REST

And now for something completely different…

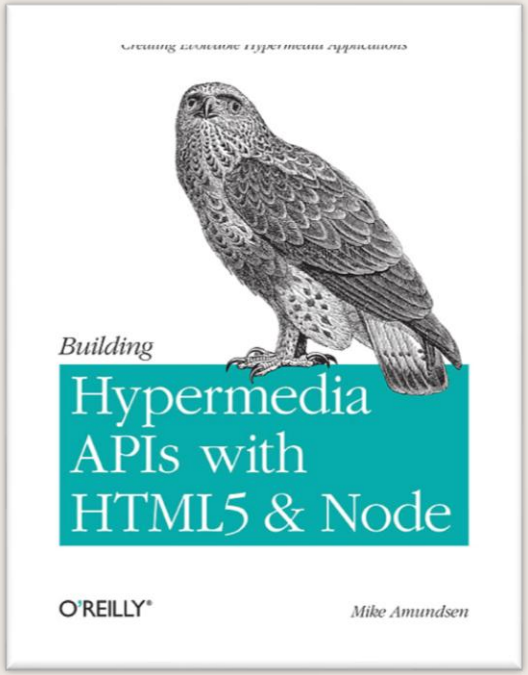
Mike Amundsen
@mamund
amundsen.com

Preliminaries

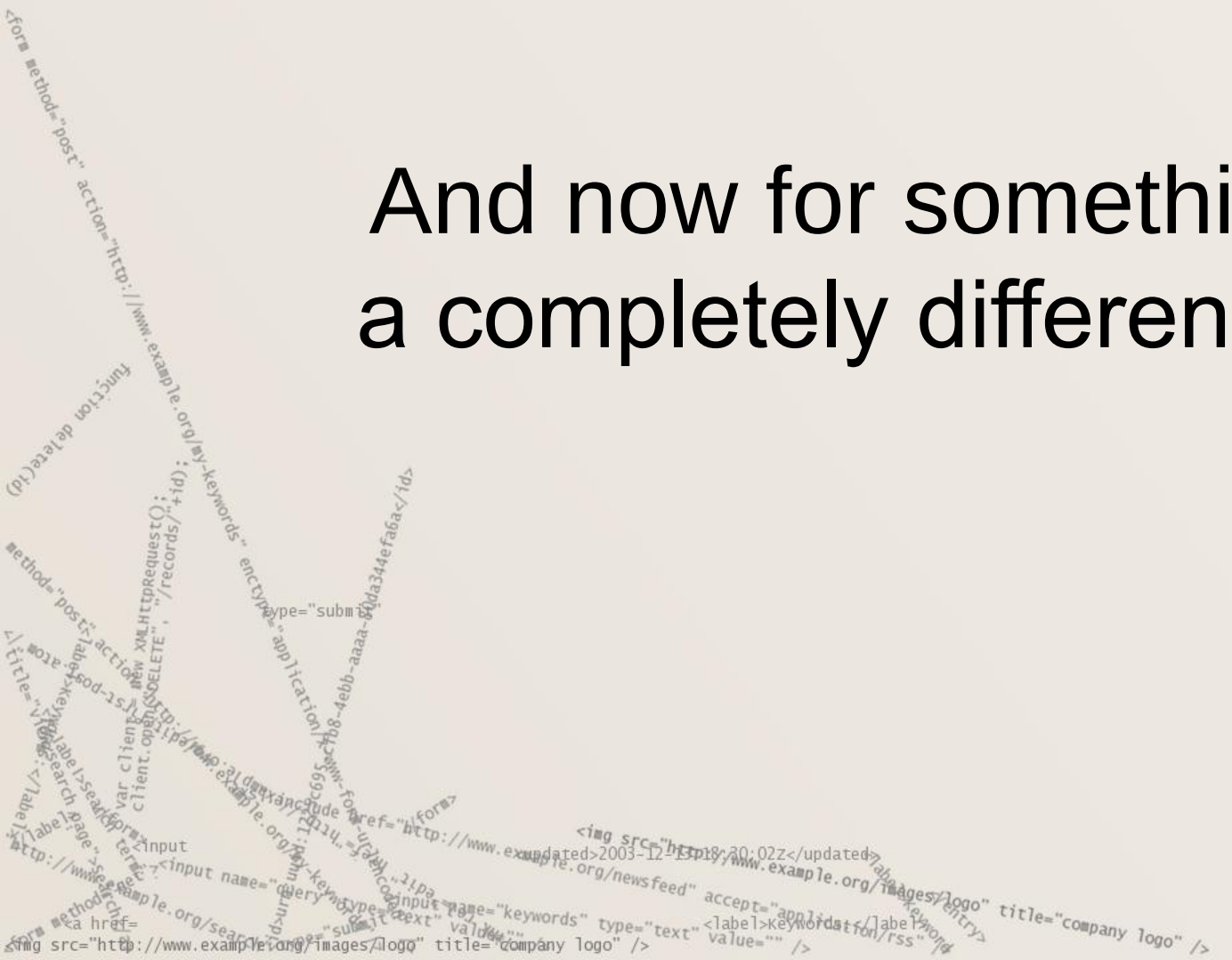
- Mike Amundsen
- Developer, Architect, Presenter
- Hypermedia Junkie

- “I program the Internet”

- ***Designing Hypermedia APIs with Node and HTML5 O’Reilly, Fall 2011***



And now for something a completely different…



“Excuse me ...
did you say ‘knives'?”

City Gent #1 (Michael Palin)
The Architects Sketch

“Consider how often we see software projects begin with adoption of the latest fad in architectural design, and only later discover whether or not the system requirements call for such an architecture”

Roy T. Fielding, 2000

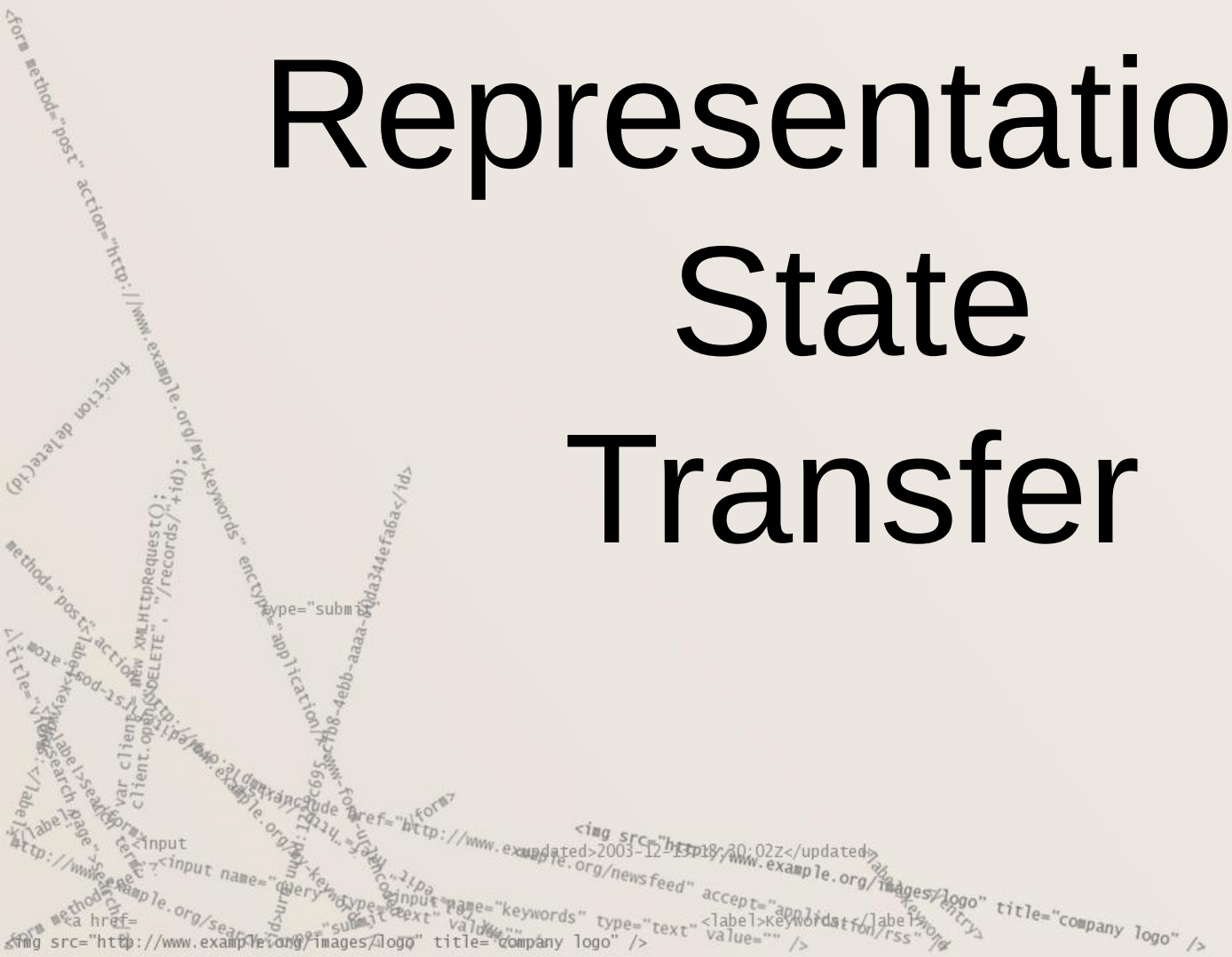
"Consider how often we see software projects begin with adoption of the latest fad in architectural design, and only later discover whether or not the system requirements call for such an architecture"

Roy T. Fielding, 2000

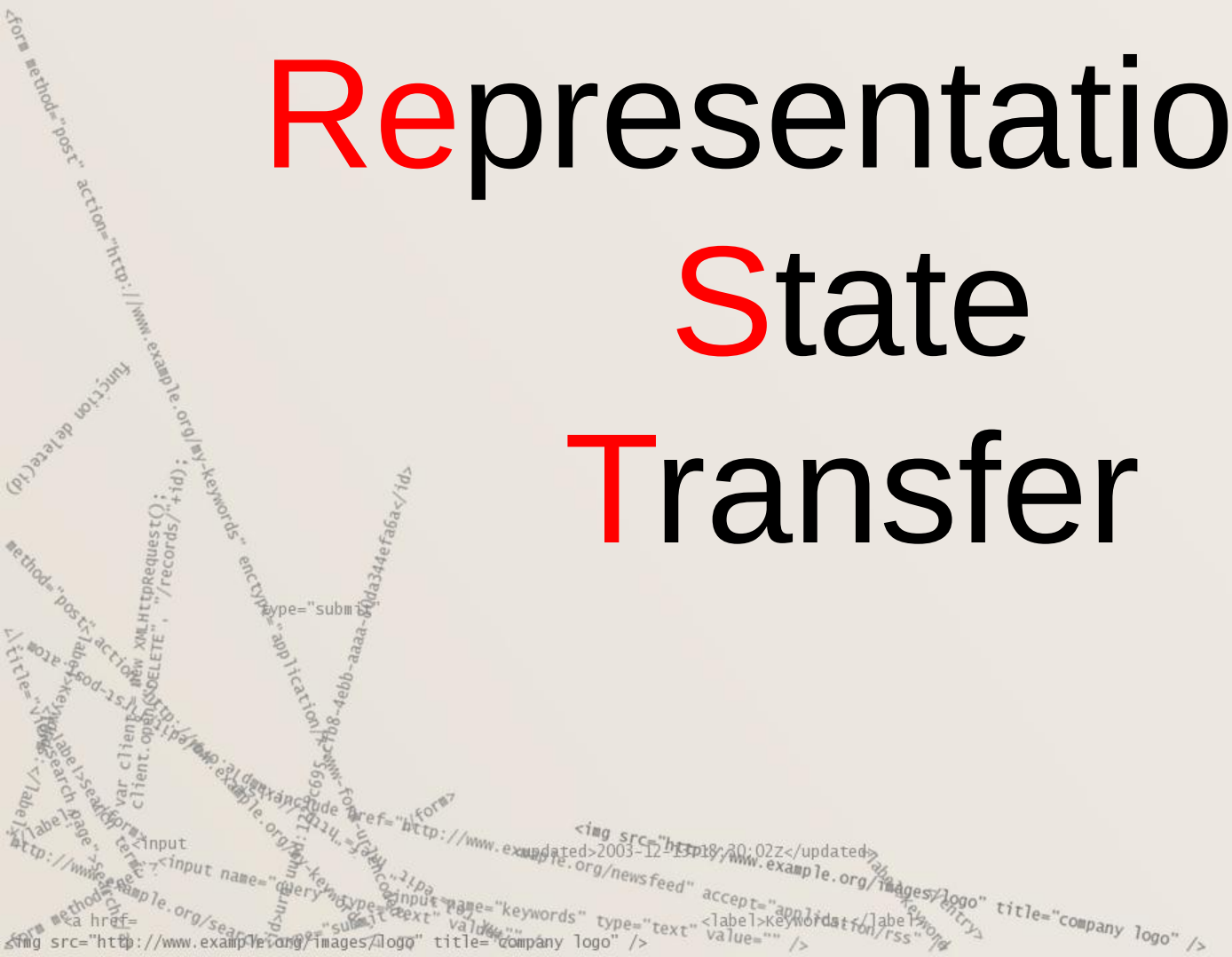
“Consider how often we see software projects begin with adoption of the latest fad in architectural design, and only later discover whether or not the system requirements call for such an architecture”

Roy T. Fielding, 2000

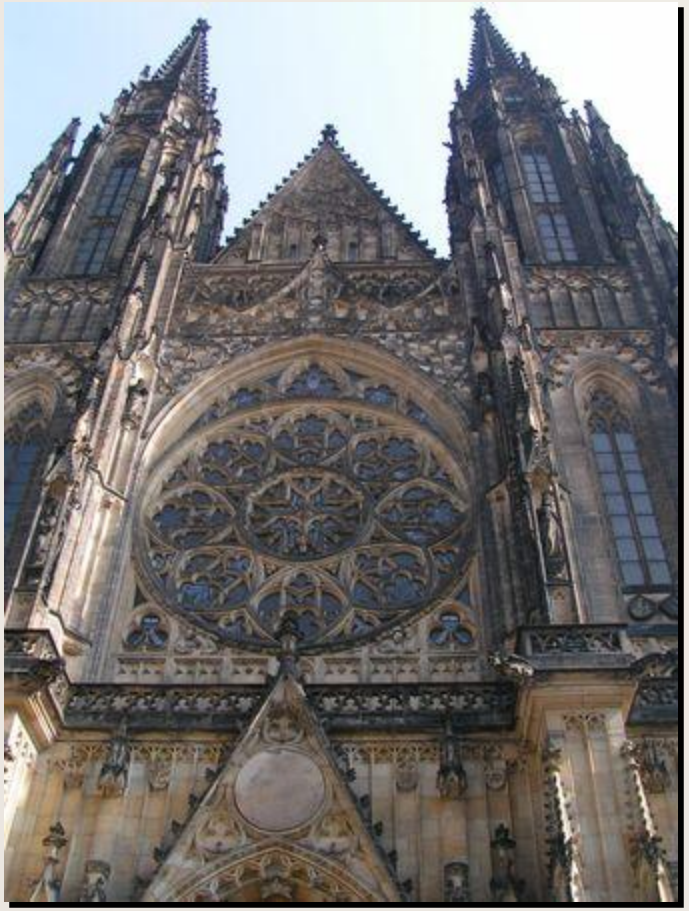
Representational State Transfer

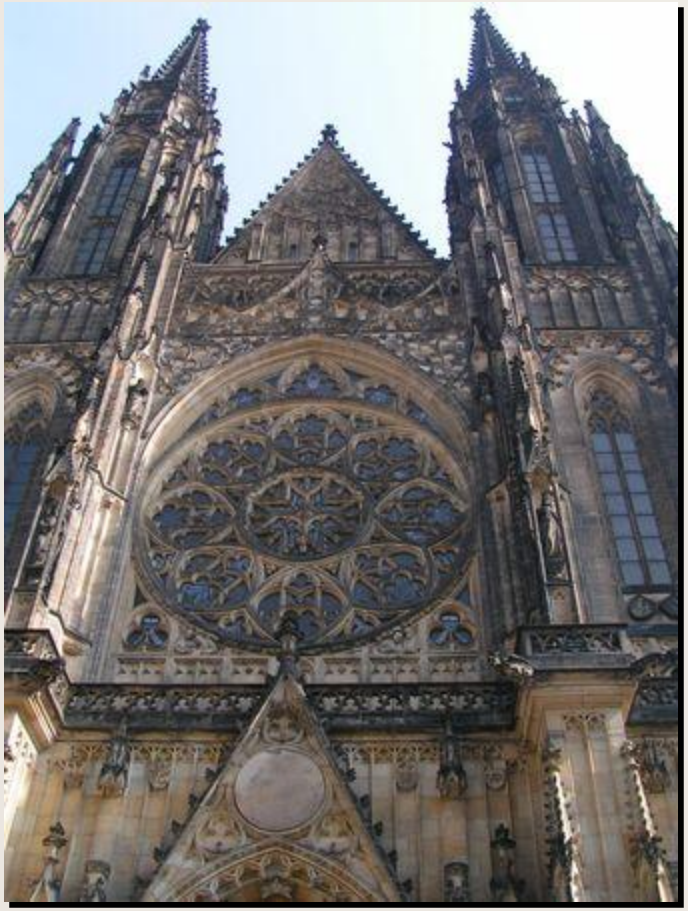


Representational State Transfer

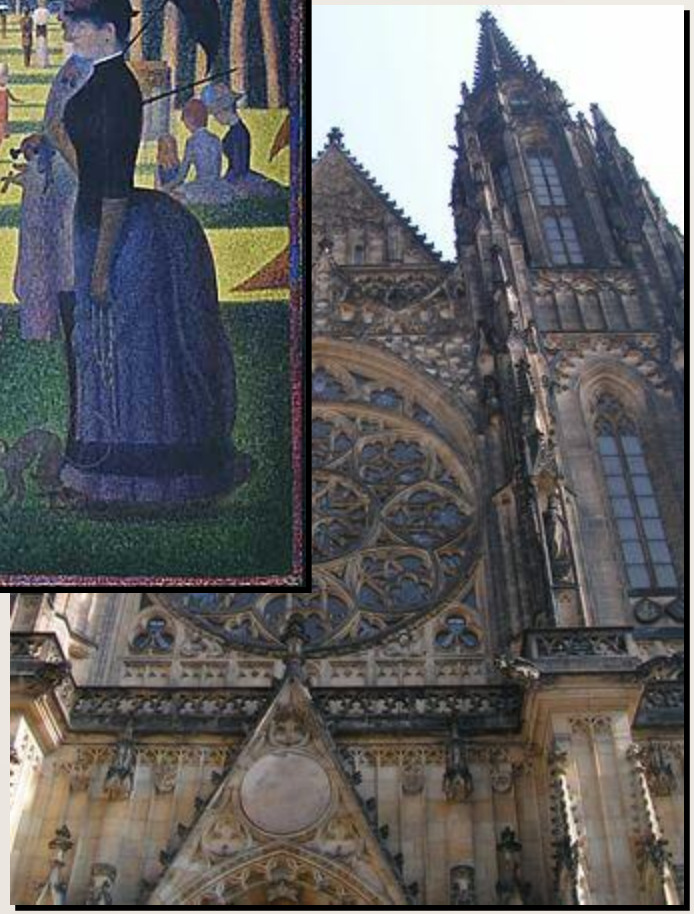


These are....?





