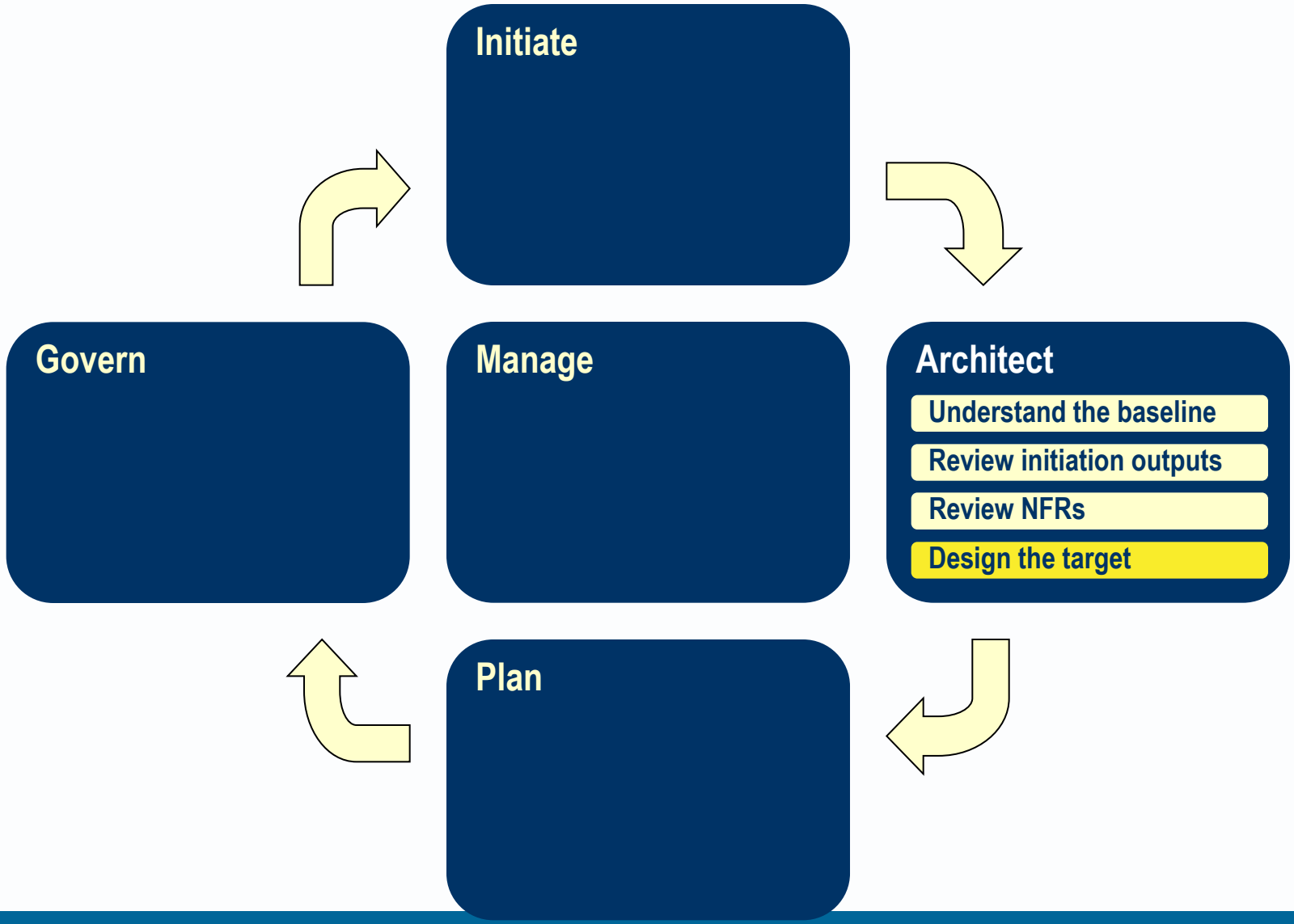
Avancier Methods (AM)
Solution Architecture

Design target infrastructure solution

Solution architecture

- Solution architects are not concerned to rationalise the enterprise technology portfolio
- They may not be motivated to use a standard enterprise application platform
- They look to deploy a solution on a platform that is the cheapest and quickest (consistent with meeting NFRs)

Where in the AM process?

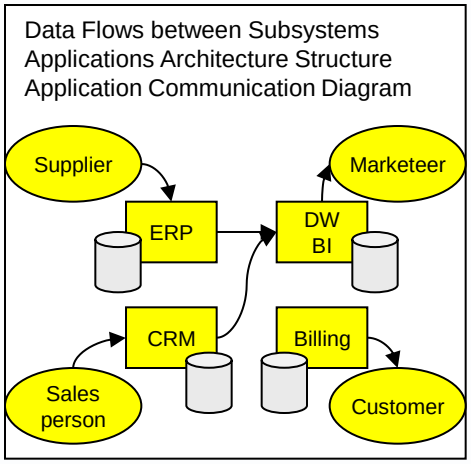


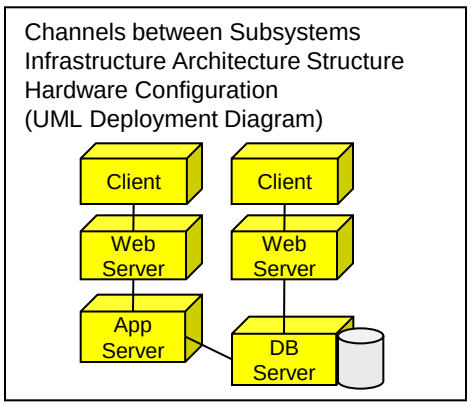
Theory: logical precedes physical

Logical applications architecture

Magical transformation

The physical infrastructure needed to run the applications





In practice, we need with a process

- Where to start?
- “At every turn you will be presented with a Chinese restaurant menu of [technology] choices:
  - Which client platform?
  - Which database server?
  - Which server platform?
  - Which network protocols?
  - Which distributed computing infrastructure?
  - Which component model?
  - Which system management base?“
- (Client/Server Survival Guide)

Design target infrastructure solution

- 1 Identify precursors (requirements and context)
- 2 Establish baseline opportunities and constraints
- 3 Define logical platform nodes
- 4 Map software to platform nodes
- 5 Map logical nodes to physical nodes
- 6 Define the network
- 7 Refine to handle NFRs
- 8 Define non-production environments
- 9 Govern deployment and transition into operations

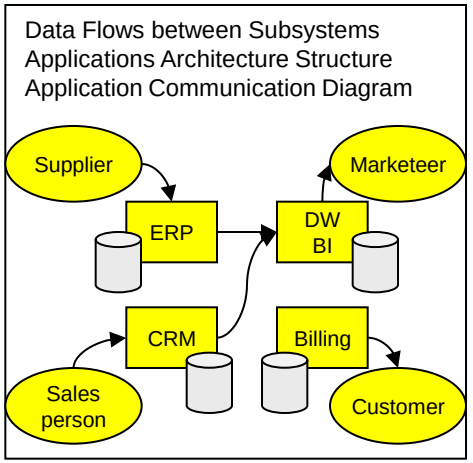
A process for defining the technologies that will support and run one or more applications.

A process that progresses through stages from a logical application-information view through progressively more physical views up to a hardware configuration diagram.

