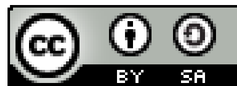
CMPUT 410: RESTful WebServices

Abram Hindle
abram.hindle@ualberta.ca
Department of Computing Science
University of Alberta
http://softwareprocess.es/



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

REST

- Representational
  - Issues related to representation, how to describe/name/show
- State
  - What is communicated
- Transfer
  - How to communicate

REST

- Architectural Style
  - Think design patterns for architecture
- Introduced by Fielding https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Method of calling remote objects
- Using URIs to Name Objects
- Using HTTP Verbs to manipulate them

Examples of REST

- https://secure.flickr.com/photos/dmelchordiaz/12441232743
  - User dmelchordiaz
  - Photo #12441232743
- https://api.github.com/users/abramhindle
  - JSON representation of abramhindle
- https://api.github.com/users/abramhindle/repos
  - Abram's repos
- https://api.github.com/repos/abramhindle/24bit-allrgb
  - A project of Abram's


```
{
  "login": "abramhindle",
  "id": 57137,
  "avatar_url": "https://gravatar.
%2F50cb3aa04fcde6eb1df8c0ff0b357fd
  "gravatar_id": "0cff4493d3270edd
  "url": "https://api.github.com/u
  "html_url": "https://github.com/
  "followers_url": "https://api.gi
  "following_url": "https://api.gi
  "gists_url": "https://api.github
  "starred_url": "https://api.gith
  "subscriptions_url": "https://ap
  "organizations_url": "https://ap
  "repos_url": "https://api.github
  "events_url": "https://api.githu
  "received_events_url": "https://
  "type": "User",
  "site_admin": false,
  "name": "Abram Hindle",
  "company": "Assistant Professor
  "blog": "http://softwareprocess.
  "location": "Edmonton, Alberta",
  "email": "my name at softwarepro
  "hireable": false,
  "bio": "* Software Engineering\r
  "public_repos": 92,
  "public_gists": 2,
  "followers": 63,
  "following": 24,
  "created_at": "2009-02-23T15:43:
  "updated_at": "2014-02-10T19:47:
}
```

Examples of REST

- http://api.stackexchange.com/2.2/search/excerpts?page=1&order=desc&sort=activity&q=ie6%20hide%20jquery&site=stackoverflow
  - Search with Stackoverflow
- http://api.stackexchange.com/2.2/posts/14556049?site=stackoverflow
  - A post from a user
- http://api.stackexchange.com/2.2/posts/14556049/comments?site=stackoverflow
  - With comments
- http://api.stackexchange.com/docs/answers

Examples of REST APIs

- Github's RESTful API
  - http://developer.github.com/v3/
- Flickr's RESTful API
  - https://www.flickr.com/services/api/request.rest.html
- Stackoverflow
  - http://api.stackexchange.com/docs/

Gist of it

- URIs refer to resources
  - Focus on URIs over parameters
- You do things (HTTP verbs) to resources
  - Delete something
  - Put something up
  - Get something
- You use existing infrastructure and caching rules (HTTP and HTTP User Agents and HTTP Proxies)

Why use HTTP for Remote Procedure Calls?

- Take advantage of performance aspects of HTTP
  - Caching
  - Proxies
  - Flexible Networking
  - Pluggable Middleware
  - Allows routing

REST Properties

- Client Server
- Stateless
- Cacheable
- Uniform Interface
- Layered

and the optional:

- Code on Demand (Javascript)

– Roy Thomas Fielding, “Architectural Styles and the Design of Network-based Software Architectures” 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

But REST is not RPC

- Remote procedure calls are calls made across a network with semantics of procedures and functions with inputs and outputs.
- HTTP has properties, headers, named URIs, verb and paramters. HTTP is transformable while remote procedure calls are not.
- “What distinguishes HTTP from RPC isn't the syntax. It isn't even the different characteristics gained from using a stream as a parameter, though that helps to explain why existing RPC mechanisms were not usable for the Web. What makes HTTP significantly different from RPC is that the requests are directed to resources using a generic interface with standard semantics that can be interpreted by intermediaries almost as well as by the machines that originate services. The result is an application that allows for layers of transformation and indirection that are independent of the information origin, which is very useful for an Internet-scale, multi-organization, anarchically scalable information system. RPC mechanisms, in contrast, are defined in terms of language APIs, not network-based applications.”

