
ORACLE®

Specialisation for Allocation Removal and Constant Folding

MMNet 2015, Glasgow

Chris Seaton
Research Manager
Oracle Labs

chris.seaton@oracle.com

Safe Harbor Statement

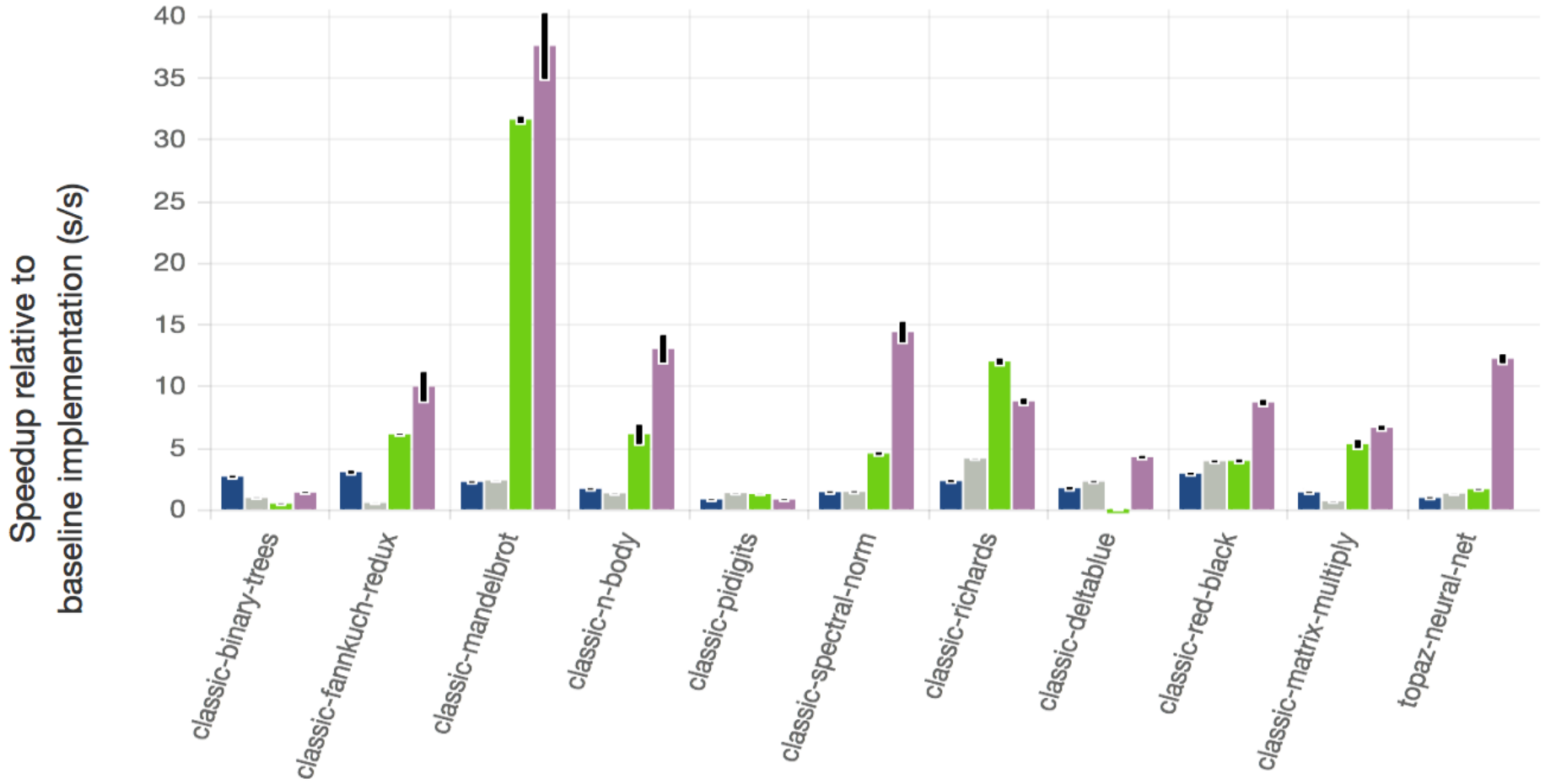
The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Overview

