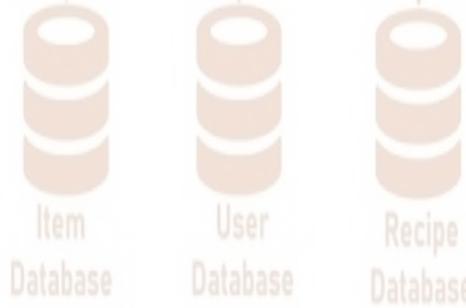


Application Architectures

Java Application Server

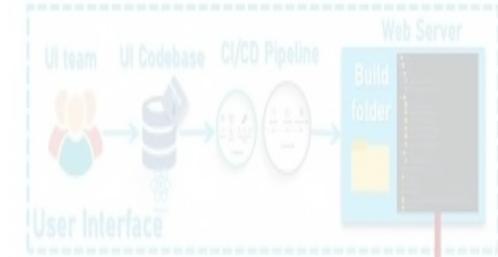


Web Server

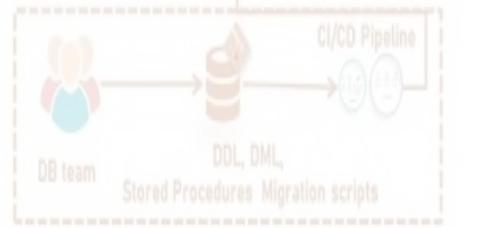
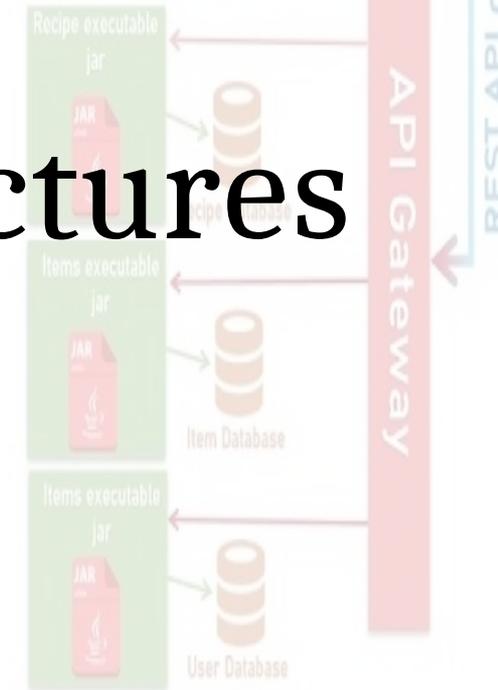


REST API calls

Java Application Server



Java Application Server



Layered structure

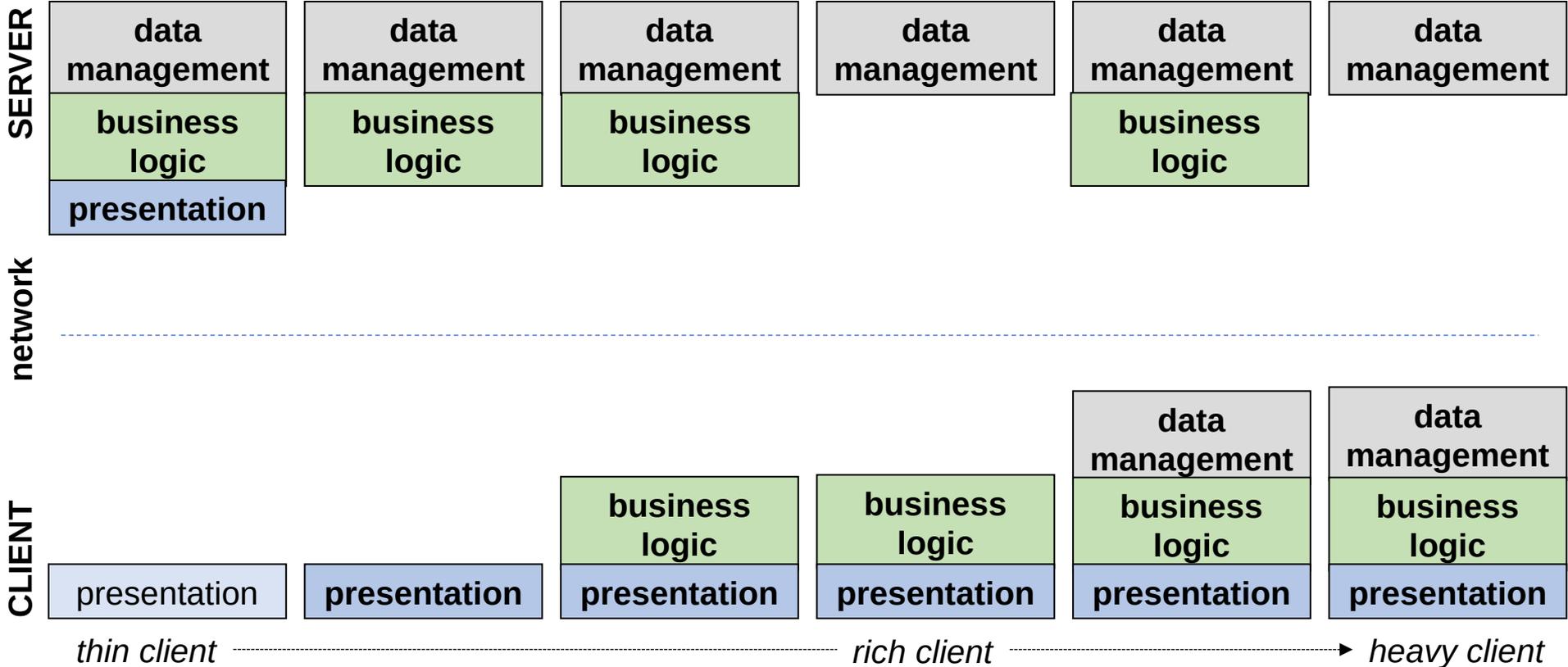
Division of the work of an application into 3 general functions, which can evolve independently:

- **Presentation:**
user input and commands, and display
- **Business logic:**
business objects, rules, processing logic, processes
- **Data:**
storage and logical access



Distribution onto « Tiers »

Distribution of the layers onto multiple machines (“tiers”) communicating over a network



Monolithic application

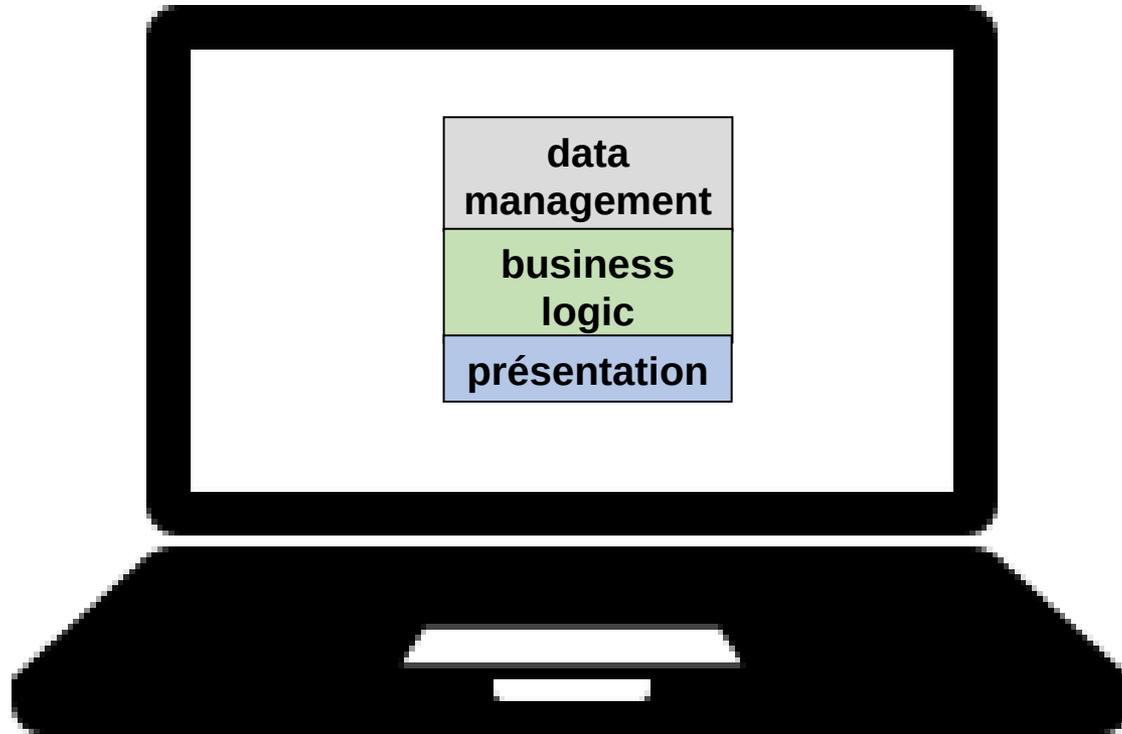
The 3 application layers are intimately interlaced in the same code base

```
import java.io.*;
public class ReadFromFile {
    public static void main(String[] args) throws Exception {
        File file = new File("C:\\Users\\galtier\\Desktop\\test.txt");
        BufferedReader br = new BufferedReader(new FileReader(file));
        String st;
        while ((st = br.readLine()) != null)
            System.out.println(st.toUpperCase());
        encrypt(file, "mySecretKey");
    }
}
```

The diagram illustrates the three application layers in a monolithic application. The code is enclosed in a light green box. Three colored boxes with arrows point to specific parts of the code: a grey box labeled 'data management' points to the File and FileReader objects; a blue box labeled 'presentation' points to the System.out.println call; and a green box labeled 'business logic' points to the encrypt call.

