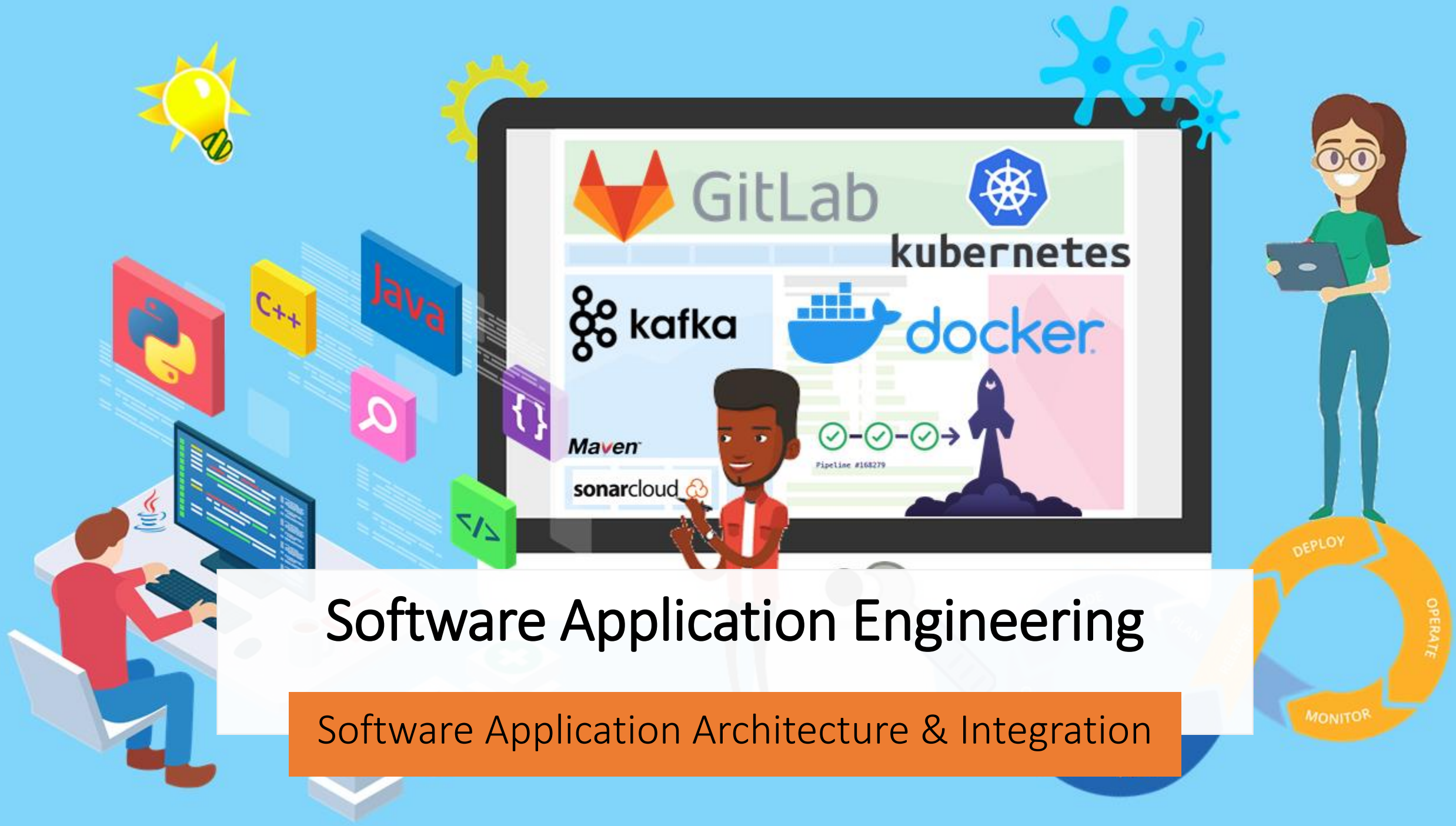



# Software Application Engineering

## Software Application Architecture & Integration

# Introduction

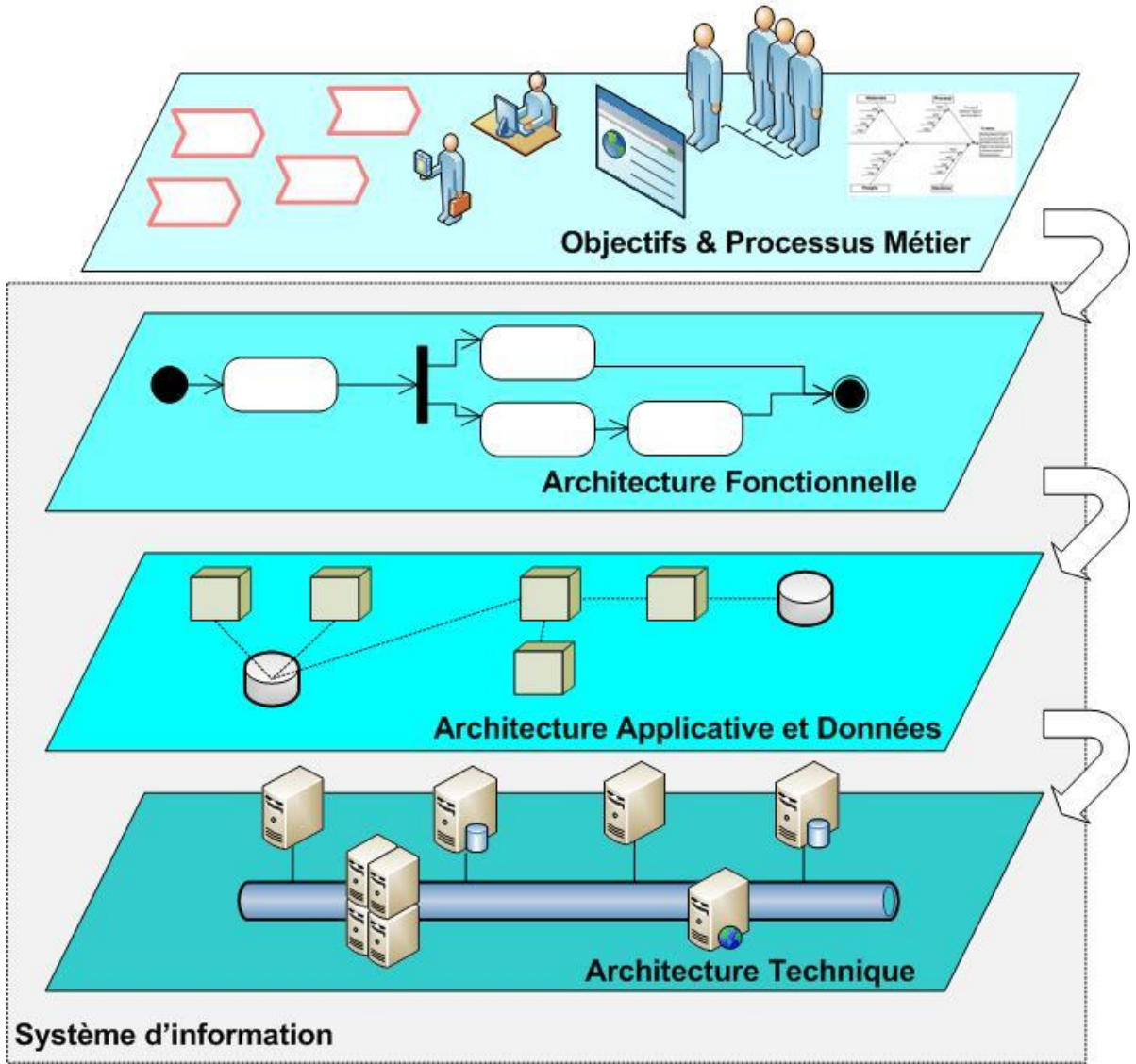
# Application architecture

- *arrangement of functions/processes, of information/data and of their interaction patterns*
- representation of the Information System as

  **communicating** (exchanging messages, a/synchronous flows)

  **application blocks** (software components / executable modules offering a service via an interface), highlighting choices made to ensure
    - scalability of the IS in the face of changes in the business
    - reuse and overall consistency of the components
- and definition of the **implementation on the technical infrastructure**, taking into account:
  - the technical architectures and the existence of software packages
  - non-functional constraints (QoS, load...)
  - organizational constraints and the level of competence of internal teams

# The 4 levels of IS urbanization




http://www.buildit2run.com/management-du-si/4-bonnes-pratiques-pour-deployer-un-referentiel-d%E2%80%99architecture/

# Objective of this chapter

- Providing a panorama of the different families of application architectures (they are all still met!)
- Discovering the principles of the basic mechanisms for application integration

# Layered application structure (“tiers”)

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

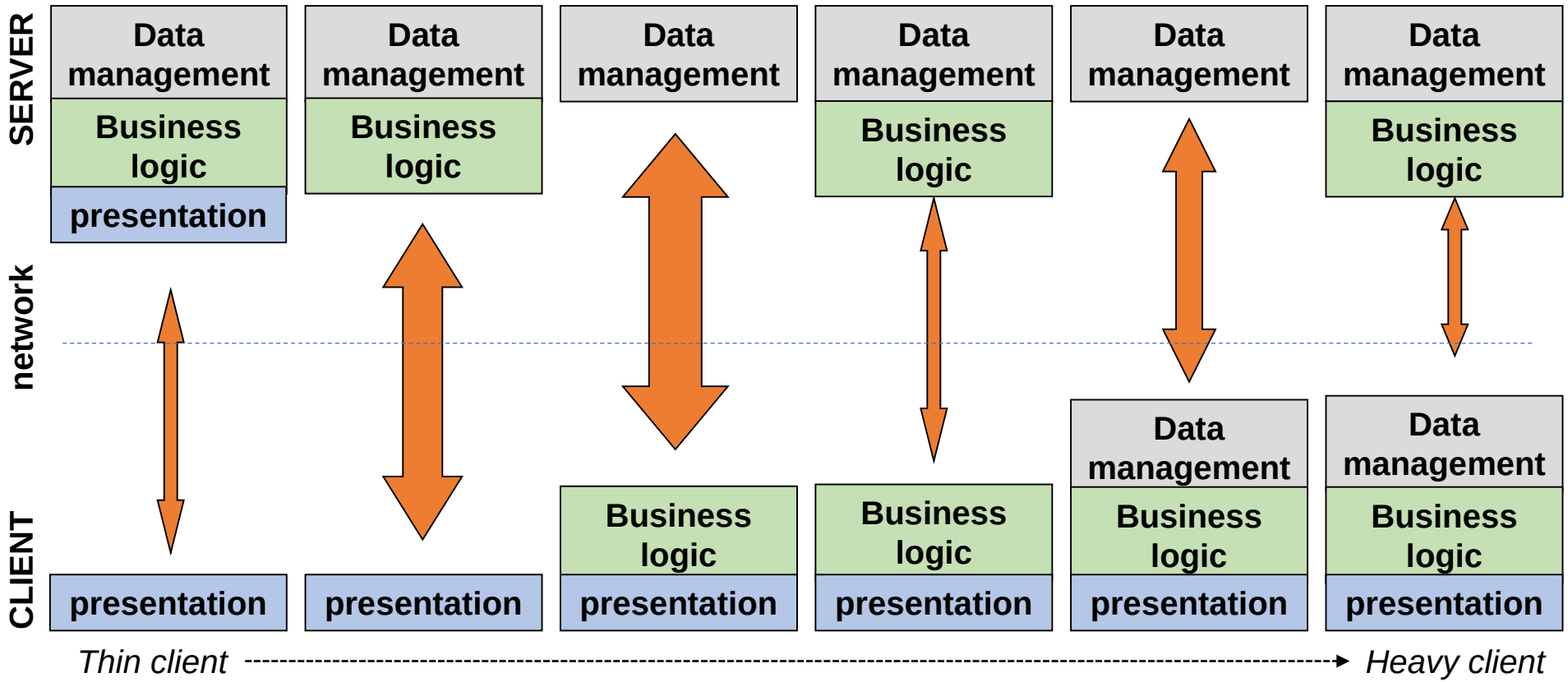
- Presentation:
  user inputs and commands, and display

- Business logic:
  business objects, rules, processing logic, processes

- Data:
  - Data access logic:
    data model and query language
  - Data storage :
    read/write, persistence mechanism, distribution system, replication system

# Application architectures

Several strategies for distributing functions between terminals and servers:



# Agenda

monolithic

modular & distributed

I. Typology of application architectures
   A. Autonomous applications
      1. Mainframes
      2. Variation et evolutions: Web Applications
      3. Mobile terminals
   B. Multi-tier architecture
      1. 2-tier architecture
      2. 3-5-tier architecture
      3. Application Servers
   C. Service-oriented architecture

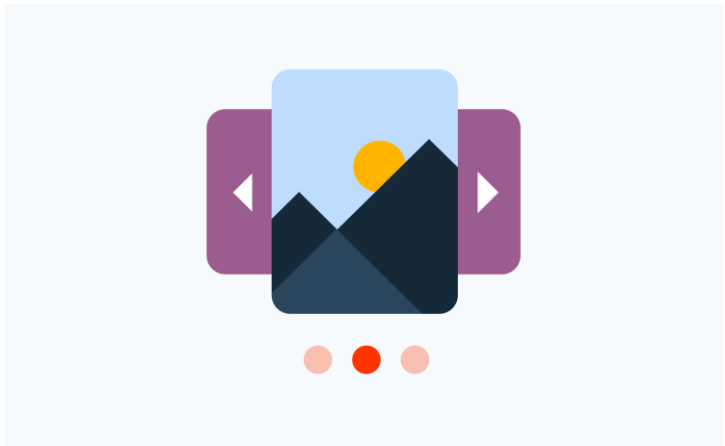
II. Deployment on the technical architecture
   1. Peer-to-peer architecture
   2. n-tier architecture
   3. Deployment in the cloud
   4. Virtualization and containers

III. Application Integration
   1. ICP
   2. RPC and ORB
   3. MOM
   4. orchestration

# A lot to cover today!

# Credits

- 10% of the slides are based upon the ones from Alexandre Feray and Cyril Rognon
- Images: references in the margins or unknown origin



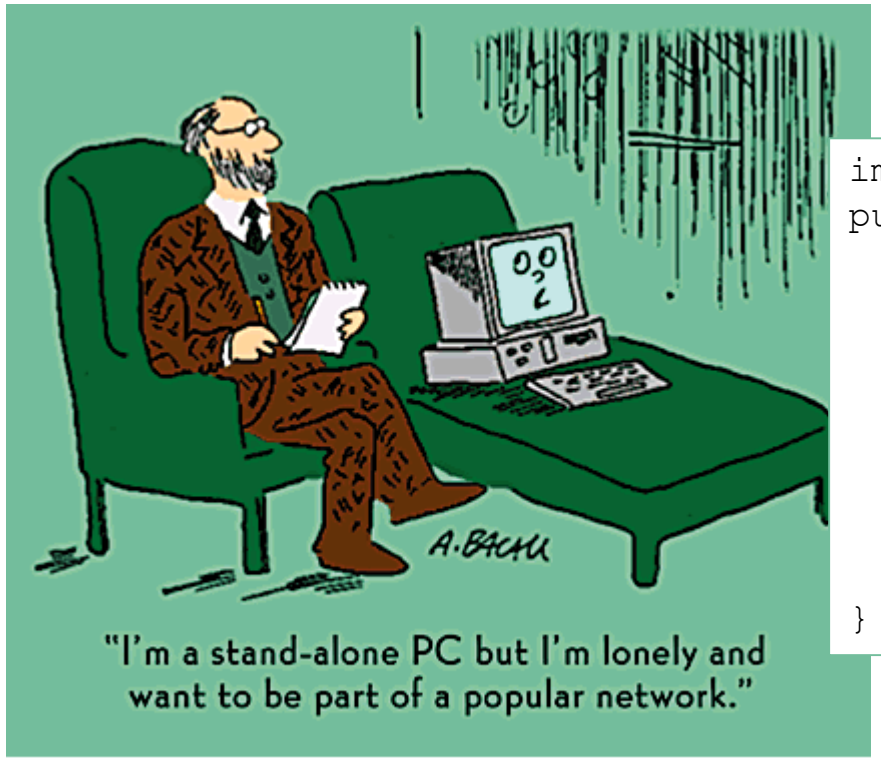
https://soliloquywp.com/addons/carousel/


Single tier

# I. A. Autonomous applications

# Autonomous application

The 3 application layers are tightly coupled and run on the same computer.






```
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

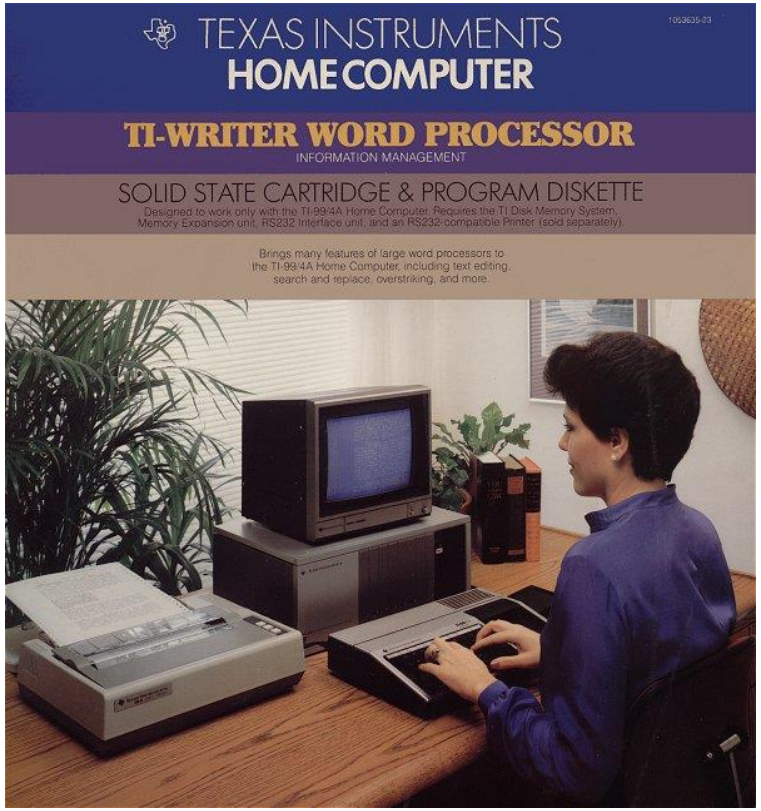
processing

presentation

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

# 2 “eras”

- “pre-network” PC (and 70 ’s – mi 80’s)




- Powerful mobile terminals

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

# Advantages

- Performance
  - 0 latency
- Ergonomics
  - application developed specifically for a machine, exploiting its I/O interface specificities
- Offline operation
- Isolation
  - no attack without physical access to the machine
- Simplicity
  - Well… it depends…

# Disadvantages

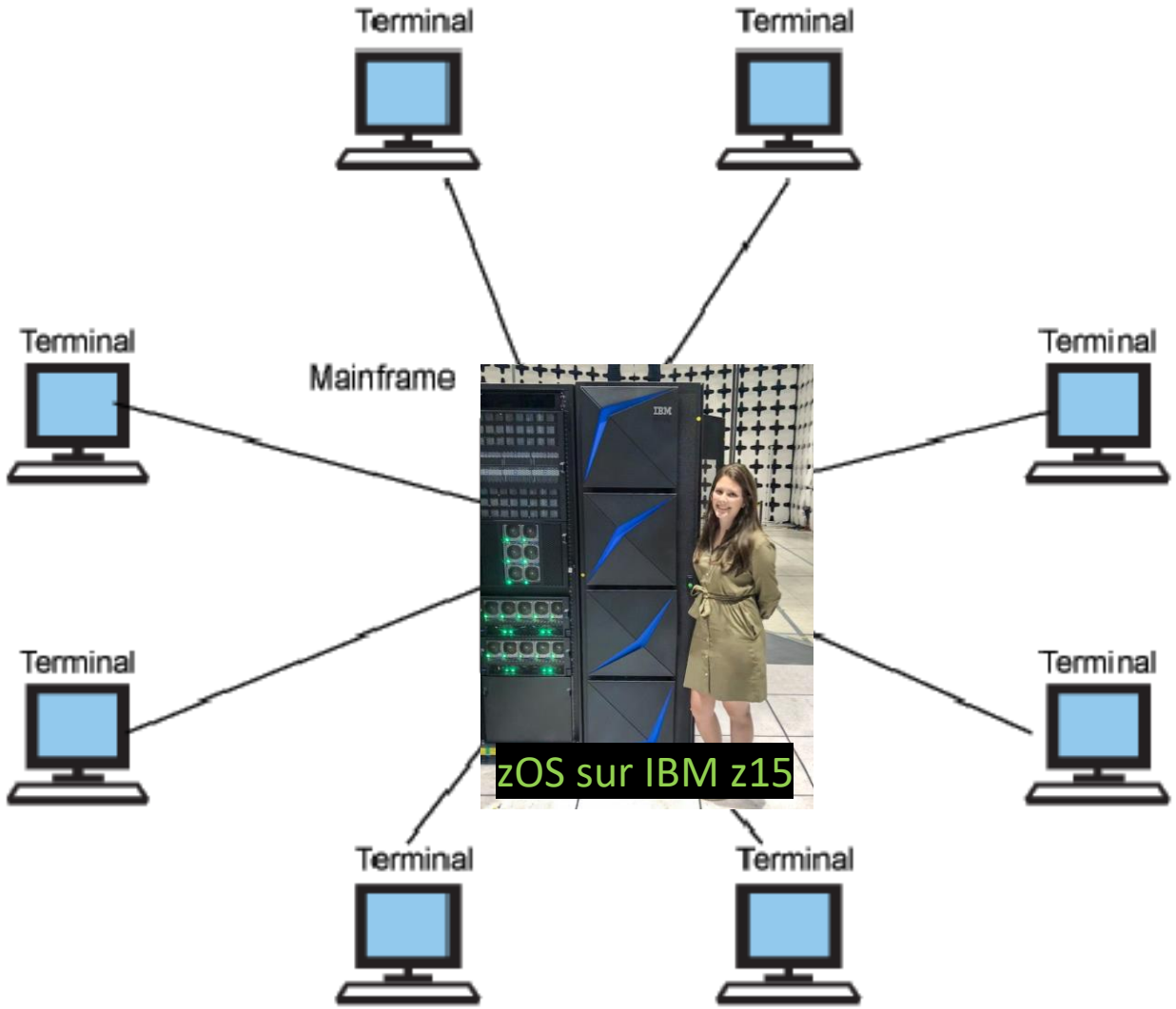
- Difficult deployment:
  - broadcast and maintain on each terminal
  - reinstall if the underlying system needs to be reinstalled
- Difficult nomadism:
  - access limited to physically connected users
  - more difficult (if not impossible) to resume a task from a different location/machine
- Isolation: same data stored several times →
  - no economy of scale (in storage resources)
  - duplications → risk of inconsistencies

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

# I. A. 1. Mainframe architectures

# Principle

- Logics for persistence, treatment, and presentation remain aggregated into an indissociable “monolithic” whole.
- Mainframe server: proprietary hardware and OS
- Supercomputer + passive clients (lightweight visualization application)



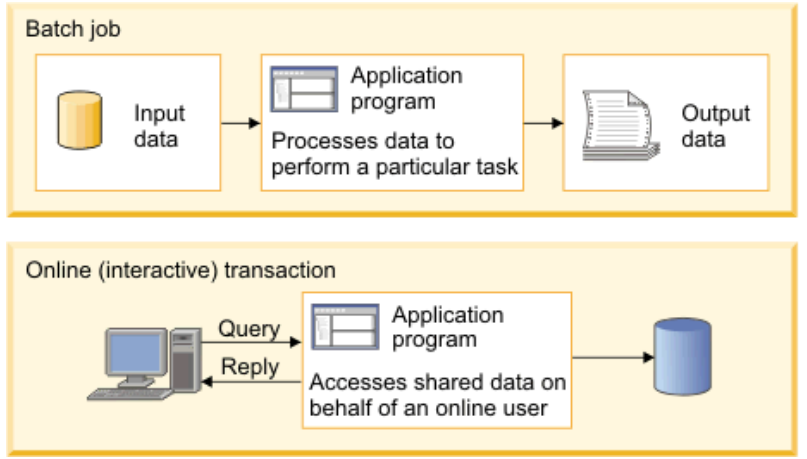

https://www.researchgate.net/figure/Mainframe-Architecture_fig2_275405572

# Fast and Many

- Industrial sectors :
  - banks, insurances, airlines...

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

  batch or real time operation:
  - batch back-office
    - offline processing of queued jobs
    - manipulation (and creation) of terabytes of data
  - transactional
    - customer interaction
    - in competition with a very large number of other transactions

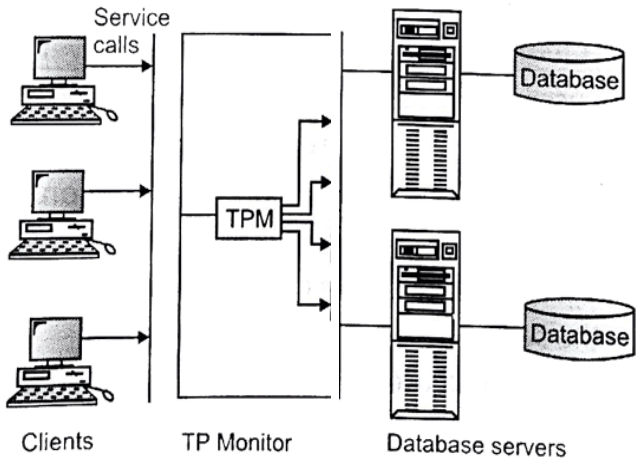



https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zmainframe/zconc_mfworklds.htm

# Transactions

- *Program accessing and/or modifying persistent data*
- A good transaction is
  - **A**tomic: all changes on the state are executed or none is (return to the initial state)
  - **C**onsistent: leaves the system in a valid state (respect of the constraints defined on the data)
  - **I**solated: if transactions are executed simultaneously, it appears with each transaction that the others have been executed either before or after; during its execution the changes made to the data are invisible to the others.
  - **D**urable: changes performed by the transaction survive failures
- Transactional monitor ("TP monitor")

  Orders transactions executed in parallel
  - Multiplexing of queries on system resources
  - Transaction management (ACID properties compliance)




http://www.imfrosty.com/2014/09/multi-user-dbms-architecture.html

# Advantages

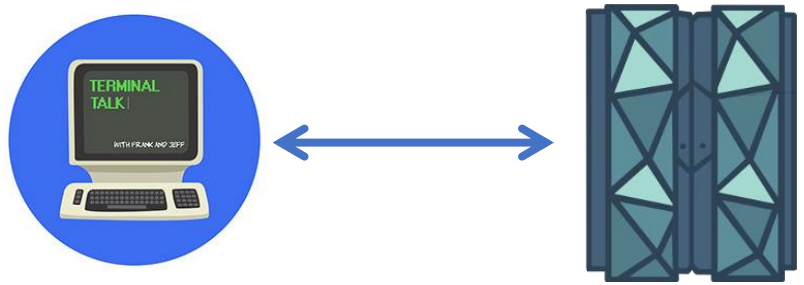
- Ability to handle a very large number of simultaneous queries on very large databases

  specialized technology, nb cores++, memory++
- Reliability (IBM Z customers: 99.9999% uptime)
  - redundancy+++
  - robustness
- Consistency, stability and long-term support
- Security
  - very restricted access (but less than it used to)
  - hardware encryption


https://www.youtube.com/watch?v=kmMn5Q_lnkk

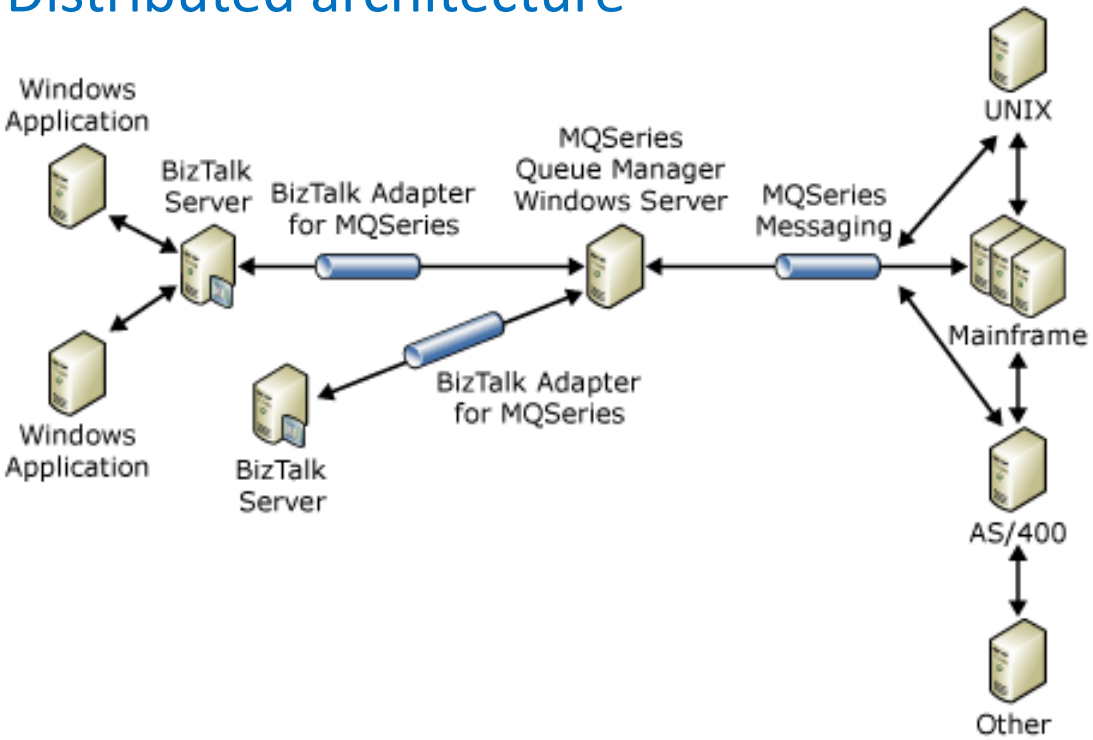
# Mainframe in IS architectures

## Host architecture



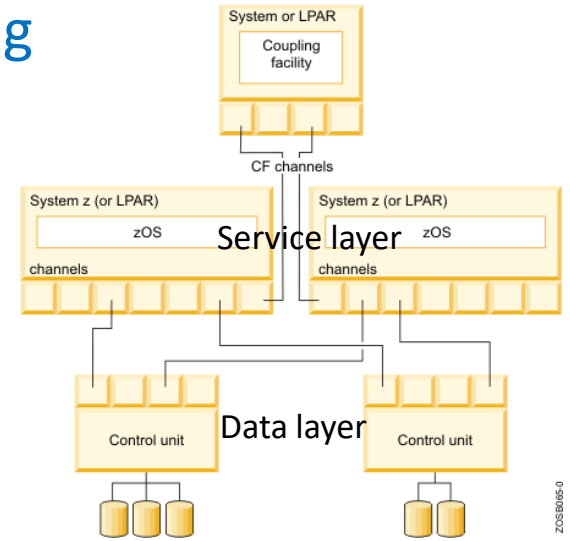

3270 terminal

## Distributed architecture




## Clustering

link of
up to 32 servers




https://docs.microsoft.com/fr-fr/biztalk/core/components-of-the-mqseries-adapter

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

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

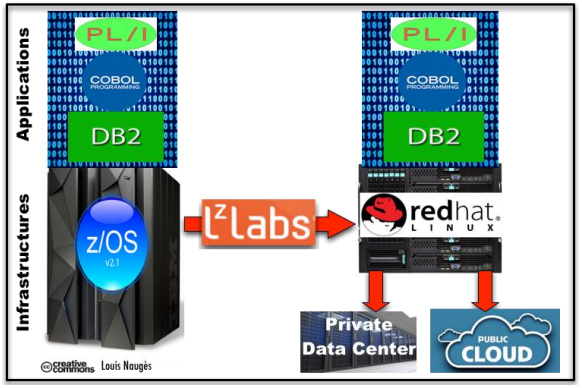
# Massively used

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

# Perspectives

- Closed world, proprietary solutions
- Costs:
  - Licenses: high (consensus!)
  - Maintenance: low, designed for robustness
  - Energy consumption? (contradictory figures)
- Mainframe skills that will become rare
  - "mainframe academy": little success
  - but Cobol simple to understand
- Credible alternatives + migration experience


https://sysadminotaur.devolutions.net/68-cobol

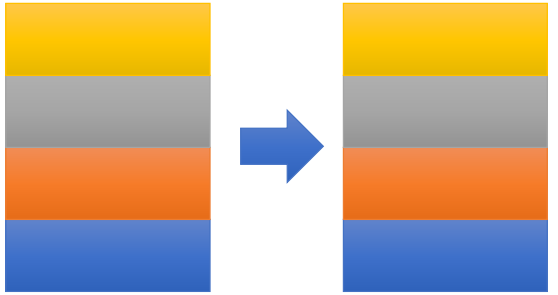



Applis COBOL

DB2 / CICS

zOS + Parallel SysPlex

Mainframe



ERP

Oracle / Websphere

Unix

Server Farms

https://nauges.typepad.com/my_weblog/management_si/page/3/

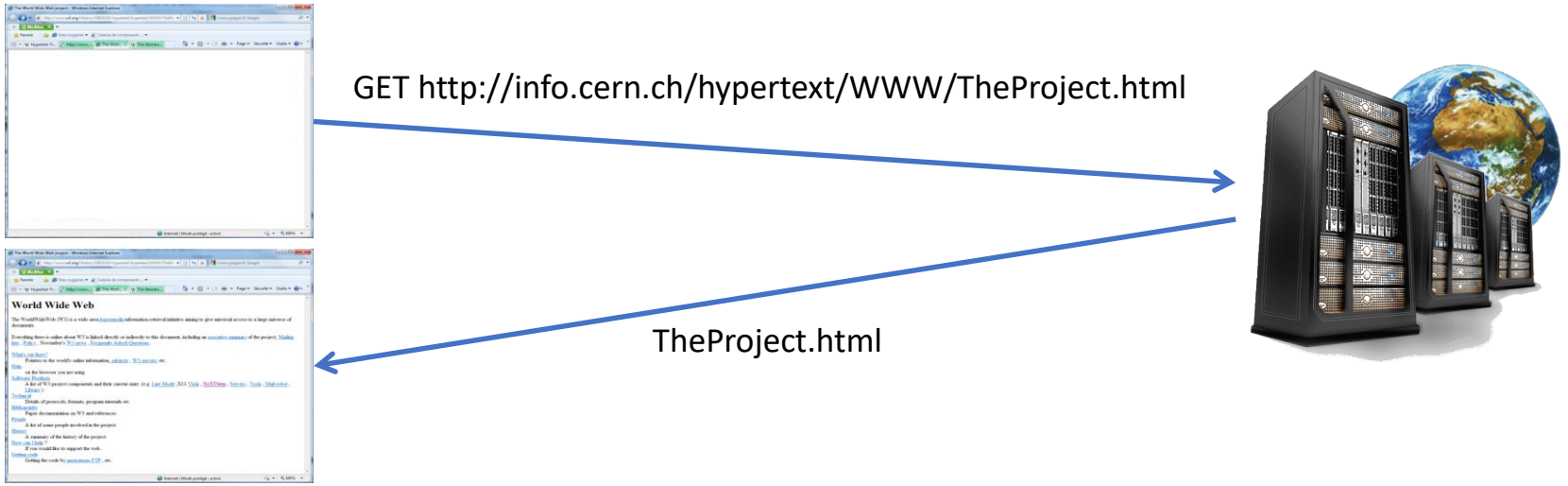

http://www.iquad.in/webapplications.html

Variations et evolutions

# 1. A. 2. Web Applications

# A particular case of the host model

- Client terminal: generic and simple
  - browser merely displays pages as-is (presentation)
- Server performs all the computations
  - provides pages/files (data storage and processing, some presentation)



# A basic model

- Technologies:
  - HTTP request/response communication protocol
  - content description with HTML
- Open and standardized context: RFC, W3C
- Easy to implement
  - Client is generic and without advanced functions (browser)
  - Anyone can develop a web server

# HTTP:
# a very simple yet robust and pervasive protocol

- 2 main commands: GET/POST (+6 others)
- stateless: no “session” concept
  + ease clustering
  - sessions must be managed another way
- able to benefit from cache/proxy

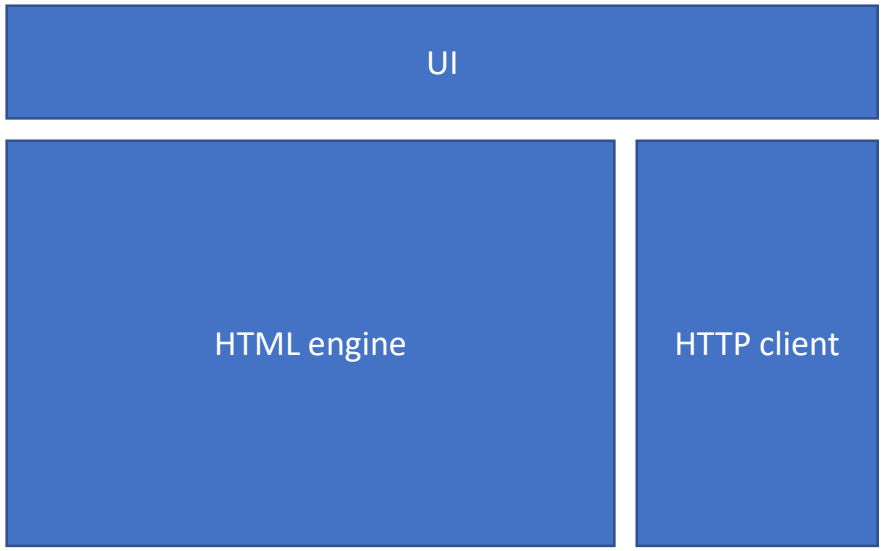
# HTML