1 Identify precursors (requirements and context)

- First, collect what is known of

- Business applications
  - And the data flows or service invocations between them
- Geography
  - Locations of users, data stores and applications
- Non-functional requirements
  - Relating to users, data stores and applications
- Platform services
  - Needed by business applications



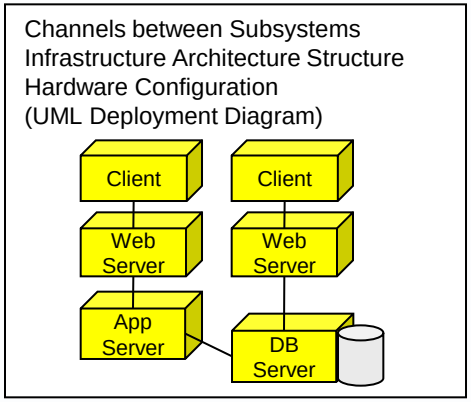
2 Establish baseline opportunity and constraints

- Does the enterprise constrain the choice and design of the infrastructure to support the target solution?
- By providing baseline infrastructure?
- By providing technology standards and directives?
- Can you use the existing infrastructure?

- Analyzing the current system configuration helps you understand what system performance depends on, the quality of the hardware, the configuration and the way the software works.
- You can use that information to tune and scale the overall architecture.

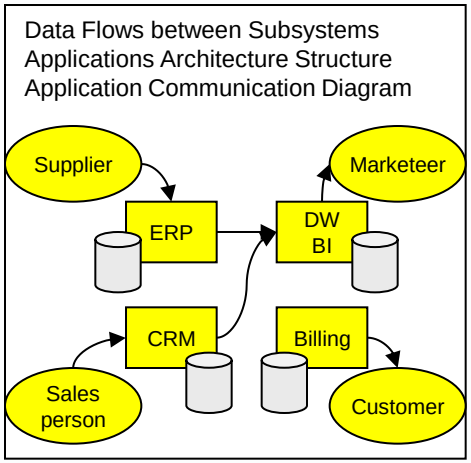
Sometimes we already have enough infrastructure

1. Let us create a domain or virtual machine on our existing server



2. Deploy our app on the virtual machine

3. And see if it works well enough



A dialogue with the capacity manager

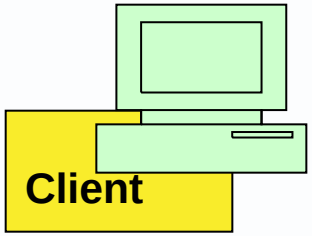
- The solution architect may ask
  - What exists already that we can use?
  - What hardware is given?
  - What network is given?
  - What performance targets and SLAs are guaranteed by the current system?
  - What enterprise standards and legal requirements are already met?
  - Are there any barriers to any possible expansion (e.g. no more rack space)?
- The capacity manager may ask
  - What is the requirement?
  - What kind of hardware do clients need?
  - What performance targets and SLAs must be met?
  - What new enterprise standards and legal requirements are to be met?
  - Is there a budget for new technology?

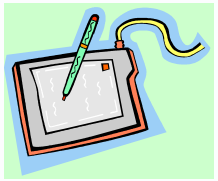
3 Define logical platform nodes

- Define client-end devices
- Define data servers
- Define the top-to-bottom server stack
- Label nodes with roles
- Label nodes with identifiers
- Define node operating systems

Define client-end devices

- Client devices can be a show stopper
- Where are they?
- What are they? Lap top specification must include any platform needed by the apps
- Who can and will use them? Employees must be willing and able to carry portable equipment
- How will they connect to servers?





Client devices

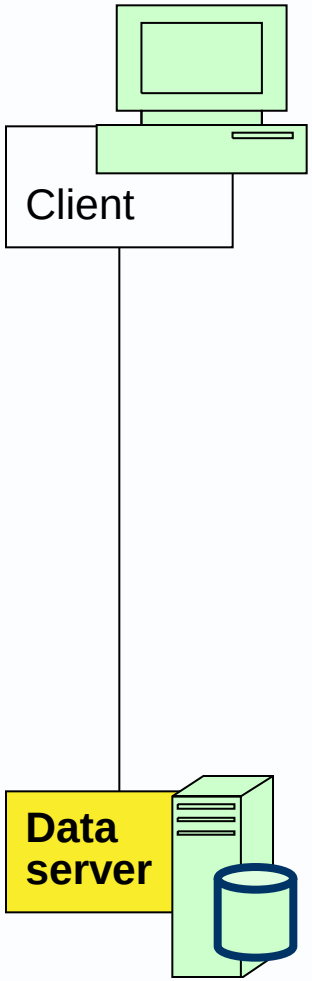
- Users and their locations
  - How many users?
  - Where are they?
  - Must they be mobile?
  - What kind of device can they afford?
  - What level of security is needed for particular users?
- Client devices
  - How many client devices.
  - What kind of devices?
  - What will be on them?
  - What kind of client machine will users work with?
  - Do they work at a desk? are they mobile?
  - What machines can they afford?
  - What do you want to restrict them to?
- Client machines might take the form of:
  - GUI clients such as workstations, laptops, home PCs connected via the Internet, and kiosks.
  - Non-GUI clients such as mobile phones, ATMs, smart cards and interactive voice response (IVR) devices.
  - Industry/domain-specific front-end data capture devices. E.g. point of sale terminals in retail, automatic teller machines in banking, and ticketing machines in the rail industry.
- Secondary client devices, that is I/O devices connected to primary client devices,
  - barcode readers
  - scanners
  - printers
  - fax machines

Define data stores and data servers

- The availability of data via data servers can be a show stopper

- Can existing data servers be used?
- Where is the data actually stored?
- Disc attached to data server? NAS? SAN?
- Is data available when needed - at right time of day?
- Are users authorised to access the data?

- In our example data is stored on a disc attached to the server.



