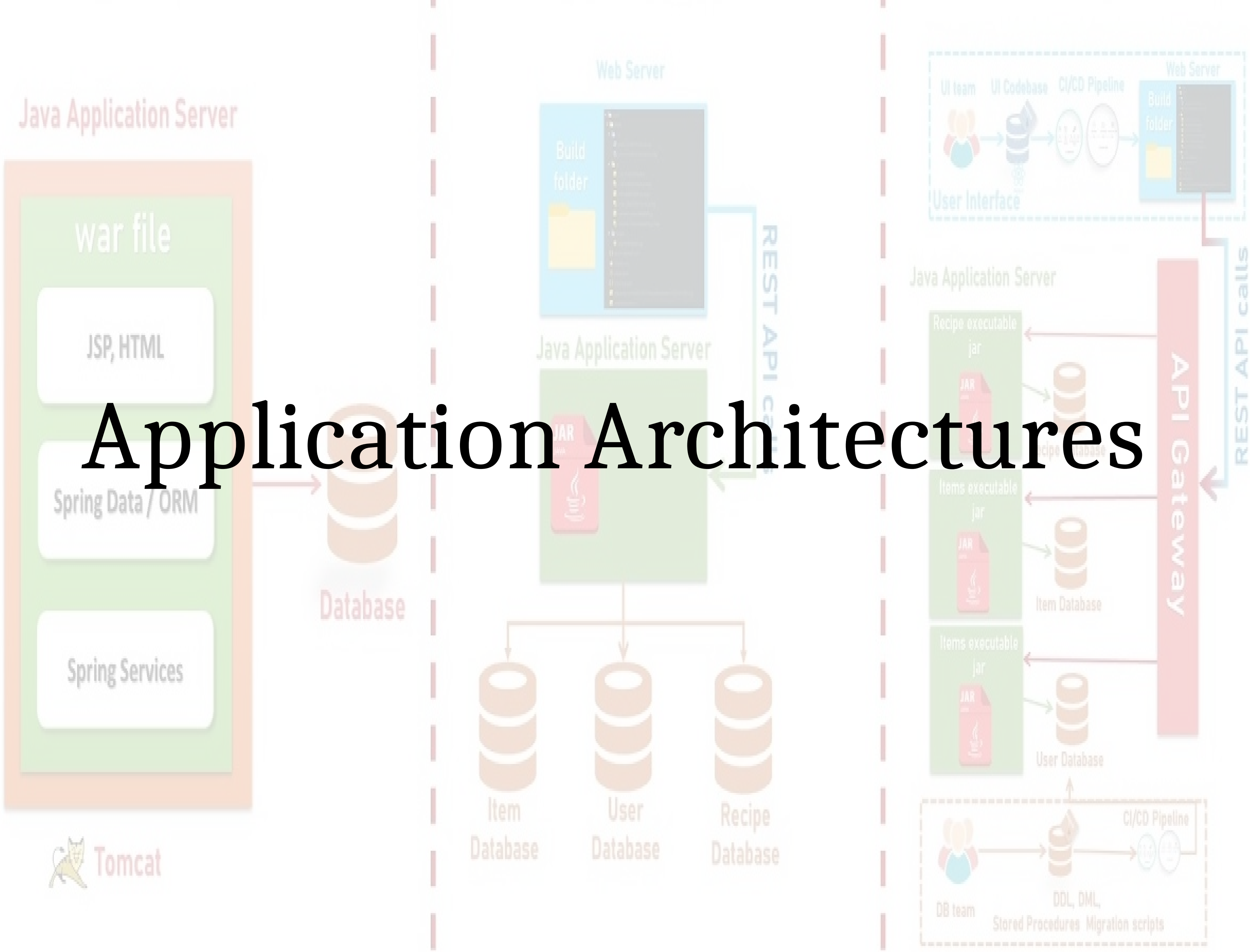
Application Architectures



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:

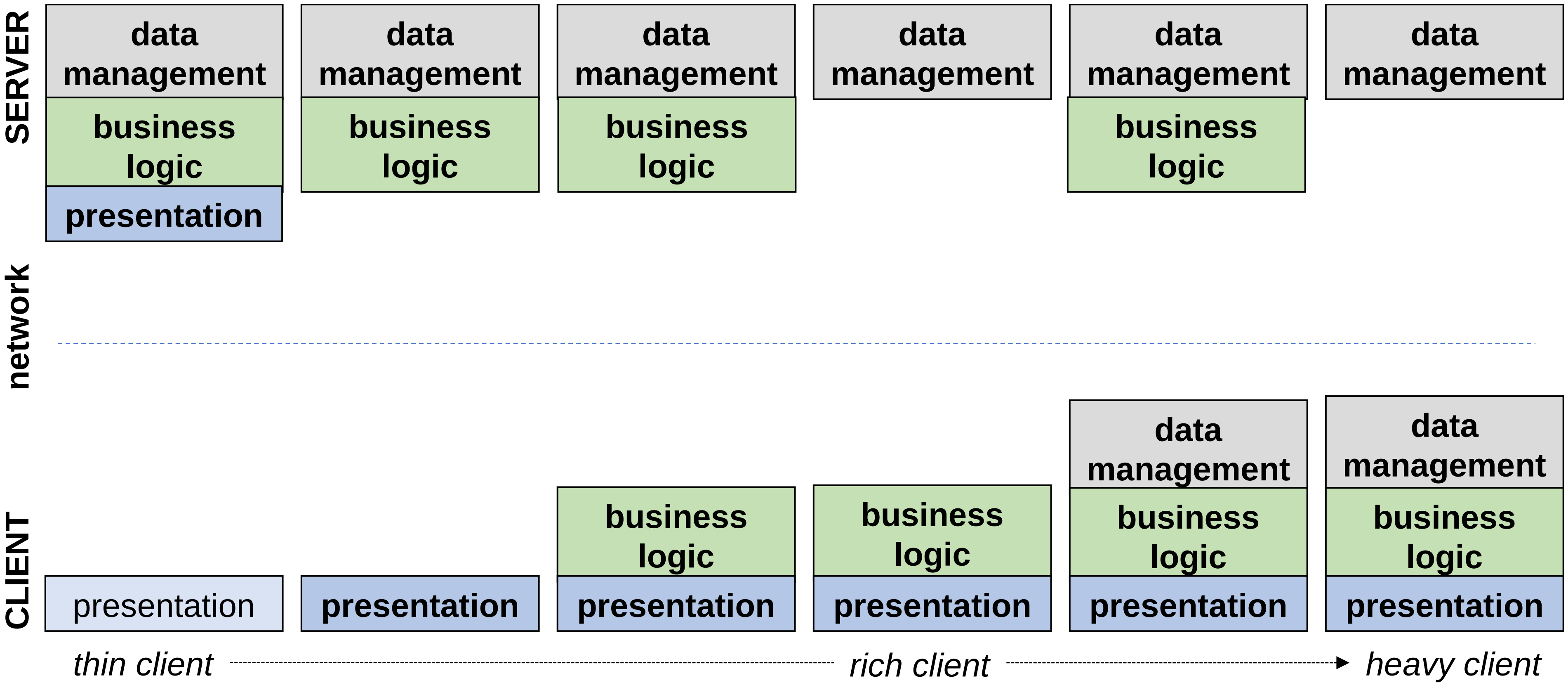
  business objects, rules, processing logic, processes

- Data:

  storage and logical access

Distribution onto « Tiers »

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



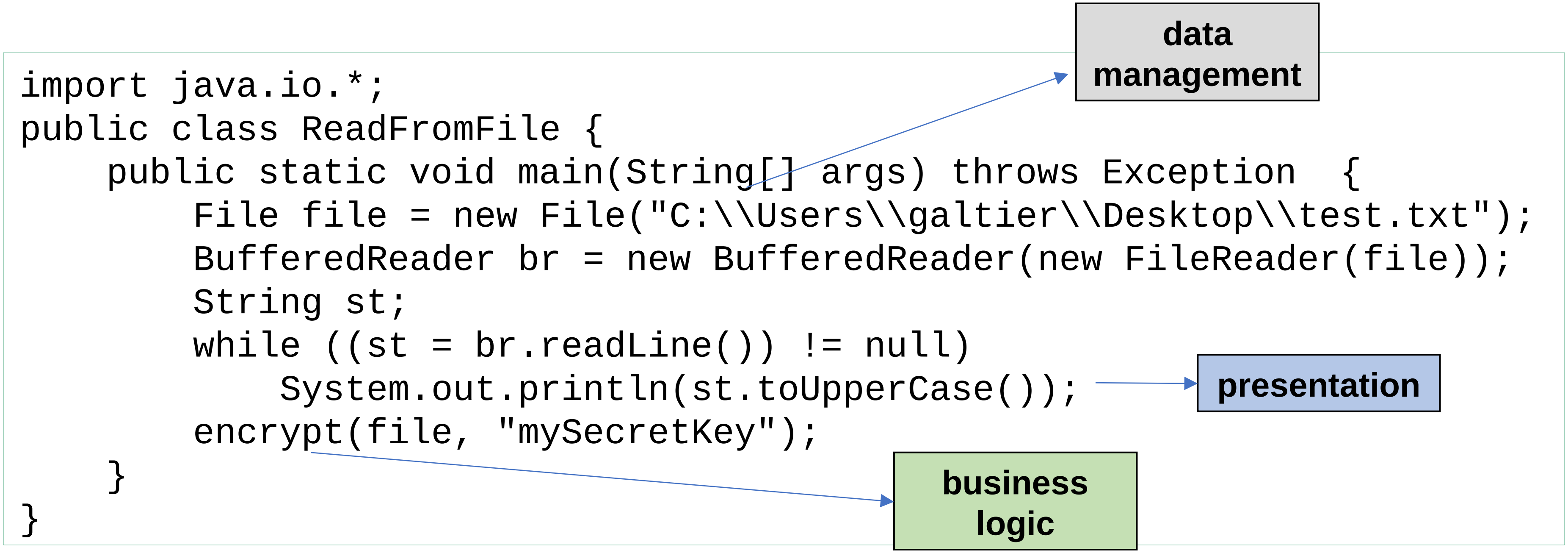
SERVER	data management business logic presentation	data management business logic	data management business logic	data management	data management business logic	data management
network						
CLIENT	presentation	presentation	business logic presentation	business logic presentation	data management business logic presentation	data management business logic presentation

