Дмитрий
Морозовский
Huawei

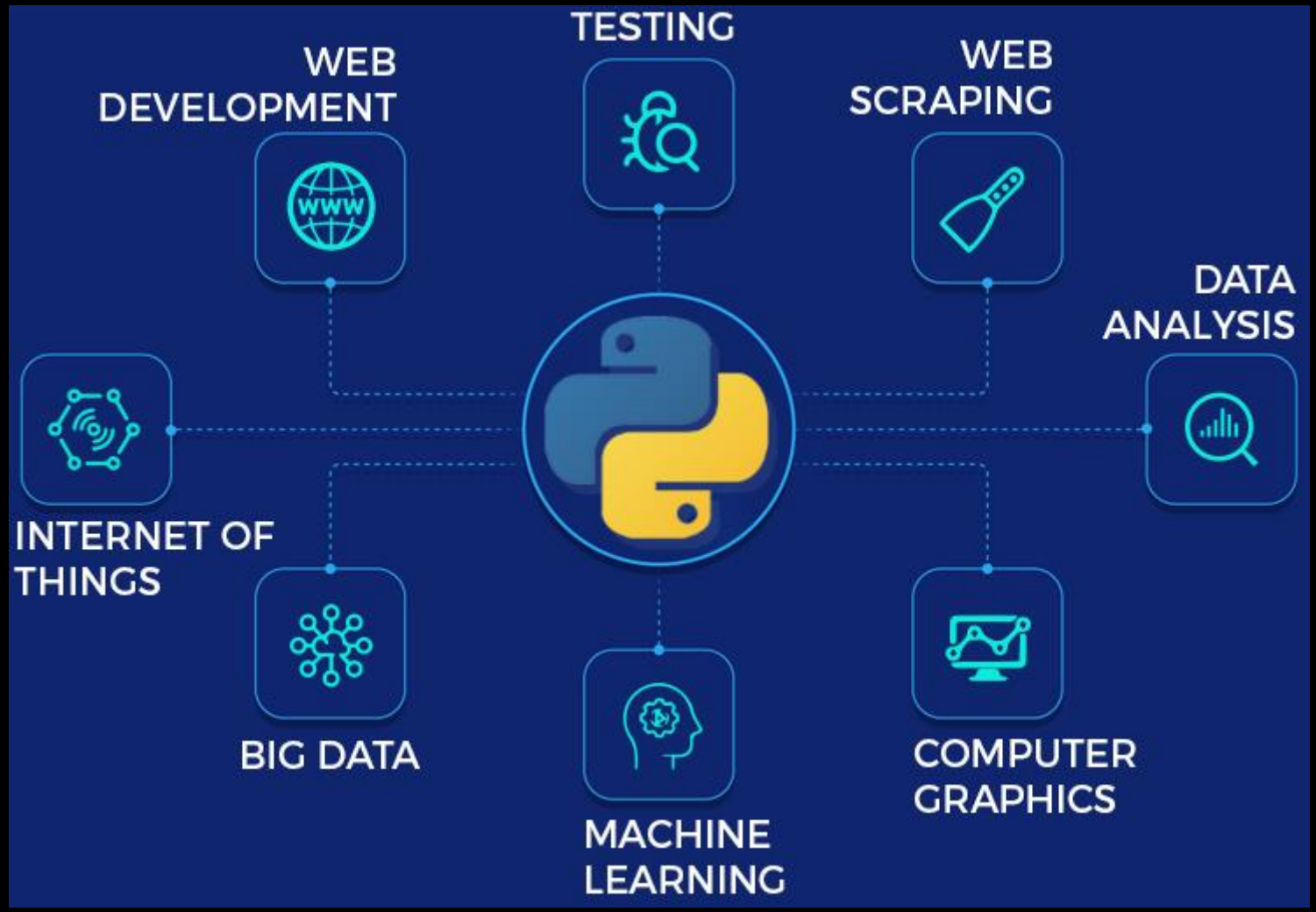
Путешествие из Java в Python

Два мира – один JEP

Структура доклада

- Зачем нужно запускать Python на Java
- Обзор инструментов взаимодействия Python и Java
- Как работает JEP
- Примеры кода и как его отлаживать.

Чем интересен Python



Из чего же сделан Python

- Основная сборка питона это **cpython**
- К нему устанавливаются дополнительные модули (pip, conda)
- В частности такие популярные модули как numpy, pandas, scikit-learn, pytorch, tensorflow, keras, opencv, scipy, matplotlib и тд
- Они зачастую написаны на C и используют python extention api
- Также есть много других рантаймов PyPy, Stackless, Micro, Pyston, Cinder
- Кросс компиляторы в другие языки Pyjs, Condon, Cython, Jython, Numba, Rpython, IronPython

Mojo 🔥 — the programming language for all AI developers.

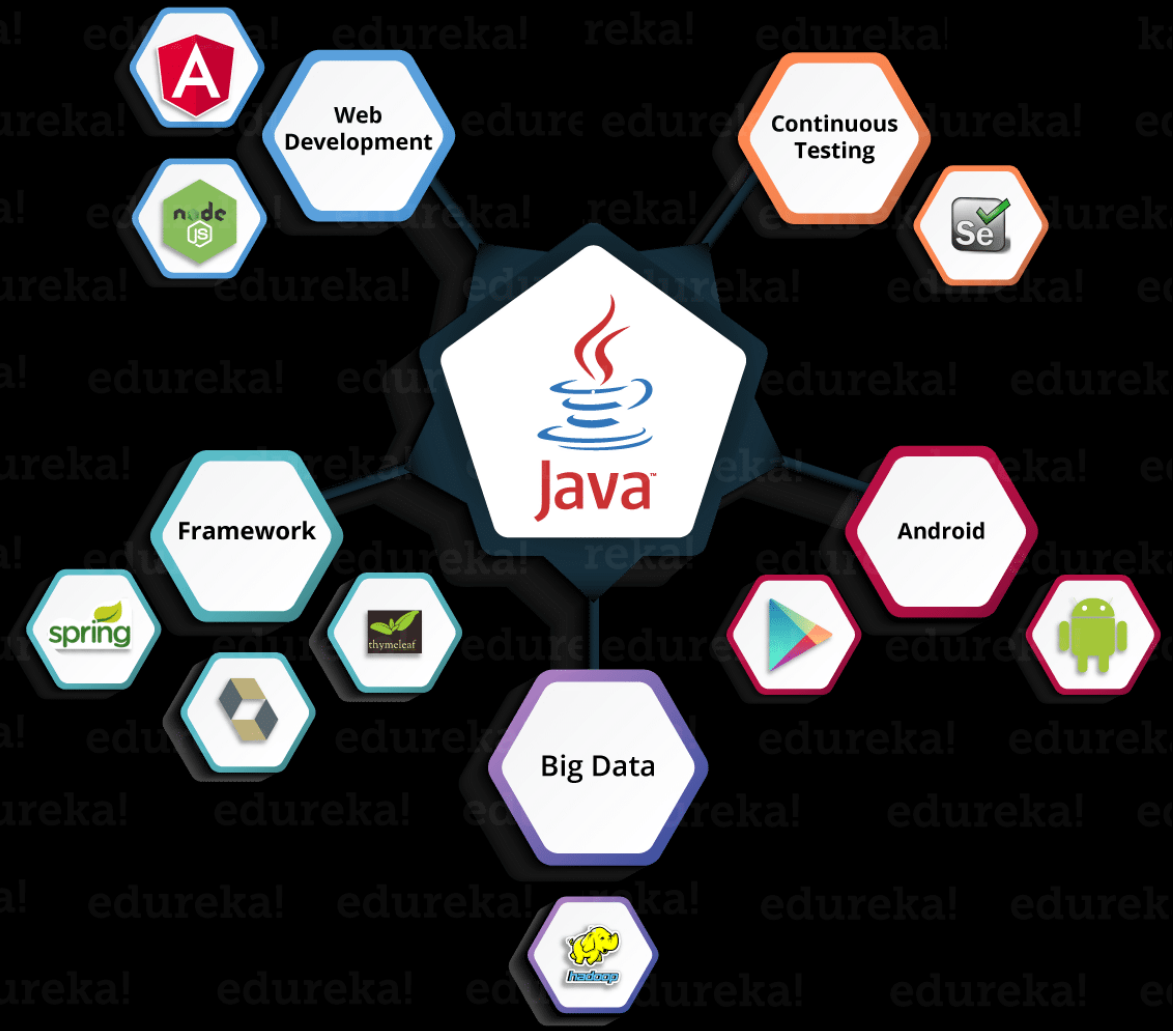
Mojo combines the usability of Python with the performance of C, unlocking unparalleled programmability of AI hardware and extensibility of AI models.

SOFTMAX.🔥

MOJO 🔥

```
def softmax(lst):
  norm = np.exp(lst - np.max(lst))
  return norm / norm.sum()

struct NDArray:
  def max(self) -> NDArray:
    return self.pmap(SIMD.max)

struct SIMD[type: DType, width: Int]:
  def max(self, rhs: Self) -> Self:
    return (self >= rhs).select(self, rhs)
```

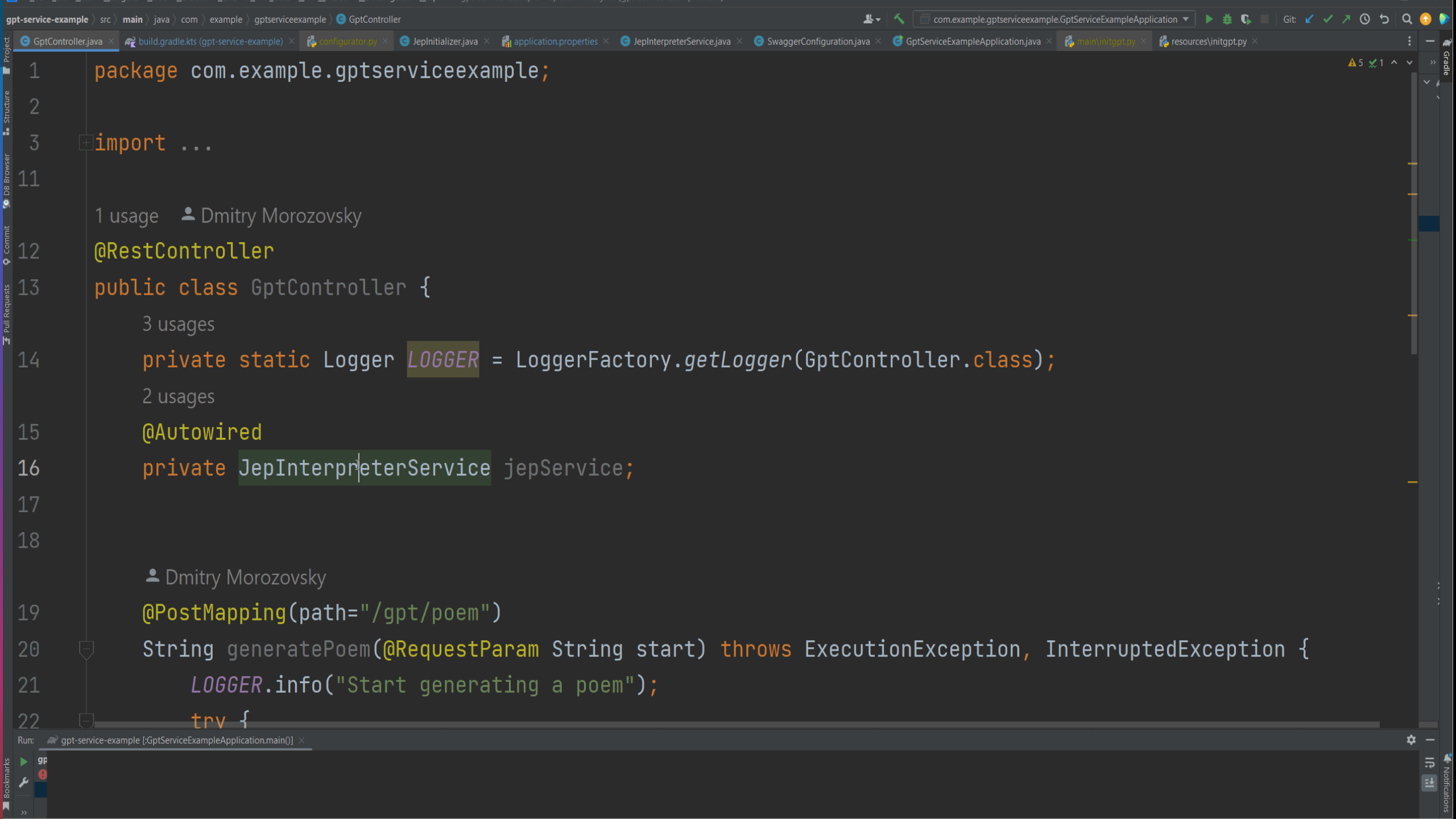
Чем интересна Java




Python
Java

Возможно ли объединить два мира?

- Такие попытки делаются давно, есть множество механизмов
- Мы можем написать с их помощью микросервис который будет нам сочинять стихи в стиле Шекспира.


gpt-service-example src main java com example gptserviceexample GptController
com.example.gptserviceexample.GptServiceExampleApplication
Git:
GptController.java
build.gradle.kts (gpt-service-example)
configurator.py
JepInitializer.java
application.properties
JepInterpreterService.java
SwaggerConfiguration.java
GptServiceExampleApplication.java
main\initgpt.py
resources\initgpt.py
5 1
package com.example.gptserviceexample;
import ...
1 usage Dmitry Morozovsky
@RestController
public class GptController {
3 usages
private static Logger LOGGER = LoggerFactory.getLogger(GptController.class);
2 usages
@Autowired
private JepInterpreterService jepService;
Dmitry Morozovsky
@PostMapping(path="/gpt/poem")
String generatePoem(@RequestParam String start) throws ExecutionException, InterruptedException {
LOGGER.info("Start generating a poem");
try {
Run: gpt-service-example [:GptServiceExampleApplication.main()]

Типы инструментов взаимодействия Python и Java

- Какой язык host а какой guest
- Схема вызова
- Тип runtime

Какой язык host а какой guest.

