

Informatyka 2 (EZ1E3012)

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr III, studia niestacjonarne I stopnia
Rok akademicki 2020/2021

Pracownia nr 4 (07.11.2020)

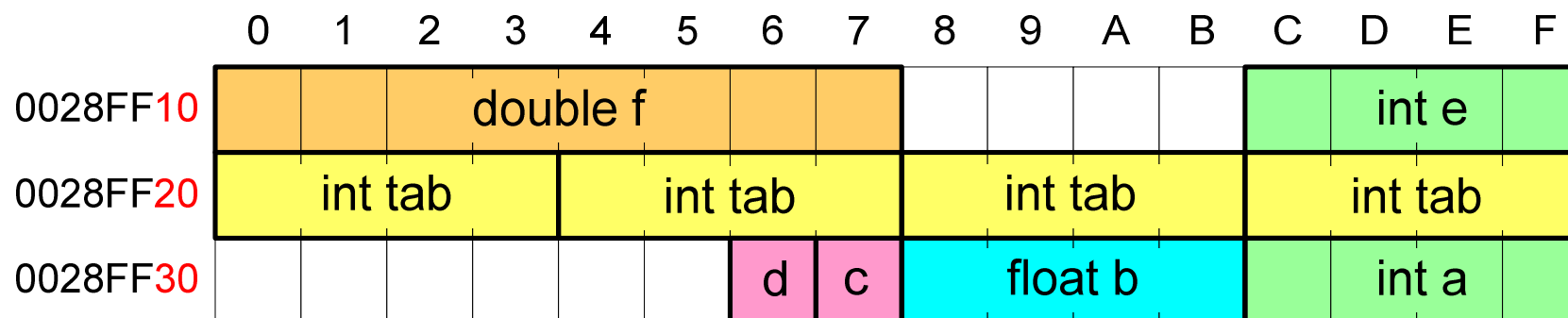
dr inż. Jarosław Forenc

Co to jest wskaźnik?

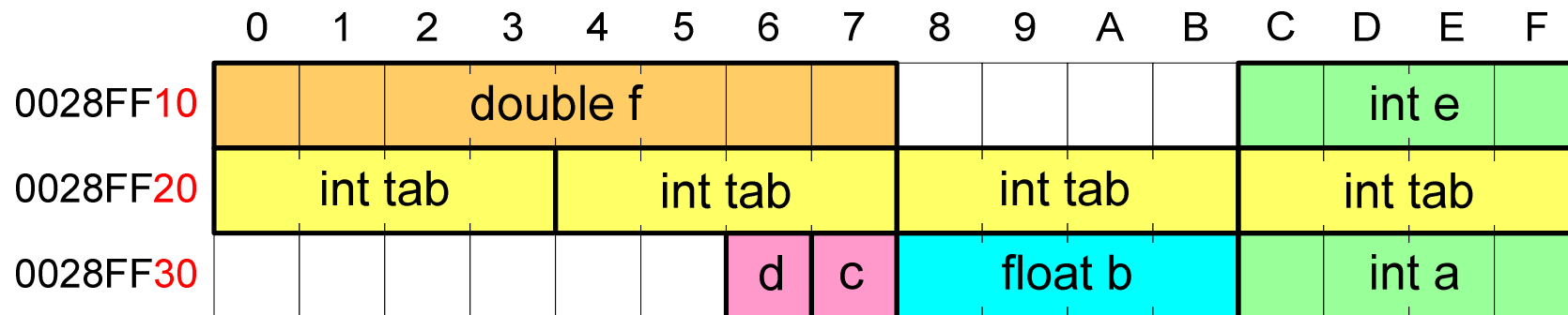
- **Wskaźnik** - zmienna mogąca zawierać adres obszaru pamięci
- najczęściej adres innej zmiennej (obiektu)

```
int a;  
float b;  
char c, d;  
int tab[4], e;  
double f;
```

