Python and ZeroMQ scale to real time robot control

Justin Bayer

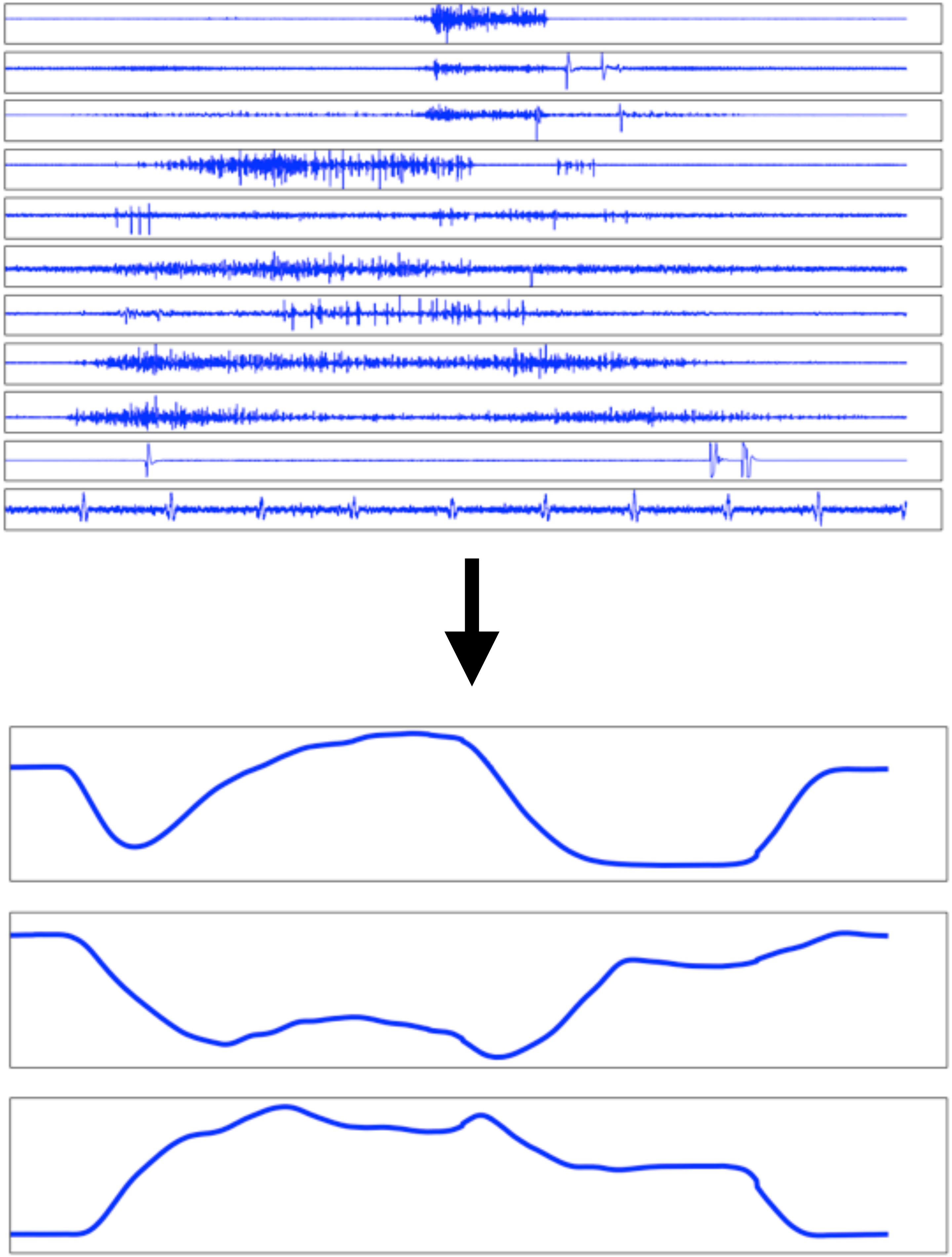
bayer.justin@googlemail.com   github.com/bayerj/

Python and ZeroMQ scale to ~~real time~~ really fast robot control

Justin Bayer

bayer.justin@googlemail.com   github.com/bayerj/

[redacted]



Hey ML guy,
please find f so that

$$\sum_t (\mathrm{position}_t - f(\mathrm{emg}_t))^2$$

is minimal!

Not happening.

“We want the robot to move

similar

to how the human moves.”

