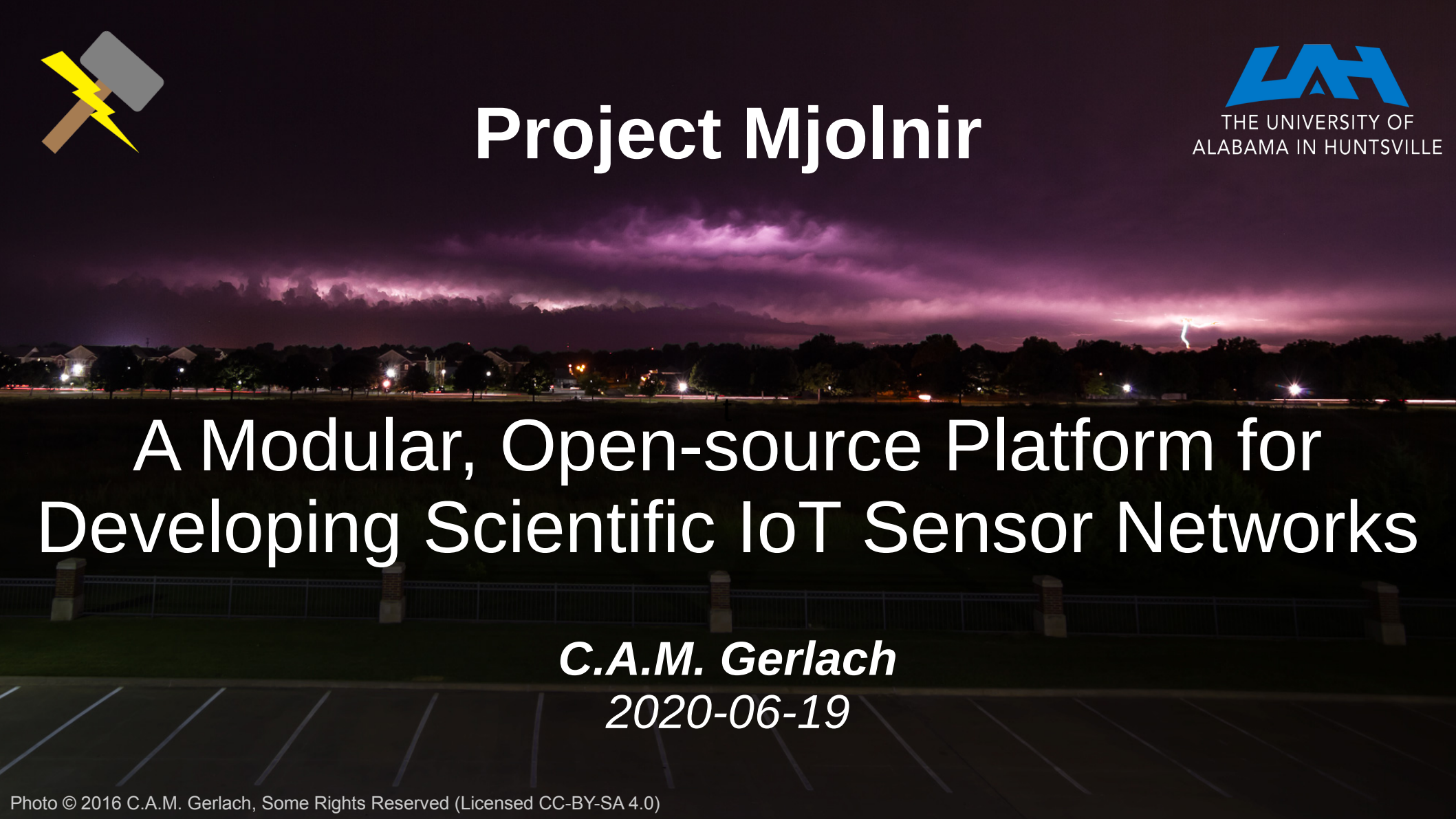


Project Mjolnir

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

A Modular, Open-source Platform for Developing Scientific IoT Sensor Networks

C.A.M. Gerlach

2020-06-19

Collaborators & Thanks

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Dr. Phillip Bitzer and the entire UAH HAMMA group for motivating & being an integral part of the project
- Dr. Udaysankar Nair and the ATS590 participants for their support extending and testing the client
- David Corredor, Ryan Driver and the NASA NSSTC Engineering/IT teams for their technical assistance
- Members of the Spyder scientific Python IDE team for their ideas, feedback and support

The big ideas

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

How and why did we get here?

What is the Mjolnir platform, and how does it work?

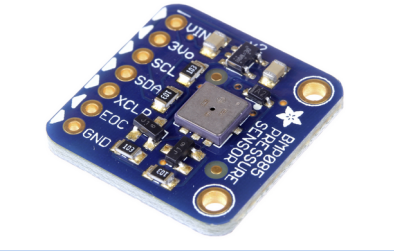
How can *you* use Mjolnir for your own projects?

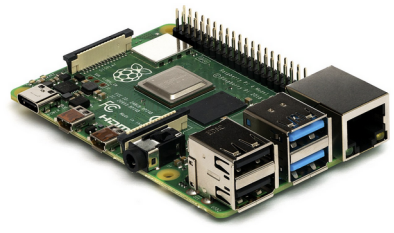
IoT for transformative science

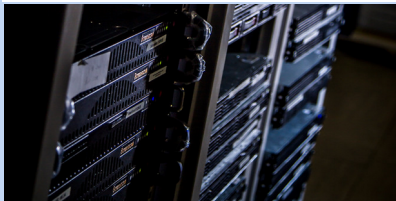
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Era of scientific sensor networks is here!
Convergence of:

- Ultra low-cost, low-power, low-profile sensors
- Widespread high speed 4G+ connectivity
- Cheap, reliable single board computers (RPi)
- VPSes and Cloud Services for hosting
- Buzzwords, and lots of them







Something's missing!

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

?

But what?

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Let’s start at the beginning...

So it begins...

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

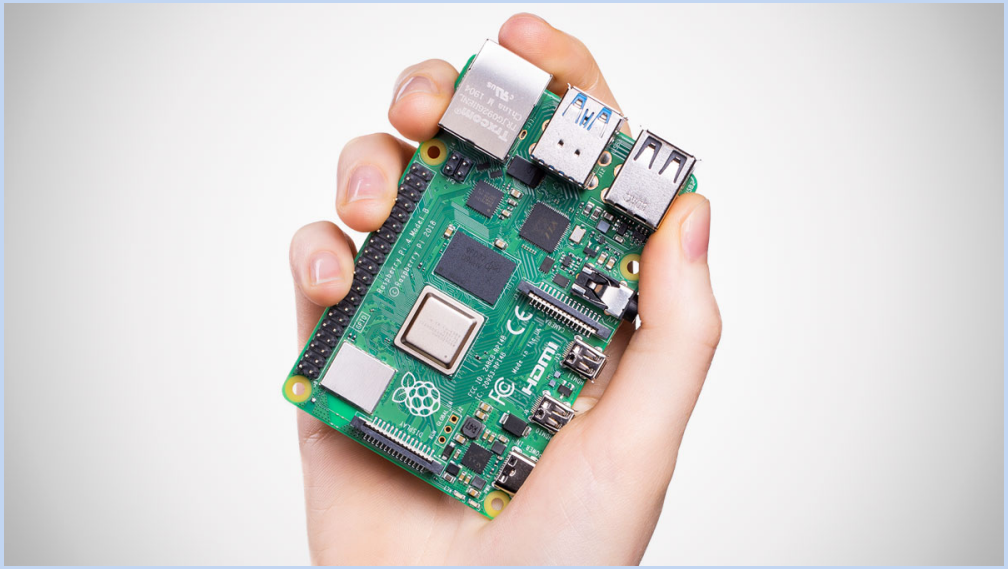


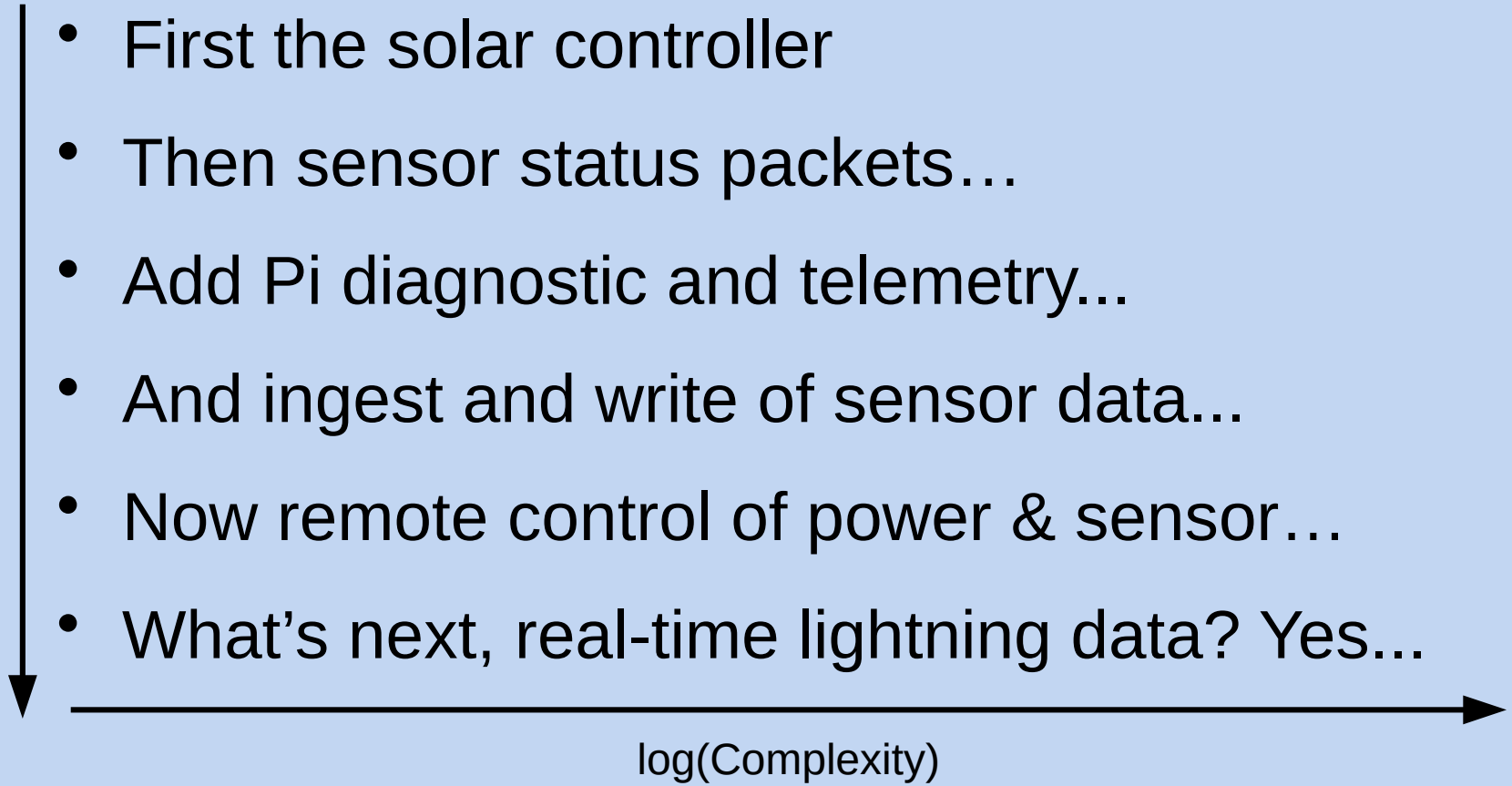
Image: Raspberry Pi Foundation – CC-BY-SA 4.0

...with a “simple” idea:
Use a Raspberry Pi single-board computer to monitor, control and get real-time data from an existing network of lightning sensors

Requirements creep

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- First the solar controller
- Then sensor status packets…
- Add Pi diagnostic and telemetry...
- And ingest and write of sensor data...
- Now remote control of power & sensor…
- What's next, real-time lightning data? Yes...