- Zmienne przechowywane są w pamięci komputera



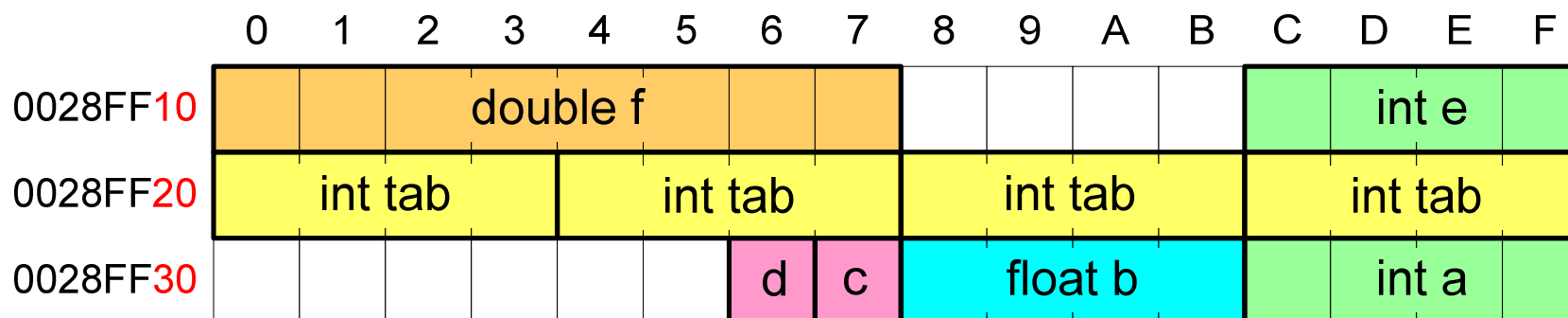
Co to jest wskaźnik?



- Każda zmienna znajduje się pod konkretnym adresem i zależnie od typu zajmuje określoną liczbę bajtów
- Podczas kompilacji wszystkie nazwy zmiennych zastępowane są ich adresami
- Wyświetlenie adresu zmiennej:

```
printf("Adres zmiennej a: %p\n", &a);  
printf("Adres tablicy tab: %p\n", tab);
```

Co to jest wskaźnik?



- Każda zmienna znajduje się pod konkretnym adresem i zależnie od typu zajmuje określoną liczbę bajtów
- Podczas kompilacji wszystkie nazwy zmiennych zastępowane są ich adresami
- Wyświetlenie adresu zmiennej:

```
Adres zmiennej a: 0028FF3C
Adres tablicy tab: 0028FF20
```

```
printf("Adres zmiennej a: %p\n", &a),
printf("Adres tablicy tab: %p\n", tab);
```

Deklaracja wskaźnika

- Deklarując wskaźnik (zmienną wskazującą) należy podać **typ** obiektu na jaki on wskazuje, a jego **nazwę** poprzedzić symbolem gwiazdki (*)

| | | | |
|--------------------------|--------------------------|---------------------------|-------------------------|
| <code>typ *nazwa;</code> | <code>typ* nazwa;</code> | <code>typ * nazwa;</code> | <code>typ*nazwa;</code> |
|--------------------------|--------------------------|---------------------------|-------------------------|

- Deklaracja zmiennej wskaźnikowej do typu **int**

```
int *ptr;
```

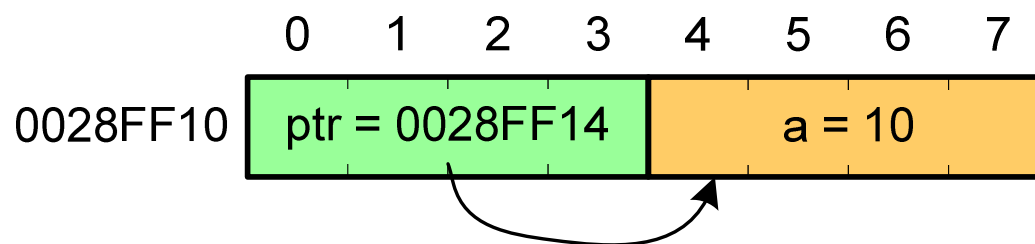
- Mówimy, że zmienna **ptr** jest typu: **wskaźnik do zmiennej typu int**
- Do przechowywania adresu zmiennej typu **double** trzeba zadeklarować zmienną typu: **wskaźnik do zmiennej typu double**

```
double *ptrd;
```

