Podstawy Informatyki –
OOP
(Object Oriented Programming)

21.11.2019

dr inż. Michał Rad

Plan wykładu:

1. Powtórka
2. Programowanie Obiektowe – wyższy poziom abstrakcji
3. Kwalifikatory dostępu
4. Konstruktory
5. Destruktory
6. Konstruktor kopiujacy, operator przypisania

Powtórka

```
struct Wektor
{
  float x,y,z;
  Wektor operator+(Wektor &v2);
  void wypisz();
  Wektor();  //konstruktor bez parametrów;
}; // ……//
int main()
{
Wektor A(1,2,3),B(0,-1,2),C;
A.wypisz();
B.wypisz();
C=A+B;
C.wypisz();
}
```

Programowanie obiektowe

- Wyższy poziom abstrakcji
- Kod programu łatwiejszy w interpretacji dla człowieka
- Łatwość rozbudowy aplikacji – *kod otwarty na rozszerzenia zamknięty na modyfikacje* (ang. *Open/Closed principle*)
- Łatwość wielokrotnego użycia kodu

Programowanie obiektowe *cd..*

- Hermetyzacja
- Dziedziczenie
- Polimorfizm
- Możliwość stosowania wzorców projektowych

» **Hermetyzacja** – zabezpiecza przed dostępem/modyfikacją/użyciem składowej bądź funkcji w sposób nieprawidłowy (nieuprawniony)

» Są trzy możliwości określenia dostępu:

- **public**
- **private**
- **protected**

Kwalifikatory dostępu:

- **public:**
  - funkcja bądź składowa dostępna jest zarówno z zewnątrz jak i z wewnątrz klasy
- **private:**
  - funkcja bądź składowa dostępna jest jedynie dla innych metod (funkcji) składowych klasy

Kwalifikatory dostępu *c.d*

- **protected:**
  - tak samo jak *private*, ale metody i składowe dostępne są także z poziomu klas wyprowadzonych (potomnych)

Przykład

```
class Kwadrat
{
  public:
   set_bok(float b){bok=b; powierzchnia=b*b;};
  private:
   float powierzchnia;
   float bok;
}; // ……//
int main()
{
Kwadrat A;
A.bok=5;  // <- błąd
A.set_bok(5); //<-OK!
return 1;
}
```

class vs struct

- Jedyna różnica w C++ miedzy obiektem zadeklarowanym ze słowem **struct** a słowem **class** jest w domyślnym poziome dostępu do składowych:
  - w **struct** domyślnie wszystko jest ***public***
  - w **class** domyślnie wszystko jest ***private***

Po co hermetyzacja?

- Daje “bezpieczniejszy” kod
- Nie należy jednak mylić tego pojęcia z “poufnością” składowych czy funkcji

Wracamy do szczegółów…

- Konstruktory: służą do tworzenia obiektów i wykonania operacji pomocniczych
- Inicjalizacja pól składowych obiektu może być realizowana również za pomocą listy inicjalizacyjnej

Przykład listy inicjalizacyjnej

```
Wektor::Wektor(): x(1),y(1),z(1)
{
// nothing to do
};

Wektor::Wektor(float _x,float _y,float _z):
x(_x),y(_y),z(_z)
{
// nothing to do
};
```

