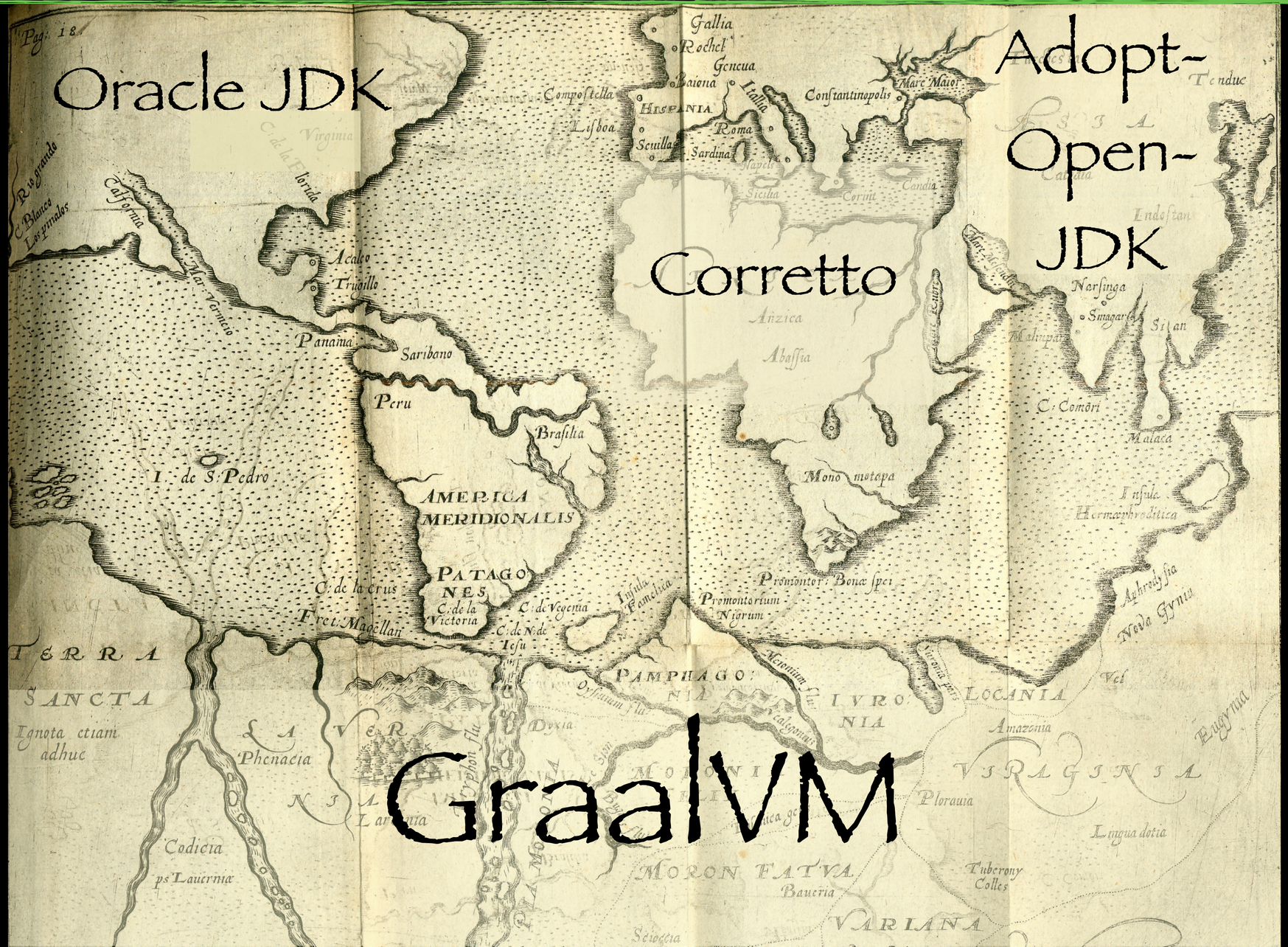
GraalVM

Stephan Rauh

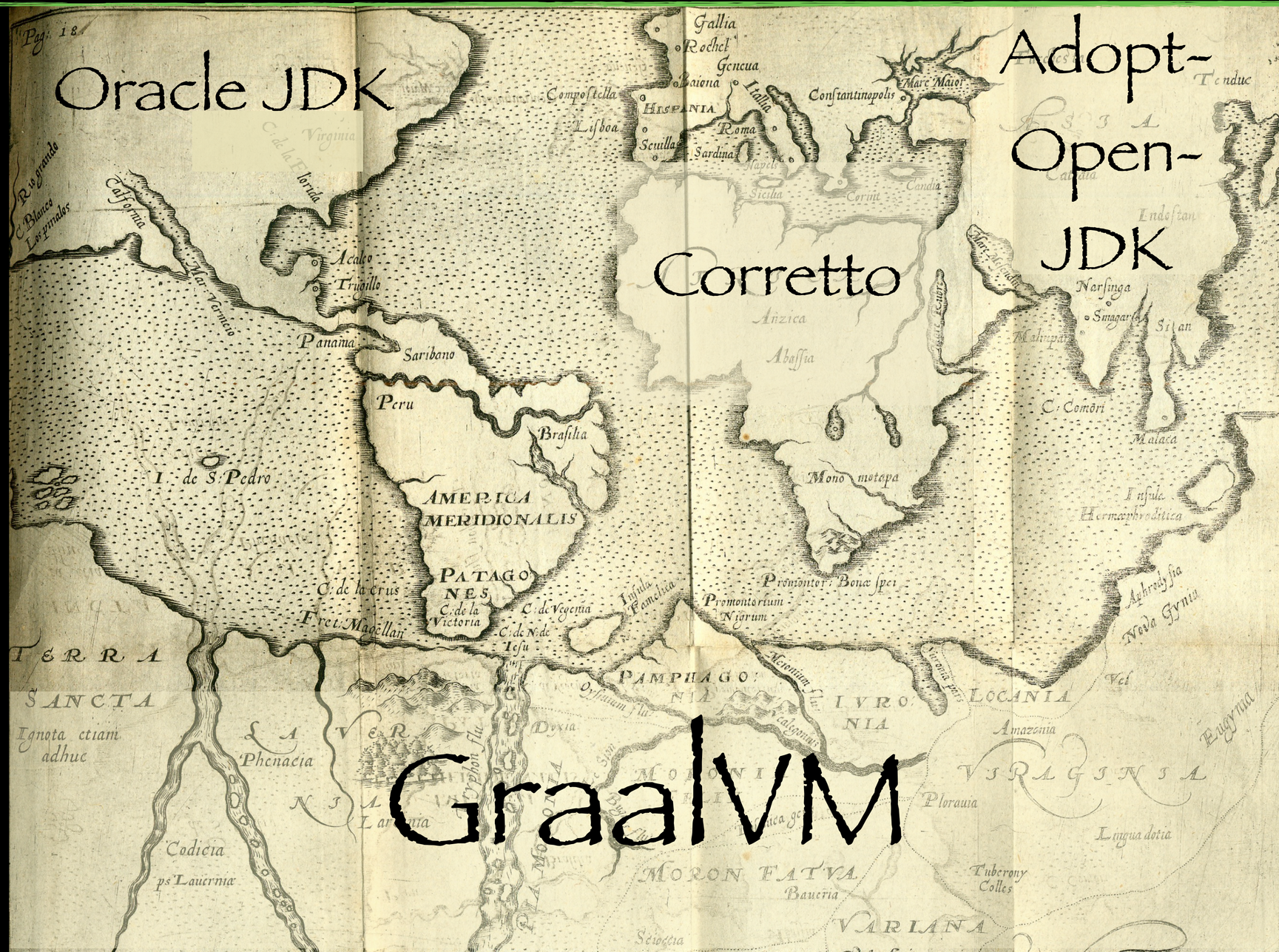
December 2020

What is GraalVM?

