Applications of Array Access Methods (1A)

Copyright (c) 2010 - 2018 Young W. Lim.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.
This document was produced by using LibreOffice.

Young Won Lim
5/11/19

1-d array pointer to consecutive 1-d arrays

```
int (*p)[4];

int a[4], b[4], c[4], d[4];
```

a pointer to a pointer array

1-d array pointer

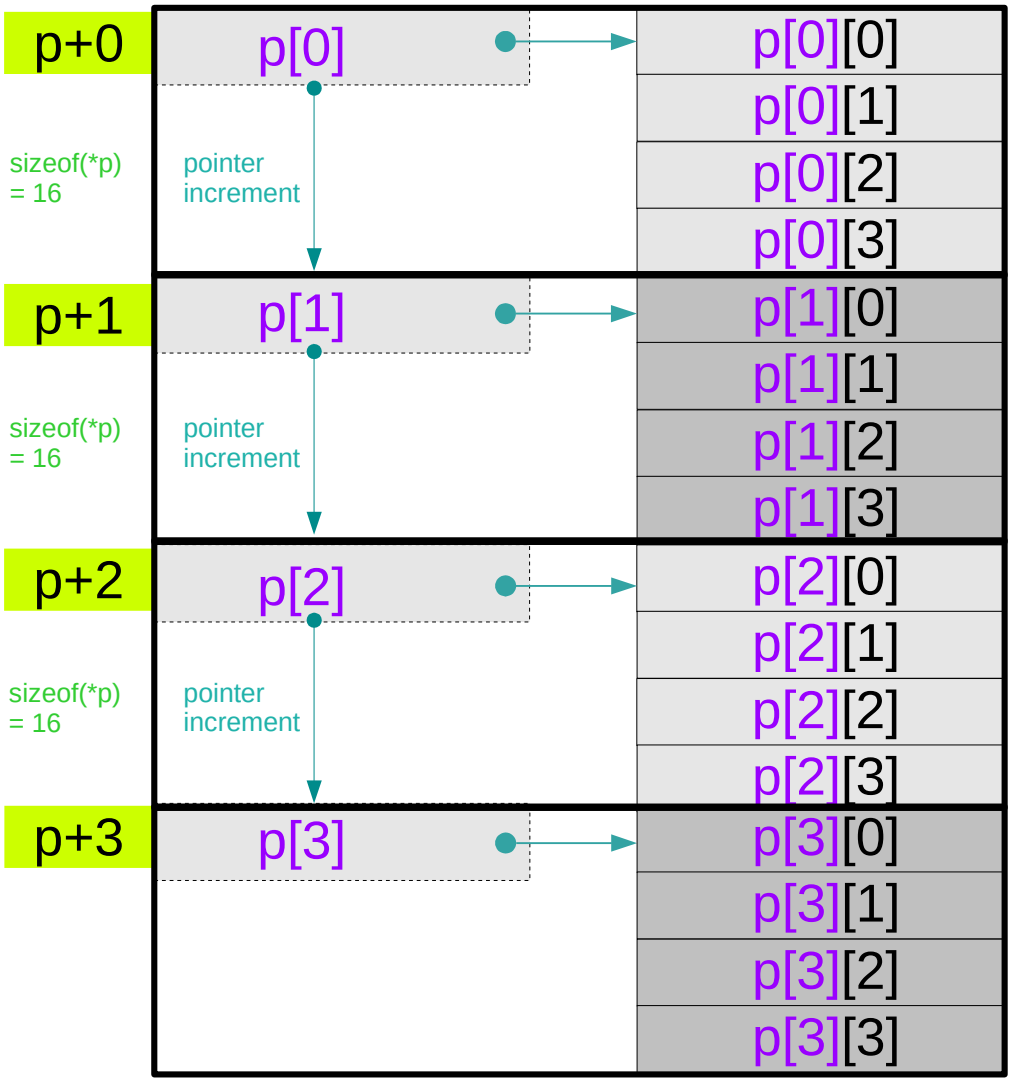
assignment

p = &a

equivalence

*(p+0) ≡ p[0] ≡ a
*(p+1) ≡ p[1] ≡ b
*(p+2) ≡ p[2] ≡ c
*(p+2) ≡ p[2] ≡ d

if arrays a, b, c, d
are consecutive



1-d array pointer to a 2-d arrays

```
int (*p)[4];

int x[4][4];
```

a pointer to a pointer array

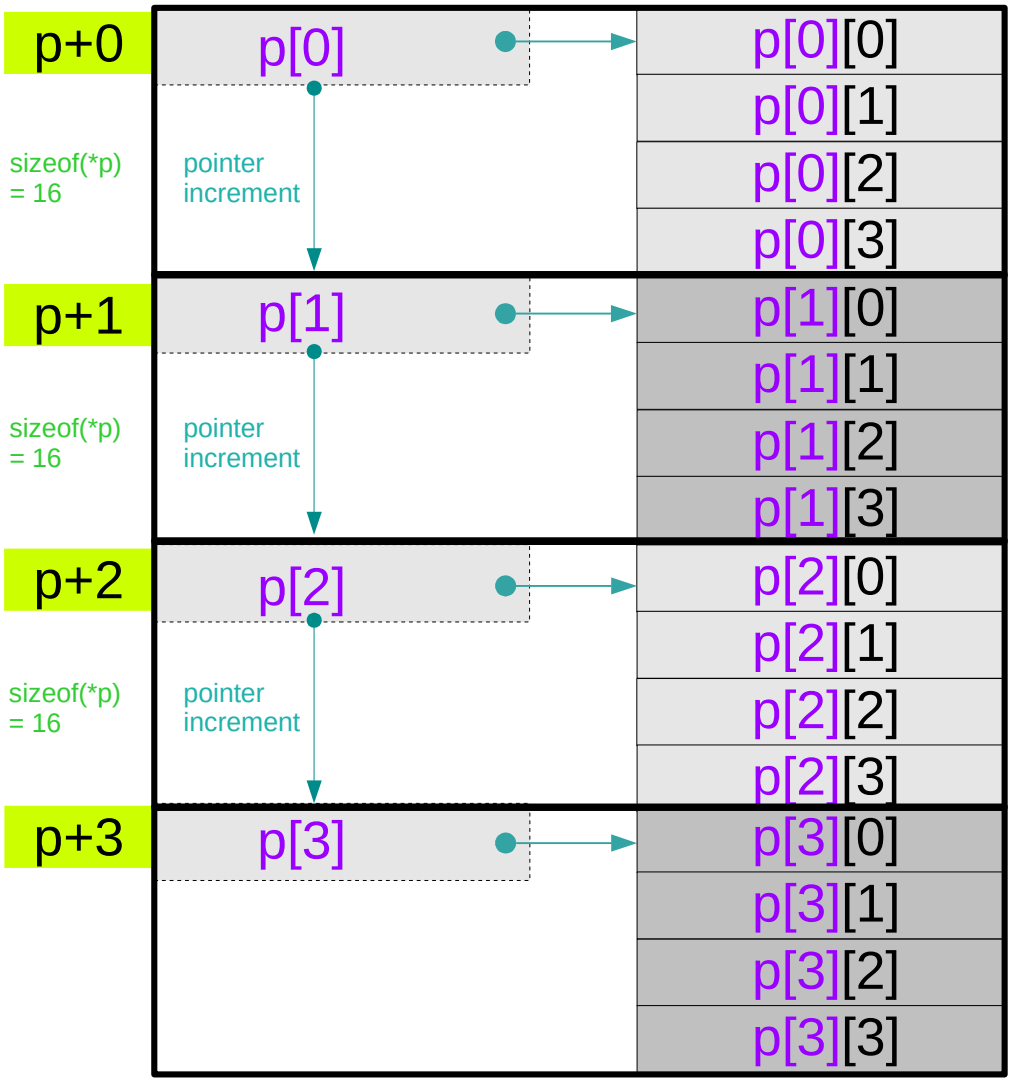
1-d array pointer

assignment

p = x

equivalence

*(p+0) ≡ p[0] ≡ x[0]
*(p+1) ≡ p[1] ≡ x[1]
*(p+2) ≡ p[2] ≡ x[2]
*(p+2) ≡ p[2] ≡ x[3]



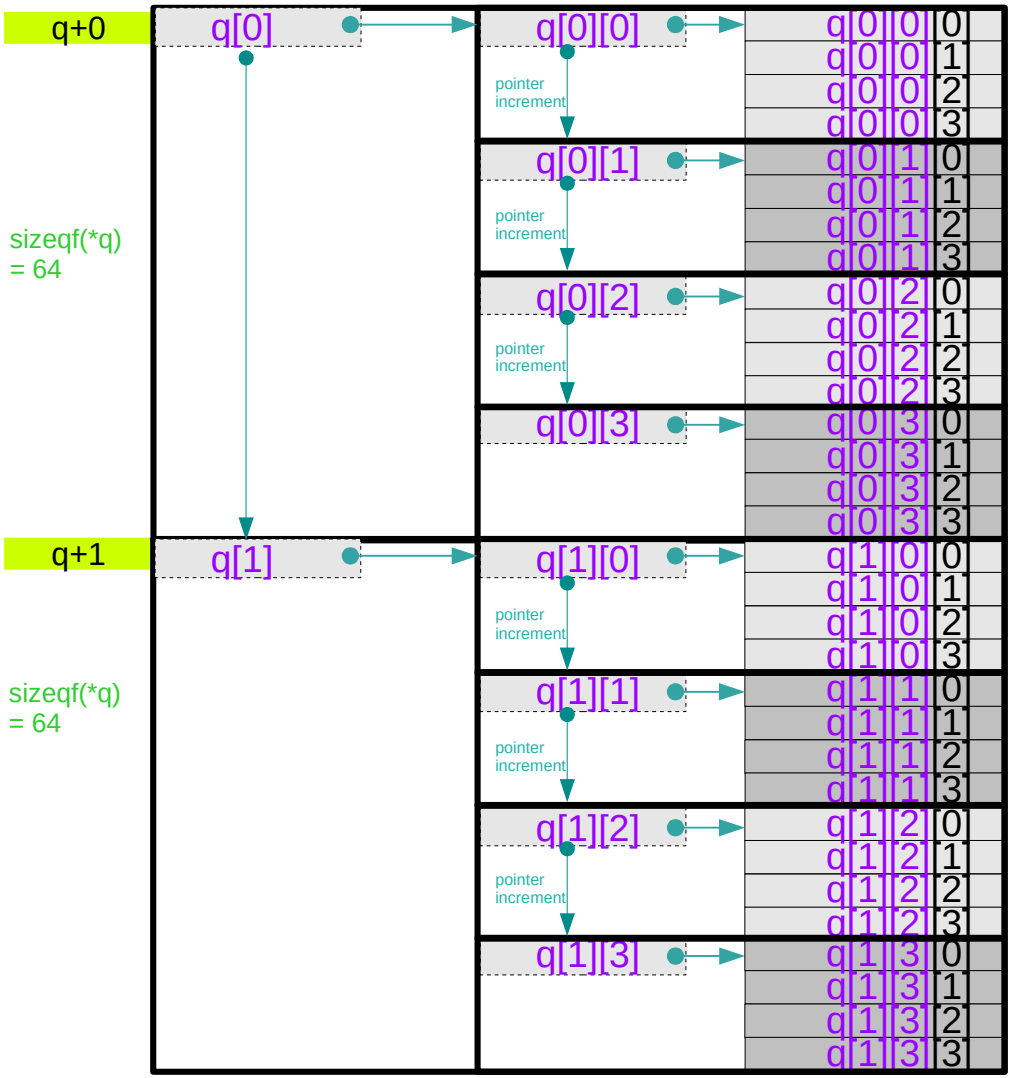
2-d array pointer to consecutive 2-d arrays

```
int (*q)[4][4];

int x[4][4], y[4][4];
```

a pointer to a pointer array

2-d array pointer



assignment

q = &x

equivalence

*(q+0) ≡ q[0] ≡ x
*(q+1) ≡ q[1] ≡ x+1 = y

~~q = &p~~

type mismatch

```
int (*p)[4];
```

if arrays x, y
are consecutive

int *p[4][1]

