

GraalVM™ + Wasm

Fast, Efficient, Portable Apps

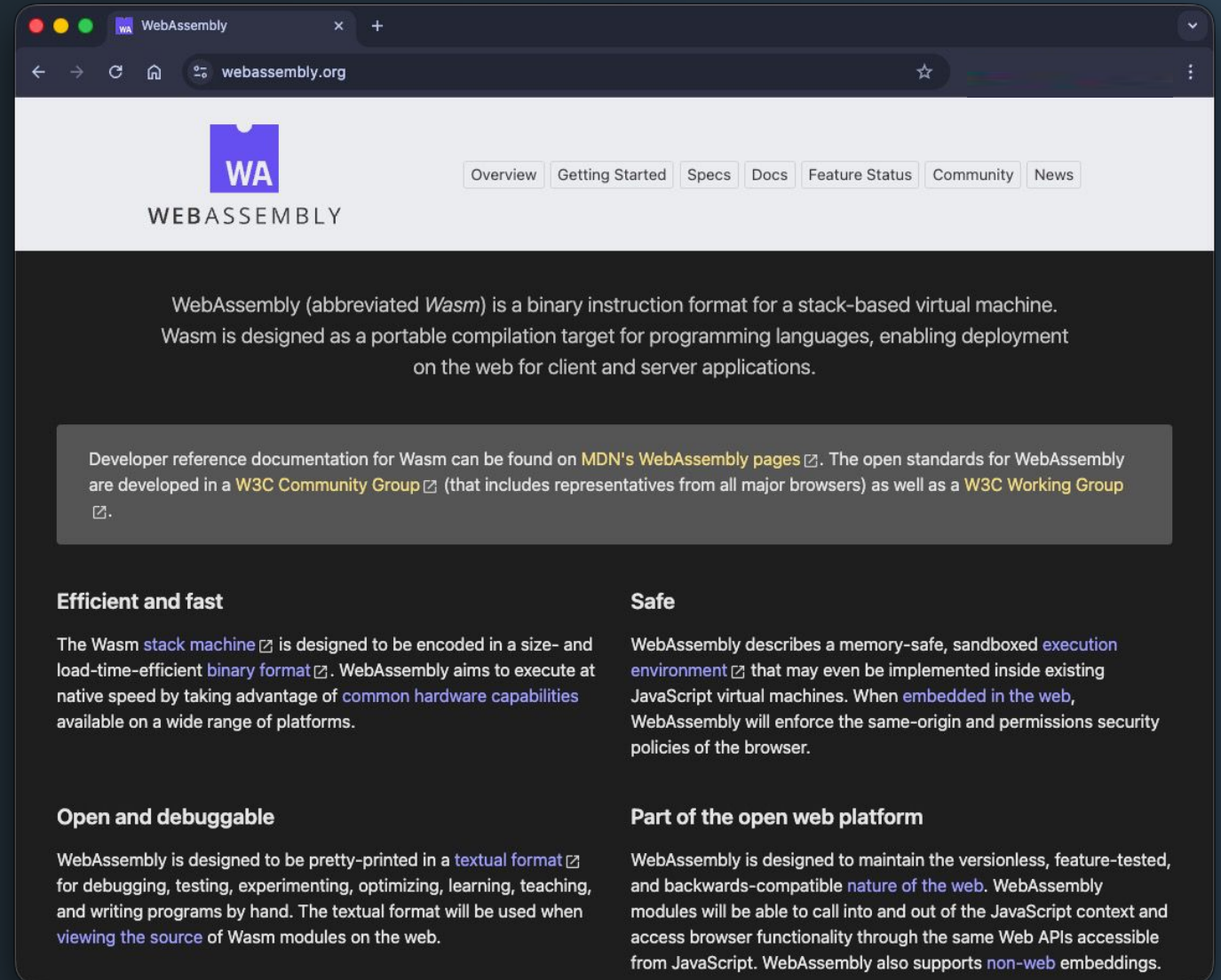
Fabio Niephaus (fniephaus.com)

Shaun Smith (@shaunmsmith)



