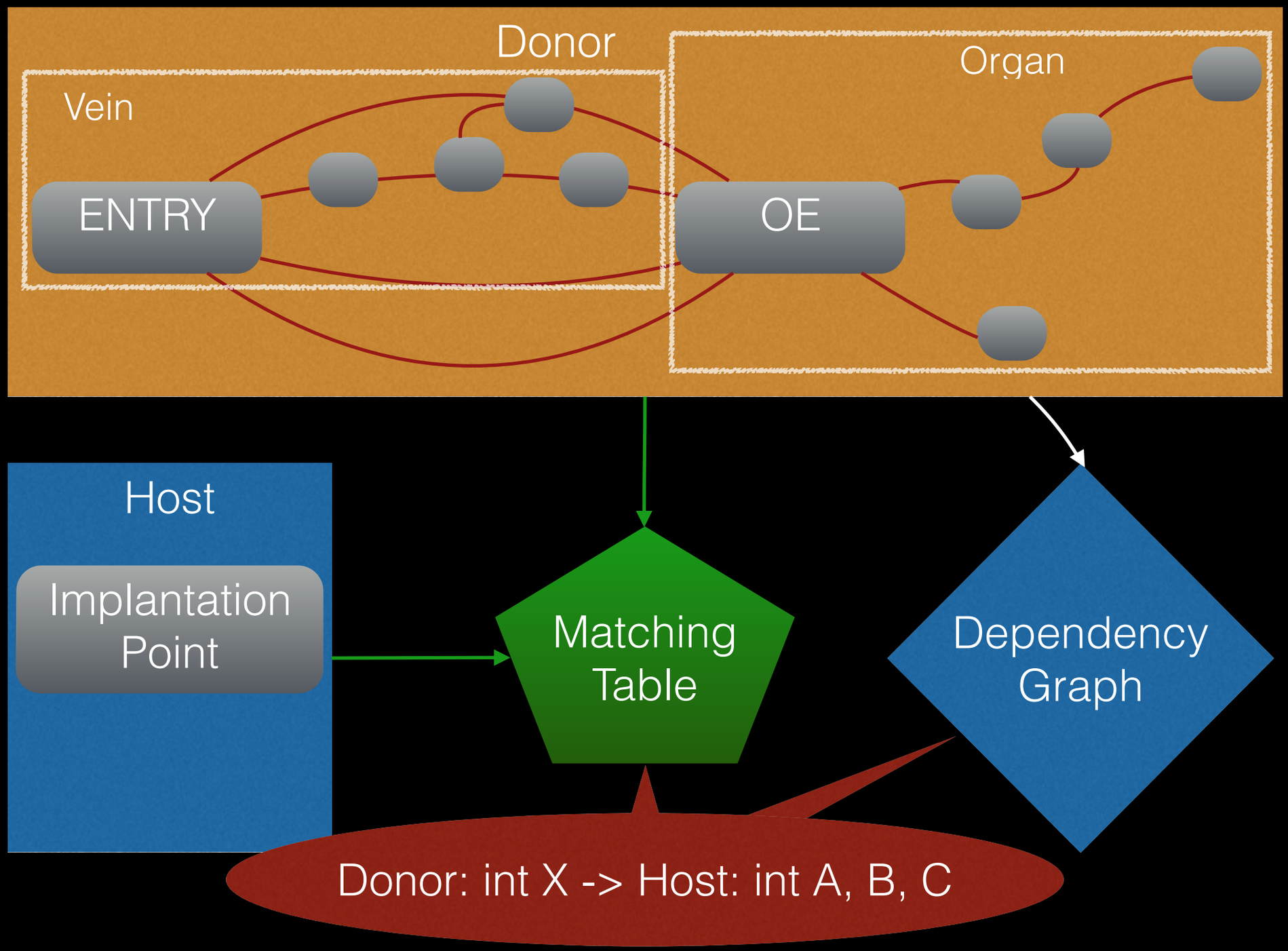
UCL

μTrans


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

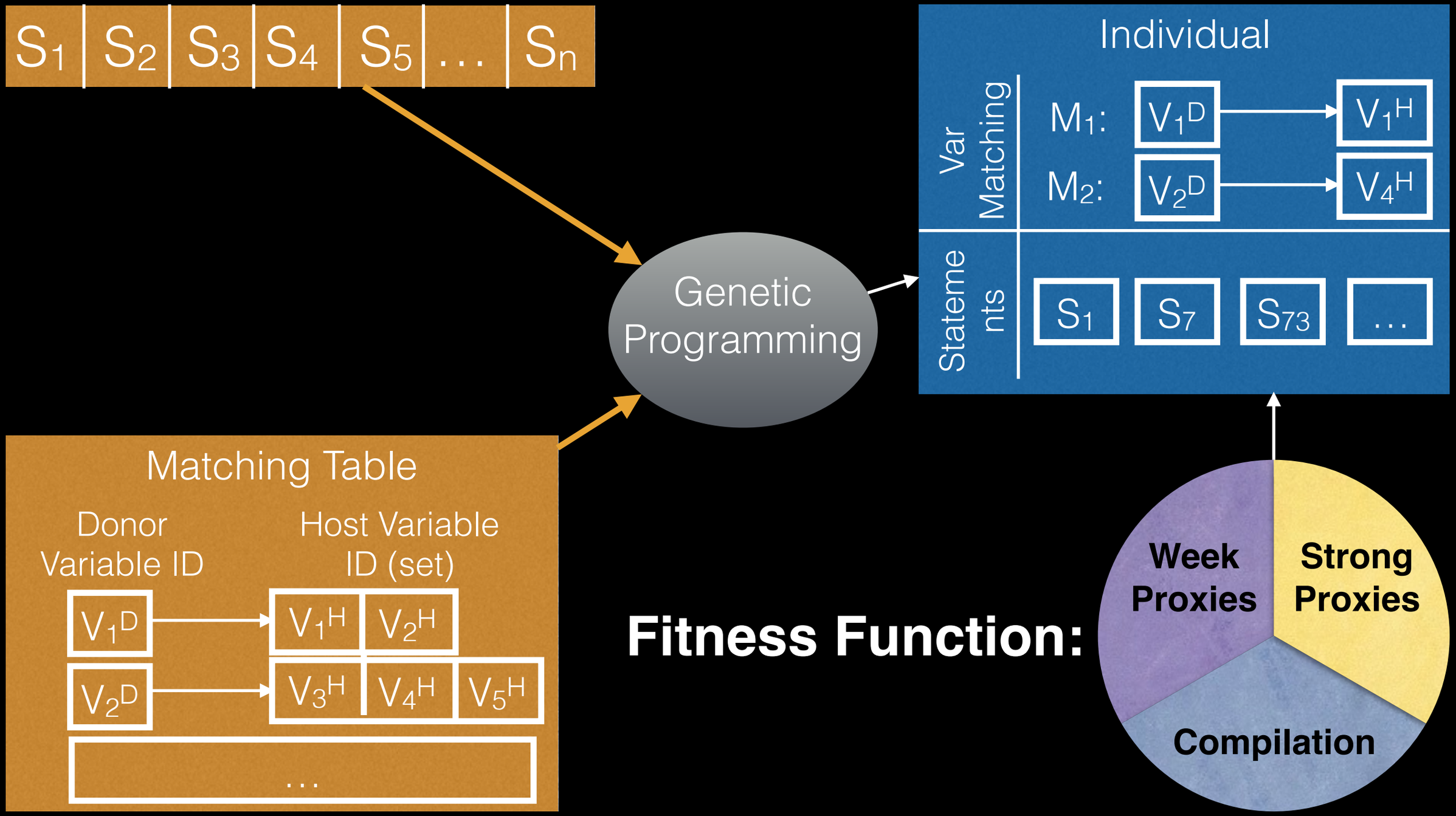
UCL

Stage 1 — Static Analysis



