Deriving REST: Constraints

Howard Dierking
http://codebetter.com/howarddierking

Overview

- Getting to REST
- Client-Server
- Stateless
- Cache
- Uniform Interface
- Layered System
- Code On-Demand

Getting to REST

- Fielding described 2 approaches to defining an architectural design
- REST is defined by identifying the forces that influence system behavior and then applying constraints so that the design works with those forces

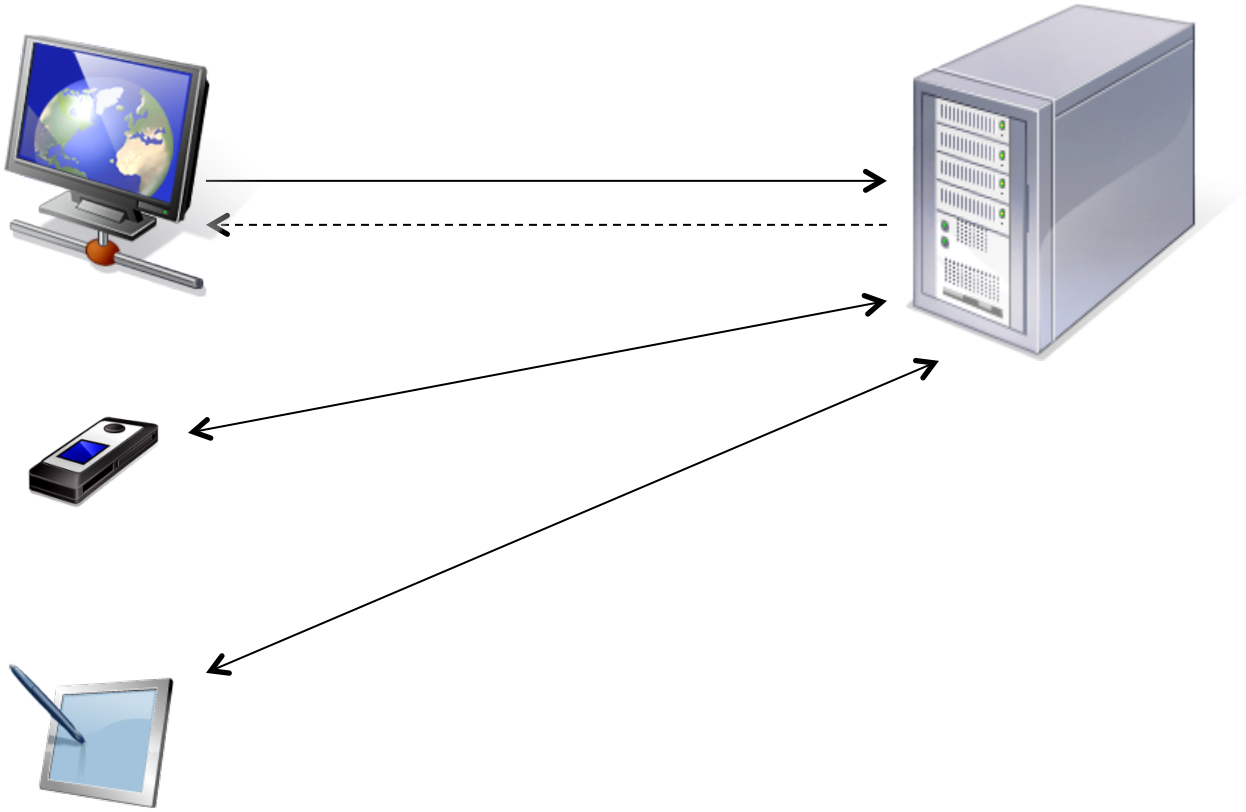
Requirements-Driven

Constraints-Driven

The “Forces”

- Network reliability
- Latency
- Bandwidth
- Security
- Network topology
- Administration
- Transport cost
- Heterogeneous network
- Complexity

