Writing Portable Building Analytics with the Brick Metadata Schema

Gabe Fierro, UC Berkeley
ACM E-Energy
gtfierro@eecs.berkeley.edu

Outline

1. Lecture on Brick Schema
2. Overview of Mortar platform
3. Guided exploration of Mortar with online tutorial

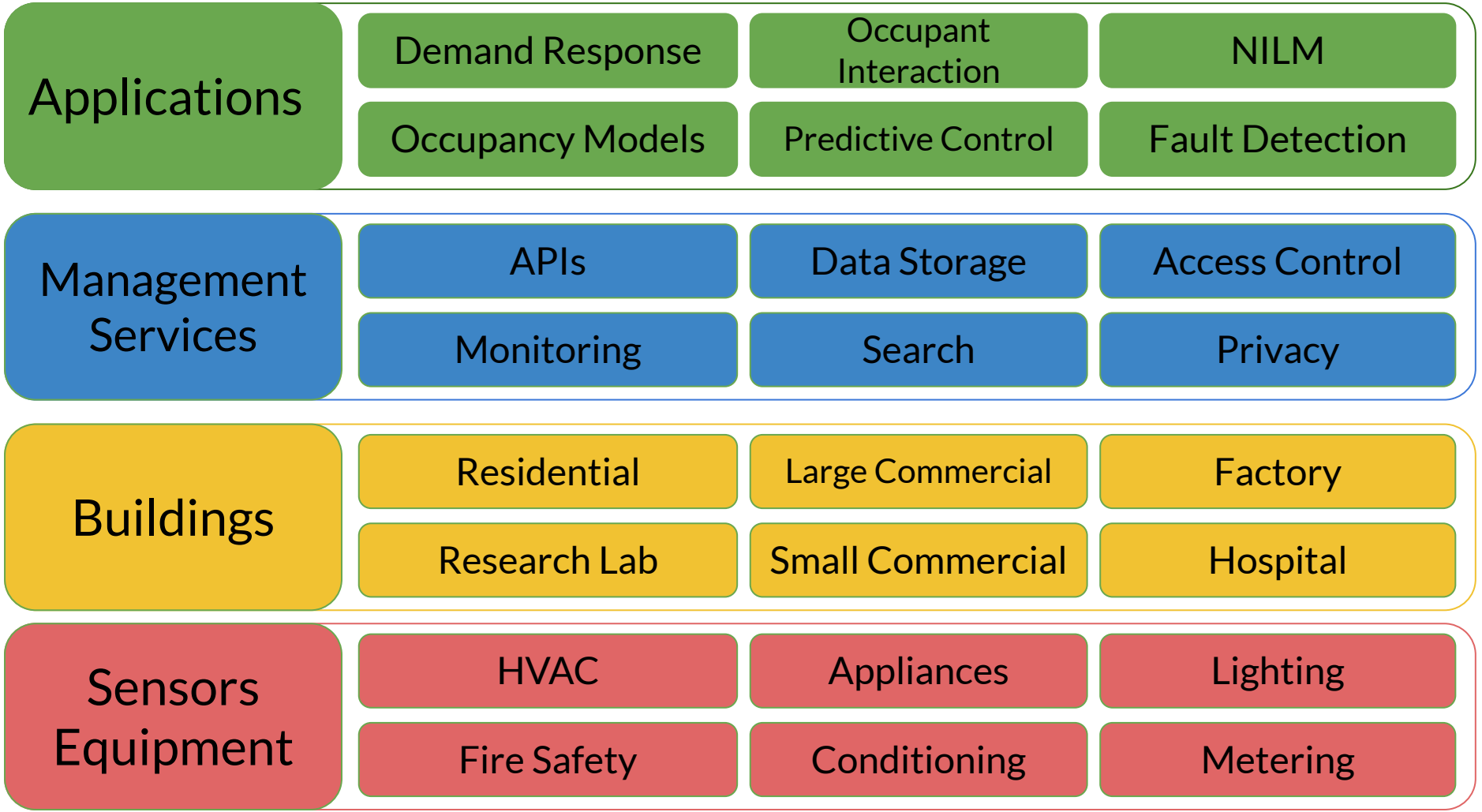
Resources

Brick Website
https://brickschema.org/

Mortar Website
https://mortardata.org/

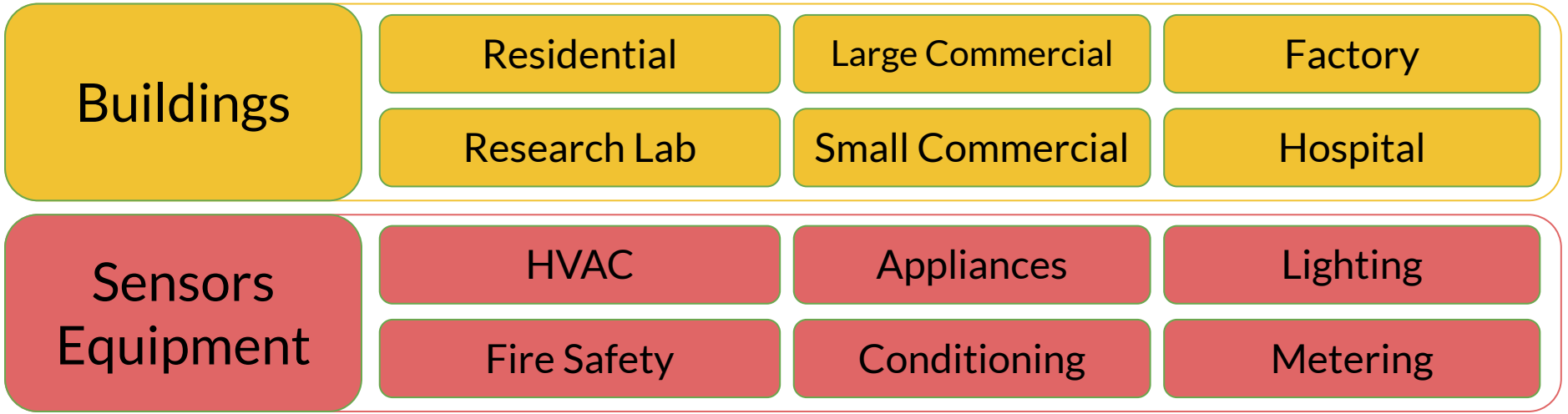
Tutorial Page
https://tutorial.mortardata.org/

Brick Explore Tool
https://querybuilder.mortardata.org/


Applications
Demand Response
Occupant Interaction
NILM
Occupancy Models
Predictive Control
Fault Detection
Management Services
APIs
Data Storage
Access Control
Monitoring
Search
Privacy
Buildings
Residential
Large Commercial
Factory
Research Lab
Small Commercial
Hospital
Sensors Equipment
HVAC
Appliances
Lighting
Fire Safety
Conditioning
Metering

Heterogeneity

- Different BMS, equipment vendors
- Custom-designed controls, systems, architecture
- All of this changes over time

