GraalVM™

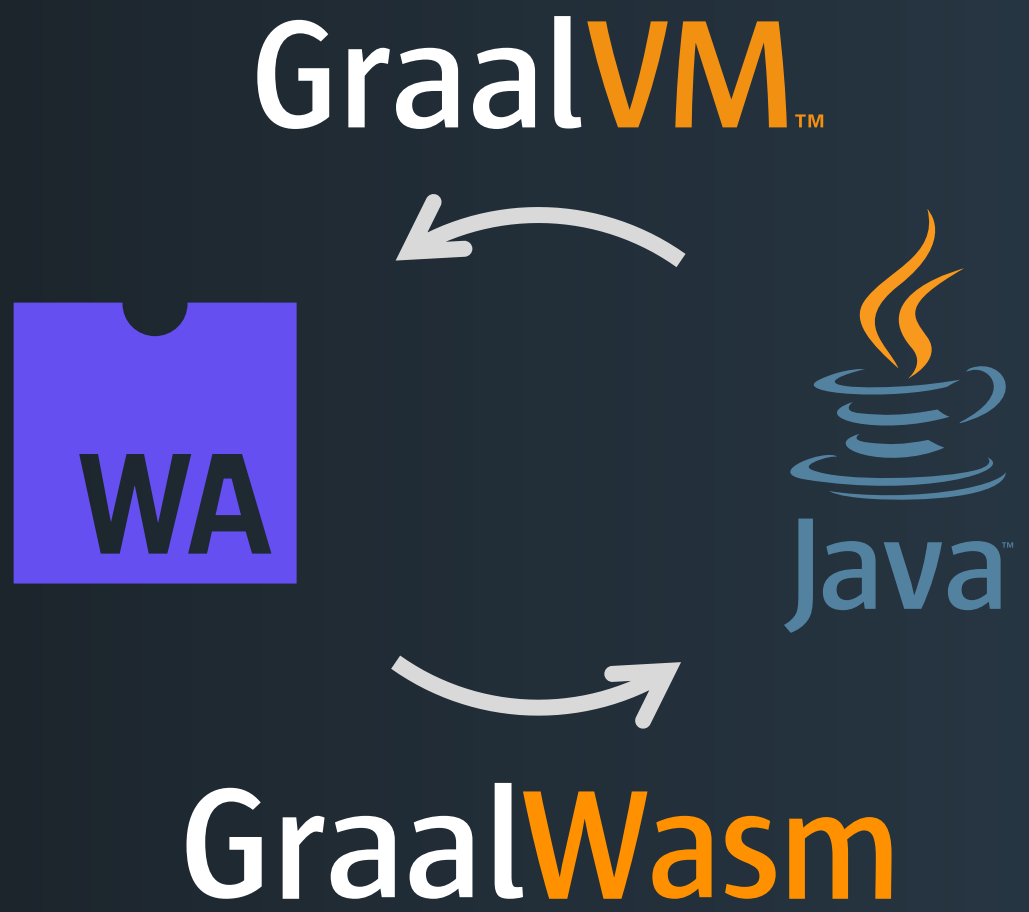
The Future of Write Once, Run Anywhere

From Java to WebAssembly

Patrick Ziegler (patrickziegler.ch)
Fabio Niephaus (fniephaus.com)
Oracle Labs

The Future of Write Once, Run Anywhere

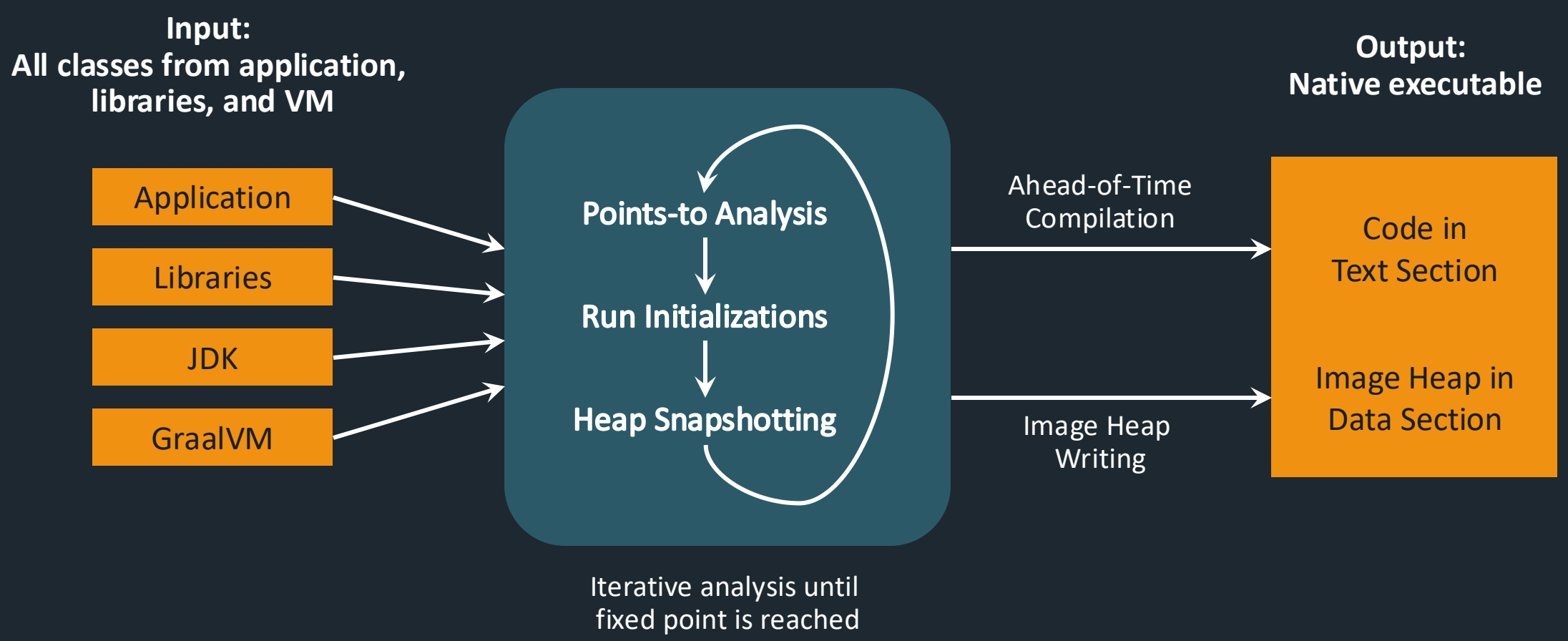
The Future of Write Once, Run Anywhere



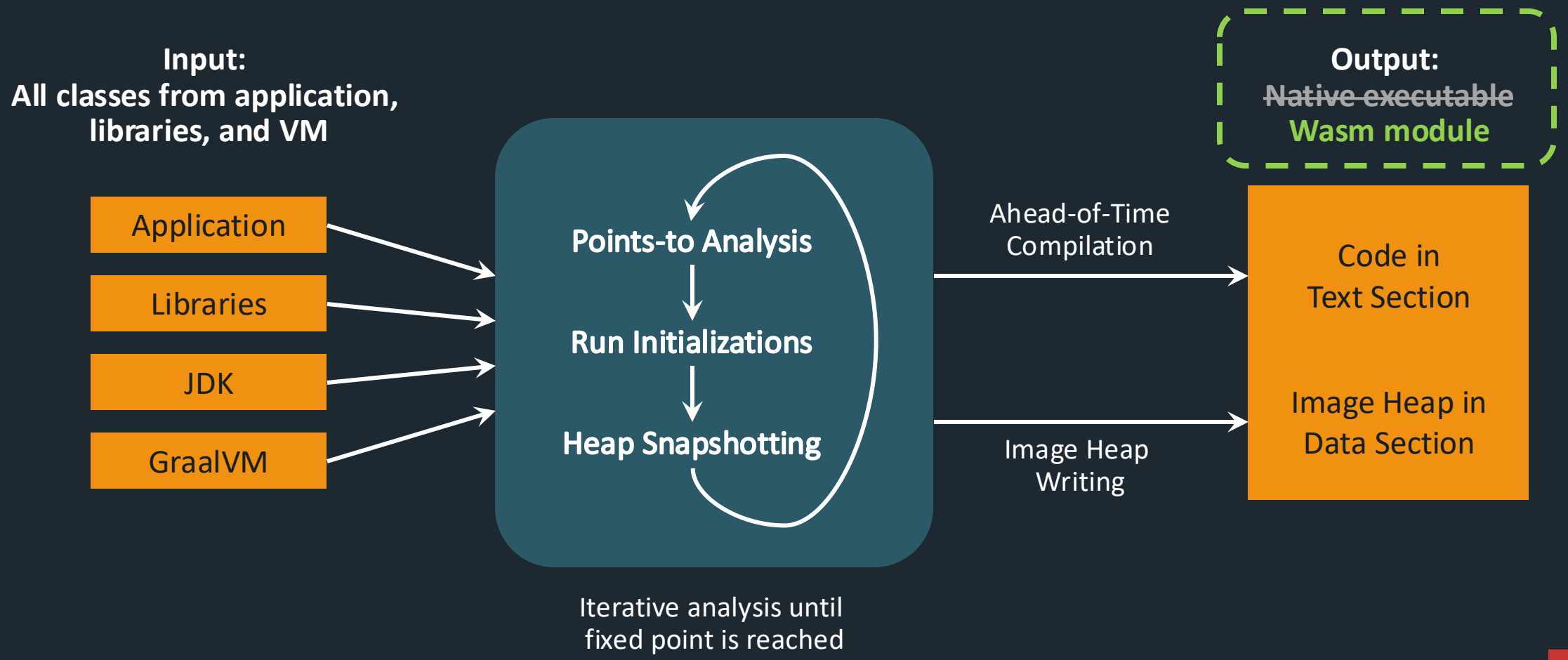
javac to WebAssembly with GraalVM™

Live Demo

GraalVM AOT Compilation



GraalVM AOT Compilation to WebAssembly