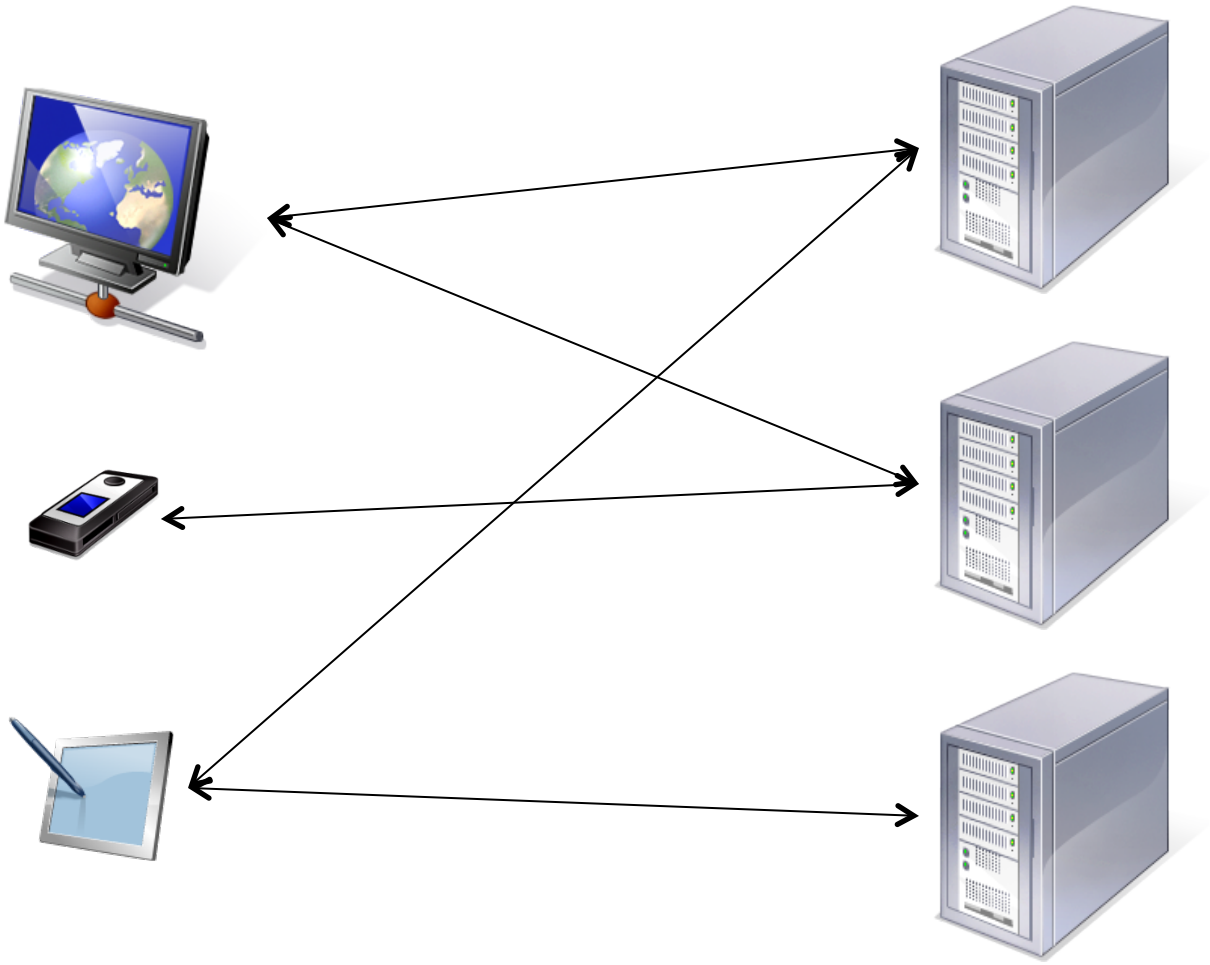
Client-Server



Concerns Addressed

- **Influencing Forces**
  - Security – network security concerns can be scoped to the connections between clients and servers
  - Administration – administration can be scoped to the connections between clients and servers
  - Heterogeneous network – multiple clients of multiple platform types can connect and disconnect from the network without impacting system state on the server
  - Complexity – clients know about servers – not each other
- **Benefits**
  - Portability of clients
  - Scalability
  - Evolvability

