Evolving Fault Localisation

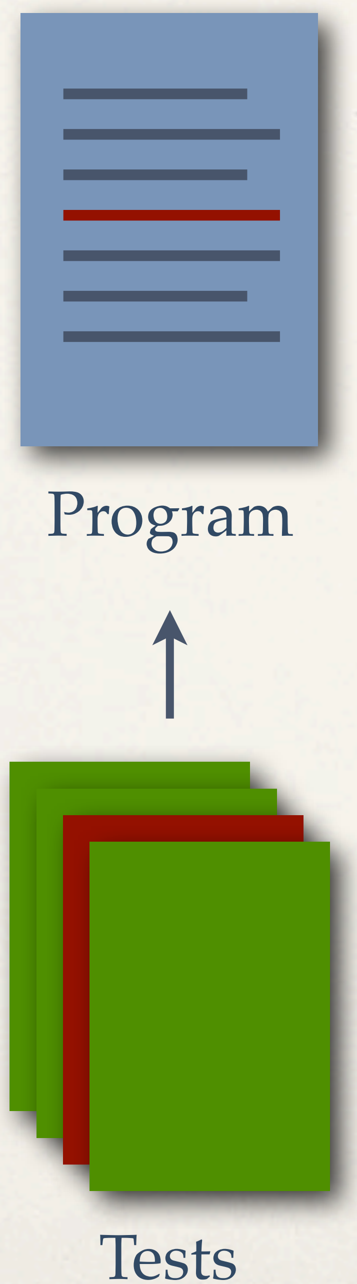
Shin Yoo, University College London, UK

Human Competitive Award, GECCO 2012, 09 July 2012

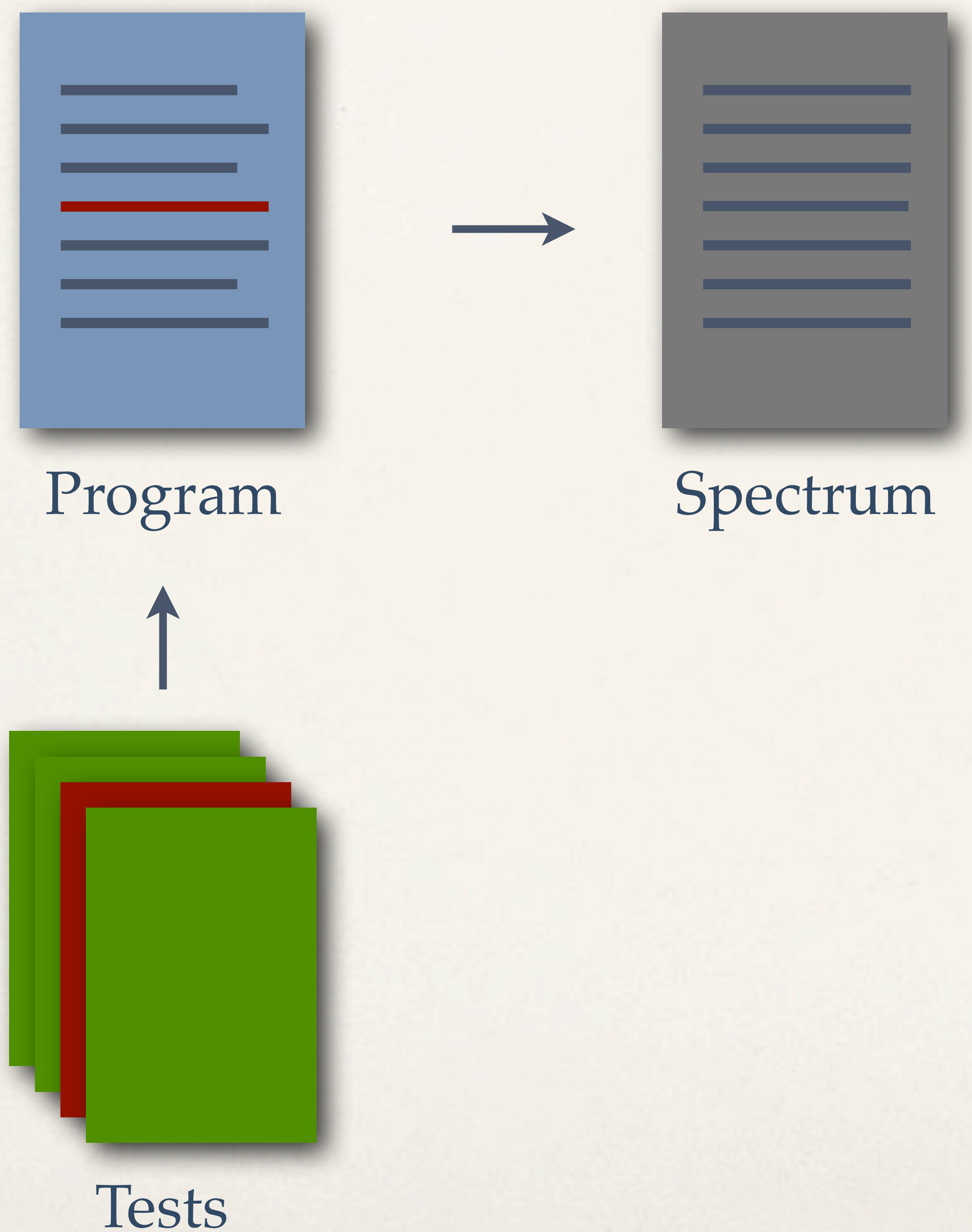
Spectra Based Fault Localisation

Program

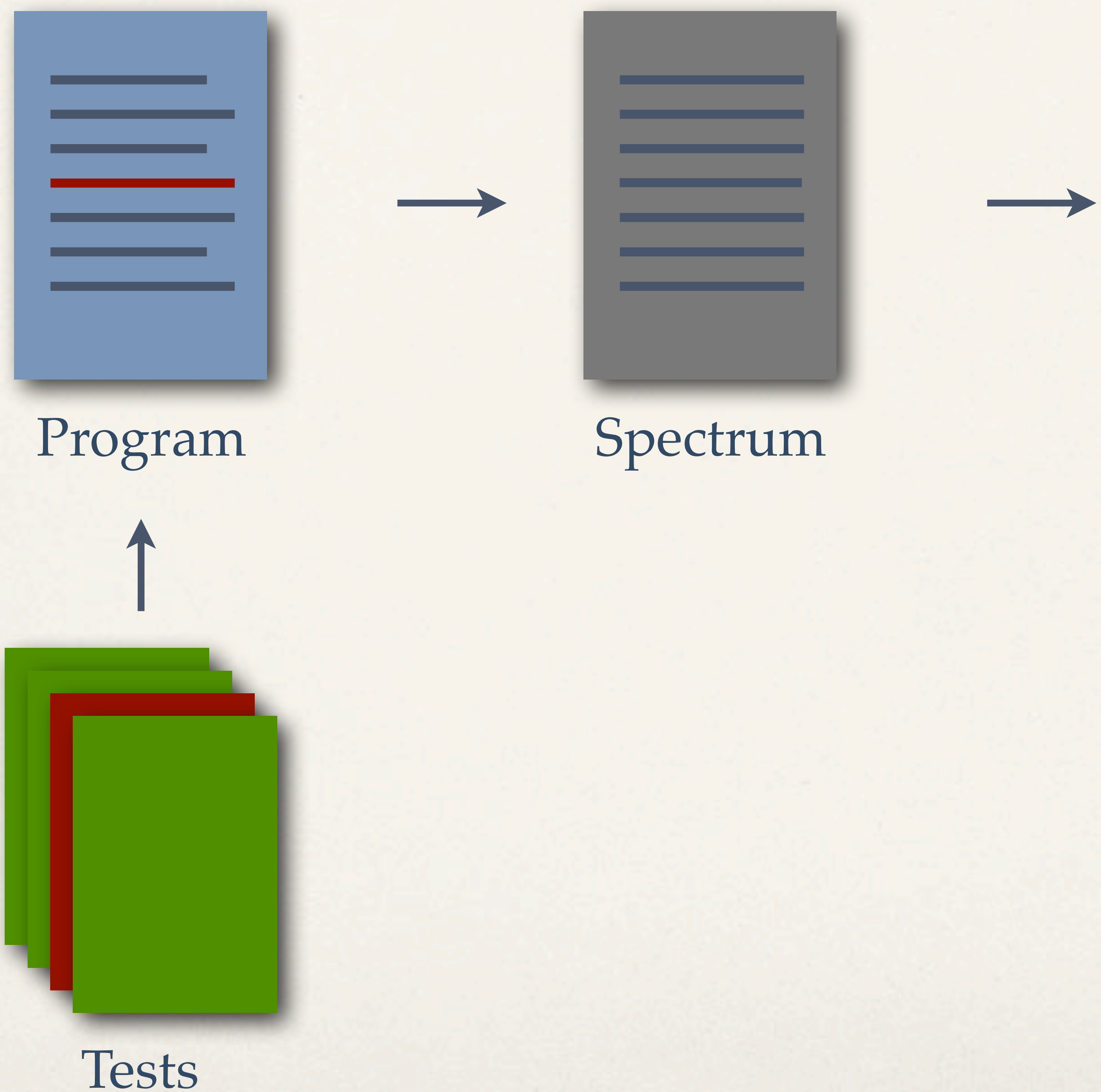
Spectra Based Fault Localisation



Spectra Based Fault Localisation



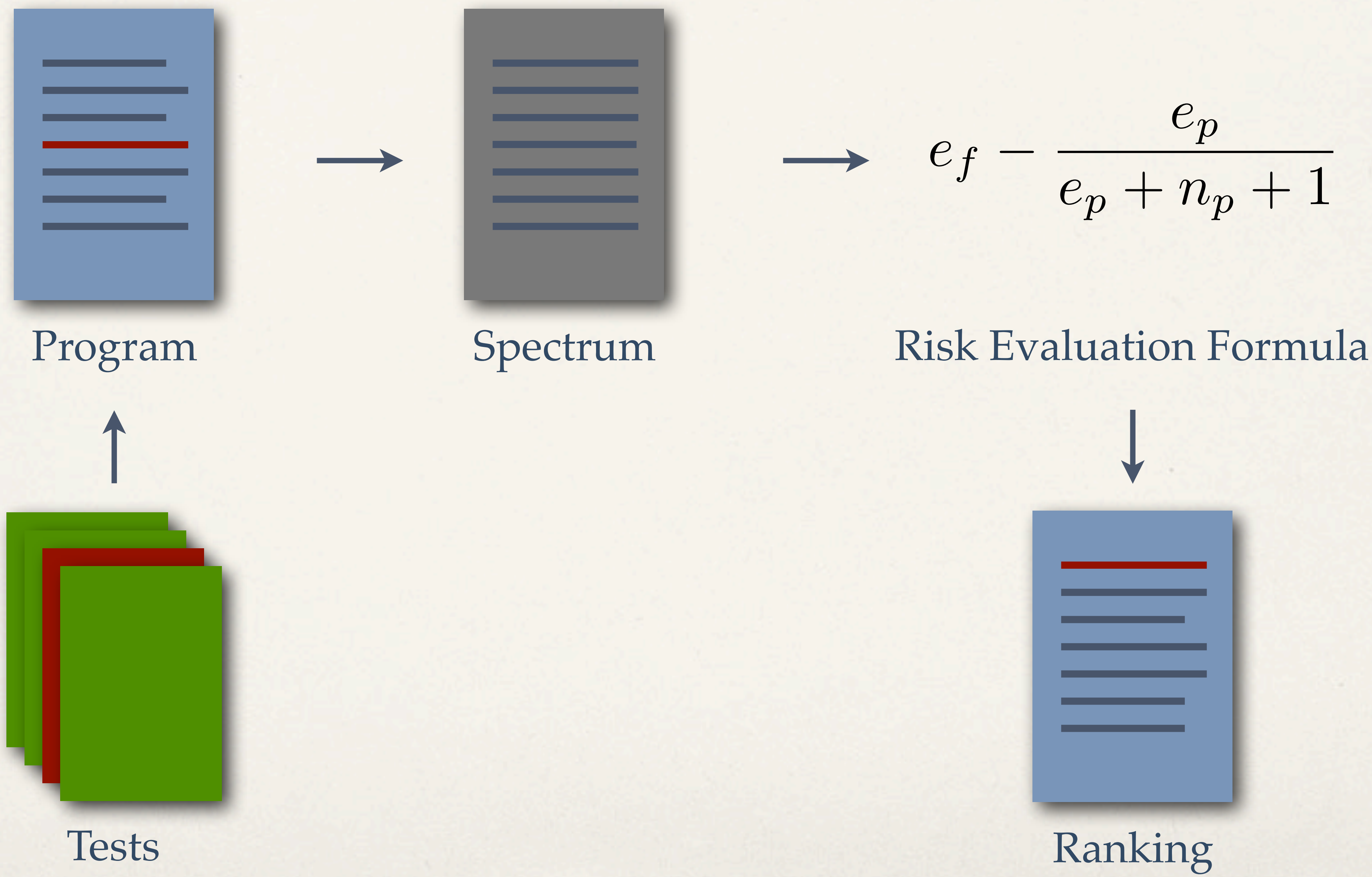
Spectra Based Fault Localisation



$$e_f - \frac{e_p}{e_p + n_p + 1}$$

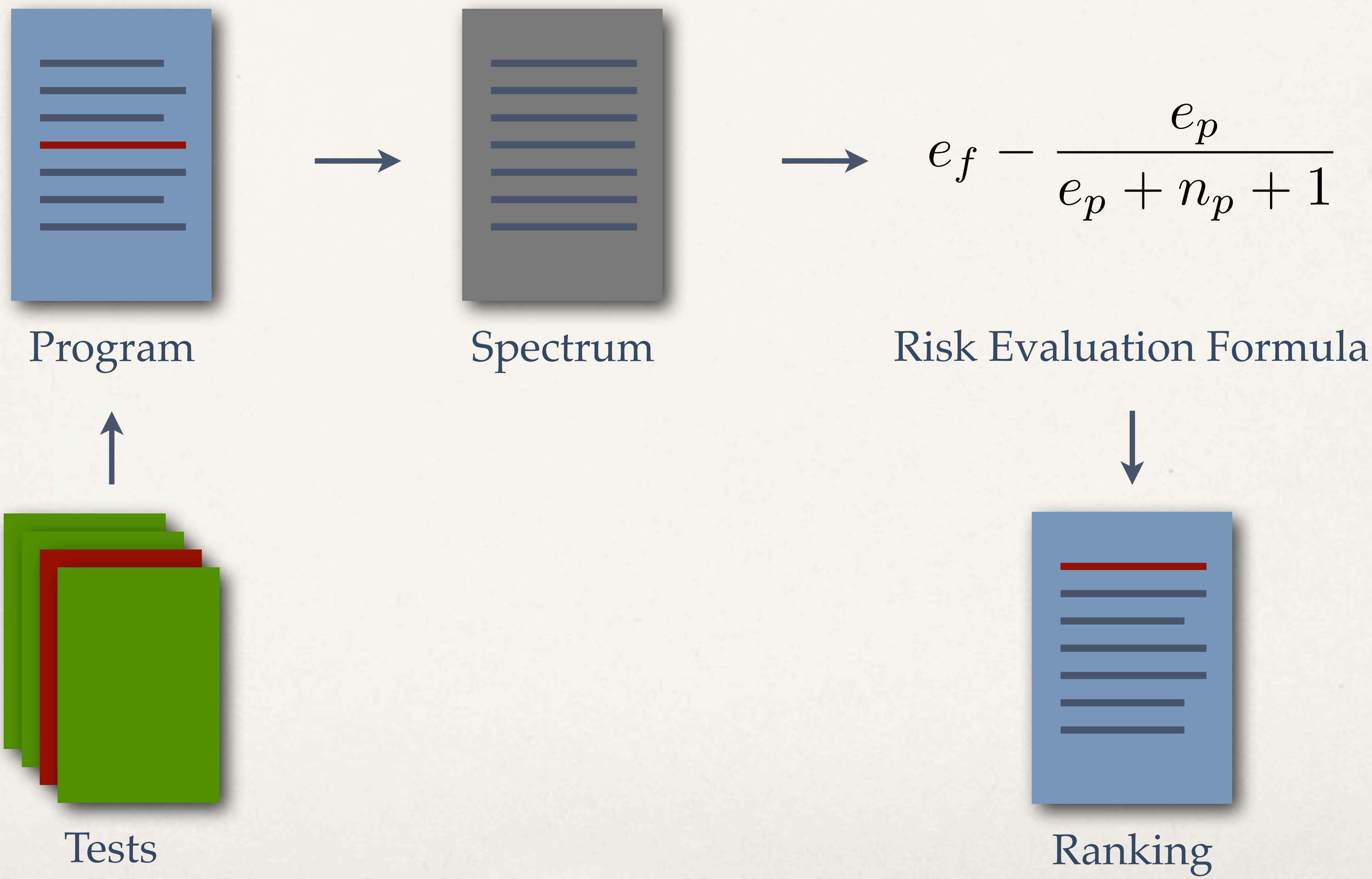
Risk Evaluation Formula

Spectra Based Fault Localisation

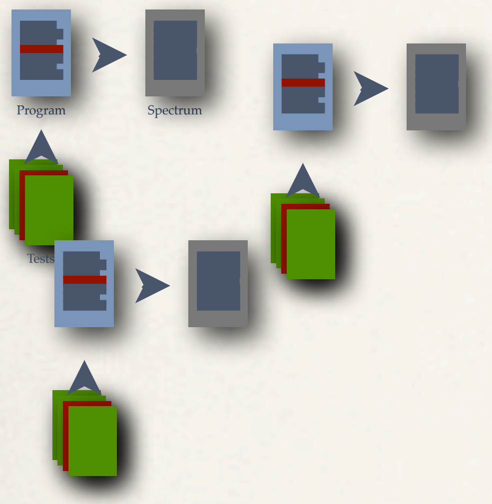


$$e_f - \frac{e_p}{e_p + n_p + 1}$$

Evolving SBFL



Evolving SBFL

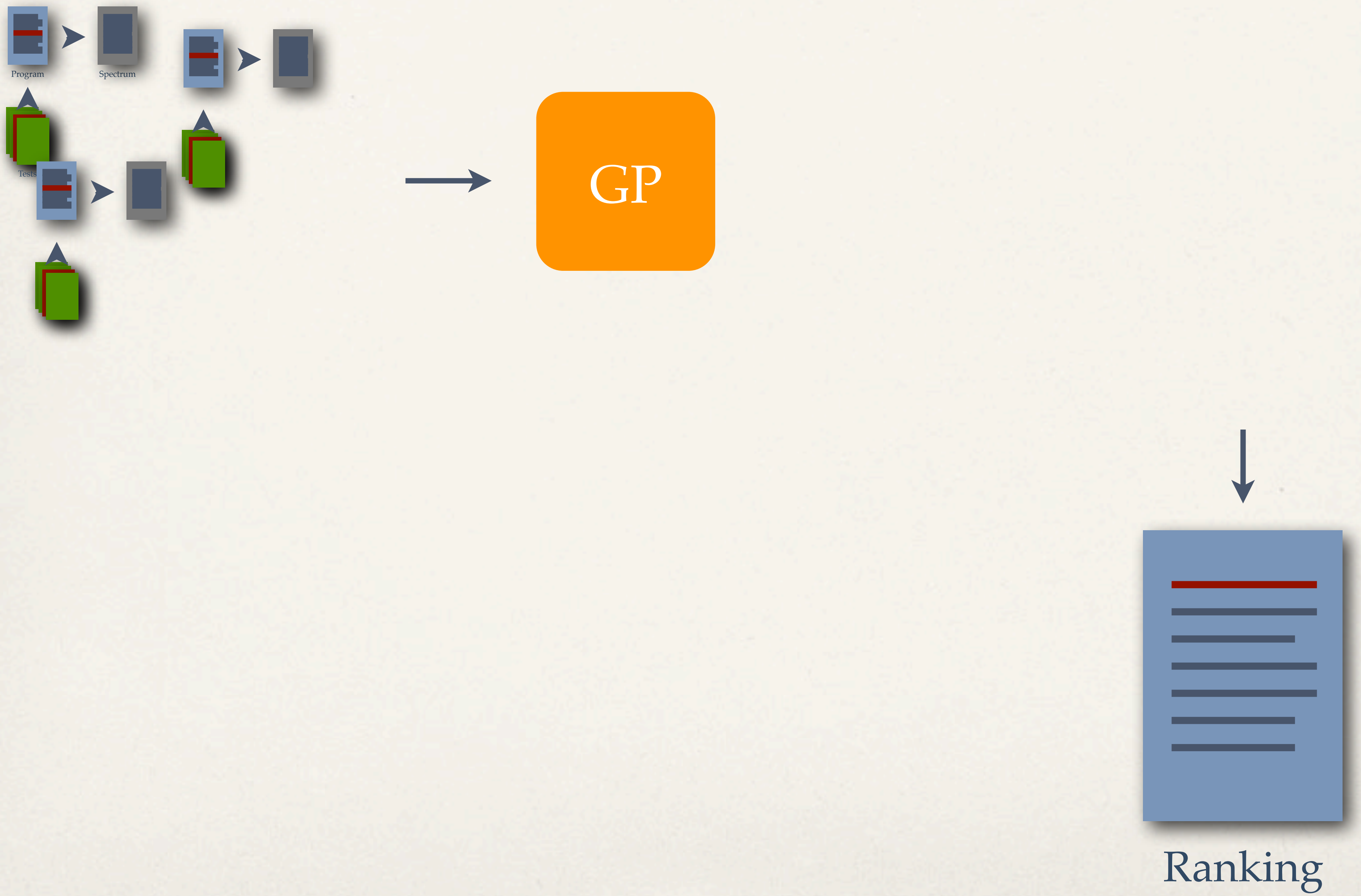


$$e_f - \frac{e_p}{e_p + n_p + 1}$$

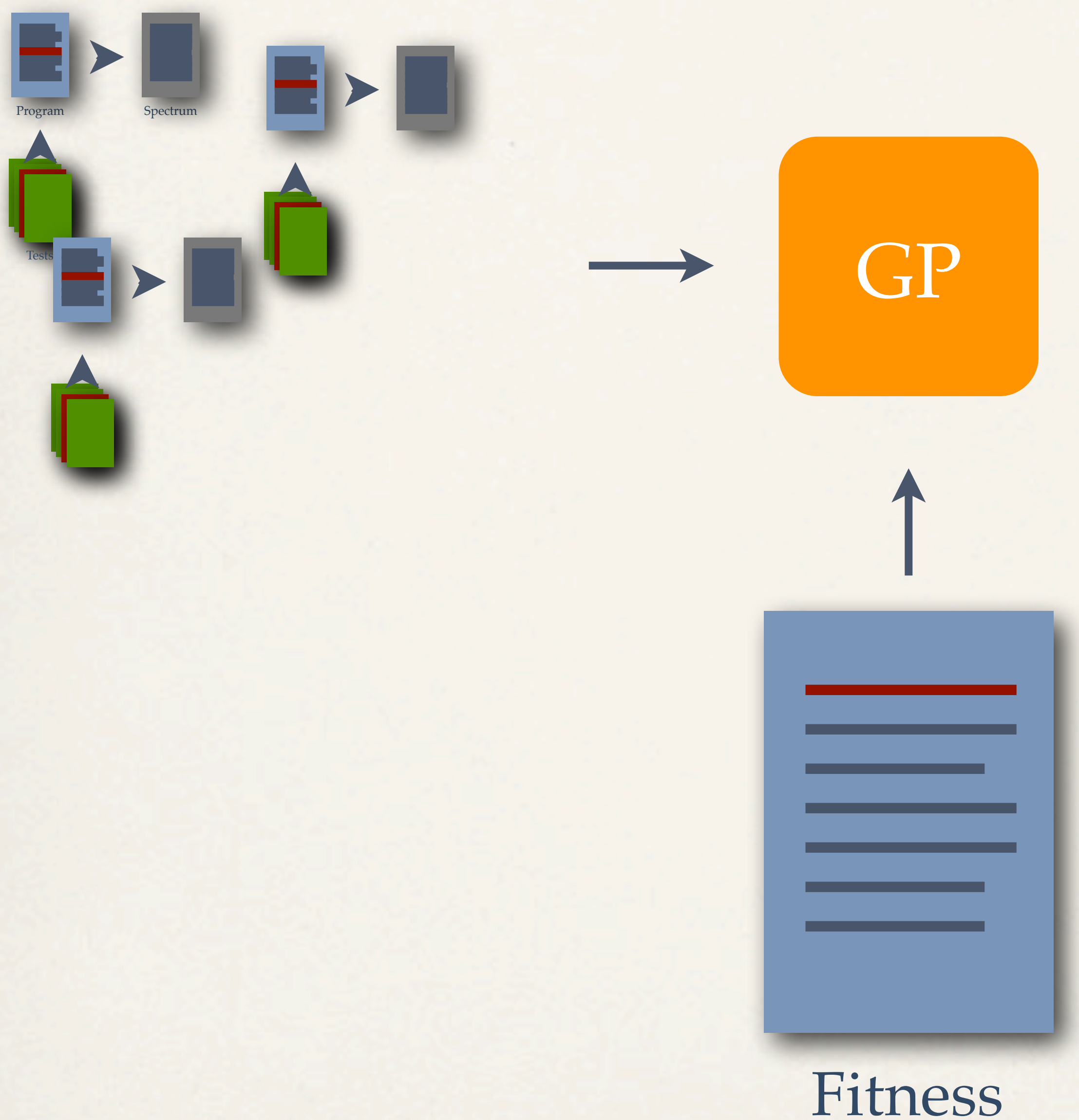
Risk Evaluation Formula

Ranking

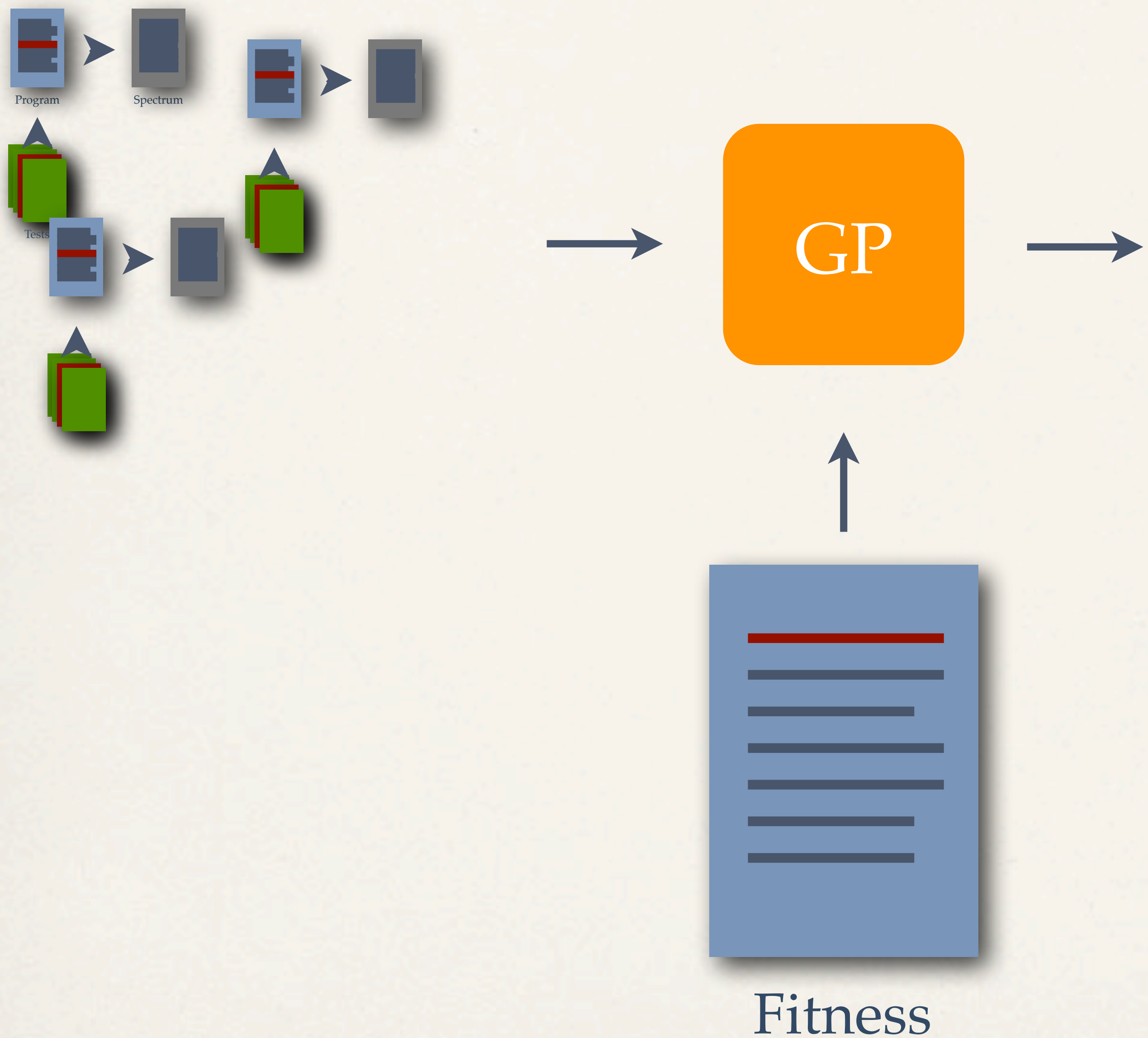
Evolving SBFL



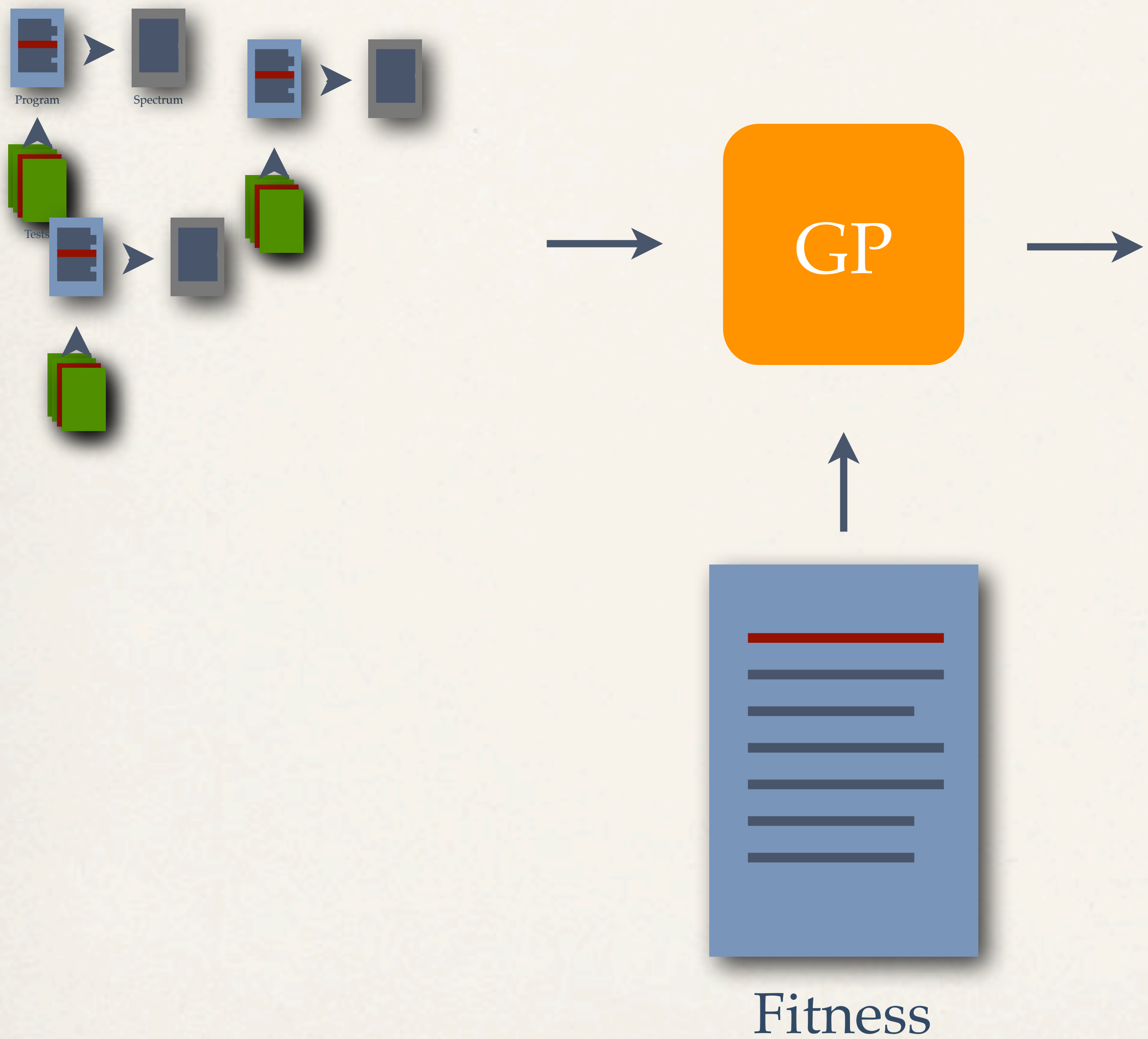
Evolving SBFL



Evolving SBFL



