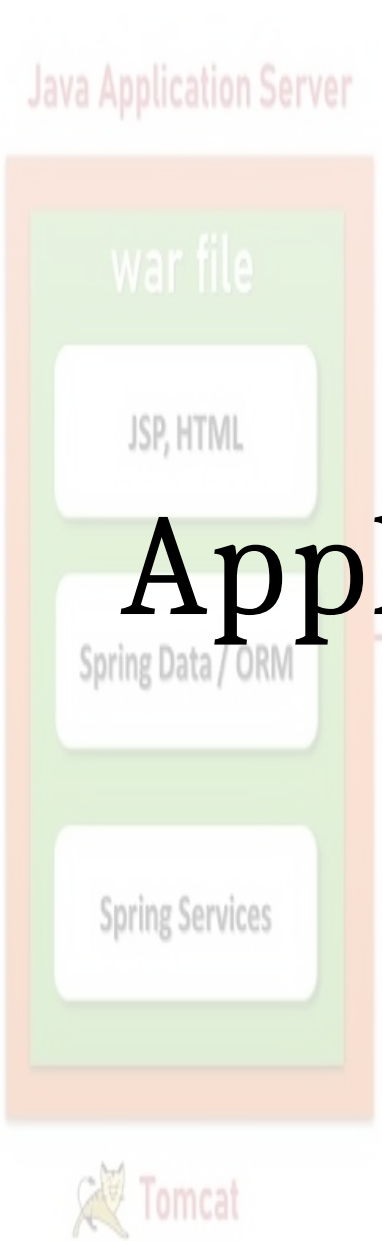
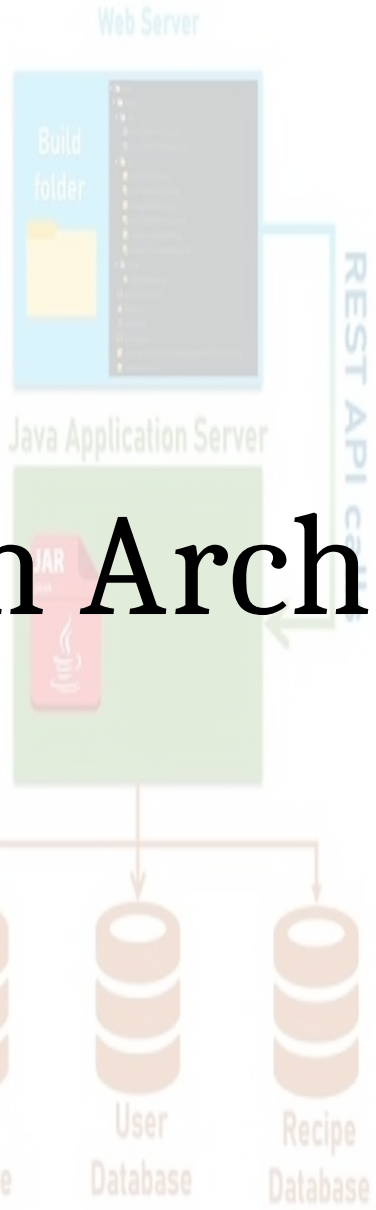
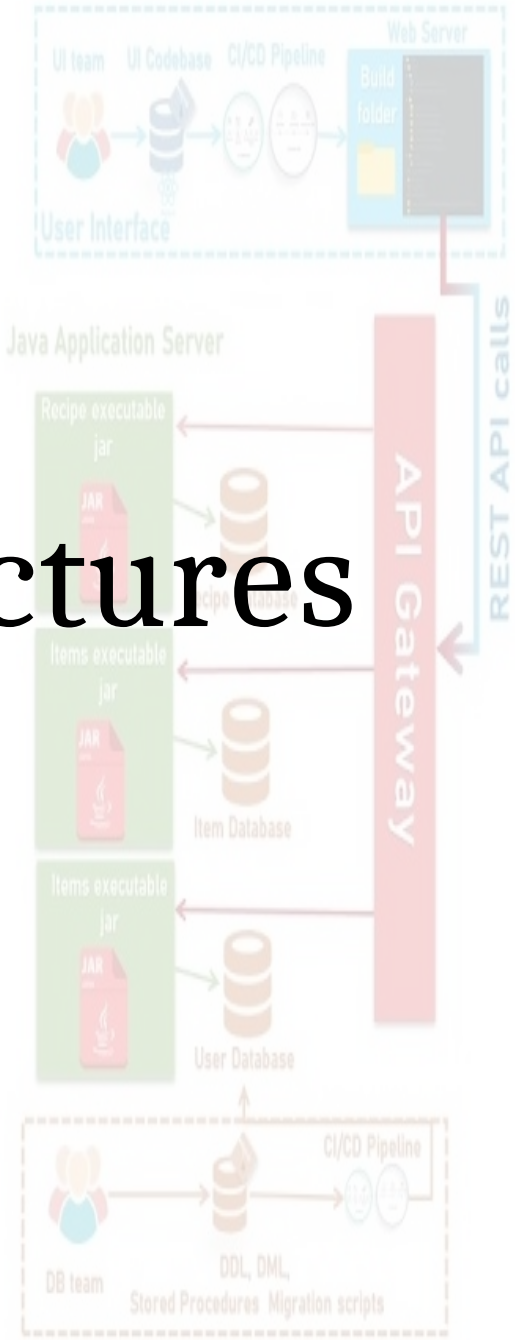
Application Architectures







Layered structure

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

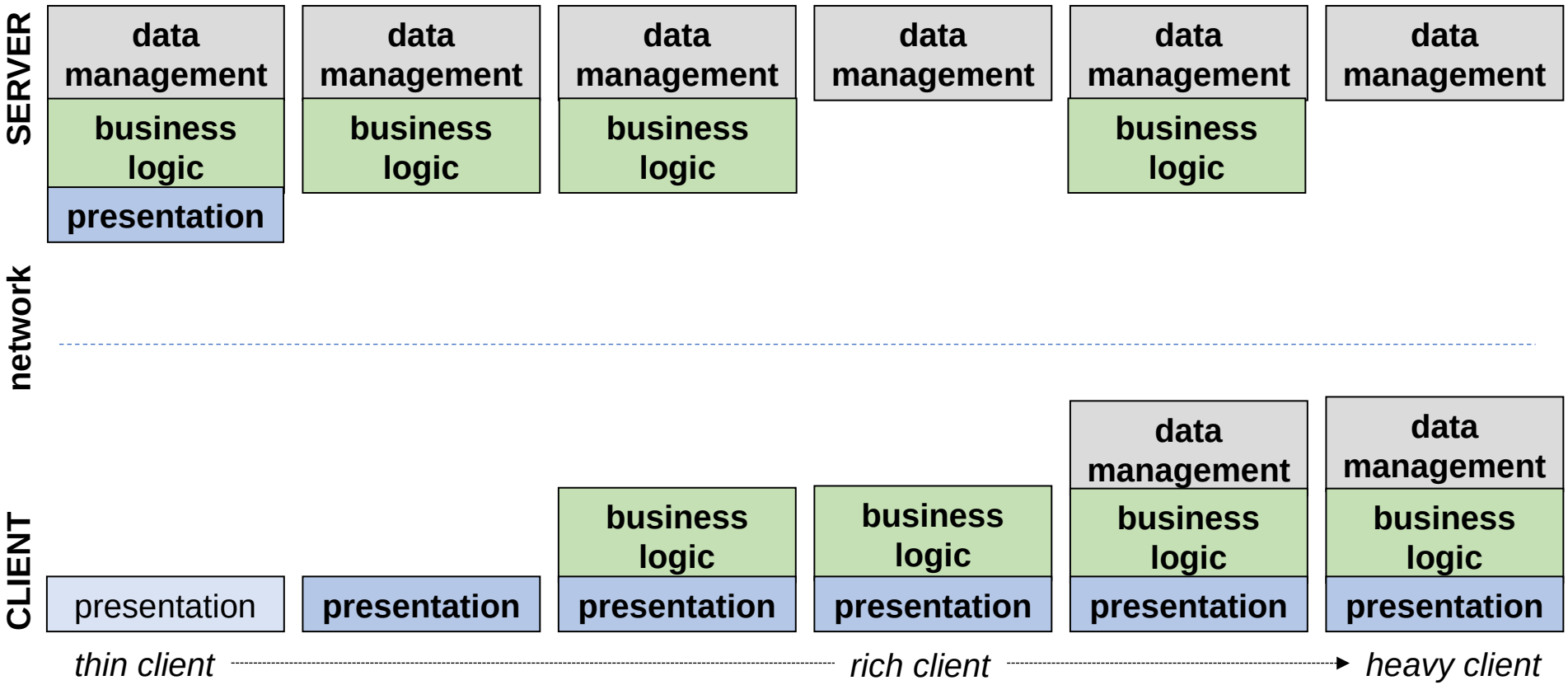
- Presentation:
  user input and commands, and display