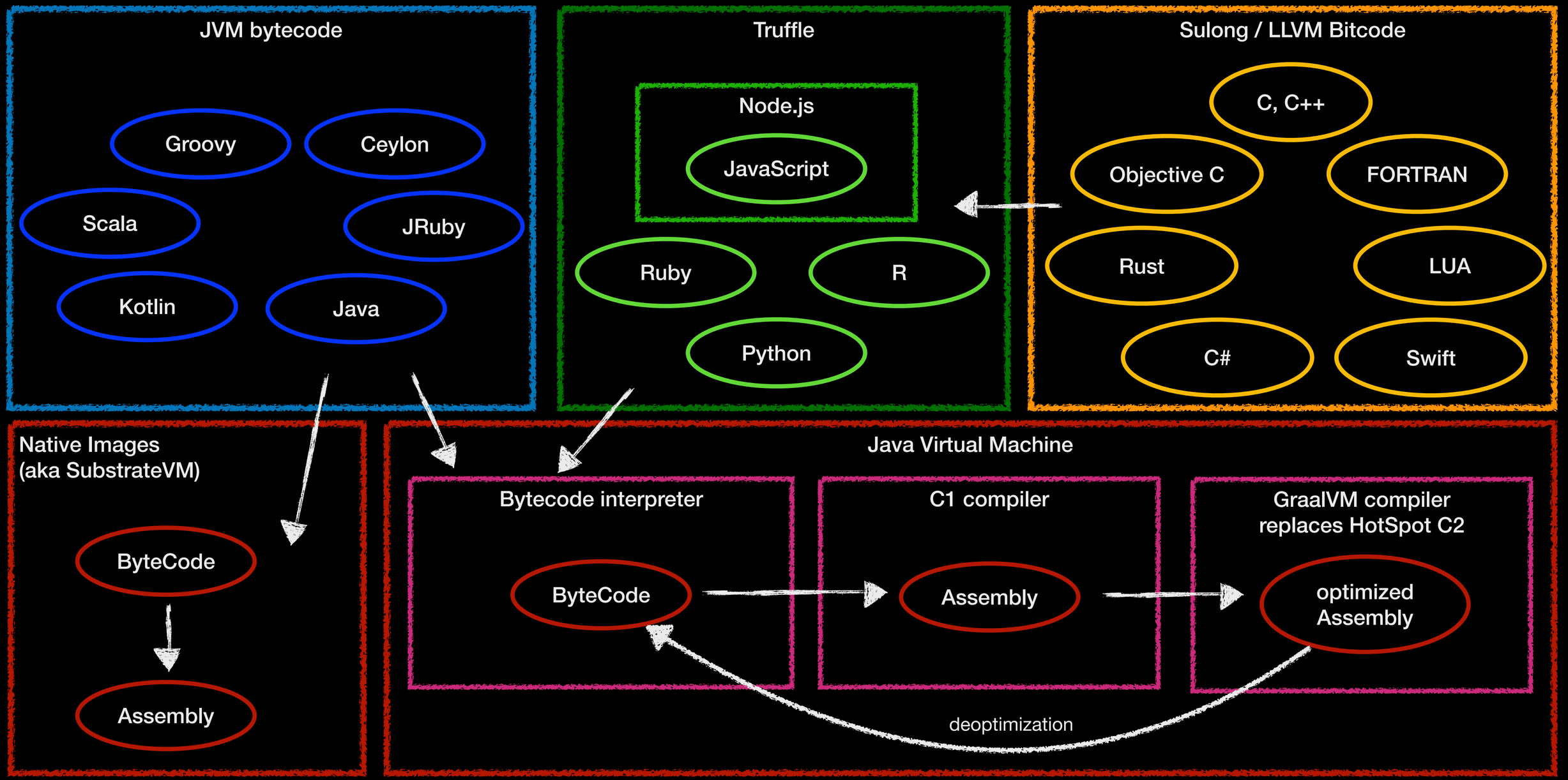


What is GraalVM?

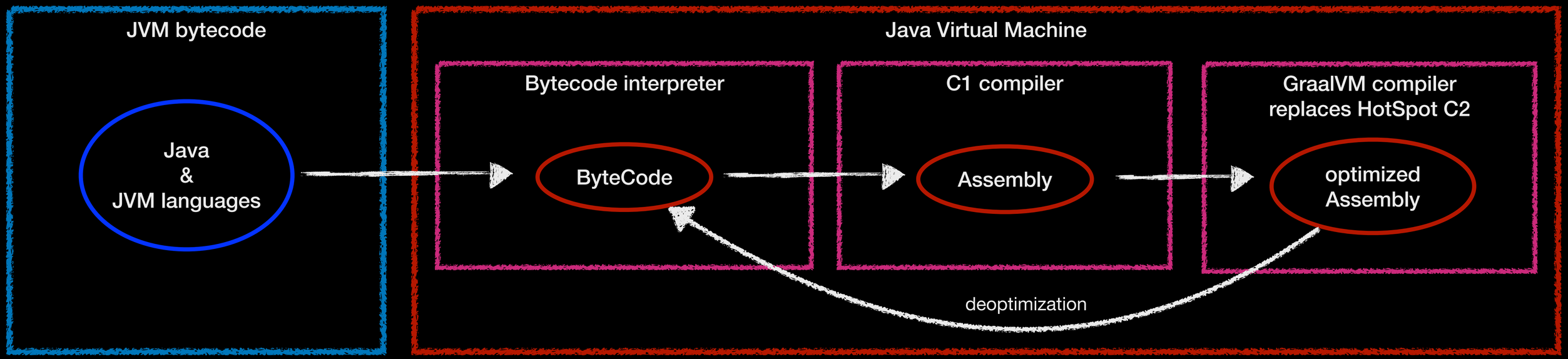
- All-new compiler
- Polyglot programming
- AOT compiler
- LLVM runtime
- Web Assembly



GraalVM at a Glance



All-New Compiler



- New compiler, written in Java
- Plug-in replacement to the OpenJDK compiler
- Getting rid of the weight of 20+ years of development
- Support new programming paradigms from day one

New & Updated Programming Paradigms

Just a few changes of the last couple of decades:

- Functional programming
  - Replacing for loops by chains of Lambdas
- Test automation
  - Megamorphism only during tests
- Dawn of the application servers
- Rise of Kotlin and Scala (among others)

New Optimizations for a New Compiler

- Aggressive inlining
- Partial Escape Analysis
- Data Flow Analysis
- Inter-Procedural Optimization
- Eliminating the price of megamorphism
- Functional programming (almost) for free
- and 60+ more

Partial Escape Analysis

- Todo: code example
- Ideas:
  - Lock elision
  - eliminating global variables
  - eliminating variables altogether

Loop Unrolling

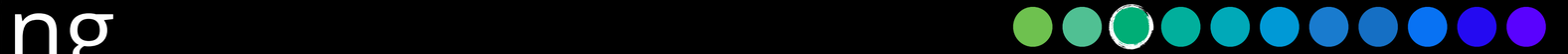
```
var result = 0;
for (int i = 0; i < 5; i++) {
    result += i;
}
```

is equivalent to

```
var result = 0;
result += 0;
result += 1;
result += 2;
result += 3;
result += 4;
```

but the latter is a lot faster!

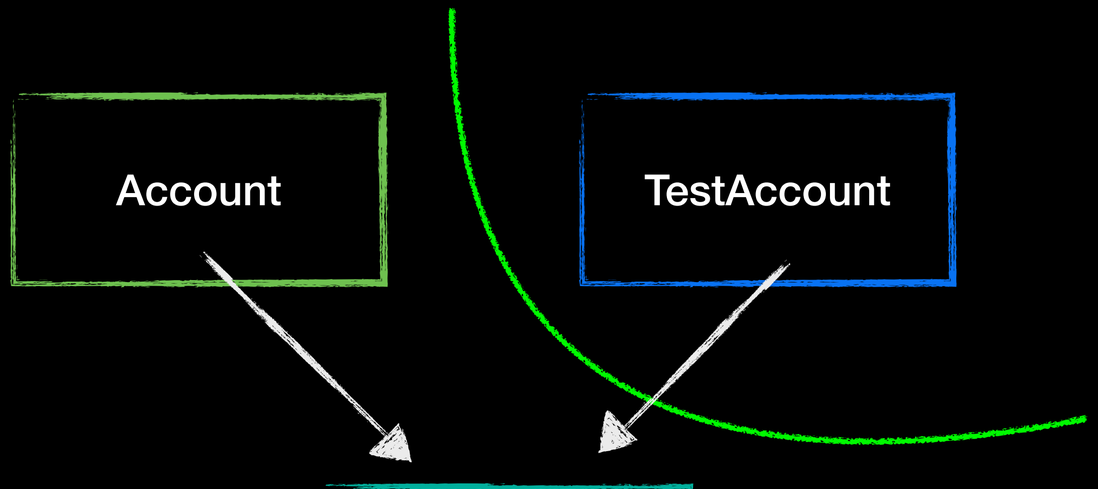
Broken Loop Unrolling



```
var result =
    IntStream.range(1, 5)
             .reduce(0,
              (x, y) -> x + y);
```

can't be unrolled (yet)

Speculative Optimization



- InvokeVirtual is expensive
- But why bother? You ain't gonna use it!

Speculative Optimization

Speculative Optimization

```
class Counter1 implements Counter {
  private int x;
  public int inc() {
    return x++;
  }
}
```

```
class Counter2 implements Counter {
  private int x;
  public int inc() {
    return x++;
  }
}
```

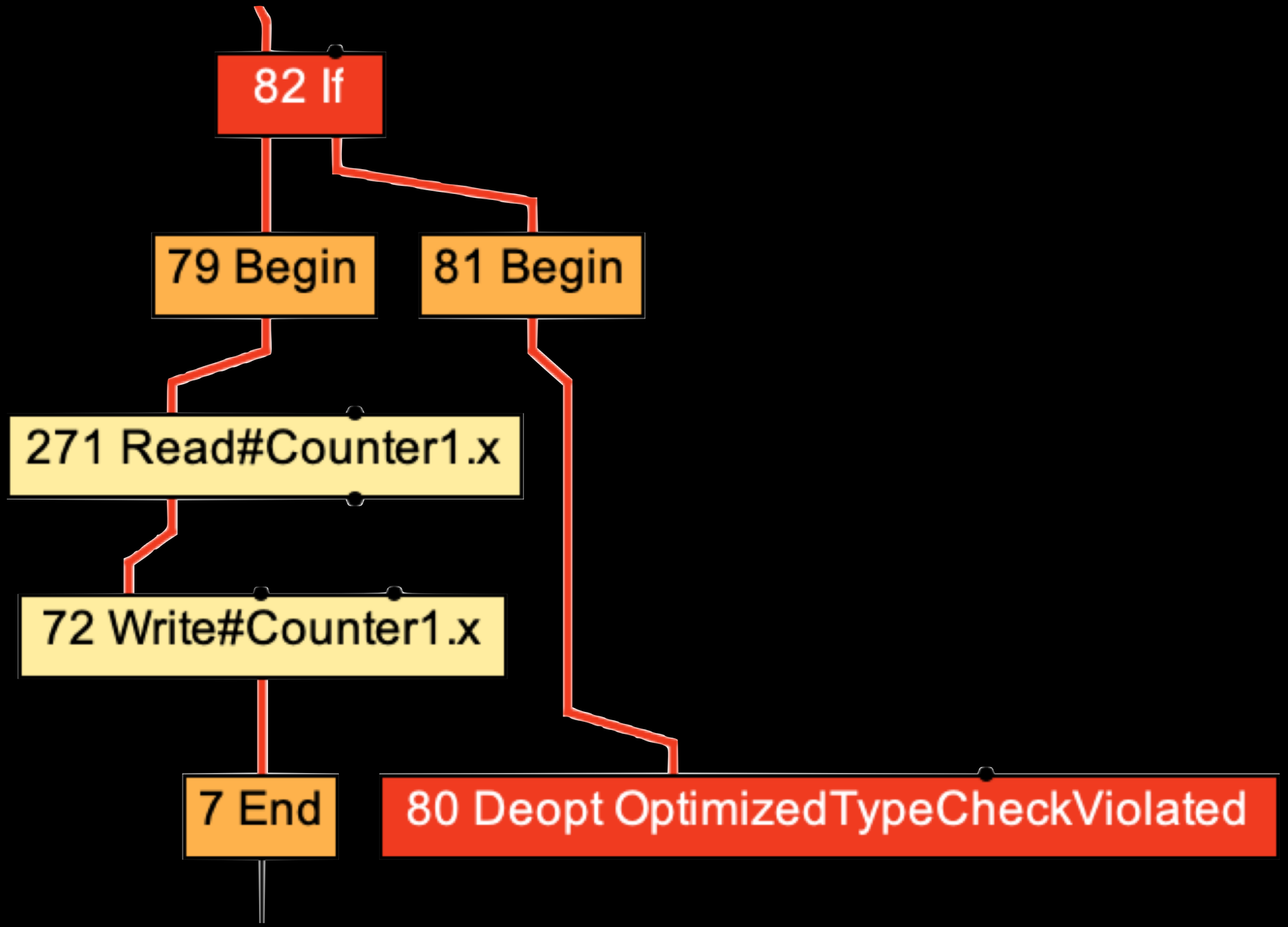
```
class Counter3 implements Counter {
  private int x;
  public int inc() {
    return x++;
  }
}
```

```
interface Counter {
  int inc();
}
```

