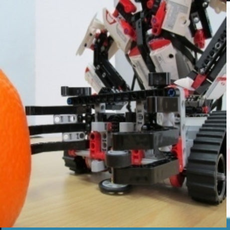
March 11, 2016
Voxxed Days Bucharest

# Reactive Java Robotics and IoT

**Trayan Iliev**
**CTO of IPT – Intellectual Products & Technologies**
http://robolearn.org/
http://iproduct.org/





# Tales of JAVA Robotics

There are so many tales to share…
Where should I start? ... from the Beginning …

- ❖ Tale of Common Sense: DDD
- ❖ Tale of Segregation between Queries and Commands, and ultimate Event Sourcing
- ❖ Tale of two cities - Imperative and Reactive
- ❖ Tale of two brave robots: LeJaRo and IPTPI
- ❖ And a lot of real reactive Java
  \+ TypeScript / Angular 2 / WebSocket code ☺

# Common Sense: DDD

There was a time upon …

When

were

and people

were

Simple times …

# Common Sense: DDD

But then computers came and became more

… and more powerful …

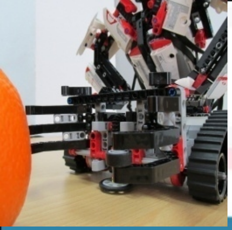
The MareNostrum supercomputer in the Barcelona Supercomputing Center - National Supercomputing Center (BSC-CNS), Spain. (Photo: IBM Research/Flickr)

# Common Sense: DDD

People could not easily cope with the complexity of problems being modeled anymore

Simple solutions are needed – cope with problems through divide and concur on different levels of abstraction: **Domain Driven Design** – back to basics: data, domain objects.

Described by Eric Evans in his book: Domain Driven Design: Tackling Complexity in the Heart of Software



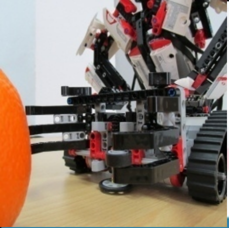


# Common Sense: DDD

Actually DDD require additional efforts (as most other divide and concur modeling approaches :)

- Ubiquitous language and Bounded Contexts
- DDD Application Layers: Infrastructure, Domain, Application, Presentation
- Hexagonal architecture :

OUTSIDE <-> transformer <-> ( application <-> domain ) [A. Cockburn]



# Common Sense: DDD

## Main concepts:

- Entities, value objects and modules
- Aggregates and Aggregate Roots [Haywood]:

**value < entity < aggregate < module < BC**

- Repositories, Factories and Services:

application services <-> domain services

- Separating interface from implementation

# CQRS and Event Sourcing

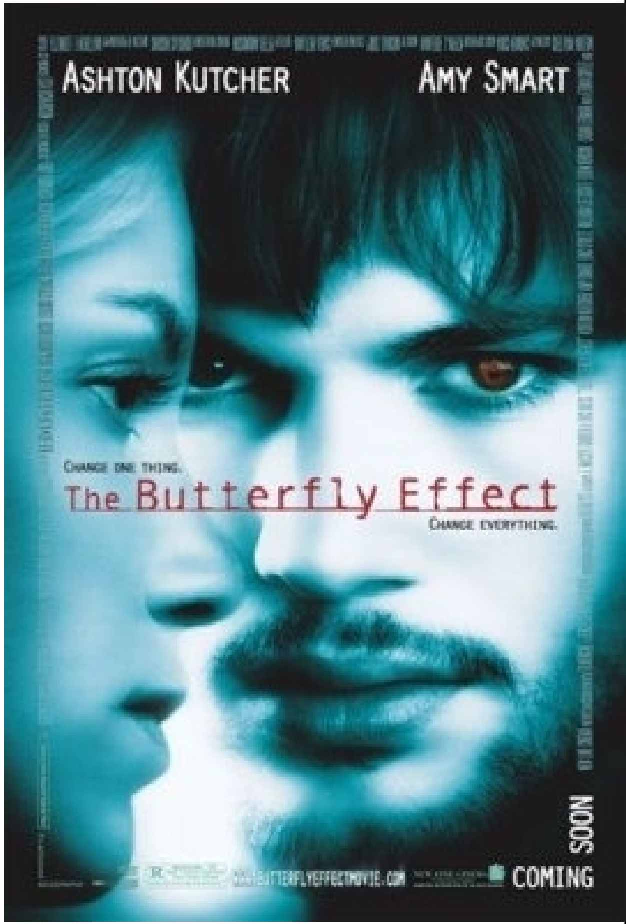
Queries and Commands have different requirements:

- ❖ Queries – eventual consistency, no need for transactions (idempotent), caching is essential, reporting DB de-normalization, often report aggregate data, Naked Objects (Material Views)
- ❖ Commands – often transactional, eventual consistency may be not ok, normalized DB, usually manage single entities

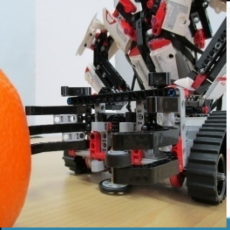
# Imperative and Reactive

## We live in a Connected Universe

The title refers to the butterfly effect, a popular hypothetical example of chaos theory which illustrates how small initial differences may activate chains of events leading to large and often unforeseen consequences in the future...




Source: https://en.wikipedia.org/wiki/The_Butterfly_Effect#/media/File:Butterflyeffect_poster.jpg
Fair use, File:Butterflyeffect poster.jpg, Uploaded by Yaminator, Uploaded: 8 July 2008



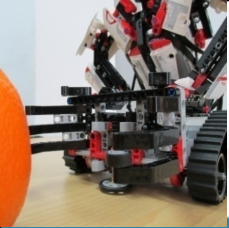


# Imperative and Reactive

## We live in a Connected Universe

... there is hypothesis that all the things in the Universe are intimately connected, and you can not change a bit without changing all.

**Action – Reaction principle is the essence of how Universe behaves.**





# Imperative and Reactive

- Reactive Programming [Wikipedia]: a programming paradigm oriented around data flows and the propagation of change. This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow. Ex: a := b + c

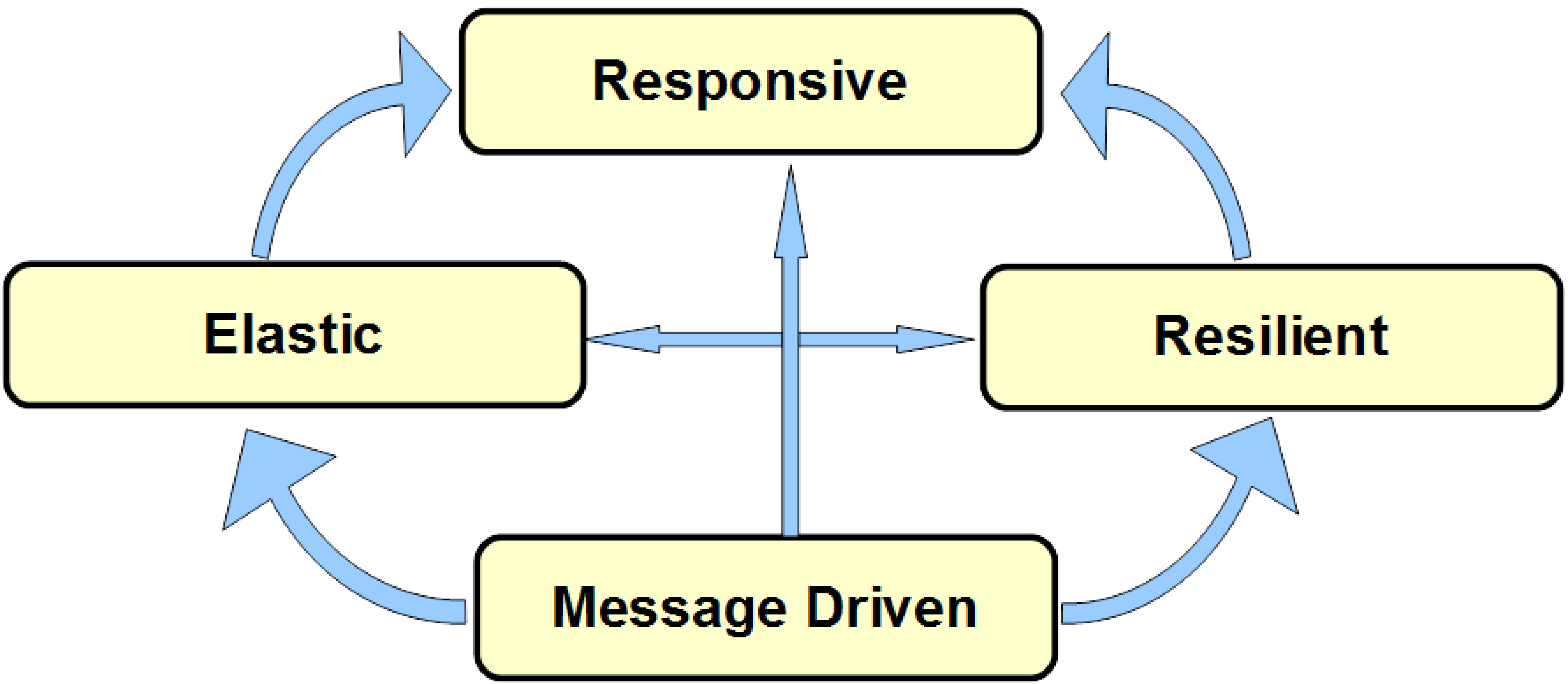
Functional Programming [Wikipedia]: a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm. Eliminating side effects can make it much easier to understand and predict the program behavior. Ex: books.stream().filter(book -> book.getYear() > 2010 ). forEach( System.out::println )

# Functional Reactive (FRP)

*According to **Connal Elliot's answer in Stack Overflow (ground-breaking paper @ Conference on Functional Programming, 1997):***
I'm glad you're starting by asking about a specification rather than implementation first. There are a lot of ideas floating around about what *FRP is. For me it's always been two things:* *(a)* ***denotative*** **and** ***(b) temporally continuous*****.** *Many folks drop both of these properties and identify FRP with various implementation notions, all of which are beside the point in my perspective.*
***" Functional Reactive Programming (FRP) = Denotative, Continuous-Time Programming (DCTP) "***

# Reactive Manifesto

[http://www.reactivemanifesto.org]



# Reactive Programming

- Microsoft® opens source polyglot project ReactiveX (Reactive Extensions) [http://reactivex.io]:

**Rx = Observables + LINQ + Schedulers :)**

Java: RxJava, JavaScript: RxJS, C#: Rx.NET, C#(Unity): UniRx, Scala: RxScala, Clojure: RxClojure, C++: RxCpp, Ruby: Rx.rb, Python: RxPY, Groovy: RxGroovy, JRuby: RxJRuby, Kotlin: RxKotlin, Swift: RxSwift

- Reactive Streams Specification [http://www.reactive-streams.org/] used by
- (Spring) Project Reactor [http://projectreactor.io/, https://github.com/reactor/reactor]

RoboLearn
Program Your Robot

# Reactive Streams Spec.

- Reactive Streams – provides standard for asynchronous stream processing with non-blocking back pressure. This encompasses efforts aimed at runtime environments (JVM & JavaScript) as well as network protocols.
- Minimal set of interfaces, methods and protocols for asynchronous data streams
- As of April 30, 2015 have been released version 1.0.0 of **Reactive Streams for the JVM, including Java API, a textual Specification, a TCK and implementation examples.**

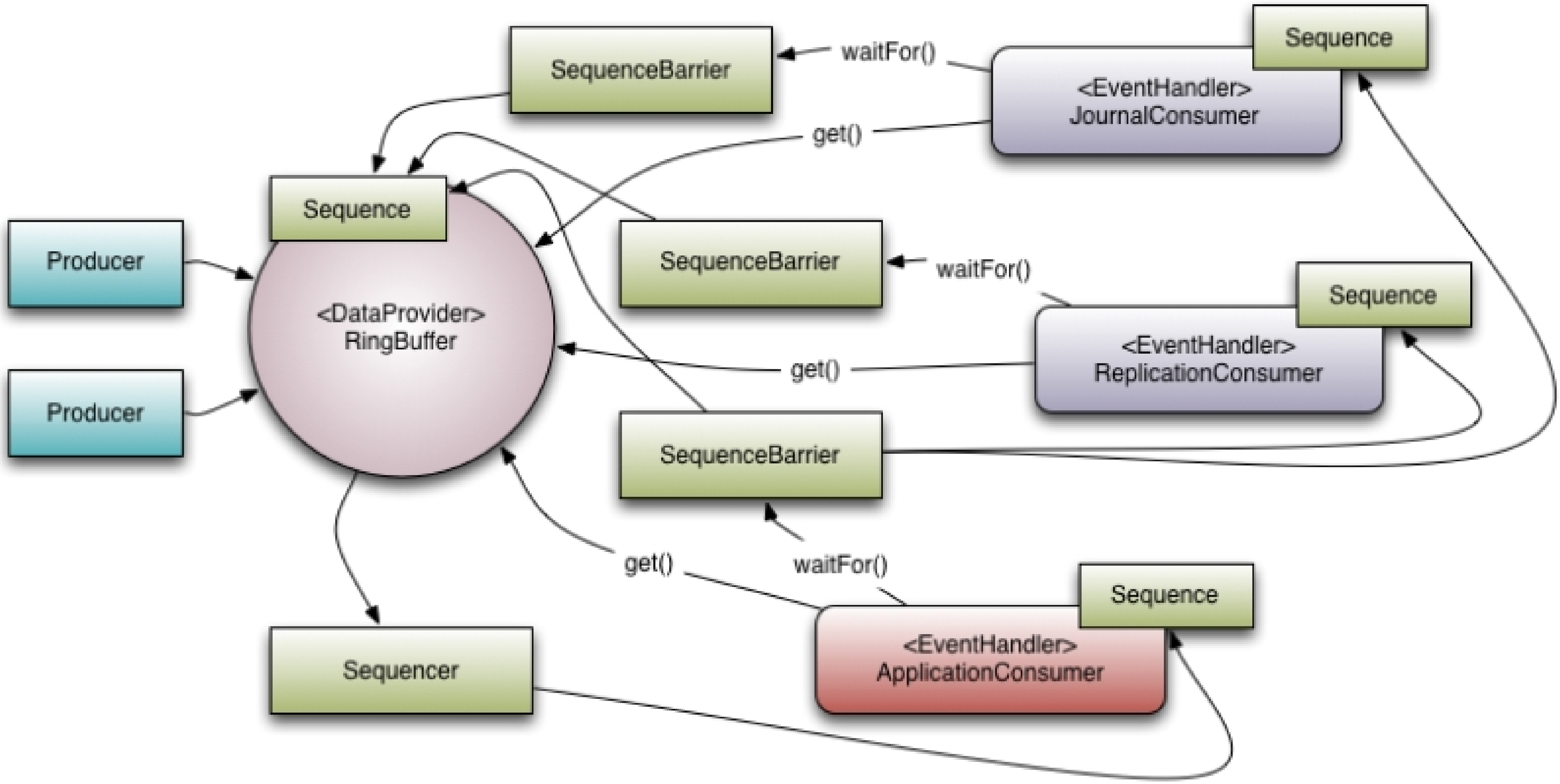
# Reactive Streams Spec.

- ❖ **Publisher** – provider of potentially unbounded number of sequenced elements, according to Subscriber(s) demand. After invoking Publisher.subscribe(Subscriber). Subscriber methods protocol is: **onSubscribe onNext* (onError | onComplete)?**
- ❖ **Subscriber** – receives call to onSubscribe(Subscription) once after passing an instance to Publisher.subscribe(Subscriber). No further notifications until Subscription.request(long) is called.
- ❖ **Subscription** – represents one-to-one lifecycle of a Subscriber subscribing to a Publisher. It is used to both signal desire for data and cancel demand (allow resource cleanup).
- ❖ **Processor** -represents a processing stage, which is both a Subscriber and Publisher and obeys the contracts of both.

# RP = Async Data Streams

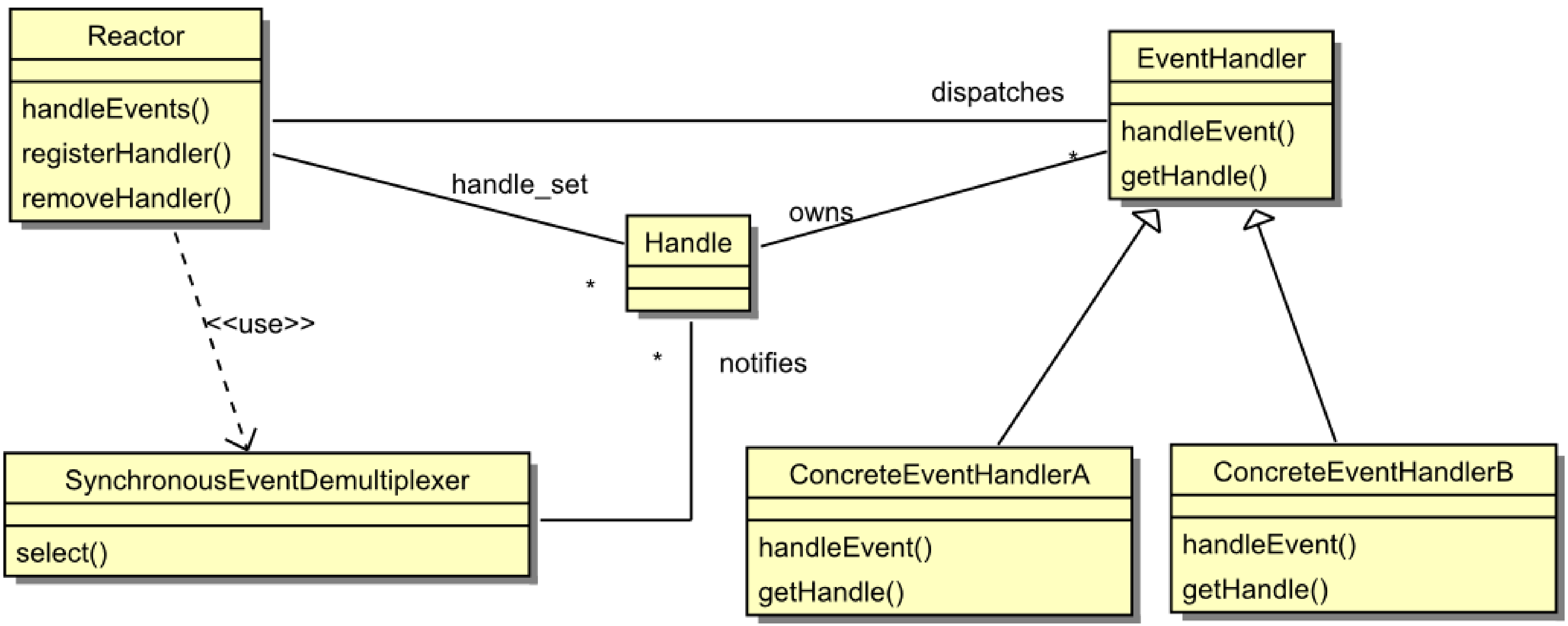
❖ **Functional Reactive Programming (FRP)** [Wikipedia]: asynchronous dataflow programming using the building blocks of functional programming (e.g. map, reduce, filter). FRP has been used for programming graphical user interfaces (GUIs), robotics, and music, aiming to simplify these problems by explicitly modeling time. Example (RxJava):

```
Observable.from(new String[]{"Reactive",
"Extensions", "Java"})
    .take(2).map(s -> s + " : on " + new Date())
    .subscribe(s -> System.out.println(s));
Result:
Reactive : on Wed Jun 17 21:54:02 GMT+02:00 2015
Extensions : on Wed Jun 17 21:54:02 GMT+02:00 2015
```

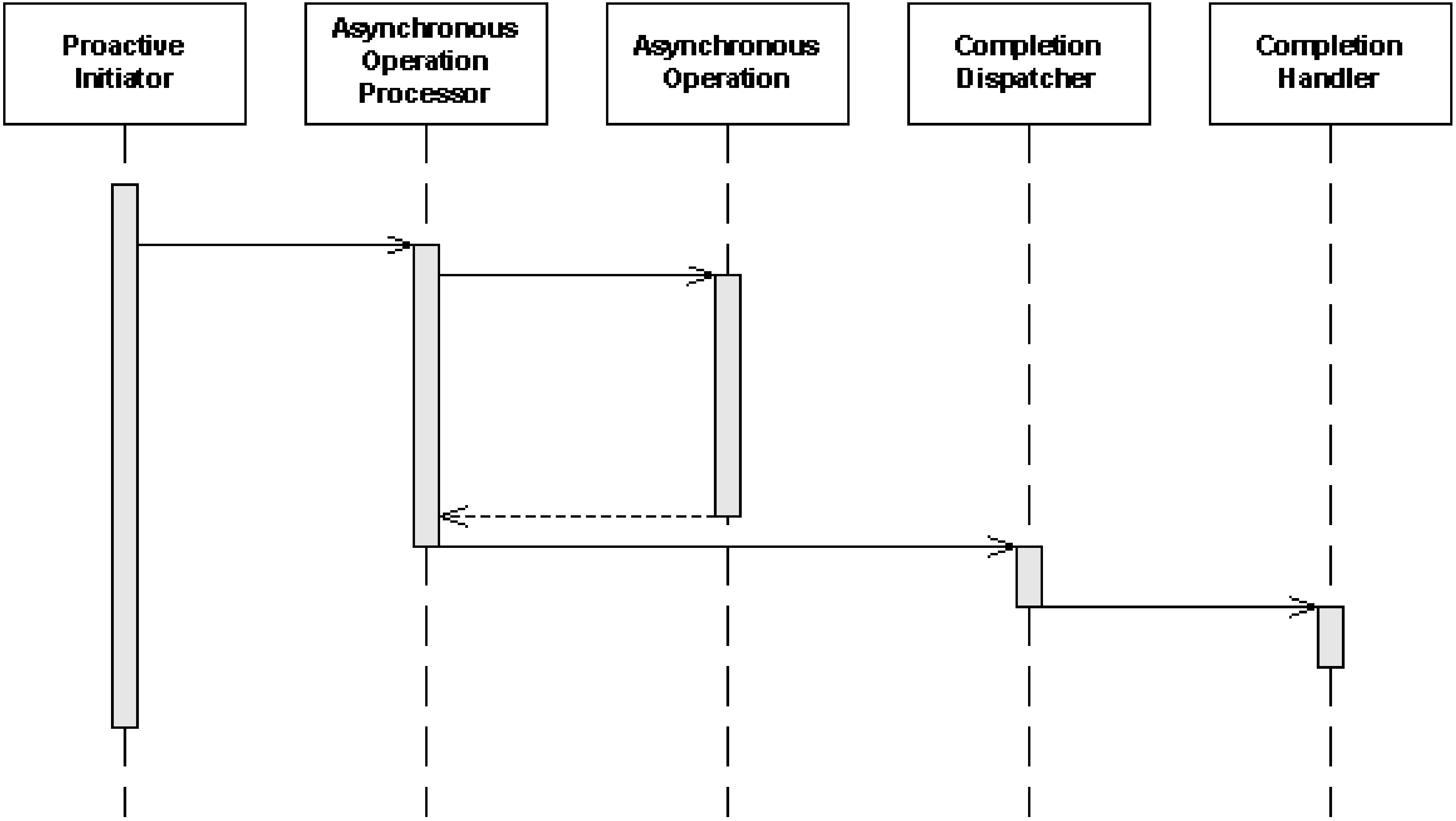



Source: LMAX Disruptor github wiki - https://raw.githubusercontent.com/wiki/LMAX-Exchange/disruptor/images/Models.png
LMAX-Exchange Disruptor License @ GitHub: Apache License Version 2.0, January 2004 - http://www.apache.org/licenses/

# Reactor Design Pattern



# Proactor Design Pattern



RoboLearn
Program Your Robot

# Project Reactor

- Reactor project allows building high-performance (low latency high throughput) non-blocking asynchronous applications on JVM.
- Reactor is designed to be extraordinarily fast and can sustain throughput rates on torder of 10's of millions of operations per second.
- Reactor has powerful API for declaring data transformations and functional composition. Makes use of the concept of Mechanical Sympathy built on top of Disruptor / RingBuffer.

RoboLearn
Program Your Robot

# Project Reactor

- Pre-allocation at startup-time;
- Message-passing structures are bounded;
- Using  Reactive and Event-Driven Architecture patterns => non-blocking end-to-end flows, replies;
- Implement Reactive Streams Specification, to make bounded structures efficient by not requesting more than their capacity;
- Applies above features to IPC and provides non-blocking IO drivers that are flow-control aware;
- Expose a Functional API - organize their code in a side-effect free way, which helps you determine you are thread-safe and fault-tolerant.

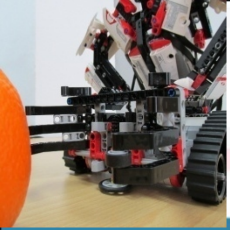
# Reactor: Hello World

```
public class ReactorHelloWorld {
  public static void main(String... args) throws
       InterruptedException {
   Broadcaster<String> sink = Broadcaster.create();
   SchedulerGroup sched = SchedulerGroup.async();
   sink.dispatchOn(sched)
     .map(String::toUpperCase)
     .filter(s -> s.startsWith("HELLO"))
     .consume(s -> System.out.printf("s=%s%n", s));
   sink.onNext("Hello World!");
   sink.onNext("Goodbye World!");
   Thread.sleep(500);
  }
}
```

RoboLearn
Program Your Robot

# Reactor Bus: IPTPI

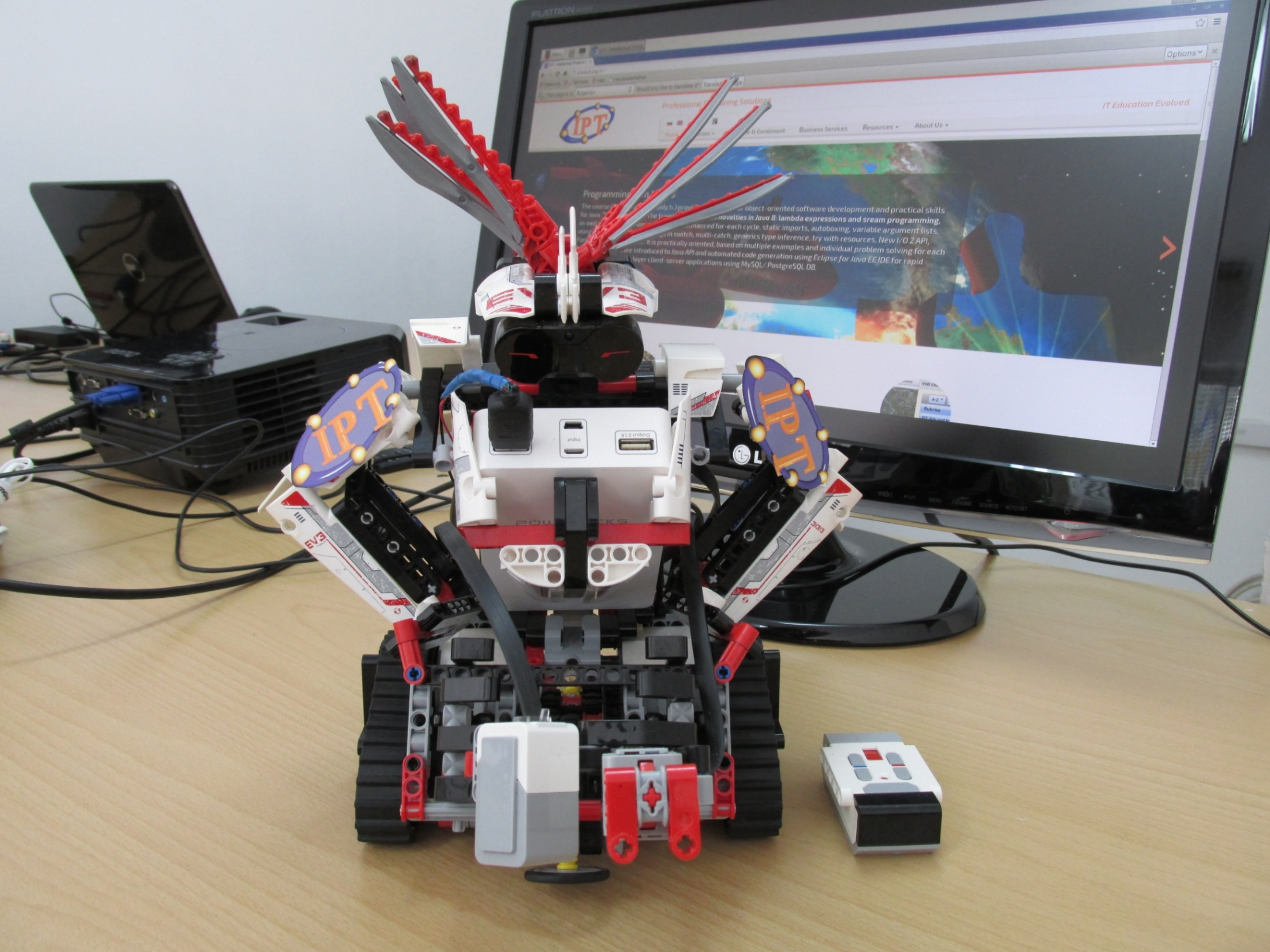
```
EventBusbus = EventBus.create();

bus.on($("iptpi.position"), (Event<String> ev) -> {
  String s = ev.getData();
  System.out.printf("Got %s on thread %s%n", s,
                                Thread.currentThread());
});

bus.notify(outEventTopic,
            Event.wrap(currentPosition));
```



RoboLearn
Program Your Robot

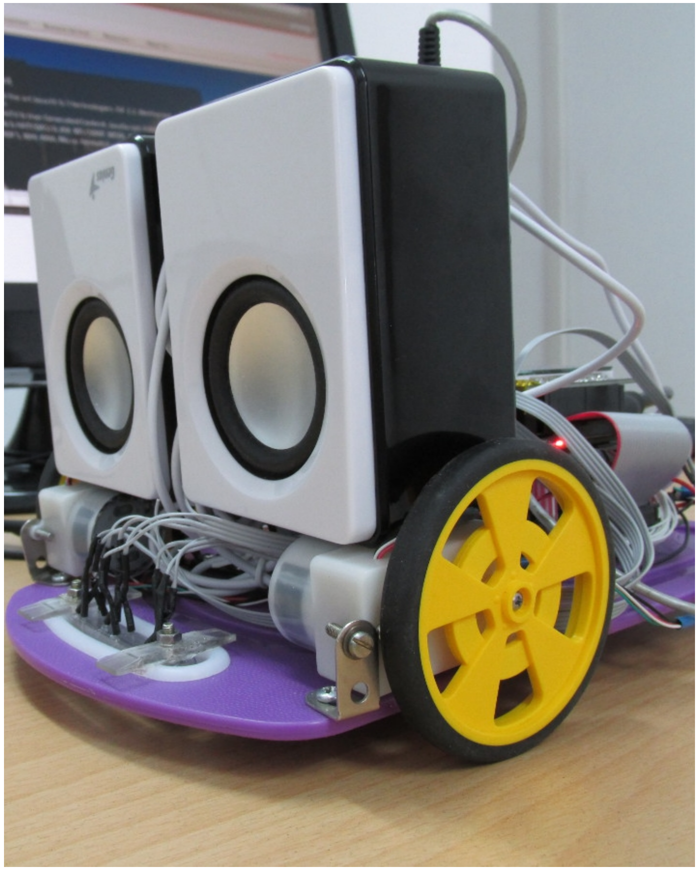
# LeJaRo: Lego® Java Robot

- Modular – *3 motors (with encoders)* – one driving each track, and third for robot clamp.
- Three sensors: *touch sensor* (obstacle avoidance), *light color sensor* (follow line), *IR sensor* (remote).
- LeJaRo is programmed in Java using ***LeJOS*** library.
- More information about LeJaRo: http://robolearn.org/lejaro/
- Programming examples available @GitHub: https://github.com/iproduct/course-social-robotics/tree/master/motors_demo

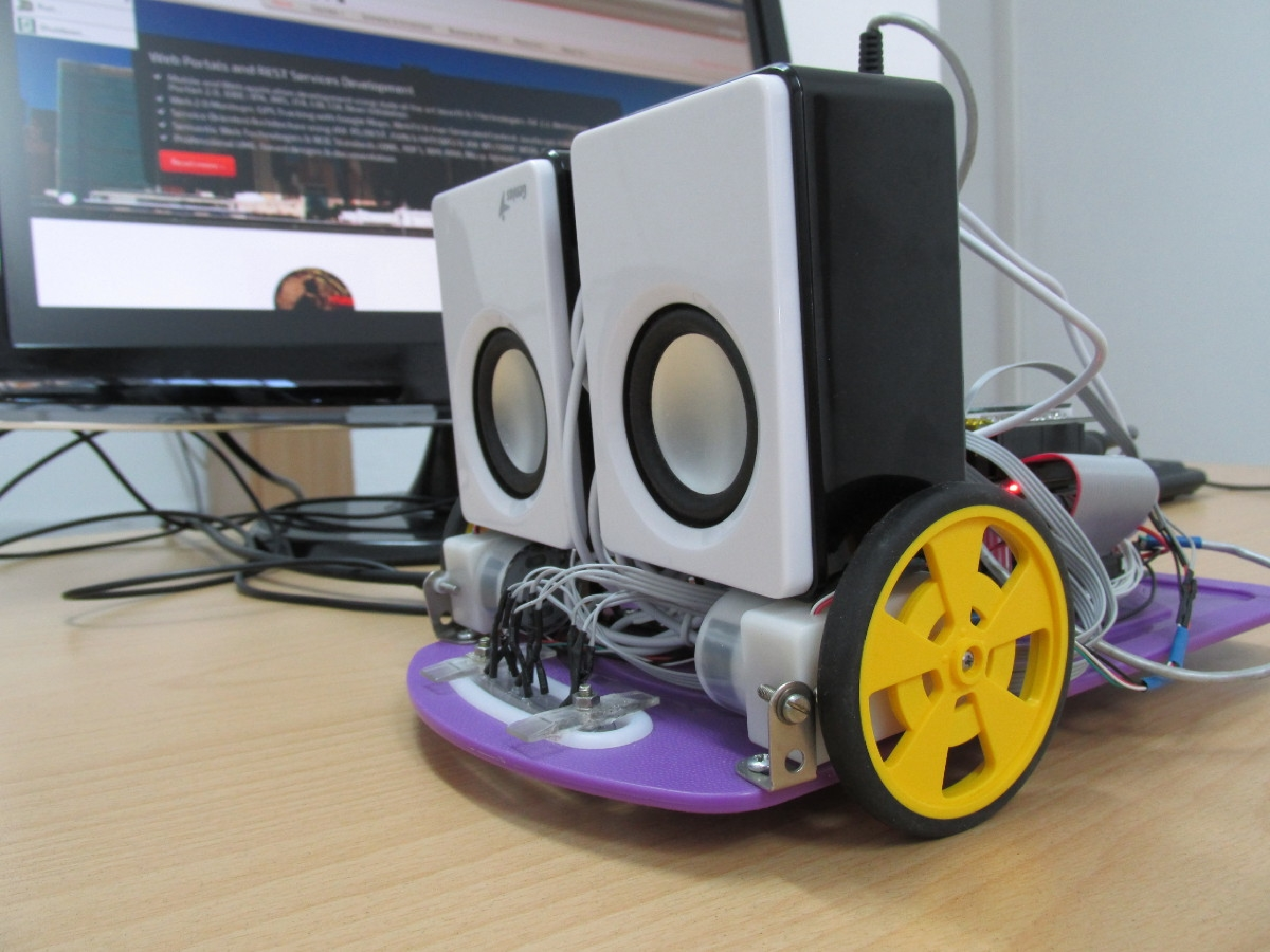

FLATRON
Options
IT Education Evolved
Business Services
Resources
About Us
IPT
IPT
IPT

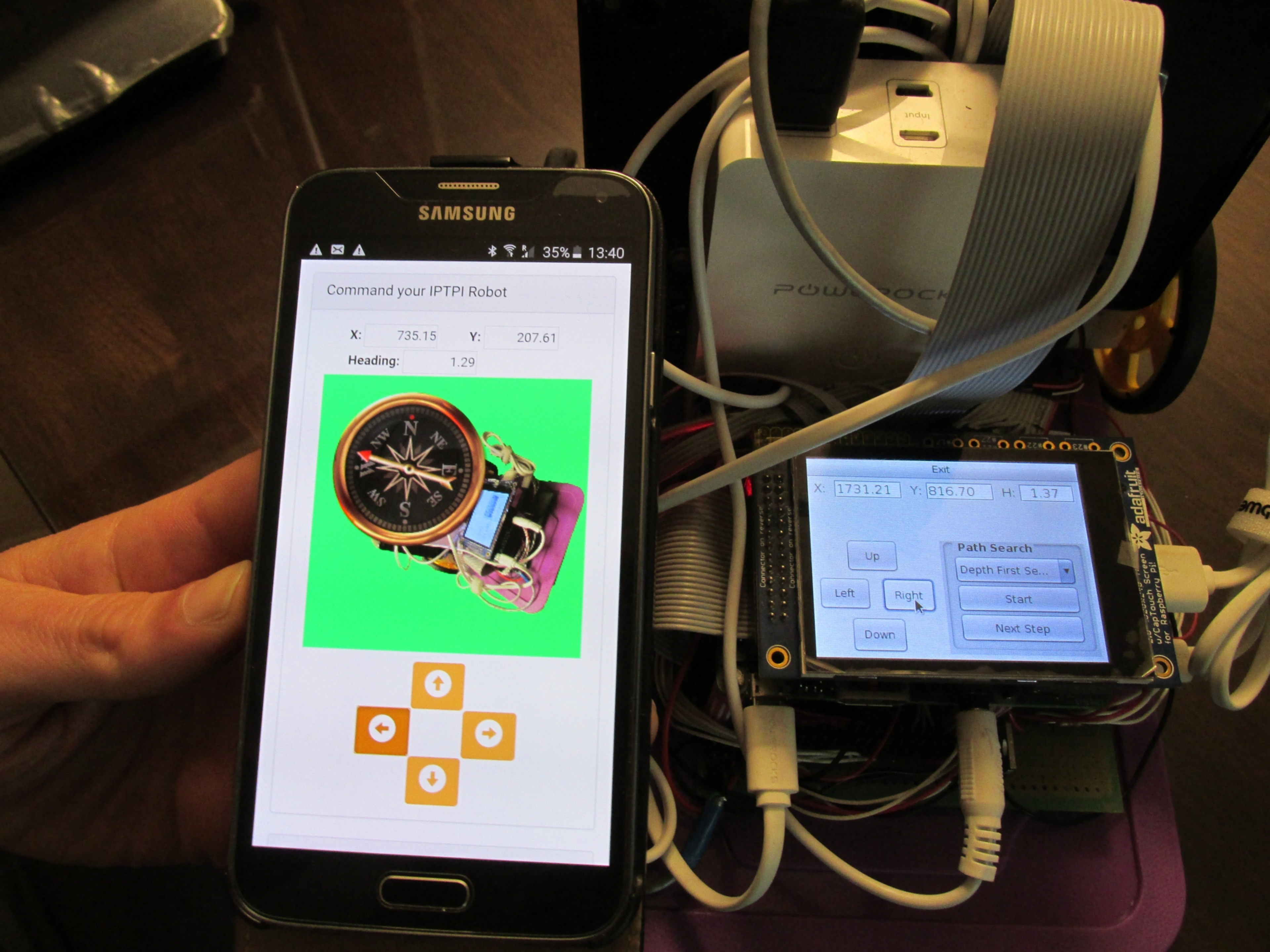
RoboLearn
Program Your Robot

# IPTPI: RPi2 + Ardunio Robot



- Raspberry Pi 2 (quad-core ARMv7 @ 900MHz) + Arduino Leonardo cloneA-Star 32U4 Micro
- *Optical encoders* (custom), IR optical array, 3D accelerometers, gyros, and compass MinIMU-9 v2
- **IPTPI** is programmed in Java using Pi4J, Reactor, RxJava, Akka
- More information about IPTPI: http://robolearn.org/iptpi-robot/

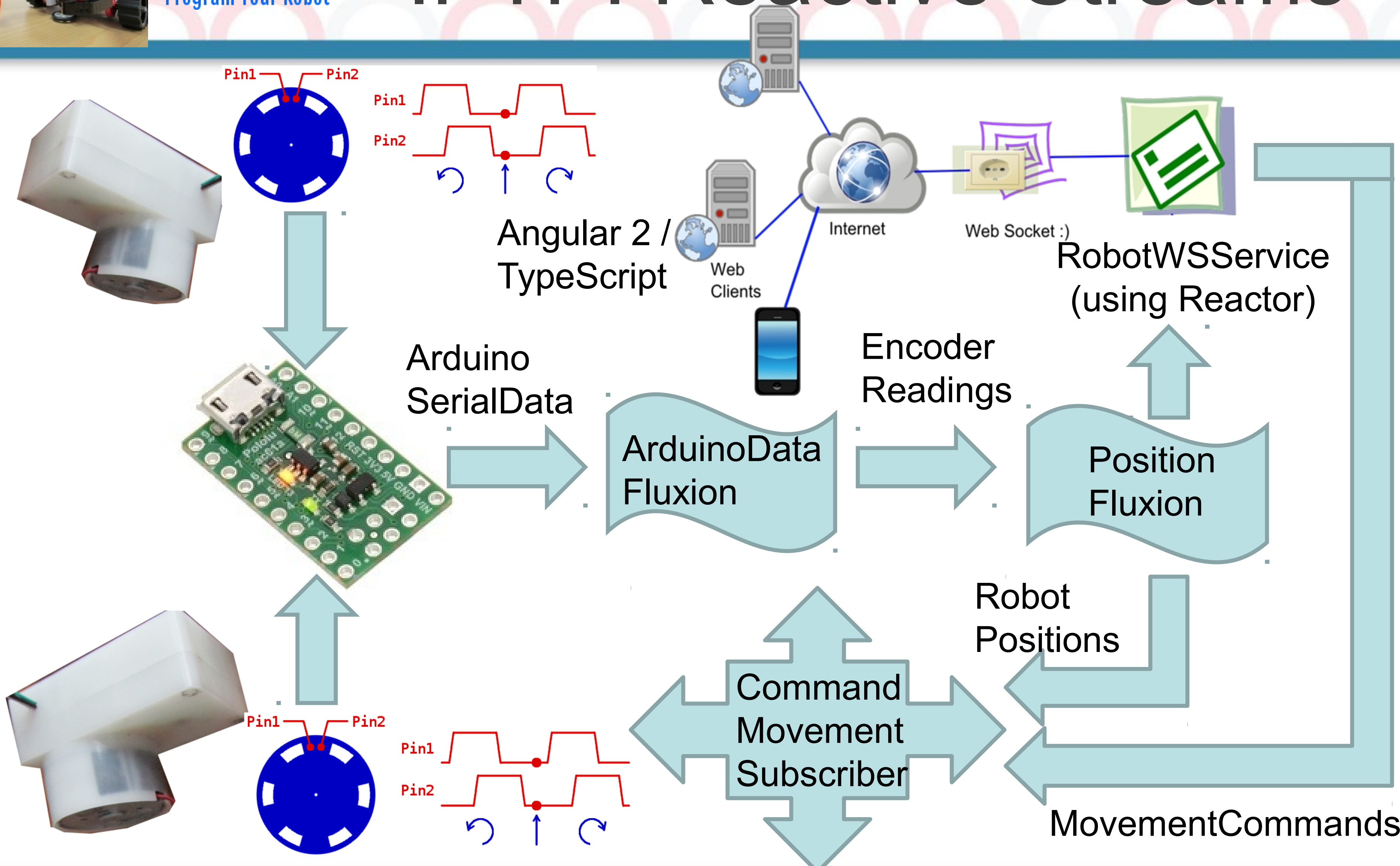

Web Portals and REST Services Development


SAMSUNG
35% 13:40
Command your IPTPI Robot
X: 735.15
Y: 207.61
Heading: 1.29
Exit
X: 1731.21
Y: 816.70
H: 1.37
Up
Left
Right
Down
Path Search
Depth First Se...
Start
Next Step
adafruit
Input

# Lets See Some Code ☺

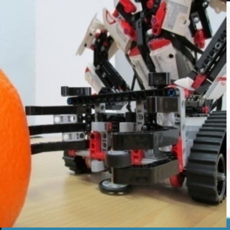
VOXXED Demo code is available @ GitHub:
**https://github.com/iproduct/voxxed-demo**

# IPTPI Reactive Streams



# IPTPI: ArduinoDataFluxion I

```
fluxion = Broadcaster.create();
emitter = fluxion.startEmitter();
final Serial serial = SerialFactory.createInstance();
serial.addListener(new SerialDataEventListener() {
    private ByteBuffer buffer = ByteBuffer.allocate(1024);
    @Override
    public void dataReceived(SerialDataEvent event) {
        try {
            ByteBuffer newBuffer = event.getByteBuffer();
            buffer.put(newBuffer);
            buffer.flip();
            ...
            buffer.get();
            long timestamp = buffer.getInt(); //get timestamp
            int encoderL = -buffer.getInt(); //motors mirrored
            int encoderR = buffer.getInt();
```



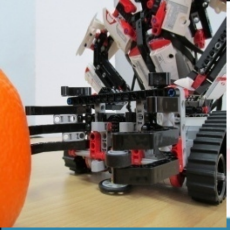
RoboLearn
Program Your Robot

# IPTPI: ArduinoDataFluxion II

```
            EncoderReadings readings =
              new EncoderReadings(encoderR, encoderL, timestamp);
            emitter.submit(readings);
            ...
            buffer.compact();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
try {
    serial.open(PORT, 38400);
} catch(SerialPortException | IOException ex) {
    System.out.println("SERIAL SETUP FAILED:"+ex.getMessage());
}
```

# IPTPI: PositionFluxion I

```
Fluxion<EncoderReadings> skip1 =
                          encoderReadings.skip(1);
fluxion = Fluxion.zip(encoderReadings, skip1)
   .scan(new Position(0, 0, 0), (last, tupple) -> {
       EncoderReadings prev = tupple.getT1();
       EncoderReadings curr = tupple.getT2();
       int prevL = prev.getEncoderL();
       int prevR = prev.getEncoderR();
       int currL = curr.getEncoderL();
       int currR = curr.getEncoderR();
       double alpha0 = last.getHeading();
       ... // actual position/heading calculations
       return new Position((float)x, (float)y,
          alpha, curr.getTimestamp());
   });
```





# CommandMovementSubscriber I

```
public class CommandMovementSubscriber extends
                  ConsumerSubscriber<Command<Movement>>  {
   private PositionFluxion positions;
   public CommandMovementSubscriber(PositionFluxion positions){
   this.positions = positions;
      Gpio.wiringPiSetupGpio(); // initialize wiringPi library
      Gpio.pinMode(5, Gpio.OUTPUT); // Motor direction pins
      Gpio.pinMode(6, Gpio.OUTPUT);
      Gpio.pinMode(12, Gpio.PWM_OUTPUT); // Motor speed pins
      Gpio.pinMode(13, Gpio.PWM_OUTPUT);
      Gpio.pwmSetMode(Gpio.PWM_MODE_MS);
      Gpio.pwmSetRange(MAX_SPEED);
      Gpio.pwmSetClock(CLOCK_DIVISOR);
   }
   @Override
   public void doNext(Command<Movement> command) { ... }
}
```

# CommandMovementSubscriber II

```
private void runMotors(MotorsCommand mc) {
    //setting motor directions
    Gpio.digitalWrite(5, mc.getDirR() > 0 ? 1 : 0);
    Gpio.digitalWrite(6, mc.getDirL() > 0 ? 1 : 0);
    //setting speed
    if(mc.getVelocityR()>=0 && mc.getVelocityR() <=MAX_SPEED)
        Gpio.pwmWrite(12, mc.getVelocityR()); // set speed
    if(mc.getVelocityL()>=0 && mc.getVelocityL() <=MAX_SPEED)
        Gpio.pwmWrite(13, mc.getVelocityL());
  }
}
```

# IPTPI: RobotWSService I

```java
private void setupServer() throws InterruptedException {
   httpServer = NetStreams.<Buffer, Buffer>httpServer(
      HttpServerSpec<Buffer,Buffer> serverSpec ->
          serverSpec.listen("172.22.0.68", 80)
   );
   httpServer.get("/", getStaticResourceHandler());
   httpServer.get("/index.html", getStaticResourceHandler());
   httpServer.get("/app/**", getStaticResourceHandler());
   ...
   httpServer.ws("/ws", getWsHandler());

   httpServer.start().subscribe(
      Subscribers.consumer(System.out::println));
}
```

# IPTPI: RobotWSService II

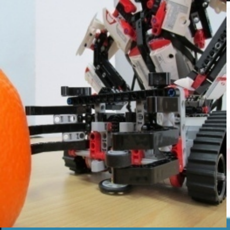
```
private ReactorHttpHandler<Buffer, Buffer> getWsHandler() {
  return channel -> {
    System.out.println("Connected a websocket client: " +
                                        channel.remoteAddress());
    channel.map(Buffer::asString).consume(
      json -> {
        System.out.printf(“WS Message: %s%n“, json);
        Movement movement = gson.fromJson(json, Movement.class);
        movementCommands.onNext(new Command<>("move", movement));
      });

    return positions.flatMap(position ->
      channel.writeWith(
      Flux.just(Buffer.wrap(gson.toJson(position)))
    ));
  };
}
```

# Additional Resources

IPT Reactive Java/JS/Typescript and Angular 2 courses: http://iproduct.org

More information about robots @RoboLearn: http://robolearn.org/

Lots of Java robotics and IoT resources @Social Robotics Course GitHub Wiki: https://github.com/iproduct/course-social-robotics/wiki/Lectures



RoboLearn

Program Your Robot

# Thank's for Your Attention!

**Trayan Iliev**

**CTO of IPT – Intellectual Products & Technologies**

**http://iproduct.org/**

**http://robolearn.org/**

**https://github.com/iproduct**

**https://twitter.com/trayaniliev**

**https://www.facebook.com/trayan.iliev**

**https://plus.google.com/+IproductOrg**