PerLa
PERvasive LAnguage

PERLA: LANGUAGE AND MIDDLEWARE

F. A. Schreiber, R. Camplani

Politecnico di Milano,
Dipartimento di Elettronica e Informazione,
Milano, Italy

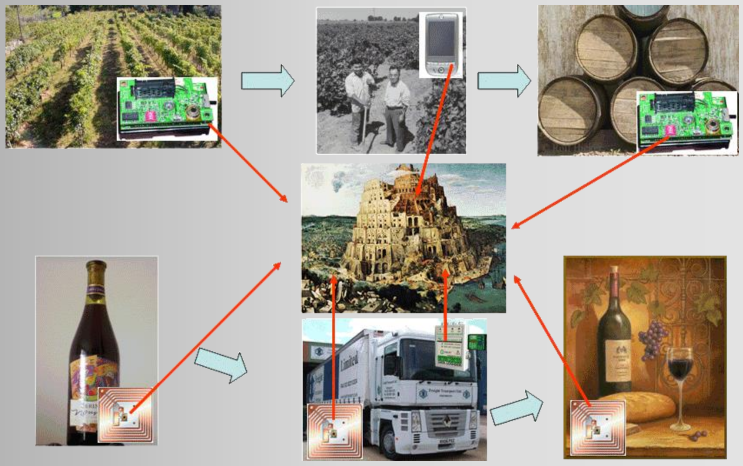
http://perlawsn.sourceforge.net/index.php

OUTLINE

- Introduction
  - Pervasive Systems
  - Open Issues
- State of the art
- Proposed solution: PerLa
  - Perla internals
    - Frontend
    - Middleware
    - Low-Levels
- Real Testbed: Lecco's deployment
- Future works

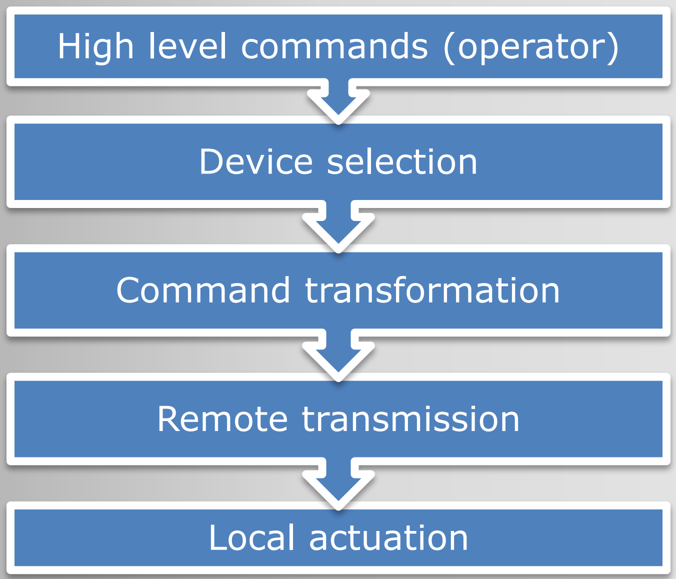
INTRODUCTION: PERVASIVE SYSTEM

- A pervasive system is composed of heterogeneous devices:
  - RFID tags
  - Sensor motes
  - PDA
  - Actuators
- Pervasive systems scenarios



TYPICAL APPLICATION IN PERVASIVE SYSTEM

- Commands life cycle



High level commands (operator)

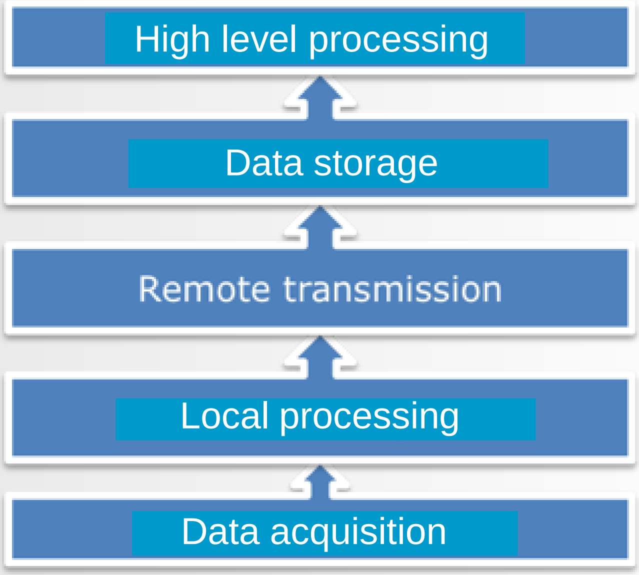
Device selection

Command transformation

Remote transmission

Local actuation

- Data life cycle



High level processing

Data storage

Remote transmission

Local processing

Data acquisition

What about a real deployment?

REAL WORLD APPLICATION OF PERVASIVE SYSTEMS

- First examples[1][2][3][4] are "embedded" systems
  - ONLY support for specific hardware
  - Ad-Hoc transmission
    - Data dependent!
  - Dedicated server application
    - "SQL-in-the-code" paradigm

A more "engineered" approach?

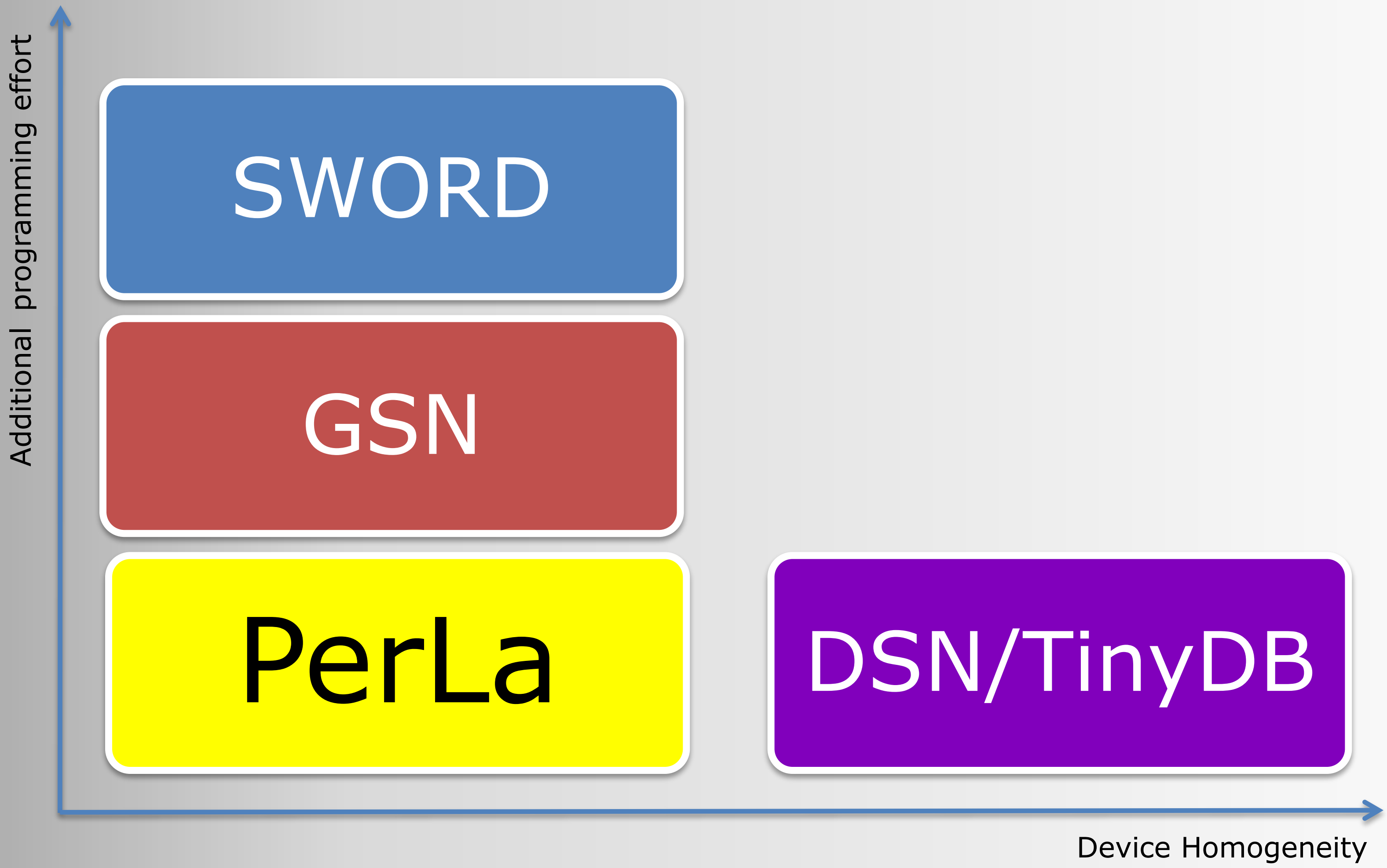
STATE OF THE ART

- There are some projects aiming at identify approaches to manage pervasive systems
  - The key idea:
    - An high level language to define the envisaged pervasive system (data, alarms, etc..)
  - Most important projects
    - TinyDB [5]
    - DSN [6]
    - GSN [7]
    - SIEMENS SWORD [8]

STATE OF THE ART (2)

	TinyDB	GSN	DSN	SWORD
Data gathering	✔	✗	✔	✗
Configurability	– –	✗	✗	✗
Data aggregation	✔	– –	✔	✗
High level integration	✔	✔	✔	✔
Re-Usability	– –	✔	– –	✔
Low Level software support	✔	✗	✔	✗
Heterogeneity supp.	✗	✔	✗	✔

STATE OF THE ART (3)