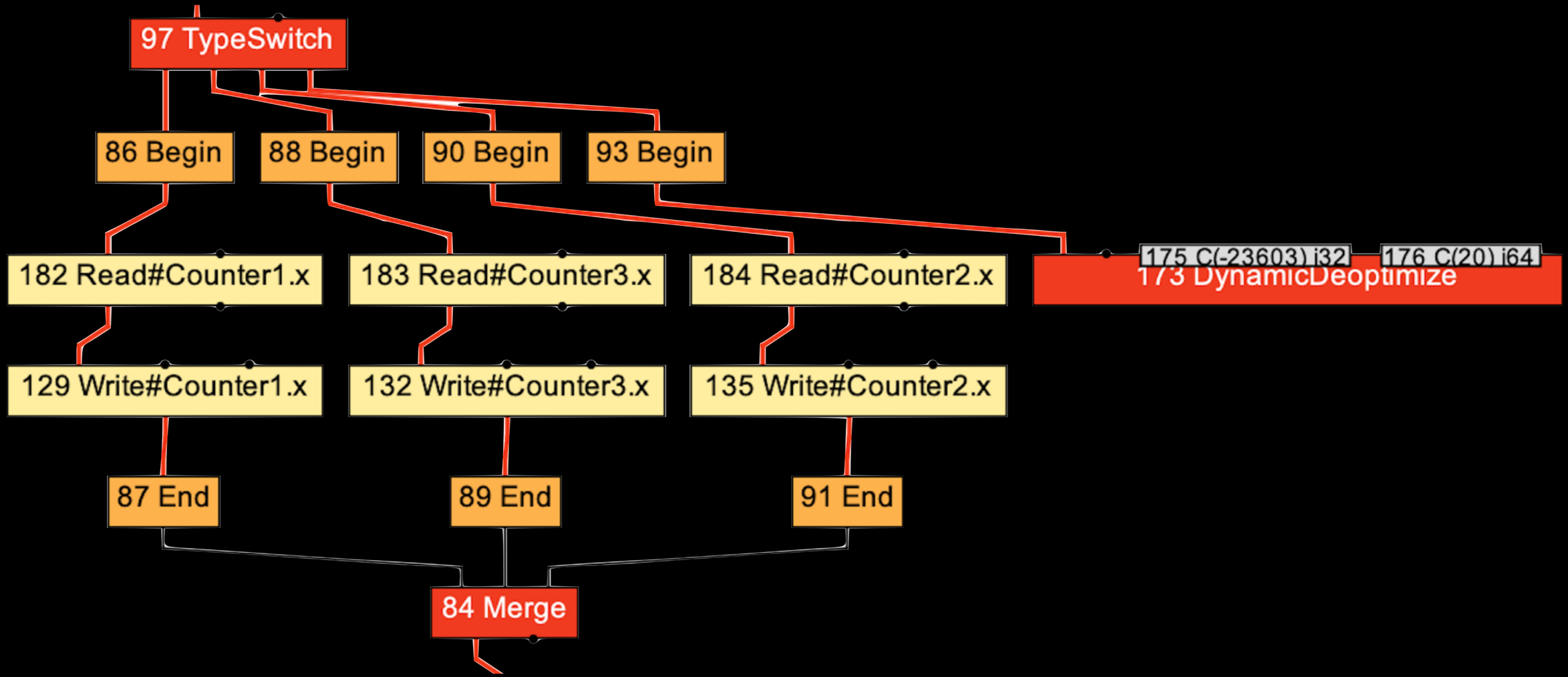
```
measure(new Counter1());
measure(new Counter2());
measure(new Counter3());
```

```
public void measure(Counter c) {
  c.inc();
}
```

