Coordinated Restore at Checkpoint

Быстрый старт OpenJDK

Антон Козлов

AZUL

Проблемы

Долгая инициализация

Проблемы

Долгая инициализация

- загрузка классов

Проблемы

Долгая инициализация

- загрузка классов
- инициализация классов

Проблемы

Долгая инициализация

- загрузка классов
- инициализация классов
- JIT-компиляция

Проблемы

Долгая инициализация

- загрузка классов
- инициализация классов
- JIT-компиляция

Обработка запросов - требует разогрева

- ... то же самое ...

Проблемы

Долгая инициализация
- загрузка классов
- инициализация классов
- JIT-компиляция

Обработка запросов - требует разогрева
- ... то же самое ...

Окружение (контейнеры, embedded):
- ОС Linux
- небыстрый процессор, мало ядер
- медленный диск

Проблемы

Долгая инициализация
- загрузка классов
- инициализация классов
- JIT-компиляция

Обработка запросов - требует разогрева
- ... то же самое ...

Окружение (контейнеры, embedded):
- ОС Linux
- небыстрый процессор, мало ядер
- медленный диск
- конкуренция за ресуры

Технологии

JIT

(App) Class Data Sharing

Ahead-of-Time compilation

GraalVM Native image

Технологии

JIT

(App) Class Data Sharing

Ahead-of-Time compilation

Coordinated Restore at Checkpoint (CRaC)

GraalVM Native image

CRIU

CRIU: Checkpoint/Restore In Userspace

- контейнеры: миграция, репликация,..

CRIU

CRIU: Checkpoint/Restore In Userspace

- контейнеры: миграция, репликация,..

- Checkpoint: сохранить состояние процесса

CRIU

CRIU: Checkpoint/Restore In Userspace

- контейнеры: миграция, репликация,..

- Checkpoint: сохранить состояние процесса

CRIU

CRIU: Checkpoint/Restore In Userspace

- контейнеры: миграция, репликация,..

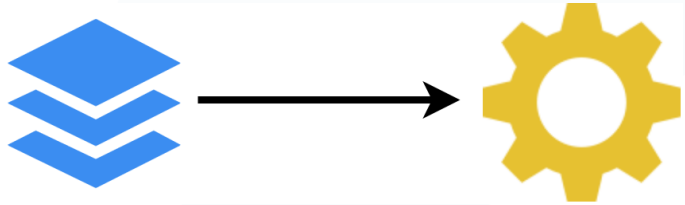
- Checkpoint: сохранить состояние процесса

CRIU

CRIU: Checkpoint/Restore In Userspace

- контейнеры: миграция, репликация,..
- Checkpoint: сохранить состояние процесса

- Restore: создать процесс из сохраненного состояния



CRIU + Java

Если мы подготовимся

- проинициализируем Java приложение
- разогреем JVM
- сделаем checkpoint

Тогда restore == старт

- с загруженными и проинициализированными классами
- с готовыми JIT-компиляциями
  - мы на пиковой производительности

Пример

```
public class Test {
    public static void main(String[] args) throws Exception {
        for (int i = 1; i <= 3; ++i) {
            System.out.println("Pre-compute something big " + i + "...");
            Thread.sleep(1_000);
        }

        System.out.println("Waiting for input");
        int c = System.in.read();
        System.out.println("processing input: " + c);
    }
}
```

Christine Flood. Checkpointing Java from outside of Java

Пример

```
public class Test {
    public static void main(String[] args) throws Exception {
        for (int i = 1; i <= 3; ++i) {
            System.out.println("Pre-compute something big " + i + "...");
            Thread.sleep(1_000);
        }
        jdk.crac.Core.checkpointRestore();
        System.out.println("Waiting for input");
        int c = System.in.read();
        System.out.println("processing input: " + c);
    }
}
```

github.com/CRaC/docs

Пример Jetty

```
import org.eclipse.jetty.server.Server;

class ServerManager {
    Server server;

    public ServerManager(int port, Handler handler) throws Exception {
        server = new Server(8080);
        server.setHandler(handler);
        server.start();
        jdk.crac.Core.checkpointRestore();
    }
}
```

Java + CRIU

Внутреннее состояние

- прошлая инициализация в неактуальных условиях

Java + CRIU

Внутреннее состояние

- прошлая инициализация в неактуальных условиях

Внешнее состояние

- конфликты при запуске нескольких экземпляров
- повторная регистрация

Coordinated Restore at Checkpoint

Приложение и Java вместе работают для Checkpoint/Restore

- приложение управляет состоянием и ресурсами
  - использует расширяемый механизм для оповещения
  - может отменить Checkpoint