```
<HEADER>
    <TITLE>The World Wide Web project</TITLE>
</HEADER>
<BODY>
    <H1>World Wide Web</H1>

    <P>The WorldWideWeb (W3) is a wide-area <A HREF="WhatIs.html">hypermedia</A> information
    retrieval initiative aiming to give universal  access to a large universe of documents.</P>

    <P>Everything there is online about W3 is linked directly or indirectly to this document, including an <A
    HREF="Summary.html">executive summary</A> of the project, <A
    HREF="Administration/Mailing/Overview.html">Mailing lists</A>, <A HREF="Policy.html">Policy</A>,
    November's <A HREF="News/9211.html">W3 news</A>, <A HREF="FAQ/List.html">Frequently Asked
    Questions</A>.</P>
</BODY>
```

# The early browser

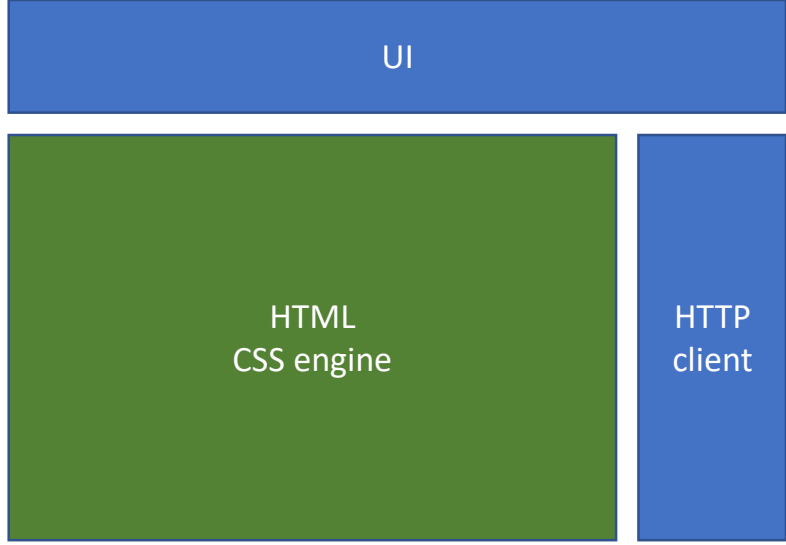


# A basic model…
# that has been able to evolve

- Towards more security: HTTPS
- Towards more dynamic and interactive pages (Rich Internet/Web Application): the application is partially animated by the browser rather than by the remote server

# Separation of the content from the style/presentation

- CSS: Cascading Style Sheet
  - Multiple possible presentation for a content
  - Homogeneous presentation of multiple contents

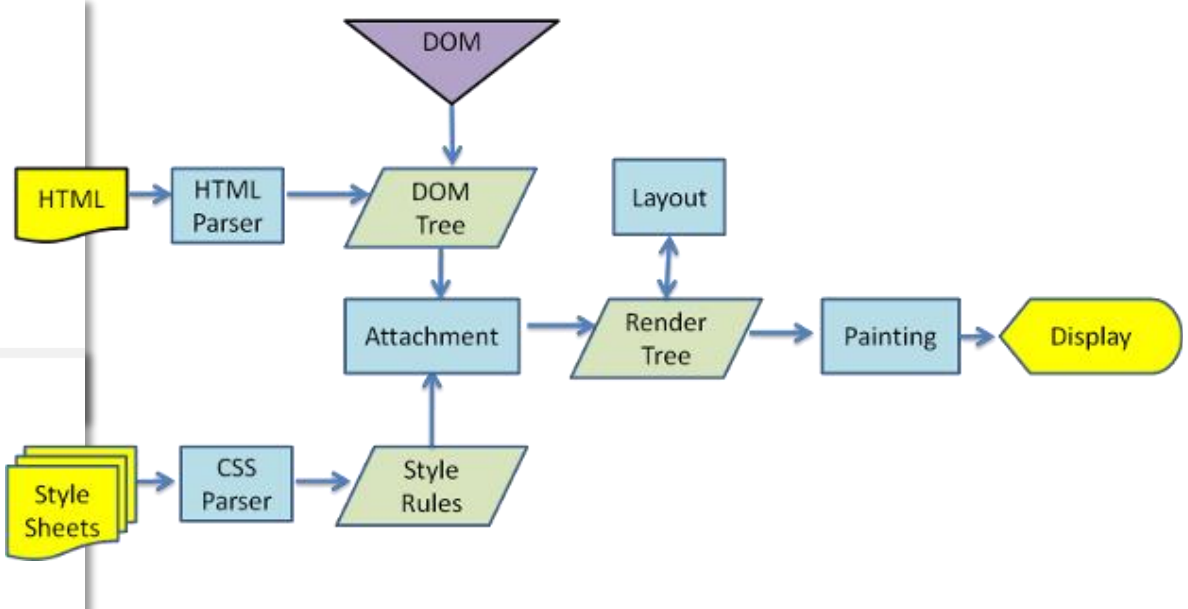



```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

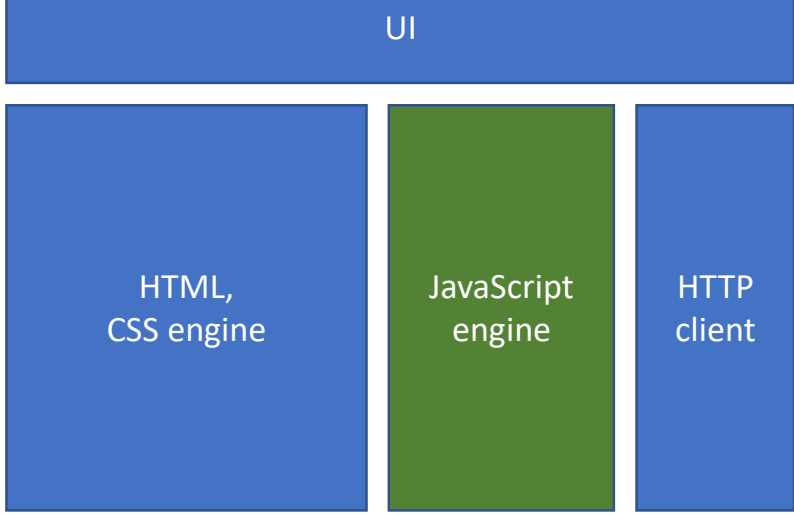
```css
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
```



# Integration of scripts on the client side

- <script> tag within the HTML
- Interpreted by the browser

→ Dynamic HTML (DHTML)




```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>

<p>JavaScript can change the content of an HTML element:</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo">This is a demonstration.</p>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Hello JavaScript!";
}
</script>

</body>
</html>
```

# Javascript

- Interpreted language
- Created in 1995 by Netscape
- Originally interpreted on the client side (now also used on the server side, in mobile app, on desktop applications, IoT… *« any application that can be written in JavaScript, will eventually be written in JavaScript »*)

# DHTML

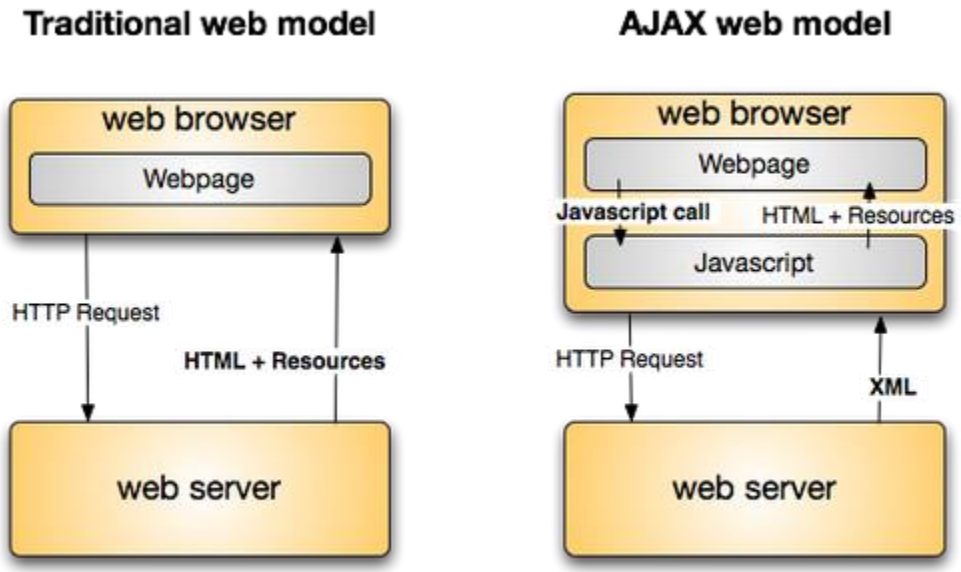
- Principle:

  Interact with the DOM to change appearance or content in response to user actions (click...)

- Usage:
  - Validate (or indicate errors to the user) a form before sending the data to the server
  - Create presentation animations (modal windows, image carousels...)
  - Facilitate navigation (drop-down menus...)

# AJAX

- *Asynchronous JavaScript + XML (*[XML] → most often replaced by JSON for structuring transported data)
- Requests the HTTP server in the background
- Allows you to change part of the page content without reloading the whole page.
  - Less latency for the user
  - Less traffic and server load
- Aggregates data from different HTTP servers before presentation



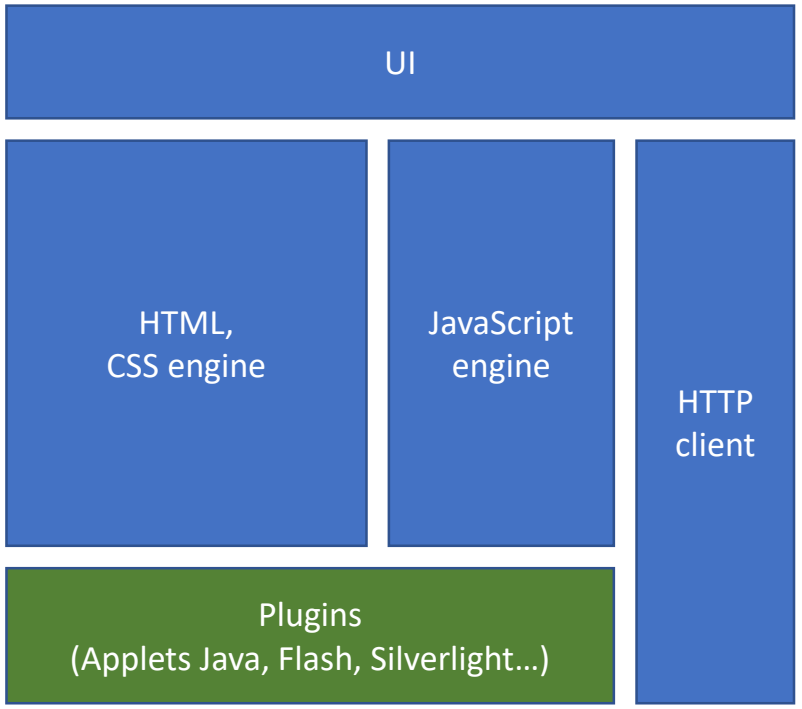

https://derivadow.com/2007/01/05/ajax-what-is-it-its-not-dhtml/

# Drawbacks

- Code is visible
- It’s difficult to index the pages
- Web caches are not used as much
- It’s difficult to navigate back (navigation history points to the original page)

# Applets

- Tag (object, embed, [applet]) or script referencing a small compiled application
- Executed by the browser in a dedicated space




```
<html>
  <head>
    <script type="text/javascript">
       swfobject.embedSWF("myContent.swf", "myContent", "300", "120", "9.0.0");
    </script>
  </head>
  <body>
    <div id="myContent"> <p>Alternative content</p> </div>
    <object width="400" height="50" data="bookmark.swf"></object>
    <embed width="400" height="50" src="bookmark.swf">
    <applet code="Bubbles.class" width="350" height="350">
          Java applet that draws animated bubbles.
    </applet>
    <object width="300" height="300"
          data="data:application/x-silverlight-2,"
          type="application/x-silverlight-2" >
         <param name="source" value="SilverlightApplication1.xap"/>
    </object>
  </body>
```

# Evolution doesn’t stop: HTML5

- does not only focus on the contents of web pages but also on interactivity (web application oriented)
- Introduction of new types

  for the input tag : tel, email...
- Introduction of new tags

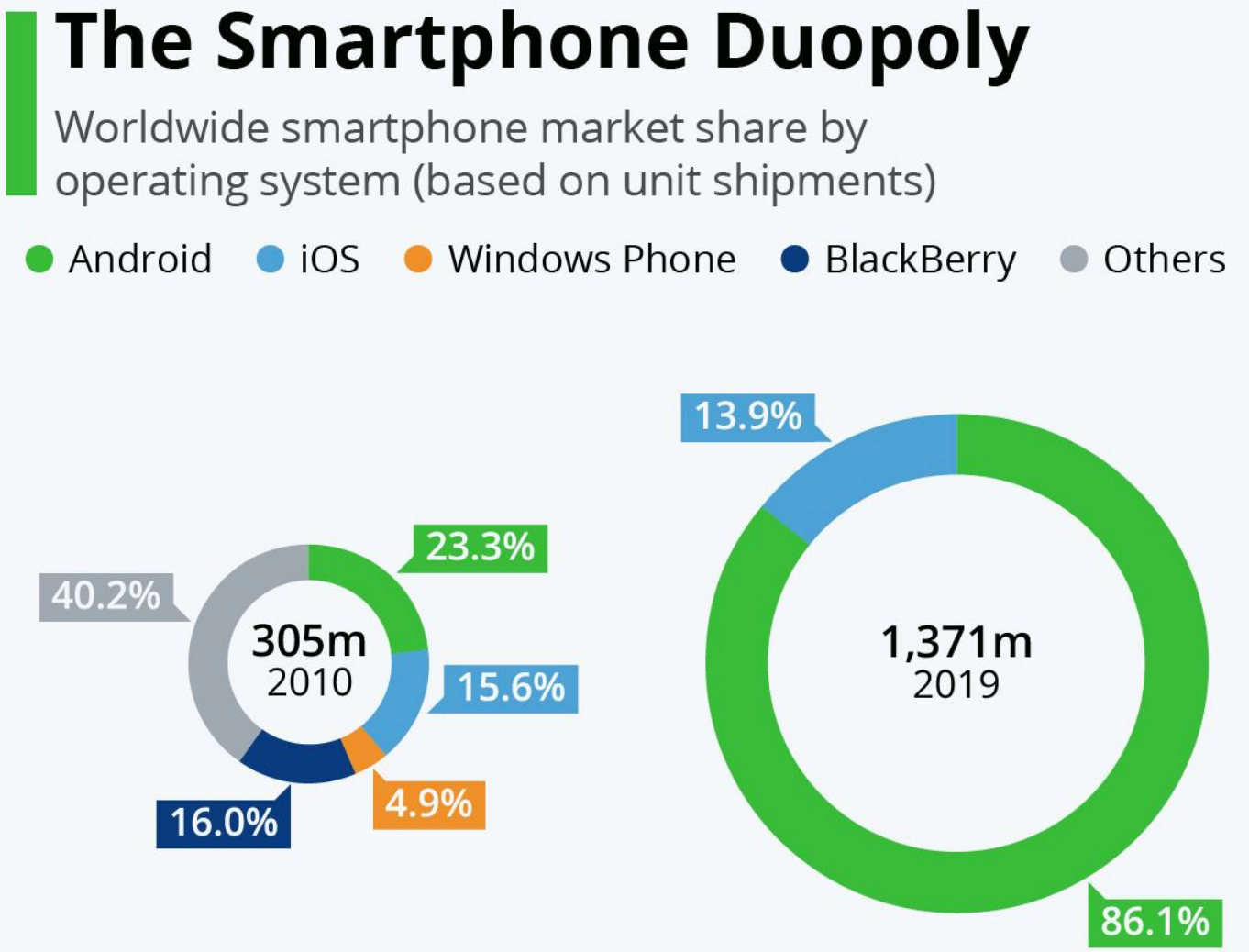
  video, track, canvas...
- New Javascript APIs:
  - 2D drawing (with canvas), 3D content (third-party WebGL API), multimedia
  - local persistence, local SQL database
  - content editing, drag-and-drop
  - browsing history
  - geolocation
  - …


https://iphonesoft.fr/2019/07/22/french-bank-lancement-officiel-ios-android

Special case

# I. A. 3. Mobile Terminals

# Market for mobile terminals




https://www.statista.com/chart/3268/smartphone-os-market-share/

# The specificities

- Intermittent, variable capacity connection
- More limited resources
- Different OSes and development environments, that evolve very quickly

What are the consequences for the application architecture?

# Not always connected

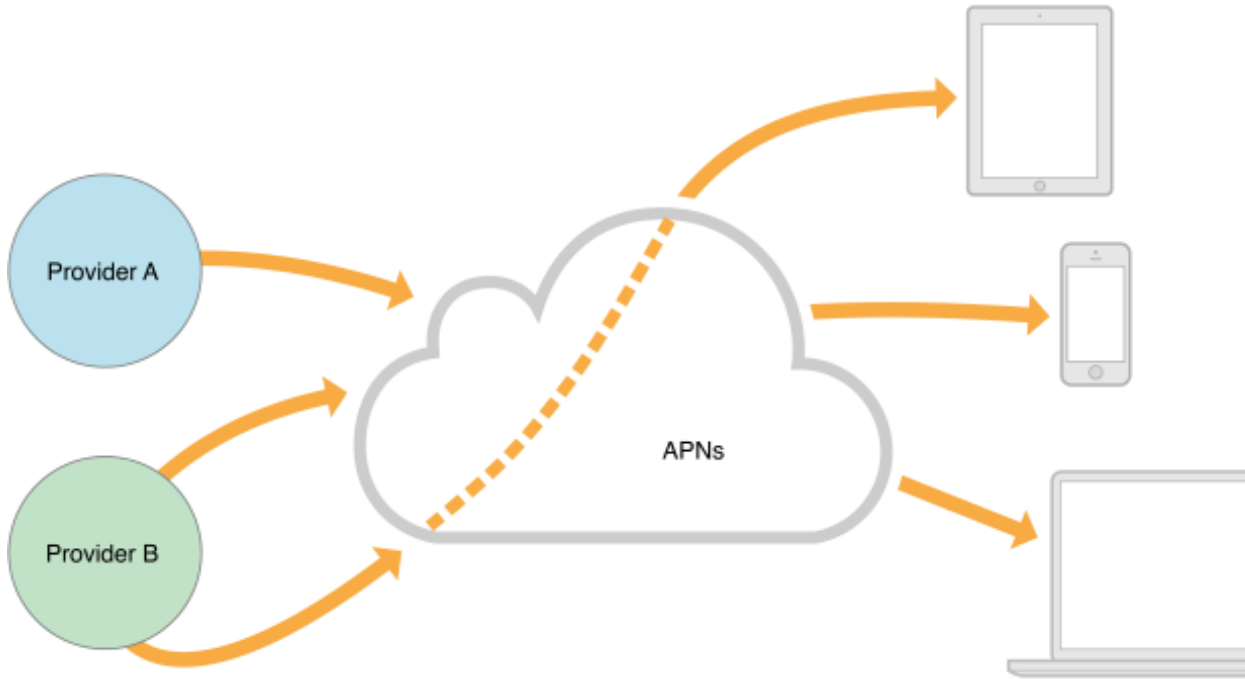
- Exploit the "disconnected" capabilities of HTML5

  Web Storage API
- Reduce traffic:
  - Lighten the payload (JSON, less verbose than XML)
  - Cache implementation
- Proactively take advantage of login periods:

  API to transfer data / update in the background, even when the app is not in use or not even started

# Push services

- Notification / message pushed by the server to a (previously registered) client interface
- Destination:
  - A single user
  - A group
  - A topic
- No guarantee of delivery



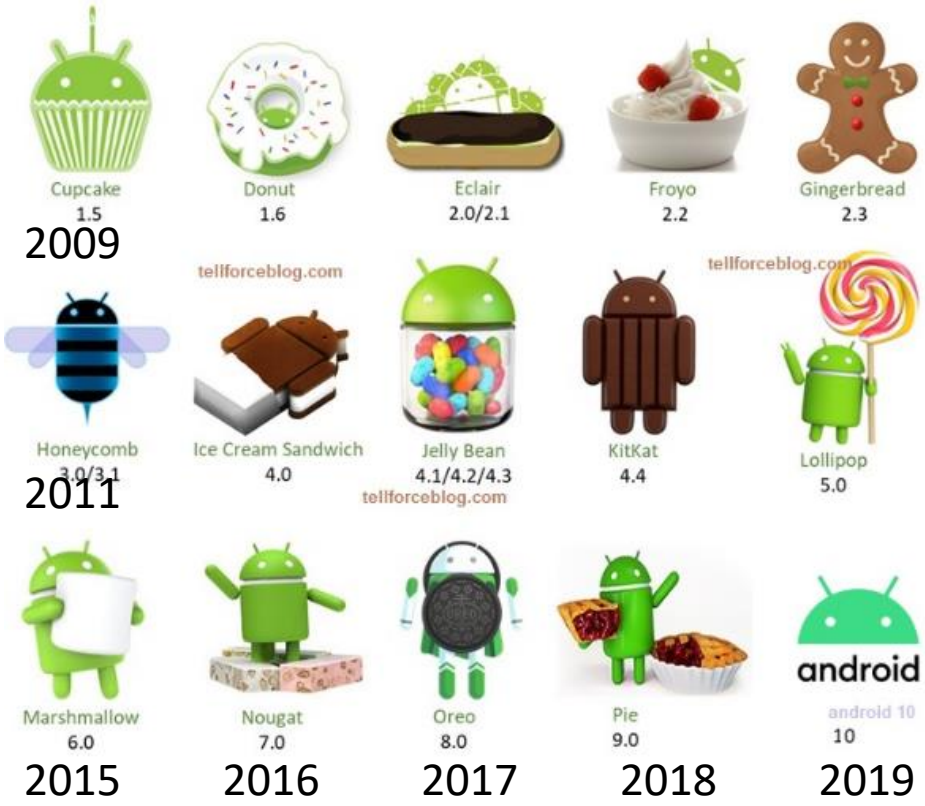

- Main notification services:
  - Apple Push Notification service
  - Google Firebase Cloud Messaging service

https://webdevdesigner.com/q/how-do-ios-push-notifications-work-44432/

# Scarce resources

- Carefully handle the life cycle of the app which can be interrupted at any time
- JSON parsing requires less resources than XML
- Restrict the App to simple functions where REST is sufficient vs SOAP that’s more rich (transactions...) but not natively supported and difficult to integrate in Javascript (vs native JSON support in Javascript)

# OS… a frantic pace





https://www.tellforceblog.com/2019/10/all-the-android-operating-system-from-version-1-0-till-date.html

| | iPhone 2.0 | 3.0 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| **Version** | **iPhone OS 2** | **iPhone OS 3** | **iOS 4** | **iOS 5** | **iOS 6** |
| **Codename** | Big Bear | Kirkwood | Apex | Telluride | Sundance |
| **New APIs** | n/a | 1000 | 1500 | 1500 | n/a |
| **New Features** | n/a | 100 | 100 | 200 | 200 |

| 7 | 8 |
|---|---|
| Innsbruck | Okemo |
| 1500 | 4000 |

https://www.imore.com/ios-8-review

# Native App Solution

- developed specifically for a platform
- installed via a store

Advantages:

- can use all the capabilities of the equipment (microphone, camera, GPS...)
- operates offline
- great performances

Limitations:

- development effort x 2
- developed using the specific technology for  the platform
  - iOS :
    - proprietary system, Xcode IDE, Objective-C and Swift languages
    - impossible de restrict or customize the OS
    - no inter-app communication
    - no shared files
  - Android
    - open system, Android Studio IDE, Java or Kotlin languages

# Cross-platform solution

- “generated native code”: one source code → two built apps (one for each platform)
- Popular solutions: React Native, Google Flutter…

# Web App Solution

Web site intended to be browsed from a mobile terminal

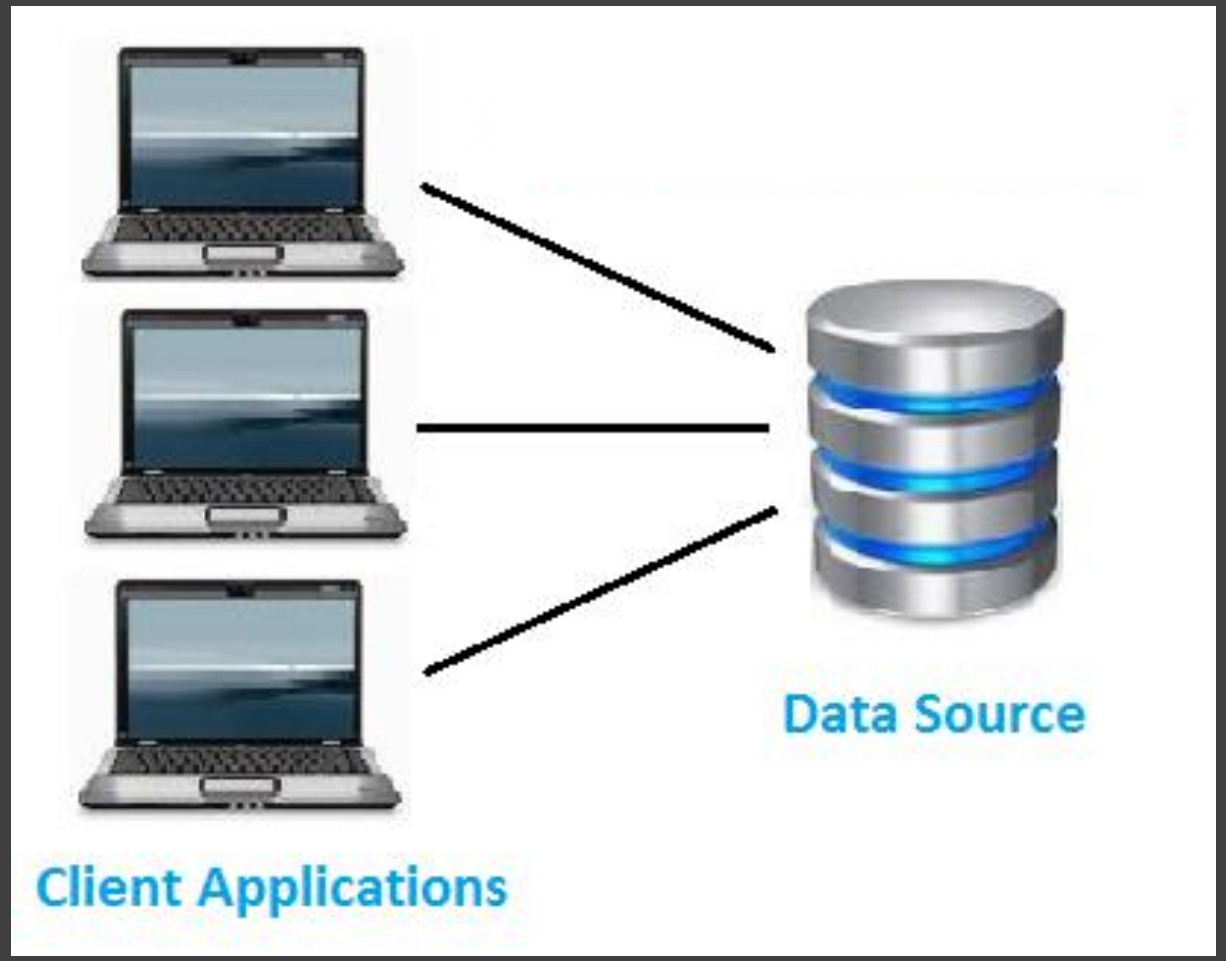
Advantages:

- available directly via the browser
- only one version for 2 OS
- web techno mastered by a lot of developers

Limitations:

- requires a connection (unless assets are cached, some storage possibilities via HTML5 )
- interactions with the underlying device are limited

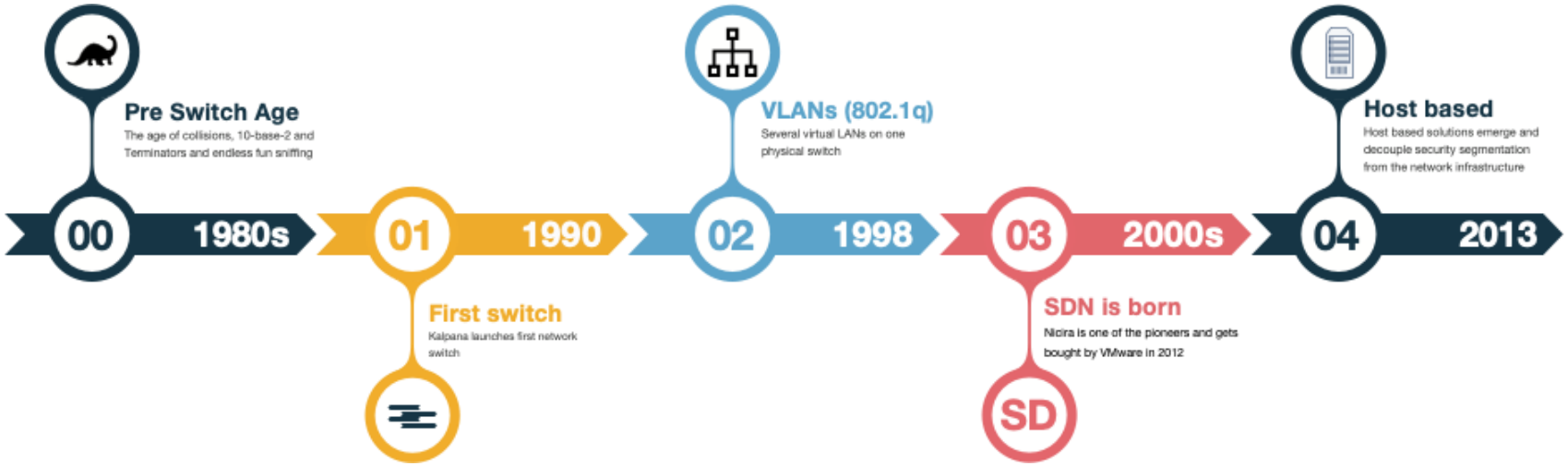
https://www.softwaretestingclass.com/what-is-difference-between-two-tier-and-three-tier-architecture/




# I. B. 1. 2-tier Architecture

# Emergence in the 90s

- More and more LAN (Local Area Networks), connecting more and more PCs
- Advances in DBMS (Database Management Systems)




https://microsegment.io/2019/08/19/a-short-history-of-segmentation/

https://www.educba.com/types-of-dbms/

# 1,5-tier architecture

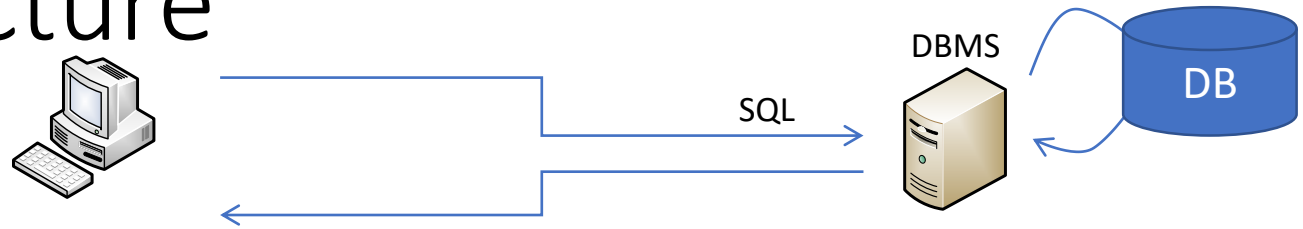
Workstations / **heavy clients**

| presentation |
| --- |
| application logic |
| data<br>(remote access) |

local network

file server
(structure = file hierarchy)

# 2-tier Architecture




- Heavy client workstation
- Local installation of the software mixing the presentation and business logic, with a thin data access layer
- Technologies :
  - Windows or Unix station
  - Lg:
    - Fortran, Cobol, C++, Java…
    - SQL, Prolog

- Central DBMS
  - Logical access to data
  - Additional services: security, transactions
  - Optimized for fast request processing
- Technologies:
  - Unix or Windows servers
  - Relational DBMS (Oracle, SQL Server, Informix)

# Roles of the DBMS

- Manages R/W on the physical disk
- Provides a logical language for data description and manipulation, checks data integrity and consistency
- Secures access to data
- Optimizes access to data
- Manages concurrent access to data

# Issues

- Deployment

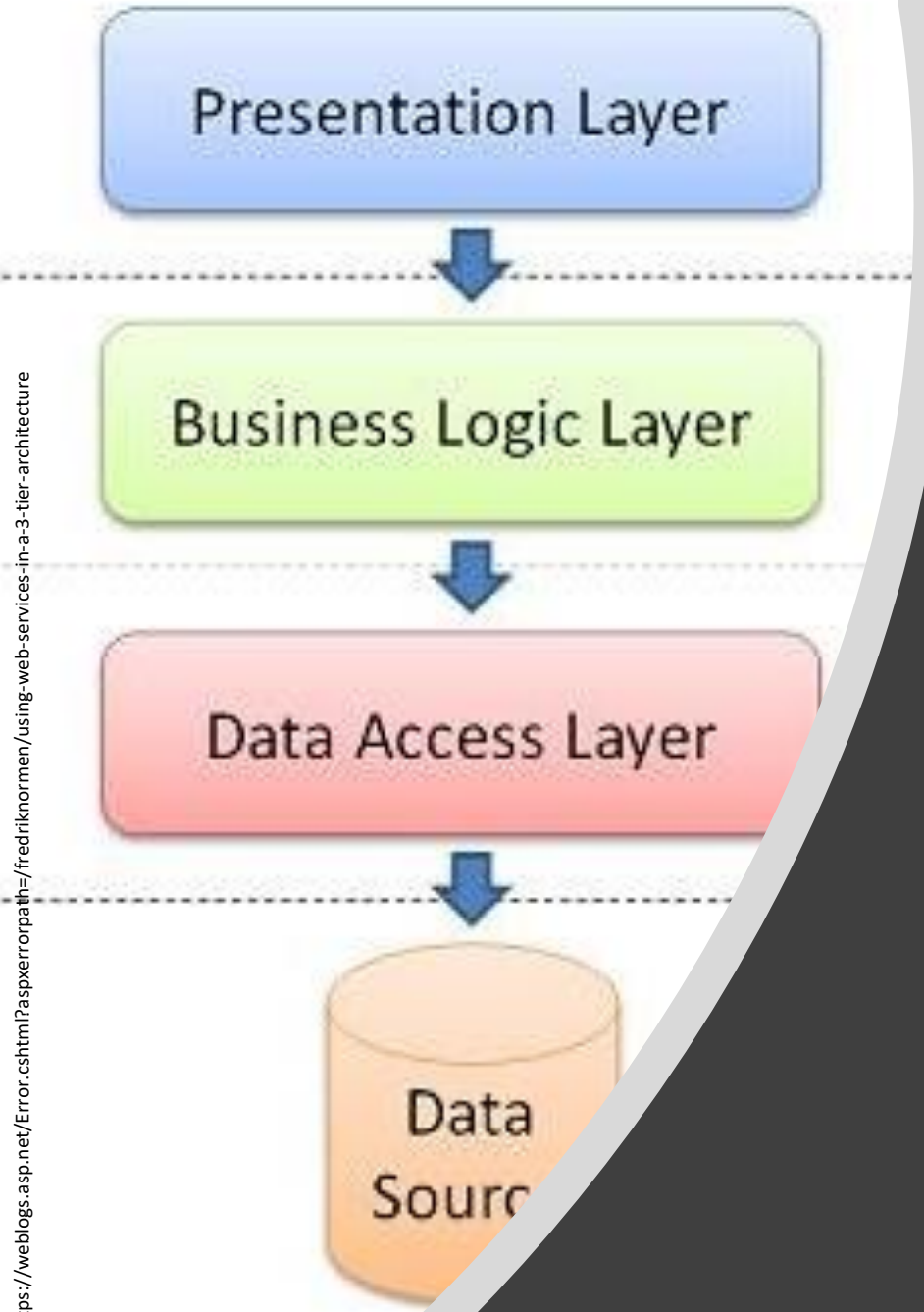
  requires the installation of a not-so-simple application on each client workstation:
  - time consuming/costly/tedious process (to be repeated at each upgrade)
  - heterogenous workstations → compatibility problems
- Maintainability

  complex client code mixing GUI, business logic, and some data access
  → inclination to over-rely on stored procedures:
    - complex to maintain
    - adherence to the physical model
- Reliability
  - FT mechanisms need to be implemented on both sides
  - few clustering / replication solutions
- Performance and scalability
  - Server = bottleneck

# Conclusion and perspectives

- led to the development of new tools, the acceleration of the development of RDBMS and SQL, the development of LANs
- triggered an evolution towards more flexible architectural proposals
- but is no longer relevant

  except maybe for very simple applications (forms…)

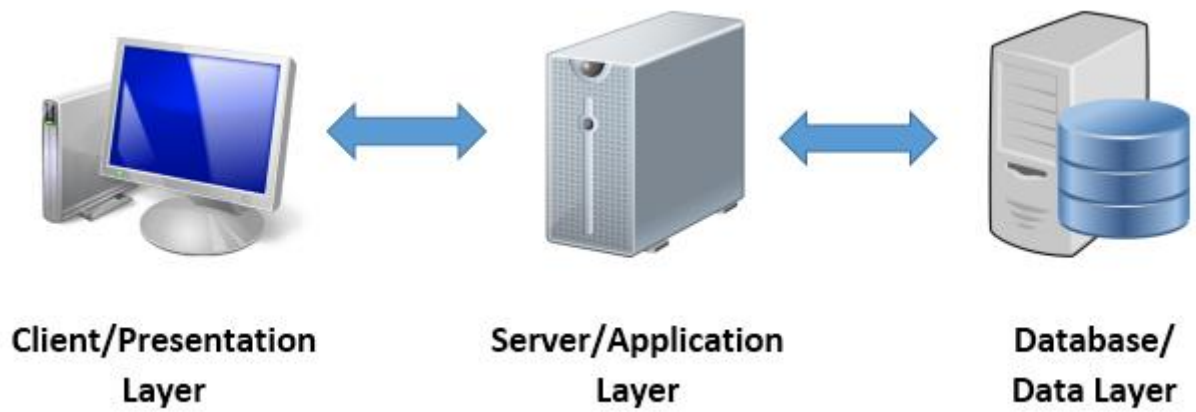



https://weblogs.asp.net/Error.cshtml?aspxerrorpath=/fredriknormen/using-web-services-in-a-3-tier-architecture

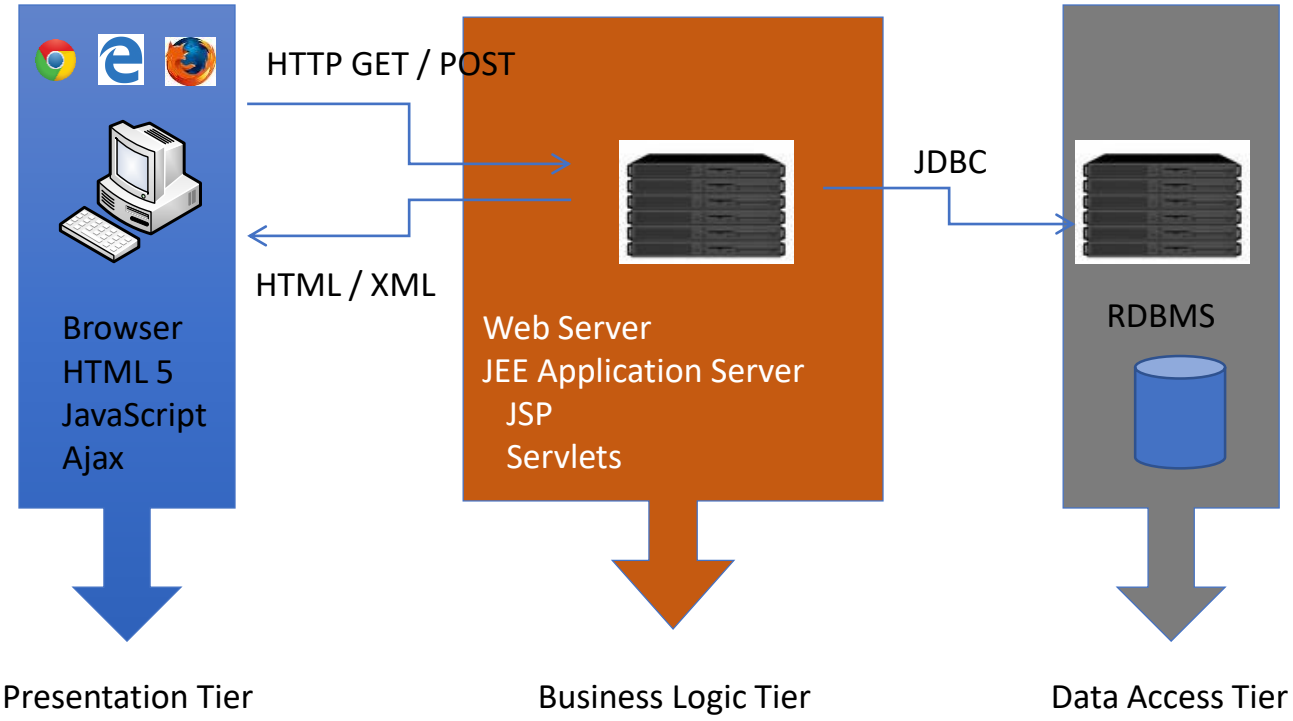
# I. B. 2.

# 3-tier to 5-tier Architectures

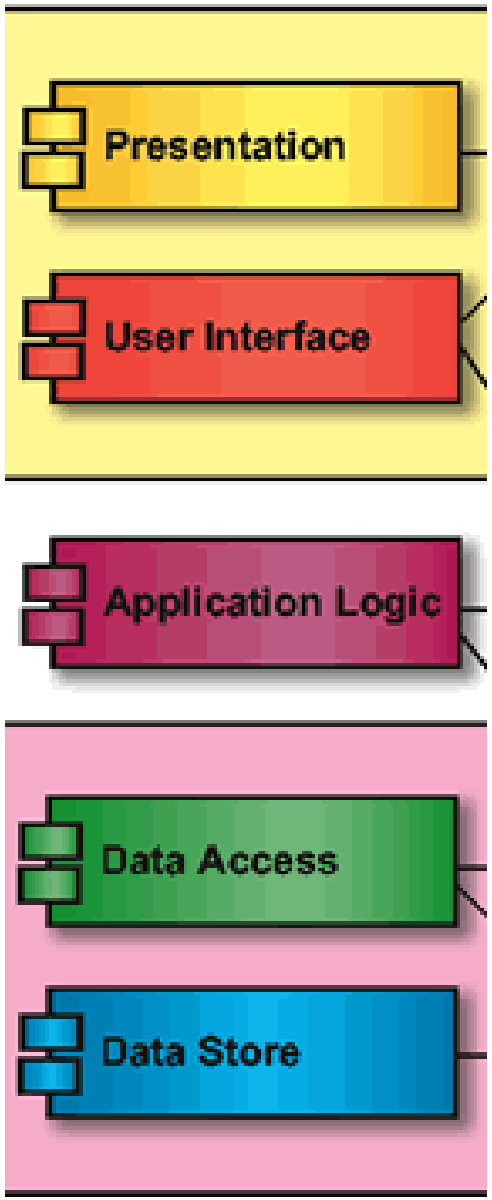
# 3-tier


Client/Presentation Layer
Server/Application Layer
Database/ Data Layer

# Example:
# A typical JEE Web Architecture



# 5-tier




look & feel of the application

actions performed when a button is clicked or data is received

hides the fact that several different data stores are used

https://pubs.opengroup.org/architecture/togaf8-doc/arch/chap31.html

# Perspectives

- Most widespread architecture
- Easier to maintain and evolve than the 2-tier architecture
- Remaining challenges to face: reliability, performance, scalability


# I. B. 3. Application Server

# Business layer: kind of stuck…

A lot of technical work to manage

- access to other applications of the IS
- fault tolerance
- load management
- security
- logging
- etc...

instead of concentrating on... the business process!

Developers are looking for :

- more simplicity
- reuse
- interoperability
- standard APIs

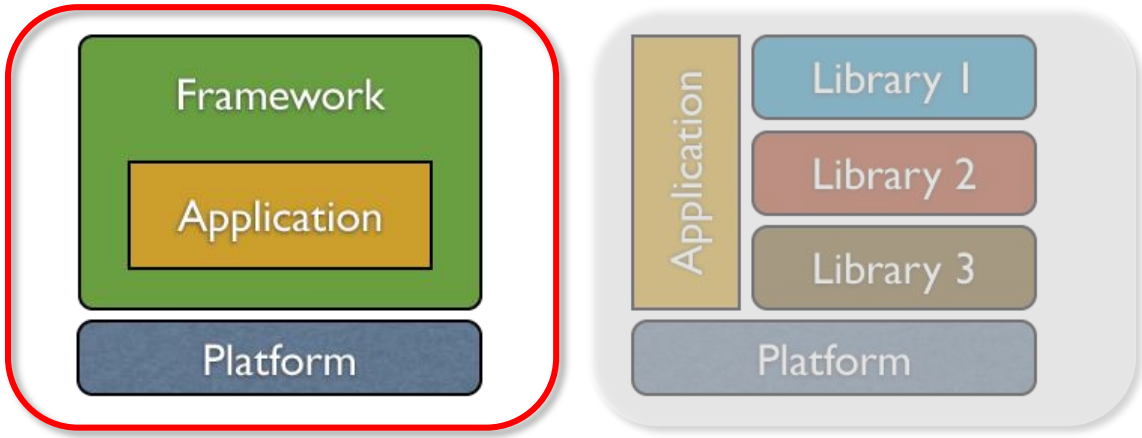
# From middleware to application server

These "architecture services" can be provided by **middleware**

- either internal
  - rarely address all problems
  - complex to develop, not very efficient, expensive to maintain
- either commercial
  - Oracle, IBM, Microsoft... have been offering middleware for several years...

# Application server

- a standard, integrated and consistent middleware stack
- infrastructure software providing execution context to application components
- Framework (as opposed to library)



# Main services offered to applications

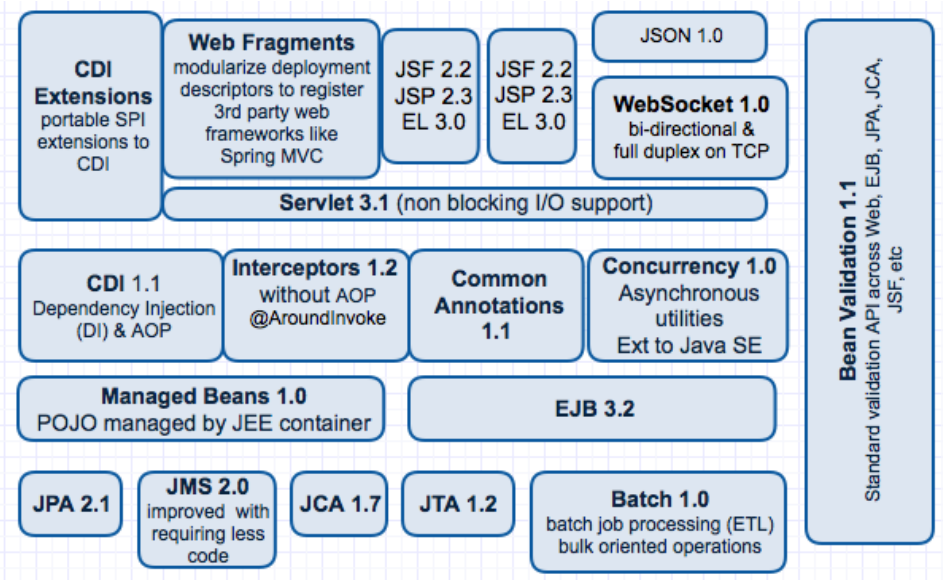
- Access to DBs (connection pool)
- Connection to the IS, to business applications via suitable connectors
- Synchronous (DCOM, Corba...) or asynchronous (MOM) communication with other applications
- Management of transactions, sessions, recovery on failure
- Security (authentication)
- High availability and scalability (clustering, load balancing, failover)
- Management via unified management console
- …

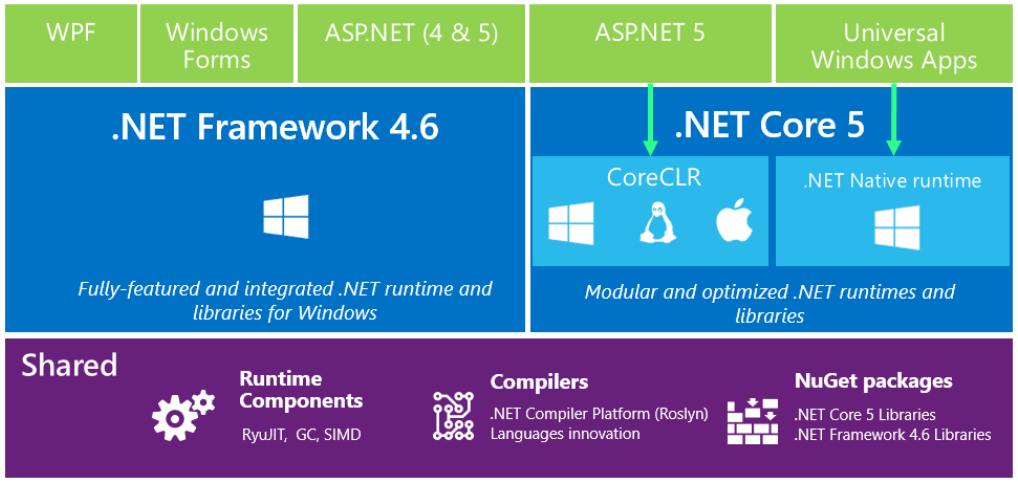
# JEE and .NET frameworks

- A collection of services to develop an application
- Virtual Machine (JVM or CLR) → OS isolation

- JEE stack
  - Multi platform
  - Mono language (Java)

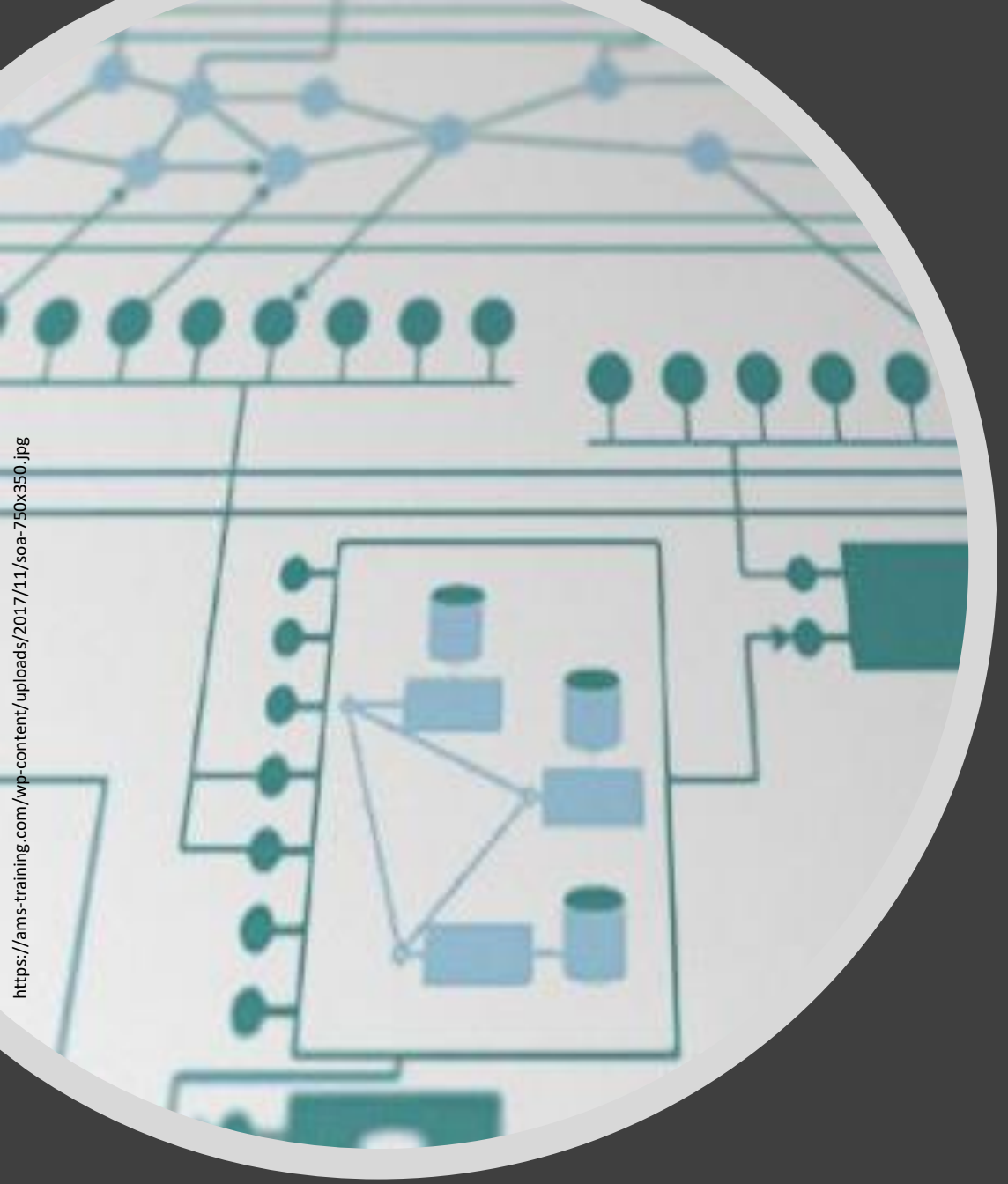
- .NET stack
  - Single platform (Windows)
  - Multi language (VB, C#, C++ etc.)



# Independence above all!

- Objective: maximum independence of the application code
  - No static link between application code elements (components)
  - No static link between application code elements and platform services
  - Avoid the use of "technical" code in the application code
- Component deployment support
  - Installation of applications and libraries (.ear, .war, .rar, .par archives)
  - Activation of applications and libraries
- Distinction between the development part (business) and the technical part (administration)
- Important step towards more reusable application code!



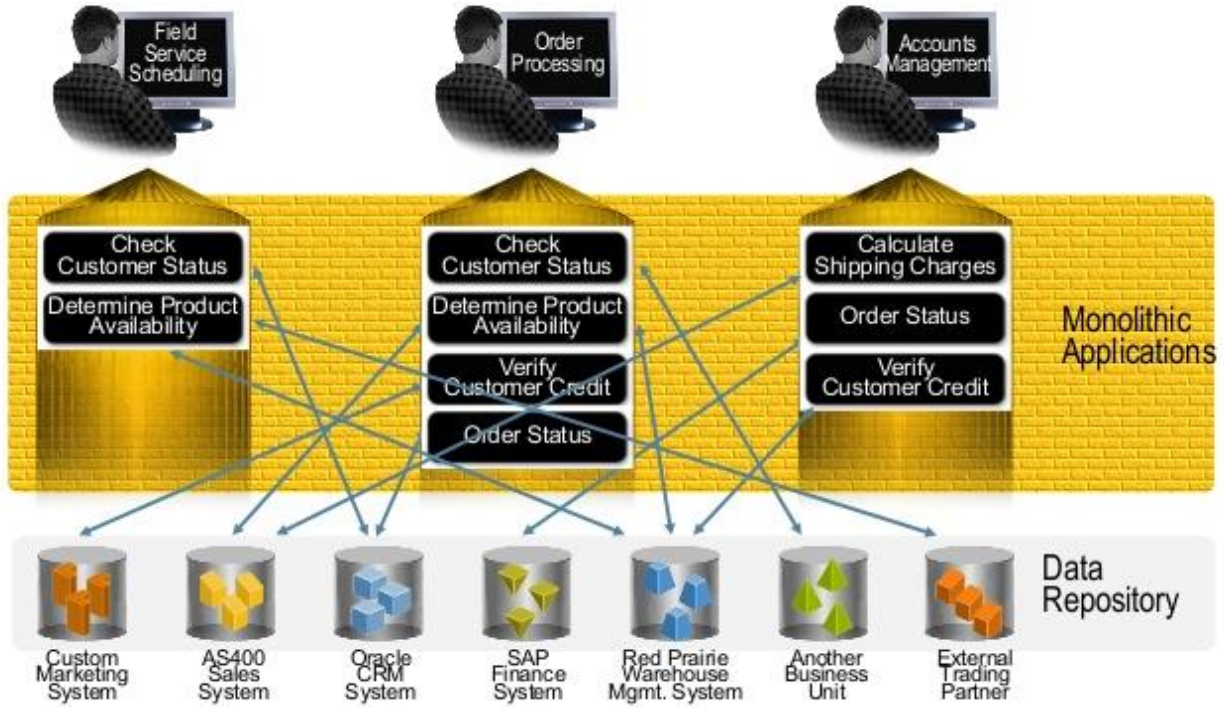
https://ams-training.com/wp-content/uploads/2017/11/soa-750x350.jpg

# I. C.
# Service Oriented Architecture

# Entropy and silos

- Entropy is part of the IS.
- The evolution of information technology has led to heterogeneous IS that are difficult to maintain:
  - Lack of internal communication
  - Data duplications and synchronization problems
  - Duplicate code
  - Technological heterogeneity
  - Maintenance cost
  - Low reuse of legacy
  - Difficulties to open to business partners (consumers, suppliers, resellers, regulators...)
  - …
- Users expect more and more from the IS.

→ How can we better manage and homogenize the IS, which grows and changes naturally?




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

# Opposite trends

**Get rid of silos**

- Technical renovation
- Auditability and regulatory constraints
- Performances
- Fluidification of exchanges
- Secure access and availability
- Consistency of internal and external information

**Create micro-silos**

- Innovative organizations need to move faster than their competitors and make just-in-time adjustments to gain new markets.
- Take advantage of new technologies
- Push forward new products and services that can grow independently of each other
- Adapt to the consumer/customer experience according to his expectations (price, lead time, quality, etc.)

often involves vertical actions: do it fast, even if it means creating new silos

# SOA to deal with entropy

In response to this antagonism:

- Service-Oriented Architecture (SOA) and micro-services
- Methodology for designing software as a service (12-factor method, Antifragile)

# What is a service?

- (usually remote) access to a business function (data, rules, process, batch)
- black box for its clients, accessed through a formalized API and a QoS contract: low coupling (clients do not need to understand or master the implementation technology, access through standards)
- software component managed (deployed, updated,...) as an independent unit with its own life cycle
- can access other services

# An injunction

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols…
- All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

- **Anyone who doesn't do this will be fired**.