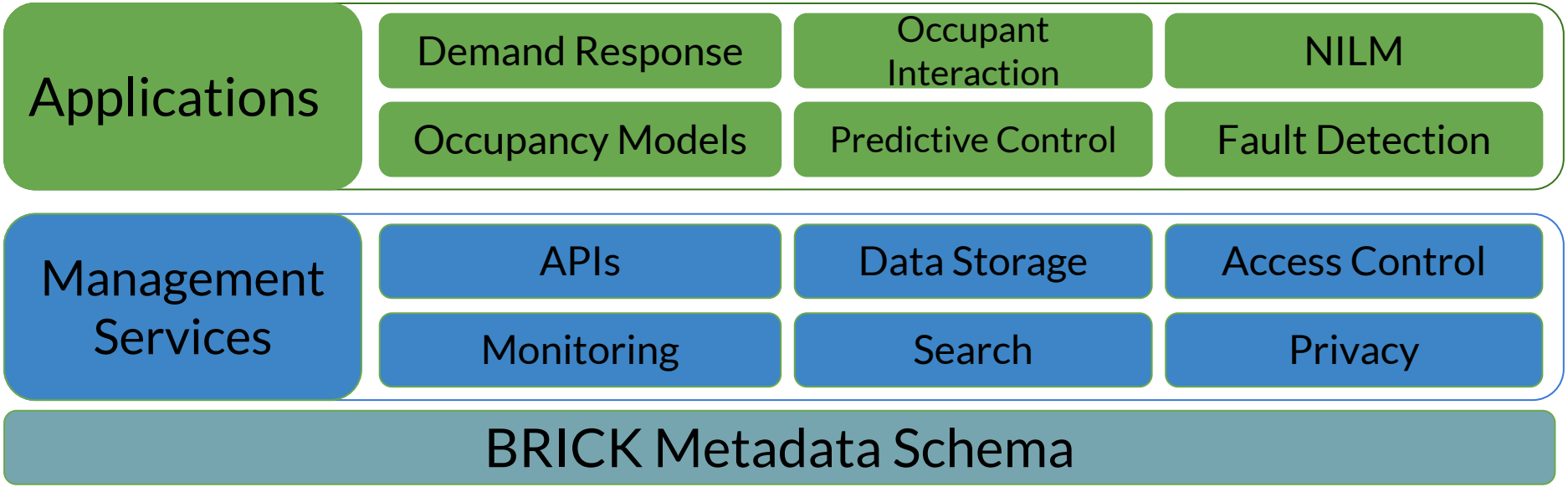


Existing Metadata Schemata

- Survey 90 apps from building science literature
- Decompose apps into “things” and “relationships”
- Can existing schemas describe what’s needed?
- **NO!**

	IFC	SSN	Haystack
Tag Coverage	29%	11%	54%
Relationships	n/a	Only spatial	n/a

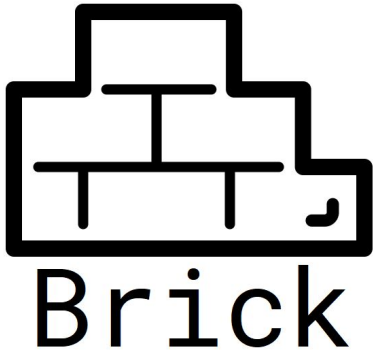
Bhattacharya, Arka, Joern Ploennigs, and David Culler. "Short Paper: Analyzing Metadata Schemas for Buildings: The Good, the Bad, and the Ugly." *BuildSys*, 2015.



- Write applications against BRICK
- Apps become *agnostic* to the underlying hardware
- No more hard-coding points in applications
- “Write once, run anywhere”

Brick Schema

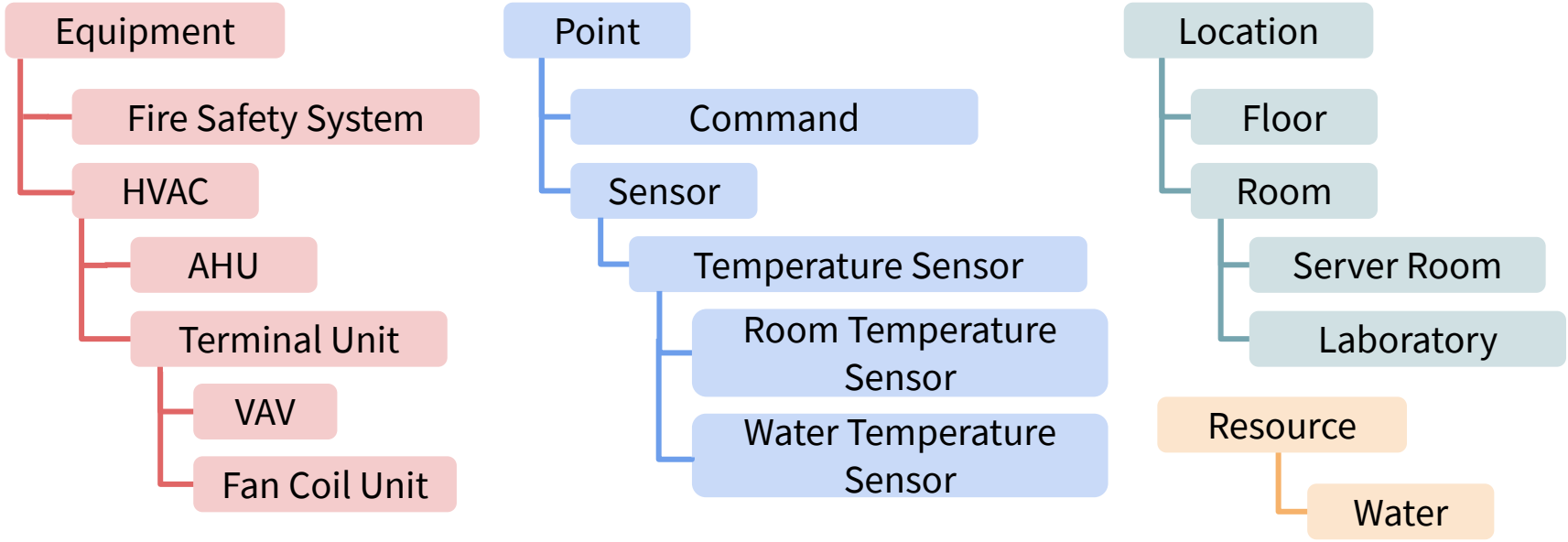
- Graph-based metadata schema for smart buildings
- Capture physical, logical, virtual **entities** in buildings using a **class hierarchy**
- Capture the necessary **relationships** between them
- Use Brick to describe timeseries data and its context



https://brickschema.org

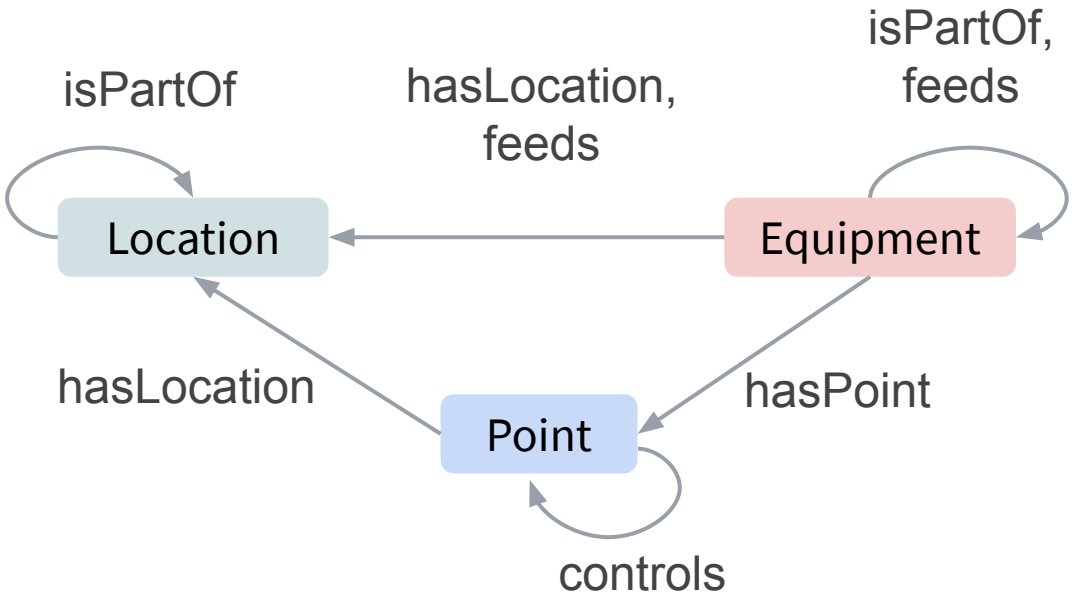
Berkeley UNIVERSITY OF CALIFORNIA
Carnegie Mellon University
Johnson Controls
IBM
UC San Diego
UCLA
SDU
University of Virginia

Brick Schema: Class Structure



- Standardized class structure enables discoverability
- Extensible: allow site/deployment-specific classes

Brick Schema: Relationships



- Relationships capture how “things” relate to each other
- Relationships can be transitive, symmetrical
- Help Brick extend to cover new settings, equipment