Destruktor

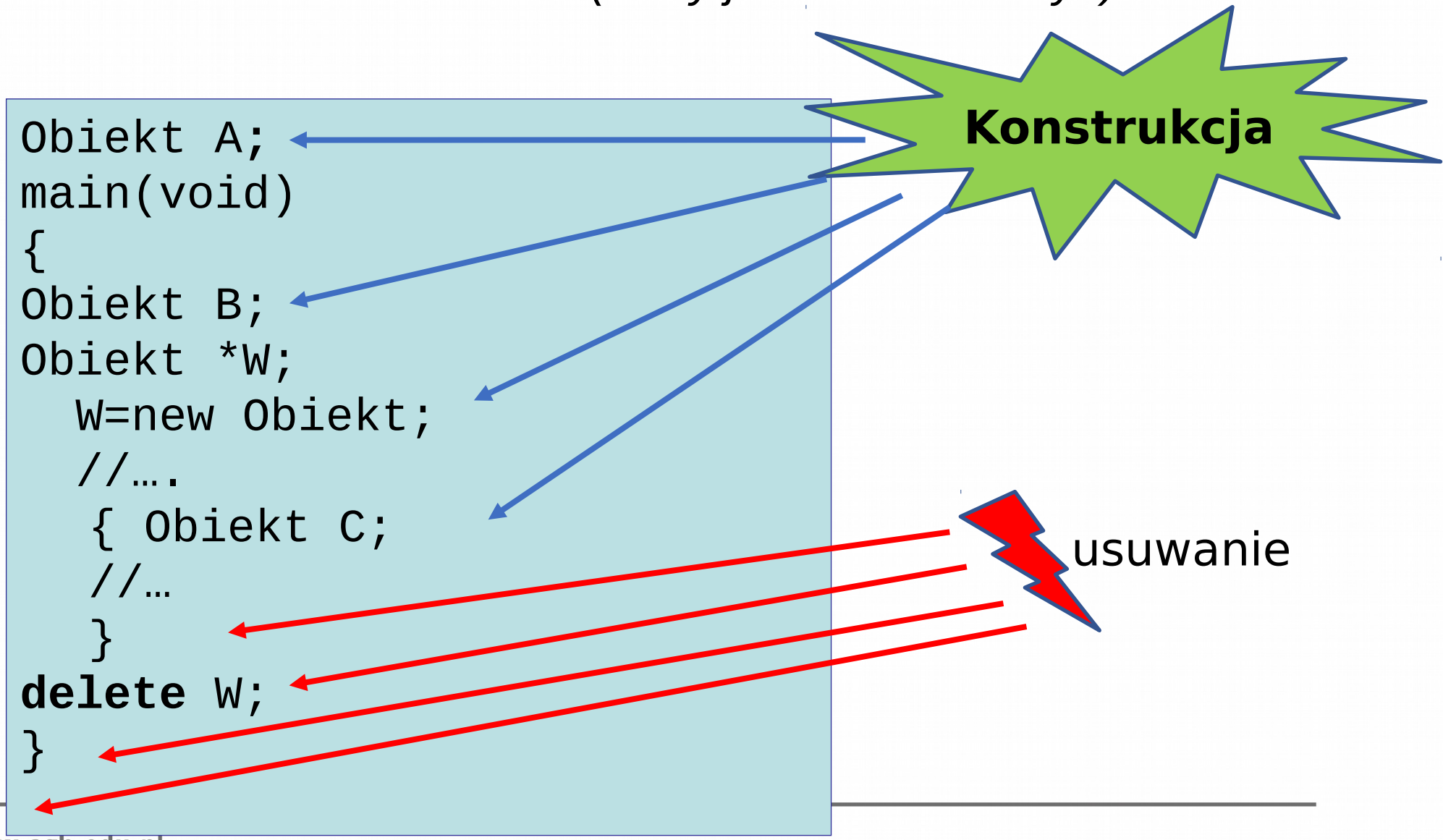
- Odpowiada za prawidłowe usunięcie obiektu i ew. funkcje dodatkowe
- Jest to funkcja uruchamiana podczas usuwania obiektu (przy końcu “życia” obiektu)
- Nie pobiera argumentów (nie może – bo jak?) i nie zwraca wartości

Przykład

- **Destruktor** ma nazwę identyczną z nazwą klasy tylko poprzedzony jest tyldą ~
- Może być tylko jeden
- Jeśli go nie zdefiniujemy kompilator go wygeneruje

```
class Obiekt
{
Obiekt() //konstruktor
~Obiekt() // destruktor
//…
}
```

Destruktor *(kiedy jest uruchamiany?)*




```
Obiekt A;
main(void)
{
Obiekt B;
Obiekt *W;
  W=new Obiekt;
  //….
   { Obiekt C;
   //…
   }
delete W;
}
```