Additional programming effort

SWORD

GSN

PerLa

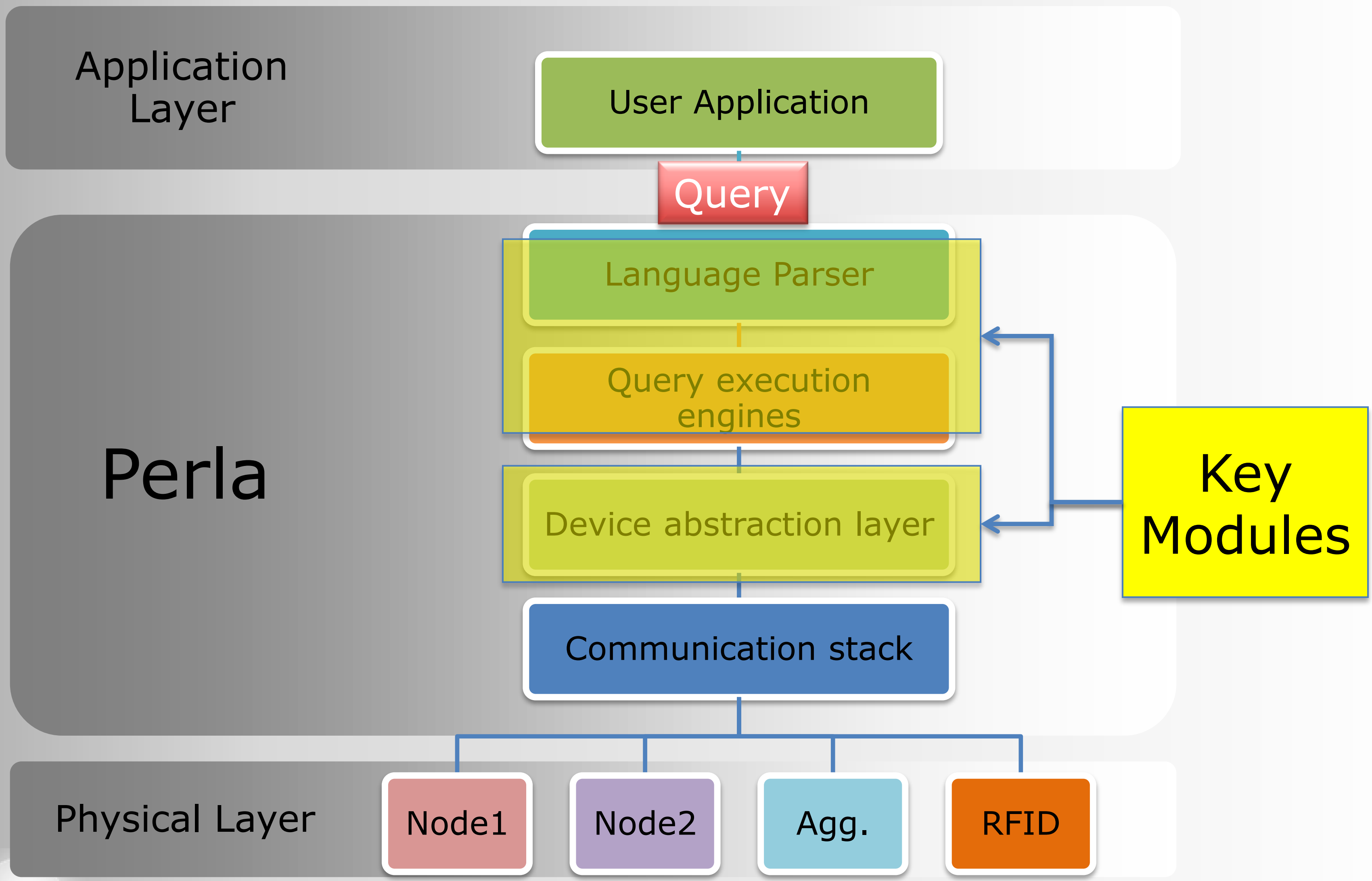
DSN/TinyDB

Device Homogeneity

PERLA: OVERVIEW

- Improvement to the state of the art:
  - Use of the DB abstraction:
    - defines a user friendly language to handle pervasive systems.
      - similar as possible to SQL
      - **DSN** is based on Snlog, not widely known.
  - Heterogeneity
    - deploy-time
    - run-time
    - **TinyDB** and **DSN** only supports a single homogenous network
  - Middleware
    - makes the support for new devices easy
    - reduces the amount of the needed low level code
    - **GSN, SWORD** do not provide low level interfaces for devices
      - TCP/IP+XML-based protocol
      - No support for low level devices firmware

PERLA: OVERVIEW



PERLA: KEY FEATURES

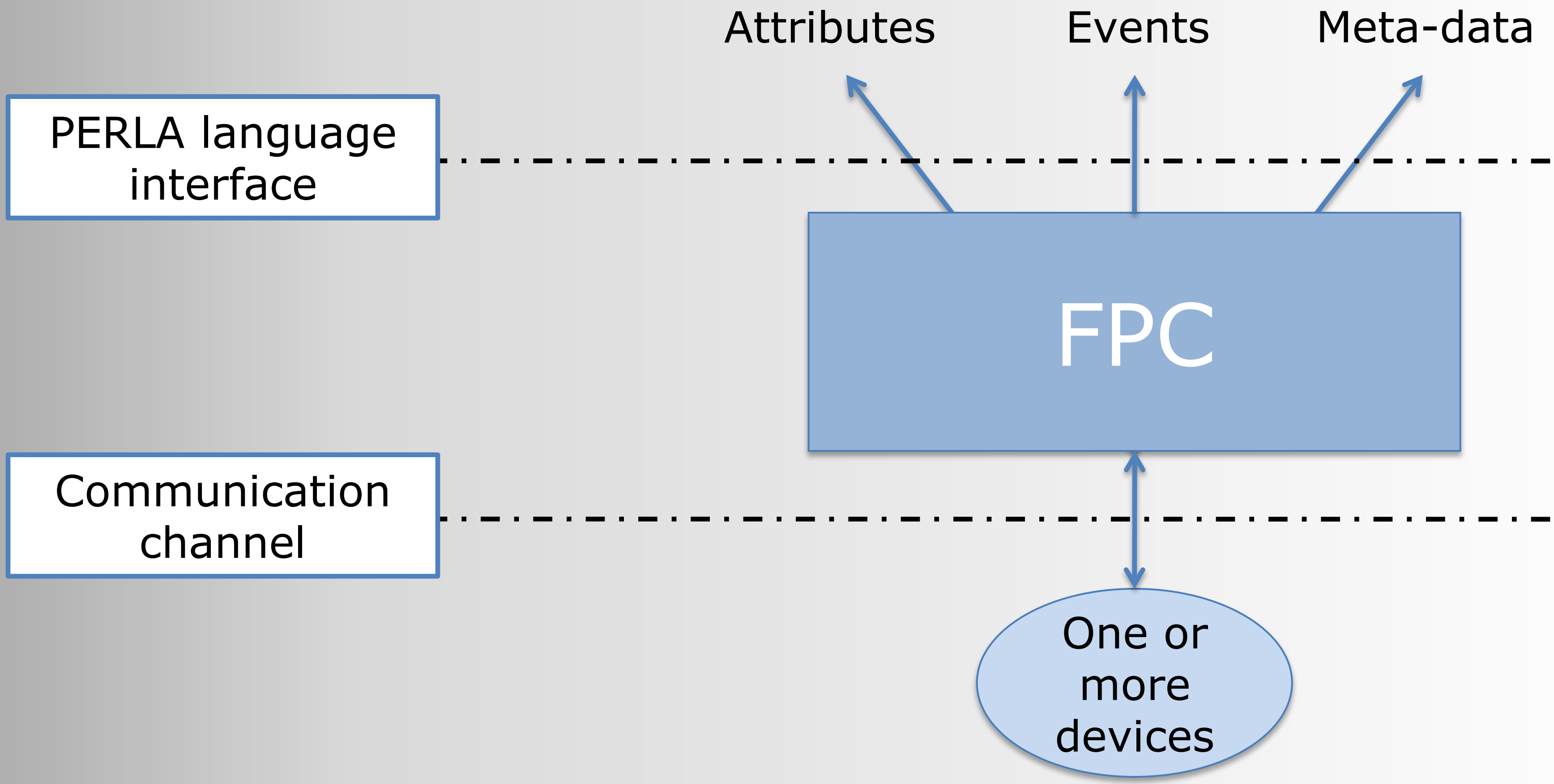
High level interface: the language

- SQL-like syntax
- Three levels of queries
  - High level query (HLQ)
    - Equivalent to SQL for streaming DB
  - Actuation query (AQ)
    - Executes commands, set parameters on devices
  - Low level query (LLQ)
    - Defines the behaviour of a single or of a group of devices

Low level interface: the hardware abstraction layer

- Devices as a *Functionality Proxy Component* (FPC)
- An FPC provides:
  - Attribute reading (*id, temperature, pressure, power level, last sensed RFID reader*, …)
  - Event notification (*last sensed RFID reader changed*, …)
  - Meta-description (*name, data type*, …)

LANGUAGE-FPC INTERFACE



Attributes

Events

Meta-data

PERLA language interface

FPC

Communication channel

One or more devices

THE LANGUAGE: OVERVIEW

- LANGUAGE FEATURES
  - Data representation (FPC abstraction)
  - Physical device management
  - Functional characteristics
    - raw data manipulation
    - provide query results
    - set sampling parameters
  - Non-functional characteristics
    - constraints on the functionality
    - QOS (mainly power management)
    - determine the participation of a node to a query

DATA STRUCTURES

- Two types of data structures
  - STREAM TABLES
    - Unbounded lists of records
    - Queries can perform
      - insert (insertion of a new record)
      - read (extract a data window [ts, size])
  - SNAPSHOT TABLES
    - Set of records produced by a query in a given period t
    - Content refreshed every period t

LOW LEVEL QUERIES

- Define the behaviour of a single or of a group of devices abstracted by an FPC
  - Precise definition of SAMPLING operations
    - read attributes from a device
    - insert values into a temporary buffer (local buffer)
  - Perform simple SQL OPERATIONS (filtering, grouping, …)
    - on data in the local buffer
  - Insert records in the final data structure

LLQ: PHYSICAL DEVICE MANAGEMENT

- Both sampling and data operations management can be executed:
  - Periodically
  - Event based

- Example: RFID abstraction
  - RFID TAG AS A SENSOR
    - sampled data → id of the last reader which sensed the tag
  - READER AS A SENSOR
    - sampled data → id of the last tag sensed by the reader
  - EVENT BASED SAMPLING
    - when the corresponding FPC senses the reader firing

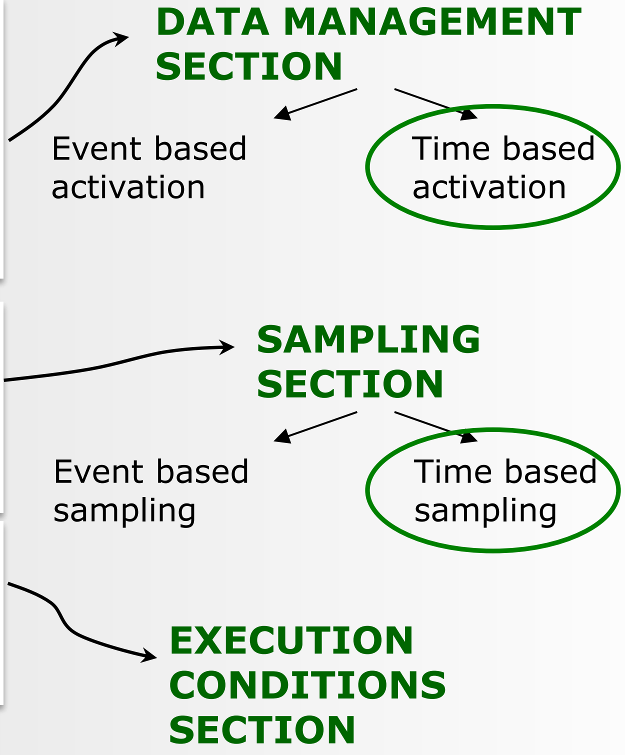
LLQ: NON FUNCTIONAL CHARACTERISTICS

- Non functional fields exposed by FPC are expressed in an abstract way and TRANSLATED in concrete values handled by physical devices
- Example: the power level in a device
  - voltage value
  - predicted from the number of performed operations
  - set to 100% for a.c. powered devices

LLQ: AN EXAMPLE

Sample the temperature every 30 seconds and, every 10 minutes, report the number of samples that exceeded a given threshold

```
INSERT INTO STREAM Table (sensorID, temperature)
LOW:
   EVERY 10 m
   SELECT ID, COUNT(temp, 10 m)

   SAMPLING
      EVERY 30 s
      WHERE temp > 100

   EXECUTE IF
      powerLevel > 0.2 AND EXISTS (temp)
```