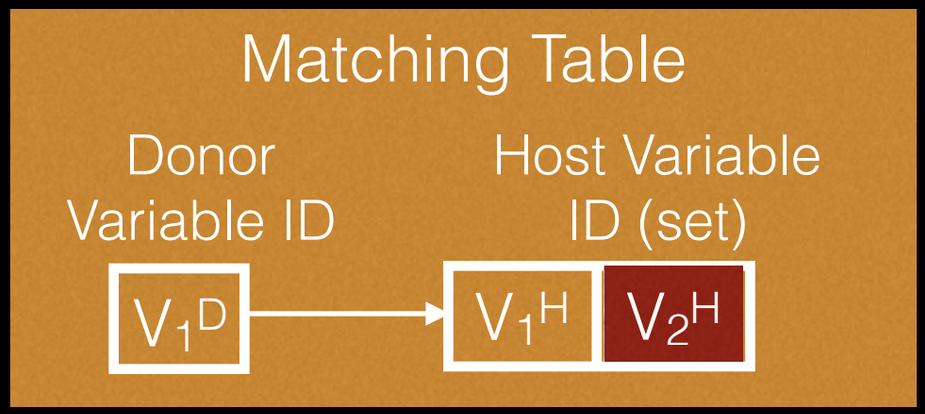
UCL

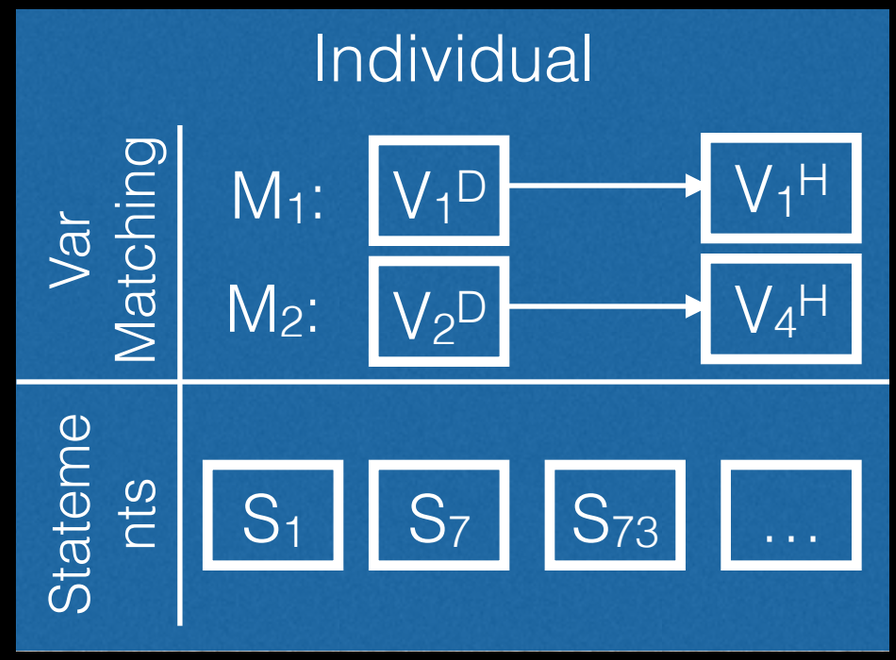
Stage 2 — GP



UCL

Stage 2 - Gp Operators

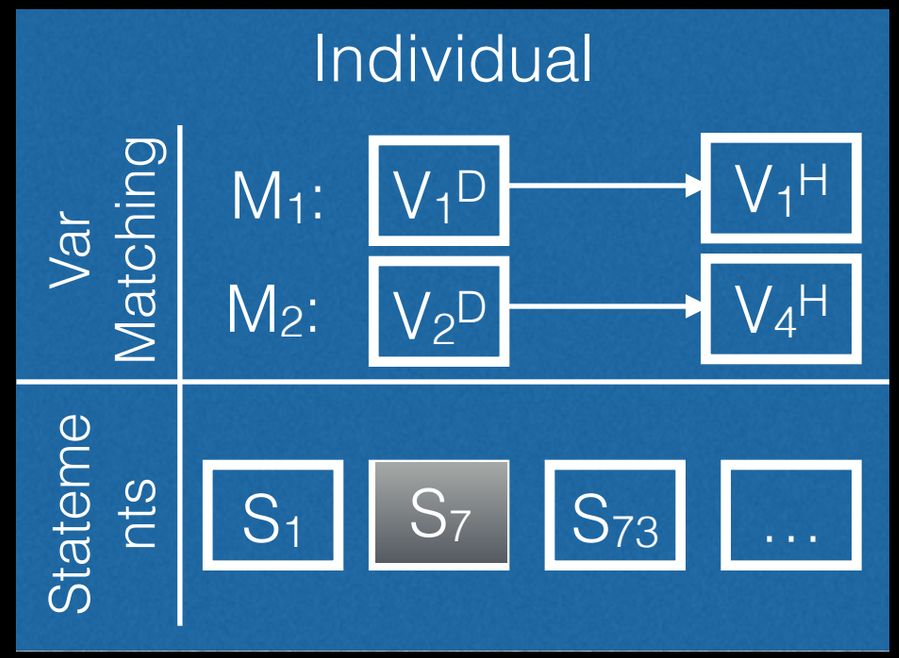