A **1-d** array **p** of integer pointers

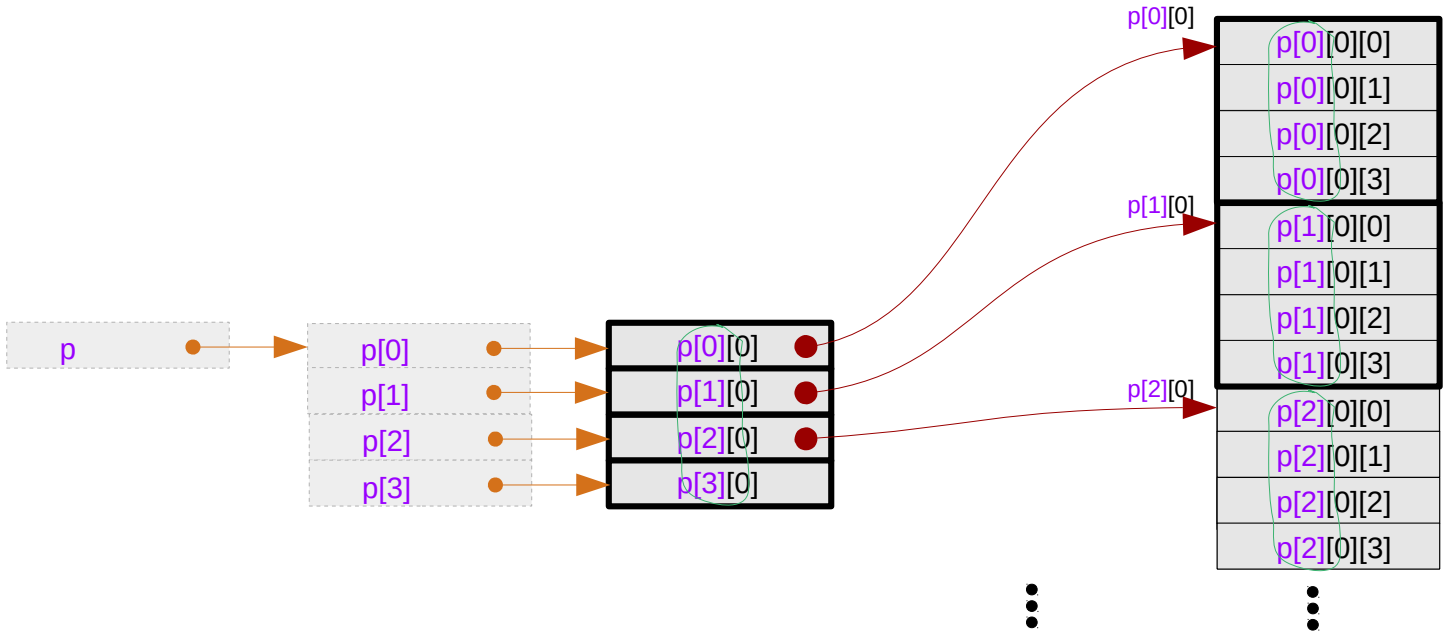
int *p[4][1] = {{x[0]}, {x[1]}, {x[2]}, {x[3]}};

int x[4][4];
int *p[4][1];

A **2-d** array of pointers

sizeof(p)=32 (4*8) a 2-d array of int pointers
sizeof(p[0])=8 a 1-d array of int pointer
sizeof(p[0][0])=8 an int pointer
sizeof(p[0][0][0])=4 an integer

An 2-d array **p**



int *p[1][4]

A **1-d** array **p** of integer pointers

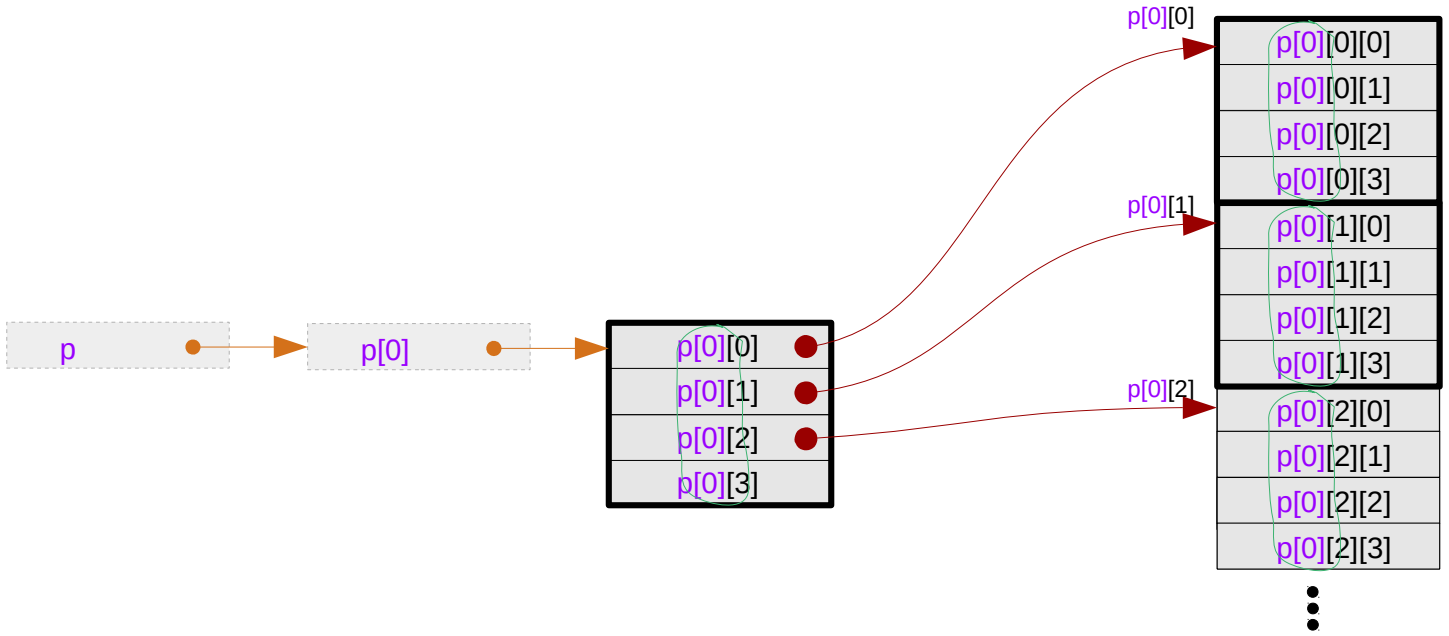
int *p[1][4] = {{x[0], x[1], x[2], x[3]}};

sizeof(p)=128 (4*4*8) a 2-d array of int pointers
sizeof(p[0])=32 a 1-d array of int pointer
sizeof(p[0][0])=8 an int pointer
sizeof(p[0][0][0])=4 an integer

An 2-d array **p**

int x[1][4];
int *p[1][4];

A **2-d** array of pointers



int *p[4][4]

A **1-d** array **p** of integer pointers

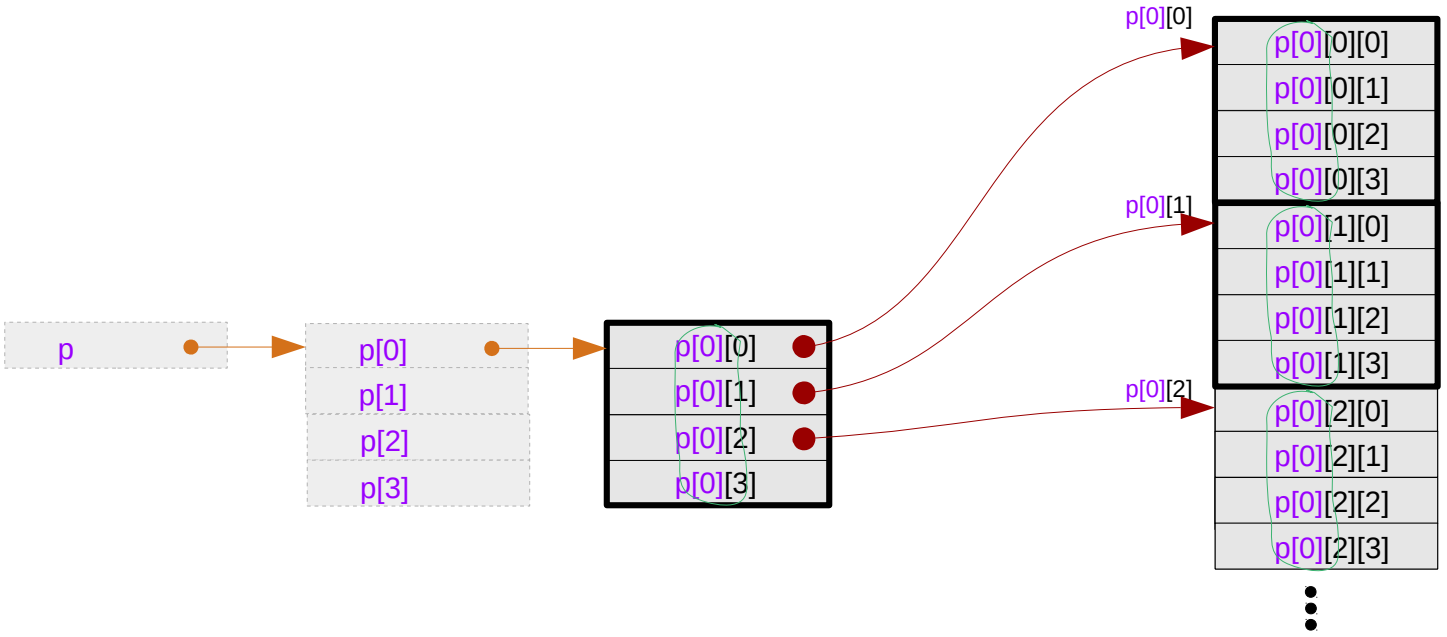
int *p[4][4] = {{x[0], x[1], x[2], x[3]}};

sizeof(p)=128 (4*4*8) a 2-d array of int pointers
sizeof(p[0])=32 a 1-d array of int pointer
sizeof(p[0][0])=8 an int pointer
sizeof(p[0][0][0])=4 an integer

