Università
della
Svizzera
italiana

**Faculty
of Informatics**

Using the Intrinsic Redundancy of Software

Andrea Mattavelli

In collaboration with:
Antonio Carzaniga, Alessandra Gorla, Alberto Goffi, Nicolò Perino, Mauro Pezzè

Redundancy

"Informally, a system is redundant when it is able to do the **same thing** in **different ways**."

Redundancy

"Informally, a system is redundant when it is able to do the **same thing** in **different ways**."

same functionality

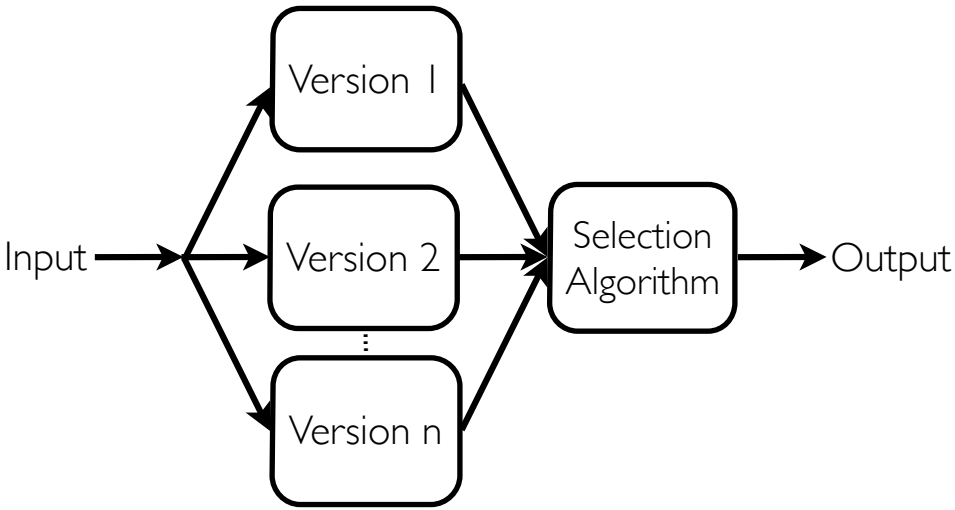
Redundancy

"Informally, a system is redundant when it is able to do the **same thing** in **different ways**."

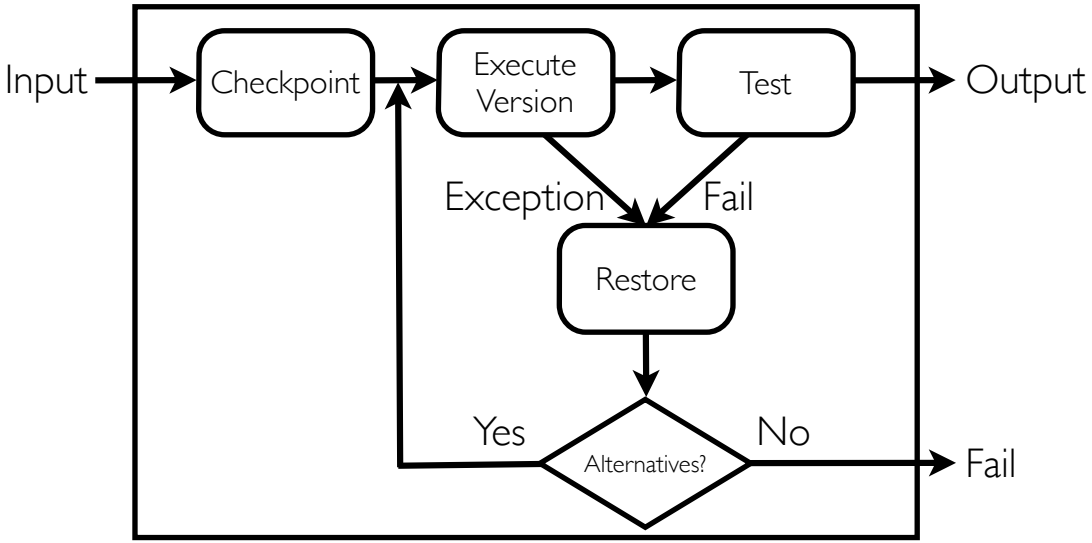
same functionality

different code

Software Redundancy

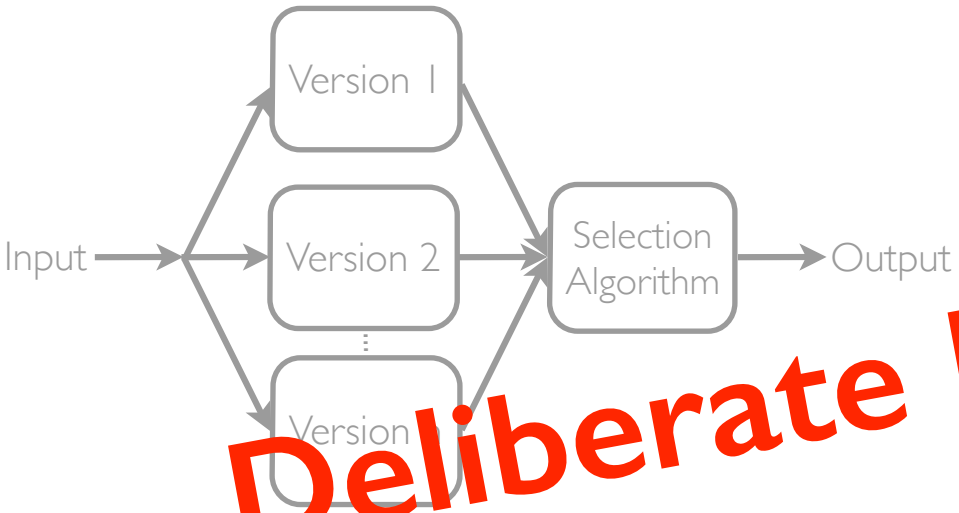


N-version

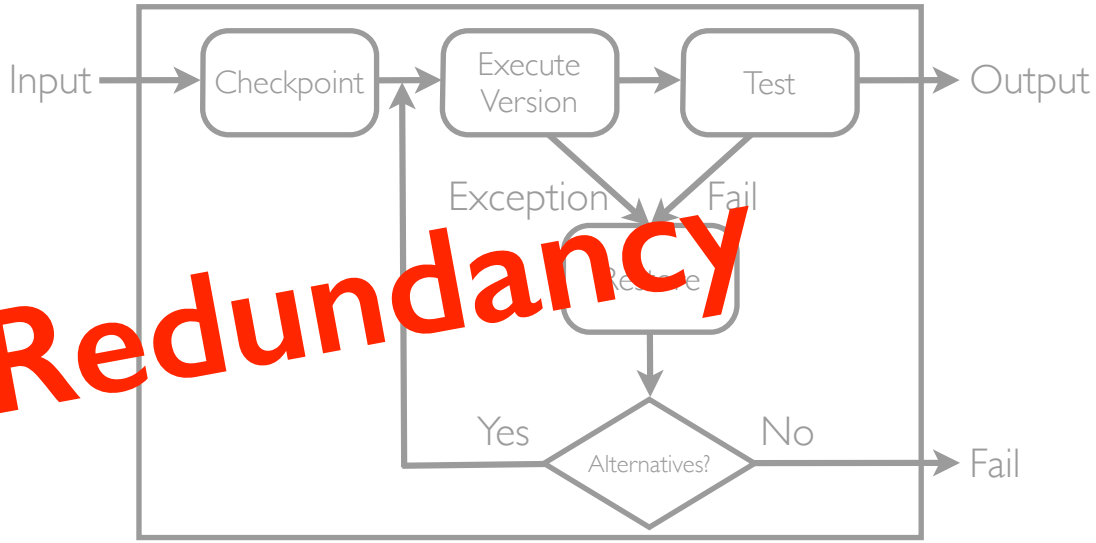


Recovery Blocks

Software Redundancy



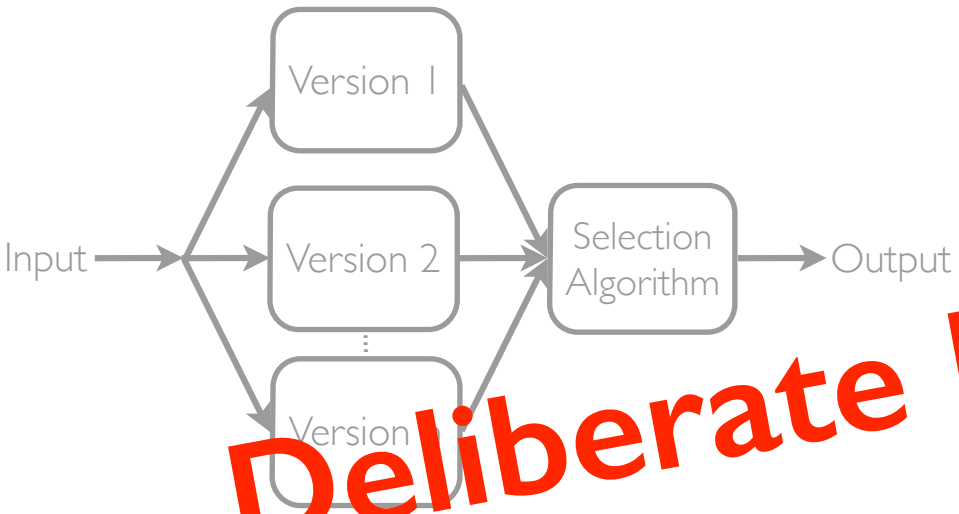
N-version



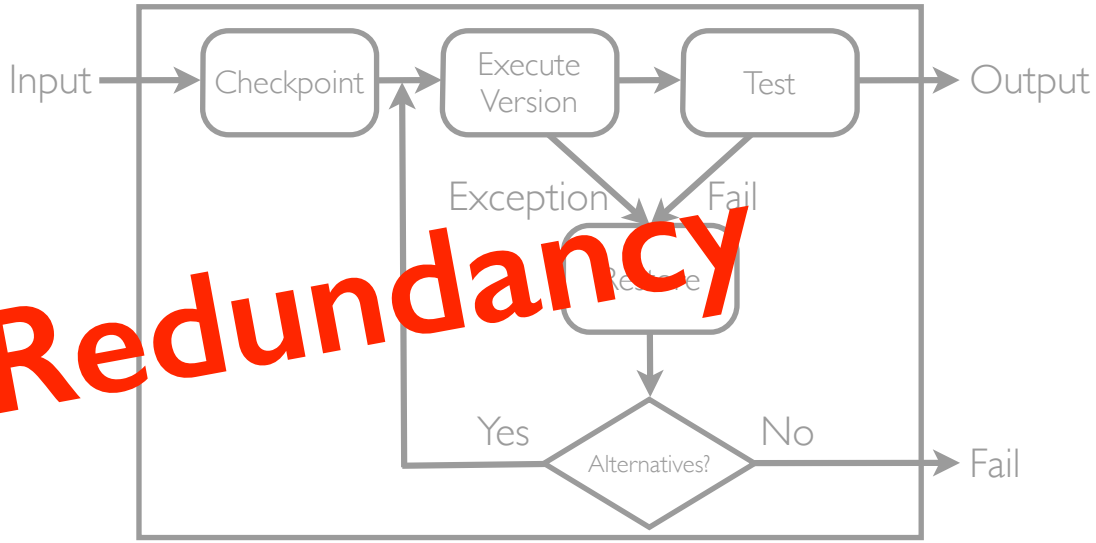
Recovery Blocks

Deliberate Redundancy

Software Redundancy



N-version

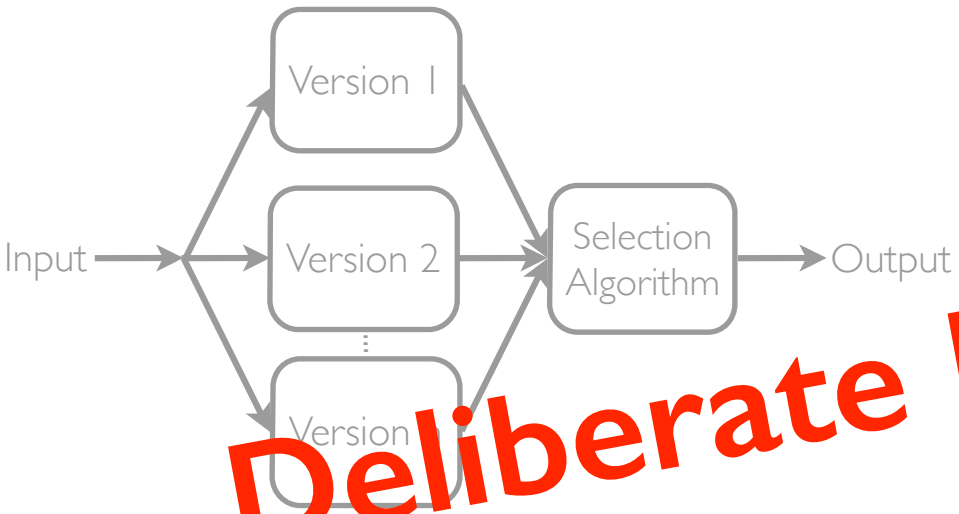


Recovery Blocks

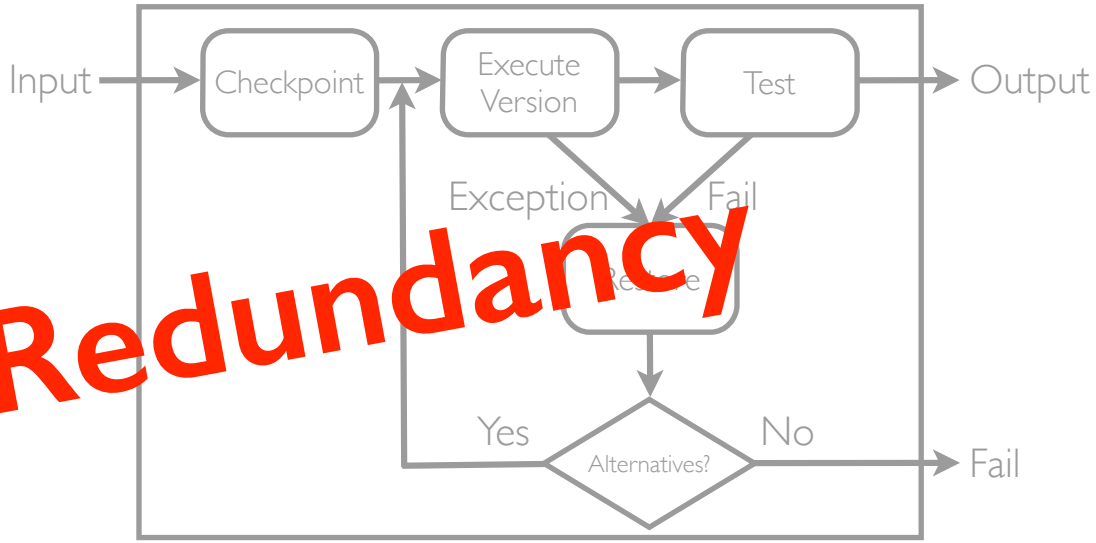
Deliberate Redundancy

High development cost

Software Redundancy



N-version

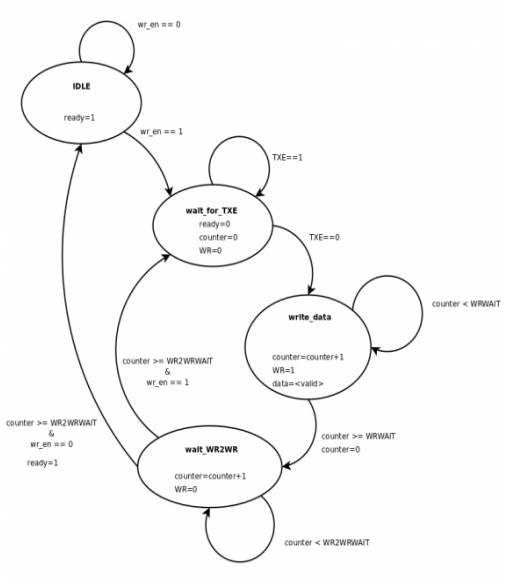


Recovery Blocks

Deliberate Redundancy

High development cost
(..and maybe ineffective!)

Software Redundancy



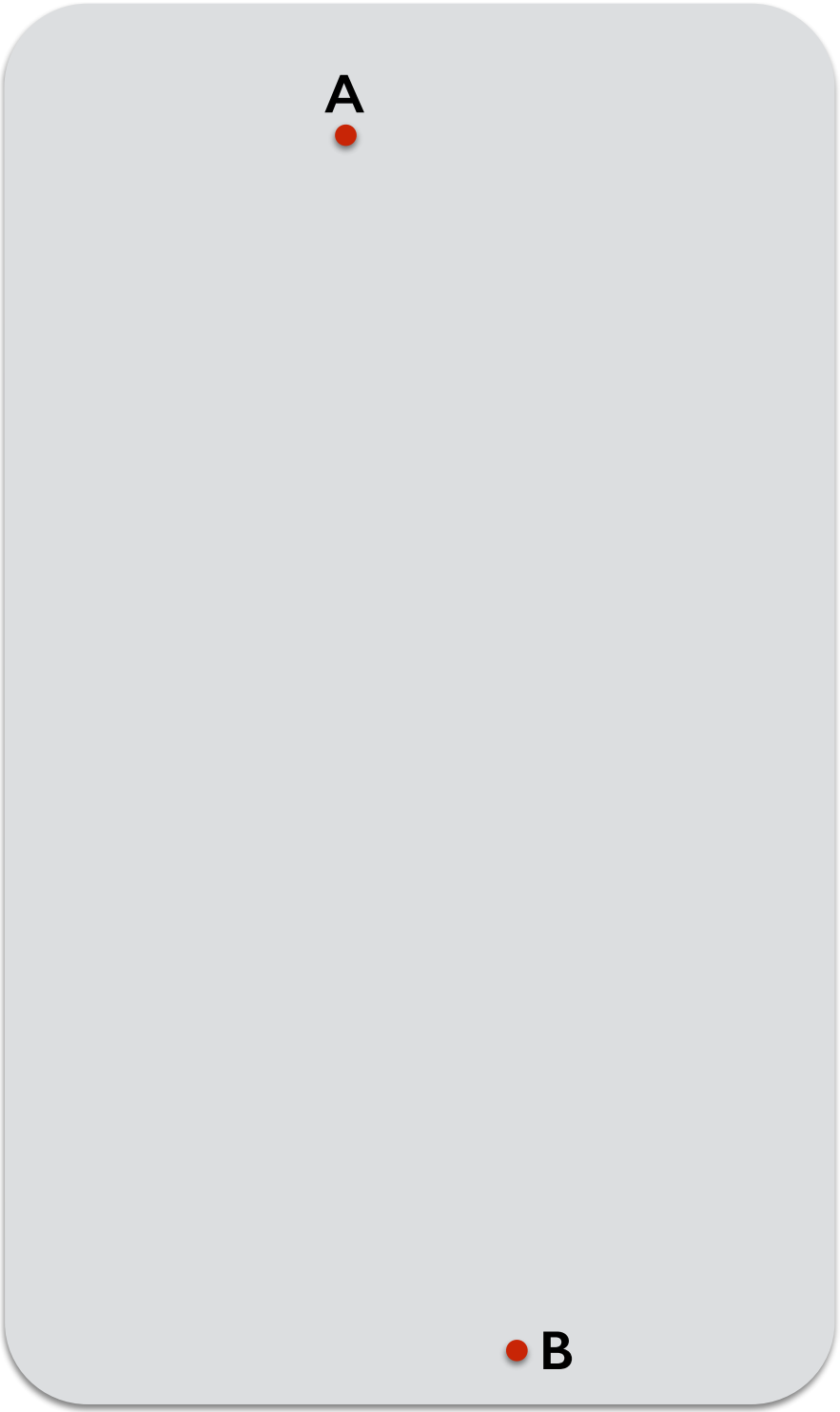
Behavioral models

```
//@ requires 0 < amount &&
      amount + balance < MAX_BALANCE;
//@ ensures balance == \old(balance) + amount;

public void credit(final int amount) {
   this.balance += amount;
}


//@ requires 0 < amount && amount <= balance;
//@ ensures balance == \old(balance) - amount;

public void debit(final int amount) {
   this.balance -= amount;
}
```

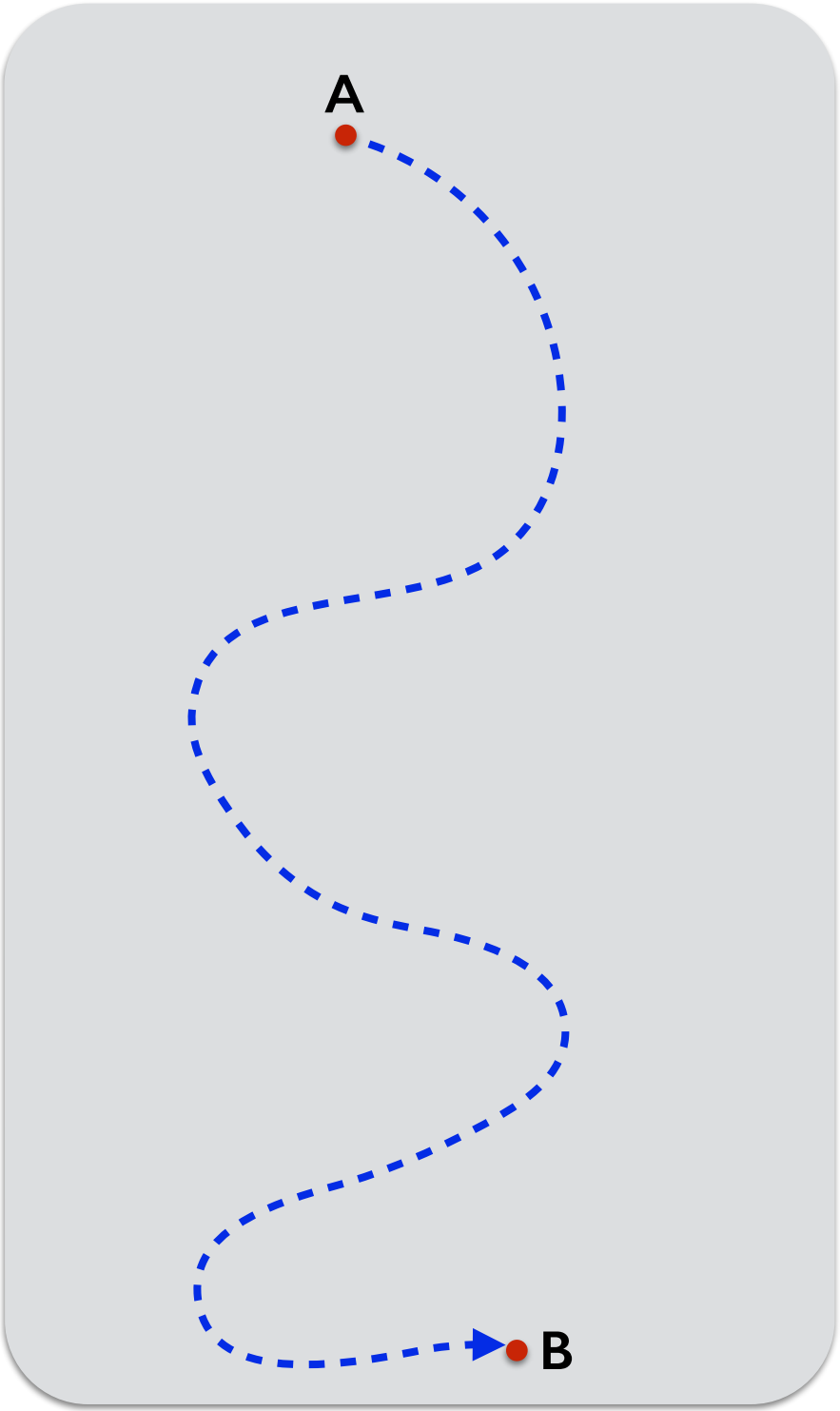
Invariants, contracts

```
assert x > 0;

assert mutex == null;

assert parameter != null;

…
```