- 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

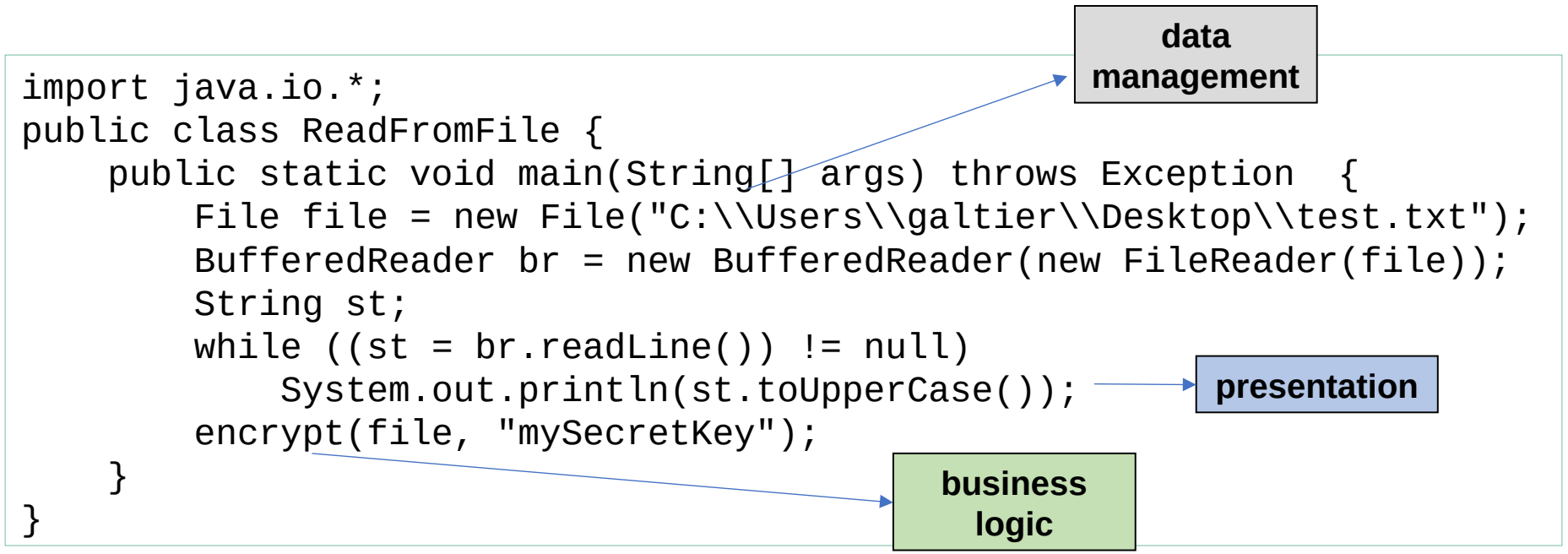


Monolithic and Single-tier Applications

Monolithic application

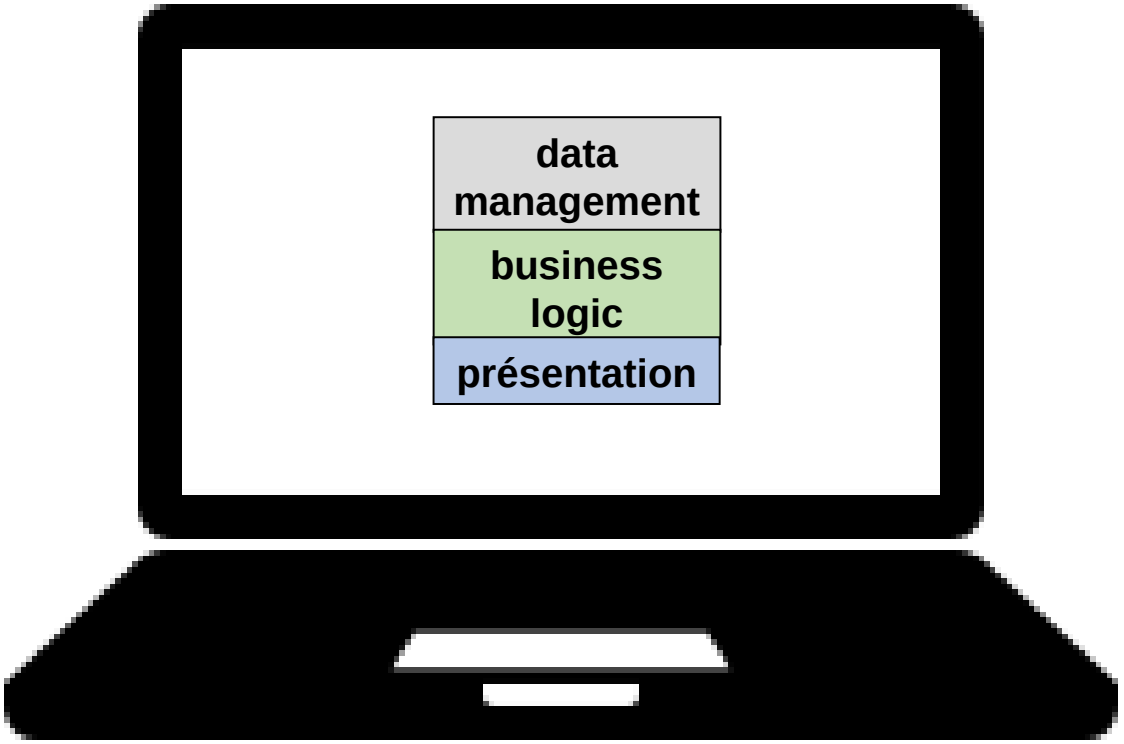
The 3 application layers are intimately interlaced in the same code base

```
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");
    }
}
```



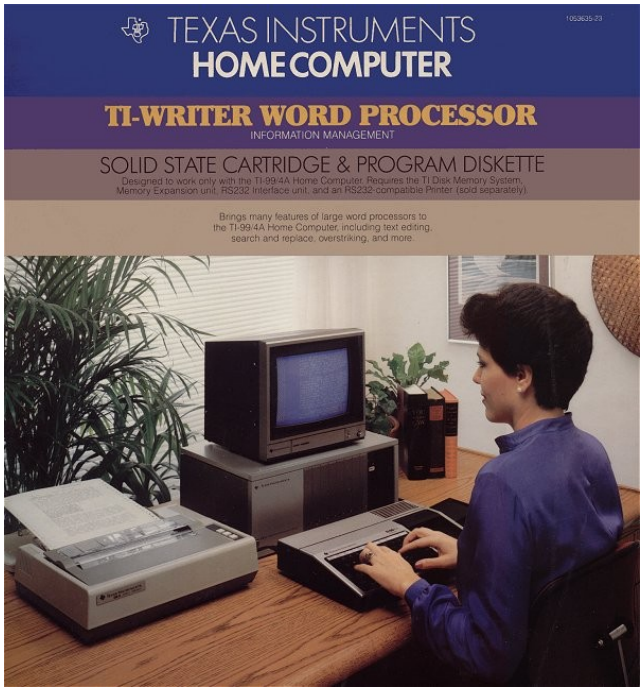
Single-tier Application

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)



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

- Still lots of stand-alone apps

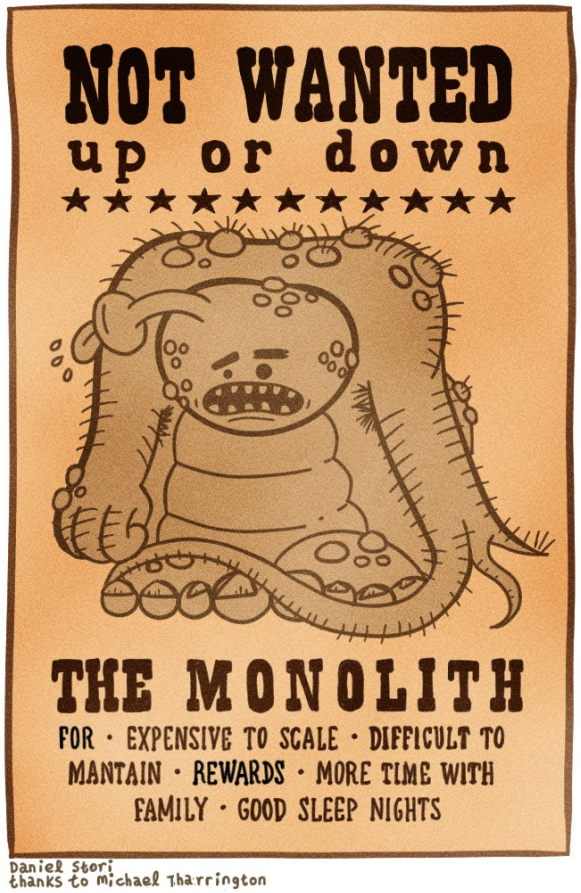
https://play.google.com/store/apps/details?id=com.medibang.android.paint.tablet&hl=en_SG&gl=US

Advantages of single-tier

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

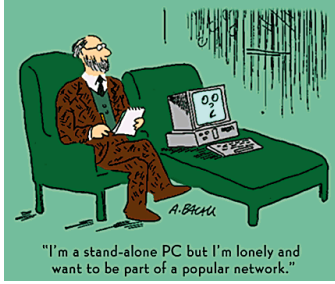
Disadvantages of monolithic applications