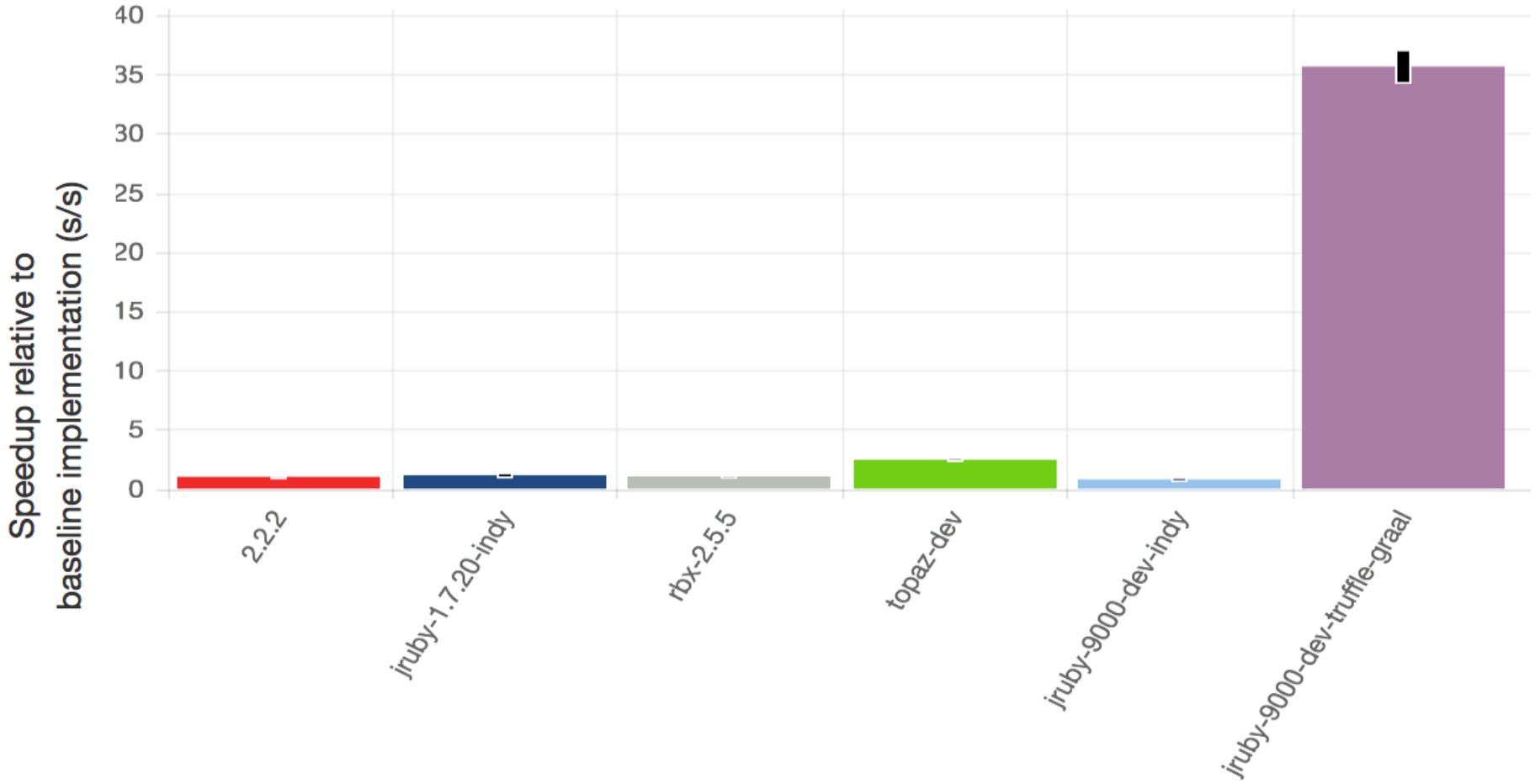
- Assumption: best way to manage memory is to not allocate it on the heap
  - Remove load on the GC
  - Reduce pauses, improve locality, generally improve performance
- Techniques available to do this
  - Escape analysis, partial escape analysis, scalar replacement, stack allocation…
  - They are powerful, but only work for some program structures
- Research problem – how to structure an interpreter so that the existing techniques are more effective
- Our solution – AST specialisation for small data structures

