Creating Objects

Lecture 4

Robb T. Koether

Hampden-Sydney College

Mon, Sep 2, 2019

Outline

Outline

Drawing a Rectangle

- In earlier versions of OpenGL, drawing a rectangle was quite simple.
  - Announce that you were going to draw a rectangle:

    ```
    glBegin(GL_RECT);
    ```

  - Pass the vertices one by one:

    ```
    glVertex2f(0.0, 1.0)
    ```

    Etc.
- It is a bit more complicated now.

Drawing a Rectangle

- The three basic steps are
  - Create an array of vertex attributes (data).
  - Create a vertex buffer object (in the GPU).
  - Create a vertex array object (structures the buffer).
  - Issue the draw command.

Outline

Vertex Attributes

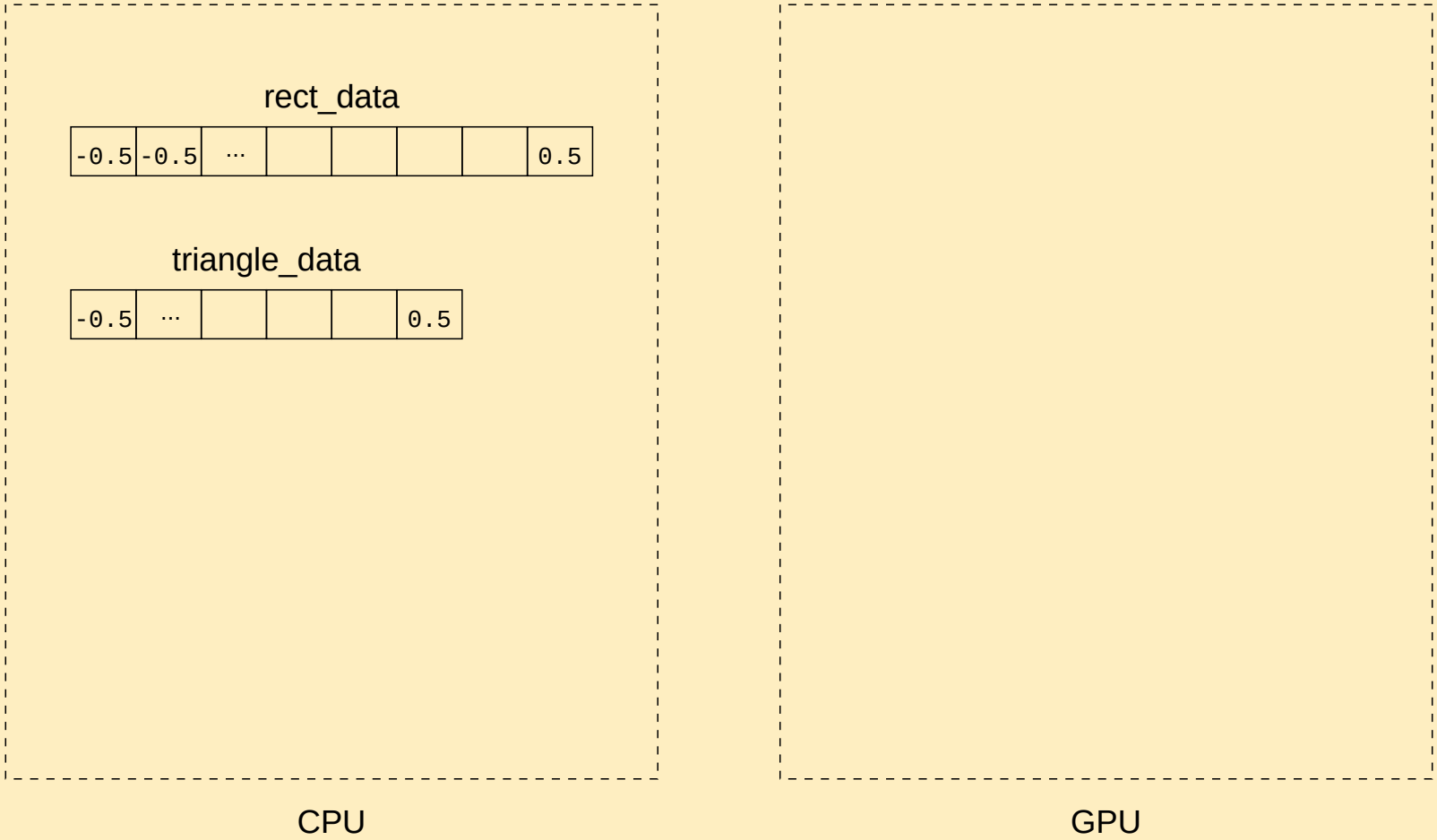
Vertex Attributes

```
GLfloat rect_data[] =
{
    -0.5f, -0.5f,
     0.5f, -0.5f,
     0.5f,  0.5f,
    -0.5f,  0.5f
};
GLfloat triangle_data[] = {...};
```

- In this first example, the only vertex attributes will be the coordinates of the 2D vertices.

Vertex Buffer Objects

Vertex Buffer Object



Outline

Vertex Buffer Objects

- A vertex buffer object (VBO) is a buffer (memory) in the GPU that contains data related to the vertices of an object.
  - Coordinates of the vertices.
  - Their color.
  - Normal vectors.
  - Etc.

Vertex Buffer Objects

- To use a VBO, we must do three things.
  - Generate a name (ID number) for the buffer object.
  - “Bind” a buffer object to the name, i.e., associate the ID number with the buffer object and make it the current (or active) buffer.
  - Copy the vertex data to the buffer object.

