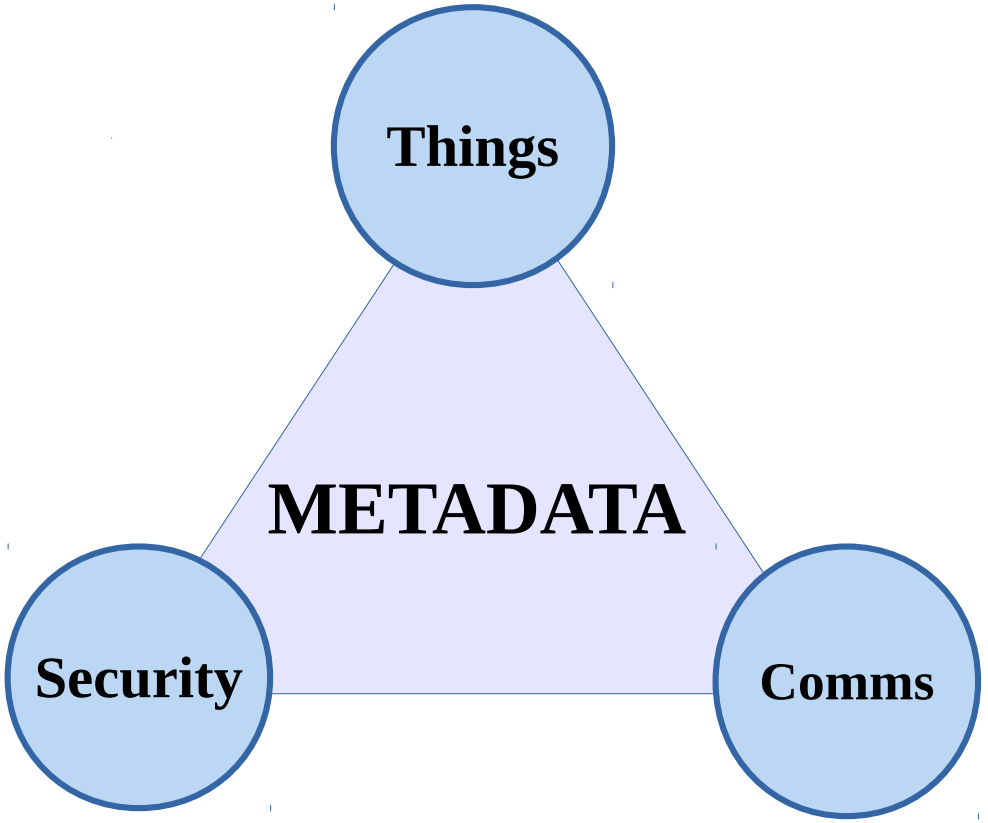
What is the Web of Things?

Things

- Things stand for physical or abstract entities
- Things as software objects in the application programs execution environment
- There are many potential protocols which will depend upon the context
- Likewise, there are multiple communication patterns, e.g. push, pull, pub-sub, and the choice will depend on the context
- By decoupling applications from the underlying protocols and messaging, we can simplify application development
- If an application wants to operate on a thing for a remote sensor/actuator, it needs a software object that acts as a proxy for the remote thing
- This can be created by through a server API that uses metadata for the thing and the remote server to create the software object

