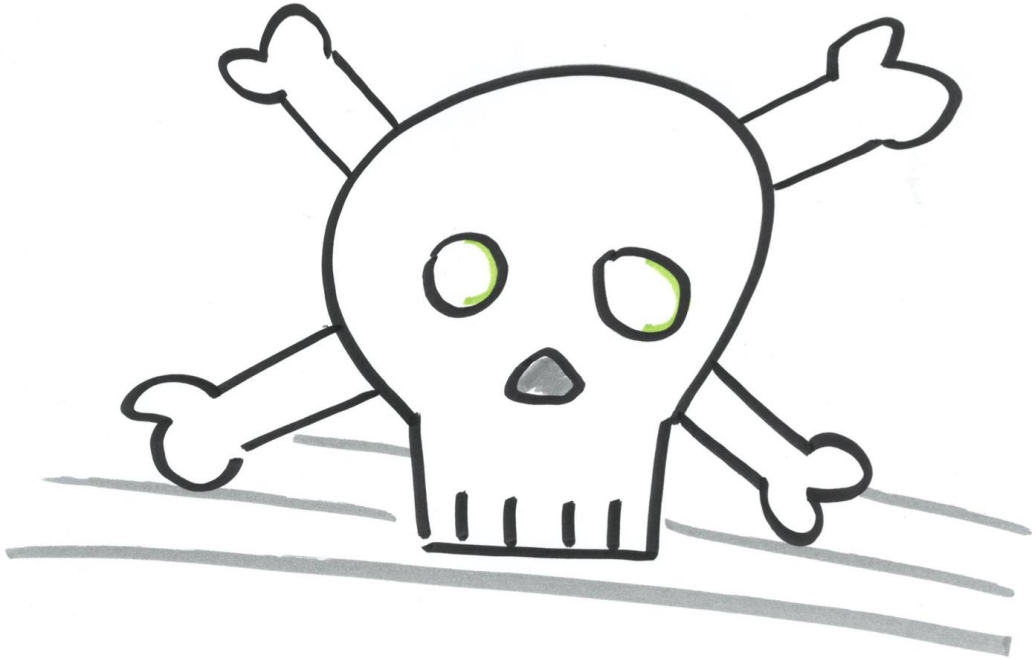


BEBECHY JVM'A DLA ŚMIERTELNIKÓW

ALEKSANDER IHNATOWICZ

WHOAMI

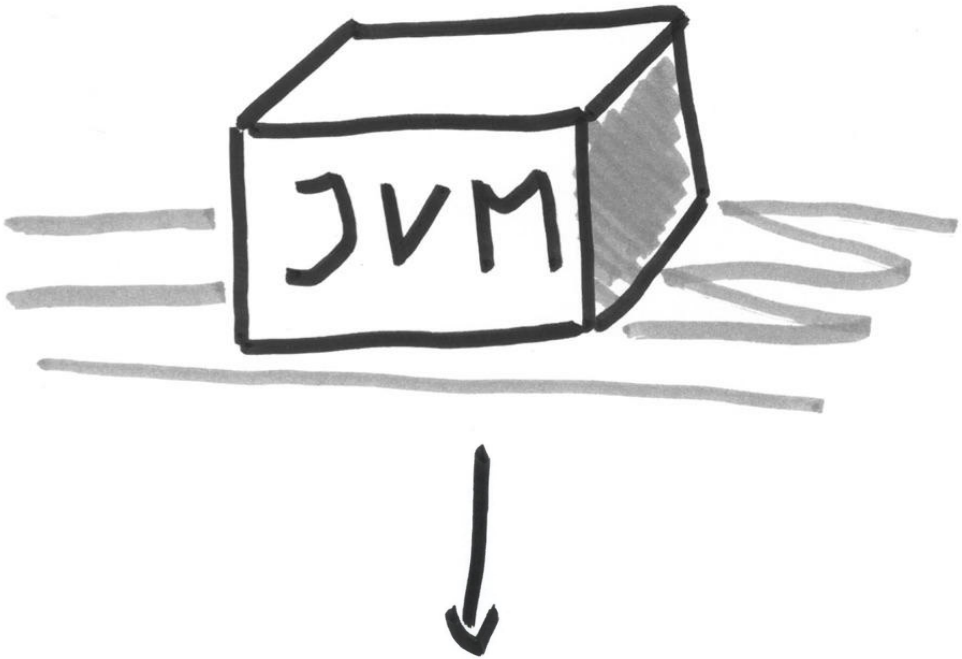
ALEKSANDER IHNATOWICZ

SENIOR SOFTWARE ENGINEER & ACTING TEAM LEADER @ ALLEGRO

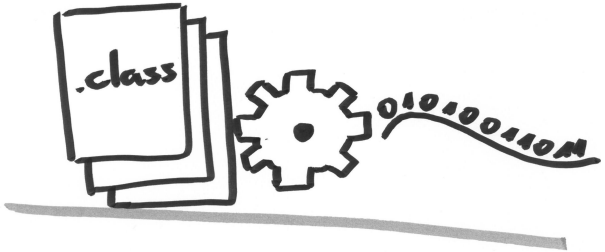
10 LAT PROGRAMUJE ZAWODOWO (4 LATA C++, TERAZ JAVA & KOTLIN)

LINKEDIN.COM/IN/ALEKSANDER-IHNATOWICZ

14

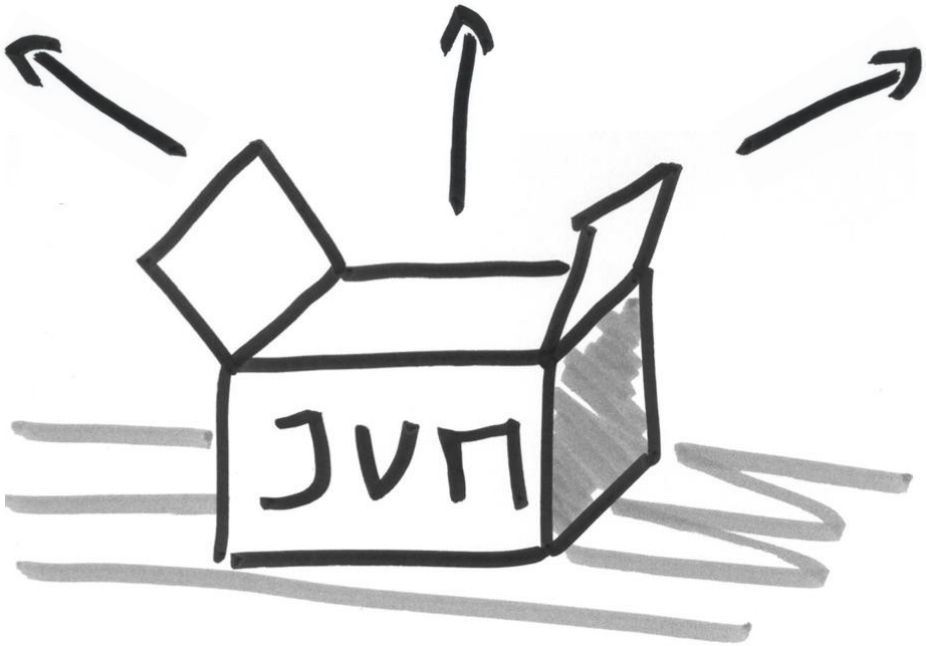

JVM

LifeCycle



Just-In-Time Compilation

Garbage Collector



Lifecycle

.class

Meta informacje: wersja java, format pliku

Informacje o klasie: flagi dostępu, nazwa klasy, klasa nadrzędna (Superklasa), interfejsy

Zawartość klasy: pola, sygnatury metod, bytecode

https://docs.oracle.com/javase/specs/jvms/se14/html/jvms-4.html

Classloader

- Bootstrap (java.lang)
- Platform (Java SE API)
- Application/System (Classpath)