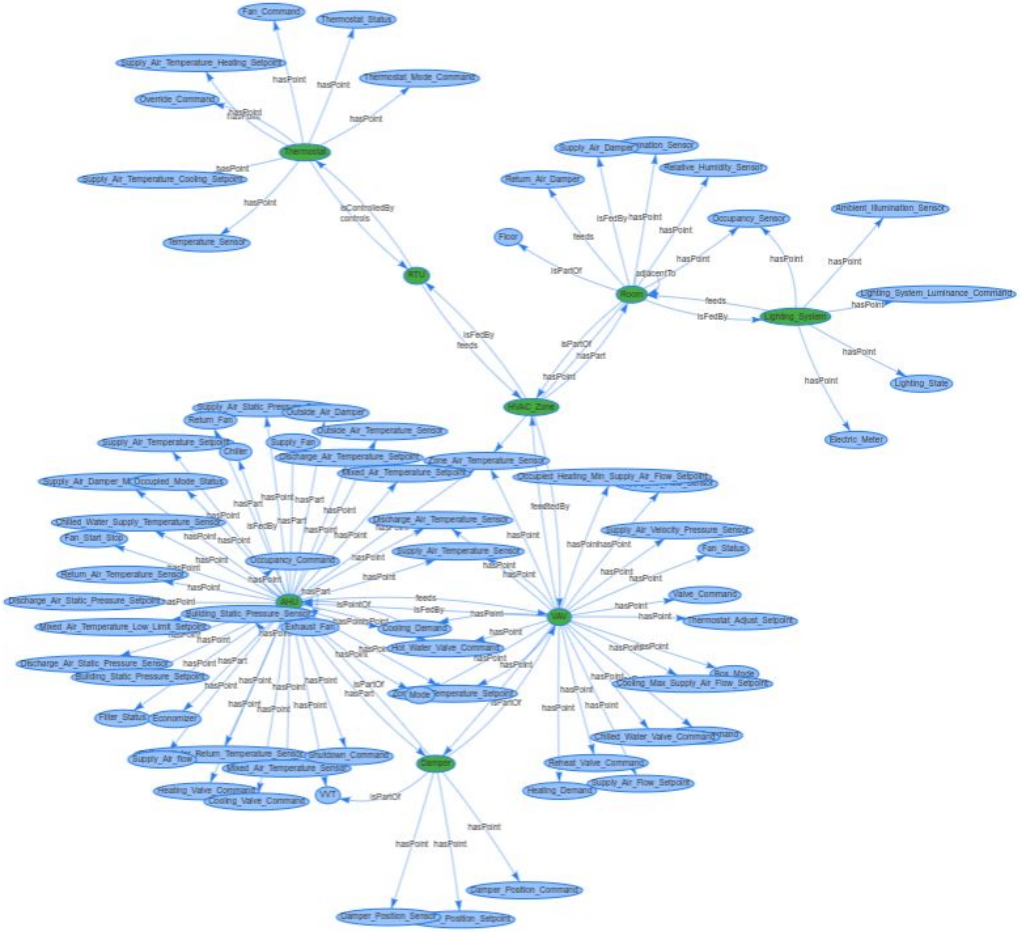
Brick Schema: Relationships

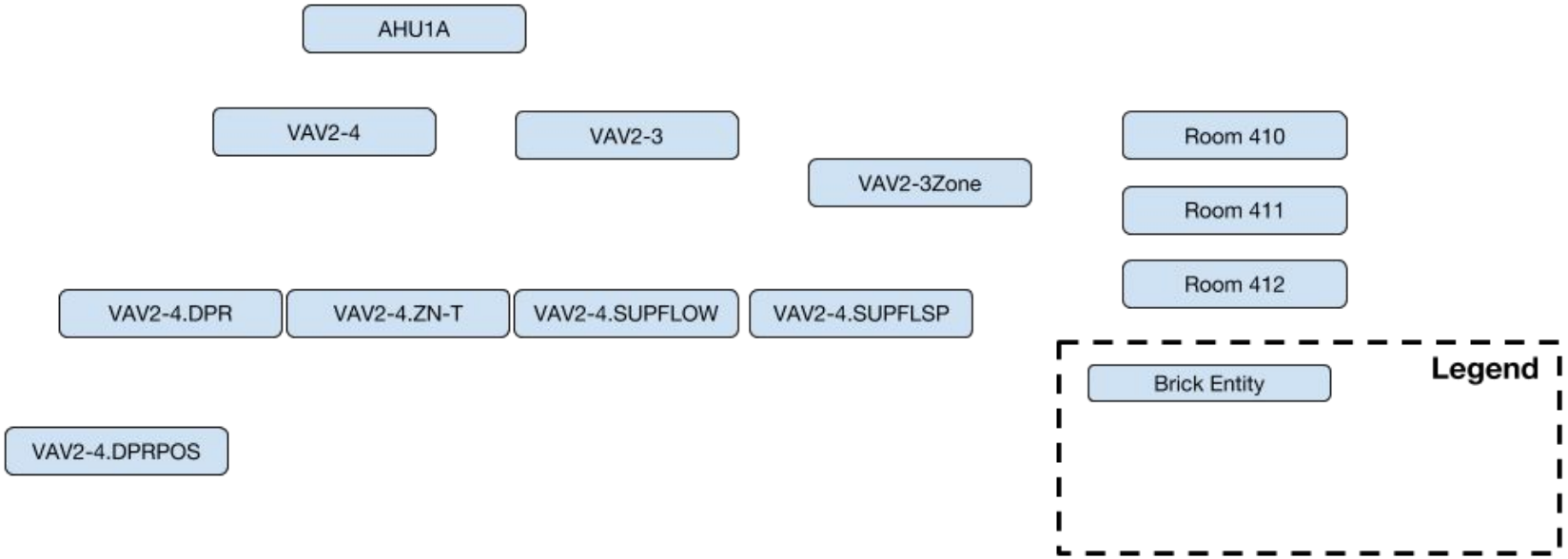
Relationship	Definition
`contains/isLocatedIn`	Spatial location of the subject: room, building, campus, city chain
`controls/isControlledBy`	Identifies the affected subject of some control block (probably a functional block)
`hasPart/isPartOf`	Mechanical composition
`hasPoint/isPointOf`	Associates measurement points/timeseries with the entity it measures for
`feeds/isFedBy`	The passage of some medium: light, air, power, water, etc
`type/isTypeOf`	Instantiates “things”

A Brick Model is a Graph

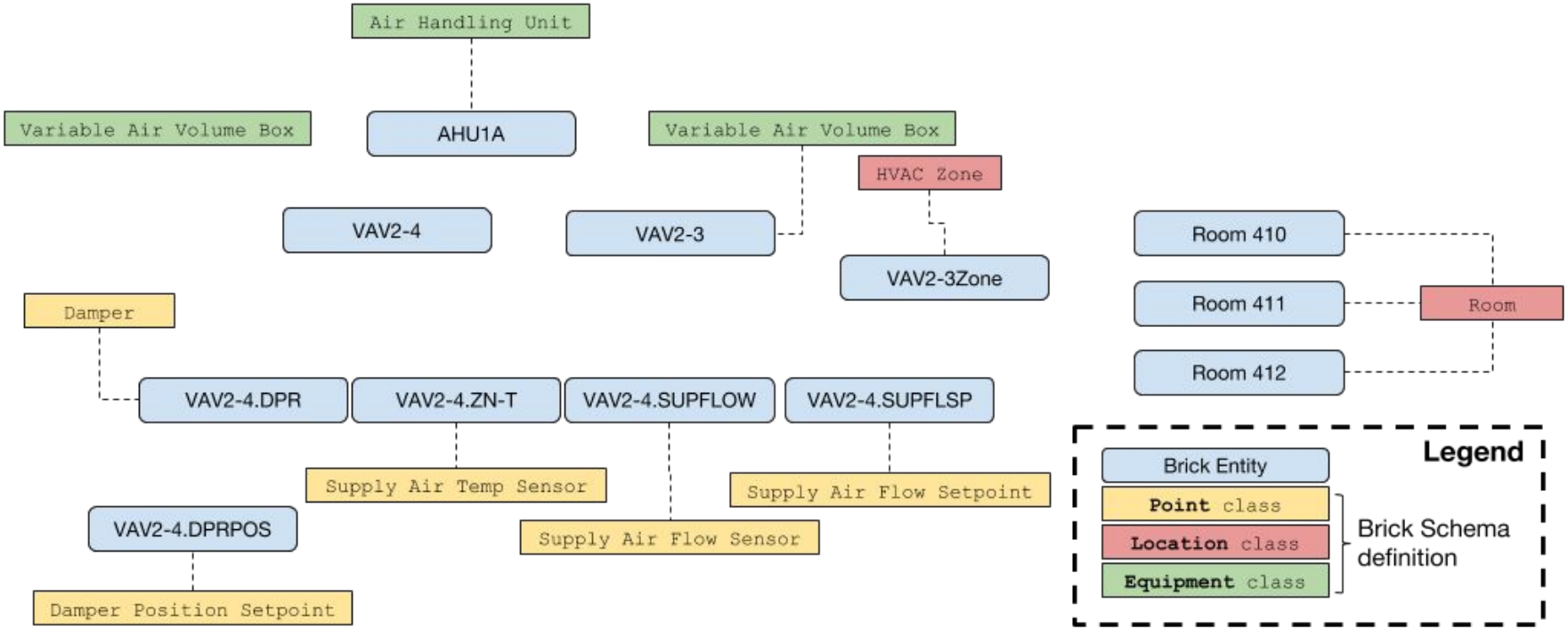
- Nodes = “things”
  - Building assets
  - Equipment
  - Subsystems
  - Class structure

