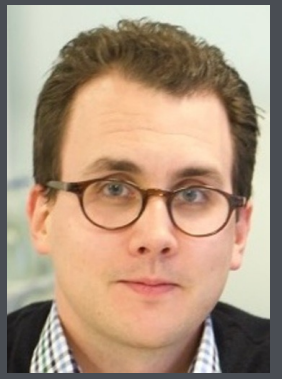
ORACLE



December 20th 2023

GraalVM and Oracle Database Multilingual Engine: languages and compilers in the Wild

And a Sneak Peek into other stuff happening at Oracle Labs Zurich

Dr. Lucas Braun
Consulting Technical Program Manager
Oracle Labs Zurich

Lucas Braun

- Consulting Technical Program Manager @ Oracle Labs
- BSc, MSc and PhD in Computer Science from ETH
- Started at Oracle Labs in 2017
- Working on Oracle Database Multilingual Engine (MLE)

lucas-braun-277102153

@lucasbraun87

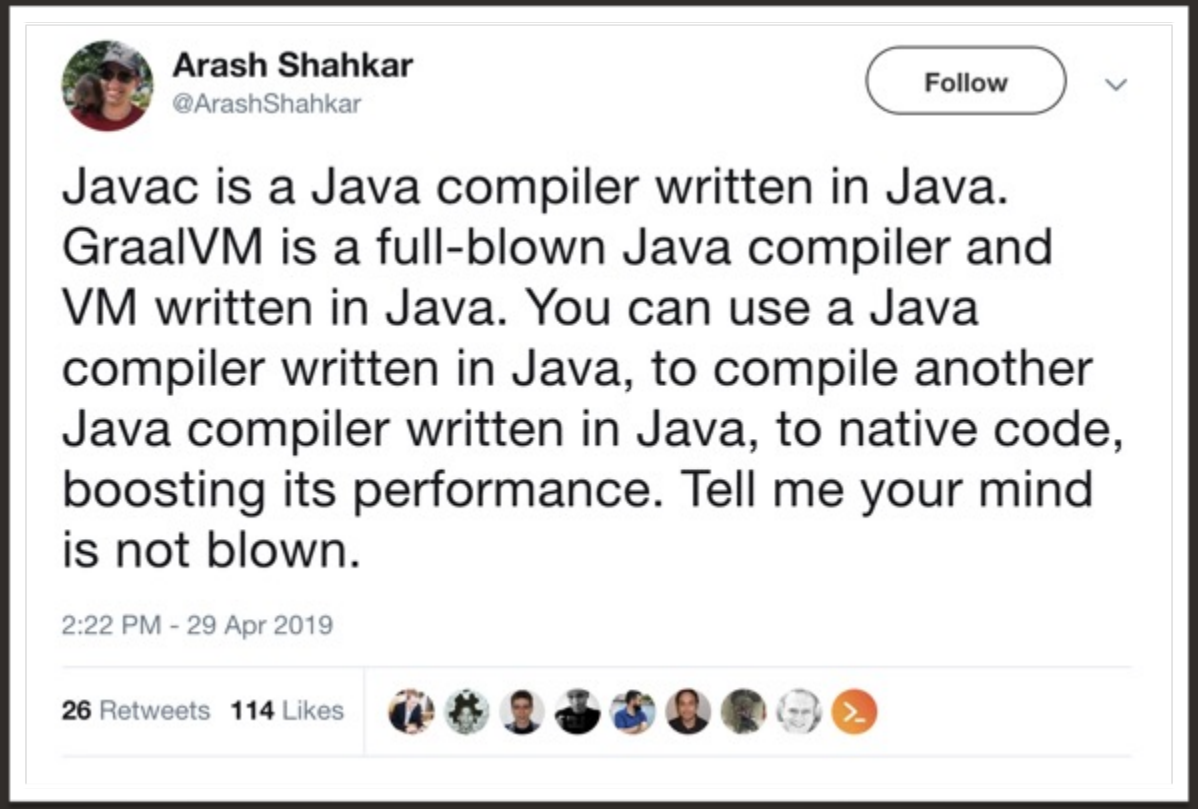
Agenda

1. GraalVM
   - GraalVM Compiler & (Partial) Escape Analysis
   - Truffle Framework & Speculative Optimizations
2. Oracle Database Multilingual Engine (MLE)
   - GraalVM in the Database: MLE Vision and Architecture
   - Demo: APEX + MLE = Low-Code + JavaScript = AWESOME DEVELOPMENT
   - MLE Roadmap and Future Work
3. A Quick Intro into Oracle Labs + Internships

Agenda

1. **GraalVM**
   - GraalVM Compiler & (Partial) Escape Analysis
   - Truffle Framework & Speculative Optimizations
2. Oracle Database Multilingual Engine (MLE)
   - GraalVM in the Database: MLE Vision and Architecture
   - Demo: APEX + MLE = Low-Code + JavaScript = AWESOME DEVELOPMENT
   - MLE Roadmap and Future Work
3. A Quick Intro into Oracle Labs + Internships

GraalVM magic in one tweet



Why GraalVM ?