JRuby+Truffle

Shootout benchmarks



Production benchmarks



93%

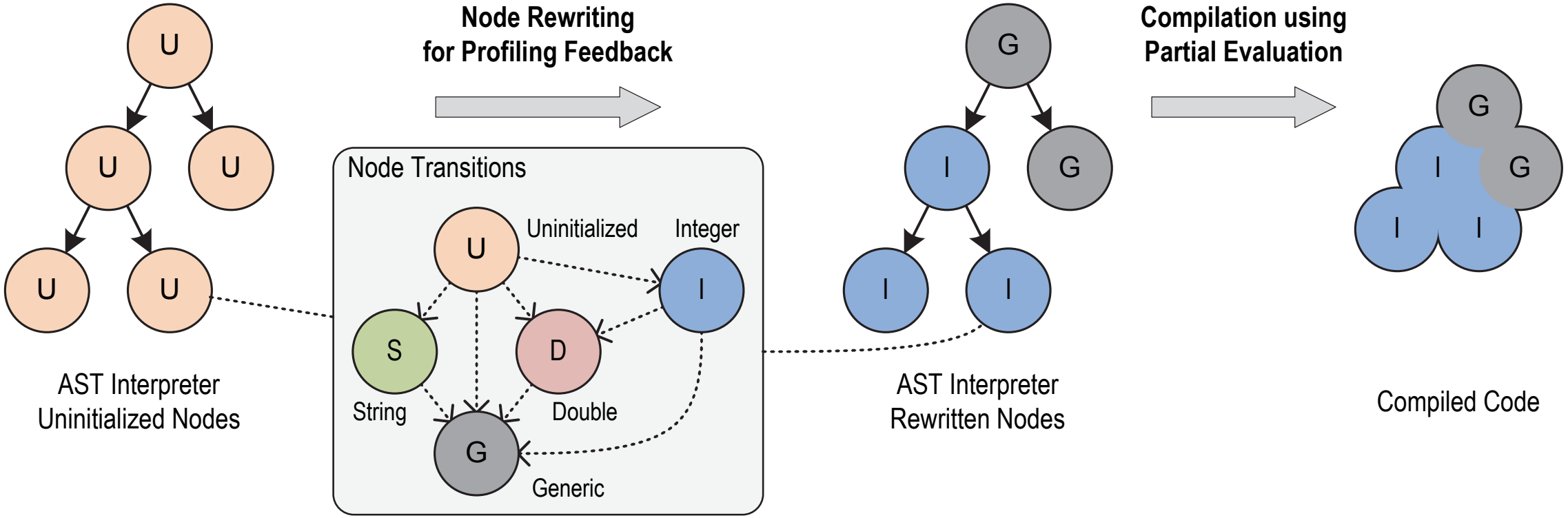
Language specs

89%

Core library specs

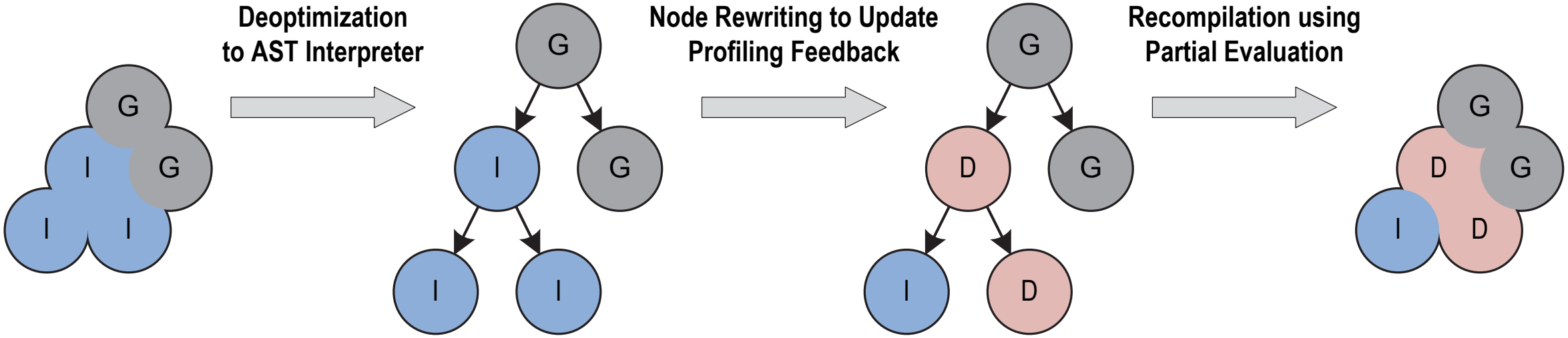
Truffle, Graal and the GraalVM

Truffle Dynamic Optimisation Model



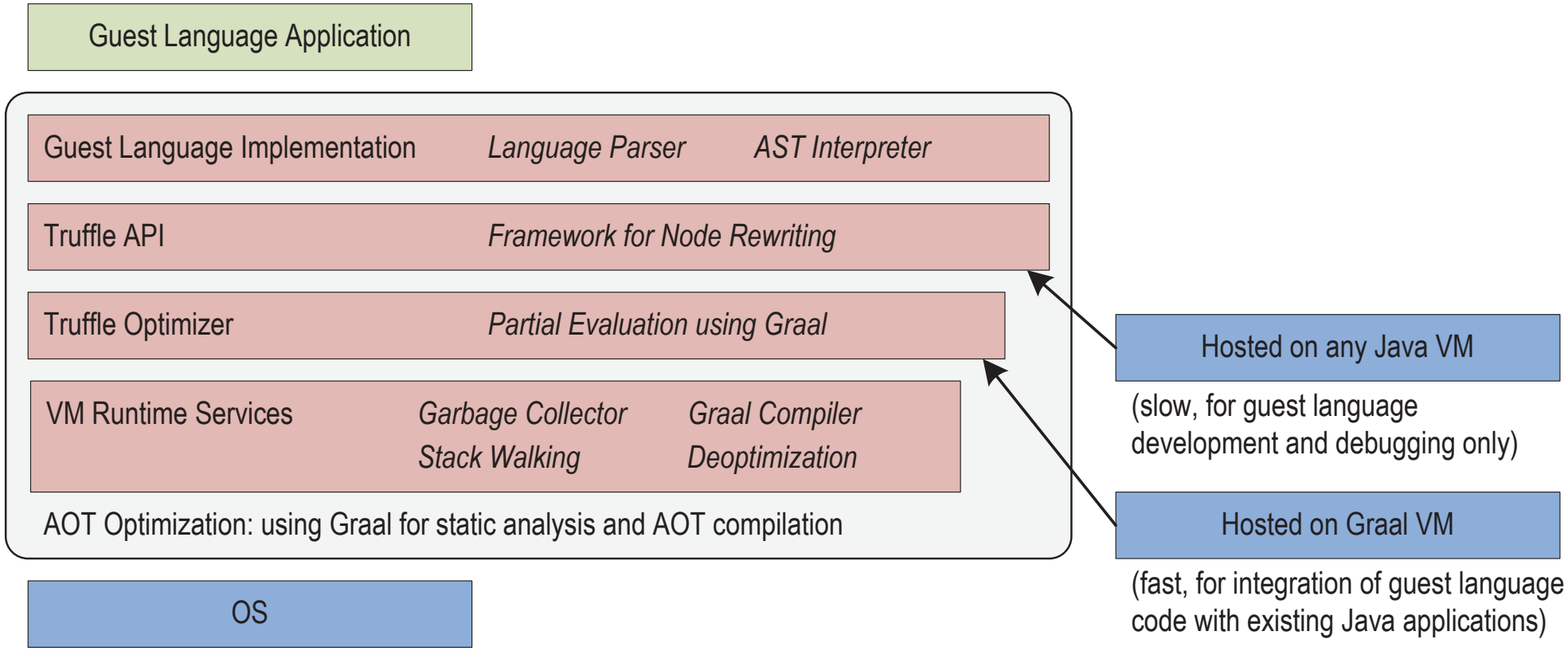
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko, “One VM to rule them all,” presented at the Onward! '13: Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software, New York, New York, USA, 2013, pp. 187–204.

Truffle Dynamic Deoptimisation Model



T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko, “One VM to rule them all,” presented at the Onward! '13: Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software, New York, New York, USA, 2013, pp. 187–204.

GraalVM Structure



T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko, “One VM to rule them all,” presented at the Onward! '13: Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software, New York, New York, USA, 2013, pp. 187–204.