- Edges = “relationships”
  - Location
  - Control
  - Connectivity
  - Composition
  - etc

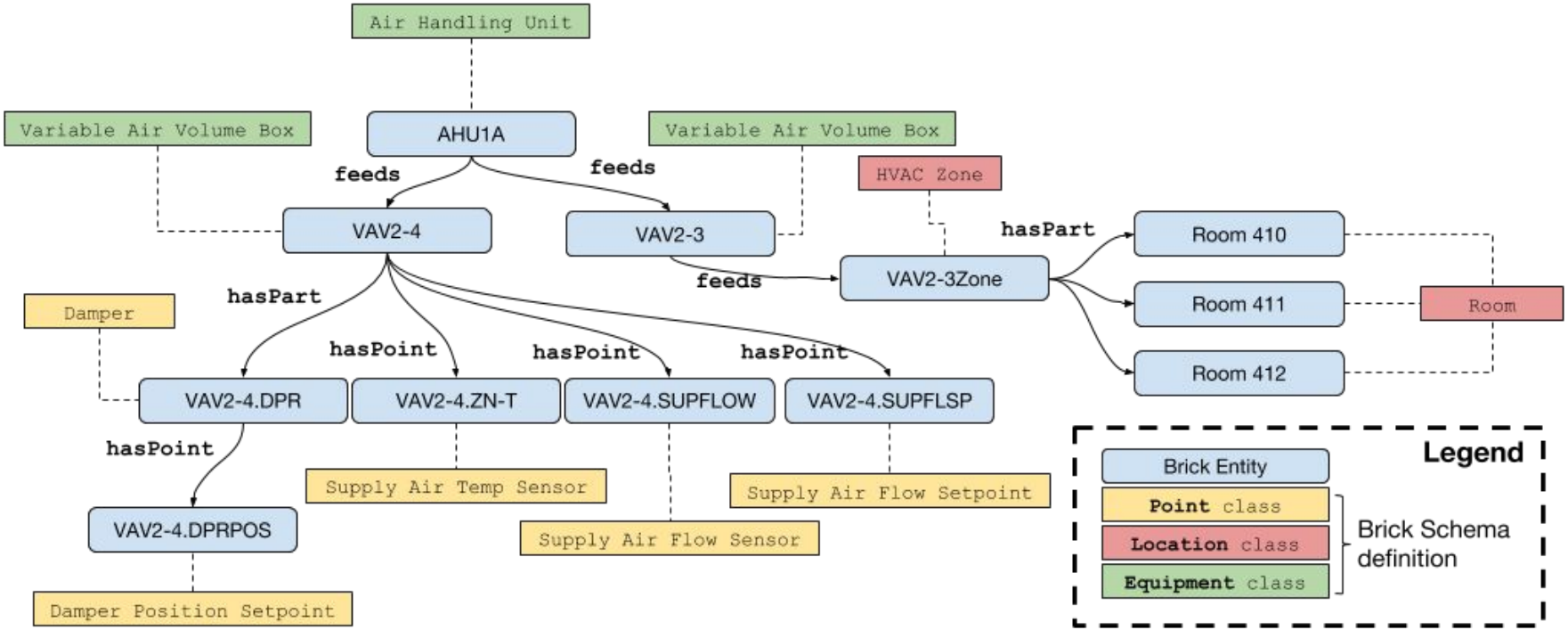




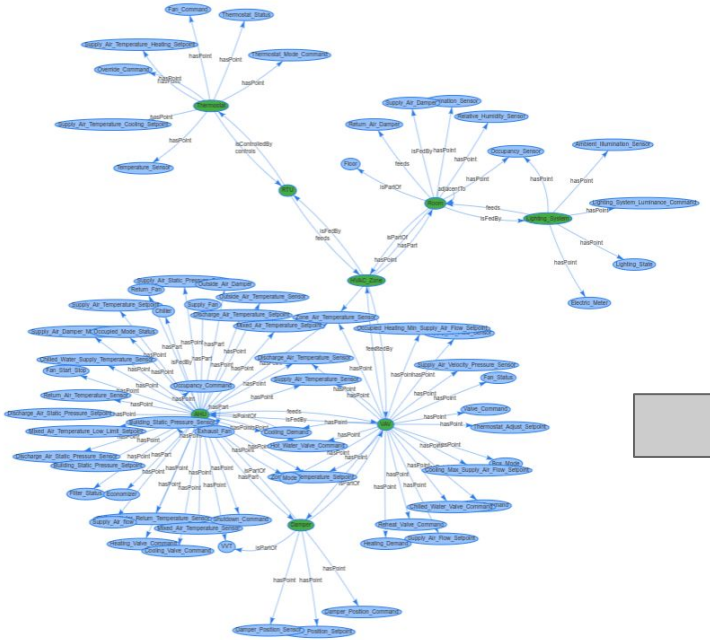
- Have a set of physical, virtual "*things*" and points that an application wants to refer to



- Have a set of physical, virtual "*things*" and points that an application wants to refer to
- Brick defines a **hierarchical class structure** to define **standard names** for equipment, points, locations, etc

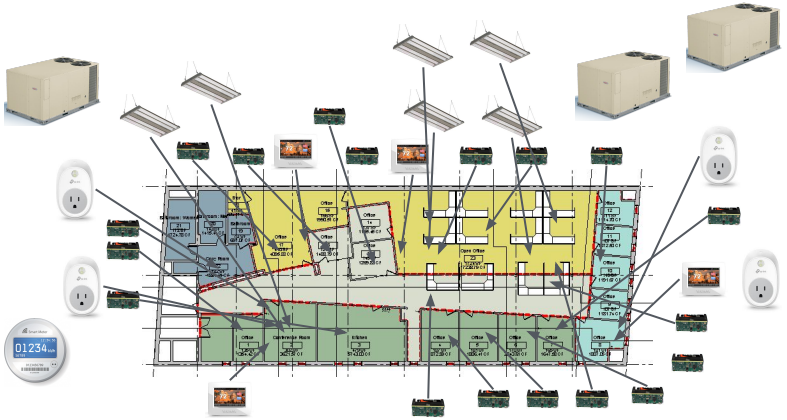


- Have a set of physical, virtual "*things*" and points that an application wants to refer to
- Brick defines a **hierarchical class structure** to define **standard names** for equipment, points, locations, etc
- Brick defines a set of **standard relationships** that describe how things are **connected**