Etapy ładowania klas

- Loading

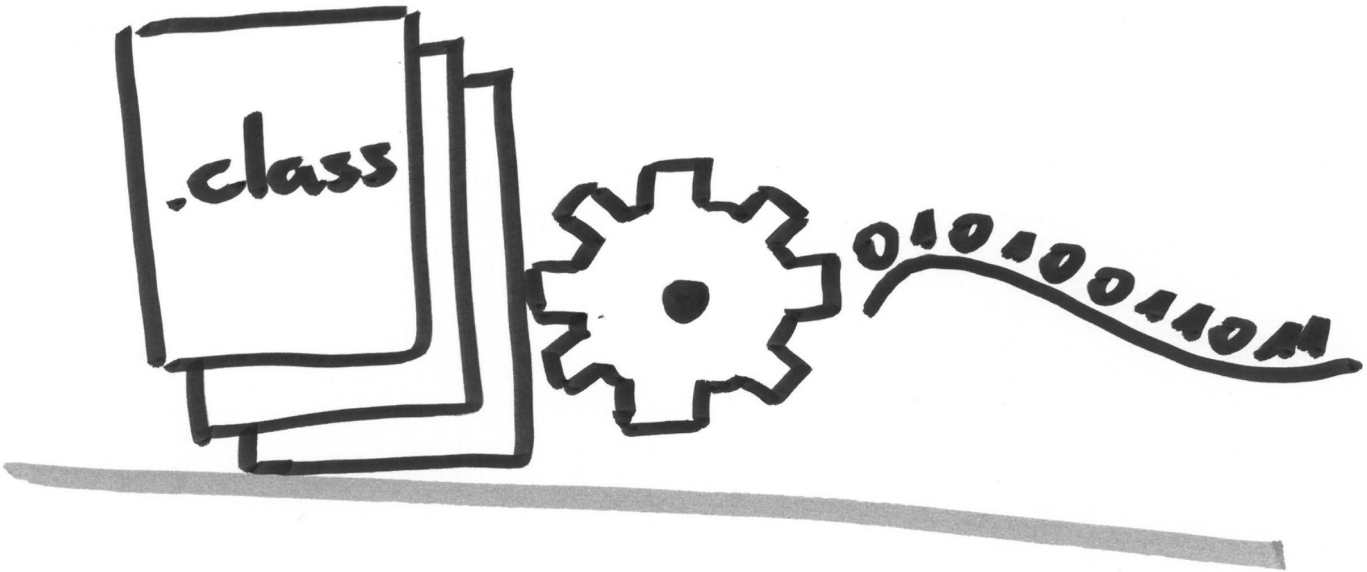
Etapy ładowania klas

- Loading
- Linking
  - Verification
  - Preparation
  - Resolution

Etapy ładowania klas

- Loading
- Linking
  - Verification
  - Preparation
  - Resolution
- Initializing

EXIT



Just-In-Time Compilation

Java Interpreter

https://metebalci.com/blog/demystifying-the-jvm-jvm-variants-cppinterpreter-and-templateinterpreter/

Kompilator Just-In-Time

```
public class CountUppercase {

    public static void main(String[] args) {
        String sentence = "In 2019 I would like to run ALL languages in one JVM";
        long total = 0, start = System.currentTimeMillis(), last = start;
        for (int i = 1; i < 10_000_000; i++) {
            total += countUppercase(sentence);
            if (i % 1_000_000 == 0) {
                long now = System.currentTimeMillis();
                System.out.printf("%d (%d ms)%n", i / 1_000_000, now - last);
                last = now;
            }
        }
        System.out.printf("total: %d (%d ms)%n", total, System.currentTimeMillis() - start);
    }

    static long countUppercase(String sentence) {
        return sentence.chars().filter(Character::isUpperCase).count();
    }
}
```

Źródło: https://www.graalvm.org/docs/examples/java-performance-examples/

```
java -XX:+UnlockExperimentalVMOptions -XX:+UseJVMCICompiler CountUppercase
1 (1062 ms)
2 (254 ms)
3 (223 ms)
4 (198 ms)
5 (164 ms)
6 (167 ms)
7 (158 ms)
8 (162 ms)
9 (162 ms)
total: 79999992 (2708 ms)
```

Tiered compilation

0: Interpreted code

1: Simple C1 compiled code (with no profiling)

2: Limited C1 compiled code (with light profiling)

3: Full C1 compiled code (with full profiling)

4: C2 compiled code (uses profile data from the previous steps)

Optymalizacje JIT

- method inlining
- dead code elimination
- loop unrolling
- escape analysis
- type sharpening
- ...

