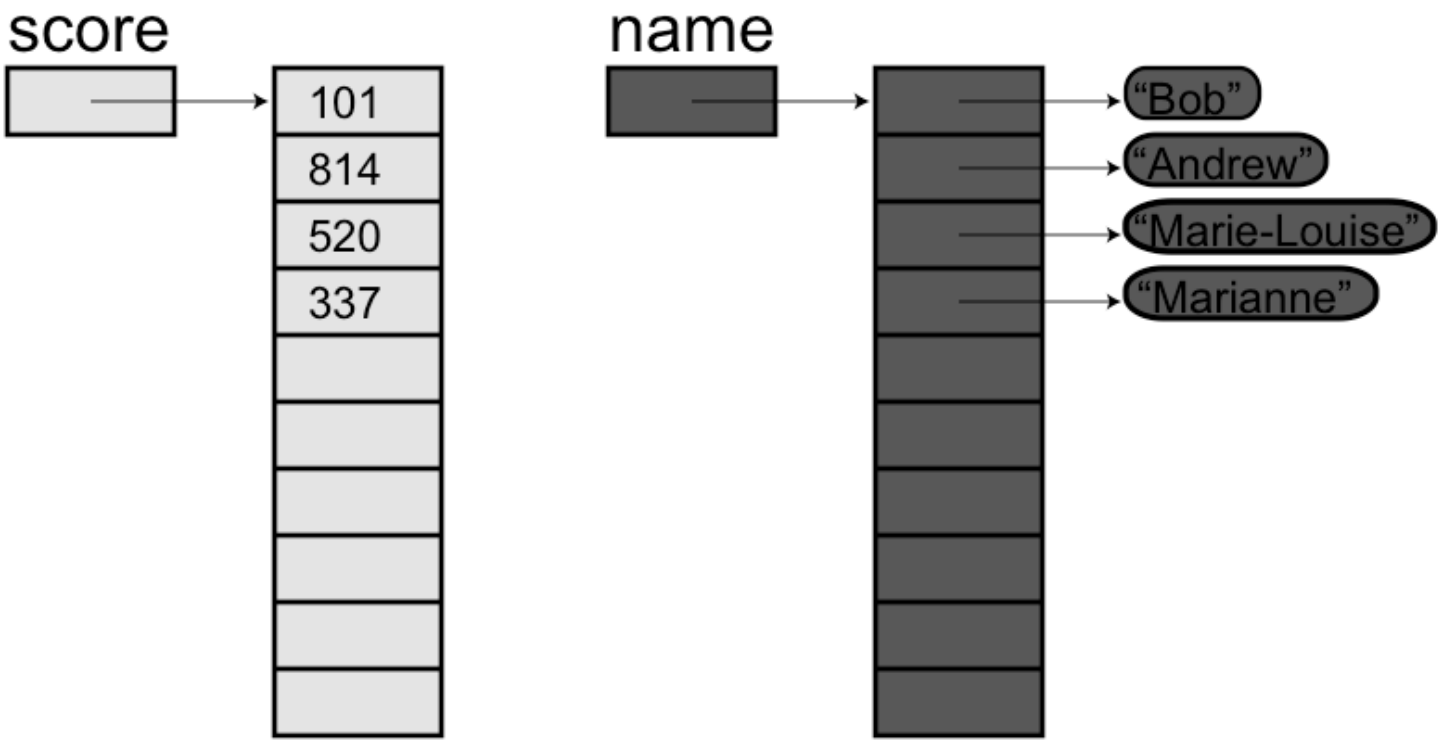
Two Dimensional Arrays and Complex Conditions

Engineering 1D04, Teaching
Session 6

Recap Arrays

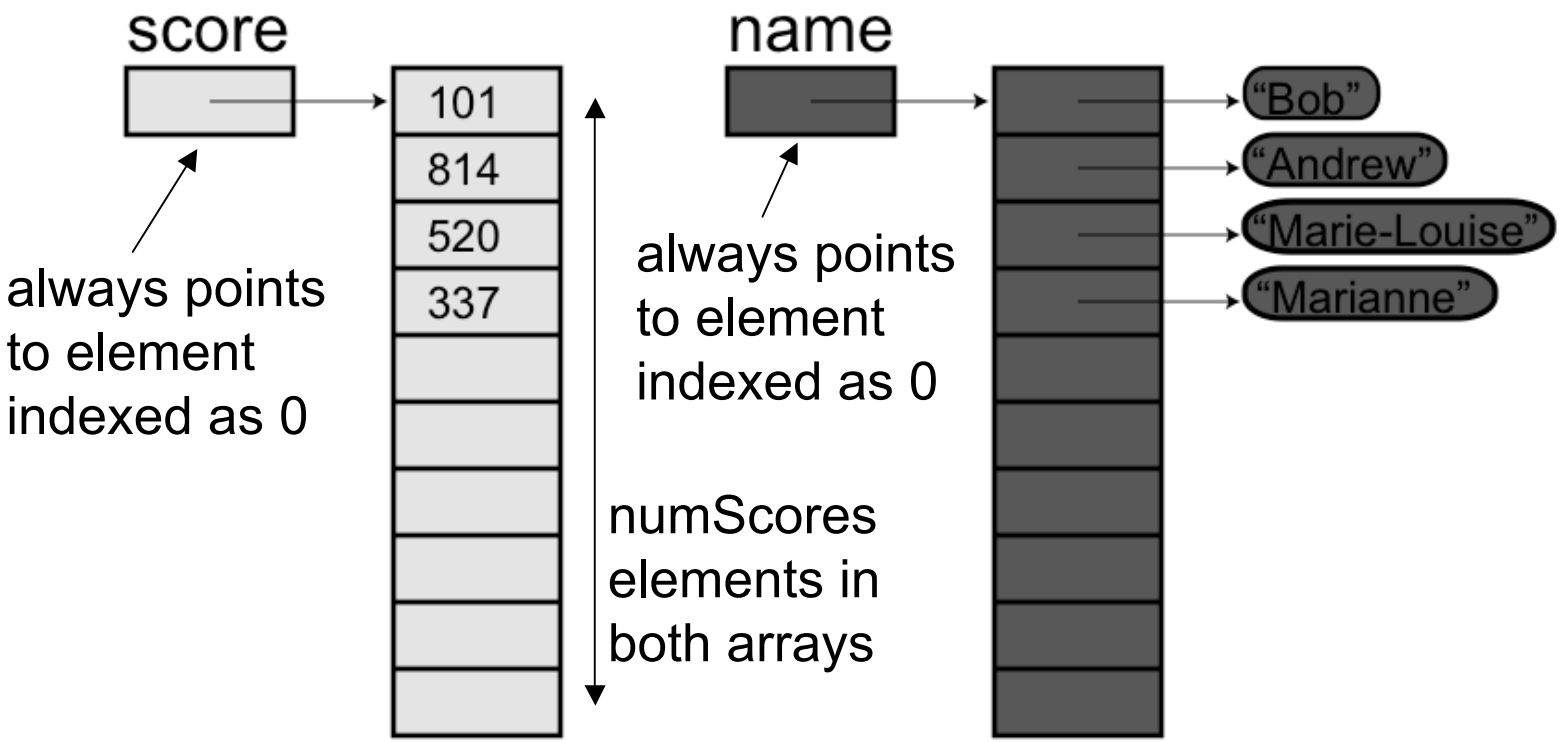
recap: Arrays

```
const int numScores = 10;

int [] score = new int[numScores];
string [] name = new string[numScores];
```



recap: Arrays

```
const int numScores = 10;

int [] score = new int[numScores];
string [] name = new string[numScores];
```



recap: Arrays

- How do we work with elements in the array?

```
int i, total;

total = 0;
for (i = 0; i < numScores; i++)
{
     //do what we want to with element i
     total += score[i];  //for example
}
```

recap: Arrays

- Alternative

```
int i, total;

total = 0;
i = 0;
while (i < numScores)
{
     //do what we want to with element i
     total += score[i];  //for example
     i++;
}
```

