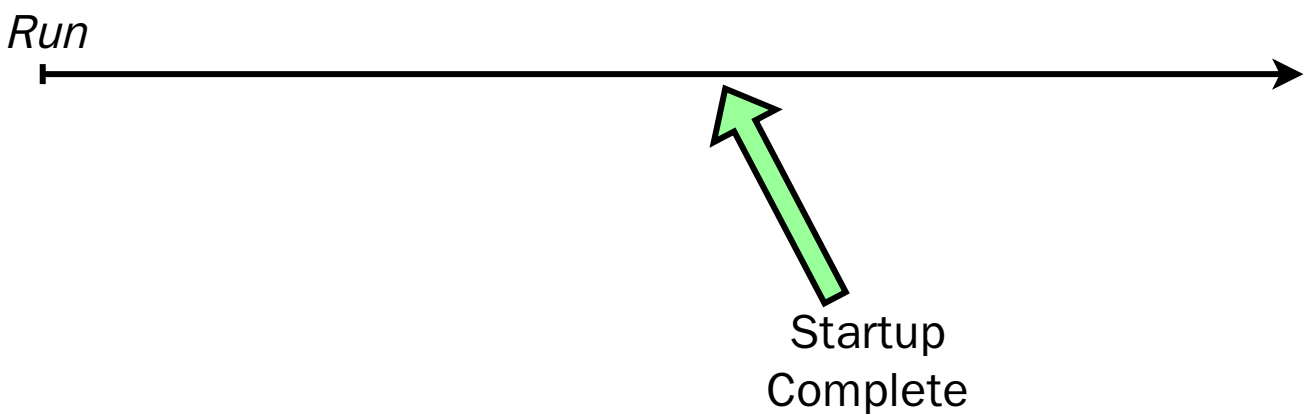
# OpenJDK Project CRaC
# Coordinated Restore at Checkpoint

