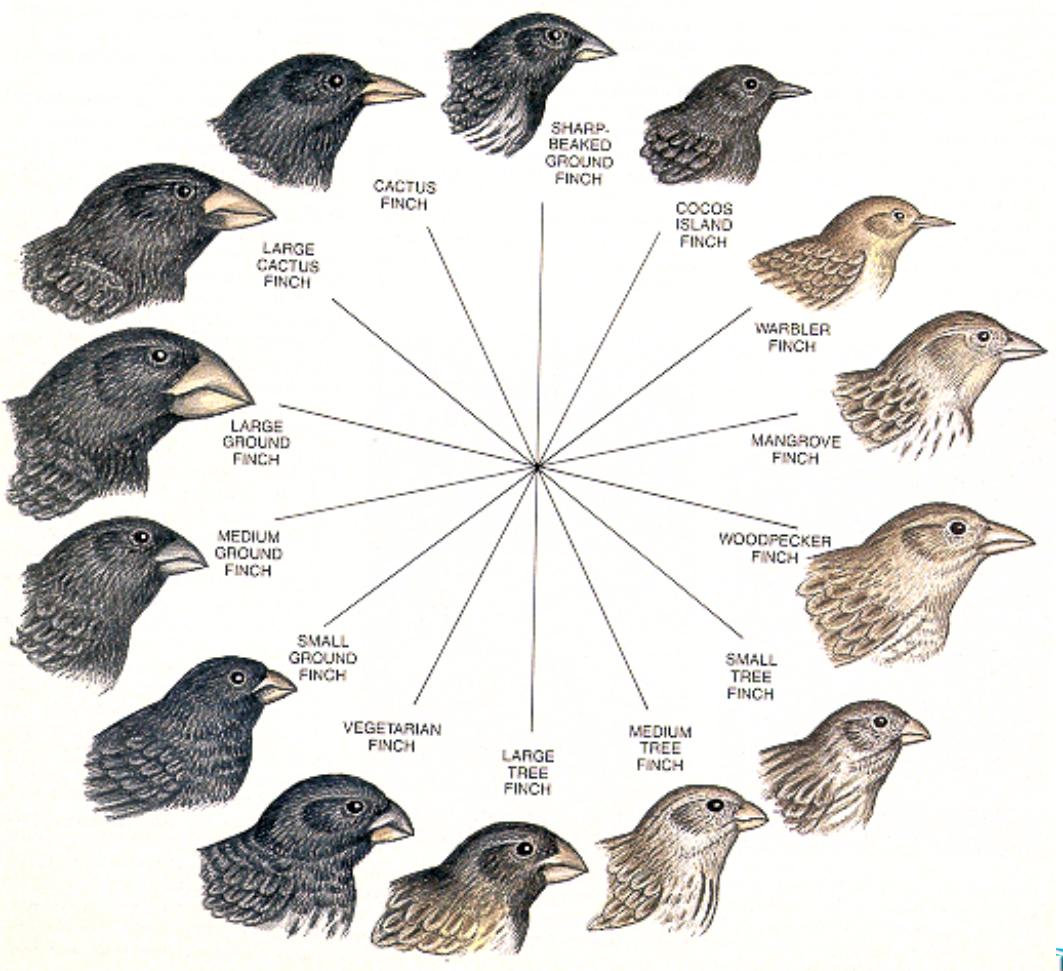