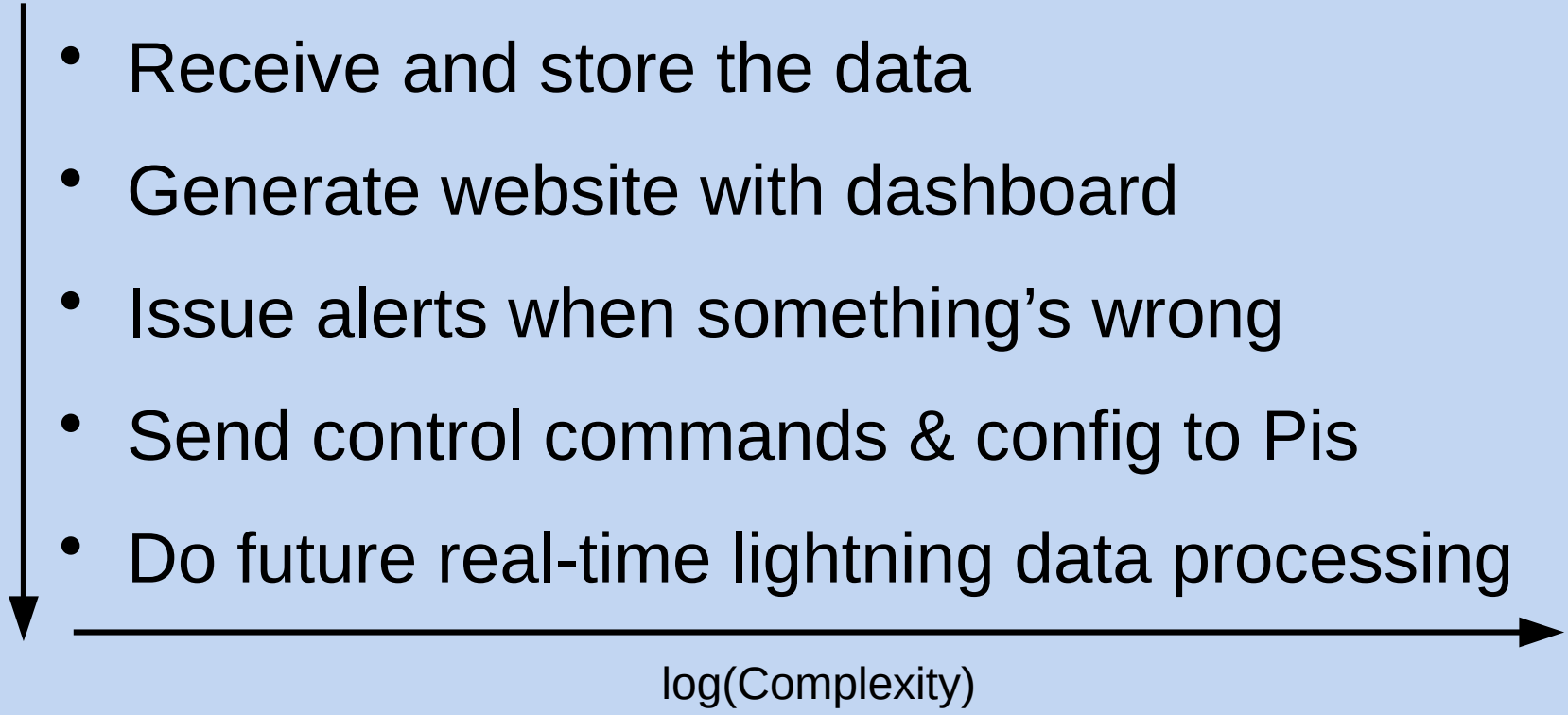


But wait...

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

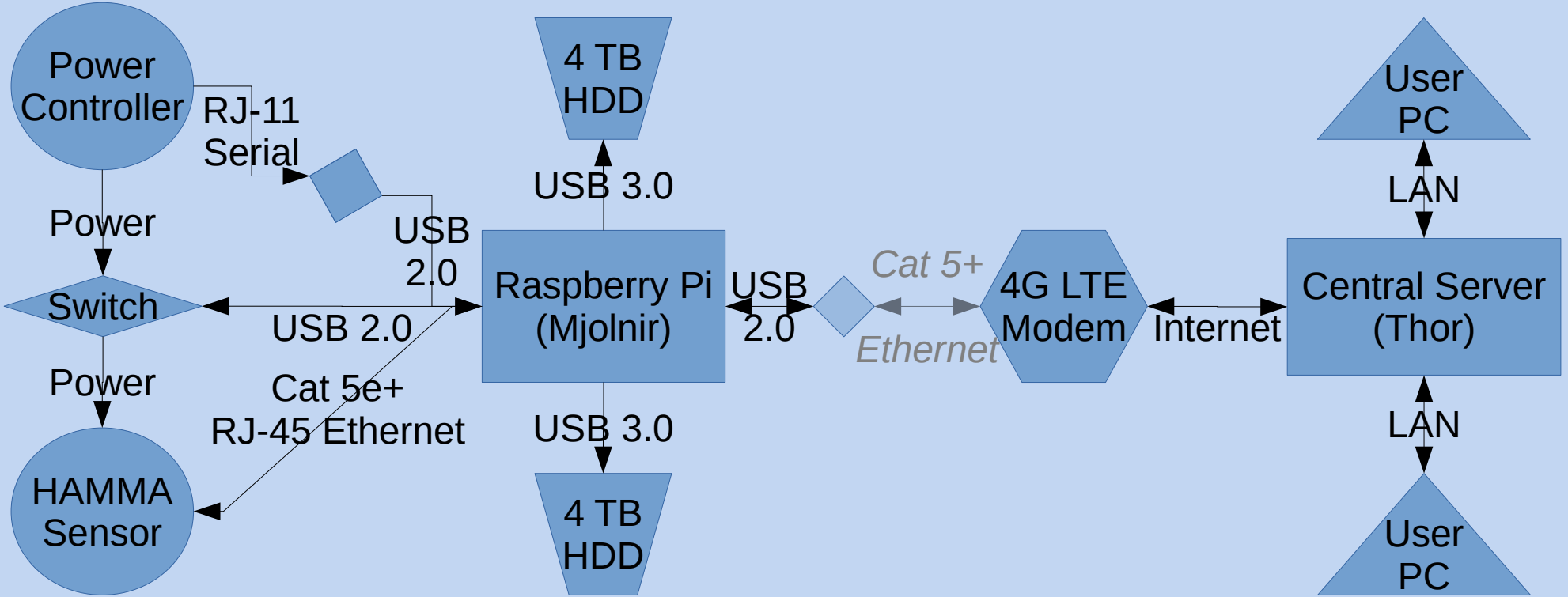
There’s more! What about the server-side component?

- Receive and store the data
- Generate website with dashboard
- Issue alerts when something’s wrong
- Send control commands & config to Pis
- Do future real-time lightning data processing



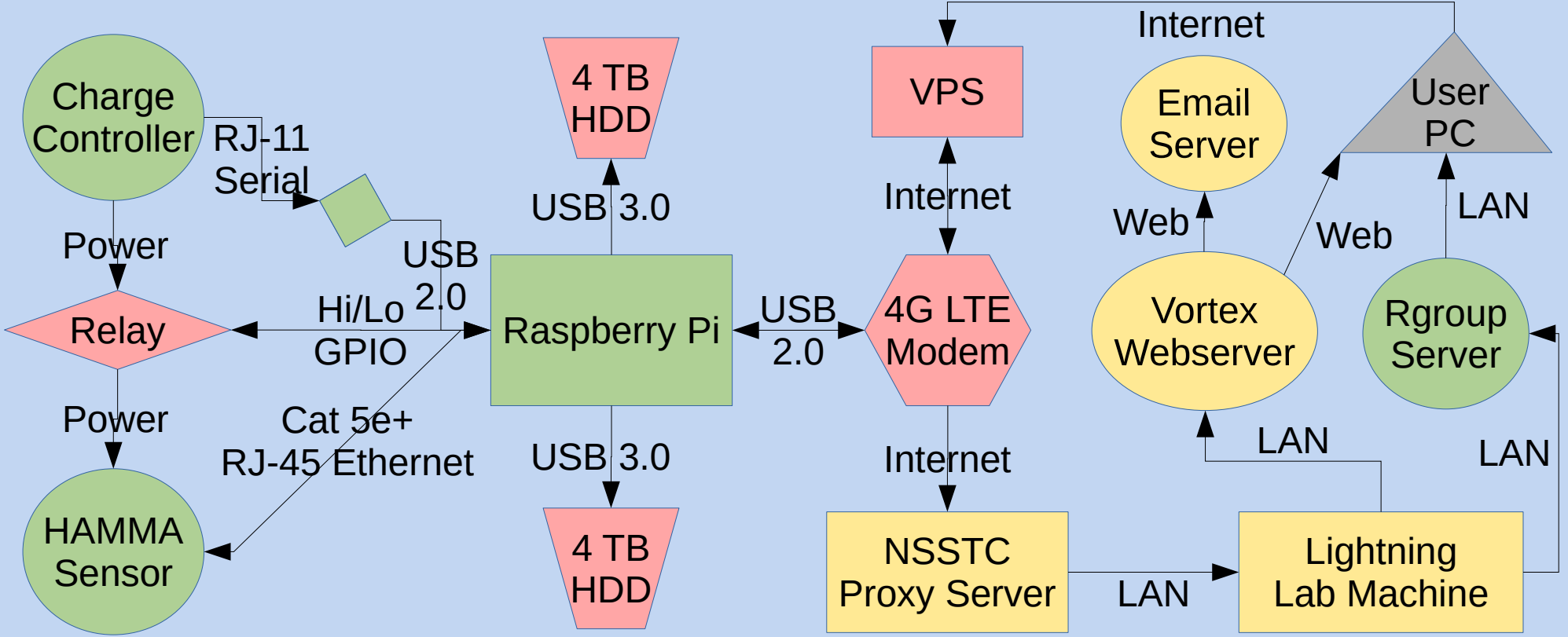
Mjolnir System Diagram: Hardware

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



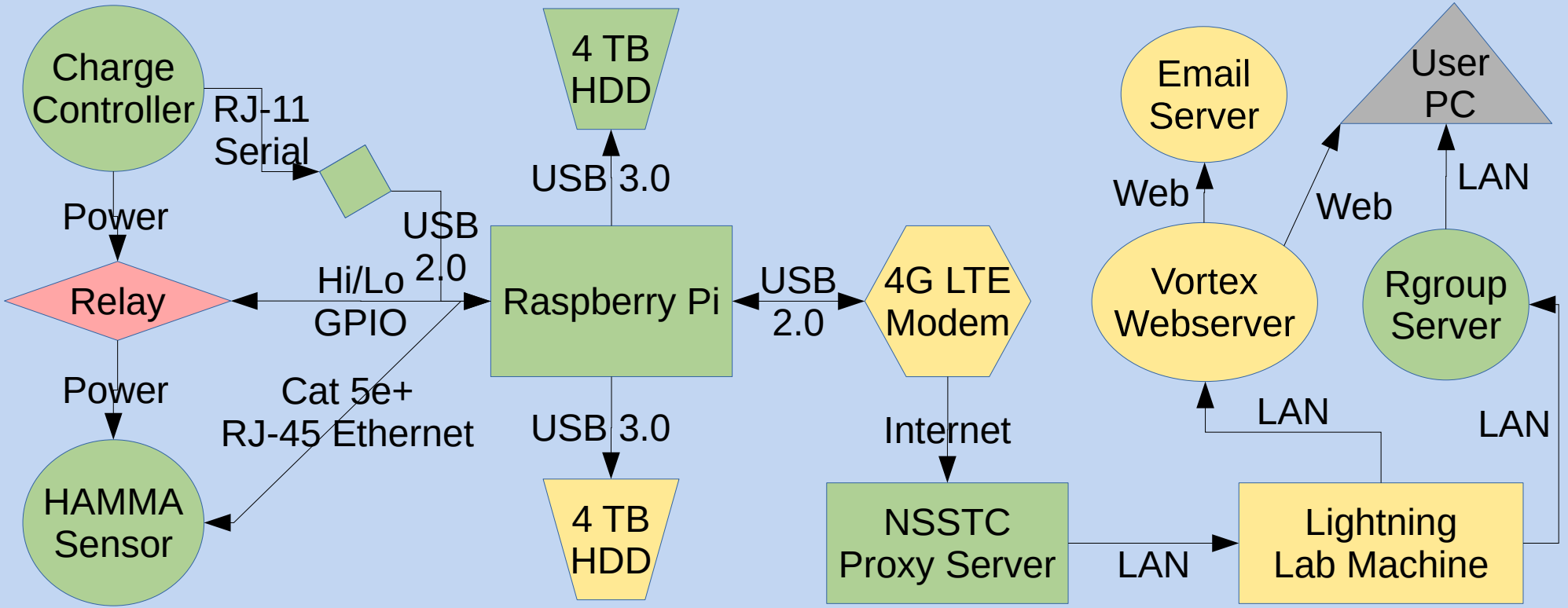
Mjolnir System Diagram: Hardware Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