Speculative Optimization

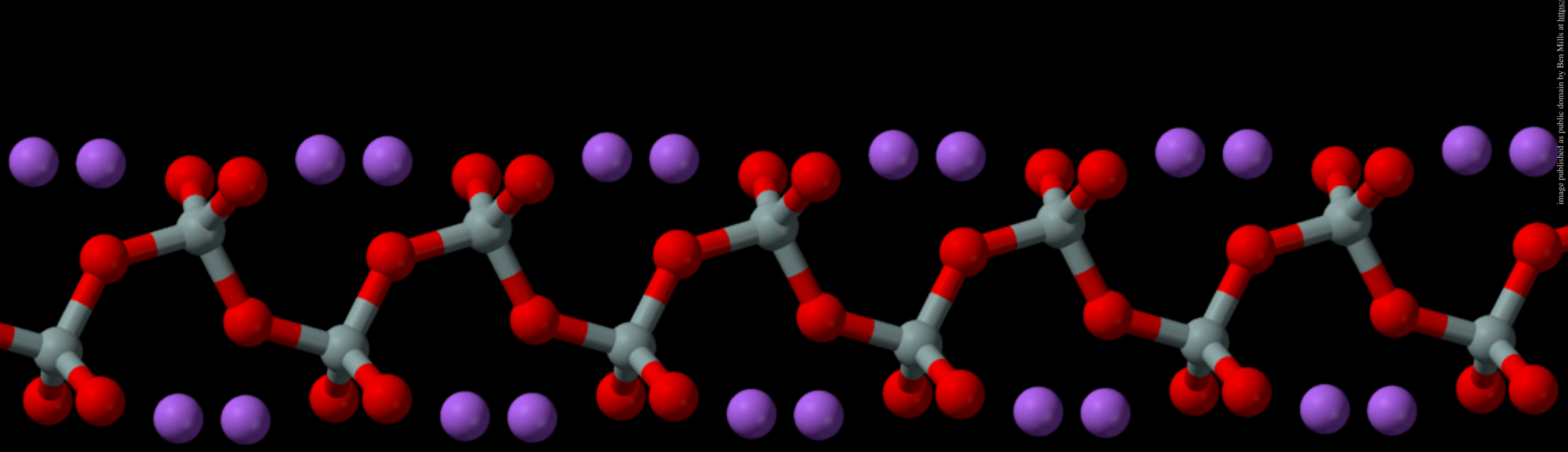


Speculative Optimization

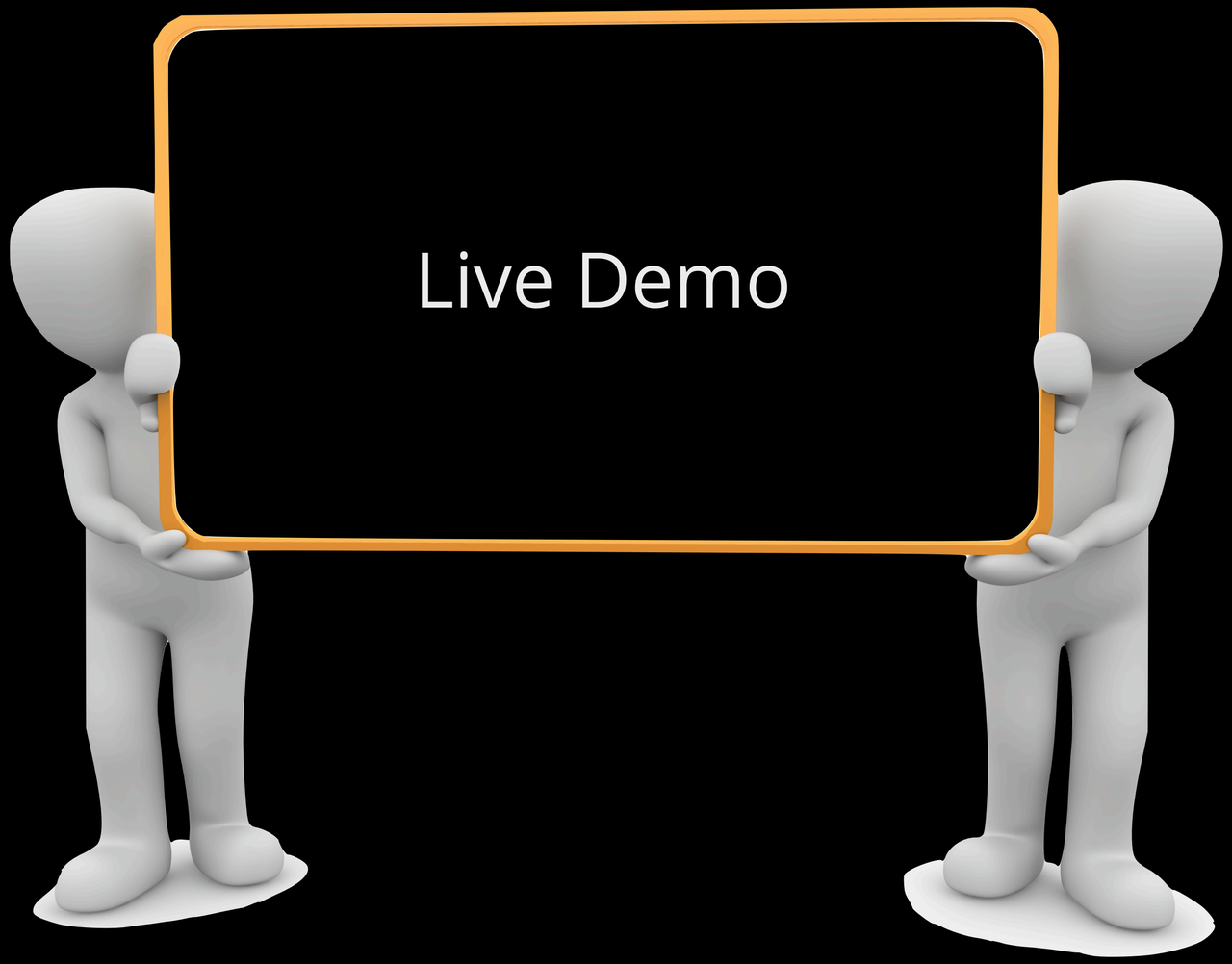


Unchaining Chains of Lambdas

```
File[] files = Arrays.stream(
    Optional.ofNullable(this.file.listFiles())
    .orElse(new File[0]))
    .sorted(Comparator.comparing(File::isDirectory).reversed()
    .thenComparing(File::getName))
    .toArray(File[]::new);
```



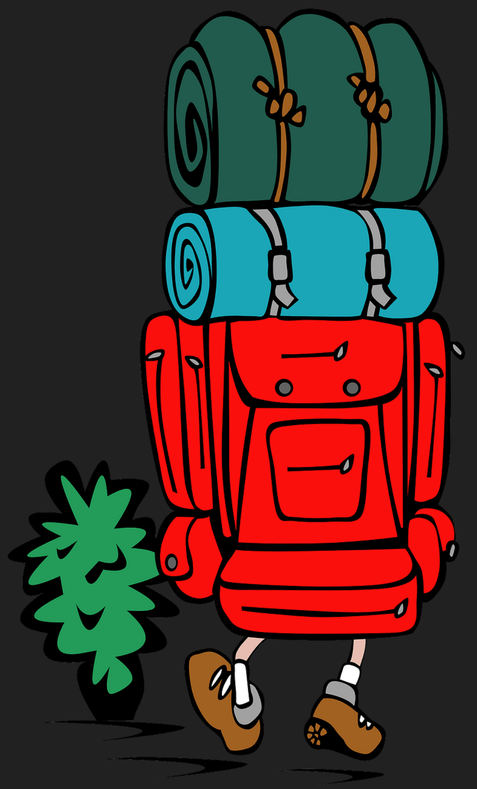
Unchaining Chains of Lambdas



Type-Based Optimization

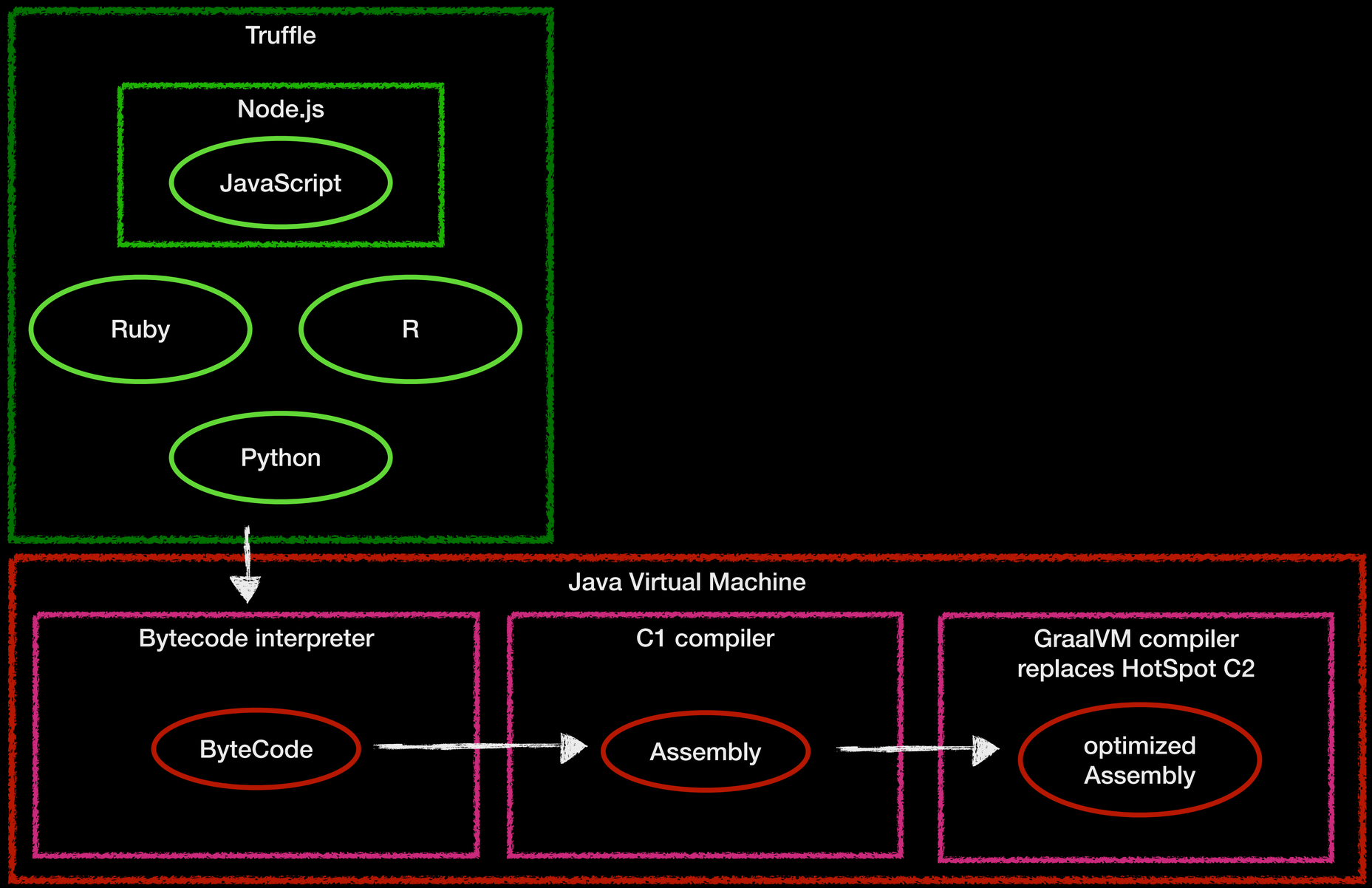
```
// https://medium.com/graalvm/stream-api-performance-with-graalvm-be6cfe7fbb52

public double averageAge(people: Person[]) {
  return Arrays.stream(people)
    .filter(p -> p.age >= 18 && p.age <= 21)
    .mapToInt(p -> p.age)
    .average()
    .getAsDouble();
}

class Person {
  public final int age;
}

/** this is an excerpt of a JVM class */
public class OptionalDouble {
  public double getAsDouble() {
    if (!isPresent) {
      throw new NoSuchElementException("No value present");
    }
    return value;
  }
}
```

Does it Pay to Use the New Compiler?

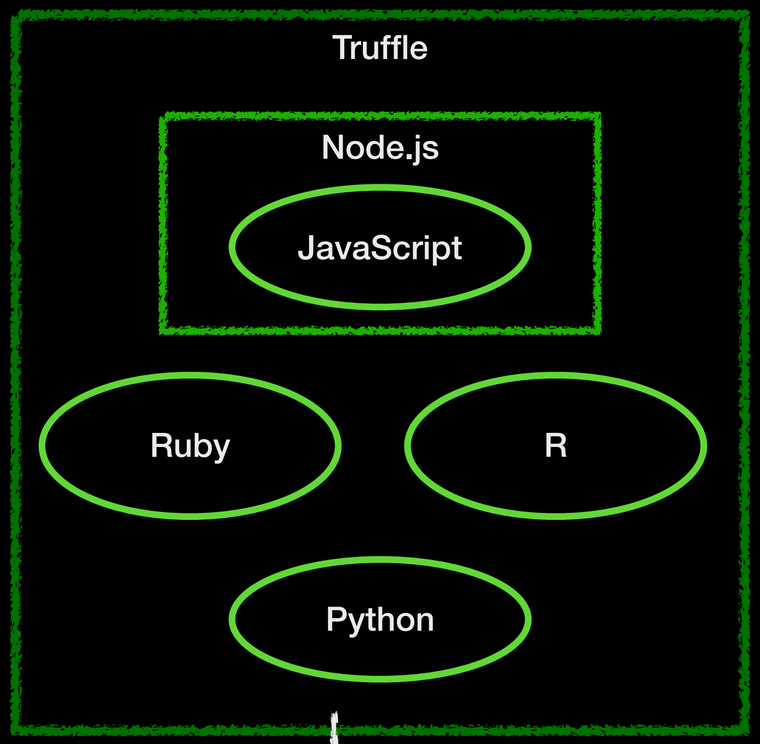


- GraalVM tends to cope better with modern language patterns than OpenJDK
- GraalVM often runs non-Java JVM languages faster
- It's easier to add improvements to GraalVM than to OpenJDK
  → there's more to come!
  → and you can contribute, too!

