Internet of Things Workshop
ST – 2015/2016

Architecture

John Carpenter, 1982

Johan Lukkien

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking

Architectural styles (patterns)

- Remember: An architecture is *"the fundamental organization of a system embodied by*
  - *its components [jl: building blocks],*
  - *their relationships to each other [jl: connectors and interfaces, dependencies] and to the environment*
  - *and the principles guiding its design [jl: rules & constraints for building blocks and connectors] and evolution"*

- An <u>architectural style</u> is a *coherent set of design decisions* concerning the architecture
  - a combination of a typical (de)composition
  - and typical choices for connectors, components (building blocks) and behavior

  ....a *generic* solution for a *class of problems*

- We also have <u>interaction styles</u>, concerning just the *interaction* between building blocks
  - the nature of the connectors, and their organization

Architectural style characterization

- Defined by
  - *motivation (guidelines, context)* for the application of the style
    - which extra-functional properties are achieved, and how
    - which problem class is solved
  - *vocabulary*
    - names for components(building blocks) and connectors, and for other concepts
  - *rules (constraints, responsibilities)* for components and connectors
  - generic *structure and behavior*
    - interfaces of components, and correspondent connectors
    - data distribution, protocols, control flow, data flow

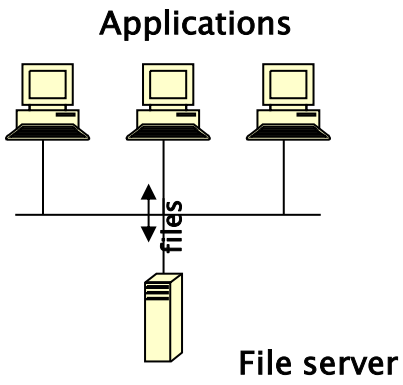
  ....documented as a *profile (e.g. UML)*

- When applied, a style yields a partial architecture
  - in fact: styles classify architectures
- Within an architecture, several styles can be applied
  - and also, several alternatives in interaction style

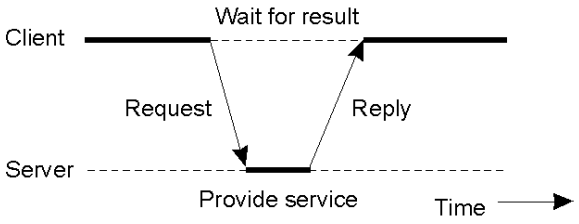
- Styles encourage *communication, reuse, comparison (of alternatives)*

Client-Server style

- Motivation:
  - sharing some localised resource (e.g. file store, compute server)
  - protecting and managing content (e.g. a database)
  - delay binding, decrease dependencies (independent development)
  - generally: separation of concerns
- Vocabulary
  - client, server (building blocks)
  - request, reply (interaction)
  - server discovery
- Rules
  - Server:
    - provides a service according to the above motivations
    - passive, awaiting requests from clients, does not know clients
    - handles data access, data integrity
  - Client
    - active, initiates activity, *discovers* server
  - no connection among clients
  - no, or limited, state on the server per client
- Structure

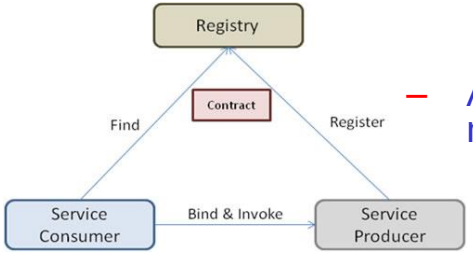


- Typical behavior
  - client finds server access point through discovery
  - regular interaction:

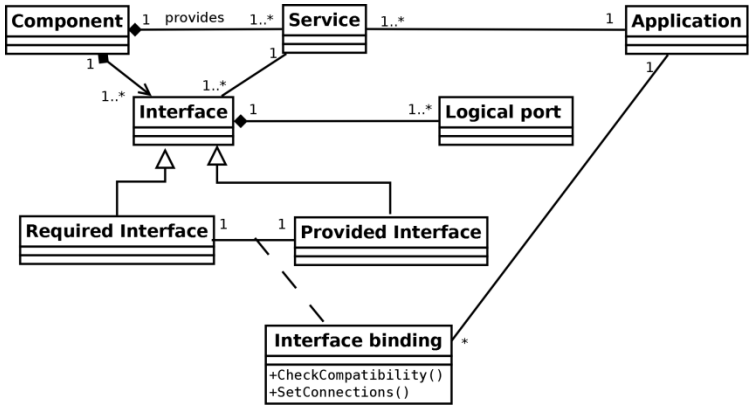


Service oriented (SOA) style

- Motivation:
  - Separate *functionality* ("service"), the *implementation*, the *deployment context* and the *application context*
  - Build applications by very late (dynamic) binding
  - Integration of enterprise information systems
- Vocabulary
  - SOA, service, interface, discovery, composition, binding, orchestration, choreography
- Rules
  - Application is built ad-hoc out of services that communicate in a standardized manner
    - via a network
    - see e.g. REST
  - Service is a self-contained functionality. It does not depend on state of other services, or of the system (OS, language) it is running on.
  - Services are discoverable

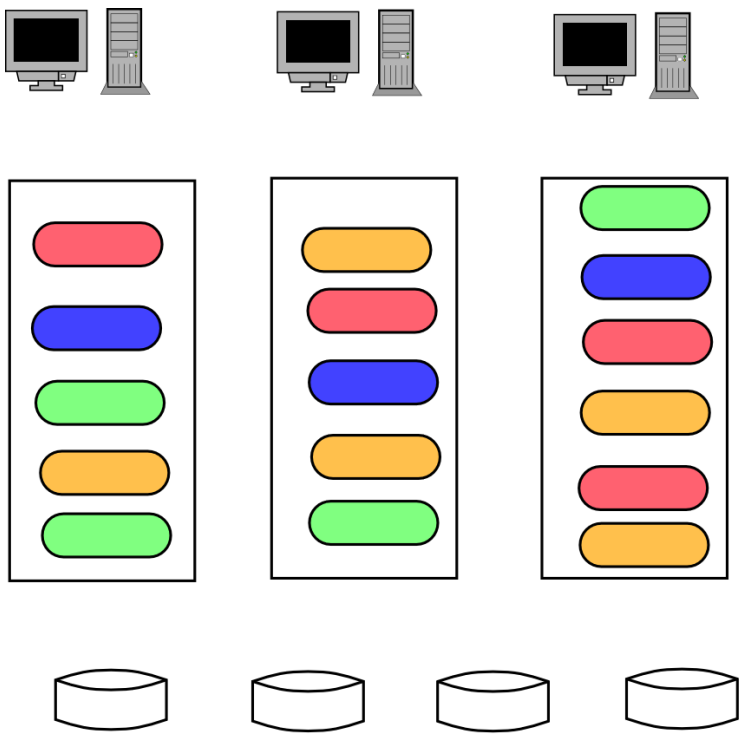


- Structure (conceptual):