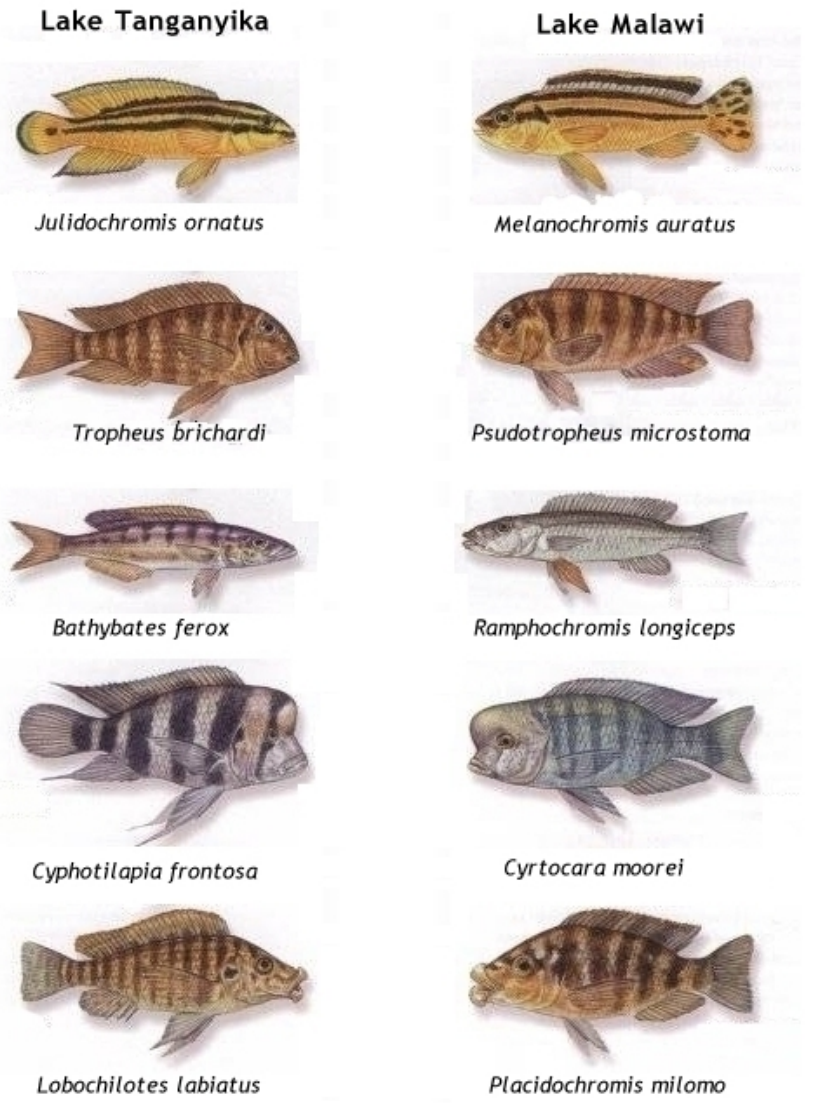
Embedding GI inside the JVM