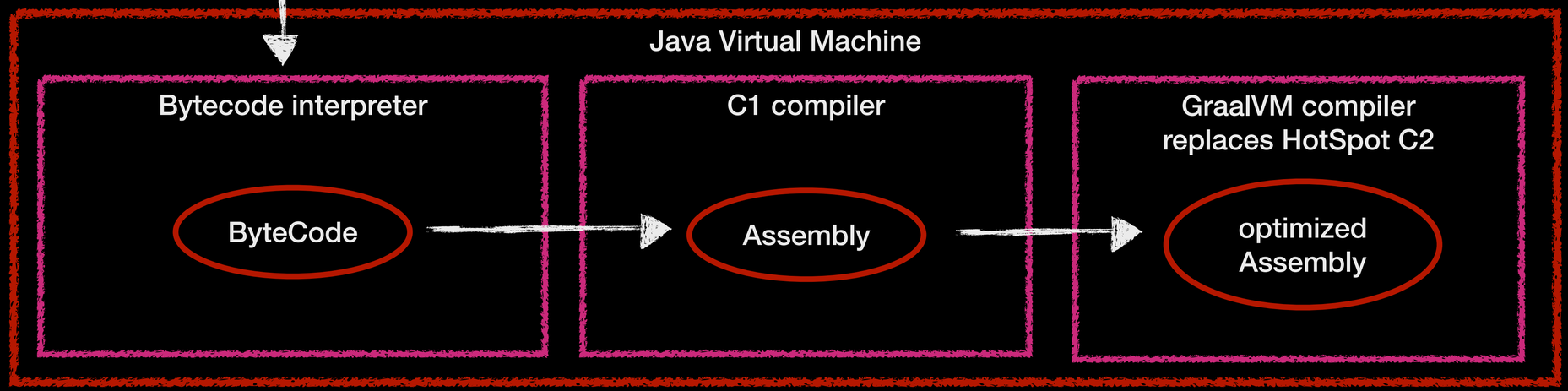
Polyglot Programming with Truffle



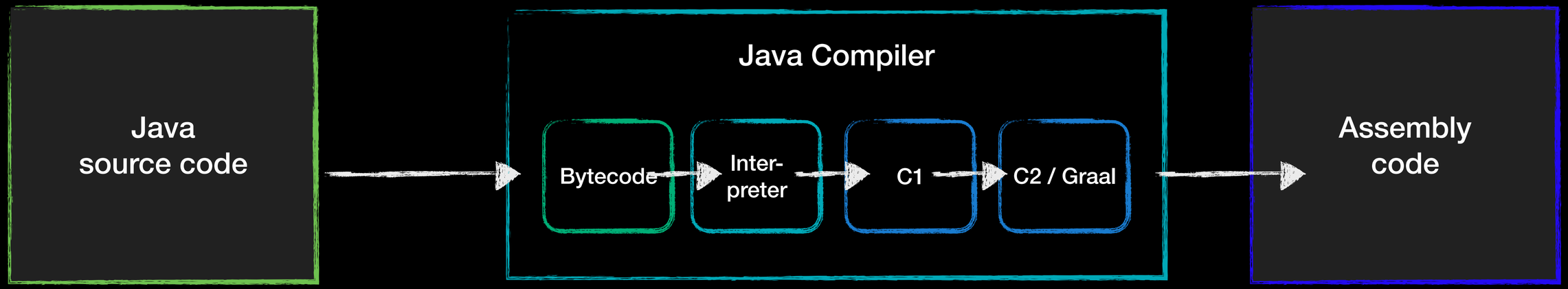
Polyglot Programming with Truffle



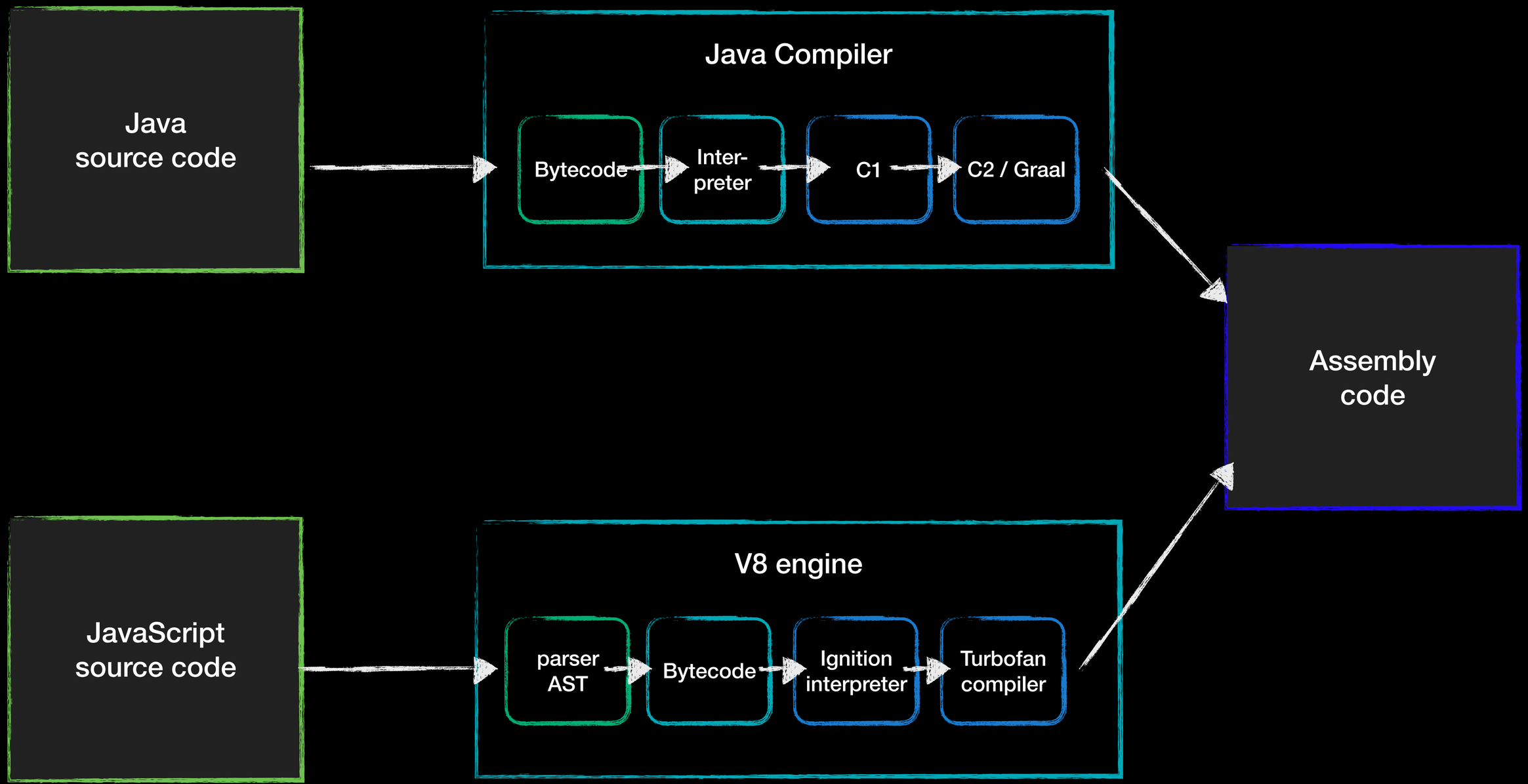
- GraalVM ships with four languages
- You can easily build your own language
  - interesting for academic purposes
  - potential source of future innovation



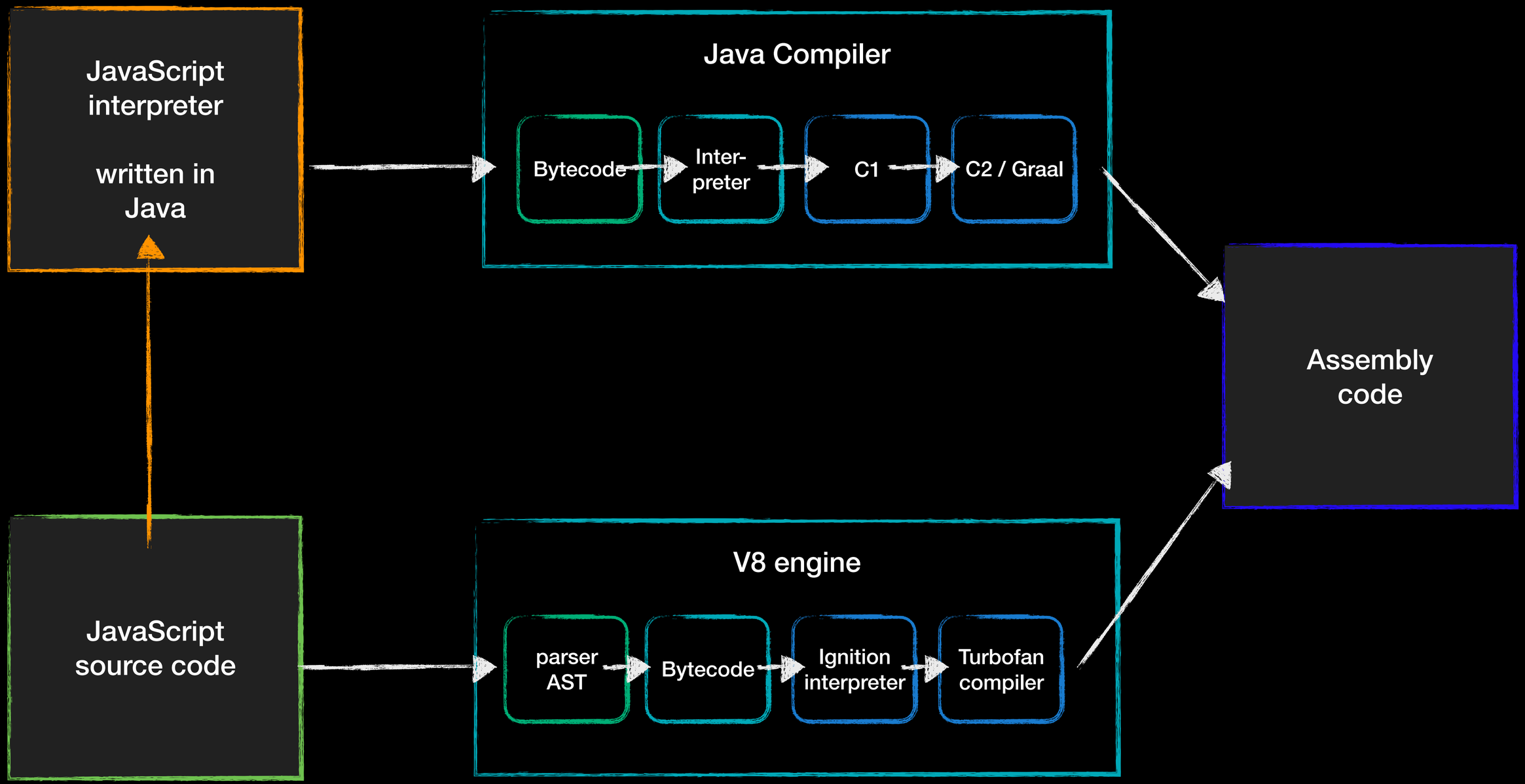
Futamura Projection (How Truffle Works)



