ssw.jku.at

graalvm.org

Interoperability between Execution Modes on GraalVM

Christoph Pichler
Johannes Kepler University
christoph.pichler@jku.at

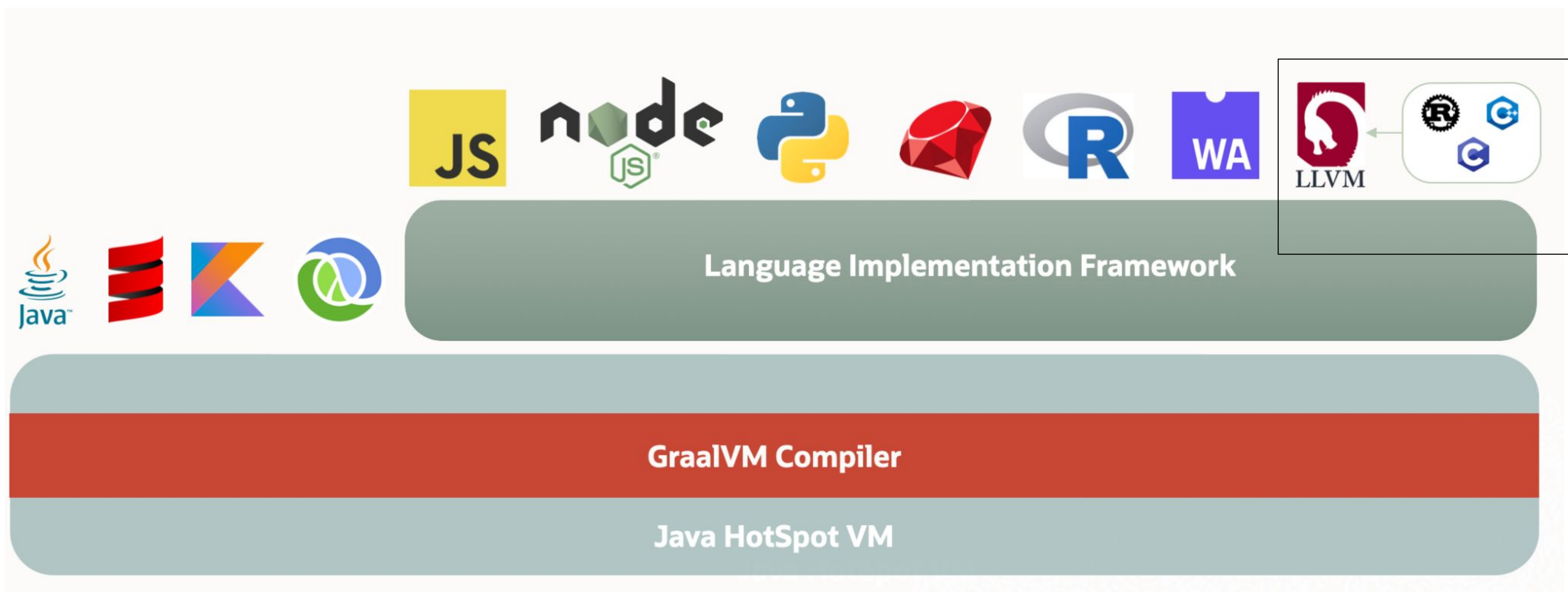
Paley Li
Oracle Labs
paley.li@oracle.com

Roland Schatz
Oracle Labs
roland.schatz@oracle.com

Hanspeter Mössenböck
Johannes Kepler University
hanspeter.moessenboeck@jku.at

GraalVM

- Polyglot virtual machine
  - No cross-language overhead
- Highly-optimizing JIT-compiler
  - Dynamic and speculative



➔ Execution of C code via clang + LLVM bitcode

Why not only native execution (C code)?