UCL

Stage 2 - Gp Operators

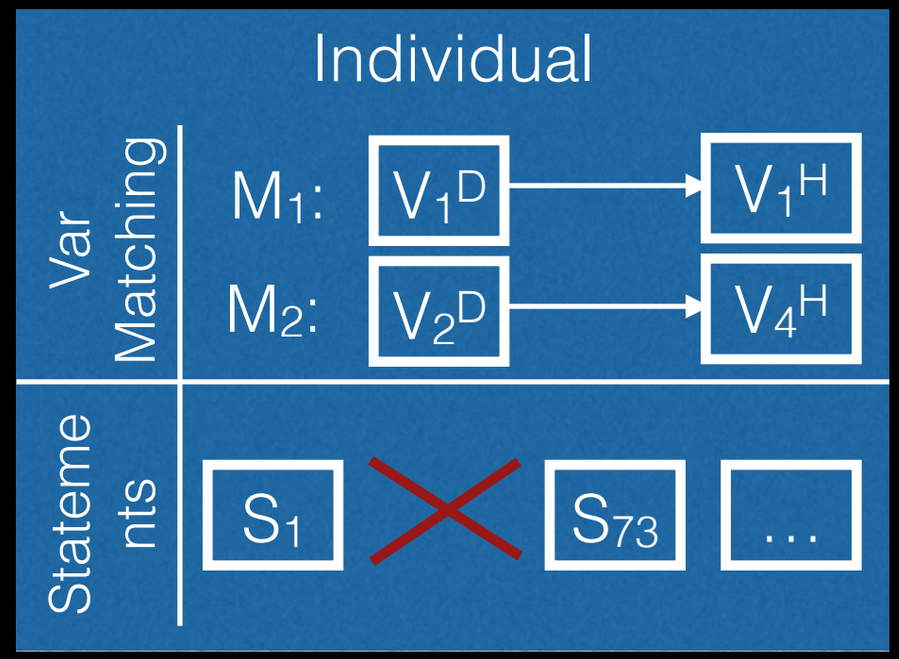
Replace Statement



UCL

Stage 2 - Gp Operators

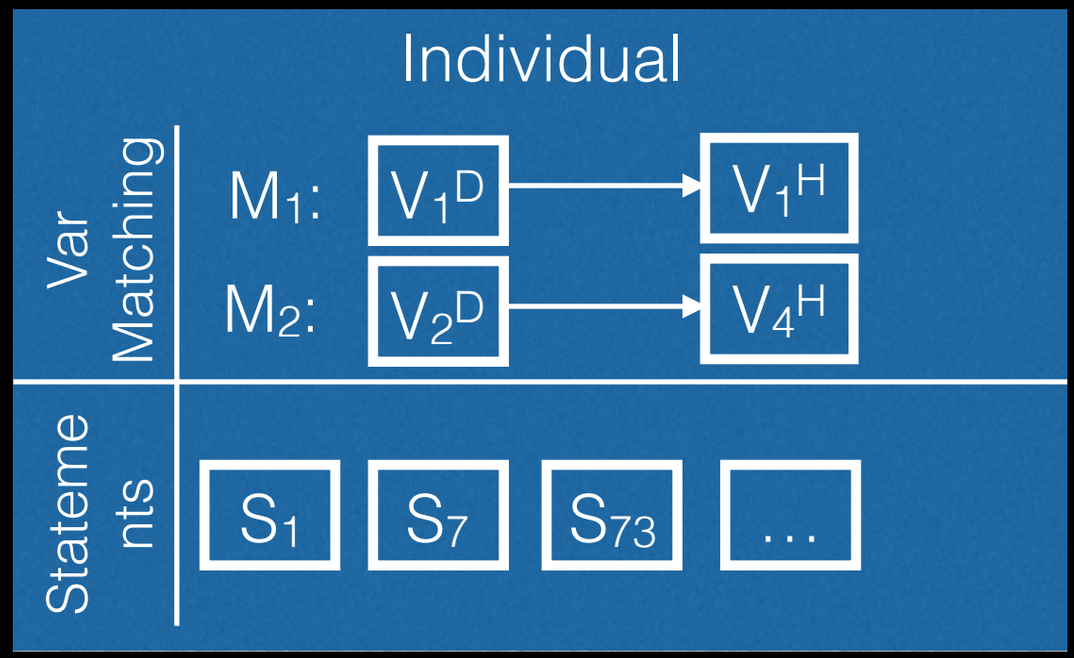
