Extending Swing to Run Multi-Touch Applications

Michael Riecken
Java Capability Leader, Trissential

www.open-table.org

Java

Sun microsystems

The goal of this presentation is to give you the information that you need to go out and build your own multi-touch device, using Swing as the user interface.

Agenda

- Multi-Touch Defined
- Delivering User Input
- “Regular” Swing Interfaces
- Multi-Touch Swing Interfaces
- A Quick Sample
- Questions

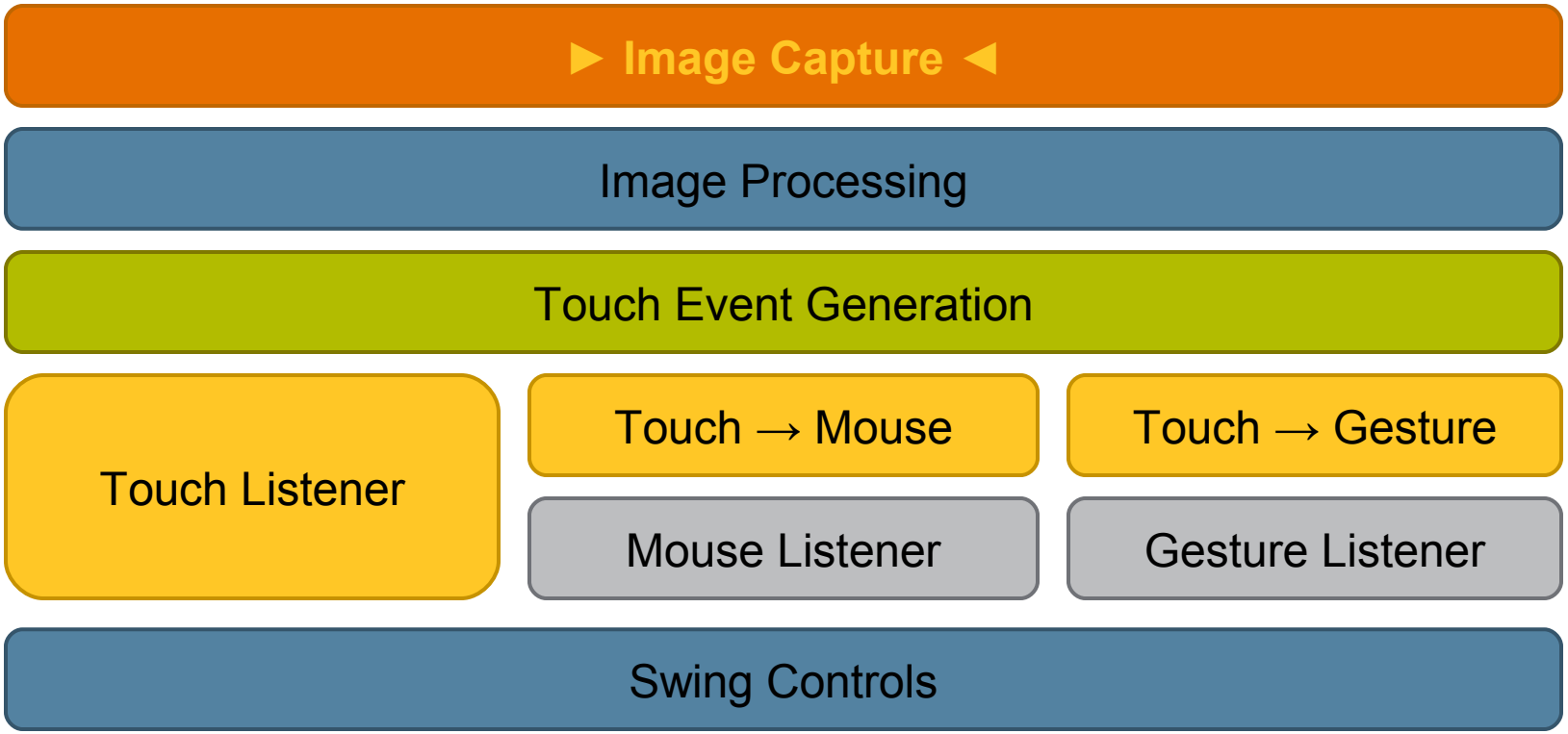
Multi-Touch Defined

- Multiple Simultaneous Mouse Inputs
- Multiple Simultaneous Keyboard Inputs
- Multiple Users

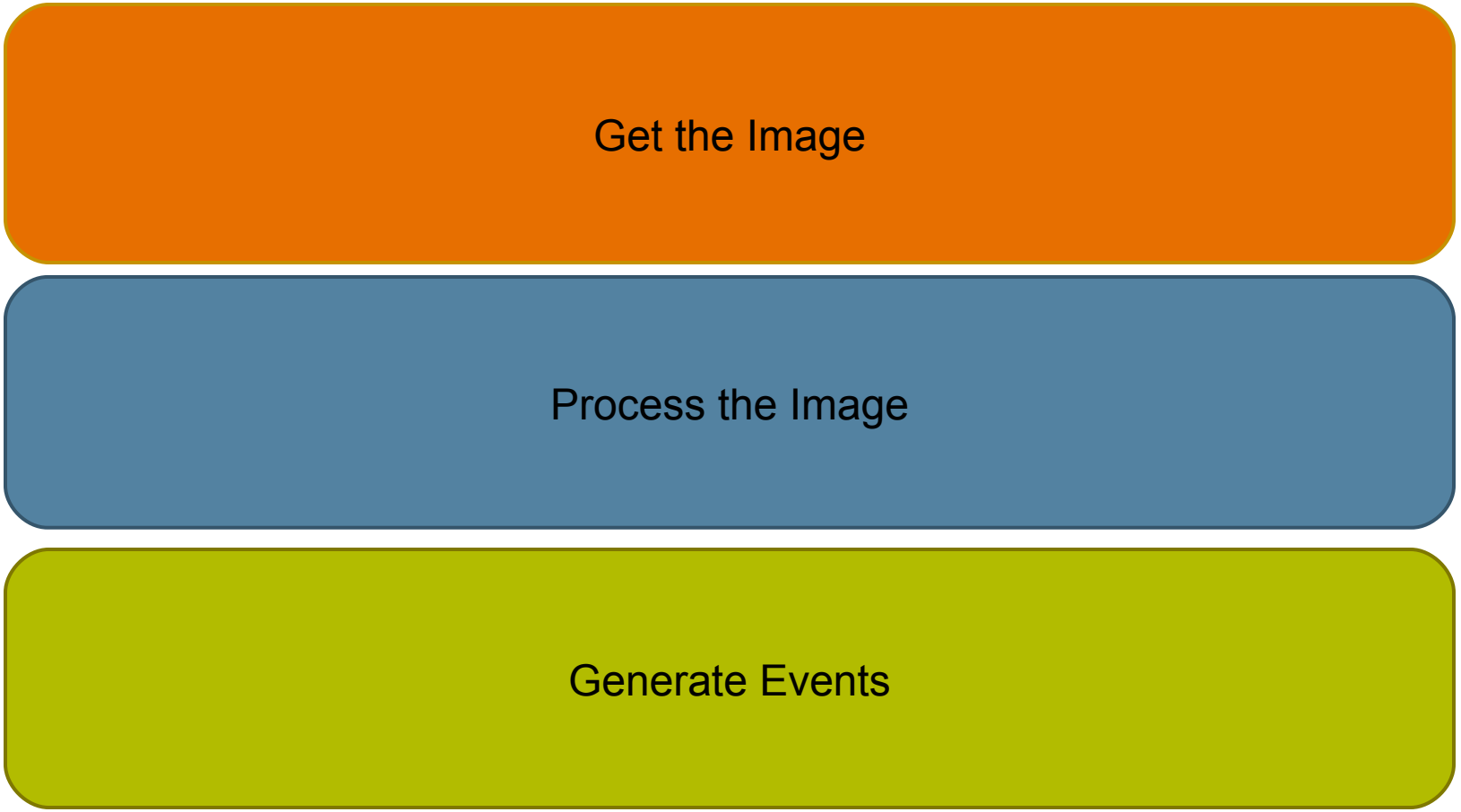
Agenda

- Multi-Touch Defined
- Delivering User Input
- “Regular” Swing Interfaces
- Multi-Touch Swing Interfaces
- A Quick Sample
- Questions

Delivering User Input



Three Main Pieces



Default Event Provider

```
public void run() {

  VideoProvider videoProvider =
                          TableManager.getVideoProvider();
  ImageParser imageParse =
                            TableManager.getImageParser();

  while( running ) {

    VideoFrame frame = videoProvider.getCurrentFrame();

    if( frame != null ) {
      List<Pattern> touches = imageParser.parse(frame);
      generateEvents( touches );
    }
  }
}
```

Inside the Table

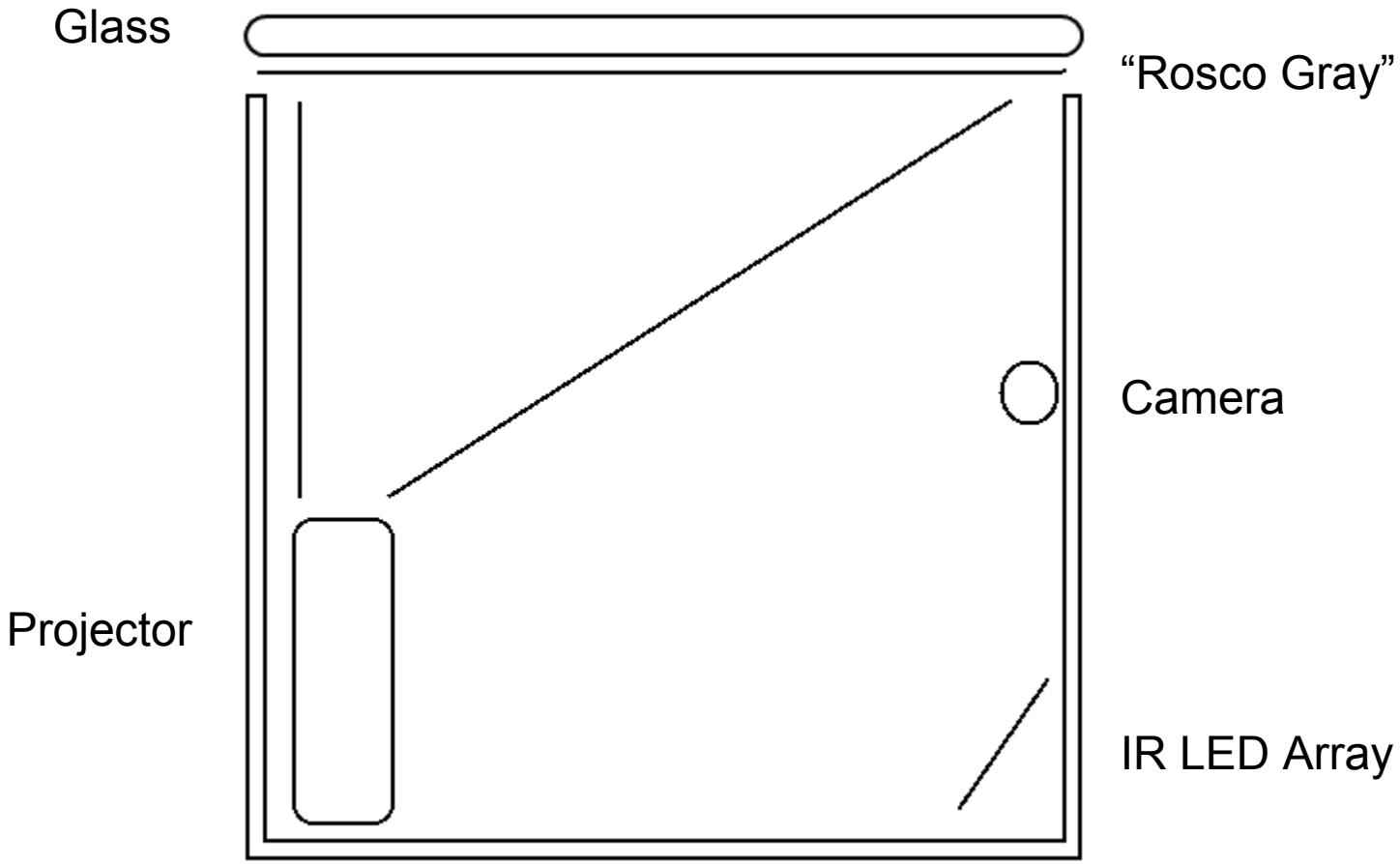


Image Acquisition

- Easily Modified Web Cam
  - Remove the IR filter
  - Add a visible light filter

What an IR Camera Sees
IR Video

Infrared Light Source

- Simple to Build, but Tricky to Position Properly
  - Reflected IR Detection
  - We use 70 880nm IR LEDs (475-1112-ND) in ten arrays of seven lights each, but other configurations are possible.
  - Power is taken directly from the computer's power supply via a 4 pin Molex connector (12v leads – yellow and black), so it turns on and off with the computer. (This also is how we drive our four cooling fans.)
- Other Configurations are Possible
  - Frustrated Total Internal Refraction
  - Direct IR
  - Visible Light
  - Combinations

Cabinets

Image Acquisition

- Software Acquisition
  - Java Media Framework
    - Windows, Solaris, Linux only
- Filtering
  - Multiple Processing Steps Available
  - Slows Down Processing
- Transformation
  - Correcting distortion due to camera placement. Technically a filter, but works in a different category.

Getting a Frame Grabber – VideoProvider

```
String devName=
                "vfw:Microsoft WDM Image Capture (Win32):0";

CaptureDeviceInfo cdi =
                    CaptureDeviceManager.getDevice(devName);

MediaLocator ml = cdi.getLocator();

Player player = null;
try {
   player = Manager.createRealizedPlayer(ml);
}
catch (Exception e) { return; }

player.start();
```

Getting a Frame Grabber – VideoProvider

```
// Wait a few seconds for camera to initialize
try {
  Thread.sleep(getDelay());
}
catch (InterruptedException e) { return; }

String controlName
                    = "javax.media.control.FrameGrabbingControl"

frameGrabber = (FrameGrabbingControl)
                                player.getControl( controlName);
```

Grabbing a Frame – Video Provider

```
public VideoFrame getCurrentFrame() {
  Buffer buf = frameGrabber.grabFrame();
  BufferToImage b2i = new BufferToImage(
                                        (VideoFormat)buf.getFormat())
  Image img = (b2i.createImage(buf));

  if( img == null ) { return null; }

  BufferedImage buffImg =
    new BufferedImage(img.getWidth(null),
                      img.getHeight(null),
                      BufferedImage.TYPE_INT_RGB);
  Graphics2D g = buffImg.createGraphics();
  g.drawImage(img, null, null);

  JMFVideoFrame frame = new JMFVideoFrame(this, buffImg);
  return frame;
}
```

Grabbing a Frame – Video Provider

```
public VideoFrame getCurrentFrame() {
  Buffer buf = frameGrabber.grabFrame();
  BufferToImage b2i = new BufferToImage(
                                    (VideoFormat)buf.getFormat())
  Image img = (b2i.createImage(buf));

  if( img == null ) { return null; }

  BufferedImage buffImg =
    new BufferedImage(img.getWidth(null),
                      img.getHeight(null),
                      BufferedImage.TYPE_INT_RGB);
  Graphics2D g = buffImg.createGraphics();
  g.drawImage(img, null, null);

  JMFVideoFrame frame = new JMFVideoFrame(this, buffImg);
  return frame;
}
```

Grabbing a Frame – Video Provider

```
public VideoFrame getCurrentFrame() {
  Buffer buf = frameGrabber.grabFrame();
  BufferToImage b2i = new BufferToImage(
                                        (VideoFormat)buf.getFormat())
  Image img = (b2i.createImage(buf));

  if( img == null ) { return null; }

  BufferedImage buffImg =
    new BufferedImage(img.getWidth(null),
                      img.getHeight(null),
                      BufferedImage.TYPE_INT_RGB);
  Graphics2D g = buffImg.createGraphics();
  g.drawImage(img, null, null);

  JMFVideoFrame frame = new JMFVideoFrame(this, buffImg);
  return frame;
}
```

Three Main Pieces

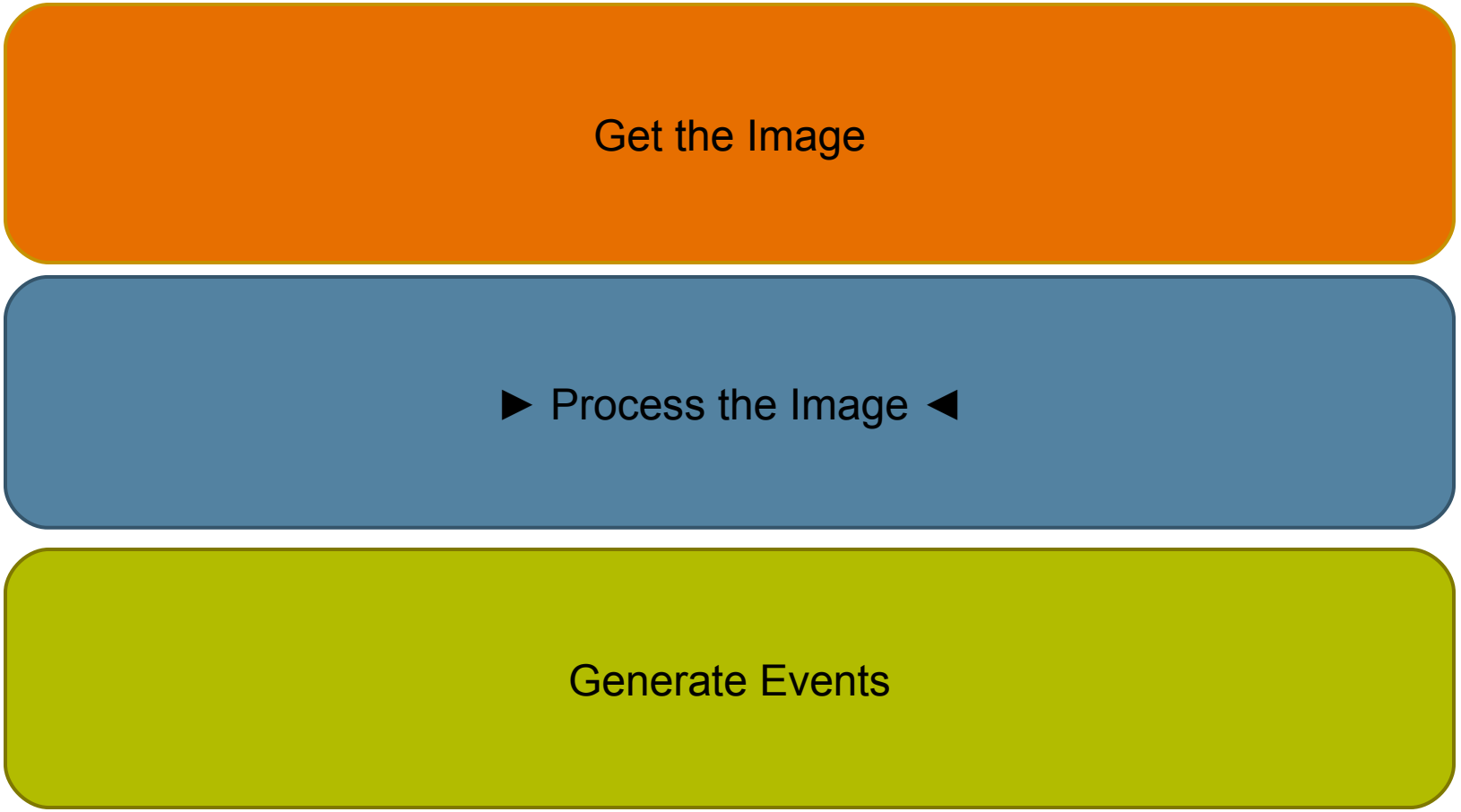


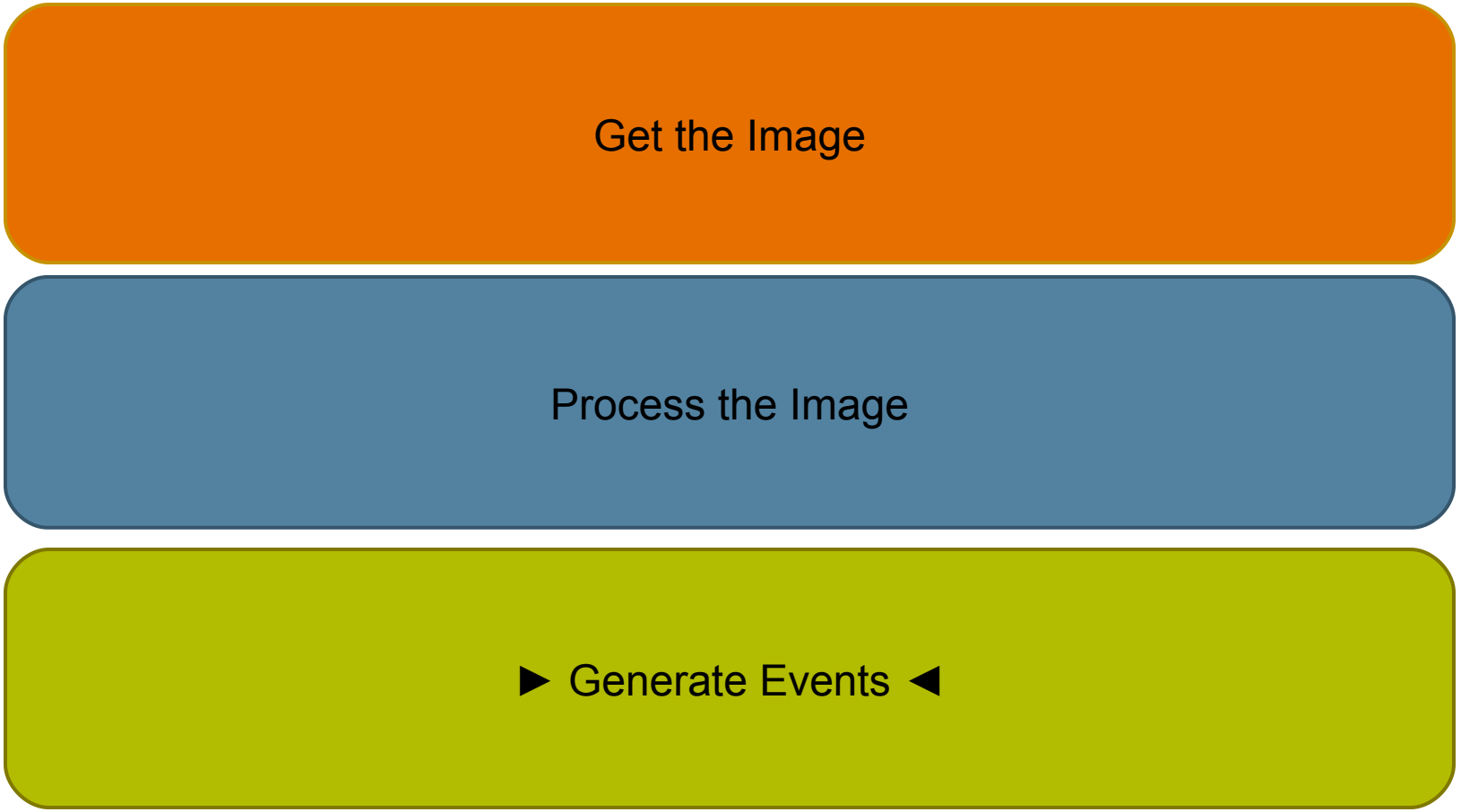
Image Processing

- Uses the Image from the Video Provider
- Identifies Blobs or other Objects (Patterns)
  - Potentially Complex Objects like “Dominos”
- Gives us a list of Patterns
  - Image
  - Location

Finding Patterns

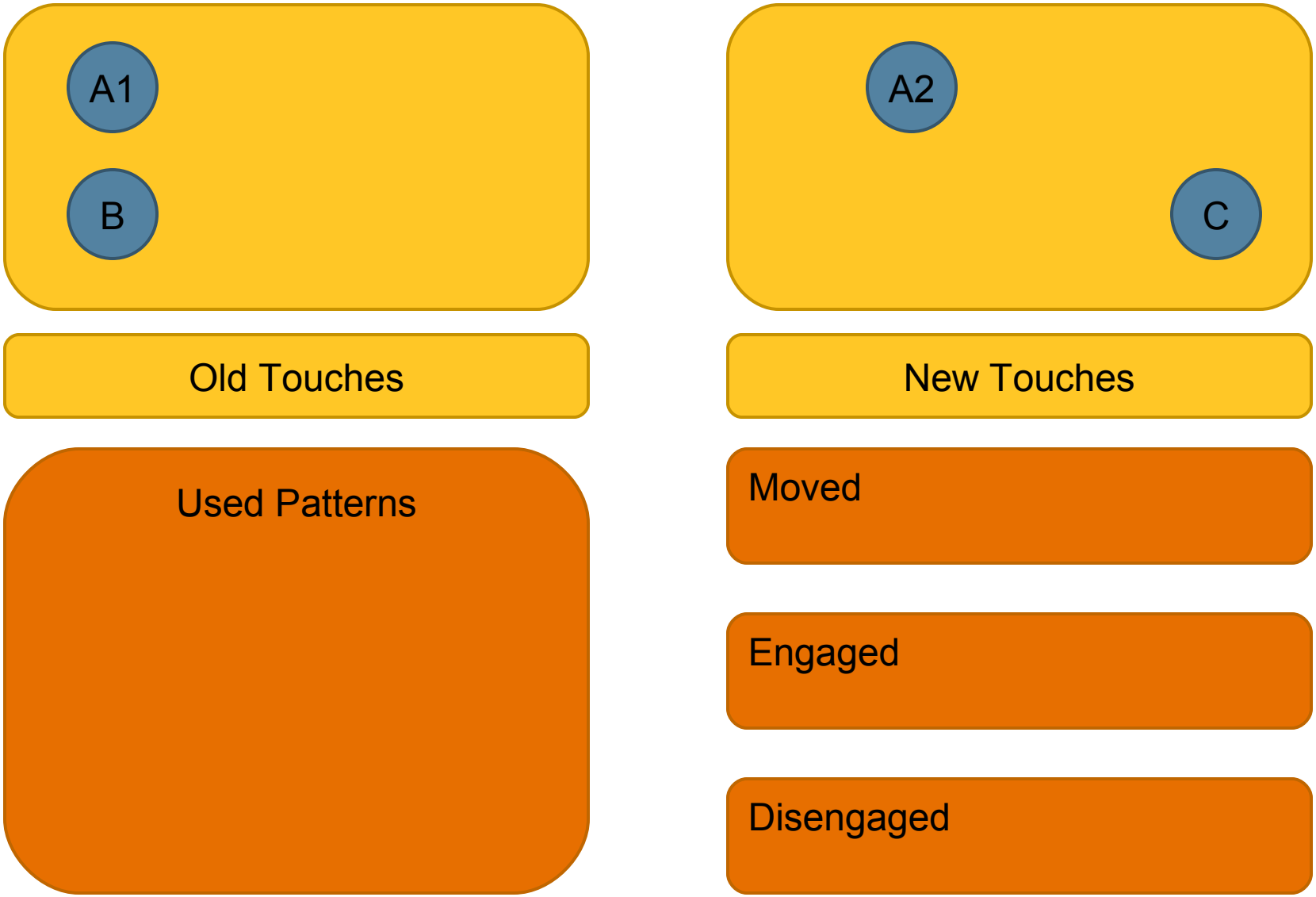
- Pluggable Image Parsers
- Default Method:
  - Scan for touches left-to-right, top-to-bottom, skipping pixels
  - Match to basic shapes
  - Snip Shape and return
- Can Be Made Very Complicated
  - Identifying Complex Objects
  - Potential Performance Hit

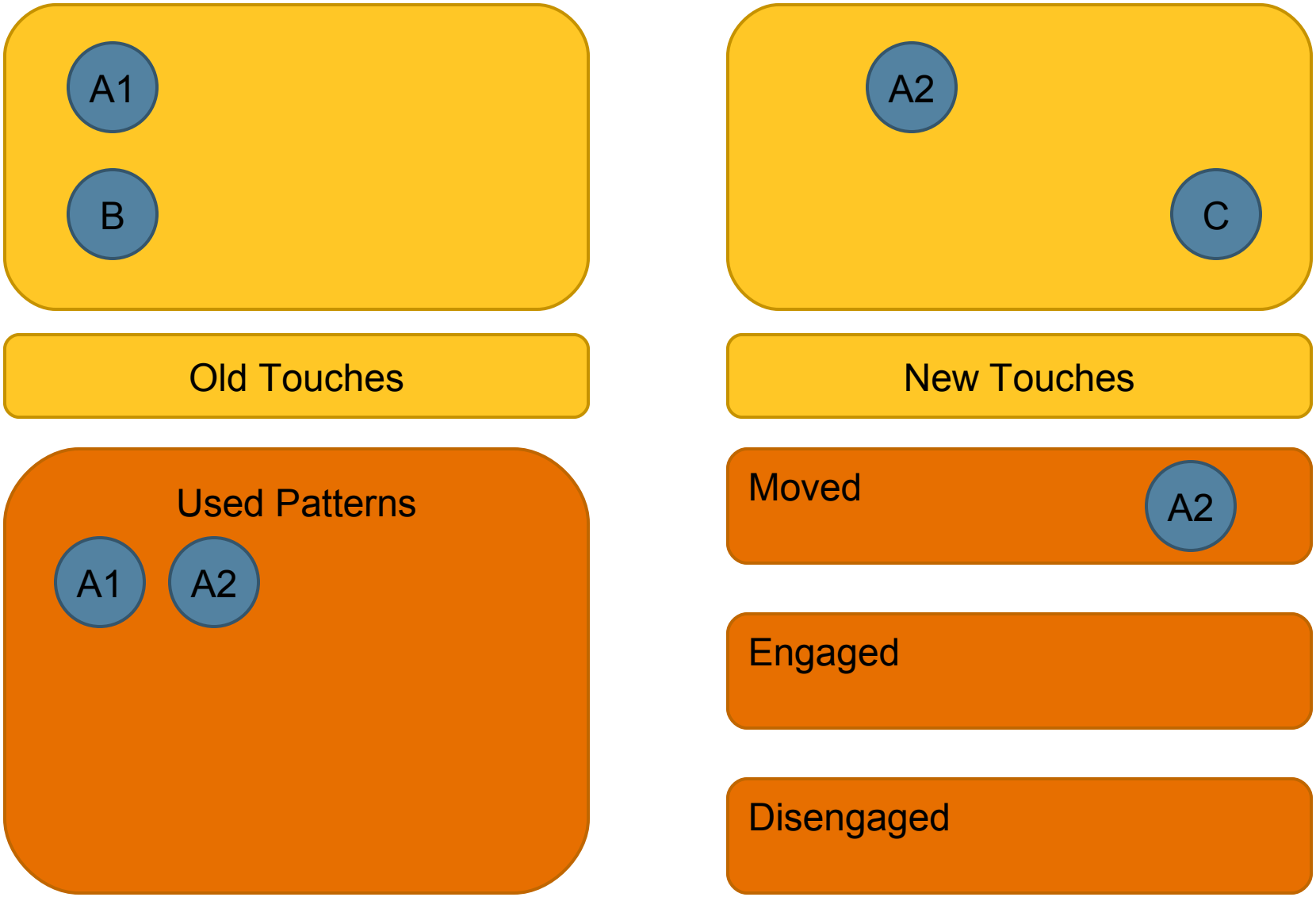
Three Main Pieces

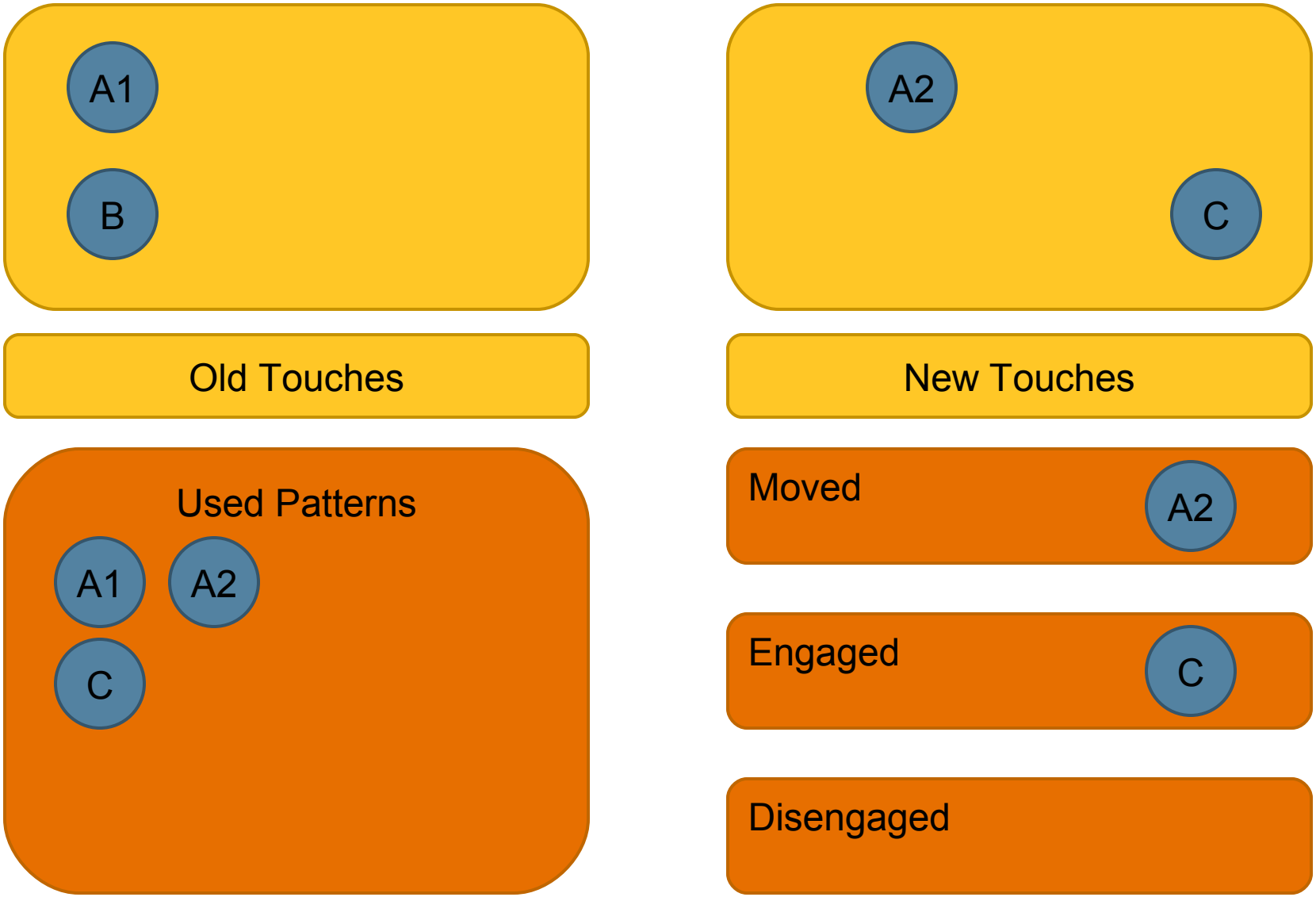


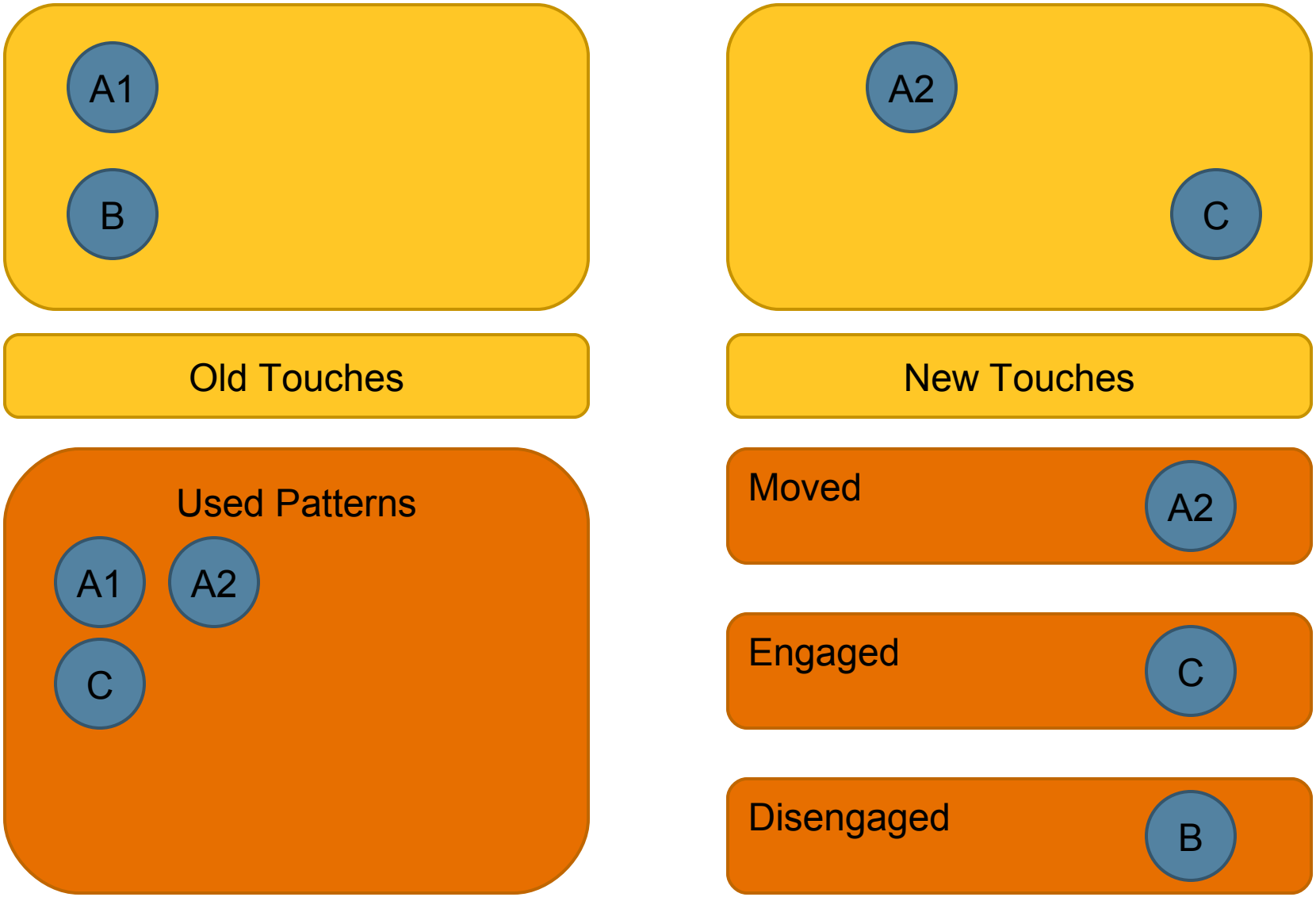
Event Generation

- Pluggable!
- Basic Strategy
  - Use Pattern List from Previous Call
    - New List and Old List
  - Match New Pattern List to Old Pattern List
    - If there is a new pattern near an old pattern and they are the same type, then this is a drag event.
      - Mark these matched patterns as “Used”
    - Anything unused in the New List is a Engage Event
      - Give this new pattern a unique ID (sequential)
    - Anything unused in the Old List is a Disengage Event


A1
B
A2
C
Old Touches
New Touches
Used Patterns
Moved
Engaged
Disengaged


A1
B
A2
C
Old Touches
New Touches
Used Patterns
A1
A2
Moved
A2
Engaged
Disengaged

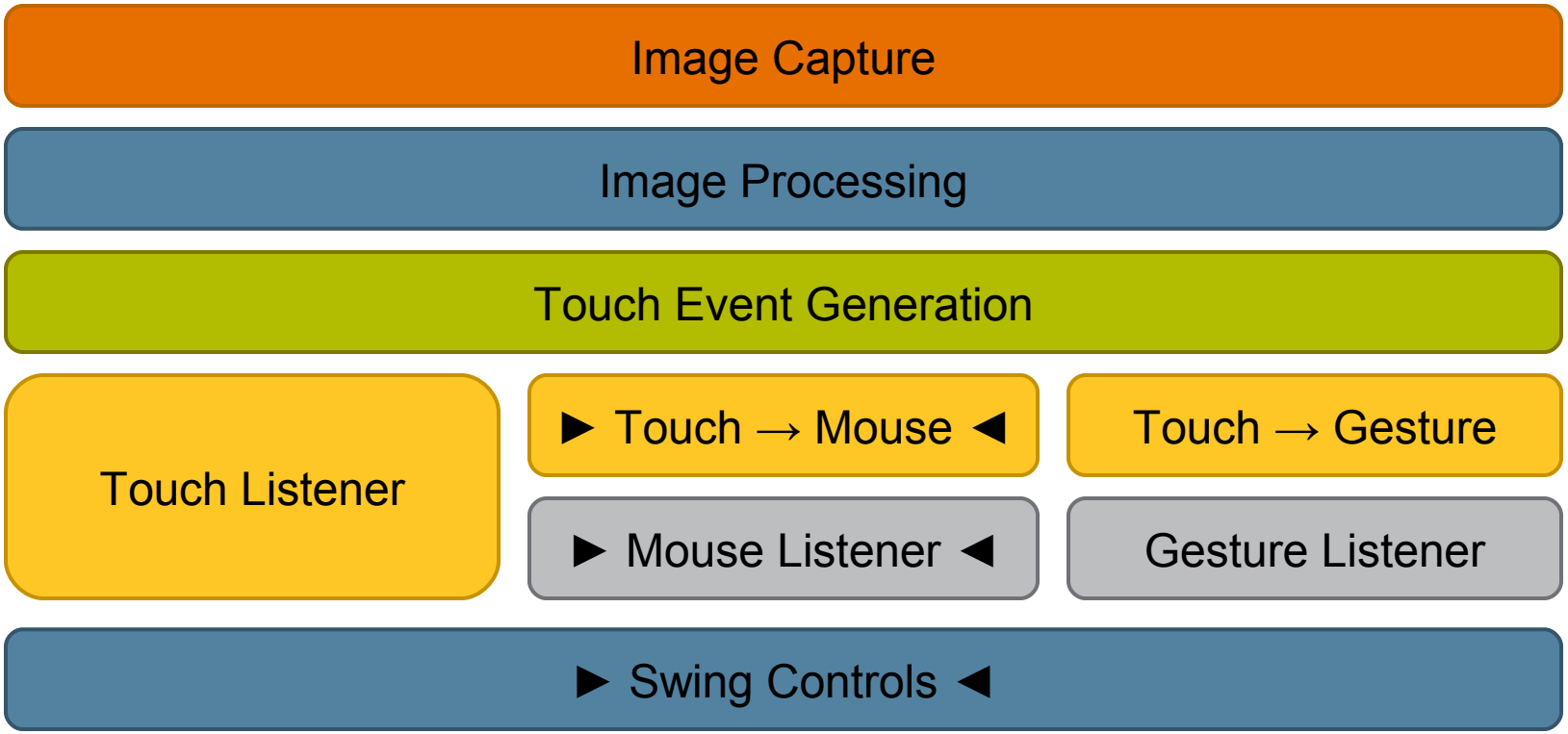

A1
B
A2
C
Old Touches
New Touches
Used Patterns
A1
A2
C
Moved
A2
Engaged
C
Disengaged


A1
B
A2
C
Old Touches
New Touches
Used Patterns
A1
A2
C
Moved
A2
Engaged
C
Disengaged
B

Agenda

- Multi-Touch Defined
- Delivering User Input
- “Regular” Swing Interfaces
- Multi-Touch Swing Interfaces
- A Quick Sample
- Questions

Swing's Place



Touches to Clicks

- SimpleSwingTableApplication extends JPanel, receives TableEvents and translates them into plain old Mouse Events

```
Component c =
    SwingUtilities.getDeepestComponentAt( this, x, y );

MouseEvent newEvent =
    new MouseEvent( c, id, time, mods, x, y, 1, false );

c.dispatchEvent( newEvent );
```

Touches to Clicks

- Works With Basic Swing Applications
  - Works best with applications designed for a single mouse
  - This **can** also work for multiple mice/inputs/fingers, but only for simple interactions. Remember that focus often gets set to the control that is clicked (which may muck things up).

Touches to Characters

- No Keyboard… what's a boy to do?
- Software Keyboard
  - Configurable location (border layout or overlay*)
  - Keyboard is simply another Swing Component
  - Key events are generated in the same way as mouse events, only they are sent to the control that **currently has focus**
  - This is still a single-user keyboard though… but that's because we're still talking about "standard" Swing applications.
  - The virtual keyboard **does not steal focus**

Agenda

- Multi-Touch Defined
- Delivering User Input
- “Regular” Swing Interfaces
- **Multi-Touch Swing Interfaces**
- A Quick Sample
- Questions

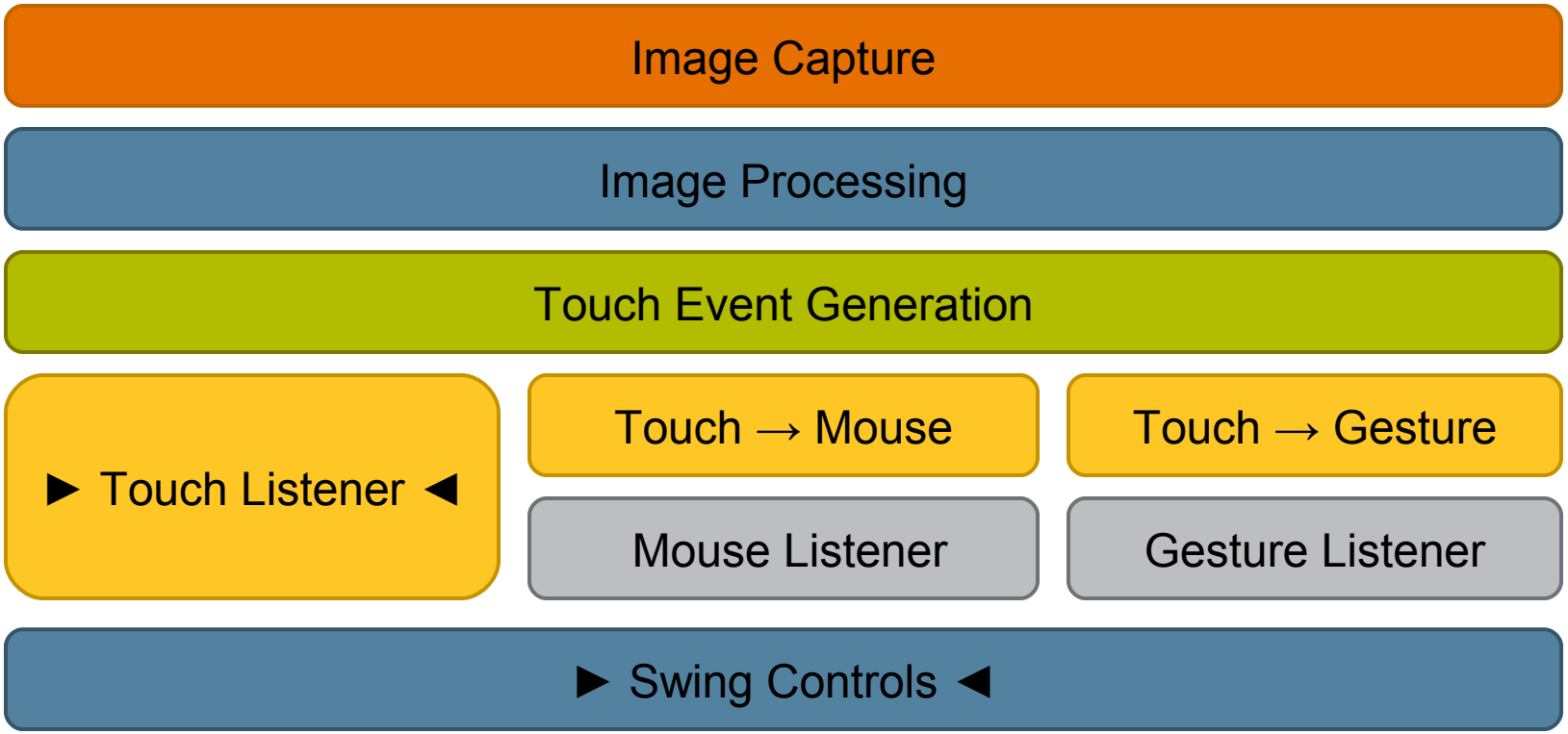
Extending Swing

- Surprising little to do... for the simple cases
- Buttons react well enough to multiple touches so long as there is only one touch per control.
- For multiple touches we need to start implementing special classes.

Extending Swing

- MTComponent (Multi-Touch component)
  - Implements TableEventProcessor – rebroadcasts Table Events to subscribed listeners.
  - Implements GestureProcessor too.
- Then we go down the hierarchy, implementing MT*
- You can “roll your own” by implementing TableEventProcessor and GestureProcessor and dispatching events to listeners (TableEventListener and GestureListener)

Swing's Place



Extending Swing

- The trick here is that we are NOT using the mouse event.
- With multiple touches (and multiple users) we must escape the concept of a single control having focus.
- Code that relies on focus being set must be modified.

Sending Table Events

```
// Just like before
Component c =
    SwingUtilities.getDeepestComponentAt( this, x, y );

// But now we're sending Table Events
if( c instanceof TableEventProcessor ) {
  TableEventProcessor tep = (TableEventProcessor)c;

  if( event instanceof TableEngagedEvent ) {
    tep.engaged( (TableEngagedEvent) event );
  }
  // dragged
  // disengaged
}
```

TableEvent

```
public abstract class TableEvent {
  private long eventID;
  private long groupEventID;
  private TableEventProvider tableEventProvider;

  // Picture & location
  private Pattern pattern;
}
```

TableDraggedEvent

```
public class TableDraggedEvent extnds TableEvent
  private Pattern startingPattern;
  private Pattern endingPattern;

  private long startTime;
  private long eventTime;

  private double distance;
}
```

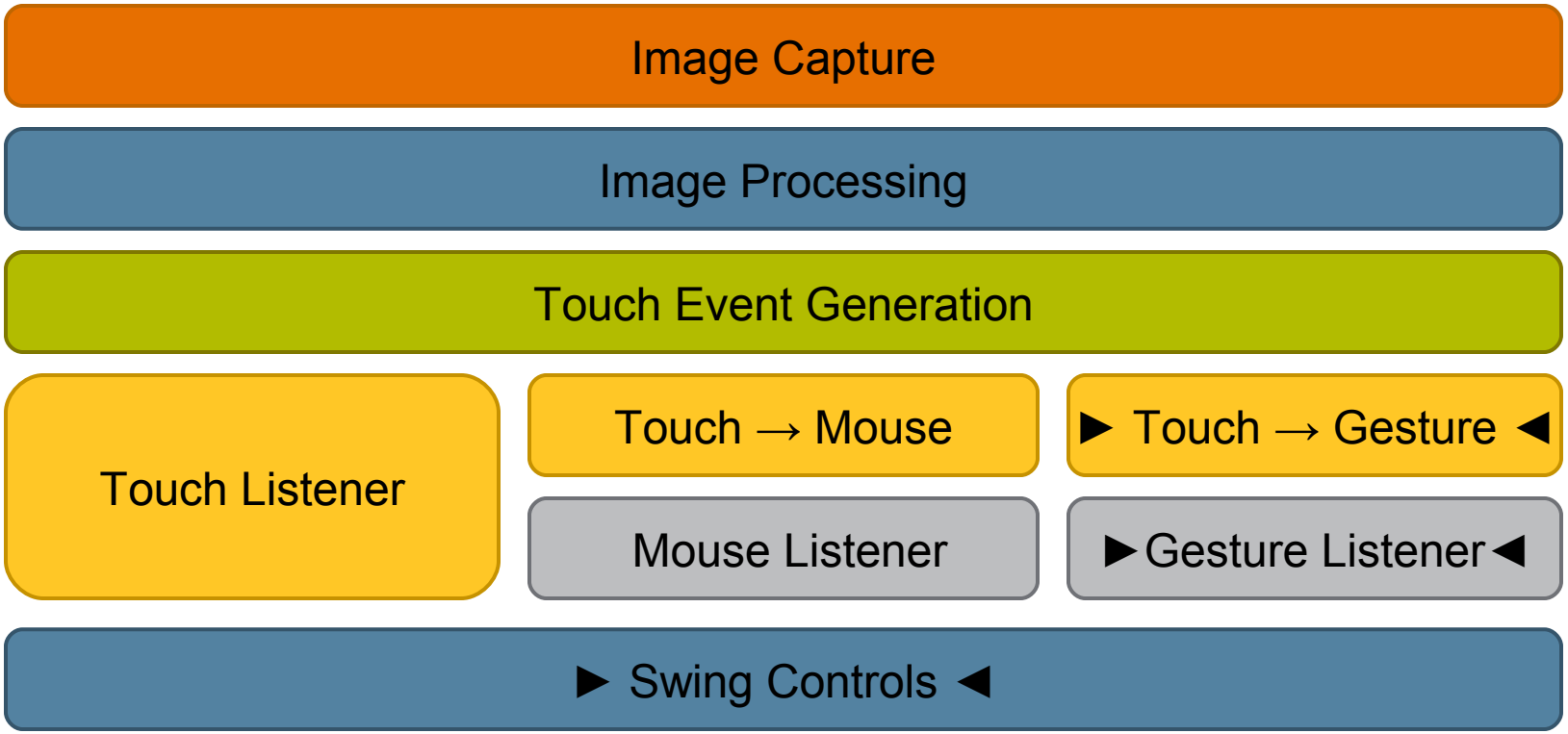
Multiple Keyboards

- When an MTComponent gets “focus” (when it is touched), it can display a “local” keyboard.
- You can either have one keyboard per application, which will close automatically –or- you can have multiple keyboards, which must be manually closed.
- Keyboards send key events to the control to which they’re bound, not to the application.
- This allows multiple users to enter data at the same time.
- Of course, you can muck with this to do your own thing…

Gestures

- Finger Movements that MEAN Something.
- Patents on Gestures
- User makes a specific motion on the device and it is interpreted as a gesture.
- Gesture event is sent to the component where the gesture **started**
- If the gesture requires two or more fingers, then all of the fingers need to be in the same component at their start

Swing's Place



Gesture Processing

- Each Gesture is tied to an action on an APPLICATION LEVEL.
- GestureListener is invoked, passing a Gesture object.
- Gesture Object contains a String with the name of the gesture, along with some telemetry that describes the gesture.

Gesture Processing

- The user can react to the gesture based on the String.
  - This is kind of how menus work
- Gestures are described in XML and loaded with the application.
- Soon, they can be “painted” using a special editor and saved, or composed and saved as part of a user profile.

Defining a Gesture – "L"

```
<SimpleGesture>
  <gestureName>LDown</gestureName>
  <actuatorName>fingertip</actuatorName>
  <actionName>Close</actionName>
  <Sequence>
    <Tolerance>...</Tolerance>
    <State>
      <pos>
        <x>0</x>
        <y>0</y>
      </pos>
    </State>
    <State>
      <pos>
        <x>0</x>
        <y>100</y>
      </pos>
    </State>
```

Defining a Gesture – “L”

```
    <State>
      <pos>
        <x>100</x>
        <y>100</y>
      </pos>
    </State>
  </Sequence>
</SimpleGesture>
```

Advanced Gestures

- Multiple Sequences
  - With corresponding time states
- Circular Gestures
  - Just a whole lot easier to use than coding to hit 360 (or 36 or whatever) points

Responding to Gestures

- Target is the Component in Which the Gesture “Started” and implements GestureProcessor

```
public interface GestureProcessor {
  public void processGesture( GestureEvent evt );
}
```

Agenda

- Multi-Touch Defined
- Delivering User Input
- “Regular” Swing Interfaces
- Multi-Touch Swing Interfaces
- A Quick Sample
- Questions

A Quick Example

```
public class FingerPaint extends TableApplication {

  // lifecycle method
  public void start() {

    // create a canvas
    this.canvas = new BufferedImage( this.width,
            this.height, BufferedImage.TYPE_INT_RGB );
  }

}
```

A Quick Example

```
public void dragged(TableDraggedEvent evt) {

  BufferedImage canvas = getCanvas();
  Graphics g = canvas.getGraphics();
  Point p1 = evt.getStartingPattern().getCenterPoint();
  Point p2 = evt.getEndingPattern().getCenterPoint();

  p1 = TableManager.translatePoint( p1 );
  p2 = TableManager.translatePoint( p2 );

  g.drawArc( p1.x, p1.y, 20, 20, 0, 360 );
  g.drawLine(p1.x + 10, p1.y + 10, p2.x + 10, p2.y + 10 );
  g.drawArc( p2.x, p2.y, 20, 20, 0, 360 );

  this.repaint();
}
```

Agenda

- Multi-Touch Defined
- Delivering User Input
- “Regular” Swing Interfaces
- Multi-Touch Swing Interfaces
- A Quick Sample
- Questions

For More Information

- Other Multi-Touch Projects
  - NUI Group
    - http://www.nuigroup.com
    - C++ / TouchLib
  - Whitespaced
    - http://blog.whitespaced.co.za/
    - C# Multi-Touch project
  - Multi-Pointer X-Server (MPX)
    - Linux Multi-Touch
    - http://wearables.unisa.edu.au/mpx/
  - More… but you know how to Google

Summary

- The Hardware Is Easy… Kind Of
  - Provided you're not afraid of a little soldering and a little woodworking
- The Software is Free…
- If You Can Code Swing…
  - You can code multi-touch
- Collaborate and Share!
  - http://www.open-table.org

QUESTIONS

THANK YOU

Michael Riecken
Java Capability Leader, Trissential LLC
michael@open-table.org (don’t be shy)