An 2-d array **p**

int x[4][4];
int *p[4][4];

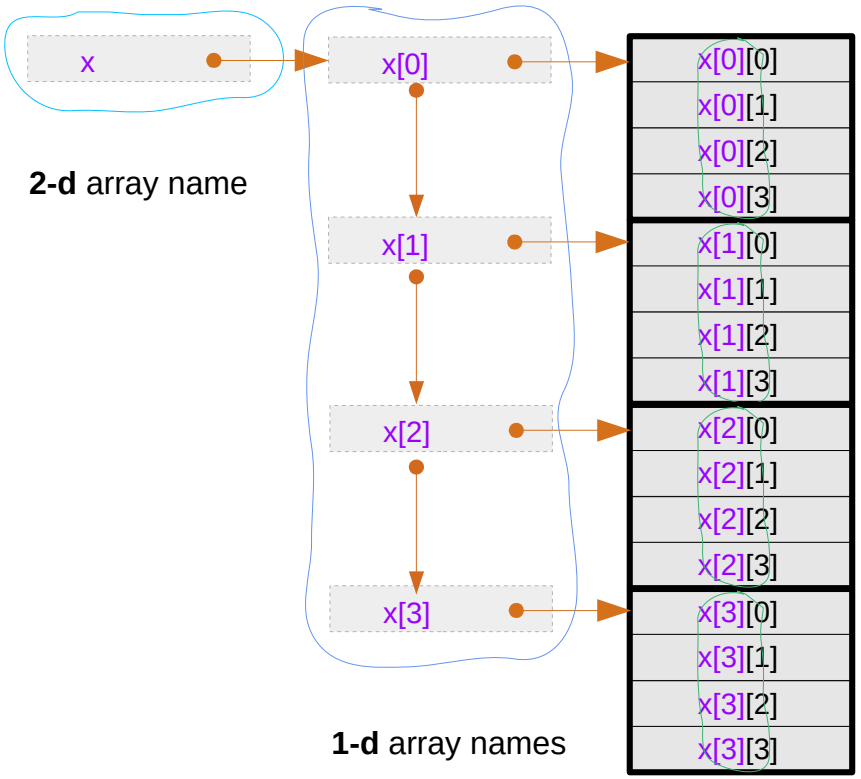
A **2-d** array of pointers



Accessing a 2-d array using pointers

int x[4][4];

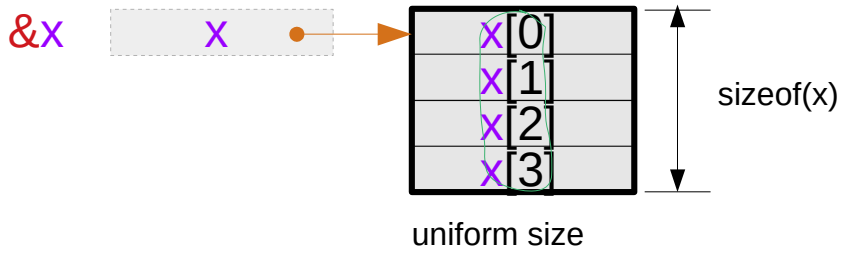
A **2-d** array



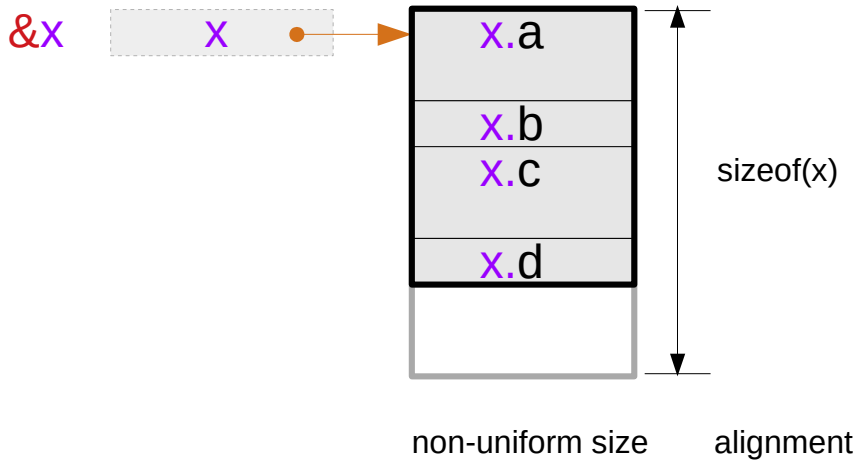
Comparison with a structure type

```
int x[4];
```

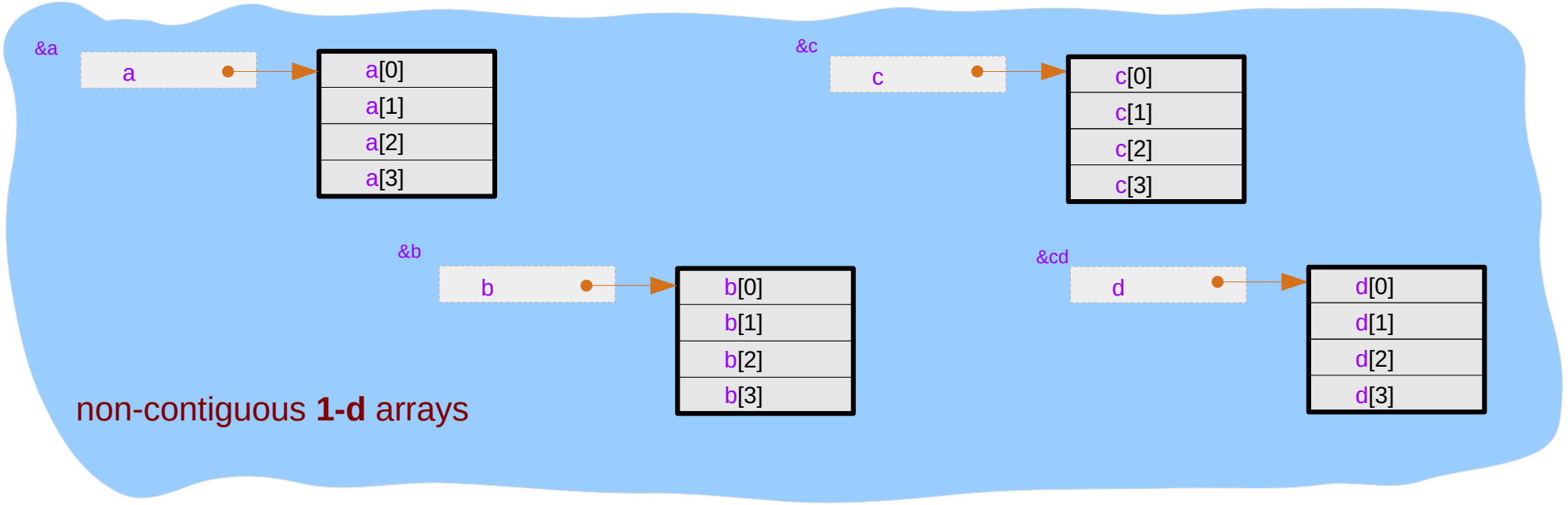
A **1-d** array




```
struct aaa {
    long  a;
    int   b;
    long  c;
    int   d;
} x;
```



Making a 2-d array from scattered 1-d arrays



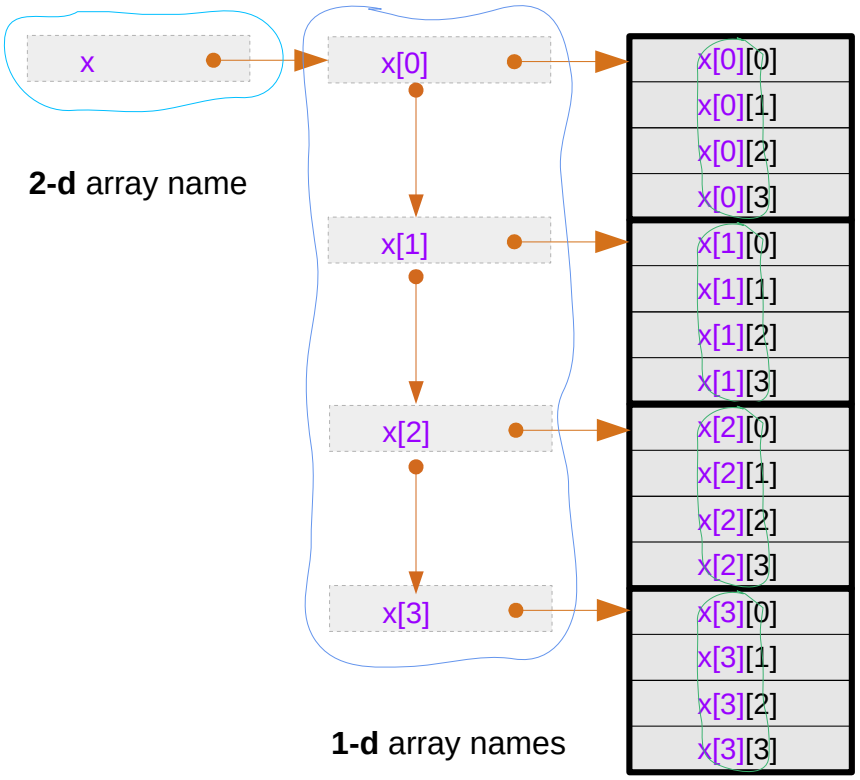
Accessing an artificial 2-**d** arrays

p[m][n] (*p[m])[n] (*p)[m][n]