- 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



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

Disadvantages of single-tier applications



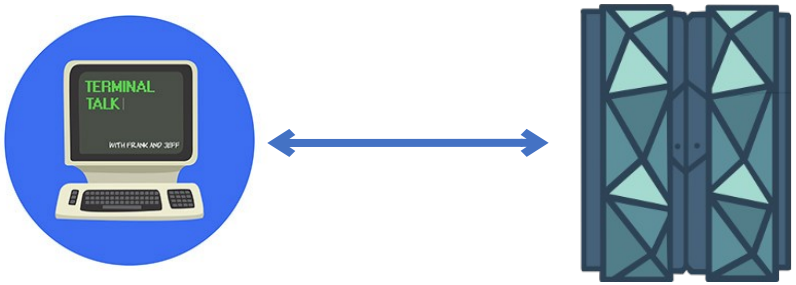
- Performances: depend on the capabilities of the host
- Shared resources impossible, requires duplicates (waste of resources)
- No fault tolerance
- Nomadism is difficult:
  - Access limited to physically logged-in users
  - More difficult (if not impossible) to continue a task from a different workstation
- 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://encyclopedia2.thefreedictionary.com/standalone+computer

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

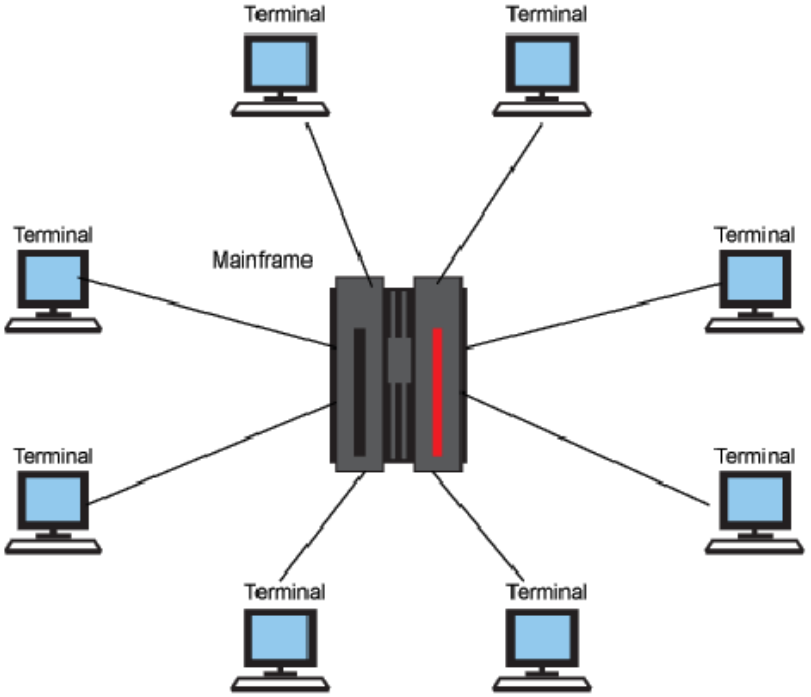
Mainframe Architectures

Principle “host” Architecture



- 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

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

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)

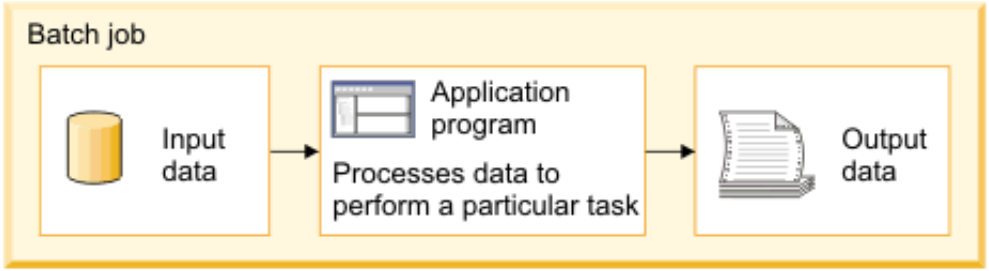
Robustness: https://www.ibmmainframeforum.com/mainframe-videos/topic10889.html

Performances

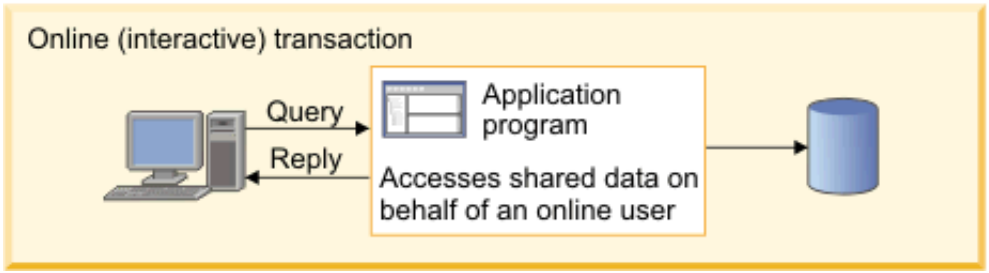
- Ability to process a very large number of simultaneous queries on very large databases

Batch or real time operation:

- Batch back-office



- Transactional



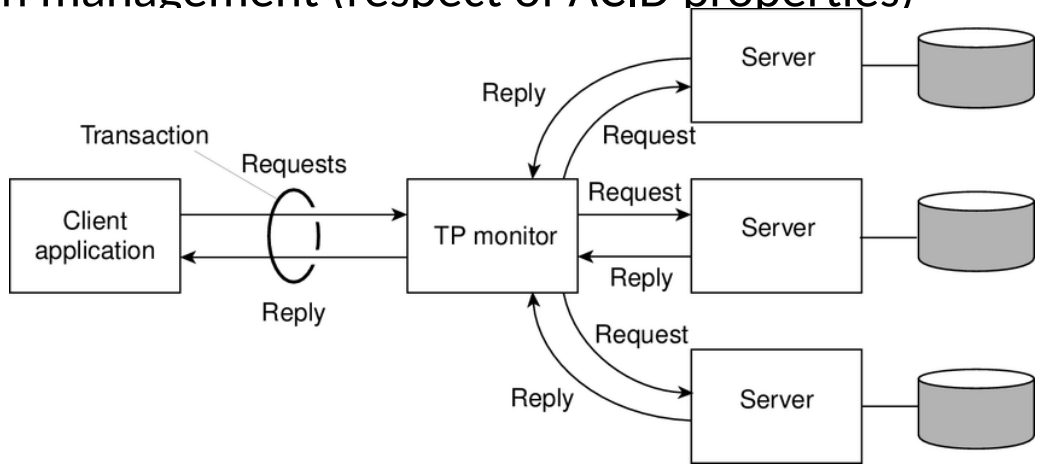
- Used in banks, insurance companies, airlines…

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)



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

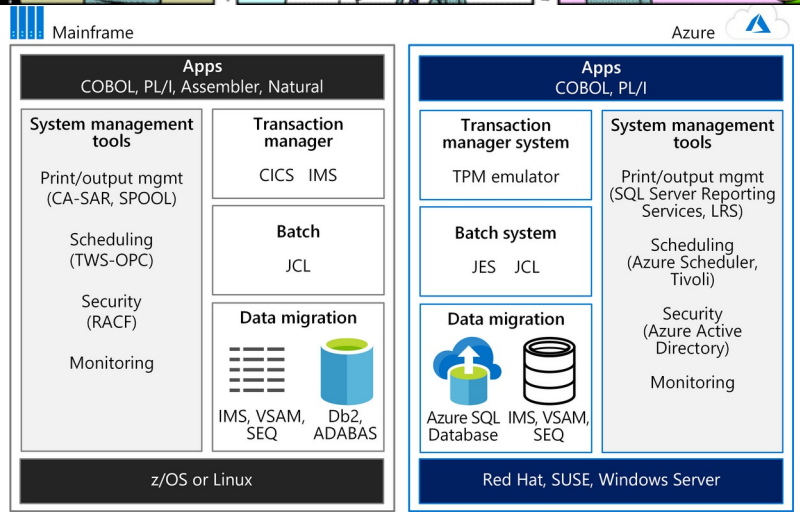
Obstacles to growth

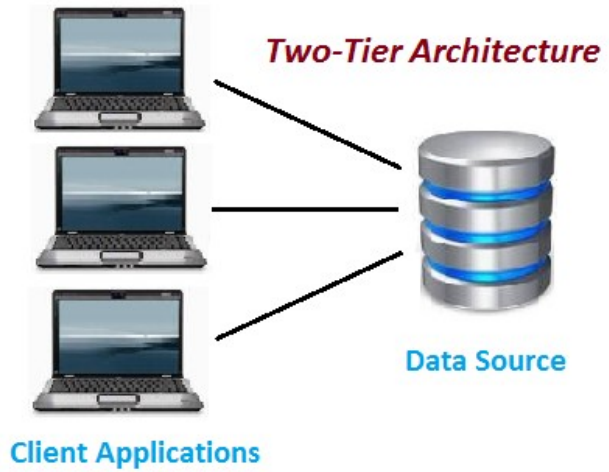
- Proprietary solutions
- Huge investment
  - but no more than a server farm