  – Roy Thomas Fielding, “Architectural Styles and the Design of Network-based Software Architectures” 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

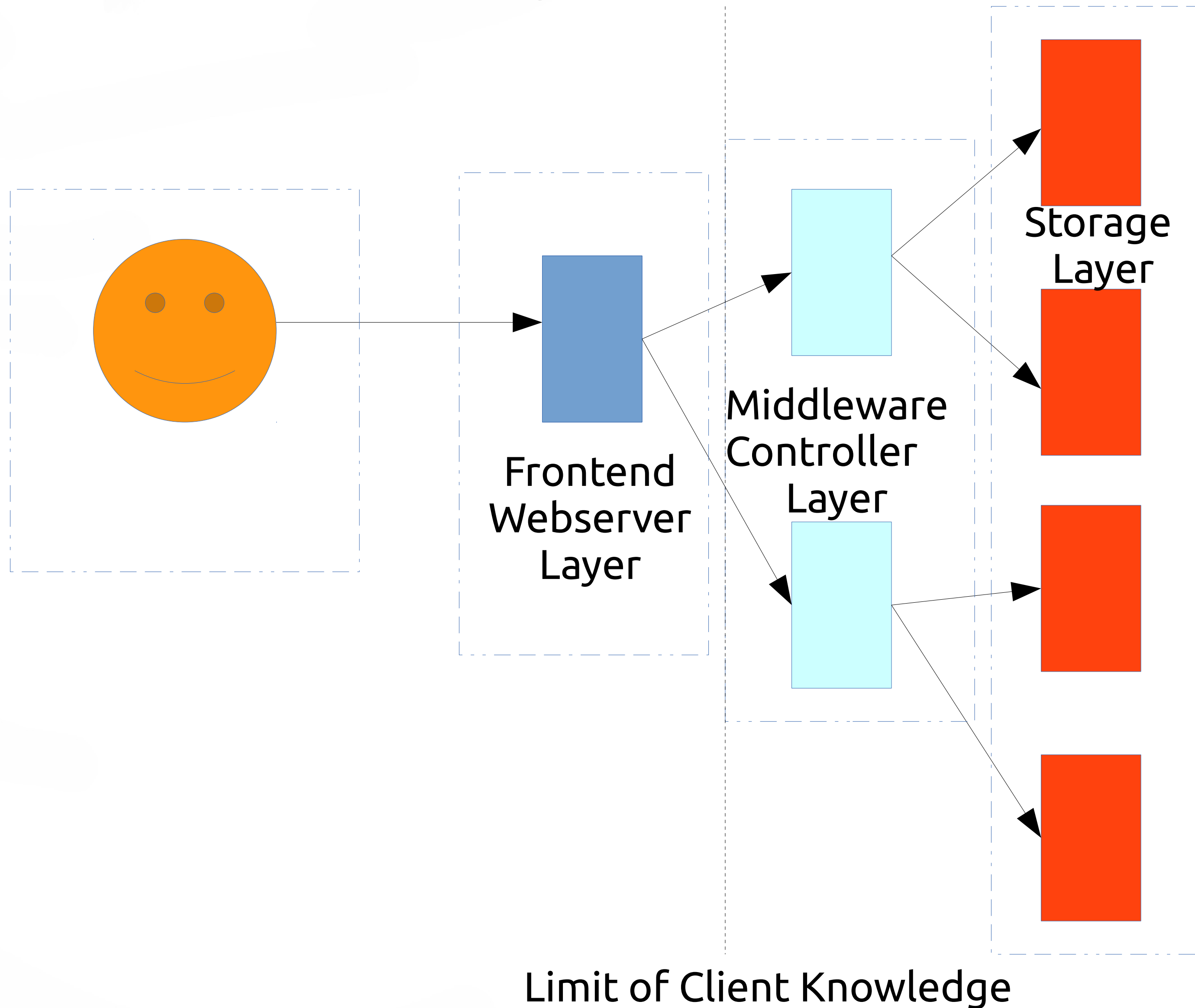
Stateless

- REST is stateless
  - in the sense that the client holds the state.
  - The server avoids holding client state
  - Requests are HEAVY with context. They are big!
- Statelessness is an important aspect of functional programming that allows for
  - Caching
  - Memoization
  - Transformation

Cacheable

- Browsers and clients can cache responses
- But so can secondary caches
- Will this entity change?
- If not why request it again?
- What verbs are cacheable?
  - Is POST cacheable?