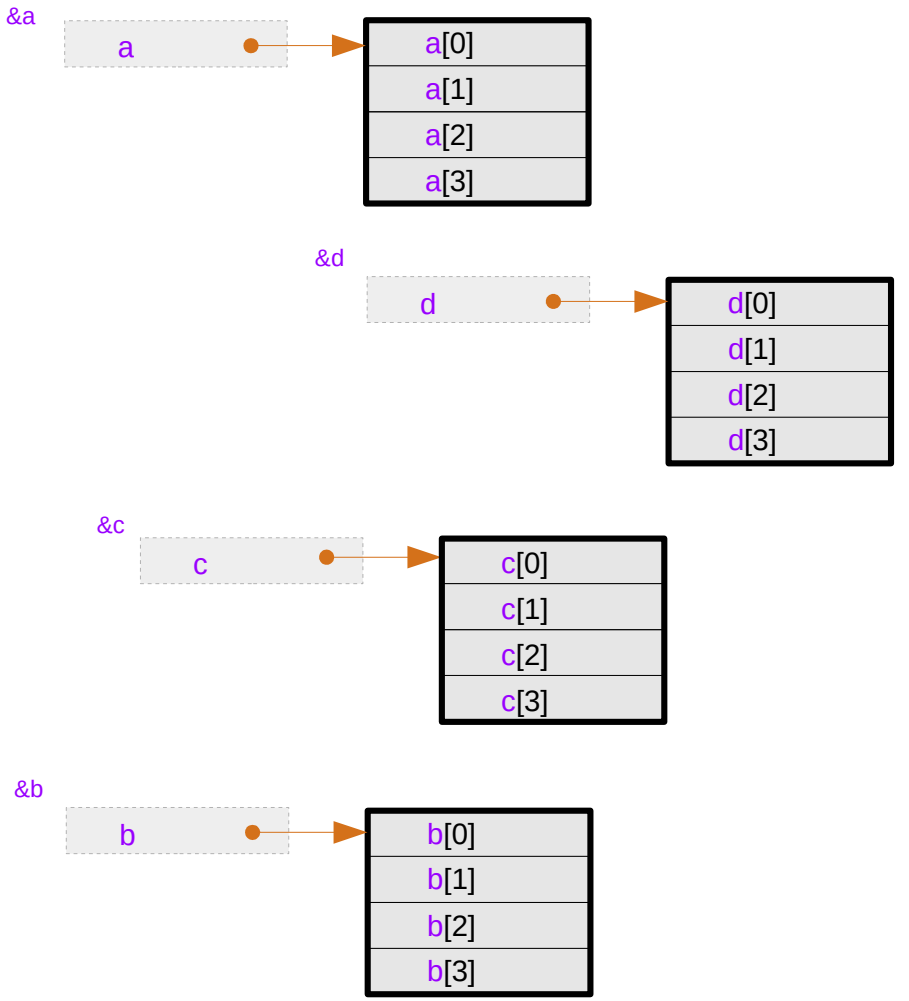
[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]

Contiguous and non-contiguous **1-d** arrays

2-d array = contiguous **1-d** arrays

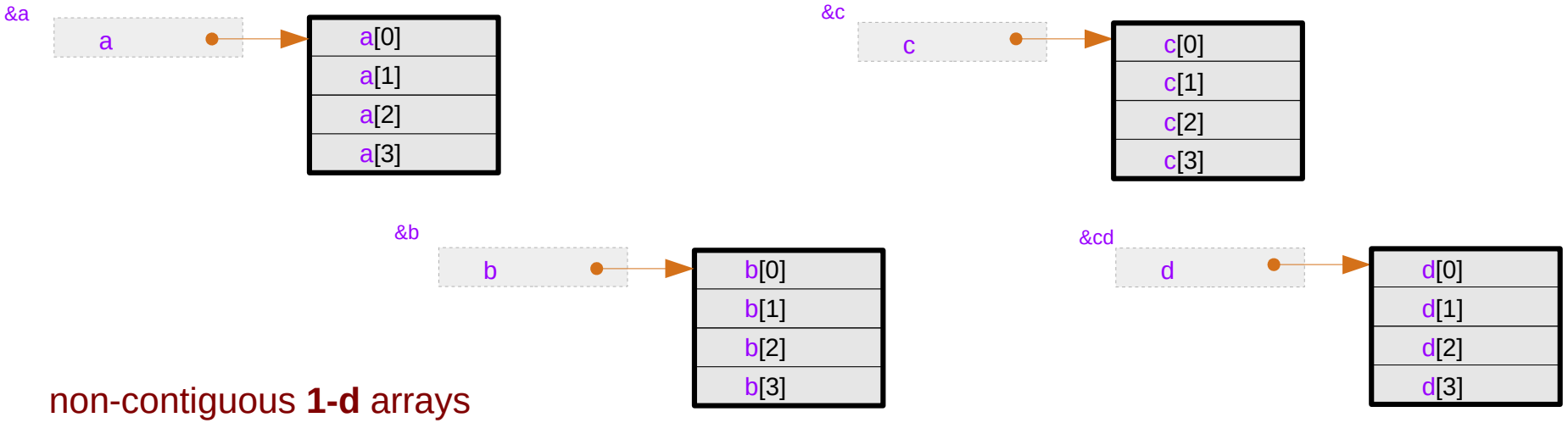


2-d array ⬅ non-contiguous **1-d** arrays



2-d array access of non-contiguous 1-d arrays

- **1-d** array of integer pointers — **int *p[4];** — **OK**
- **1-d** array pointer — **int (*p)[4];** — **X**
- **2-d** array pointer — **int (*p)[4][4];** — **X**
- array of **1-d** array pointers — **int (*p[4])[4];** — **OK**



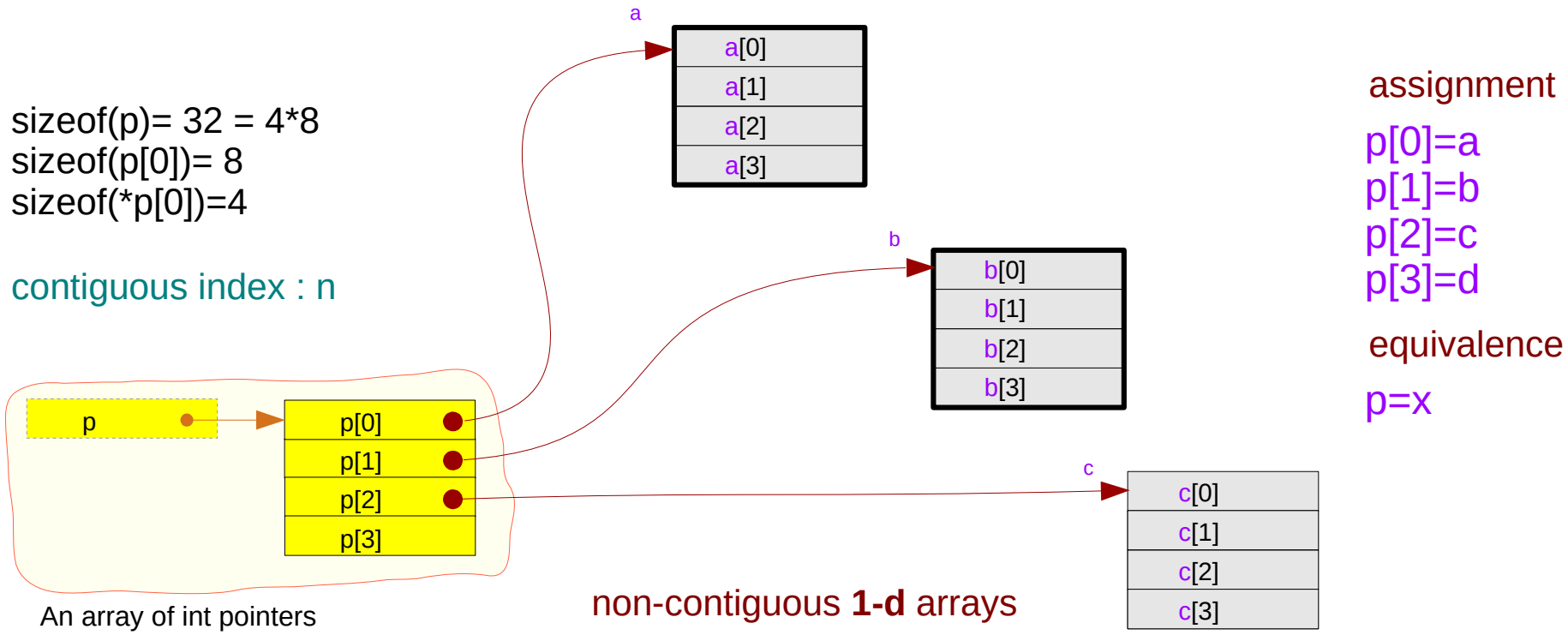
non-contiguous **1-d** arrays

int *p[4] = { a, b, c, d };

Type Definition

*(p[m]+n) ≡ p[m][n]

Access Method



sizeof(p)= 32 = 4*8
sizeof(p[0])= 8
sizeof(*p[0])=4

contiguous index : n

assignment

p[0]=a
p[1]=b
p[2]=c
p[3]=d

equivalence

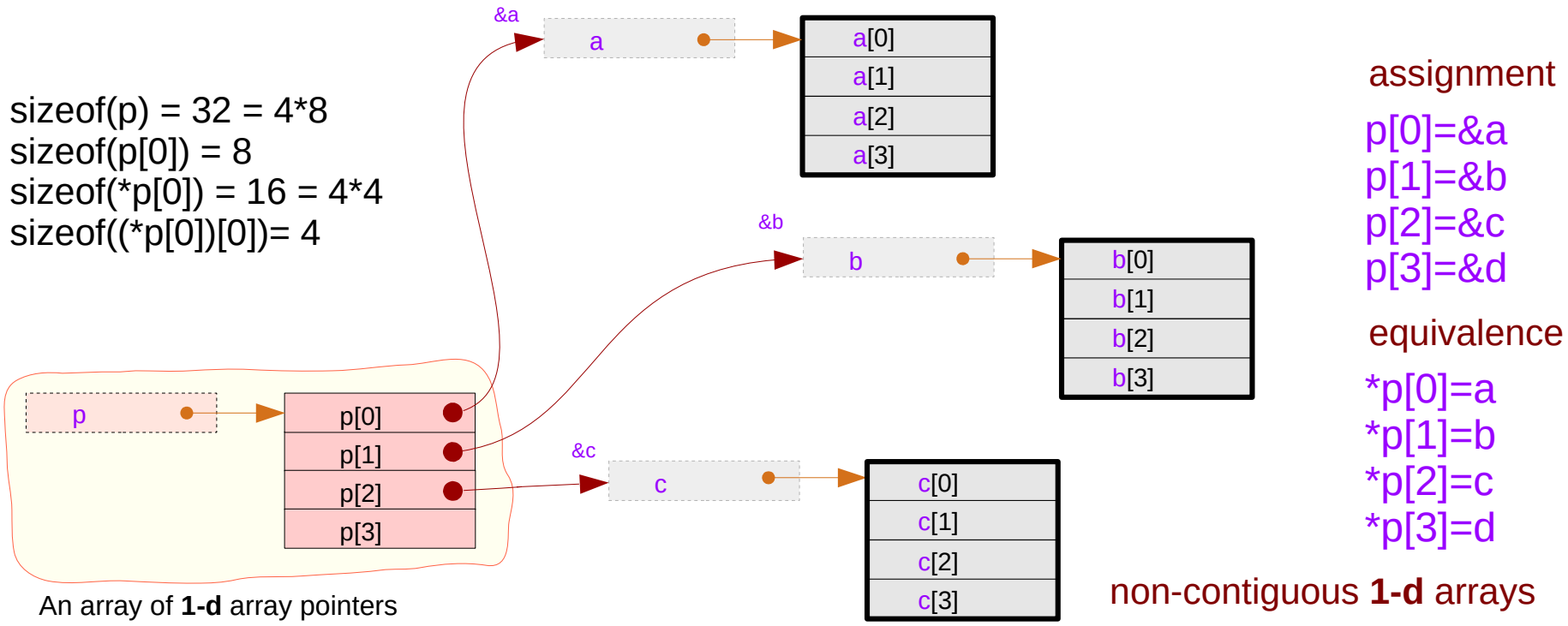
p=x

```
int (*p[4])[4] = { &a, &b, &c, &d };
```