- Python вызывает Java:
  - Jpype, Py4J, **JCC**, **JavaBridge**, Jnius
- Java вызывает Python
  - Jython, GraalVM, JEP, PEMJA, Py4J

Схема вызова:

Межпроцессное взаимодействие (IPC)

- Взаимодействие по сети (Network):
  - Socket (PySpark Runtime)
  - Py4j (PyFlink&PySpark Client, Alink Runtime)
  - GRPC (PyFlink On Apache Beam)
- Через разделяемую память (Shared memory)
  - Tensorflow on Flink
  - PyArrow Plasma

Схема вызова:

Межпроцессное взаимодействие (IPC)

- Недостатки:
  - Overhead на передачу данных
  - Сложность управления процессами, в т.ч. в облаке (nodes).
- Приемущества:
  - большая изоляция

Схема вызова:
Python работающий на JVM

- Преобразование Python кода в Java:
  - p2j
  - voc
- Интерпретатор Python работающий на JVM
  - Jython
  - GraalVM Python extention

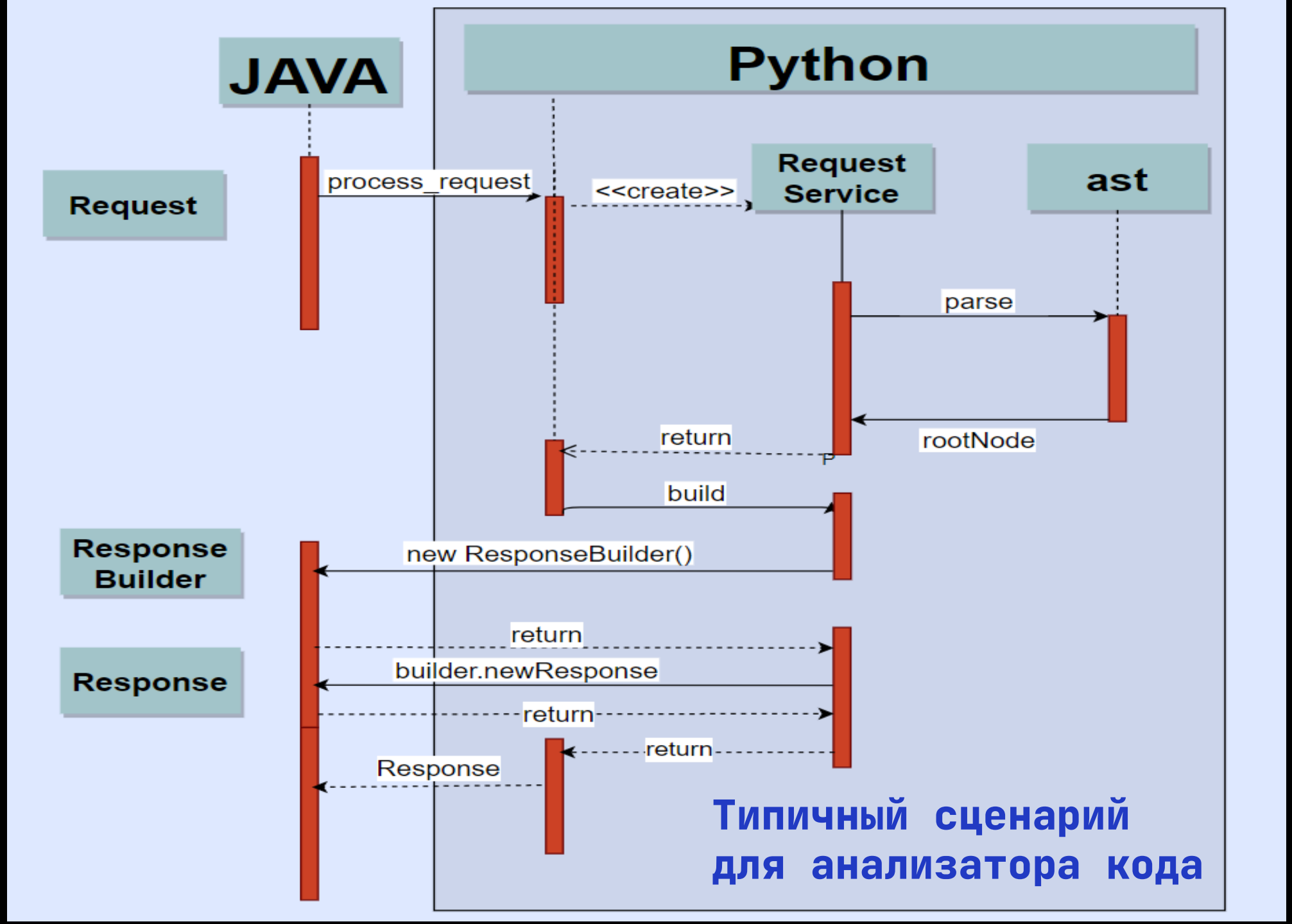
Схема вызова:

Решения на основе FFI

- Используются JNI и CFFI wrappers
- JPype - Вызов JVM из PVM
- JEP - вызов PVM из JVM
- PEMJA - вызов PVM из JVM

Инструменты запуска Python из Java

- **Jython** – интерпретатор питона на Java
- **GraalVM Python** – специальный runtime python для GraalVM который можно вызывать из Java на GraalVM.
- **Java Embedded Python(JEP)** – библиотека позволяющая запускать cpython интерпретатор внутри JVM процесса
- **Python Embedded In Java (PEMJA)** – примерно то же что и JEP, достаточно новая библиотека от Alibaba, обгоняющая JEP по performance в отдельных случаях.


JAVA
Python
Request
process_request
<<create>>
Request Service
ast
parse
rootNode
return
build
Response Builder
new ResponseBuilder()
return
Response
builder.newResponse
return
return
Response
Типичный сценарий для анализатора кода

Код на Python для вызова из JVM

Example.py

```python
from ExampleModule import PythonService


def process_request(request):
    service = PythonService(request)
    service.build()

    response = service.response

    return response
```

ExampleModule/__init__.py

```python
from com.icemachined import ResponseBuilder
from com.icemachined import Request
import ast

class PythonService:
    def __init__(self, request):
        rootNode = ast.parse(request.getRequest())
        self.request = str(rootNode.body[0].name)


    def build(self):
        builder = ResponseBuilder()
        self.response = builder.newResponse( Request(self.request))
```

Jython

- Это довольно старый стабильный интерпретатор написанный на Java
- Позволяет плотно взаимодействовать с java code
- В пакет входят большинство стандартных библиотек написанных на python не использующих cpython extensions.
- У него есть некоторные отличия по сравнению с оригинальным python
- Тк написан на java то можно пользоваться приемуществами java multithreading и garbage collection
- Большой недостаток в том что последняя поддерживаемая версия python 2.7, что делает его непригодным к задаче с анализом кода

```
val properties = Properties()
val pythonPath = File(JepInitializer.javaClass.getResource("/example.py").file).canonicalPath
properties.setProperty("python.path", pythonPath)
PythonInterpreter.initialize(System.getProperties(), properties, arrayOf(""))
val interp = PythonInterpreter()

interp.execfile(pythonPath)


val pr = interp.get("process_request") as PyFunction
val request = Request(
    """
    def foo():
        x = source()
        if x < MAX:
            y = 2 * x
            sink(y)
    """.trimIndent()
)

val result = pr.__call__(Py.java2py(request)).__tojava__(Response::class.java) as Response
println("Response: ${result.response}")

interp.set("myparam", request)  // global variables can lead to memory leaks
val result1 = interp.eval("process_request(myparam)" ).__tojava__(Response::class.java) as
Response
println("Response1: ${result1.response}")
```

Jython вызов из JVM

Graalvm python runtime

- Принцип действия – общий IR.
- Требует установки специальных рантаймов
- Отстает от последних версий cpython.
- Слабая поддержка Windows.
- Debug местами очень тормозит,
- Отсутствует поддержка некоторых функций Poliglot API.
- Первая цель: то ли POC для numpy, то ли поддержка PyTorch и SciPy

Код на Graal Polyglot API

```
URL exampleUrl = PolyglotPythonExample.class.getResource("example.py");
Source exampleSrc = Source.newBuilder("python", exampleUrl).build();
String rootPath = Paths.get(exampleUrl.toURI()).getParent().toString();
Context ctx = Context.newBuilder().allowAllAccess(true)
                                  .option("python.PythonPath", rootPath).build();

ctx.eval(exampleSrc);

Value processRequest = ctx.getBindings("python").getMember("process_request");
Request request = new Request(
    "def foo():\n" +
    "    x = source()\n" +
    "    if x < MAX:\n" +
    "        y = 2 * x\n" +
    "        sink(y)\n");
Response r = processRequest.execute(request).as(Response.class);

System.err.println(r);
```

Особенность Python кода для GraalVM

ExampleModule/__init__.py

```
import ast
import java

class PythonService:
    def __init__(self, request):
        rootNode = ast.parse(request.getRequest())
        self.request = str(rootNode.body[0].name)


    def build(self):
        ResponseBuilder = java.type('org.graalvm.demos.ResponseBuilder')
        Request = java.type('org.graalvm.demos.Request')
        builder = ResponseBuilder()
        self.response = builder.newResponse( Request(self.request))
```

Java Embedded Python (JEP)

- Запуск cpython внутри JVM, вызов кода друг друга.
- Работа с обьектами java в python коде и python обьектами в java
- Код питона выполняется в родной для него среде cpython
- Работают все модули с cpython extentions

Особенности инициализации JEP

JepInitializer.kt

```
val classLoader = javaClass
val packageName = "ExampleModule"
val packageInitFile = classLoader.getResource("/$packageName/__init__.py")

config.redirectStdErr(System.err)
config.redirectStdout(System.out)

config.addIncludePaths(Paths.get(packageInitFile.toURI()).parent.parent.toString())


val jepLibrary = if (System.getenv("JEP_LIBRARY_PATH") != null) {
    File(System.getenv("JEP_LIBRARY_PATH"))
} else {
    val jepRoot = Paths.get(packageInitFile.toURI())
                       .parent.parent.parent.resolve("jep-distro").resolve("jep")
    when (getOS()!!) {
        OS.WINDOWS -> File(jepRoot.resolve("jep.dll").toString())
        OS.LINUX -> File(jepRoot.resolve("libjep.so").toString())
        OS.MAC -> File(jepRoot.resolve("libjep.jnilib").toString())
    }

}

MainInterpreter.setJepLibraryPath(jepLibrary.path)
config.addIncludePaths(
    jepLibrary.toPath().parent.parent.toString()
)
```

JEP: вызов Python кода из JVM

```
println("Initializing interpreter")

val interp = SubInterpreter(JepInitializer.config)

val initScript =
File(JepInitializer.javaClass.getResource("/example.py").file).canonicalPath
interp.runScript(initScript)

val request = Request(
    """
    def foo():
        x = source()
        if x < MAX:
            y = 2 * x
            sink(y)
    """.trimIndent()
)
val res = interp.invoke("process_request", request) as Response
print("res = $res")
```

Python Embedded In Java (PEMJA)

- Новый фреймворк для pyflink
- Тот же что и JEP но с упором на производительность
- В отличие от JEP имеет дистрибутивы в бинарном виде
- Всего лишь 77 звезд на github, в то время как JEP 1.1k
- Не поддерживает windows
- Не поддерживает взаимодействие с numpy и много из того что есть в JEP.

Особенности Python кода для PEMJA

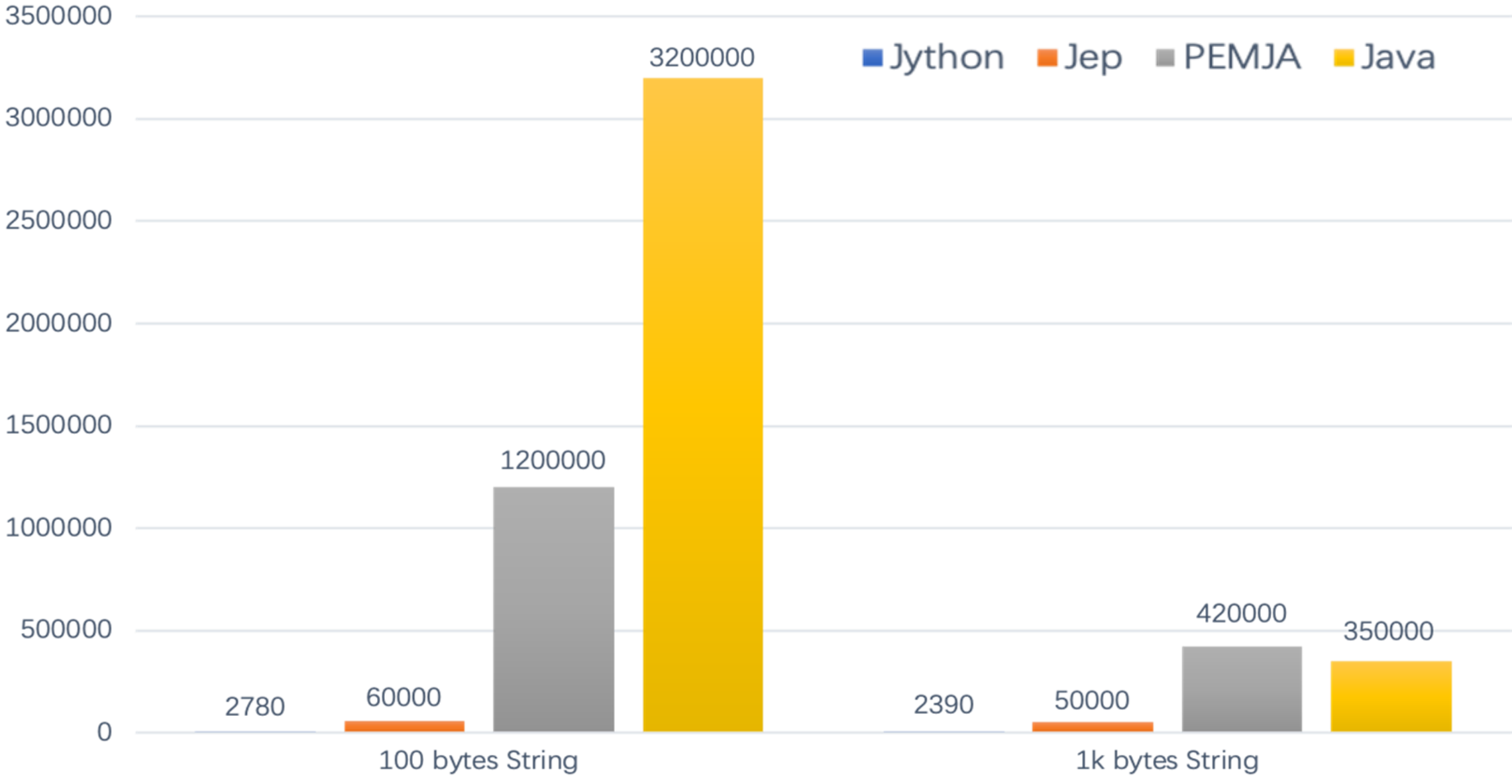
ExampleModulePemja/__init__.py

```python
from pemja import findClass
import ast

class PythonService:
    def __init__(self, request):
        rootNode = ast.parse(request.getRequest())
        self.request = str(rootNode.body[0].name)


    def build(self):
        ResponseBuilder = findClass('com.icemachined.ResponseBuilder')
        Request = findClass('com.icemachined.Request')

        builder = ResponseBuilder()
        self.response = builder.newResponse( Request(self.request))
```