Przypisywanie wartości wskaźnikom

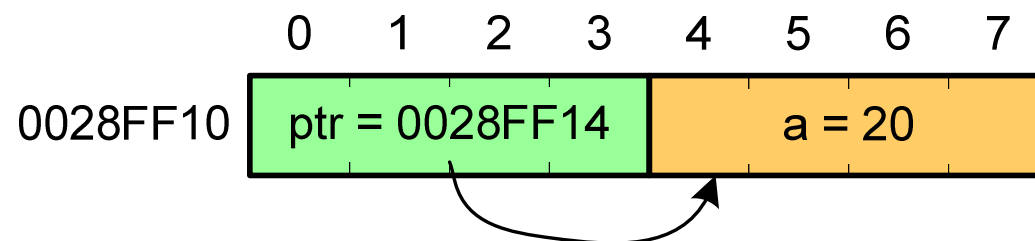
- Wskaźnikom można przypisywać adresy zmiennych
- Adresy takie tworzy się za pomocą operatora pobierania adresu **&**

```
int a = 10, *ptr;  
ptr = &a;
```



- Mając adres zmiennej można „dostać się” do jej wartości używając tzw. operatora wyłuskania (odwołania pośredniego) - gwiazdki (*****)

```
*ptr = 20;
```



- Wskaźnik pusty:

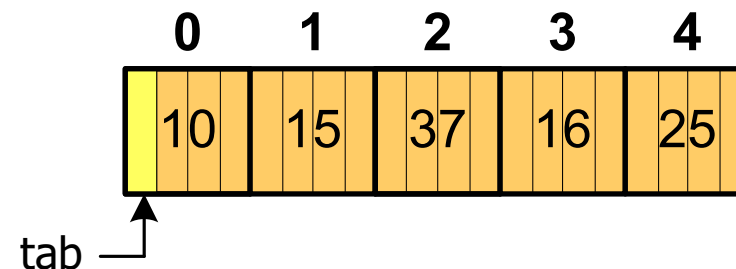
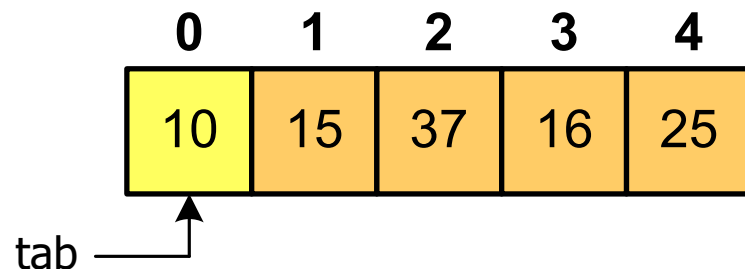
```
int *ptr = 0;
```

```
int *ptr = NULL;
```

Wskaźniki a tablice

- Nazwa tablicy jest jej adresem (dokładniej - adresem elementu o indeksie 0)

```
int tab[5] = {10, 15, 37, 16, 25};
```

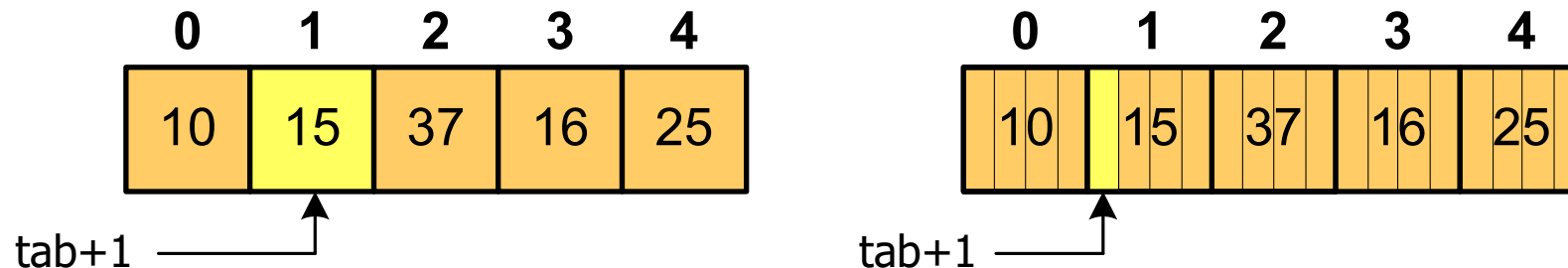


- Zastosowanie operatora ***** przed nazwą tablicy pozwala „dostać się” do zawartości elementu o indeksie 0

