REST API

Umme Habiba.

KTH-AIS LAB

Dated by : February 14, 2013

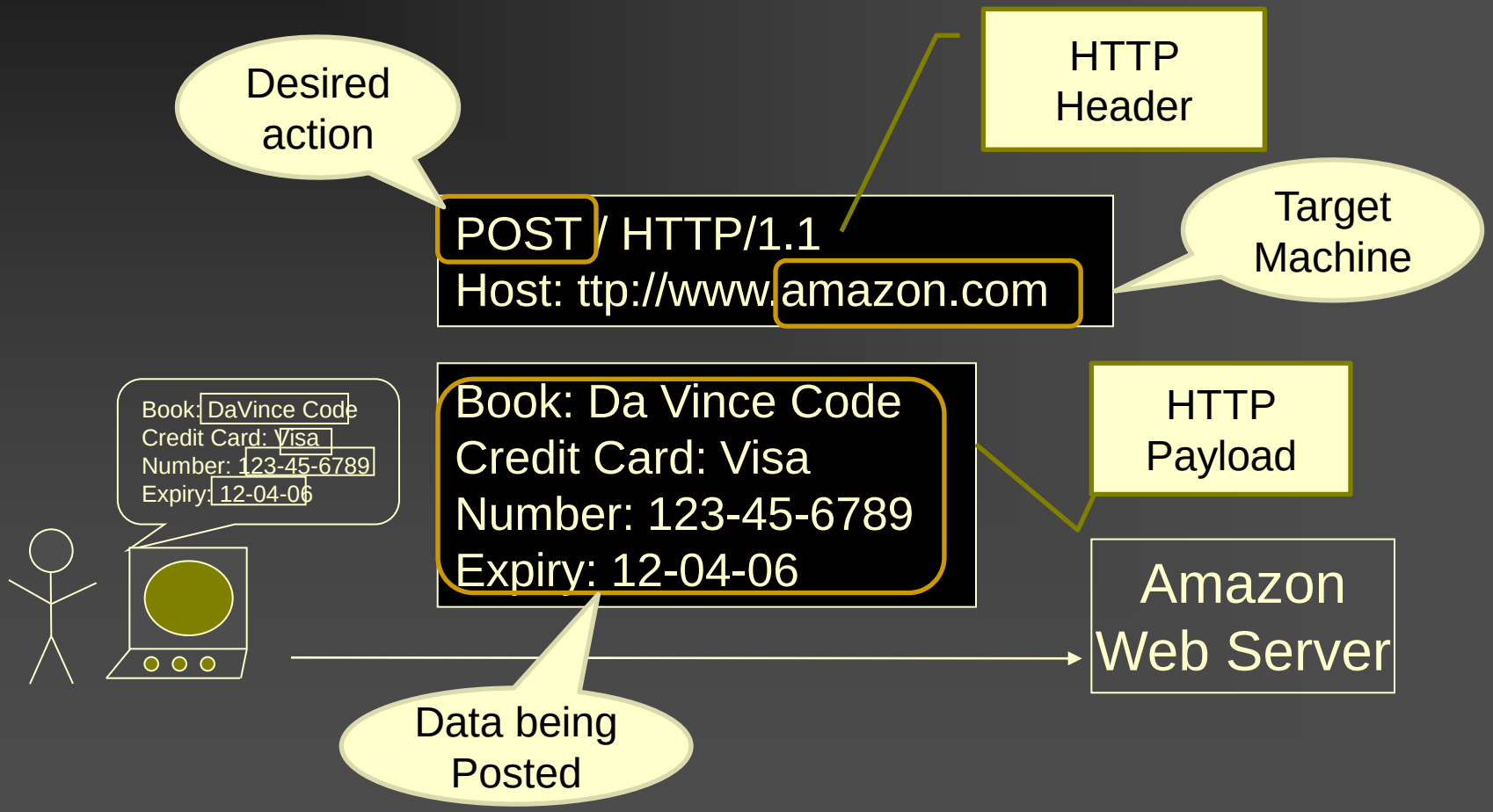
Web Basics: Operations, via the HTTP API

- HTTP provides a simple set of operations. Amazingly, all Web exchanges are done using this simple HTTP API:
  - GET
    - Properties: Safe, Idempotent
    - Usage: Retrieving a resource
  - POST
    - Properties: UNSAFE
    - Creating a resource within a collection (resource URI unknown)
  - PUT
    - Properties: Idempotent
    - Usage: Creating or updating a resource at a known URI
  - DELETE
    - Properties: Idempotent
    - Usage: Deleting a resource

ROA?