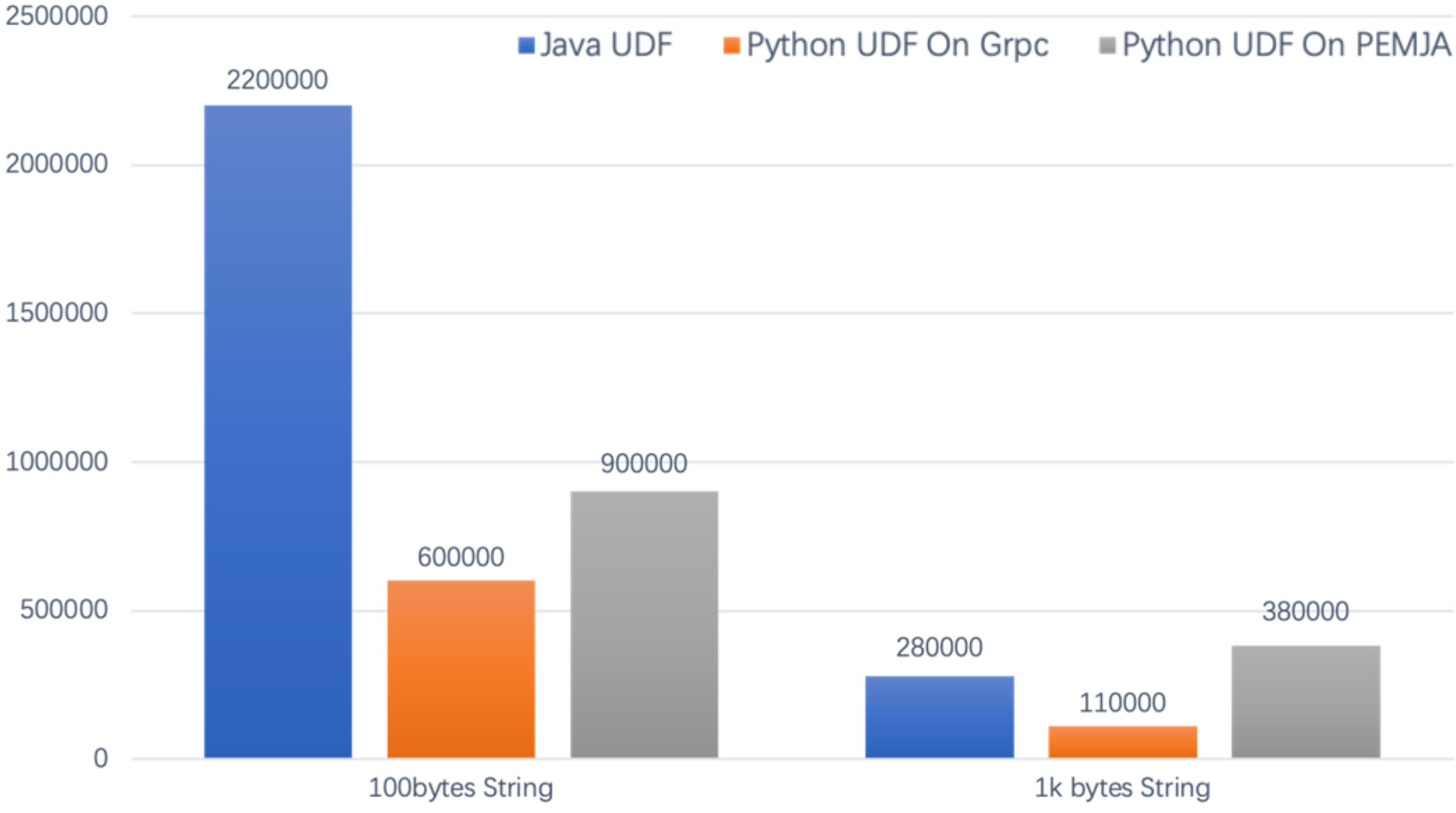
PEMJA: вызов Python кода из JVM

```
val pythonPath = File(JepInitializer.javaClass.getResource("/example_pemja.py").file)
val config = PythonInterpreterConfig
    .newBuilder()
    .setPythonExec("python3") // specify python exec
    .addPythonPaths(pythonPath.parent) // add path to search path
    .build()
val interpreter = PythonInterpreter(config);

interpreter.exec("import example_pemja")

val request = Request(
    """
    def foo():
        x = source()
        if x < MAX:
            y = 2 * x
            sink(y)
    """.trimIndent()
)
val res = interpreter.invoke("example_pemja.process_request", request);
print("res = $res")
```

Производительность “string.toUppercase”



Сравнение производительности между Inprocess и Outprocess интерпретатором



Java Embedded Python (JEP)

Особенности установки

- Установка pip install jep из исходников.
- Наличие build tools тулы для соответствующей операционной системы
- Правильная dev сборка cpython с хедерами, JDK.
- Проблемы с определенными версиями Cloud Native Buildpacks

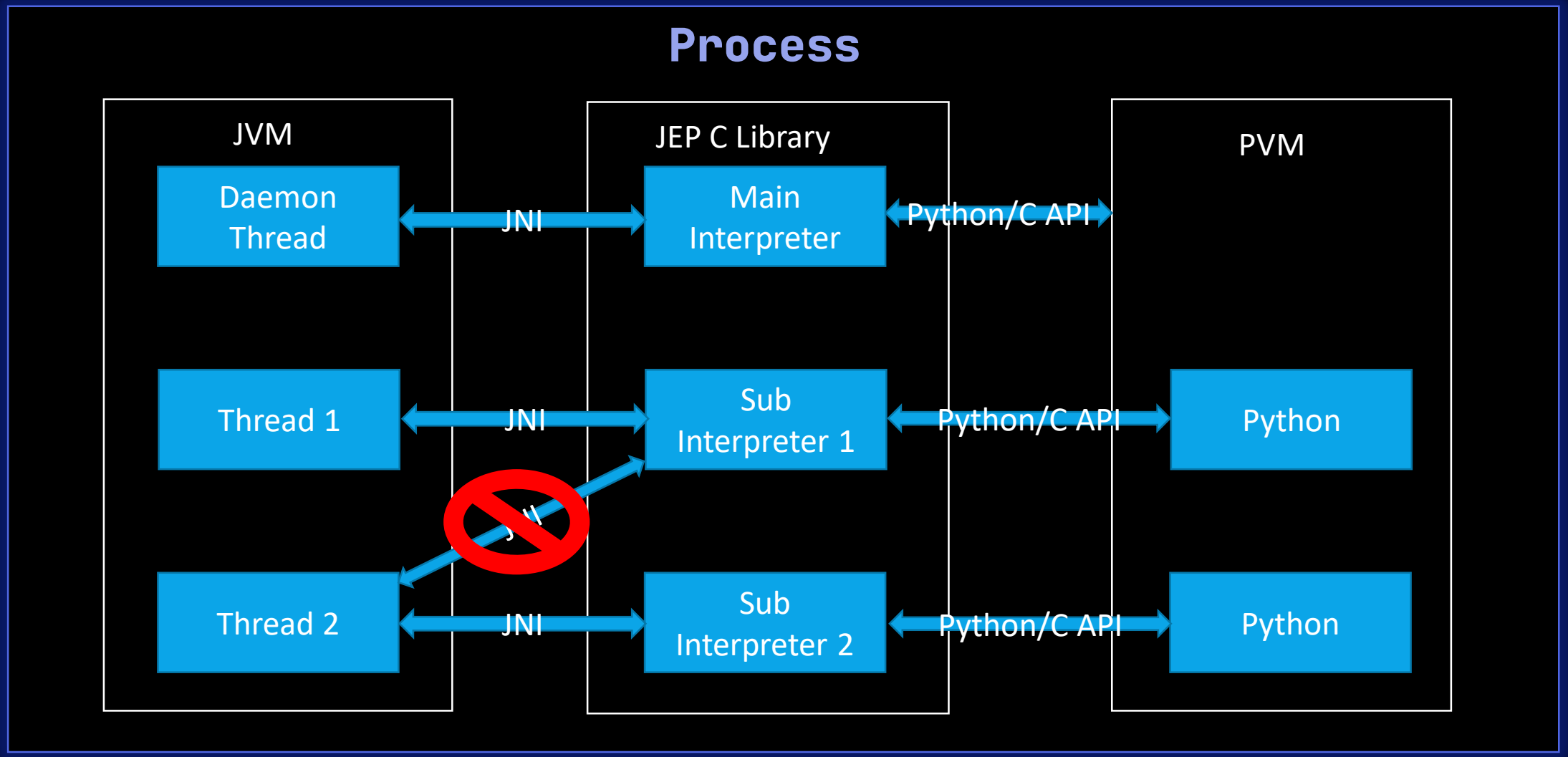
Java Embedded Python (JEP)

Особенности установки

- Я решил эту проблему создав сборку нативных библиотек с публикацией архива в maven central.

Java Embedded Python (JEP)