Layered



Layering

- Your application might not be multi-machine
  - But it can have layering that operates with the HTTP request
  - You could have an auth layer that validates authentication and strips auth information before passing the request off to the next layer
  - HTTP Request handling allows routing and the decoration of routes to handle requests in a layered manner.

Verbs and Meaning in REST

- GET
  - Repeatable
  - Stateless
  - Cacheable
  - Safe
- PUT
  - Repeatable
  - Stateless
  - Cacheable
- DELETE
  - Repeatable
  - Stateless
  - Cacheable
- POST
  - Anything goes
  - Not cacheable

Examples – Do you even REST?

- Search Queries?
  - [ ] Safe?
  - [ ] Repeatable?
  - [ ] Cacheable
  - [ ] Stateless

Examples – I can't put up with this

- Weather
  - [ ] Safe?
  - [ ] Repeatable?
  - [ ] Cacheable
  - [ ] Stateless

Examples – Pain in the POSTerior

- A Photo
  - [ ] Safe?
  - [ ] Repeatable?
  - [ ] Cacheable
  - [ ] Stateless

Examples – I'm going to CTRL-ALT-DEL

- Payment
  - [ ] Safe?
  - [ ] Repeatable?
  - [ ] Cacheable
  - [ ] Stateless

Examples – I don't GET it

- Authentication
  - [ ] Safe?
  - [ ] Repeatable?
  - [ ] Cacheable
  - [ ] Stateless

Authentication

- Why aren't session IDs stateless?
- When are cookies “stateless”
- The suggestion is do authentication via headers
  - HTTP-Auth
  - HTTP-Digest
  - Or your own headers

Design Tip

- Sometimes a URL is for both HTML and XML/JSON
- Some applications make a seperate api domain that supports api calls and REST calls, meaning their website is aimed at browsers but can interact with the API domain.
- HTML representationhttp://abramstuff.com/user/abram
- Versus an XML/JSON representation http://api.abramstuff.com/user/abram
- Alternatively use the Accept headers

Performance: Caching

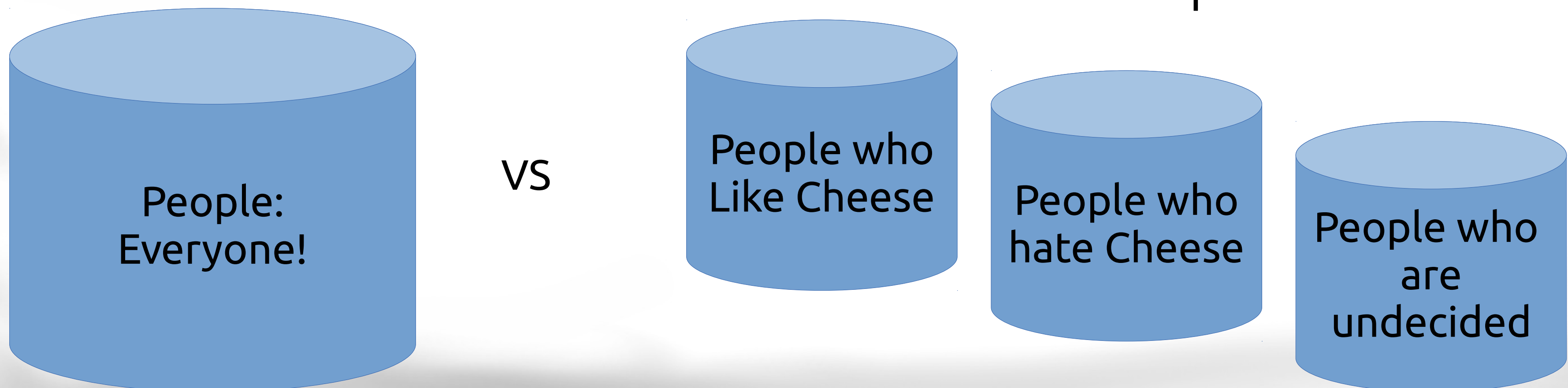
- But we're using HTTP
  - But we can increase locality
    - In browser
    - In layers
    - Cache between layers
    - Cache transformations
- Caching is the most common performance trick on the web today:
  - http://memcached.org/
  - In Memory Key store, can cache common and large objects