DATA MANAGEMENT SECTION

Event based activation

Time based activation

SAMPLING SECTION

Event based sampling

Time based sampling

EXECUTION CONDITIONS SECTION

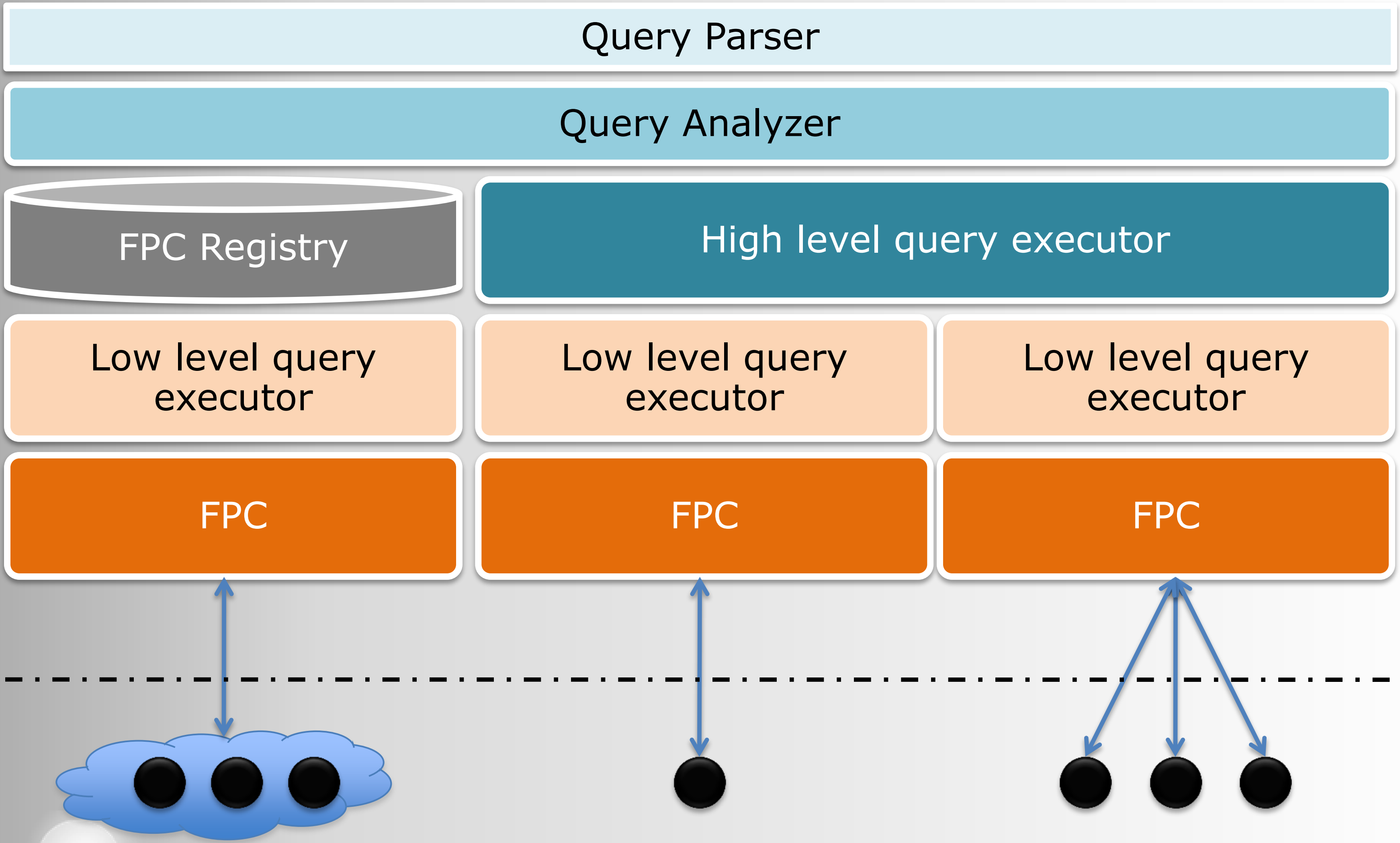
HIGH LEVEL QUERIES

- Perform complex SQL queries on windows extracted from one or more input streams
  - Time driven
  - Event driven
- Every record is time-stamped

QUERY EXAMPLE 2

```
CREATE OUTPUT STREAM LowPoweredDevices (sensorID ID) AS LOW:
    EVERY ONE
    SELECT ID
    SAMPLING EVERY 24 h
        WHERE powerLevel < 0.15
    EXECUTE IF deviceType = "WirelessNode"

CREATE OUTPUT STREAM NumberOfLowPoweredDevices (counter INTEGER) AS HIGH:
    EVERY 24 h
    SELECT COUNT(*)
    FROM LowPoweredDevices(24 h)
```

PERLA MIDLLEWARE

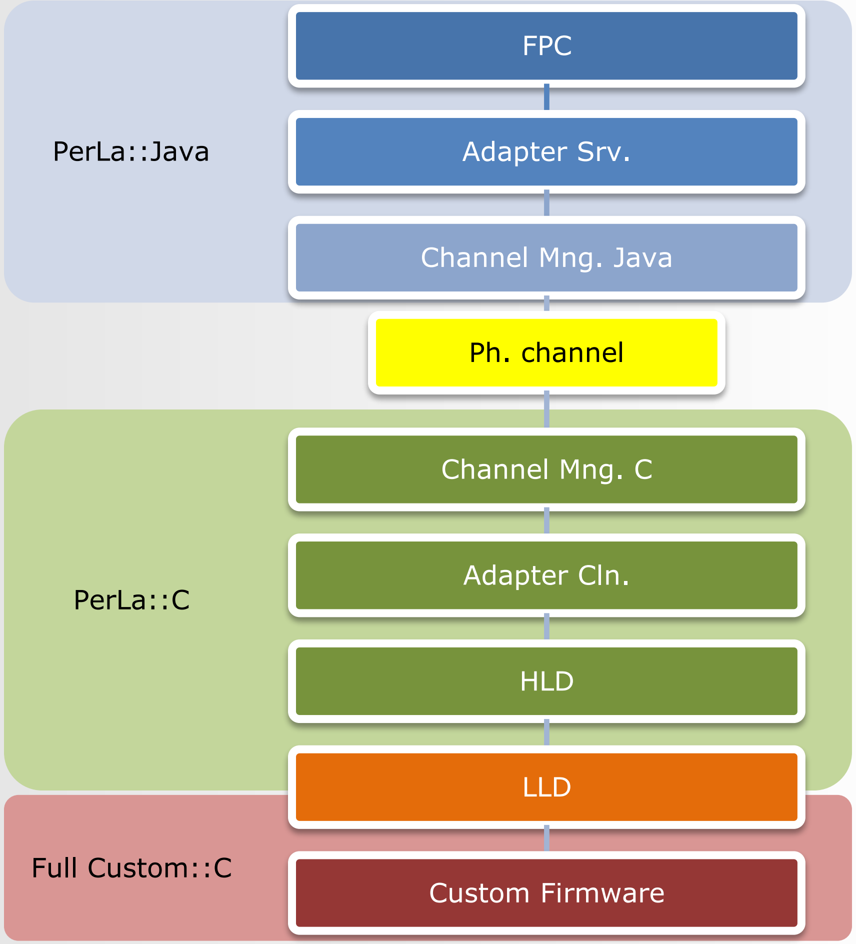


MIDDLEWARE GOALS

- Providing an ABSTRACTION for each device
- Supporting the EXECUTION OF PERLA QUERIES
- PLUG & PLAY support: allow devices to automatically start query execution when they are powered on
- Making the DEFINITION and the ADDITION of new devices (and new technologies) easy, reducing the amount of the needed low level code

FUNCTIONALITY PROXY COMPONENT (FPC)

- The FPC is defined as a Java object representing a physical device.
- The FPC must be instantiated on a system capable of:
  - Running a Java Virtual Machine (JVM)
  - Connecting to a TCP-IP network
- The middleware manages the **COMMUNICATION PROTOCOL** between the FPC and the physical device



LOW LEVEL SUPPORT: HLD AND LLD

- PerLa provides a portable framework, called *HLD* (High Level Driver), which completely abstracts the hardware of the single device

- *HLD* is a set of common components that takes care of the communication with the *FPC*
  - Channel virtualization and data encapsulation (Channel Manager)
  - Multiplexing and routing (Adapter)

- The *LLD* (Low Level Driver) is the software needed by the HLD to access the hardware features of the sensor
  - It has to be written by the user
  - PerLa provides bindings and interfaces

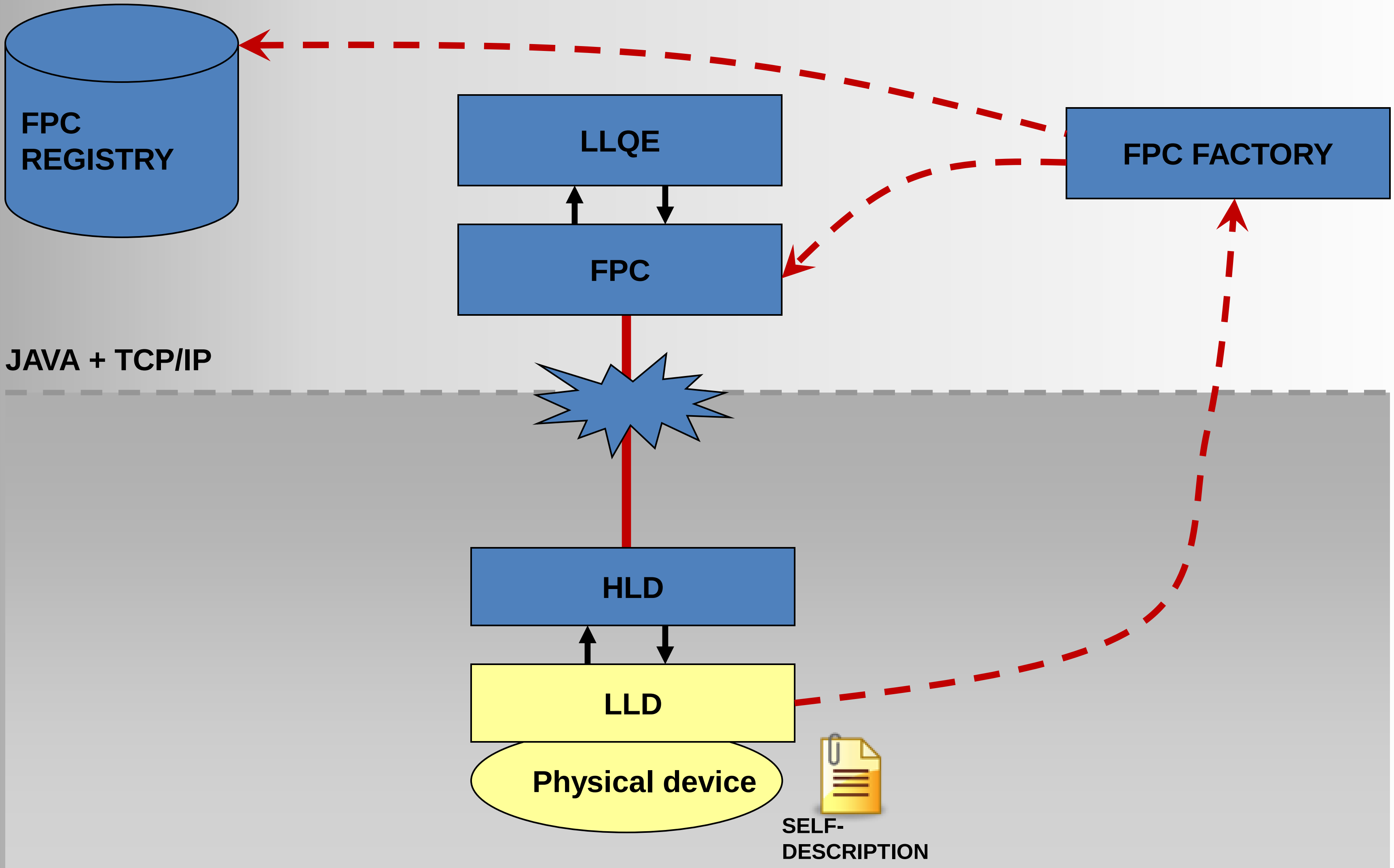
PLUG&PLAY SUPPORT

- PLUG & PLAY at device start-up requires:
  - Dynamic generation of the FPC
  - On the fly binding mechanism to handle connections between the FPC and the physical device
  - Insertion of new FPCs into the Registry

How to build an FPC to handle a new device?

