ORACLE

Oracle GraalVM

Petr Novotny
GraalVM Software Development Manager
Oracle Labs
11.11. 2023

Oracle Labs

Oracle Labs je výzkumná a vývojová organizace v rámci firmy Oracle.

V České republice má Oracle Labs týmy v Brně a v Praze.

Úzce spolupracujeme s VUT FIT a MatFyz UK formou studentských stáží a výzkumných grantů.

ORACLE Red Bull RACING

Oracle Open Source

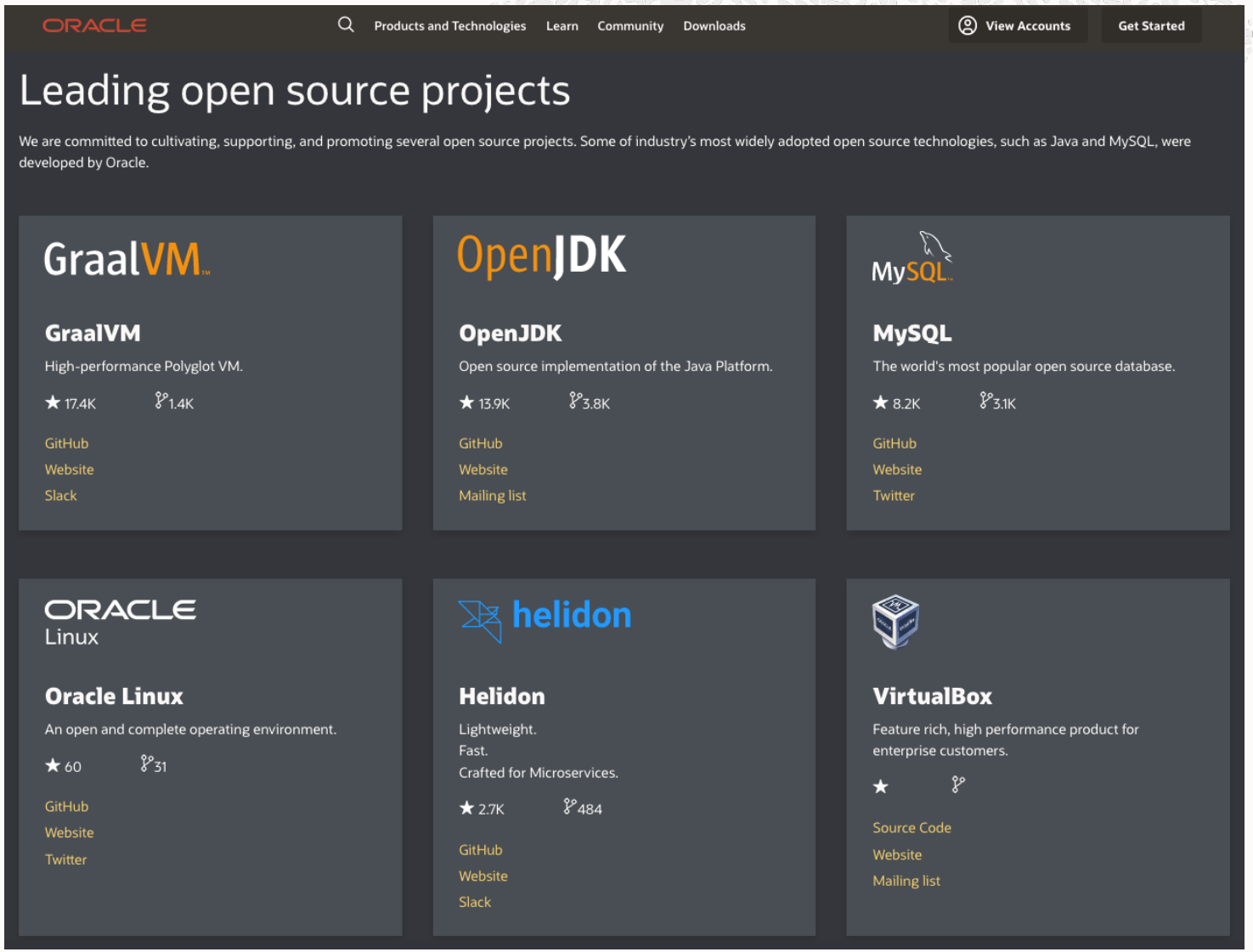
Oracle Labs iniciovaly a rozvíjejí hlavní open source projekty podporované firmou Oracle.

GraalVM™

OpenJDK™

MySQL®

HeatWave

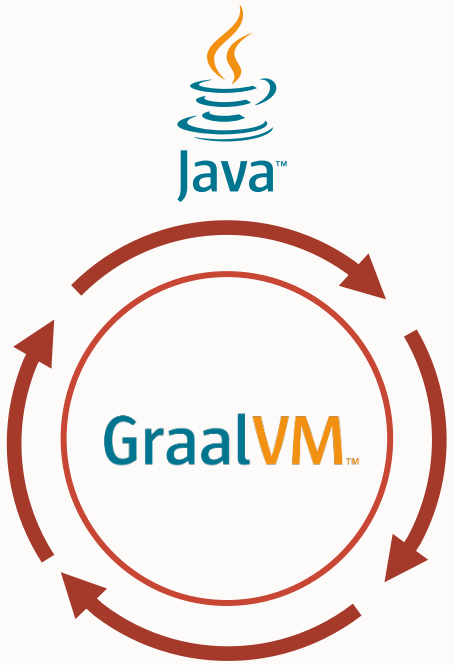


GraalVM, Graal Cloud Native, GraalOS

Více informací na graal.cloud.

Co je GraalVM

Více informací na graalvm.org.



Just-in-Time (JIT) kompilátor

Ahead-of-Time (AOT) kompilátor Native Image

GraalVM jako prostředí pro běh aplikací v různých jazycích

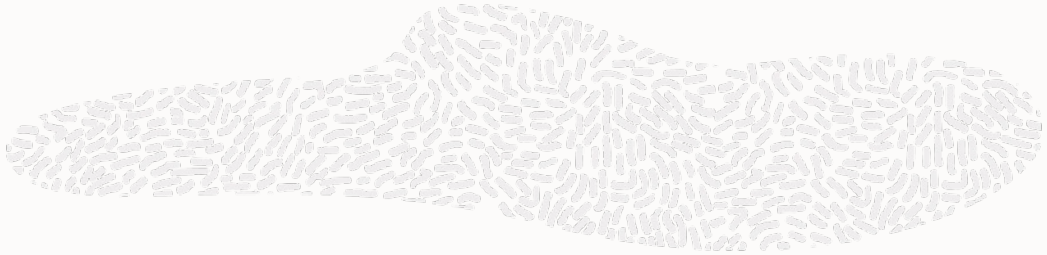
Komunita kolem GraalVM

GraalVM má rozsáhlou a aktivní komunitu.

