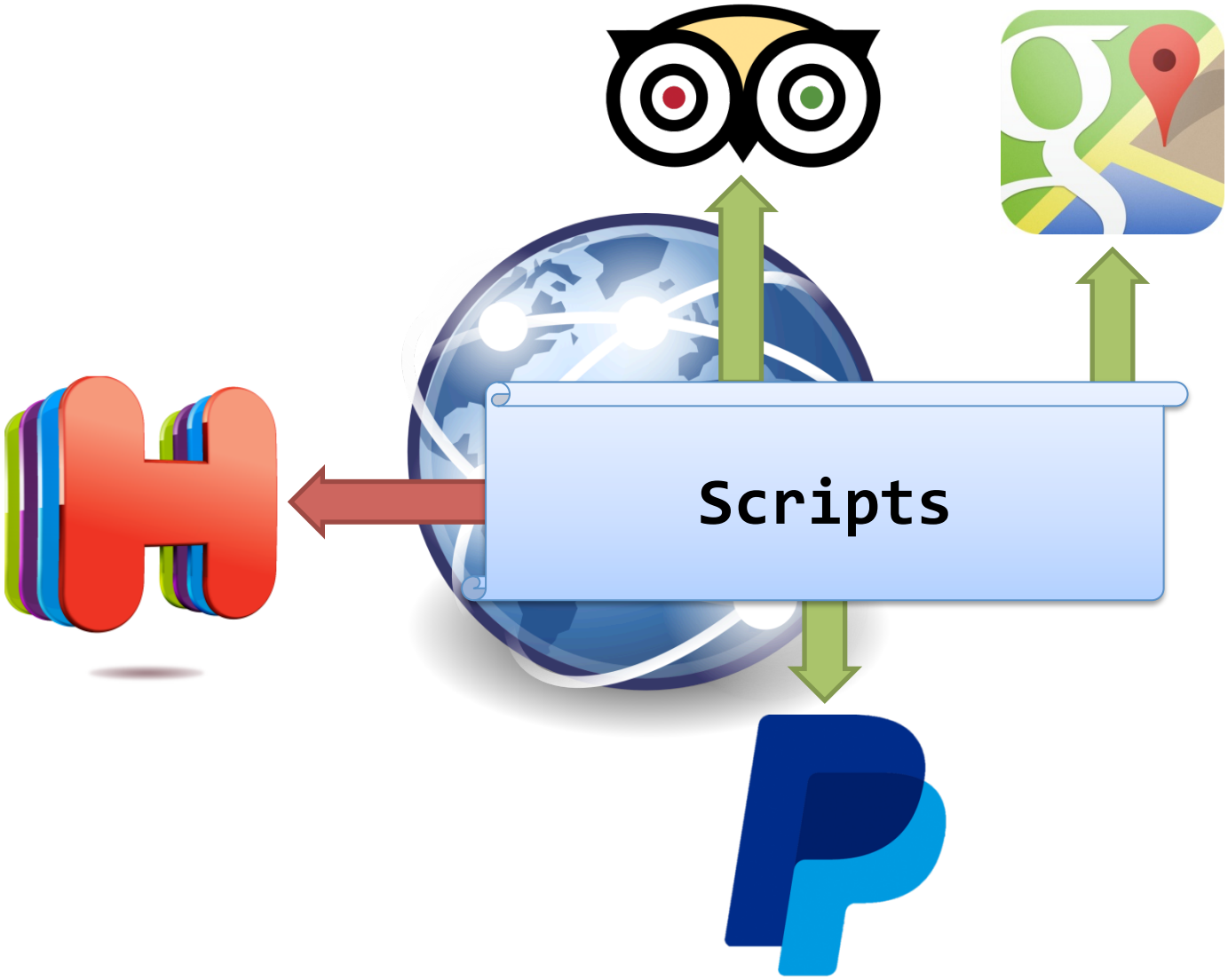
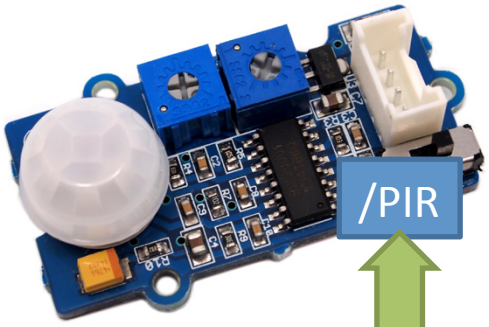
Web Mashups



