Università
della
Svizzera
italiana

Faculty
of Informatics

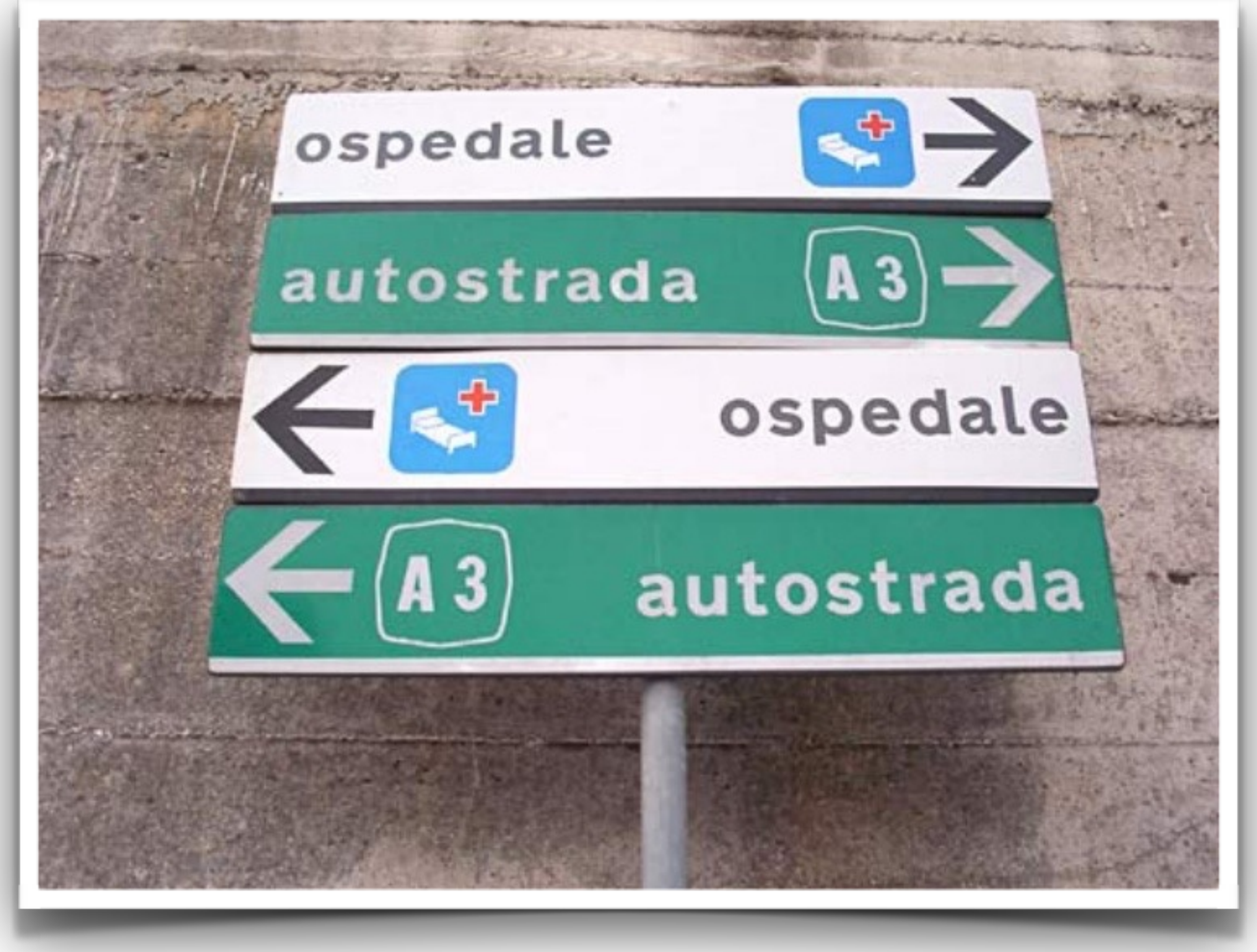
Software
Testing and
Analysis
Research group

Measuring Software Redundancy

Antonio Carzaniga, **Andrea Mattavelli**, Mauro Pezzè

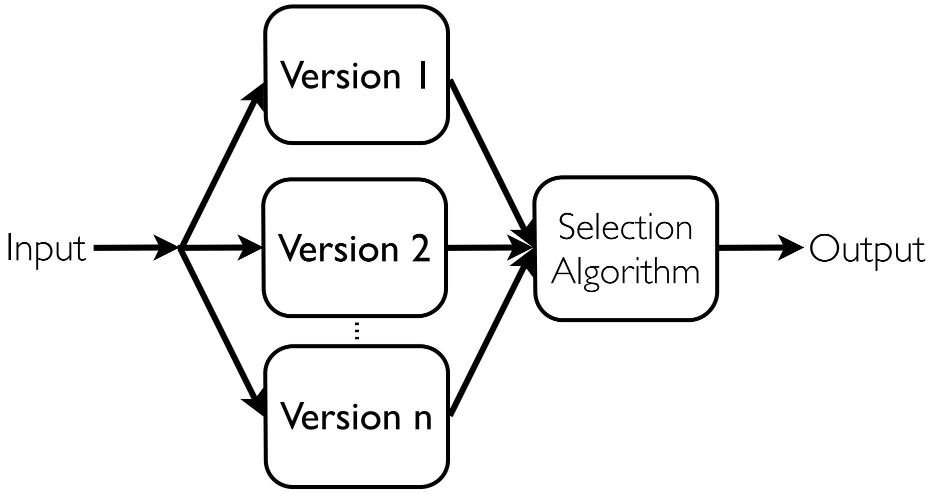
Università della Svizzera italiana (USI), Switzerland

Redundancy



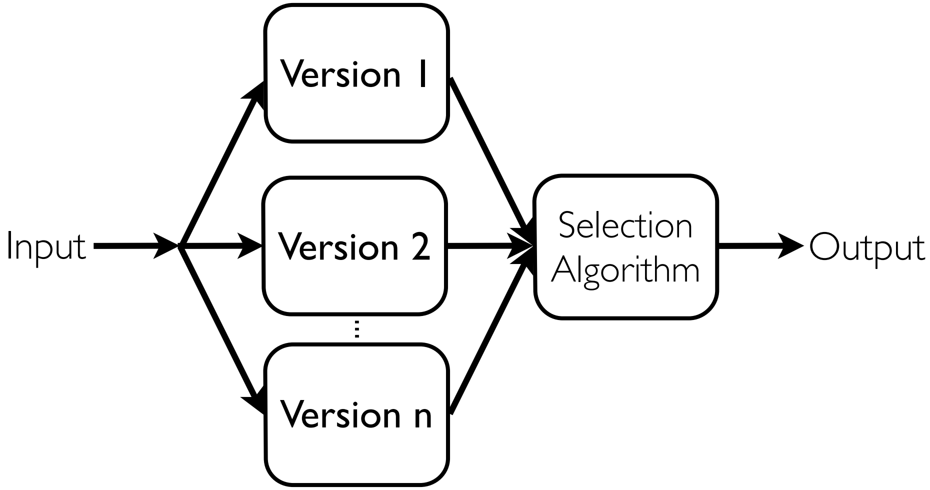
Software Redundancy

Software Redundancy

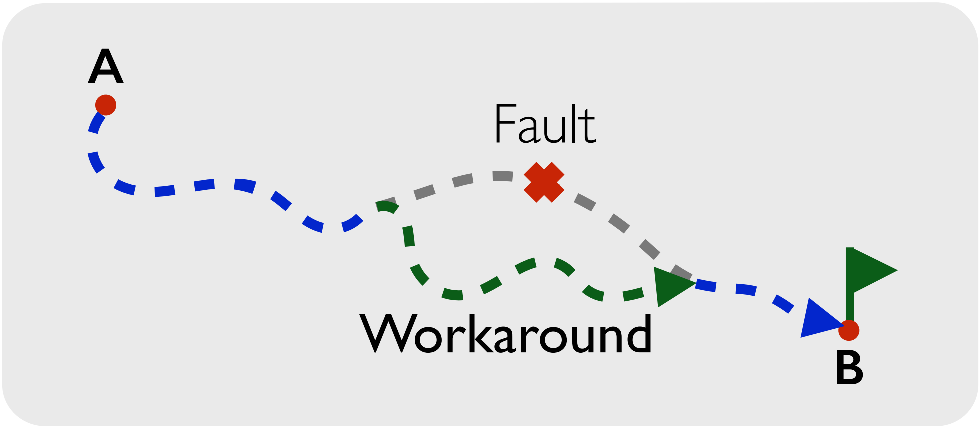


N-version

Software Redundancy

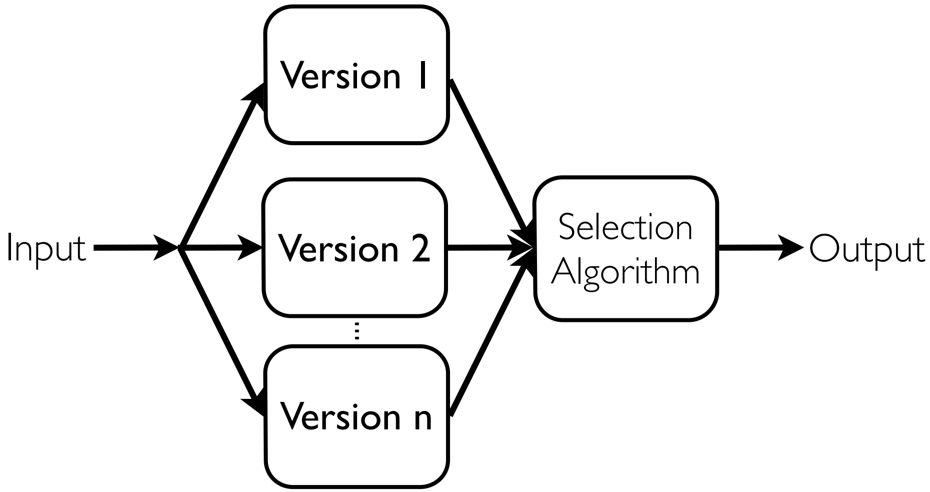


N-version



Automatic workarounds

Software Redundancy



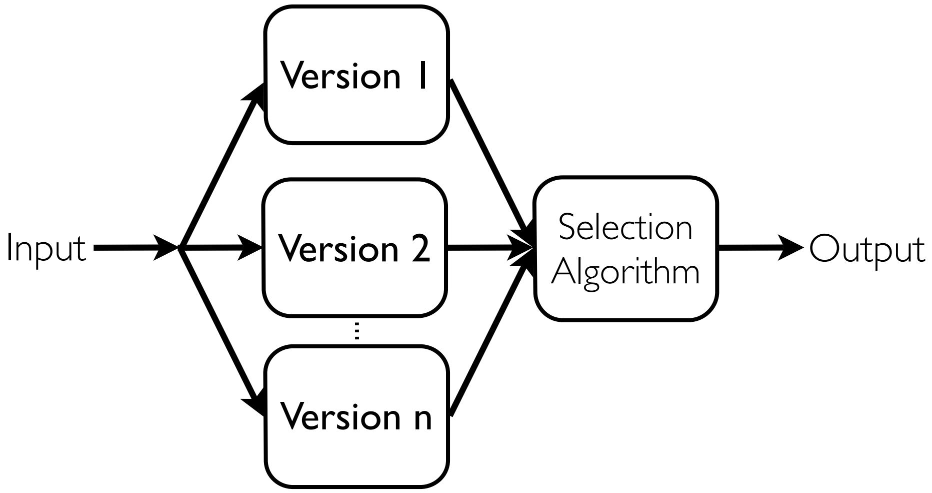
N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
  m.putAll(key, new List().add(value))
  m.entrySet().add(new Entry(key, value))
```

Automatic workarounds

Software Redundancy?



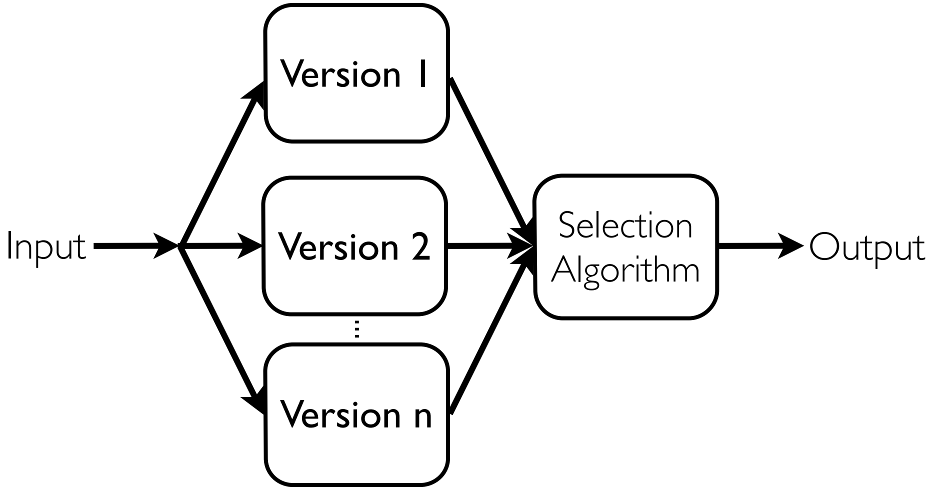
N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
  m.putAll(key, new List().add(value))
  m.entrySet().add(new Entry(key, value))
```

Automatic workarounds

Software Redundancy?



N-version

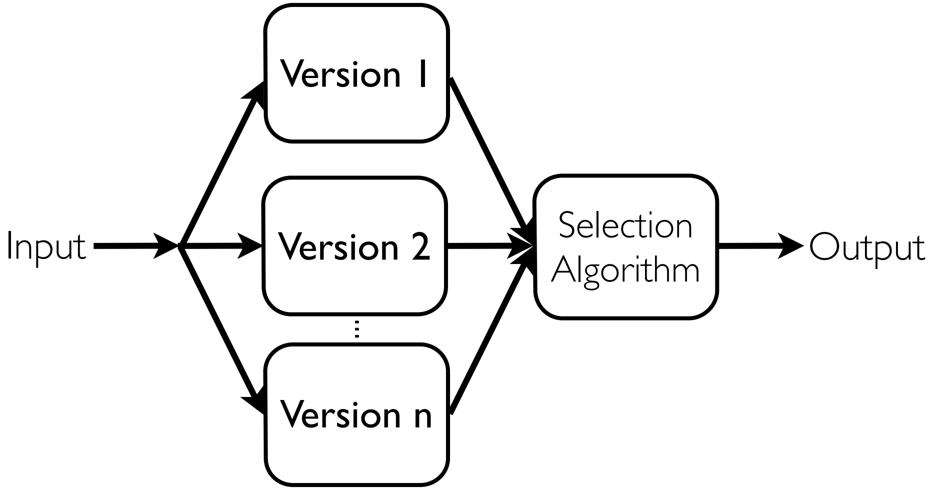
Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
   m.putAll(key, new List().add(value))
   m.entrySet().add(new Entry(key, value))
```

Automatic workarounds

Failures are correlated
[Knight et al.]

Software Redundancy?



N-version

Google Guava

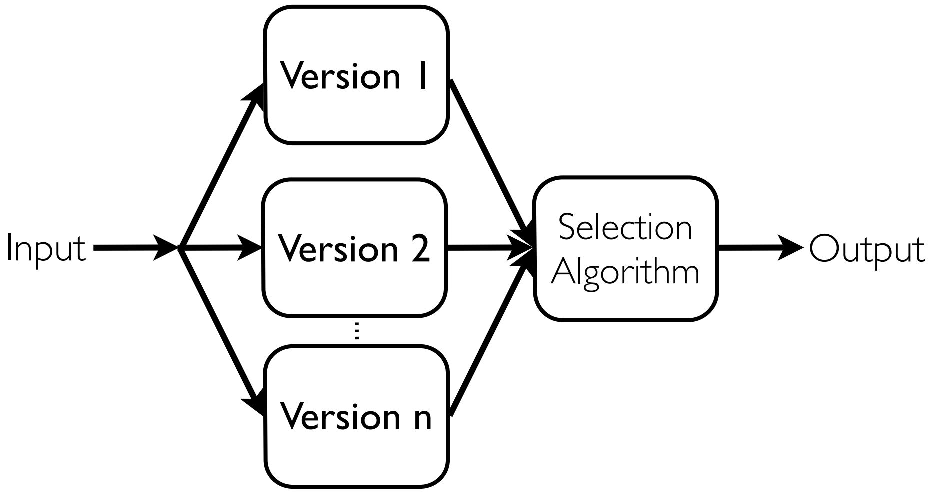
```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
  m.putAll(key, new List().add(value))
  m.entrySet().add(new Entry(key, value))
```

Automatic workarounds

Failures are correlated
[Knight et al.]

How much code
do they share?

Software Redundancy?



N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
   m.putAll(key, new List().add(value))
   m.entrySet().add(new Entry(key, value))
```

