Principled Design of the Modern Web Architecture

Papers We Love - Athens - March 2018

Anastasopoulos Spyros
@anastasop

The Paper

Roy Fielding’s influence on the design of URI, HTTP and the modern Web

- Architectural Styles and the Design of Network-based Software Architectures
  - PhD dissertation
  - Chapter 5: Representational State Transfer
- Principled Design of the Modern Web Architecture [PDF]
  - Web Architecture - International Conference on Software Engineering 2000
- Principled Design of the Modern Web Architecture [PDF] ←
  - Transactions on Internet Technology, Vol. 2, No. 2, May 2002

A two part paper that assumes familiarity with the state of the web at the time:

1. High level description of the REST architectural style, goals and benefits
2. How it influenced the standardization of the web

World Wide Web

Tim Berners-Lee: “A shared information space through which people and machines could communicate” - TBL book “weaving the web” [amazon]

Born out of frustration for heterogeneous environments and a vision for consistent access to structured information

- 1989 - Initial proposal
- 1990 - Working prototype
- 1991 - First publication outside CERN
- 1993 - CERN release
- 1994 - w3c

World Wide Web

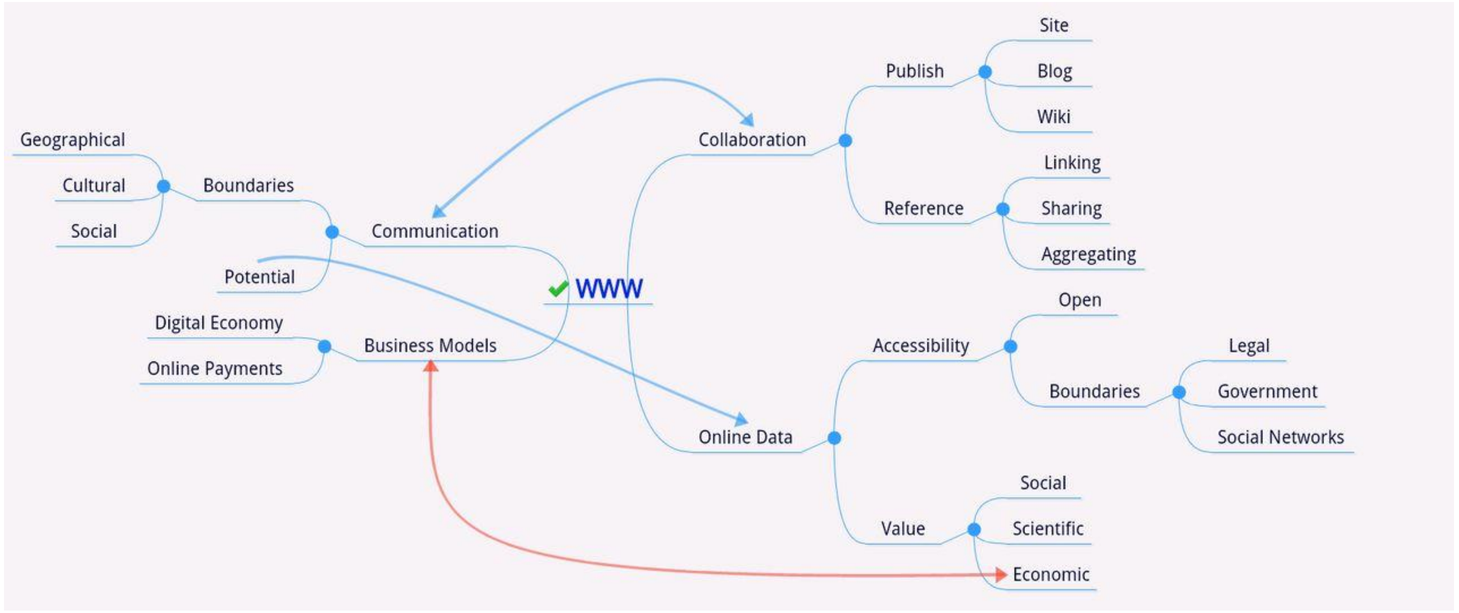
- The combination of internet and hypertext
  - Internet gives distribution and scalability
  - Hypertext structures the information space
- Glue
  - URL a system of globally unique identifiers for resources
  - HTML the publishing language HyperText Markup Language
  - HTTP the Hypertext Transfer Protocol
- Internet Scale Distributed Hypermedia
  - Network performance is important
  - Anarchic scalability: multiple organizational boundaries
  - Independent deployment: evolution and extensibility

