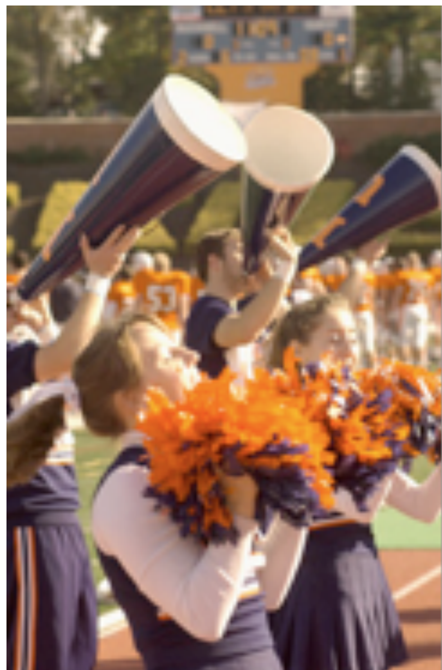
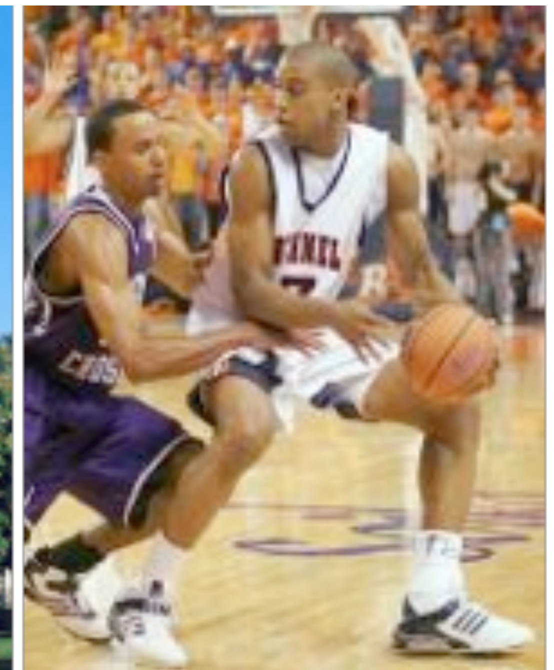
FlyLoop: A Micro Framework for Rapid Development of Physiological Computing Systems

Evan M. Peck, Eleanor Easse, Nick Marshall, William Stratton, and L. Felipe Perrone

Department of Computer Science
Bucknell University, PA, U.S.A.



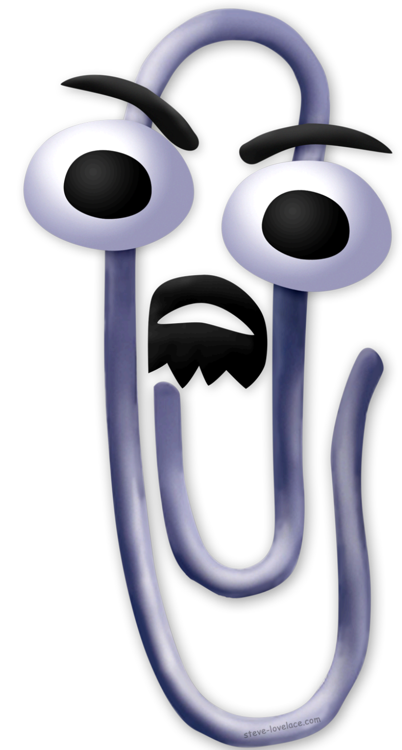


Physiological Computing Systems

"Research on adaptive systems, especially in the area of affective computing, places enormous emphasis on the capacity of the machine to monitor and make accurate inferences about the psychological state of the user."

S.H. Fairclough 2015

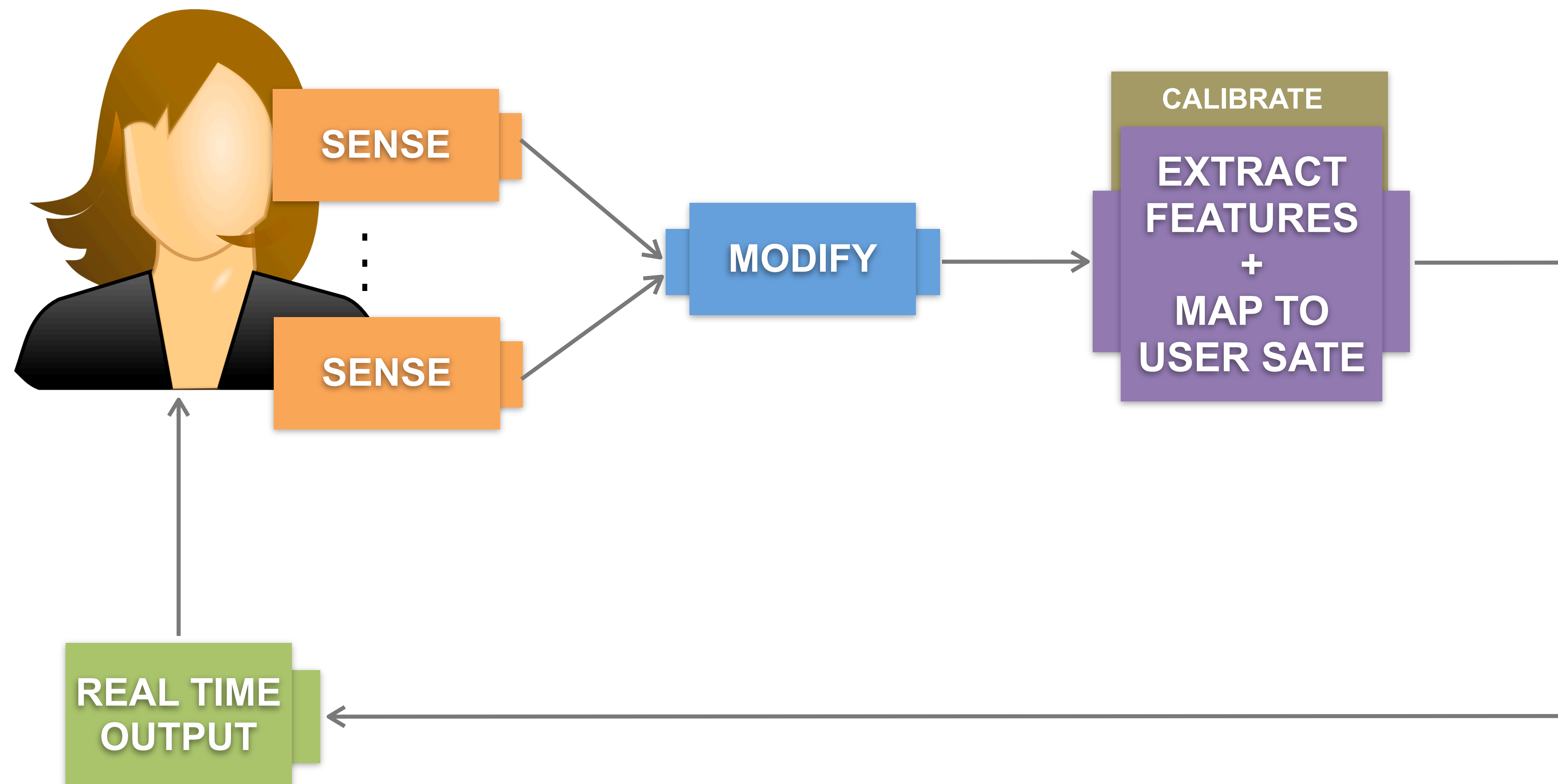
http://physiologicalcomputing.org/2015/03/we-need-to-talk-about-clippy/



Wish List

- Adaptive systems that can sense the psychological state of the user and can react intelligently to physiological input.
- Easily built custom physiological computing systems for rapid prototyping, replication, and validation.

The Biocybernetic Loop



MUSE Brain Sensing Headband

Tobii EyeX Controller

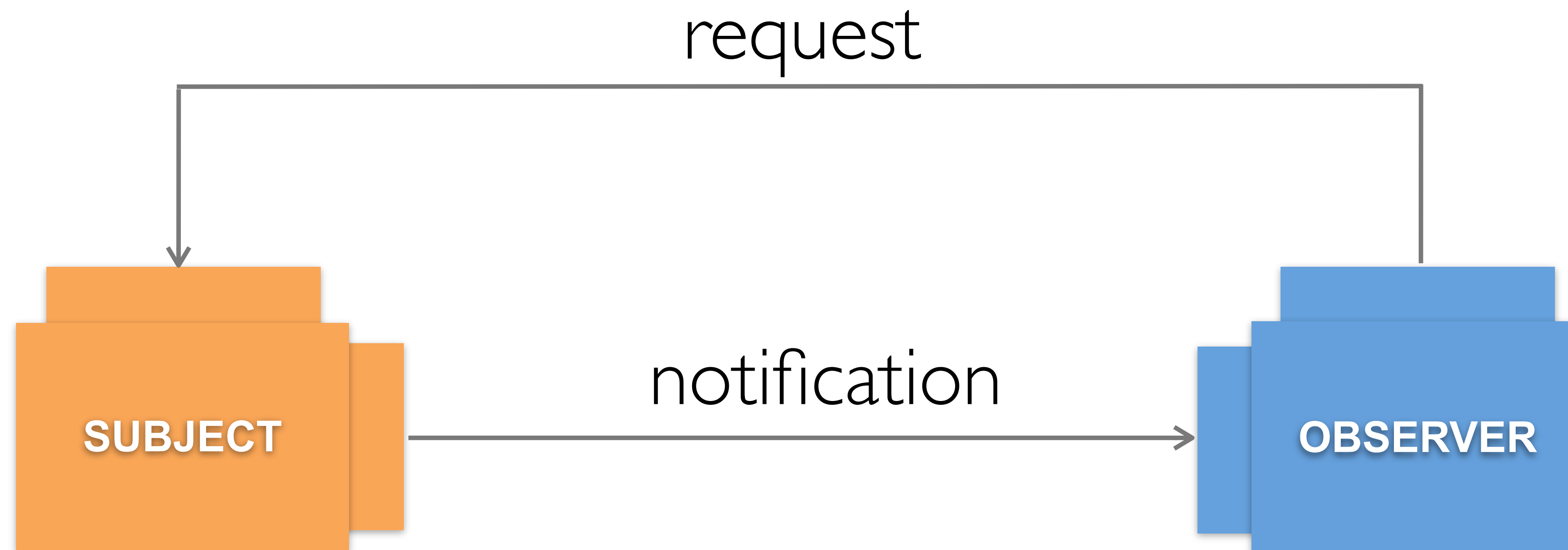
Apple Watch

IMAGE: APPLE

Challenges

- Build custom systems for proof of concept investigations and rapid prototyping. Easy to make them inflexible.
- Support any number of data sources with varying velocity and modality.
- Couplings between components of the pipeline can be unwieldy.

Observer Design Pattern (Gamma et al. 1995)

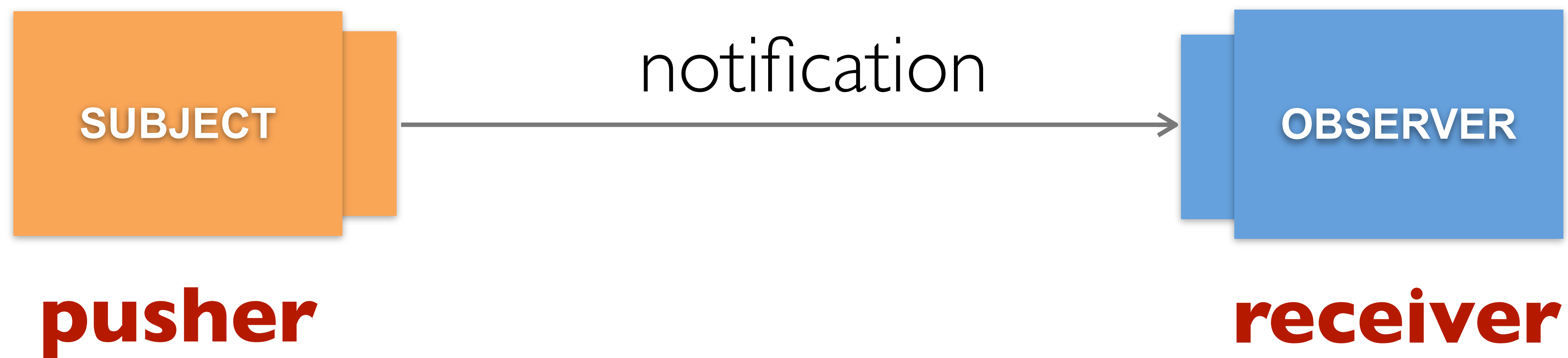


Observer Design Pattern (Gamma et al. 1995)

Push Model

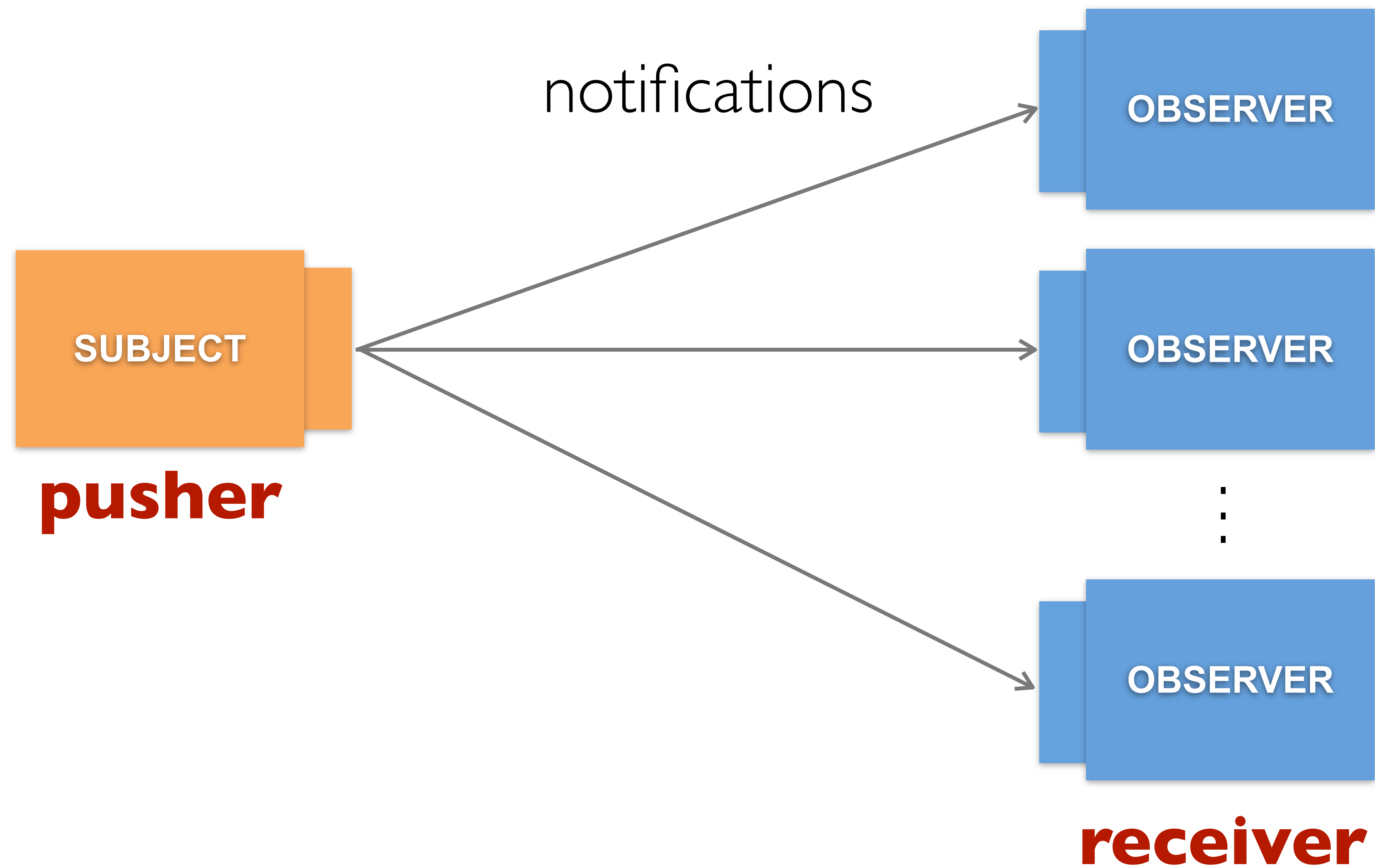
Observer Design Pattern (Gamma et al. 1995)

Push Model



Observer Design Pattern (Gamma et al. 1995)

Push Model



FlyLoop

A lightweight, minimal programming framework whose goal is usability for the developer.

FlyLoop is a Java microframework with foundation classes and interfaces based on a data flow model.

FlyLoop Core Modules

RECEIVER PUSHER

DATASOURCE

- Wrapping sensor specific data streams

FILTER

- Manipulation of data streams

CALIBRATOR

LEARNER

- Mapping of data to user state

OUTPUT

- Marshall output to specific needs

FlyLoop Core Modules

- Modules inherit from a Receiver class and/or implement a Pusher interface
- Data is transferred in or out according to a system-wide polling rate
- Data is passed around as a Java Object

DataSource

Interfaces with any kind of streaming sensor; makes no assumptions about incoming data

A few core operations:

- startCollection : activates component
- getOutput : returns single data point
- push : sends data point to receivers

Filter

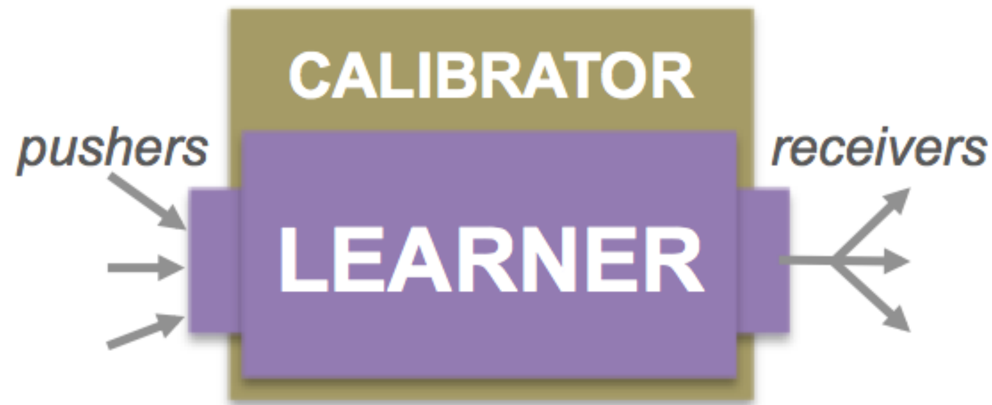
Applies some transformation on data stream

A core operation:

- getDataPoints : ask for a window of data of arbitrary size

Designed to allow developer to focus on signal processing algorithm rather than on timing of data transfer

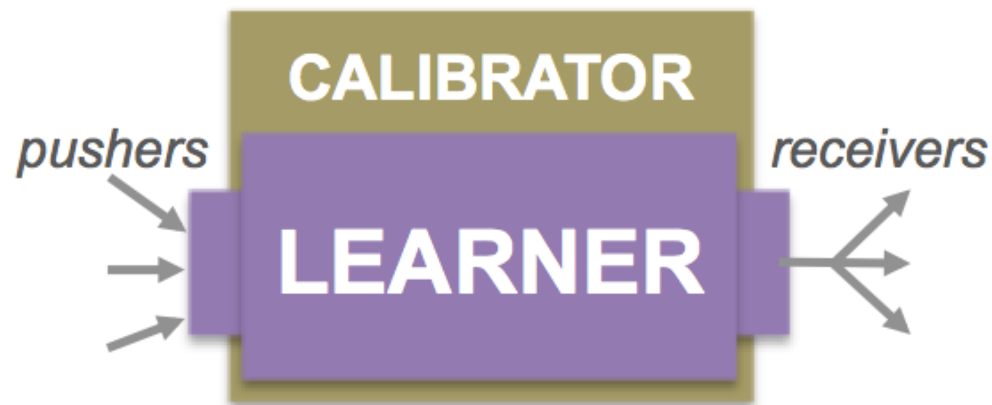
Calibrator



Small set of built-in functions to communicate labels on training data to Learner and to determine mode of operation (training and predicting)

Encapsulates training tasks.

Learner



Takes input from any DataSource or Filter; builds model based on training data or outputs real-time classification based on new data

A lot of complexity is encapsulated in this module (statistics, machine learning).

A core operation:

- learn : ask for a window of data of arbitrary size

Output

Any component in the pipeline can push to it.

This generic component marshals data into different formats for different purposes. For instance:

- Create logs,
- Interface with data visualization tools,
- Transmit model classifications over the network,
- etc.

Synchronization

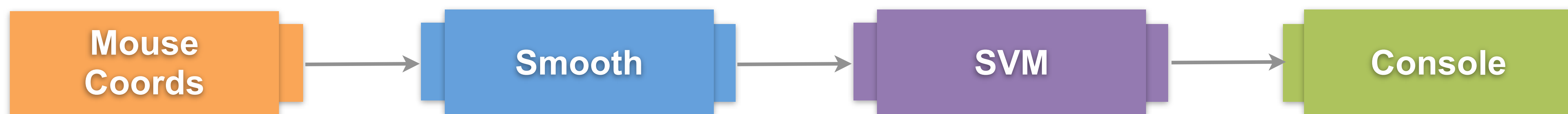
When a component takes input from multiple DataSources, sampling rates may not match.

A sample is available from one source, but not from another. Possible behaviors:

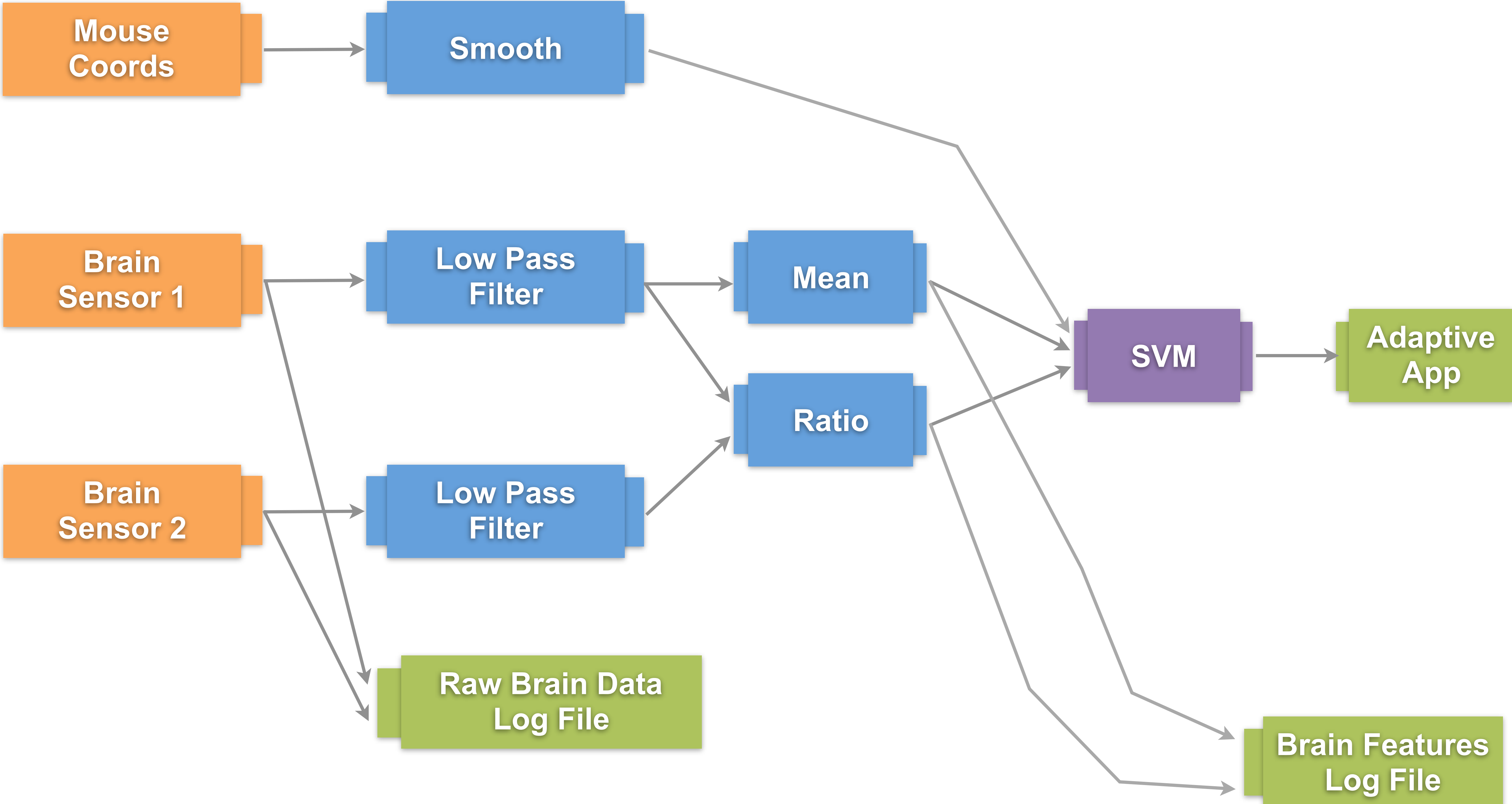
(1) Repeat the previous input when there is no new data.

(2) Push null values when there is no new data.

Simple Example



Complex Example



Limitations

- Filters are constrained to online algorithms
- Component structure makes it challenging to create manipulations that require a global view of data

Ongoing and Future Work

- Refactoring / Cleaning up
- Public release under MIT license
- Creating persistent configuration files to enable replication
- Creating a configuration language on top of FlyLoop (compiles to Java) - possibly visual

Thanks for your attention!

Questions?

Pusher (Interface)

```
public interface Pusher {

  public Receiver[] getReceivers();
  public void setReceivers(Receiver receivers[]);
  public void setReceivers(Receiver receiver);
  public void push();

}
```

Receiver (Abstract Class)

```
public abstract class Receiver {

  public void addSource(int source);
  public void addSources(int[] sources);
  public ArrayList<Integer> getSourceIDs();
  void receive(Object data, int id);
  public Object[] getData();
  public Object getDataPoint(int i);
  public abstract void processData();

}
```

Filter (Abstract Class)

```
public abstract class Filter extends Receiver implements Pusher {

    private Receiver[] receivers;
    private Object outputBuffer;
    private Queue<Object[]> allData;
    private int interval;
    public Filter(int interval, int max);
    public Filter(int interval);
    public Filter();
    public Object[] getInterval();
    public Object[] getInterval(int i);
    public Object[] getInterval(int i, int n);
    public abstract Object filterData();
    public void push();
    public Receiver[] getReceivers();
    public void setReceivers(Receiver[] receivers);
    public void setReceivers(Receiver receiver);
    public void processData();

}
```

Calibrator (Abstract Class)

```
public abstract class Calibrator implements Runnable {

  public Calibrator(String[] states, Learner[] learners,
                    String inFile, String outFile);

  public Calibrator(String[] states, Learner learner,
                     String inFile, String outFile);

  public Calibrator(String[] states, Learner learner);

  public Calibrator(String[] states, Learner[] learners);

  public void startCalibration();
  public abstract void skipCalibrator();
  public abstract void initCalibrator();
  public void finishCalibrating();
  public abstract void calibrate();
}
```

Learner (Abstract Class)

```
public abstract class Learner extends Receiver implements Pusher {
  public Learner(int max, boolean outputConfidence);
  public void processData();
  public String getState();
  public boolean isCalibrating();
  public void setCalibrating(boolean isCalibrating);
  public void startCalibrating();
  public void stopCalibrating();
  public void pauseCalibrating();
  public String getCalibrationState();
  public void setCalibrationState(String calibrationState);
  protected void setState(String state);
  public void push();
  public Double getConfidence();
  public void setConfidence(Double confidence);
  public Receiver[] getReceivers();
  public void setReceivers(Receiver[] receivers);
  public void setReceivers(Receiver receiver);
  public abstract void learn();
}
```

Data Source (Abstract Class)

```
public abstract class DataSource implements Pusher {

  public DataSource();
  public DataSource(boolean repeat);
  public abstract void startCollection();
  public void push();
  public Receiver[] getReceivers();
  public void setReceivers(Receiver[] receivers);
  public void setReceivers(Receiver receiver);
  public abstract Object getOutput();

}
```

Output (Abstract Class)

```
public abstract class Output extends Receiver {

  public Output(boolean stateChange, int max);
  public void processData();
  public abstract void output();

}
```