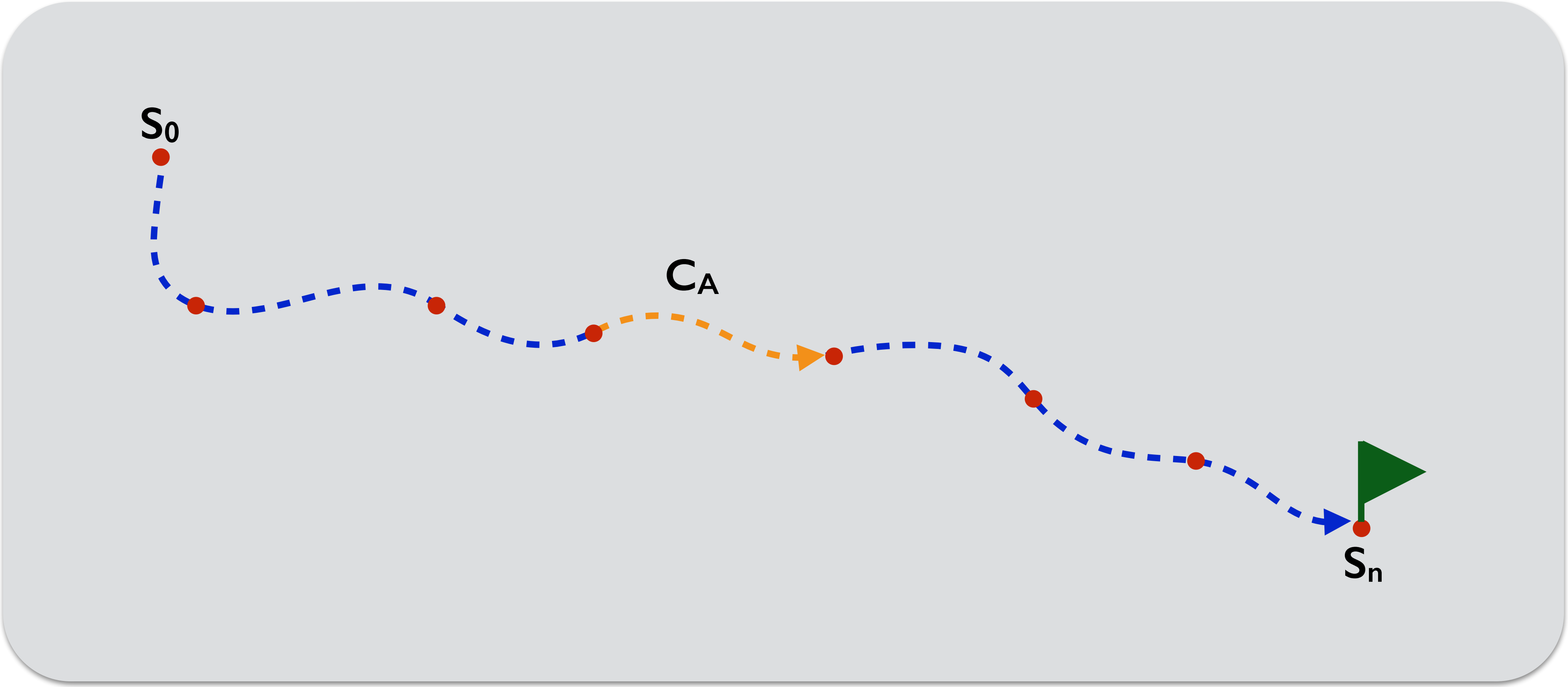
Automatic workarounds

Measuring software redundancy

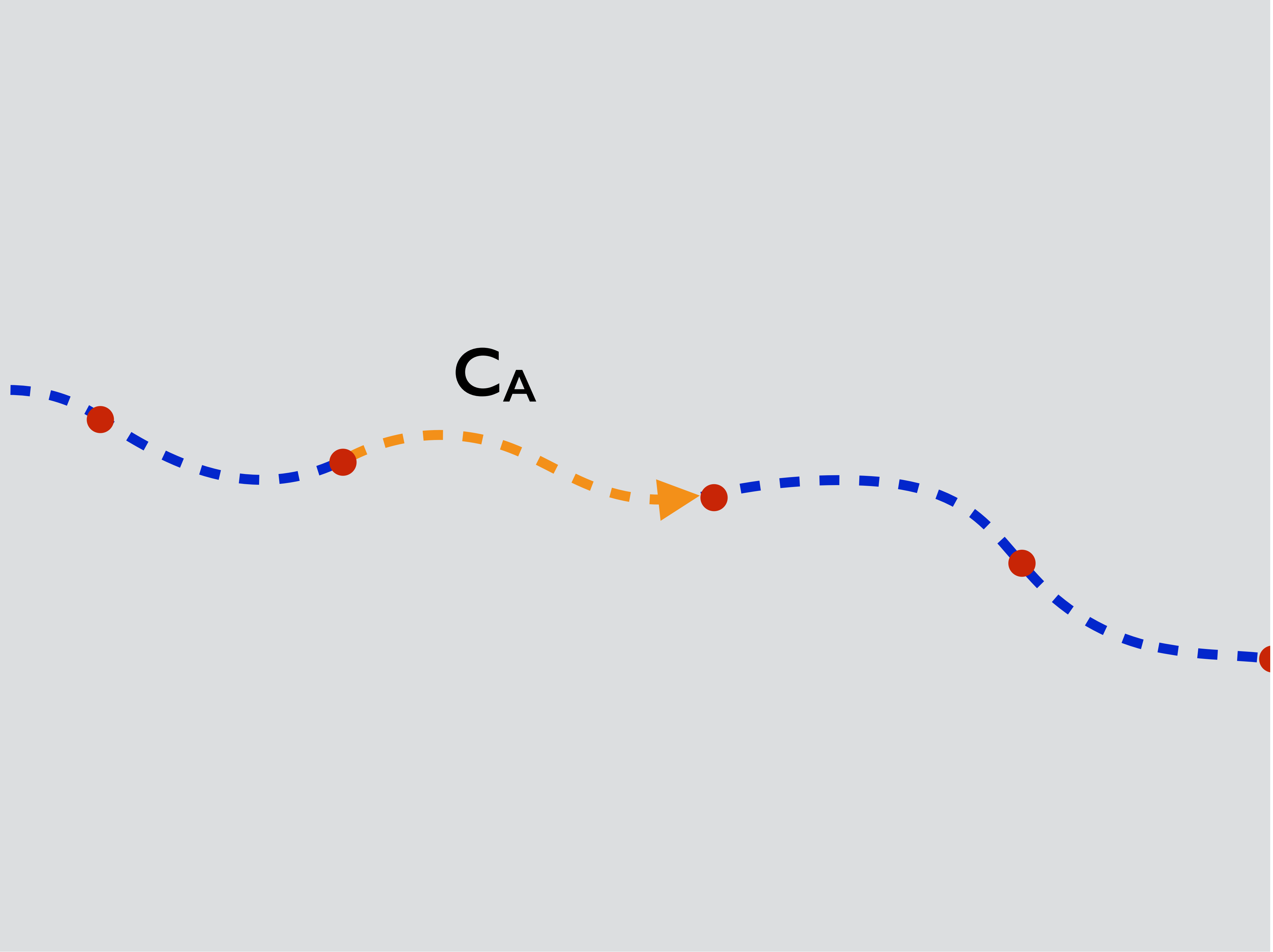
Informal Definition of Redundancy

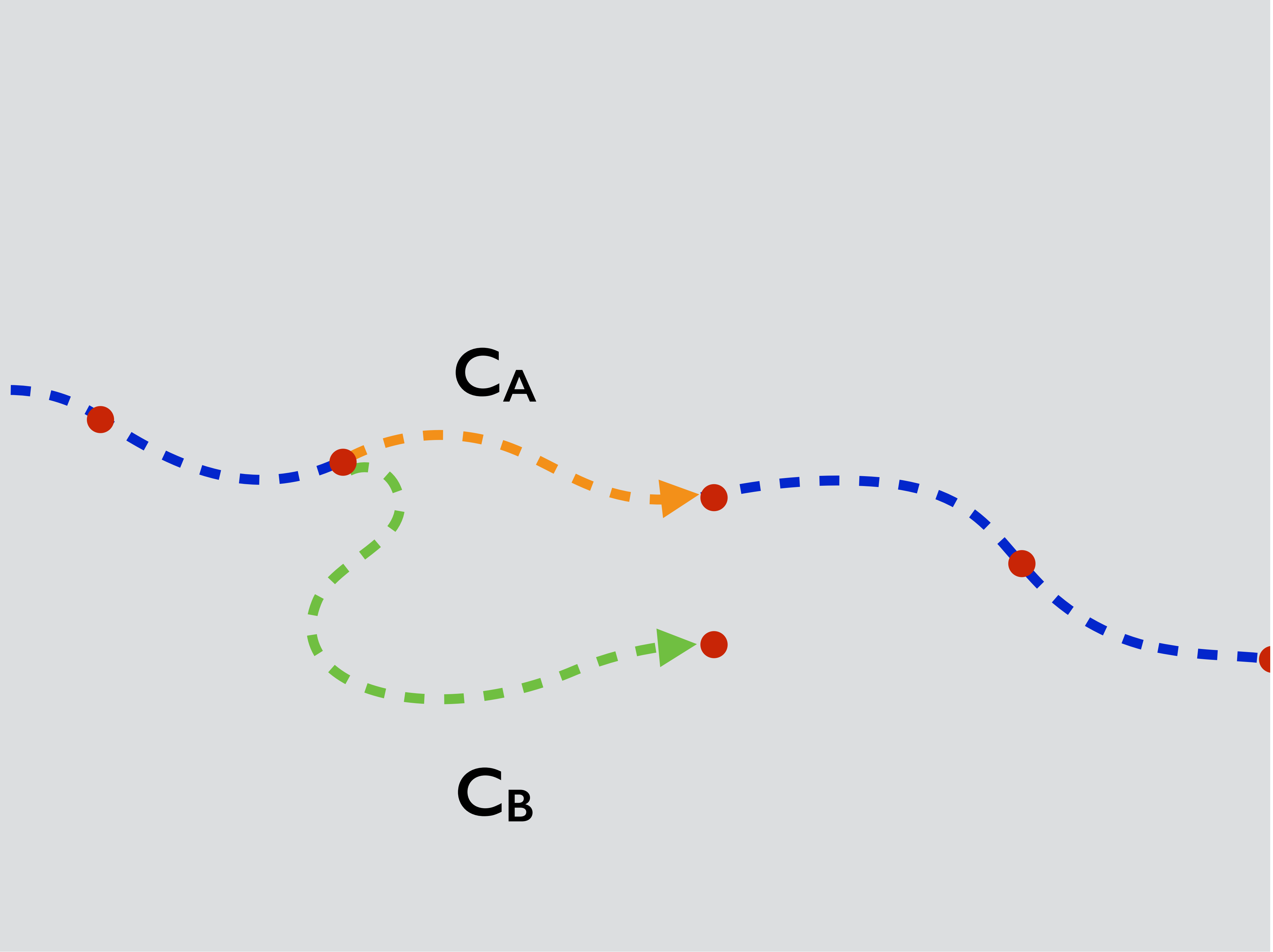
> “ Two fragments are redundant when they are **functionally equivalent** and at the same time their **executions are different.**