Performance Claims

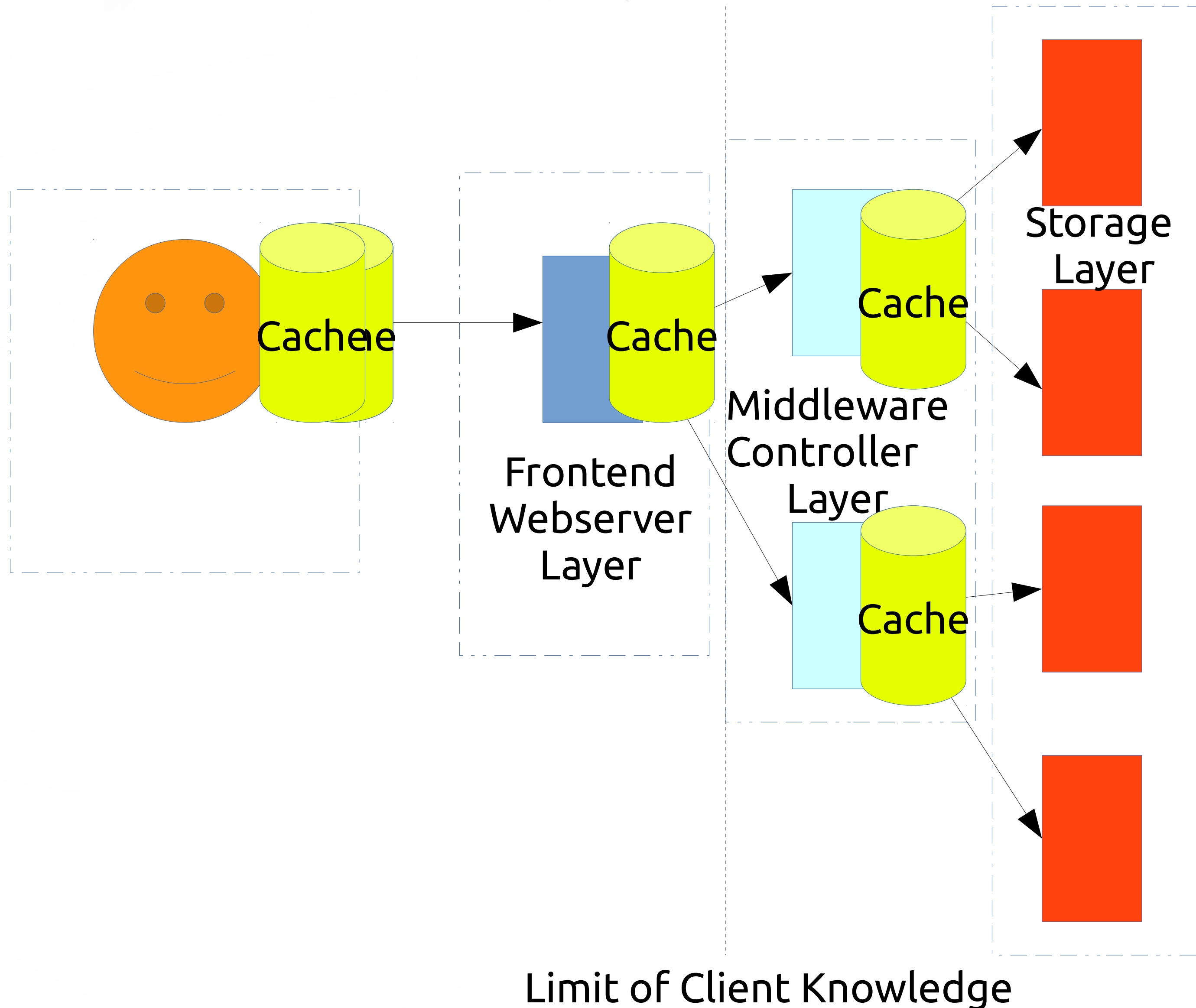
- Reliability
- Scalability
- Latencies
- Efficiency
- Distributivity

Performance: Layering

- Improve reliability by layering and making more machines responsible for the same task
  - If one machine dies, the middleware can just route around the failure
  - Distributed operations are easy
- Improve responsiveness by relying on more than 1 machine and returning the most immediate result.
- Can shard data across machines if it independent



