Bringing Smart Buildings to Life

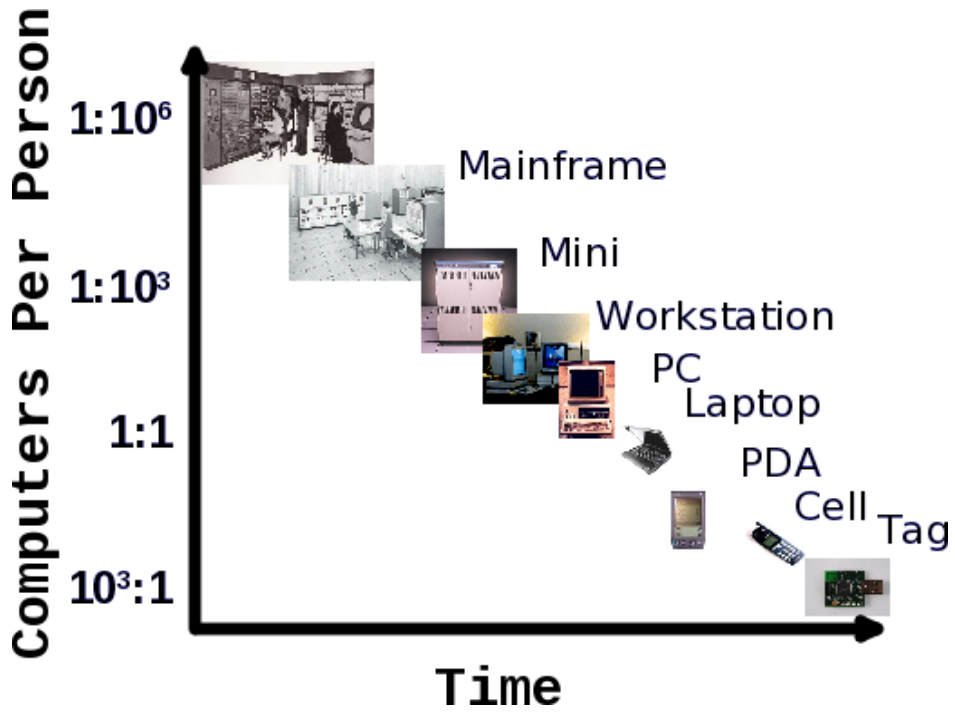
Rutgers University, WINLAB

RUTGERS WINLAB

The Smart Building Vision

The idea of smart buildings was born from the increasing availability of computers.

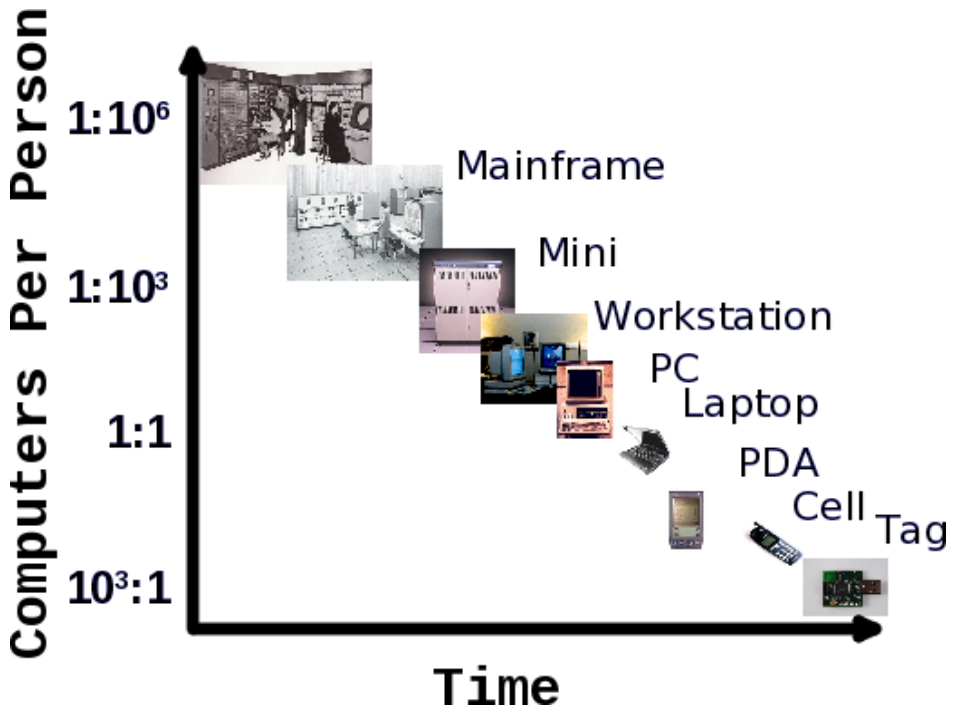
- A computer in every home
- ... in every room
- ... in every appliance
- ... in every thing



The Smart Building Vision

The idea of smart buildings was born from the increasing availability of computers.

- A computer in every home
- ... in every room
- ... in every appliance
- ... in every thing
- So where are the smart buildings?