Patterns of Logical Allocation in Ruby Code

```ruby
def clamp(num, min, max)
  [min, num, max].sort[1]
end
```

```
VALUE psd_native_util_clamp(VALUE self, VALUE r_num, VALUE r_min, VALUE r_max) {
  int num = FIX2INT(r_num);
  int min = FIX2INT(r_min);
  int max = FIX2INT(r_max);

  return num > max ? r_max : (num < min ? r_min : r_num);
}
```

AST Specialisation for Small Data Structures

Escape analysis

- Answers the question: is this object accessible to anyone outside this compilation unit?
  - ‘Compilation unit’ could be method, loop body, control flow trace
  - Likely include inlining of at least methods called on the object
- Algorithms such as Choi et al 1999
  - Applied in C2

J.-D. Choi, M. Gupta, M. Serrano, V. C. Sreedhar, and S. Midkiff, “Escape analysis for Java,” presented at the OOPSLA '99: Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, New York, New York, USA, 1999, vol. 34, no. 10, pp. 1–19.

What to do with escape analysis results?

- If an object doesn't escape, it doesn't need to be visible to anyone else
  - Allocate explicitly on the stack (like alloca instead of malloc)
  - Scalar replacement (represent fields in the object as local variables)
  - Turn the fields into dataflow edges in the IR
- No need to involve the GC
- Enables other optimisations
  - Partial evaluation
  - Constant folding

Partial escape analysis

- Weakens the requirement for ‘not accessible outside the compilation unit’ to ‘not accessible on a subset of control flow paths’
  - Common / uncommon paths
  - Fast path / slow path
- Algorithms such as Stadler et al 2014
  - Applied in the Graal dynamic compiler

L. Stadler, T. Würthinger, and H. Mössenböck, “Partial Escape Analysis and Scalar Replacement for Java,” presented at the Proceedings of the Symposium on Code Generation and Optimization (CGO), 2014.

Limitations in practice

- It doesn’t take much to cause an object to escape
- Limited ability to inline
- Unbounded loops
- Unbounded recursion
- Standard library methods

```java
private IRubyObject sortInternal(final ThreadContext context, final Block block) {
    IRubyObject[] newValues = new IRubyObject[realLength];
    int length = realLength;

    safeArrayCopy(values, begin, newValues, 0, length);
    Qsort.sort(newValues, 0, length, (o1, o2) -> {
            IRubyObject obj1 = (IRubyObject) o1;
            IRubyObject obj2 = (IRubyObject) o2;
            IRubyObject ret = block.yieldArray(context,
                    getRuntime().newArray(obj1, obj2), null);
            //TODO: ary_sort_check should be done here
            return RubyComparable.cmpint(context, ret, obj1, obj2);
    });

    values = newValues;
    begin = 0;
    realLength = length;
    return this;
}
```

```
@ExplodeLoop
@Specialization(guards = {"isIntArray(array)", "isSmall(array)"})
public RubyBasicObject sortVeryShortIntegerFixnum(
        VirtualFrame frame, RubyBasicObject array, NotProvided block) {
    final int[] store = (int[]) getStore(array);
    final int[] newStore = new int[store.length];

    final int size = getSize(array);

    for (int i = 0; i < ARRAYS_SMALL; i++) {
        if (i < size) {
            for (int j = i + 1; j < ARRAYS_SMALL; j++) {
                if (j < size) {
                    if (castSortValue(compareDispatchNode
                            .call(frame, store[j], "<=>", null, store[i])) < 0) {
                        final int temp = store[j];
                        store[j] = store[i];
                        store[i] = temp;
                    }
                }
            }
            newStore[i] = store[i];
        }
    }

    return createArray(newStore, size);
}
```

```
@ExplodeLoop
@Specialization(guards = {"isIntArray(array)", "isSmall(array)"})
public RubyBasicObject sortVeryShortIntegerFixnum(
```

