Advanced Interactive Web Development

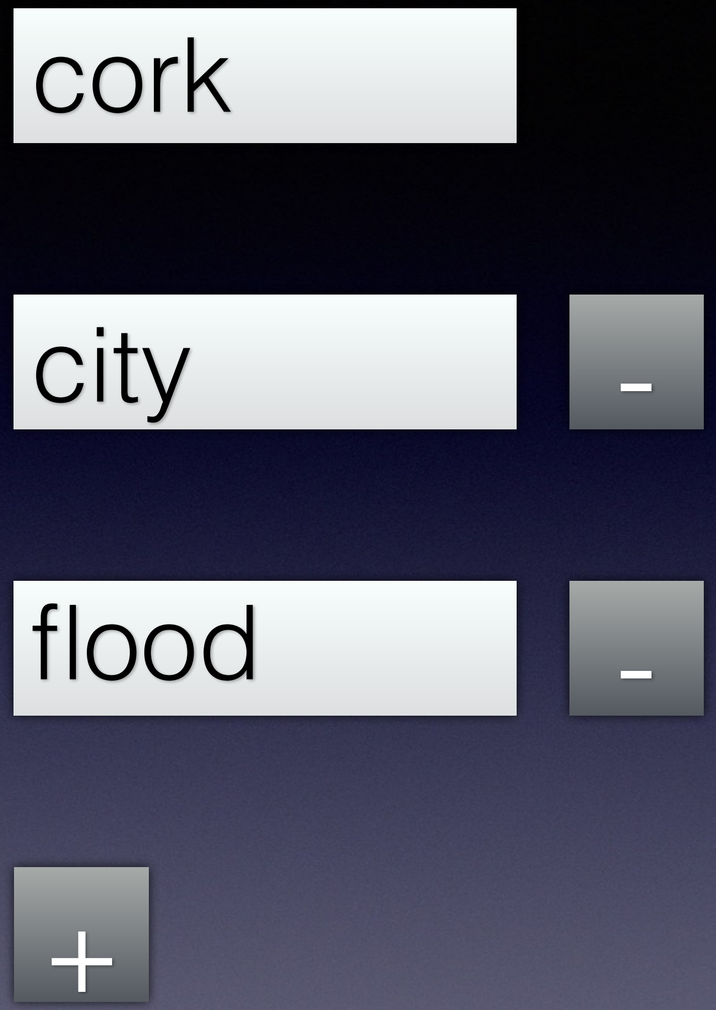
Assignment 2

Technical Notes

Form Elements

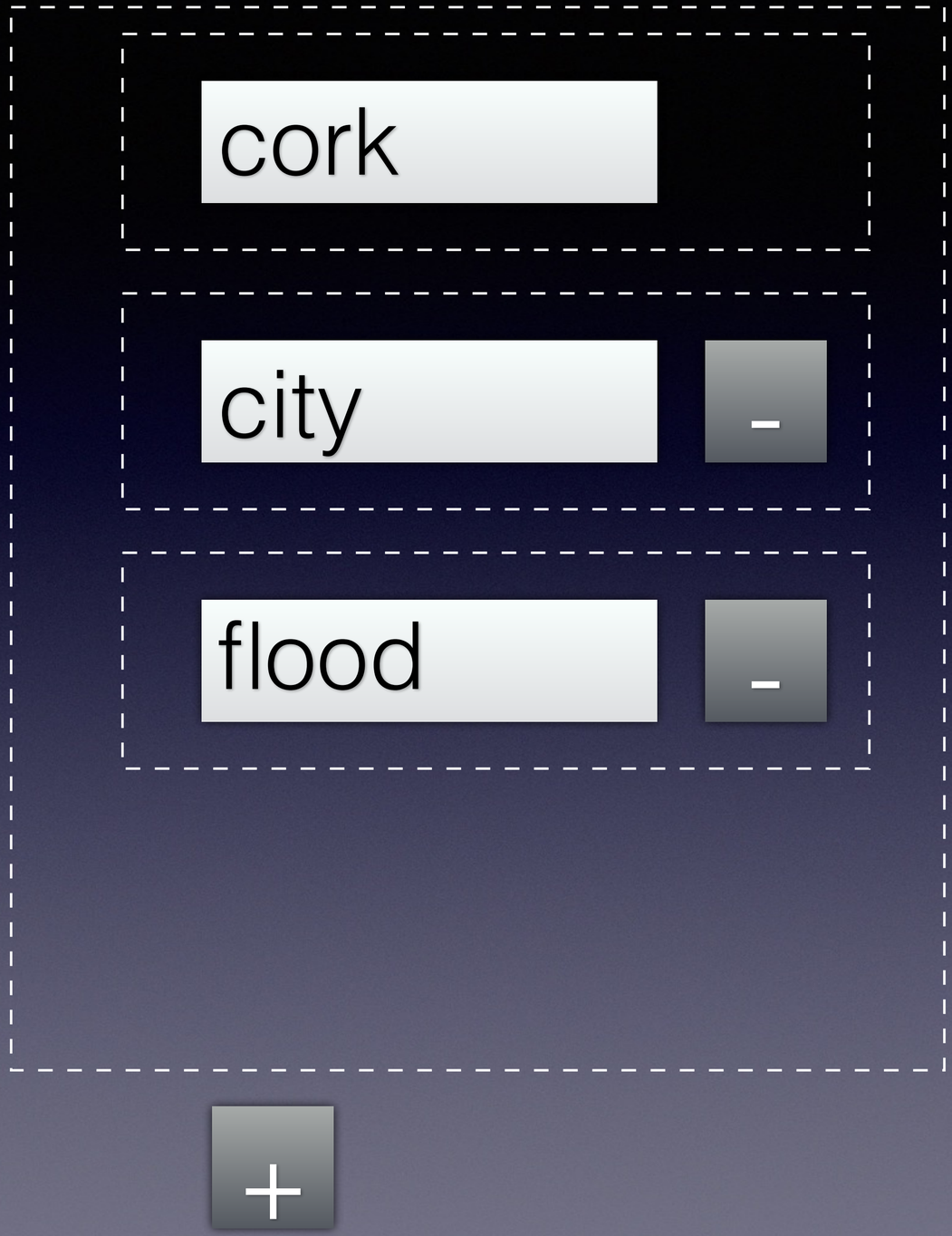
Collecting Data

The simplest solution for collecting the data in the forms is to use DOM scripting to get an array/NodeList/Collection of the text fields and loop through that array collecting the data.

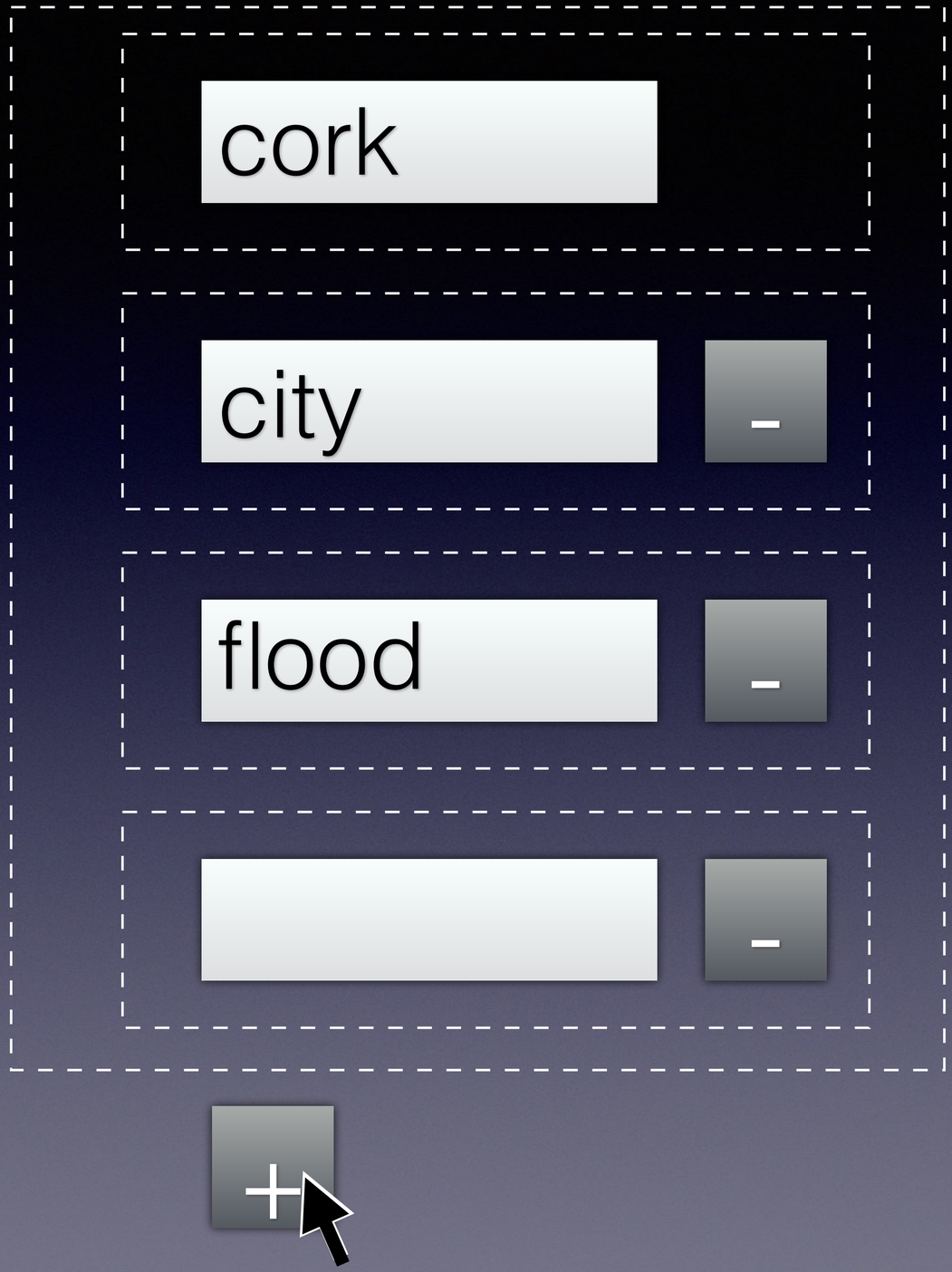


Adding/Removing Fields

If you have an element containing all the text fields then clicking the "+" button just has to add an element (containing a new text field and "-" button) as a child of that element.

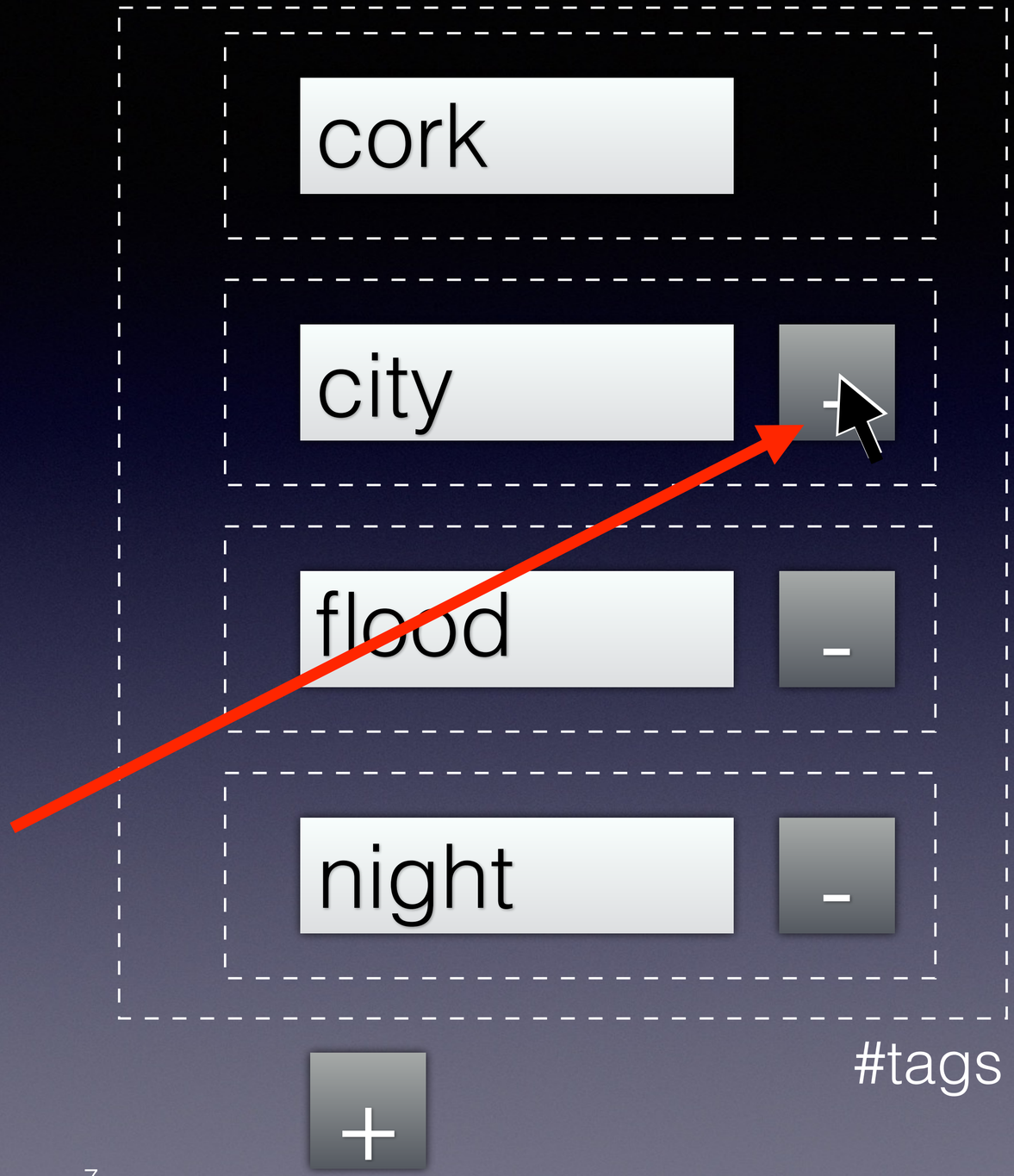


Appending that element will automatically put it at the end of the collection of text fields.

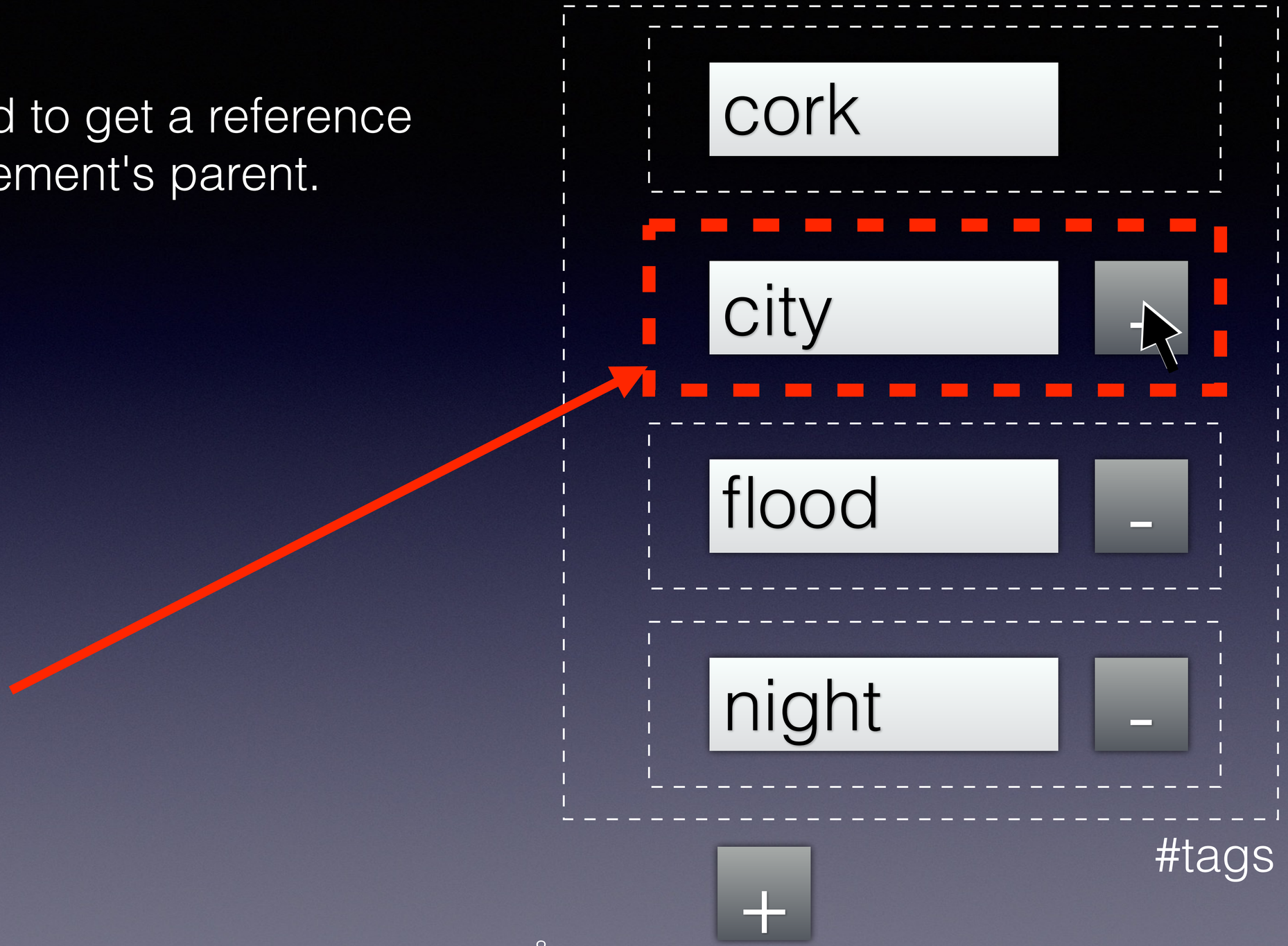


By having the text field and its "-" button as children of the same unique element you can easily remove the text field and button from the page.

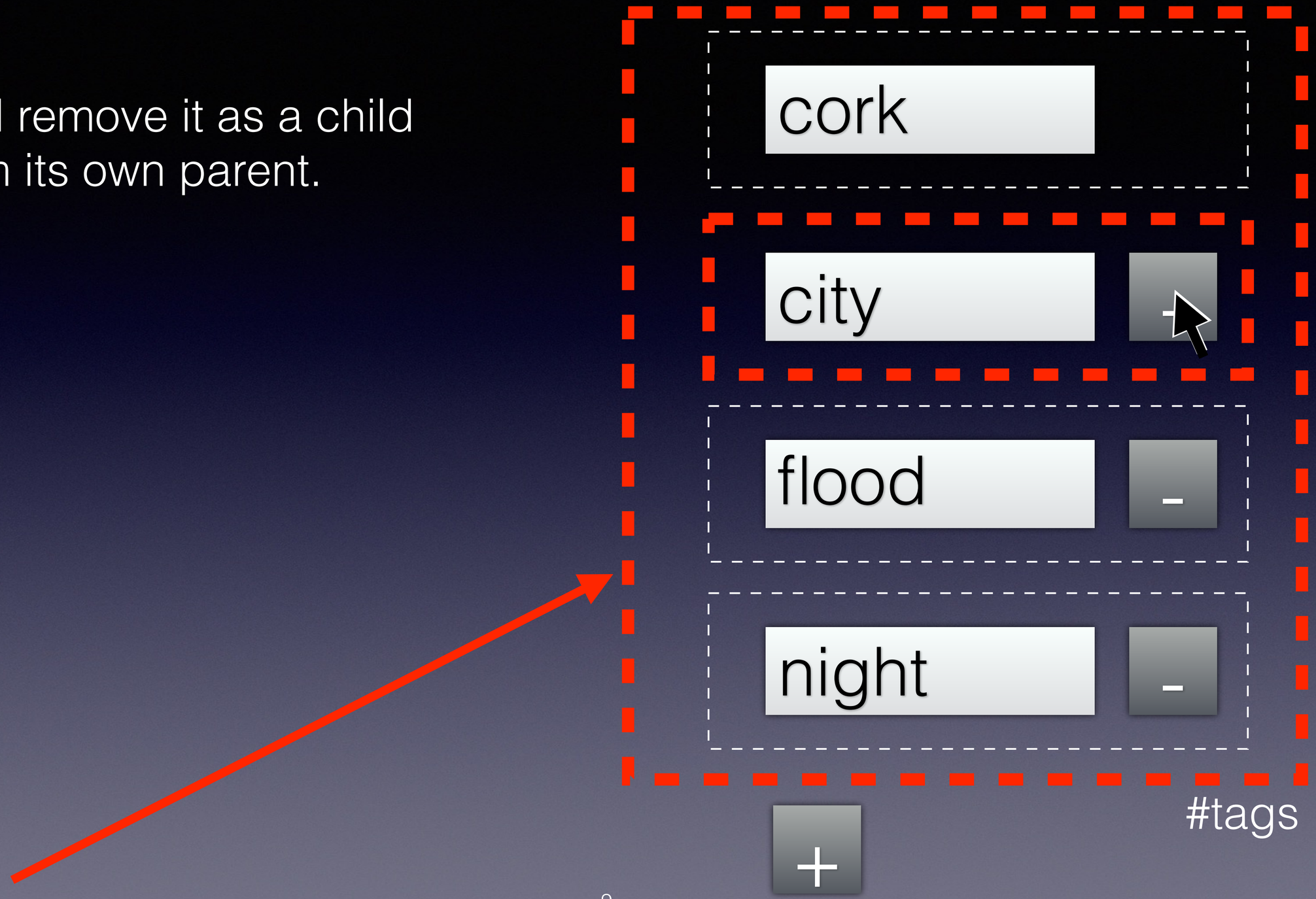
Clicking on a "-" button just has to remove its parent from the page.



