



Evolving Fault Localisation

Shin Yoo, University College London, UK

Human Competitive Award, GECCO 2012, 09 July 2012

Spectra Based Fault Localisation

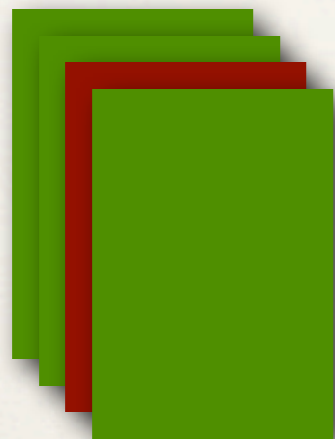


Program

Spectra Based Fault Localisation

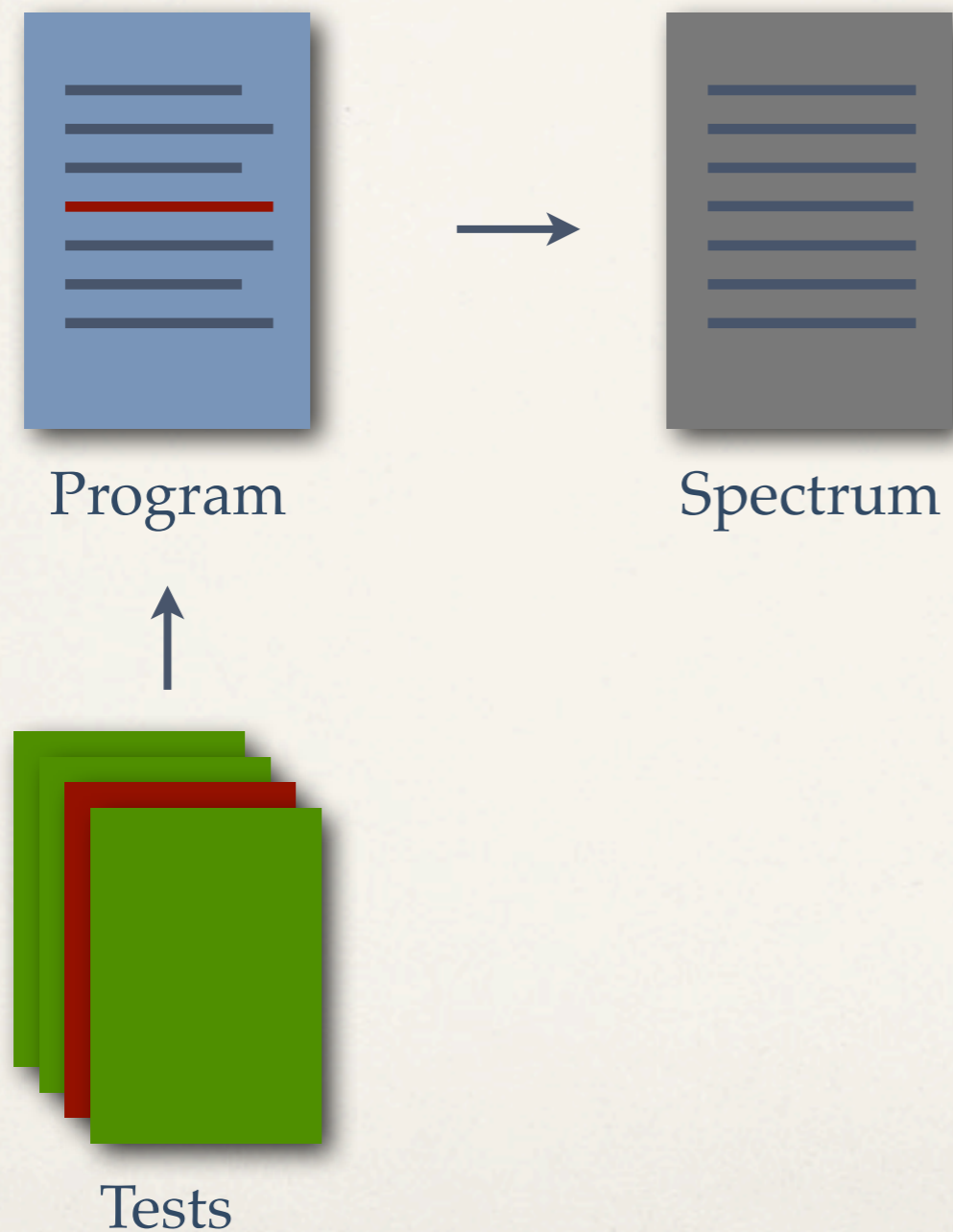


Program

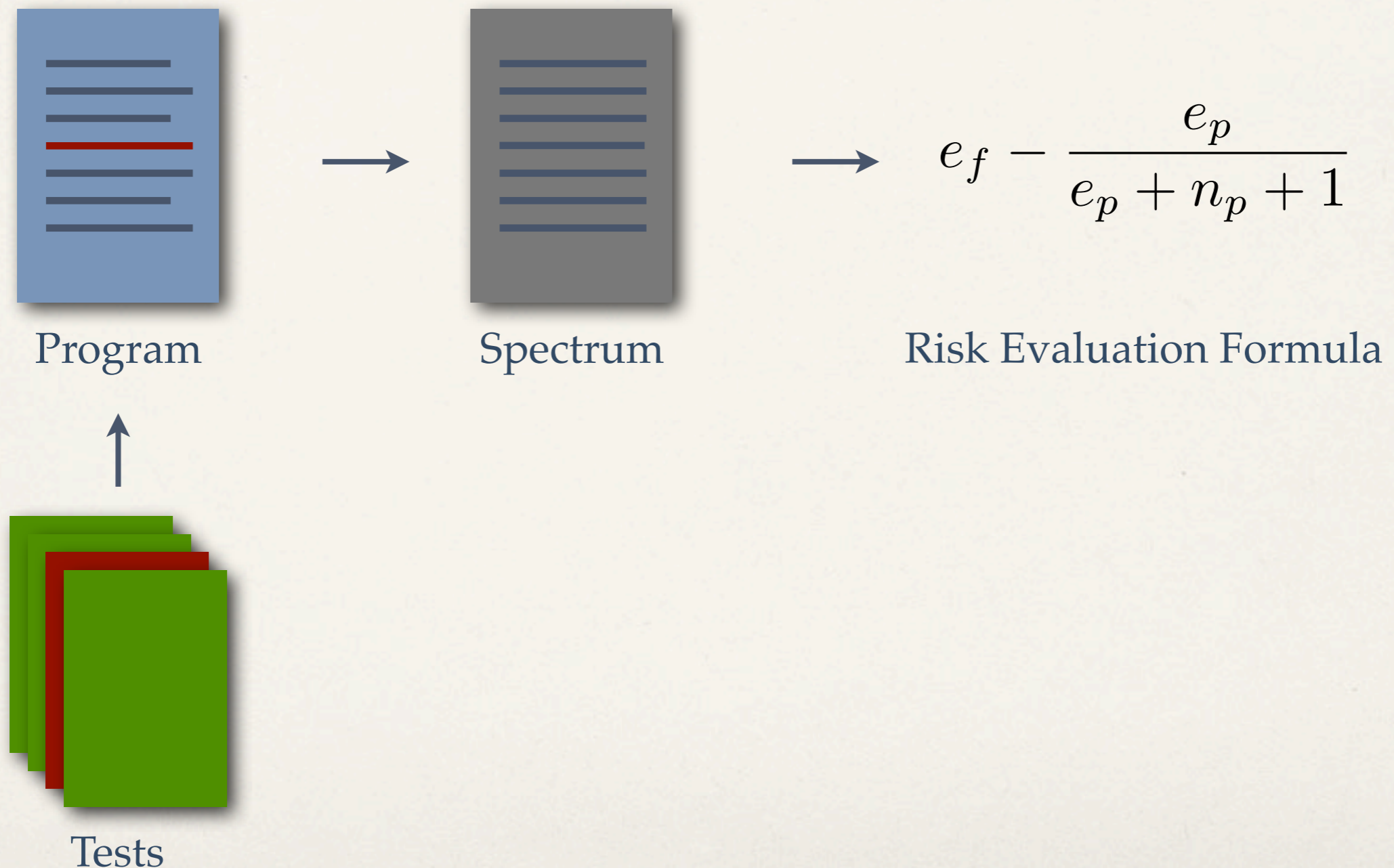


Tests

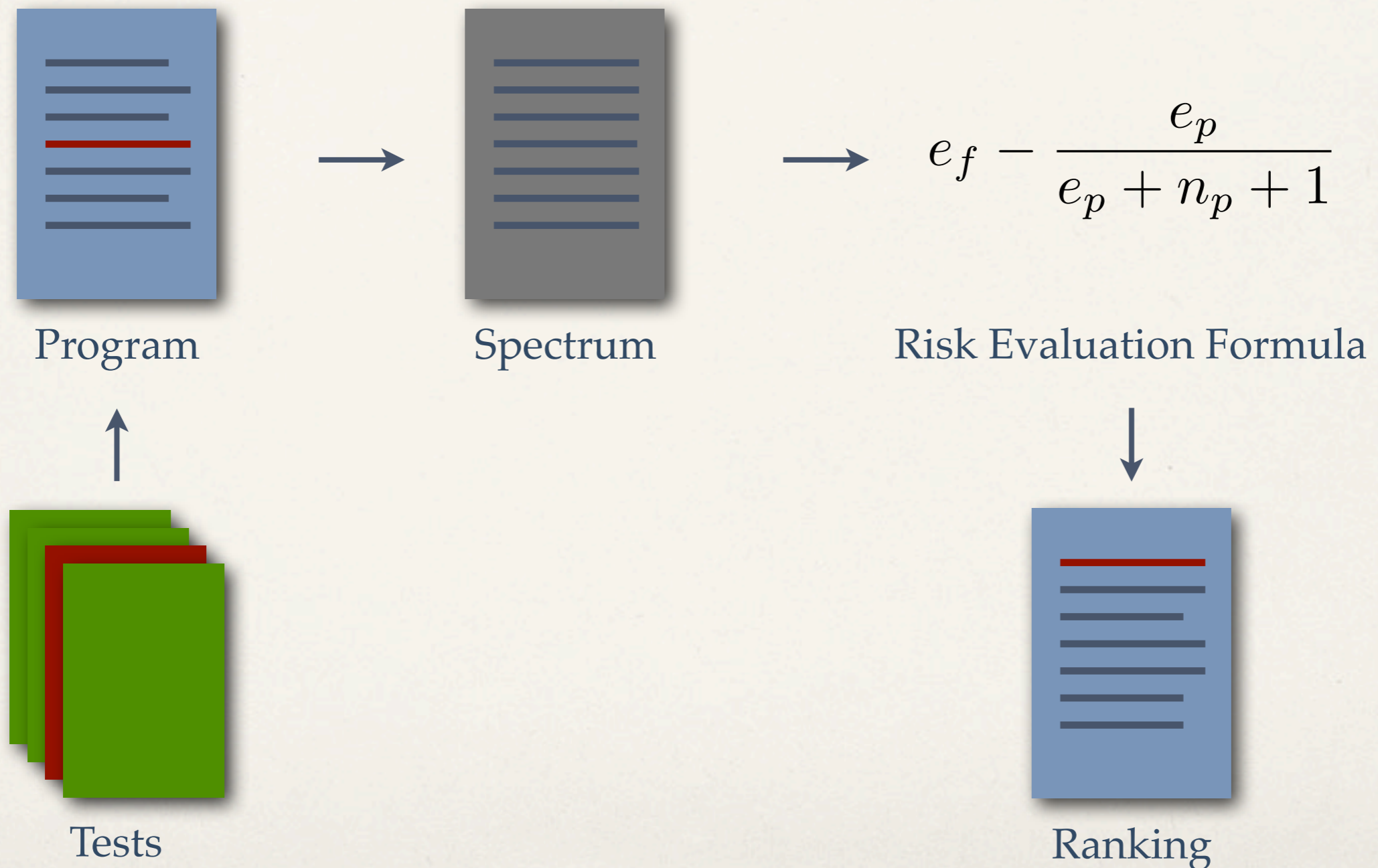
Spectra Based Fault Localisation



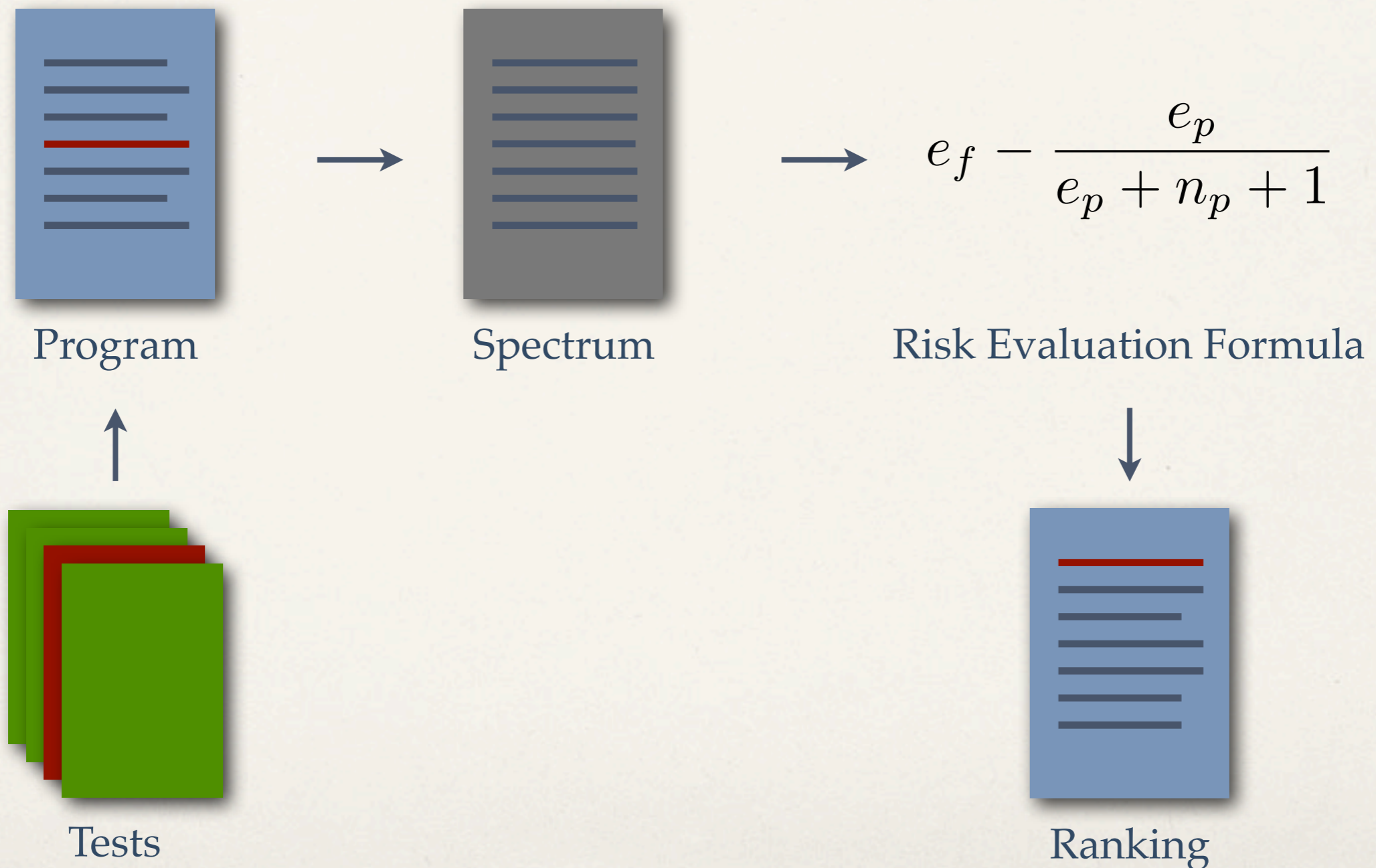
Spectra Based Fault Localisation



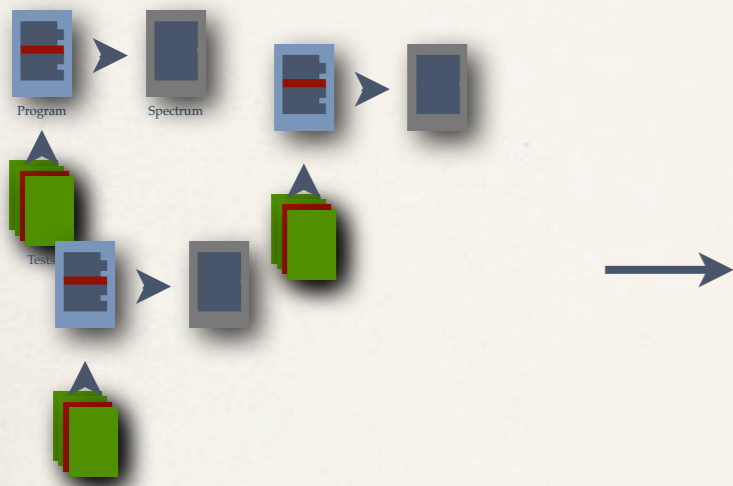
Spectra Based Fault Localisation



Evolving SBFL

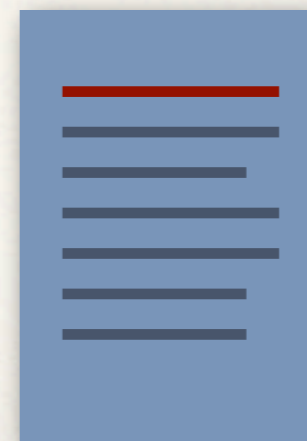


Evolving SBFL



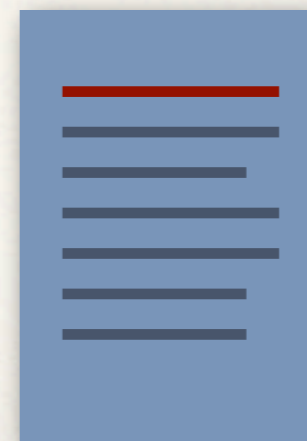
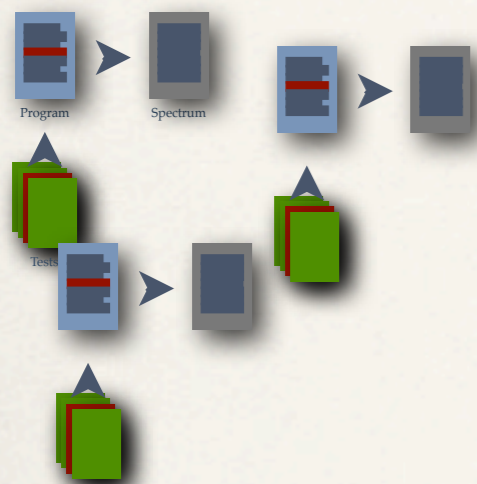
$$e_f = \frac{e_p}{e_p + n_p + 1}$$