- ROA is the term for REST on HTTP/URI
- A Service consists of all the resources available within a certain domain of control
- Since REST is a type of SOA, ROA is an implementation of SOA as well.

Web Basics: Simple Set of Operations, via the HTTP API



REST

Roy Fielding described REST as an architecture style which attempts "to minimize latency and network communication, while at the same time maximizing the independence and scalability of component implementations"

Roy T Fielding. REST Dissertation at http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST - Not a Standard

- REST is not a standard
- REST is just a **design pattern**
- REST does prescribe the **use** of standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/*etc.* **(Resource Representations)**
  - text/xml, text/html, image/gif, image/jpeg, *etc.* (Resource Types, MIME Types)

REST API

Why is it called "Representational State Transfer? "

The Client references a Web resource using a URL.
A **representation** of the resource is returned (in this case as an HTML document).
The representation (*e.g.*, Boeing747.html) places the client in a new **state**.
When the client selects a hyperlink in Boeing747.html, it accesses another resource.
The new representation places the client application into yet another state.
Thus, the client application **transfers** state with each resource representation.

REST Constraints

- Important ‘things’ (Noun) are Resources
  - Addressed through a URI
- Uniform interface (Verb)
  - In HTTP: GET, PUT, POST, DELETE
- Verb-Noun separation makes integration easier
  - GET /customer/45

  Instead of getCustomer(45) OR viewCustomer(45) OR showCustomer(45)

REST

- Create a resource for every service.
- Separation of resource from representation
- The data that a Web service returns should link to other data.
- Resources are identified by URIs
- Resources are manipulated through their representations
- Self-descriptive messages

Why not plain HTML?

- Web pages are designed to be understood by people,
  - layout and styling, not just raw data
- Every URI could have a human-readable and a machine-process-able representation:
  - Web Services clients ask for the machine-readable one
  - Browsers ask for the human-readable one
- A web page is a *representation* of a resource
- URIs tell a client that there's a concept somewhere
- Clients can then request a specific representation of the concept from the representations the server makes available

Why hypertext?

- Because the links mirror the structure of how a user makes progress through an application
- The user is in control, thanks to the Back button and other non-local actions
- In a Web service, the client should be in control in the same sense