Evolving SBFL



$$e_f^2(2e_p + 2e_f + 3n_p)$$

$$e_f^2(e_f^2 + \sqrt{n_p})$$

$$\dots$$

State of the Art

Over 30 formulæ in the literature, with various empirical studies with slightly different results

State of the Art

$$\frac{2e_f}{e_f + n_f + e_p}$$

$$\frac{e_f}{e_f + n_p + 2(e_p + n_f)}$$

$$\frac{e_f}{e_f + n_f + e_p}$$

$$\frac{2(e_f + n_p)}{2(e_f + n_p) + e_p + n_f}$$

$$\frac{e_f}{e_f + 2(n_f + e_p)}$$

$$\frac{e_f}{n_f + e_p}$$

Over 30 formulæ in the literature, with various empirical studies with slightly different results

$$\frac{e_f + n_p}{n_f + e_p}$$

$$\frac{e_f}{e_f + n_f + e_p + n_p}$$

$$\frac{e_f + n_p}{e_f + n_f + e_p + n_p}$$

$$\frac{2e_f}{2e_f + n_f + e_p}$$

$$\frac{e_f + n_p - n_f - e_p}{e_f + n_f + e_p + n_p}$$

$$\frac{1}{2}\left(\frac{e_f}{e_f + n_f} + \frac{e_f}{e_f + e_p}\right)$$

$$\frac{\frac{e_f}{e_f+n_f}}{\frac{e_p}{e_p+n_p} + \frac{e_f}{e_f+n_f}}$$

State of the Art

A Model for Spectra-based Software Diagnosis

LEE NAISH,
HUA JIE LEE
and
KOTAGIRI RAMAMOHANARAO
University of Melbourne

This paper presents an improved approach to assist diagnosis of failures in software (fault localisation) by ranking program statements or blocks according to how likely they are to be buggy. We present a very simple single-bug program to model the problem. By examining different possible execution paths through this model program over a number of test cases, the effectiveness of different proposed spectral ranking methods can be evaluated in idealised conditions. The results are remarkably consistent to those arrived at empirically using the Siemens test suite and Space benchmarks. The model also helps identify groups of metrics which are equivalent for ranking. Due to the simplicity of the model, an optimal ranking method can be devised. This new method out-performs previously proposed methods for the model program, the Siemens test suite and Space. It also helps provide insight into other ranking methods.

Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*

General Terms: Performance, Theory

Additional Key Words and Phrases: fault localisation, program spectra, statistical debugging

1. INTRODUCTION

Despite the achievements made in software development, bugs are still pervasive and diagnosis of software failures remains an active research area. One of many useful sources of data to help diagnosis is the dynamic behaviour of software as it is executed over a set of test cases where it can be determined if each result is correct or not (each test case passes or fails). Software can be instrumented automatically to gather data such as the statements that are executed in each test case. A summary of this data, often called program spectra, can be used to rank the parts of the program according to how likely it is they contain a bug. Ranking is done by sorting based on the value of a numeric function (we use the term *ranking metric* or simply metric) applied to the data for each part of the program. There is extensive literature on spectra-based methods in other domains, notably classification in botany, and this is the source for many ranking metrics that can be used for software diagnosis. We make the following contributions to this area:

(1) We propose a model-based approach to gain insight into software diagnosis.

Thanks to Tim Miller and the anonymous referees for their comments on drafts of this paper.

Optimality Proof (Naish et al. 2011)

$$Op1 = \begin{cases} -1 & \text{if } n_f > 0 \\ n_p & \text{otherwise} \end{cases} \qquad Op2 = e_f - \frac{e_p}{e_p + n_p + 1}$$

State of the Art

A Model for Spectra-based Software Diagnosis

LEE NAISH,
HUA JIE LEE
and
KOTAGIRI RAMAMOHANARAO
University of Melbourne

This paper presents an improved approach to assist diagnosis of failures in software (fault localisation) by ranking program statements or blocks according to how likely they are to be buggy. We present a very simple single-bug program to model the problem. By examining different possible execution paths through this model program over a number of test cases, the effectiveness of different proposed spectral ranking methods can be evaluated in idealised conditions. The results are remarkably consistent to those arrived at empirically using the Siemens test suite and Space benchmarks. The model also helps identify groups of metrics which are equivalent for ranking. Due to the simplicity of the model, an optimal ranking method can be devised. This new method out-performs previously proposed methods for the model program, the Siemens test suite and Space. It also helps provide insight into other ranking methods.

Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*

General Terms: Performance, Theory

Additional Key Words and Phrases: fault localisation, program spectra, statistical debugging

1. INTRODUCTION

Despite the achievements made in software development, bugs are still pervasive and diagnosis of software failures remains an active research area. One of many useful sources of data to help diagnosis is the dynamic behaviour of software as it is executed over a set of test cases where it can be determined if each result is correct or not (each test case passes or fails). Software can be instrumented automatically to gather data such as the statements that are executed in each test case. A summary of this data, often called program spectra, can be used to rank the parts of the program according to how likely it is they contain a bug. Ranking is done by sorting based on the value of a numeric function (we use the term *ranking metric* or simply metric) applied to the data for each part of the program. There is extensive literature on spectra-based methods in other domains, notably classification in botany, and this is the source for many ranking metrics that can be used for software diagnosis. We make the following contributions to this area:

(1) We propose a model-based approach to gain insight into software diagnosis.

Thanks to Tim Miller and the anonymous referees for their comments on drafts of this paper.

Optimality Proof (Naish et al. 2011)

$$Op1 = \begin{cases} -1 & \text{if } n_f > 0 \\ n_p & \text{otherwise} \end{cases} \qquad Op2 = e_f - \frac{e_p}{e_p + n_p + 1}$$

But the proof is against a specific model

```
if (t1())
    s1();          /* S1 */
else
    s2();          /* S2 */
if (t2())
    x = True;      /* S3 */
else
    x = t3();      /* S4 - BUG */
```

State of the Art

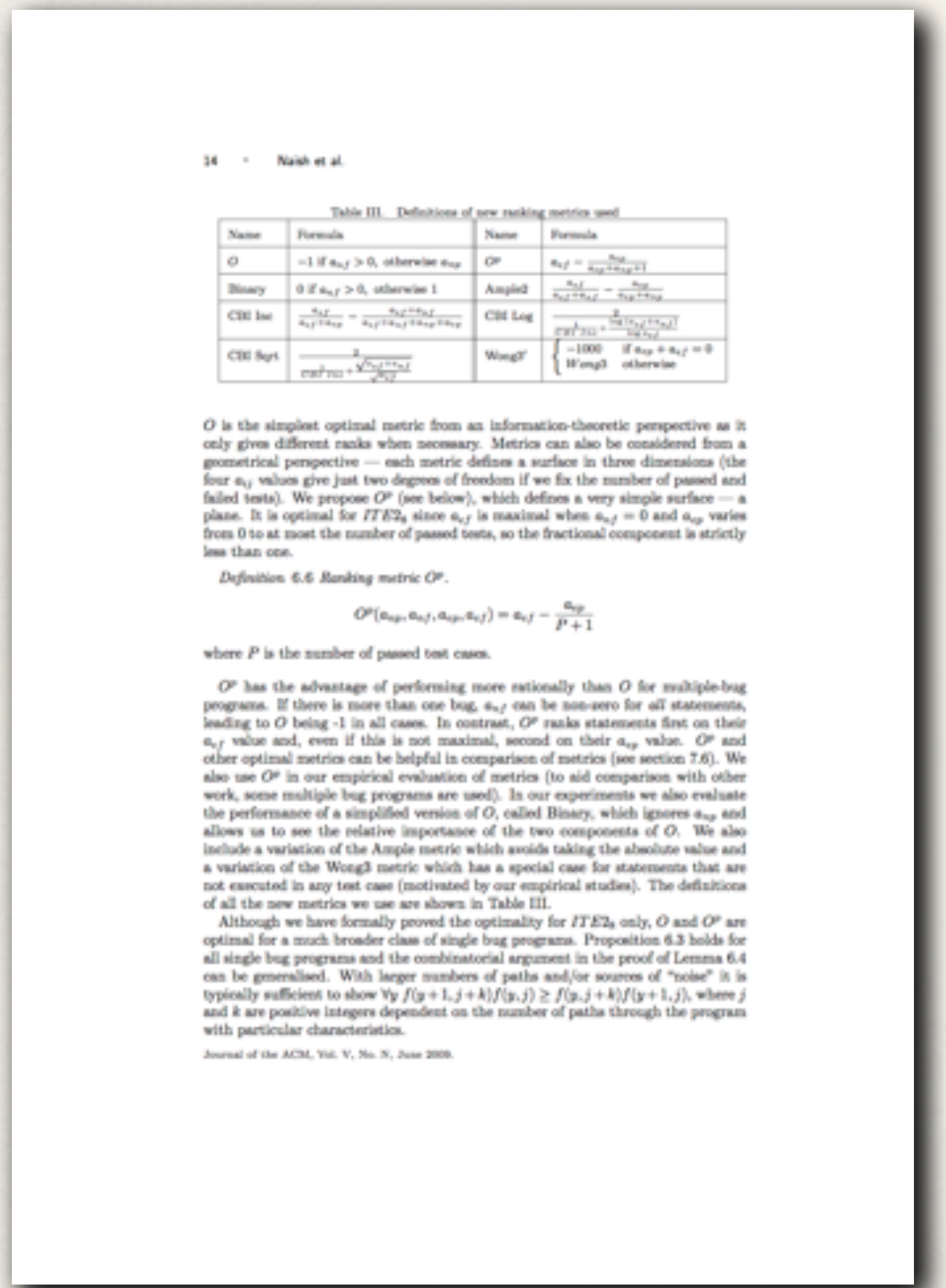

14 · Naish et al.

Table III. Definitions of new ranking metrics used

Journal of the ACM, Vol. V, No. N, June 2009.

Optimality Proof (Naish et al. 2011)

$$Op1 = \begin{cases} -1 & \text{if } n_f > 0 \\ n_p & \text{otherwise} \end{cases} \qquad Op2 = e_f - \frac{e_p}{e_p + n_p + 1}$$

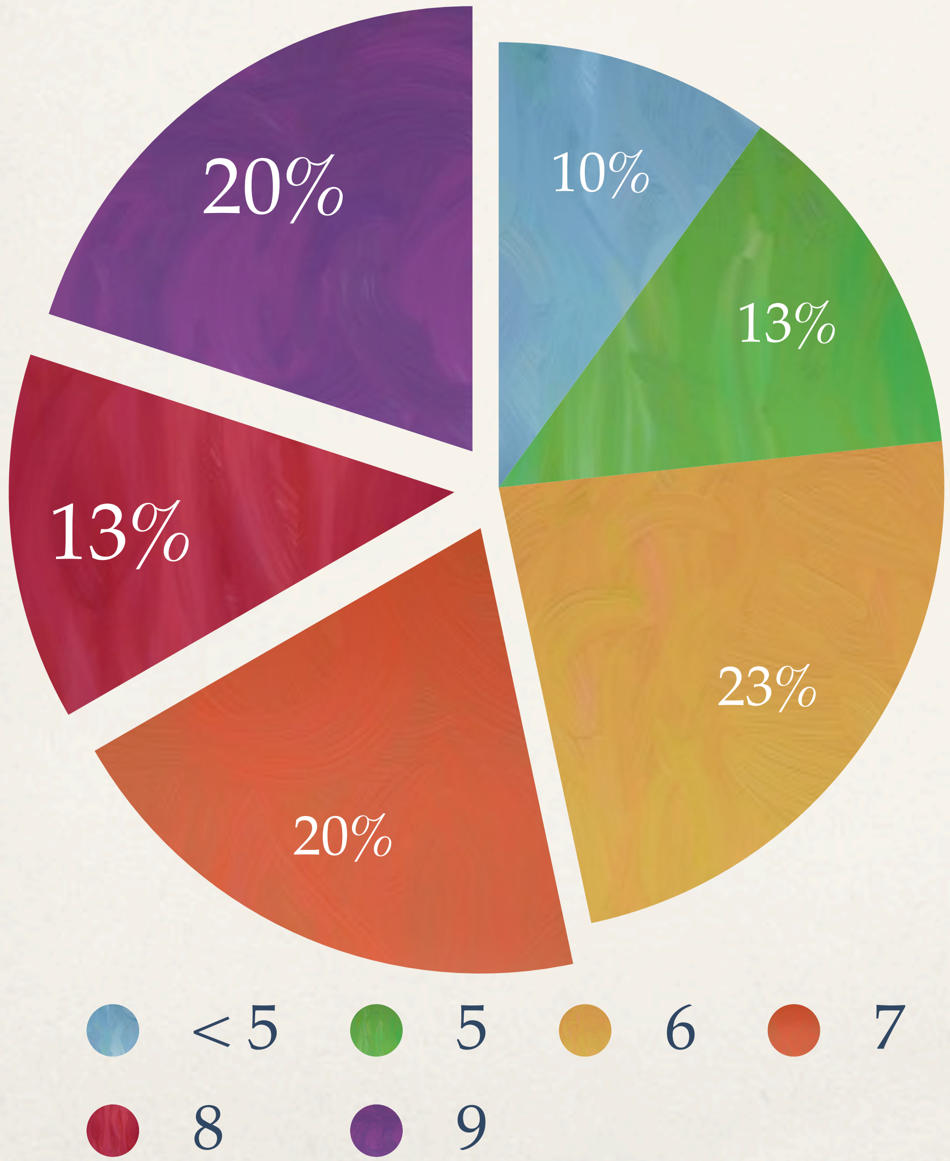
But the proof is against a specific model

```
if (t1())
    s1();          /* S1 */
else
    s2();          /* S2 */
if (t2())
    x = True;      /* S3 */
else
    x = t3();      /* S4 - BUG */
```

... not to mention hard.