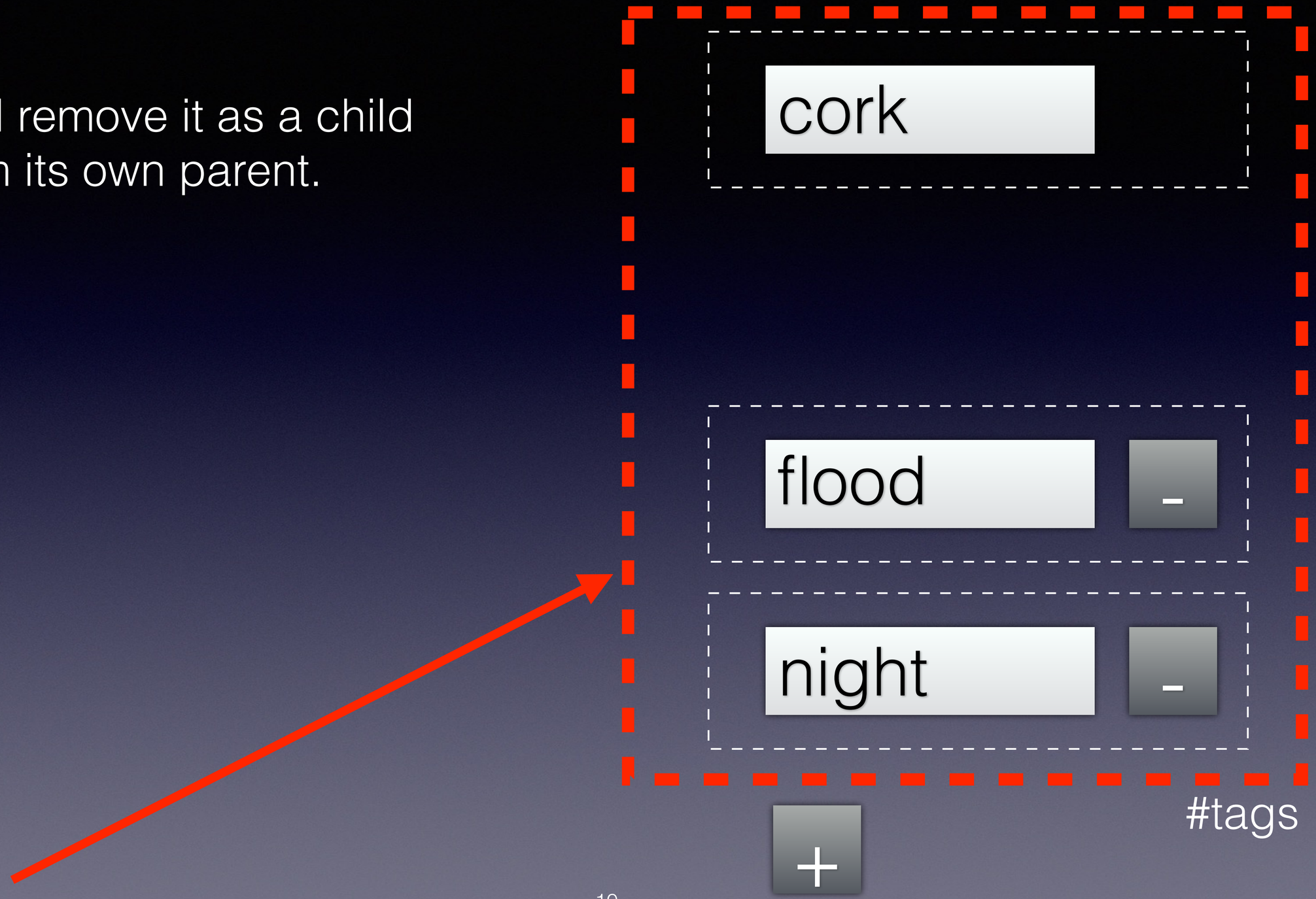
You need to get a reference to the element's parent.



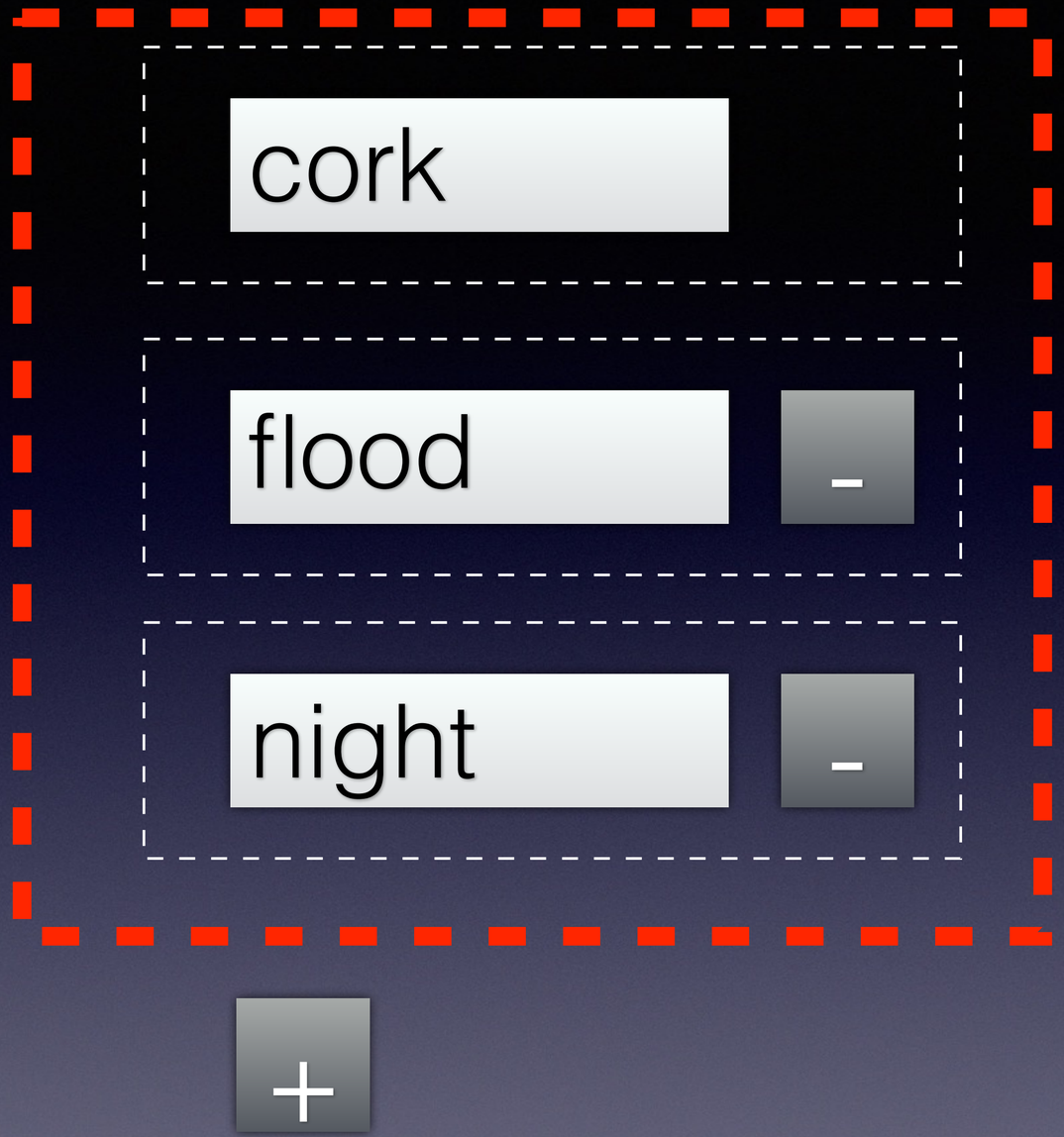
And remove it as a child from its own parent.



And remove it as a child
from its own parent.



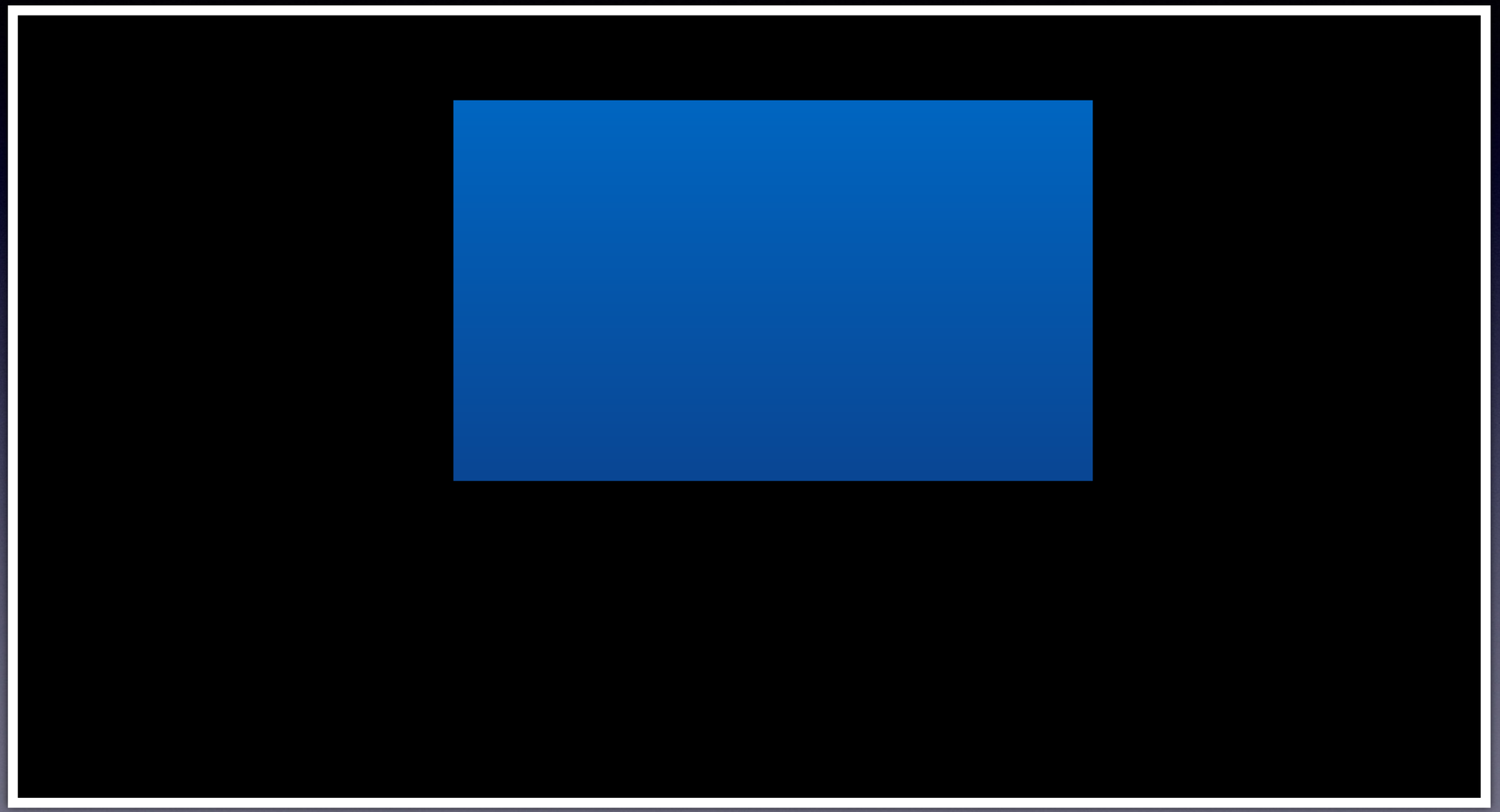
And remove it as a child from its own parent.



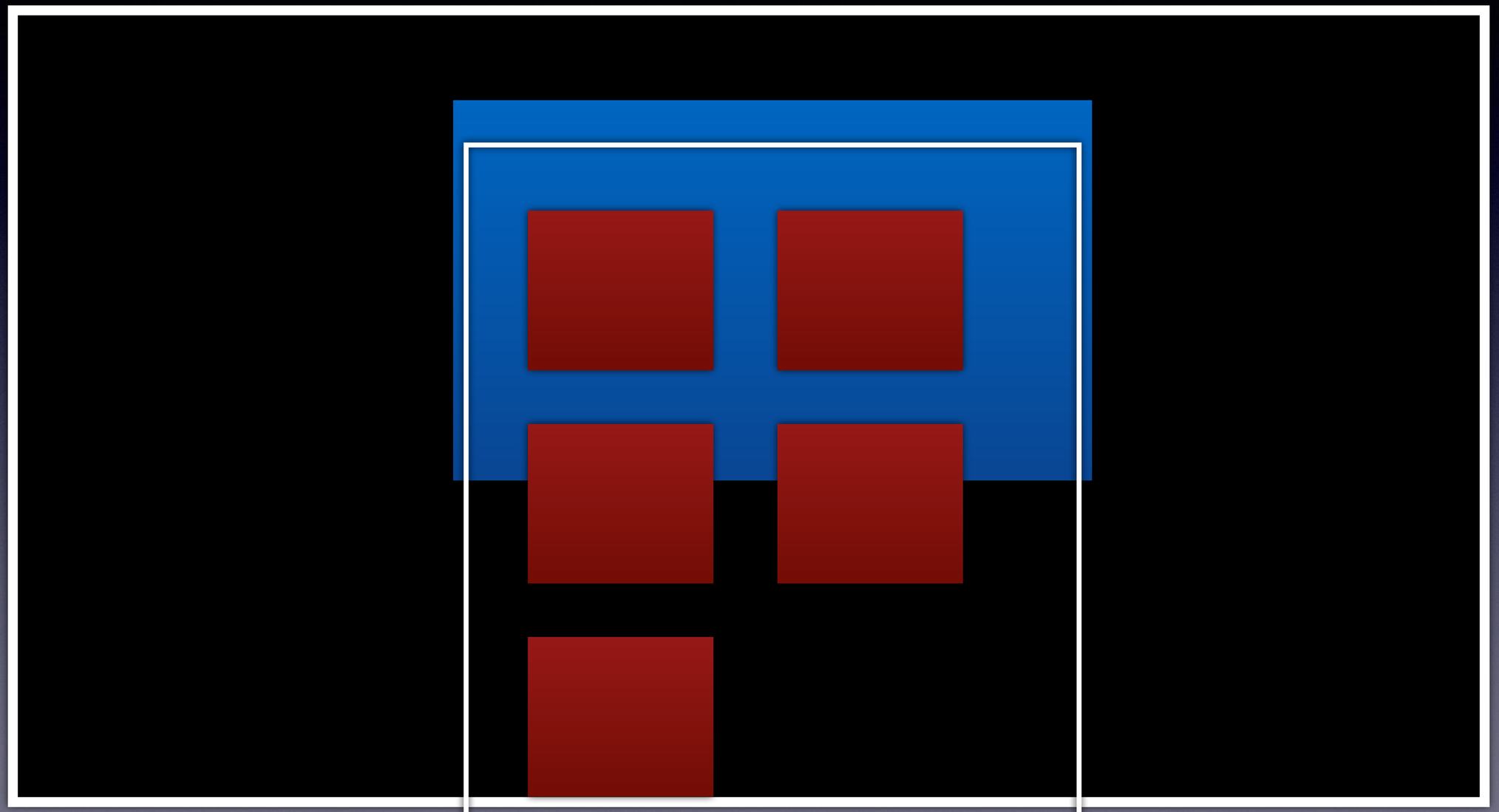
#tags

Carousel

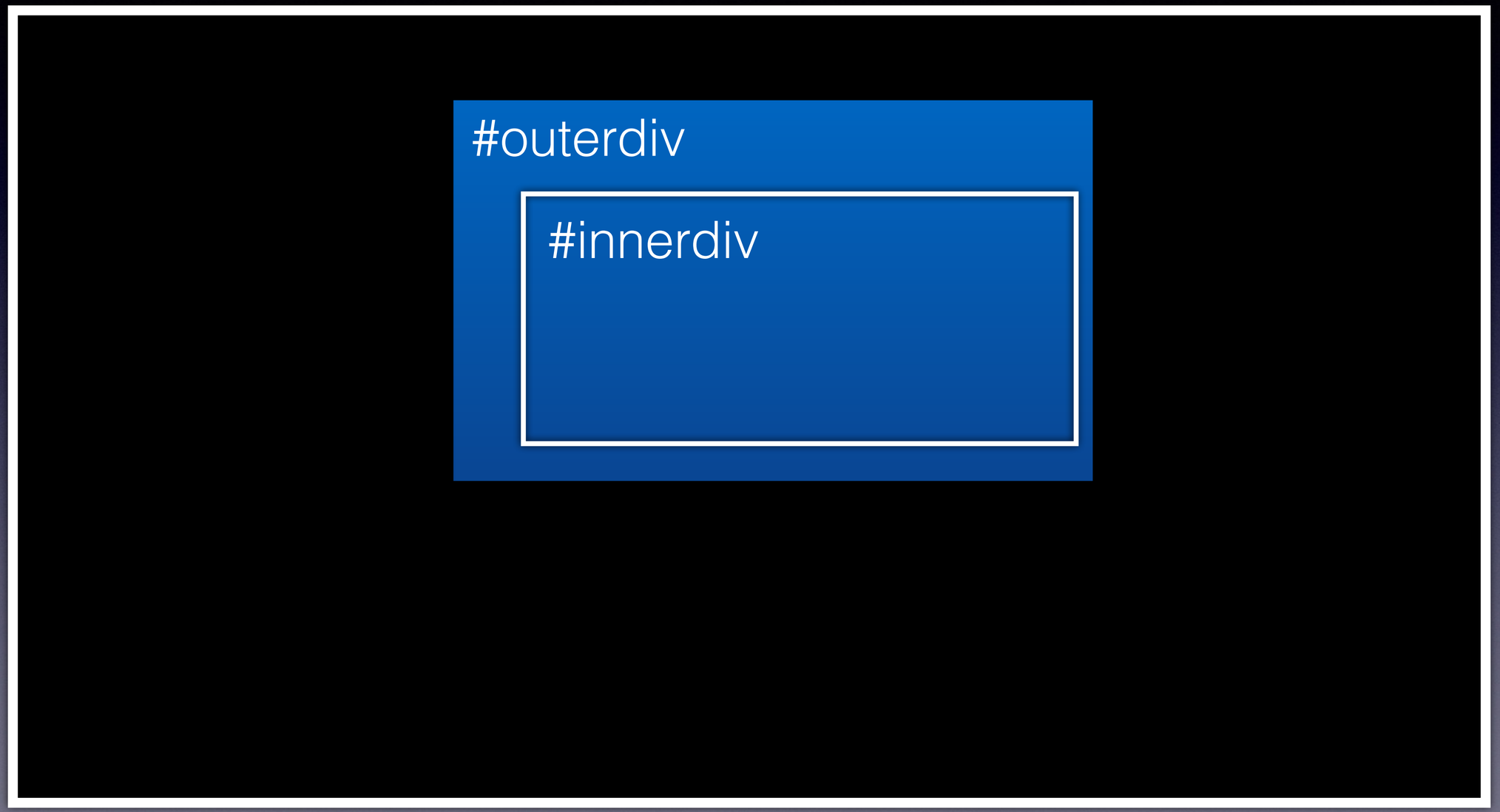
Place a container element on the page. I will assume we are using divs in these notes (you may be using a different element).