- By means of an xml-based DEVICE DESCRIPTION
  - Sent by the device itself
  - Defines available data streams and events raised
  - Specifies the message protocol used by the device
    - Commands format
    - Data format

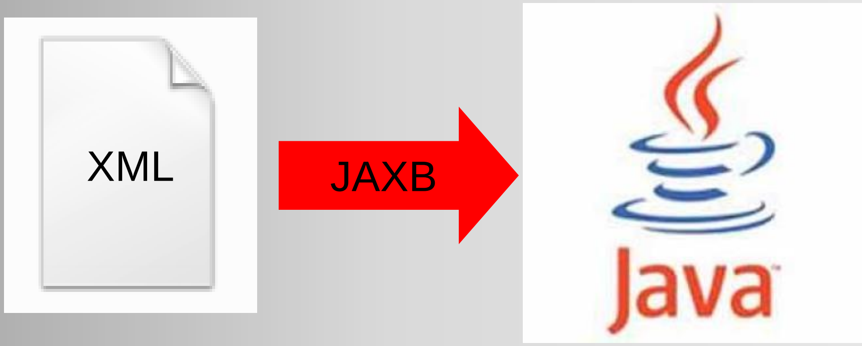
PLUG&PLAY SUPPORT(2)



PLUG & PLAY – FPC FACTORY

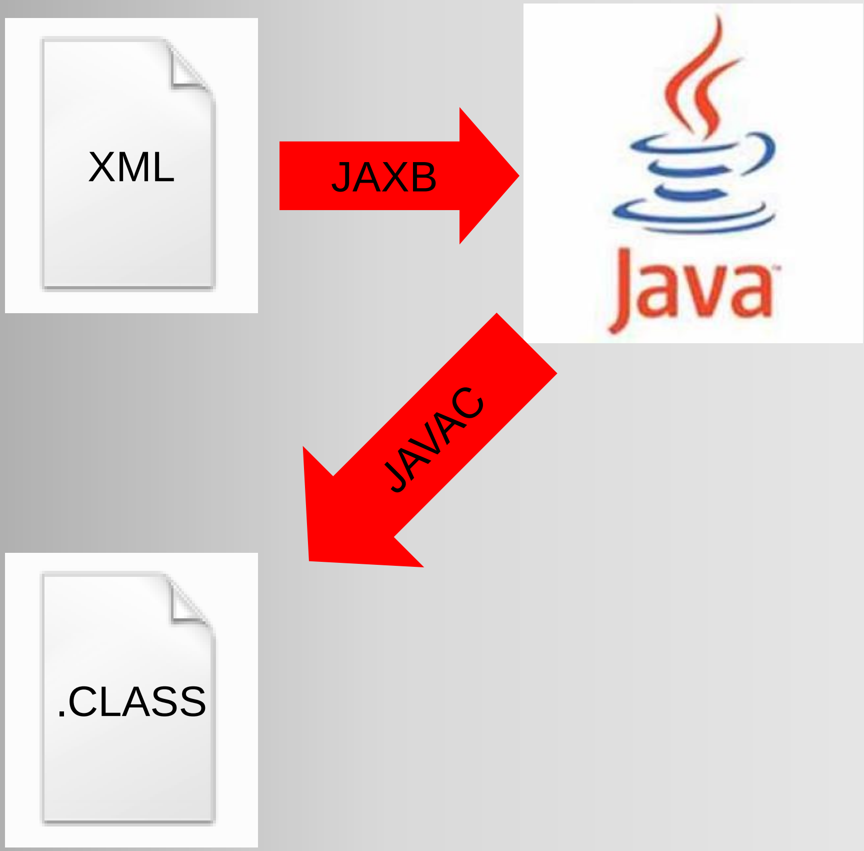
- XML descriptor **validated** by a formerly defined XML Schema

PLUG & PLAY – FPC FACTORY



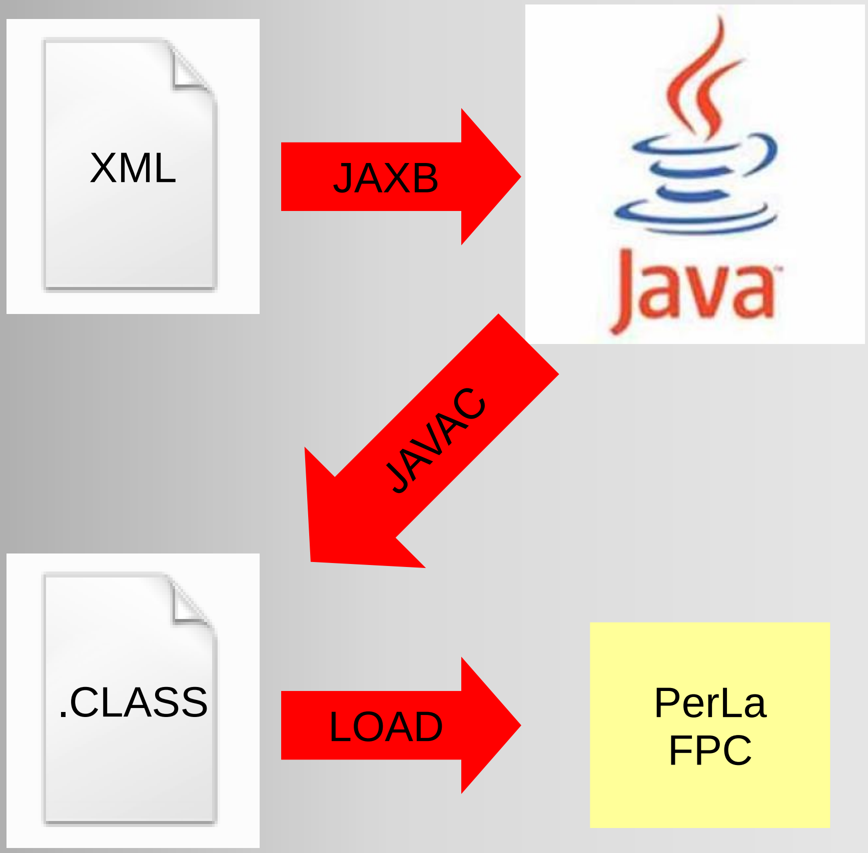
- XML descriptor **validated** by a formerly defined XML Schema
- Run time mapping XML – Java
  - JAXB

PLUG & PLAY – FPC FACTORY



- XML descriptor **validated** by a formerly defined XML Schema
- Run time mapping XML – Java
  - JAXB
- Generated code compilation

PLUG & PLAY – FPC FACTORY



XML

JAXB

JAVAC

.CLASS

LOAD

PerLa FPC

- XML descriptor **validated** by a formerly defined XML Schema
- Run time mapping XML – Java
  - JAXB
- Generated code compilation
- Wrapping

PLUG & PLAY – MAPPING

```
<perlaDeviceElement
name="esempio">
 <perlaSingleDevice
nodeId="1">
  <parameterStructure
name="e">
   <parameterElement
name="param">
   <length>2</length>
   <type nameType="int">
    <sign>signed</sign>
   </type>
   </parameterElement>
   <type>EsempioXML</type>
   <size>2</size>

<endianess>BigEndian</endian
ess>
  </parameterStructure>
 </perlaSingleDevice>
</perlaDeviceElement>
```

```
package
   org.dei.perla.sys.device.fpc.esempio;

   /* IMPORT */

   @StructInfo(endianness =
   Endianness.BIG_ENDIAN,
   totalStructSize = 2)
   public class EsempioXML extends
   AbstractData{

    public EsempioXML() {
     super();
    }
 @SimpleField(size = 2, sign =
   Sign.SIGNED)
    private int param;

    public int getparam() {
     return param;
    }

    public void setparam(int param) {
     this.param = param;
    }
   }
```

PLUG & PLAY – MAPPING

```
<perlaDeviceElement name="esempio">
 <perlaSingleDevice nodeId="1">
   <parameterStructure name="e">
     <parameterElement name="param">
     <length>
     <type na
      <sign>s
     </type>
     </parame
     <type>Es
     <size>2<

<endianess>BigEndian</endianess>
   </parameterStructure>
 </perlaSingleDevice>
</perlaDeviceElement>
```

```
package
   org.dei.perla.sys.device.fpc.esempio;

   /* IMPORT */

   @StructInfo(endianness =
   Endianness.BIG_ENDIAN,
   totalStructSize = 2)
   public class EsempioXML extends

                                          n =

    public int getparam() {
     return param;
    }

    public void setparam(int param) {
     this.param = param;
    }
   }
```

Root element.
It specifies the name of the package
in which the generated classes will reside

PLUG & PLAY – MAPPING

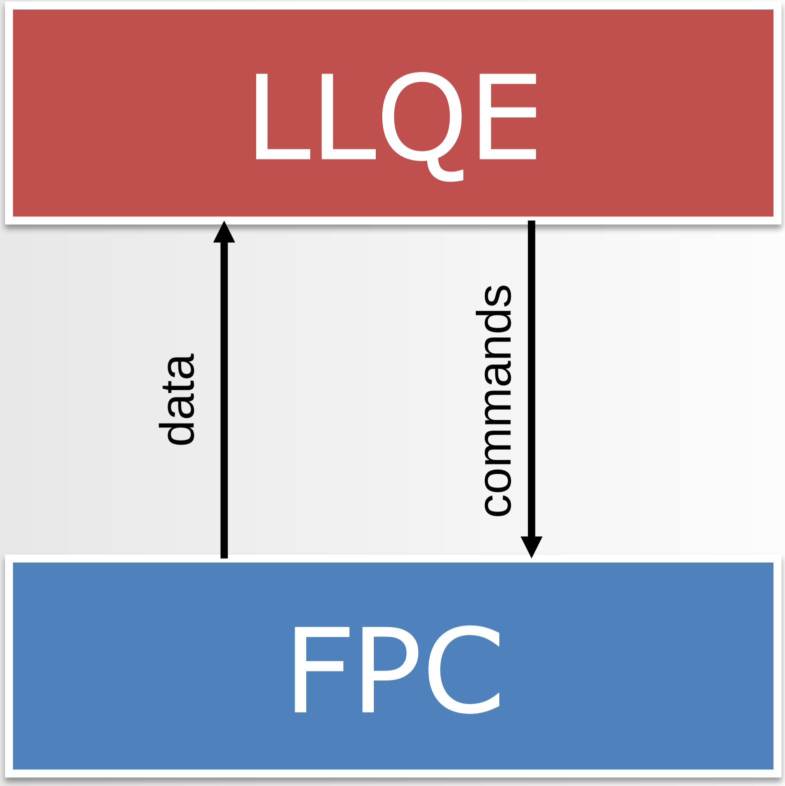
```
<perlaDeviceElement
name="esempio">
  <perlaSingleDevice
nodeId="1">
    <parameterStructure
name="e">
      <parameterElement
name="param">
      <length>2</length>
      <type nameType="int">
        <sign>signed</sign>
      </type>
      </parameterElement>
      <type>EsempioXML</type>
```

A parameter
is represented as a
Java variable.