Benoit Baudry

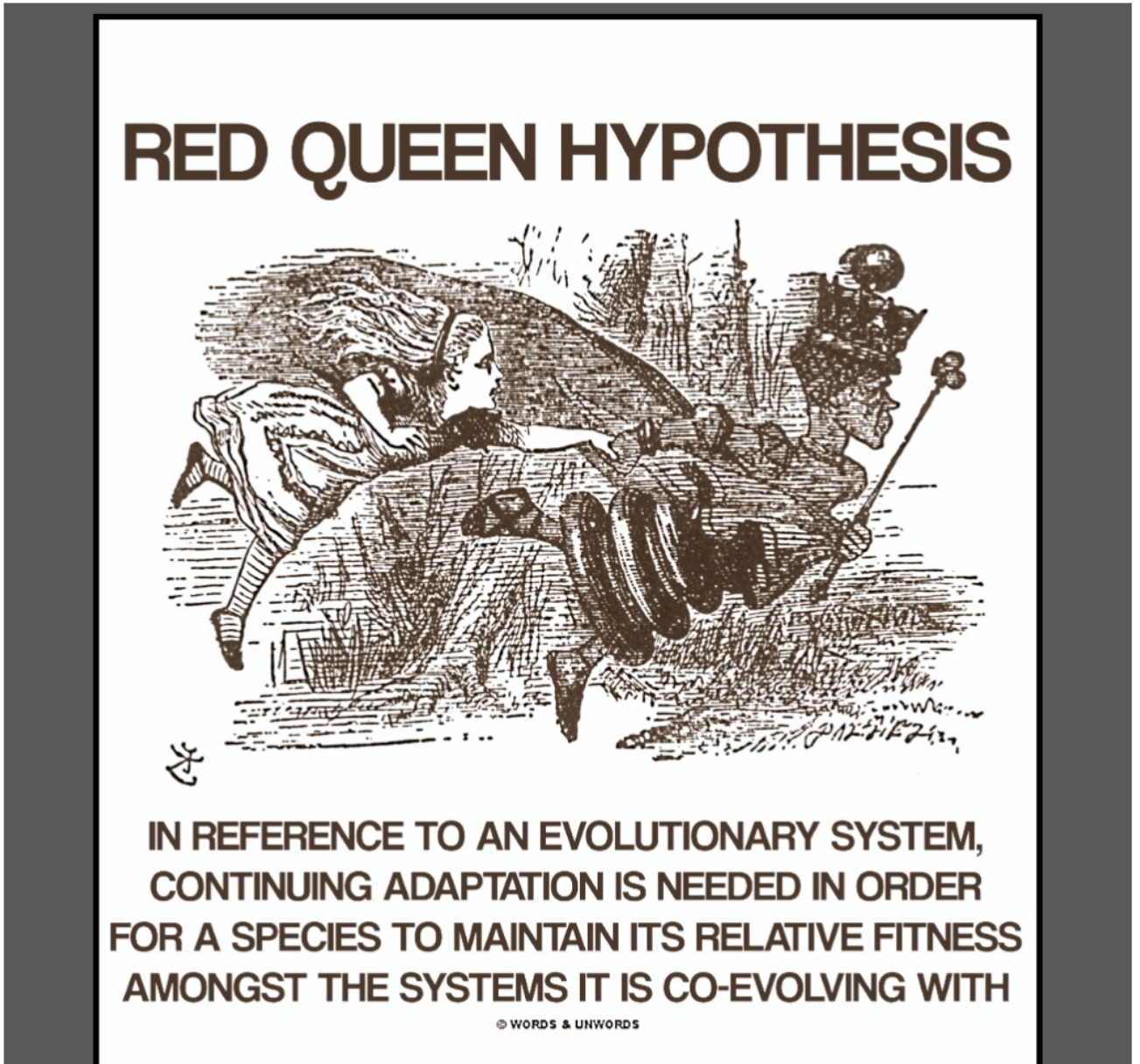
Kwaku Yeboah-Antwi

Specialize to environment .. or not





Open-ended evolution



Continuously evolve

In software

- Code specialization
  - API are large and generic
  - exact usage (e.g., surface really needed) is known only at runtime
- Spontaneous diversification
  - Instantiation creates large quantities of clones
  - diversification at instantiation can increase resilience