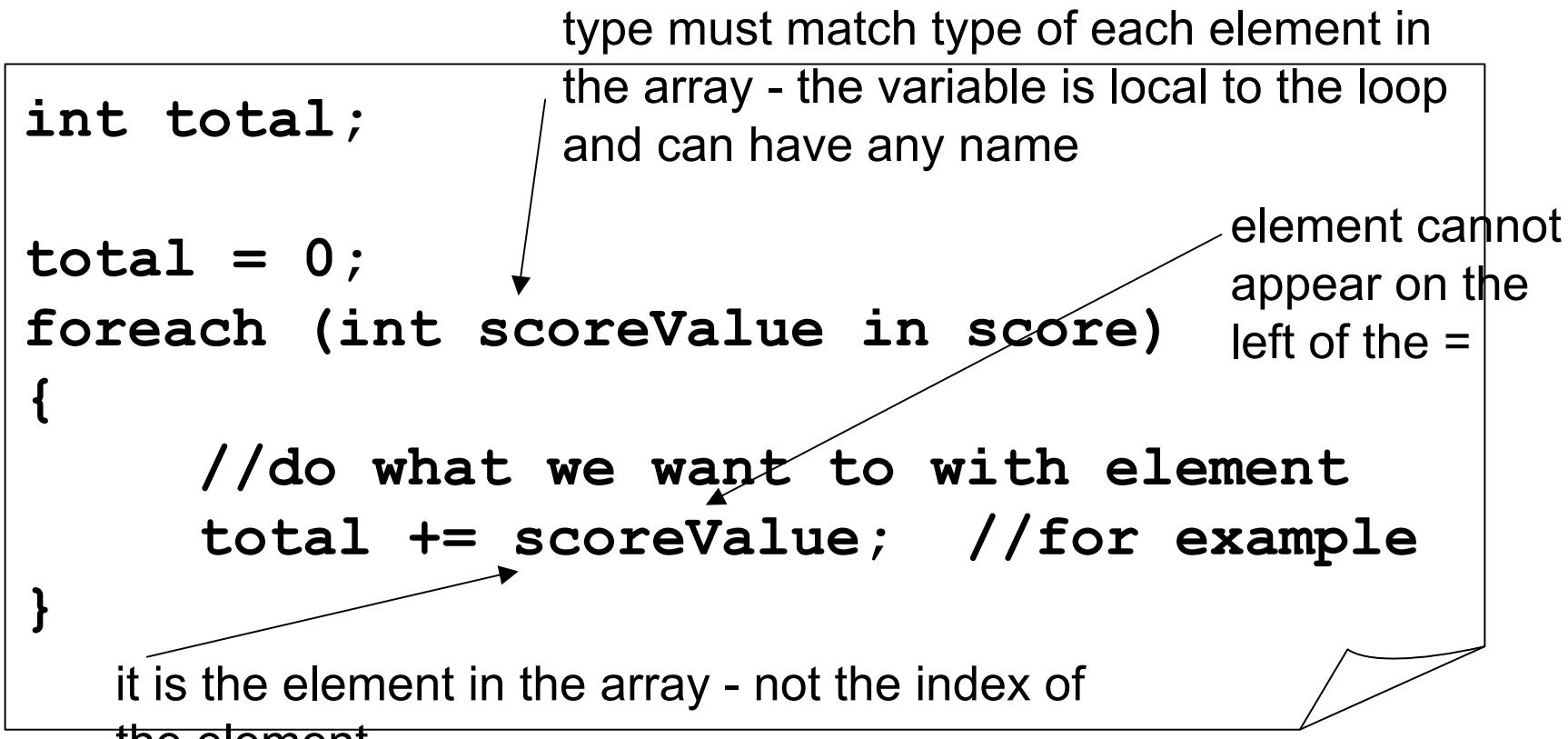
A new kind of loop

- Another (very safe) construct for loops.

```
int total;

total = 0;
foreach (int scoreValue in score)
{
     //do what we want to with element
     total += scoreValue;  //for example
}
```

A new kind of loop

- Another (very safe) construct for loops.



Two Dimensional Arrays

Two-Dimensional Arrays

- A very common data structure is a 2-D array. A mathematical matrix is a good example.

$$\begin{matrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{matrix}$$

traditionally $a_{i,j}$ where i gives the row and j gives the column

Two-Dimensional Arrays

- Let us assume that we have an array of this form, where each element is an integer.
- What algorithm could we construct to total each row and each column?

$$\begin{array}{cccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{array}$$

Two-Dimensional Arrays

- Let us assume that we have an array of this form, where each element is an integer.
- What algorithm could we construct to total each row and each column?

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	r_1
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	r_2
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	r_3
c_1	c_2	c_3	c_4	

The first thing to realize is we are trying to calculate elements of two new 1-D arrays, r and c. Each element of r and each element of c is an integer.

Two-Dimensional Arrays

- Let us assume that we have an array of this form, where each element is an integer.
- What algorithm could we construct to total each row and each column?

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$r_1 = a_{1,1}+a_{1,2}+a_{1,3}+a_{1,4}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$r_2 = a_{2,1}+a_{2,2}+a_{2,3}+a_{2,4}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$r_3 = a_{3,1}+a_{3,2}+a_{3,3}+a_{3,4}$
c_1	c_2	c_3	c_4	

Two-Dimensional Arrays

- Let us assume that we have an array of this form, where each element is an integer.
- What algorithm could we construct to total each row and each column?

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$
c_1	c_2	c_3	c_4

for i = 1,..,3

$r_i = a_{i,1}+a_{i,2}+a_{i,3}+a_{i,4}$

Two-Dimensional Arrays

- Let us assume that we have an array of this form, where each element is an integer.
- What algorithm could we construct to total each row and each column?

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$
c_1	c_2	c_3	c_4

```
for i = 1,..,3
    r_i = 0
    for j = 1,2,..,4
        r_i = r_i + a_i,j
```

Two-Dimensional Arrays

- Let us assume that we have an array of this form, where each element is an integer.
- What algorithm could we construct to total each row and each column?

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$
c_1	c_2	c_3	c_4

```
for i = 1,..,3
    r_i = 0
    for j = 1,2,..,4
        r_i = r_i + a_i,j
```

So, what about c?

Two-Dimensional Arrays

- Total each row and each column?