Type Definition

```
(*p[m])[n];
```

Access Method



int (*p)[4] = &a;

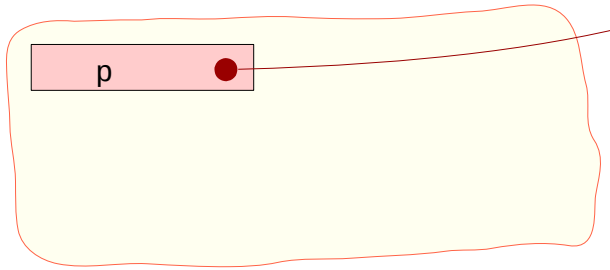
Type Definition

(*(p+m))[n]; ≡ p[m][n];

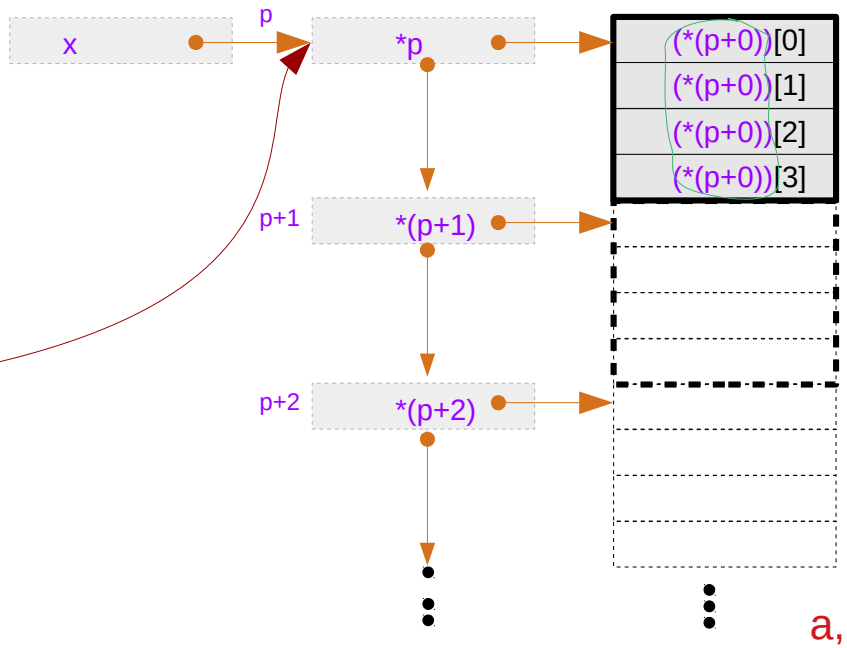
Access Method

sizeof(p)= 8
sizeof(*p)= 16 = 4*4
sizeof((*p)[0])= 4

contiguous index : m, n



A **1-d** array pointer



assignment

p=x

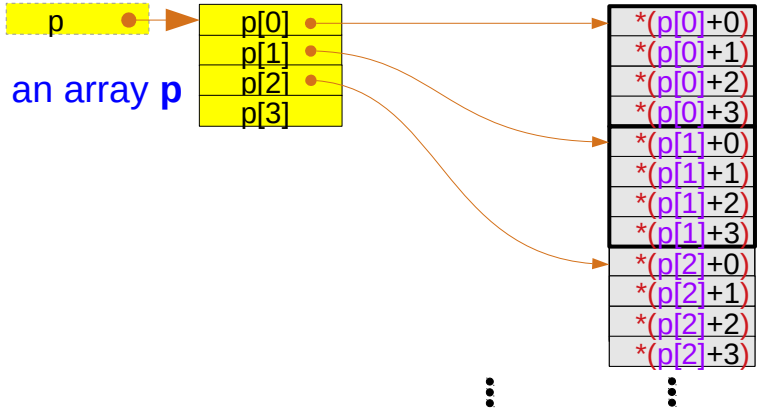
equivalence

p[0]=x[0]
p[1]=x[1]
p[2]=x[2]
p[3]=x[3]

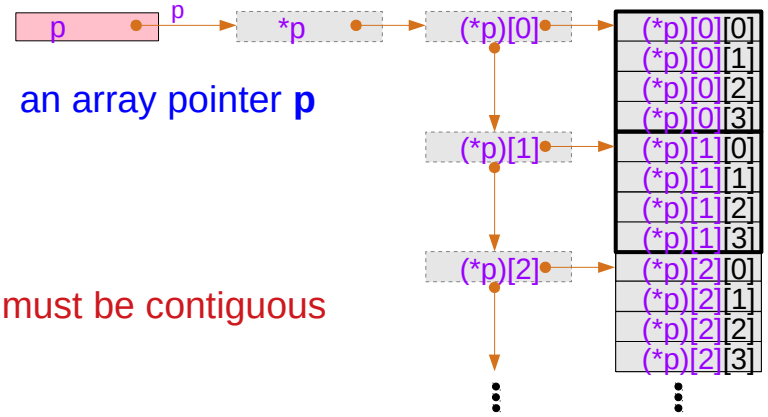
a, b, c, d : non-contiguous

Comparision

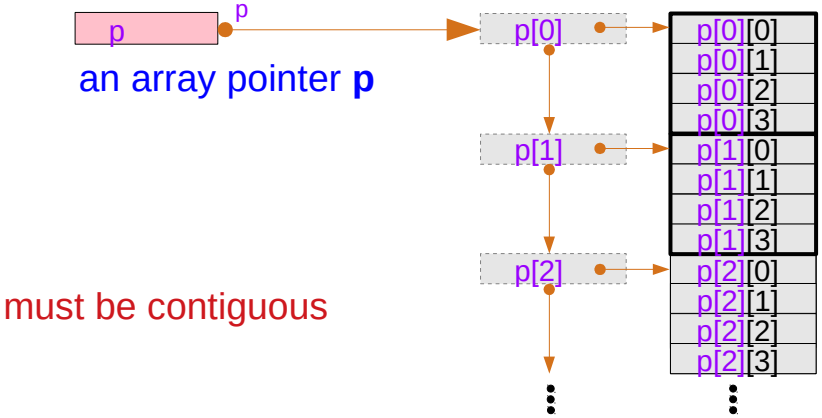
A **1-d** array **p** of integer pointers



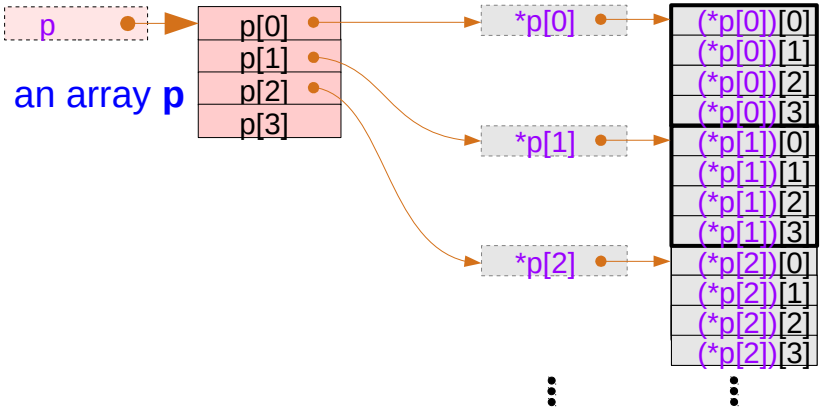
A **2-d** array pointer **p**



A **1-d** array pointer **p**



An array **p** of **1-d** array pointers



2-d array access over non-contiguous 1-d arrays

```
int *p[4] = { a, b, c, d };
```

Type Definition

```
*(p[m]+n)      ≡   p[m][n]
```

Access Method

```
int (*p[4])[4] = { &a, &b, &c, &d };
```

Type Definition

```
(*p[m])[n];        (*p)[m][n];
```

Access Methods

Example of **2-d** accessing non-contiguous **1-d** arrays

```
int a[4] = {1,2,3,4};
int c[4] = {9,10,11,12};
int b[4] = {5,6,7,8};
int d[4] = {13,14,15,16};
```

a, c, b, d : contiguous
a, b, c, d : non-contiguous

```
int (*p)[4] = {a, b, c, d};

   printf(" %d", p[i][j]);
```

incorrect results

```
int (*p)[4] = &a;

   printf(" %d", p[i][j]);
```

```
int (*p[4])[4] = {&a, &b, &c, &d};

    printf(" %d", (*p[i])[j]);
    printf(" %d", (*p)[i][j]);
```

No correct syntax

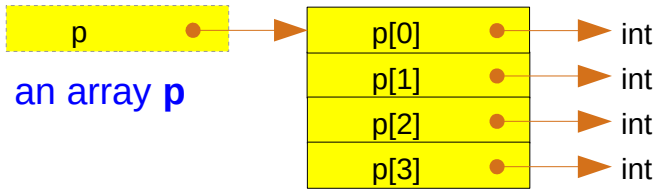
```
int (*p)[4][4] =

    printf(" %d", (*p)[i][j]);
```

