ORACLE

Software Language Engineering at Oracle Labs

Guido Wachsmuth, Martijn Dwars
Researchers at Oracle Labs Zurich

Safe Harbor Statement

The following is intended to outline our general product direction. 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. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Agenda

1. Meet Oracle Labs
2. Software Language Engineering for Parallel Graph Analytics
3. Graal, Truffle, and Active Libraries
4. Internships at Oracle Labs

   Hangout

ORACLE

Meet Oracle Labs

ORACLE

Identify, **explore**, and **transfer** new technologies that have the potential to substantially improve Oracle's business.

Oracle Labs mission

Oracle Labs

Look for novel approaches and methodologies

- projects with high risk or uncertainty
- projects difficult to tackle within product organization

Focus on real-world outcomes

- examples: chip multi-threading, Java programming languages

Global research team

- 220+ researchers
- Zurich: 80+ researchers

Exploratory research

- new ideas within domains relevant to Oracle

Directed research

- difficult, future-looking problems
- driven by product requirements
- in collaboration with product teams

Consulting

- provide expertise to product organizations

Product incubation

- grow new products from Oracle Labs research

Oracle Labs Zurich

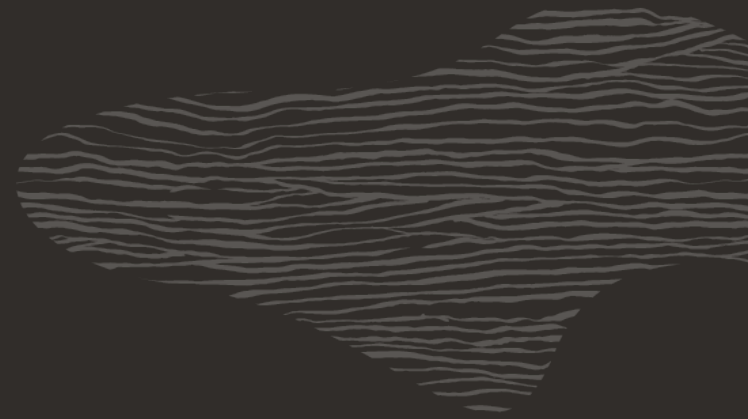
Guido Wachsmuth

- Principal Researcher at Oracle Labs Zurich
- Former Assistant Professor in PL group
- Started at Oracle in 2016
- Working on Green-Marl DSL compilers
- Recruiting talented students for internships

Martijn Dwars

- Senior Researcher at Oracle Labs Zurich
- MSc Computer Science from TU Delft
- Graduated from the PL group
- Started at Oracle as an intern in 2017
- Worked on Green-Marl DSL (2018-2019)
- Working on Active Libraries (2019)

ORACLE



Software Language Engineering for Parallel Graph Analytics

Our mission is to help people
see data in new ways, discover insights,
unlock endless possibilities.

Your Data is a Graph!

Represent it as a **property graph**

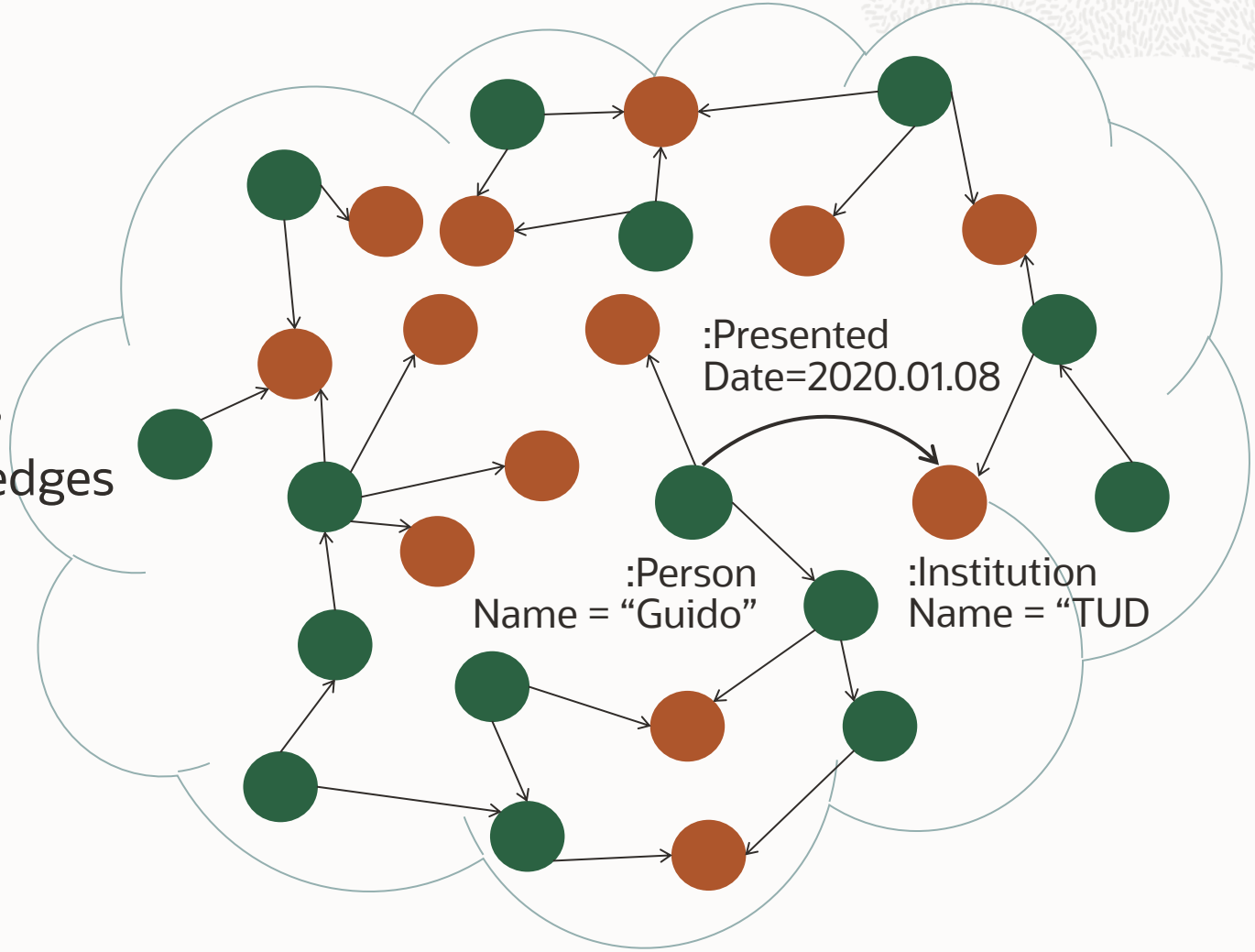
- Entities are **vertices**
- Relationships are **edges**

Annotate your graph

- **Labels** identify vertices and edges
- **Properties** describe vertices and edges

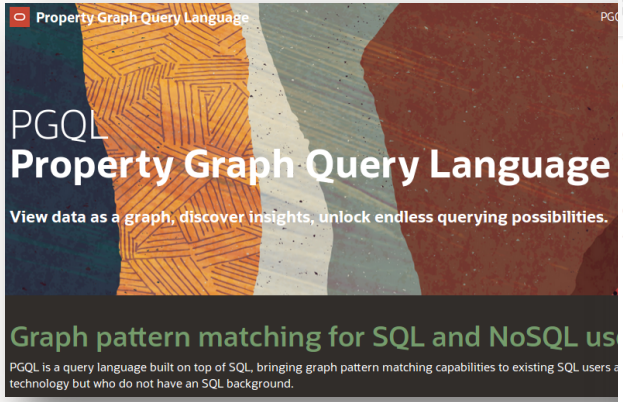
For the purpose of

- Data modeling
- Data analysis

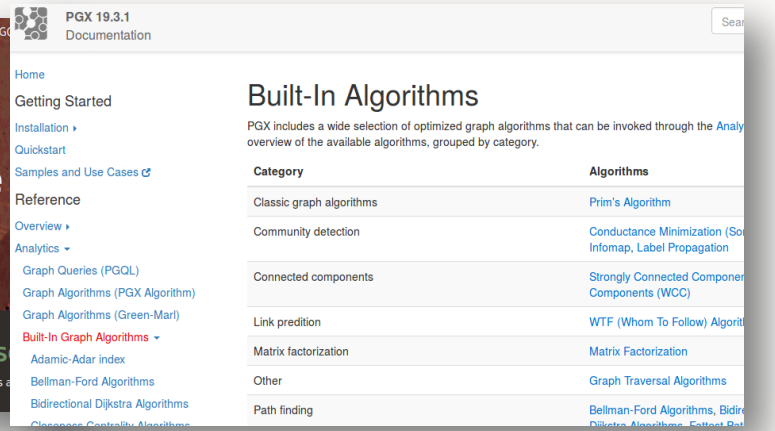


Oracle Labs PGX – Parallel Graph Analytix

- A fast, parallel, in-memory graph processing framework
- Efficient graph analytics & queries
  - 35+ built-in graph algorithms
- Embedded in Oracle products; active research project
- Available to try at Oracle Technology Network



