Uniwersytet Wrocławski

Rafał Lewandków

PROGRAMOWANIE URZĄDZEŃ MOBILNYCH 1

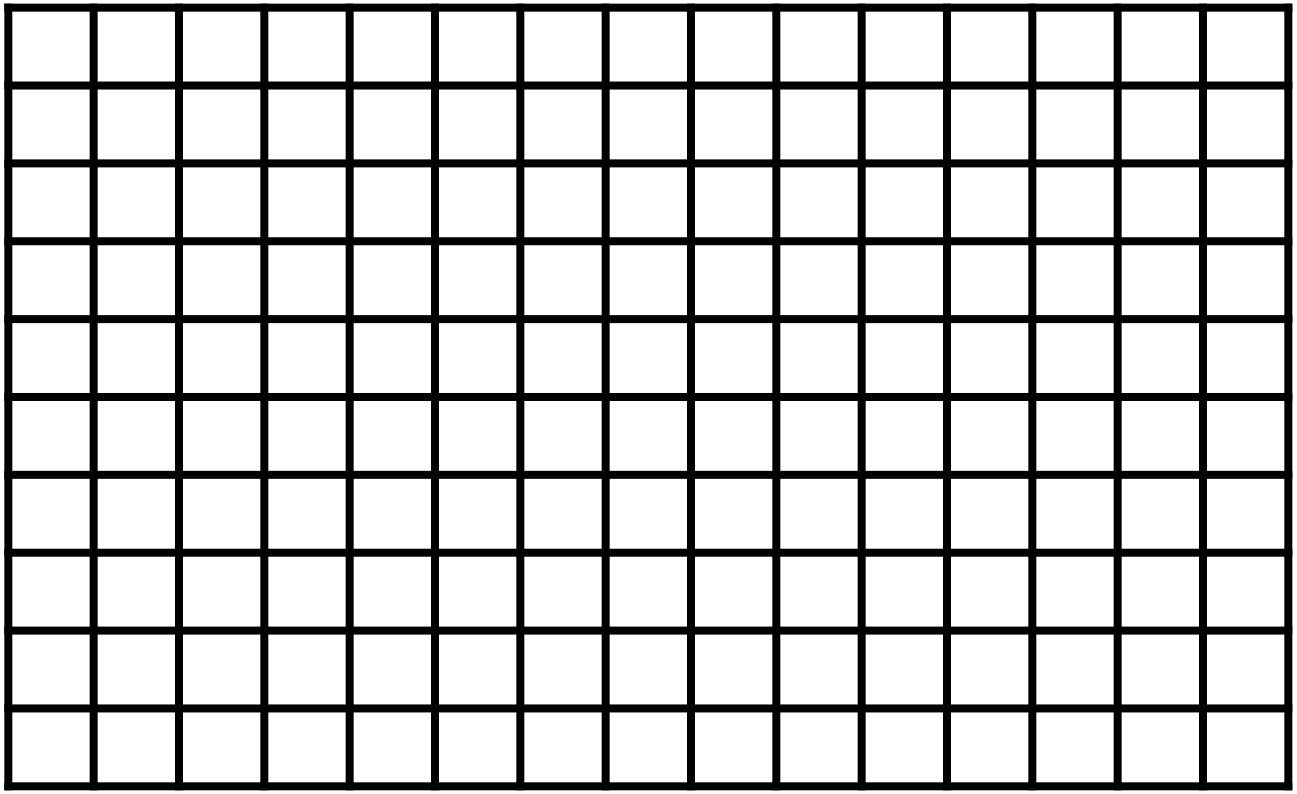
WYKŁAD 3

- Garbage Collection

Typy danych

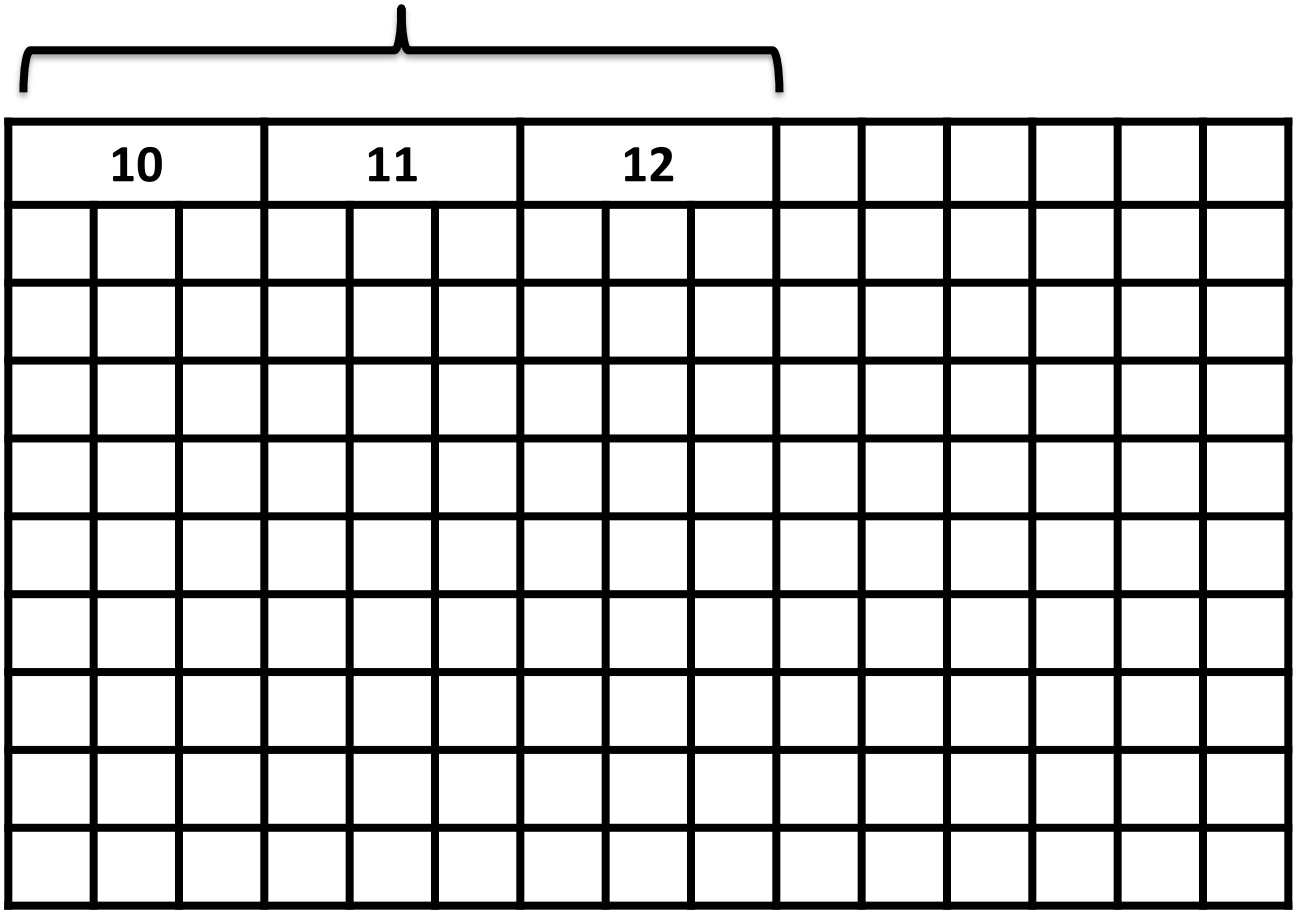
- Bool - 1 byte
- Char - 1 byte
- Double - 8 bytes
- Float - 4 bytes
- Int - 4 bytes
- Long - 8 bytes

