BRAIN·WORK

RESEARCH CENTRE

MIDAS WORKSHOP

Finnish Institute of Occupational Health

All workshop material available at

github.com/jtorniainen/midas_workshop

Outline

1. Background and overview
2. Components of a MIDAS network
3. Installation
4. How to use MIDAS
5. The MIDAS API
6. Building a MIDAS node -- case example
7. Supporting tools
8. Contributing to MIDAS
9. Future
10. Q & A

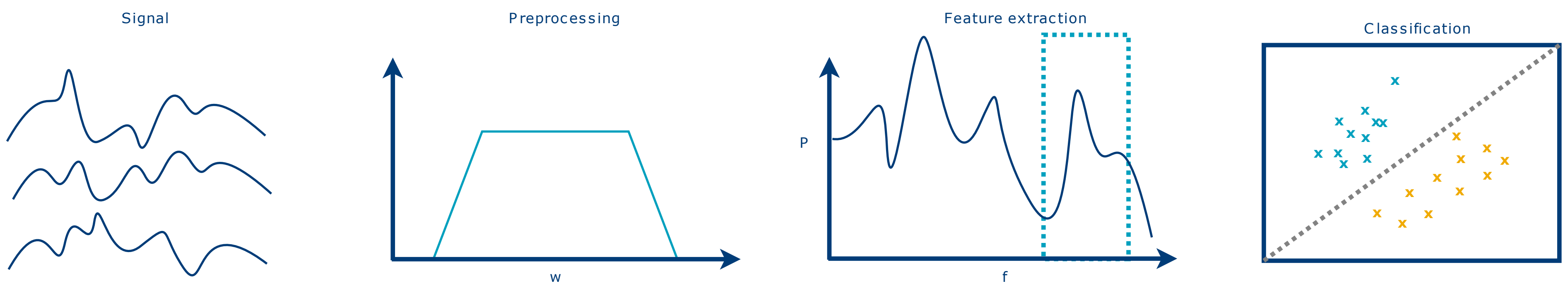
MIDAS

MODULAR INTEGRATED DISTRIBUTED ANALYSIS SYSTEM

1. INTRODUCTION

MIDAS (Modular Integrated Distributed Analysis System) is a system for online analysis of physiologic signals or other streaming time series and easy integration into online machine learning frameworks.

1.1 ANALYZING PHYSIOLOGIC SIGNALS

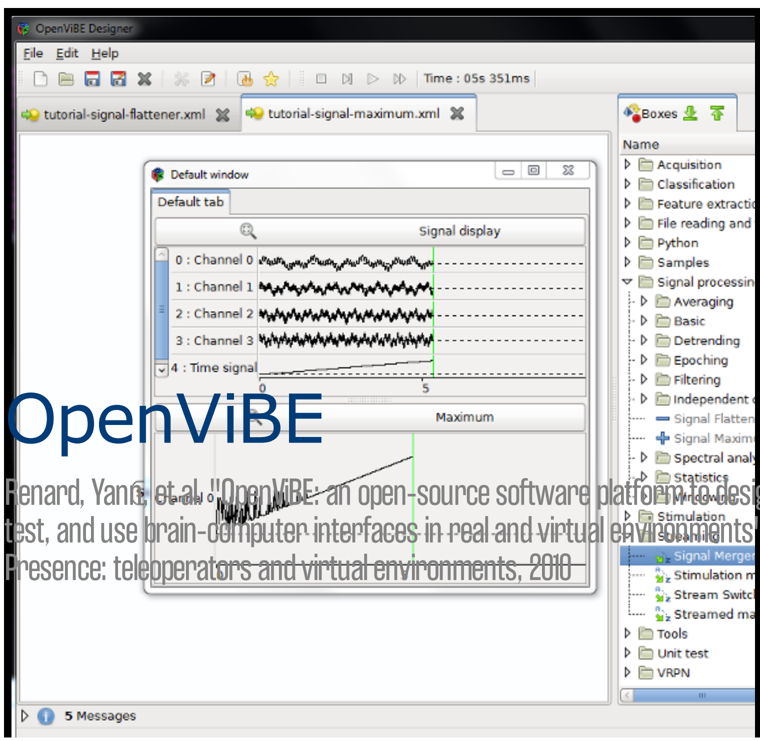


1.2 QUICK FACTS ABOUT MIDAS

- Modular and distributed
- Open source (MIT license)
- Easy to add functionality

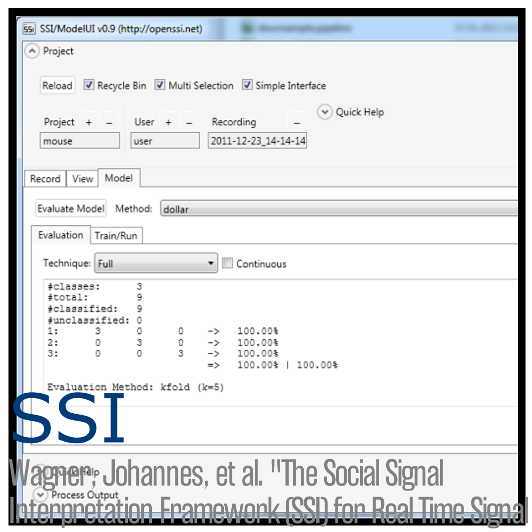
- Easy to integrate with web applications
- Uses open standard communication protocols
- Written in Python

1.3 COMPETITON



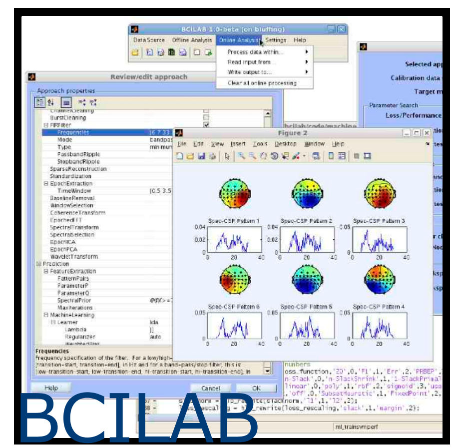
OpenViBE

Renard, Yann, et al. "OpenViBE: an open-source software platform to design, test, and use brain-computer interfaces in real and virtual environments" Presence: teleoperators and virtual environments, 2010



SSI

Wagner, Johannes, et al. "The Social Signal Interpretation Framework (SSI) for Real Time Signal Processing and Recognition" INTERSPEECH, 2011



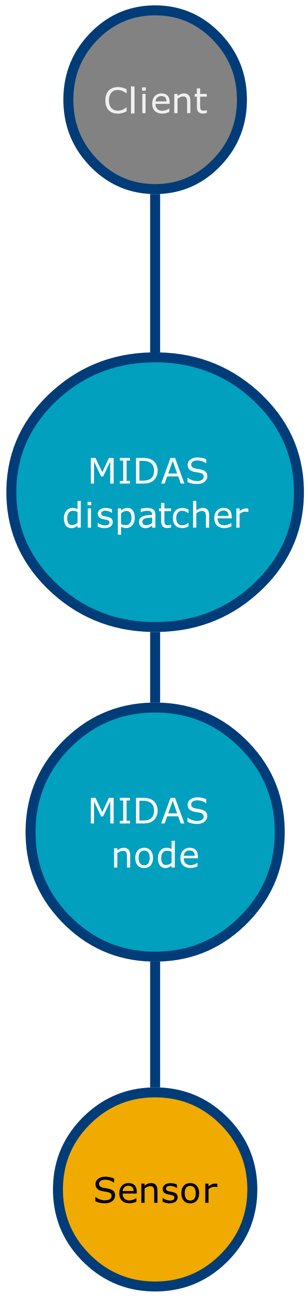
BCILAB

C Kothe, S Makeig. "BCILAB: A platform for brain-computer interface development" Journal of Neural Engineering, 2013