http://pgql-lang.org/

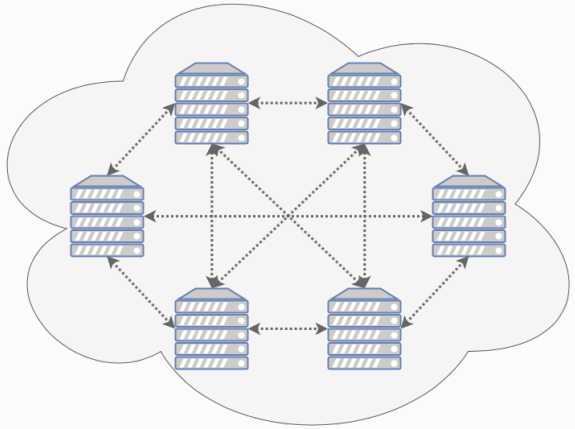


https://www.oracle.com/middleware/technologies/parallel-graph-analytix.html

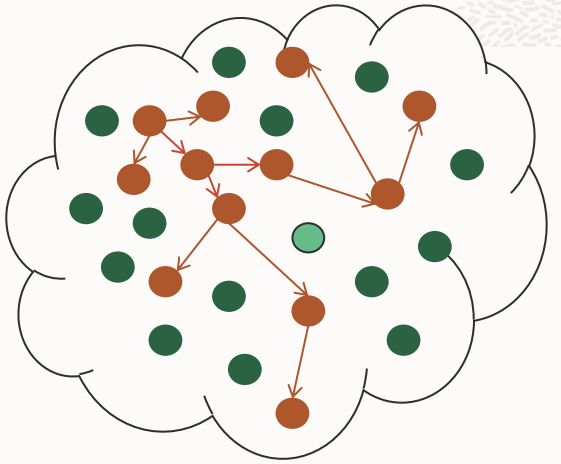
Single machine & distributed

PGX.SM
Java based

PGX.D
Scalable, cloud ready
C++ based



Main Approaches of Graph Processing



$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

Graph querying and pattern matching

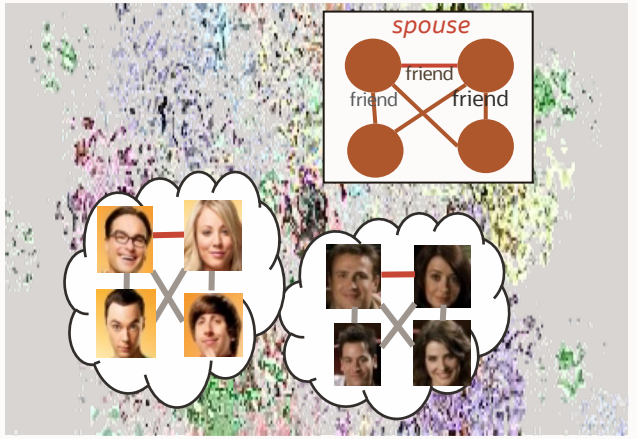
- Query the graph to find sub-graphs that match to the given relationship pattern

Computational graph analytics

- Iterate the graph multiple times and compute mathematical properties, e.g. PageRank

Graph-based machine learning

- Use the structural information latent in graphs, e.g. graph similarity



PGQL – Property Graph Query Language

```
SELECT p.name, COUNT(*) AS num_movies
FROM movies_graph
MATCH
  (p:Person) -[:Directed]-> (m:Movie),
  (p) -[:Played_in]-> (m:Movie)
  /* same person, same movie */
GROUP BY p
ORDER BY num_movies DESC
LIMIT 5
```

```
+-----------------------------------+
| p.name              | num_movies  |
+-----------------------------------+
| Clint Eastwood      | 10          |
| Woody Allen         | 9           |
| Michael Moore       | 5           |
| David Hewlett       | 4           |
| Jay Chandrasekhar   | 3           |
+-----------------------------------+
```