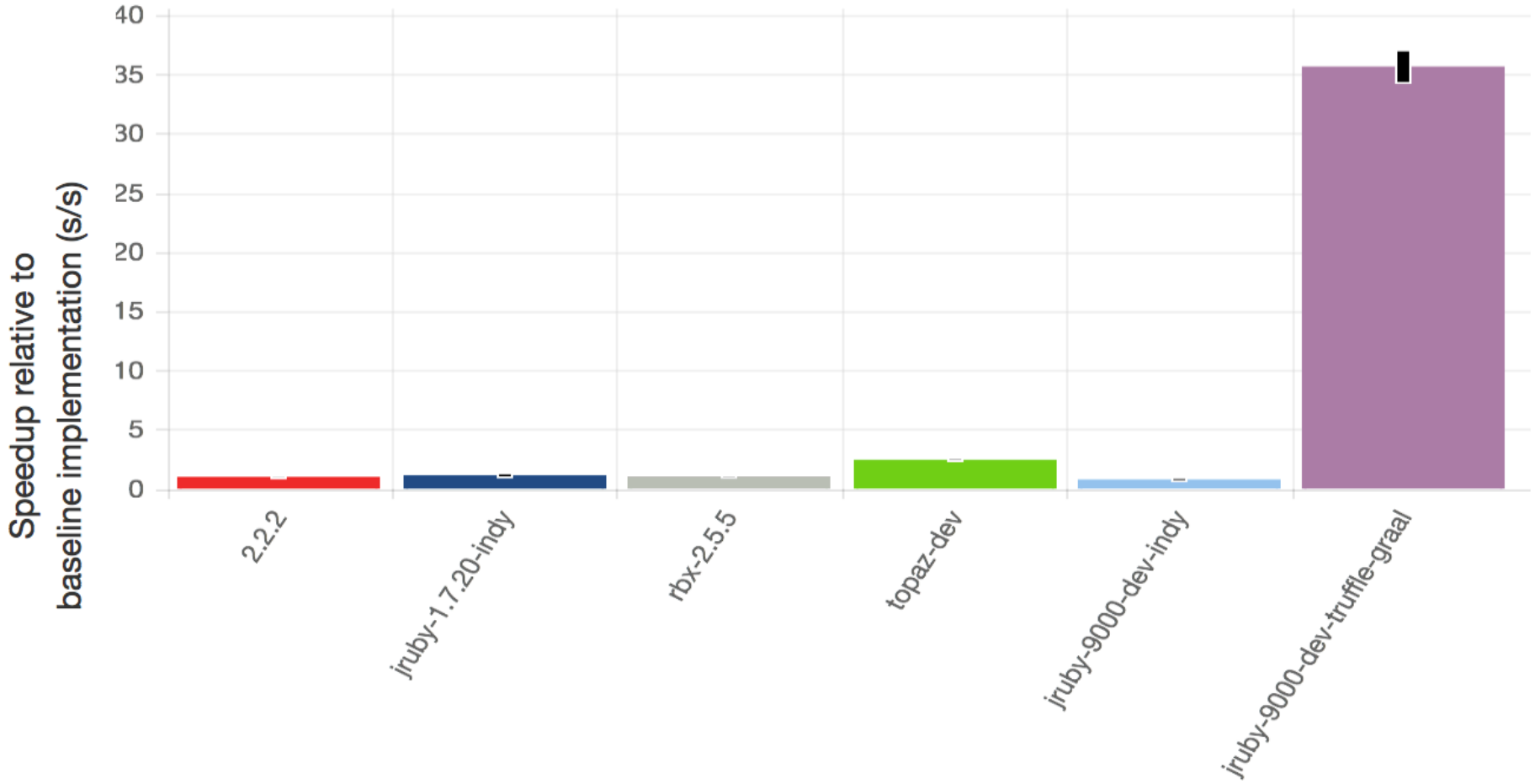
```
final int size = getSize(array);

for (int i = 0; i < ARRAYS_SMALL; i++) {
    if (i < size) {
        for (int j = i + 1; j < ARRAYS_SMALL; j++) {
            if (j < size) {
                if (castSortValue(compareDispatchNode
                        .call(frame, store[j], "<=>", null, store[i])) < 0) {
                    final int temp = store[j];
                    store[j] = store[i];
                    store[i] = temp;
                }
            }
        }
        newStore[i] = store[i];
    }
}
```

Impact on the Bottom-Line

```ruby
loop do
  start = Time.now
  100_000_000.times do
    [3, 1, 2].sort[0]
  end
  puts Time.now - start
end
```