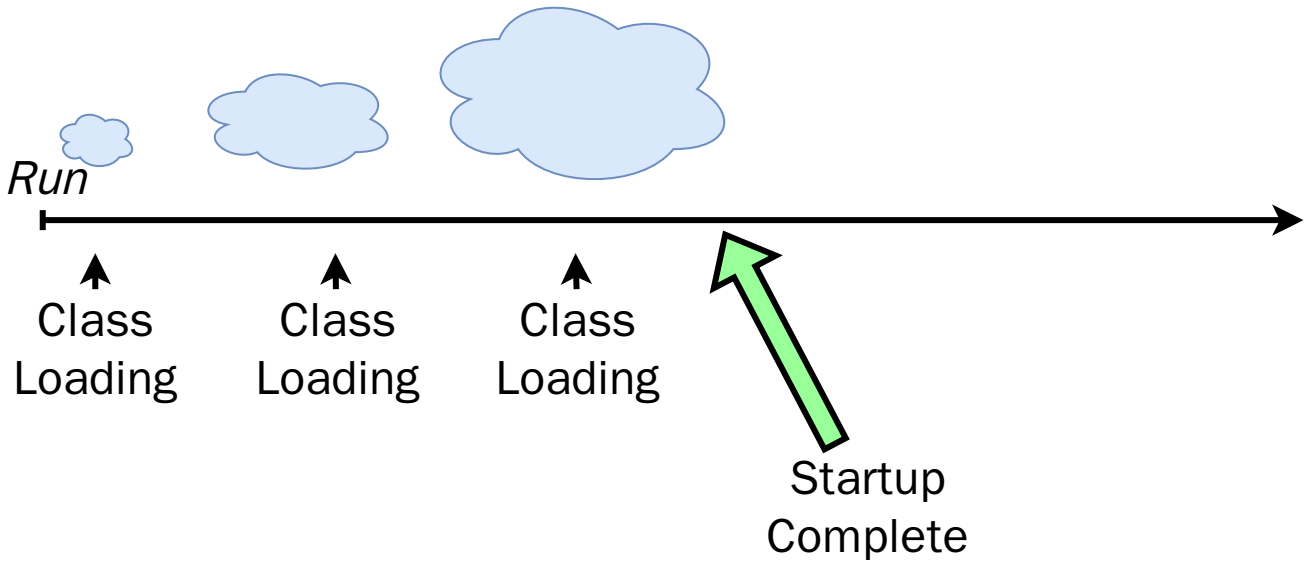
## Проблемы и решения

Антон Козлов

# Проблема start-up

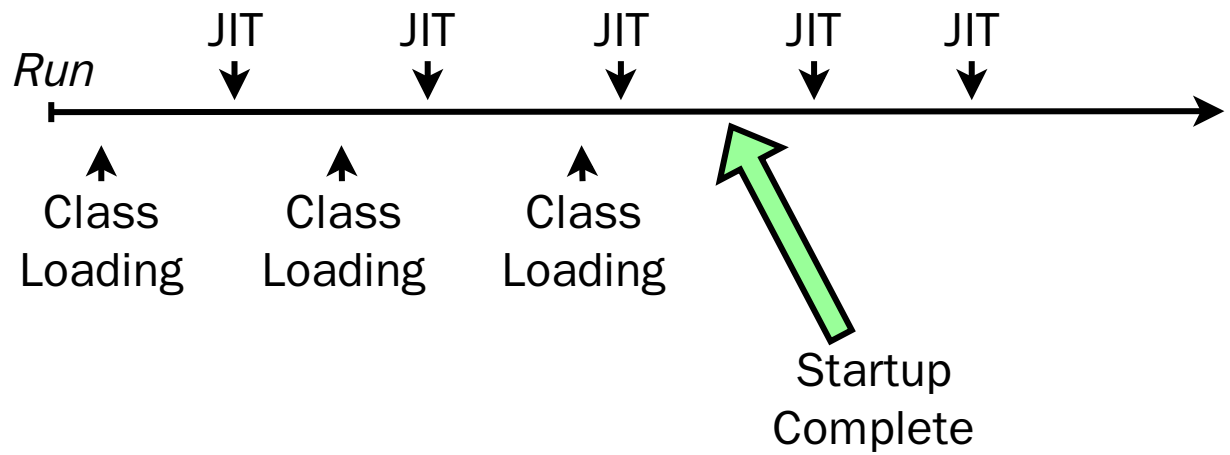


# Проблема start-up

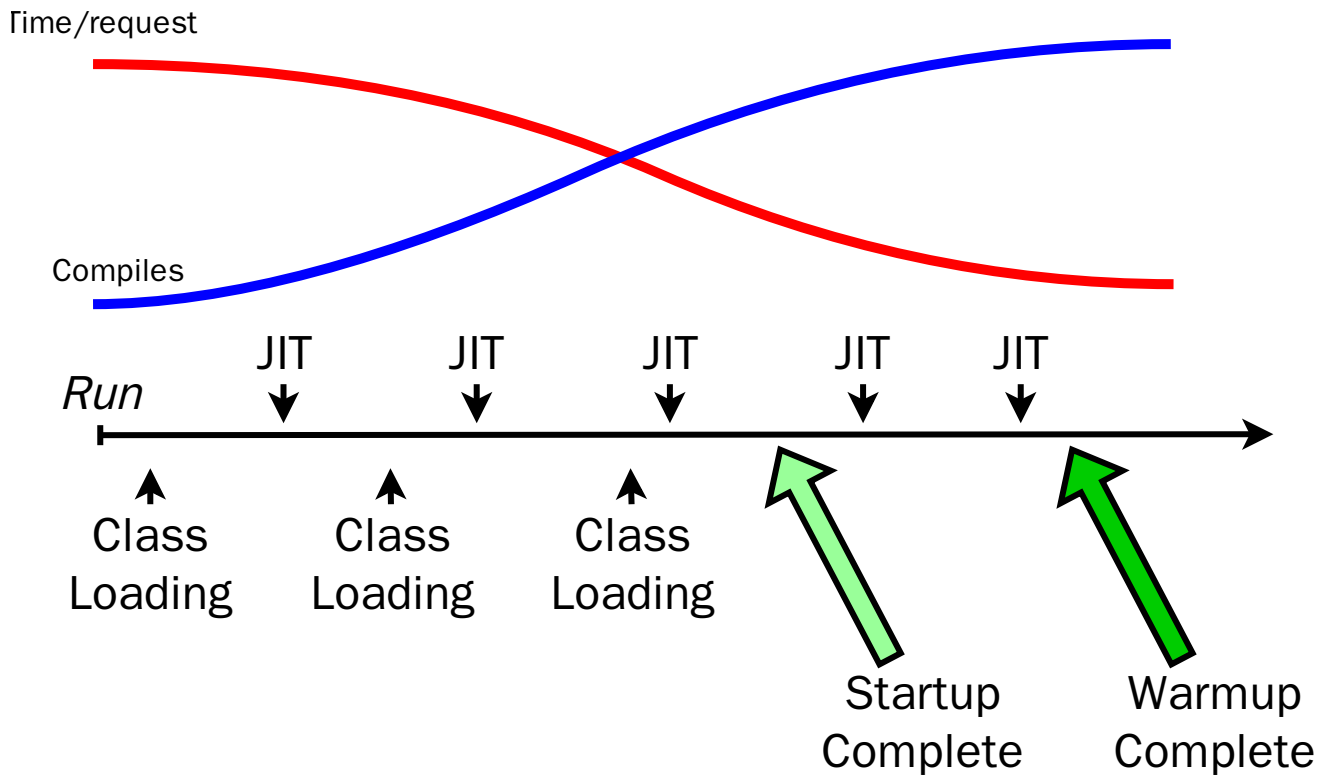



- получить байткод лениво **при первом обращении**:
  - с файловой системы: JAR, ...
  - из сети
  - сгенерировать

# Проблема start-up / warm-up



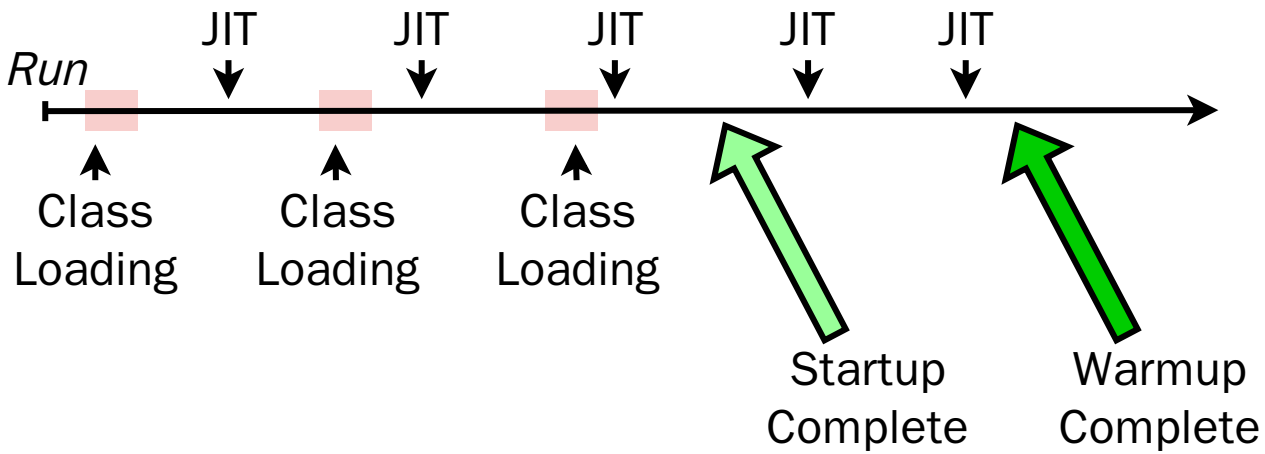
# Проблема start-up / warm-up




JIT нужен байткод

# Class Data Sharing

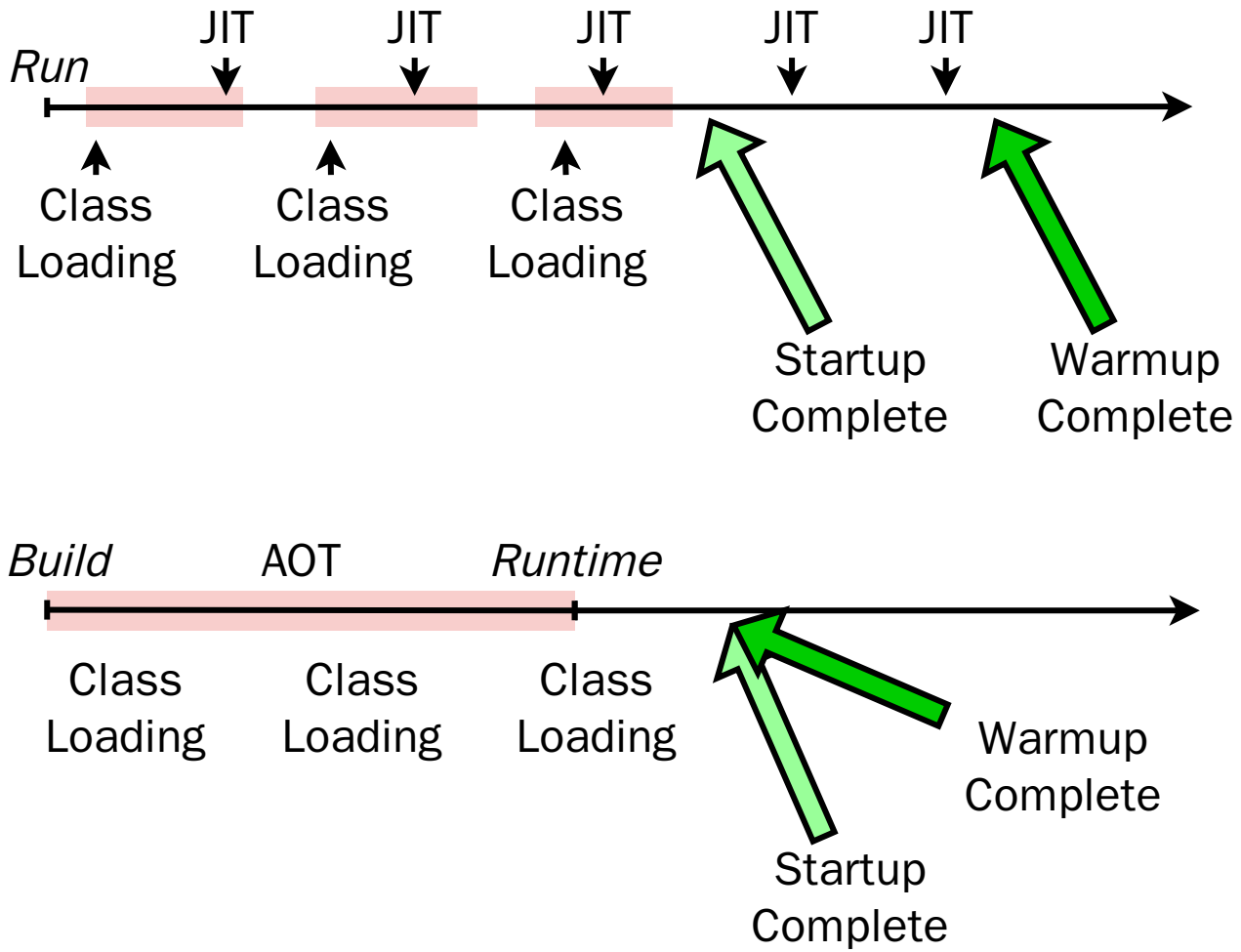
- распознавать класс при запросе на загрузку
- использовать предподоготовленный классы