Python/JS/… pseudo code

```
p = Point(x: 4, y: -3)
diff = calcDiff(p)
```

passing managed (e.g. JS) object to C code

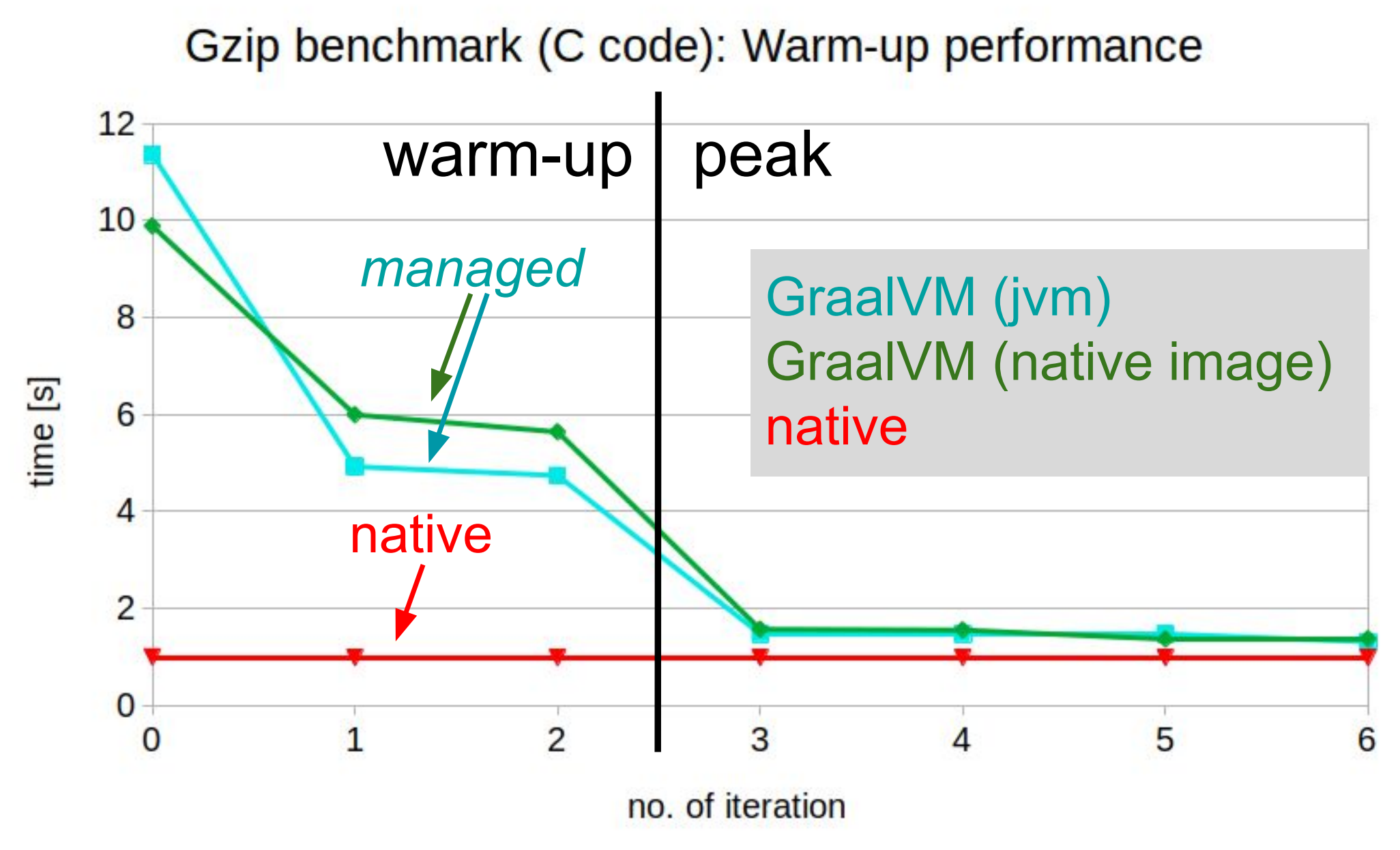
C code

```
double calcDiff(struct Point *p) {
  return sqrt(p->x*p->x+p->y*p->y);
}
```

"ERROR: managed object p not accessible in (natively executed) C code"

- Existing polyglot systems: Overhead for cross-language calls

Why not all functions on GraalVM?