thin client → *rich client* → *heavy client*

Monolithic and Single-tier Applications

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");
    }
}
```

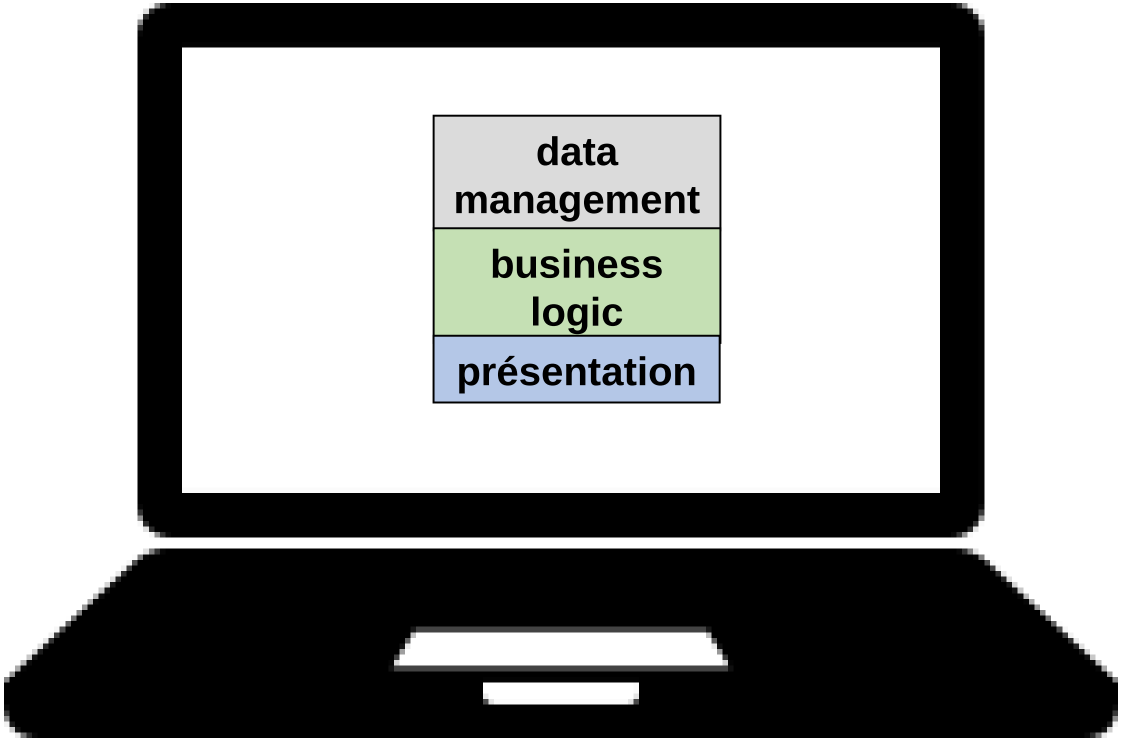
data management

presentation

business logic

Single-tier Application

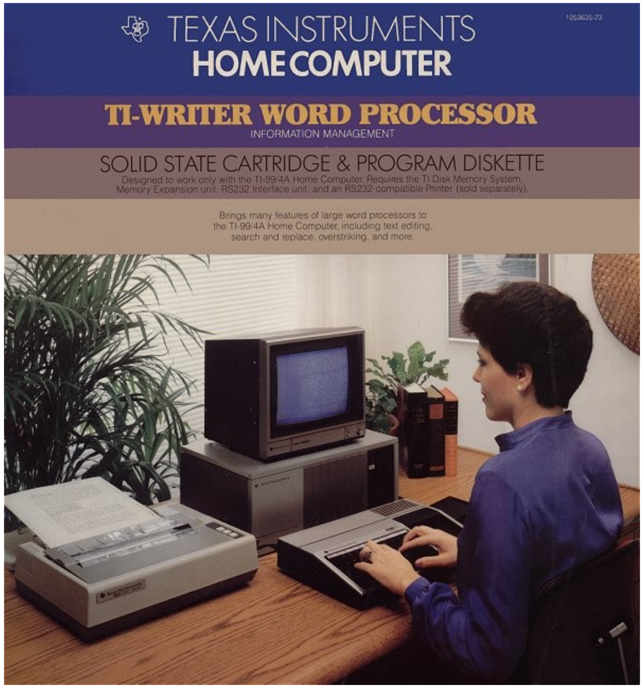
The 3 application layers run on the same computer



1st architectural style, but still relevant

- The area of “pre-network” PCs (late 70 ’s – mid 80’s)

- Still lots of stand-alone apps



Designed to work only with the TI-99/4A Home Computer. Requires the TI Disk Memory System, Memory Expansion unit, RS232 Interface unit, and an RS232-compatible Printer (sold separately).

Brings many features of large word processors to the TI-99/4A Home Computer, including text editing, search and replace, overstriking, and more.

https://games.alphacoders.com/games/boxart/49494

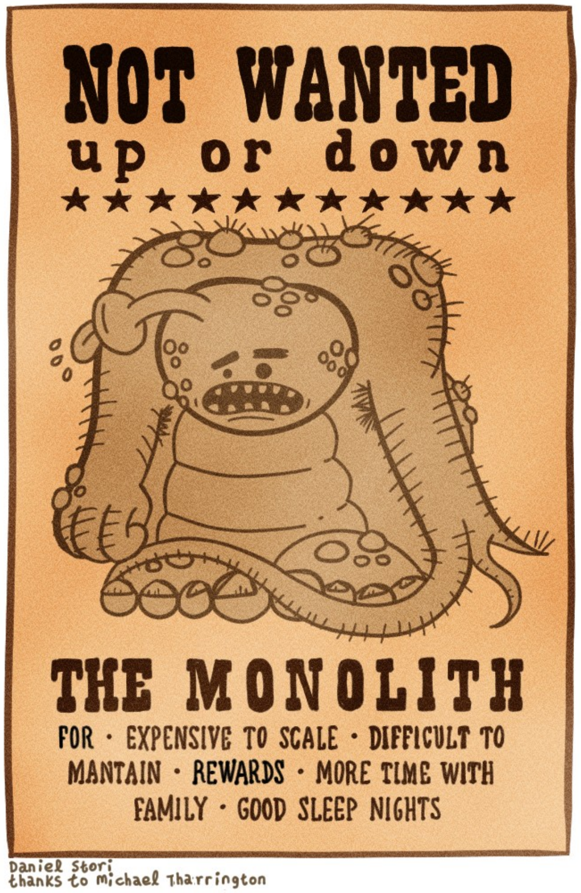
https://play.google.com/store/apps/details?id=com.medibang.android.paint.tablet&hl=en_SG&gl=US

Advantages of single-tier

- Performance: 0 latency
- Safety by isolation
- Operate even in disconnected mode
- Simplicity (complexity reduced to the one of the code)

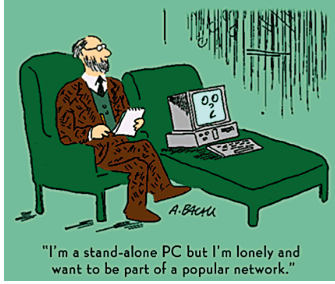
Disadvantages of monolithic applications

- Code is complex to learn, debug and evolve
- Even a minor upgrade requires a complete reinstallation of the entire application
- A failure in one “layer” renders the application completely unusable
- Inability to leverage heterogeneous technologies
- Not cloud-ready


NOT WANTED
up or down
★★★★★★★★★★★
THE MONOLITH
FOR · EXPENSIVE TO SCALE · DIFFICULT TO MANTAIN · REWARDS · MORE TIME WITH FAMILY · GOOD SLEEP NIGHTS
Daniel Stori
thanks to Michael Tharrington

https://dzone.com/articles/not-wanted-comic

Disadvantages of single-tier applications



"I'm a stand-alone PC but I'm lonely and want to be part of a popular network."

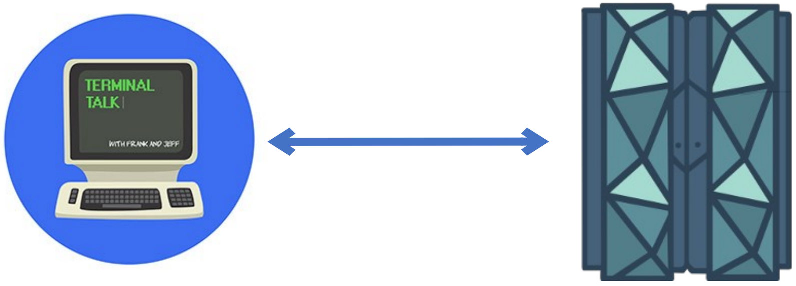
https://encyclopedia2.thefreedictionary.com/standalone+computer

- Performances: depend on the capabilities of the host
- Shared resources impossible, requires duplicates (waste of resources)
- No fault tolerance
- Nomadism is difficult:
  - Access limited to physically logged-in users
  - More difficult (if not impossible) to continue a task from a different workstation
- Deployment is difficult:
  - Requires actions on each terminal
  - To be reinstalled if the underlying system needs to be reinstalled
- From the publisher's point of view:
  - No fix possible without user action
  - Application vulnerable to reverse engineering

https://www.zdnet.fr/actualites/6-idees-recues-qui-empoisonnent-le-mainframe-39892381.htm

Mainframe Architectures

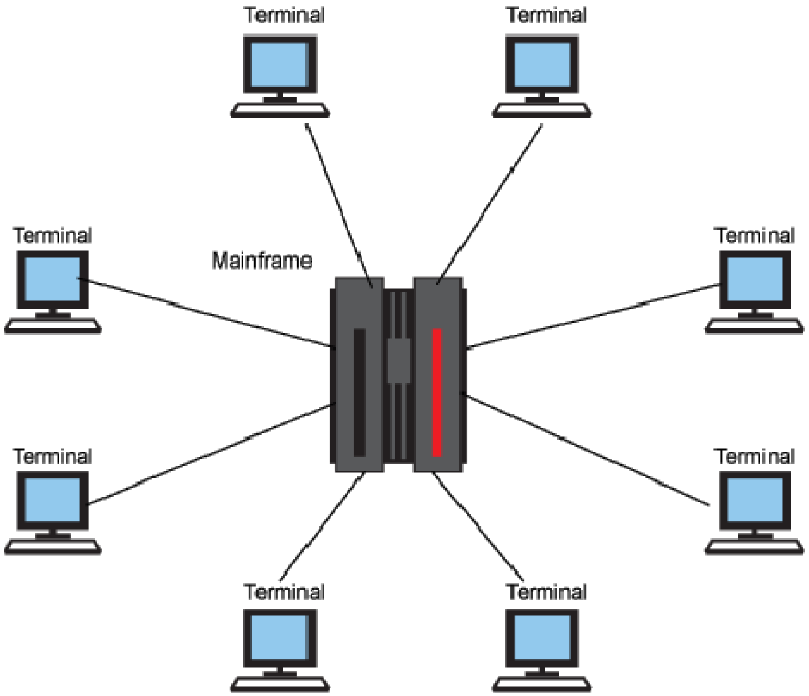
Principle "host" Architecture



https://www.reddit.com/r/mainframe/comments/dq8pny/online_ibm_z_day_on_november_21st/

- Supercomputer :
  - ensures the data persistence, processing, and presentation
  - proprietary hardware and OS (IBM)
- passive clients :

  thin client visualization application



https://www.researchgate.net/figure/Mainframe-Architecture_fig2_275405572

Advantages

- Performances: handle a very large number of simultaneous queries on very large databases
- Consistency, stability and long-term support
- Security
- Reliability (IBM Z customers: 99.9999% uptime)

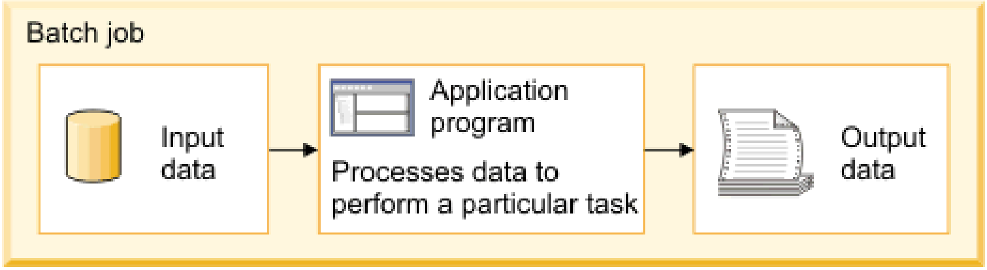
Robustness: https://www.ibmmainframeforum.com/mainframe-videos/topic10889.html

Performances

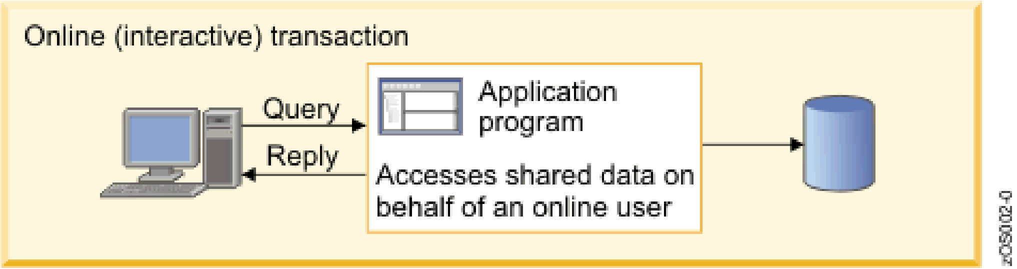
- Ability to process a very large number of simultaneous queries on very large databases

Batch or real time operation:

- Batch back-office



- Transactional


Online (interactive) transaction
Query
Reply
Application program
Accesses shared data on behalf of an online user
zOS002-0

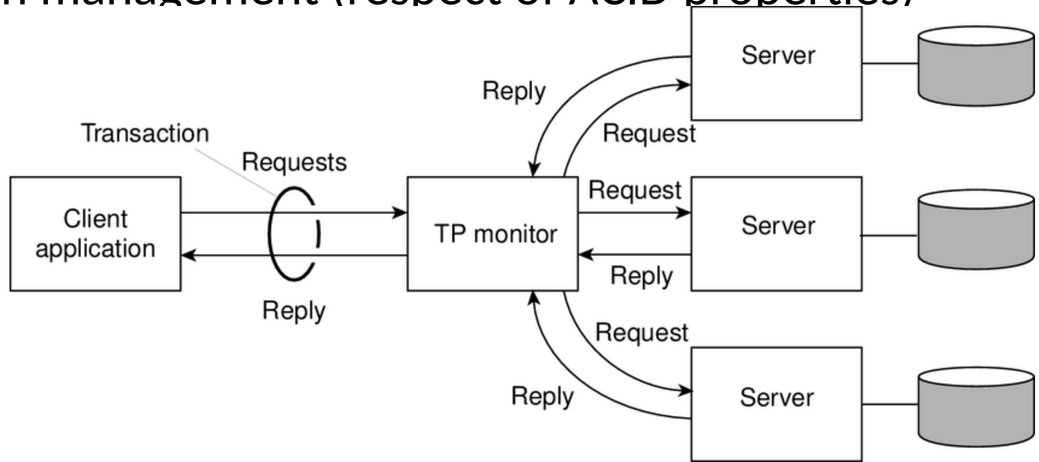
- Used in banks, insurance companies, airlines...

https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zmainframe/zconc_mfworklds.htm

Transactions

- *Program accessing and/or modifying persistent data*
- A good transaction is
  - **A**tomic
  - **C**onsistent
  - **I**solated
  - **D**urable
- Transactional monitor ("TP monitor")

  Schedules transactions executed in parallel
  - Multiplexing of requests on system resources
  - Transaction management (respect of ACID properties)



https://www.researchgate.net/figure/The-role-of-a-TP-monitor-in-distributed-systems_fig7_29821518б

Extensively used

- 71% of the Fortune 500, 96 of the top 100 banks use mainframes
- process 30 billion business transactions per day, 87% of credit card transactions
- 250 billion lines of COBOL code, and 5 billion new lines each year

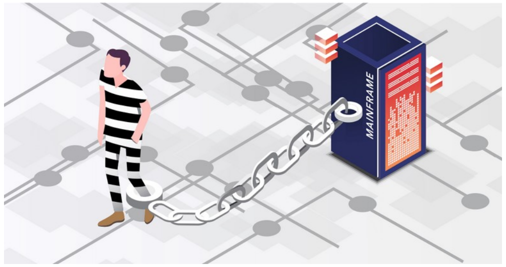
- Growth Outlook:
  - demand for HPC
  - increase in the number of banking transactions
  - development of blockchain

Obstacles to growth

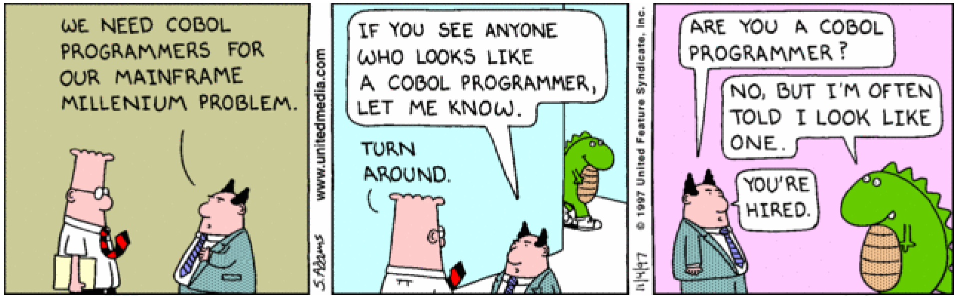
- Proprietary solutions
- Huge investment
  - but no more than a server farm

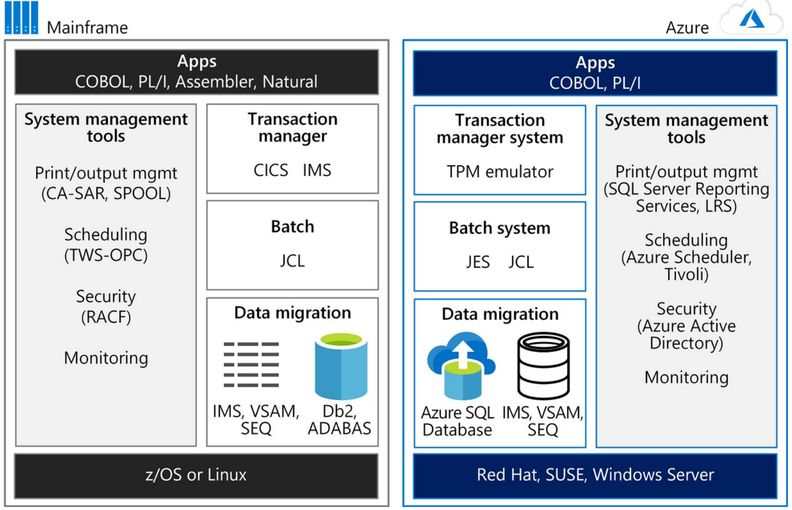
(https://planetmainframe.com/2021/09/the-ibm-mainframe-the-most-powerful-and-cost-effective-computing-platform-for-business/

- Shortage of skilled mainframe staff
  - but Cobol is easy to learn
- Real alternatives + migration experience