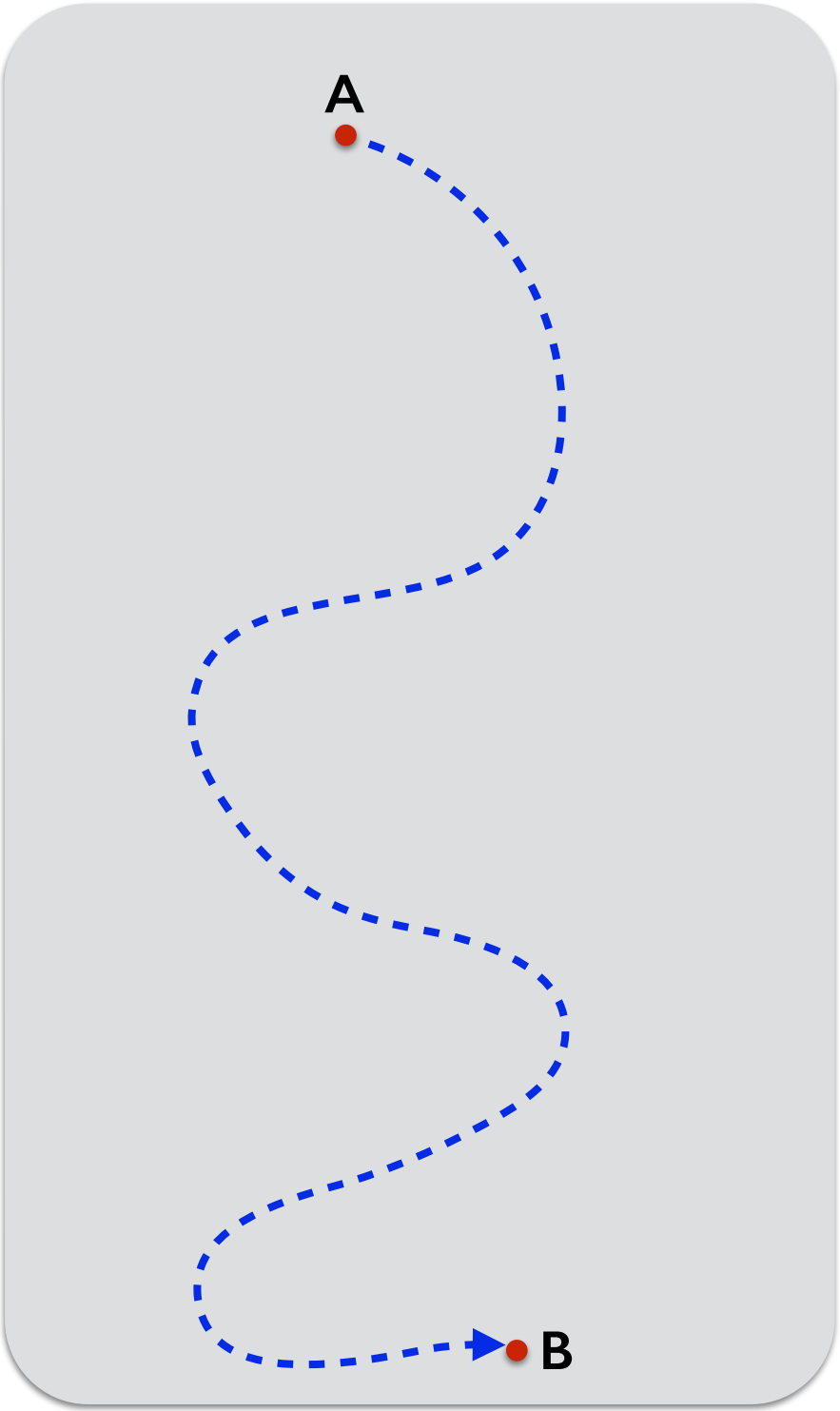
Assertions



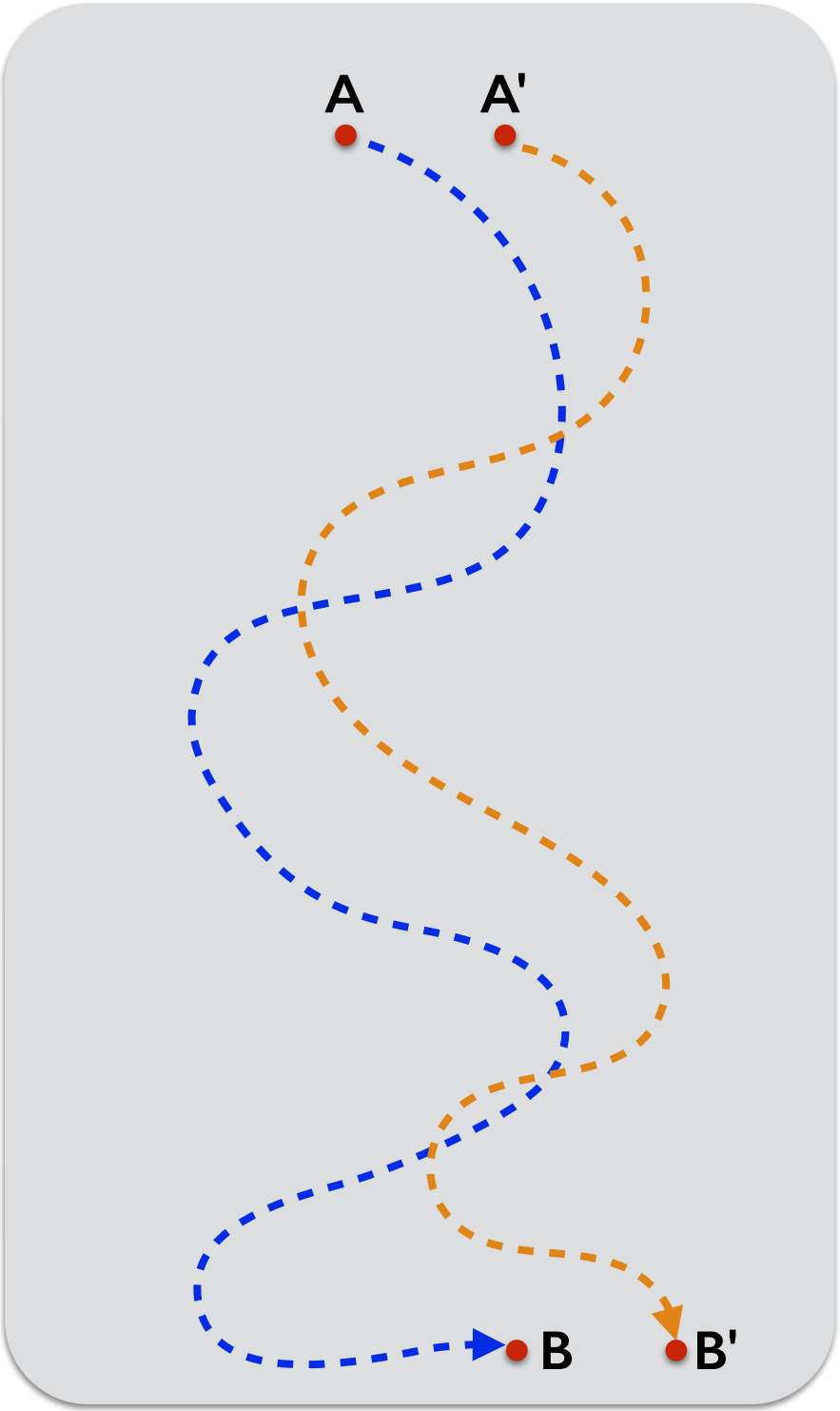

```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            return 1;
        }
        if (keys[h] == k)
            return ++counters[h];

        h = (h + step) & size_mask;
    } while (true);
}

private void create_table(int size) {
    table_size = size;
    keys = new SFFFilter[size];
    counters = new int[size];
    size_mask = size - 1;
}

private void rehash() {
    int [] old_counters = counters;
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```



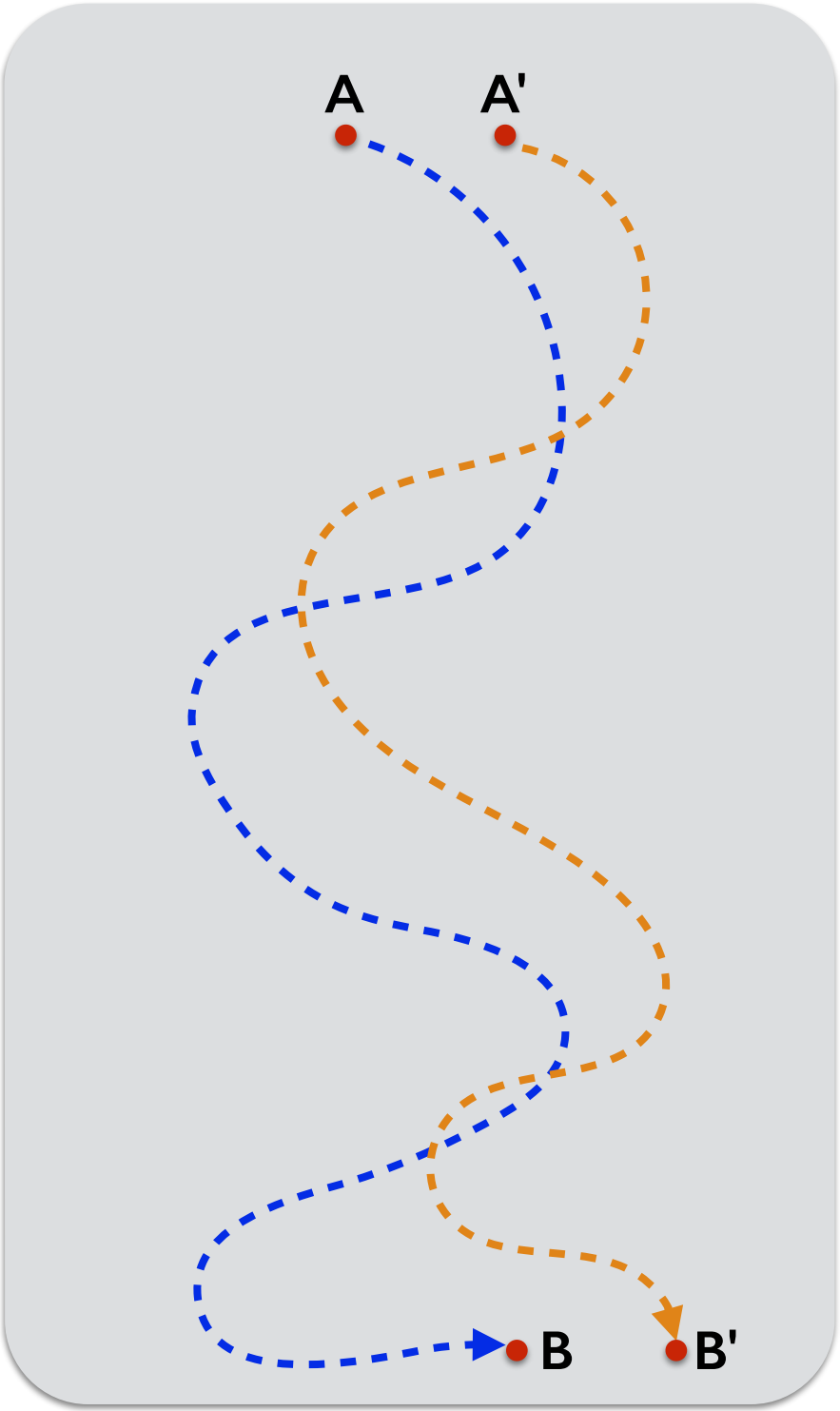

```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            return 1;
        }
        if (keys[h] == k)
            return ++counters[h];

        h = (h + step) & size_mask;
    } while (true);
}

private void create_table(int size) {
    table_size = size;
    keys = new SFFFilter[size];
    counters = new int[size];
    size_mask = size - 1;
}

private void rehash() {
    int [] old_counters = counters;
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```




```
private int size_mask;

pub
}
pri
}

private void cp_save_object_field(Object obj, Field fld) {
    int hc = FieldHandle.hash_code(obj, fld);
    int i = hc & size_mask;

    if (handles[i] == null) {
        hlist = handles[i] = new FieldHandle(obj, fld, hlist);
        if ((++load << 1) > handles.length)
            rehash();
        return;
    }
    if (handles[i] instanceof FieldHandle
        && ((FieldHandle)handles[i]).equals(obj,fld)) {
        return;
    }
    int step = (((hc << 16) | (hc >> 16)) ^ hc) & size_mask | 1;
    while(true) {
        i = (i + step) & size_mask;
        if (handles[i] == null) {
            hlist = handles[i] = new FieldHandle(obj, fld, hlist);
            if ((++load << 1) > handles.length)
                rehash();
            return;
        }
        if (handles[i] instanceof FieldHandle
            && ((FieldHandle)handles[i]).equals(obj,fld)) {
            return;
        }
    }
}

private void rehash() {
    int new_size = handles.length << 1;
    handles = new Handle[new_size];
    size_mask = new_size - 1;

    for(Handle h = hlist; h != null; h = h.next) {
        int hc = h.hash_code();
        int i = hc & size_mask;

        if (handles[i] == null) {
            handles[i] = h;
        } else {
            int step = (((hc << 16) | (hc >> 16)) ^ hc) & size_mask | 1;
            do {
                i = (i + step) & size_mask;
            } while (handles[i] != null);
            handles[i] = h;
        }
    }
}
```



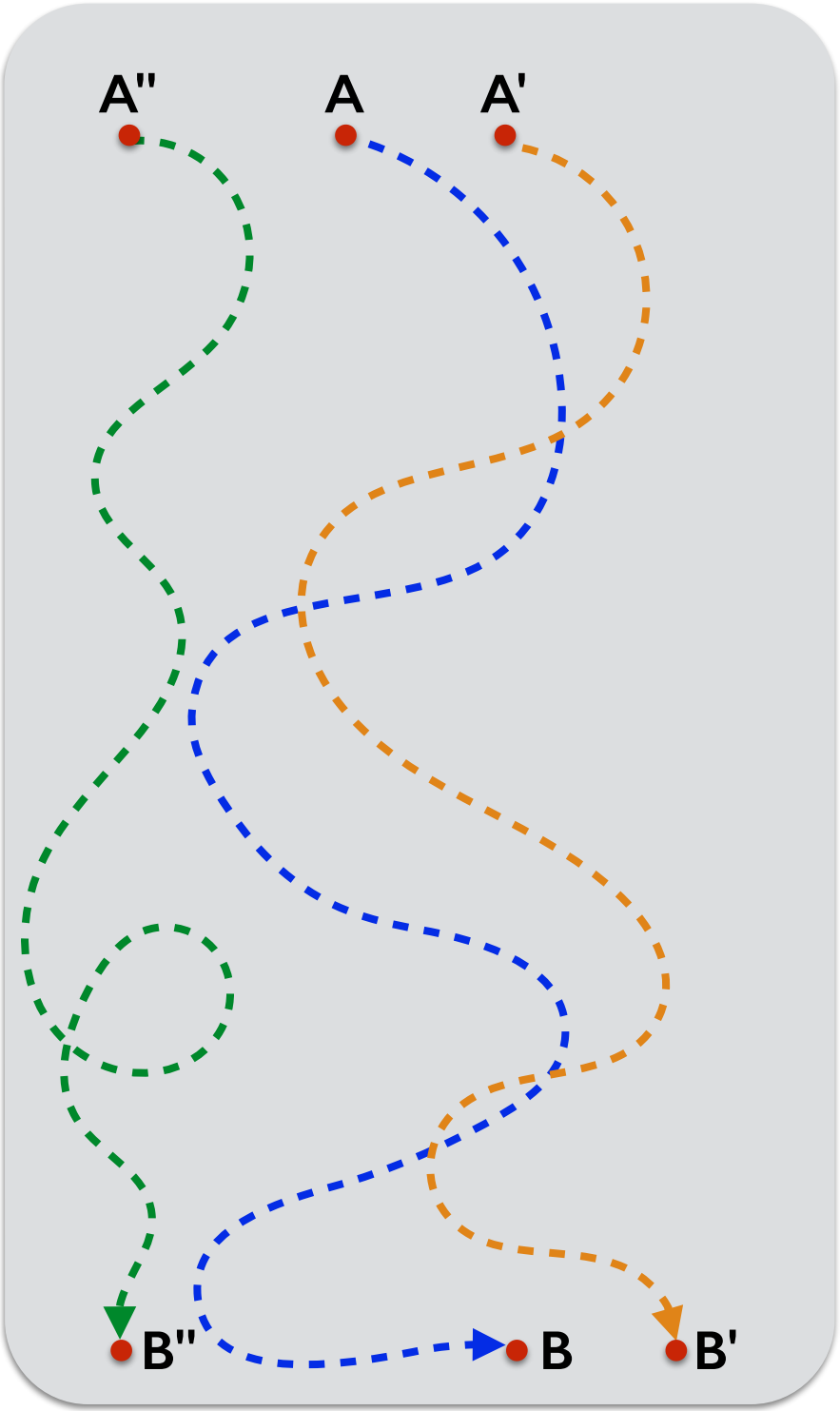

```
private int size_mask;

pub
}
pri
}

private void cp_save_object_field(Object obj, Field fld) {
    int hc = FieldHandle.hash_code(obj, fld);
    int i = hc & size_mask;

    if (handles[i] == null) {
        hlist = handles[i] = new FieldHandle(obj, fld, hlist);
        if ((++load << 1) > handles.length)
            rehash();
        return;
    }
    if (handles[i] instanceof FieldHandle
        && ((FieldHandle)handles[i]).equals(obj,fld)) {
        return;
    }
    int step = (((hc << 16) | (hc >> 16)) ^ hc) & size_mask | 1;
    while(true) {
        i = (i + step) & size_mask;
        if (handles[i] == null) {
            hlist = handles[i] = new FieldHandle(obj, fld, hlist);
            if ((++load << 1) > handles.length)
                rehash();
            return;
        }
        if (handles[i] instanceof FieldHandle
            && ((FieldHandle)handles[i]).equals(obj,fld)) {
            return;
        }
    }
}

private void rehash() {
    int new_size = handles.length << 1;
    handles = new Handle[new_size];
    size_mask = new_size - 1;

    for(Handle h = hlist; h != null; h = h.next) {
        int hc = h.hash_code();
        int i = hc & size_mask;

        if (handles[i] == null) {
            handles[i] = h;
        } else {
            int step = (((hc << 16) | (hc >> 16)) ^ hc) & size_mask | 1;
            do {
                i = (i + step) & size_mask;
            } while (handles[i] != null);
            handles[i] = h;
        }
    }
}
```



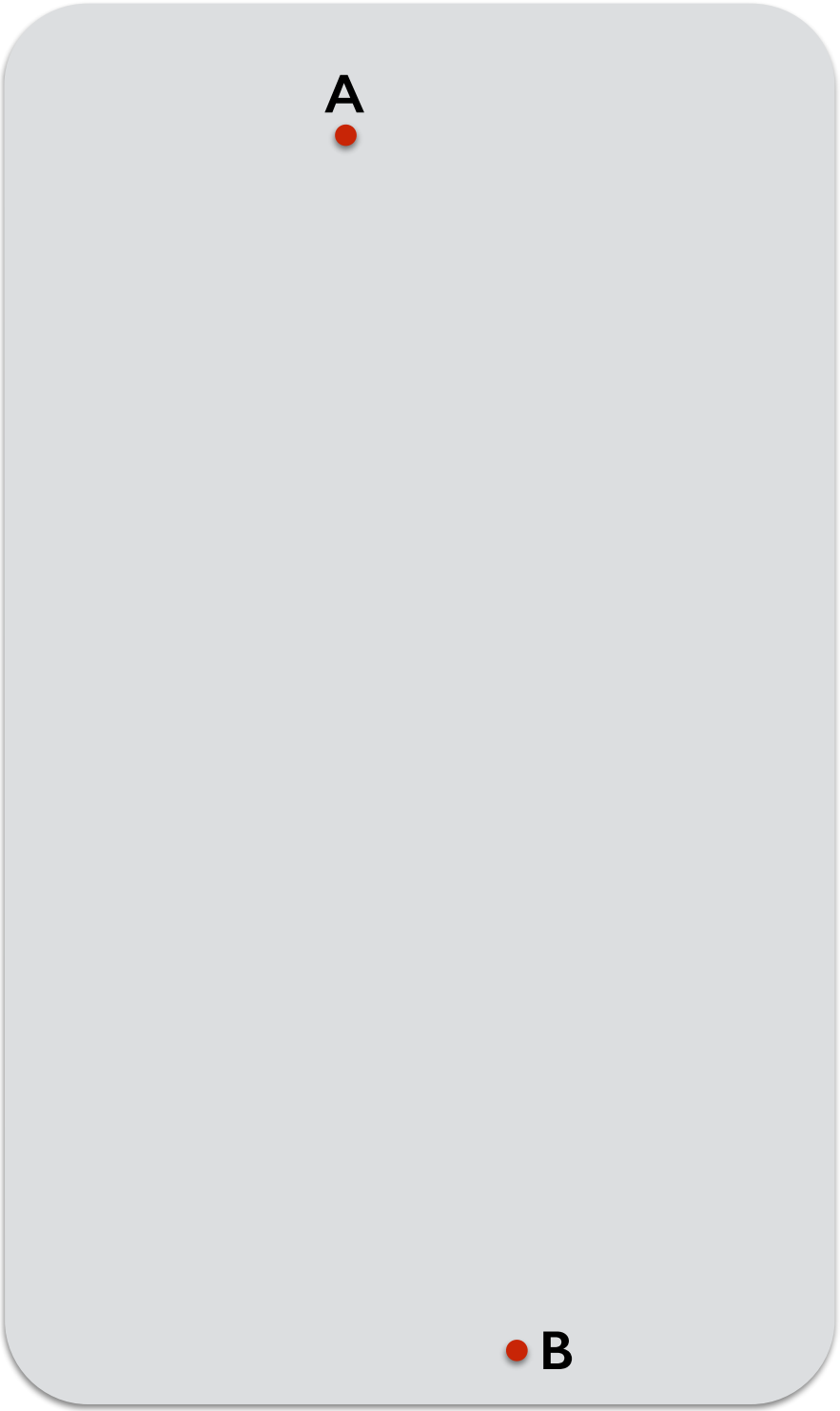

```
private int size_mask;

pub
}

pri
}

pri
}

pri
}
```

```
private void rehash() {
    int new_size = handles.length << 1;
}

priv
}
```

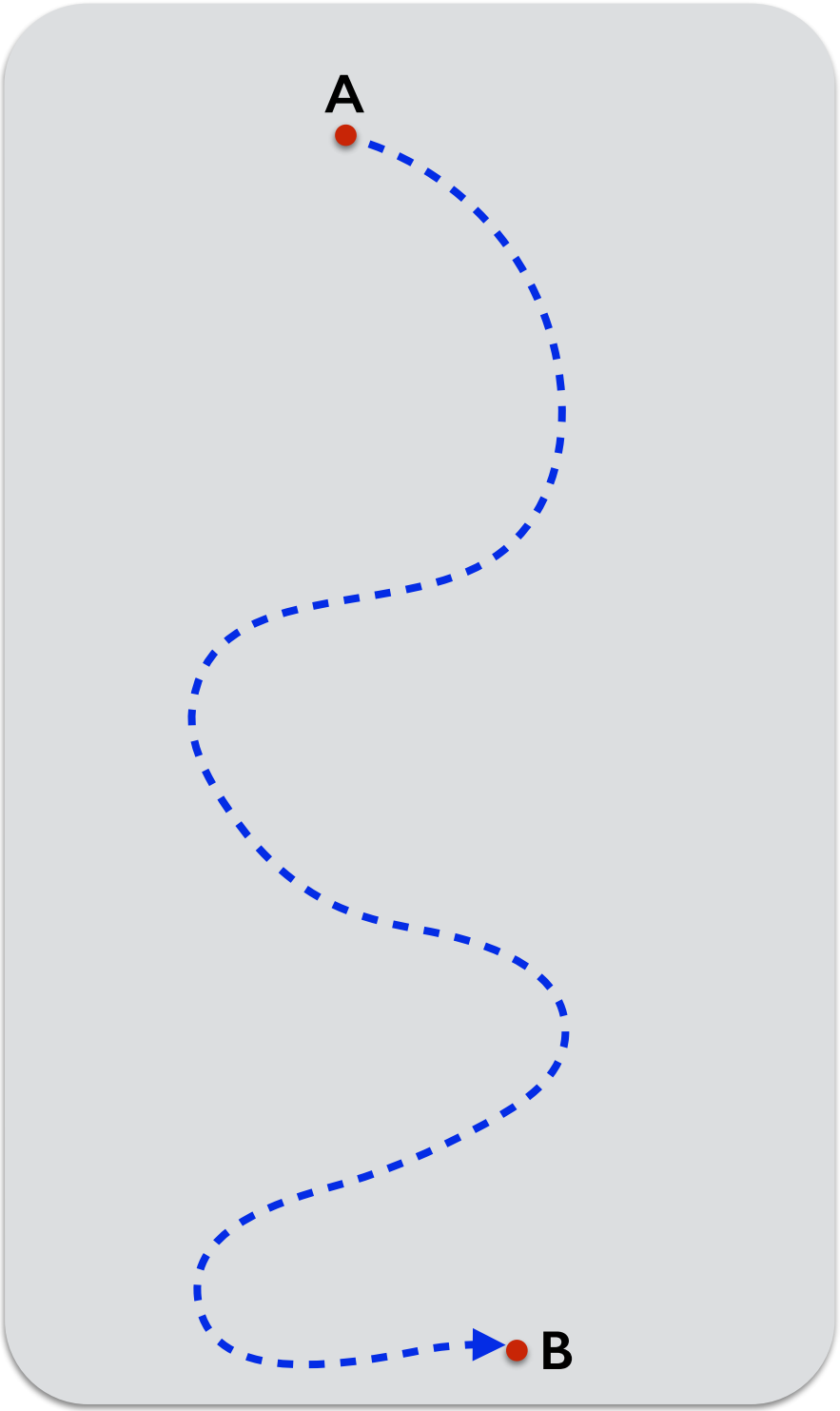
```
private void cp_save_array_element(Object array, int index) {
    if (index < 0 || index >= Array.getLength(array))
        // we simply let this slide, because this should cause the
        // application to throw an ArrayOutOfBoundsException
        return;

    int hc = ElementHandle.hash_code(array, index);
    int i = hc & size_mask;

    if (handles[i] == null) {
        hlist = handles[i] = new ElementHandle(array, index, hlist);
        if ((++load << 1) > handles.length)
            rehash();
        return;
    }
    if (handles[i] instanceof ElementHandle
        && ((ElementHandle)handles[i]).equals(array,index)) {
        return;
    }

    int step = (((hc << 16) | (hc >> 16)) ^ hc) & size_mask | 1;

    while (true) {
        i = (i + step) & size_mask;
        if (handles[i] == null) {
            hlist = handles[i] = new ElementHandle(array, index, hlist);
            if ((++load << 1) > handles.length)
                rehash();
            return;
        }
        if (handles[i] instanceof ElementHandle
            && ((ElementHandle)handles[i]).equals(array,index)) {
            return;
        }
    }
}

public static void save_object_field(Object obj, String fld) {
    ICheckPoint cp = current_cp.get();
    if (cp == null || cp.suspended)
        return;
    try {
        cp.cp_save_object_field(obj, field_finder.get().find(obj, fld));
    } catch (NoSuchFieldException ignored) {
        ignored.printStackTrace();
        // this should never happen, because this method is called
        // with the exact method name produced during the
        // instrumentation.
    }
}
```



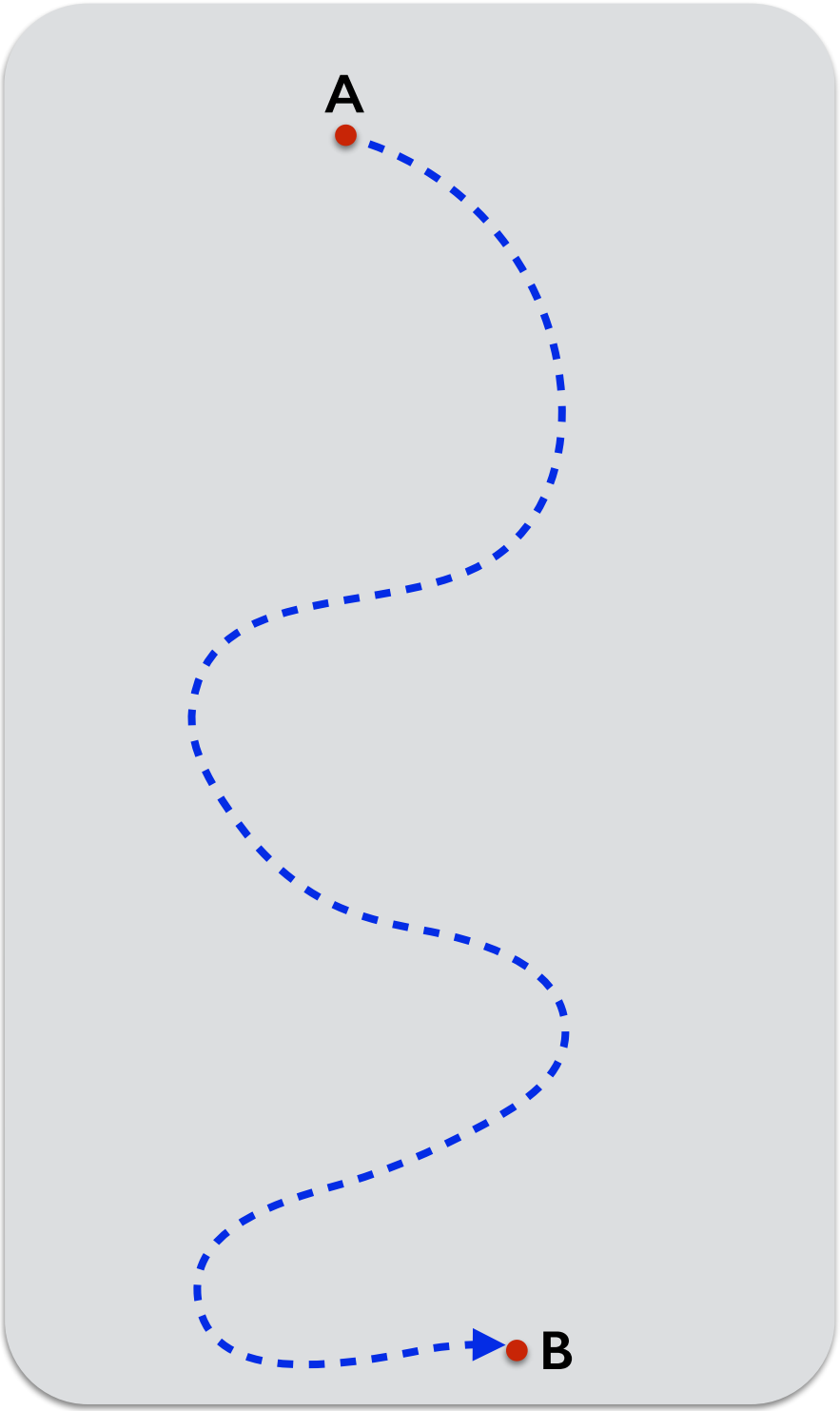

```
private int size_mask;

pub
}

pri
}

pri
}

private void rehash() {
    int new_size = handles.length << 1;
priv

}

private void cp_save_array_element(Object array, int index) {
    if (index < 0 || index >= Array.getLength(array))
        // we simply let this slide, because this should cause the
        // application to throw an ArrayOutOfBoundsException
        return;

    int hc = ElementHandle.hash_code(array, index);
    int i = hc & size_mask;

    if (handles[i] == null) {
        hlist = handles[i] = new ElementHandle(array, index, hlist);
        if ((++load << 1) > handles.length)
            rehash();
        return;
    }
    if (handles[i] instanceof ElementHandle
        && ((ElementHandle)handles[i]).equals(array,index)) {
        return;
    }

    int step = (((hc << 16) | (hc >> 16)) ^ hc) & size_mask | 1;

    while (true) {
        i = (i + step) & size_mask;
        if (handles[i] == null) {
            hlist = handles[i] = new ElementHandle(array, index, hlist);
            if ((++load << 1) > handles.length)
                rehash();
            return;
        }
        if (handles[i] instanceof ElementHandle
            && ((ElementHandle)handles[i]).equals(array,index)) {
            return;
        }
    }
}

public static void save_object_field(Object obj, String fld) {
    ICheckPoint cp = current_cp.get();
    if (cp == null || cp.suspended)
        return;
    try {
        cp.cp_save_object_field(obj, field_finder.get().find(obj, fld));
    } catch (NoSuchFieldException ignored) {
        ignored.printStackTrace();
        // this should never happen, because this method is called
        // with the exact method name produced during the
        // instrumentation.
    }
}
```