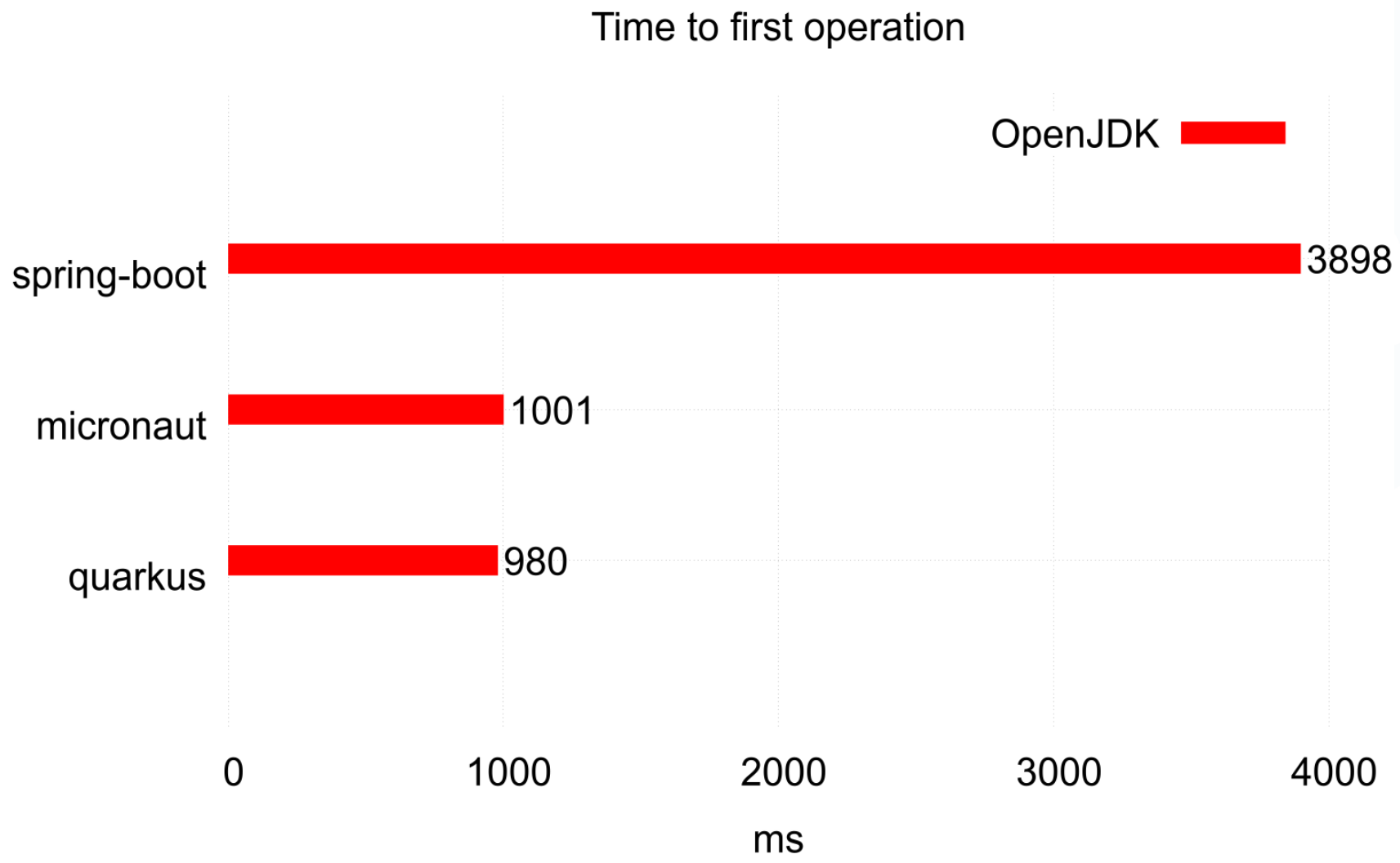
Coordinated Restore at Checkpoint

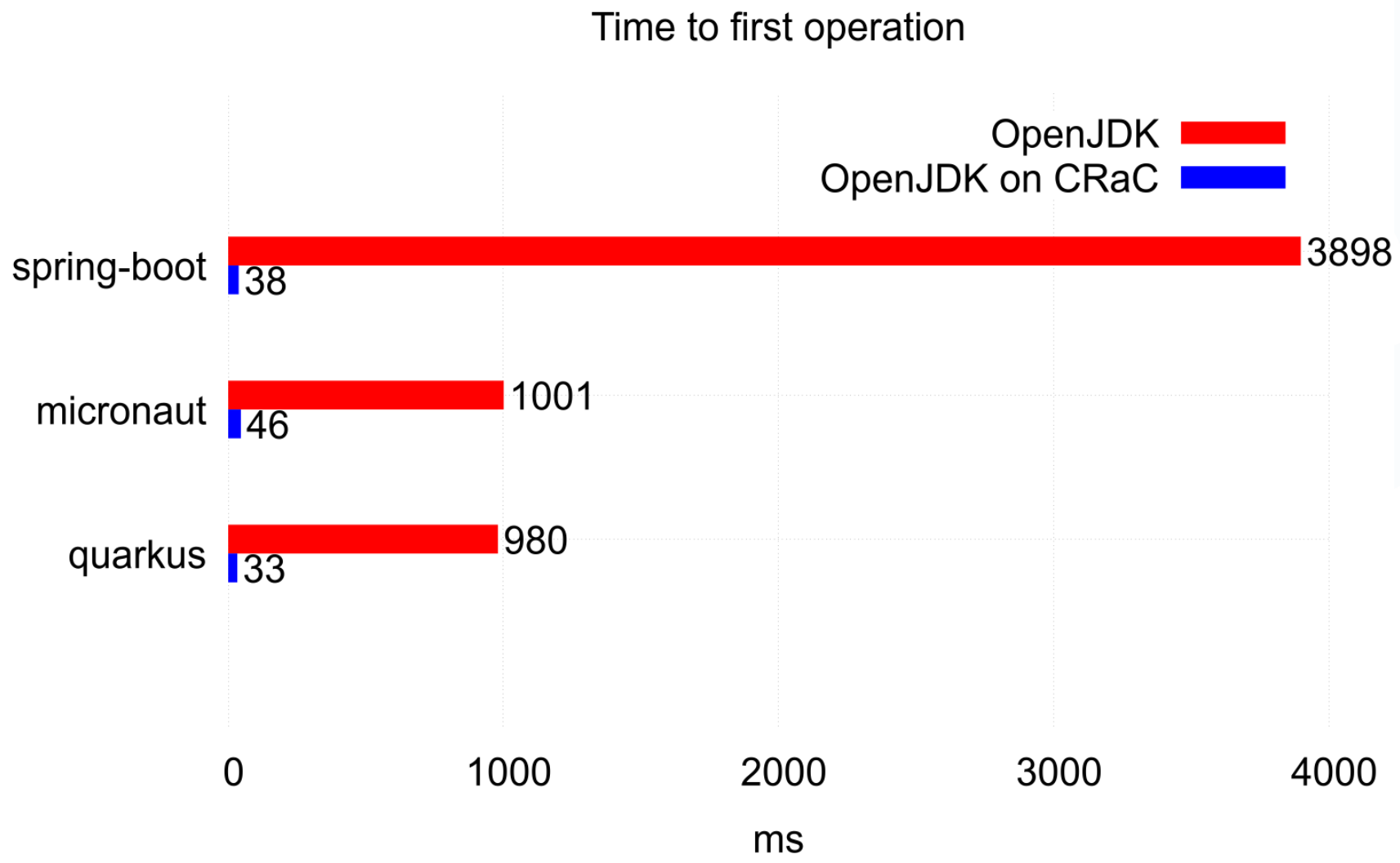
Приложение и Java вместе работают для Checkpoint/Restore