Data Challenges

- What is similar? What is human like?
- How far into the past/future do we have to look?
- Noisy data.
- Online Prediction.

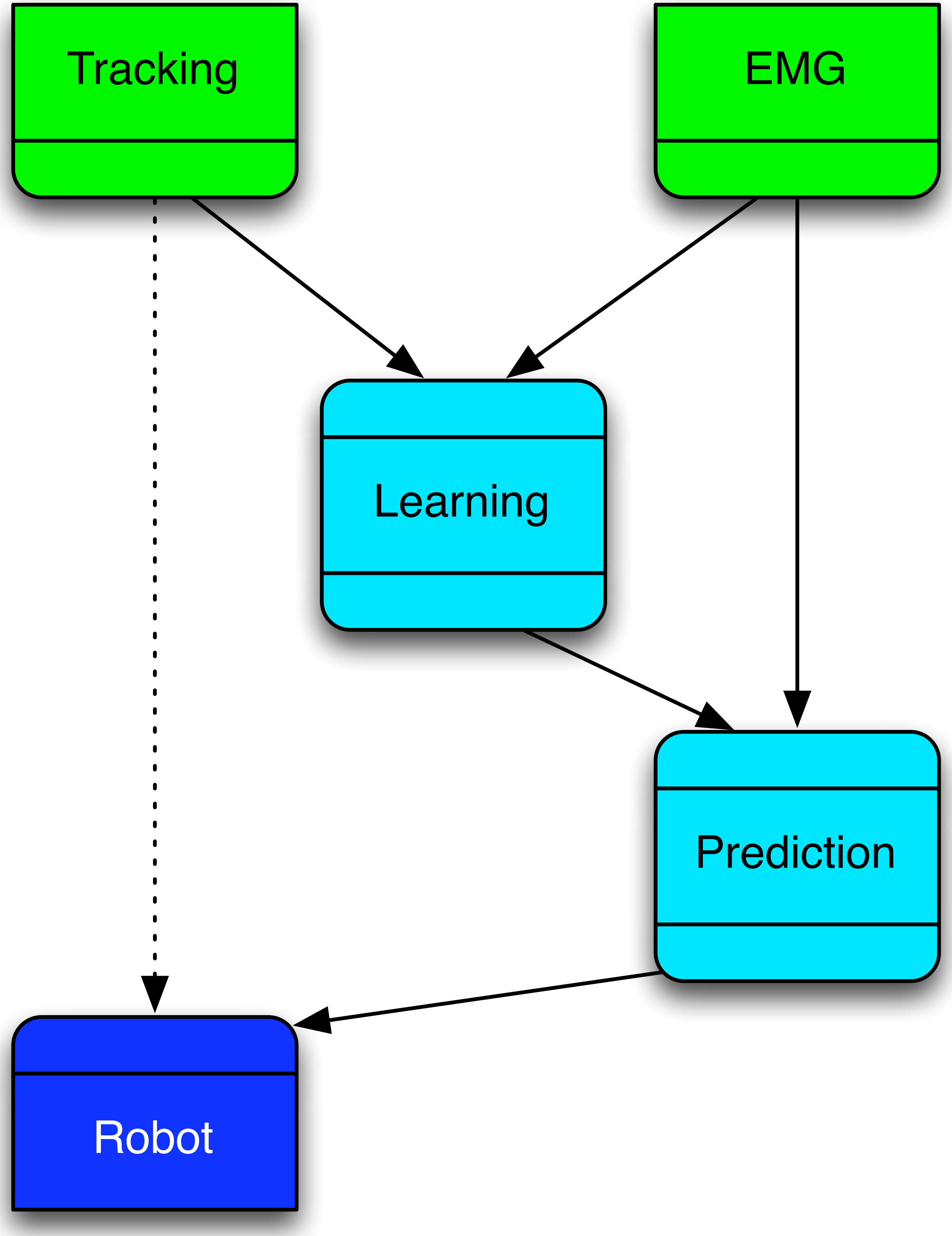
System Challenges

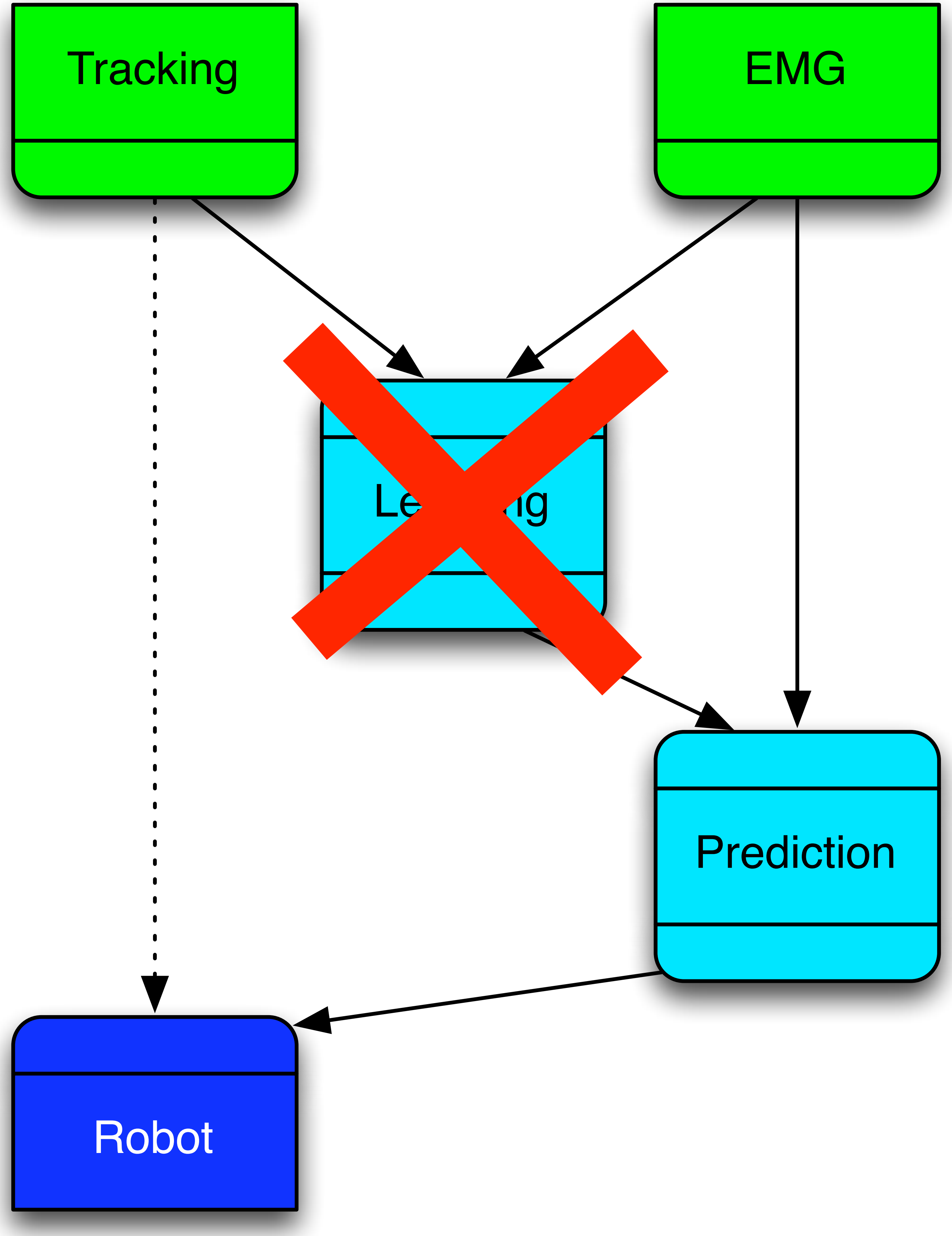
- Many different heterogenous components in soft- and hardware.

- Notorious failure of single ones. (Crash, need for code changes, human failure.)

Requirements

- Fast.
- Low latency.
- Maximally isolated units.
- Short deployment cycle.