Accessing contiguous and non-contiguous **1-d** arrays

A **1-d** array **p** of integer pointers

```
int *p[4];
```

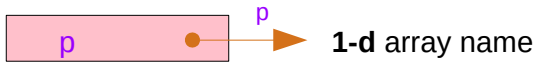


an array **p**

non-contiguous 1-d arrays are ok

A **1-d** array pointer **p**

```
int (*p)[4];
```

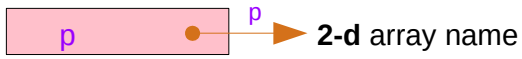


an array pointer **p**

Only for **contiguous** 1-d arrays

A **2-d** array pointer **p**

```
int (*p)[4][4];
```



an array pointer **p**

Only for **contiguous** 1-d arrays

An array **p** of **1-d** array pointers

```
int (*p[4])[4];
```

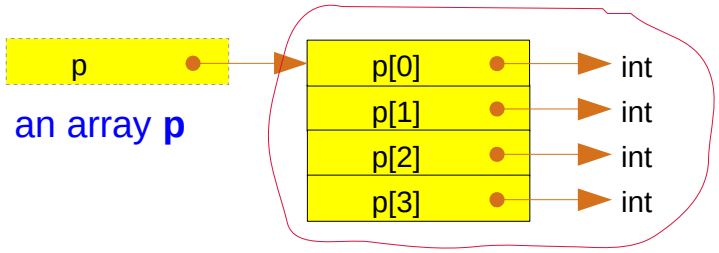
an array **p**

non-contiguous 1-d arrays are ok

Relaxing contiguity constraints

A **1-d** array **p** of integer pointers

```
int *p[4];
```



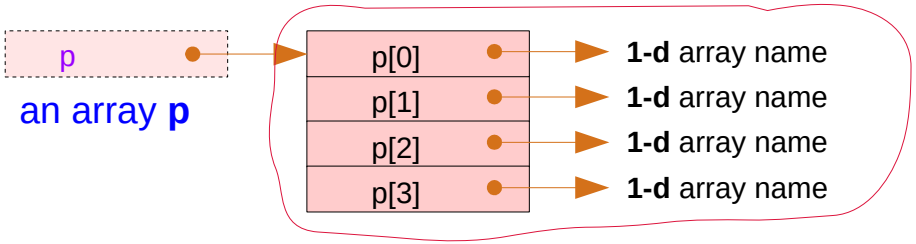
non-contiguous 1-d arrays are ok

these allocations can relax the contiguity constraints of 1-d arrays

these allocations can relax the contiguity constraints of 1-d arrays

An array **p** of **1-d** array pointers

```
int (*p[4])[4];
```



non-contiguous 1-d arrays are ok

Dynamic memory allocations

- Using a single pointer
- Using an array of pointers
- Using pointer to pointer (double pointers)
- Using double pointer and one malloc call

- Using a 1-d array pointer
- Using a 2-d array pointer
- Using an array of 1-d array pointers
- Using a 2-d array of pointers

Dynamic memory allocation methods (I)

1) using a single pointer

```
int *a = (int *) malloc(4 * 4 * sizeof(int));          *(a + i*4 +j) = i*4+j;
```

2) using an array of pointers

```
int *a[4];
for (i=0; i<4; i++)
    a[i] = (int *) malloc(4 * sizeof(int));          a[i][j] = i*4+j;
```

3) using pointer to pointer (double pointers)

```
int **a = (int **) malloc(4 * sizeof(int *));
for (i=0; i<4; ++i)
    a[i] = (int *) malloc(4 * sizeof(int));          a[i][j] = i*4+j;
```

4) using double pointer and one malloc call

```
int siz = sizeof(int *) * 4 + sizeof(int) * 4 * 4;
int **a = (int **) malloc(siz);
int *p = (int *) a + 4*sizeof(int *) / sizeof(int);
for (i=0; i<4; i++)     a[i] = p + 4*i;          a[i][j] = i*4+j;
```

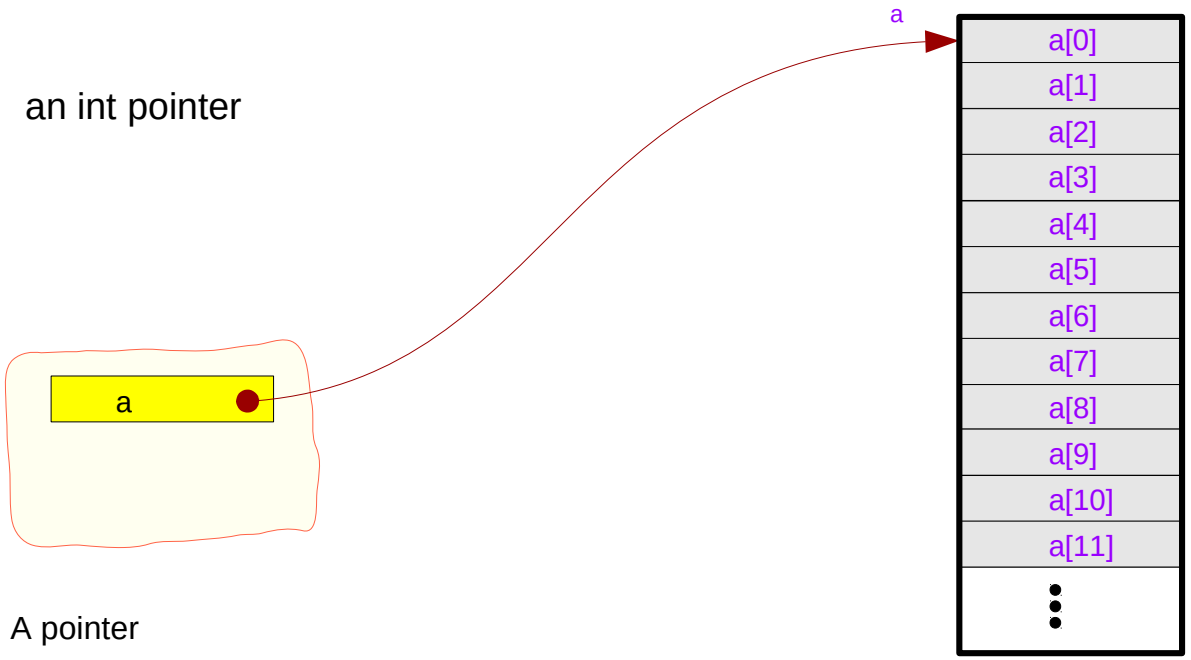
https://www.geeksforgeeks.org/dynamically-allocate-2d-array-c/

Case 1) 1-d access

```
int *a = (int *) malloc(4 * 4 * sizeof(int));
```

```
*(a + i*4 +j) = i*4+j;
```