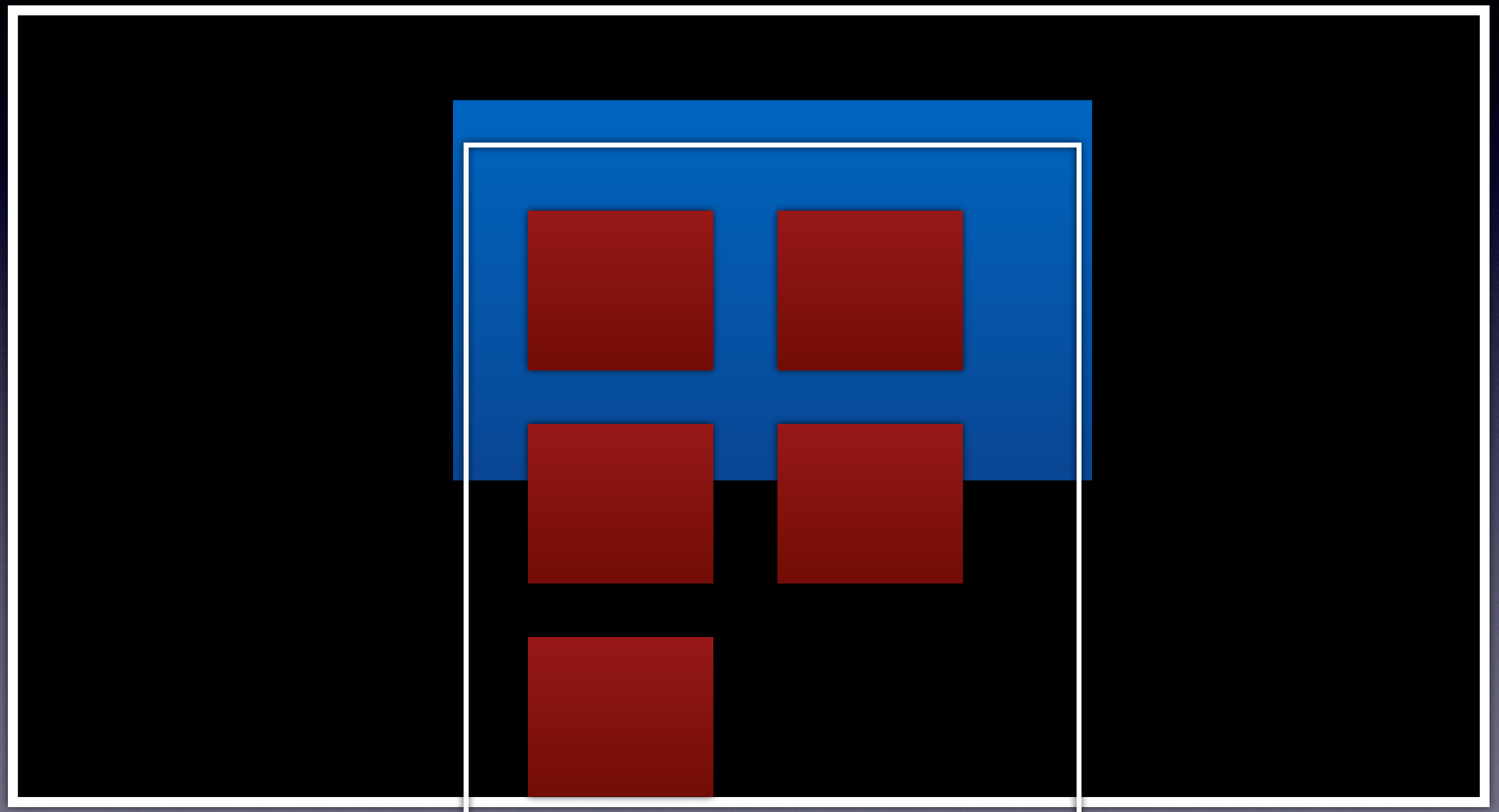
Place a new div containing the images inside the container div



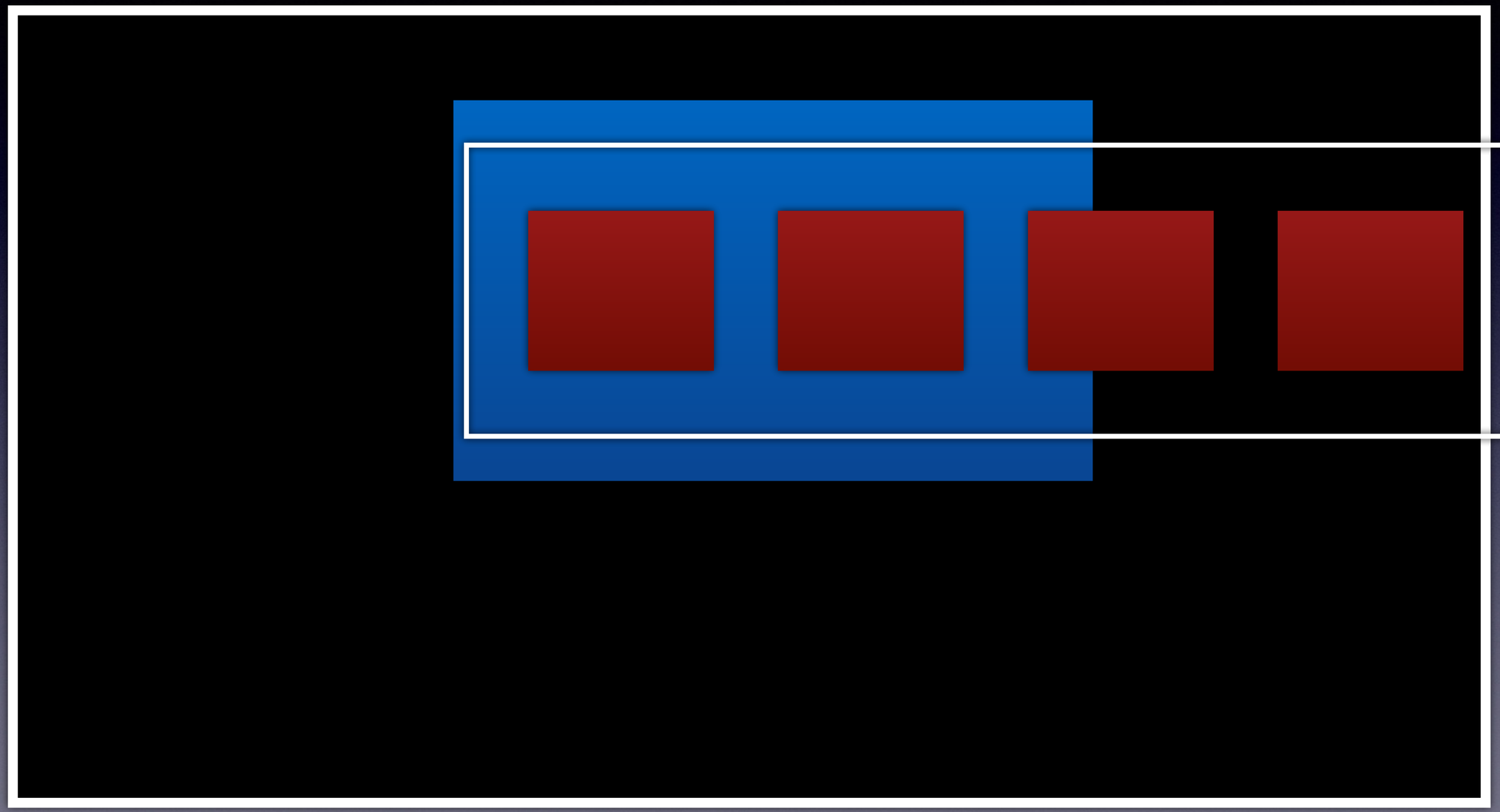
We will call these the *outer div* and *inner div* respectively (and give them similar ids in the following examples).



Position the inner div with absolute positioning **relative to the outer div**.



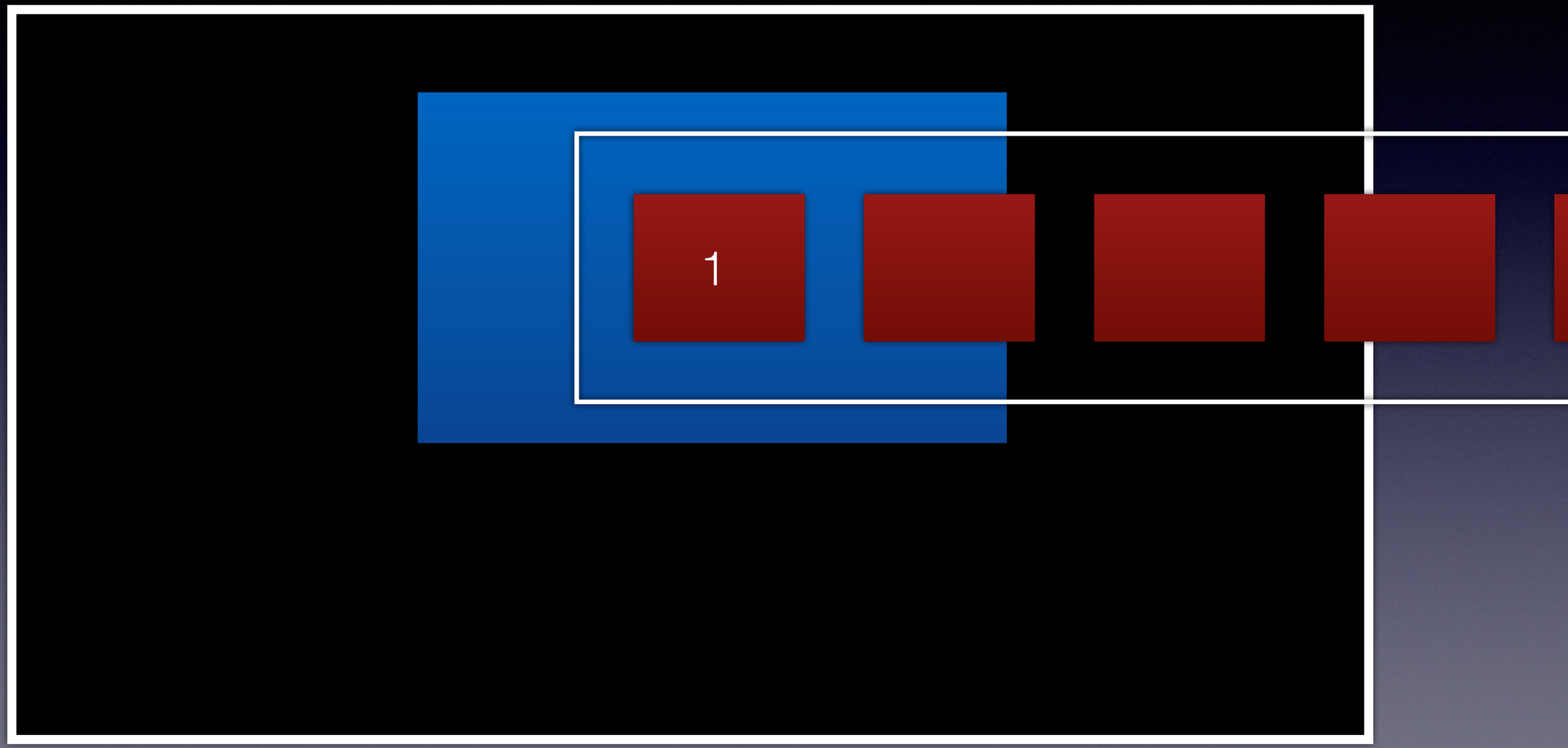
Make sure the pictures all fit on one line.

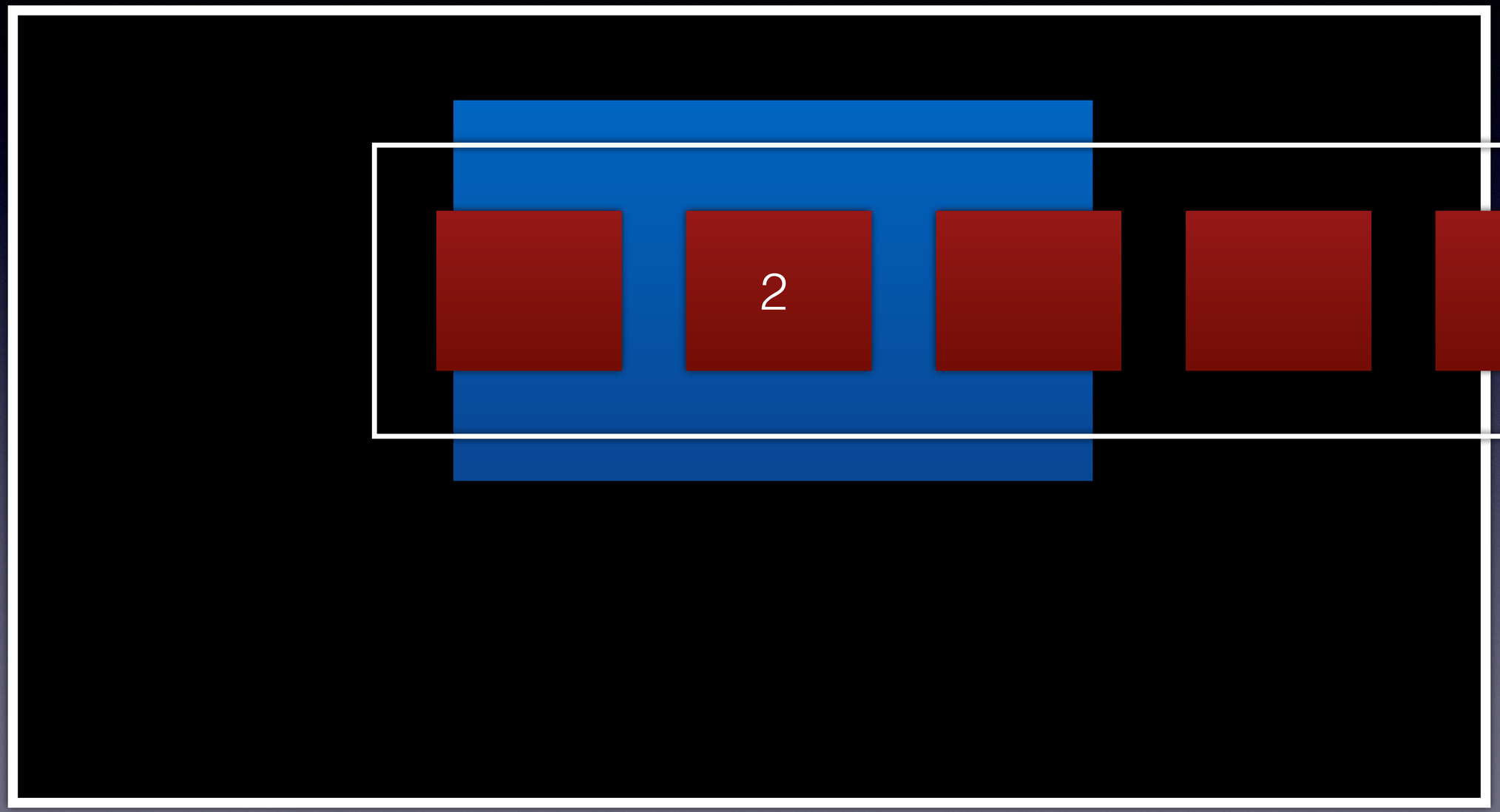


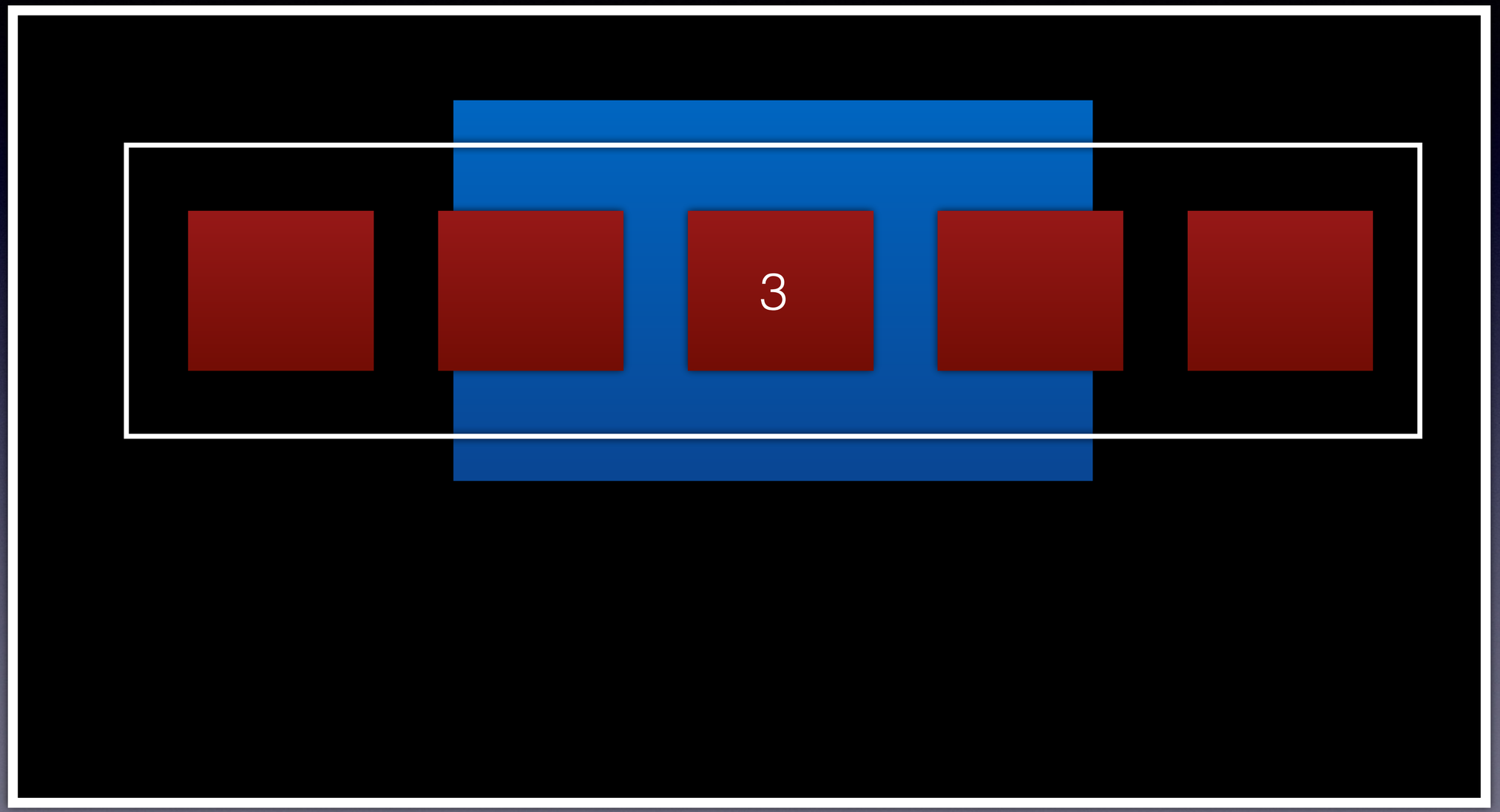
This can be done with the following CSS:

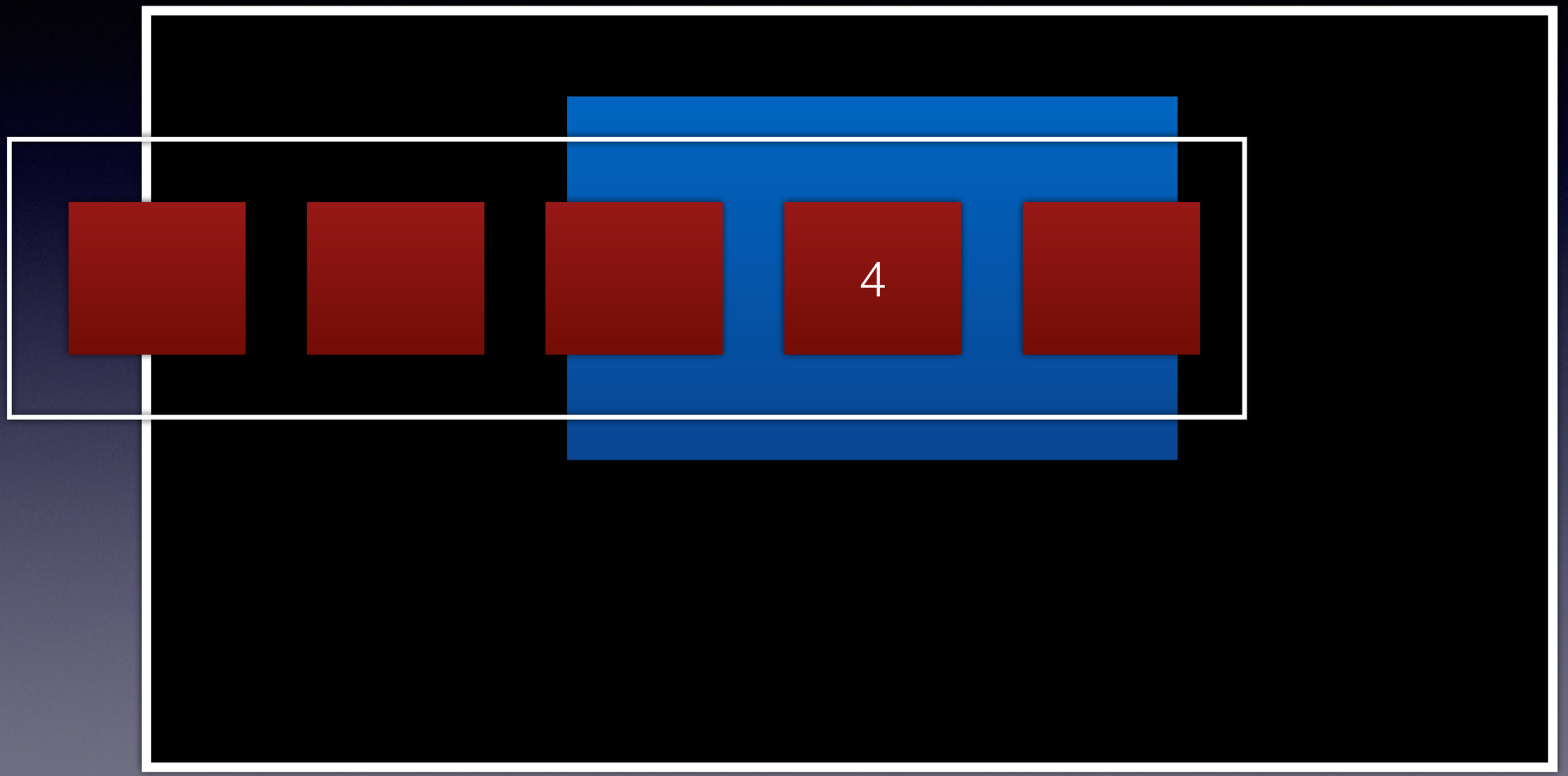
```
white-space: nowrap;
```

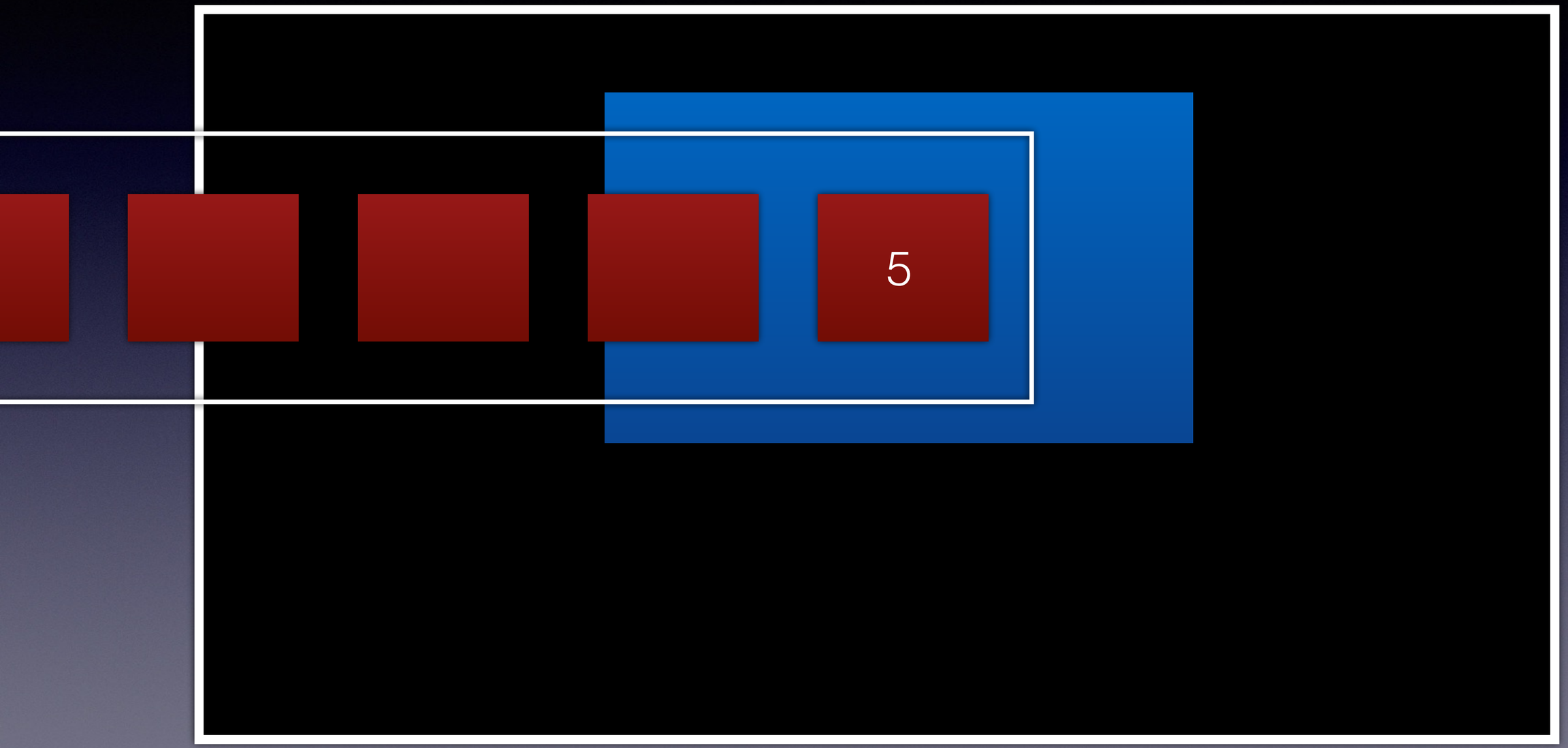
We then position the inner div so a specific image is in the center of the outer div.



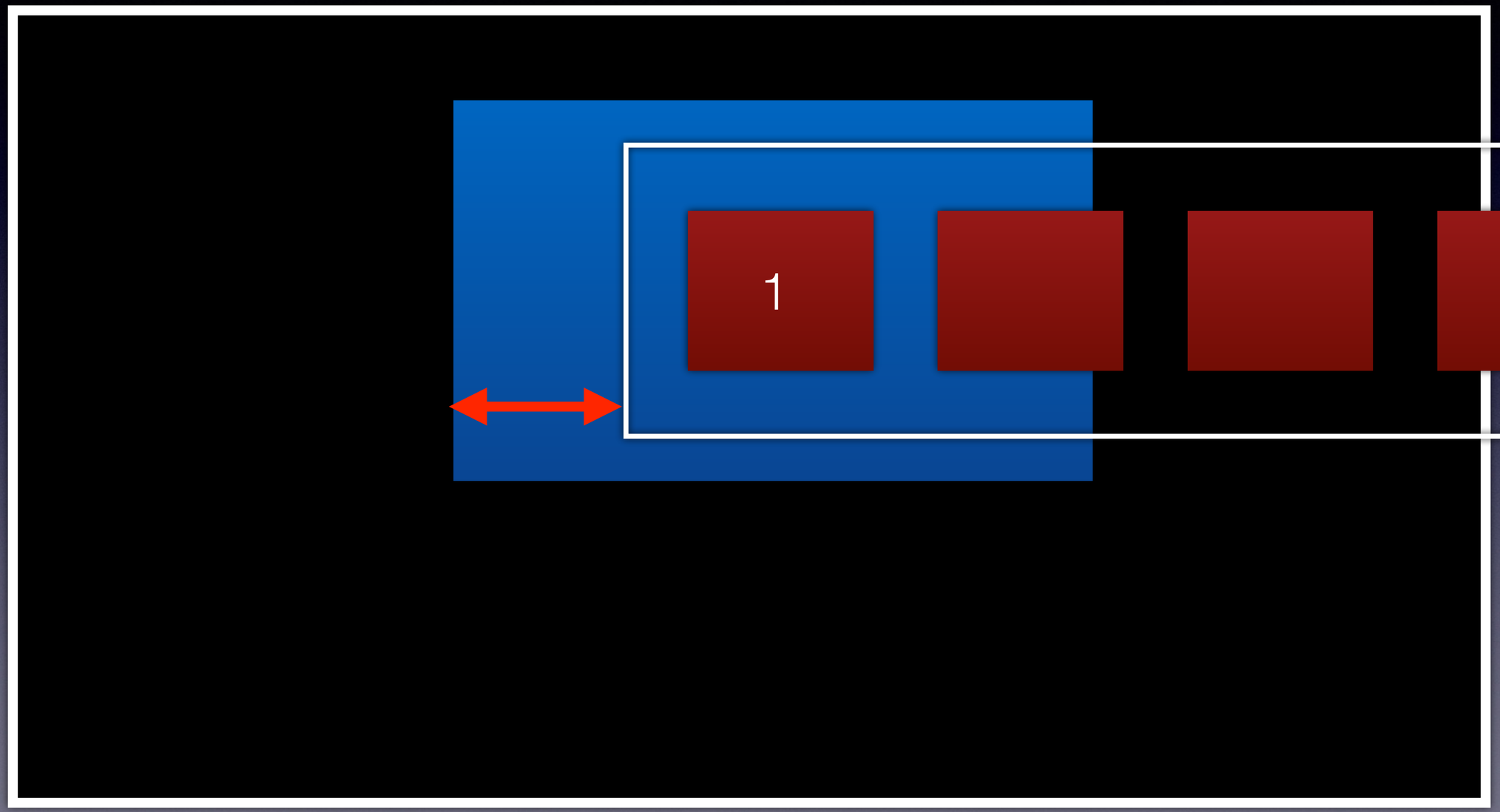

2


3

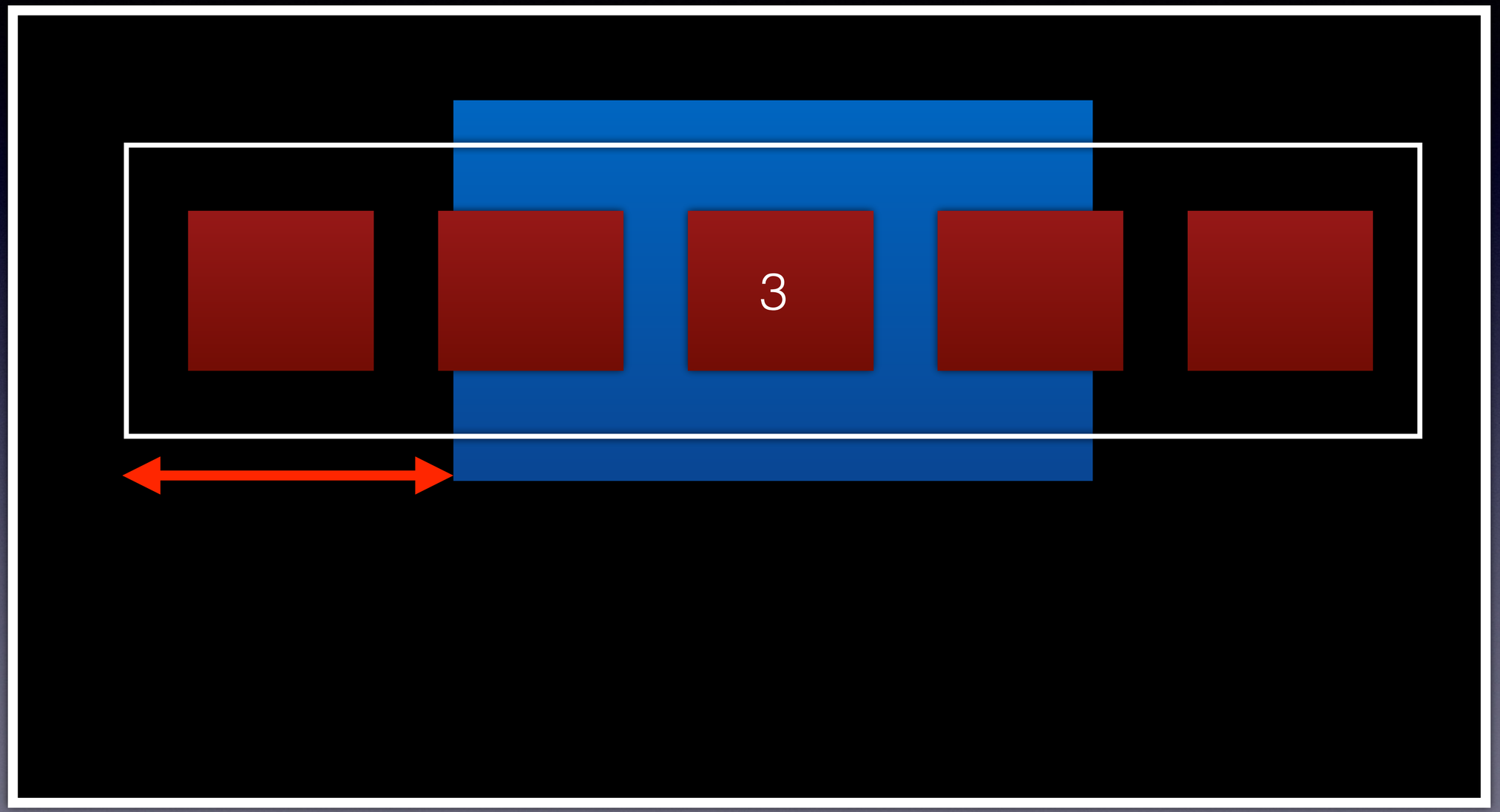

4


5

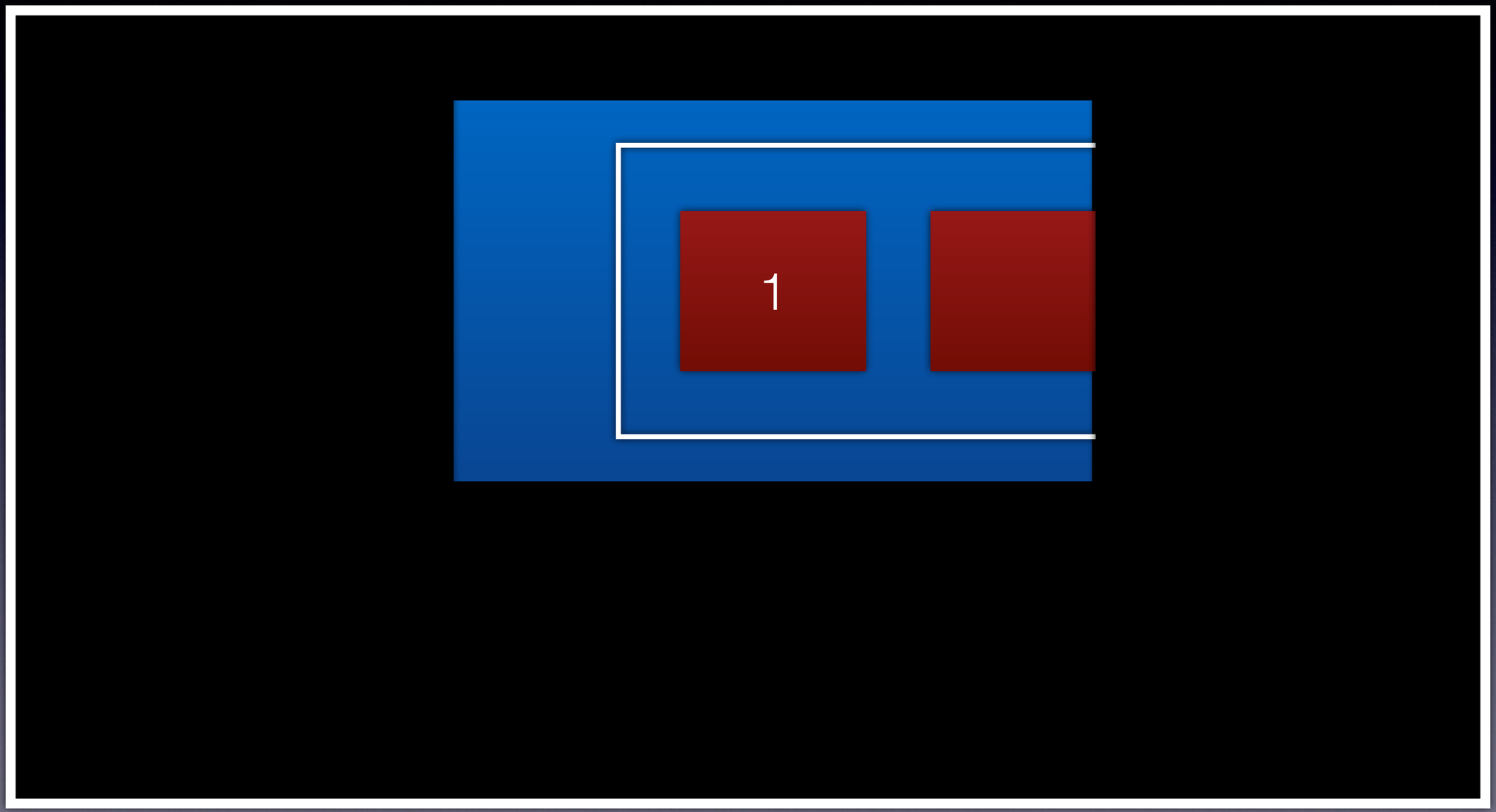
To do this we need to calculate the left offset of the inner div.