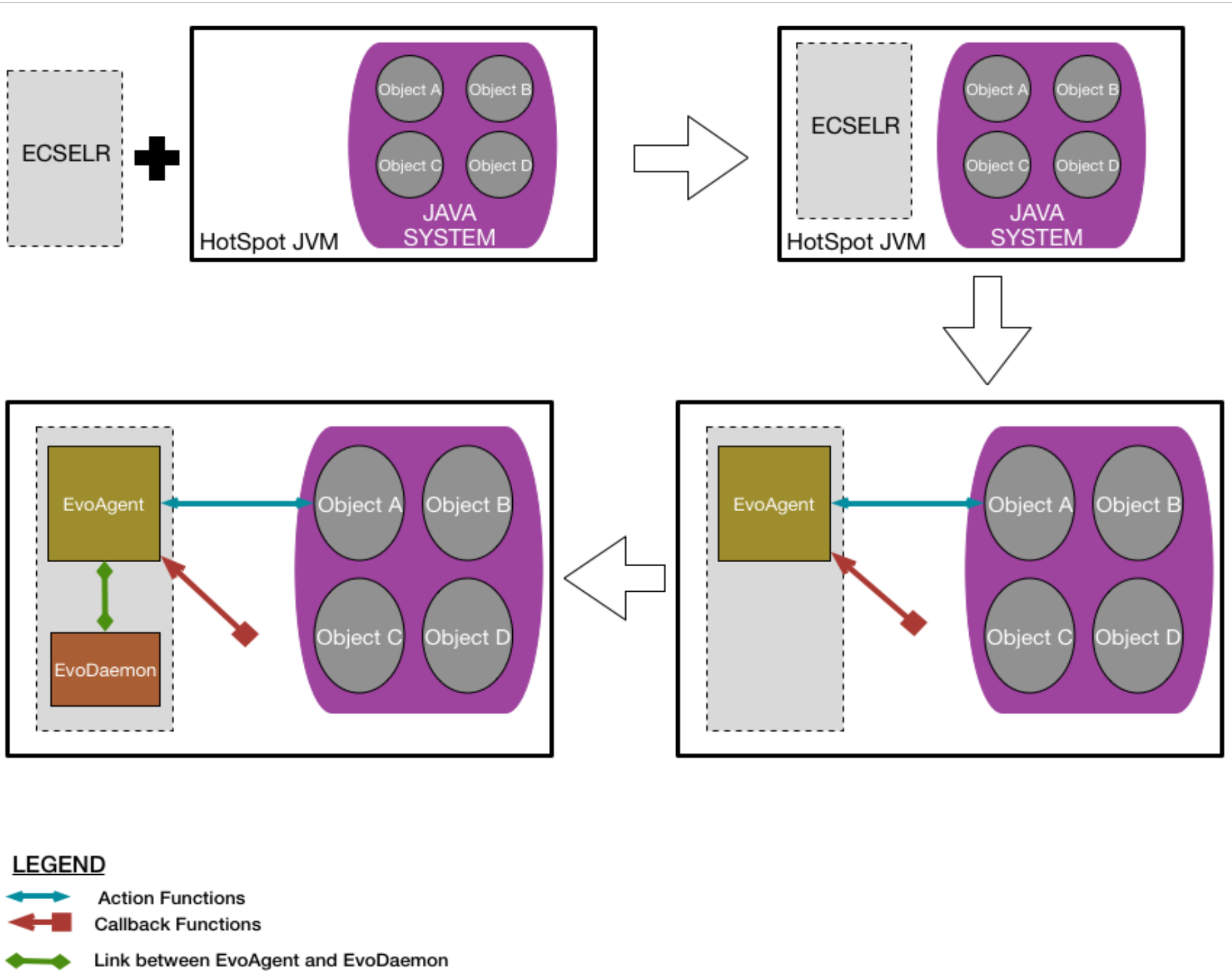
Let's do it in Java

- JVM hardly supports runtime modifications of the bytecode
  - E.g., add, delete or rename fields, methods
- No built-in support
  - to edit the bytecode
  - to steer the search for new code