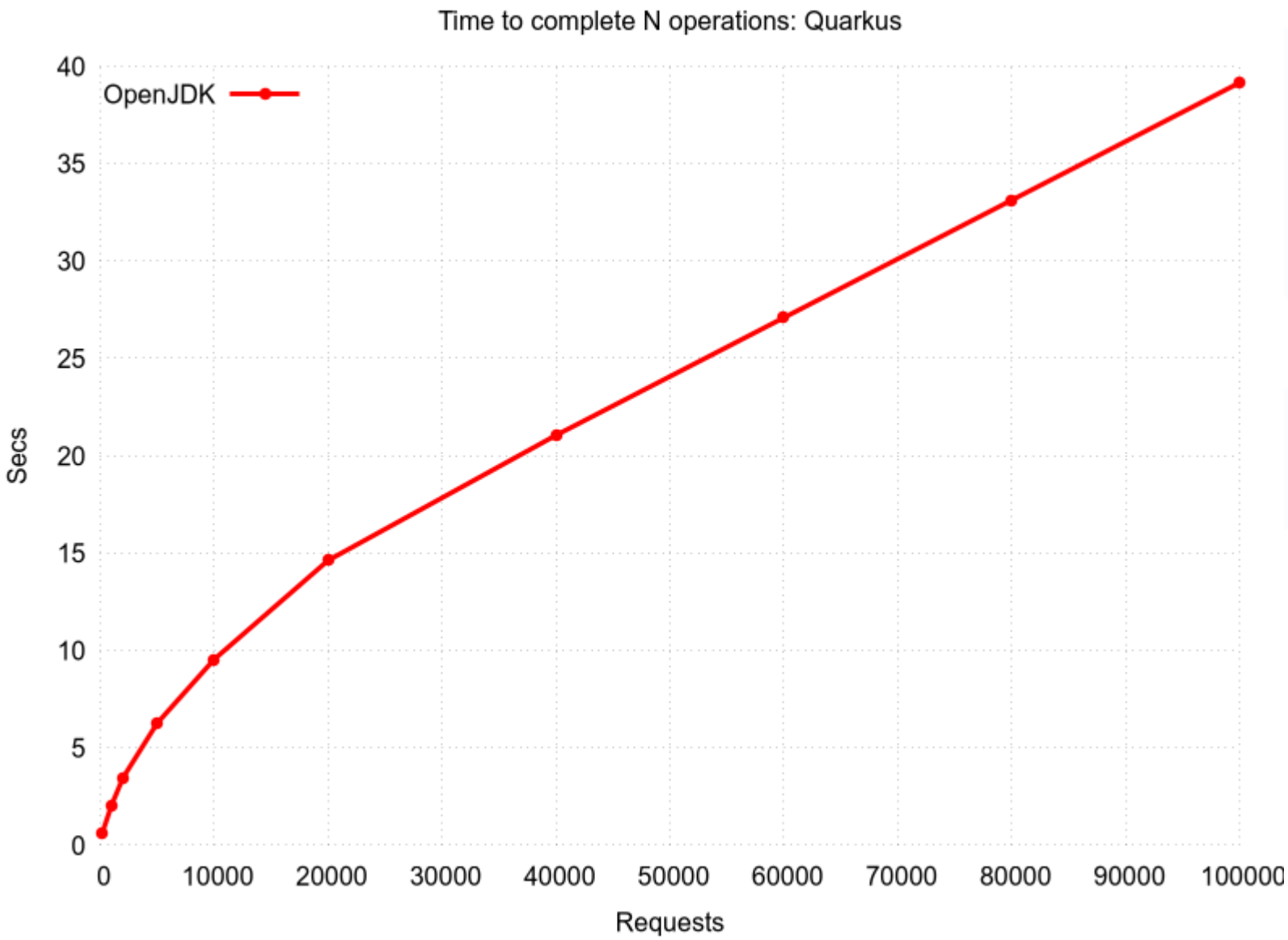
- приложение управляет состоянием и ресурсами
  - использует расширяемый механизм для оповещения
  - может отменить Checkpoint
- CRaC помогает находить опасные состояния
  - обнаруживает ресурсы ОС
    - сокеты
    - открытые файлы
    - межпроцессные взаимодействия
  - может отменить Checkpoint

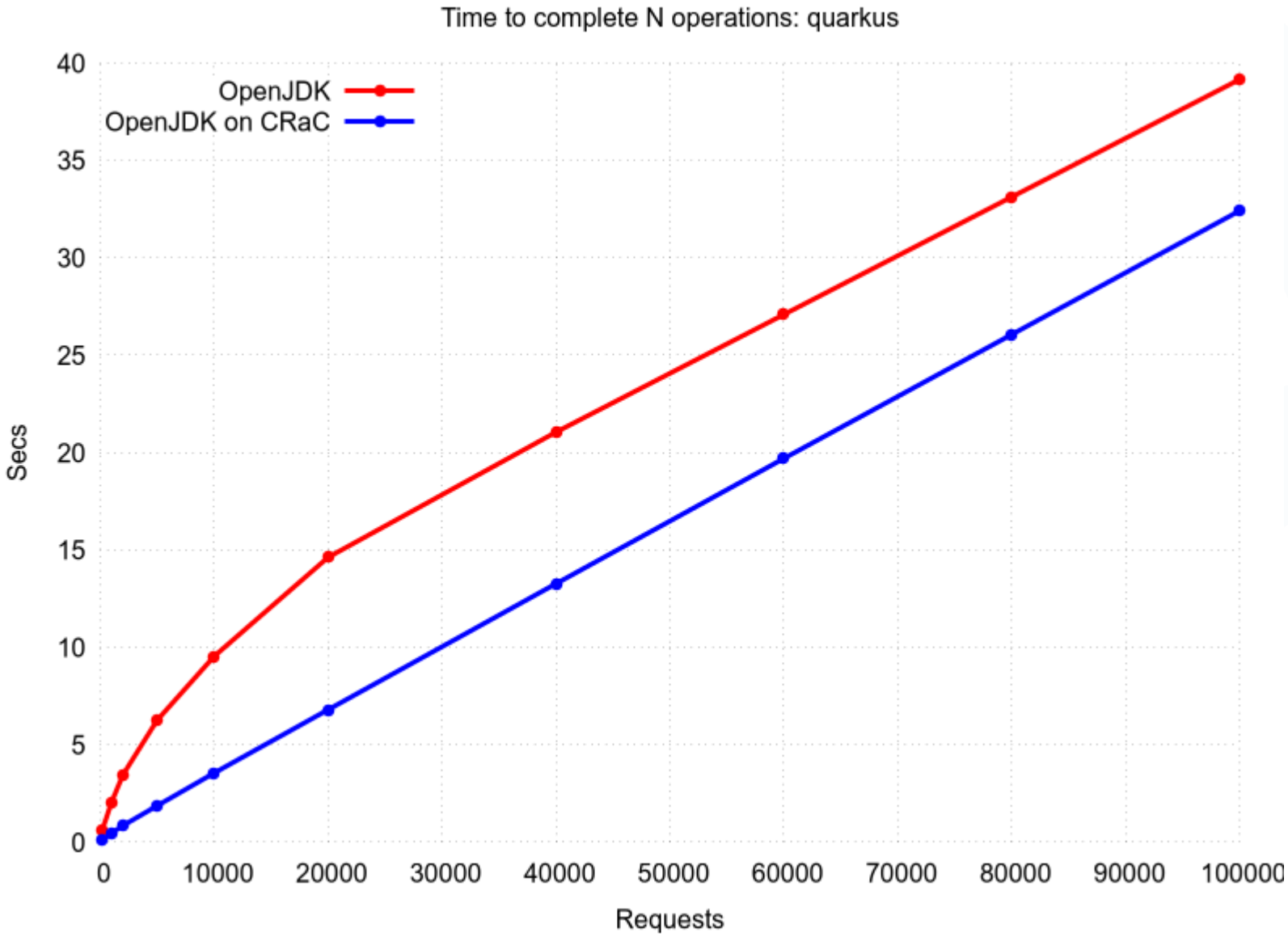
Пример Jetty с CRaC API

```
import jdk.crac.Context;
import jdk.crac.Core;
import jdk.crac.Resource;

import org.eclipse.jetty.server.Server;

class ServerManager implements Resource {
    Server server;

    @Override
    public void beforeCheckpoint(Context<? extends Resource> context) throws Exception {
        server.stop();
    }

    @Override
    public void afterRestore(Context<? extends Resource> context) throws Exception {
        server.start();
    }

    public ServerManager(int port, Handler handler) throws Exception {
        ...
        Core.getGlobalContext().register(this);
    }
}
```