Harder Than We Think

- More than a single app
  - People in buildings have many needs
  - Not just power monitoring
- More than a mash-up
  - Smart phone + Sensors $\neq$ Smart building
  - RFID + Database $\neq$ Smart building

Harder Than We Think

- More than a single app
  - People in buildings have many needs
  - Not just power monitoring
- More than a mash-up
  - Smart phone + Sensors $\neq$ Smart building
  - RFID + Database $\neq$ Smart building
- We need to plan ahead

Harder Than We Think

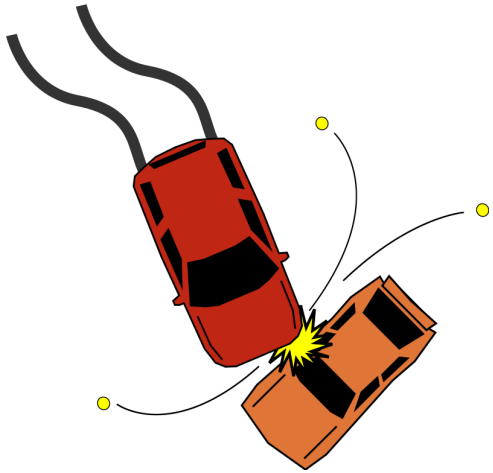
- More than a single app
  - People in buildings have many needs
  - Not just power monitoring
- More than a mash-up
  - Smart phone + Sensors $\neq$ Smart building
  - RFID + Database $\neq$ Smart building
- We need to plan ahead
  - build systems for users, not researchers

An Example of Success: Cars

- Some sensors are obvious:
  - Guages, warning lights, speedometer
- Cars also have sensors that we don't see
  - ABS
  - Noise cancellation within sound system
  - Warning sounds under specific conditions
    - Door open and the lights are on
    - Door open and keys are in the ignition

Buildings Are More Challenging than Cars

- Cars last 10s of years, buildings can last for 100s
- Cars are not retrofitted - they are scrapped and we buy a new one
- Cars are much more standardized than buildings
- Goal of sensors in cars is very clear (safety, comfort)



Smart Buildings Are Difficult

1. Deployments not standardized
   - Must use existing infrastructure
   - Must be robust in changing environments

Smart Buildings Are Difficult

1. Deployments not standardized
2. Use cases not clear
   - Can't make assumptions that constrain the system
     - For example, assuming everything is location-based
   - Best if skilled users can make their own applications (and share them!)

Smart Buildings Are Difficult

1. Deployments not standardized
2. Use cases not clear
3. Must be manageable for building owner or IT staff
   - Low maintenance, long-lived hardware
   - Reusable software components
   - Easy to add new sensing hardware