WebAssembly (Wasm) in a nutshell

- **Binary instruction format** for a stack-based virtual machine
- Designed as a **portable** compilation target for programming languages
- Runs in **modern web browsers** and in **server-side environments**
- Can run in **sandboxed** environments for untrusted code execution



WebAssembly in Action



Live Demo

Why WebAssembly is interesting for us

- Portable native extensions and libraries
 - Integrate C, C++, Rust, and Go libraries using **Wasm as an alternative to JNI or FFM API**
- Bringing JVM languages into WebAssembly ecosystem
 - Turn JVM code into **Wasm components that can interoperate with others** from other languages
- Server-side Wasm
 - **Cost-effective deployments** on managed, Wasm-enabled clouds
 - Fast startup, low resource usage, compact packaging, minimize attack surface, ...
- Wasm in the browser
 - Run **JVM applications on the client**, including mobile devices

GraalWasm in Action

—
Live Demo



Introducing GraalWasm



Wasm for the JVM

Load and use WebAssembly modules and functions



WebAssembly 2.0 Support

Full WebAssembly 2.0 compatibility and WASI feature extensions



Portable Native Extensions

Integrate C, C++, Rust, and Go libraries using platform-independent Wasm (as an alternative to JNI and Panama)



JavaScript Integration

Simplifies use of WebAssembly modules with JavaScript wrappers via GraalJS



Fastest Wasm on the JVM

Graal JIT compiles WebAssembly for native code speed



100% Java

Written in pure Java with zero native dependencies

How to embed GraalWasm and other Graal Languages on JDK 21+

Just add to your build



```
pom.xml

1 <dependency>
2   <groupId>org.graalvm.polyglot</groupId>
3   <artifactId>polyglot</artifactId>
4   <version>25.0.2</version>
5 </dependency>
6 <dependency>
7   <groupId>org.graalvm.polyglot</groupId>
8   <artifactId>wasm</artifactId> <!-- or js, python, ... -->
9   <version>25.0.2</version>
10  <type>pom</type>
11 </dependency>
```

```
build.gradle.kts

1 implementation("org.graalvm.polyglot:polyglot:25.0.2")
2 implementation("org.graalvm.polyglot:wasm:25.0.2")
```



Just specify
the language

GraalWasm at a glance



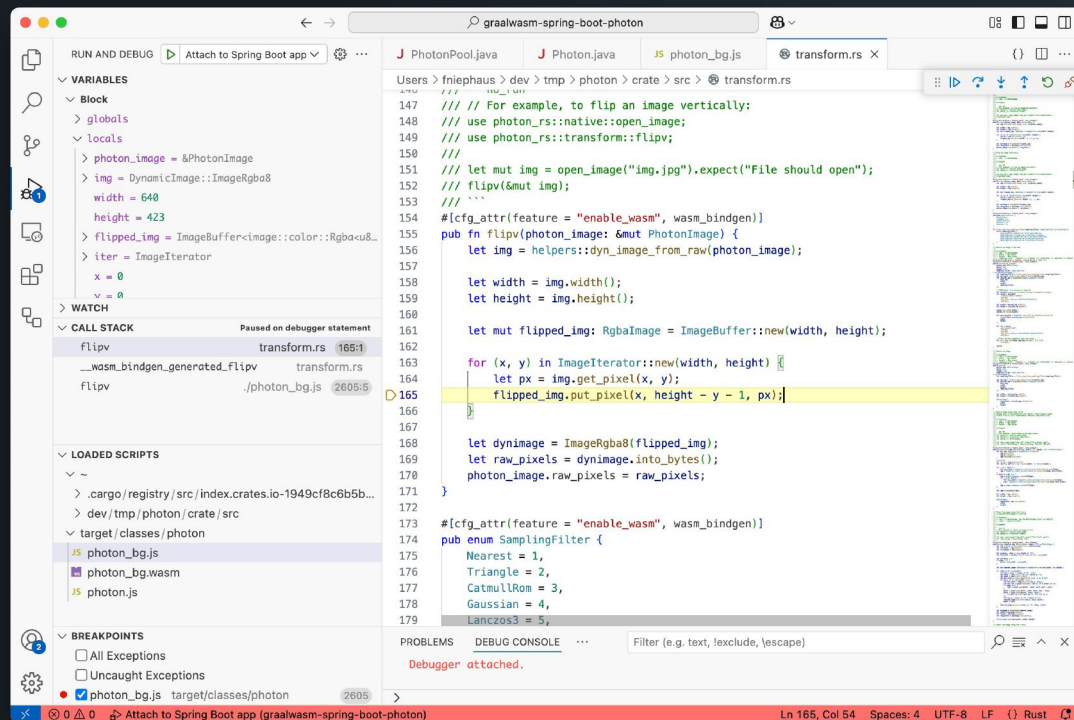
- Stable and **ready for production**
- **Embeddable in JVM applications**, Wasm, JS+Wasm, and Node standalone distributions available
- WebAssembly/ES module integration **simplifies Java ↔ Wasm interop**
- SIMD proposal and **Dwarf debugging** landed in latest 25.0 release
- WasmGC proposal soon in EA builds