Method inlining

```
int sum(int a, int b) {
        return a + b;
}

int compute(int a, int b, int c, int d) {
        return sum(sum(a, b), sum(c, d));
}
```

```
int compute(int a, int b, int c, int d) {
        return a + b + c + d;
}
```

Dead code elimination

```
public void myMethod() {
        for (int i = 0; i < THRESHOLD; i++) {
                new String("test");
        }
}
```

```
public void myMethod() {}
```

Loop unrolling

```
void sum() {
        int sum = 0
        for(int i = 0; i < 5; i++) {
                sum =+ i
        }
}
```

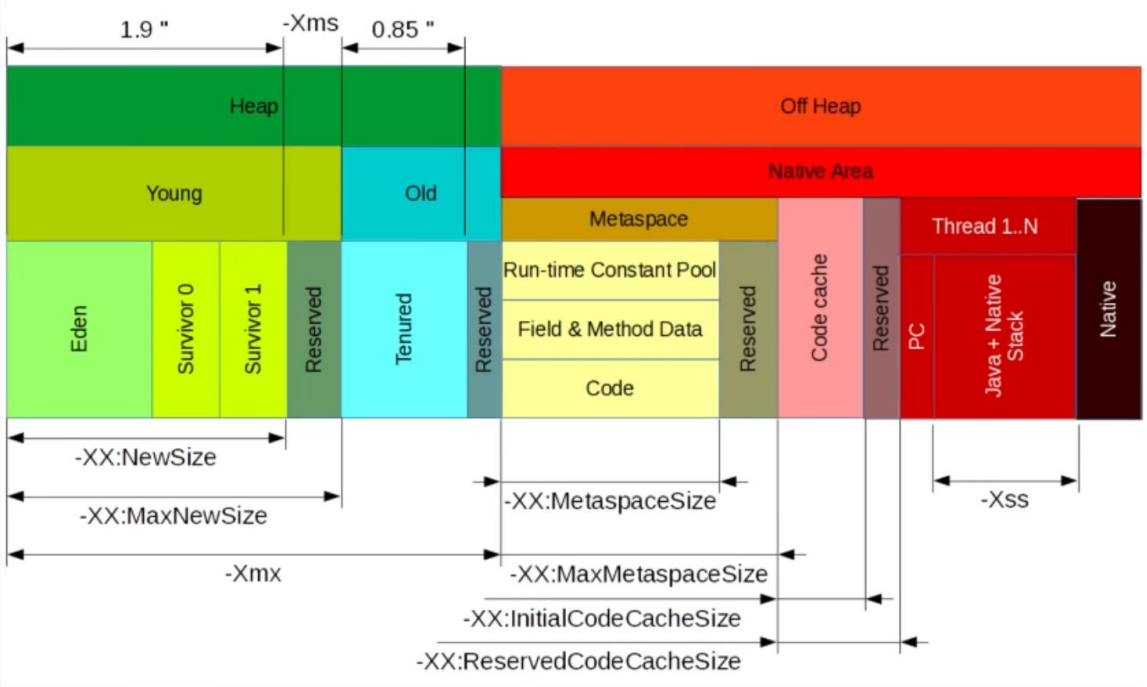
```
void sum() {
        int sum = 0
        sum =+ 0
        sum =+ 1
        sum =+ 2
        sum =+ 3
        sum =+ 4
}
```

Graal

https://chrisseaton.com/truffleruby/jokerconf17/
https://www.youtube.com/watch?v=_7yIUkP5LiQ