Spolupracujeme s dalšími open source projekty, např. v oblasti microservices jsou to projekty Spring, Quarkus, Micronaut a Helidon.

David Kozák

2020+ Research Assistant, Oracle Labs
2021+ PhD student, FIT VUT



GraalVM™

LLVM (Sulong)

WASM

Espresso (JVM)

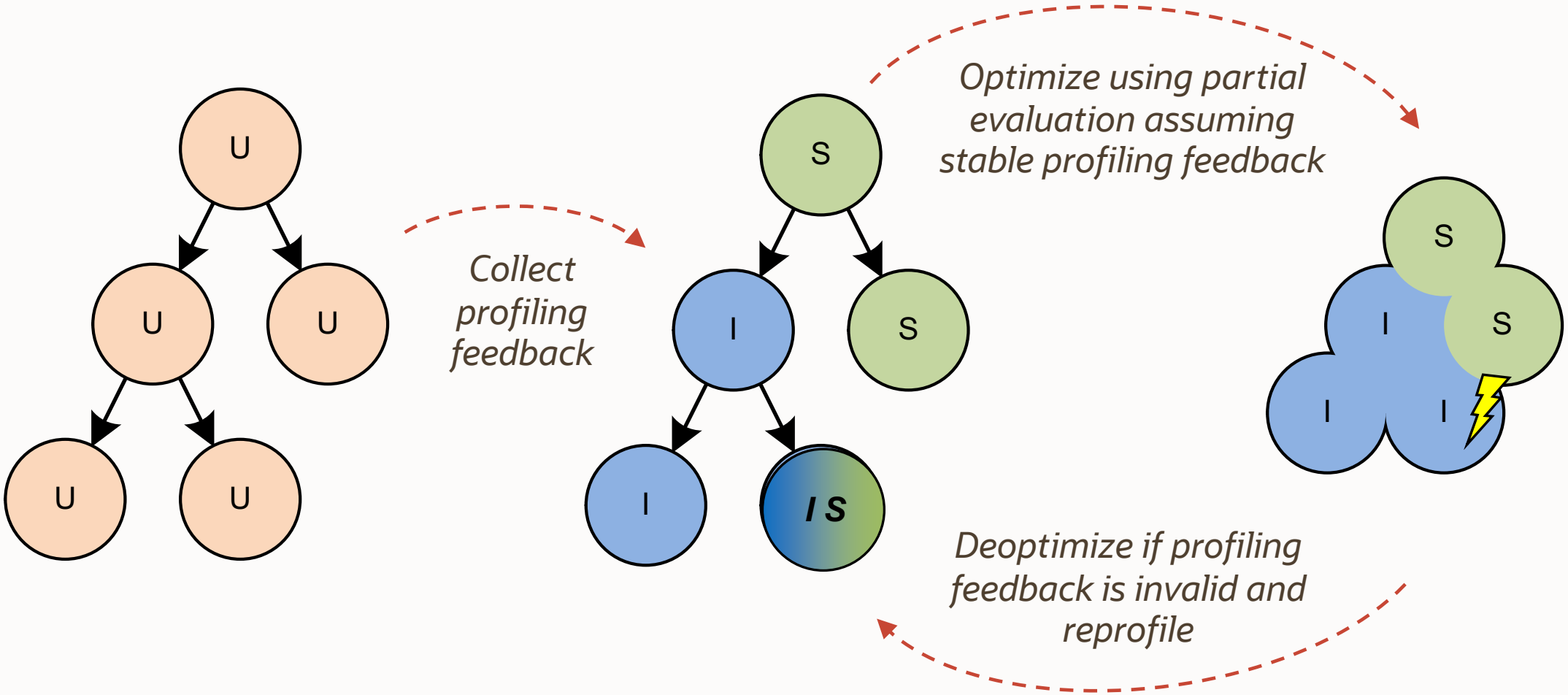
Polyglot Truffle Runtime

Graal Compiler

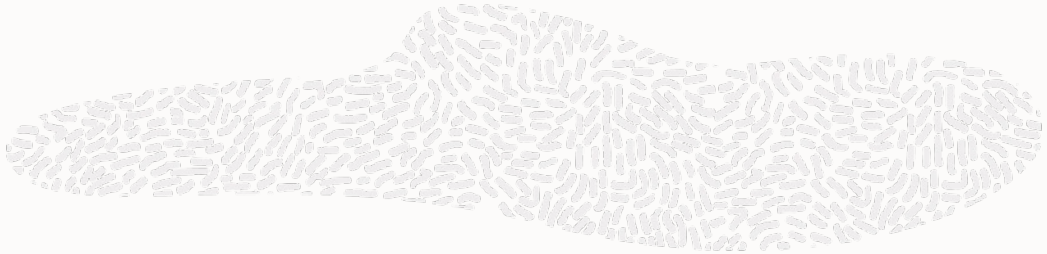
Java VM

Dynamic speculation and deoptimization

One VM to Rule Them All
Onward! 2013



GraalVM v Netsuite



NetSuite

- Integrovaný soubor cloud business aplikací.
- Rozsáhlá a komplexní Java aplikace.
- Nasazená na vysokém počtu serverů.