https://www.astadia.com/blog/break-free-from-your-z13-mainframe


WE NEED COBOL PROGRAMMERS FOR OUR MAINFRAME MILLENIUM PROBLEM.
www.unitedmedia.com
S.Adams
IF YOU SEE ANYONE WHO LOOKS LIKE A COBOL PROGRAMMER, LET ME KNOW.
TURN AROUND.
© 1997 United Feature Syndicate, Inc.
11/4/97
ARE YOU A COBOL PROGRAMMER?
NO, BUT I'M OFTEN TOLD I LOOK LIKE ONE.
YOU'RE HIRED.


Mainframe

Apps
COBOL, PL/I, Assembler, Natural

System management tools

Print/output mgmt (CA-SAR, SPOOL)

Scheduling (TWS-OPC)

Security (RACF)

Monitoring

Transaction manager

CICS IMS

Batch

JCL

Data migration

IMS, VSAM, SEQ

Db2, ADABAS

z/OS or Linux

Azure

Apps
COBOL, PL/I

Transaction manager system

TPM emulator

Batch system

JES JCL

Data migration

Azure SQL Database

IMS, VSAM, SEQ

System management tools

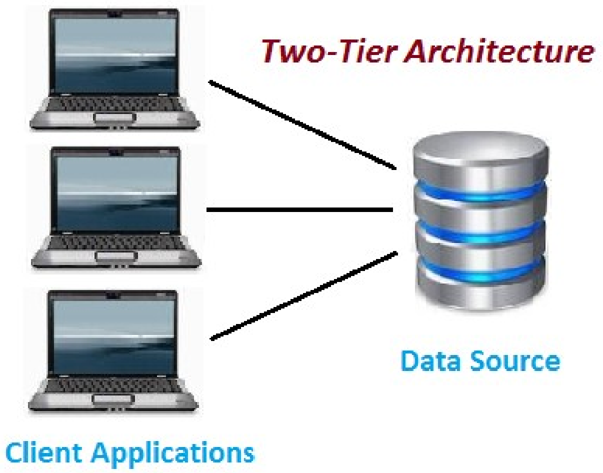
Print/output mgmt (SQL Server Reporting Services, LRS)

Scheduling (Azure Scheduler, Tivoli)

Security (Azure Active Directory)

Monitoring

Red Hat, SUSE, Windows Server

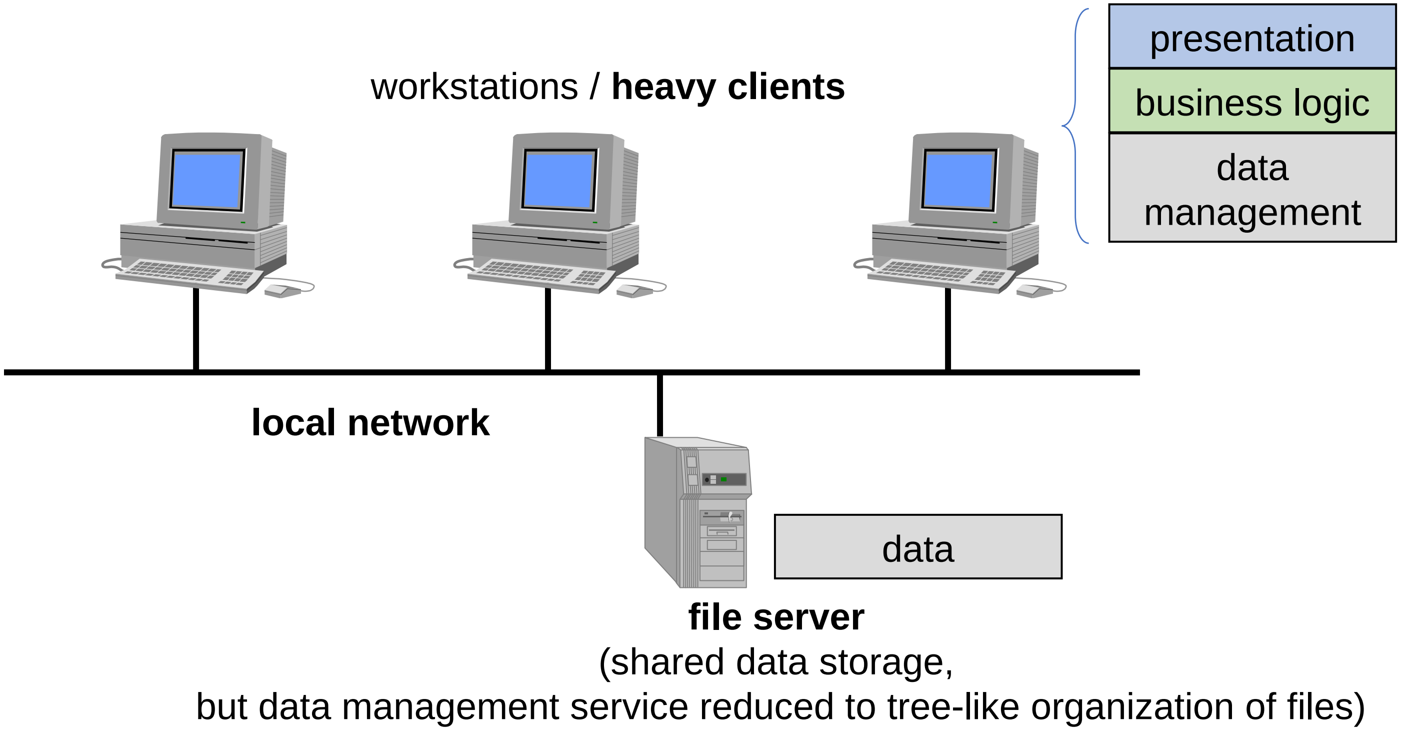


https://www.softwaretestingclass.com/what-is-difference-between-two-tier-and-three-tier-architecture/

2-tier Architecture

The origin: “1.5-tier” Architecture

- Development of LANs



workstations / **heavy clients**

presentation

business logic

data management

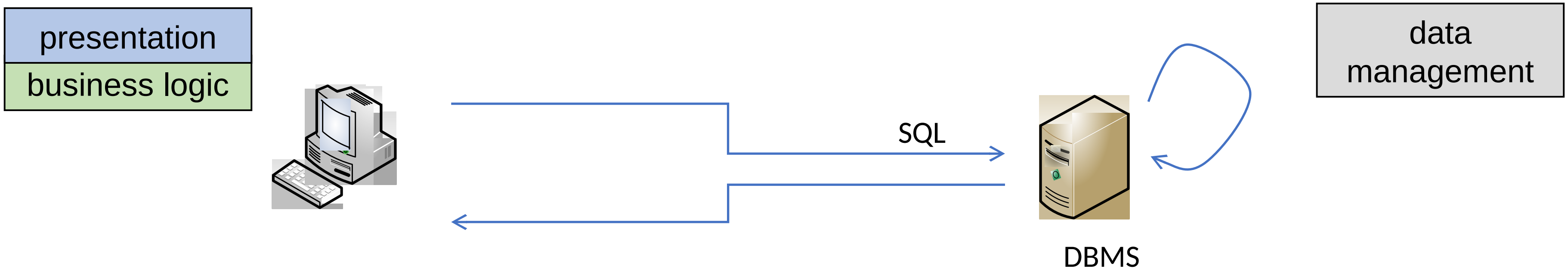
local network

data

file server
(shared data storage,
but data management service reduced to tree-like organization of files)

- 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

2-tier Architecture limits

- identical problems to single-tier:

  Not tolerant to client or server failures, updates require user’s action…

- excessive use of stored procedures:
  - breaks the principle of single responsibility
  - complex to maintain
  - adherence with the physical model
- performance :

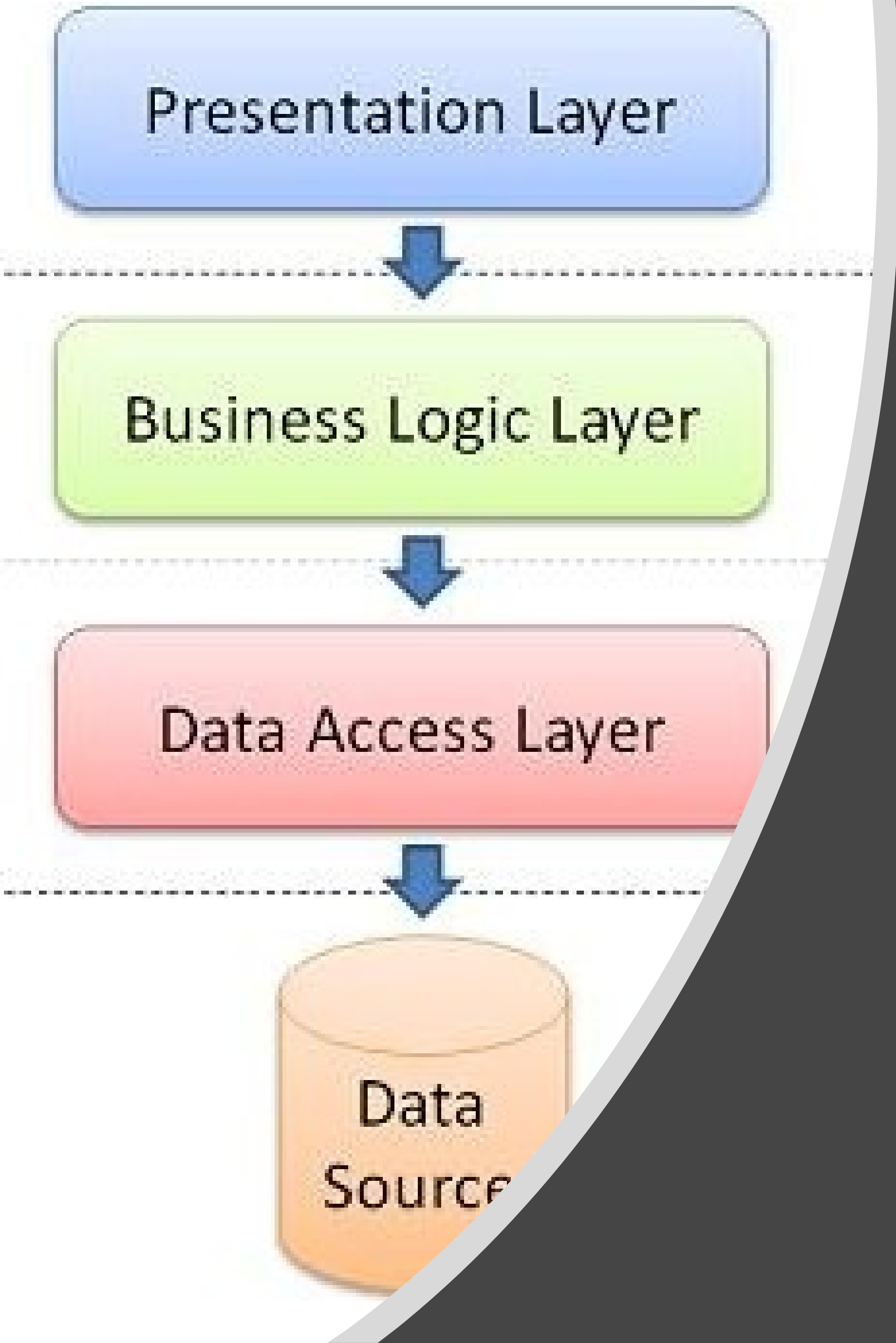
  Server and access network = bottlenecks

Thank you, 2-tier Architecture

- Microcomputing (previously confined to office automation) has taken on a growing role in IS
- The DBMS offer has grown, SQL has become widespread
- Has triggered the evolution towards more flexible architectural proposals

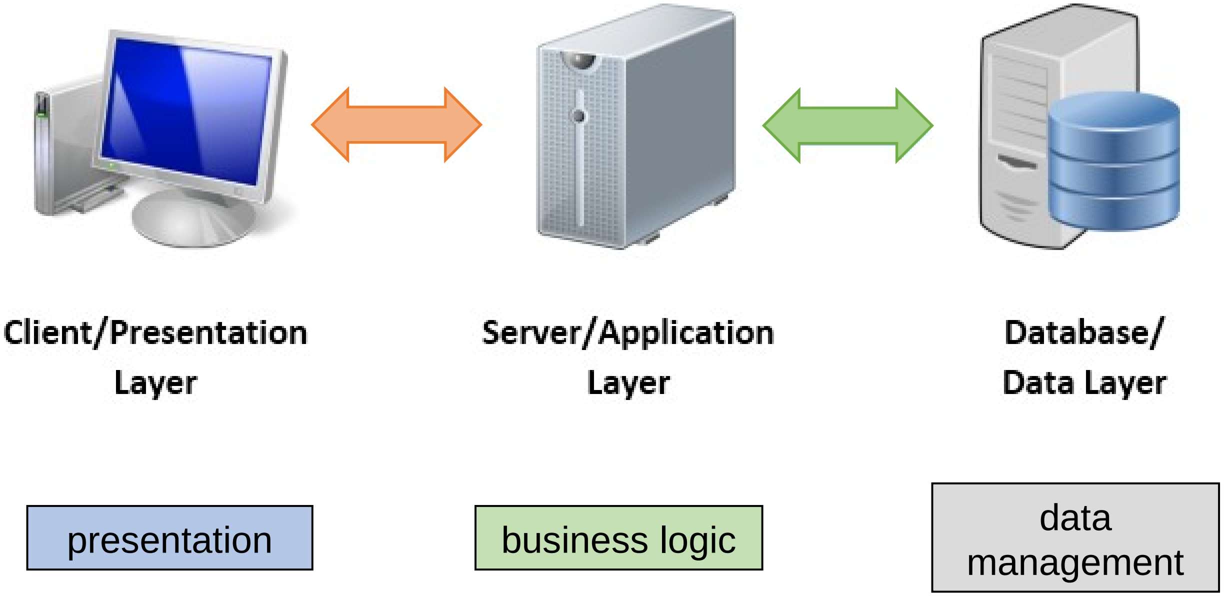
- Still relevant for simple applications

3-tier to 5-tier Architectures



https://weblogs.asp.net/Error.cshtml?aspxerrorpath=/fredriknormen/using-web-services-in-a-3-tier-architecture

3-tier



Client/Presentation Layer

Server/Application Layer

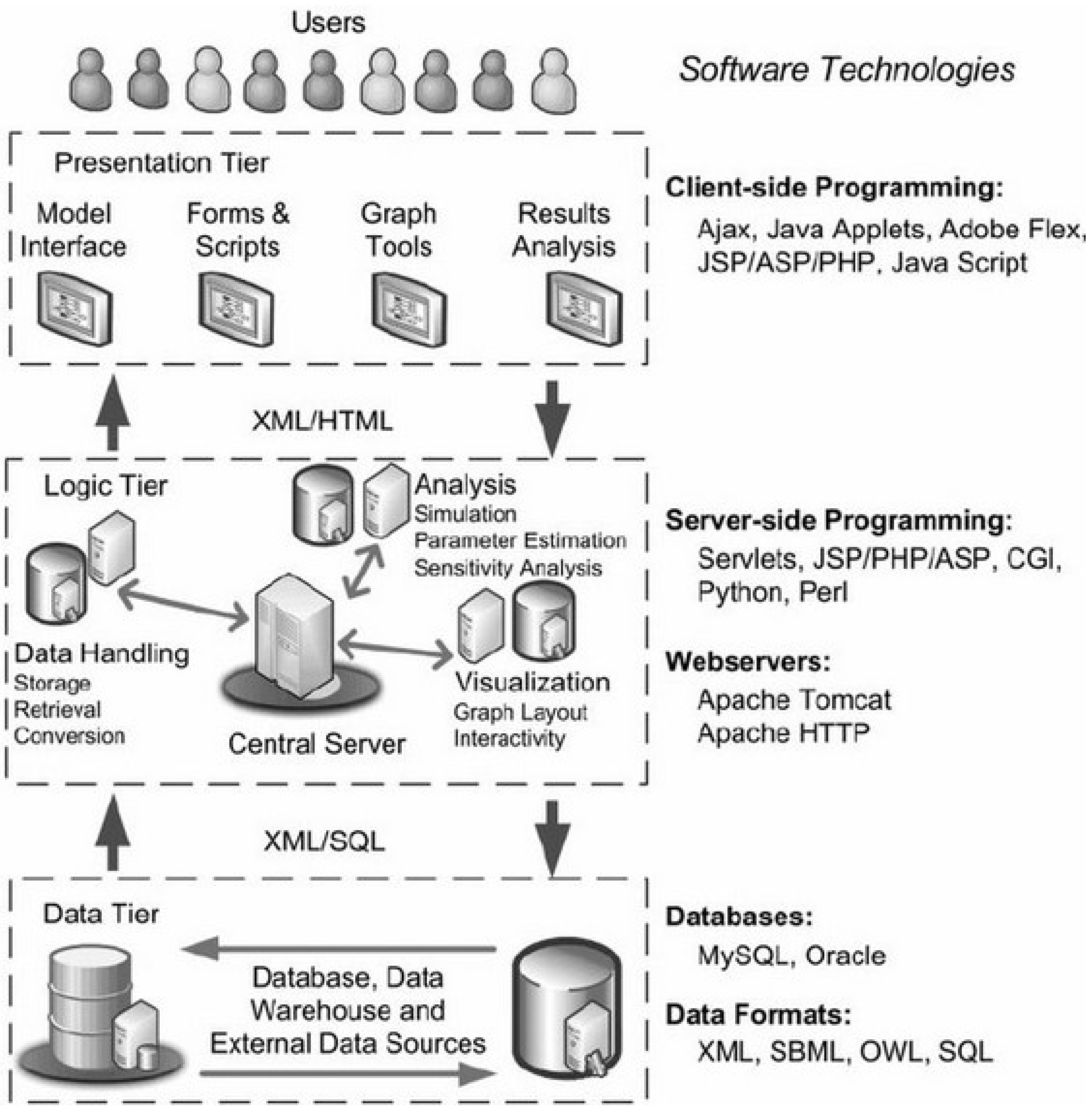
Database/ Data Layer

presentation

business logic

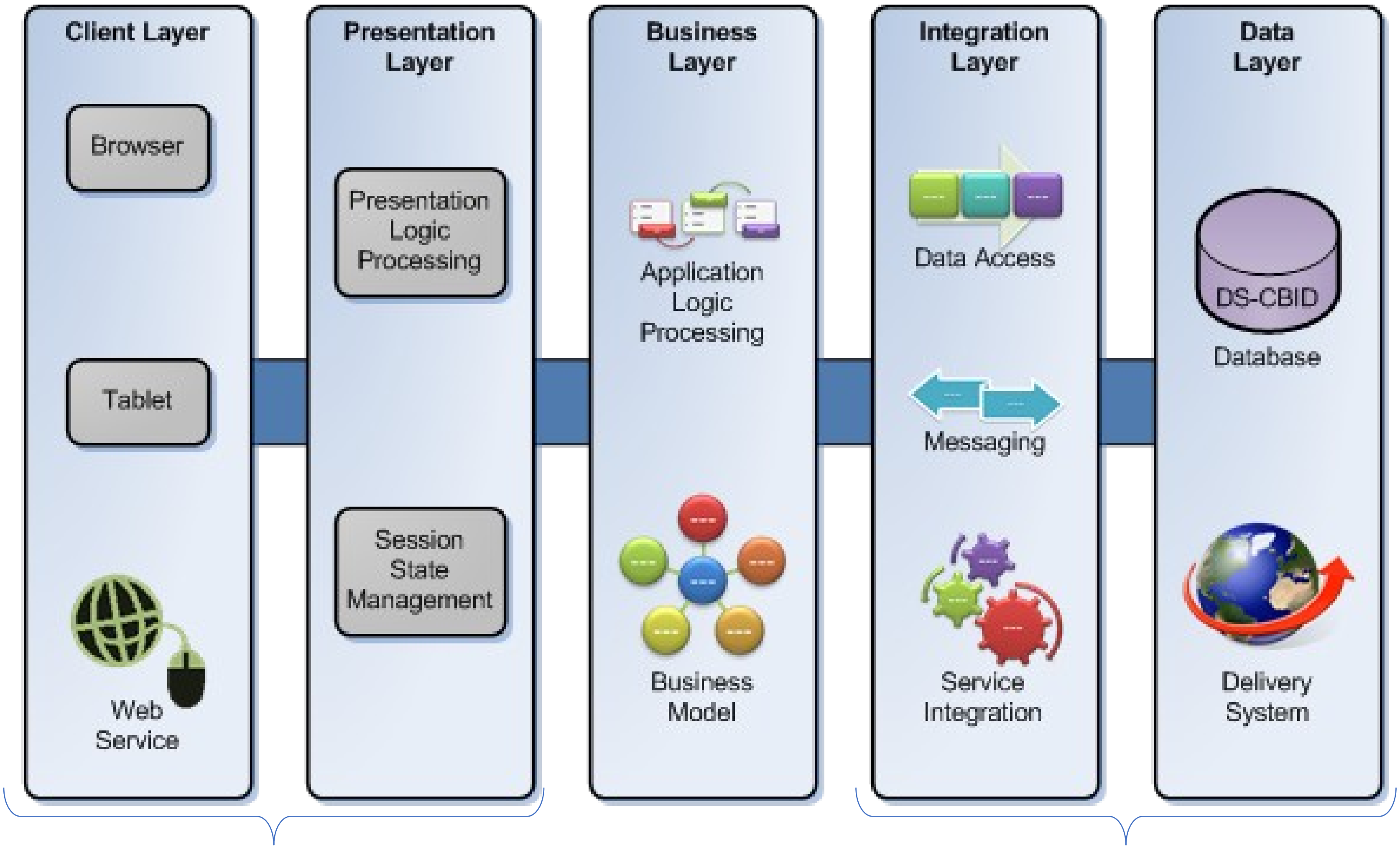
data management

Example:
Classical Web Architecture



Web-based applications for building, managing and analysing kinetic models of biological systems - October 2008 - Briefings in Bioinformatics

4-tier, 5-tier



presentation

business logic

data management

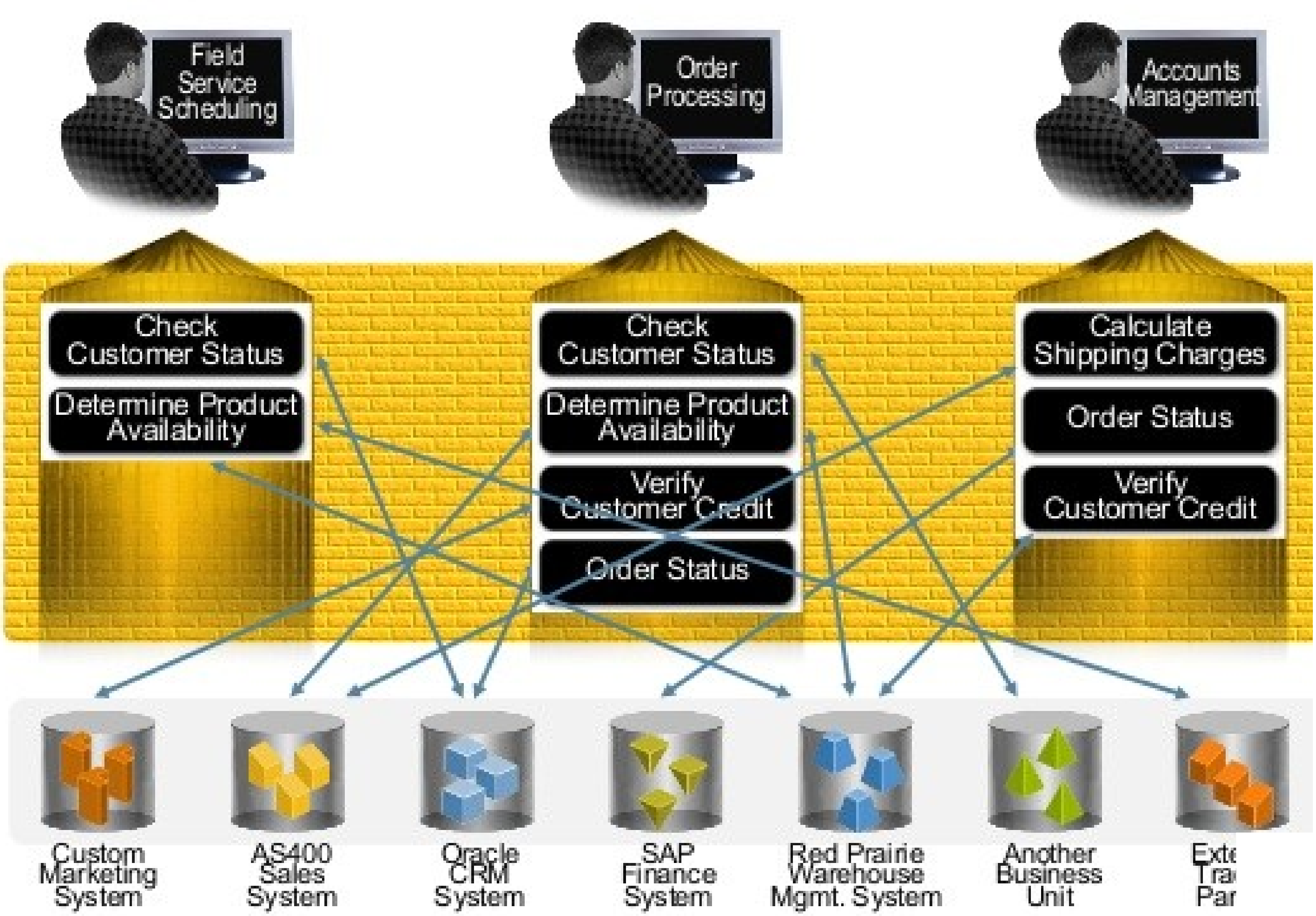
https://www.ajboggs.com/our-experiences/health-information-technology/sisonline-supports-assessment/sisonline-system-architecture/sissa-sis5_big/

Perspectives for multi-tier architecture

- Corrects some of the problems of 2-tier architecture
  - Maintainability, evolvability, deployment
- Very popular model for non-intensive systems
- But to be completed to meet the challenges of reliability, performance, and scalability

Micro-services Architecture

Siloed Architecture

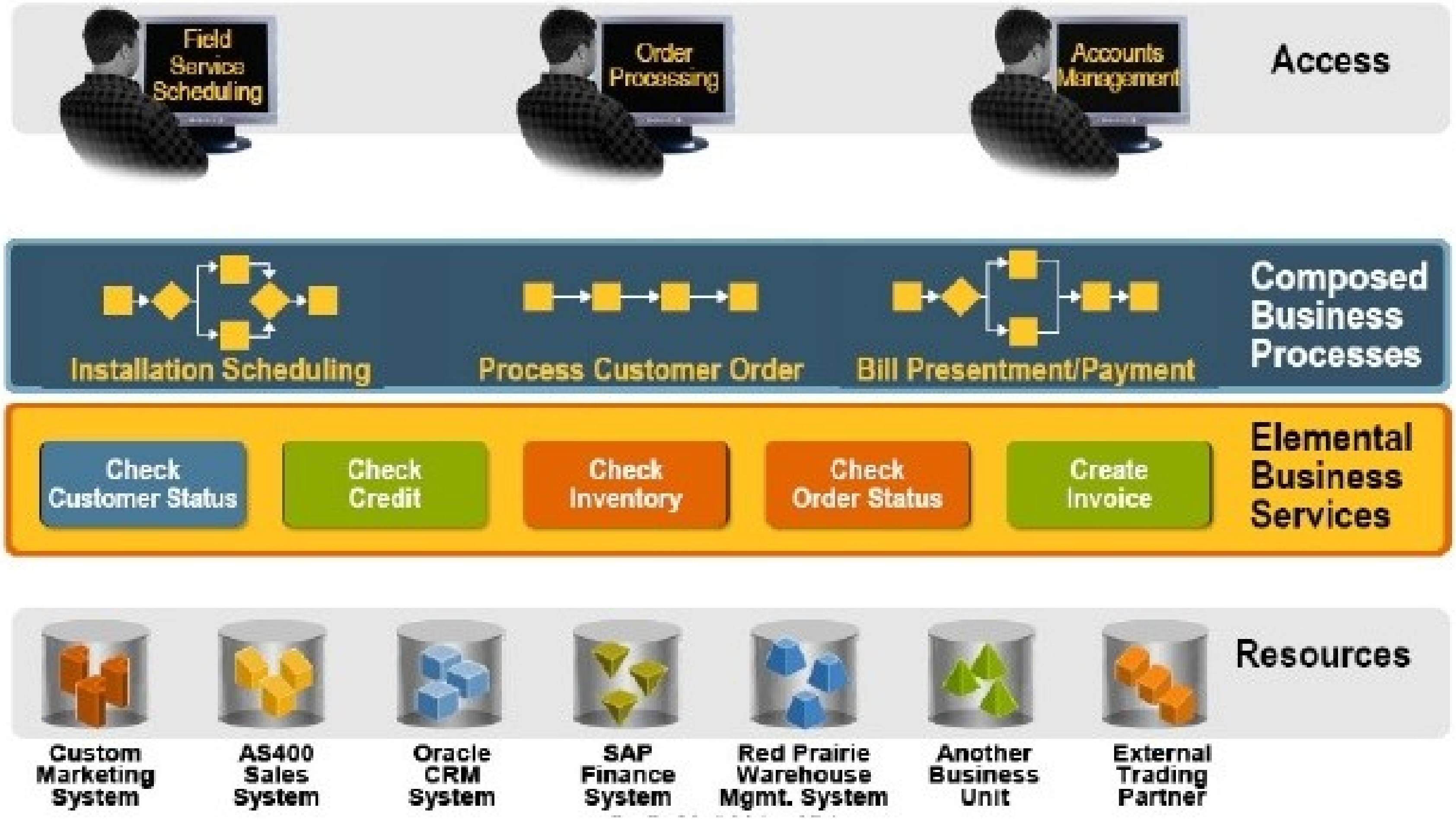


https://www.slideshare.net/imcinstitute/service-oriented-architecture-soa-15-introduction-to-soa

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

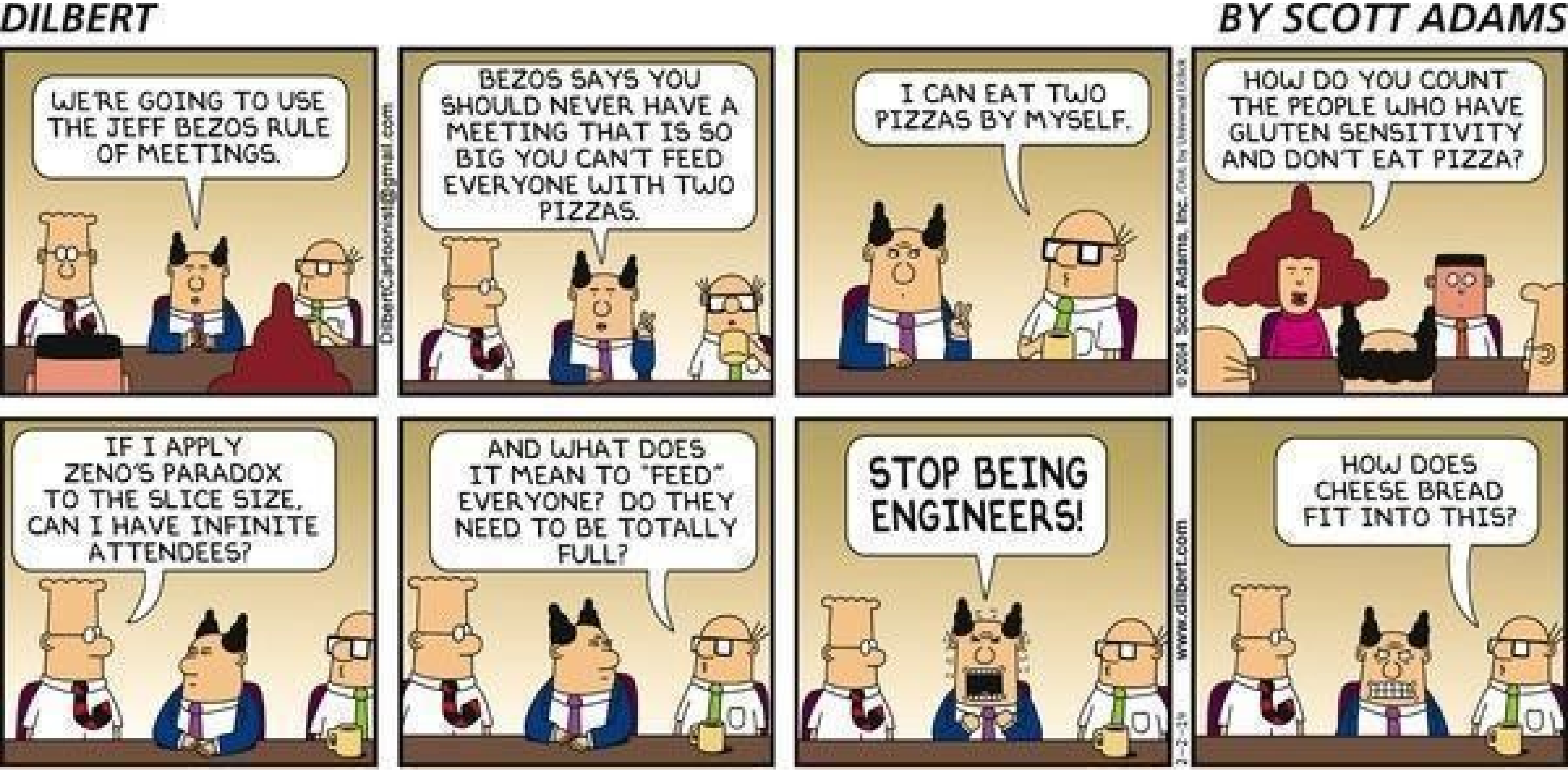


(Micro)Service Concept

- 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*

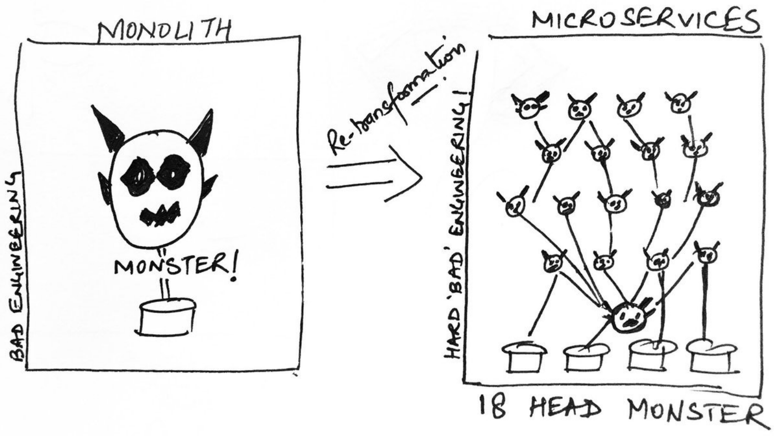
Advantages of the microservice architecture

- 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


DILBERT
BY SCOTT ADAMS
WE'RE GOING TO USE THE JEFF BEZOS RULE OF MEETINGS.
BEZOS SAYS YOU SHOULD NEVER HAVE A MEETING THAT IS SO BIG YOU CAN'T FEED EVERYONE WITH TWO PIZZAS.
I CAN EAT TWO PIZZAS BY MYSELF.
HOW DO YOU COUNT THE PEOPLE WHO HAVE GLUTEN SENSITIVITY AND DON'T EAT PIZZA?
IF I APPLY ZENO'S PARADOX TO THE SLICE SIZE, CAN I HAVE INFINITE ATTENDEES?
AND WHAT DOES IT MEAN TO "FEED" EVERYONE? DO THEY NEED TO BE TOTALLY FULL?
STOP BEING ENGINEERS!
HOW DOES CHEESE BREAD FIT INTO THIS?
DilbertCartoonist@gmail.com
© 2014 Scott Adams, Inc. /Dist. by Universal Uclick
www.dilbert.com
3-2-14

No silver bullet…

- The entropy of the IS increases as well!
- Several examples of strategic retreats on a monolithic solution!
- Microservices do not correct design errors.



https://twitter.com/_zenx_/status/999333740122902529

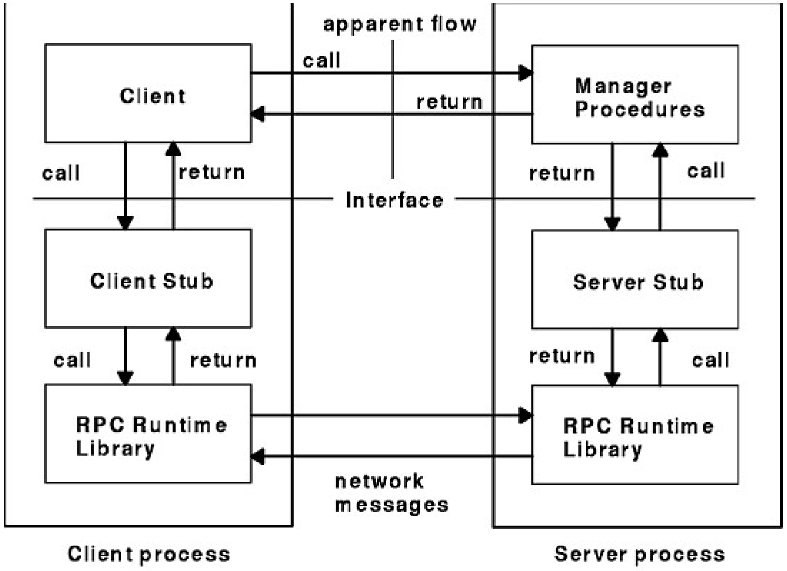
- Intercommunication between services can lead to a higher latency of the application and the network quality becomes crucial

U
I
O
P
Middleware
M
<
>

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



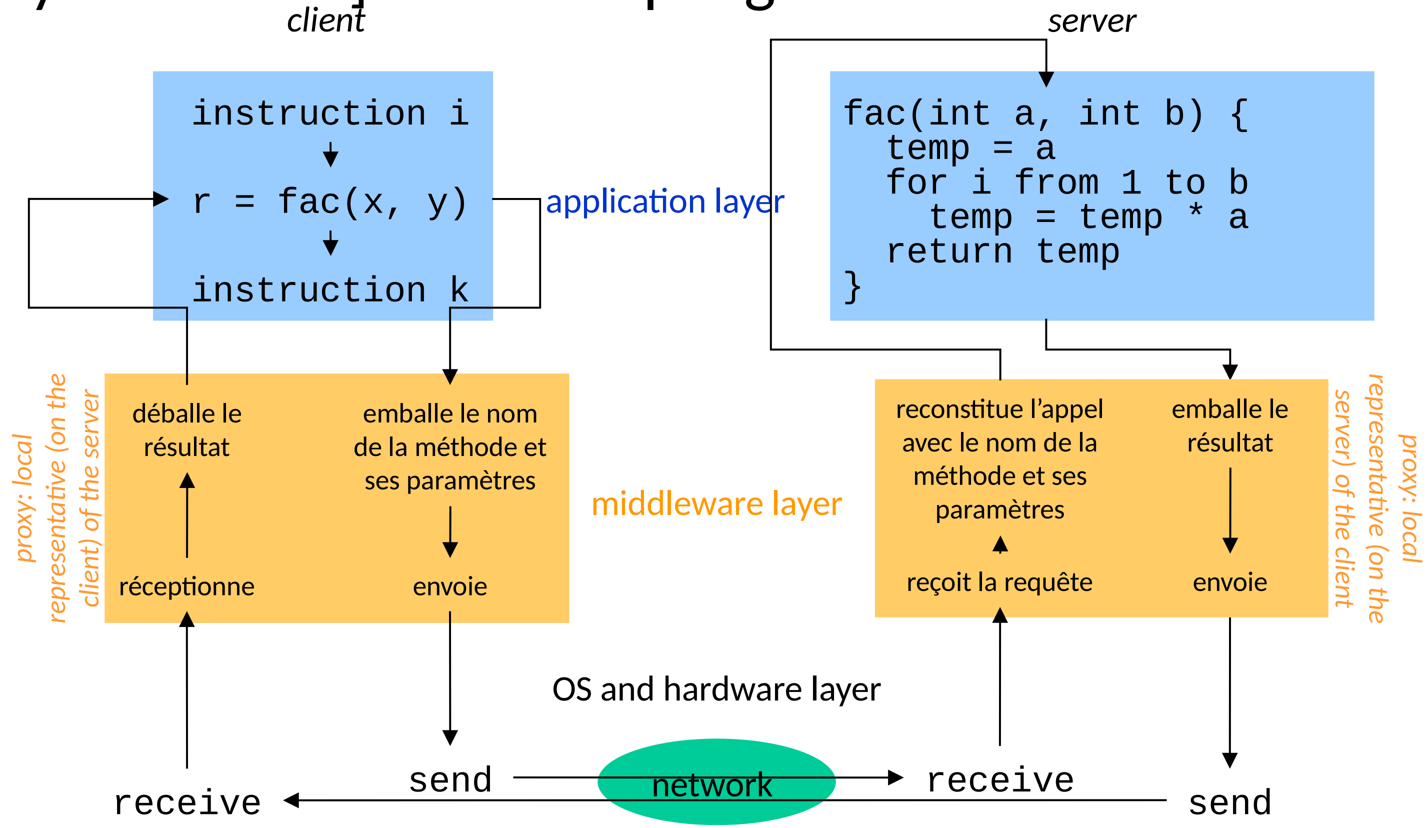
Remote Procedure Call Flow

Remote Procedure Call (RPC) and Object Request Broker (ORB)

https://www.ibm.com/support/knowledgecenter/ssw_aix_72/commprogramming/rpc_mod.html

RPC

- [asynchronous] loose coupling between client and server



client