Risk Evaluation Formula



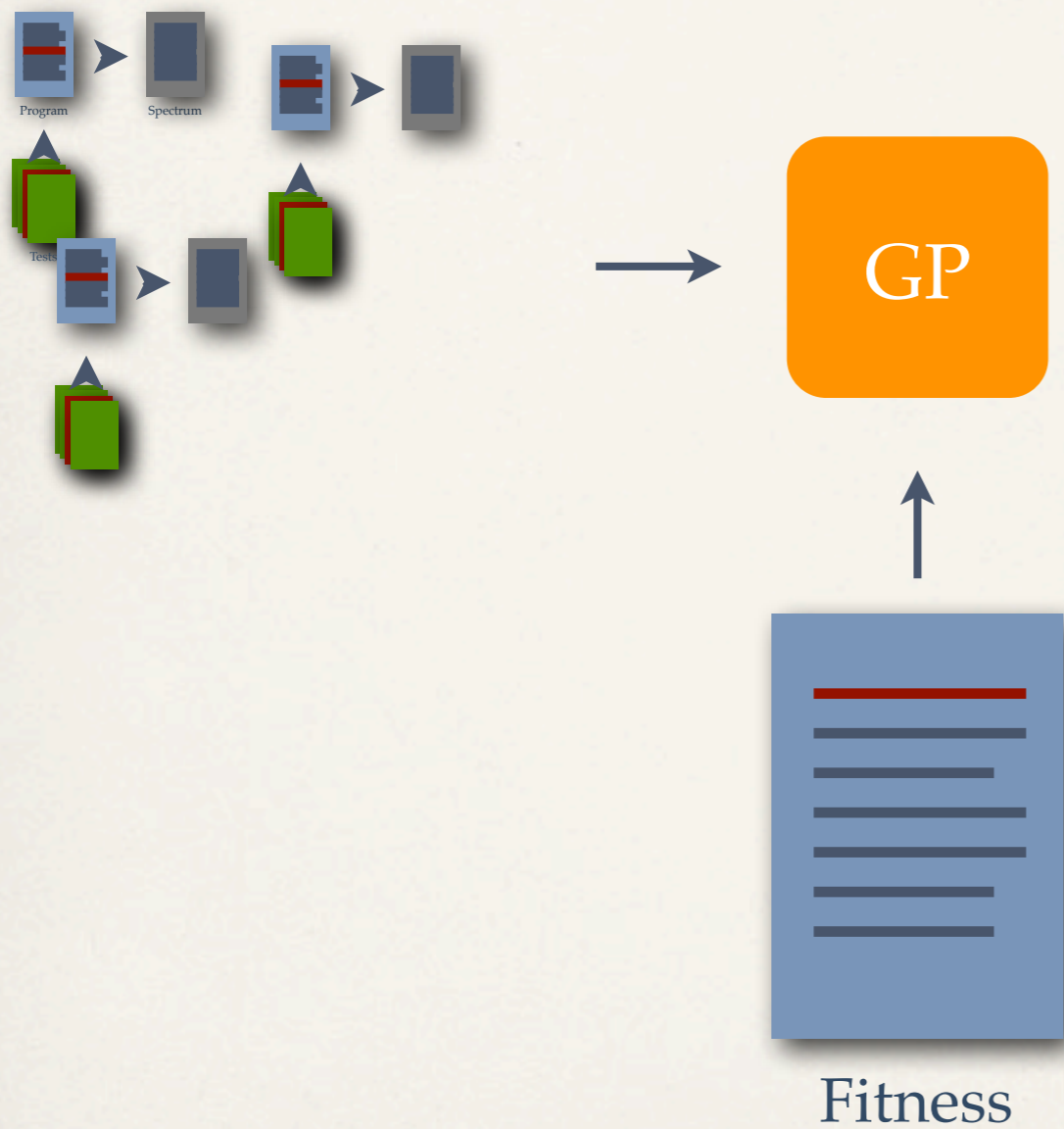
Ranking

Evolving SBFL

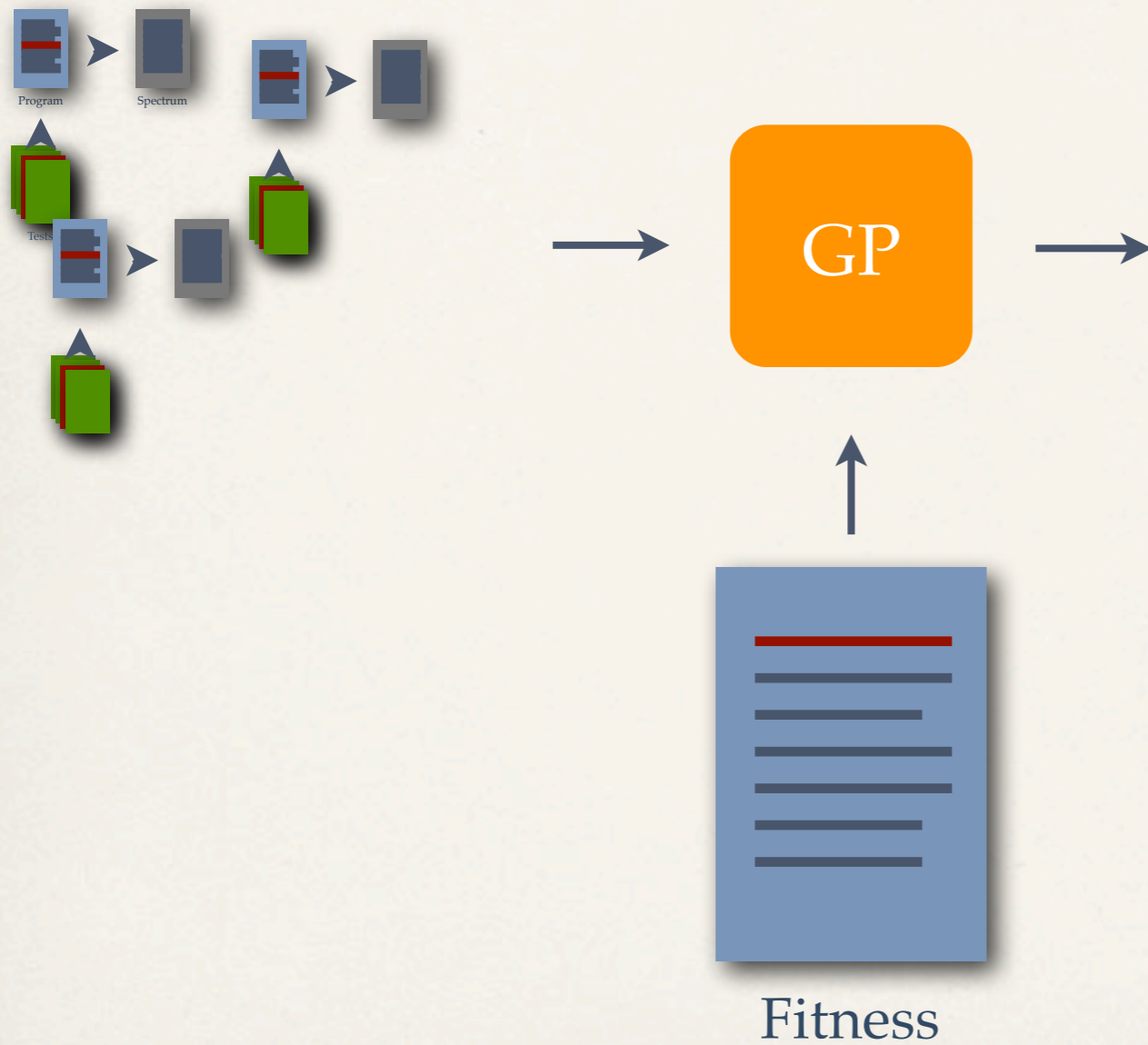


Ranking

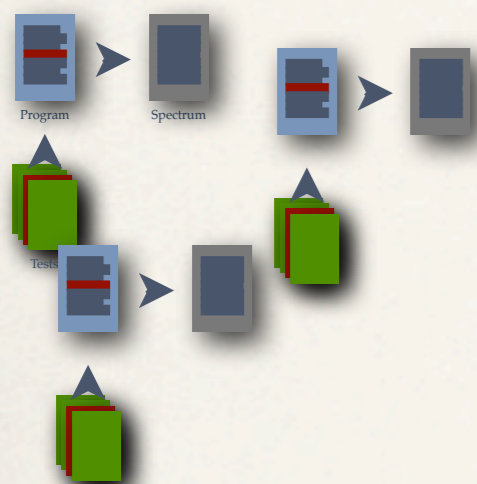
Evolving SBFL



Evolving SBFL



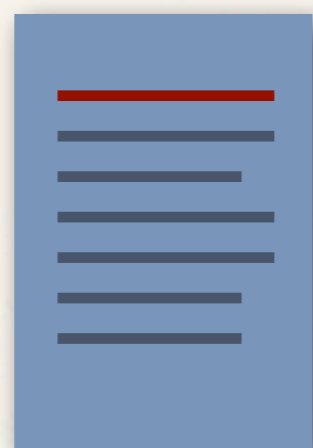
Evolving SBFL



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

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

...



Fitness

State of the Art

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

State of the Art

$$\begin{array}{ccc}
 \frac{e_f}{e_f + n_f + e_p} & \frac{\frac{2e_f}{e_f + n_f + e_p}}{2(e_f + n_p) + e_p + n_f} & \frac{e_f}{e_f + n_p + 2(e_p + n_f)} \\
 \frac{e_f}{n_f + e_p} & \text{Over 30 formulæ in the literature, with various} & \frac{e_f}{e_f + 2(n_f + e_p)} \\
 & \text{empirical studies with slightly different results} & \frac{e_f + n_p}{n_f + e_p} \\
 \frac{e_f}{e_f + n_f + e_p + n_p} & \frac{e_f + n_p}{e_f + n_f + e_p + n_p} & \frac{2e_f}{2e_f + n_f + e_p} \\
 \frac{1}{2} \left(\frac{e_f}{e_f + n_f} + \frac{e_f}{e_f + e_p} \right) & \frac{\frac{e_f}{e_f + n_f}}{\frac{e_p}{e_p + n_p} + \frac{e_f}{e_f + n_f}} & \frac{e_f + n_p - n_f - e_p}{e_f + n_f + e_p + n_p}
 \end{array}$$

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 outperforms 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. Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2009 ACM 0004-5411/2009/0205-0001 \$5.00

Journal of the ACM, Vol. V, No. N, June 2009, Pages 1–37.

Optimality Proof (Naish et al. 2011)

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

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 outperforms 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. Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2009 ACM 0004-5411/2009/0205-0001 \$5.00

Journal of the ACM, Vol. V, No. N, June 2009, Pages 1–37.

Optimality Proof (Naish et al. 2011)

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

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

Table III. Definitions of new ranking metrics used

Name	Formula	Name	Formula
O	-1 if $a_{n_f} > 0$, otherwise a_{n_p}	Op	$a_{n_f} - \frac{a_{n_p}}{a_{n_f} + a_{n_p} + 1}$
Binary	0 if $a_{n_f} > 0$, otherwise 1	Ampld	$\frac{a_{n_f}}{a_{n_f} + a_{n_p}} - \frac{a_{n_p}}{a_{n_f} + a_{n_p}}$
CHI lin	$\frac{a_{n_f}}{a_{n_f} + a_{n_p}} - \frac{a_{n_f} + a_{n_p}}{a_{n_f} + a_{n_p} + a_{n_f} + a_{n_p}}$	CHI Log	$\frac{2}{\frac{a_{n_f} + a_{n_p}}{a_{n_f} + a_{n_p} + 1} + \frac{a_{n_f} + a_{n_p}}{a_{n_f} + a_{n_p} + 1}}$
CHI Sqrt	$\frac{2}{\frac{a_{n_f} + a_{n_p}}{a_{n_f} + a_{n_p} + 1} + \sqrt{\frac{a_{n_f} + a_{n_p}}{a_{n_f} + a_{n_p} + 1}}}$	WongP	$\begin{cases} -1000 & \text{if } a_{n_p} + a_{n_f} = 0 \\ \text{if } \text{Ampld} & \text{otherwise} \end{cases}$

O is the simplest optimal metric from an information-theoretic perspective as it only gives different ranks when necessary. Metrics can also be considered from a geometrical perspective — each metric defines a surface in three dimensions (the four a_{ij} values give just two degrees of freedom if we fix the number of passed and failed tests). We propose Op (see below), which defines a very simple surface — a plane. It is optimal for $ITE2_s$ since a_{n_f} is maximal when $a_{n_p} = 0$ and a_{n_p} varies from 0 to at most the number of passed tests, so the fractional component is strictly less than one.