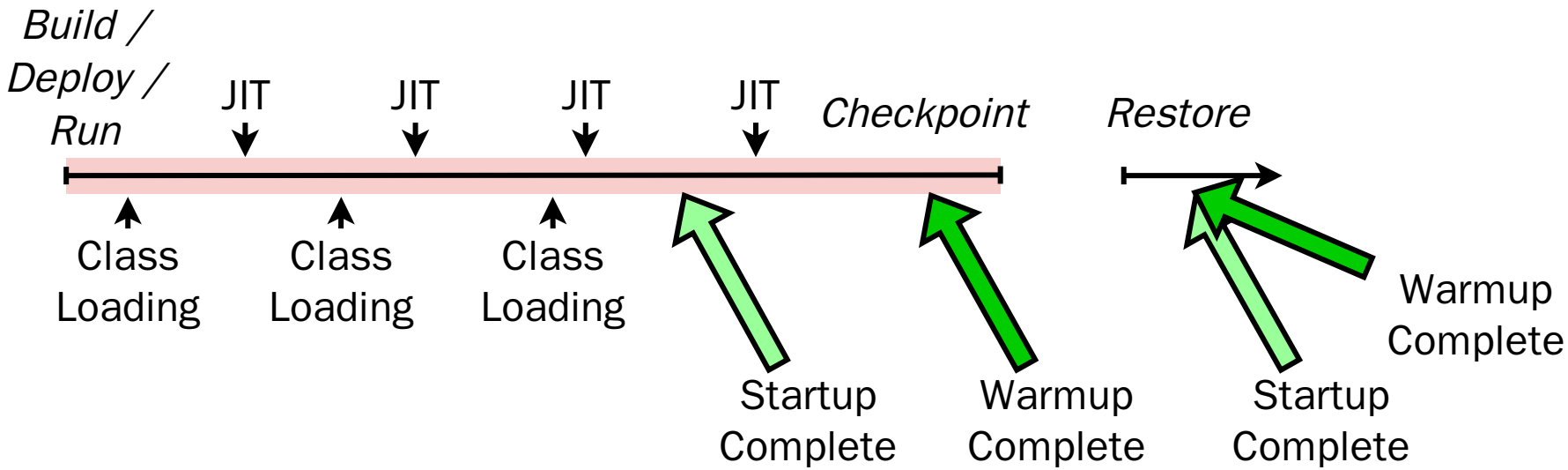

[Class data sharing in the HotSpot VM (Volker Simonis, JBreak 2018)]

JEP-350 (JDK 13): Dynamic CDS Archives

# AOT: GraalVM Native Image



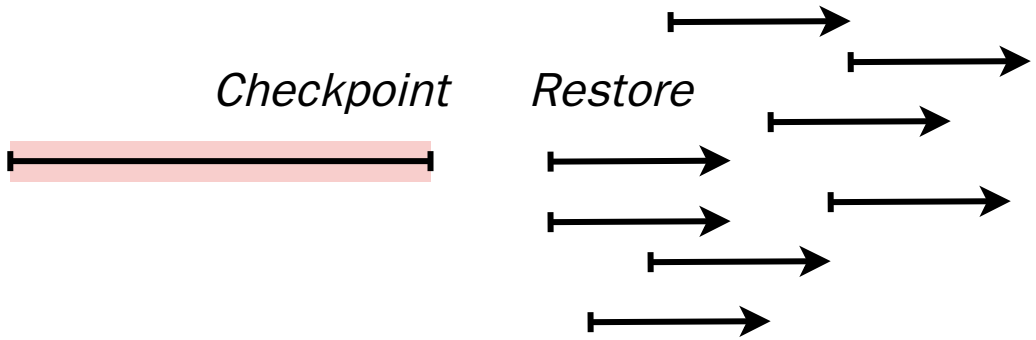
# Checkpoint и Restore



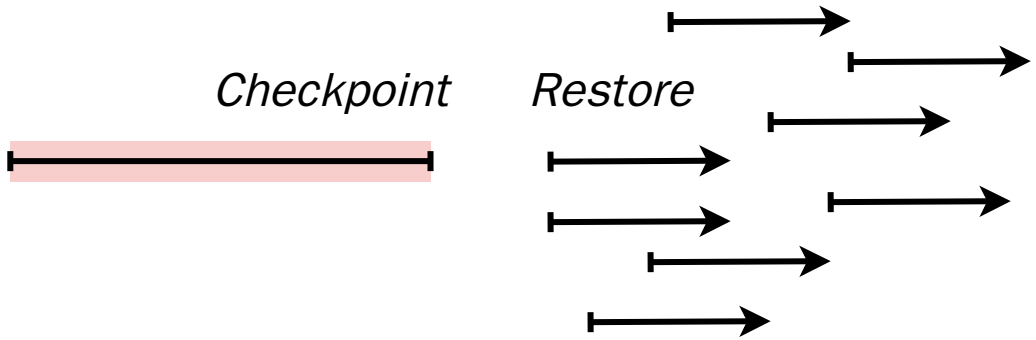

Образ JVM:

- куча
- треды и их состояния
- загруженные классы
- JIT комплияции

# Coordinated Restore at Checkpoint



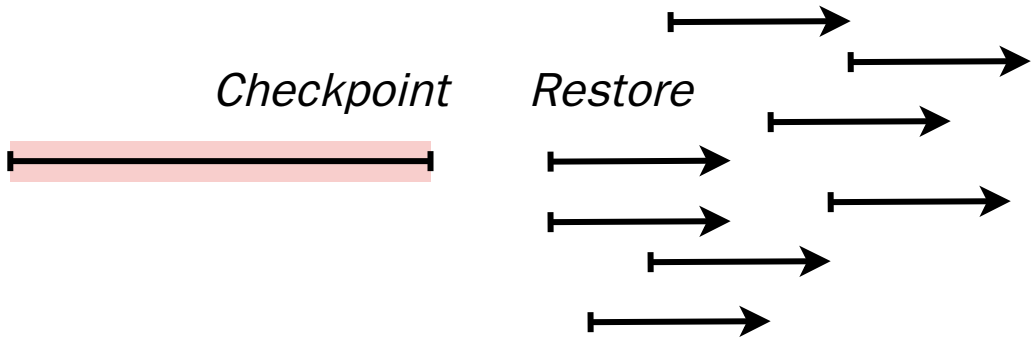
# Coordinated Restore at Checkpoint




Приложение *может* знать о Checkpoint и Restore

- обрабатывать эти события как угодно

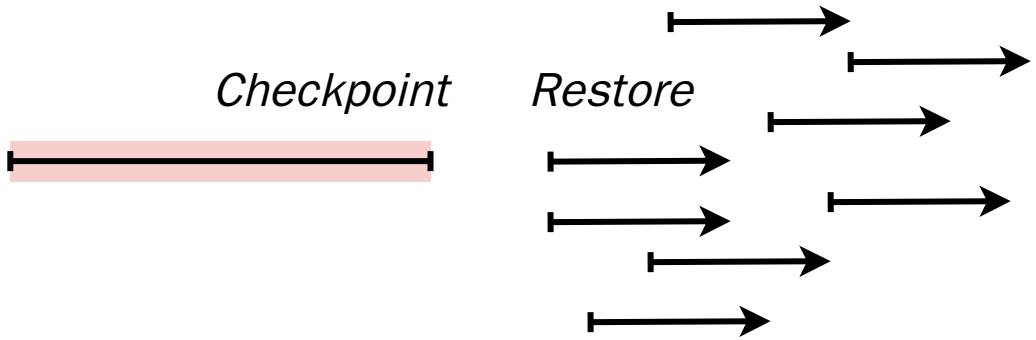
# Coordinated Restore at Checkpoint




Приложение *может* знать о Checkpoint и Restore

- обрабатывать эти события как угодно
- участвовать: приложение может отменить Checkpoint

# Coordinated Restore at Checkpoint




Приложение *может* знать о Checkpoint и Restore

- обрабатывать эти события как угодно
- участвовать: приложение может отменить Checkpoint

Приложение *должно* не иметь сетевых соединений, открытых файловых дескрипторов, ...

# OpenJDK Project!

https://openjdk.org/projects/crac

- мейллист
  - Request For Reviews
  - дизайн, идеи, ...
- репозиторий с кодом
  - сейчас JDK 17
  - (?) дальше JDK 17u + JDK 19/20/...

https://github.com/CRaC -- вспомогательный проект

- EA билды: https://crac.github.io/openjdk-builds

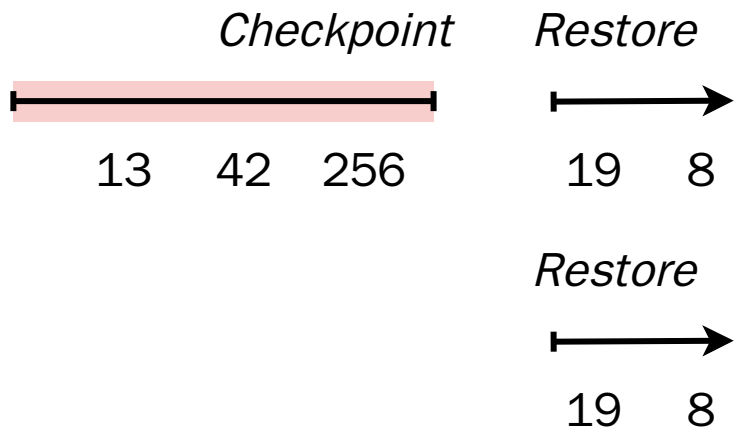
JDK -- первый и важный потребитель API для координации

- теперь можно работать вместе

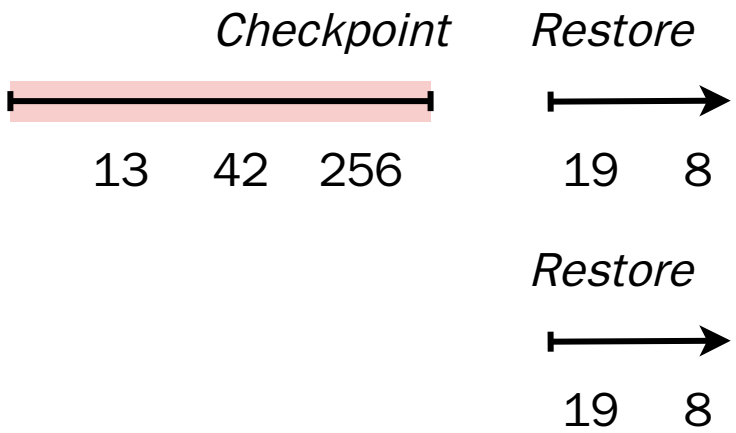
# CRaC API

```
import jdk.crac.*;

class ResourceManager implements Resource {
    Server server;

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

    }

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

    }

    public Manager() {
        ...
        Core.getGlobalContext().register(this);
    }
}
```

# java.util.Random



# `java.util.Random`




- PseudoRandom Number Generator
  - `random, state = f(state)`, `f` известна
- никак не обрабатывается
  - seed устанвливается явно или неявно во время конструктора

# java.util.Random

```
public class RandomUser {
    final static Random rand = new Random();

    @Override
    void afterRestore() {
        rand.setSeed((new Random()).nextLong());
    }
}
```

(Примерный код!)

# java.security.SecureRandom

- нельзя предсказать seed после восстановления
- нельзя считать seed из файла образа

# java.security.SecureRandom

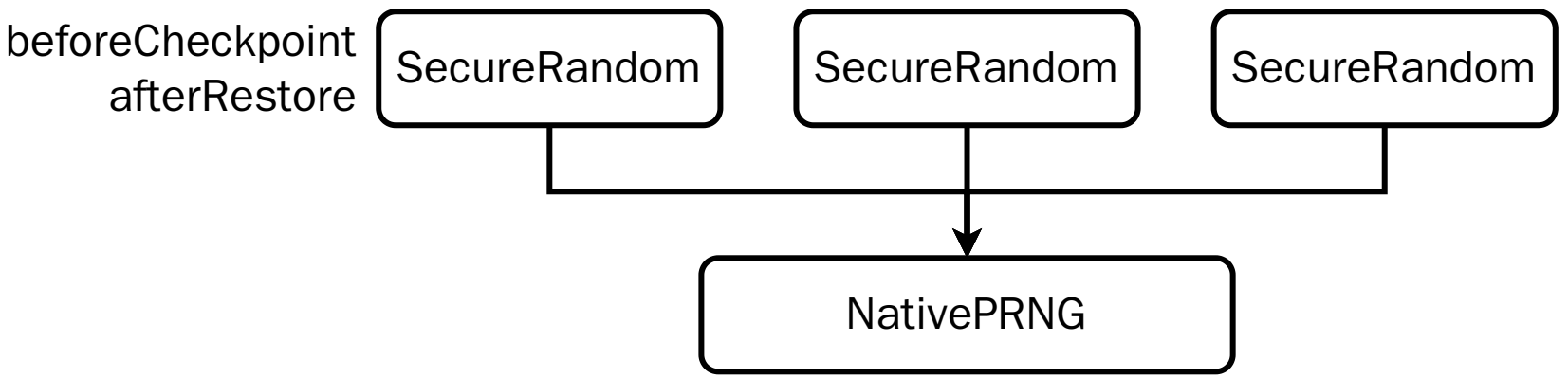
- нельзя предсказать seed после восстановления
- нельзя считать seed из файла образа
- объект блокируется между beforeCheckpoint и afterRestore