Algorithmic Graph Processing

PageRank

WCC

Bellman-Ford

PGX

shared-memory engine

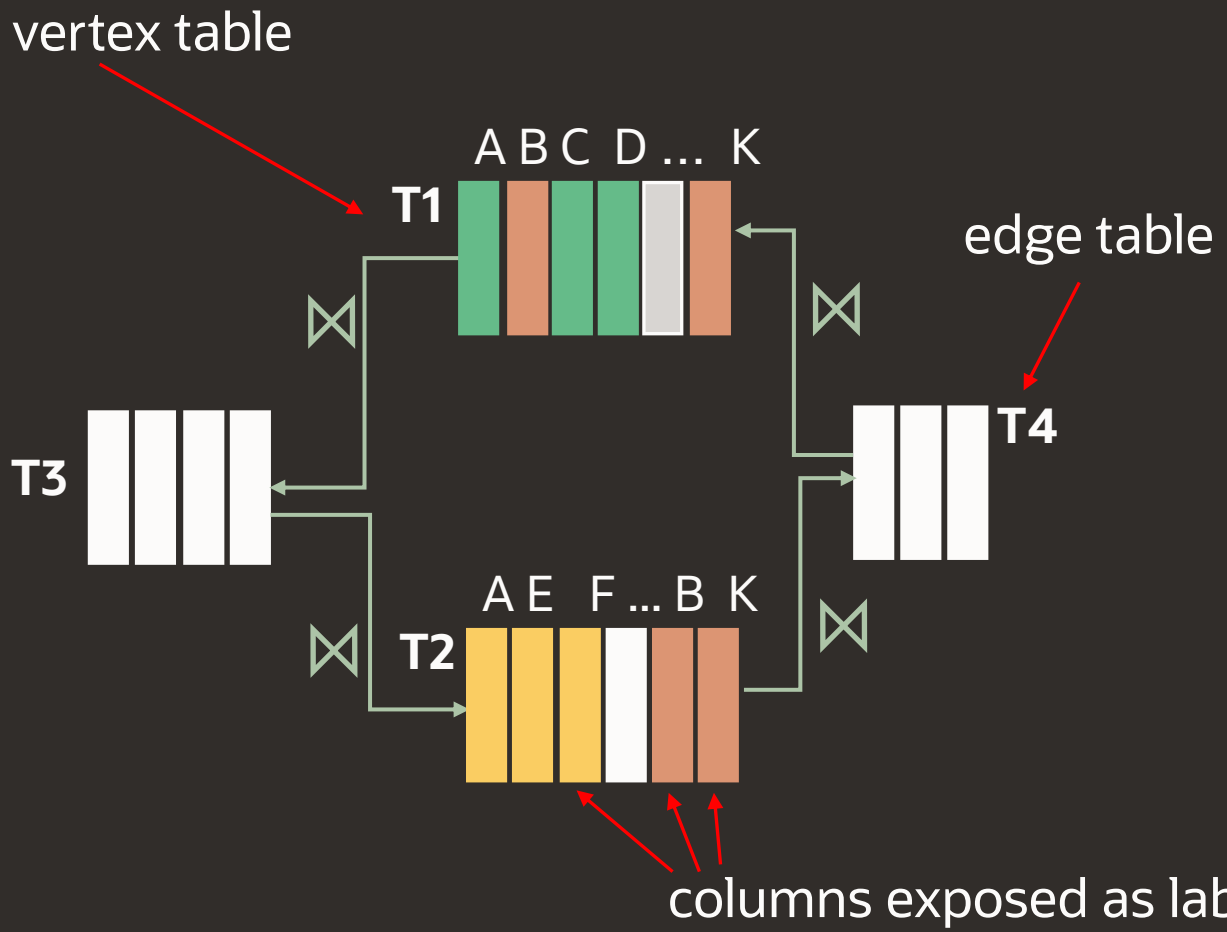
distributed engine

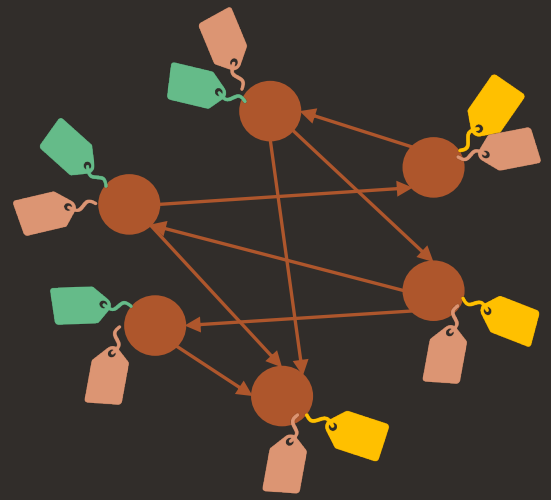
Oracle DB

SQL engine

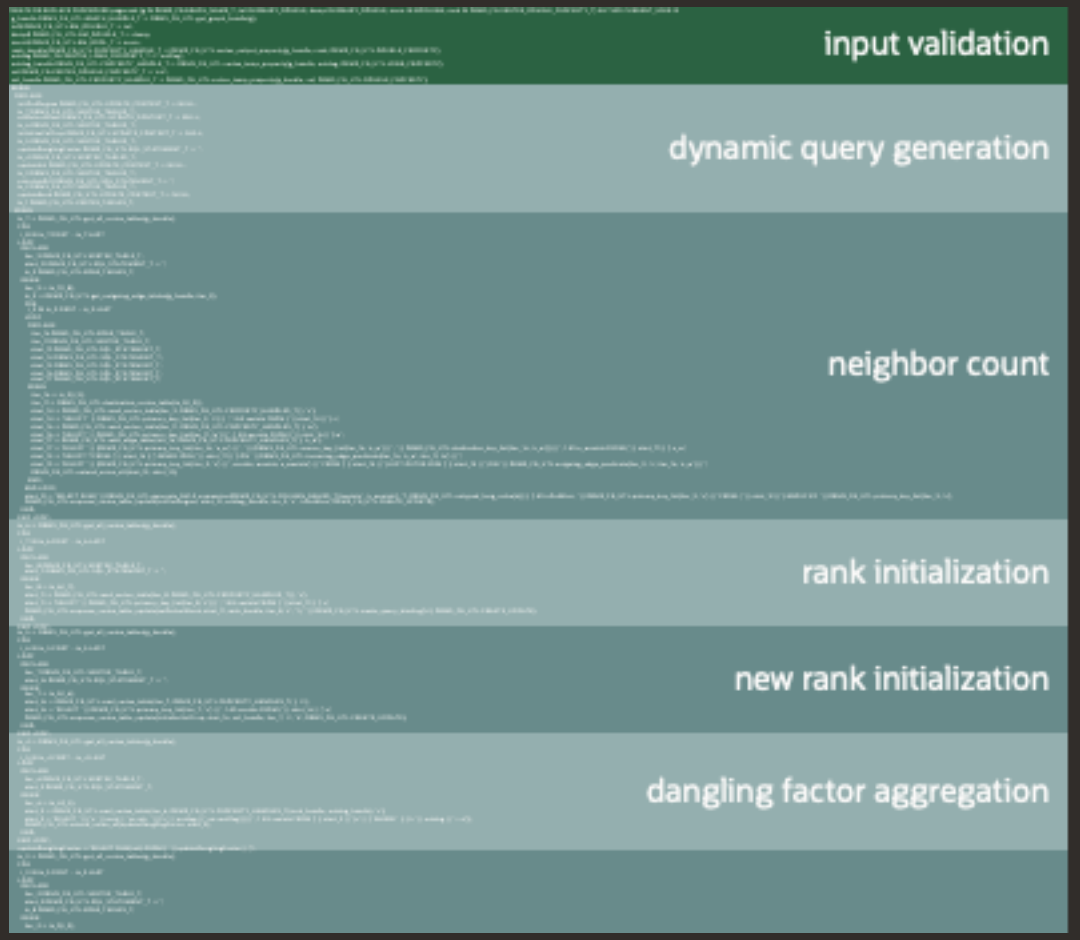
in-memory engine

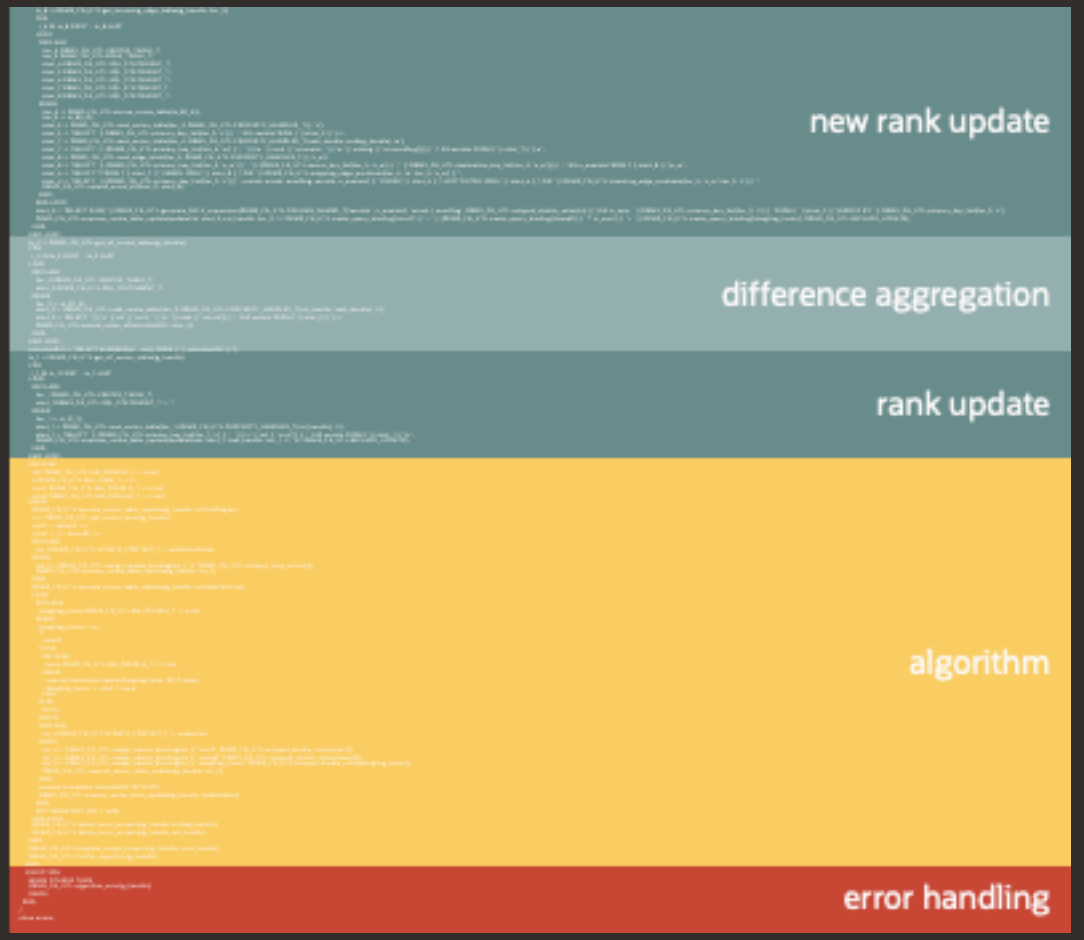
Property Graph Data Model Explained



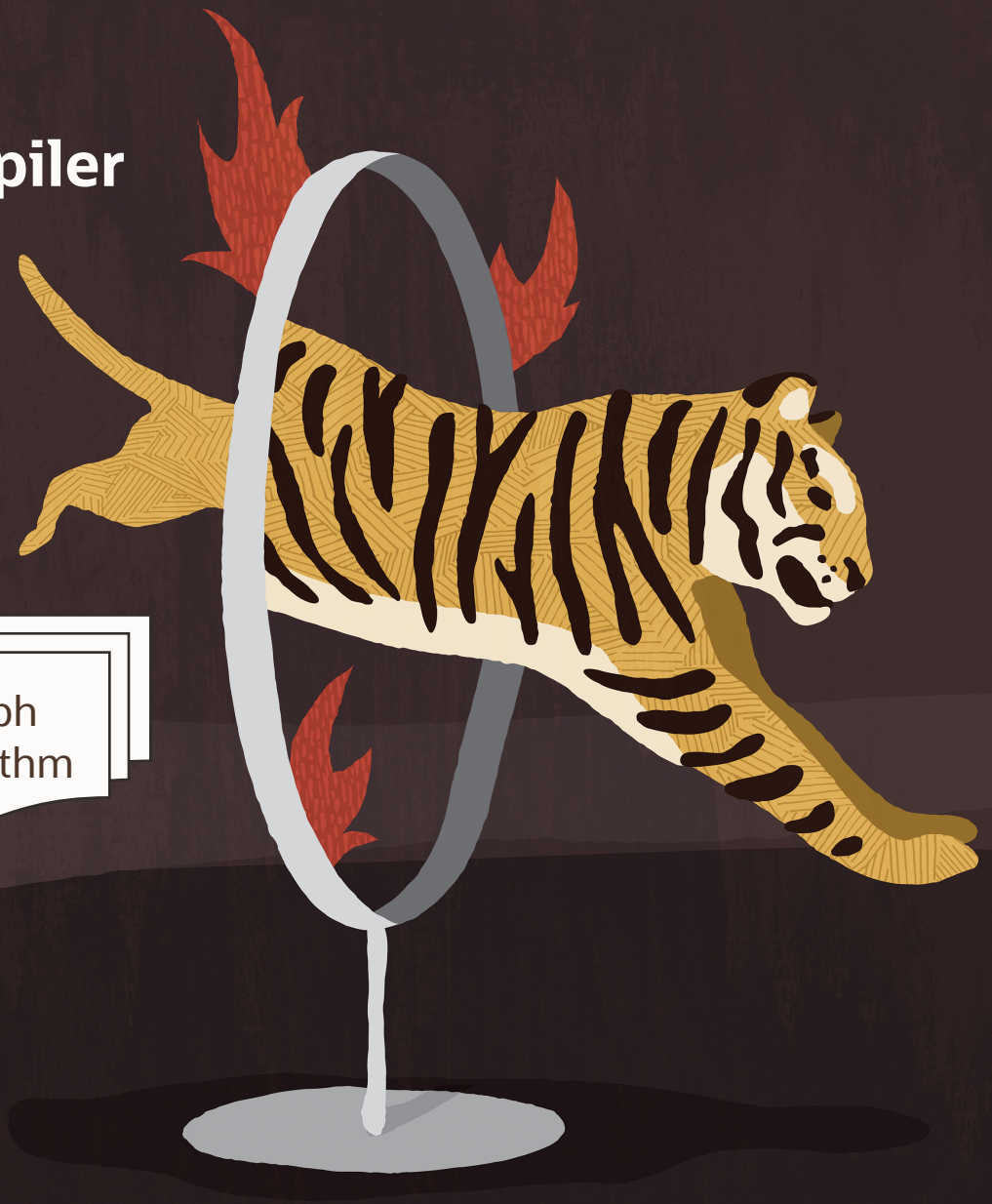


PageRank in PL/SQL





Compiler



graph algorithm

PGX

- shared-memory engine
- distributed engine

Oracle DB

- SQL engine
- in-memory engine

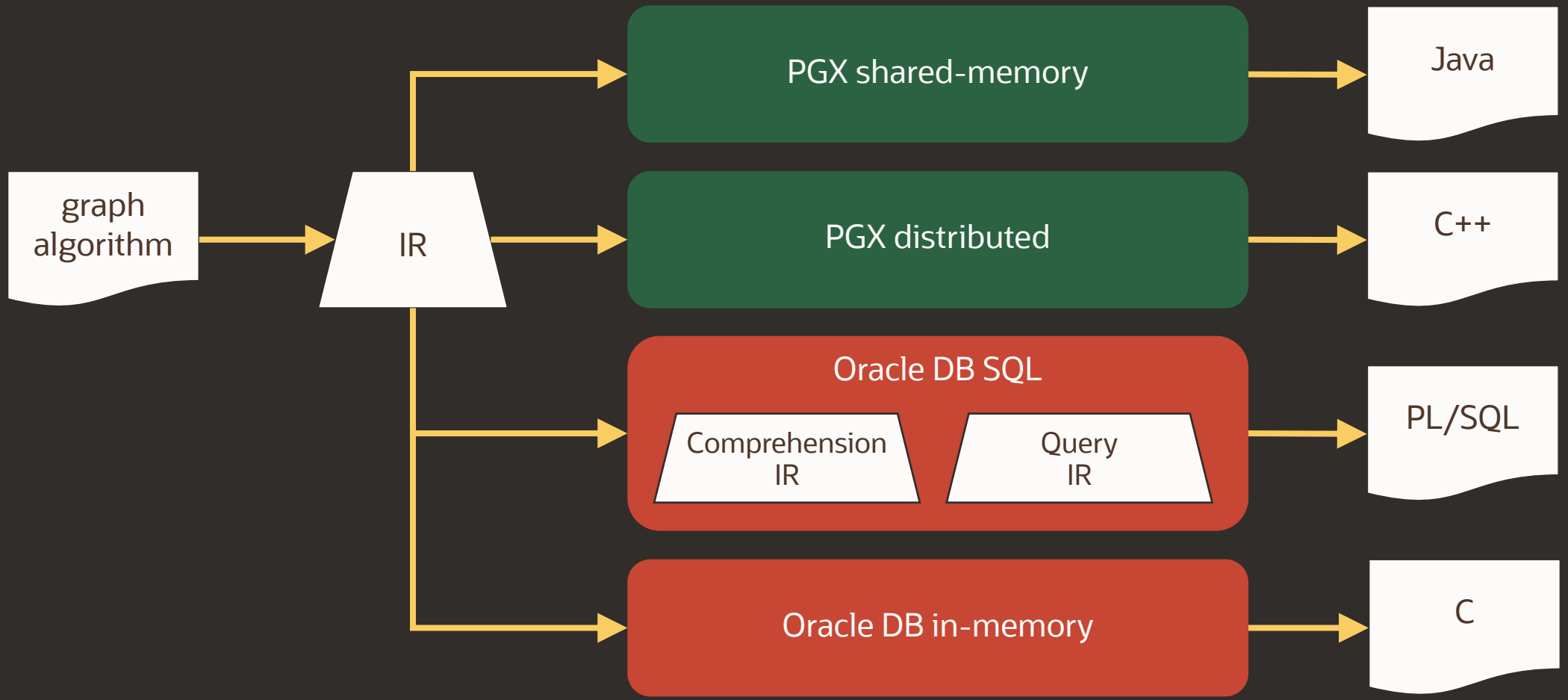
PageRank in Green-Marl

```
procedure pagerank(graph G, double tol, double damp, bool norm; nodeProp<double> rank) {

  double N = G.numNodes();
  G.rank = 1 / N;
  double diff;
  double dangling_factor = 0;
  do {
    diff = 0.0;
    if (norm) {
      dangling_factor = damp / N * sum(v: G.nodes) (v.outDegree() == 0) { v.rank };
    }

    foreach (t: G.nodes) {
      double in_sum = sum(w: t.inNbrs) {w.rank / w.outDegree()};
      double val = (1 - damp) / N + damp * in_sum + dangling_factor;
      diff += | val - t.rank |;
      t.rank <= val;
    }
  } while (diff > tol);
}
```

PageRank in PGX Algorithm

```
@GraphAlgorithm public class Pagerank {
  public void pagerank(PgxGraph G, double tol, double damp, boolean norm, @Out VertexProperty<Double> rank) {
    double N = G.getNumVertices();
    rank.setAll(1 / N);
    Scalar<Double> diff = Scalar.create();
    Scalar<Double> danglingFactor = Scalar.create(0d);
    do {
      diff.set(0d);
      if (norm) {