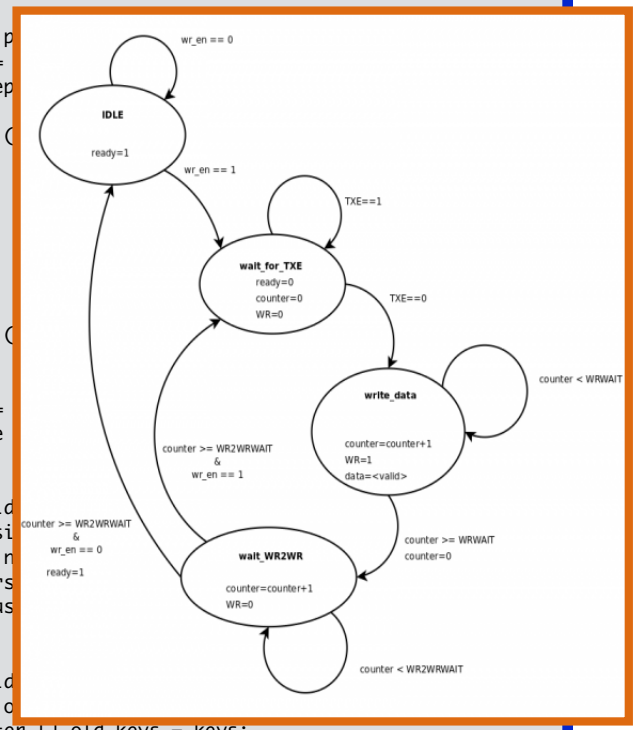

```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            return 1;
        }
        if (keys[h] == k)
            return ++counters[h];

        h = (h + step) & size_mask;
    } while (true);
}

private void create_table(int size) {
    table_size = size;
    keys = new SFFFilter[size];
    counters = new int[size];
    size_mask = size - 1;
}

private void rehash() {
    int [] old_counters = counters;
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```

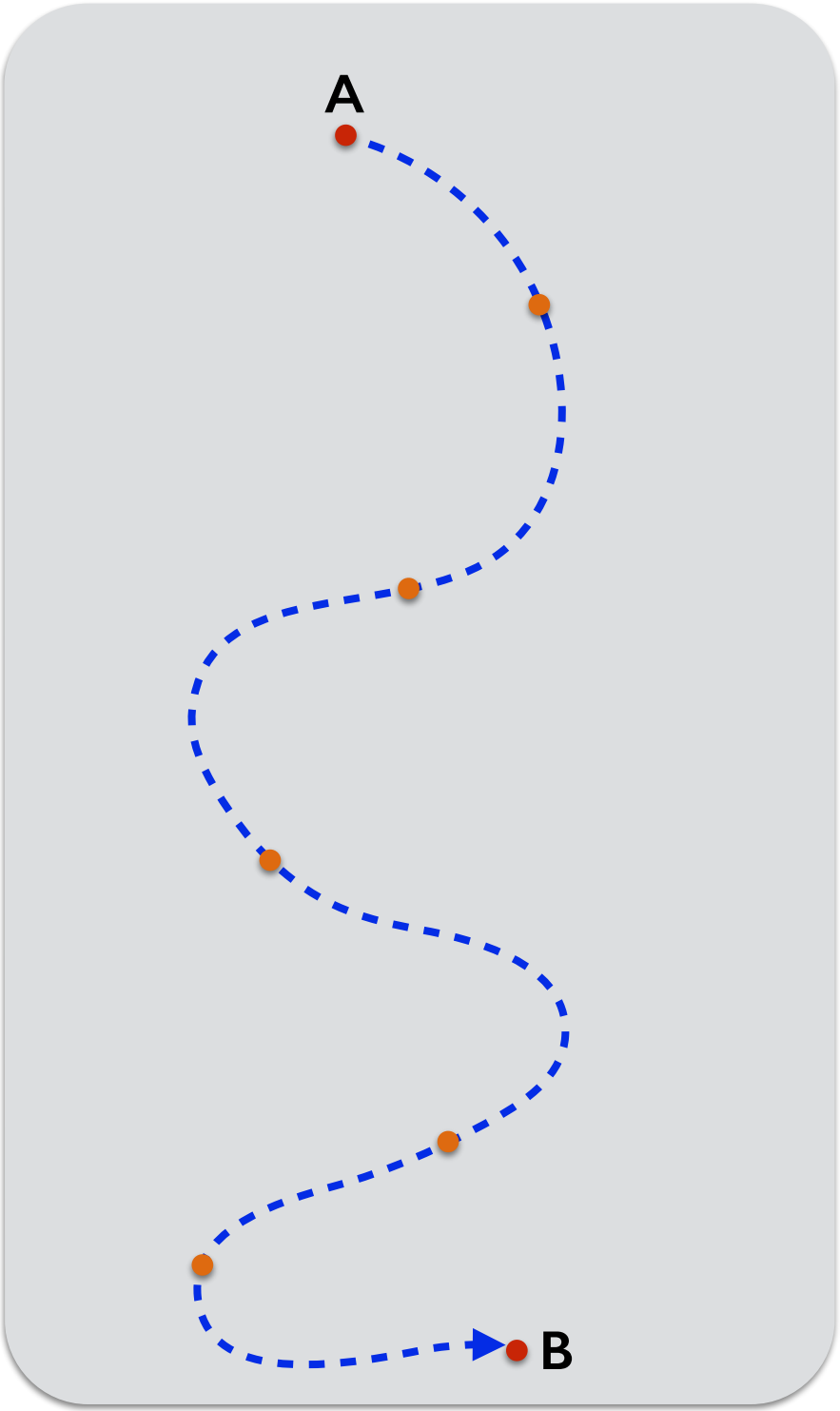



```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            return 1;
        }
        if (keys[h] == k)
            return ++counters[h];

        h = (h + step) & size_mask;
    } while (true);
}

private void create_table(int size) {
    table_size = size;
    keys = new SFFFilter[size];
    counters = new int[size];
    size_mask = size - 1;
}

private void rehash() {
    int [] old_counters = counters;
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```

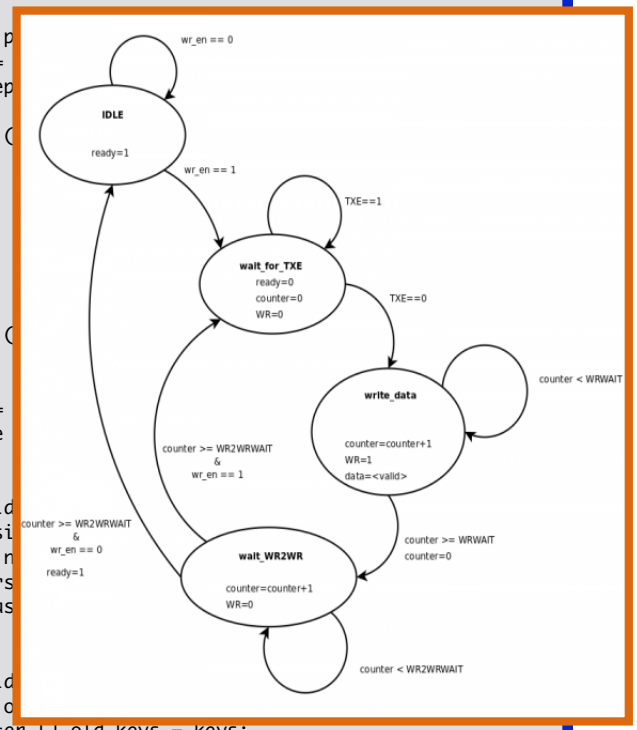


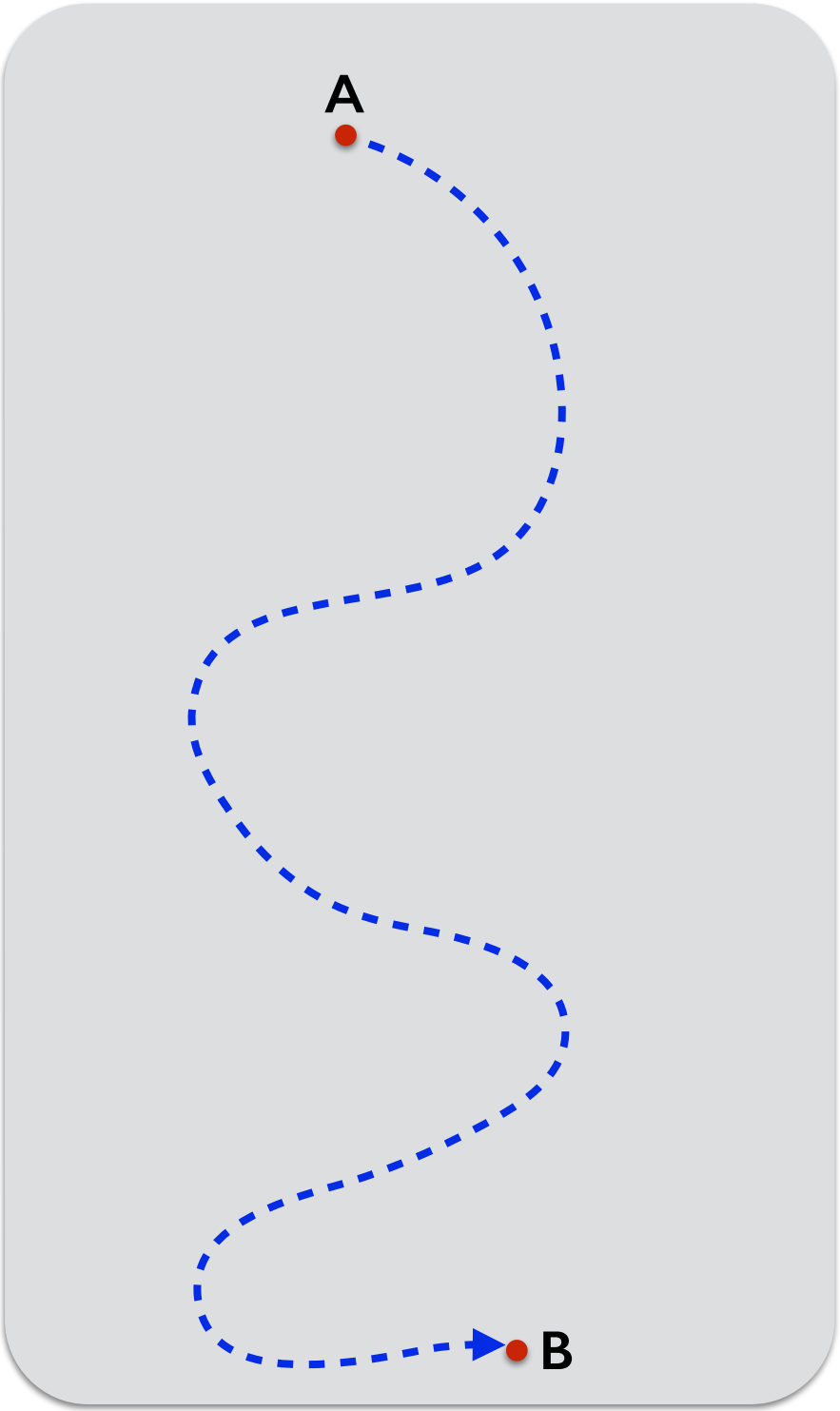

```
private int size_mask;

public int p
    int h =
    int step
    do {
        if (

        }
        if (

        h =
    } while
}

private void
    table_si
    keys = n
    counters
    size_mas
}

private void
    int [] o
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```





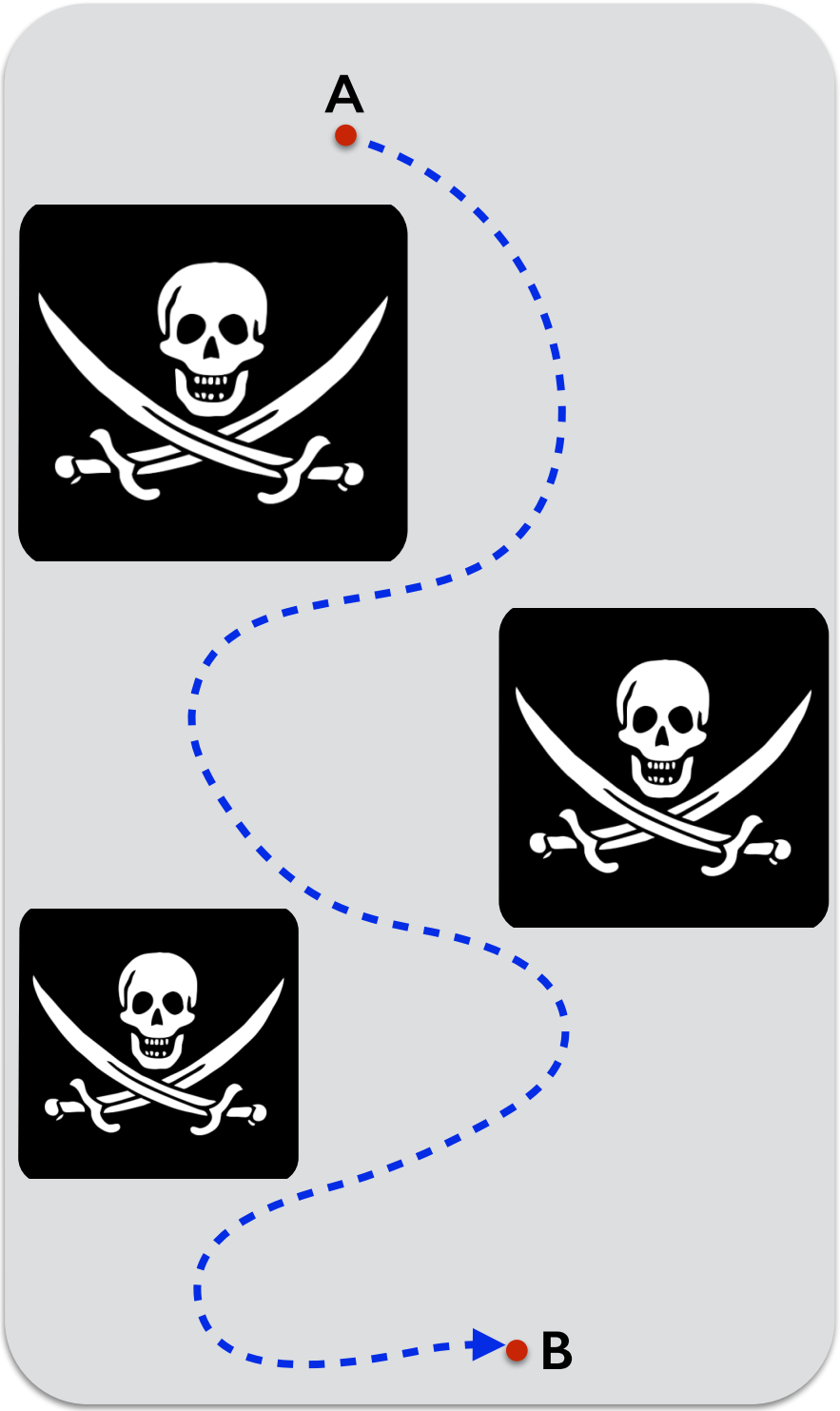

```
private int size_mask;

public int p
    int h =
    int step
    do {
        if (

        }
        if (

        h =
    } while
}

private void
    table_si
    keys = n
    counters
    size_mas
}

private void
    int [] o
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```





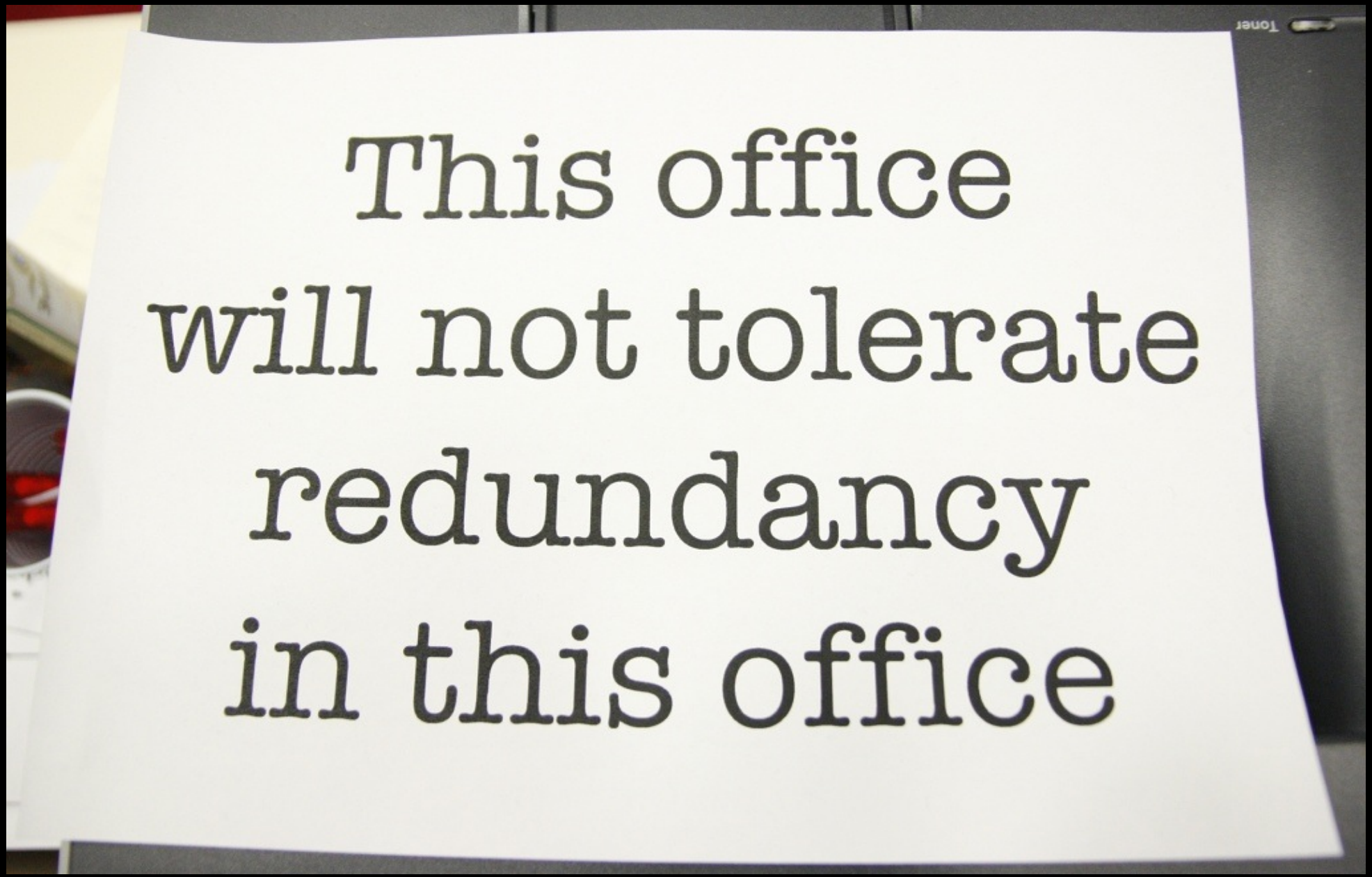

```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            return 1;
        }
        if (keys[h] == k)
            return ++counters[h];

        h = (h + ste
    } while (true);
}

private void create
    table_size = siz
    keys = new SFFFi
    counters = new i
    size_mask = size
}

private void rehash(
    int [] old_counters = counters;
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```

```
assert x > 0;

assert mutex == null;

assert parameter != null;

…
```




```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            return 1;
        }
        if (keys[h] == k)
            return ++counters[h];

        h = (h + ste
    } while (true);
}

private void create
    table_size = siz
    keys = new SFFFi
    counters = new i
    size_mask = size
}

private void rehash(
    int [] old_counters = counters;
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```

```
assert x > 0;

assert mutex == null;

assert parameter != null;

…
```


This office
will not tolerate
redundancy
in this office

Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

...and this intrinsic redundancy can be exploited for good purposes with no design cost."

Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

Does it exist?

...and this intrinsic redundancy can be exploited for good purposes with no design cost."

Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

Does it exist?

...and this intrinsic redundancy can be exploited for good purposes with no design cost."

How?

Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

Does it exist?

...and this intrinsic redundancy can be exploited for good purposes with no design cost."

How?

For what?

Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

...and this intrinsic redundancy can be exploited for good purposes with no design cost."

Does it exist?

How?

For what?

Is it effective?

Intrinsic Redundancy: Examples

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
```

Intrinsic Redundancy: Examples

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
 DateTime beginDay = t.toDateMidnight().toDateTime();
```

Intrinsic Redundancy: Examples

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
 DateTime beginDay = t.toDateMidnight().toDateTime();
 DateTime beginDay = t.withTimeAtStartOfDay();
```

Intrinsic Redundancy: Examples

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
 DateTime beginDay = t.toDateMidnight().toDateTime();
 DateTime beginDay = t.withTimeAtStartOfDay();
```

Google Guava

```
MultiMap m = new MultiMap();
//…
//check if element is already in map
if (m.contains(x))
```

Intrinsic Redundancy: Examples

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
 DateTime beginDay = t.toDateMidnight().toDateTime();
 DateTime beginDay = t.withTimeAtStartOfDay();
```

Google Guava

```
MultiMap m = new MultiMap();
//…
//check if element is already in map
if (m.contains(x))
 if (m.elementSet().contains(x))
```

Intrinsic Redundancy: Examples

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
 DateTime beginDay = t.toDateMidnight().toDateTime();
 DateTime beginDay = t.withTimeAtStartOfDay();
```

Google Guava

```
MultiMap m = new MultiMap();
//…
//check if element is already in map
if (m.contains(x))
 if (m.elementSet().contains(x))
 if (m.count(x) > 0)
```

Intrinsic Redundancy: Examples

Joda-Time

```
DateTime t = new DateTime();
//...
//get the beginning of the day for time t
DateTime beginDay = t.millisOfDay().withMinimumValue();
 DateTime beginDay = t.toDateMidnight().toDateTime();
 DateTime beginDay = t.withTimeAtStartOfDay();
```

Google Guava

```
MultiMap m = new MultiMap();
//…
//check if element is already in map
if (m.contains(x))
 if (m.elementSet().contains(x))
 if (m.count(x) > 0)
```

Multimap.contains(key) ≡
{ map.elementSet().contains(key);
}

Intrinsic Redundancy: Examples

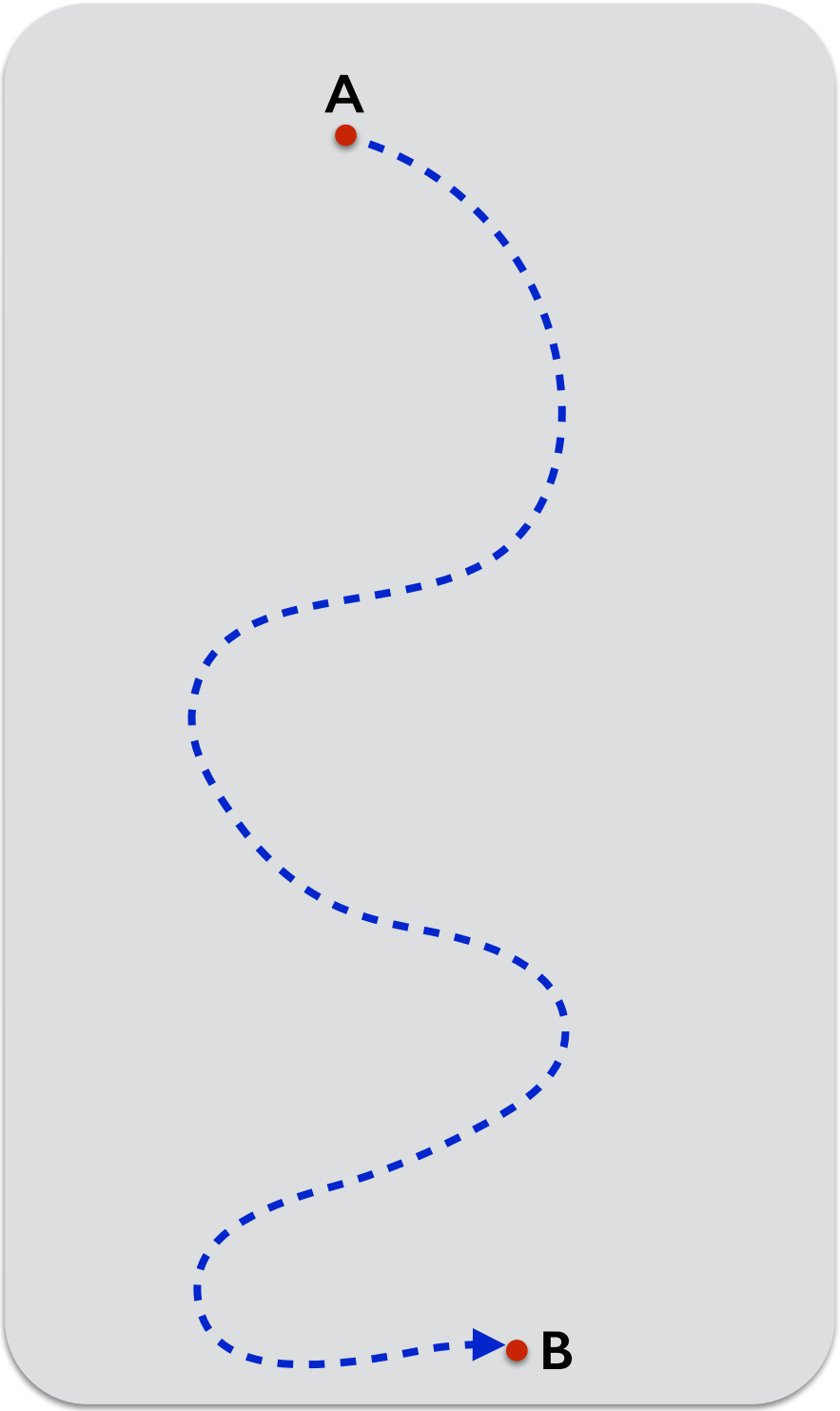
Google Guava

Stores a key-value pair in the multimap.