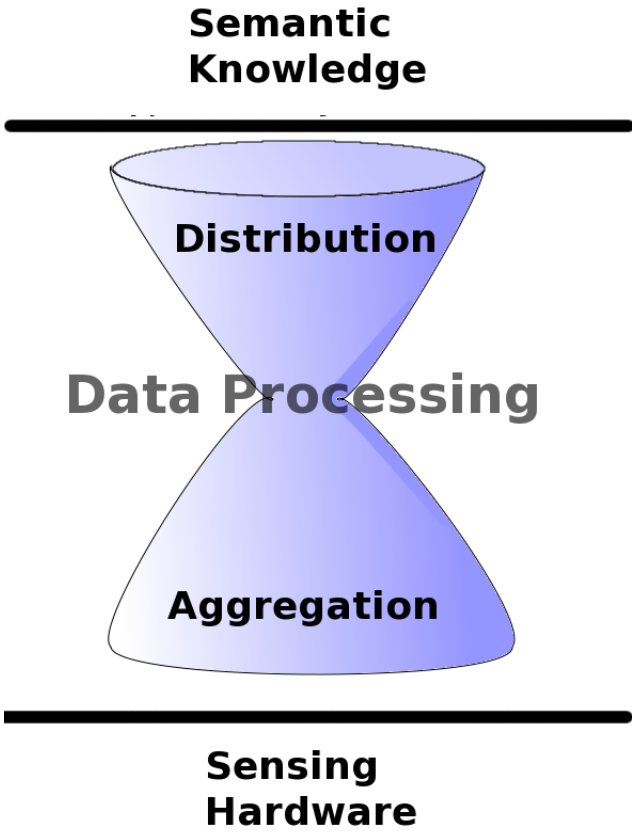
Smart Buildings Are Difficult

1. Deployments not standardized
2. Use cases not clear
3. Must be manageable for building owners or IT staff

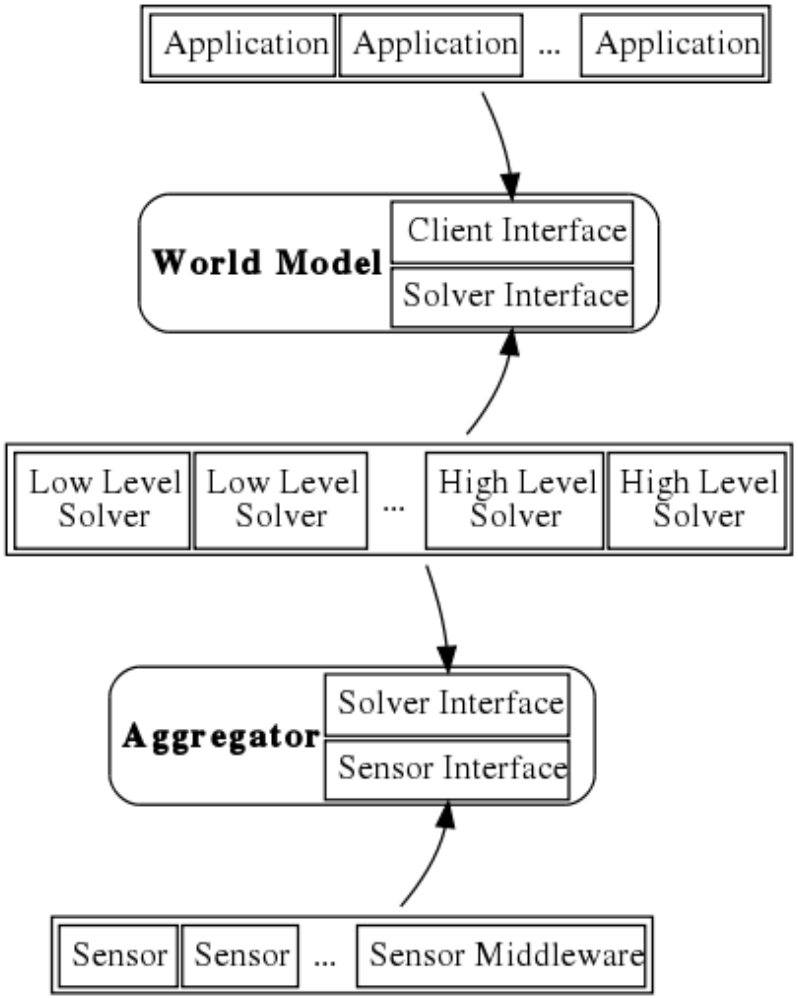
- These problems must be answered in software and hardware
  - Going to focus on software in this talk

Well Known Solutions: The Hourglass

- We can look to the hourglass design
  - Wide variety of sensors on the bottom
  - Wide variety of semantic information on the top
  - Modular data processing in the center

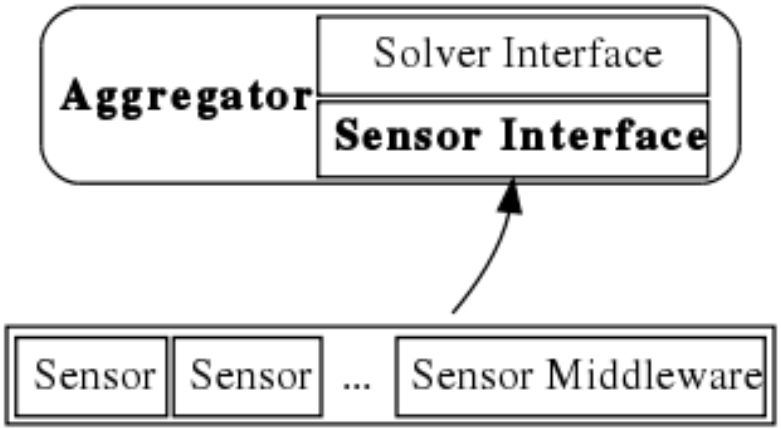


Our Solution - Octopus



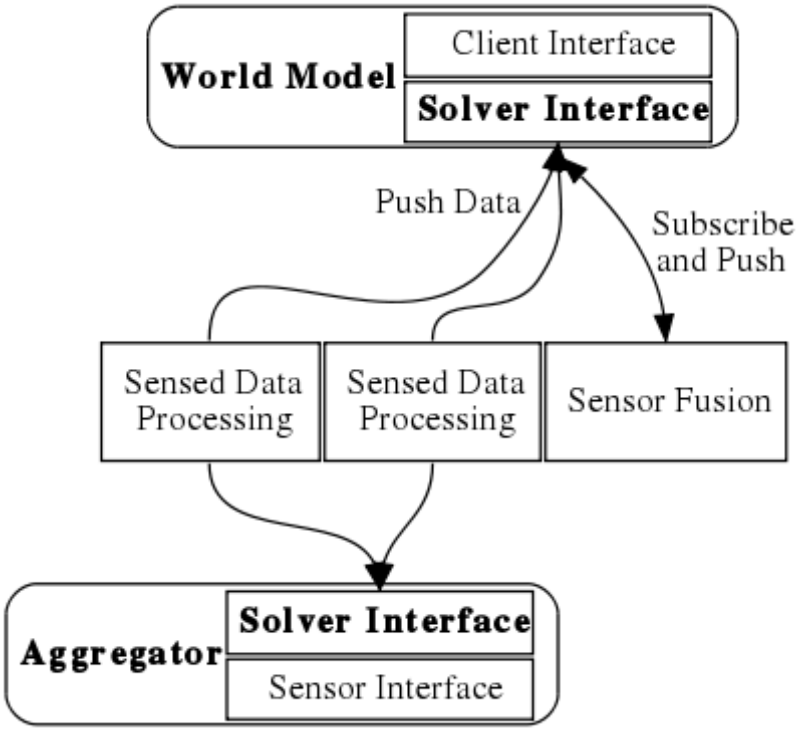
- Two abstraction layers
  - Aggregator
  - World Model
- Runs on TCP so tools in many languages can be used
- Divides concerns into three views
  - Sensor designer's view
  - Data analyst's view
  - Application developer's view