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

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

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



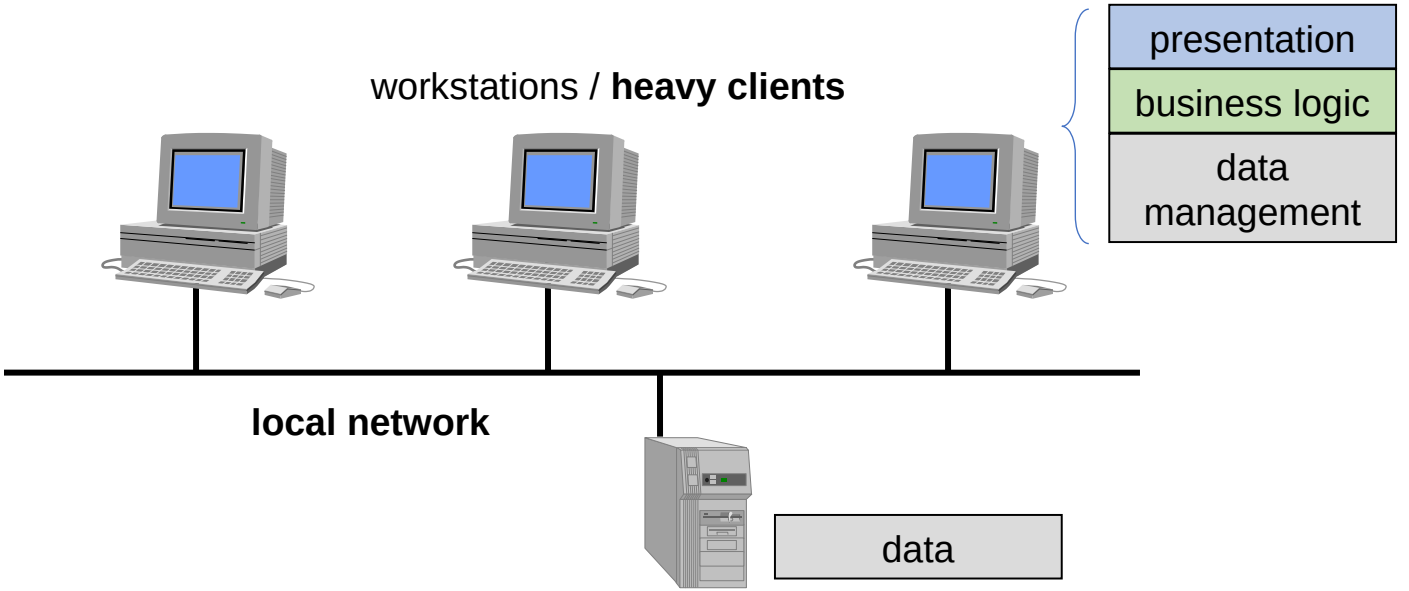


2-tier Architecture

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

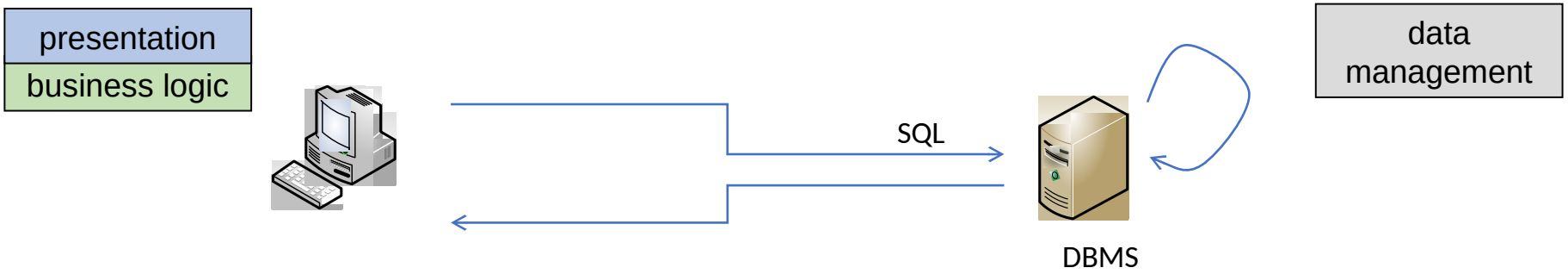
The origin: “1.5-tier” Architecture

- Development of LANs



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

2-tier Architecture



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

2-tier Architecture limits

- identical problems to single-tier:

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

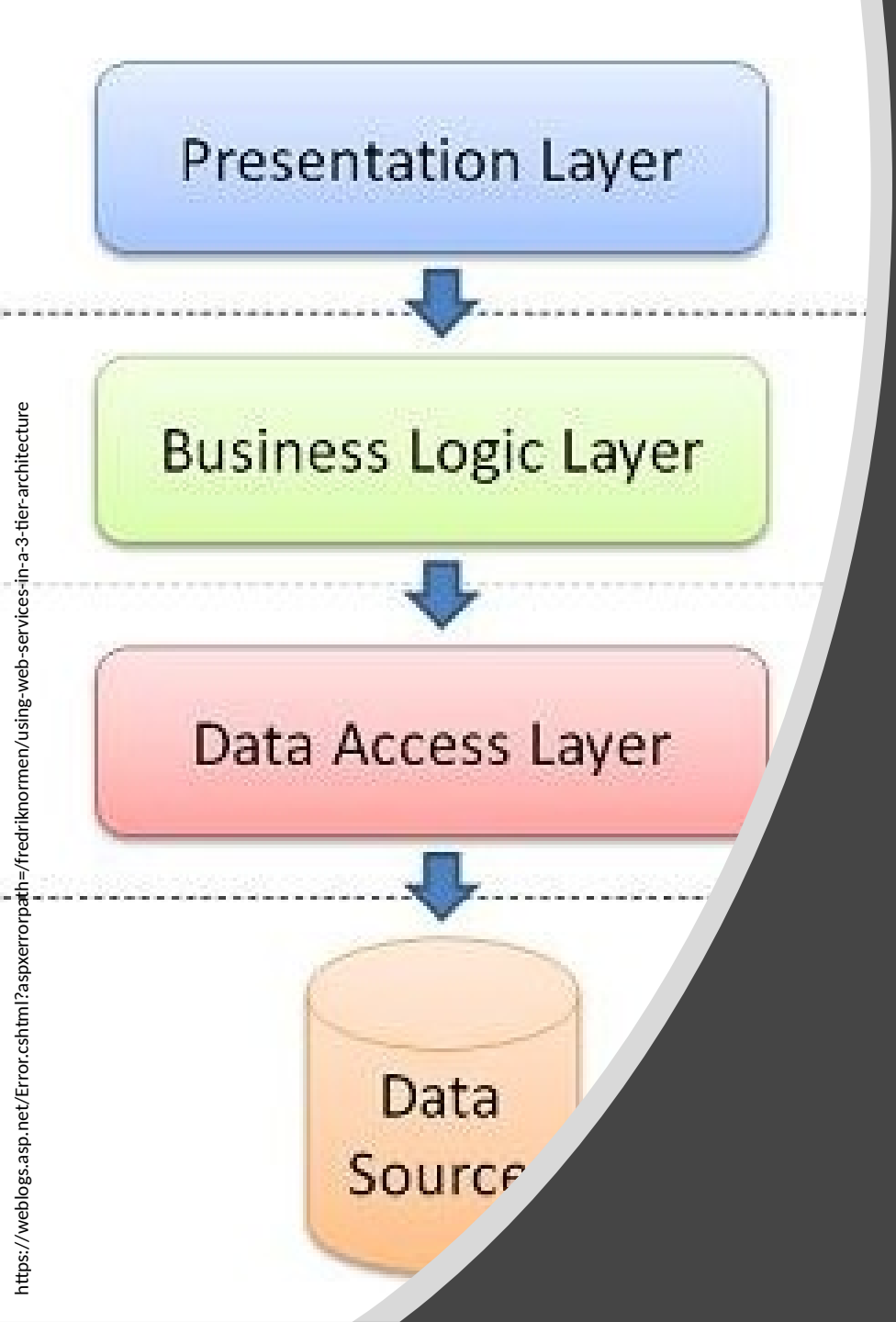
- excessive use of stored procedures:
  - breaks the principle of single responsibility
  - complex to maintain
  - adherence with the physical model
- performance :