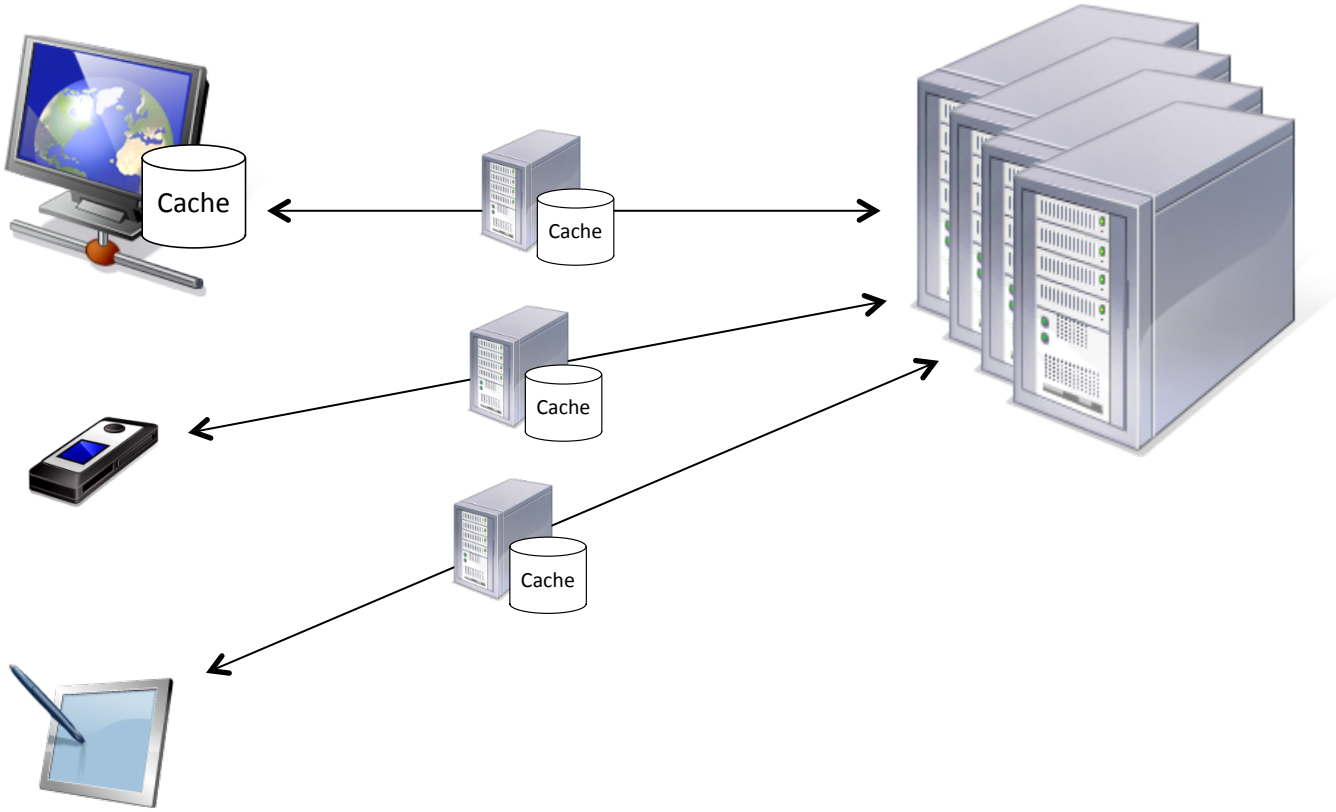
Stateless



Concerns Addressed

- **Influencing Forces**
  - Network reliability – storing state on the client and stateless communications to the server enable clients to recover from network errors
  - Network topology – clients and servers can come and go on the network without corrupting system state
  - Complexity – new processing nodes can be attached without complex state management/replication schemes
  - Administration – Visibility is improved with stateless client/server interactions, simplifying management
- **Benefits**
  - Visibility
  - Reliability
  - Scalability