The New WebAssembly Backend for GraalVM

- Graal compiler targets WasmGC in a JS embedding

  (`helloworld` program ~1MB in size, before `wasm-opt` or compression)
- Uses Garbage Collection and Exception Handling proposals
- Support for Java ↔ JavaScript interoperability
- Programs can include arbitrary JDK code, which is substituted appropriately where necessary (for example for filesystem access)
- Support for threading, networking, graphics, and others are still missing
- We plan to contribute it to the open-source GraalVM Community Edition

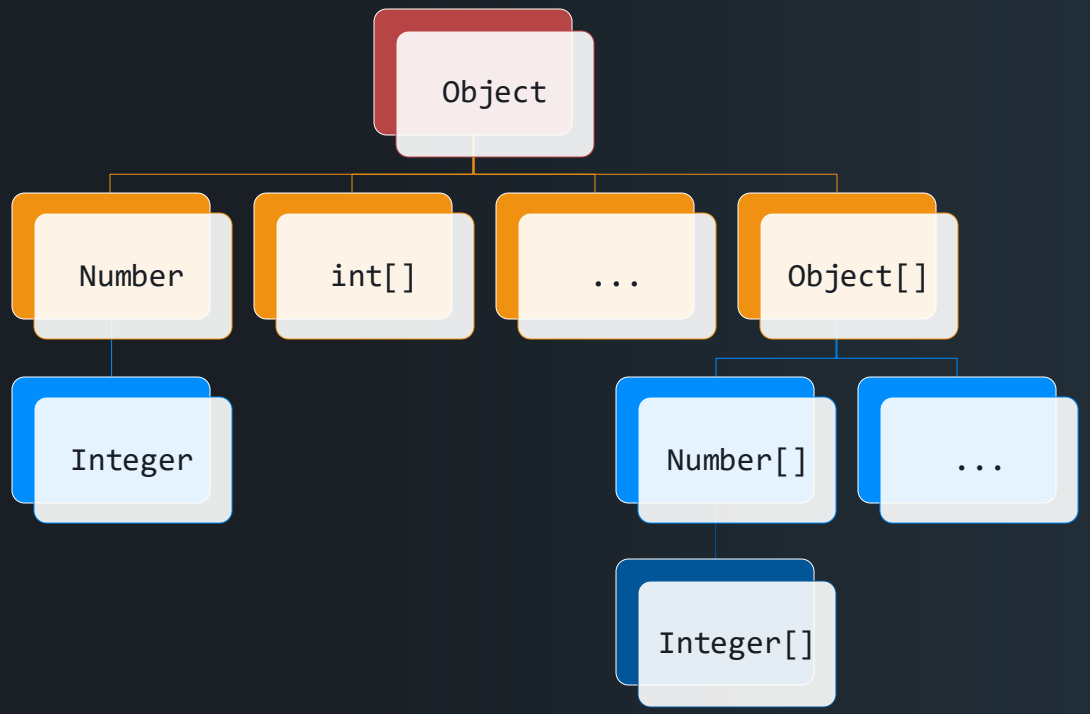
How to Try It Out Today

1. Install the latest early access build of Oracle GraalVM
   For example, using SDKMAN!: `sdk install java 25.ea.15-graal`
2. Make sure the Binaryen toolchain is on the system path
   For example, using Homebrew: `brew install binaryen`
3. Compile JVM bytecode to Wasm using the `--tool:svm-wasm` option:

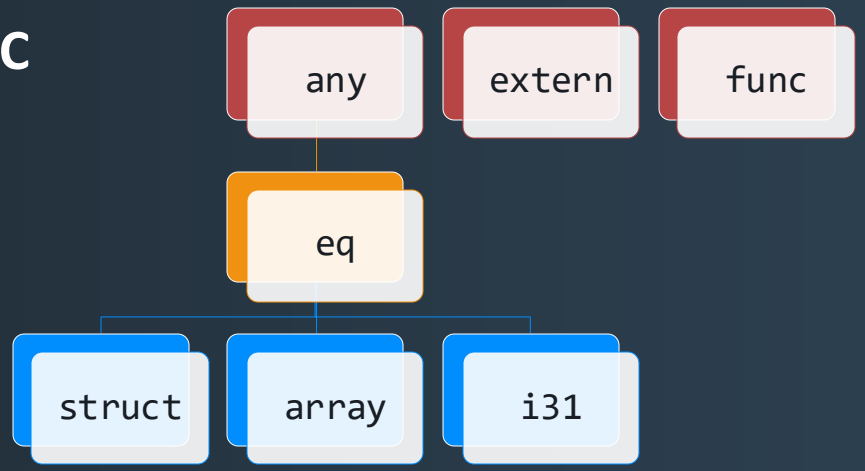
```
$ native-image --tool:svm-wasm HelloWasm
================================================================
GraalVM Native Image: Generating 'hellowasm' (executable)...
================================================================
...
Build artifacts:
/Users/janedoe/dev/hellowasm.js (executable)
/Users/janedoe/dev/hellowasm.js.wasm (executable)
/Users/janedoe/dev/hellowasm.js.wat (build_info)
================================================================
Finished generating 'hellowasm' in 5.3s.
```