Informal Definition of Redundancy

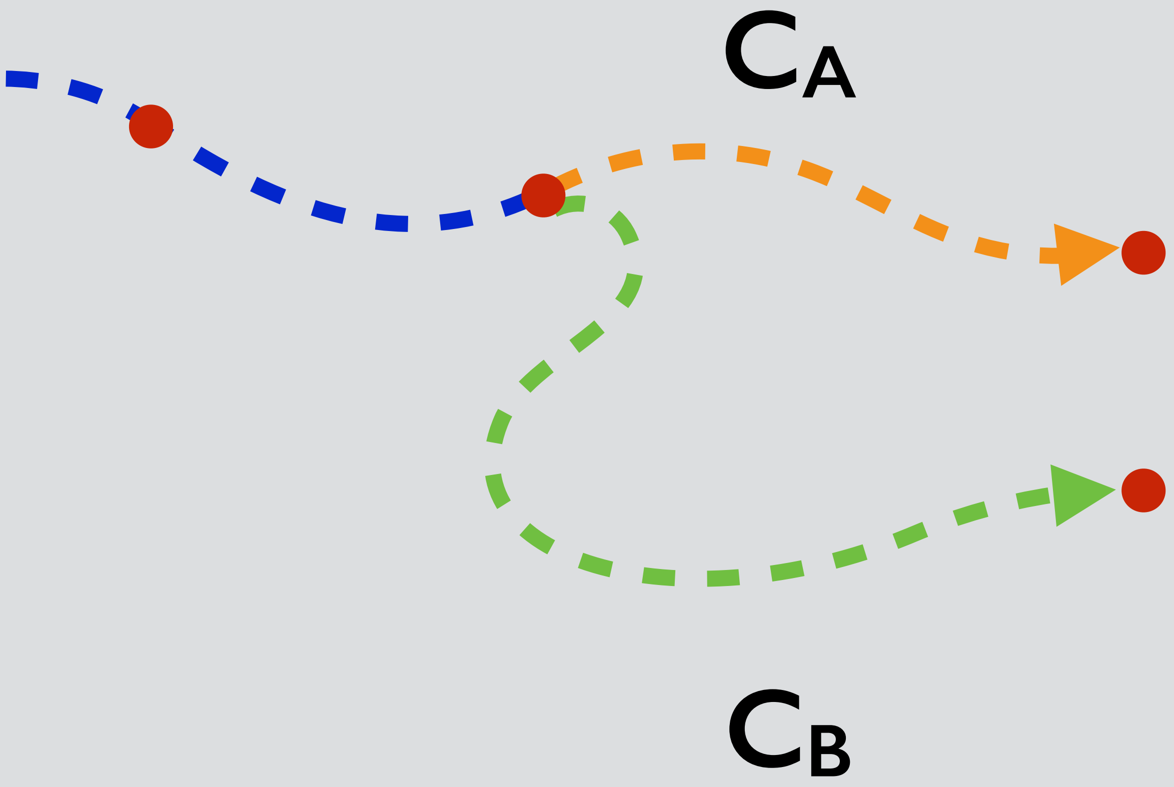


Application state space

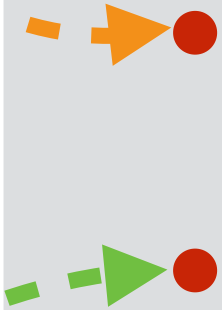

C_A


C_A
C_B

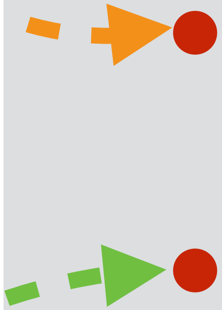
Functional Equivalence



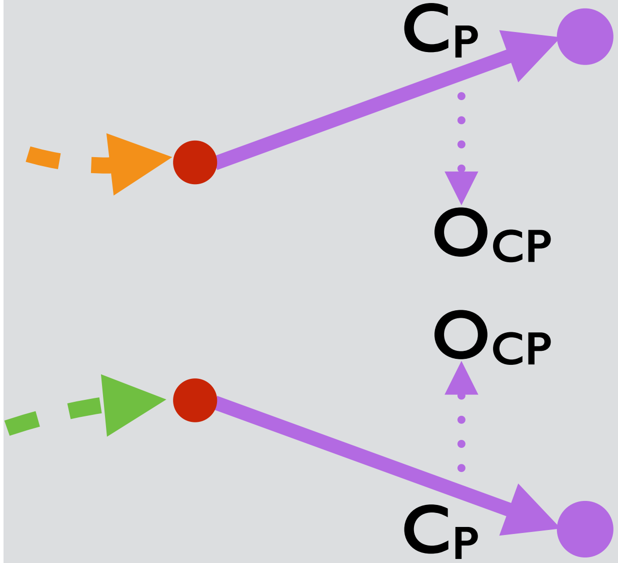
Functional Equivalence



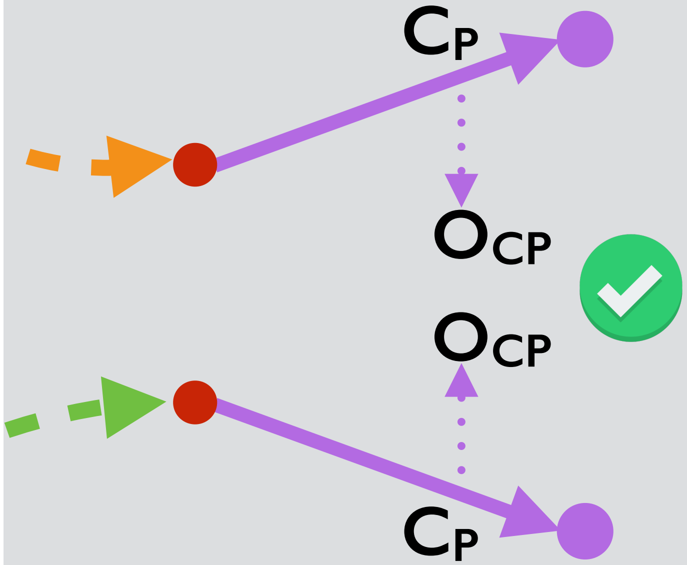
Observational Equivalence



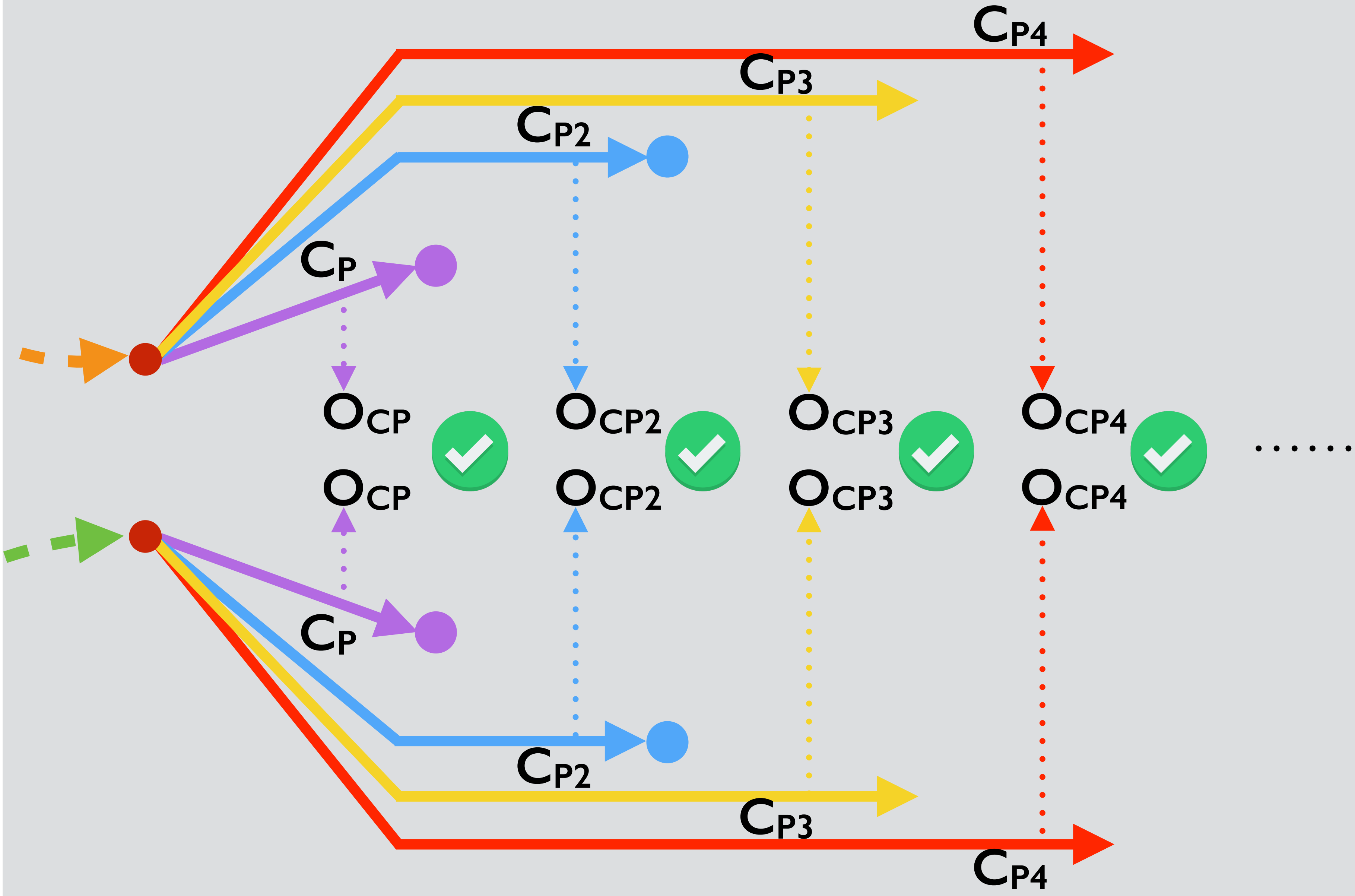
Observational Equivalence

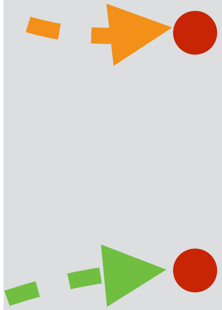


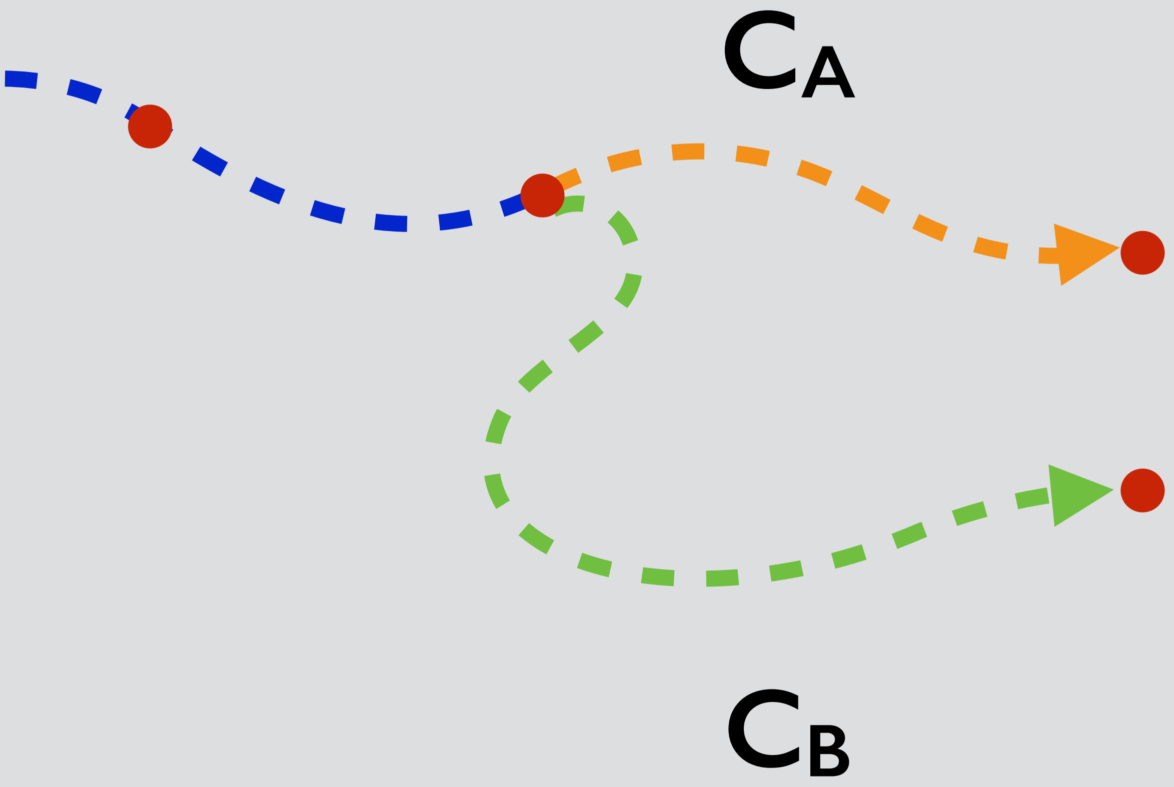
Observational Equivalence



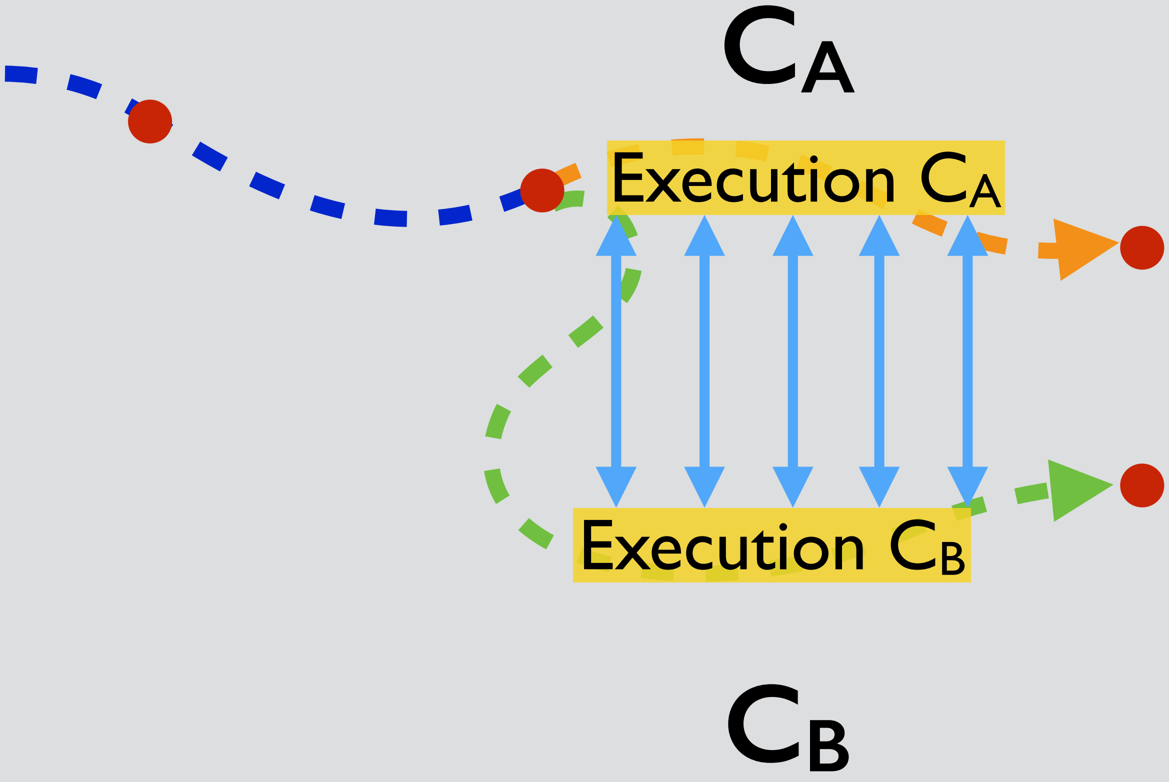
Observational Equivalence



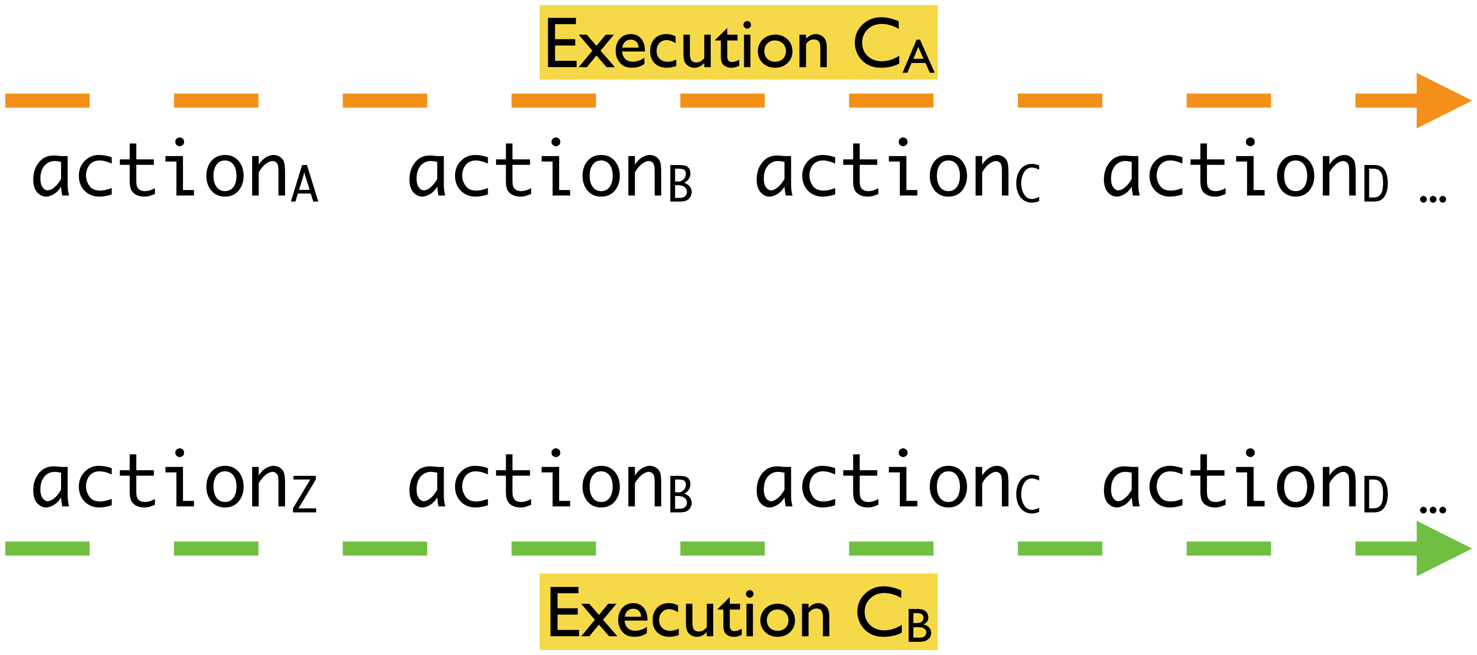



C_A
C_B

Execution Diversity



Execution Diversity



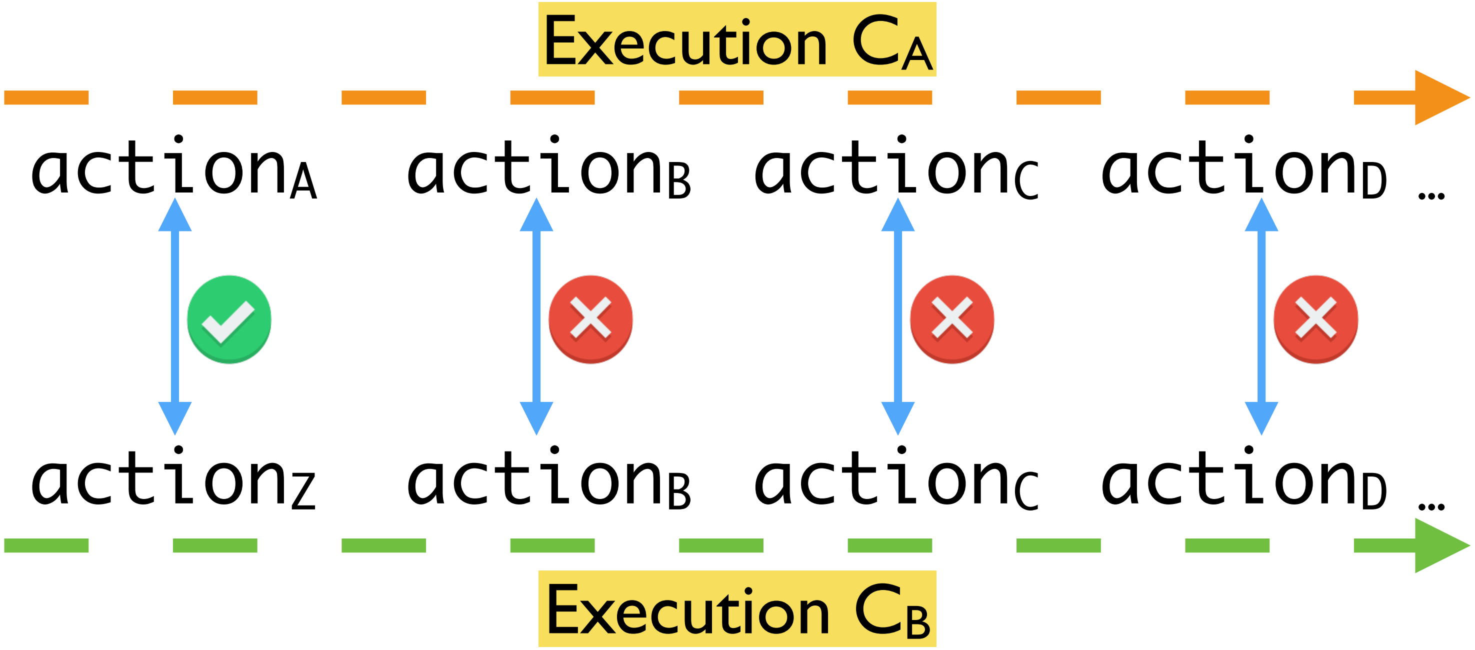
Execution C_A

action$_A$ action$_B$ action$_C$ action$_D$ …

action$_Z$ action$_B$ action$_C$ action$_D$ …

Execution C_B

Execution Diversity



Software Redundancy

Observational Equivalence && **Execution Diversity**

Software Redundancy

Observational Equivalence && **Execution Diversity**

- **Binary measure**

Software Redundancy

Observational Equivalence && **Execution Diversity**

- **Binary measure**
- **Not practical**

A Measure of Redundancy

A Measure of Redundancy

$$R = f\left(\begin{array}{l}\textbf{Degree} \text{ of} \\ \text{Equivalence}\end{array}, \begin{array}{l}\textbf{Degree} \text{ of} \\ \text{Diversity}\end{array}\right)$$