```
instruction i
r = fac(x, y)
instruction k
```

application layer

server

```
fac(int a, int b) {
  temp = a
  for i from 1 to b
    temp = temp * a
  return temp
}
```

proxy: local representative (on the server) of the client

déballe le résultat

réceptionne

emballe le nom de la méthode et ses paramètres

envoie

middleware layer

reconstitue l'appel avec le nom de la méthode et ses paramètres

reçoit la requête

emballe le résultat

envoie

proxy: local representative (on the client) of the server

OS and hardware layer

```
receive
send
```

network

```
receive
send
```

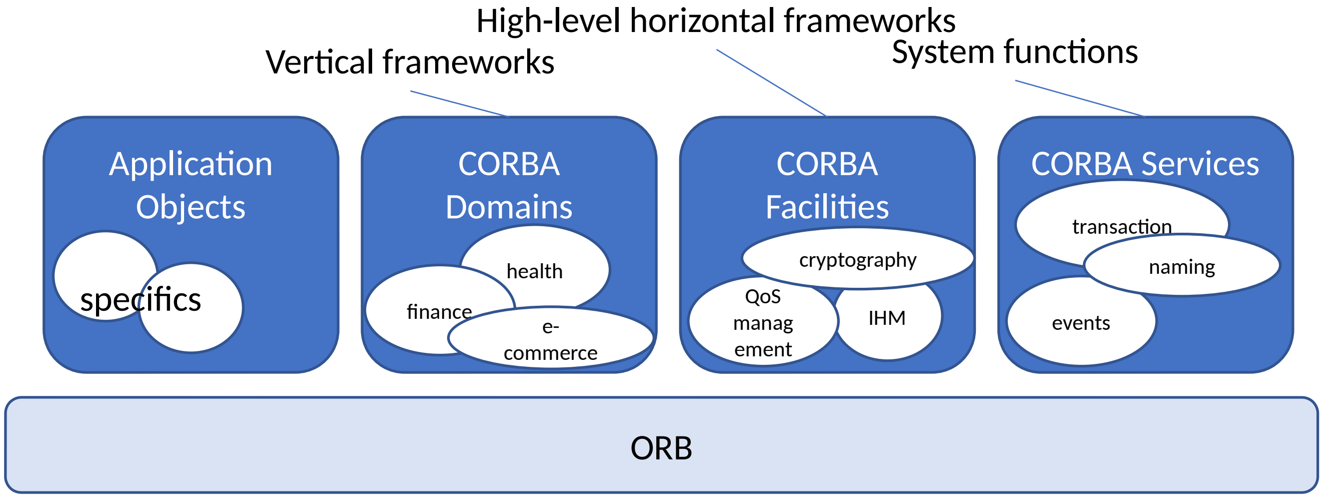
- The proxies handle:
    - network calls
    - format transformations between the client and server

(some) RPC implementations and frameworks

- Rise:
  - 80’s: Sun RPC (as part of NFS protocol): simple, limited to Unix systems
  - 90’s: DCE RPC (Open Software Foundation): platform-independent, rich set of functionalities (transactions, encryption…), more complex to use
- Fall:
  - 94: RPC is “fundamentally flawed”: communication latency, partial failures and concurrency issues…
  - Message passing alternatives
- Rise, again: more features, more supported formats/transports…
  - 98: XML-RPC: data are XML-formatted and exchanged over HTTP -> SOAP
  - 2005: JSON-RPC, lightweight
  - 2007: Apache Thrift (init. Facebook): support for multiple serialization format (including binary), support for multiple transport protocols, complete stack for creating clients and servers
  - 2009: Avro (Apache Hadoop)
  - 2016: gRPC (Google, open source): messages serialized using Protocol Buffers (binary), transported by HTTP/2, multiple features
  - 2021: Cap’n Proto (now developed by Cloudflare): performances!

Object Request Broker

- Object oriented RPC: method calls on remote objects
- Most popular technologies:
  - CORBA (Common Object Request Broker Architecture) (1991)
    - OO-RPC for heterogeneous objects
    - but also a set of services



  - DCOM (Distributed Component Object Model) (1995), .Net Remoting
    - Microsoft-equivalent to CORBA
  - Java RMI (Remote Method Invocation) (1998)
    - for Java objects

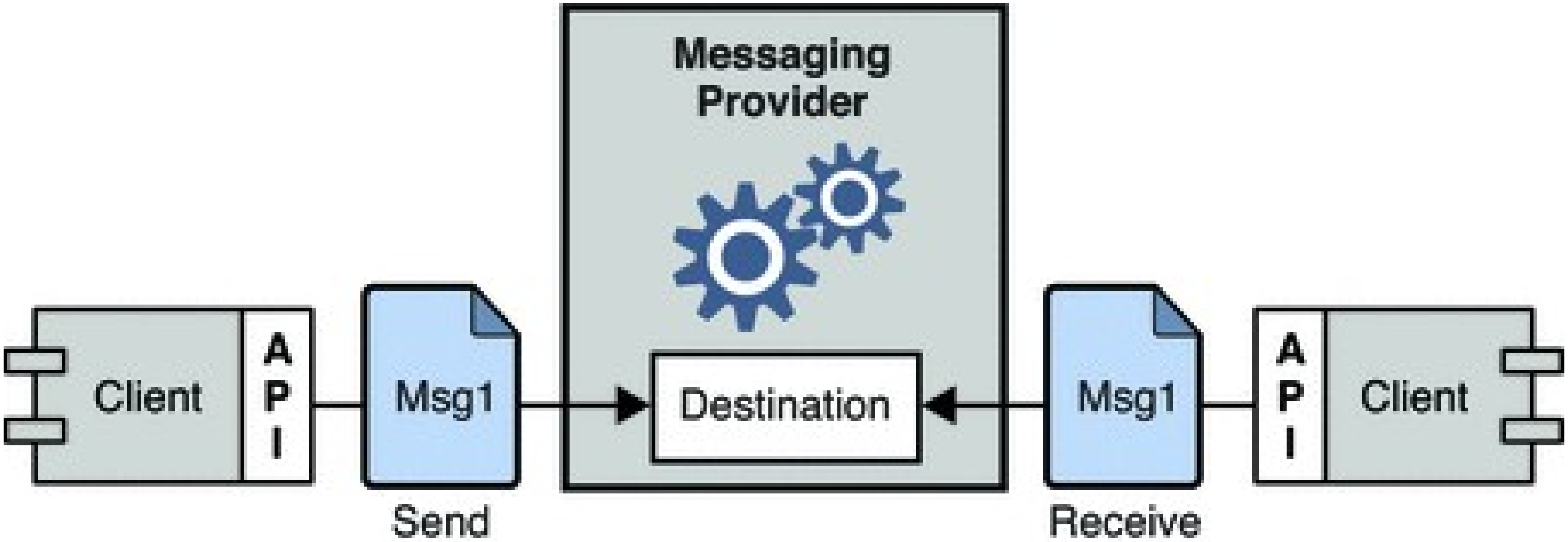
CORBA perspectives

- Limitations:
  - local calls are treated the same as remote calls → inefficient
  - complex standard
  - difficult to have different versions of a service coexisting
  - fewer and fewer experts
- Why hasn't it disappeared?
  - still important legacy
  - one of the few candidates (with DDS) when there are strong real time constraints

    Alcatel-Lucent network management system, communications between military planes and ESA satellites, air control systems, Siemens electrical power plant management system…

Service call

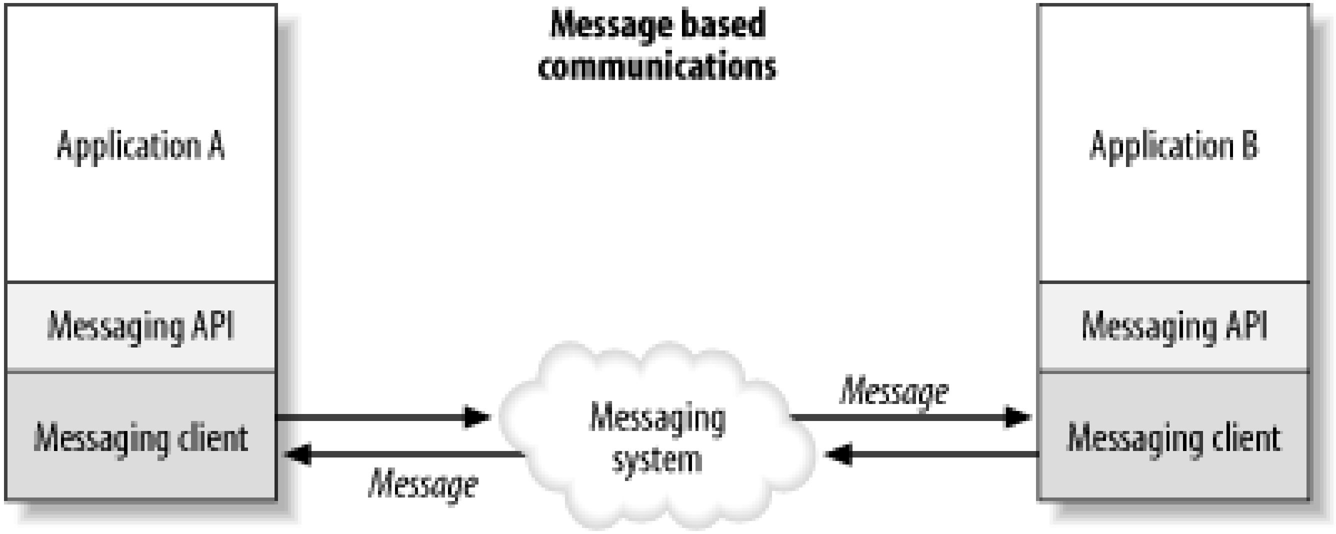
- 1srt generation Web Services:
  - Requests and responses transported by SOAP messages, usually on top of HTTP
  - 4 patterns supported by WSDL:
    - Request - response
    - One way request
    - Notification
    - Request - response
  - WS-*: myriad of specifications to complete the messaging service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



Message Oriented Middleware

Message Oriented Middleware

- Structure allowing one or more sources to transmit messages asynchronously to one or more destinations
  - No need to be connected simultaneously
  - Not need to know the source / the destination



Message oriented middleware

https://www.oreilly.com/library/view/enterprise-service-bus/0596006756/ch05.html

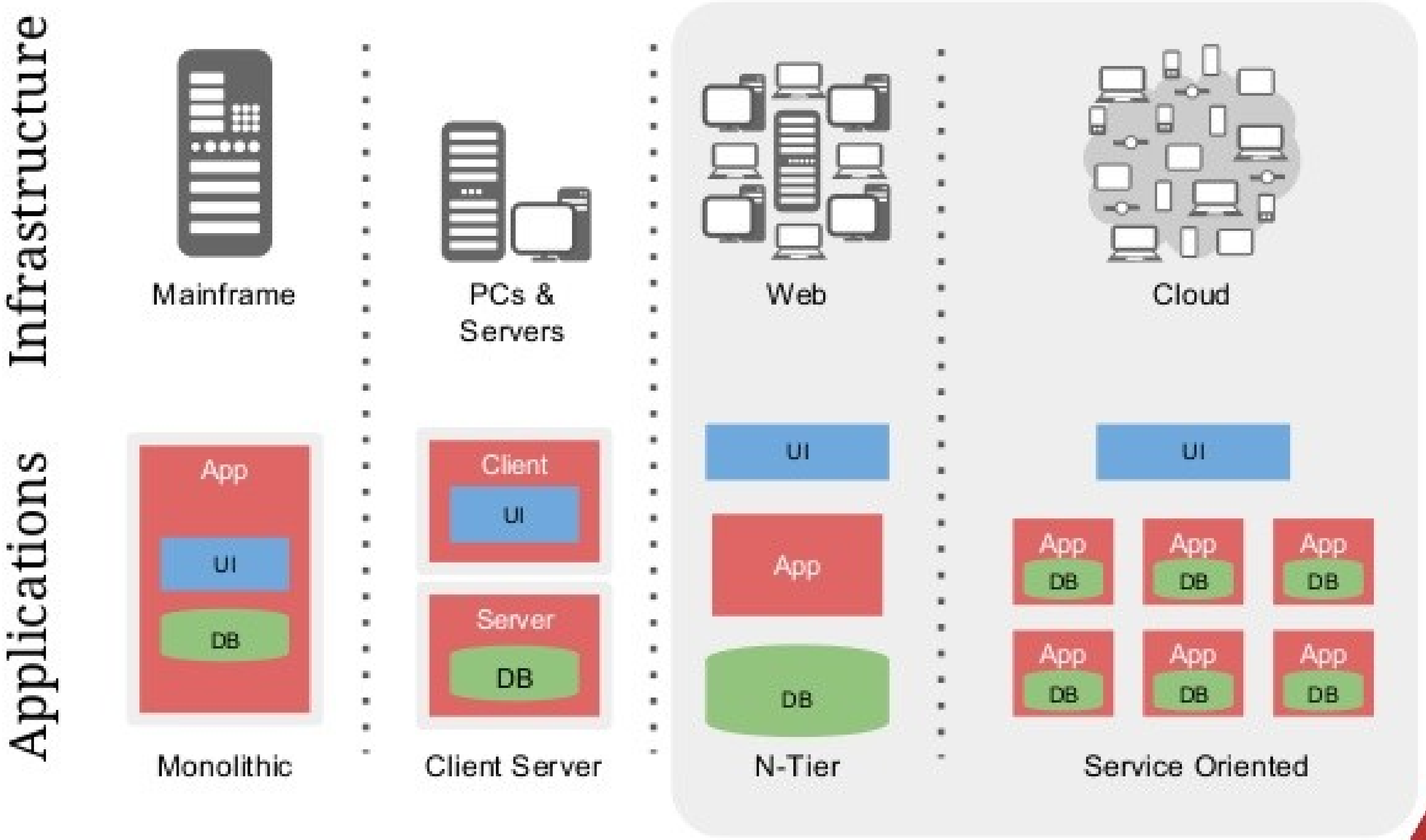
Optional Features

- Strict FIFO (, guaranteed delivery of messages in the right order) or hierarchical organization of messages, priority levels
- Point-to-point: a message read by a destination is no longer available for the others, or Publish-Subscribe : all subscribers to the queue receive a copy of each message (guaranteed delivery: at least once or exactly once)
- message filtering
- encryption/decryption functions, compression/decompression, format transformation
- message retention for offline consumers
- message expiration or validity date
- persistence (on physical media)
- reliability (Ack from MOM to sender and Ack from receiver to MOM)
- transactions
- …

Evolution of MOMs

- 95-2010: Earlier versions
  - 1994: IBM MQSeries (now IBM MQ): pioneer commercial MOM
  - 1994: TIBCO Rendezvous: high performance
  - 1996: Microsoft MSMQ, part of Microsoft Windows Server platform
  - 1998: Oracle MQ, now open source
  - 1999: FioranoMQ: HP for trading and finance
  - 2004: Apache ActiveMQ (open-source, java-based)
  - 2007: RabbitMQ (open-source, Erlang-based)
- 2010: Additional features:
  - 2011: Kafka: HA, replicate…
- 2010’s: Integration with cloud technologies:
  - 2011: Amazon Simple QS
  - 2015: Google Cloud Pub/Sub
  - 2018: IBM Event Stream (based on Kafka), easily integrates with IBM cloud services
  - 2018: Azure Service Bus
  - 2019: CloudAMQP (based on RabbitMQ): automatic scaling

Overview


Infrastructure
Applications
Mainframe
PCs & Servers
Web
Cloud
App
UI
DB
Client
UI
Server
DB
UI
App
DB
UI
App
DB
App
DB
App
DB
App
DB
App
DB
App
DB
Monolithic
Client Server
N-Tier
Service Oriented

Web
services
SOAP
WSDL
Services
service
REST
XML
SOA
RPC
Language
Object
two
support
method
rather
Description
Service-oriented
Service
set
messages
use
often
using
architecture
operations
Transfer
Representational
typically
system
may
RESTful
Architecture
format
based
used
software
client-side
Apache
interface
edit
W3C
HTTP
tools
also
endpoint
supported
Protocol
frameworks
web

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

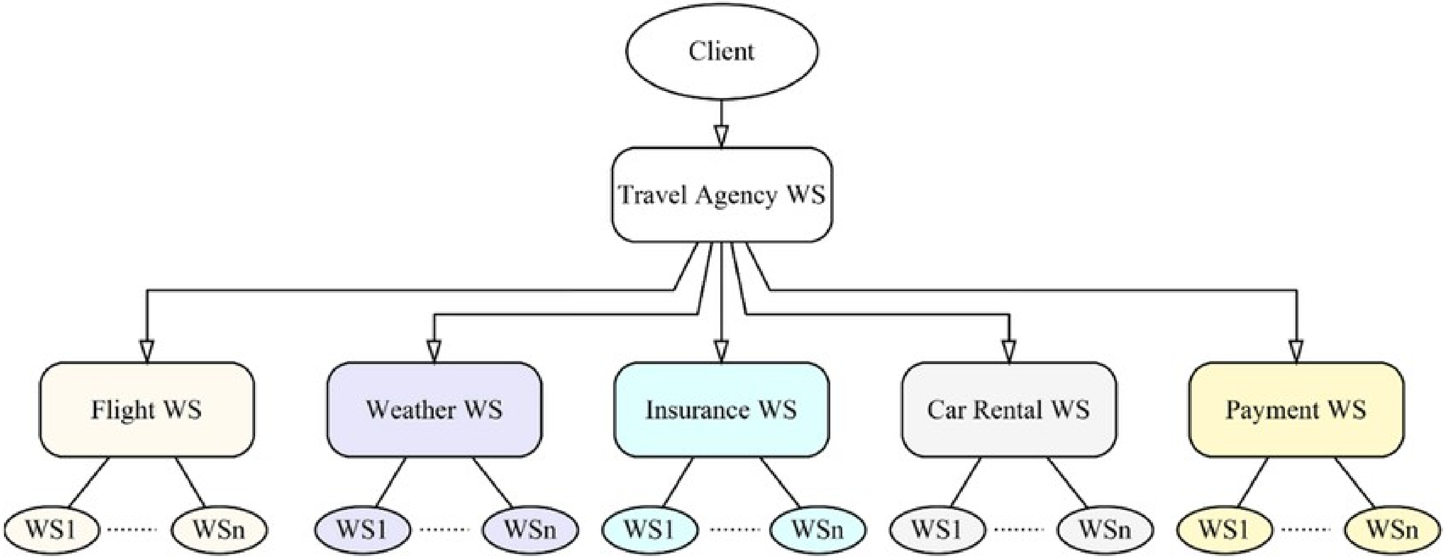
Usage

- Services are used as software libraries to build applications

- 2 contexts:
  - External services
  - Internal services

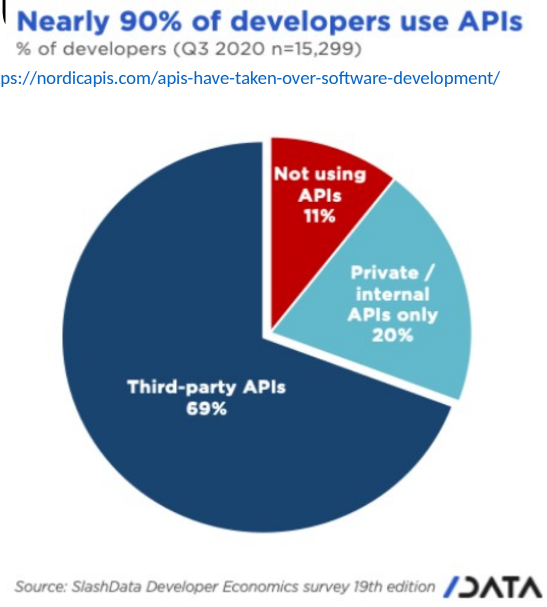
External Services (Partner or Public Services)

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



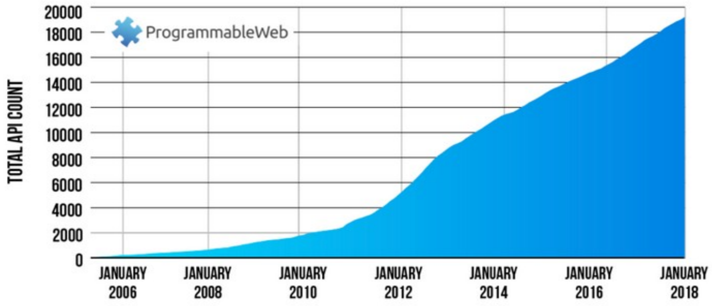
Arun et al.. A novel quality-of-service-aware web services composition using biogeography-based optimization algorithm (2020)

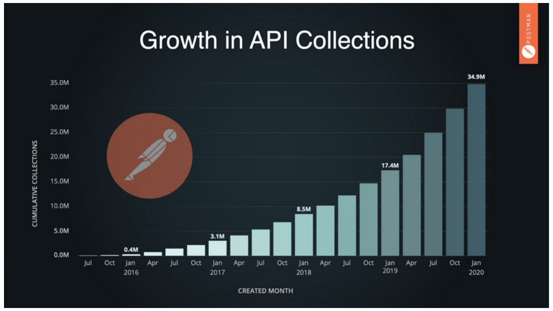
API becomes more of a priority than UI



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

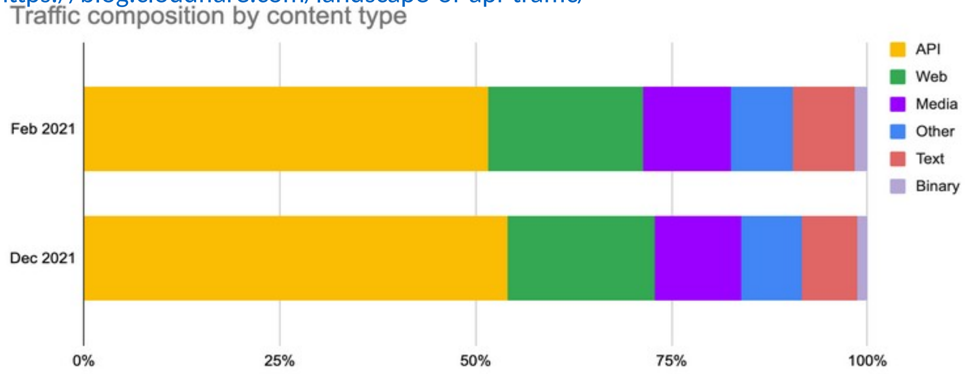
Source: SlashData Developer Economics survey 19th edition





Cloudflare traffic: API use in 2021

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



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

Community

How APIs became the building blocks for software

Iddo Gino, Rap

@iddogino

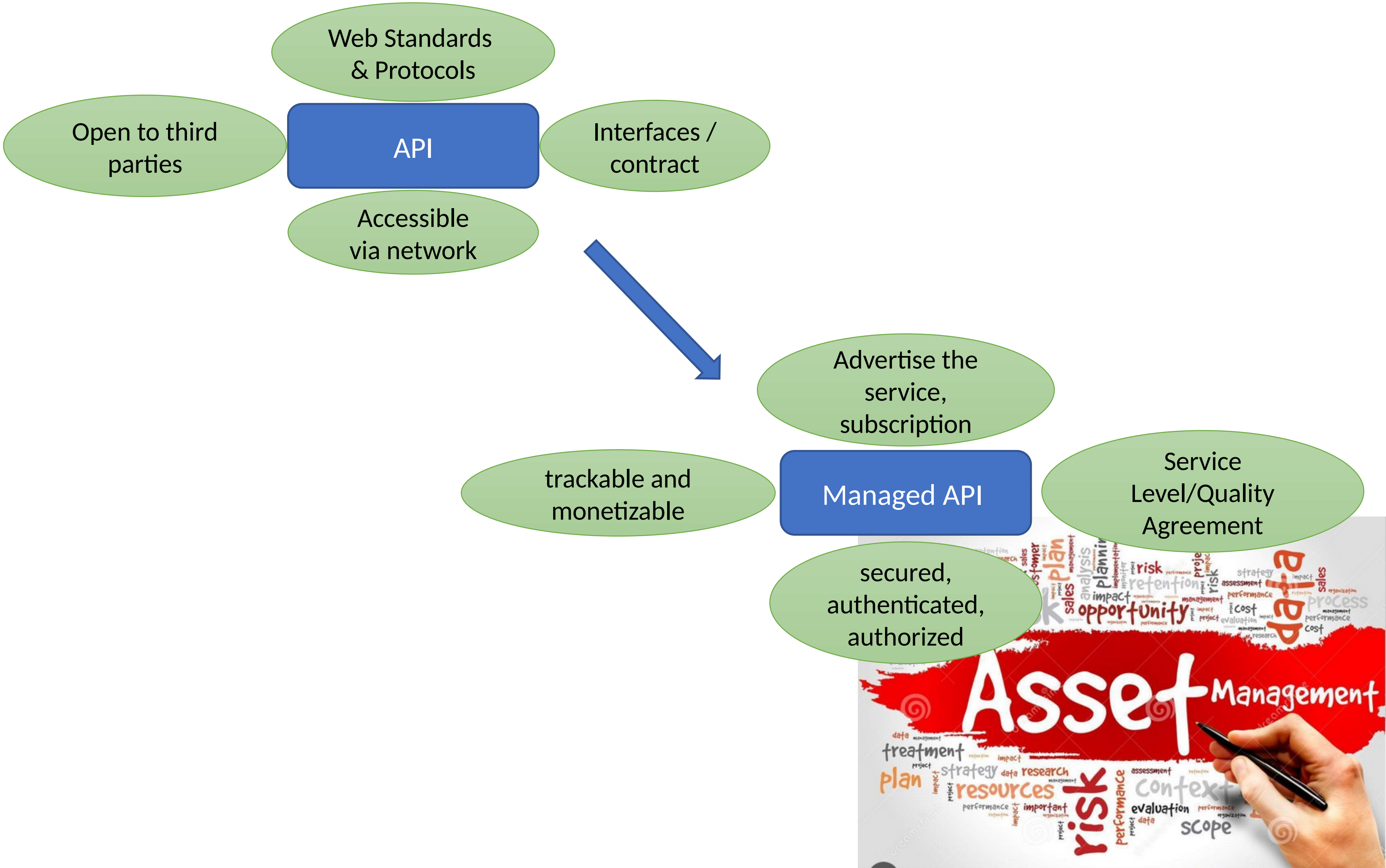
June 23, 2022

API-First As The Norm

Adopting an API-first strategy will be increasingly common in the future. In fact, Postman's 202
found that 39.2% of teams have already designed and defined APIs and schema before they ever

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

From Basic WS to Managed API



Web Standards & Protocols

Open to third parties

API

Interfaces / contract

Accessible via network

Advertise the service, subscription

trackable and monetizable

Managed API

Service Level/Quality Agreement

secured, authenticated, authorized

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

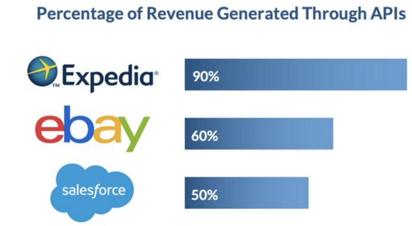
- Take advantage of third-party data or programs without having to:
  - develop, test, update and maintain code
  - acquire and maintain a hosting infrastructure
- Easily compose services and replace one component by an alternative

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



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

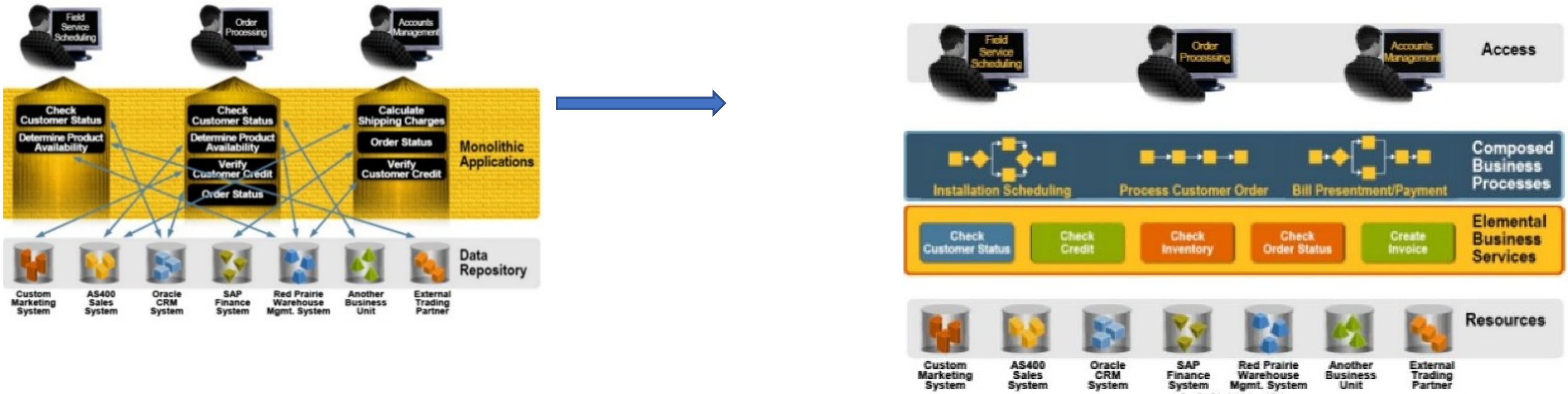
- Benefits
  - Increases revenue
  - Extends customer reach
    - New form of marketing : B2D “business to developer”
  - Stimulates innovation
- Risks
  - Decreases ad revenue
  - No more control on the final user’s experience

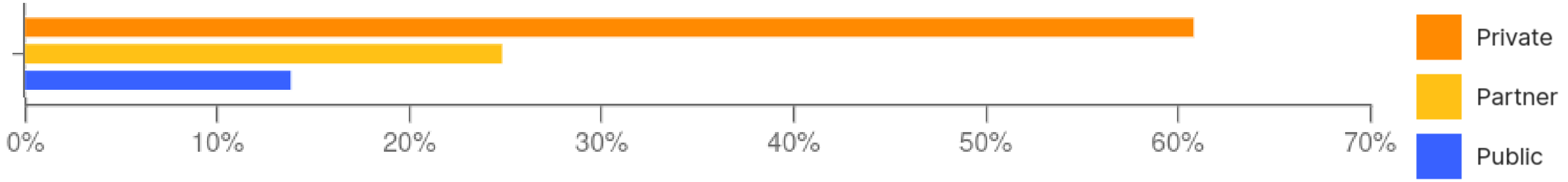
Internal Services (Private Services)

- access restricted to the organization

Monolithic Systems

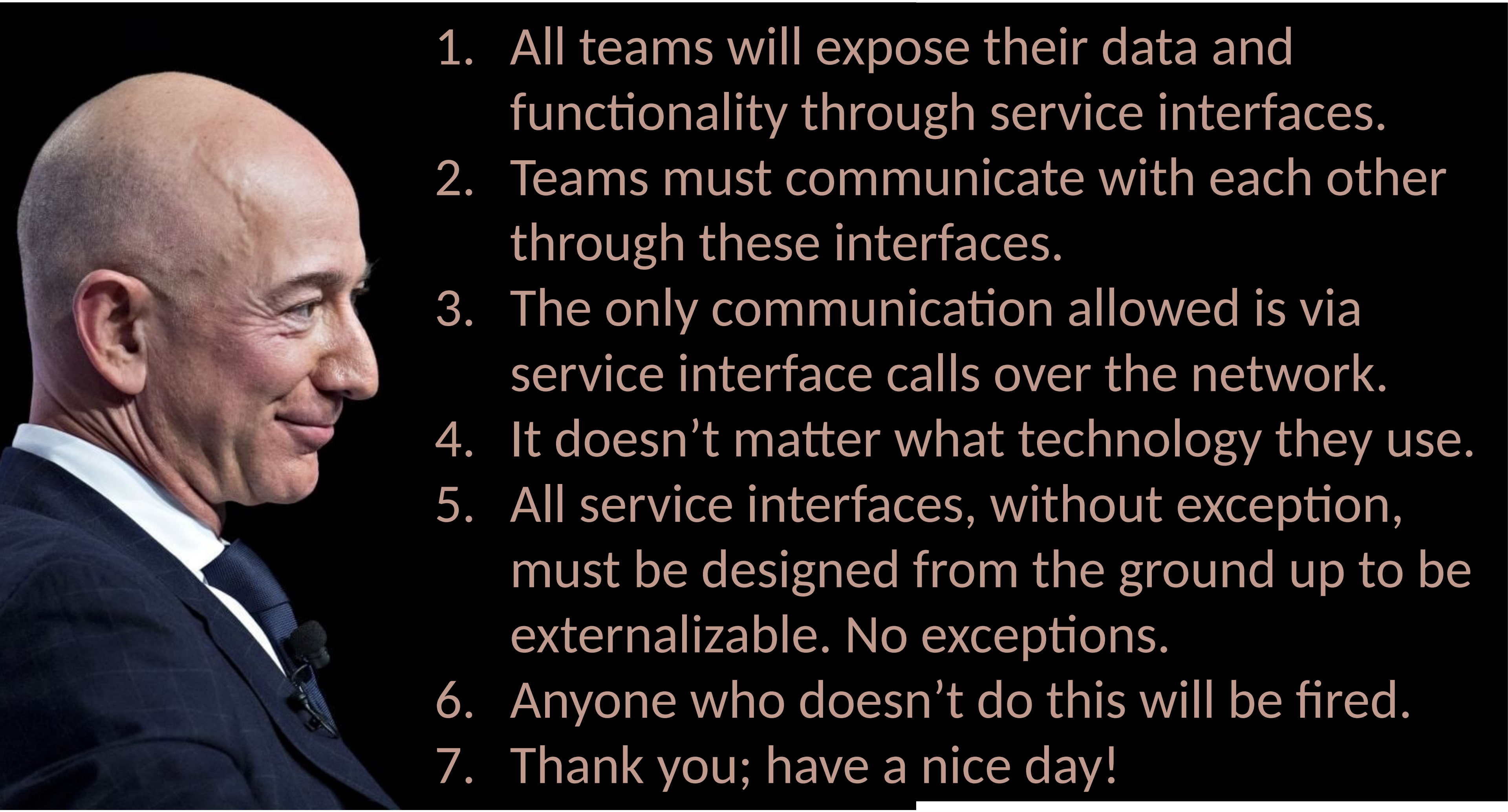
Reuse Services via Re-composition





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!

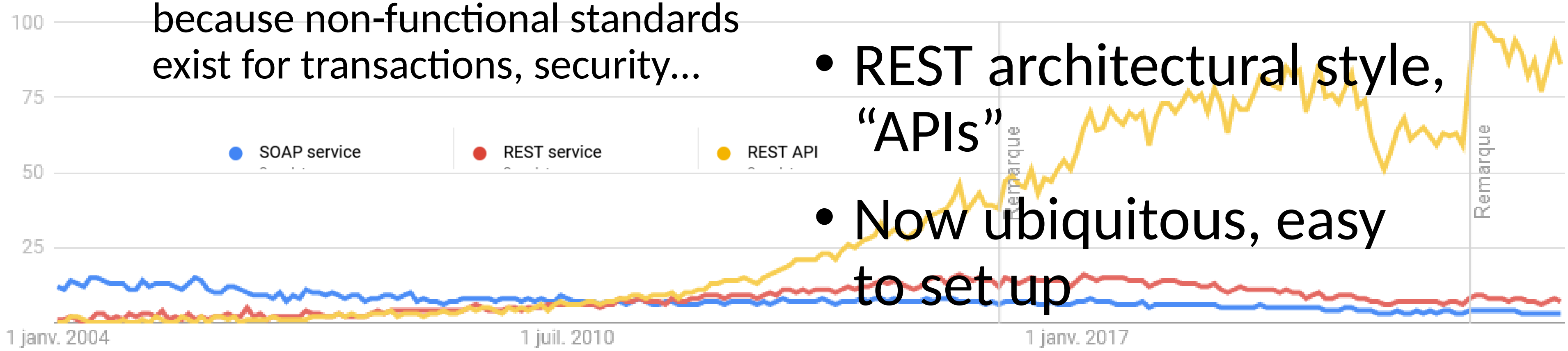
2 “Flavors” of Web Services

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



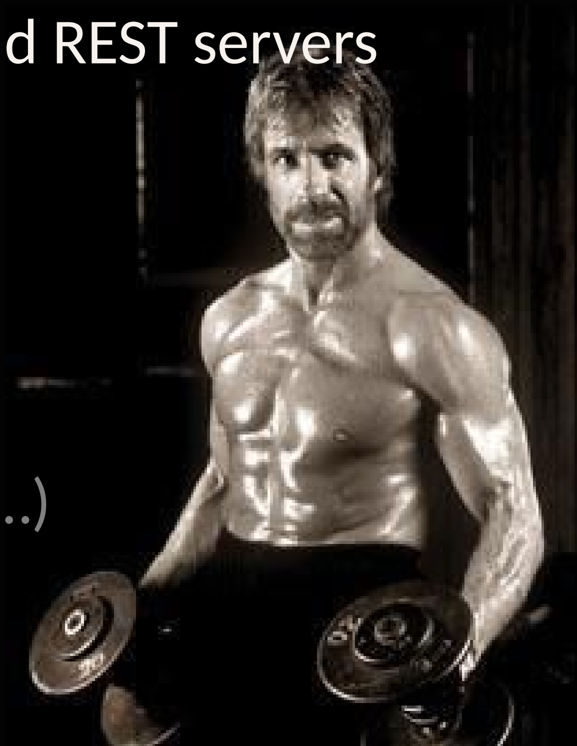
After this Course…

You will be able to design, set up and take advantage of a Service-Oriented Architecture

- find Web Services and understand their interfaces, including GraphQL
- write well-designed and documented APIs
- implement in Python and deploy on the cloud REST servers
- write Python clients
- cite several Chuck Norris’s facts

Not covered:

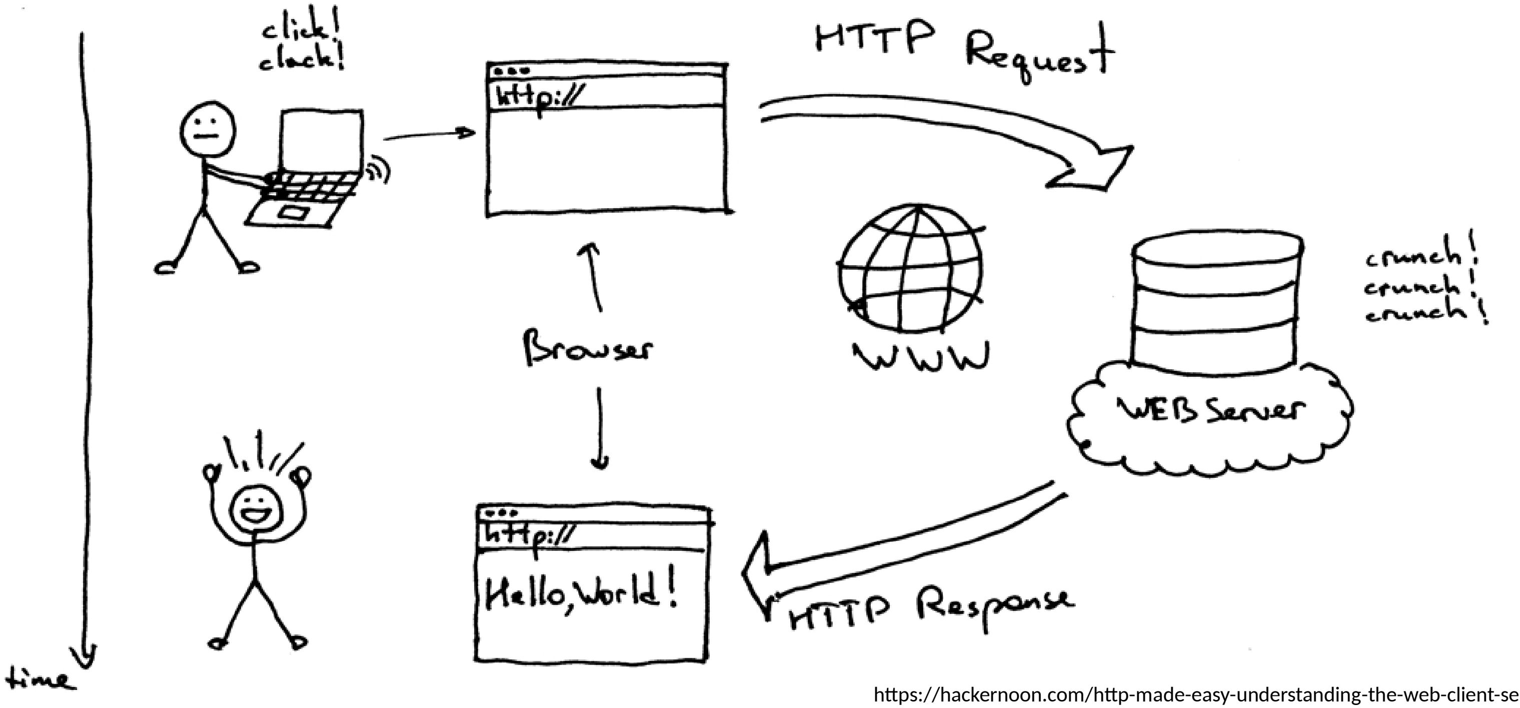
- SOAP and WSDL
- DevOps (deployment, mock tests, load tests…)
- security
- scripted composition of services



http://www.

The Web Services Protocol

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

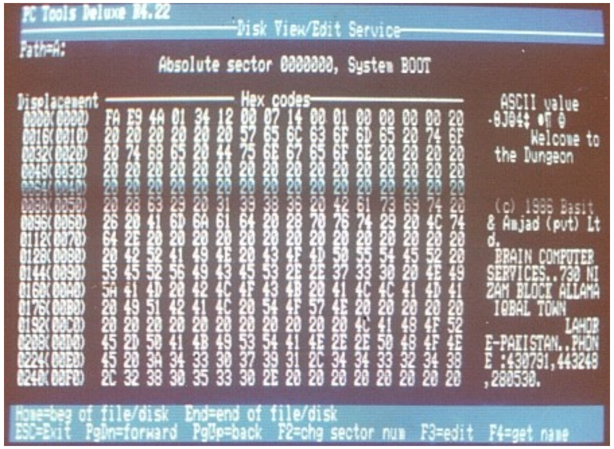


https://hackernoon.com/http-made-easy-understanding-the-web-client-server-communication-yz783vg3

When was HTTP first specified?

1981

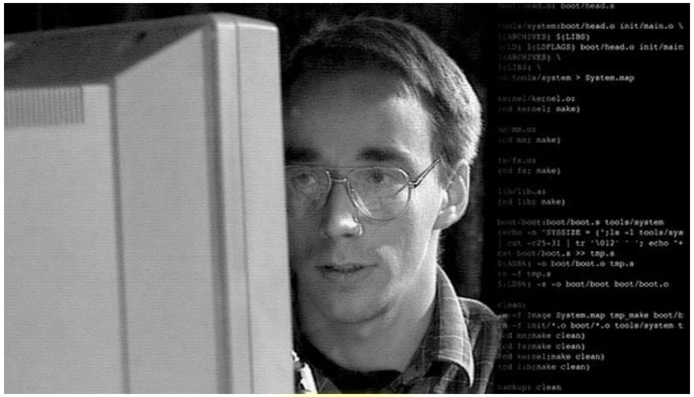
(IBM PC 5150)