Brick model

represents



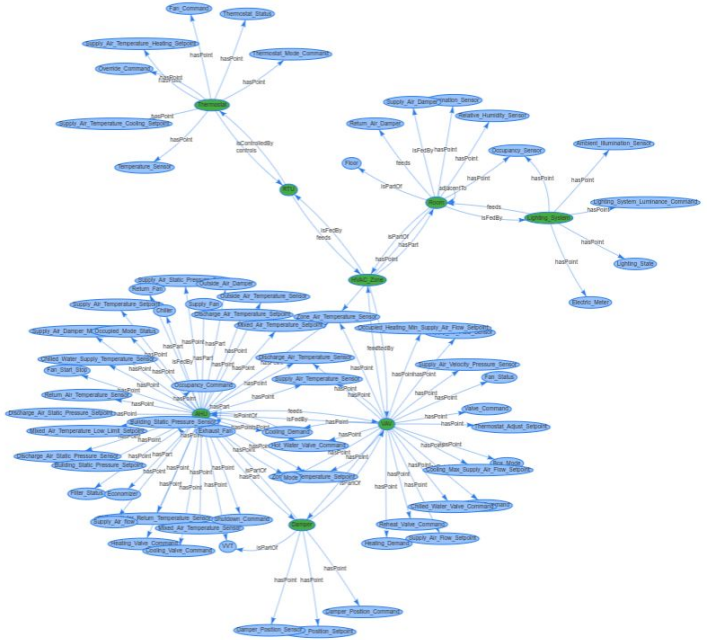
A **Brick model** represents the **assets** and **relationships** and **data** in a building

Query

```
SELECT ?tstat ?zone ?tstat_uri ?state_uuid ?temp_uuid ?weather_uuid FROM ciee WHERE {
    ?zone rdf:type brick:HVAC_Zone .
    ?rtu bf:feeds+ ?zone .

    ?tstat rdf:type/rdfs:subClassOf* brick:Thermostat .
    ?tstat bf:controls+ ?rtu .
    ?tstat bf:uri ?tstat_uri.

    ?tstat bf:hasPoint ?state .
    ?state rdf:type brick:Thermostat_HVAC_Operation_Status .
    ?state bf:uuid ?state_uuid .

    ?tstat bf:hasPoint ?temp .
    ?temp rdf:type/rdfs:subClassOf* brick:Temperature_Sensor .
    ?temp bf:uuid ?temp_uuid .

    ?tstat bf:hasSite ?site .
    ?weather_sensor bf:hasSite ?site .
    ?weather_sensor rdf:type/rdfs:subClassOf* brick:Temperature_Sensor .
    ?weather_sensor bf:uuid ?weather_uuid
};
```

Application

queries



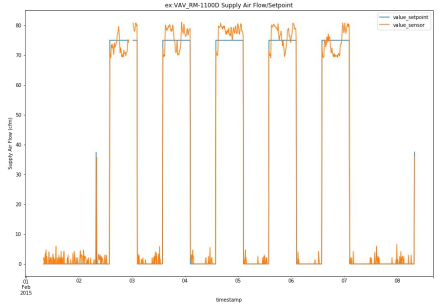
Brick model

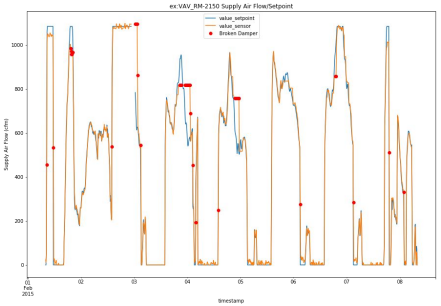
An application **queries** a Brick model to retrieve the data + configuration it needs

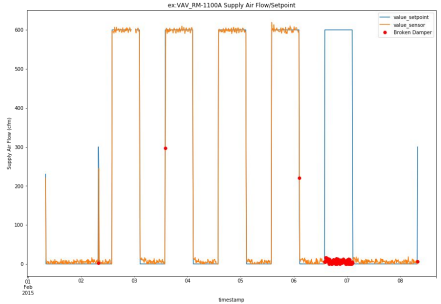
Query

```
SELECT ?tstat ?zone ?tstat_uri ?state_uuid ?temp_uuid ?weather_uuid FROM ciee WHERE {
    ?zone rdf:type brick:HVAC_Zone .
    ?rtu bf:feeds+ ?zone .

    ?tstat rdf:type/rdfs:subClassOf* brick:Thermostat .
    ?tstat bf:controls+ ?rtu .
    ?tstat bf:uri ?tstat_uri.

    ?tstat bf:hasPoint ?state .
    ?state rdf:type brick:Thermostat_HVAC_Operation_Status .
    ?state bf:uuid ?state_uuid .

    ?tstat bf:hasPoint ?temp .
    ?temp rdf:type/rdfs:subClassOf* brick:Temperature_Sensor .
    ?temp bf:uuid ?temp_uuid .

    ?tstat bf:hasSite ?site .
    ?weather_sensor bf:hasSite ?site .
    ?weather_sensor rdf:type/rdfs:subClassOf* brick:Temperature_Sensor .
    ?weather_sensor bf:uuid ?weather_uuid
};
```

executes







Queries allow apps to account for building heterogeneity and **customize their operation** to each building.

This is called **application portability**

Outline

1. ~~Lecture on Brick Schema~~
2. Overview of Mortar platform
3. Guided exploration of Mortar with online tutorial

Resources

Brick Website
https://brickschema.org/

Mortar Website
https://mortardata.org/

Tutorial Page
https://tutorial.mortardata.org/

Brick Explore Tool
https://querybuilder.mortardata.org/

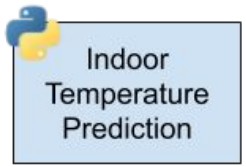
How do we build a platform that enables reproducible building research?

Why are Portable Applications Hard to Write?

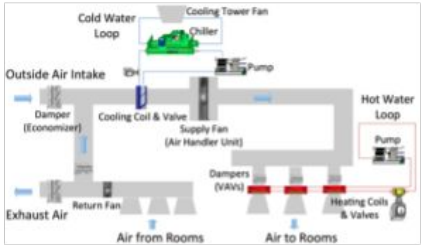
- Hardcoded point names:
  - *Lights in this building are* `WS86004.RELAYXX`
- Assumptions about building subsystems
  - *This building has room-level temperature sensors*
- Tightly-coupled phases of operation:
  - Downloading, cleaning data often specific to the building at hand
- Access to different buildings:
  - Why write portably if you don't have to?
- How to name, describe and discover data
  - **Brick** gives us this abstraction: describe data in terms of types and relationships

Mortar Platform Contributions

- Open data set of buildings with timeseries data and Brick models
- Modular, extensible architecture for portable analytics applications
- Data platform for the development, execution and evaluation of portable analytics

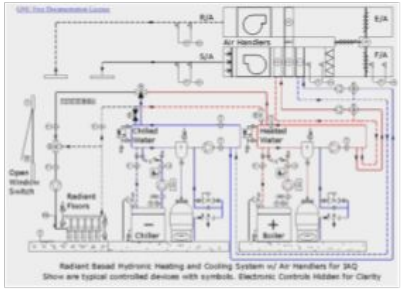


Indoor Temperature Prediction (AHU-based)



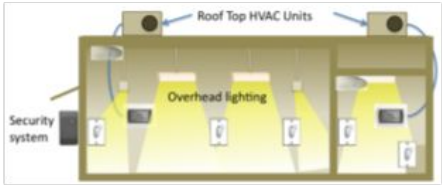
Air-based HVAC Systems

Indoor Temperature Prediction (Radiant)



Radiant Heating/Cooling Systems

Indoor Temperature Prediction (RTU-basedl)

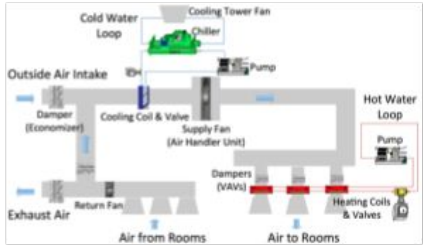


Rooftop Unit Systems

Indoor Temperature Prediction (Building 1)