Sensor Designer's View



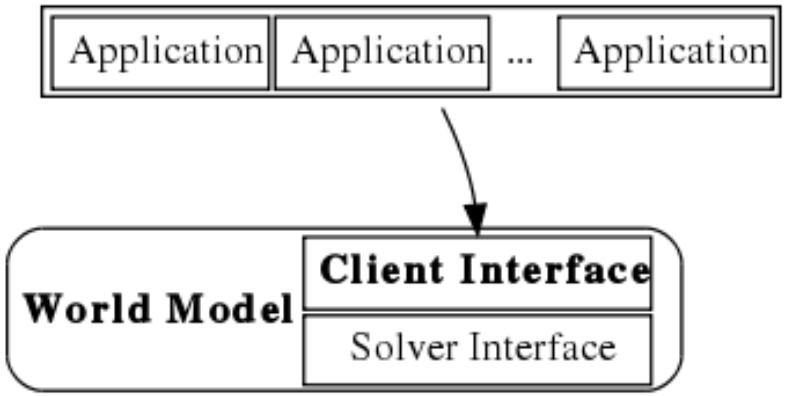
- Just needs to get formatted data into the system.
- Details of data storage and processing are hidden

Data Analyst's View



- Data processed by stand-alone processes called *solvers*
- Subscribe to raw sensor data from the aggregator
- Push higher-level information to the world model
- Can also search or subscribe to world model information
- Details of data creation and use are hidden

Application Developer's View



- Can subscribe to and search information
- Details of data analysis are hidden

Data In the World Model

- All data in the world model is associated with objects
- Objects are a name and a set of attributes
  - Names are stored as hierarchical URIs
  - Relationships can be implicit in names or explicit in attributes
    - "mug.Ben" is a mug
    - "WINLAB.room.D104" is a room in WINLAB
    - General naming scheme is "location.object type.name"

Example View

Enter URI filter pattern: .*

Fetch/Refresh

Enter a new URI:

Create URI

Update Attributes | Delete URI/Attribute

Object URI	Attribute Name	Origin	Data	Created	Expires
▼ winlab					
▶ anchor					
▼ chair					
▶ 28					
▶ 527					
▼ 534					
winlab.chair.534	location.uri	grail/discriminator solver	unknown	1320076669682	0
winlab.chair.534	empty	grail/switch_solver	0x01	1321054128429	0
winlab.chair.534	mobility	grail/mobility_solver	0x00	1321059585318	0

Interpreting Object Attributes

- Data types are determined by attribute names.
- If a solver creates data called “location.xoffset” a user (or any other solver) can rely on this being a double.
  - Similar to MIME, we must maintain a list of known types
- Octopus supports composite types, such as vectors.
  - Eases storing configuration data in the world model
    - Example: Histogram of signal values for location discrimination

Interpreting Object Attributes: Origins

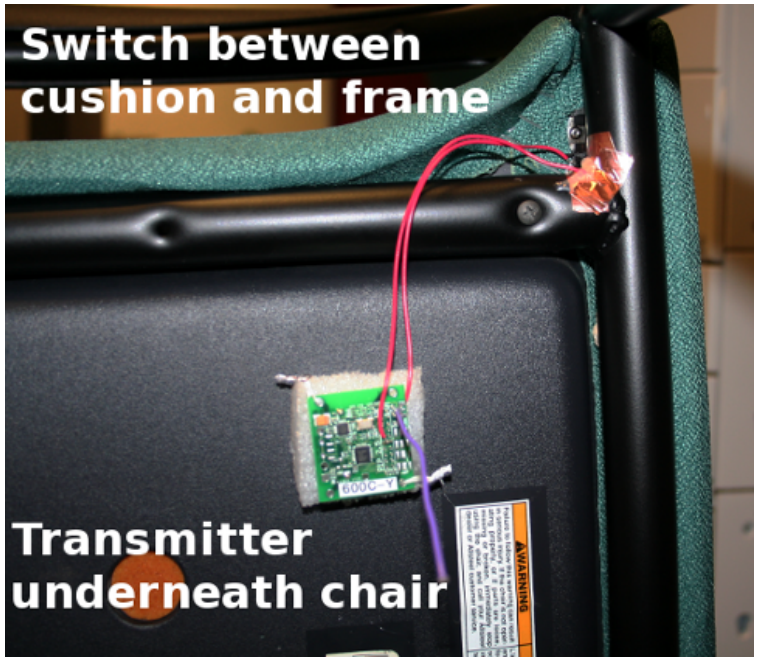
Each attribute has an origin that identifies its source. Solvers and client programs can specify an ordered preference for different origins - this solves two problems:

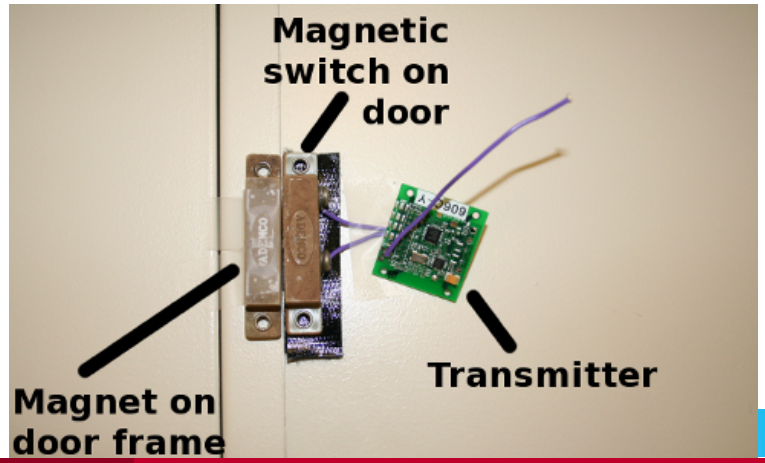
- Multiple solvers could provide the same data but some have better information
  - The most preferred data is returned when it is requested
- Some data processing algorithms can fail or software might crash
  - The world model can expire data
  - When data from one source expires the world model searches for the next most preferable source of that data

Using the System

As an example let's check if my advisor is in her office.

- Is her chair in use?
- Maybe she is in the office but not sitting - is anything moving?
- Has she been in today? Let's see if the door opened this morning.
- I can quickly deploy chair and door switch sensors





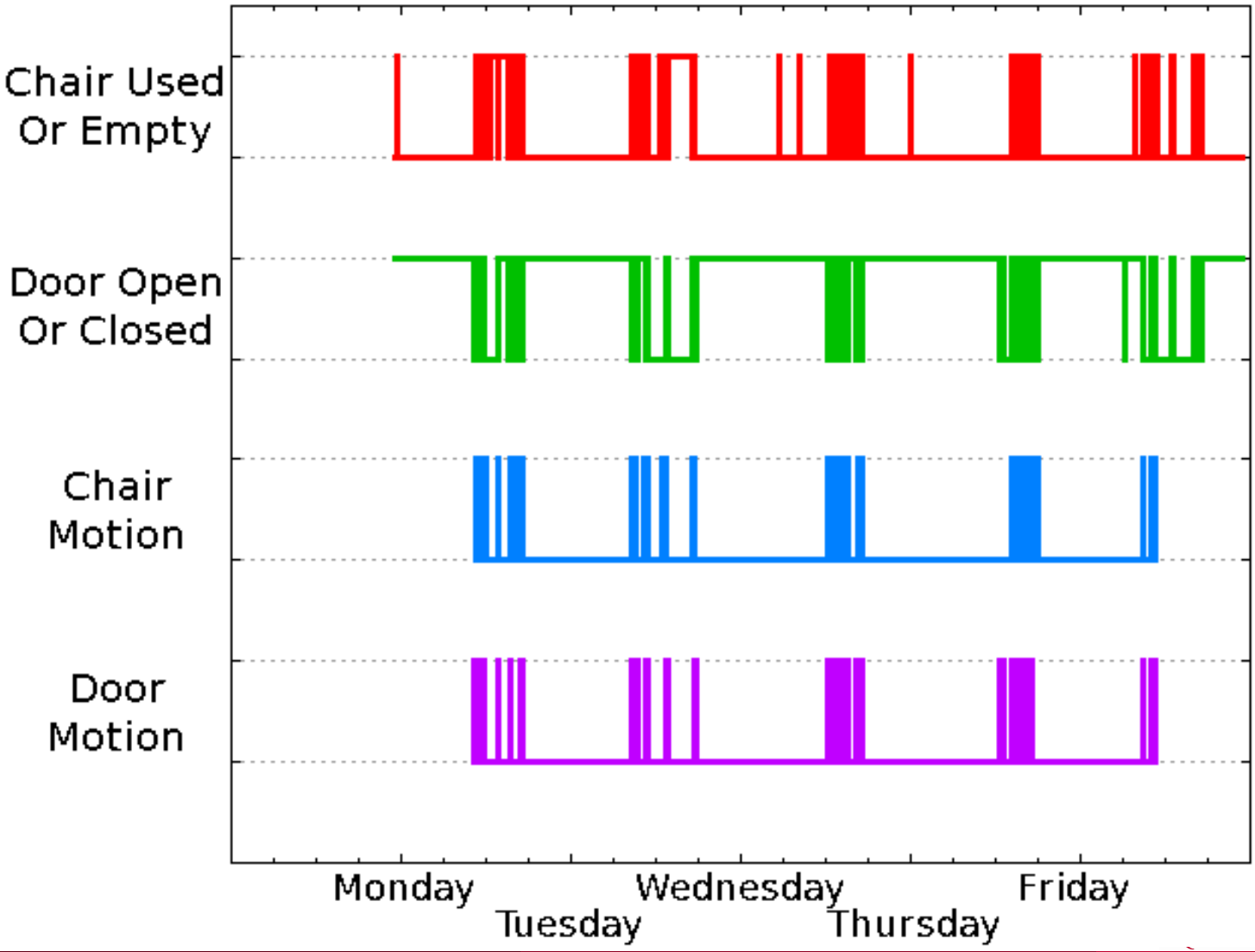
Raw Data

Raw data is sent to the aggregator - not very useful for most users.