Pamięć



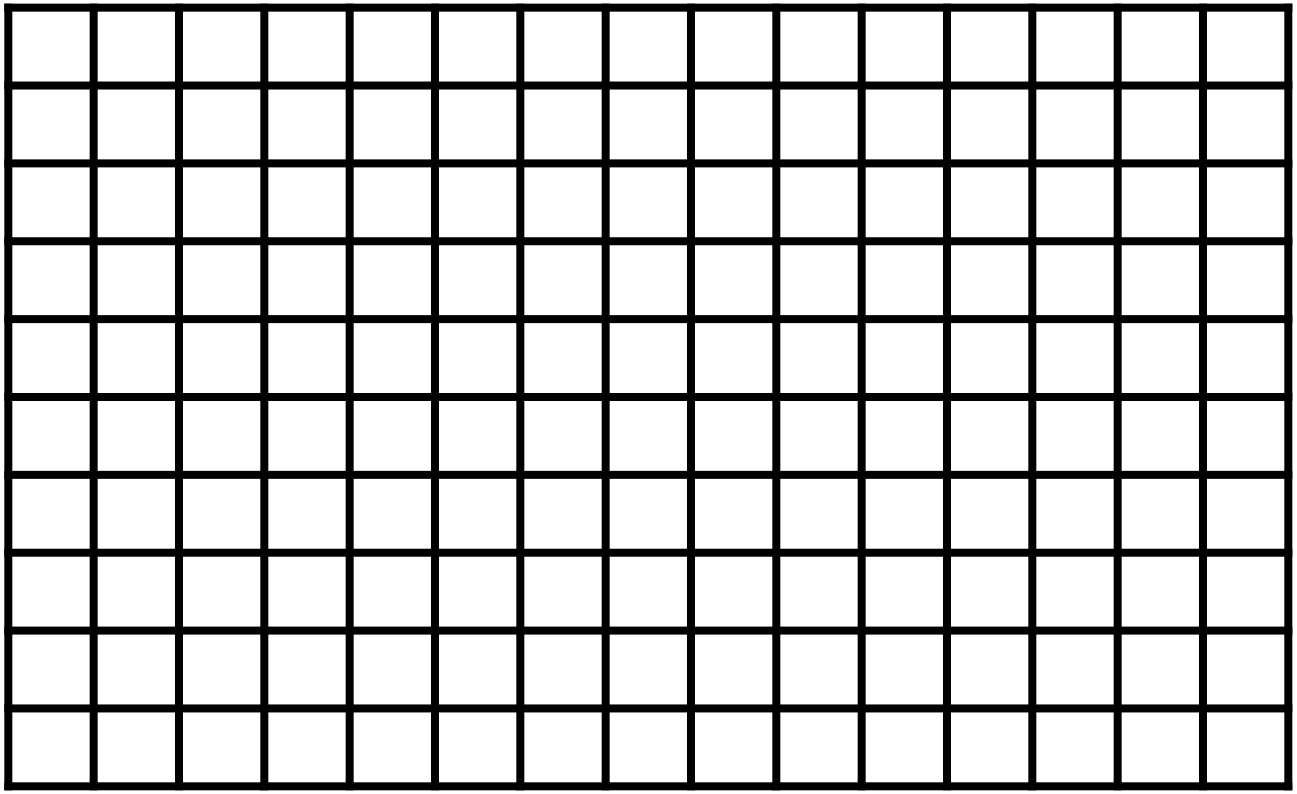
Pamięć

```
int scores[3] = {10, 11, 12};
```