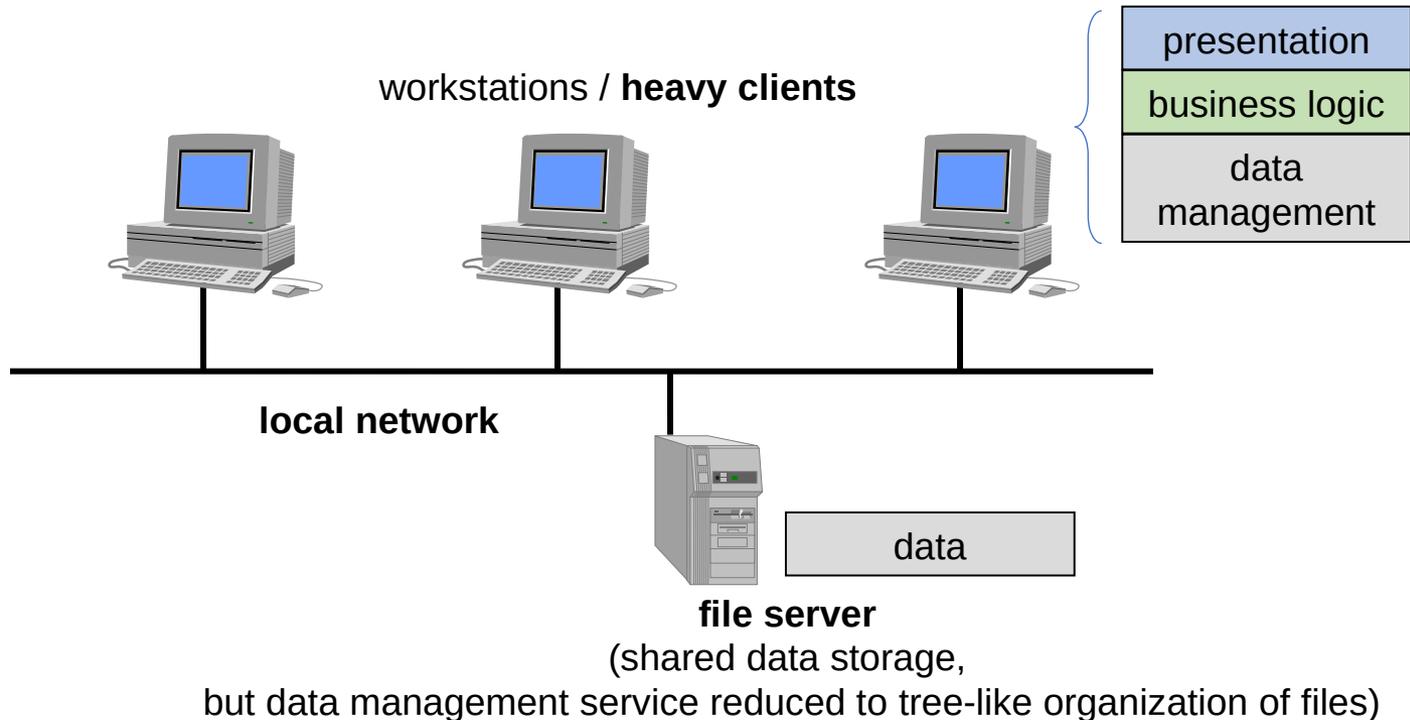
Single-tier Application

The 3 application layers run on the same computer



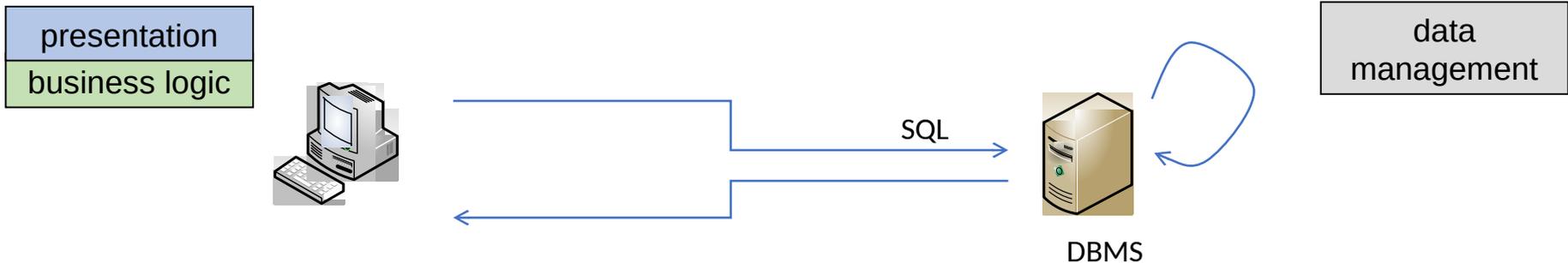
The origin: “1.5-tier” Architecture

- Development of LANs



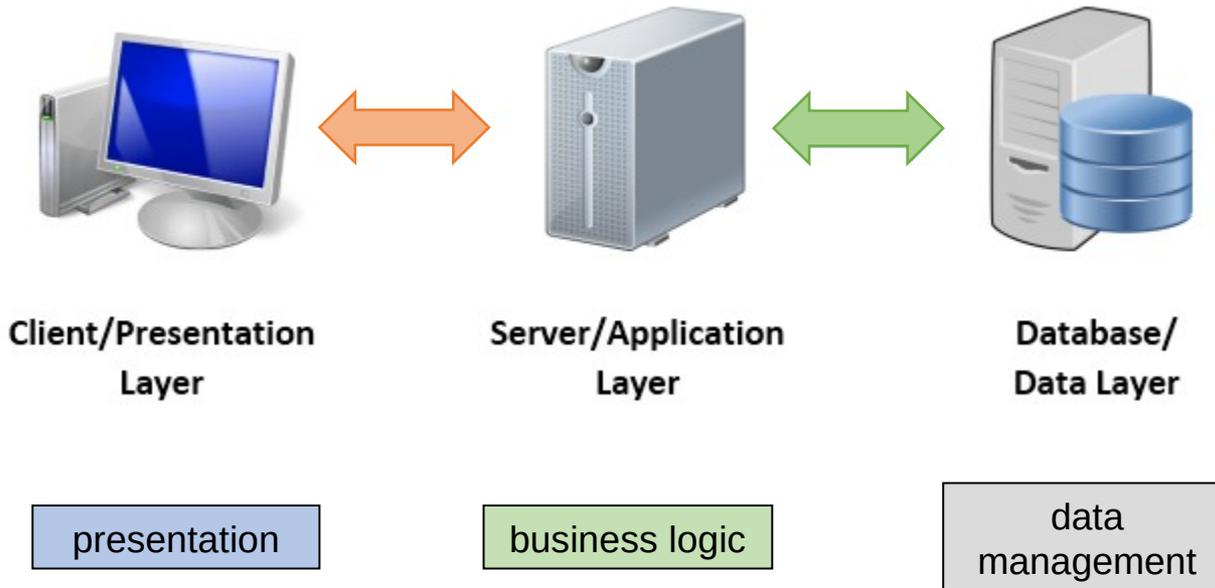
- Advantages: information sharing:
 - better communication
 - requires less resources

2-tier Architecture

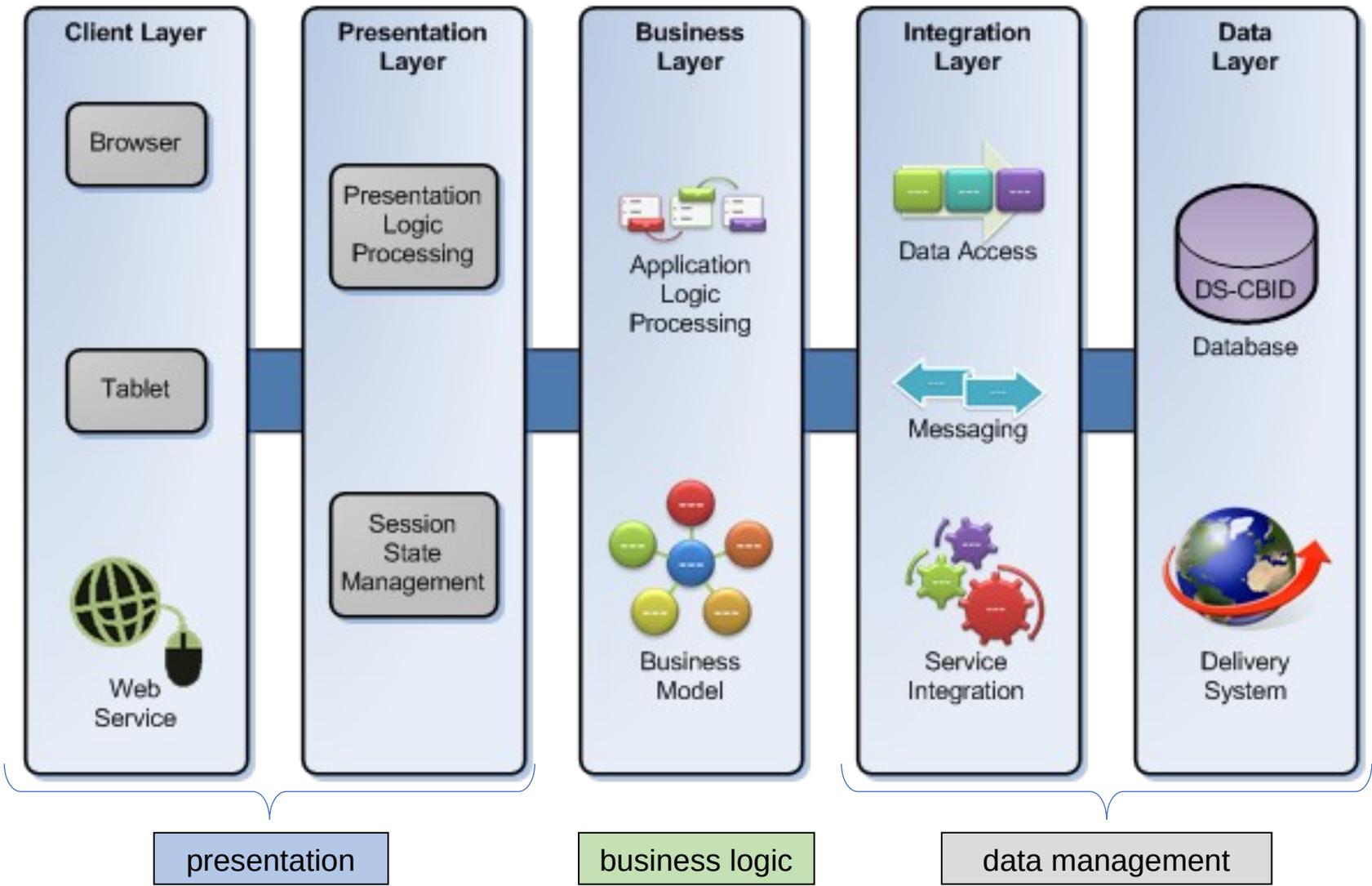


- Central database server
 - Manages physical I/O and provides logical data manipulation
 - Integrity control
 - Secure, optimized, transactional access
- Data handling is decoupled from its representation on disk, closer to the application logic