World Wide Web

- Low entry barrier
  - HTML could be written with a simple text editor
  - HTTP was easy to monitor even with `nc,` easy to implement (`httpd.c ~1000 loc`)
  - Browser: type url, click link, back, forwards, refresh, bookmark
  - Server: free and open source
  - Hosting: `~/public_html/index.html`
- Continuous evolution and change
- User experience: hypertext driven

World Wide Web - A landmark in history

A mind map of mine created from a post of Tim Berners-Lee on the 25th birthday



World Wide Web - The next steps

- Scale
- Evolution
- Integration
- Access
- Data Web
- Semantic Web
- Web Services
- Web Applications
- Mobile Web
- Business Models
- And much much more

World Wide Web - Formalization & Standardization

Roy Fielding’s PhD dissertation: “Architectural Styles and the Design of Network-based Software Architectures”

Chapter 5: Representation Style Transfer(REST)

Heavily Influenced the Design of

RFC 7230 family - HTTP/1.1

RFC 3986 - URI

Architecture of the World Wide Web, Volume One

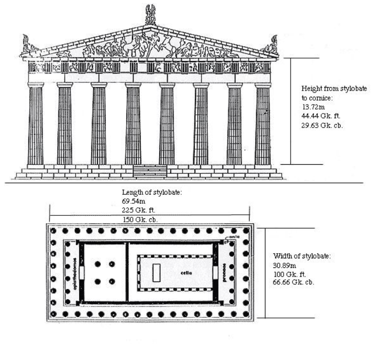
Enter REST

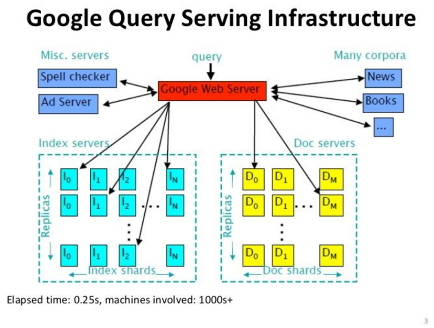
“The Representational State Transfer(REST) style is an abstraction of the architectural elements within a distributed hypermedia system.”

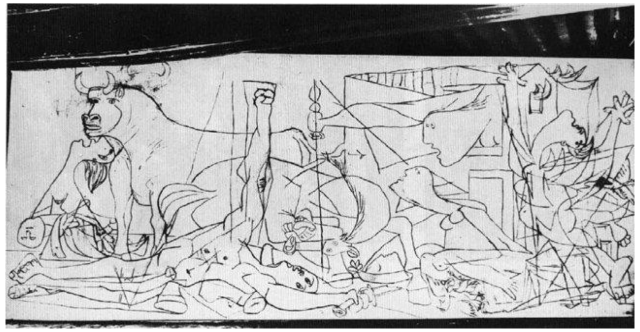
How to build an information space on top of a data network?

Architecture - A high level description

- An abstraction of the system runtime elements during a phase of operation
  - “the server accepts a request, asks the cache, writes to storage, sends to broker”
  - “A scheduler extracts data, forms batches and distributes jobs across the cluster”
- Determines how system elements are identified, interact and communicate
  - 3-tier architecture: presentation, application, business, data access layers

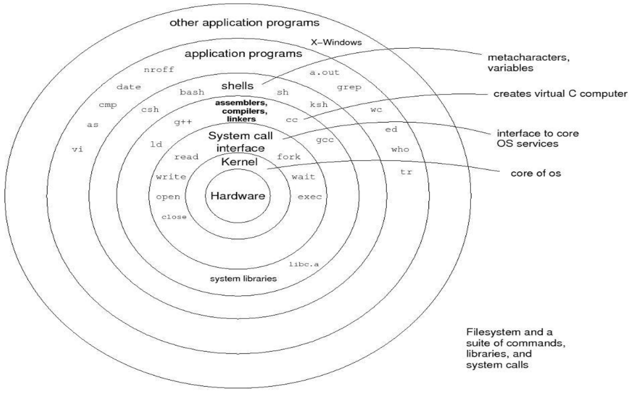

Height from stylobate to cornice: 13.72m 44.44 Gk. ft. 29.63 Gk. cb.
Length of stylobate: 69.54m 225 Gk. ft. 150 Gk. cb.
opisthodomos
cella
pronaos
Width of stylobate: 30.89m 100 Gk. ft. 66.66 Gk. cb.