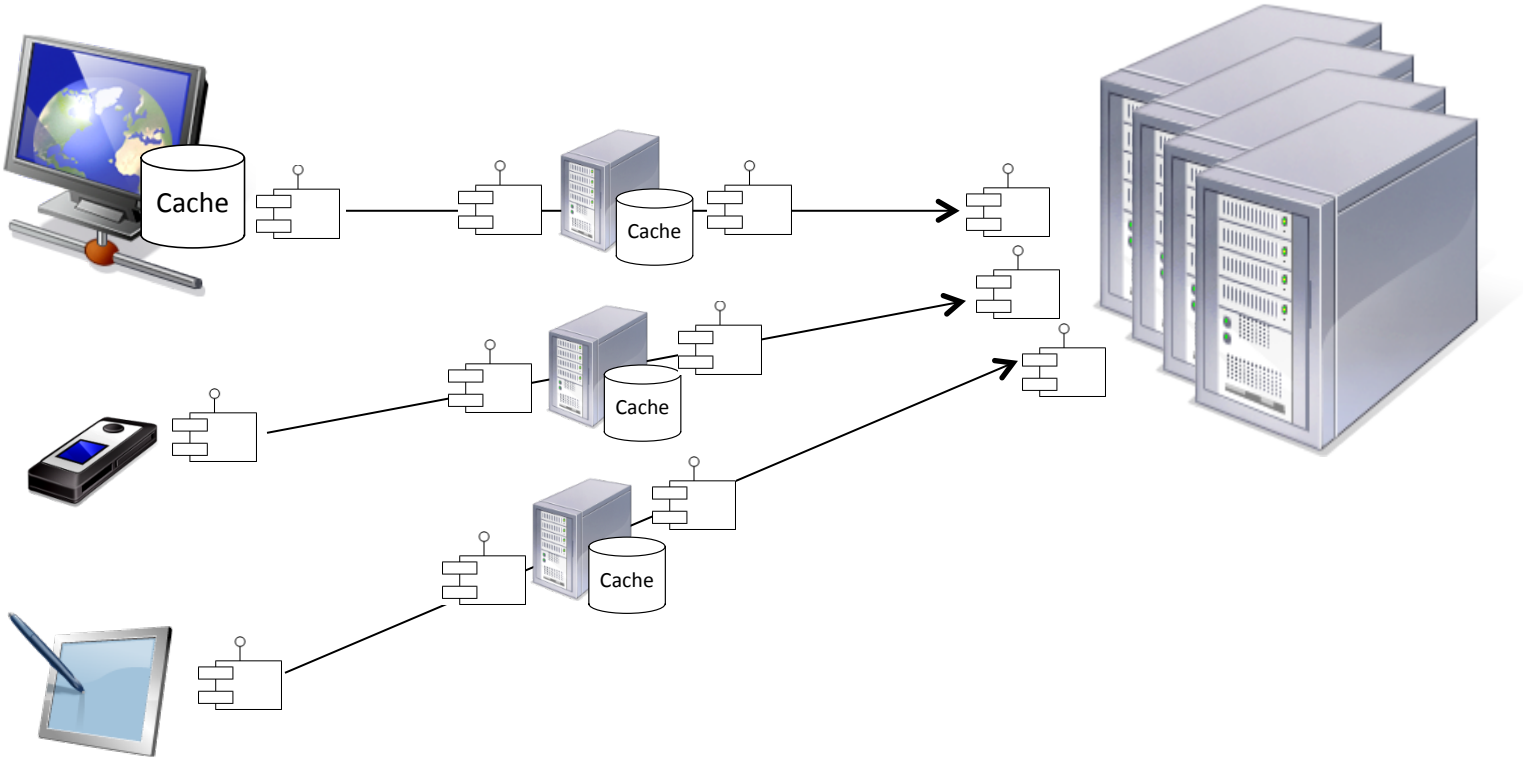
Cache



Concerns Addressed

- **Influencing Forces**
  - Latency – caching can dramatically reduce latency by eliminating the need to make some requests
  - Bandwidth – local caching reduces the amount of data that needs to be consumed by the client while intermediary caching reducing the amount of data needed from the origin server
  - Transport cost – caching reduces the total number of network requests needed
- **Benefits**
  - Efficiency
  - Scalability
  - Performance