ORACLE®
NETSUITE

SuiteScript

- Rozšíření JavaScriptu běžící na serveru.
- Přizpůsobení a automatizace business procesů.
- V současnosti JIT kompilován pomocí **GraalJS.**

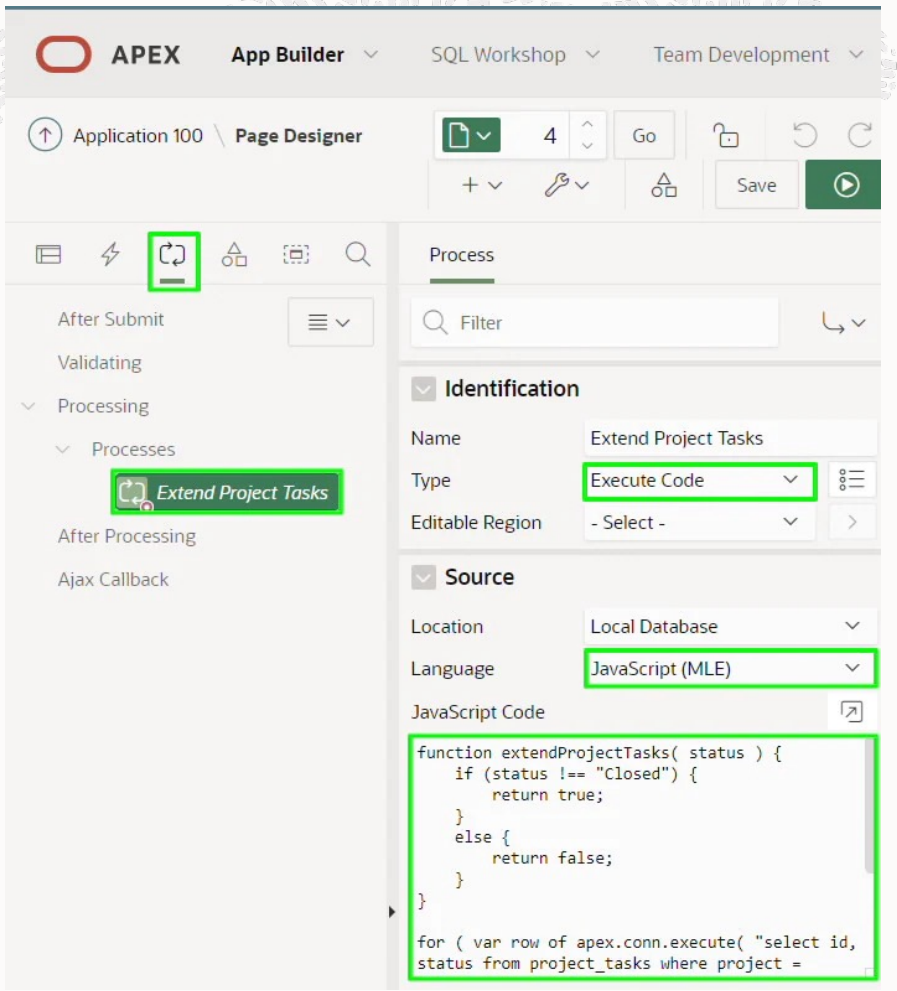
Více informací v plánovaném blog postu.

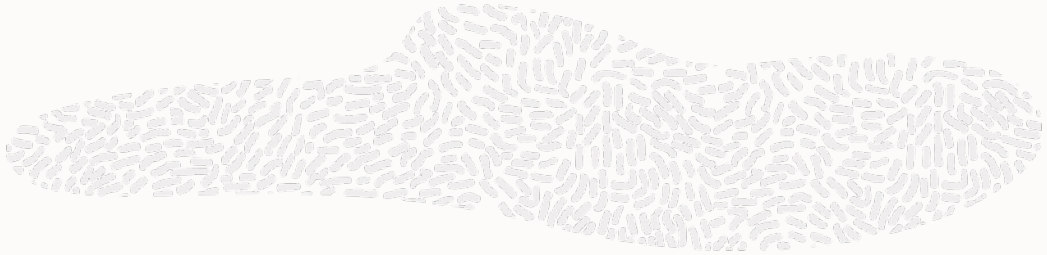
From research to production

MLE and the Future of Server-Side Programming in Oracle APEX

February 11, 2021 | 11 minute read

Salim Hlayel
Principal Product Manager





GraalVM Native Image

GraalVM™

JIT

```
java MyMainClass
```

AOT

```
native-image MyMainClass
./mymainclass
```

JIT

Load JAR files from disk

⬇

Uncompress class files

⬇

Verify class definitions

⬇

Execute in interpreter (~20x slower)