Definition 6.6 Ranking metric Op .

$$Op(a_{n_p}, a_{n_f}, a_{n_p}, a_{n_f}) = a_{n_f} - \frac{a_{n_p}}{P+1}$$

where P is the number of passed test cases.

Op has the advantage of performing more rationally than O for multiple-bug programs. If there is more than one bug, a_{n_f} can be non-zero for all statements, leading to O being -1 in all cases. In contrast, Op ranks statements first on their a_{n_f} value and, even if this is not maximal, second on their a_{n_p} value. Op and other optimal metrics can be helpful in comparison of metrics (see section 7.6). We also use Op in our empirical evaluation of metrics (to aid comparison with other work, some multiple bug programs are used). In our experiments we also evaluate the performance of a simplified version of O , called Binary, which ignores a_{n_p} and allows us to see the relative importance of the two components of O . We also include a variation of the Ample metric which avoids taking the absolute value and a variation of the WongP metric which has a special case for statements that are not executed in any test case (motivated by our empirical studies). The definitions of all the new metrics we use are shown in Table III.

Although we have formally proved the optimality for $ITE2_s$ only, O and Op are optimal for a much broader class of single bug programs. Proposition 6.3 holds for all single bug programs and the combinatorial argument in the proof of Lemma 6.4 can be generalised. With larger numbers of paths and/or sources of “noise” it is typically sufficient to show $\forall y \ f(y+1, j+k) f(y, j) \geq f(y, j+k) f(y+1, j)$, where j and k are positive integers dependent on the number of paths through the program with particular characteristics.

Optimality Proof (Naish et al. 2011)