$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,4}$

$a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$

$a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $a_{3,4}$

```
for i = 1,..,3
    r_i = 0
    for j = 1,2,..,4
        r_i = r_i + a_i,j
```

```
for j = 1,2,..,4
    c_j = 0
    for i = 1,..,3
        c_j = c_j + a_i,j
```

Two-Dimensional Arrays

- Total each row and each column?

$$\begin{array}{cccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{array}$$

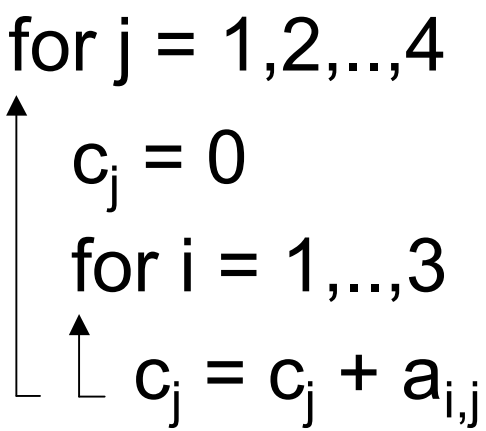
```
for i = 1,..,3
    r_i = 0
    for j = 1,2,..,4
        r_i = r_i + a_i,j
```

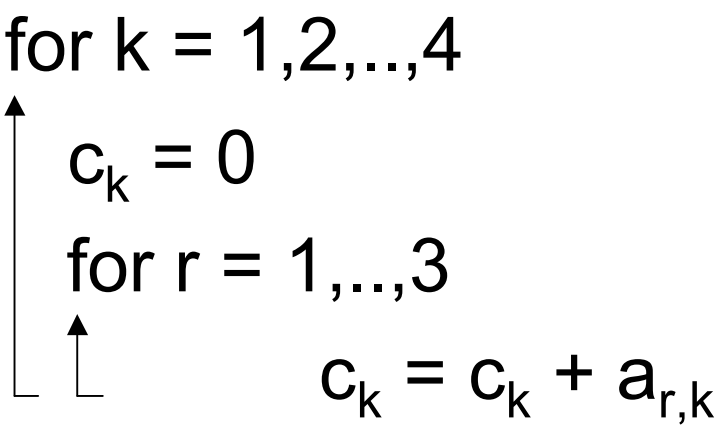
```
for j = 1,2,..,4
    c_j = 0
    for i = 1,..,3
        c_j = c_j + a_i,j
```

These are called nested loops

Two-Dimensional Arrays

- Before we see how we implement 2-D arrays in C# - a simple question.
- Are these two algorithms equal?





Two-Dimensional Arrays

- Before we see how we implement 2-D arrays in C# - a simple question.
- Are these two algorithms equal?

Yes they are! The names of the indices are irrelevant - they are often called “dummy indices”. Step through the algorithms to see …

```
for j = 1,2,..,4
    c_j = 0
    for i = 1,..,3
        c_j = c_j + a_i,j
```

```
for k = 1,2,..,4
    c_k = 0
    for r = 1,..,3
        c_k = c_k + a_r,k
```

Two-Dimensional Arrays

- We can also generalize the algorithms for n rows and m columns, for example

Sum columns

```
for j = 1,2,..,m
    c_j = 0
    for i = 1,2,..,n
        c_j = c_j + a_i,j
```

Sum rows

```
for i = 1,2,..,n
    r_i = 0
    for j = 1,2,..,m
        r_i = r_i + a_i,j
```

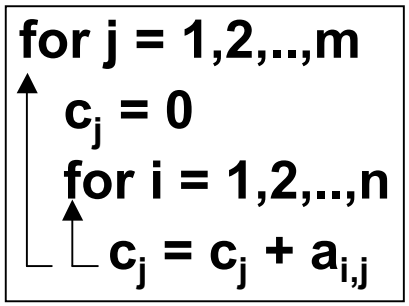
Two-Dimensional Arrays

- The only new concept really is how to declare multi-dimension arrays.
- We declare our 2-D array by:

```
const int m = 4;
const int n = 3;

int [ , ] a = new int [n, m];
```

Two-Dimensional Arrays

- Then we implement the algorithm:



Adding the rows is obviously very similar - try it yourself

```
const int m = 4;
const int n = 3;
int [ , ] a = new int [n, m];
int [] c = new int[m];
//assume values entered in a
for (j = 0; j < m; j++)
{
   c[j] = 0;
   for (i = 0; i < n; i++)
   {
      c[j] += a[i,j]
   }
}
```

Complex Conditions

recap: Conditions

- Write a program that determines if an integer year is a leap year.

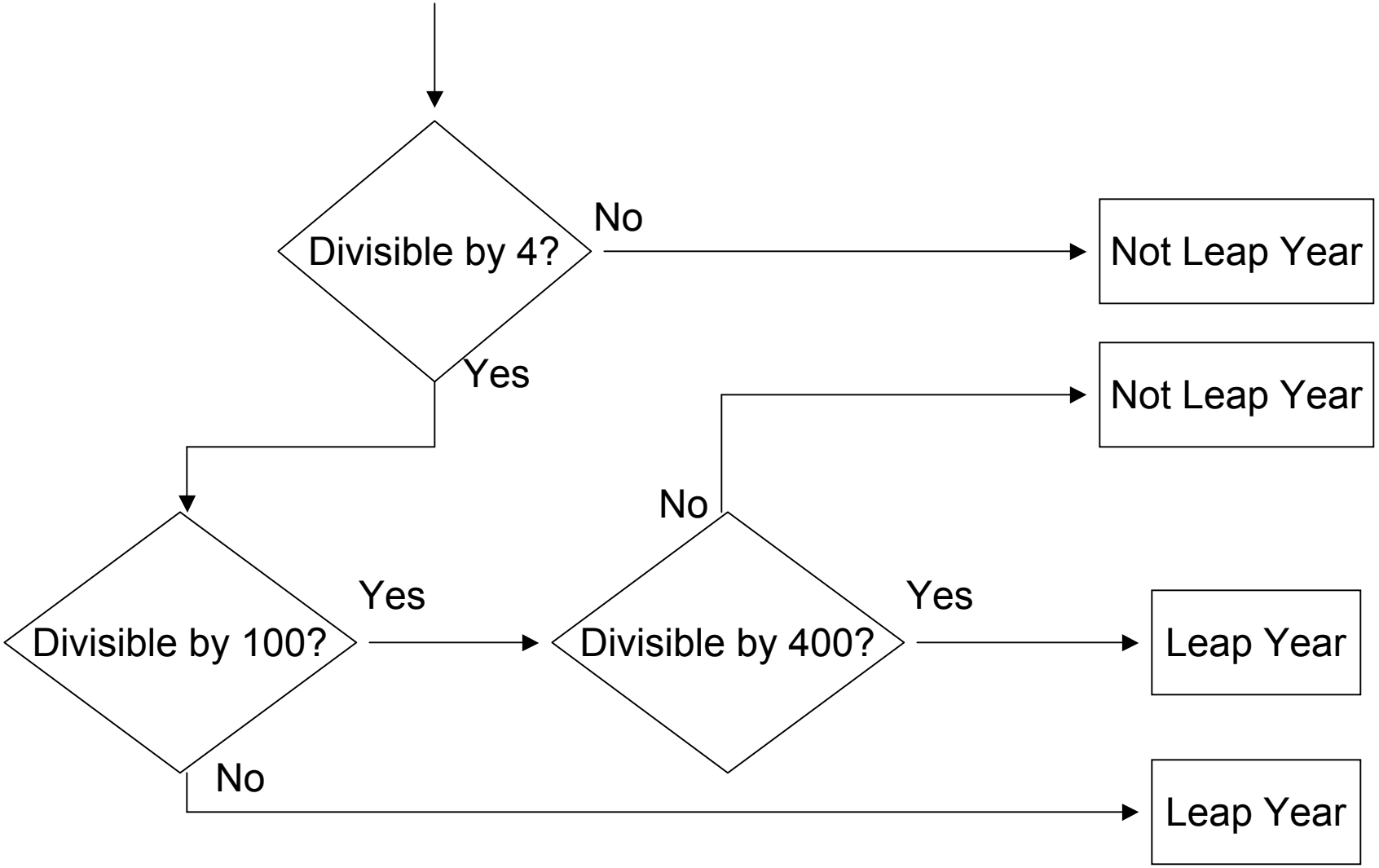
- What conditions are there on being a leap year?

recap: Conditions

- Write a program that determines if an integer year is a leap year.