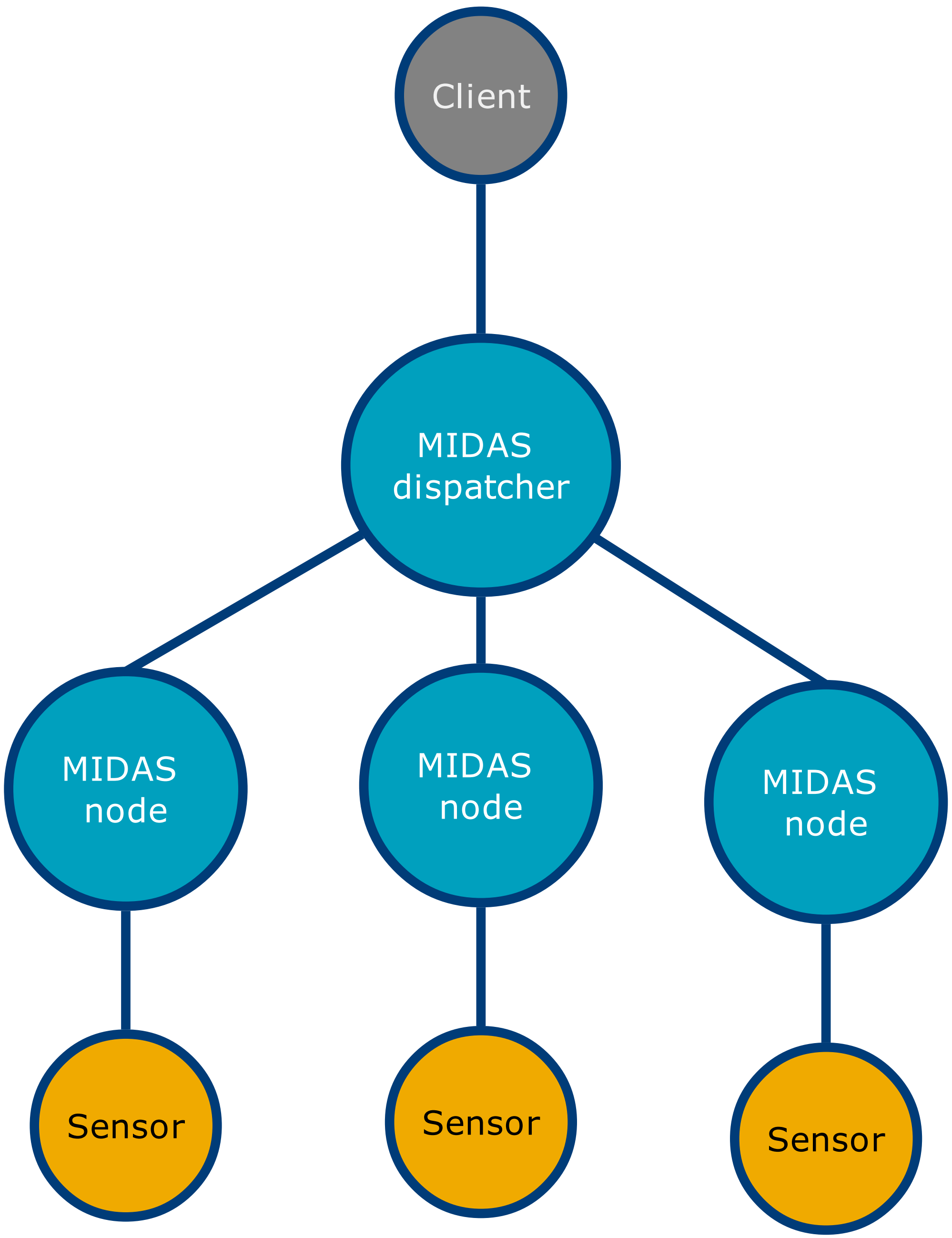
HOW IS MIDAS DIFFERENT?

- **SLIM:** Only bare bones, no clutter
- **MODULAR:** Add functionality as needed
- **DISTRIBUTED:** Runs everywhere (locally, over network, on hand-held, in a cloud)