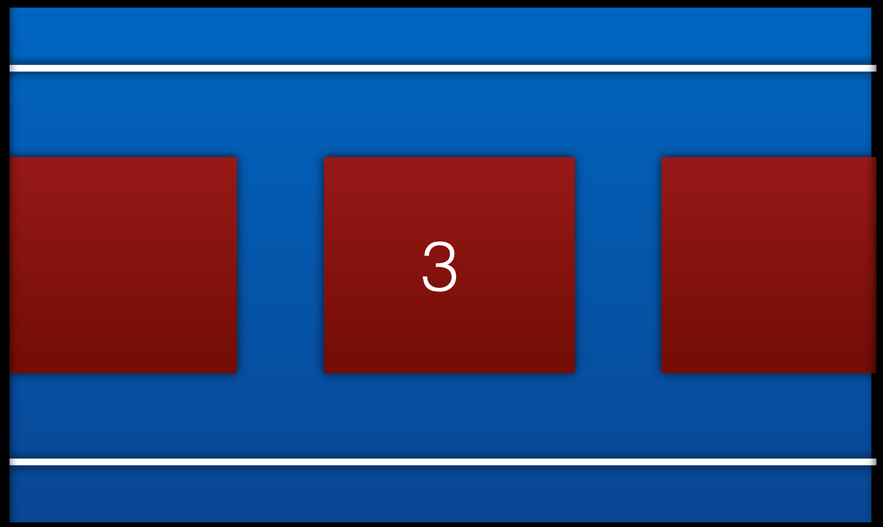
This can include negative values



The content that doesn't fit inside the outer div should be hidden



The content that doesn't fit inside the outer div should be hidden



This can be done by setting the overflow property:

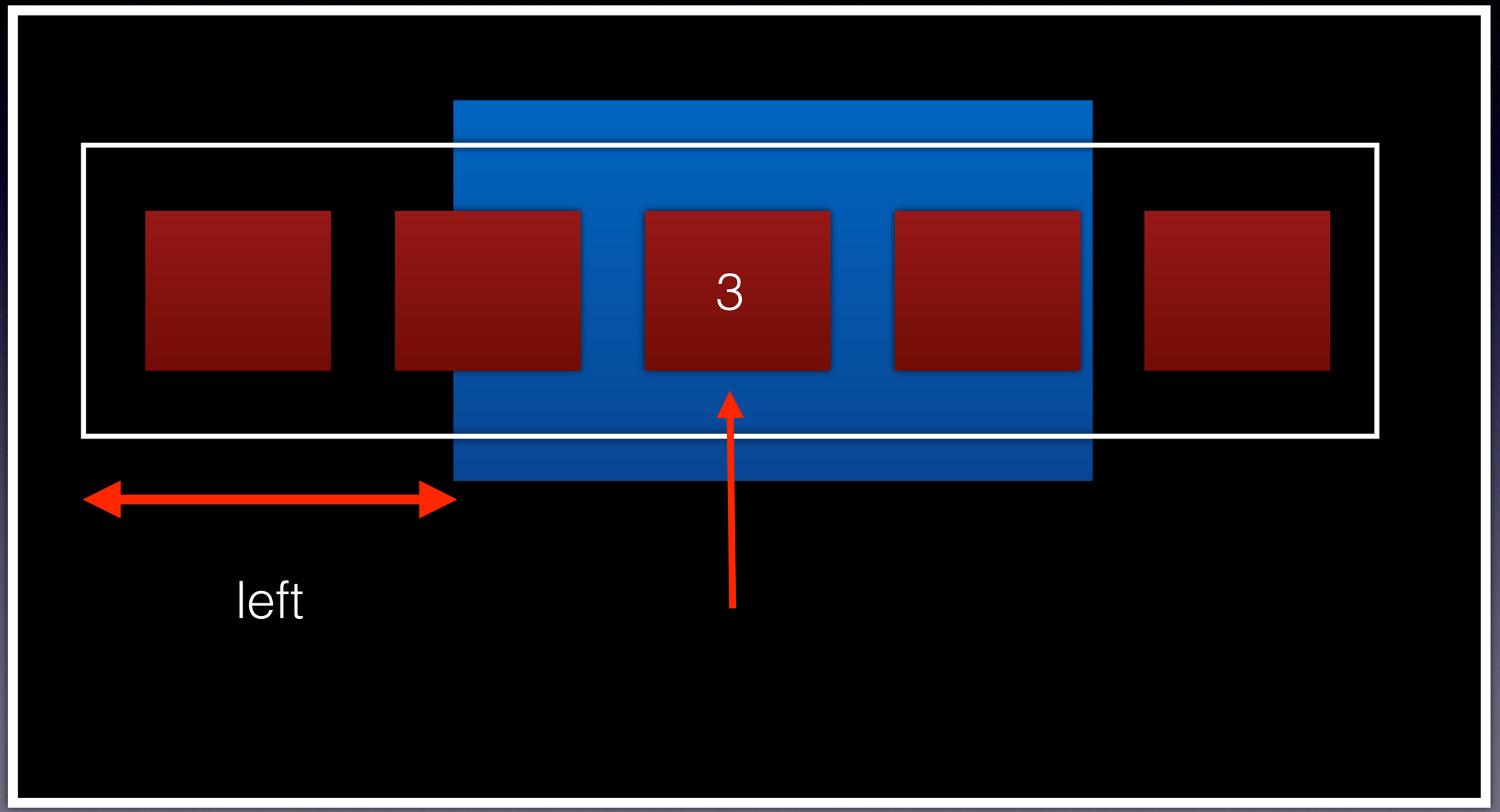
```
outerdiv {  overflow: hidden;  }
```

If the inner div is positioned with absolute positioning then the element that masks the inner div when `overflow` is set to `hidden` is the same one that is used to calculate its offsets (i.e. the first ancestor element with non-static positioning) so we must make sure that the outer div doesn't use static positioning.

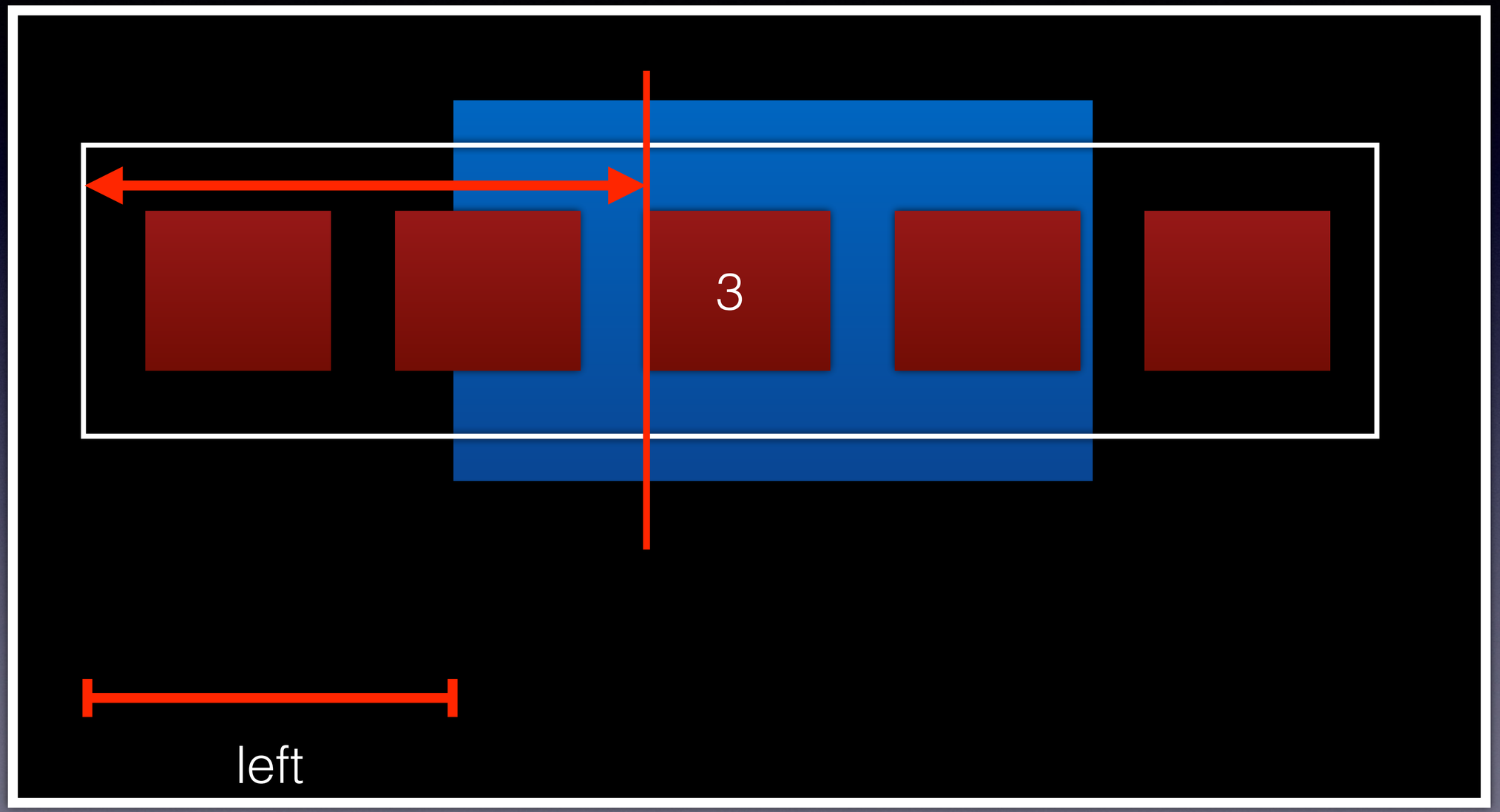
```
outerdiv {  position: relative;  }
```

Centering the Image

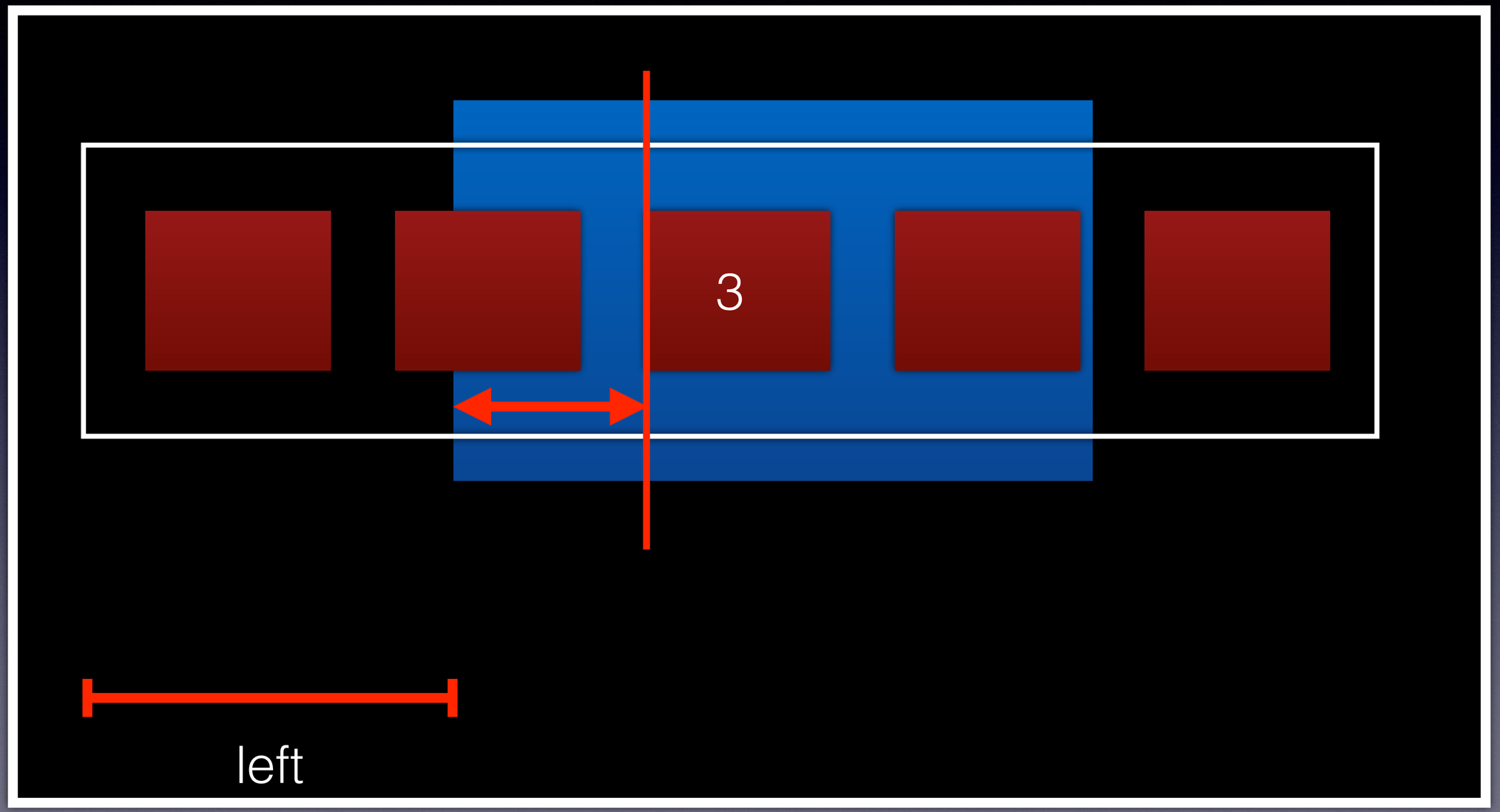
To display an image in the center of the ***outer*** div we need to calculate the **left** value.



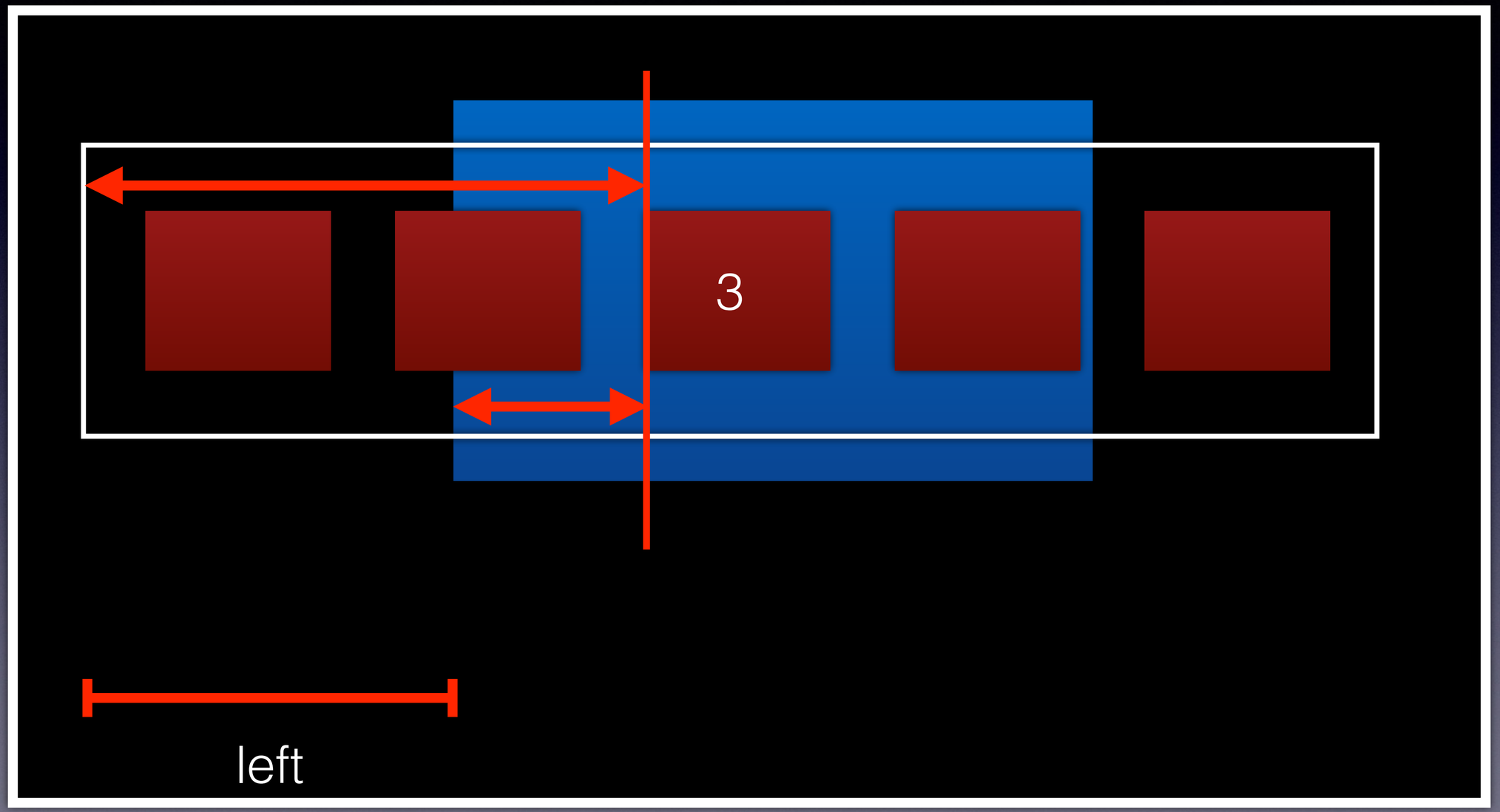
To calculate this we need to work out the distance from the image to the left-hand side of the **inner** div.



We also need to work out the distance we will want there to be from the image to the left-hand side of the **outer** div.

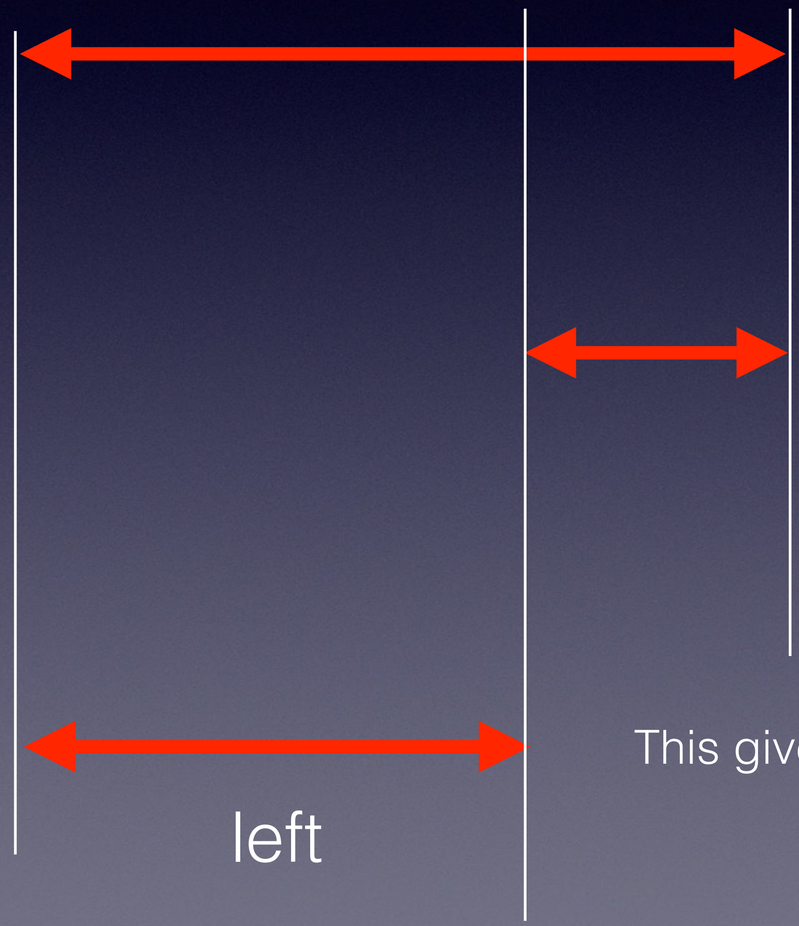


Once we have these values we can calculate the left offset position of the inner div.



Once we have these values we can calculate the left position.

get the distance from the left of the inner div

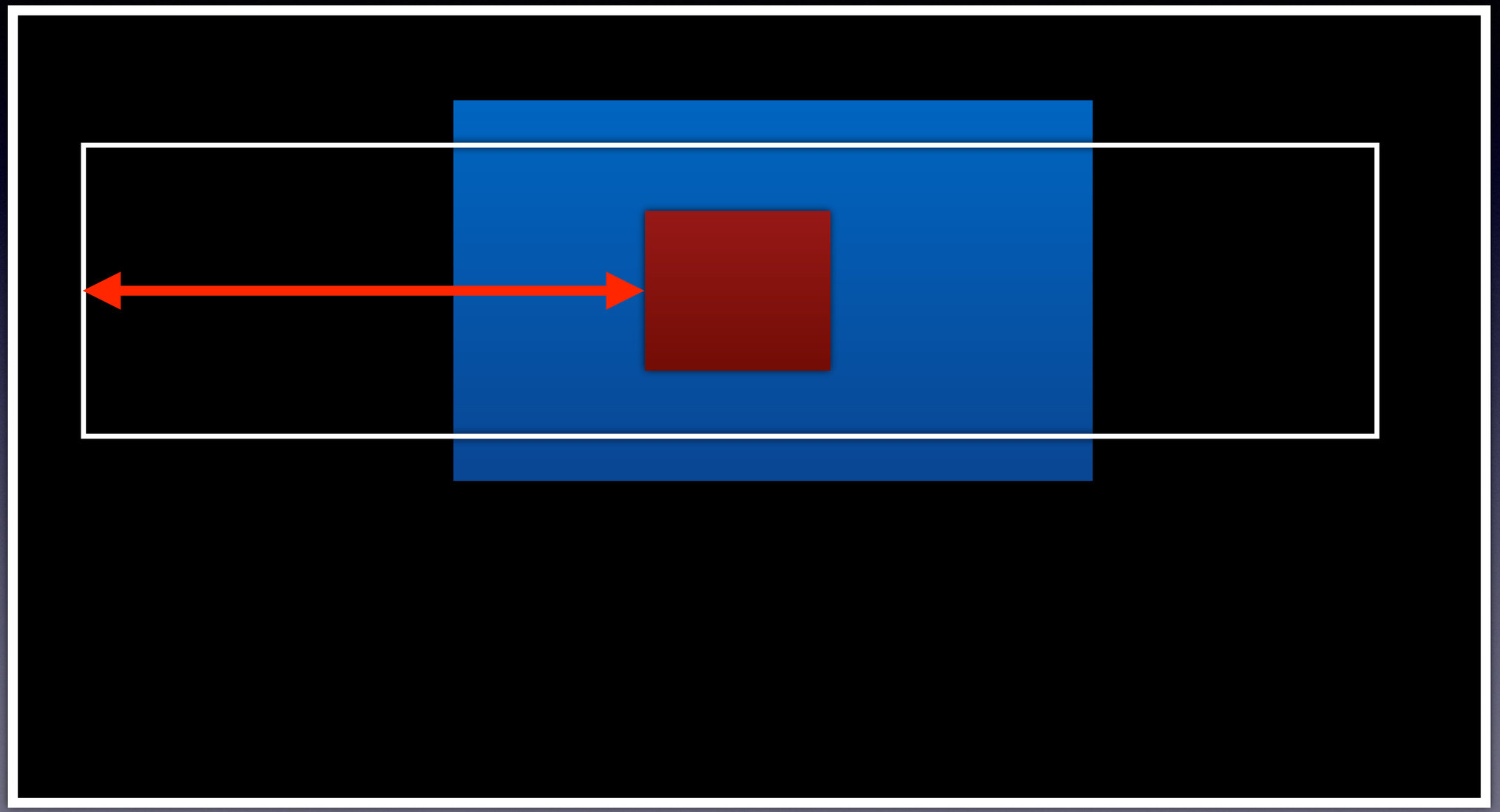


subtract it from the distance
from the left of the outer div

This gives us the **left** property for the inner div.