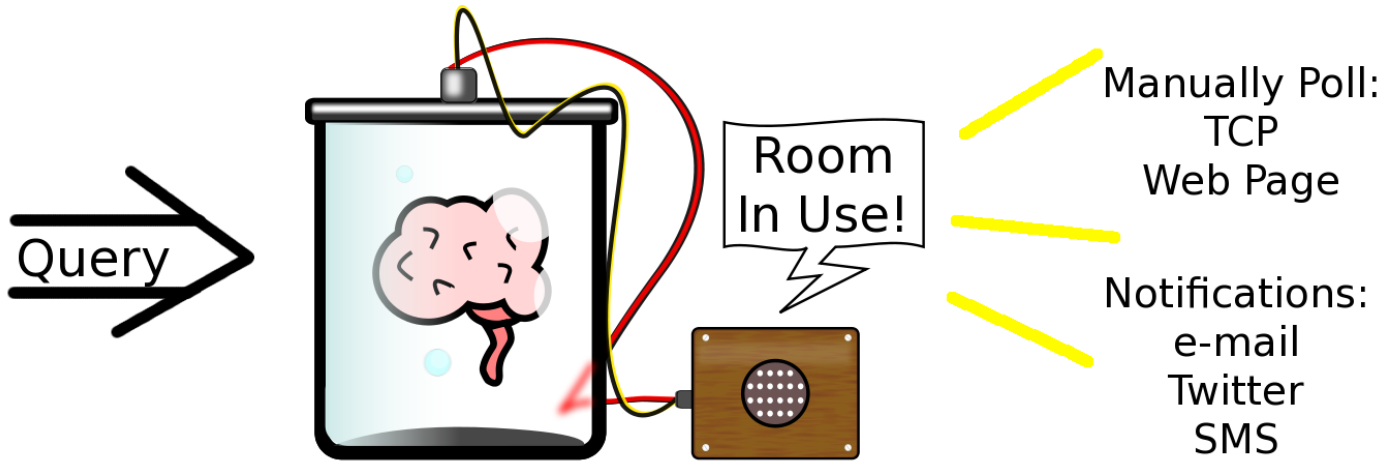
```
662     872743990       528     0       -54.5   0x00    Extra:0         Dropped:0
676     872743992       528     0       -73     0x00    Extra:0         Dropped:0
662     872745010       528     0       -54     0x00    Extra:0         Dropped:0
676     872745012       528     0       -72.5   0x00    Extra:0         Dropped:0
662     872746031       528     0       -54     0x00    Extra:0         Dropped:0
676     872746032       528     0       -72.5   0x00    Extra:0         Dropped:0
662     872747051       528     0       -54     0x00    Extra:0         Dropped:0
676     872747053       528     0       -73     0x00    Extra:0         Dropped:0
662     872748069       528     0       -54.5   0x00    Extra:0         Dropped:0
676     872748071       528     0       -73     0x00    Extra:0         Dropped:0
662     872749089       528     0       -54.5   0x00    Extra:0         Dropped:0
676     872749091       528     0       -72.5   0x00    Extra:0         Dropped:0
676     872750109       528     0       -72.5   0x00    Extra:0         Dropped:0
662     872750110       528     0       -54     0x00    Extra:0         Dropped:0
662     872751130       528     0       -54.5   0x00    Extra:0         Dropped:0
676     872751131       528     0       -72.5   0x00    Extra:0         Dropped:0
662     872752149       528     0       -54.5   0x00    Extra:0         Dropped:0
676     872752151       528     0       -72.5   0x00    Extra:0         Dropped:0
662     872753168       528     0       -54     0x00    Extra:0         Dropped:0
676     872753169       528     0       -72.5   0x00    Extra:0         Dropped:0
```

Processed Data

Solvers create chair use, door status, and mobility information.



The Client's View



- A solver combines lower level information into an "in use" attribute.
- A client's task is simple:
  1. Ask world model for status of URI winlab.room.A104 with attribute named "in use"
  2. Check if that is true or false.

Some Deployment Experiences

- Octopus deployed for about 1 year in WINLAB
- A few user interfaces
  - Live status map
  - Notifications via email, SMS, and Twitter
- Used by other researchers for demos data collection
  - Software reuse made system easy to use