A Measure of Redundancy

$$R_S = e_S(C_A, C_B) \times d_S(C_A, C_B)$$

$$e_S, d_S \in [0,1]$$

A Measure of Redundancy

$$R_S = e_S(C_A, C_B) \times d_S(C_A, C_B)$$

$$e_S, d_S \in [0, 1]$$

$$R_{C_A, C_B} = \text{AGGREGATE}(R_S)$$

A **Practical** Measure of Redundancy

$$R_S = e_S(C_A, C_B) \times d_S(C_A, C_B)$$

$$e_S, d_S \in [0,1]$$

$$R_{C_A,C_B} = \textsc{aggregate}(R_S)$$

A **Practical** Measure of Redundancy

Sample the state space

$$R_S = e_S(C_A, C_B) \times d_S(C_A, C_B)$$

$$e_S, d_S \in [0,1]$$

$$R_{C_A,C_B} = \text{AGGREGATE}(R_S)$$

A **Practical** Measure of Redundancy

Observational equivalence measure

$$R_S = e_S(C_A, C_B) \times d_S(C_A, C_B)$$

$$e_S, d_S \in [0, 1]$$

$$R_{C_A, C_B} = \text{AGGREGATE}(R_S)$$

A **Practical** Measure of Redundancy

Difference between executions

$$R_S = e_S(C_A,C_B) \times d_S(C_A,C_B)$$

$$e_S, d_S \in [0,1]$$

$$R_{C_A,C_B} = \text{AGGREGATE}(R_S)$$

A **Practical** Measure of Redundancy

$$R_S = e_S(C_A, C_B) \times d_S(C_A, C_B)$$

$$e_S, d_S \in [0, 1]$$

$$R_{C_A, C_B} = \boxed{\text{AGGREGATE}}(R_S)$$

Aggregate the redundancy measure

Sampling the State Space

Sampling the State Space

```
ArrayListMultimap var0 = ArrayListMultimap.create();
var0.clear();
ArrayListMultimap var3 = ArrayListMultimap.create();
var3.clear();
boolean var5 = var3.isEmpty();
ArrayListMultimap var6 = ArrayListMultimap.create();
var6.clear();
boolean var8 = var6.isEmpty();
boolean var9 = var3.putAll((Multimap) var6);
java.util.List var11 = var3.removeAll("hi!");
boolean var12 = var0.putAll((short) (-1), (java.lang.Iterable) var11);
var0.clear();
ArrayListMultimap var14 = ArrayListMultimap.create((Multimap) var0);
ArrayListMultimap var17 = ArrayListMultimap.create(1, 10);
var17.clear();
ArrayListMultimap var19 = ArrayListMultimap.create();
var19.clear();
ArrayListMultimap var21 = ArrayListMultimap.create((Multimap) var19);
boolean var22 = var14.put(var17, var19); // Code fragment A
```

Sampling the State Space

```
ArrayListMultimap var0 = ArrayListMultimap.create();
var0.clear();
ArrayListMultimap var3 = ArrayListMultimap.create();
var3.clear();
boolean var5 = var3.isEmpty();
ArrayListMultimap var6 = ArrayListMultimap.create();
var6.clear();
boolean var8 = var6.isEmpty();
boolean var9 = var3.putAll((Multimap) var6);
java.util.List var11 = var3.removeAll("hi!");
boolean var12 = var0.putAll((short) (-1), (java.lang.Iterable) var11);
var0.clear();
ArrayListMultimap var14 = ArrayListMultimap.create((Multimap) var0);
ArrayListMultimap var17 = ArrayListMultimap.create(1, 10);
var17.clear();
ArrayListMultimap var19 = ArrayListMultimap.create();
var19.clear();
ArrayListMultimap var21 = ArrayListMultimap.create((Multimap) var19);
boolean var22 = var14.put(var17, var19); // Code fragment A
```

Measuring Equivalence

```java
ArrayListMultimap var0 = ArrayListMultimap.create();
var0.clear();
ArrayListMultimap var3 = ArrayListMultimap.create();
var3.clear();
boolean var5 = var3.isEmpty();
ArrayListMultimap var6 = ArrayListMultimap.create();
var6.clear();
boolean var8 = var6.isEmpty();
boolean var9 = var3.putAll((Multimap) var6);
java.util.List var11 = var3.removeAll("hi!");
boolean var12 = var0.putAll((short) (-1), (java.lang.Iterable) var11);
var0.clear();
ArrayListMultimap var14 = ArrayListMultimap.create((Multimap) var0);
ArrayListMultimap var17 = ArrayListMultimap.create(1, 10);
var17.clear();
ArrayListMultimap var19 = ArrayListMultimap.create();
var19.clear();
ArrayListMultimap var21 = ArrayListMultimap.create((Multimap) var19);
boolean var22 = var14.put(var17, var19); // Code fragment A
```

Measuring Equivalence

```
boolean var22 = var14.put(var17, var19); // Code fragment A
```

Measuring Equivalence

```
boolean var22 = var14.put(var17, var19); // Code fragment A
```

Linkage: `boolean var22; ArrayListMultimap var14; Object var17, var19`

Measuring Equivalence

```java
boolean var22 = var14.put(var17, var19); // Code fragment A
```

Linkage: `boolean var22; ArrayListMultimap var14; Object var17, var19`

```java
// generated probing code:
System.out.println(var22);
boolean x0 = var14.isEmpty();
System.out.println(x0);
var14.clear();
java.util.Map x1 = var14.asMap();
int x2 = var14.size();
System.out.println(x2);
int x3 = x1.size();
System.out.println(x3);
java.util.Set x4 = x1.entrySet();
java.util.Iterator x5 = x4.iterator();
boolean x6 = x4.isEmpty();
System.out.println(x6);
// ... probing code continues
```

Measuring Equivalence

```
// Code fragment A
boolean var22 = var14.put(var17, var19);
```

```
// Code fragment B
List list = new List(); list.add(var19);
boolean var22 = var14.putAll(var17, list);
```

Linkage: `boolean var22; ArrayListMultimap var14; Object var17, var19`

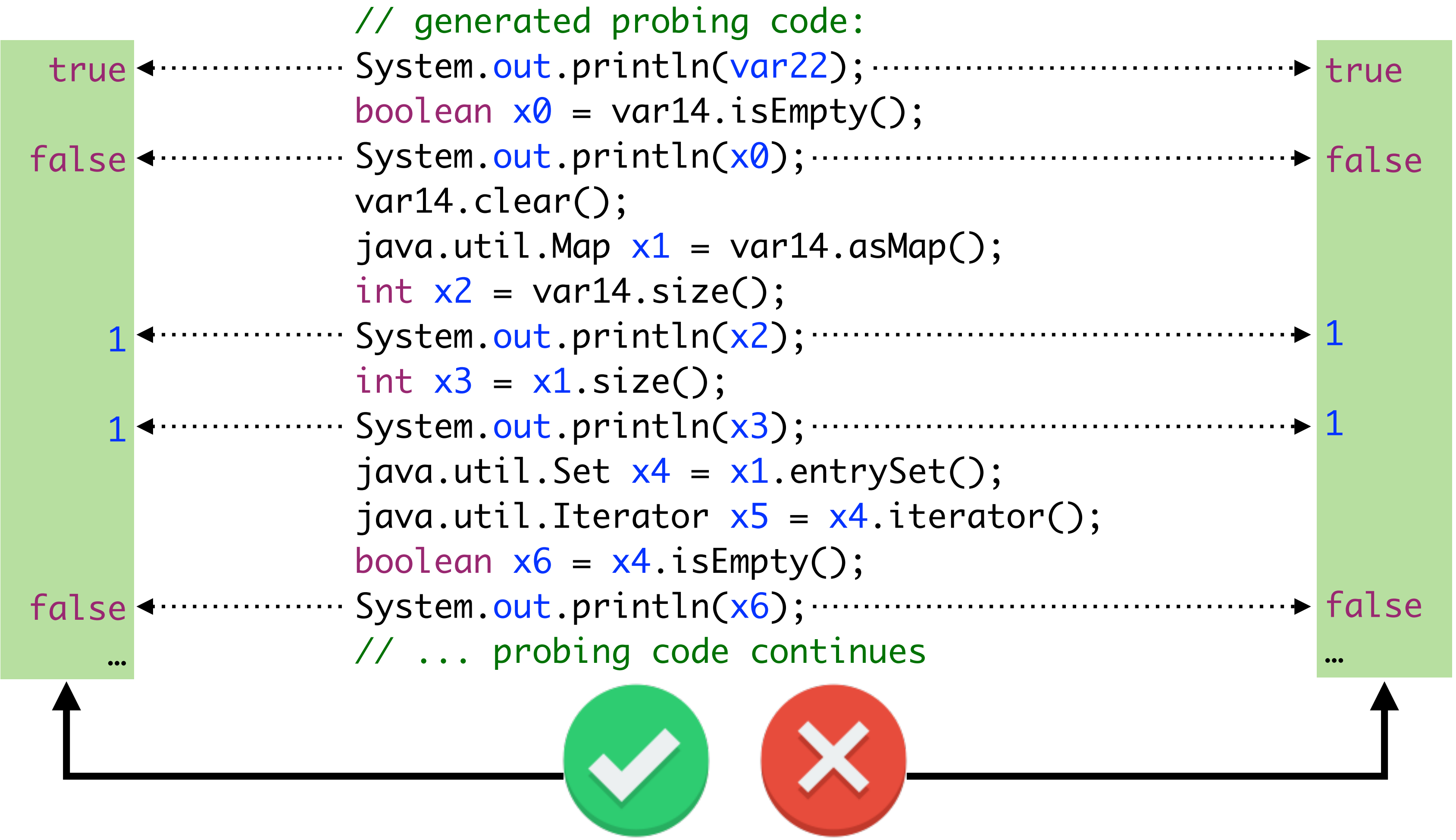
```
// generated probing code:
System.out.println(var22);
boolean x0 = var14.isEmpty();
System.out.println(x0);
var14.clear();
java.util.Map x1 = var14.asMap();
int x2 = var14.size();
System.out.println(x2);
int x3 = x1.size();
System.out.println(x3);
java.util.Set x4 = x1.entrySet();
java.util.Iterator x5 = x4.iterator();
boolean x6 = x4.isEmpty();
System.out.println(x6);
// ... probing code continues
```

Measuring Equivalence

```
// Code fragment A
boolean var22 = var14.put(var17, var19);
```

```
// Code fragment B
List list = new List(); list.add(var19);
boolean var22 = var14.putAll(var17, list);
```

Linkage: `boolean var22; ArrayListMultimap var14; Object var17, var19`



```
// generated probing code:
System.out.println(var22);
boolean x0 = var14.isEmpty();
System.out.println(x0);
var14.clear();
java.util.Map x1 = var14.asMap();
int x2 = var14.size();
System.out.println(x2);
int x3 = x1.size();
System.out.println(x3);
java.util.Set x4 = x1.entrySet();
java.util.Iterator x5 = x4.iterator();
boolean x6 = x4.isEmpty();
System.out.println(x6);
// ... probing code continues
```

```
true
false
1
1
false
…
```

```
true
false
1
1
false
…
```

Measuring Equivalence

$$e_S(C_A,C_B) = \frac{\text{successful}}{\text{total}}$$

CP1	✓
CP2	✓
CP3	✓
CP4	✓
CP5	✓
CP6	✓
CP7	✓
CP8	✓
CP9	✓
CP10	✓

$$e_S(C_A,C_B) = 1.0$$

CP1	✓
CP2	✗
CP3	✓
CP4	✗
CP5	✓
CP6	✗
CP7	✓
CP8	✓
CP9	✓
CP10	✓

$$e_S(C_A,C_B) = 0.7$$

Measuring Diversity

```
boolean var22 = var14.put(var17, var19); // Code fragment A
```

Measuring Diversity

```
boolean var22 = var14.put(var17, var19); // Execution of code fragment A
```

Measuring Diversity

```
boolean var22 = var14.put(var17, var19); // Execution of code fragment A
```

Projection

Measuring Diversity

```
boolean var22 = var14.put(var17, var19); // Execution of code fragment A
```

Code Projection **Data**

Measuring Diversity

```
boolean var22 = var14.put(var17, var19); // Execution of code fragment A
```

Code Projection **Data**

Statement

```
ArrayListMultimap.put(LObject;LObject;)Z@66
AbstractListMultimap.put(LObject;LObject;)Z@95
AbstractMultimap.put(LObject;LObject;)Z@200
```

Statement, Depth

```
3:ArrayListMultimap.put(LObject;LObject;)Z@66
4:AbstractListMultimap.put(LObject;LObject;)Z@95
5:AbstractMultimap.put(LObject;LObject;)Z@200
```

Measuring Diversity

```
boolean var22 = var14.put(var17, var19); // Execution of code fragment A
```

Code Projection **Data**

Type, Value

```
Ljava/util/Map;→{}
Ljava/util/Set;→[]
Ljava/util/HashMap;→{}
I→1
I←1
```

Class, Field, Value

```
AbstractMultimap.map→{}
HashMap.entrySet→[]
HashMap$EntrySet.this$0→{}
HashMap$HashIterator.modCount→1
HashMap$HashIterator.expectedModCount←1
```

Measuring Diversity

```java
// Code fragment A
boolean var22 = var14.put(var17, var19);
```

```java
// Code fragment B
List list = new List(); list.add(var19);
boolean var22 = var14.putAll(var17, list);
```

Code Projection

Measuring Diversity

```
// Code fragment A
boolean var22 = var14.put(var17, var19);
```

```
// Code fragment B
List list = new List(); list.add(var19);
boolean var22 = var14.putAll(var17, list);
```

Code Projection

```
ArrayListMultimap.put(LObject;LObject;)Z@66
AbstractListMultimap.put(LObject;LObject;)Z@95
AbstractMultimap.put(LObject;LObject;)Z@200
ArrayListMultimap.hashCode()I@66
AbstractMultimap.hashCode()I@1380
AbstractMap.hashCode()I@491
AbstractMap.hashCode()I@492
HashMap.entrySet()LSet;@953
HashMap.entrySet0()LSet;@957
HashMap.entrySet0()LSet;@958
```

```
ArrayListMultimap.putAll(LObject;LIterable;)Z@66
AbstractMultimap.putAll(LObject;LIterable;)Z@248
ArrayList.iterator()LIterator;@774
ArrayList$Itr.<init>(LArrayList;LArrayList$1;)V@780
ArrayList$Itr.<init>(LArrayList;)V@780
ArrayList$Itr.<init>(LArrayList;)V@782
ArrayList$Itr.<init>(LArrayList;)V@783
ArrayList$Itr.hasNext()Z@786
ArrayList.access$100(LArrayList;)I@102
AbstractMultimap.putAll(LObject;LIterable;)Z@252
AbstractMultimap.getOrCreateCollection(LObject;)LC;@219
HashMap.get(LObject;)LObject;@315
HashMap.get(LObject;)LObject;@317
HashMap.hash(I)I@268
HashMap.hash(I)I@269
HashMap.get(LObject;)LObject;@318
HashMap.indexFor(II)I@276
```

Measuring Diversity

```
// Code fragment A
boolean var22 = var14.put(var17, var19);
```

```
// Code fragment B
List list = new List(); list.add(var19);
boolean var22 = var14.putAll(var17, list);
```