Type Hierarchy

Java

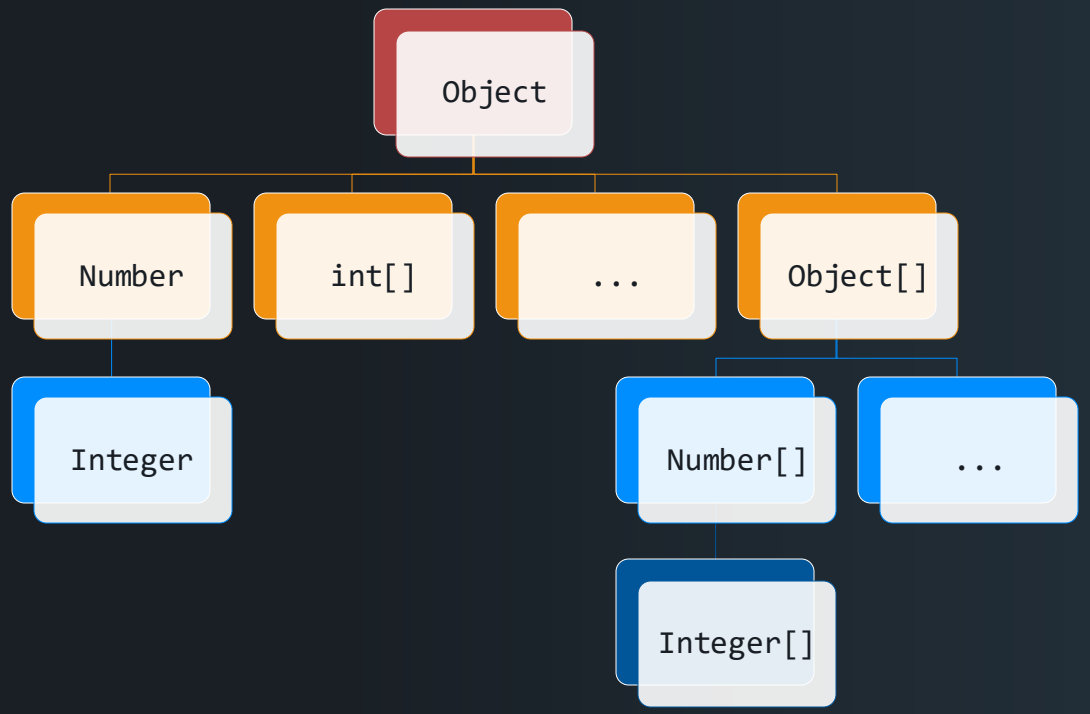


WasmGC

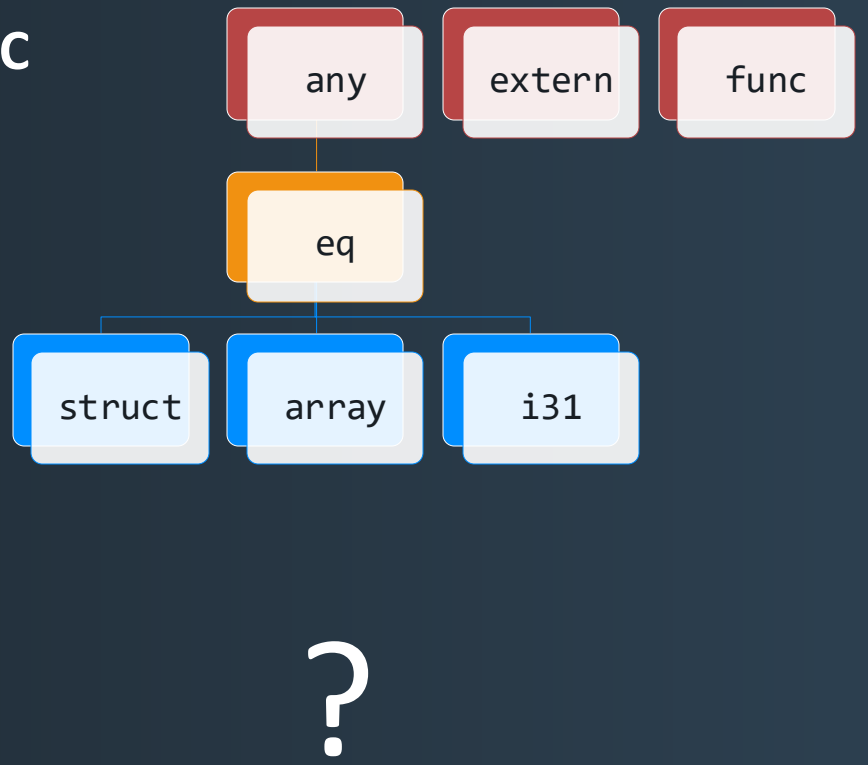


Type Hierarchy

Java

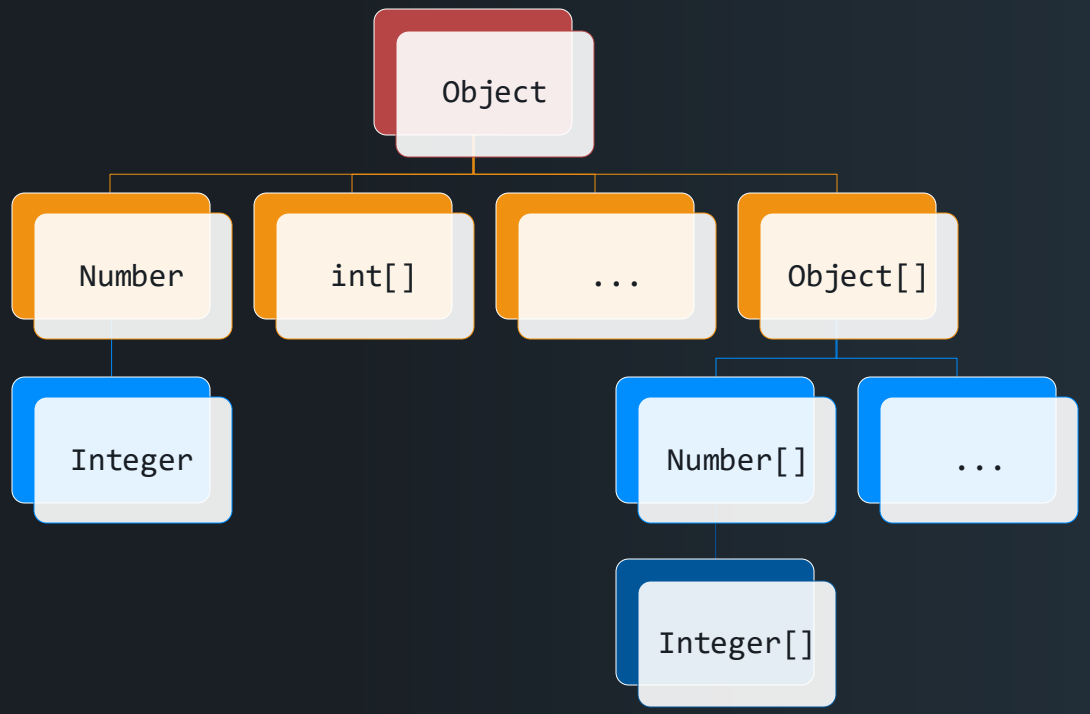


WasmGC

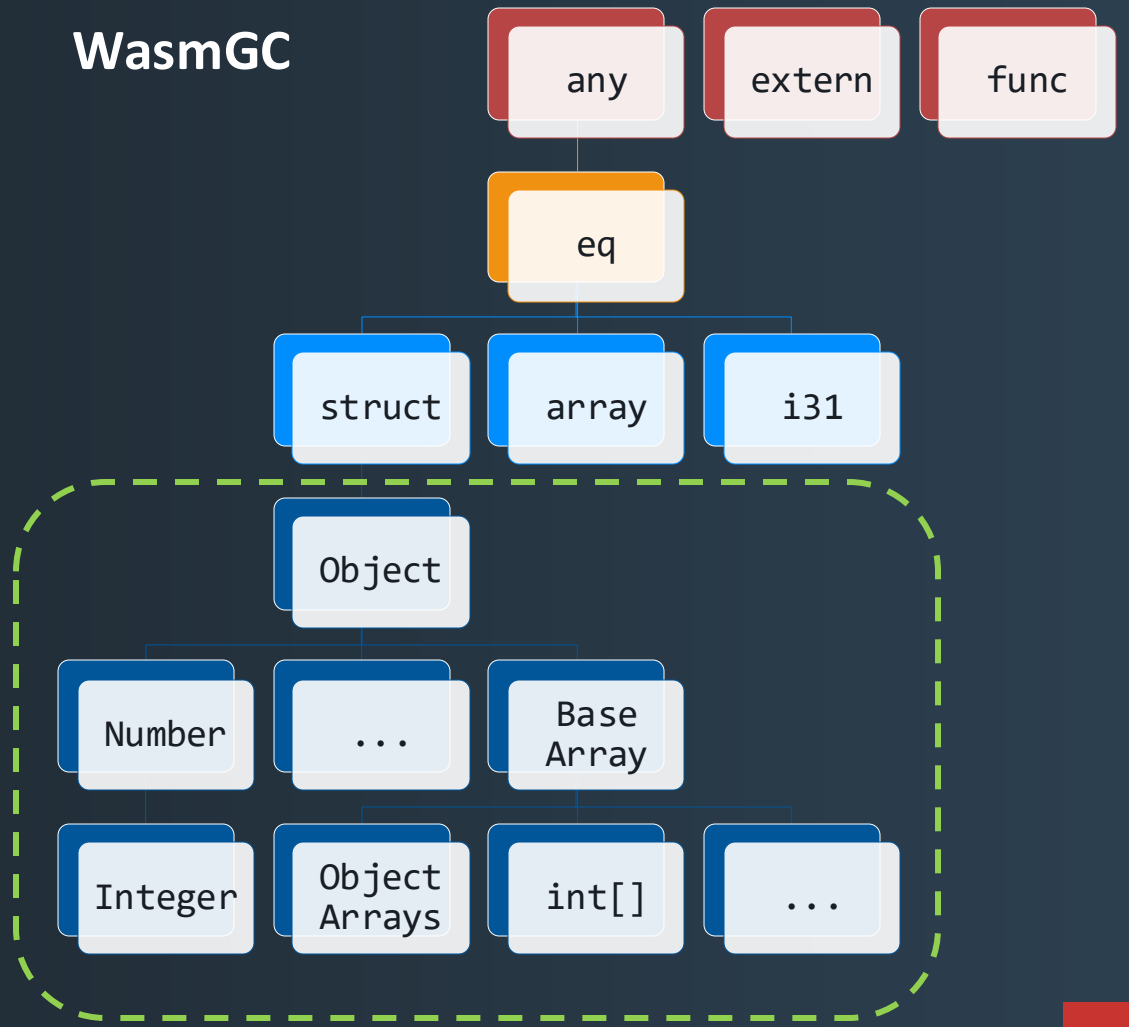


Type Hierarchy

Java



WasmGC



How Type Checks Work

Live Demo

From Java CLI to Client-Side Web Application

Graal Dev Kit for Micronaut

About | Get Started | Docs | Graal Projects | Create App

Using the Graal Development Kit for Micronaut Command Line Interface

This guide describes how to use the Graal Development Kit for Micronaut (GDK) Command Line Interface (CLI) to generate a project containing an application. (To install the GDK CLI, see Installing the GDK CLI.)

The GDK CLI can create the following types of project:

Type	Command
Application: a server application	`create-app NAME`
Function: a serverless cloud function	`create-function NAME`
Gateway Function: a serverless cloud gateway function	`create-gateway-function NAME`

Where `NAME` is the name of the project to create.

The **synopsis** to create a server application, a cloud function, or a cloud gateway function is:

```
gdk create-app [-ehvVx] \