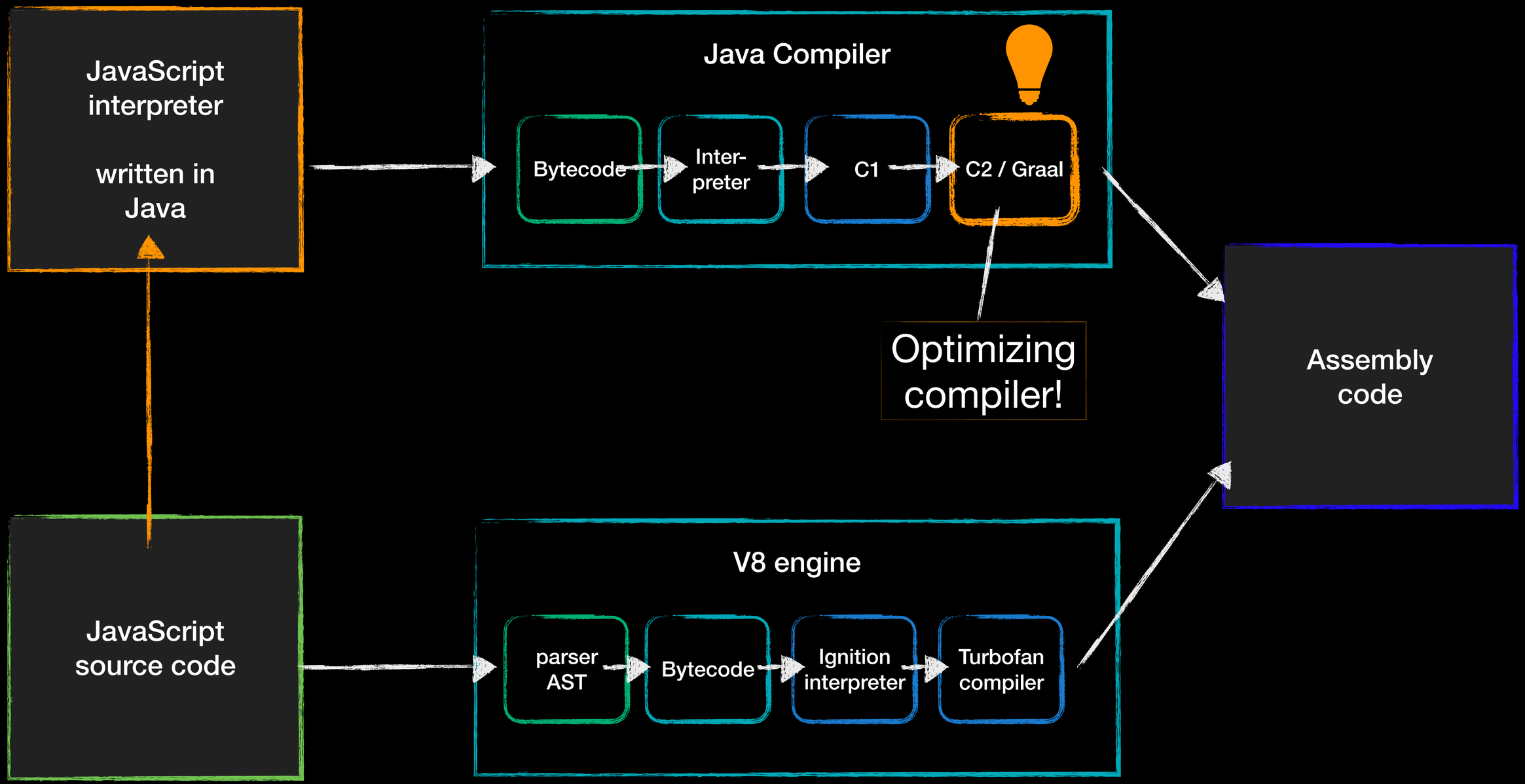
Futamura Projection (How Truffle Works)



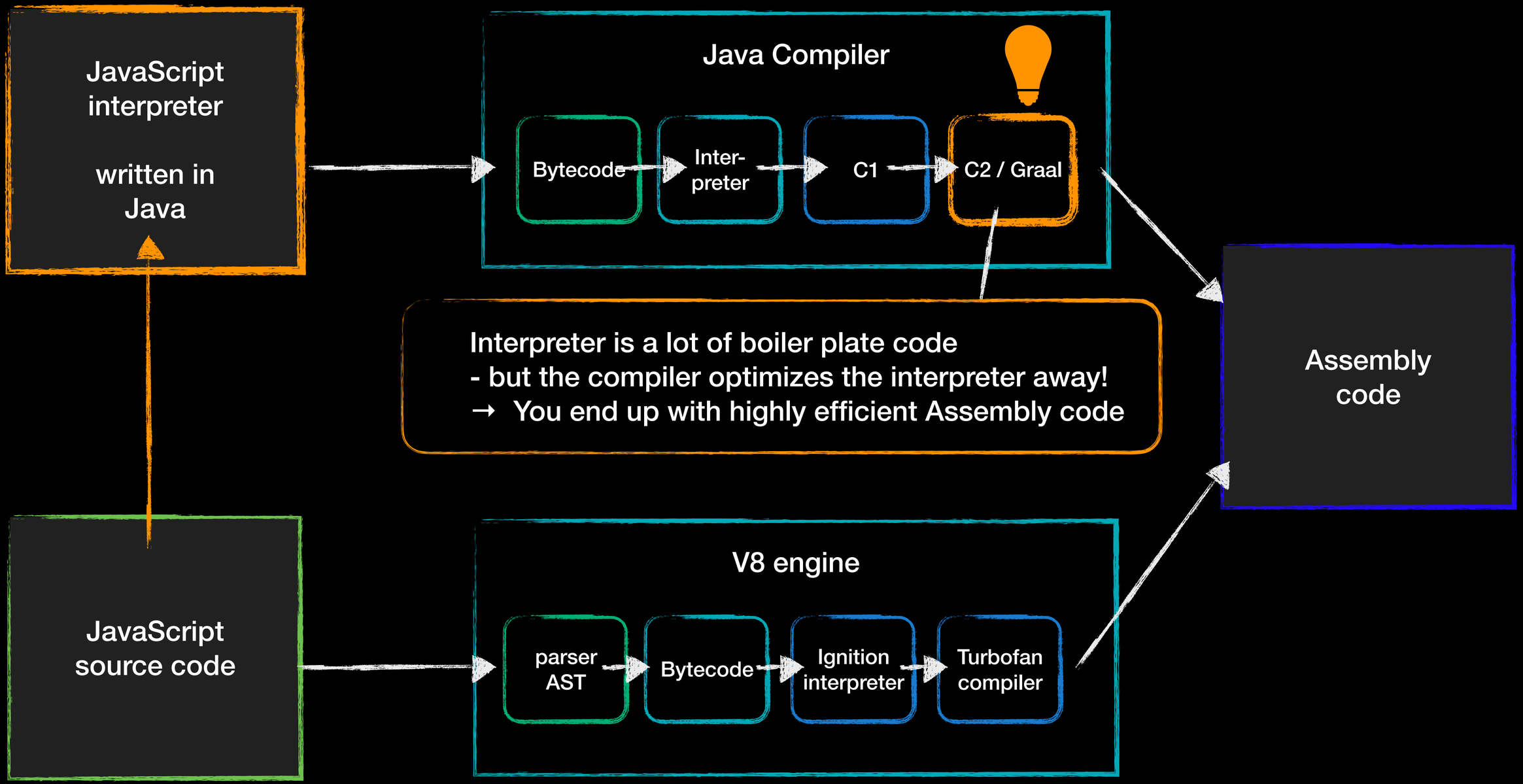
Futamura Projection (How Truffle Works)



Futamura Projection (How Truffle Works)



Futamura Projection (How Truffle Works)