High Performance

Optimize application performance with GraalVM compiler

Fast Startup

Compile your application AOT and start instantly

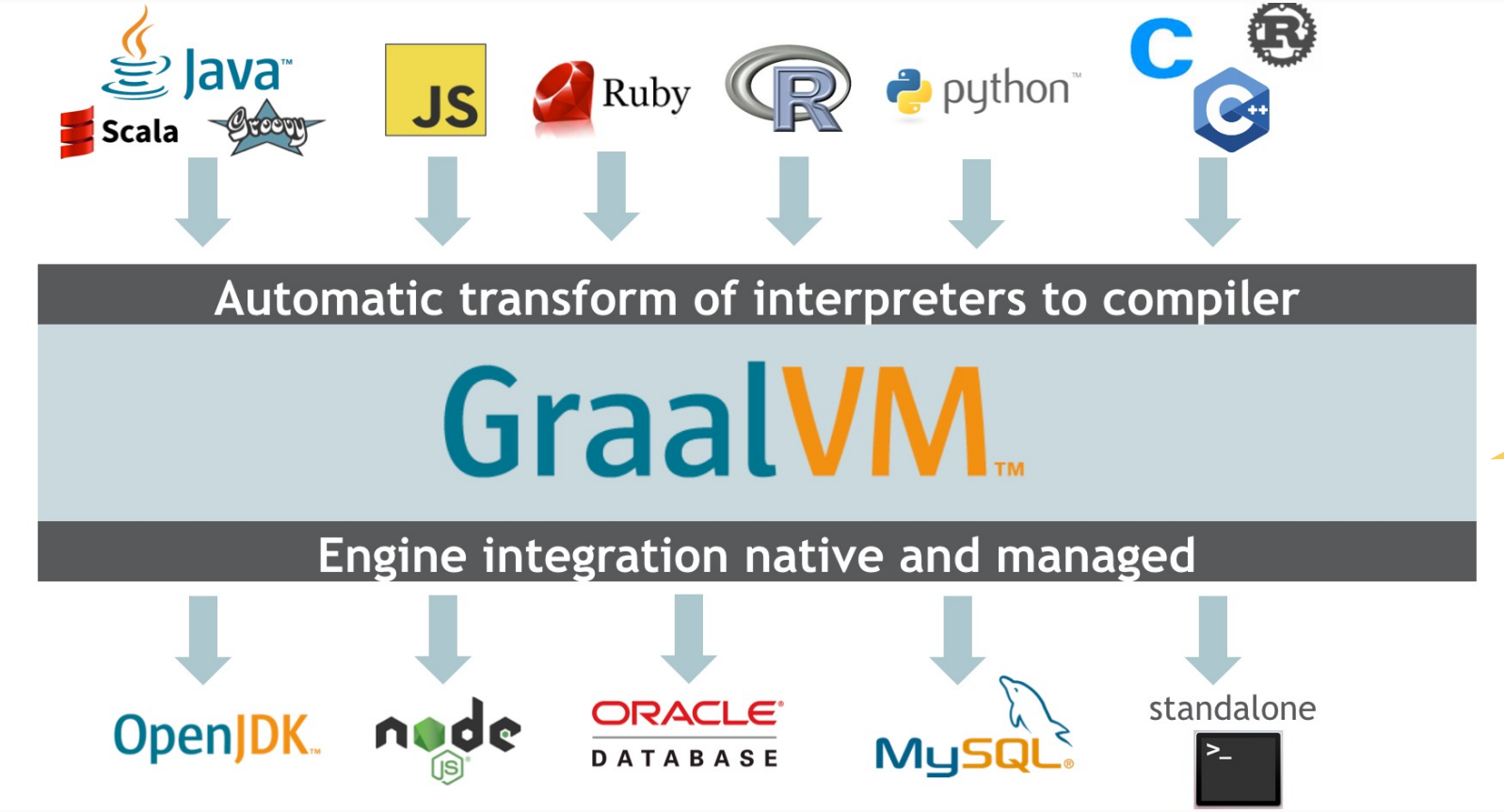
Polyglot

Mix & match languages with seamless interop

Open Source

See what's inside, track features progress, contribute

GraalVM: Multi-Lingual, Embeddable Virtual Machine



Multi-Lingual

Twitter gets between **12% (CE) and 24% (EE)** improvement by running on Graal. This directly translates to cost savings!

Embeddable

Production-Ready

GraalVM Architecture

LLVM Langs

JVM Languages

Truffle Languages

Language Implementation API (Truffle Framework)

GraalVM JIT Compiler

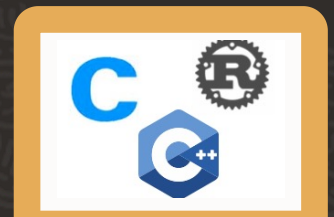
Java HotSpot VM

GraalVM is Open-source and Free to use

Agenda

1. GraalVM
   - **GraalVM Compiler & (Partial) Escape Analysis**
   - Truffle Framework & Speculative Optimizations
2. Oracle Database Multilingual Engine (MLE)
   - GraalVM in the Database: MLE Vision and Architecture
   - Demo: APEX + MLE = Low-Code + JavaScript = AWESOME DEVELOPMENT
   - MLE Roadmap and Future Work
3. A Quick Intro into Oracle Labs + Internships

GraalVM Architecture



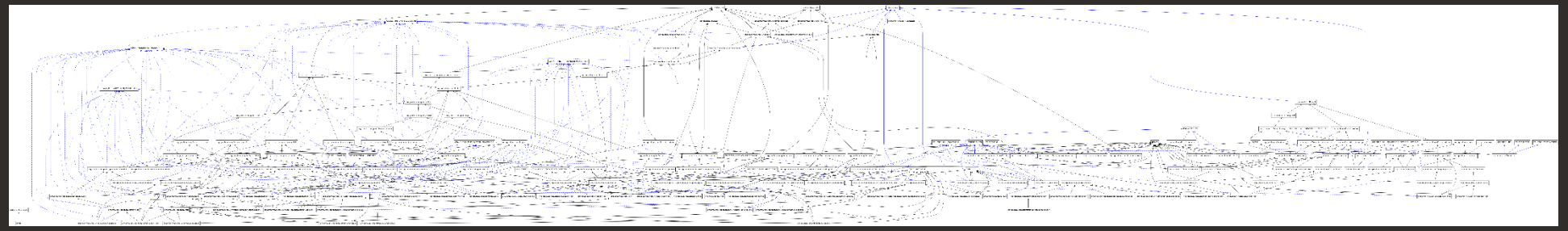
Language Implementation API (Truffle Framework)

GraalVM JIT Compiler

Java HotSpot VM

GraalVM compiler

- Java JIT Compiler implemented itself in Java
- Highly Modular Design



- Novel Optimizations
- Foundation for more than 60 publications at premier venues (PLDI, CGO, OOPSLA, …)
- Strong open source development and community

GraalVM compiler

- **Highly Optimizing**
- **Based on Deoptimization & Speculation**
- **Various Optimizations**
  - Peephole (Several Levels)
  - Constant Folding
  - Inlining
  - Partial Escape Analysis
  - Lock Elimination
  - Duplication
  - Loop Unrolling/Unswitching/Peeling/LICM/…
  - Null Check Elimination
  - Constant InstanceOf Checks
  - SafeVector Elimination
  - Compressed Vectorers
  - Biased Locking
  - Speculative Optimizations
    - Only compile executed code
    - Only compile executed exception paths
    - Class Hierarchy Analysis
    - ….

Escape Analysis

- **Compiler analysis used in optimizations**
  - Objects are always allocated
  - Objects “escape” the current scope if they leave the compiler’s field of view (i.e., the current compilation unit)
    - Escaping means
      - Used as a parameter
      - Written to static field
      - Return value
      - Thrown
      - ….

Escape Analysis Example

```
static class Example {
        int x;
        public Example(int x) { this.x = x; }
}
static Object SideEffect;
static final Example Cached = new Example(12);
public Example foo(int x) {
        Example e = new Example(x);
        if (e.x % 2 == 0) {
                System.out.println("Foo");
        } else {
                System.out.println(e);
        }
        if (e.x % 3 == 0) {
                SideEffect = e;
        }
        if (e.x == 17) {
                return e;
        }
        return Cached;
}
```

Allocation

Escape

Escape

Escape

Escape Analysis

- **What to do with this knowledge**

  Object Allocations are expensive

  Want to avoid them if possible

  Scalar Replacement

  Replace the usage of an object with its fields

Scalar Replacement

```
static class Vector {
   int x, y;
   public Vector(int x, int y) {
       this.x = x;
       this.y = y;
   }
   Vector add(Vector p) {
      return new Vector(this.x + p.x, this.y +
p.y);
   }
   int abssq() {
      return x * x + y * y;
   }
}
```

Replace with Scalar Values

```
static int vectorUsage(int x1, int y1, int x2, int y2)
{
   Vector p1 = new Vector(x1, y1);
   Vector p2 = new Vector(x2, y2);
   return p1.add(p2).abssq();
}
```

Inline add & abssq

```
static int vectorUsageInlined(int x1, int y1, int x2,
int y2) {
  Vector p1 = new Vector(x1, y1);
  Vector p2 = new Vector(x2, y2);
  Vector tmp = new Vector(p1.x + p2.x, p1.y + p2.y);
  return tmp.x * tmp.x + tmp.y * tmp.y;
}
```

```
static int vectorUsageScalar(int x1, int y1, int x2,
int y2) {
   int x = x1 + x2;
   int y = y1 + y2;
   return x * x + y * y;
}
```

Escape Analysis & Scalar Replacement

- **Multiple Possible Levels**

  Object does not escape compilation unit

  Object allocations are removed & fields are replaced with their stack allocated values (typically reside in registers in the compiled code)

  Object does not escape thread

  Lock Removal: If no other thread can see the object, we can remove locking

  Object escapes

  No Optimization possible (but optimized allocation → TLABs [1])

[1] TLABs: thread local allocation buffers

Traditional Escape Analysis

- **Traditional escape analysis is done on an entire compilation unit**

  If an object escapes anywhere in the scope, scalar replacement is prohibited

```
public Example fooSimple(int x) {
    Example e = new Example(x);
    if (e.x % 2 == 0) {
      System.out.println("Foo");
      return Cached;
    } else {
      return e;
    }
}
```

e does not escape

e escapes

But escaping e in false branch disables EA and scalar replacement on the entire method

What if ?