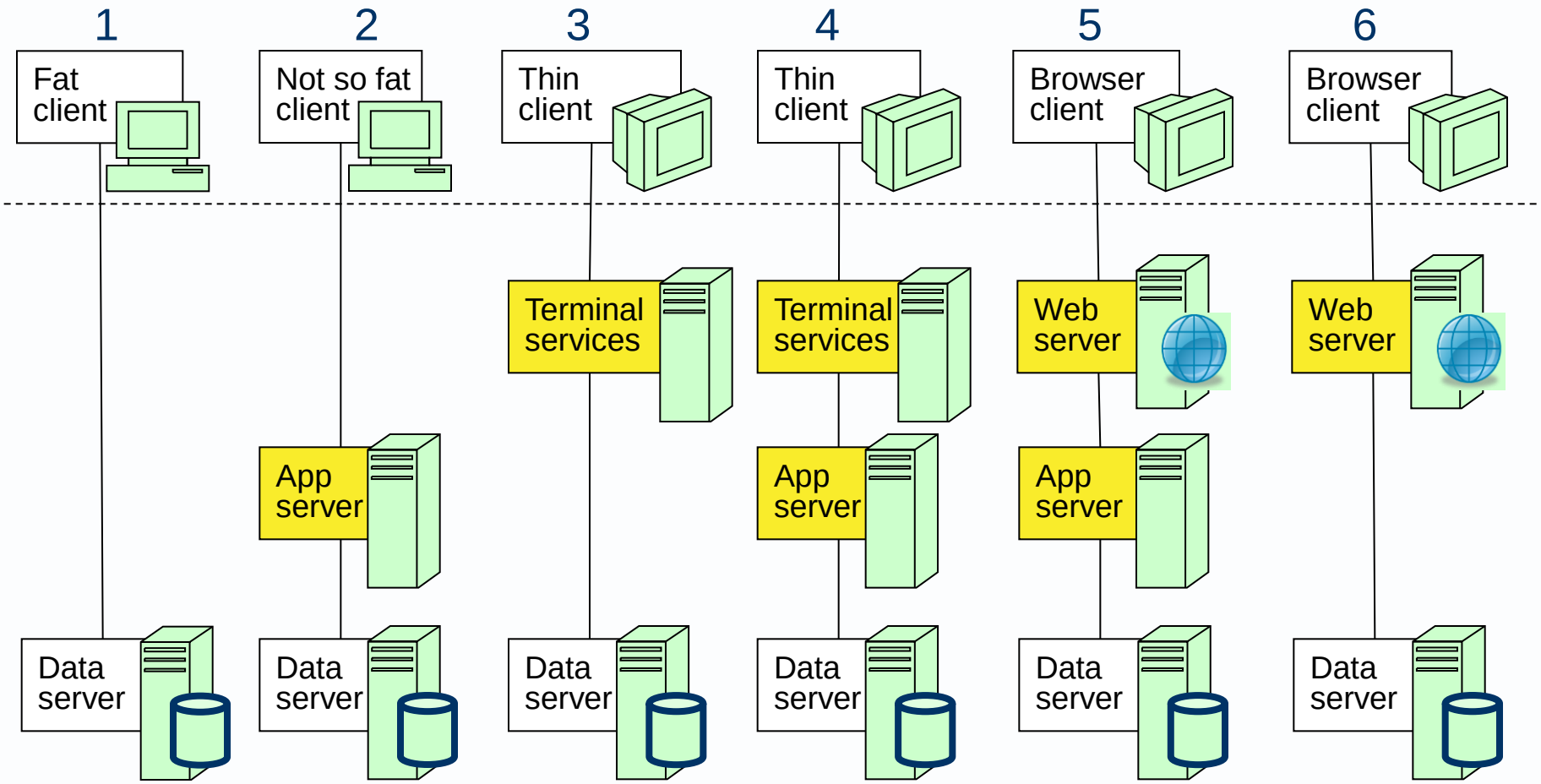
A data server is

- a computer program that provides database services
- a computer dedicated to running the DBMS and related software and maintaining databases
- usually a multiprocessor computer connected to a disc or via a high-speed channel to a RAID disk array or a SAN.

- A DBMS provides:
  - A data modeling language:
    - to define the schema of each database
  - Data storage structure (fields, records and files):
    - optimized to deal with very large amounts of data stored on a permanent data storage device (which implies very slow access compared to volatile main memory).
  - A database query language and report writer:
    - to allow users to interactively interrogate the database, analyse its data and update it according to the users privileges on data. It also controls the security of the database by preventing unauthorised users from viewing or updating the database. User access to the entire database schema or subschemas is limited by passwords.
  - A transaction commit/rollback mechanism:
    - ideally guarantees ACID properties, ensures data integrity despite concurrent user accesses (concurrency control), and faults (fault tolerance).

Define the top-to-bottom server stack

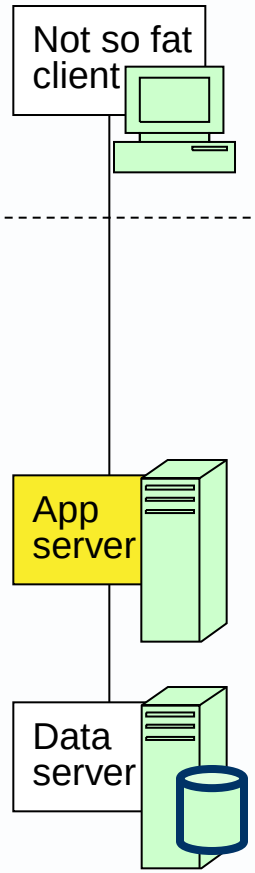
- Intermediate platform nodes should have a defined role, be logically isolated from each other, occupy a specific tier



1 Fat client + data server (no intermediate servers)

- OK for departmental apps

2 Not so fat client + app server + data server

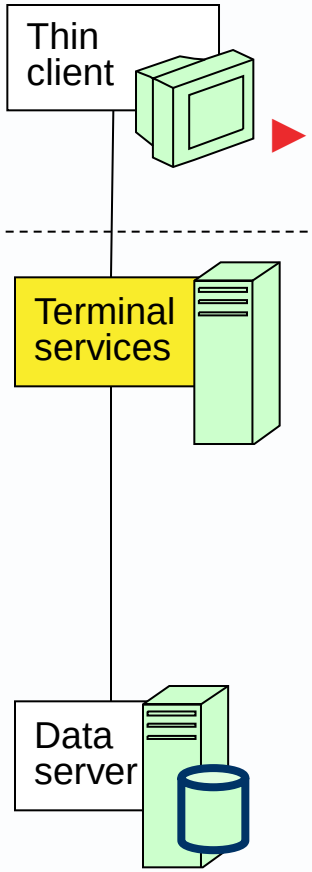


- Client could be a PC GUI, a web server, another app server, whatever
- Client can do whatever it wants with data returned

- The app server
  - provides client apps with automated services
  - provides any kind of data, including display markup
  - makes services accessible through protocols, possibly HTTP, but usually a component API, such as the EJB component model found on J2EE application servers.
- The app server also manages its own resources.
  - security
  - transaction processing
  - resource pooling
  - messaging (with various qualities of service: reliability, security, non-repudiation)
  - may use scalability and fault-tolerance techniques.

3 Thin client + terminal services + data server

- Hosted desktops
- Virtual desktop infrastructure (VDI)



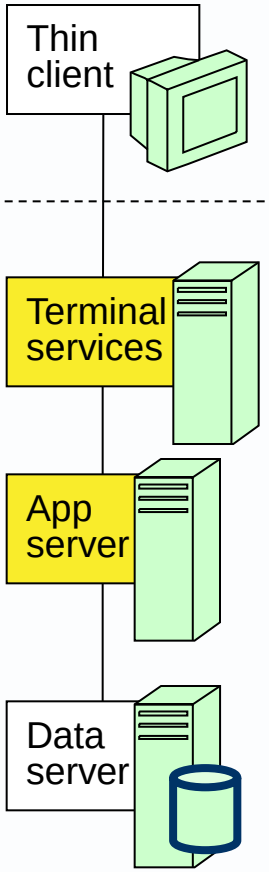
- Relocating desktop computing to the data centre offers flexibility

---

- Access from any PC in the organisation
- Remote access from home PC, internet café etc
- Business continuity
  - The DR facility can easily provide both server and desktop functionality.
  - Users continue working from home,
- Desktop can bought as a service from a hosting supplier
- Fewer PC/laptops need to be issued?