        danglingFactor.set(damp / N * G.getVertices().filter(v -> v.getOutDegree() == 0).sum(rank::get)));
      }
      G.getVertices().forEach(t -> {
        double in_sum = t.getInNeighbors().sum(w -> rank.get(w) / w.getOutDegree());
        double val = (1 - damp) / N + damp * in_sum + danglingFactor.get();
        diff.reduceAdd(Math.abs(val - rank.get(t)));
        rank.setDeferred(t, val);
      });
    } while (diff.get() > tol);
  }
}
```

Compilation Pipelines



Challenges for Language Workbenches

usage modes

- ~~DSL editor~~
- command line
- product integration

language composition

- open-source frontends
- proprietary backends

language build

- build system integration
- build times

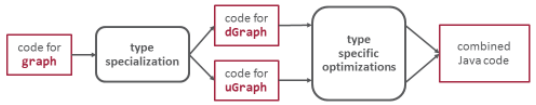
3rd party runtime libraries

- vulnerabilities
- legal
- memory footprint
- start-up time

Opportunities: Innovation & Competiveness

Directed & Undirected Graph Support

- directed and undirected graph as explicit types in Green-Marl
- specify for which type of graph an algorithm is defined
- `dGraph:` directed graphs only
  `uGraph:` undirected graphs only
  `graph:` both types
- compiler can generate specialized code for each type



Undirected Traversal

- more control over the traversal direction in BFS & DFS
- traverse the graph in undirected way

```
inBFS (n: G.nodes from root using outEdges) {
  // traverse outgoing edges (default)
}

inBFS (n: G.nodes from root using inEdges) {
  // traverse incoming edges
}

inBFS (n: G.nodes from root using inOutEdges) {
  // traverse undirected (outgoing & incoming) edges
}
```