Destruktor

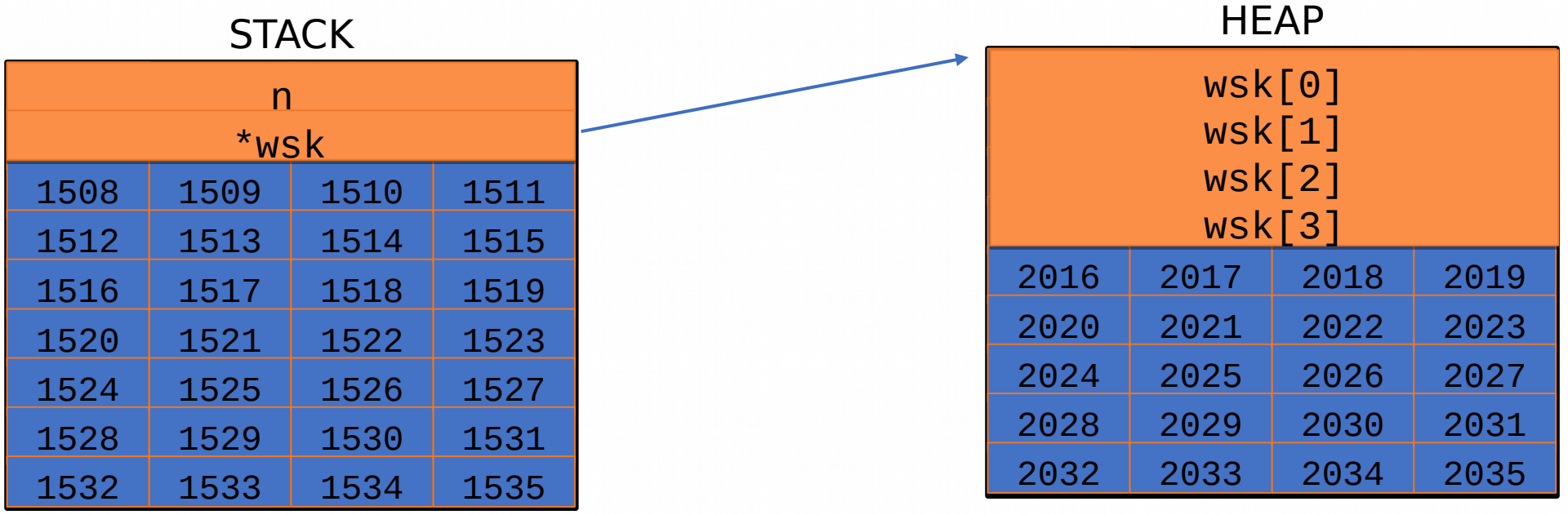
- Musimy go zdefiniować poprawnie sami, kiedy zachodzi potrzeba zwolnienia pamięci alokowanej w czasie tworzenia obiektu.

Przykład

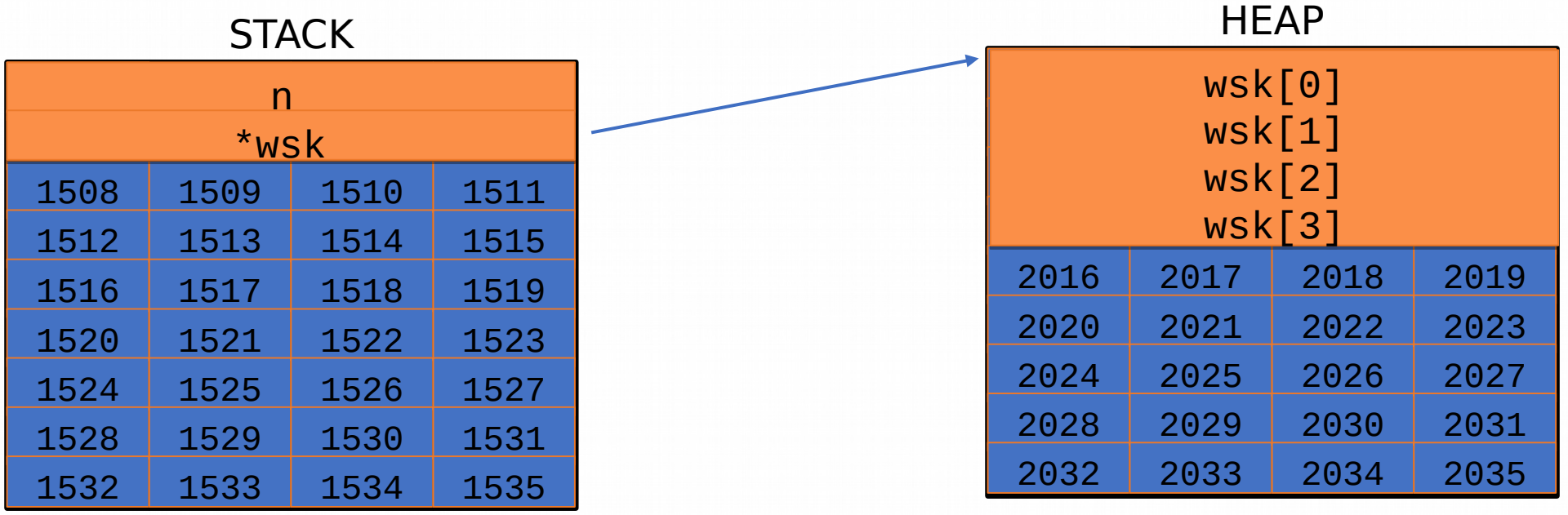
```
class Memory
{
float *wsk;
int ile;
Memory(int n);
~Memory();
}
```

```
Memory::Memory(int n)
{
wsk = new float[n];
}
```