4 Thin client + terminal services + app server

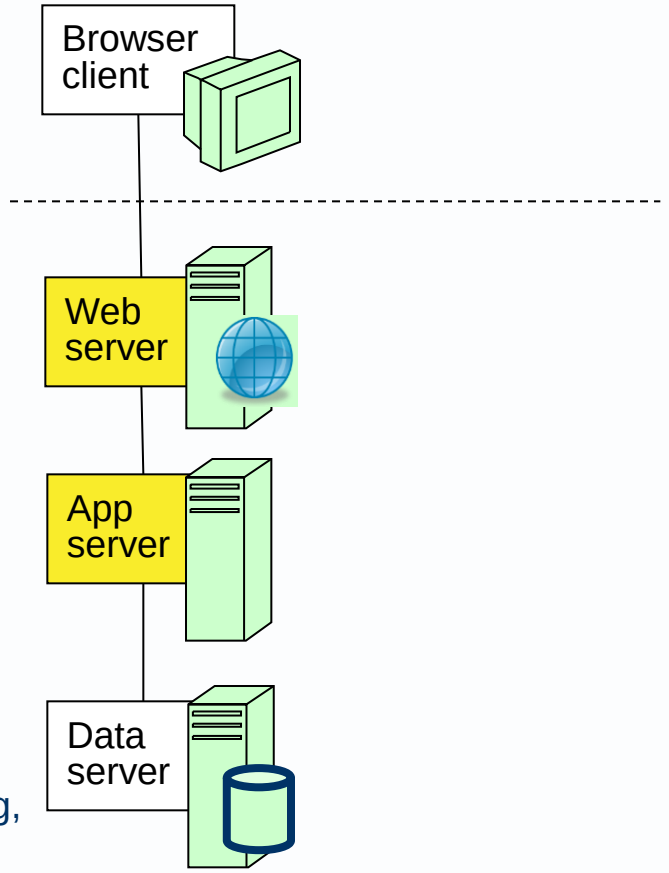
- Terminal services + hosted client app
- Reusable server app



- Combines desktop relocation with multi-tier architecture
  - Advantages of desktop virtualization
    - UI possibilities of fat clients
    - Flexible software distribution
  - Fat clients for power users
  - New presentation channels
    - Third party client
    - Web application

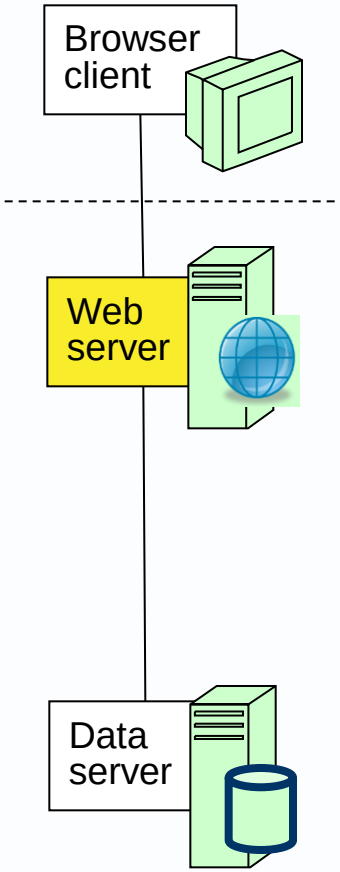
5 Classic browser + web server + app server + data server

- Web Server
  - receives an HTTP request from a client
  - delegates to a suitable server-side program
    - CGI scripts, JSPs (JavaServer Pages), Servlets,
    - ASPs (Active Server Pages), server-side JavaScripts
  - provides an environment in which the server-side program can execute and pass back the generated responses.
  - responds to client with an HTTP response
    - Usually an HTML page or image for viewing in a Web browser.
    - Could be static content or dynamic (generated) content.
    - Could be any type of file
    - Could redirect response to another server?
- Does not have to do anything else
  - no transaction processing, database connectivity, messaging
- But is likely to offer
  - Server-side programming language support.
    - perl, python, and PHP. Container for Java code (Tomcat).
  - Security authentication schemes.
    - Popular modules inc. mod_access, mod_auth, and mod_digest.
  - Strategies for fault tolerance and scalability, load balancing, caching, and clustering



6 Browser + web server + data server

- Not so strong reasons to retain an app server?
  - To deploy the 'business logic'? This logic can sit on other servers
  - To enable the business logic to be written in an OOP? This can lead to needless object-relational mapping complexities.
  - To ease testing? Adding a tier can make this harder.
  - To ease deployment or managed operations? Ditto
  - To increase security? How will adding an extra tier do this?
  - To improve response time? Business logic is usually swamped by infrastructure (load balancing, transaction management, auditing, logging, and database I/O)
  - To port code between databases? "We did once port SQL between Oracle and SQLServer, and it turned out to be quite easy."
- Stronger reasons to retain the app server
  - To help application developers
  - To handle high throughput: if scaling up the database is too costly.
  - To anticipate high scalability: if the growth numbers are convincing and scaling up the database is too costly
  - To deploy non-database processing business logic.



Label nodes with roles

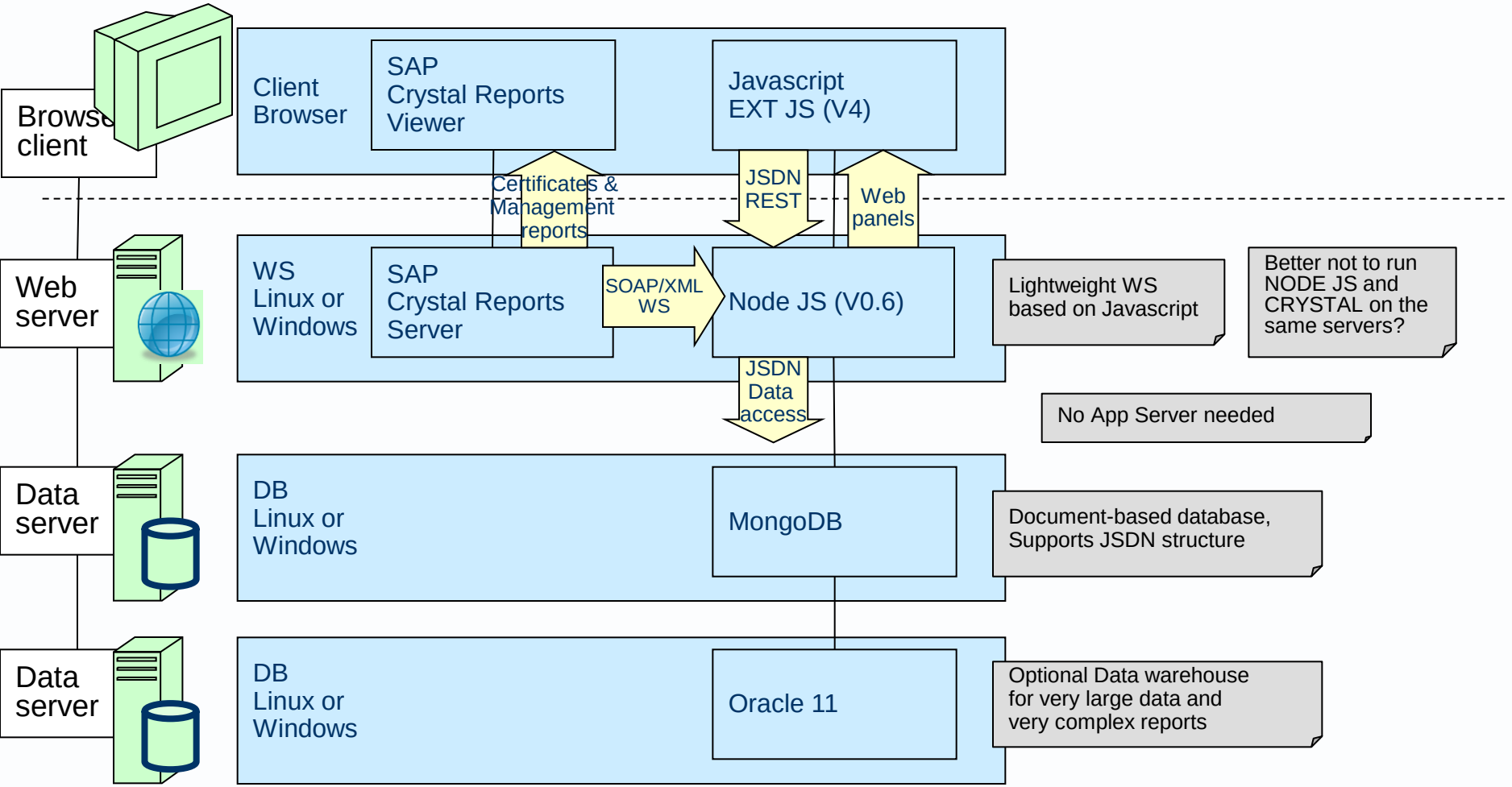
- Having divided the server stack into levels
- Clarify and label the role of each server