***tab** jest równoważne **tab[0]**

Wskaźniki a tablice

- Dodanie **1** do adresu tablicy przenosi nas do elementu tablicy o indeksie **1**



zatem: $*(tab+1)$ jest równoważne $tab[1]$

ogólnie: $*(tab+i)$ jest równoważne $tab[i]$

- W zapisie $*(tab+i)$ nawiasy są konieczne, gdyż operator $*$ ma bardzo wysoki priorytet

$x = *tab+1;$ jest równoważne $x = tab[0]+1;$

Dynamiczny przydział pamięci w języku C

- Kiedy stosuje się dynamiczny przydział pamięci?
 - gdy rozmiar tablicy będzie znany dopiero podczas wykonania programu a nie podczas jego kompilacji
 - gdy rozmiar tablicy jest bardzo duży
- Do dynamicznego przydziału pamięci stosowane są funkcje:
 - `calloc()`
 - `malloc()`
- Przydział pamięci następuje w obszarze **sterty** (stosu zmiennych dynamicznych)
- Przydzieloną pamięć należy zwolnić wywołując funkcję:
 - `free()`

Dynamiczny przydział pamięci w języku C

CALLOC

stdlib.h

```
void *calloc(size_t num, size_t size);
```

- Przydziela blok pamięci o rozmiarze **num*size** (mogący pomieścić tablicę **num**-elementów, każdy rozmiaru **size**)
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość **NULL**
- Przydzielona pamięć jest inicjowana zerami (bitowo)
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;  
tab = (int *) calloc(10, sizeof(int));
```

Dynamiczny przydział pamięci w języku C

MALLOC

stdlib.h

```
void *malloc(size_t size);
```

- Przydziela blok pamięci o rozmiarze określonym parametrem **size**
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość **NULL**
- Przydzielona pamięć nie jest inicjowana
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;  
tab = (int *) malloc(10*sizeof(int));
```

Dynamiczny przydział pamięci w języku C

FREE

stdlib.h

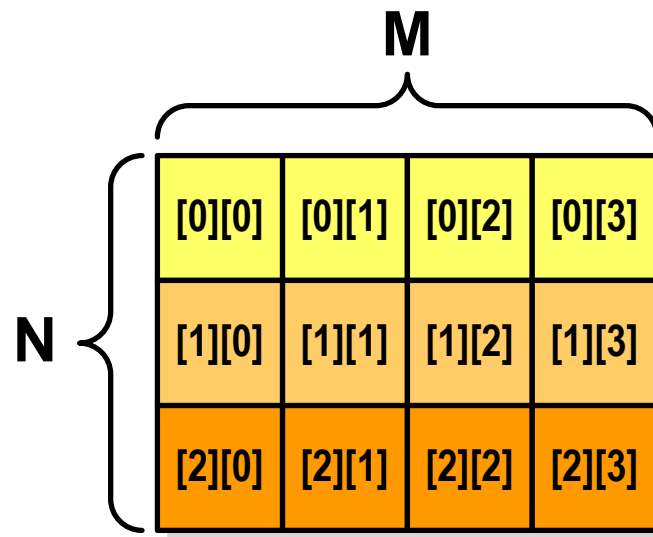
```
void *free(void *ptr);
```

- Zwalnia blok pamięci wskazywany parametrem **ptr**
- Wartość **ptr** musi być wynikiem wywołania funkcji **calloc()** lub **malloc()**

```
int *tab;  
tab = (int *) calloc(10, sizeof(int));  
/* ... */  
free(tab);
```