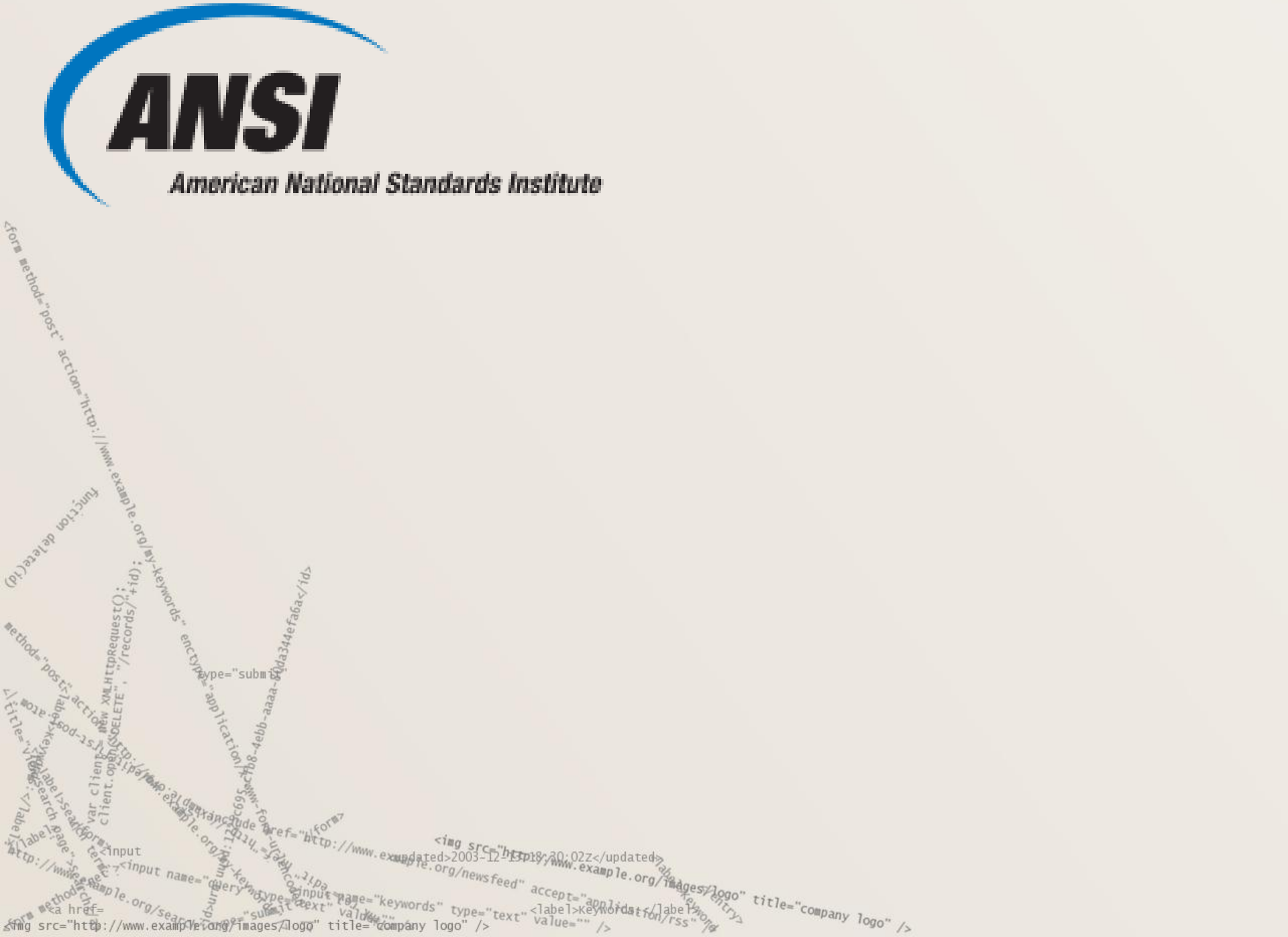

AIC



REST
is a
style

REST is a style of software architecture.

In contrast to…

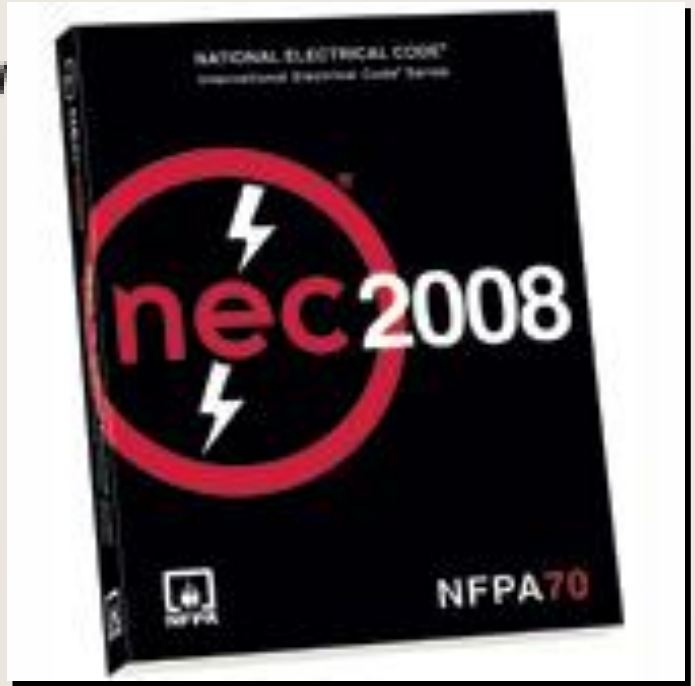

ANSI
American National Standards Institute


ANSI
American National Standards Institute


NIST

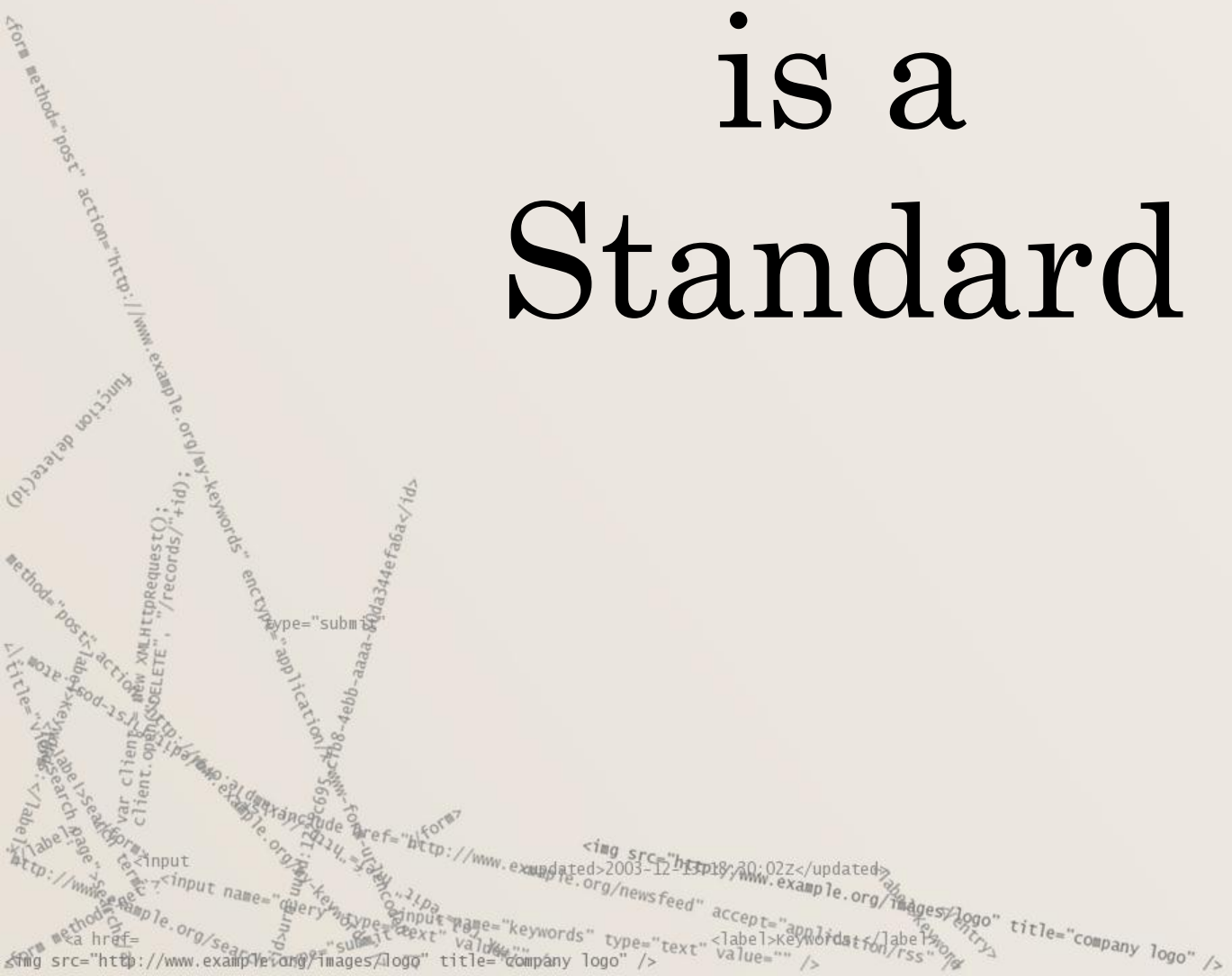
ANSI

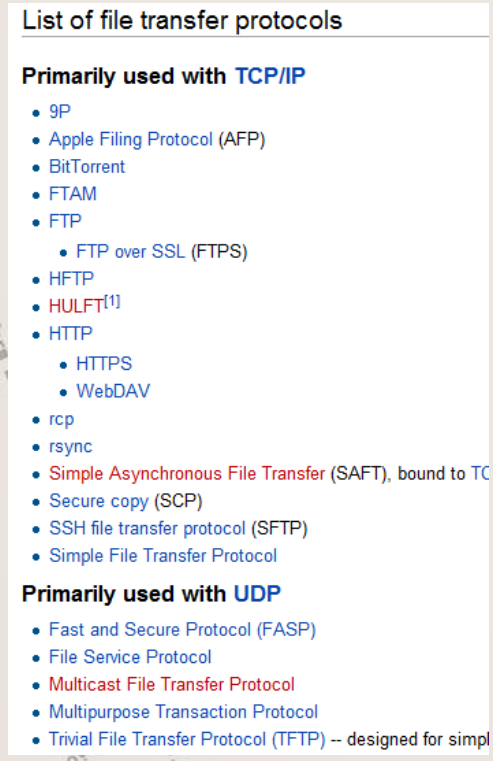
American National



NIST

HTTP
is a
Standard

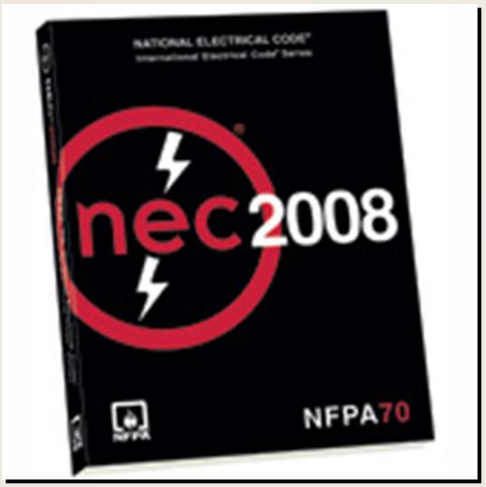




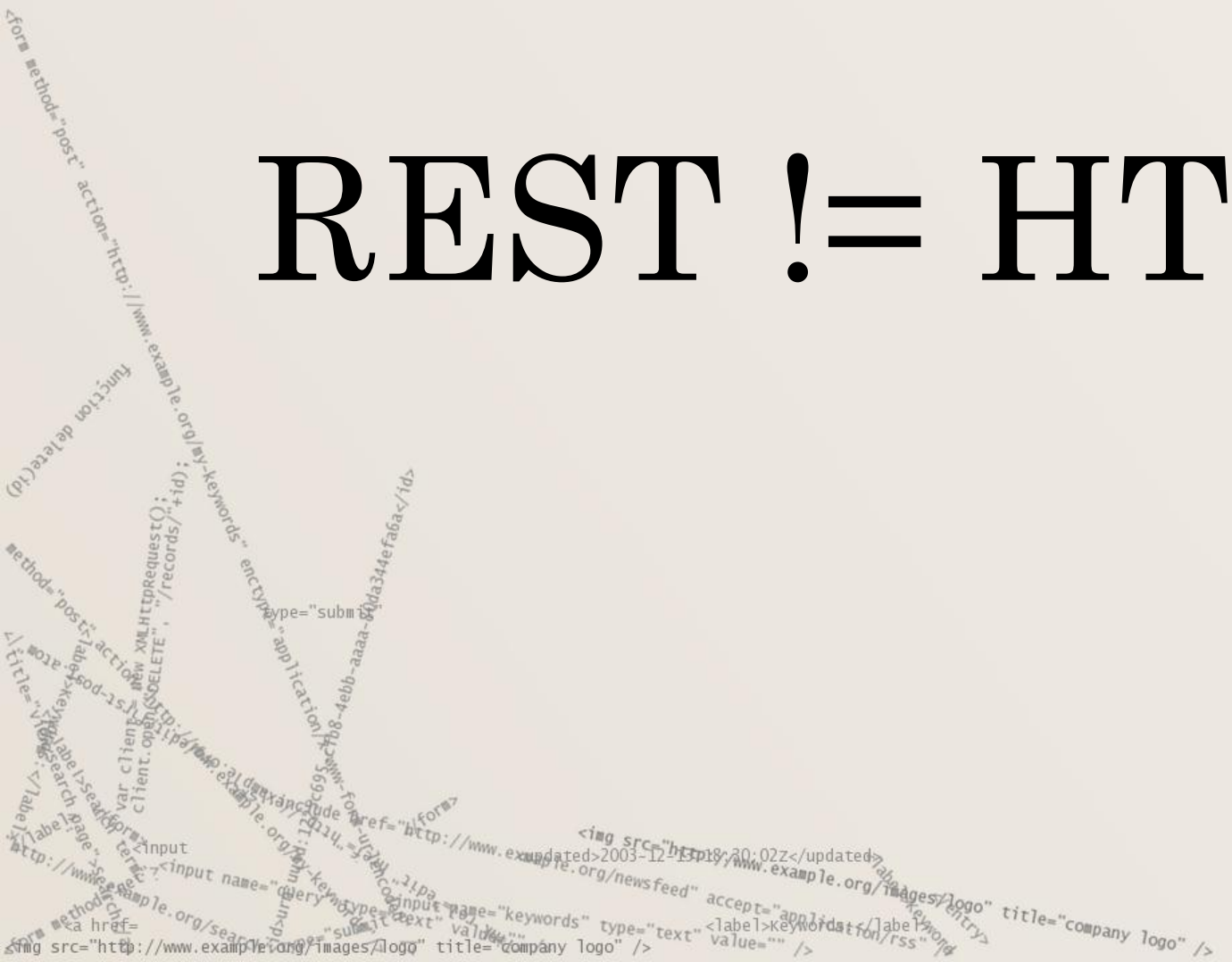
HTTP is a Standard Transfer Protocol


AIC

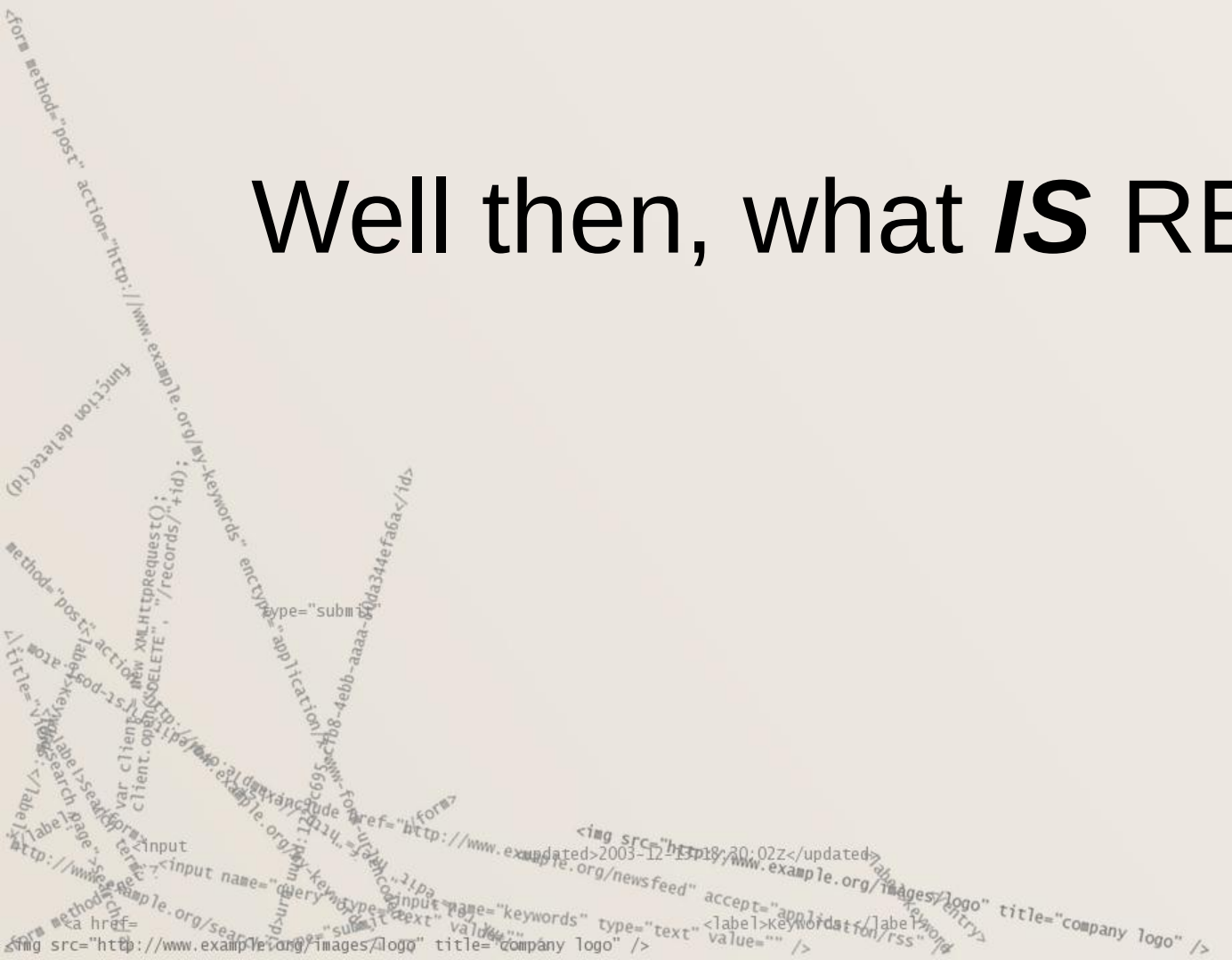
!=


NATIONAL ELECTRICAL CODE
nec2008
NFPA70

REST != HTTP


<form method="post" action="http://www.example.org/by-keywords" enctype="application/x-www-form-urlencoded">
function del(id)
var client = new XMLHttpRequest();
client.open("DELETE", "/records/"+id);
type="submit"
<label>keywords</label>
<input name="keywords" type="text" value="" />
<updated>2003-12-13T18:30:02Z</updated>

Well then, what ***IS*** REST?



UNIVERSITY OF CALIFORNIA, IRVINE

Architectural Styles and the Design of Network-based Software Architectures

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Roy Thomas Fielding

2000

Roy T. Fielding, 2000

“REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations.”

Roy T. Fielding, 2000

“REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations.”

Roy T. Fielding, 2000

“REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations.”

Roy T. Fielding, 2000

“REST is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations.”

Roy T. Fielding, 2000

“REST attempts to … minimize latency while maximizing scalability”

Roy T. Fielding, 2000

Srsly?