- Possible abbreviations
  - Client = End user device
  - FS= File server,
  - WS = Web Server,
  - APP = App server,
  - DB = Database server
  - FW = Firewall
  - LB = Load balancer
  - Mainframe = MF
  - Monitor

Define node operating systems

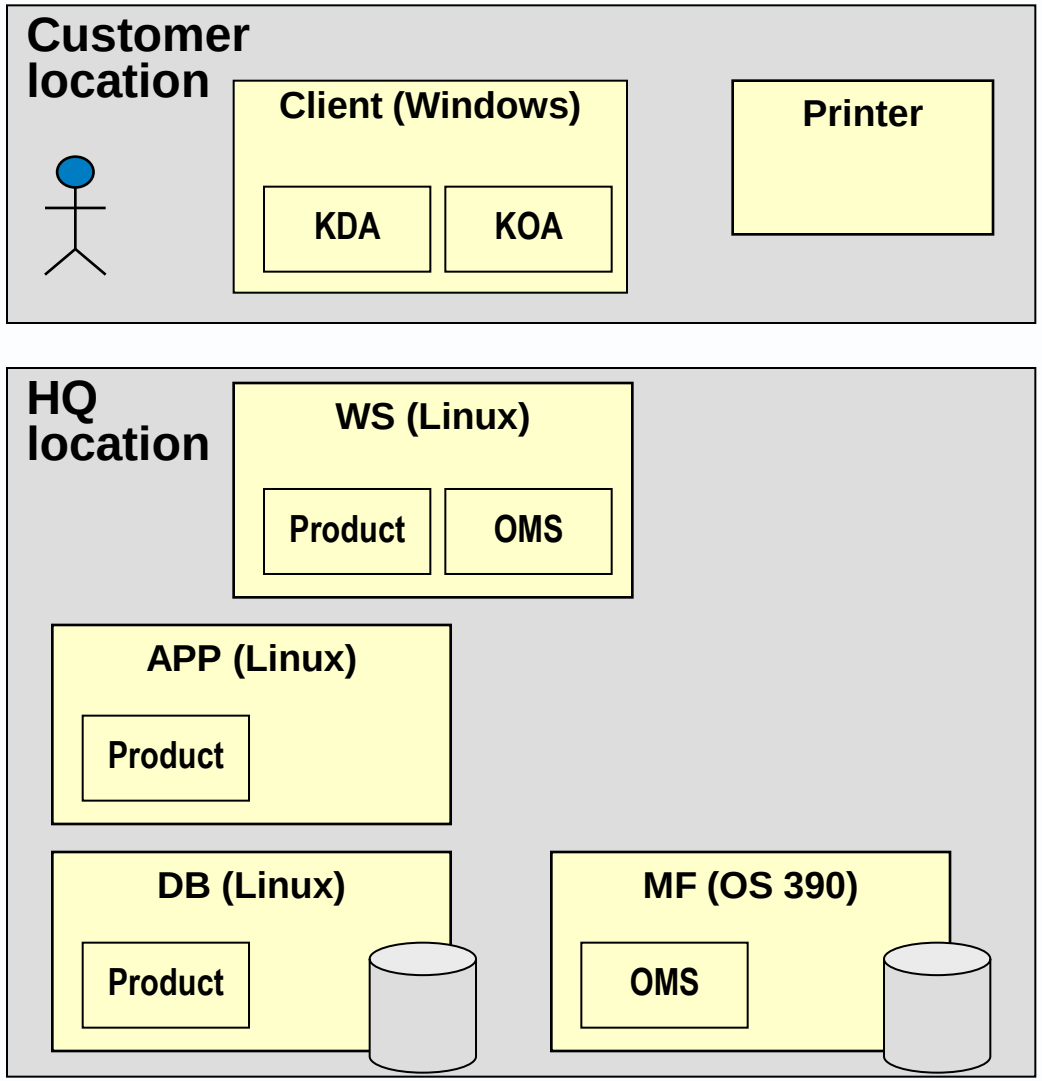
Avancier

A recent architecture specification



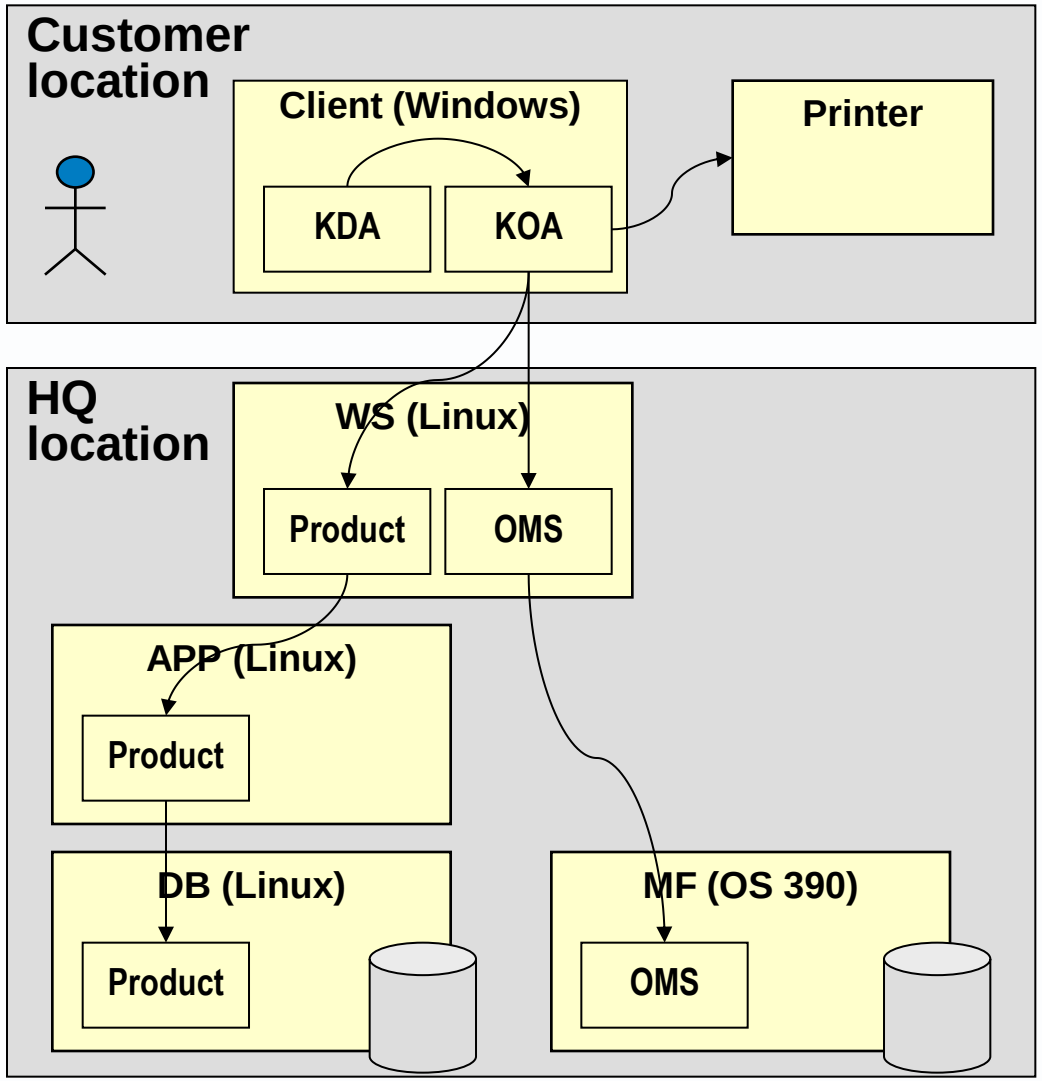
4 Map software to platform

- Assign each application software component to a platform node, or an execution environment within a platform node
- An **e**xecution **e**nvironment is a type or part of a node that represents a particular execution platform, such as an operating system or a database management system.