Dynamiczny przydział pamięci na macierz

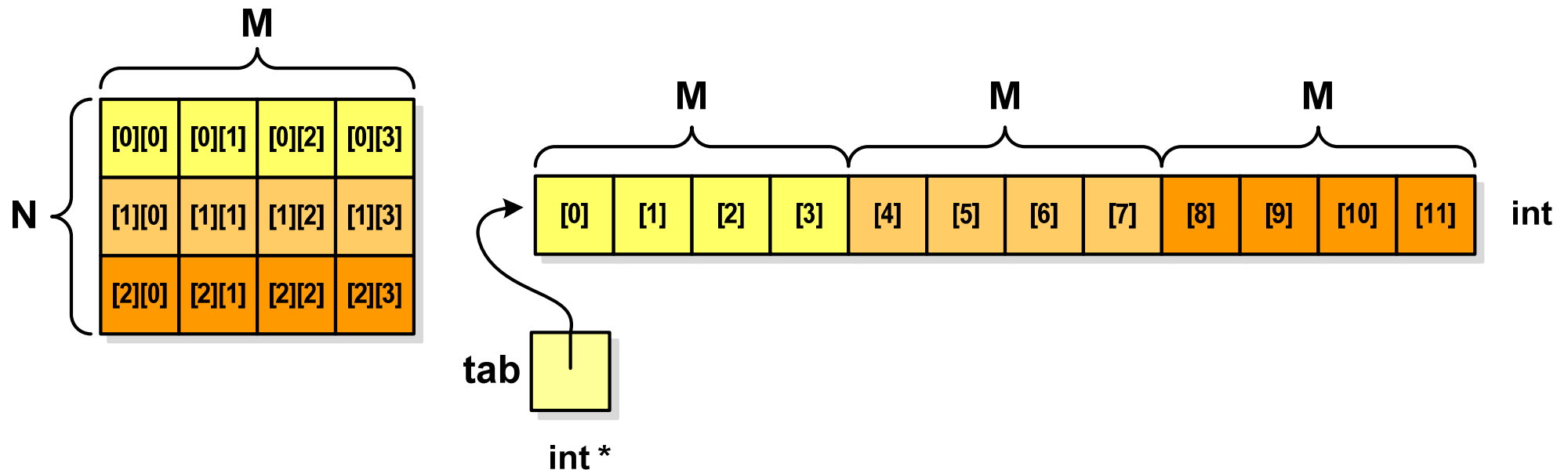
- Funkcje `calloc()` i `malloc()` umożliwiają bezpośrednio przydział pamięci tylko na wektor elementów
- Dynamiczny przydział pamięci na macierz wymaga zastosowania specjalnych metod
- Przydzielamy pamięć na macierz zawierającą **N-wierszy** i **M-kolumn**



Dynamiczny przydział pamięci na macierz (1)

- Wektor $N \times M$ -elementowy
- Przydział pamięci:

```
int *tab = (int *) calloc(N*M, sizeof(int));
```



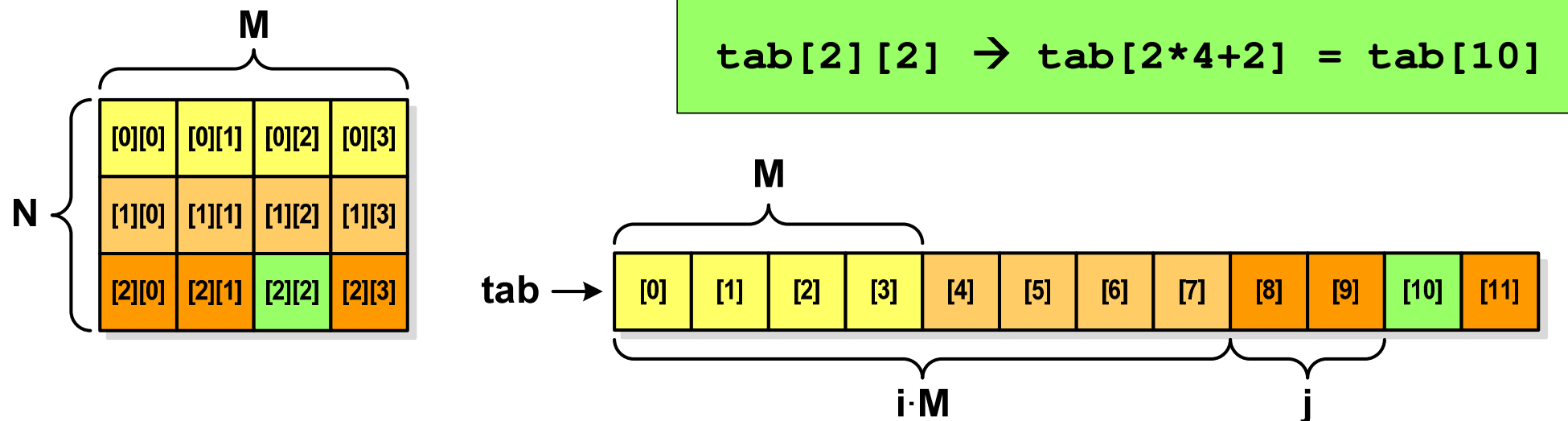
Dynamiczny przydział pamięci na macierz (1)