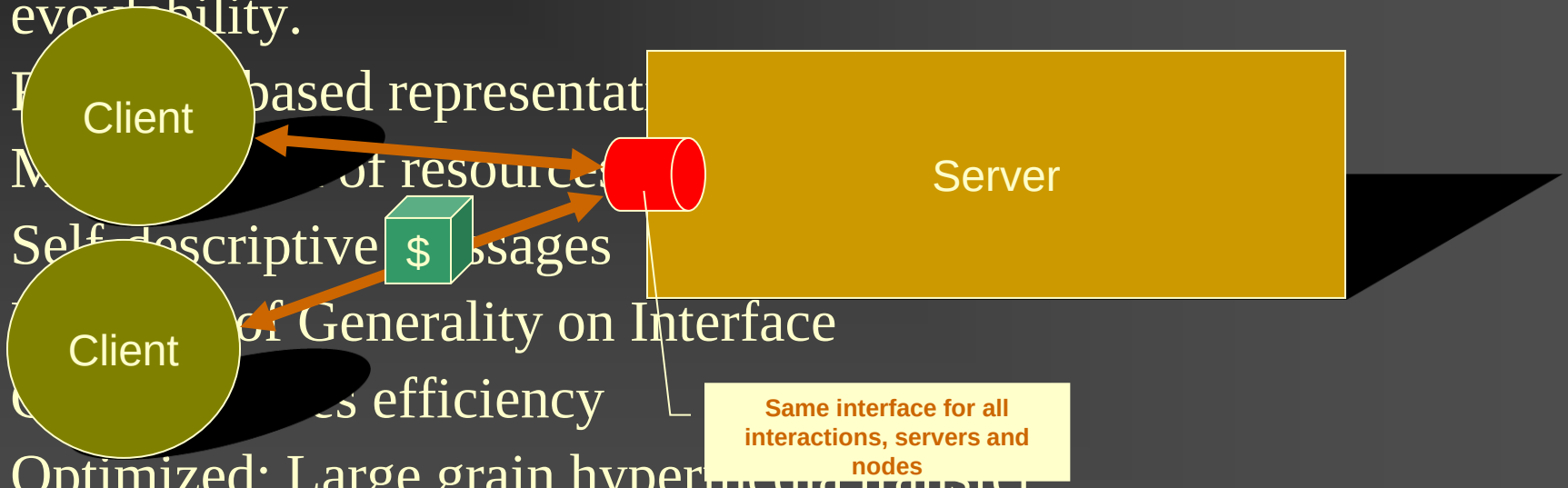
```
<order self='http://example.com/customers/1234' >
    <amount>23</amount>
    <product ref='http://example.com/products/4554' />
    <customer ref='http://example.com/customers/1234' />
</order>
```

What is REST??

- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (Optional)

Uniform Interface

- Simplifies & decouples Architecture for better visibility & evolvability.
- [illegible]based representat[illegible]
- M[illegible] of resource[illegible]
- Self-descriptive [illegible]ssages
- [illegible] of Generality on Interface
- [illegible]s efficiency
- Optimized: Large grain hypermedia transfer

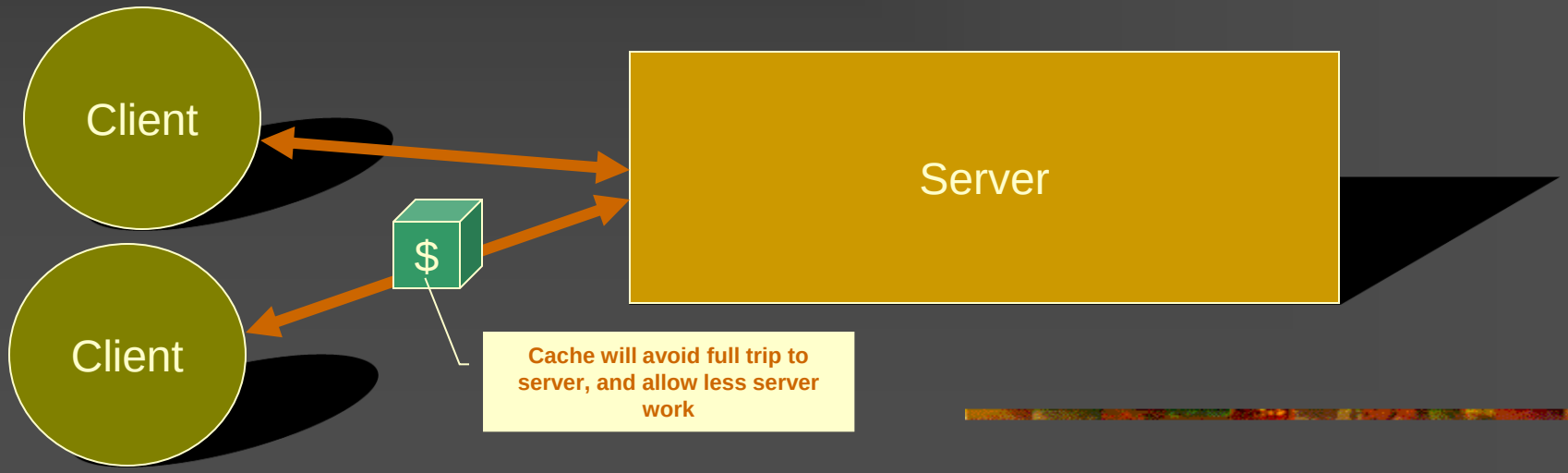


Client Server

- Separation of Concerns
- Improve Portability of UI
- Scalability per simple server components
- Independent evolution