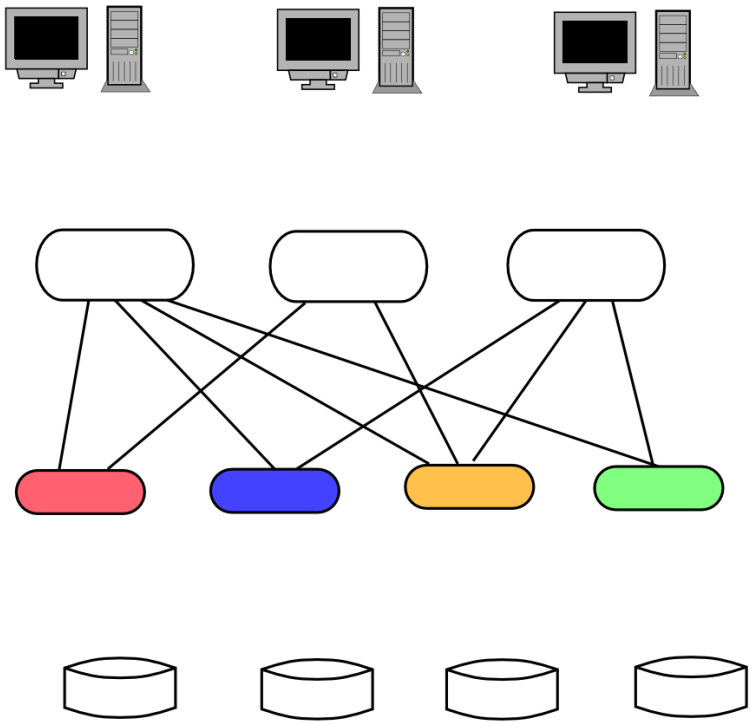


- Typical behavior
  - Providers publish services; Applications ("orchestrations") discover services, and bind their interfaces
    - Service broker (registry) can exist to manage discovery process
  - Applications send data objects through a number of services as a work flow
    - Often XML based RPC (SOAP)
    - Services are typically kept simple and focused on single task
- Note: SOA is a huge research domain in business information systems

Enterprise application organization



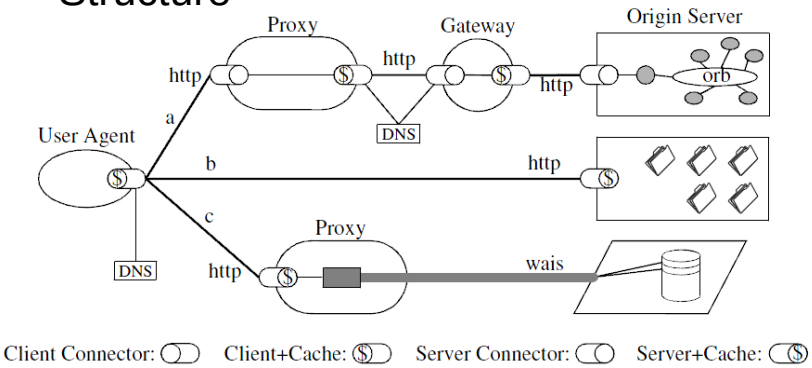
Monolitic applications



Tiered service architecture

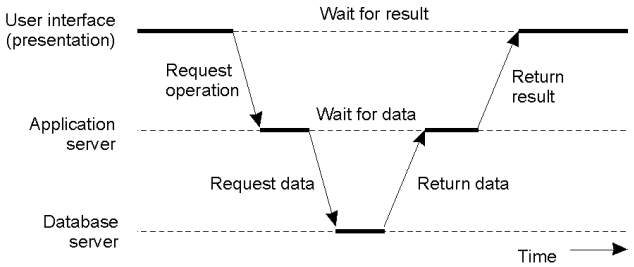
REST: REpresentational State Transfer

- Motivation:
  - special usage of C&S aiming at
    - portability
    - independent development & deployment
    - reduction of interaction complexity
    - reliability
    - scalability
- Vocabulary
  - user agent, origin server, gateway, proxy
  - cache, layer
  - state
  - code-on-demand
  - resource, resource identifier (e.g. URL), representation, metadata
- Rules
  - stateless communication (no server state)
  - response is labeled as cache-able, and can then be cached
  - uniform interface between components
    - decouples structure from functionality
  - layering
  - client functionality can be adapted by code-on-demand
- Structure



(model in the process view from R. Fielding, Architectural Styles and the Design of Network-based Software Architectures)

- Typical behavior
  - as (layered) C&S, but only a single service (interface)



