

# CS 4350: Fundamentals of Software Engineering

## Lesson 4.3: REST Protocols

---

Jonathan Bell, Adeel Bhutta, Ferdinand Vesely, Mitch Wand  
Khoury College of Computer Sciences

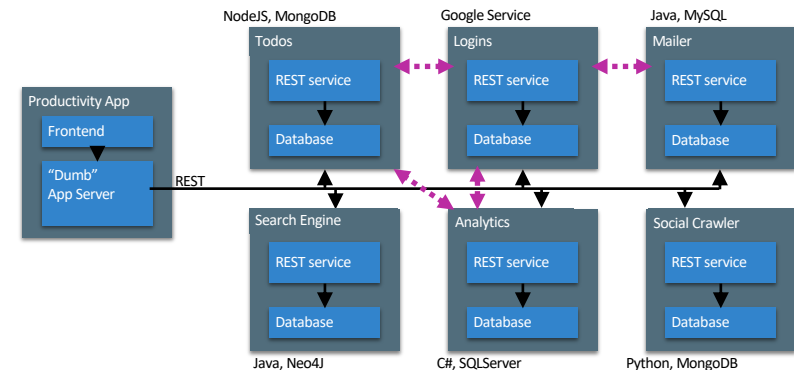
# Learning Objectives for this Lesson

---

- By the end of this lesson you should be able to:
  - Explain the basic principles of RESTful protocols
  - Examine a protocol and suggest ways in which it either adheres to or violates the REST principles.

# Your app relies on other apps for services

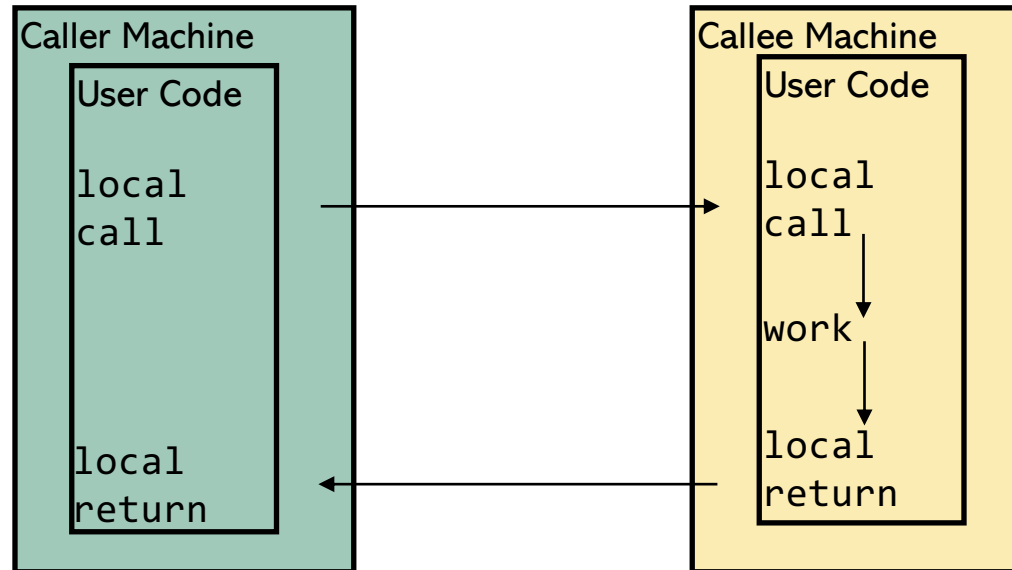
- Authentication (Login with Google/Apple/Facebook)
- Sending/receiving email (SendGrid, MailGun, MailChimp)
- Telephony, text messaging, video chat (Twilio)



# What we'd like

---

- A magic abstraction: remote procedure call (RPC)



# Obstacles to magic RPC

---

- transmission delays (latency)
- can the client do something useful in the meantime?
  - asynchrony
  - "mask latency with multiprocessing" → complexity
- client/server mismatch
  - different languages,
  - different data representations
  - → wire-transmission formats
  - → more complexity

# A Solution(?): use the web!

---

- Implement your protocol via http.
- Of course, then you have to define your protocol
- You'll want to define it in some standard metalanguage, so client and server can agree on its meaning.
- But that means the client-human and server-human have to agree on a standard metalanguage
- Lots of choices: XML/RPC, SOAP, WSDL, or ...

XML/RPC or SOAP  
or REST or ...