- **What if the code at runtime hardly requires the allocation?**

```
public Example fooSimple(int x) {
    Example e = new Example(x);
    if (e.x % 2 == 0) {
      System.out.println("Foo");
      return Cached;
    } else {
      return e;
    }
}
```

What if the probability for the true successor is 99% ? → We would allocate although we only need the new e in 1% of the cases

e does not escape

e escapes

Partial Escape Analysis

- What if the code at runtime hardly requires the allocation?

```
public Example fooSimple(int x) {
    Example e = new Example(x);
    if (e.x % 2 == 0) {
      System.out.println("Foo");
      return Cached;
    } else {
      return e;
    }
}
```

e does not escape

e escapes

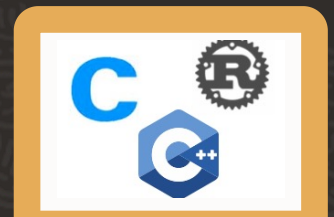
```
public Example fooSimple_PEA(int x) {
    if (x % 2 == 0) {
      System.out.println("Foo");
      return Cached;
    } else {
      return new Example(x);
    }
}
```

e still escapes but the allocation is only done in 1% of the cases

Agenda

1. GraalVM
   - GraalVM Compiler & (Partial) Escape Analysis
   - **Truffle Framework & Speculative Optimizations**
2. Oracle Database Multilingual Engine (MLE)
   - GraalVM in the Database: MLE Vision and Architecture
   - Demo: APEX + MLE = Low-Code + JavaScript = AWESOME DEVELOPMENT
   - MLE Roadmap and Future Work
3. A Quick Intro into Oracle Labs + Internships

GraalVM Architecture



LLVM Langs

JVM Languages

Truffle Languages

Language Implementation API (Truffle Framework)

GraalVM JIT Compiler

Java HotSpot VM

Truffle Language Implementation Framework

From the Truffle Homepage:

"The Truffle language implementation framework (henceforth "Truffle") is an open source library for building tools and programming languages implementations **as interpreters** for **self-modifying Abstract Syntax Trees**. Together with the open source GraalVM compiler, Truffle represents a significant step forward in programming language implementation technology in the current era of dynamic languages."

- Programming language implementers only need to **implement an interpreter**.
- If interpreter runs on a default JVM, language execution is correct, but slow (always interpreted).
- If interpreter runs GraalVM, GraalVM handles compilation and performance.

AST Interpreters

AST = Abstract Syntax Tree

- The tree produced by a parser of a high-level language compiler

Every node can be executed

- For our purposes, we implement nodes as a class hierarchy
- Abstract `execute` method defined in `Node` base class
- Execute overwritten in every subclass

Children of an AST node produce input operand values

- Example: `AddNode` to perform addition has two children: `left` and `right`
  - AddNode.execute first calls left.execute and right.execute to compute the operand values
  - Then peforms the addition and returns the result
- Example: `IfNode` has three children: `condition`, `thenBranch`, `elseBranch`
  - `IfNode.execute` first calls `condition.execute` to compute the condition value
  - Based on the condition value, it either calls `thenBranch.execute` or `elseBranch.execute` (but never both of them)

Textbook summary

- Execution in an AST interpreter is slow (virtual call for every executed node)
- But, easy to write and reason about; portable

Partial Evaluation Example

```
abstract class Expression extends Node {
    abstract int execute(int[] arguments);
}

class Add extends Expression {
    @Child Expression left;
    @Child Expression right;

    Add(Expression left, Expression right) {
        this.left = left;
        this.right = right;
    }

    int execute(int[] args) {
        return left.execute(args) + right.execute(args);
    }
}
```

```
class Arg extends Expression {
    final int index;

    Arg(int index) { this.index = index; }

    int execute(int[] args) {
        return args[index];
    }
}
```

```
int interpret(Expression expression, int[] args) {
    return expression.execute(args);
}
```

```
// Sample program (arg[0] + arg[1]) + arg[2]
sample = new Add(new Add(new Arg(0), new Arg(1)), new Arg(2));
```

Partial Evaluation Example (2)

```
// Sample program (arg[0] + arg[1]) + arg[2]
sample = new Add(new Add(new Arg(0), new Arg(1)), new Arg(2));
```

```
int interpret(Expression expression, int[] args) {
    return expression.execute(args);
}
```

partiallyEvaluate(interpret, sample)

```
int interpretSample(int[] args) {
    return sample.execute(args);
}
```

Partial Evaluation Example (3)

```
// Sample program (arg[0] + arg[1]) + arg[2]
sample = new Add(new Add(new Arg(0), new Arg(1)), new Arg(2));
```

```
int interpretSample(int[] args) {
    return sample.execute(args);
}
```

```
int interpretSample(int[] args) {
    return sample.left.execute(args)
         + sample.right.execute(args);
}
```

```
int interpretSample(int[] args) {
    return sample.left.left.execute(args)
         + sample.left.right.execute(args)
         + args[sample.right.index];
}
```

```
int interpretSample(int[] args) {
    return args[sample.left.left.index]
         + args[sample.left.right.index]
         + args[sample.right.index];
}
```

```
int interpretSample(int[] args) {
    return args[0]
         + args[1]
         + args[2];
}
```

Truffle DSL for Specialization

```
@NodeChildren({@NodeChild("leftNode"), @NodeChild("rightNode")})
public abstract class SLBinaryNode extends SLExpressionNode { }

public abstract class SLAddNode extends SLBinaryNode {

  @Specialization(rewriteOn = ArithmeticException.class)
  protected final long add(long left, long right) {
    return ExactMath.addExact(left, right);
  }

  @Specialization
  protected final BigInteger add(BigInteger left, BigInteger right) {
    return left.add(right);
  }

  @Specialization(guards = "isString(left, right)")
  protected final String add(Object left, Object right) {
    return left.toString() + right.toString();
  }

  protected final boolean isString(Object a, Object b) {
    return a instanceof String || b instanceof String;
  }
}
```

The order of the @Specialization methods is important: the first matching specialization is selected

For all other specializations, guards are implicit based on method signature

Truffle annotation processor converts this into a giant if-then-else

Compilation

Branch and Class profiles collect statistics

Automatic **partial evaluation** of AST

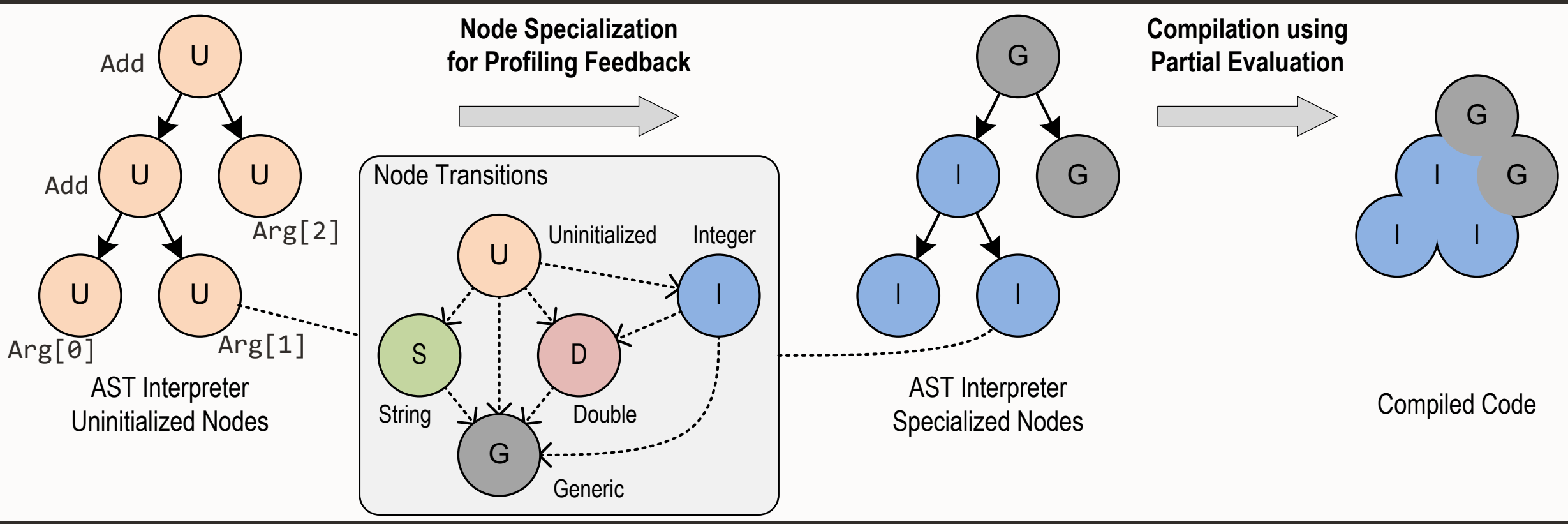
- Automatically triggered by function execution count
- Typically happening in a background thread
- Eventually compiled to Machine Code and stored in Code Cache
- Ready to be executed from Code Cache on next execution