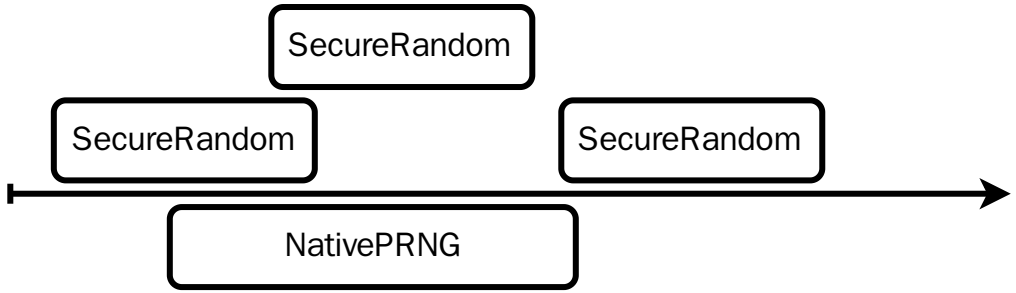
```
    public void operation() {
        lock.lock();
        // ...
        lock.unlock();
    }
    @Override
    public void beforeCheckpoint(Context<? extends Resource> context) throws Exception {
        lock.lock();
        // ...
    }
    @Override
    public void afterRestore(Context<? extends Resource> context) throws Exception {
        // ...
        lock.unlock();
    }
```

# NativePRNG

Cryptographically Strong RNG -- (?)RNG

- для скорости: PRNG + энтропия
- энтропия дорогая и имеет свойство заканчиваться
- `j.s.SecureRandom`ы получают энтропию из NativePRNG
  - нужно сделать Resource






```
public void op1() {
    rwLock.readLock().lock();
    // ...
    rwLock.readLock().unlock();
}
public void op2() {
    rwLock.readLock().lock();
    // ...
    rwLock.readLock().unlock();
}
@Override
public void beforeCheckpoint(Context<? extends Resource> context) throws Exception {
    rwLock.writeLock().lock();
    // ...
}
@Override
public void afterRestore(Context<? extends Resource> context) throws Exception {
    // ...
    rwLock.writeLock().unlock();
}
```

# Context & JDKContext

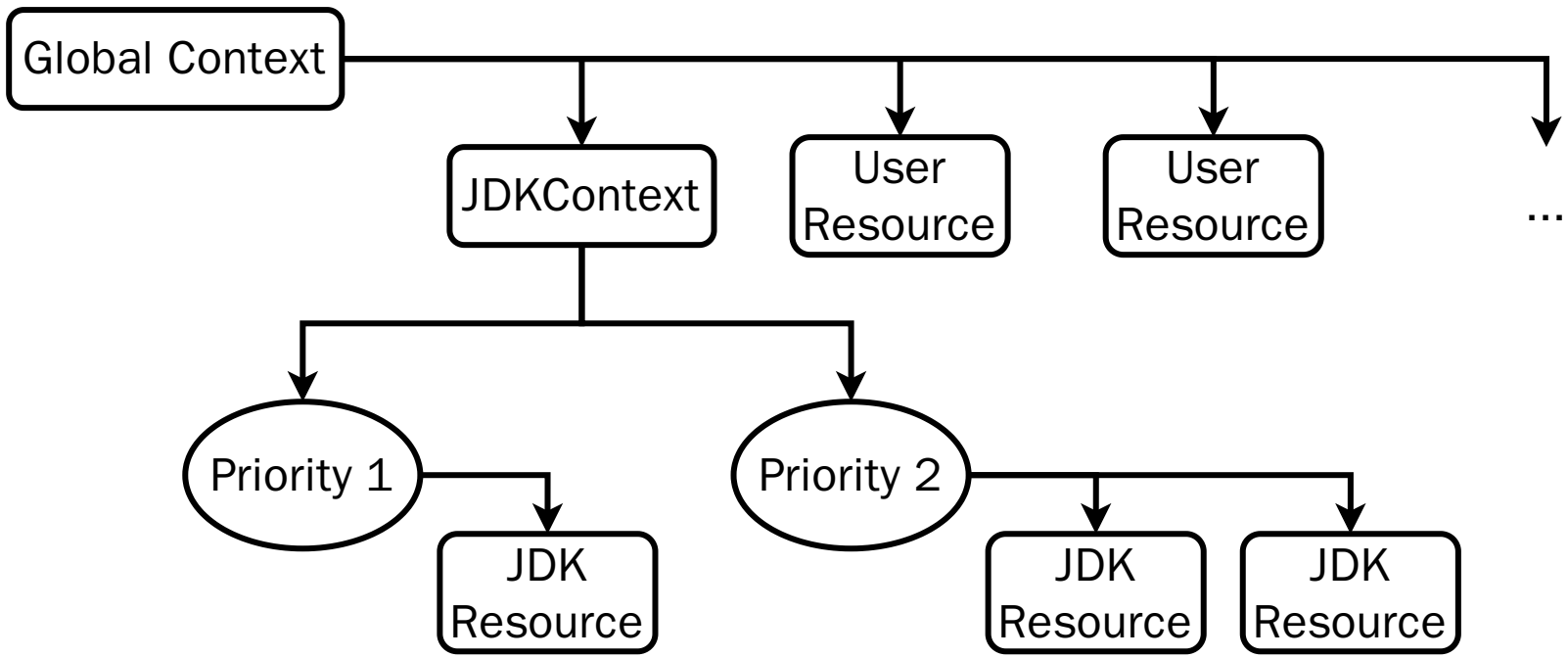
## Упорядочение Resource

```
    Core.getGlobalContext().register((Resrouce)r);
```

```
public abstract class Context<R extends Resource> implements Resource {
    public abstract void register(R resource);
}
```

```
public interface JDKResource extends Resource {
    enum Priority { ... };
    Priority getPriority();
}

public class JDKContext extends Context<JDKResource> {
    int compare(Map.Entry<JDKResource, Void> o1, Map.Entry<JDKResource, Void> o2) {
    }
}
```

# Структура ресурсов




t

# Использование API

Не обязательно!

# Использование API

Не обязательно!

Может использоваться:

- для объекта
- класса (обновление статических полей)
- модуля приложения
- приложения

# Использование API

Не обязательно!

Может использоваться:

- для объекта
- класса (обновление статических полей)
- модуля приложения
- приложения

`Context.register()` создаёт слабую ссылку

- не влияет на достижимость, объекты могут быть собраны GC
- afterRestore гарантирован, если был вызван beforeCheckpoint

# JVM

- CRIU для реализации Checkpoint и Restore
  - JVM не зависит от CRIU напрямую

# JVM

- CRIU для реализации Checkpoint и Restore
  - JVM не зависит от CRIU напрямую
- не CRIU:
  - симуляция Checkpoint, немедленный Restore
    - для разработки самого CRaC
    - или приложений
    - основа для будущих реализаций

# JVM

- CRIU для реализации Checkpoint и Restore
  - JVM не зависит от CRIU напрямую
- не CRIU:
  - симуляция Checkpoint, немедленный Restore
    - для разработки самого CRaC
    - или приложений
    - основа для будущих реализаций
  - симуляция Checkpoint, пауза
    - эффекты с реальным временем
    - основа для снэпшотов VM, docker pause, ...

# CRIU

Unprivileged Restore!