Как работает JEP



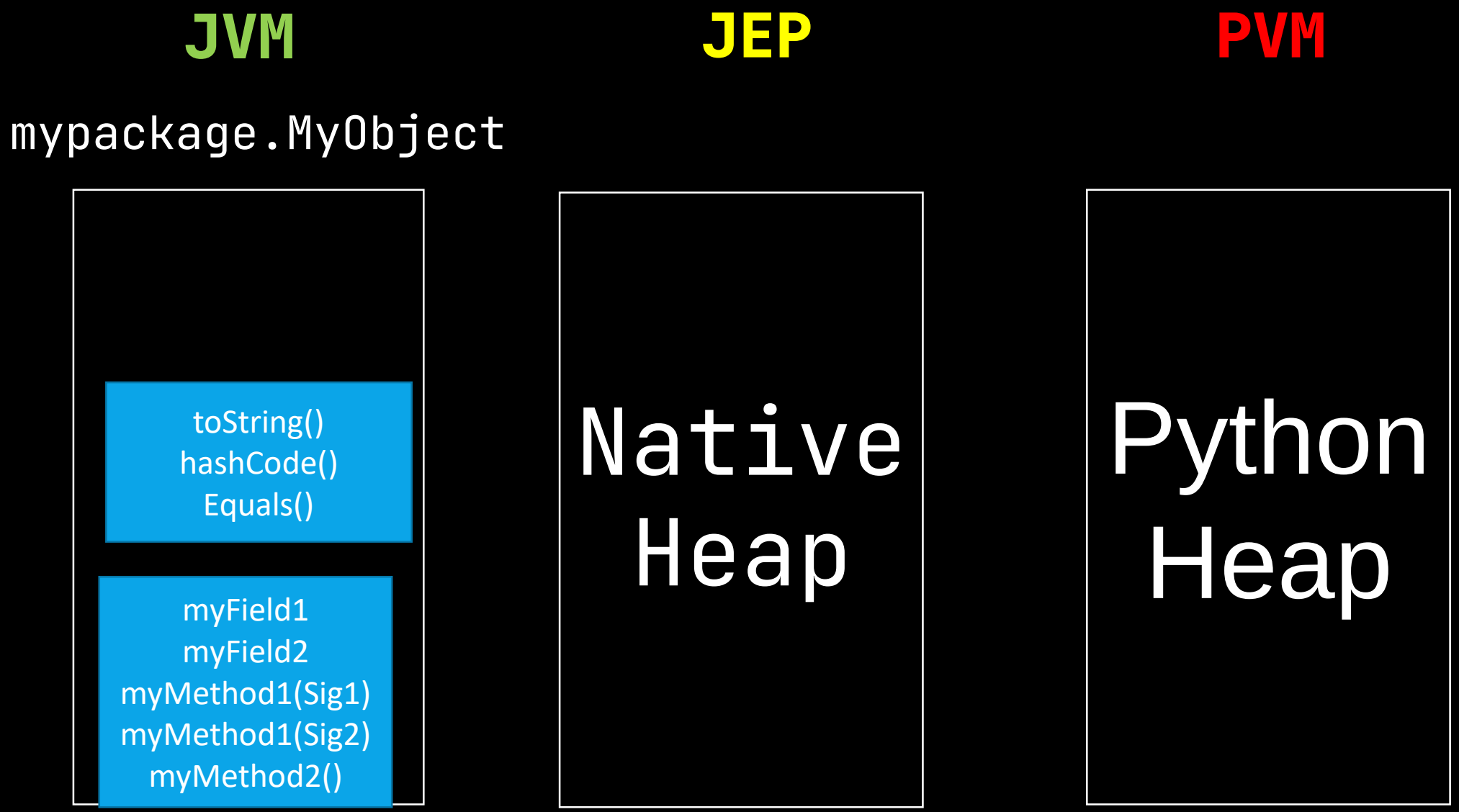
Представление Java Object в Python

JVM

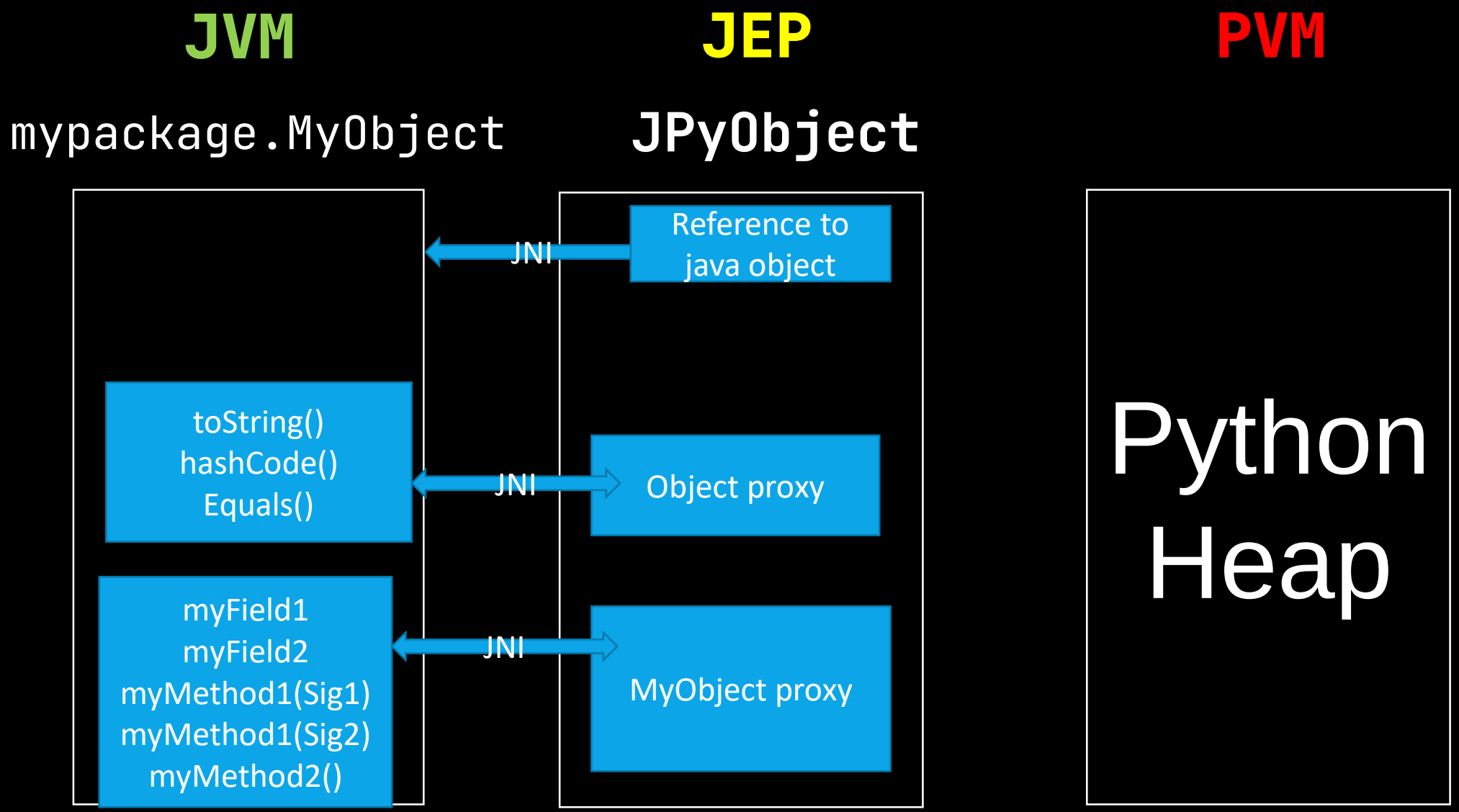
JEP

PVM

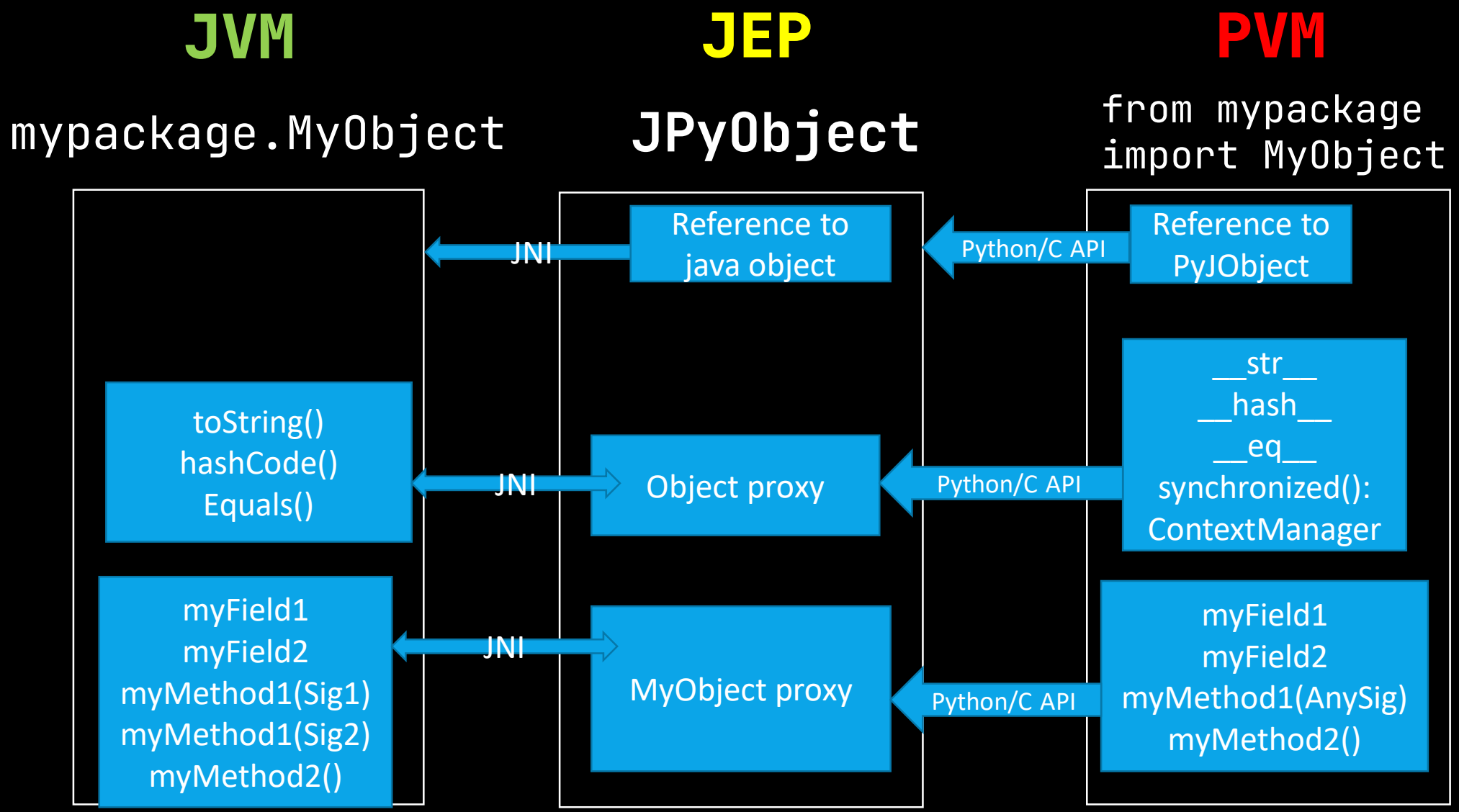
Представление Java Object в Python



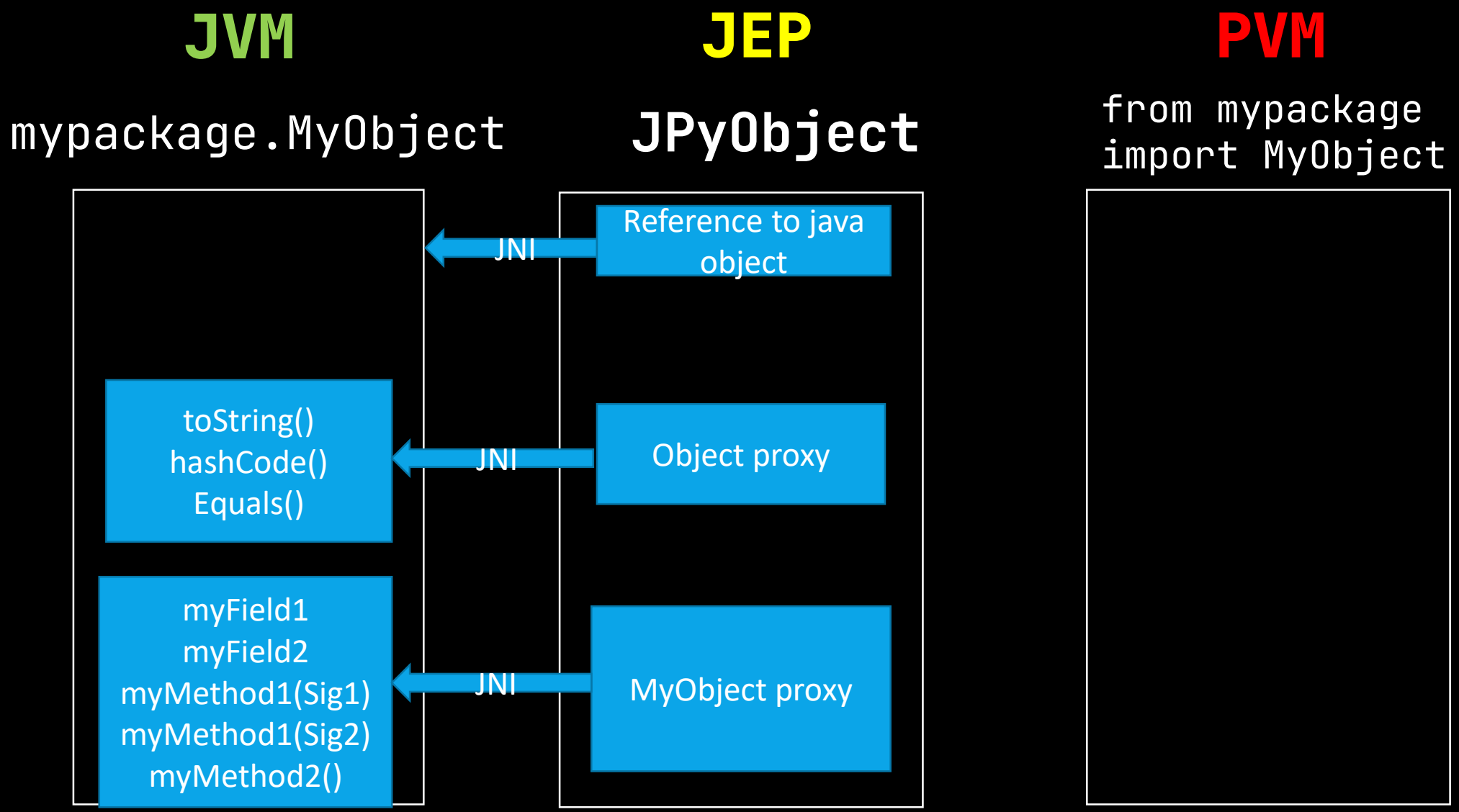
Представление Java Object в Python



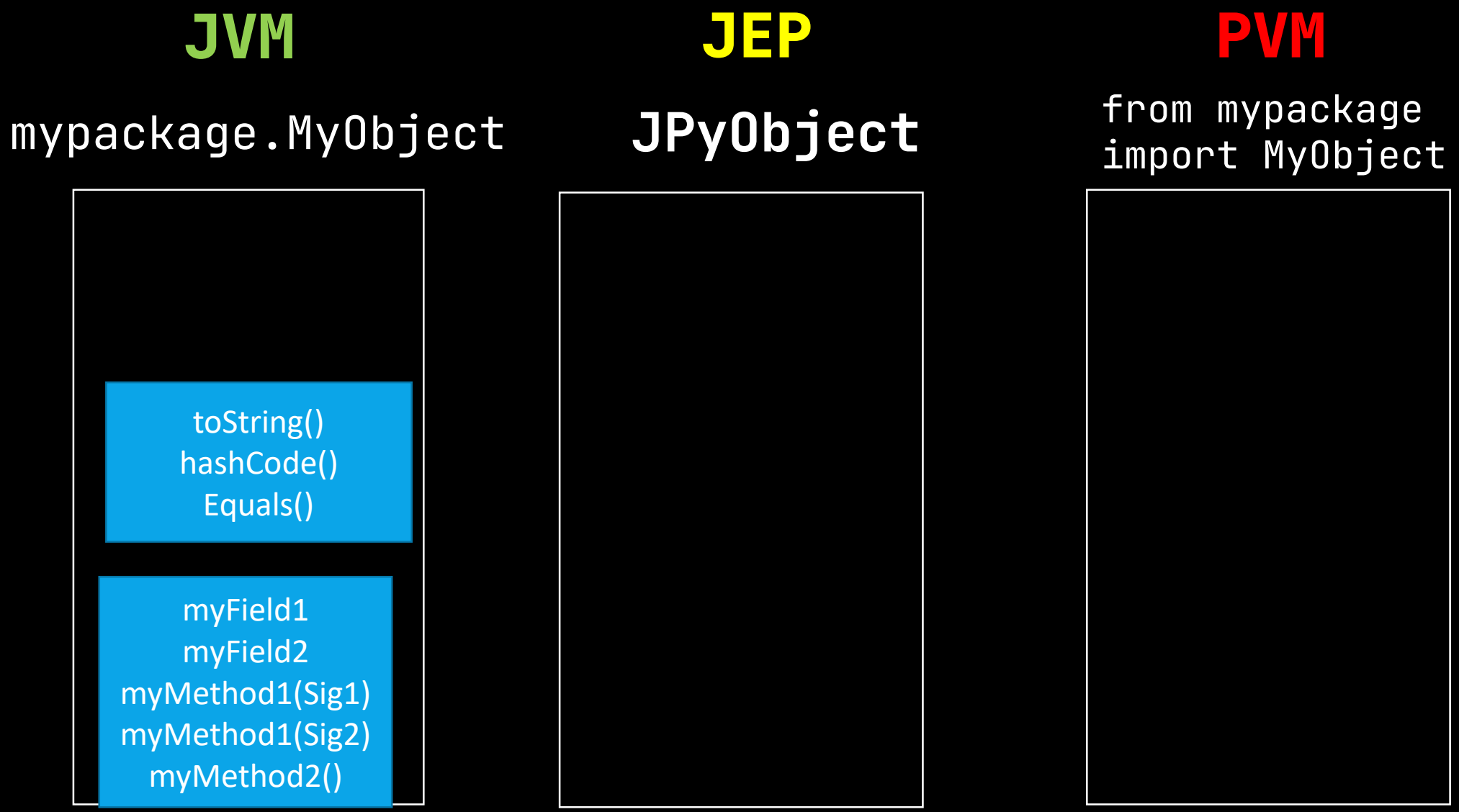
Представление Java Object в Python



Сборка мусора Java Object в Python

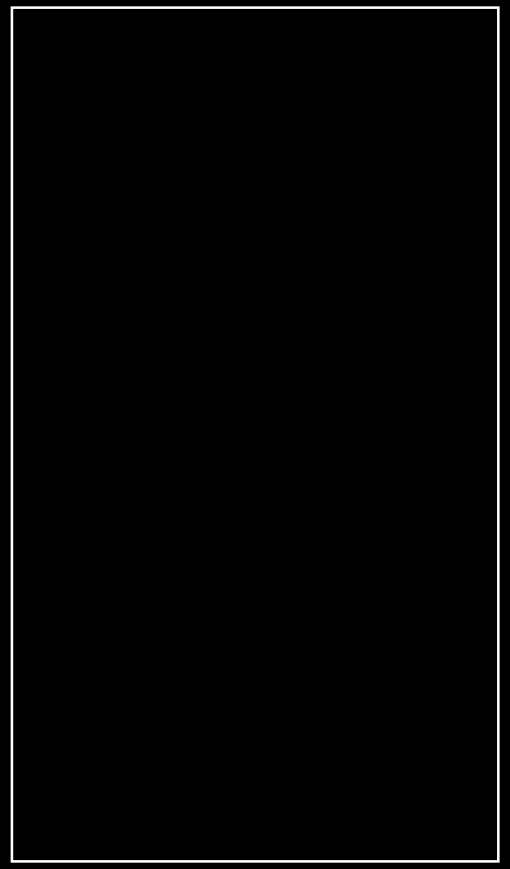


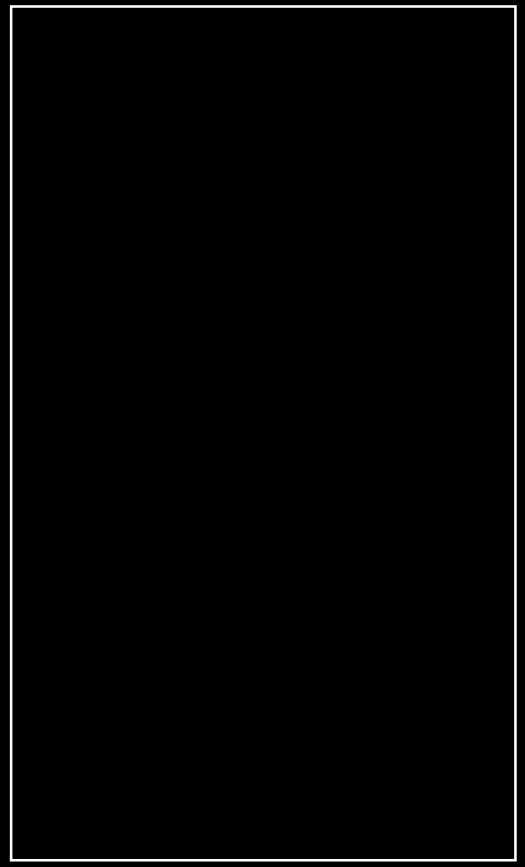
Сборка мусора Java Object в Python

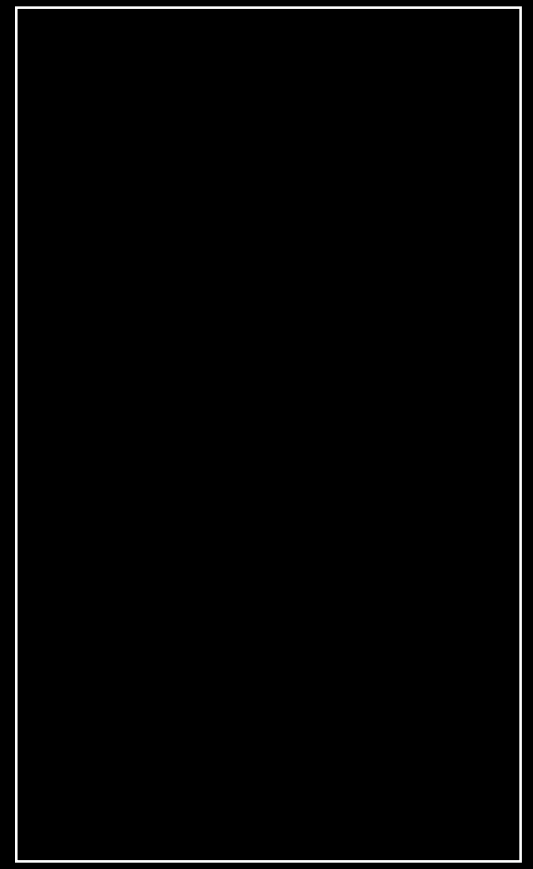