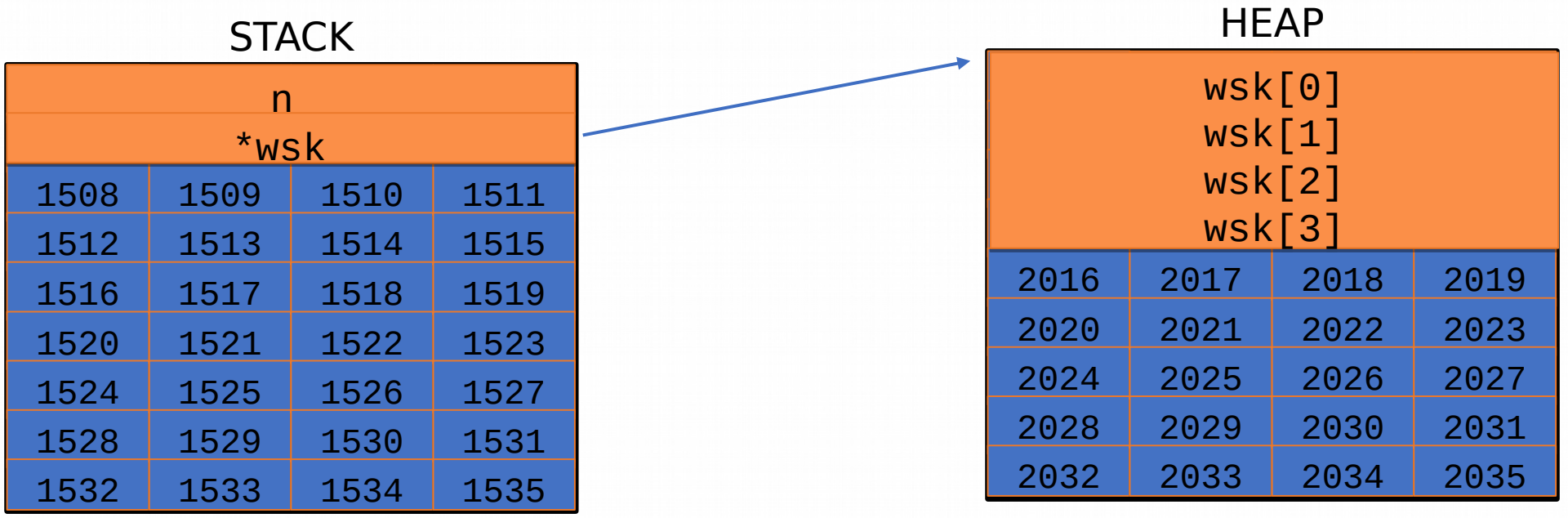
Jak działa konstruktor



Jak działa destruktor domyślny?



A jak powinien działać nasz destruktor?



Konstruktor kopiujący

```
class Memory
{
float *wsk;
int ile;
Memory(int n);
Memory(const Memory&); //konstruktor kopiujący
~Memory();
}
```

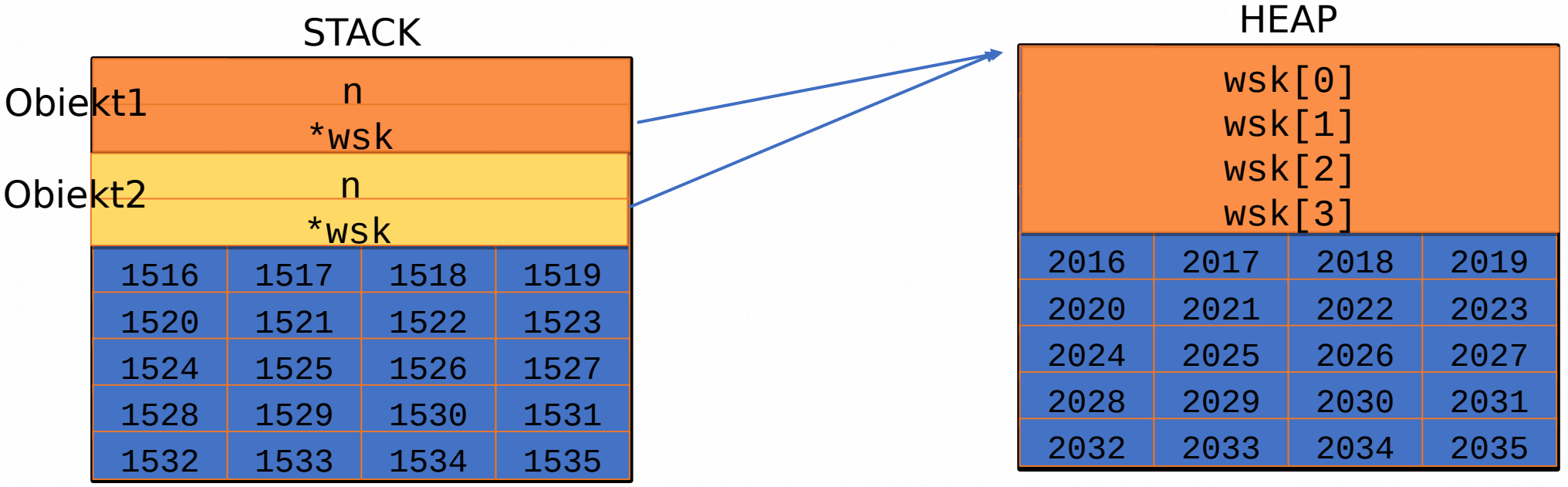
```
main(void)
{
Memory A;
Memory B(A);
funkcja(A);
}
```