IoT Mashups

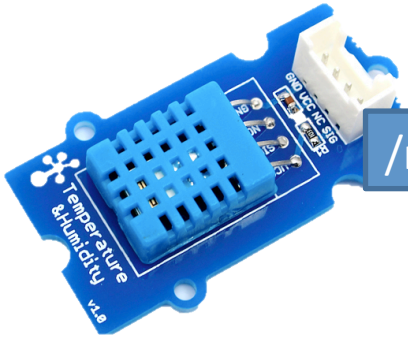


/PIR

The Weather Channel

/set/valve

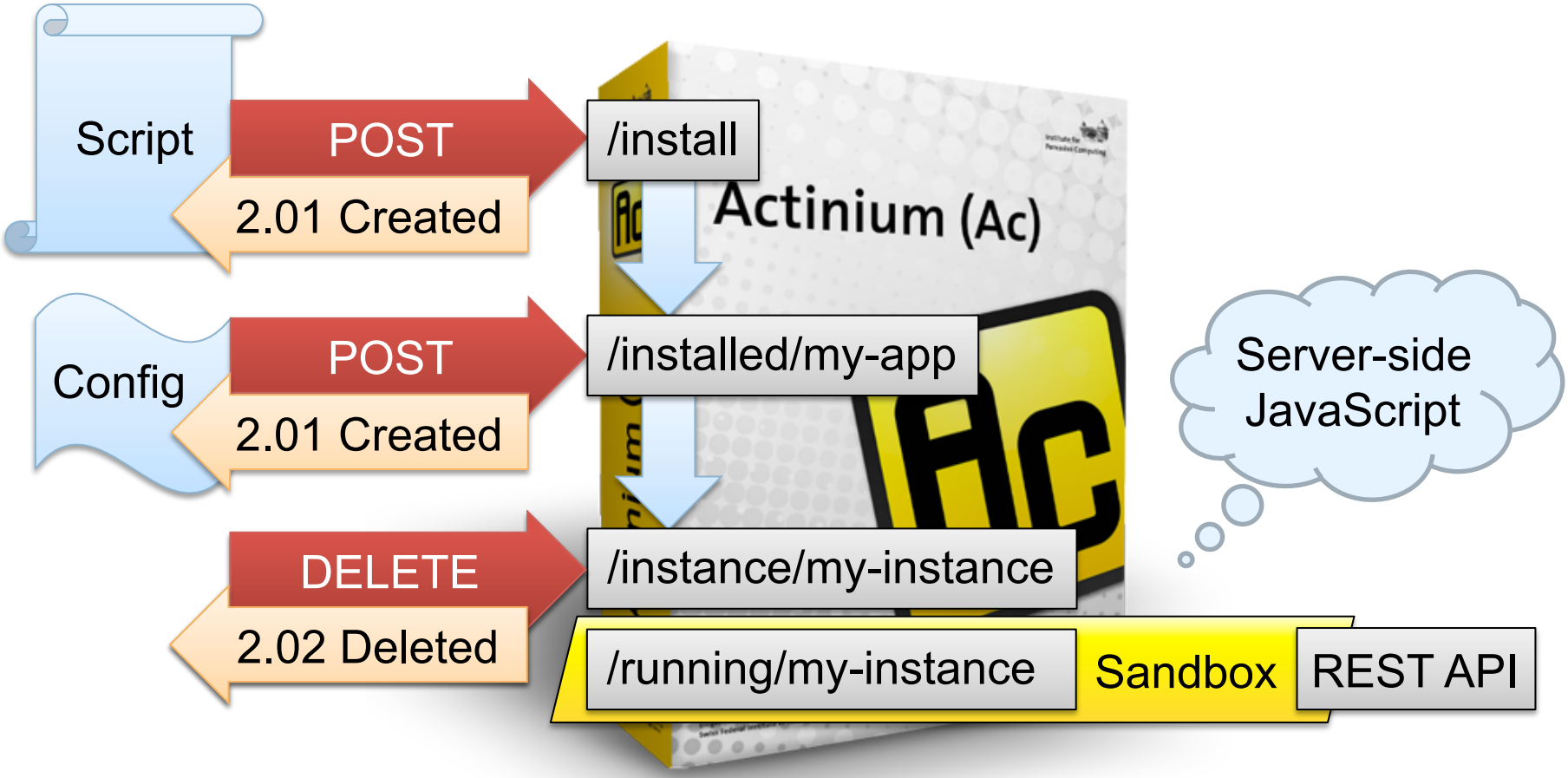
```
if ( occupied ) {
   heatRoom( COMFORT_TEMP );
}
```



/raw/ADC2

/sensors/humidity

Actinium RESTful Runtime Container



Actinium Apps Are REST Resources

```
// handler for GET requests to "/"
app.root.onget = function(request) {
// returns CoAP's "2.05 Content" with payload
        request.respond(2.05, "Hello world");
    };

// sub-resource "/config"
var sub = new AppResource("config");

sub.onput = function(request) {
        variable = request.payloadText;
        request.respond(2.04); // "2.04 Changed"
    };
app.root.add(sub);
```