```
PC Tools Deluxe R4.22
                         Disk View/Edit Service
Path=A:
                    Absolute sector 0000000, System BOOT

Displacement  ------------- Hex codes ----------------------   ASCII value
 0000(0000)  FA E9 4A 01 34 12 00 07 14 00 01 00 00 00 00 20  ·ΘJ◘4‡ ●¶ ◘
 0016(0010)  20 20 20 20 20 20 57 65 6C 63 6F 6D 65 20 74 6F        Welcome to
 0032(0020)  20 74 68 65 20 44 75 6E 67 65 6F 6E 20 20 20 20   the Dungeon
 0048(0030)  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 0064(0040)  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 0080(0050)  20 28 63 29 20 31 39 38 36 20 42 61 73 69 74 20   (c) 1986 Basit
 0096(0060)  26 20 41 6D 6A 61 64 20 28 70 76 74 29 20 4C 74  & Amjad (pvt) Lt
 0112(0070)  64 2E 20 20 20 20 20 20 20 20 20 20 20 20 20 20  d.
 0128(0080)  20 42 52 41 49 4E 20 43 4F 4D 50 55 54 45 52 20   BRAIN COMPUTER
 0144(0090)  53 45 52 56 49 43 45 53 2E 2E 37 33 30 20 4E 49  SERVICES..730 NI
 0160(00A0)  5A 41 4D 20 42 4C 4F 43 4B 20 41 4C 4C 41 4D 41  ZAM BLOCK ALLAMA
 0176(00B0)  20 49 51 42 41 4C 20 54 4F 57 4E 20 20 20 20 20   IQBAL TOWN
 0192(00C0)  20 20 20 20 20 20 20 20 20 20 20 4C 41 48 4F 52             LAHOR
 0208(00D0)  45 2D 50 41 4B 49 53 54 41 4E 2E 2E 50 48 4F 4E  E-PAKISTAN..PHON
 0224(00E0)  45 20 3A 34 33 30 37 39 31 2C 34 34 33 32 34 38  E :430791,443248
 0240(00F0)  2C 32 38 30 35 33 30 2E 20 20 20 20 20 20 20 20  ,280530.

Home=beg of file/disk  End=end of file/disk
ESC=Exit  PgDn=forward  PgUp=back  F2=chg sector num  F3=edit  F4=get name
```

1986

(Brain: first computer virus for MS-DOS)



```
boot/head.o: boot/head.s

tools/system:boot/head.o init/main.o \
$(ARCHIVES) $(LIBS)
$(LD) $(LDFLAGS) boot/head.o init/main
$(ARCHIVES) \
$(LIBS) \
-o tools/system > System.map

kernel/kernel.o:
(cd kernel; make)

mm/mm.o:
(cd mm; make)

fs/fs.o:
(cd fs; make)

lib/lib.a:
(cd lib; make)

boot/boot:boot/boot.s tools/system
(echo -n "SYSSIZE = (";ls -l tools/sys
| cut -c25-31 | tr '\012' ' '; echo "*
cat boot/boot.s >> tmp.s
$(AS86) -o boot/boot.o tmp.s
rm -f tmp.s
$(LD86) -s -o boot/boot boot/boot.o

clean:
rm -f Image System.map tmp_make boot/b
rm -f init/*.o boot/*.o tools/system t
(cd mm;make clean)
(cd fs;make clean)
(cd kernel;make clean)
(cd lib;make clean)

backup: clean
```

1991

(Linus Torvalds introduces Linux)

1996

(Google search engine)

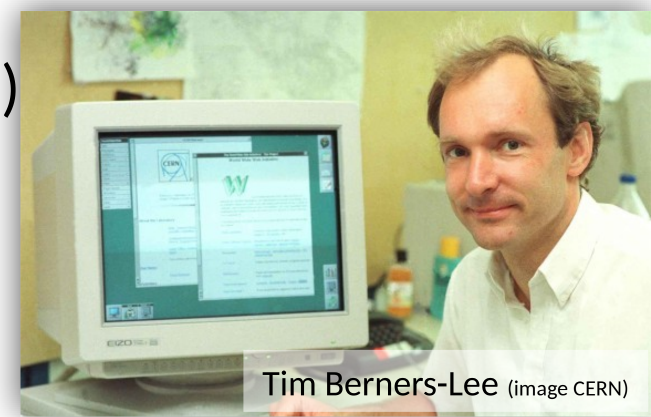
Hypertext Transport Protocol

- 1989-90, Tim Berners-Lee’s problem at CERN:

  how to integrate and exchange information held on different computers in scattered places?

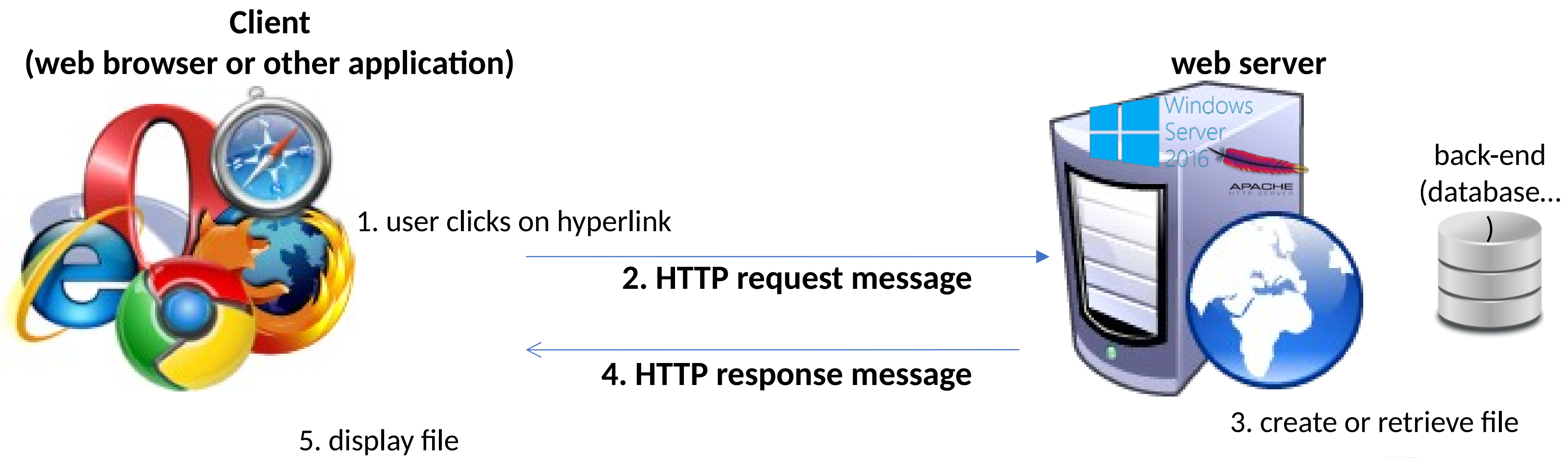
- Already exist:
  - TCP: reliable transport of information on the Internet
  - DNS: domain name (“www.centralesupelec.fr”) ↔ IP @ (“138.195.9.117”)

    Human friendly Computer friendly
  - object in a database that *references* others
- Put them all together: HTTP
  - Retrieve linked documents (resources)
  - Accessible via the Internet



Tim Berners-Lee (image CERN)

Client-Server Protocol request / response



Client
(web browser or other application)

1. user clicks on hyperlink

2. HTTP request message

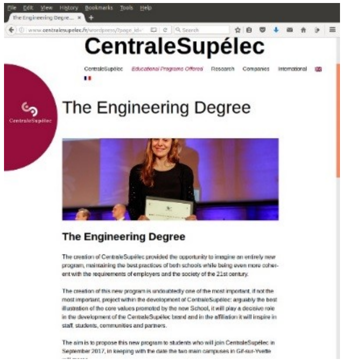
web server

back-end
(database…
)

3. create or retrieve file

4. HTTP response message

5. display file


File Edit View History Bookmarks Tools Help
The Engineering Degre…
www.centralesupelec.fr/wordpress/?page_id=
Search
CentraleSupélec
CentraleSupélec Educational Programs Offered Research Companies International

The Engineering Degree

The Engineering Degree

The creation of CentraleSupélec provided the opportunity to imagine an entirely new program, maintaining the best practices of both schools while being even more coherent with the requirements of employers and the society of the 21st century.

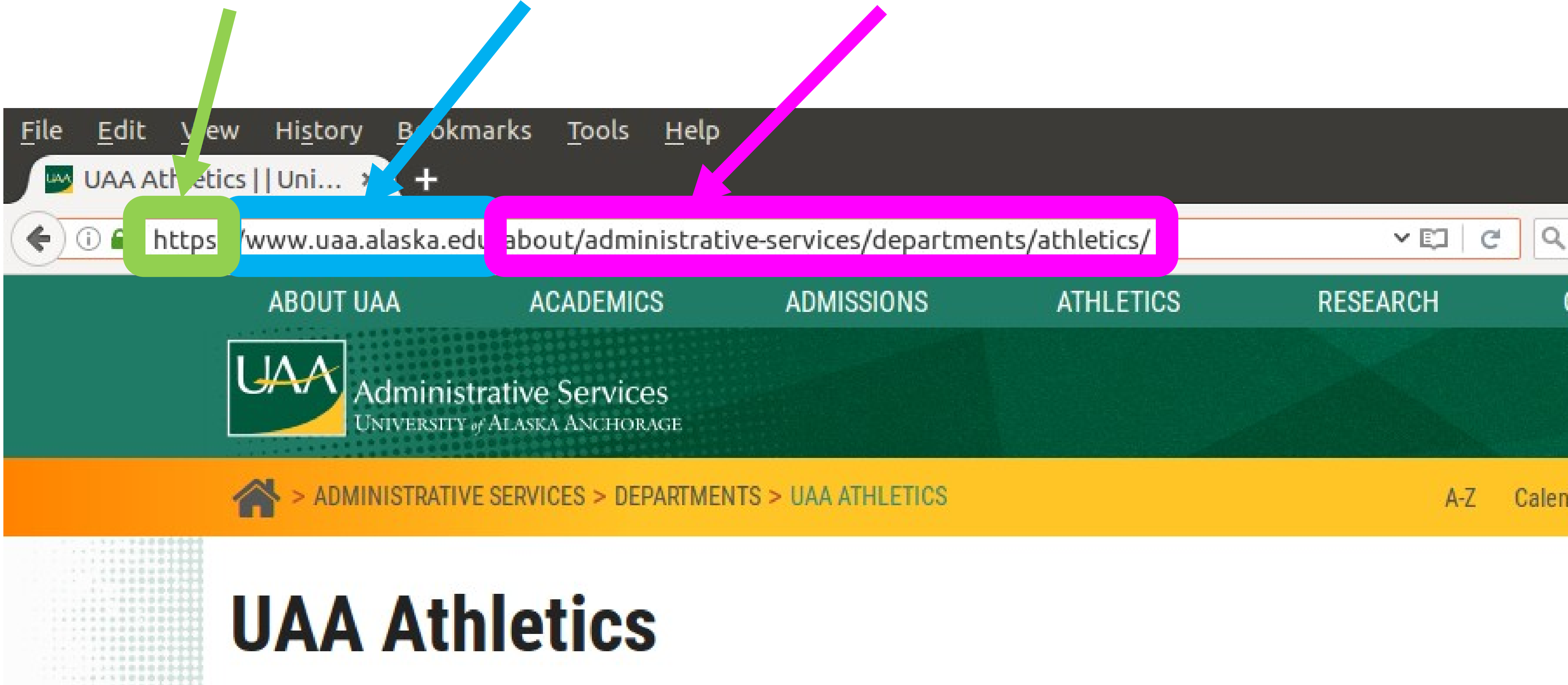
The creation of this new program is undoubtedly one of the most important, if not the most important, project within the development of CentraleSupélec: arguably the best illustration of the core values promoted by the new School, it will play a decisive role in the development of the CentraleSupélec brand and in the affiliation it will inspire in staff, students, communities and partners.

The aim is to propose this new program to students who will join CentraleSupélec in September 2017, in keeping with the date the two main campuses in Gif-sur-Yvette

Resources = addressable files

- Any kind of file: HTML file, JPG image file, binary file…
- URL (Uniform Resource Locator)

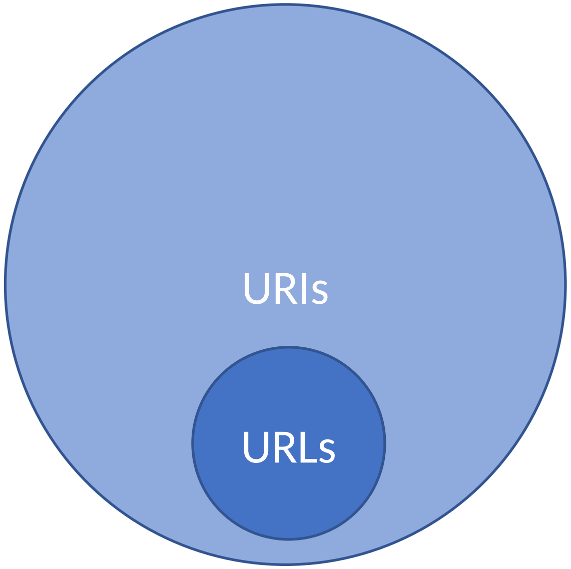
  = protocol + server host name + path on server



https /www.uaa.alaska.edu about/administrative-services/departments/athletics/

Side note about URL and URI

- URI: identifier (name of a restaurant)
- URL: locator (GPS coordinates of the restaurant)



All URLs are URIs:

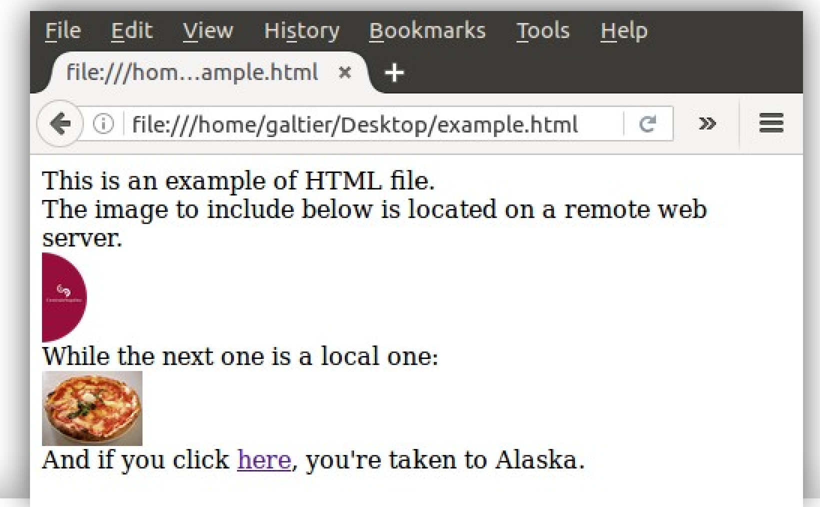
with the GPS coordinates I arrive to the right restaurant

Not all URIs are URLs:

the name of the restaurant gives no information on its location

HTML file may include references to others resources

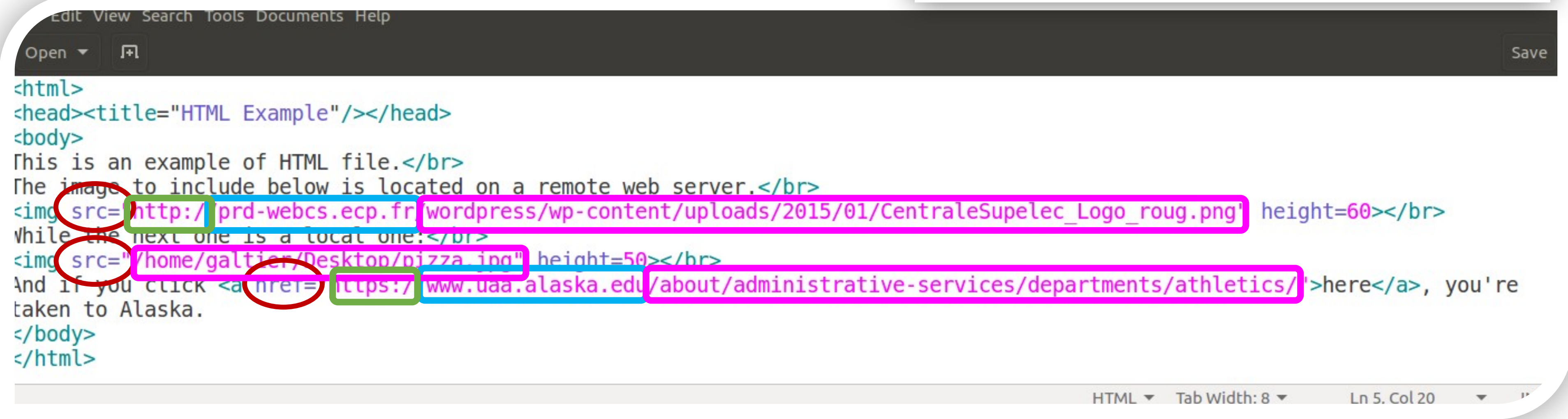
- 3 resources are required to display this web page:
  - HTML file
  - CentraleSupelec logo image
  - Pizza image


File Edit View History Bookmarks Tools Help
file:///hom...ample.html ×
+
file:///home/galtier/Desktop/example.html

This is an example of HTML file.
The image to include below is located on a remote web server.

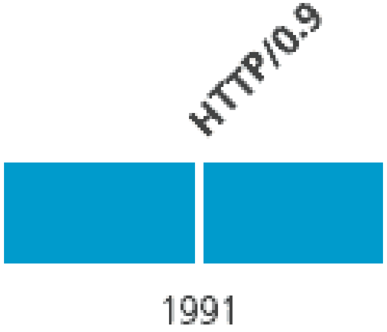
While the next one is a local one:

And if you click here, you're taken to Alaska.

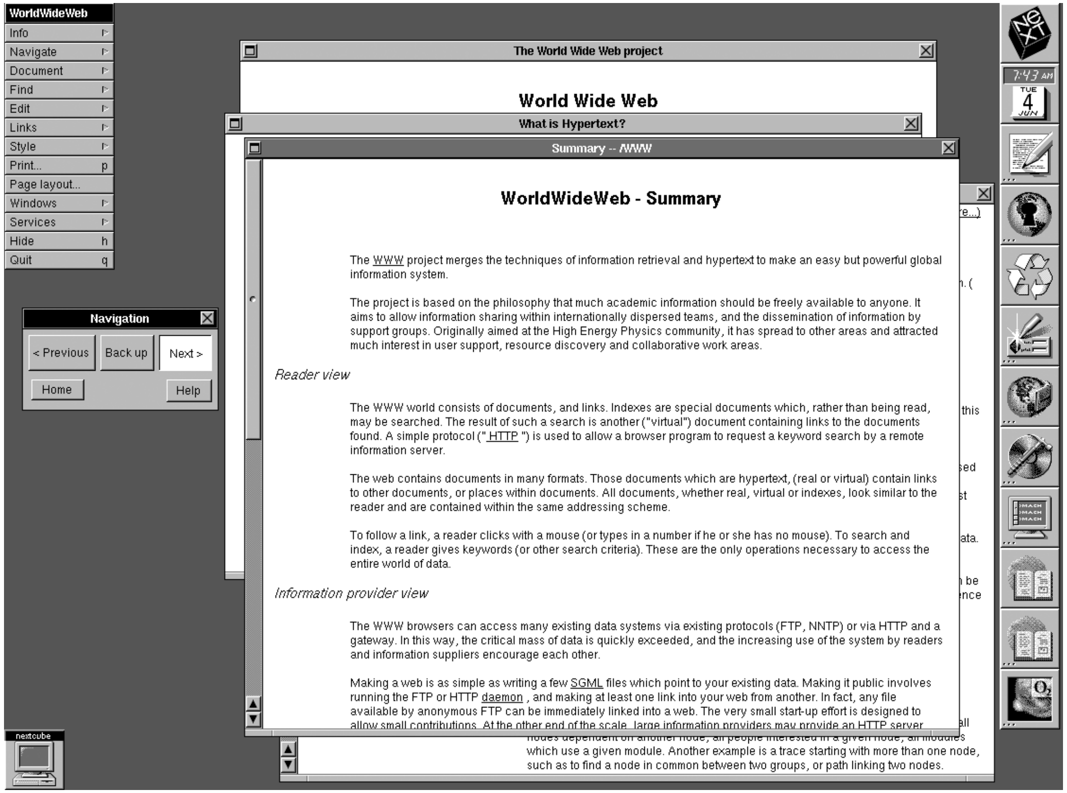