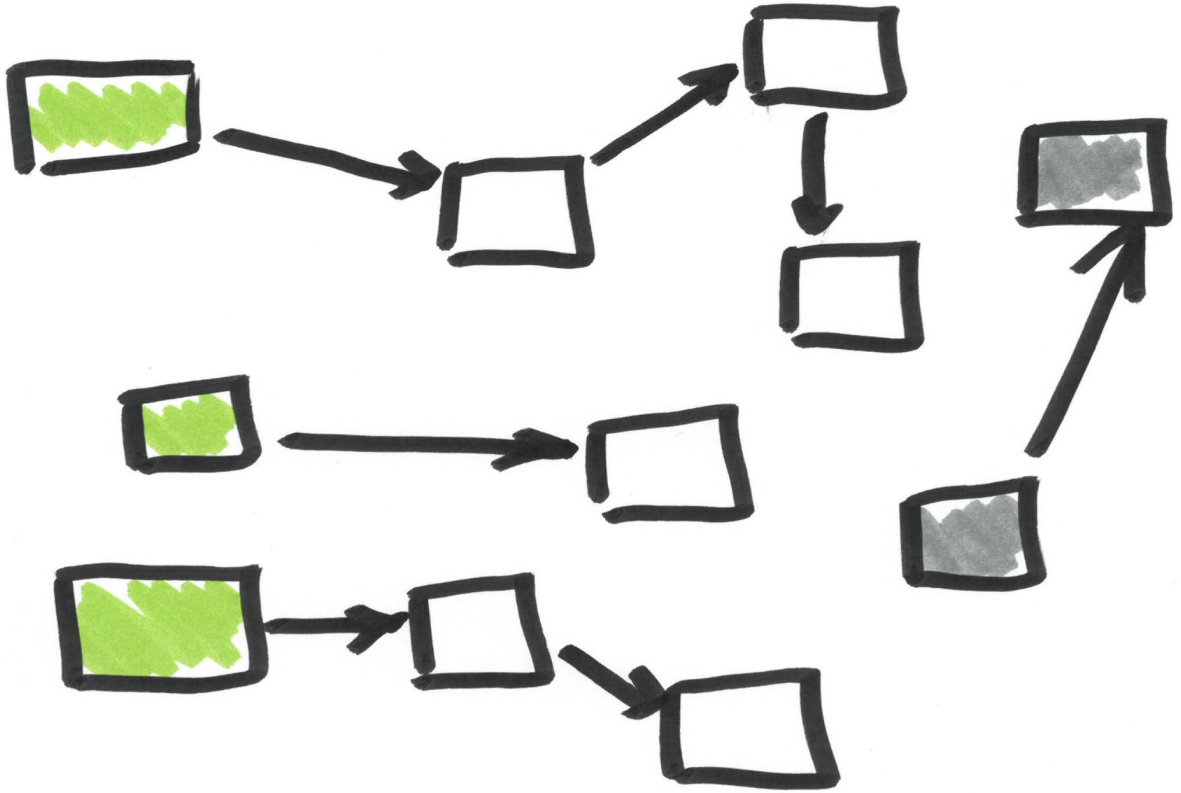
Zarządzanie Pamięcią

JVM - podział pamięci



Źródło: Jinkubator #40 - JVM Internals - Jakub Kubryński

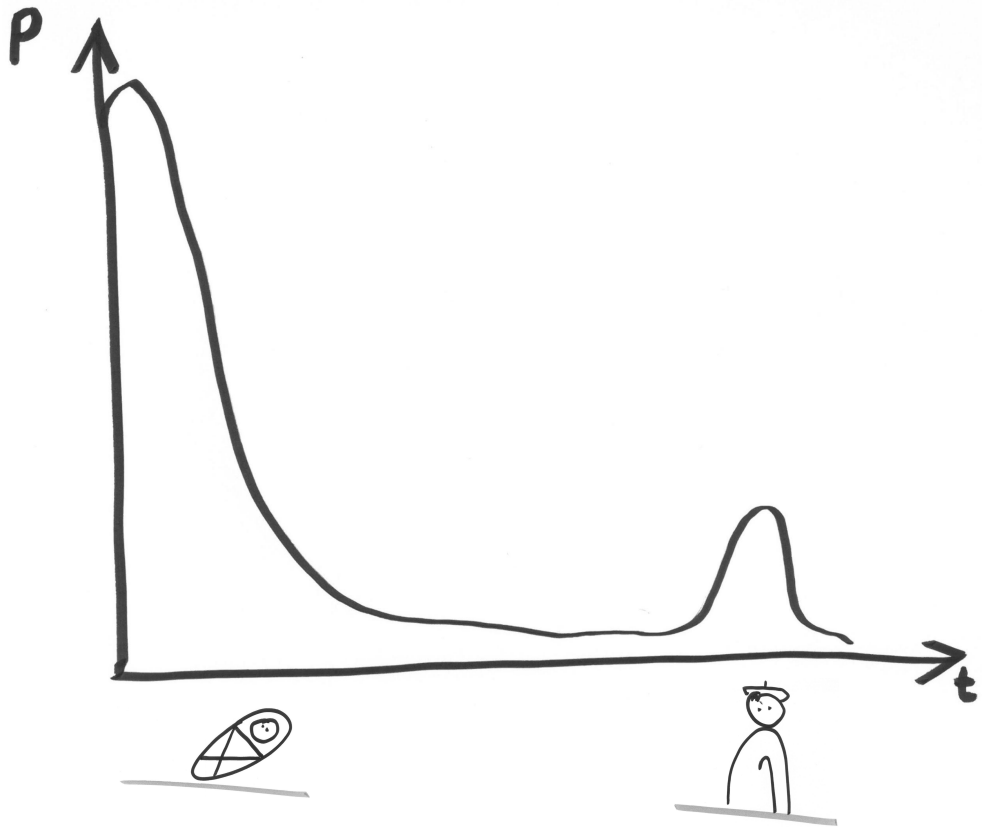
Algorytmy GC



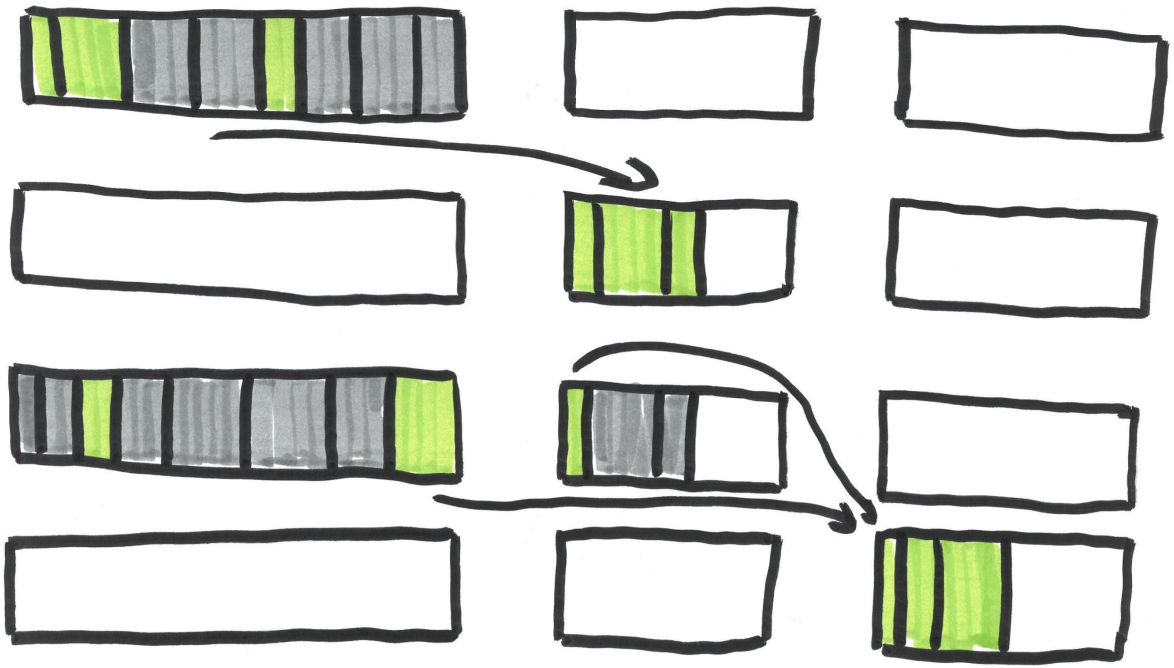
Algorytm GC

```
markGCRoots()
forEach(root) {
        markAllReferences()
}
forEach(object) {
        if (isReachable) {//marked as reachable
                unmark()
        } else { //not marked
                remove()
        }
}
```