Time to first operation
OpenJDK
spring-boot
3898
micronaut
1001
quarkus
980
0
1000
2000
3000
4000
ms


Time to first operation
OpenJDK
OpenJDK on CRaC
spring-boot
3898
38
micronaut
1001
46
quarkus
980
33
0
1000
2000
3000
4000
ms


Time to complete N operations: Quarkus
OpenJDK
Secs
Requests
0
5
10
15
20
25
30
35
40
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000


Time to complete N operations: quarkus
OpenJDK
OpenJDK on CRaC
Secs
Requests
0
5
10
15
20
25
30
35
40
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000

CRaC

- Сохранить образ программы, использовать для развертывания
- Программа управляет состоянием
- CRaC помогает находить проблемы при сохранении
- Быстрое развертывание, Java не требует прогрева

Координация - сложно?

Координация - сложно?

Мы пользуемся фреймворками, библиотеками и часто не управляем ресурсами

Координация - сложно?

Мы пользуемся фреймворками, библиотеками и часто не управляем ресурсами

Координация требует удивительно мало кода

- В “спокойном” режиме: отпустить ресурс и приостанавливать доступ к нему

Координация - сложно?

Мы пользуемся фреймворками, библиотеками и часто не управляем ресурсами

Координация требует удивительно мало кода

- В “спокойном” режиме: отпустить ресурс и приостанавливать доступ к нему
- В “активном” режиме:
  - блокировать Checkpoint, перейти в спокойное
  - отменить Checkpoint
  - предоставить mock для ресурса

Координация - сложно?

Мы пользуемся фреймворками, библиотеками и часто не управляем ресурсами

Координация требует удивительно мало кода

- В “спокойном” режиме: отпустить ресурс и приостанавливать доступ к нему
- В “активном” режиме:
  - блокировать Checkpoint, перейти в спокойное
  - отменить Checkpoint
  - предоставить mock для ресурса

Может быть, этот код уже написан:

- Fault-tolerance: восстанавление после ошибок

Proof-of-Concepts

Фреймворки
- Spring Boot (и другие на основе Tomcat Embed)
- Quarkus
- Micronaut

Библиотеки
- JDBC Connection Pool

Приложения
- Tomcat Catalina

github.com/CRaC/quarkus/compare/master..crac-master

```
716     -     private static class WebDeploymentVerticle extends AbstractVerticle {
        717   +     private static class WebDeploymentVerticle extends AbstractVerticle implements Resource {
717     718
718     719           private HttpServer httpServer;
719     720           private HttpServer httpsServer;
        @@ -734,6 +735,8 @@ public WebDeploymentVerticle(HttpServerOptions httpOptions, HttpServerOptions ht
734     735               this.launchMode = launchMode;
735     736               this.domainSocketOptions = domainSocketOptions;
736     737               this.insecureRequests = insecureRequests;
        738   +           org.crac.Core.getGlobalContext().register(this);
        739   +
737     740           }
738     741
739     742           @Override
        @@ -882,6 +885,24 @@ public void stop(Future<Void> stopFuture) {
882     885                   domainSocketServer.close(handleClose);
883     886               }
884     887           }
        888   +
        889   +       @Override
        890   +       public void beforeCheckpoint(org.crac.Context<? extends Resource> context) throws Exception {
        891   +           CountDownLatch latch = new CountDownLatch(1);
        892   +           stop(Future.future(event -> {
        893   +               latch.countDown();
        894   +           }));
        895   +           latch.await();
        896   +       }
        897   +
        898   +       @Override
        899   +       public void afterRestore(org.crac.Context<? extends Resource> context) throws Exception {
        900   +           CountDownLatch latch = new CountDownLatch(1);
        901   +           start(Future.future(event -> {
        902   +               latch.countDown();
        903   +           }));
        904   +           latch.await();
        905   +       }
```

API

```
import jdk.crac.Core;
import jdk.crac.Resource;
```

`jdk.crac` реализуется сборкой java-crac

`javax.crac`?

API

```
import jdk.crac.Core;
import jdk.crac.Resource;
```

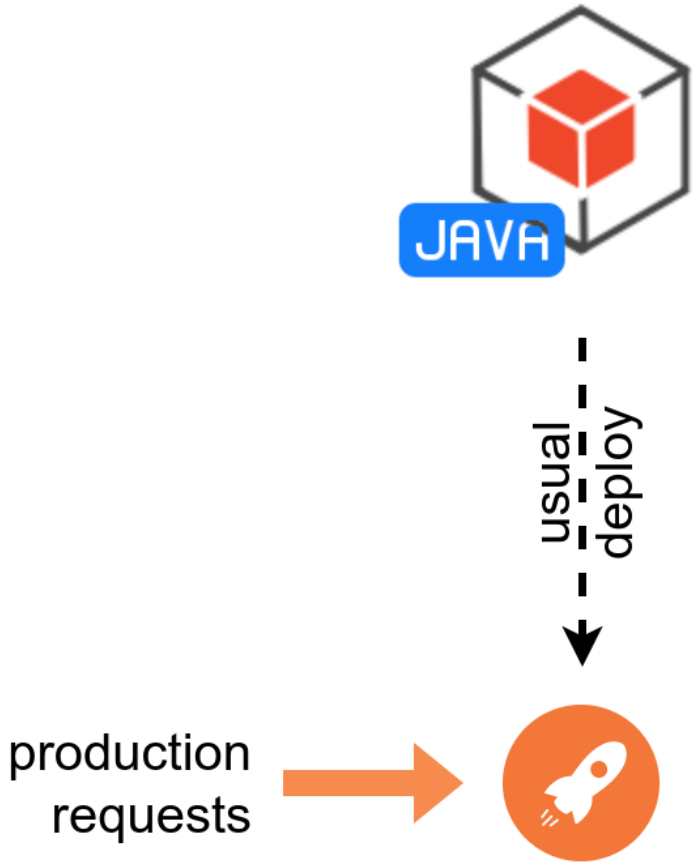
`jdk.crac` реализуется сборкой java-crac

`javax.crac`?