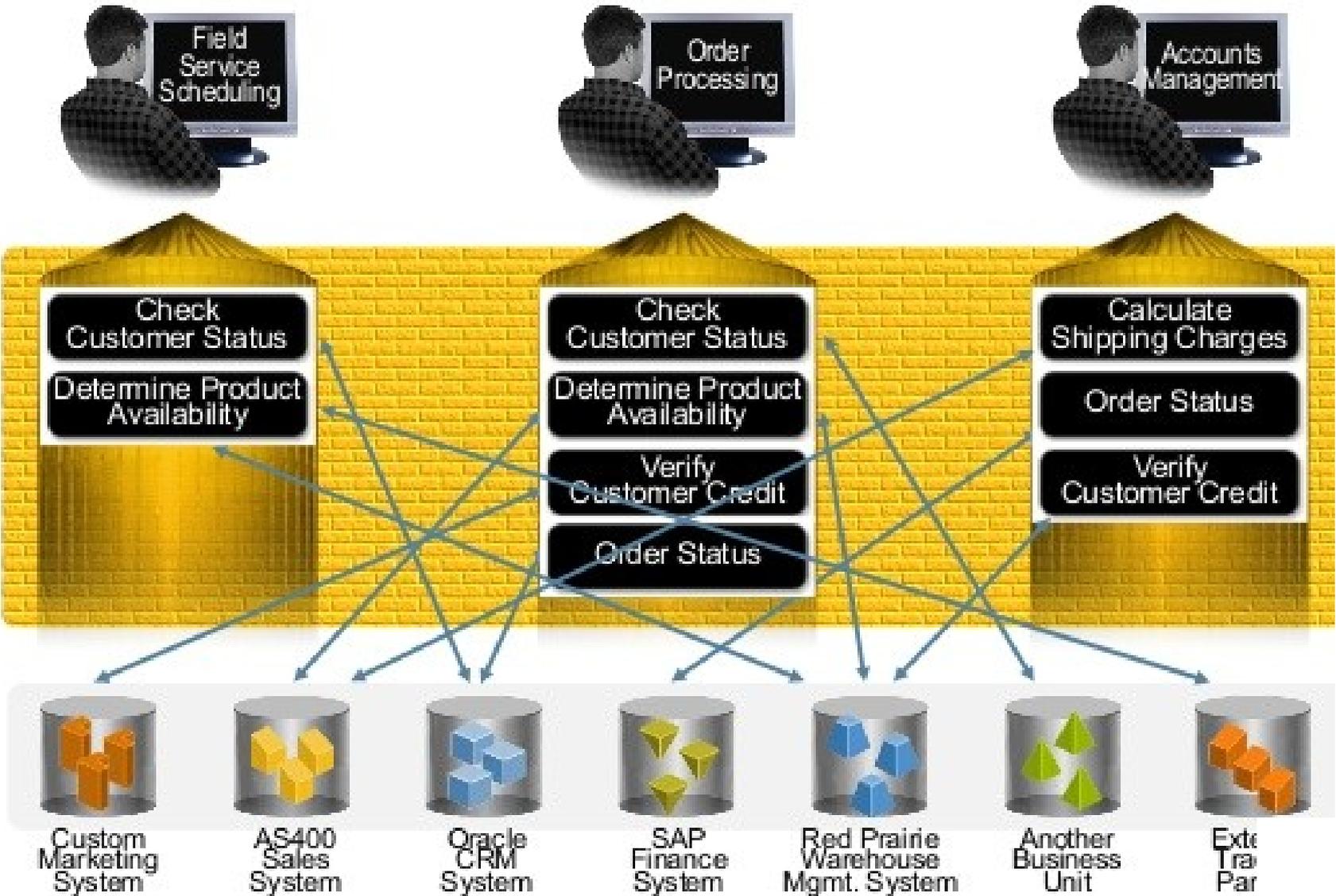
3-tier



4-tier, 5-tier



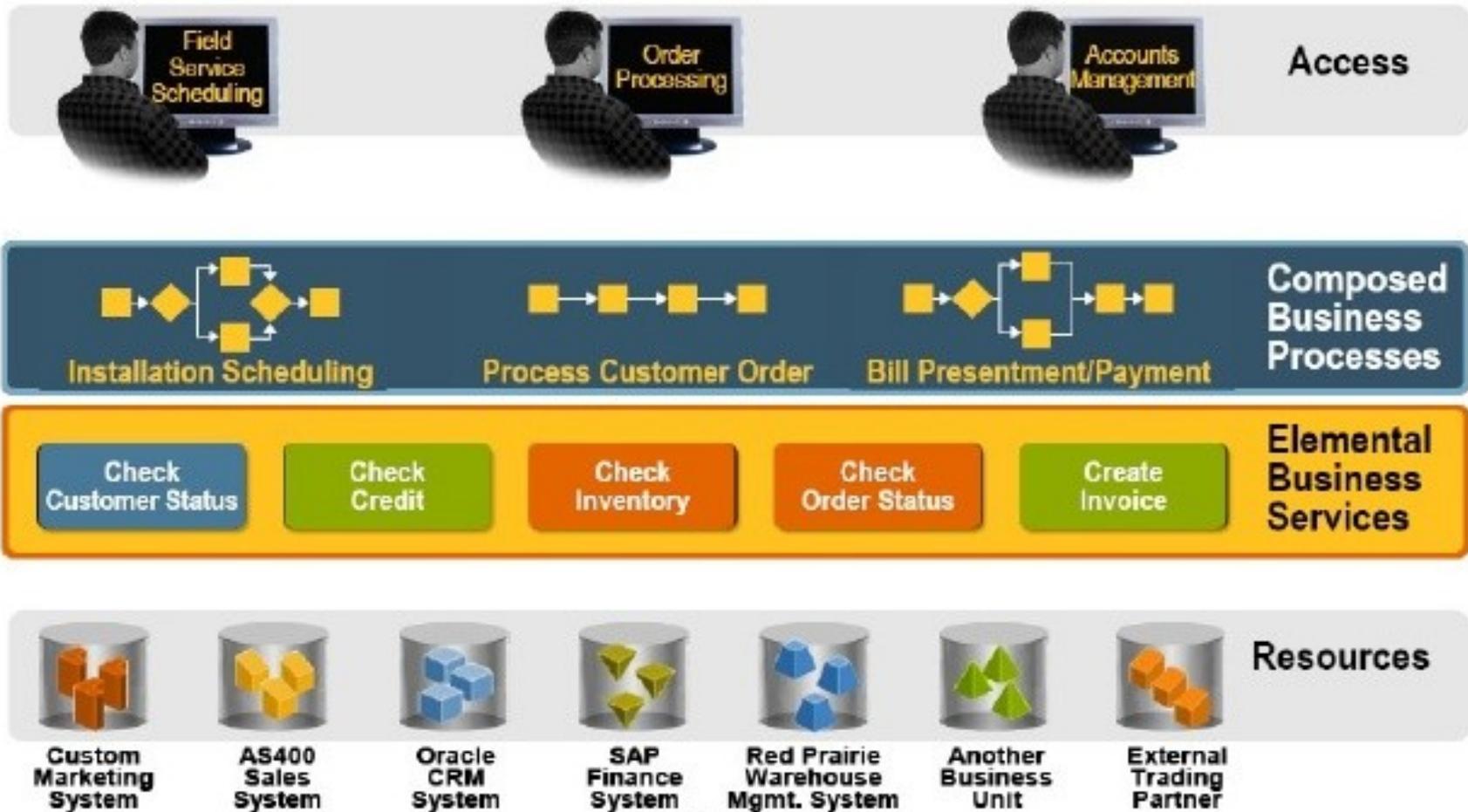
Siloed Architecture



Problems with siloed architecture

- Waste of resources
- Complex maintenance
- Lack of data sharing and consistency
- Complexity of IAM (Identity and Access Management)
- Difficult to scale up
- ...

Microservices Architecture



(Micro)Service Concept

- Black box performing 1 specific task (business or technical function)
- Can be used via an API (= contract between the customer and the supplier)
- Can call on other services
- Designed to be duplicated → *stateless*:
 - *No application state*
 - *Or client-specific state provided in the request*
 - *Or state on external storage shared with other services*

Advantages of the microservice architecture

- Reuse
- Scaling and fault tolerance thanks to easy duplication
- Fault isolation
- Independent development and deployment
- Ability to use the most appropriate technology for each module



- Small development teams

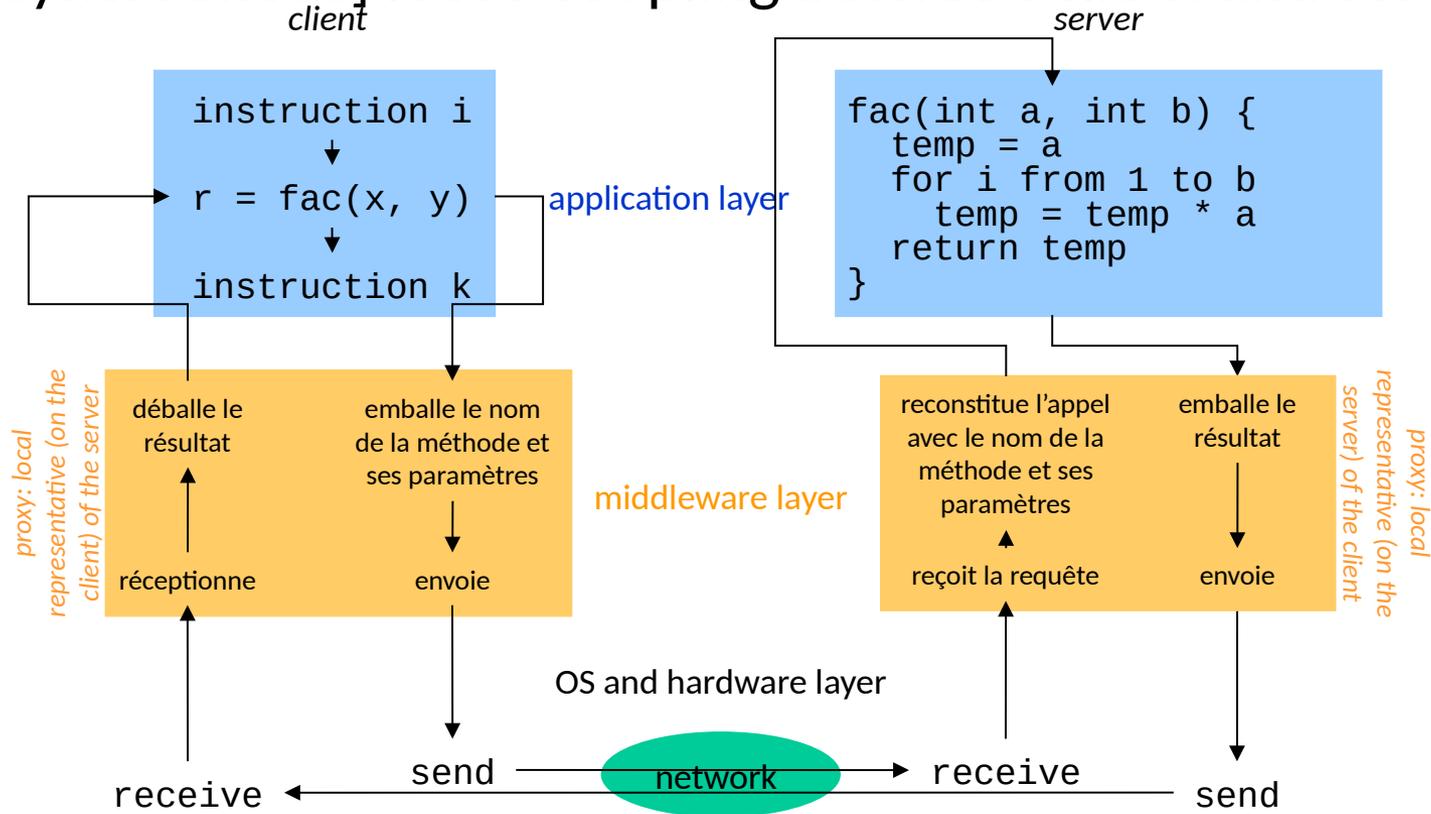
Middleware

Solutions to ease the connection between services:

- Locally:
 - Inter-process communication: system, MPI, Unix Domain Socket, etc
- Across the network:
 - Synchronous Remote Procedure Call
 - Asynchronous Messages

RPC (Remote Procedure Call)

- [asynchronous] loose coupling between client and server



- The proxies handle:
 - network calls
 - format transformations between the client and server

Service call

- 1st generation Web Services:
 - Requests and responses transported by SOAP messages, usually on top of HTTP
 - 4 patterns supported by WSDL:
 - Request - response
 - One way request
 - Notification
 - Request - response
 - WS-*: myriad of specifications to complete the messaging service
- Web service in a REST architecture:
 - URI-addressed resources
 - Requests and responses typically carried over HTTP, exploiting the semantics of HTTP methods

Web Service: Generic Definition

- Software function (tool, resource, data...)
- Accessed via the network (remote, deployed, @)
- Offered to other software units (M2M)
- Platform- and language-independent

- Can be described and advertised

- 2 roles:
 - Service requester = client
 - Service provider = server

What is an API?