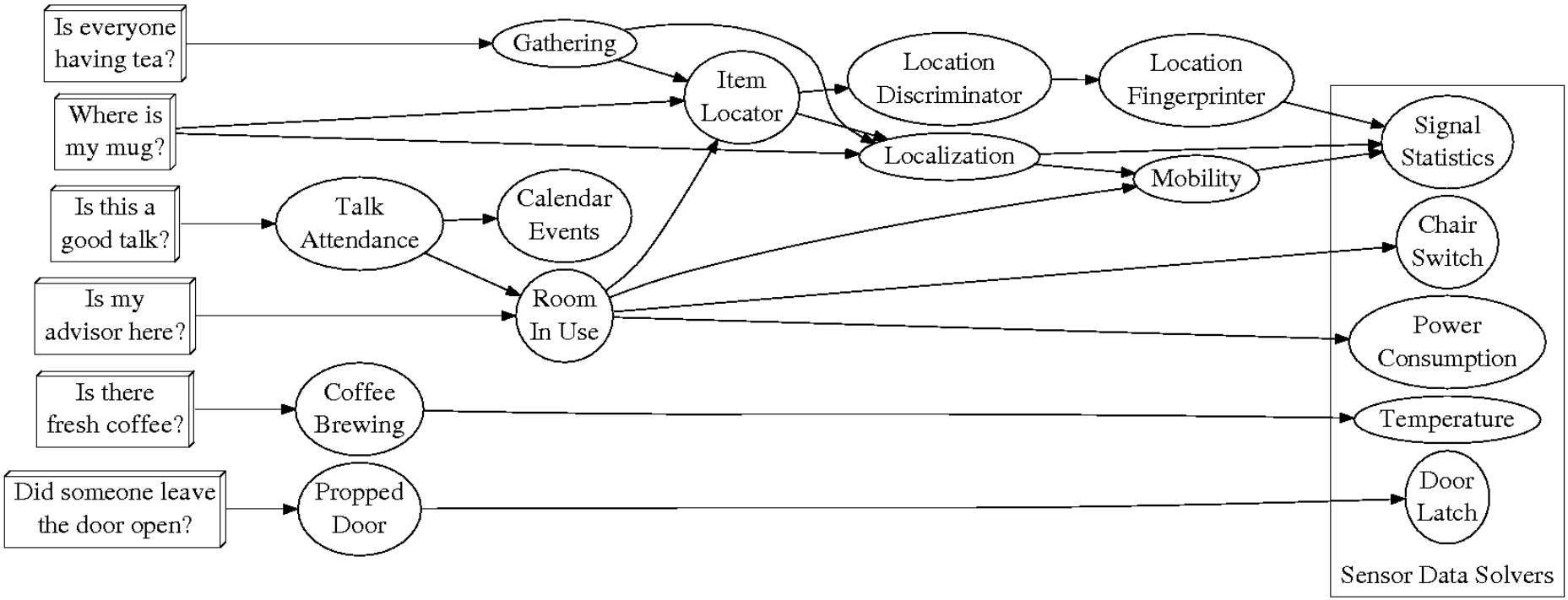
Example System Information

- Sensor data is turned into higher-level knowledge to answer user needs.

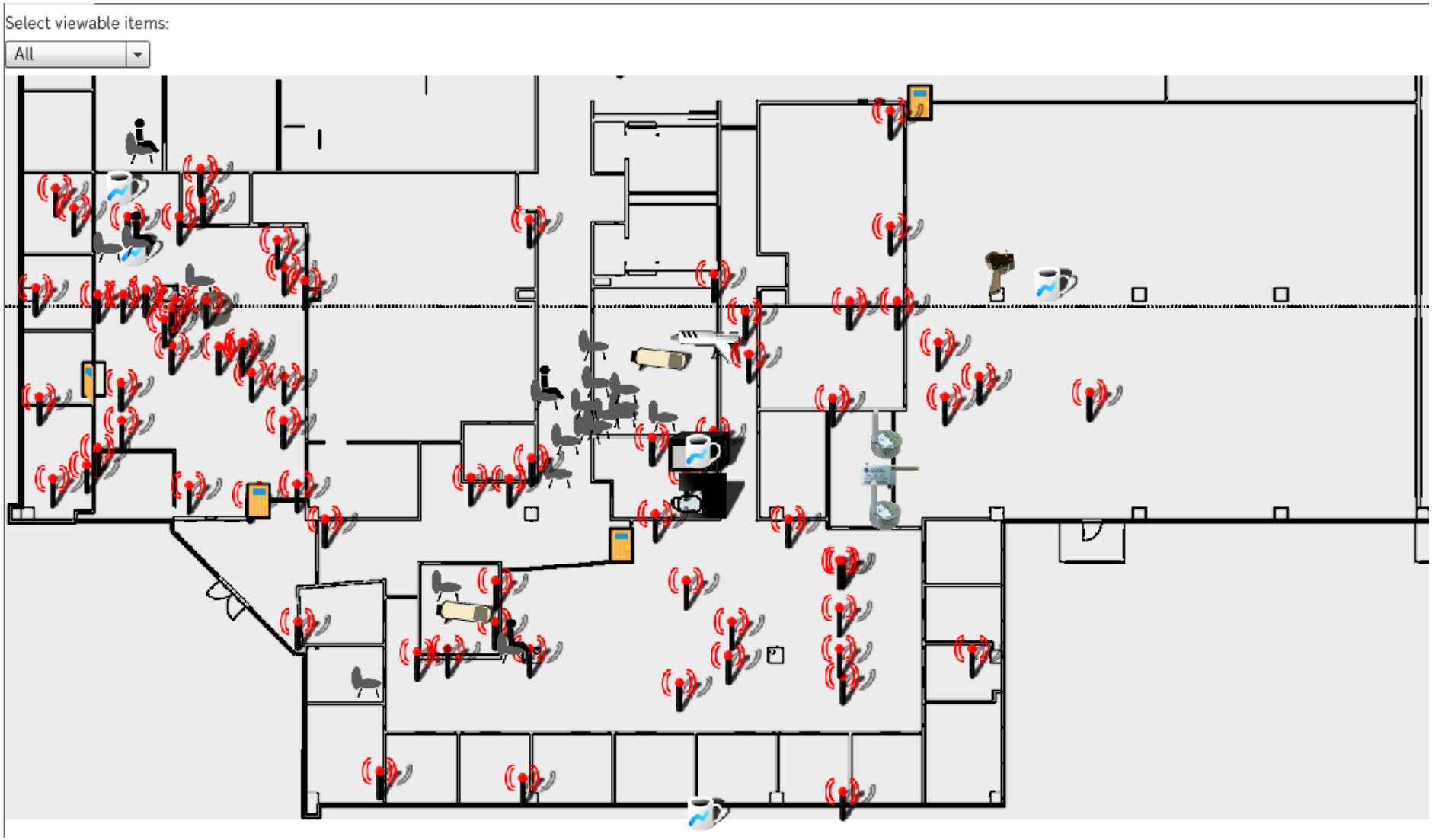
Localization	Where is my mug?
Propped Door	Did someone leave the back door propped open?
Coffee Brewing	Is there any fresh coffee?
Room In Use	Is the conference room in use?
Calendar Events	Are there upcoming talks?
Tea Time	Is everyone having tea?
Smart Printers	I want to print to the closest printer.