Human Competitiveness

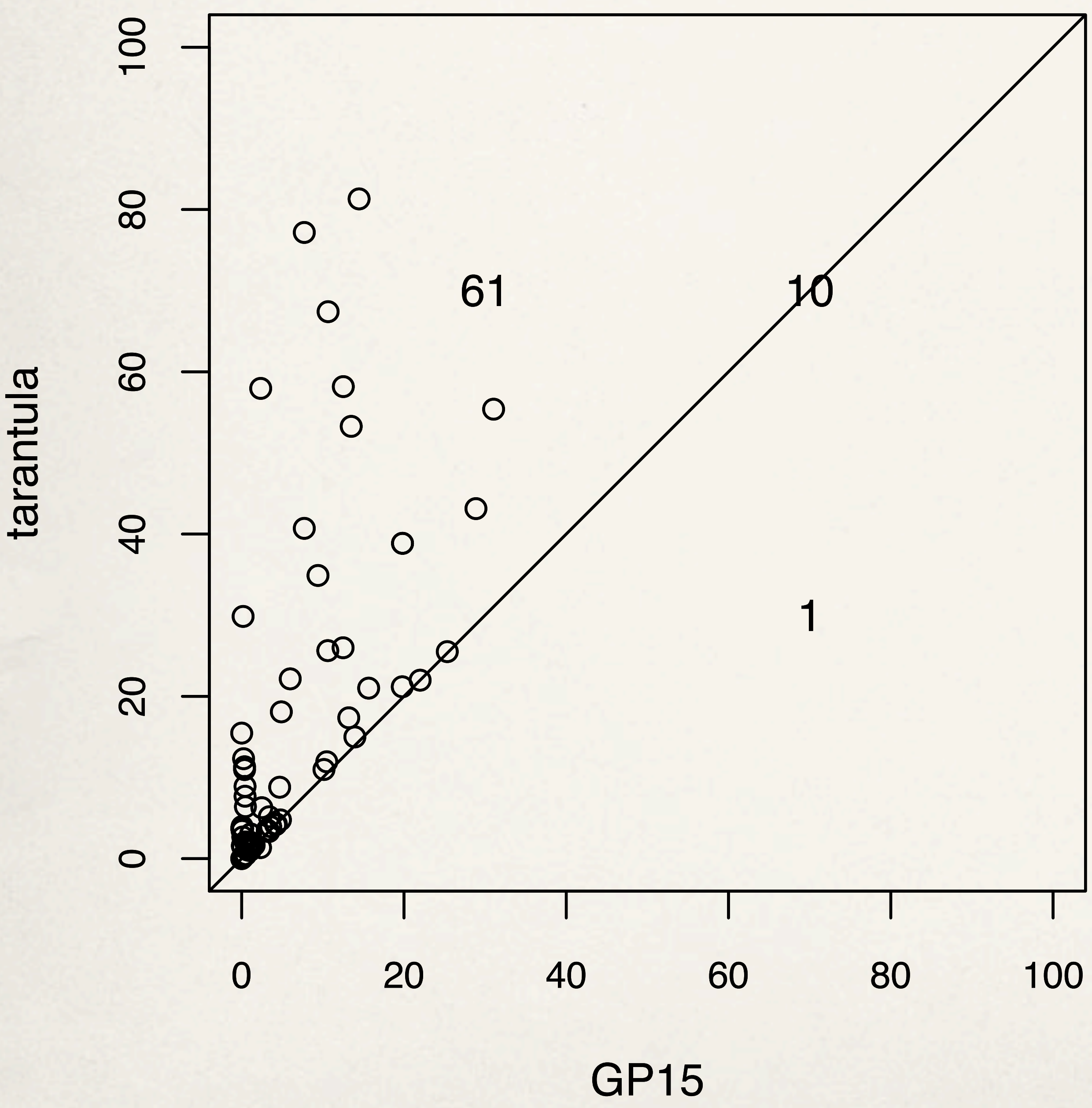
How many of 9 Existing Techniques can 30 GP runs match and/or outperform?



- 6 runs outperform 8 existing techniques and match/outperform one of the state of the art with proof (Op1 and Op2).
- 16 runs outperform all 7 existing techniques without proof.

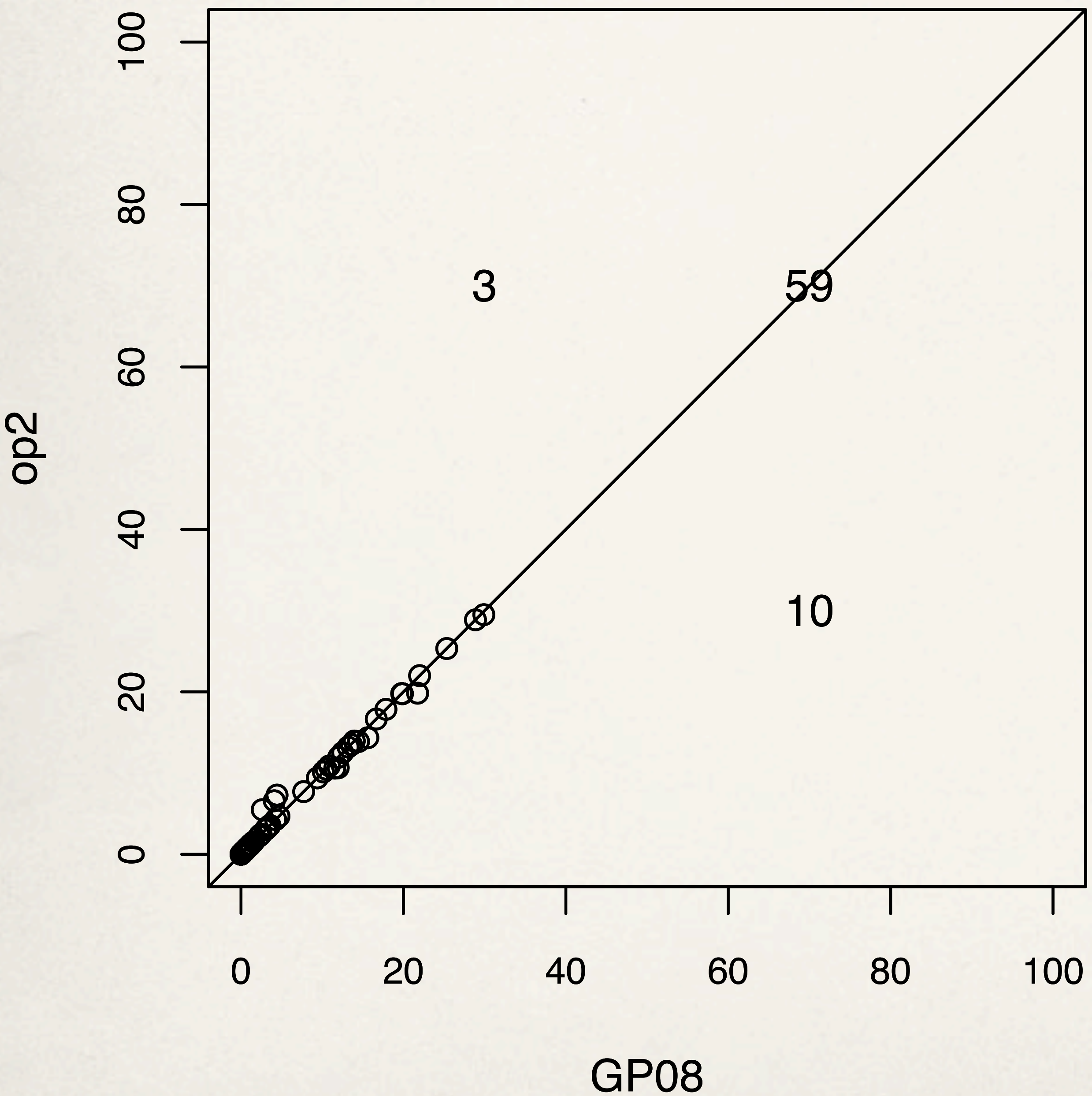
Four Unix tools with 92 faults: 20 random faults for training, 72 for evaluation.

Human Competitiveness



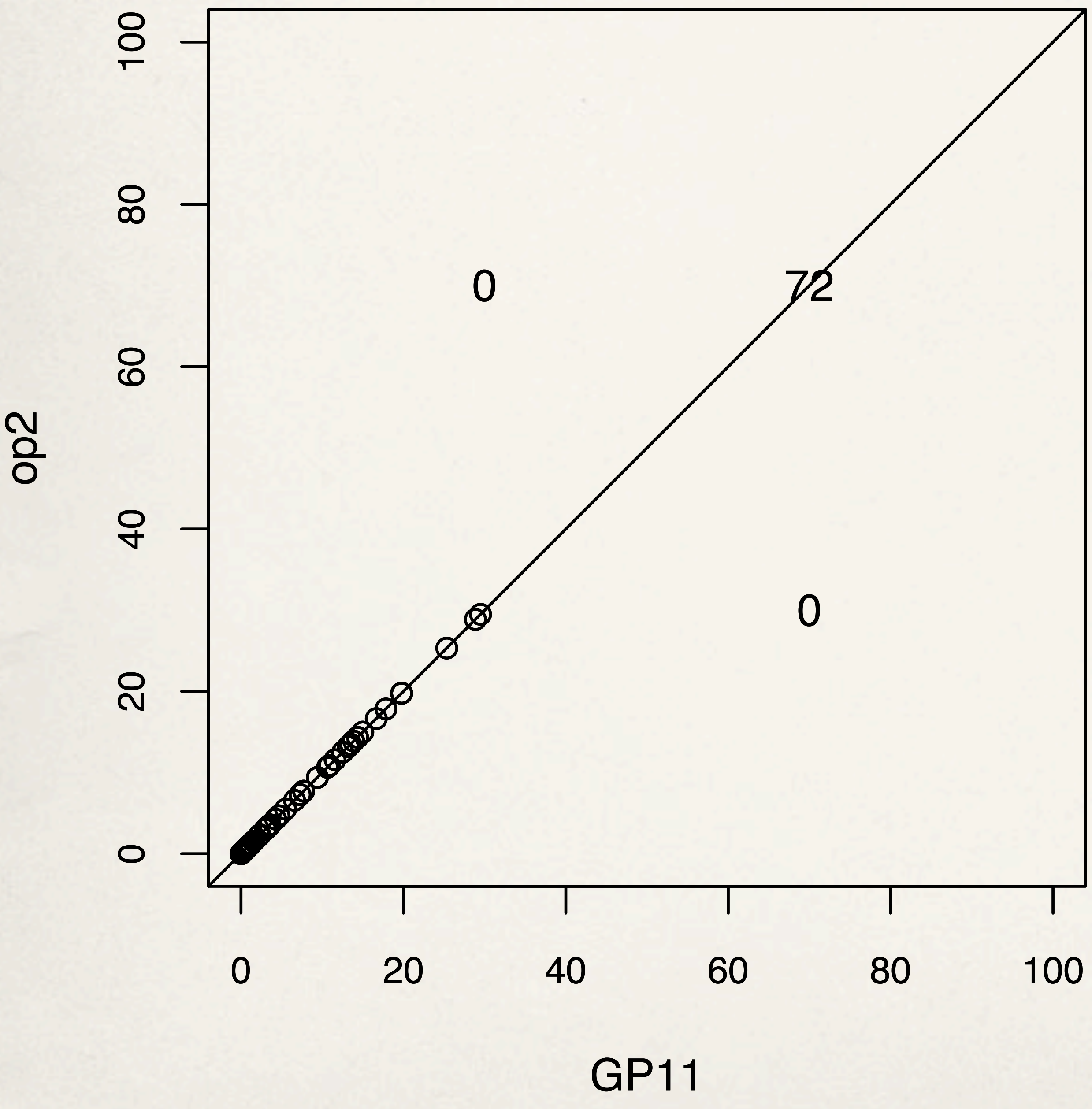
- Per-fault view shows that evolved techniques can outperform ones with optimality proofs.