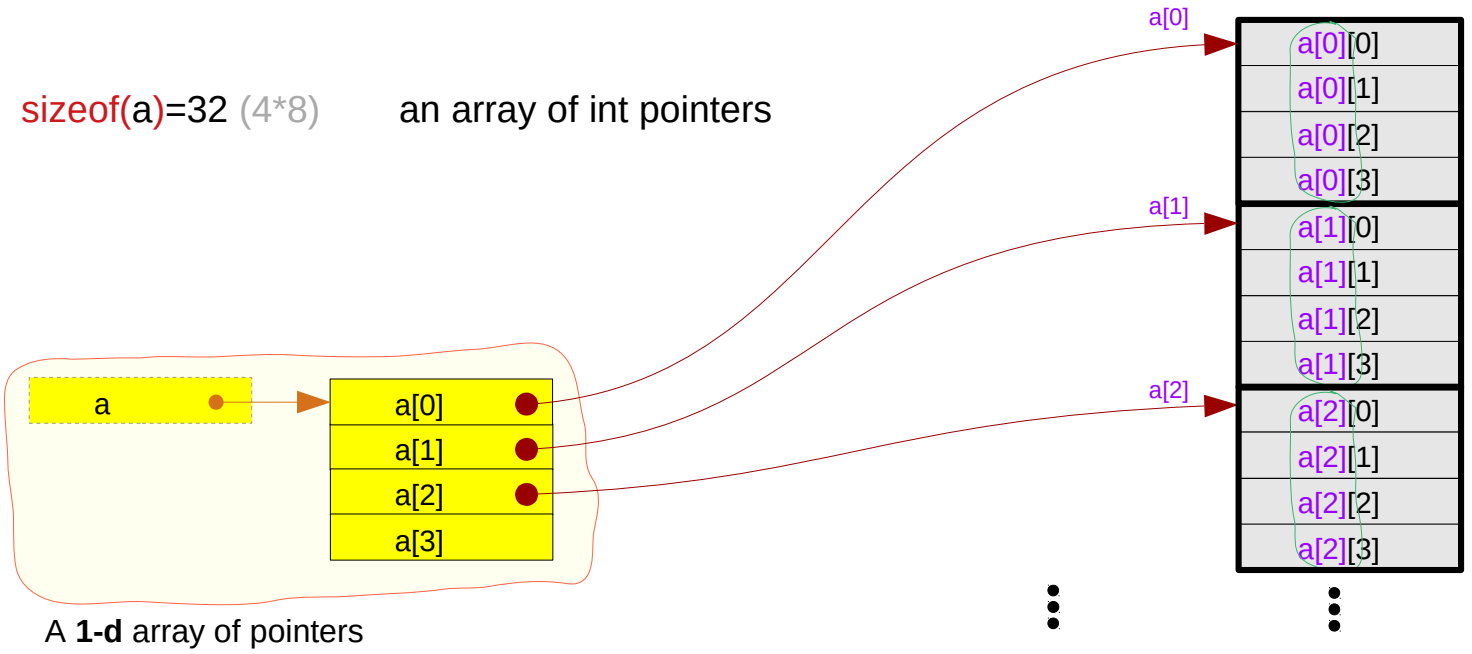
sizeof(a)=8 an int pointer



A pointer

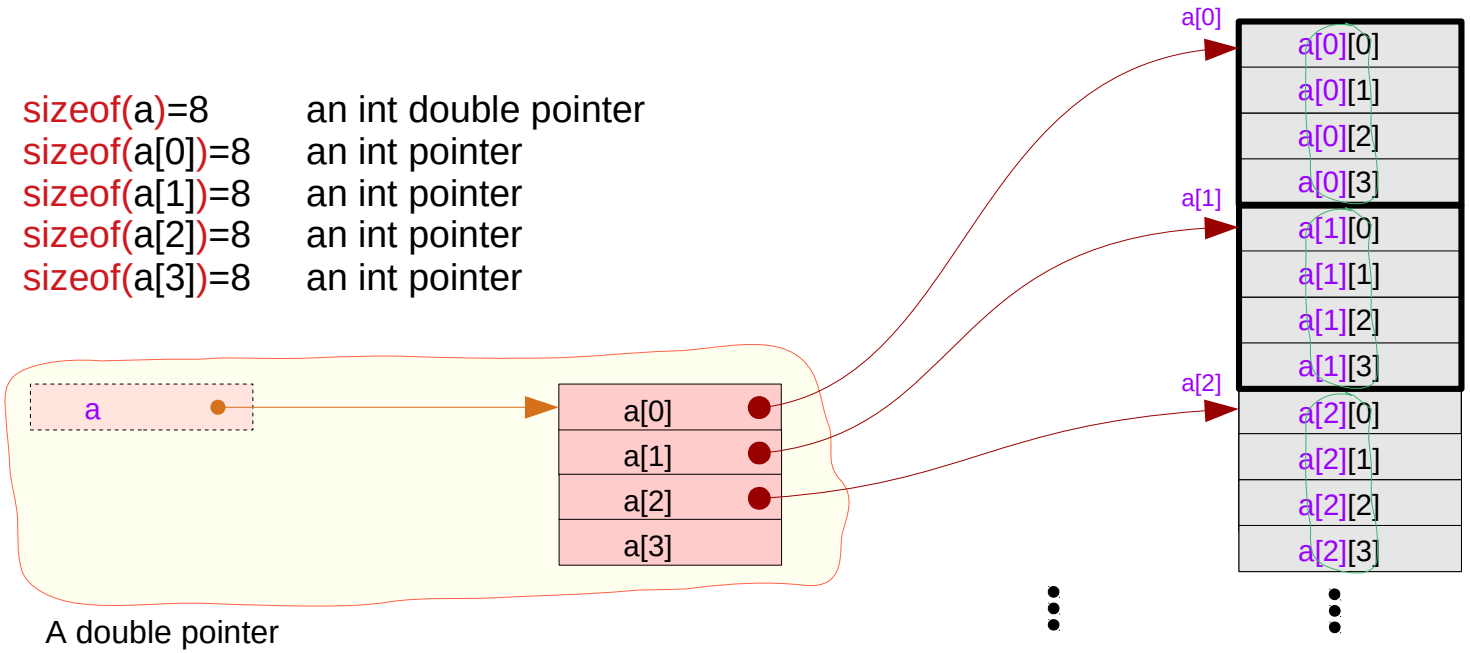
Case 2) an array of integer pointers

```
int *a[4];
for (i=0; i<4; i++)
    a[i] = (int *) malloc(4 * sizeof(int));          a[i][j] = i*4+j;
```



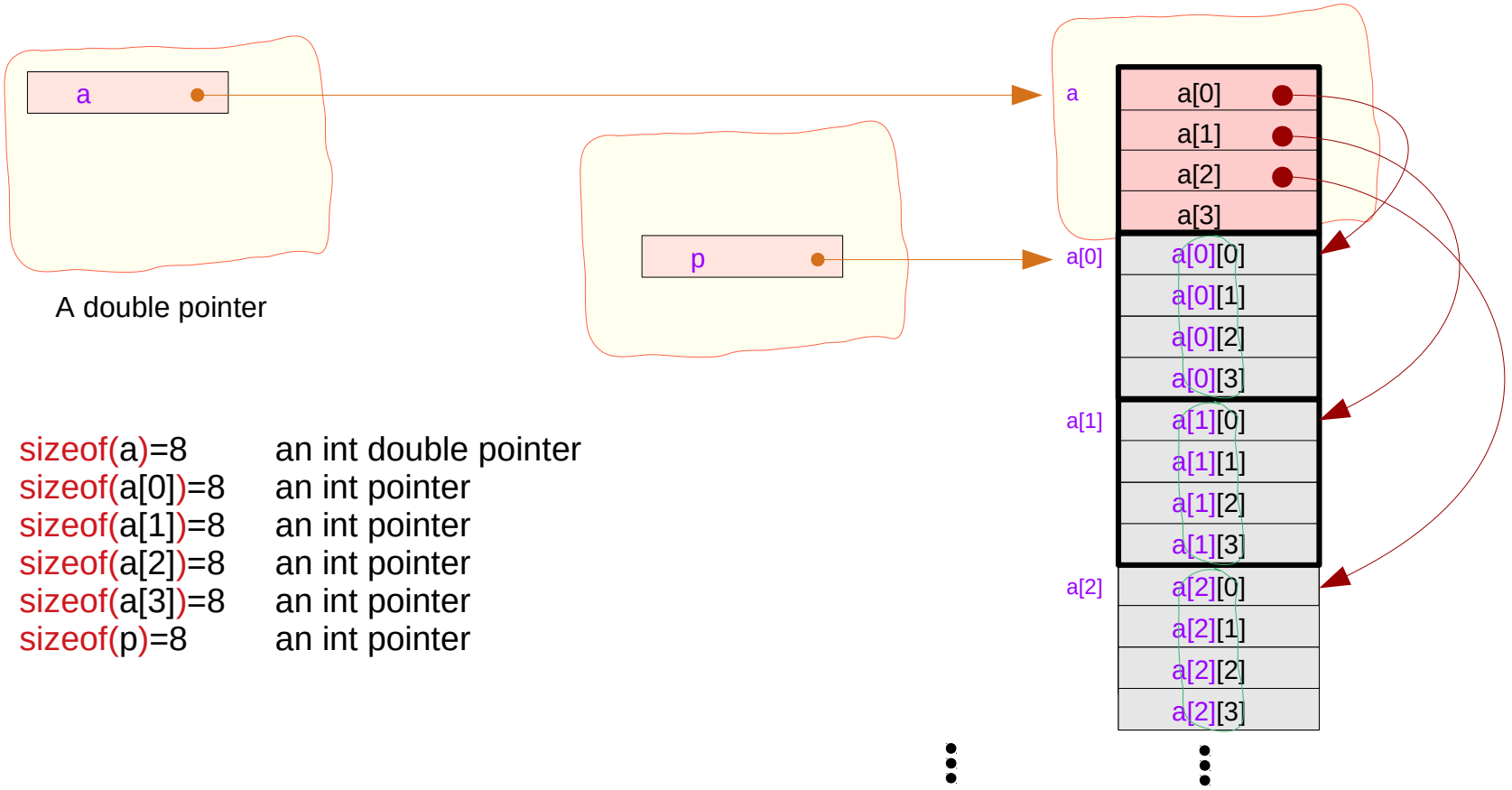
Case 3) an integer pointer array

```
int **a = (int **) malloc(4 * sizeof(int *));
for (i=0; i<4; ++i)
    a[i] = (int *) malloc(4 * sizeof(int));          a[i][j] = i*4+j;
```



Case 4) an integer pointer array

```
int siz = sizeof(int *) * 4 + sizeof(int) * 4 * 4;
int **a = (int **) malloc(siz);
int *p = (int *) a + 4*sizeof(int *) / sizeof(int);
for (i=0; i<4; i++)     a[i] = p + 4*i;                    a[i][j] = i*4+j;
```



sizeof(a)=8 an int double pointer
sizeof(a[0])=8 an int pointer
sizeof(a[1])=8 an int pointer
sizeof(a[2])=8 an int pointer
sizeof(a[3])=8 an int pointer
sizeof(p)=8 an int pointer

Dynamic memory allocation methods (II)

5) using a 1-d array pointer

```
int (*a)[4] = (int (*)[4]) malloc(4 * 4 * sizeof(int));          a[i][j] = i*4+j;
```

6) using a 2-d array pointer

```
int (*a)[4][4] = (int (*)[4][4]) malloc(4 * 4 * sizeof(int));    (*a)[i][j] = i*4+j;
```

7) using an array of 1-d array pointers

```
int (*a[4])[4];
for (i=0; i<4; ++i)
   a[i] = (int (*)[4]) malloc(4 * sizeof(int));                  (*a[i])[j] = i*4+j;
```

8) using a 2-d array of pointers

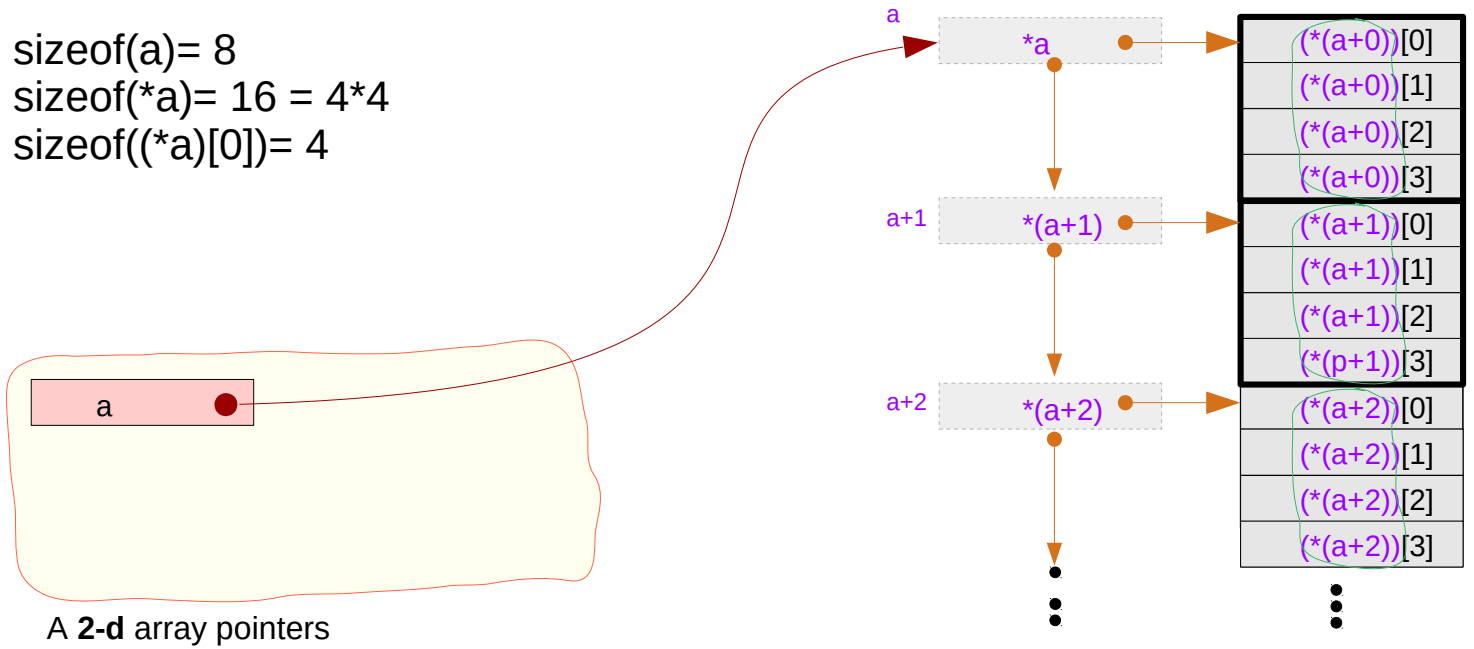
```
int *a[4][4];
a[0][0] = (int *) malloc(4 * 4 * sizeof(int));
for (i=0; i<4; ++i)
   for (j=0; j<4; ++j)
      a[i][j] = a[0][0] + (i*4 + j);                             *a[i][j] = i*4+j;
```