Vertex Buffer Objects

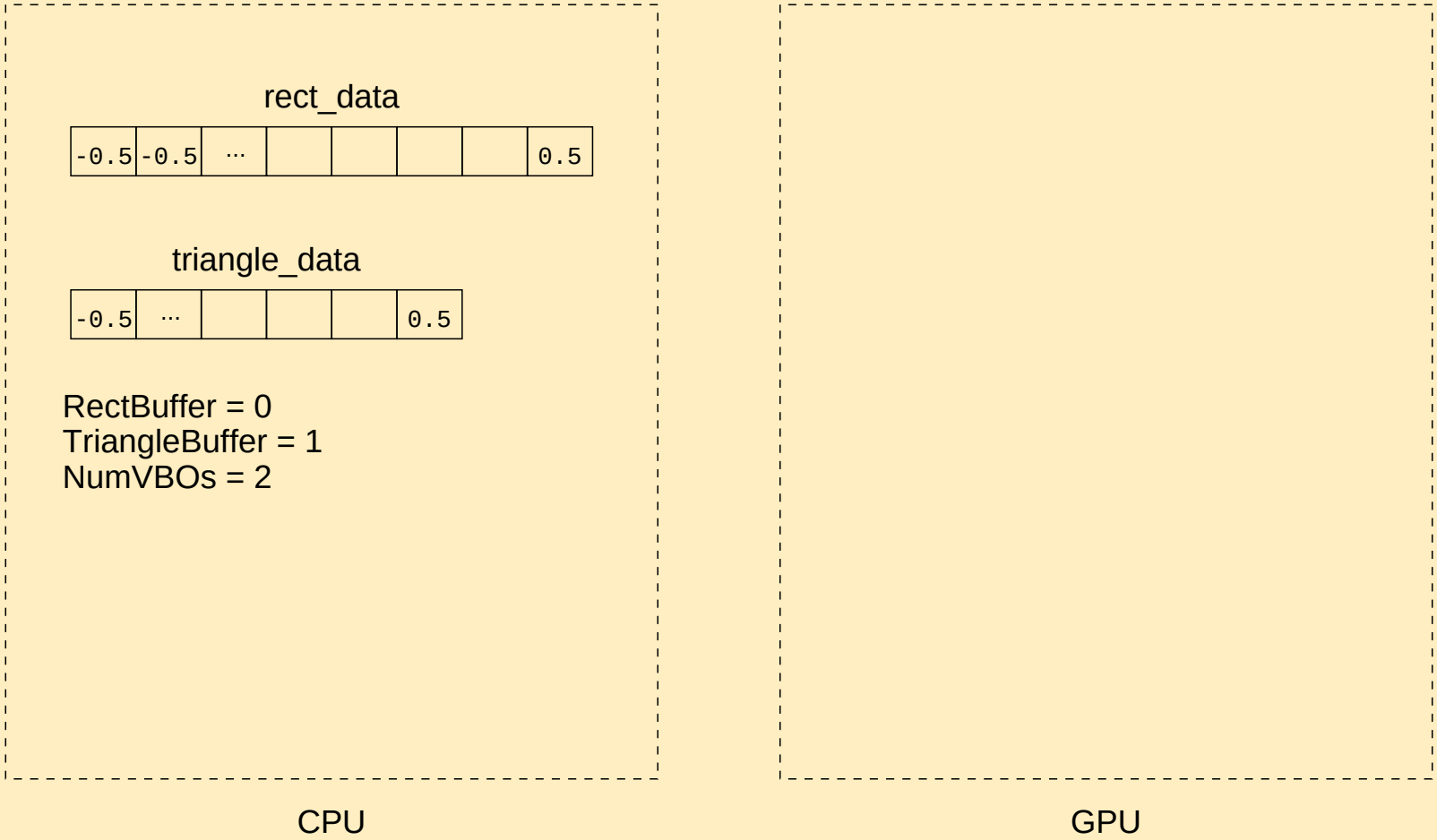
Symbolic Names for the VBOs

```
enum {RectBuffer, TriangleBuffer, NumVBOs};
```

- The **enum** statement will assign the values `0`, `1`, and `2` to `RectBuffer`, `TriangleBuffer`, and `NumVBOs`, respectively.
- Note that value of `numVBOs` will automatically be the number of buffer objects.

Vertex Buffer Objects

Vertex Buffer Object



Vertex Buffer Objects

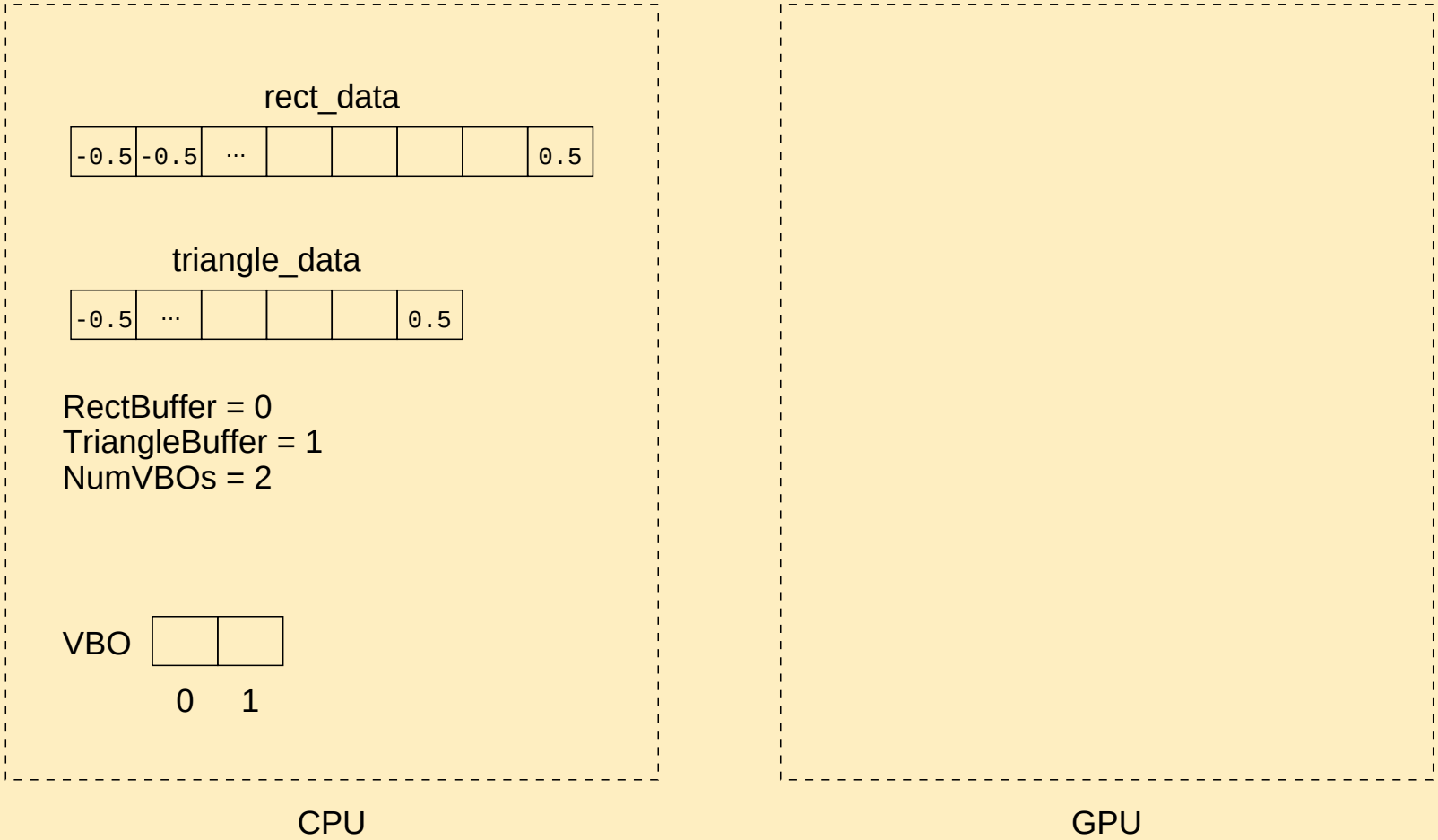
Array of VBO IDs

```
GLuint VBO[NumVBOs];
```