HTTP

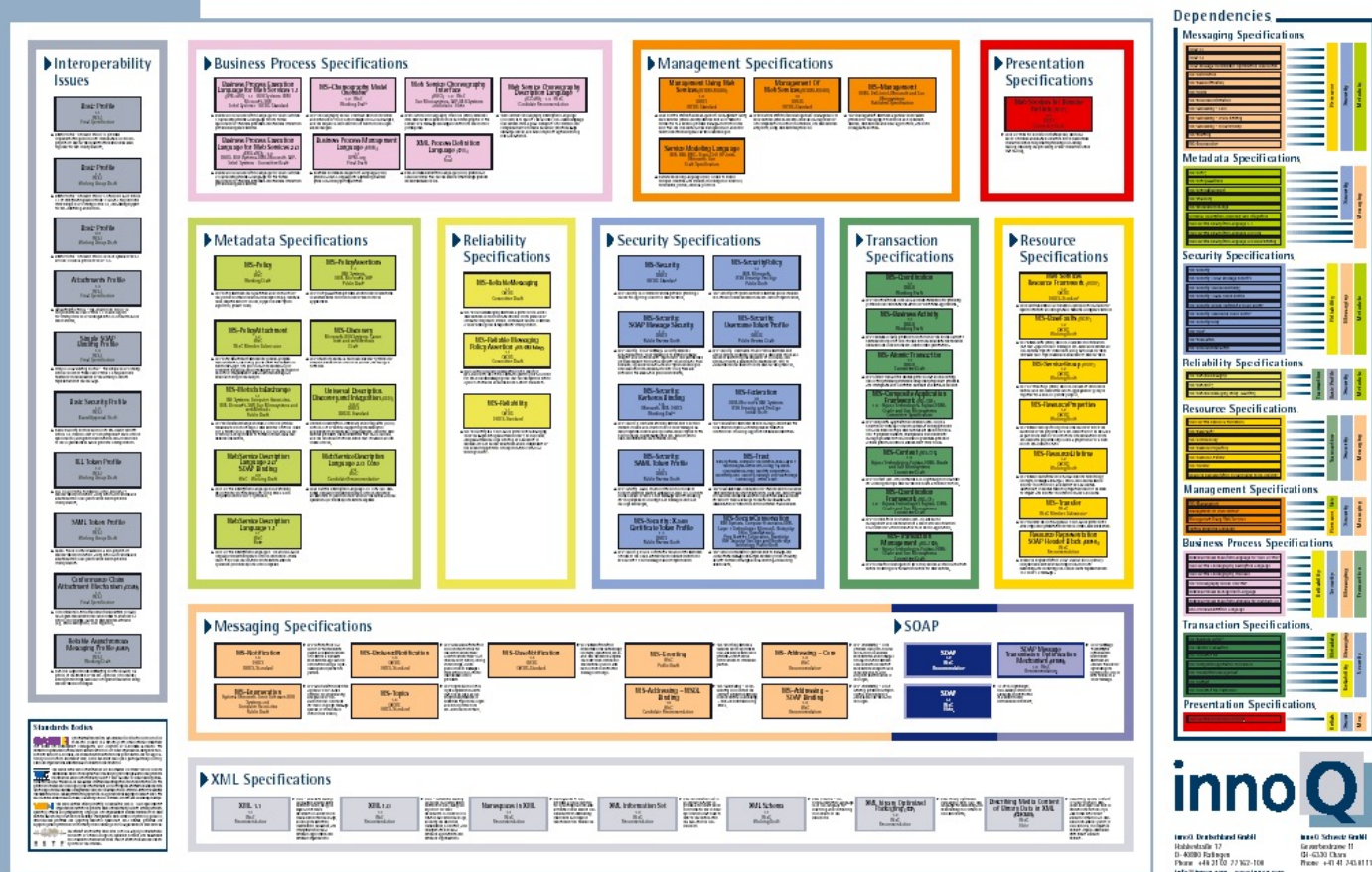
TCP

Network layer

Link layer

# Aagh!

# Web Services Standards Overview



Now take a deep breath, and start again...