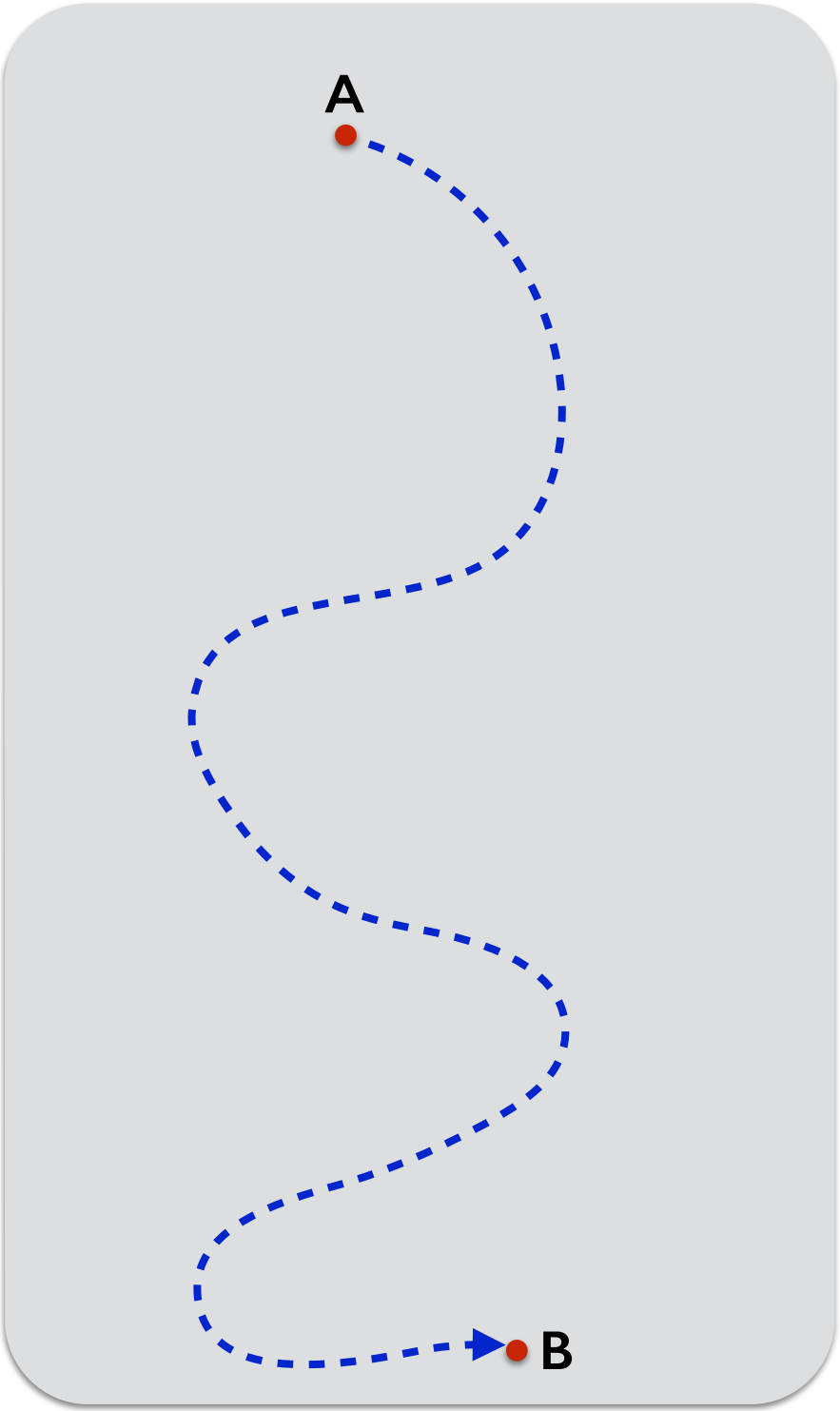
```
public boolean put(K key, V value) {
    Collection<V> collection = map.get(key);
    if (collection == null) {
      collection = createCollection(key);
      if (collection.add(value)) {
        totalSize++;
        map.put(key, collection);
        return true;
      } else {
        throw new AssertionError("...");
      }
    } else if (collection.add(value)) {
      totalSize++;
      return true;
    } else {
      return false;
    }
}
```

Stores a key-value pairs in the multimap with one key and multiple values.

```
public boolean putAll(K key, Iterable<? extends V>
                values) {
  if (!values.iterator().hasNext()) {
    return false;
  }
  Collection<V> collection =
          getOrCreateCollection(key);
  int oldSize = collection.size();

  boolean changed = false;
  if (values instanceof Collection) {
    Collection<? extends V> c =
          Collections2.cast(values);
    changed = collection.addAll(c);
  } else {
    for (V value : values) {
      changed |= collection.add(value);
    }
  }

  totalSize += (collection.size() - oldSize);
  return changed;
}
```

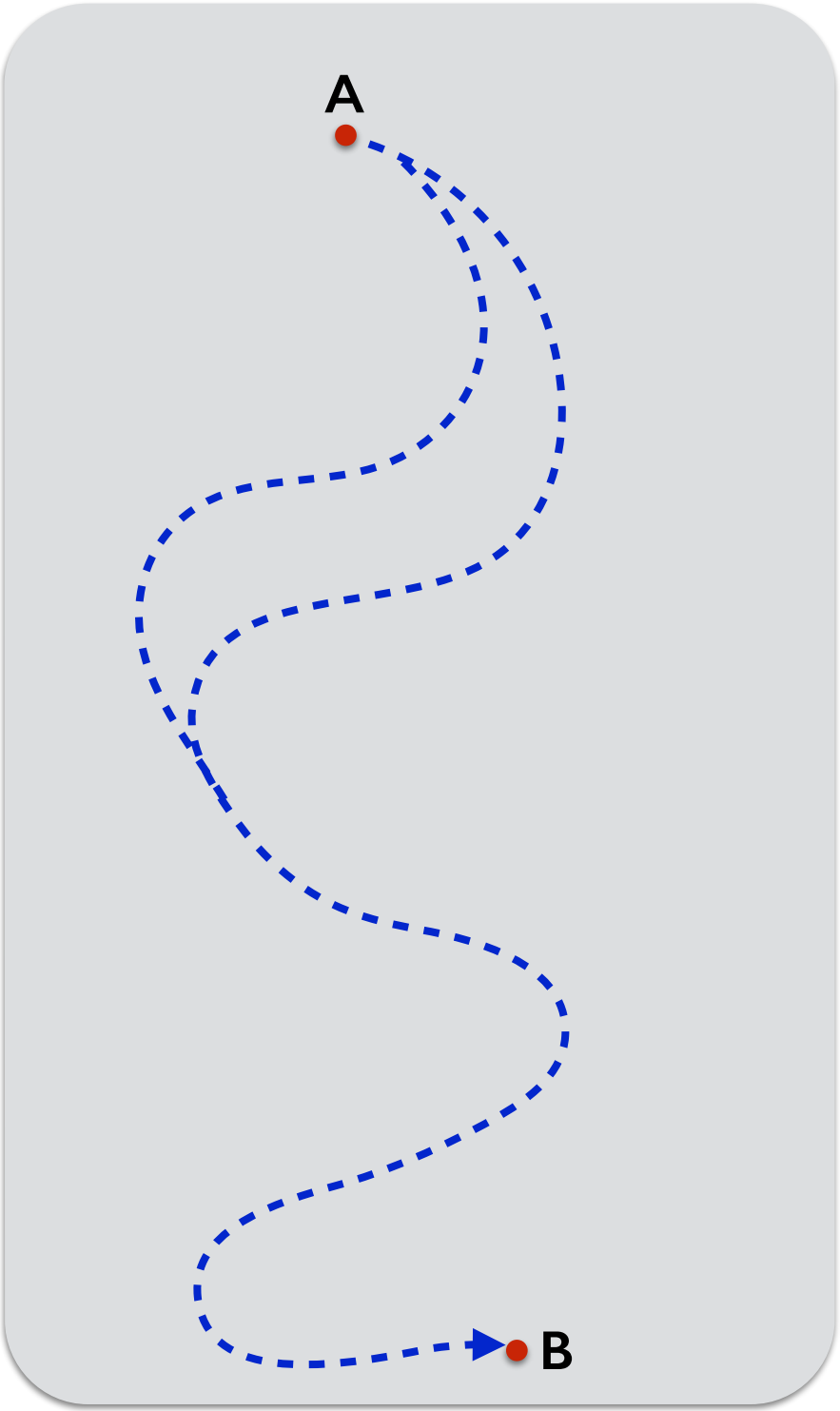



```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            return 1;
        }
        if (keys[h] == k)
            return ++counters[h];

        h = (h + step) & size_mask;
    } while (true);
}

private void create_table(int size) {
    table_size = size;
    keys = new SFFFilter[size];
    counters = new int[size];
    size_mask = size - 1;
}

private void rehash() {
    int [] old_counters = counters;
    SFFFilter [] old_keys = keys;
    int old_table_size = table_size;

    create_table(table_size << 1);

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```



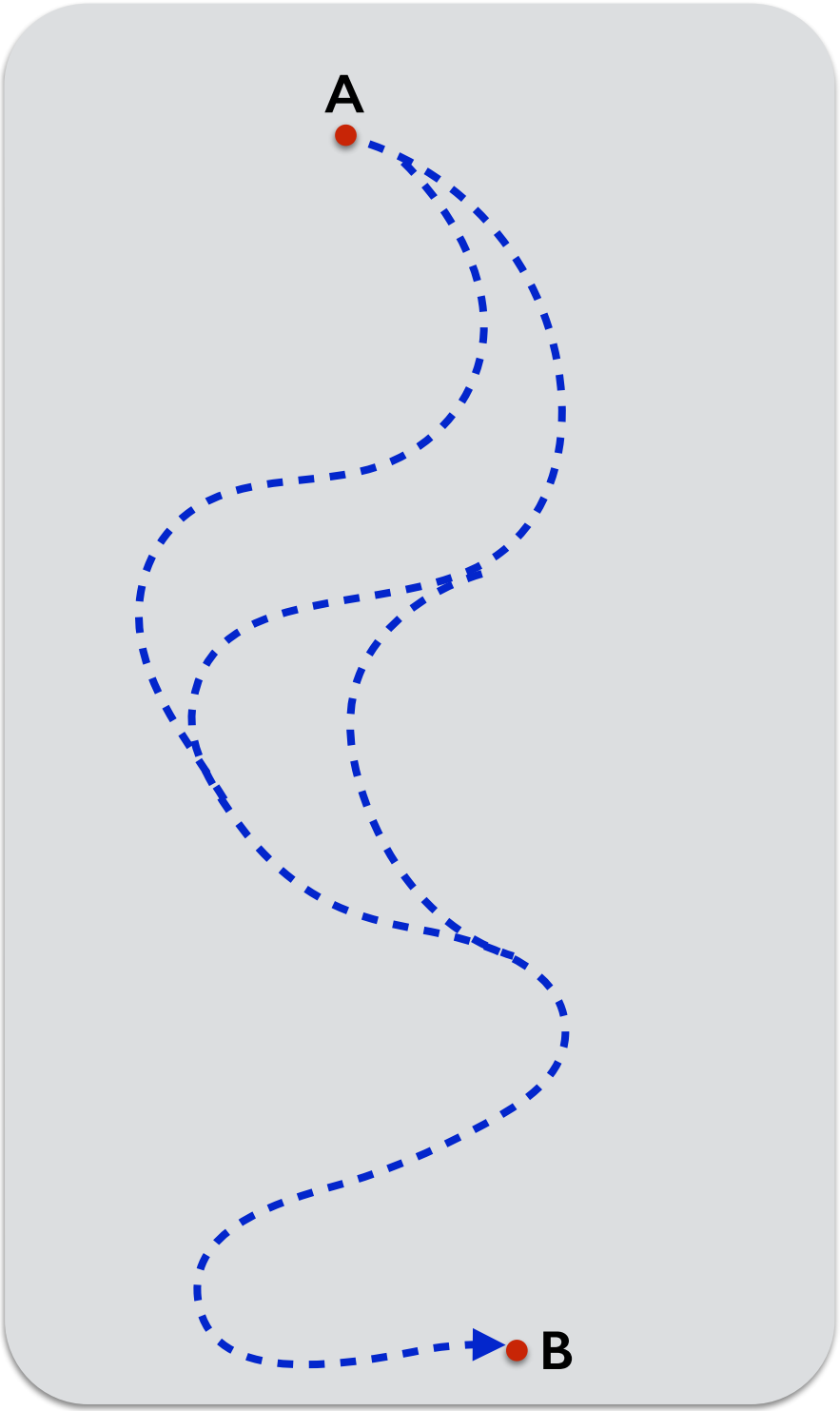

```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            retur
        }
        if (keys[
            retur

        h = (h +
    } while (true
}

private void crea
    table_size =
    keys = new SF
    counters = ne
    size_mask = s
}

private void reha
    int [] old_co
    SFFFilter [] 
    int old_table

    create_table(

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```

```
AbstractMultimap.put(key, value)
≡ {   List list = new ArrayList();
      list.add(value);
      target.putAll(key,list);
  }

AbstractMultimap.clear()
≡ {   for (String k : target.keySet())
        target.removeAll(k);
  }
```



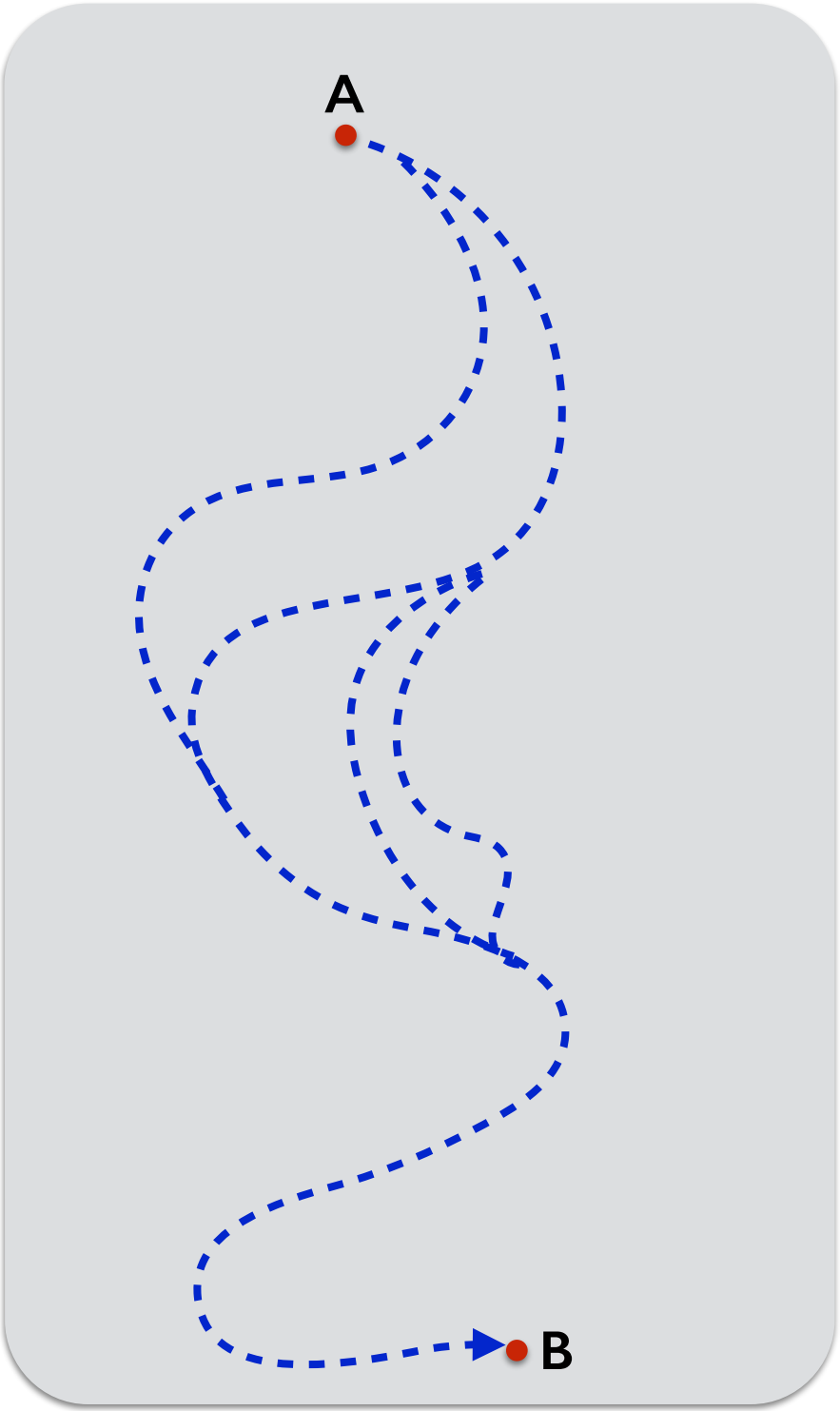

```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            retur
        }
        if (keys[
            retur

        h = (h +
    } while (true
}

private void crea
    table_size =
    keys = new SF
    counters = ne
    size_mask = s
}

private void reha
    int [] old_co
    SFFFilter [] 
    int old_table

    create_table(

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```

```
AbstractMultimap.put(key, value)
≡ {   List list = new ArrayList();
      list.add(value);
      target.putAll(key,list);
   }

AbstractMultimap.clear()
≡ {   for (String k : target.keySet())
        target.removeAll(k);
   }
```




```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            retur
        }
        if (keys[
            retur

        h = (h +
    } while (true
}

private void crea
    table_size =
    keys = new SF
    counters = ne
    size_mask = s
}

private void reha
    int [] old_co
    SFFFilter [] 
    int old_table

    create_table(

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```

```
AbstractMultimap.put(key, value)
≡ {   List list = new ArrayList();
      list.add(value);
      target.putAll(key,list);
  }

AbstractMultimap.clear()
≡ {   for (String k : target.keySet())
        target.removeAll(k);
  }
```




```
private int size_mask;

public int plus_one(SFFFilter k) {
    int h = System.identityHashCode(k) & size_mask;
    int step = (~h | 1) & size_mask;
    do {
        if (keys[h] == null) {
            keys[h] = k;
            counters[h] = 1;
            ++load;
            if (load * 2 > table_size)
                rehash();
            retur
        }
        if (keys[
            retur

        h = (h +
    } while (true
}

private void crea
    table_size =
    keys = new SF
    counters = ne
    size_mask = s
}

private void reha
    int [] old_co
    SFFFilter [] 
    int old_table

    create_table(

    for(int i = 0; i < old_table_size; ++i) {
        SFFFilter k = old_keys[i];
        if (k != null) {
            // See the documentation of plus_one().
            int h = System.identityHashCode(k) & size_mask;
            int step = (~h | 1) & size_mask;

            while (keys[h] != null && keys[h] != k) {
                h = (h + step) & size_mask;
            }
            keys[h] = k;
            counters[h] = old_counters[i];
        }
    }
}
```

```
AbstractMultimap.put(key, value)
≡ {   List list = new ArrayList();
      list.add(value);
      target.putAll(key,list);
   }

AbstractMultimap.clear()
≡ {   for (String k : target.keySet())
        target.removeAll(k);
   }
```

Plausibility of Intrinsic Redundancy

Design for reusability

ImmutableList<E> list = ImmutableList.**create()**;

ImmutableList<E> list = ImmutableList.**of()**;

ImmutableList<E> list = ImmutableList.<E>**builder().build()**;

Plausibility of Intrinsic Redundancy

Design for reusability

Non-functional requirements

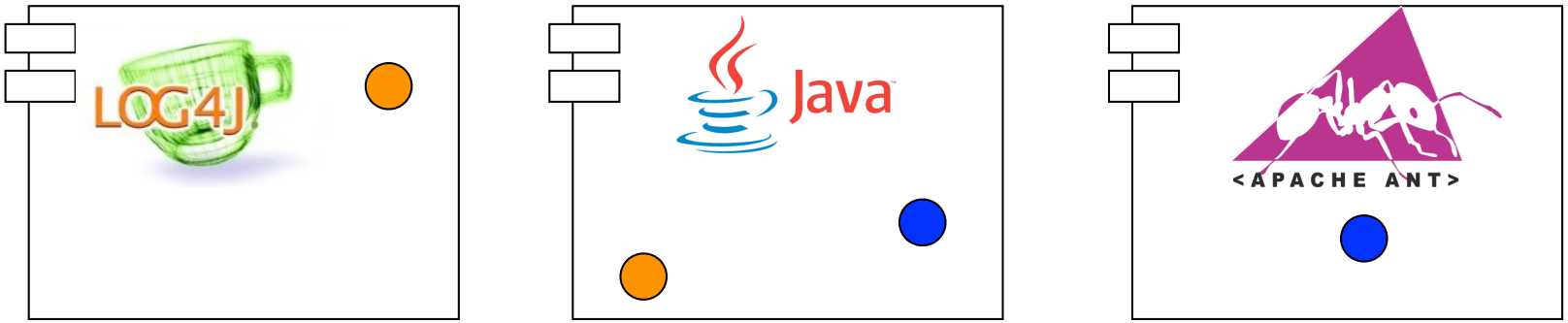
```
/**
    * Breaks a path up into a Vector of path elements, tokenizing on
    *
    * @param path Path to tokenize. Must not be <code>null</code>.
    * @param separator the separator against which to tokenize.
    *
    * @return a Vector of path elements from the tokenized path
    */
   public static Vector tokenizePath (String path) {...}

/**
    * Same as {@link #tokenizePath tokenizePath} but hopefully faster.
    */
   public static String[] tokenizePathAsArray(String path) {...}
```

Plausibility of Intrinsic Redundancy

Design for reusability

Non-functional requirements

Replicated Functionalities



Plausibility of Intrinsic Redundancy

Design for reusability

Non-functional requirements

Replicated Functionalities

Backward compatibility

```
@deprecated
public static String getText() {...}
```

Does Intrinsic Redundancy Exist?

Joda-Time

SWT

4000+

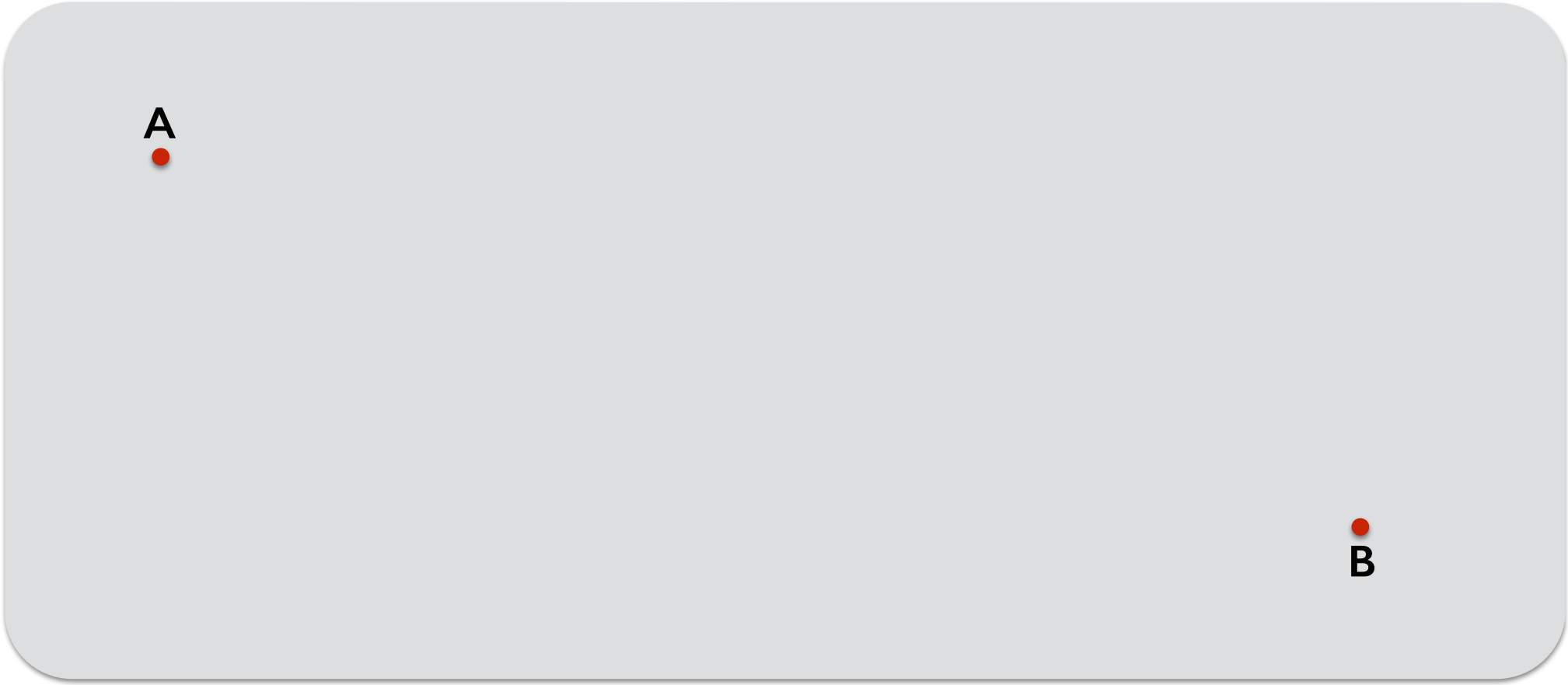
redundancy specifications

GraphStream

Using the Intrinsic Redundancy of Software

Redundancy for Reliability

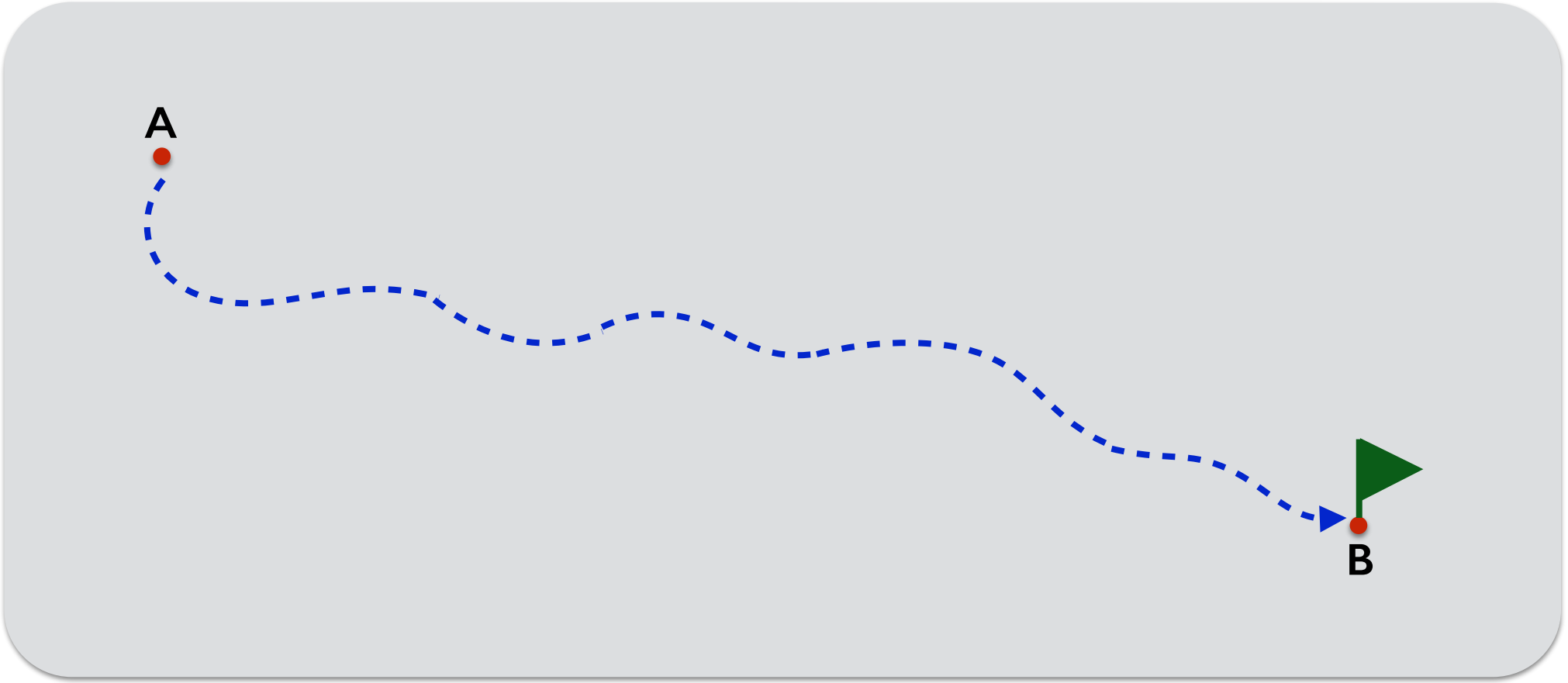
[ICSE'13]