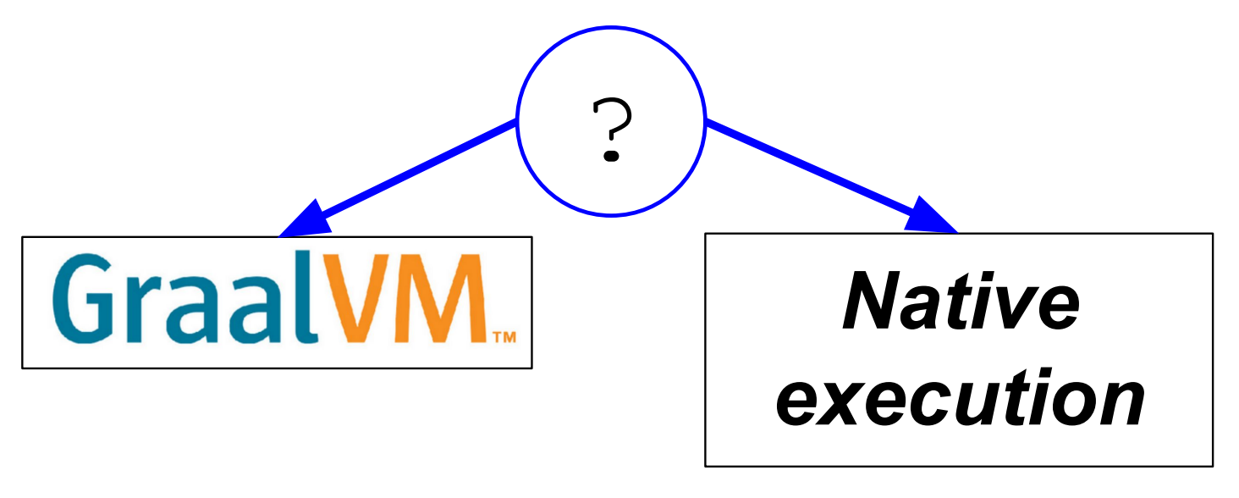


- Slow warm-up for managed execution in GraalVM (due to dynamic compilation)
- Thus: Less managed/more native code improves (warm-up) performance