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

But the proof is against a specific model

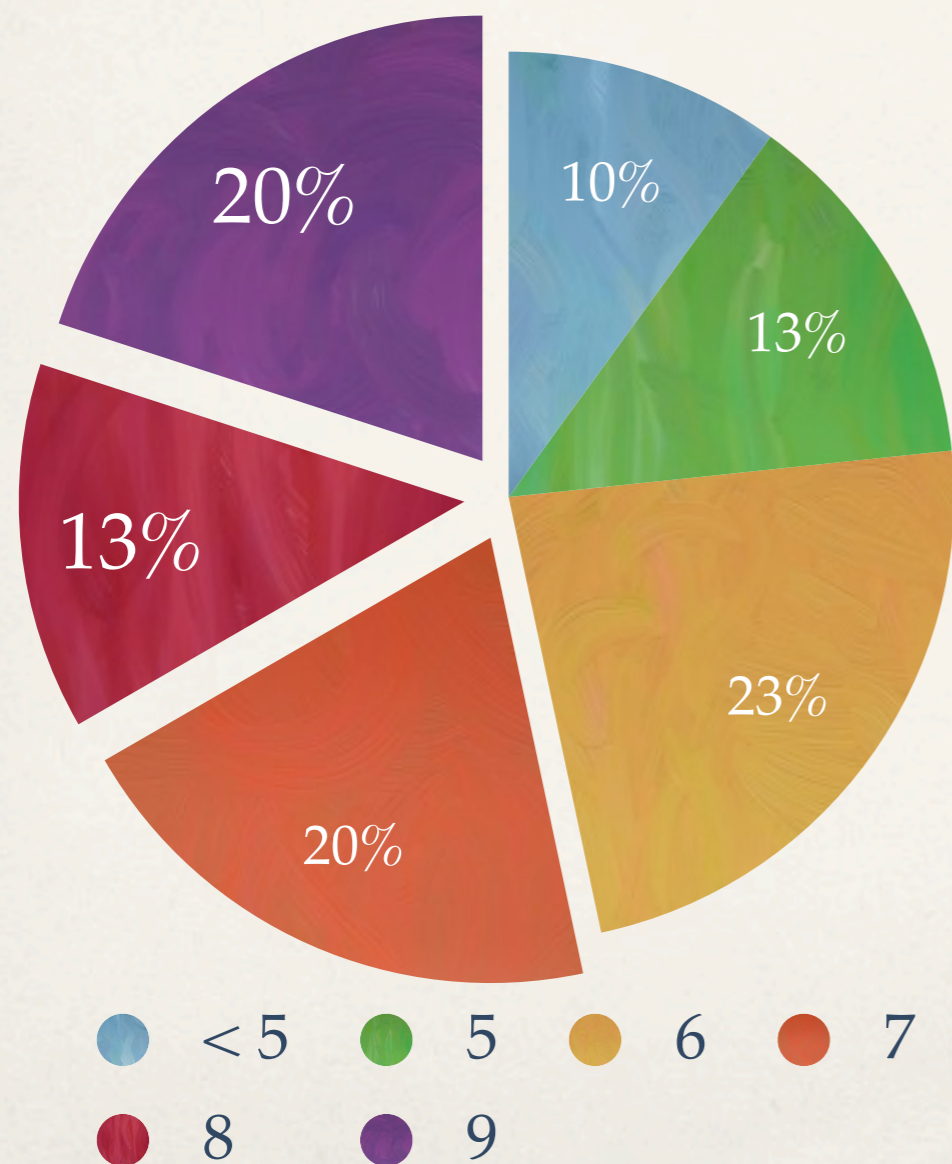
```

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

... not to mention hard.

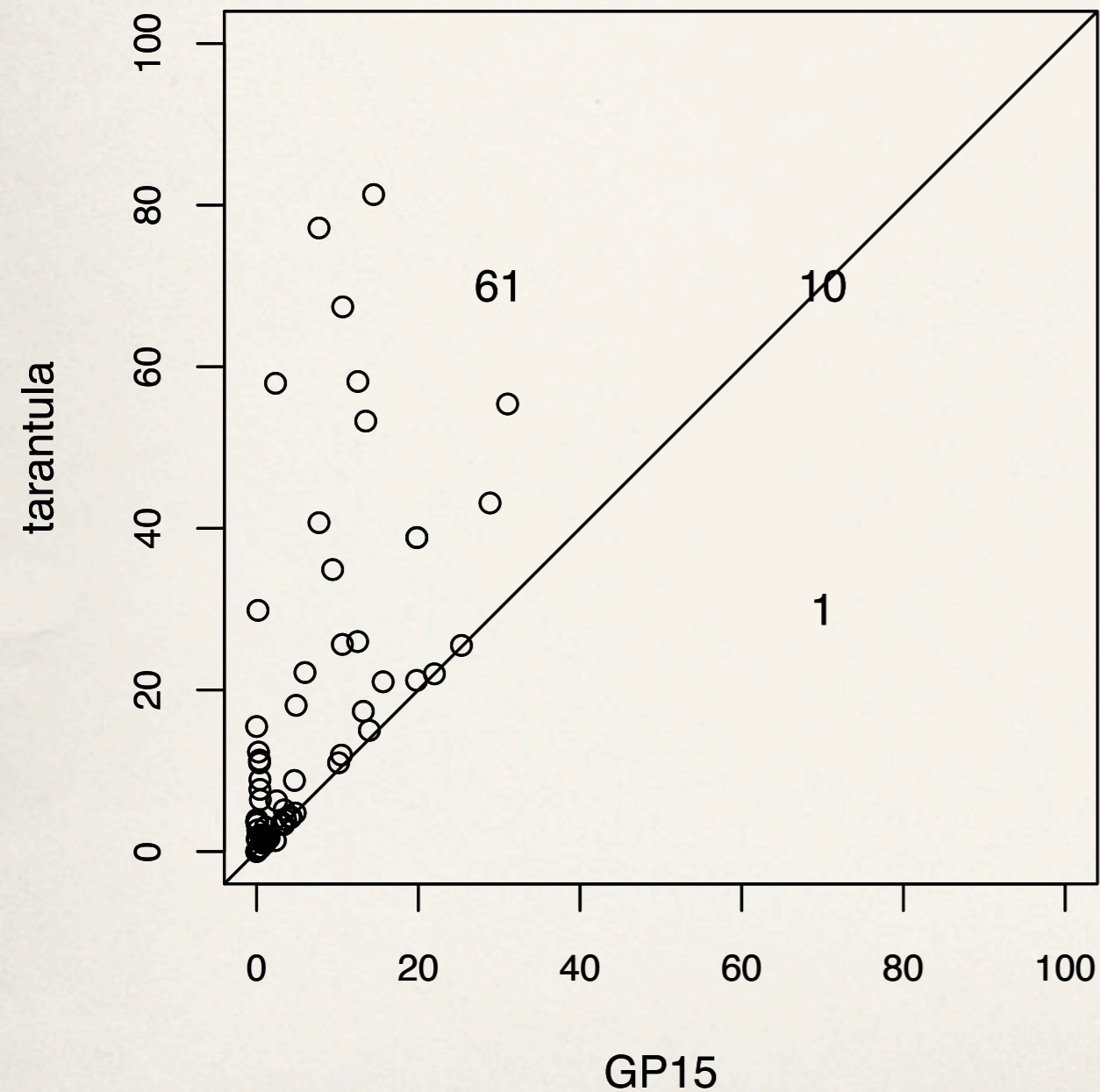
Human Competitiveness

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



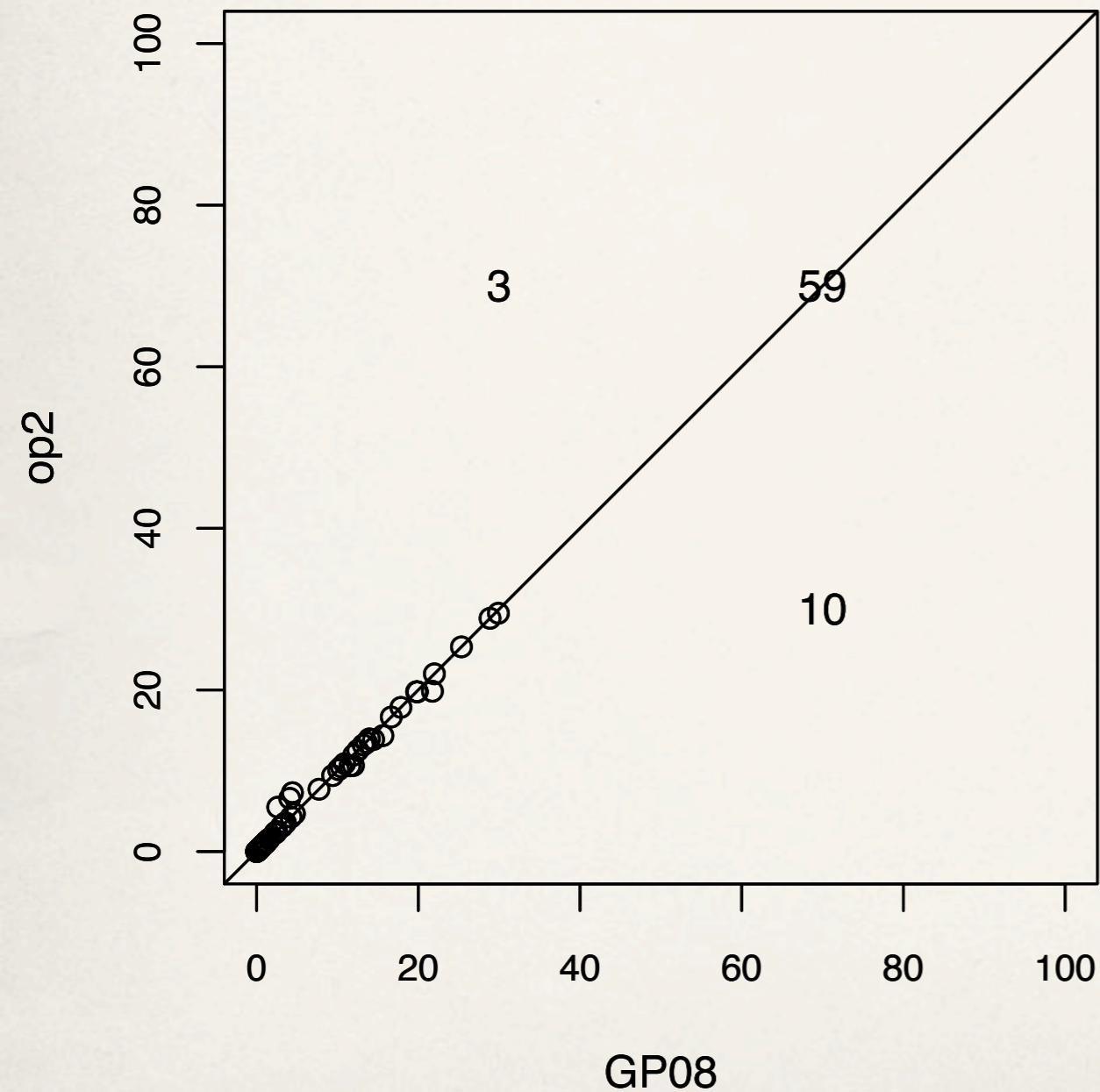
- * 6 runs outperform 8 existing techniques and match/outperform one of the state of the art with proof (Op1 and Op2).
- * 16 runs outperform all 7 existing techniques without proof.

Human Competitiveness



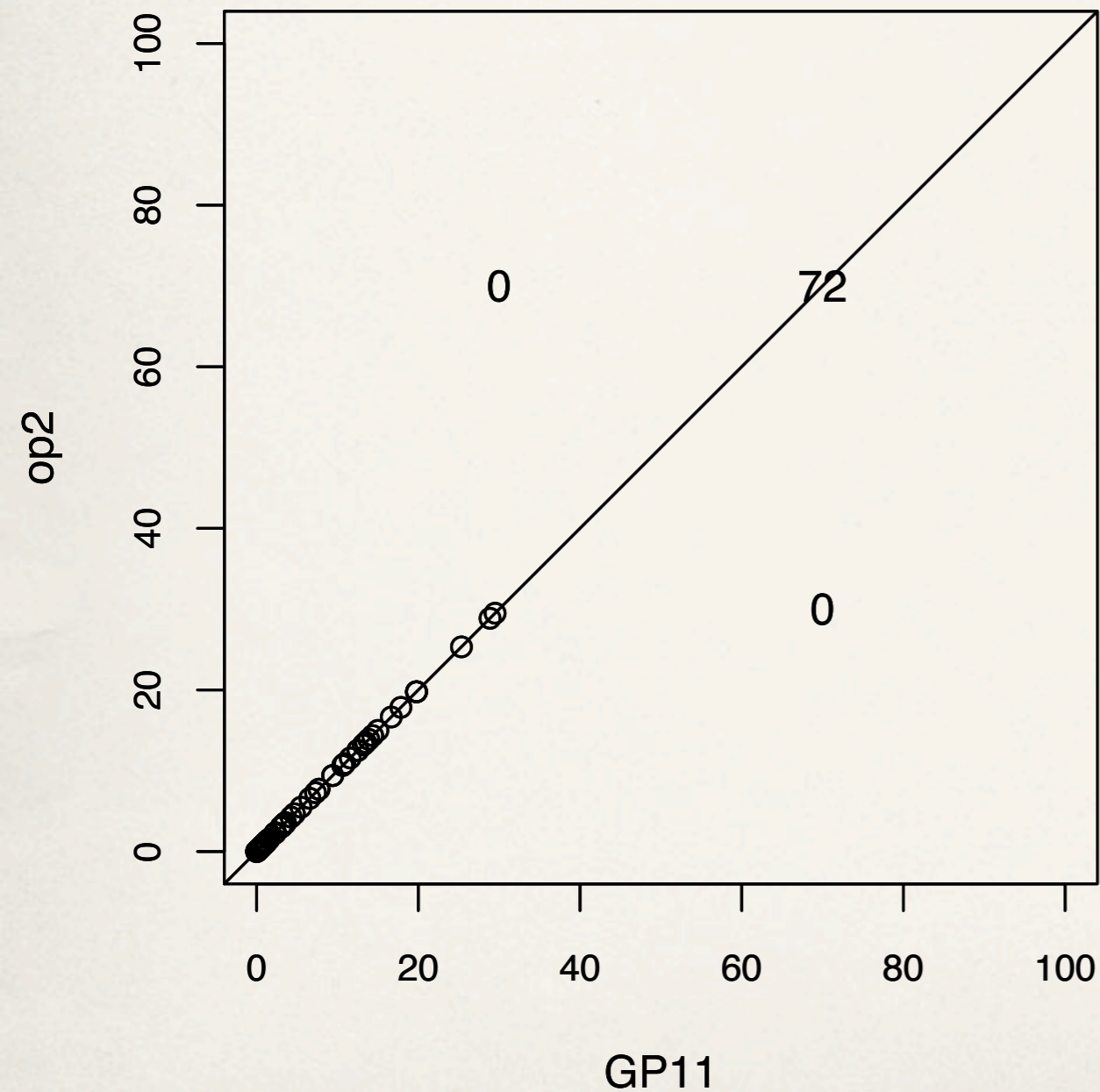
- ✦ Per-fault view shows that evolved techniques can outperform ones with optimality proofs.