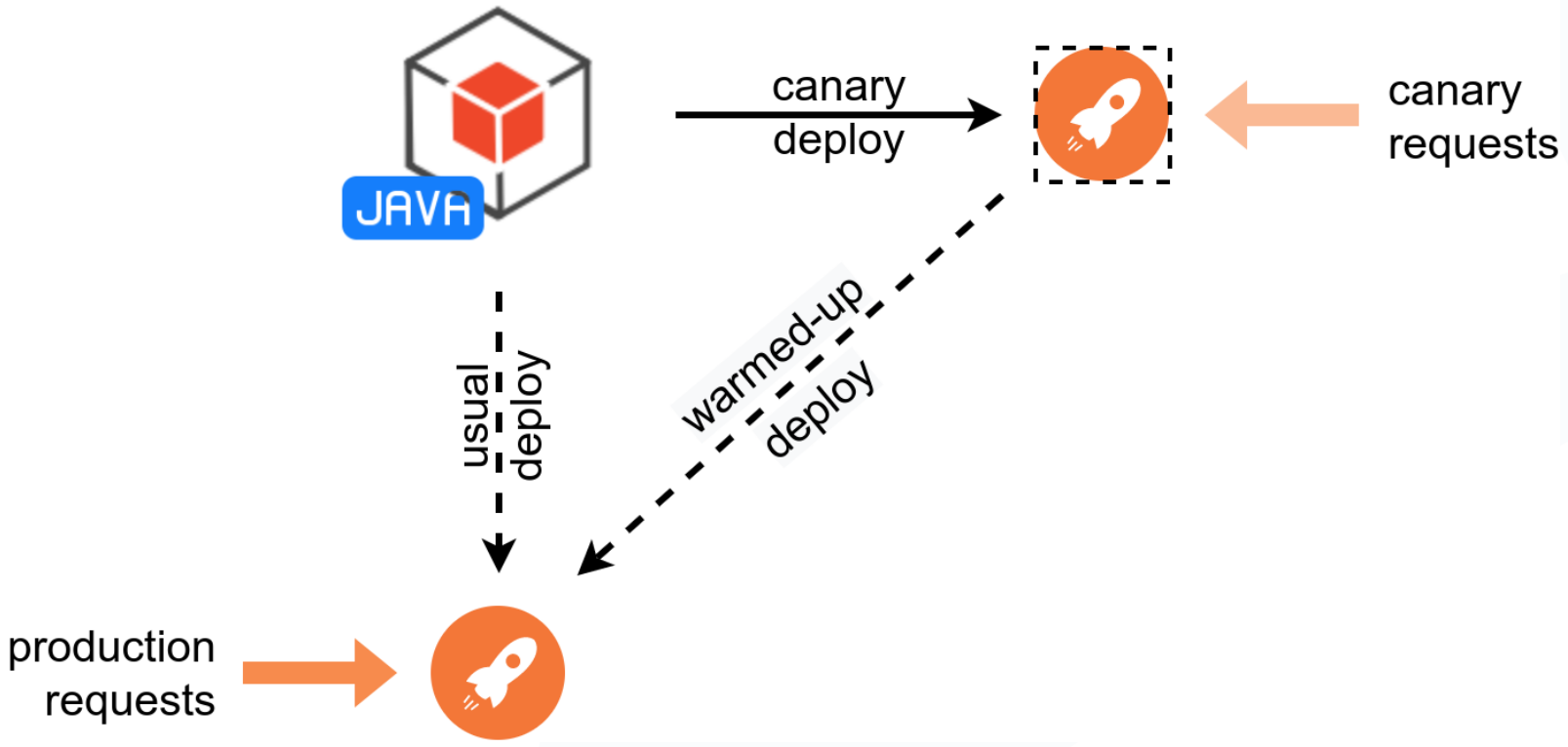
`org.crac` (библиотека)

- не зависит от JDK, не требует java-crac для сборки
- определяет наличие jdk.crac и javax.crac во время работы

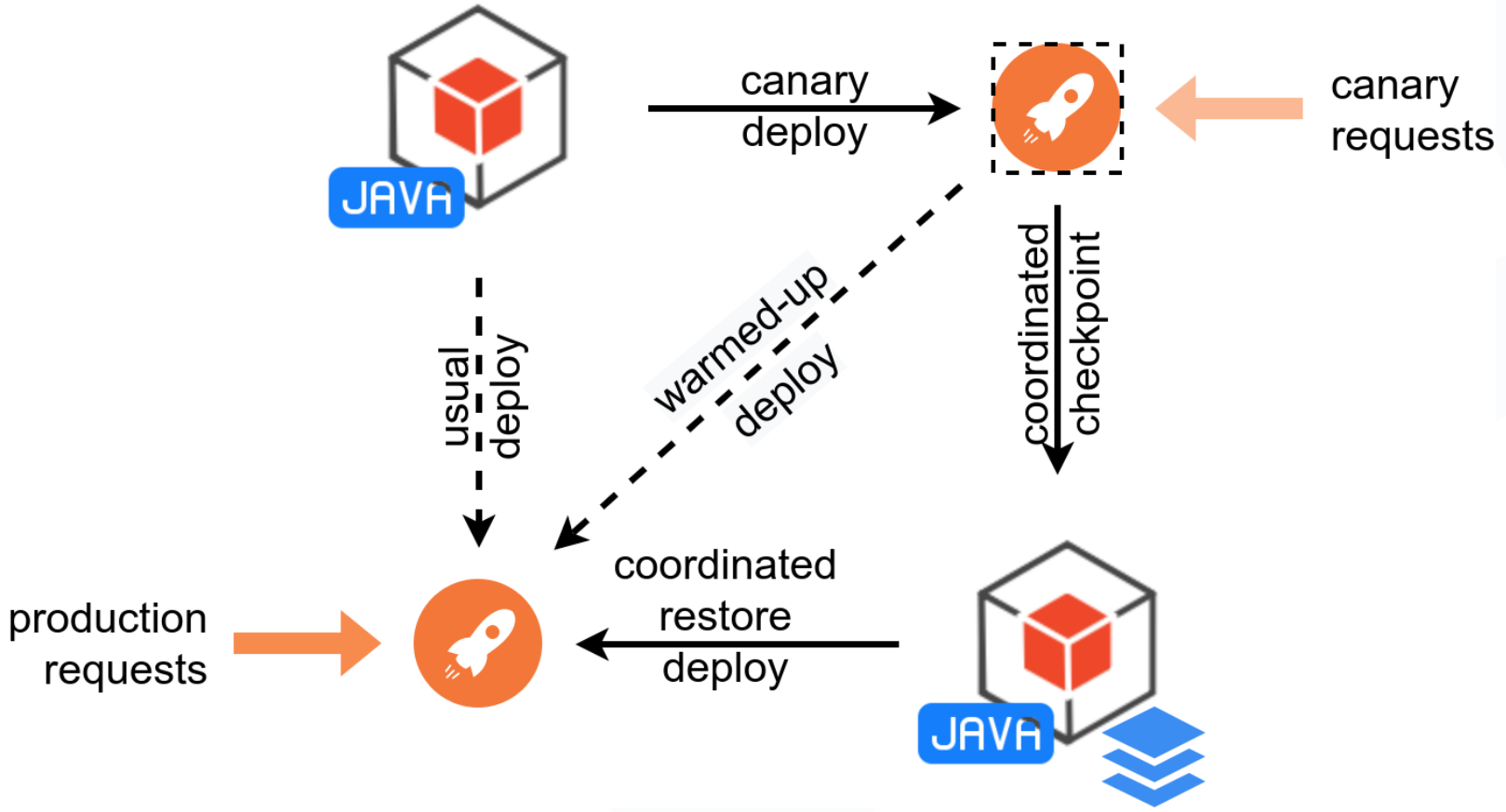
Развёртывание



Развёртывание



Развёртывание



API

- Можно прервать checkpoint (beforeCheckpoint выбрасывает исключение)