  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



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

3-tier to 5-tier Architectures

3-tier

Client/Presentation Layer

Server/Application Layer

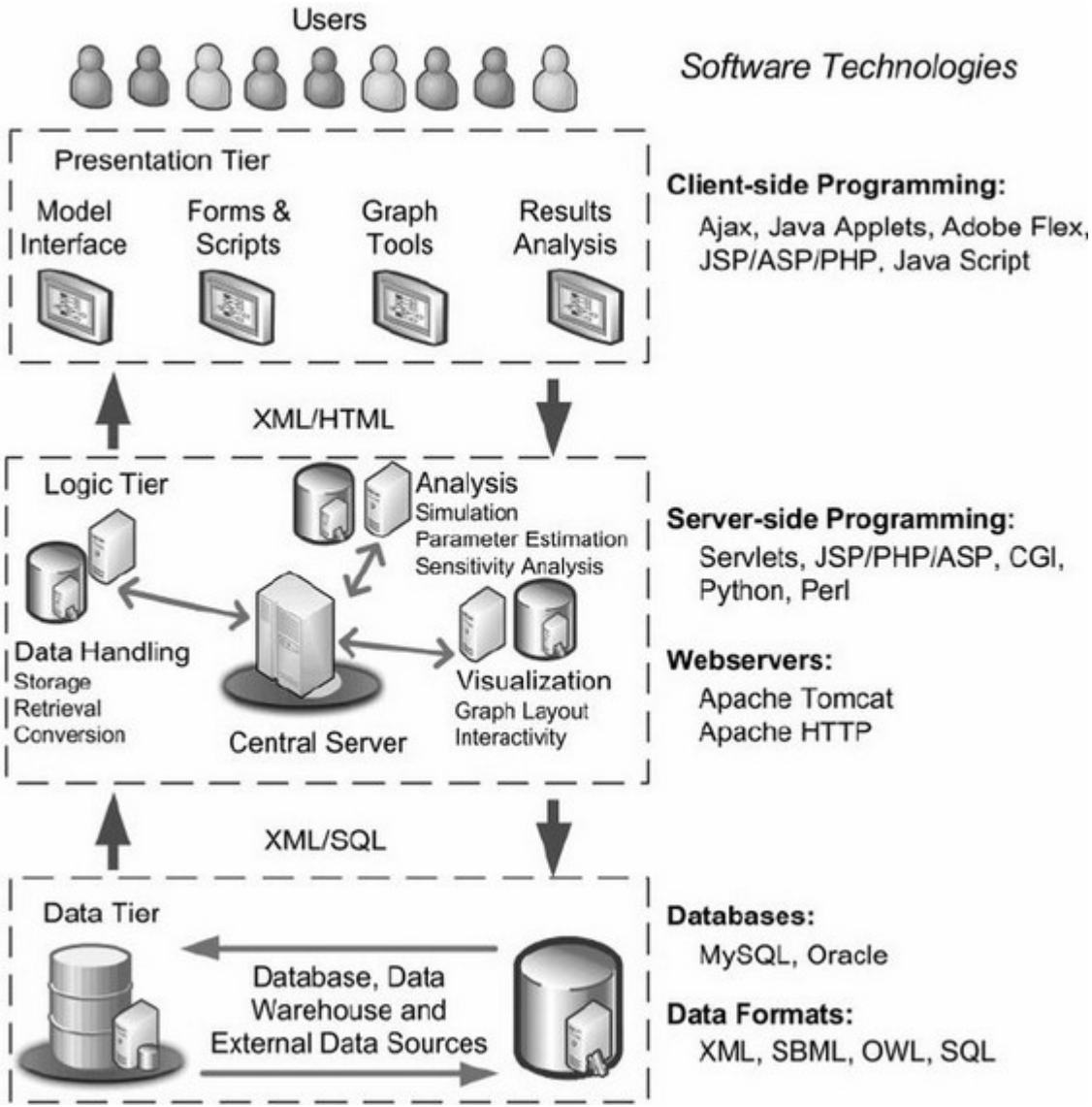
Database/ Data Layer

presentation

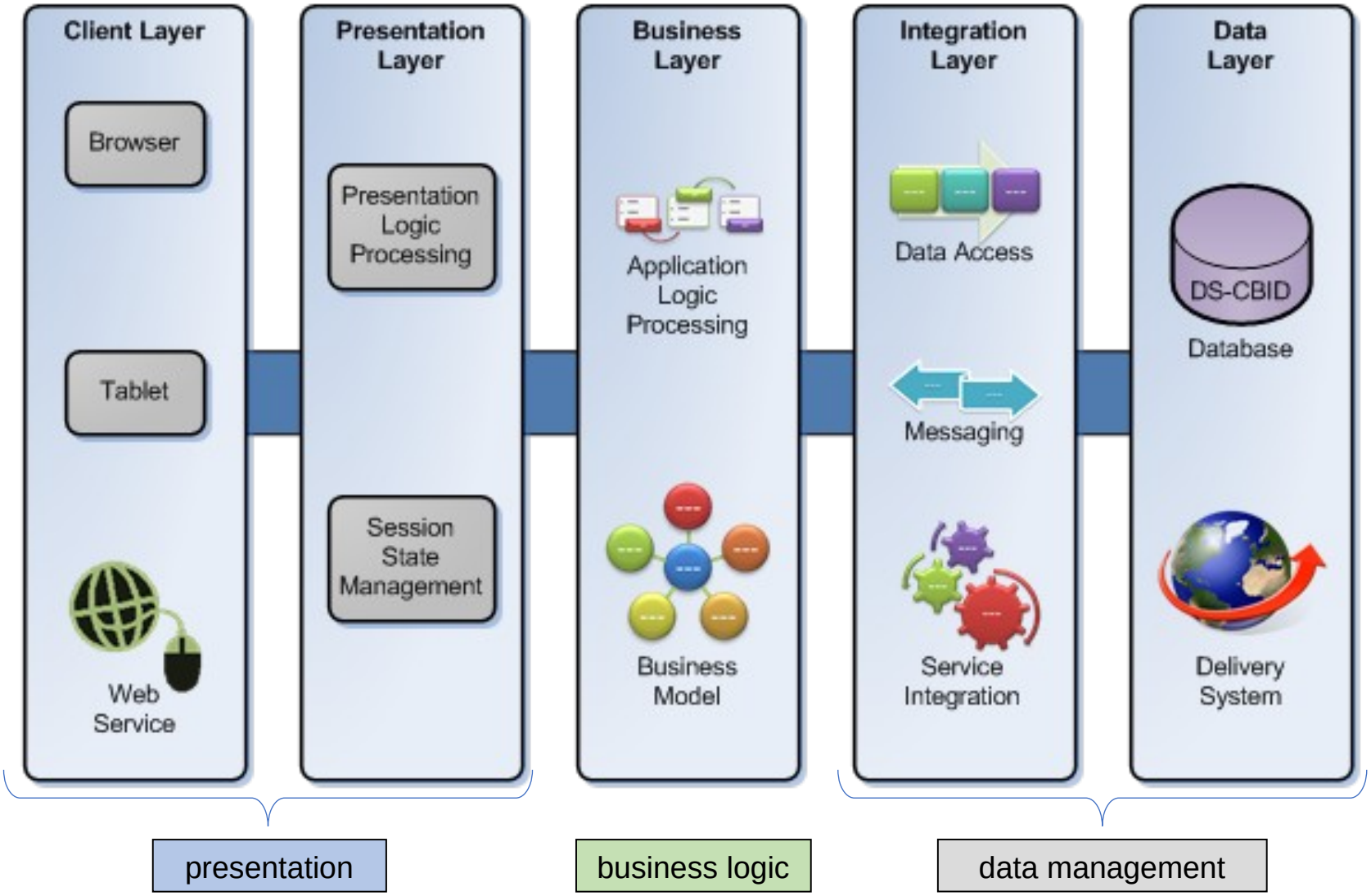
business logic

data management

Example:
Classical Web Architecture



4-tier, 5-tier



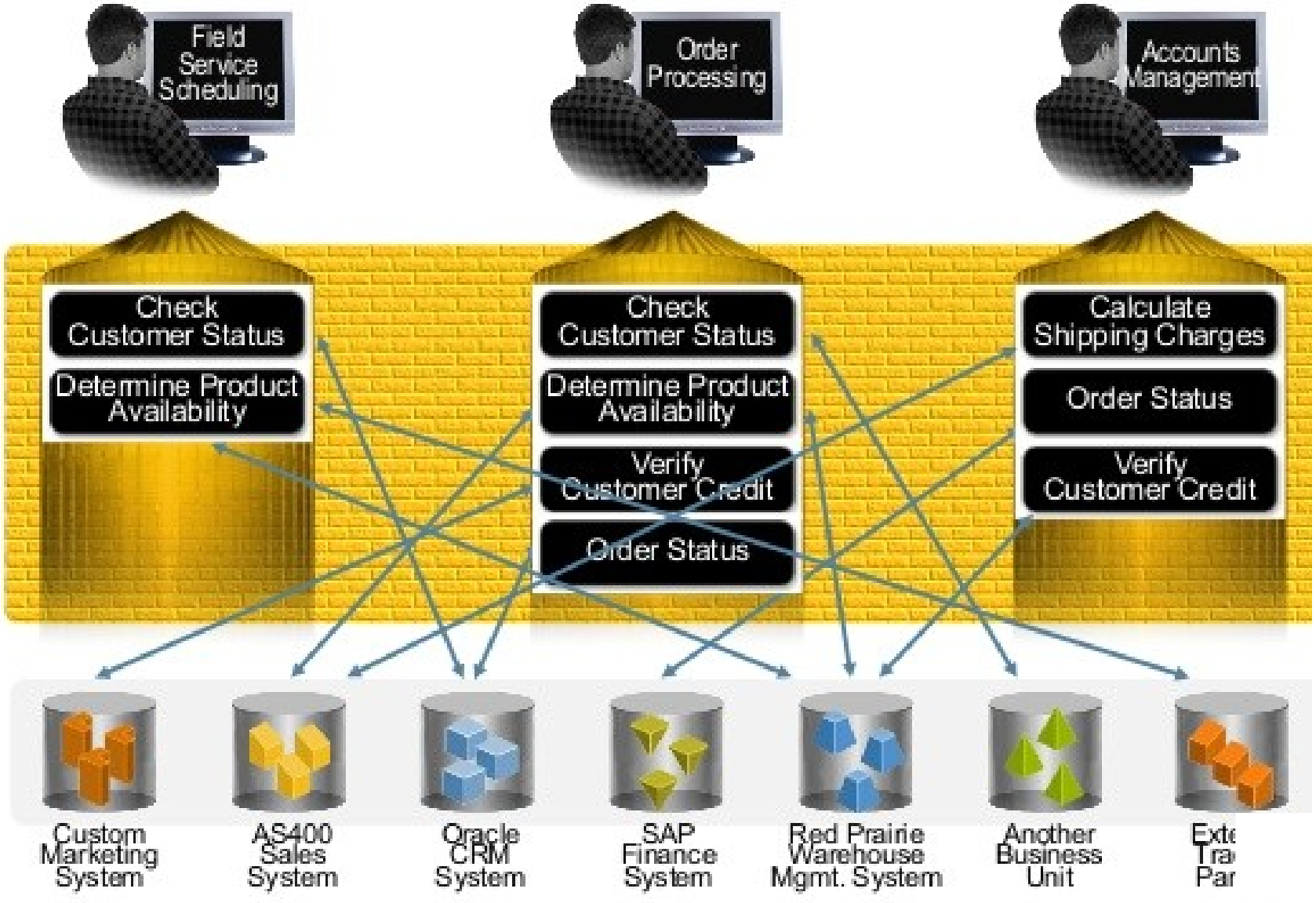
https://www.ajboggs.com/our-experiences/health-information-technology/sisonline-supports-assessment/sisonline-system-architecture/sissa-sis5_big/