Jeff BEZOS – AMAZON (2002)


"search" Service
"Customer" Service
"shopping" Service
"best sales" Service
"recommendations" Service
"similar" Service
amazon.fr
Toutes nos boutiques
Livres
Musique et DVD
Photo, Image et Son
Informatique et Logiciels
Jeux vidéo et Consoles
Jouets et Enfants
Cuisine et Maison
Montres et Bijoux
Nos meilleures ventes Informatique et Internet
À découvrir
Vous avez regardé
Vous apprécierez peut-être également
LIVRAISON GRATUITE
pour des millions d'articles
Le projet d'urbanisation du S.I...
Les architectures orientées...
Des soirées torrides !

# Web service definition

generic definition for 2 families that have in common:

- software functionality
- accessible via the network
- provided to another computer program
- self-descriptive
- low dependency on technologies and protocols

# WS: benefits for the client

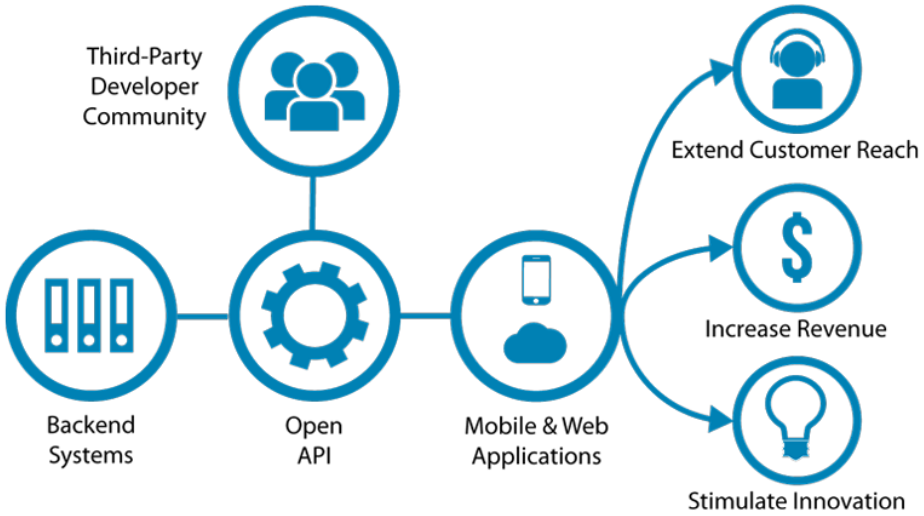
- take advantage of third party data or algorithms without having to develop, update, host etc.
- pool of composable and interchangeable bricks

# WS: risks for the client

- service unavailability
- poor performance
- security issues
- uncertain sustainability
- a free service that becomes a paying service

# motivations of WS providers

- revenues
- brand image
- increase its audience
- develop new usages
- outsource R&D and then buy the best startups




but potentially...

- lower advertising revenues
- loss of control over the final UX

# 2 “flavors”

- Process-oriented web services
  - SOA in IS → remote procedure call
  - *WSDL, SOAP protocol*
- Resource-oriented web services
  - Web → access to remote resource
  - *JSON, HTTP, REST architectural style*

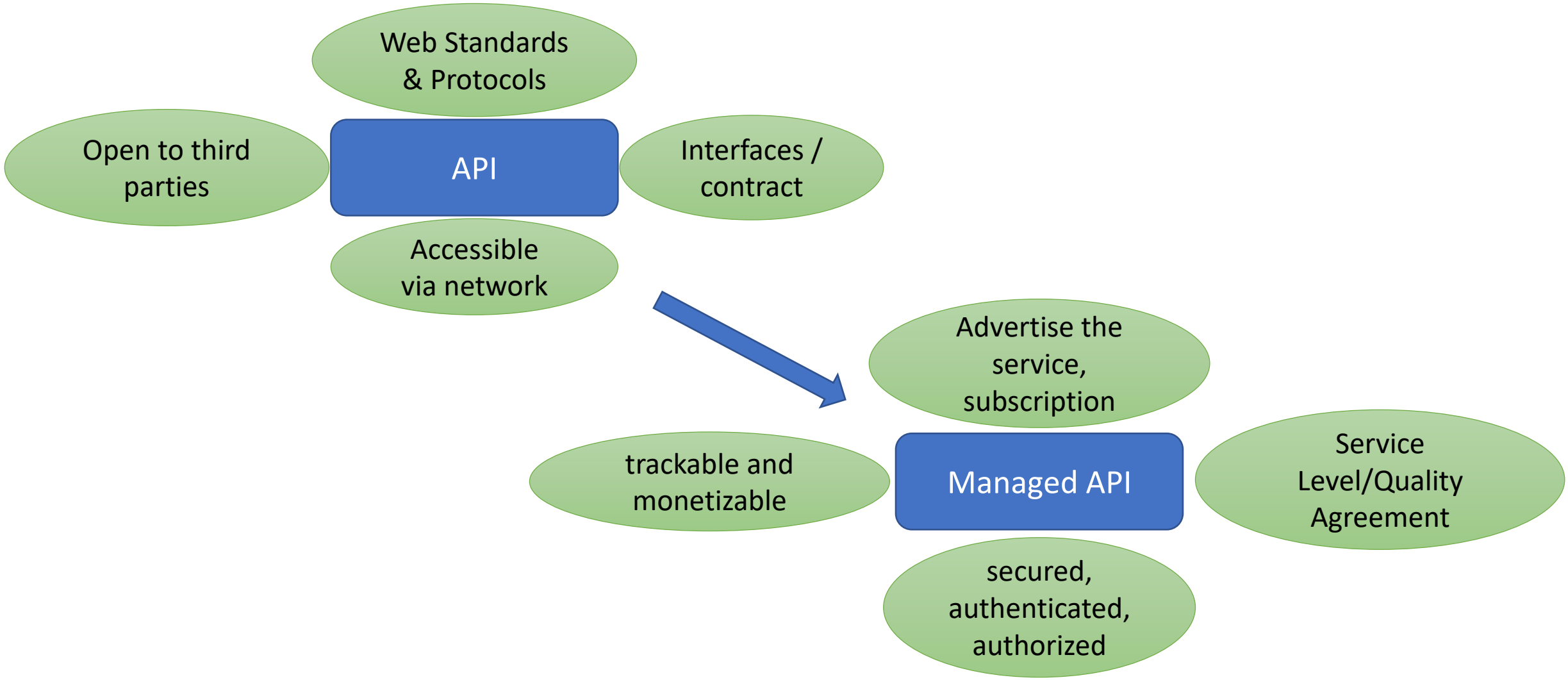
# What is an API?

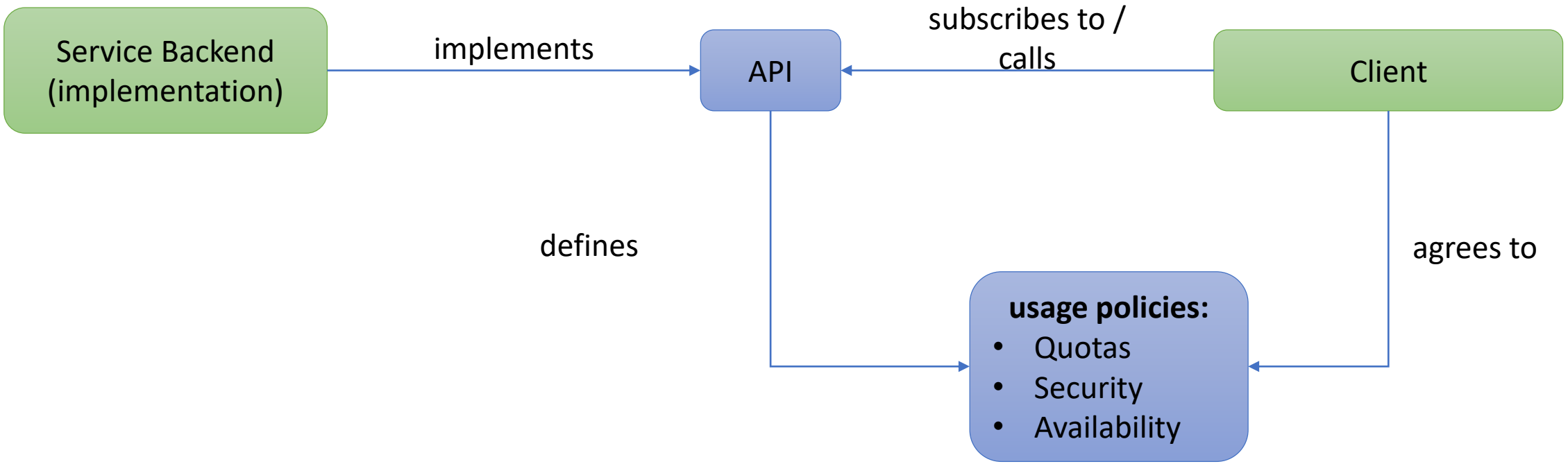
- a means of exposing company resources via the Internet to external or internal (software) consumers
  - Well-defined interface: contract
  - Easily accessible by third parties
  - Use of standard protocol(s)
- ≈ Web Service


- Programmatic access is considered at least as vital as human access, if not more so. API becomes more of a priority than IU :

  January 2019: 83% of the WEB traffic goes through APIs (a result to be mitigated by the important role of streaming APIs such as Netflix)

# From basic WS to managed API




Service Backend
(implementation)
implements
API
subscribes to /
calls
Client
defines
agrees to
usage policies:
• Quotas
• Security
• Availability

# Microservice

- delivers a complete business capability (anti-pattern: nano-service)
- designed, developed, deployed by a single team (Agile, DevOps)
- autonomous component, with own independent life cycle and own scalability goal
- synchronized with other capabilities through asynchronous data replication
- each team is free to select “best of breed” languages, tools, even cloud provider

**From a monolith (2010) to 500+ microservices & 50+ independant engineering teams (2015)**

**From 1 deployment a week (2001) to 50 millions a year (2017)**

**From 0 (2008) to 87 millions paying users (2018) with 800+ microservices & 90+ teams (600 devs)**

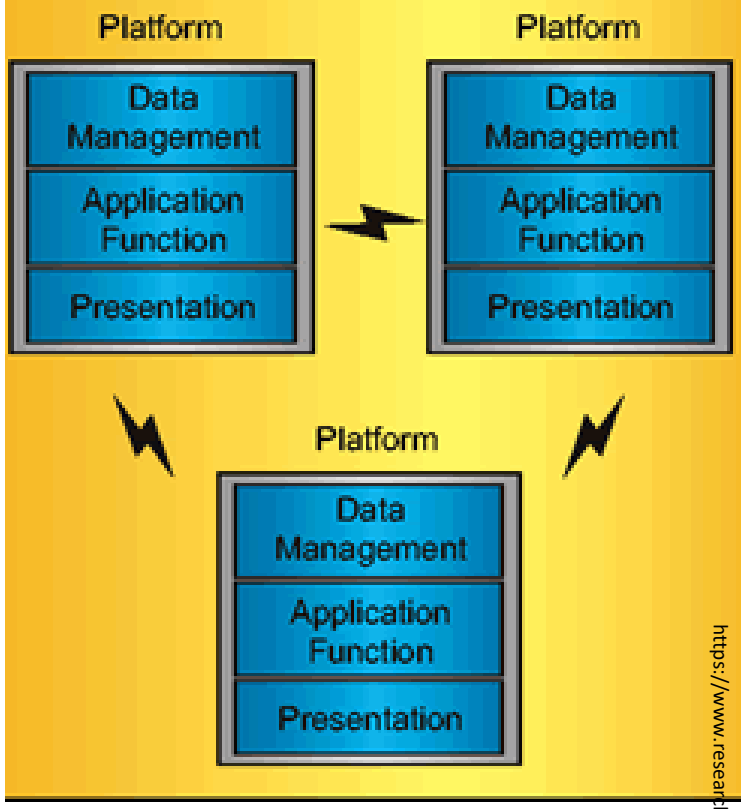
https://www.altexsoft.com/blog/engineering/using-microservices-for-legacy-system-modernization/

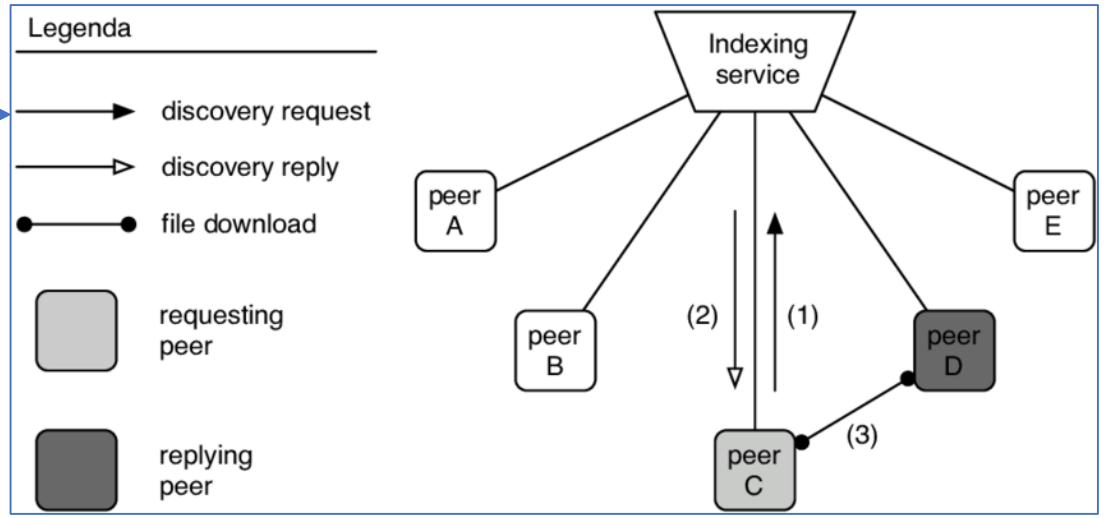
# II. 1.
# Peer-to-peer Architectures

https://blockchainfrance.net/2018/05/22/comprendre-le-web-decentralise/

# P2P Network

- Each computer hosts the complete application stack
- Each computer can simultaneously
  - consume services offered by others (act as a client)
  - provide services to others (act as a server)
- Variants:
  - "pure": equipotent peers
  - “hybrid”:
    1. Register with a central server
    2. Find peers by asking to the central server
    3. Direct data exchanges between peers




https://pubs.opengroup.org/architecture/togaf8-doc/arch/chap31.html

https://www.researchgate.net/figure/An-example-of-hybrid-P2P-network_fig2_251517264

# Advantages: example of file sharing

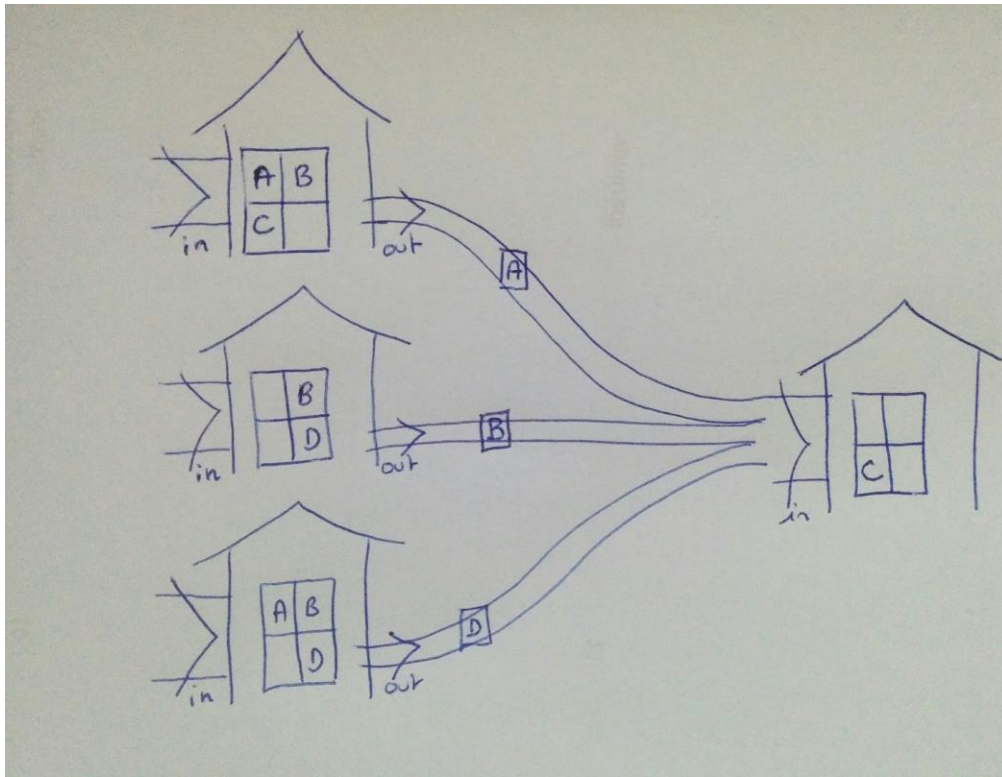
- Scale-up

  the more a file is requested, the more it is available

- Robustness

  The only single point of failure is the central server in the hybrid variant case

- System of "chunks":
  - "erases" the asymmetry of ADSL
  - limits the effects of abrupt disconnections from the supplier
  - a file can be globally available without anyone having the whole file



# Other usages

- Cryptomoney transactions
- Distributed computing: BOINC

# Difficulties

- Peer Discovery

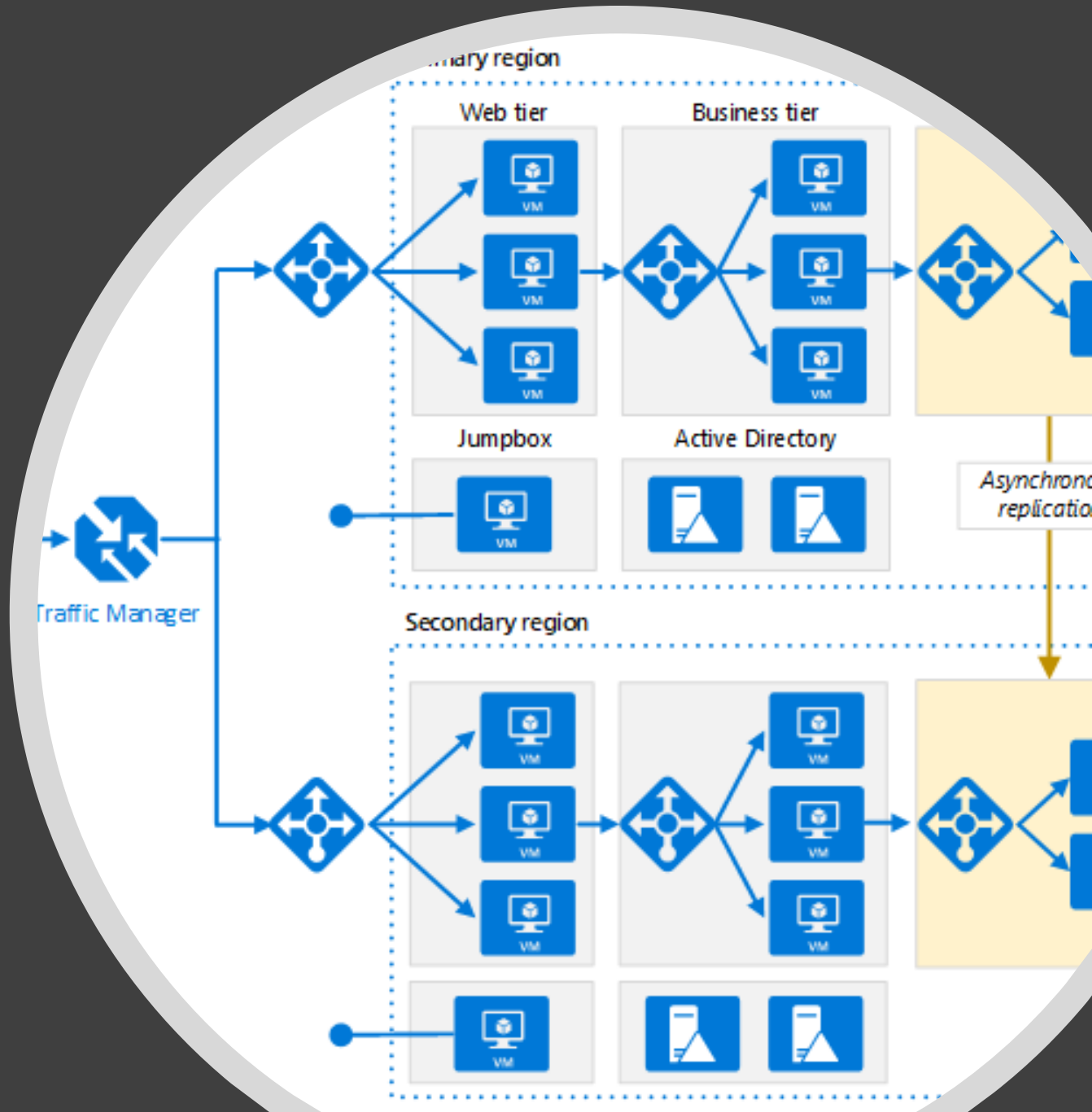
  solutions :

  - central node
  - Distributed Hash Table
- Free-riders

  solution: incentive mechanisms
- Peer volatility
- Confidence

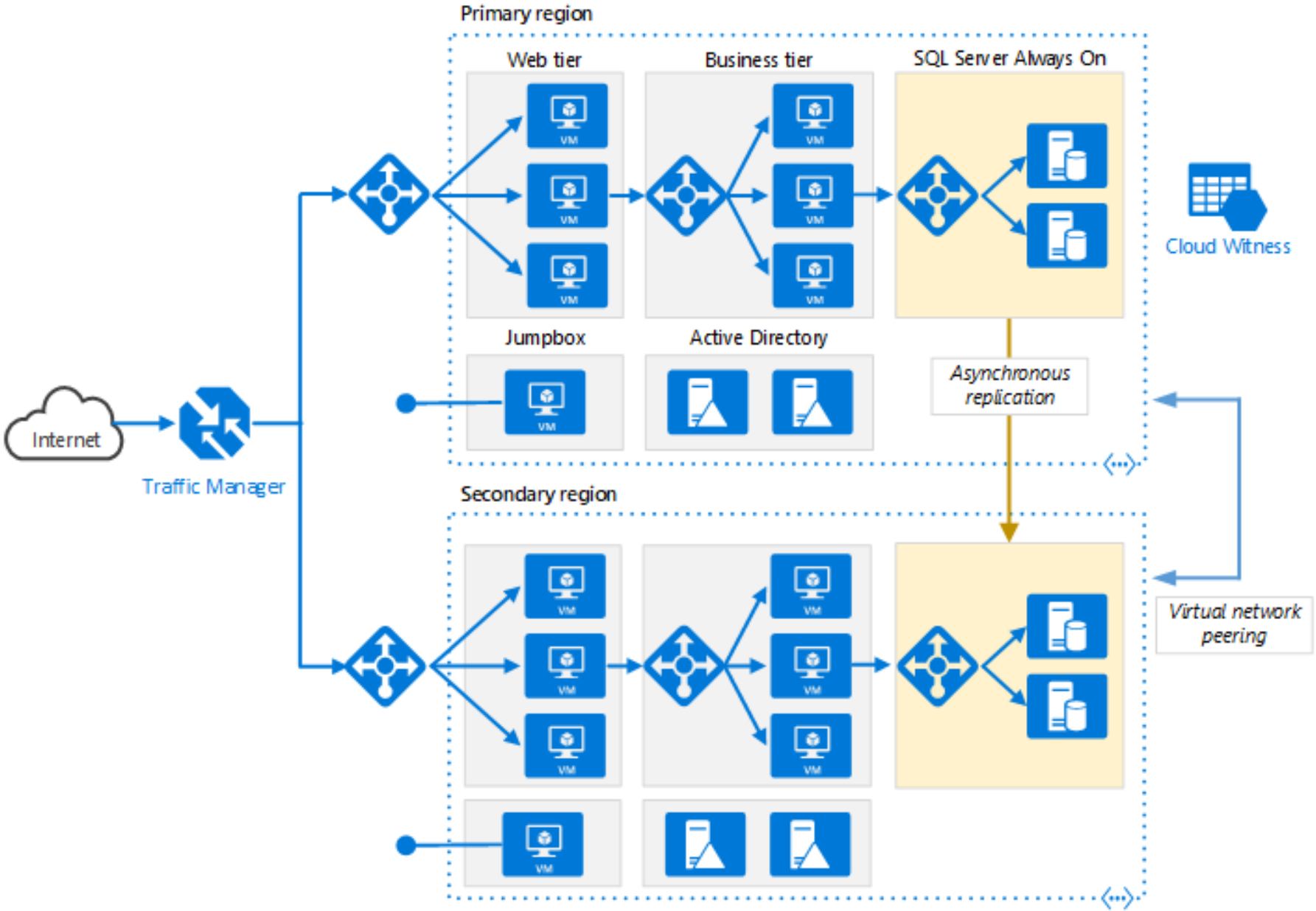
  solution: redundancy and comparison

# II. 2. n-tier Architecture



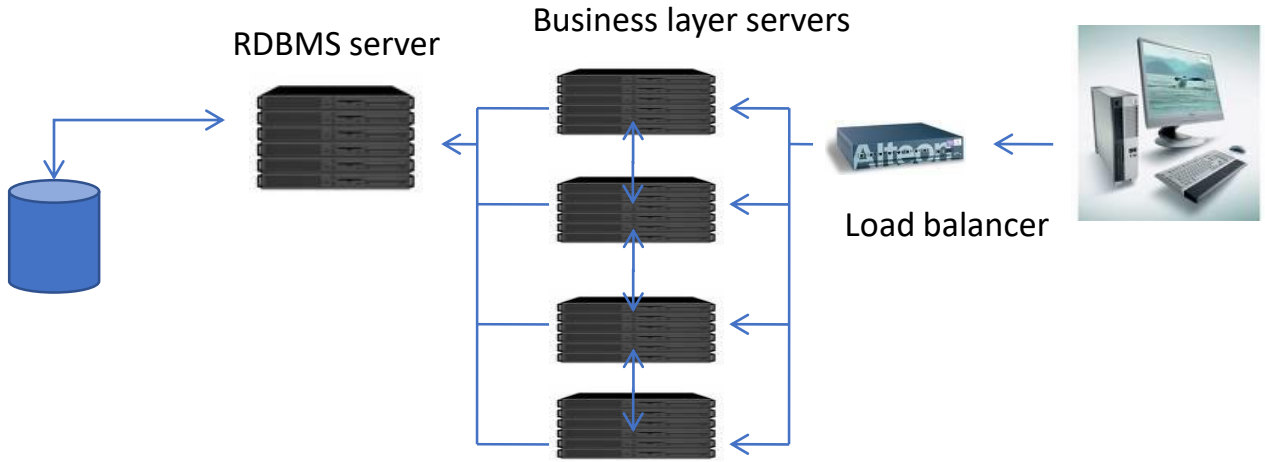

https://docs.microsoft.com/fr-fr/azure/architecture/reference-architectures/n-tier/multi-region-sql-server

# Multiplication of elements in a given layer

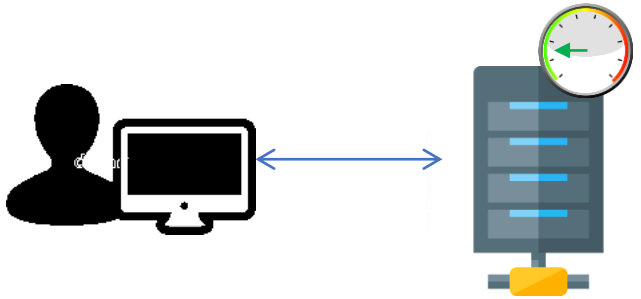


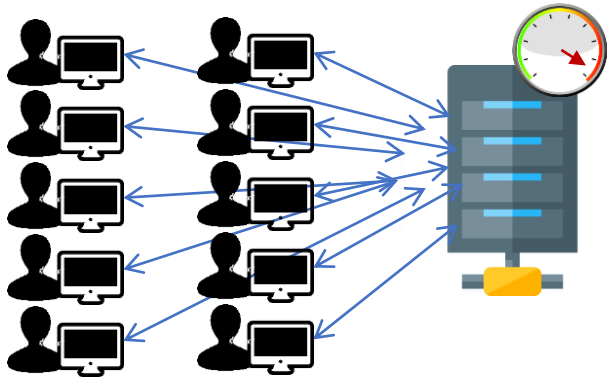

https://docs.microsoft.com/fr-fr/azure/architecture/reference-architectures/n-tier/multi-region-sql-server

