

CSCI 2410 Systems Programming

Fall 2026

Instructor: Jie Zhou

Department of Computer Science
George Washington University



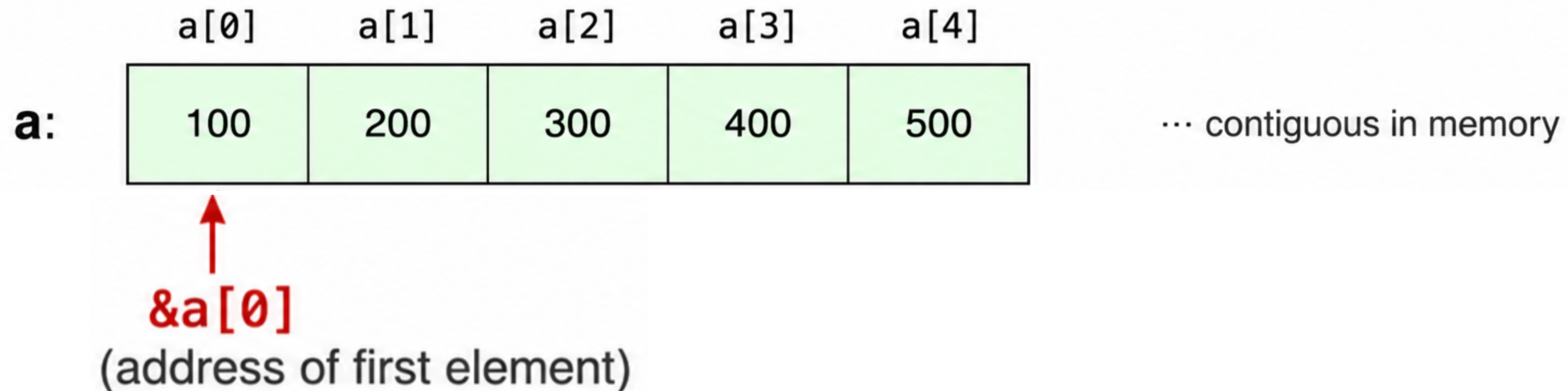
Slides materials are mostly credited to Gabriel Parmer and Sibin Mohan at GWU.

Memory Layout

Array Example

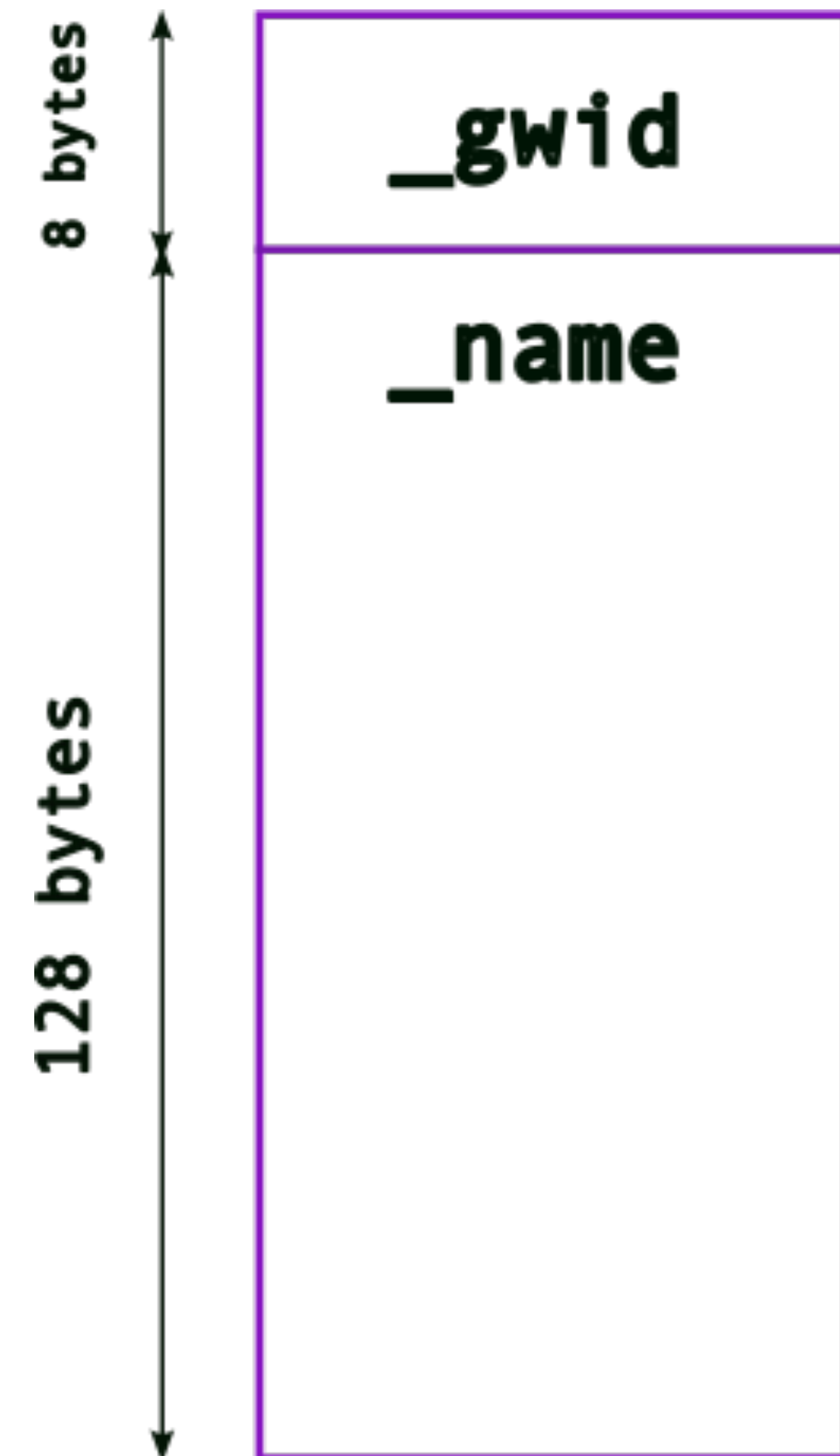
```
1  int a[5] = { 100, 200, 300, 400, 500 } ;
```

This is what it looks like in memory:



Memory Layout of Objects

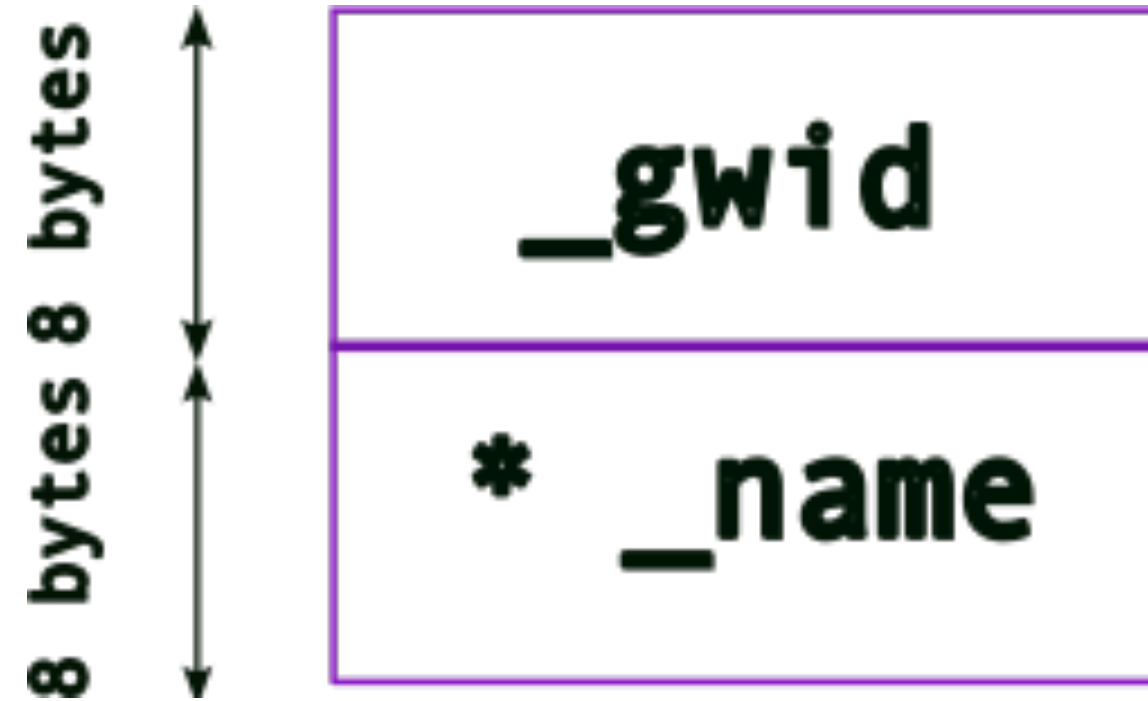
```
struct student{  
    unsigned long _gwid;  
    char _name [128];  
};
```



- `_name [128]` have a fixed size.
- This can be problematic.
 - Could be not long enough for certain names
 - May be too long for most names. Waste of memory!
 - If most names take up to 64 bytes, 100,000 records would waste $100,000 * 64 = 6,400,000$ 6.4 million bytes!

Dynamic Allocation

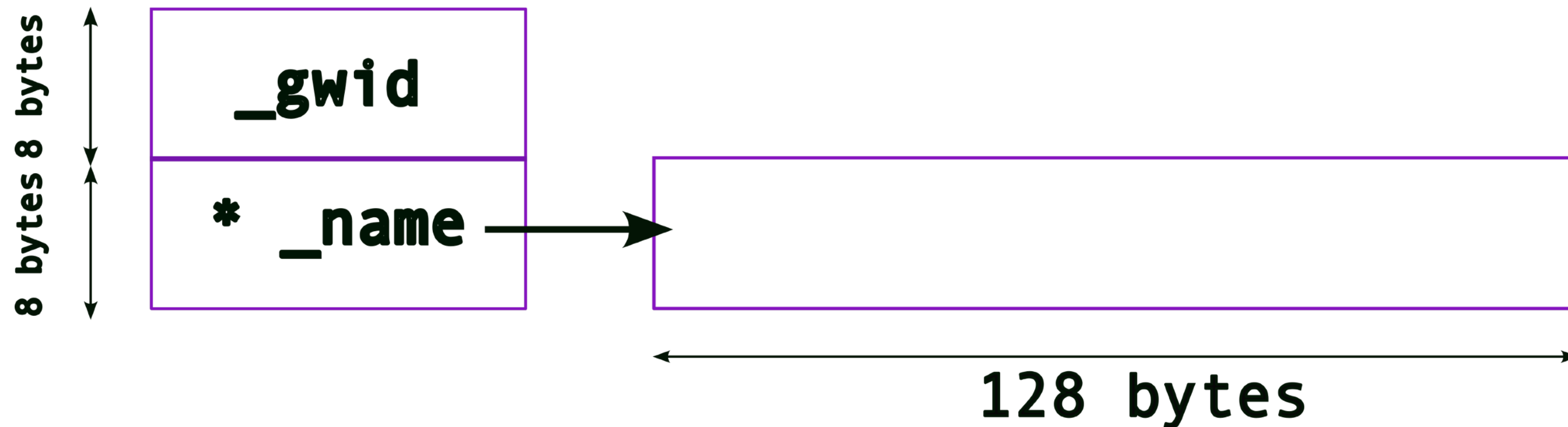
```
struct student{  
    unsigned long _gwid;  
    char *_name;  
};
```



- Student name now is a pointer.
- We must allocate memory to store names.

Dynamic Allocation

```
struct student{  
    unsigned long _gwid;  
    char *_name;  
};  
  
new_student._name = malloc(sizeof(char) * 128);
```



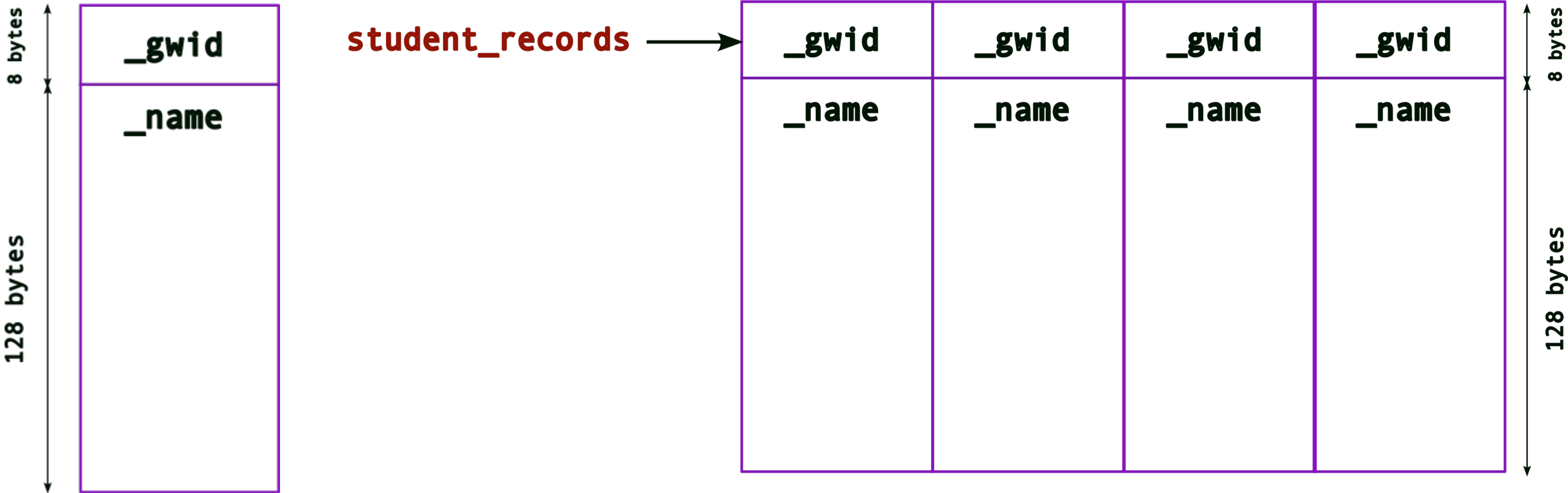
- `_name` now points to newly allocated memory.