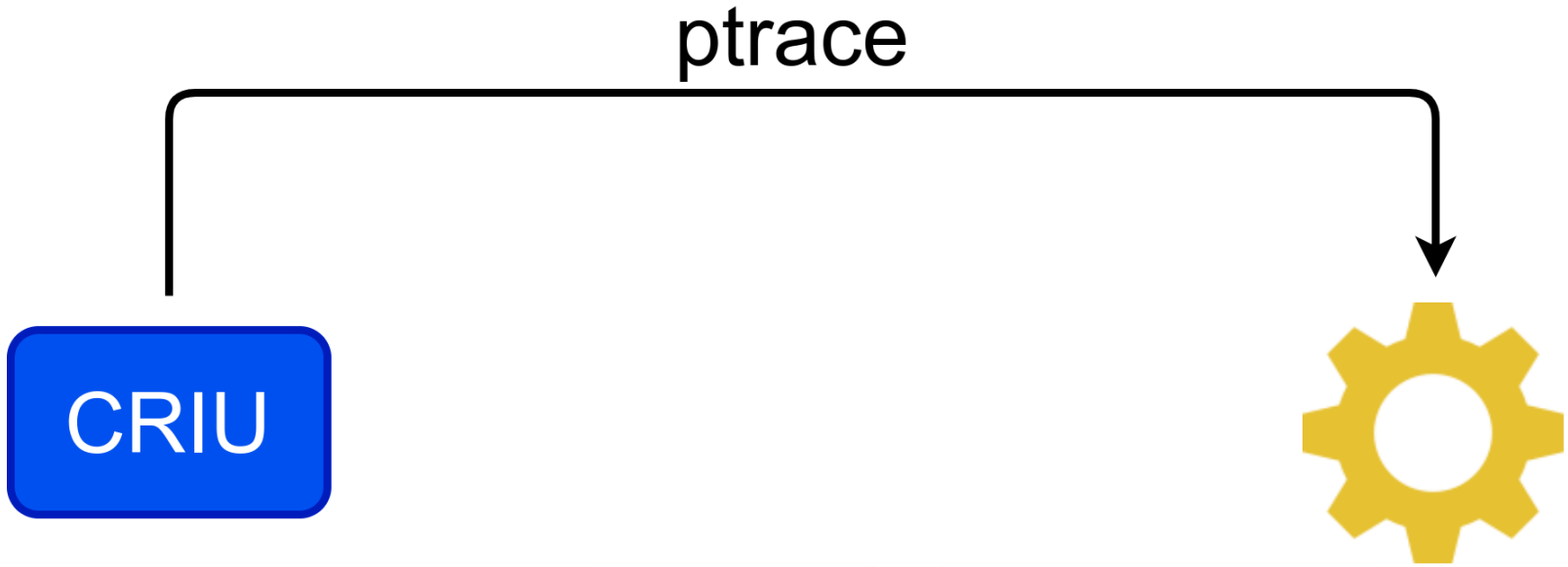
```
@Override
public void beforeCheckpoint(Context<? extends Resource> context) throws Exception;
```

- Можно восстановиться после restore (даже если afterRestore выбрасывает исключение)

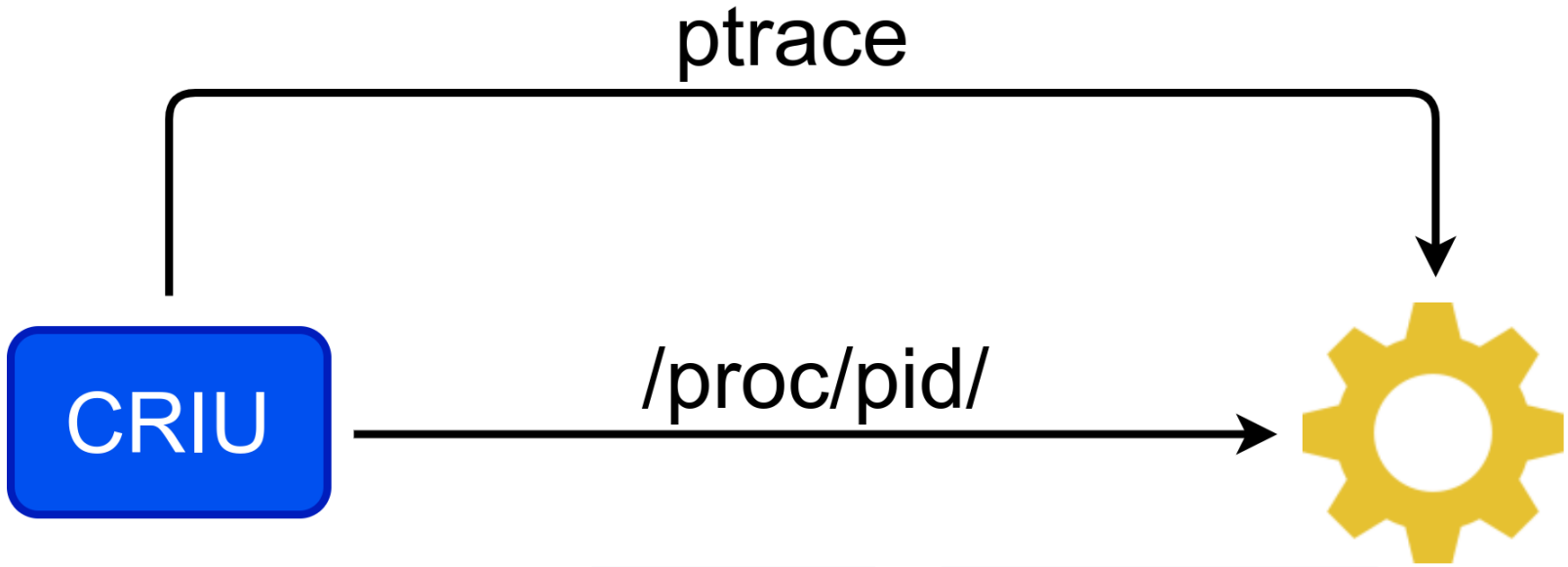
```
@Override
public void afterRestore(Context<? extends Resource> context) throws Exception;
```