- Prime example: WWW

RESTful Web service

A RESTful Web service is a Web service

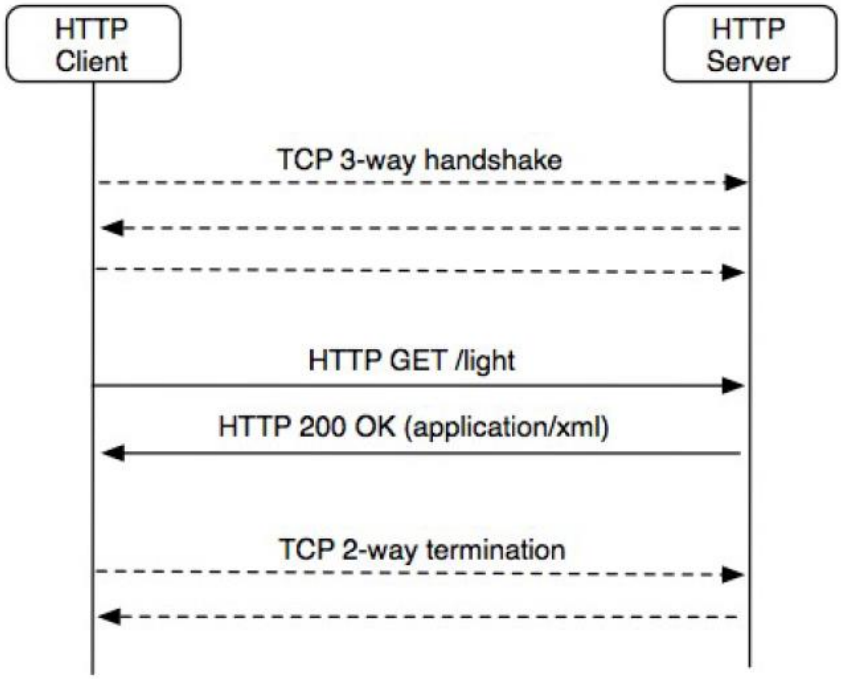
- that is resource-oriented, i.e., organized as a set of resources
  - Documents of various types are served to clients (similar to WWW)
  - Not only static documents, but also dynamically generated upon request
  - Resources may have multiple representations (e.g. in English and Chinese)
  - Format for representations is free, although XML is popular
- in which all operations on resources are cast in the form of a CRUD-operation (create, read, update, delete)
  - HTTP-messages (GET,PUT,POST,DELETE) are used to specify those operations;
  - URLs both specify the (location of) the resource and parameters for the operation; often a HTTP message consist of a header only, thus creating a light-weight protocol with a uniform interface
    - e.g. GET http://www.example.com/MyBookmarks?topic=webservices to obtain a list of bookmarks, instead of a RPC `getMyBookmarks(webservices)`

RESTful Web service

- that is stateless;
  - servers do not maintain *application state*, i.e., information from previous interactions with a client, hence the client must supply this information in successive operations
  - of course, servers maintain *resource state*, in the form of representations
- that supports client-side caching
  - Through HTTP cache control mechanisms
- that support code on demand
  - Servers may send code necessary to execute other representations they have sent
  - Usually JAVAscript