Architectural Style

- A coordinated set of architectural constraints
  - “Presentation must not talk directly with business”
  - “Data storage must not be exposed to application tier”
  - “Do one thing and do it well”
- It provides a name for a set of design decisions and properties
  - 3-tier
  - UNIX


other application programs
X-Windows
application programs
nroff
a.out
date
grep
shells
bash
sh
cmp
csh
ksh
wc
assemblers, compilers, linkers
as
g++
cc
ed
System call interface
vi
ld
gcc
who
read
fork
Kernel
tr
write
wait
open
Hardware
exec
close
libc.a
system libraries
metacharacters, variables
creates virtual C computer
interface to core OS services
core of os
Filesystem and a suite of commands, libraries, and system calls

Conceptual Architecture of UNIX SYSTEMS

REST Style - Client/Server

Clients ask for data and can manipulate it, servers can only provide data

- Promotes separation of concerns: UI, storage
  - Improves UI portability
  - Simplifies servers and helps scaling
  - Supports independent evolution of clients and servers
  - Allows multiple organizational domains

➔ Clients move in the information space, servers are fixed

REST Style - Stateless Interactions

No context stored on the server, session state entirely on client

- Promotes
  - Visibility: monitoring tools should only examine request data
  - Reliability: easier recovery from failures
  - Scalability: less resources on server, simpler implementation
- Demotes
  - Performance: many network requests
  - Consistency: applications must support multiple different types of clients

➔ Very difficult to handle state, context and mutability at internet scale

REST Style - Caching

Data in requests and responses must be identified as cacheable or not

- Promotes
  - Scalability
  - Efficiency
  - Performance
- Demotes
  - Reliability

➔ A well known pattern to increase performance which was hurt by stateless.

REST Style - Layered Structure

There are hierarchical layers and each component can see only the immediate layer of interaction: client - proxy, client - cache, reverse proxy - server

- Promotes
  - Legacy encapsulation
  - Simpler components by using intermediaries: balancing, shared caching, security
  - Scalability
- Demotes
  - Network performance: adds overhead and latency

➔ This is to handle complexity at internet scale. Reminds of IP gateways.

REST Style - Code on Demand (Optional)

A client can extend its functionality by downloading code

- Useful for multiple organizational domains
- Hurts visibility
- Defined with VMs in mind: JVM, Silverlight, Flash, JS, WebAssembly

➔ Extends client capabilities for manipulating data
➔ Is not applied to the uniform REST interface
➔ Misused. Much work on presentation, not much on capabilities enhancements

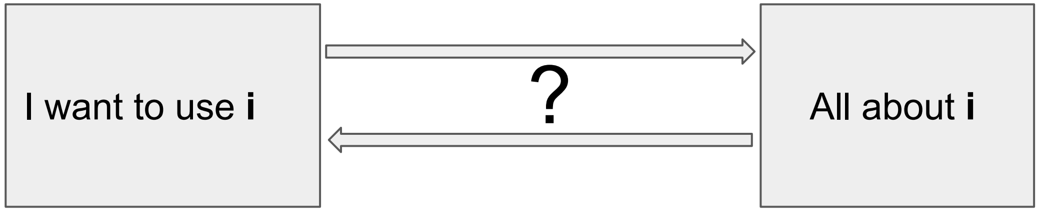
REST Style - Uniform Interface

All components adhere to a single, uniform interface

- Generality, simplicity, visibility, decoupled implementation, evolvability
- Degrades efficiency, can’t be suitable for all kind of applications
- Optimized for large-grain distributed hypermedia systems like the web
- It matters the uniformity of the interface not its operations

