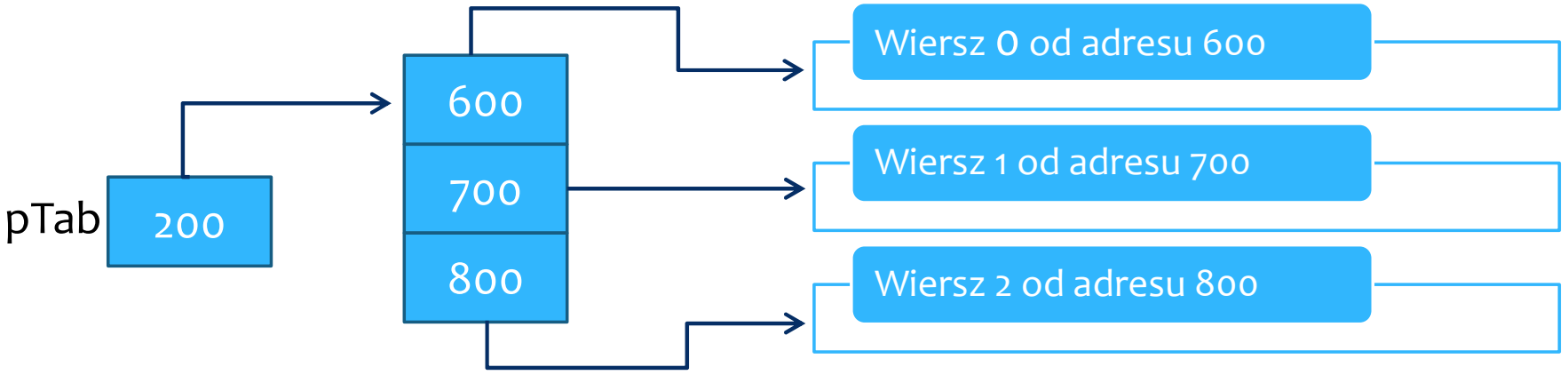
Alokacja pamięci dla tablic dwuwymiarowych oraz przekazywanie takiej tablicy do funkcji

Alokacja tablicy 2D

* Trzy sposoby dynamicznej alokacji
  1. Alokacja **najpierw** tablicy wskaźników na wiersze a **potem** alokacja poszczególnych wierszy (w pętli, niezależnie)
  2. Alokacja **najpierw** tablicy wskaźników do wierszy, a **potem** sumarycznego ciągłego obszaru na wiersze i podwiązanie elementów tablicy wskaźników do wierszy (w pętli) do kolejnych wierszy w zaalokowanym ciągłym obszarze
  3. Alokacja wyłącznie jednego ciągłego obszaru o ilości elementów równej sumie elementów wierszy (ilość wierszy * ilość kolumn)

Metoda 1

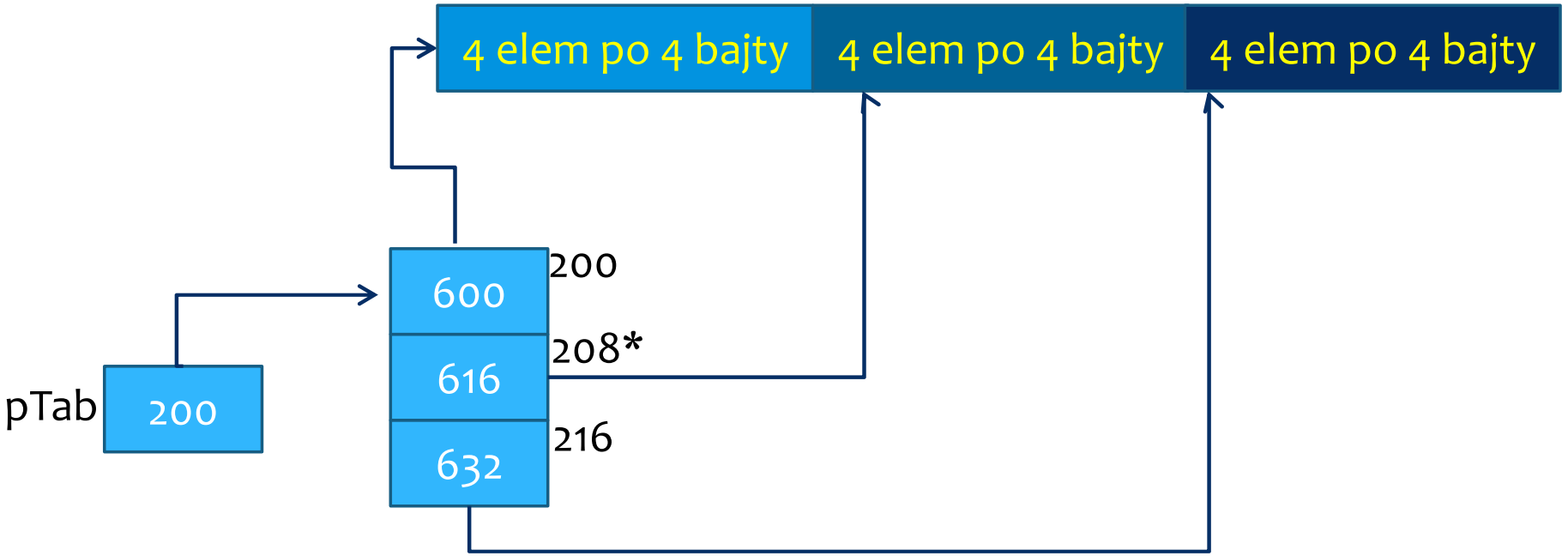
* **Nie ma gwaracji,** że tablica będzie zajmować **ciągły** obszar pamięci (jak jest przy niedynamicznej deklaracji tablicy dwu-wymiarowej np. `int t[3][4]; //` kolejne wiersze są w pamięci w obszarze ciągłym wiersz za wierszem).
* Można używać dalej tablicy w zwykły sposób `t[1][2] = 2;`
* Sposób alokacji dla tablicy (3x4) `int** pTab = NULL;`