Сборка мусора Java Object в Python

JVM	JEP	PVM
`mypackage.MyObject`	`JPyObject`	`from mypackage import MyObject`







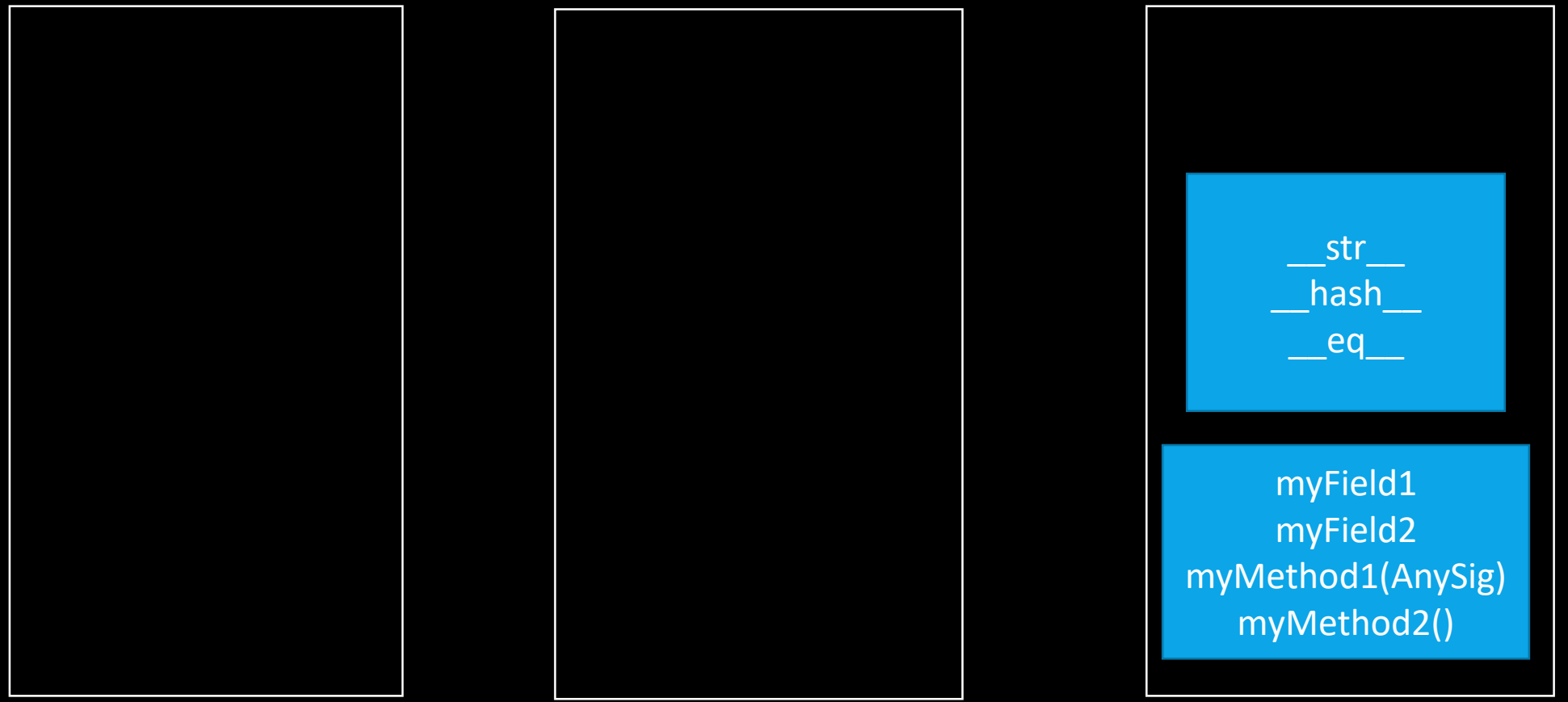
Представление Python Object в Java

JVM

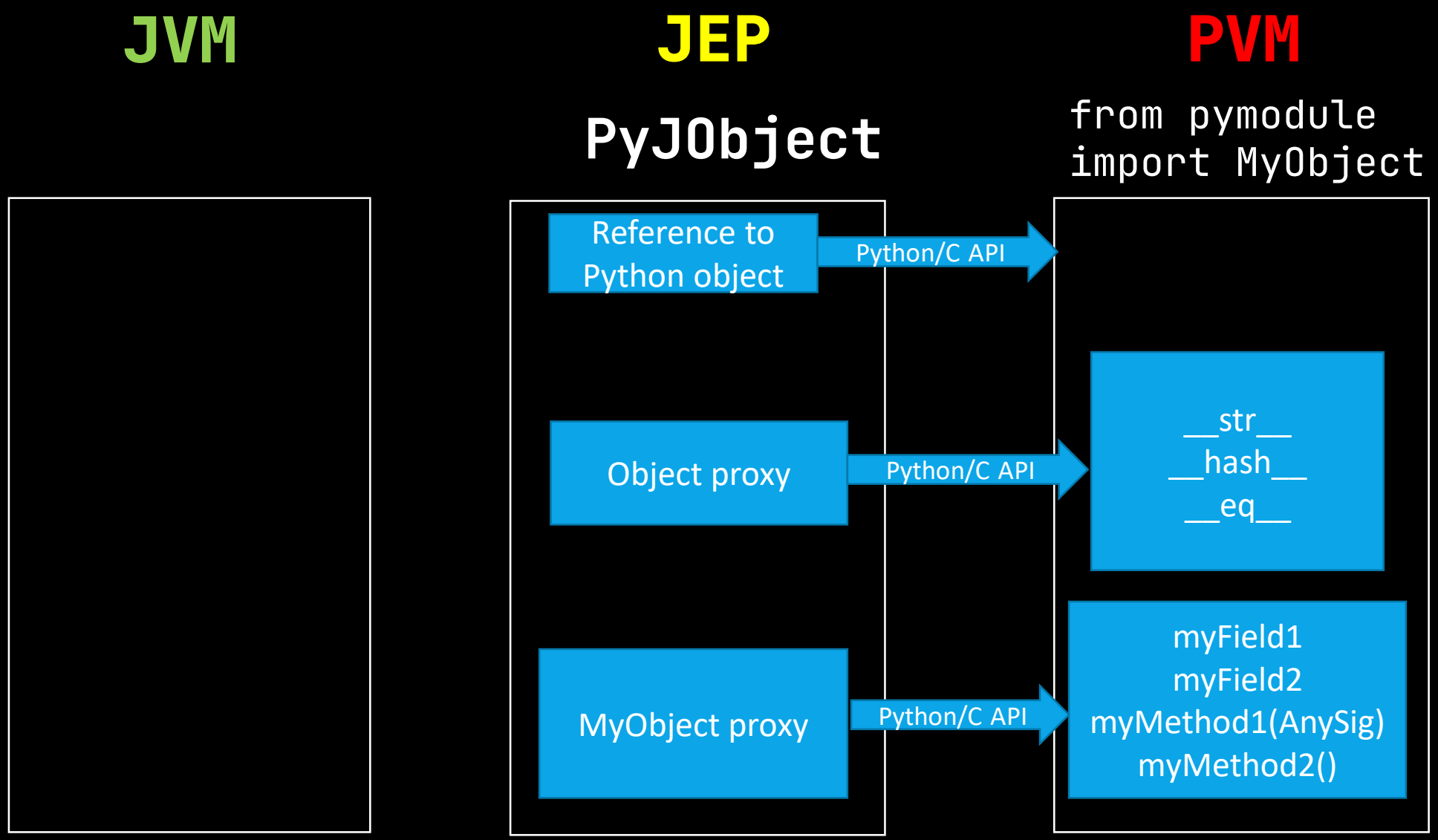
JEP

PVM

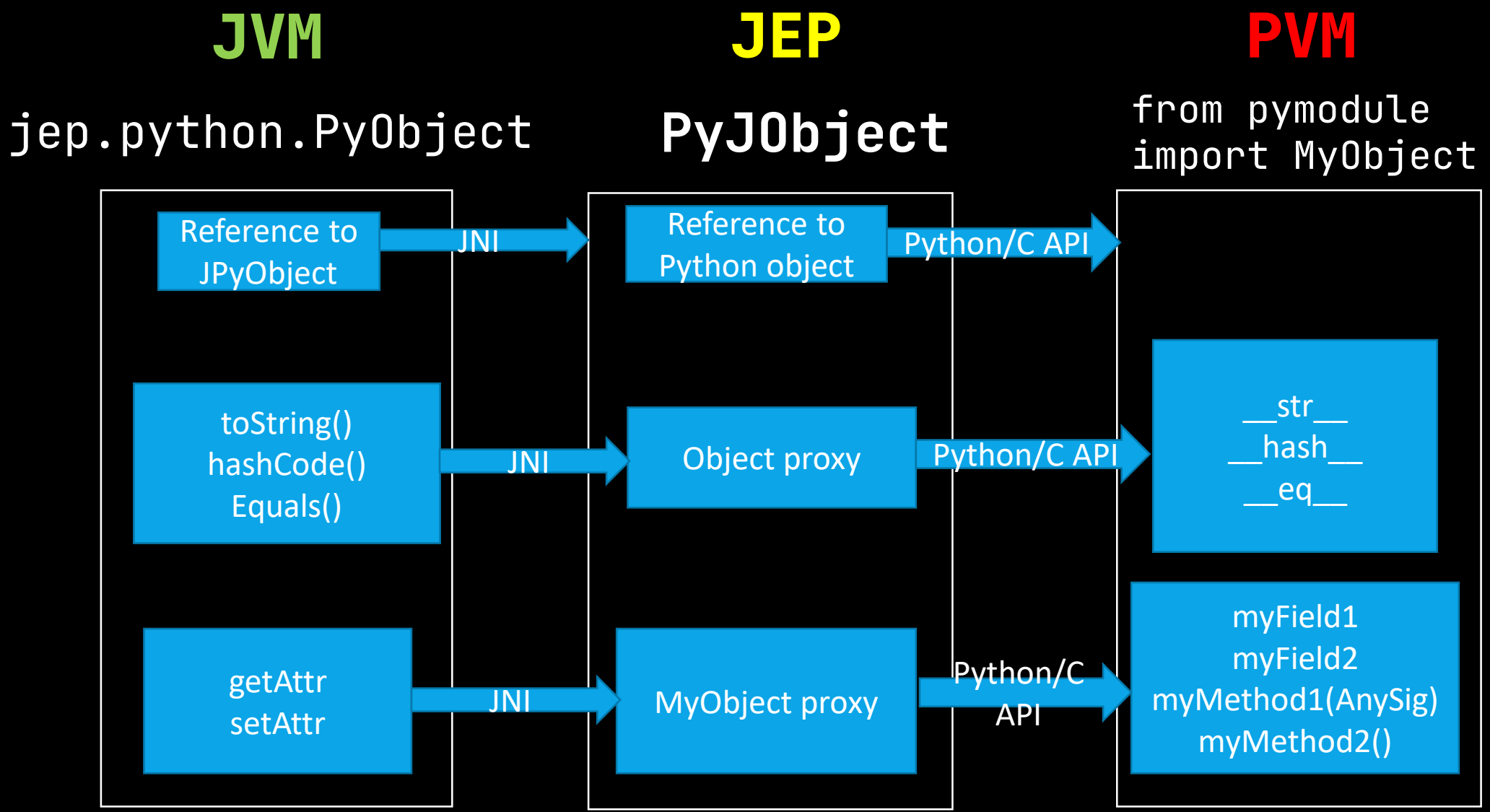
```
from pymodule
import MyObject
```



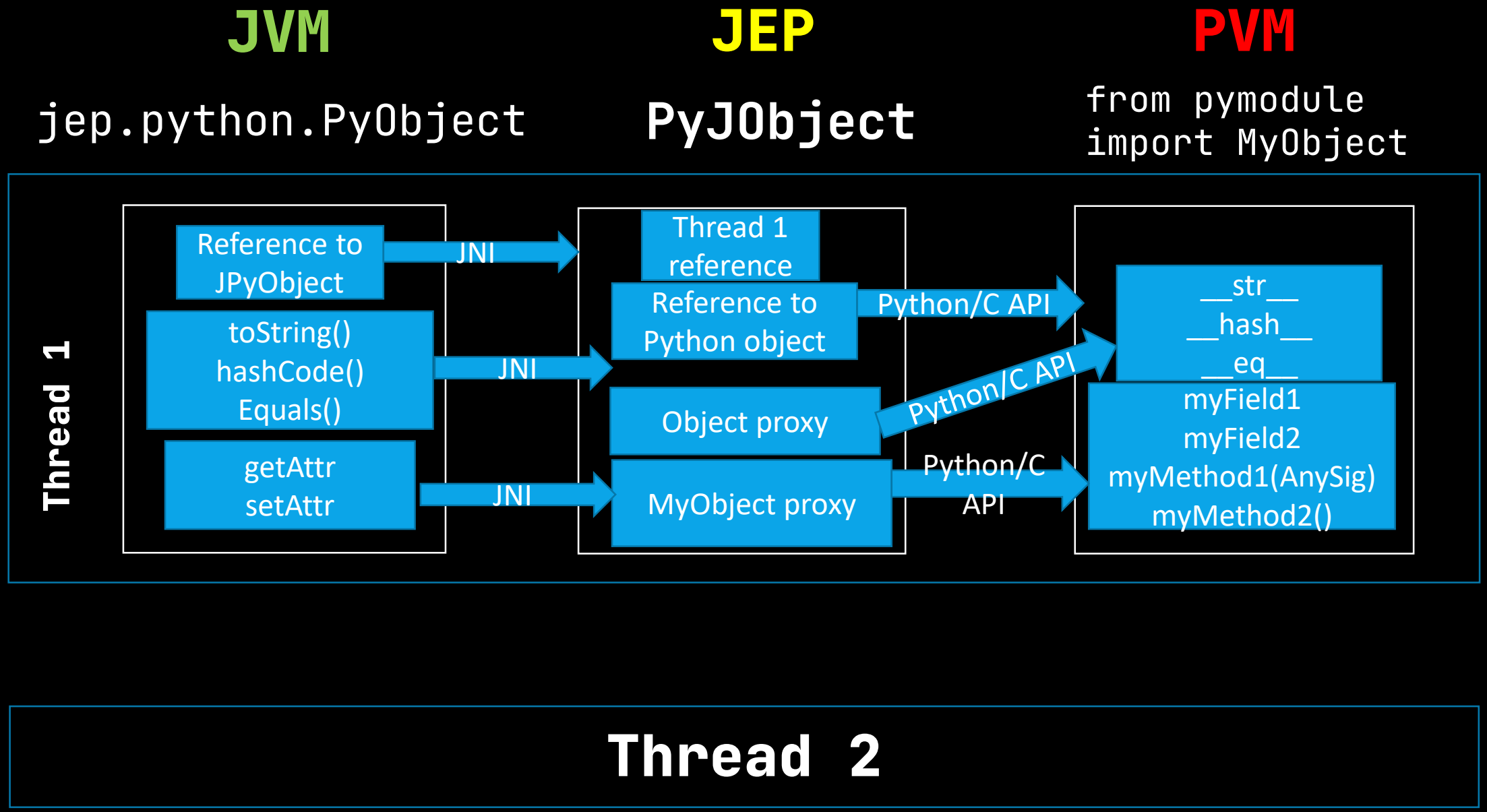
Представление Python Object в Java



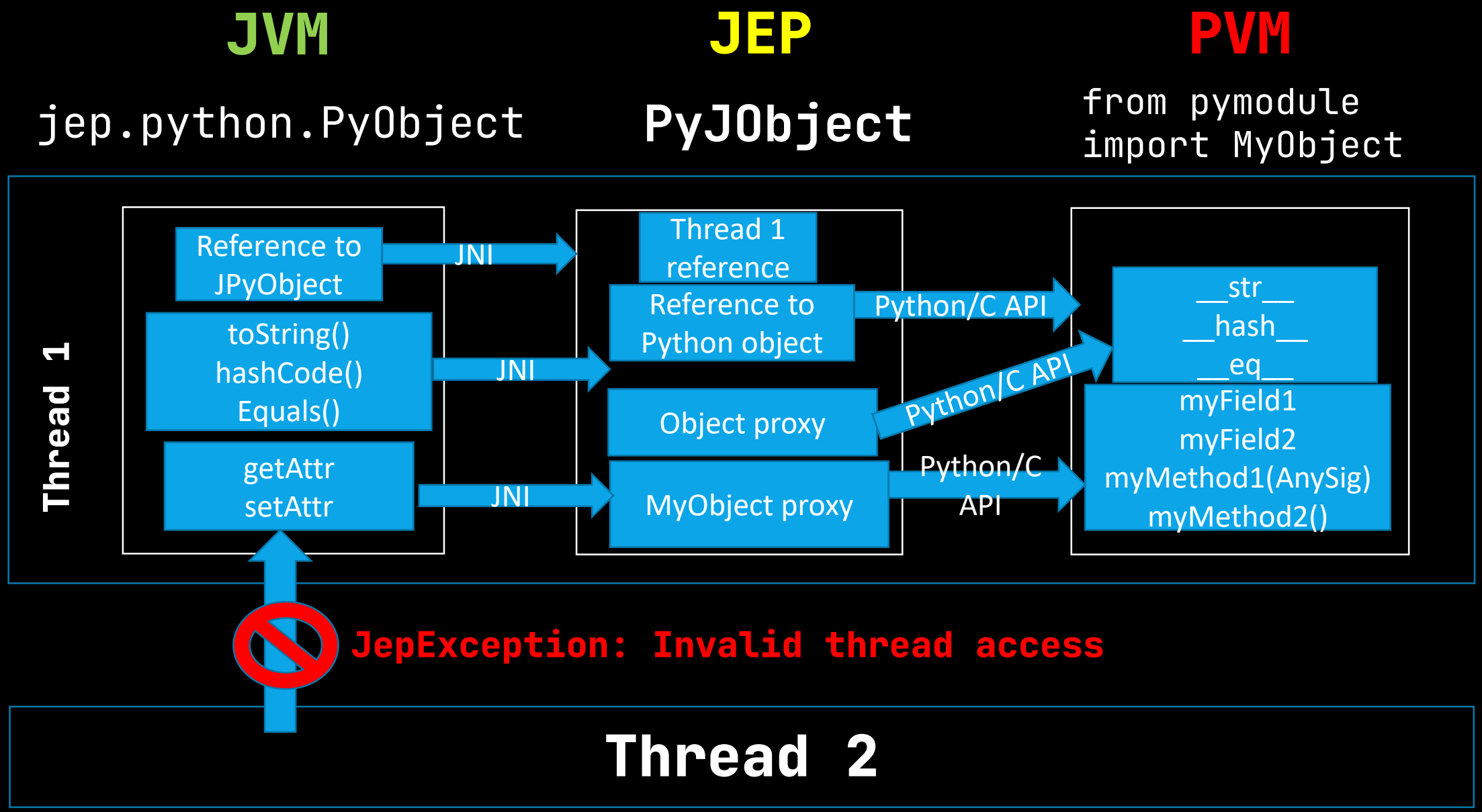
Представление Python Object в Java



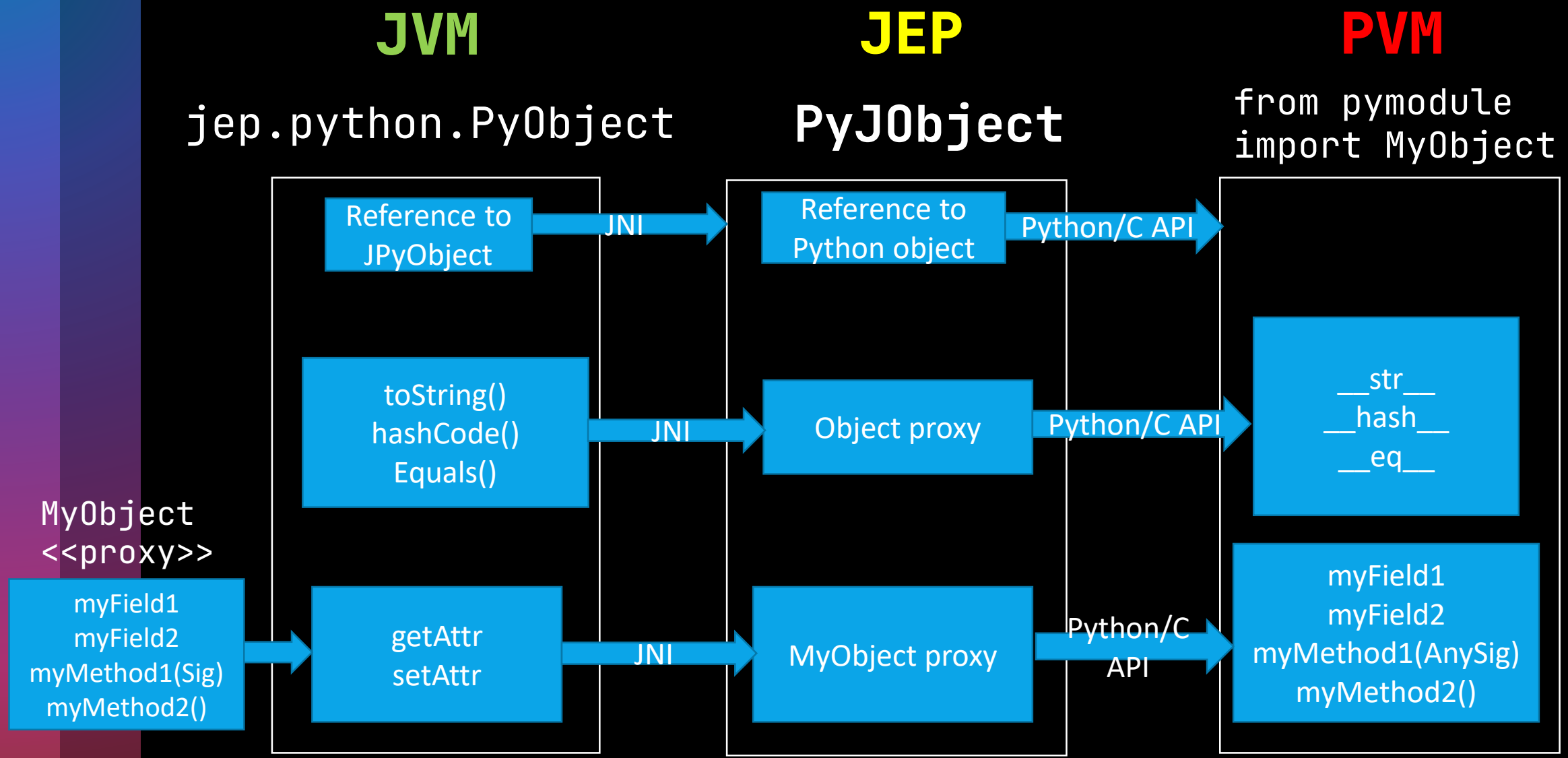
Многопоточность и PyObject