CRIU: Checkpoint

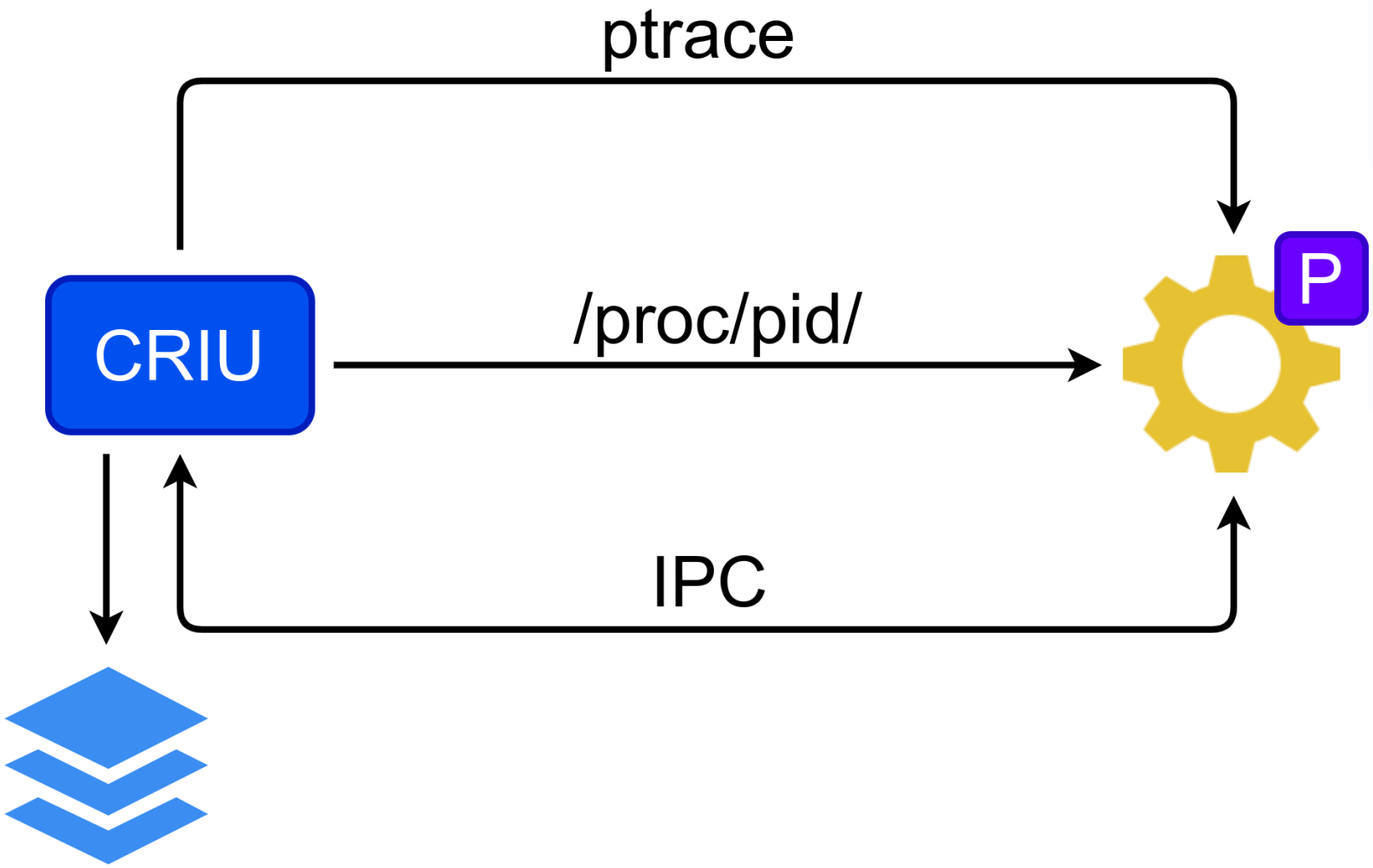
CRIU: Checkpoint



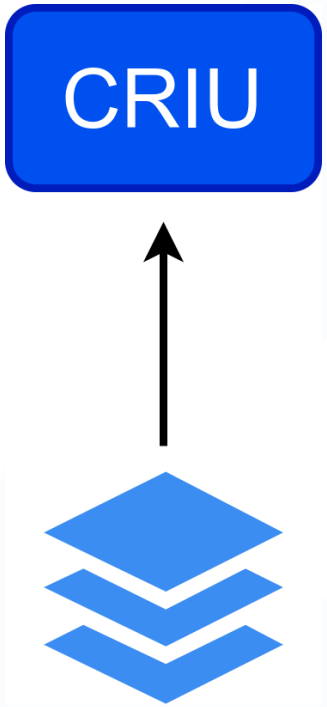
CRIU: Checkpoint



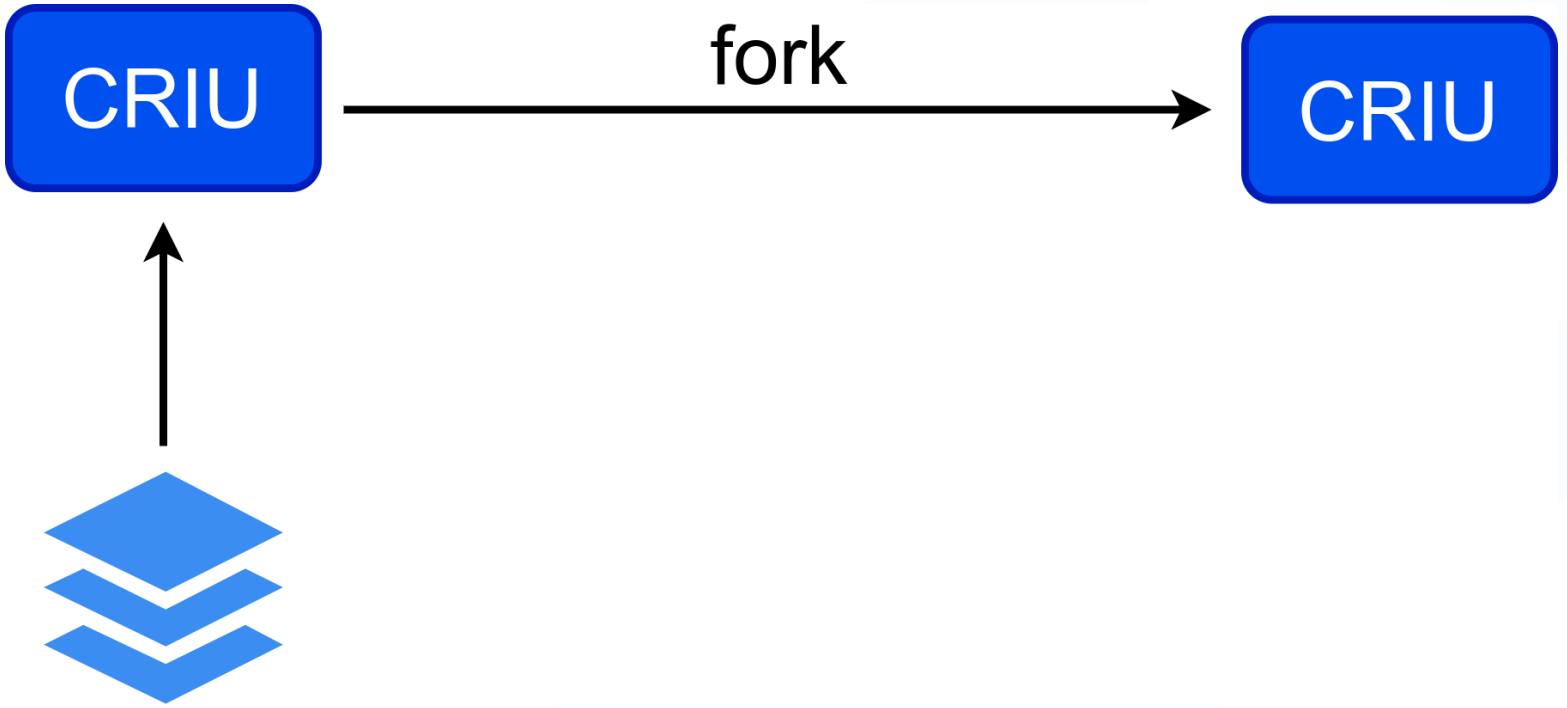
CRIU: Checkpoint



CRIU: Restore



CRIU: Restore



CRIU: Restore

CRIU

Процесс:

- регистры потоков
- общая память
- контекст операционной системы

Задачи:

- сохранить весь контекст ОС
- дерево процессов: `producer | filter`
- активные сетевые соединения

Координация с JVM

beforeCheckpoint, afterRestore вызываются в указанных фазах.

- Координированный checkpoint:
  - beforeCheckpoint's
  - safepoint:
    - проверка состояния (дескрипторы,..)
    - [подготовка JVM]
    - ожидание restore
    - CRIU checkpoint
- Координированный restore:
  - CRIU restore + оповещение JVM
  - safepoint:
    - конец ожидания
    - [инициализация JVM]
  - afterRestore's

Проблема: jcmd

- jcmd/jps/... обнаруживают другие JVM с помощью специального файла
  - `/tmp/hsperfdata_<user>/<pid>`
  - `-XX:+UsePerfData`
- файл удаляется при завершении VM

Проблема: jcmd

- jcmd/jps/... обнаруживают другие JVM с помощью специального файла
  - `/tmp/hsperfdata_<user>/<pid>`
  - `-XX:+UsePerfData`
- файл удаляется при завершении VM

Решение:

- до checkpoint, сохраняем содержимое в память или другой файл
- после restore, восстанавливаем файл

Проблема: размер образа

- выделяем много мусора до checkpoint
- CRIU сохраняет полный образ процесса

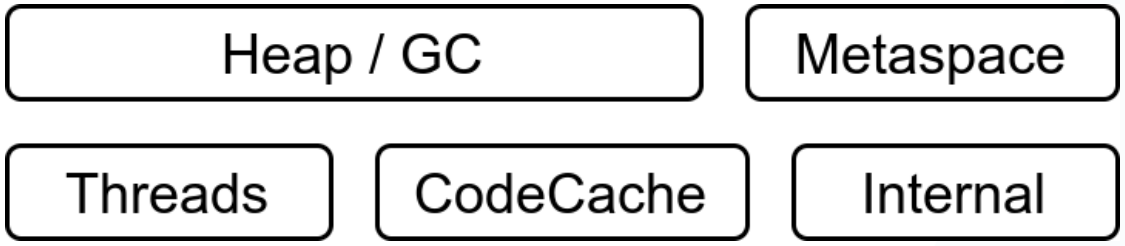
Проблема: размер образа

- выделяем много мусора до checkpoint
- CRIU сохраняет полный образ процесса

Решение:

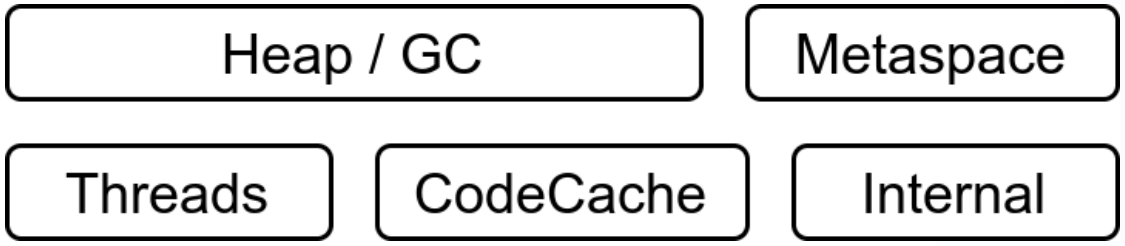
- чистые страницы не отличимы ещё не выделенных
- CRIU не включает полностью чистые страницы в образ