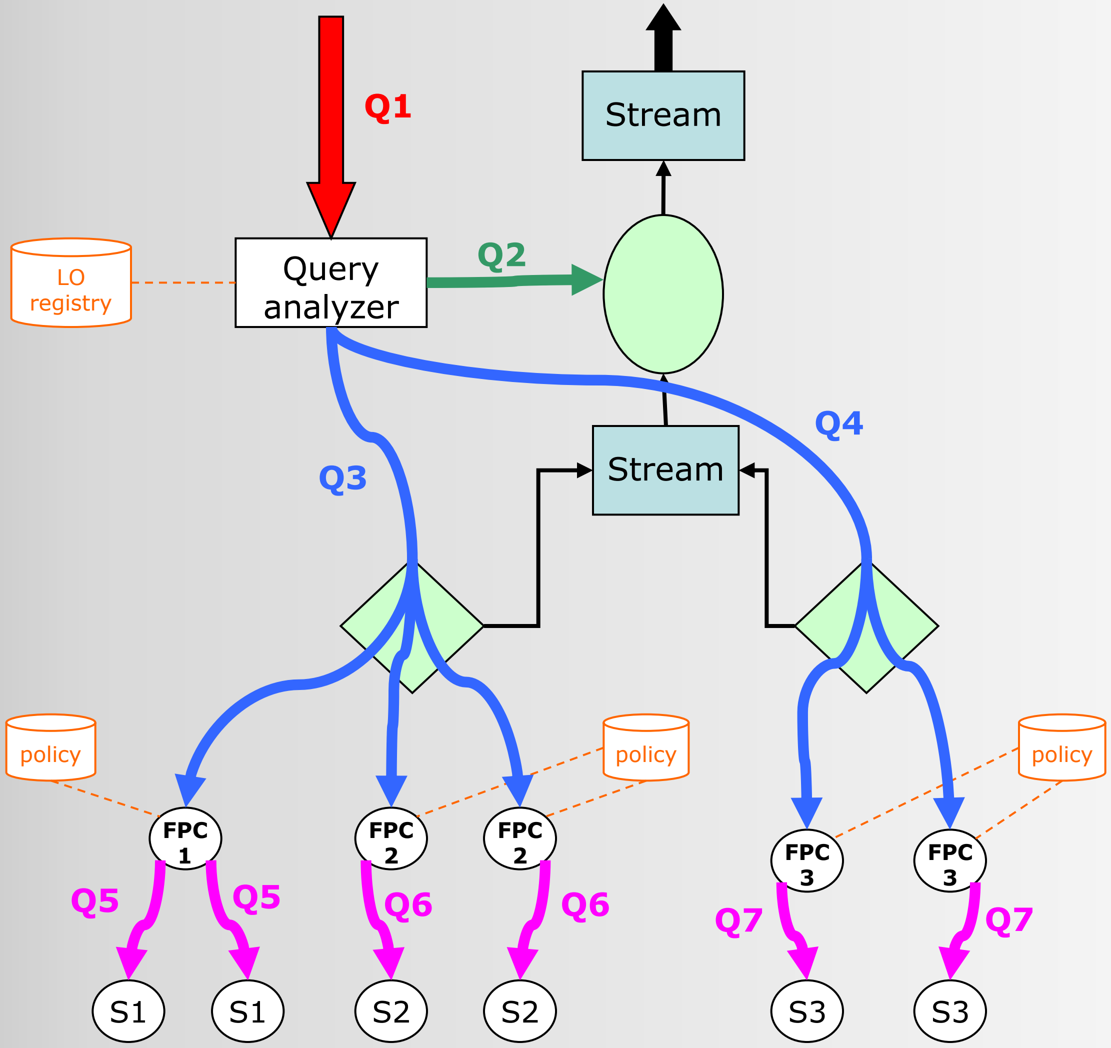
```
package
    org.dei.perla.sys.device.fpc.esempio
    ;

    /* IMPORT */

    @StructInfo(endianness =
    Endianness.BIG_ENDIAN,
    totalStructSize = 2)
    public class EsempioXML extends
    AbstractData{

     public EsempioXML() {
      super();
     }
  @SimpleField(size = 2, sign =
    Sign.SIGNED)
     private int param;

     public int getparam() {
      return param;
     }

     public void setparam(int param) {
      this.param = param;
     }
    }
```

LOW LEVEL QUERY EXECUTOR

- The LLQE (Low Level Queries Executor) is a Java component placed on top of FPC.
  - Retrieve needed data from the underlying FPC and to compute QUERY RESULTS.
- An LLQE supports the simultaneous execution of all the low level queries running on the node.



QUERY DEPLOYMENT



PerLa PERvasive LAnguage

PerLa MODELING AND QUERY STYLE

- In PerLa the RFID system can be modeled in two equivalent ways:
  - RFID readers as data stream generators, RFID tags as data (“***Which Tags passed under Reader R1?***”)
    - Limited number of data streams (one per reader)
    - Adding new tags does not modify the PerLa internal state
  - RFID Tags as (virtual) data stream generators (“***Which Reader read tag #1 in the last hour?***”)
    - Many short data streams
    - Adding a new tag affects PerLa internal state (new FPC required)

RFID READERS AS DATA SOURCES

Which Tags passed under Reader R1 in the last ten minutes?

RFID READERS AS DATA SOURCES

Which Tags passed under Reader R1 in the last ten minutes?

First: device attributes and event definition

RFID READERS AS DATA SOURCES

Which Tags passed under Reader R1 in the last ten minutes?

First: device attributes and event definition

RFID Reader attributes		
Attribute	Role	Data Type
R_id	Reader identifier (static attribute)	ID
deviceType	Device type identifier(static attribute)	STRING
Tag_id	Last RFID tag identifier read	ID
Time	Last reading Timestamp	TIMESTAMP
Events		
tagRead	Notifies when a tag is read	