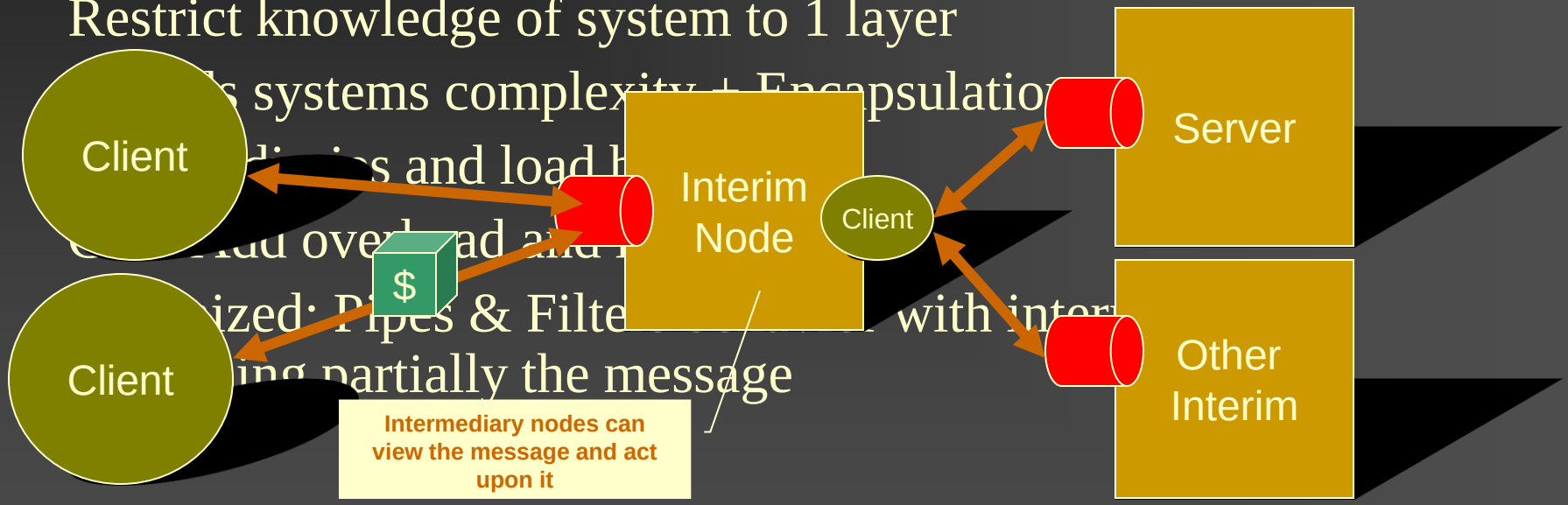
Cache

- Eliminates Client server Interactions, partially or completely
- Improves Scalability and performance
- Reduced latency in average
- Con: Decrease on reliability, cached data may not be the recently updated