⬇

Gather profiling feedback

⬇

Compile to machine code

⬇

Execute at peak performance

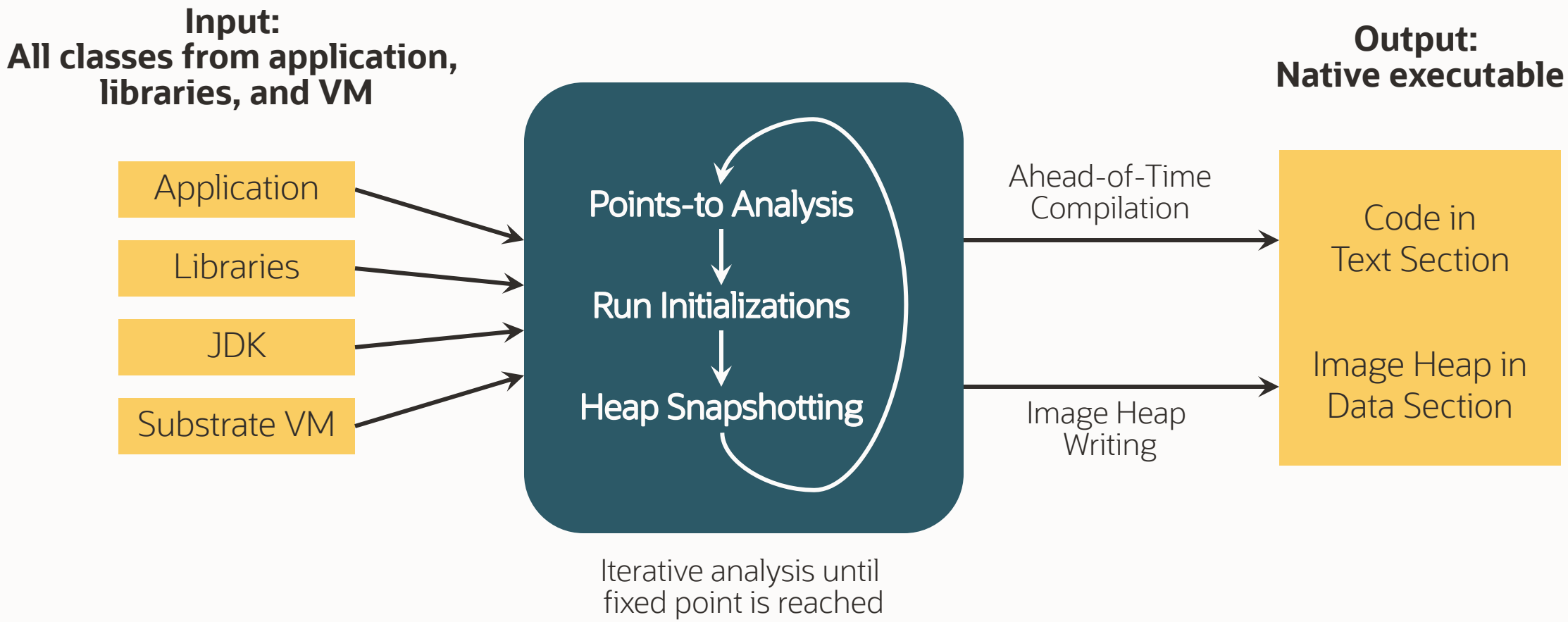
AOT

Load executable from disk

⬇

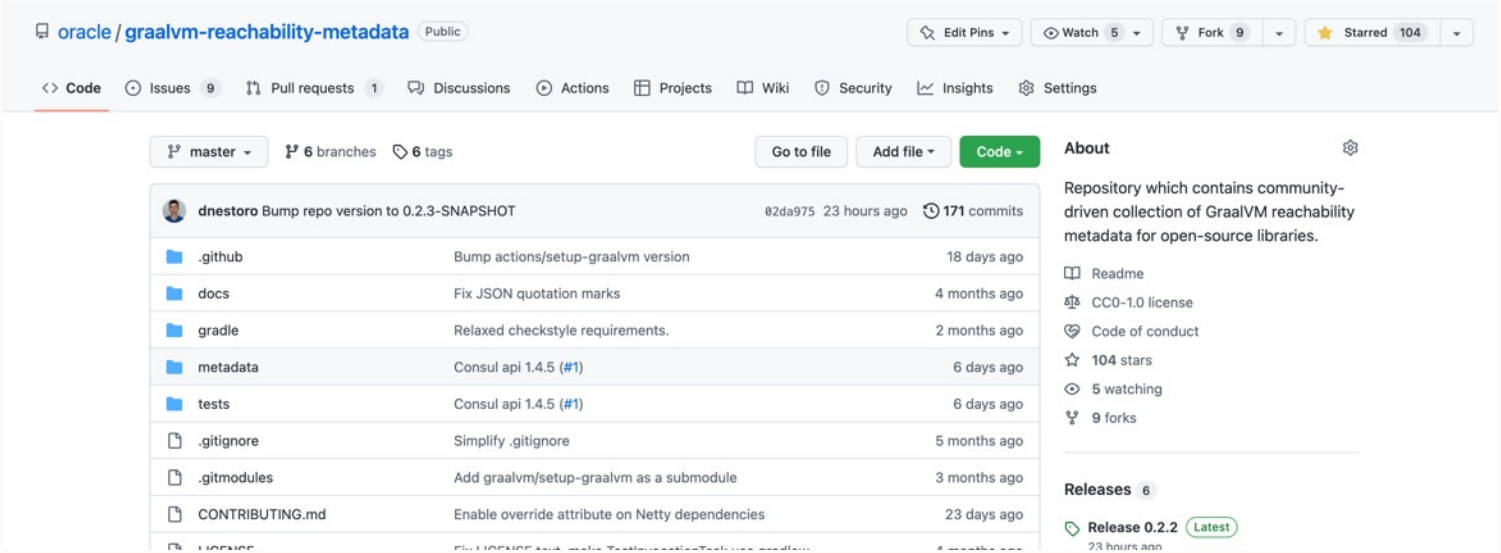
Execute at peak performance

Native Image Build Process



GraalVM & Reflection?

- GraalVM 🤝 Reflection!
- Native Image tries to resolve the target elements through a static analysis that detects calls to the Reflection API
  - If the analysis can not automatically detect your use of reflection, you might need additional configuration
- Trace reflection, JNI, resource usage on the JVM with the tracing agent
  - Manual adjustment / addition might still be necessary



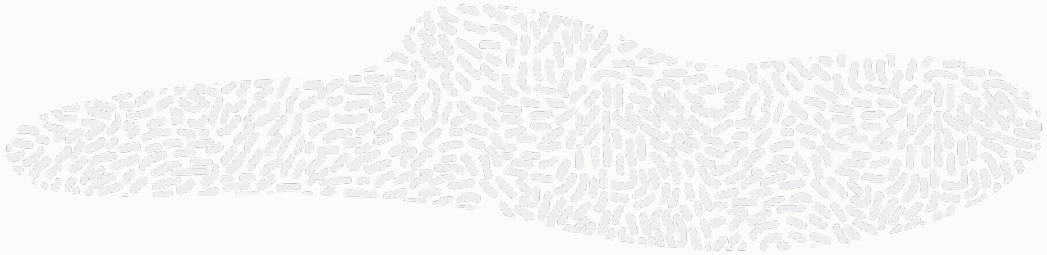
Java in the Cloud - Goals

Start Fast

Low Resource Usage

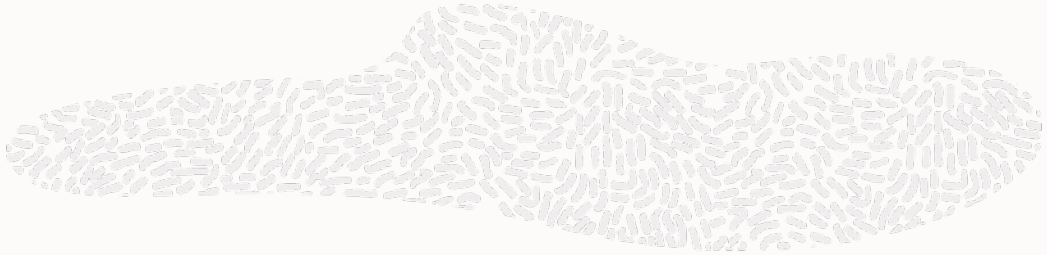
Minimize Vulnerability

Compact Packaging



Vývoj Native Image

Mé zkušenosti z dlouhodobé stáže



První větší úkol - Lokalizace

Lokalizace v Javě

Řešena pomocí **Resource Bundles:**

- Mapování klíč => hodnota s jednoduchým konceptem dědičnosti.