Application state space

Redundancy for Reliability

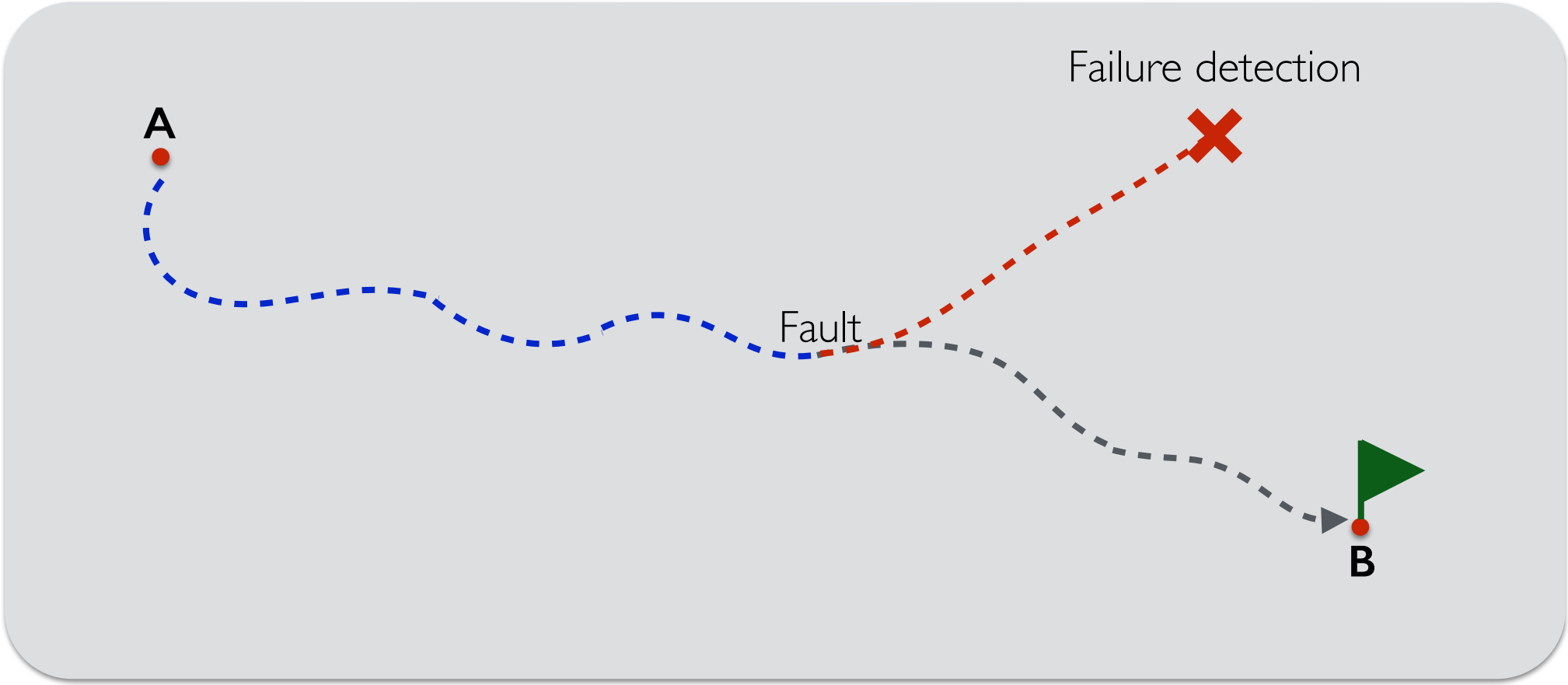
[ICSE'13]



Application state space

Redundancy for Reliability

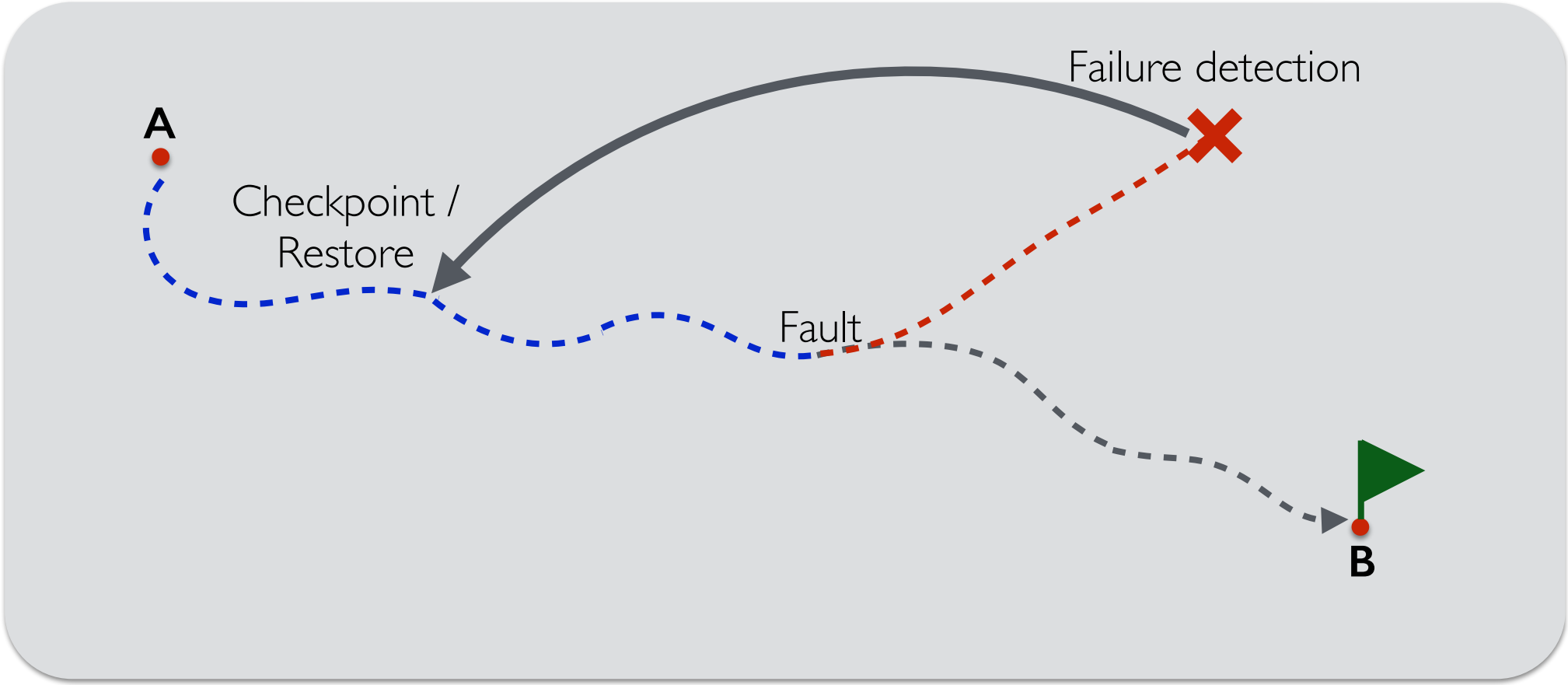
[ICSE'13]



Application state space

Redundancy for Reliability

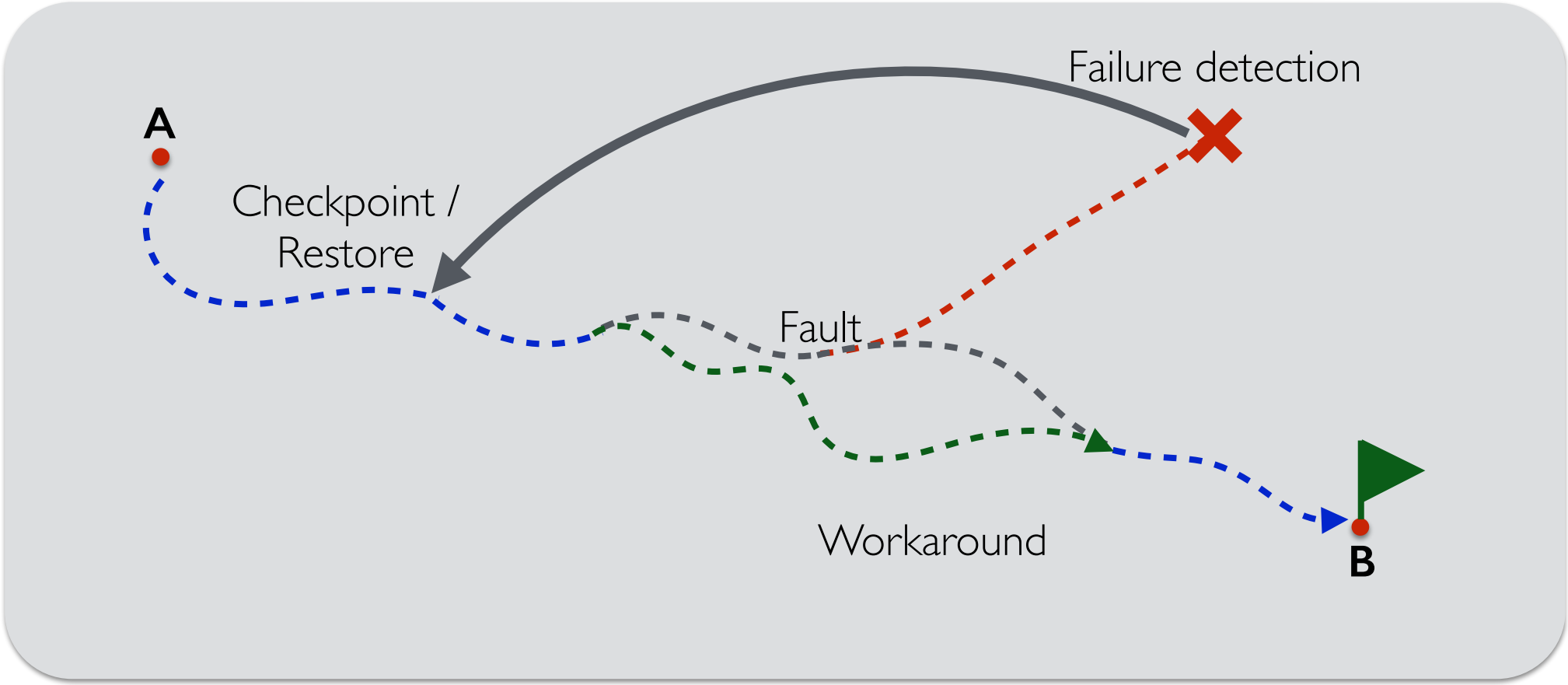
[ICSE'13]



Application state space

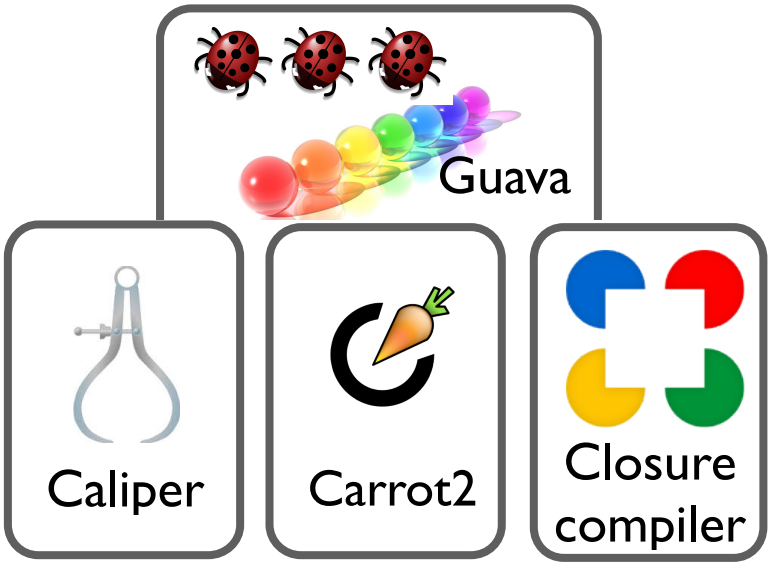
Redundancy for Reliability

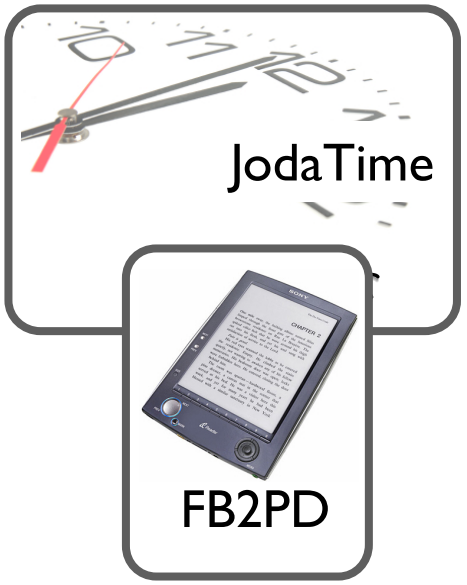
[ICSE'13]



Application state space

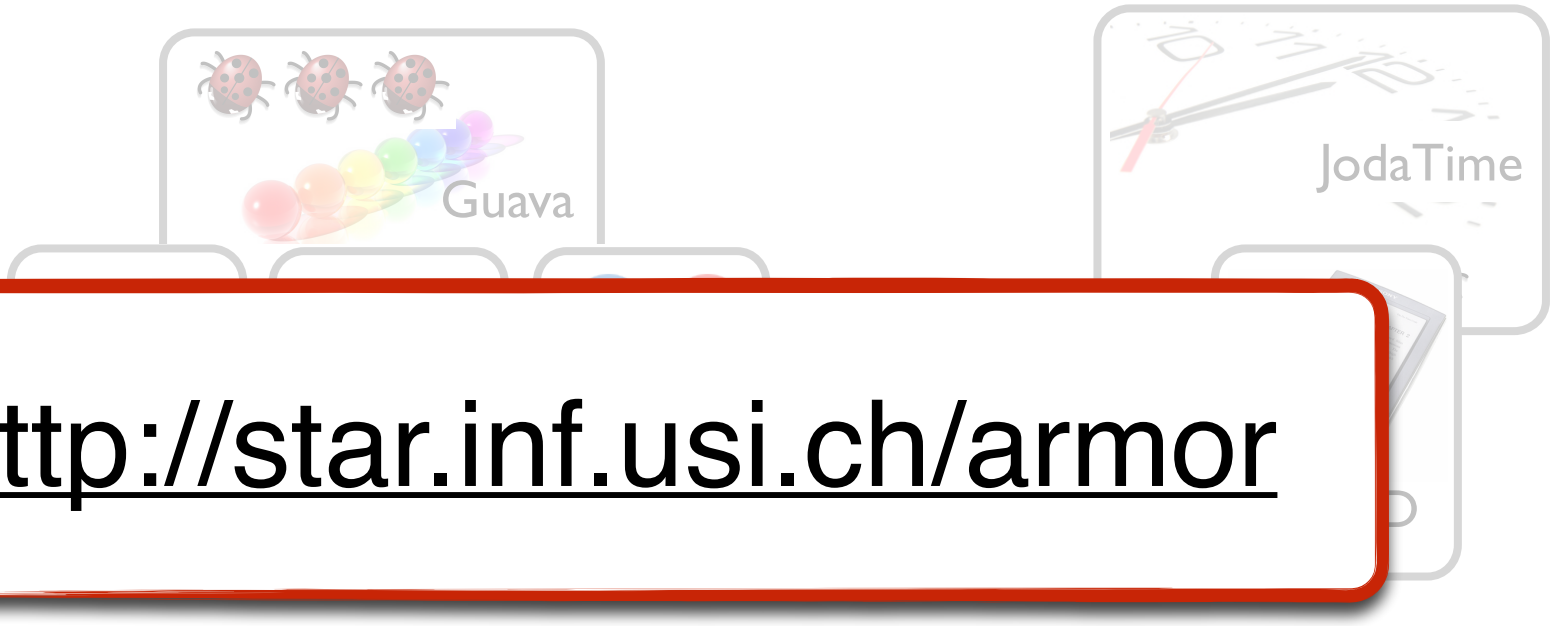
Redundancy for Reliability





Mutants	309	187	344	2200
Considered	87	50	148	347
Successfully handled	24	24	64	67
	27.6%	48%	43.24%	19.3%

Redundancy for Reliability



http://star.inf.usi.ch/armor

Mutants	309	187	344	2200
Considered	87	50	148	347
Successfully handled	24	24	64	67
	27.6%	48%	43.24%	19.3%

Redundancy as Test Oracles

Redundancy as Test Oracles

"Software that applies a pass/fail criterion to a program execution" [1]

[1] M. Pezzè and M. Young. Software Testing and Analysis: Process, Principles and Techniques. 2008.

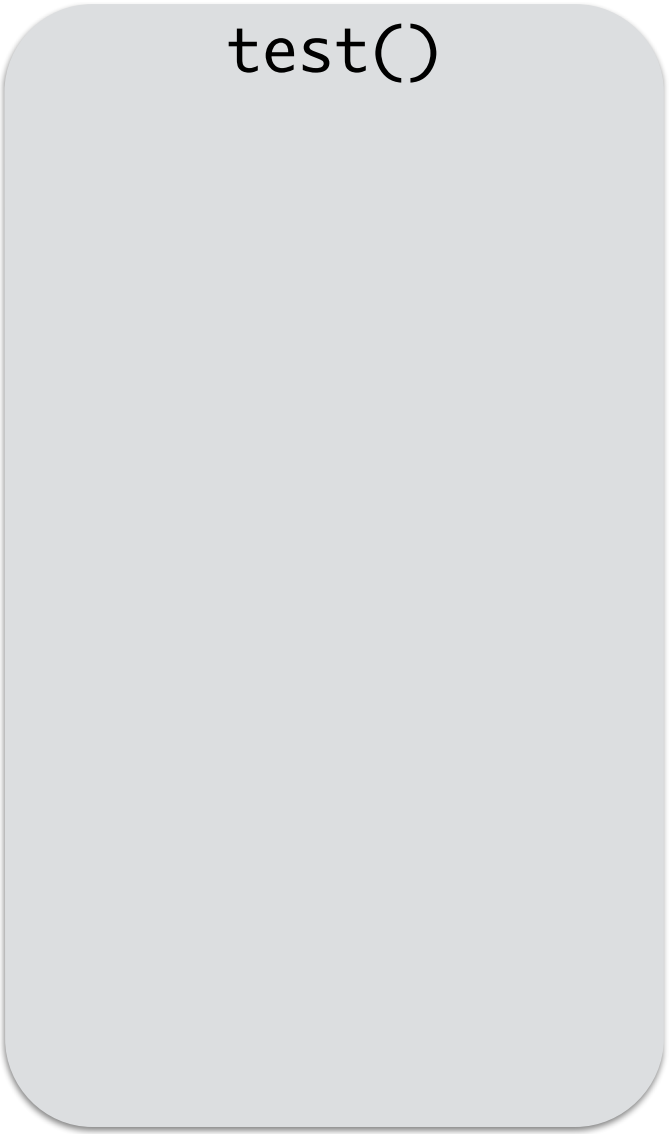
Redundancy as Test Oracles

General: applicable to wide range of inputs

Simple: easy to write, easy to determine correctness

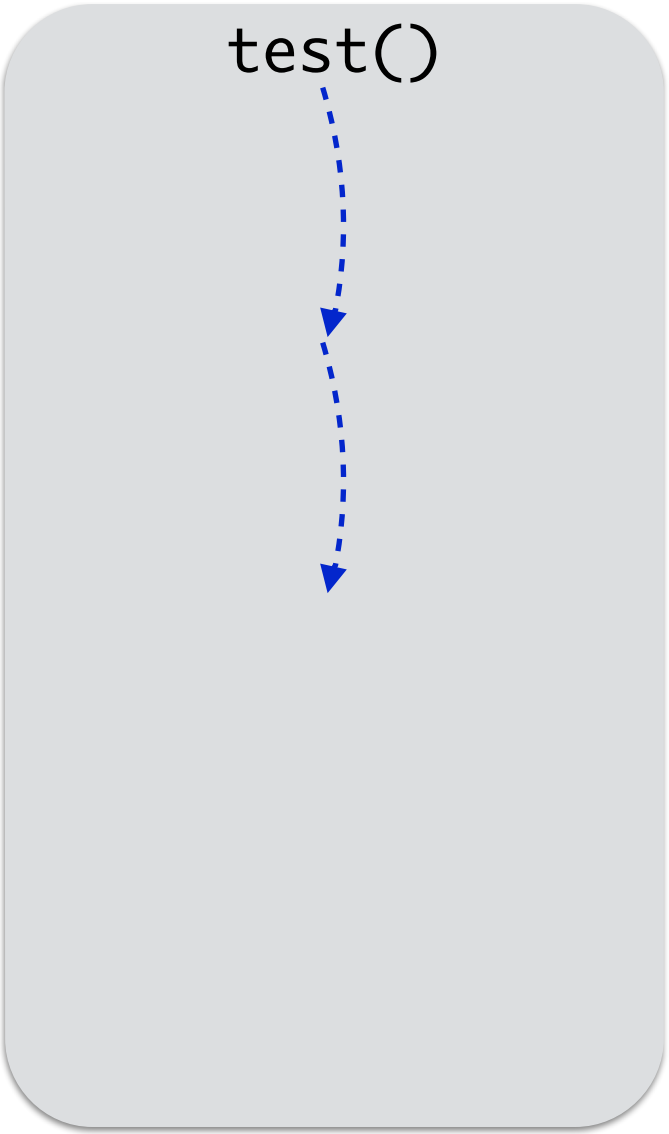
Accurate: completeness and soundness

Redundancy as Test Oracles



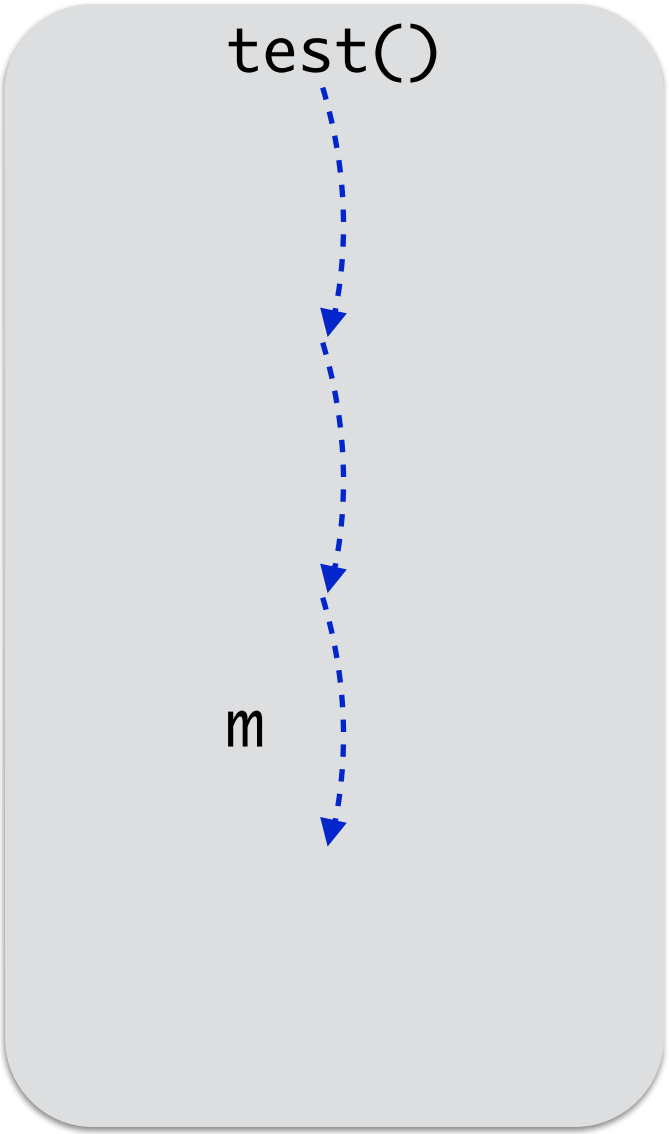

```
test() {
  …
  m();
  …
}
```

Redundancy as Test Oracles



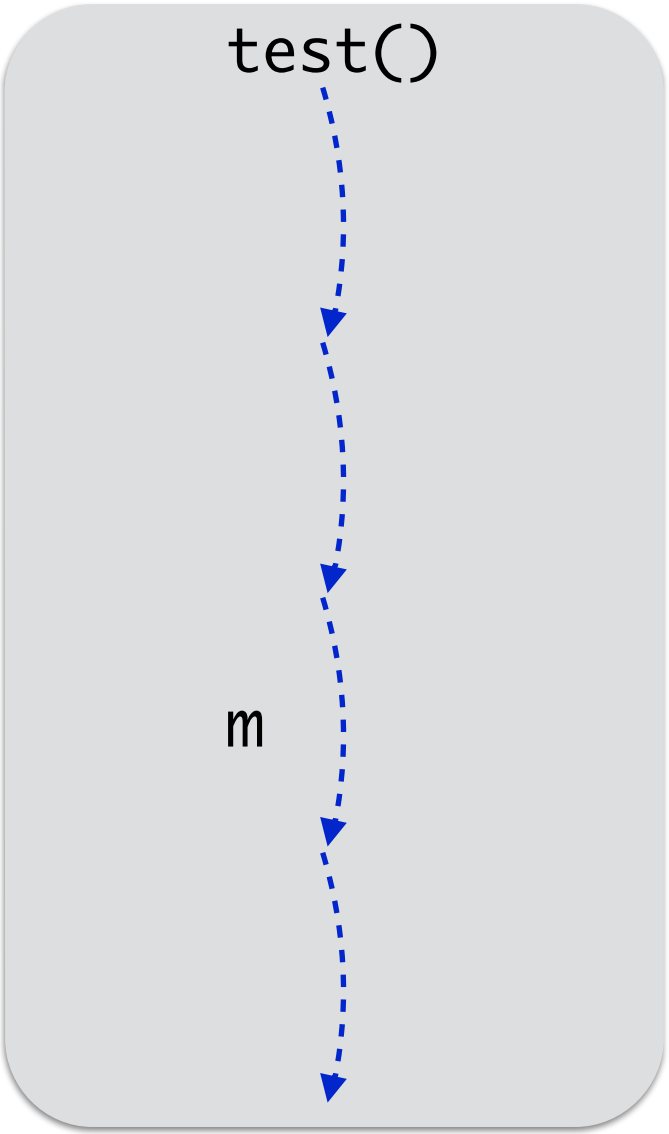

```
test() {
  …
  m();
  …
}
```

Redundancy as Test Oracles



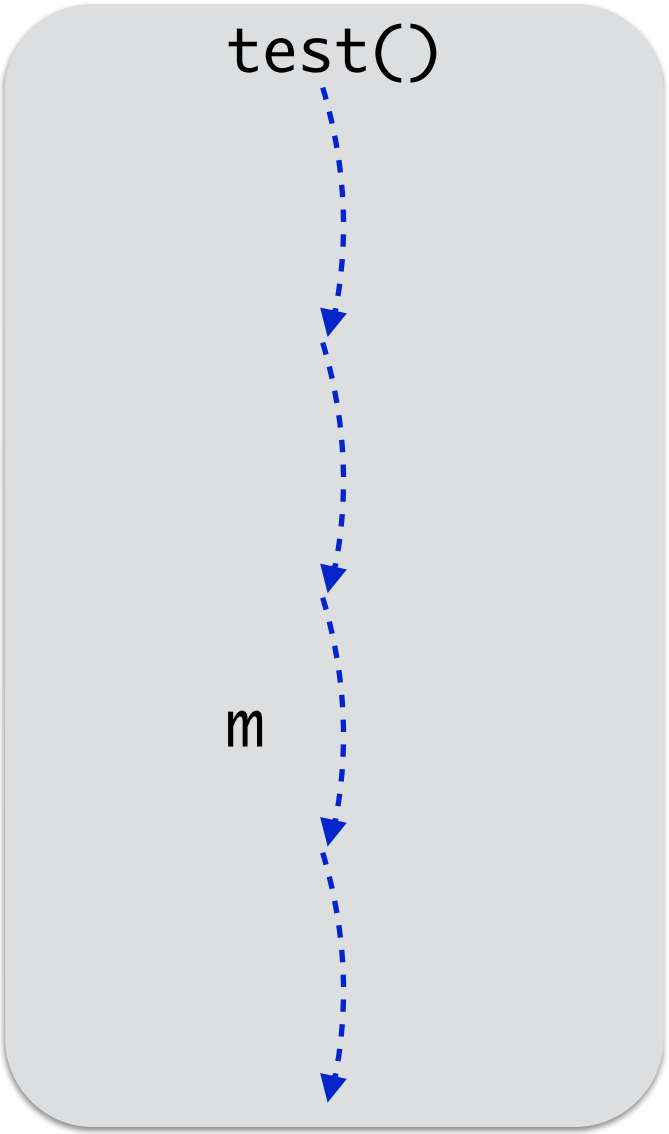

```
test() {
  …
  m();
  …
}
```

Redundancy as Test Oracles




```
test() {
  …
  m();
  …
}
```

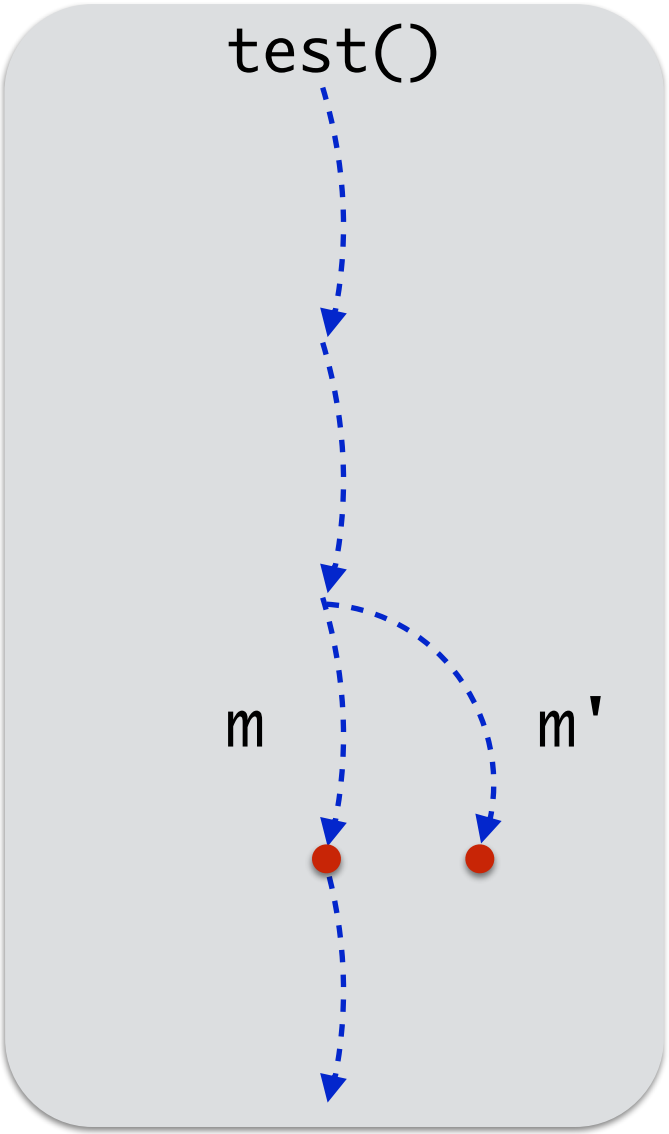
Redundancy as Test Oracles




```
test() {
  …
  m();
  …
}
```

```
m ≡ m'
```

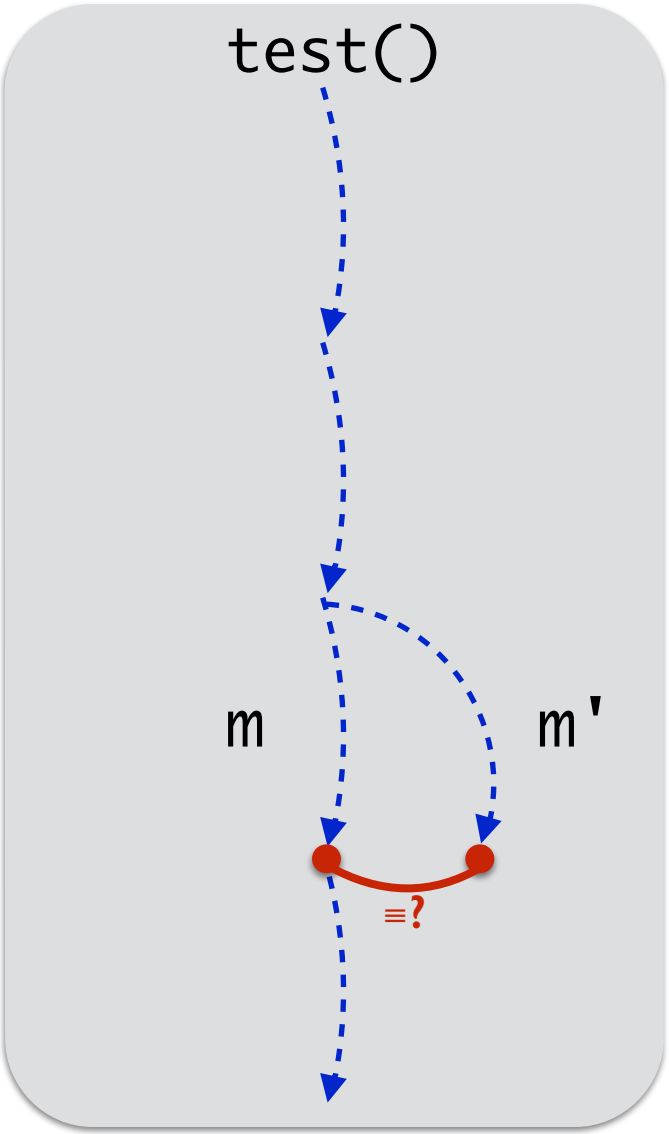
Redundancy as Test Oracles




```
test() {
  …
  m();
  …
}
```

```
m ≡ m'
```

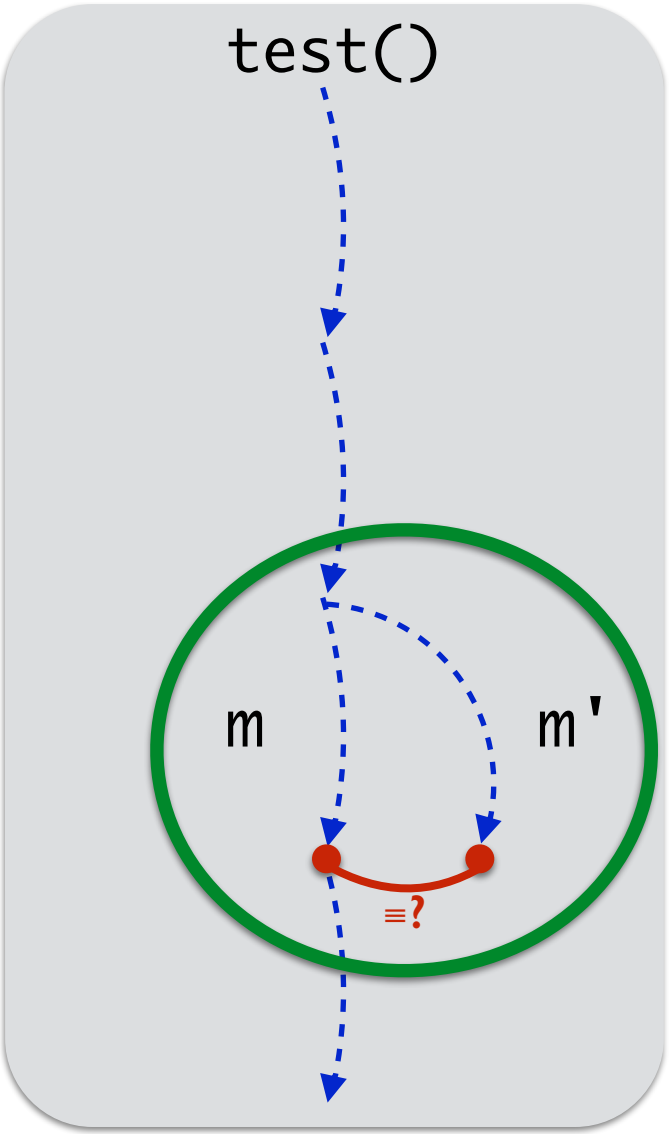
Redundancy as Test Oracles




```
test() {
  …
  m();
  …
}
```

```
m ≡ m'
```

Redundancy as Test Oracles

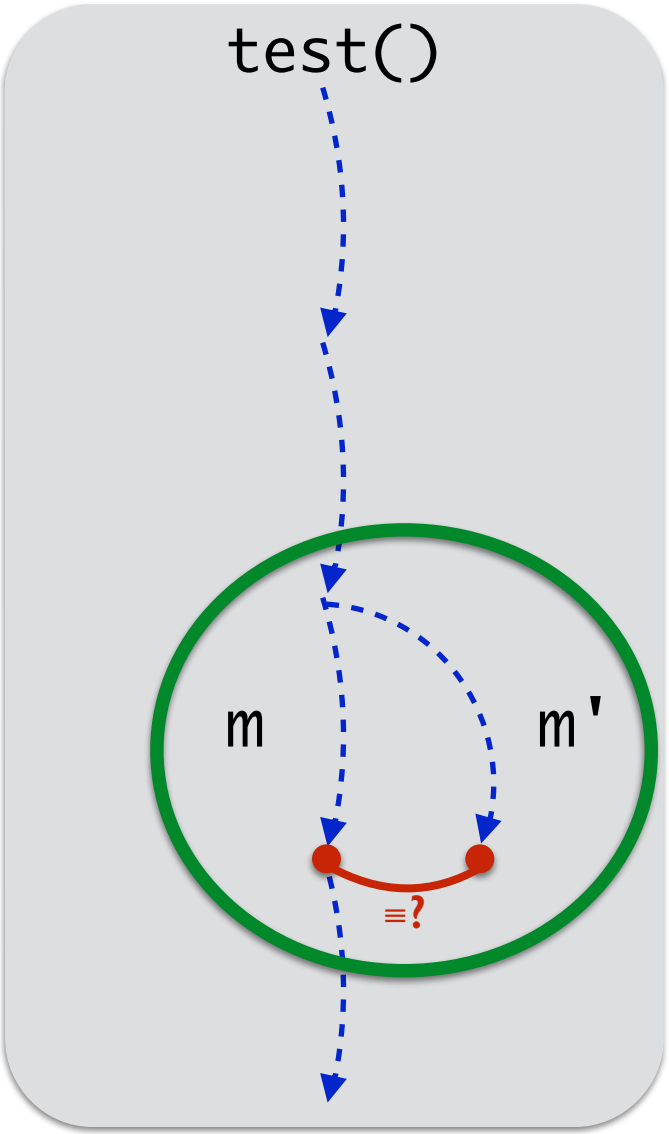



```
test() {
  …
  m();
  …
}
```

```
m ≡ m'
```

Cross-Checking Oracle

Redundancy as Test Oracles




```
test() {
  …
  m();
  …
}
```

```
m ≡ m'
```

Cross-Checking Oracle

1. Find where to apply equivalence
2. Run both original and equivalent code
3. Compare outcome

Where to Apply Equivalence

```
public class ArrayListMultimapTest {
    public void testGetRandomAccess() {
        Multimap<String, Integer> multimap = create();
        multimap.put(”foo”, 1);
        multimap.put(”foo”, 3);
        assertTrue(multimap.get("foo") instanceof RandomAccess);
        assertTrue(multimap.get("bar") instanceof RandomAccess);
    }
}
```

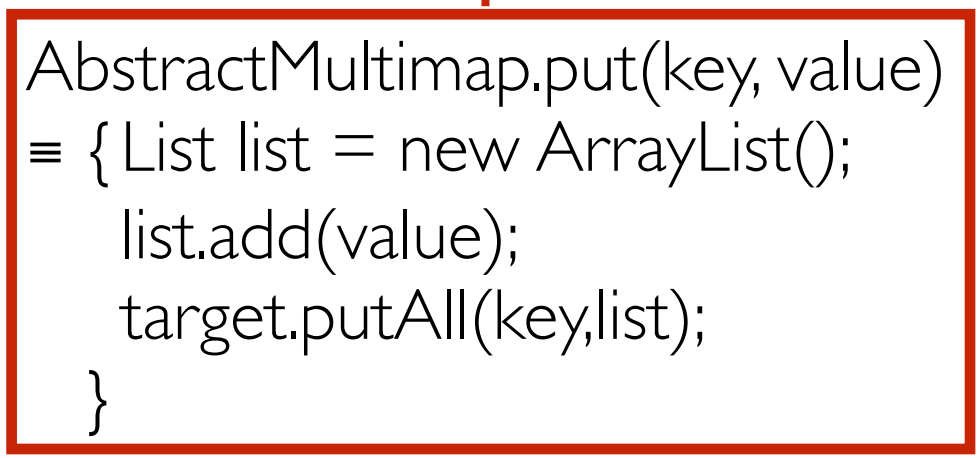
Where to Apply Equivalence

```
public class ArrayListMultimapTest {
    public void testGetRandomAccess() {
        Multimap<String, Integer> multimap = create();
        multimap.put(”foo”, 1);
        multimap.put(”foo”, 3);
        assertTrue(multimap.get("foo") instanceof RandomAccess);
        assertTrue(multimap.get("bar") instanceof RandomAccess);
    }
}
```

```
AbstractMultimap.put(key, value)
≡ { List list = new ArrayList();
    list.add(value);
    target.putAll(key,list);
  }
```

Where to Apply Equivalence

```
public class ArrayListMultimapTest {
    public void testGetRandomAccess() {
        Multimap<String, Integer> multimap = create();
        multimap.put(”foo”, 1);
        multimap.put(”foo”, 3);
        assertTrue(multimap.get("foo") instanceof RandomAccess);
        assertTrue(multimap.get("bar") instanceof RandomAccess);
    }
}
```



Where to Apply Equivalence

```
public class ExampleOracle extends CrossCheckOracle {

    @PointCut("call(boolean com.google.collect.AbstractMultimap.put(Object, Object))")
     public boolean advice(Multimap<Object, Object> map, Object key, Object value) {
         this.target = map;
         this.key = key;
         this.value = value;

         return executeCrossCheck();
     }

     boolean originalCall() {
         return target.put(key,value);
     }

     boolean equivalentCode() {
         List list = new ArrayList();
         list.add(value);
         return target.putAll(key,list);
     }

}
```

AbstractMultimap.put(key, value)
≡ { List list = new ArrayList();
  list.add(value);
  target.putAll(key,list);
 }

Where to Apply Equivalence

```
public class ExampleOracle extends CrossCheckOracle {

    @PointCut("call(boolean com.google.collect.AbstractMultimap.put(Object, Object))")
     public boolean advice(Multimap<Object, Object> map, Object key, Object value) {
         this.target = map;
         this.key = key;
         this.value = value;

         return executeCrossCheck();
     }

     boolean originalCall() {
         return target.put(key,value);
     }

     boolean equivalentCode() {
         List list = new ArrayList();
         list.add(value);
         return target.putAll(key,list);
     }

}
```

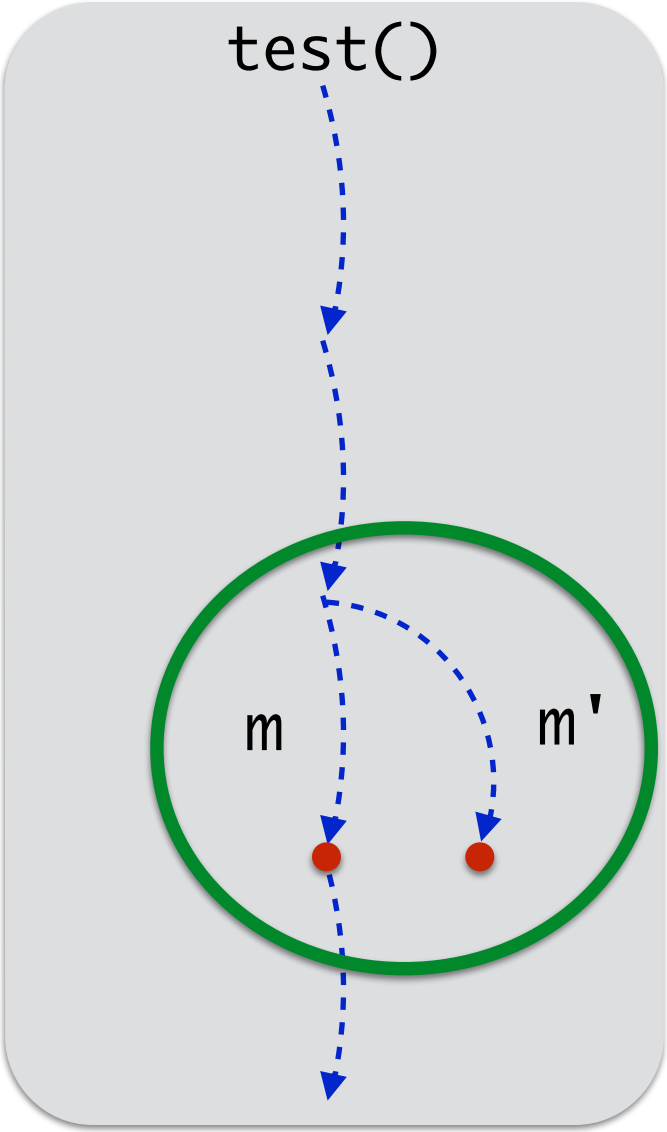
AbstractMultimap.put(key, value)
≡ { List list = new ArrayList();
  list.add(value);
  target.putAll(key,list);
 }

Where to Apply Equivalence

```
public class ExampleOracle extends CrossCheckOracle {

    @PointCut("call(boolean com.google.collect.AbstractMultimap.put(Object, Object))")
     public boolean advice(Multimap<Object, Object> map, Object key, Object value) {
         this.target = map;
         this.key = key;
         this.value = value;

         return executeCrossCheck();
     }

     boolean originalCall() {
         return target.put(key,value);
     }

     boolean equivalentCode() {
         List list = new ArrayList();
         list.add(value);
         return target.putAll(key,list);
     }

}
```

AbstractMultimap.put(key, value)
≡ { List list = new ArrayList();
list.add(value);
target.putAll(key,list);
}

Run Code Independently



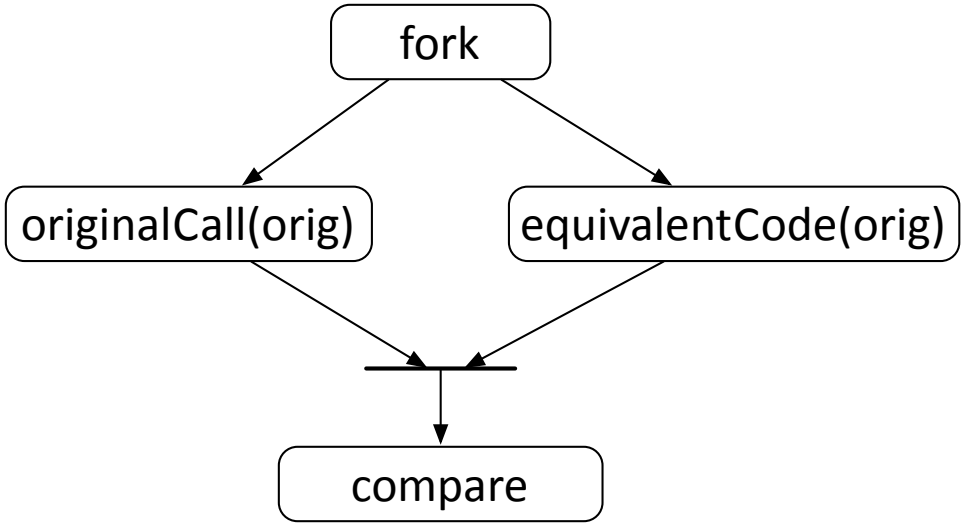
As **independent** as possible..

(to avoid side effects)

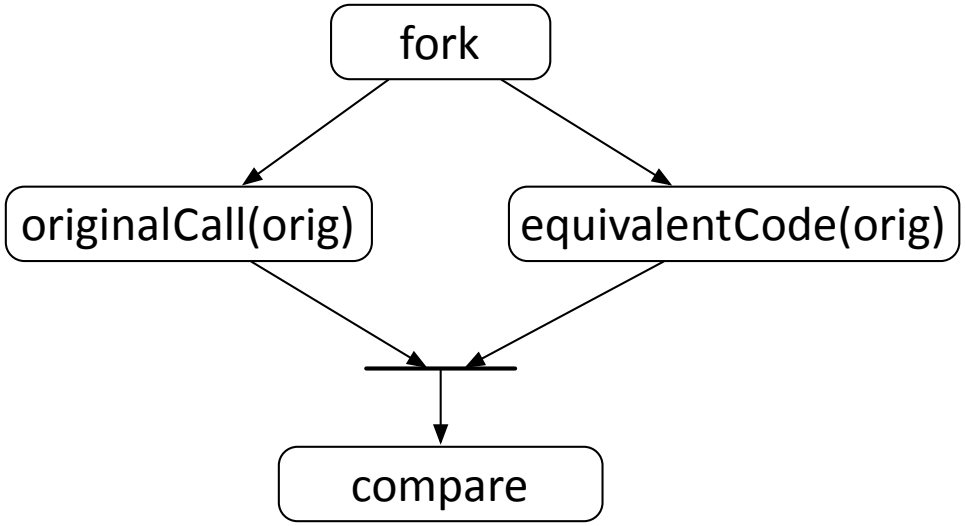
..but still **coupled**

(to allow comparison)

Run Code Independently

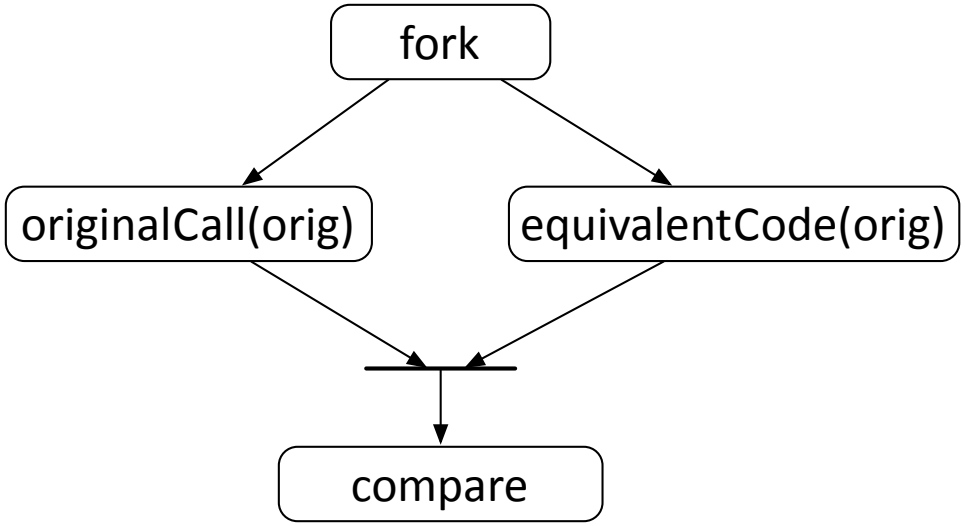


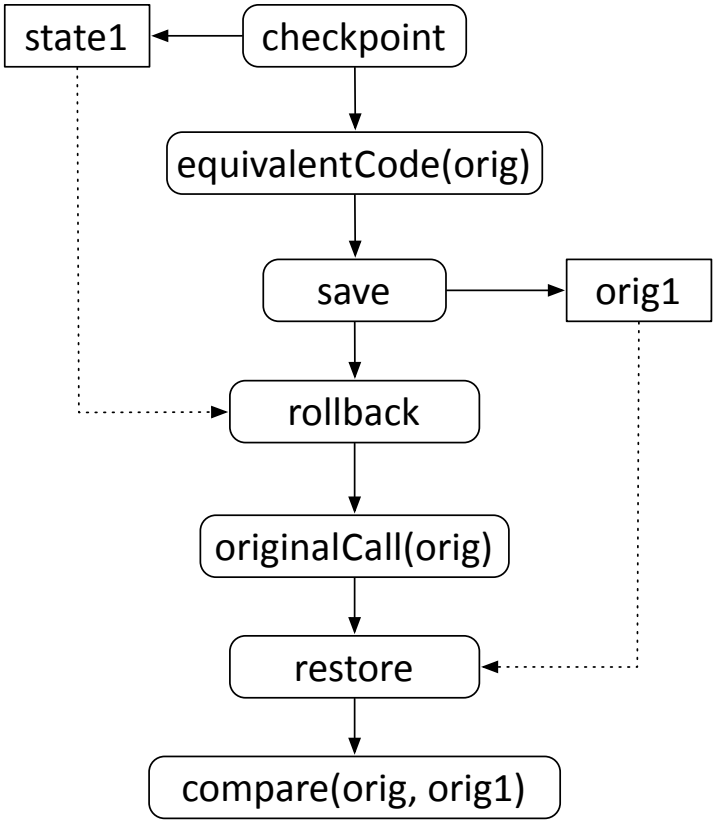
Run Code Independently



Fork a JVM?