Tracking
EMG
Learning
Prediction
Robot



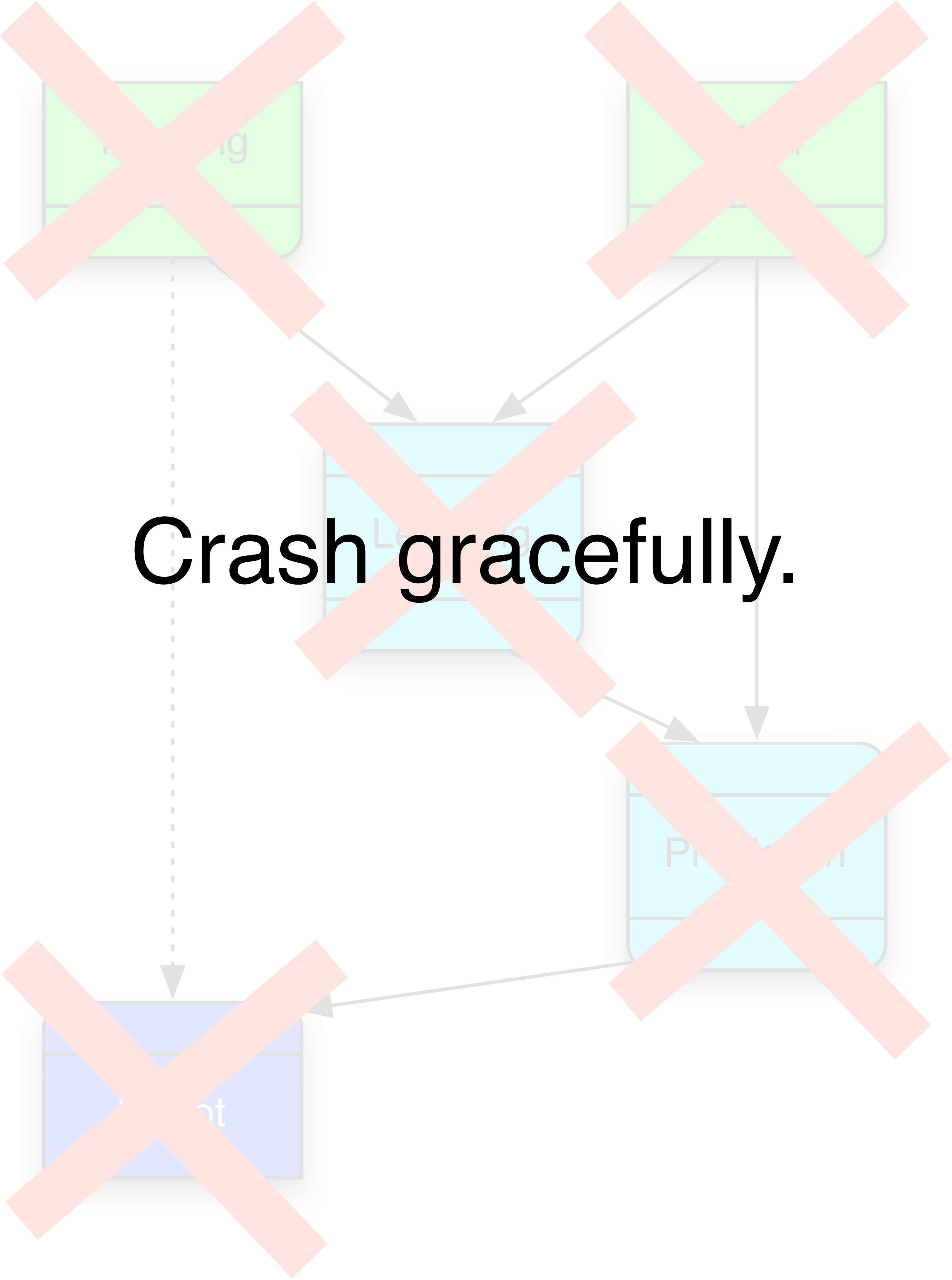
Tracking

EMG

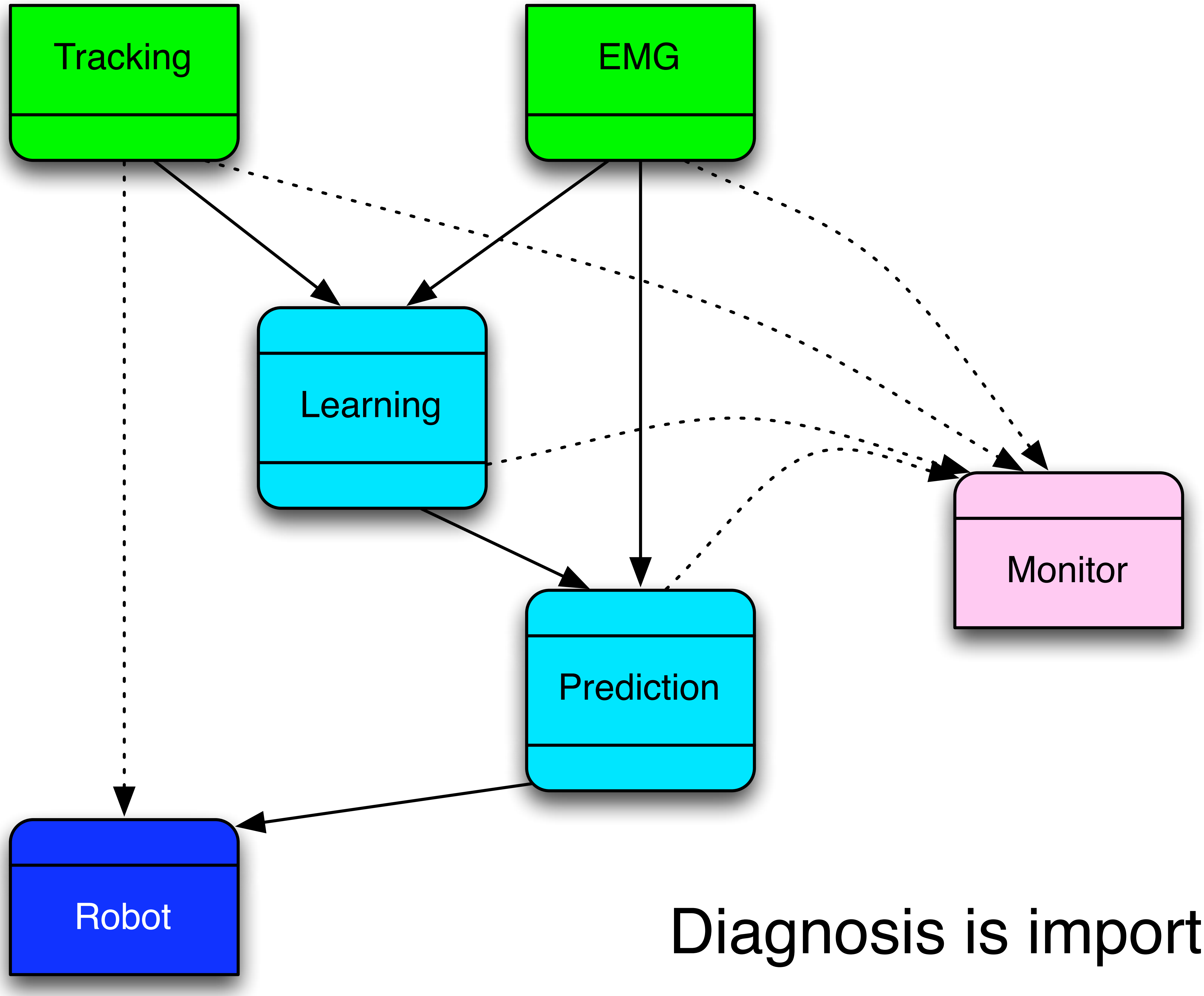
Learning

Prediction

Robot



Crash gracefully.



Diagnosis is important.

Python

- Slow.
- Dangerous.
- Not parallel.

Doesn’t matter.