Remove Statement



UCL

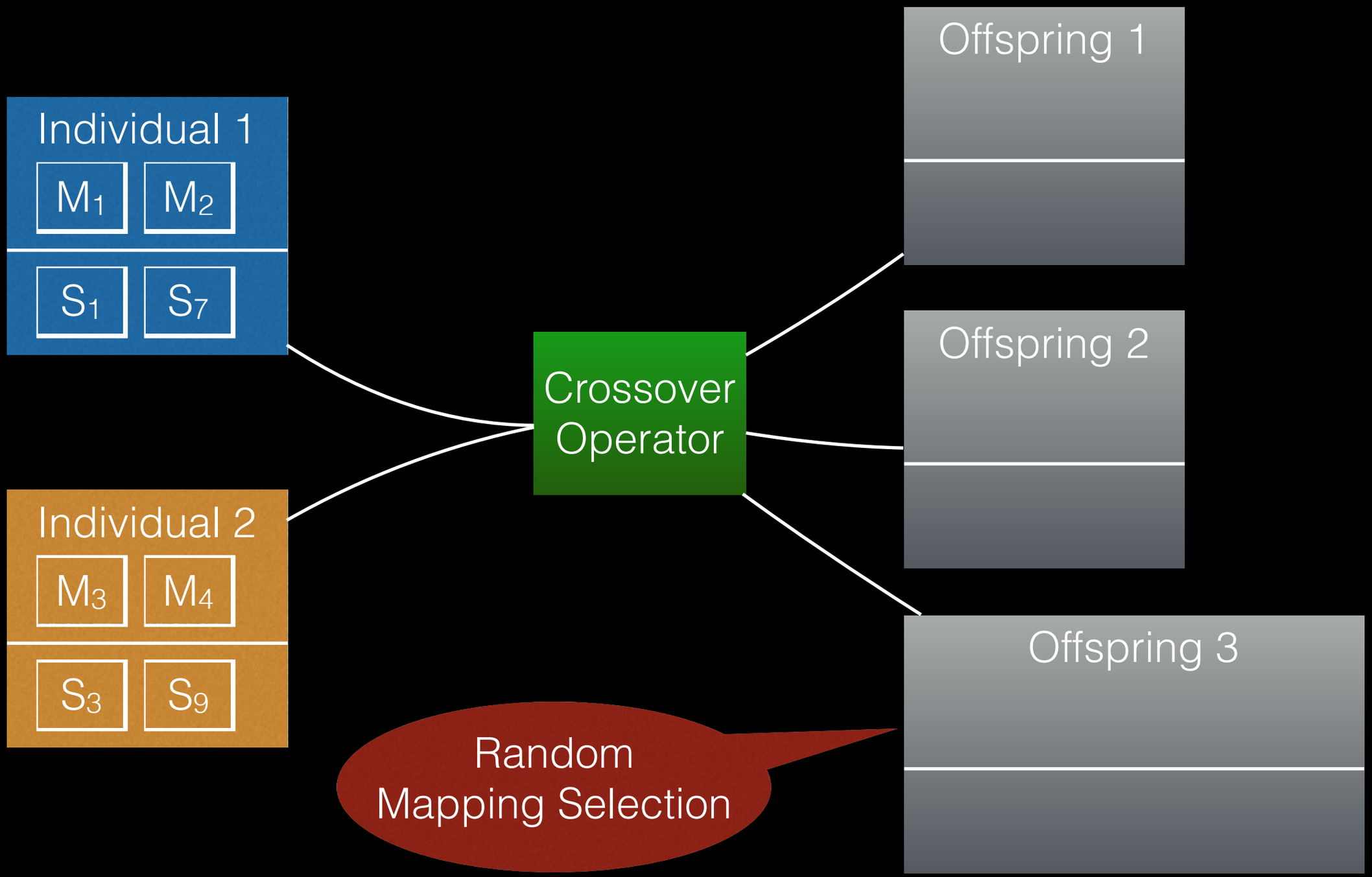
Stage 2 - Gp Operators

Add Statement



UCL

Stage 2 - Gp Operators



UCL

Research Questions

Acceptance Tests