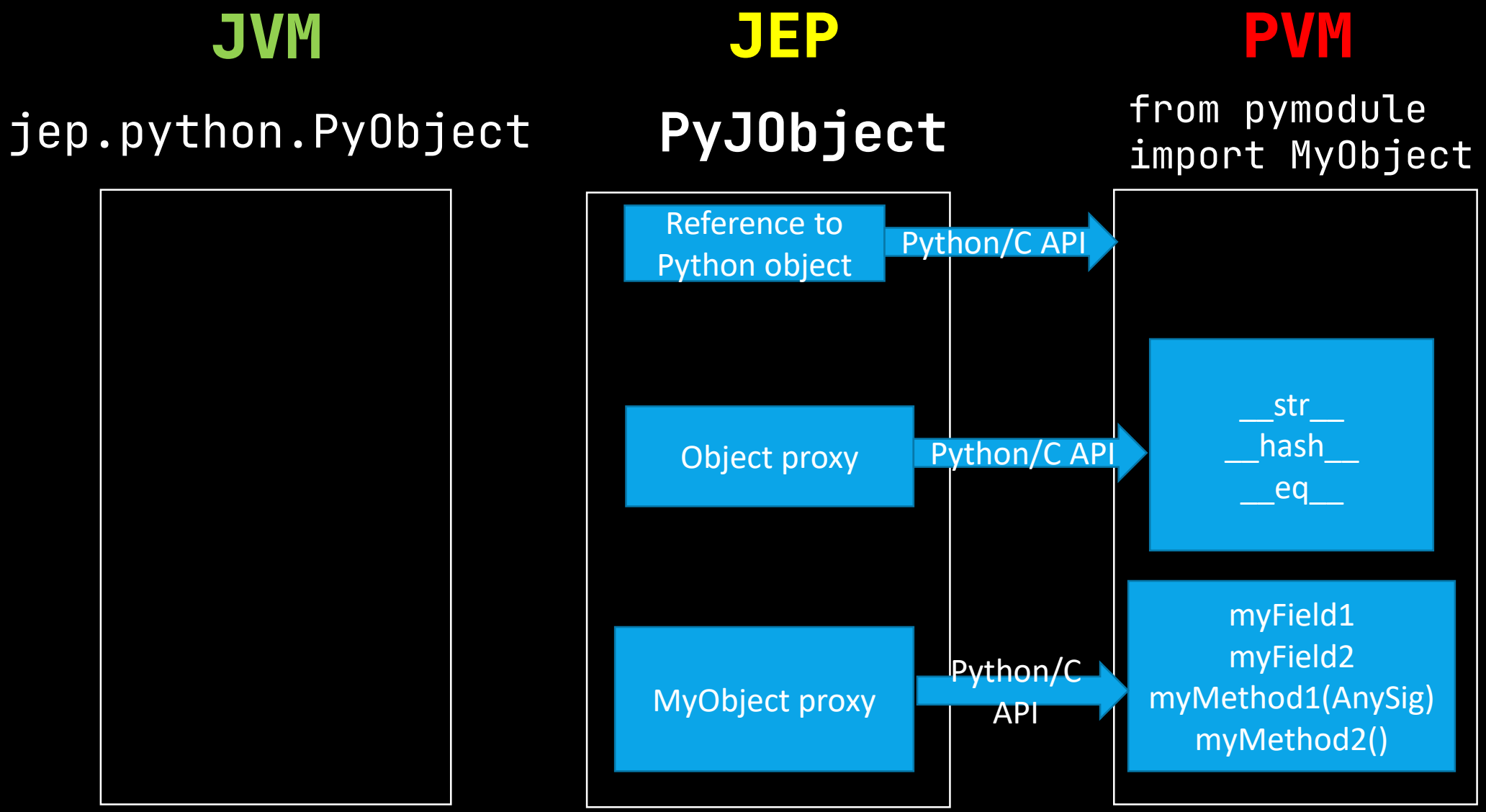
Многопоточность и PyObject



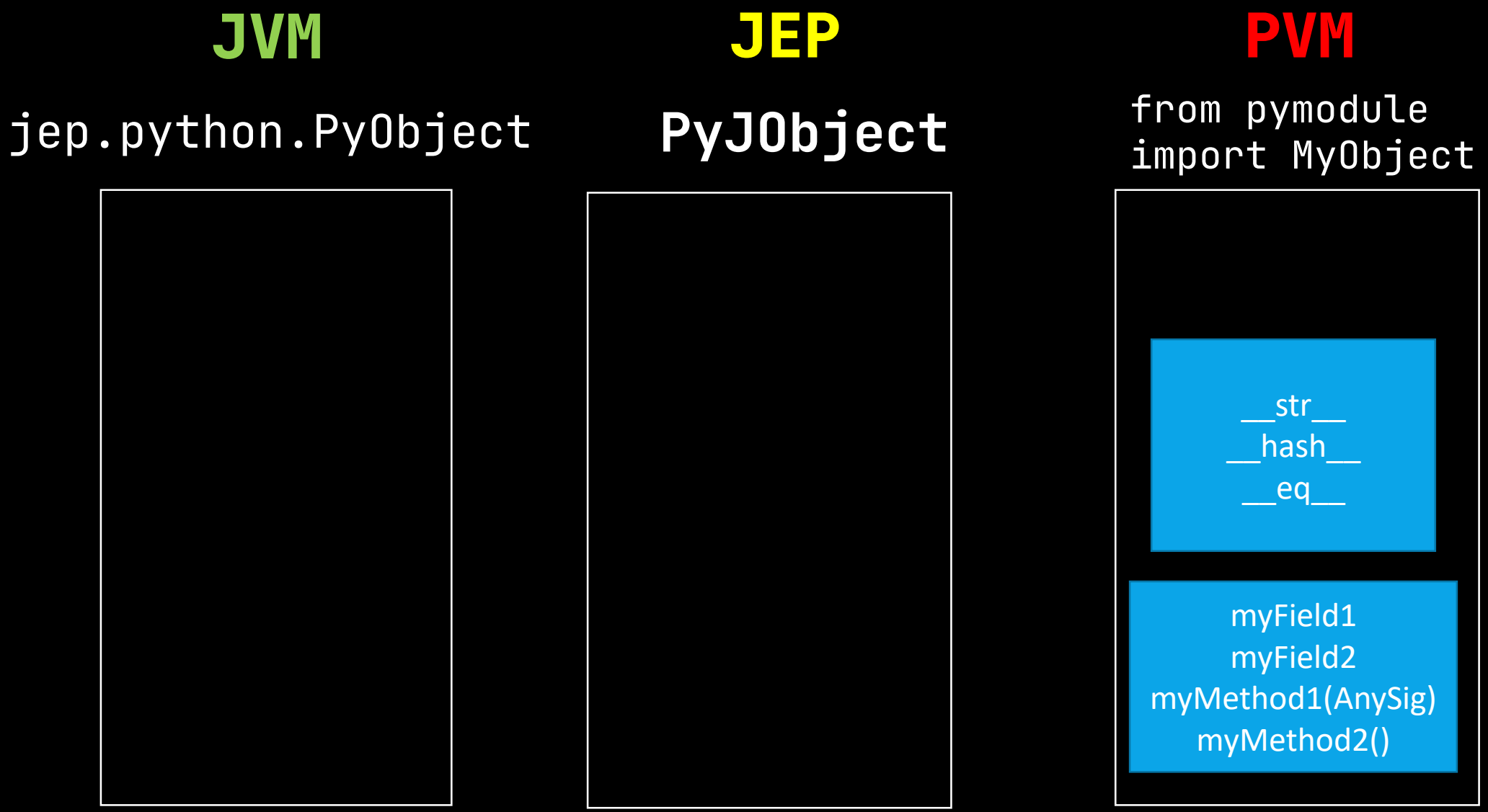
Представление Python Object в Java



Сборка мусора Python Object в Java



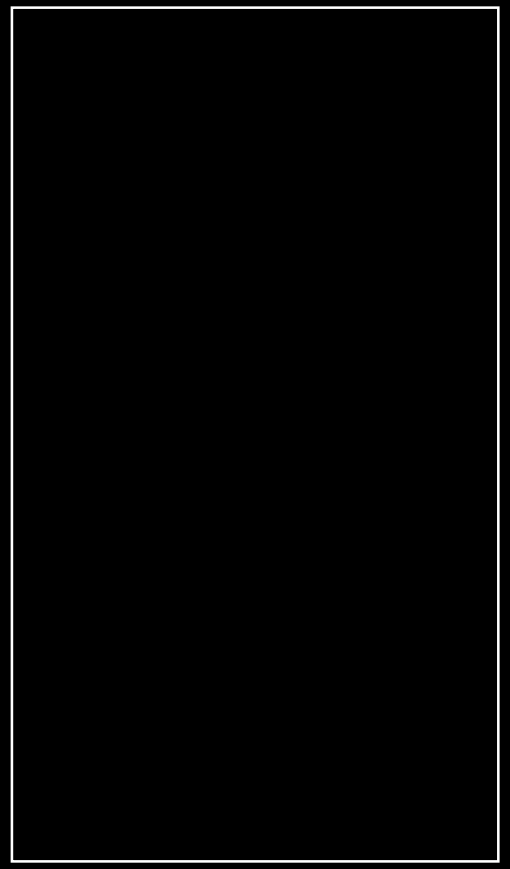
Сборка мусора Python Object в Java



Сборка мусора Python Object в Java

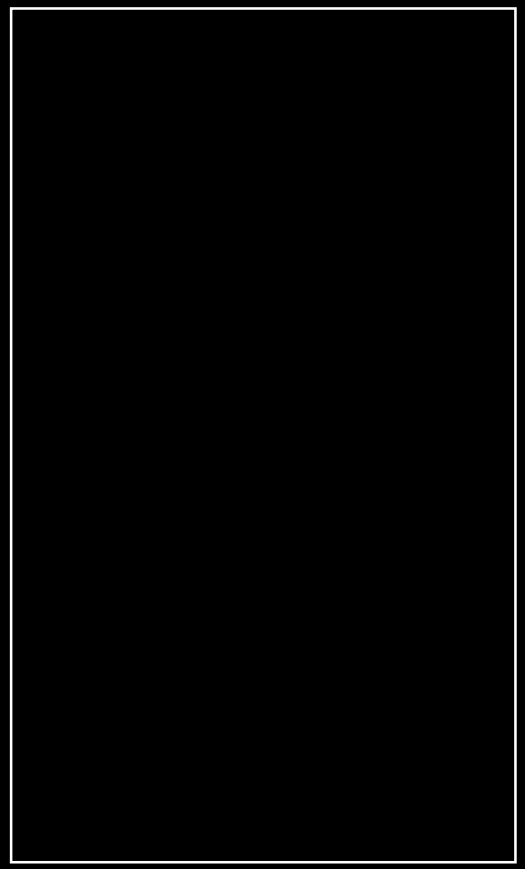
JVM

`jep.python.PyObject`



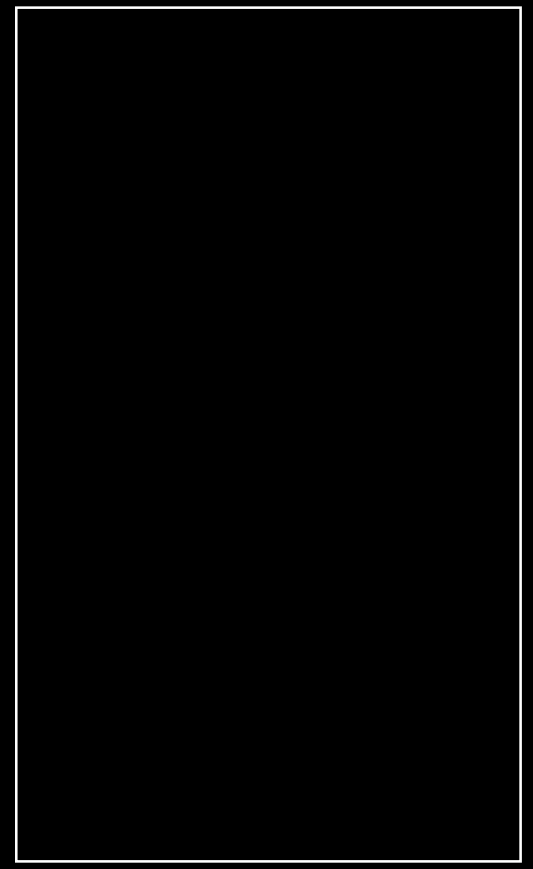
JEP

PyJObject



PVM

```
from pymodule
import MyObject
```



Встроенные преобразования типов

- Преобразования примитивных и боксинговых типов