Operator przypisania

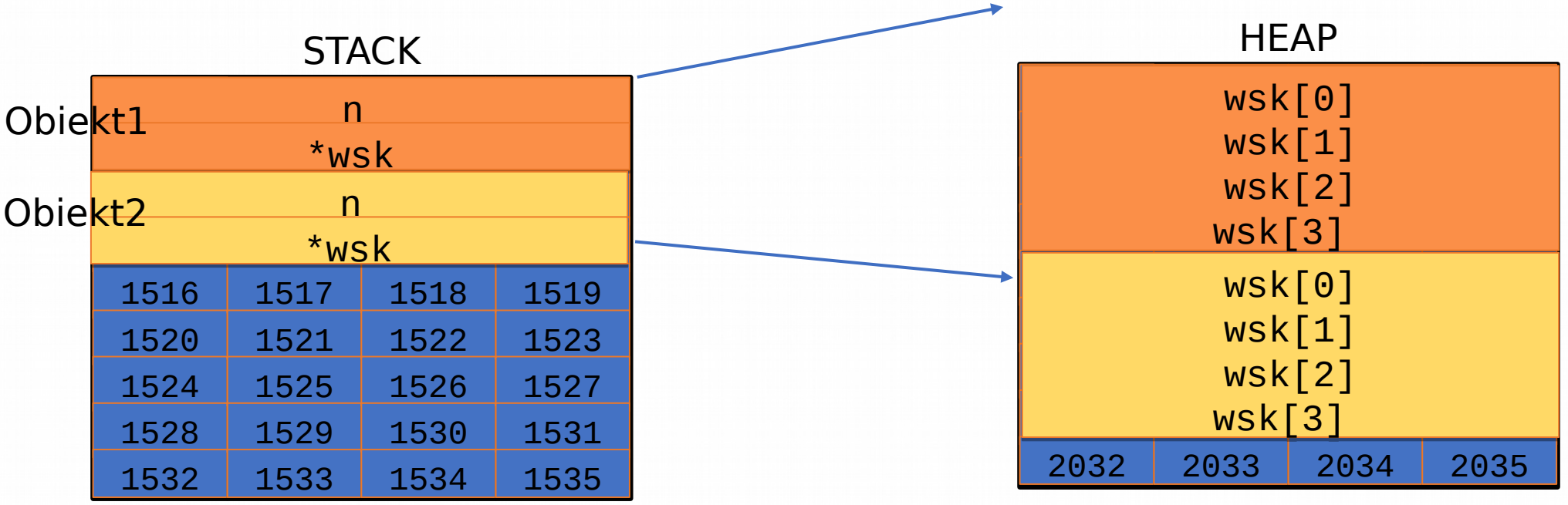
```
class Memory
{
float *wsk;
int ile;
Memory(int n);
Memory();
Memory(const Memory&); //konstruktor kopiujący
Memory& operator=(const Memory&);
          //operator przypisania
~Memory();
}
```

```
main(void)
{
Memory A(5),B;
B=A;
}
```

Jak działa domyślny konstruktor kopiujący?



A jak powinien działać nasz konstruktor kopiujący



Reguła trzech

» Jeśli musimy sami zdefiniować destruktor lub konstruktor kopiujący lub operator przypisania – to musimy również zdefiniować dwie pozostałe funkcje.

Przykład

» Z pliku …

UML

- Podczas projektowania aplikacji obiekty, ich składowe oraz metody wygodniej opisywać symbolicznie
- Wykorzystywany jest do tego pół-formalny język UML (Unified Modeling Language)

Przykład opisu klasy w UML