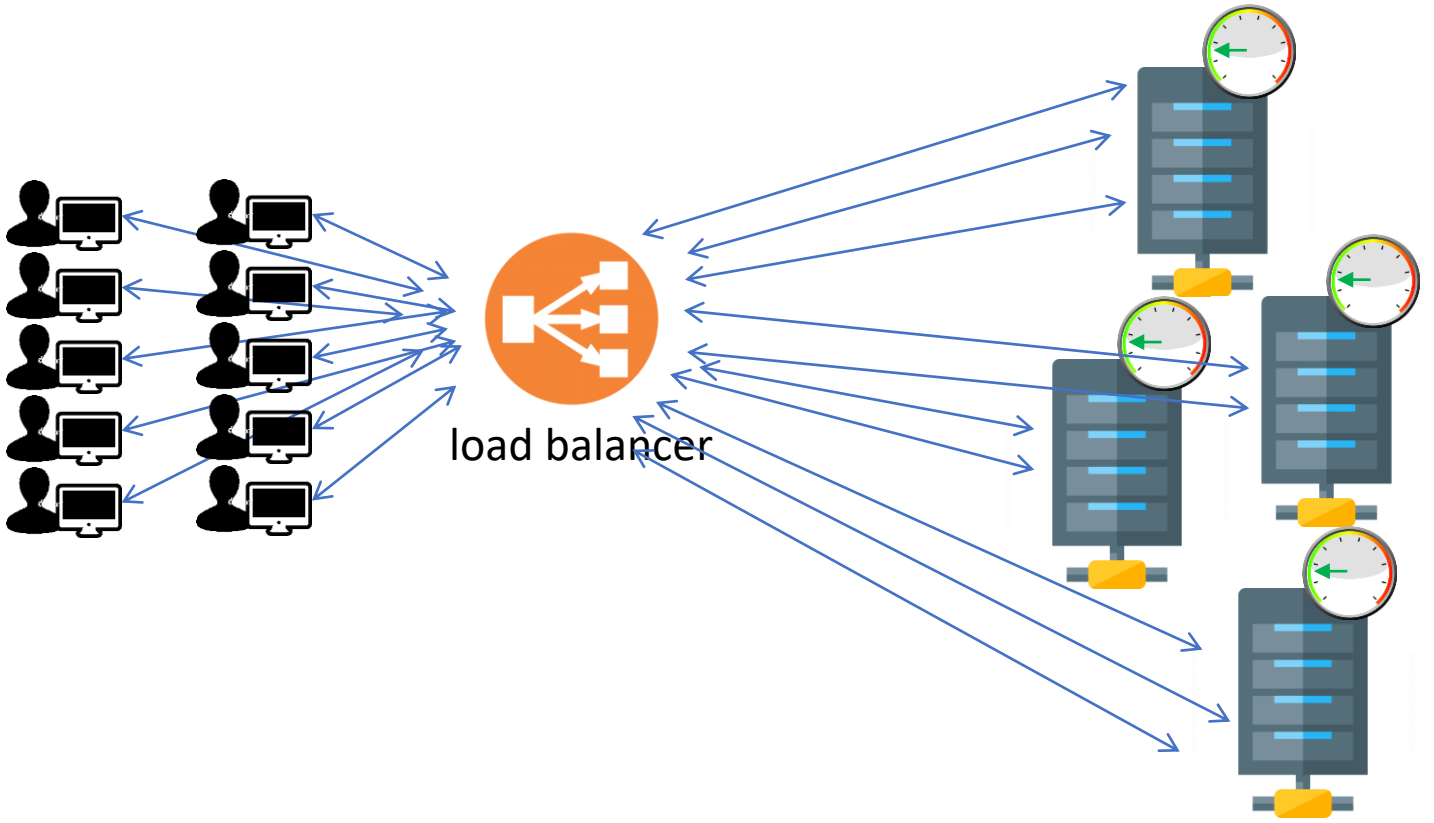
# Scale-out



# Scale-out

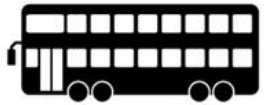




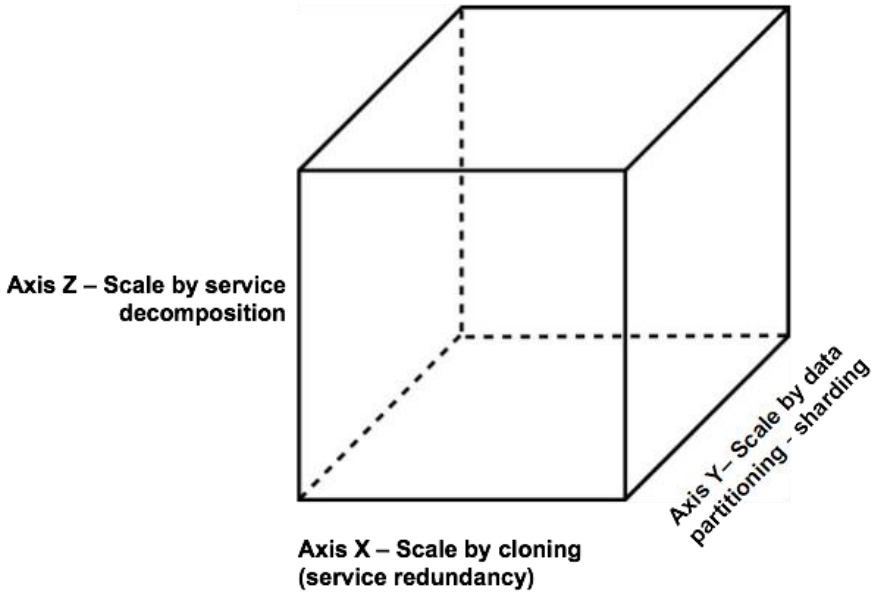


# Scalability

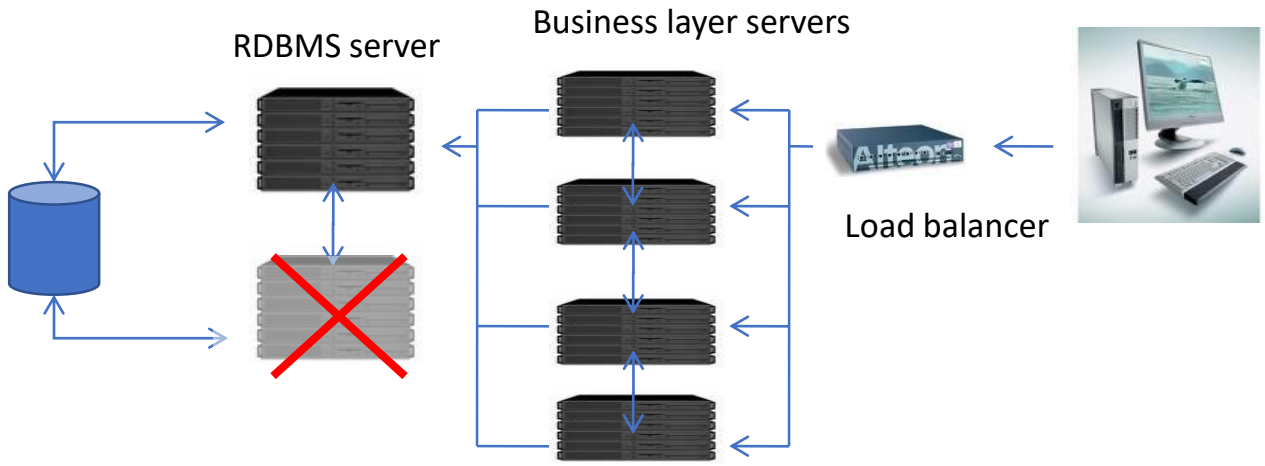
- Vertical vs. Horizontal Scalability
  - Vertical (scale up) = increase server power (CPU, memory, I/O)
  - Horizontal (scale out) = multiplication of resources + load balancing



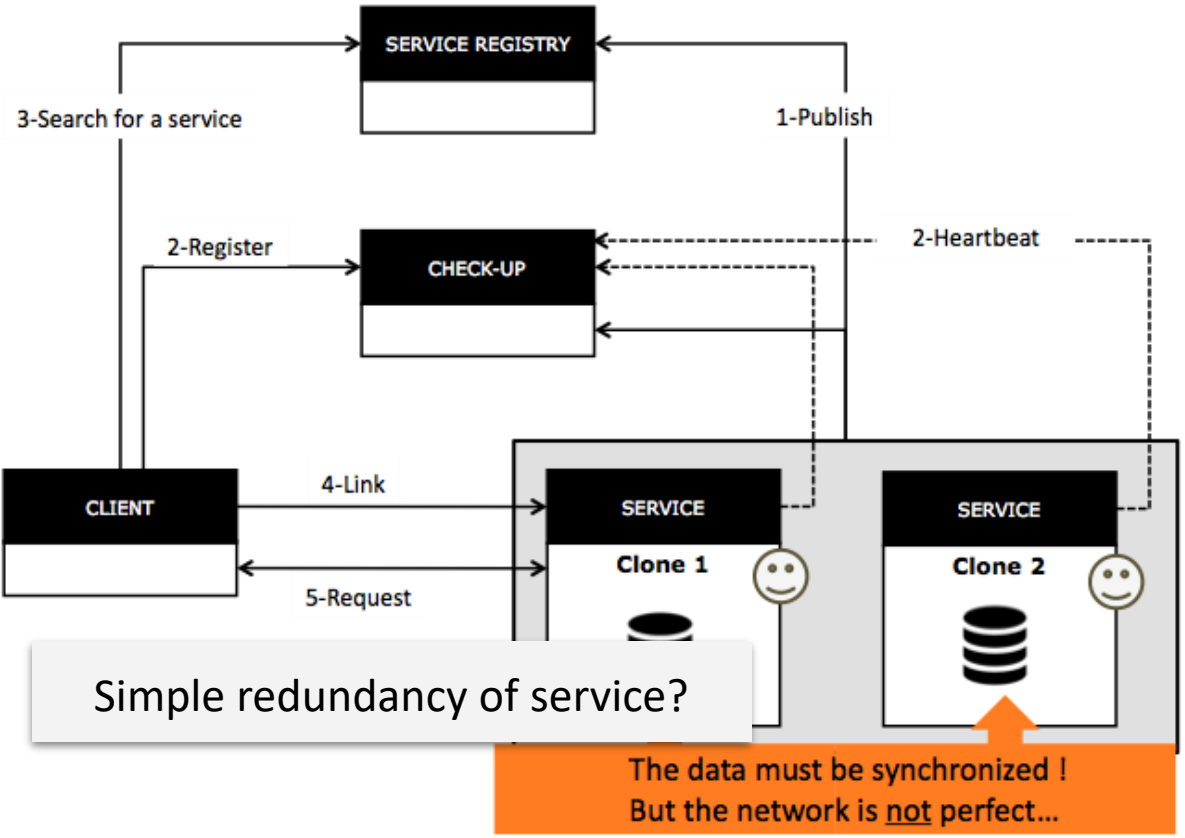
- Pro-active vs. reactive scalability
  - Pro-active = planned in advance
  - Reactive = dynamically adapted to the demand

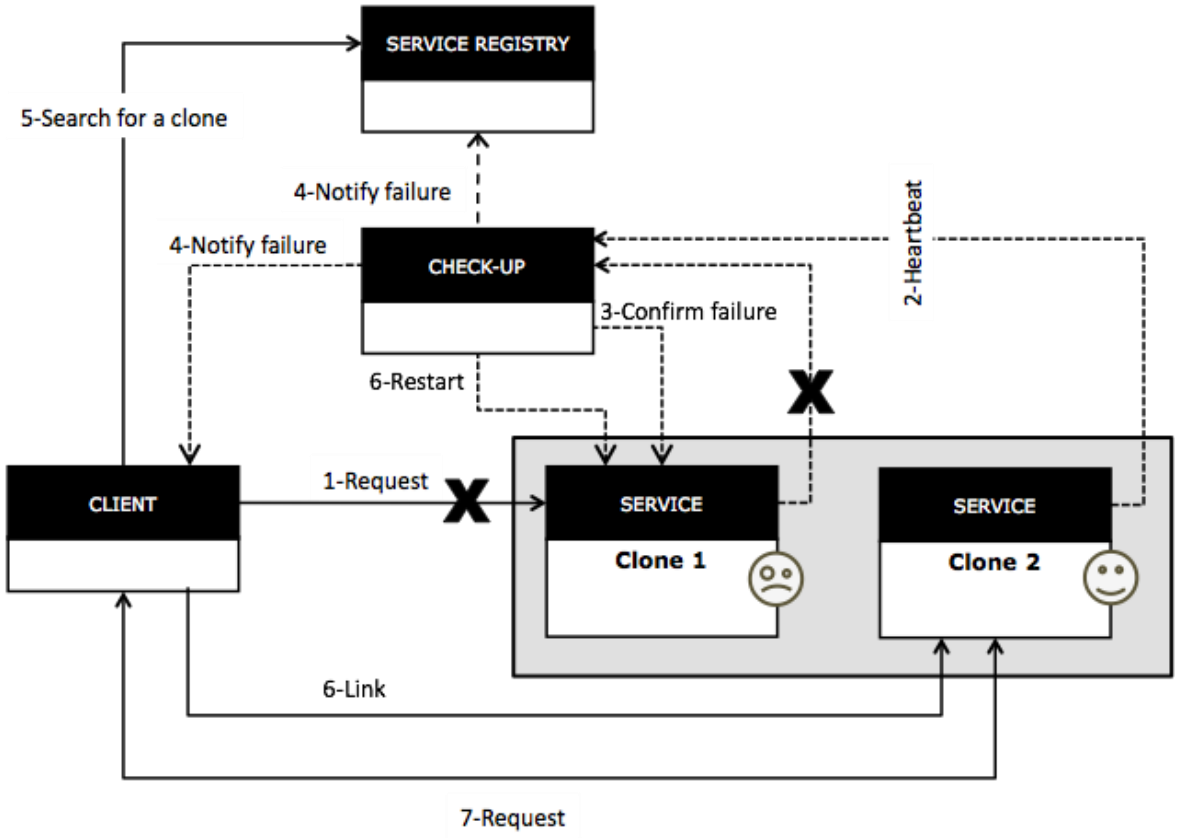


# Fault tolerance



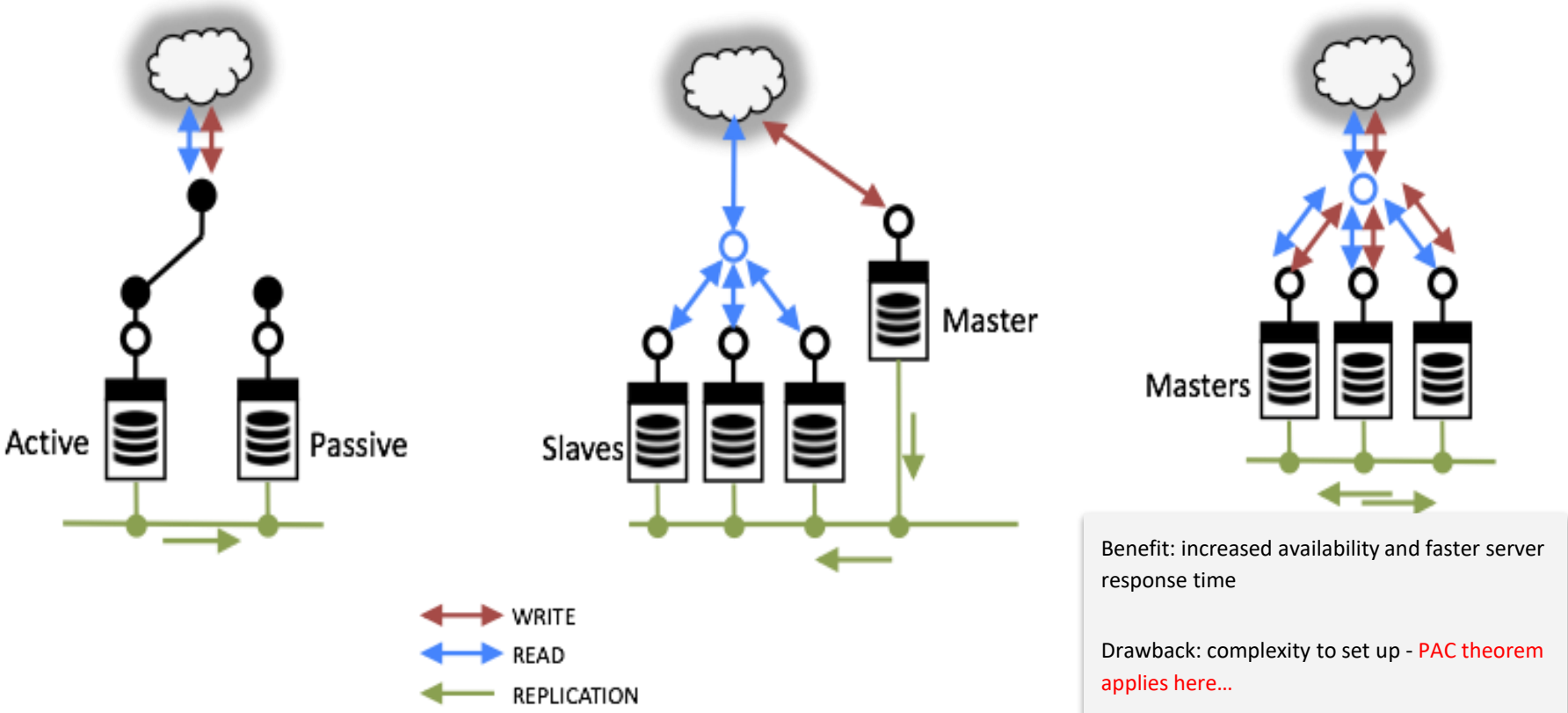
# Robustness


SERVICE REGISTRY
3-Search for a service
1-Publish
2-Register
CHECK-UP
2-Heartbeat
CLIENT
4-Link
5-Request
SERVICE
Clone 1
SERVICE
Clone 2
Simple redundancy of service?
The data must be synchronized !
But the network is not perfect…


SERVICE REGISTRY
5-Search for a clone
4-Notify failure
4-Notify failure
CHECK-UP
2-Heartbeat
3-Confirm failure
6-Restart
CLIENT
1-Request
SERVICE
Clone 1
SERVICE
Clone 2
6-Link
7-Request

# Robustness and Performance

updates are performed on the master and requests are distributed among the slaves



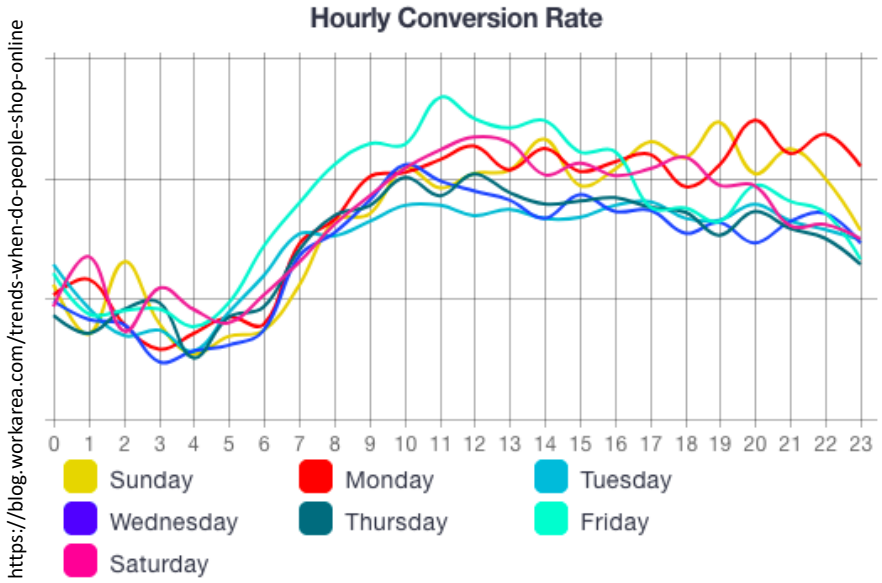

Benefit: increased availability and faster server response time

Drawback: complexity to set up - PAC theorem applies here…

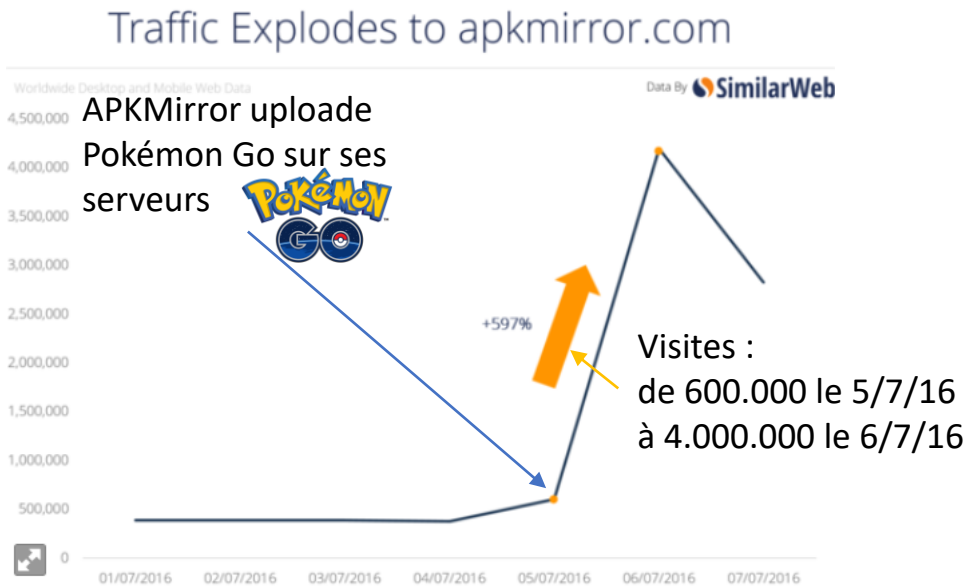
# II. 3.
# Cloud

# Unpredictable or irregular load

- Seasonal activity with foreseeable peaks of activity (daily, monthly, annual...)
- fast growth and/or with little visibility (marketing effects and buzz...)



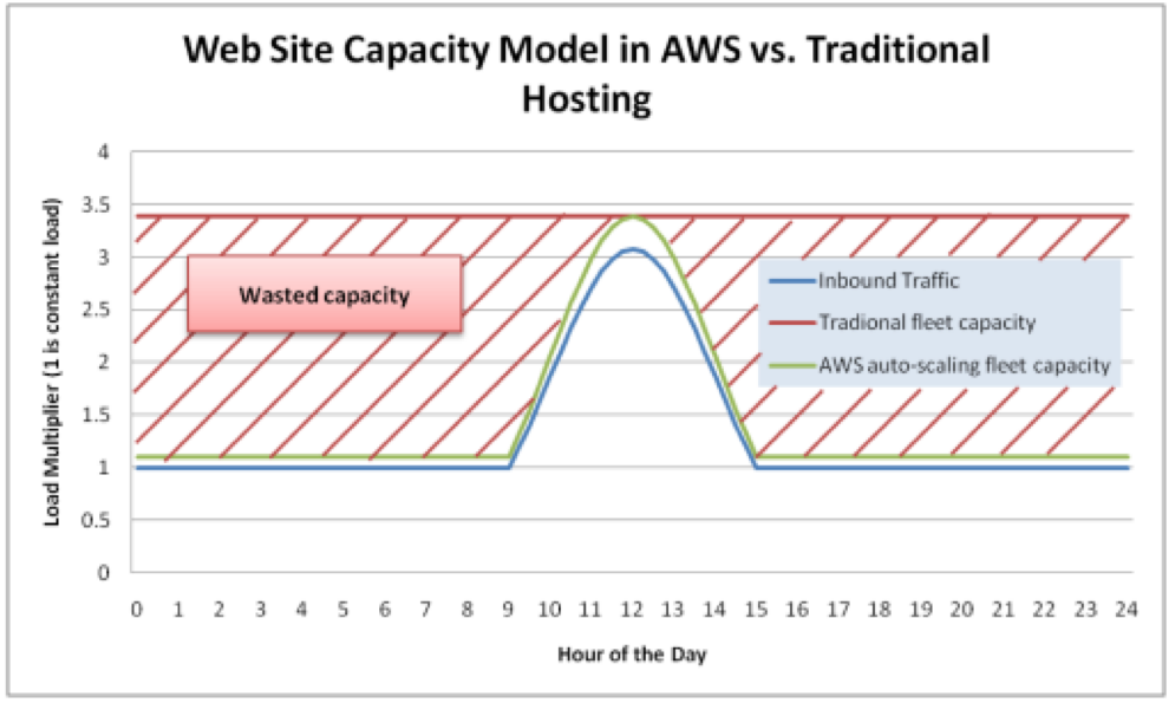

https://blog.workarea.com/trends-when-do-people-shop-online




https://www.pcworld.com/article/3092969/gotta-catch-em-all-5-facts-that-prove-pokmon-go-is-eating-the-world.html

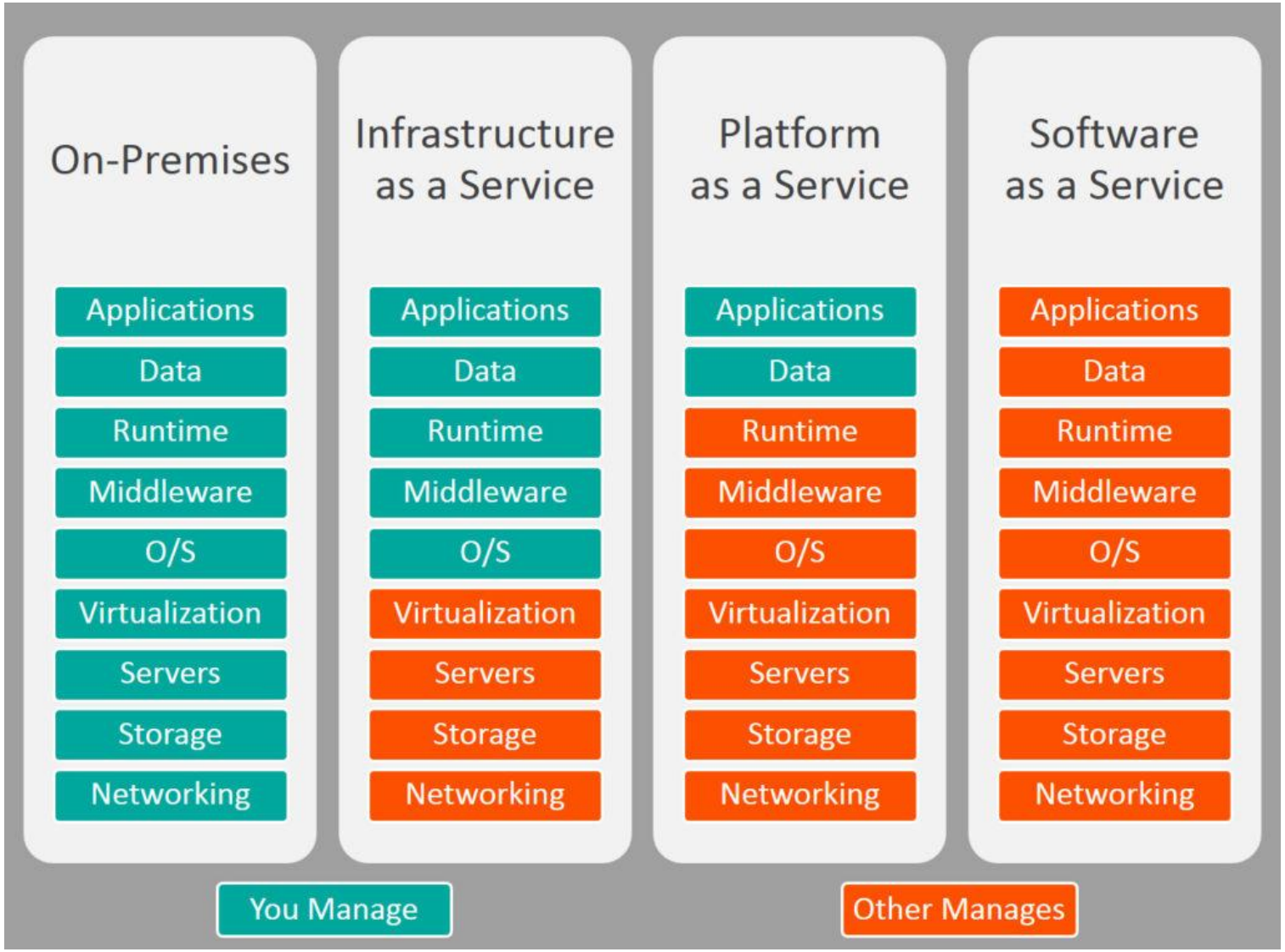
# Tricky resources sizing

- infrastructures must be dimensioned according to the peaks
- How to absorb the load
  - quickly
  - easily
  - with flexibility (possibility of going back to the previous sizing)




https://d1.awsstatic.com/whitepapers/aws-web-hosting-best-practices.pdf

# Solution: rent resources in the cloud




https://blog.dys.com/cloud-iaas-paas-saas/

# IaaS – Infrastructure cloud

- Provides the hardware/physical infrastructure:
  - File servers
  - CPU
  - network
- Main IaaS market players:


**Worldwide IaaS Public Cloud Services Market Share, 2017-2018 (Millions of U.S. Dollars)**

| Company | 2018 Revenue | 2018 Market Share (%) | 2017 Revenue | 2017 Market Share (%) | 2018-2017 Growth (%) |
|---|---|---|---|---|---|
| Amazon | 15,495 | 47.8 | 12,221 | 49.4 | 26.8 |
| Microsoft | 5,038 | 15.5 | 3,130 | 12.7 | 60.9 |
| Alibaba | 2,499 | 7.7 | 1,298 | 5.3 | 92.6 |
| Google | 1,314 | 4.0 | 820 | 3.3 | 60.2 |
| IBM | 577 | 1.8 | 463 | 1.9 | 24.7 |
| Others | 7,519 | 23.2 | 6,768 | 27.4 | 11.1 |
| **Total** | **32,441** | **100.0** | **24,699** | **100.0** | **31.3** |

Source: Gartner (July 2019)



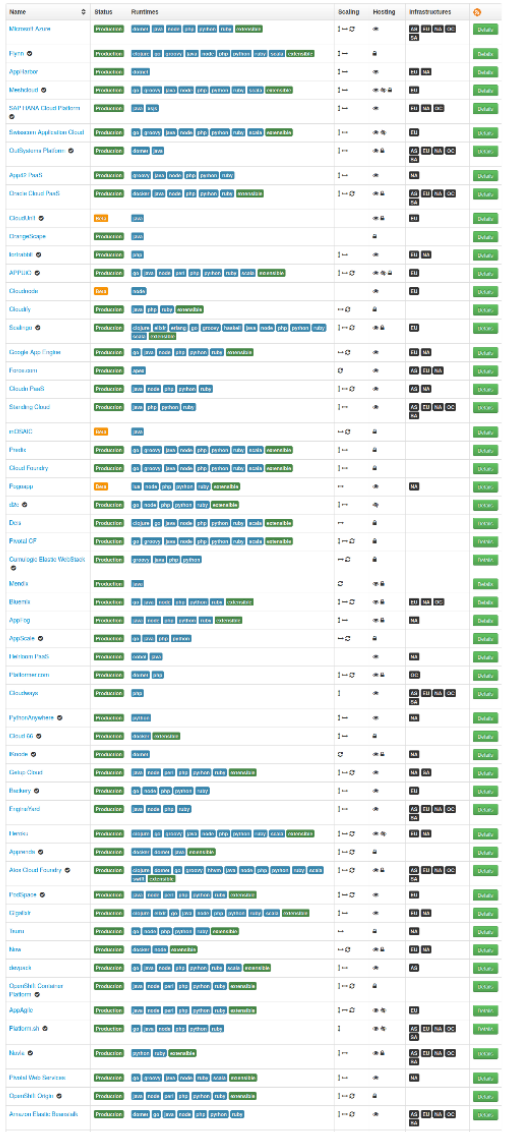
https://www.mactech.com/2019/08/09/gartner-iaas-public-services-market-grew-31-3-in-2018/

# PaaS – Platform cloud


- Hardware infrastructure + OS, middleware and frameworks, RDBMS
- AWS marketplace: thousands of pre-configured VM instances, with or without included licenses

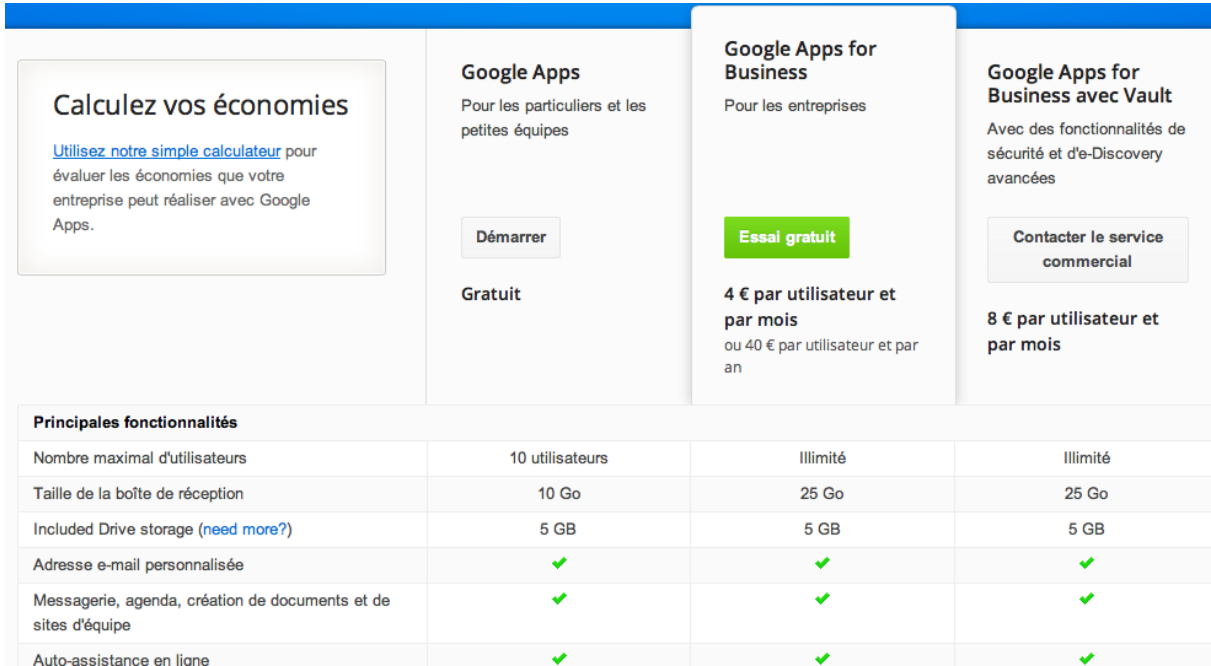
https://paasfinder.org

| Name | Status | Runtimes | Scaling | Hosting | Infrastructures | |
|---|---|---|---|---|---|---|
| Cloudn PaaS | Production | java node php python ruby | | | AS NA | Details |
| Platformer.com | Production | dotnet php | | | OC | Details |
| Pivotal Web Services | Production | go groovy java node ruby scala extensible | | | NA | Details |
| Bluemix | Production | go java node php python ruby extensible | | | EU NA OC | Details |