Code Projection

```
ArrayListMultimap.put(LObject;LObject;)Z@66
AbstractListMultimap.put(LObject;LObject;)Z@95
AbstractMultimap.put(LObject;LObject;)Z@200
ArrayListMultimap.hashCode()I@66
AbstractMultimap.hashCode()I@1380
AbstractMap.hashCode()I@491
AbstractMap.hashCode()I@492
HashMap.entrySet()LSet;@953
HashMap.entrySet0()LSet;@957
HashMap.entrySet0()LSet;@958
```

```
ArrayListMultimap.putAll(LObject;LIterable;)Z@66
AbstractMultimap.putAll(LObject;LIterable;)Z@248
ArrayList.iterator()LIterator;@774
ArrayList$Itr.<init>(LArrayList;LArrayList$1;)V@780
ArrayList$Itr.<init>(LArrayList;)V@780
ArrayList$Itr.<init>(LArrayList;)V@782
ArrayList$Itr.<init>(LArrayList;)V@783
ArrayList$Itr.hasNext()Z@786
ArrayList.access$100(LArrayList;)I@102
AbstractMultimap.putAll(LObject;LIterable;)Z@252
AbstractMultimap.getOrCreateCollection(LObject;)LC;@219
HashMap.get(LObject;)LObject;@315
HashMap.get(LObject;)LObject;@317
HashMap.hash(I)I@268
HashMap.hash(I)I@269
HashMap.get(LObject;)LObject;@318
HashMap.indexFor(II)I@276
```

$$d_S(C_A, C_B) = 1 - \text{SIMILARITY}(P_{S,A}, P_{S,B})$$

A **Practical** Measure of Redundancy

A **Practical** Measure of Redundancy

S0

S1

S2

S3

S4

S5

S6

S7

S8

S9

S10

A **Practical** Measure of Redundancy

	e_s
S0	1.0
S1	1.0
S2	1.0
S3	1.0
S4	1.0
S5	1.0
S6	1.0
S7	1.0
S8	0.9
S9	1.0
S10	1.0

A **Practical** Measure of Redundancy

	e_s	d_s
S0	1.0	0.32989693
S1	1.0	0.51781228
S2	1.0	0.32989693
S3	1.0	0.51781228
S4	1.0	0.51781228
S5	1.0	0.32989693
S6	1.0	0.32989693
S7	1.0	0.51781228
S8	0.9	0.61892315
S9	1.0	0.32989693
S10	1.0	0.32989693

A **Practical** Measure of Redundancy

	e_s	d_s	R_s
S0	1.0	0.32989693	0.32989693
S1	1.0	0.51781228	0.51781228
S2	1.0	0.32989693	0.32989693
S3	1.0	0.51781228	0.51781228
S4	1.0	0.51781228	0.51781228
S5	1.0	0.32989693	0.32989693
S6	1.0	0.32989693	0.32989693
S7	1.0	0.51781228	0.51781228
S8	0.9	0.61892315	0.55703083
S9	1.0	0.32989693	0.32989693
S10	1.0	0.32989693	0.32989693

A **Practical** Measure of Redundancy

	e_s	d_s	R_s
S0	1.0	0.32989693	0.32989693
S1	1.0	0.51781228	0.51781228
S2	1.0	0.32989693	0.32989693
S3	1.0	0.51781228	0.51781228
S4	1.0	0.51781228	0.51781228
S5	1.0	0.32989693	0.32989693
S6	1.0	0.32989693	0.32989693
S7	1.0	0.51781228	0.51781228
S8	0.9	0.61892315	0.55703083
S9	1.0	0.32989693	0.32989693
S10	1.0	0.32989693	0.32989693

$$R = \text{AVG}(R_s) = 0.418 \pm 0.10$$

Evaluation

Consistency

Significance and **usefulness**

Evaluation

Consistency

1. Non-reflexivity
2. Stability
3. Equivalence measure

Evaluation

Consistency

1. Non-reflexivity
2. **Stability**
3. Equivalence measure

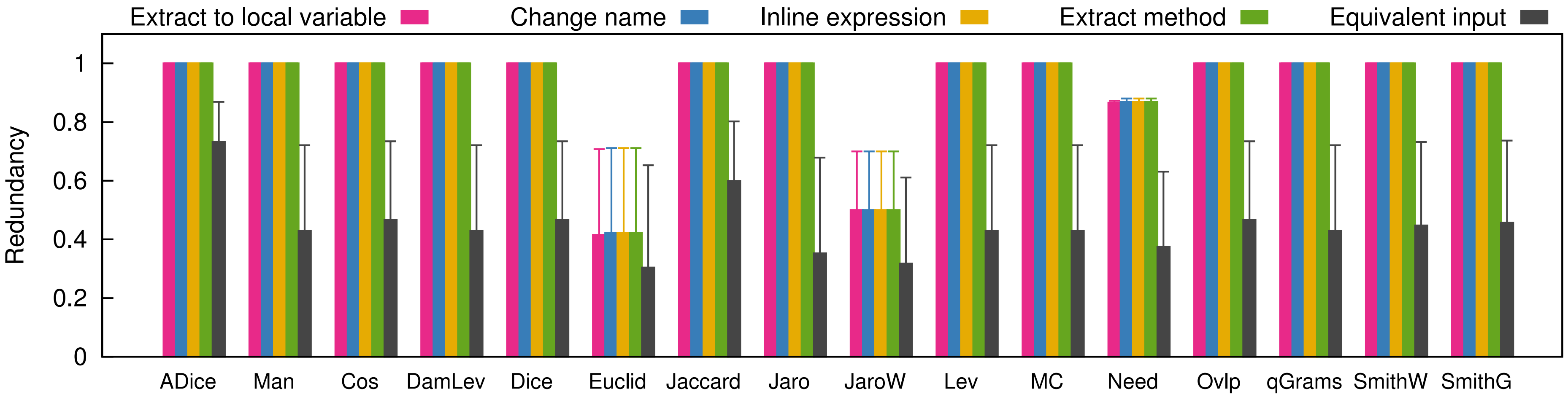
Stability

Stability

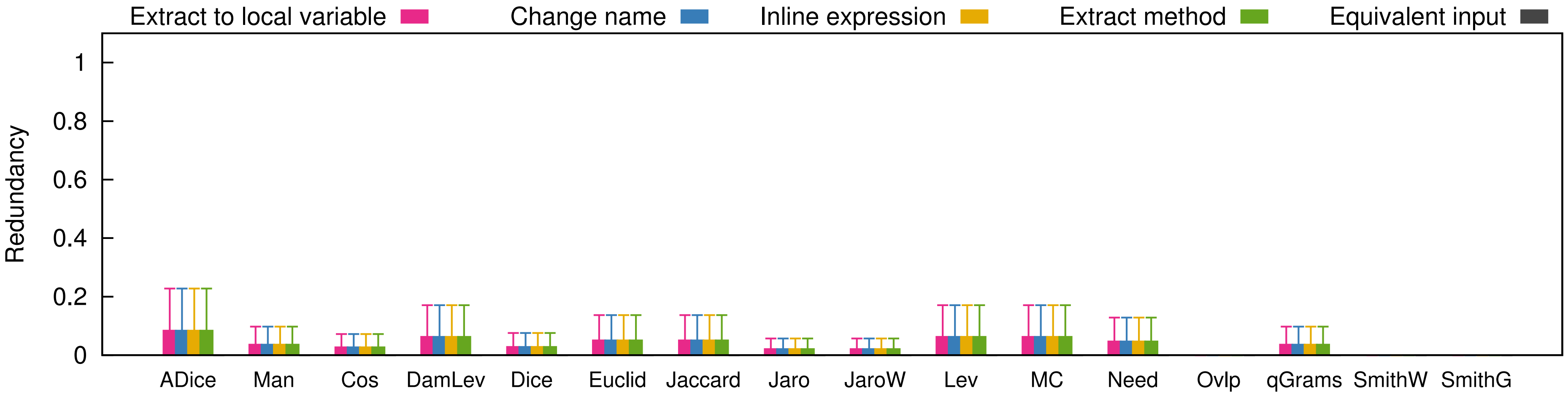
	Algorithm	**# Impl.**
Search	Binary search	4
	Linear search	4
Sorting	Bubble sort	7
	Insertion sort	3
	Merge sort	4
	Quicksort	3

Stability

Code Projections



Data Projections



Evaluation

Significance and **usefulness**

1. Low-level vs high-level
2. Predictive ability

Low-level vs High-level

Code Redundancy vs Algorithmic Redundancy

Low-level vs High-level

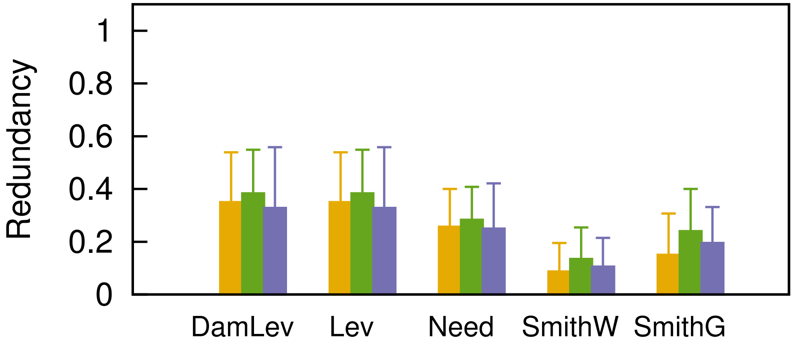
Code Redundancy vs Algorithmic Redundancy

	Algorithm	**# Impl.**
Search	Binary search	4
	Linear search	4
Sorting	Bubble sort	7
	Insertion sort	3
	Merge sort	4
	Quicksort	3

