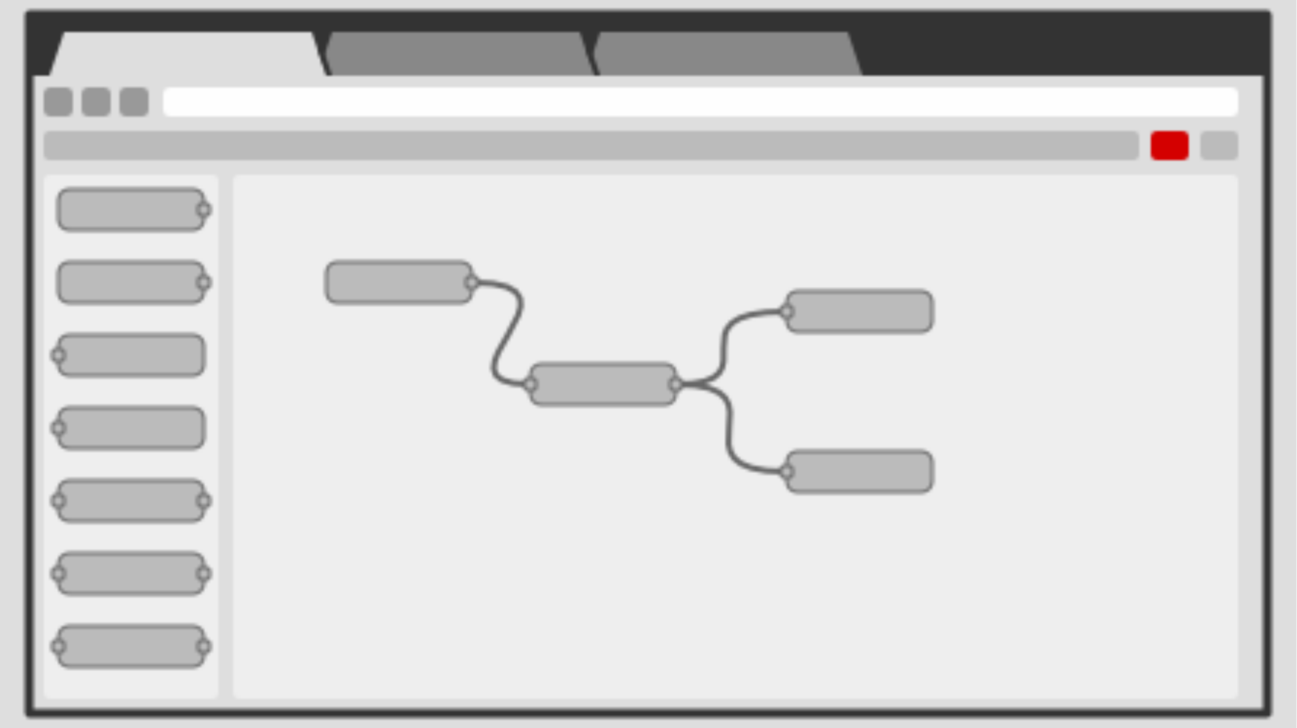
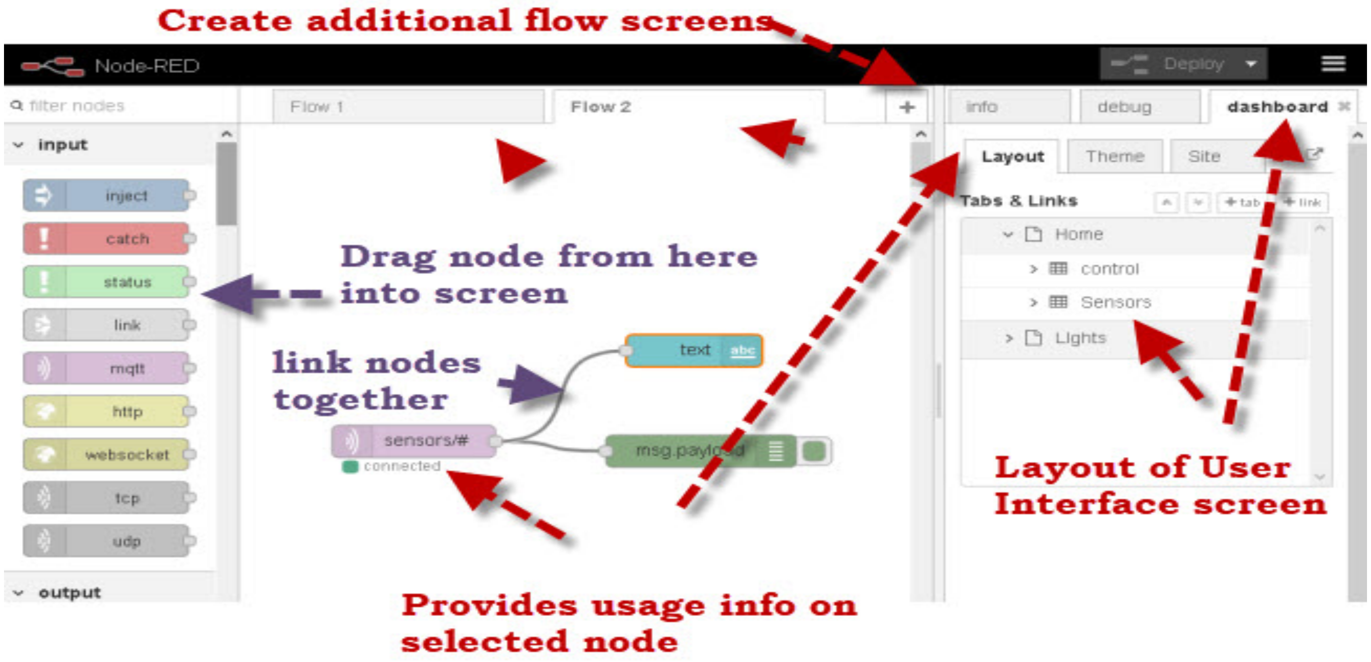
Carnegie
Mellon
University

95-733 Internet of Things
Flow Based Programming and Edge Analytics

Node-Red is a browser based flow language



Node-Red



Create additional flow screens

Drag node from here into screen

link nodes together

Provides usage info on selected node

Layout of User Interface screen

Node Red

- Flow Based Programming created by J. Paul Morrison (1970's).
- Node-Red is a visual flow based tool based on Node.js.
- Each black box does one thing well. >750 boxes available.
- Built for programmers and non-programmers.
- No or little programming. Hmmm.
- Two short videos:

  Node Red Introduction:

  https://developer.ibm.com/components/node-red/videos/node-red-essentials

  Node Red Fundamentals

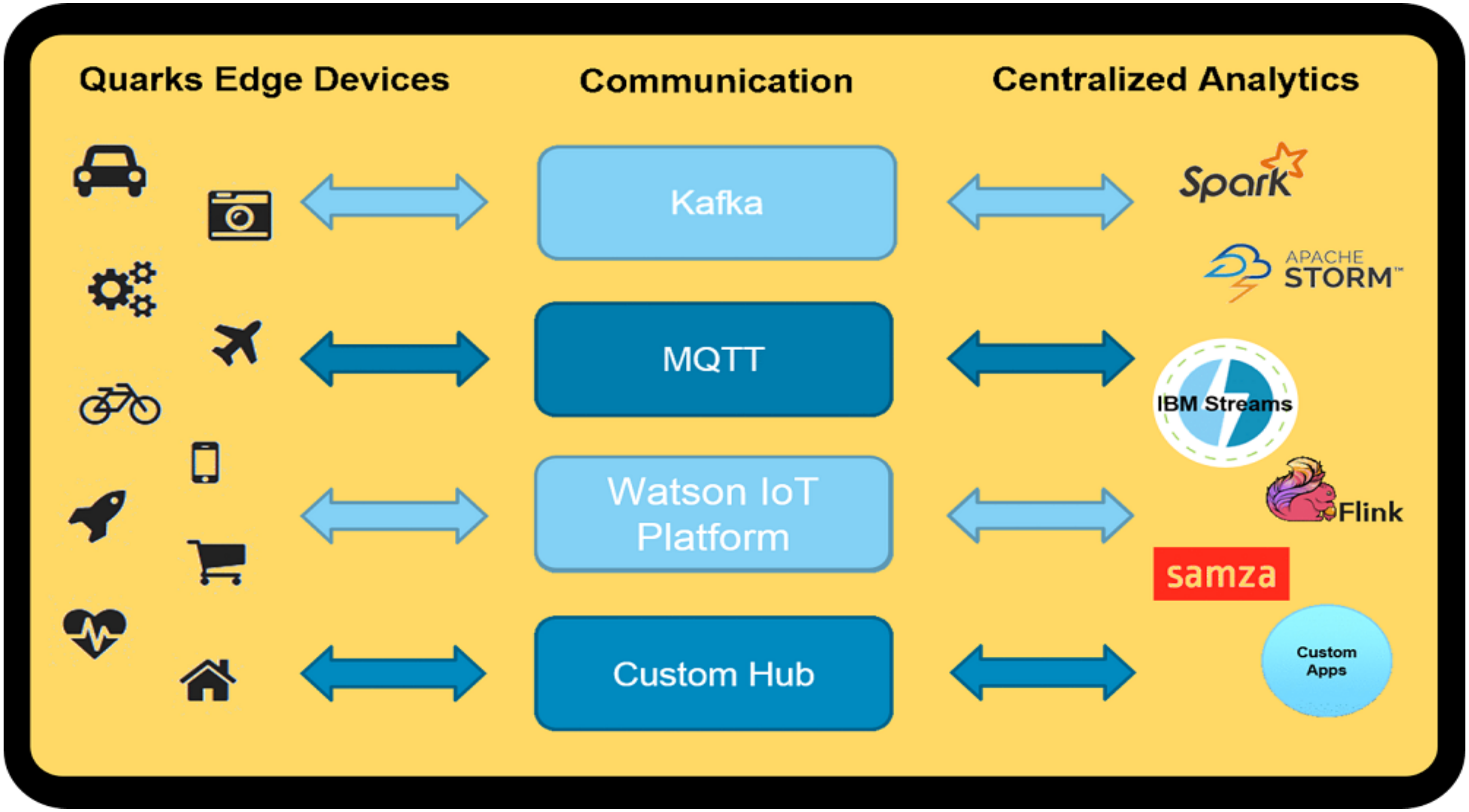
  https://www.youtube.com/watch?v=3AR432bguOY

Definitions

- Edge analytics is an approach to data collection and analysis in which an automated analytical computation is performed on data at a sensor, network switch or other device instead of waiting for the data to be sent back to a centralized data store. – WhatIs.com.
- The edge itself is a constrained area:

  Constraints include weight, space, cost, battery life, disconnected operation, intermittent networks, limited connectivity, cost of network usage, etc.
- An edge environment may contain a half dozen sensors or thousands of sensors.
- We might need a global view of what is going on on the edge.

Centralized and Edge Analytics



From http://edgent.incubator.apache.org

Apache Edgent

- IBM Quarks launched in February 2016.
- Became Edgent and open sourced to Apache.
- Designed for edge analytics on a constrained device.
- IBM’s Node Red, Apache Spark Streaming and Apache Flink are typically found on the back end.
- Front end analytics important but may not be as rich as data stores on the backend.
- Edgent is an SDK for the edge (you pick and choose what to deploy).
- You may run on the edge with no communications or only intermittent connectivity.

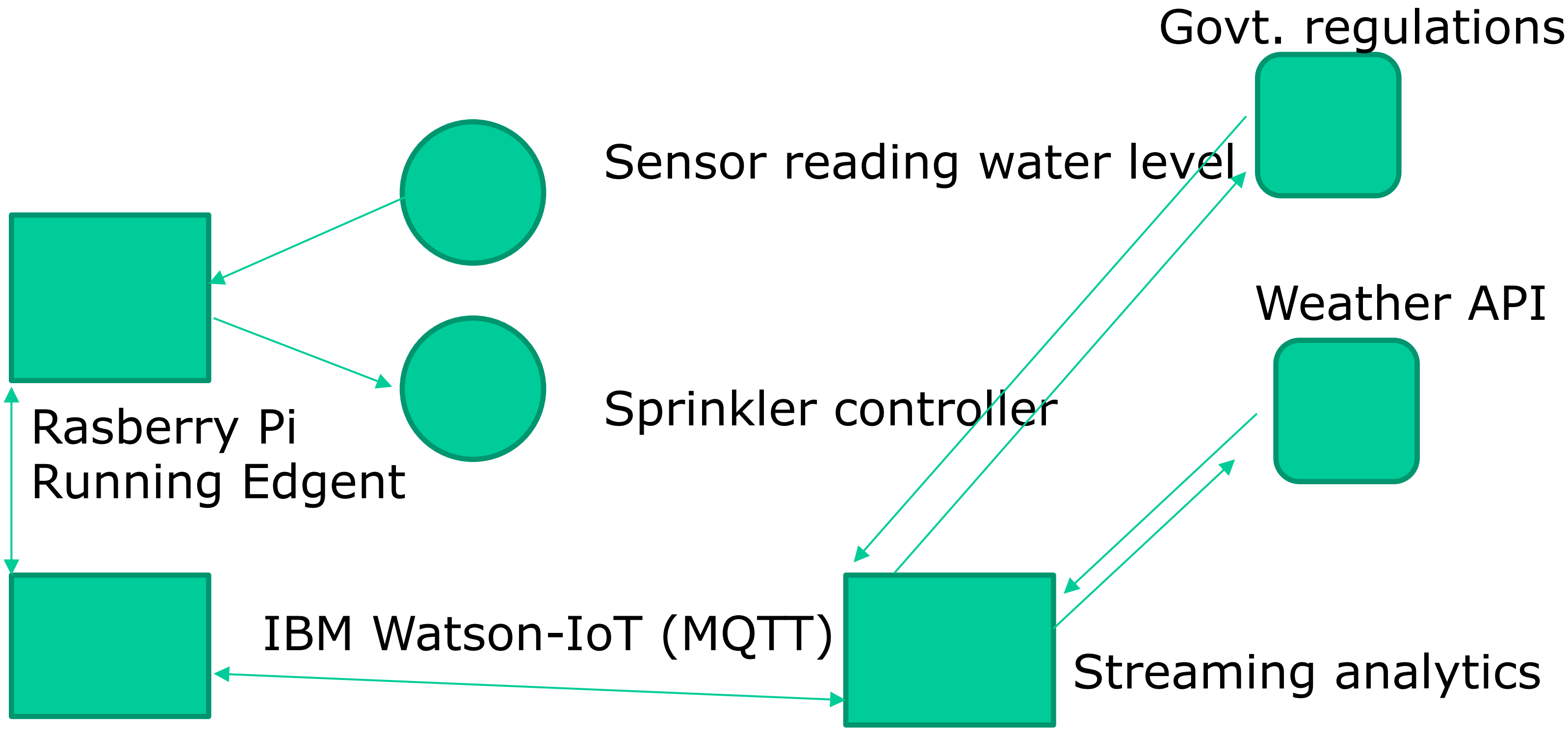
Edgent

- May run on Rasberry Pi or Android devices
- Currently Java based and does not run on Swift or iPhone
- A simple linux box on the edge can run Java and Edgent
- Edgent is a programming model (functional flow API) and a lightweight embeddable runtime for edge analytics

Edge and Centralized Analytics

- Less and more selective communication to backend.
- Make local decisions (valuable especially when disconnected).
- Central analytics system is not constrained like the edge. Multiple devices may be reporting to the central analytics system.
- The edge may receive commands from the central analytics system, for example, central may ask the edge to report more often if conditions require.
- Central analytics is not required but is a likely pattern. Perhaps you only require local decision making.
- The Central analytics system may have access to systems of record as well as a much wider variety of data over many devices and types of data.

Cool Edgent Use Case



Govt. regulations

Sensor reading water level

Weather API

Sprinkler controller

Rasberry Pi
Running Edgent

IBM Watson-IoT (MQTT)

Streaming analytics

https://youtu.be/Rvc1CqNJkOA?list=PLhZR82i0P9NqrksME13f2t8tDMIhxUtCH

Edgent

- Functional flow API for streaming analytics (Map, Flatmap, Filter, Aggregate, Split, Union, Join, Deadband filter)
- Connectors (MQTT, HTTP, Websockets, JDBC, File, Kafka, IBM IoT Watson)
- For example, the Java API allows you to send JSON to an MQTT device
- Bi-directional communications with the backend
- Edgent uses Java Lambda expressions.
- Let’s pause and look at Lambda expressions…

Java Lambda Expressions (1)

```
// ListenerTest, an example not from Edgent
package java.awt.event;
import java.util.EventListener;
public interface ActionListener extends EventListener {
        public void actionPerformed(ActionEvent e);
}
// An interface with only one method is called a functional interface.
// These interfaces are common in Java. See Runnable and Comparator.
//  What is required to implement this interface?
// Use lambda expressions for functional interfaces.
```

Java Lambda Expressions (2)

```
// Suppose we do not use lambdas and create an anonymous
// inner class to listen on a button

JButton testButton = new JButton("Test Button");
testButton.addActionListener(new ActionListener(){
   @Override public void actionPerformed(ActionEvent ae){
         System.out.println("Click Detected by Anon Class");
    }
  }
);
```

Java Lambda Expressions (3)

```
 // add a second action listener using lambdas
testButton.addActionListener(
            j -> System.out.println("This click  Detected by Lambda Listner"));


  JFrame frame = new JFrame("Listener Test");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.add(testButton, BorderLayout.CENTER);
  frame.pack();
  frame.setVisible(true);
 }
}
```

The single method takes a single argument. We are implementing the method with the lambda expression. In this case, we are not using j in the method.

Java Lambda Expressions (4)

A lambda expression is composed of three parts:

Argument List	**Arrow Token**	**Body**
(int x, int y)	->	x + y;
(String x)	->	Sytem.out.println(x);
j	->	System.out.println("Hi"));

The body can be either a single expression or a statement block.
It completes the single abstract method in a functional interface.
The class of j may be figured out by the compiler.

Java Lambda Expressions (5)

```
// This interface is functional - only one method
interface TestInterface {
      public void sayHelloToWhoever();
}

// This interface is functional - only one method
interface TestInterface2 {
public void sayHelloToWhoever(String x);
}
```

Java Lambda Expressions (6)

```
// Make a call on an implementation of
// TestInterface2
public static void foo(TestInterface2 y) {
     y.sayHelloToWhoever("Amy");
}
```

Java Lambda Expressions (7)

```
public class TestLambda {
    public static void main(String...args) {
    // We need an implemenation of the TestInterface interface.
    // The lambda expression provides that.
    // The method takes no parameters.
    TestInterface i = () -> System.out.println("Mike");
     i.sayHelloToWhoever();

     // In TestInterface2, we need to handle x in the method.
    //  The compiler can figure that x is a String. We can drop “String”.
    TestInterface2 j = (String x) -> System.out.println(x + " is cool.");
     j.sayHelloToWhoever("Sam");

    // pass around a code block
    foo(j);
    // again
    foo(x -> System.out.println("Wow"));
}
```

Java Lambda Expressions (8)

```
package runabletest;
public class RunnnableTest {
    public static void main(String[] args) {
         System.out.println("=== RunnableTest ===");
         // Anonymous classes - provide the implementation
         // of run
        Runnable r1 = new Runnable(){
                 @Override  public void run(){
                      System.out.println("Hello world one!");
        }
};
```

Java Lambda Expressions (9)

```
// Lambda Runnable
Runnable r2 = () -> System.out.println("Hello world two!");
    r1.run();
    r2.run();
 }
 }
=== RunnableTest ===
Hello world one!
Hello world two!
```

Edgent Flow Programming

http://edgent.incubator.apache.org/docs/streaming-concepts

Edgent Example(1)

```
import java.util.Random;
import quarks.function.Supplier;
// Every time get() is called, TempSensor
// generates a temperature reading.
public class TempSensor implements Supplier<Double> {
        double currentTemp = 65.0;
        Random rand;
        TempSensor(){
            rand = new Random();
        }
```

Edgent Example(2)

```
@Override   // the get() method defined in Supplier
public Double get() {
  // Change the current temperature some random amount
  double newTemp = rand.nextGaussian() + currentTemp;
  currentTemp = newTemp;
  return currentTemp;
 }
}
```

Edgent Example(3)

```
// First download the appropriate jars

import java.util.concurrent.TimeUnit;
import org.apache.edgent.providers.direct.DirectProvider;
import org.apache.edgent.topology.TStream;
import org.apache.edgent.topology.Topology;
```

Edgent Example(4)

```
public class TempSensorApplication {
    public static void main(String[] args) throws Exception {
        // implements Supplier
        TempSensor sensor = new TempSensor();

        DirectProvider dp = new DirectProvider();
        Topology topology = dp.newTopology();
        TStream<Double> tempReadings = topology.poll(sensor, 1,
                                              TimeUnit.MILLISECONDS);
        TStream<Double> filteredReadings =
                tempReadings.filter(reading -> reading < 50 || reading > 80);
        filteredReadings.print();
        dp.submit(topology);
    }
}
```

Edgent Example(5)

```
42.21773497632803
43.778600196956134
43.50474973480867
43.825909511894686
45.161912344306764
46.12672565018012
47.566025733982215
47.660160245707836
```