Step I: Get the distance from the left-hand side of the image we want to center and the left-hand side of the **inner** div.

I.e. first we need to calculate this distance



To calculate this distance we can use the **offsetLeft** property of a DOM object.

This contains the distance between the left-hand side of an element object and its containing object (i.e. the image and #innerdiv in this case)

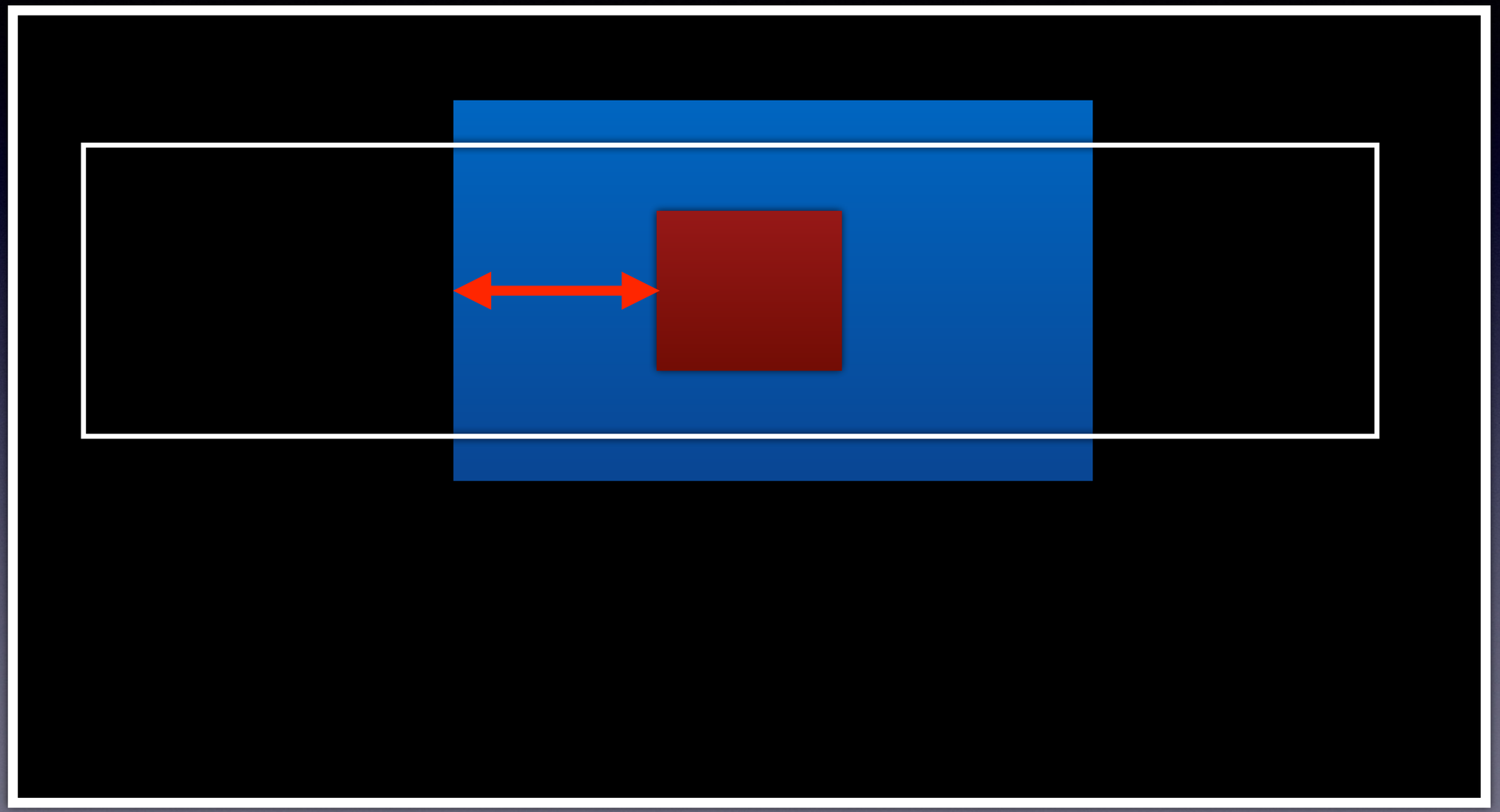
```
innerLeftOffset = document.getElementById("image").offsetLeft;
```

How you get this reference to the image object will depend on your implementation.

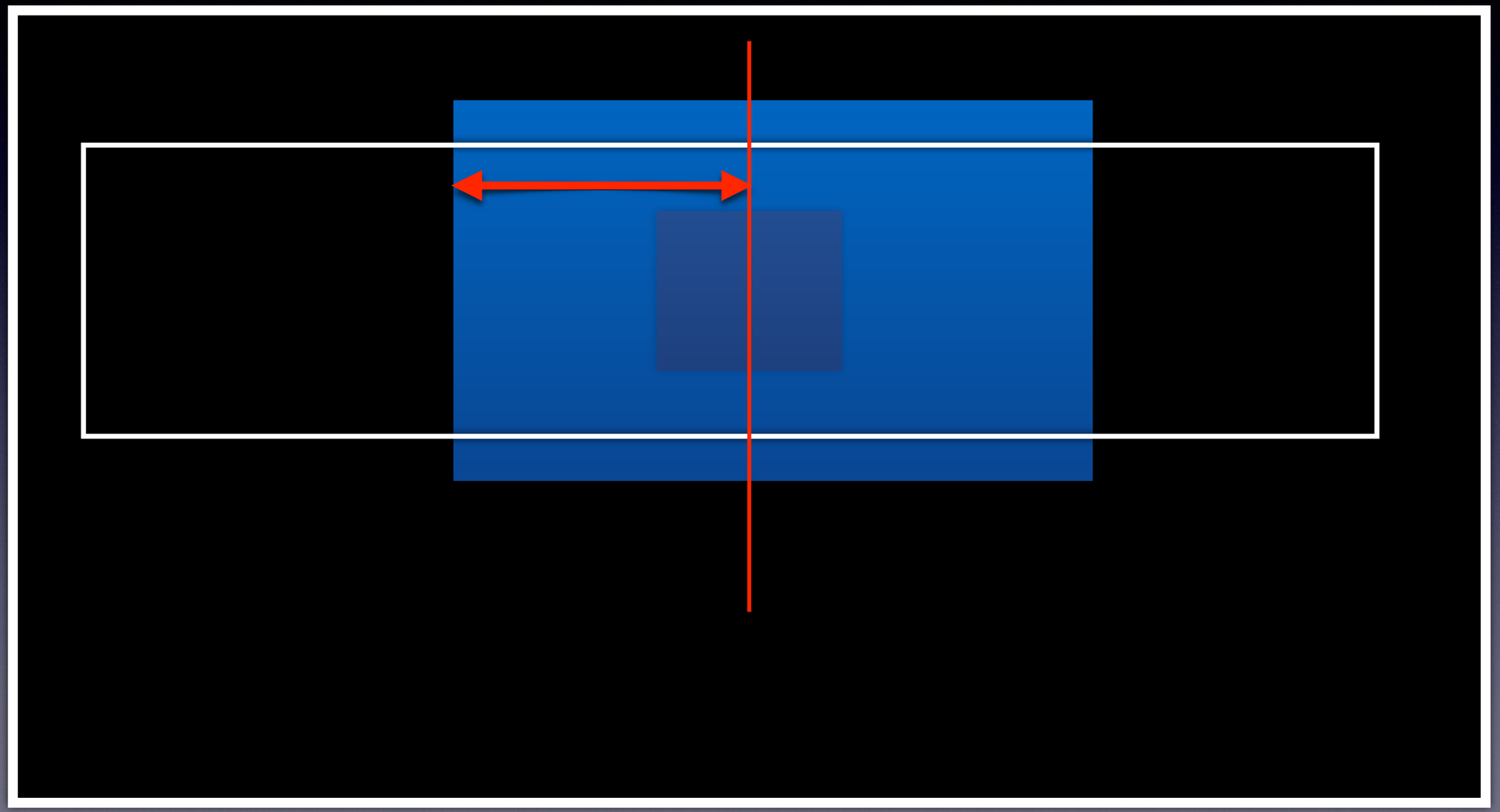
```
innerLeftOffset = document.getElementById("image").offsetLeft;
```

Step I: Get the distance from the left-hand side of the image we want to center and the left-hand side of the **outer** div.

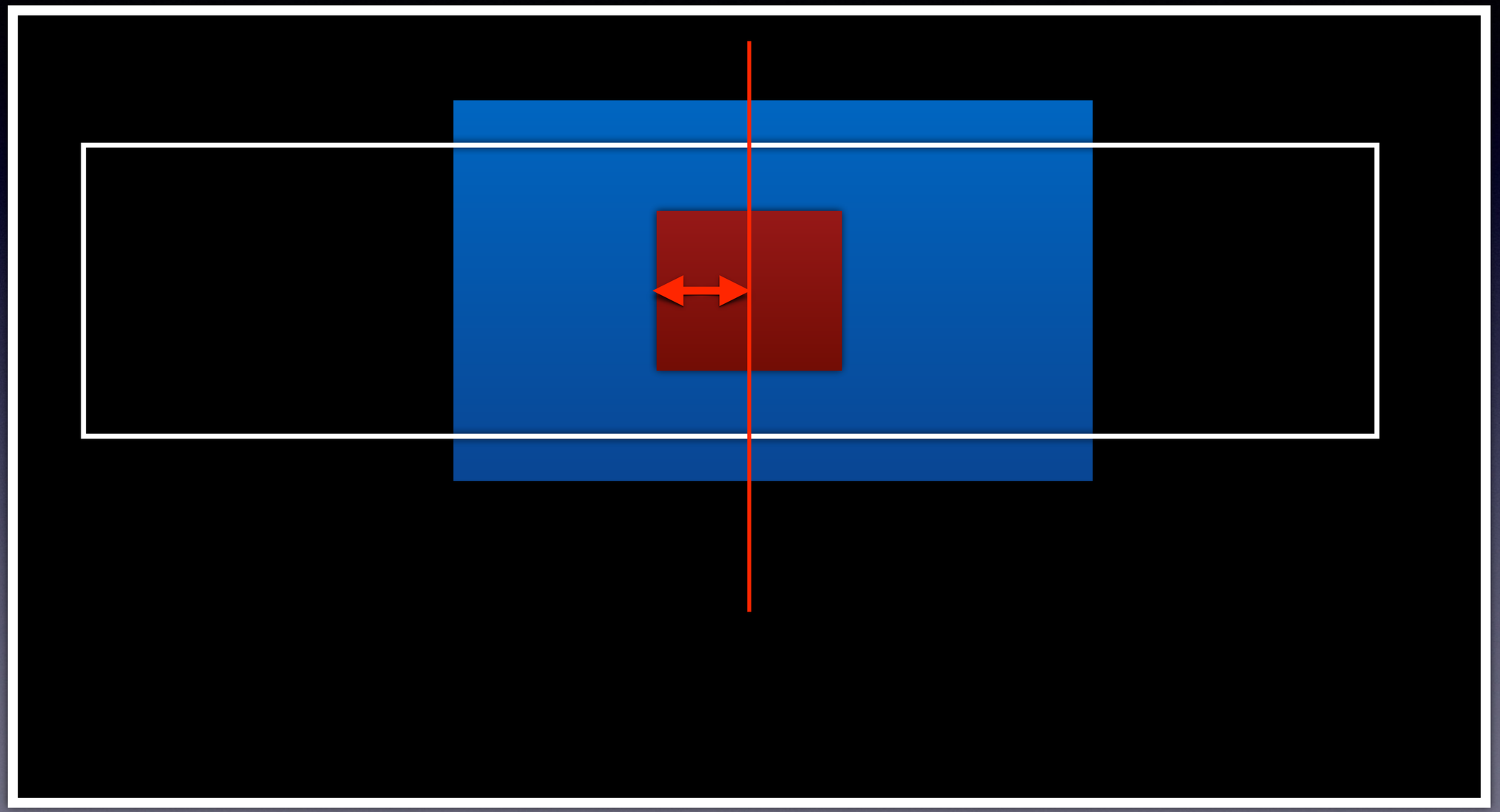
Next we want the target distance from the left-hand side of the image to the left-hand side of the outer div.



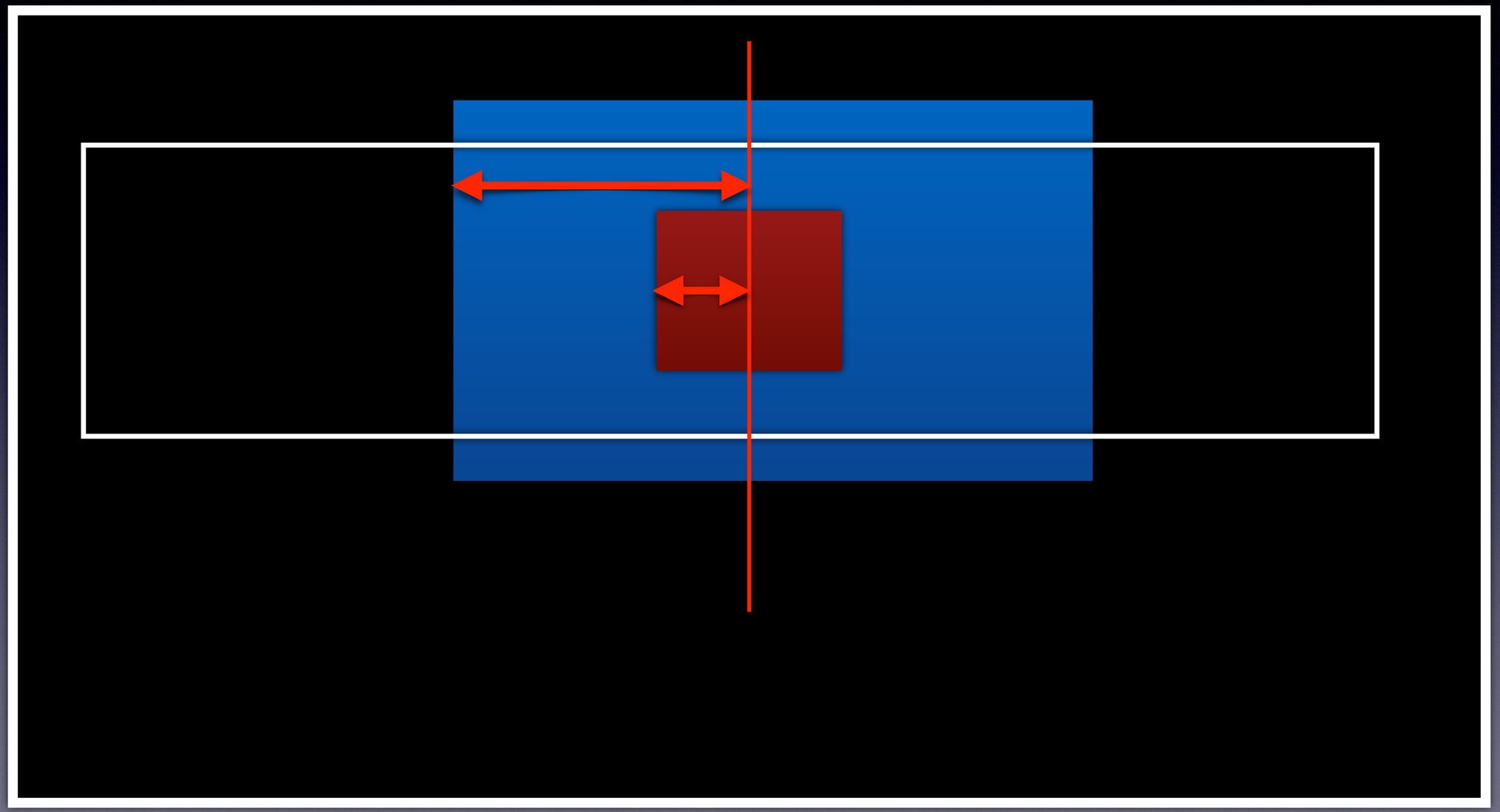
One way to do this is to calculate the distance to the center of the **outer** div (i.e. half its width)



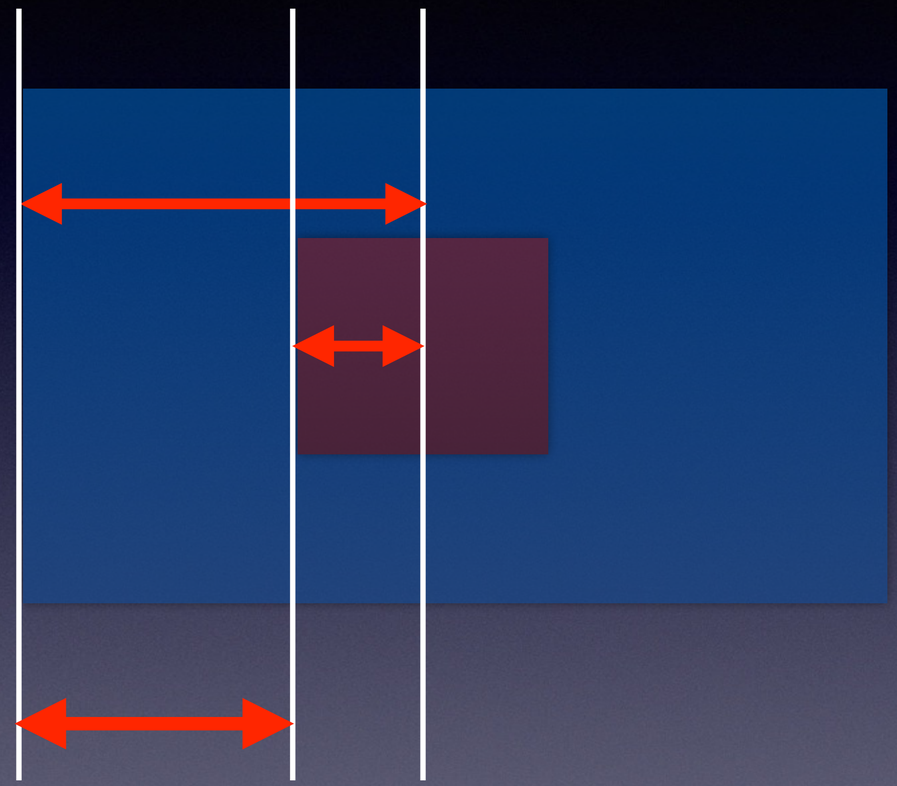
Then we calculate half the width of the image Object.



If we subtract one from the other ...



... we get the distance we want between the left-hand side of the image to the left-hand side of the outer div.