Solvers Depend Upon Each Other

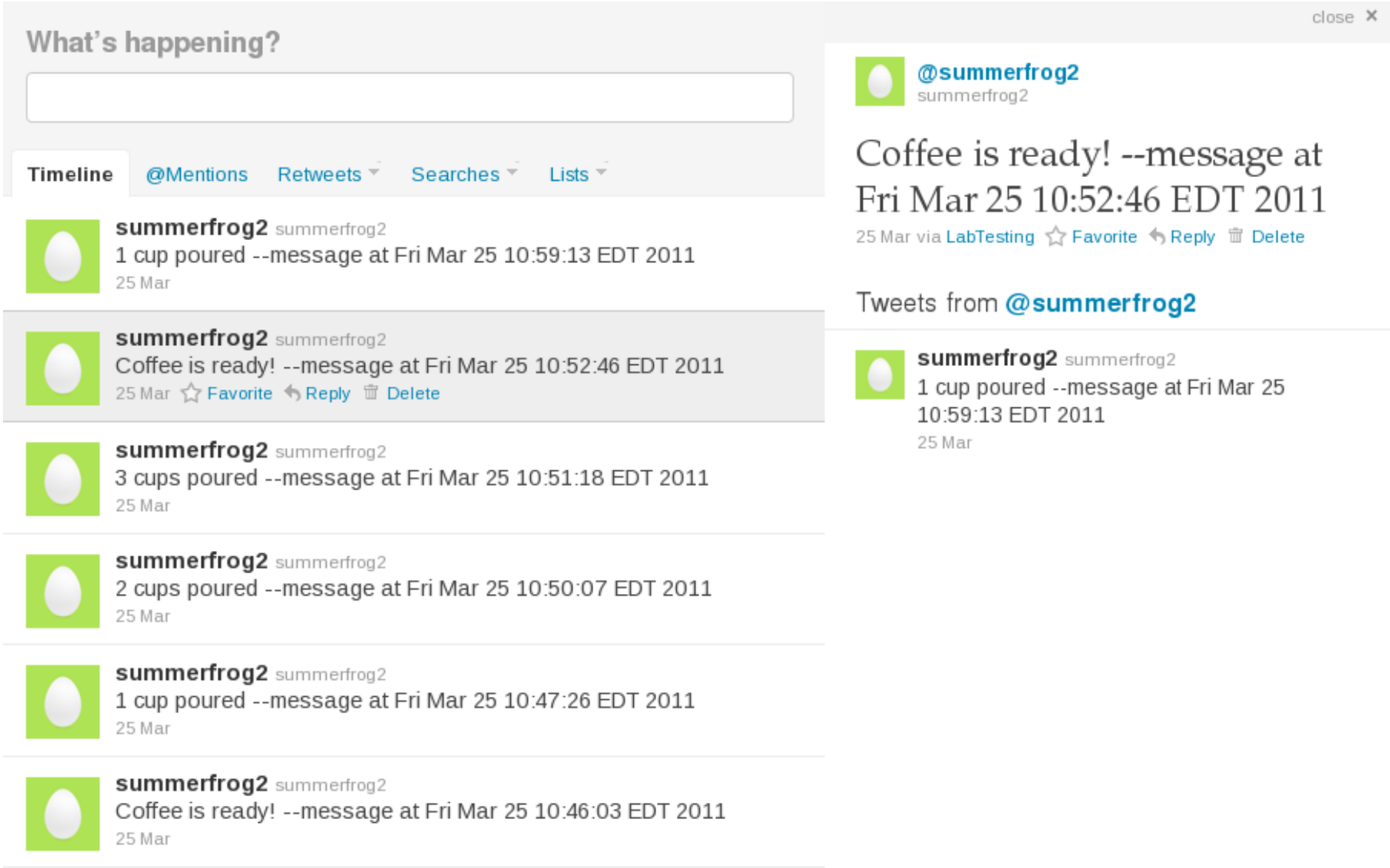
Information reuse makes small applications feasible



The Whole Deployment



Coffee Status for Caffeine Addicts



Conclusions

- Separating concerns makes it easy to focus on individual tasks
  - Create and deploy new sensors
  - Test new analysis modules
  - Quickly create new applications for users
- Future Directions
  - Get more people involved - find out what users want
  - Large scale deployments