PerLa
PERvasive LAnguage

RFID READERS AS DATA SOURCES

Second: output stream definition

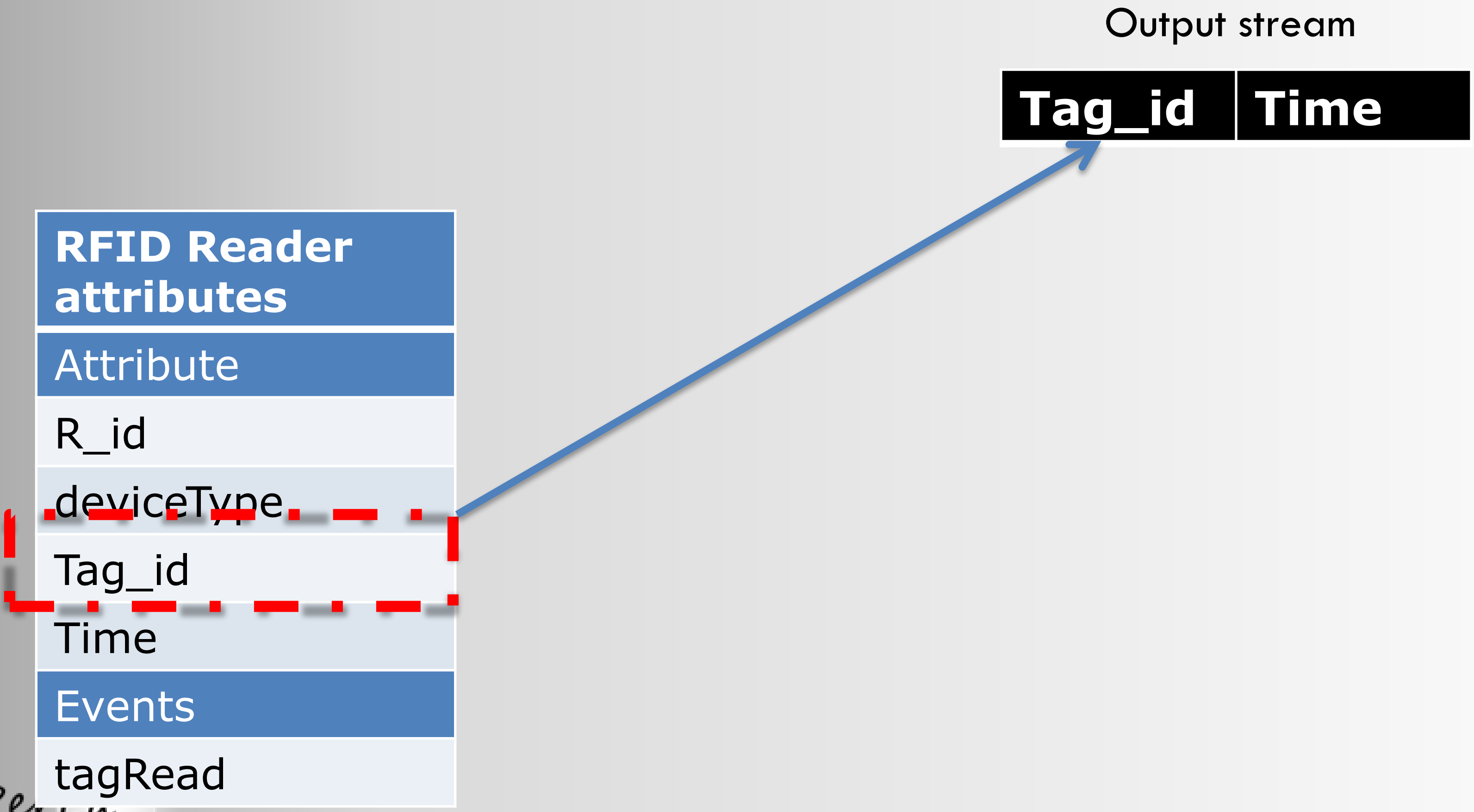
Output stream

Tag_id	Time

RFID Reader attributes
Attribute
R_id
deviceType
Tag_id
Time
Events
tagRead

RFID READERS AS DATA SOURCES

Second: output stream definition



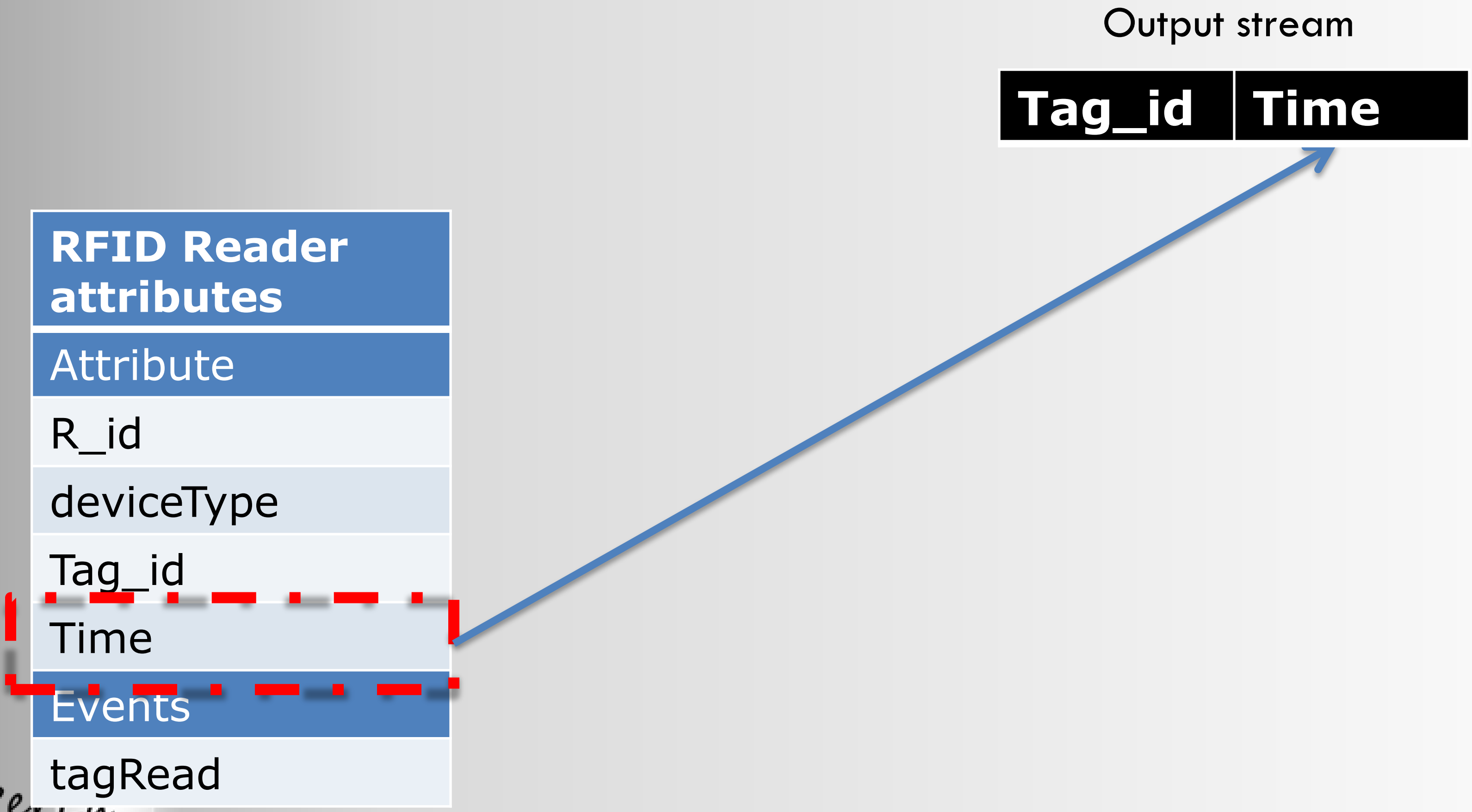
Output stream

Tag_id	Time

RFID Reader attributes
Attribute
R_id
deviceType
Tag_id
Time
Events
tagRead

RFID READERS AS DATA SOURCES

Second: output stream definition



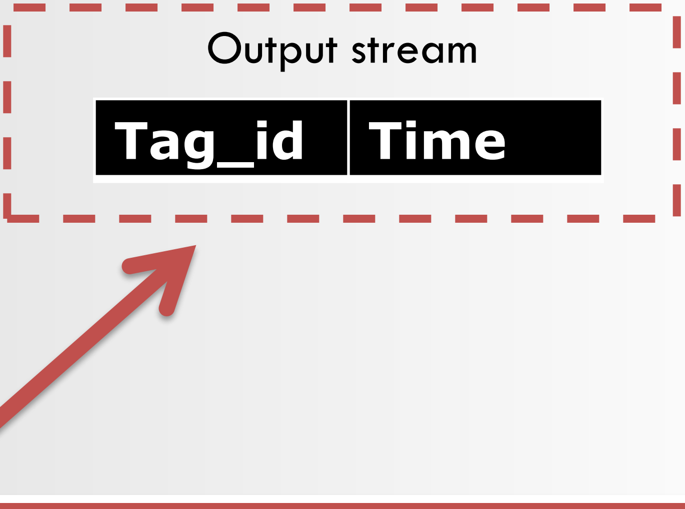
Output stream

Tag_id	Time

RFID Reader attributes
Attribute
R_id
deviceType
Tag_id
Time
Events
tagRead

RFID READERS AS DATA SOURCES

Second: output stream definition



Output stream

Tag_id	Time

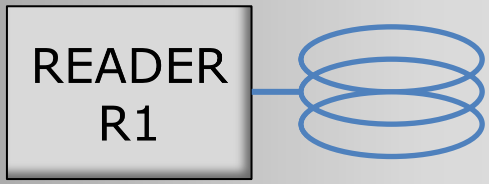
```
CREATE OUTPUT STREAM
        Readings(Tag_id ID, Time TIMESTAMP)
```

RFID READERS AS DATA SOURCES

Third: feed the output stream

Output stream

Tag_id	Time

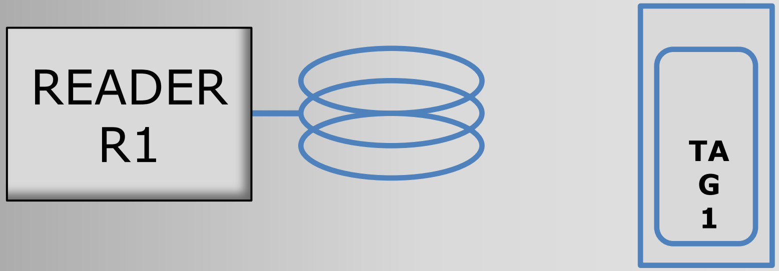


RFID READERS AS DATA SOURCES

Third: feed the output stream

Output stream

Tag_id	Time

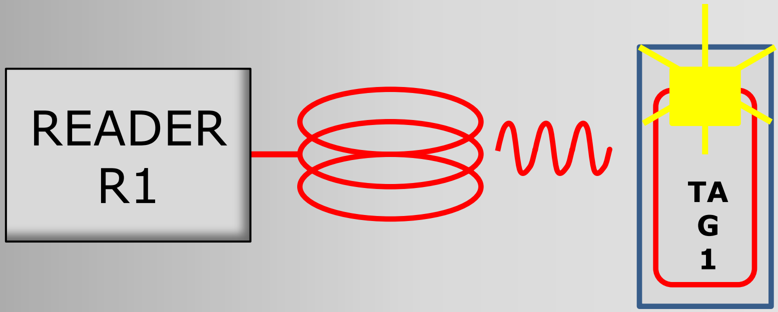


RFID READERS AS DATA SOURCES

Third: feed the output stream

Output stream

Tag_id	Time
1	t1



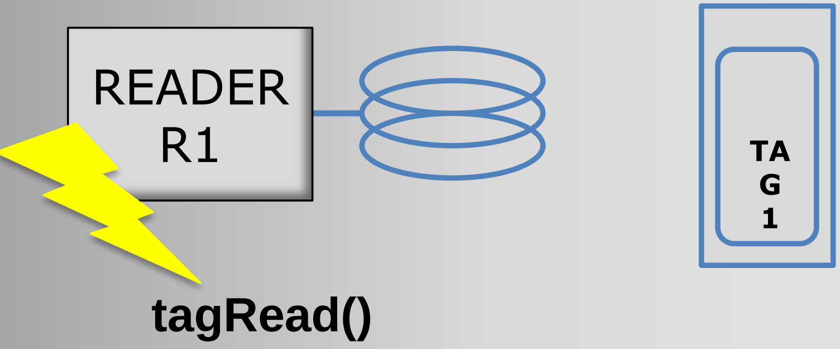
PerLa
PERvasive LAnguage

RFID READERS AS DATA SOURCES

Third: feed the output stream

Output stream

Tag_id	Time
1	t1



RFID READERS AS DATA SOURCES

Third: feed the output stream

Output stream

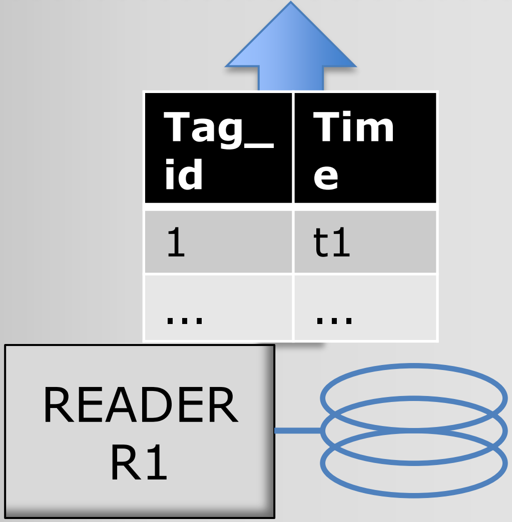
Tag_id	Time
1	t1

```
INSERT INTO STREAM Readings(Tag_id, Time)
EVERY ONE
        SELECT Tag_id, Time
        SAMPLING ON EVENT tagRead()
        EXECUTE IF
                deviceType = “RFID_READER” AND
                R_id = “R1”
```