Human Competitiveness



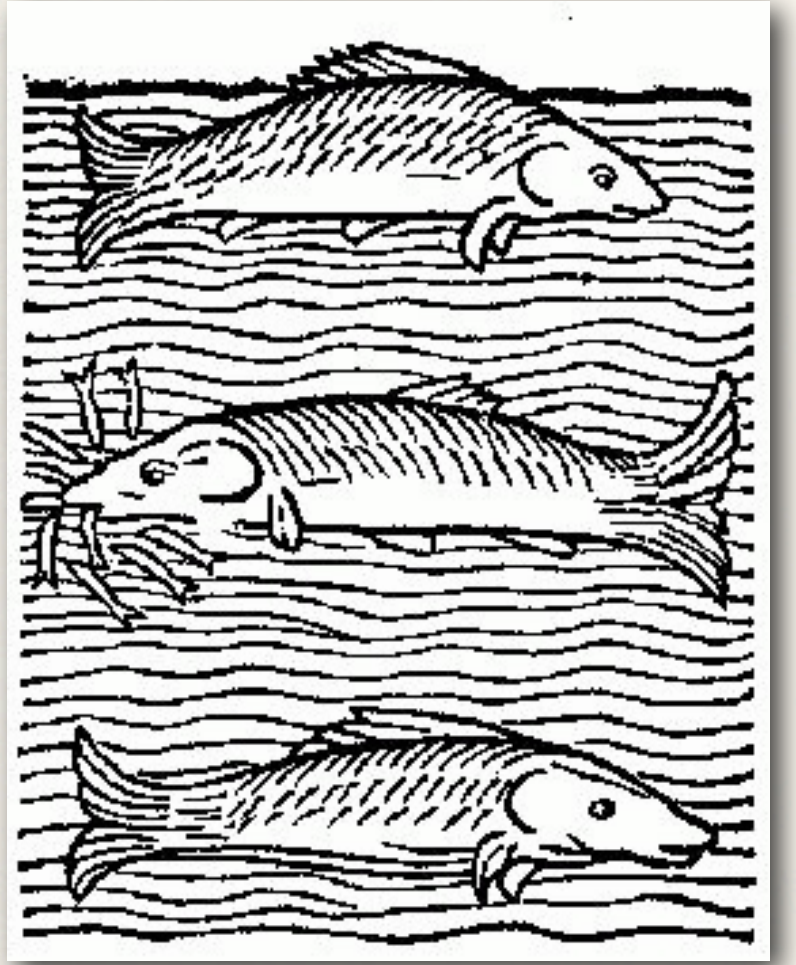
- Per-fault view shows that evolved techniques can outperform ones with optimality proofs.

Human Competitiveness

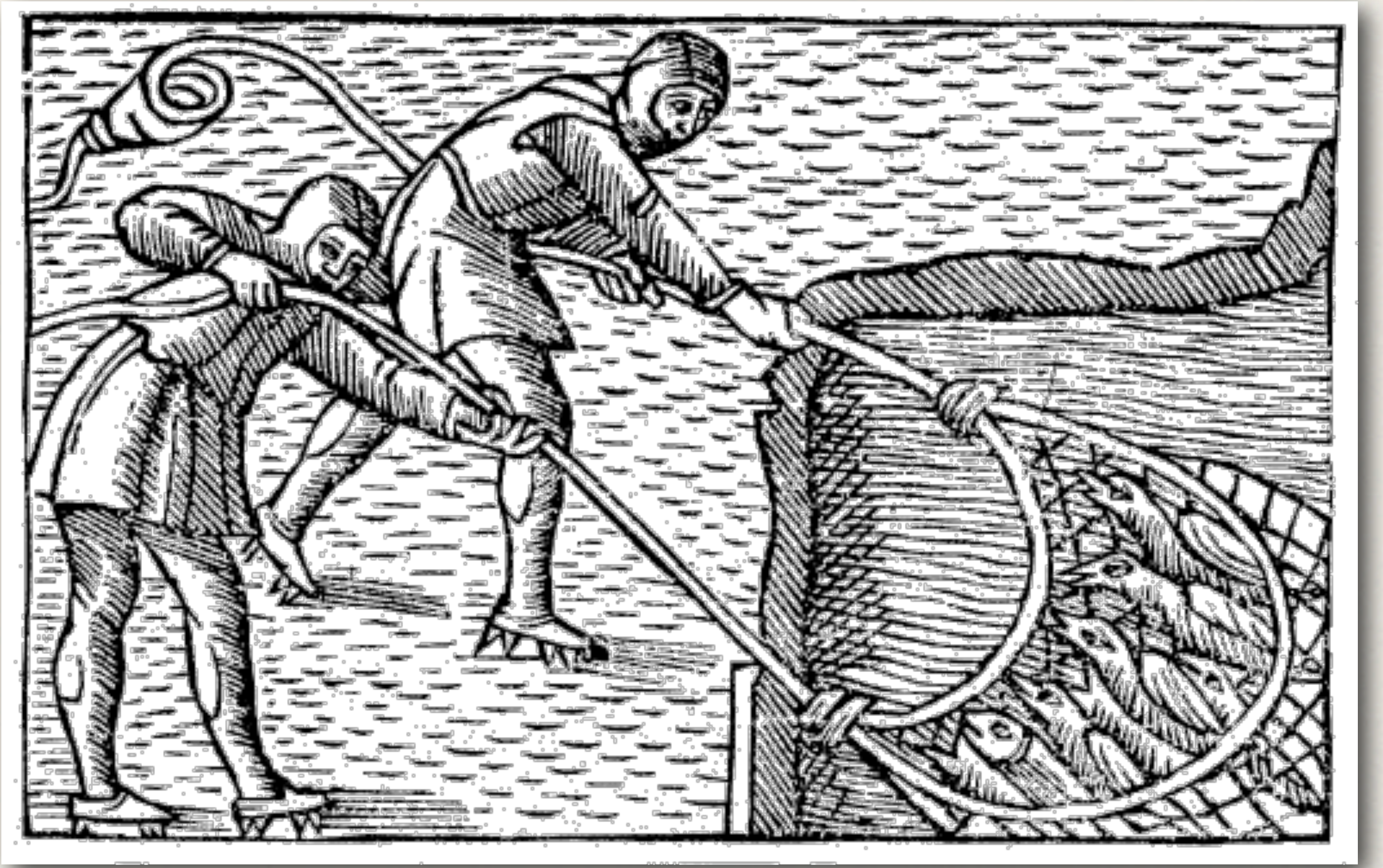


- Per-fault view shows that evolved techniques can outperform ones with optimality proofs.

Future of Search-Based Software Engineering



From Solutions to
Generic Problems...



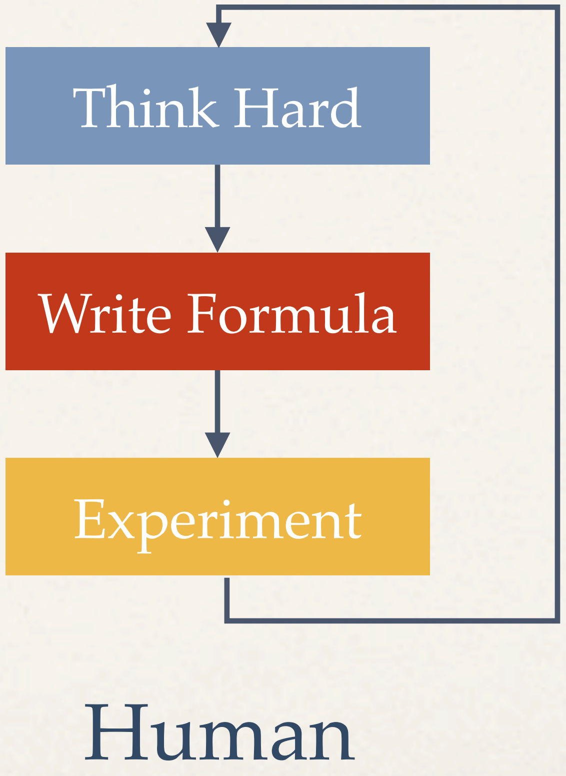
To Techniques and Strategies
for **Your** Problems.

The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.

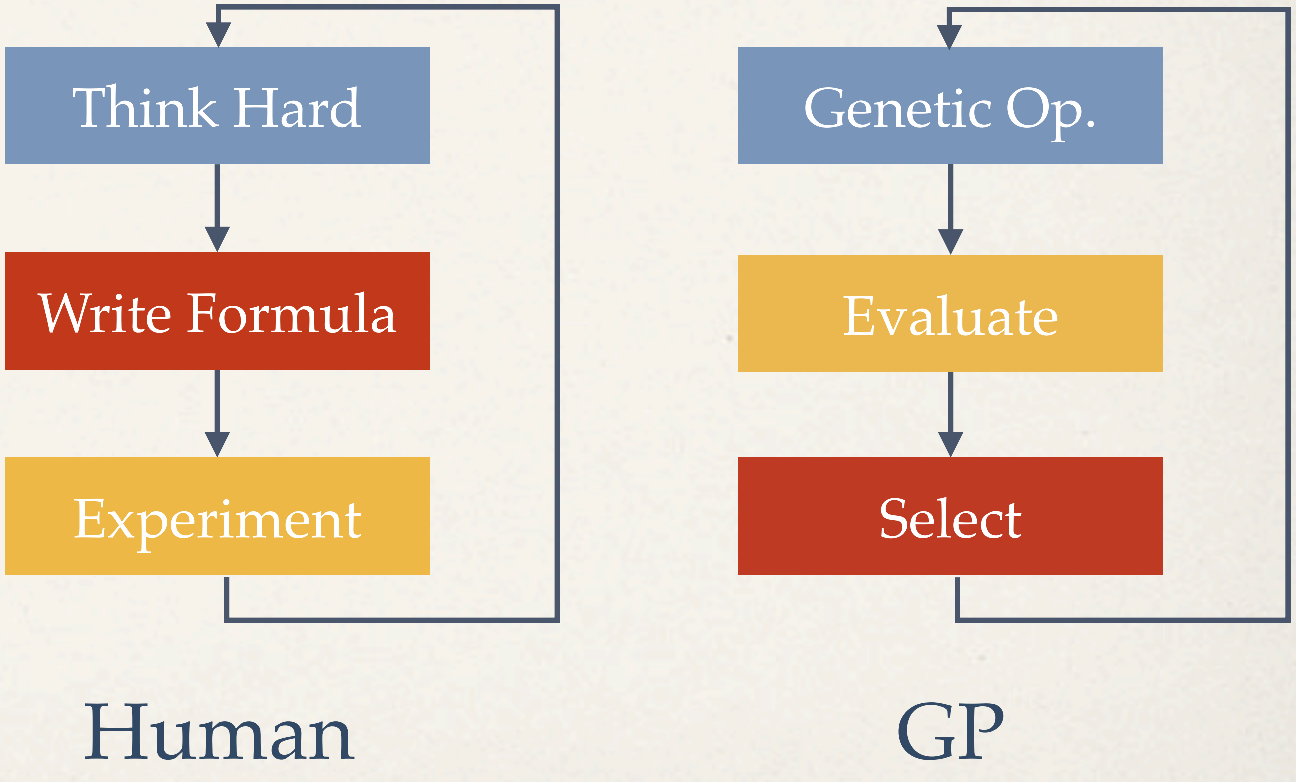
The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.