- a means of exposing business/enterprise resources via the Internet to external or internal software consumers
 - Well-defined interface: contract
 - Easily accessible by third parties
 - Use of standard protocol(s)

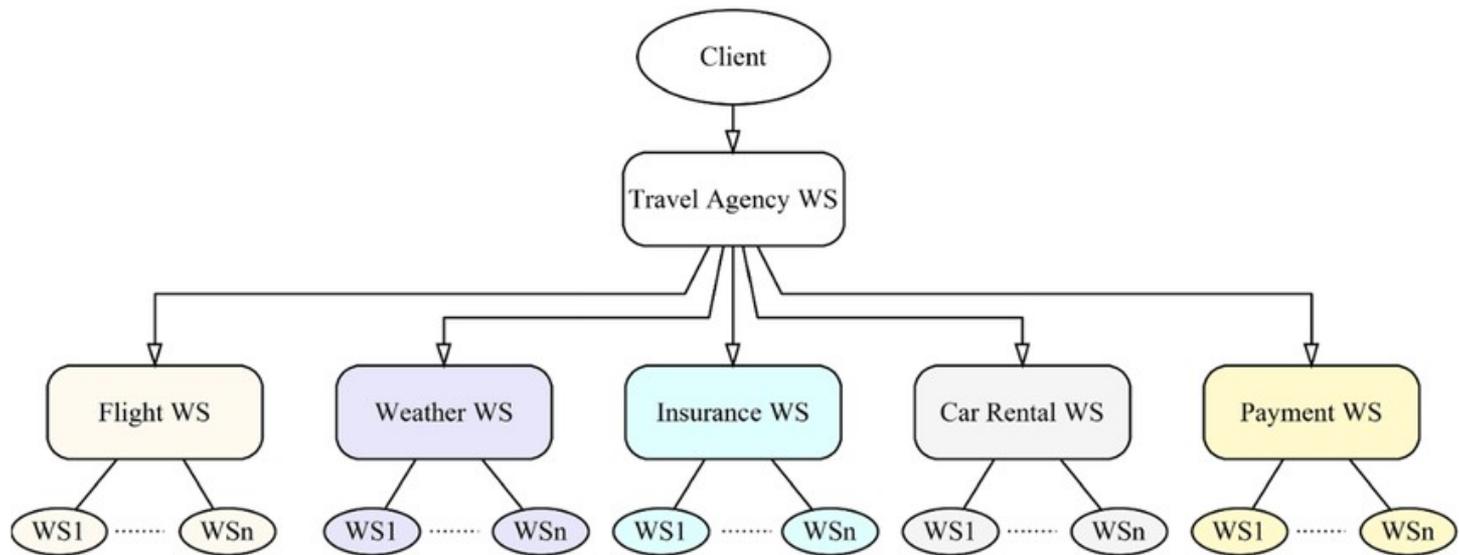
≈ Web Service

Usage

- Services are used as software libraries to build applications
- 2 contexts:
 - External services
 - Internal services

External Services (Partner or Public Services)

- open to the partners of the organization (B2B, B2C)

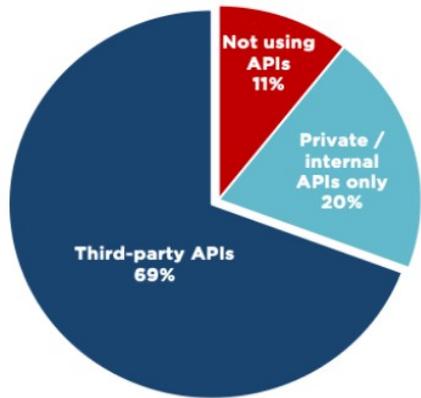


API becomes more of a priority than UI

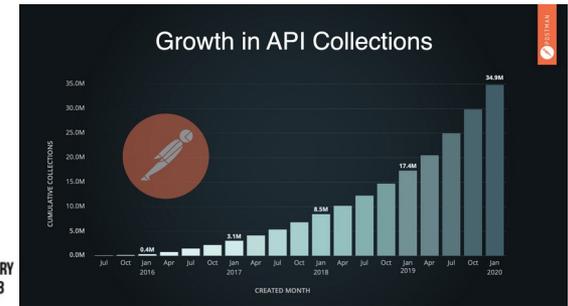
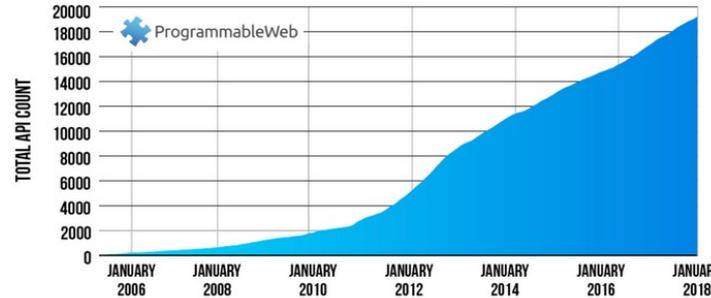
Nearly 90% of developers use APIs

% of developers (Q3 2020 n=15,299)

<https://nordicapis.com/apis-have-taken-over-software-development/>



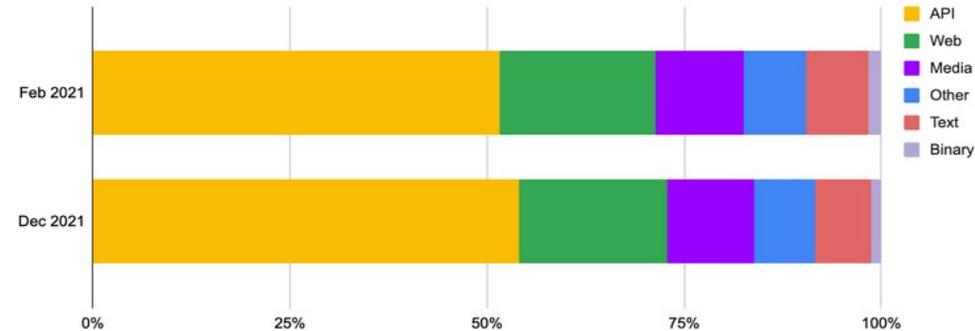
Source: SlashData Developer Economics survey 19th edition / DATA



Cloudflare traffic: API use in 2021

<https://blog.cloudflare.com/landscape-of-api-traffic/>

Traffic composition by content type



Programmatic access is considered at least as vital as human access, if not more so.

<https://www.postman.com/state-of-api/>



API-First As The Norm

Adopting an API-first strategy will be increasingly common in the future. In fact, Postman's 2021 found that 39.2% of teams have already designed and defined APIs and schema before they ever

Benefits of Web Services from the Client's Point of View

- Take advantage of third-party data or programs without having to:
 - develop, test, update and maintain code
 - acquire and maintain a hosting infrastructure
- Easily compose services and replace one component by an alternative

Trade-offs for the Client

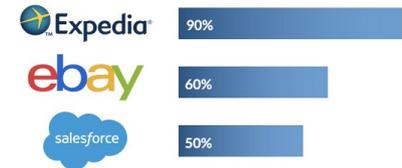
Developers lost control of the services and the services are remote →

- A service might be temporally unavailable
- Performances might become poor
- Data of the client can get lost, divulgated, corrupted...
- A service might not longer be maintained
- The service fee might increase
- ...

From the Service Provider's Point of View

- Benefits
 - Increases revenue
 - Extends customer reach
 - New form of marketing : B2D “business to developer”
 - Stimulates innovation
- Risks
 - Decreases ad revenue
 - No more control on the final user's experience

Percentage of Revenue Generated Through APIs

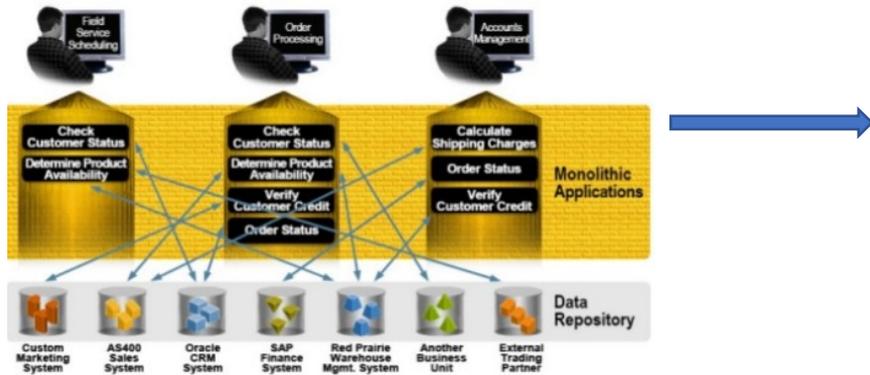


Source: Harvard Business Review, The Strategic Value of APIs, 2015.

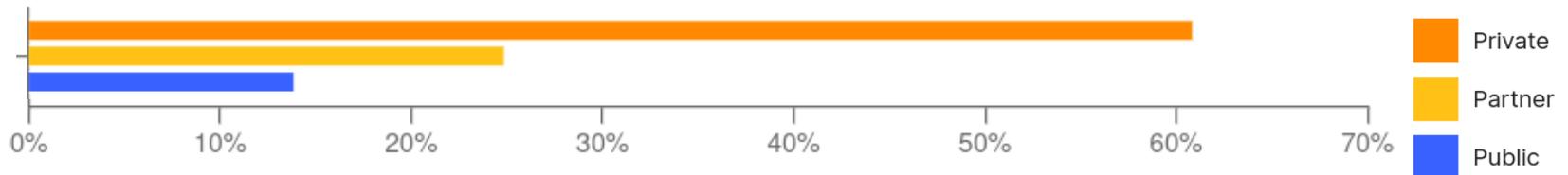
Internal Services (Private Services)

- access restricted to the organization

Monolithic Systems



Reuse Services via Re-composition



Internal Services: an injunction!

- Jeff Bezos's mandate (2002)



1. All teams will expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.
3. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

2 “Flavors” of Web Services

Process-Oriented Services

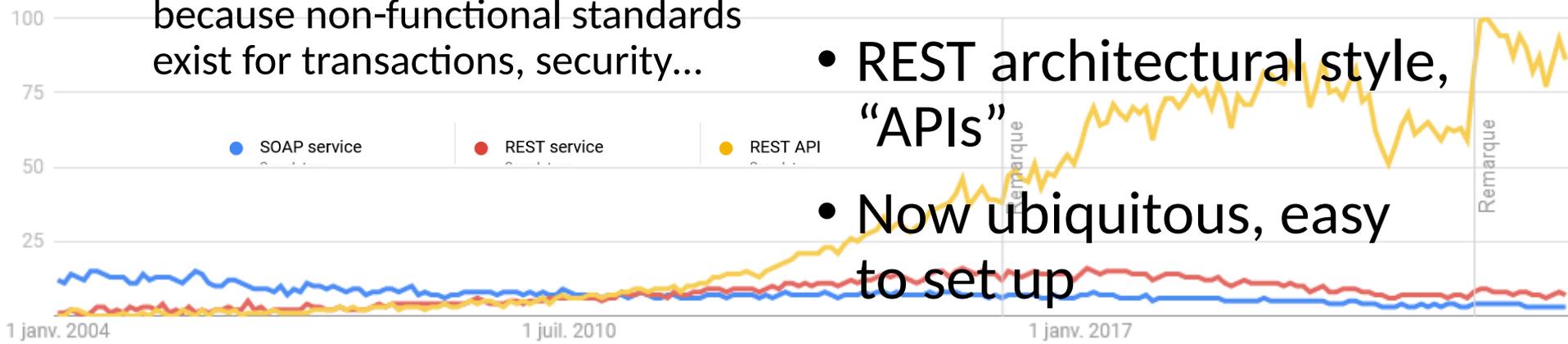
- Distributed Information Systems required middleware, RPC and Object Brokers, were poorly adapted to B2B → use web protocols for transport and XML as IDL and format
- SOAP + WSDL standards
- “first-generation” web services, still used for complex applications because non-functional standards exist for transactions, security...