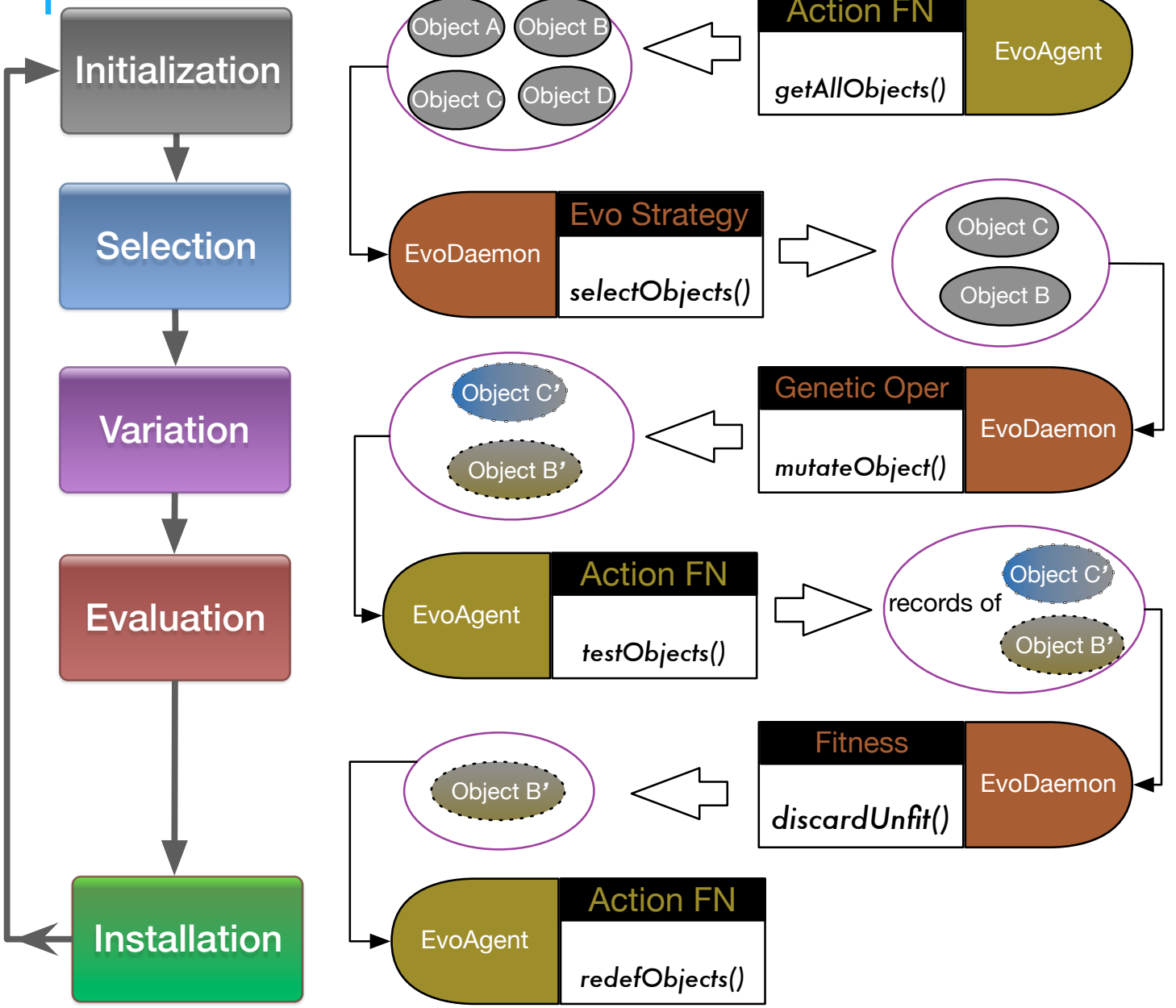
Our Contribution

ECSELR

- ECologically Inspired Software EvoLution @ Runtime
- A patch in the JVM + evolutionnary capacities for Java programs


ECSELR
HotSpot JVM
Object A
Object B
Object C
Object D
JAVA SYSTEM
ECSELR
HotSpot JVM
Object A
Object B
Object C
Object D
JAVA SYSTEM
EvoAgent
EvoDaemon
Object A
Object B
Object C
Object D
EvoAgent
Object A
Object B
Object C
Object D
LEGEND
Action Functions
Callback Functions
Link between EvoAgent and EvoDaemon

The evo loop


Initialization
Selection
Variation
Evaluation
Installation
Object A
Object B
Object C
Object D
Action FN
getAllObjects()
EvoAgent
EvoDaemon
Evo Strategy
selectObjects()
Object C
Object B
Object C'
Object B'
Genetic Oper
mutateObject()
EvoDaemon
EvoAgent
Action FN
testObjects()
records of
Object C'
Object B'
Object B'
Fitness
discardUnfit()
EvoDaemon
EvoAgent
Action FN
redefObjects()

The Initialization and Selection steps

- Analyze all live objects on the heap
- Select a subset according to some function
  - random
  - by size
  - by frequency of usage
- Copy the bytecode of selected objects

The Variation step

- ECSELR embeds default evo operations
  - Method Addition
  - Merge Methods
  - Method Deletion
  - Field Addition Operator
  - Transplantation Operator
  - Random Instruction Operator
  - Passthrough Operator Return

The Evaluation step

- Static check
  - ensure syntactic validity
  - check consistency after evolution: ECSLER embeds checks about the well-formedness of bytecode operatinos
- Dynamic check
  - parallel execution of original and evolved
  - fitness evaluation

Static check

```
public int greaterThan(int intOne, intTwo) {
      if (intOne > intTwo) {
        return 0;
      } else {
        return 1;
      }
}

greaterThan(10,20);
```

Bytecode

```
0: iload_1
1: iload_2
2: nop
3: if_icmple 8
6: iconst_0
7: ireturn
8: iconst_1
9: ireturn
```

Static check

```
public int greaterThan(int intOne, intTwo) {
      if (intOne > intTwo) {
         return 0;
      } else {
         return 1;
      }
}

greaterThan(10,20);
```