Ordered Iteration

- iterate over ranges in an ordered way
- only allowed for sequential iterations

```
for (n: G.nodes order by n.prop) {
  …
}

for (n: nSet.items order by 2 * n.prop desc) {
  …
}

for (n: G.nodes order by uniform()) {
  …
}
```

Local Procedures

- helper procedures that are visible only inside the same compilation unit
- allows more modular and cleaner code

```
proc random_walk(graph G, int length; nodeProp<string> walks) {
  foreach (n: G.nodes)
    n.walks = create_walk(G, n, length);
}

local create_walk(graph G, node n, int length) : string {
  string walk = ""; int i = 0;
  node current = n;
  while (i++ < length) {
    walk += current + " ";
    current = current.pickRandomNbr();
  }
  return walk;
}
```

Experimental: Graph Schemas

- define a local graph schema with properties
- use the graph schema as type in procedures
- specify properties once - use them everywhere
- prevents explosion of argument lists with local procedures

```
schema graph myGraph {
  nodeProp<int> size,
  mutable edgeProp<double> weight
}

procedure test(myGraph G, node n) {
  println(n.size);
}
```

specify properties once…

… (re)use them everywhere

Graph Mutation

- create new graphs
- add new nodes/edges
- remove nodes/edges

```
graph G;

mutation(G) {
  node n = G.createNode();
  node n2 = n.createNbr();
}

foreach (n: G.nodes) {
  …
}
```

ORACLE

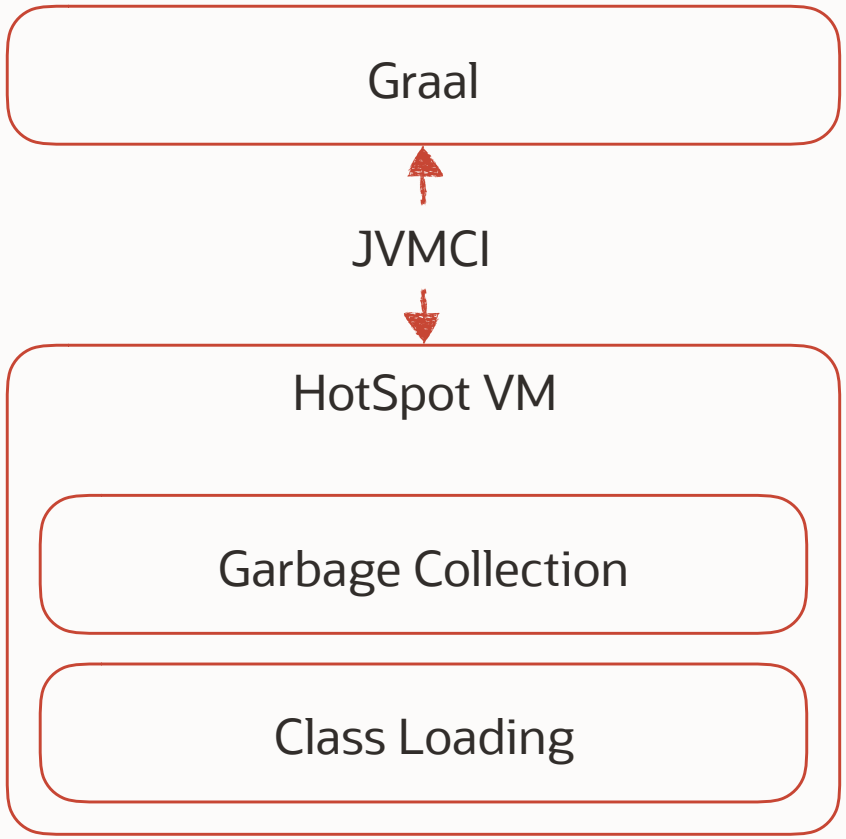
Graal, Truffle, and Active Libraries

Martijn Dwars

Structure

- Graal vs. C2
- Truffle
- Active Libraries: FAD.js, Oracle JDBC
- Internship opportunities

Graal vs. C2




```
interface JVMCICompiler {
    byte[] compileMethod(byte[] bytecode);
}
```

Graal vs. C2

- What is Graal?
  - Graal is an *aggressively optimising* JIT compiler for Java bytecode, written in Java.

Graal vs. C2

- What is Graal?
  - Graal is an *aggressively optimising* JIT compiler for Java bytecode, written in Java.

- How is Graal different from C2?
  - Partial Escape Analysis
  - Improved Inlining
  - Improved Data-Flow Analysis

Graal vs. C2: Partial Escape Analysis

```
class MyObject {
  final int x;

  public MyObject(int x) {
    this.x = x;
  }
}

int x;

@Benchmark
public void single(Blackhole blackhole) {
  for (int i = 0; i < 300; i++) {
    MyObject o = new MyObject(x));

    blackhole.consume(o.x);
  }
}
```

Graal vs. C2: Partial Escape Analysis

```
class MyObject {
  final int x;

  public MyObject(int x) {
    this.x = x;
  }
}

int x;

@Benchmark
public void single(Blackhole blackhole) {
  for (int i = 0; i < 300; i++) {
    MyObject o = new MyObject(x));

    blackhole.consume(o.x);
  }
}
```

- Escape Analysis enables:
  - Stack Allocation
  - Scalar Replacement
  - Lock Elision

Graal vs. C2: Partial Escape Analysis

```
class MyObject {
  final int x;

  public MyObject(int x) {
    this.x = x;
  }
}

int x;

@Benchmark
public void single(Blackhole blackhole) {
  for (int i = 0; i < 300; i++) {
    MyObject o = new MyObject(x));

    blackhole.consume(o.x);
  }
}
```

- Escape Analysis enables:
  - Stack Allocation
  - Scalar Replacement
  - Lock Elision

- **C2**: 683.497 ± 26 ns
- **C2-gc.count:** 0
- **Graal**: 561.988 ± 21 ns
- **Graal-gc.count:** 0

Graal vs. C2: Partial Escape Analysis (cont.)

```
int x;
boolean flag;

@Setup(Level.Iteration)
public void shake() {
  flag = ThreadLocalRandom.current().nextBoolean();
}

@Benchmark
public void split(Blackhole blackhole) {
  for (int i = 0; i < 300; i++) {
    MyObject o;

    if (flag) { o = new MyObject(x); }
    else {      o = new MyObject(x); }

    blackhole.consume(o.x);
  }
}
```

Graal vs. C2: Partial Escape Analysis (cont.)

```
int x;
boolean flag;

@Setup(Level.Iteration)
public void shake() {
  flag = ThreadLocalRandom.current().nextBoolean();
}

@Benchmark
public void split(Blackhole blackhole) {
  for (int i = 0; i < 300; i++) {
    MyObject o;

    if (flag) { o = new MyObject(x); }
    else {      o = new MyObject(x); }

    blackhole.consume(o.x);
  }
}
```

- **C2:** 1217.931 ± 92 ns
- **Graal:** 629.041 ± 13 ns

Graal vs. C2: Partial Escape Analysis (cont.)

```
int x;
boolean flag;

@Setup(Level.Iteration)
public void shake() {
  flag = ThreadLocalRandom.current().nextBoolean();
}

@Benchmark
public void split(Blackhole blackhole) {
  for (int i = 0; i < 300; i++) {
    MyObject o;

    if (flag) { o = new MyObject(x); }
    else {      o = new MyObject(x); }

    blackhole.consume(o.x);
  }
}
```

- **C2:** 1217.931 ± 92 ns
- **Graal:** 629.041 ± 13 ns

- **C2:gc.count:** 84
- **Graal:gc.count:** 0

Graal vs. C2: Partial Escape Analysis (cont.)

```
0xc0: mov    %r11d,0x2c(%rsp)
0xc5: mov    0xc(%r10),%r9d      ; *getfield x
0xc9: mov    %rdi,%rsi
0xcc: mov    %r9d,%edx
0xcf: callq  0x00000001116dc0a0 ; *invokevirtual consume
0xd4: nop
0xd5: mov    0x2c(%rsp),%r11d
0xda: inc    %r11d               ; *iinc
0xdd: mov    0x8(%rsp),%r10
0xe2: mov    0x10(%rsp),%rdi     ; *iload_2
0xe7: cmp    $300,%r11d
0xee: jl     0xc0                ; *if_icmpge
```

`MyObject`, where you at!? 😐

Graal vs. C2: Improved Inlining

```
public abstract static class A {
  int c1, c2, c3;
  public abstract void m();
}

public static class C1 extends A {
  public void m() { c1++; }
}

public static class C2 extends A {
  public void m() { c2++; }
}

public static class C3 extends A {
  public void m() { c3++; }
}
```