Resources-Oriented Services

- From static web pages to dynamic web pages to web applications (UI = web browser, business processes are executed on the server) to web services

- REST architectural style, “APIs”

- Now ubiquitous, easy to set up

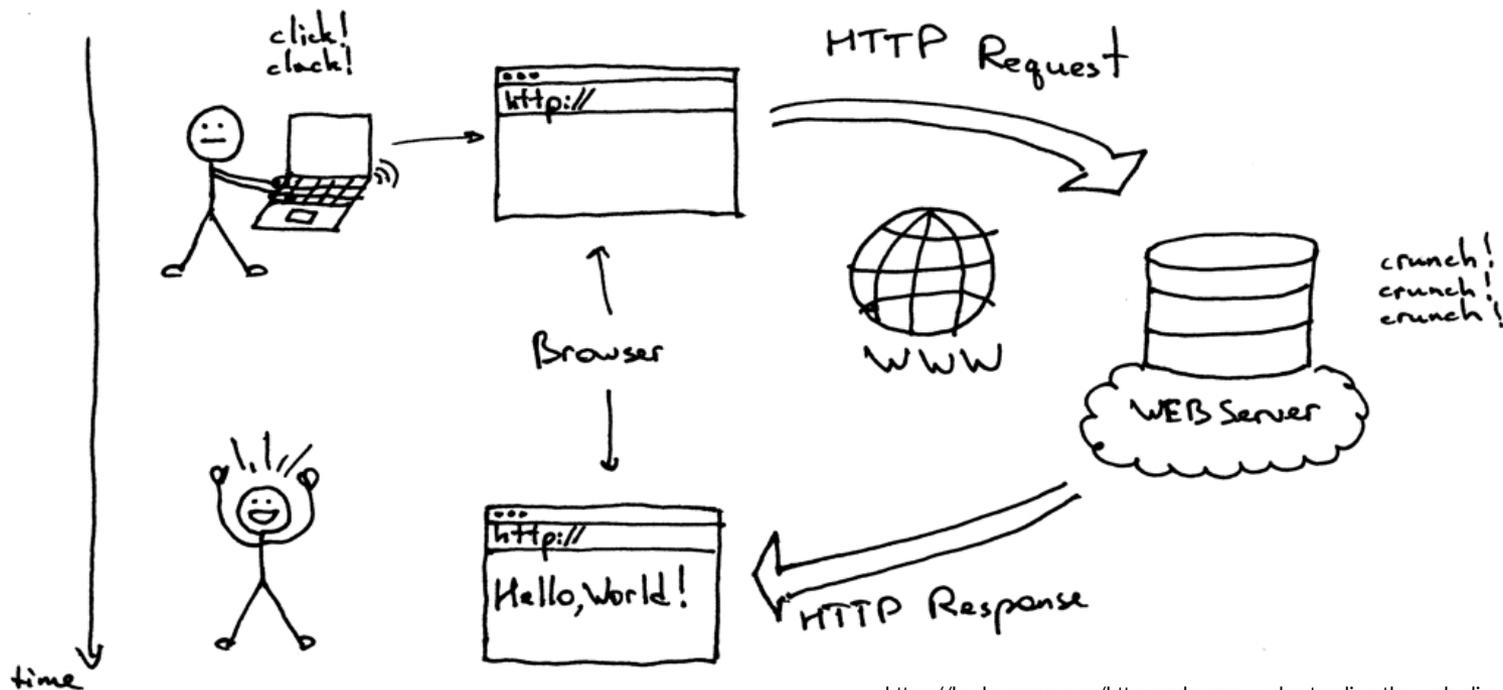


<http://www.>



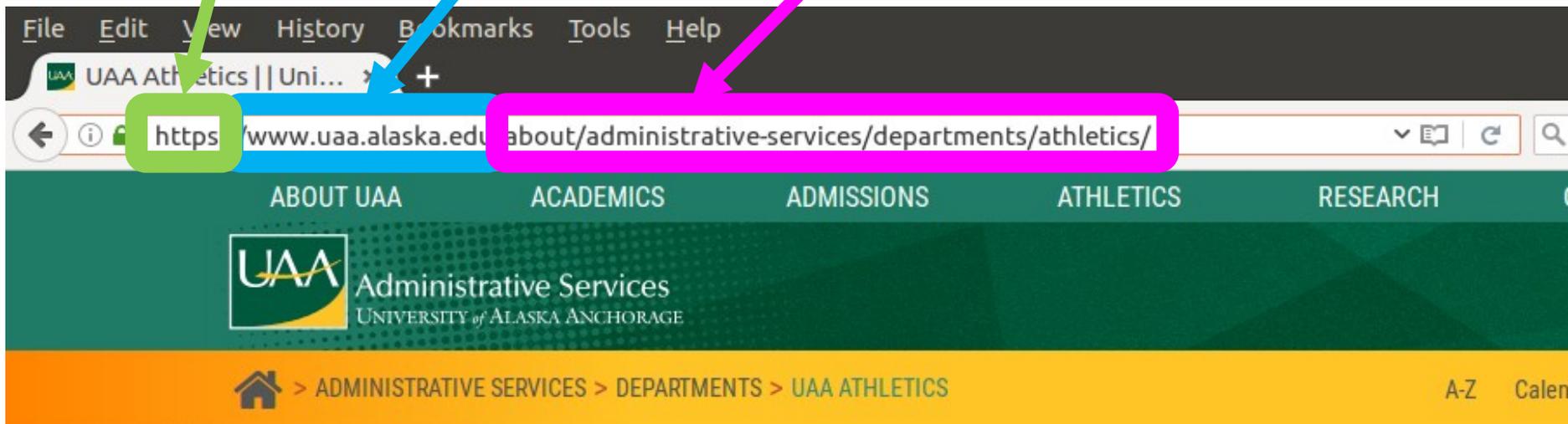
The Web Services Protocol

- Application-layer protocol
- Client sends service requests using HTTP messages
- Server replies using HTTP messages



Resources = addressable files

- Any kind of file: HTML file, JPG image file, binary file...
- URL (Uniform Resource Locator)
= protocol + server host name + path on server



UAA Athletics

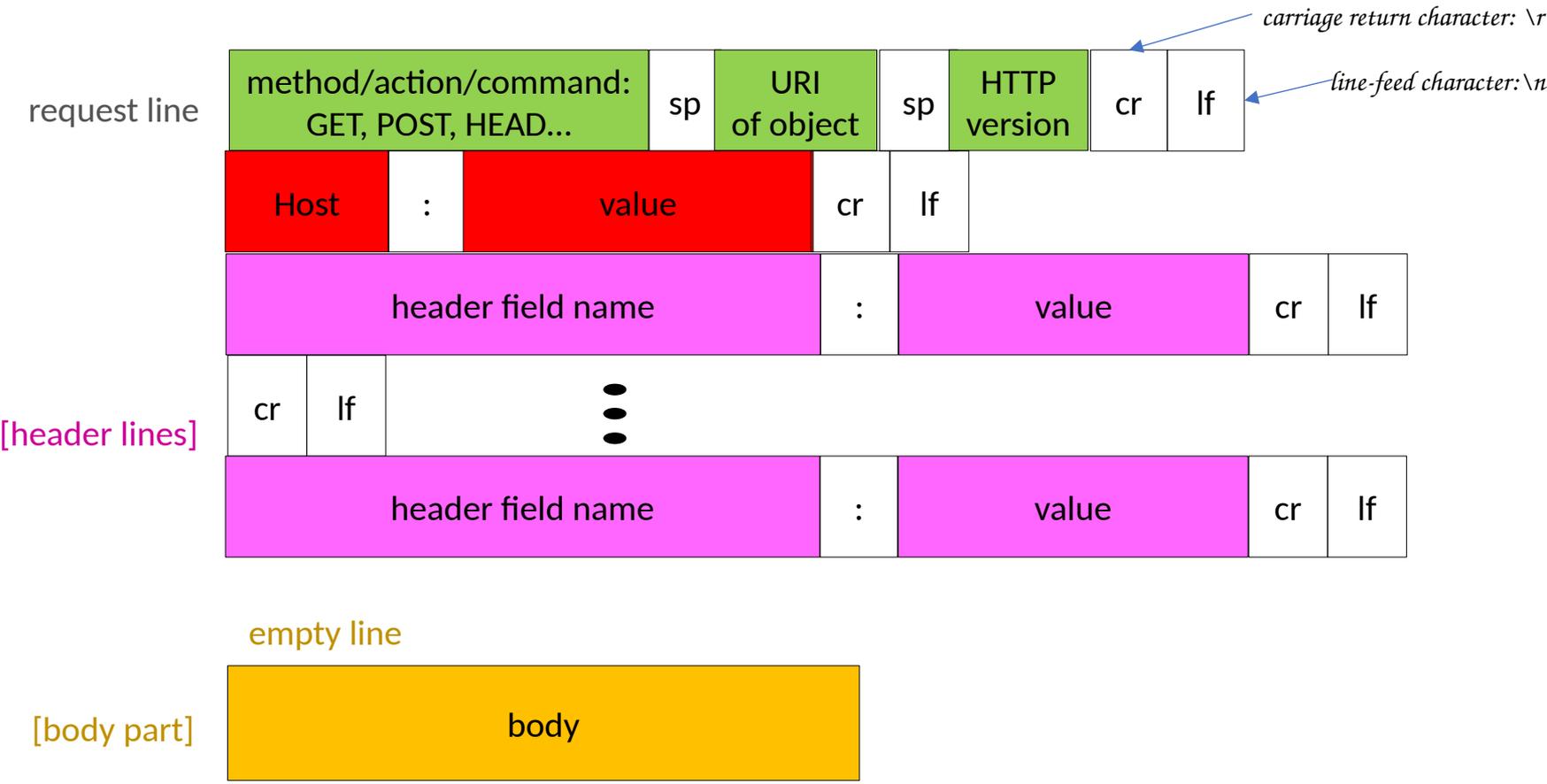
HTTP Messages

- 2 kinds of messages
 - Request
 - Response
- In ASCII (HTTP 1.x)

HTTP Requests Commands

- GET
 - retrieves an object
 - no request body
- HEAD
 - same response as GET but empty response body (used to test the access to or the "freshness" of the object without actually downloading it)
- POST
 - results in the creation of a new resource on the server
 - usual request: contains data
 - usual response: URL of the created resource
- PUT
 - updates an existing resource
 - request usually contains data
- DELETE
 - deletes a resource