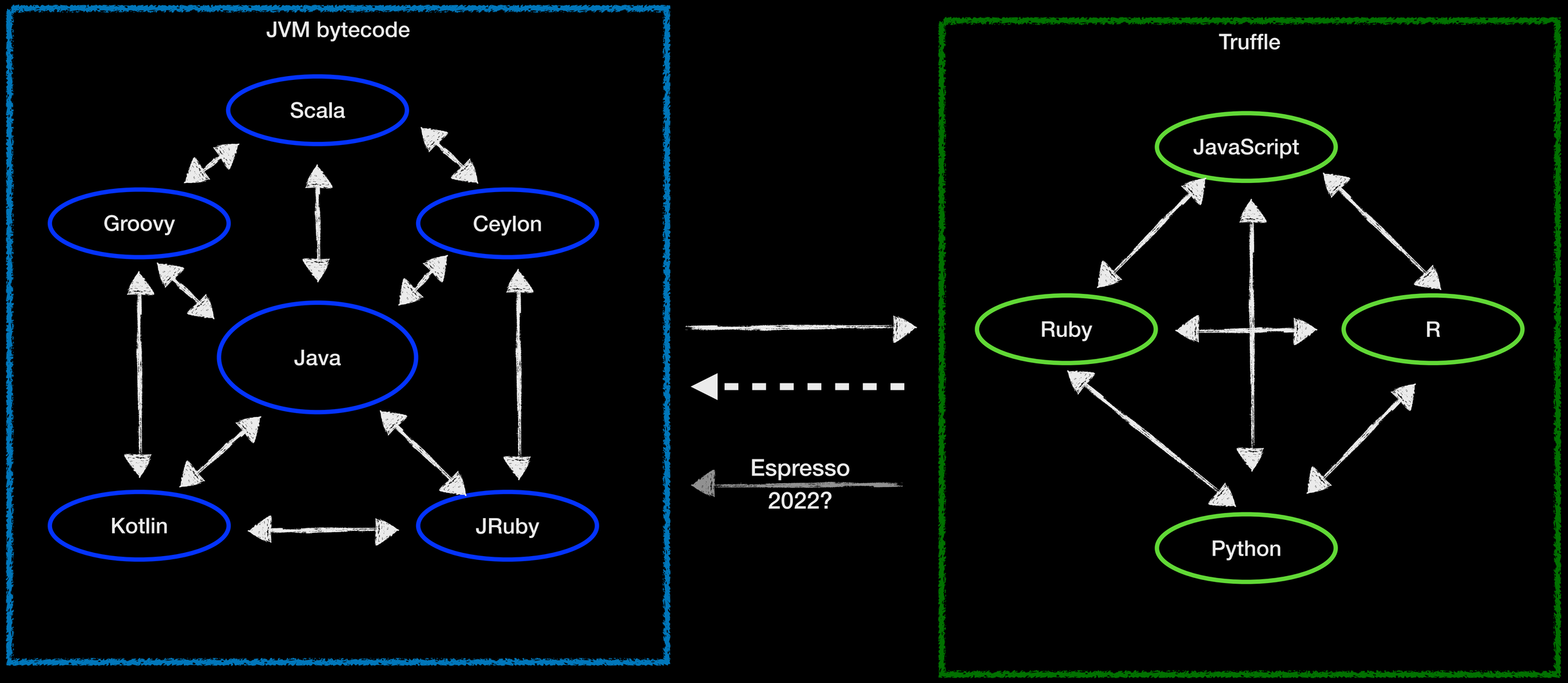
Polyglot Programming with Truffle

- JavaScript:
  - way faster than Nashorn and Rhino
  - severe cold start penalty
- Node.js
  - supported
  - even runs the Angular CLI!
  - but suffers from cold start penalty

Polyglot Programming with Truffle

- Ruby:
  - very good peak performance
  - interesting alternative to the native Ruby interpreters
  - Give it a try!

Is it Truly Polyglot?



Truffle API

register Java code with Truffle (i.e. language-specific API)

LLVM - Project Sulong

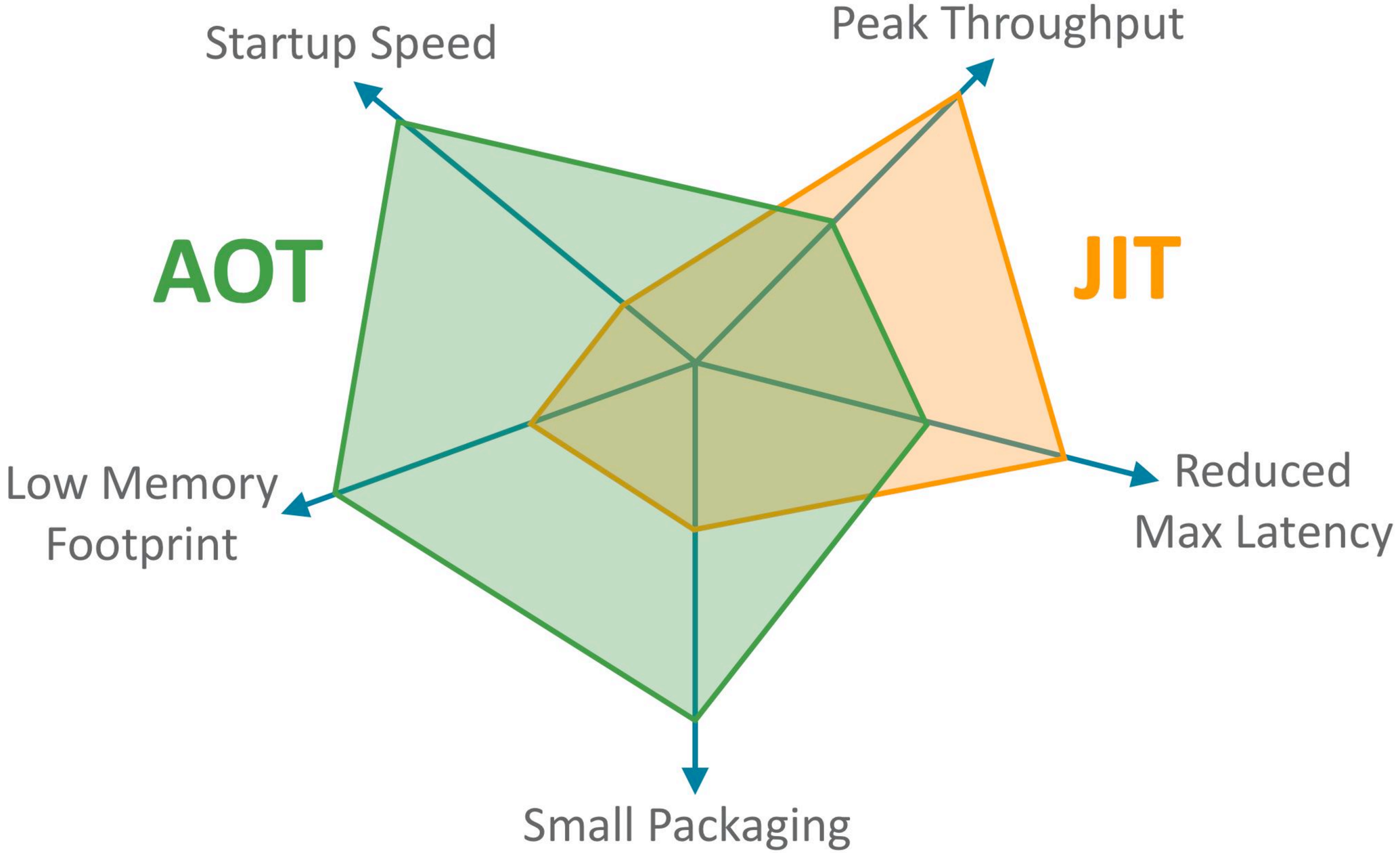
Image source: LLVM Twitter logo https://twitter.com/llvmorg/photo

Native Images (aka AOT, aka SubstrateVM)

- GraalVM ships with an ahead-of-time (AOT) compiler
- Many predecessors, such as Jikes
- AOT compiler were long since dead

What's changed?

Native Images (aka AOT, aka SubstrateVM)



Using Java in AWS Lambda Functions

- Traditionally Java suffers from a slow start
- Great progress recently
- but still, node.js starts way faster
- GraalVM AOT to the rescue!