But some people say…

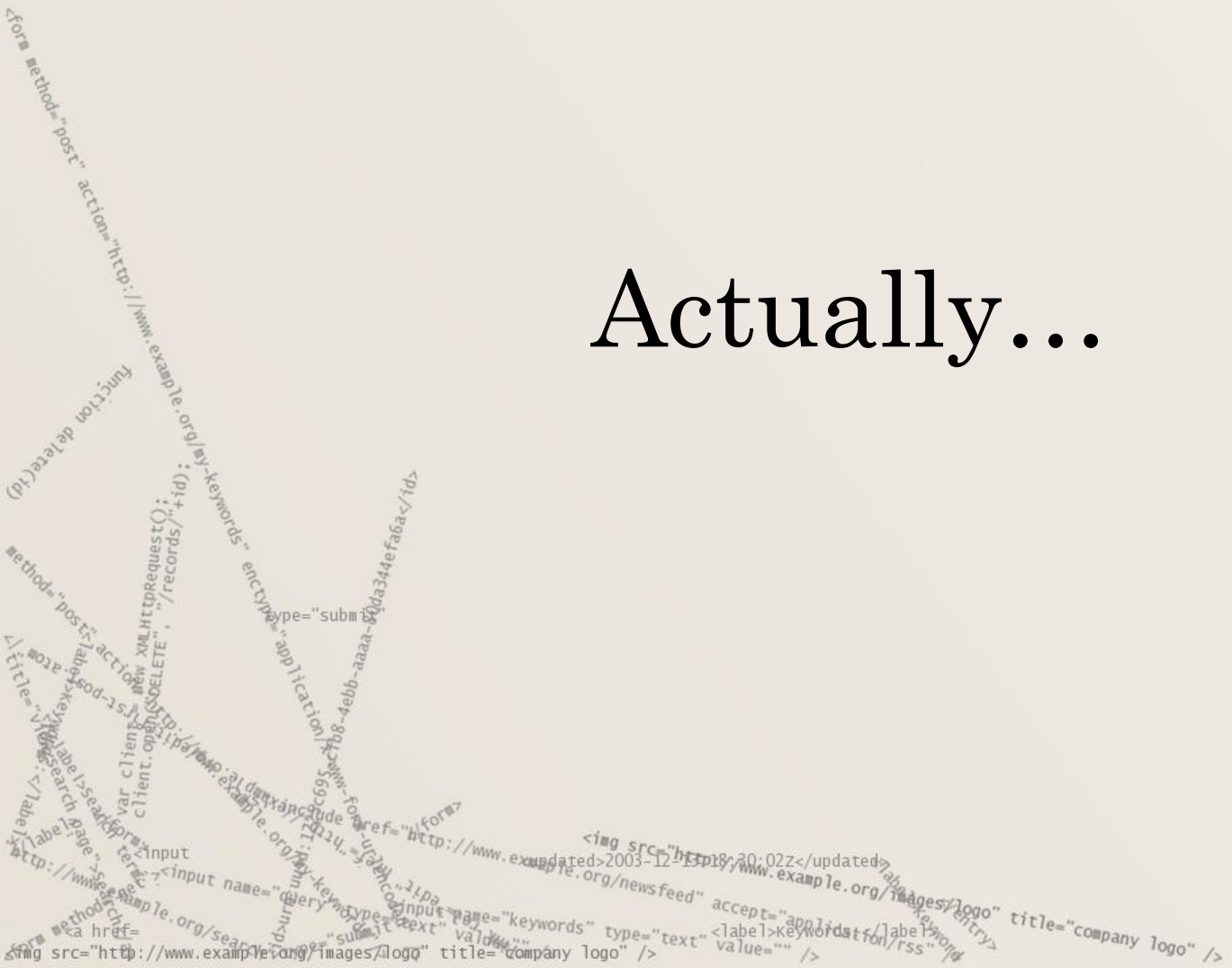
REST IS…

- Crafting URIs
- Identifying Resources
- Designing Responses (JSON, XML, etc.)
- HTTP Verbs (GET, PUT, POST, DELETE)
- Headers (Caching, etc.)
- HTTP Response Codes (200, 404, 418, etc.)

REST IS…

- Crafting URIs
- Identifying Resources
- Designing Responses (JSON, XML, etc.)
- HTTP Verbs (GET, PUT, POST, DELETE)
- Headers (Caching, etc.)
- HTTP Response Codes (200, 404, 418, etc.)

Actually…


<form method="post" action="http://www.example.org/by-keywords" enctype="application/x-www-form-urlencoded">
function delete(id)
var client = new XMLHttpRequest();
client.open("DELETE", "/records/"+id);
type="submit"
<updated>2003-12-13T18:30:02Z</updated>
<label>keywords</label>
<input name="keywords" type="text" value="" />
<label>search page</label>
<input name="query" type="text" />
<a href=

REST IS NOT…

- Crafting URIs
- Identifying Resources
- Designing Responses (JSON, XML, etc.)
- HTTP Verbs (GET, PUT, POST, DELETE)
- Headers (Caching, etc.)
- HTTP Response Codes (200, 404, 418, etc.)

REST IS NOT…

- Crafting URIs
- Identifying Resources
- Designing Responses (JSON, XML, etc.)
- HTTP Verbs (GET, PUT, POST, DELETE)
- Headers (Caching, etc.)
- HTTP Response Codes (200, 404, 418, etc.)

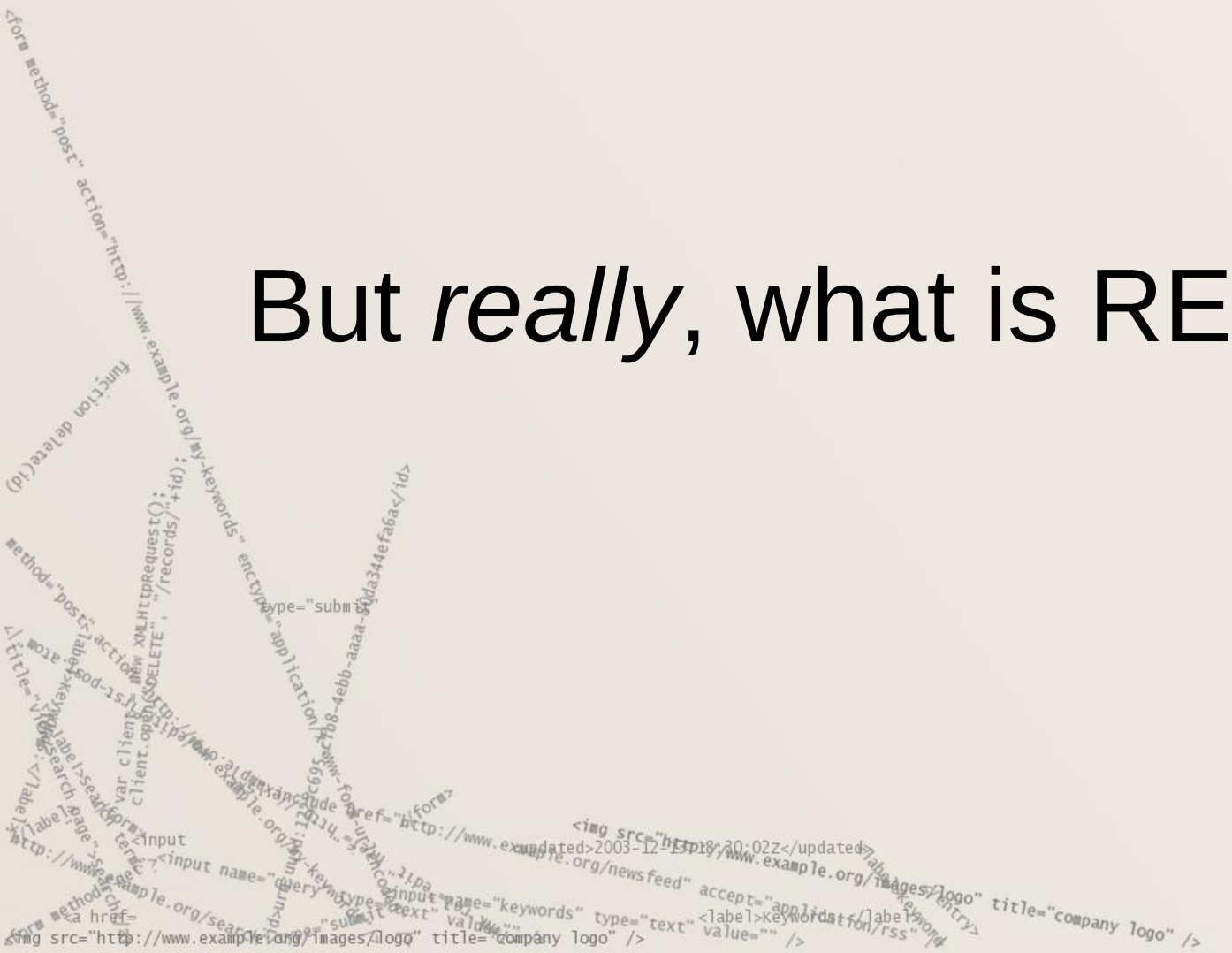
REST IS NOT…

- Crafting URIs
- Identifying Resources
- Designing Responses (JSON, XML, etc.)
- HTTP Verbs (GET, PUT, POST, DELETE)
- Headers (Caching, etc.)
- HTTP Response Codes (200, 404, 418, etc.)

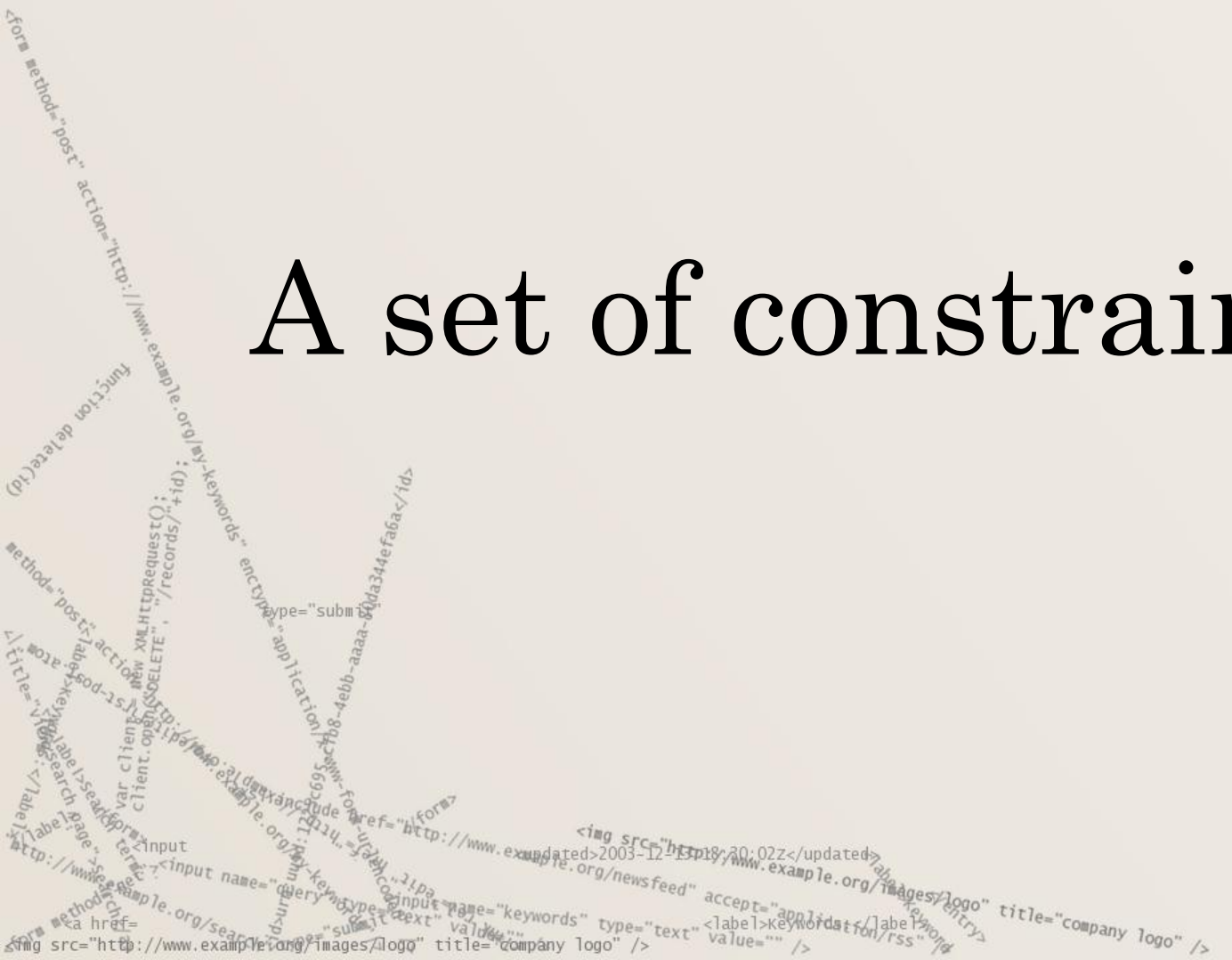
REST IS NOT…

- Crafting URIs
- Identifying Resources
- Designing Responses (JSON, XML, etc.)
- HTTP Verbs (GET, PUT, POST, DELETE)
- Headers (Caching, etc.)
- HTTP Response Codes (200, 404, 418, etc.)

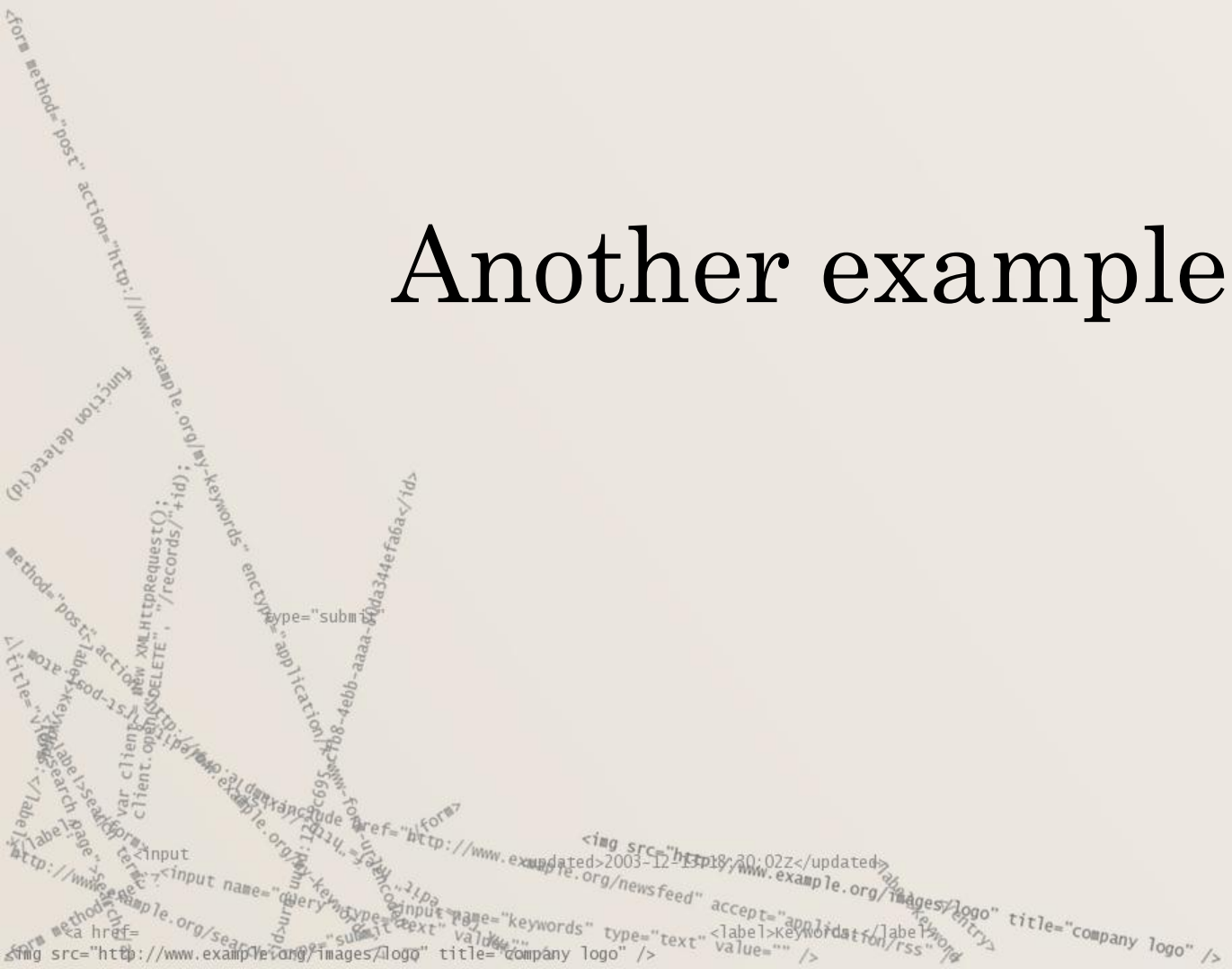
But *really*, what is REST?



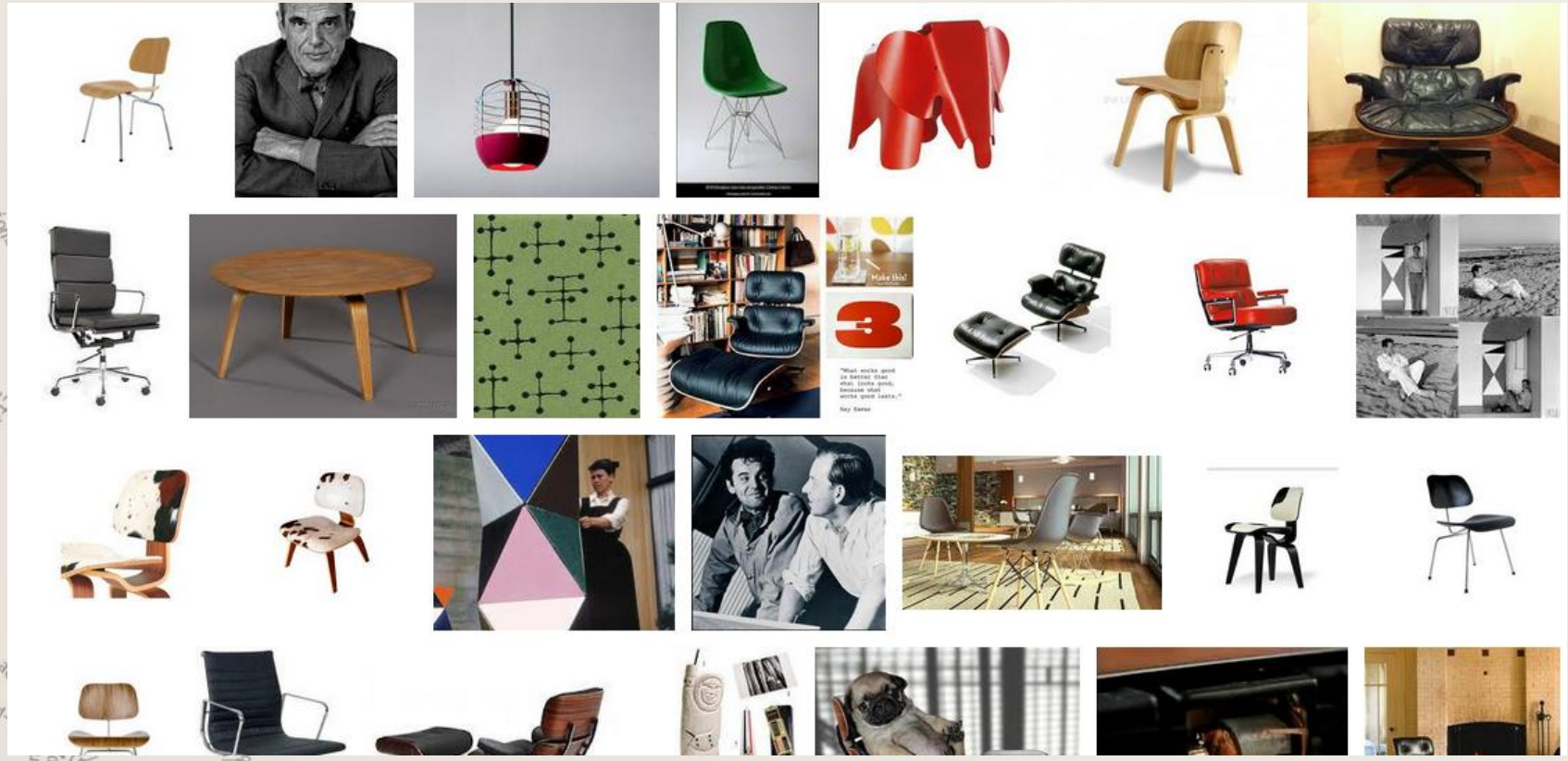
A set of constraints.