# SaaS - Application Cloud


- Provides hosted applications, usually accessed from a browser
- Freemium model or subscription
- Examples :
  - Google Apps
    - Gmail, Google Calendar, Google Docs
    - Google Maps
  - Salesforce
    - CRM, Call Center Mgt, Radian 6
  - Microsoft
    - Office 365
  - Adobe
    - Creative Cloud



Calculez vos économies

Utilisez notre simple calculateur pour évaluer les économies que votre entreprise peut réaliser avec Google Apps.

| | Google Apps<br>Pour les particuliers et les petites équipes | Google Apps for Business<br>Pour les entreprises | Google Apps for Business avec Vault<br>Avec des fonctionnalités de sécurité et d'e-Discovery avancées |
|---|---|---|---|
| | Démarrer | Essai gratuit | Contacter le service commercial |
| | Gratuit | 4 € par utilisateur et par mois<br>ou 40 € par utilisateur et par an | 8 € par utilisateur et par mois |
| **Principales fonctionnalités** | | | |
| Nombre maximal d'utilisateurs | 10 utilisateurs | Illimité | Illimité |
| Taille de la boîte de réception | 10 Go | 25 Go | 25 Go |
| Included Drive storage (need more?) | 5 GB | 5 GB | 5 GB |
| Adresse e-mail personnalisée | ✔ | ✔ | ✔ |
| Messagerie, agenda, création de documents et de sites d'équipe | ✔ | ✔ | ✔ |
| Auto-assistance en ligne | ✔ | ✔ | ✔ |

# Typical architecture

**Exterior Firewall**
Hardware or Software Solution to open standard Ports (80, 443)

**Web Load Balancer**
Hardware or Software solution to distribute traffic over web servers

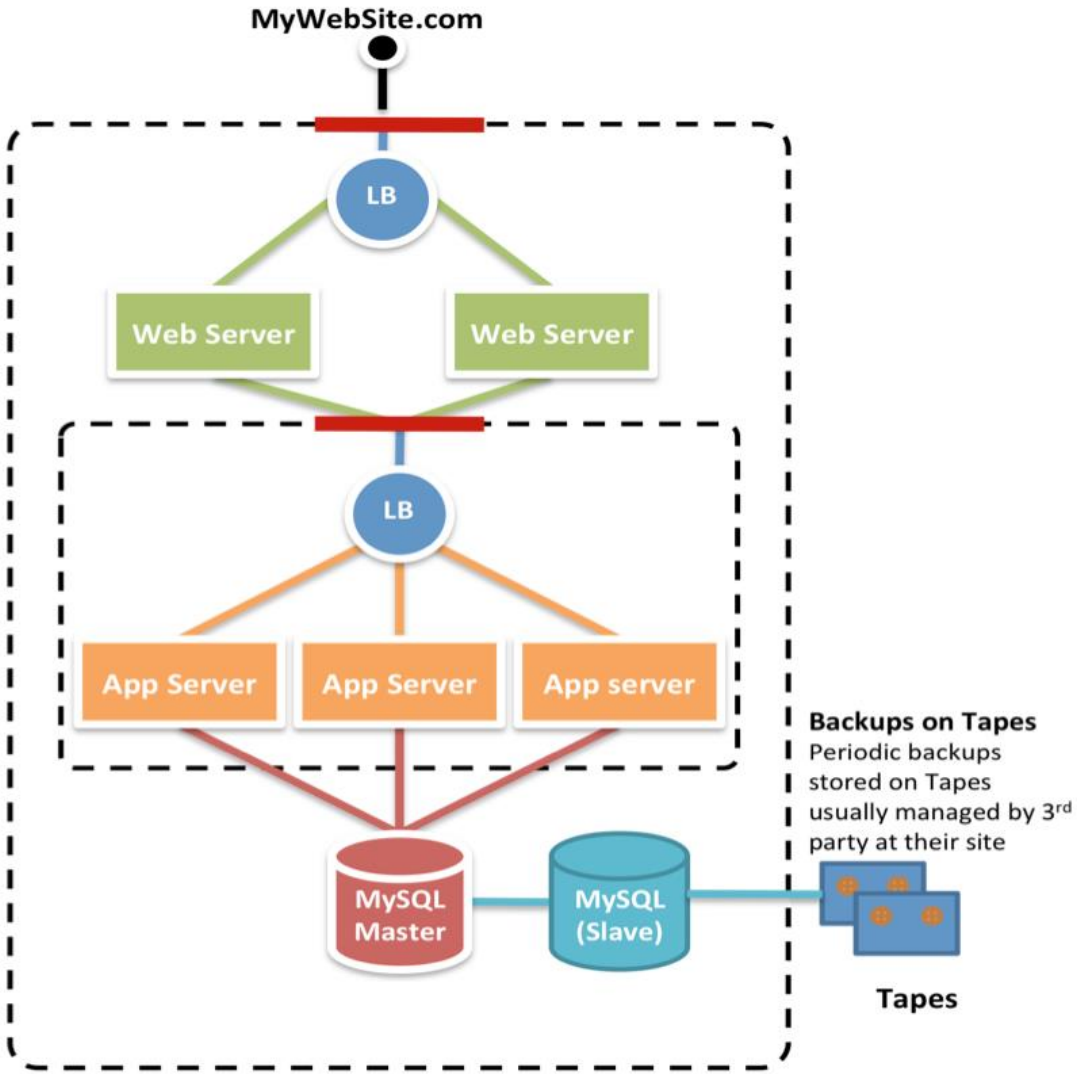
**Web Tier**
Fleet of machines handling HTTP requests.

**Backend Firewall** limits access to application tier from web tier

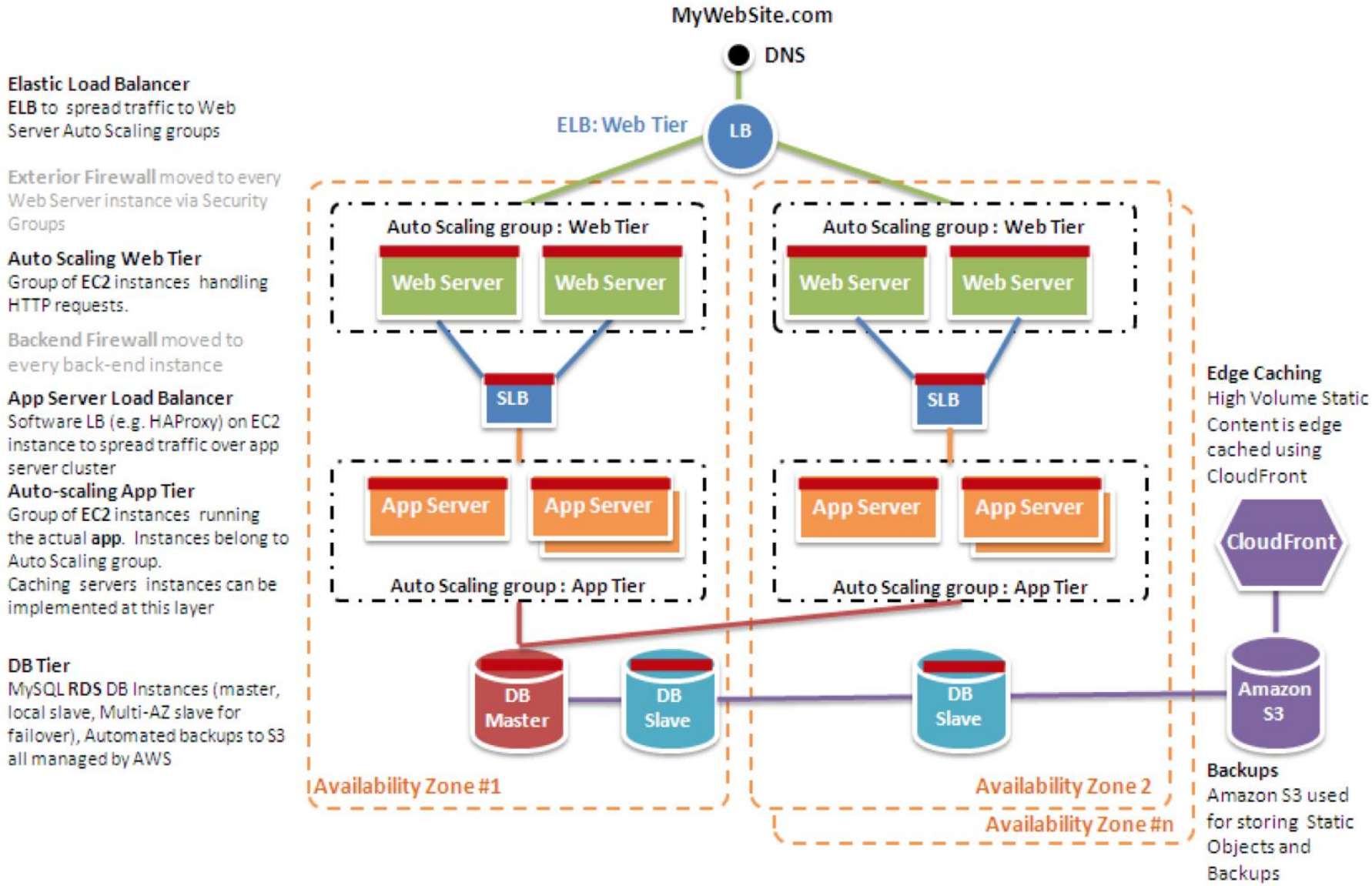
**App Load Balancer**
Hardware or Software solution to spread traffic over app servers

**App Server Tier**
Fleet of machines handling Application specific workloads
Caching server machines can be implemented at this layer

**Data Tier**
Database Server machines with master and local running separately, Network storage for Static objects



# Elastic architecture



# Conclusion… interesting

- An interesting model
  - automated management
  - quick set up
  - virtually no start-up costs
  - elasticity
  - high availability
- Able to respond to needs
  - variable load with peaks
  - rapid growth
  - growth with little visibility
- Offering a reduced adherence between infrastructure and applications

# Conclusion… BUT…

- Not in all situations
  - probably not the most economical for predictable and constant loads
  - not suitable for applications that cannot withstand a start-up delay
- Requires additional skills
  - architecture, development, deployment
  - understanding the offer and APIs
  - real management to ensure financial gain

  → Get support from a Cloud Technology Partner
- Contributes to the entropy of the information system and the complexity of monitoring and debugging
  - possible data exchange between the IS and the cloud via Internet
  - need to secure these exchange flows
  - deal with Internet network latency
  - strive to unify authentication of users accessing internal and external IS services
  - etc.
- And other concerns
  - Golden handcuffs syndrome and all the eggs in the same basket

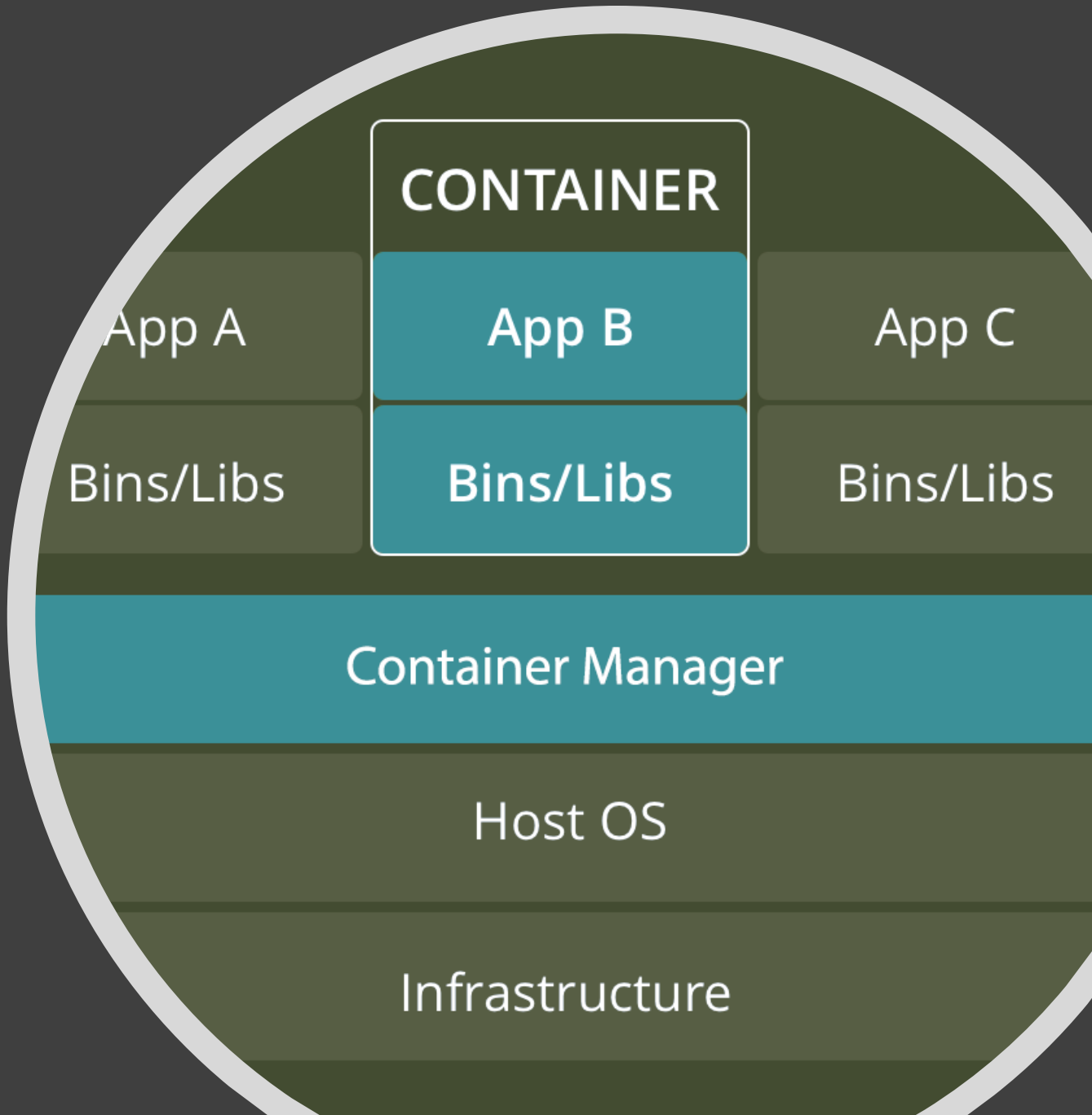
    Not-so-reduced adherence, variable portability, what if the cloud service provider goes bankrupt or makes a major change to its product?
  - Reliability and service commitment

    SLA at 99.5% but several major breakdowns
  - Data security

    Can you trust a third party to preserve critical and/or confidential data?
  - Legal impacts of data location

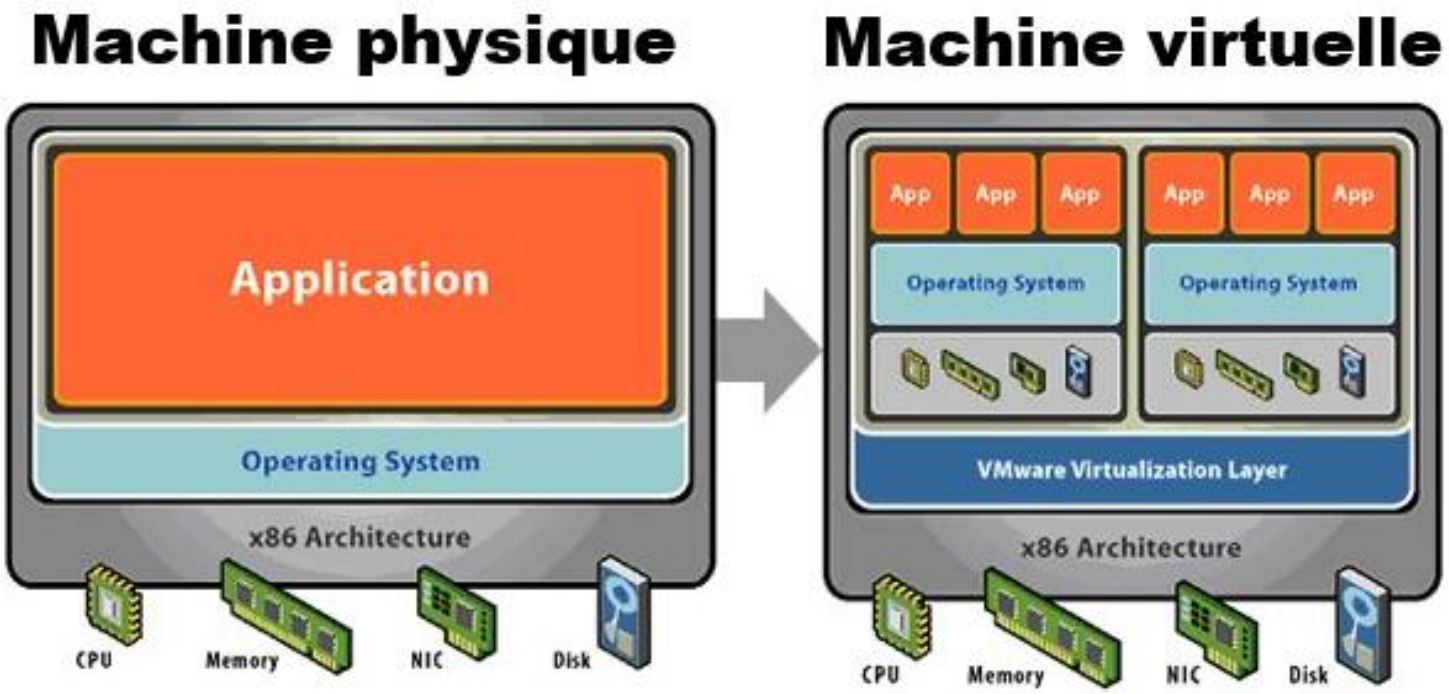
    The Cloud Act set the cats among the pigeons

# II. 4.
# Virtual Machines
# Containers



# Virtualization

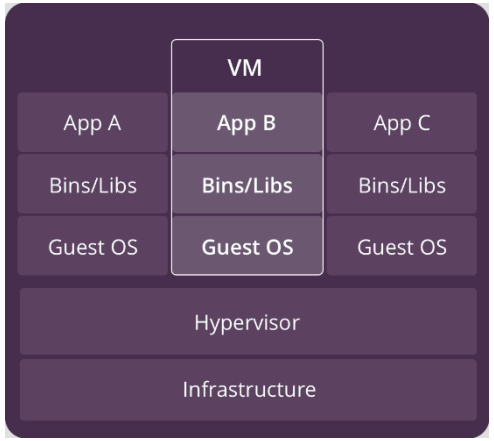
- Good for virtualizing the infrastructure
- But burdensome to follow the evolutions of an application

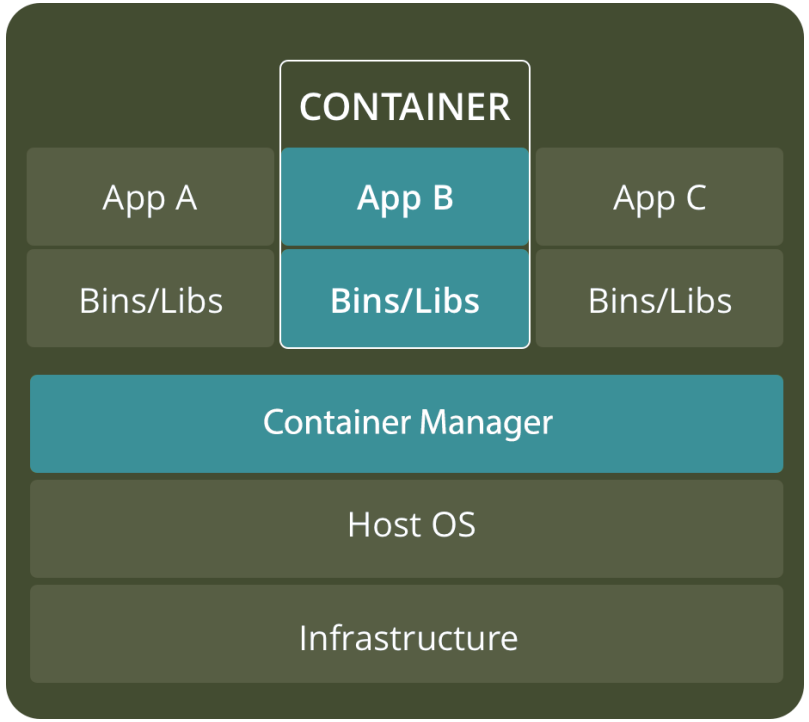



http://www.it-ns.com/virtualisation-et-storage.php

# Container

- Lighter
- (Nearly) no start time
- Native performances (no hypervisor overhead)



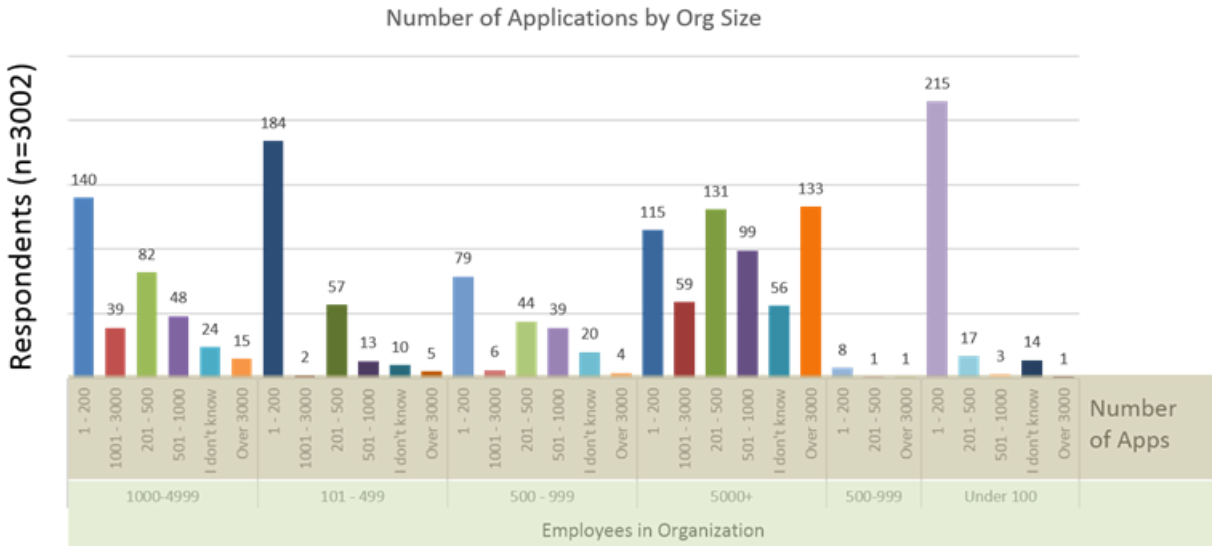




https://www.backblaze.com/blog/vm-vs-containers/

# III. Application Integration

# The inevitable disparity of applications

- Each organization is unique, universal software is a utopia.
- Willingness to take advantage of the best application on the market for a given task (best-of-breed)
- Willingness to limit dependence on a single technology or publisher
- IS merger following a merger/acquisition
- …




https://www.f5.com/company/blog/applications-applications-everywhere

# What is heterogenic?

- Underlying operating system
- Programming language
- Database system
- Data format (structure, type, file format...)
- Versions of libraries
- Trends and History of Computer Science

# Issues

- Operational: manage several applications (installation, with variable dependencies, updates, etc.)
- Architectural: each application has its own specialty but often cannot work in isolation, it needs input data, some of which is generated by other applications (and it is potentially a source of data for a downstream application in the business process).

manual propagation of information? Bad solution (time, errors...)

→ need an automated solution!

# SOA also requires integration solutions

- SOA → application is built from services
  - Find the services
  - Adapt parameter data to service and send input to service
  - Wait for service responses, transform response to another format
  - etc

# Application Integration

- Processes that allow independently designed applications (or services) to work together
- Merging of data and processing between two disparate software applications
- Software solution, architectural solution and governance solution

# To expose an interface is not enough

- Hypothesis: each application exposes an API (web or not) allowing to interact with it from the outside
- Coupling between applications:
  - Ad-hoc
    - Manual programming of point-to-point interactions between two applications
    - Difficult to maintain and extend (adding an application to a group of *n*: *n* new couplings to be done)
  - Use of an application integration platform

# Main components of application integration platforms

- Communication system for reliable transfer of messages/data between endpoints
- Synchronous and asynchronous call mechanisms
- Connectors to/from mainframe, ERP and other systems
- Dynamic linking system of supplier and customer endpoints
- Data conversion mechanism
- Message transformation, enrichment, mapping and validation mechanisms
- Orchestration
- Support for multiple interaction patterns
- Transaction management, security
- Administration: services directory, deployers, discovery protocol
- Supervision: monitoring console, alerts
- …

# Application Integration Solutions / Ecosystem

Integration is a jungle world!

A few basics on which the majority of solutions are based :

- Local: IPC
- Synchronous solution: RPC and ORB
- Asynchronous: messages
- Orchestration


# III. 1.
# IPC

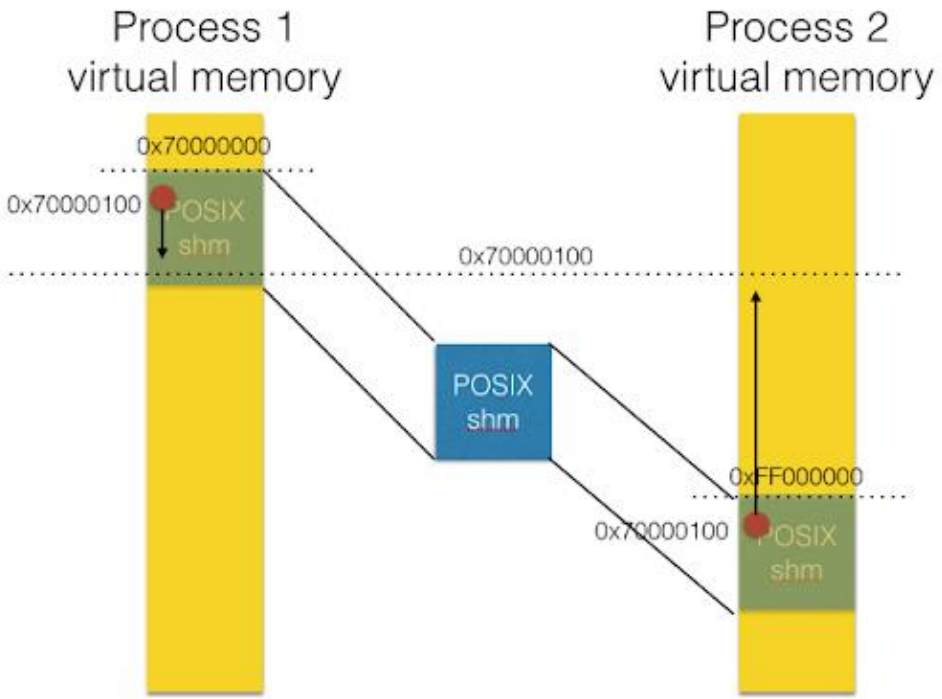
# Inter Process Communication (IPC)

**OS-provided** mechanism allowing two concurrent processes

- to synchronize their actions
- and/or to exchange data

# Shared memory

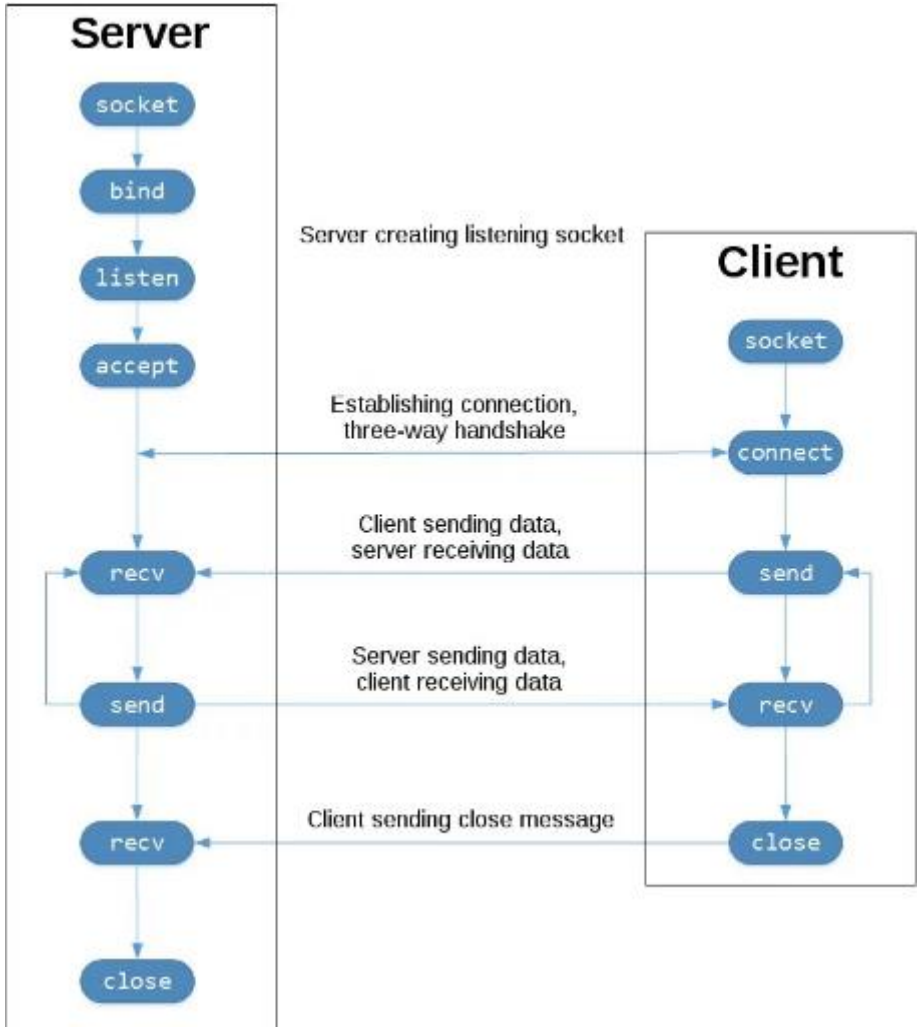
- Multiple processes can access the same memory block.