Human Competitiveness



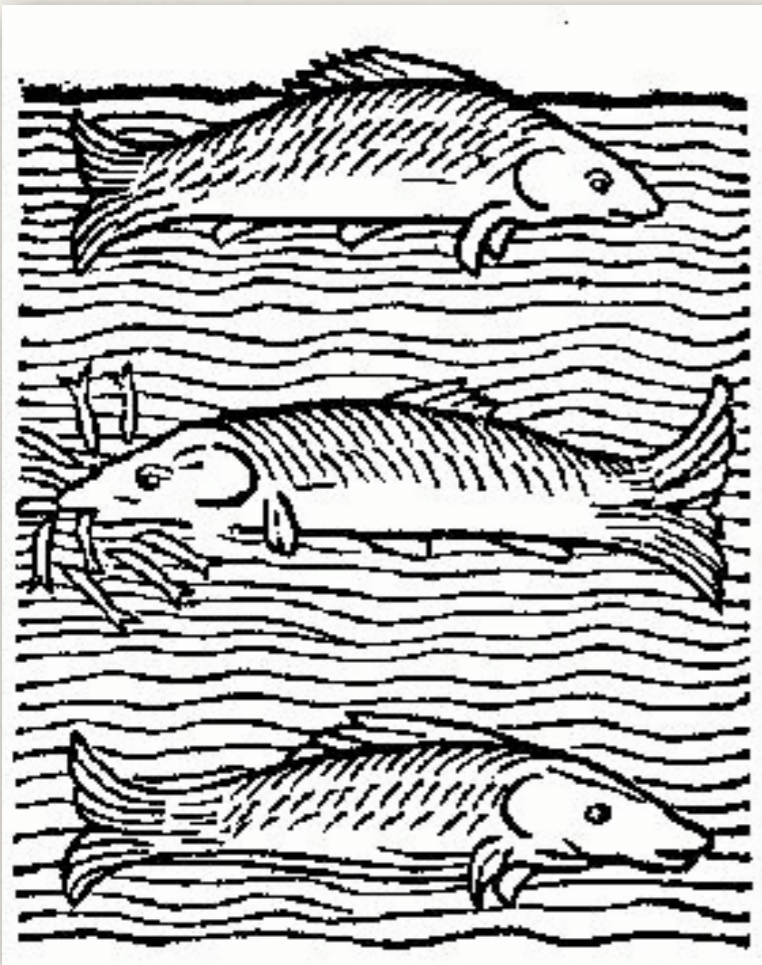
- ✦ Per-fault view shows that evolved techniques can outperform ones with optimality proofs.

Human Competitiveness

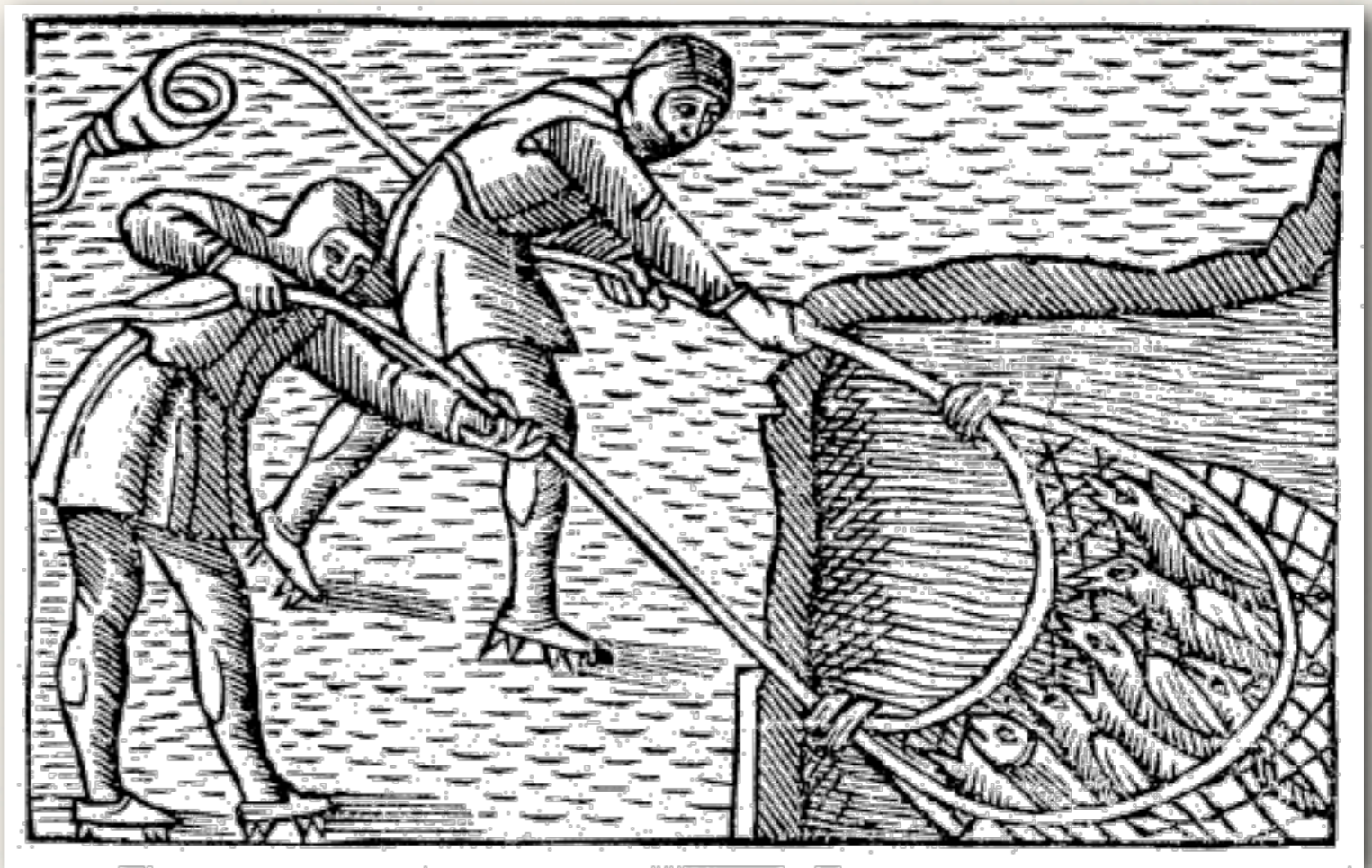


- ✦ Per-fault view shows that evolved techniques can outperform ones with optimality proofs.

Future of Search-Based Software Engineering



From Solutions to
Generic Problems...



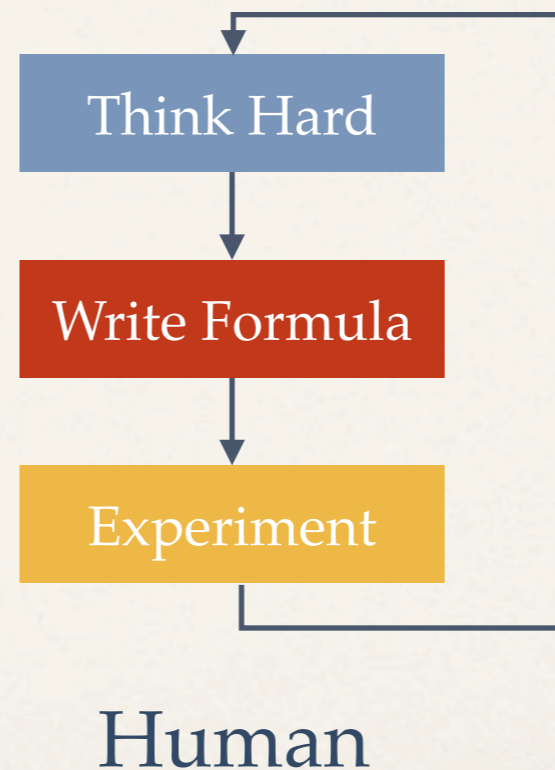
To Techniques and Strategies
for **Your** Problems.

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**.

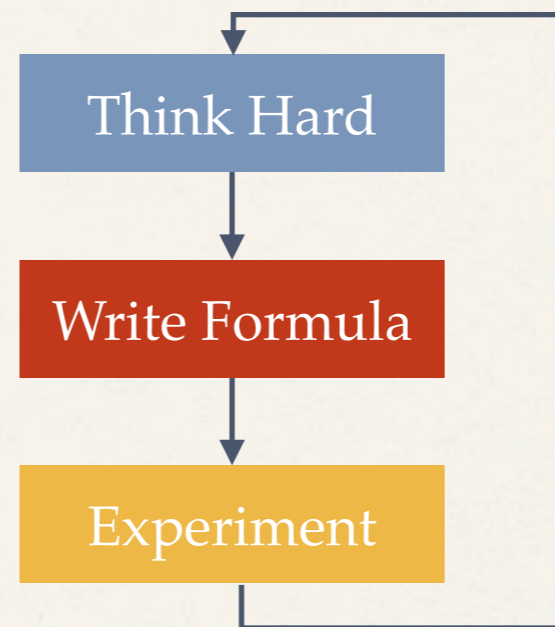
The most effective way to do it, is to do it.

- ❖ 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**.

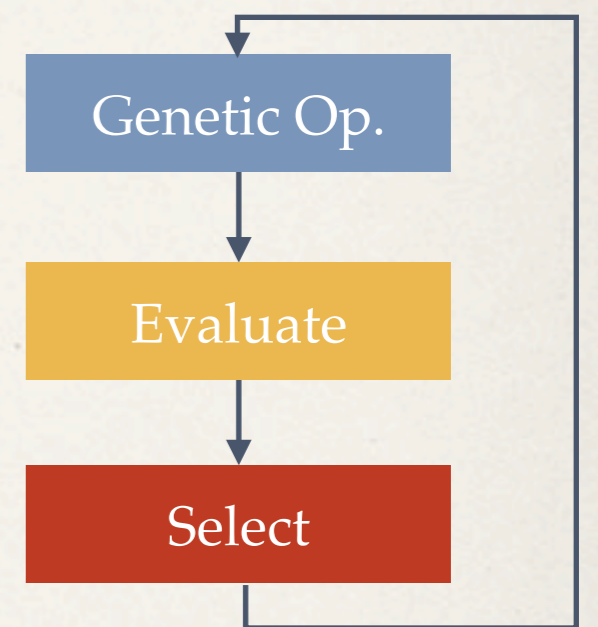