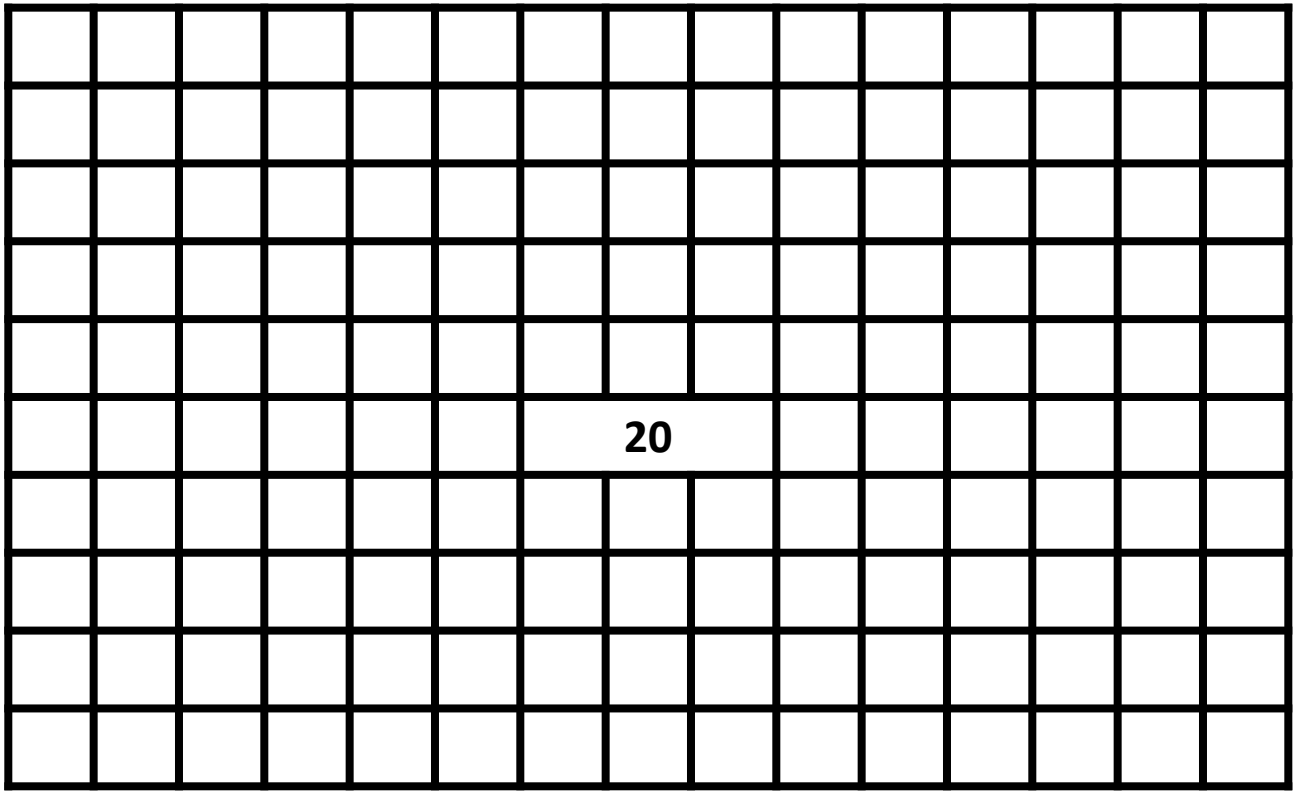
Pamięć

```
int n = 20;
```



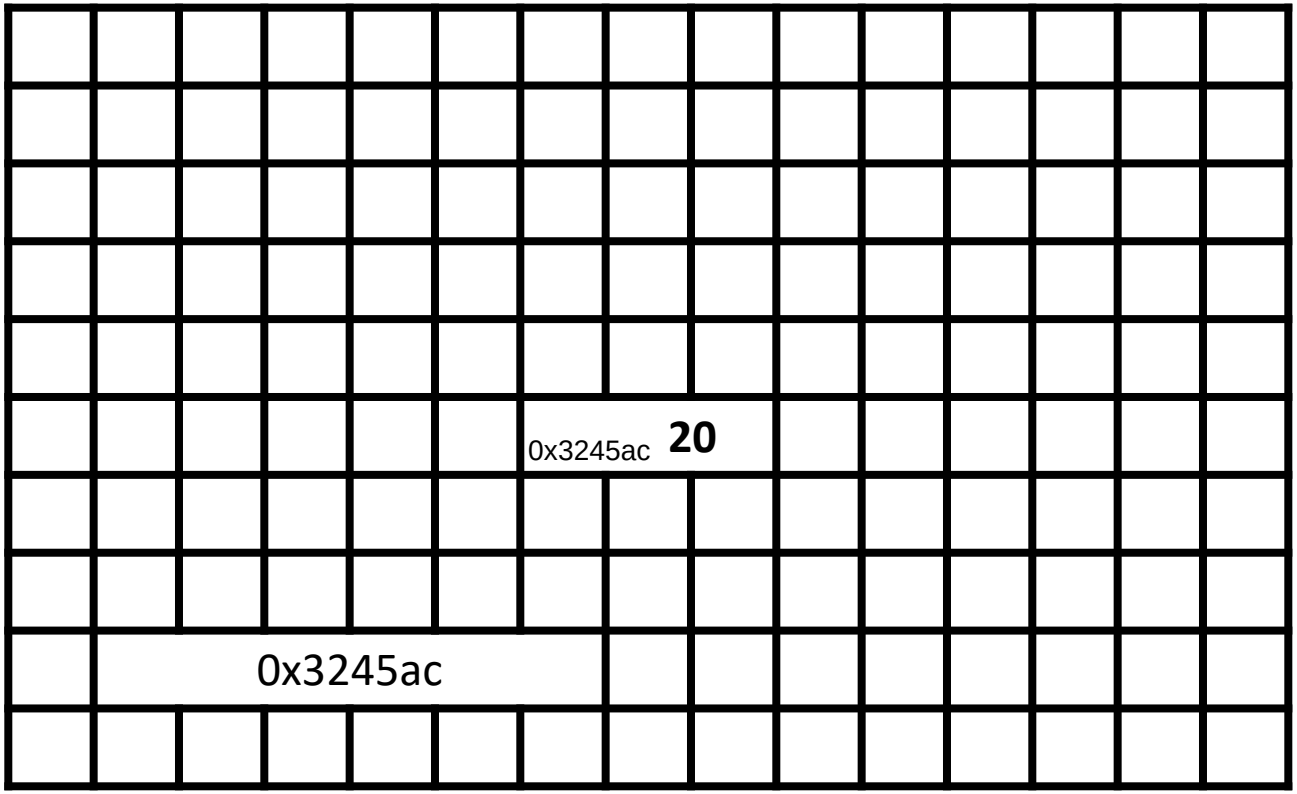
Pamięć

```
int n = 20;
```



Wskaźnik

```
int n = 20;
```

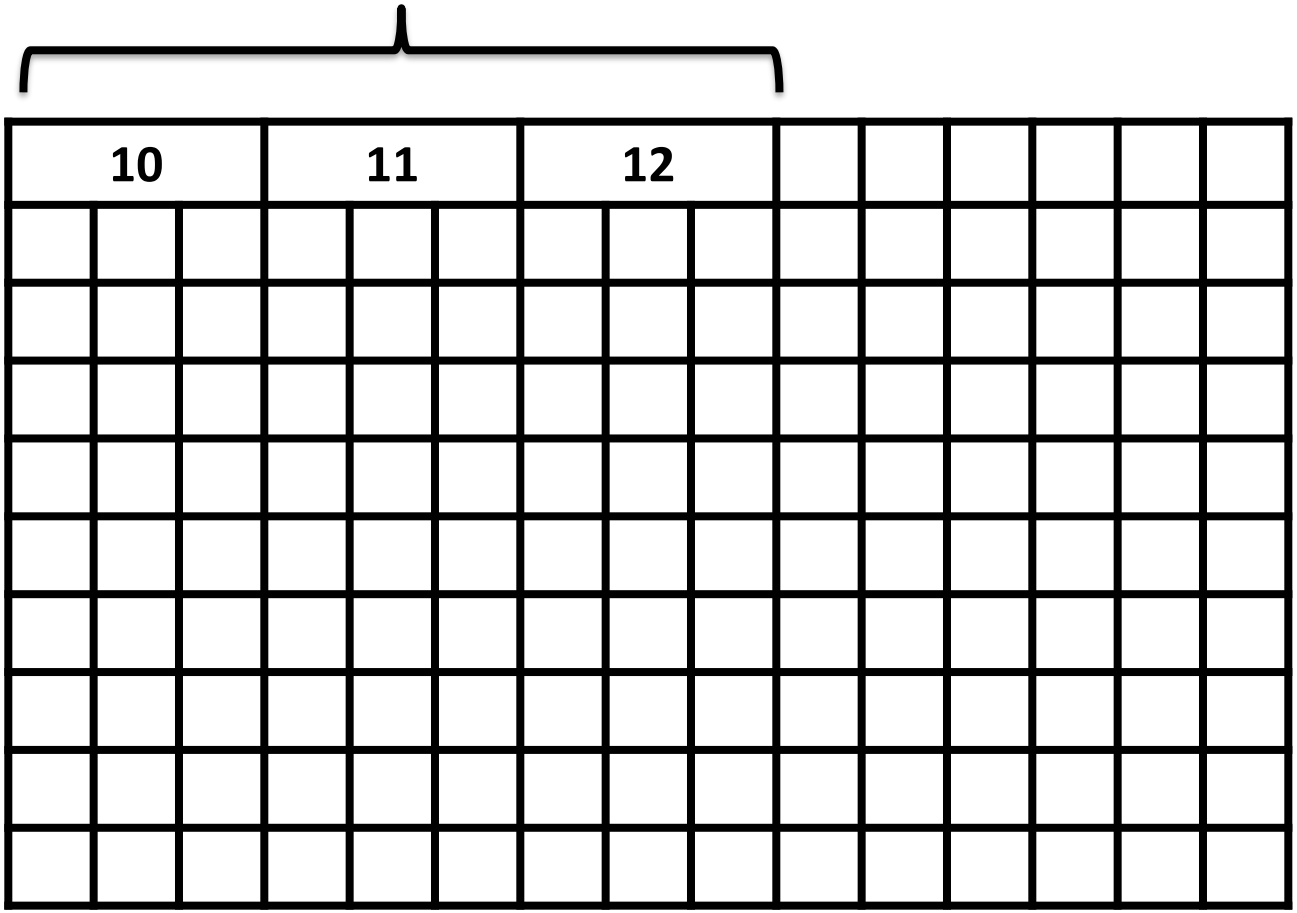


malloc()

```
int n [2] = {2, 3};
int *p = malloc(2 * sizeof(int));

 if(p == NULL) {
      exit(0);
 }

 for(int i = 0; i < n.length; ++i) {
    p [ i ] = n [ i ]
 }

free(p)
```