- Odwołanie do elementów macierzy:

`tab[i*M+j]`

lub

`*(tab+i*M+j)`



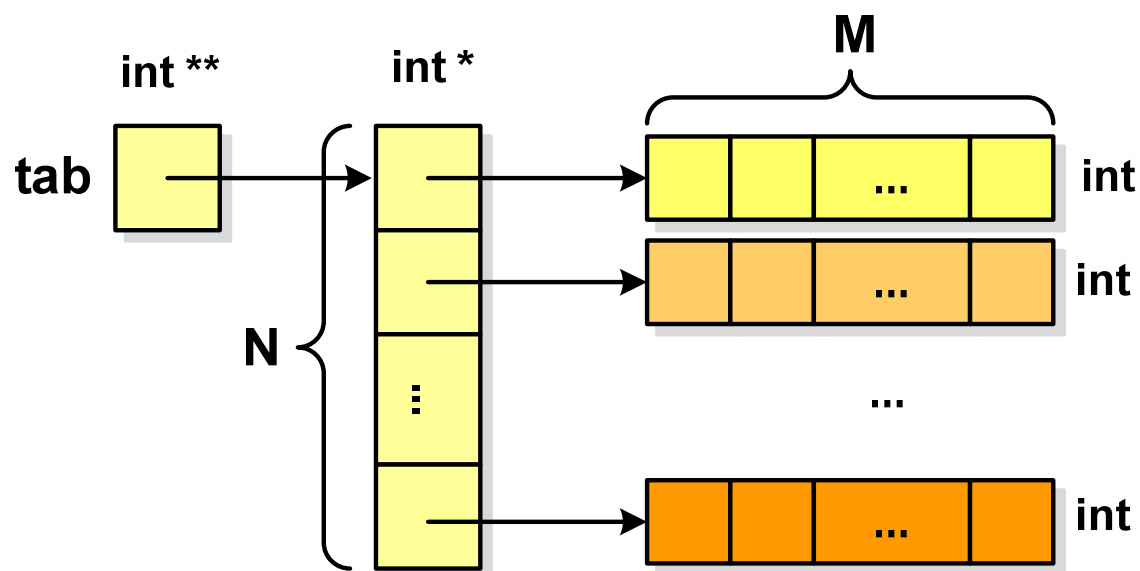
- Zwolnienie pamięci:

`free(tab);`

Dynamiczny przydział pamięci na macierz (2)

- N-elementowy wektor wskaźników + N-wektorów M-elementowych
- Przydział pamięci:

```
int **tab = (int **) calloc(N, sizeof(int *));  
for (i=0; i<N; i++)  
    tab[i] = (int *) calloc(M, sizeof(int));
```