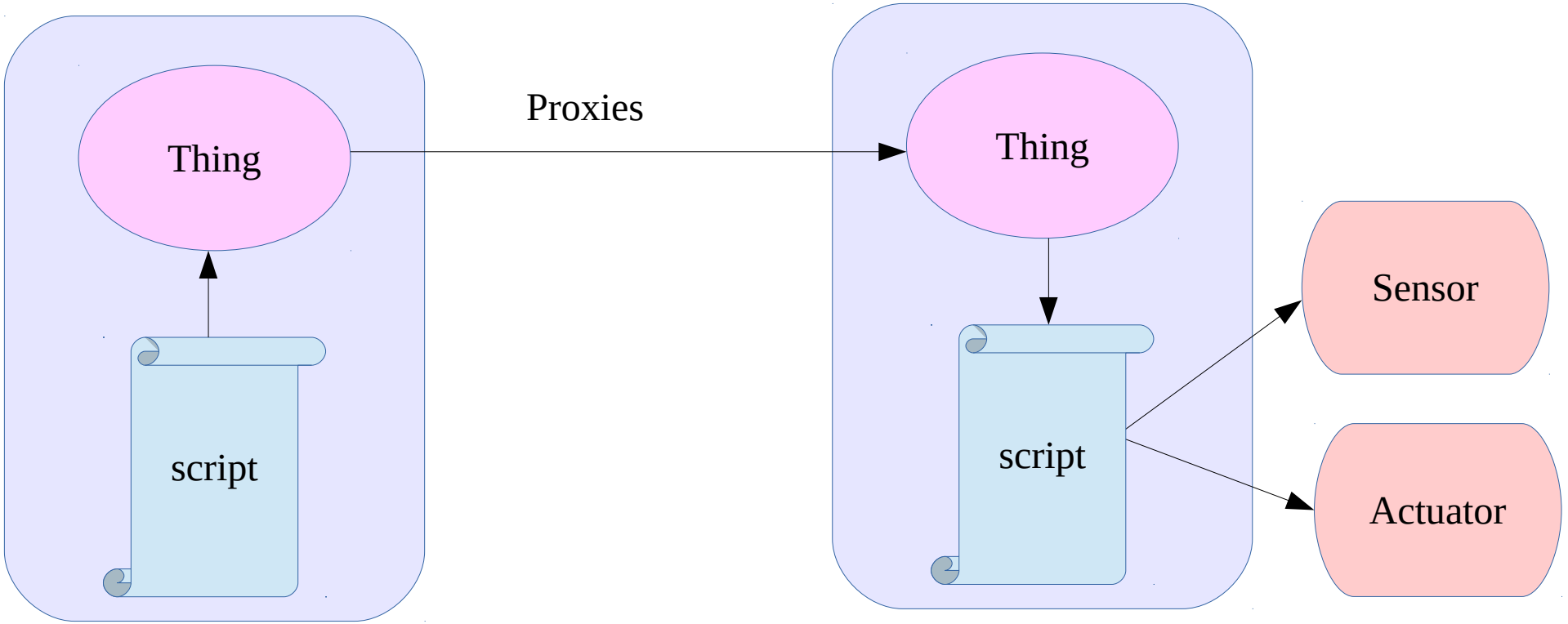
Core Metadata



- **Thing descriptions**
  - Links to thing semantics
  - Data models & relationships between things
  - Dependencies and version management
  - Discovery and provisioning
  - Bindings to APIs and protocols
- **Security related metadata**
  - Security practices
  - Mutual authentication
  - Access control
  - Terms & conditions
    - Relationship to “Liability”
  - Payments
  - Trust and Identity Verification
  - Privacy and Provenance
  - Resilience
- **Communication related metadata**
  - Protocols and ports
  - Data formats & encodings
  - Multiplexing and buffering of data
  - Efficient use of protocols
  - Devices which sleep most of the time

Distributed Web of Things

- Virtual representations of physical or abstract entities for use by application scripts
  - Each thing has a URI for a description that can be used to create a proxy for that thing, allowing scripts to interact with a local proxy on behalf of a remote entity
  - Scripting things in terms of their metadata, events, properties and actions
  - Web page scripts can create proxies for things on servers



A Question of Relativity

- The Web of Things is essentially a system of agents as defined by Carl Hewitt in 1973
- A set of things hosted on different servers
- Application scripts define the agent behaviour
- These servers exchange messages over a variety of protocols
- Messages take a variable amount of time to transfer that depends on the protocol, network, and devices
- Application scripts only have access to the state of the instances of the things on that server
- Even if a server asks another server for the state for a thing, by the time it gets a result, it may be out of date
- Even time is uncertain due to the limitations of synchronising clocks across servers

Some Requirements

- Create a thing from its metadata and an implementation
- Destroy a thing and all of its proxies
- Register a proxy for a thing
- Unregister a proxy for a thing
- Access to metadata for things and servers
- Notify events and updates to properties and metadata
- Invoke actions and asynchronously return their results
- These can be considered as abstract messages and mapped to communication patterns over specific protocols
- Allow for cyclic dependencies between things

Dependencies across Things

- One thing may depend upon another
  - Agent example which depends on door and light
- The dependent things may be on different servers
- When you're setting up a thing, the things it depends upon may not be available right now even it is on the same server
  - This requires a means to wait for them to become ready
- Cyclic dependencies
  - A depends upon B which depends upon C which depends upon A
- Server hold messages for things until they are ready
  - Avoids the need for messages that signal when things are ready
- I've got this working on my NodeJS server

Communication Patterns

- The properties for a given thing can be updated by the application script on the server hosting the thing, and by applications scripts on servers hosting proxies for that thing
- The proxies form a tree rooted in a thing
- Updates can be pushed from the thing to its proxies
- Updates can be pushed from a proxy to the thing, and from there to the other proxies
- Push can be related to pub-sub and message routing
- Another approach is to pull updates via polling

Simple Message Exchange

- Reliable in-sequence delivery of messages
- Asynchronous messaging across a communications channel
- Extensible message format, e.g. JSON or XML
- Efficient encodings for messages
- Can be layered on top of transport protocols
  - Web Sockets over TCP
  - Reliable messaging layer over UDP
  - Non-IP based communications technologies

Protocol Bindings

- REST family of protocols, e.g. HTTP & CoAP
  - See Ari's document on recommended practices
- Pub-Sub protocols, e.g. MQTT & XMPP
- Non-IP based technologies
  - e.g. ZigBee, Bluetooth, KNX, EnOcean and communication technologies with very small messages

API Patterns

- Different languages and conventions
  - Static vs dynamically typed languages
    - Dynamic languages allow objects to be given properties at run-time
    - Static languages force thing properties to be passed as method arguments
  - Call backs, e.g. for handling events, or results from actions
  - JSON & Objects as arguments
  - Promises as a popular pattern for JavaScript
- Locally generated events to allow applications scripts to listen for changes
  - Changes to properties
  - Changes to metadata
  - Lifetime events, e.g. when a thing is destroyed
  - Distinct from the events declared in the thing's model

Example

```
wot.thing("door12", {
        "events": {
            "bell": null,
            "key": {
                "valid": "boolean"
        }
    },
    "properties": {
        "is_open": "boolean"
    },
    "actions": {
        "unlock": null
    }
}, {
    start: function(thing) {
         thing.is_open = false;
    },
    stop: function(thing) {},
    unlock: function(thing) {
         logger.info("  unlocking" + thing._name);
    }
});
```