What about an array of struct?

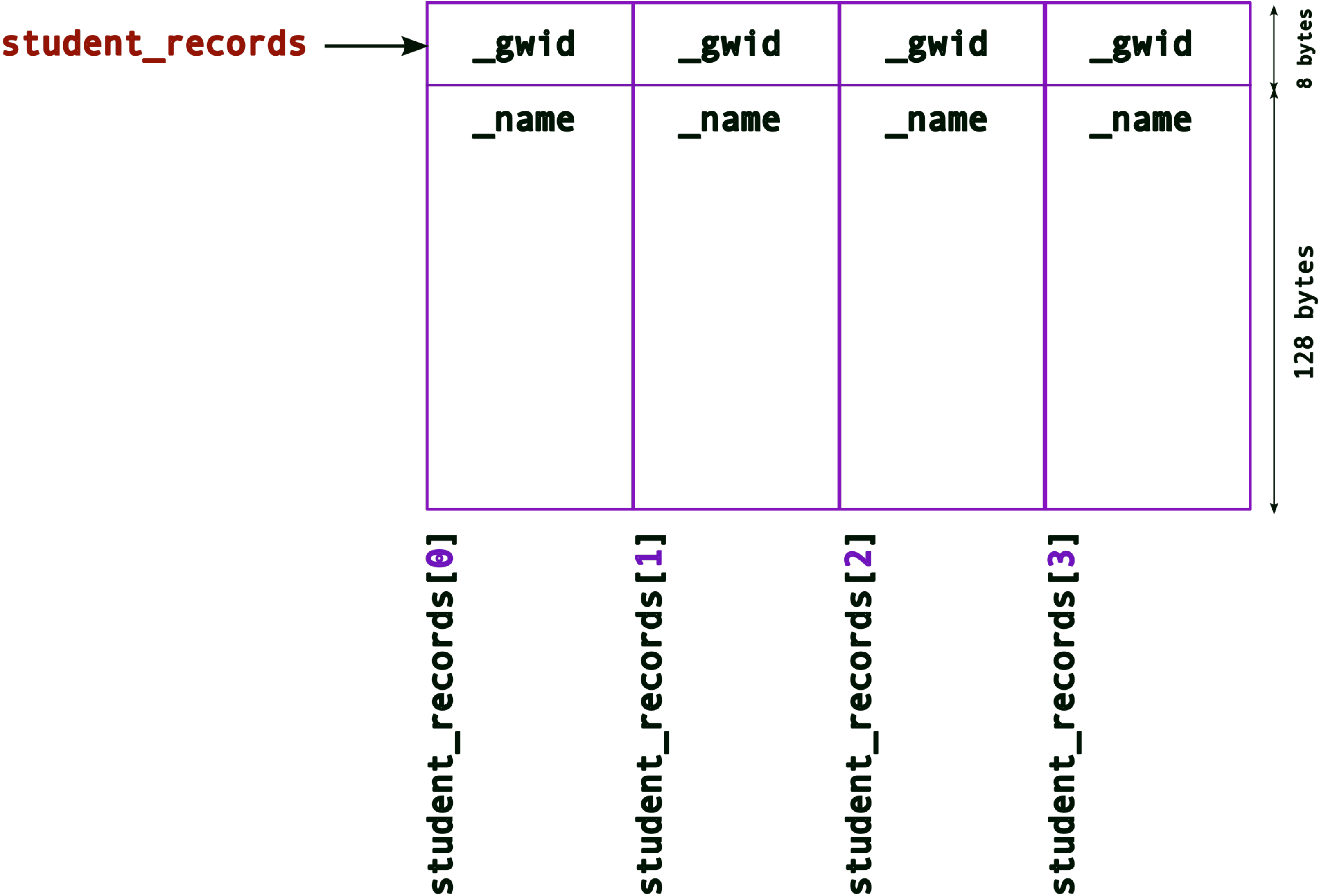
Array of Struct

```
struct student{
    unsigned long _gwid;
    char _name [128];
};

struct student student_records[4] ;
```

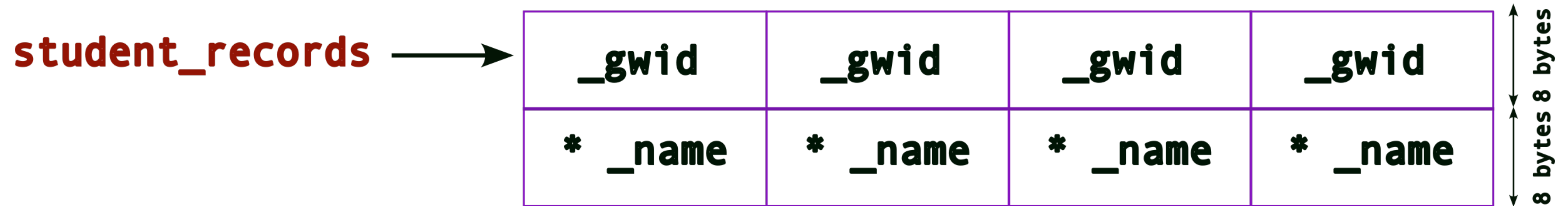


Array of Struct



Array of Struct (with Pointers)

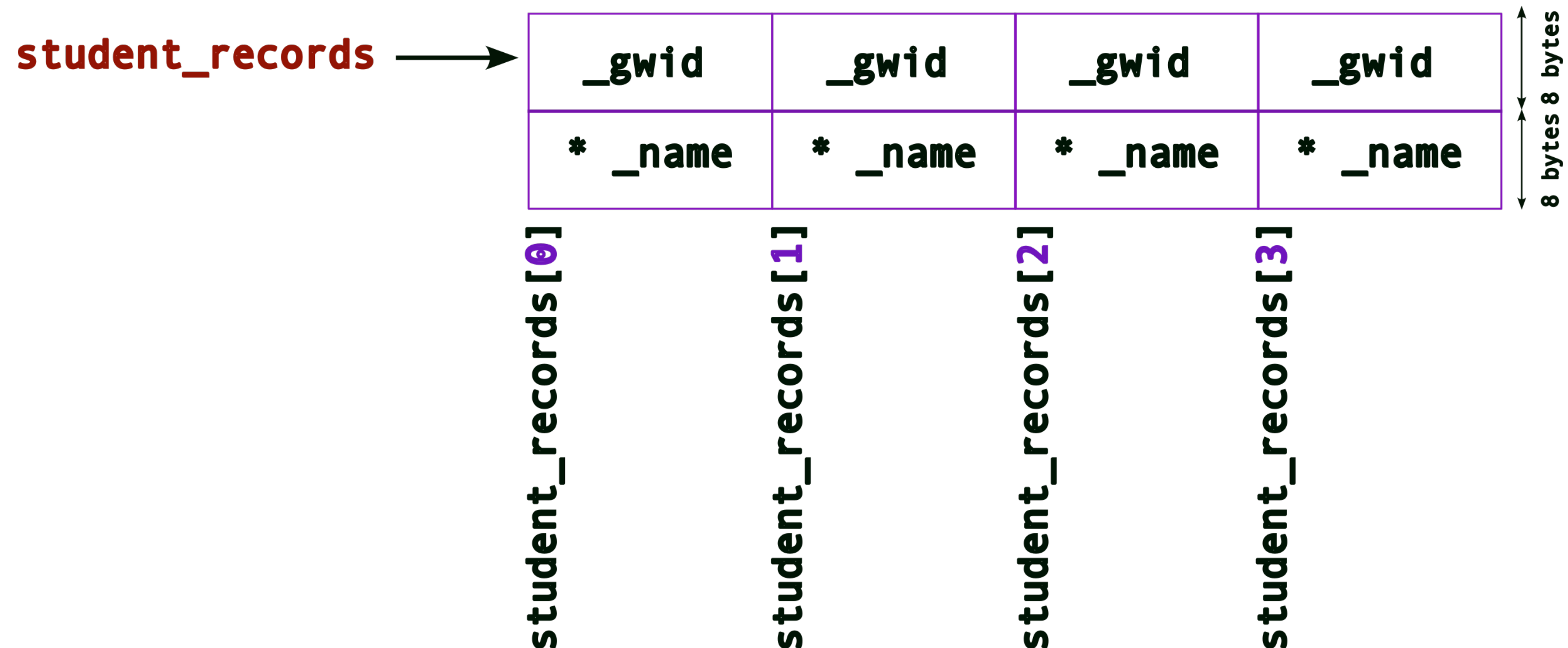
```
struct student{  
    unsigned long _gwid;  
    char *_name;  
};  
  
struct student student_records[4];
```



Array of Struct (with Pointers)

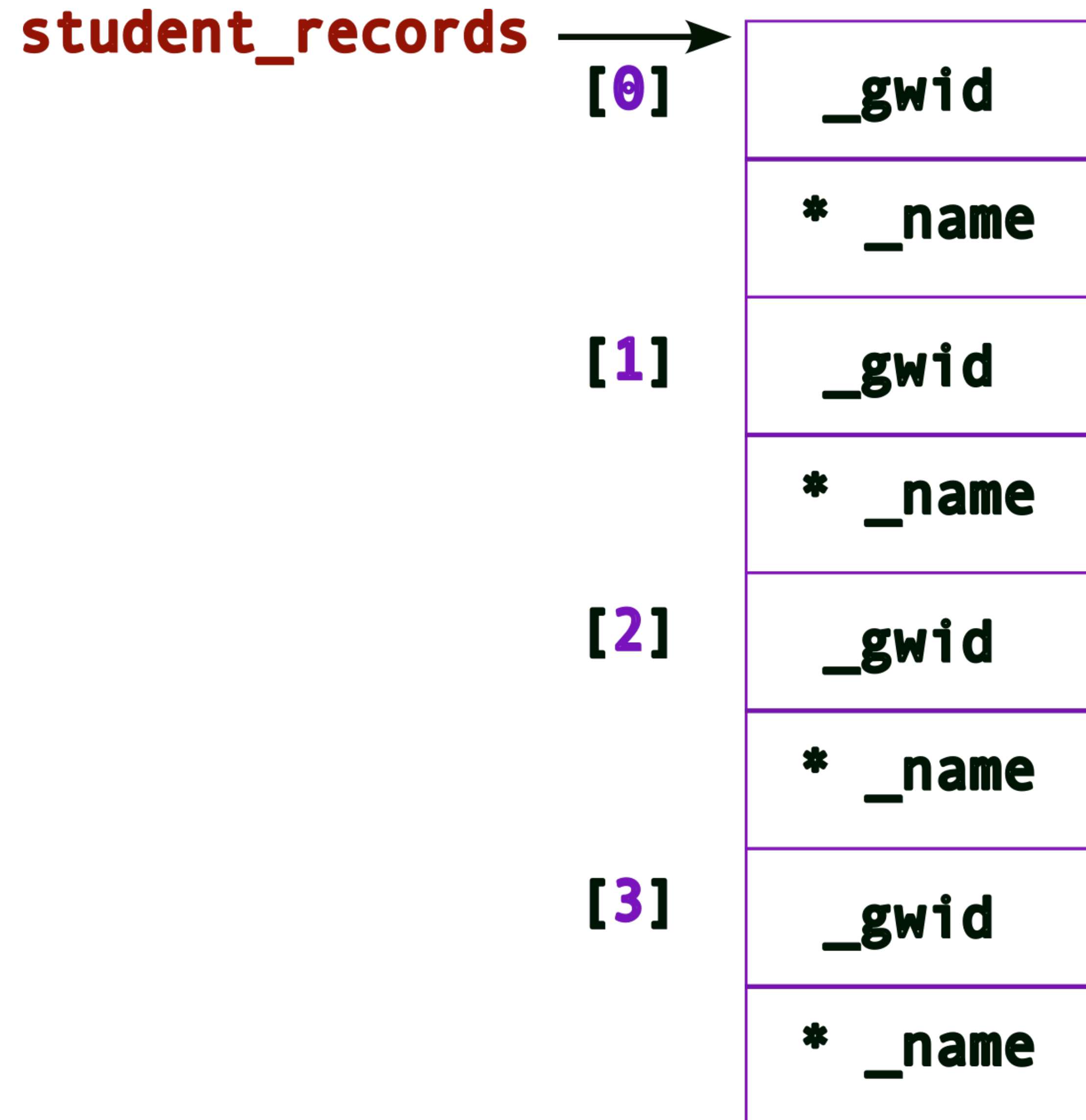
```
struct student{
    unsigned long _gwid;
    char *_name;
};

struct student student_records[4];
```



Array of Struct (with Pointers)

```
struct student{  
    unsigned long _gwid;  
    char *_name;  
};  
  
struct student student_records[4];
```

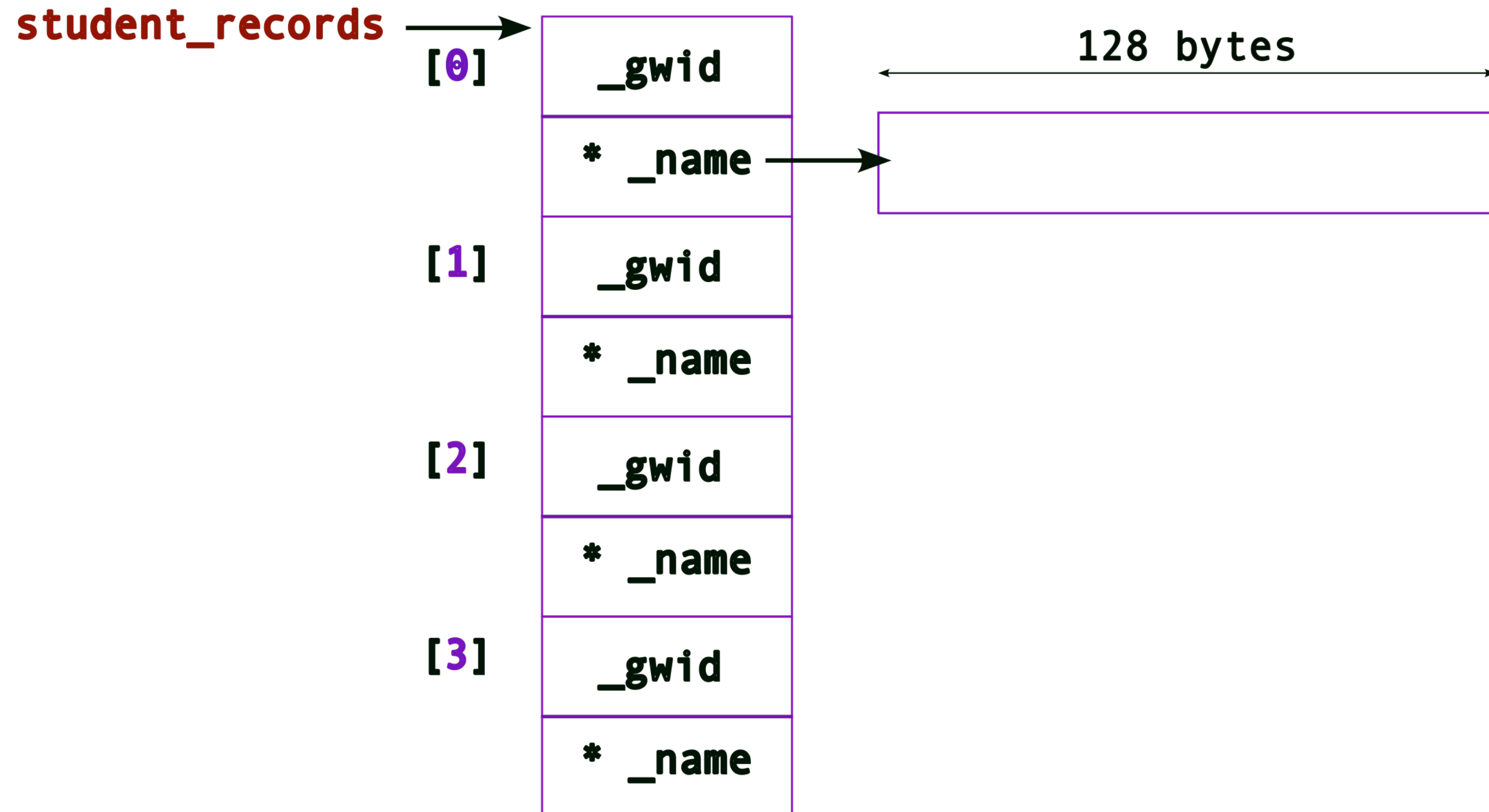


Allocate Memory for Student Name

```
struct student{
    unsigned long _gwid;
    char *_name;
};

struct student student_records[4];
student_records[0]._name = malloc(sizeof(char) * 128);
```

Allocate Memory for Student Name



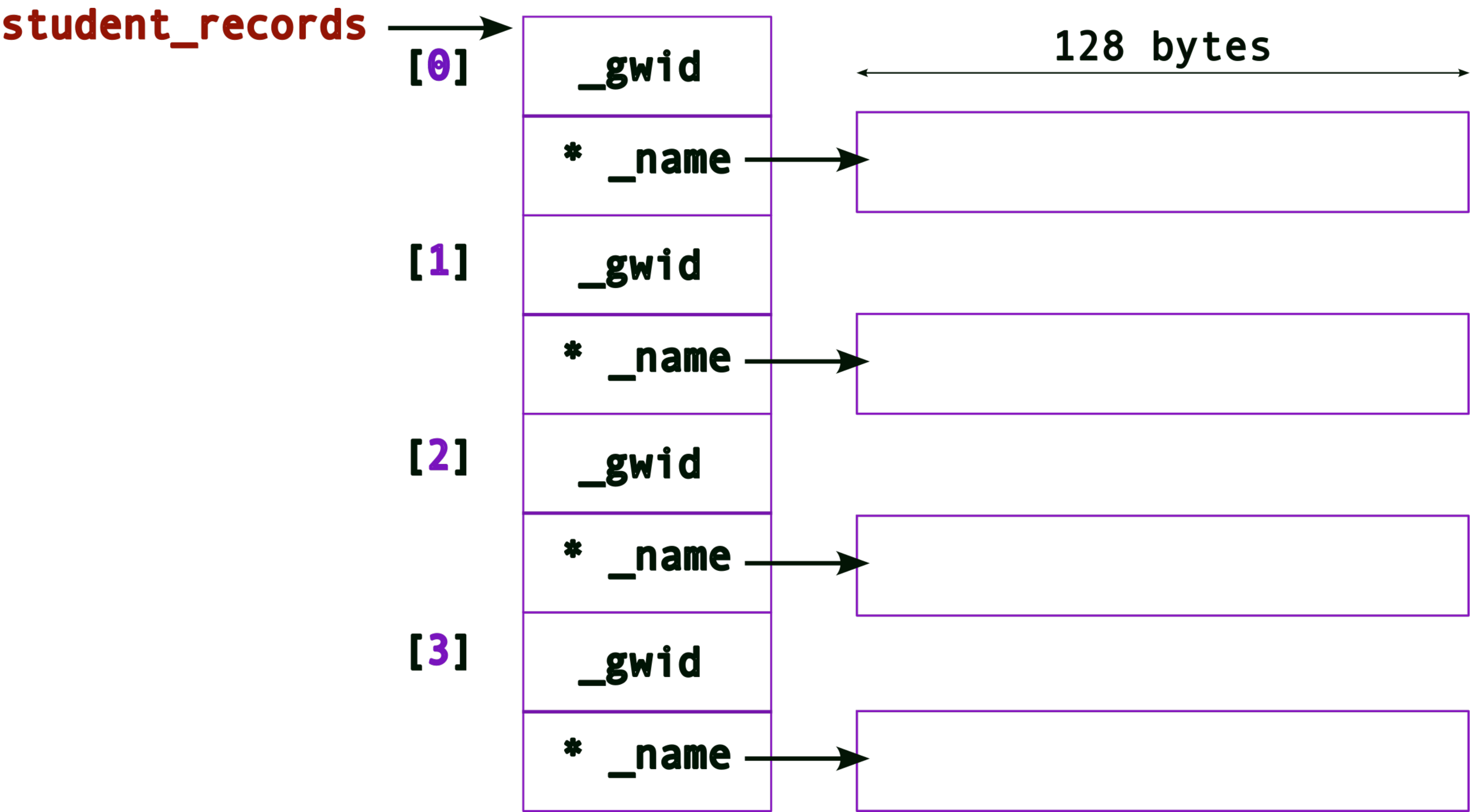
Allocate Memory for Student Name

```
struct student{
    unsigned long _gwid;
    char *_name;
};

struct student student_records[4];

student_records[0]._name = malloc(sizeof(char) * 128);
student_records[1]._name = malloc(sizeof(char) * 128);
student_records[2]._name = malloc(sizeof(char) * 128);
student_records[3]._name = malloc(sizeof(char) * 128);
```

Allocate Memory for Student Name



What does this do?

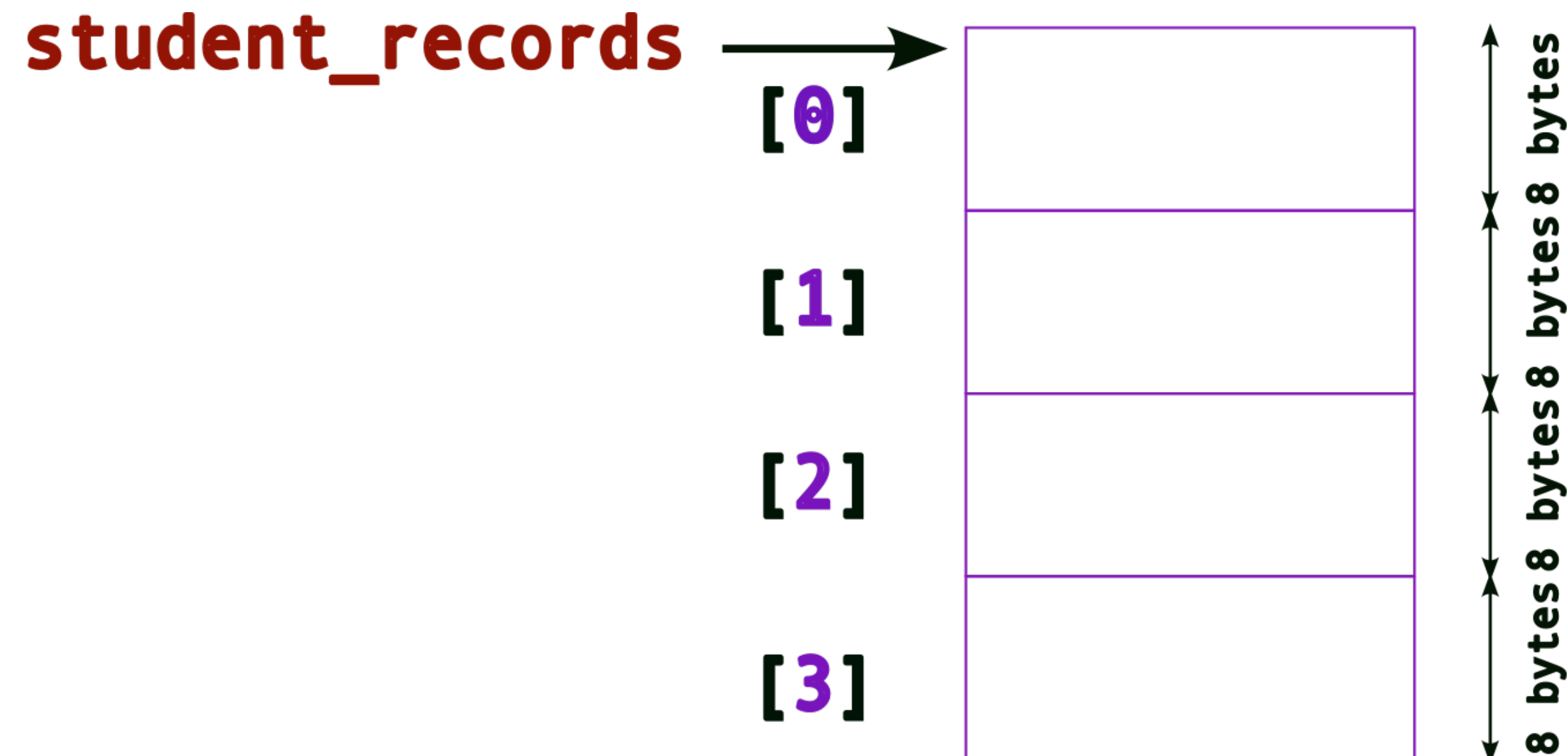
```
struct student{  
    unsigned long _gwid;  
    char *_name;  
};  
  
struct student *student_records[4];
```

Array of Pointers

```
struct student{
    unsigned long _gwid;
    char *_name;
};

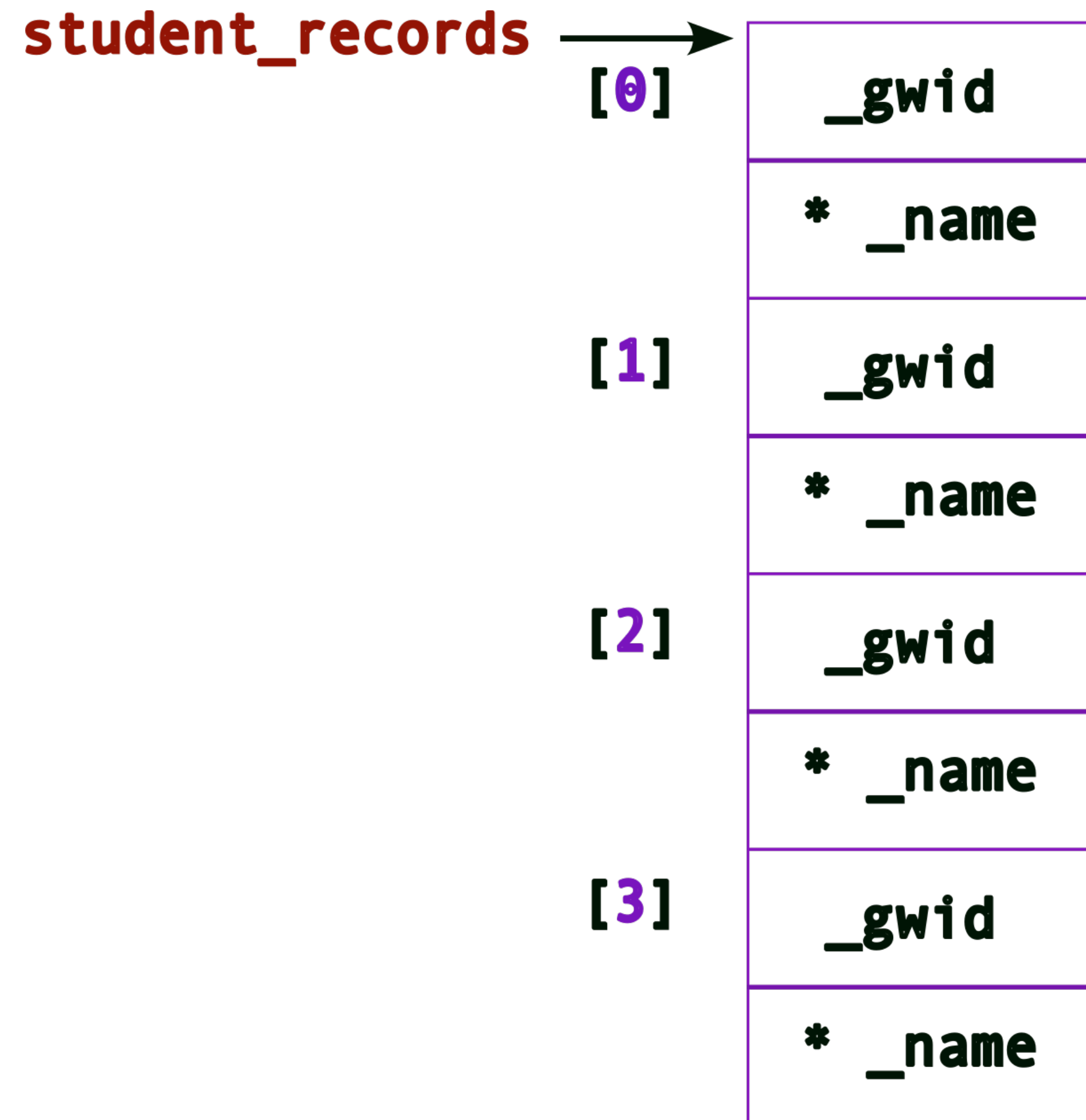
struct student *student_records[4];
```

- Creates an array of pointers to struct student

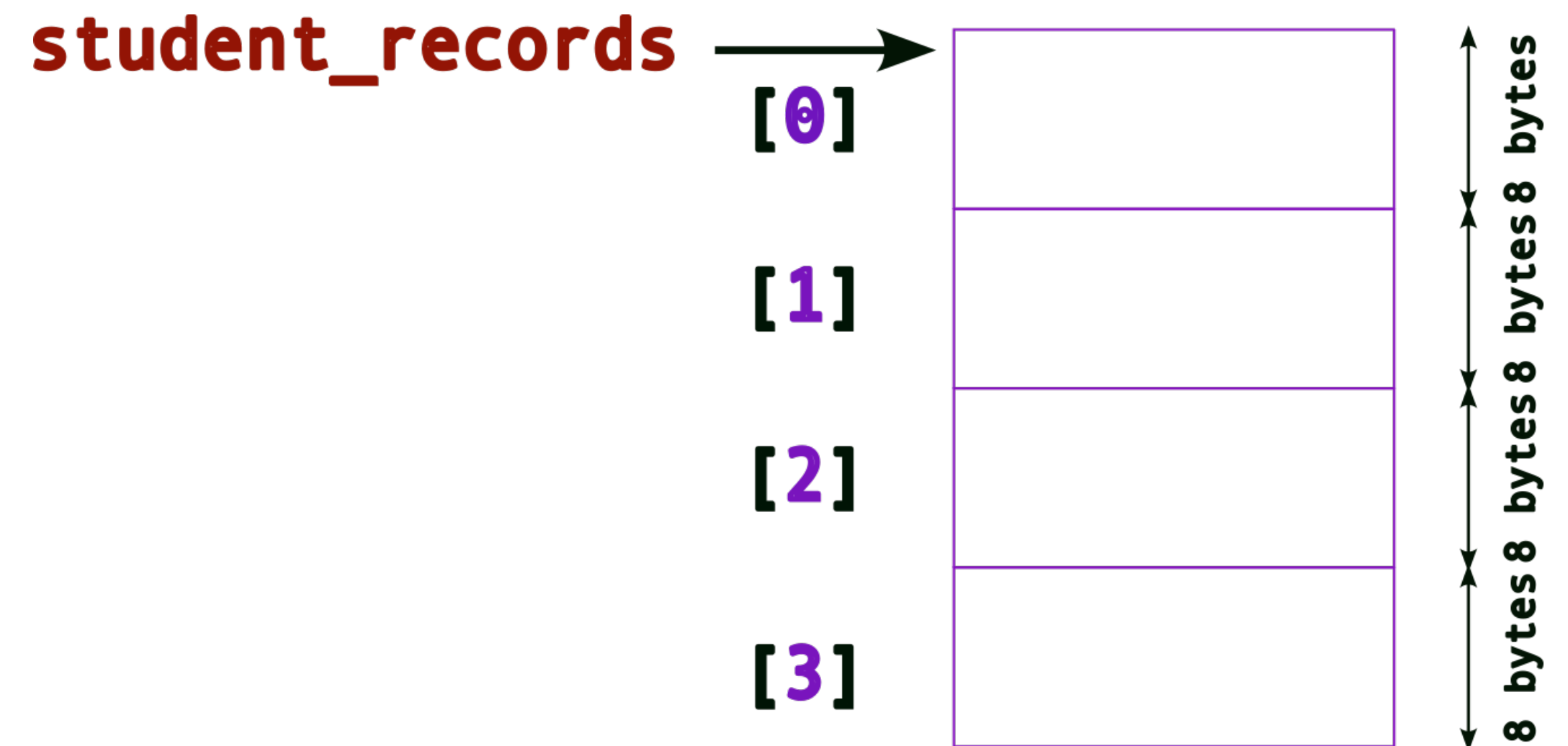


Array of Struct vs Array of Pointers

```
struct student student_records[4];
```



```
struct student *student_records[4];
```

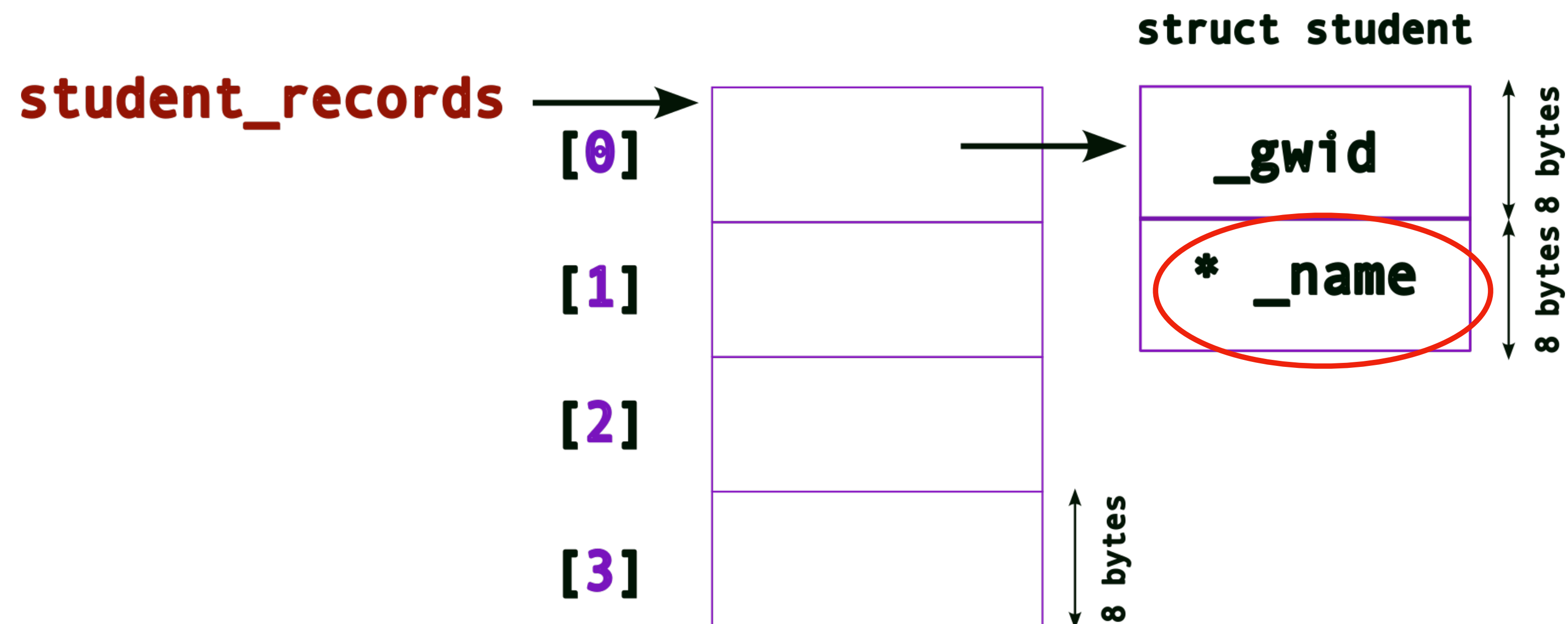


Allocate Struct

```
struct student{
    unsigned long _gwid;
    char *_name;
};

struct student *student_records[4];
// allocate memory for a struct first!
student_records[0] = malloc(sizeof(struct student));
```

What are we still missing?



Allocate Memory for Student Name

```
struct student{
    unsigned long _gwid;
    char *_name;
};

struct student *student_records[4];
// allocate memory for a struct first!
student_records[0] = malloc(sizeof(struct student));
// allocate memory for name, note the -> operator
student_records[0]->_name = malloc(sizeof(char) * 128);
```

→ operator

- Using pointers to access members of a struct or union

Variable type

Access examples

normal variable
struct student alice;

```
alice->_name = malloc(...);  
printf("name = %s\n", alice->_name);
```

pointer variable:
struct student *p_alice;

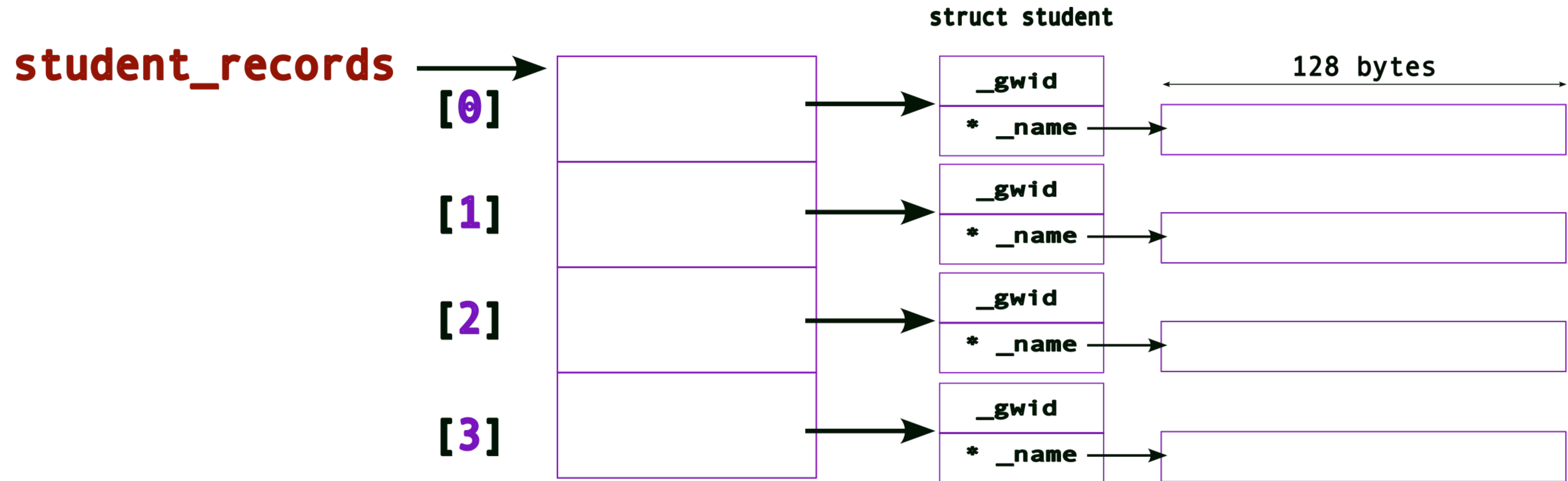
```
alice->_name = malloc(...);  
printf("gwid = %ul\n", alice->_gwid);
```

Allocate Memory for All

```
struct student{
    unsigned long _gwid;
    char *_name;
};

struct student *student_records[4];
// allocate memory for a struct first!
student_records[0] = malloc(sizeof(struct student));
// allocate memory for name, note the -> operator
student_records[0]->_name = malloc(sizeof(char) * 128);
student_records[1] = malloc(sizeof(struct student));
student_records[1]->_name = malloc(sizeof(char) * 128);
student_records[2] = malloc(sizeof(struct student));
student_records[2]->name = malloc(sizeof(char) * 128);
student_records[3] = malloc(sizeof(struct student));
student_records[3]->_name = malloc(sizeof(char) * 128);
```

Final Memory Layout



- Now we have pointers to pointers, i.e., double/multi-level pointers!