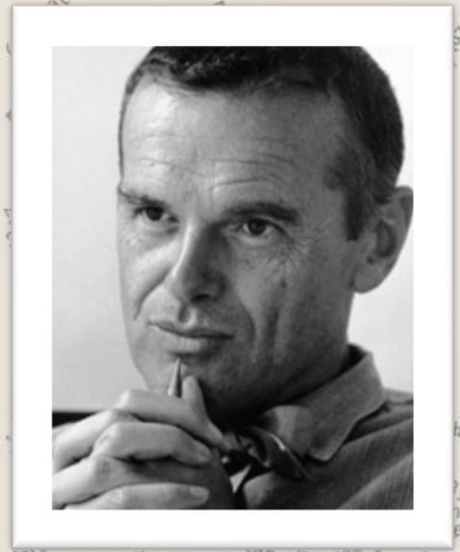
Another example…



Charles Eames

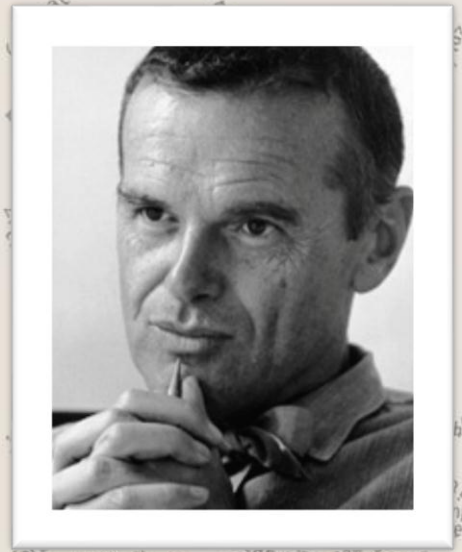


"I have never been forced to accept compromises but I have willingly accepted constraints."



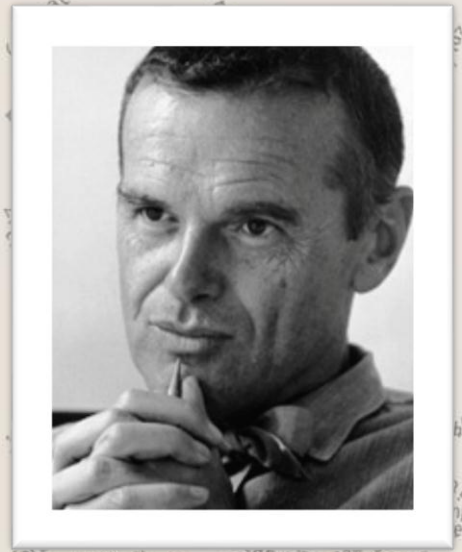
Charles Eames,
Eames Design, 1989

"I have never been forced to accept compromises but I have willingly accepted constraints."



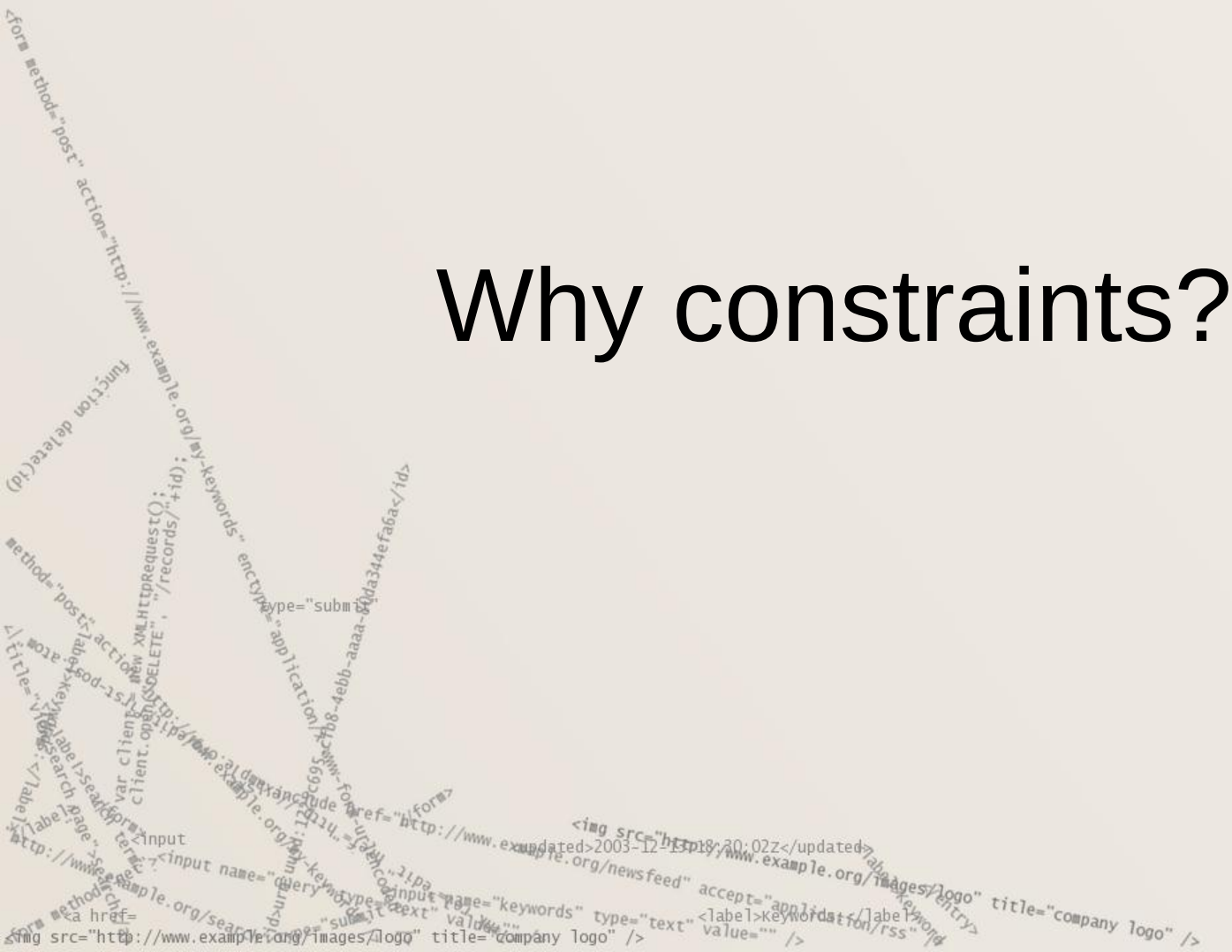
Charles Eames,
Eames Design, 1989

"I have never been forced to accept compromises but I have willingly accepted constraints."



Charles Eames,
Eames Design, 1989

Why constraints?



Is
there
anything
more
intimidating…?

OK, what are REST's constraints?