HTTP Request Format



HTTP GET Request Example

GET /node/44 HTTP/1.1\r\n

Host: mapi.centralesupelec.fr\r\n

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n

Accept-Language: en-US,en;q=0.5\r\n

Accept-Encoding: gzip, deflate\r\n

Connection: keep-alive\r\n

\r\n

HTTP POST Request Example

POST /post.php HTTP/1.1\r\n

Host: posttestserver.com\r\n

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n

Accept-Language: en-US,en;q=0.5\r\n

Accept-Encoding: gzip, deflate\r\n

Content-Type: text/xml\r\n

Content-Length: 27\r\n

Connection: keep-alive\r\n

\r\n

firstname=John\nlastname=Doe

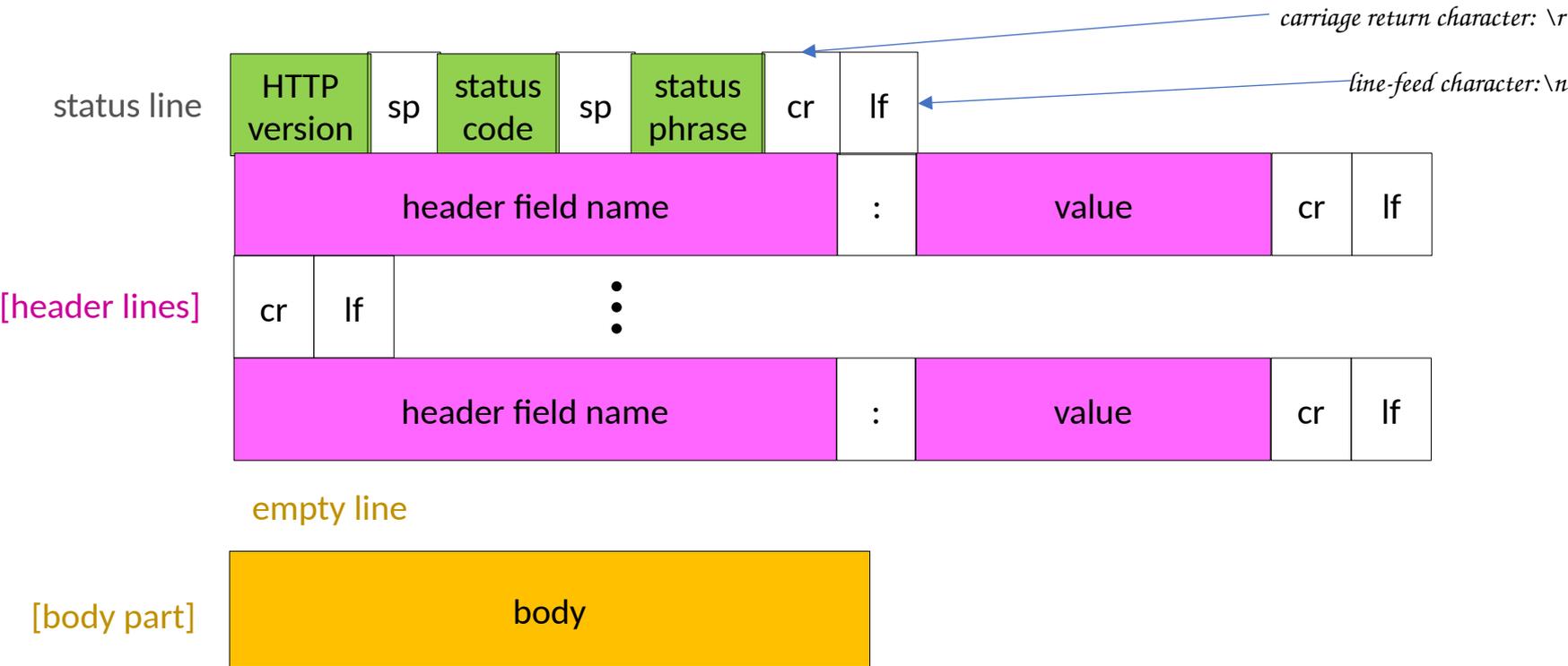
Request Parameters

3 symbols to add parameters to an URL:

- **?** concatenates the URL and the string of parameters
- **&** separates multiple parameters
- **=** assigns a value to a parameter

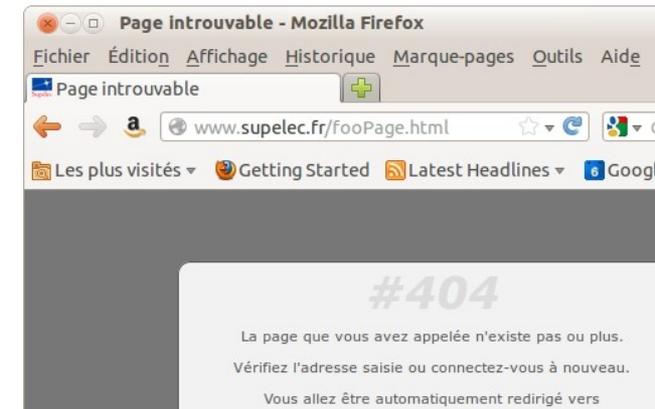
GET /products?priceMin=10&priceMax=40

HTTP Response Format



Status Codes

- 2xx: success
 - 200 OK
- 3xx: further action required
 - 301 Moved Permanently: the new URL is specified in a header field
- 4xx: client error
 - 400 Bad Request: badly formulated query
 - 404 Not Found: object does not exist on the server
- 5xx: server-side error
 - 505 HTTP Version Not Supported



HTTP Response Example

HTTP/1.1 200 OK

Date: Wed, 01 Feb 2017 12:48:22 GMT

Server: Apache/2.4.10 (Debian)

[...]

Content-language: fr

Content-Encoding: gzip

Content-Length: 4740

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html; charset=UTF-8

.....;r.8...W...-{.(.KO...!K..-.\$..q;.(...D.4..T.v.\.....q.{...Z.2_....b.....(l....Df" ..Hn... |....}.WQ..m....
WD.sR)..J.....L:9.C..MC...X.I...
J..(...'"J. D....d%bN,. \$..Y.....z.....y(.MS....#.qV.....>.9.j.0
s&...v.M...'').....m8..<=.i..%B.....S.x}.J.:V..{." .HM..4b..!YJ.....X{i...l.;.TX}....N.r .<d...#.....S..
..#Oa. ...V..EPj..G...A..D.K...Z1..c.h,b.4..b.3...l.6..La..>.L8#l.U.\.....2..y!...S.,.....%.....>..ID...
N..^

HTTP Response Example

HTTP/1.1 404 Not Found

Date: Wed, 01 Feb 2017 13:14:55 GMT

Server: Apache/2.4.10 (Debian)

[...]

Content-language: fr

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html; charset=UTF-8

305d

```
<!DOCTYPE html>
```

```
<html lang="fr" dir="ltr" prefix="content: http://purl.org/rss/1.0/modules/content/ dc: ht
```

```
<div id="block-zircon-content" class="block block-system block-system-main-block">
```

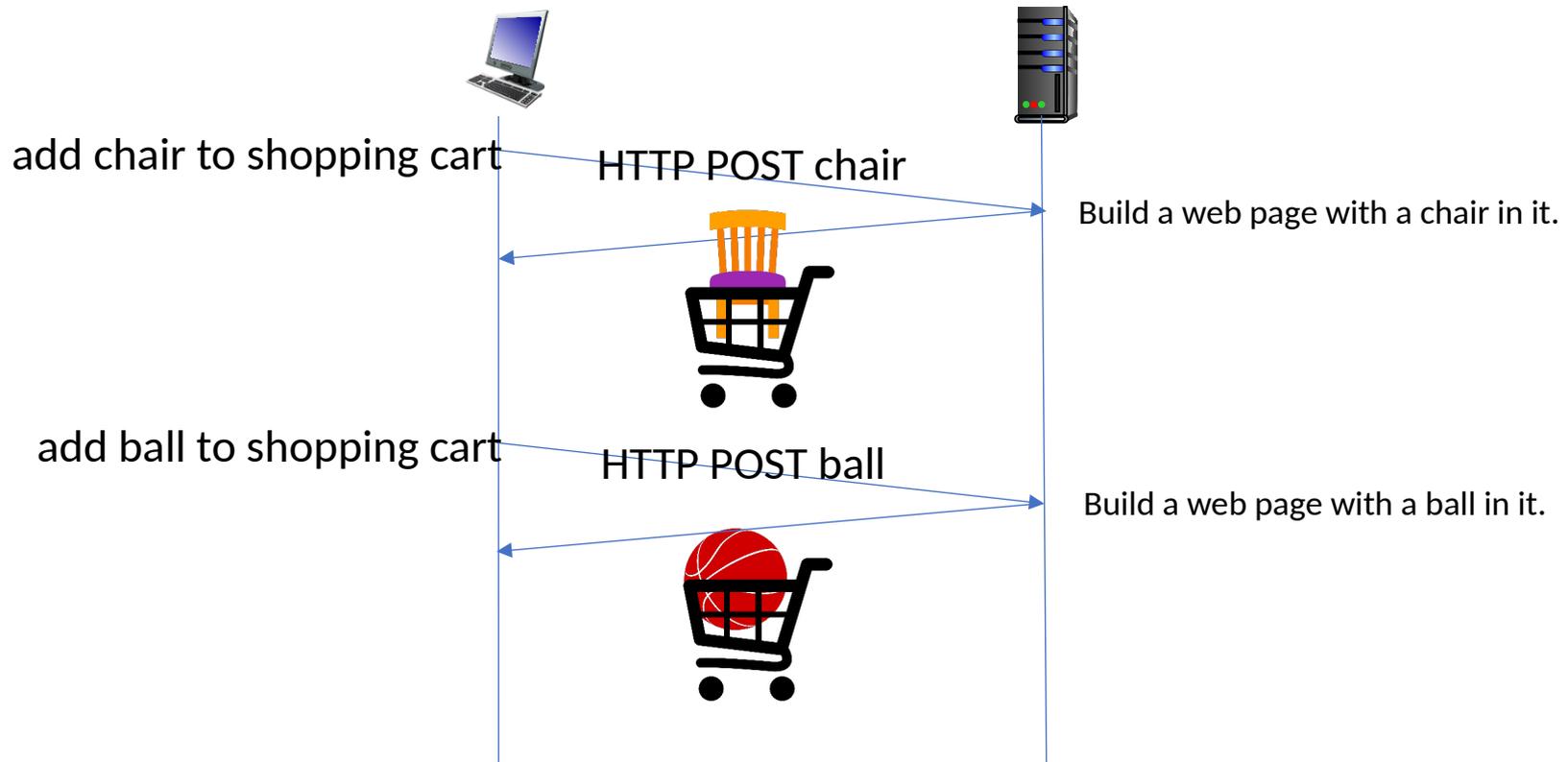
```
  La page demand..e n'a pas pu ..tre trouv..e.
```

```
</div>
```