С контролем PID на Checkpoint и Restore

- PID не должен меняться
  - Реализация Hotspot, libc, CRIU, ...
  - `j.l.ProcessHandle` (?) (JDK9+)

# CRIU

Unprivileged Restore!

С контролем PID на Checkpoint и Restore

- PID не должен меняться
  - Реализация Hotspot, libc, CRIU, ...
  - `j.l.ProcessHandle` (?) (JDK9+)
- Linux: установить PID -- привелегированная операция

# CRIU

Unprivileged Restore!

С контролем PID на Checkpoint и Restore

- PID не должен меняться
  - Реализация Hotspot, libc, CRIU, ...
  - `j.l.ProcessHandle` (?) (JDK9+)
- Linux: установить PID -- привелегированная операция
- PID bumping
  - PID выдаются последовательно
  - создавать пустые треды/процессы

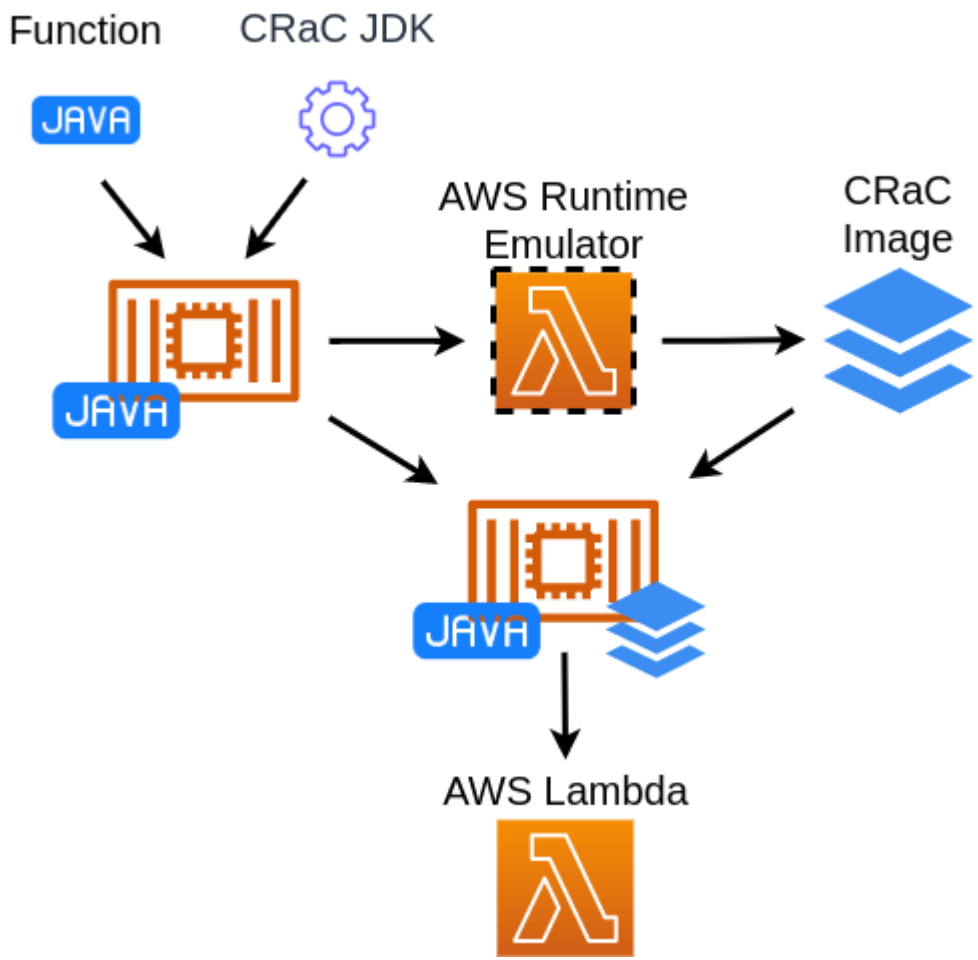
# CRIU

Unprivileged Restore!

С контролем PID на Checkpoint и Restore

- PID не должен меняться
  - Реализация Hotspot, libc, CRIU, ...
  - `j.l.ProcessHandle` (?) (JDK9+)
- Linux: установить PID -- привелегированная операция
- PID bumping
  - PID выдаются последовательно
  - создавать пустые треды/процессы
- Обеспечить
  - PID должен быть небольшим на Checkpoint
  - PID должен быть свободен на Restore

# AWS Java Lambda on CRaC



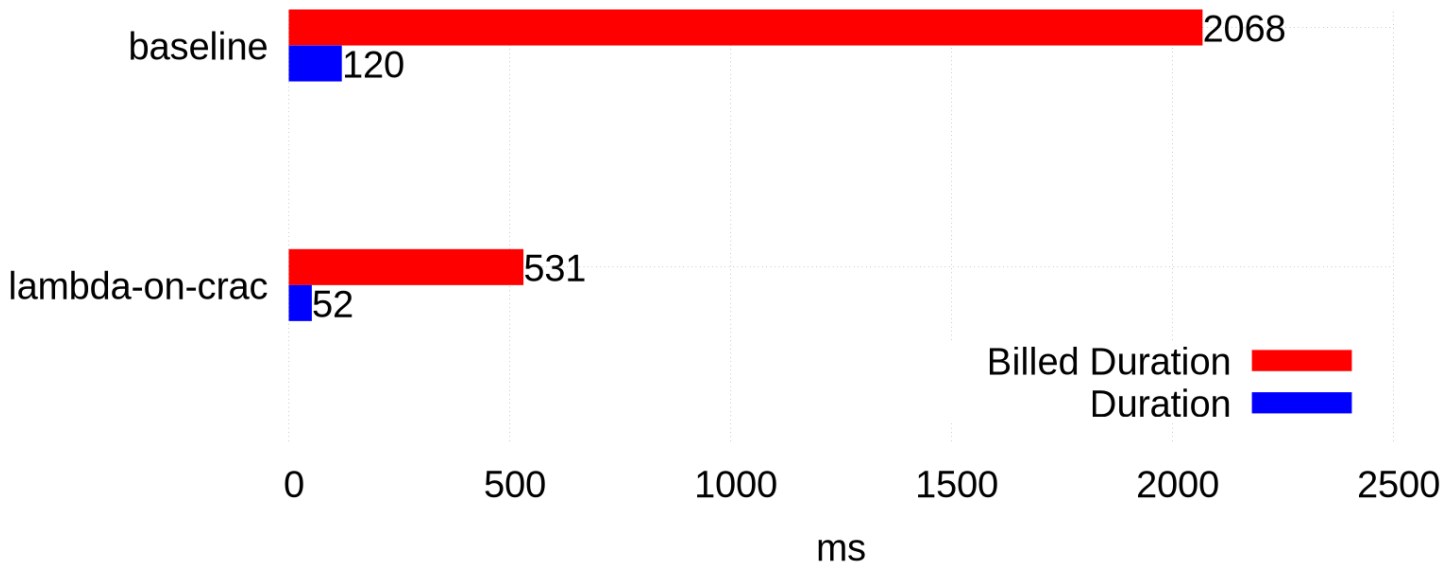
# AWS Java Lambda on CRaC

https://github.com/CRaC/example-lambda

- Управление PID, оключить CDS для CRaC
- Ограничить io/cpu для локального тестирования
- Компрессия образа CRaC