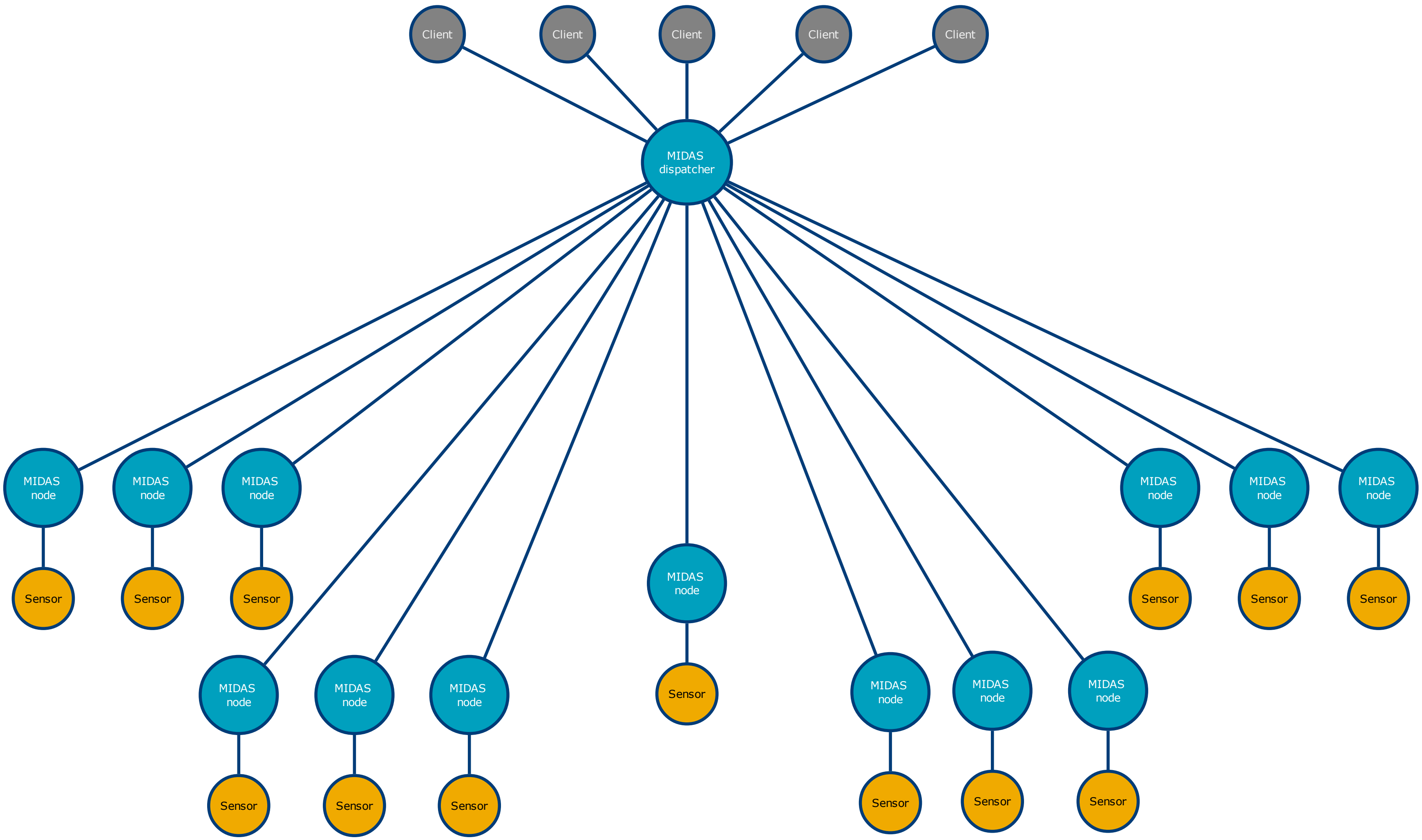
2. MIDAS ARCHITECTURE



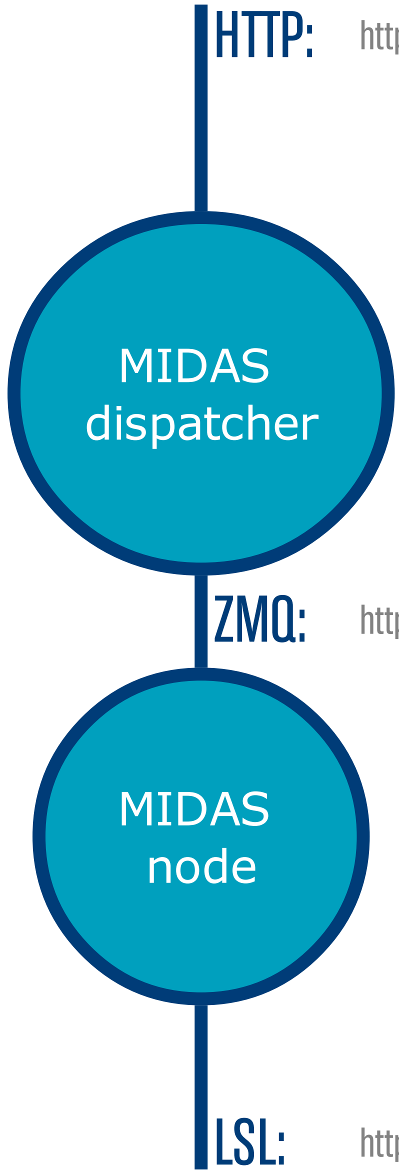
2.1 MIDAS ARCHITECTURE



2.2 MIDAS ARCHITECTURE



2.3 COMMUNICATION PROTOCOLS IN MIDAS



HTTP: http://en.wikipedia.org/wiki/Representational_state_transfer

ZMQ: http://zeromq.org/

LSL: https://code.google.com/p/labstreaminglayer/

3. INSTALLATION

Installing MIDAS is super easy!

1. Install prerequisites
2. Install MIDAS

Remember to use **Python 3!**

3.1 INSTALLATION EASY WAY

```
$ wget https://raw.githubusercontent.com/bwrc/midas/master/requirem
ents.txt

$ pip3 install -r requirements.txt
$ pip3 install git+https://github.com/bwrc/midas/
```

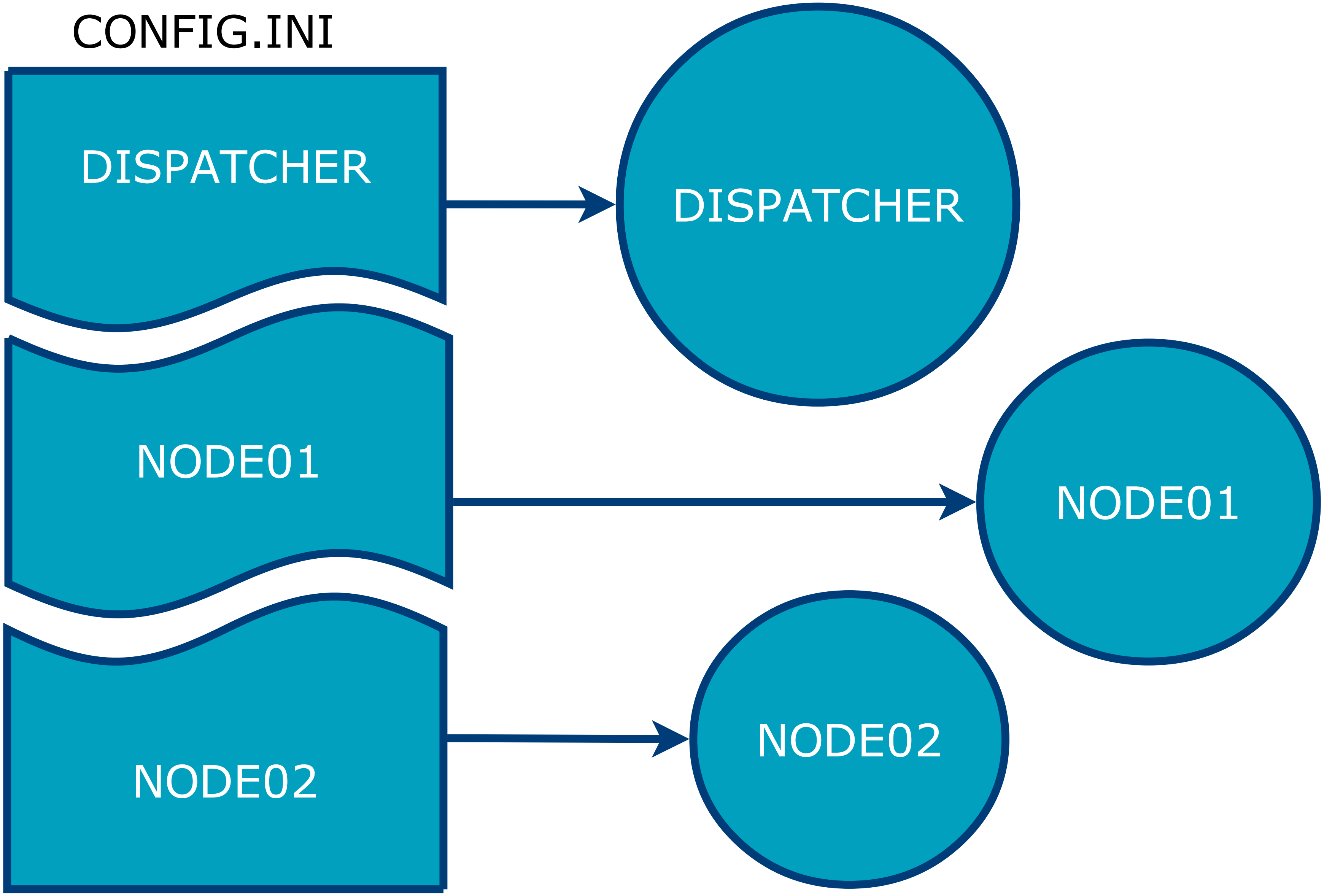
4. BASIC USE

1. Start streaming the data
2. Start the node(s)
3. Start the dispatcher
4. Access nodes through the MIDAS API

4.1 STREAMERS

- Transmit raw data over **lab streaming layer**
- Some sensor devices have built-in LSL support (Neuroelectrics Enobio)
- For some devices you can find LSL drivers from http://code.google.com/p/labstreaminglayer/
- If no streamer exist it must be made manually (not difficult)

4.2 STARTING MIDAS



5 THE MIDAS API

Full reference accessible from wiki

github.com/bwrc/midas/wiki/API

Below are some examples how to use the API

5.2 CALLING MIDAS

Python

```
import requests
r = requests.get("localhost:8080/example_node/metric/metric_a:c
```

R

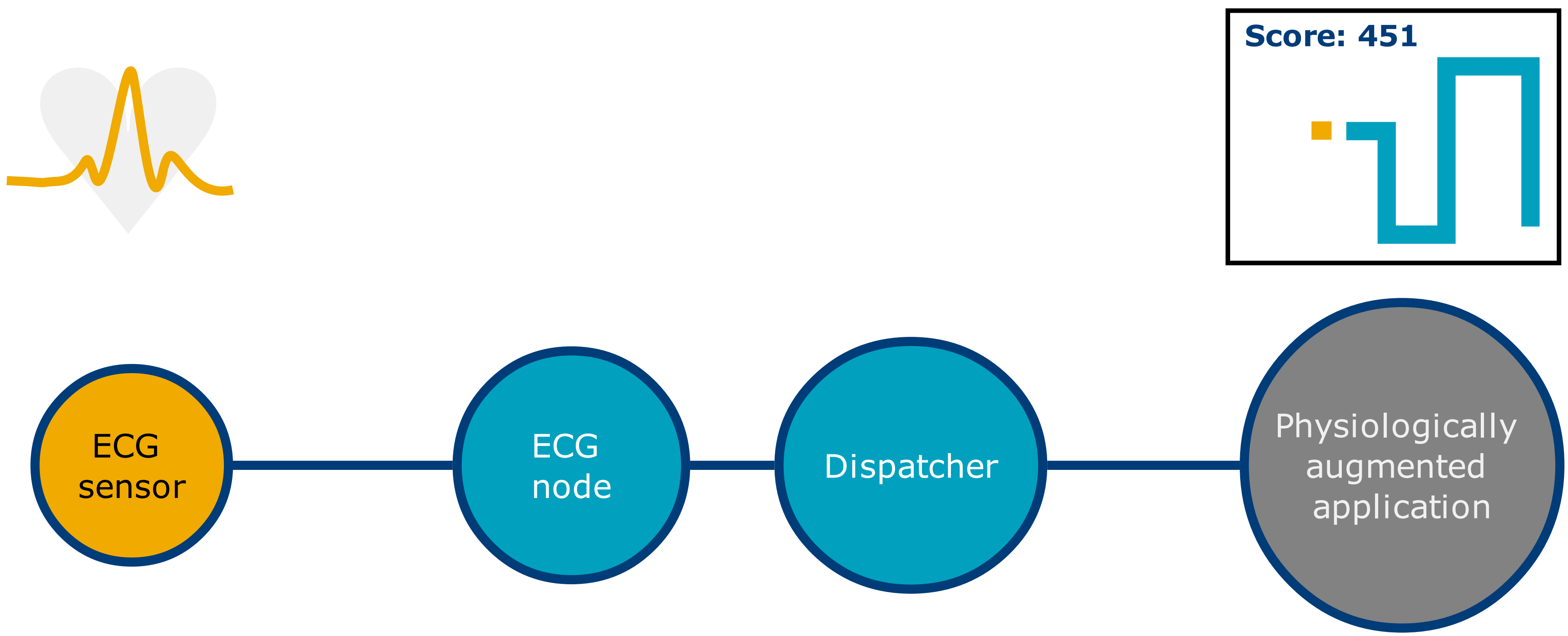
```
library('RCurl')
r <- getURL("localhost:8080/example_node/metric/metric_a:c
```

MATLAB

```
r = urlread('localhost:8080/example_node/metric/metric_a:ch
```

6. BUILDING A MIDAS NODE

Case example describing how to ...



Score: 451
ECG
sensor
ECG
node
Dispatcher
Physiologically
augmented
application

6.1 THE DATA

In this example we will be using simulated electrocardiogram (ECG) data.

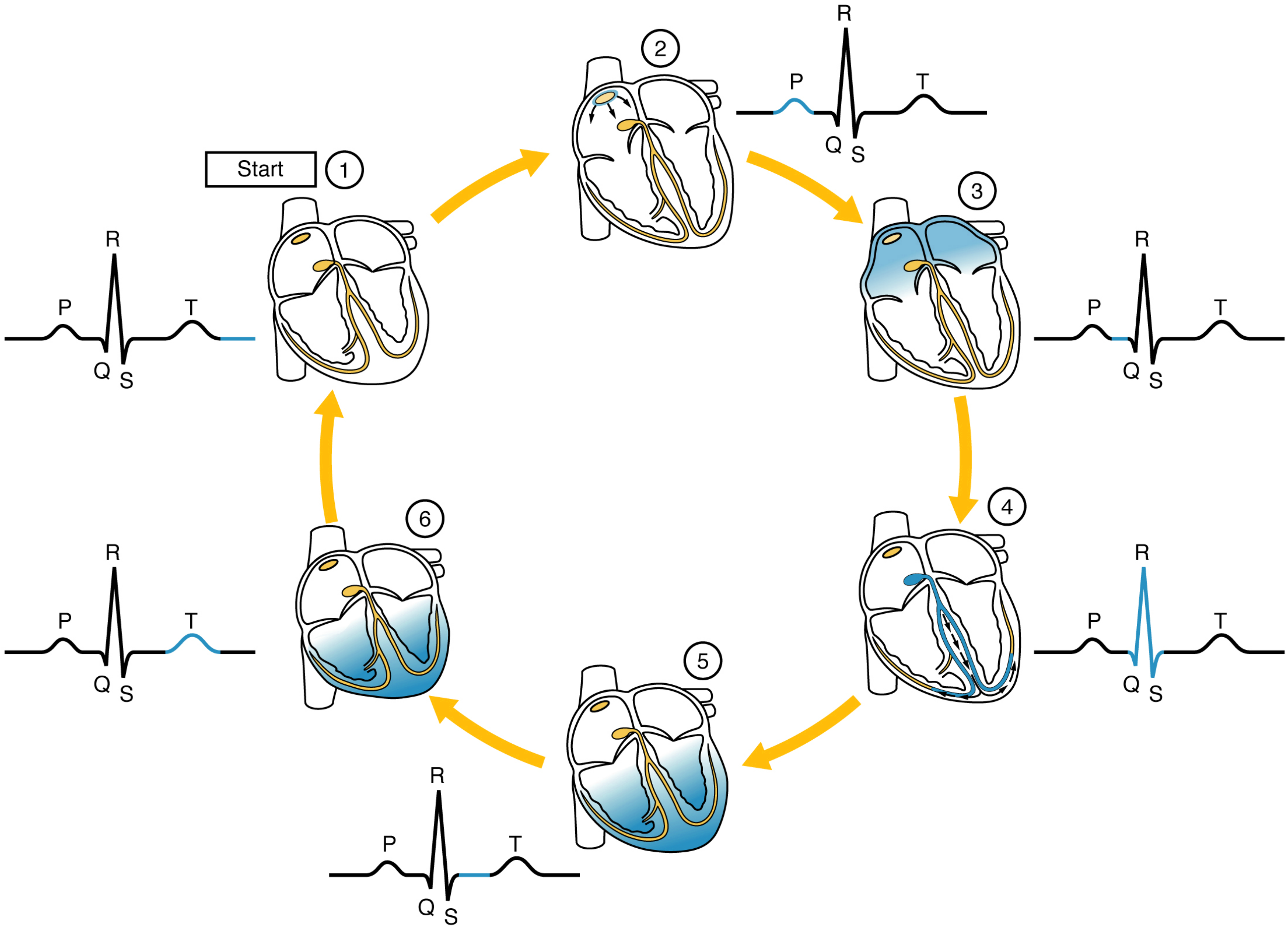
The simulator (and other utilities) available at:

github.com/bwrc/lsltools

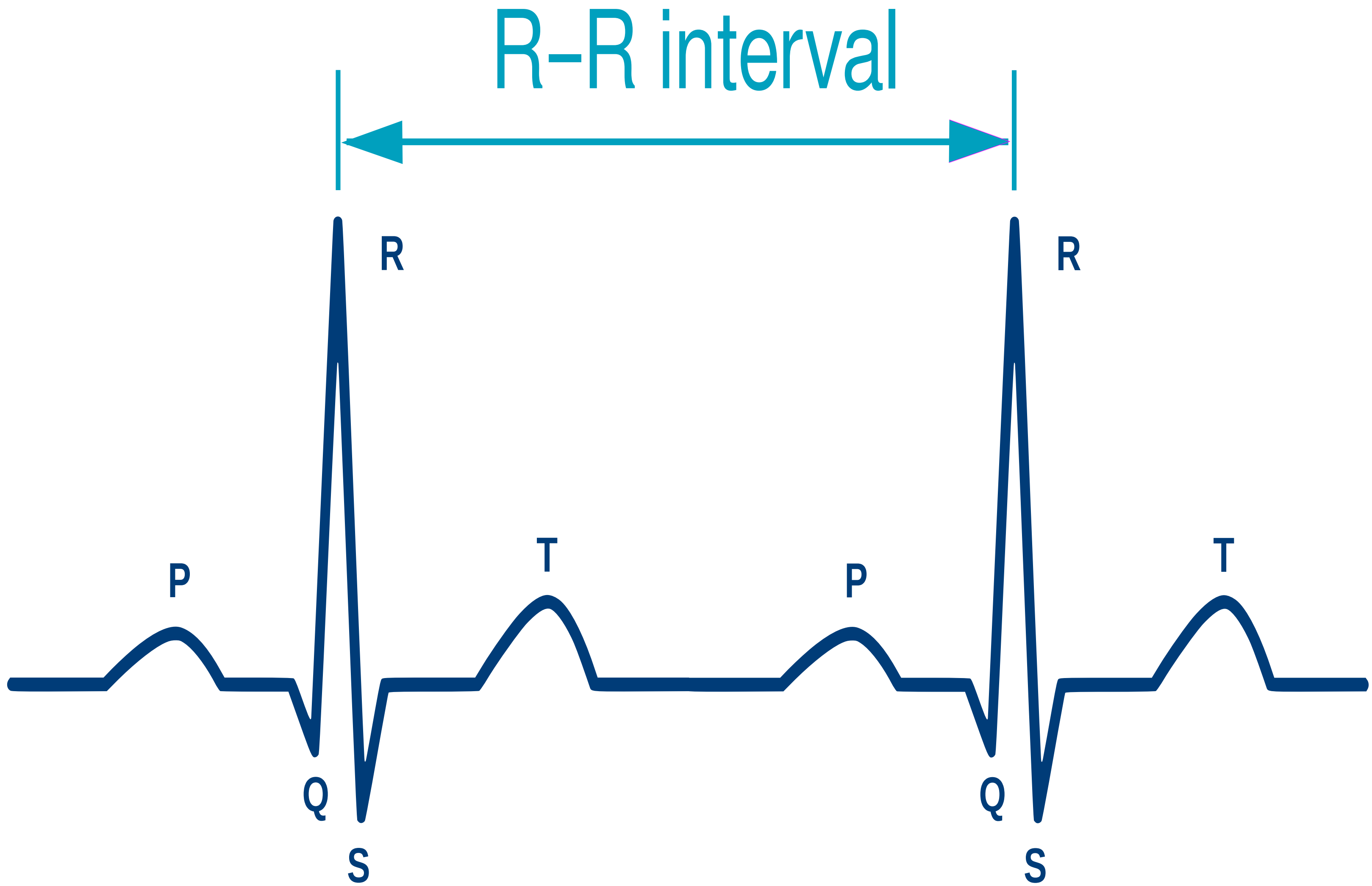
but requires:

physionet.org/physiotools/ecgsyn

in order to work!



Start
1
2
3
4
5
6
P
Q
R
S
T

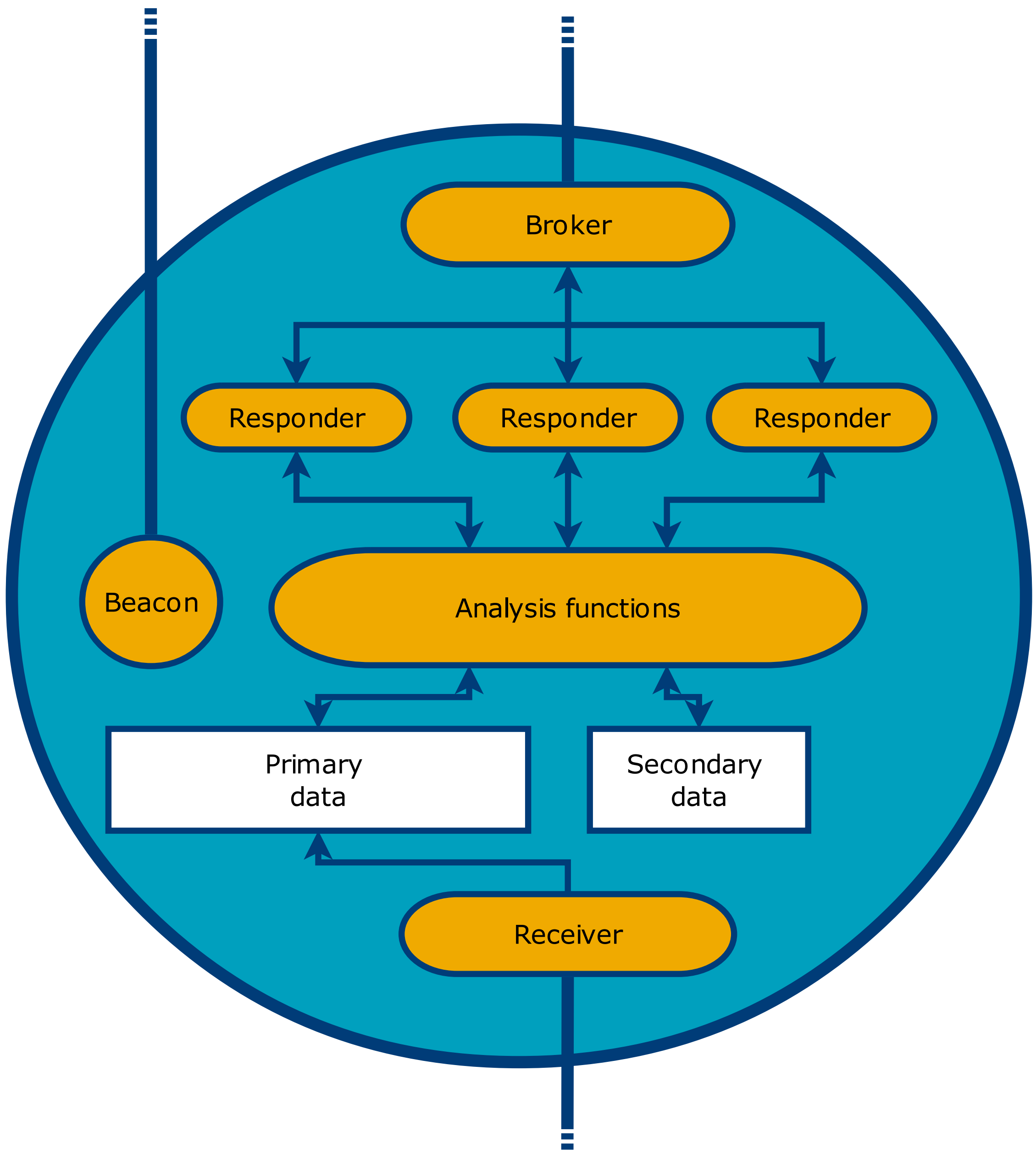

R-R interval
R
R
P
T
P
T
Q
S
Q
S

6.2 SUMMARY OF TERMINOLOGY

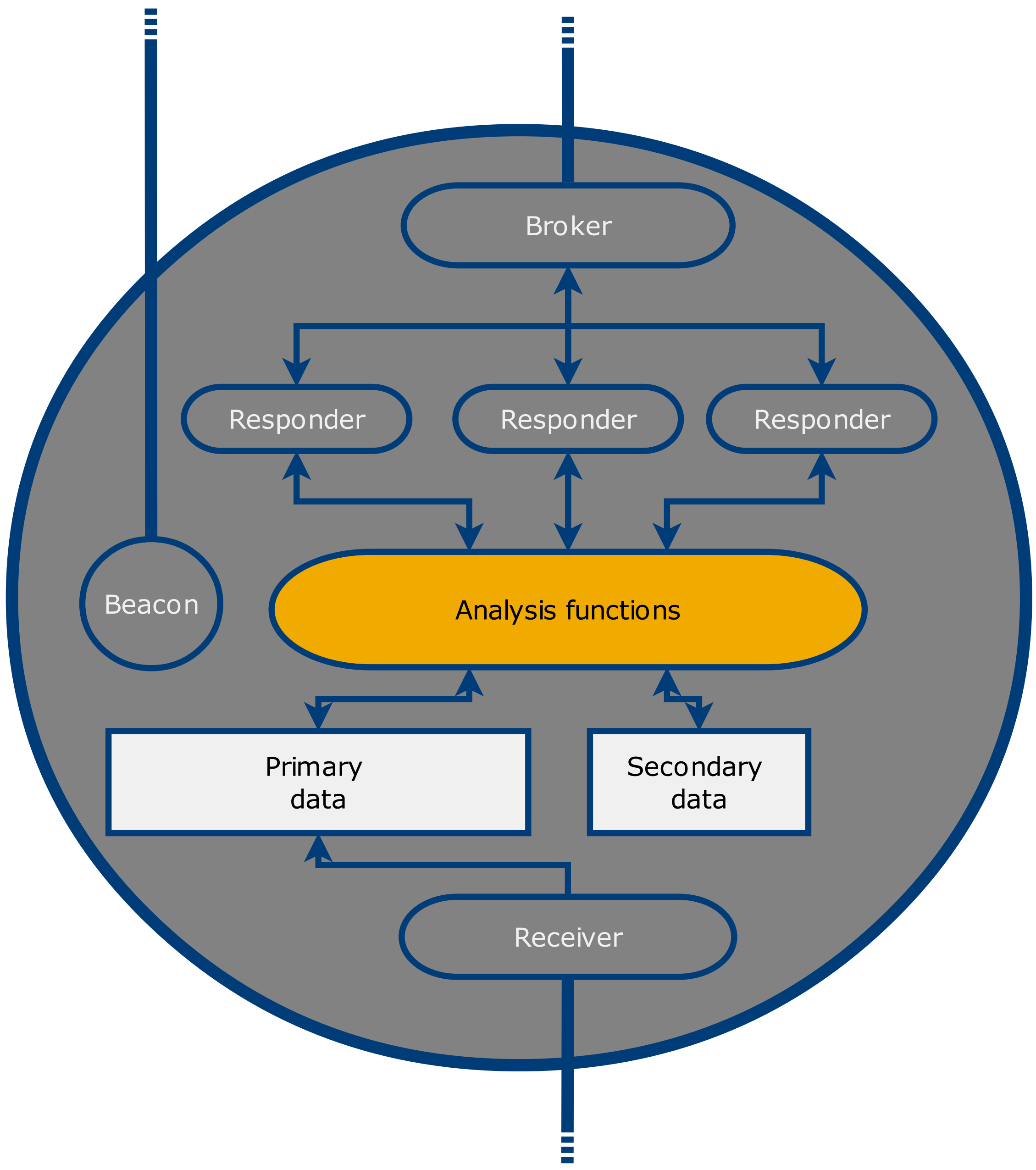
- **ECG:** Electrocardiogram
- **RR:** Interval between two R-peaks (in milliseconds)
- **IBI:** Vector of RR intervals
- **Heart rate:** Temporal average of the IBI vector (often expressed as beats-per-minute or BPM)

```python
from lsltools.sim import ECGData

def stream_ecg():
    ecg = ECGData('/home/jtorniai/ecgsyn_c/ecgsyn', stream_name='ecg_data')
    ecg.start()

    while True:
      try:
          val = int(input('ecg>'))
          print("***\nSetting average heart rate to %d\n***" % val)

          ecg.set_h(val)
          ecg.reset()

      except ValueError:
          print("Input not integer")
```

6.3 NODES

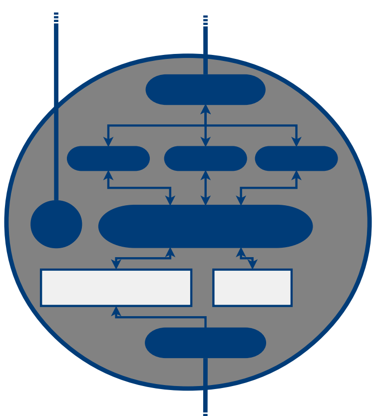

Broker
Responder
Responder
Responder
Beacon
Analysis functions
Primary data
Secondary data
Receiver

6.3 NODES



Broker
Responder
Responder
Responder
Beacon
Analysis functions
Primary data
Secondary data
Receiver

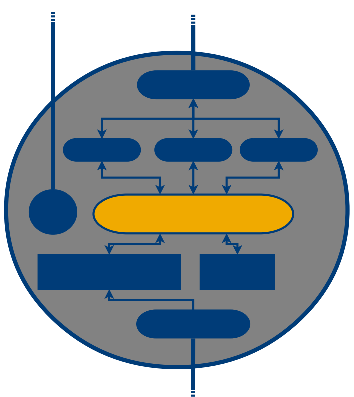
6.4 DATA CONTAINERS



- Receiver automatically stores all incoming data into primary data buffers
- Secondary data buffers can be used to store values calculated from the primary data
- Both primary and secondary data can be requested through the dispatcher
- Both primary and secondary data can be used as an input for the analysis functions

6.5 ANALYSIS FUNCTIONS

Nodes in the MIDAS framework have two types of analysis functions: **metrics** and **processes**

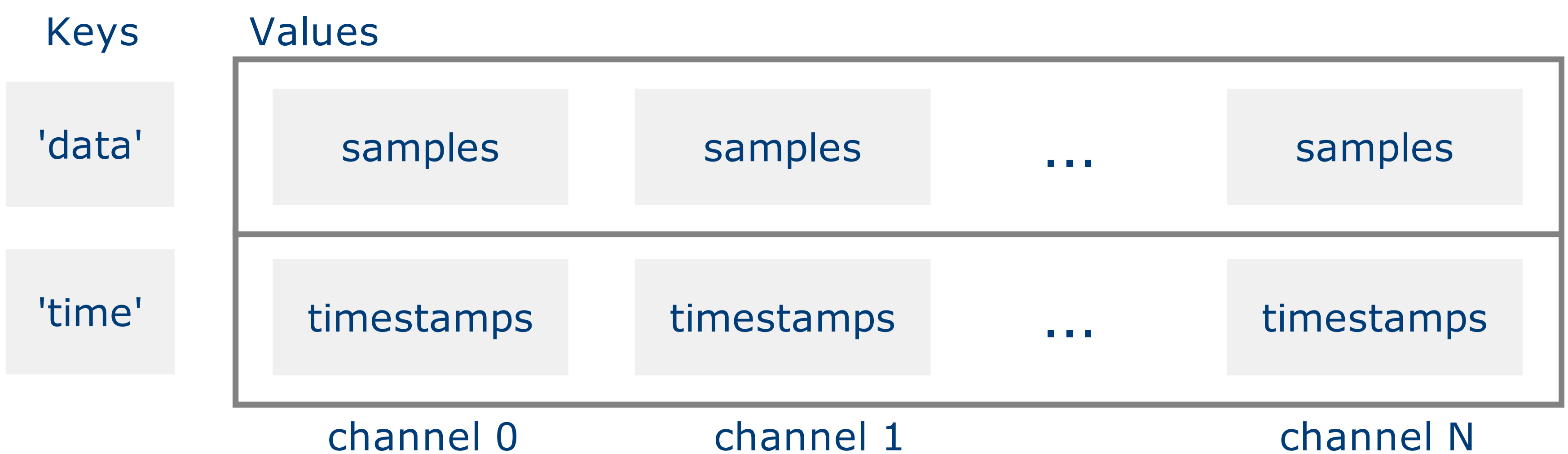


6.6 METRIC FUNCTION

- y = f(x)
- Calculates 'a metric' using data as input
- Input data is specified when the request is made by the client
- Return value can be a scalar or a vector/array
- Can be a method of the Node-class or an external function

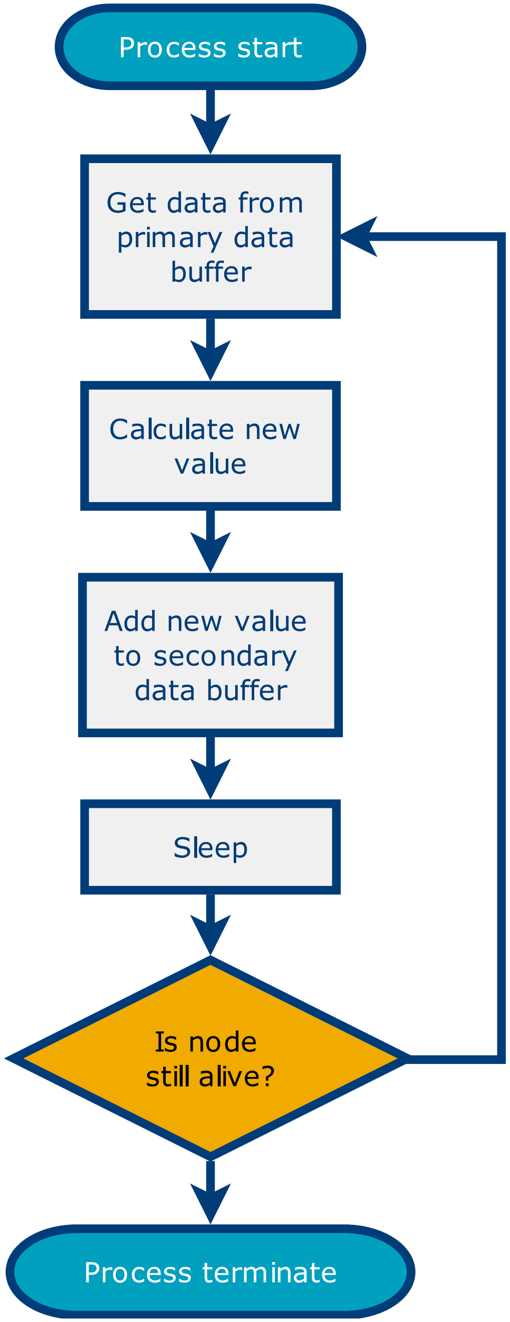
6.6 METRIC FUNCTION

- The first positional argument of the metric function always contains the data
- The incoming data is wrapped inside a dict:



6.7 PROCESS

- Automatically calculate values from the primary data at set intervals
- Runs in a separate process
- Pushes calculated values into secondary data buffer
- Must be a method of the Node-class


Process start
Get data from primary data buffer
Calculate new value
Add new value to secondary data buffer
Sleep
Is node still alive?
Process terminate

6.8 OUR ECG-NODE SPECIFICATIONS

Goal: Be able to calculate average heart rate of N last seconds

1. **Process** for automatically calculating inter-beat intervals
2. **Metric function** for calculating average heart rate from the IBI vector

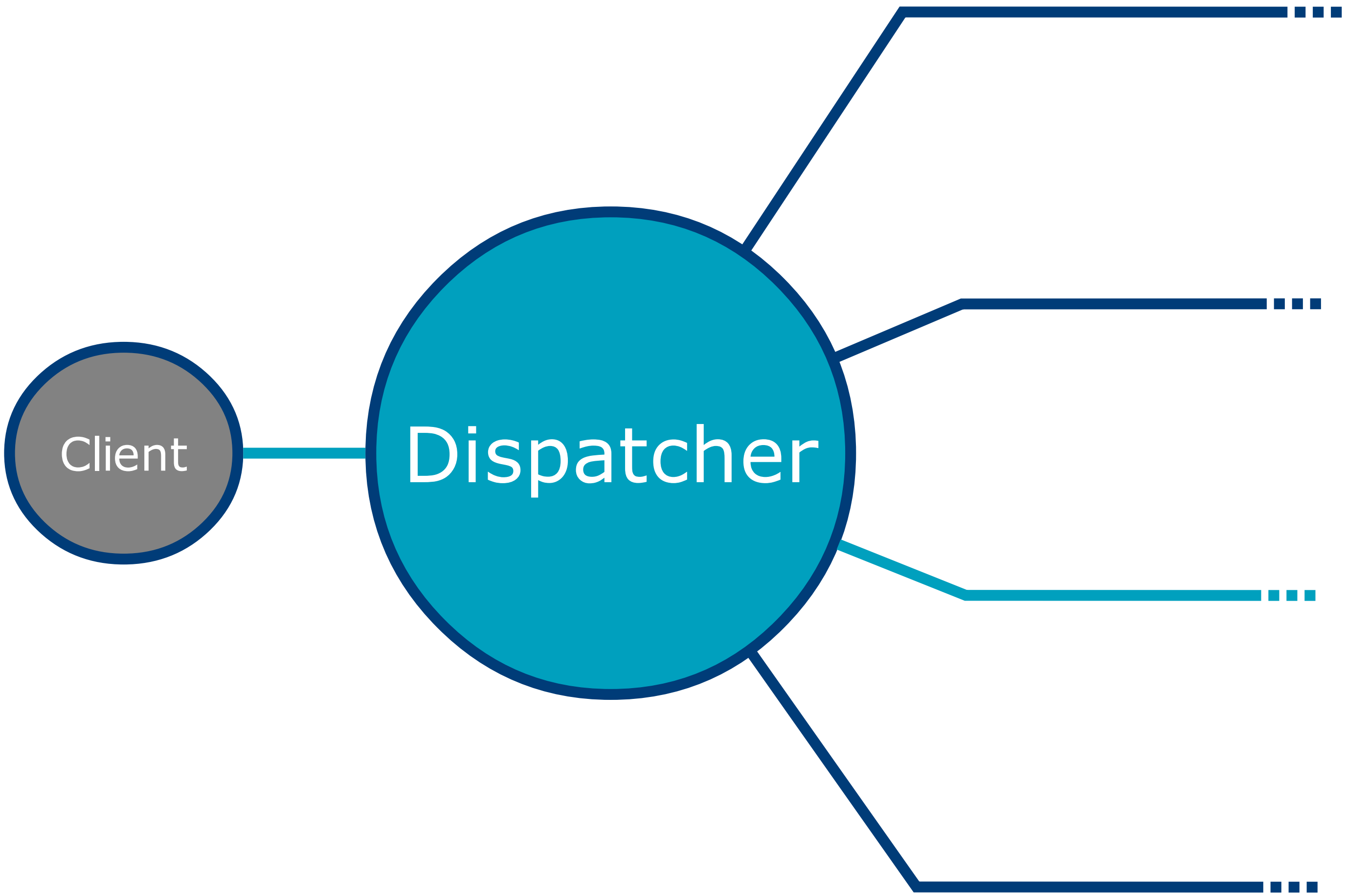
```python
#!/usr/bin/env python3
import sys
from midas.node import BaseNode
import midas.utilities as mu

class ECGNode(BaseNode):
  def __init__(self, *args):
      super().__init__(*args)

      self.metric_functions.append()
      self.process_list.append()

if __name__ == '__main__':
  node = mu.midas_parse_config(ECGNode, sys.argv)

  if node is not None:
      node.start()
      node.show_ui()
```

6.9 DISPATCHER

Photo courtesy Orange County Archives

6.10 DISPATCHER IN A NUTSHELL

The MIDAS dispatcher is basically an HTTP-server, that receives incoming requests from clients and forwards these to the nodes.



6.10 DISPATCHER IN A NUTSHELL

The goal of using the dispatcher is to make the connections in the network easier, since the clients only need to communicate with the dispatcher, and hence only need to be configured with the address of the dispatcher instead of being aware of the entire topology of the MIDAS network.

```
#!/usr/bin/env python3

import sys
from midas import utilities as mu
from midas.dispatcher import Dispatcher

if __name__ == "__main__":
  dp = mu.midas_parse_config(Dispatcher, sys.argv)

if dp is not None:
  dp.start()
```

6.12 MIDAS API

- Clients can access metrics functions and data through the dispatcher using HTTP-requests
- Returned values are wrapped in JSON
- Full API available here: **bit.ly/1AgKbqh**

127.0.0.1:8080/status/nodes


```
{
"ecgnode" : {
"address" : "tcp://10.40.3.53:5021",
"id" : "01",
"name" : "ecgnode",
"status" : "online",
"type" : "ECG"
}
}
```


```
HTTP-ET 127.0.0.1:8080/ecgnode/status/data
```


```
{
"ECG" : "raw ECG",
"ibi" : "interbeat intervals"
}
```

6.13 PUTTING IT ALL TOGETHER

- Dispatcher and nodes must be configured before the network is started
- Easiest to do using configuration files
- Follows standard INI-file structure