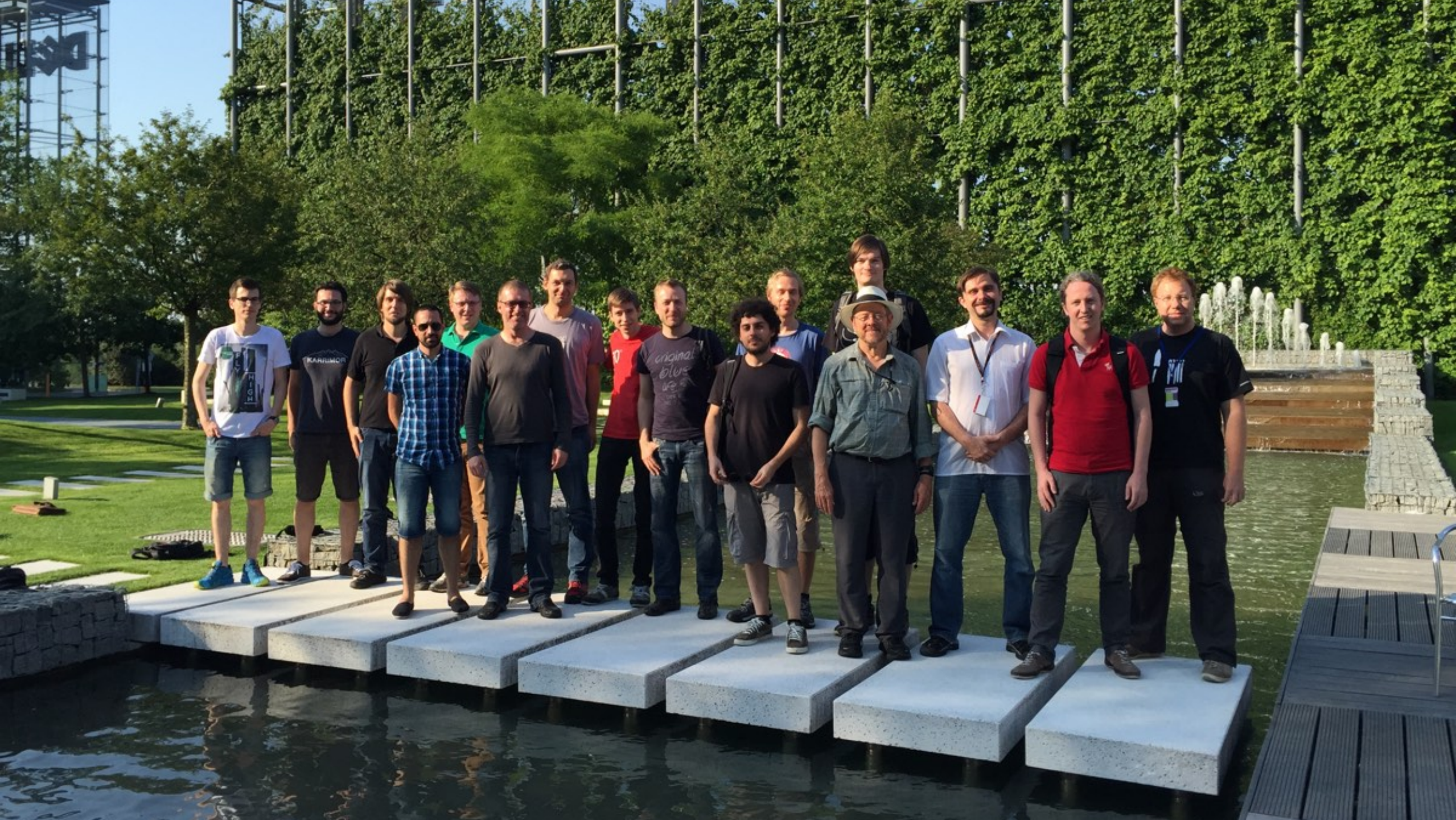
	With specialisation	Without specialisation	Conventional JRuby
Time per 100mm iteration	0.06 s	1.8 s	14 s
Allocations per iteration	5 KB/s	1.6 MB/s	1.2 GB/s

Production benchmarks



Conclusions

- The JVM already has the ability to remove allocations
- The GraalVM is more powerful still, with partial escape analysis
- But these techniques can’t always be applied – tends to fall down pretty quickly



Safe Harbor Statement

The preceding is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Hardware and Software
Engineered to Work Together


ORACLE®