Pamięć

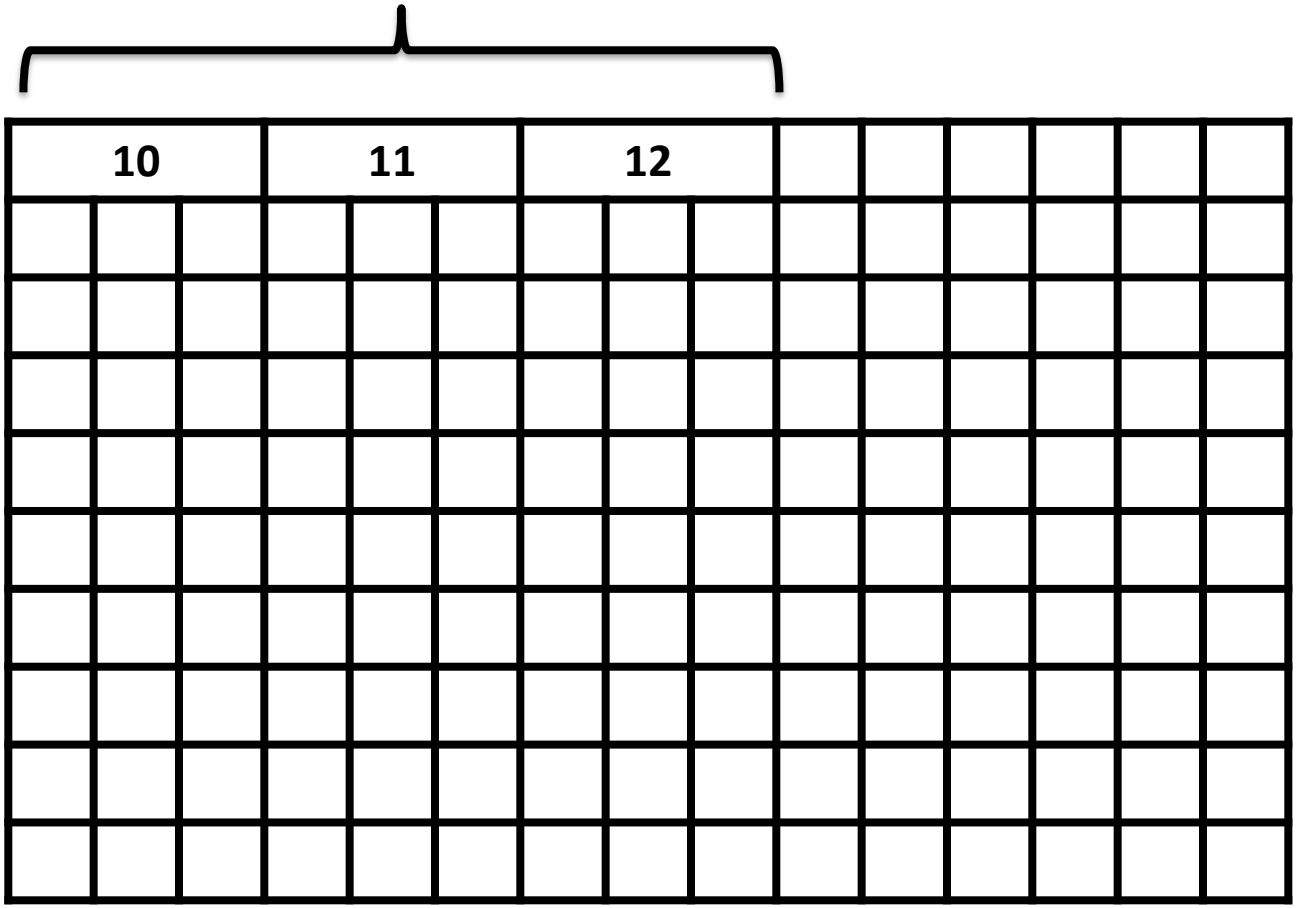
```
int scores[3] = {10, 11, 12};
```



Pamięć

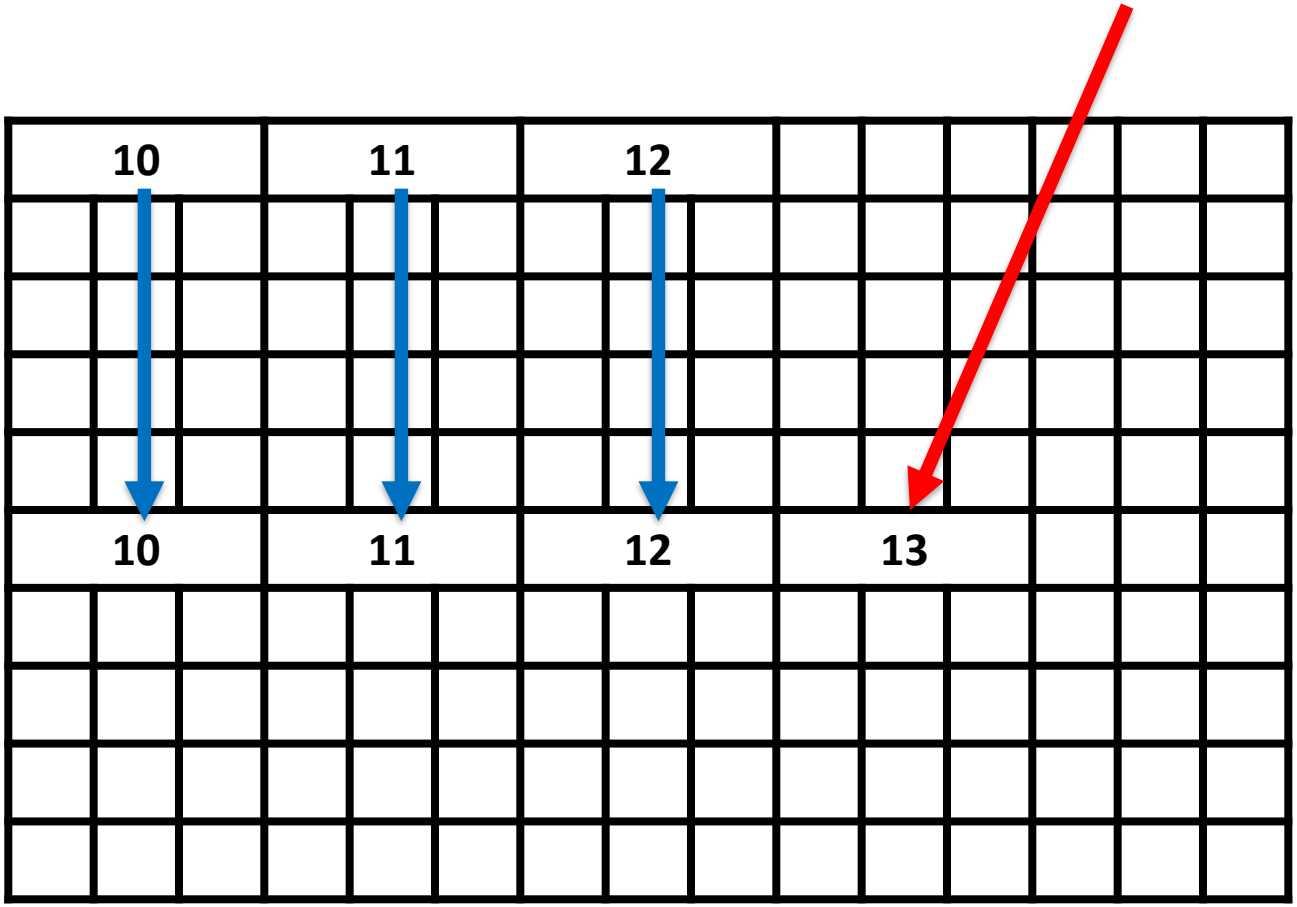
13

```
int scores[3] = {10, 11, 12};
```