```
<html>
<head><title="HTML Example"/></head>
<body>
This is an example of HTML file.</br>
The image to include below is located on a remote web server.</br>
<img src="http://prd-webcs.ecp.fr/wordpress/wp-content/uploads/2015/01/CentraleSupelec_Logo_roug.png" height=60></br>
While the next one is a local one:</br>
<img src="/home/galtier/Desktop/pizza.jpg" height=50></br>
And if you click <a href="https://www.uaa.alaska.edu/about/administrative-services/departments/athletics/">here</a>, you're
taken to Alaska.
</body>
</html>
```

HTTP Versions



- 1991 – v0.9
  - First documented version
  - First web browser



WorldWideWeb - Summary

The WWW project merges the techniques of information retrieval and hypertext to make an easy but powerful global information system.

The project is based on the philosophy that much academic information should be freely available to anyone. It aims to allow information sharing within internationally dispersed teams, and the dissemination of information by support groups. Originally aimed at the High Energy Physics community, it has spread to other areas and attracted much interest in user support, resource discovery and collaborative work areas.

Reader view

The WWW world consists of documents, and links. Indexes are special documents which, rather than being read, may be searched. The result of such a search is another ("virtual") document containing links to the documents found. A simple protocol (" HTTP ") is used to allow a browser program to request a keyword search by a remote information server.

The web contains documents in many formats. Those documents which are hypertext, (real or virtual) contain links to other documents, or places within documents. All documents, whether real, virtual or indexes, look similar to the reader and are contained within the same addressing scheme.

To follow a link, a reader clicks with a mouse (or types in a number if he or she has no mouse). To search and index, a reader gives keywords (or other search criteria). These are the only operations necessary to access the entire world of data.

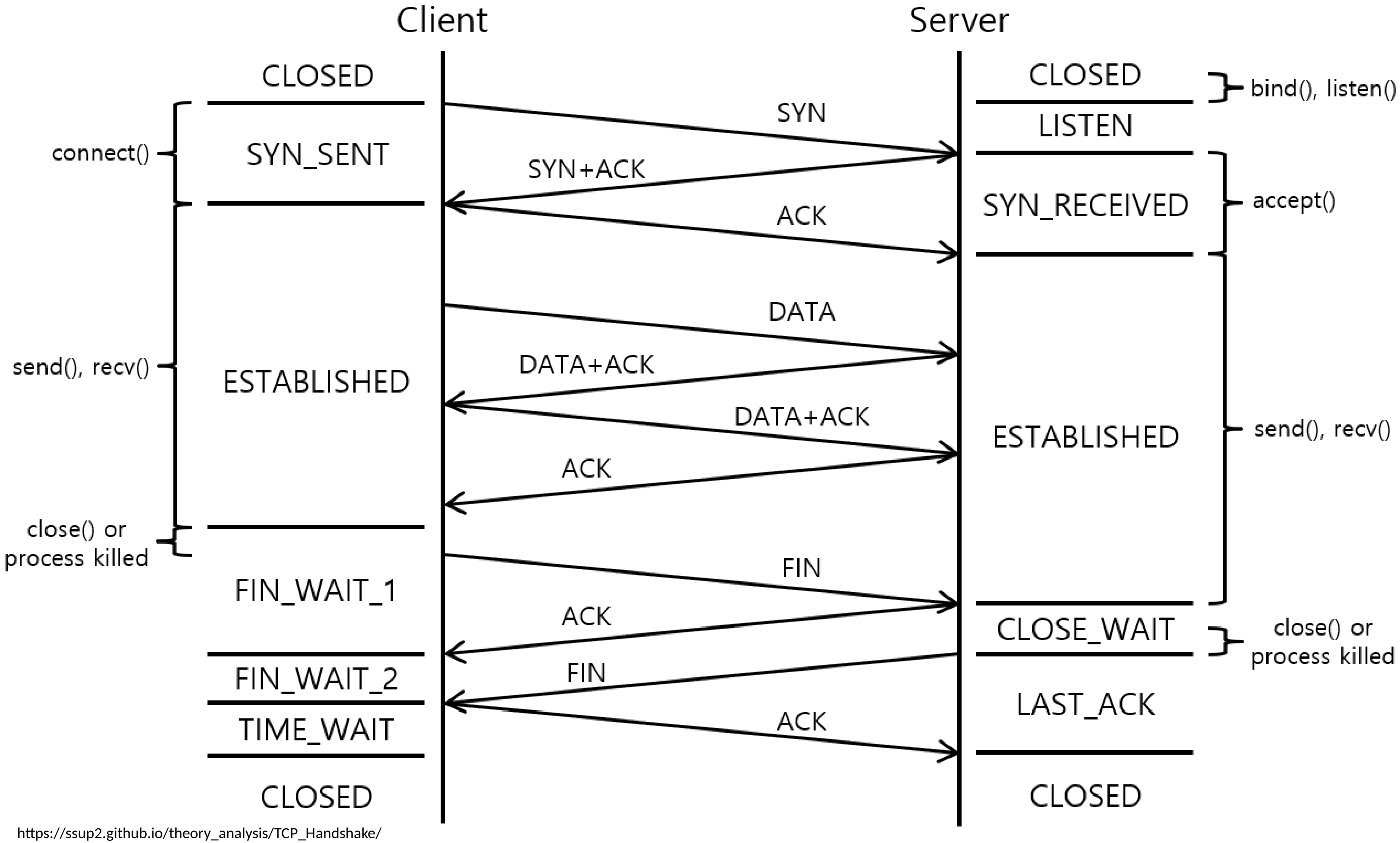
Information provider view

The WWW browsers can access many existing data systems via existing protocols (FTP, NNTP) or via HTTP and a gateway. In this way, the critical mass of data is quickly exceeded, and the increasing use of the system by readers and information suppliers encourage each other.

Making a web is as simple as writing a few SGML files which point to your existing data. Making it public involves running the FTP or HTTP daemon , and making at least one link into your web from another. In fact, any file available by anonymous FTP can be immediately linked into a web. The very small start-up effort is designed to allow small contributions. At the other end of the scale, large information providers may provide an HTTP server

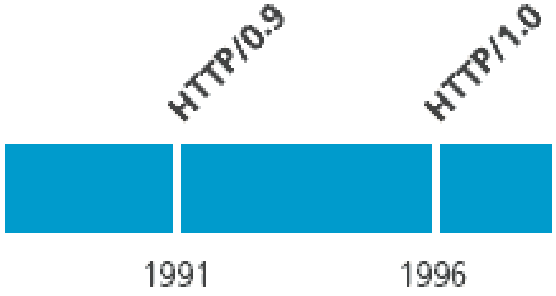
which use a given module. Another example is a trace starting with more than one node, such as to find a node in common between two groups, or path linking two nodes.

Side note on TCP 3-way handshake

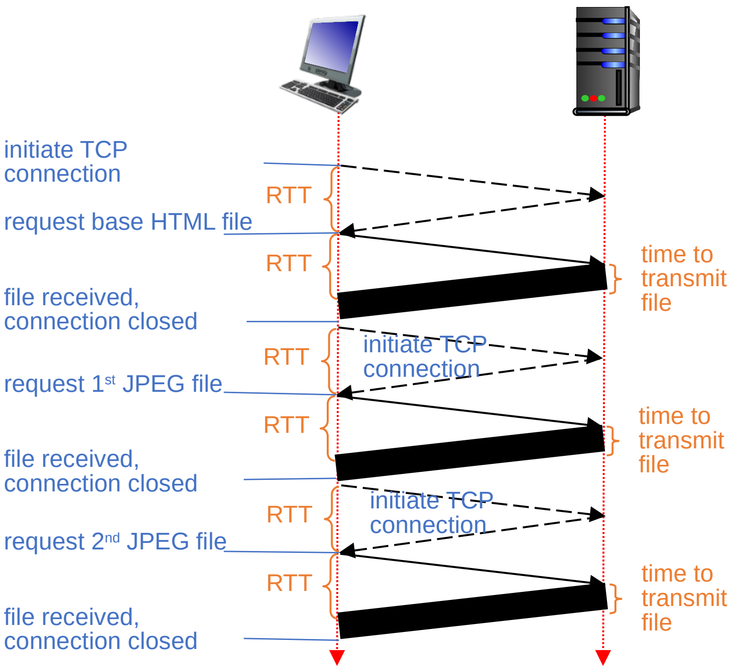


https://ssup2.github.io/theory_analysis/TCP_Handshake/

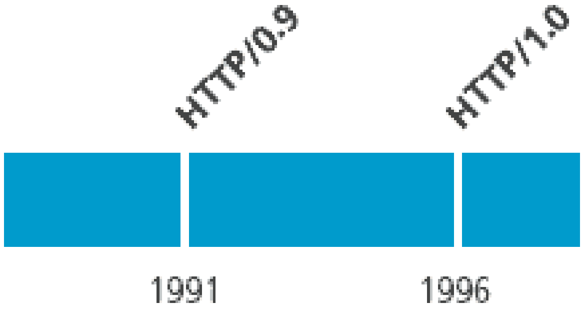
HTTP Versions



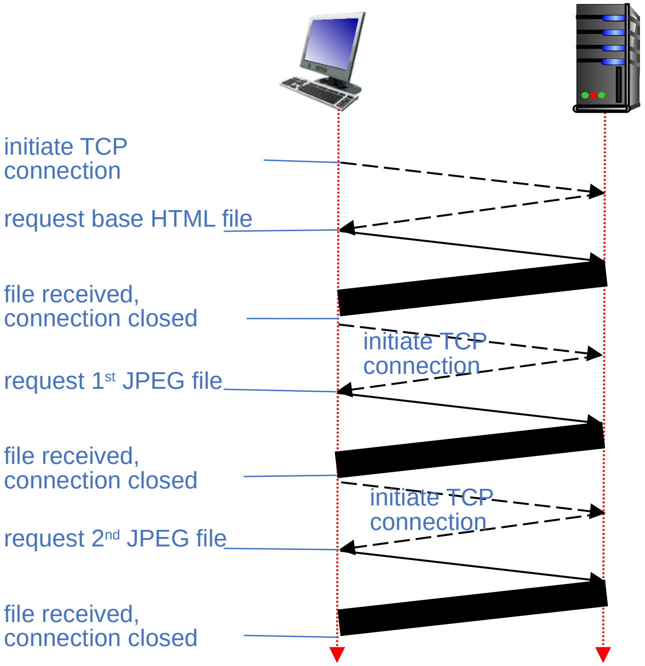
- 1996 – v1.0
    - One TCP connection per resource



HTTP Versions



- 1996 – v1.0
  - One TCP connection per resource



initiate TCP connection

request base HTML file

file received, connection closed

initiate TCP connection

request 1st JPEG file

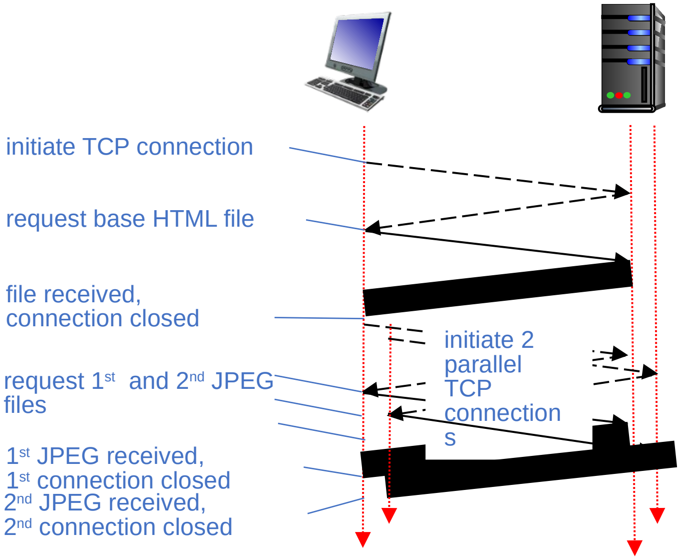
file received, connection closed

initiate TCP connection

request 2nd JPEG file

file received, connection closed

with parallel connections



initiate TCP connection

request base HTML file

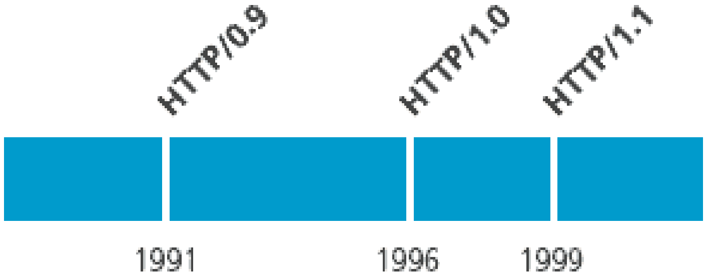
file received, connection closed

initiate 2 parallel TCP connections

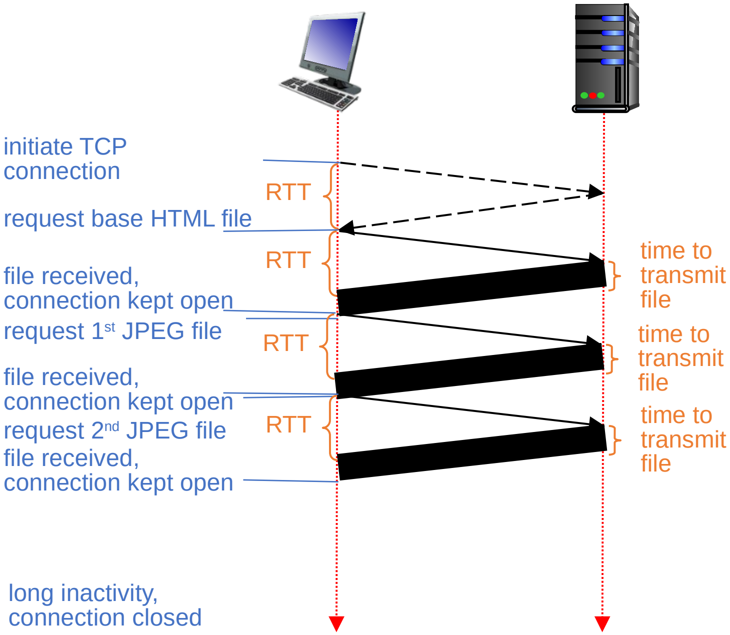
request 1st and 2nd JPEG files

1st JPEG received, 1st connection closed
2nd JPEG received, 2nd connection closed

HTTP Versions

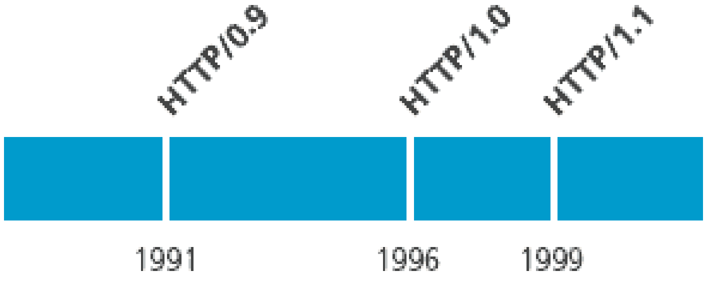


- 1999 – v1.1
  - Persistent connection

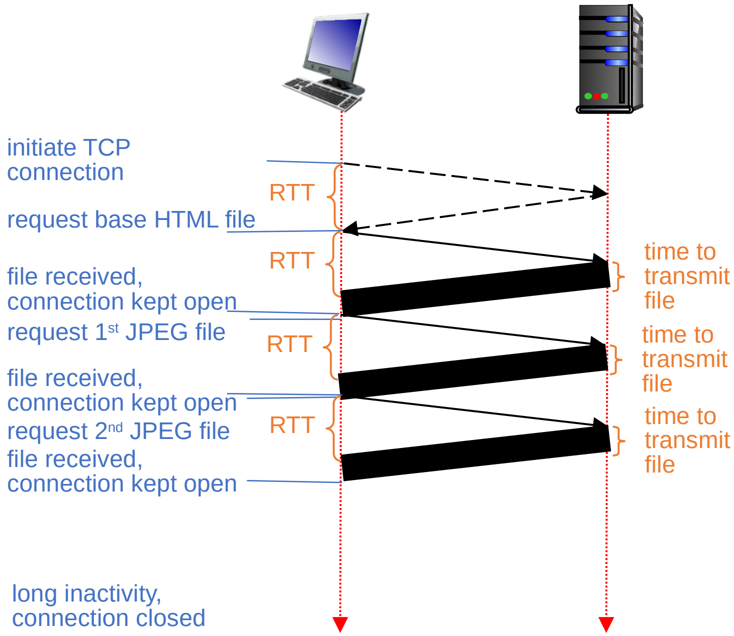


```
GET https://www.centralesupelec.fr/ HTTP/1.1
Host: www.centralesupelec.fr
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86
Accept: text/html,application/xhtml+xml,applica
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

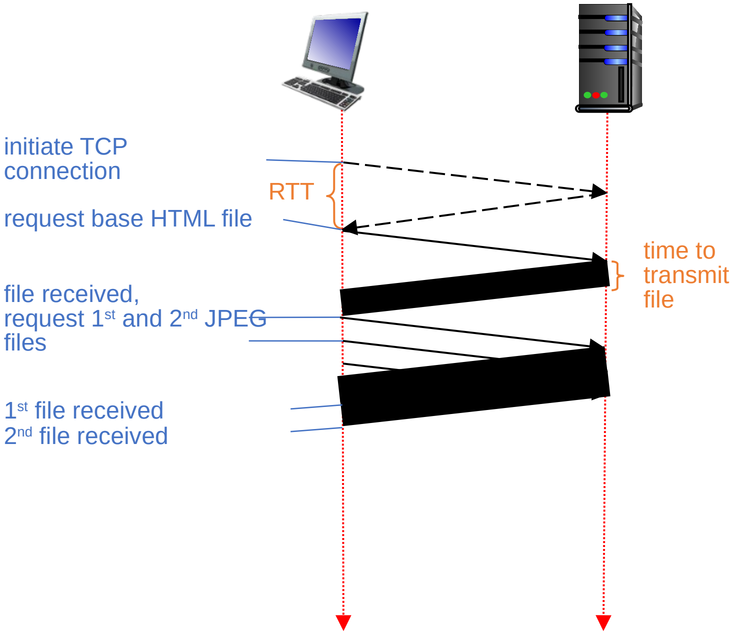
HTTP Versions



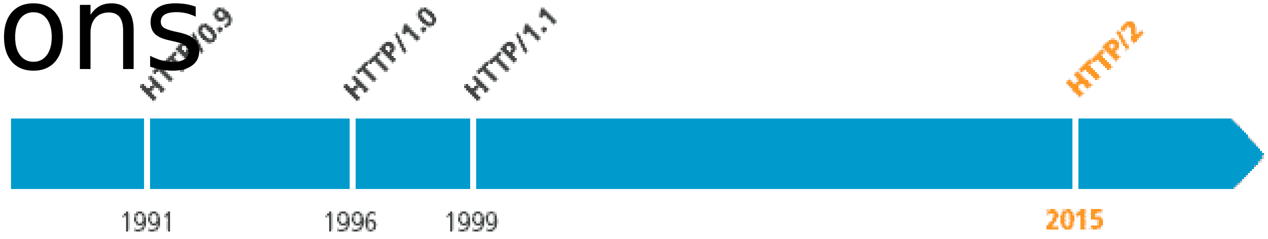
- 1999 – v1.1
  - Persistent connection



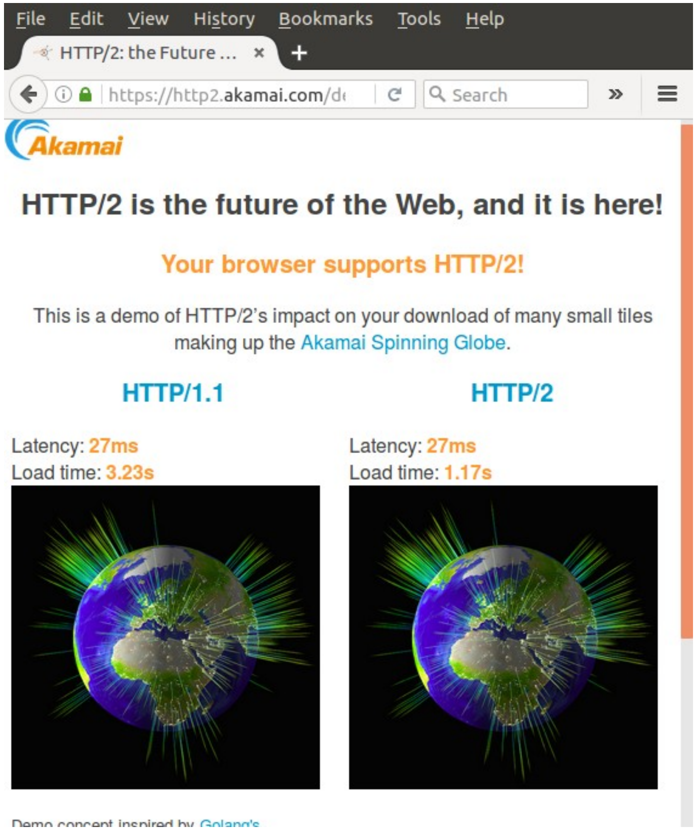
with pipelining



HTTP Versions



- 2015 – v2
  - Server “pushes” content
  - [and other optimizations]


File Edit View History Bookmarks Tools Help
HTTP/2: the Future ...
https://http2.akamai.com/de
Search
Akamai

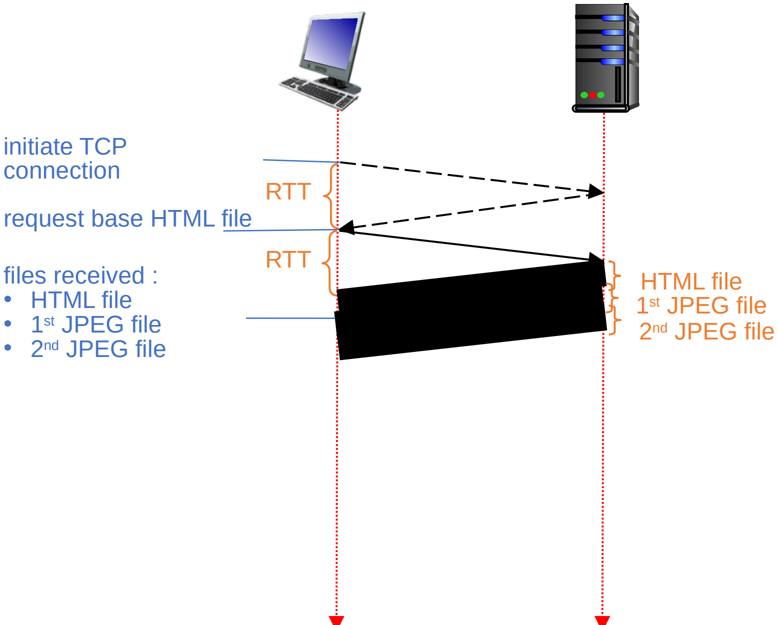
HTTP/2 is the future of the Web, and it is here!

Your browser supports HTTP/2!

This is a demo of HTTP/2’s impact on your download of many small tiles making up the Akamai Spinning Globe.

HTTP/1.1	HTTP/2
Latency: 27ms Load time: 3.23s	Latency: 27ms Load time: 1.17s

Demo concept inspired by Golang's



Optional Reading Exercise

- Find the document which describes HTTP/2.
- What is the “head-of-line blocking” (HOL blocking) problem observed in HTTP/1.1?
- Read the beginning of the FAQ at https://http2.github.io/faq/

Reading: Results

- HTTP/2 is defined in RFC 7540.
- HOL blocking:
  - Imagine a HTTP client that sends to a server 2 requests over the same TCP connection, and that the first response is "large" in content length while the second response is "small" in content length.
  - Due to the nature of the HTTP 1.x protocol, the second response must wait for the first response to complete: the second response is *head-of-line blocked* by the first response.
  - HTTP/2 is fully multiplexed (instead of ordered and blocking), allowing multiple request and response messages to be in flight at the same time (it’s even possible to intermingle parts of one message with another on the wire).

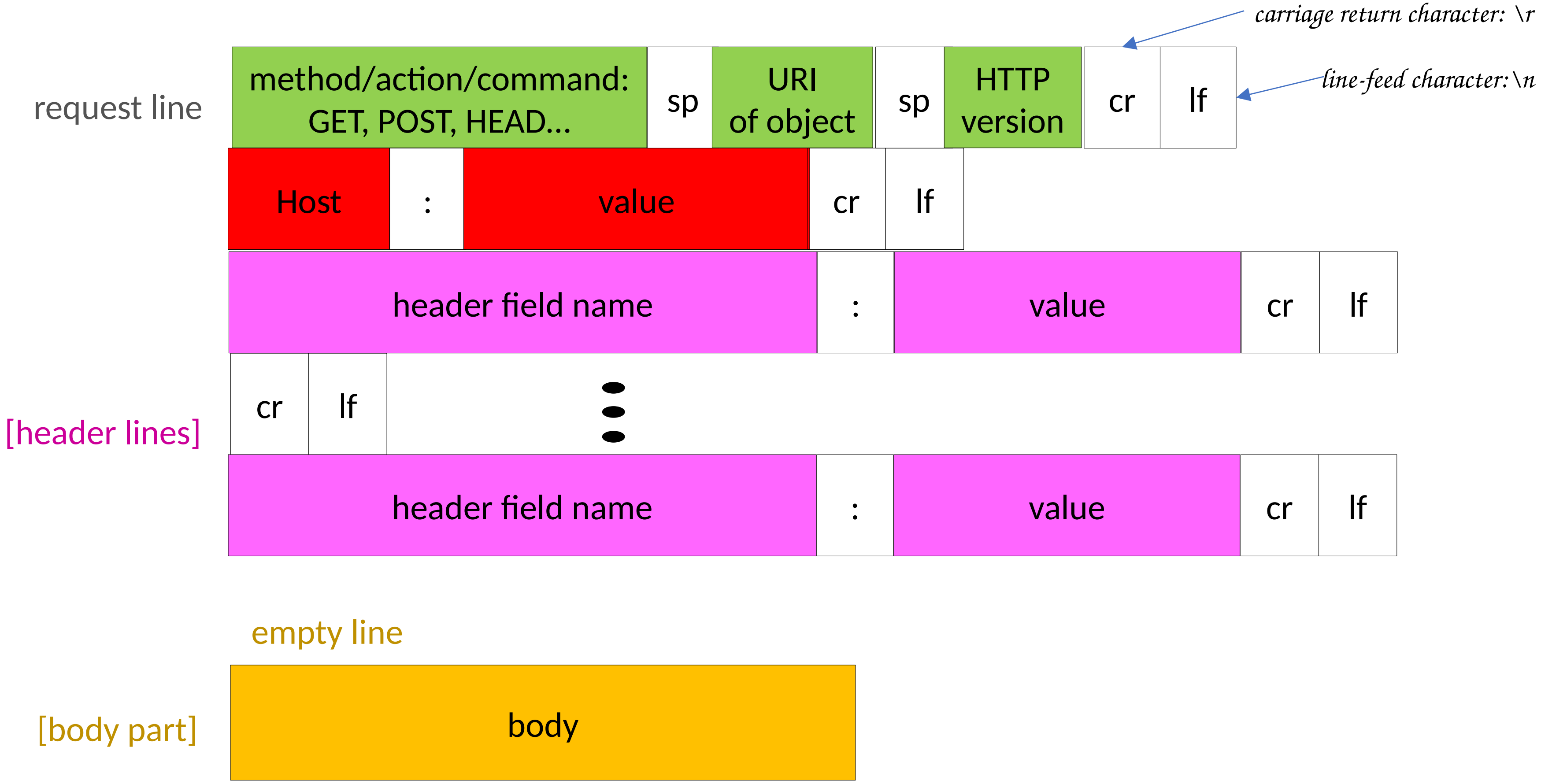
HTTP Messages

- 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



carriage return character: \r

line-feed character:\n

request line

method/action/command: GET, POST, HEAD… | sp | URI of object | sp | HTTP version | cr | lf