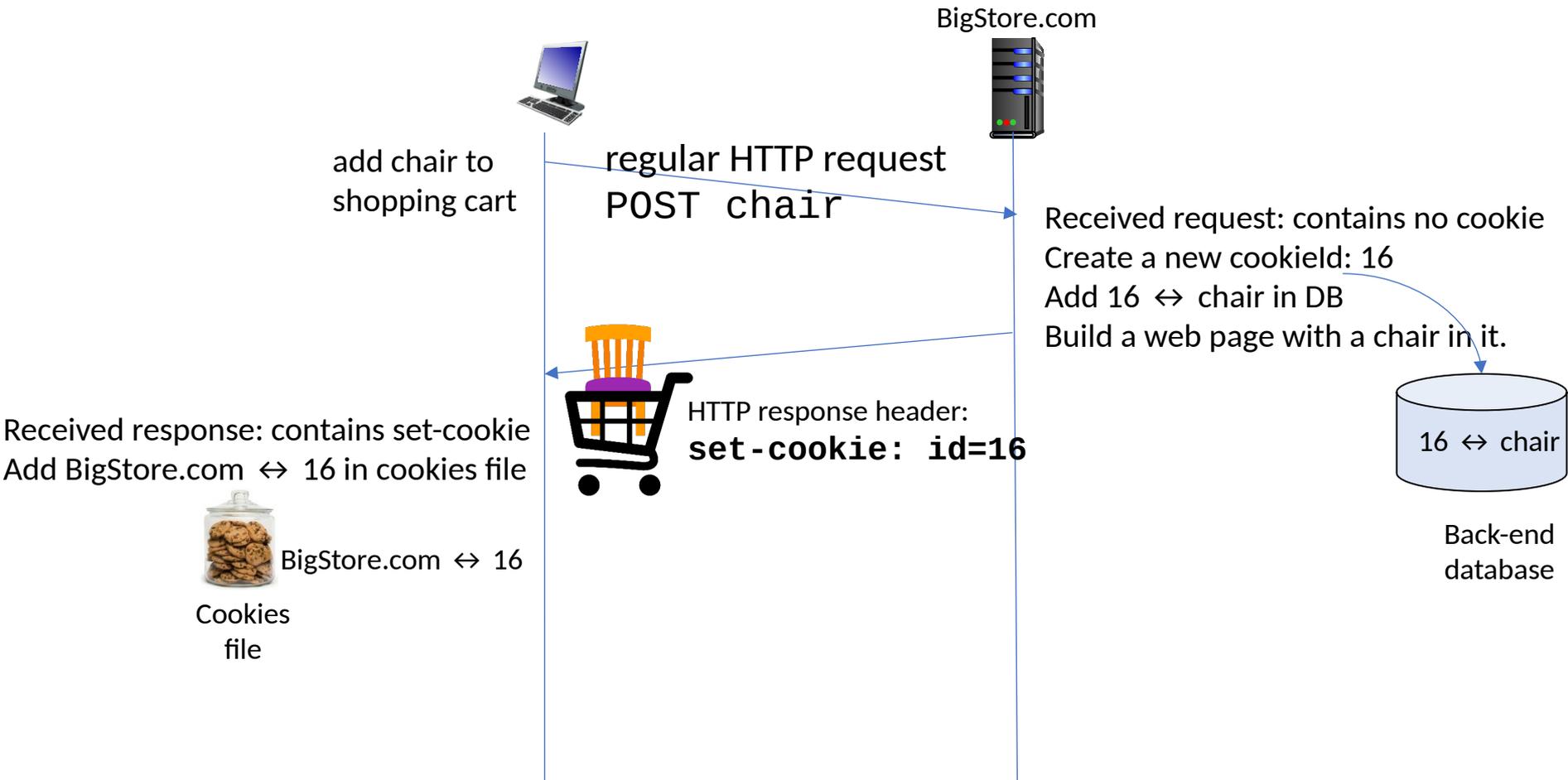
HTTP Server is Stateless

- A stateless protocol does not require the server to retain information or status about each user for the duration of multiple requests.
- Successive requests from a given client to a server are not treated as a chain but rather as separate requests, independent from the previous ones.

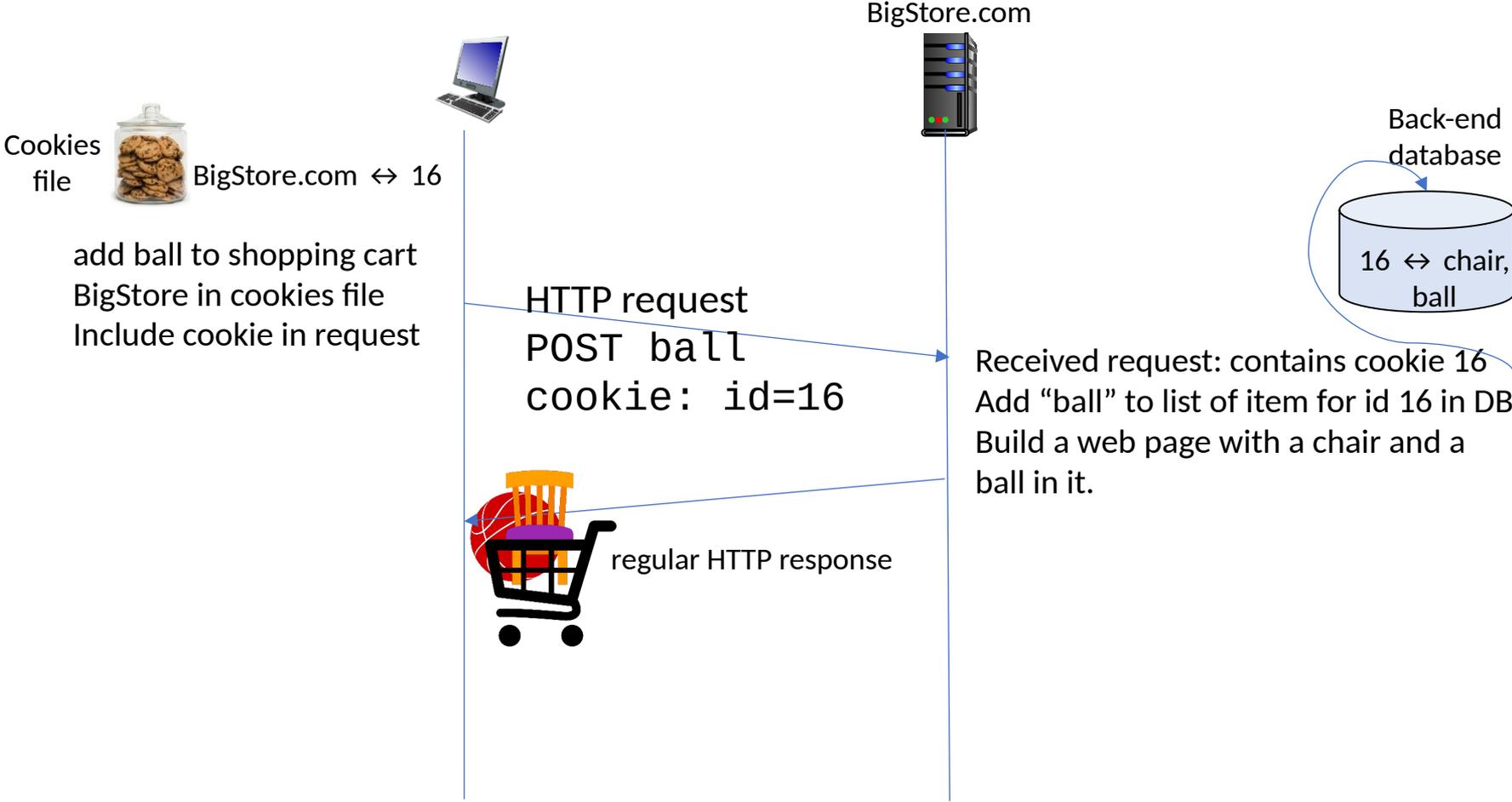
What we get is not what we want.



Cookie Example



Cookie Example



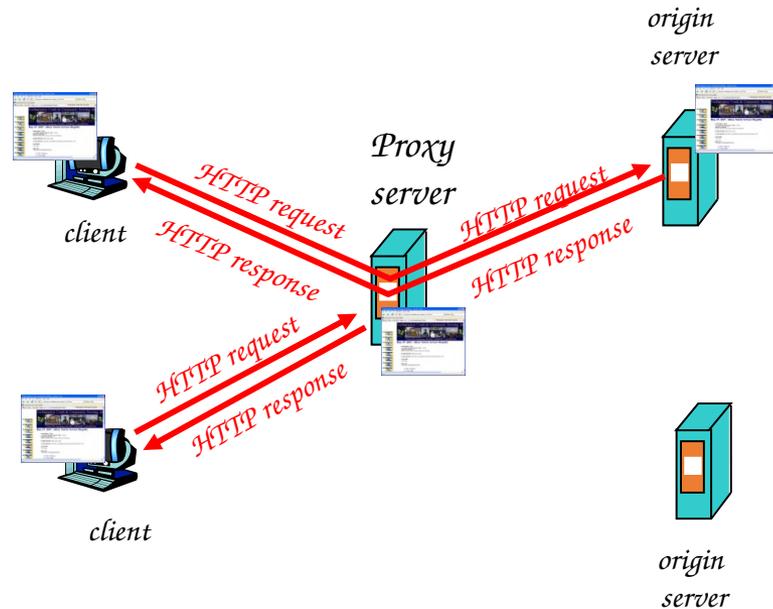
Uses

create a user session layer on top of stateless HTTP

- content adaptation (recommendation based on previous visits etc.).
- shopping carts (e-business)
- session definition at application layer (Web mail)
- authorization
- ...

Web Cache (proxy server)

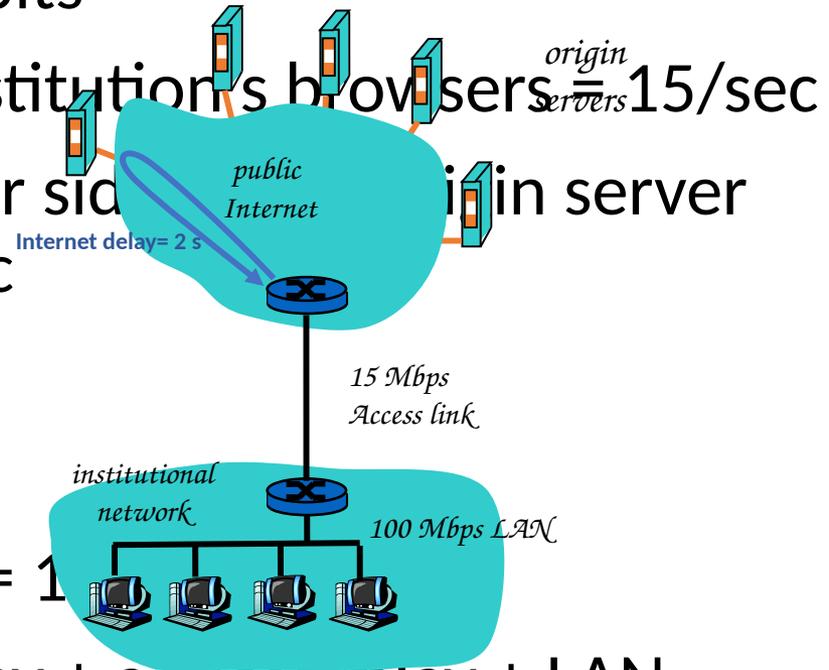
- to satisfy the requests without involving the real server
- browser must be configured to send all HTTP requests to cache
- reduced traffic on Internet, improved response time