- The array `VBO` will contain the ID numbers (to be assigned by OpenGL) of the buffer objects.
- The **`enums`** `RectBuffer` and `TriangleBuffer` are symbolic names for the indexes of the IDs in the array `VBO`.

Vertex Buffer Objects

Vertex Buffer Object



Vertex Buffer Objects

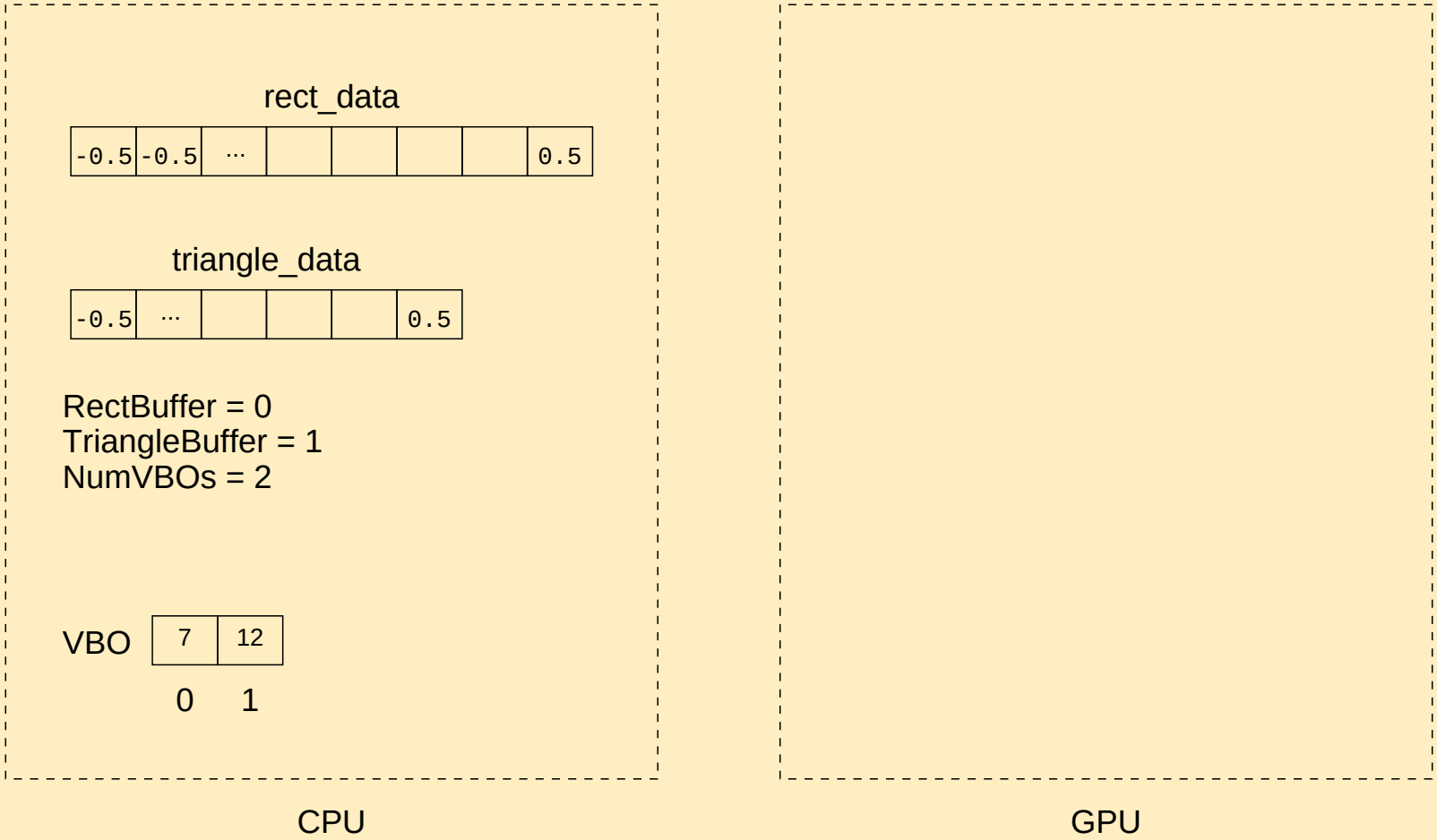
Vertex Buffer Object

```
glGenBuffers(NumVBOs, VBO);
```

- Generate ID numbers for each of the buffers and store them in `VBO[0]` and `VBO[1]`, also known as `VBO[RectBuffer]` and `VBO[TriangleBuffer]`.