	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]	✘ ^[v]	✘
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	✘

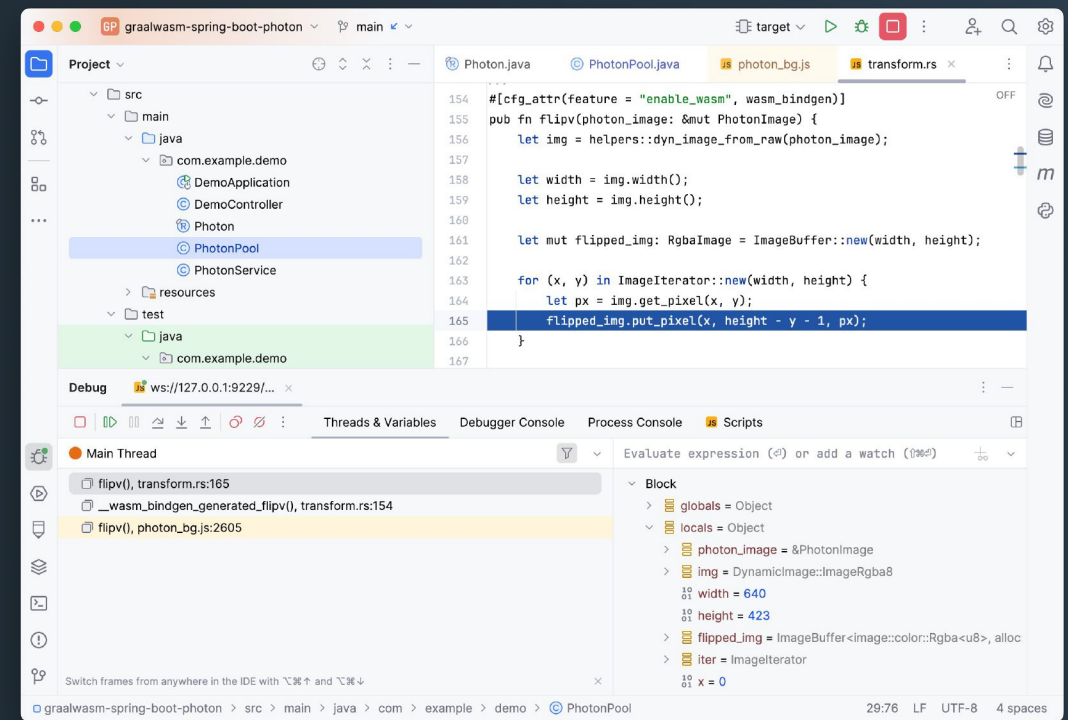
webassembly.org/features/

Debugging Rust compiled to Wasm embedded in Spring Boot

using VS Code via Debug Adapter Protocol



using IntelliJ IDEA via Chrome Inspector Protocol



GraalVMTM Web Image in Action

—
Live Demo



From CLI tool to client-side web application

Graal Dev Kit
for Micronaut

AboutGet StartedDocs ▾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	<code>create-app NAME</code>
Function: a serverless cloud function	<code>create-function NAME</code>
Gateway Function: a serverless cloud gateway function	<code>create-gateway-function NAME</code>