Host

Donor

RQ4: Is autotransplantation useful?

UCL

ch Que

RQ1: Do we break the initial functionality?

RQ2: Have we really added new functionality?

Empirical Study

15 Transplantations
300 Runs
5 Donors
3 Hosts

Case Studies:

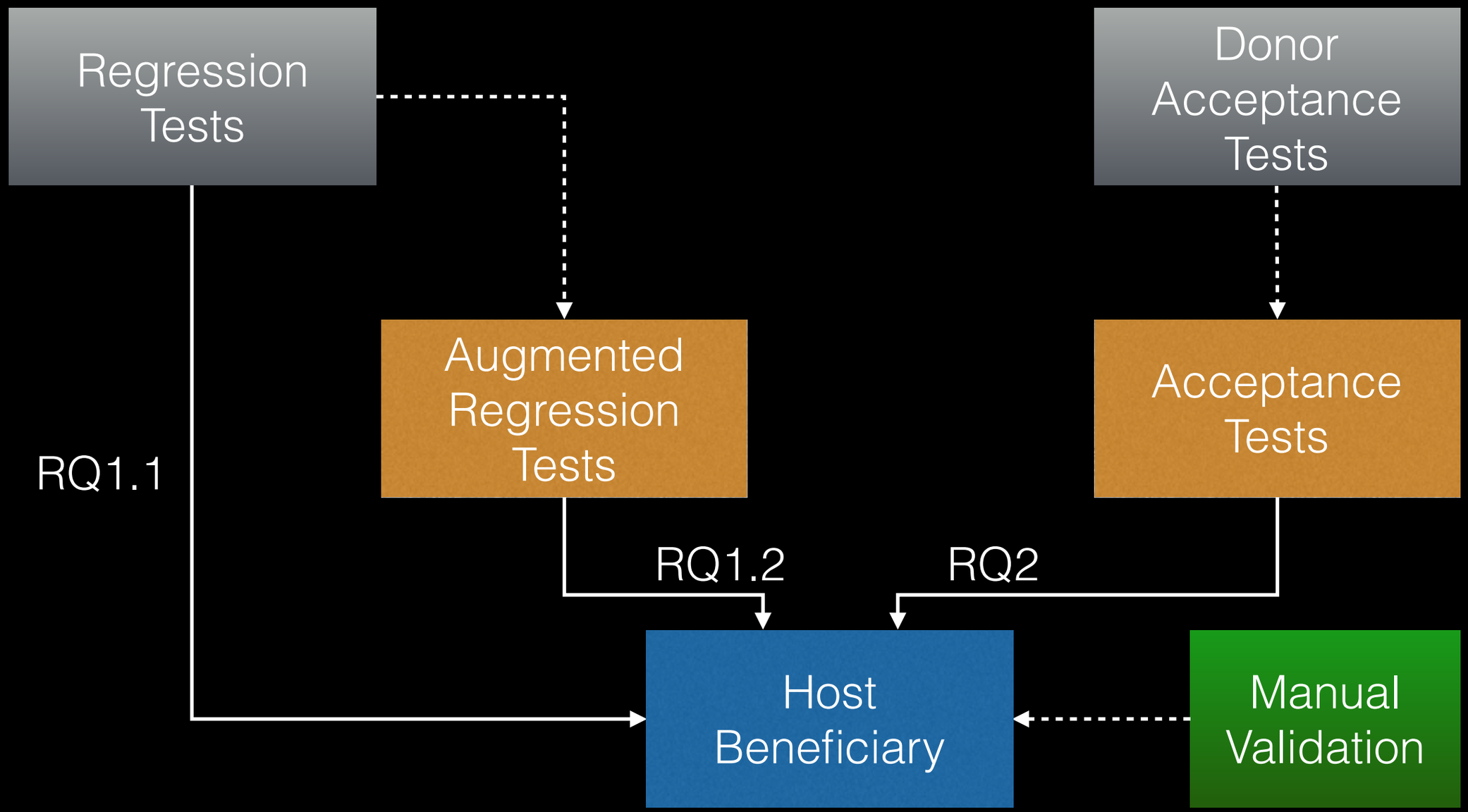
H.264 Encoding Transplantation;
Kate - call graph generation & C indentation;