Compilation assumes that the AST is stable

- All `@Child` and `@Children` fields treated like `final` fields
- If-then-else branches replaced by the concrete specialized method based on branch profiles
- Abstract classes replaced by concrete classes

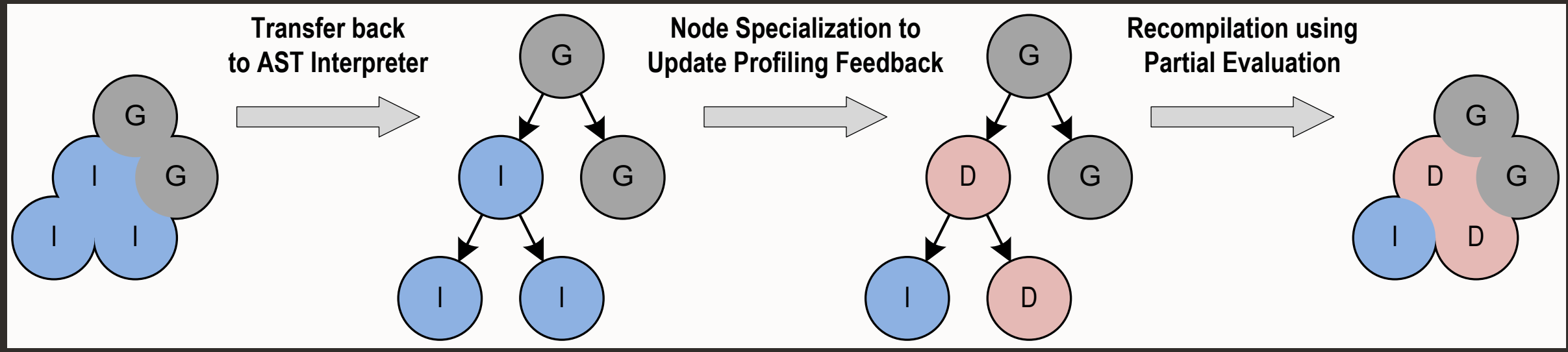
Speculate and Optimize …

```
// Sample program (arg[0] + arg[1]) + arg[2]
sample = new Add(new Add(new Arg(0), new Arg(1)), new Arg(2));
```



… and Transfer to Interpreter and Reoptimize!

```
// Sample program (arg[0] + arg[1]) + arg[2]
sample = new Add(new Add(new Arg(0), new Arg(1)), new Arg(2));
```



In Summary

GraalVM

- GraalVM is a multilingual JIT compiler and runtime
- GraalVM allows writing your own language with the Truffle API
- GraalVM achieves high performance execution by speculative (and other) optimizations

Any questions?

Agenda

1. GraalVM
   - GraalVM Compiler & (Partial) Escape Analysis
   - Truffle Framework & Speculative Optimizations
2. Oracle Database Multilingual Engine (MLE)
   - **GraalVM in the Database: MLE Vision and Architecture**
   - Demo: APEX + MLE = Low-Code + JavaScript = AWESOME DEVELOPMENT
   - MLE Roadmap and Future Work
3. A Quick Intro into Oracle Labs + Internships

MLE Vision

MLE Vision

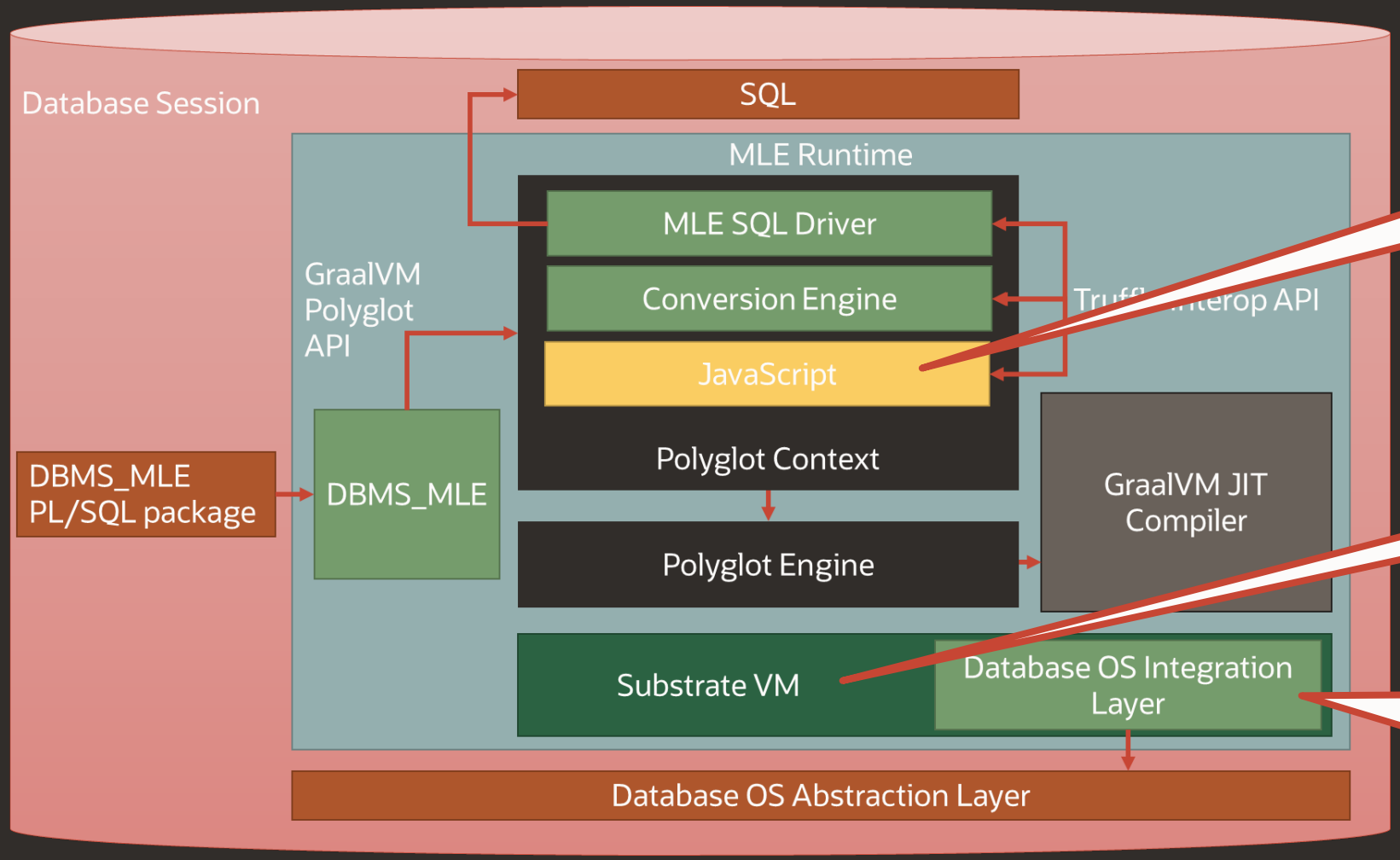
Multilingual Engine

Oracle Database Multilingual Engine: GraalVM in the Database

- Started as an Oracle Labs research project some time in 2013.
- Was **released on December 8, 2020** as part of the Oracle Database 21c in the Oracle Cloud [1].
- First release features dynamic JavaScript execution through DBMS_MLE [2].
- Oracle 23c adds support for code management, function calls, debugging [3].