Graal vs. C2: Improved Inlining

```
public abstract static class A {
  int c1, c2, c3;
  public abstract void m();
}

public static class C1 extends A {
  public void m() { c1++; }
}

public static class C2 extends A {
  public void m() { c2++; }
}

public static class C3 extends A {
  public void m() { c3++; }
}
```

```
@Setup
public void setup() {
  /* mono: as[] = {c1, c1, c1, c1, c1, …} */
  /* mega: as[] = {c1, c2, c3, c1, c2, …} */
}

@Benchmark
public void test() {
  for (A a : as) {
    a.m();
  }
}
```

Graal vs. C2: Improved Inlining

```
public abstract static class A {
  int c1, c2, c3;
  public abstract void m();
}

public static class C1 extends A {
  public void m() { c1++; }
}

public static class C2 extends A {
  public void m() { c2++; }
}

public static class C3 extends A {
  public void m() { c3++; }
}
```

```
@Setup
public void setup() {
  /* mono: as[] = {c1, c1, c1, c1, c1, …} */
  /* mega: as[] = {c1, c2, c3, c1, c2, …} */
}

@Benchmark
public void test() {
  for (A a : as) {
    a.m();
  }
}
```

	Mono	Mega
C2	238.4 ± 0.6 ns	633.8 ± 15.5 ns
Graal	202.4 ± 3.0 ns	423.3 ± 14.1 ns

Graal vs. C2: Improved Data-Flow Analysis

```
@Benchmark
public int benchmark() {
  Random random = new Random();
  int sum = 0;

  for (int i = 0; i < 256; i++) {
    Circle circle = new Circle(random.nextInt(2));

    if (circle.r > 1)
      sum += circle.r;
    }
  }

  return sum;
}
```

Graal vs. C2: Improved Data-Flow Analysis

```
@Benchmark
public int benchmark() {
  Random random = new Random();
  int sum = 0;

  for (int i = 0; i < 256; i++) {
    Circle circle = new Circle(random.nextInt(2));

    if (circle.r > 1)
      sum += circle.r;
    }
  }

  return sum;
}
```

- **Graal**: 48.404 ± 2.300 ns
- **C2**: 3025.120 ± 199.282 ns

Graal vs. C2: Improved Data-Flow Analysis

```java
@Benchmark
public int benchmark() {
  Random random = new Random();
  int sum = 0;

  for (int i = 0; i < 256; i++) {
    Circle circle = new Circle(random.nextInt(2));

    if (circle.r > 1)
      sum += circle.r;
    }
  }

  return sum;
}
```

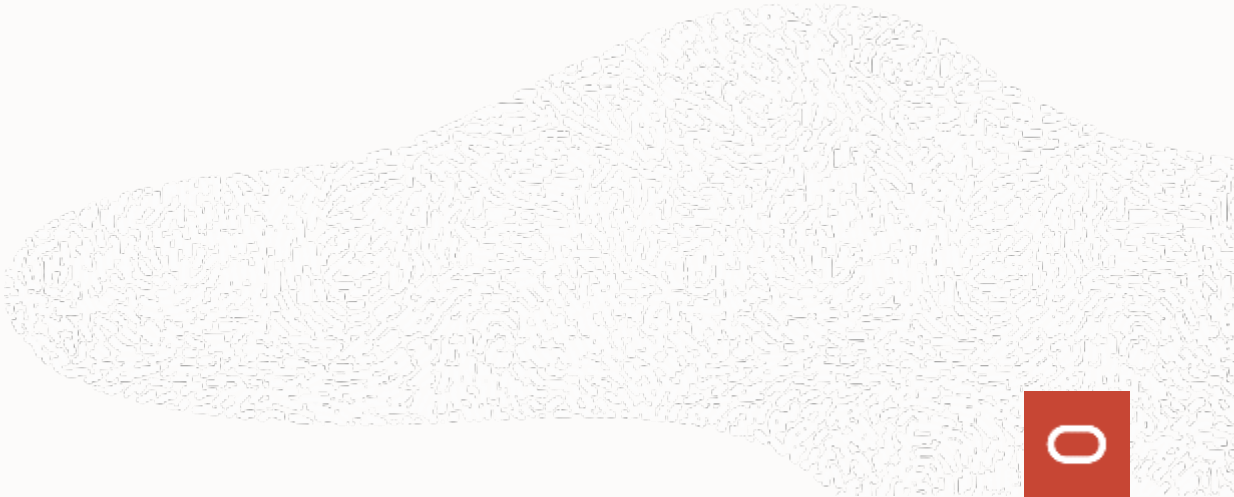
- **Graal**: 48.404 ± 2.300 ns
- **C2**: 3025.120 ± 199.282 ns

Graal:

```
0x00: inc    %esi          ; *iinc
0x01: cmp    $256,%esi
0x02: jl     0x00          ; *if_icmpge
```

Truffle

- Framework for writing "*high-performant self-optimising*" interpreters.
- Graal is aware of Truffle and partially evaluates the AST node.
- Truffle allows you to speculate and optimize.



Truffle

Guest language:

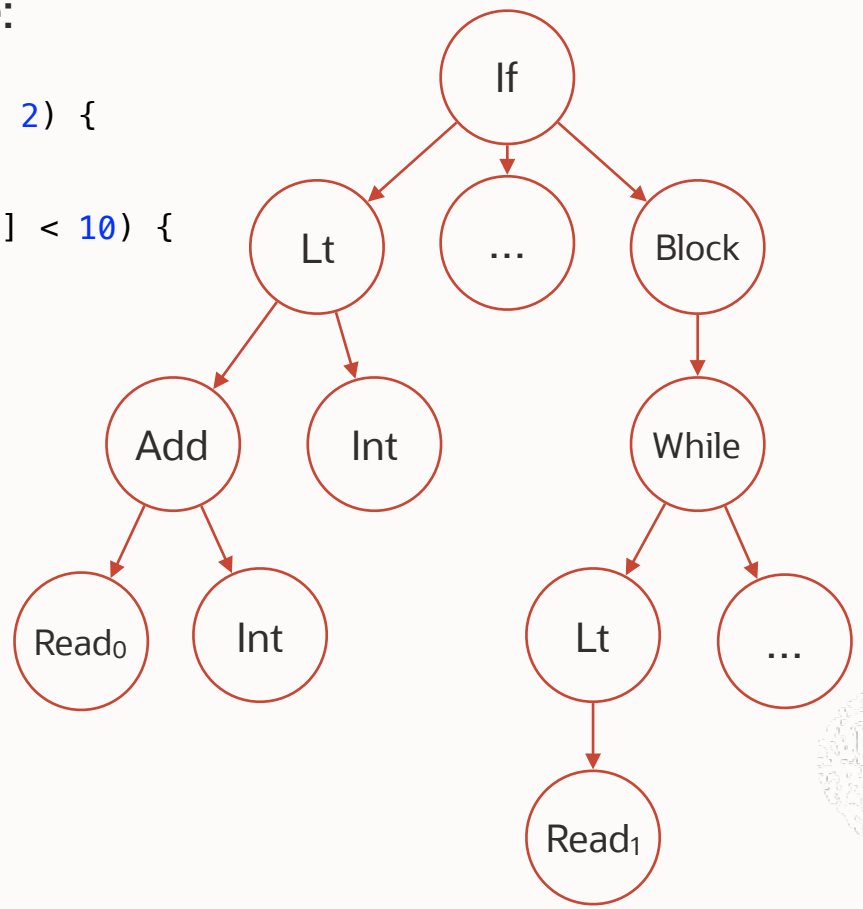
```
if (args[0] + 1 < 2) {
    // ...
} else {
    while (args[1] < 10) {
        // ...
    }
}
```

Truffle

Guest language:

```
if (args[0] + 1 < 2) {
    // ...
} else {
    while (args[1] < 10) {
        // ...
    }
}
```

AST representation:

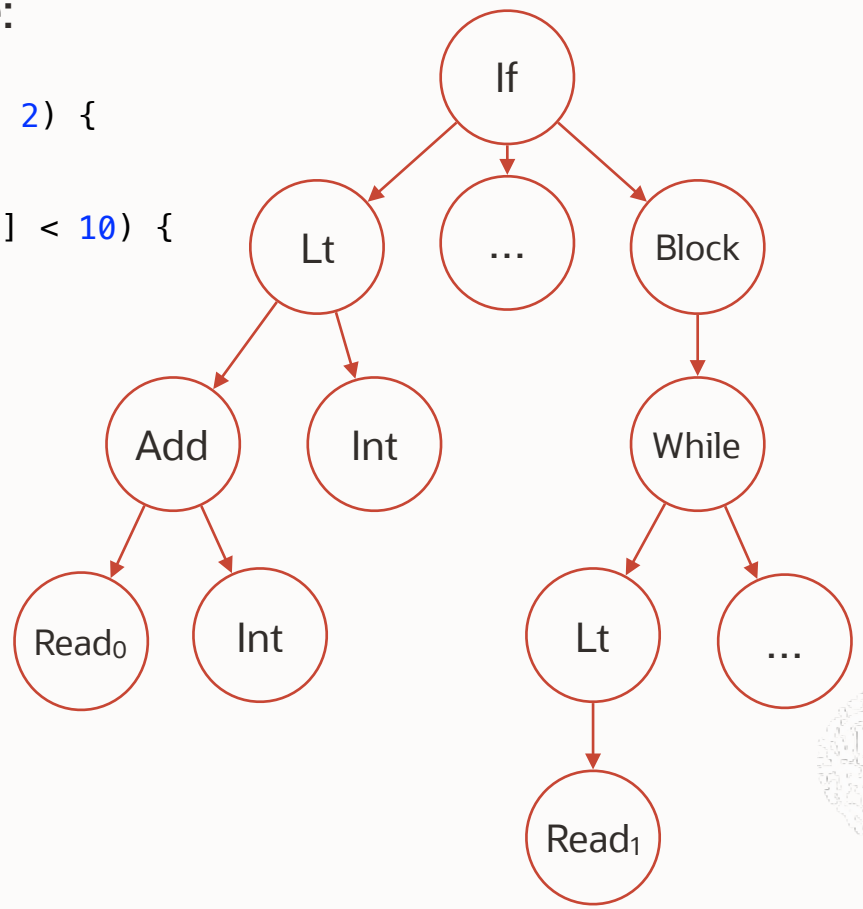


Truffle

Guest language:

```
if (args[0] + 1 < 2) {
    // ...
} else {
    while (args[1] < 10) {
        // ...
    }
}
```

AST representation:



Implementation of AST interpreter:

```
interface Node {
  int execute(Frame f);
}

class Add extends Node {
  @Child Node l, r;

  public int execute(Frame f) {
    return l.execute(f) + r.execute(f);
  }
}
```

Truffle

```
class If extends Statement {
  Condition c;
  Statement then, else;

  void execute(Frame f) {
    if (c.execute(f)) {
      then.execute(f);
    } else {
      else.execute(f);
    }
  }
}
```

```
class Lt extends Condition {
  Node l, r;

  boolean execute(Frame f) {
    return l.execute(f) < r.execute(f);
  }
}
```

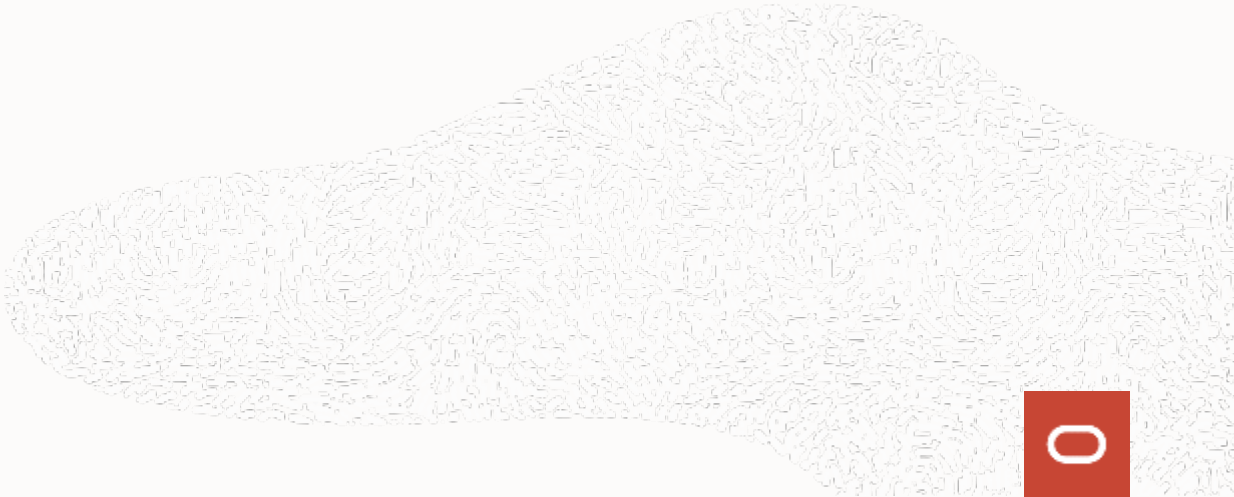
```
class Add extends Node {
  Node l, r;

  int execute(Frame f) {
    return l.execute(f) + r.execute(f);
  }
}
```

```
class Read extends Node {
  Slot s;

  int execute(Frame f) {
    return f.read(s);
  }
}
```

```
class Block extends Statement {
  Statement[] statements;

  void execute(Frame f) {
    for (Statement s: statements) {
      s.execute(f);
    }
  }
}
```

Truffle

- Our interpreter is slow because of the **interpreter overhead**:
  - Dispatch overhead: virtual method calls
  - Heap-based frame: memory access for locals
  - Loop overhead: boundary check for loop over constant collection
  - Primitive values passed as objects: boxing



Truffle: Partial Evaluation

```
class If extends Statement {
  Condition c;
  Statement then, else;

  void execute(Frame f) {
    if (c.execute(f)) {
      then.execute(f);
    } else {
      else.execute(f);
    }
  }
}
```

```
class Lt extends Condition {
  Node l, r;

  boolean execute(Frame f) {
    return l.execute(f) < r.execute(f);
  }
}
```

```
class Add extends Node {
  Node l, r;

  int execute(Frame f) {
    return l.execute(f) + r.execute(f);
  }
}
```

```
class Read extends Node {
  Slot s;

  int execute(Frame f) {
    return f.read(s);
  }
}
```

```
class Block extends Statement {
  Statement[] statements;

  void execute(Frame f) {
    for (Statement s: statements) {
      s.execute(f);
    }
  }
}
```

Truffle: Partial Evaluation

```
class Add extends Node {
  Node l, r;

  int execute(Frame f) {
    return l.execute(f) + r.execute(f);
  }
}
```

```
class If extends Statement {
  Condition c;
  Statement then, else;

  void execute(Frame f) {
    if (c.l.execute(f) < c.r.execute(f)) {
      then.execute(f);
    } else {
      else.execute(f);
    }
  }
}
```

```
class Read extends Node {
  Slot s;

  int execute(Frame f) {
    return f.read(s);
  }
}
```

```
class Block extends Statement {
  Statement[] statements;

  void execute(Frame f) {
    for (Statement s: statements) {
      s.execute(f);
    }
  }
}
```

Truffle: Partial Evaluation

```
class If extends Statement {
  Condition c;
  Statement then, else;

  void execute(Frame f) {
    if (c.l.execute(f) < c.r.execute(f)) {
      then.execute(f);
    } else {
      for (Statement s: else.statements) {
        s.execute(f);
      }
    }
  }
}
```

```
class Add extends Node {
  Node l, r;

  int execute(Frame f) {
    return l.execute(f) + r.execute(f);
  }
}
```

```
class Read extends Node {
  Slot s;

  int execute(Frame f) {
    return f.read(s);
  }
}
```

Truffle: Partial Evaluation

```
class If extends Statement {
  Condition c;
  Statement then, else;

  void execute(Frame f) {
    if (c.l.l.execute(f) + c.l.r.execute(f) < c.r.execute(f)) {
      then.execute(f);
    } else {
      for (Statement s: else.statements) {
        s.execute(f);
      }
    }
  }
}
```

```
class Read extends Node {
  Slot s;

  int execute(Frame f) {
    return f.read(s);
  }
}
```

Truffle: Partial Evaluation

```
class If extends Statement {
  Condition c;
  Statement then, else;

  void execute(Frame f) {
    if (f.read(c.l.l.s) + c.l.r.execute(f) < c.r.execute(f)) {
      then.execute(f);
    } else {
      for (Statement s: else.statements) {
        s.execute(f);
      }
    }
  }
}
```

Truffle: Specialization

```
class Add extends Node {
  Node left, right;

  Object execute(Frame frame) {
    Object l = left.execute(frame);
    Object r = right.execute(frame);

    if (l instanceof Integer && r instanceof Integer) {
      return l + r;
    } else if (l instanceof String || r instanceof String) {
      return left.toString() + right.toString();
    } else {
      throw new RuntimeException("Type error");
    }
  }
}
```


Truffle: Specialization

```
abstract class Add extends Node {
  @Specialization
  int executeInt(int x, int y) {
    return x + y;
  }

  @Specialization
  String executeString(String x, String y) {
    return x + y;
  }
}
```

ASM code:

- Check if operands are indeed integers
- If not, jump and deoptimize.
- If yes, perform integer addition.

- Check if operands are indeed Strings
- If not, jump and deoptimize.
- If yes, perform String concatenation.