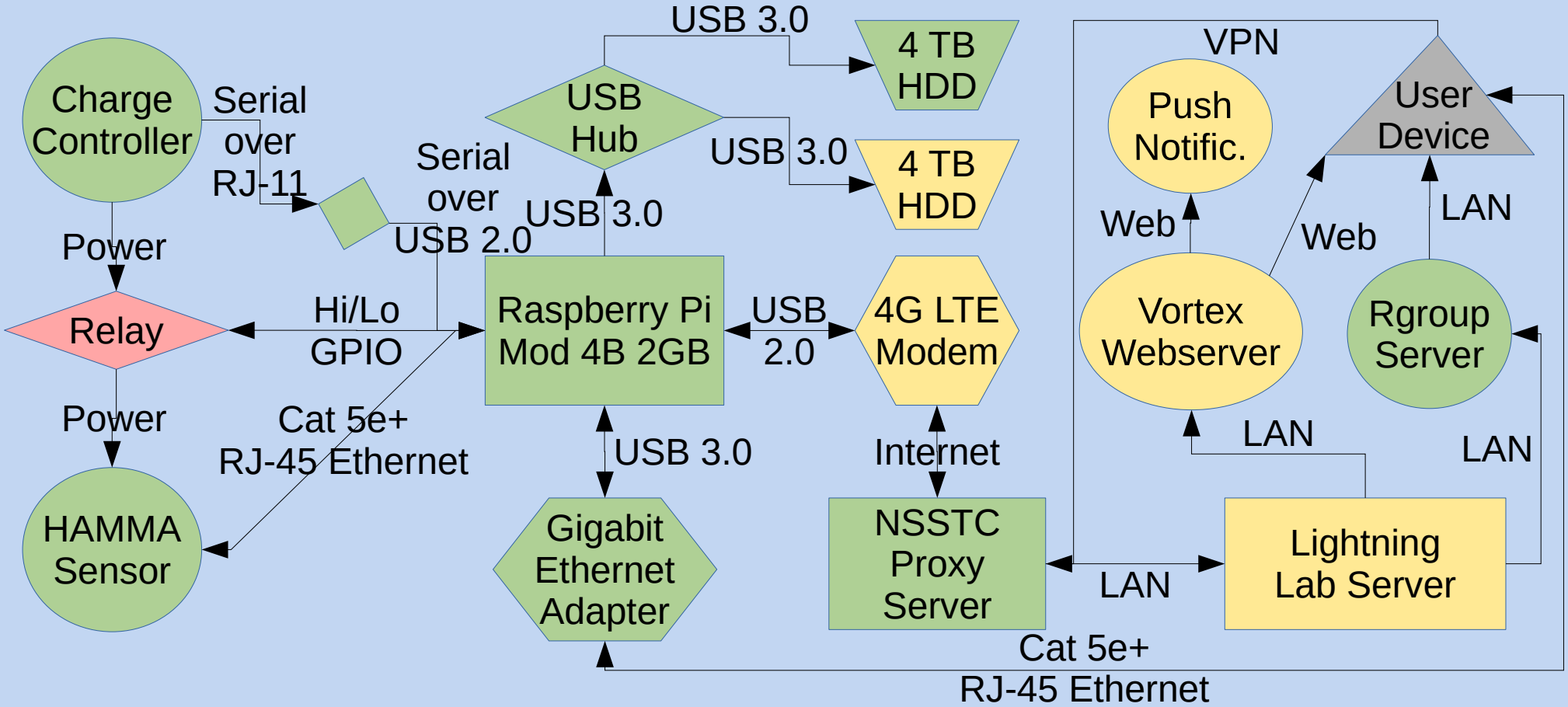
Mjolnir System Diagram: Hardware Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



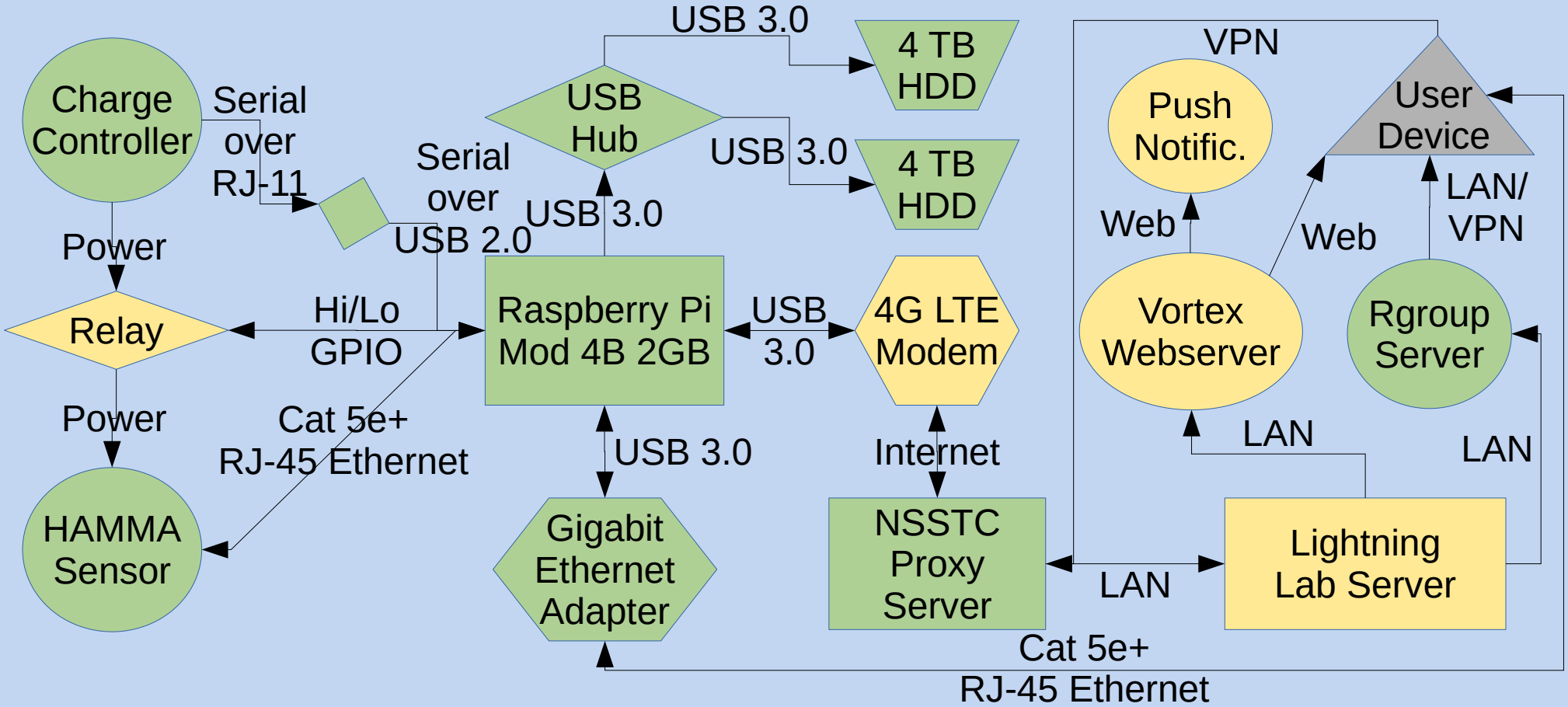
Mjolnir System Diagram: Hardware Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



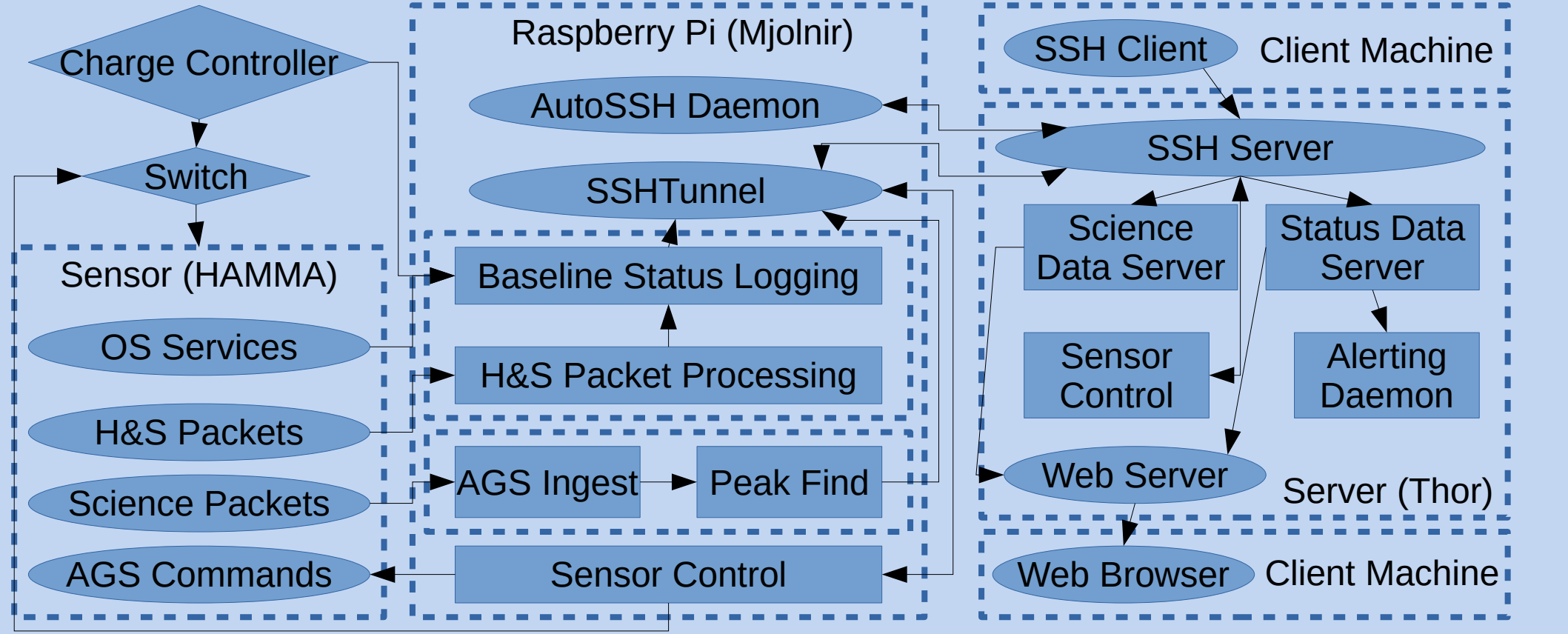
Mjolnir System Diagram: Hardware Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