---



# A Typical HTTP Request

---

```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

- This request probably started out as `http://www.nowhere123.com/docs/index.html`
- [www.nowhere123.com](http://www.nowhere123.com) identifies the host (the server's location)
- the rest of the request is the **path**, here `/docs/index.html`
- this might be a path in the server's file system,
- OR it could be anything at all—
- it's entirely up to the server to interpret the path

# That means the client can ask the server to do things other than retrieve files

---

- Just has to be an **agreement** (a **protocol**) between client and server about how these tasks are to be described.
- Need a general framework to help us design such protocols.
- We will talk about one such philosophy, called **REST**

# REST: Representational State Transfer

---

- Defined by Roy Fielding in his 2000 [Ph.D. dissertation](#)
- “Throughout the HTTP standardization process, I was called on to defend the design choices of the Web. That is an extremely difficult thing to do... I had comments from well over 500 developers, many of whom were distinguished engineers with decades of experience. That process honed my model down to a core set of principles, properties, and constraints that are now called REST.”
- Not just a transport protocol, not a protocol definition language: a design philosophy
- Interfaces that follow REST principles are called **RESTful**

# REST Principles

---

- Single Server
  - Client calls server, server responds. That's it.
  - Separation of concerns: client doesn't worry about data, server doesn't worry about UI
  - Server may pass request on to other machines, but that's not visible to the client
- Stateless
  - No session state in the server
  - Each client request must contain all the information the server needs to process the request
- Uniform Cacheability
  - requests must classify themselves as cacheable or not.
- Uniform Interface
  - associate URIs with resources

# Single Server

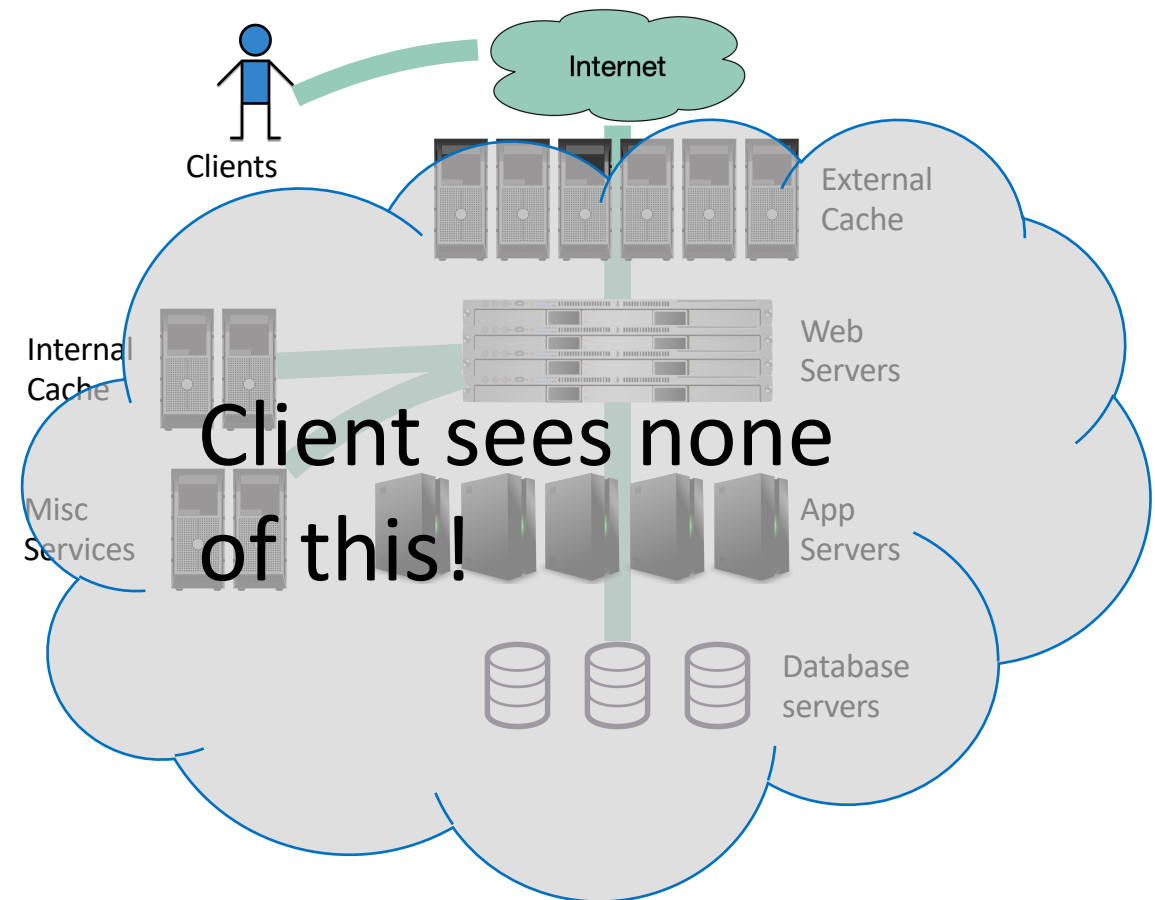
---

- Server is abstracted as a single box
- Client calls the server, server doesn't call the client
- Enables separation of concerns:
  - Client doesn't worry about how the server does its business