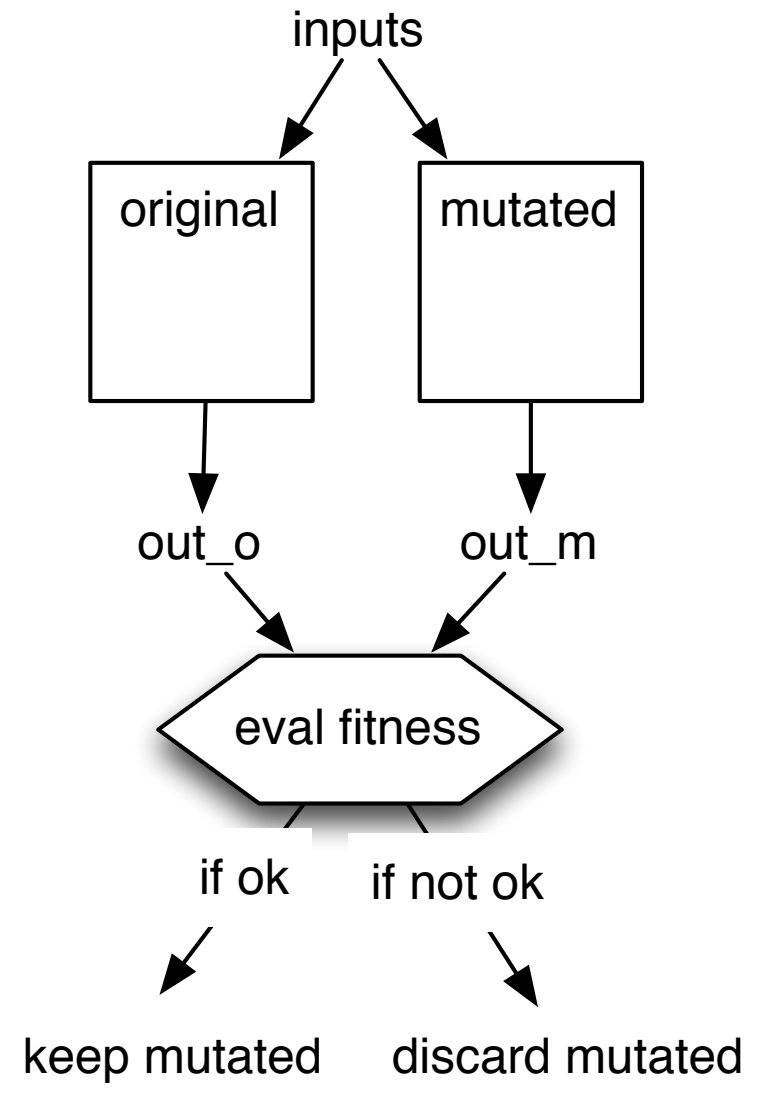
Bytecode

```
0: iload_1
1: iload_2
2: if_icmple 7
5: iconst_0
6: ireturn
7: iconst_1
8: ireturn
```