Mashups with Devices and other Apps

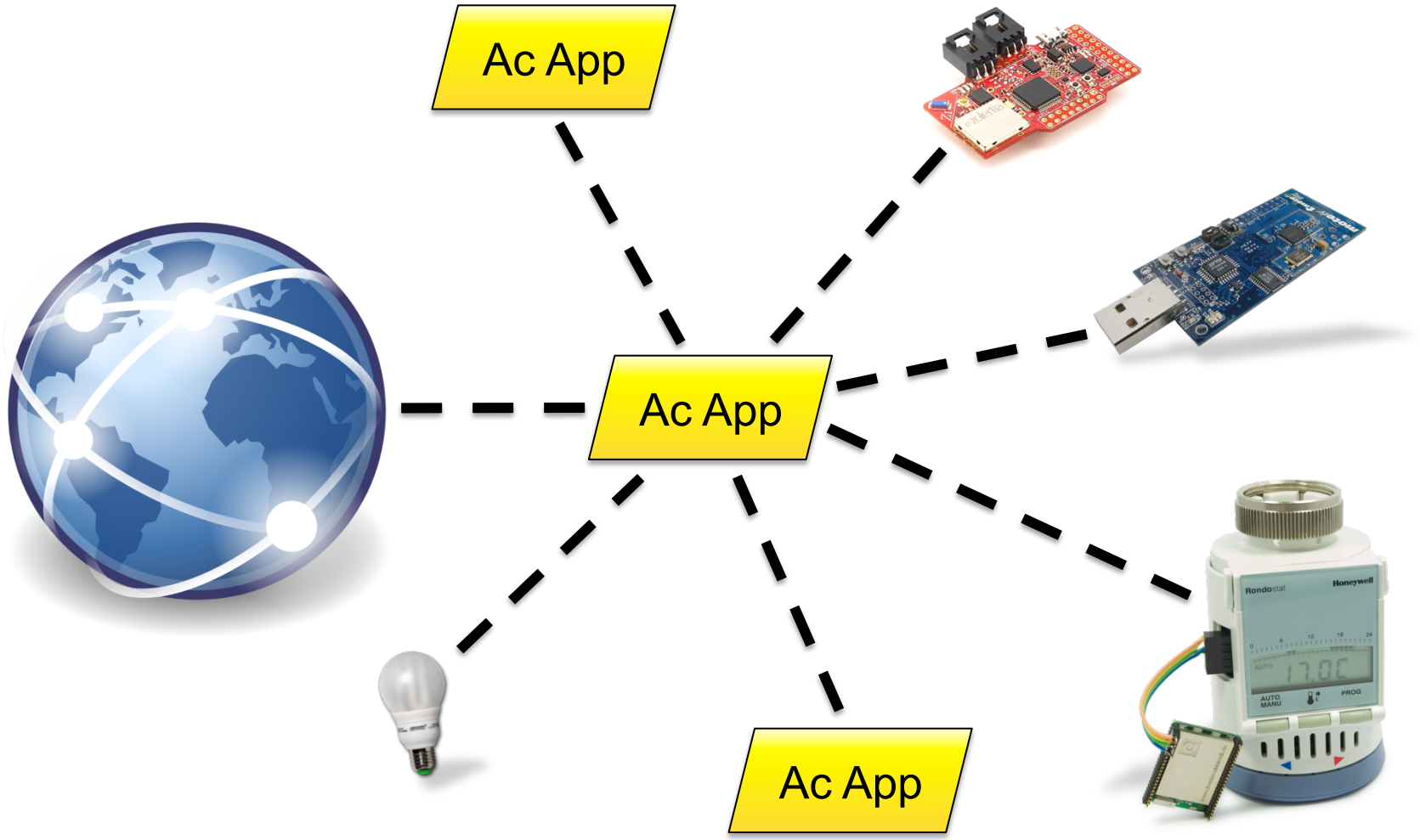
```
var req = new CoapRequest();
// Request the temperature sensor reading via CoAP
req.open("GET", "coap://mote1.example.com/sensors/temp",
                        false /*synchronous*/);

// Set Accept header to application/json
req.setRequestHeader("Accept", "application/json");

req.send(); // blocking

// Use IoT data just like Web data
var roomTemperature = JSON.parse(req.responseText);
```