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

HomeInfo

BasicAdvanced

Project TypeProject NameBase package

Application ▾

GDK-demo

com.example

Generate Project

CloudsBuild ToolJava Version

☒ OCID☒ AWS☐ GCP☐ Azure

☒ Gradle☐ Maven

☒ 17☐ 21

Sample Code

☒ Yes☐ No

Cloud Services

☒ Database
Provides a database access toolkit for repository interfaces

☐ Email
Provides integration with multiple email providers

☐ Kubernetes
Provides integration with Kubernetes

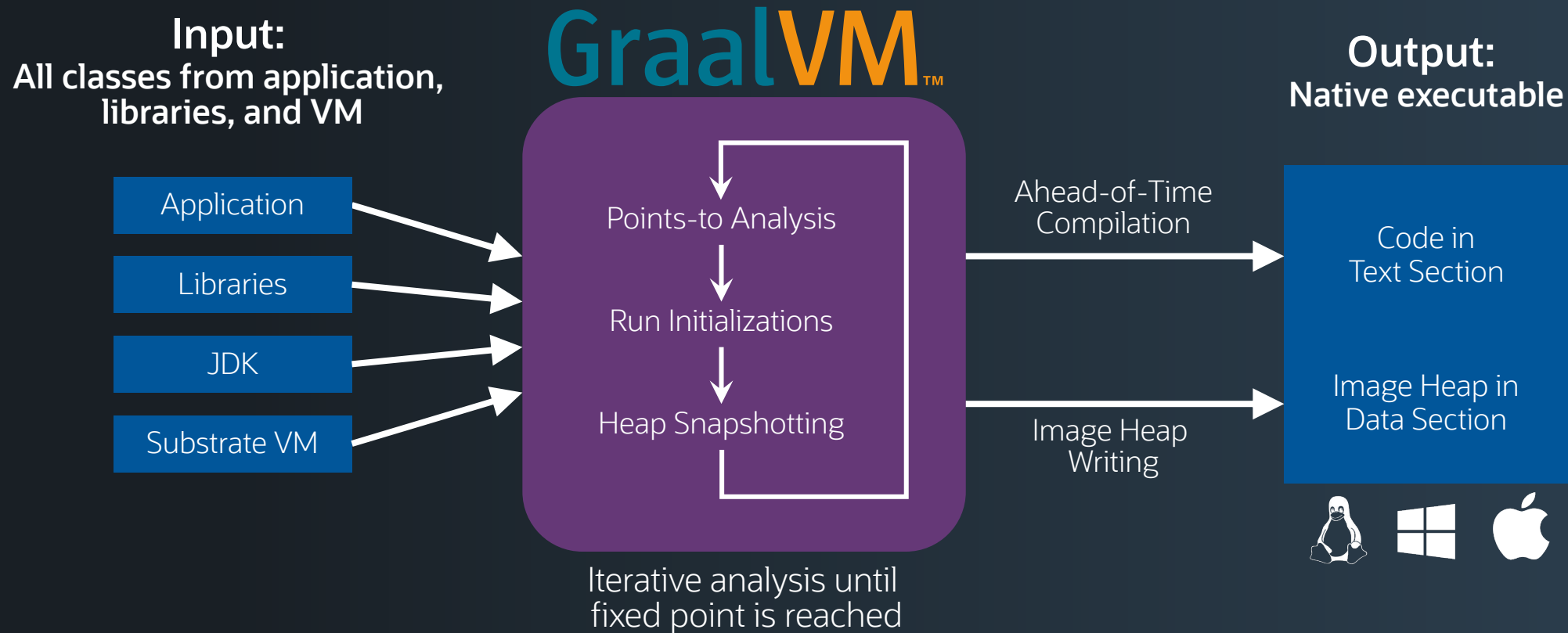
☒ Logging
Provides integration with multiple logging frameworks and various other appenders (including email and databases)