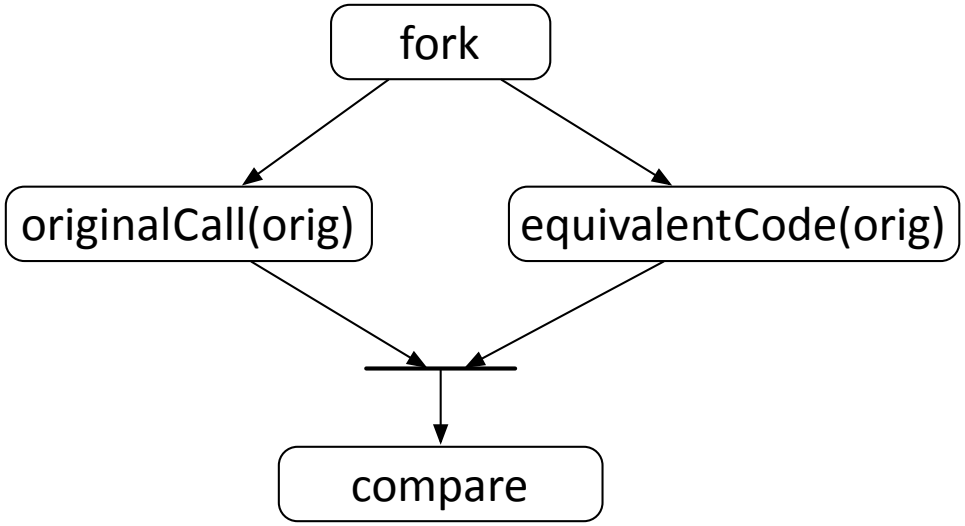
Run Code Independently

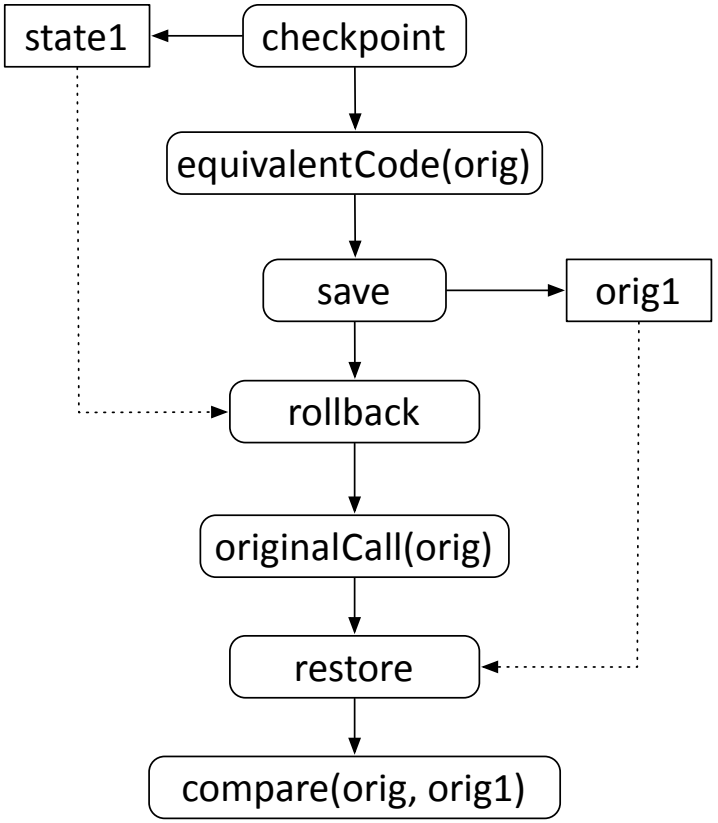




Fork a JVM?

Run Code Independently





Fork a JVM?

Store/load states?

Run Code Independently

```
public class ExampleOracle extends CrossCheckOracle {

   @PointCut("call(boolean com.google.collect.AbstractMultimap.put(Object, Object))")
    public boolean advice(Multimap<Object, Object> map, Object key, Object value) {
        ...
        return executeCrossCheck();
    }

    boolean executeCrossCheck() {
        Object target_orig = target;
        Object target_copy = deep_clone(target);

        Object orig_res = originalCall(target_orig);
        Object copy_res = equivalentCode(target_copy);

        return (orig_res ≡ copy_res) && (target_orig ≡ target_copy);
    }

}
```

Run Code Independently

```
public class ExampleOracle extends CrossCheckOracle {

   @PointCut("call(boolean com.google.collect.AbstractMultimap.put(Object, Object))")
    public boolean advice(Multimap<Object, Object> map, Object key, Object value) {
        ...
        return executeCrossCheck();
    }

    boolean executeCrossCheck() {
        Object target_orig = target;
        Object target_copy = deep_clone(target);

        Object orig_res = originalCall(target_orig);
        Object copy_res = equivalentCode(target_copy);

        return (orig_res ≡ copy_res) && (target_orig ≡ target_copy);
    }

}
```

Run Code Independently

```
public class ExampleOracle extends CrossCheckOracle {

   @PointCut("call(boolean com.google.collect.AbstractMultimap.put(Object, Object))")
    public boolean advice(Multimap<Object, Object> map, Object key, Object value) {
        ...
        return executeCrossCheck();
    }

    boolean executeCrossCheck() {
        Object target_orig = target;
        Object target_copy = deep_clone(target);

        Object orig_res = originalCall(target_orig);
        Object copy_res = equivalentCode(target_copy);

        return (orig_res ≡ copy_res) && (target_orig ≡ target_copy);
    }

}
```

Deep-clone object

Sequential execution

Run Code Independently

```
public class ExampleOracle extends CrossCheckOracle {

   @PointCut("call(boolean com.google.collect.AbstractMultimap.put(Object, Object))")
    public boolean advice(Multimap<Object, Object> map, Object key, Object value) {
        ...
        return executeCrossCheck();
    }

    boolean executeCrossCheck() {
        Object target_orig = target;
        Object target_copy = deep_clone(target);

        Object orig_res = originalCall(target_orig);
        Object copy_res = equivalentCode(target_copy);

        return (orig_res ≡ copy_res) && (target_orig ≡ target_copy);
    }

}
```

Deep-clone object

Sequential execution

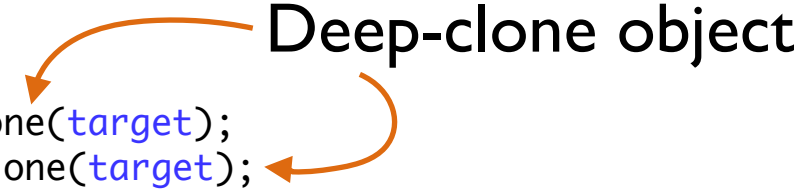
Outcome comparison

Run Code Independently

```
public class ExampleOracle extends CrossCheckOracle {
    ...

    boolean executeCrossCheck() {
        Object target_orig = target;
        Object target_copy = deep_clone(target);
        Object target_copy2 = deep_clone(target);

        if (target_orig ≠ target_copy) return true;

        Object orig_res = originalCall(target_orig);

        Object orig_res2 = originalCall(target_copy2);
        if (orig_res ≠ orig_res2 && target_orig ≠ target_copy2) return true;

        Object copy_res = equivalentCode(target_copy);

        return (orig_res ≡ copy_res) && (target_orig ≡ target_copy);
    }

}
```

Run Code Independently

```
public class ExampleOracle extends CrossCheckOracle {
    ...

    boolean executeCrossCheck() {
        Object target_orig = target;
        Object target_copy = deep_clone(target);
        Object target_copy2 = deep_clone(target);

        if (target_orig ≠ target_copy) return true;

        Object orig_res = originalCall(target_orig);

        Object orig_res2 = originalCall(target_copy2);
        if (orig_res ≠ orig_res2 && target_orig ≠ target_copy2) return true;

        Object copy_res = equivalentCode(target_copy);

        return (orig_res ≡ copy_res) && (target_orig ≡ target_copy);
    }

}
```



Run Code Independently

```
public class ExampleOracle extends CrossCheckOracle {
    ...

    boolean executeCrossCheck() {
        Object target_orig = target;
        Object target_copy = deep_clone(target);
        Object target_copy2 = deep_clone(target);

        if (target_orig ≠ target_copy) return true;

        Object orig_res = originalCall(target_orig);

        Object orig_res2 = originalCall(target_copy2);
        if (orig_res ≠ orig_res2 && target_orig ≠ target_copy2) return true;

        Object copy_res = equivalentCode(target_copy);

        return (orig_res ≡ copy_res) && (target_orig ≡ target_copy);
    }

}
```

Deep-clone object

Test for spurious false positives

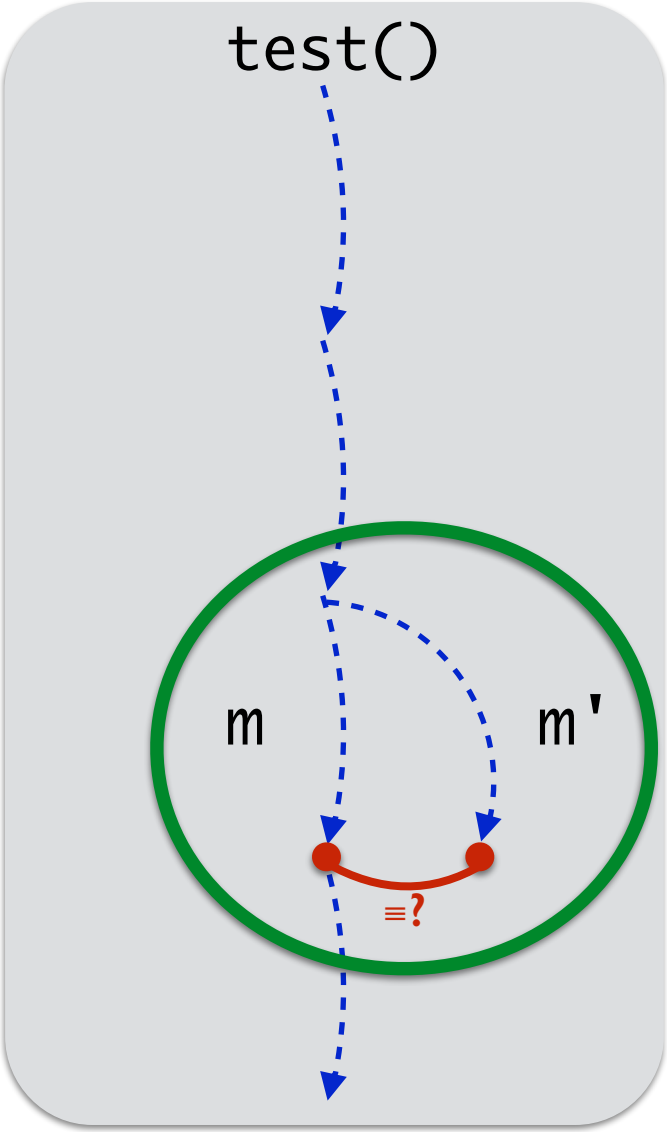
Run Code Independently

```
public class ExampleOracle extends CrossCheckOracle {
    ...

    boolean executeCrossCheck() {
        Object target_orig = target;
        Object target_copy = deep_clone(target);
        Object target_copy2 = deep_clone(target);

        if (target_orig ≠ target_copy) return true;

        Object orig_res = originalCall(target_orig);

        Object orig_res2 = originalCall(target_copy2);
        if (orig_res ≠ orig_res2 && target_orig ≠ target_copy2) return true;

        Object copy_res = equivalentCode(target_copy);

        return (orig_res ≡ copy_res) && (target_orig ≡ target_copy);
    }

}
```

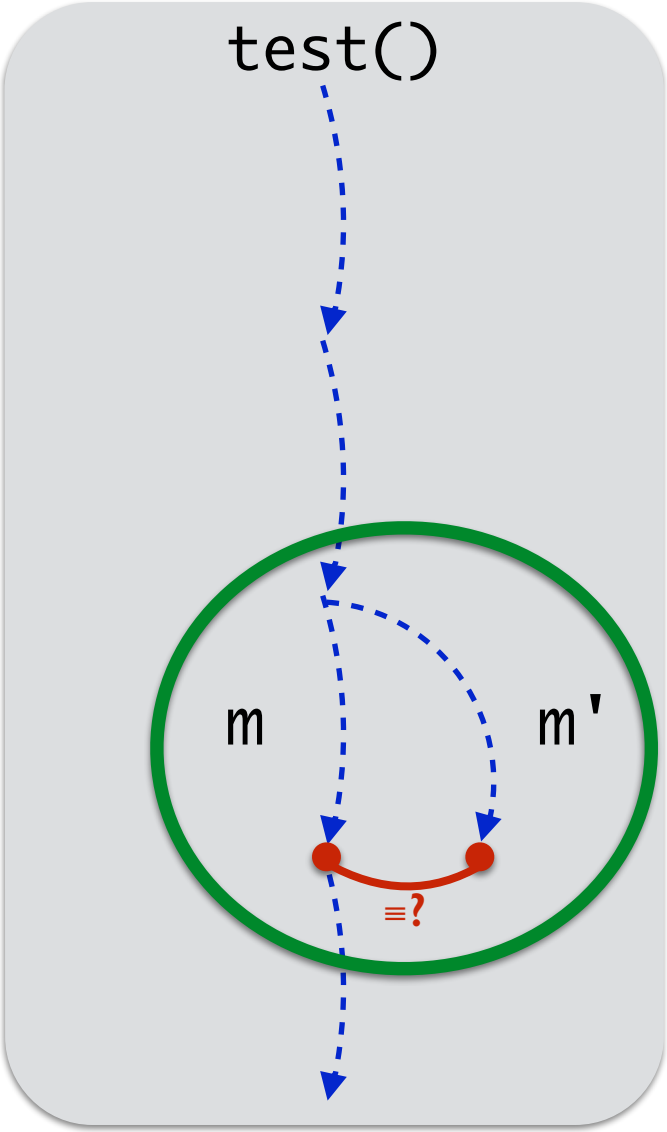
Deep-clone object

Test for spurious false positives

Compare Outcome

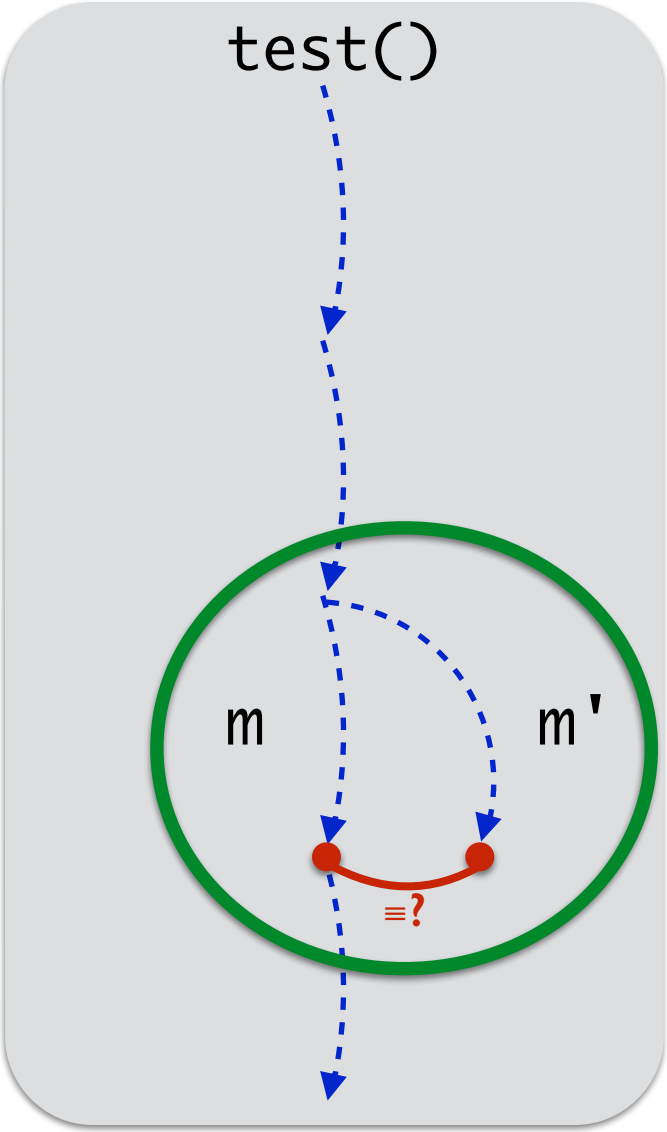


Compare Outcome



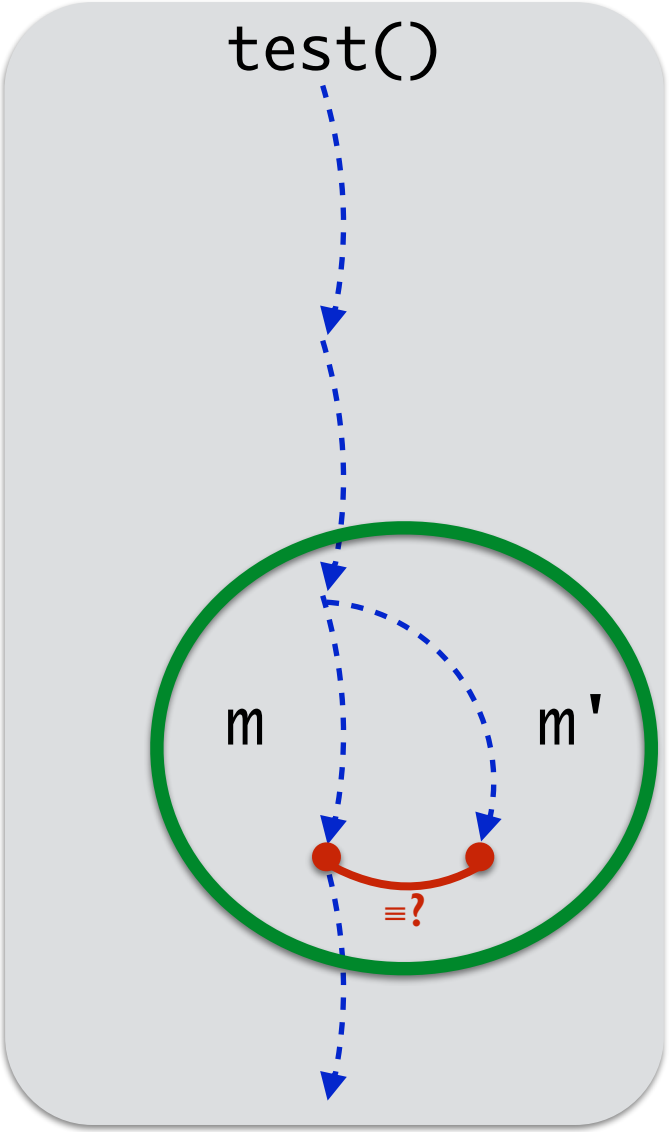

```
public boolean equals(Object object) {
    if (object == this) {
      return true;
    }
    if (object instanceof Multimap) {
      Multimap that = (Multimap) object;
      return this.map.equals(that.asMap());
    }
    return false;
}
```

Compare Outcome




```
public boolean equals(Object object) {
    if (object == this) {
      return true;
    }
    if (object instanceof Multimap) {
      Multimap that = (Multimap) object;
      return this.map.equals(that.asMap());
    }
    return false;
}


public class Object {
     ...
     public boolean equals(Object obj) {
          return (this == obj);
     }
}
```

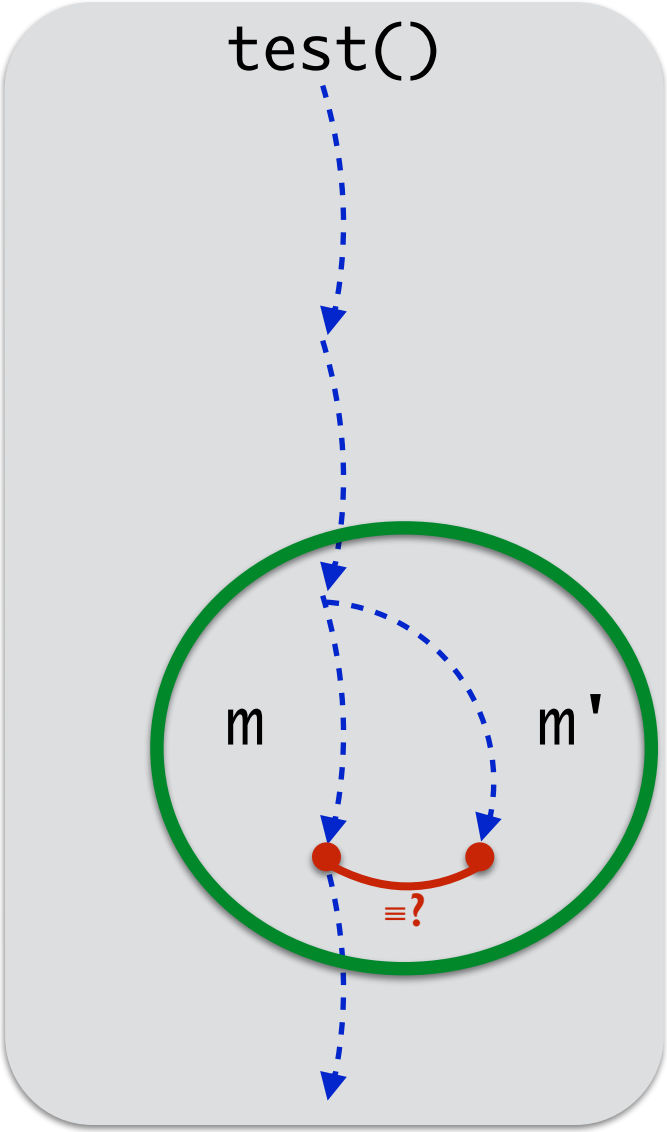
Compare Outcome




```
public boolean equals(Object object) {
    if (object == this) {
      return true;
    }
    if (object instanceof Multimap) {
      Multimap that = (Multimap) object;
      return this.map.equals(that.asMap());
    }
    return false;
}


public class Object {
     ...
     public boolean equals(Object obj) {
          return (this == obj);
     }
}
```

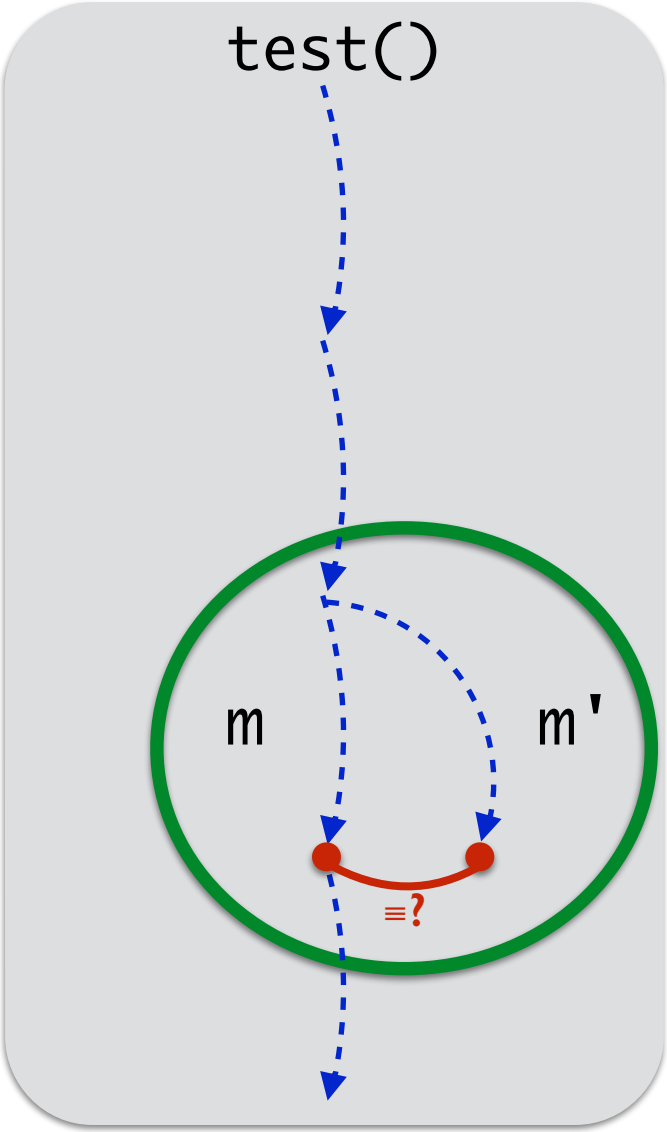
False positive!