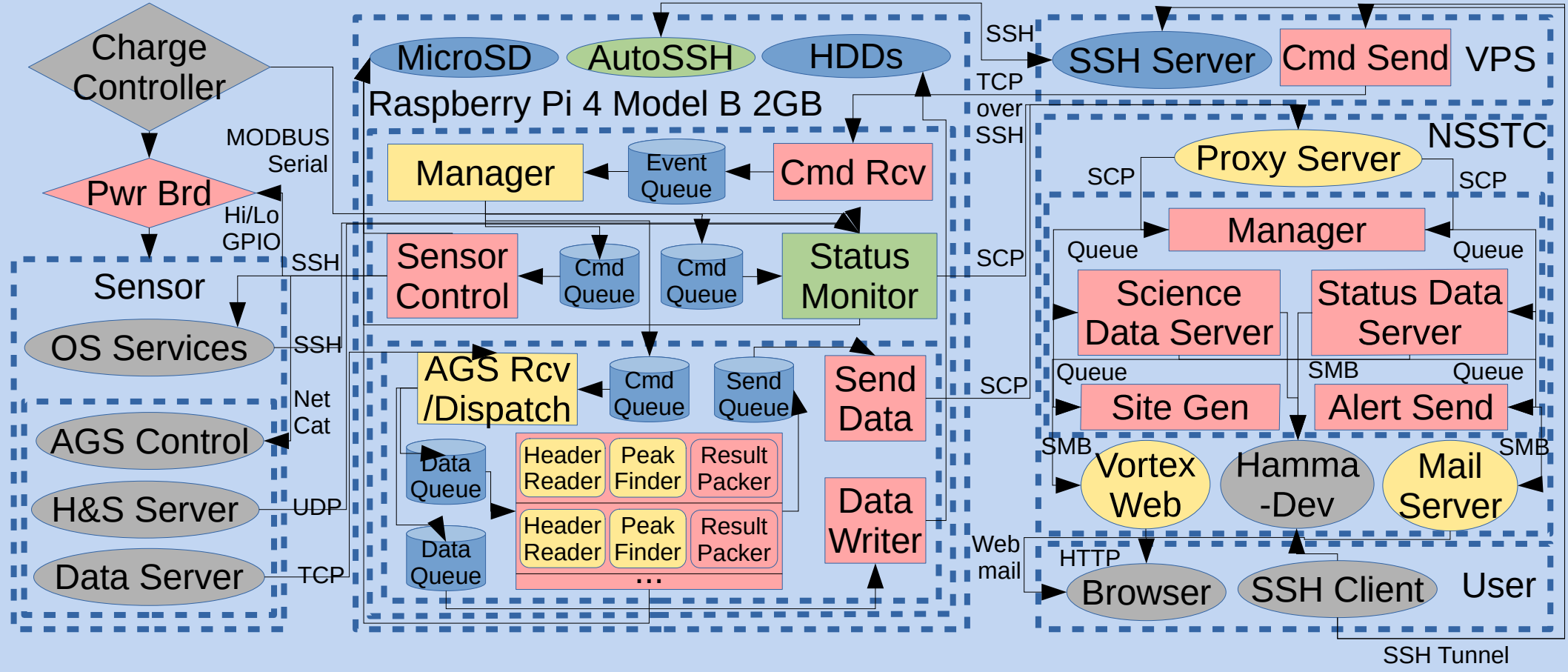
Mjolnir System Diagram: Software Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



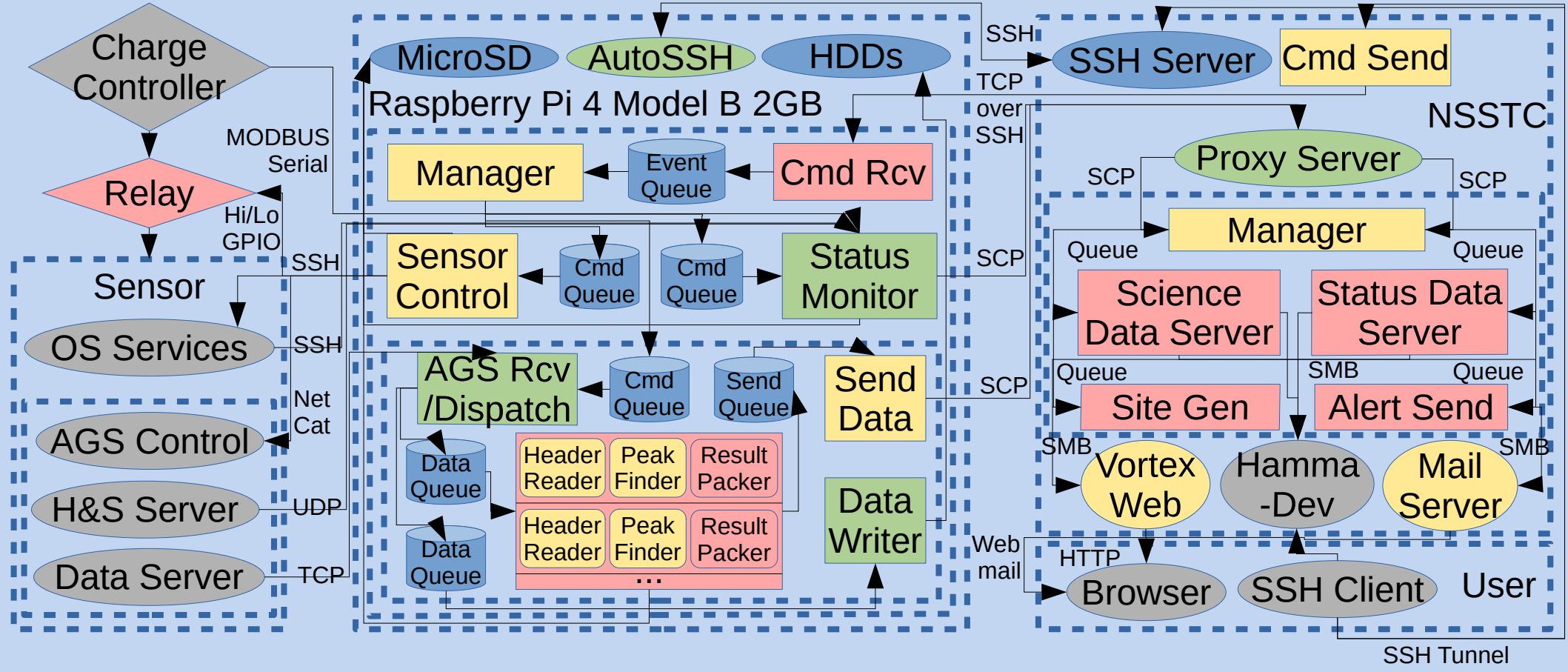
Mjolnir System Diagram: Software Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



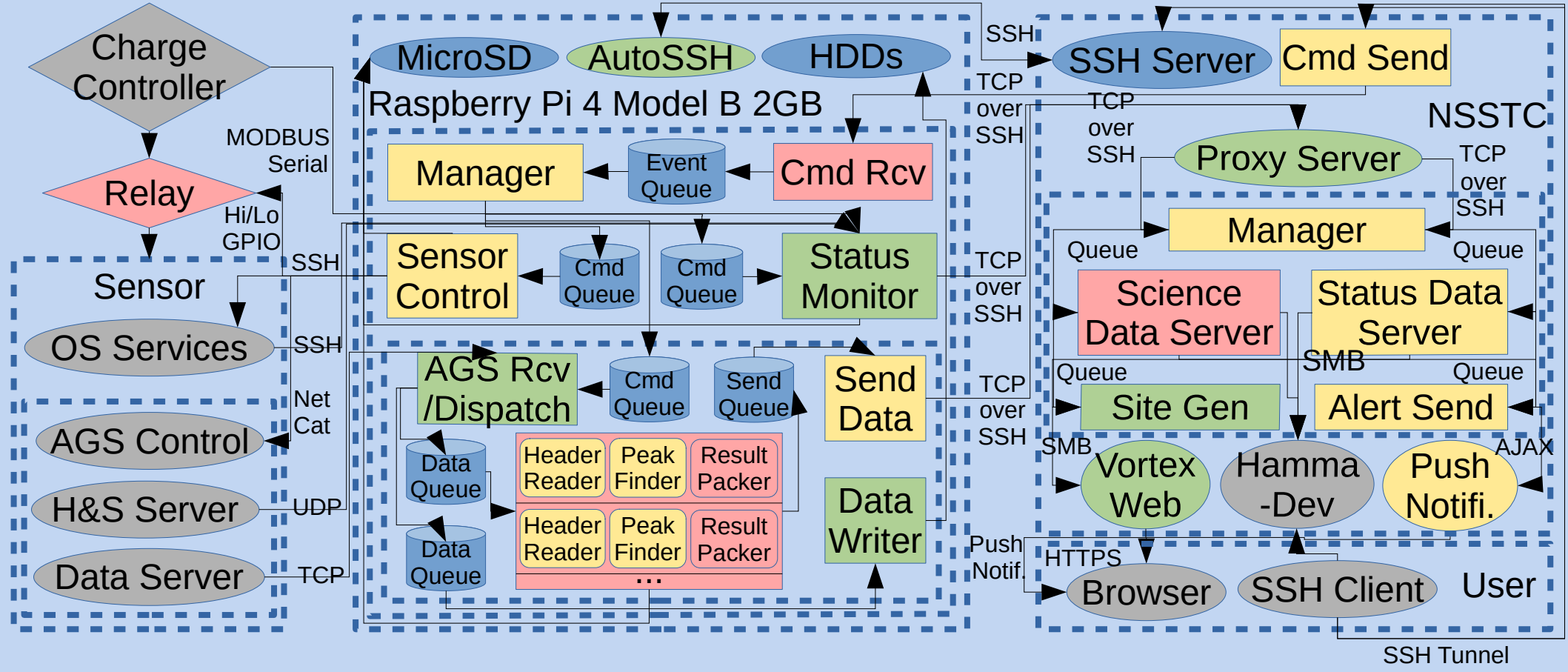
Mjolnir System Diagram: Software Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



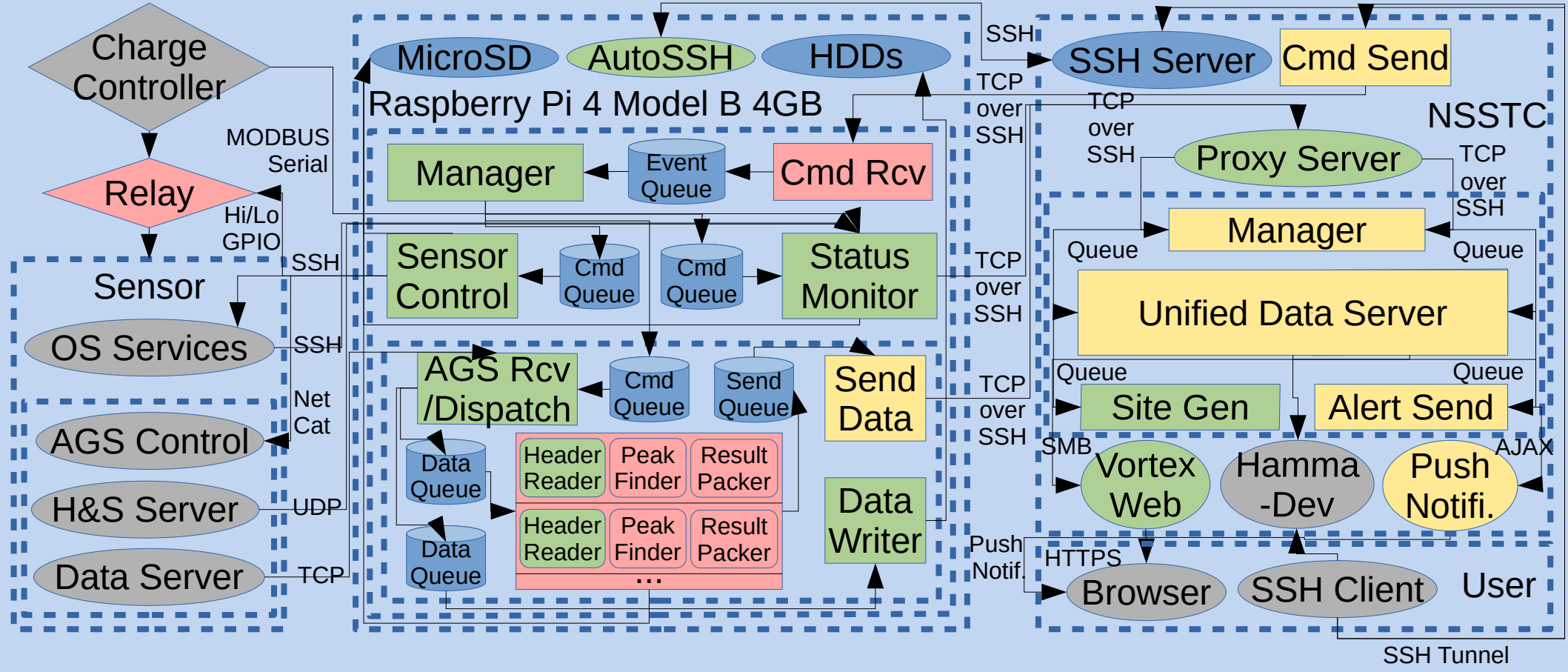
Mjolnir System Diagram: Software Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