ECSELR detects the mistake:
it knows that `if_icmple` requires two operands

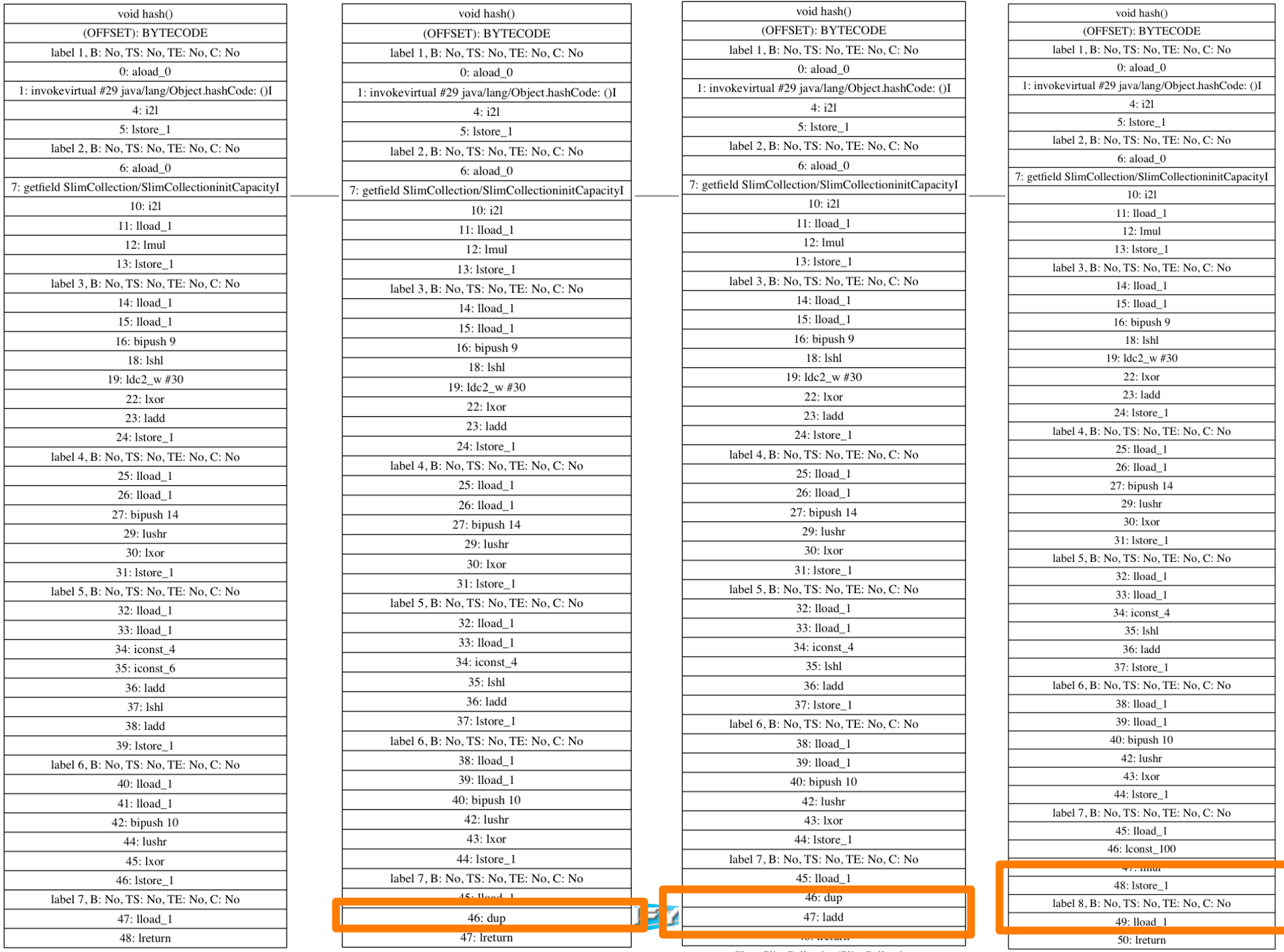
Dynamic checks

- Run both versions in parallel
  - for a given period
- This requires our patched JVM



Diversification example

```
int hash(final Object key) {
    int h = key.hashCode();
    h += ~(h << 9);
    h ^=  h >>> 14;
    h +=  h << 4;
    h ^=  h >>> 10;
    return h;
}
```