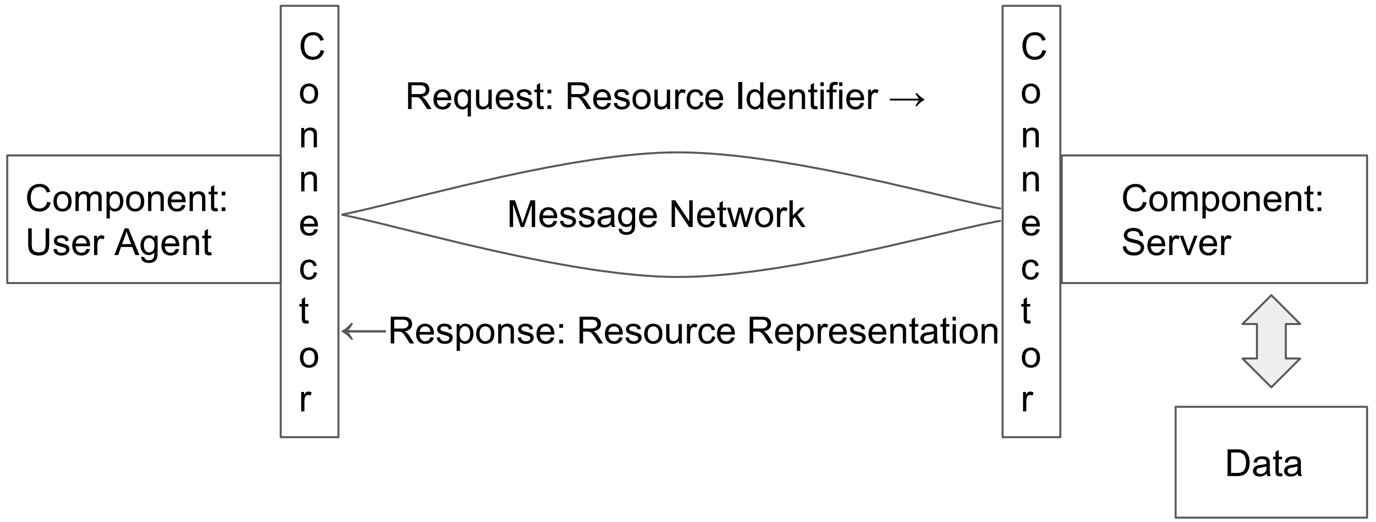
➔ The uniform interface was most successful in UNIX filters and pipes. It is a building block for OO and the cornerstone of REST

Client/Server: Many Decisions



- Data
  - Service, Object, Database (table, row), File system(path), Text
- Operations
  - `incrCounter` vs `getCounter && setCounter`
- Protocol
  - Textual, binary, sessions, transactions

Bird’s eye view of REST



REST is optimized for distributed hypermedia systems

Data Elements

- Information is moved from the location is stored to the location it will be used
  - Quite the opposite of object models
- Information Hiding becomes an issue
  - In essence how much freedom the client has and how coupled is to the server
    - Full: expose storage details
    - None: send a pre-rendered result
    - Some: ACLs, VMs, Restricted interface

REST hybrid approach: A restricted interface to a representation of the resource

Resources

- Resource is any information that can be named
- Resource is a conceptual mapping to a set M of equivalent entities
- The entities set can be static or variant or even empty
- Identified via Resource Identifiers
  - Should not reflect location instructions or storage implementation
  - Ideally they are opaque identifiers
- REST provides an interface to access and manipulate representations of M
  - Identification must be separated from interaction
  - REST is communication protocol independent
- The naming authority is responsible for the semantic validity of the mapping
  - Impossible to have centralized link servers on the internet

Resource Representations

- Captures the current or intended state of a resource
- Transferred between components
- A representation consists of
  - Data
    - Must be from a standard media type
  - Metadata: usually standardized name-value pairs
    - Resource metadata
    - Representation metadata

Connectors

- Access resources and transfer resource representations
- Exchange of self-descriptive messages (representations + metadata)
- Used by components
  - Separation of concerns
  - Hiding implementation of resources
  - Hiding communication mechanisms
- All interactions are stateless
  - Connectors do not store application state. Less physical resources are needed
  - Parallel processing of interactions
  - Intermediaries can understand requests in isolation
- Request: resource id, representation, metadata(request, control, resource)
- Response: representation, response id,metadata(response, control, resource)