- Работа java коллекций и number в питоне через встроенные операторы
- Преобразования ndarray

Python -> Java

Python type	Java type
list	ArrayList
dict	HashMap

Python <-> Java

Java	Python
None	null
double	float
long	int

Java -> Python

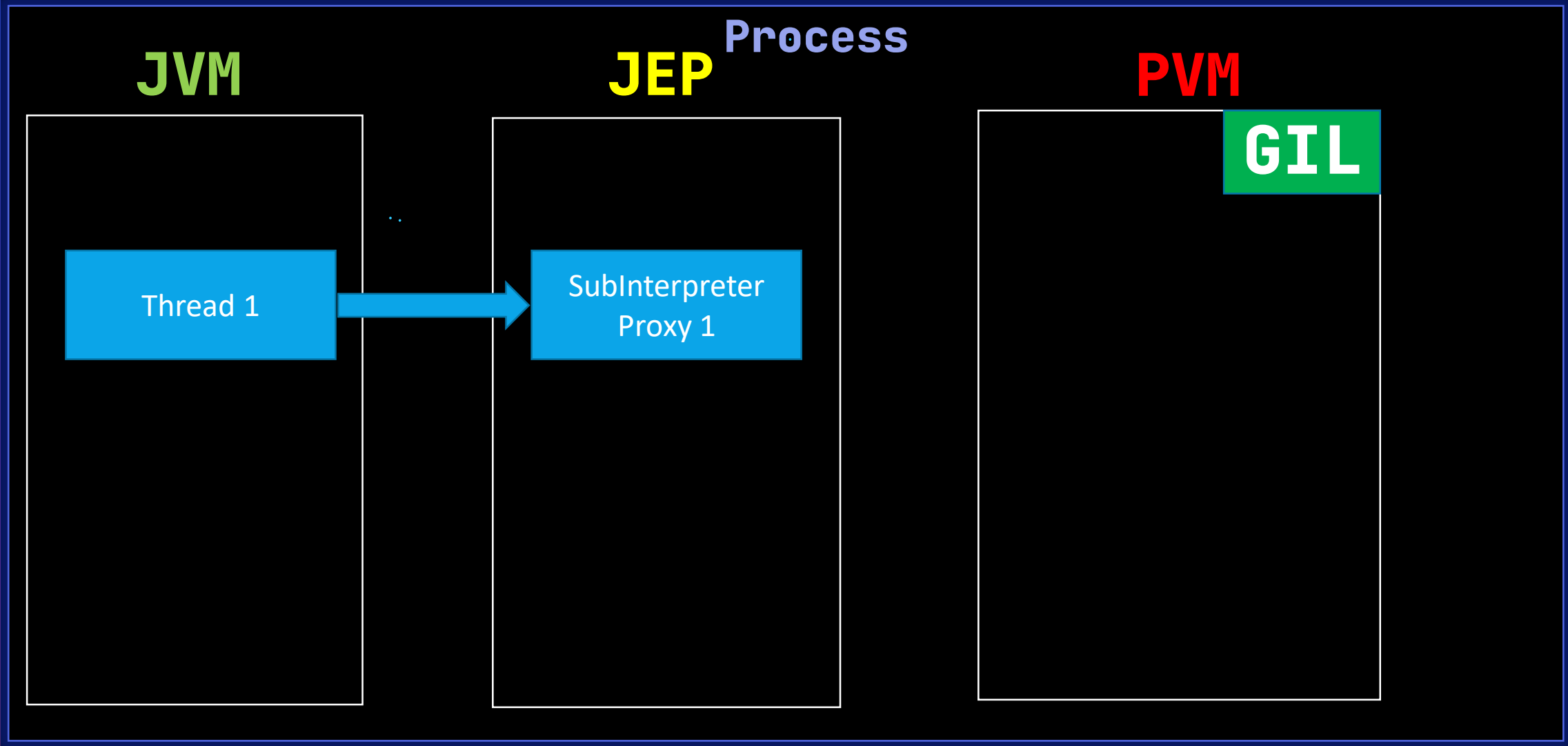
Java Type	Python Behavior
Primitive arrays/List	Ведет себя как python list
java.util.Map	Ведет себя как Python dict

Java Embedded Python (JEP)

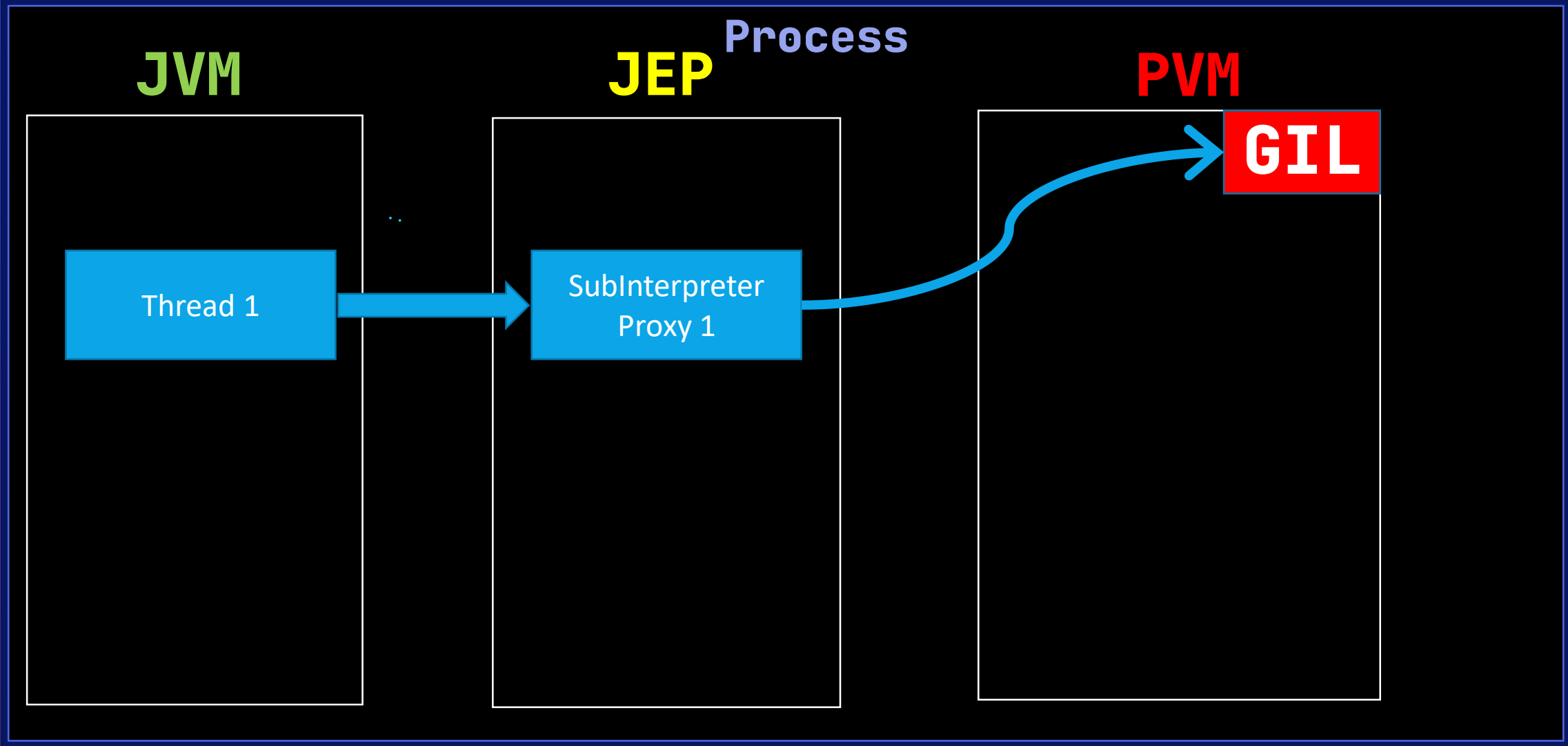
Примеры кода

- Примеры кода и возможности JEP на примерах
  https://github.com/icemachined/jep-distro-example