- For a year to be a leap year:
  - It must be divisible by 4
  - It must not be divisible by 100
  - However, if it is divisible by 100 and 400 it is a leap year

Complex Conditions



Complex Conditions

```
private void button1_Click(object sender, EventArgs e)
{
   int year = Convert.ToInt32(inputBox.Text);
   if ((year % 4) == 0){
      if ((year % 100) == 0){
         if ((year % 400) == 0){
            outputBox.Text = "Leap Year";
         }else {
            outputBox.Text = "Not Leap Year";
         }
      }else {
         outputBox.Text = "Leap Year";
      }
   }else {
      outputBox.Text = "Not Leap Year";
   }
}
```

alternative convention (saves space)

it's a bit of a mess ☹

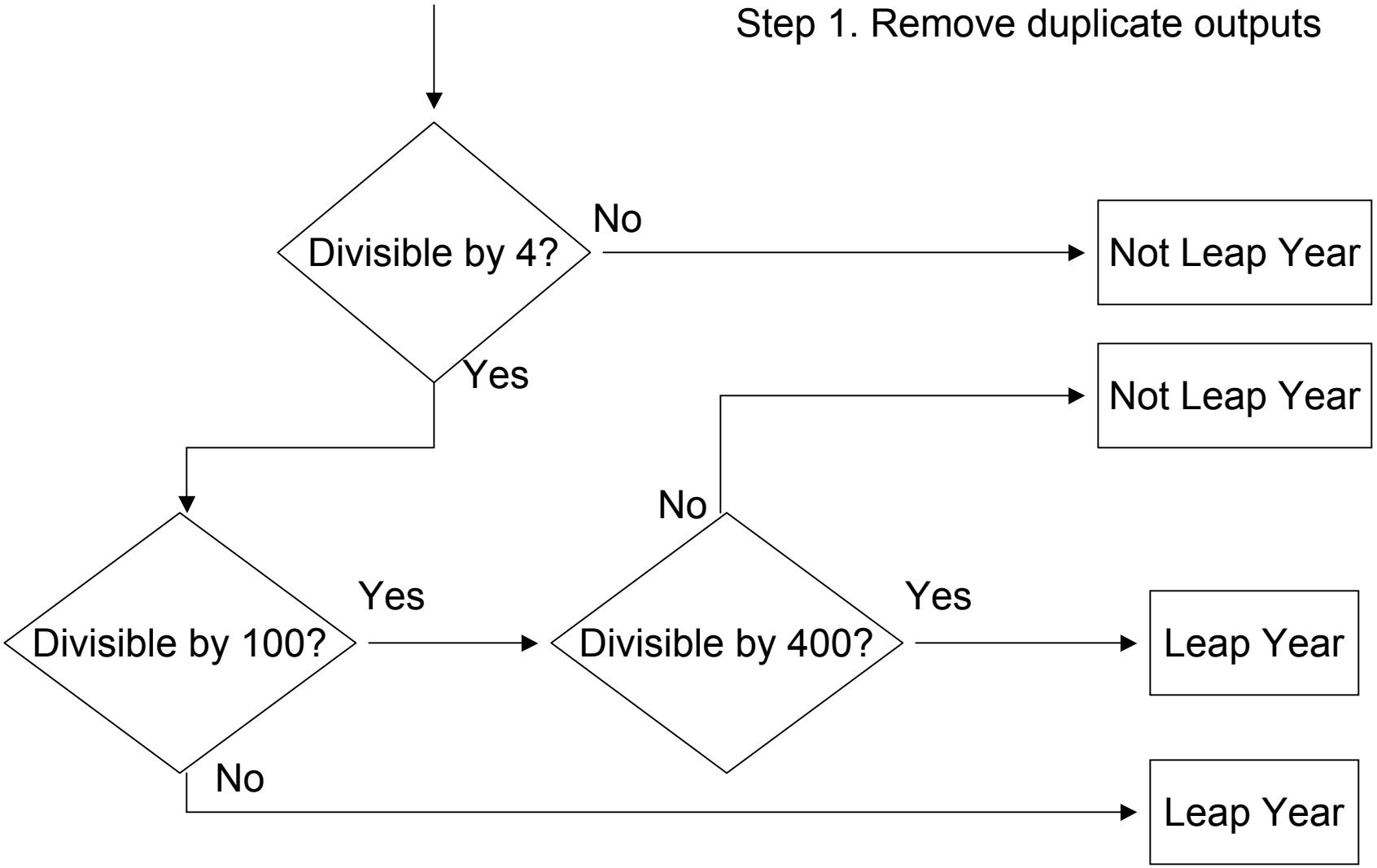
Complex Conditions

- One big problem is that there are multiple paths to get to the same result.
- Another is that we have unnecessarily nested if statements.

- *This is a good example of why we need to work on analysis and design before coding.*