`outerLeftOffset =`

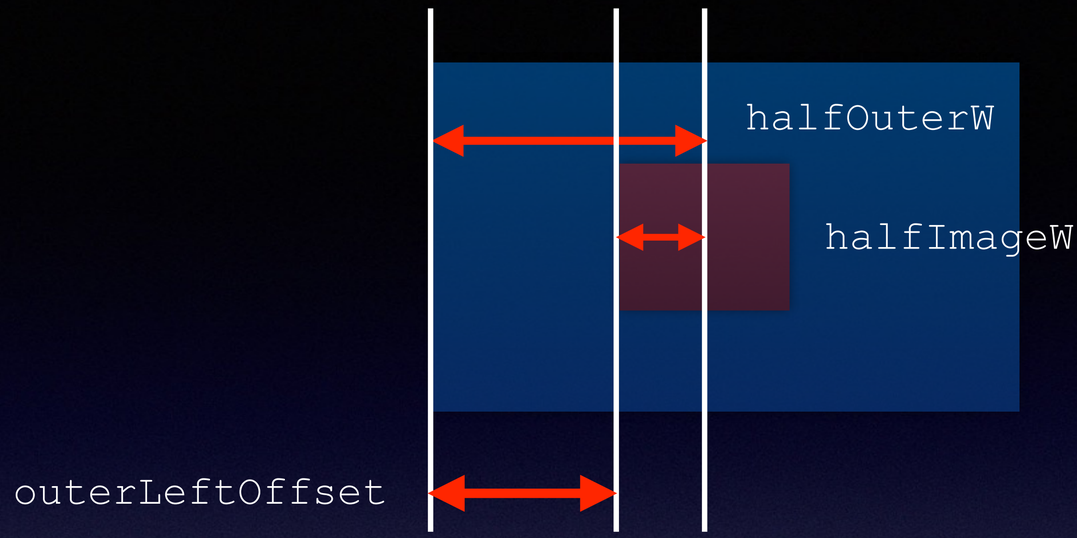
We need to calculate all these value so that our code is dynamic. I.e. we can make changes to the CSS without having to change the JavaScript, and it will can cater for different sizes of image.

To calculate the width of an element we can use **offsetWidth**.

```
halfOuterW = document.getElementById("outerdiv").offsetWidth / 2;

halfImageW = document.getElementById("image").offsetWidth / 2;

outerLeftOffset = halfOuterW - halfImageW;
```




```
halfOuterW = document.getElementById("outerdiv").offsetWidth / 2;

halfImageW = document.getElementById("image").offsetWidth / 2;

outerLeftOffset = halfOuterW - halfImageW;
```

Then our final **left** property for the inner div is calculated by subtracting one of these offsets from the other.

```
LeftPos = outerLeftOffset - innerLeftOffset;

document.getElementById("innerdiv").style.left = leftPos + "px";
```

Animation

To animate the movement of the inner div (and the various fade in/out animations) you can do one of the following:

Use CSS3 animation

Use jQuery

Use JavaScript (Keep changing a value until we reach our target value)

Loading Images: Carousel Images

When adding images to the page you need to wait until the image has actually downloaded before showing it.

```
img = create image

img.src = "image.jpg";


img.onload = function() {

   show image

};
```

Note: the code samples/ algorithms here are just suggestions (and incomplete). Your code may vary.

It's important to note that you can't calculate the width of an image until it has fully downloaded.

When initially positioning the inner div of the carousel you should wait until **all the images** are downloaded before calculating positions.

For the carousel you need to wait for all the images in to download before showing it. One way you can check for this is to count them as they load by incrementing/ decrementing a counter variable).

When each individual image is downloaded you can check how many have downloaded in total.

```
int nrOfImages = number of images

for (var i = 0; i < number of images; i++)
{
   img = create new image;

   img.src = current image filename;

   add image to carousel

   img.onload = function() {

       nrOfImages--;

       if (nrOfImages <= 0)
       {
            carousel is ready to show
       }
    };
}
```

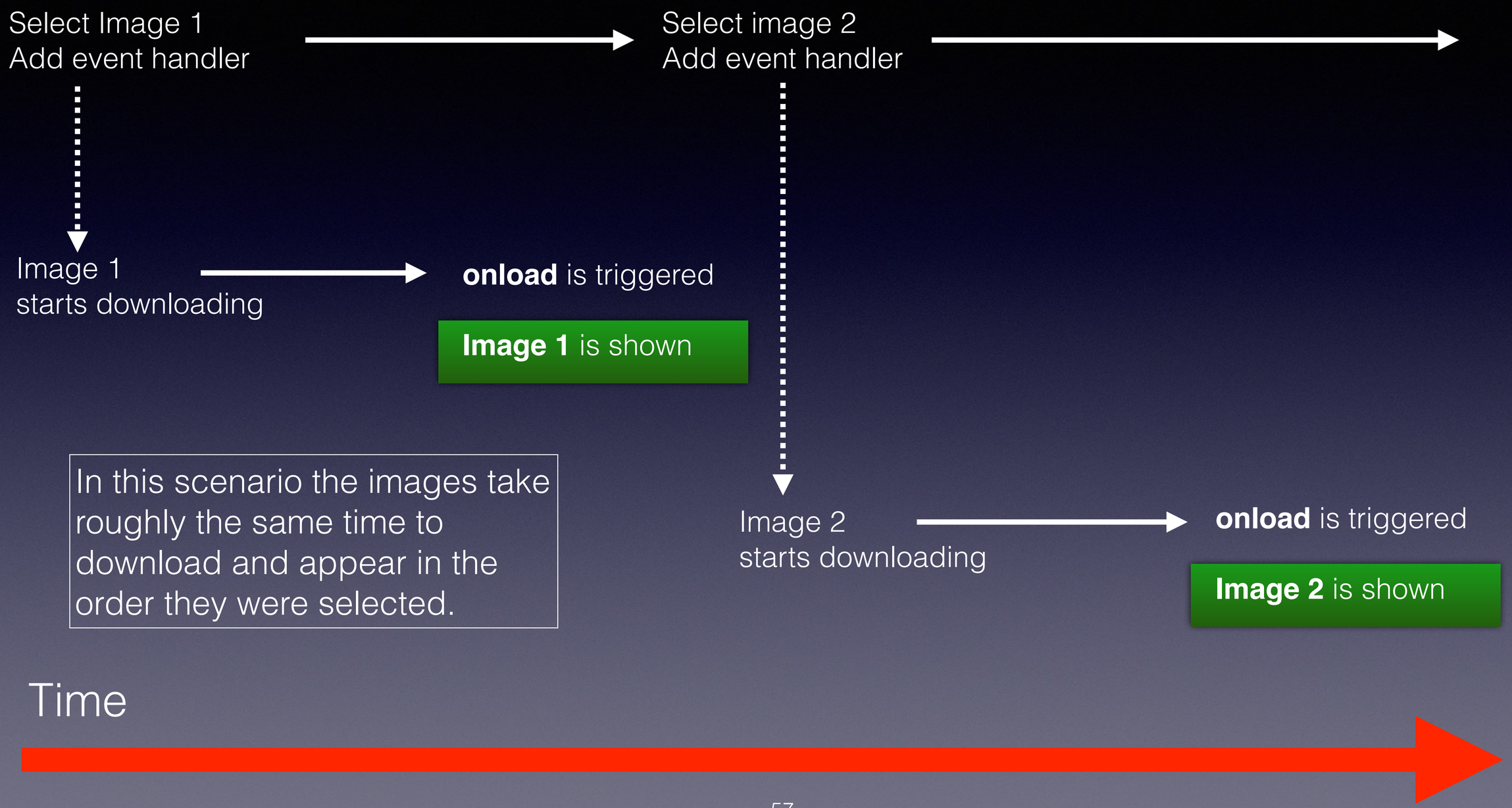
Note: We could also add a timeout function to handle situations where not all images may have download (or write code to handle error events). This will not be necessary for the purposes of this assignment.

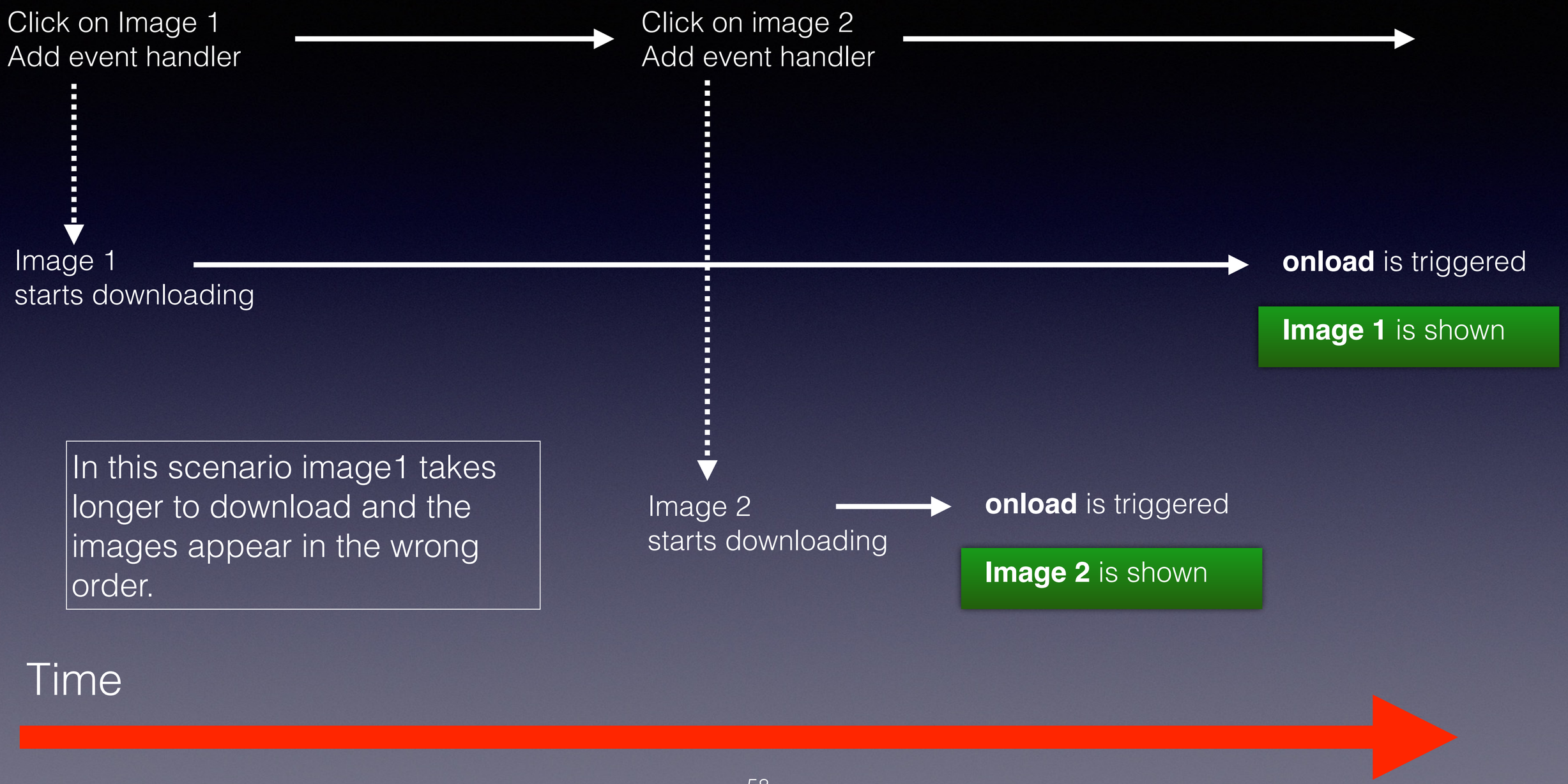
Note: There are also other ways to check if the images have downloaded. E.g. you can check the **complete** property of the the image element objects to see if they have downloaded.

Loading Images: Main Image

When selecting an image in the carousel you start to download the larger version of the image to display in the main area. We should add an event handler to show the image once it has fully downloaded.

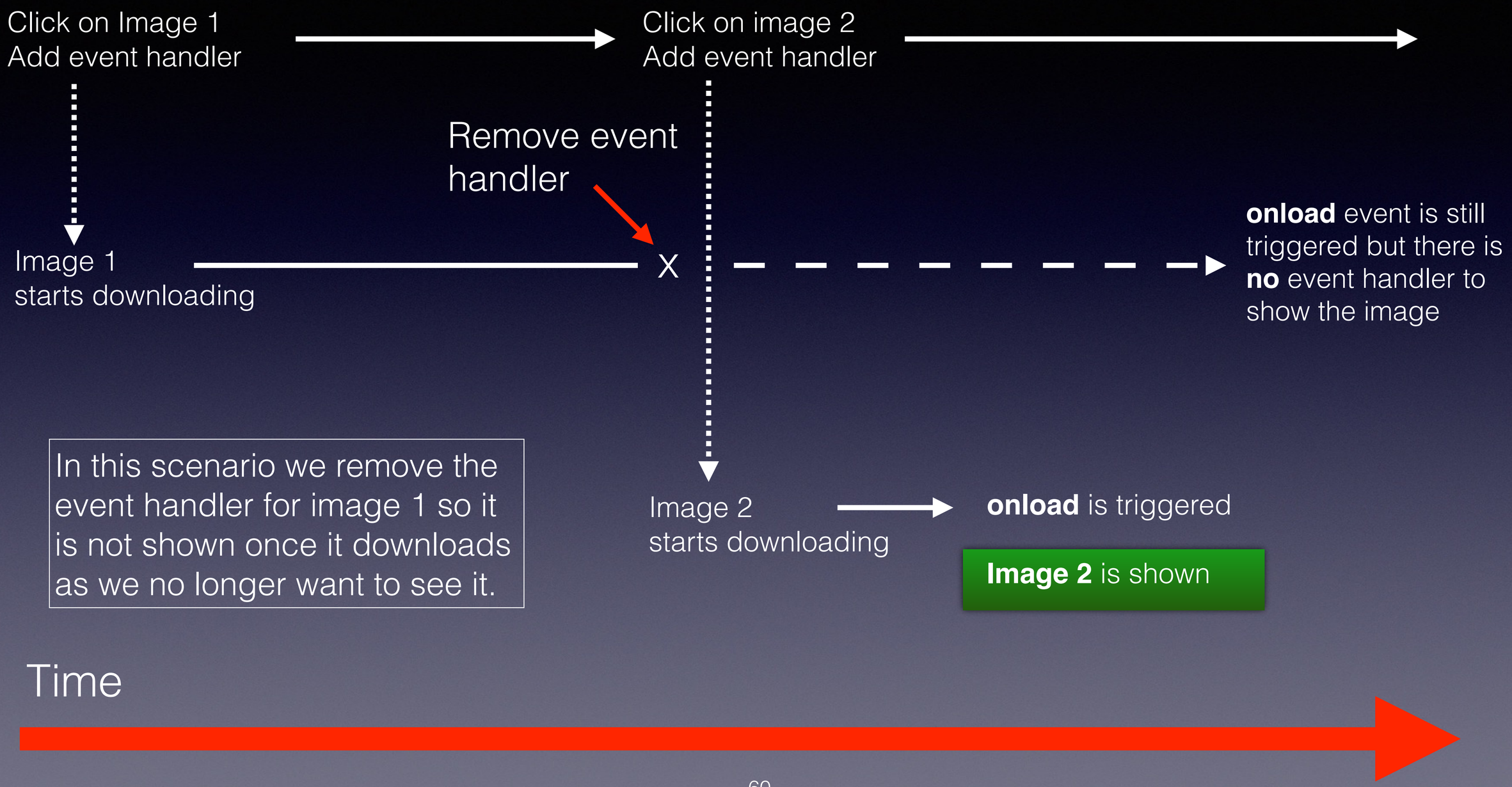
However, if you are rapidly swapping between images you need to remember that **onload** events will not necessarily be triggered in the order you added them and the images may appear in a seemingly random order.


Select Image 1
Add event handler
Select image 2
Add event handler
Image 1
starts downloading
onload is triggered
Image 1 is shown
In this scenario the images take roughly the same time to download and appear in the order they were selected.
Image 2
starts downloading
onload is triggered
Image 2 is shown
Time


Click on Image 1
Add event handler
Click on image 2
Add event handler
Image 1
starts downloading
onload is triggered
Image 1 is shown
In this scenario image1 takes longer to download and the images appear in the wrong order.
Image 2
starts downloading
onload is triggered
Image 2 is shown
Time

A solution is to remove the **onload** event handlers for all images when we select a new image to view.

Then we add an **onload** event handler only to the newly selected image. This way the most recently selected image is the only one that will be shown.


Click on Image 1
Add event handler
Click on image 2
Add event handler
Remove event handler
Image 1
starts downloading
X
onload event is still triggered but there is no event handler to show the image
In this scenario we remove the event handler for image 1 so it is not shown once it downloads as we no longer want to see it.
Image 2
starts downloading
onload is triggered
Image 2 is shown
Time

How you remove the eventHandler will depend on how you added it.

E.g.

If you added it as:

```
imageObject.onclick = function() { .... };
```

Then you can remove it with:

```
imageObject.onclick = null;
```

Misc Notes

To simplify your interface you could create an array of the data you need for each image in the carousel. Then you can use the array index to specify which image you want to display.

Then write a function that displays the image stored at a provided index. This includes moving the thumbnail to the center of the carousel, etc.

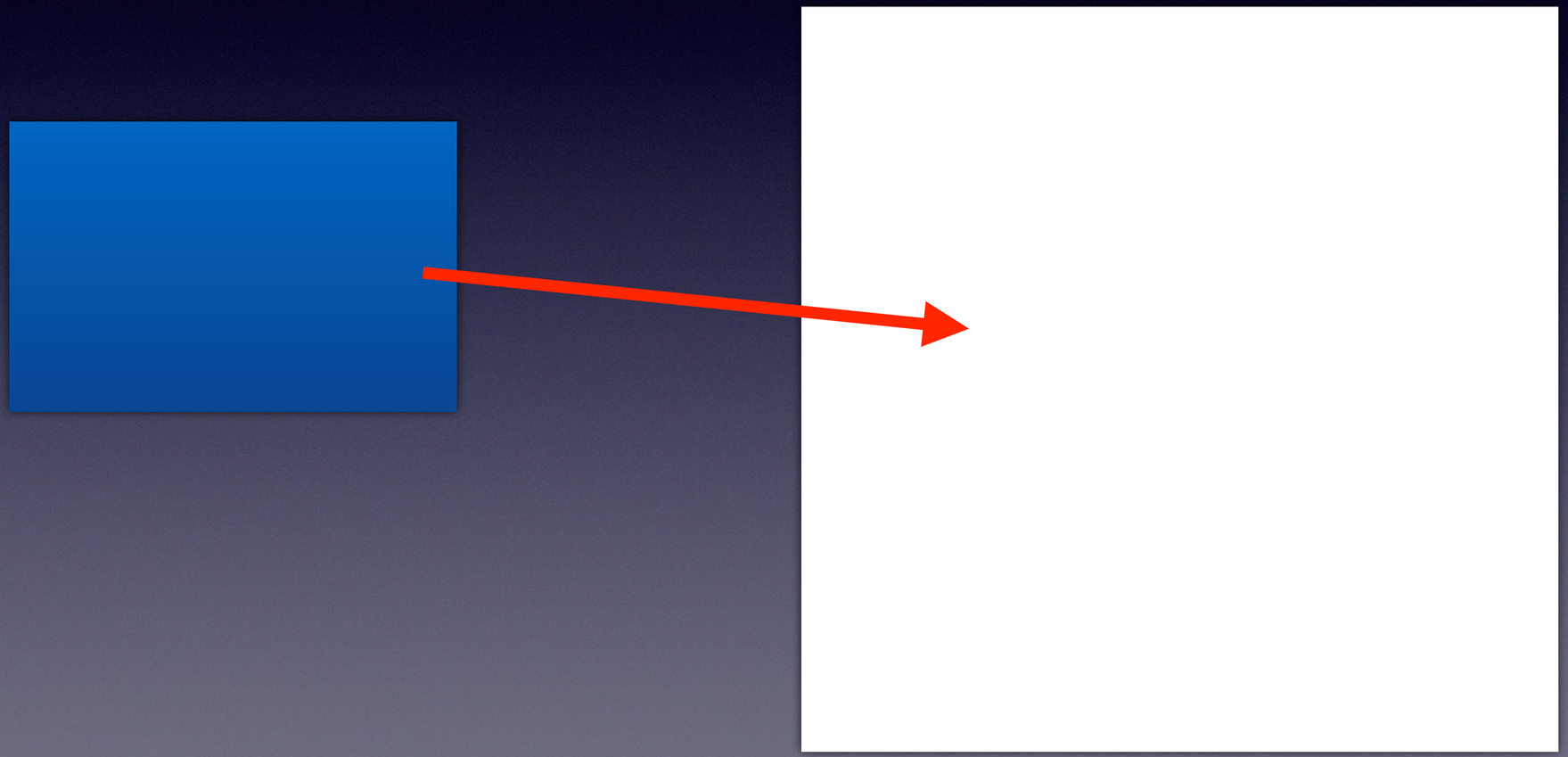
Then to select a new image you just have to calculate the new index of the Image in question and call the function.