There are six REST constraints…

Client-Server

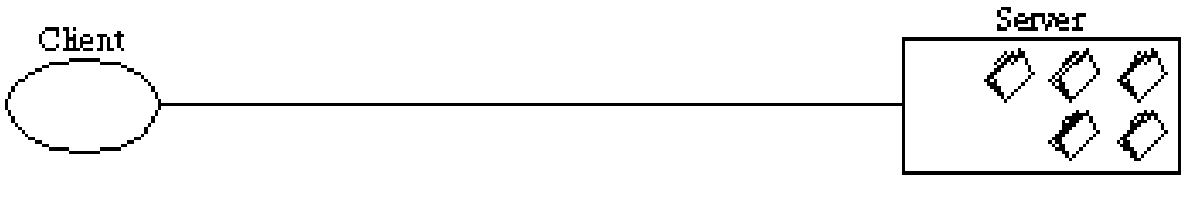


Figure 5-2. Client-Server

Not Peer-to-Peer

Stateless

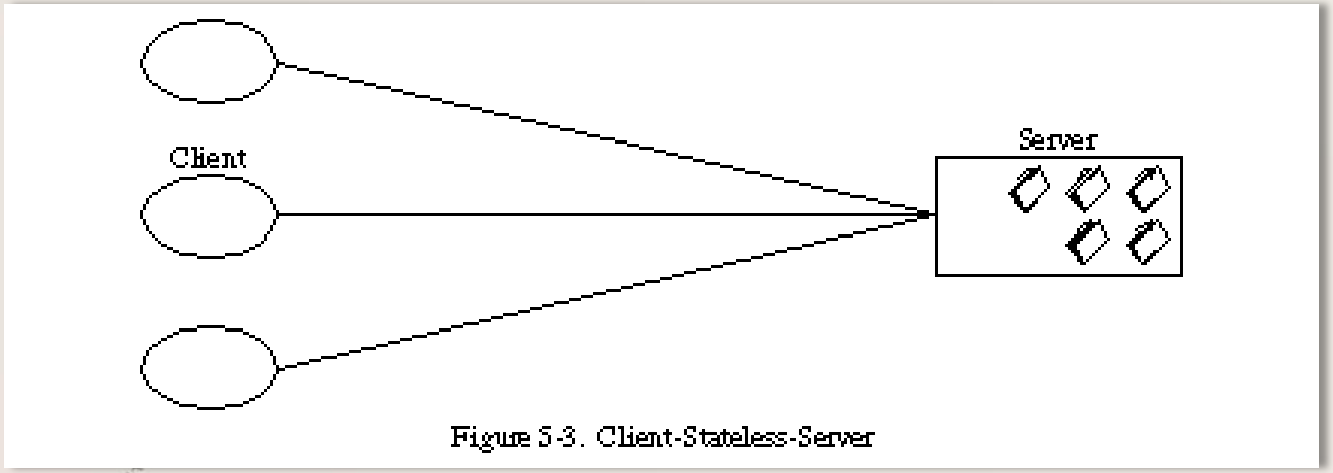


Figure 5-3. Client-Stateless-Server

No Server-Side Session

Cache

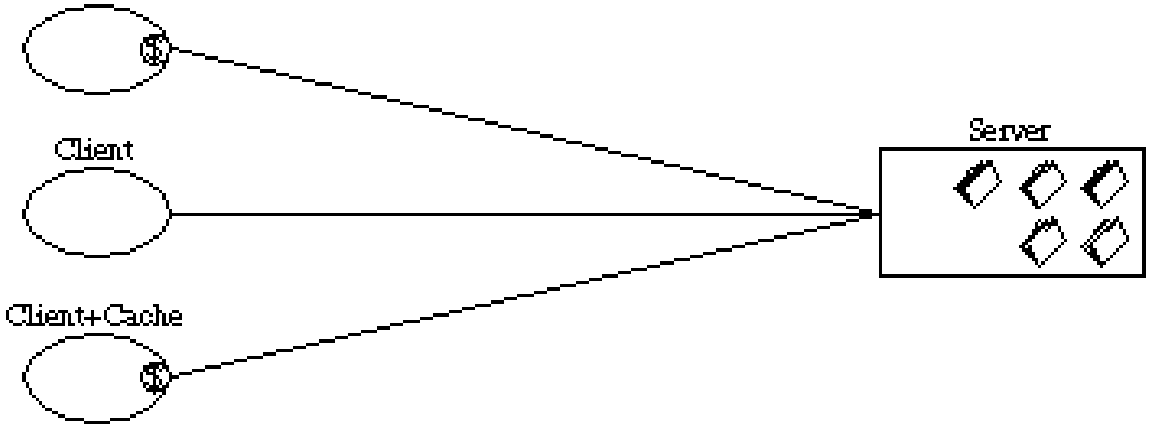


Figure 5-4. Client-Cache-Stateless-Server

Don't hit server every time

Uniform Interface

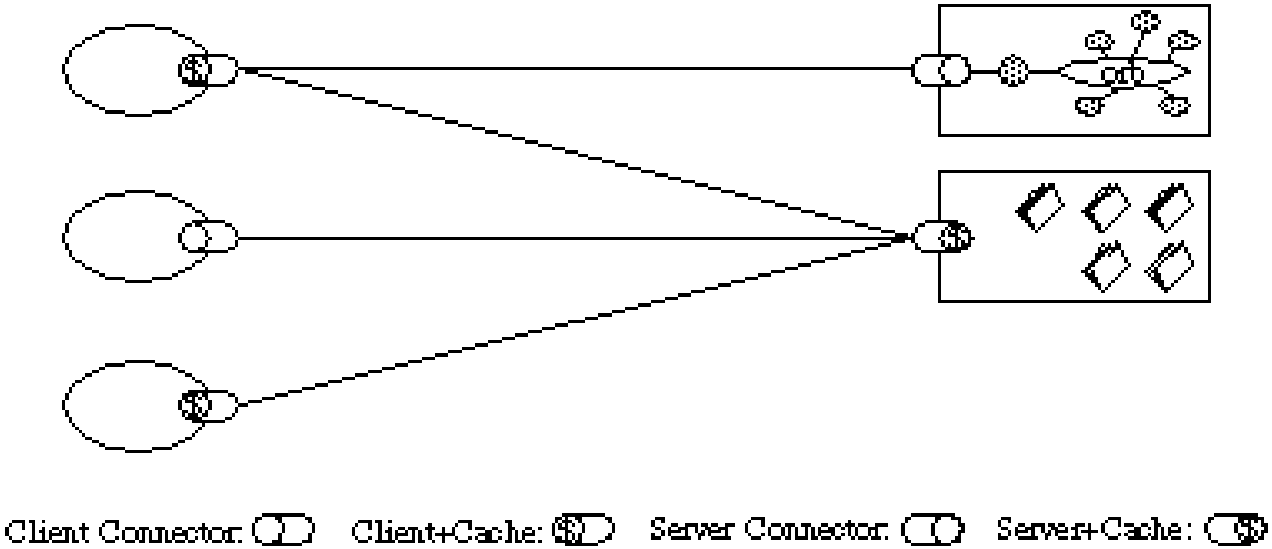


Figure 5-6. Uniform-Client-Cache-Stateless-Server

Same API for everyone

Layered System

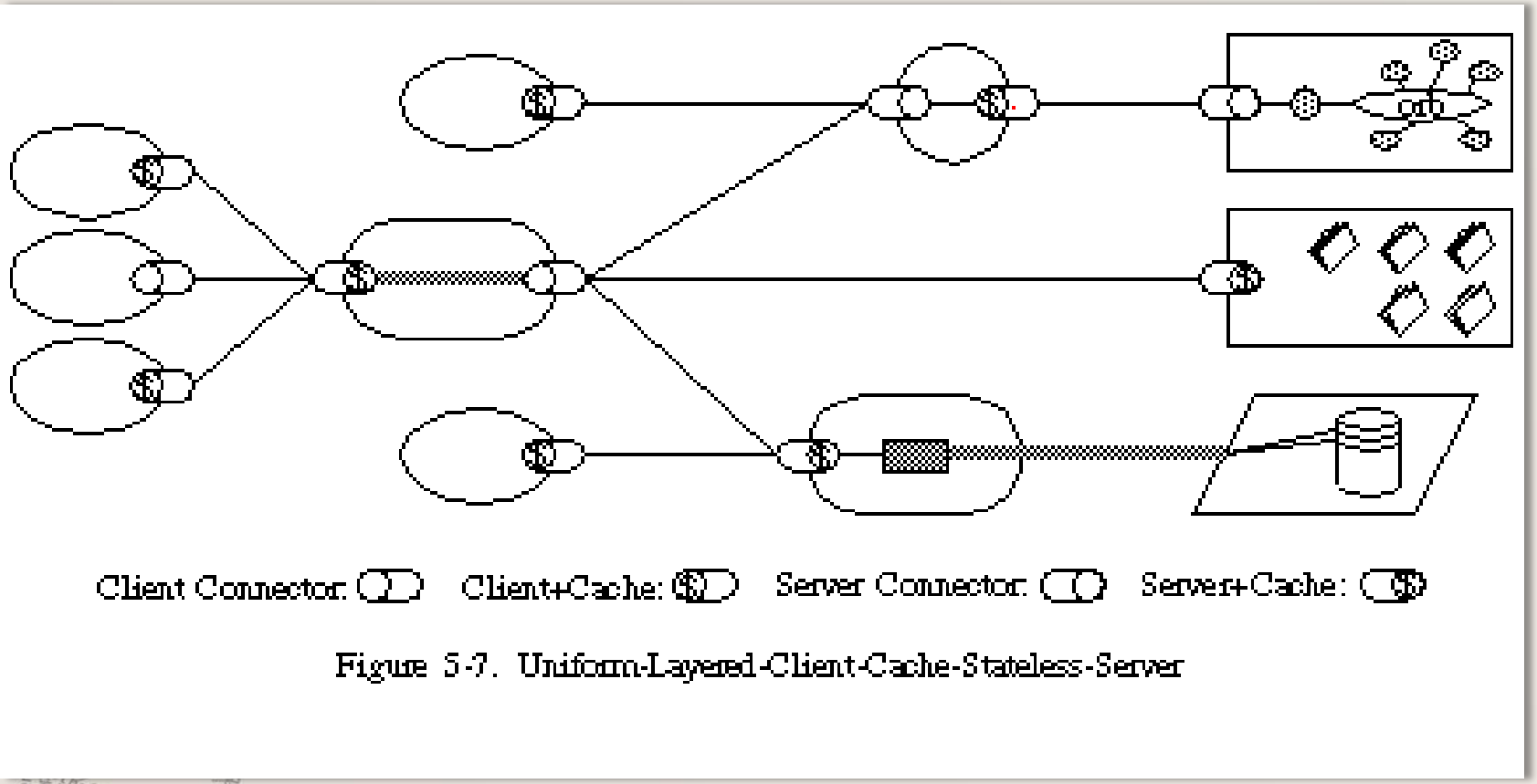


Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

Add hardware any time

Code On Demand *

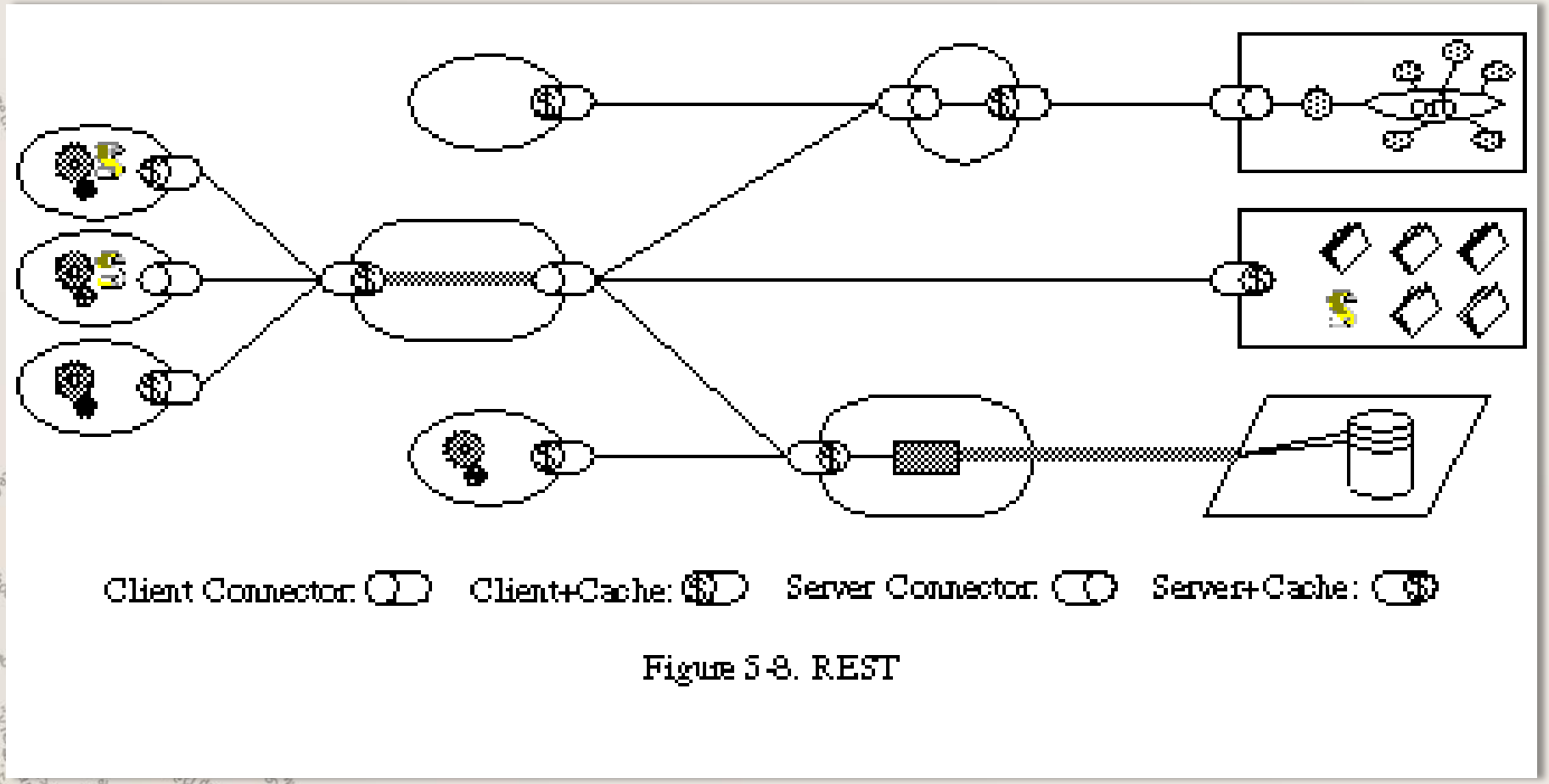


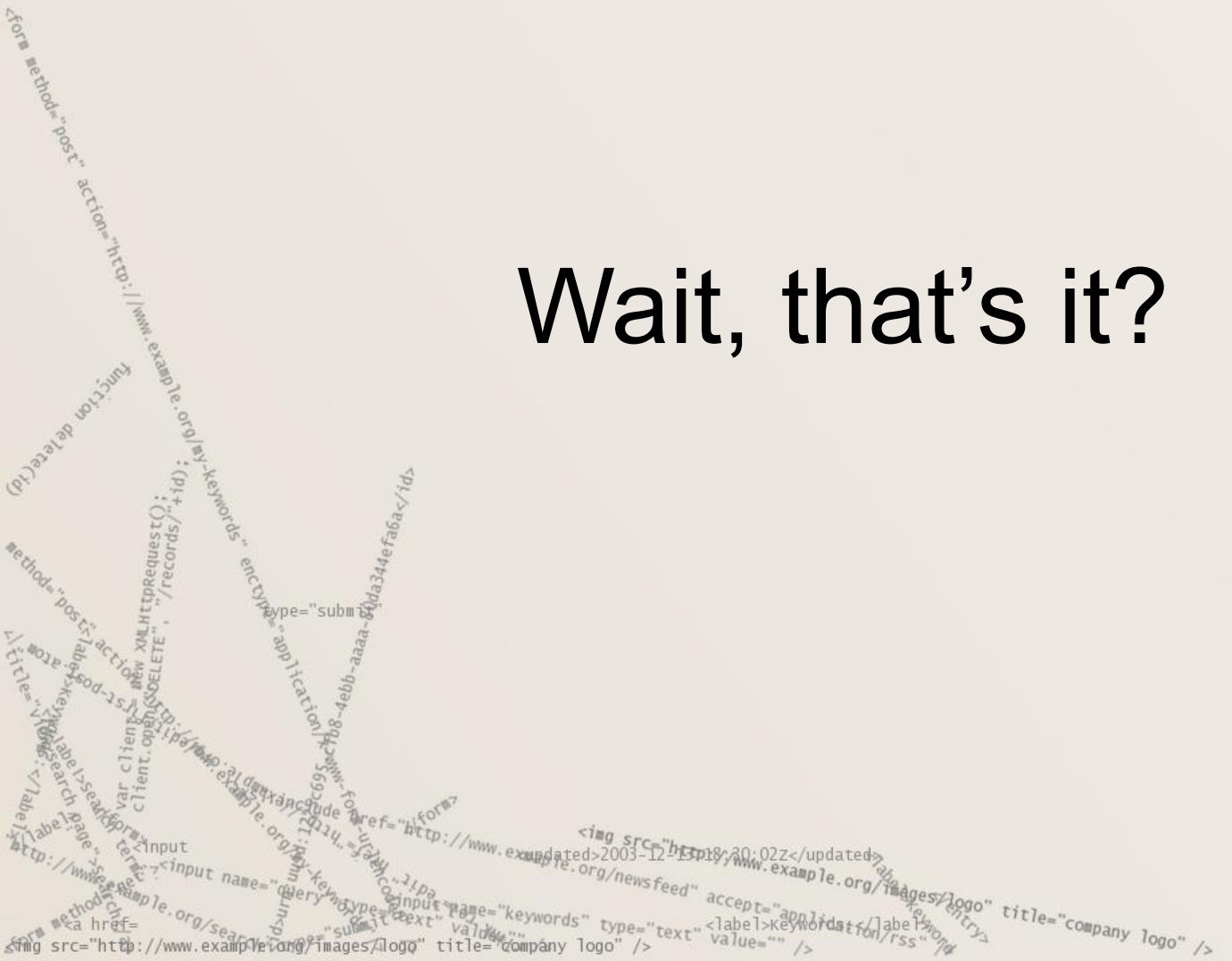
Figure 5-8. REST

Don't hard-code clients, send code to them.

REST's Constraints

1. Client-Server
2. Stateless
3. Cache
4. Uniform Interface
5. Layered System
6. Code on Demand

Wait, that’s it?



Well…

"[REST] is achieved by placing constraints on connector semantics where other styles have focused on component semantics."

Roy T. Fielding, 2000

"[REST] is achieved by placing constraints on connector semantics where other styles have focused on component semantics."

Roy T. Fielding, 2000

"[REST] is achieved by placing constraints on connector semantics where other styles have focused on component semantics."