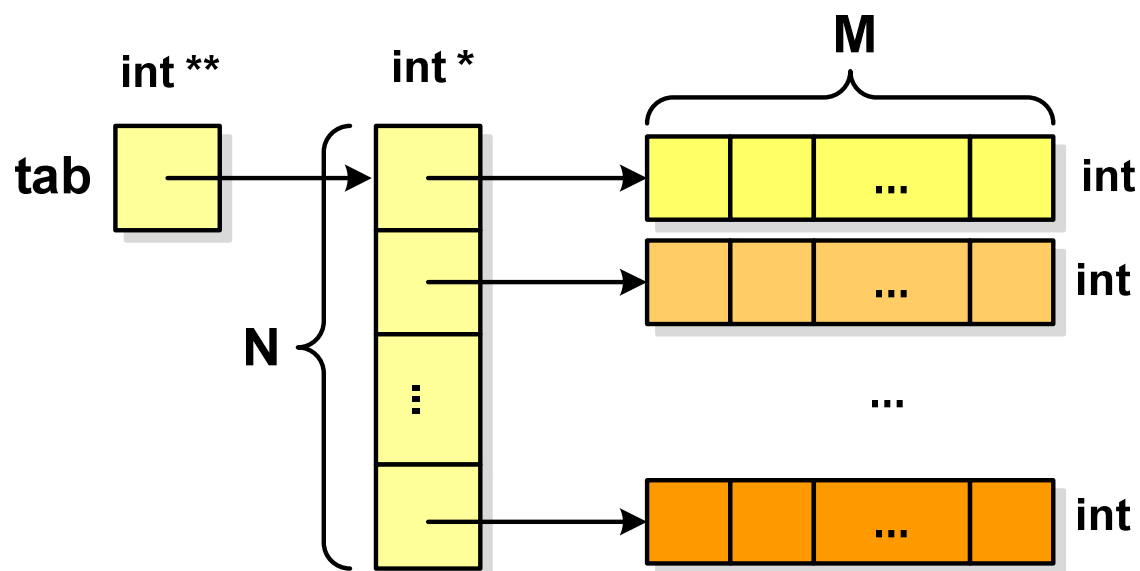


Dynamiczny przydział pamięci na macierz (2)

- Odwołania do elementów macierzy:
- Zwolnienie pamięci:

`tab[i][j]`

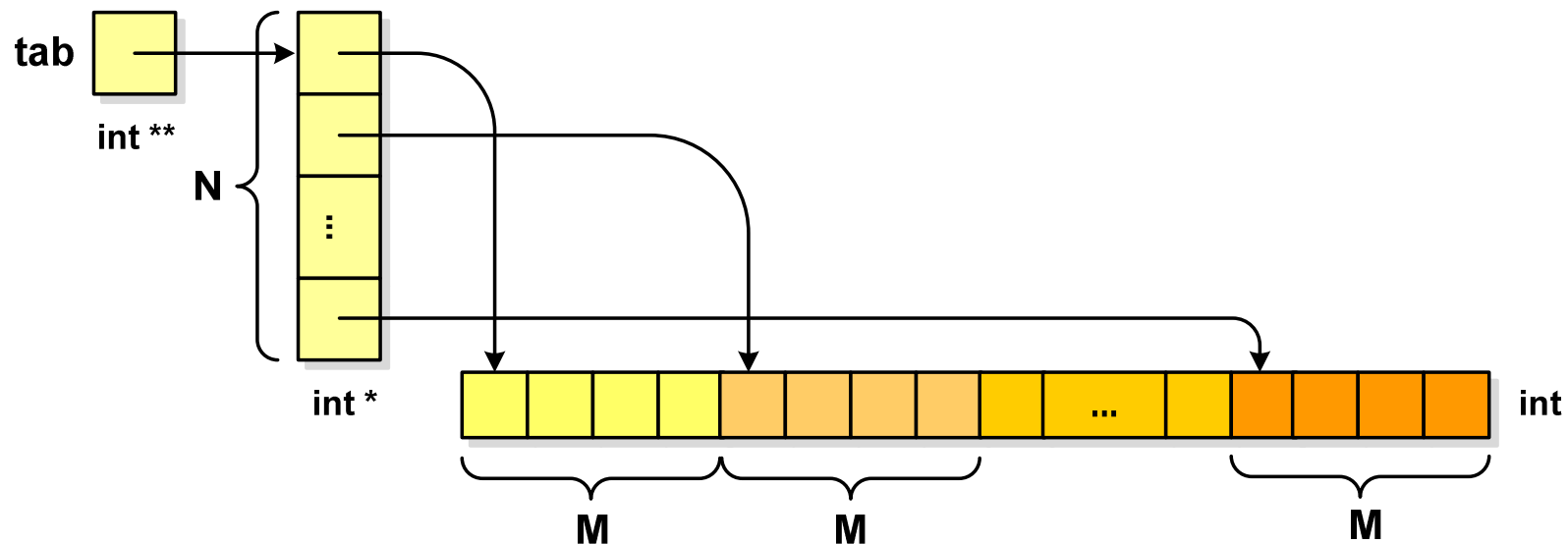
```
for (i=0; i<N; i++)  
    free(tab[i]);  
free(tab);
```



Dynamiczny przydział pamięci na macierz (3)

- N-elementowy wektor wskaźników + wektor N×M-elementowy
- Przydział pamięci:

```
int **tab = (int **) malloc(N*sizeof(int *));  
tab[0] = (int *) malloc(N*M*sizeof(int));  
for (i=1; i<N; i++)  
    tab[i] = tab[0]+i*M;
```



Dynamiczny przydział pamięci na macierz (3)

- Odwołania do elementów macierzy:
- Zwolnienie pamięci:

`tab[i][j]`

```
free (tab[0]);  
free (tab);
```