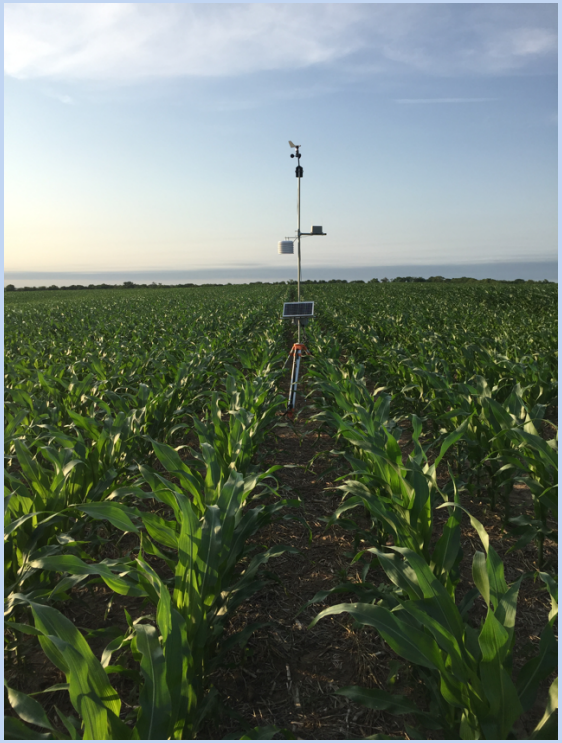
Mjolnir System Diagram: Software Overview

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



Common ground (and air)

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



A number of other research groups interested in a shared IoT client framework:

- HAMMA2 lightning sensor network
- EMESH low-cost/high-density sensor grid
- Experimental sensors for specific uses
- UAV and aircraft-deployed instruments
- More on the horizon...

A motivation for Mjolnir

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Lots of IoT middleware/cloud platforms; few client
- Expensive, inflexible proprietary data loggers
- Open-source options bespoke, limited, not robust
- Many sensor interface libraries; no common API
- Requires substantial time, effort and programming to build a custom client for every use, every time
- Huge wasted potential to do novel, quality science

So...what's Mjolnir?

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

A flexible, easy-to-use, open-source **Python framework** designed for low-cost **scientific sensor networks**, featuring a robust **client**, experimental **server**, dynamic **data dashboard**, lightweight **plugin system**, declarative **device presets,** and centralized **config**.

Common infrastructure

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Fills typical needs for scientific IoT client system:

- Automated install/deploy
- Periodic/on-demand data ingest
- Recording and uplink
- Multithreaded real-time processing
- Status logging
- Robust error handling
- Config management
- Service installation
- Dynamic data visualization

Built-in core I/O support

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Supported at the protocol level; a declarative preset may be required for support of specific devices and services

Inputs

- ✔ SPI
- ✔ I2C
- ✔ SMBus
- ✔ GPIO
- ✔ Analog
- ✔ UART/serial
- ✔ Modbus
- ✔ TCP packets
- ✔ UDP packets
- ✔ Onewire/1-wire

Outputs *(*WiP)*

- ✔ Raw print
- ✔ Pretty print
- ✔ Binary file
- ✔ CSV file
- ✔ Syslog
- ✔ HTTP
- ▫ *TCP**
- ▫ *REST**
- ▫ *UART**
- ▫ *Wireless**

Included device presets

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Built-in support with a single line in your config included for the following:

Basic: *Current time, Brokkr runtime, ping status, pseudorandom test data*

Adafruit I2C temperature/humidity/pressure sensors:
BME280 I2C, BMP280 I2C, HTU21D, MPL311A2, MLX90614, SHT31D, Si7021

Adafruit SPI temperature/humidity/pressure sensors: *BME280 SPI, BMP280 SPI*

Adafruit I2C ADC boards: *ADS1015, ADS1115, analog anemometer*

Adafruit onewire temperature/humidity: *DHT11, DHT22*

Sparkfun/generic I2C/SMBus sensors: *HIH6130*

Meter ECH2O soil moisture: *EC-5, EC-10, EC-20*

Maxim 1-wire temp: *DS18B20, DS18S20, DS1822, DS1825, DS28EA, MAX31850*

Modbus devices: *Morningstar Sunsaver MPPT-15L solar charge controller*

GPIO devices: *Generic switch, generic counter, generic digital anemometer*

WiP: *MB7386 ultrasonic sensor, MTK3339 GPS, 433 MHz transceiver, BMP80/180...*

Server/dashboard features

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Data ingest and processing
- Real-time interactive data dashboard
- Backend: Code generation + Lektor/Icon
- Frontend: Plot.ly/D3/AJAX/JS
- Static server-side; can run right on IoT client, GH pages, your webserver, etc.
- Allows remote/local monitoring, data download, APIs, diagnostics...

Key concepts

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Unit

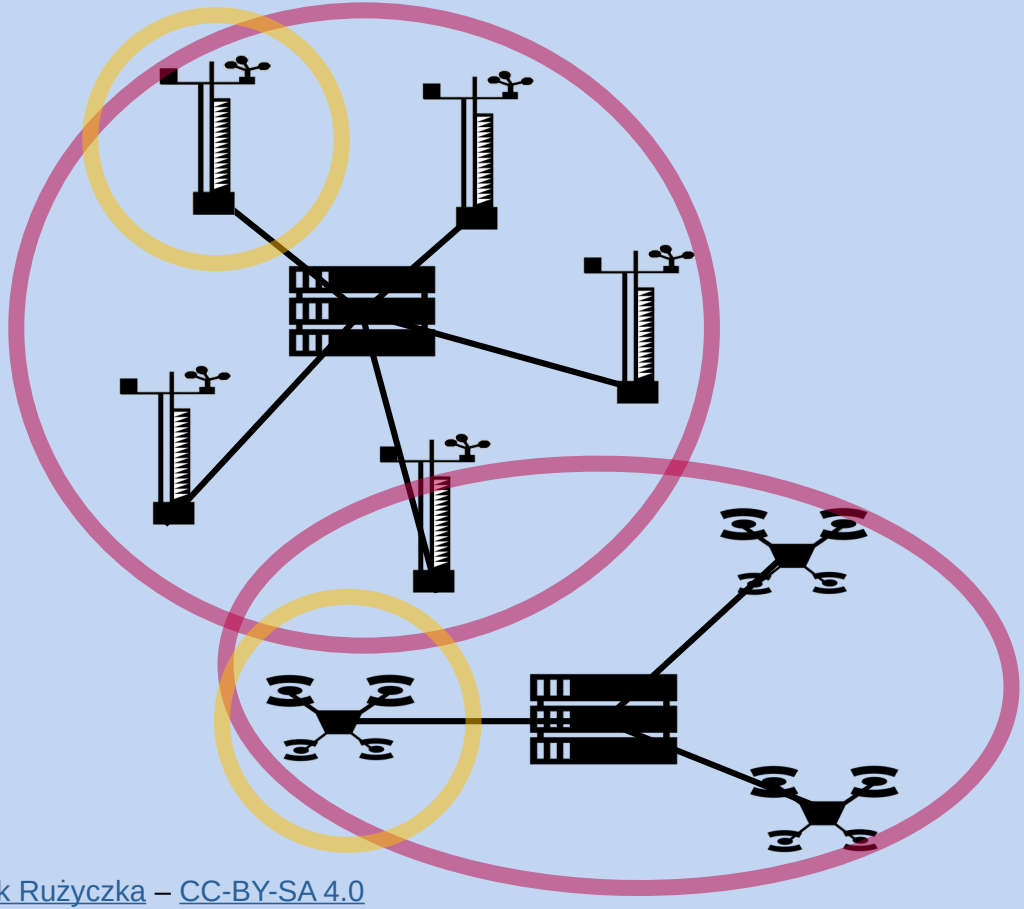
An individual client node in a networked Mjolnir system

System

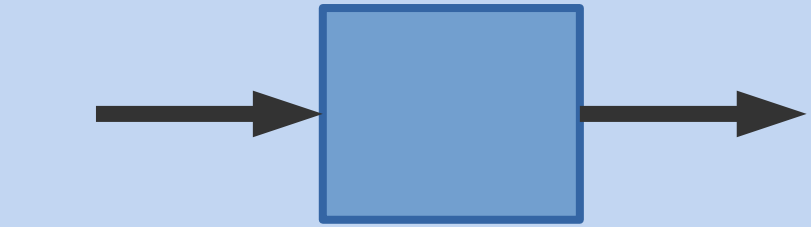
A set of units with a common purpose and configuration