The most effective way to do it, is to do it.

- ❖ 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**.



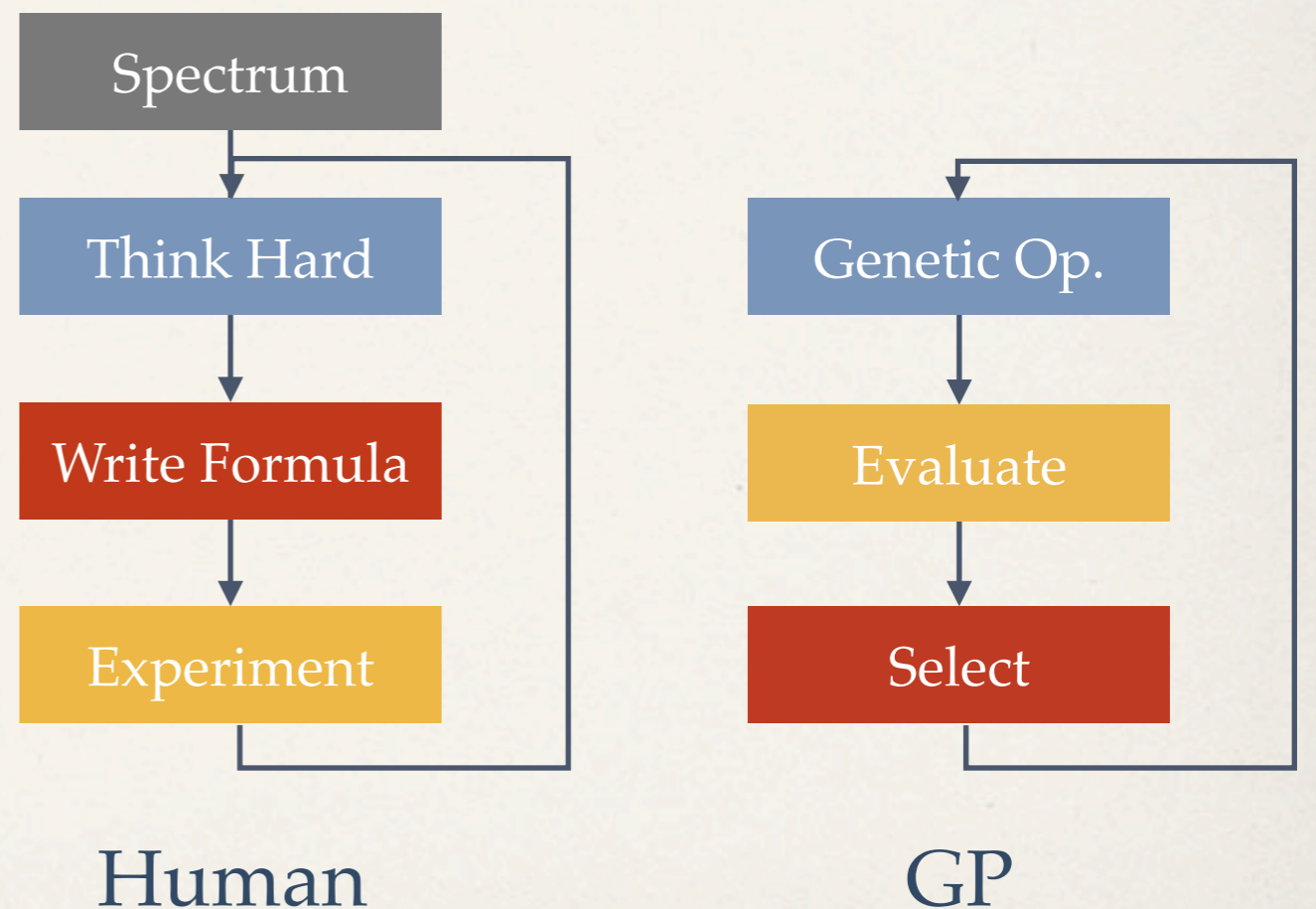
Human



GP

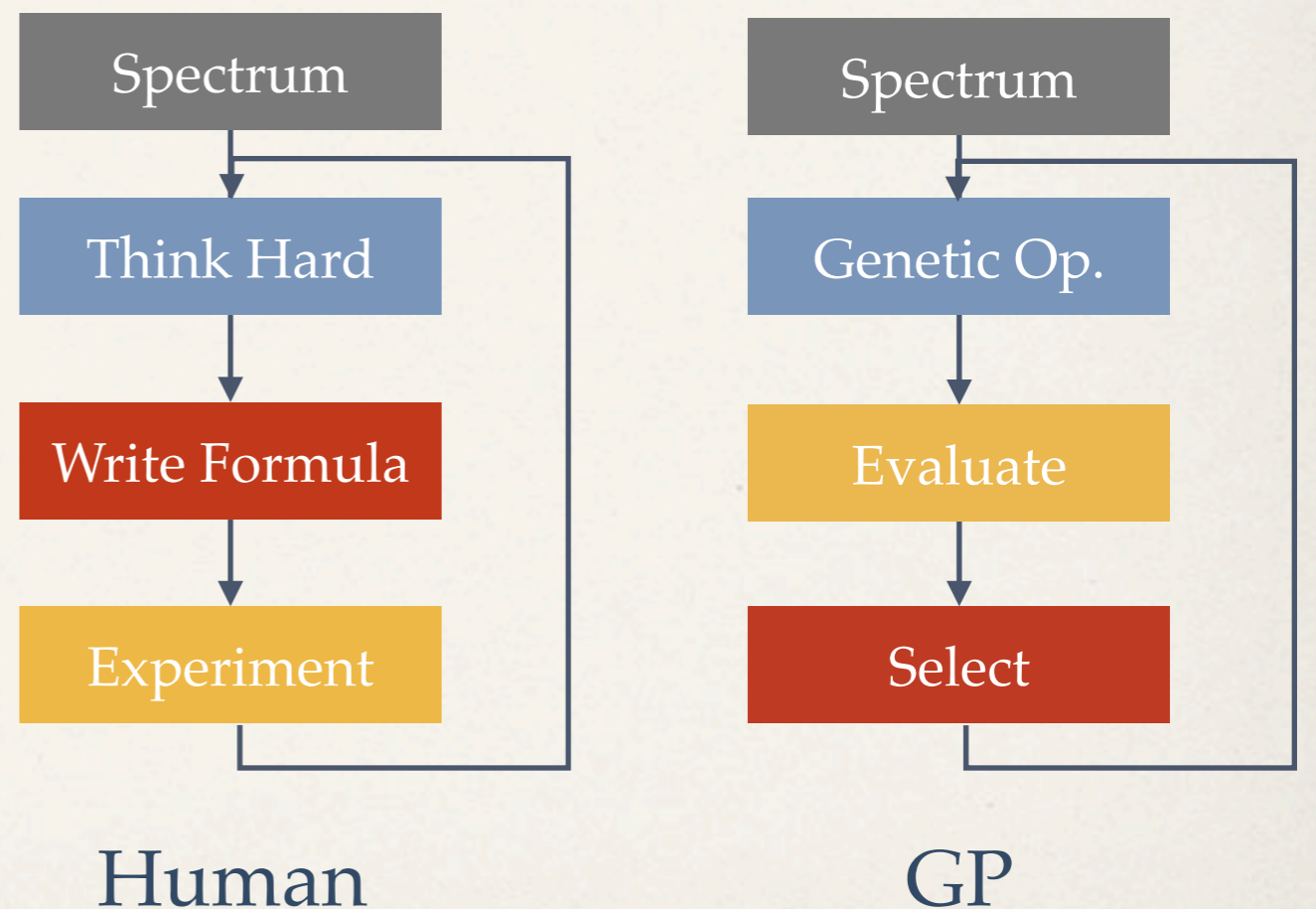
The most effective way to do it, is to do it.

- ❖ 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**.



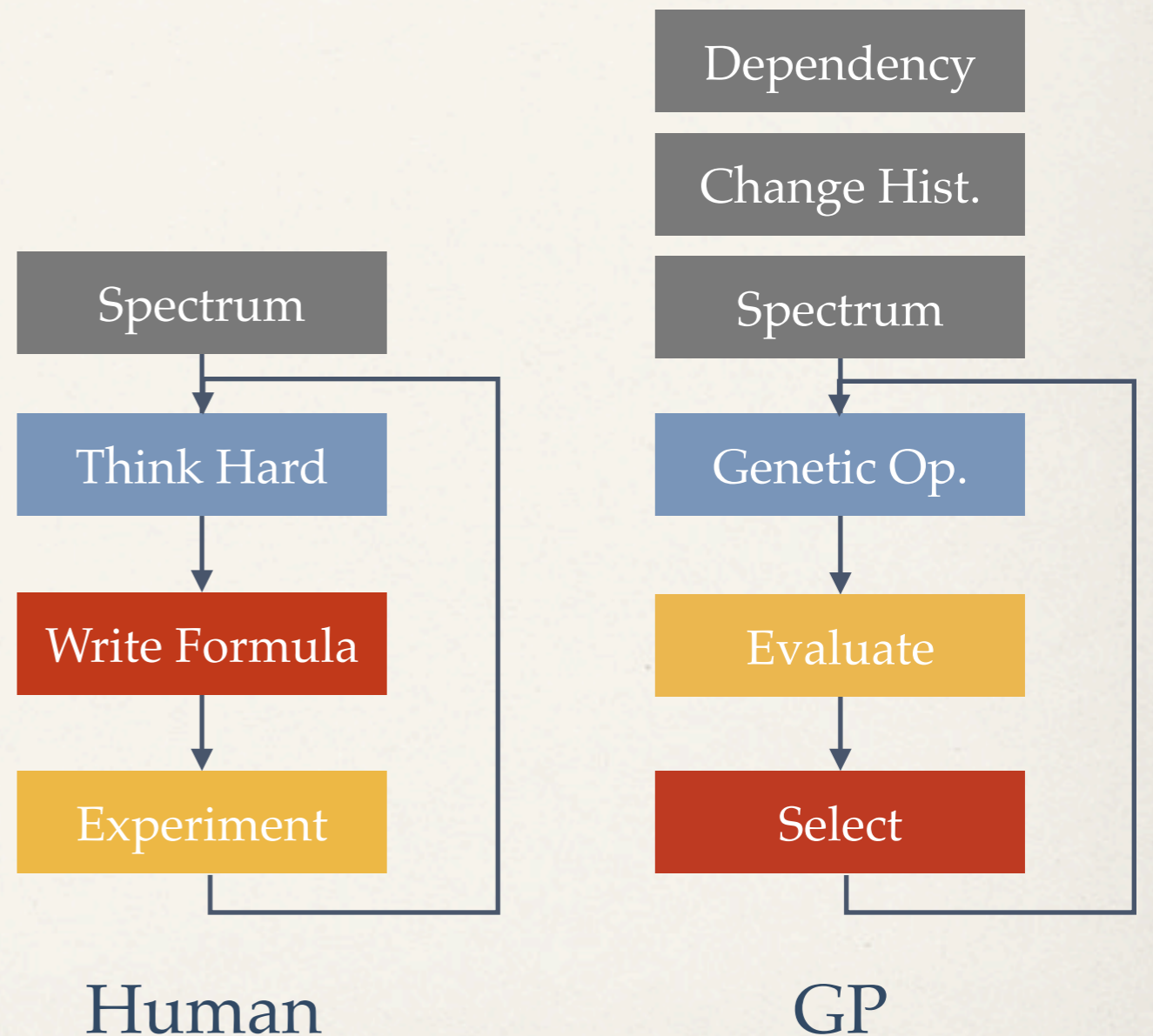
The most effective way to do it, is to do it.

- ❖ 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**.



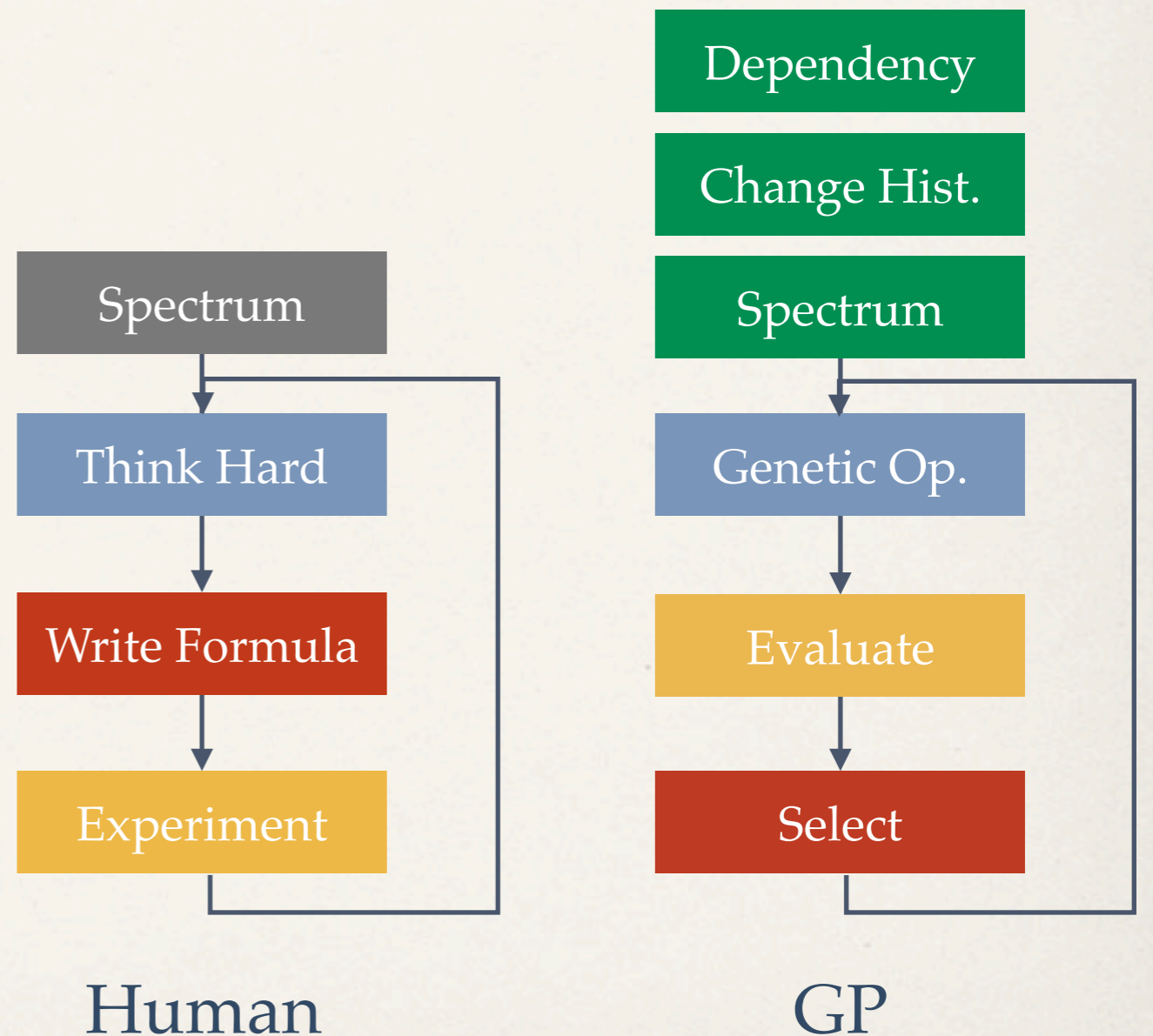
The most effective way to do it, is to do it.

- ❖ 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**.

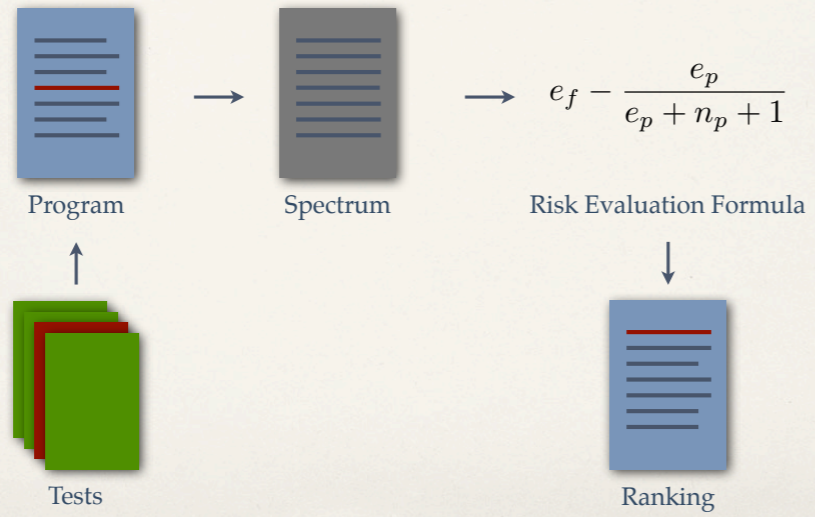


The most effective way to do it, is to do it.