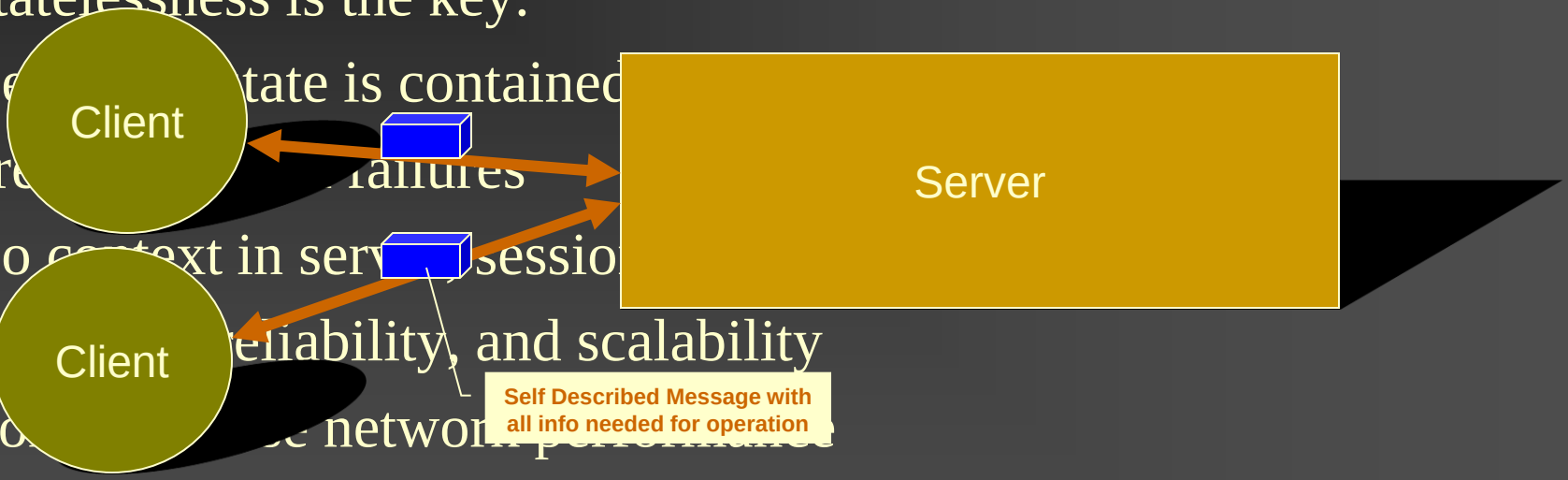
Layered System

- Restrict knowledge of system to 1 layer
- s systems complexity + Encapsulatio
- s and load
- dd ove ad a
- ized: Pipes & Filte with inter
- ing partially the message