Host | : | value | cr | lf

header field name | : | value | cr | lf

cr | lf

⋮

[header lines]

header field name | : | value | cr | lf

empty line

[body part]

body

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
```

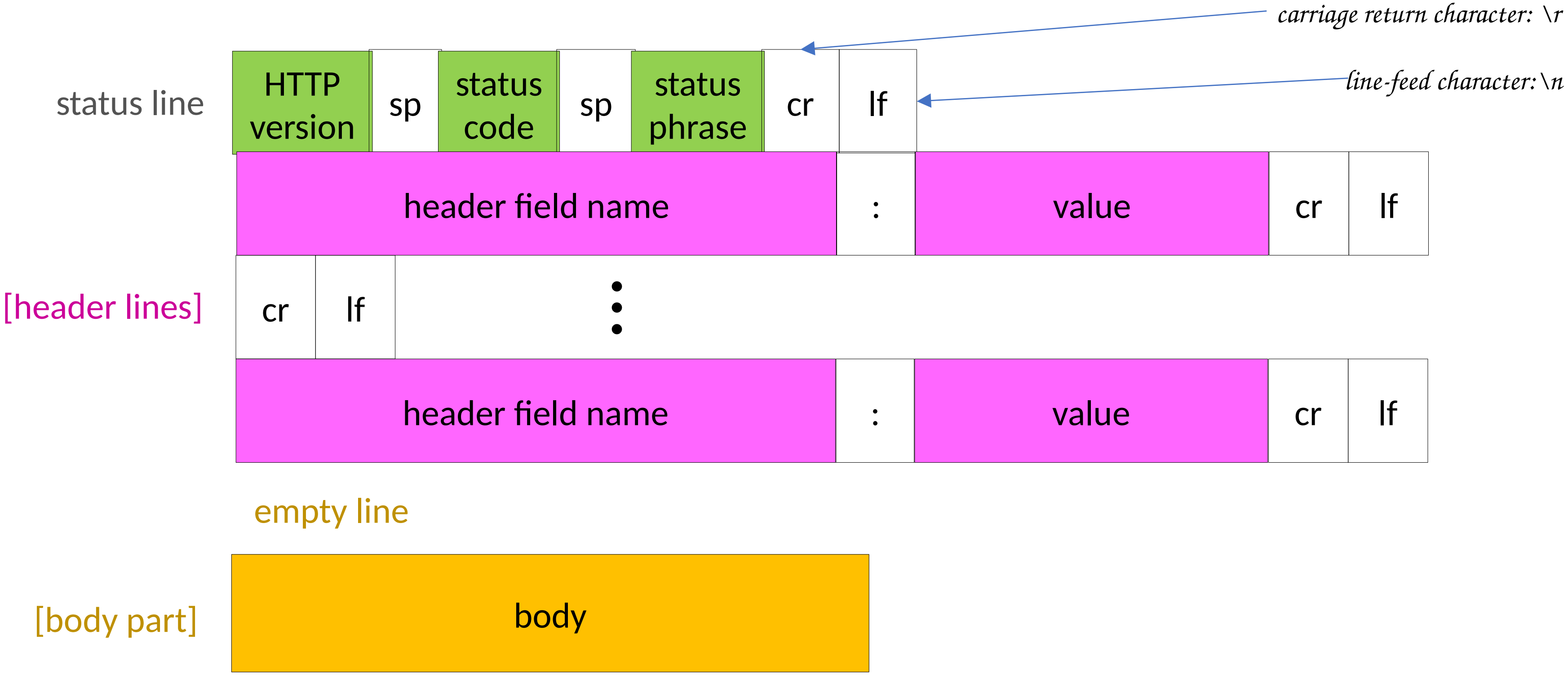
Request Parameters

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



carriage return character: \r

line-feed character:\n

status line

HTTP version | sp | status code | sp | status phrase | cr | lf

header field name | : | value | cr | lf

[header lines]

cr | lf

⋮

header field name | : | value | cr | lf

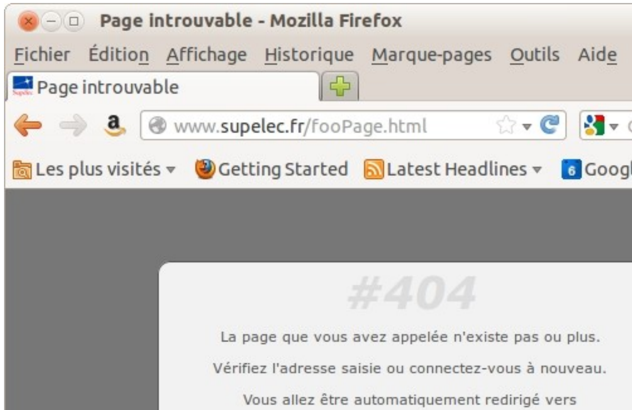
empty line

[body part]

body

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



www.supelec.fr/fooPage.html

#404

La page que vous avez appelée n'existe pas ou plus.

Vérifiez l'adresse saisie ou connectez-vous à nouveau.

Vous allez être automatiquement redirigé vers

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.....(I....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...I.;.T.X}....N.r .<d...#.........S..
..#Oa.       ...V..EPj..G...A..D.K...Z1..c.h,b.4..b.3...I.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>
```

Optional Reading

- What is HTTP error code 418?

https://www.neper.fr/2023/01/16/les-easter-eggs-de-google-418-teapot/

Hyper Text Coffee Pot Control Protocol

14 languages

Article Talk Read Edit View history

From Wikipedia, the free encyclopedia

The **Hyper Text Coffee Pot Control Protocol** (**HTCPCP**) is a facetious communication protocol for controlling, monitoring, and diagnosing coffee pots. It is specified in RFC 2324, published on 1 April 1998 as an April Fools' Day RFC,[2] as part of an April Fools prank.[3] An extension, HTCPCP-TEA, was published as RFC 7168 on 1 April 2014[4] to support brewing teas, which is also an April Fools' Day RFC.

Hyper Text Coffee Pot Control Protocol

Back-end infrastructure of error418.net, which implements HTCPCP

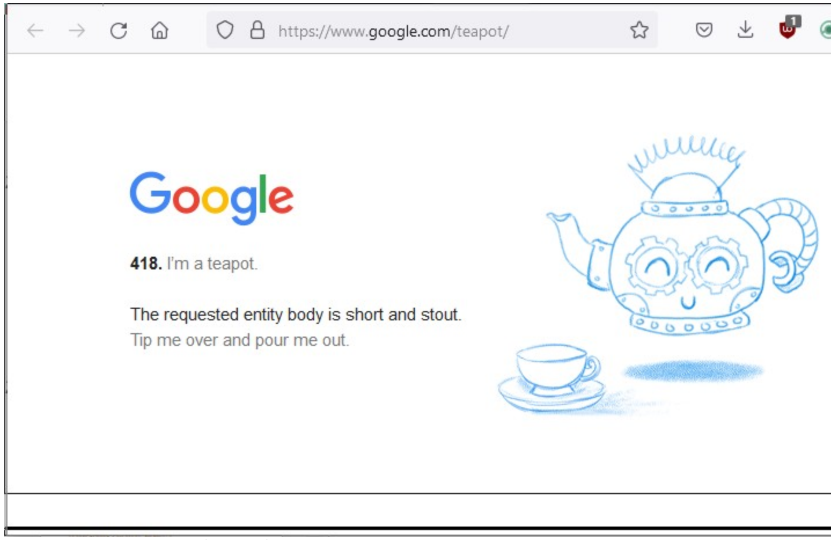
International standard	Internet Engineering Task Force
Developed by	Larry Masinter
Introduced	April 1, 1998
Website	rfc2324

Protocol [edit]

RFC 2324 was written by Larry Masinter, who describes it as a satire, saying "This has a serious purpose – it identifies many of the ways in which HTTP has been extended inappropriately."[5] The wording of the protocol made it clear that it was not entirely serious; for example, it notes that "there is a strong, dark, rich requirement for a protocol designed espressoly [*sic*] for the brewing of coffee".

Despite the joking nature of its origins, or perhaps because of it, the protocol has remained as a minor presence online. The editor Emacs includes a fully functional client side implementation of it,[6] and a number of bug reports exist complaining about Mozilla's lack of support for the protocol.[7] Ten years after the publication of HTCPCP, the *Web-Controlled Coffee Consortium* (*WC3*) published a first draft of "HTCPCP Vocabulary in RDF"[8] in parody of the World Wide Web Consortium's (W3C) "HTTP Vocabulary in RDF".[9]

On April 1, 2014, RFC 7168 extended HTCPCP to fully handle teapots.[4]



https://www.google.com/teapot/

418. I'm a teapot.

The requested entity body is short and stout.
Tip me over and pour me out.

Commands and replies [edit]

HTCPCP is an extension of HTTP. HTCPCP requests are identified with the Uniform Resource Identifier (URI) scheme `coffee` (or the corresponding word in any other of the 29 listed languages) and contain several additions to the HTTP methods:

`BREW` or `POST`	Causes the HTCPCP server to brew coffee. Using POST for this purpose is deprecated. A new HTTP request header field "Accept-Additions" is proposed, supporting optional additions including Cream, Whole-milk, Vanilla, Raspberry, Whisky, Aquavit, etc.
`GET`	"Retrieves" coffee from the HTCPCP server.
`PROPFIND`	Returns metadata about the coffee.
`WHEN`	Says "when", causing the HTCPCP server to stop pouring milk into the coffee (if applicable).

It also defines two error responses:

`406 Not Acceptable`	The HTCPCP server is unable to provide the requested addition for some reason; the response should indicate a list of available additions. The RFC observes that "In practice, most automated coffee pots cannot currently provide additions."
`418 I'm a teapot`	The HTCPCP server is a teapot; the resulting entity body "may be short and stout" (a reference to the song "I'm a Little Teapot"). Demonstrations of this behaviour exist.[1][10]

Optional Lab Exercise

- Use putty or telnet to connect to port 80 of a web server (http://www.columbia.edu for instance) and issue HTTP/1.x requests (get /~fdc/sample.html). Observe the responses.

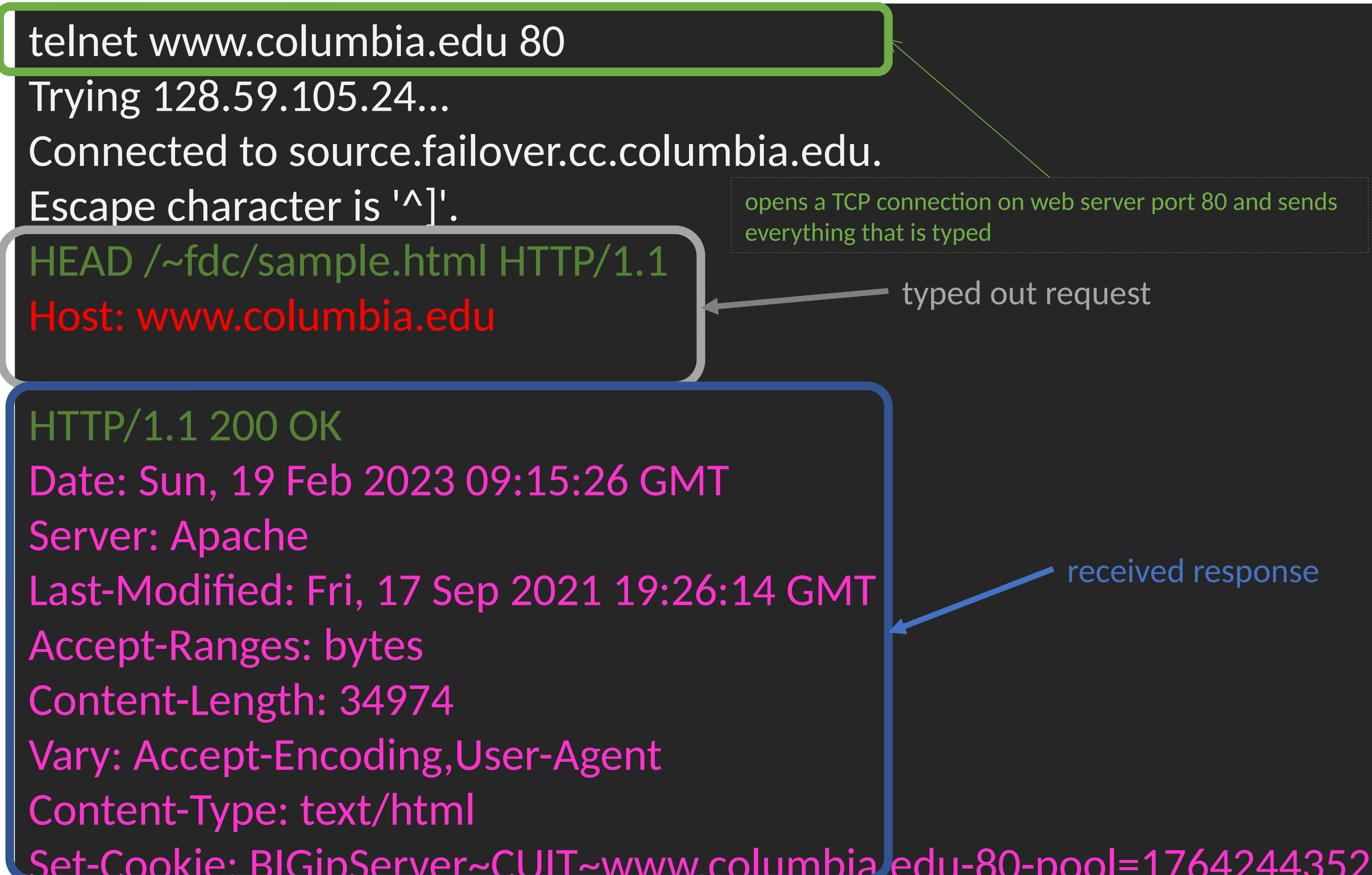
  ```
  telnet serverName 80
  ```

- For HTTPS, use:

  ```
  openssl s_client -connect
  serverName:443
  ```

(note: this exercise is limited to HTTP/1.x because HTTP/2 is no longer textual but uses binary format commands)

Lab: Results



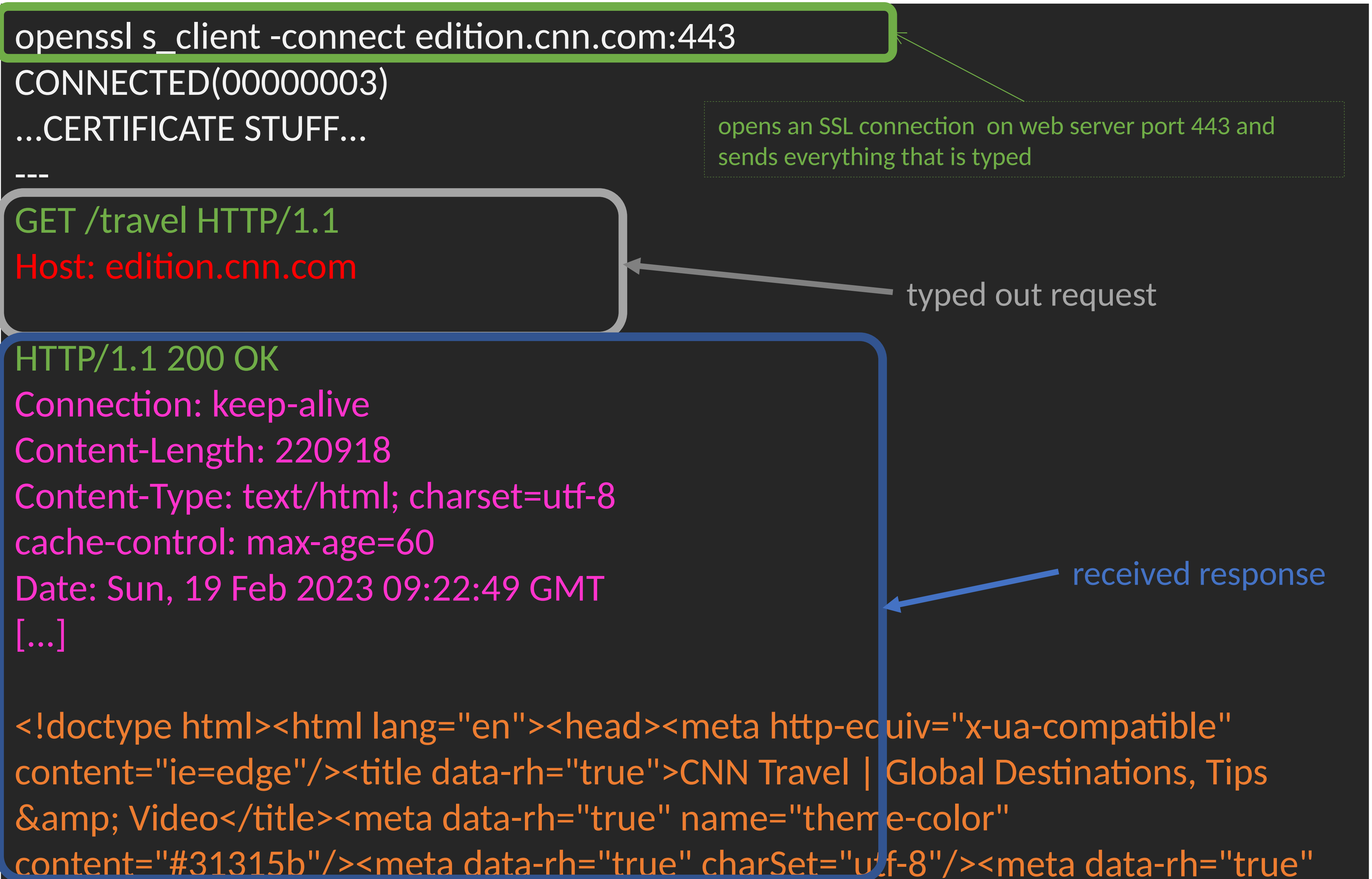
```
telnet www.columbia.edu 80
Trying 128.59.105.24...
Connected to source.failover.cc.columbia.edu.
Escape character is '^]'.
HEAD /~fdc/sample.html HTTP/1.1
Host: www.columbia.edu

HTTP/1.1 200 OK
Date: Sun, 19 Feb 2023 09:15:26 GMT
Server: Apache
Last-Modified: Fri, 17 Sep 2021 19:26:14 GMT
Accept-Ranges: bytes
Content-Length: 34974
Vary: Accept-Encoding,User-Agent
Content-Type: text/html
Set-Cookie: BIGipServer~CUIT~www.columbia.edu-80-pool=1764244352
```

opens a TCP connection on web server port 80 and sends everything that is typed

typed out request

received response

Lab: Results



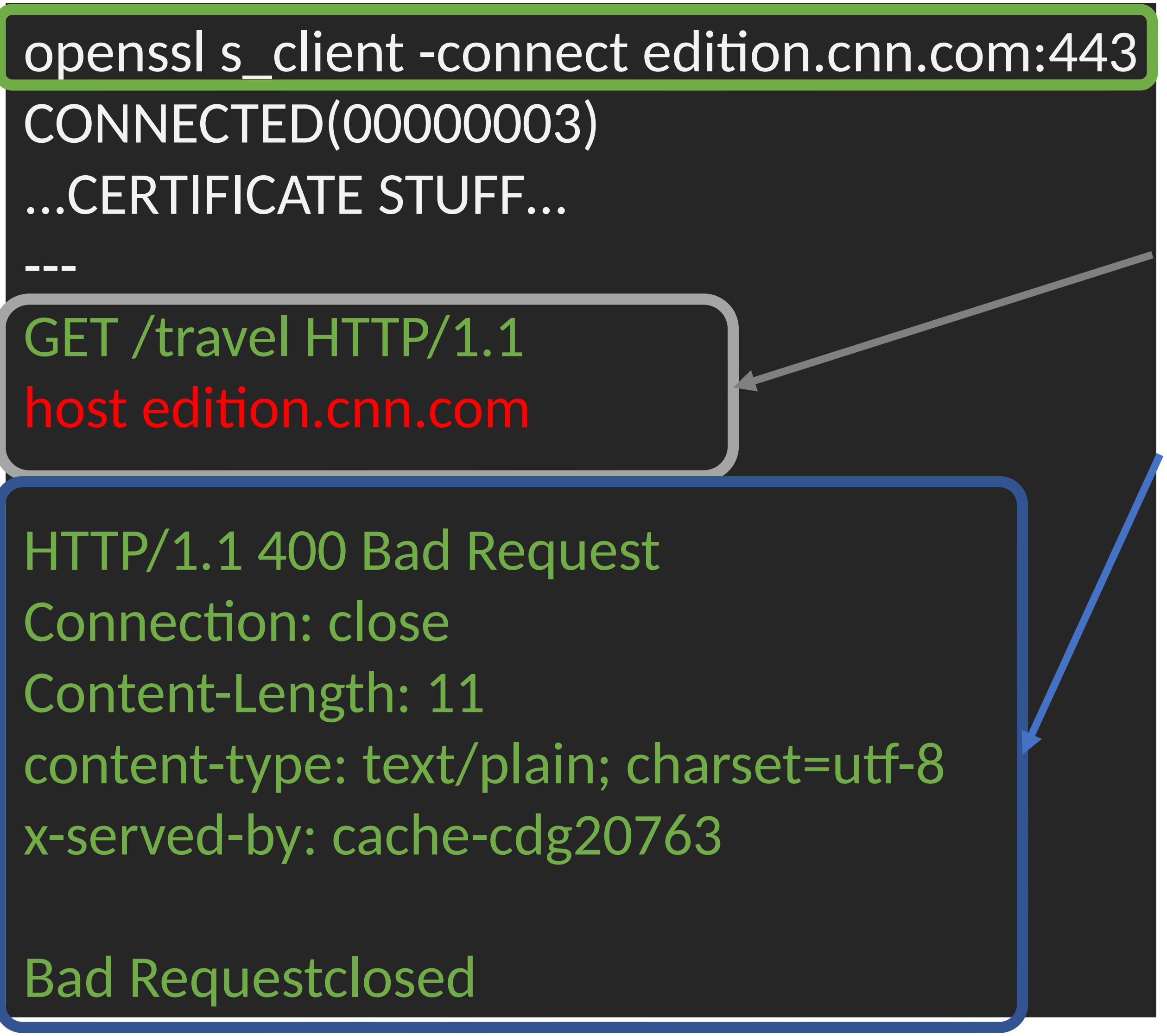
```
openssl s_client -connect edition.cnn.com:443
CONNECTED(00000003)
...CERTIFICATE STUFF...
---
GET /travel HTTP/1.1
Host: edition.cnn.com

HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 220918
Content-Type: text/html; charset=utf-8
cache-control: max-age=60
Date: Sun, 19 Feb 2023 09:22:49 GMT
[...]

<!doctype html><html lang="en"><head><meta http-equiv="x-ua-compatible"
content="ie=edge"/><title data-rh="true">CNN Travel | Global Destinations, Tips
&amp; Video</title><meta data-rh="true" name="theme-color"
content="#31315b"/><meta data-rh="true" charSet="utf-8"/><meta data-rh="true"
```

opens an SSL connection on web server port 443 and sends everything that is typed

typed out request

received response

Lab: Results



```
openssl s_client -connect edition.cnn.com:443
CONNECTED(00000003)
...CERTIFICATE STUFF...
---
GET /travel HTTP/1.1
host edition.cnn.com

HTTP/1.1 400 Bad Request
Connection: close
Content-Length: 11
content-type: text/plain; charset=utf-8
x-served-by: cache-cdg20763

Bad Requestclosed
```

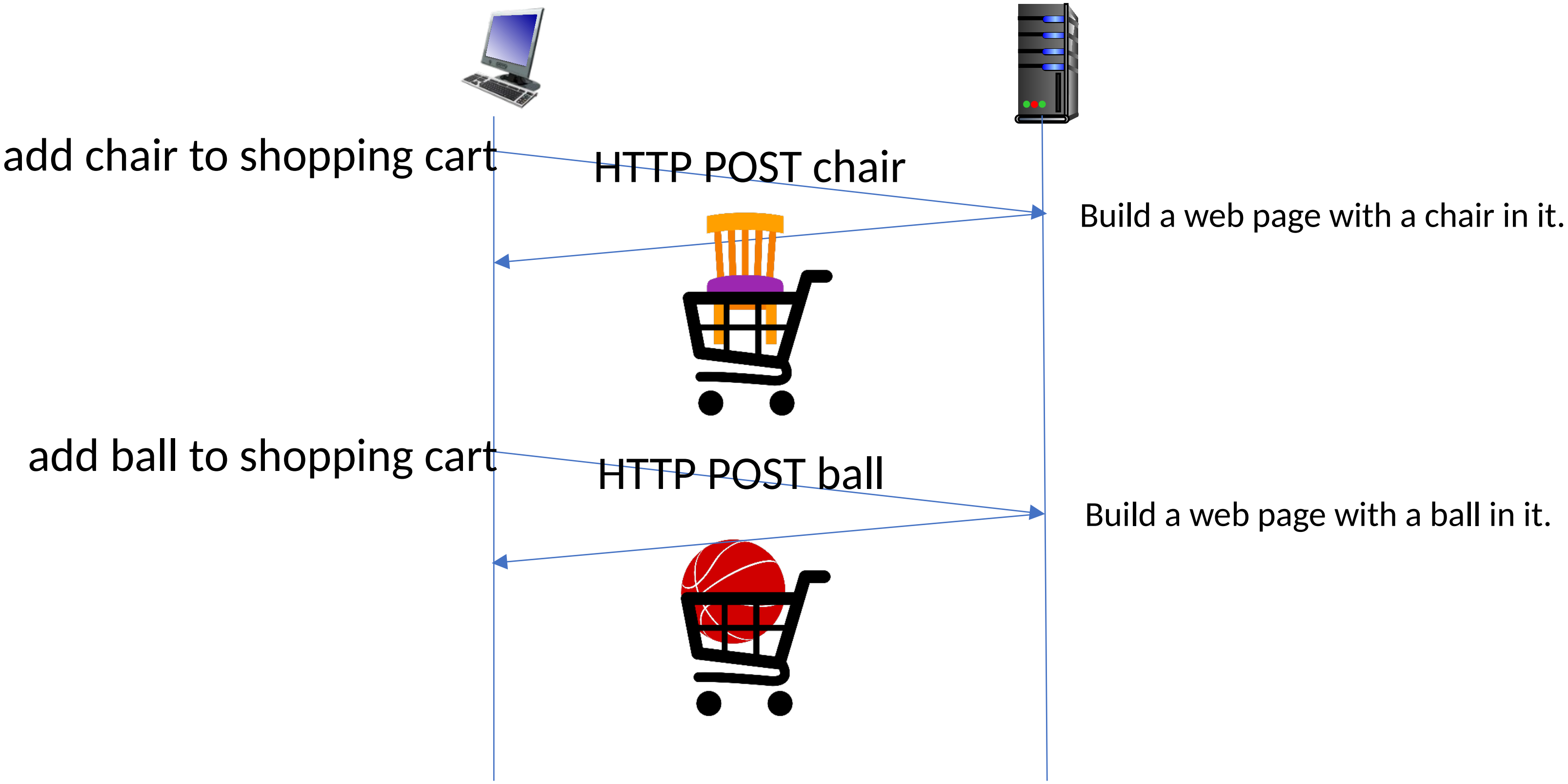
typed out request
(correct syntax is
Host: edition.cnn.com)

received response

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.



add chair to shopping cart

HTTP POST chair

Build a web page with a chair in it.

add ball to shopping cart

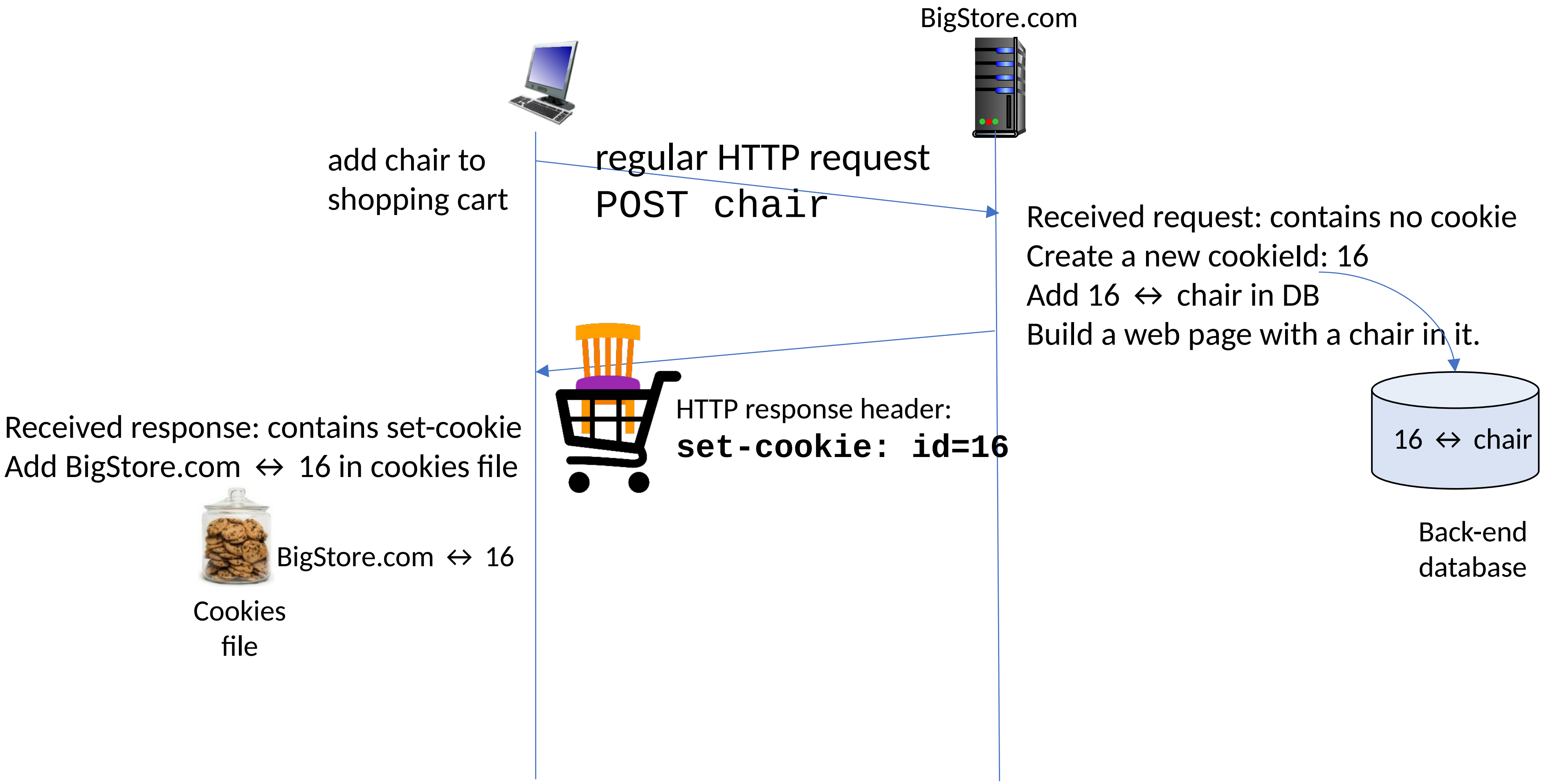
HTTP POST ball

Build a web page with a ball in it.