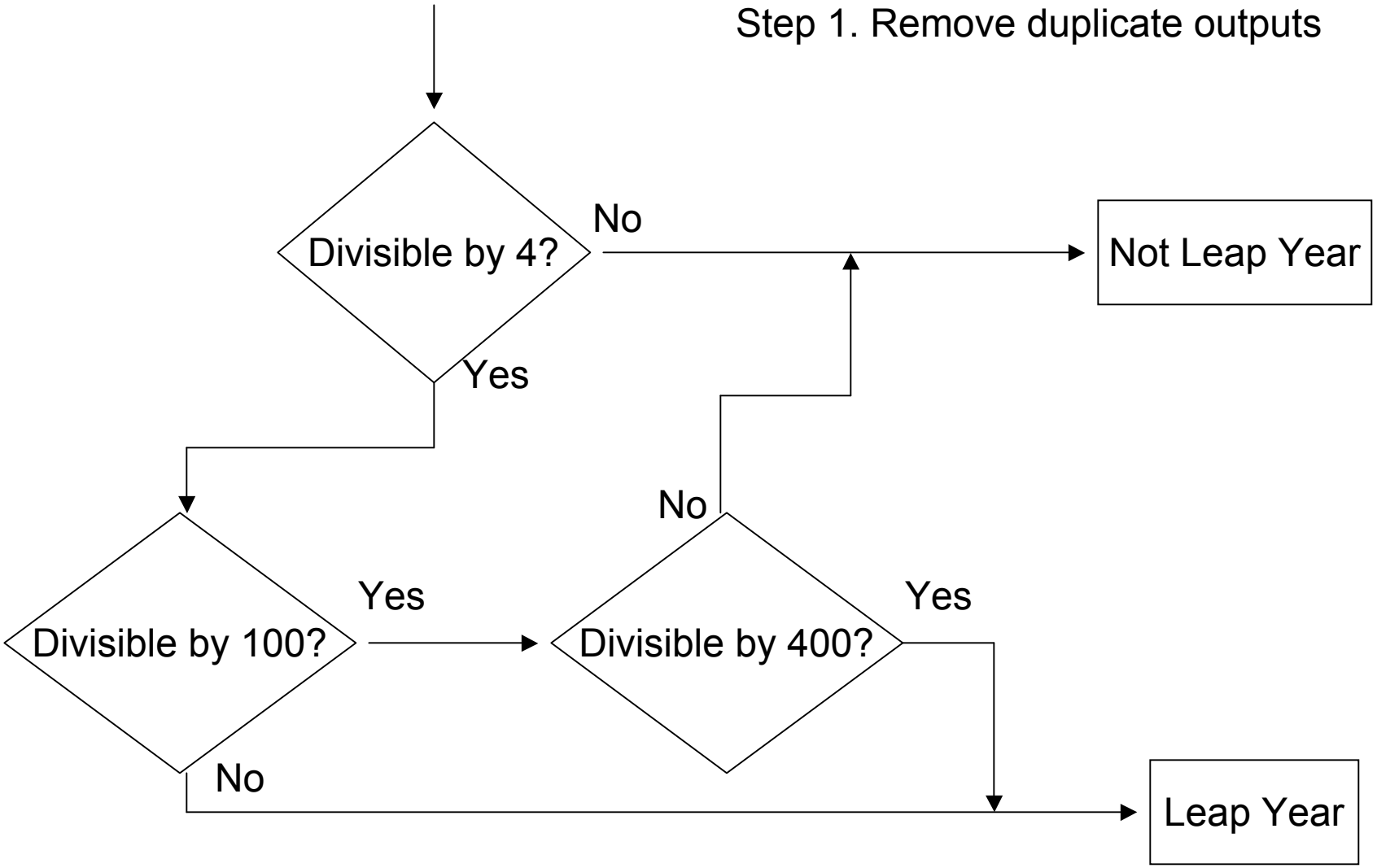
Complex Conditions

Step 1. Remove duplicate outputs

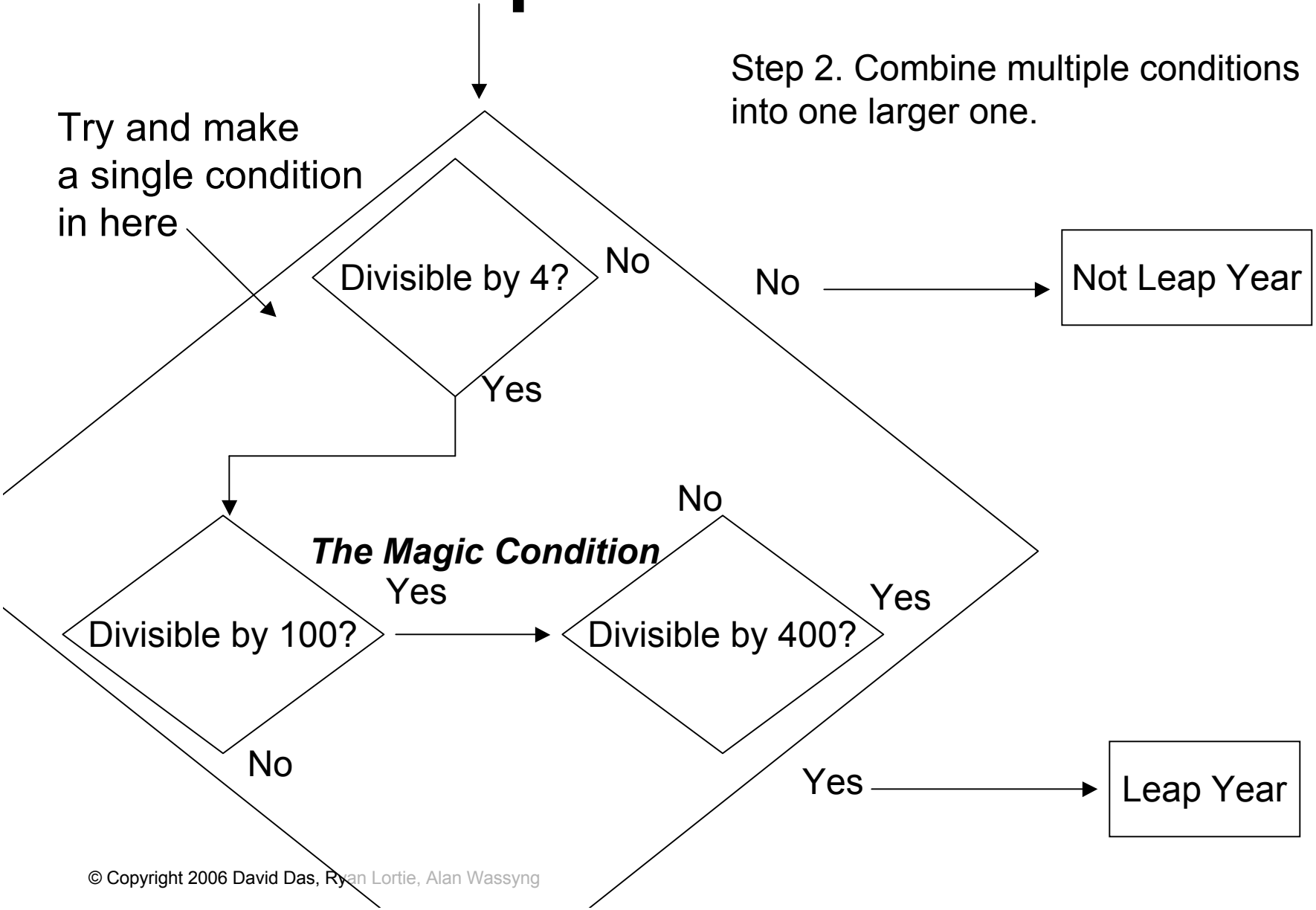


Complex Conditions

Step 1. Remove duplicate outputs