System Config Package

Directory with a system's plugins, presets and config

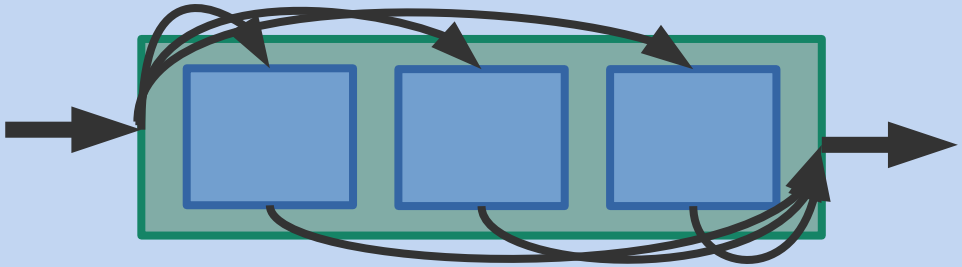


Core Pipeline classes



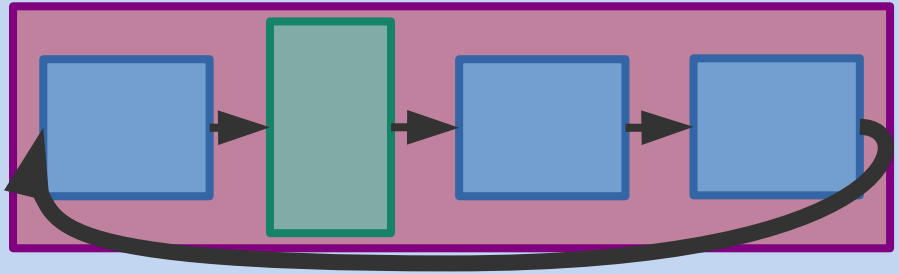
PipelineStep

An atomic input, processing or output action to perform



MultiStep

A set of `PipelineSteps`, executed in series or parallel

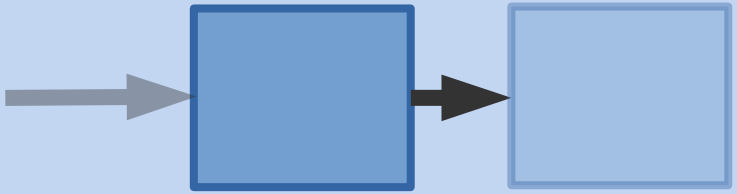


Pipeline

A set of `Pipeline`/`MultiSteps` run as a process

PipelineStep Types

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

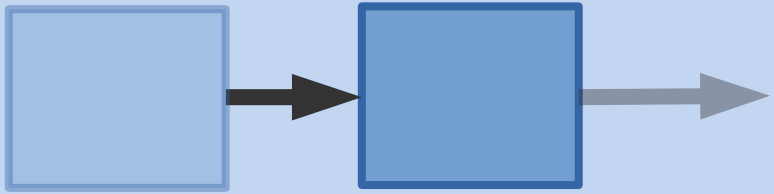


InputStep

Sensor, IPC queue, device

UnitStep

Self-contained steps, logging



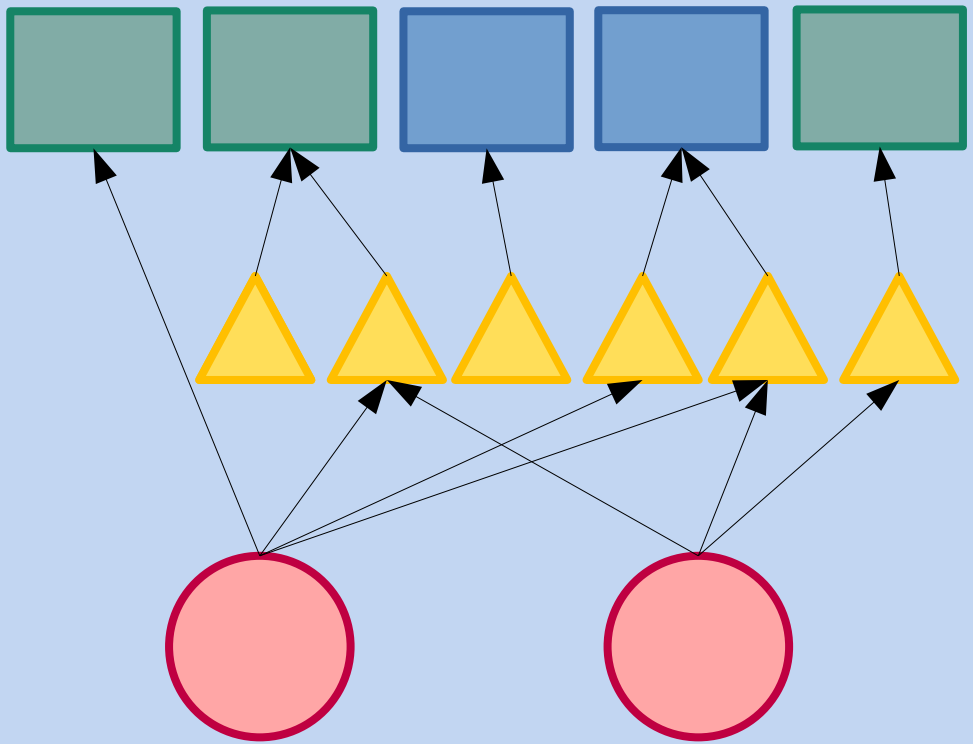
OutputStep

File, REST API, network link

TransformStep

Filtering, smoothing, actions

SCP elements



Plugin

A `PipelineStep` implemented in an external file or package

Preset

A device-specific declarative config for a `PipelineStep`

Main Config

File defining the `Pipelines`, presets and settings to use

Config Schema

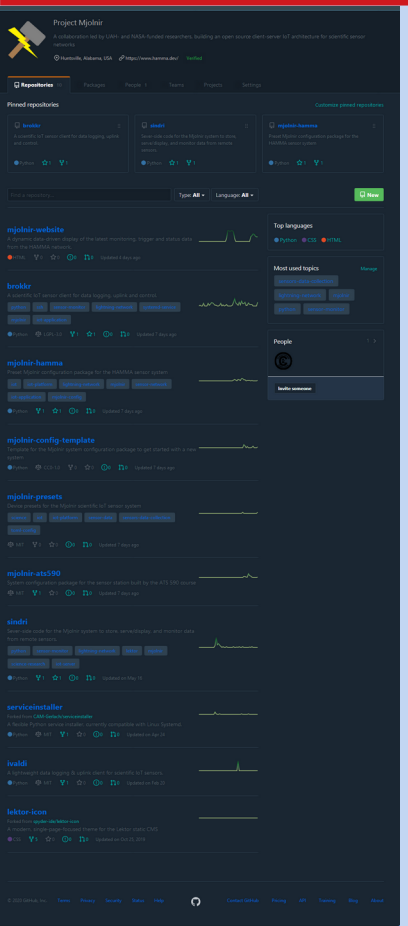
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- TOML for human-facing config, JSON for machine
- System config package centralizes setup for whole system of many units
- Any parameter can be overridden in local config
- Config can be managed and updated remotely via git

- mjolnir-system-name
  - config
    - log_client.toml
    - main.toml
    - mode.toml
    - unit.toml
  - plugins
    - custom_input.py
    - …
  - presets
    - custom_input.preset.toml
    - other_input.preset.toml
    - …
  - metadata.toml

Primary subprojects

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE



[Brokkr](#) IoT client package, run on each unit device

[Sindri](#) IoT server and per-device dashboard

[Mjolnir-Config-Template](#) Ex. system config package

[Mjolnir-Presets](#) Presets for common sensors/devices

[Mjolnir-Website](#) GHP repo of the live demo server site

[Mjolnir-HAMMA](#) System config package for HAMMA2

[Mjolnir-ATS590](#) System config package for EMESH

So...

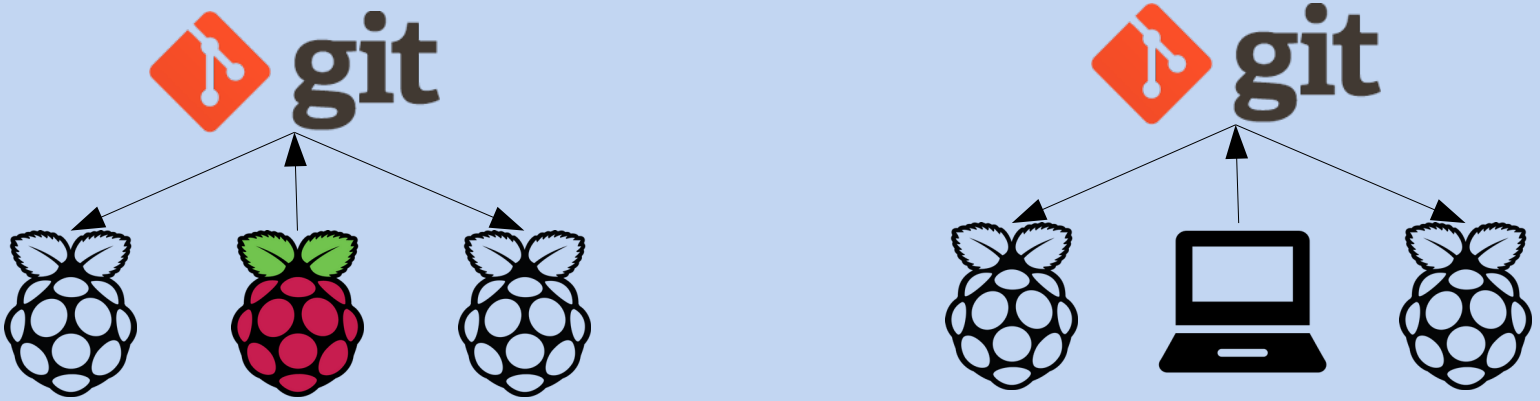
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

How can *you* use Mjolnir for your project?

Platform support

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Supports single board computer running Linux (e.g. RPi) for production and Windows, macOS and *nix for development



Can work directly on a SBC, or develop and test system config package locally then deploy via git/GitHub/GitLab

Zero to Mjolnir in 5 steps

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
$ brokkr --system template status
No local unit config found at 'C:/Users/C. A. M. Gerlach/.config
te/unit.toml', falling back to defaults
Time (UTC): 2020-06-23 08:20:01.245796+00:00 s +/-0.015625 s
Runtime: 0.172 s +/-0.015625 s
Ping Retcode: 0
Example Volts: 15.043 V +/-0.1 V
Example Current: 16 A +/-0.5 A
Example Time: 1996-06-18 16:21:01.124921+00:00 s
Example Str: 0.5865718178967154
Example Bool: False
Ping Retcode: 1
```

0) Create & activate a venv/conda env

1) Install Brokkr: `pip install brokkr`

2) Clone Git repo for your SCP, e.g. Mjolnir-Config-Template
   `git clone --recursive https://github.com/hamma-dev/mjolnir-config-template.git`

3) Register the system with Brokkr:
   `brokkr configure-system template path/to/mjolnir-config-template`

4) Initialize the local configuration: `brokkr configure-init`

5) Run `brokkr status`, `brokkr monitor` or `brokkr start`

Create your own SCP

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

0) Copy Mjolnir-Config-Template to a new directory, named what you wish (e.g. `mjolnir-yoursystem`)

1) In `metadata.toml`, change name to be unique to your system, and update the other metadata fields accordingly

2) Register your system with Brokkr and set it as default:
```
brokkr configure-system --default yoursystem path/to/mjolnir-yoursystem
```

3) Check that Brokkr is using the new system by default with `brokkr --version` or `brokkr configure-system`

```
C. A. M. Gerlach@QUASIMODO MINGW64 ~
$ brokkr --version
Brokkr version 0.3.0, Mjolnir Config Template System version 0.2.0
(brokkr)
C. A. M. Gerlach@QUASIMODO MINGW64 ~
$
```

Metadata.toml

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```toml
# Metadata for this specific system configuration preset

config_version = 1

# Name of the system; used internally to reference it and for paths, etc.
name = "template"

# Full "pretty" name of the system, e.g. for UI text
name_full = "Mjolnir Config Template System"
# Freeform description of the system's author
author = "Your name here"
# Freeform description of the system's intended function and purp
description = "System configuration for the template package"
# Official website for this specific system
homepage = "https://www.example.com"
# Repository containing the system's source code
repo = "https://github.com"
# SPDX license identifier for this configuration package (CC0/PD)
license = "CC0-1.0"

# Version of the config package; should follow SemVer and PEP 396
version = "0.4.0dev0"
# Minimum Brokkr version this version of the config package needs to work
brokkr_version_min = "0.3.0"
# As above, but maximum version (e.g. "0.3.99" for < 0.4.x)
brokkr_version_max = ""
# Ibid, for Sindri
sindri_version_min = "0.3.0"
sindri_version_max = ""

```

mjolnir-yoursystem/metadata.toml

`name` uniquely identifies the system in filenames, etc.

`name_full` is the display name shown in the UI

`Metadata` is useful for users of your system

`Version` defines its compat with client/server versions

Set up your configuration

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

All configuration is under `mjolnir-yoursystem/config`

`Log_client.toml` controls the Python `logging` config

`Main.toml` is the primary system configuration file, containing general settings and the steps & pipelines to use

`Mode.toml` defines sets of config overrides (e.g. `test` or `realtime`), activated with the `--mode NAME` CLI option

`Unit.toml` contains the defaults for unit-specific settings, like number or net interface, set with `brokkr configure-unit`

Main.toml config file

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

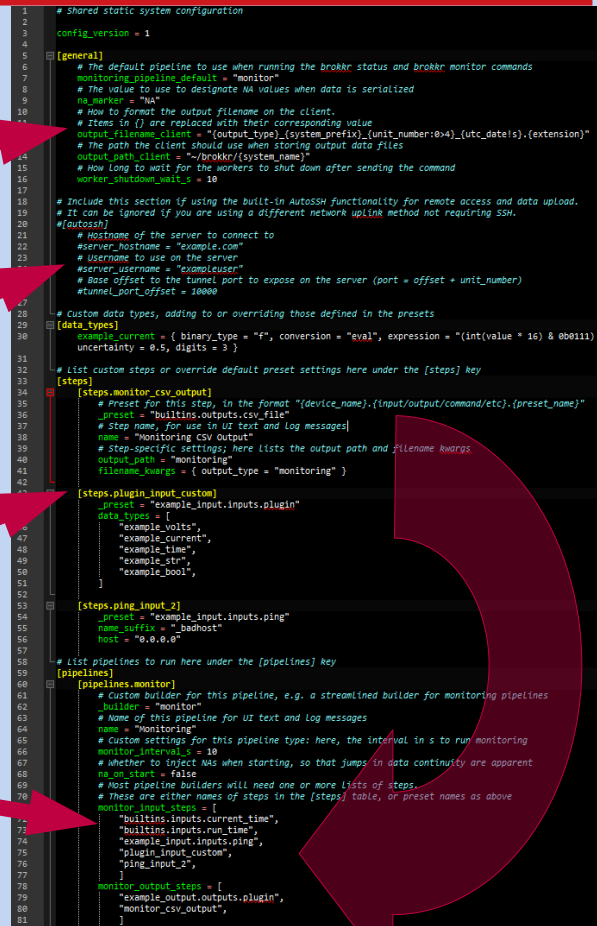
`mjolnir-yoursystem/config/main.toml`

`General` options affect Brokkr's overall operation, e.g. default output path

`AutoSSH` settings control the reverse SSH tunnel for client-server comms

`Steps` specifies custom params for the `PipelineSteps` referenced in `Pipelines`