Uniform Interface



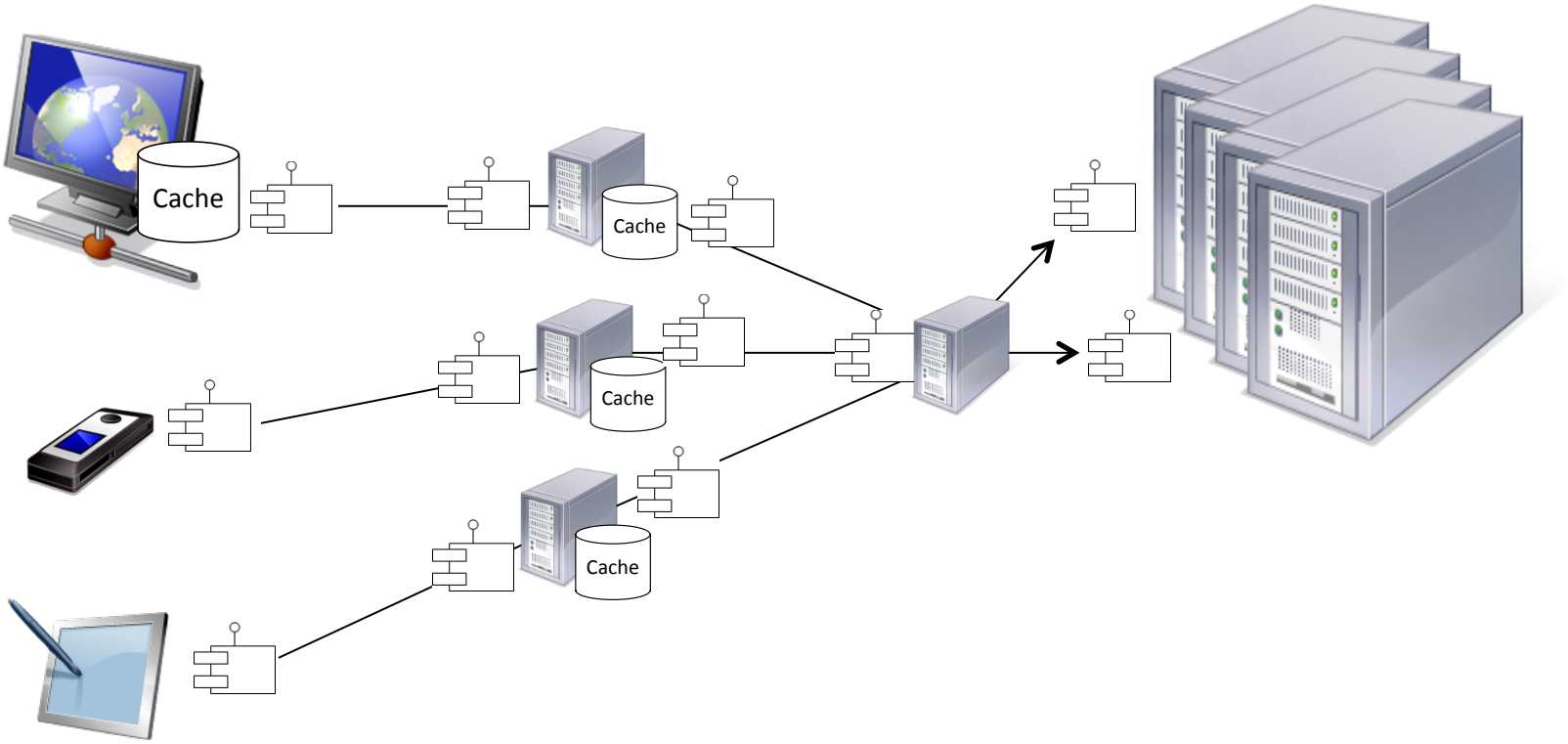
Elements of the Uniform Interface

- Identification of resources
- Manipulation through representations
- Self-descriptive messages
- Hypermedia as the engine of application state (HATEOAS)

Concerns Addressed

- **Influencing Forces**
  - Network reliability – consistent semantics increases transparency, enabling clients and servers to more reliably handle failures
  - Network topology – providing a unified set of constraints governing how clients and servers communicate enables the system elements (including intermediaries) to be created and evolve independently
  - Administration – general management tools can be introduced for optimizing the network
  - Heterogeneous network – different client server platforms can easily interoperate
  - Complexity – the complexity of writing a networked application is constrained to the complexity of the uniform interface
- **Benefits**
  - Visibility
  - Evolvability