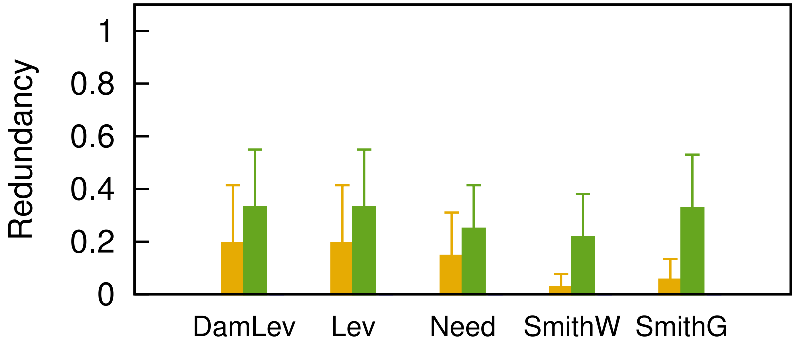
Low-level vs High-level

Code Redundancy vs Algorithmic Redundancy

Binary search

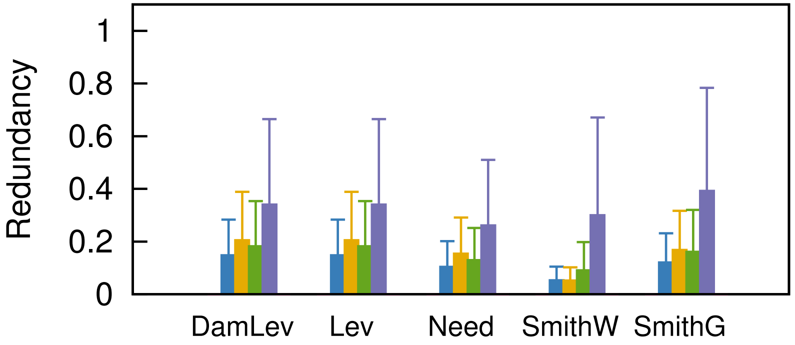


Linear search

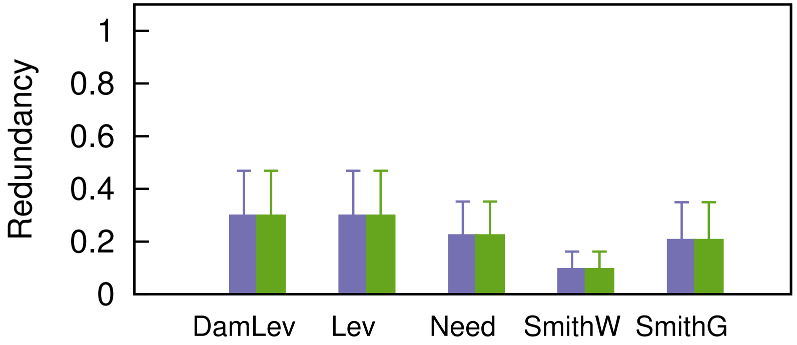


Same algorithm, different implementation

Bubble sort



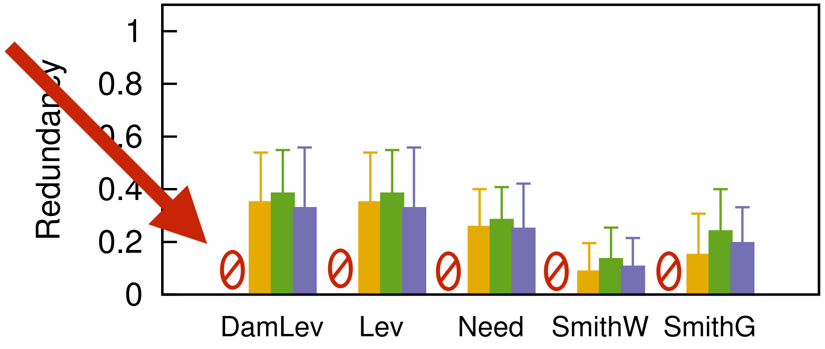
Insertion sort



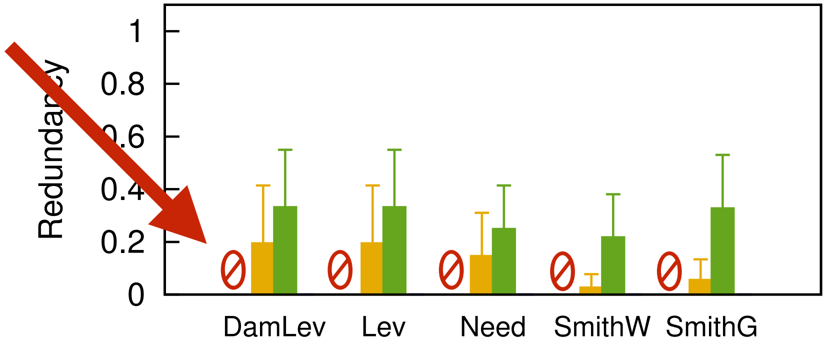
Low-level vs High-level

Code Redundancy vs Algorithmic Redundancy

Binary search

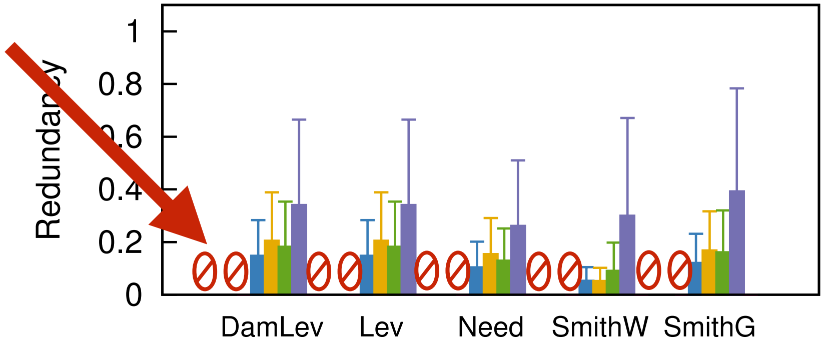


Linear search

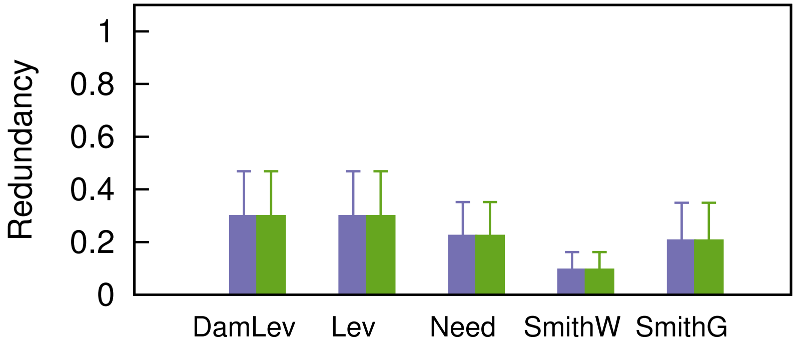


Same algorithm, different implementation

Bubble sort

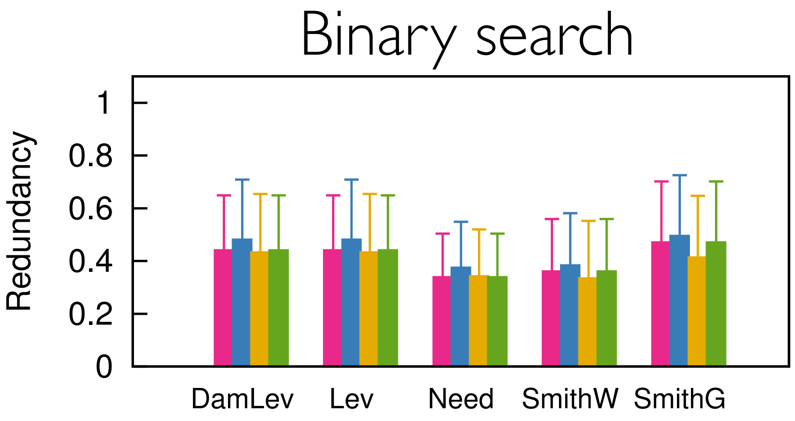


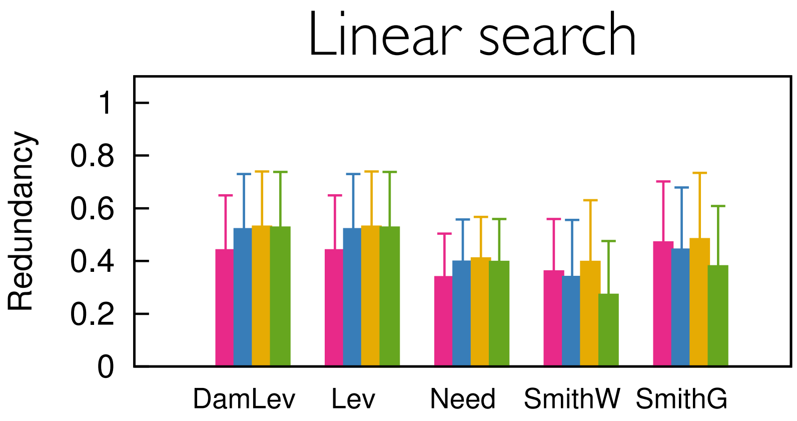
Insertion sort



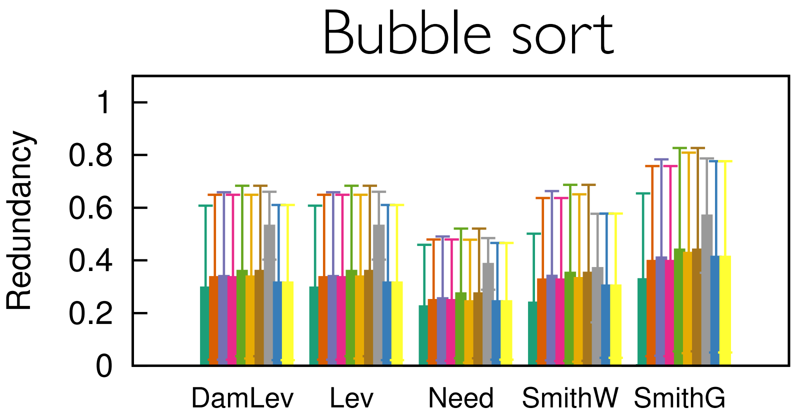
Low-level vs High-level

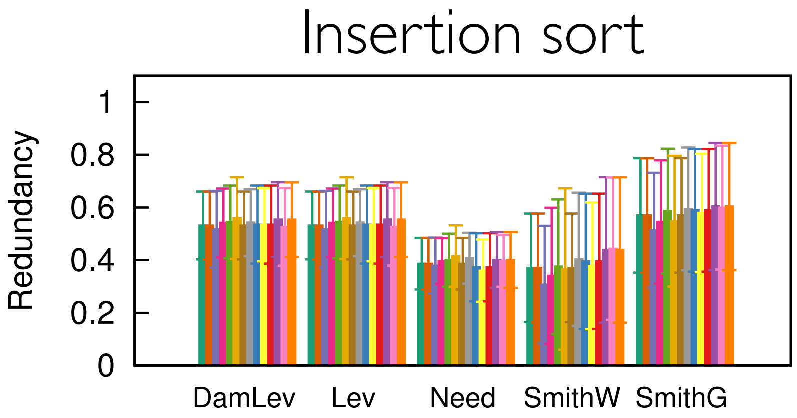
Code Redundancy vs Algorithmic Redundancy


Binary search


Linear search

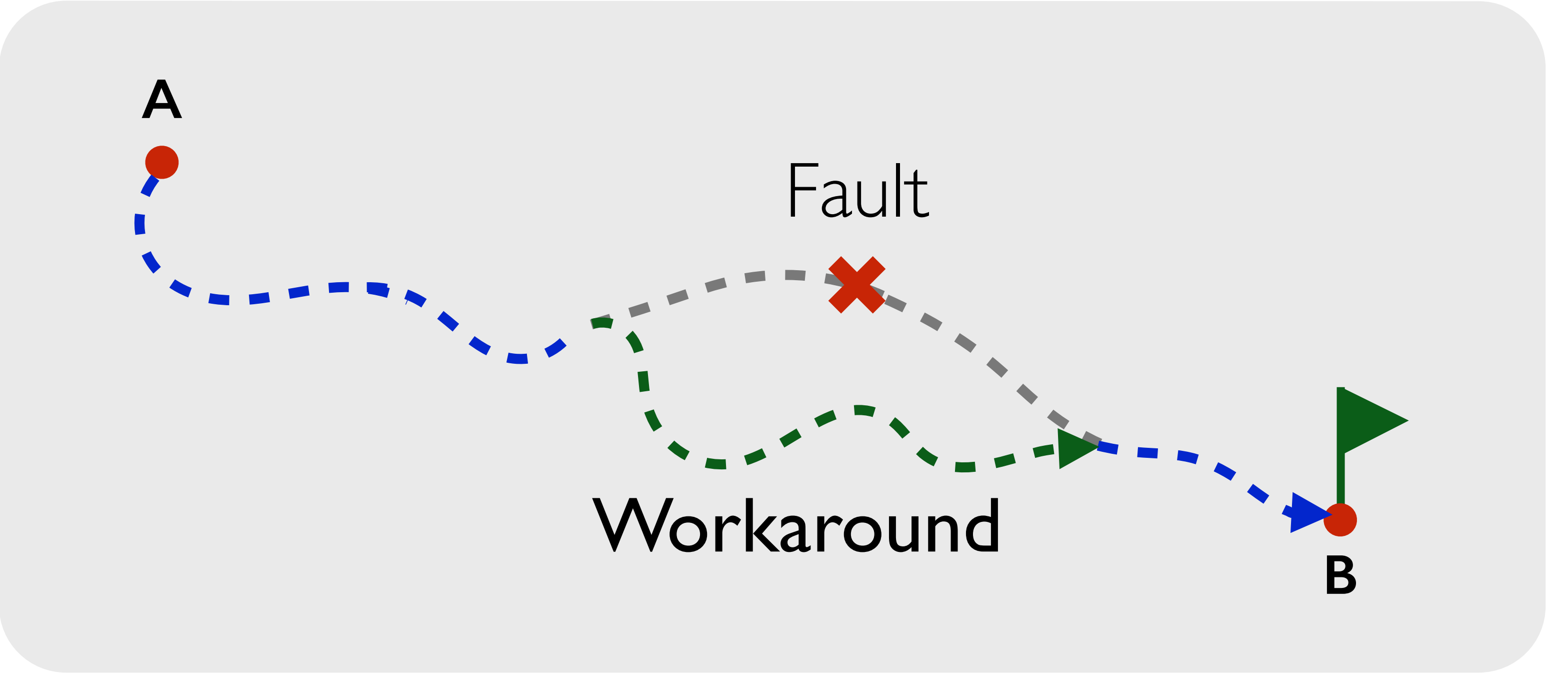
Different algorithm


Bubble sort


Insertion sort

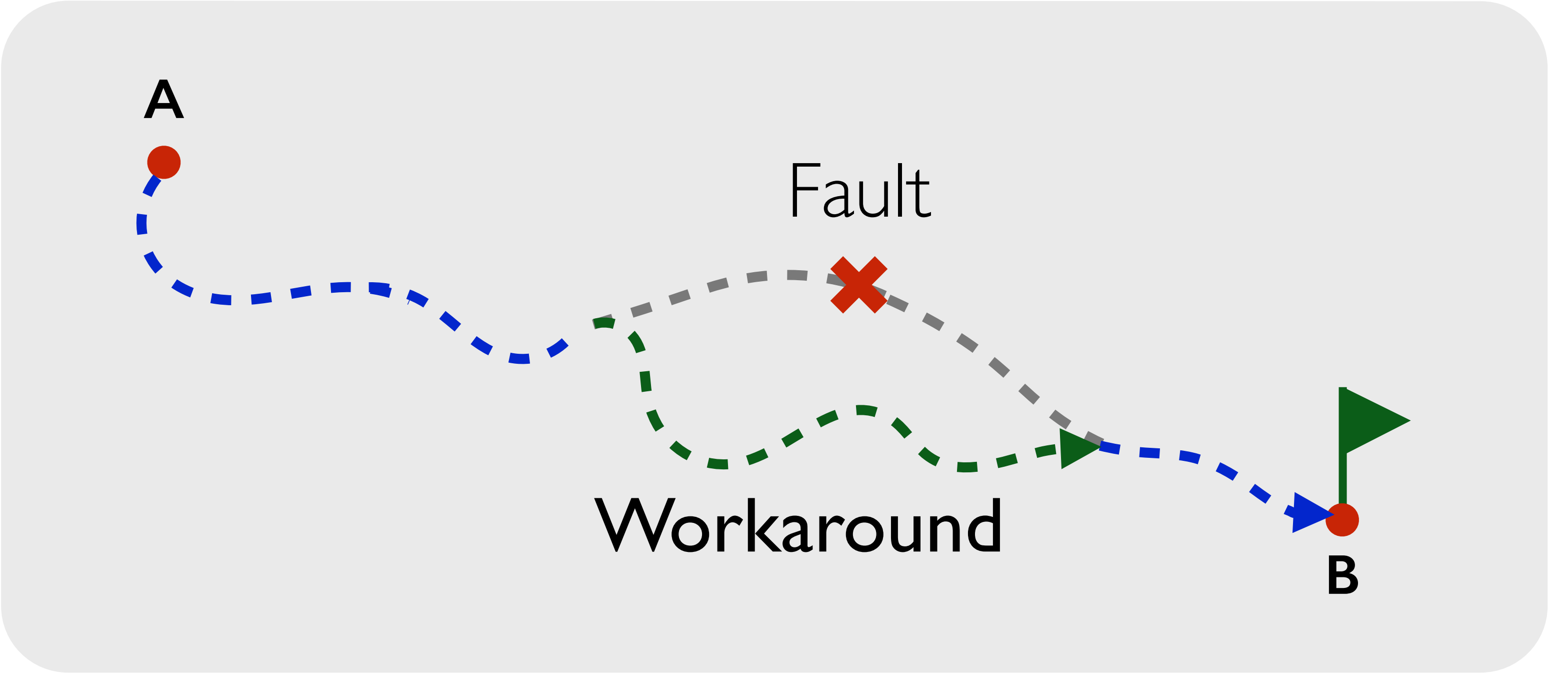
Predictive Ability

Predictive Ability



Automatic workarounds

Predictive Ability



Automatic workarounds

Does redundancy correlate with success?

Predictive Ability

System

Caliper

Carrot2

Predictive Ability

System	Method (C_A)	Workaround (C_B)
Caliper	Iterators.forArray(a)	Arrays.asList(a).iterator()
	LinkedHashMultiset.retainAll(Collection c)	foreach(o in m) if(o not in c) m.remove(o);
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);
	LinkedHashMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);
	LinkedHashMultimap.create()	create(100,100)
	LinkedHashMultimap.create(int,int)	create()
	LinkedHashMultimap.isEmpty()	size() == 0 ? true : false
Carrot2	ImmutableMultiset.of(Object..c)	foreach(o in c) build().setCount(o,count(o))
	ImmutableMultiset.of(Object..c)	builder().add(..c).build()
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);
	ImmutableMultiset.of(Object o)	builder().add(o).build()
	Lists.newArrayList()	new ArrayList()
	Lists.newArrayList()	new ArrayList(10)
	Lists.newArrayListWithCapacity(int c)	new ArrayList()
	Lists.newArrayListWithCapacity(int c)	new ArrayList(c)
	Maps.newHashMap()	Maps.newHashMapWithExpectedSize(16)
	Maps.newHashMap()	new HashMap()
	Maps.newHashMap()	new HashMap(16)

Predictive Ability

System	Method (C_A)	Workaround (C_B)	Success Rate
Caliper	Iterators.forArray(a)	Arrays.asList(a).iterator()	3/3 (100%)
	LinkedHashMultiset.retainAll(Collection c)	foreach(o in m) if(o not in c) m.remove(o);	1/2 (50%)
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	8/41 (20%)
	LinkedHashMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	0/1 (0%)
	LinkedHashMultimap.create()	create(100,100)	0/207 (0%)
	LinkedHashMultimap.create(int,int)	create()	0/202 (0%)
	LinkedHashMultimap.isEmpty()	size() == 0 ? true : false	0/34 (0%)
Carrot2	ImmutableMultiset.of(Object..c)	foreach(o in c) build().setCount(o,count(o))	13/22 (59%)
	ImmutableMultiset.of(Object..c)	builder().add(..c).build()	7/19 (37%)
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	1/13 (8%)
	ImmutableMultiset.of(Object o)	builder().add(o).build()	0/1 (0%)
	Lists.newArrayList()	new ArrayList()	0/24 (0%)
	Lists.newArrayList()	new ArrayList(10)	0/24 (0%)
	Lists.newArrayListWithCapacity(int c)	new ArrayList()	0/20 (0%)
	Lists.newArrayListWithCapacity(int c)	new ArrayList(c)	0/20 (0%)
	Maps.newHashMap()	Maps.newHashMapWithExpectedSize(16)	0/54 (0%)
	Maps.newHashMap()	new HashMap()	0/54 (0%)
	Maps.newHashMap()	new HashMap(16)	0/54 (0%)

Predictive Ability

System	Method (C_A)	Workaround (C_B)	Success Rate	Redundancy
Caliper	Iterators.forArray(a)	Arrays.asList(a).iterator()	3/3 (100%)	1.00 ± 0.00
	LinkedHashMultiset.retainAll(Collection c)	foreach(o in m) if(o not in c) m.remove(o);	1/2 (50%)	0.61 ± 0.01
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	8/41 (20%)	0.37 ± 0.32
	LinkedHashMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	0/1 (0%)	0.00 ± 0.00
	LinkedHashMultimap.create()	create(100,100)	0/207 (0%)	0.12 ± 0.15
	LinkedHashMultimap.create(int,int)	create()	0/202 (0%)	0.12 ± 0.15
	LinkedHashMultimap.isEmpty()	size() == 0 ? true : false	0/34 (0%)	0.00 ± 0.00
Carrot2	ImmutableMultiset.of(Object..c)	foreach(o in c) build().setCount(o,count(o))	13/22 (59%)	0.56 ± 0.07
	ImmutableMultiset.of(Object..c)	builder().add(..c).build()	7/19 (37%)	0.24 ± 0.12
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	1/13 (8%)	0.37 ± 0.32
	ImmutableMultiset.of(Object o)	builder().add(o).build()	0/1 (0%)	0.32 ± 0.14
	Lists.newArrayList()	new ArrayList()	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayList()	new ArrayList(10)	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList()	0/20 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList(c)	0/20 (0%)	0.00 ± 0.00
	Maps.newHashMap()	Maps.newHashMapWithExpectedSize(16)	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap()	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap(16)	0/54 (0%)	0.00 ± 0.00

Predictive Ability

System	Method (C_A)	Workaround (C_B)	Success Rate	Redundancy
Caliper	Iterators.forArray(a)	Arrays.asList(a).iterator()	3/3 (100%)	1.00 ± 0.00
	LinkedHashMultiset.retainAll(Collection c)	foreach(o in m) if(o not in c) m.remove(o);	1/2 (50%)	0.61 ± 0.01
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	8/41 (20%)	0.37 ± 0.32
	LinkedHashMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	0/1 (0%)	0.00 ± 0.00
	LinkedHashMultimap.create()	create(100,100)	0/207 (0%)	0.12 ± 0.15
	LinkedHashMultimap.create(int,int)	create()	0/202 (0%)	0.12 ± 0.15
	LinkedHashMultimap.isEmpty()	size() == 0 ? true : false	0/34 (0%)	0.00 ± 0.00
Carrot2	ImmutableMultiset.of(Object..c)	foreach(o in c) build().setCount(o,count(o))	13/22 (59%)	0.56 ± 0.07
	ImmutableMultiset.of(Object..c)	builder().add(..c).build()	7/19 (37%)	0.24 ± 0.12
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	1/13 (8%)	0.37 ± 0.32
	ImmutableMultiset.of(Object o)	builder().add(o).build()	0/1 (0%)	0.32 ± 0.14
	Lists.newArrayList()	new ArrayList()	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayList()	new ArrayList(10)	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList()	0/20 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList(c)	0/20 (0%)	0.00 ± 0.00
	Maps.newHashMap()	Maps.newHashMapWithExpectedSize(16)	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap()	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap(16)	0/54 (0%)	0.00 ± 0.00