`Pipelines` defines the properties and steps of the `Pipelines` to execute



Main.toml: Set general config

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```toml
# Shared static system configuration

config_version = 1

[general]
    # The default pipeline to use when running the brokkr status and brokkr monitor commands
    monitoring_pipeline_default = "monitor"
    # The value to use to designate NA values when data is serialized
    na_marker = "NA"
    # How to format the output filename on the client.
    # Items in {} are replaced with their corresponding value
    output_filename_client = "{output_type}_{system_prefix}_{unit_number:0>4}_{utc_date!s}.{extension}"
    # The path the client should use when storing output data files
    output_path_client = "~/brokkr/{system_name}"
    # How long to wait for the workers to shut down after sending the command
    worker_shutdown_wait_s = 10

# Include this section if using the built-in AutoSSH functionality for remote access and data upload.
# It can be ignored if you are using a different network uplink method not requiring SSH.
#[autossh]
    # Hostname of the server to connect to
    #server_hostname = "example.com"
    # Username to use on the server
    #server_username = "exampleuser"
    # Base offset to the tunnel port to expose on the server (port = offset + unit_number)
    #tunnel_port_offset = 10000

```

Main.toml: Define pipelines

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
64
65  # List pipelines to run here under the [pipelines] key
66  [pipelines]
67      [pipelines.monitor]
68          # Custom builder for this pipeline, e.g. a streamlined builder for monitoring pipelines
69          _builder = "monitor"
70          # Name of this pipeline for UI text and log messages
71          name = "Monitoring"
72          # Custom settings for this pipeline type: here, the interval in s to run monitoring
73          monitor_interval_s = 10
74          # Whether to inject NAs when starting, so that jumps in data continuity are apparent
75          na_on_start = false
76          # Most pipeline builders will need one or more lists of steps.
77          # These are either names of steps in the [steps] table, or preset names as above
78          monitor_input_steps = [
79              "builtins.inputs.current_time",
80              "builtins.inputs.run_time",
81              "example_input.inputs.ping",
82              "plugin_input_custom",
83              "ping_input_2",
84              "example_input_minimal.inputs.plugin",
85              "example_input_sensor.inputs.sensor_property",
86          ]
87          monitor_process_steps = [
88              "example_transform.transforms.plugin"
89          ]
90          monitor_output_steps = [
91              "monitor_csv_output",
92              "monitor_repr_output",
93          ]
94
```

Main.toml: Customize steps

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```toml
# Custom data types, adding to or overriding those defined in the presets
[data_types]
    example_current = { binary_type = "f", conversion = "eval", expression = "(int(value * 16) & 0b0111) ** 2",
    unit = "A", uncertainty = 0.5, digits = 3 }

# List custom steps or override default preset settings here under the [steps] key
[steps]
    [steps.monitor_csv_output]
        # Preset for this step, in the format "{device_name}.{input/output/command/etc}.{preset_name}"
        _preset = "builtins.outputs.csv_file"
        # Step name, for use in UI text and log messages
        name = "Monitoring CSV Output"
        # Step-specific settings; here lists the output path and filename kwargs
        output_path = "monitoring"
        filename_kwargs = { output_type = "monitoring" }

    [steps.monitor_repr_output]
        # Preset for this step, in the format "{device_name}.{input/output/command/etc}.{preset_name}"
        _preset = "example_output.outputs.repr_file"
        name = "Monitoring Repr Output"
        output_path = "monitoring"
        filename_kwargs = { output_type = "monitoring" }

    [steps.plugin_input_custom]
        _preset = "example_input.inputs.plugin"
        data_types = [
            "example_volts",
            "example_current",
            "example_time",
            "example_str",
            "example_bool",
        ]

    [steps.ping_input_2]
        _preset = "example_input.inputs.ping"
        name_suffix = "_badhost"
        host = "0.0.0.0"
```

Device preset file

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
config_version = 1
# Name of this device preset, used for lookup
name = "adafruit_dht"
# Type of this preset; currently the only type is device
type = "device"
# Dependencies required by all inputs in this device preset
_dependencies = [ "adafruit-circuitpython-dht" ]

# Preset-level metadata; format follows that specified in the top-level
[metadata]
    name_full = "Adafruit DHTxx Temp and Humidity Sensors"
    author = "C.A.M. Gerlach"
    description = "Device preset for reading the AAdafruit DHTxx (DHTI2C Temp/Humidity sensor
    homepage = "https://www.Gerlach.CAM"
    repo = "https://github.com/CAM-Gerlach"
    preset_version = "0.1.0"
    brokkr_version_min = "0.3.0"

[type_presets]
    dht_temperature = { binary_type = "f", unit = "C", custom_attrs.attribute_name =
    "temperature", digits = 3 }
    dht_humidity = { binary_type = "f", unit = "%", custom_attrs.attribute_name = "humidity",
    digits = 3 }

# Data types referenced in individual presets
[data_types]
    dht11_temperature = { type_presets = "dht_temperature", full_name = "Temperature
    (DHT11)", uncertainty = 2.0, range_min = 0, range_max = 50 }
    dht11_humidity = { type_presets = "dht_humidity", full_name = "Humidity (DHT11)",
    uncertainty = 5.0, range_min = 20, range_max = 80 }
    dht22_temperature = {  type_presets = "dht_temperature", full_name = "Temperature
    (DHT22)", uncertainty = 0.5, range_min = -40, range_max = 80 }
    dht22_humidity = { type_presets = "dht_humidity", full_name = "Humidity (DHT22)",
    uncertainty = 2.0, range_min = 0, range_max = 100 }

[inputs]
    # Adafruit DHT11 data input
    [inputs.dht11]
        _module_path = "brokkr.inputs.adafruitonewire"
        _class_name = "AdafruitOnewireInput"
        _dependencies = []
        name = "DHT11 Temp & Humidity Sensor"
        sensor_module = "adafruit_dht"
        sensor_class = "DHT11"
        data_types = [
            "dht11_temperature",
            "dht11_humidity",
        ]

    # Adafruit DHT22 data input
        [inputs.dht22]
        _module_path = "brokkr.inputs.adafruitonewire"
        _class_name = "AdafruitOnewireInput"
        _dependencies = []
        name = "DHT22 Temp & Humidity Sensor"
        sensor_module = "adafruit_dht"
        sensor_class = "DHT22"
        data_types = [
            "dht22_temperature",
            "dht22_humidity",
        ]
```

mjolnir-yoursystem/presets/**/*.preset.toml

Parameters for the whole device preset

`Metadata` follows `metadata.toml`

`Type_presets` allow sharing of common attributes between `data_types`

`Data_types` define key properties and metadata for each data value

`Input/Process/Output` presets

Minimal Input Example

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
1  # Example of minimal (no frills) preset for the BME280 for example purposes
2  config_version = 1
3  name = "adafruit_bme280_minimal"
4
5  [data_types]
6      bme280_temperature = { full_name = "Temperature (BME280)", binary_type = "f", unit = "C", uncertainty = 1.0,
       custom_attrs.attribute_name = "temperature", digits = 3 }
7      bme280_pressure = { full_name = "Pressure (BME280)", binary_type = "f", unit = "hPa", uncertainty = 1.0,
       custom_attrs.attribute_name = "pressure", digits = 3 }
8      bme280_humidity = { full_name = "Humidity (BME280)", binary_type = "f", unit = "%", uncertainty = 3.0,
       custom_attrs.attribute_name = "humidity", digits = 3 }
9
10 [inputs]
11     [inputs.bme280_i2c]
12         _module_path = "brokkr.inputs.adafruiti2c"
13         _class_name = "AdafruitI2CInput"
14         _dependencies = [ "adafruit-circuitpython-bme280" ]
15         sensor_module = "adafruit_bme280"
16         sensor_class = "Adafruit_BME280_I2C"
17         data_types = [ "bme280_temperature", "bme280_pressure", "bme280_humidity" ]
18
```

Minimal Output Example

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
1  # Example minimal device preset demonstrating a sample output preset
2  config_version = 1
3  name = "example_output_minimal"
4
5  [outputs]
6      [outputs.tsv_file]
7          _module_path = "brokkr.outputs.csvfile"
8          _class_name = "CSVFileOutput"
9          csv_kwargs.delimiter = "\t"
10         extension = "tsv"
11         name = "TSV File Output"
12         output_path = "data"
13
```

Mjolnir Plugins

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

`mjolnir-yoursystem/plugins/*.py`

Loaded from file or an installed Python module

`Superclass` a subclass of PipelineStep

`__init__` method can take arbitrary arguments specified in presets/steps table; just needs to pass `**kwargs` to `super().__init__()`

`Main logic` typically the one abstract method the superclass contains and needs to be overridden by the plugin class

```python
"""
Example lightweight input plugin for the Mjolnir-Config-Template.
"""

# Standard library imports
import random

# Local imports
import brokkr.pipeline.baseinput


class ExampleInputPlugin(brokkr.pipeline.baseinput.ValueInputStep):
    """Simulate an example Brokkr input plugin with random data."""

    def __init__(
            self,
            seed=None,
            na_chance=0.5,
            error_chance=0,
            **value_input_kwargs):
        """
        Simulate an example Brokkr input plugin with random data.

        Parameters
        ----------
        seed : int, str, bytes, bytearray or None, optional
            Seed to initialize the random number generator, as described in
            the docs for random.seed.
            The default is None, which uses the current system time.
        na_chance : float, optional
            The probability that a given data value will be NA.
            The default is 0.5.
        **value_input_kwargs : **kwargs
            Keyword arguments to pass to the ValueInputStep constructor.

        Returns
        -------
        None.

        """
        super().__init__(binary_decoder=False, **value_input_kwargs)
        # YOUR __INIT__ CODE HERE
        self._na_chance = na_chance
        self._error_chance = error_chance

        random.seed(seed)

    def read_raw_data(self, input_data=None):
        """
        Read in raw data from an example random data source.

        Parameters
        ----------
        input_data : any, optional
            Per iteration input data passed to this function from previous
            PipelineSteps. Not used here but retained for compatibility with
            the generalized PipelineStep API. The default is None.

        Returns
        -------
        raw_data : list of [float or None]
            Output raw data as read by this function.

        """
        self.logger.debug("Reading example data")

        raw_data = []
        for data_type in self.data_types:
            # YOUR DATA READ CODE HERE
            try:
                raw_data_value = random.random()
                if random.random() < self._na_chance:
                    raise RuntimeError("Error reading example data!")
            except Exception as e:
                if random.random() < self._error_chance:
                    self.logger.error(
                        "%s getting data value %s on step %s: %s",
                        type(e).__name__, data_type.full_name, self.name, e)
                    self.logger.info("Error details:", exc_info=True)
                raw_data_value = None
            raw_data.append(raw_data_value)

        return raw_data
```

Transform Plugin Example

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

brokkr.pipelines.base.PipelineStep

execute(self, input_data=None)

Returns arbitrary object per contract w/next steps

```python
"""Example ultra-minimal transform plugin for the Mjolnir-Config-Template."""

# Local imports
import brokkr.pipeline.base


class ExampleTransformPlugin(brokkr.pipeline.base.TransformStep):
    def __init__(self, **transform_step_kwargs):
        # Pass arguments to superclass init
        super().__init__(**transform_step_kwargs)

        # YOUR __INIT__ CODE HERE
        self._previous_data = None

    def execute(self, input_data=None):
        # YOUR EXECUTE CODE HERE
        if self._previous_data is None:
            self._previous_data = input_data
        if (input_data['example_bool'].value
                != self._previous_data['example_bool'].value):
            self.logger.info("Zoinks! Example_bool changed from %s to %s",
                             input_data['example_bool'].value,
                             self._previous_data['example_bool'].value)
        self._previous_data = input_data

        # Passthrough the input for consumption by any further steps
        return input_data
```

```toml
# Example ultra-minimal device preset demonstrating a sample transform preset
config_version = 1
name = "example_transform_minimal"

[transforms]
    [transforms.alerts]
        _module_path = "example_transform_minimal"
        _class_name = "ExampleTransformPlugin"
        _is_plugin = true
        name = "Example TransformStep Alert Plugin"
```