Vertex Buffer Objects

Vertex Buffer Object



Vertex Buffer Objects

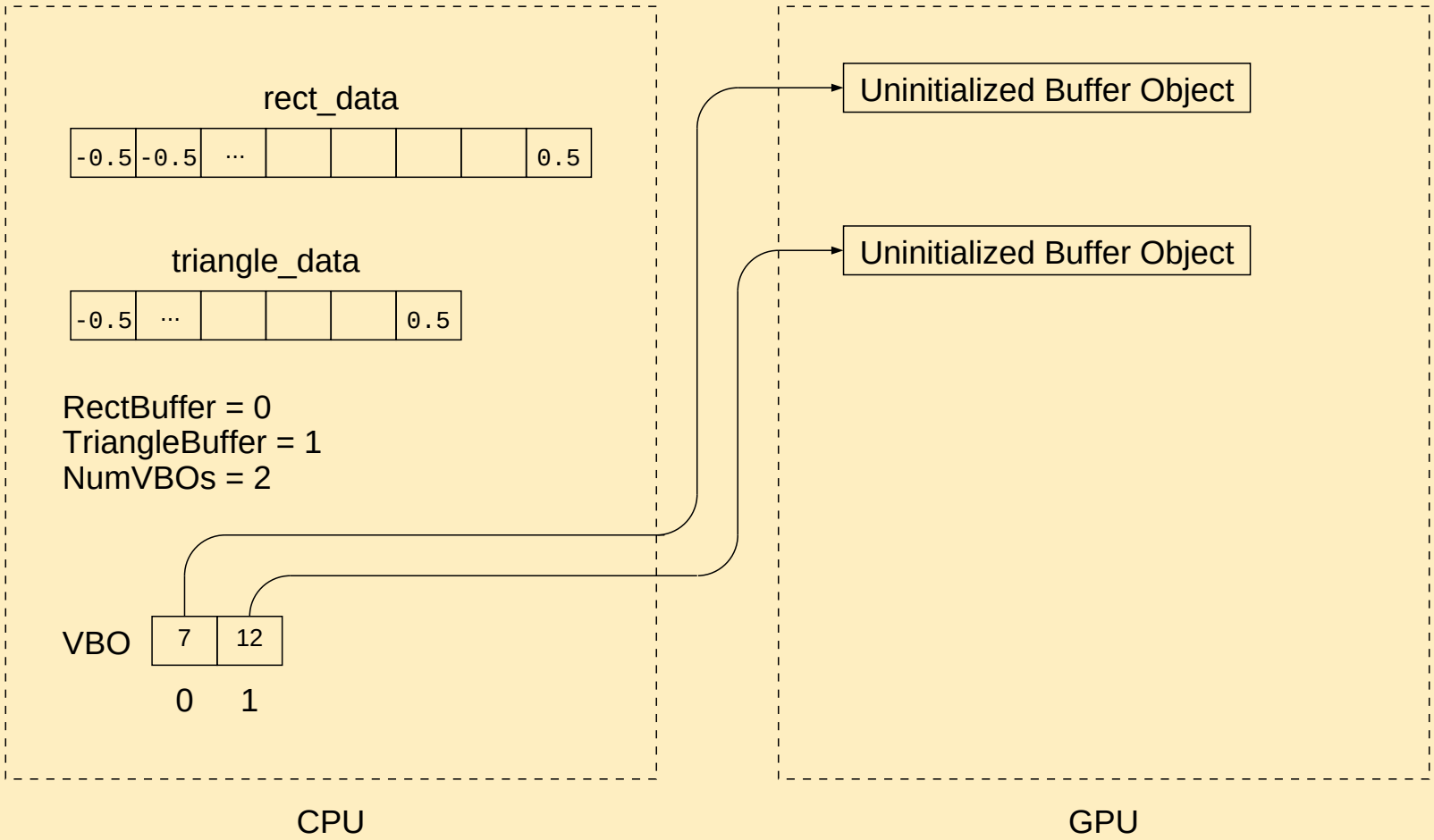
Vertex Buffer Object

```
glBindBuffer(GL_ARRAY_BUFFER, VBO[RectBuffer]);
```

- `glBindBuffer()` binds (associates) the buffer ID `VBO[RectBuffer]` to a new buffer object in the GPU and makes that buffer object the current buffer.
- When `glBindBuffer()` is called subsequently with the same buffer ID, it simply makes that buffer object the current one.

Vertex Buffer Objects

Vertex Buffer Object



Vertex Buffer Objects

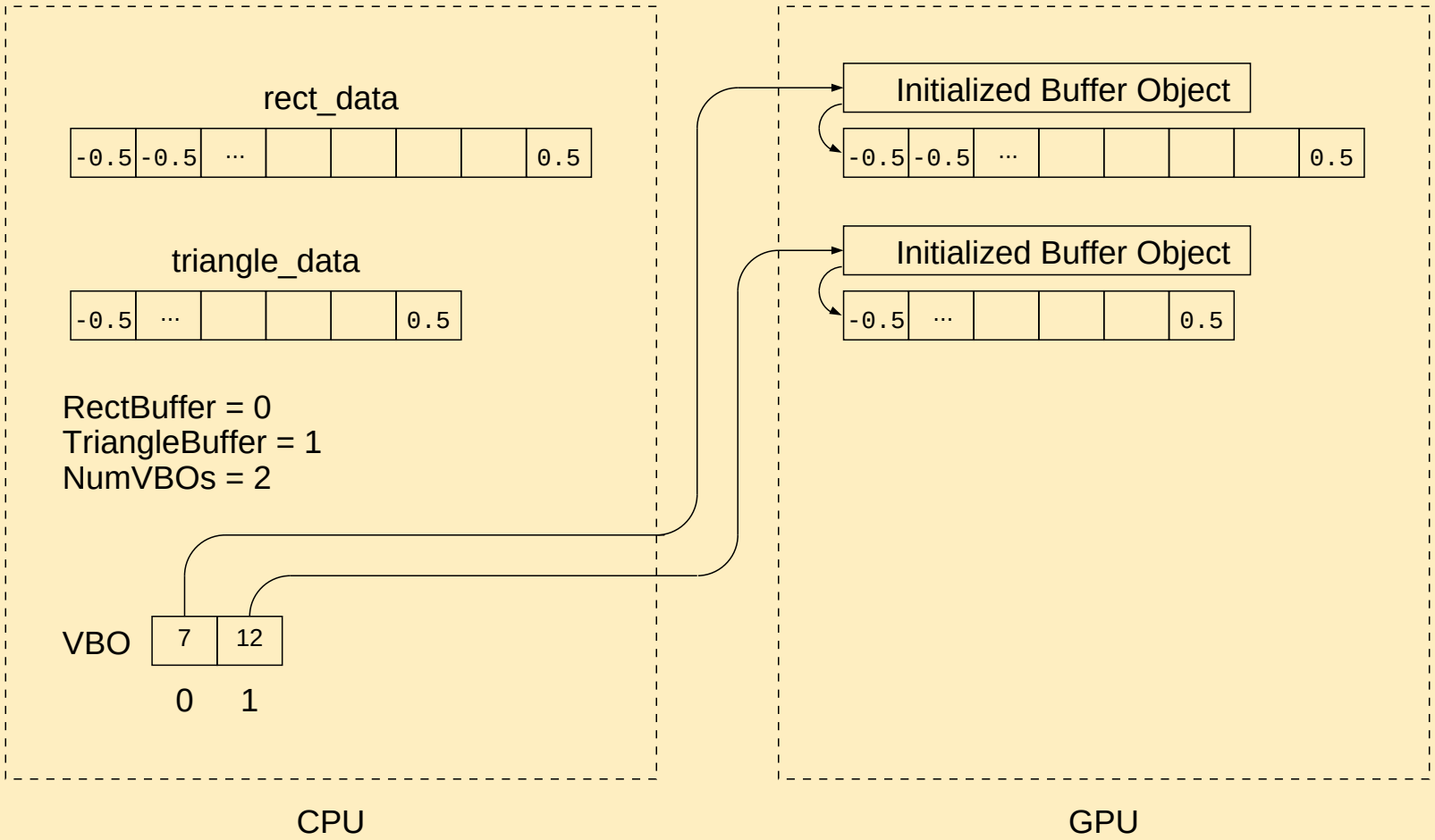
Vertex Buffer Object

```
glNamedBufferStorage(VBO[RectBuffer],
    sizeof(rect_data), rect_data, 0);
```

- `glNamedBufferStorage()` copies the data from `rect_data` into the named buffer (`VBO[RectBuffer]`).