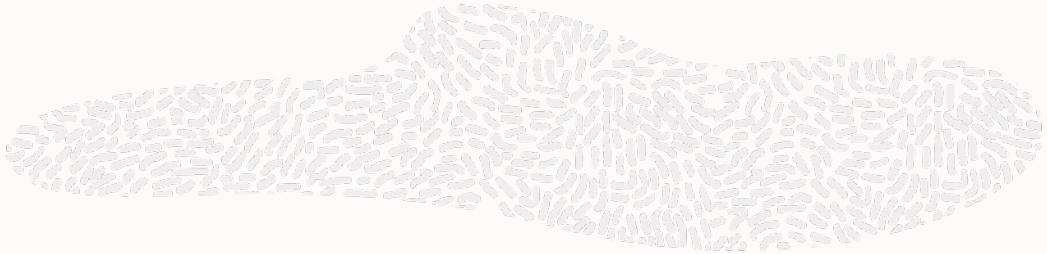
```
abstract class ResourceBundle {
   /** Get the string value for a given key */
   public String getString(String key) ;
   /** Get the object for a given key */
   public Object getObject(String key);
}
```



```
class PropertyResourceBundle
```

```
abstract class ListResourceBundle {
   abstract Object[][] getContents();
}
```

Lokalizace v Javě



Jak identifikujeme správnou Resource Bundle?

```
public static ResourceBundle getBundle(String baseName, Locale targetLocale,
                                       ClassLoader loader, Control control)
```

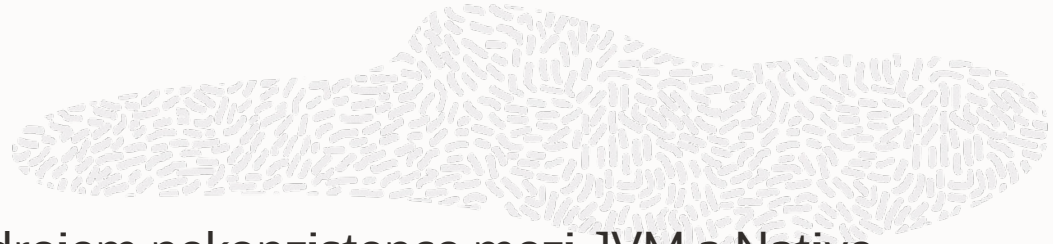
- Pomerně komplexní implementace.
- Využívá reflexi. 

Lokalizace v Native Image

1. Všechny **ResourceBundles** dostupné za běhu musí být inicializovány při překladu.
2. Objekty uloženy v **mapě**.
3. Kód ResourceBundle.getBundle() **substituován**.

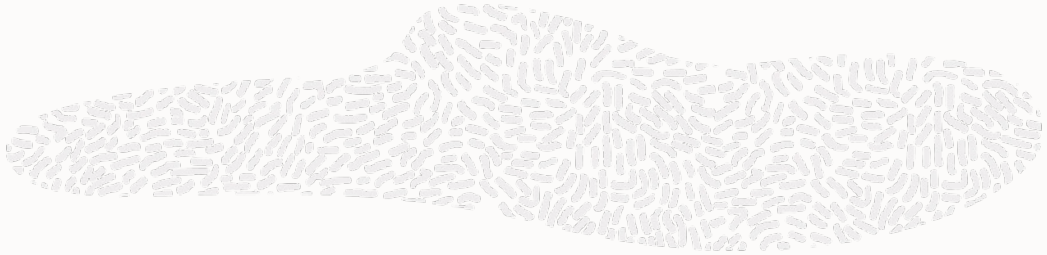
```
@TargetClass(java.util.ResourceBundle.class)
class Target_java_util_ResourceBundle {
   @Substitute
   ResourceBundle getBundle(…) {…}
}
```

Problémy lokalizace



- Substituování ResourceBundle.getBundle() může být zdrojem nekonzistence mezi JVM a Native Image.
  - Implementujeme vyhledávání opravdu totožným způsobem?
- Ne všechny **ResourceBundles** mohou být inicializovány v době překladu.
  - Riziko selhání překladu s kryptickým chybovým hlášením.

“Můžeme využít původní JDK kód místo substituce?”

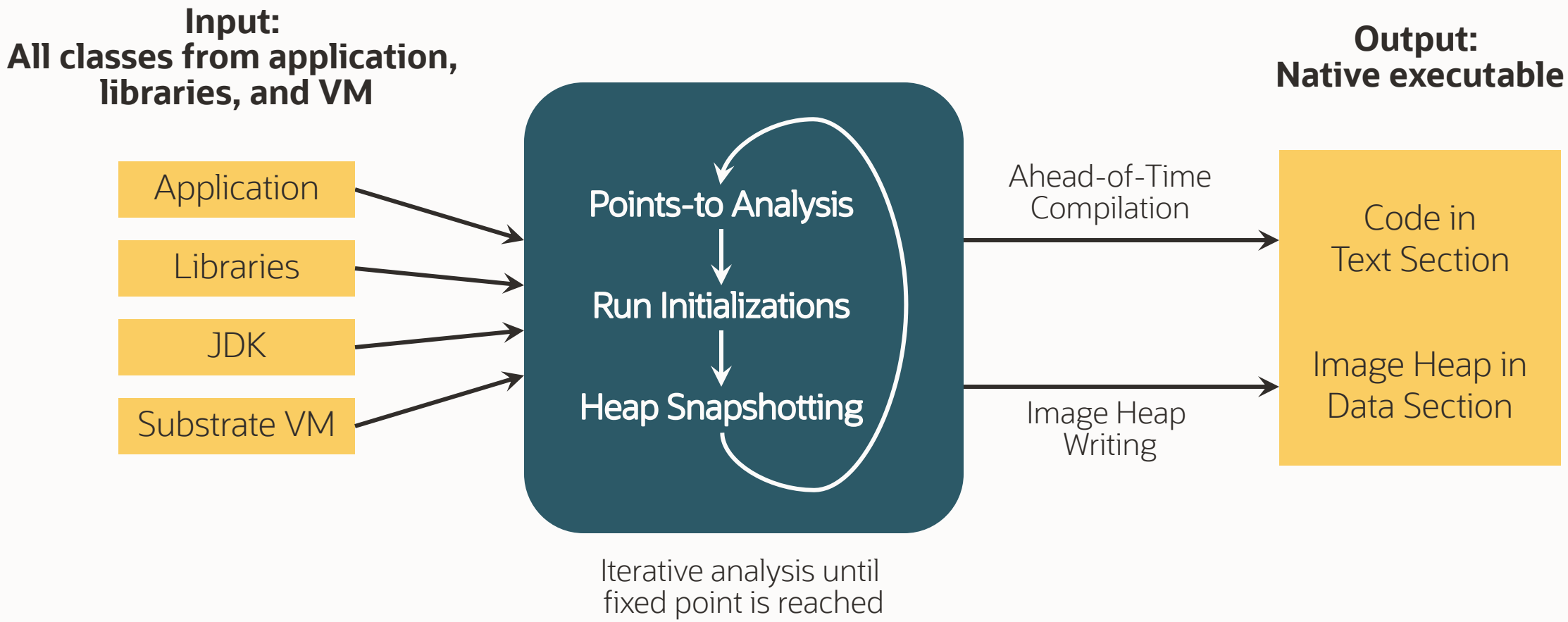


Jak jsem vyrobil 300MB HelloWorld...

```
class FormatData_af extends ListResourceBundle {
   Object[][] getContents() {...}
}
```