Compare Outcome




```
public boolean equals(Object object) {
    if (object == this) {
      return true;
    }
    if (object instanceof Multimap) {
      Multimap that = (Multimap) object;
      return this.map.equals(that.asMap());
    }
    return false;
}


public class Object {
     ...
     public boolean equals(Object obj) {
          if (executingComparingOutcome) {
               return observationalEquals(this, obj);
          }
          else {
               return (this == obj);
          }
     }
}
```

Compare Outcome



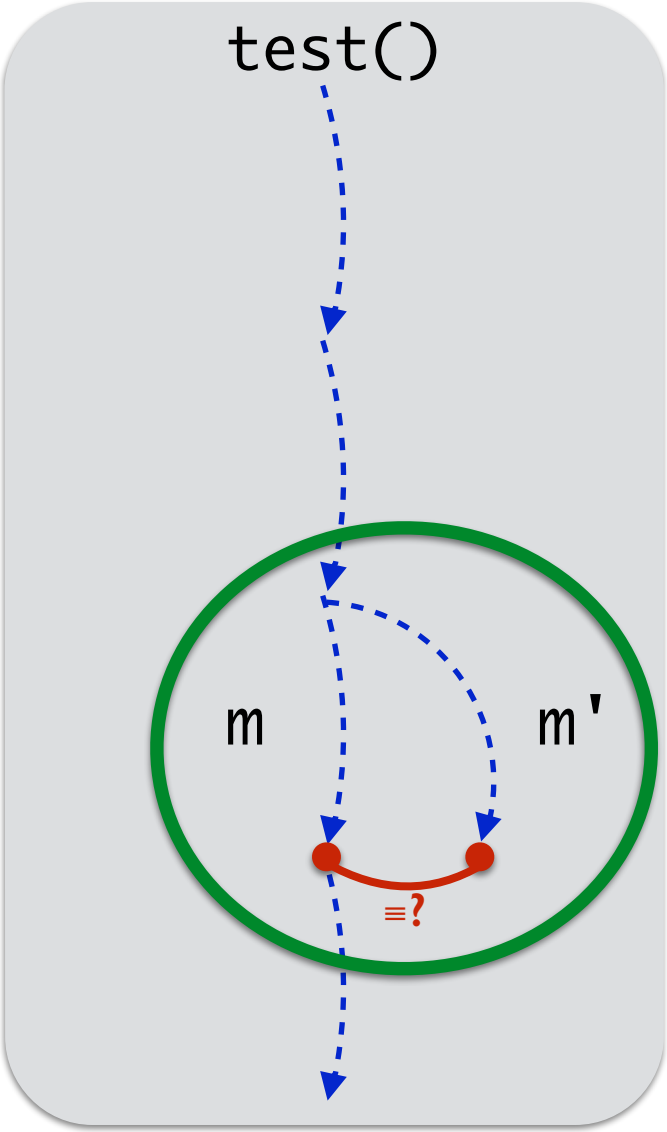

```
public boolean equals(Object object) {
    if (object == this) {
      return true;
    }
    if (object instanceof Multimap) {
      Multimap that = (Multimap) object;
      return this.map.equals(that.asMap());
    }
    return false;
}
```

Observational equivalence

obj1

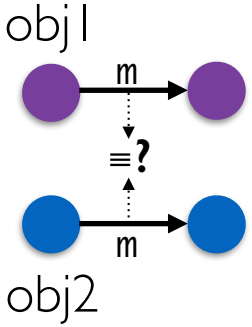
obj2

Compare Outcome

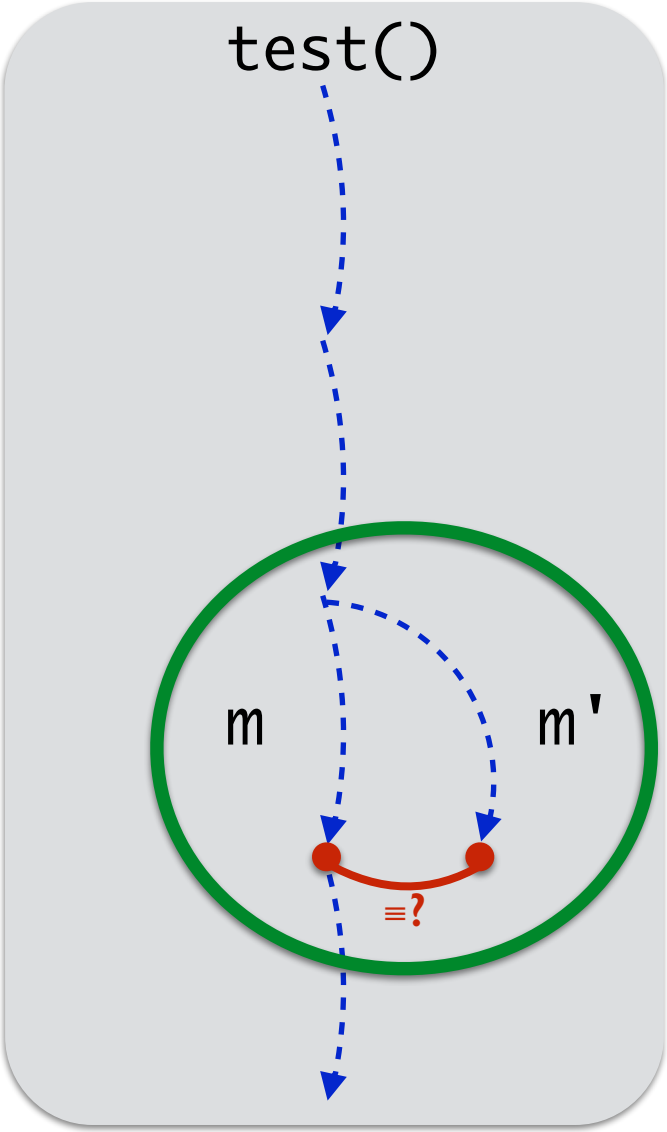



```
public boolean equals(Object object) {
    if (object == this) {
      return true;
    }
    if (object instanceof Multimap) {
      Multimap that = (Multimap) object;
      return this.map.equals(that.asMap());
    }
    return false;
}
```

Observational equivalence

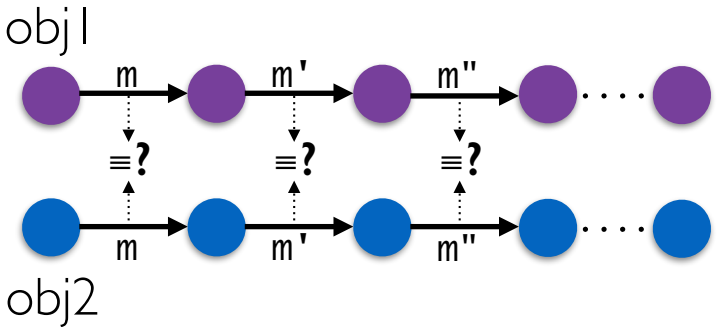


Compare Outcome




```
public boolean equals(Object object) {
    if (object == this) {
      return true;
    }
    if (object instanceof Multimap) {
      Multimap that = (Multimap) object;
      return this.map.equals(that.asMap());
    }
    return false;
}
```

Observational equivalence



Evaluation

Can we exploit the intrinsic redundancy of software
to generate cross-checking oracles?
And are those oracles effective in revealing faults in real systems?

Evaluation

Can we exploit the intrinsic redundancy of software to generate cross-checking oracles?
And are those oracles effective in revealing faults in real systems?

RQ1 Are cross-checking oracles effective and useful with automatically generated test suites?

RQ2 Are cross-checking oracles effective and useful with hand-written test suites?

Evaluation

Can we exploit the intrinsic redundancy of software
to generate cross-checking oracles?
And are those oracles effective in revealing faults in real systems?

RQ1 Are cross-checking oracles effective and useful with automatically generated test suites?

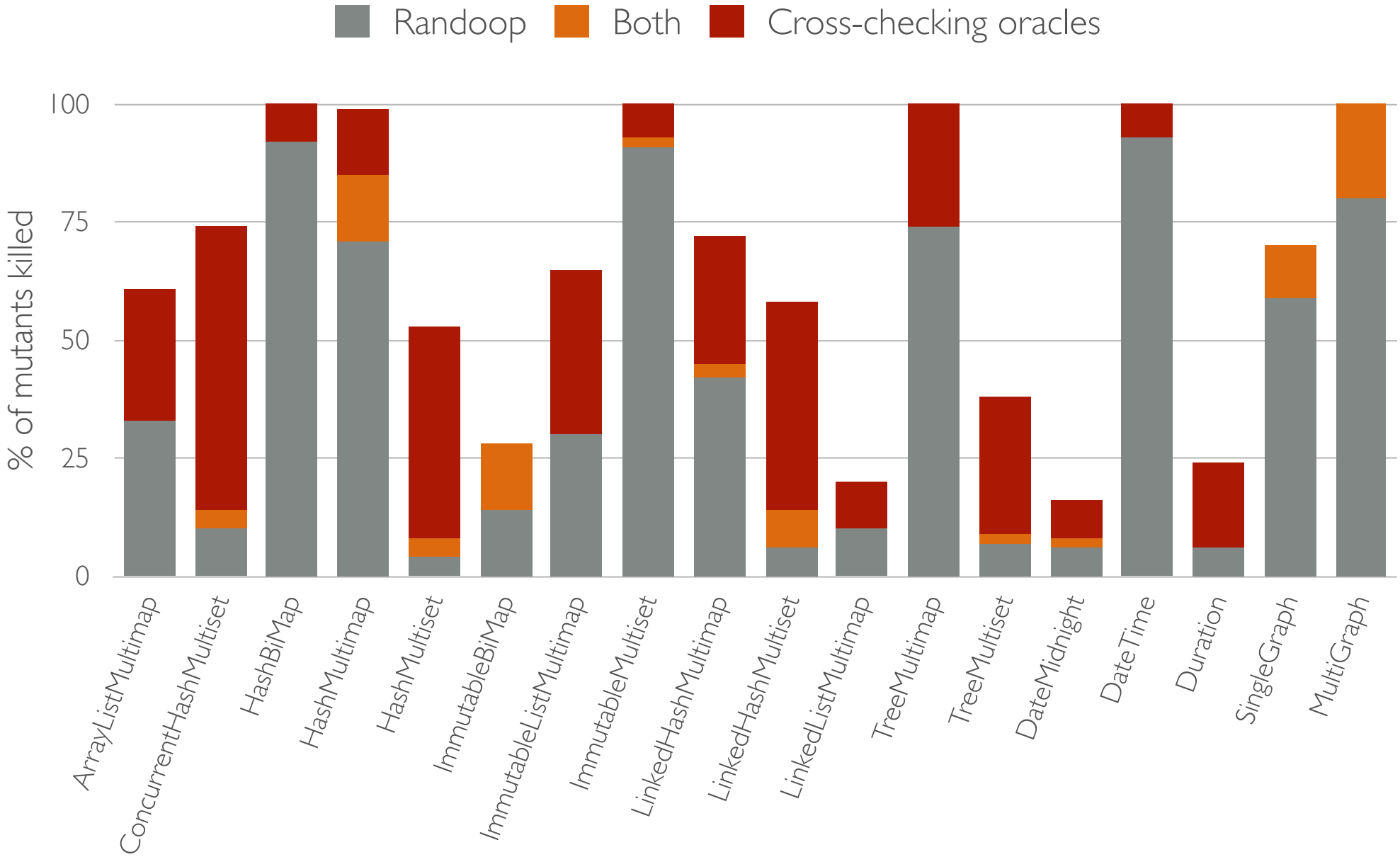
RQ2 Are cross-checking oracles effective and useful with hand-written test suites?

Mutation analysis

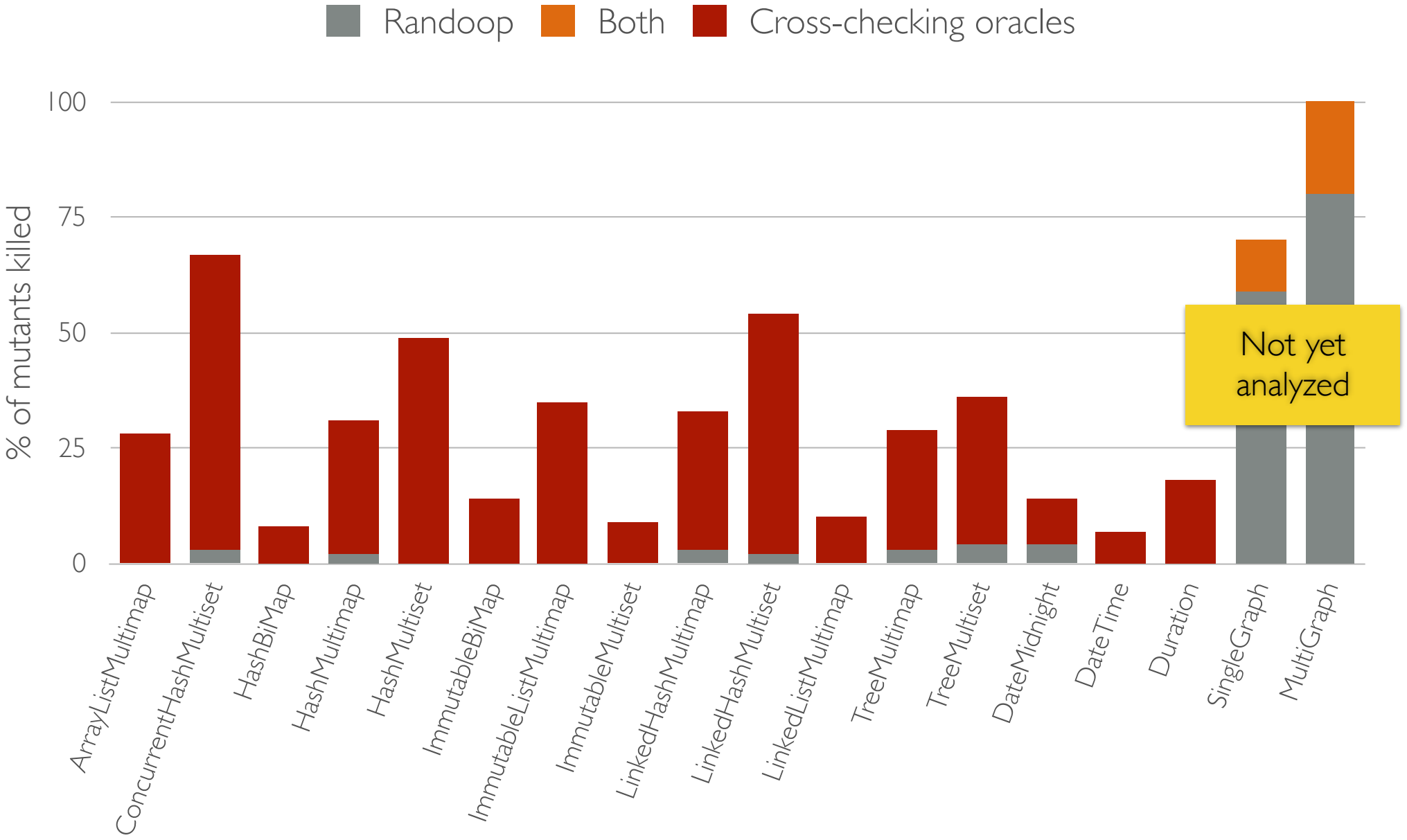
Evaluation

Subject Class		# Methods	# Redundancy Specs.
Guava	ArrayListMultimap	25	29
	ConcurrentHashMultiset	27	32
	HashBiMap	20	16
	HashMultimap	24	29
	HashMultiset	26	30
	ImmutableBimap	25	19
	ImmutableListMultimap	30	34
	ImmutableMultiset	32	50
	LinkedHashMultimap	24	30
	LinkedHashMultiset	26	31
	LinkedListMultimap	24	29
	TreeMultimap	26	28
	TreeMultiset	35	37
JodaTime	DateMidnight	118	20
	DateTime	153	27
	Duration	44	6
GraphStream	SingleGraph	107	34
	MultiGraph	107	34

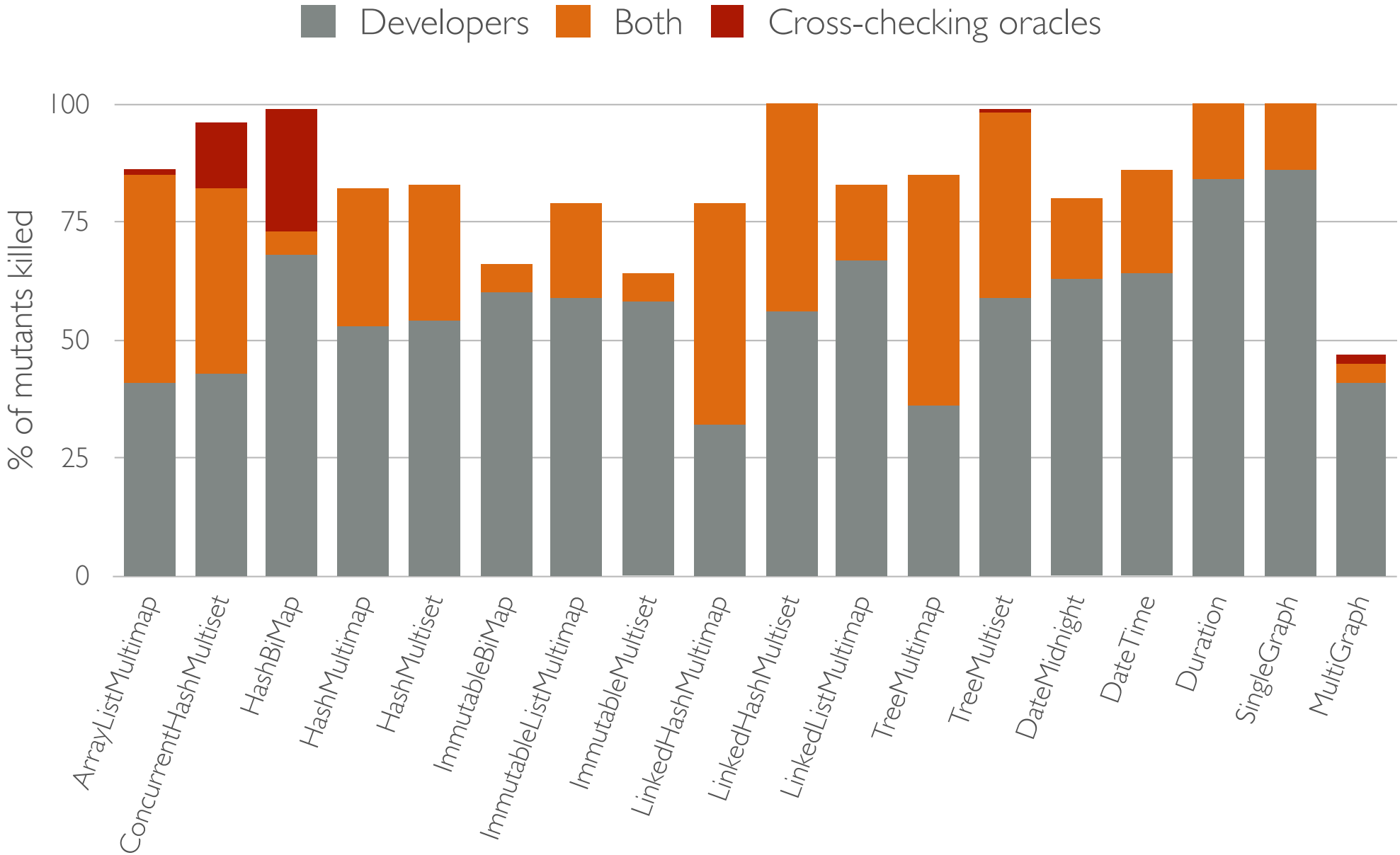
Results: Randoop



Results: Randoop



Results: Hand Written Tests



Results: Accuracy

Reported Test Failures

Subject Class	Generated		Hand-Written	
	total	false	total	false
ArrayListMultimap	11	0	41	0
ConcurrentHashMultiset	64	0	78	0
HashBiMap	1	0	6	0
HashMultimap	12	0	26	0
HashMultiset	35	0	38	0
ImmutableBimap	2	0	2	0
ImmutableListMultimap	8	0	9	0
ImmutableMultiset	7	2	4	0
LinkedHashMultimap	10	0	45	0
LinkedHashMultiset	33	0	39	0
LinkedListMultimap	4	0	12	0
TreeMultimap	10	0	43	0
TreeMultiset	37	0	58	0

Results: Accuracy

Reported Test Failures

Subject Class	Generated		Hand-Written	
	total	false	total	false
ArrayListMultimap	11	0	41	0
ConcurrentHashMultiset	64	0	78	0
HashBiMap	1	0	6	0
HashMultimap	12	0	26	0
HashMultiset	35	0	38	0
ImmutableBimap	2	0	2	0
ImmutableListMultimap	8	0	9	0
ImmutableMultiset	7	2	4	0
LinkedHashMultimap	10	0	45	0
LinkedHashMultiset	33	0	39	0
LinkedListMultimap	4	0	12	0
TreeMultimap	10	0	43	0
TreeMultiset	37	0	58	0

Results: Accuracy

Reported Test Failures

Subject Class	Generated		Hand-Written	
	total	false	total	false
ArrayListMultimap	11	0	41	0
ConcurrentHashMultiset	64	0	78	0
HashBiMap	1	0	6	0
HashMultimap	12	0	26	0
HashMultiset	35	0	38	0
ImmutableBimap	2	0	2	0
ImmutableListMultimap	8	0	9	0
ImmutableMultiset	7	2	4	0
LinkedHashMultimap	10	0	45	0
LinkedHashMultiset	33	0	39	0
LinkedListMultimap	4	0	12	0
TreeMultimap	10	0	43	0
TreeMultiset	37	0	58	0
DateMidnight	11	0	24	0
DateTime	8	0	34	1
Duration	26	0	26	0

Results: Accuracy

Reported Test Failures

Subject Class	Generated		Hand-Written	
	total	false	total	false
ArrayListMultimap	11	0	41	0
ConcurrentHashMultiset	64	0	78	0
HashBiMap	1	0	6	0
HashMultimap	12	0	26	0
HashMultiset	35	0	38	0
ImmutableBimap	2	0	2	0
ImmutableListMultimap	8	0	9	0
ImmutableMultiset	7	2	4	0
LinkedHashMultimap	10	0	45	0
LinkedHashMultiset	33	0	39	0
LinkedListMultimap	4	0	12	0
TreeMultimap	10	0	43	0
TreeMultiset	37	0	58	0
DateMidnight	11	0	24	0
DateTime	8	0	34	1
Duration	26	0	26	0
SingleGraph	52	31	18	0
MultiGraph	230	189	17	0

Future Work

- Independent execution
- Equivalence comparison
- Extensive evaluation

- Implications for fault localization/debugging?
- Effort vs. benefits for developers?

Future Work

Test Oracles

- Independent execution
- Equivalence comparison
- Extensive evaluation

- Implications for fault localization/debugging?
- Effort vs. benefits for developers?

Future Work

Test Oracles

- Independent execution
- Equivalence comparison
- Extensive evaluation

- Implications for fault localization/debugging?
- Effort vs. benefits for developers?

Understand Redundancy

- Study redundancy
- Characterize redundancy
- Why is software redundant?

- Can we automatically identify intrinsic redundancy?

Future Work

Test Oracles

- Independent execution
- Equivalence comparison
- Extensive evaluation

- Implications for fault localization/debugging?
- Effort vs. benefits for developers?

Understand Redundancy

- Study redundancy
- Characterize redundancy
- Why is software redundant?

- Can we automatically identify intrinsic redundancy?

Uses of intrinsic redundancy?

Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

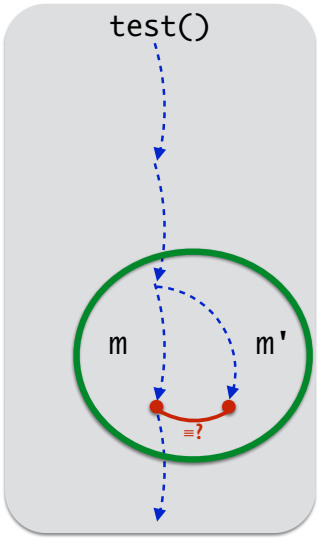
...and this intrinsic redundancy can be exploited for good purposes with no design cost."

Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

...and this intrinsic redundancy can be exploited for good purposes with no design cost."

Redundancy as Test Oracles




```
test() {
  …
  m();
  …
}
```

```
m ≡ m'
```

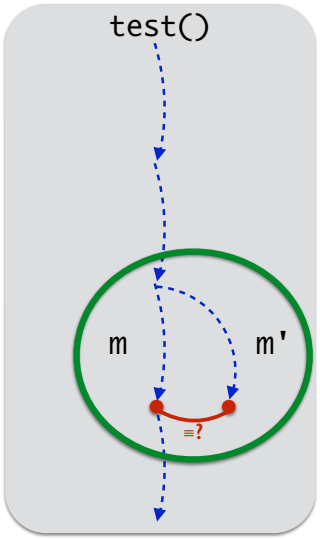
Cross-Checking Oracle

Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

...and this intrinsic redundancy can be exploited for good purposes with no design cost."

Redundancy as Test Oracles

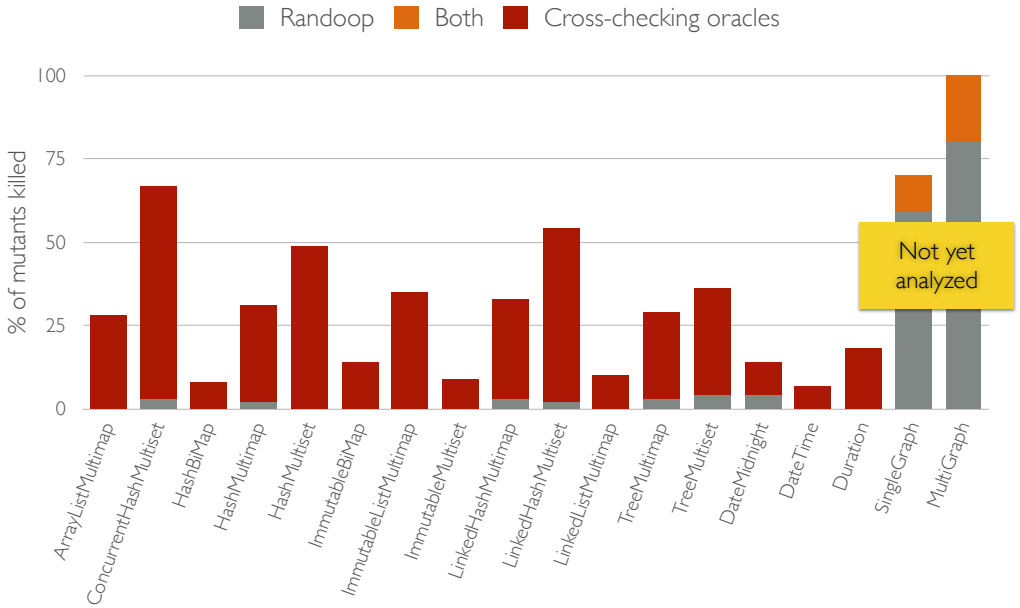



```
test() {
  …
  m();
  …
}
```

`m ≡ m'`

Cross-Checking Oracle

Results: Randoop

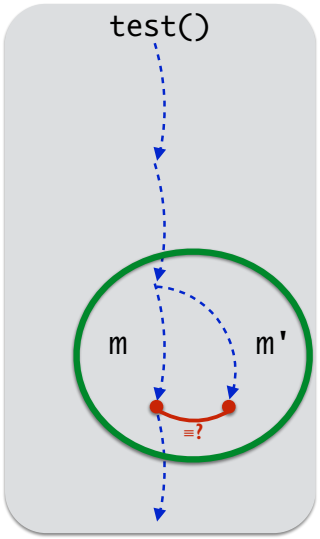


Intrinsic Redundancy

"Modern software systems are indeed **intrinsically** redundant...

...and this intrinsic redundancy can be exploited for good purposes with no design cost."

Redundancy as Test Oracles

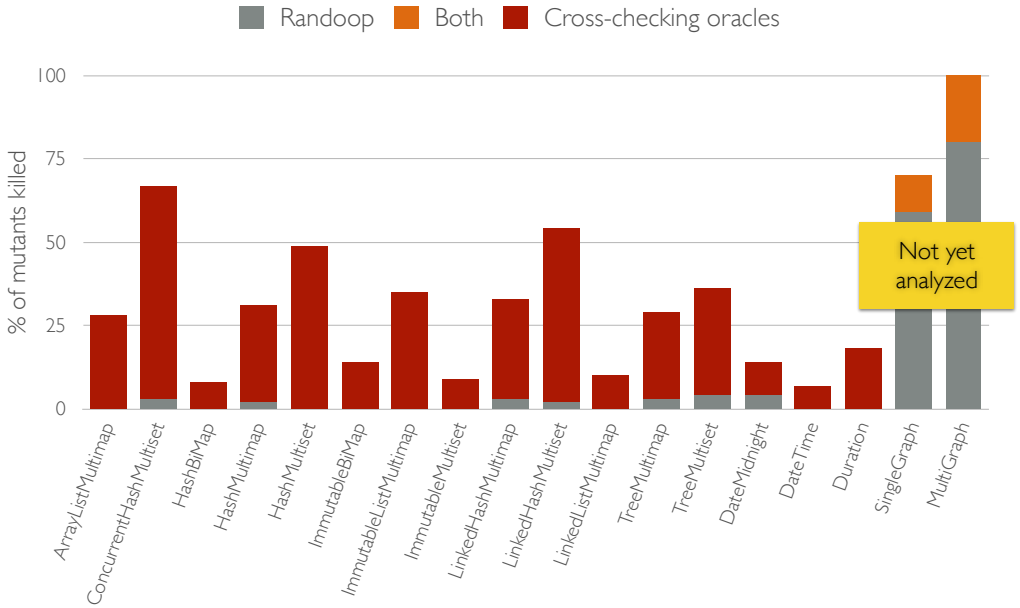



```
test() {
  …
  m();
  …
}
```

```
m ≡ m'
```

Cross-Checking Oracle

Results: Randoop



Results: Hand Written Tests