On slowness wrt prediction

On slowness wrt I/O

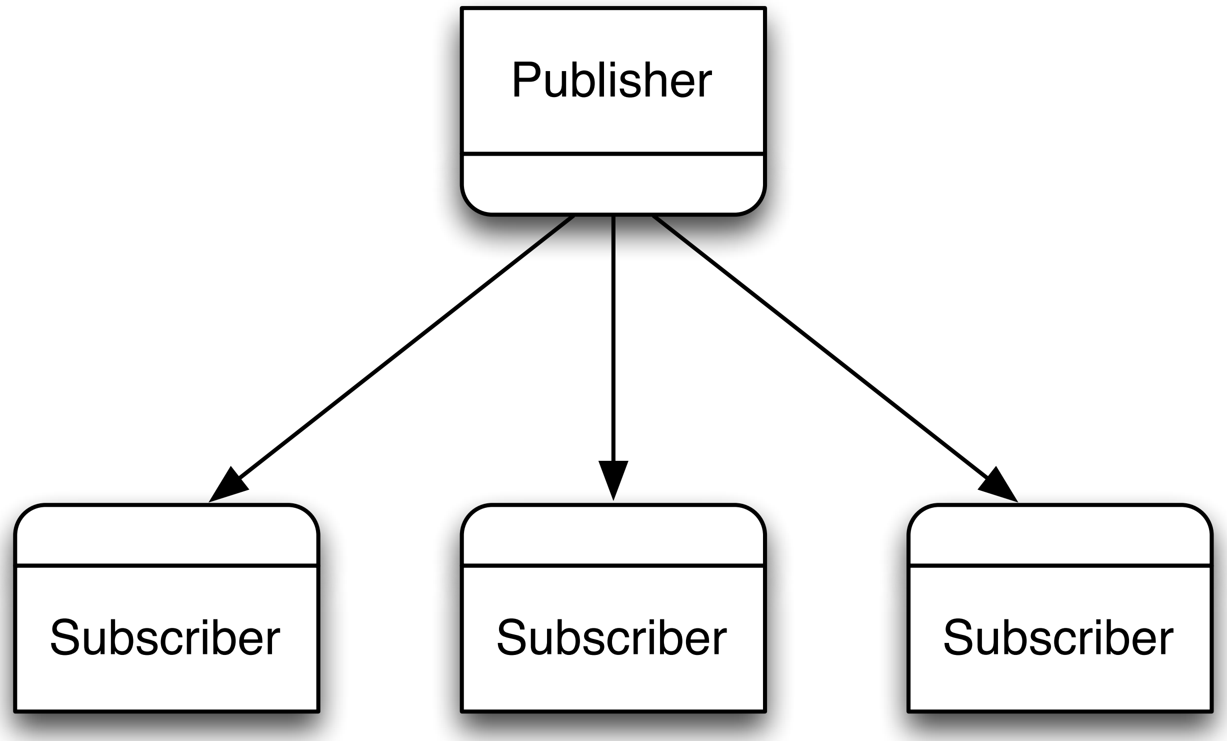
- Send/receive messages at 100Hz.
- Decode messages into machine learning compatible representation.

ZeroMQ

- Alternative to sockets.
- Cross platform and cross language. (40+ programming languages supported.)
- Really fast. (Designed for high frequency trading)
- Minor code adaptions for messages via intra process, inter process, TCP.
- Some neat abstractions for network traffic.