Predictive Ability

System	Method (C_A)	Workaround (C_B)	Success Rate	Redundancy
Caliper	Iterators.forArray(a)	Arrays.asList(a).iterator()	3/3 (100%)	1.00 ± 0.00
	LinkedHashMultiset.retainAll(Collection c)	foreach(o in m) if(o not in c) m.remove(o);	1/2 (50%)	0.61 ± 0.01
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	8/41 (20%)	0.37 ± 0.32
	LinkedHashMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	0/1 (0%)	0.00 ± 0.00
	LinkedHashMultimap.create()	create(100,100)	0/207 (0%)	0.12 ± 0.15
	LinkedHashMultimap.create(int,int)	create()	0/202 (0%)	0.12 ± 0.15
	LinkedHashMultimap.isEmpty()	size() == 0 ? true : false	0/34 (0%)	0.00 ± 0.00
Carrot2	ImmutableMultiset.of(Object..c)	foreach(o in c) build().setCount(o,count(o))	13/22 (59%)	0.56 ± 0.07
	ImmutableMultiset.of(Object..c)	builder().add(..c).build()	7/19 (37%)	0.24 ± 0.12
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	1/13 (8%)	0.37 ± 0.32
	ImmutableMultiset.of(Object o)	builder().add(o).build()	0/1 (0%)	0.32 ± 0.14
	Lists.newArrayList()	new ArrayList()	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayList()	new ArrayList(10)	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList()	0/20 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList(c)	0/20 (0%)	0.00 ± 0.00
	Maps.newHashMap()	Maps.newHashMapWithExpectedSize(16)	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap()	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap(16)	0/54 (0%)	0.00 ± 0.00

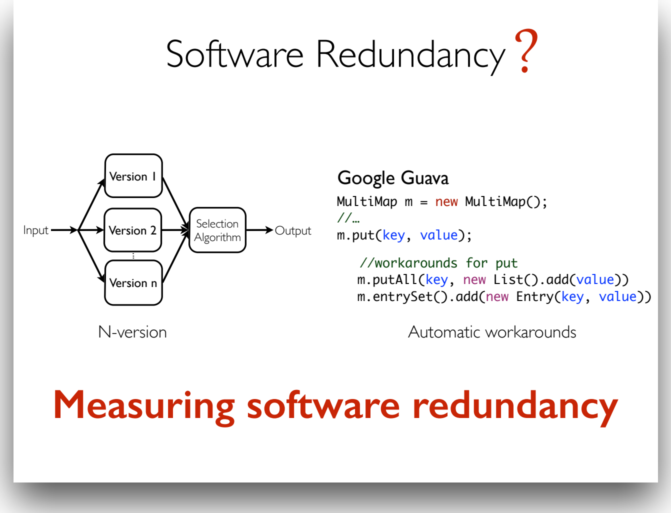
Predictive Ability

System	Method (C_A)	Workaround (C_B)	Success Rate		Redundancy
Caliper	Iterators.forArray(a)	Arrays.asList(a).iterator()	3/3	(100%)	1.00 ± 0.00
	LinkedHashMultiset.retainAll(Collection c)	foreach(o in m) if(o not in c) m.remove(o);	1/2	(50%)	0.61 ± 0.01
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	8/41	(20%)	0.37 ± 0.32
	LinkedHashMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	0/1	(0%)	0.00 ± 0.00
	LinkedHashMultimap.create()	create(100,100)	0/207	(0%)	0.12 ± 0.15
	LinkedHashMultimap.create(int,int)	create()	0/202	(0%)	0.12 ± 0.15
	LinkedHashMultimap.isEmpty()	size() == 0 ? true : false	0/34	(0%)	0.00 ± 0.00
Carrot2	ImmutableMultiset.of(Object..c)	foreach(o in c) build().setCount(o,count(o))	13/22	(59%)	0.56 ± 0.07
	ImmutableMultiset.of(Object..c)	builder().add(..c).build()	7/19	(37%)	0.24 ± 0.12
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	1/13	(8%)	0.37 ± 0.32
	ImmutableMultiset.of(Object o)	builder().add(o).build()	0/1	(0%)	0.32 ± 0.14
	Lists.newArrayList()	new ArrayList()	0/24	(0%)	0.00 ± 0.00
	Lists.newArrayList()	new ArrayList(10)	0/24	(0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList()	0/20	(0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList(c)	0/20	(0%)	0.00 ± 0.00
	Maps.newHashMap()	Maps.newHashMapWithExpectedSize(16)	0/54	(0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap()	0/54	(0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap(16)	0/54	(0%)	0.00 ± 0.00

Predictive Ability

System	Method (C_A)	Workaround (C_B)	Success Rate	Redundancy
Caliper	Iterators.forArray(a)	Arrays.asList(a).iterator()	3/3 (100%)	1.00 ± 0.00
	LinkedHashMultiset.retainAll(Collection c)	foreach(o in m) if(o not in c) m.remove(o);	1/2 (50%)	0.61 ± 0.01
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	8/41 (20%)	0.37 ± 0.32
	LinkedHashMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	0/1 (0%)	0.00 ± 0.00
	LinkedHashMultimap.cre		0/207 (0%)	0.12 ± 0.15
	LinkedHashMultimap.cre		0/202 (0%)	0.12 ± 0.15
	LinkedHashMultimap.isE		0/34 (0%)	0.00 ± 0.00
Carrot2	ImmutableMultiset.of(Ob	(o))	13/22 (59%)	0.56 ± 0.07
	ImmutableMultiset.of(Ob		7/19 (37%)	0.24 ± 0.12
	ArrayListMultimap.putAll		1/13 (8%)	0.37 ± 0.32
	ImmutableMultiset.of(Object o)	builder().add(o).build()	0/1 (0%)	0.32 ± 0.14
	Lists.newArrayList()	new ArrayList()	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayList()	new ArrayList(10)	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList()	0/20 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList(c)	0/20 (0%)	0.00 ± 0.00
	Maps.newHashMap()	Maps.newHashMapWithExpectedSize(16)	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap()	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap(16)	0/54 (0%)	0.00 ± 0.00

Correlation: 0.94



Software Redundancy ?

Input
Version 1
Version 2
Version n
Selection Algorithm
Output

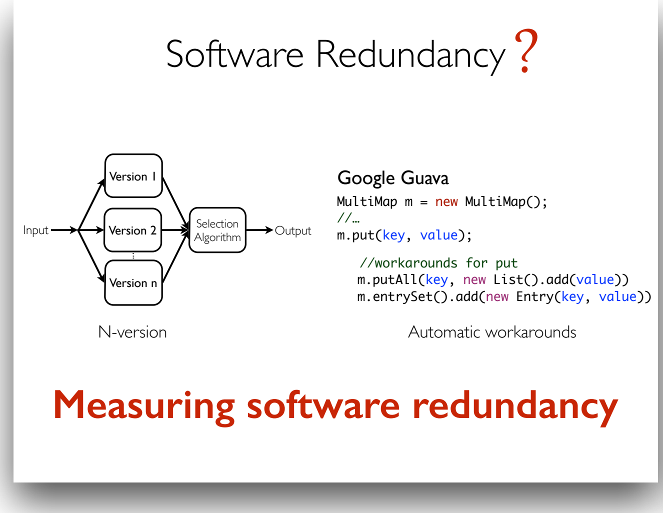
N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
   m.putAll(key, new List().add(value))
   m.entrySet().add(new Entry(key, value))
```

Automatic workarounds

Measuring software redundancy



Software Redundancy ?

Input
Version 1
Version 2
Version n
Selection Algorithm
Output

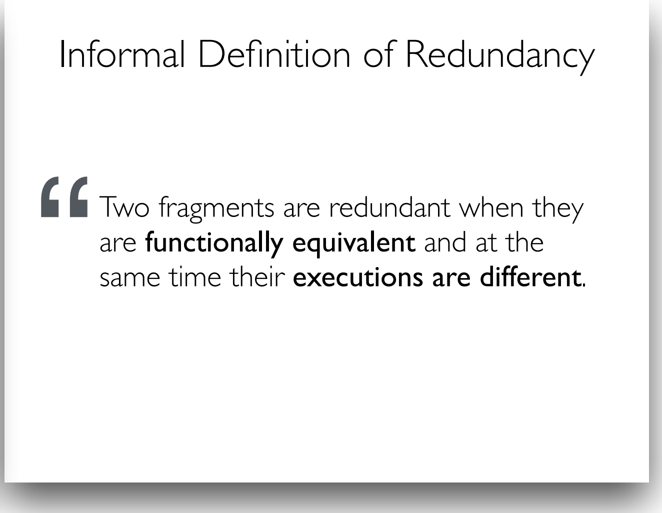
N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
   m.putAll(key, new List().add(value))
   m.entrySet().add(new Entry(key, value))
```

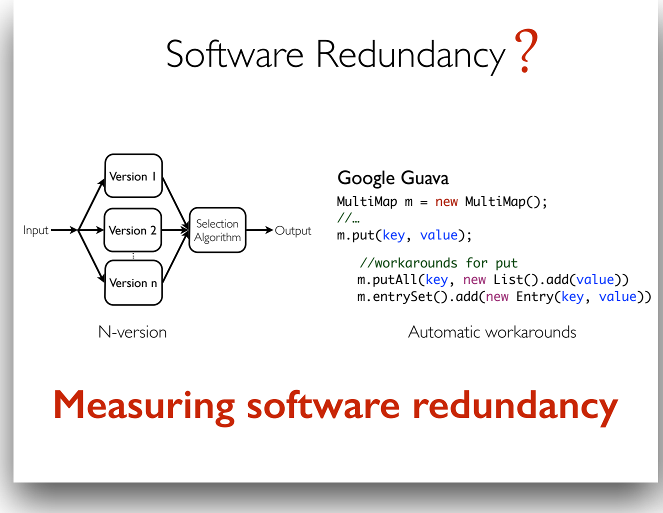
Automatic workarounds

Measuring software redundancy



Informal Definition of Redundancy

> “Two fragments are redundant when they are **functionally equivalent** and at the same time their **executions are different**.



Software Redundancy ?

Input
Version 1
Version 2
Version n
Selection Algorithm
Output

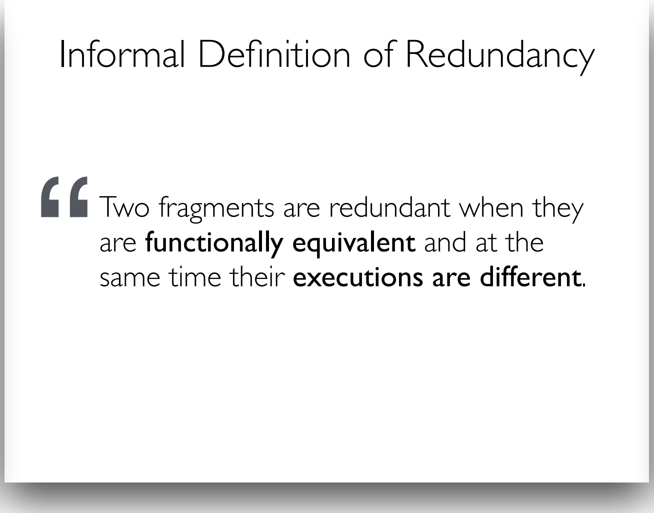
N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
   m.putAll(key, new List().add(value))
   m.entrySet().add(new Entry(key, value))
```

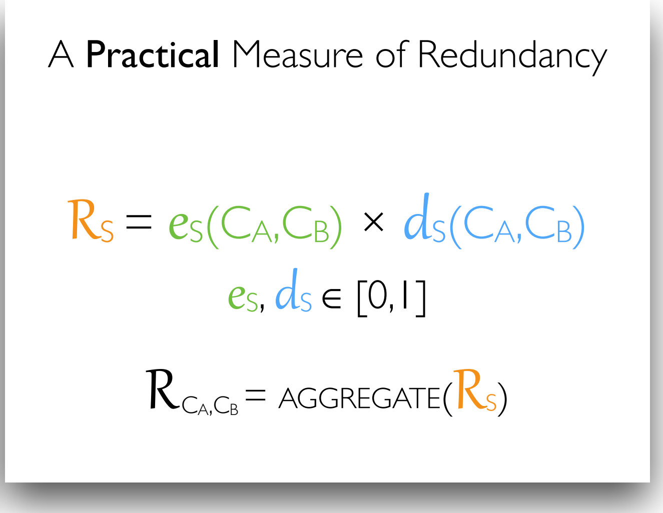
Automatic workarounds

Measuring software redundancy



Informal Definition of Redundancy

> “Two fragments are redundant when they are **functionally equivalent** and at the same time their **executions are different**.

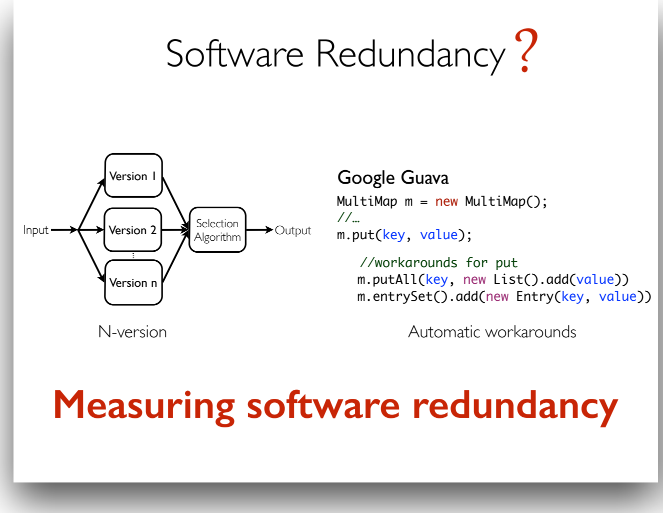


A **Practical** Measure of Redundancy

$$R_S = e_S(C_A,C_B) \times d_S(C_A,C_B)$$

$$e_S, d_S \in [0,1]$$

$$R_{C_A,C_B} = \text{AGGREGATE}(R_S)$$



Software Redundancy ?

Input
Version 1
Version 2
Version n
Selection Algorithm
Output

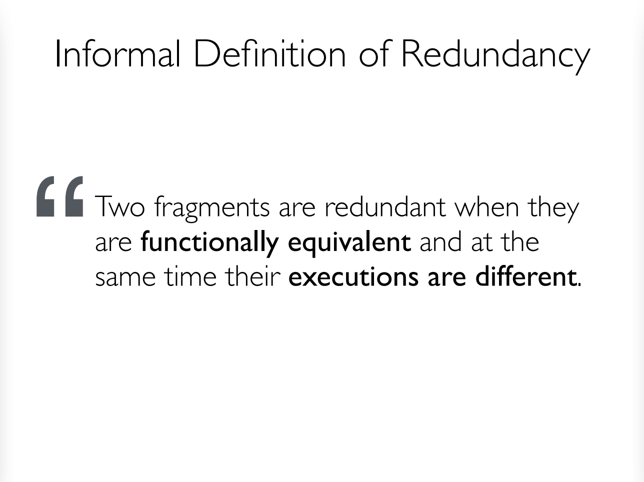
N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
   m.putAll(key, new List().add(value))
   m.entrySet().add(new Entry(key, value))
```

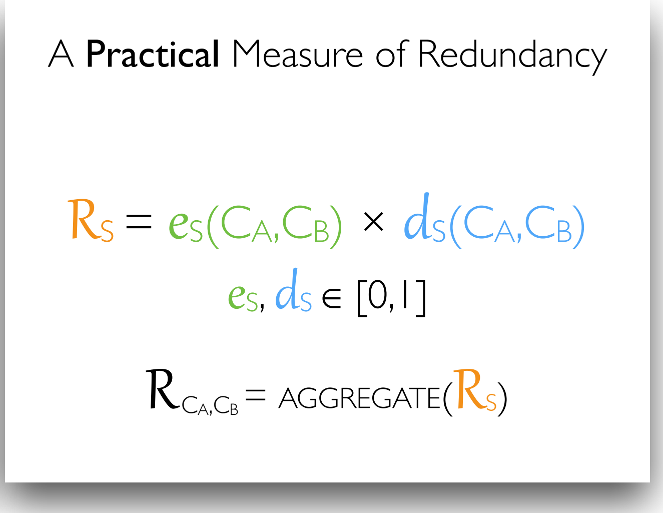
Automatic workarounds

Measuring software redundancy



Informal Definition of Redundancy

> “Two fragments are redundant when they are **functionally equivalent** and at the same time their **executions are different**.