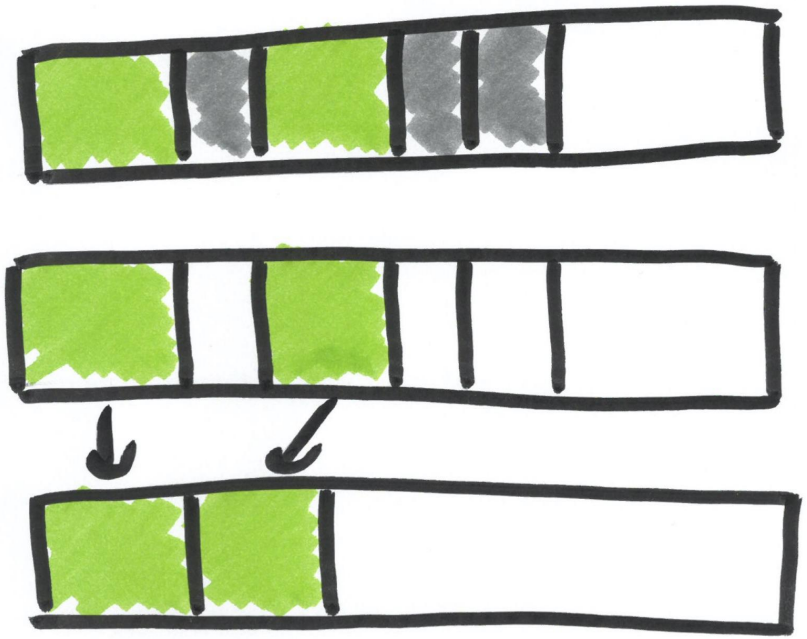
Hipoteza Generacyjności



Algorytm kopiujący

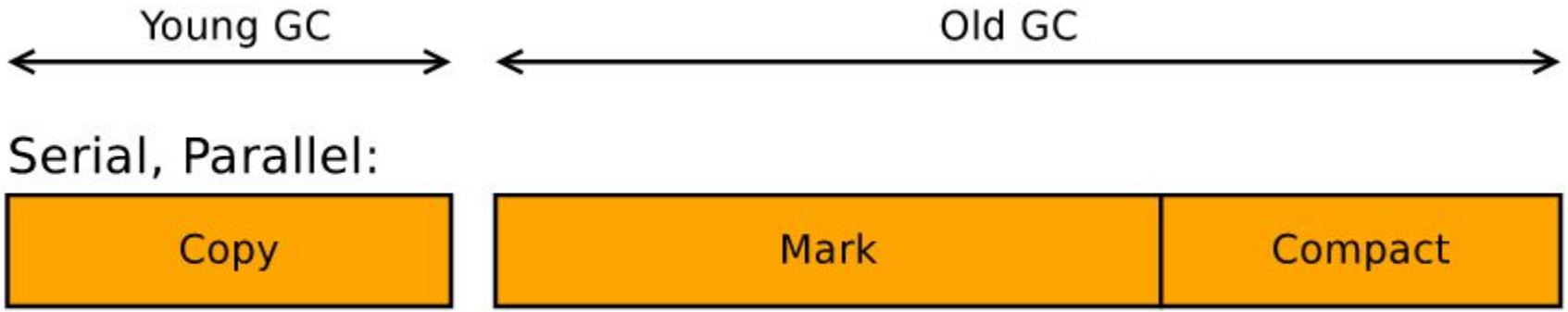


Algorytm Mark-Sweep-Compact



Parallel

- Optymalizuje przepustowość
- Minor: wielowątkowy algorytm kopiujący
- Major: wielowątkowy algorytm mark-sweep-compact