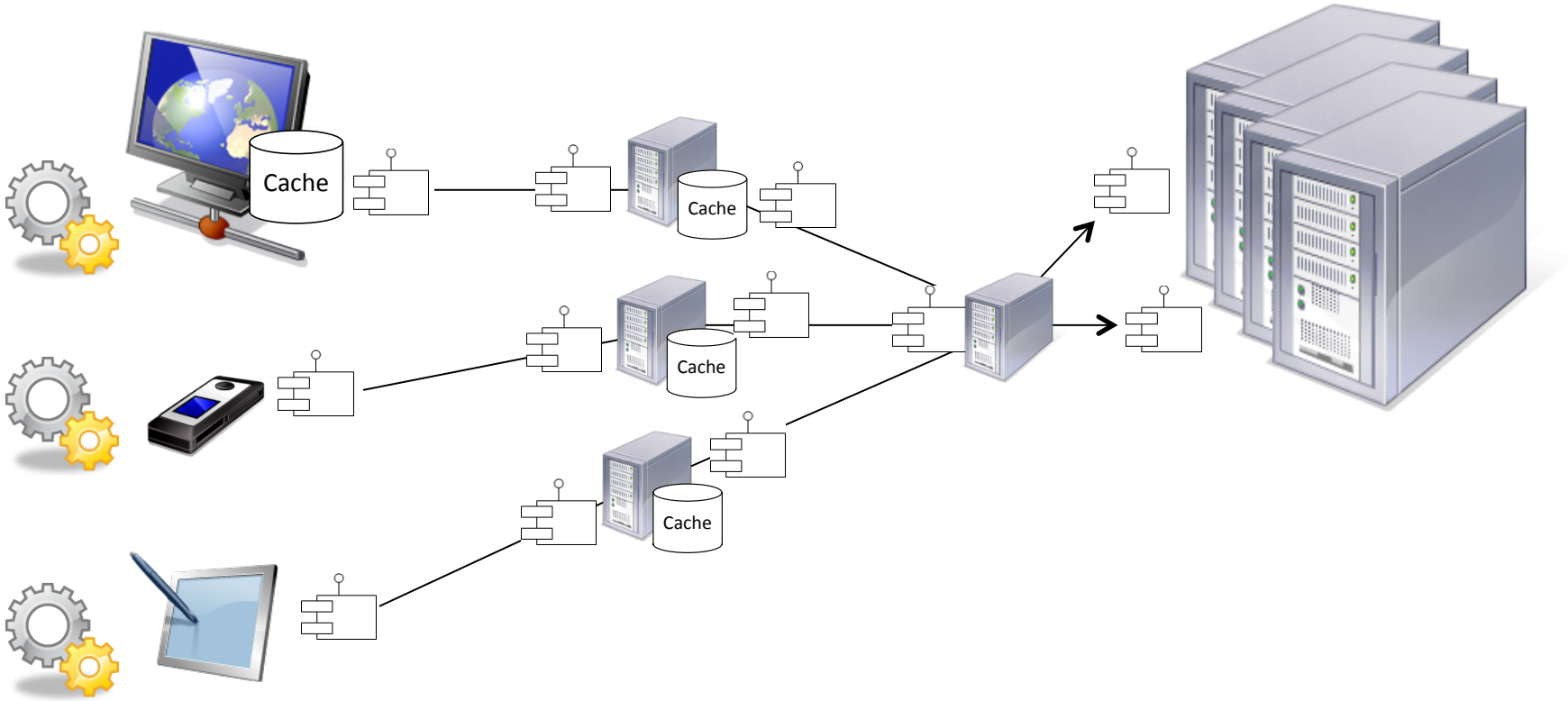
Layered System



Concerns Addressed

- **Influencing Forces**
  - Network topology – Changing network components only impact those elements directly below.
  - Complexity – Limiting the number of components that a component can interact with limits the amount of complexity you can introduce
  - Security – Layering enables intermediaries to be placed at trust boundaries to manage security policies for all components inside the boundary
- **Benefits**
  - Scale
  - Manageability

Code-On-Demand



Code-On-Demand

- **Initially described in terms of Java applets - becoming more relevant with JavaScript**
- **Optional constraint**
  - Helps to manage complexity
  - The trade-off is visibility
- **The key take away with an “optional” constraint here is that if you implement a code-on-demand solution, it should not be required for clients to make progress through the system**

Summary

- Getting to REST
- Client-Server
- Stateless
- Cache
- Uniform Interface
- Layered System
- Code On-Demand

- **Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.**
- **Rotem-Gal-Oz. Fallacies of Distributed Computing Explained.**