- Can be used for
  - Memory-mapped files shared between processes
  - Shared in-memory message queue

# Network Socket

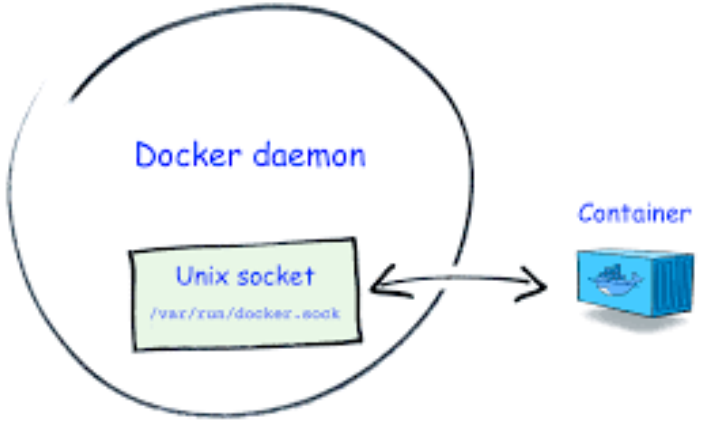
sends data through network interface




https://realpython.com/python-sockets/

# Unix Domain Socket

- sends data (or a file descriptor) to a process on the same machine (communication takes place in the OS kernel rather than via a network protocol)
- 2 interfaces :
  - Stream (comparable to TCP)
  - Datagram (comparable to UDP, but reliable and orderly)




https://medium.com/better-programming/about-var-run-docker-sock-3bfd276e12fd

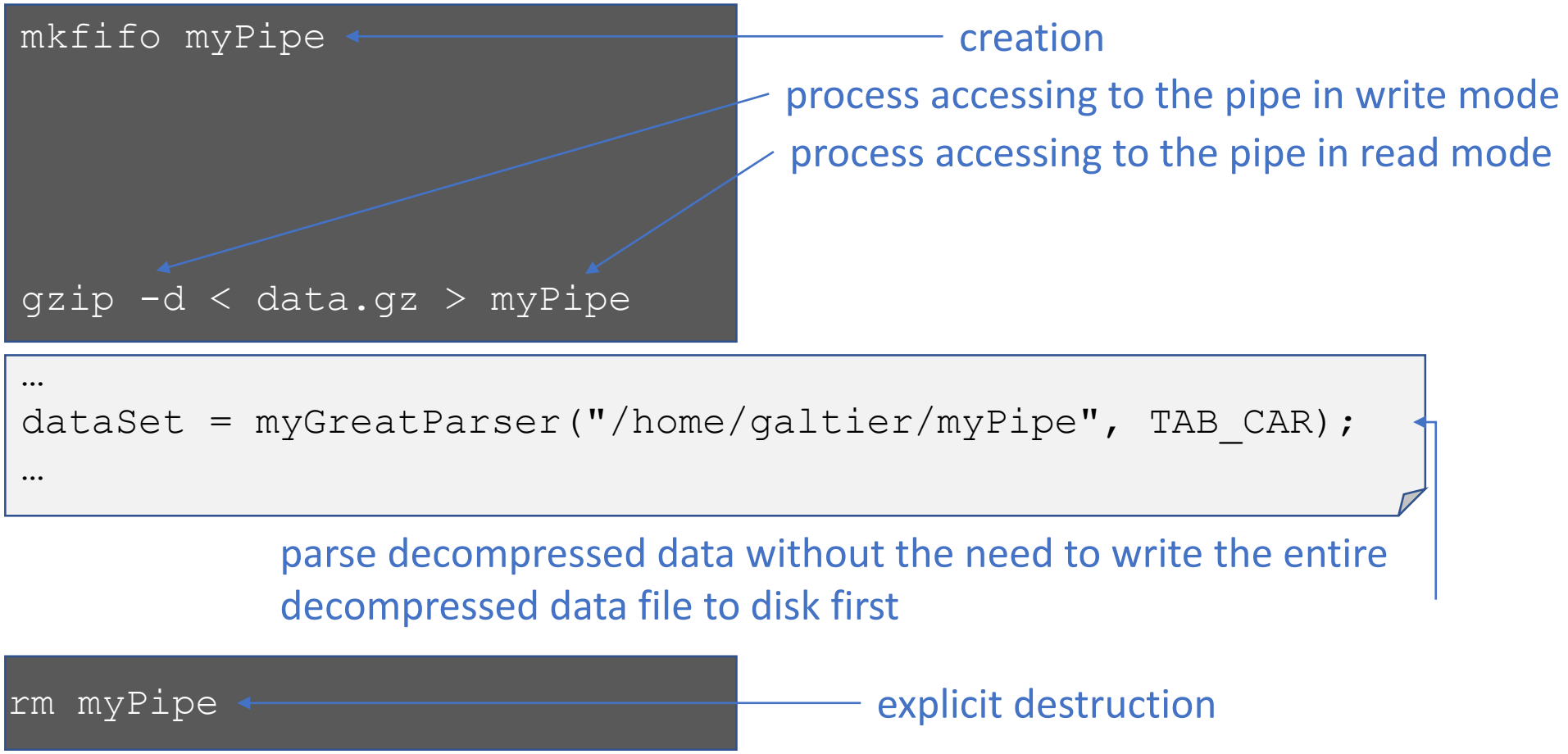
# (Anonymous) Pipe

redirects the standard output of one program to the standard input of another program

```
cat myfile | wc
```

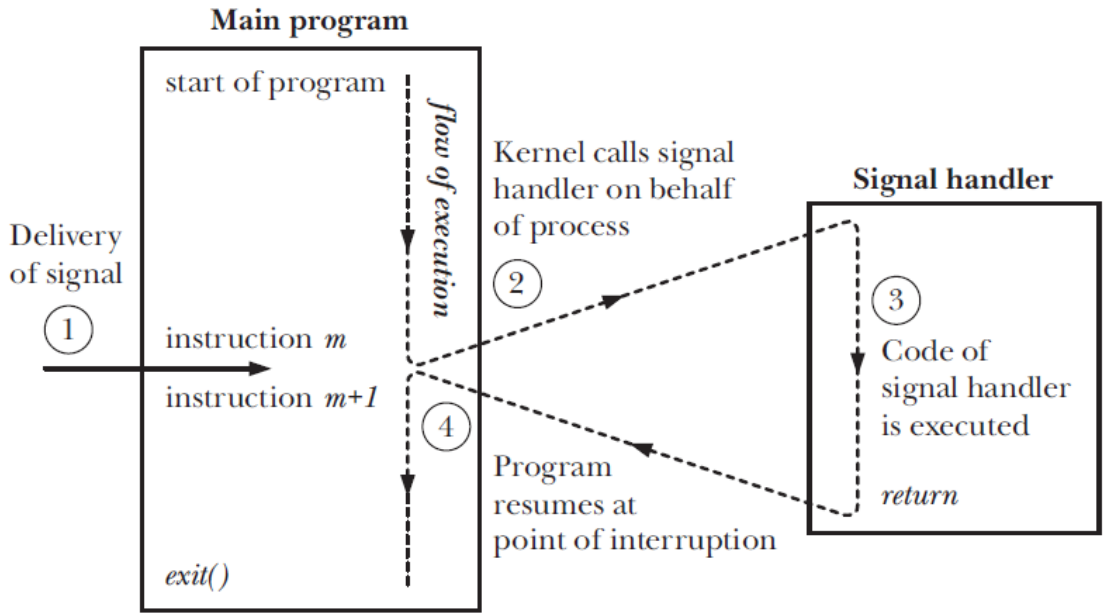
# Named pipe

persistent, appears as a file but does not need to be written to disk



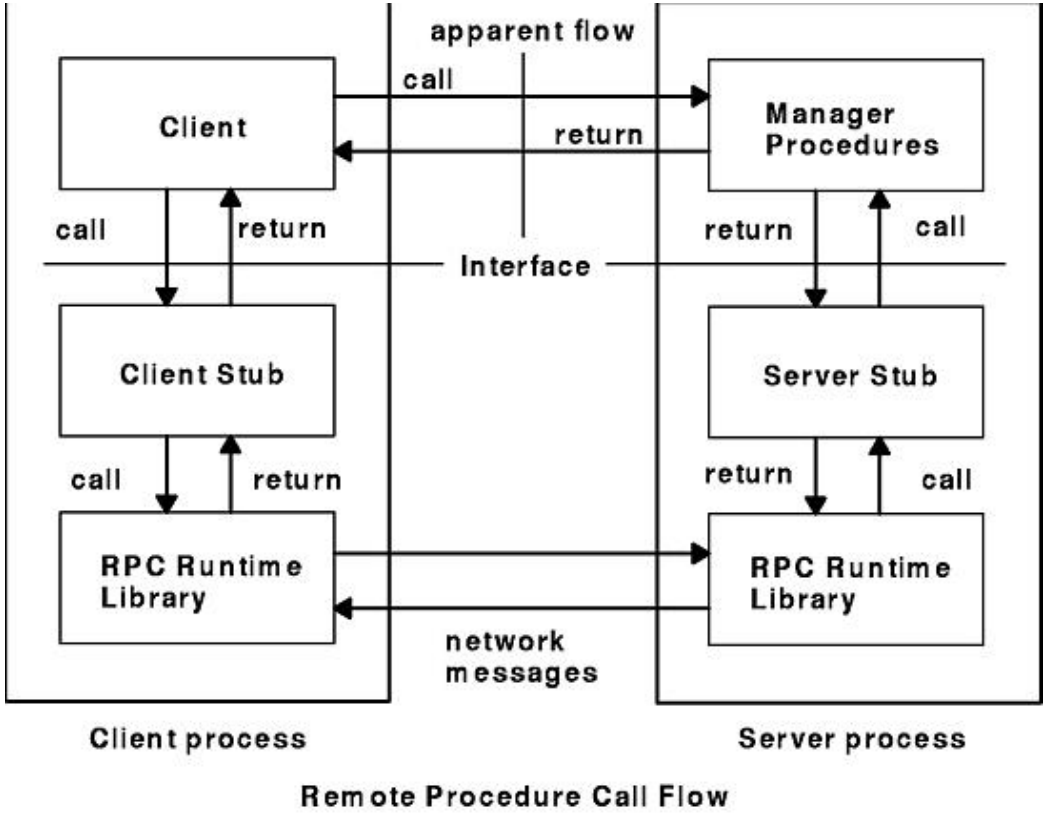
# Interrupt signal

asynchronous notification sent to a process to notify it of an event; if the notification is not “masked”, the OS interrupts the execution of the process and starts its processing routine




https://stackoverflow.com/questions/53367146/when-is-a-signal-handled-and-why-does-some-info-freeze-up

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

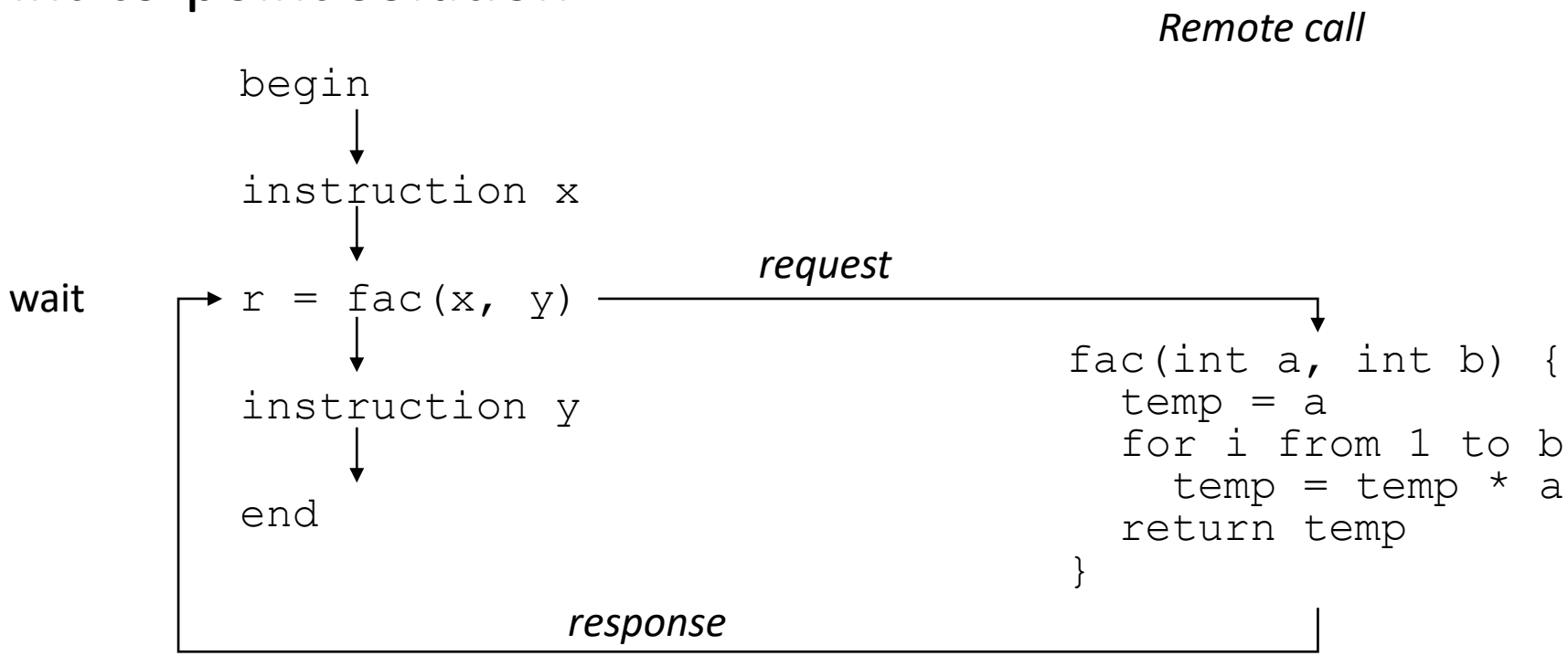



Remote Procedure Call Flow

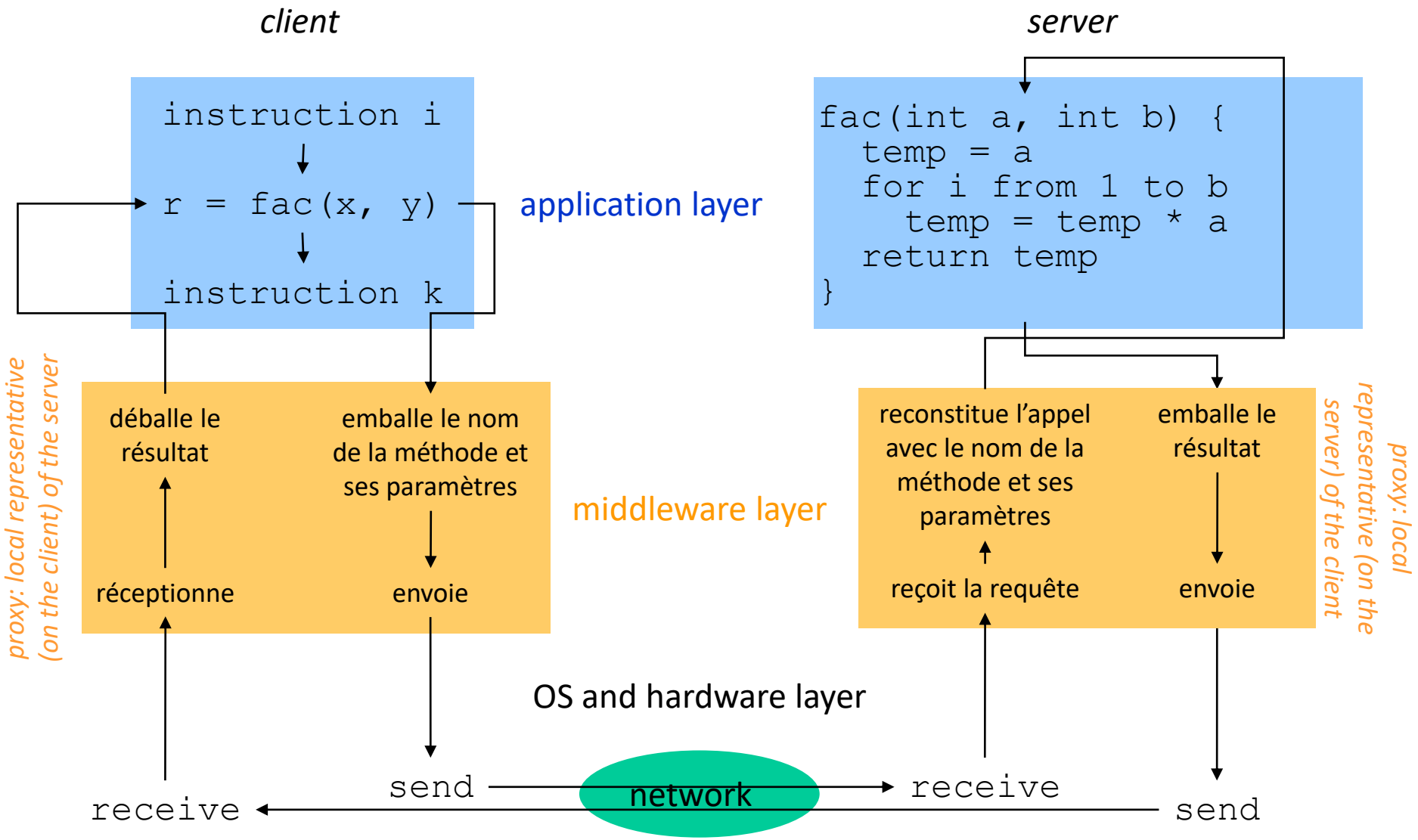
# III. 2. a. RPC

# RPC Remote Procedure Call

- Oldest solution (70's)
- Client/server architecture
- Call a remote service synchronously (mostly)
  (almost) as if it were local
- Point-to-point solution



# Key elements: proxies



# transparency

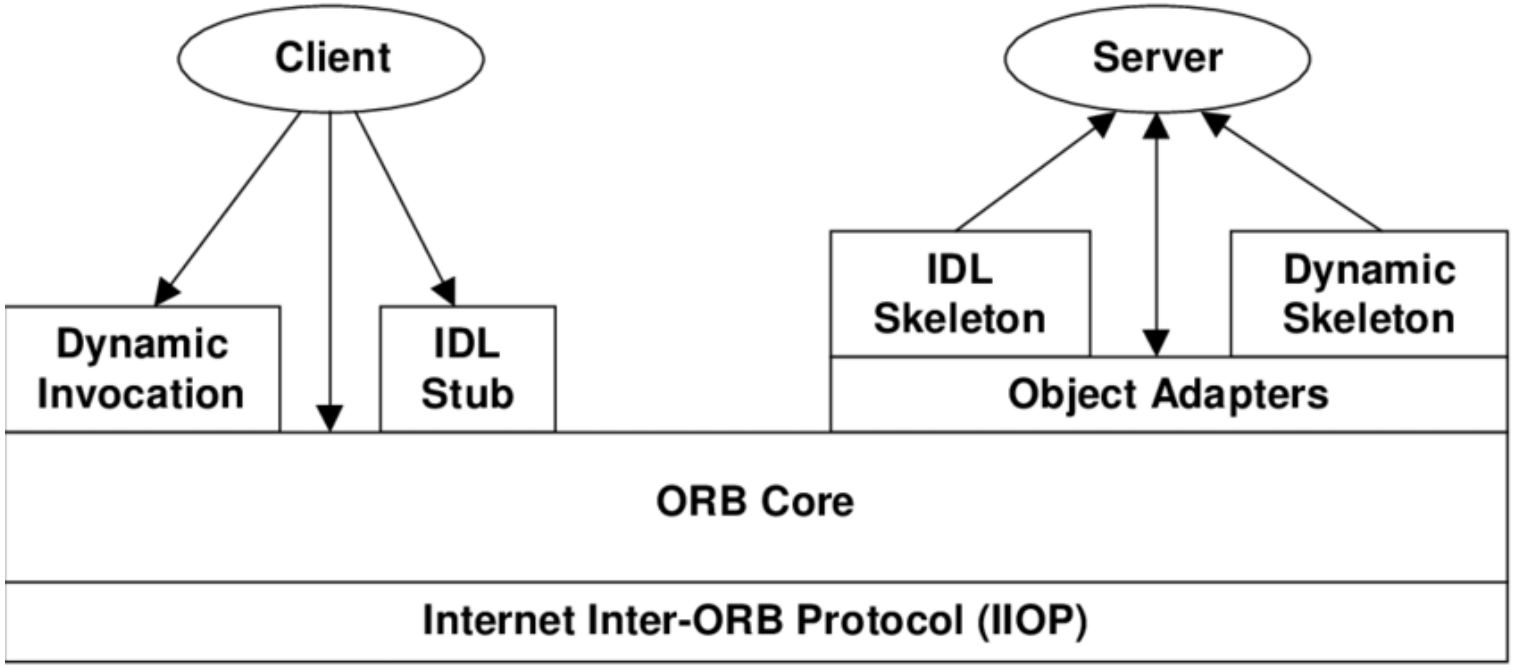
Proxies take care of:

- network operations
- transformations between client and server formats

→ loose client / server coupling

# Main players of RPC middleware

- early 90s, dominant until 1995:
  - DCE RPC (Apollo/HP then Open Group)
  - Sun RPC (used by NFS)
- 1998: XML RPC (led to SOAP)


Client
Server
Dynamic Invocation
IDL Stub
IDL Skeleton
Dynamic Skeleton
Object Adapters
ORB Core
Internet Inter-ORB Protocol (IIOP)

# III. 2. b.
# ORB

# Object Request Broker

RPC for Object-Oriented: method calls on remote objects

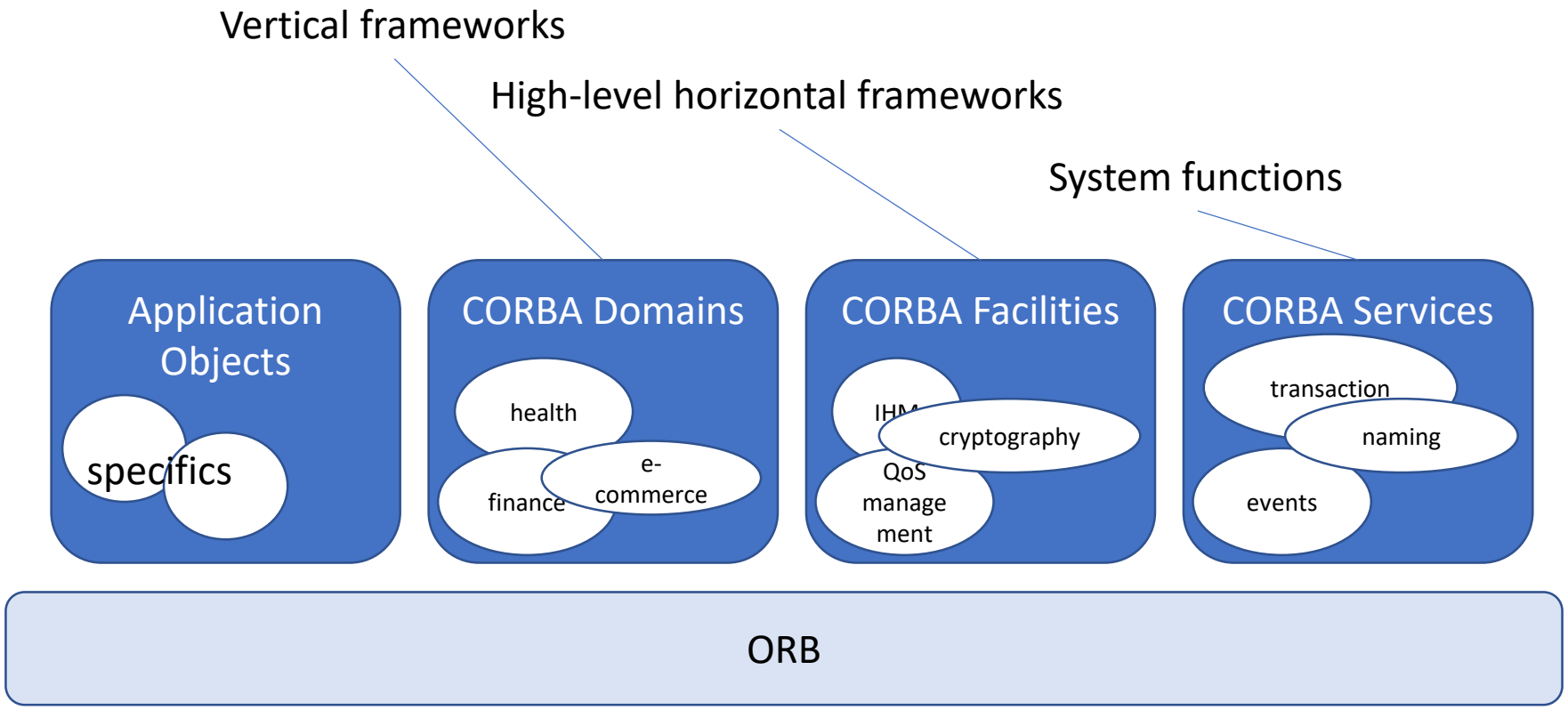
- Heterogeneous objects:
  - CORBA (Common Object Request Broker Architecture) (1991)
  - DCOM (Distributed Component Object Model) (1995), .Net Remoting : Microsoft equivalent of CORBA, possible bridge with CORBA (COM objects can appear as CORBA objects)
- Java objects: Java RMI (Remote Method Invocation) (1998)

# Object Management Architecture

CORBA: “OO-RPC”

but also

a **set of services**:



# Limitations

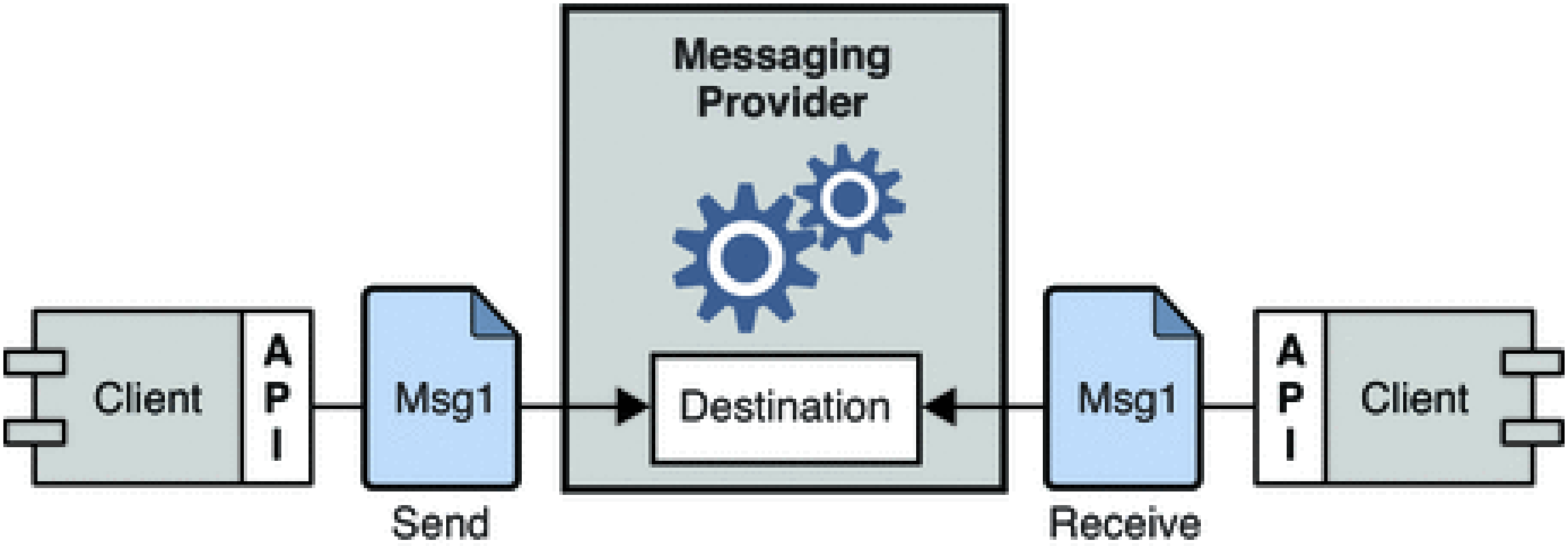

- Local calls are treated exactly the same as remote calls → inefficient
- Standard designed without governance between proposals → complex, ambiguous
- Difficult to make different versions of a service co-exist
- Not easy to learn, fewer and fewer experts
- In the past:
  - Incompatibility between first implementations
  - Failed or partial implementations
  - Difficulties getting through firewalls

# Perspectives


- Legacy still important (especially as one of the first)
  3G telephony base stations, air traffic control, Hubble telescope...
- One of the few candidates (with DDS) when there are hard real-time constraints
- But few new developments
  - Java.corba module deprecated since Java SE 9
  - Little taught

# RMI vs. CORBA

- All java (go through JNI if not)
- Part of the JDK and JRE
- Easier to implement
- Few services
- Depending on the scenario, slower than CORBA

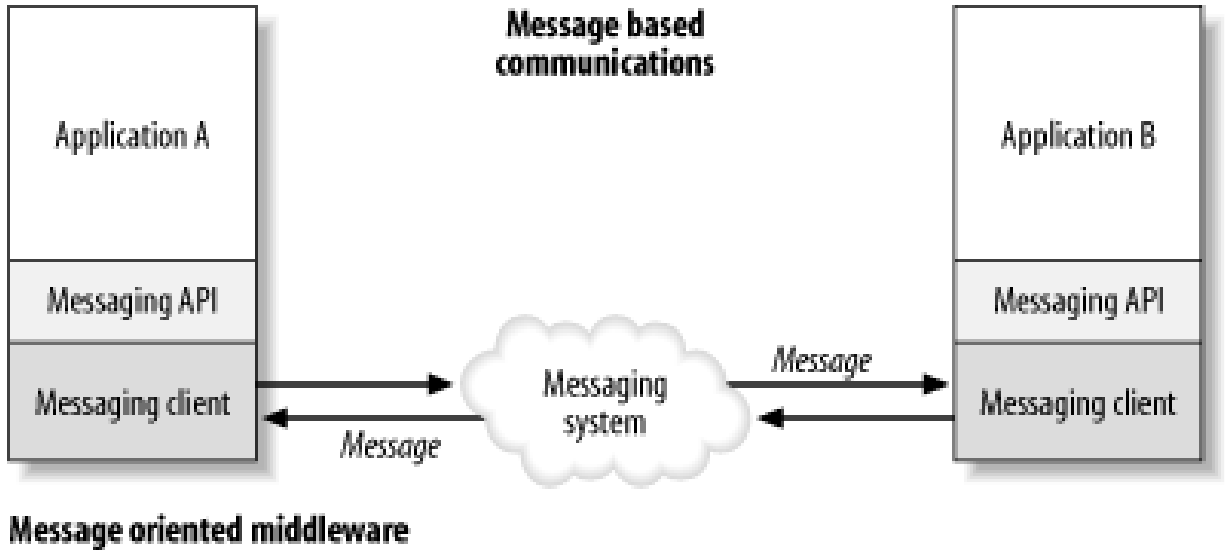

Messaging
Provider
Client
API
Msg1
Send
Destination
Msg1
Receive
API
Client

# III. 3.
# MOM

# Message Oriented Middleware

- Principle :

  communication of data by messages written into a queue

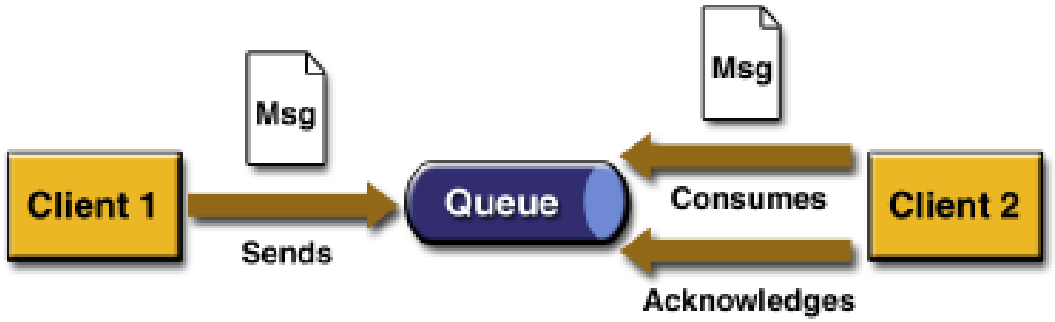


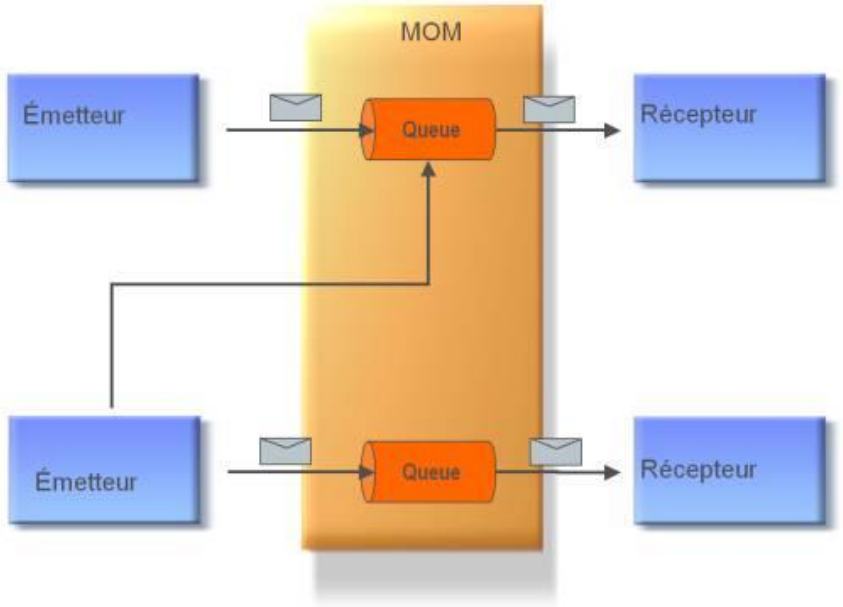

- Characteristics:
  - Asynchronous: transmitter and receiver do not need to be active at the same time
  - Reliable: messages cannot be lost
  - Causal: the messages are delivered in the order they were written