- делаем Full GC, после очищаем страницы с мусором

Проблема: время restore



- CRIU восстанавливает карту памяти
- считывает из образа содержимое памяти
- запускает процесс

Проблема: время restore



- CRIU восстанавливает карту памяти
- считывает из образа содержимое памяти
- запускает процесс

Решение(?): отображать образ лениво

Поздние CRIU: опция lazy-pages

В будущем

- управление загрузкой страниц
  - предзагрузка критичных областей памяти
  - сжатие образа

В будущем

- управление загрузкой страниц
  - предзагрузка критичных областей памяти
  - сжатие образа
- обновление предположений JVM
  - сгенерированный код
  - размер кучи

В будущем

- управление загрузкой страниц
  - предзагрузка критичных областей памяти
  - сжатие образа
- обновление предположений JVM
  - сгенерированный код
  - размер кучи
- отказ от root привелегий для CRIU
  - ptrace

В будущем

- управление загрузкой страниц
  - предзагрузка критичных областей памяти
  - сжатие образа
- обновление предположений JVM
  - сгенерированный код
  - размер кучи
- отказ от root привелегий для CRIU
  - ptrace
- альтернативные C/R механизмы
  - не-Linux ОС?

[Dan Heidinga. Everyone wants fast startup]

CRaC

Time/space trade-off: разменять время на память

- ahead-of-time native image:
  - compile-time: проанализировать все пути исполнения
  - статическая компиляция/profile-guided-optimization/динамическая рекомпиляция (VM)

- CRaC:
  - checkpoint: выполнить один конкретный путь исполнения
  - полная JVM после Restore

CRaC

github.com/CRaC

- `java-crac`
  - CRaC API
  - реализация на CRIU
- улучшенная CRIU
- модифицированные фреймворки
- примеры использования, тесты
  - Jetty, Quarkus, Micronaut, Spring boot
- user's & programmer's tutorials

akozlov@azul.com