Define each inter-component connection

- Here, the arrow direction is
  - source -> target
  - client -> server
  - sender -> receiver
- Annotations can show additional information such as
  - CMax (Maximum Connections)
  - (Tout: connection time out)

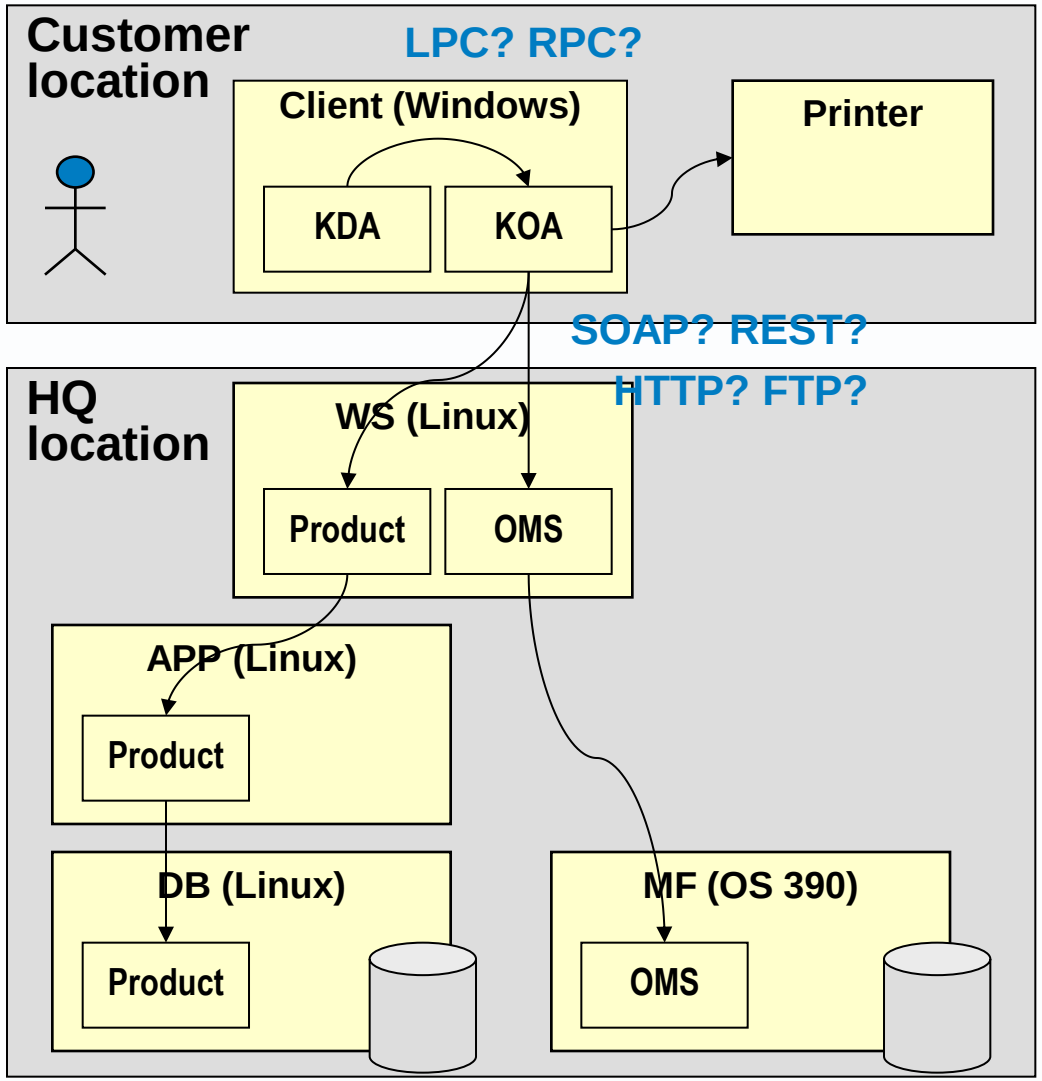


Define the protocol used to make a connection

- Connections are generally annotated with an application layer protocol such as http or ftp

Who decides which application layer protocol? And how?

Who decides which interoperation style? And how?