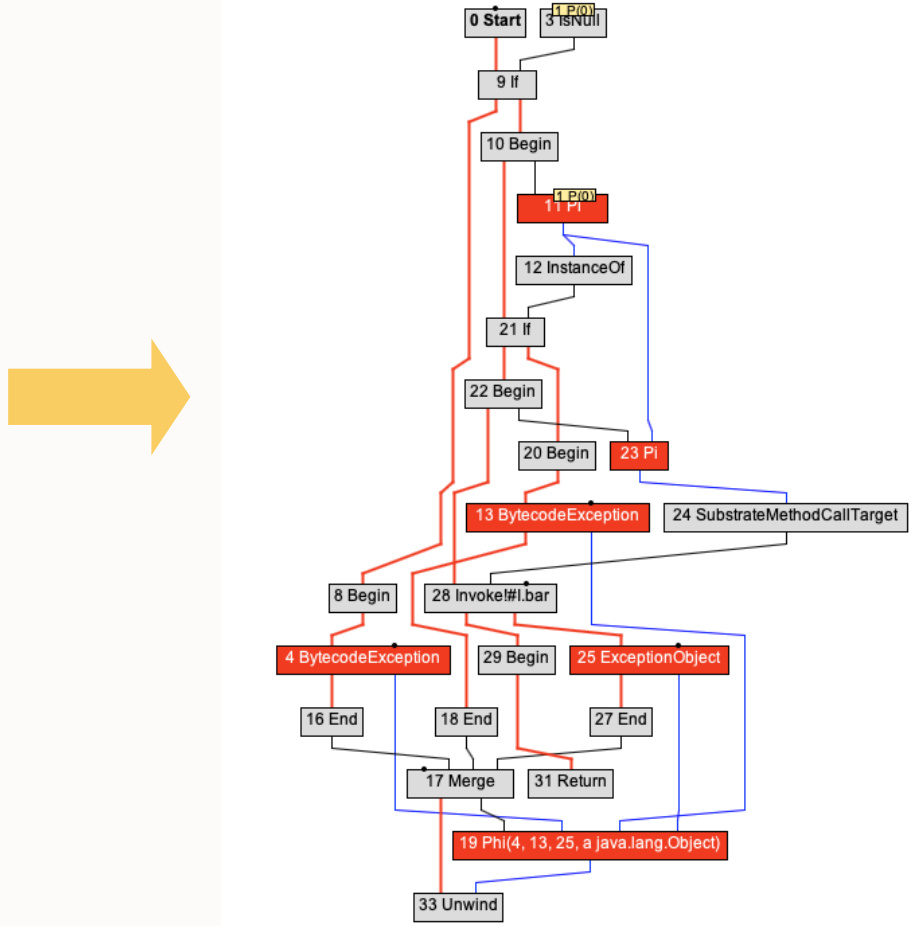
```
@TargetClass(java.util.ListResourceBundle.class)
class Target_java_util_ListResourceBundle {
   @Substitute
   private void loadLookup() {...}
}
```

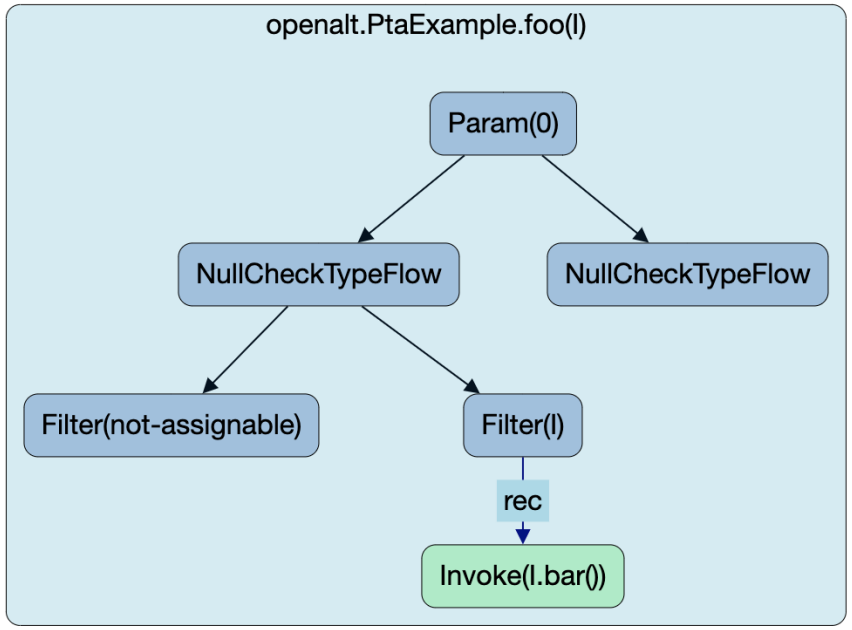
Native Image Build Process



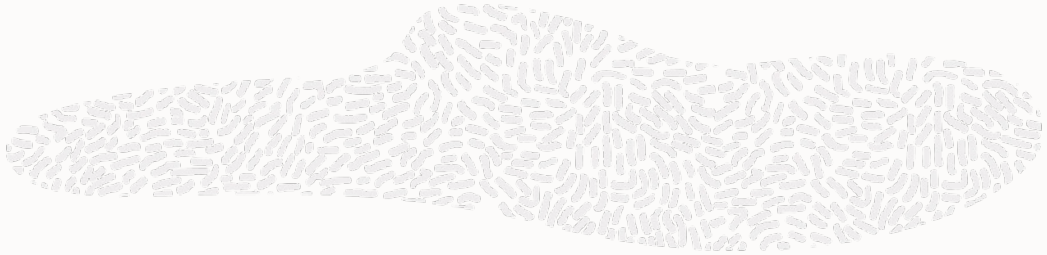
Points-to analýza pod pokličkou

```
void foo(I i) {
    i. bar();
}
```





Analýza Spring Petclinic trvá přes 90 vteřin...



Můžeme použít jinou metodu?