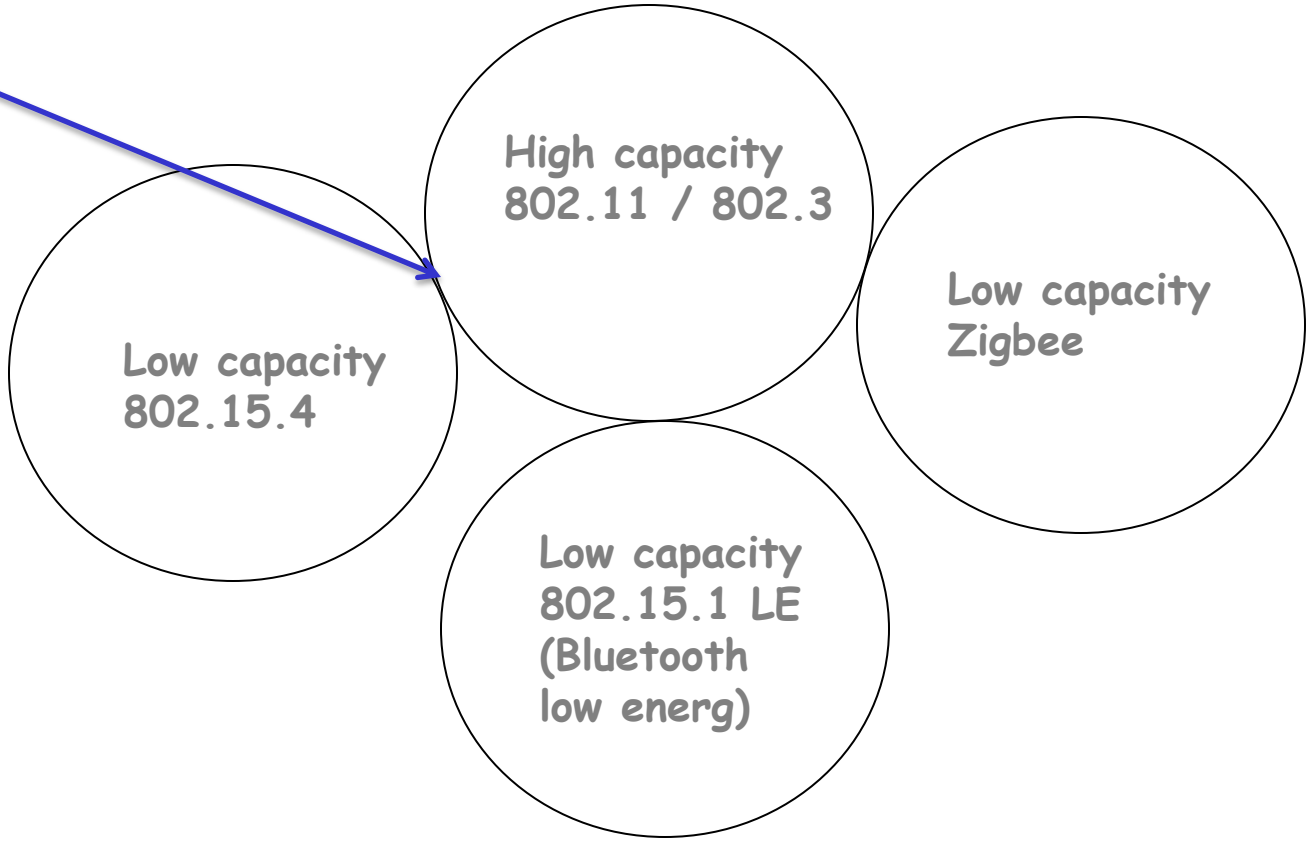
RESTful by HTTP

- Message exchange on top of a (reliable) TCP connection
- Significant overhead for (small) calls
- Small devices don't do TCP and can't process complex data (xml)