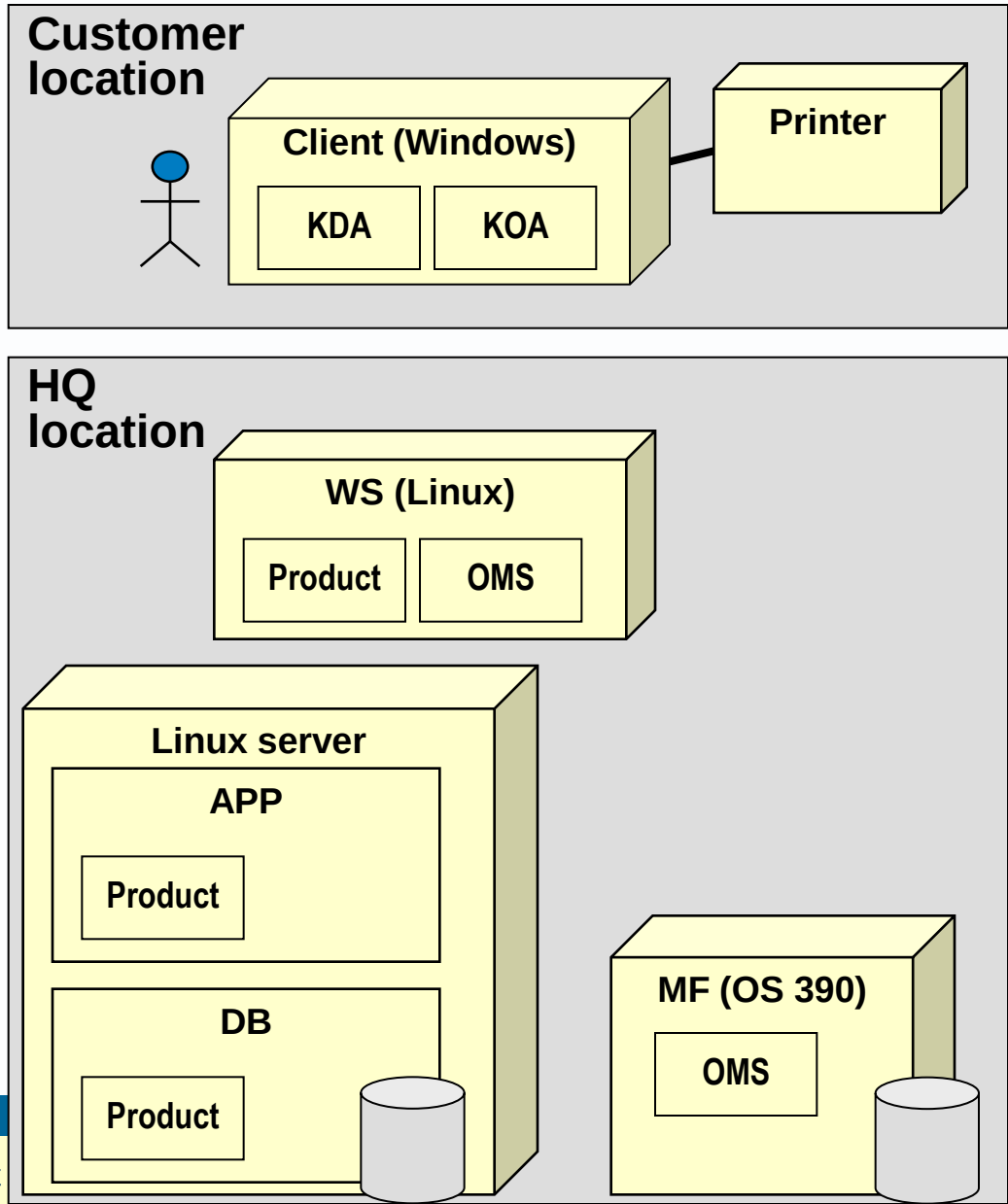


Define the port used for a connection

- The application layer protocol suggests the connection purpose and the ports it will likely be using.
- However, showing ports helps the guys who need to manage the firewalls.

5. Map logical nodes to physical nodes

- A physical node is a computer or other computing device such as a printer.
- Each has a unique network address, sometimes called a *Data Link Control (DLC) address* or *Media Access Control (MAC) address*.
- In this example, let us deploy the APP and DB servers on one computer



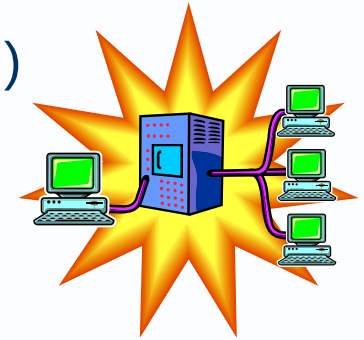
Size the physical node resources needed

- General sizing (2012) of processor and memory

Platform tier	Character	Processor CPU	Memory RAM
Web Server	memory intensive	1	Large
APP server	processor intensive	4	Medium
DB server	memory and processor intensive	4	Large

6 Define the network

- Use data volume and frequency numbers to calculate the network bandwidth
- Consider network scope (PAN, LAN, MAN, WAN)
- Required network service layer (1, 2 or 3)
- Internal network devices (switches and routers)



7 Refine to handle NFRs

- Iterate through the whole design to make sure all NFRs are considered and met

- Design for throughput and response time
- Design for availability
- Design for security (firewalls etc.)
- Design for serviceability
- Design for recoverability

WS: Design for throughput, availability and response time

- A web site may have to handle thousands of users at once.
- First, a load balancer shares work between parallel web servers

- Then, the web site cache can be important for performance
- Make the web-tier do as much as it can e.g. serve static content quickly.
- Having spent ages getting the HTML and images for a given URL from the app tier, cache it as static information, so you can serve a request for the same URL much faster.
- The web tier cache is very fast in comparison to making the app do some processing
- Without the cache you might need many app servers to serve the same number of users.

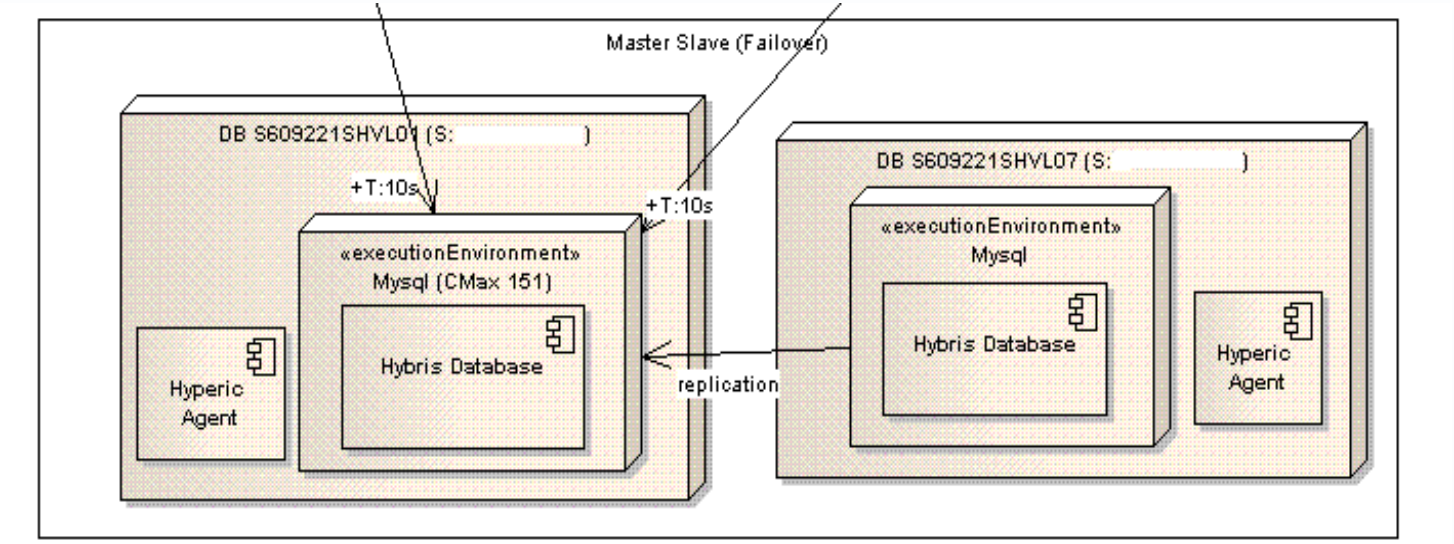
APP: Design for throughput and availability

- Parallel app servers - configured in a cluster for resilience.