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

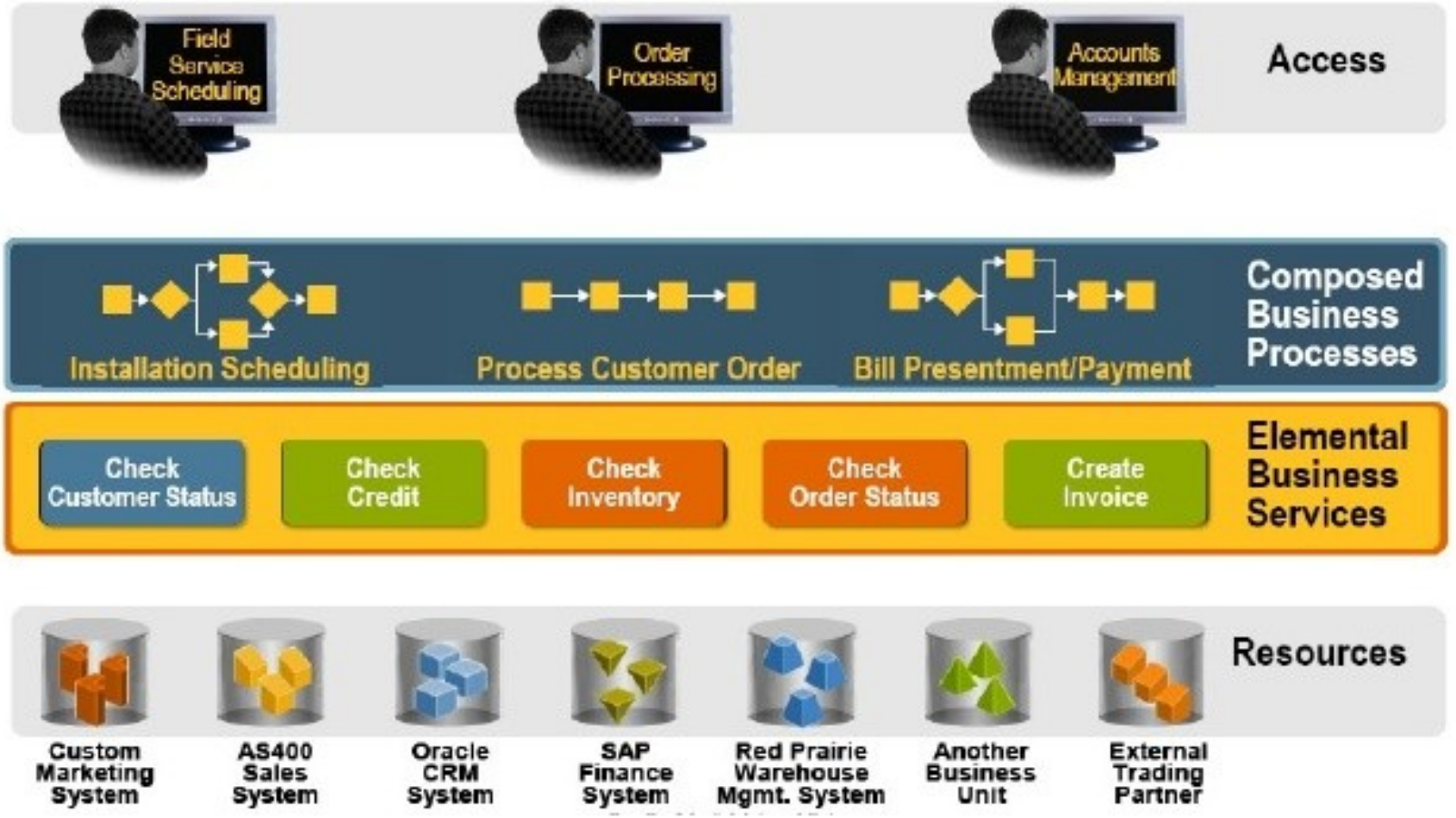
Siloed Architecture



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

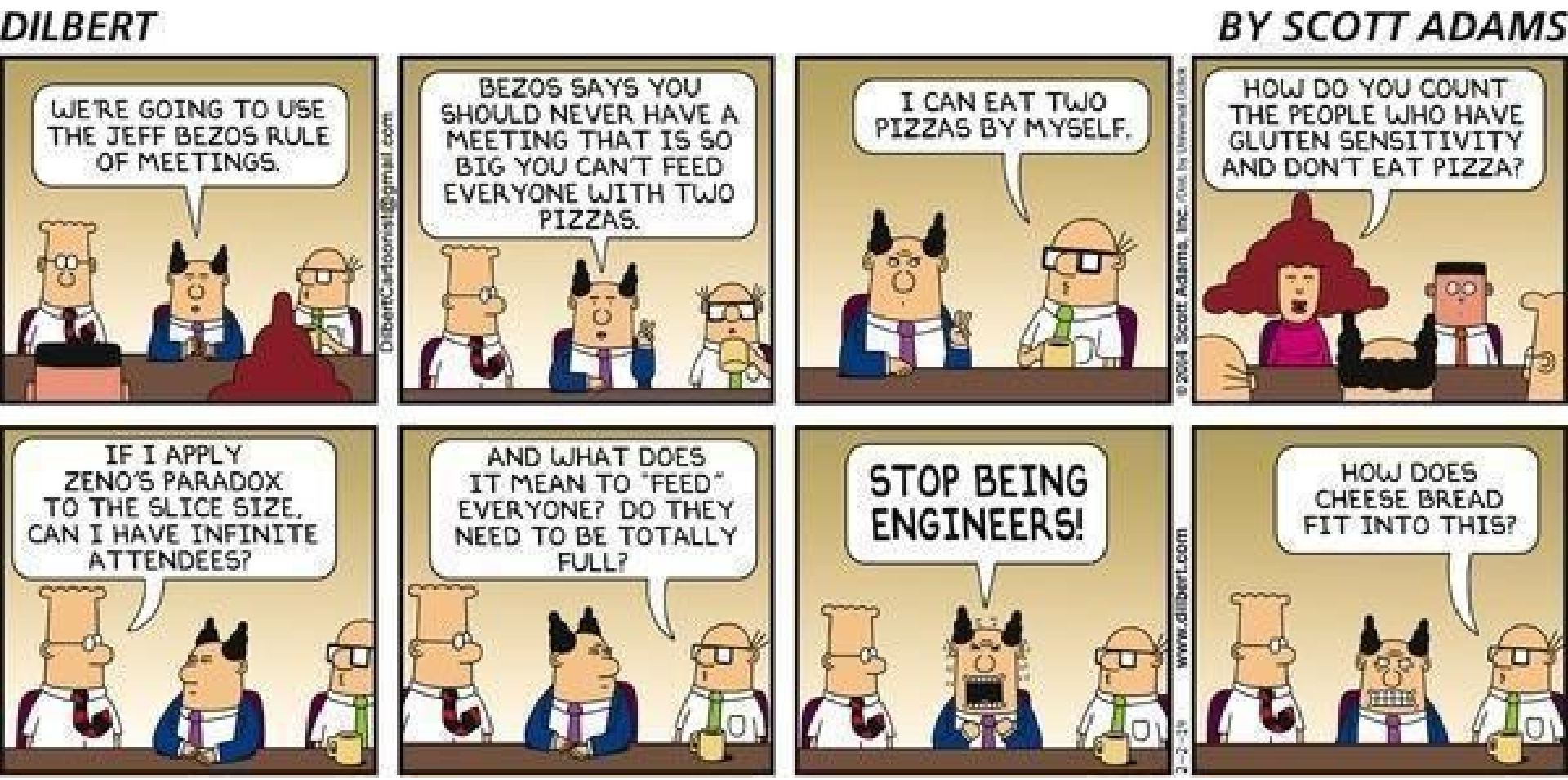


(Micro)Service Concept

- 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*

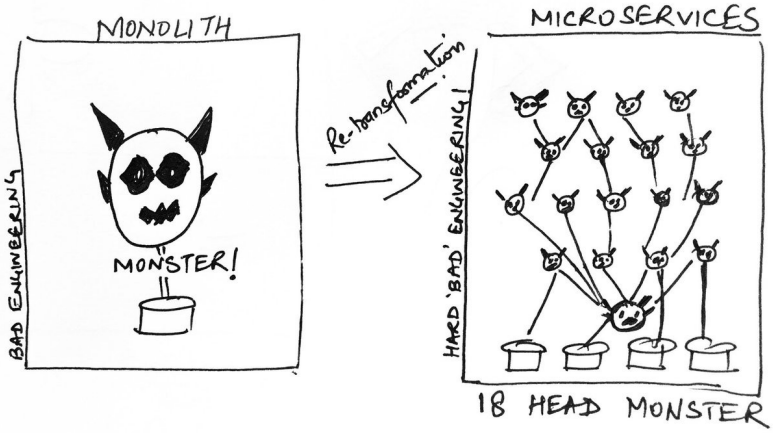
Advantages of the microservice architecture

- 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.
DilbertCartoonist@gmail.com
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.
© 2014 Scott Adams, Inc. /Dist. by Universal Uclick
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!
www.dilbert.com
HOW DOES CHEESE BREAD FIT INTO THIS?