Comparing Rapid Type Analysis with Points-To Analysis in GraalVM Native Image

David Kozak
ikozak@fit.vut.cz
Brno University of Technology
Czechia

Vojin Jovanovic
vojin.jovanovic@oracle.com
Oracle Labs
Switzerland

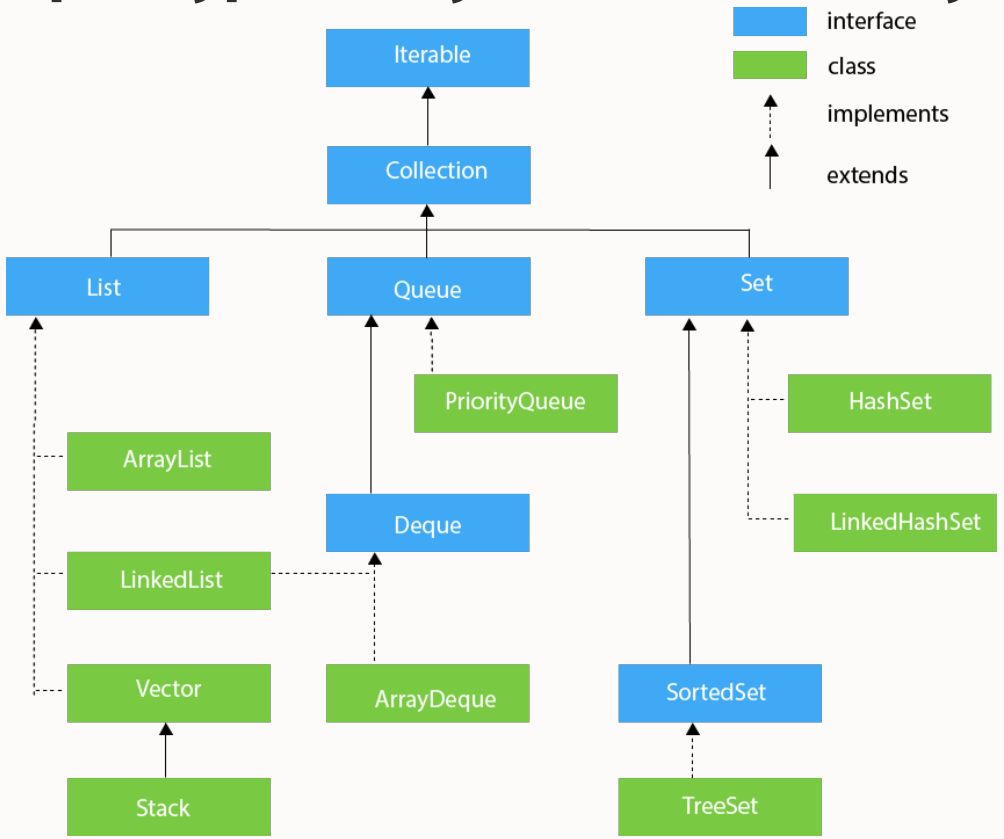
Codrut Stancu
codrut.stancu@oracle.com
Oracle Labs
Switzerland

Tomáš Vojnar
vojnar@fit.vut.cz
Brno University of Technology
Czechia

Christian Wimmer
christian.wimmer@oracle.com
Oracle Labs
USA

https://dl.acm.org/doi/abs/10.1145/3617651.3622980

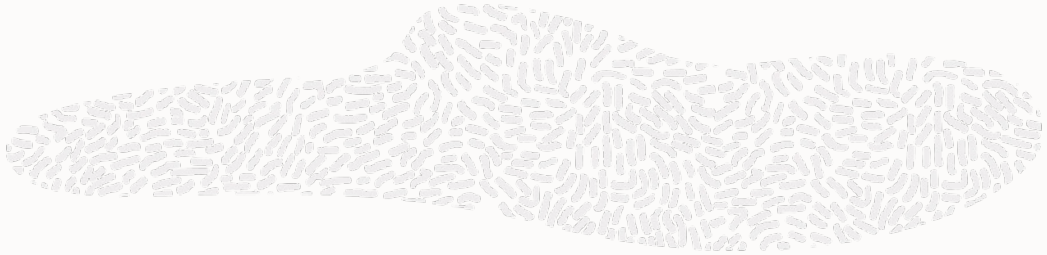
Rapid type analýza – základní myšlenka



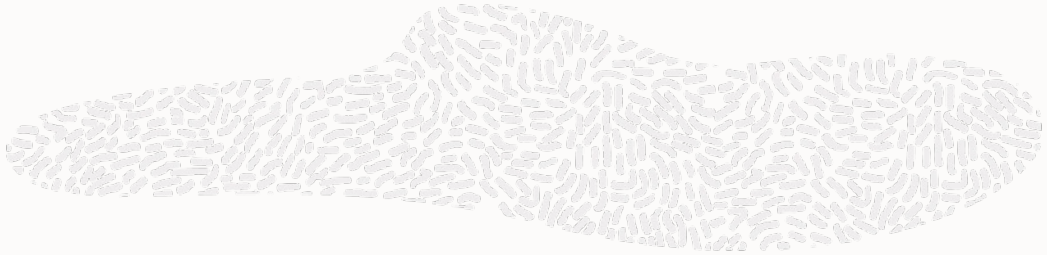

```
List<Object> list = new ArrayList<>();
```

Typová hierarchie + Instanciované typy

Suite	Benchmark	Reachable Methods PTA	Reachable Methods RTA	Analysis Time (s) PTA	Analysis Time (s) RTA	Total time (s) PTA	Total time (s) RTA	Binary size (MB) PTA	Binary size (MB) RTA
Console	helloworld	18	**+17%**	14	**+21%**	36	**+17%**	13	**+23%**
Dacapo	avrora	24	**+25%**	12	-8%	51	+6%	23	**+30%**
	fop	94	+4%	46	-30%	128	-10%	105	+11%
	jython	71	+8%	55	**-35%**	140	**-26%**	134	+9%
	luindex	26	+23%	13	-8%	54	**+7%**	32	+25%
Microservices	micronaut-helloworld-wrk	74	+4%	34	-32%	88	-9%	45	+18%
	mushop:order	168	+2%	102	-59%	209	-30%	104	+13%
	mushop:payment	82	+4%	36	-33%	91	-10%	50	+14%
	mushop:user	115	+3%	57	-44%	135	-18%	76	+13%
	petclinic-wrk	207	+4%	159	**-64%**	297	**-35%**	144	+15%
	quarkus-helloworld-wrk	52	+6%	18	-22%	69	-3%	50	+4%
	quarkus:registry	111	+5%	49	-39%	126	-16%	69	**+19%**
	spring-helloworld-wrk	67	+4%	30	-33%	87	-10%	47	+13%
	tika-wrk	82	**+6%**	29	-28%	117	-6%	88	+6%
Renaissance	chi-square	173	+8%	129	-60%	260	-30%	100	+17%
	dec-tree	324	+6%	2009	-95%	X	X	X	X
	future-genetic	27	**+22%**	15	0%	44	+5%	19	**+21%**
	gauss-mix	189	+8%	146	-61%	286	**-32%**	107	+17%
	log-regression	334	+7%	2215	**-95%**	X	X	X	X
	page-rank	171	+8%	129	-60%	258	-31%	119	+13%
	reactors	30	+13%	19	+16%	47	+11%	19	+21%
	scala-stm-bench7	30	+20%	19	**+26%**	49	**+14%**	19	+21%