# AWS Java Lambda on CRaC

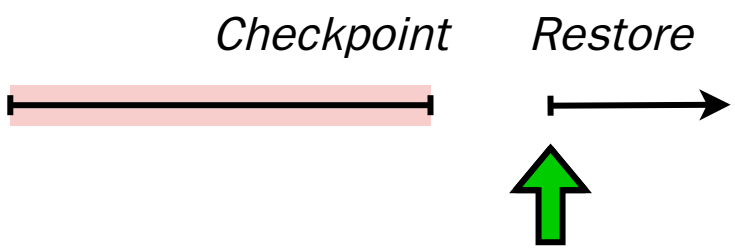
https://github.com/CRaC/example-lambda



# Restore с аргументами

```
java -XX:CRaCCheckpointTo=image ...
```

```
java -XX:CRaCRestoreFrom=image -Dprop=val Class arg1 arg2 ...
```




```
System.setProperty("prop", "val");
Class.main("arg1", "arg2");
```

# Javac on CRaC

```
com.sun.tools.javac.Main.compile(checkpointArgs);
if (checkpoint())
  Core.checkpointRestore();
```

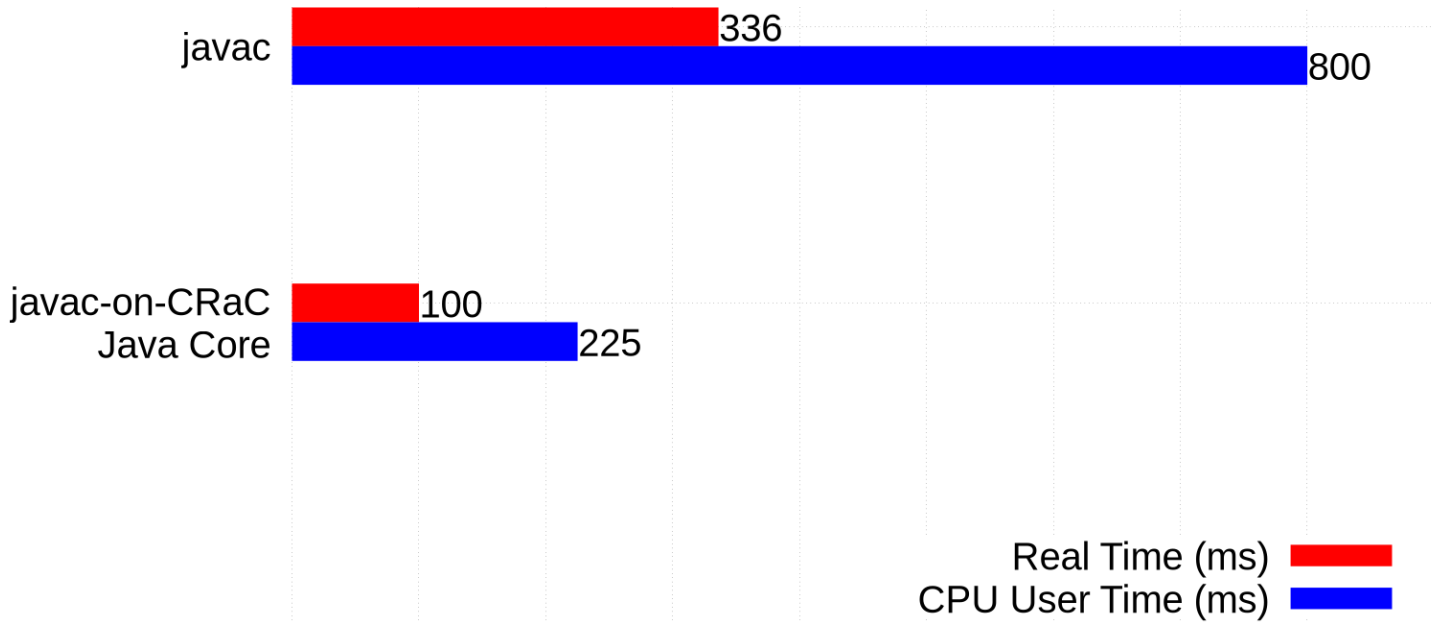
Workflow:

```
java -XX:CRaCCheckpointTo=image Compile CompileAndSaveTheData.java
java -XX:CRaCRestoreFrom=image  Compile CompileFast.java
```

# Javac on CRaC

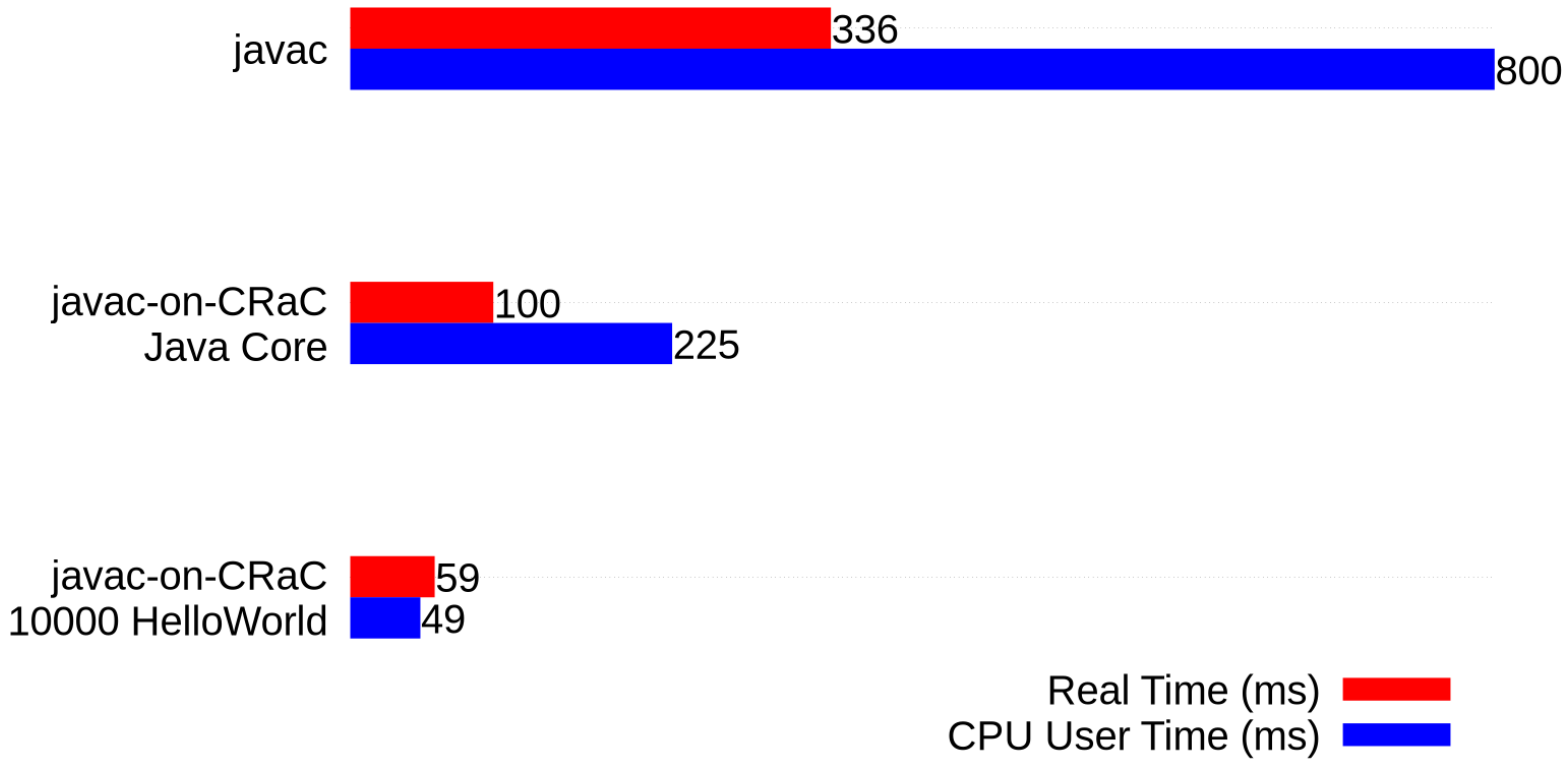
```
com.sun.tools.javac.Main.compile(checkpointArgs);
if (checkpoint())
  Core.checkpointRestore();
```

```
java -XX:CRaCCheckpointTo=image Compile java.base/** java.desktop/** java.xml/**
java -XX:CRaCRestoreFrom=image  Compile HelloWorld.java
```