void hash()
(OFFSET): BYTECODE
label 1, B: No, TS: No, TE: No, C: No
0: aload_0
1: invokevirtual #29 java/lang/Object.hashCode: ()I
4: i2l
5: lstore_1
label 2, B: No, TS: No, TE: No, C: No
6: aload_0
7: getfield SlimCollection/SlimCollectioninitCapacityI
10: i2l
11: lload_1
12: lmul
13: lstore_1
label 3, B: No, TS: No, TE: No, C: No
14: lload_1
15: lload_1
16: bipush 9
18: lshl
19: ldc2_w #30
22: lxor
23: ladd
24: lstore_1
label 4, B: No, TS: No, TE: No, C: No
25: lload_1
26: lload_1
27: bipush 14
29: lushr
30: lxor
31: lstore_1
label 5, B: No, TS: No, TE: No, C: No
32: lload_1
33: lload_1
34: iconst_4
35: iconst_6
36: ladd
37: lshl
38: ladd
39: lstore_1
label 6, B: No, TS: No, TE: No, C: No
40: lload_1
41: lload_1
42: bipush 10
44: lushr
45: lxor
46: lstore_1
label 7, B: No, TS: No, TE: No, C: No
47: lload_1
48: lreturn
Class SlimCollection/SlimCollection
void hash()
(OFFSET): BYTECODE
label 1, B: No, TS: No, TE: No, C: No
0: aload_0
1: invokevirtual #29 java/lang/Object.hashCode: ()I
4: i2l
5: lstore_1
label 2, B: No, TS: No, TE: No, C: No
6: aload_0
7: getfield SlimCollection/SlimCollectioninitCapacityI
10: i2l
11: lload_1
12: lmul
13: lstore_1
label 3, B: No, TS: No, TE: No, C: No
14: lload_1
15: lload_1
16: bipush 9
18: lshl
19: ldc2_w #30
22: lxor
23: ladd
24: lstore_1
label 4, B: No, TS: No, TE: No, C: No
25: lload_1
26: lload_1
27: bipush 14
29: lushr
30: lxor
31: lstore_1
label 5, B: No, TS: No, TE: No, C: No
32: lload_1
33: lload_1
34: iconst_4
35: lshl
36: ladd
37: lstore_1
label 6, B: No, TS: No, TE: No, C: No
38: lload_1
39: lload_1
40: bipush 10
42: lushr
43: lxor
44: lstore_1
label 7, B: No, TS: No, TE: No, C: No
46: dup
47: lreturn
Class SlimCollection/SlimCollection
void hash()
(OFFSET): BYTECODE
label 1, B: No, TS: No, TE: No, C: No
0: aload_0
1: invokevirtual #29 java/lang/Object.hashCode: ()I
4: i2l
5: lstore_1
label 2, B: No, TS: No, TE: No, C: No
6: aload_0
7: getfield SlimCollection/SlimCollectioninitCapacityI
10: i2l
11: lload_1
12: lmul
13: lstore_1
label 3, B: No, TS: No, TE: No, C: No
14: lload_1
15: lload_1
16: bipush 9
18: lshl
19: ldc2_w #30
22: lxor
23: ladd
24: lstore_1
label 4, B: No, TS: No, TE: No, C: No
25: lload_1
26: lload_1
27: bipush 14
29: lushr
30: lxor
31: lstore_1
label 5, B: No, TS: No, TE: No, C: No
32: lload_1
33: lload_1
34: iconst_4
35: lshl
36: ladd
37: lstore_1
label 6, B: No, TS: No, TE: No, C: No
38: lload_1
39: lload_1
40: bipush 10
42: lushr
43: lxor
44: lstore_1
label 7, B: No, TS: No, TE: No, C: No
45: lload_1
46: dup
47: ladd
Class SlimCollection/SlimCollection
void hash()
(OFFSET): BYTECODE
label 1, B: No, TS: No, TE: No, C: No
0: aload_0
1: invokevirtual #29 java/lang/Object.hashCode: ()I
4: i2l
5: lstore_1
label 2, B: No, TS: No, TE: No, C: No
6: aload_0
7: getfield SlimCollection/SlimCollectioninitCapacityI
10: i2l
11: lload_1
12: lmul
13: lstore_1
label 3, B: No, TS: No, TE: No, C: No
14: lload_1
15: lload_1
16: bipush 9
18: lshl
19: ldc2_w #30
22: lxor
23: ladd
24: lstore_1
label 4, B: No, TS: No, TE: No, C: No
25: lload_1
26: lload_1
27: bipush 14
29: lushr
30: lxor
31: lstore_1
label 5, B: No, TS: No, TE: No, C: No
32: lload_1
33: lload_1
34: iconst_4
35: lshl
36: ladd
37: lstore_1
label 6, B: No, TS: No, TE: No, C: No
38: lload_1
39: lload_1
40: bipush 10
42: lushr
43: lxor
44: lstore_1
label 7, B: No, TS: No, TE: No, C: No
45: lload_1
46: lconst_100
48: lstore_1
label 8, B: No, TS: No, TE: No, C: No
49: lload_1
50: lreturn
Class SlimCollection/SlimCollection

Diversification example

```java
public long hash() {
        long h = this.hashCode();
        h = this.initCapacity * h;
        h += ~(h << 9);
        h ^=  h >>> 14;
        h +=  h << 4;
        h ^=  h >>> 10;
        return h;
    }
```

```java
public long hash() {
        long h = this.hashCode();
        h = this.initCapacity * h;
        h += ~(h << 9);
        h ^=  h >>> 14;
        h +=  h << 4;
        h ^=  h >>> 10;
        return h + h;
    }
```

```java
public long hash() {
        long h = this.hashCode();
        h = this.initCapacity * h;
        h += ~(h << 9);
        h ^=  h >>> 14;
        h +=  h << (4+6);
        h ^=  h >>> 10;
        return h;
    }
```

```java
public long hash() {
        long h = this.hashCode();
        h = this.initCapacity * h;
        h += ~(h << 9);
        h ^=  h >>> 14;
        h +=  h << (4+6);
        h ^=  h >>> 10;
        h = h * 100;
        return h;
    }
```

Conclusion

- We now have a machine to evolve Java programs at runtime
- Next step: use it for runtime GI

- https://bitbucket.org/Kwaku/agentd/src