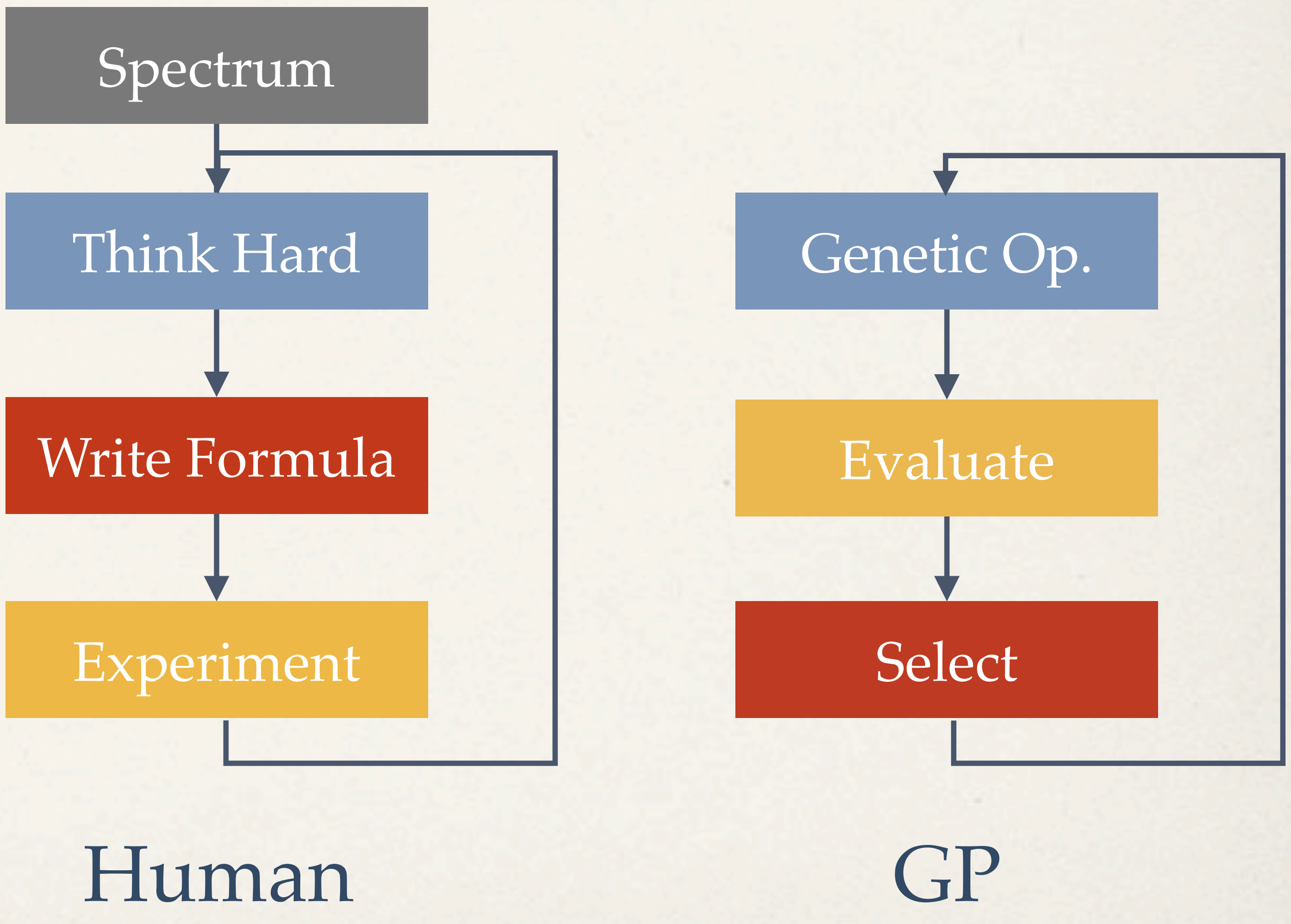
The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.



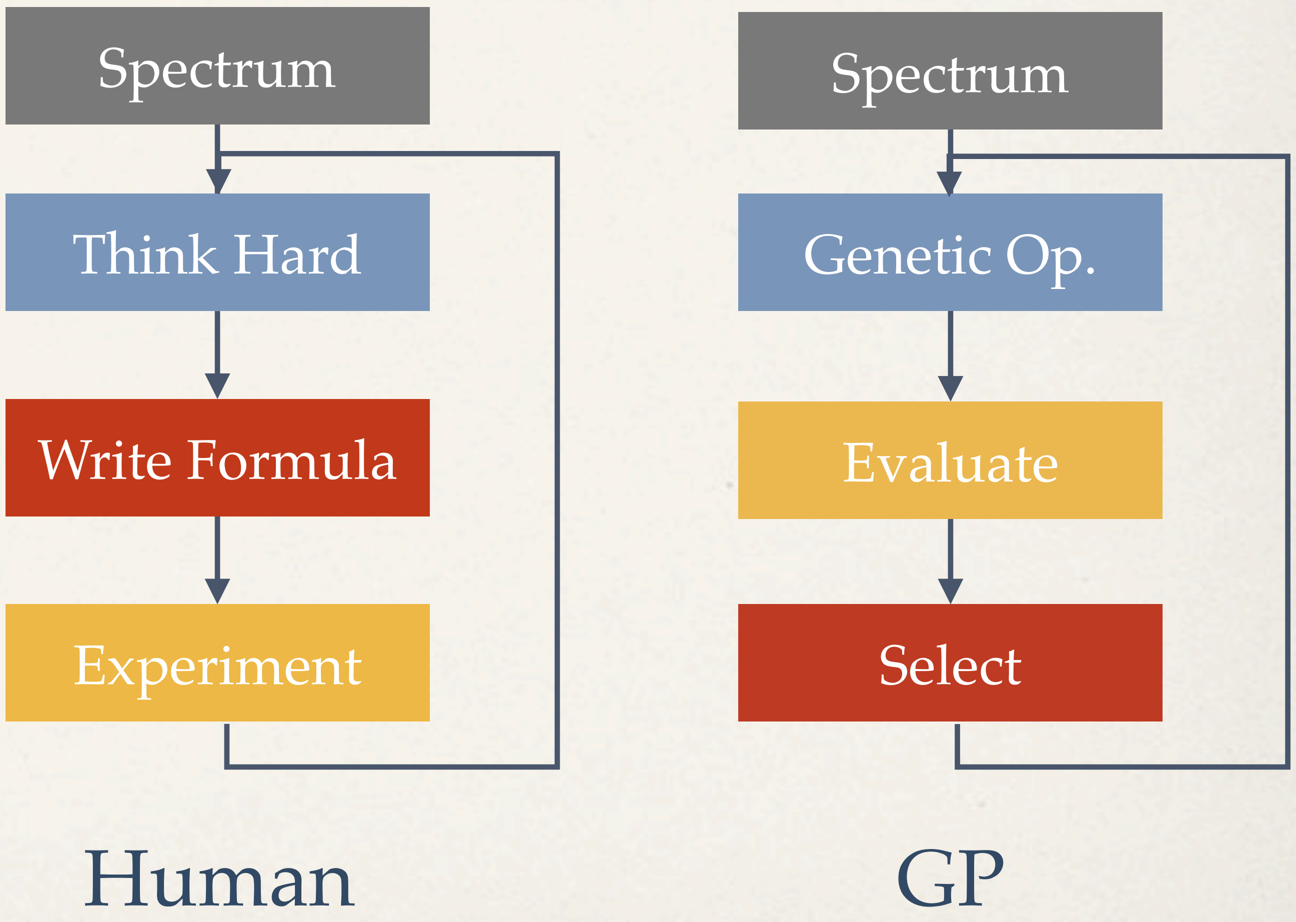
The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.



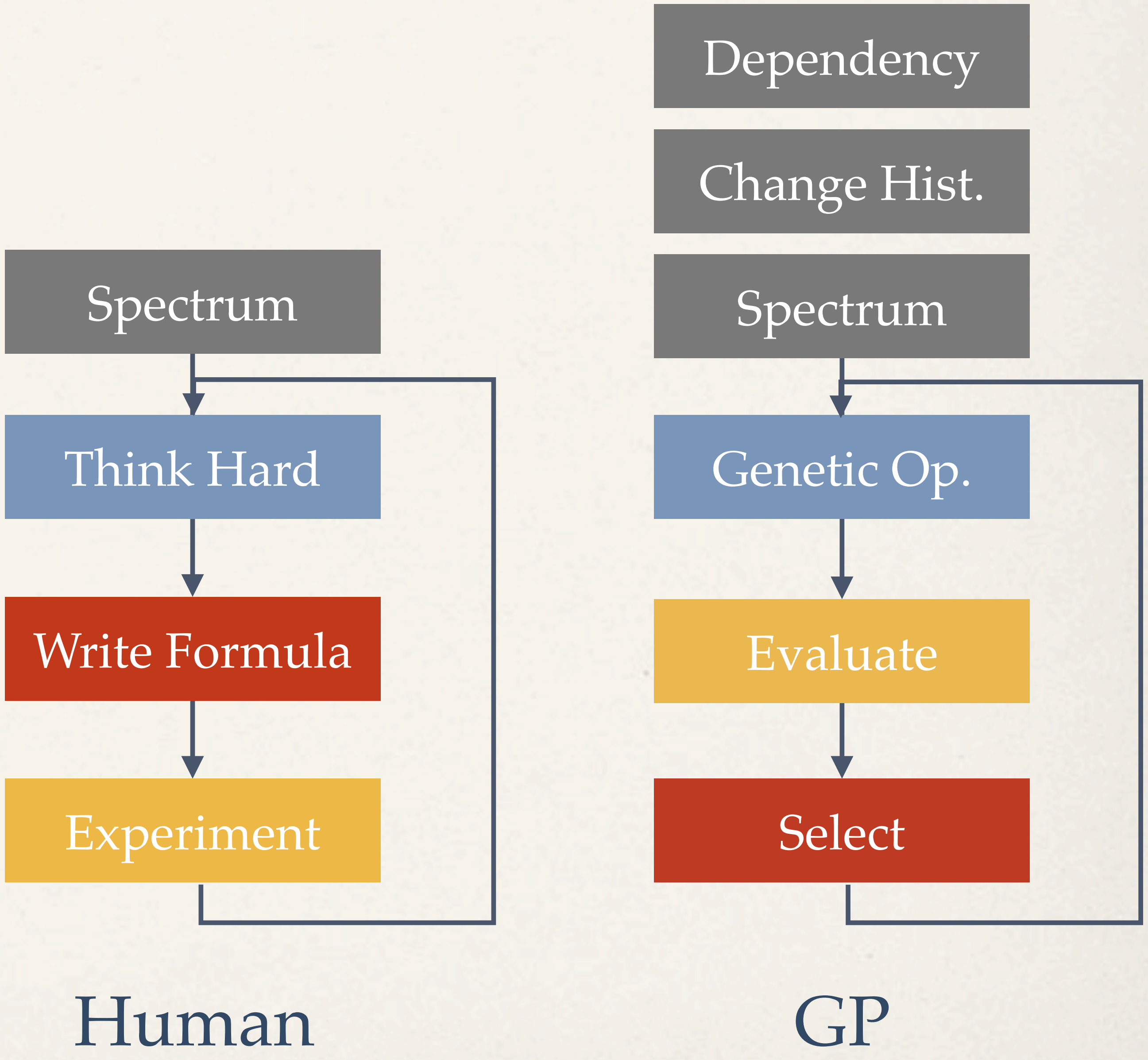
The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.



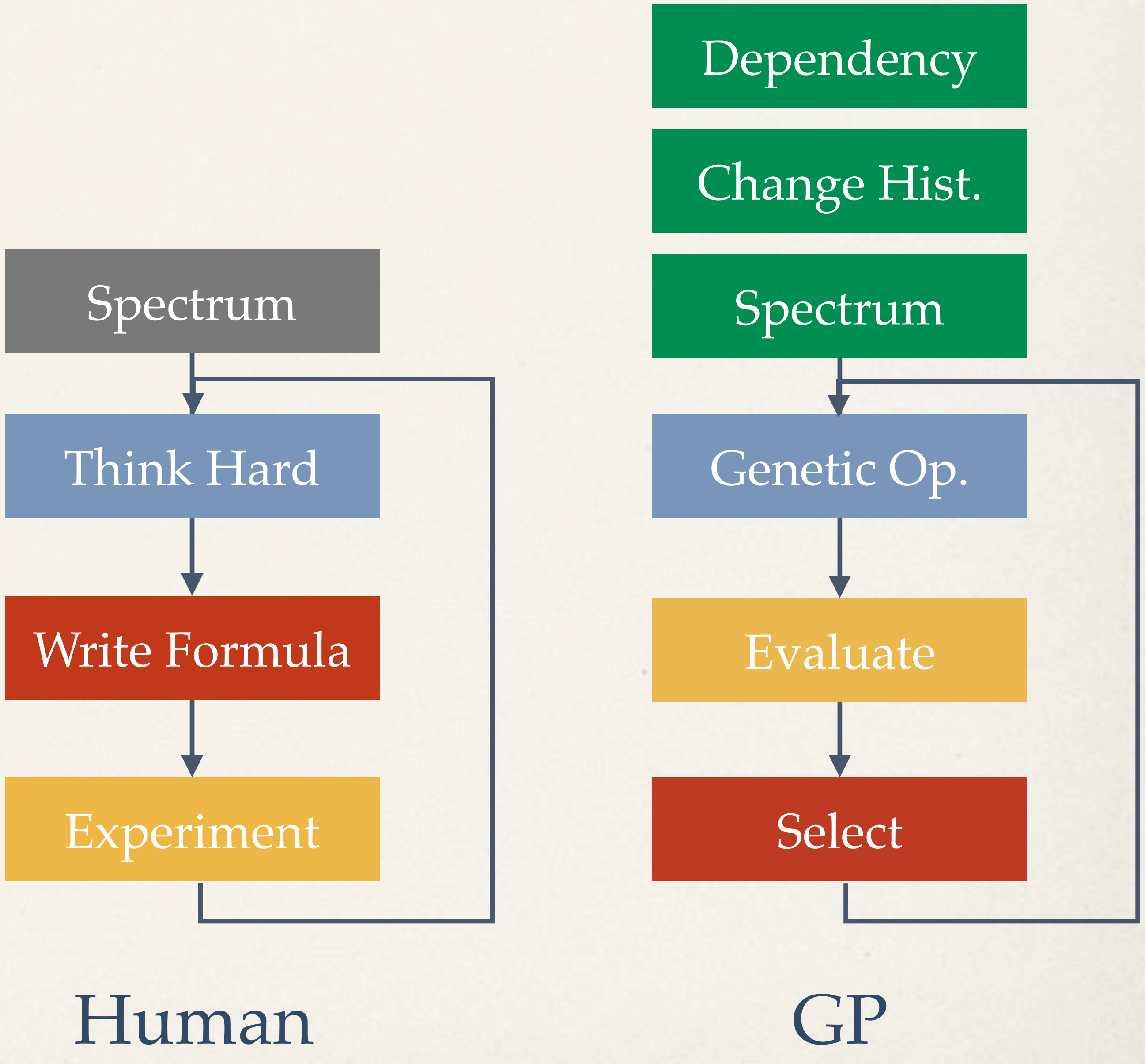
The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.