Vertex Buffer Objects

Vertex Buffer Object



Outline

Vertex Array Objects

- A vertex array object (VAO) describes the structure imposed on the data stored in the buffer object.
- We follow a similar pattern with VAOs as we did with VBOs.
- To use a VAO, we must do three things.
  - Generate an ID number for the vertex array object.
  - “Bind” that vertex array object to the active buffer object.
  - Describe the structure (i.e., attributes) of the data in the buffer.
  - Enable the vertex attributes.
- Then we are ready to draw the object.

Vertex Array Objects

Symbolic Names for the VAOs

```
enum {Rect, Triangle, NumVAOs};
enum {vPosition = 0};
```

- We use an enumerated type to create symbolic names for the VAOs.
- We also use an enumerated type to create symbolic names for the vertex attributes.
- In this example, the only attribute is the position.

Vertex Array Objects

Array of VAO IDs

```
GLuint VAO[NumVAOs];
```

- Create an array of vertex array objects.
- As with the VBOs, this array will hold the ID number of the VAOs in the GPU.

Vertex Array Objects

Vertex Array Object

```
glBindVertexArray(VAO[Rect]);
```

- `glBindVertexArray()` will create vertex array objects in the GPU and store their IDs in the `VAO` array.
- This statement will store the ID for the rectangle VBO in `VAO[0]`.
- It is necessary that `VBO[RectBuffer]` be the current VBO.

Vertex Array Objects

Vertex Array Object

```
glVertexAttribPointer(vPosition, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(0));
```

- This statement associates the attribute ID `vPosition` (i.e., `0`) with the following information.

Vertex Array Objects

Vertex Array Object

```
glVertexAttribPointer(vPosition, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(0));
```

- This statement associates the attribute ID `vPosition` (i.e., `0`) with the following information.
  - The `2` indicates the number of objects that constitute a single attribute (2 floats = a 2D point).

Vertex Array Objects

Vertex Array Object

```
glVertexAttribPointer(vPosition, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(0));
```

- This statement associates the attribute ID `vPosition` (i.e., `0`) with the following information.
  - The `2` indicates the number of objects that constitute a single attribute (2 floats = a 2D point).
  - `GL_FLOAT` tells the type of object in the attribute.

Vertex Array Objects

Vertex Array Object

```
glVertexAttribPointer(vPosition, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(0));
```

- This statement associates the attribute ID `vPosition` (i.e., `0`) with the following information.
  - The `2` indicates the number of objects that constitute a single attribute (2 floats = a 2D point).
  - `GL_FLOAT` tells the type of object in the attribute.
  - `GL_FALSE` tells the GPU not to “normalize” the data (more on that later).

Vertex Array Objects

Vertex Array Object

```
glVertexAttribPointer(vPosition, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(0));
```

- This statement associates the attribute ID `vPosition` (i.e., `0`) with the following information.
  - The `2` indicates the number of objects that constitute a single attribute (2 floats = a 2D point).
  - `GL_FLOAT` tells the type of object in the attribute.
  - `GL_FALSE` tells the GPU not to "normalize" the data (more on that later).
  - the `0` is the stride, i.e., the number of bytes to skip over from one attribute value to the next. The value `0` means that the data are packed.

Vertex Array Objects

Vertex Array Object

```
glVertexAttribPointer(vPosition, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(0));
```

- This statement associates the attribute ID `vPosition` (i.e., `0`) with the following information.
  - The `2` indicates the number of objects that constitute a single attribute (2 floats = a 2D point).
  - `GL_FLOAT` tells the type of object in the attribute.
  - `GL_FALSE` tells the GPU not to "normalize" the data (more on that later).
  - the `0` is the stride, i.e., the number of bytes to skip over from one attribute value to the next. The value `0` means that the data are packed.
  - `BUFFER_OFFSET(0)` gives the offset, in bytes, to the first attribute value.

Vertex Array Objects

Enable the Attribute

```
glEnableVertexAttribArray(vPosition);
```

- This statement makes the attribute with index `vPosition` (i.e., `0`) active.
- The values will be available in the shader programs.