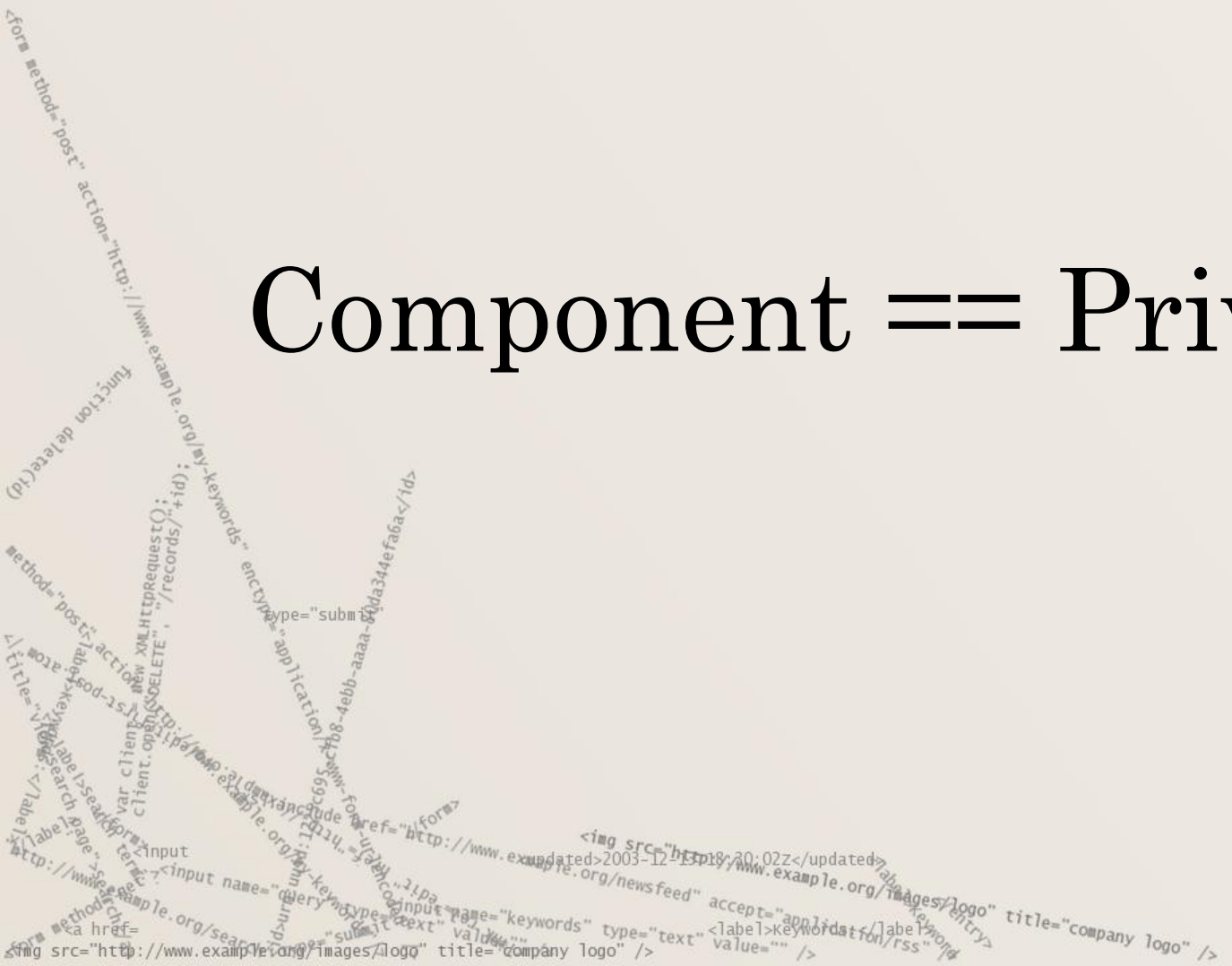
Roy T. Fielding, 2000

Connector != Component

Component

- Database
- File System
- Message Queue
- Transaction Manager
- Source Code

Component == Private

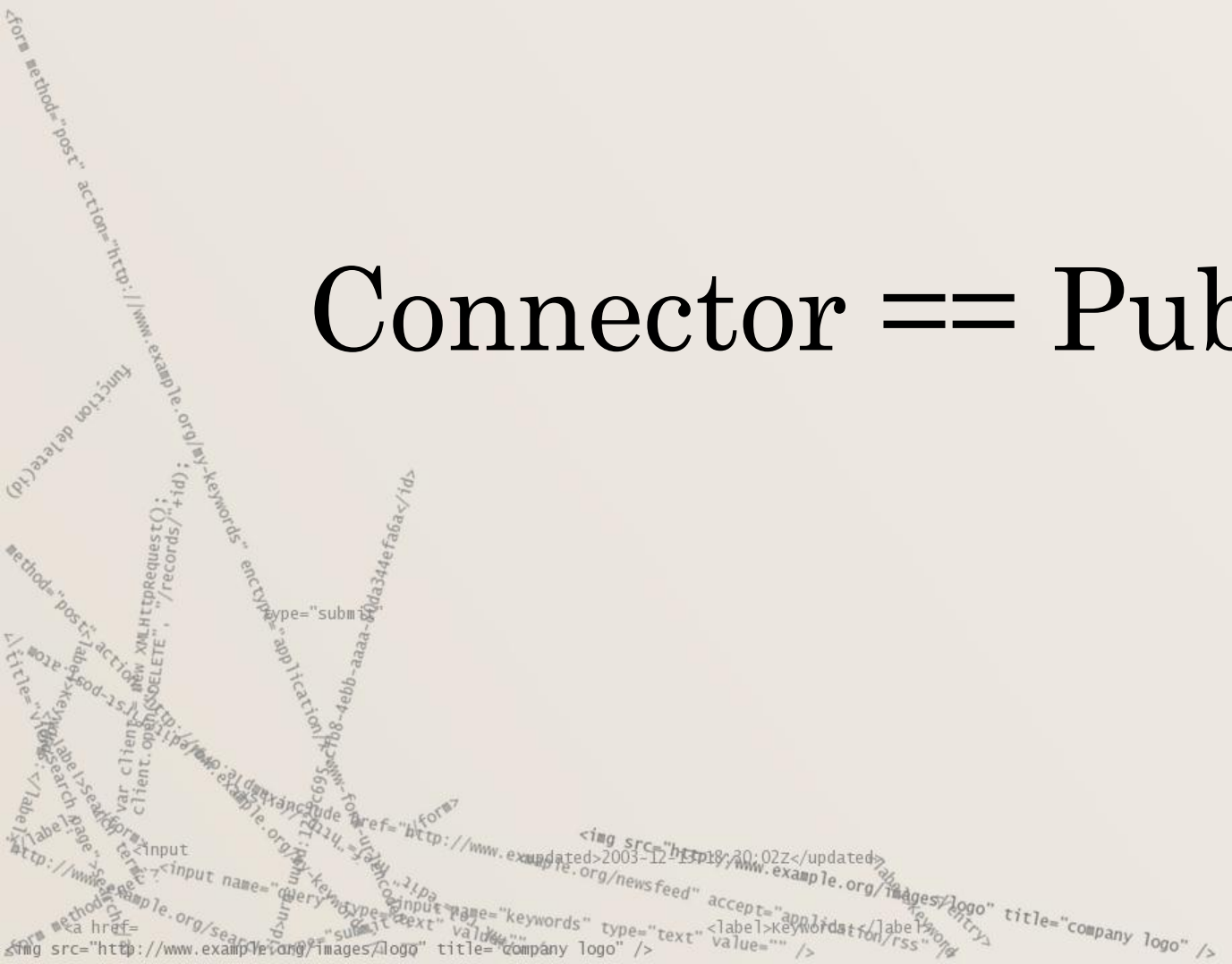


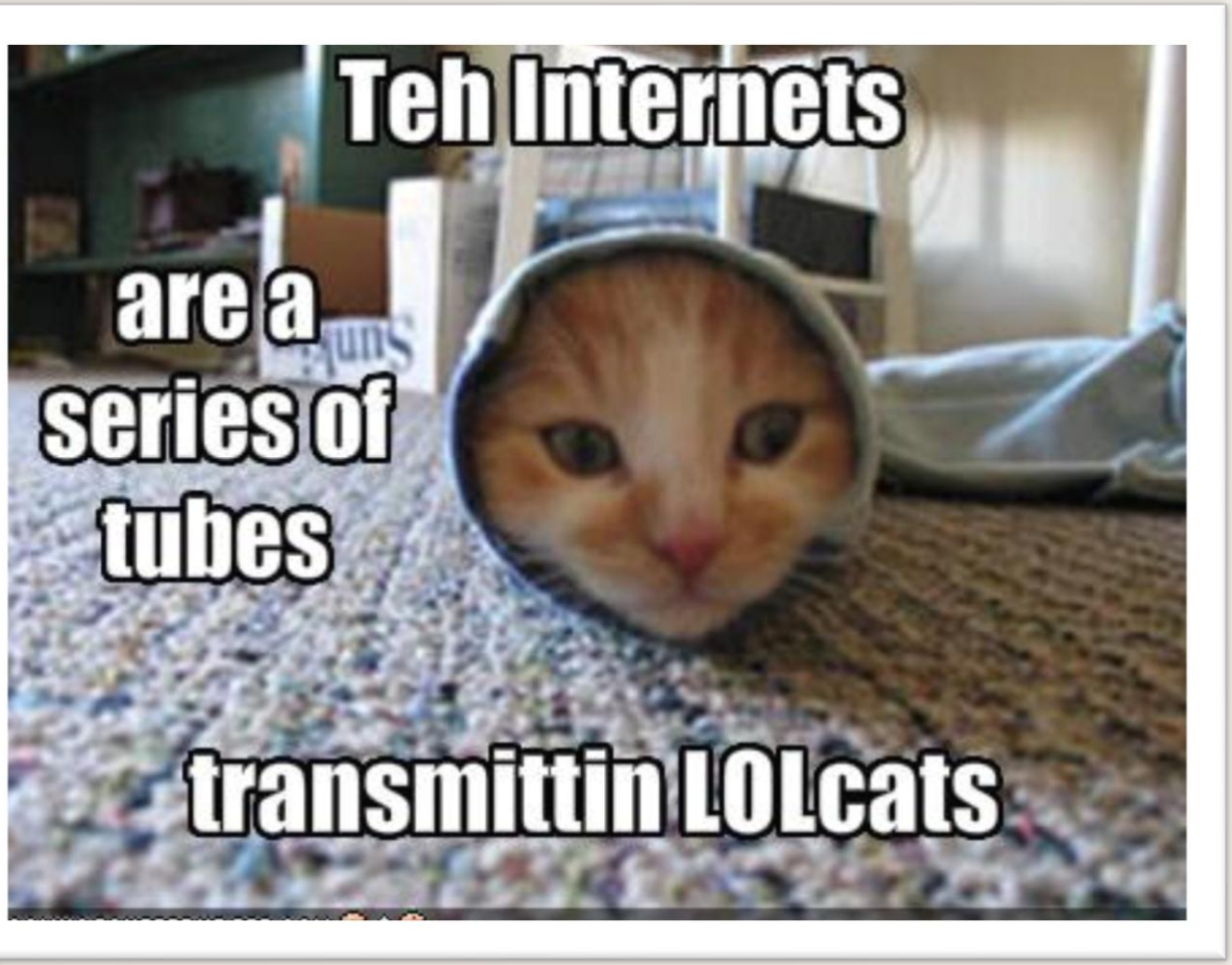

I ar hiding
u kant c mee

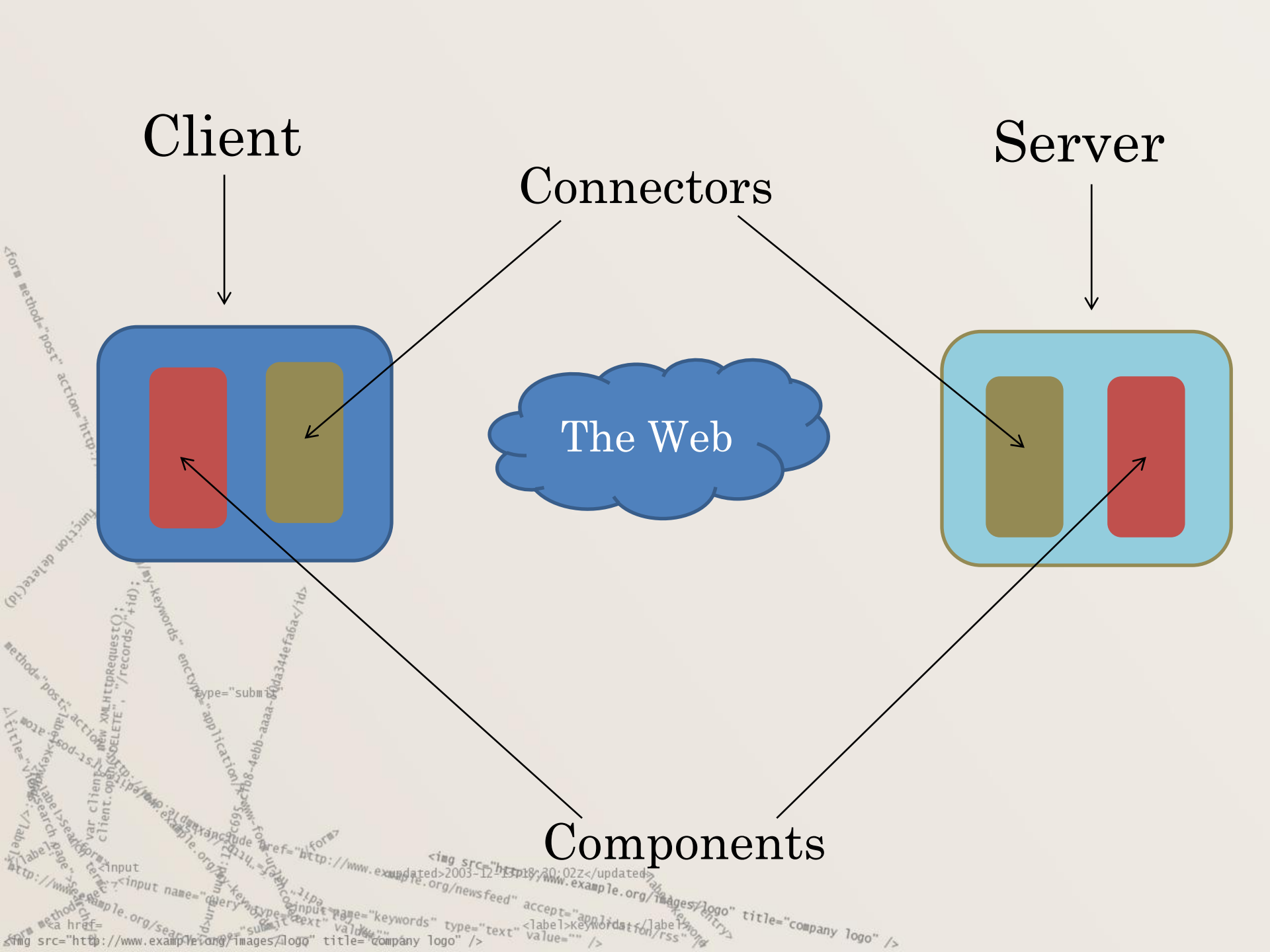
Connector

- Web Server
- Browser Agent
- Proxy Server
- Shared Cache

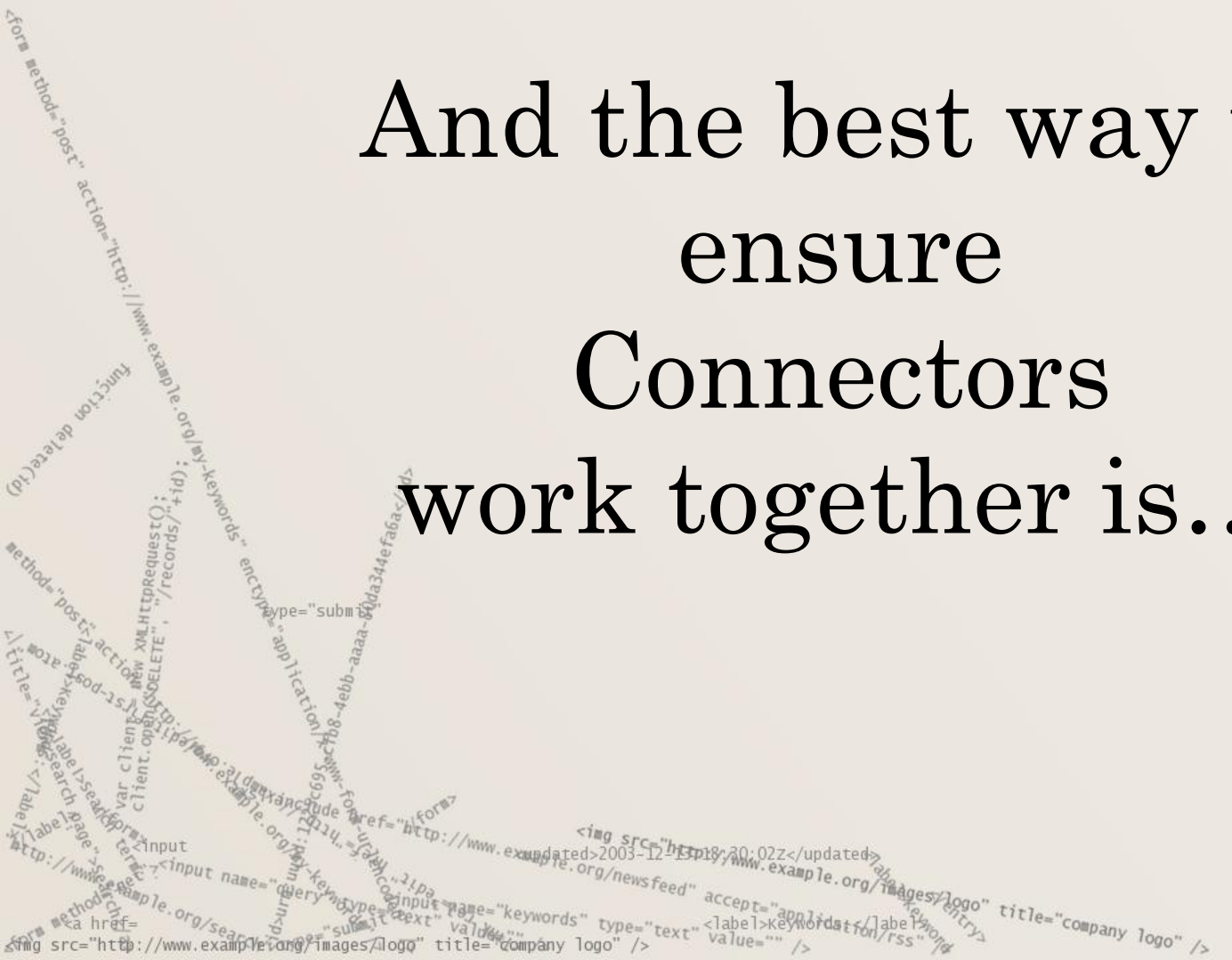
Connector == Public




Teh Internets
are a
series of
tubes
transmittin LOLcats


Client
Connectors
Server
The Web
Components

And the best way to ensure Connectors work together is…



Uniform Interface

1. Identification of Resources
2. Resource Representations
3. Self-Descriptive Messages
4. Hypermedia

Uniform Interface

1. Identification of Resources **(URIs)**
2. Resource Representations
3. Self-Descriptive Messages
4. Hypermedia

Uniform Interface

1. Identification of Resources **(URIs)**
2. Resource Representations **(Media-Types)**
3. Self-Descriptive Messages
4. Hypermedia

Uniform Interface

1. Identification of Resources **(URIs)**
2. Resource Representations **(Media-Types)**
3. Self-Descriptive Messages **(Header+Body)**
4. Hypermedia

Uniform Interface

1. Identification of Resources **(URIs)**
2. Resource Representations **(Media-Types)**
3. Self-Descriptive Messages **(Header+Body)**
4. Hypermedia **(Links & Forms)**

URIs

```
<scheme>://<authority><path>?<query>
```

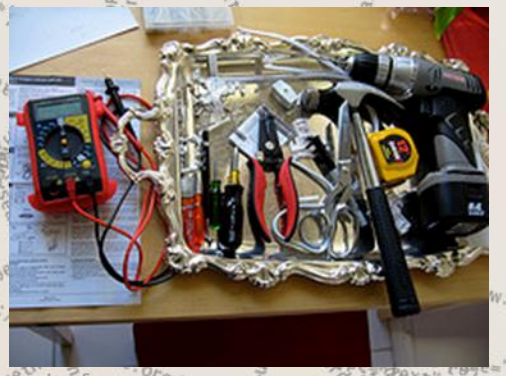
"A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource."

RFC2396 – URI Generic Syntax

URIs

```
<scheme>://<authority><path>?<query>
```

"A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource."



RFC2396 – URI Generic Syntax

MIME Media Types

```
type := discrete-type / composite-type
```

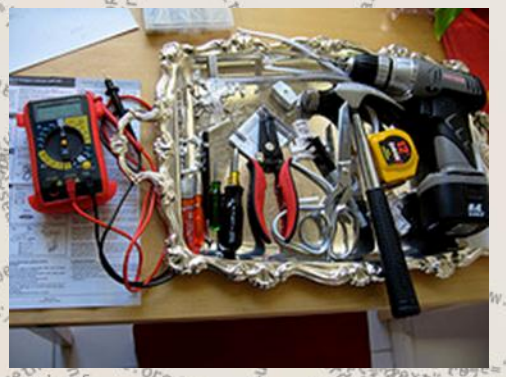
"Specify the nature of the data in the body of a MIME entity and auxiliary information that may be required for certain media types."

RFC2046 – MIME Media Types

MIME Media Types

```
type := discrete-type / composite-type
```

"Specify the nature of the data in the body of a MIME entity and auxiliary information that may be required for certain media types."

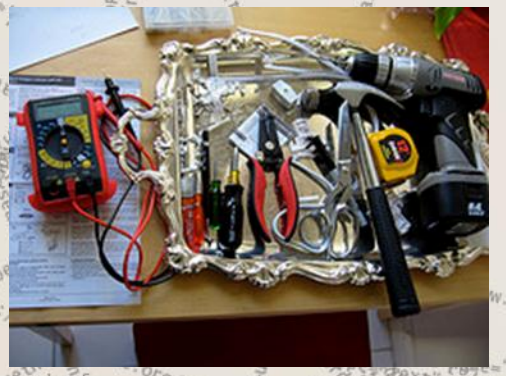


RFC2046 – MIME Media Types

Header + Body

```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
```

"Message[s] consist of a start-line, zero or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and possibly a message-body."

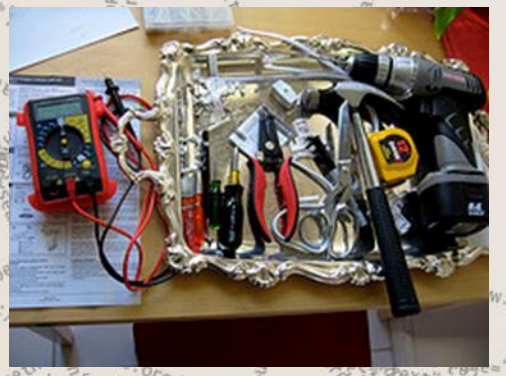


RFC2616 – Hypertext Transfer Protocol HTTP/1.1

Header + Body

```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
```

"Message[s] consist of a start-line, zero or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and possibly a message-body."



RFC2616 – Hypertext Transfer Protocol HTTP/1.1

Links & Forms

```
<form method="get">
  <label>Search term:</label>
  <input name="query" type="text" value="" />
  <input type="submit" />
</form>
```

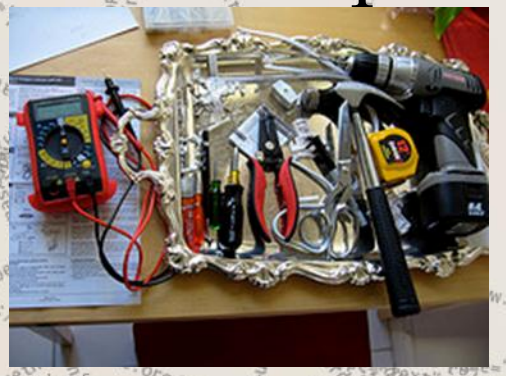
"Hypermedia is defined by the presence of application control information embedded within, or as a layer above, the presentation of information."

Roy. T. Fielding, 2000

Links & Forms

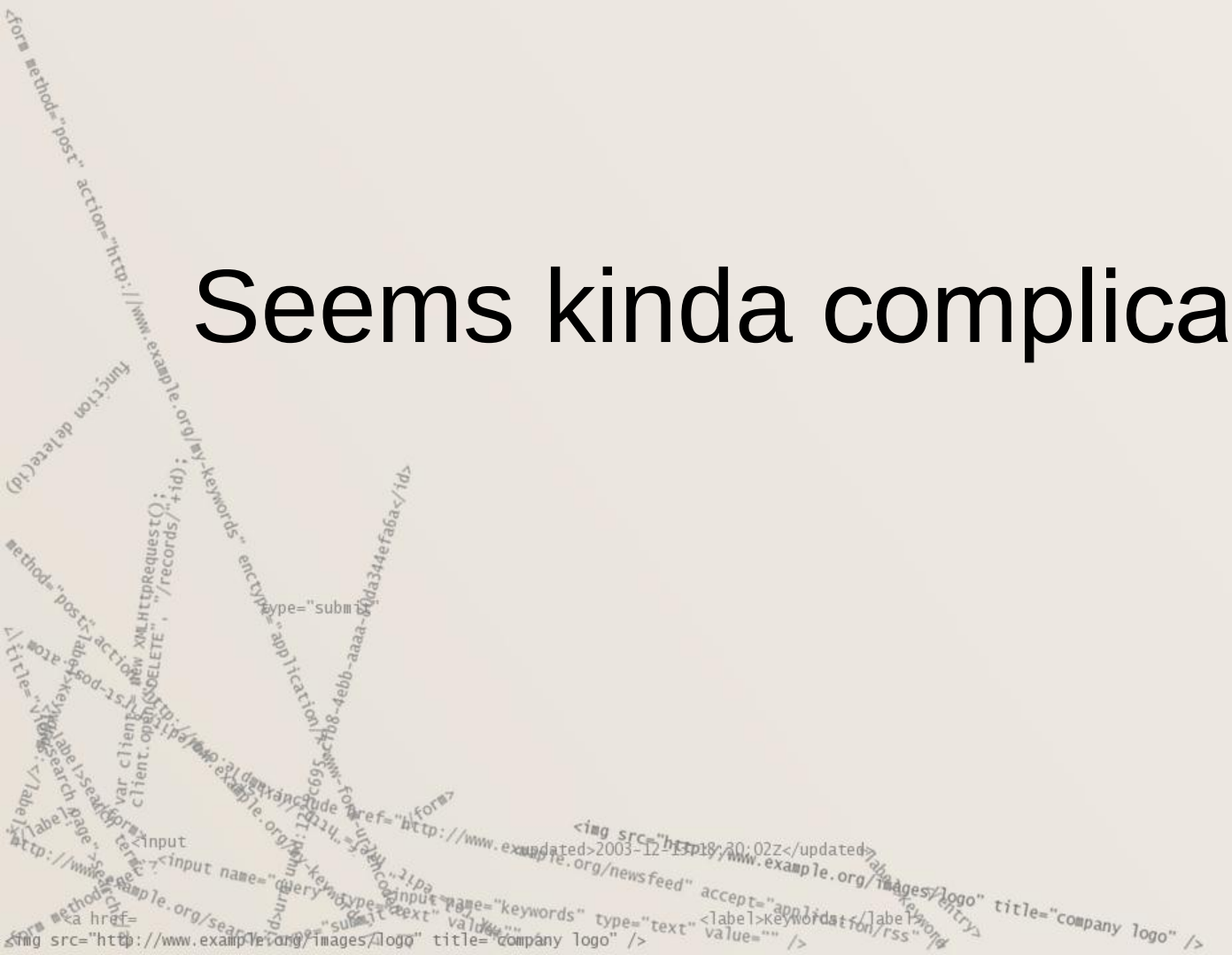
```
<form method="get">
  <label>Search term:</label>
  <input name="query" type="text" value="" />
  <input type="submit" />
</form>
```

"Hypermedia is defined by the presence of application control information embedded within, or as a layer above, the presentation of information."



Roy. T. Fielding, 2000

Seems kinda complicated…



REST in one slide

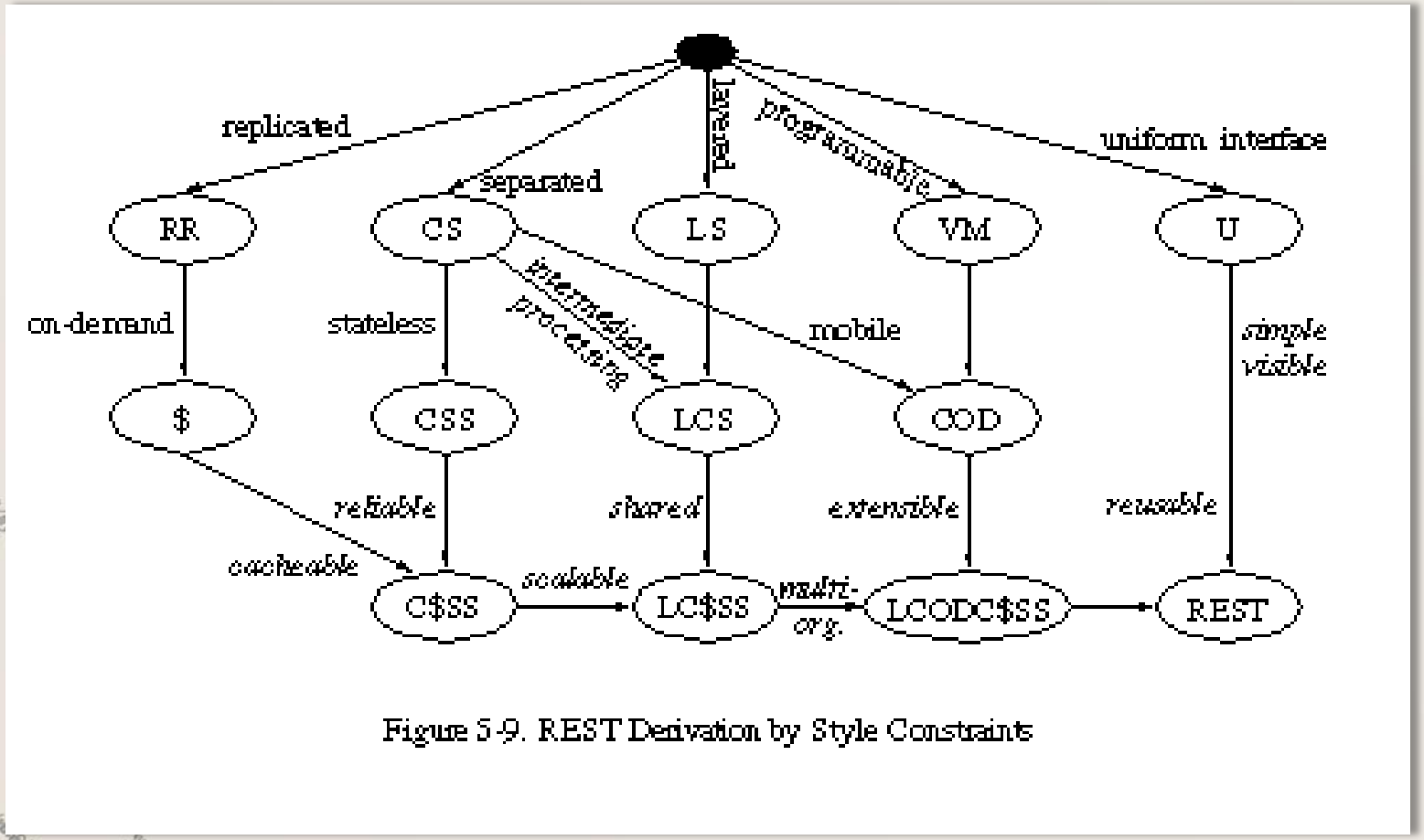
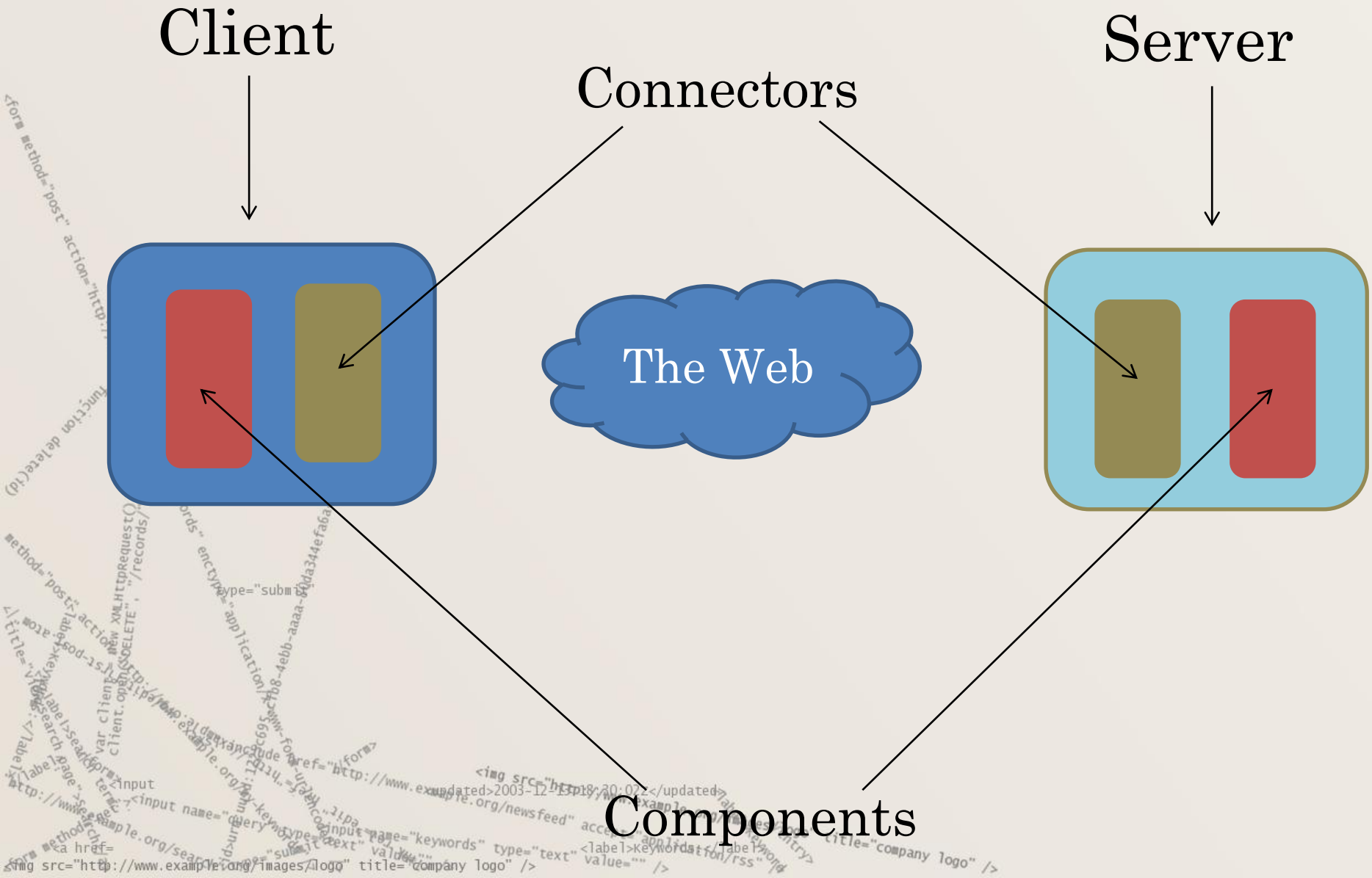


Figure 5-9. REST Derivation by Style Constraints

Read the dissertation, really!