  - Server doesn't worry about UI



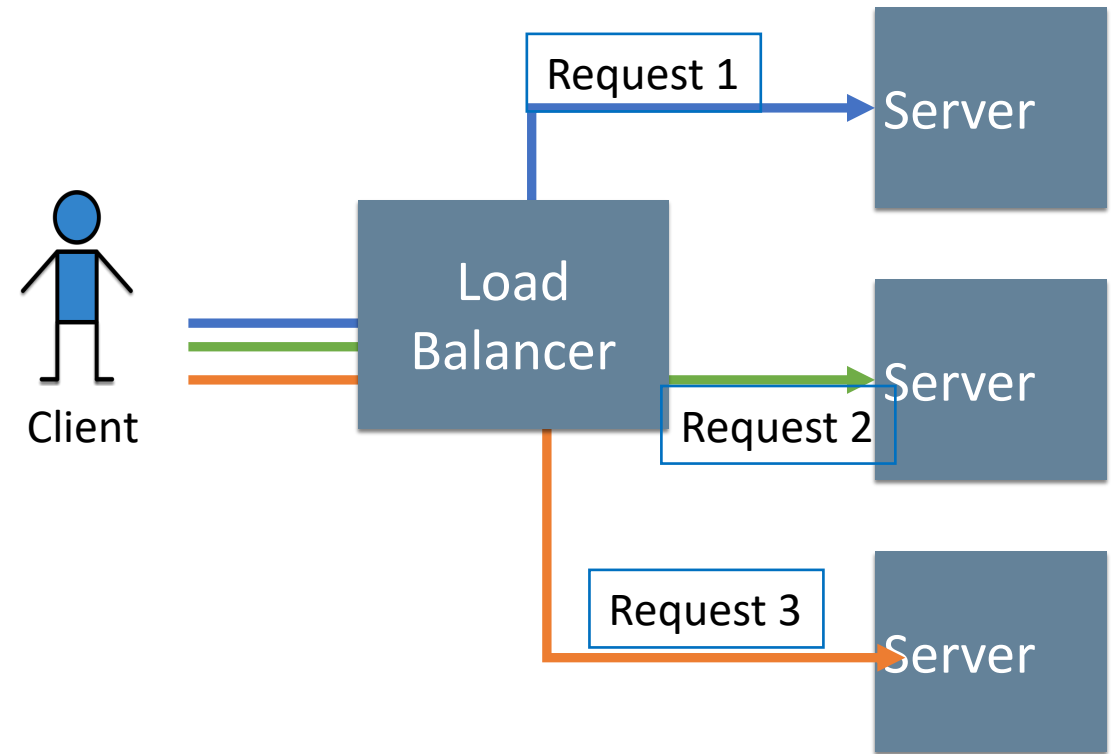
# Client sees only a single server

- Enables flexible design: different servers can have different responsibilities, but client sees just a single server



# Stateless

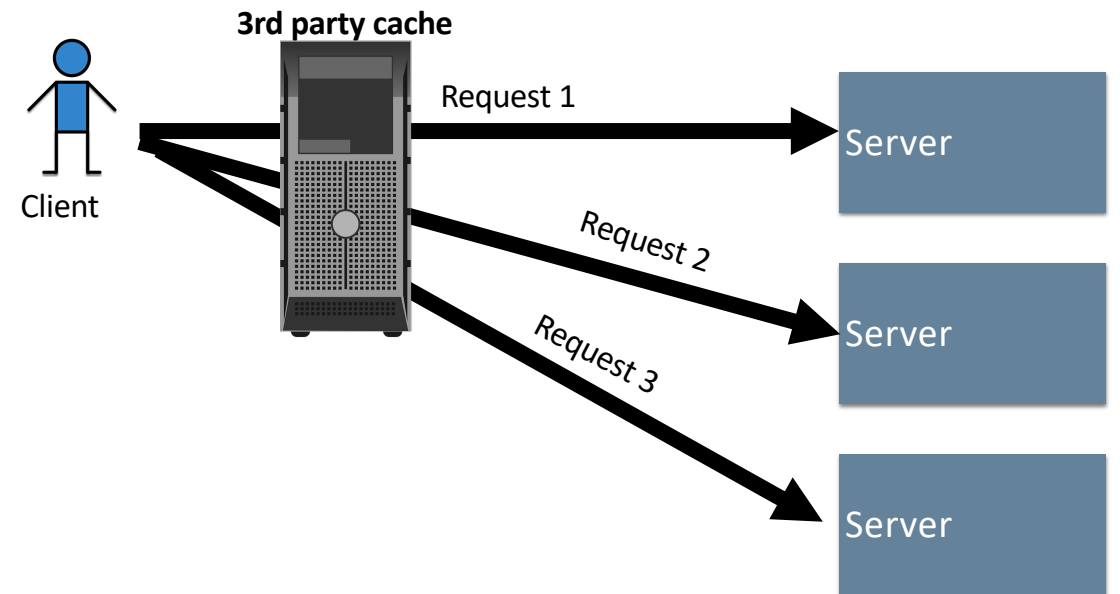
- Each client request contains all information necessary to service the request
  - The client doesn't have to write a sequence of requests to get their work done.
  - So requests can be farmed out to different servers



# Uniform cacheability

- Requests and responses are clearly classified as cacheable or not
- Enables use of generic caches that don't know anything about the structure of what they cache - just what can be cached

This involves more systems stuff than we will normally get involved with, so you don't have to worry about this immediately.





# Uniform Interface

---

- URIs should hierarchically identify **nouns** describing resources that exist
- Actions that can be taken with resources are specified by the HTTP methods (**verbs**)
  - more on this later

# Nouns are represented as URIs

---

- In a RESTful system, the server is visualized as a store of resources (nouns), each of which has some data associated with it.
- URIs represent these resources
- Examples:
  - /cities/losangeles
  - /transcripts/00345/graduate (student 00345 has several transcripts in the system; this is the graduate one)
- Anti-examples:
  - /getCity/losangeles
  - /getCitybyID/50654
  - /Cities.php?id=50654

We prefer plural nouns for toplevel resources, as you see here.

Useful heuristic: if you were keeping this data in a bunch of files, what would the directory structure look like? But you don't have to actually keep the data in that way.

# Verbs are represented as http methods

---

- In REST, there are four things you can do with a resource
- POST: requests the server to create a resource
  - there are several ways in which the value for the new resource can be transmitted (more In a minute)
- GET: requests the server to respond with a representation of the resource
- PUT: requests the server to replace the value of the resource by the given value
- DELETE: requests the server to delete the resource

# You say you want parameters?

---