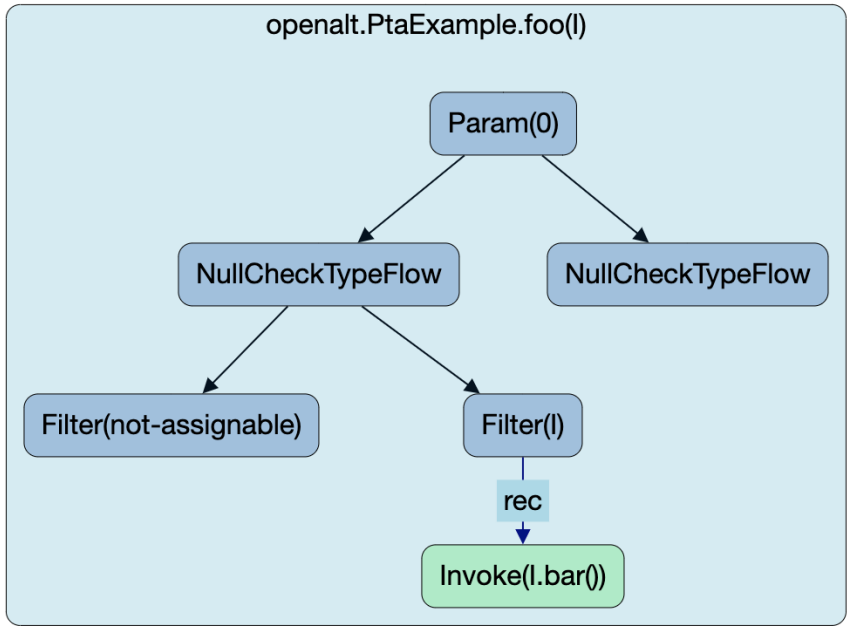
Super, vyhrála RTA?

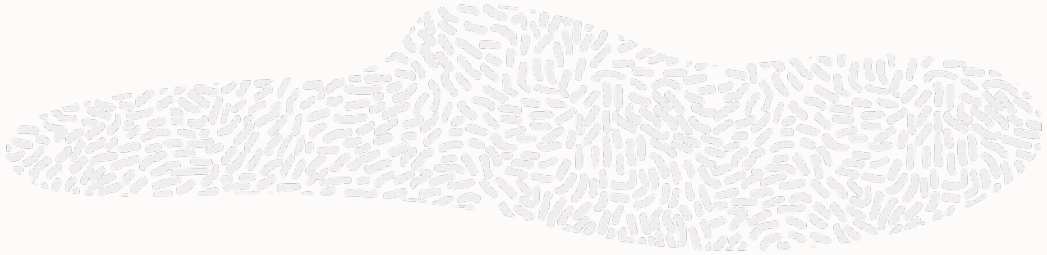


Ne tak docela...

Záleží na tom, kolik typů je v našem invoke?

- Malý počet je pro překladač zajímavý.
- 20 vs 25 už tolik ne…

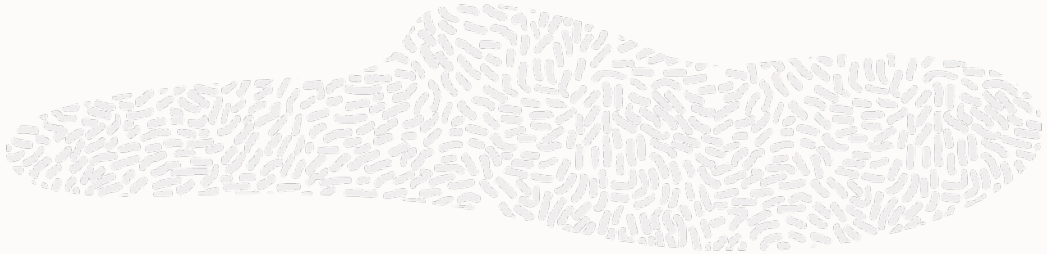




If you can’t beat them…

…join them!

Závěr



- GraalVM
- Native Image
- Zkušenosti ze stáže v Native Image
  - GraalVM je velký projekt – **možnosti stáže jsou velmi rozmanité**
- Možnosti do budoucna
  - Specializované prezentace
  - GraalVM, Truffle, GraalJS, GraalOS, Graal Could Next, Micronaut

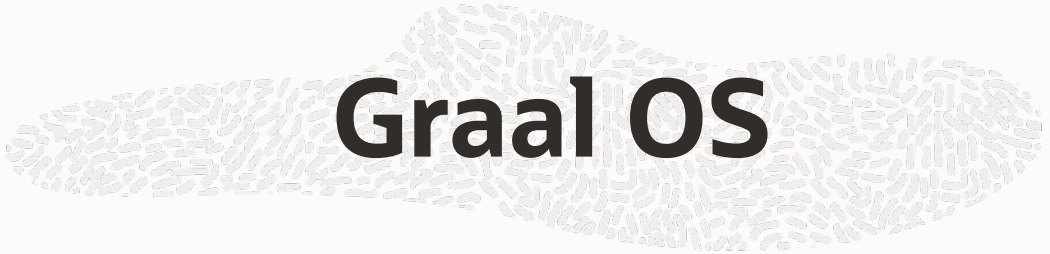
Děkujeme za pozornost

Q & A

Jak jsem se ke GraalVM dostal?

+

Graal Cloud Native



“Build portable cloud native Java microservices that start instantly and use fewer resources to reduce compute costs.”

“High-performance serverless application deployment platform.”

https://graal.cloud/