☒ Metrics
Provides integration with cloud-specific monitoring services

☐ Object Storage

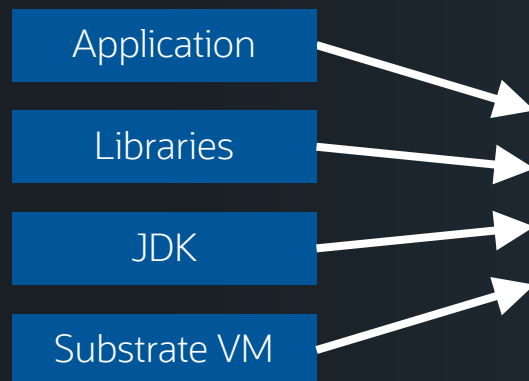


GraalVM Native Image Compilation

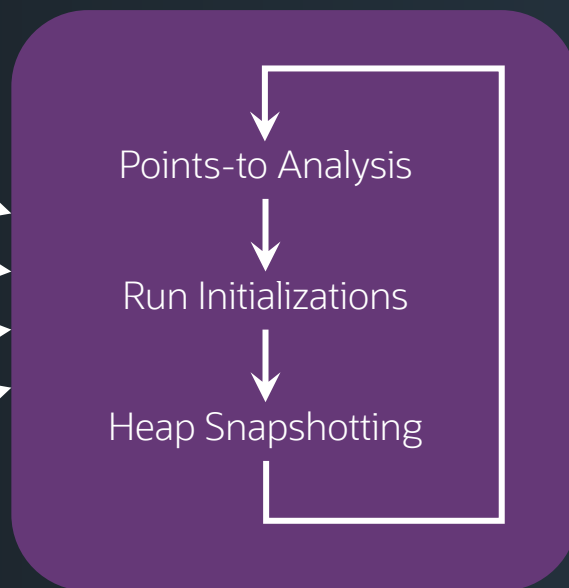


GraalVM ~~Native~~ Web Image Compilation

Input:
All classes from application,
libraries, and VM



GraalVMTM

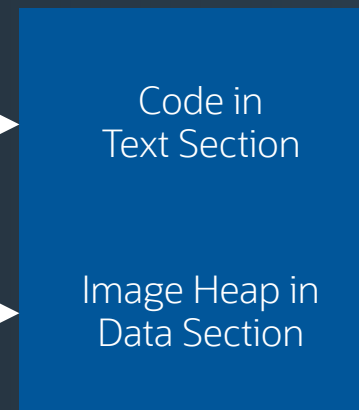


Iterative analysis until
fixed point is reached

Ahead-of-Time
Compilation

Image Heap
Writing

Output:
~~Native-executable~~
Wasm module



GraalVM Web Image at a glance

- Graal compiler targets **WasmGC** in a JavaScript embedding
 - `helloWorld` program ~1MB in size, before `wasm-opt` or compression
- 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
- Not targeting the Component Model and System Interface (WASI) yet
- Developed as part of the **open-source** GraalVM Community Edition

Web Image API Preview for JavaScript Interoperability

Package org.graalvm.webimage.api

package org.graalvm.webimage.api

This package provides an interface between Java code and the embedding JavaScript code.

This functionality is limited to the WebAssembly backend for Native Image.

Any functionality specified here is experimental, subject to change, and may not fully work yet.

Since:

25.0

```
<dependency>
  <groupId>org.graalvm.sdk</groupId>
  <artifactId>webimage-preview</artifactId>
  <version>25.0.2</version>
</dependency>
```

All Classes and Interfaces	Classes	Exception Classes	Annotation Interfaces
Class	Description		
JS	Replaces the body of a Java method with JavaScript code.		
JS.Code	A snippet of JavaScript code that must be included in the image.		
JS.Code.Include	Designates the path to the JavaScript file with the code that must be included in the image.		
JS.Code.Include.Group			
JS.Coerce	When this annotation is used together with the JS annotation, the arguments and return values of the respective method undergo implicit Java-to-JavaScript and JavaScript-to-Java conversions.		
JS.Export	Denotes that the annotated Java class should be exported from the VM class.		
JS.Import	Denotes that the annotated class represents a JavaScript class from the global scope.		