Outline

Drawing the Object

Drawing the Objectbject

```
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

- Invoke the `glDrawArrays()` function, with parameters
  - The type of object to draw (e.g., `GL_TRIANGLE_FAN`).
  - The starting index in the array.
  - The number of vertices.
- This example will draw a rectangle.

Drawing the Object

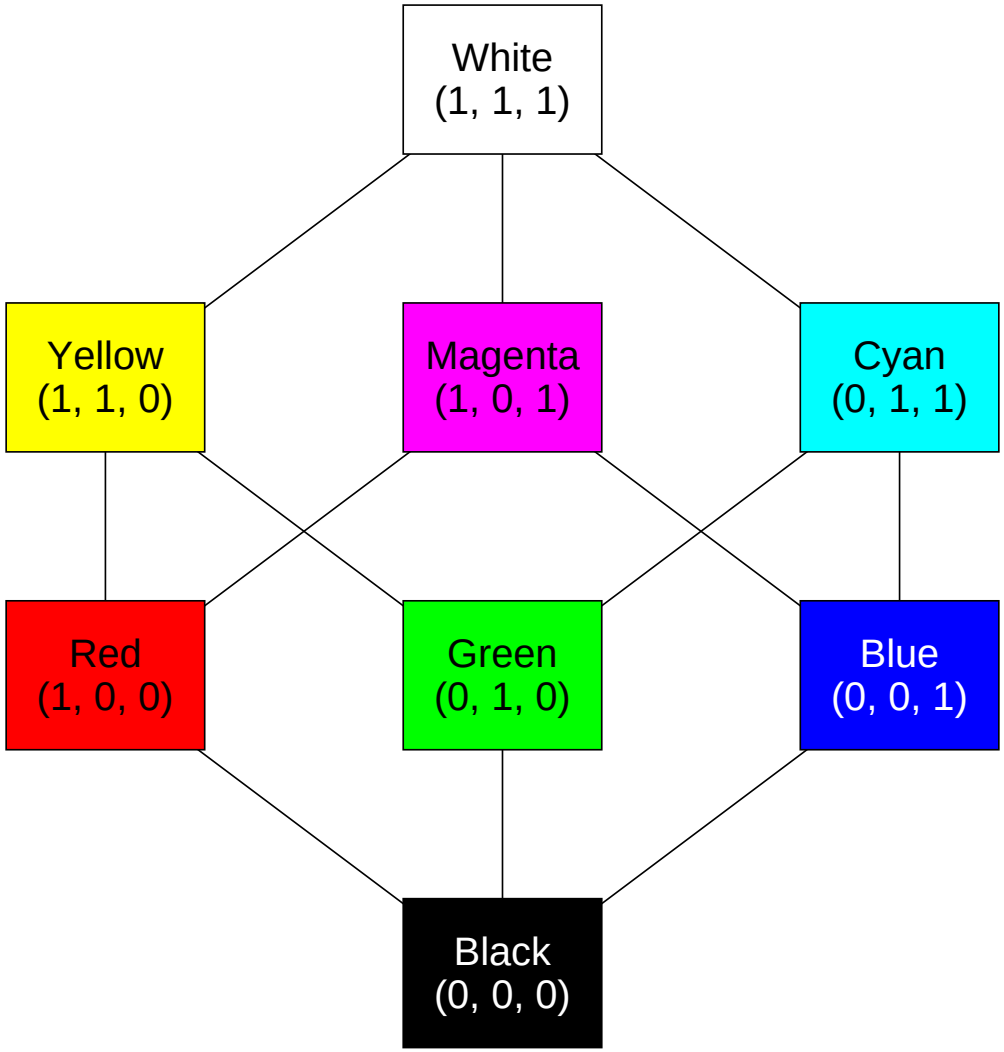
- There are several types of objects to draw.
- Primitives
  - `GL_POINTS` – individual points
  - `GL_LINES` – line segments
  - `GL_TRIANGLES` – triangles
- Nonprimitives
  - `GL_LINE_STRIP` – line segments joined in sequence
  - `GL_LINE_LOOP` – line segments joined in a circuit
  - `GL_TRIANGLE_FAN` – triangles fanning out from a base point
  - `GL_TRIANGLE_STRIP` – triangles forming a strip

Outline

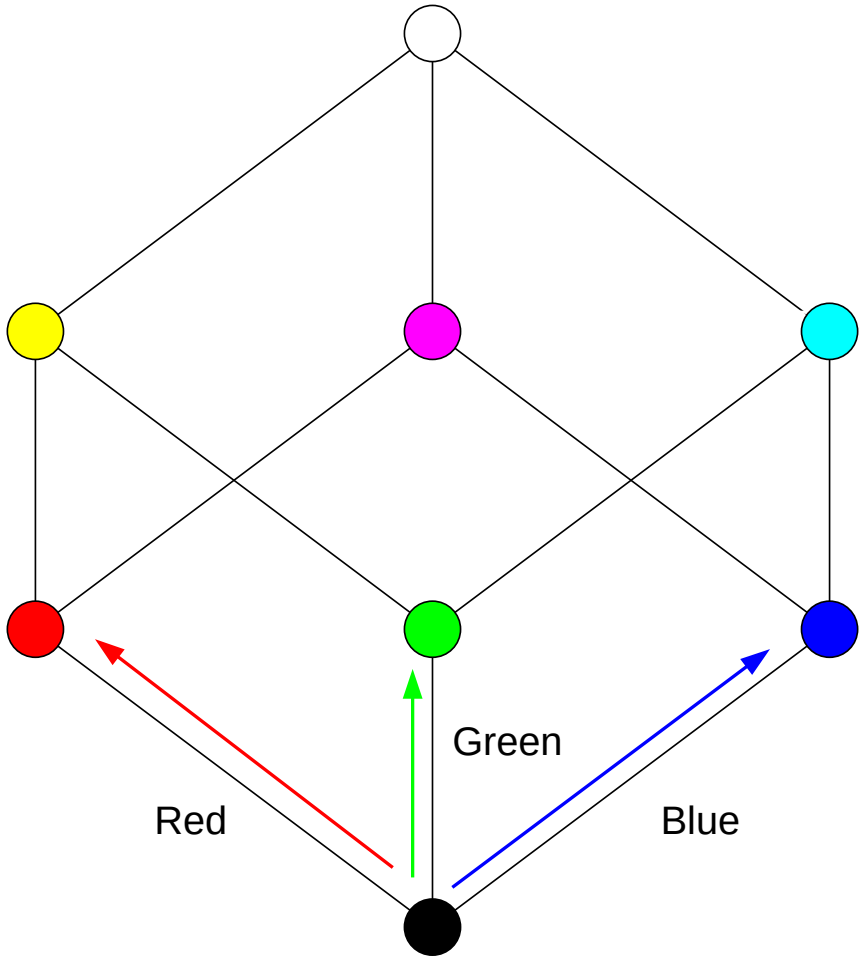
Color

- In computer graphics, every color has three components.
  - Red
  - Green
  - Blue
- Any specific color is represented by a triple (r, g, b), with each component between 0.0 and 1.0.
- The RGB values are clamped to the range $[0, 1]$.

Color


White
(1, 1, 1)
Yellow
(1, 1, 0)
Magenta
(1, 0, 1)
Cyan
(0, 1, 1)
Red
(1, 0, 0)
Green
(0, 1, 0)
Blue
(0, 0, 1)
Black
(0, 0, 0)

Color



Color

- What RGB triple would appear gray?

Color

- What RGB triple would appear gray?
- Orange?

Color

- What RGB triple would appear gray?
- Orange?
- Brown?

Color

- What RGB triple would appear gray?
- Orange?
- Brown?
- Pink?

Color

- What RGB triple would appear gray?
- Orange?
- Brown?
- Pink?
- Beige?

Color

- What RGB triple would appear gray?
- Orange?
- Brown?
- Pink?
- Beige?
- Garnet?

Outline

Coloring a Rectangle

- To color a rectangle, we need to include the color data in the buffer along with the vertex coordinates.
- There are several ways to do this.

Outline

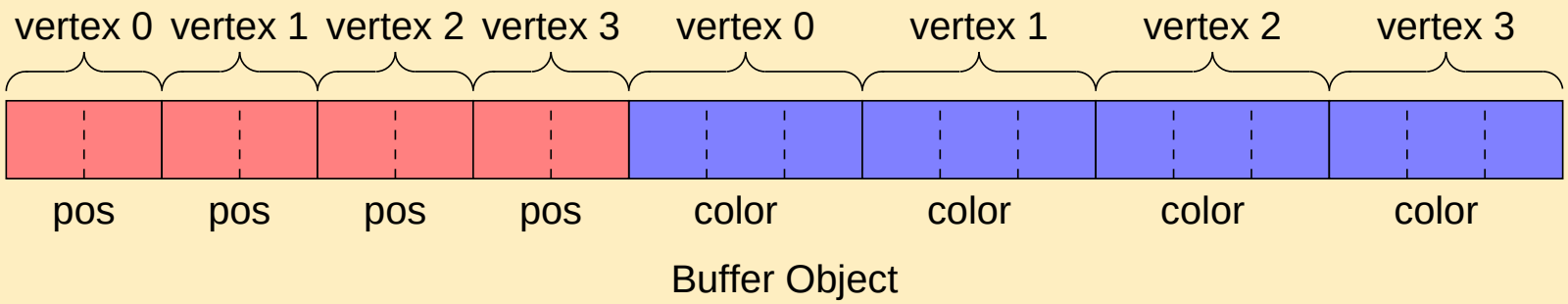
One Array, Segregated Attributes

One Array, Segregated Attributes