[1] Oracle Database Always-Free Tier, check it out on oracle.com/free
[2] docs.oracle.com/en/database/oracle/oracle-database/21/arpls/dbms_mle.html#GUID-3F5B47A5-2C73-4317-ACD7-E93AE8B8E301
[3] https://www.youtube.com/watch?v=jvosCJb-bA8&list=PLcFwxJMrxygCzLt6Ct-9fJjWWAfvlyBRM&index=22

MLE Architecture

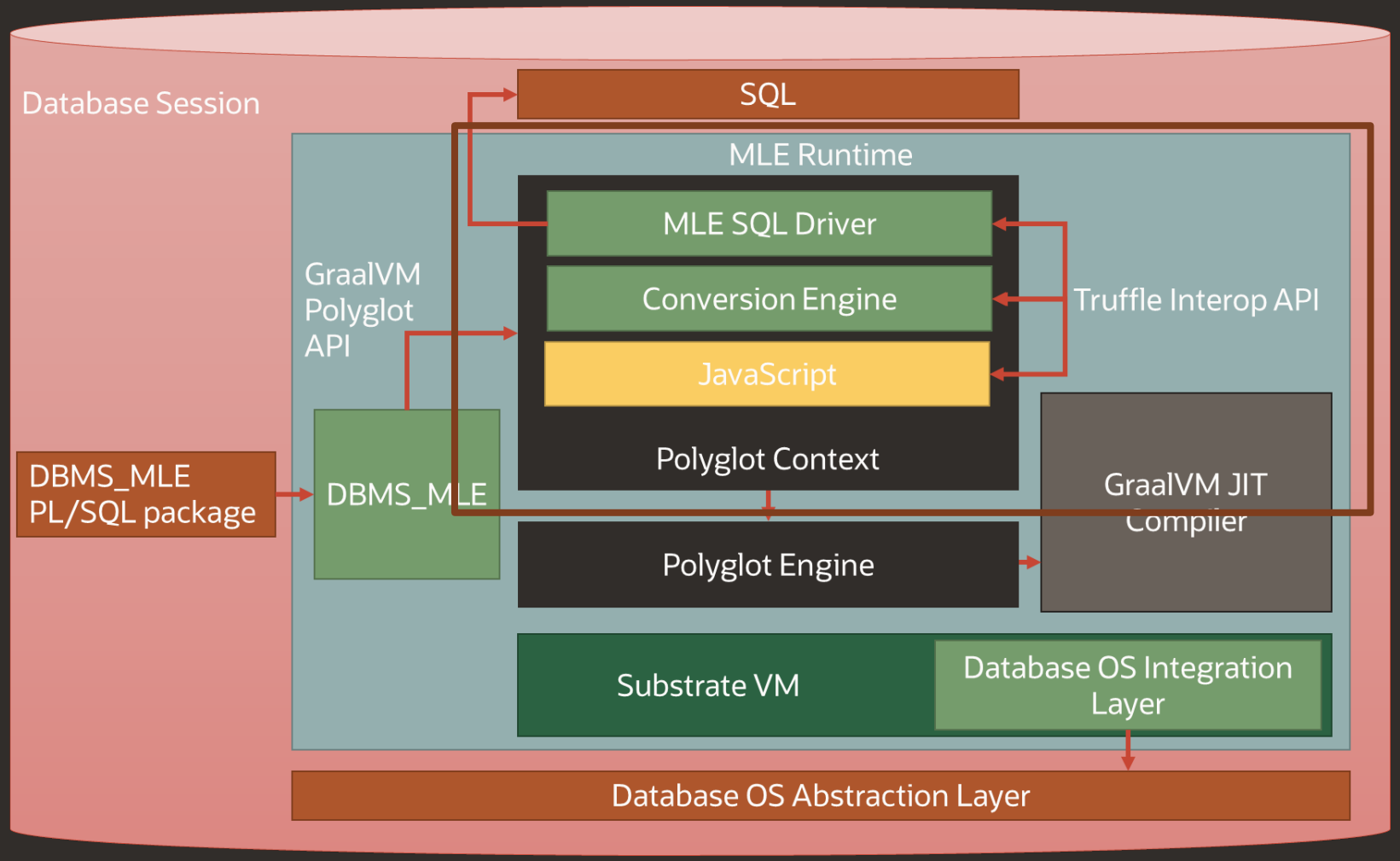


runs newest ECMAScript

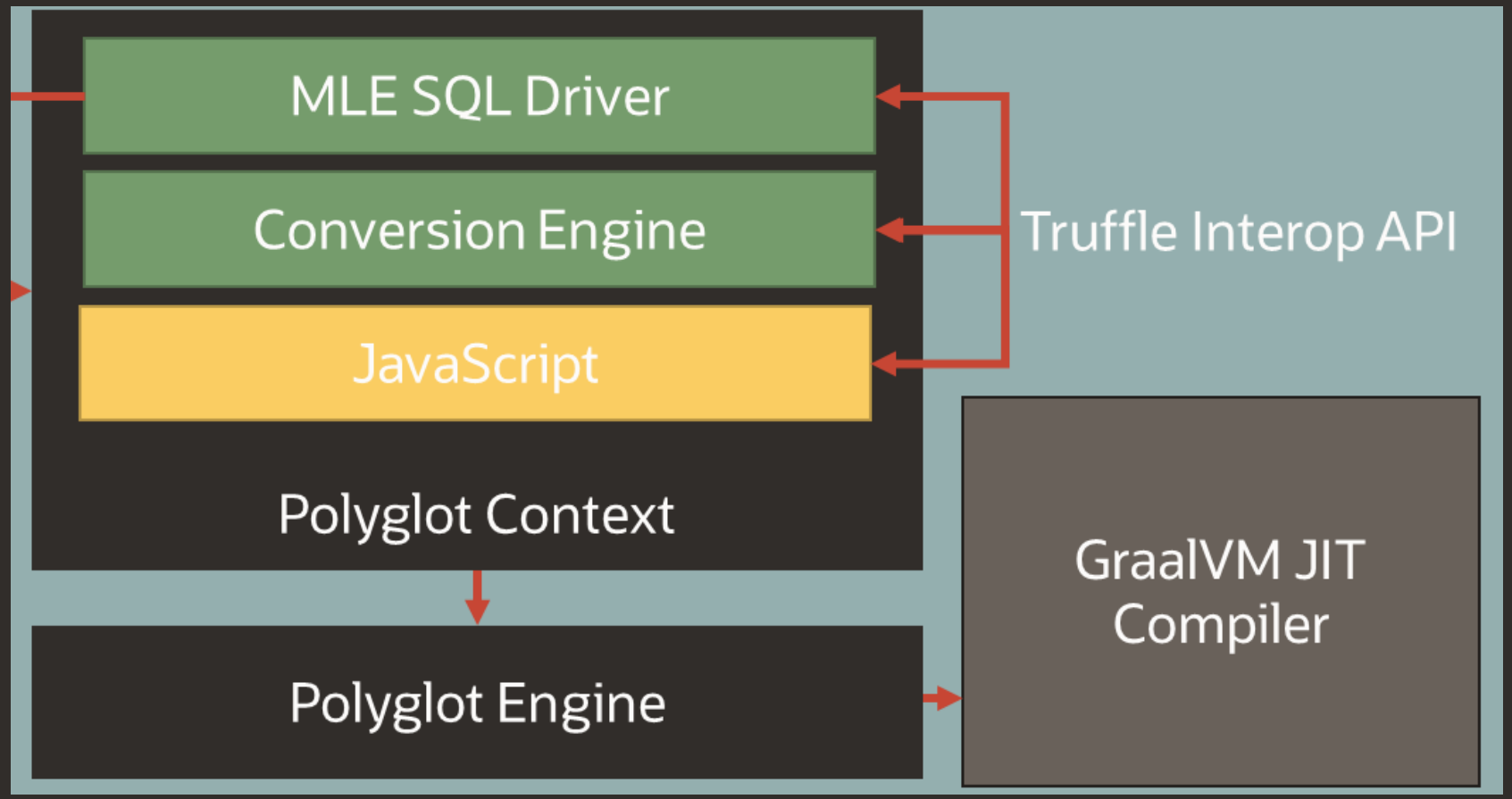
provides fast startup and low memory footprint