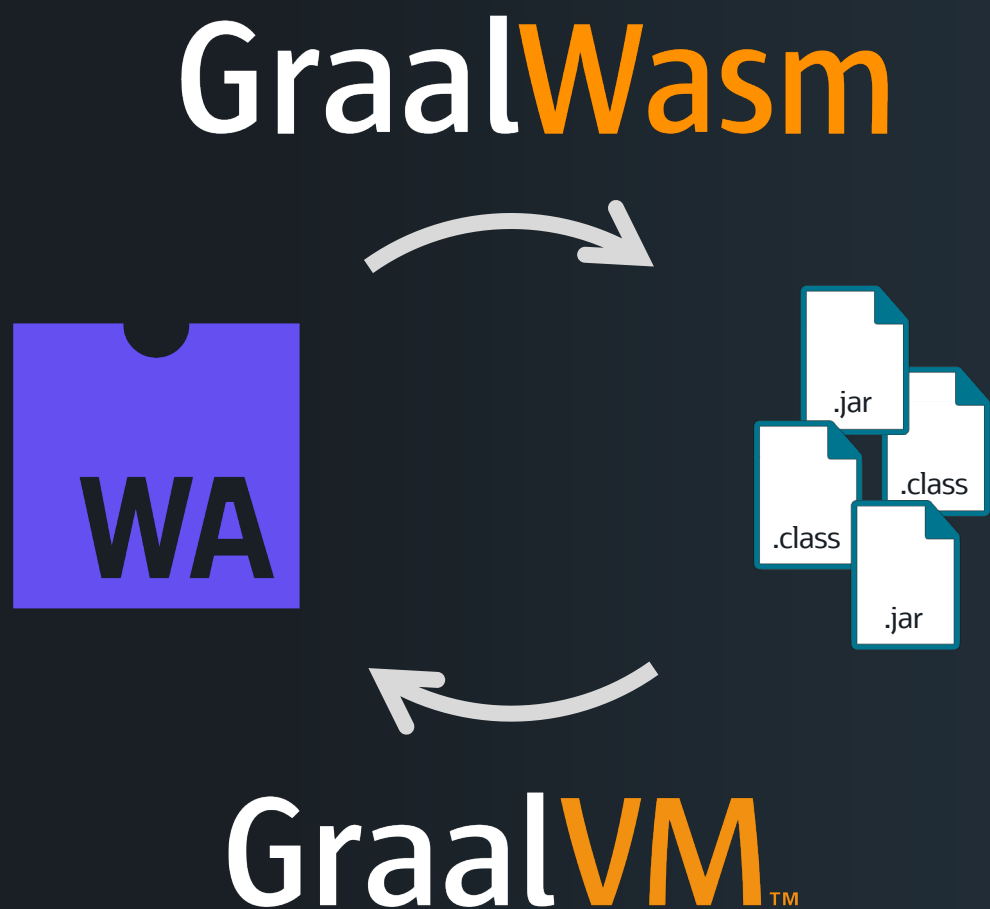
How to try it out today

1. Install the Oracle GraalVM 25 release
For example, using SDKMAN!: `sdk install java 25.0.2-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 HelloWorld
=====
GraalVM Native Image: Generating 'helloworldasm' (executable)...
=====
...
Build artifacts:
/Users/janedoe/dev/helloworldasm.js (executable)
/Users/janedoe/dev/helloworldasm.js.wasm (executable)
/Users/janedoe/dev/helloworldasm.js.wat (build_info)
=====
Finished generating 'helloworldasm' in 5.3s.
```

Conclusion

The GraalWasm runtime and GraalVM Web Image



- GraalWasm makes it easy to **extend JVM applications with portable Wasm modules**
- GraalJS allows use of **JavaScript bindings for Java ↔ Wasm interactions**
- Oracle GraalVM 25 can **generate WasmGC now!**
(experimental but already useful)
- Give them a try, **provide feedback**, and contribute!

The GraalWasm runtime and GraalVM Web Image

GraalWasm, GraalJS, GraalPy demos on GitHub:



javac on WebAssembly demo:

