OpenRVDAS

An Open Source Framework for
Building Data Acquisition Systems

David Pablo Cohn
openrvdas.org
oceandata.tools

RVTEC 2019

(A lot of this is going to be familiar to folks who were at INMARTECH last year)

Why OpenRVDAS?

- Many ships each running homebrew derivatives of legacy systems (*dsLog*, *LDS* and others)

- Massive duplication of effort to support

- Common, open source codebase would allow pooling expertise and best practices

- MIT License allows unrestricted use/copying/modification/distribution/sublicensing for commercial/non-commercial purposes

An **framework**, not a **system**

(Systems change as requirements change; a good framework lets you easily put together whatever system meets current requirements)

Read from serial port, prefix with timestamp and instrument id, write to file

Everyone’s needs are different **now**
Everyone’s needs will be different in 5 years

Solution: small set of Lego-like components that can be easily “snapped together” to create what you need

Read from serial port, prefix with timestamp and instrument id, write to file

Requirements

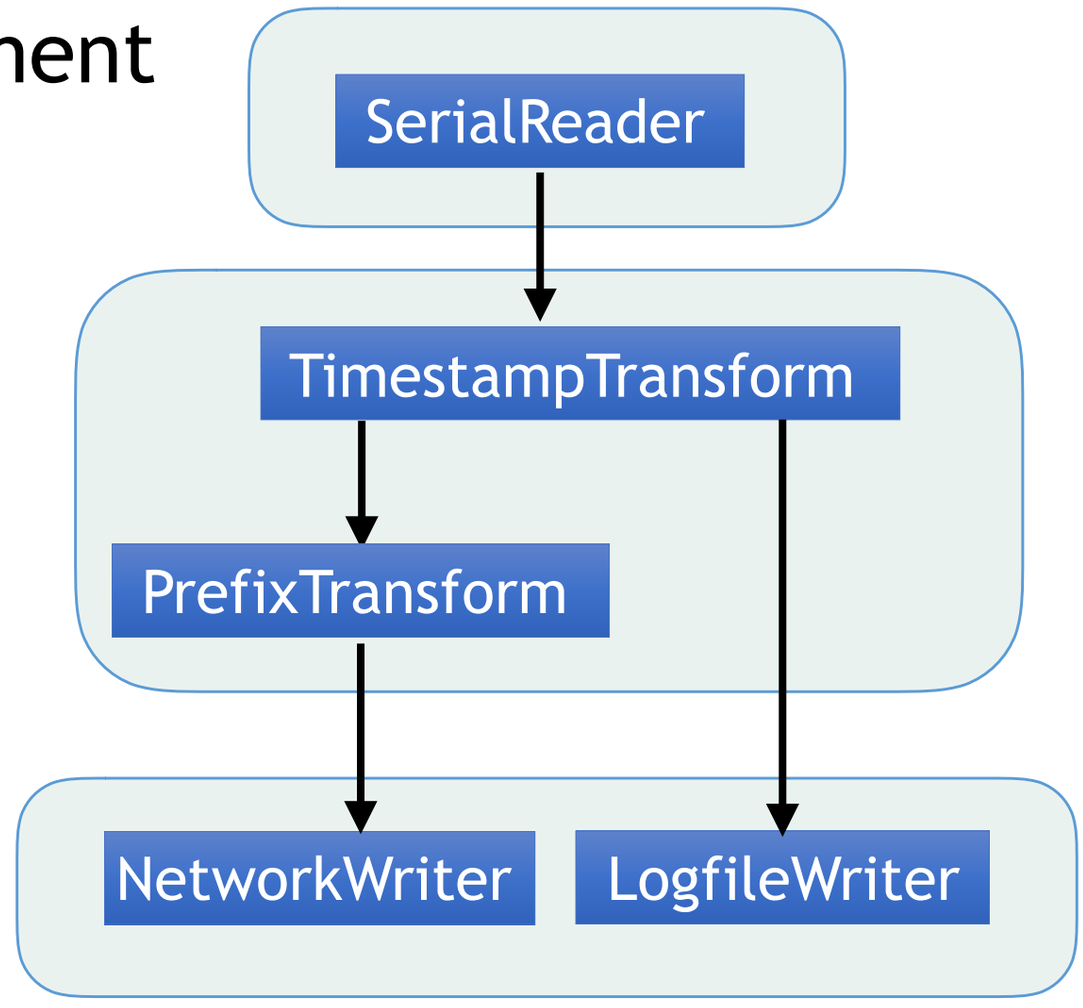
- Python 3.6+
- Should run on all POSIX-compliant systems
- Yes, it will run on a Raspberry Pi Zero
- Installation scripts available for CentOS, RedHat, Ubuntu (and kind of MacOS)

Outline

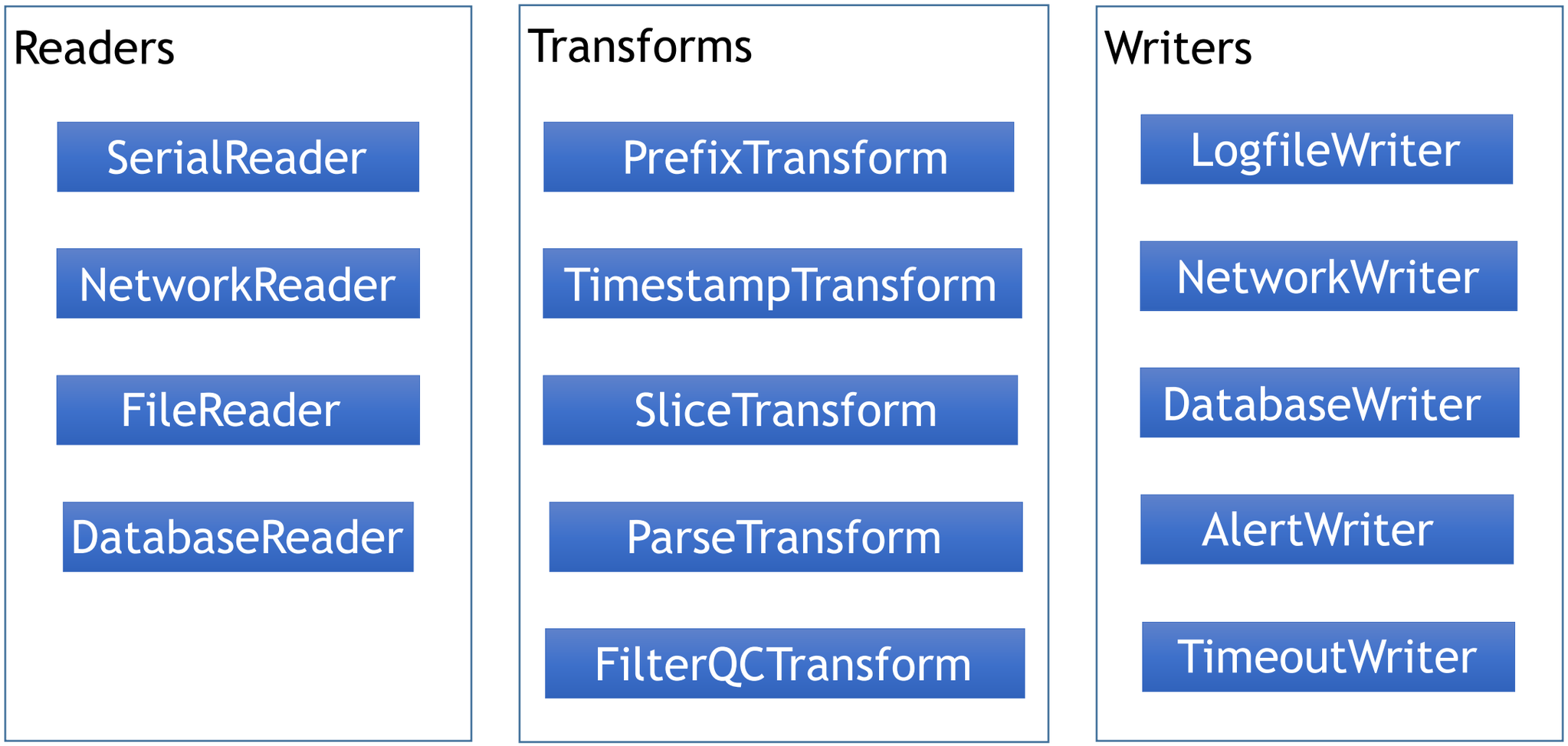
- Building loggers out of components
- Running and controlling loggers
- Displaying and manipulating data from loggers
- What needs to be done next

Loggers - basic unit of data acquisition

1. **Read** data from an instrument
   - serial port
   - network
2. (Optionally) **transform** it
   - timestamp
   - parse
3. **Write** it to storage
   - file
   - network
   - database



Readers, Transforms and Writers



Simple API makes it easy to create your own as needs arise

Three easy ways to combine

In Code

```
reader = SerialReader(port='/dev/ttys1')
transform = TimestampTransform()
writer = LogfileWriter(filebase='/var/logs/knud')
while True:
  in_record = reader.read()
  out_record = transform.transform(in_record)
  writer.write(out_record)
```

Command line

```
listener.py --serial port=/dev/ttys1 \
    --transform_timestamp \
    --transform_prefix knud \
    --write_logfile /var/logs/knud \
    --write_network :6221 \
    --write_database rvdas@openrvdas:test
```

Config file

```
{"knud->net": {
   "name": "knud->net",
    "readers": {
      "class": "SerialReader",
        "kwargs": {"port": "/dev/ttys1",
                   "baudrate": 9600
                  }
    ...
```

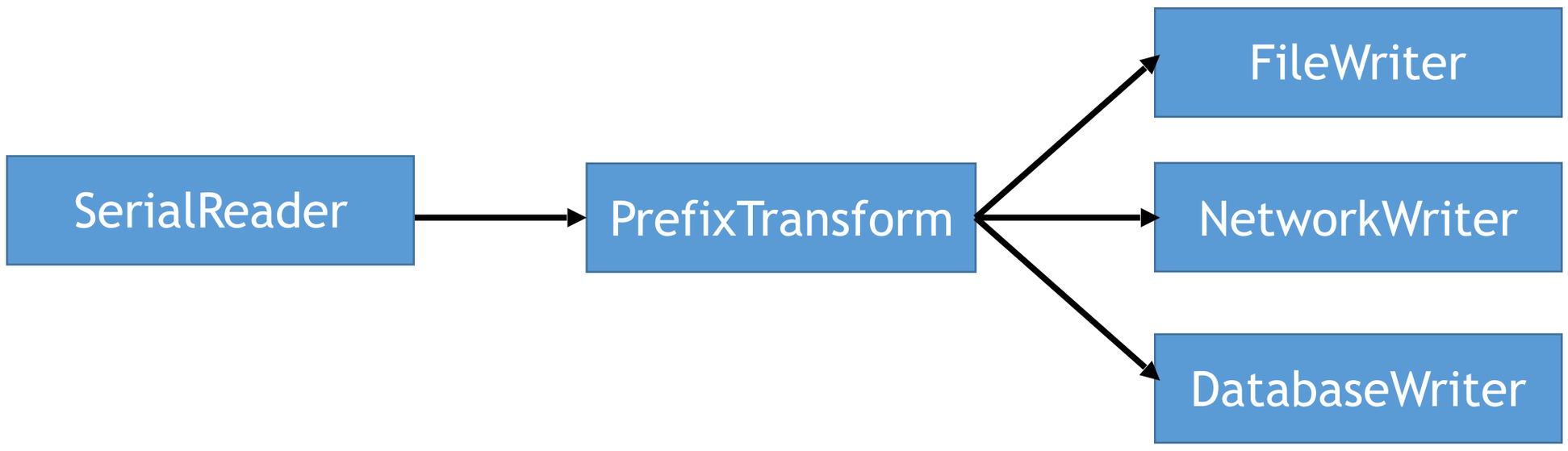
Python code API

```
reader = SerialReader(port='/dev/ttys1')
transform = TimestampTransform()
writer = LogfileWriter(filebase='/var/logs/knud')
while True:
  in_record = reader.read()
  out_record = transform.transform(in_record)
  writer.write(out_record)
```

API makes it easy to combine existing Readers/ Writers/Transforms (or create your own as needed)

Listeners - a command line interface

```
listener.py --serial port=/dev/ttys1 \
    --transform_timestamp \
    --transform_prefix knud \
    --write_logfile /var/log/knud \
    --write_network :6221 \
    --write_database rvdas@openrvdas:test
```



Loggers from config files

```
listener.py --config_file gyr1.yaml
```

```
readers:
- class: SerialReader
  kwargs: {baudrate: 9600, port: /dev/ttys1}
transforms:
- class: TimestampTransform
- class PrefixTransform
  kwargs: {prefix: gyr1}
writers:
- class: LogfileWriter
  kwargs: {filebase: /var/tmp/log/gyr1}
- class: NetworkWriter
  kwargs: {network: ':6224'}
```

Loggers from config files

```
listener.py --config_file gyr1.yaml
```

```
readers:
- class: SerialReader
  kwargs: {baudrate: 9600, port: /dev/ttys1}
transforms:
- class: TimestampTransform
- class PrefixTransform
  kwargs: {prefix: gyr1}
writers:
- class: LogfileWriter
  kwargs: {filebase: /var/tmp/log/gyr1}
- class: NetworkWriter
  kwargs: {network: ':6224'}
stderr_writers:
- class: LogfileWriter
  kwargs: {filebase: /var/tmp/log/stderr}
```

Outline

- Building loggers out of components
- Running and controlling loggers
- Displaying and manipulating data from loggers
- What needs to be done next

Multiple loggers: the Logger Runner

```
logger_runner.py --config sample_config.json
```

```
s330->net:
  name: s330->net
  readers:  ...
  transforms: ...
  writers: ...
mwx1->net:
  name: mwx1->net
  readers:  ...
  transforms: ...
  writers: ...
eng1->net:
  name: eng1->net
  readers:  ...
  transforms: ...
  writers: ...
...
```

Cruise control: the Logger Manager

Frequently have set of common modes that a collection of loggers should be in (e.g. "off", "port", "underway")

Logger manager script allows users to

- Switch between modes
- Monitor, enable/disable individual loggers

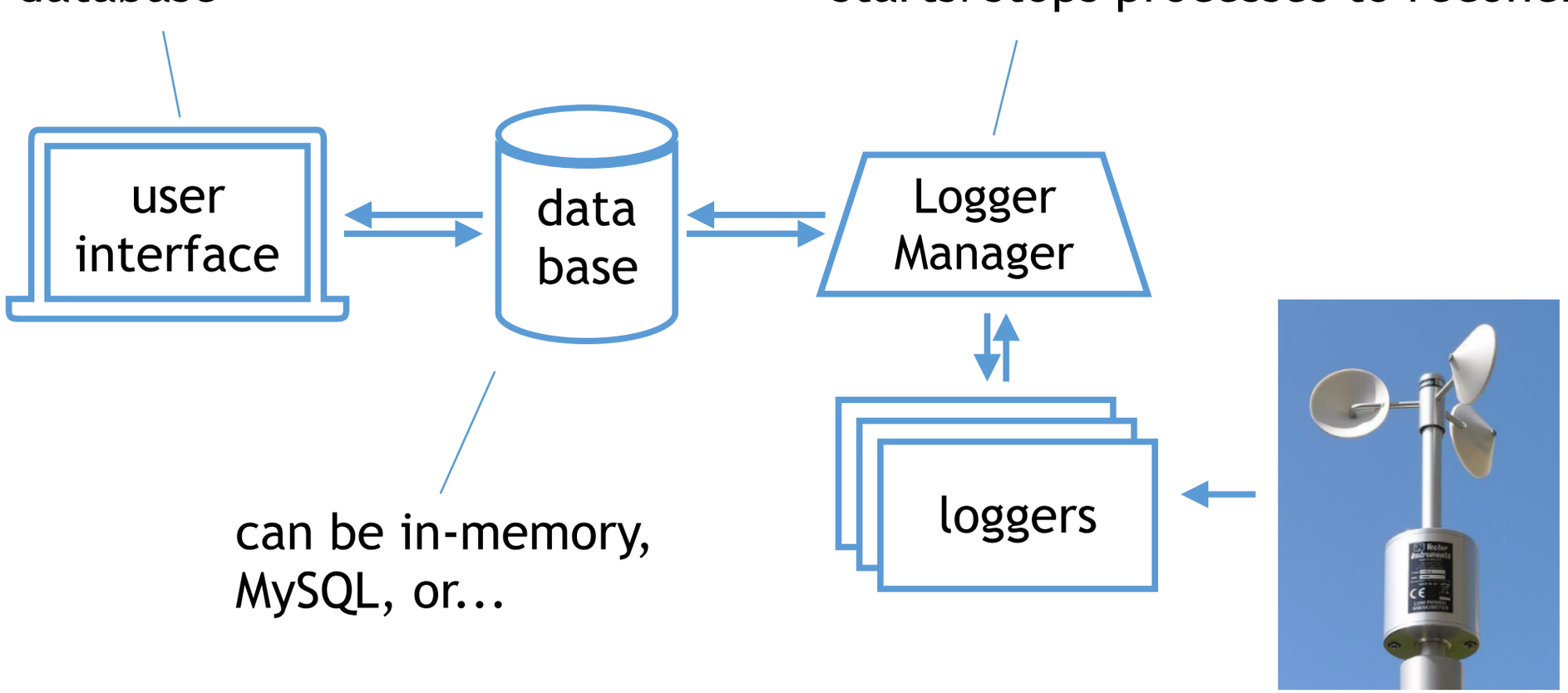
Command line interface

```
openrvdas> server/logger_manager.py

command? load_configuration NBP1406_cruise.yaml
command? get_modes
Available Modes: off, monitor, log, log+db

command? set_active_mode underway
command? get_loggers
Loggers: PCOD, cwnc, gp02, gyr1, adcp, eng1, svp1,
twnc, mbdp, knud, grv1, mwx1, pco2, pguv, s330, tsg1,
rtmp, hdas, tsg2, seap, true_wind, subsample

command? get_logger_configs s330
Configs for s330: s330->off, s330->net, s330->file/net,
s330->file/net/db

command? set_active_logger_config s330 s330->off
command? quit
```

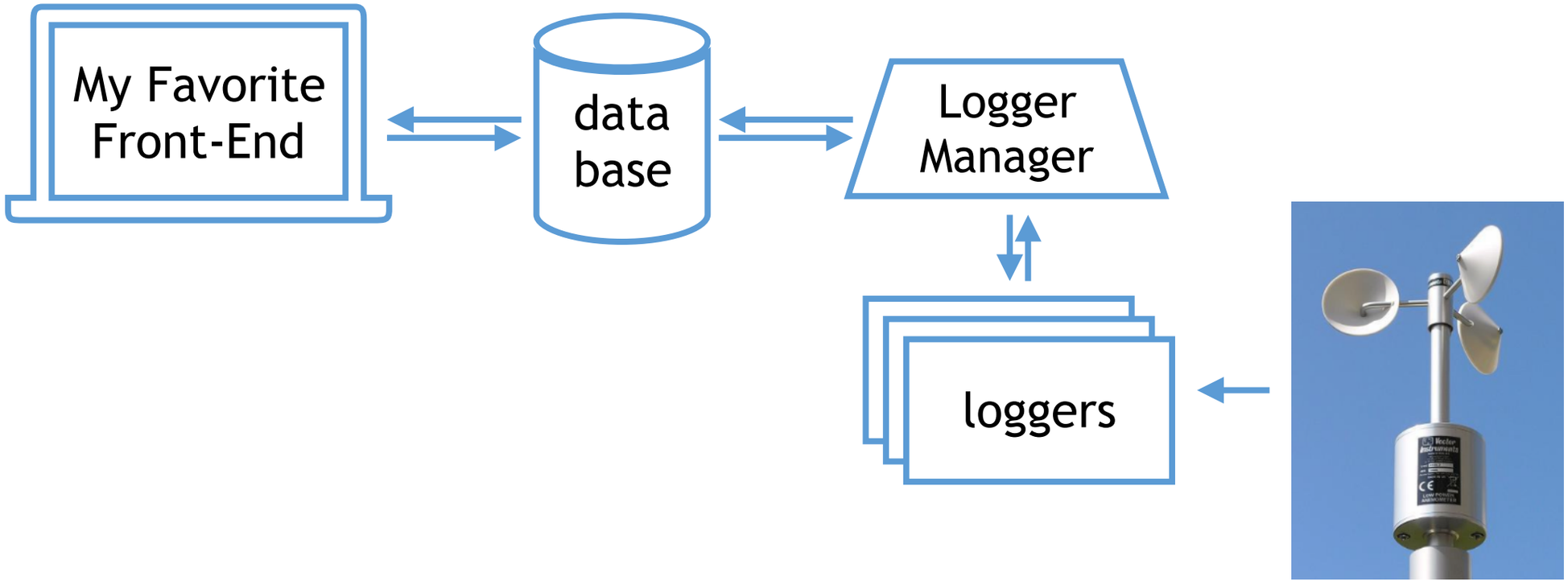
Control architecture – a database approach

- writes desired state to database
- retrieves latest observed status from database

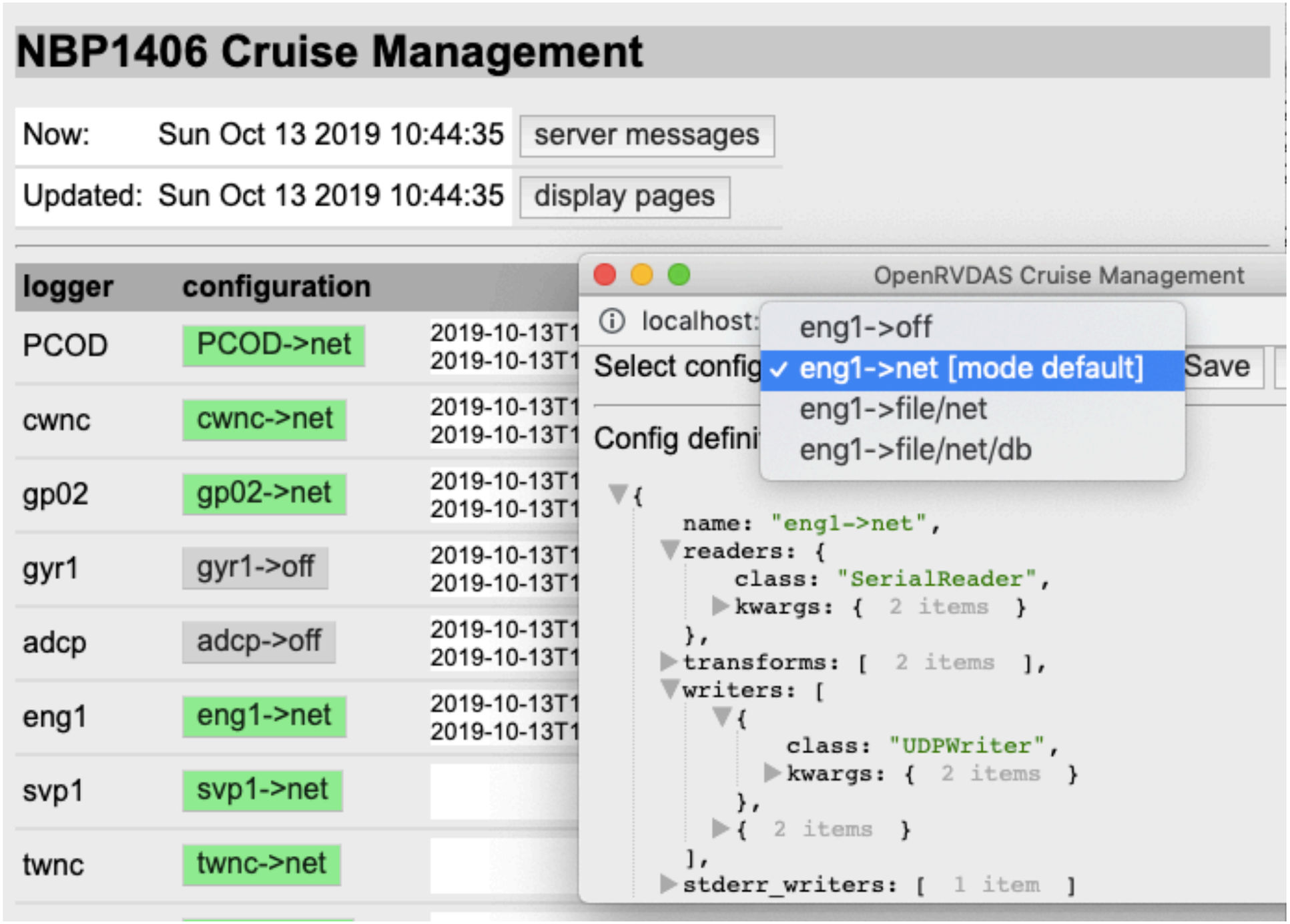
- reads desired state from database
- checks observed state from system
- starts/stops processes to reconcile



API allows creating interface to your favorite database and/or favorite front-end system



Django-based web interface



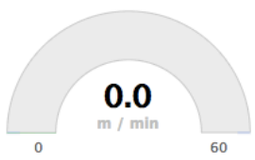
Outline

- Building loggers out of components
- Running and controlling loggers
- Displaying and manipulating data from loggers
- What needs to be done next

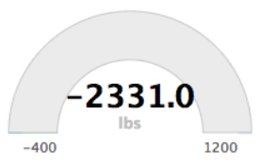
Displaying and Manipulating Data

Port Wind Dir	339° M	Speed	
Stbd Wind Dir	334° M	Speed	
Lat	22.0099° S	Lon	
Heading	217.93° T	SOG	
Pitch	-1.3°	Roll	
RTemp	21.7942° C	Air Temp	
Eng Voltage	12.26V		
Aq Room Flow	507.5		
Helo Deck Flow	565.7		
Hydro Lab Flow	233.8		

Waterfall Winch Speed



Tension

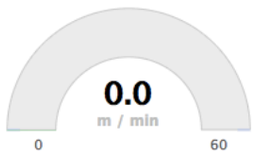


Payout: 0 m

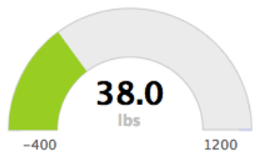
Warnings

2018-09-21T1
2018-09-21T1
2018-09-21T1
2018-09-21T1
2018-09-21T1
2018-09-21T1
2018-09-21T1

Trawl Winch Speed



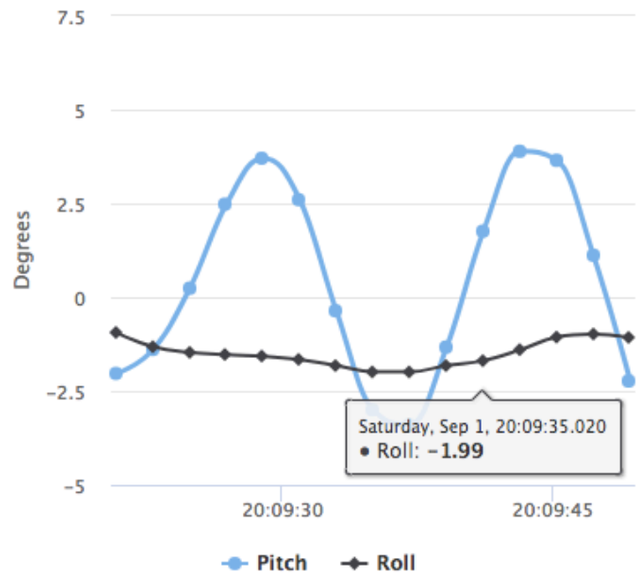
Tension



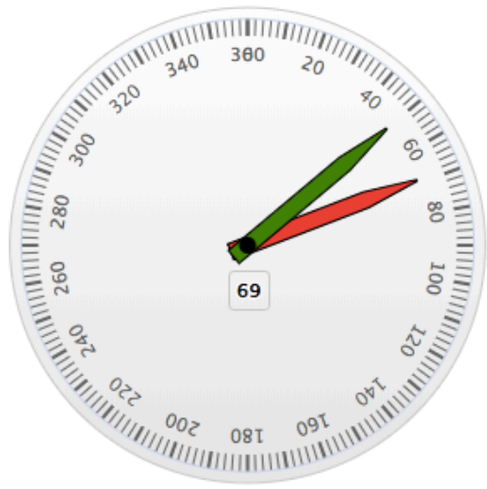
Payout: -34

Warnings

Pitch, Roll



Port Relative Wind, Stbd Relative Wind



Attitude: Pitch -2.21°, Roll -1.08°

Relative Speed: Port: 11.68 m/s, Stbd: 10.45 m/s

Position: Lat 62.1508°S, Lon 57.6808°W

The Cached Data Server ("CDS")

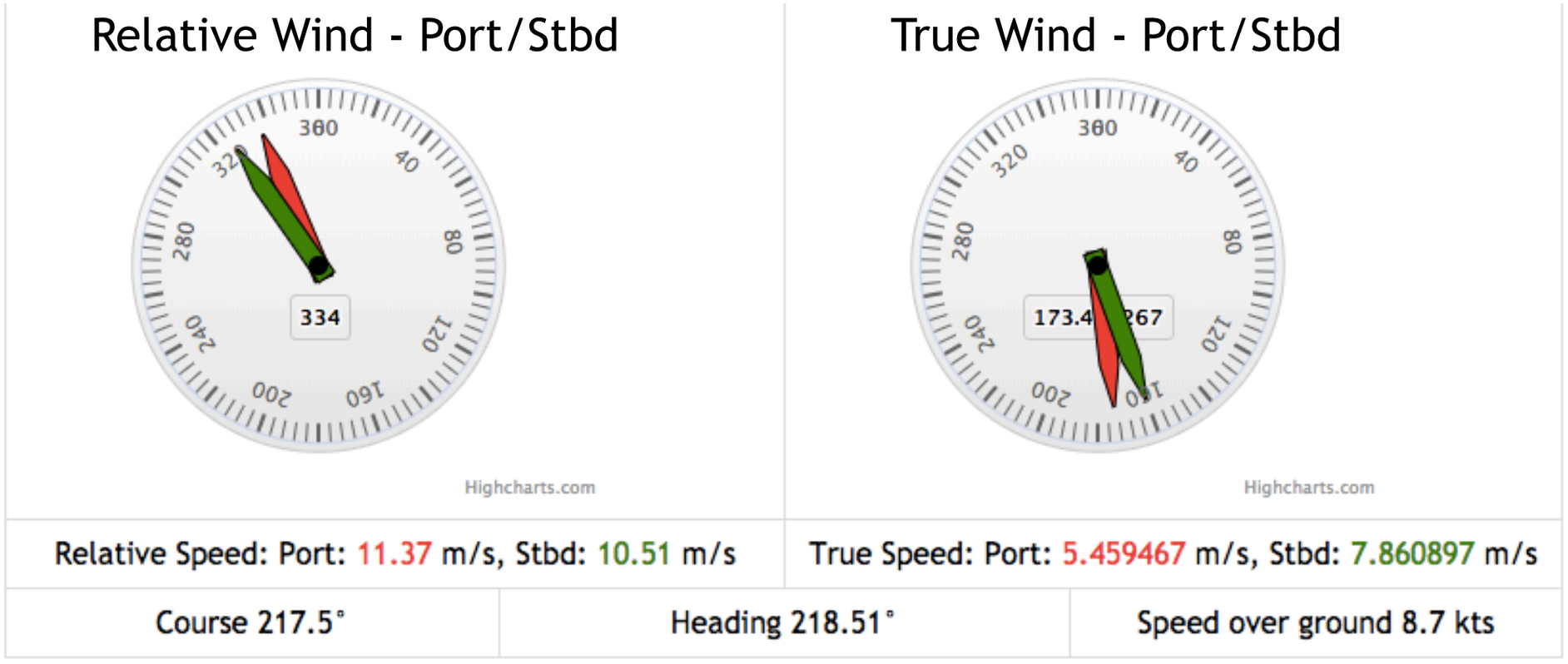
- A simple but specialized pub-sub server
  - communicates via websockets

- Loggers can write data to it
  (via CachedDataWriter)

- Loggers can read data from it
  (via CachedDataReader)

Using the CDS for derived values

Read inputs from server, compute values,
inject outputs back into server

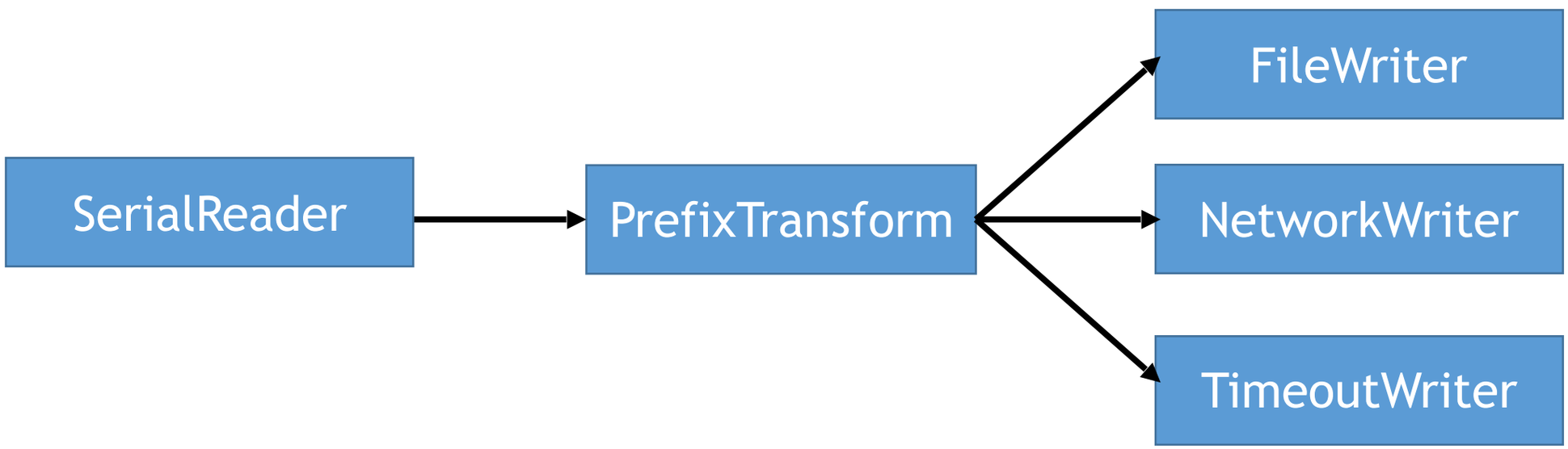


Using the CDS for quality control

```
readers:
- class: CachedDataReader
  kwargs:
    data_server: localhost:8766
    subscription:
      fields:
       TWNCTension: {seconds: 0}
       TWNCPayout: {seconds: 0}
transforms:
- class: QCFilterTransform
  kwargs:
    bounds: TWNCTension:-150:10000,TWNCPayout:-60:175000
writers:
- class: AlertWriter
- class: LogfileWriter
  kwargs:
    filebase: /var/log/openrvdas/winch_errors
```

Using the CDS for quality control

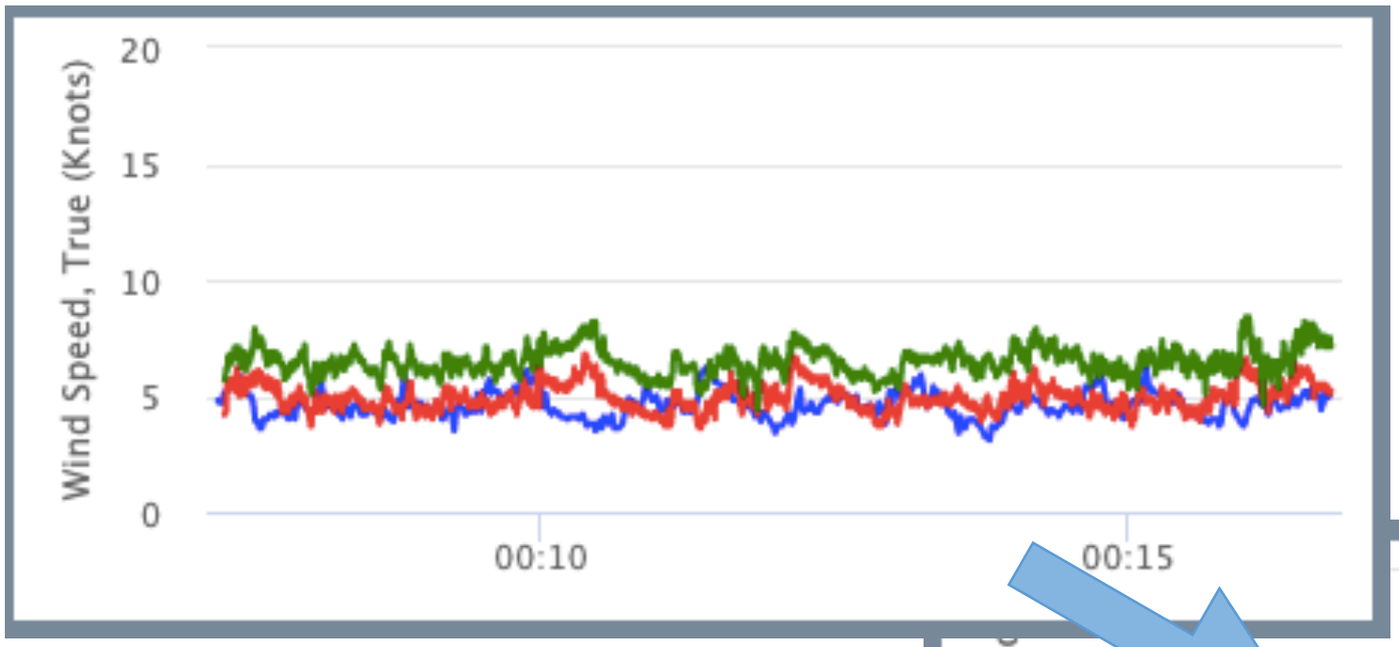
```
- class: TimeoutWriter
  kwargs:
    timeout: 60
    message: No Gyro data received for 60 seconds
    resume_message: Gyro data has resumed
    writer:
    - class: LogfileWriter
      kwargs:
        filebase: /var/log/openrvdas/winch_errors
```

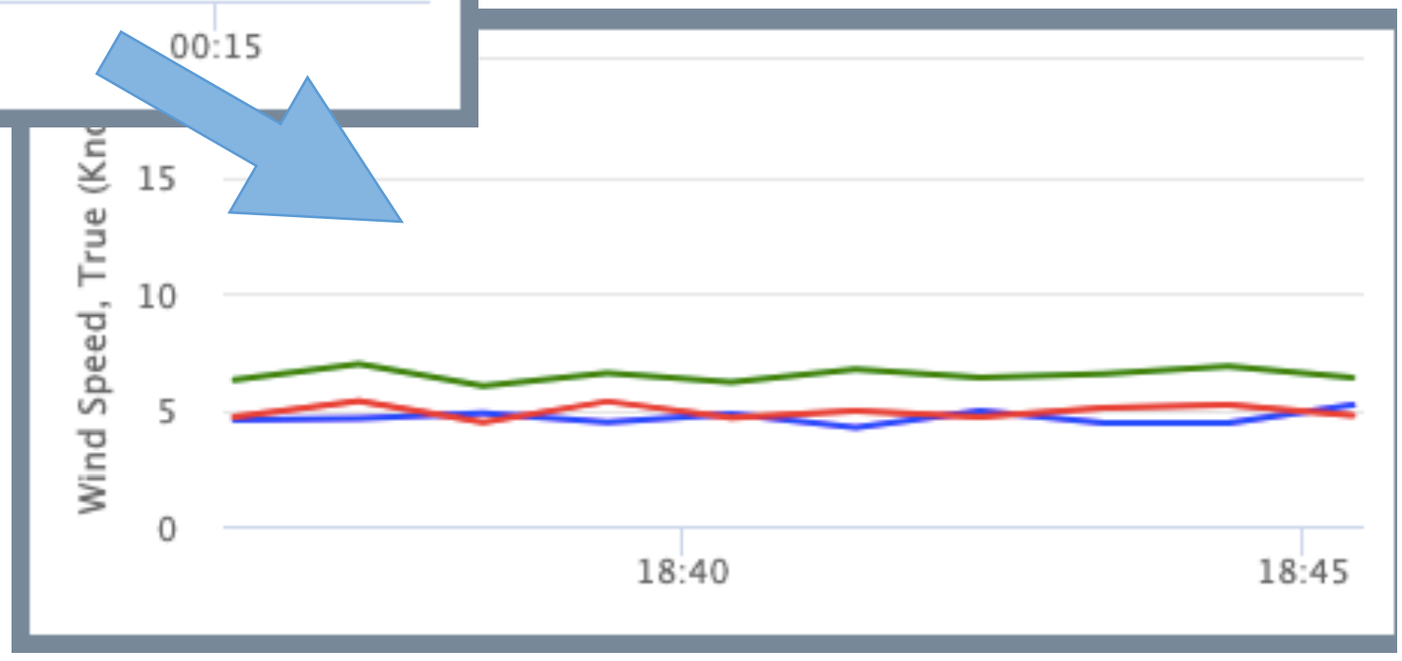


Using the CDS for smoothing

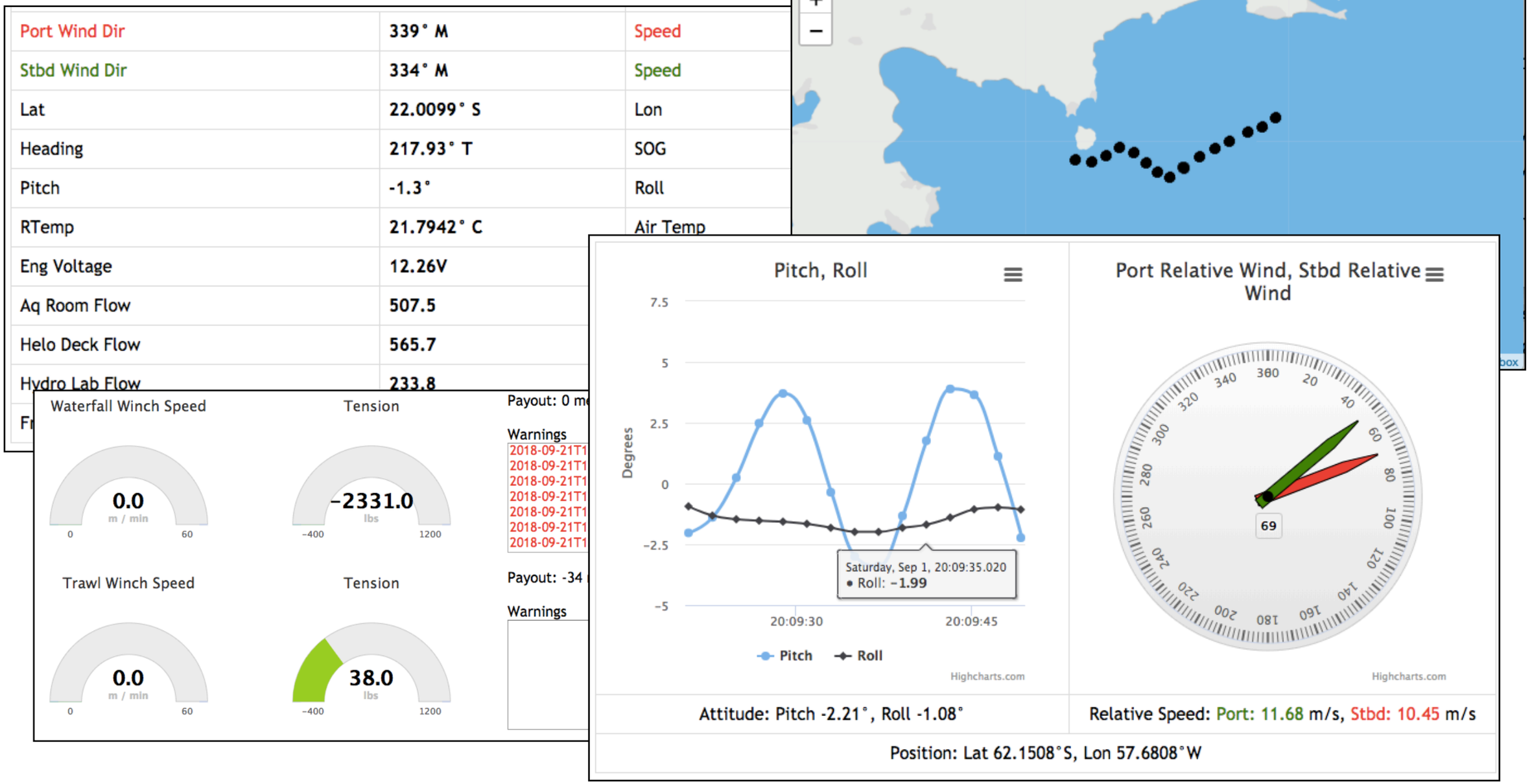
```
- class: SubsampleTransform
  kwargs:
    field_spec:
      wind_mast_port_true_speed_knots:
        output: avg_wind_mast_port_true_speed_knots
        subsample:
          type: boxcar_average
          window: 60
          interval: 60
    wind_mast_stbd_true_speed_knots:
      output: avg_wind_mast_stbd_true_speed_knots
      subsample:
        type: boxcar_average
        window: 60
        interval: 60
```

Using the CDS for smoothing



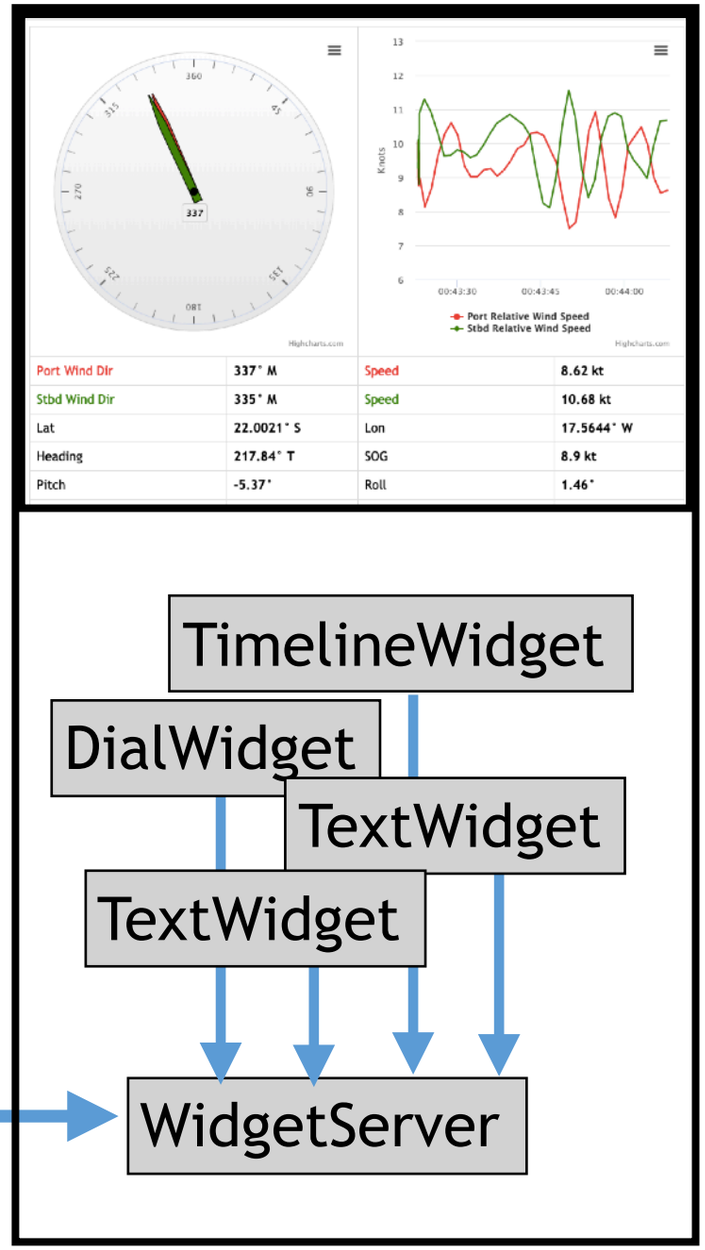


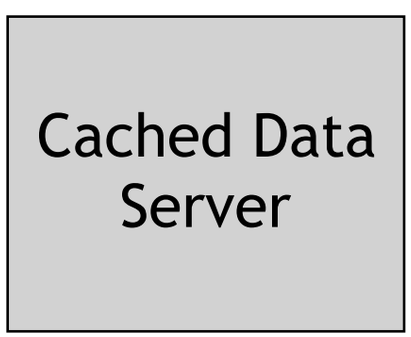
Javascript-based display widgets



Display widgets

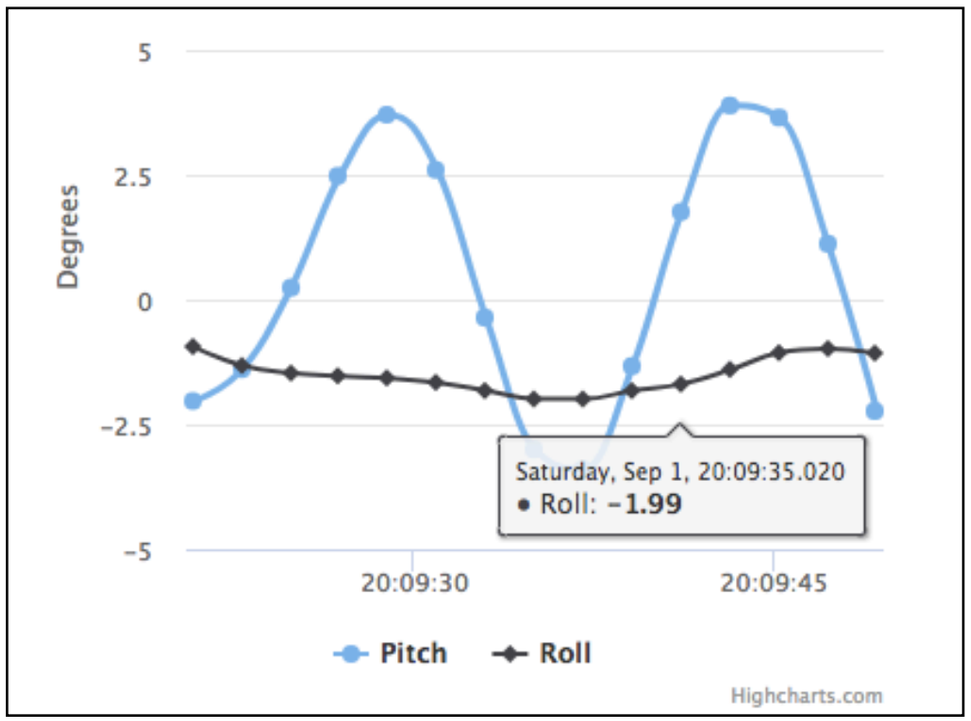
- Embed widgets on arbitrary web page
- Widget Server code aggregates all widget subscriptions from a page, open single websocket to CDS
- Simple API makes creation/ integration of new widget types easy





Display widgets

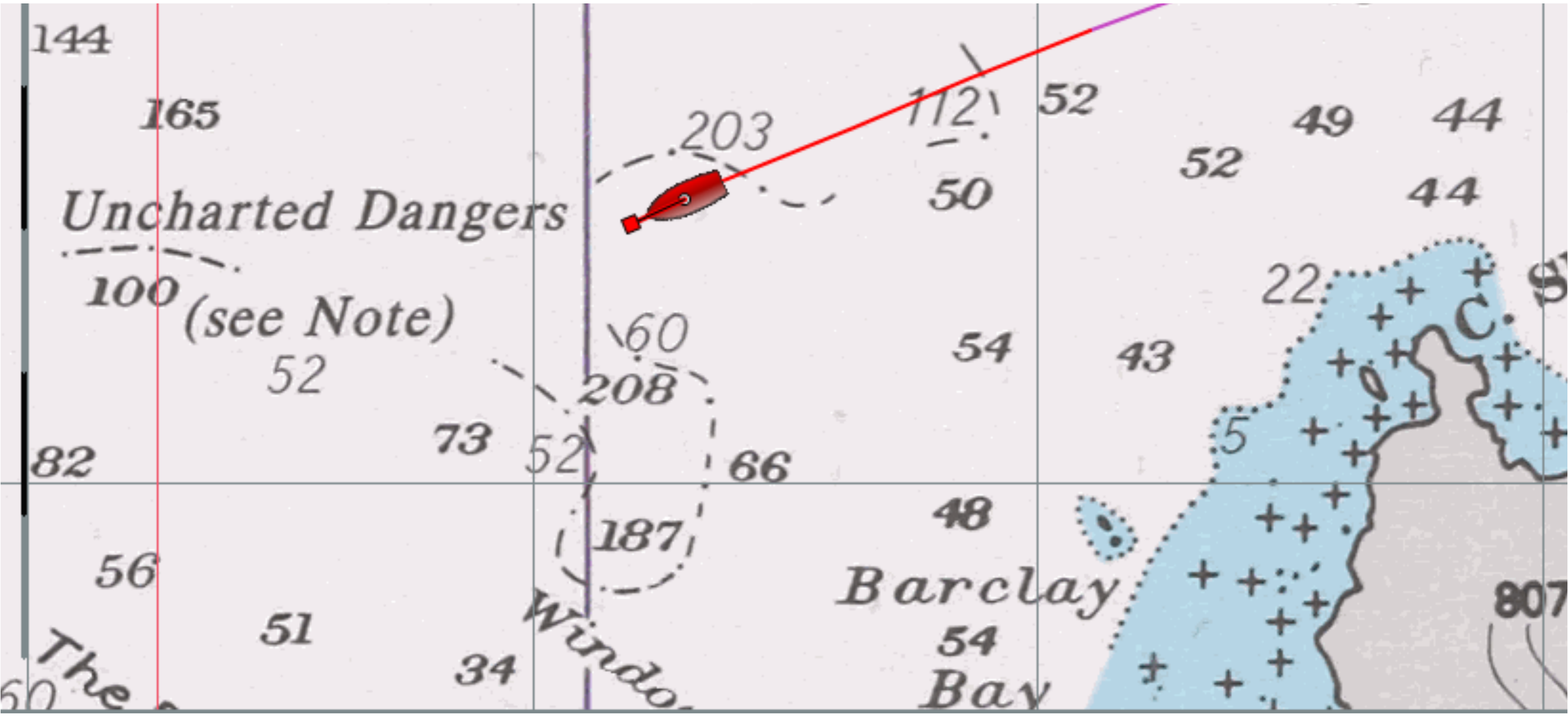
```
<div id="line-container"></div>

<script type="text/javascript">
  var line_fields = {
    S330Pitch: {
      name: "Pitch",
      seconds: 30
    },
    S330Roll: {
      name: "Roll",
      seconds: 30
    }
  };
  var tl_widget = new TimelineWidget('line-container',
                                     line_fields, 'Degrees'));
  var widget_server = new WidgetServer([tl_widget],
                                       'localhost:8766');
  widget_server.serve();
</script>
```



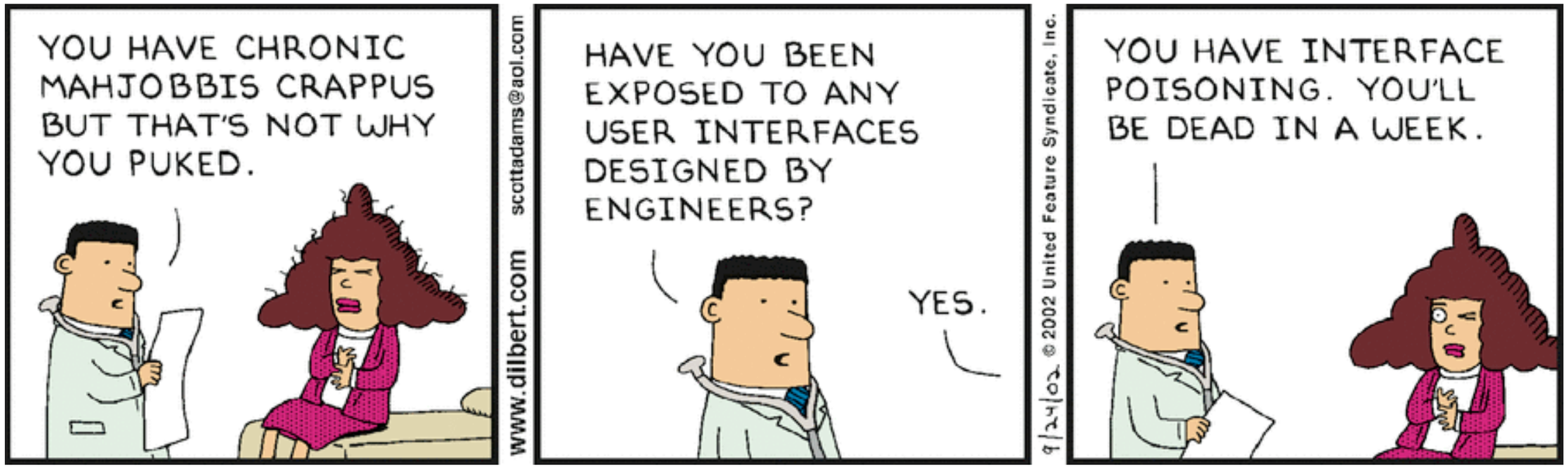
Outline

- Building loggers out of components
- Running and controlling loggers
- Displaying and manipulating data from loggers
- What needs to be done next

OpenRVDAS issues, projects and milestones at
https://github.com/oceandatatools/openrvdas
for a complete list of where we're going

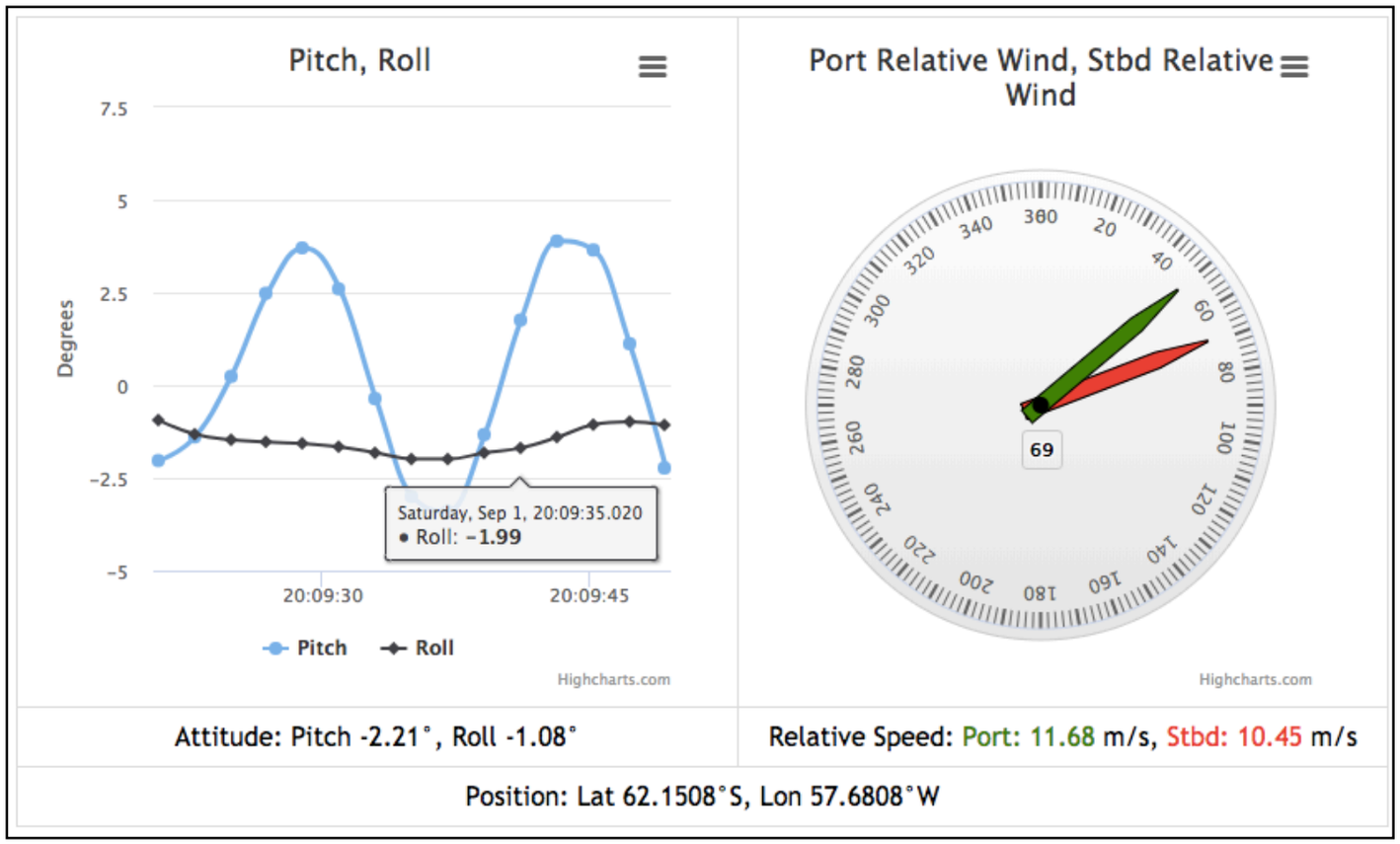


Better control UX



Still using the original, vaguely-appalling GUI

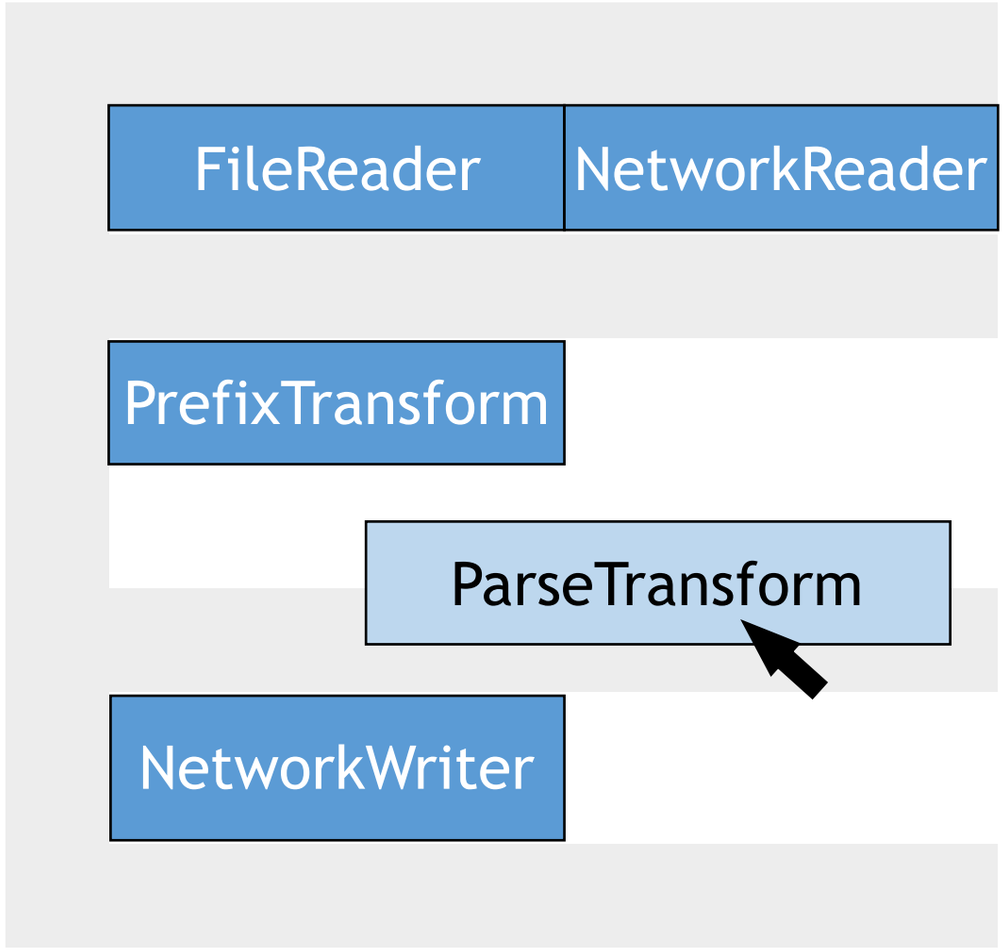
Open source display widgets



Timeline and *Dial* widgets are based on Highcharts; offers a free license for academic institutions, but still a proprietary solution.

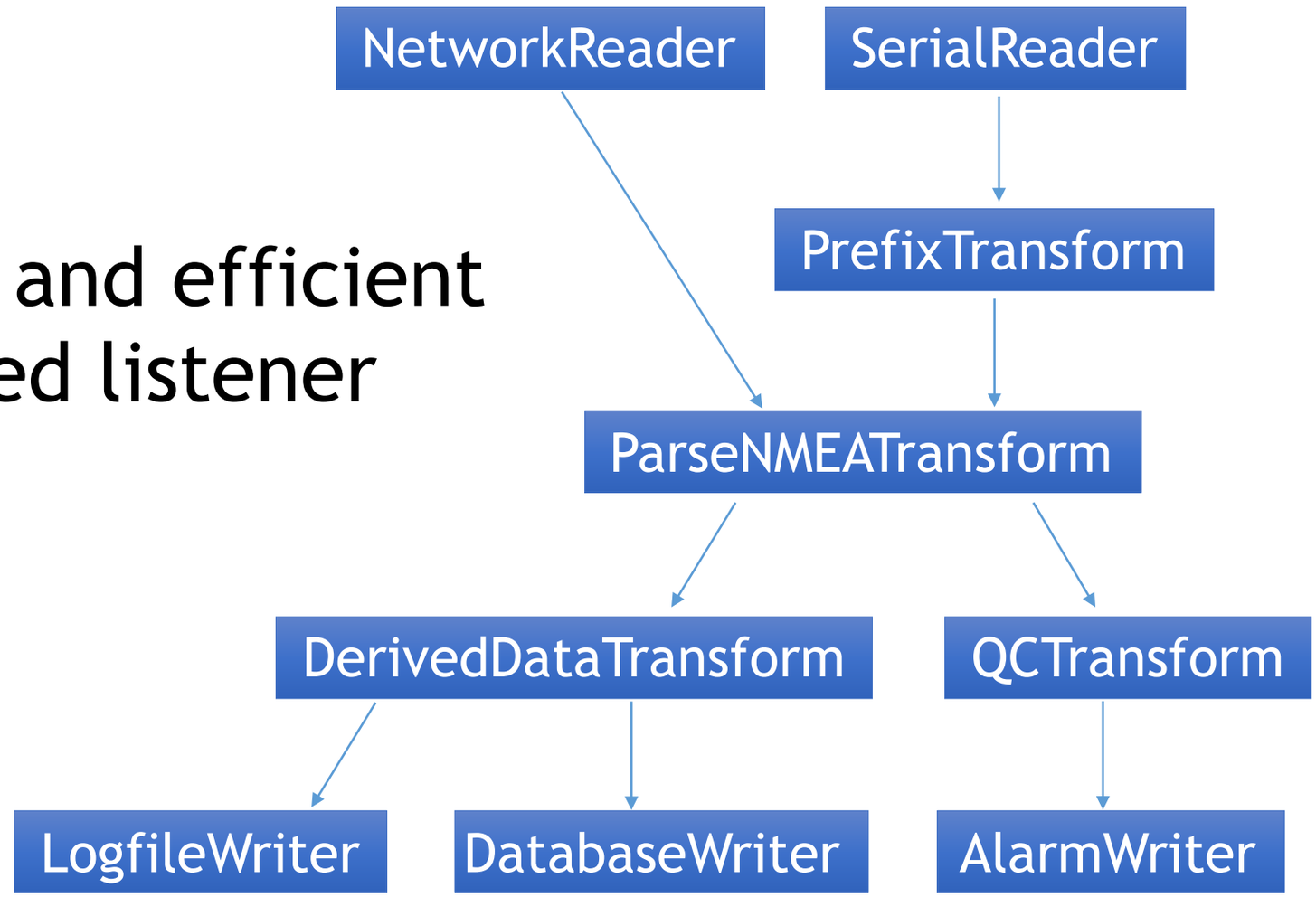
Better ways to compose Readers/Transforms/Writers

Maybe a Scratch-like visual interface?

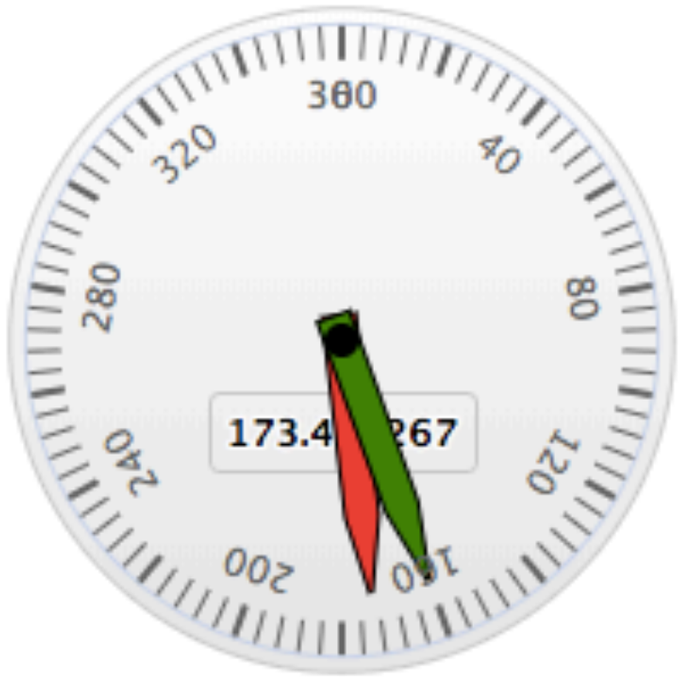


Dataflow-based Listener

More flexible and efficient dataflow-based listener



GUI-based display creation



(More) sea trials!

For more information

http://openrvdas.org
http://github.com/oceandatatools/openrvdas
david.cohn@openrvdas.org

Many thanks to

OCEANX

Oregon State University

UNITED STATES ANTARCTIC PROGRAM
NATIONAL SCIENCE FOUNDATION

UAF
UNIVERSITY OF ALASKA FAIRBANKS