Performance: Layered + Caching



Performance

- Degrades
  - Optimization via state
  - Network performance using TCP and HTTP
  - Bandwidth, requests are large
  - Every layer of abstraction is a danger to performance
  - Multiple layers increases latency

Photo Gallery Example

- http://photogallery/photog/Abram
  - An example photographer
- http://photogallery/photog/Abram/photos
  - Photos of Abram
- http://photogallery/photog/Abram/set/portfolio
  - Sets of Photos
- http://photogallery/photo/7774273
  - A photograph stored on the system

Photo Gallery Example

- http://photogallery/photog/Abram
  - GET PUT DELETE POST ?
- http://photogallery/photog/Abram/photos
  - GET PUT DELETE POST ?
- http://photogallery/photog/Abram/set/portfolio
  - GET PUT DELETE POST ?
- http://photogallery/photo/7774273
  - GET PUT DELETE POST ?

Design Tips

- Consider what is world readable, what the read or read-only interface
- What needs to be private or authenticated?
- Remember OOA?
  - Think about your nouns
  - Think about your verbs
  - Think about what relationships can be become URIs

Design Tips

- Take advantage of caching.
  - Use GETs avoid POST
- Consider how independent data is
  - Can you split it up safely?
  - How coupled is it?
  - Does it need to be coupled?
- Heavy state can stored client side.
  - Take advantage of locality, your client side is the fastest side.

Verbs+Nouns?

- GetAllTweets
  - NO!
    - tweets/
- GetATweet?
  - No!
    - Tweets/3423
- Can we use POST?
  - Use POST like a PUT when you don't have an ID

Should we mass update tweets on a PUT?

Nouns

- Brian Mulloy https://blog.apigee.com/detail/restful_api_design_plural_nouns_and_concrete_names/

  Recommends:
  - Usually you chose a noun because you have a collection of items. Maybe use the plural.
  - Use good concrete names. Don't generalize, you can generalize the routes in your code but be concrete
    - e.g. item versus todoNote
    -

General REST recommendations

- Vinay Sahni suggests http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api
  - Focus noun first, then do verbs to those nouns
  - Plural URLs (tweets/)
  - Relations relation to first object /tweets/2/responses and /tweets/2/responses/4
  - Use /verb for obviously important tasks like /search
  - Version the API in the URL
  - Search using get params
  - Use Status code 201 for creation of a new resource
  - Use JSON
  - Don't wrap data up by default {“data”: } versus {“my”:”object”}
  - For large data use pages
  - Use simple HTTP auth + SSL over anything more complicated

HATEOAS

- Hypermedia As The Engine of Application State
  - This means include hyperlinks in your REST API Responses!
  - Either “link”:”http://url-to-this-obj”
  - Give hints to where the app or the reader could go to next.
  - It's the web! Use hypermedia!
    - Furthermore if you use STATUS CODE 201 you are using Location: to allow redirection to the new object!

Errors

- Vinay Sahni suggests:
  - 200 OK
  - 201 Created – creation of object with POST, redirection via Location: to new object
  - 204 No Content – respond to a delete
  - 304 Not Modified – you've cached it
  - 401/403/404
  - 405 – method not allowed (don't POST here!)

PUT

- Should you make a new object on a PUT?
- Or just update?
- Should you force creation via POST?

Books

- RESTful Web Services
  - http://restfulwebapis.org/RESTful_Web_Services.pdf
  - http://proquest.safaribooksonline.com.login.ezproxy.library.ualberta.ca/book/web-development/web-services/9780596529260
- Restful Web Services CookBook
  - http://proquest.safaribooksonline.com.login.ezproxy.library.ualberta.ca/book/web-development/web-services/9780596809140

Code

- flask-restful
  - http://flask-restful.readthedocs.org/en/latest/quickstart.html

Resources: RFCs

- URIs https://tools.ietf.org/html/rfc3986
- HTTP http://tools.ietf.org/html/rfc2616
- Best Practices for Restful APIs http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

Videos

- SOAP versus REST
  - http://www.youtube.com/watch?v=v3OMEAU_4HI
- Intro to REST
  - http://www.youtube.com/watch?v=llpr5924N7E
- Intro to REST
  - http://www.youtube.com/watch?v=YCcAE2SCQ6k
- Intro to REST API Style HATEOAS
  - https://blog.apigee.com/detail/hateoas_101_introduction_to_a_rest_api_style_video_slides