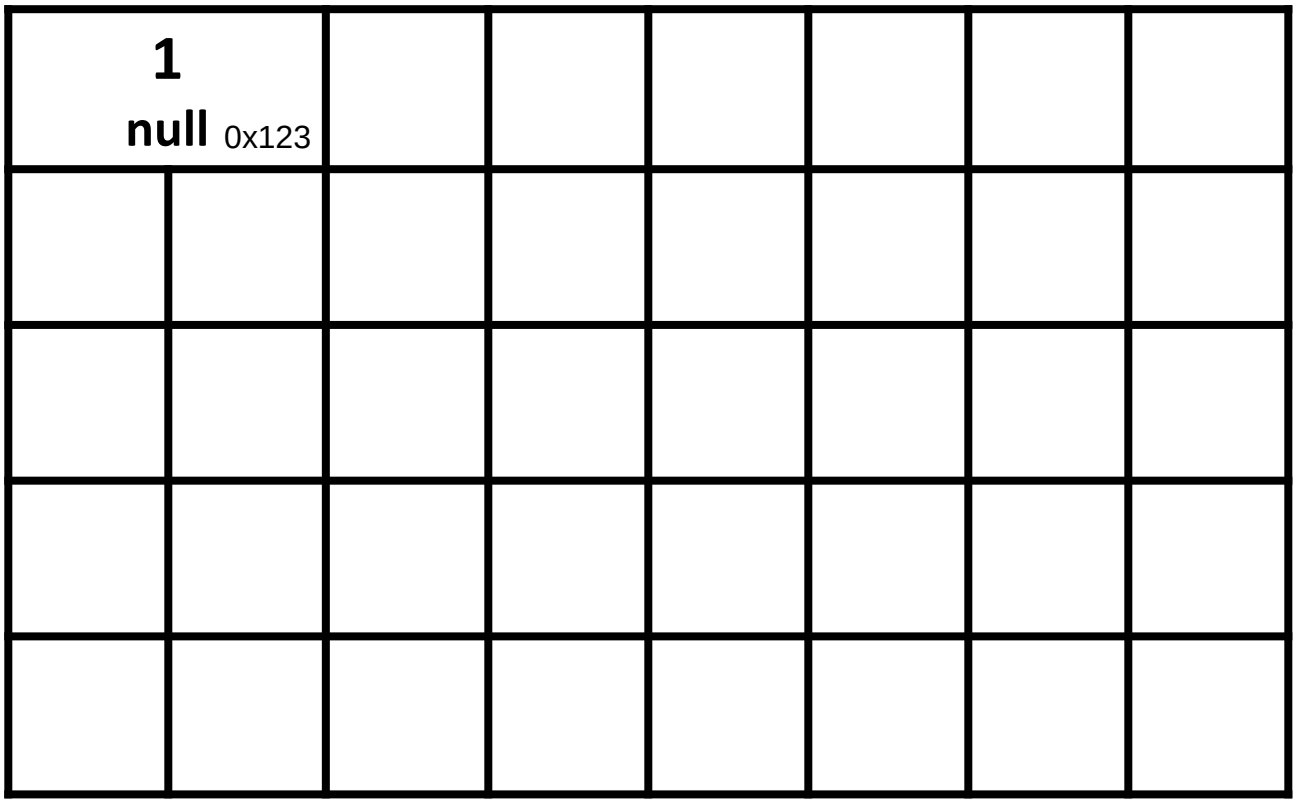
ArrayList

```
int scores[3] = {10, 11, 12};
int cscores[4] = {10, 11, 12, 13};
```