- blocks file access
- blocks network I/O
- controls memory allocation and threading

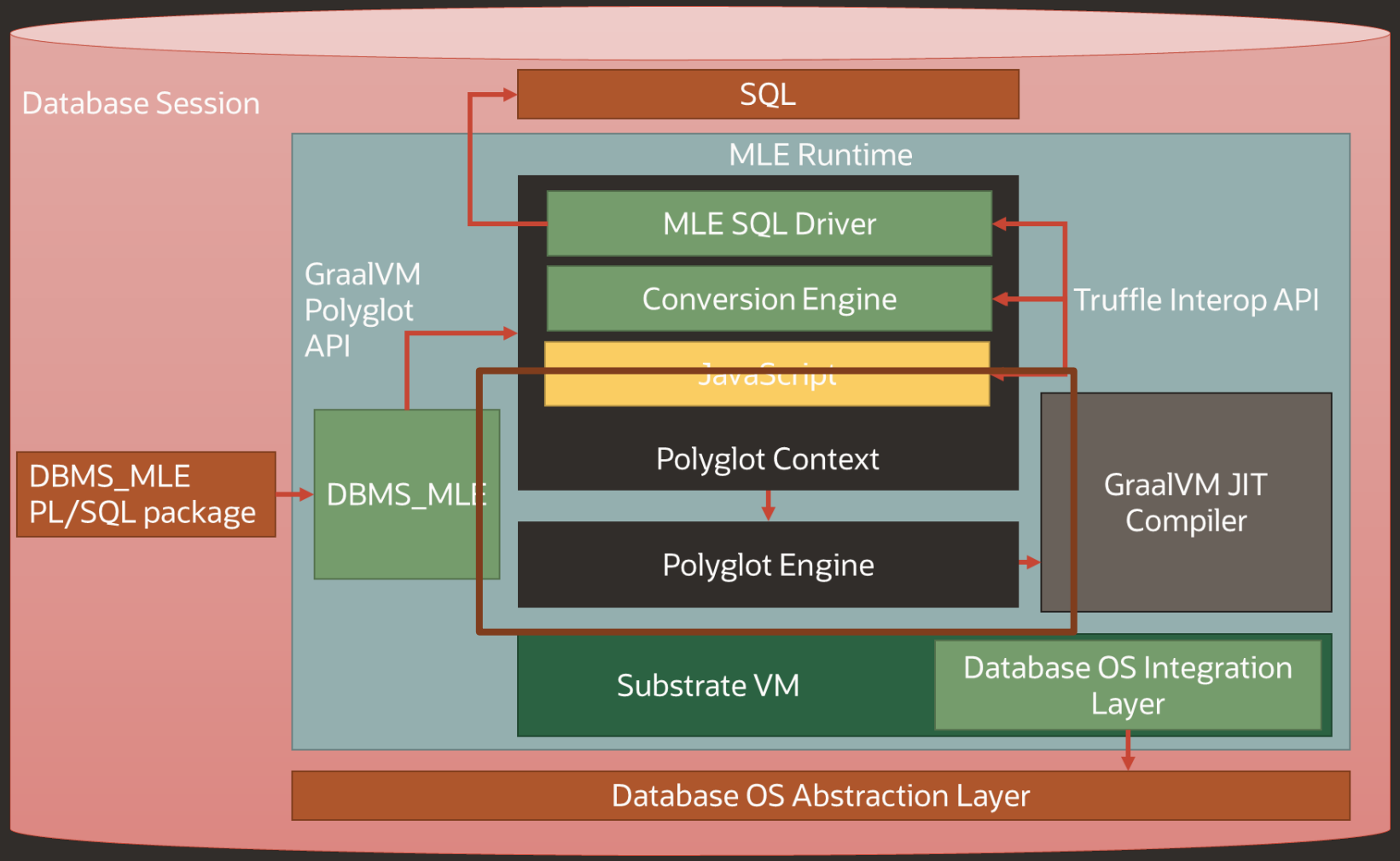
MLE Architecture



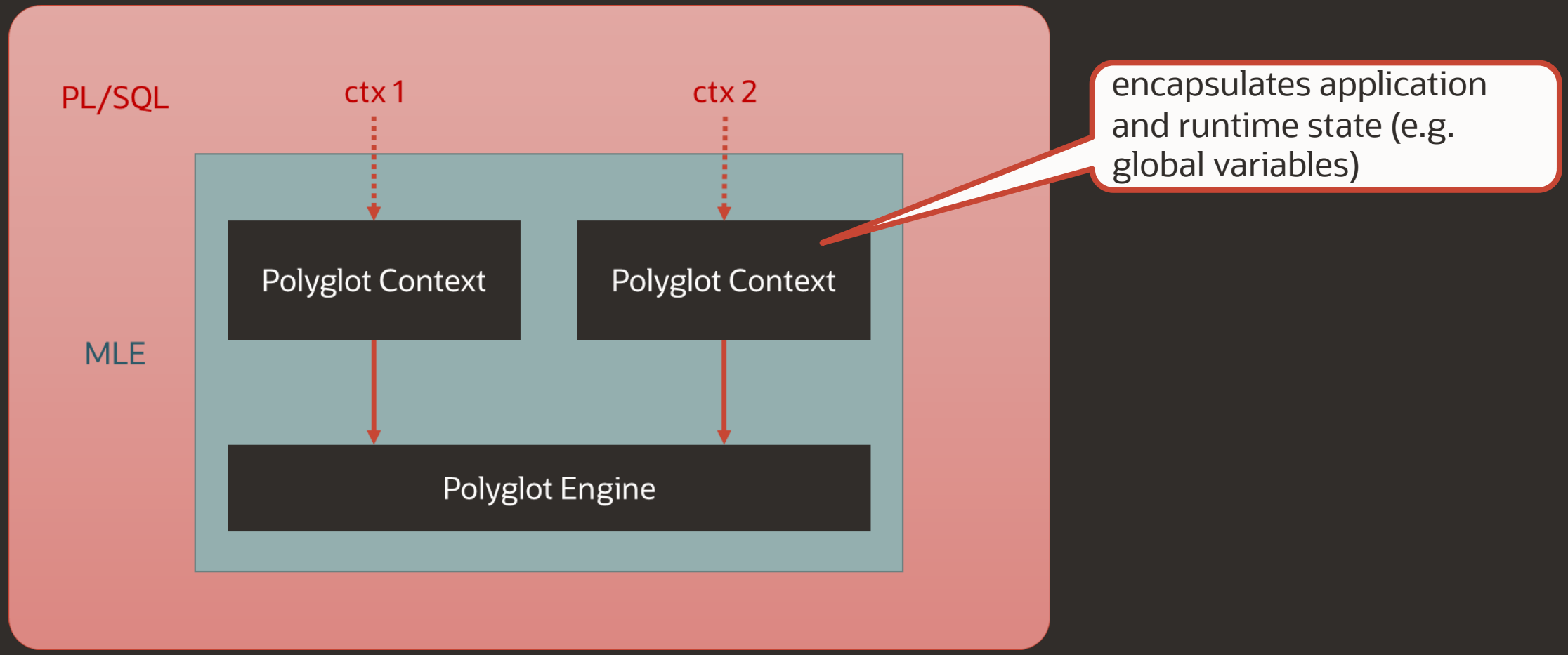
MLE leverages Truffle for Optimizing Across Boundaries