When you click on an image it can select its own index within the array as the current index to be shown. To do this it has to know its own index.

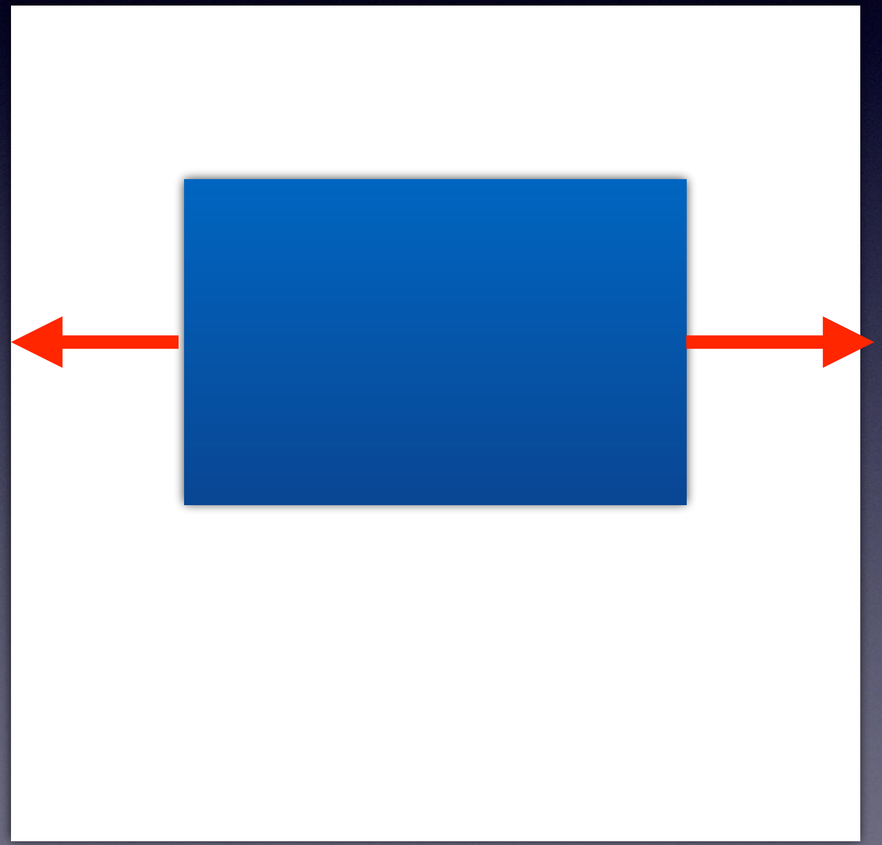
There are several solutions to this. E.g. each event handler could be passed its index when it is created. If using this approach pay special attention to the issues covered in the **Looping/Scope** notes we did in class.

Positioning Images

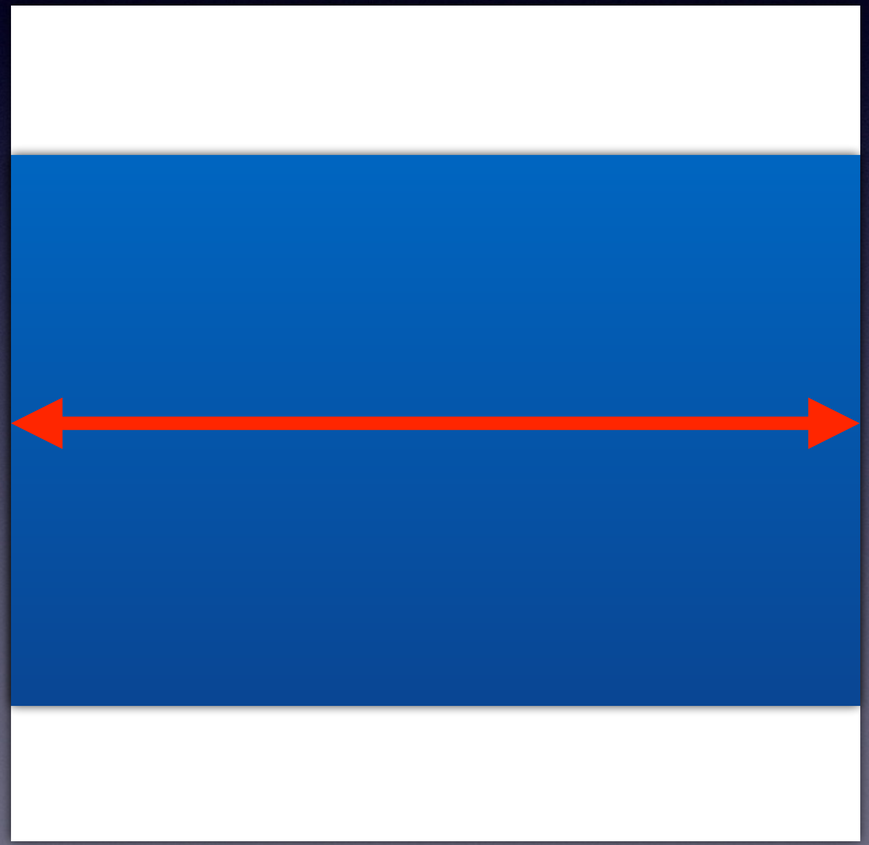
When showing an image in the designated area you need to expand or shrink it so it fills the maximum area with all of the image being visible.



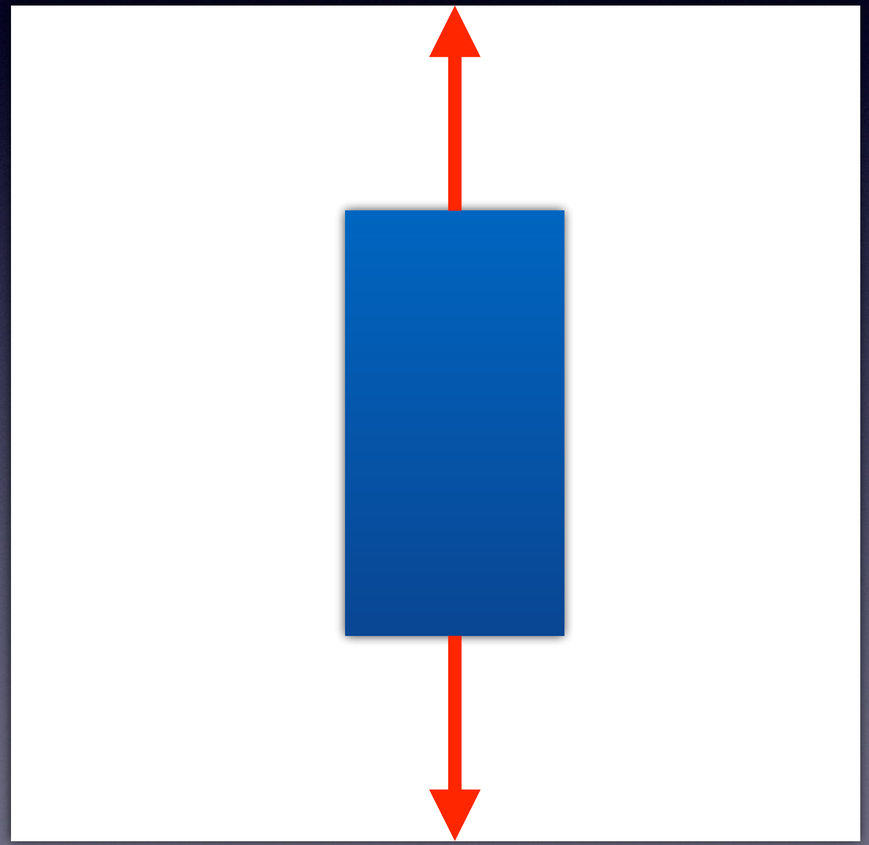
If the image is wider than it is tall then expand (or shrink) its width to 100% (or similar percentage) of its container. The height will change proportionally.



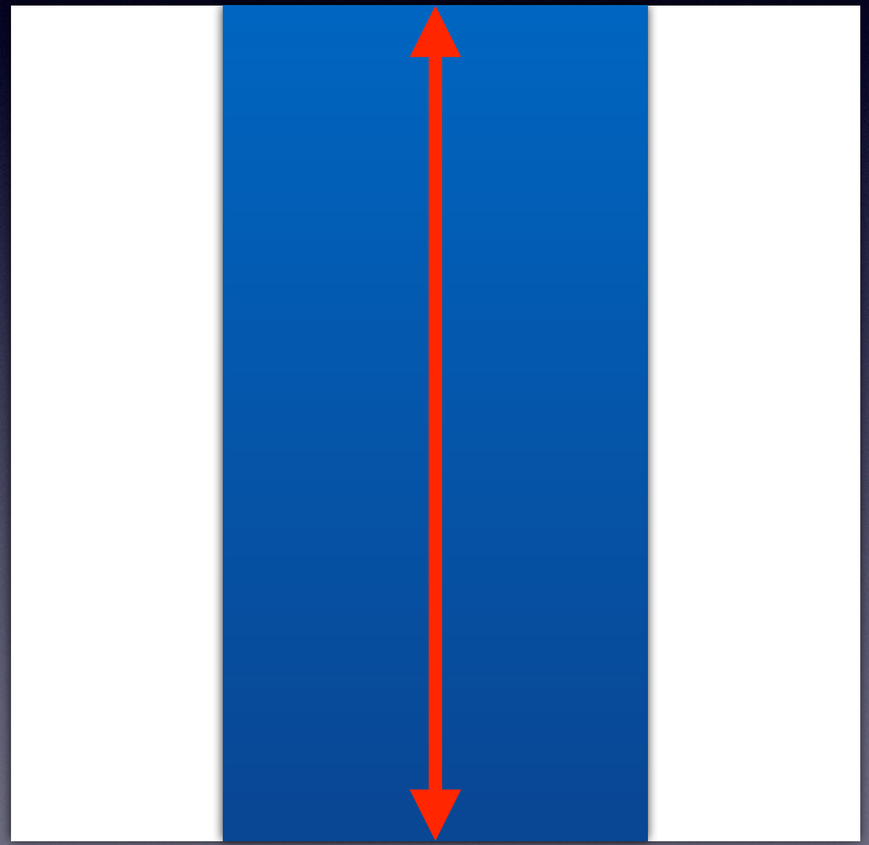
If the image is wider than it is tall then expand (or shrink) its width to 100% (or similar percentage) of its container. The height will change proportionally.



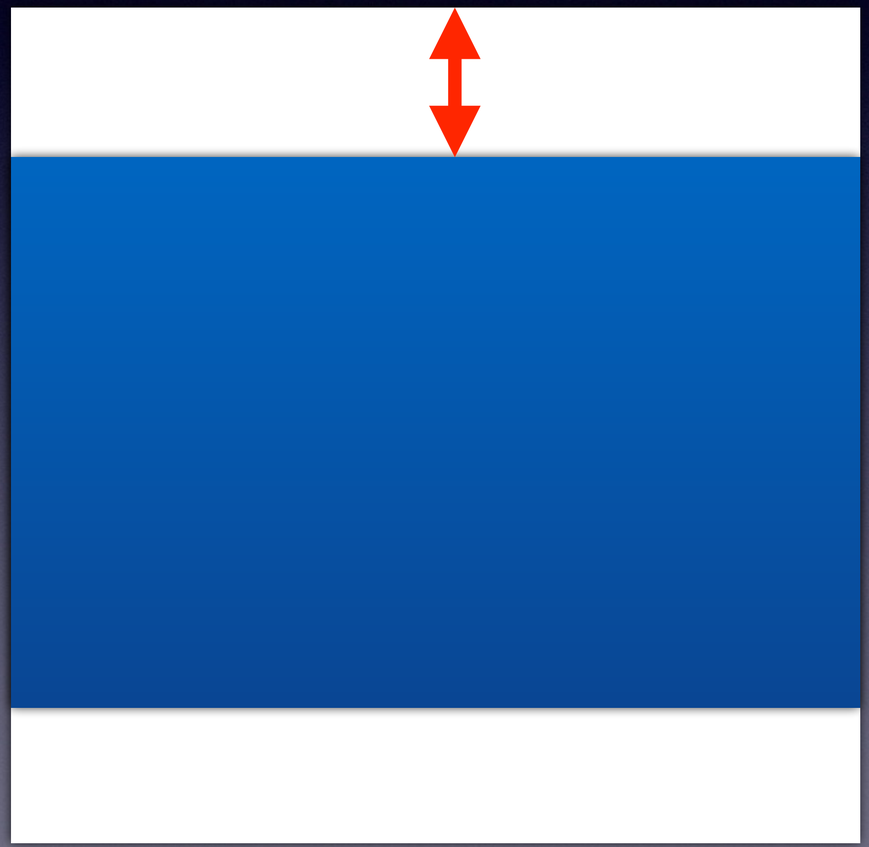
If the image is taller than it is wide then expand its height to 100% (or similar percentage) of its container. The width will change proportionally.



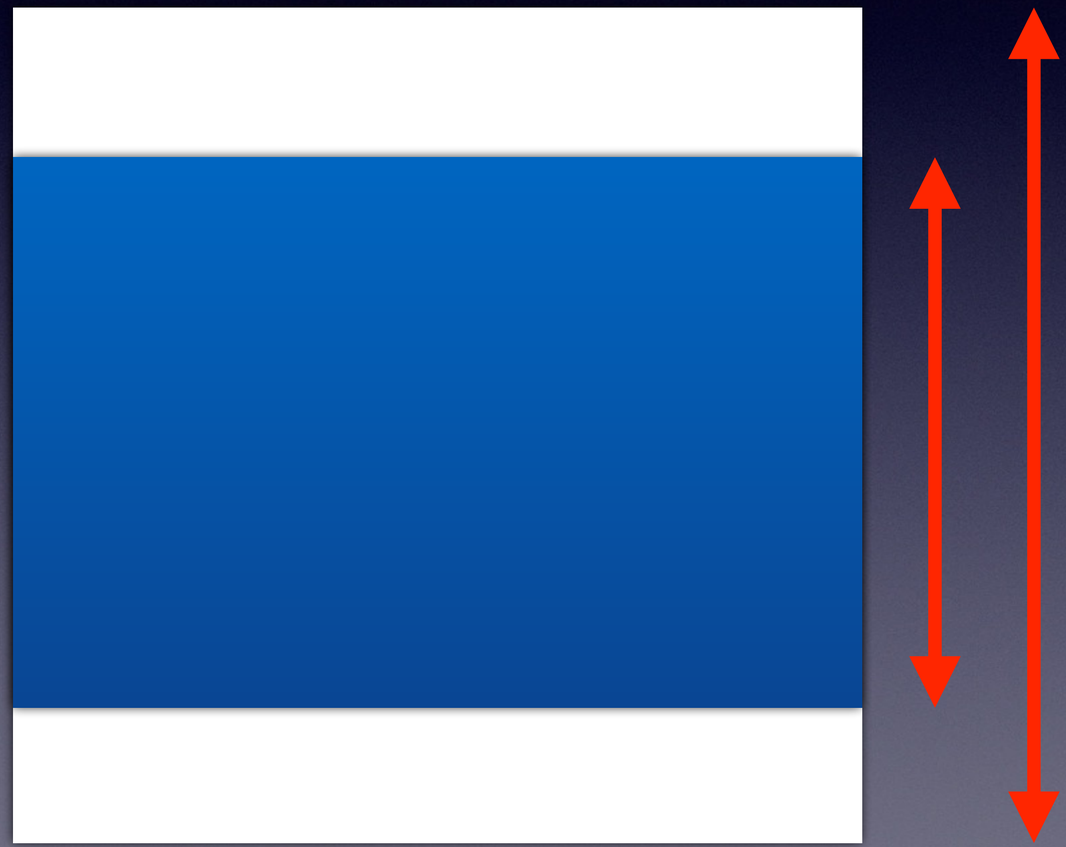
If the image is taller than it is wide then expand its height to 100% (or similar percentage) of its container. The width will change proportionally.



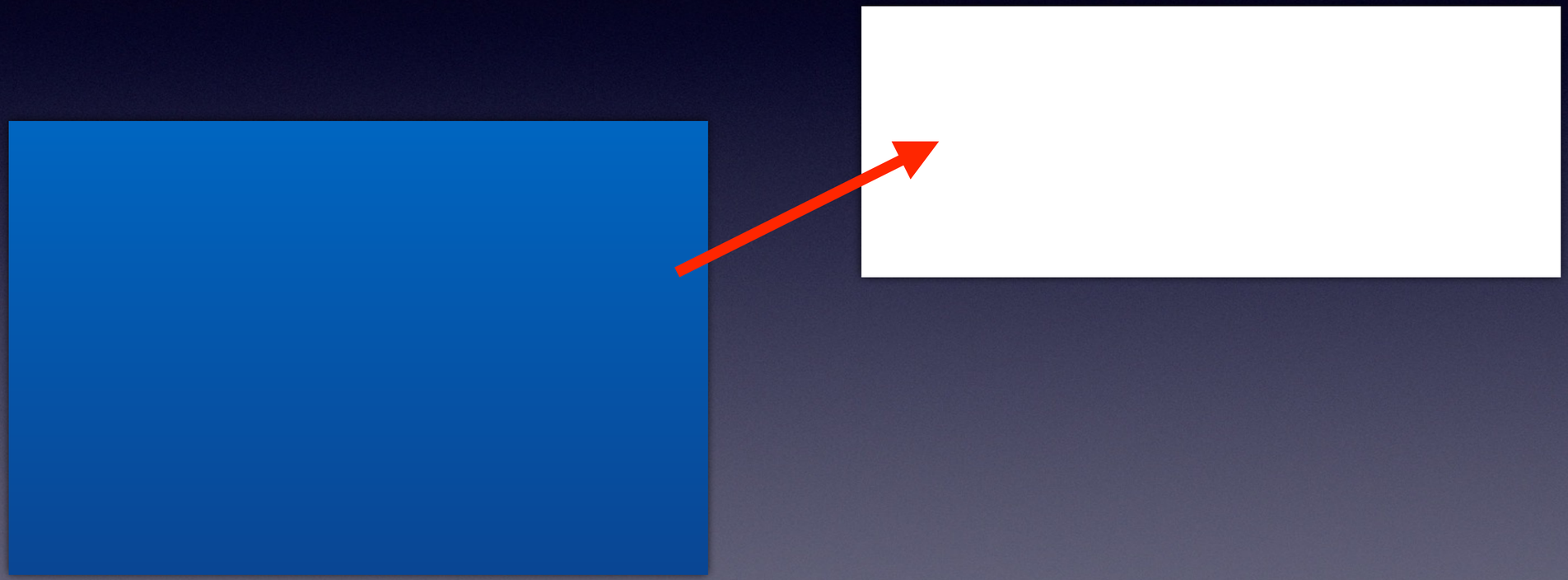
The image can be centred horizontally using **text-align: center**. It can be centred vertically by using JavaScript to calculate the **margin-top**.



You can use the **offsetHeight** property to calculate the height of the image and its container. The **margin-top** is half the difference.



Note that the aspect ratio of the container has to be taken into account.



In this case the image is too tall because the height increased in proportion to the width (which stretched to fill the display area).

I.e. only if the aspect ratio of the image is greater then the display area and the image's width is greater than its height do we set the width to be 100%.

```
var imageratio = img_width/img_height;

var displayratio = display_area_width/display_area_height;
```

In our example the aspect ratio of the display area is greater than the image so we set the **height** to be 100% instead.

In this example the aspect ratio of the display area is less than the image so we set the **width** to be 100% instead.