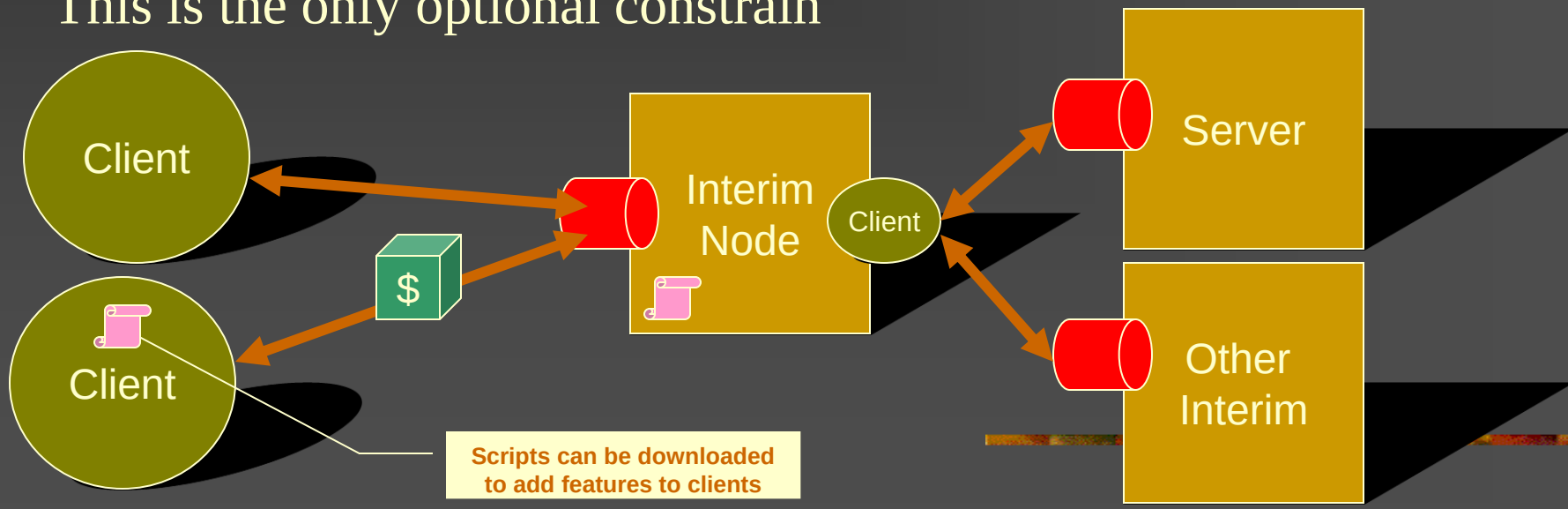
Stateless

- Statelessness is the key.
- Ne tate is contained
- Pr ailures
- No context in serv sessio
- V eliability, and scalability
- Co e networ

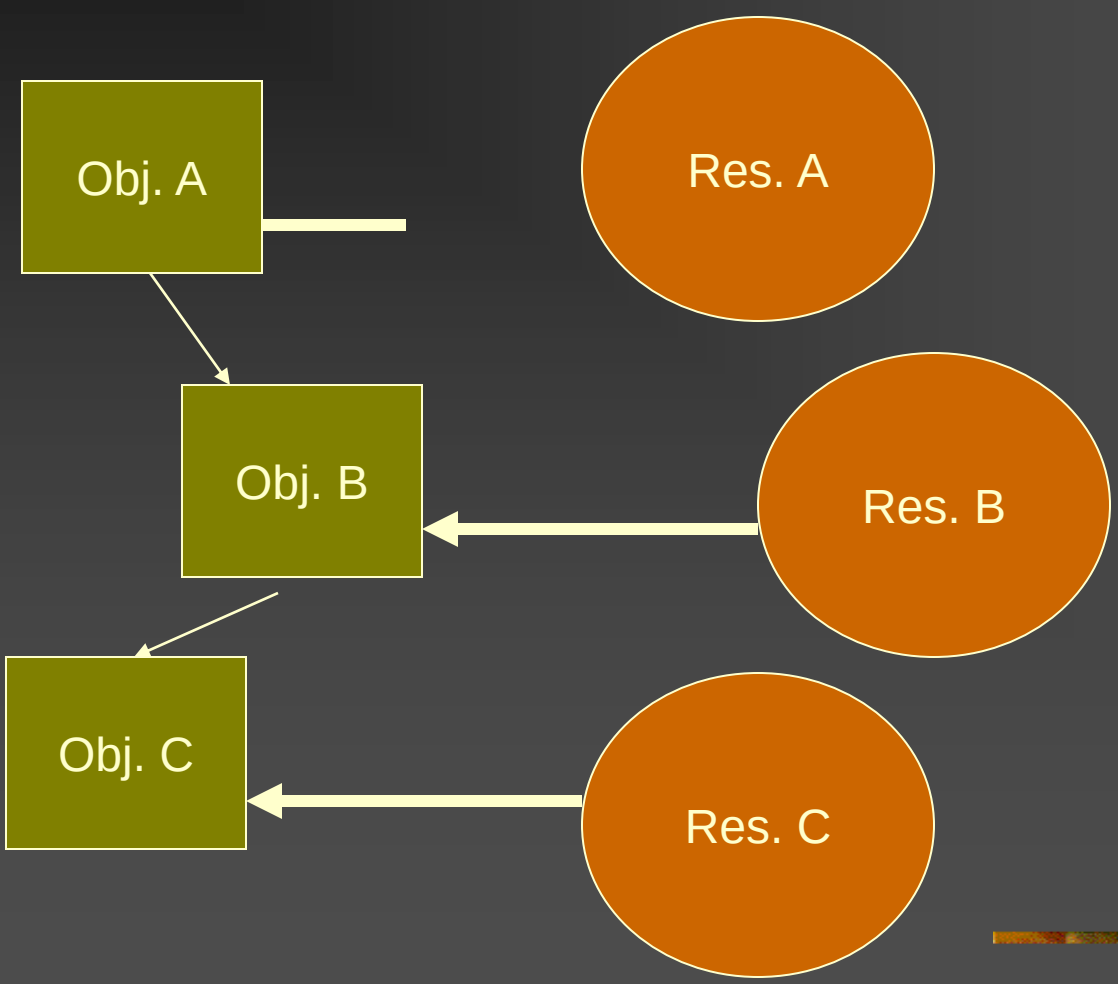


Code On Demand (Optional)

- Temporary extend or customize client functionality (extensibility- java scripts and java applets)
- Client simplification
- Con: Reduces visibility
- This is the only optional constrain



REST – Example



Example - Cont..