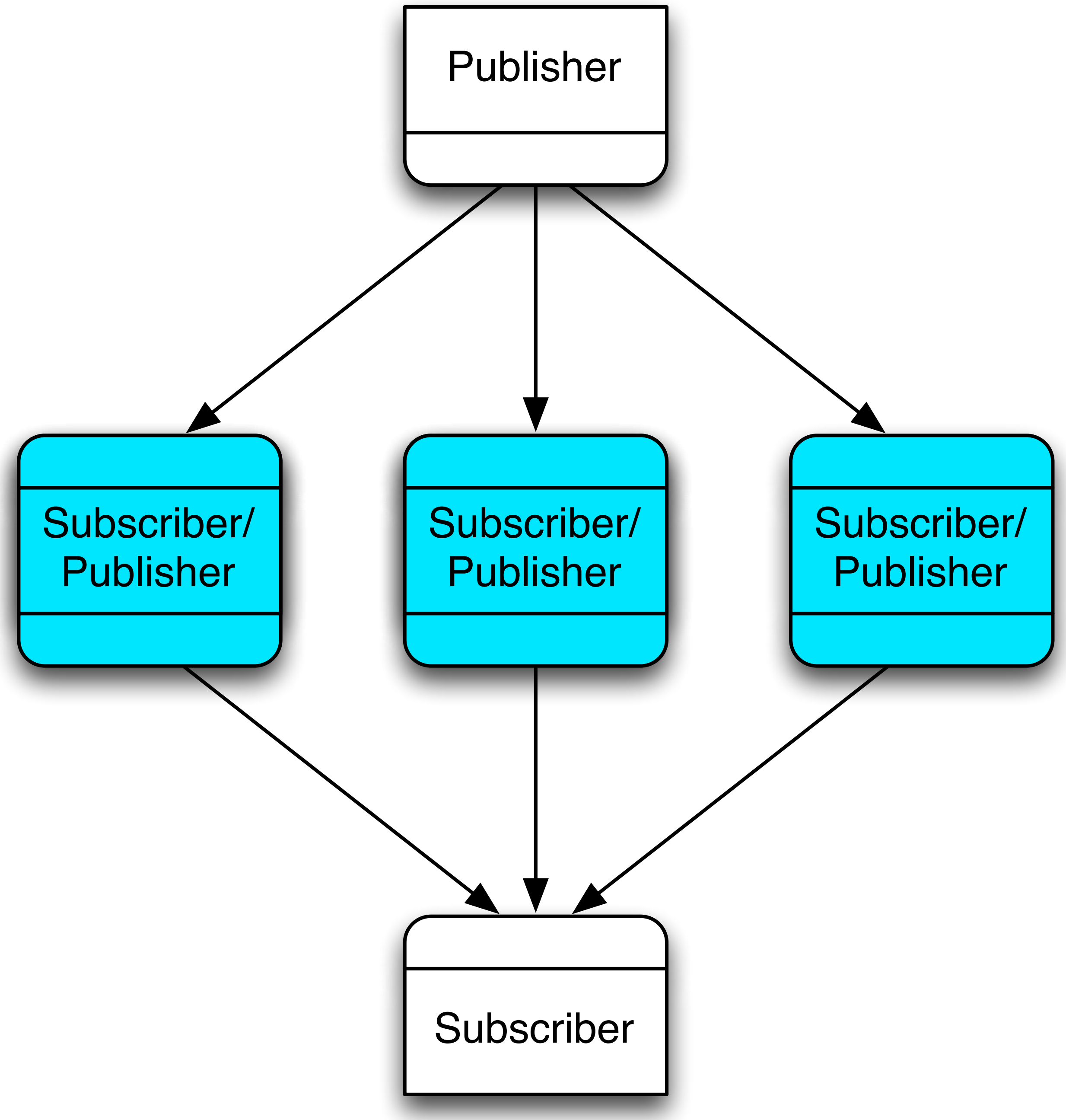
Publisher/Subscriber

- Publisher puts out a stream of messages.
- A subscriber can subscribe to a publisher and will receive messages.
- One publisher can have many subscribers.