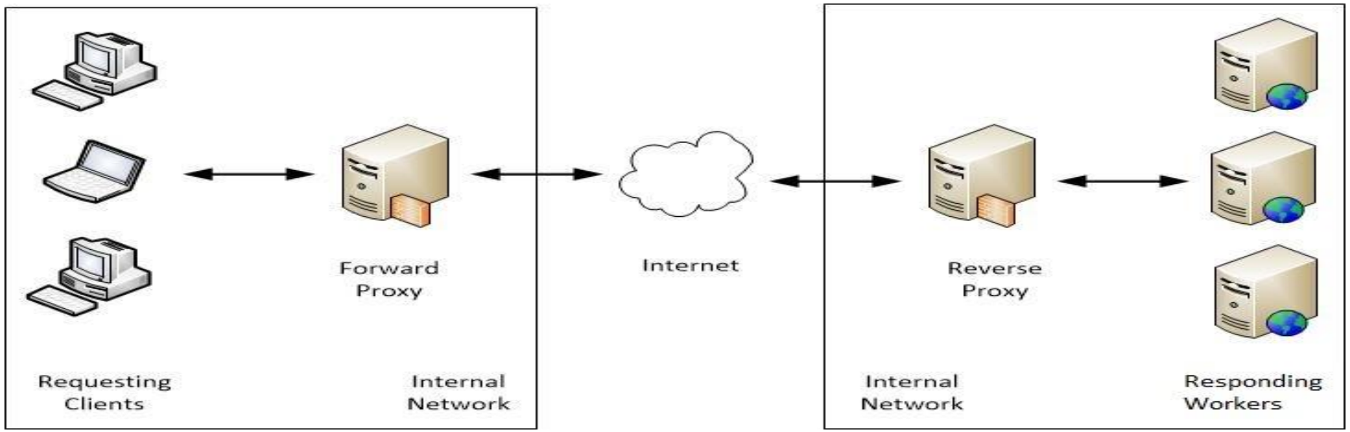
Connector Types

- Client
  - Apache httpclient, Ruby faraday, Python requests,
- Server
  - Java servlets, ruby rack, python WSGI
- Cache
  - Implemented alongside the above. MUST support the same interface not `Map<URL, byte[]>`
  - Shared caching cannot be properly standardized
- Resolver
  - Translated resource identifiers to network addresses: DNS
- Tunnel
  - Relays communication across a connection boundary ex HTTP CONNECT for SSL

Components

- The processing elements of REST
- User agent client connector
  - Initiates a request, gets a response, manipulates resource representation
- Origin Server server connector
  - Handles namespace for resources
  - Definite source of truth for its resources
- Proxy client and server connectors
  - Provides services to clients and hides the network from them
- Reverse Proxy client and server connectors
  - Provides services to origin servers and shields them from the network

Process View



- Shows data path and components interactions
- Dynamic data paths and configurations
- Client/server ➔ scalability, performance tuning, reduced complexity
- Stateless ➔ independent interactions, no need to know the overall topology
- Generic REST interface ➔ pipe and filter style

Connector View

- Integration of other systems. Only REST is supported out of the box
- Mechanics of communication between components
- Constraints of the REST interface when
  - Translating requests, for example which Proxy to choose
  - Communicating with non-REST components for example FTP

Data View

- Application: information and control alternatives
  - Simple state machine: state and transitions
  - Program: memory R/W and control `if, for`
- Application state
  - OO program: objects, their state and messages in transit
  - Unix process: list of open fds, memory dump
- Networked application
  - Information and control via messages
- Networked application state
  - Pending requests, topology of connected components, active requests on connectors, dataflow of representations, processing of representation
  - Steady state when no outstanding requests
  - Latency between steady states is a performance metric

Data View - REST

- Reveal application state as information flows through components
- REST concentrates all the control state into the representations received
  - Control data and control representations in messages
  - Server scalability: no need to maintain awareness of client state
- The next control state is in the representation of the requested resource
- The application state is controlled and stored by the user agent
  - Direct manipulation of state ex. history
  - Anticipate changes in state ex. prefetching
  - Switch applications ex. bookmarks

“Application is an engine that moves from one state to the next by examining and choosing from among the alternative state transitions in the current set of representations”

Data View - HATEOAS

Hypertext As The Engine Of Application State

- Fetch the resource identified by id
- Hypermedia controls - choose one to go to next state
  - `<link rel=”prev” uri=”/page/1”> <link rel=”next” uri=”/page/3”>`
  - `<a href=”more/info”>`
  - `<form submit=”/register”>`
- Discoverability of actions on resources
  - Clients should not build URLs
  - Decouple URL from action
- Independent Evolution, Decoupled Implementation
- Standard link types, Extensions, Documentation

REST in a nutshell

1. Client/Server
2. Stateless
3. Caching
4. Layered Structure
5. Code on Demand
6. Uniform Interface
    a. Identification of resources
    b. Manipulation of resources through representations
    c. Self-Descriptive Messages
    d. Hypermedia as the Engine of Application State

Applied REST

Or was it clear?