- Mapping
  - Each object in the domain is one resource
  - There may be the need for additional resources
  - HTTP operations to manipulate known resources
  - Client to know what and how to post for processing.
- Problems
  - This approach exposes model domain
  - There is not encapsulation, nor data hiding
  - Resources are passive
  - Not using HATEOAS

REST – Alternative Example

- Use Services as resources
- GET to bank/services resource returns an list of request forms for consuming services
- Follow the links to forms, fill the forms and post.
- Post the form to a bank/cashier resource
- bank/cashier is a service! And a resource!
- Post a Loan Request form and get a Loan Request ID

REST – Example Goodies

- This approach is business domain oriented
- Encapsulates processing
- Resources are active.
- Follows HATEOAS constrain.

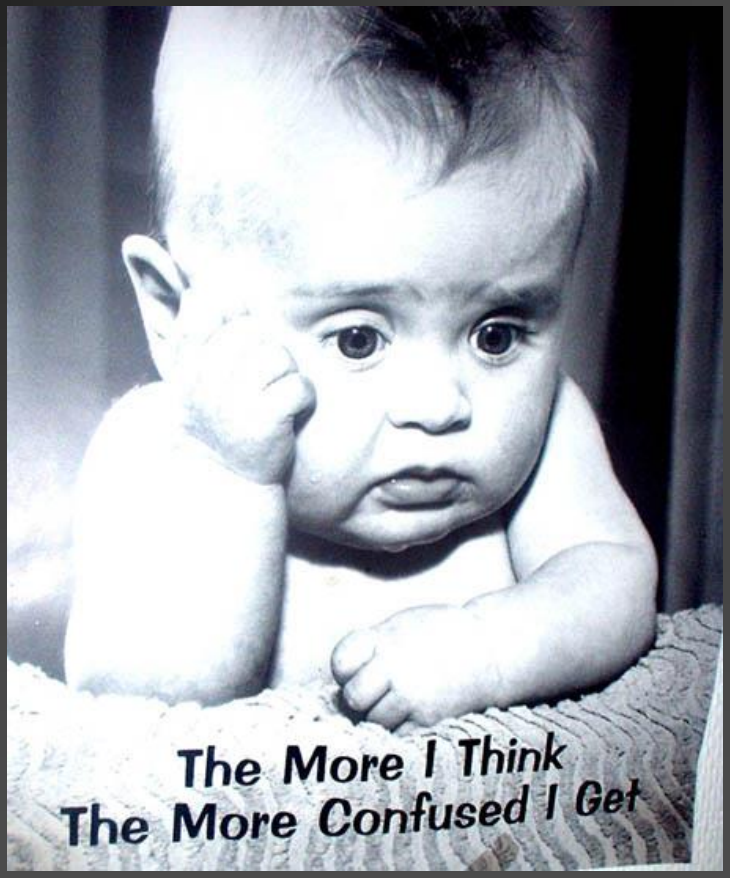
Who is using REST?

- Google
  - GData, OpenSocial
- Standards
  - Atom, WebDAV
- Amazon
  - S3, SimpleDB
- Microsoft (!)
  - Project Astoria, Web3S

Advantages of REST

- Its architectural constraints *when applied as a whole*, generate:
  - Scalable component interactions
  - General interfaces
  - Independently deployed connectors
  - Reduced interaction latency
  - Strengthened security
  - Safe encapsulation of legacy systems
- Separates server implementation from the client's perception of resources
- Scales well to large numbers of clients
- Enables transfer of data in streams of unlimited size and type

Thank You


The More I Think
The More Confused I Get