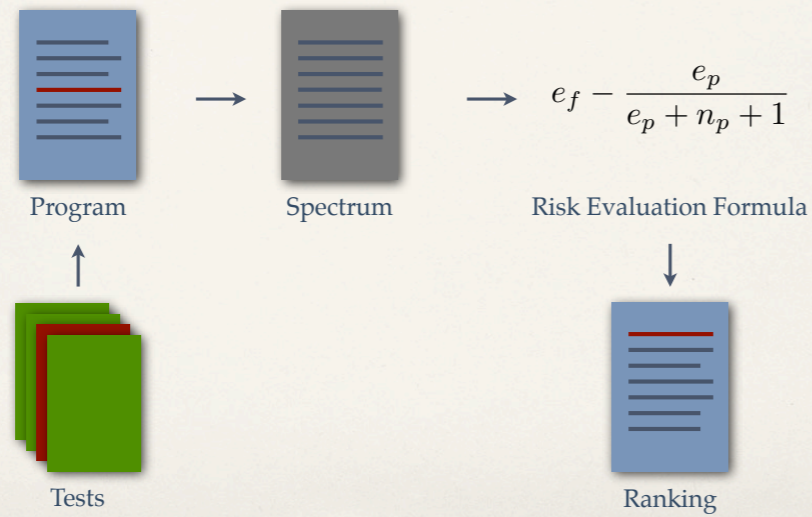
- ❖ GP provides a structured, automated way of doing iterative design.
- ❖ It can cope with a much diverse spectra and other meta-data.
- ❖ GP can evolve to suit **your project**.



Evolving SBFL

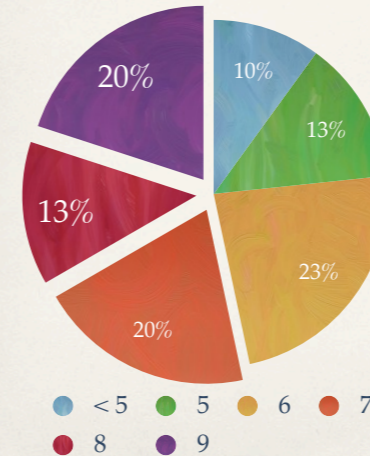


Evolving SBFL



Human Competitiveness

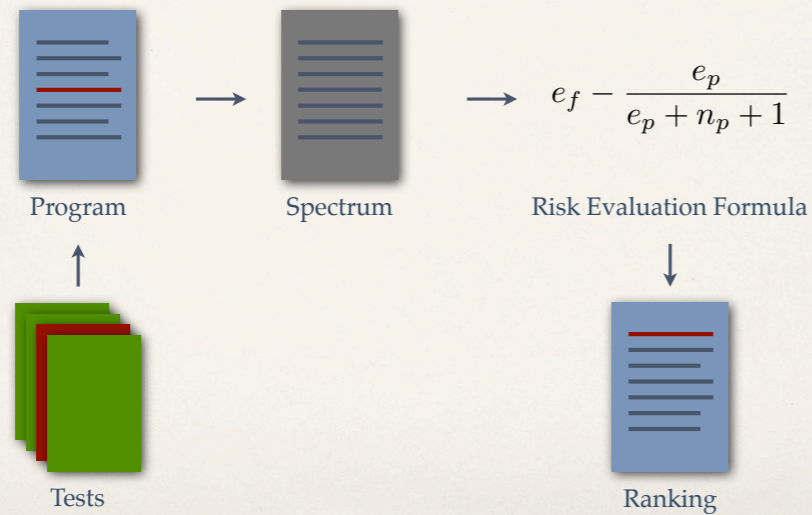
How many of 9 Existing Techniques can 30 GP runs match and/or outperform?



- * 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.

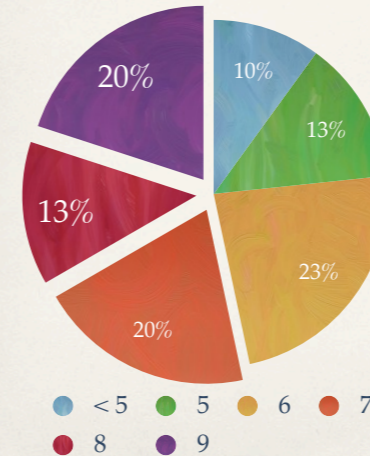
Four Unix tools with 92 faults: 20 random faults for training, 72 for evaluation.

Evolving SBFL



Human Competitiveness

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



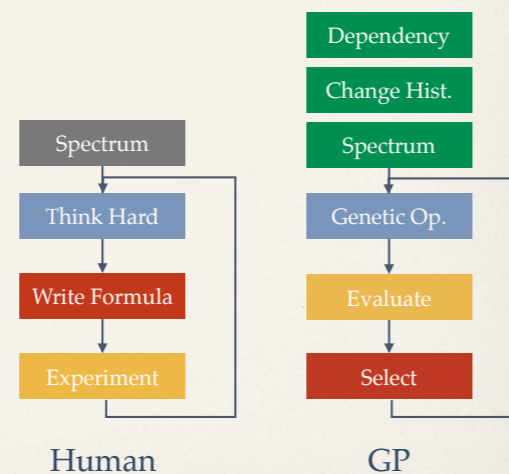