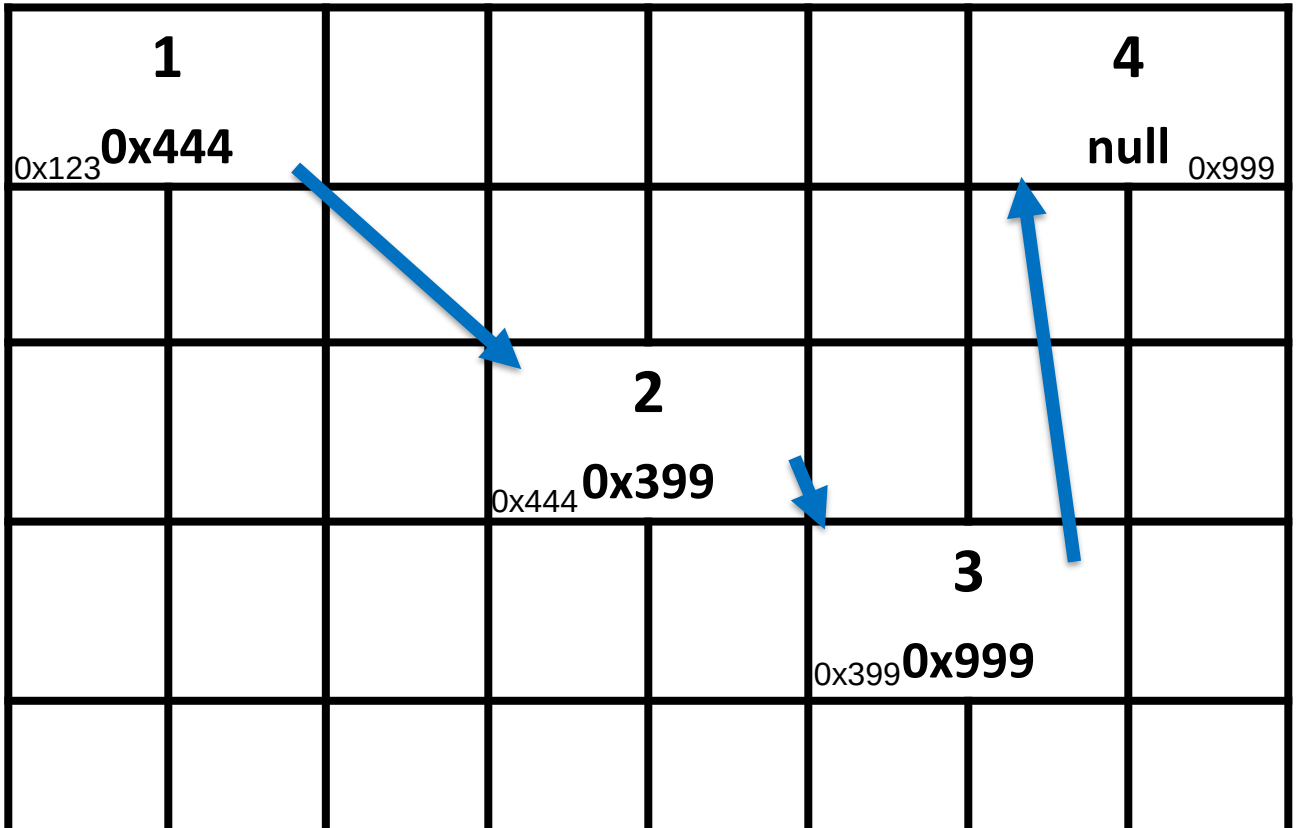
LinkedList

```
typedef struct node {
    int number;
    struct node *next;
}
node;
```



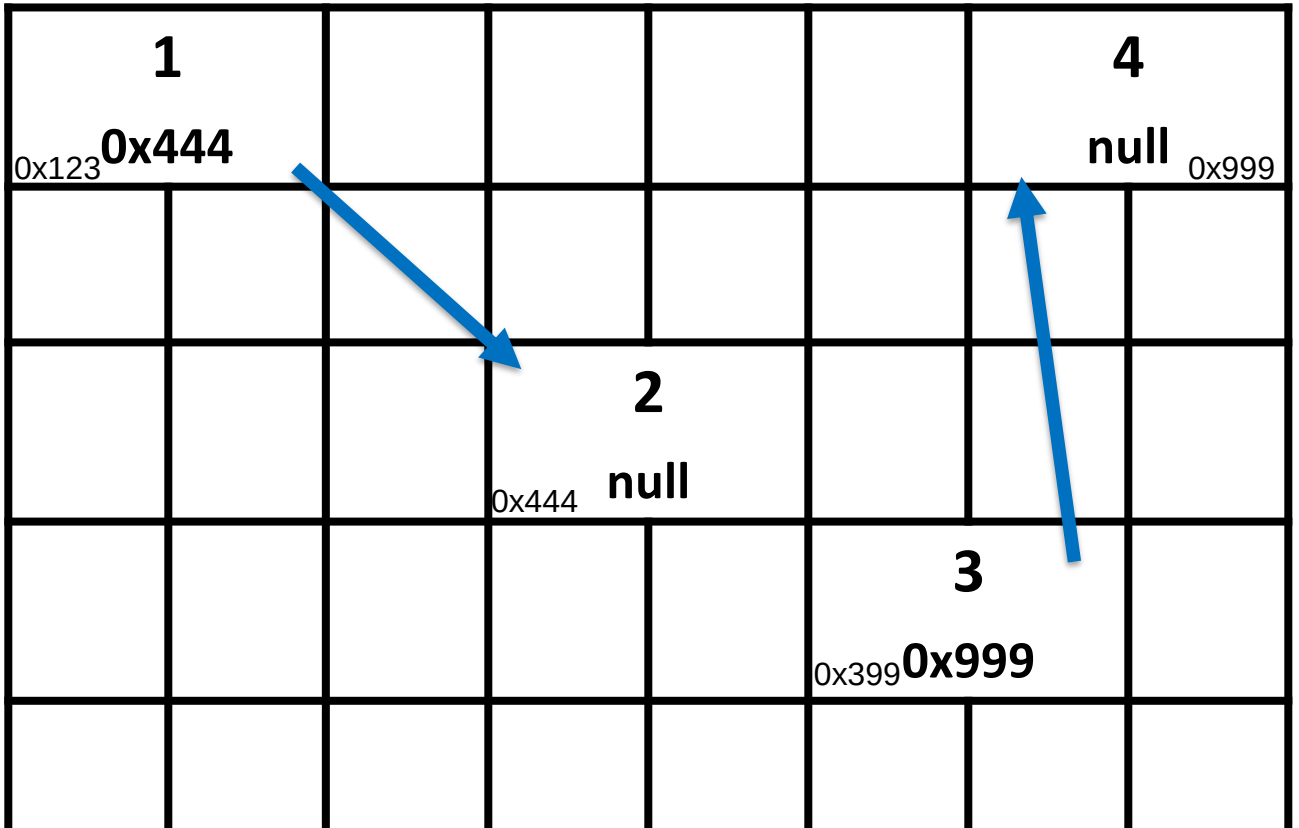
LinkedList

```
typedef struct node {
    int number;
    struct node *next;
}
node;
```



LinkedList

```
typedef struct node {
    int number;
    struct node *next;
}
node;
```



Garbage Collection

W językach takich jak C/C++ musimy zarządzać pamięcią - zaalokować oraz zwolnić.

Malloc()

Realloc()

Calloc()

free(n)

destructors

Garbage Collection

Java wprowadziła automatyczne zarządzanie pamięcią – **Garbage Collector.**

Usuwa obiekty które już nie są używane.

- **Live objects –** obiekty osiągalne (do którego odwołuje się inny obiekt)
- **Dead objects –** obiekty nieosiągalne (do którego nie odwołuje się żaden inny obiekt)

Zbieranie nieużytków jest realizowane przez **demon** (wątek działający niezależnie od użytkownika) **– Garbage Collector**

Garbage Collection

Garbage Collector wykorzystuje specjalne obiekty **– GC Root.** Są to punkty startowe dla procesu zbierania nieużytków.

Rodzaje **GC Root:**

- **Klasy —** wszystkie obiekty przechowywane w statycznych polach klas załadowanych przez główny loader są traktowane jako GC Root.
- **Lokalne zmienne**
- **Aktywne wątki —** każdy aktywny wątek w Javie jest GC Rootem. Obiekty, które wątki wykorzystują bezpośrednio, są traktowane jako osiągalne.
- **JNI (*Java Native Interface*) —** referencje utworzone przez natywny kod (np. w C/C++) i przechowywane w tzw. *JNI Handles* są GC Rootami.