Publisher/Subscriber

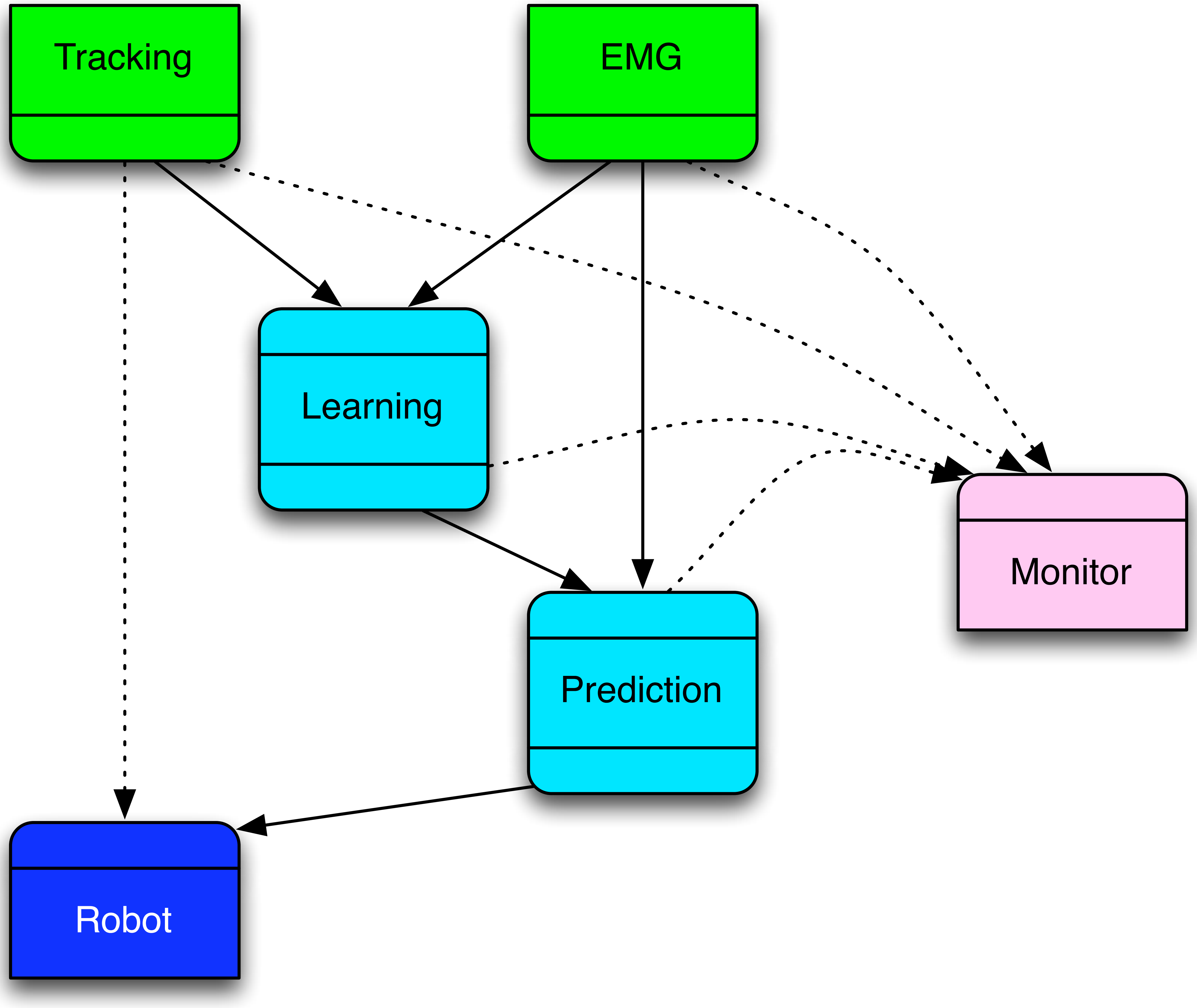
- Publisher puts out a stream of messages.
- A subscriber can subscribe to a publisher and will receive messages.
- One publisher can have many subscribers.