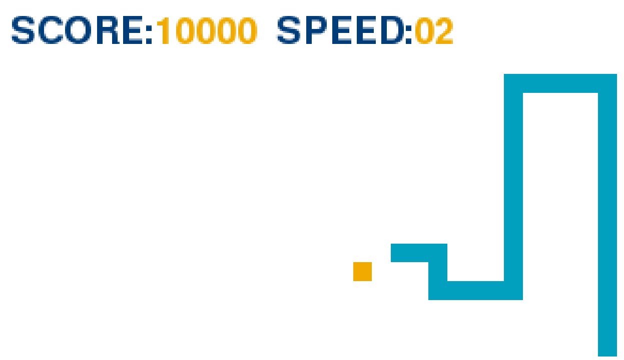
```
[dispatcher]
port                    = 8080
node_list               = ecgnode
ip                      = localhost
n_threads               = 5
run_pubsub_proxy        = False

[ecg]
nodename                    = ecgnode
nodetype                    = ECG
nodeid                      = 01
nodedesc                    = Real-time IBI node
primary_node                = True
port_frontend               = 5021
port_backend                = 5022
port_publisher              = 5023
run_publisher               = False
n_workers                   = 3

n_channels                  = 1
channel_names               = ECG
channel_descriptions        = raw ECG
sampling_rate               = 256
buffer_size_s               = 30
lsl_stream_name             = ecg_data
secondary_data              = True
n_channels_secondary        = 1
buffer_size_secondary       = 30
channel_names_secondary             = ibi
channel_descriptions_secondary      = IBI vector
```

FULL LIST OF CONFIGURATION OPTIONS

github.com/bwrc/midas/wiki/Configuring-the-node
github.com/bwrc/midas/wiki/Configuring-the-dispatcher

6.14 USING MIDAS

Our toy example: Control the speed of a Snake-game using the heart rate of the player.



All we need to do now is to add a periodical request for...

127.0.0.1:8080/ecgnode/metric/median_bpm:ibi/5

...and map the return value to frame rate.

6.15 ASSUMPTIONS

1. Heart rate will be between 40 - 120 beats-per-minute
2. Snake speed should be between 10 - 50 frames-per-second

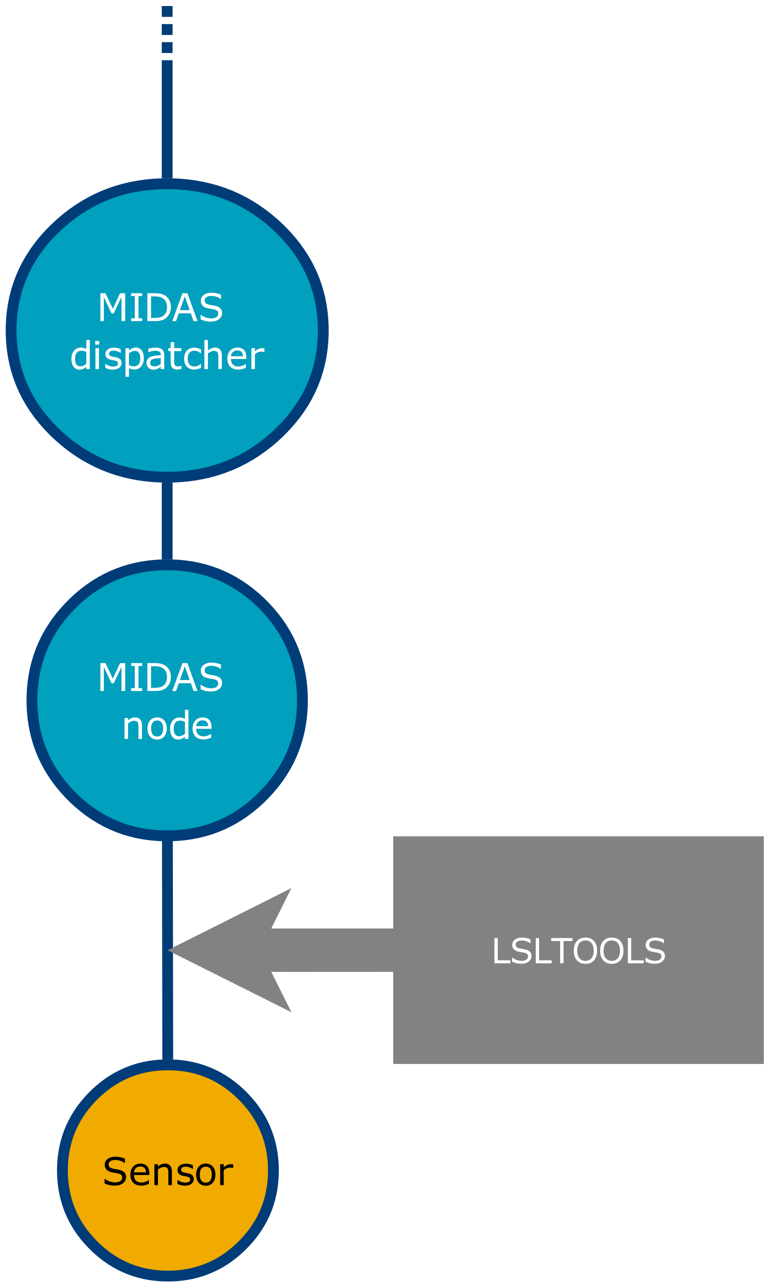
$$fr = \frac{bpm - bpm_{min}}{bpm_{max} - bpm_{min}}(fr_{max} - fr_{min}) + fr_{min}$$

$$fr = \frac{bpm}{2} - 10$$

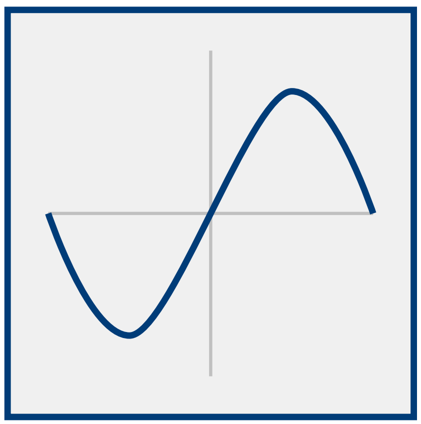
7. SUPPORTING TOOLS

lsltools-package is geared for working with the lab streaming layer

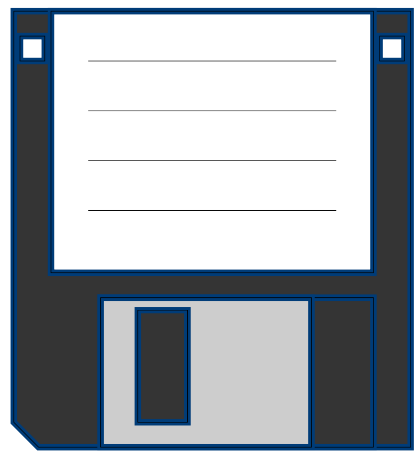
7.1 LSLTOOLS



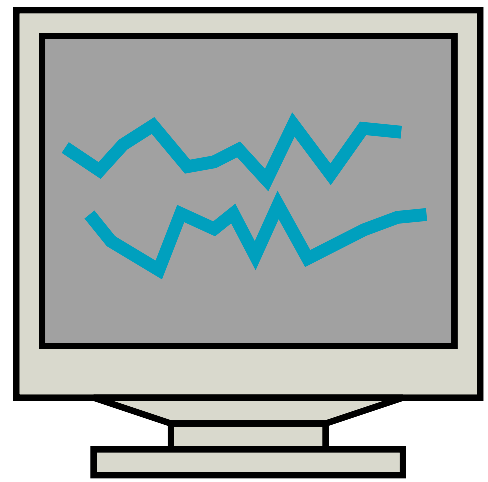
7.2 CONTENTS



Simulation



Recording



Visualization

8. CONTRIBUTING

9. FUTURE

Mobile Applications: Move experiments out of the lab

Internet of Things: Utilize wearable health technology

10. Q & A