Our approach: Hybrid execution

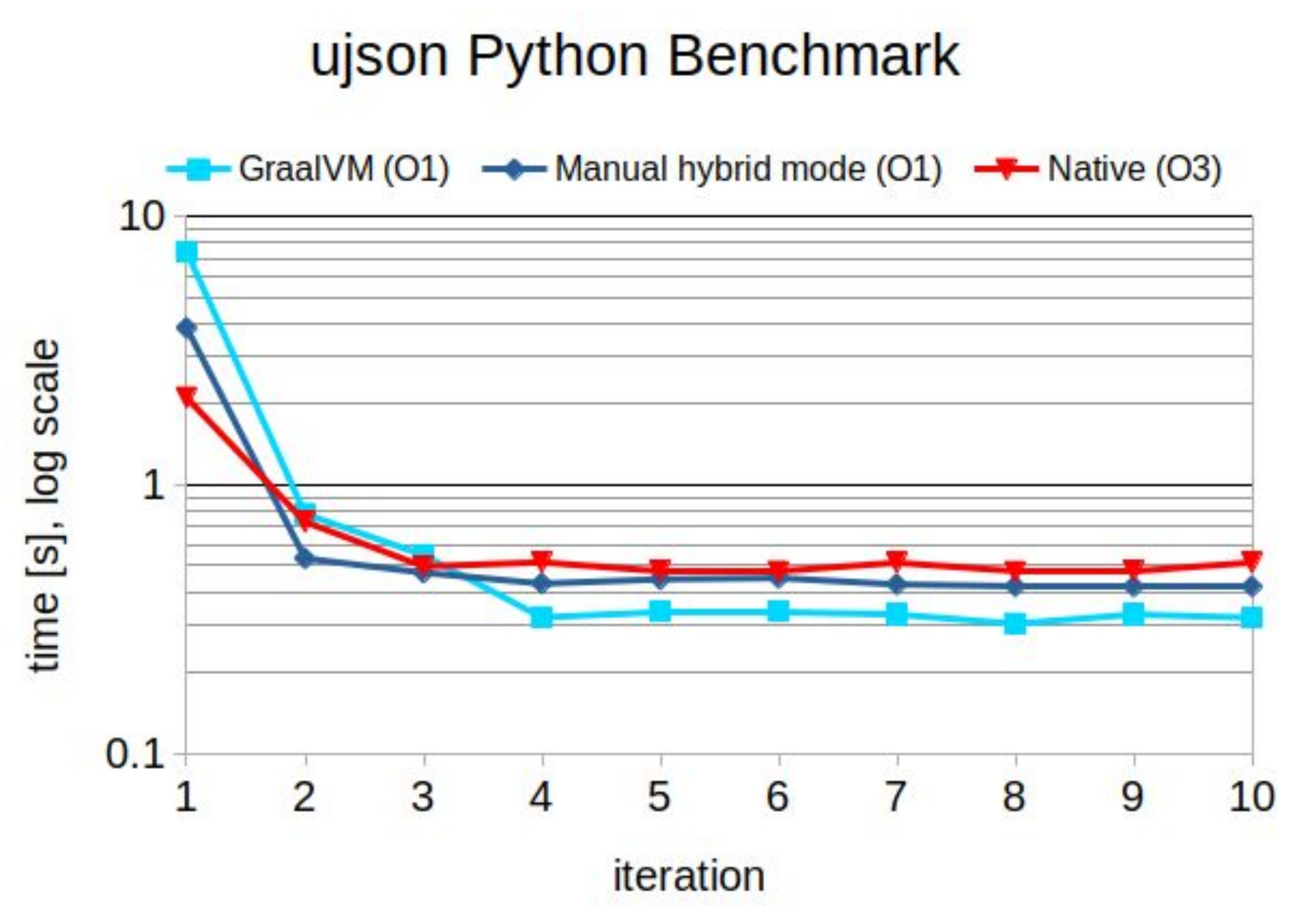
→(let) decide execution mode per callee

```
for each call of C function f:
  decide how f is executed
```



- e.g. Python/numpy:
  - Few functions access managed objects
  - Majority of C code can be run without GraalVM
- Goal: automatic decision at run time (based on call frequency, function size, managed accesses)

Current state: Results and restrictions



- Benchmark: JSON parser written in Python and C
- Forcing 3 (leaf) functions to run natively already improves warm-up (1.7x) performance
- Restrictions in current state
  - Manual decision how a callee is run (GraalVM/native)
  - Pure functions only

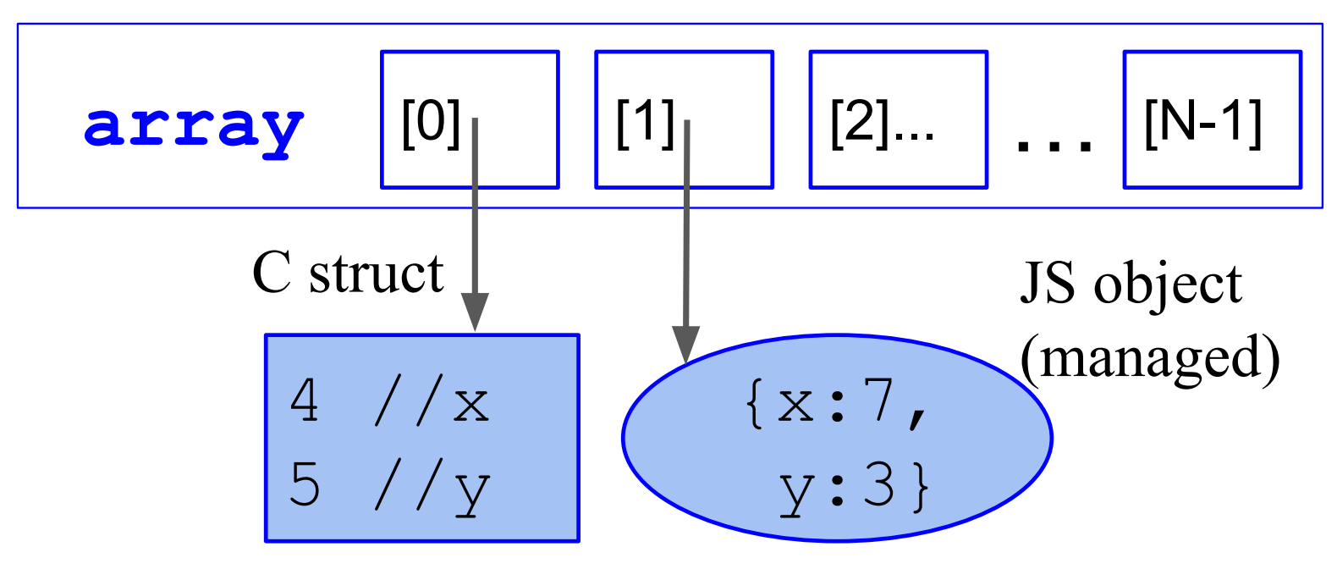
GraalVM: all (Python/C) functions run on GraalVM, inlining enabled.
Manual hybrid mode: 3 often called C functions are executed natively, all others on GraalVM. No inlining.
Native: GraalVM accesses natively compiled C code (no polyglot access possible)