orakle

(http://github.com/bayerj/orakle)

- Send/receive numerical data via zeromq pub/sub.
- 162 lines of python (including docs).
- Each type of array (e.g. prediction, tracking, emg) has an associated class for overhead/bookeeping.
- Uses “coroutines” to establish a “just in time” pipeline.


Tracking
EMG
Learning
Monitor
Prediction
Robot

```python
@coroutine
def subscribe_to_arrays(socket, msg_class):
    """Yield arrays encoded by `msg_class` from `socket`."""
    (yield)
    while True:
        data = socket.recv()
        msg = msg_class.fromstring(data)
        if msg.status != 0:
            yield None
            continue
        yield msg.data
```

```python
@coroutine
def publish_arrays(socket, msg_class):
    """Publish arrays encoded by `msg_class` to `socket`."""
    while True:
        arr = (yield)
        if arr.size == 0:
            msg = msg_class(1, arr)
        else:
            msg = msg_class(0, arr)
        socket.send(msg.tostring())
```

```
def sync_sockets(sockets, msg_classes):
    """Receive messages given by `msg_classes` published
    at `sockets` until all sources are somewhat in
    sync."""
    assert len(sockets) == len(msg_classes)

    # Wait until all sockets are sending.
    for socket in sockets:
        socket.recv()

    # Loop through all sockets until no socket has a
    message pending.
    while True:
        received_sth = False
        for socket in sockets:
            try:
                socket.recv(zmq.NOBLOCK)
            except zmq.ZMQError:
                continue
            received_sth = True
        if not received_sth:
            break
```

```python
def sync_receive(sockets, msg_classes):
    """Receive from sockets in synchronization."""
    rcvrs = [subscribe_to_arrays(i, j)
           for i, j in zip(sockets, msg_classes)]
    for msgs in itertools.izip(*rcvrs):
        if not None in msgs:
            yield msgs
```

```python
def collect_data_set(emg_socket, track_socket, n_msgs):
    orakle.sync_sockets(sockets, msg_classes)
    pairwise_msgs = orakle.sync_receive(
        [emg_socket, track_socket],
        [message.EmgMessage, message.TrackMessage])

    emg_msgs = []
    track_msgs = []
    for i, (emg_msg, track_msg) in enumerate(pairwise_msgs):
        emg_msgs.append(emg_msg)
        track_msgs.append(track_msg)
        if i >= n_msgs - 1:
            break

    return emg_msgs, track_msgs
```

```
def fit_on_sub(model, emg_socket, track_socket, n_msgs):
    emg, track = collect_data_set(
        emg_socket, track_socket, n_msgs)
    model.fit(X, Z)
```

```
def predict_and_pub(model, emg_socket, predict_socket):
    message.EmgMessage.emptysocket(emg_socket)

    sub = orakle.subscribe_to_arrays(emg_socket, message.EmgMessage)
    pub = orakle.publish_arrays(
        predict_socket, message.PredictMessage)

    for arr in sub:
        y = model.predict(arr)
        pub.send(y)
```

http://python.org

http://zeromq.org/

http://numpy.org

http://scipy.org/

http://github.com/bayerj/orakle

http://brml.de

Thanks.