Cookies

Shhhhh… they're in here somewhere. Dad's always talking about the cookies in his computer.

Cookie Example



BigStore.com

add chair to shopping cart

regular HTTP request

`POST chair`

Received request: contains no cookie
Create a new cookieId: 16
Add 16 ↔ chair in DB
Build a web page with a chair in it.

HTTP response header:

`set-cookie: id=16`

Received response: contains set-cookie
Add BigStore.com ↔ 16 in cookies file

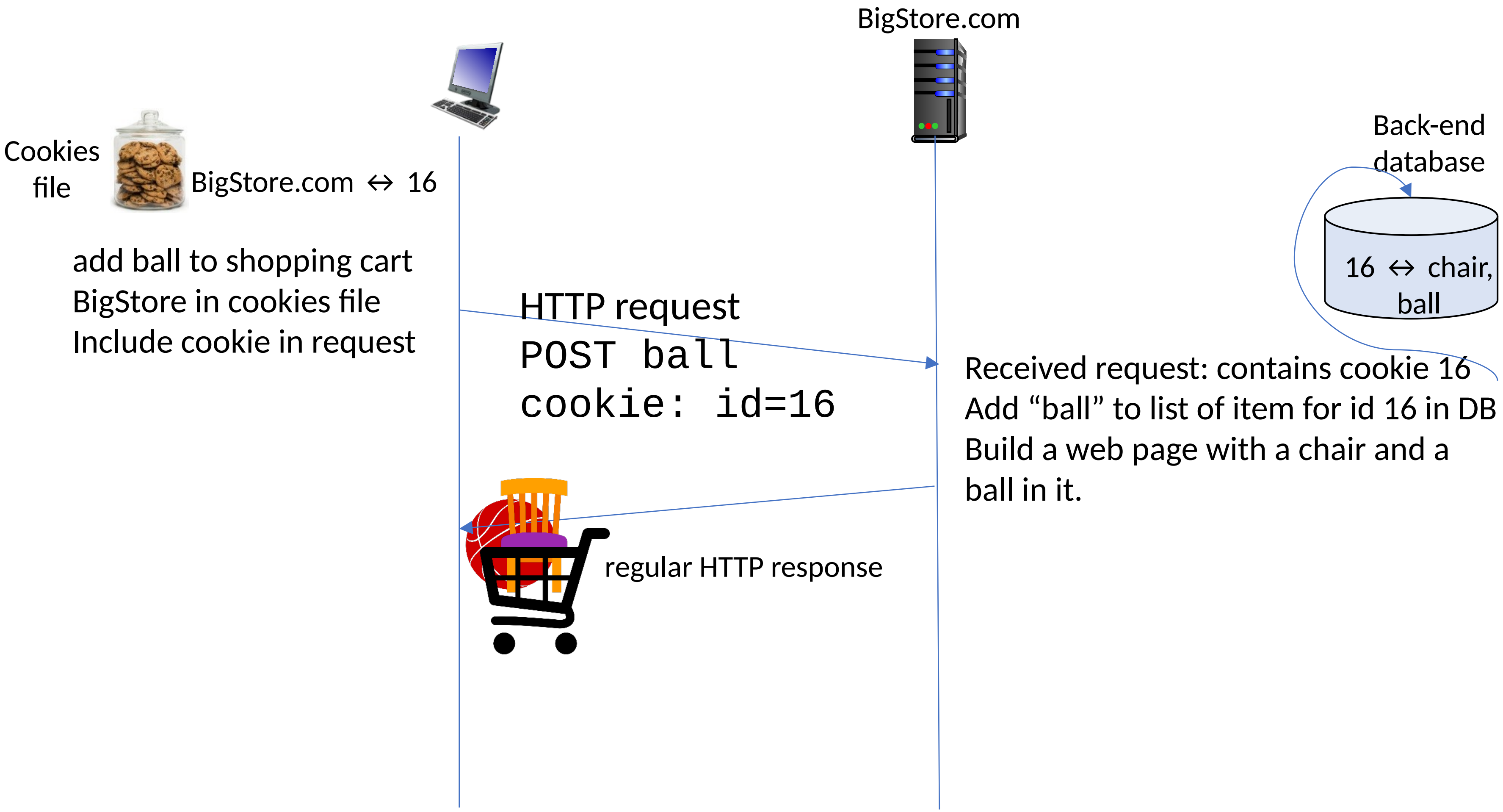
BigStore.com ↔ 16

Cookies file

16 ↔ chair

Back-end database

Cookie Example



BigStore.com

Cookies file

BigStore.com ↔ 16

add ball to shopping cart
BigStore in cookies file
Include cookie in request

HTTP request
`POST ball`
`cookie: id=16`

Back-end database

16 ↔ chair, ball

Received request: contains cookie 16
Add “ball” to list of item for id 16 in DB
Build a web page with a chair and a ball in it.

regular HTTP response

Uses

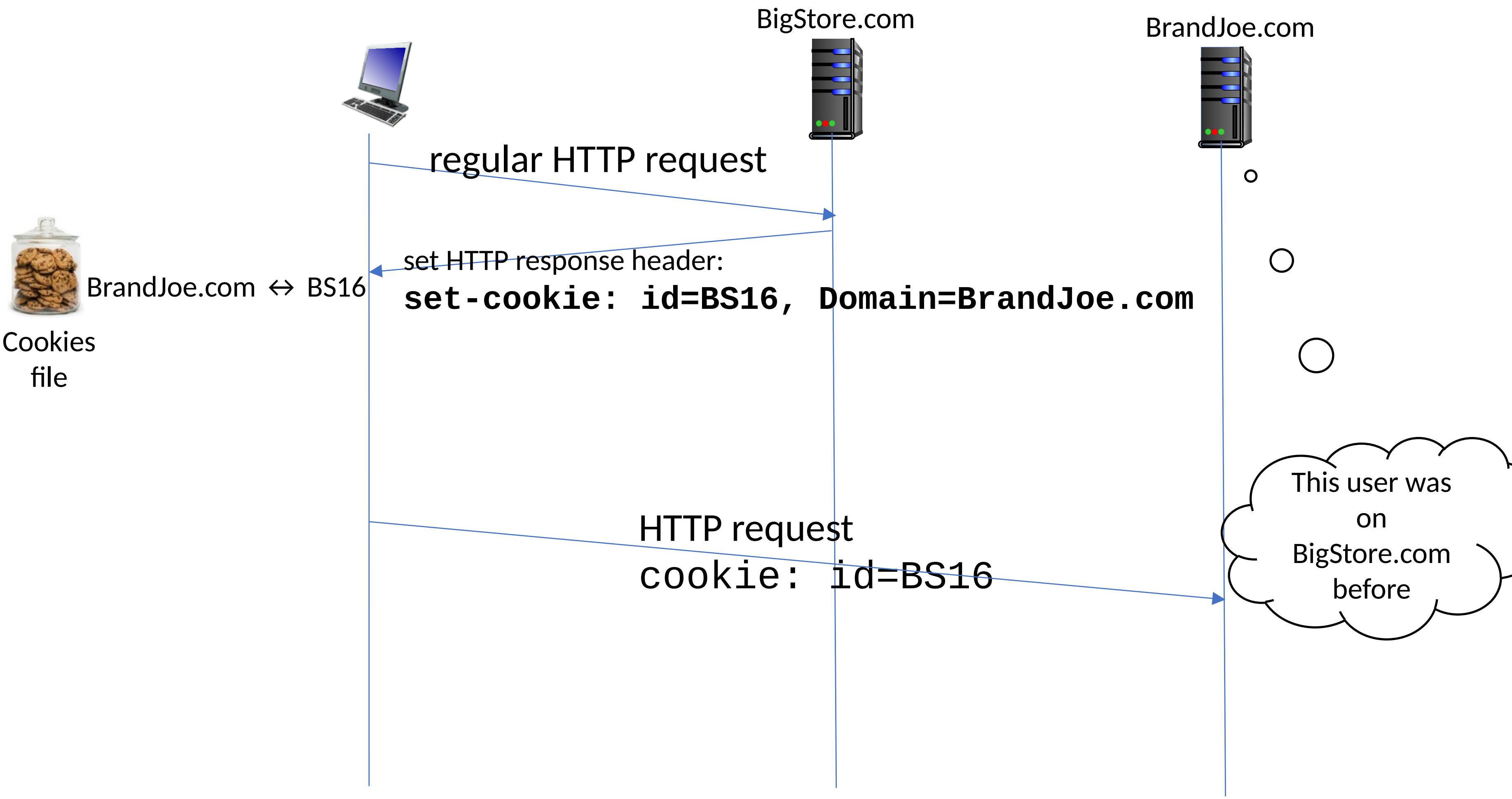
create a user session layer on top of stateless HTTP

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

Suspicions

- Invasion of privacy

Third-party advertising cookies



BigStore.com

BrandJoe.com

regular HTTP request

set HTTP response header:

`set-cookie: id=BS16, Domain=BrandJoe.com`

BrandJoe.com ↔ BS16

Cookies file

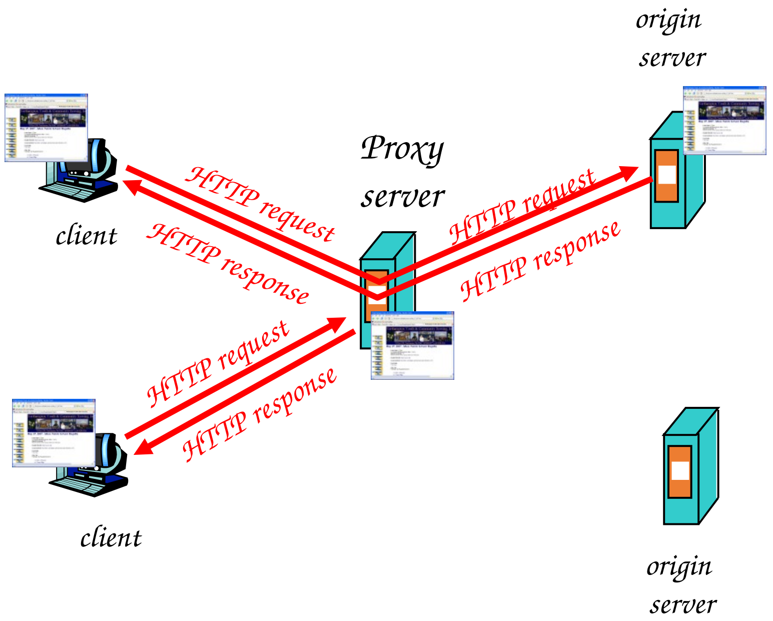
HTTP request

`cookie: id=BS16`

This user was on BigStore.com before

Web Cache (proxy server)

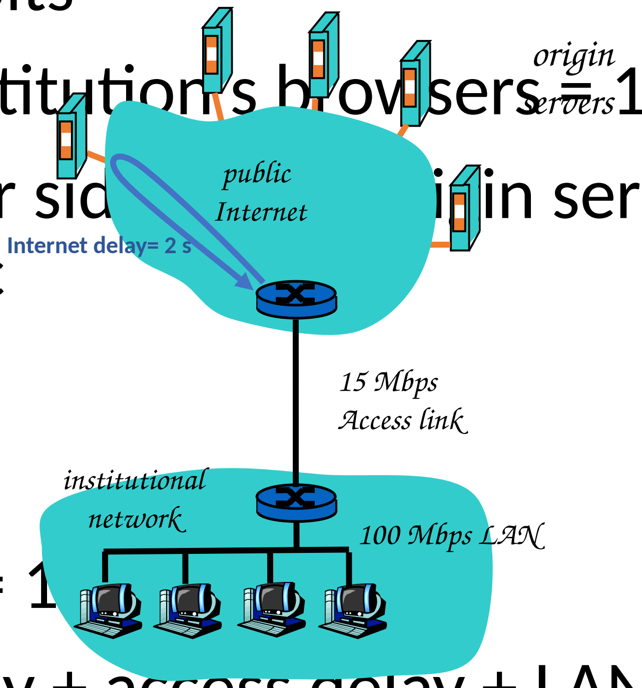
- to satisfy the requests without involving the real server
- browser must be configured to send all HTTP requests to cache
- reduced traffic on Internet, improved response time



Caching Example

assumptions

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



consequences

- utilization on LAN = 15%
- utilization on access link = 1
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds

Caching Example (cont)

possible solution

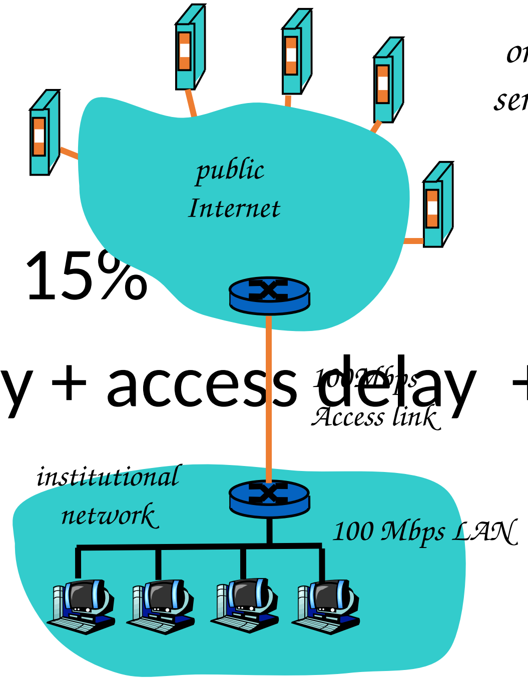
- increase bandwidth of access link to 100 Mbps

consequence

- utilization on LAN = 15%
- utilization on access link = 15%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + msecs + msecs

often a costly upgrade



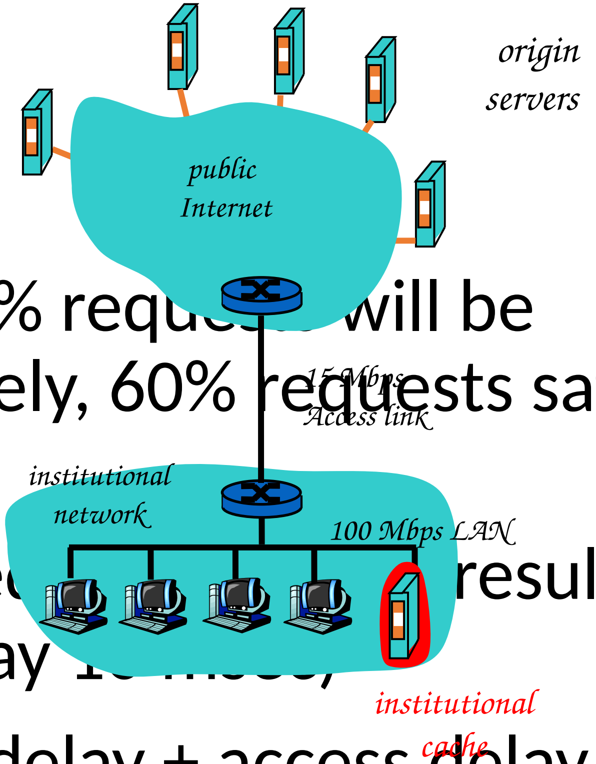
Caching Example (cont)

possible solution:

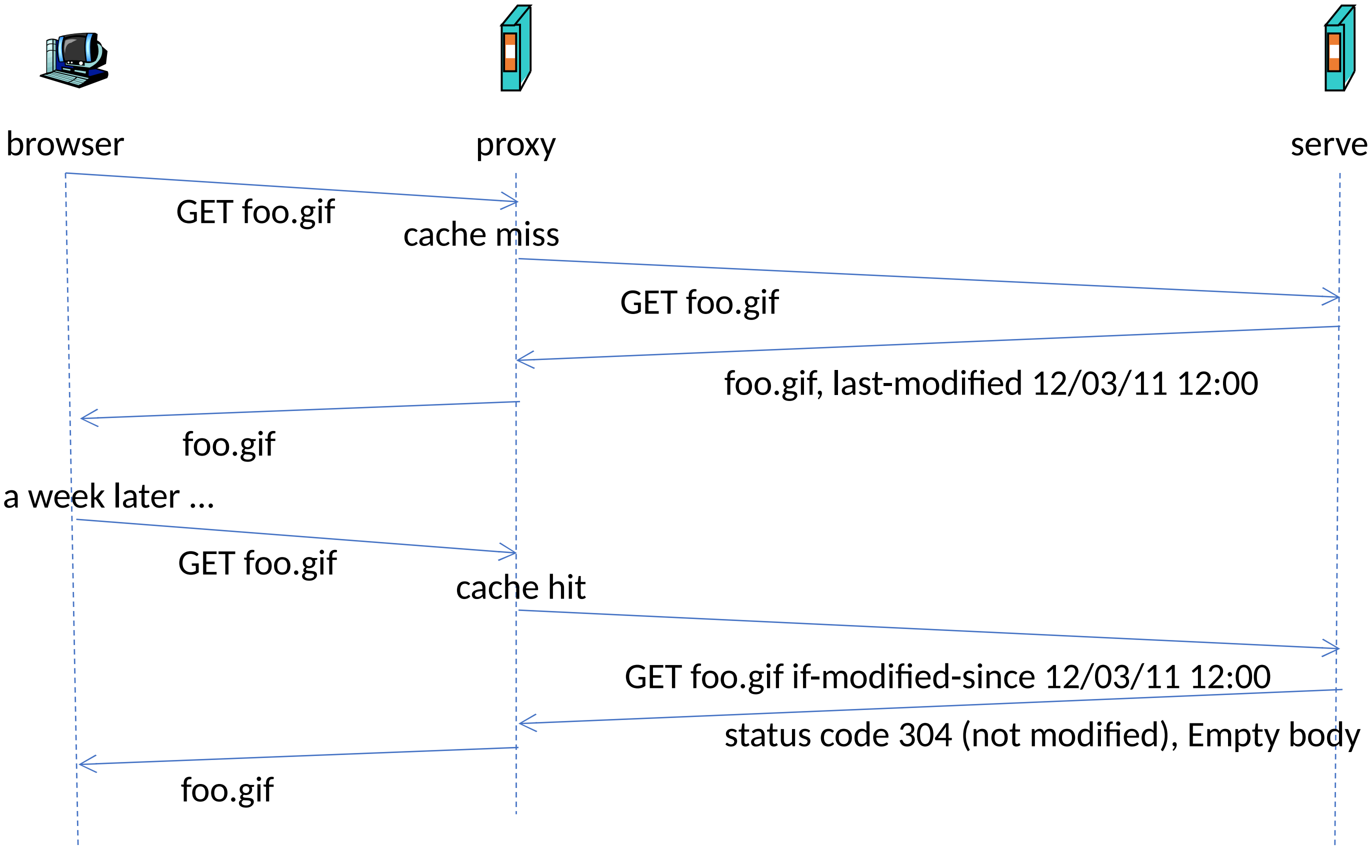
- install cache

consequence

- suppose hit rate is 0.4 (40% requests will be satisfied almost immediately, 60% requests satisfied by origin server)
- utilization of access link re[illegible] resulting and in negligible delays (say 10 msecs)
- avg total delay = Internet delay + access delay + LAN delay =0.6 * 2.01 secs * + 0.4 * 10 millisecs <1.3 secs



Conditional GET


browser
proxy
server
GET foo.gif
cache miss
GET foo.gif
foo.gif, last-modified 12/03/11 12:00
foo.gif
a week later …
GET foo.gif
cache hit
GET foo.gif if-modified-since 12/03/11 12:00
status code 304 (not modified), Empty body
foo.gif

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.

{REST}

Origins

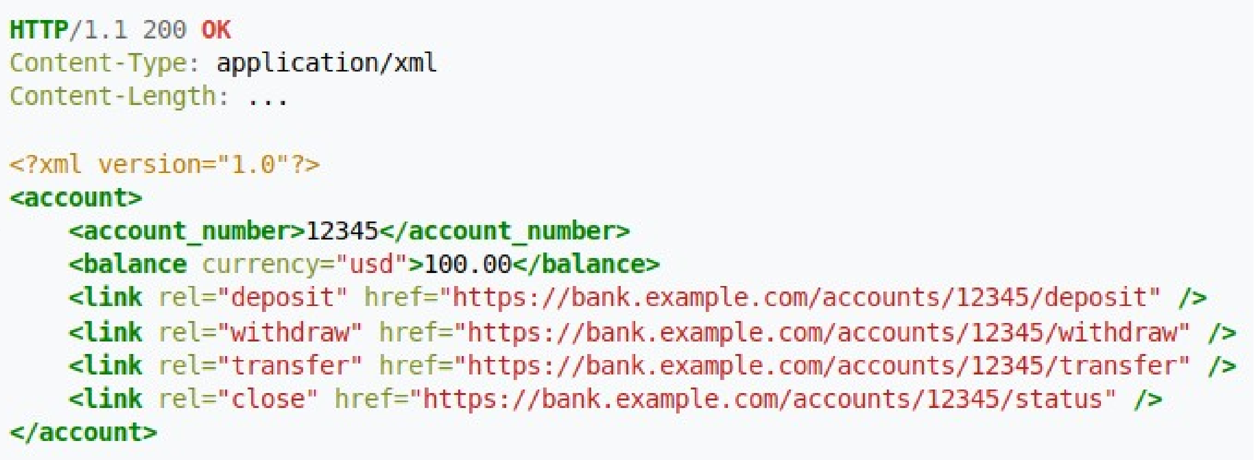
- *Representational State Transfer* – REST: defined in 2000 Roy Fielding's PhD dissertation (after he worked on HTTP 1.1 and URI RFCs)
- Web application =
  - network of Web resources (a virtual state-machine)
  - where the user progresses through the application by selecting resource identifiers and 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)*

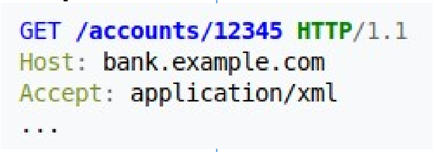
HATEOAS Example

response if balance > 0

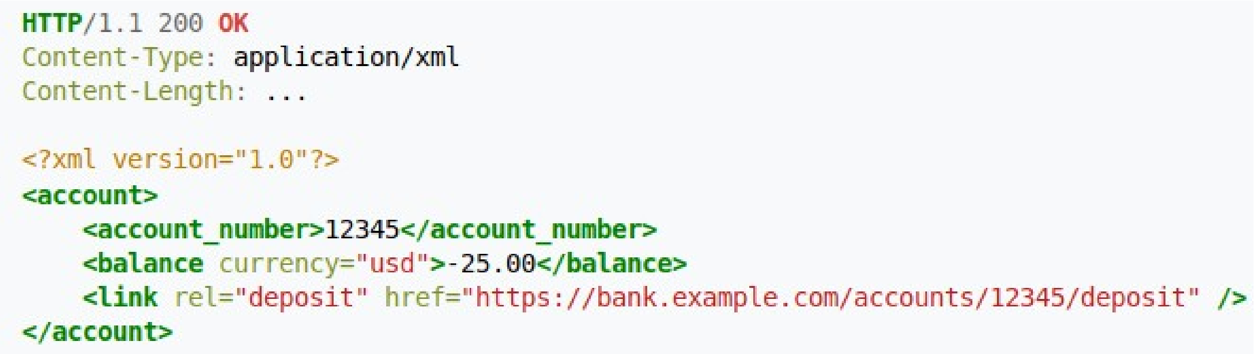


```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...

<?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>
```

request



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



```
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>
```

response if balance < 0

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	*location* 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
PATCH
	update
	partial update
		PUT /serv/users/34
		# body
		{
		name: “Jacques"
		}
			200, 204 No Content	no	no	Yes

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

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