Checklists everywhere

1. A plural noun should be used for collection names
2. DELETE must be used to delete a resource from its parent
3. 201 “Created” must be used to indicate successful resource creation
4. Caching should be encouraged
5. A consistent form should be used to represent links
6. New URIs should be used to introduce new concepts

502. ………

⇒ “There is no REST standard or REST reference implementation”

s/REST/CRUD/g

```
>./framework --rest --new-endpoint --name foo

Created FooController with endpoints:
GET /foos
POST /foos
GET /foos/:id
PUT /foos/:id
DELETE /foos/:id
```

⇒ “But no tool or even guidelines on how to write the remaining code”

Richardson Maturity Model

1. Use HTTP for remote interactions
   a. Free to do anything you like with a single URL
2. Use resources
   a. Create some URLs and still free to do anything with URLs
3. Use HTTP verbs
   a. GET, POST, PUT, DELETE at the URLs
4. Hypermedia Controls
   a. `<link rel=”next” uri=”/54”>`

⇒ “Closer but REST is independent of HTTP”

Fielding clarifies

REST APIs must be hypertext-driven: A 2008 blog post by Roy Fielding

“If the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period.”

“A REST API must not define fixed resource names or hierarchies”

“A REST API should be entered with no prior knowledge beyond the initial URI and a set of standardized media types”

⇒ Avoid client-server coupling, avoid out-of-band data, client drives via hypertext

Nevertheless REST suits the Web

REST applied to URI

- Redefinition of resource
  - Concepts not documents
- Manipulating Shadows
  - Representation are not the actual data
- Remote Authoring
  - Semantics of storage
  - The writable web is still inconsistent with the readable web
- Binding semantics to URI
  - URI must have semantics not structure
- REST mismatches in URI
  - Syntax does not enforce semantics
  - The web is not a file system

REST applied to HTTP

- Extensibility
  - Protocol versioning HTTP/1.0 HTTP/1.1 HTTP/2.0
  - Extensible protocol elements return codes, methods, semantics
- Self-Descriptive Messages
  - New Headers: `Host`, `Transfer-Encoding`, `Content-Encoding`
  - Chunked transfer, size limits, cache control, content negotiation
- Performance
  - Persistent connections
  - Write-through caching

REST mismatches in HTTP Extensions

- Differentiating non authoritative responses
  - Is the response from the origin server?
  - `Warning, Via, no-cache` headers failed to solve this
- Cookies
  - A well-known mechanism to have state between client and server
  - Cause problems because they
    - Don’t have semantics
    - They are associated with URIs not particular application state
  - A better solution was to use full HATEOAS
- Header extensions ”`X-Whatever`” do not obey REST semantics

Reflections

- @2002 very optimistic for the foundations of REST and the future of the web
- @2017 Reflections on the REST Architectural Style
  - Paper
  - YouTube

Resources

- Hyperland by Douglas Adams
  - A short film about hypermedia produced before the web and how far we are
- Hypermedia APIs by Jon Moore
  - A talk about HATEOAS
- REST APIs must be hypertext driven by Roy Fielding
  - A must read blog post about REST and RESTful
- Your Service is not REST by Kostis Kapelonis
  - A talk about REST compliance

Almost 20 years later

REST today

- Nice to have but not a priority
  - Open source clients and cloud services prevail
- RPC returned
  - gRPC
- Alternative Web API specs
  - json:api
  - Siren
  - HAL
- Beyond REST
  - GraphQL
  - Introspected REST

The world wide web today

REST had only technical issues to face. Today **NONE** of the issues is technical.

- WORLD WIDE WEB
  - Success: “Attract large crowds, don’t care for diversity, ignore connectedness”
  - A must read post The web we have to save
- Centralization
  - No control of personal data, silos
- Digital Divide
  - Access to the web, net neutrality
- Privacy
  - Trackers have gone too far. Personal data are sold in market.
- Trust
  - misinformation - weaponized web

```
<a rel=”questions” href=”/now”>
```

```
HTTP/1.1 200 OK
Connection: close

<h1>Thank You</h1>
```