Component ← → Connector ← → Connector ← → Component



"Those of us who have been trained as architects have this desire perhaps at the very center of our lives: that one day, somewhere, somehow, we shall build one building which is wonderful, beautiful, breathtaking, a place where people can walk and dream for centuries."

Christopher Alexander,
The Timeless Way of Building - 1979

"Those of us who have been trained as architects have this desire perhaps at the very center of our lives: that one day, somewhere, somehow, we shall build one building which is wonderful, beautiful, breathtaking, a place where people can walk and dream for centuries."

Christopher Alexander,
The Timeless Way of Building - 1979

"Those of us who have been trained as architects have this desire perhaps at the very center of our lives: that one day, somewhere, somehow, we shall build one building which is wonderful, beautiful, breathtaking, a place where people can walk and dream for centuries."

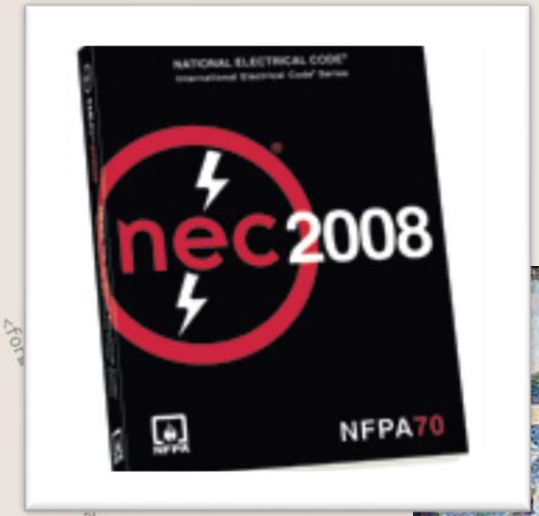
Christopher Alexander,
The Timeless Way of Building - 1979

"Those of us who have been trained as architects have this desire perhaps at the very center of our lives: that one day, somewhere, somehow, we shall build one building which is wonderful, beautiful, breathtaking, a place where people can walk and dream for centuries."

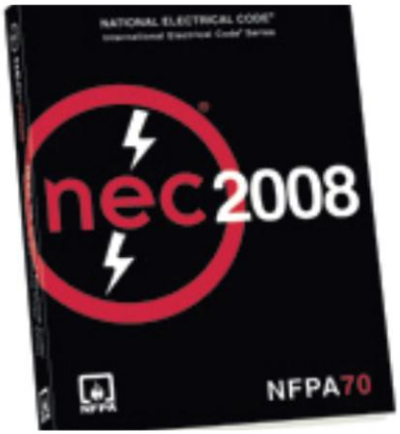
Christopher Alexander,
The Timeless Way of Building - 1979

So…

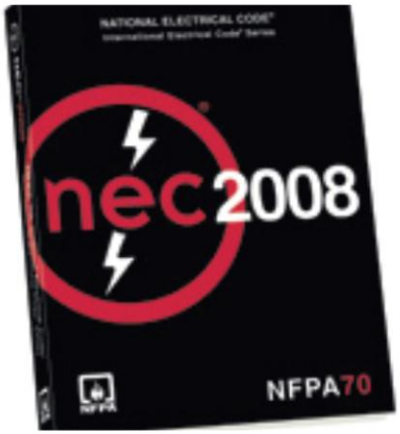

AIC


NATIONAL ELECTRICAL CODE
nec2008
NFPA70


AIC


NATIONAL ELECTRICAL CODE
nec2008
NFPA70


AIC



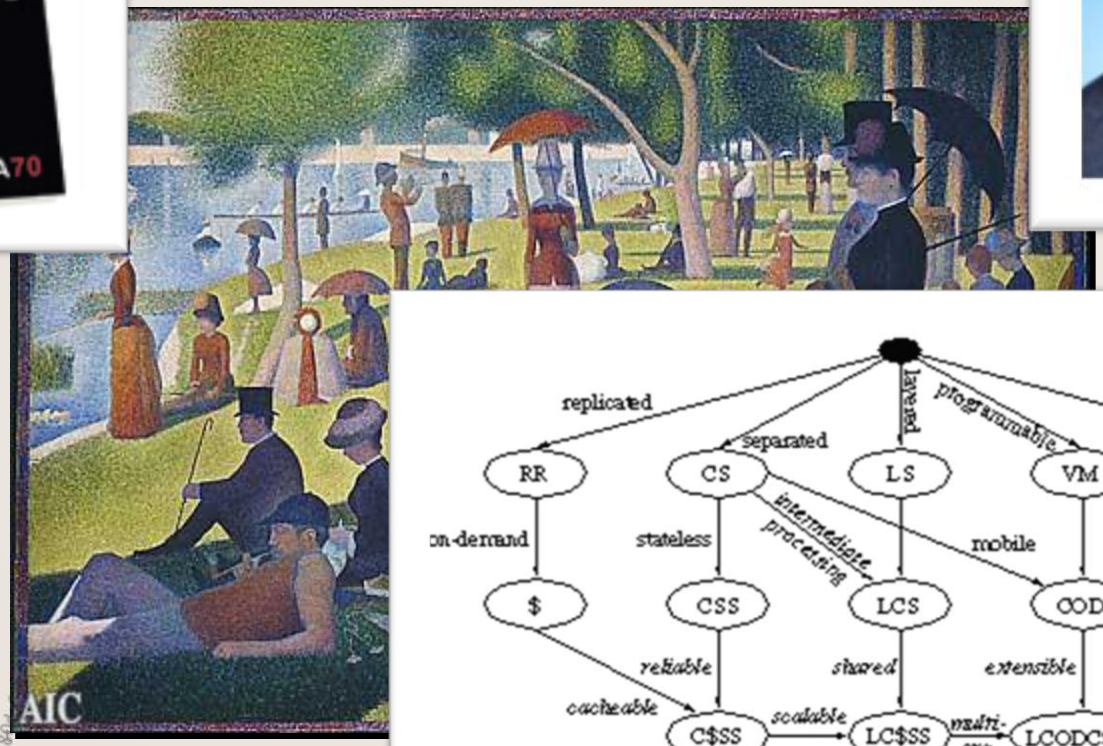
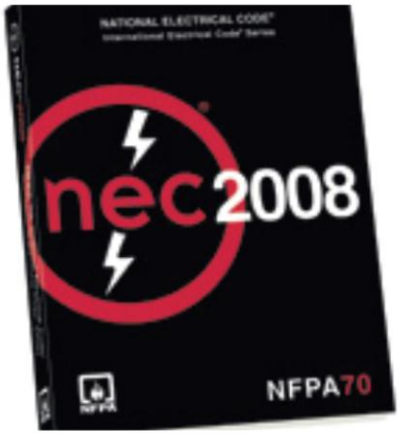
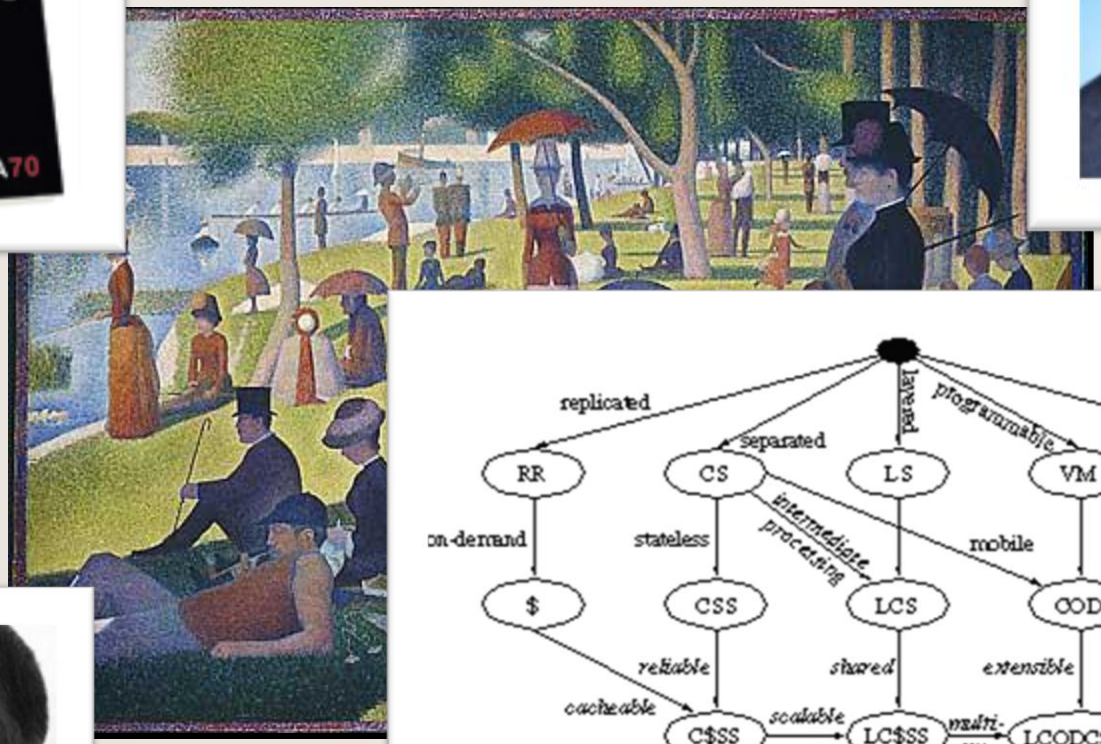


Figure 5-9. REST Derivation by Style Constraints





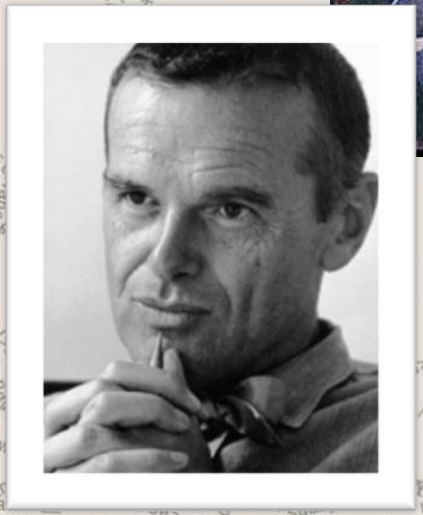
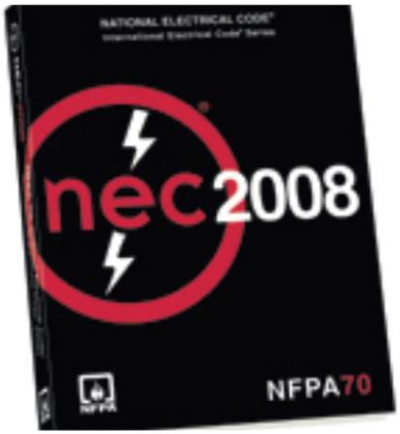


Figure 5-9. REST Derivation by Style Constraints

<updated>2003-12-13T18:30:02Z</updated>
<label>keywords</label>
<label>/rss</label>



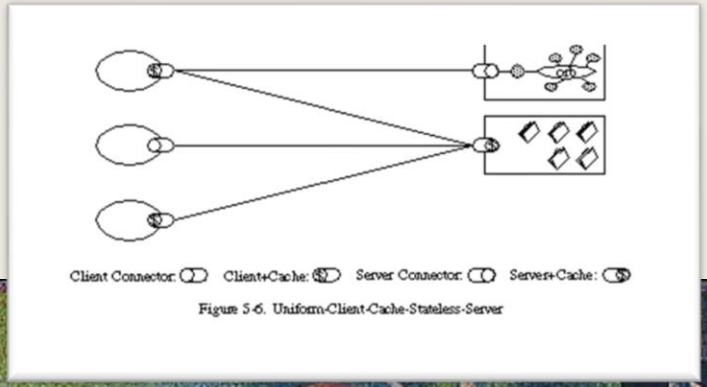
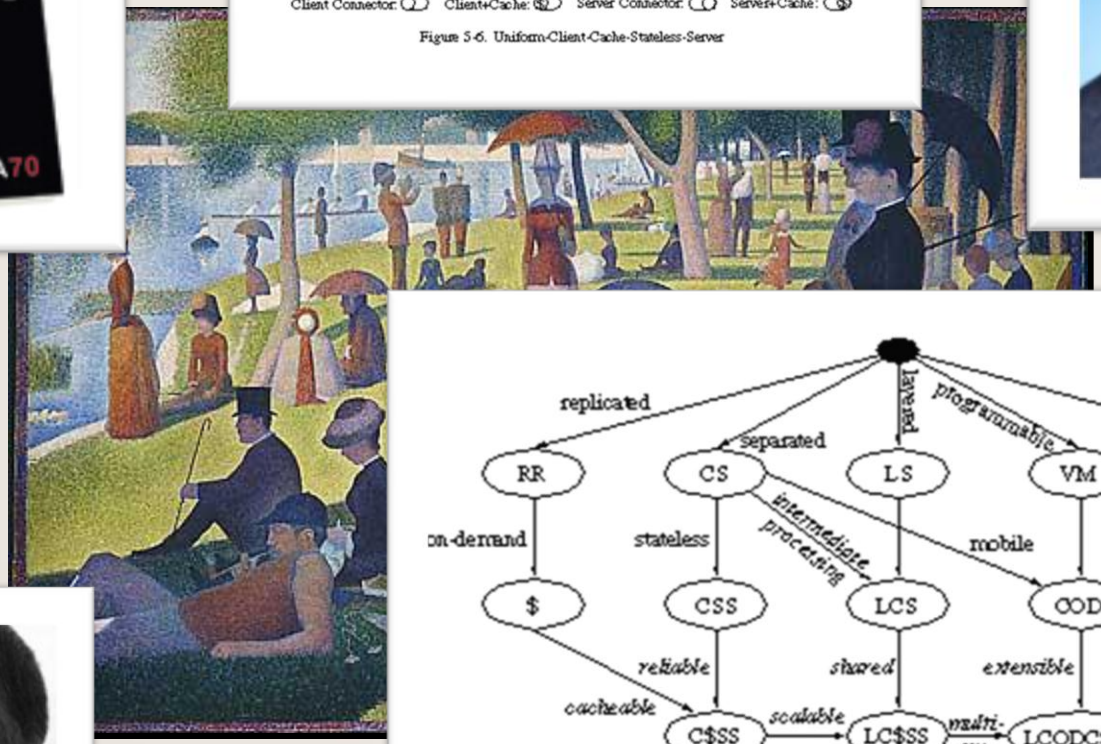


Figure 5-6. Uniform-Client-Cache-Stateless-Server



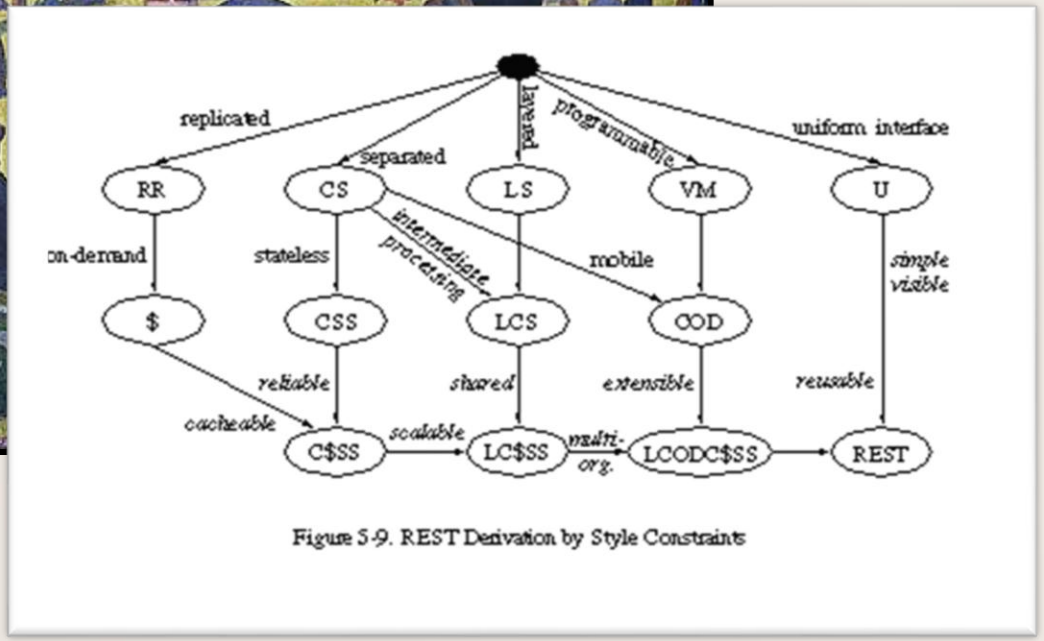
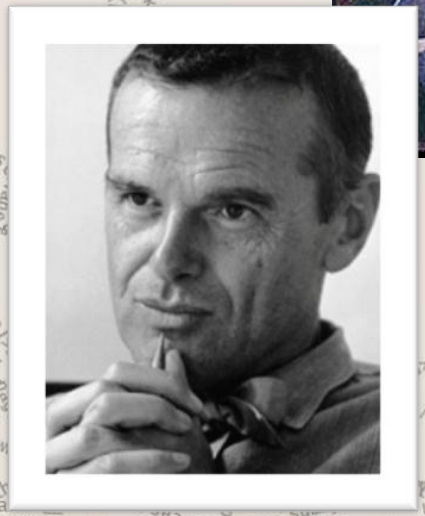
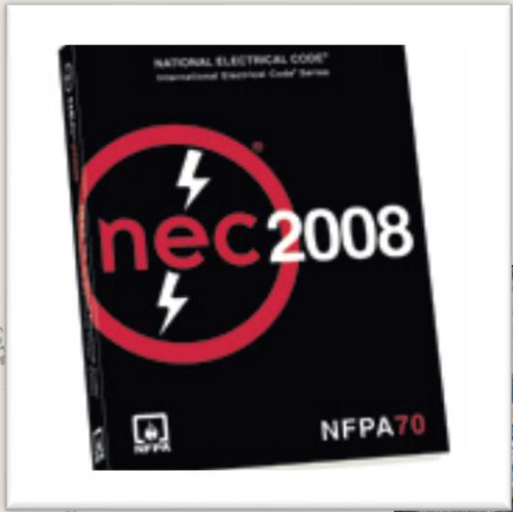