PerLa
PERvasive LAnguage

RFID READERS AS DATA SOURCES

Fourth: create a fixed width (in terms of time) output snapshot table



Tag_id	Time
1	t1
…	…

LLQ

RFID READERS AS DATA SOURCES

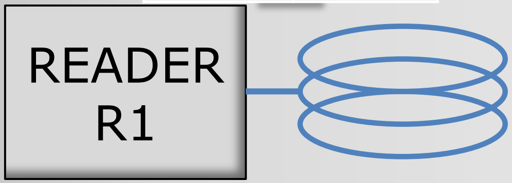
Tag_id	Time
1	t1
1	t2
2	t3
….	….
M	tR

Snapshot table time constraints
|tR-t1| <= 10min

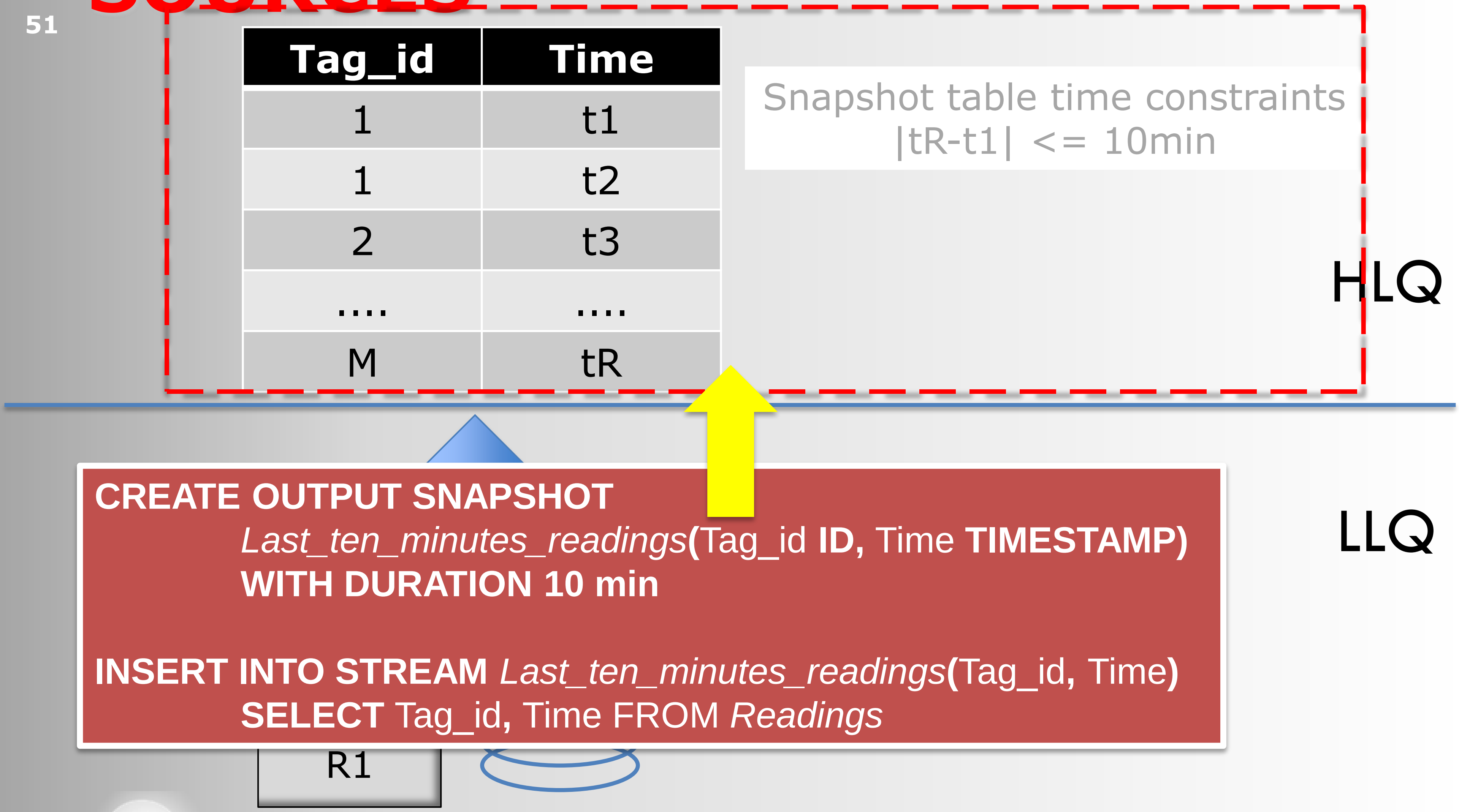
HLQ

LLQ

Tag_id	Time
1	t1
…	…



RFID READERS AS DATA SOURCES



Tag_id	Time
1	t1
1	t2
2	t3
….	….
M	tR

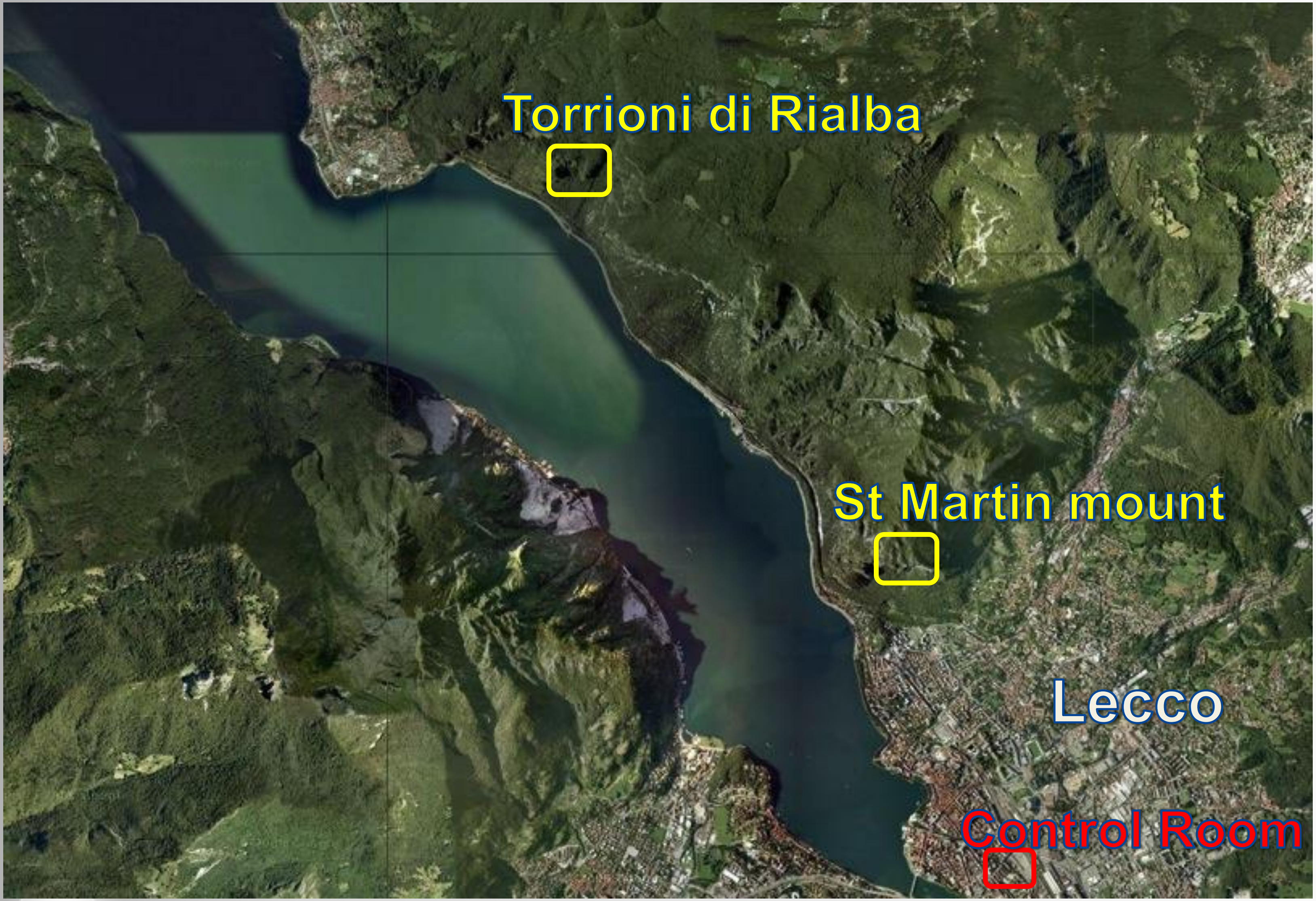
Snapshot table time constraints
|tR-t1| <= 10min

CREATE OUTPUT SNAPSHOT
Last_ten_minutes_readings(Tag_id **ID,** Time **TIMESTAMP)**
WITH DURATION 10 min

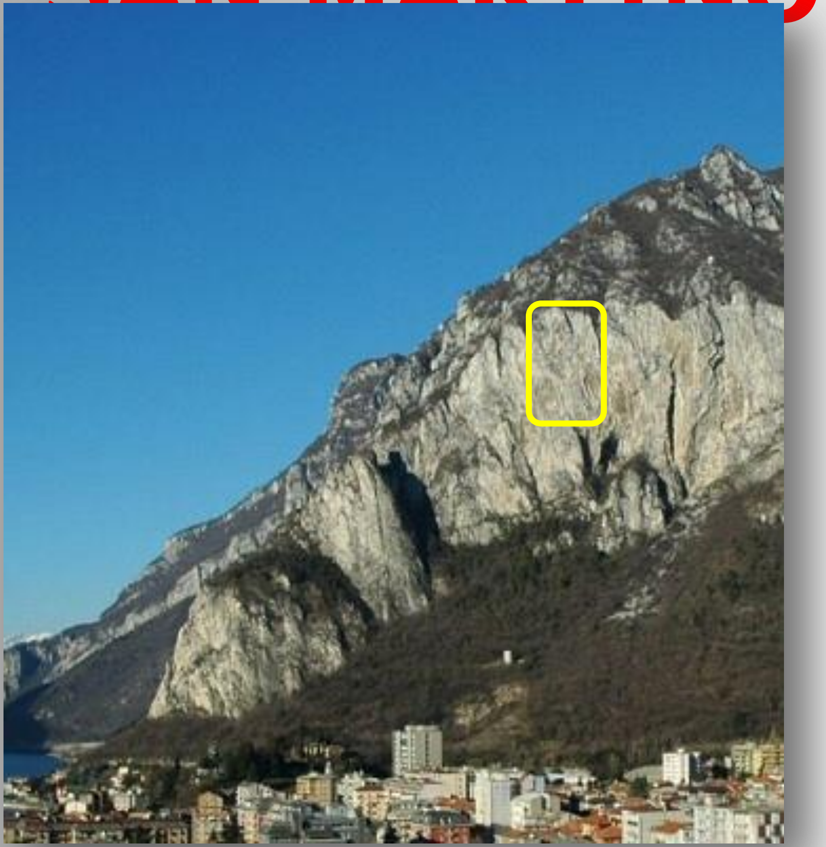
INSERT INTO STREAM *Last_ten_minutes_readings*(Tag_id, Time)
SELECT Tag_id, Time FROM *Readings*

ROCK FALL FORECASTING

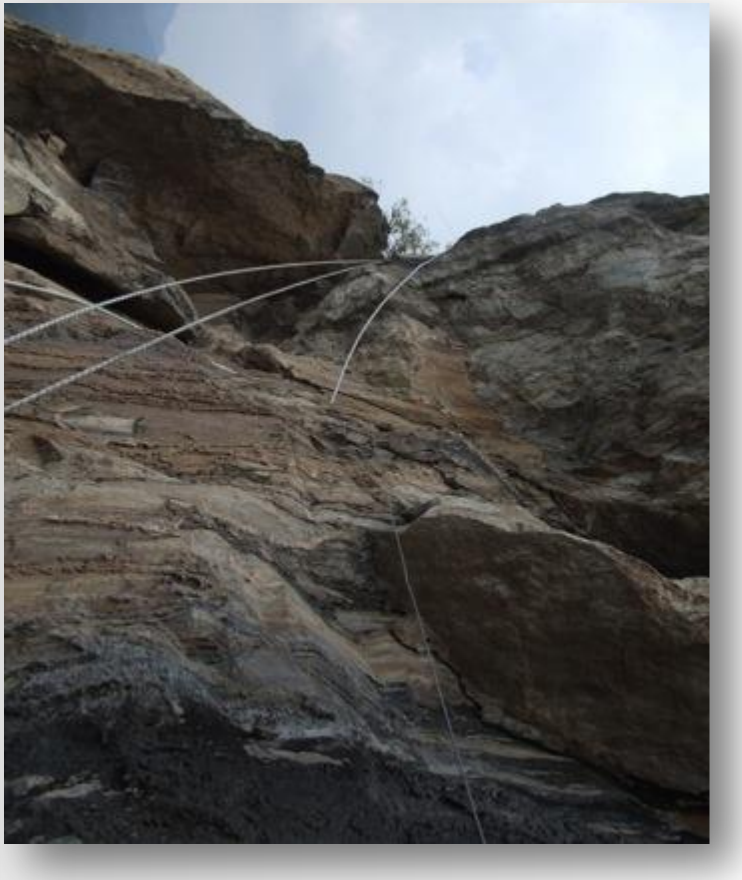
ROCK FALL FORECASTING



ROCK FALL FORECASTING: M. SAN MARTINO



The St. Martin Mount



Detail of the rock face

A POSSIBLE DEPLOYMENT OF THE REAL-TIME MONITORING SYSTEM

Particular of the crown where sensors will be deployed: already collapsed site size (LxHxD) 10x40x10m