[--b=BUILD-TOOL] \
[--jdk=<javaVersion>] \
[--lang=LANG] \
[--t=TEST] \
[--clouds=CLOUD[,CLOUD...]]... \
[--features=FEATURE[,FEATURE...]]... \
[--services=SERVICE[,SERVICE...]]... \
[NAME]
```

Copy

The options you can use are:

GDK Launcher

GDK Version: 4.7.3.2

Basic | Advanced

Project Type: Application

Project Name: GDK-demo

Base package: com.example

Generate Project

Clouds: OCI, AWS, GCP, Azure

Build Tool: Gradle, Maven

Java Version: 17, 21

Sample Code: Yes, No

Cloud Services

- [x] **Database** — Provides a database access toolkit for repository interfaces
- [ ] **Email** — Provides integration with multiple email providers
- [ ] **Kubernetes** — Provides integration with Kubernetes
- [x] **Logging** — Provides integration with multiple logging frameworks and various other appenders (including email and databases)
- [x] **Metrics** — Provides integration with cloud-specific monitoring services
- [ ] **Object Storage**

The Future of Write Once, Run Anywhere

The Future of Write Once, Run Anywhere

What's New in GraalWasm

- Stable and ready for production, with support for many standardized Wasm features
- Embeddable in Java, standalone distribution available
- WebAssembly/ES module integration makes it easy to use JavaScript bindings for Java ↔ Wasm interactions
- SIMD proposal and Dwarf debugging planned for next release

	Your browser	Chrome	Firefox	Safari	Node.js	Deno	GraalWasm
Phase 5 - The Feature is Standardized							
JS BigInt to Wasm i64 Integration	✓	85	78	15[k]	15.0	1.1.2	21.3
Branch Hinting	?	[a]	[g]	16	[p]	[y]	×
Bulk Memory Operations	✓	75	79	15	12.5	0.4	23.0
Custom Text Format Annotations	?	N/A	N/A	N/A	N/A	N/A	N/A
Extended Constant Expressions	✓	114	112	17.4	21.0	1.33	[ag]
Garbage Collection	✓	119	120	18.2	22.0	1.38	×
Multiple Memories	✓	120	125	×	22.0	1.38	[ai]
Multi-value	✓	85	78	13.1	15.0	1.3.2	22.3
Import/Export of Mutable Globals	✓	74	61	12	12.0	0.1	21.3
Reference Types	✓	96	79	15	17.2	1.16	23.0
Relaxed SIMD	✓	114	[i]	[m]	21.0	1.33	×
Non-trapping float-to-int Conversions	✓	75	64	15	12.5	0.4	22.3
Sign-extension Operators	✓	74	62	14.1[n]	12.0	0.1	22.3
Fixed-width SIMD	✓	91	89	16.4	16.4	1.9	24.1
Tail Call	✓	112	121	18.2	20.0	1.32	×
Typed Function References	✓	119	120	18	22.0	1.38	×
Phase 4 - Standardize the Feature							

webassembly.org/features/

Debugging with GraalWasm

Live Demo

Debugging Rust compiled to Wasm embedded in Spring Boot

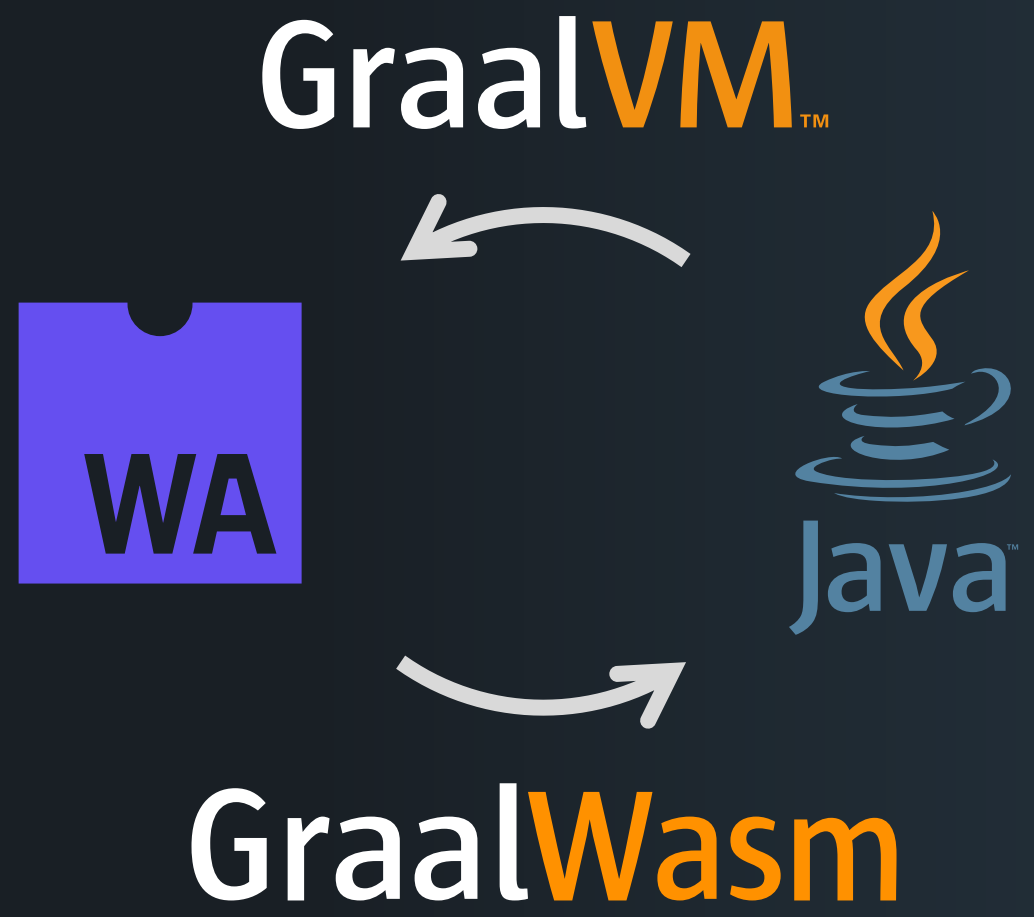
using VS Code via Debug Adapter Protocol

using IntelliJ IDEA via Chrome Inspector Procotol

The Future of Write Once, Run Anywhere

- GraalWasm makes it easy to extend Java applications with WebAssembly
- GraalJS allows use of JavaScript bindings for Java ↔ Wasm interactions

The Future of Write Once, Run Anywhere



- GraalVM can generate WasmGC now!
- Wasm modules built with JDK 25 EA
- Wasm backend provides Java ↔ JavaScript interoperability

- GraalWasm makes it easy to extend Java applications with WebAssembly
- GraalJS allows use of JavaScript bindings for Java ↔ Wasm interactions

The Future of Write Once, Run Anywhere

`javac` demo and code on GitHub:

- GraalVM can generate WasmGC now!
- Wasm modules built with JDK 25 EA
- Wasm backend provides Java ↔ JavaScript interoperability
- GraalWasm makes it easy to extend Java applications with WebAssembly
- GraalJS allows use of JavaScript bindings for Java ↔ Wasm interactions