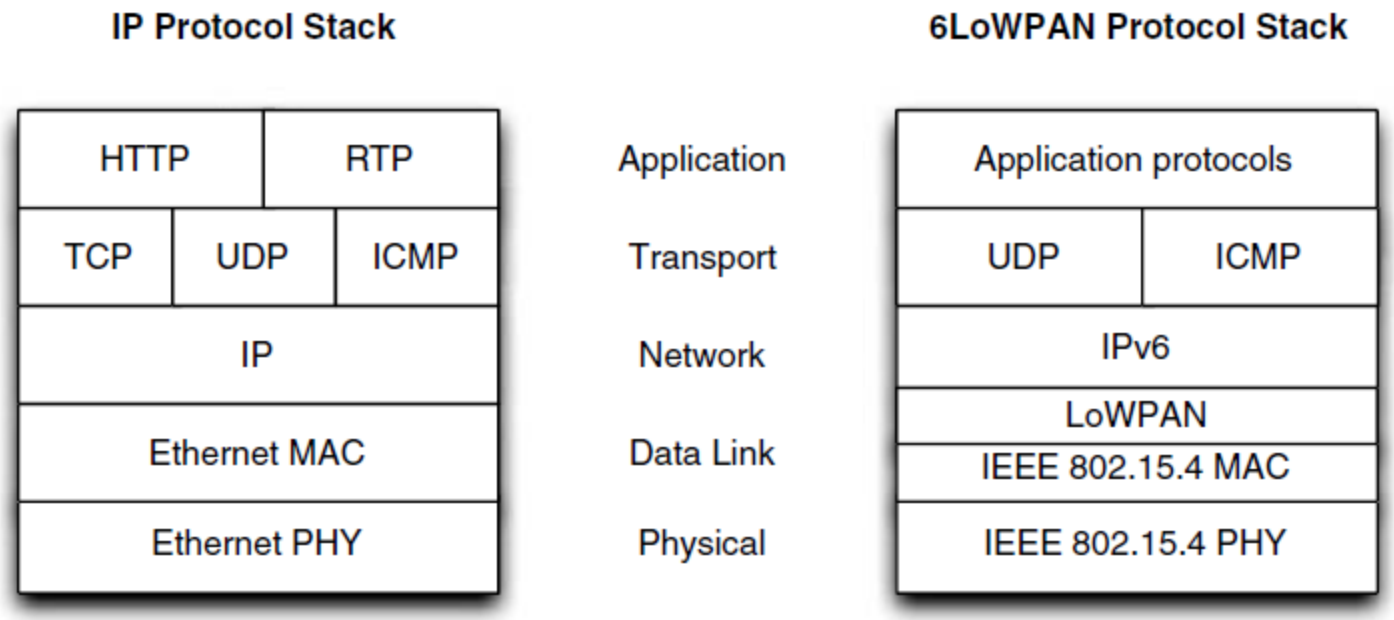


Low and High capacity

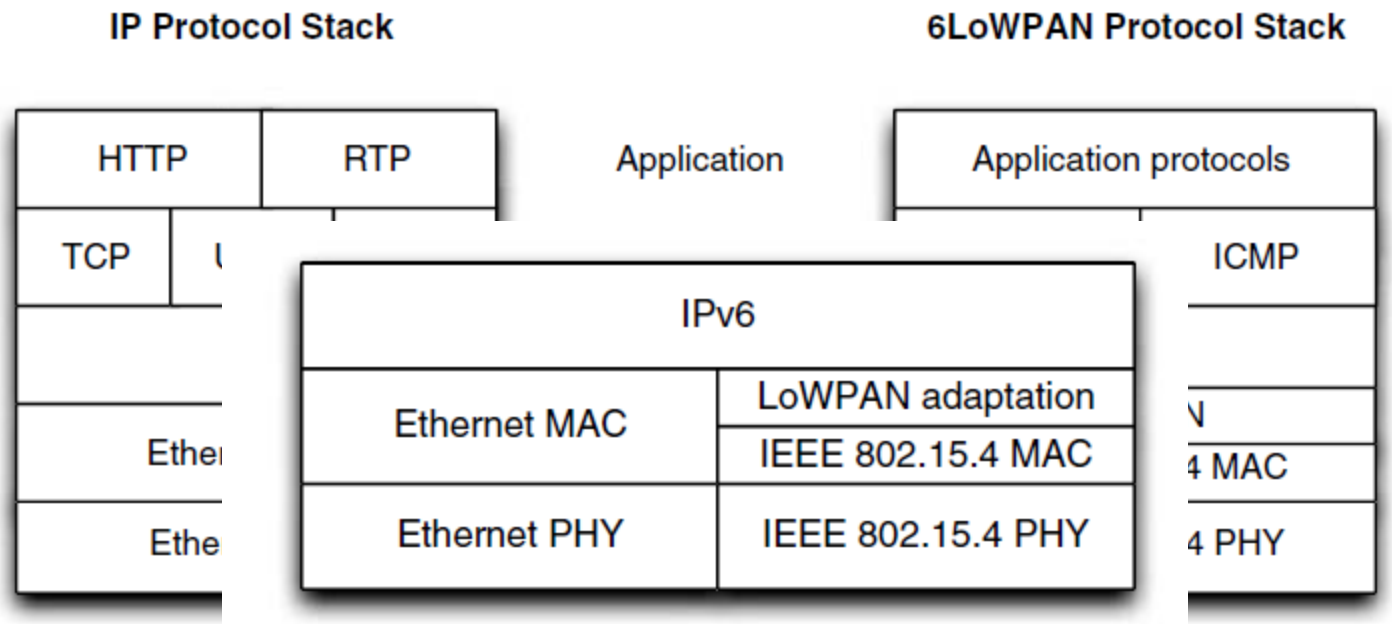
- Gateway:
  - at least change technology
- Other possible tasks:
  - adaptation to IP spec in LC domain
  - application level gateway