Caching Example

assumptions

- average object size = 1Mbits
- avg. request rate from institution's browsers = 15/sec
- delay from Internet router side to origin server and back to router = 2 sec



consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + milliseconds

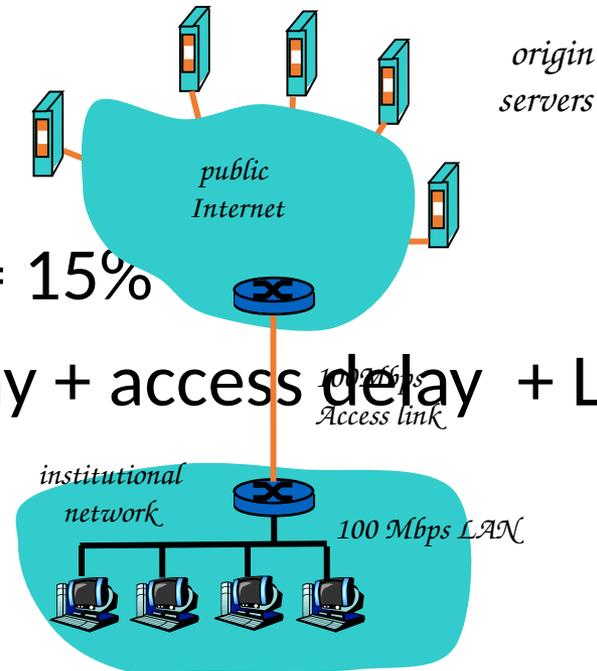
Caching Example (cont)

possible solution

- increase bandwidth of access link to 100 Mbps

consequence

- utilization on LAN = 15%
 - utilization on access link = 15%
 - total delay = Internet delay + access delay + LAN delay
= 2 sec + msec + msec
- often a costly upgrade



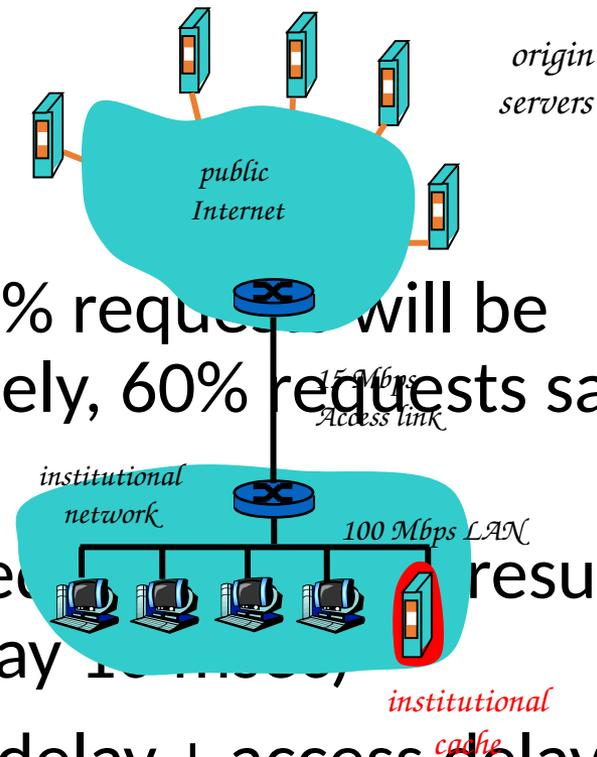
Caching Example (cont)

possible solution:

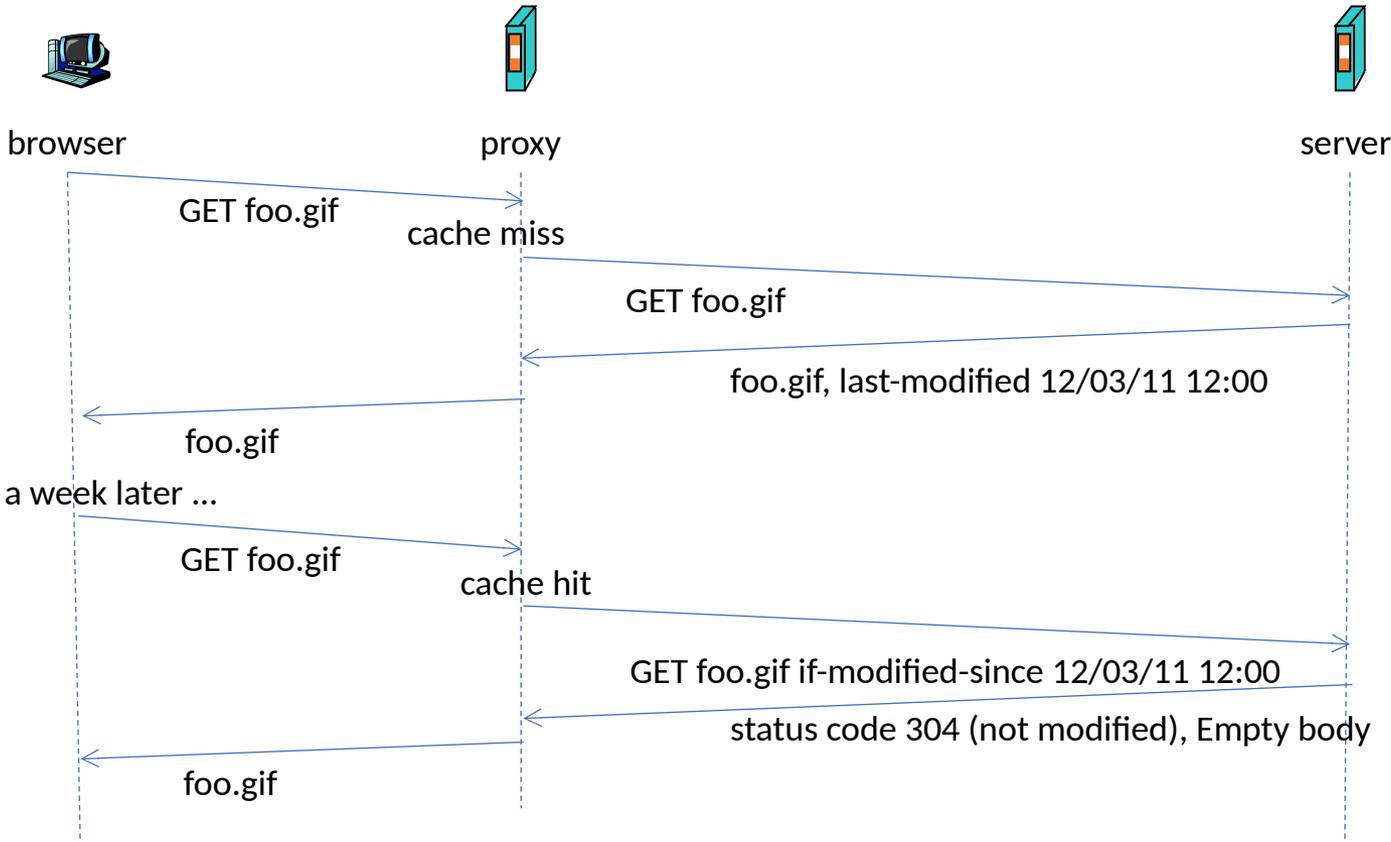
- install cache

consequence

- suppose hit rate is 0.4 (40% requests will be satisfied almost immediately, 60% requests satisfied by origin server)
- utilization of access link reduced, resulting in negligible delays (say 10 milliseconds)
- avg total delay = Internet delay + access delay + LAN delay = $0.6 * 2.01 \text{ secs} * + 0.4 * 10 \text{ millisecs} < 1.3 \text{ secs}$



Conditional GET



Other Uses

- Allow multiple users to get a resource which access is limited to the proxy.
- Track and log web accesses.
- Deny access to a list of web sites.



Origins

- *Representational State Transfer* – REST: defined in 2000 Roy Fielding's PhD dissertation (after he worked on HTTP 1.1 and URI RFCs)
- Web application =
 - network of Web resources (a virtual state-machine)
 - where the user progresses through the application by selecting resource identifiers and resource operations (application state transitions), resulting in the next resource's representation (the next application state) being transferred to the end user for their use.
- An architectural style, not a standard nor a protocol

Principles of RESTful Architecture (1/2)

- A resource
 - is identified using an URI,
 - references
 - one entity (eg. user Paul) or
 - a set of entities (eg. all male users)
 - URI doesn't change (but the referenced entity might)
 - and can have multiple representations (JSON, XML...).
- The representation of a resource contains enough information for the client to request a change to its state.
 - Messages include enough information to describe how to process them (eg. Content type)
 - HATEOS (*Hypermedia as the Engine of Application State*)

HATEOAS Example

request

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com
Accept: application/xml
...
```

response if balance > 0

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">100.00</balance>
  <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
  <link rel="withdraw" href="https://bank.example.com/accounts/12345/withdraw" />
  <link rel="transfer" href="https://bank.example.com/accounts/12345/transfer" />
  <link rel="close" href="https://bank.example.com/accounts/12345/status" />
</account>
```

response if balance < 0

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">-25.00</balance>
  <link rel="deposit" href="https://bank.example.com/accounts/12345/deposit" />
</account>
```

Principles of RESTful Architecture (2/2)

- Separation of concerns between the client (user interface concerns) and the server (data storage and processing concerns)
- Stateless communication: the server only stores resources states while the client is in charge of providing the application state.
- Responses should define the extent to which they can be cached.
- A client may not be directly connected to the end-server: there can be proxies, an additional security layer, and the server might call other servers to complete the service.

Semantics of HTTP methods

HTTP method	Operation on the resource	URIs: examples	HTTP response status	<i>location</i> header	safe	idem potent
GET	read	GET /serv/users GET /serv/users/34	200 OK	no	yes	Yes
POST	create	POST /serv/users # body { name: "Toto" }	201 Created	Yes	no	no
PUT	update	PUT /serv/users/34 # body { name: "Jacques" }	200, 204 No Content	no	no	Yes
PATCH	partial update					
DELETE	delete	DELETE /serv/users/34	200, 204, 202 Accepted	no	no	yes

Example of scenario

- Book a room:

POST [http://myhotel.com/reservations?date="12/03/2021"&nights=2&persons=4](http://myhotel.com/reservations?date=)

Server replies with reservation number 123

- Display reservation:

GET <http://myhotel.com/reservations/123>

- Update the reservation:

PATCH <http://myhotel.com/reservations/123?persons=3>

- Cancel the reservation:

DELETE <http://myhotel.com/reservations/123>

Best Practices for well-designed RESTful APIs

- Use only nouns for a URI:

~~/getAllReservations~~ GET /reservations

- Use plural nouns:

GET /reservations for all reservations

GET /reservations/123 for a specific reservation

- GET method should not alter the state of a resource
- Use sub-resources for relationships between resources

GET /reservations/123/persons/1: first occupant of the reservation #123

- Use “content-type” and “accept” HTTP headers to specify input/output format
- Provide proper HTTP status codes

Best Practices for well-designed RESTful APIs

- Offer filtering and paging capabilities for large data sets

[GET /reservations?date=28/02/2021](#)

[GET /reservations?from=5&to=25](#)

- Version the API

2 strategies:

- In the URI: [GET /api/v2/reservations/123](#)



Easy to use with a web browser



Non-compliant with REST principle “one resource = one URI”

- In the accept header:

[GET /api/reservations/123 accept: application/v2](#)



More complex for the client



More REST-compliant