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.



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

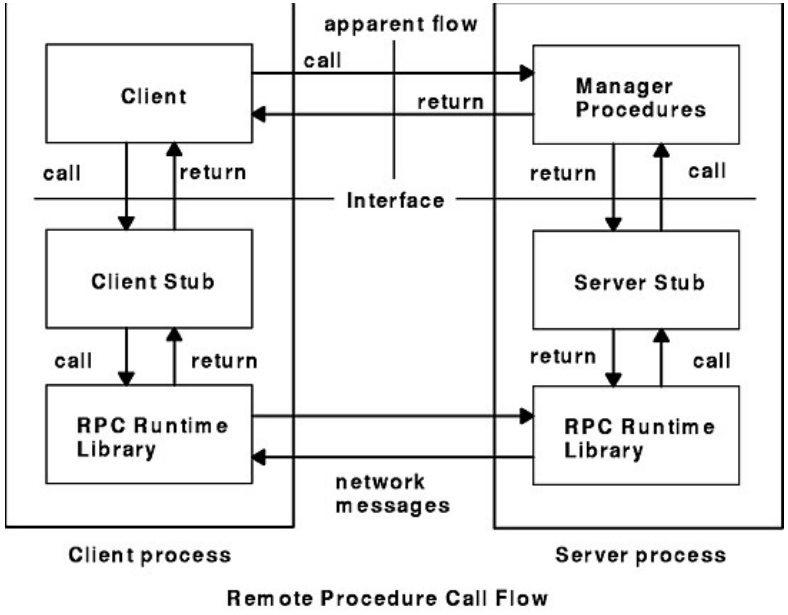
https://twitter.com/_zenx_/status/999333740122902529


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



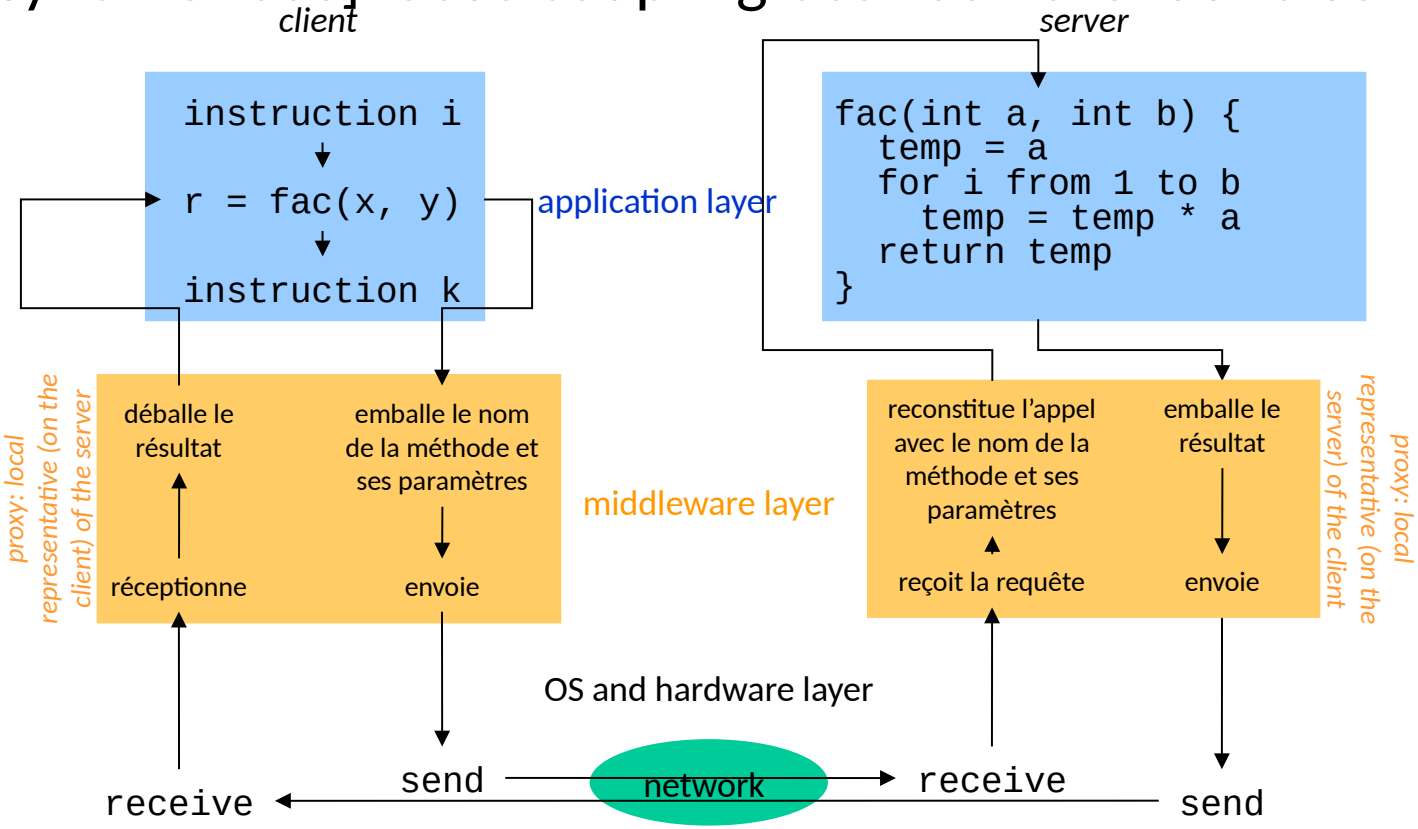
Remote Procedure Call Flow

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



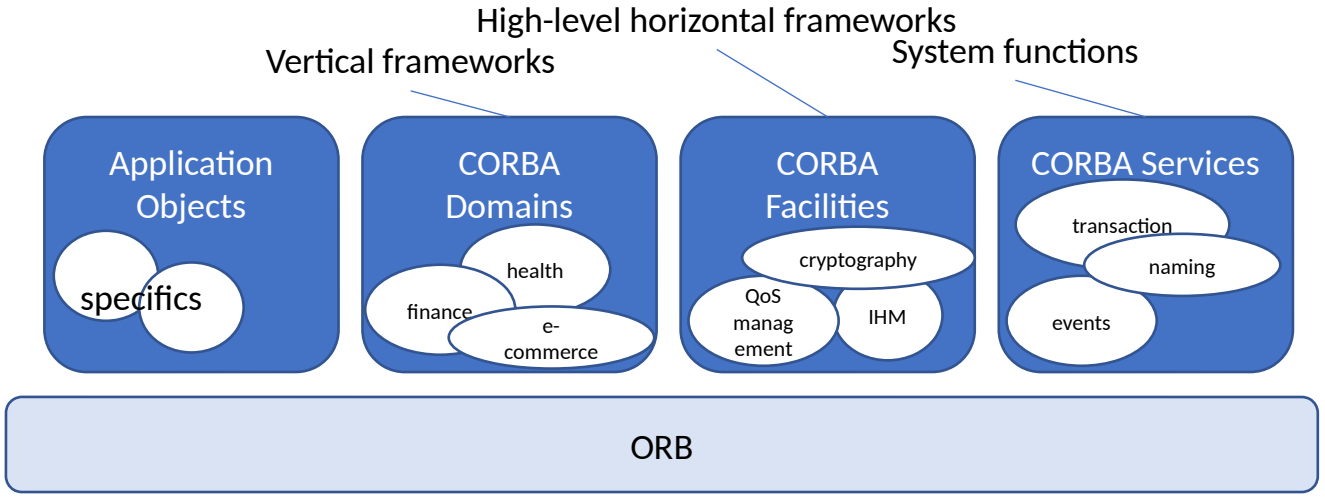
- The proxies handle:
  - network calls
  - format transformations between the client and server