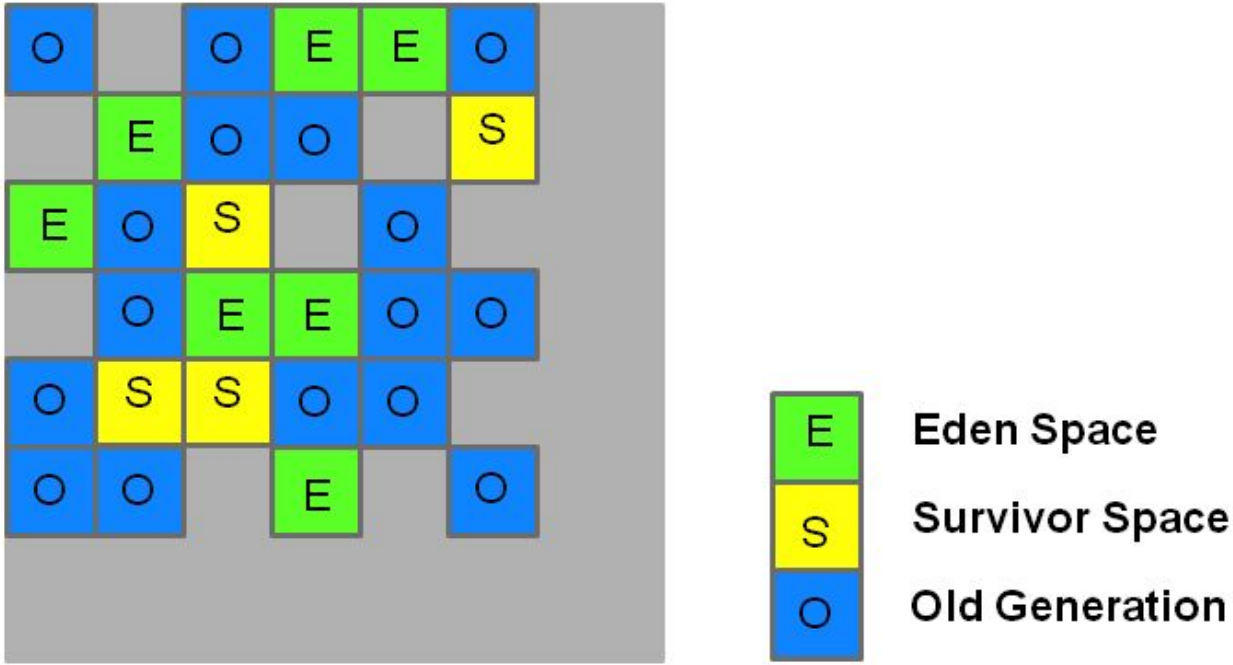


Źródło: https://shipilev.net/talks/jugbb-Sep2019-shenandoah.pdf

G1GC - Garbage First Garbage Collector

- Generacyjny
- Dzieli Stertę na regiony
- Domyślny Algorytm od java 9
- Obsługuje sterty > 4 GB
- Algorytm niskiej latencji
- "Prosty" w dostrajaniu

G1GC - Garbage First Garbage Collector



Źródło: oracle.com

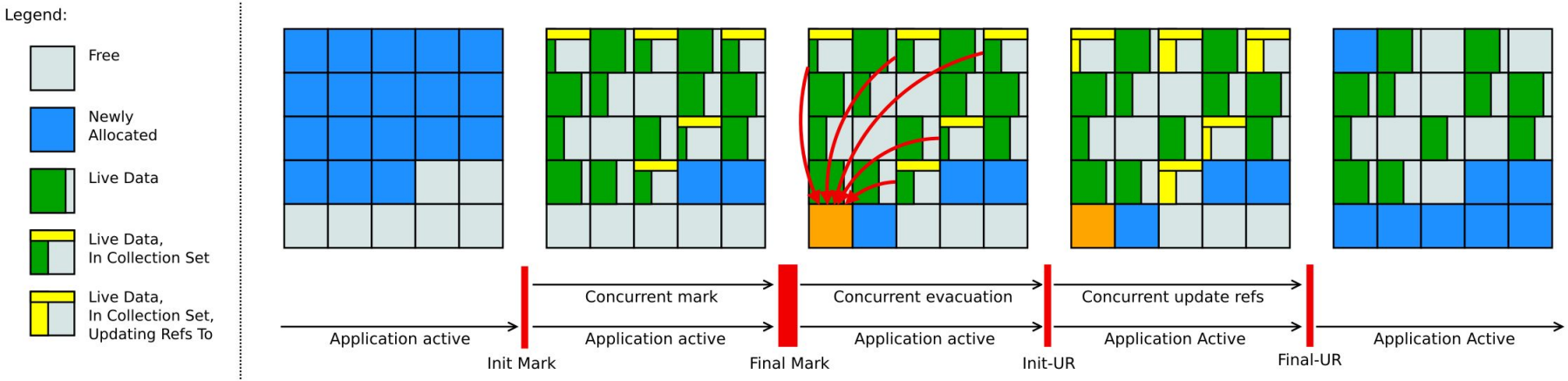
G1GC - Garbage First Garbage Collector

Źródło: https://shipilev.net/talks/jugbb-Sep2019-shenandoah.pdf

Shenandoah GC

- Niegeneracyjny
- Stos podzielony na regiony
- Bazuje na G1GC z współbieżnym etapem kompaktowania
- Pauzy na poziomie 10 ms

Shenandoah GC



Źródło: https://wiki.openjdk.java.net/display/shenandoah/Main

Shenandoah GC

Źródło: https://shipilev.net/talks/jugbb-Sep2019-shenandoah.pdf

Algorytmy GC - podsumowanie

- -Xmx to nie wszystko
- typy prymitywne vs Obiekty
- dobieranie algorytmów do potrzeb (nie istnieje uniwersalny algorytm dla wszystkich przypadków)
- podstawa to wypisywanie logów gc (-Xlog:gc*) i narzędzia do analizy logów (Censum, GCEasy): https://allegro.tech/2018/05/a-comedy-of-errors-debugging-java-memory-leaks.html