The sensing unit

The monitored mountain wall

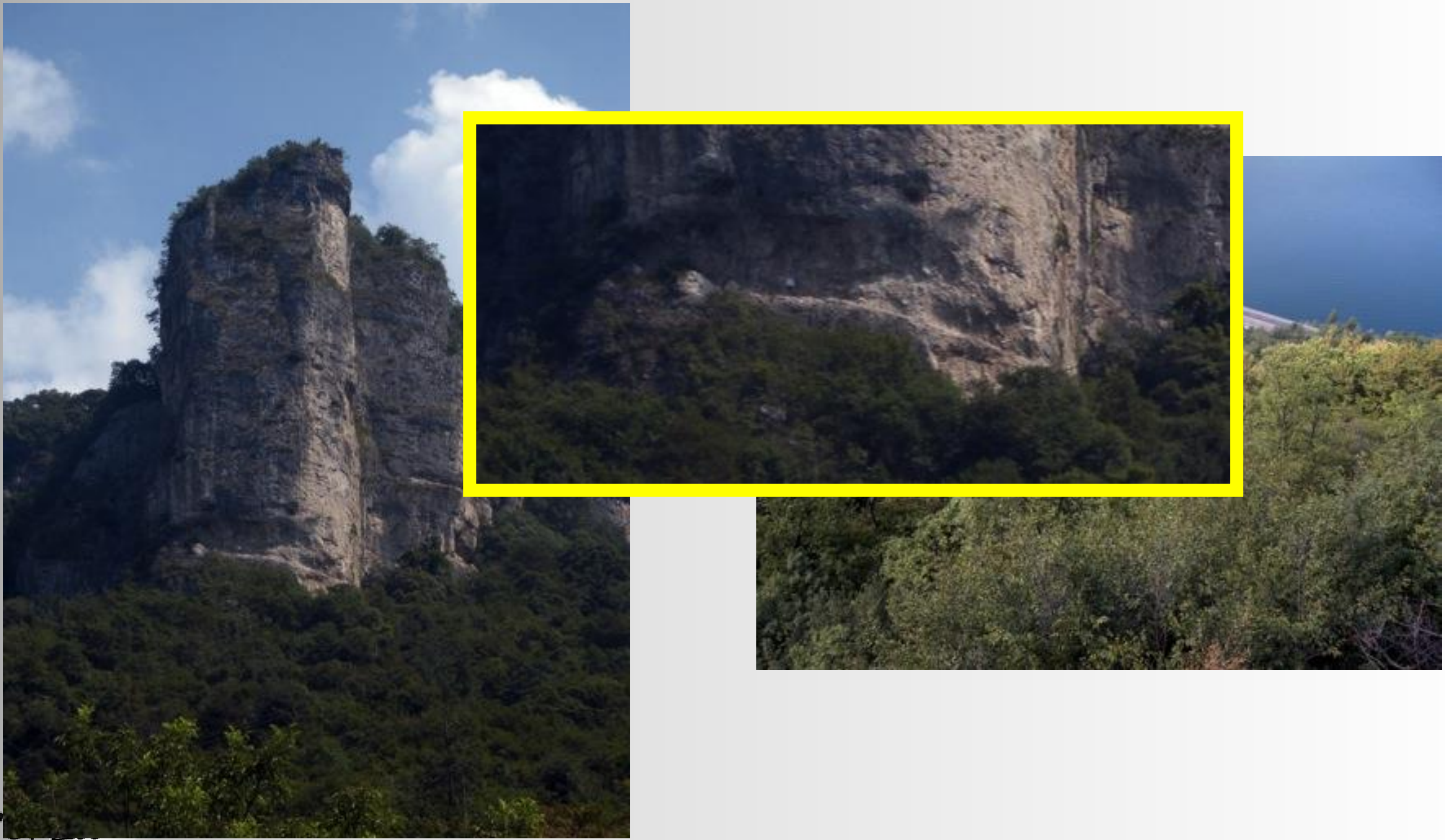
Campus Point with the control room @ 2.5Km

PerLa
PERvasive LAnguage

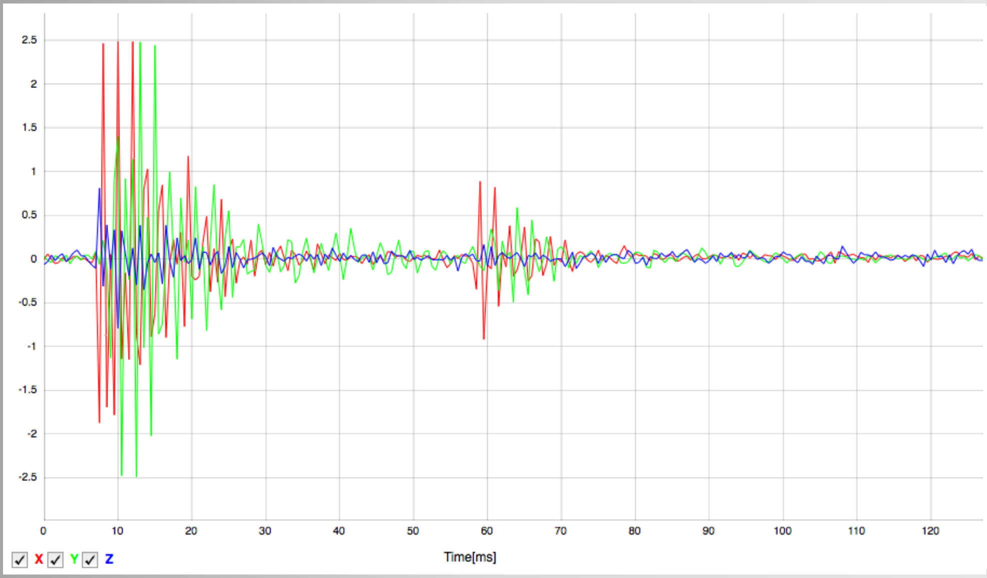
ROCK FALL FORECASTING: TORRIONI DI RIALBA

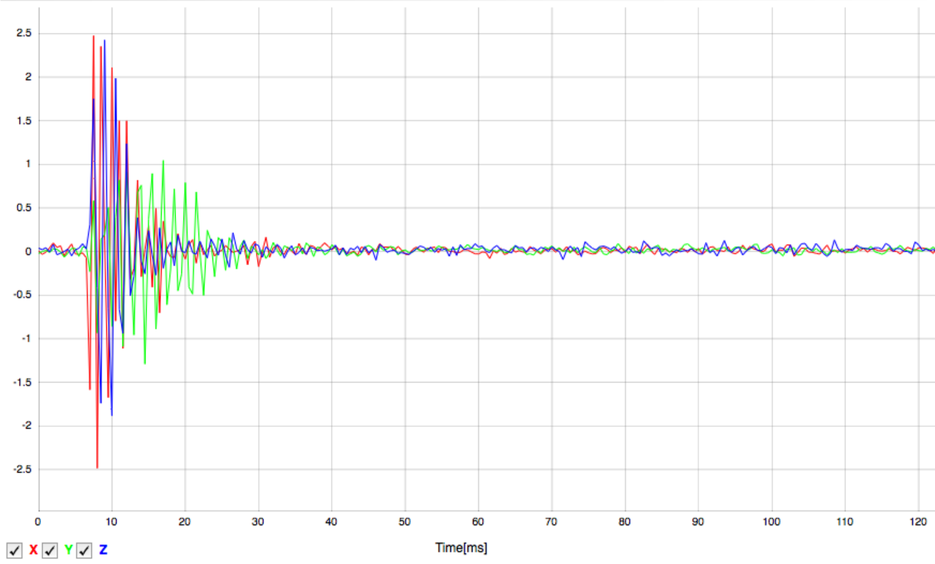
PERvasive LAnguage

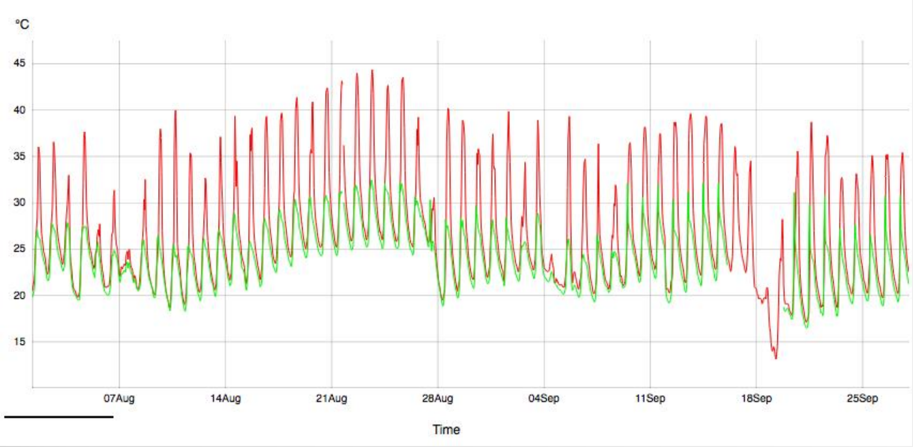
ROCK FALL FORECASTING: TORRIONI DI RIALBA



Deployment Phase







PerLa
PERvasive LAnguage

Deployment Phase

PerLa
PERvasive LAnguage

ON-GOING WORK

- Context Aware Language
  - Context Definition statements
  - Middleware extension to support context
  - Context Management
    - Conflict detection a design time and runtime

- Energy saving data aggregation

BIBLIOGRAPHY

- [1] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk and J. Anderson, “Wireless sensor networks for habitat monitoring”, in Proceedings of ACM international workshop on Wireless sensor networks and applications, pp. 88–97, (2002).
- [2] C. Hartung, R. Han, C. Seielstad and S. Holbrook, “FireWxNet: a multitiered portable wireless system for monitoring weather conditions in wildland fire environments”, in Proceedings of International conference on Mobile systems, applications and services, pp. 28–41 (2006).
- [3] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein, “Energy-efficient computing for wildlife tracking: Design tradoffs and early experiences with zebranet”, in Arthicetrual Support for Programming Languages and Operating Systems (ASPLOS 2002), October 2002.
- [4] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees and M. Welsh, “Monitoring volcanic eruptions with a wireless sensor network”, Wireless Sensor Networks, Proceeedings, pp. 108–120 (2005).
- [5] D. Chu, L. Popa, A. Tavakoli, J. Hellerstein, P. Levis, S. Shenker, and I. Stoica, “The design and implementation of a declarative sensor network systems,” T.R. UCB/EECS-2006-132, pp. 1–14, 2006.
- [6] D. Chu, L. Popa, A. Tavakoli, J. Hellerstein, P. Levis, S. Shenker, and I. Stoica, “The design and implementation of a declarative sensor network systems,” T.R. UCB/EECS-2006-132, pp. 1–14, 2006.
- [7] K. Aberer, M. Hauswirth, and A. Salehi, “A middleware for fast and flexible sensor network deployment,” Proceedings of the 32nd international conference on Very large data bases, pp. 1199–1202, 2006.
- [8] Siemens Sword: internal communication

THANK YOU

PerLa

PERvasive LAnguage

http://perlawsn.sourceforge.net/index.php