Mashups



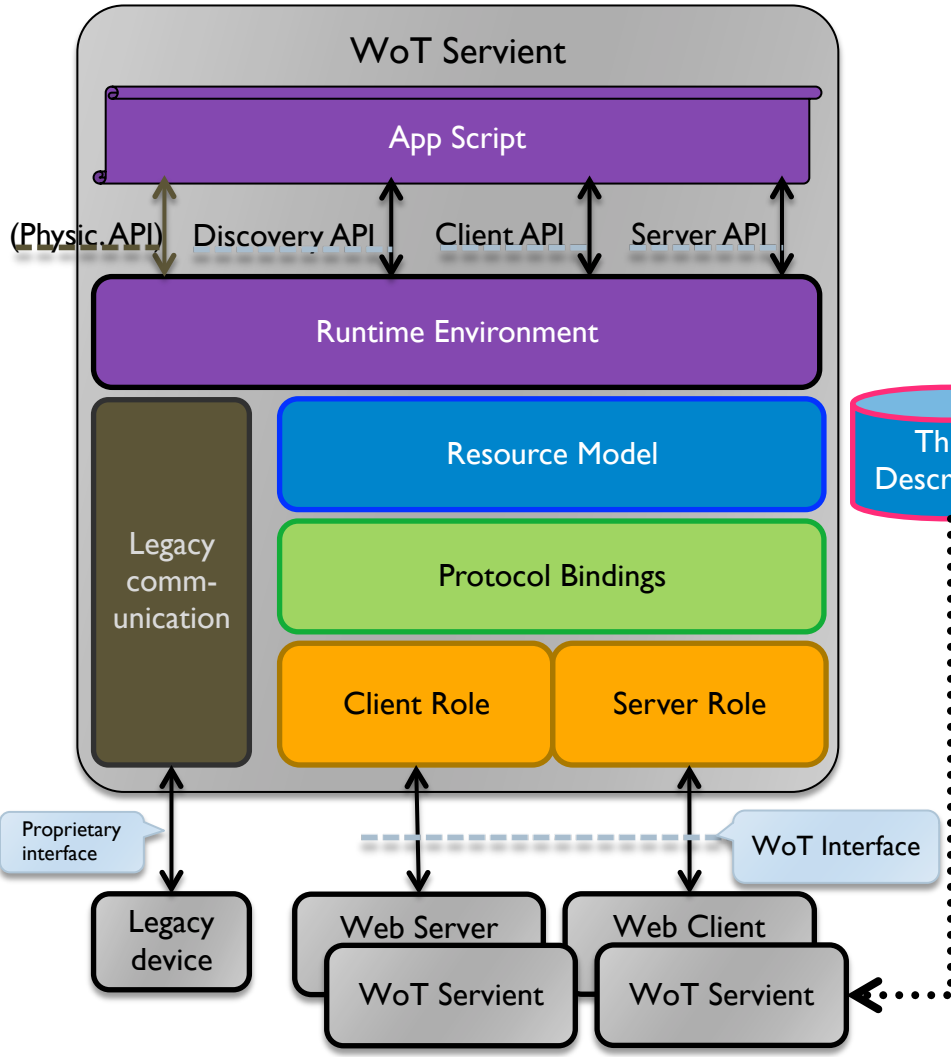
Script Example (Expose Thing)

```
// create software object to represent local Thing
WoT.newThing("counter")
    .then(function(thing) {
        thing
            // programmatically add interactions
            .addProperty("count", {"type": "integer"})
            .addAction("increment")
            .onInvokeAction("increment", function() {
                console.log("incrementing counter");
                // persistent state is managed by runtime environment
                var value = thing.getProperty("count") + 1;
                thing.setProperty("count", value);
                return value;
            })
        // initialize state (no builder pattern anymore)
        thing.setProperty("count", 0);
    })
    ._catch(console.err);
```

Script Example (Consume Thing)

```
// create software object to represent remote Thing based on TD URI
WoT.consumeDescriptionUri("http://servient.example.com/things/counter")
    // use promise to handle asynchronous creation
    .then(function(counter) {
        counter
            // invoke an Action without arguments
            .invokeAction("increment", {})
                // which is an asynchronous call -> promise
                .then(function() {
                    console.log("incremented");
                    counter
                        // read Property (async.) to confirm increment
                        .getProperty("count").then(function(count) {
                            console.log("new count state is " + count);
                        });
                })._catch(console.error);
    })
    ._catch(console.error);
```

Thing Implementation: **WoT Servient**



Application Logic:

Can consume remote Things through the Client API, local hardware and connected legacy devices through a Physical API (t.b.d.), and expose Things through the Server API. To allow portable app scripts, the Servient must povide a runtime environment.

Resource Model:

Provides a common abstraction with uniform interface across the different protocols. Like the Web, it allows to identify and address interaction points through URIs.

Thing Description (TD):

Declares WoT Interface for interaction and provides (semantic) metadata for the Thing. TD is used by WoT clients to instantiate local software object of the Thing.

Protocol Binding:

Converts abstract interactions with Things to different protocols using the information from TD.