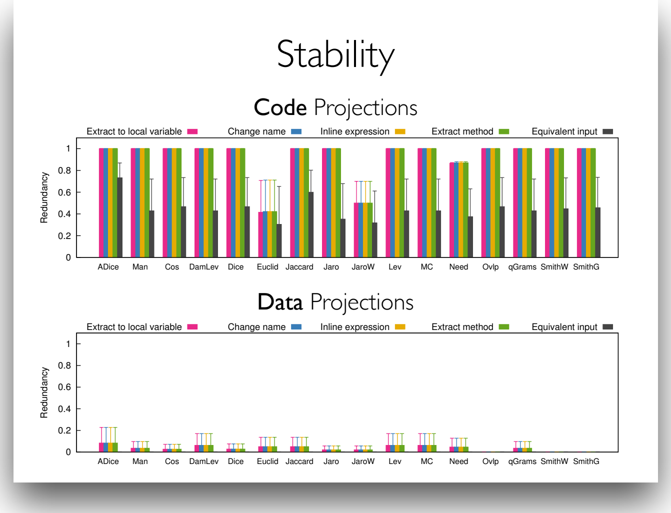


A **Practical** Measure of Redundancy

$$R_S = e_S(C_A,C_B) \times d_S(C_A,C_B)$$

$$e_S, d_S \in [0,1]$$

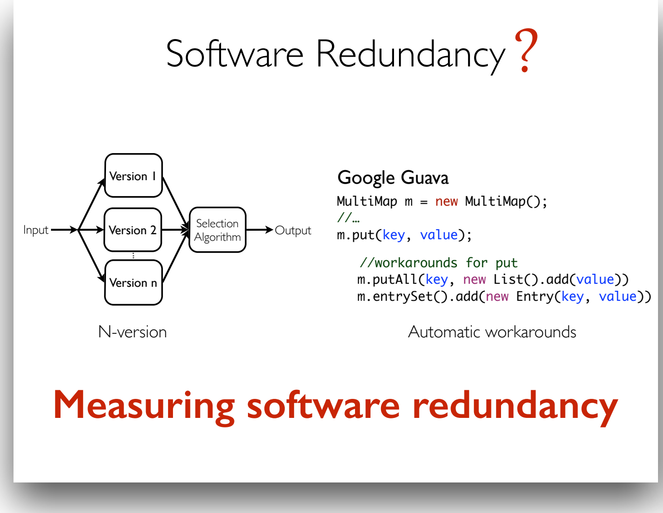
$$R_{C_A,C_B} = \text{AGGREGATE}(R_S)$$



Stability

Code Projections

Data Projections



Software Redundancy ?

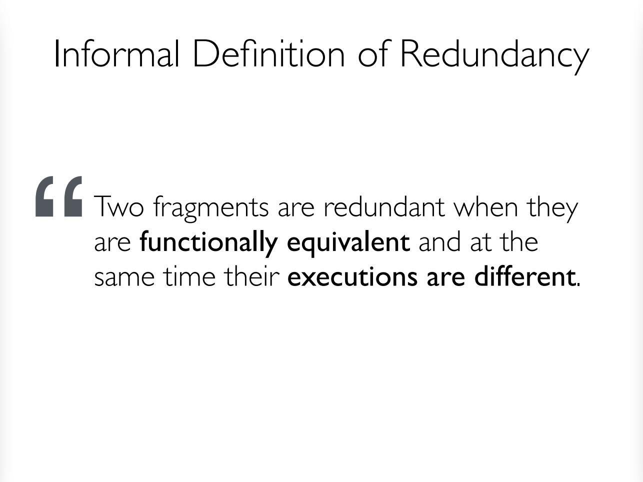
N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
   m.putAll(key, new List().add(value))
   m.entrySet().add(new Entry(key, value))
```

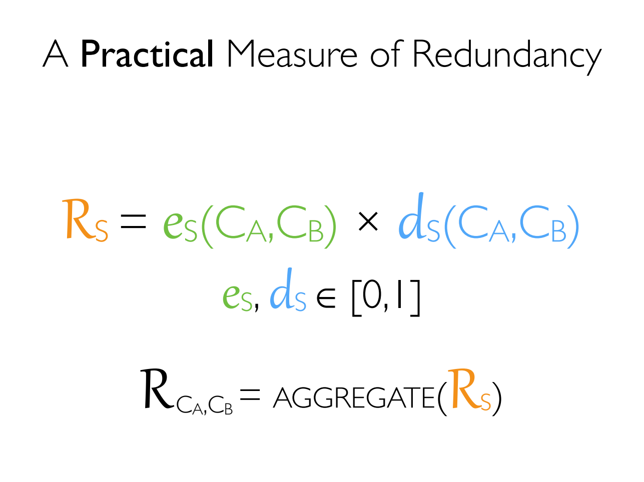
Automatic workarounds

Measuring software redundancy



Informal Definition of Redundancy

> “Two fragments are redundant when they are **functionally equivalent** and at the same time their **executions are different**.

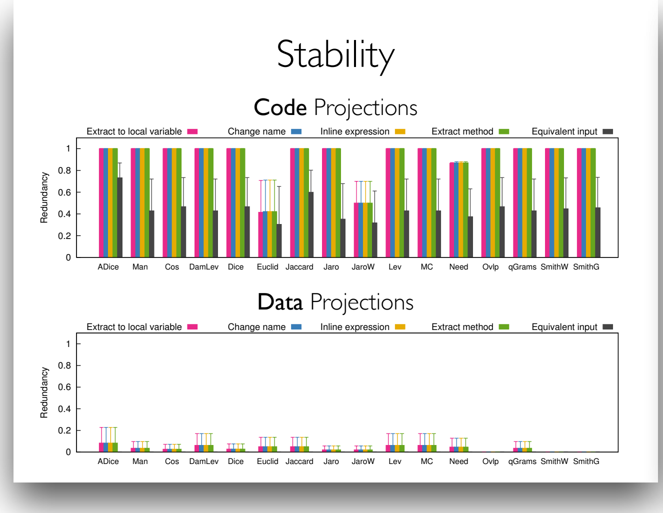


A **Practical** Measure of Redundancy

$$R_S = e_S(C_A,C_B) \times d_S(C_A,C_B)$$

$$e_S, d_S \in [0,1]$$

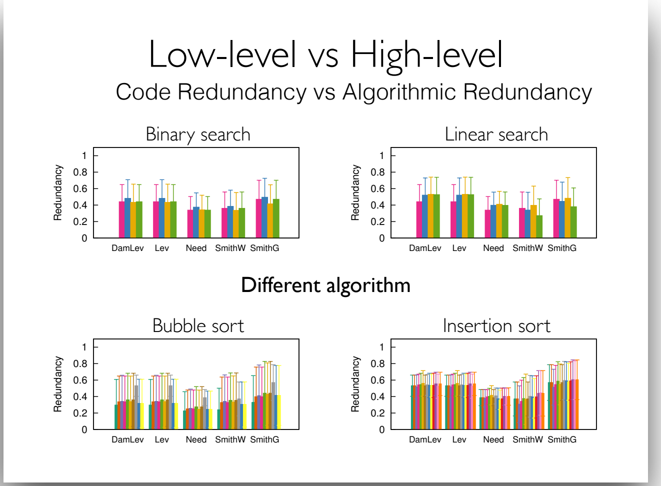
$$R_{C_A,C_B} = \text{AGGREGATE}(R_S)$$



Stability

Code Projections

Data Projections



Low-level vs High-level

Code Redundancy vs Algorithmic Redundancy

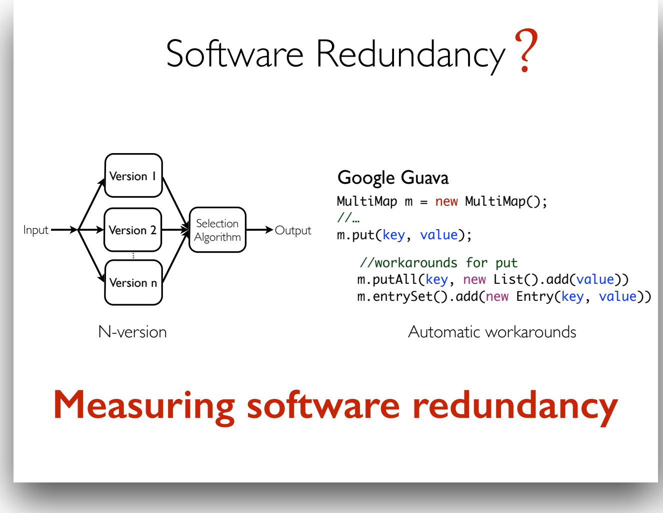
Binary search

Linear search

Different algorithm

Bubble sort

Insertion sort



Software Redundancy ?

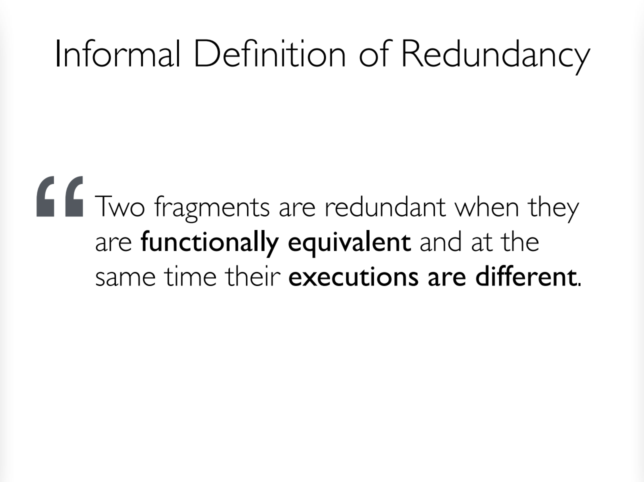
N-version

Google Guava

```
MultiMap m = new MultiMap();
//…
m.put(key, value);

   //workarounds for put
   m.putAll(key, new List().add(value))
   m.entrySet().add(new Entry(key, value))
```

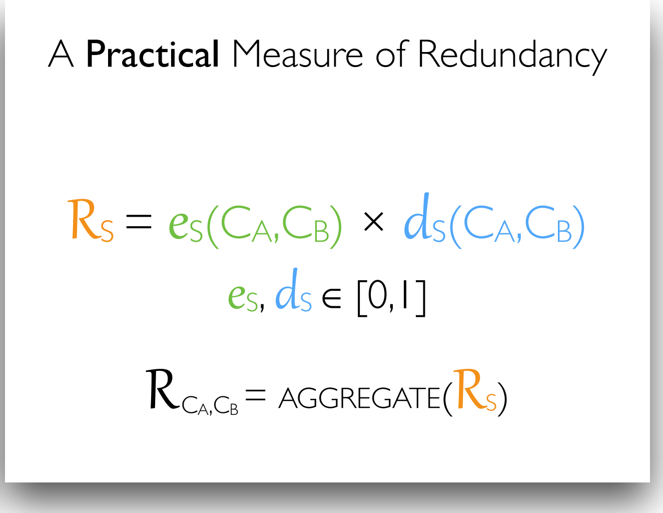
Automatic workarounds

Measuring software redundancy



Informal Definition of Redundancy

> “Two fragments are redundant when they are **functionally equivalent** and at the same time their **executions are different**.

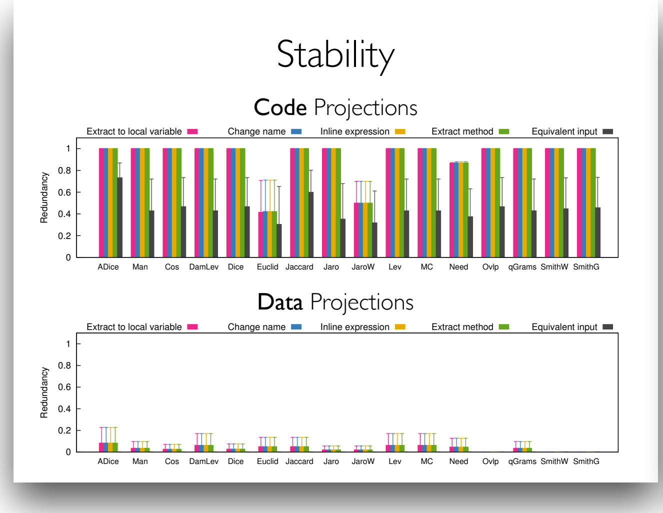


A **Practical** Measure of Redundancy

$$R_S = e_S(C_A,C_B) \times d_S(C_A,C_B)$$

$$e_S, d_S \in [0,1]$$

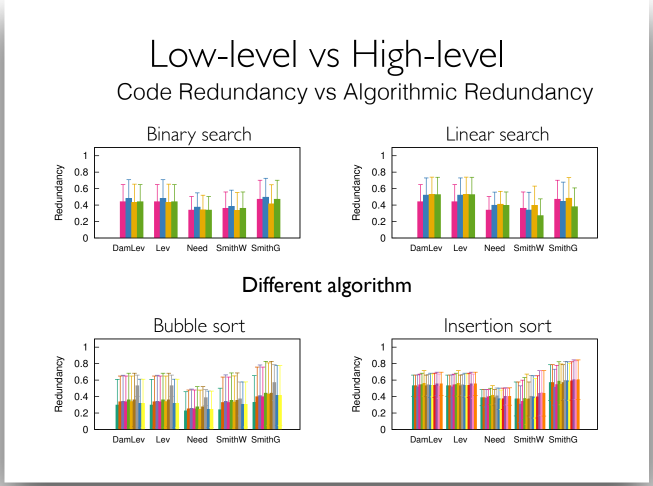
$$R_{C_A,C_B} = \mathrm{AGGREGATE}(R_S)$$



Stability

Code Projections

Data Projections



Low-level vs High-level

Code Redundancy vs Algorithmic Redundancy

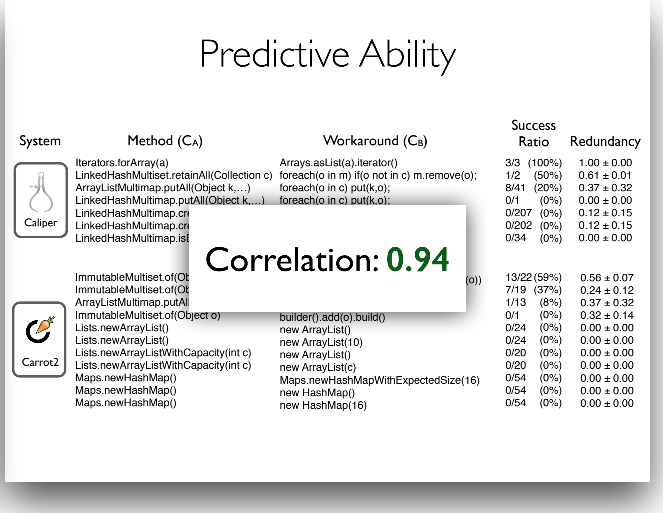
Binary search

Linear search

Different algorithm

Bubble sort

Insertion sort



Predictive Ability

System	Method (C_A)	Workaround (C_B)	Success Ratio	Redundancy
Caliper	Iterators.forArray(a)	Arrays.asList(a).iterator()	3/3 (100%)	1.00 ± 0.00
	LinkedHashMultiset.retainAll(Collection c)	foreach(o in m) if(o not in c) m.remove(o);	1/2 (50%)	0.61 ± 0.01
	ArrayListMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	8/41 (20%)	0.37 ± 0.32
	LinkedHashMultimap.putAll(Object k,…)	foreach(o in c) put(k,o);	0/1 (0%)	0.00 ± 0.00
	LinkedHashMultimap.cr		0/207 (0%)	0.12 ± 0.15
	LinkedHashMultimap.cr		0/202 (0%)	0.12 ± 0.15
	LinkedHashMultimap.isE		0/34 (0%)	0.00 ± 0.00
Carrot2	ImmutableMultiset.of(Ob	(o))	13/22 (59%)	0.56 ± 0.07
	ImmutableMultiset.of(Ob		7/19 (37%)	0.24 ± 0.12
	ArrayListMultimap.putAll		1/13 (8%)	0.37 ± 0.32
	ImmutableMultiset.of(Object o)	builder().add(o).build()	0/1 (0%)	0.32 ± 0.14
	Lists.newArrayList()	new ArrayList()	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayList()	new ArrayList(10)	0/24 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList()	0/20 (0%)	0.00 ± 0.00
	Lists.newArrayListWithCapacity(int c)	new ArrayList(c)	0/20 (0%)	0.00 ± 0.00
	Maps.newHashMap()	Maps.newHashMapWithExpectedSize(16)	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap()	0/54 (0%)	0.00 ± 0.00
	Maps.newHashMap()	new HashMap(16)	0/54 (0%)	0.00 ± 0.00

Correlation: **0.94**