Garbage Collection

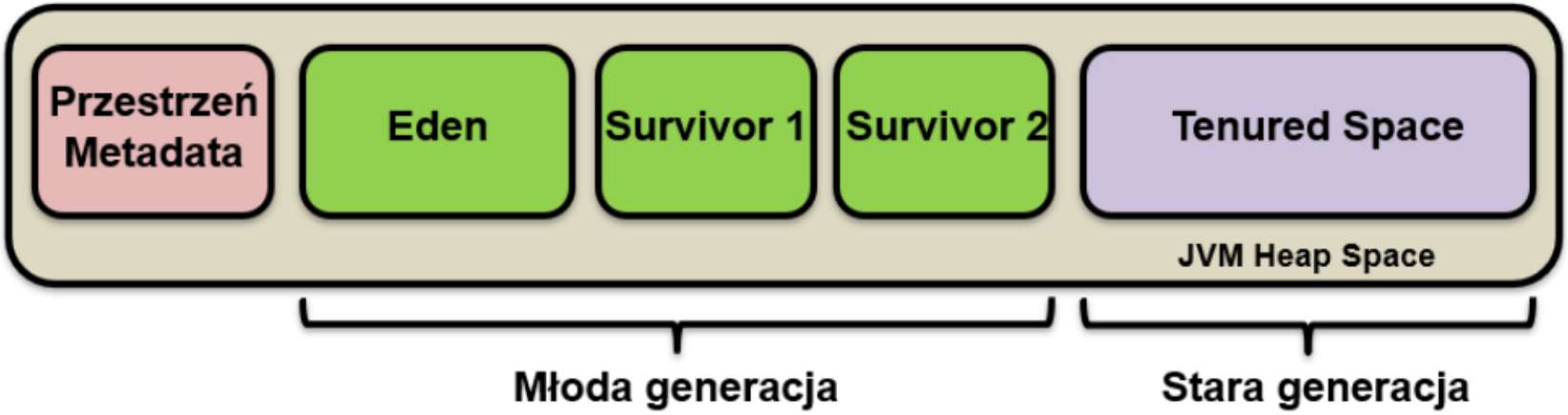
Kroki gc:

1. **Mark —** oznaczanie obiektów żywych — GC przeszukuje graf obiektów, rozpoczynając od GC Root.

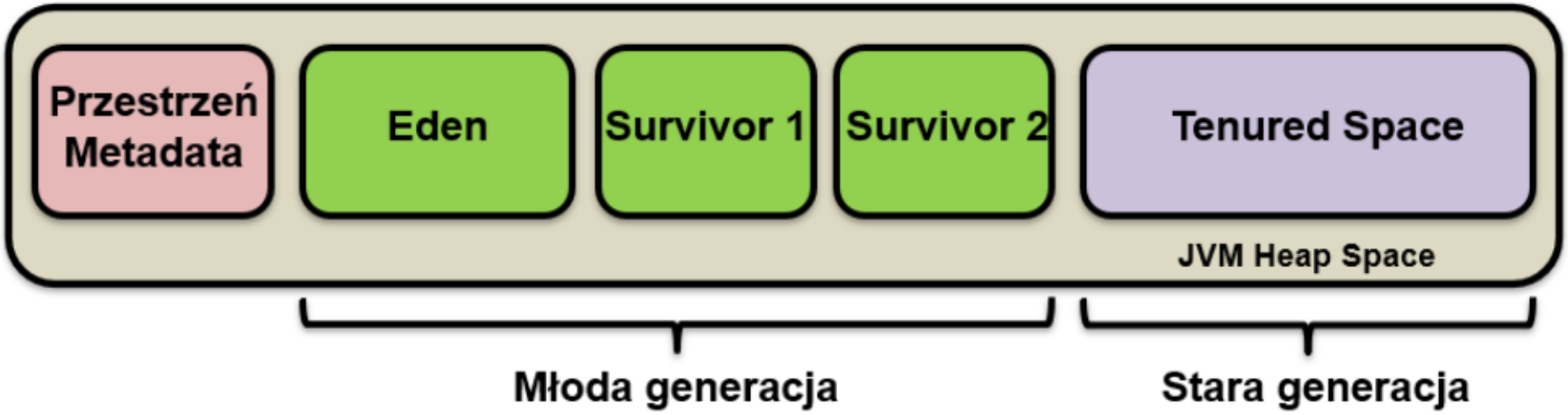
2. **Sweep —** po zakończeniu etapu oznaczania, GC przechodzi przez pamięć i usuwa wszystkie obiekty, które nie zostały oznaczone jako żywe.

3. **Compacting —** Aby zapobiec fragmentacji pamięci, GC przesuwa żywe obiekty w pamięci, aby wypełnić wolne przestrzenie.

Generational Collectors



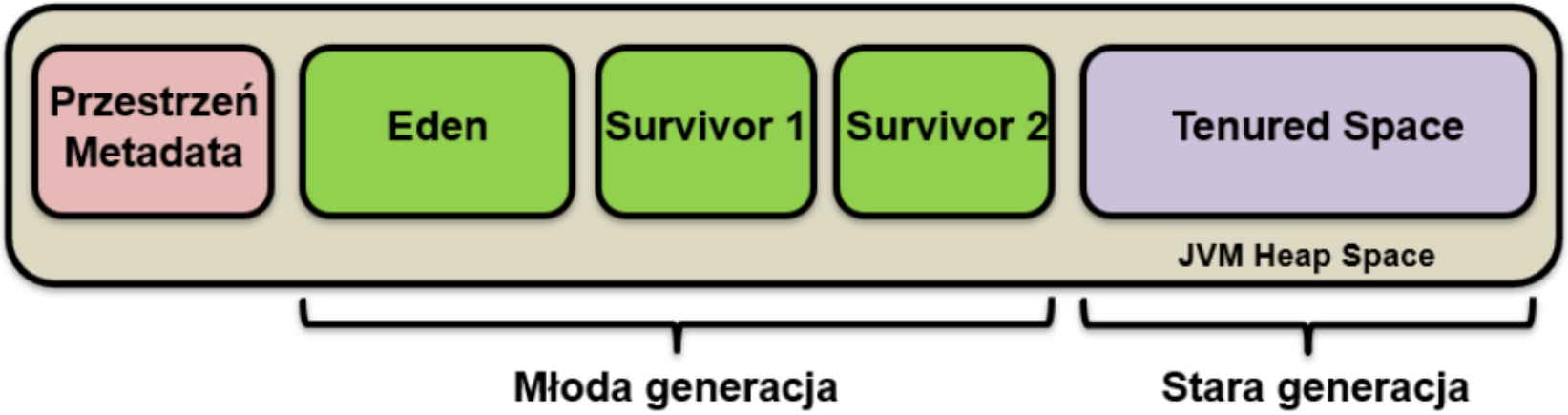
Generational Collectors



Zasada działania:

- **Young Generation –** nowe obiekty dopiero utworzone – dzieli się na trzy strefy
  - **Eden**
  - **Survivor space 1**
  - **Survivor space 2**