* założenie: adresy są 8 bajtowe czyli komórka zawierająca adres zajmuje 8 bajtów (wartości adresów są przykładowe)

Metoda 2

* Wiersze tablicy zajmują **ciągły** obszar pamięci (wiersz za wierszem).
* Można używać dalej tablicy w zwykły sposób t[1][2] = 2;
* Sposób alokacji dla tablicy (3x4) int** pTab = NULL;



* założenie: adresy są 8 bajtowe czyli komórka zawierająca adres zajmuje 8 bajtów, każdy element tej tablicy jest typu int* (int zajmuje 4 bajty czyli 4 int-y 16 bajtów)

Metoda 3

* Wiersze tablicy zajmują **ciągły** obszar pamięci (wiersz za wierszem).
* **Nie można** używać dalej tablicy w zwykły sposób `t[1][2] = 2;`
* Aby dostać się do elementu tablicy należy obliczyć adres tego elementu np.*(pTab + (1*4)+2) // 1-numer wiersza, 2-numer kolumny, 4-ilość kolumn)
* Sposób alokacji dla tablicy (3x4) `int* pTab = NULL;`

Przekazywanie do funkcji

```
// Takie podejście pozwala przekazać do funkcji macierz
// o dowolnym rozmiarze

#define MATRIX_SIZE   5

double** pTab = NULL;
CreateMatrix( &pTab, MATRIX_SIZE );
 // można inaczej ale na razie będziemy to robić tak (projekt z macierzą)
 // initialize pTab, parametrem jest adres pTab bo jest to parametr wyjsciowy!!!
printMatrix( pTab, MATRIX_SIZE  );
double det = Det( pTab, MATRIX_SIZE );

//================================ nagłówki funckji =========
int CreateMatrix( double*** pMatrix, int nDim );
// zwraca 0 lub 1, 0-gdy alokacja się nie powiodła
// każdy wiersz macierzy musi być wyzerowany – memset()
double Det( double** pMatrix, int nDim );
// oblicza rekurencyjnie wyznacznik macierzy (np. poprzez rozwinięcie
// względem 0-wego wiersza
void printMatrix( double** pMatrix, int nDim );
```

Zadanie

* Proszę napisać funkcje:

```
int CreateMatrix( double*** pMatrix, int nDim );
void readMatrix( double** pMatrix, int nDim );
void printMatrix( double** pMatrix, int nDim );
void freeMatrix( double*** pMatrix, int nDim );
// po zwolnieniu macierzy wskaznik na macierz ma być pusty
```

* Kreowanie ma być pierwszą metodą
* Funkcja czytająca ma czytać macierz 5x5 z pliku wejsciowego `macierz.txt` (zapisane 5 wierszy po 5 liczb) . Zastosować funkcję `fscanf();`
* Przetestować w funkcji `main()`

Przekazywanie do funkcji

```
//INACZEJ
#define MATRIX_SIZE   5

double** pTab = CreateMatrix( MATRIX_SIZE );

//================================ nagłówki funckji =========
double** CreateMatrix( int nDim );
// zwraca NULL lub adres zaalokowanej tablicy,
// NULL - gdy alokacja się nie powiodła
// każdy wiersz macierzy musi być wyzerowany – memset()
```