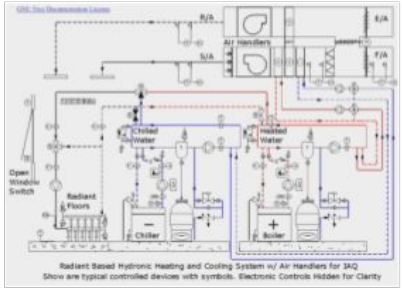
Indoor Temperature Prediction (Building 2)

Custom Implementations



Air-based HVAC Systems

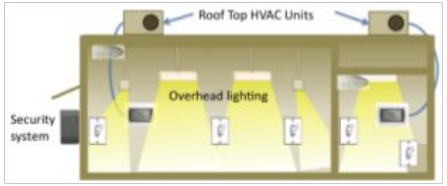
Indoor Temperature Prediction (Building 3)



Radiant Heating/Cooling Systems

Indoor Temperature Prediction (Building 4)

Indoor Temperature Prediction (Building 5)

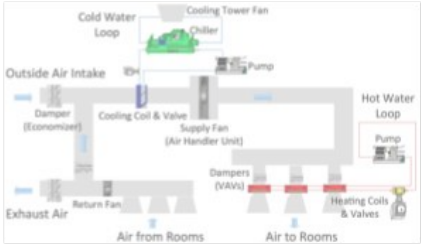


Rooftop Unit Systems

Indoor Temperature Prediction (Building 1)

Custom Implementations

Indoor Temperature Prediction (Bu



Systems

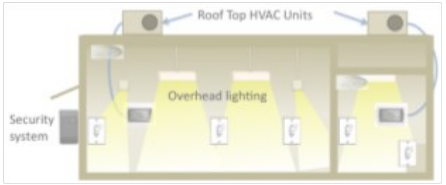
Facilitating reproducibility means reducing the effort of implementation/evaluation across contexts:

1. Expressive description of data + context (Brick)
2. Large, diverse dataset
3. Application-specific normalization of data

ing Systems

Indoor Temperature Prediction (Building 4)

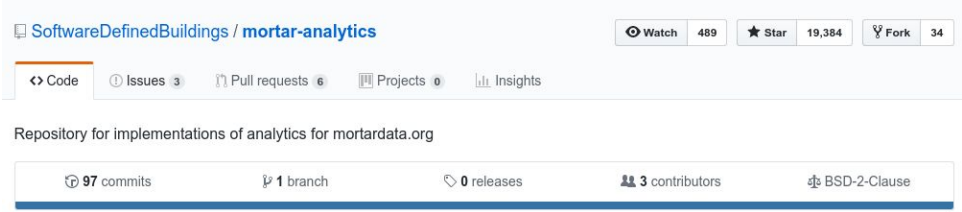
Indoor Temperature Prediction (Building 5)

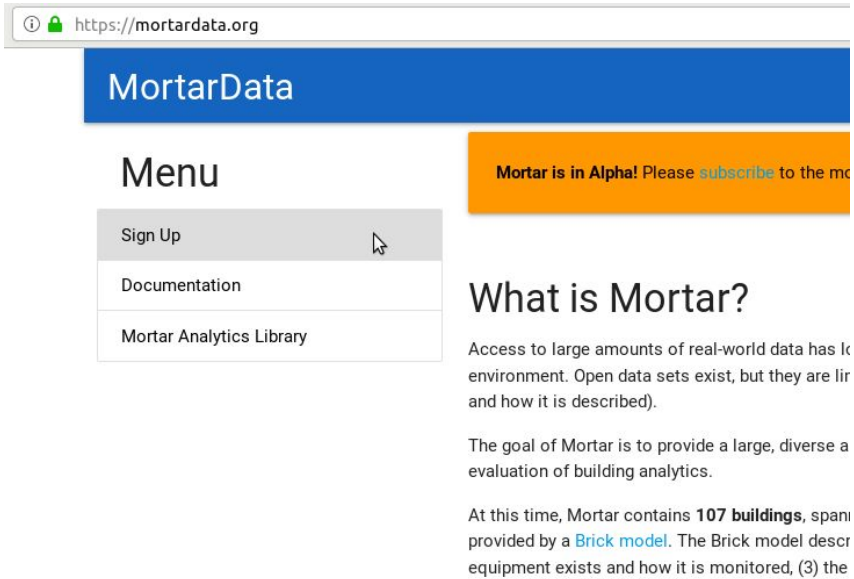


Rooftop Unit Systems

Open Dataset

- Live data for > 100 buildings
  - Up to 6 years of data
- Contextualized timeseries data
  - Each building has a Brick model
- Public API for data access
  - `pymortar` Python library
- Open source library of analytics applications





Dataset Overview

- **107** buildings
- **> 13.5 billion** data points
- **26,000** data streams

- All data streams described in per-site Brick models

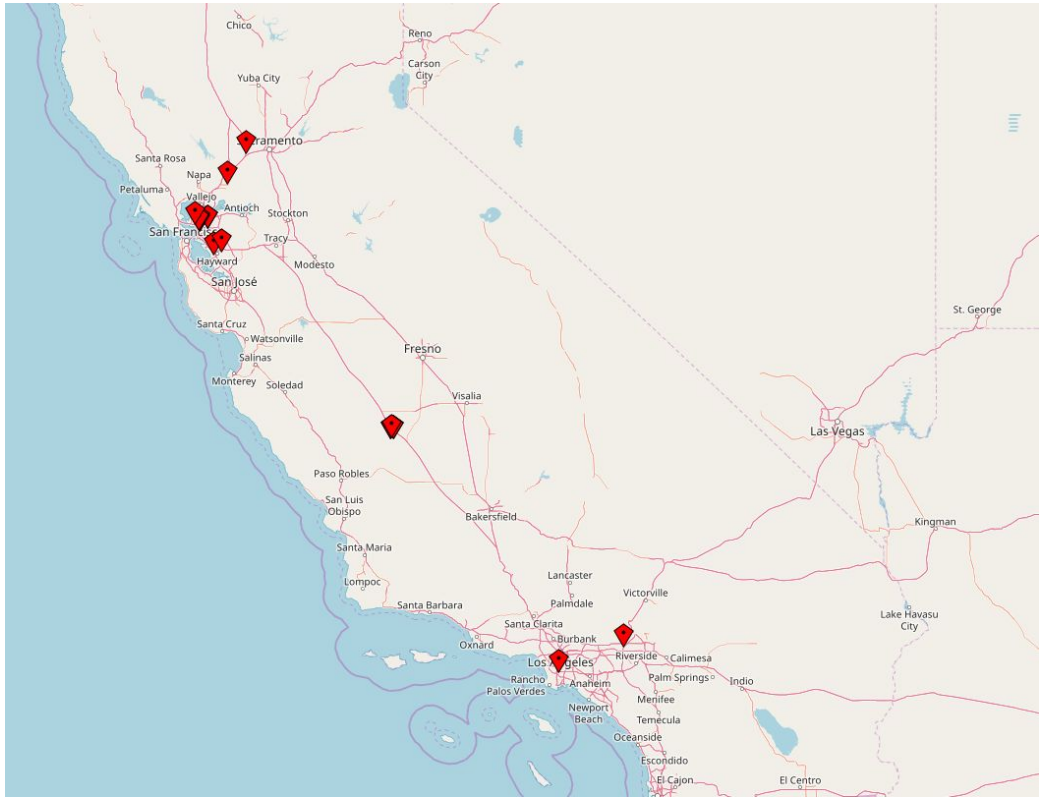
- **All data available on https://mortardata.org/**

Temperature Sensor	7380
Luminance Sensor	257
Occupancy Sensor	445
Pressure Sensor	148
Outside Air Temperature Sensor	362
Cloud Cover	32
Setpoints (generic)	2331
Power Meters	77
VAVs	4724
AHUs	467
HVAC Zones	4887
Dampers	1662
Non BMS Thermostats	123

Dataset Overview

- UC Davis campus data
  - ~ 90 buildings (not all with data)
  - 3-6 years of data
  - ~15min collection interval
  - HVAC system, building meter