(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

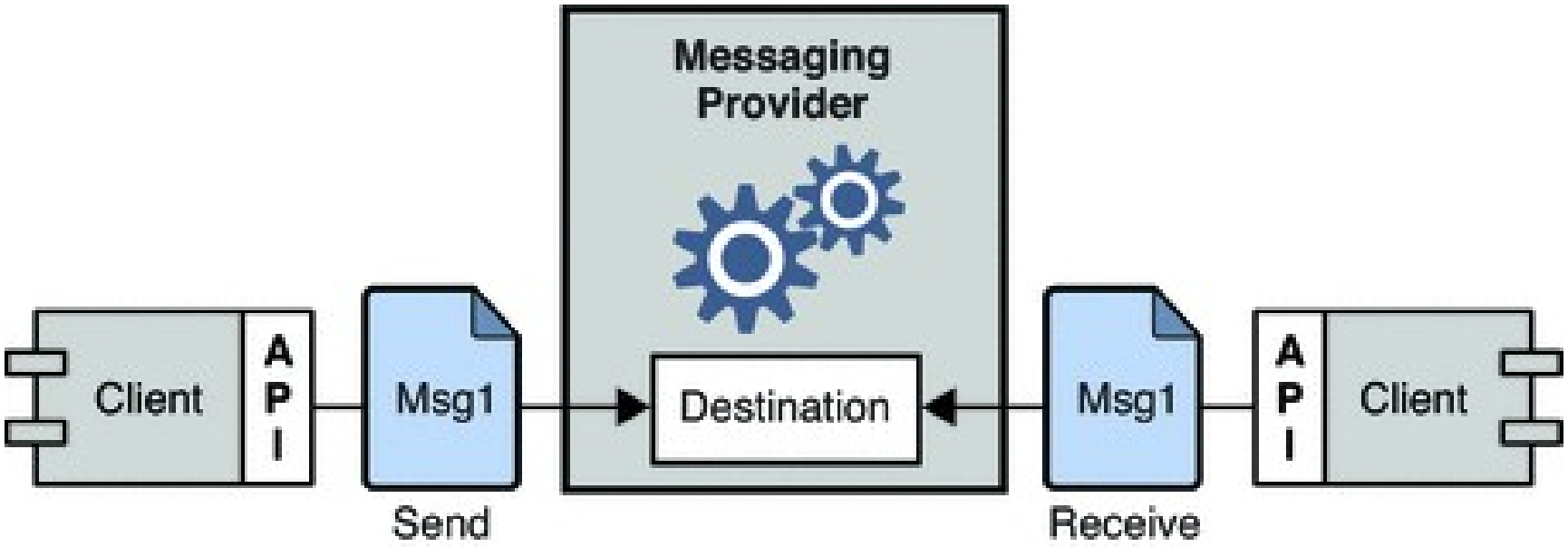
CORBA perspectives

- 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

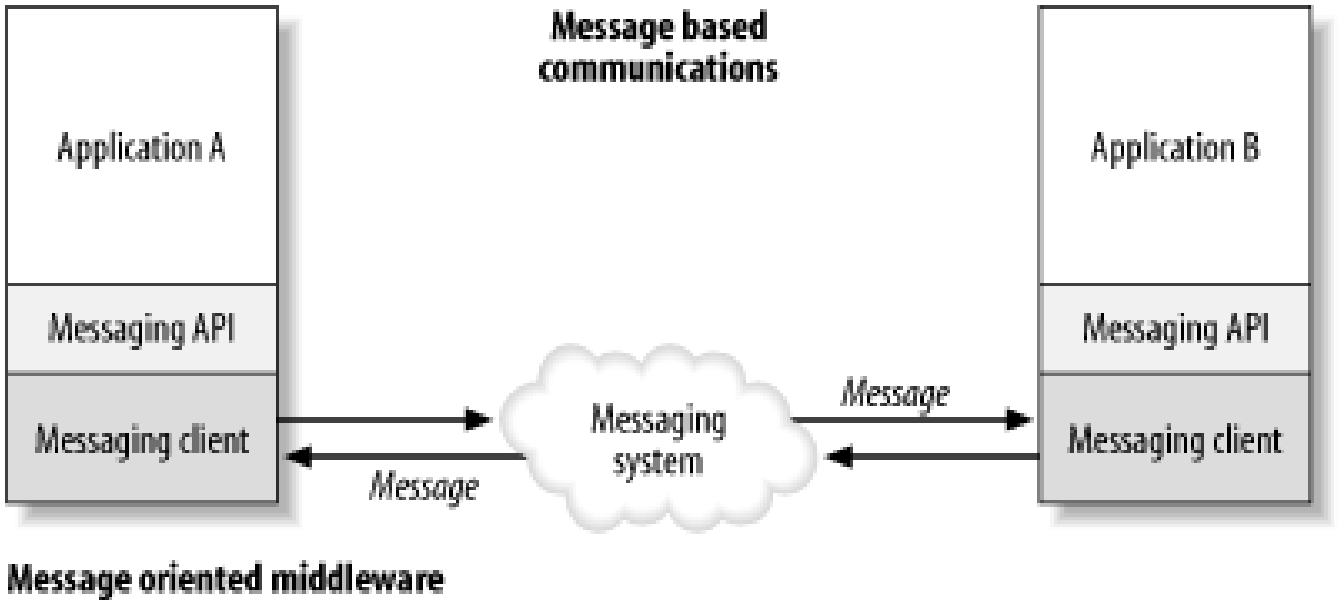
- 1srt generation Web Services:
  - Requests and responses transported by SOAP messages, usually on top of HTTP
  - 4 patterns supported by WSDL:
    - Request - response
    - One way request
    - Notification
    - Request - response
  - WS-*: myriad of specifications to complete the messaging service
- 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



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

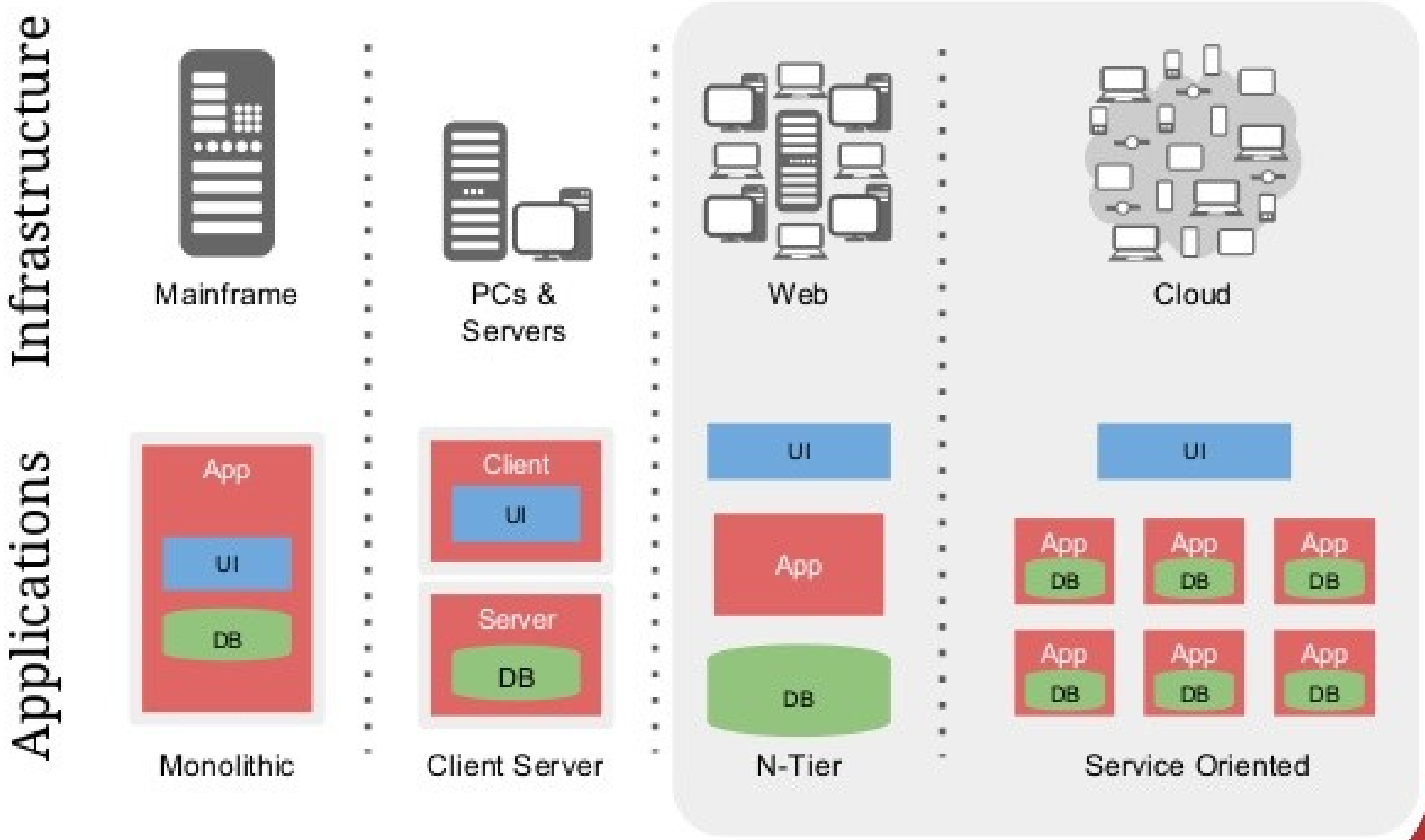
Optional Features

- 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
Monolithic
Client
UI
Server
DB
Client Server
UI
App
DB
N-Tier
UI
App
DB
Service Oriented