Figure 5-9. REST Derivation by Style Constraints





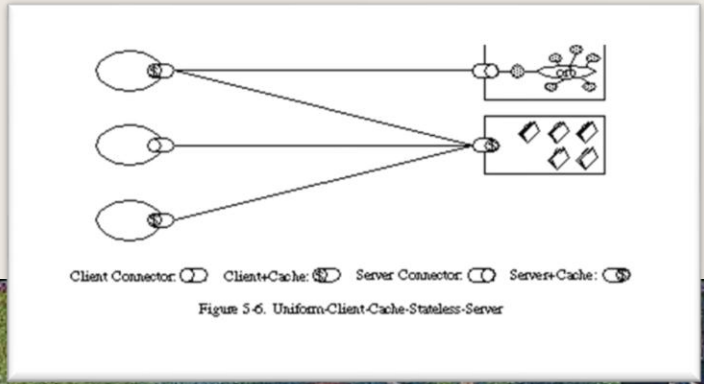
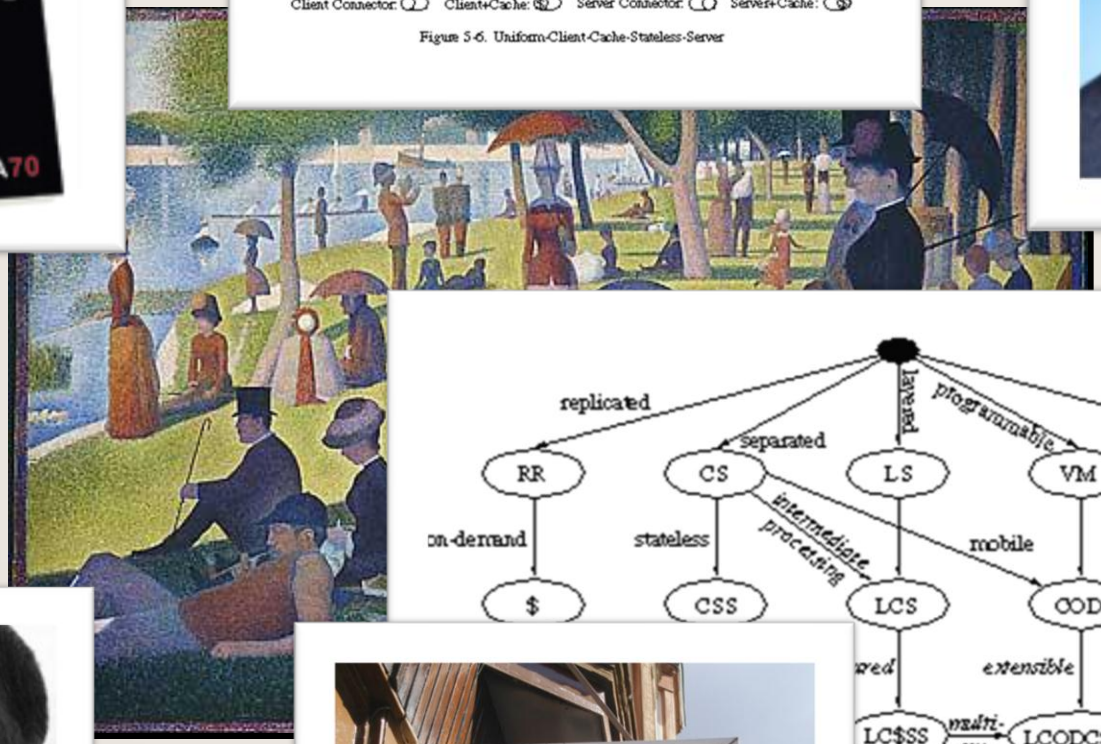
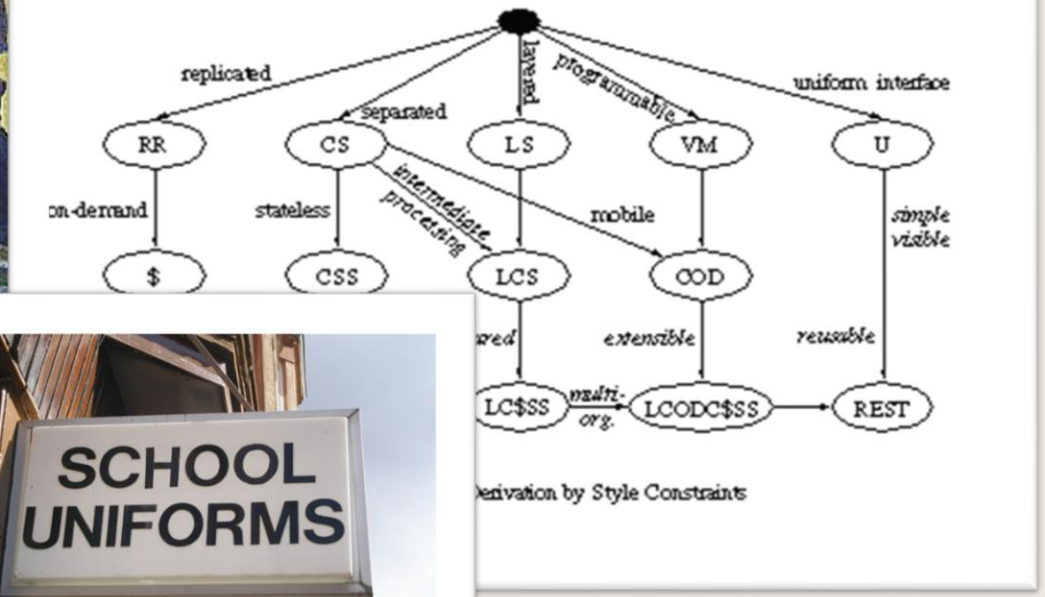
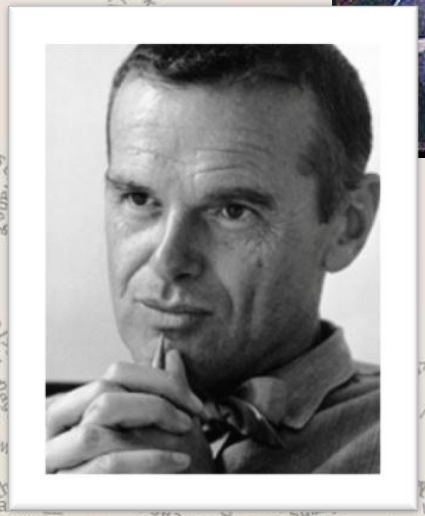


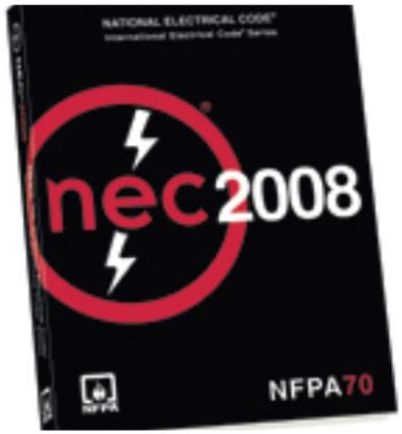
Figure 5-6. Uniform-Client-Cache-Stateless-Server





enivation by Style Constraints





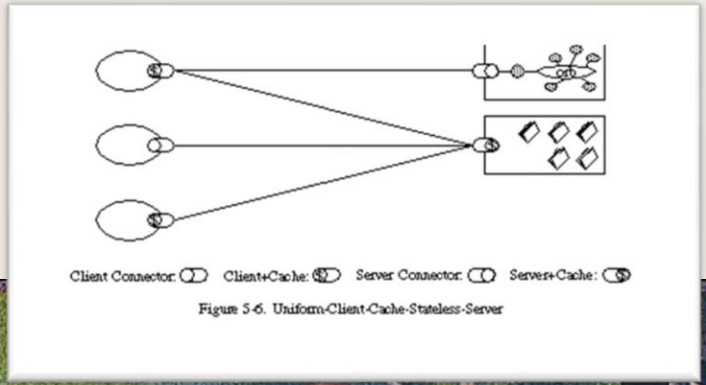
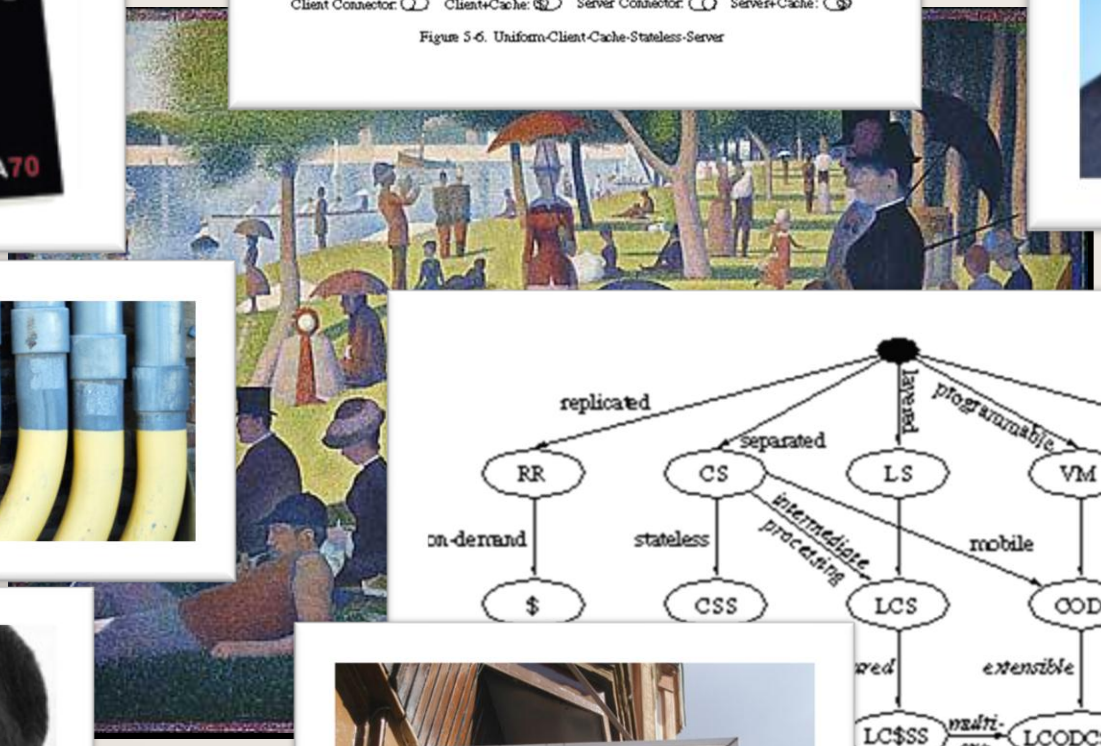
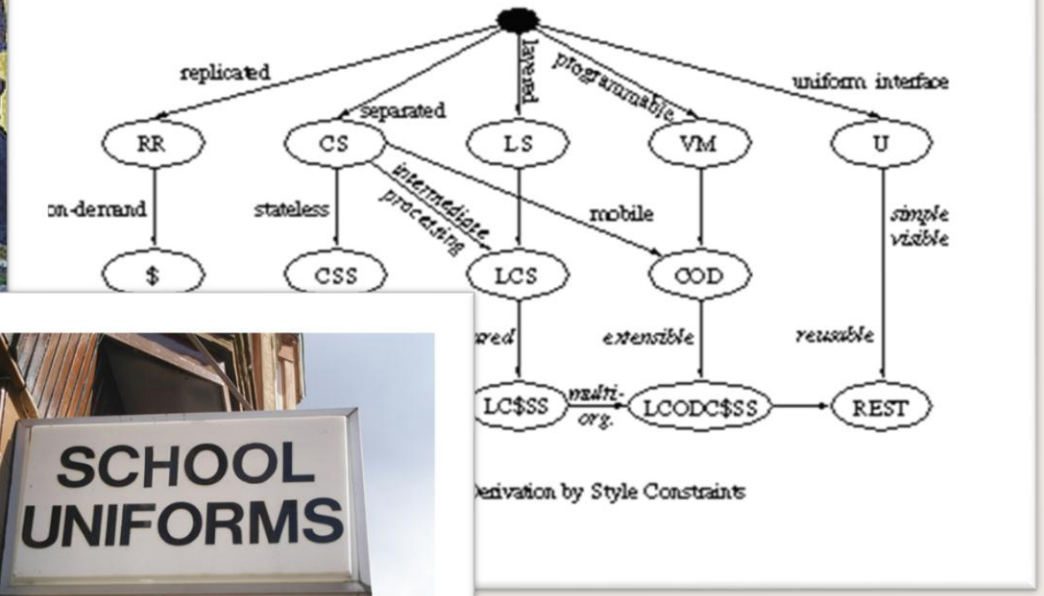
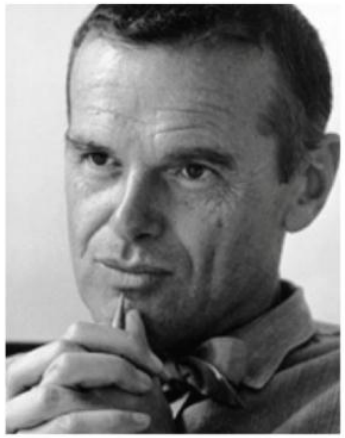


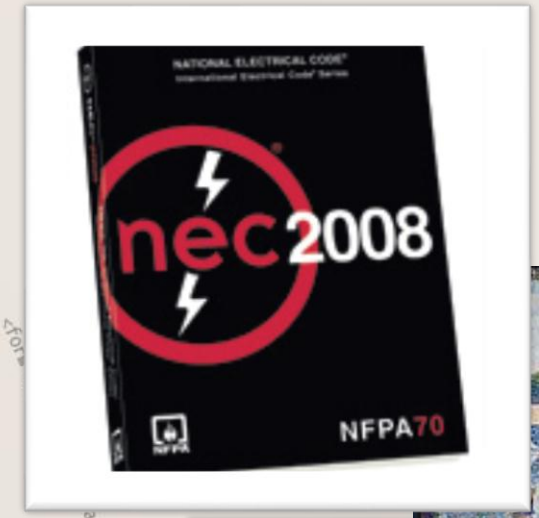
Figure 5-6. Uniform-Client-Cache-Stateless-Server





enivation by Style Constraints





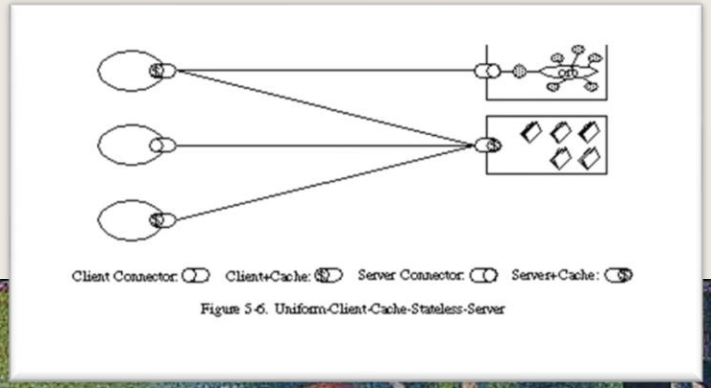
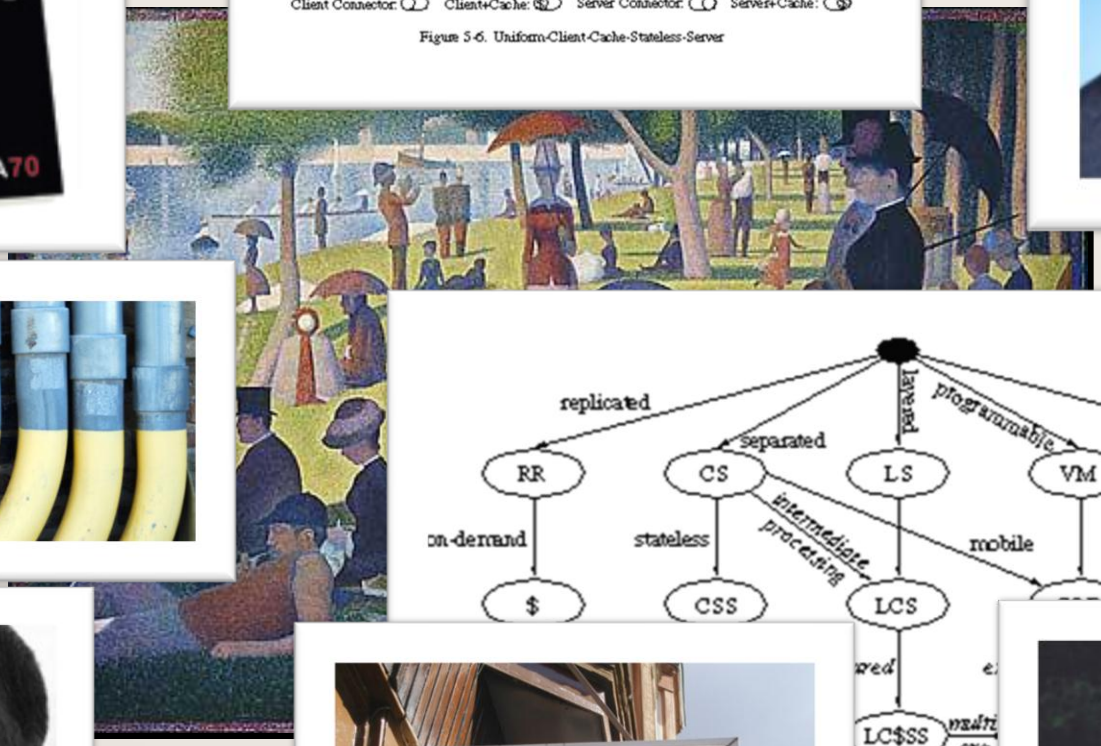
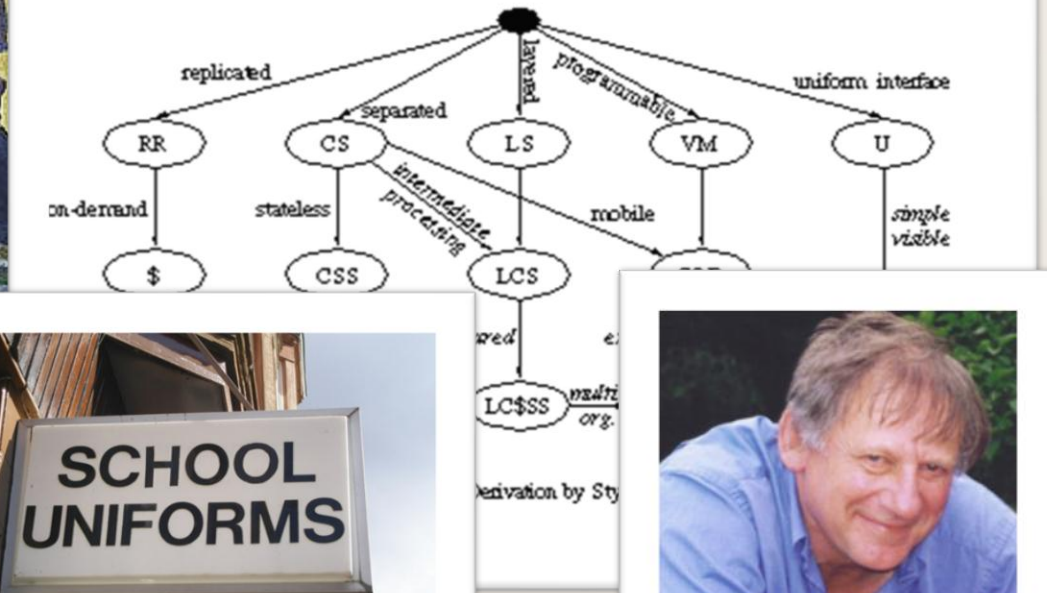
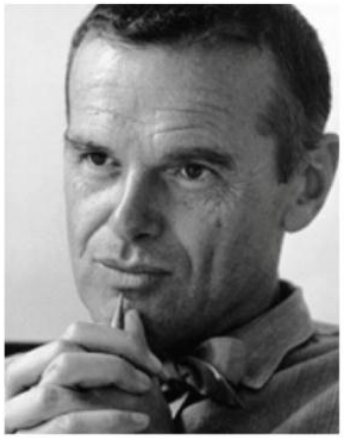


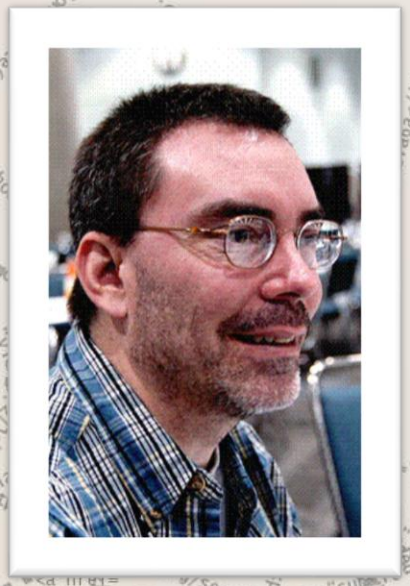
Figure 5-6. Uniform-Client-Cache-Stateless-Server







“Excuse me ...
did you say ‘knives'?”



City Gent #1 (Michael Palin)
The Architects Sketch

REST

Mike Amundsen
@mamund
amundsen.com