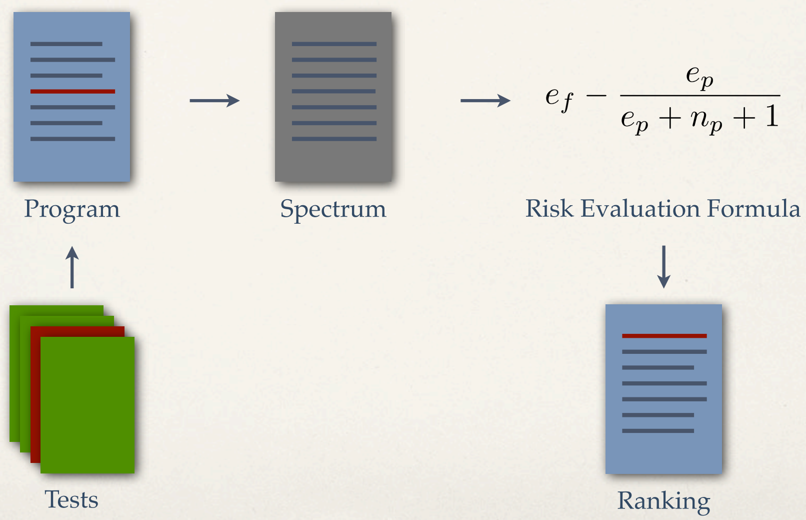


The most effective way to do it, is to do it.

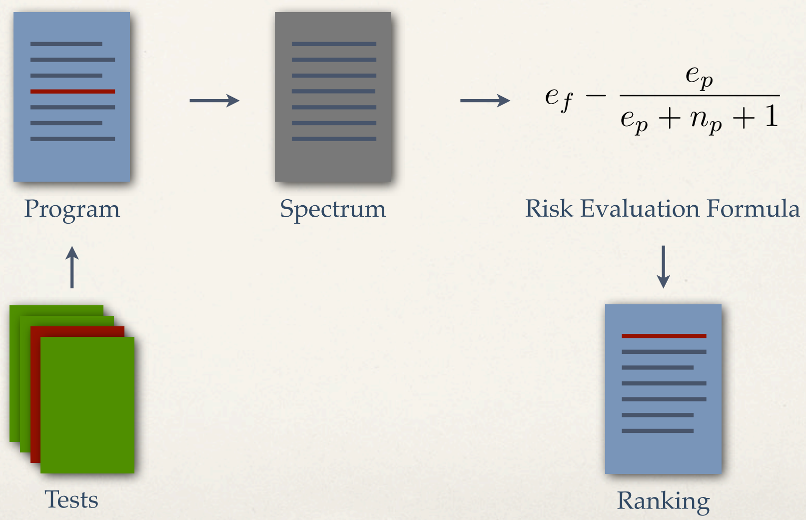
- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.



Evolving SBFL

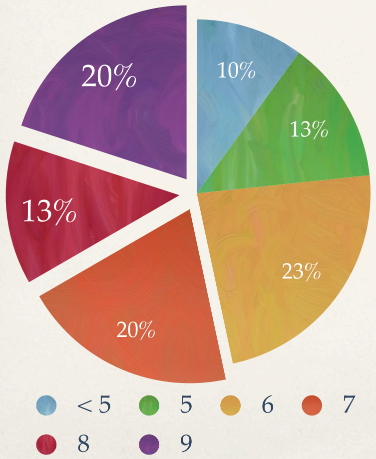


Evolving SBFL



Human Competitiveness

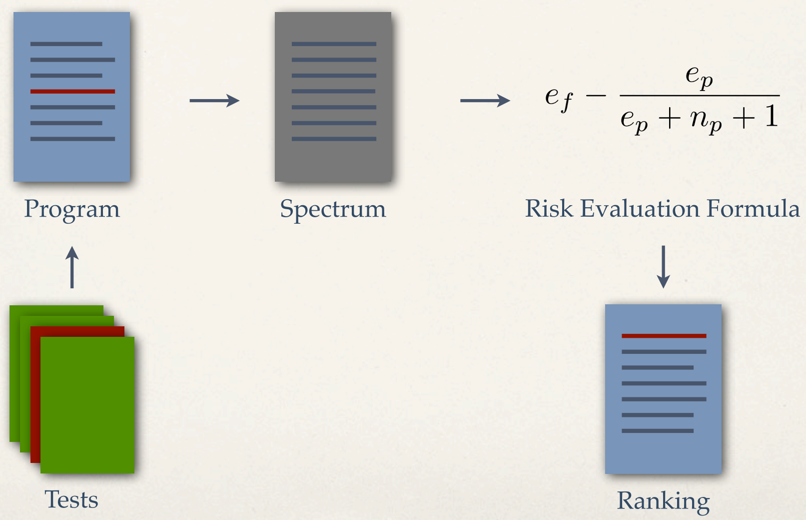
How many of 9 Existing Techniques
can 30 GP runs match and / or outperform?



- 6 runs outperform 8 existing techniques and match / outperform one of the state of the art with proof (Op1 and Op2).
- 16 runs outperform all 7 existing techniques without proof.

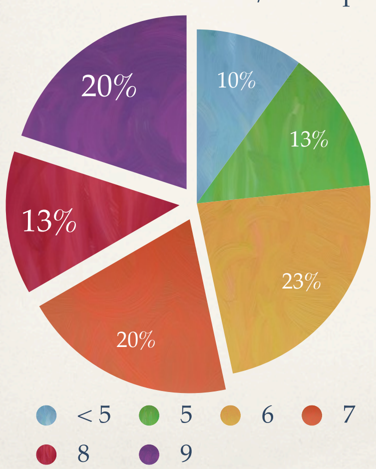
Four Unix tools with 92 faults: 20 random faults for training, 72 for evaluation.

Evolving SBFL



Human Competitiveness

How many of 9 Existing Techniques can 30 GP runs match and / or outperform?

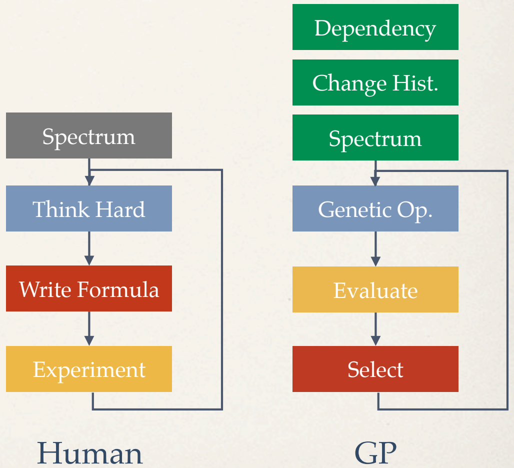


- 6 runs outperform 8 existing techniques and match / outperform one of the state of the art with proof (Op1 and Op2).
- 16 runs outperform all 7 existing techniques without proof.

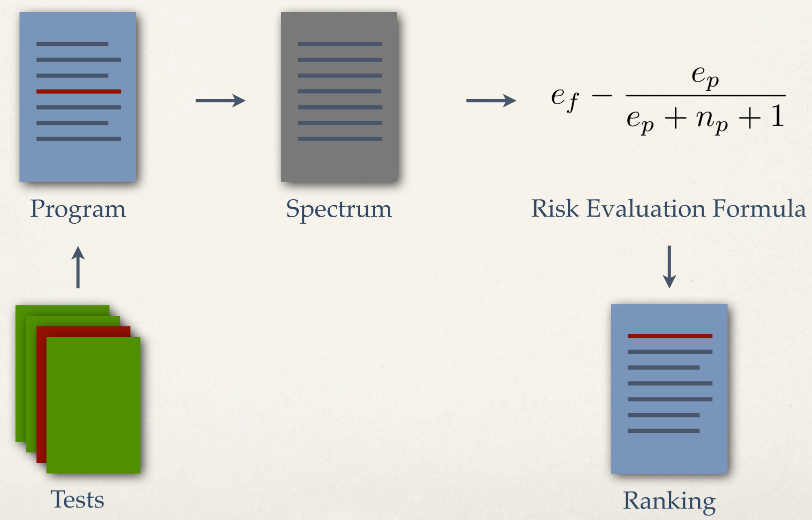
Four Unix tools with 92 faults: 20 random faults for training, 72 for evaluation.

The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.

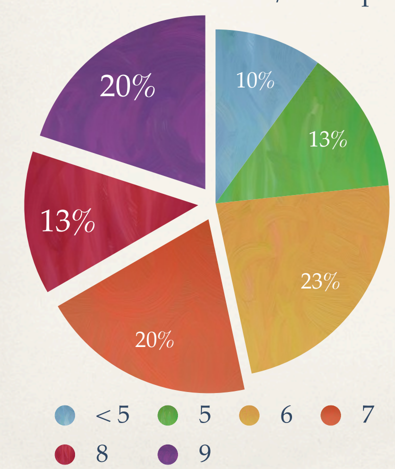


Evolving SBFL



Human Competitiveness

How many of 9 Existing Techniques
can 30 GP runs match and/or outperform?

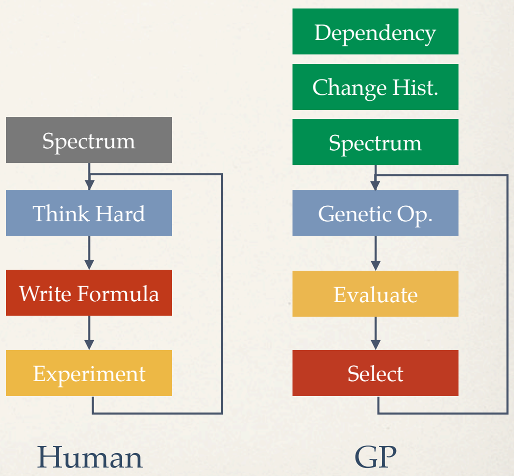


- 6 runs outperform 8 existing techniques and match/outperform one of the state of the art with proof (Op1 and Op2).
- 16 runs outperform all 7 existing techniques without proof.

Four Unix tools with 92 faults: 20 random faults for training, 72 for evaluation.

The most effective way to do it, is to do it.

- GP provides a structured, automated way of doing iterative design.
- It can cope with a much diverse spectra and other meta-data.
- GP can evolve to suit **your project**.



Detailed Statistics & Spectra Data

http://www.cs.ucl.ac.uk/staff/s.yoo/evolving-sbfl.html