RQ3: How about the computational effort?

RQ4: Is autotransplantation useful?

UCL

Validation



UCL

Subjects

Subjects	Type	Size KLOC	Reg. Tests	Organ Tests
Idct	Donor	2.3	-	3-5
Mytar	Donor	0.4	-	4
Cflow	Donor	25	-	6-20
Webserver	Donor	1.7	-	3
TuxCrypt	Donor	2.7	-	4-5
Pidgin	Host	363	88	-
Cflow	Host	25	21	-
SoX	Host	43	157	-

Case Study

VLC	Host	422	27	-
Kate	Host	50	238	-
x264	Donor	63	-	5
Cflow	Donor	22	-	13
Indent	Donor	26	-	7

Minimal size: 0.4k

Max size: 422k

Average Donor:16k

Average Host: 213k

UCL

Experimental Methodology and [illegible]

Count LOC
CLOC

Host

Implantation Point

Organ Test Suite

Donor

OE

Validation Test Suites

Coverage Information: Gcov

x 20

GNU Time

64 bit Ubuntu 14.10
16 GB RAM
8 threads

μS[illegible]EL

Host Beneficiary

Implantation Point

Organ

UCL

Empirical Study RQ1,2

Donor	Host	All Passed	Regression	Regression++	Acceptance
Idct	Pidgin	**16**	**20**	**17**	**16**
Mytar	Pidgin	**16**	**20**	**18**	**20**
Web	Pidgin	**0**	**20**	**0**	**18**
Cflow	Pidgin	**15**	**20**	**15**	**16**
Tux	Pidgin	**15**	**20**	**17**	**16**
Idct	Cflow	**16**	**17**	**16**	**16**
Mytar	Cflow	**17**	**17**	**17**	**20**
Web	Cflow	**0**	**0**	**0**	**17**
Cflow	Cflow	**20**	**20**	**20**	**20**
Tux	Cflow	**14**	**15**	**14**	**16**
Idct	SoX	**15**	**18**	**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
			RQ1.1	RQ1.2	RQ2

Empirical Study
RQ3

		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
VLC

Transplant Time & Test Suites

	Time (hours)	Regression	Regression++	Acceptance
H.264	26	1	1	1

UCL

Case Study - Kate

Donor	Host	All Passed	Organ Test Suite	Regression	Regression++	Acceptance
Cflow	Kate	**16**	**18**	**20**	**17**	**18**
Indent	Kate	**18**	**19**	**20**	**18**	**19**
TOTAL		34/40	37/40	40/40	35/40	37/40
		All Passed	—	RQ1.1	RQ1.2	RQ2

		Execution Time (minutes)		
Donor	Host	Average (min)	Std. Dev. (min)	Total (hours)
Cflow	Kate	**101**	**31**	**33**
Indent	Kate	**31**	**6**	**11**
Total		132	18.5	44

UCL