Arising problems for switching the execution mode

No possibility to statically detect managed access

- Native execution whenever possible
- However: Managed access must be done on GraalVM
- Statically undecidable if native function accesses managed objects (cf. example below) → switch at run time necessary!

```
struct Point* array[] = …
for(int i=0; i < N; i++) {
  sumY += array[i]->y;
}
```



Functions with side effects on global variables

- Native globals cannot store GraalVM data
- GraalVM globals cannot be accessed from native code
- Enabling both global types at the same time lets globals exist twice →Might/Will lead to inconsistency for impure functions!

```
static int nElems;

void  add(...) {nElems++;...}

void* remove() {nElems--;...}
```

Instruction	Execution mode	native value of ***nElems***	GraalVM value of ***nElems***	correct value of ***nElems***
…	…	0	0	0
`add(...)`	GraalVM	0 ⚠	1	1
`remove()`	native	-1 ⚠	1 ⚠	0

References

- graalvm.org
- Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko. 2013. One VM to rule them all. In Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software (Onward! 2013). Association for Computing Machinery, New York, NY, USA, 187–204. https://doi.org/10.1145/2509578.2509581
- Manuel Rigger, Matthias Grimmer, Christian Wimmer, Thomas Würthinger, and Hanspeter Mössenböck. 2016. Bringing low-level languages to the JVM: efficient execution of LLVM IR on Truffle. In Proceedings of the 8th International Workshop on Virtual Machines and Intermediate Languages (VMIL 2016). Association for Computing Machinery, New York, NY, USA, 6–15. https://doi.org/10.1145/2998415.2998416
- Manuel Rigger, Roland Schatz, Jacob Kreindl, Christian Häubl, and Hanspeter Mössenböck. 2018. Sulong, and thanks for all the fish. In Companion Proceedings of the 2nd International Conference on the Art, Science, and Engineering of Programming (Programming '18). Association for Computing Machinery, New York, NY, USA, 58–60. https://doi.org/10.1145/3191697.3191726
- T. Pittman. 1987. Two-level hybrid interpreter/native code execution for combined space-time program efficiency. In Papers of the Symposium on Interpreters and interpretive techniques (SIGPLAN '87). Association for Computing Machinery, New York, NY, USA, 150–152. https://doi.org/10.1145/29650.29666
- Manel Grichi, Mouna Abidi, Yann-Gaël Guéhéneuc, and Foutse Khomh. 2019. State of practices of Java native interface. In Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering (CASCON '19). IBM Corp., USA, 274–283.
- Matthias Grimmer, Roland Schatz, Chris Seaton, Thomas Würthinger, Mikel Luján, and Hanspeter Mössenböck. 2018. Cross-Language Interoperability in a Multi-Language Runtime. ACM Trans. Program. Lang. Syst. 40, 2, Article 8 (June 2018), 43 pages. https://doi.org/10.1145/3201898