Reflection and AOT

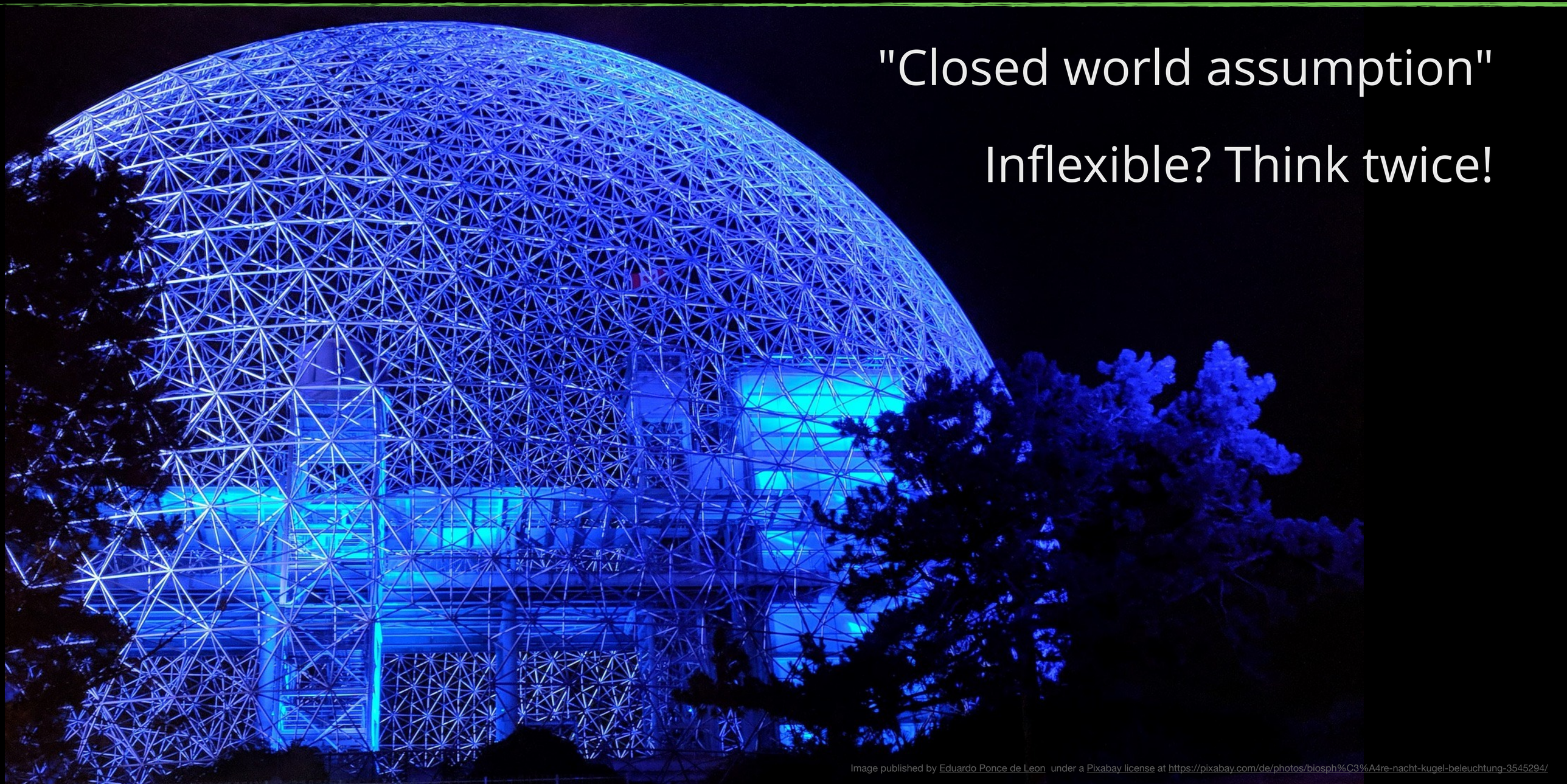
AOT eliminates
info about
classes & methods

→ reflection is broken

→ configure
a configuration file
for reflection

"Closed world
assumption"

Reflection and AOT



"Closed world assumption"

Inflexible? Think twice!

Android, iOS, and Embedded Devices

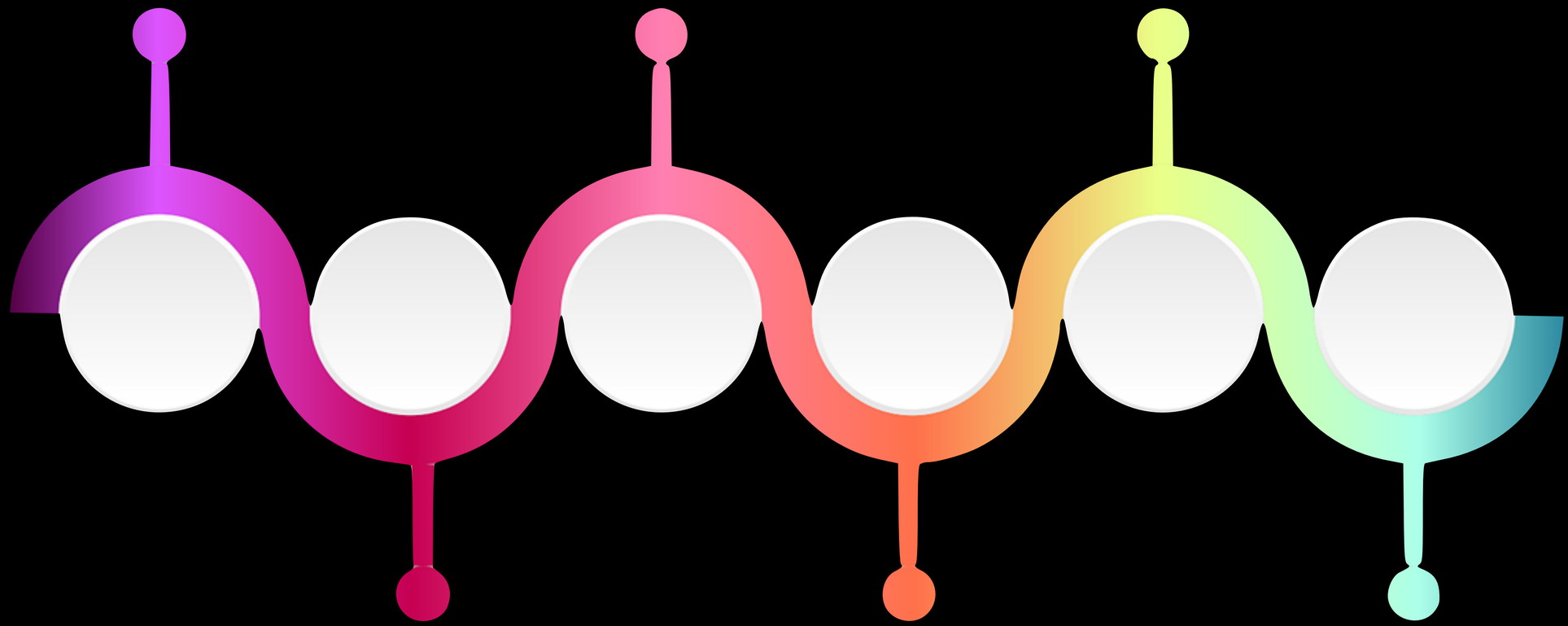
AOT makes many things possible:

- Mobile apps start fast
- no need to bundle the JRE
- JavaFX on Android and iOS w/out the pain
- Java on embedded devices
  - predictable speed, low memory footprint
- Utilitities like babashka

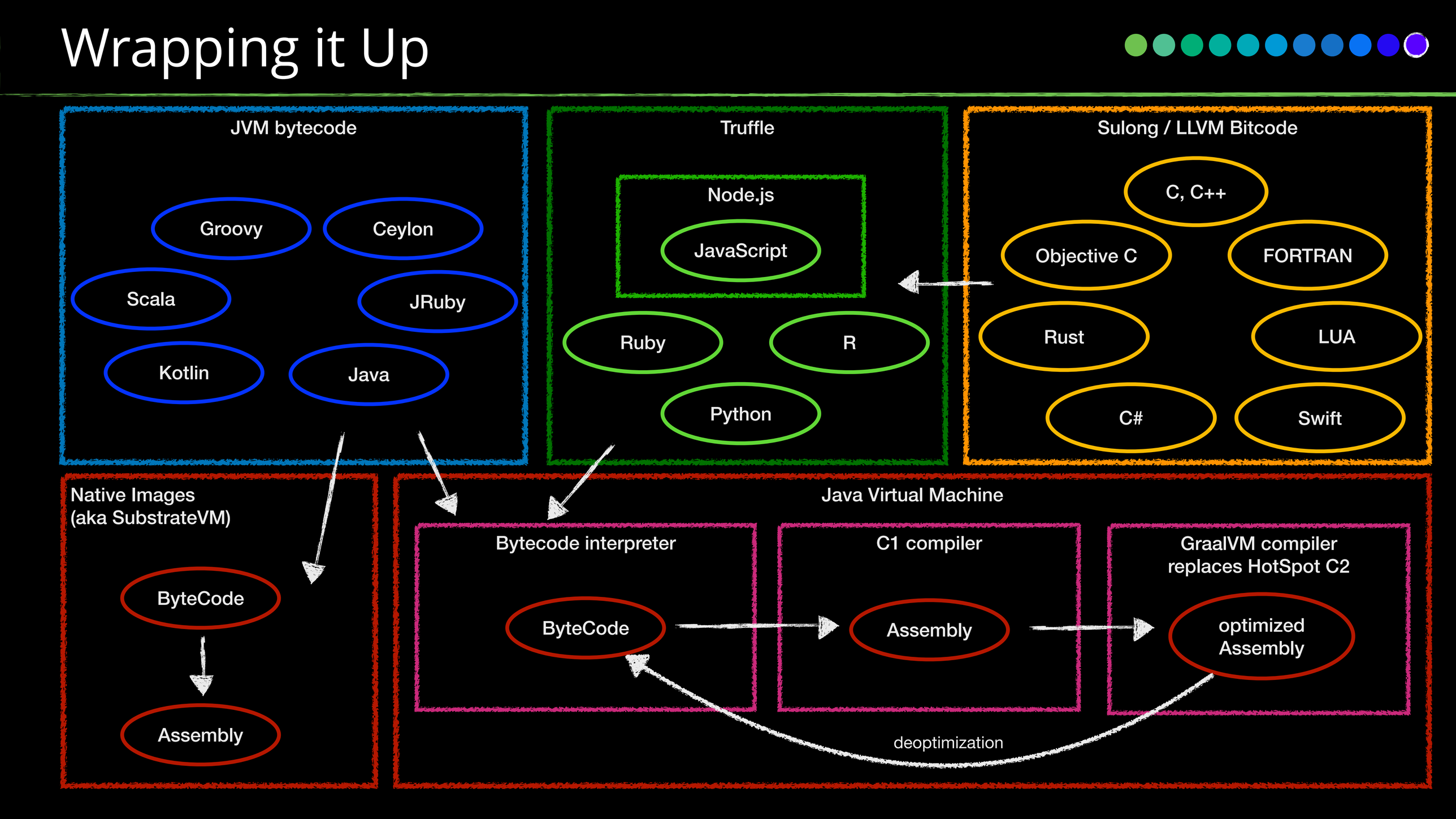
Enterprise Edition vs. Community Edition

Community Edition

Enterprise Edition



Wrapping it Up



Any Questions?

Don't hesitate to reach out to me!

stephan.rauh@opitz-consulting.com

articles@beyondjava.de

@beyondjava

www.beyondjava.net