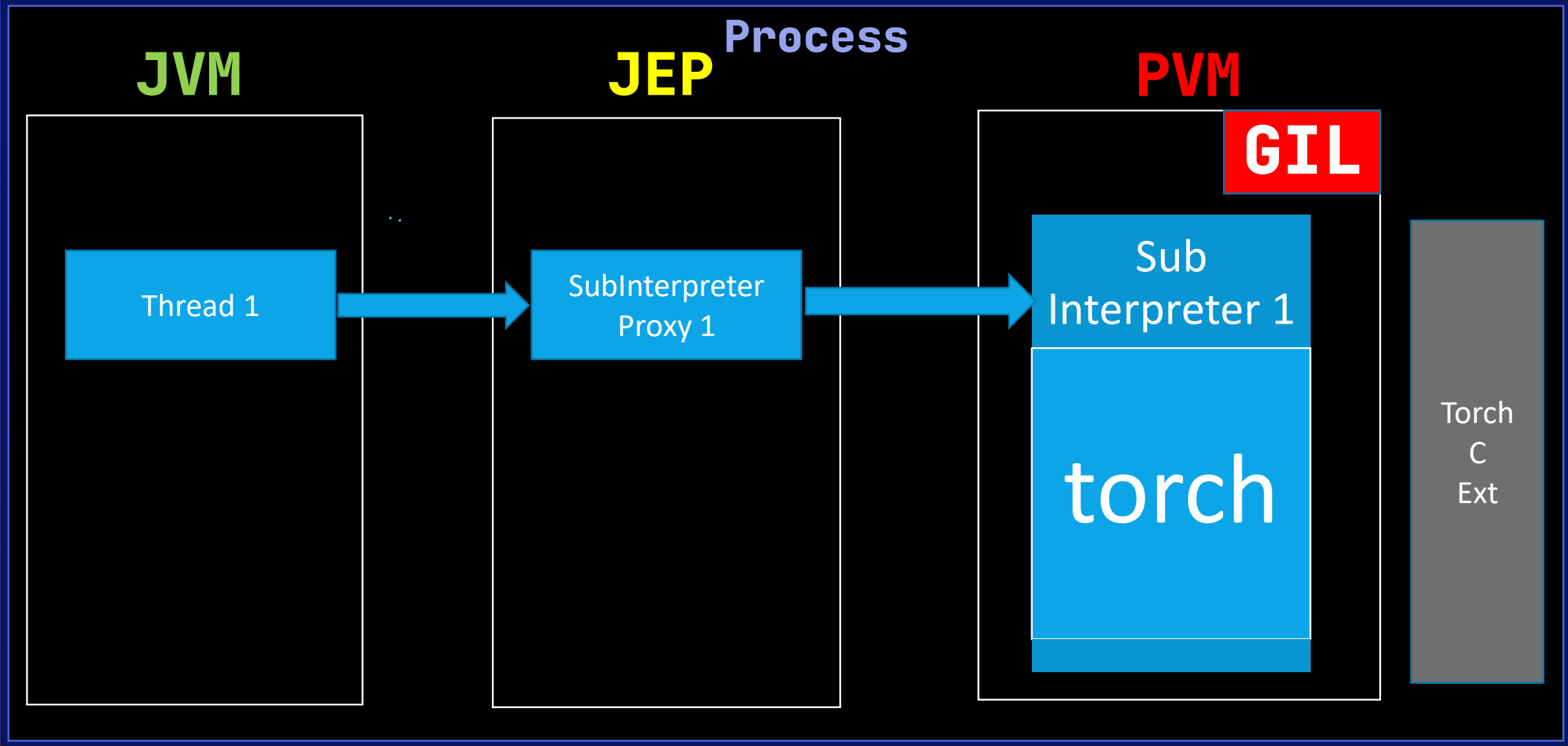
Проблема работы с python с extentions



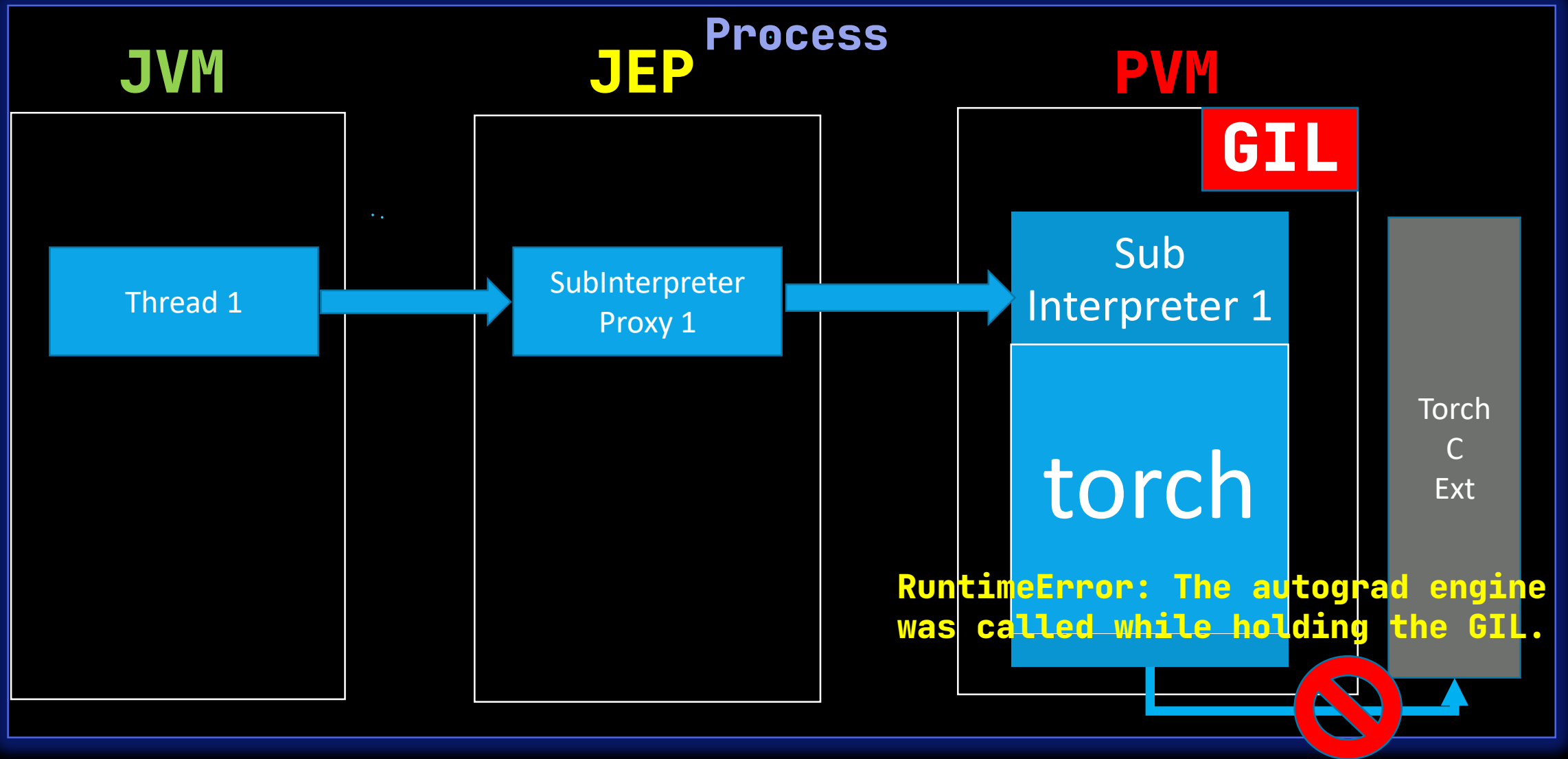
Проблема работы с python с extentions



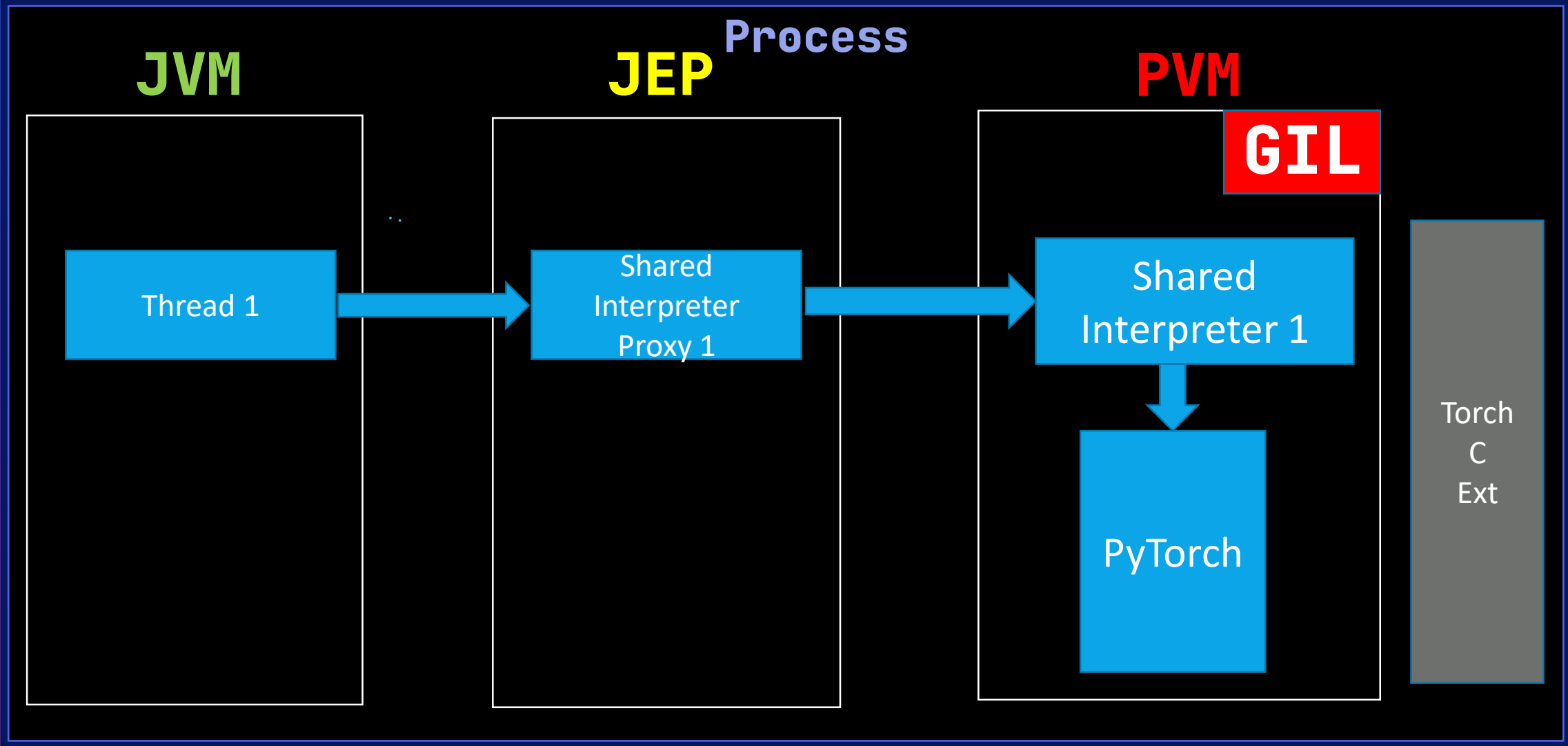
Проблема работы с python с extentions



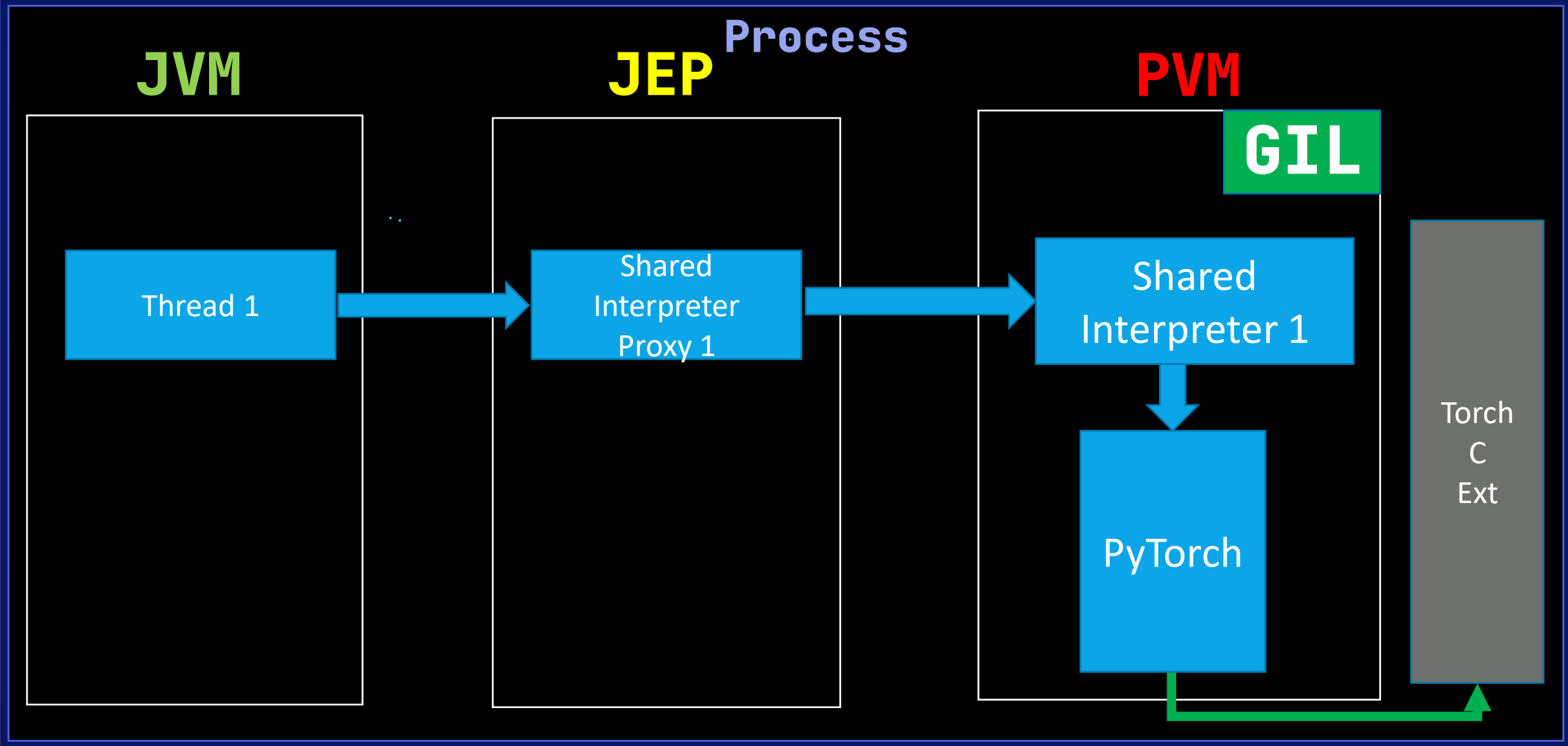
Проблема работы с python с extentions



Проблема работы с python с extentions



Проблема работы с python с extentions



Что мы узнали

- Мы разобрались какие есть способы поженить Python на Java
- Для каких задач это может быть полезно
- Что такое JEP и как он работает
- Как с помощью JEP запустить GPT внутри Spring Boot.


Java
Python

Полезные ссылки

- https://github.com/ninia/jep/
- https://github.com/icemachined/jep-distro
- https://github.com/icemachined/jep-distro-example
- https://github.com/icemachined/gpt-service-example
- https://github.com/karpathy/nanoGPT
- https://www.jython.org/
- https://www.graalvm.org/python/
- https://github.com/alibaba/pemja
- Pemja FLIP-206
- https://github.com/HuangXingBo/pyflink-benchmark