Output Plugin Example

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

brokkr.pipelines.baseoutput.FileOutputStep

write_file(self, input_data, output_file_path)

Returns None

```python
"""Example output plugin demonstrating a custom file format."""

# Local imports
import brokkr.pipeline.baseoutput


class ExampleFileOutput(brokkr.pipeline.baseoutput.FileOutputStep):
    def __init__(self, extension="txt", **file_kwargs):
        super().__init__(extension=extension, **file_kwar
        # YOUR INIT CODE HERE

    def write_file(self, input_data, output_file_path):
        # YOUR FILE WRITING CODE HERE
        with open(output_file_path, mode="a",
                  encoding="utf-8", newline="\n") as output_file:
            try:
                output_data = {
                    key: value.value for key, value in input_data.items()}
            except AttributeError:  # If input data is a dict already
                output_data = input_data
            output_file.write(repr(output_data) + "\n")
```

```
# Example minimal device preset
demonstrating a sample output preset
config_version = 1
name = "example_output_minimal"

[outputs]
    [outputs.repr_file]
        _module_path = "example_output"
        _class_name = "ExampleFileOutput"
        _is_plugin = true
        name = "Example File Output Plugin"
        extension = "txt"
```

ValueInput Plugin Example

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

brokkr.pipelines.baseinput.ValueInputStep

read_raw_data(self, input_data=None)

Returns list of raw data type

```python
"""Example minimal input plugin for the Mjolnir-Config-Template."""

# Local imports
import brokkr.pipeline.baseinput


class ExampleMinimalInput(brokkr.pipeline.baseinput.ValueInputStep):
    def __init__(
            self,
            example_argument=True,
            **value_input_kwargs):
        super().__init__(binary_decoder=False, **value_input_kwargs)

        # YOUR INIT LOGIC AND ARGUMENT HANDLING HERE
        self._example_attribute = example_argument

    def read_raw_data(self, input_data=None):
        # YOUR DATA READING LOGIC HERE
        if not self._example_attribute:
            return None
        raw_data = []
        for data_type in self.data_types:
            try:
                raw_data_value = data_type.example_value
            except Exception as e:
                self.logger.error("%s occurred: %s", type(e).__name__, e)
                raw_data_value = None
            raw_data.append(raw_data_value)
        return raw_data
```

```toml
# Example minimal device preset demonstrating a plugin input preset
config_version = 1
name = "example_input_minimal"

[data_types]
    example_str = { binary_type = "s", custom_attrs.example_value = "Test" }
    example_int = { binary_type = "i", custom_attrs.example_value = 42 }

[inputs]
    [inputs.plugin]
        _module_path = "example_input_minimal"
        _class_name = "ExampleMinimalInput"
        _is_plugin = true
        name = "Example Minimal Input Plugin"
        example_argument = true
        data_types = [
            "example_str",
            "example_int",
        ]
```

AttributeInput Plugin Example

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

`brokkr.pipelines.baseinput.AttributeInputStep`

Only needs sensor objects with properties or method attributes

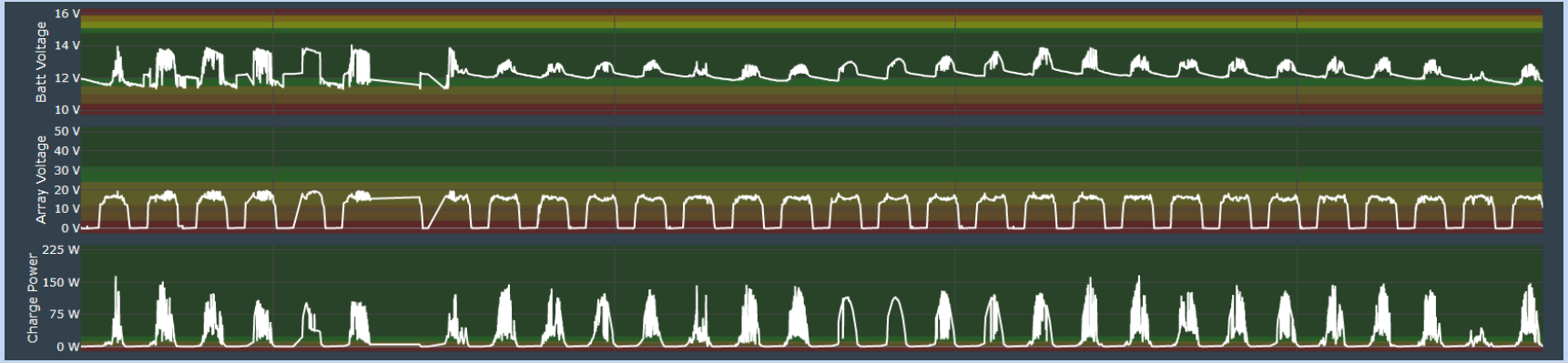
```python
"""Example minimal sensor class plugin for the Mjolnir-Config-Template."""

# Local imports
import brokkr.pipeline.baseinput


class ExampleSensorClass():
    def __init__(self, attribute_dict=None, as_methods=False):
        attribute_dict = {} if attribute_dict is None else attribute_dict
        for attribute_key, attribute_value in attribute_dict.items():
            if as_methods:
                setattr(self, attribute_key, lambda val=attribute_value: val)
            else:
                setattr(self, attribute_key, attribute_value)


class ExampleSensorPropertyInput(brokkr.pipeline.baseinput.PropertyInputStep):
    def __init__(self, **property_input_kwargs):
        super().__init__(
            sensor_class=ExampleSensorClass, **property_input_kwargs)


class ExampleSensorFunctionInput(brokkr.pipeline.baseinput.MethodInputStep):
    def __init__(self, **method_input_kwargs):
        super().__init__(
            sensor_class=ExampleSensorClass, **method_input_kwargs)
```

```toml
# Example minimal device preset demonstrating a sensor input preset
config_version = 1
name = "example_input_sensor_minimal"

[data_types]
    example_sensor_temperature = { full_name = "Temperature (Example)",
    custom_attrs.attribute_name = "temperature", digits = 3 }
    example_sensor_pressure = { full_name = "Pressure (Example)", binary_type =
    "f", unit = "hPa", uncertainty = 1.0, custom_attrs.attribute_name =
    "pressure", digits = 3 }
    example_sensor_humidity = { full_name = "Humidity (Example)", binary_type =
    "f", unit = "%", uncertainty = 3.0, custom_attrs.attribute_name =
    "humidity", digits = 3 }

[inputs]
    [inputs.sensor_method]
        _module_path = "example_input_sensor"
        _class_name = "ExampleSensorPropertyInput"
        _is_plugin = true
        sensor_kwargs.attribute_dict = { temperature = 25, pressure = 1013,
        humidity = 50 }
        sensor_kwargs.as_methods = true
        data_types = [ "example_sensor_temperature", "example_sensor_pressure",
        "example_sensor_humidity" ]
```

Current project status

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- 3 alphas, 3 feature releases, 13 point releases so far
- Has been in use for 6+ months on HAMMA2 sensors
- End to end uptime increased from >95% to >99%
- Test deployments w/ EMESH & experimental sensors



Brokkr roadmap

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- **Brokkr 0.4.0**: System monitoring, IPC queues, more sensors and protocols, async datastreams
- **Brokkr 0.5.0**: Full uplink support; automated setup, deployment & checkout; watchdog service
- **Brokkr 1.0.0**: Unit/CI tests, docs and docstrings; split modular components into packages, polish
- **Brokkr 1.1.0**: Full remote config and command uplink capability, config validation, more core steps

Sindri roadmap

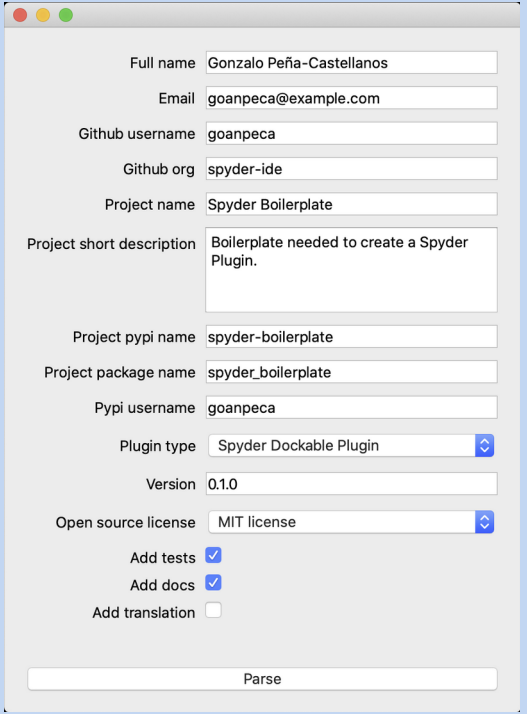
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

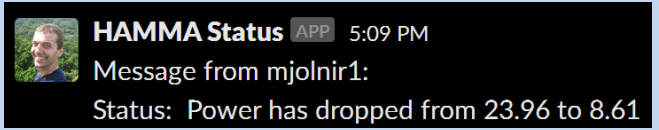
- **Sindri 0.3.0**: Full support for new telemetry and trigger data, and many minor UI/UX improvements
- **Sindri 0.4.0**: Add remote uplink of sensor data, master all-unit dashboard, multi-unit support
- **Sindri 0.5.0**: Port to common Mjolnir multiprocess, pipeline, log and config arch; add high-level API
- **Sindri 1.0.0**: Add unit/CI tests, docs and docstrings
- **Sindri 1.1.0***:* Handle remote settings & commands

What else is next?

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Presets, plugins and SCPs being converted to Cookiecutter templates
- GUI for easily creating a preset/plugin/SCP with minimal boilerplate
- Plugin integration w/the Spyder IDE
- Slack bot, alerts, processing and more in development as Mjolnir plugins
- Comprehensive guide & docs





Long Term Vision

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

The goal of Project Mjolnir isn't just to create a framework—we want to build a *community*, with an ecosystem of contributors, plugins, presets and more, collectively benefiting the cause of low-cost, high-impact, accessible open science for everyone!

But...

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

To reach that lofty goal, we need

YOU!

Interested?

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Whether you're a student, researcher, programmer or anything in between, we've got something for everyone!

- Use it in your work? Please cite us and drop us a line!
- Find a bug? Need a feature? Open an issue!
- Want to add a new device? Write a preset!
- Looking to support a new protocol? Create a plugin!
- Need new core functionality? Contribute on Github!

That's all, folks!

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Any Questions?
Feedback? Suggestions?

CAM.Gerlach@Gerlach.CAM

Additional Slides

Introducing: Me

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Nobody
- Grad student, atmospheric scientist
- Only 3 years Python
- ML systems
- Spyder
- NASA GLM
- Mjolnir

Project Objectives

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Log and uplink power, storage and sensor status to allow early detection and mitigation of problems
- Allow remote access and control of sensors
- Offload data to external drives to enable longer-duration deployments between site visits
- Enable remote de/activation to save power
- Transmit filtered data to allow real-time use

Data to Log/Send

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Charge controller: battery voltage, current usage, temperatures, charging status...
- AGS science packet header: temp, humidity, trigger info, GPS, sensor status, thresholds…
- Health and status packets: Trigger count, disk usage and remaining, dropped packets, errors...
- General: Pi and sensor CPU, RAM, storage, and uptime; connection to modem, internet and sensor

Components Needed

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- Raspberry Pi Model 4B 4GB with suitable case
- USB 4G LTE modem with active SIM card
- Waterproof Cat 5e+ Ethernet to sensor
- RJ-11 → RS-232 → USB for charge controller
- 12V-5V buck converter w/USB out to power RPi
- Remote machine (at NSSTC or VPS) for server

Additional Components

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

- 2 x 4TB portable HDDs (e.g. WD My Passport) for data offloading, connected via USB 3.0 to RPi
- GPIO relay on-line to sensor, to allow the RPi to power it up and down remotely
- USB 3.0 hub with separate 12V in to power HDDs
- Gigabit Ethernet USB adapter for direct connection