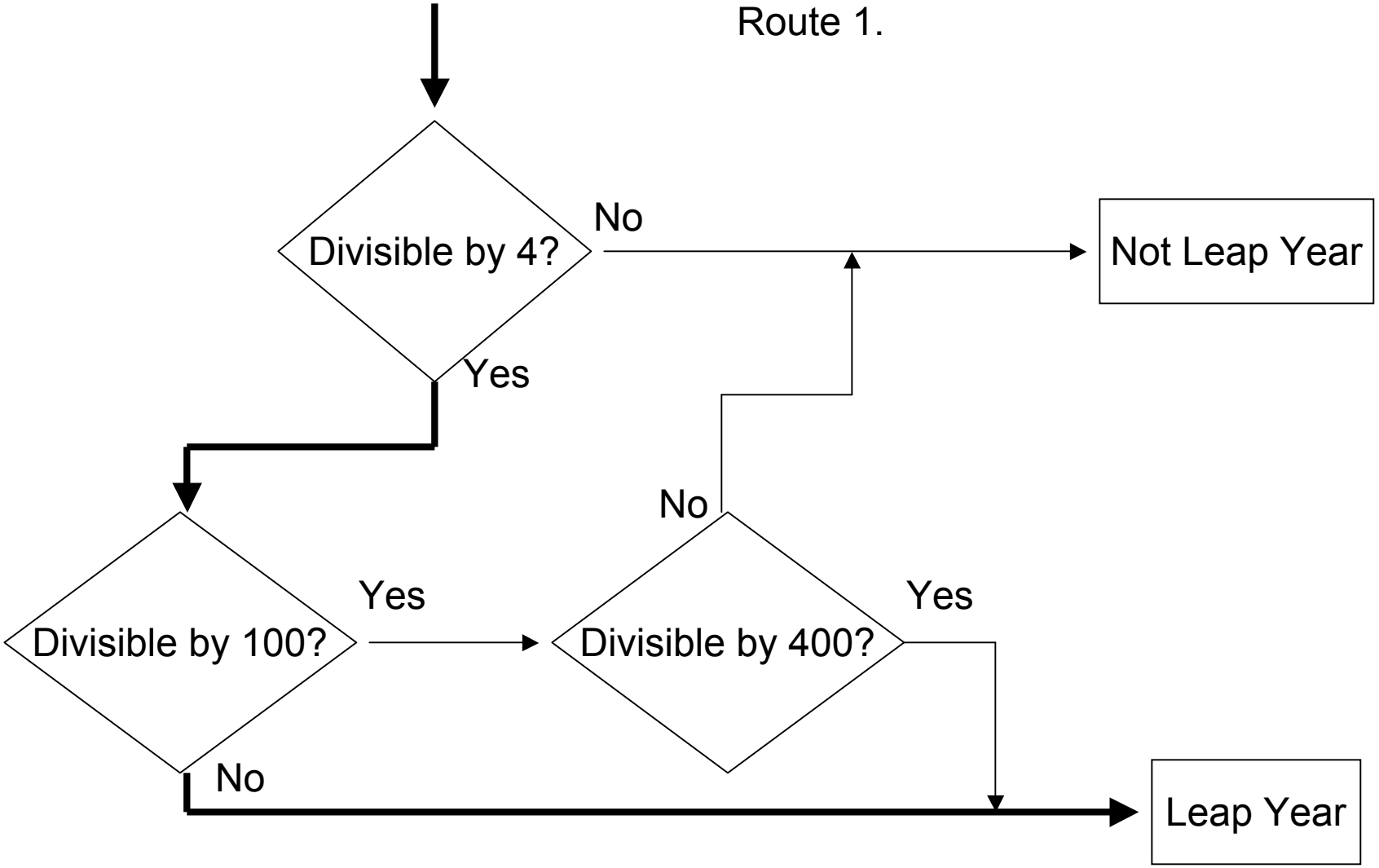
Complex Conditions



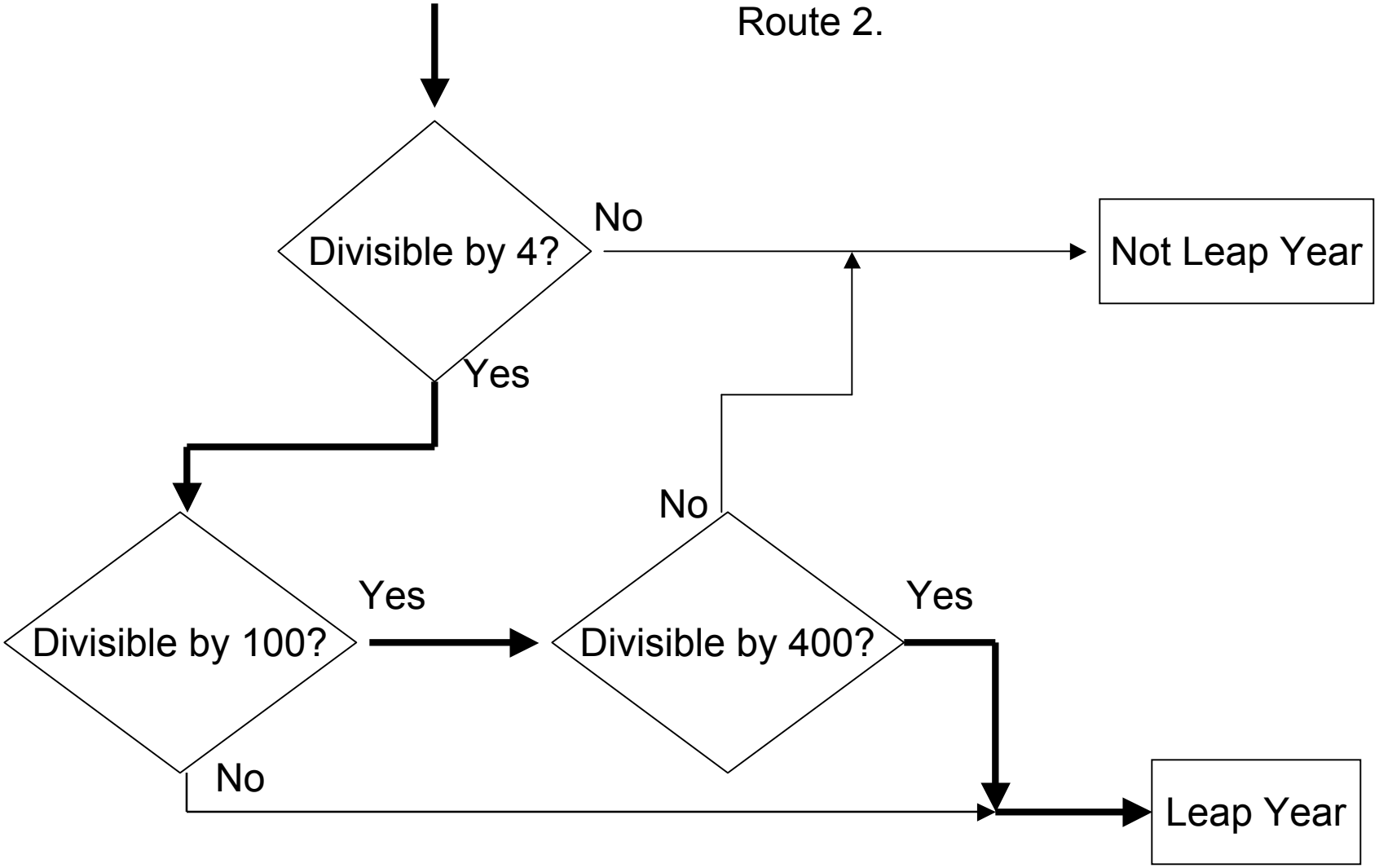
Complex Conditions

- How do we determine what the Magic Condition is?
- Look for the routes that make the condition true.