There are at least 3 ways to associate parameters with a request:

- **path parameters**. These specify portions of the path to the resource. For example, your REST protocol might allow a path like

`/transcripts/00345/graduate`

- **query parameters**. These are part of the URI and are typically used as search items. For example, your REST protocol might allow a path like

`/transcripts/graduate?lastname=covey&firstname=avery`

- **body parameters**. You can put additional parameters or information in the body, using any coding that you like.

# Example interface #1: a todo-list manager

---

- Resource: /todos
  - GET /todos - get list all of my todo items
  - POST /todos - create a new todo item (data in body)
- Resource: /todos/:todoItemID
  - :todoItemID is a path parameter
  - GET /todos/:todoItemID - fetch a single item by id
  - PUT /todos/:todoItemID - update a single item (new data in body)
  - DELETE /todos/:todoItemID - delete a single item

# Example Interface #2: a database of transcripts

Remember the heuristic:  
if you were keeping this  
data in a bunch of files,  
what would the directory  
structure look like?

POST /transcripts

- adds a new student to the database,
- returns an ID for this student.
- requires a body parameter 'name', url-encoded (eg name=avery)
- Multiple students may have the same name.

GET /transcripts/:ID

- returns transcript for student with given ID. Fails if no such student

DELETE /transcripts/:ID

- deletes transcript for student with the given ID, fails if no such student

POST /transcripts/:studentID/:courseNumber

- adds an entry in this student's transcript with given name and course.
- Requires a body parameter 'grade', url-encoded
- Fails if there is already an entry for this course in the student's transcript

GET /transcripts/:studentID/:courseNumber

- returns the student's grade in the specified course.
- Fails if student or course is missing.

GET /studentids?name=string

- returns list of IDs for student with the given name

Didn't seem to fit  
the model, sorry ☹

# Review: Learning Objectives for this Lesson

---

- You should now be able to:
  - Explain the basic principles of RESTful protocols
  - Examine a protocol and suggest ways in which it either adheres to or violates the REST principles.