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

# Point-to-point mode

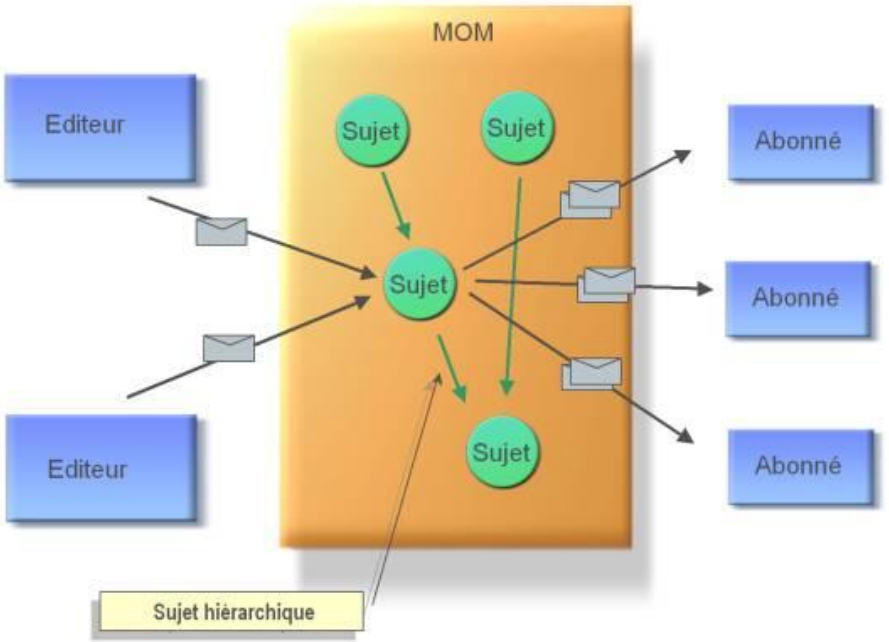
- one producer – one consumer
- read operation removes the message (analogy: physical mailbox)



# Publish/subscribe mode

- Definition of a "topic”
- Consumer subscribes to a topic
- Message is maintained until all subscribers have read the message
- analogy: bulletin board

# Optional Services

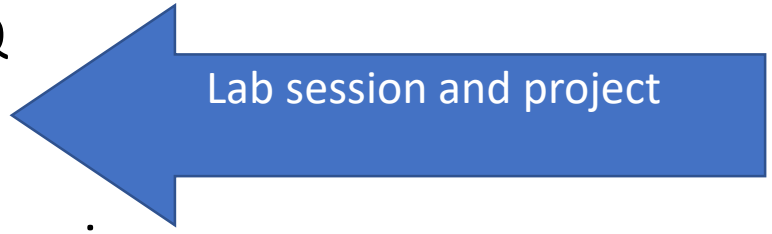
- message filtering
- safety / security functions
- hierarchical organization of messages
- messages retention for offline consumers
- priority between messages
- expiration or validity date on messages
- notification
- compression/decompression functions
- functions for transforming messages from the sender's native format into one or more other formats
- persistent file (on physical support)
- reliability (Ack from MOM to transmitter and Ack from receiver to MOM)
- guarantee of delivery: at least once or exactly once
- guaranteed delivery of messages in the right order
- transactions
- ring or blocking queue management
- …

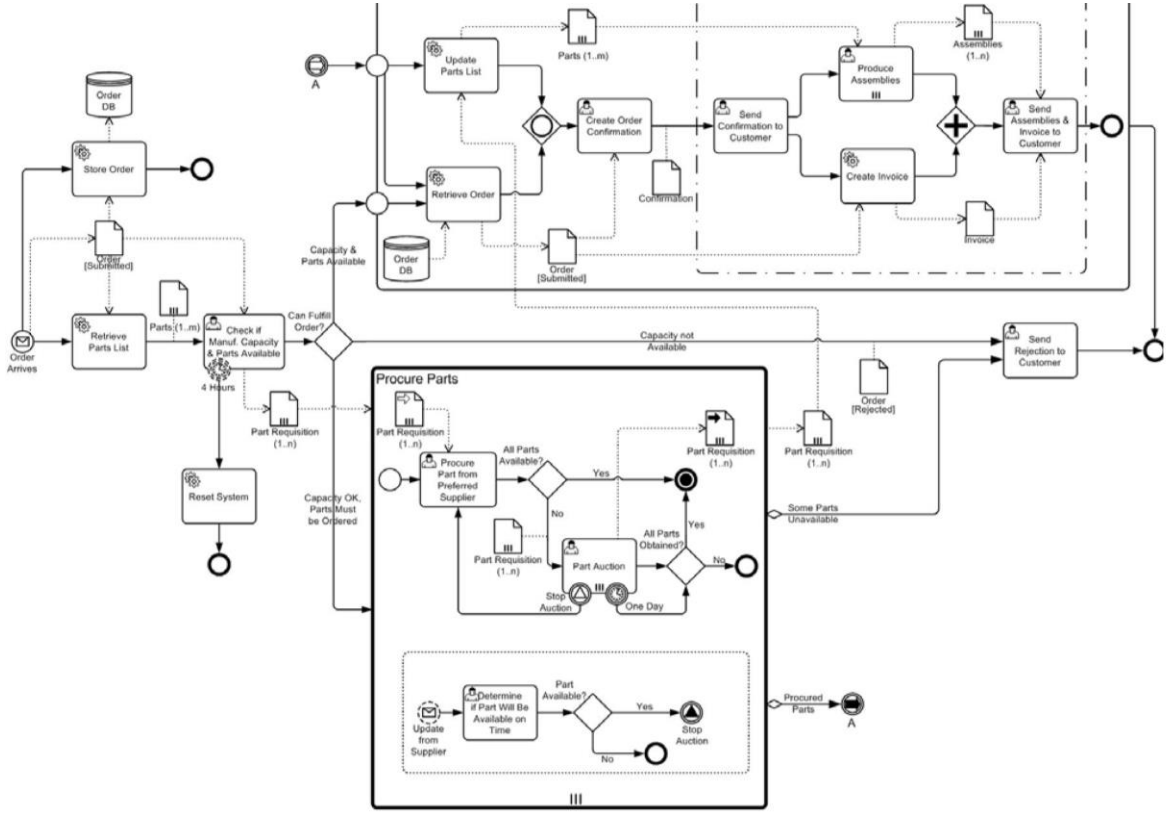
# Standards

- Advanced Message Queuing Protocol (AMQP): OASIS and ISO, quite exhaustive
- MQ Telemetry Transport (MQTT): OASIS and ISO, lightweight publish/subscribe protocol over TCP
- Simple/Streaming Text Oriented Message Protocol (STOMP): similar to HTTP, writing a client is really easy
- Data Distribution Service (DDS): OMG : real-time, large-scale, and FT publish/subscribe
- HLA: simulation interoperability (military)
- eXtensible Messaging and Presence Protocol (XMPP): IETF
- API Java Message Service (JMS): Java standard

# Examples of MOM

- Open-source:
  - ActiveMQ
  - RabbitMQ
  - **Kafka**
  - ZeroMQ
  - Jboss Messaging
  - Joram
  - …
- Proprietary:
  - IBM WebSphere MQ/Series
  - Tibco RendezVous
  - Microsoft Message Queuing (MSMQ)
  - Amazon SQS (Simple Queue Service)
  - …

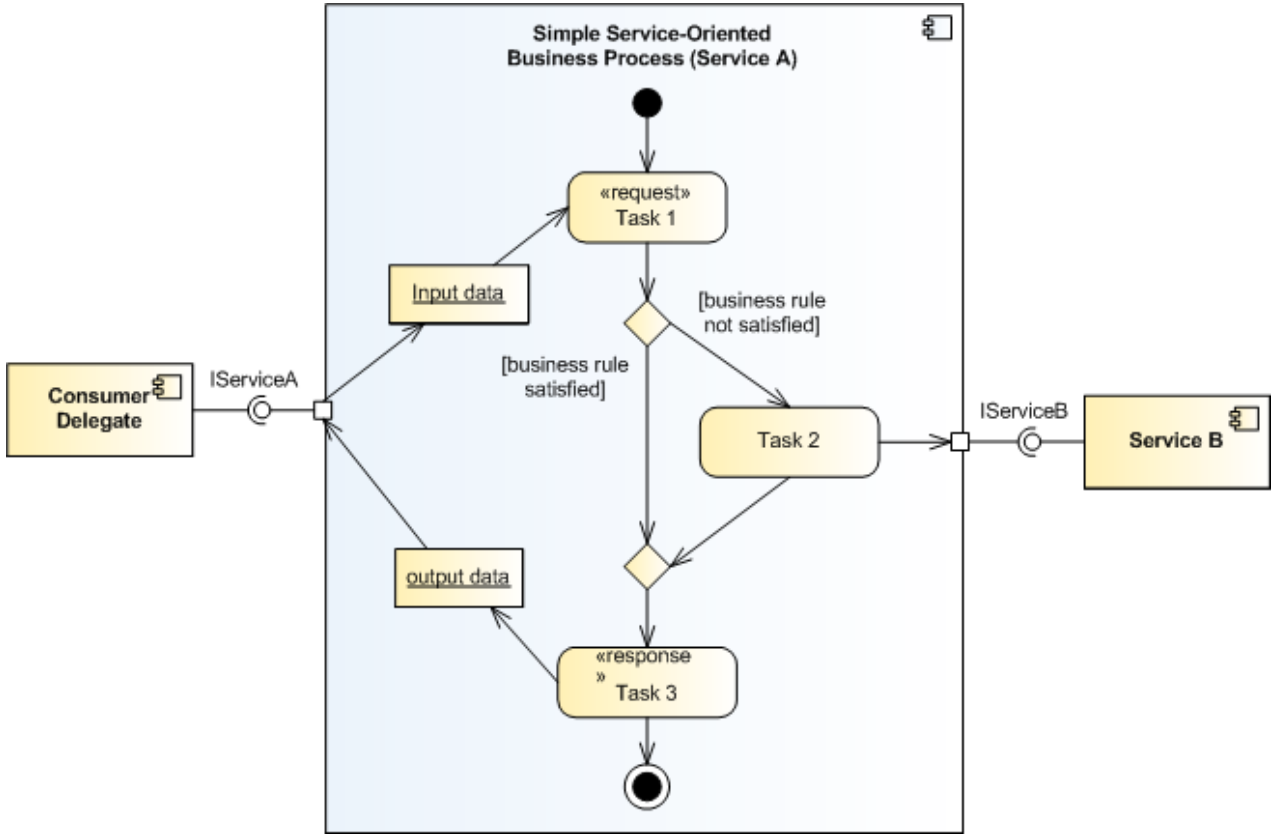




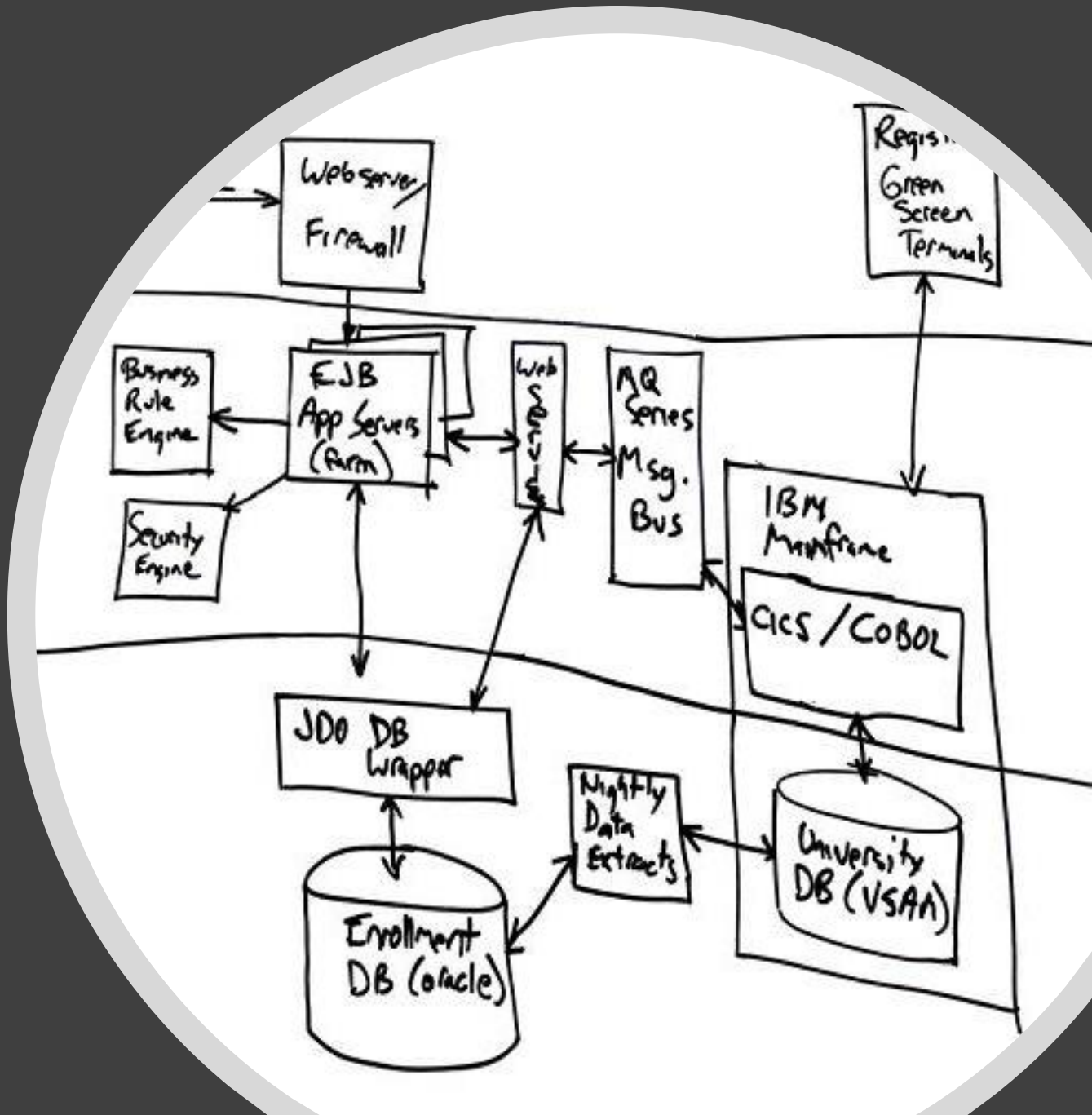

# III. 4. orchestration

# Orchestration

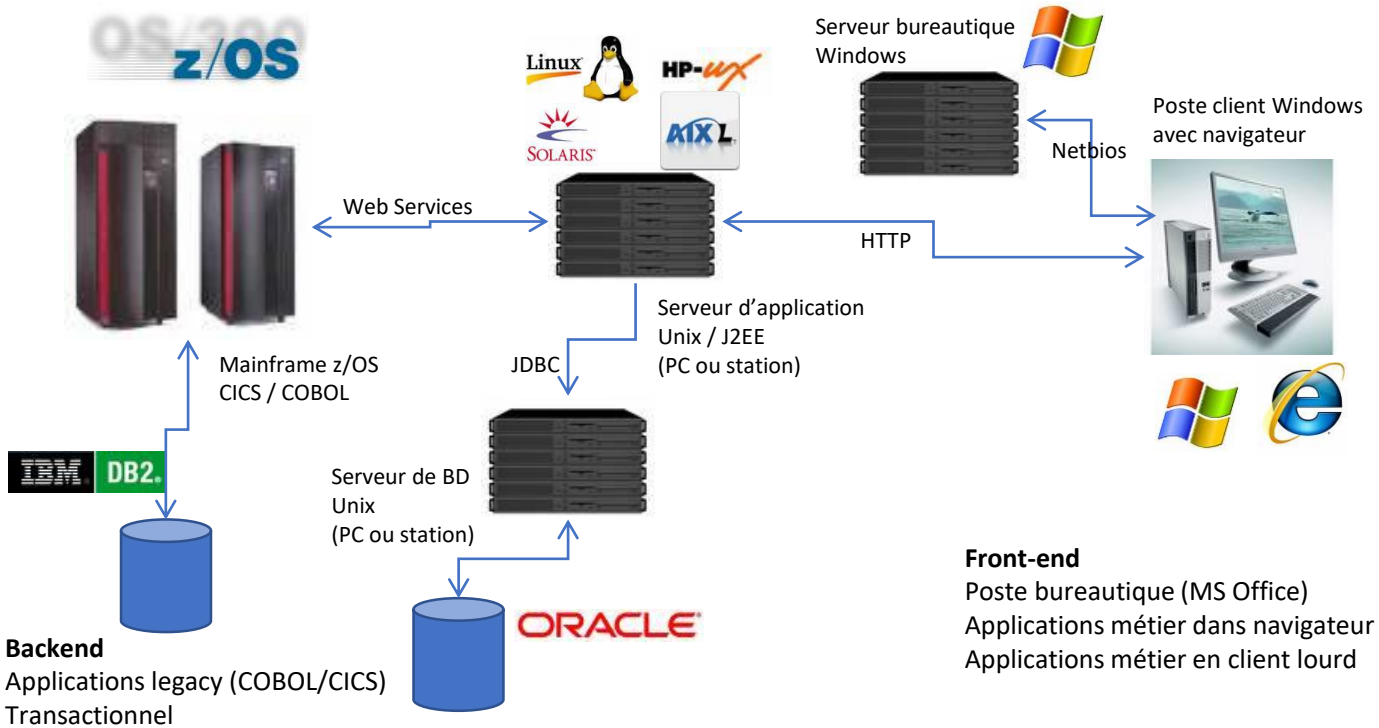
- Example: BPEL (Business Process Execution Language)
- High level programming language for orchestrating web services
  - Loops, switch, if-then-else, repeatUntil...
  - Invocation: invoke, receive, reply, wait
  - Sequential or parallel invocation of services
  - Possibility to use code snippets expressed in classical languages
- + runtime engine interpreting the BPEL



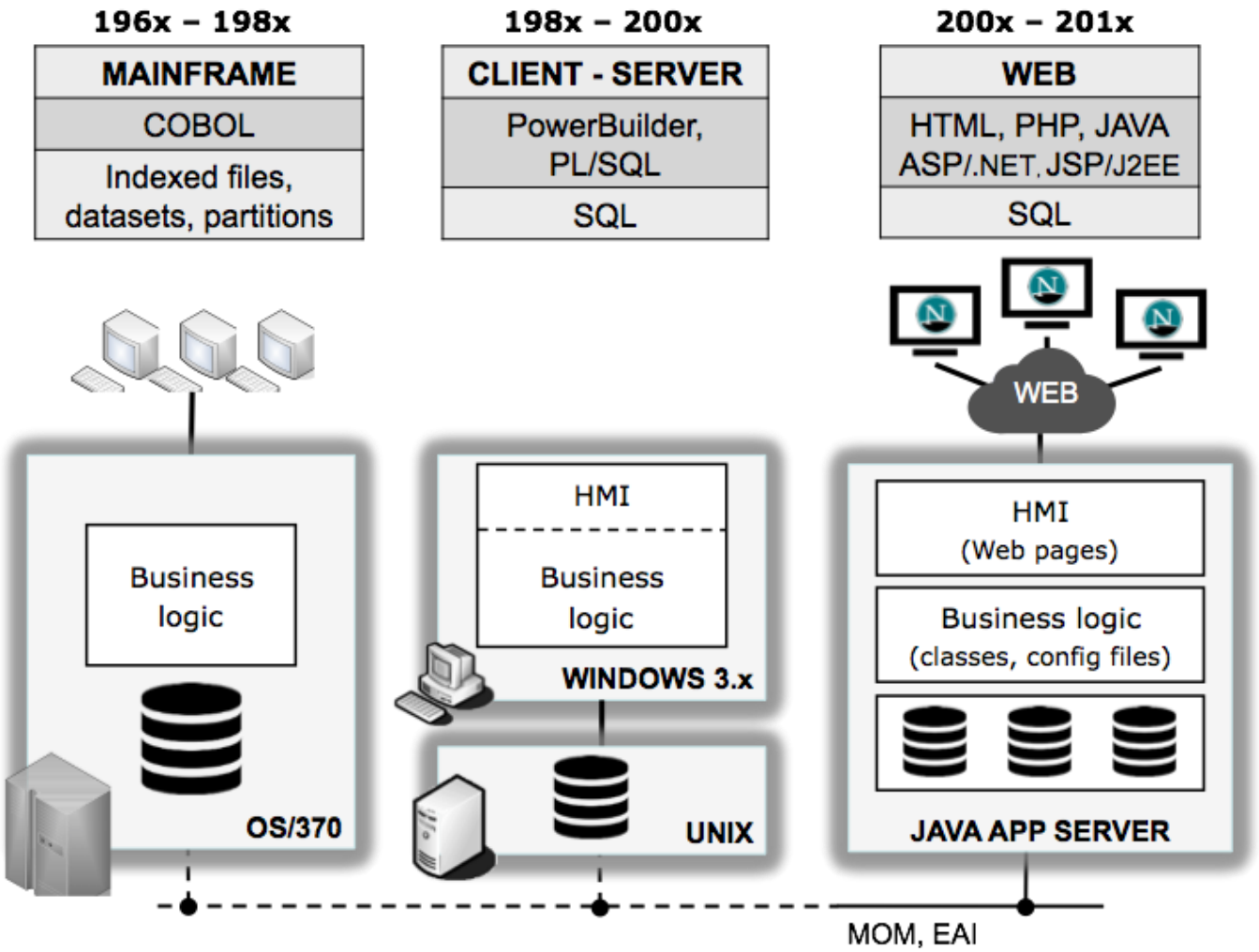
# Perspectives & Conclusion (temporary)



# A typical architecture…



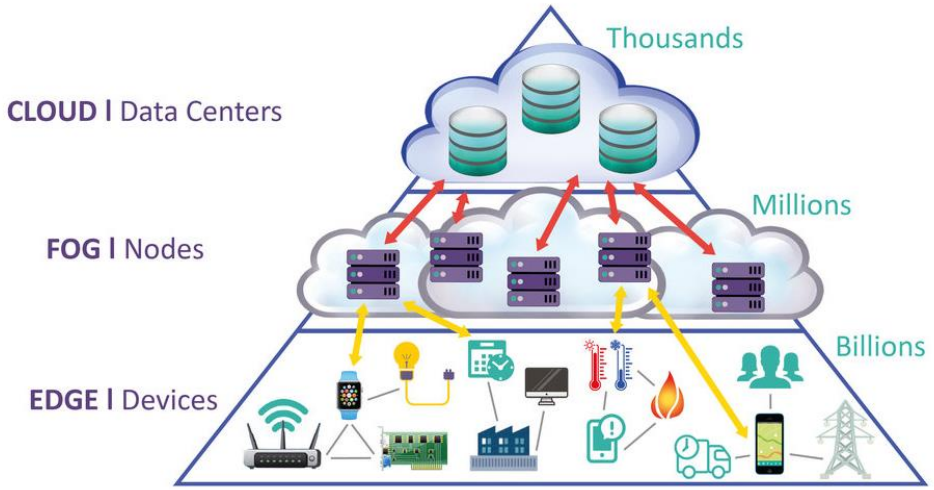
# Evolution is not over yet



# Ongoing evolution…
# Fog computing

- IoT objects generate a lot of data
- Sending them to the cloud for processing and storage is problematic.
  - Too much traffic
  - Too much latency

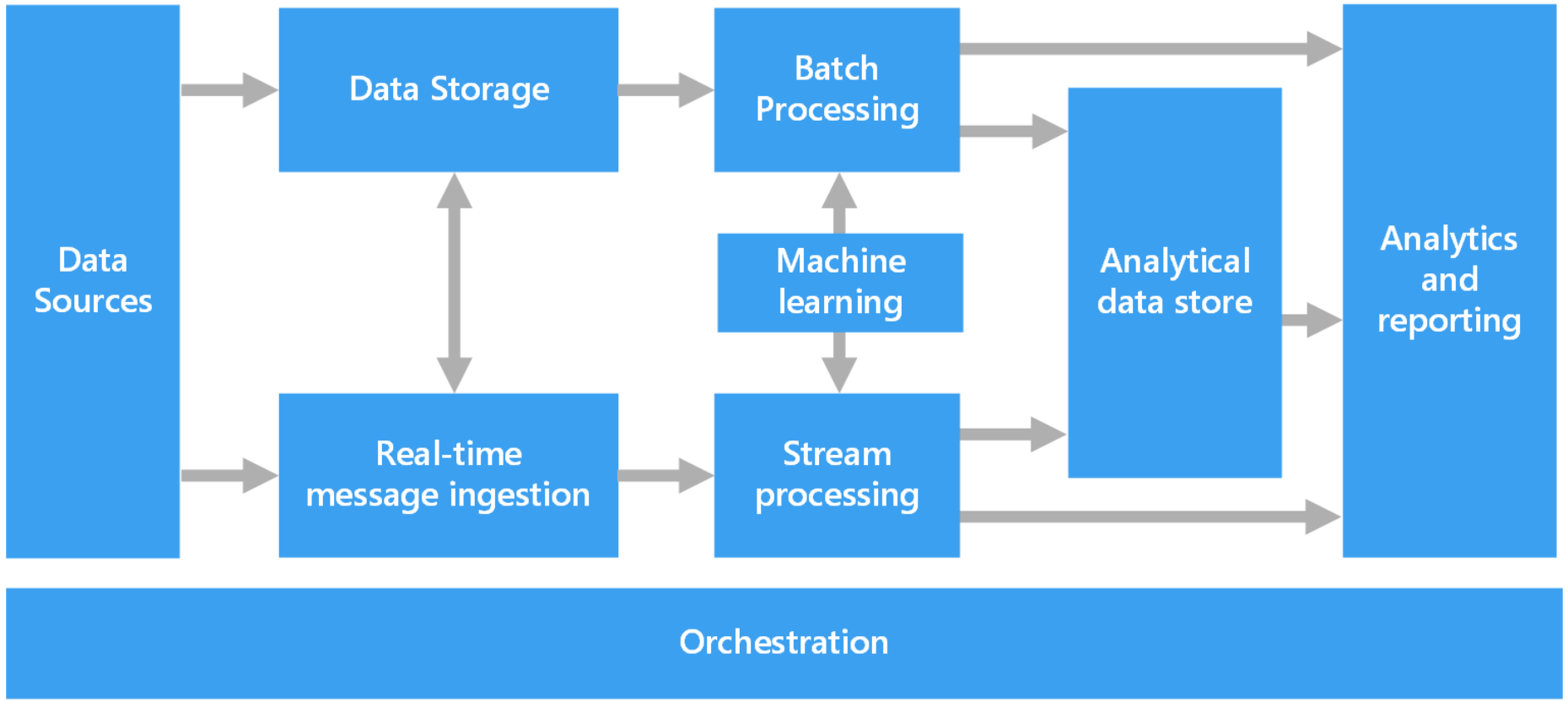
- idea: add an intermediate layer (the fog) between the cloud and the objects, closer to the objects




- Reinforces the entropy even more :)

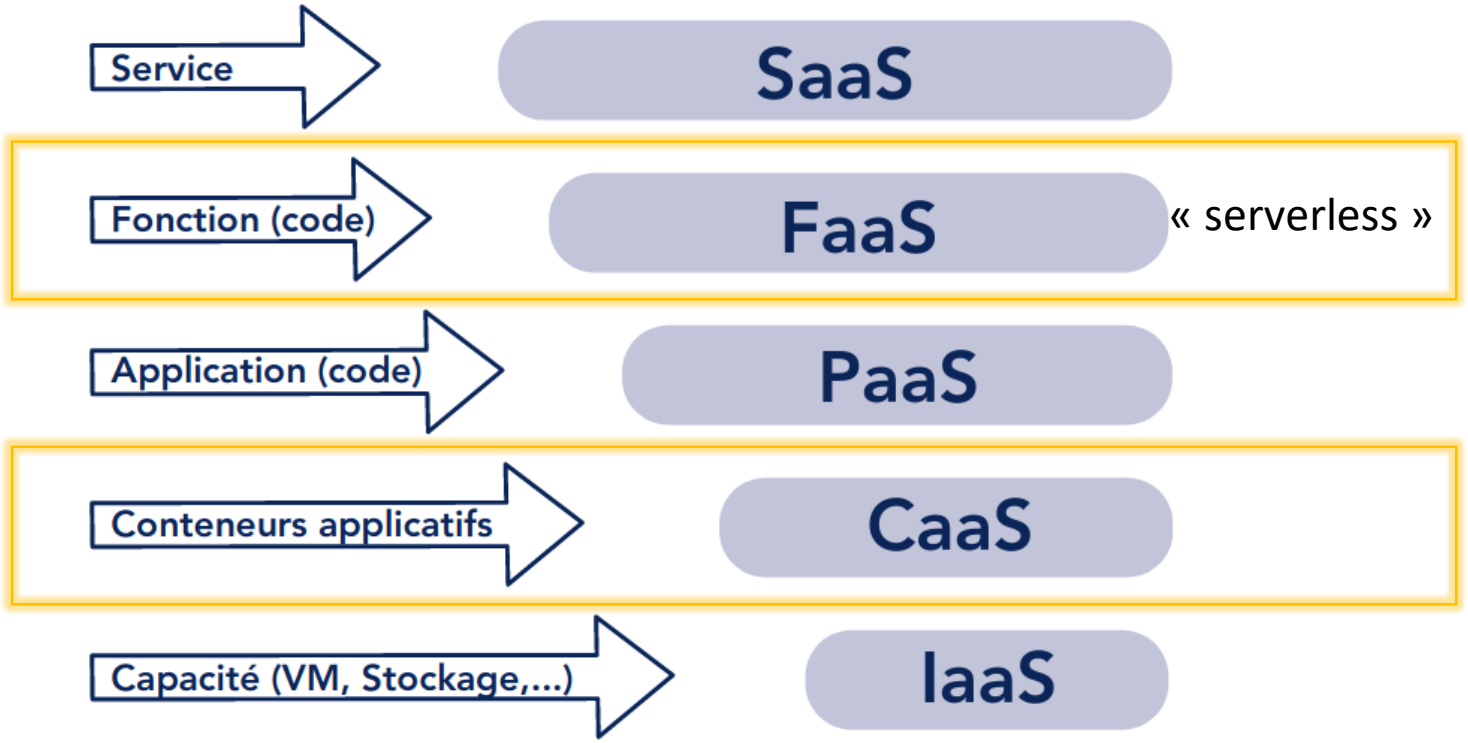
https://www.ntccloud.vn/dien-toan-bien-edge-computing-p3447.html

# Ongoing evolution…
# Big Data




https://docs.microsoft.com/bs-latn-ba/azure/architecture/data-guide/big-data/

# Ongoing evolution…
# new “aaS”



# Ongoing evolution…
# Startups

- Companies not quite like the others
  - Agility / Simplicity
  - Lack of financial resources
  - Strong technical competence / very keen on innovation
  - Strong growth
- Faster and lighter development methods
  - Agility and fast iterations are favored
    - Minimum Viable Product
    - Agile methods: Scrum, Kanban...
    - Full stack developers
  - Shorten production cycles
    - Devops trend
  - Languages/tools that allow rapid prototyping are preferred
    - Interpreted languages
    - Schema-less databases

# Startups : solutions privilégiées

- Mobile First strategy
  - Mobiles applications
  - one-page-app websites
  - REST Architectures, API, lightweight middleware
- Open Source solutions
  - Linux, PostgreSQL …
- Cloud solutions
  - Virtualization and containers
  - Amazon Web Services, Azure, Google
- New DB models
  - NoSQL
  - Big Data