Complex Conditions



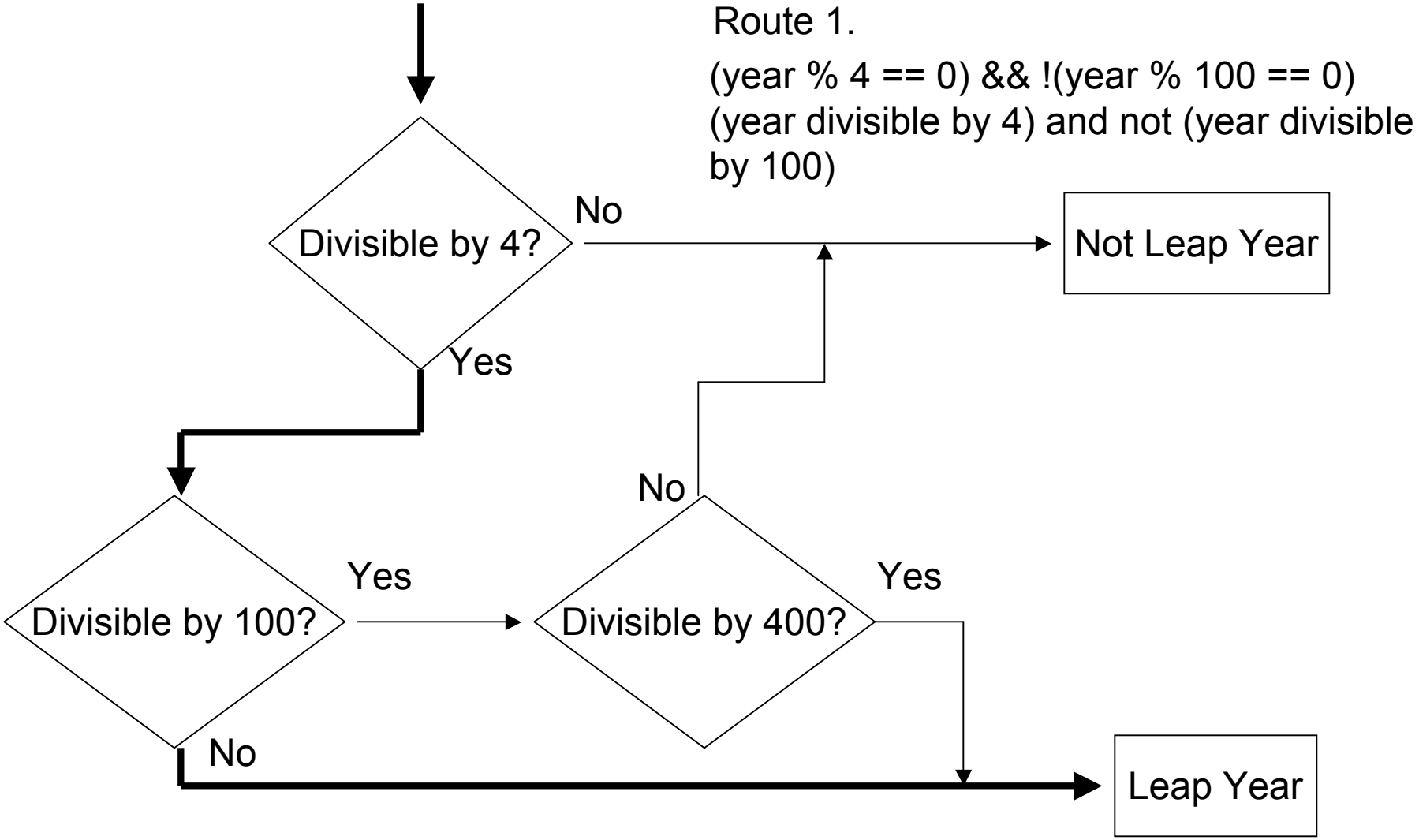
More Complex Conditions



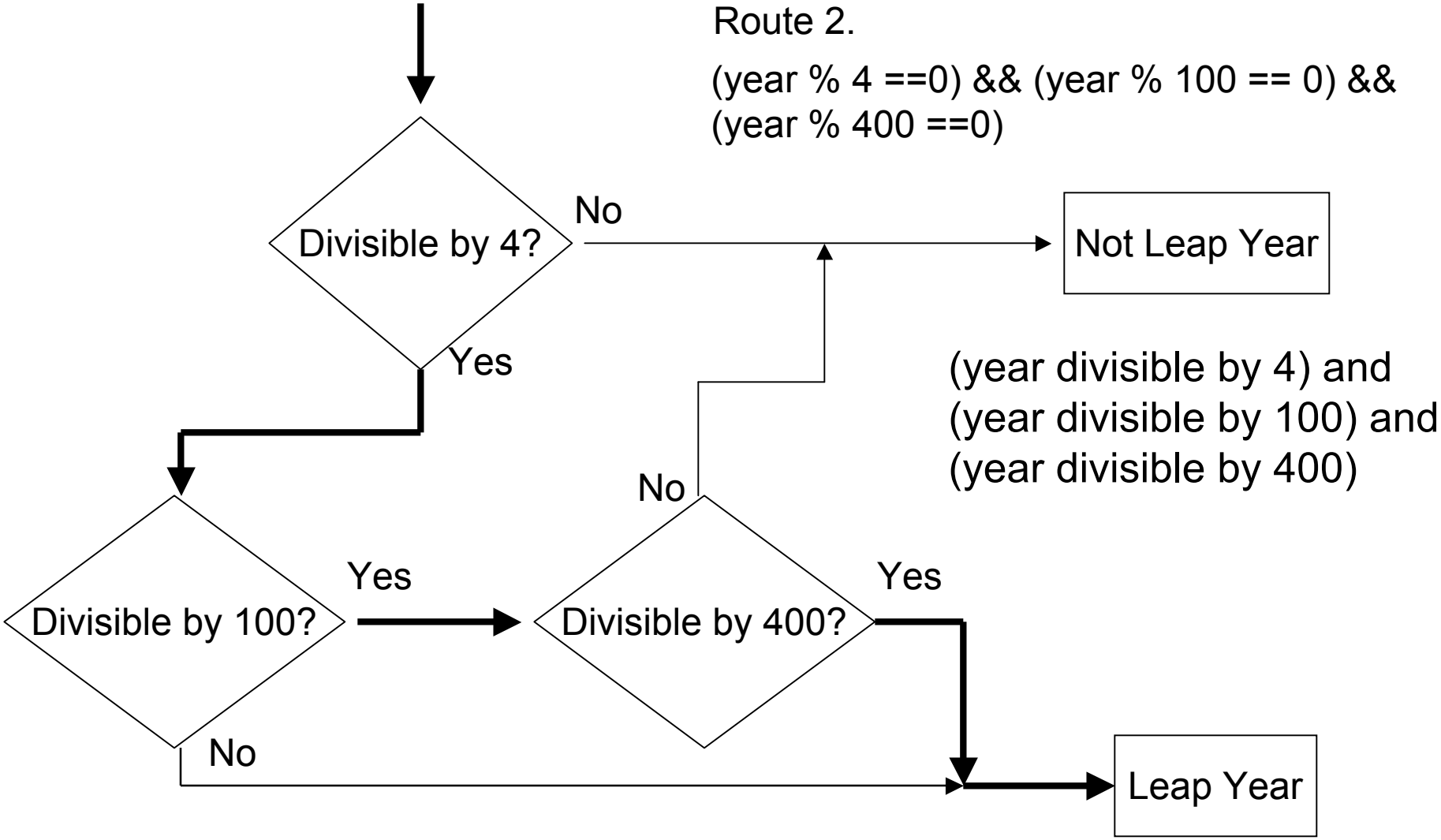
More Complex Conditions

- We need to map our way through each individual route and determine what it takes for it to be true.
- Each individual condition needs to be grouped with && (and) because we need all the conditions in the route to be satisfied.
- If we take the *no* branch of a condition, we need to put a ! in front of it.

Complex Conditions



Complex Conditions



Complex Conditions

- Since either route could result in a leap year we can say that to get a leap year we must have that route1 or route2 is true.

```
((year % 4 == 0) && (year % 100 != 0)) ||
((year % 4 == 0) && (year % 100 == 0) &&
 (year % 400 == 0))
```