```
GLfloat rect_data[] =
{
    -0.5f, -0.5f,         // 1st vertex
     0.5f, -0.5f,         // 2nd vertex
     0.5f,  0.5f,         // 3rd vertex
    -0.5f,  0.5f,         // 4th vertex
     1.0f, 0.0f, 0.0f,    // Color of 1st
     1.0f, 1.0f, 0.0f,    // Color of 2nd
     0.0f, 1.0f, 0.0f,    // Color of 3rd
     0.0f, 0.0f, 1.0f     // Color of 4th
};
```

- We can pack all the data contiguously into one array, with the attributes segregated.

One Array, Segregated Attributes



Color a Rectangle

Color a Rectangle

```
enum {vPosition = 0, vColor = 1};
```

- Create a symbolic name for the color attribute.

Color a Rectangle

Color a Rectangle

```
glNamedBufferStorage(VBO[RectBuffer], sizeof(rect_data),
    rect_data, 0);
```

- Store the data in the buffer and bind the vertex array object, as before.

Color a Rectangle

Color a Rectangle

```
glBindVertexArray(VAOs[Rect]);
glVertexAttribPointer(vPosition, 2, GL_FLOAT, GL_FALSE,
    0, BUFFER_OFFSET(0));
glVertexAttribPointer(vColor, 3, GL_FLOAT, GL_FALSE,
    0, BUFFER_OFFSET(8*sizeof(GLfloat)));
```

- Set the position attribute as before.
- Give the color attribute an offset equal to the size of the position data.
- Both attributes have a stride of `0`.

Outline

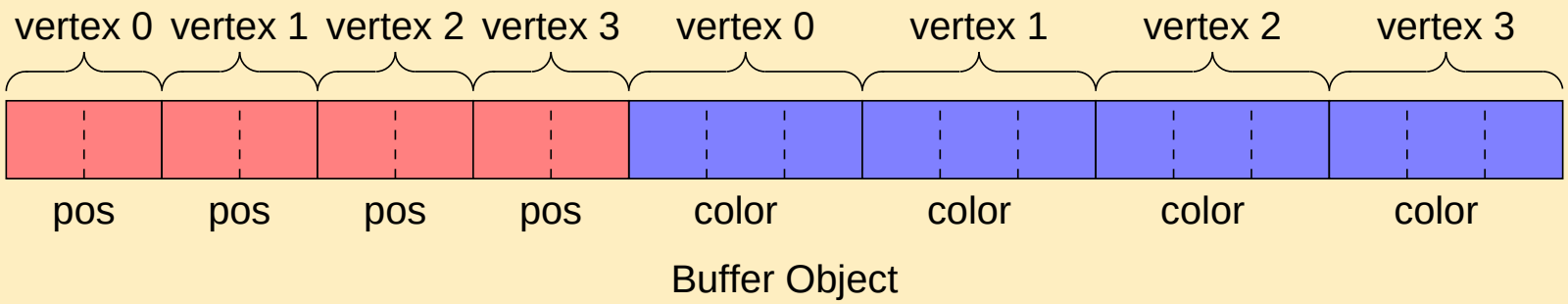
Two Arrays, Segregated Attributes

Two Arrays, Segregated Attributes

```
GLfloat rect_pos[] =
{
    -0.5f, -0.5f,        // 1st vertex
     0.5f, -0.5f,        // 2nd vertex
     0.5f,  0.5f,        // 3rd vertex
    -0.5f,  0.5f         // 4th vertex
};
GLfloat rect_color[] =
{
    1.0f, 0.0f, 0.0f,   // Color of 1st
    1.0f, 1.0f, 0.0f,   // Color of 2nd
    0.0f, 1.0f, 0.0f,   // Color of 3rd
    0.0f, 0.0f, 1.0f    // Color of 4th
};
```

- We can create two separate arrays, with the attributes necessarily segregated.

One Array, Segregated Attributes



Two Arrays, Segregated Attributes

Two Arrays, Segregated Attributes

```
glNamedBufferStorage(VBO[RectBuffer], sizeof(rect_pos)
    + sizeof(rect_color), NULL, 0);
glNamedBufferSubData(VBO[RectBuffer], 0, sizeof(rect_pos),
    rect_pos);
glNamedBufferSubData(VBO[RectBuffer], sizeof(rect_pos),
    sizeof(rect_color), rect_color);
```

- We must first reserve the memory and then separately store the two arrays using `glNamedBufferSubData()`.

Two Arrays, Segregated Attributes

Two Arrays, Segregated Attributes