# Javac on CRaC

```
java -XX:CRaCCheckpointTo=image Compile -n 10000 HelloWorld.java
java -XX:CRaCRestoreFrom=image  Compile          HelloWorld.java
```



# CRaC: Work in Progress

CRaC и время:

- java.lang.*::*(int timeout) успешно переживают Checkpoint и Restore
- делается в Safepoint
  - Safepoint – и пусть весь мир подождёт (Андрей Паньгин, JPoint 2020
- … ?

# CRaC: Work in Progress

CRaC и время:

- java.lang.*::*(int timeout) успешно переживают Checkpoint и Restore
- делается в Safepoint
  - Safepoint – и пусть весь мир подождёт (Андрей Паньгин, JPoint 2020
- ... ?

CRaC и UI: https://github.com/openjdk/crac/pull/19

- успшный Checkpoint на Linux/X11
- можно нарисовать новый интерфейс после Restore

# Spring Boot Example

`io.github.crac.*` в Maven Central:

```
21  22          <dependencies>
    23  +           <dependency>
    24  +               <groupId>io.github.crac.org.apache.tomcat.embed</groupId>
    25  +               <artifactId>tomcat-embed-core</artifactId>
    26  +               <version>8.5.75</version>
    27  +           </dependency>
22  28          <dependency>
23  29              <groupId>org.springframework.boot</groupId>
24  30              <artifactId>spring-boot-starter-web</artifactId>
    31  +           <exclusions>
    32  +               <exclusion>
    33  +                   <groupId>org.apache.tomcat.embed</groupId>
    34  +                   <artifactId>tomcat-embed-core</artifactId>
    35  +               </exclusion>
    36  +           </exclusions>
25  37          </dependency>
```

# Micronaut Example

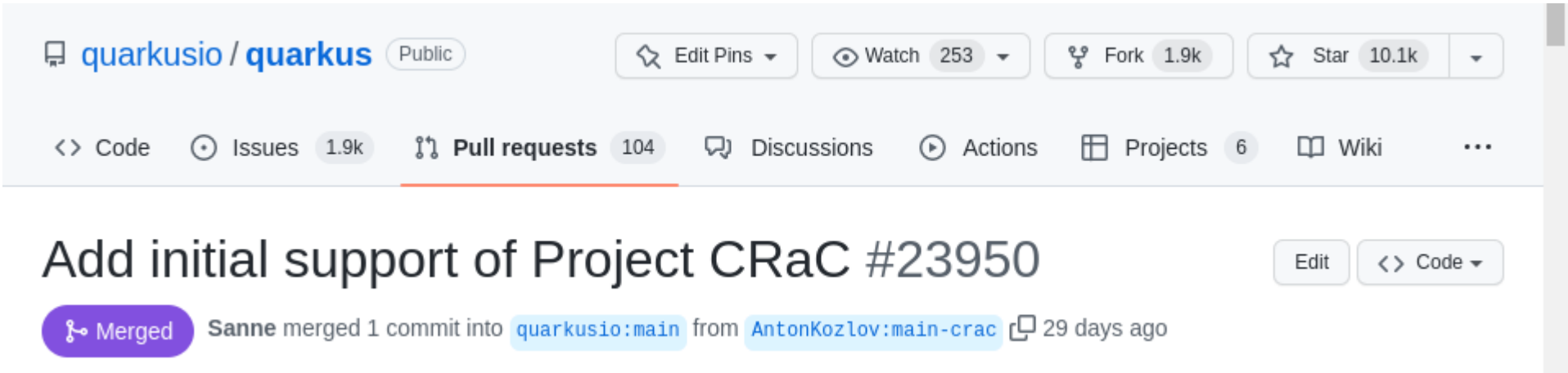
2 build.gradle

```
@@ -27,7 +27,7 @@ dependencies {
27 27      implementation "io.swagger.core.v3:swagger-annotations"
28 28      implementation "io.micronaut:micronaut-validation"
29 29      implementation "io.micronaut:micronaut-runtime"
30    -    implementation "io.micronaut:micronaut-http-server-netty"
   30 +    implementation "io.github.crac.io.micronaut:micronaut-http-server-netty:$micronautVersion"
31 31      implementation "io.micronaut:micronaut-http-client"
32 32      runtimeOnly "ch.qos.logback:logback-classic:1.2.3"
33 33      testAnnotationProcessor platform("io.micronaut:micronaut-bom:$micronautVersion")
```

2 gradle.properties

```
@@ -1 +1 @@
1    - micronautVersion=1.3.3
   1 + micronautVersion=1.3.7
```

# CRaC в mainline Quarkus!




Планируется в v2.10+

# Выводы: Project CRaC

Код: openjdk.org/projects/crac
- Билды: github.com/CRaC/openjdk-builds/releases

Примеры: github.com/CRaC/
- example-spring-boot
- example-quarkus
- example-micronaut
- example-lambda

Рассылка: mail.openjdk.org/mailman/listinfo/crac-dev

# Bonus

# JarFile Cache & ...

Quarkus: кеширует открытые JarFileы

- нельзя Checkpoint

# JarFile Cache & ...

Quarkus: кеширует открытые JarFileы

- нельзя Checkpoint

Кеш в JDK открываемых JarFile

- обычно состояние открытых JarFile не так важно
- но не в CRaC

# Reference Handling

j.l.ref.Reference: представление "сырой" ссылки на объект

- объект становится недостижим => Ref добавляется в RefQueue
- Ref достается из RefQueue и обрабатывается

Обрабтка Ref:

- привязано к GC
- можно делать по событию
  - WeakHashMap: очистка при вызове некоторых методов
- может делать отдельный тред

# CRaC и References

Для CRaC важно, *когда* будет обработана Ref

- обработка может отпускать нативные ресуры
- может не успеть до checkpoint

Проблемы:

- RefQueue и обработчик(и) связаны только семантически
  - `remove()` блокирует тред(ы) до извлечения Ref
- обработчик может породить новый недостижимый объект
  - произвольная новая RefQueue

# CRaC и References

RefQueue: новый метод `waitForQueueProcessed(nThreads)`

- удостовериться в обработке Refs *на момент вызова*
  - сейчас: запускает GC
- вызывается строрнним кодом координации

Отделён от интерфейса RefQueue