- Or is represented by ||

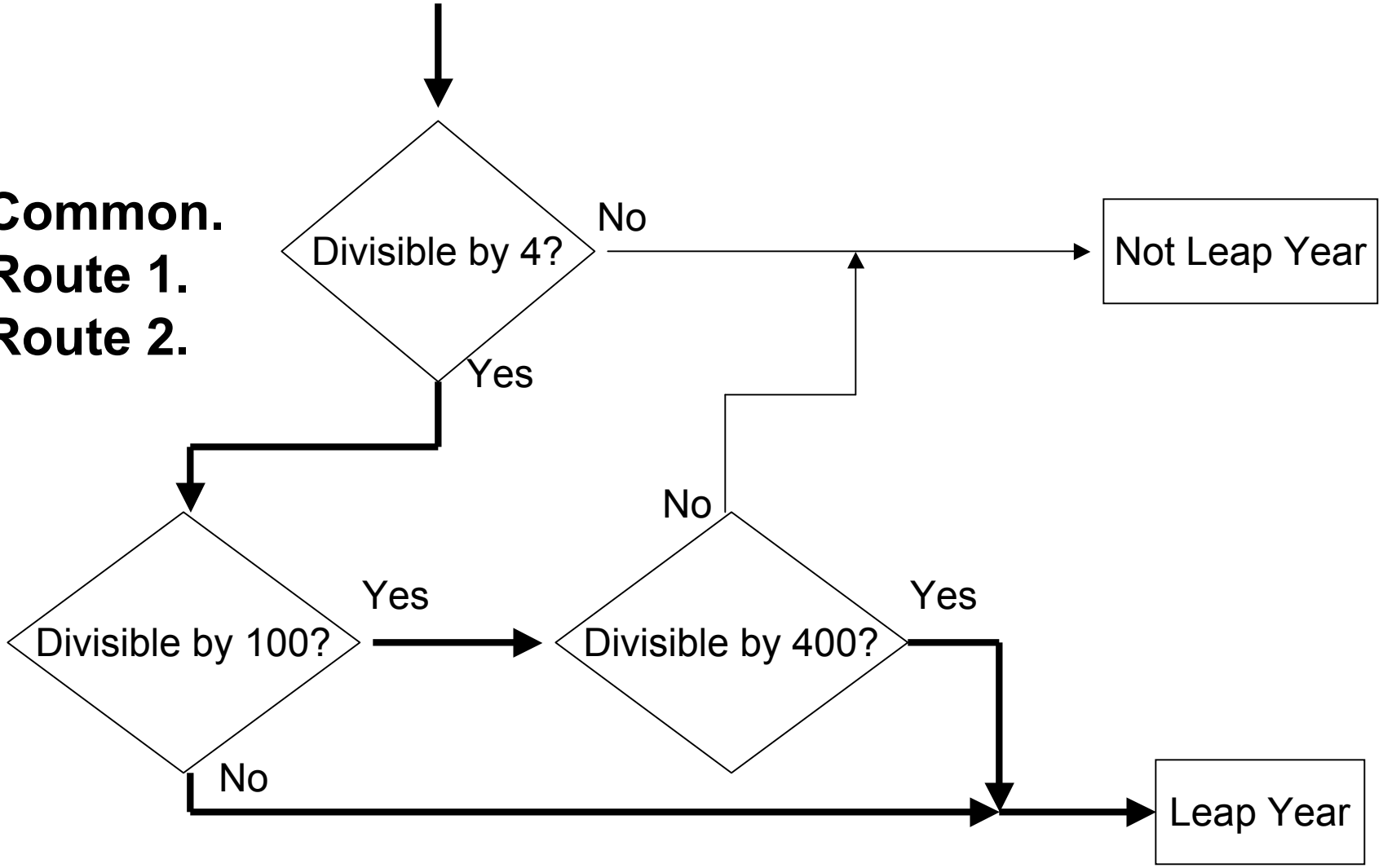
Complex Conditions

- We can look at the routes differently to make this simpler.

- The first decision box is part of two routes.

Complex Conditions

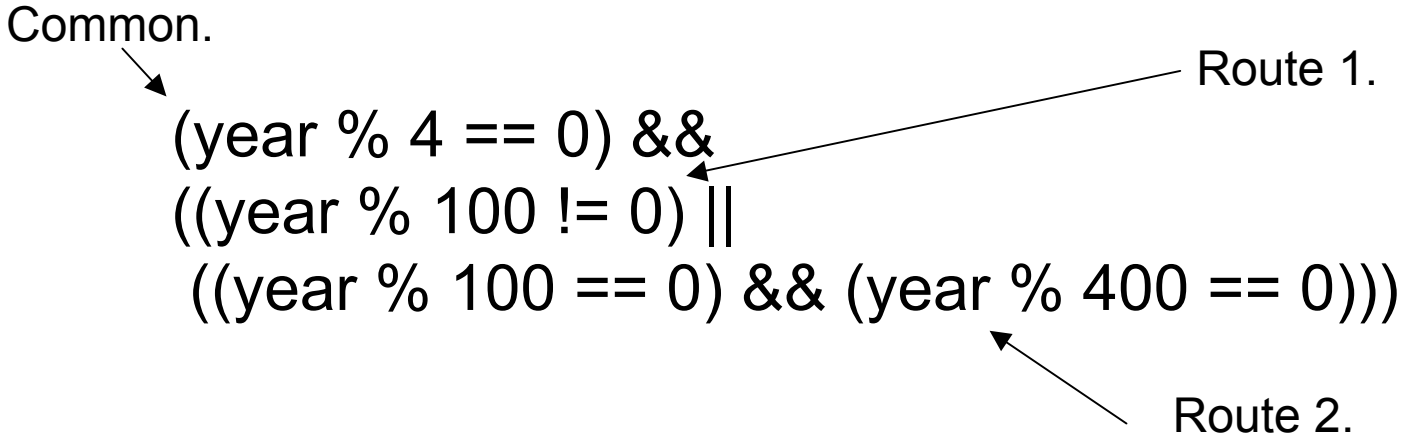
Common.
Route 1.
Route 2.



Complex Conditions

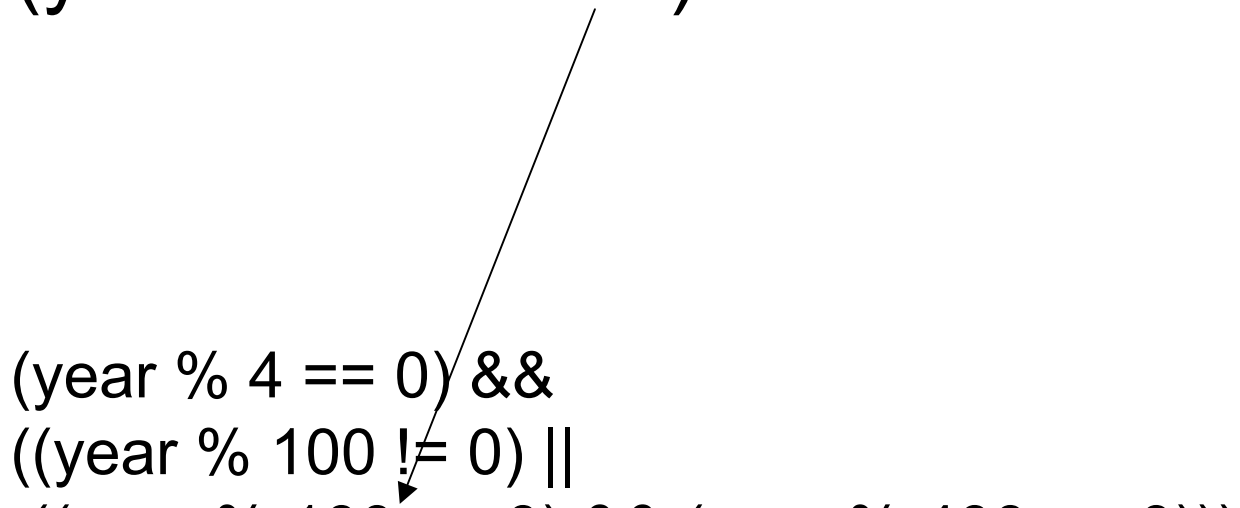
- We can look at the routes differently to make this a little simpler.

- Both successful routes have to go through the divisible by 4 box.



Complex Conditions

- There is one more simplification.
- (year % 100 == 0) is redundant. Why?



```
(year % 4 == 0) &&
((year % 100 != 0) ||
 ((year % 100 == 0) && (year % 400 == 0)))
```

Complex Conditions

- So it comes down to this!

```
(year % 4 == 0) &&
((year % 100 != 0) || (year % 400 == 0))
```

- and we can implement this very clearly in our code.

Complex Conditions

```
private void button1_Click(object sender, EventArgs e)
{
   int year = Convert.ToInt32(inputBox.Text);

   if ((year % 4 == 0)&&
      ((year % 100 != 0) || (year % 400 == 0)))
   {
      outputBox.Text = "Leap Year";
   } else
   {
      outputBox.Text = "Not Leap Year";
   }
}
```

much better ☺

Complex Conditions

```
private void button1_Click(object sender, EventArgs e)
{
   int year = Convert.ToInt32(inputBox.Text);

   if ((year % 4 == 0)&&
      ((year % 100 != 0) || (yea
   {
      outputBox.Text = "Leap Yea
   } else
   {
      outputBox.Text = "Not Leap
   }
}
```