- * 6 runs outperform 8 existing techniques and match/outperform one of the state of the art with proof (Op1 and Op2).

- * 16 runs outperform all 7 existing techniques without proof.

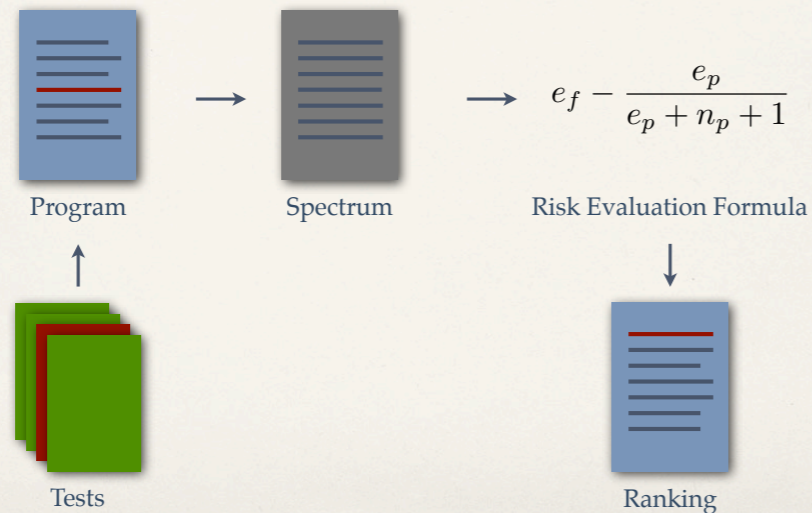
Four Unix tools with 92 faults: 20 random faults for training, 72 for evaluation.

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**.

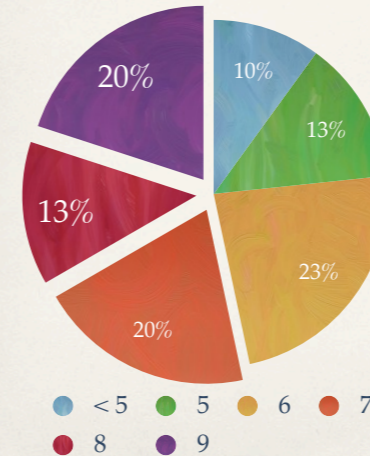


Evolving SBFL



Human Competitiveness

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



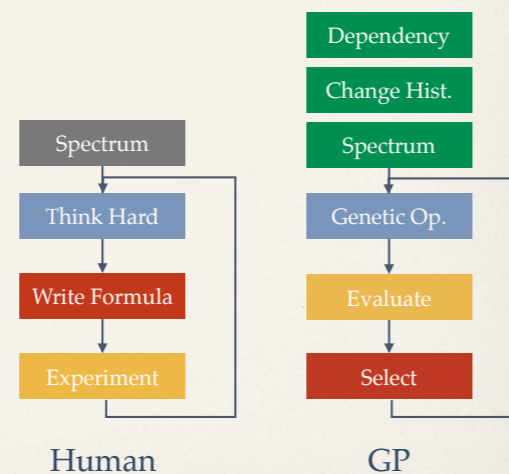
- * 6 runs outperform 8 existing techniques and match/outperform one of the state of the art with proof (Op1 and Op2).

- * 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>