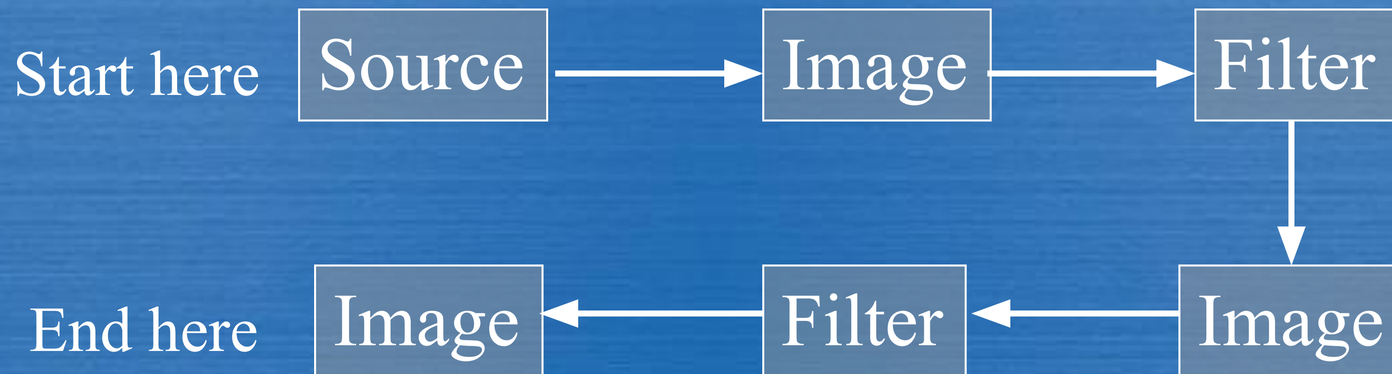
ITK Lecture 6 - The Pipeline

Damion Shelton

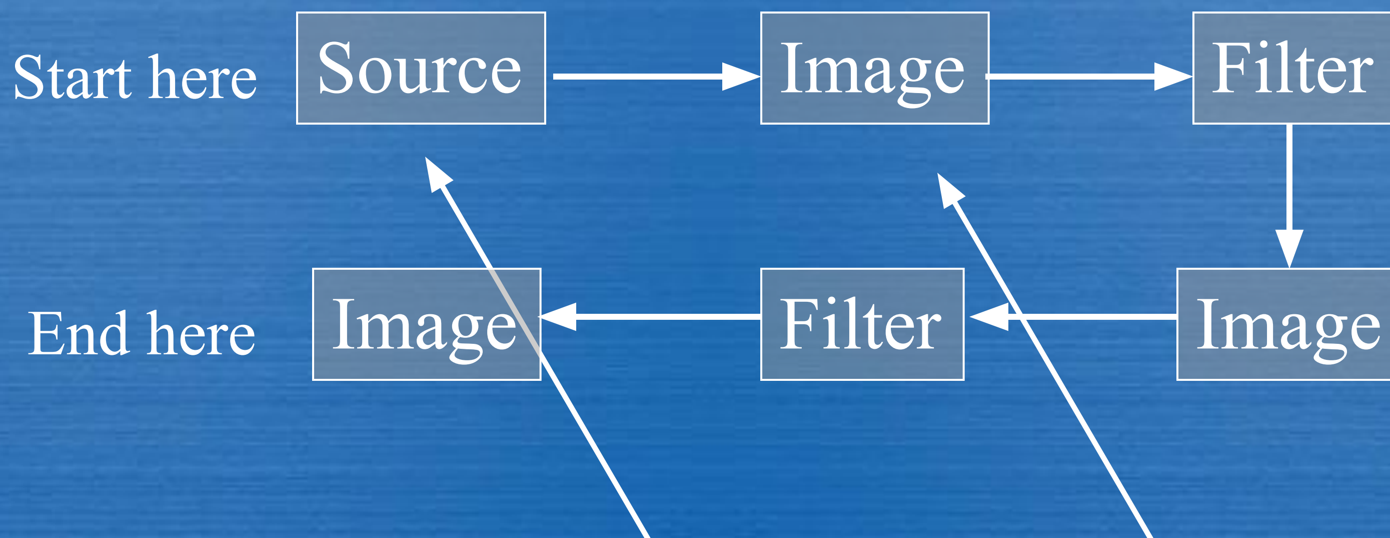
Methods in Image Analysis
CMU Robotics Institute 16-725
U. Pitt Bioengineering 2630
Spring Term, 2006

- You may recall that ITK is organized around *data objects* and *process objects*
- You should now be somewhat familiar with the primary data object, itk::Image
- Today we'll talk about how to do cool things to images, using process objects



The pipeline consists of data objects, and things that create data objects (i.e. process objects).

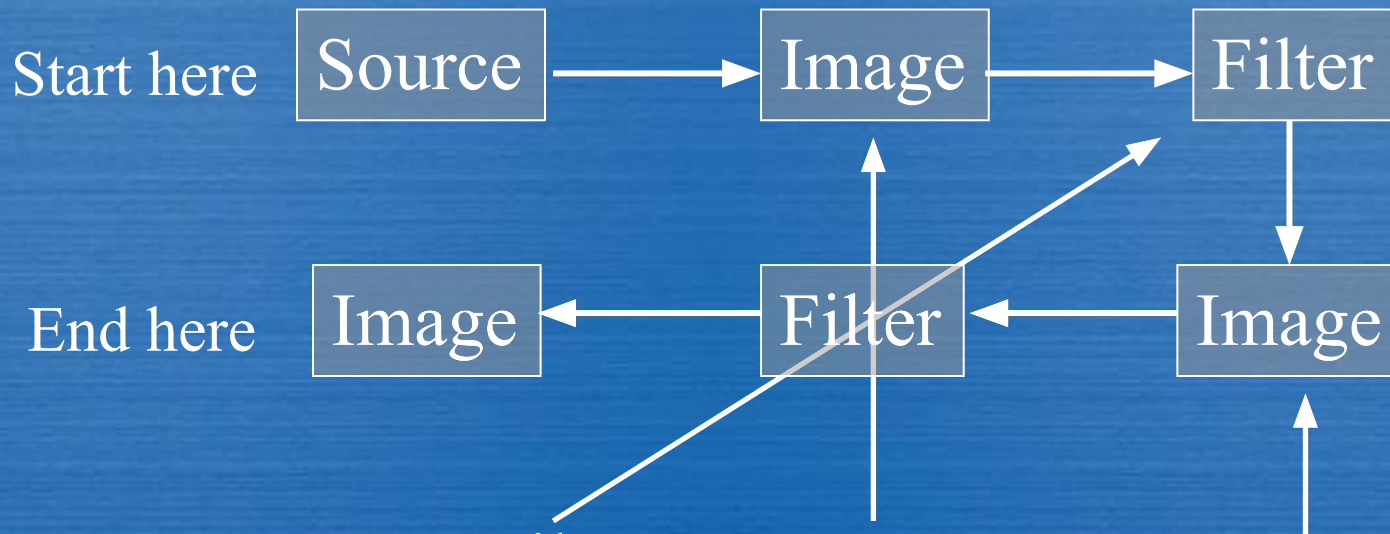
Image sources



itk::ImageSource<TOutputImage>

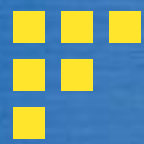
The base class for all process objects that produce images without an input image

Image to image filters



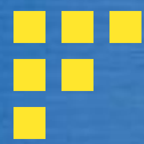
itk::ImageToImageFilter<TInputImage, TOutputImage>

The base class for all process objects that produce images when provided with an image as input.

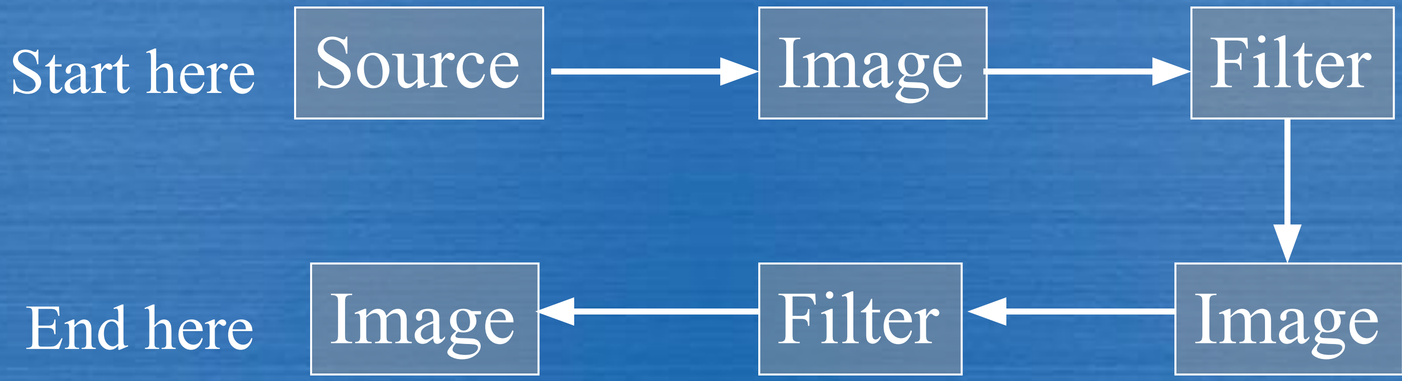


Input and output

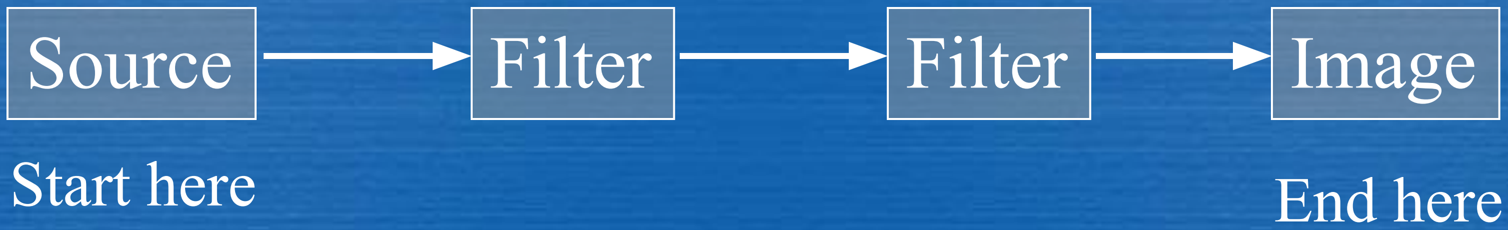
- ImageSource’s do not require input, so they have only a GetOutput() function
- ImageToImageFilter’s have both SetInput() and GetOutput() functions



Ignoring intermediate images

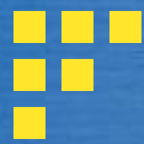


=



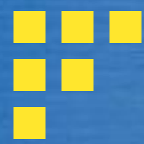
How this looks in code

```
SrcType::Pointer src = SrcType::New();
FilAType::Pointer filterA = FilAType::New();
FilBType::Pointer filterB = FilBType::New();

src->SetupTheSource();
filterA->SetInput( src->GetOutput() );
filterB->SetInput( filterA->GetOutput() );

ImageType::Pointer im = filterB->GetOutput();
```



When execution occurs

- The previous page of code *only* sets up the pipeline - i.e., what connects to what
- This *does not* cause the pipeline to execute
- In order to “run” the pipeline, you must call Update() on the last filter in the pipeline



Propagation of Update()

- When Update() is called on a filter, the update propagates back “up” the pipeline until it reaches a process object that does not need to be updated, or the start of the pipeline

- If the input to the process object has changed
- If the process object itself has been modified - e.g., I change the radius of a Gaussian blur filter

Detecting process object modification

The easy way is to use

```
itkSetMacro(MemberName, type);
```

which produces the function

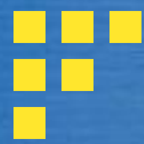
```
void SetMemberName(type);
```

that calls Modified() for you when a new value is set in the class.

For example:

```
itkSetMacro(DistanceMin, double);
```

sets member variable m_DistanceMin



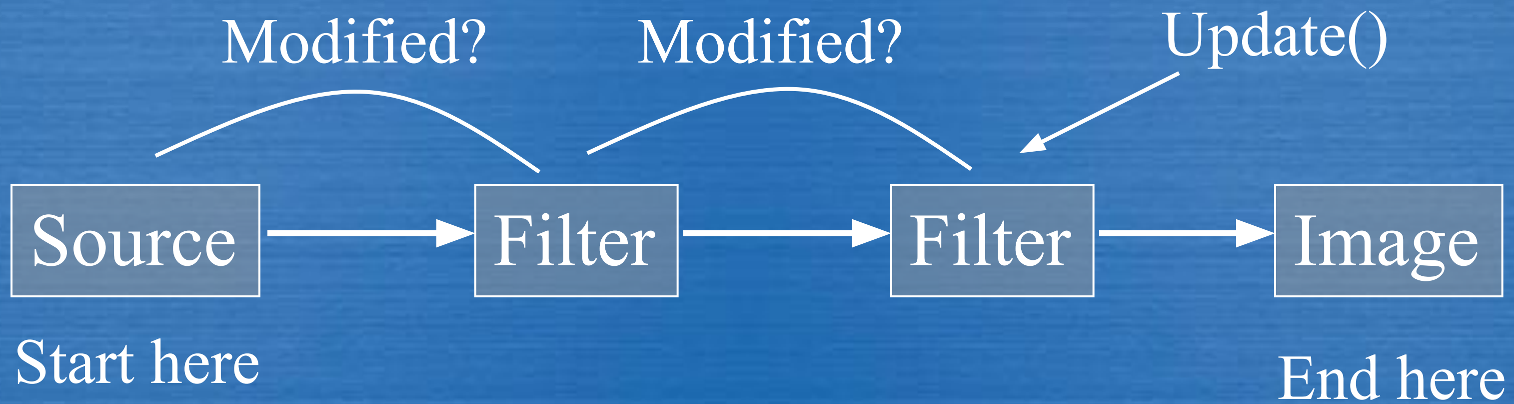
Process object modification, cont.

- The other way is to call Modified() from within a process object function when you know something has changed

```
this->Modified();
```

- You can call Modified() from outside the class as well, to force an update
- Using the macros is a better idea though...

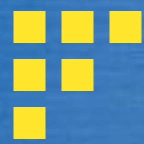
Running the pipeline - Step 1



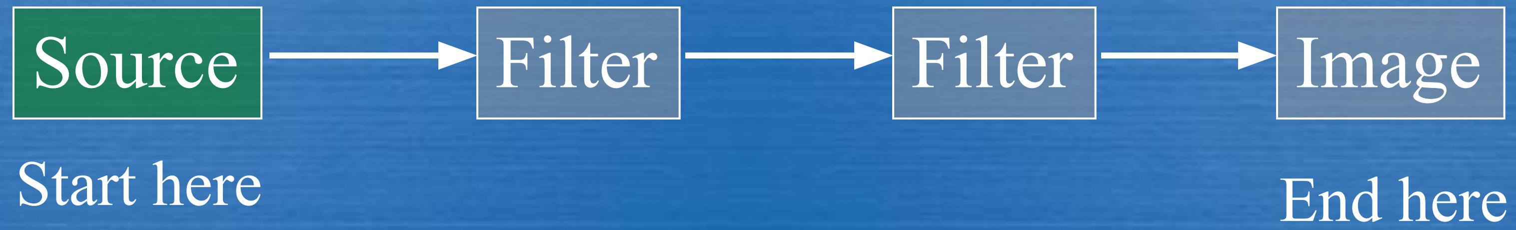
Not sure

Modified

Updated



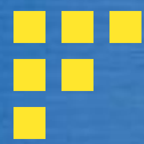
Running the pipeline - Step 2



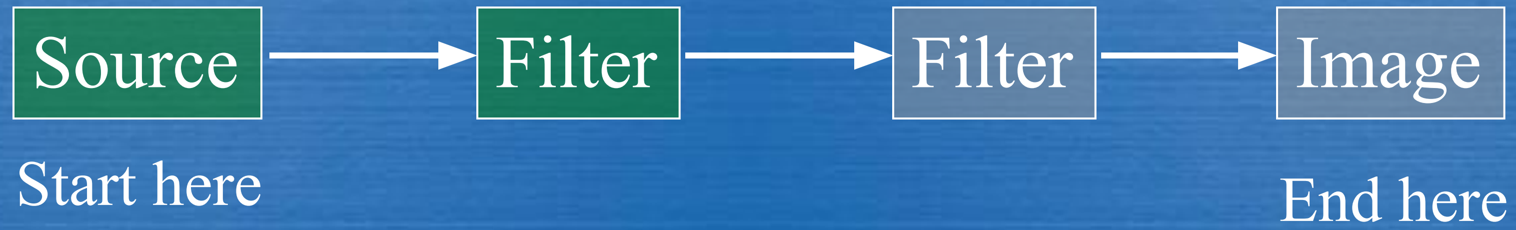
Not sure

Modified

Updated



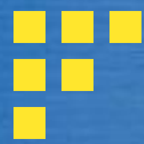
Running the pipeline - Step 3



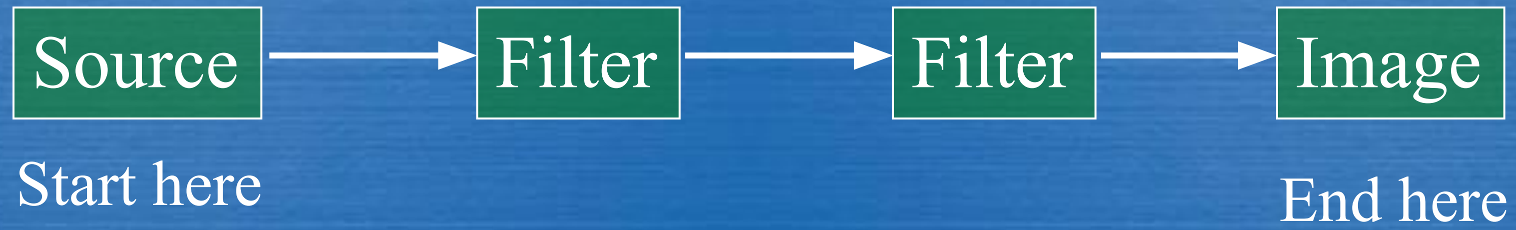
Not sure

Modified

Updated



Running the pipeline - Step 4

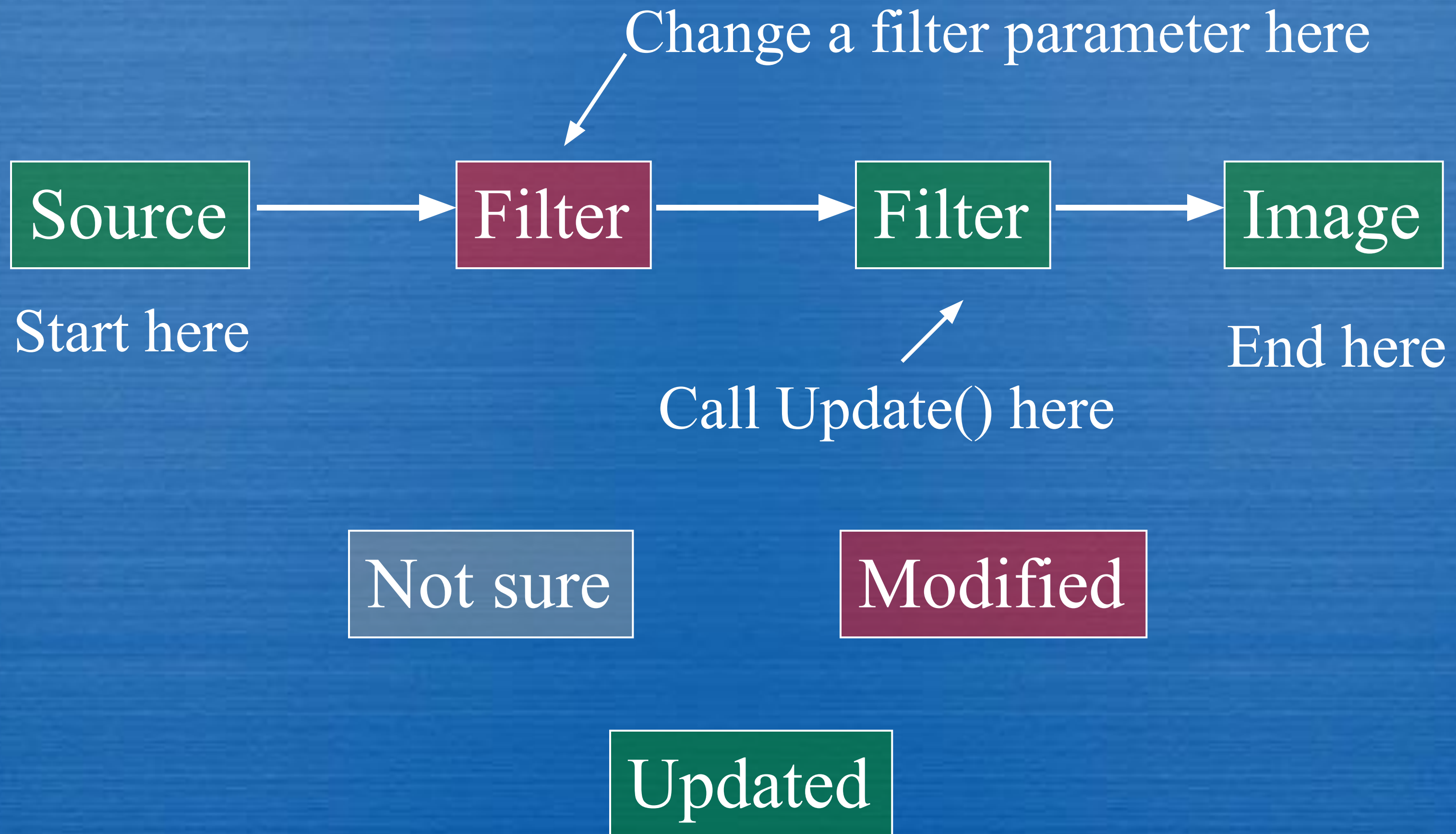


Not sure

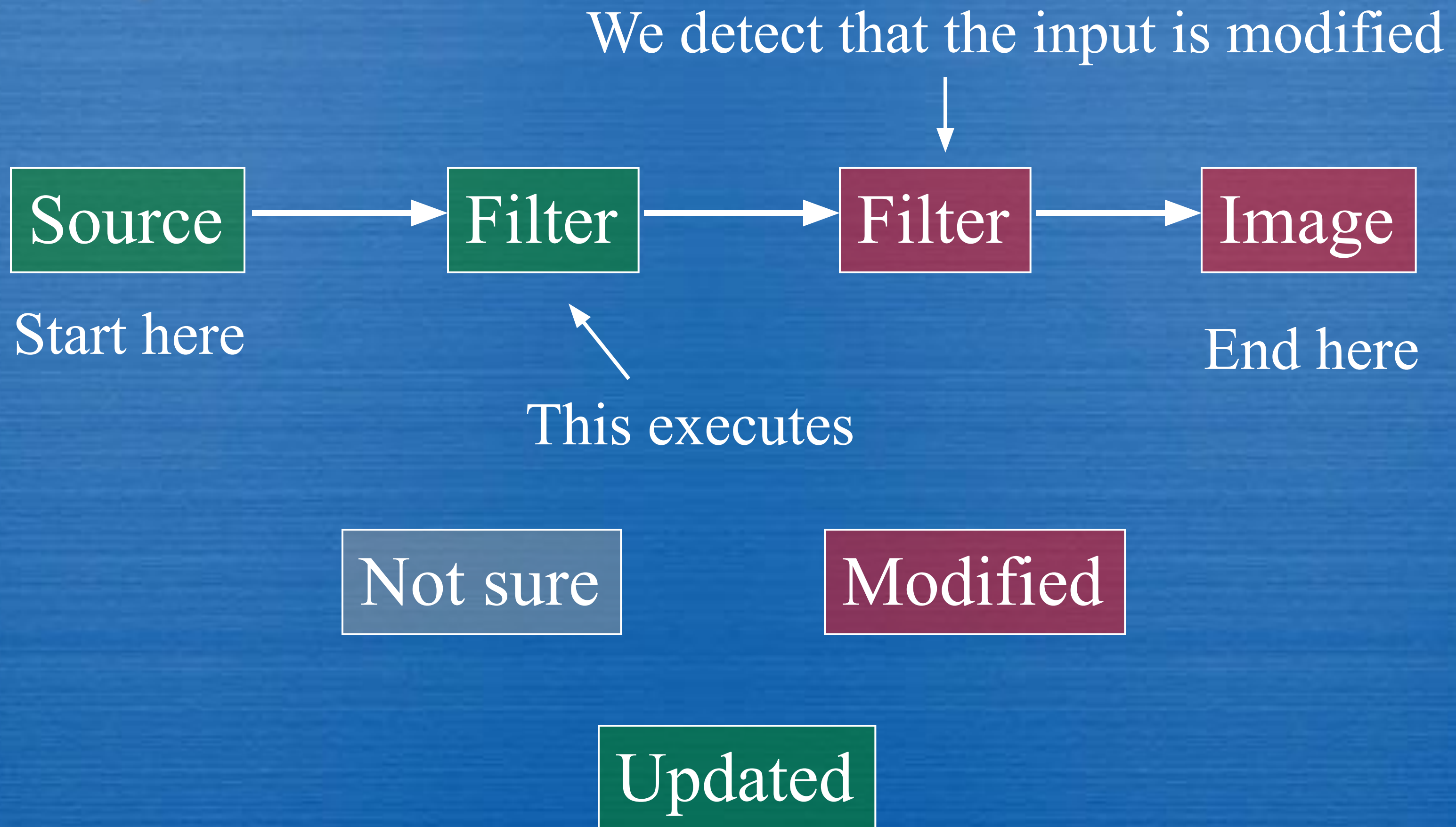
Modified

Updated

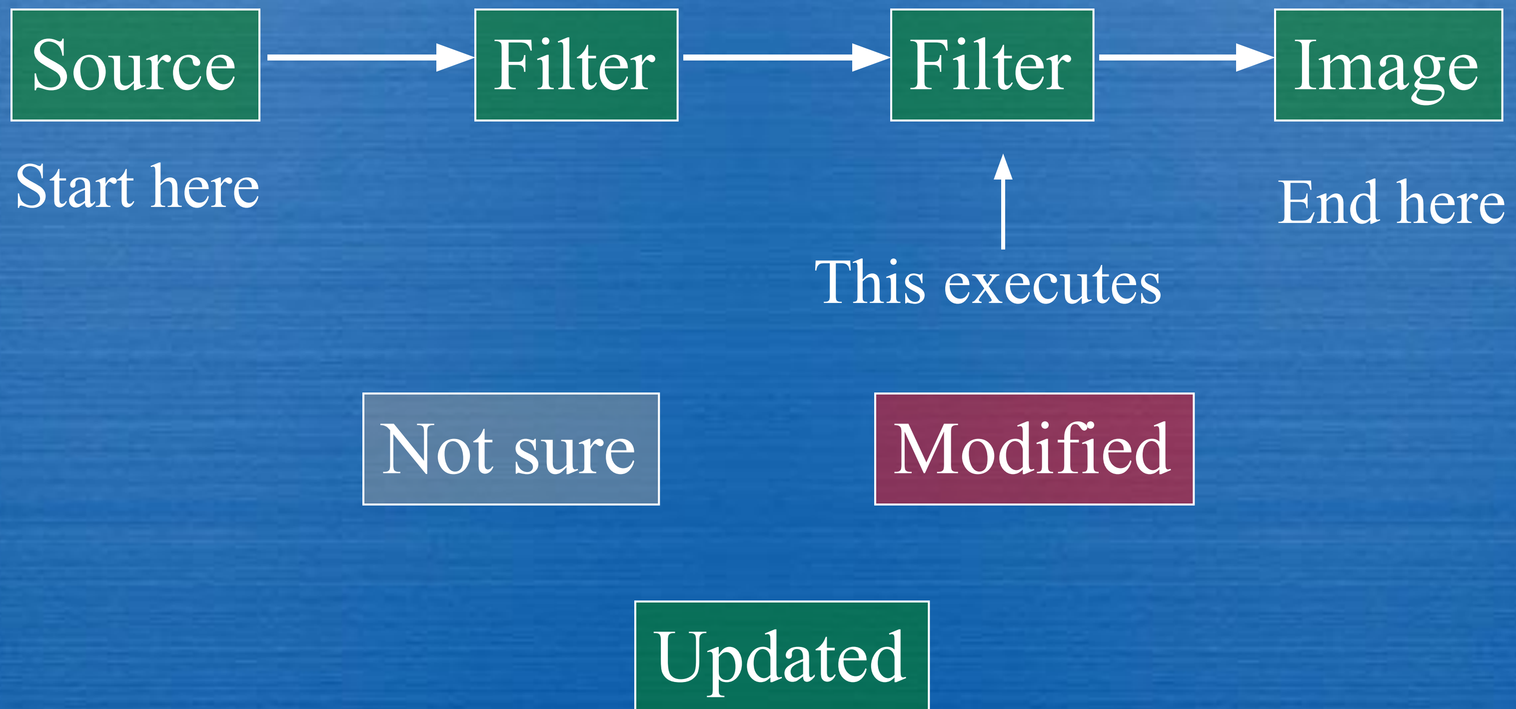
Modifying the pipeline - Step 1

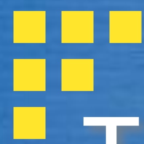


Modifying the pipeline - Step 2



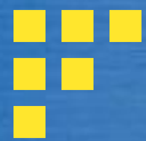
Modifying the pipeline - Step 3





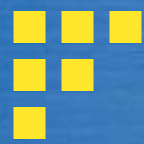
Thoughts on pipeline modification

- Note that in the previous example the source never re-executed; it had no input and it was never modified, so the output cannot have changed
- This is good! We can change things at the end of the pipeline without wasting time recomputing things at the beginning



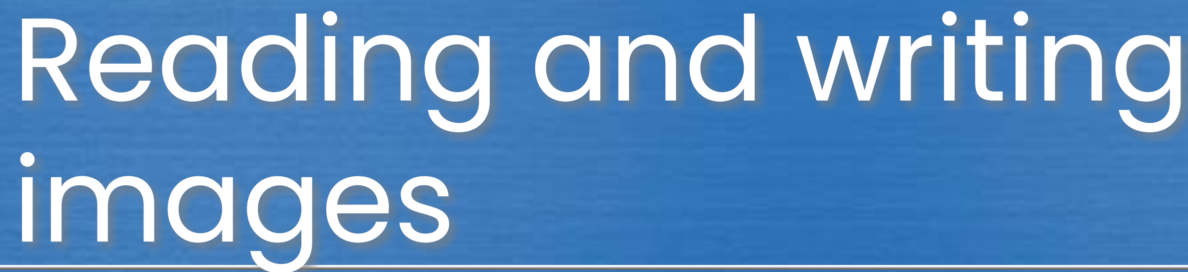
It’s easy in practice

1. Build a pipeline
2. Call Update() on the last filter - get the output
3. Tweak some of the filters
4. Call Update() on the last filter - get the output
5. ...ad nauseam

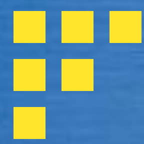


Reading & writing

- You will often begin and end pipelines with readers and writers
- Fortunately, ITK knows how to read a wide variety of image types!

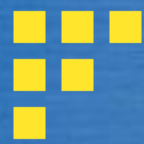


- Use itk::ImageFileReader<ImageType> to read images
- Use itk::ImageFileWriter<ImageType> to write images
- Both classes have a SetImageIO(ImageIOBase*) function used to specify a particular type of image to read or write



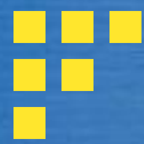
Reading an image (4.1.2)

- Create a reader
- Create an instance of an ImageIOBase derived class (e.g. PNGImageIO)
- Pass the IO object to the reader
- Set the file name of the reader
- Update the reader



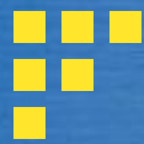
Reader notes

- The ImageType template parameter is the type of image you want to convert the stored image *to*, not necessarily the type of image stored in the file
- ITK assumes a valid conversion exists between the stored pixel type and the target pixel type



Writing an image

- Almost identical to the reader case, but you use an ImageFileWriter instead of a reader
- If you've already created an IO object during the read stage, you can recycle it for use with the writer



More read/write notes

- ITK actually has several different ways of reading files - what I've presented is the simplest conceptually
- Other methods exist to let you read files without knowing their format