MLE Architecture



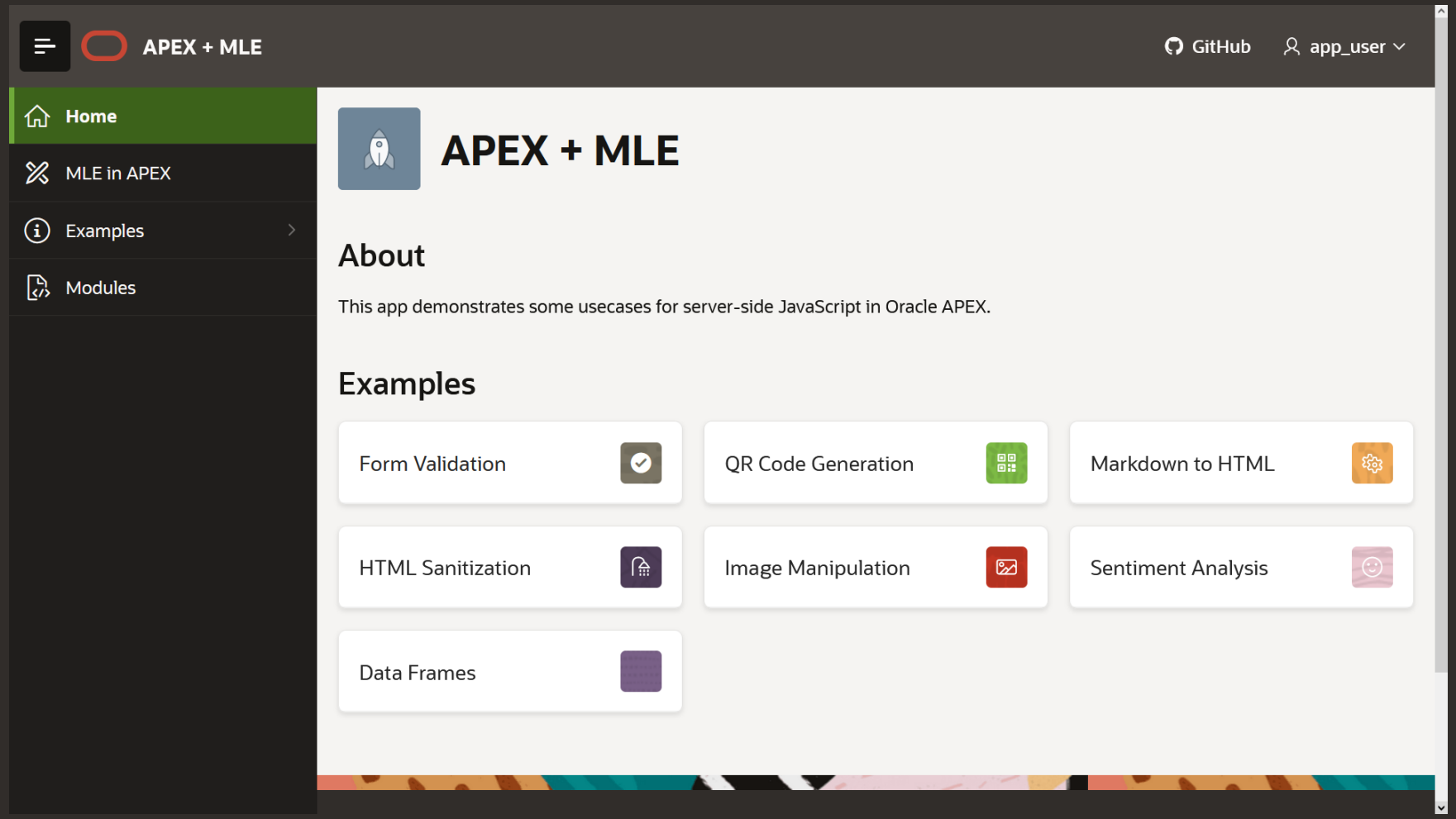
MLE Context Management



Agenda

1. GraalVM
   - GraalVM Compiler & (Partial) Escape Analysis
   - Truffle Framework & Speculative Optimizations
2. Oracle Database Multilingual Engine (MLE)
   - GraalVM in the Database: MLE Vision and Architecture
   - **Demo: APEX + MLE = Low-Code + JavaScript = AWESOME**
   - MLE Roadmap and Future Work
3. A Quick Intro into Oracle Labs + Internships

Demo 1: JavaScript in APEX



Using Server-Side Scripting in APEX

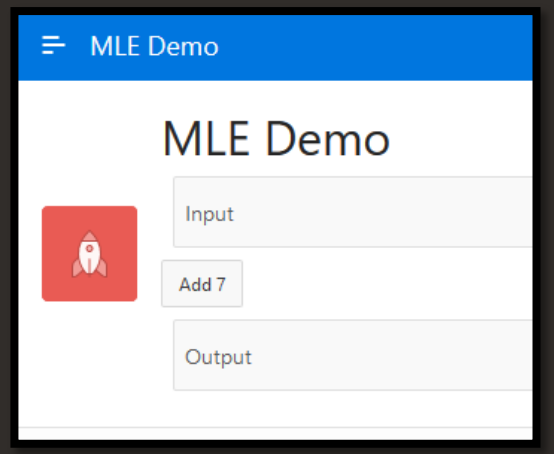
Using DBMS_MLE standalone

```
DECLARE
  ctx dbms_mle.context_handle_t;
  user_code varchar2(1024) := q'~
    let bindings = require("mle-js-bindings");
    let input = bindings.importValue("P1_INPUT");
    bindings.exportValue("P2_OUTPUT", input + 7);
  ~';
BEGIN
  ctx := dbms_mle.create_context();

  dbms_mle.export_to_mle(ctx, 'P1_INPUT', P1_INPUT);
  dbms_mle.eval(ctx, 'JAVASCRIPT', user_code);
  dbms_mle.import_from_mle(ctx, 'P2_OUTPUT', P2_OUTPUT);

  dbms_mle.drop_context(ctx);
END;
/
```

Using DBMS_MLE through APEX




```
apex.env.P2_OUTPUT = (apex.env.P1_INPUT / 1) + 7;
```

Agenda

1. GraalVM
   - GraalVM Compiler & (Partial) Escape Analysis
   - Truffle Framework & Speculative Optimizations
2. Oracle Database Multilingual Engine (MLE)
   - GraalVM in the Database: MLE Vision and Architecture
   - Demo: APEX + MLE = Low-Code + JavaScript = AWESOME DEVELOPMENT
   - **MLE Roadmap and Future Work**
3. A Quick Intro into Oracle Labs + Internships