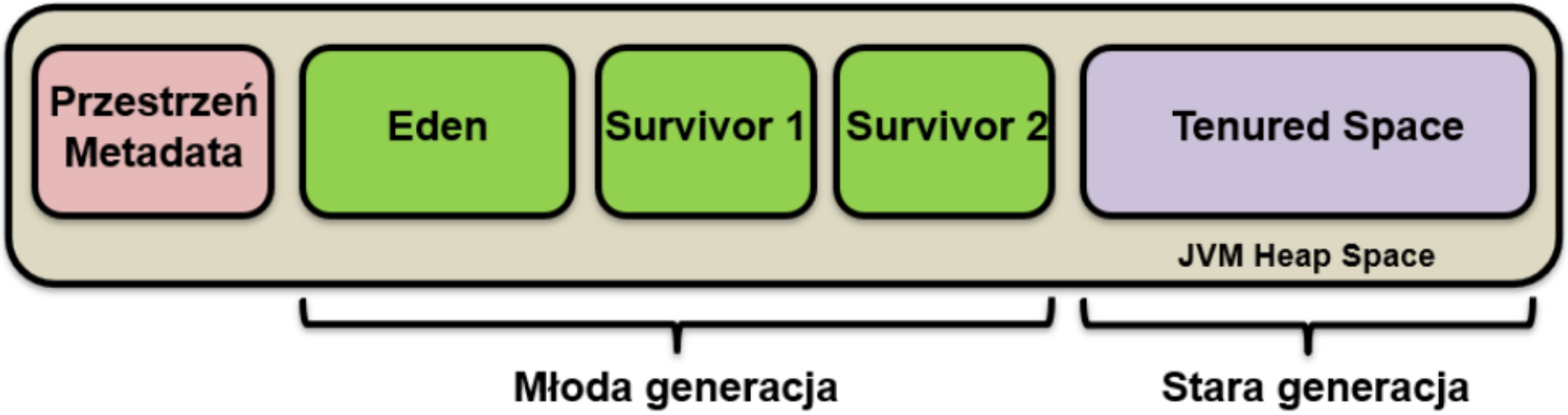
Generational Collectors



Zasada działania:

- **Wraz z zapełnianiem dostępnej pamięci edenu gc rozpoczyna działanie i wykonuje marking**
- **Przenosi żywe obiekty do z edenu do s1**
- **Usuwa nieosiągalne**

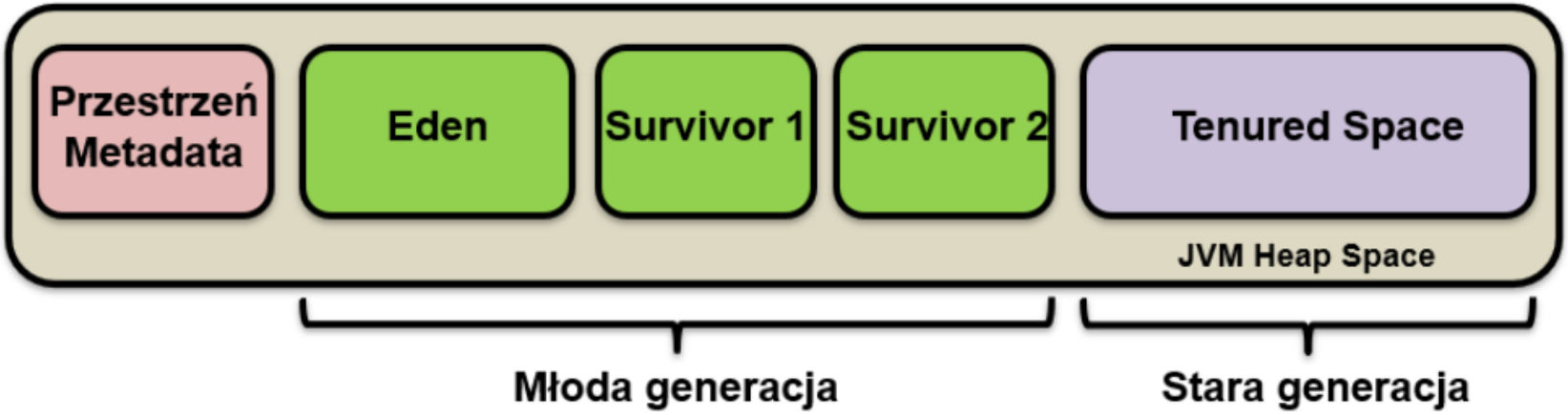
Generational Collectors



Zasada działania:

- **W kolejnej iteracji gc wykonuje marking s1 i żywe obiekty przenosi do s2, następnie marking edenu i żywe obiekty przenosi do s1**
- **W każdej kolejnej iteracji role s1 i s2 są odwracane**

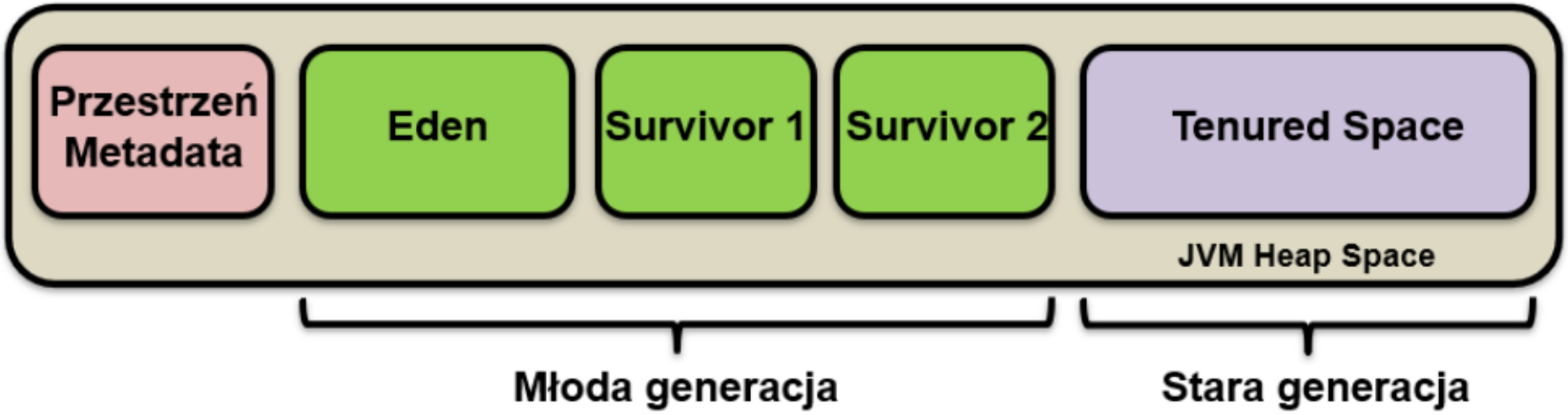
Generational Collectors



Zasada działania:

- **Eden jest czyszczony za każdym razem**
- **Po określonej liczbie iteracji s1 -> s2 -> s1 … obiekty są promowane do old generation**
- **Iteracje s1 -> s2 -> s1 … zapobiegają fragmentacji**

Generational Collectors



Zasada działania:

- **Procesy odpowiadające za czyszczenie:**
  - **Młoda generacja – Minor GC**
  - **Stara generacja – Major GC**