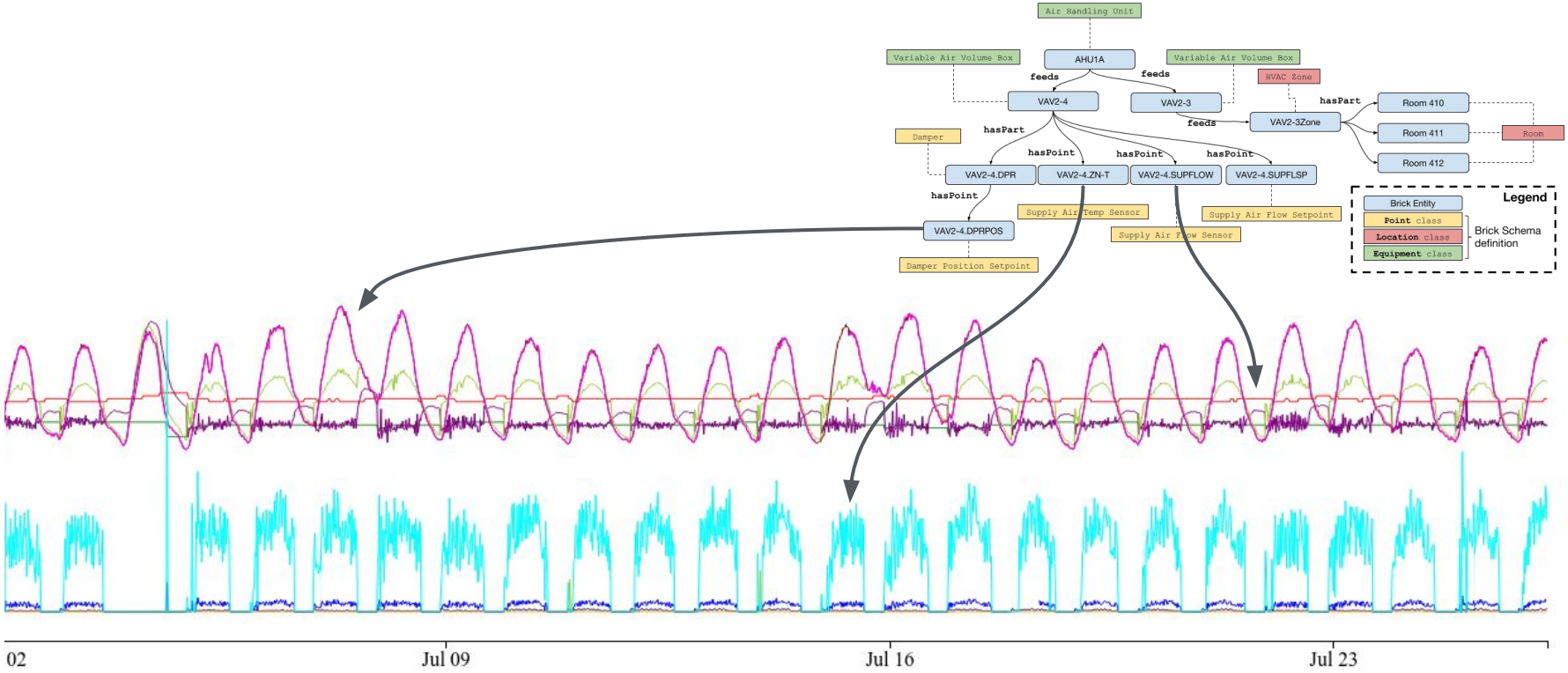
- XBOS-DR/V data
  - 15 RTU-based buildings
  - 1-9 months of data
  - ~10-30sec collection interval
  - HVAC, Lighting, building meter, occupancy, PV, EVSE



API for Heterogeneous Data

- How do apps actually make use of this data?
- Need data representations that are application-appropriate
  - Simplify expressive and semantically rich underlying data model
- Application-specific dataset as a **materialized view**
  - Expressed as queries over Brick metadata + timeseries data

Contextualizing Timeseries Data



Two Flavors of “Views”

Metadata (Context)

Timeseries Data

Two Flavors of “Views”

Metadata (Context)

```
temperature_sensors = pymortar.View(
    name="temp_sensors",
    definition="""SELECT ?temp ?room ?zone
        WHERE {
            ?temp rdf:type/rdfs:subClassOf*
                        brick:Temperature_Sensor .
            ?temp bf:isLocatedIn ?room .
            ?room rdf:type brick:Room .
            ?room bf:isPartOf ?zone .
            ?zone rdf:type brick:HVAC_Zone
        };""",
)
```

Declarative specification of app dataset

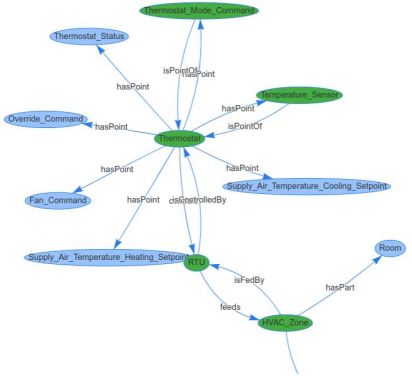
Timeseries Data

```
temperature_streams = pymortar.DataFrame(
    name="temperature_sensor_data",
    aggregation=pymortar.MEAN,
    window="15m",
    timeseries=[
        pymortar.Timeseries(
            view="temp_sensors",
            dataVars=["?temp"],
        ),
    ]
)
```


Two Flavors of “Views”

Metadata (Context)

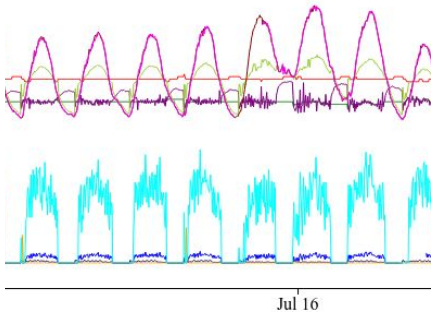
```
temperature_sensors = pymortar.View(
    name="temp_sensors",
    definition="""SELECT ?temp ?room ?zone
        WHERE {
            ?temp rdf:type/rdfs:subClassOf*
                        brick:Temperature_Sensor .
            ?temp bf:isLocatedIn ?room .
            ?room rdf:type brick:Room .
            ?room bf:isPartOf ?zone .
            ?zone rdf:type brick:HVAC_Zone
        };""",
)
```



Execute against underlying data model

Timeseries Data

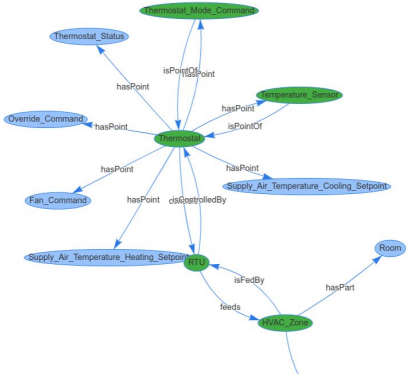
```
temperature_streams = pymortar.DataFrame(
    name="temperature_sensor_data",
    aggregation=pymortar.MEAN,
    window="15m",
    timeseries=[
        pymortar.Timeseries(
            view="temp_sensors",
            dataVars=["?temp"],
        ),
    ]
)
```



Two Flavors of “Views”

Metadata (Context)

```
temperature_sensors = pymortar.View(
    name="temp_sensors",
    definition="""SELECT ?temp ?room ?zone
        WHERE {
            ?temp rdf:type/rdfs:subClassOf*
                        brick:Temperature_Sensor .
            ?temp bf:isLocatedIn ?room .
            ?room rdf:type brick:Room .
            ?room bf:isPartOf ?zone .
            ?zone rdf:type brick:HVAC_Zone
        };""",
)
```

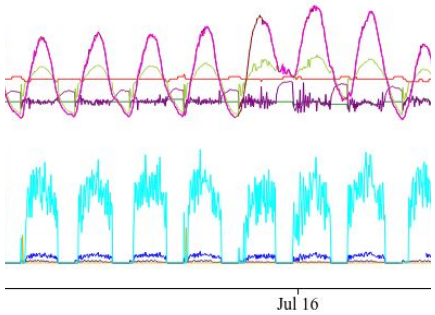


From **SELECT** clause | Autogenerated

temp	room	zone	temp_uuid	site
sen1	Room 410	Zone 4	...-9788bdf63cf0	CompSciHall
sen2	Room 410	Zone 4	...-d7c96010da5c	CompSciHall
sen3	Room 411	Zone 4	...-b1607a419aa4	CompSciHall
sen4	Room 501	Zone 5	...-89f84f4ec34e	CompSciHall
...	...	...	...	...