MLE Roadmap and Future Work

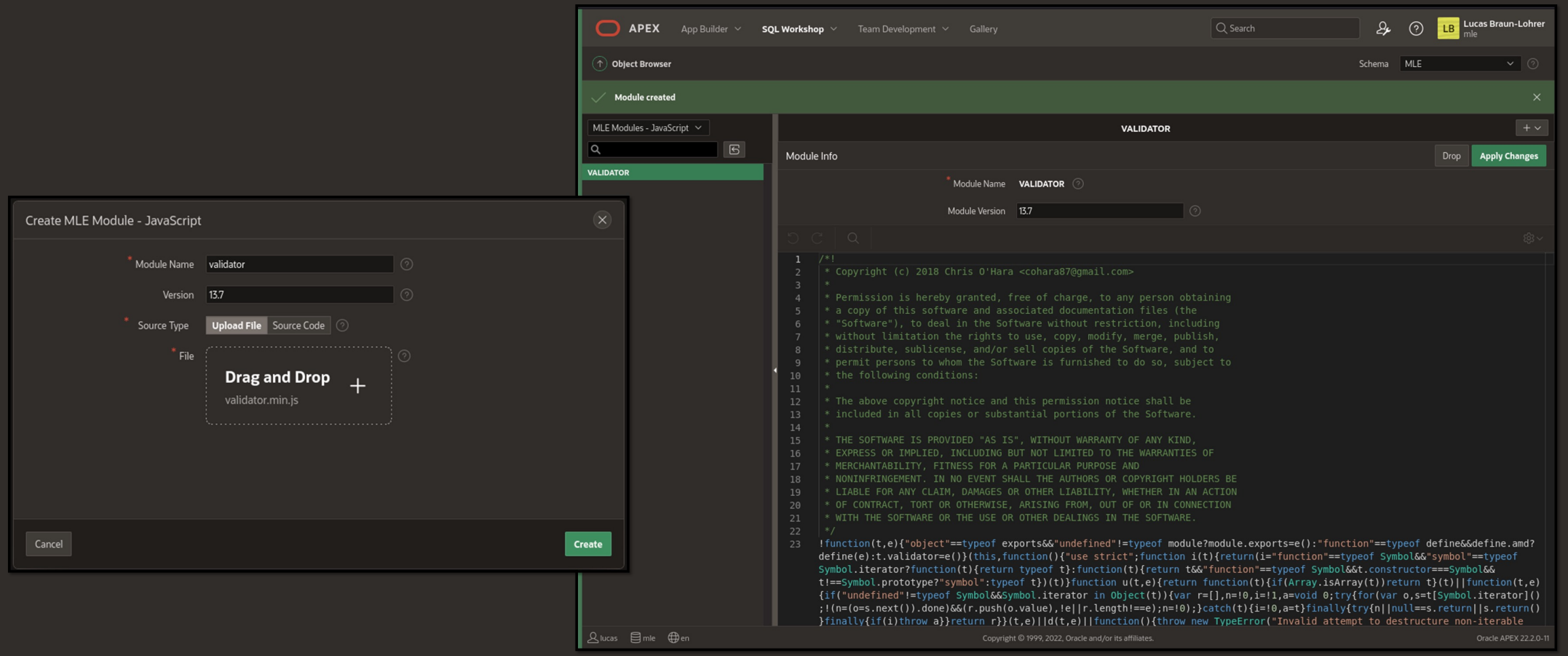
New MLE features in Oracle 23c

- code management (modules & envs)
- function calls
- debugging
- support the JSON SQL type and SODA APIs
- performance improvements
  - compilation in background threads
  - share read-only heap across sessions

Roadmap and Future Work

- support for more modules (and WASM)
  - mage manipulation modules
  - PDF generation modules
- further boost performance of MLE execution
- support more data types (vectors!)
- more languages… ☺

Demo 2: Using JavaScript from Open-Source



Getting Started – Tutorial and Blog Posts

21c

23c

Using the Oracle Cloud for free

Everybody
Oracle Cloud Always-Free Tier: oracle.com/cloud/free/

Universities and Schools
Oracle Academy: academy.oracle.com

In Summary

GraalVM

- GraalVM is a multilingual JIT compiler and runtime
- GraalVM allows writing your own language with the Truffle API
- GraalVM achieves high performance execution by speculative (and other) optimizations

Multilingual Engine

- GraalVM in Oracle Database
- Employs Truffle languages, language interop, native image AOT compilation
- Enables Low-Code rapid application development with JavaScript
- Allows adopting open-source code in Oracle Database

Agenda

1. GraalVM
   - GraalVM Compiler & (Partial) Escape Analysis
   - Truffle Framework & Speculative Optimizations
2. Oracle Database Multilingual Engine (MLE)
   - GraalVM in the Database: MLE Vision and Architecture
   - Demo: APEX + MLE = Low-Code + JavaScript = AWESOME DEVELOPMENT
   - MLE Roadmap and Future Work
3. **A Quick Intro into Oracle Labs + Internships**

Oracle Labs' Four Approaches to a Balanced Research Portfolio

Exploratory Research

- Pursue new ideas within domains relevant to Oracle

GraalVM™

Directed Research

- In collaboration with product teams
- Difficult, future-looking problems
- Driven by product requirements

ORACLE Database Multilingual Engine

Consulting

- Provide unique expertise
- Small engagement across product organizations

Product Incubation

- Grow new products from Oracle Labs research

A global research team

Hundreds of researchers worldwide

Zurich: The biggest location with two floors at the Prime Tower (and growing!)

The geographic spread allows Oracle Labs to take advantage of a **tremendous pool of scientific and engineering talent** and enables Labs researchers to **collaborate with colleagues** from a **wide range of industries and universities**.

Oracle Labs locations

- Zurich, Switzerland
- Prague & Brno, Czech Republic
- Casablanca, Morocco
- Linz, Austria
- Redwood Shores, California, USA
- Austin, Texas, USA
- Belgrade, Serbia
- Brisbane, Australia
- … and more!

Selection of projects with involvement of the Zurich Lab

- **Parallel Graph AnalytiX (PGX)** – High-performance graph toolkit (single machine, distributed, in DB)
- **Data Studio (DS)** – Notebook technology for visualizing graphs and more
- **GraalVM** – A universal, polyglot VM environment
- **MultiLingual Engine (MLE)** – Bringing modern languages into the Oracle DB
- **Oracle Labs Apps (Apps)** – designing, building and operating apps using the principles of modern app development
- **Application Dependency Management (ADM) and App Platform** – develop an app platform built around Oracle's opinionated view of how cloud-native apps should be architected

Several other topics across the other offices

- ML / AI applications, code analysis and security, concurrent programming, …

If you are interested in Computer or Data Science, we have a great topic for you!

Check them out on labs.oracle.com/pls/apex/labs/r/labs/internships and reach out to labs-hiring_ww@oracle.com.

What former interns say...

I initially joined Oracle Labs for a short internship where I was working on a distributed graph processing engine. I **designed and implemented major components** for the system in collaboration with well established Oracle Labs members and got the opportunity to **learn from very skilled people**. While I did enjoy the task, the highlight for me were the people. They were very **welcoming and helpful from the beginning** to the end of the internship. I ended up extending my internship and accepting a full time offer afterwards.

Irfan Bunjaku
ETH student, 6-month intern + MSc thesis with Oracle Labs in 2022

Internships at Oracle Labs Zurich*

Regular internships or MSc theses

Typical duration of 3 to 12 months

Competitive salary

Apply on
labs.oracle.com/pls/apex/labs/r/labs/internships
and/or contact us via labs-hiring_ww@oracle.com!

In Summary

GraalVM

- GraalVM is a multilingual JIT compiler and runtime
- GraalVM allows writing your own language with the Truffle API
- GraalVM achieves high performance execution by speculative (and other) optimizations

Multilingual Engine

- GraalVM in Oracle Database
- Employs Truffle languages, language interop, native image AOT compilation
- Enables Low-Code rapid application development with JavaScript
- Allows adopting open-source code in Oracle Database

Oracle Labs and Internships

- If you are interested in Computer or Data Science, we have a great topic for you!

GraalVM Compiler

Multilingual Engine

Free Cloud & Oracle Academy

Jobs at Oracle

Thank you

Have a look at out our internship topics in the VIS Job Emails – we'd love to get your application.

Questions?

ORACLE