```
glBindVertexArray(VAOs[Rect]);
glVertexAttribPointer(vPosition, 2, GL_FLOAT, GL_FALSE,
    0, BUFFER_OFFSET(0));
glVertexAttribPointer(vColor, 3, GL_FLOAT, GL_FALSE,
    0, BUFFER_OFFSET(sizeof(rect_pos)));
```

- Set the position attribute as before.
- Give the color attribute an offset equal to the size of the position data.

Outline

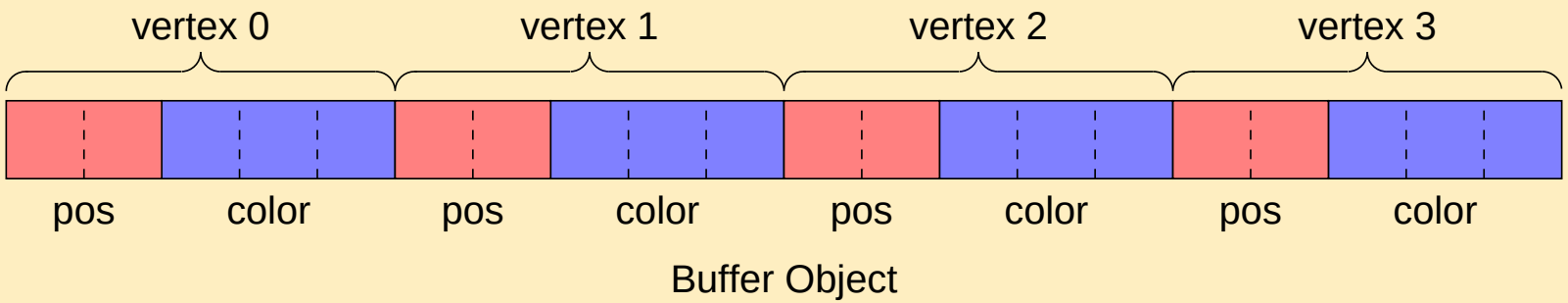
One Array, Integrated Attributes

One Array, Integrated Attributes

```
GLfloat rect_data[] =
{
    -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, // 1st vertex
     0.5f, -0.5f, 1.0f, 1.0f, 0.0f, // 2nd vertex
     0.5f,  0.5f, 0.0f, 1.0f, 0.0f, // 3rd vertex
    -0.5f,  0.5f, 0.0f, 0.0f, 1.0f  // 4th vertex
};
```

- We can create one array, with the attributes integrated.

One Array, Integrated Attributes



One Array, Integrated Attributes

One Array, Integrated Attributes

```
glNamedBufferStorage(VBO[RectBuffer], sizeof(rect_data),
    rect_data, 0);
```

- Store the data in the buffer and bind the vertex array object, as before.

One Array, Integrated Attributes

One Array, Integrated Attributes

```
glBindVertexArray(VAOs[Rect]);
glVertexAttribPointer(vPosition, 2, GL_FLOAT, GL_FALSE,
    5*sizeof(GL_FLOAT), BUFFER_OFFSET(0));
glVertexAttribPointer(vColor, 3, GL_FLOAT, GL_FALSE,
    5*sizeof(GL_FLOAT), BUFFER_OFFSET(2*sizeof(GLfloat)));
```

- Set the position attribute as before.
- Give the color attribute an offset equal to the size of a position.
- Give the position and color a stride equal to the size of the data for a vertex.

Outline

One Array, Structured Data

One Array, Structured Data

```
struct VertexData2D
{
    GL_FLOAT pos[2];
    GL_FLOAT color[3];
};
```

- Create a `VertexData2D` structure.

One Array, Structured Data

One Array, Structured Data

```
struct VertexData2D
{
    vec2 pos;
    vec3 color;
};
```

- Create a `VertexData2D` structure.

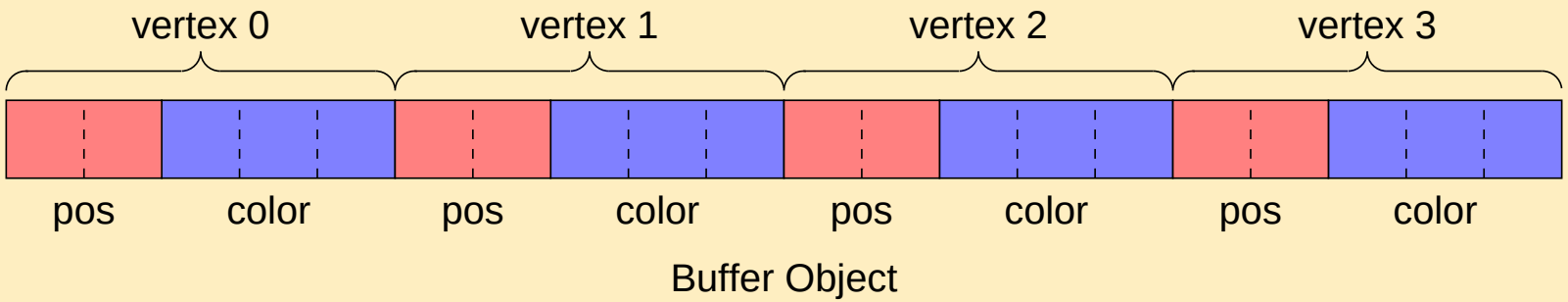
One Array, Structured Data

One Array, Structured Data

```
VertexData2D rect_data[] =
{
    {{-0.5f, -0.5f}, {1.0f, 0.0f, 0.0f}},   // 1st vertex
    {{ 0.5f, -0.5f}, {1.0f, 1.0f, 0.0f}},   // 2nd vertex
    {{ 0.5f,  0.5f}, {0.0f, 1.0f, 0.0f}},   // 3rd vertex
    {{-0.5f,  0.5f}, {0.0f, 0.0f, 1.0f}}    // 4th vertex
};
```

- We can create one array of type `VertexData2D`.

One Array, Structured Data



One Array, Integrated Attributes

One Array, Integrated Attributes

```
glNamedBufferStorage(GL_ARRAY_BUFFER, sizeof(rect_data),
    rect_data, 0);
```

- Store the data in the buffer and bind the vertex array object, as before.

One Array, Integrated Attributes

One Array, Integrated Attributes

```
glBindVertexArray(VAOs[Rect]);
glVertexAttribPointer(vPosition, 2, GL_FLOAT, GL_FALSE,
    sizeof(VertexData2D), BUFFER_OFFSET(0));
glVertexAttribPointer(vColor, 3, GL_FLOAT, GL_FALSE,
    sizeof(VertexData2D),
    BUFFER_OFFSET(sizeof(vec2)));
```

- Set the position attribute as before.
- Give the color attribute an offset equal to the size of a position.
- Give the position a stride equal to the size of a color.
- Give the color a stride equal to the size of a position.

Outline

Assignment

Assignment

- Read pp. 16 - 22 in The Red Book.