Timeseries Data

```
temperature_streams = pymortar.DataFrame(
    name="temperature_sensor_data",
    aggregation=pymortar.MEAN,
    window="15m",
    timeseries=[
        pymortar.Timeseries(
            view="temp_sensors",
            dataVars=["?temp"],
        ),
    ]
)
```



Yields app-specific (relational) table

	ffee4857-0a4e-3a18-808d-c94ad6801642	de32b7a8-38ae-322b-a98f-4d4dddd84e37	7d49226d-32a0-328f-b8cb-5c3dddbd40c5	dfde85f(0a94-3e4(9dc(05921a84088
2018-08-31 21:30:00+00:00	1306.0	52.0	105.0	Na
2018-08-31 22:00:00+00:00	1312.0	52.0	105.0	Na
2018-08-31 22:30:00+00:00	1312.0	52.0	105.0	Na
2018-08-31 23:00:00+00:00	1392.0	52.0	105.0	Na
2018-08-31 23:30:00+00:00	1350.0	52.0	105.0	Na

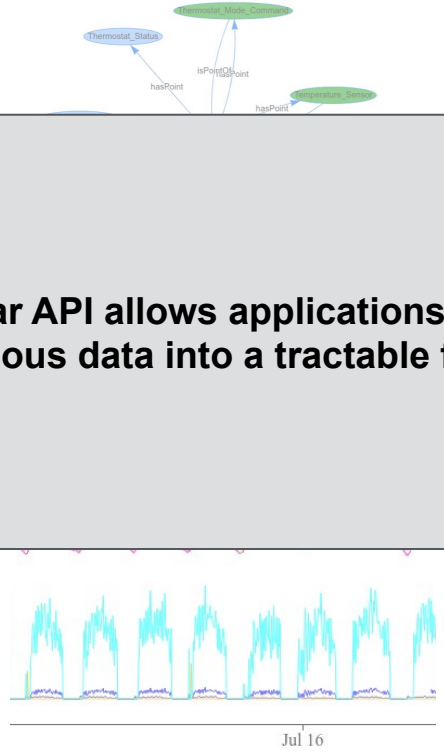
Two Flavors of "Views"

Metadata (Context)

```
temperature_sensors = pymortar.View(
    name="temp_sensors",
    definition="""SELECT ?temp ?room ?zone
        WHERE {
            ?temp rdf:

            ?temp bf:i
            ?room rdf:
            ?room bf:i
            ?zone rdf:
        };""",
)
```

Timeseries Data

```
temperature_streams
    name="temperatu
    aggregation=pymo
    window="15m",
    timeseries=[
        pymortar.Timeseries(
            view="temp_sensors",
            dataVars=["?temp"],
        ),
    ]
)
```



Declarative Mortar API allows applications to project heterogeneous data into a tractable form

From **SELECT** clause | Autogenerated

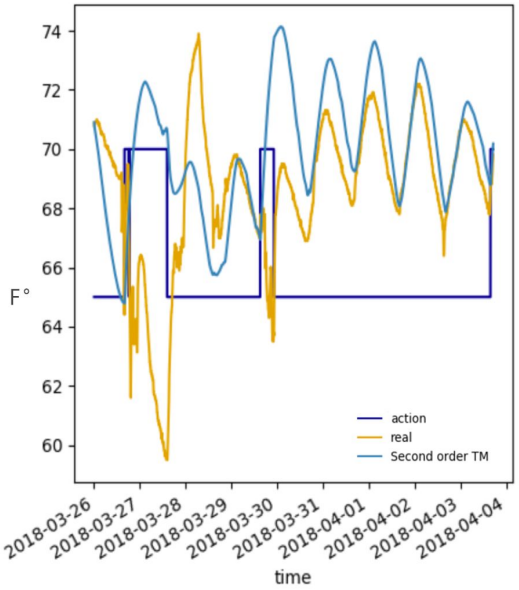
temp	room	zone	temp_uuid	site
n1	Room 410	Zone 4	...-9788bdf63cf0	CompSciHall
n2	Room 410	Zone 4	...-d7c96010da5c	CompSciHall
n3	Room 411	Zone 4	...-b1607a419aa4	CompSciHall
n4	Room 501	Zone 5	...-89f84f4ec34e	CompSciHall
..	...	...	...	...

Yields app-specific (relational) table

	ffee4857-0a4e-3a18-808d-c94ad6801642	de32b7a8-38ae-322b-a98f-4d4dddd84e37	7d49226d-32a0-328f-b8cb-5c3dddbd40c5	dfde85f(0a94-3e4(9dc(05921a8408
08-31 00:00	1306.0	52.0	105.0	Na
2018-08-31 22:00:00+00:00	1312.0	52.0	105.0	Na
2018-08-31 22:30:00+00:00	1312.0	52.0	105.0	Na
2018-08-31 23:00:00+00:00	1392.0	52.0	105.0	Na
2018-08-31 23:30:00+00:00	1350.0	52.0	105.0	Na

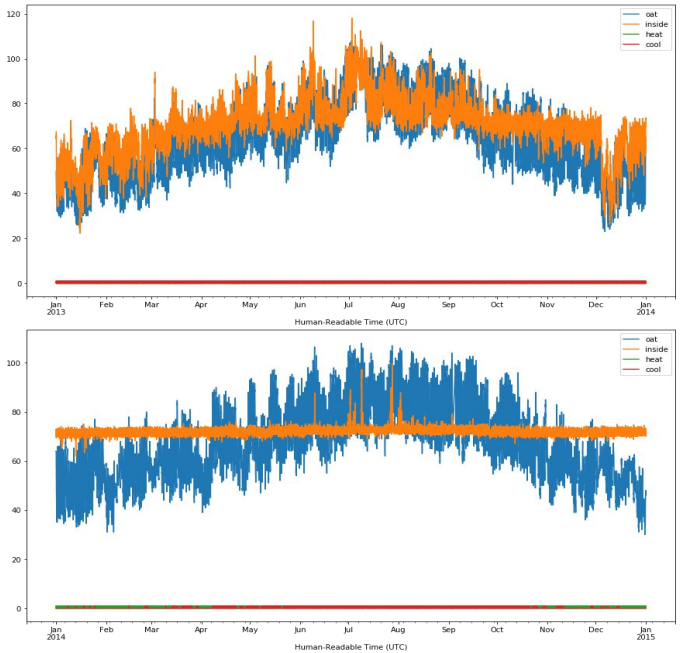
Example: Self-configuring Thermal Model

- Running optimal thermostatic control in ~20 buildings
  - Butcher shop, movie theatre, vets hall, senior center, office buildings, residential, etc
  - Different {equipment, sensors, architecture, climates}
- Requires data-driven thermal model:
  - ~~Hand-code building-specific thermal model~~
  - Code a thermal model *template* app
  - App uses Mortar to configure itself to a building
  - 20 different thermal models with the same code base



Looking Forward: Applying RL to Building Controls

- Reduce the role of the expert in designing control schemes for buildings
  - Expert doesn't scale
- Need robust simulations to apply RL
  - Work on thermal model enables creating "digital twins" at scale
- How to write/deploy RL agents (at scale) for different deployment context?



Outline

1. ~~Lecture on Brick Schema~~

2. ~~Overview of Mortar platform~~

3. Guided exploration of Mortar with online tutorial

Resources

Brick Website
https://brickschema.org/

Mortar Website
https://mortardata.org/

Tutorial Page
https://tutorial.mortardata.org/

Brick Explore Tool
https://querybuilder.mortardata.org/

Interactive Portion

1. Go to mortardata.org and click “Sign Up” to make an account

2. Go to tutorial.mortardata.org

3. Complete the first 2 cells