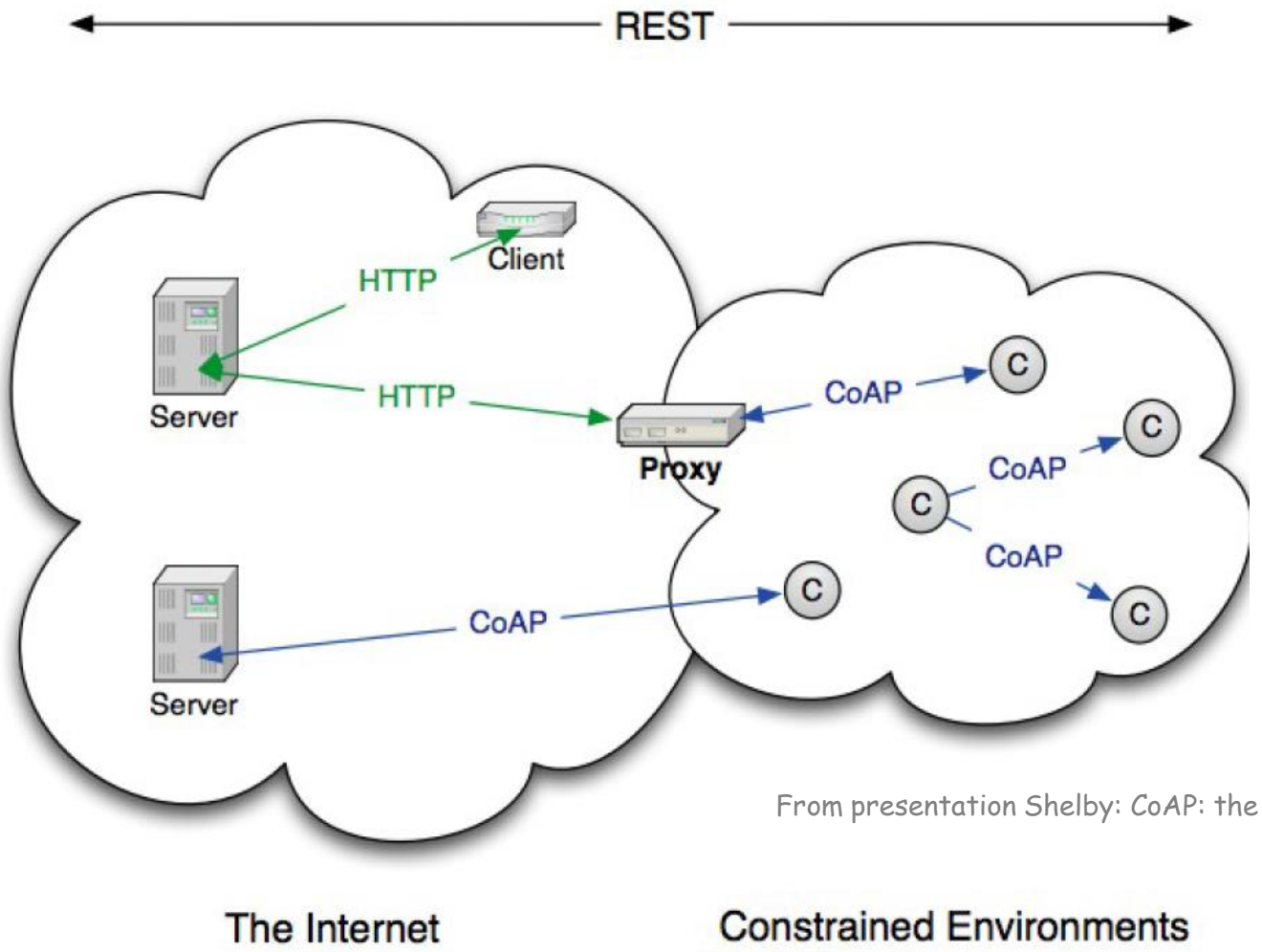
Example: connecting 802.15.4



Example: connecting 802.15.4



The CoAP Architecture



From presentation Shelby: CoAP: the IoT protocol