Case 5) a 1-d array pointer

```
int (*a)[4] = (int (*)[4]) malloc(4 * 4 * sizeof(int));
```

```
a[i][j] = i*4+j;
```

sizeof(a)= 8
sizeof(*a)= 16 = 4*4
sizeof((*a)[0])= 4

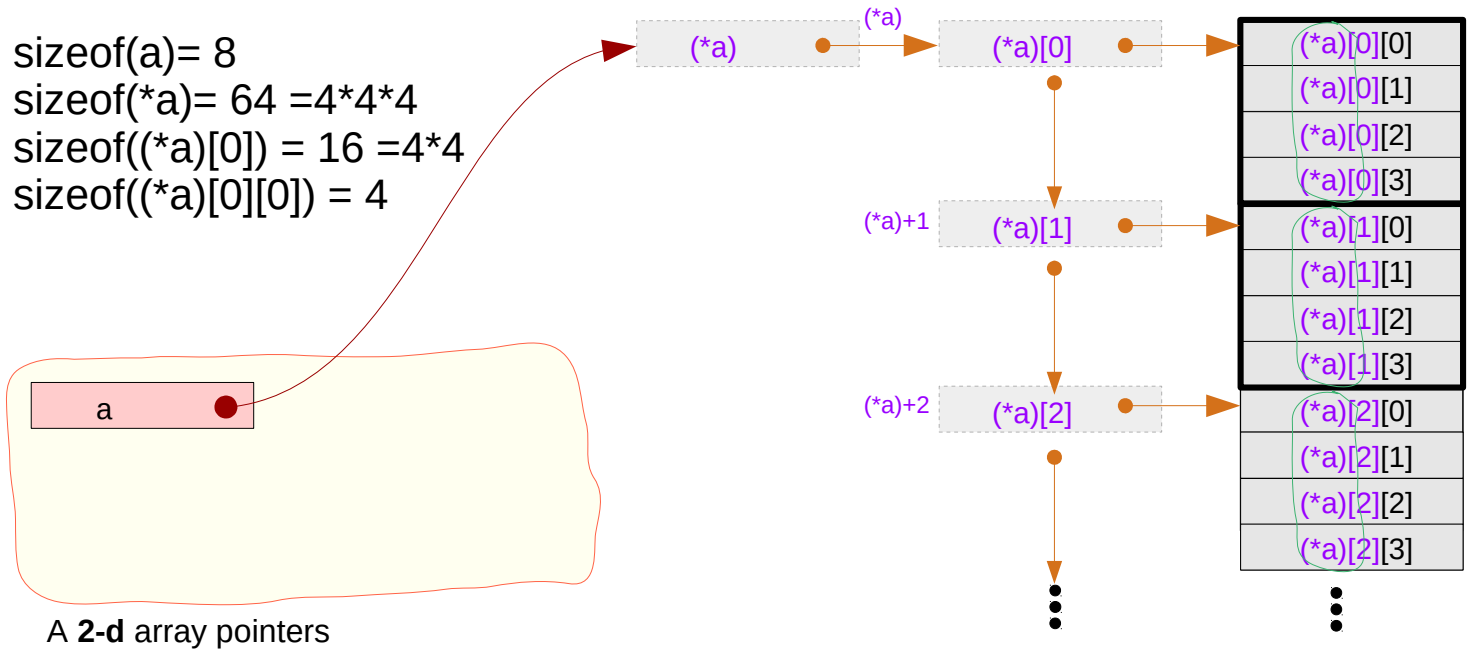


A **2-d** array pointers

Case 6) a 2-d array pointer

```
int (*a)[4][4] = (int (*)[4][4]) malloc(4 * 4 * sizeof(int));        (*a)[i][j] = i*4+j;
```

sizeof(a)= 8
sizeof(*a)= 64 =4*4*4
sizeof((*a)[0]) = 16 =4*4
sizeof((*a)[0][0]) = 4

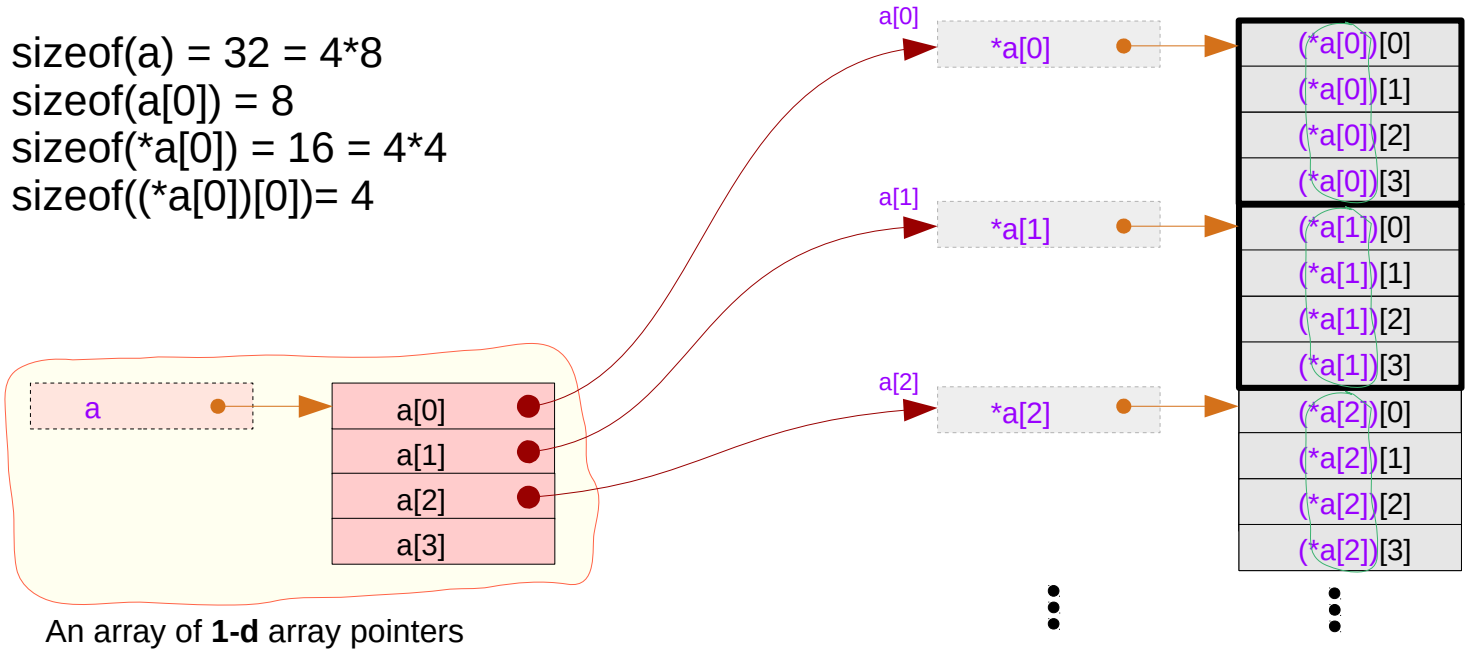


A **2-d** array pointers

Case 7) an array of 1-d array pointers

```
int (*a[4])[4];
for (i=0; i<4; ++i)
    a[i] = (int (*)[4]) malloc(4 * sizeof(int));          (*a[i])[j] = i*4+j;
```

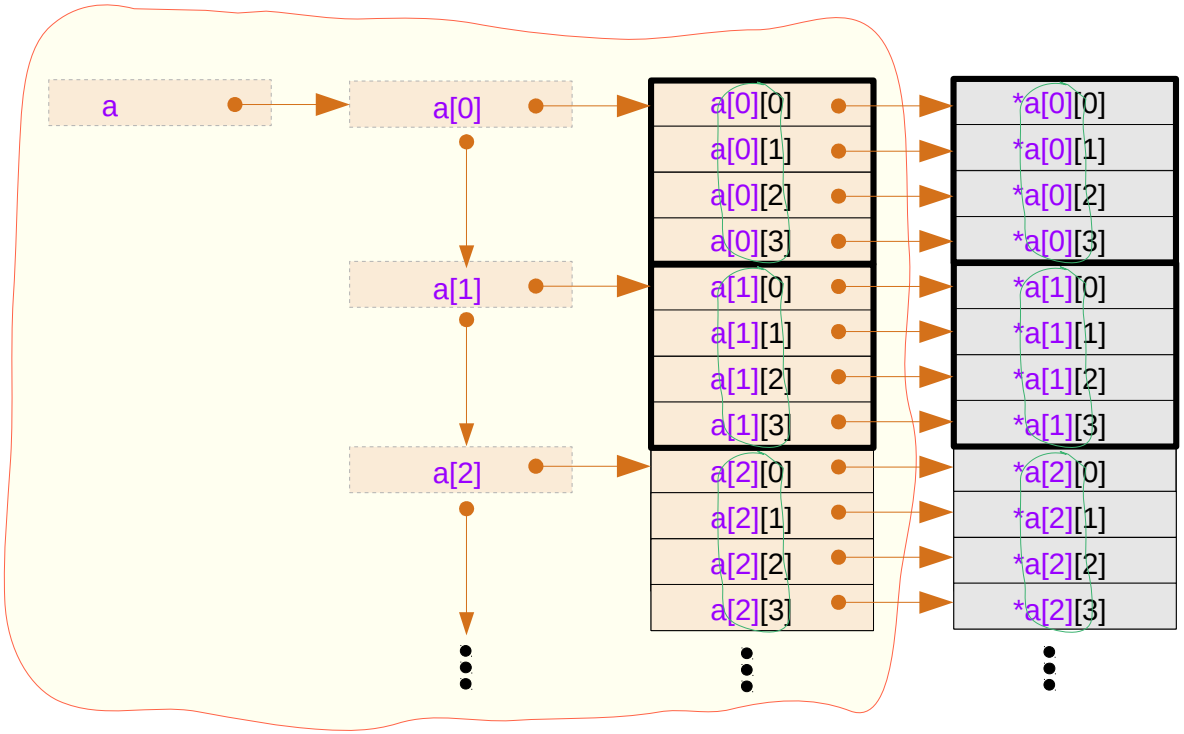
sizeof(a) = 32 = 4*8
sizeof(a[0]) = 8
sizeof(*a[0]) = 16 = 4*4
sizeof((*a[0])[0])= 4



An array of **1-d** array pointers

Case 8) a 2-d array of integers

```
int *a[4][4];
a[0][0] = (int *) malloc(4 * 4 * sizeof(int));
for (i=0; i<4; ++i)
   for (j=0; j<4; ++j)
      a[i][j] = a[0][0] + (i*4 + j);                    *a[i][j] = i*4+j;
```



sizeof(a) = 128 = 4*4*8
sizeof(a[0]) = 32 = 4*8
sizeof(a[0][0]) = 8
sizeof(*a[0][0])= 4

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun