

# RHEA

A REACTIVE, HETEROGENEOUS, EXTENSIBLE, AND ABSTRACT  
FRAMEWORK FOR DATAFLOW PROGRAMMING

---

Orestis Melkonian, Angelos Charalambidis

October 29, 2018

Utrecht University, National Center for Scientific Research "Demokritos"

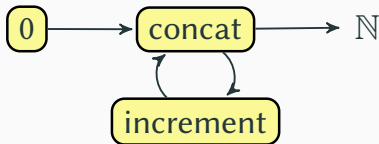


Everything  
flows...



# DATAFLOW COMPUTATIONAL MODEL

- Completely decentralized
  - Independent nodes communicating with each other
- Demand-driven
- No control-flow
- Implicit concurrency

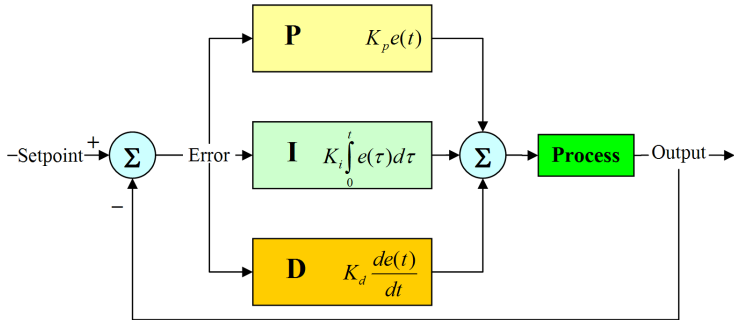


# MOTIVATION

- Sensor-based systems
  - Robotics (ROS)
  - IoT (MQTT)
- Big Data
  - Apache Flink
  - Map-Reduce
  - RX framework
- Interactive Systems
  - UIs (ReactJS)
  - Games (Yampa)
- Neural Networks (TensorFlow)

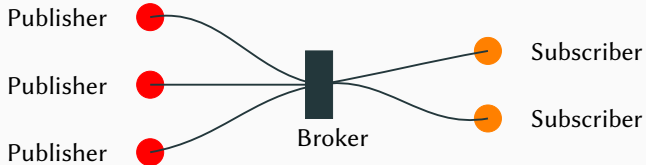
# MOTIVATION: ROBOTICS

- Robot Perception Architecture (RPA)
- Many dataflow examples in control theory



# ROBOT OPERATING SYSTEM (ROS)

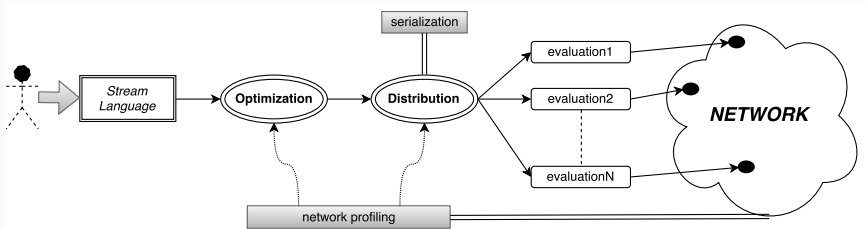
- Most popular middleware for robotic applications
- Provides a Publish-Subscribe messaging platform



# ROS CODE

```
bool scanReceived = FALSE, imageReceived = TRUE;
LaserScan scan; Image image;
subscribe<LaserScan>("/scan", scanCallback);
subscribe<Image>("/camera/rgb", imageCallback);
// Main ROS loop
while (ros::ok()) {
    if (scanReceived && imageReceived) {
        window.show(embedLaser(scan, image));
        scanReceived = FALSE; imageReceived = FALSE;
    }
    ros::spinOnce();
}
// Callback for topic "/scan"
void scanCallback(LaserScan newScan) {
    if (!scanReceived) {
        scan = newScan;
        scanReceived = TRUE;
    }
}
// Callback for topic "/camera/rgb"
void imageCallback(Image newImage) {
    if (!imageReceived) {
        image = new Image(newImage);
        imageReceived = TRUE;
    }
}
// OpenCV stuff...
Mat embedLaser(LaserScan scan, Image image) { ... }
```





- Dataflow framework for the JVM
- Current frontends only in Java & Scala
- Set of libraries in [github.com/rhea-flow](https://github.com/rhea-flow)

- **rhea-core**
- graph-viz
- rx-eval
- ros-eval
- mqtt-eval
- hazelcast-distribution
- scala-wrapper
- rhea-music
- example applications. . .

**REACTIVE**

**HETEROGENEOUS**

**EXTENSIBLE**

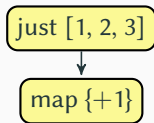
**ABSTRACT ✓**

## STREAM LANGUAGE: SOURCES AND SINGLE-INPUT NODES

---

```
Stream.just(1, 2, 3)  
      .map(x -> x + 1);
```

---

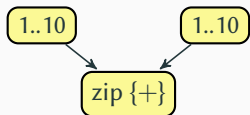


## STREAM LANGUAGE: MULTIPLE-INPUT NODES

---

```
Stream.zip(  
  Stream.range(1, 10),  
  Stream.range(1, 10),  
  (x, y) -> x + y  
);
```

---

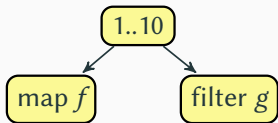


## STREAM LANGUAGE: SPLIT

---

```
Stream<Int> st =  
    Stream.range(1, 10);  
st.map(f)  
st.filter(g)
```

---

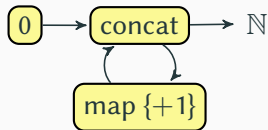


## STREAM LANGUAGE: CYCLE

---

```
Stream  
.just(0)  
.loop(s -> s.map(i -> i + 1));
```

---



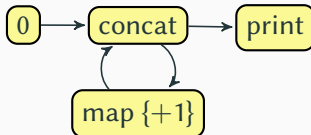
## STREAM LANGUAGE: ACTIONS

---

Stream

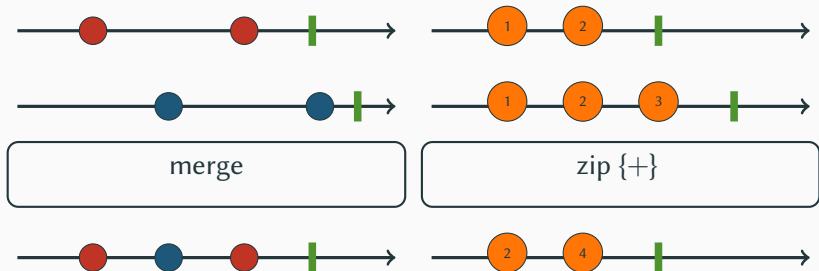
```
.just(0)  
.loop(s -> s.map(i -> i + 1))  
.subscribe(System::println);
```

---

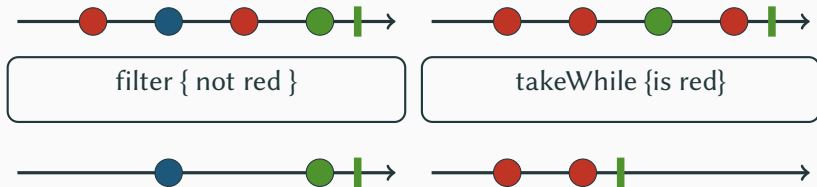




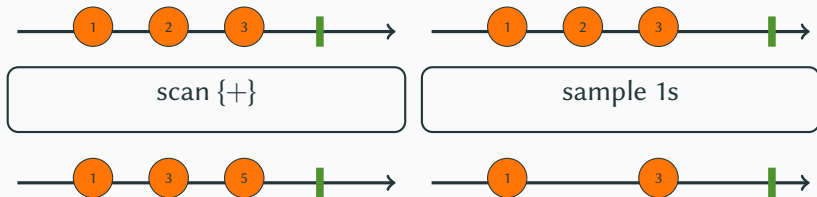
# STREAM LANGUAGE: MARBLE DIAGRAMS



## STREAM LANGUAGE: MARBLE DIAGRAMS



# STREAM LANGUAGE: MARBLE DIAGRAMS

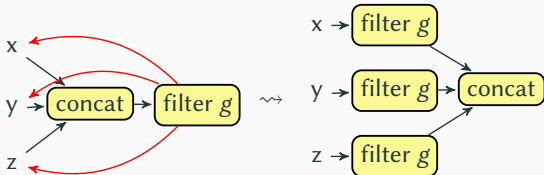


Series of semantics-preserving graph transformations

- Proactive filtering
- Granularity adjustment

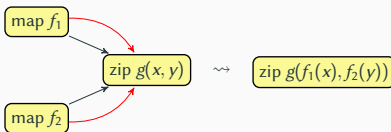
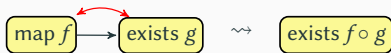
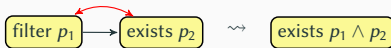
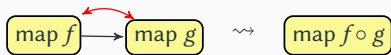
# OPTIMIZATIONS: PROACTIVE FILTERING

Transfer as few elements as possible

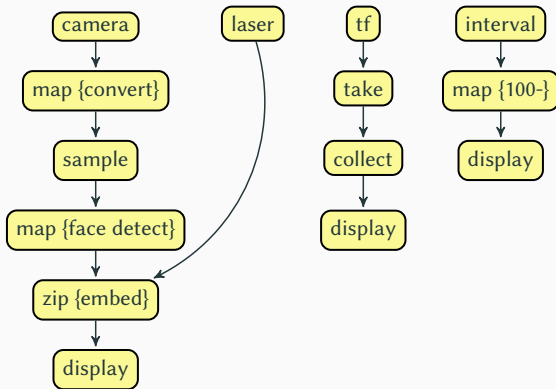


# OPTIMIZATIONS: GRANULARITY ADJUSTMENT

## Merge nodes



## APPLICATION: ROBOT PANEL



# ROBOT PANEL DEMO



**REACTIVE ✓**

**HETEROGENEOUS**

**EXTENSIBLE**

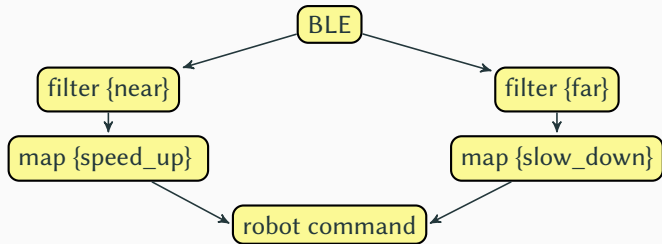
**ABSTRACT ✓**

## APPLICATION: ROBOT HOSPITAL GUIDE

---

1. Robot guides patients to parts of the hospital
2. Patient holds a smartphone that broadcasts bluetooth signals
3. Robot adjusts its speed, according to distance

## APPLICATION: ROBOT HOSPITAL GUIDE



## APPLICATION: ROBOT HOSPITAL GUIDE

```
// RHEA setup
Stream.configure(new HazelcastDistributionStrategy(
    RxjavaEvaluationStrategy::new,
    RosEvaluationStrategy::new,
    MqttEvaluationStrategy::new));

// Topics
Topic<RobotCommand> vel = new RosTopic<>("/robot/cmd");
Topic<Proximity> ble = new MqttTopic<>("/ble");
// Running on smartphone
Stream.from(ReactiveBeacons.observe())
    .map(Beacon::getProximity)
    .subscribe(ble);
// Running on robot
Stream<Proximity> prox = Stream.from(ble);
prox.filter(Proximity::isNear)
    .map(d -> Commands.SPEED_UP)
    .subscribe(vel);
prox.filter(Proximity::isFar)
    .map(d -> Commands.SLOW_DOWN)
    .subscribe(vel);
```

- Declarative glue code
- Multiple heterogeneous devices/streams
- Dataflow in the large, whatever in the small

## DISTRIBUTION: TASK FUSION

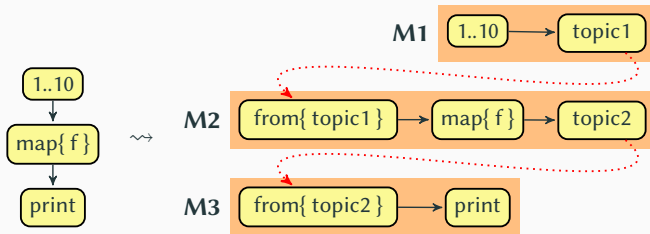
1. If desired granularity not reached, perform task fusion:



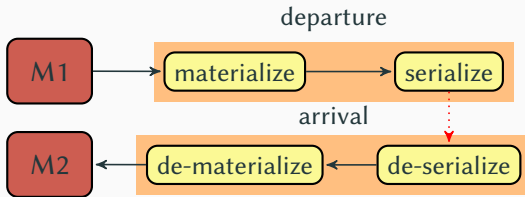
## DISTRIBUTION: NODE PLACEMENT

2. Place nodes in the available machines, in order to:

- minimize communication overhead
- satisfy hard constraints (e.g. ROS not available on raspberry)



3. Streams can terminate either with Complete or Error.
- Necessary to materialize them when transferring





**REACTIVE ✓**

**HETEROGENEOUS ✓**

**EXTENSIBLE**

**ABSTRACT ✓**



- Music datatypes defined in [rhea-flow/rhea-music](#)
  - e.g. Tone, Pitch, Scale, etc...
- Scala's implicit conversions for music-specific operators
  - working only on type `Stream<Music>`

# CHAOTIC FUNCTIONS

- Chaotic functions (1 variable)

$$x = 1 - \alpha * x^2$$

- Complex functions (multiple variables)

$$x = y - \alpha * x^2$$

$$y = \beta * x$$

- Hence, these functions have type

$$f :: (\alpha, \beta) \rightarrow ([Double] \rightarrow [Double])$$

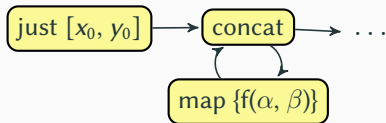
- Slight change of initial values leads to drastic change in output

# CHAOTIC FUNCTIONS AS DATAFLOWS

---

```
Stream  
.just(0, 0)  
.loop(s -> s.map(f(1.4, .3)))
```

---



# CHAOS MUSIC DEMO

**REACTIVE ✓**

**HETEROGENEOUS ✓**

**EXTENSIBLE ✓**

**ABSTRACT ✓**

# LIMITATIONS

- Difficult to extend the available operators
- The surface syntax is not strict enough
  - Only single-assignment in `Stream` variables
  - Specific program structure (configuration  $\rightarrow$  dataflow)
  - Only pure functions as arguments to higher-order operators



- More sophisticated optimizations
- Reinforcement learning for node placement
- Dynamic reconfiguration (hot-swapping code)
- Erlang-style error handling
- Machine-learning backend
- ...

- A dataflow DSL for wireless system programming
- Compiles to highly optimal vectorized C code
- In contrast to SORA (low-level C++ library):
  - Wifi Receiver: 3k vs 23k LOC
  - Scrambler: 20 vs 90 LOC
  - Same performance!

# CONCLUSION

- Some domains are still full of low-level techniques
- The FP paradigm can overcome this quite nicely
- Higher, higher, higher!

**QUESTIONS?**