Aside: GraalVM

GraalVM™

- GraalVM is the distribution that contains:
  - Graal: An optimising JIT compiler for Java bytecode.
  - Truffle Languages: JavaScript, Sulong.
  - An ahead-of-time compiler for Java bytecode (`native-image`).
  - Embeddable in existing systems (MLE = Database Multilingual Engine).

Active Libraries

Generative Programming and Active Libraries

Extended Abstract

Krzysztof Czarnecki[1], Ulrich Eisenecker[2], Robert Glück[3],
David Vandevoorde[4], and Todd Veldhuizen[5]

Active libraries are not passive collections of routines or objects, as are traditional libraries, but take an active role in generating code. Active libraries provide abstractions and can optimize those abstractions themselves. They may generate components, specialize algorithms, optimize code, automatically configure and tune themselves for a target machine, check source code for correctness, and report compile-time, domain-specific errors and warnings. They may also describe

Fast Access Data (FAD.js)

- A Truffle “language” that works together with Graal.js to improve JSON processing using:
  - Constant structure encoding
  - Direct structure decoding

FAD.js: Constant structure encoding

```
connection.on('data', function (data) {
    var entry = JSON.stringify(data) + '\n';
    fs.appendFileSync('/some/file', entry);
});
```

A generic JSON encoder needs to:

- Recursively walk the object graph
- Read the property names of the object
- Read the value of each property
- Append the property names and values
- Perform necessary formatting

FAD.js: Constant structure encoding

```
connection.on('data', function (data) {
    var entry = JSON.stringify(data) + '\n';
    fs.appendFileSync('/some/file', entry);
});
```

```
var data = {
    id: /* any number, always present */,
    name: /* any string, always present */,
    value: { /* any object value, or empty */ }
};
```

A generic JSON encoder needs to:

- Recursively walk the object graph
- Read the property names of the object
- Read the value of each property
- Append the property names and values
- Perform necessary formatting

FAD.js: Constant structure encoding

```
public class EncodingNode extends ASTNode {
  final Shape expectedShape;
  final String encA = "{\'id\':";
  final String encB = ",\'name\':\'";
  final String encC = ",\'value\':";

  @Child final ReadNode[] prop;
  @Child final EncodingNode next;

  public void executeNode(JSObject input, StringBuilder result) {
    if (input.getShape() != expectedShape) {
      throw new RewriteASTException();
    }

    String valueA = prop[0].read("id");
    String valueB = prop[1].read("name");
    String valueC = next.executeNode(prop[2].read("value"));

    result.append(encA + valueA);
    result.append(encB + valueB);
    result.append(encC + valueC + "}");
  }
}
```

FAD.js: Constant structure encoding

- By assuming that shape is constant and specializing on this assumption:
  - No need for a full recursive walk of the object graph.
  - No need to retrieve the list of properties.
  - Offset-based property access (instead of hash-based).
  - Prepare the target string (including ‘{‘, ‘:’, ‘,’, ‘}’, ‘[’, ‘]’)

- But:
  - Constant-time check on the shape of the provided object.
  - Deoptimize when the shape turns out to be non-constant.

Oracle JDBC Active Library (OJDBC.AL)

- Java API that defines how a client may access a database.
- Trying to do the same trick as FAD.js:
  - Specialize on server version
  - Specialize on character encoding always being AL32UTF8
  - Specialize on kind of query (i.e. SELECT, non-PL/SQL)
  - Speculate that character decoding only sees ASCII

Recap

- Graal is a **JIT compiler** for Java bytecode written in Java. It can achieve better performance than C2, partly because of better **inlining** and **partial escape analysis**.
- Truffle is a **language implementation** framework. Truffle nodes are **partially evaluated** and can **self-optimize**.
- **Active libraries** take an active role in generating code. **FAD.js** is an active library that build on top of Graal & Truffle that generates specialized code for JSON encoding based on the shapes it has seen.

References

- Stadler, L., Würthinger, T., & Mössenböck, H. (2014, February). Partial escape analysis and scalar replacement for java. In *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization* (p. 165). ACM.
- Würthinger, T., Wimmer, C., Humer, C., Wöß, A., Stadler, L., Seaton, C., ... & Grimmer, M. (2017). Practical partial evaluation for high-performance dynamic language runtimes. *ACM SIGPLAN Notices*, 52(6), 662-676.
- *Truffle: A Self-Optimizing Runtime System* by Christian Wimmer (https://pdfs.semanticscholar.org/1d04/1caf21de4f3b527094daac7dc348128c5083.pdf)
- *JVM Anatomy Quark #16: Megamorphic Virtual Calls* by Aleksey Shipilëv (https://shipilev.net/jvm/anatomy-quarks/16-megamorphic-virtual-calls/)
- *A race of two compilers: GraalVM JIT versus HotSpot JIT C2* by Ionut Balosin (https://youtu.be/lunJmMBkqLo)
- *Graal: Not Just a New JIT for the JVM* by Duncan MacGregor (https://youtu.be/giC1Vs7-xuU)
- *One VM to Rule Them All, One VM to Bind Them* by Christian Wimmer (https://youtu.be/FJY96_6Y3a4)

ORACLE

Internships at Oracle Labs

Internships at Oracle Labs Zurich

regular internships or MSc thesis

3 to 12 months

CHF 4000 monthly gross salary

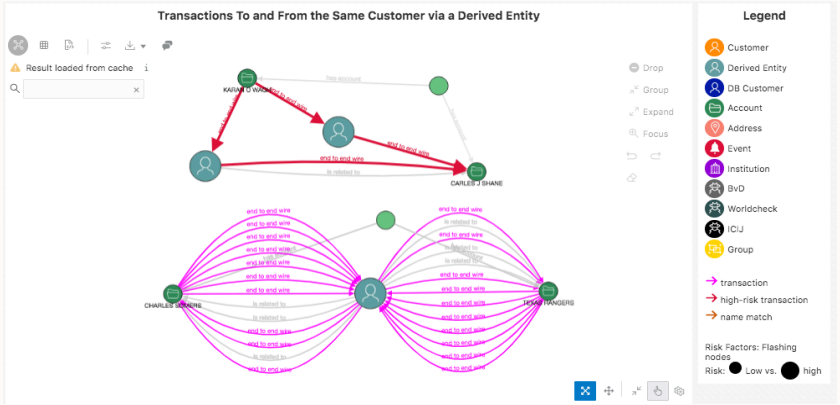
Internship Topics at Oracle Labs

Oracle Labs PGX: fast in-memory graph processing

- runtimes, concurrency, distributed programming
- Java, C, C++

Oracle Labs Data Studio: a web-based notebook platform

- web technologies (frontend & backend)
- graph visualization

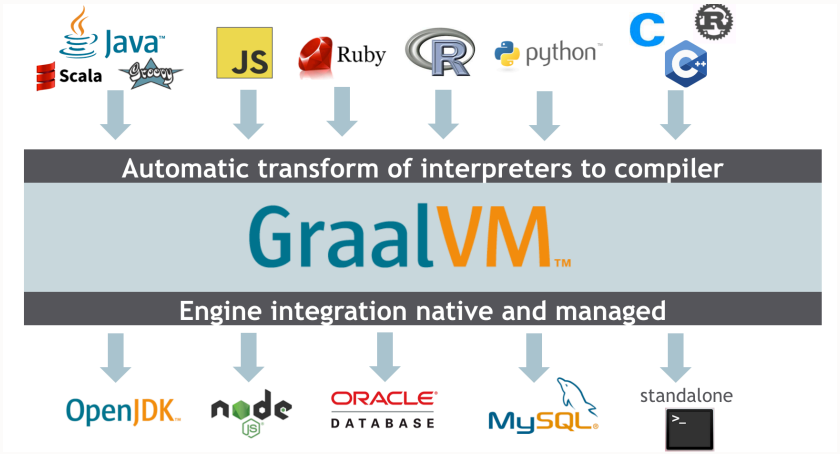


Oracle Financial Services Crime and Compliance Studio

- machine learning, system integration, elasticity
- Spark

Oracle Database Multilingual Engine

- database, compilers, data science
- C, Javascript, Python



Internships

- **Apache Avro**: data serialisation framework similar to Apache Thrift and Google Protocol Buffers.
  - Write a (de)serializer that optimises itself according to certain access patterns.
    - Skills required: Java, compilation, one dynamic language.
    - Skills learned: GraalVM Truffle, compiler optimizations, big data stack.

- **PackedDecimal**: efficient and compact representation of decimal number. Used by the Oracle DB.
  - Skills required: mathematics, Java, compilation (low level, assembly)
  - Skills learned: numerical computation, low-level representation of data types

ORACLE®
DATABASE

Get in touch.

guido.wachsmuth@oracle.com