- You may need to synchronise cached objects on each server.
- No problem if the occasional update is missed
- The broadcast feature of UDP means that scaling out can happen easier, additional servers brought online without explicit re-configuration of the existing server.

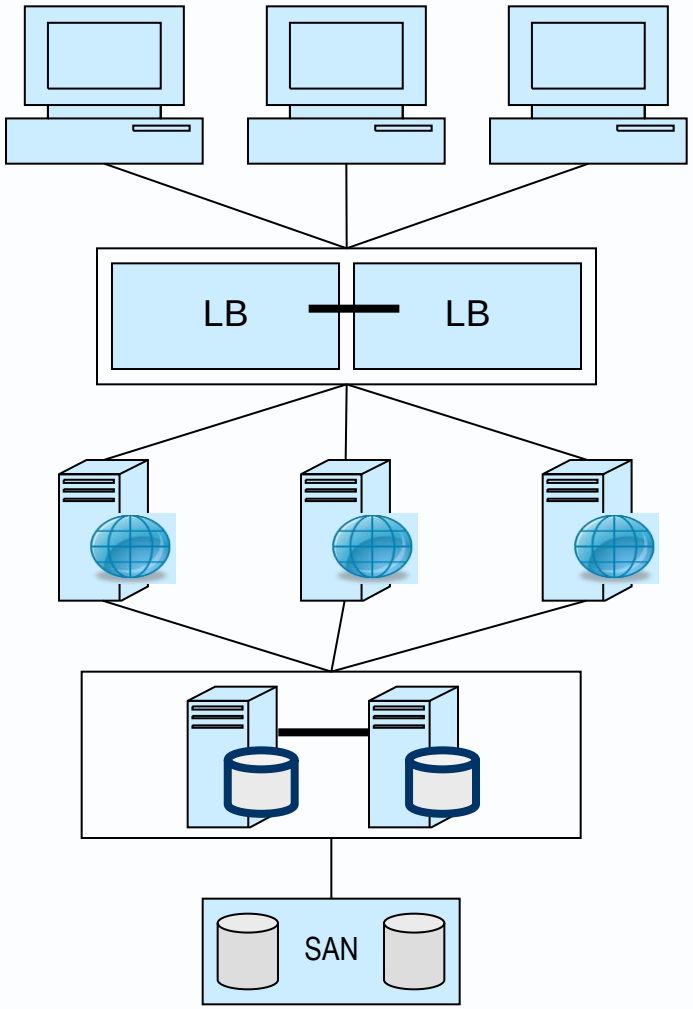
DB: Design for availability

- E.g. active-passive dual data server
- Dual data servers in a master-slave configuration
- The wider system only talks to the master.
- The slave makes regular copies of the data in the master.
- In the event of a failure of the master the system can be re-configured quickly to point to the slave which will then assume the role of master.



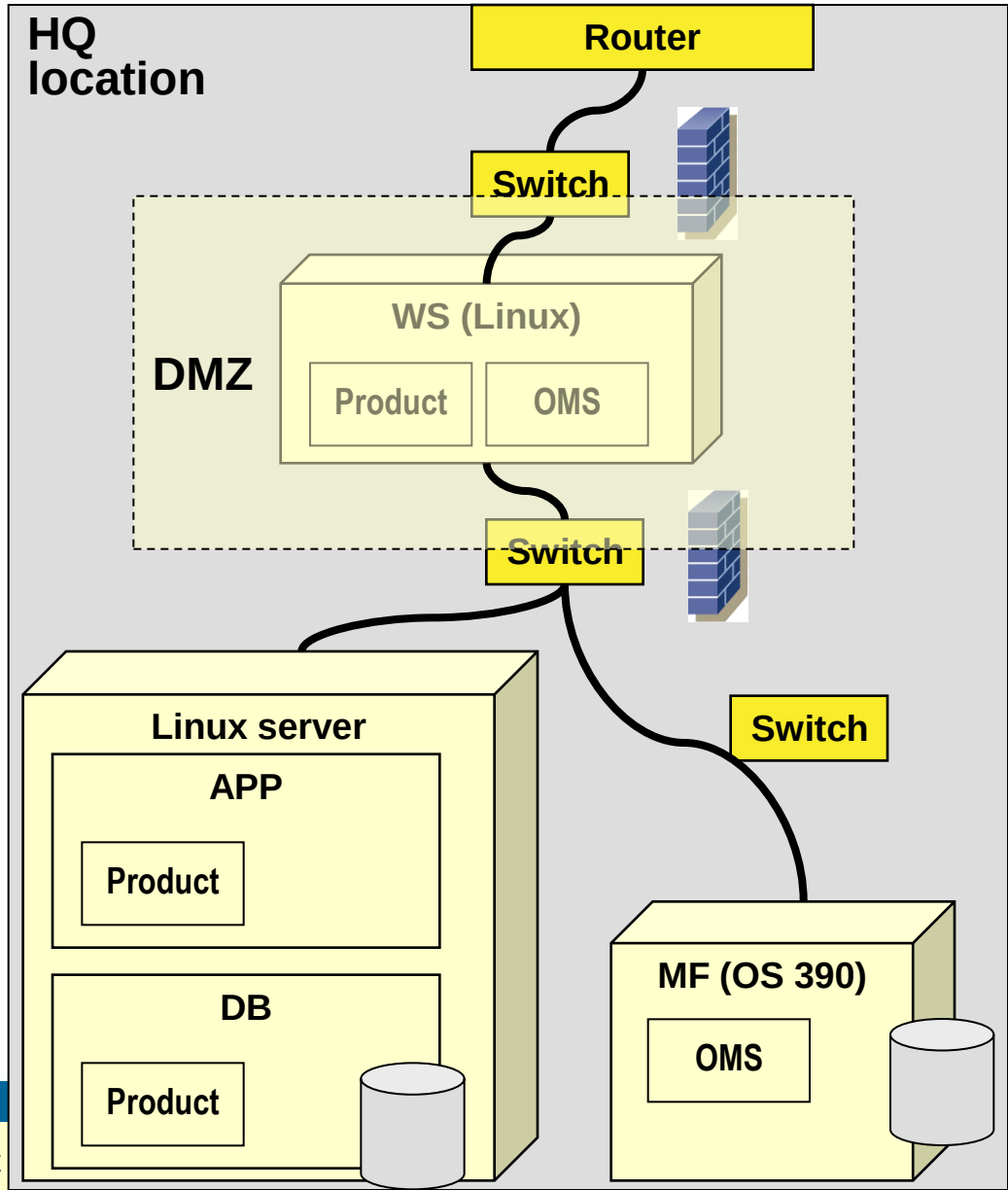
Design for availability

- Parallel web servers
- Parallel app servers
- Dual data servers



Design for security (firewalls etc.)

- Adjacent tiers separated by a firewall
- The firewalls implement strict packet filtering for both inward connection (ingres) and outward connections (egres)
- This ensures communication between tiers can only happen on the designated ports in support of the whole application.
- E.g. if a hacker managed to get onto the database server, they cannot establish a connection to the internet etc.

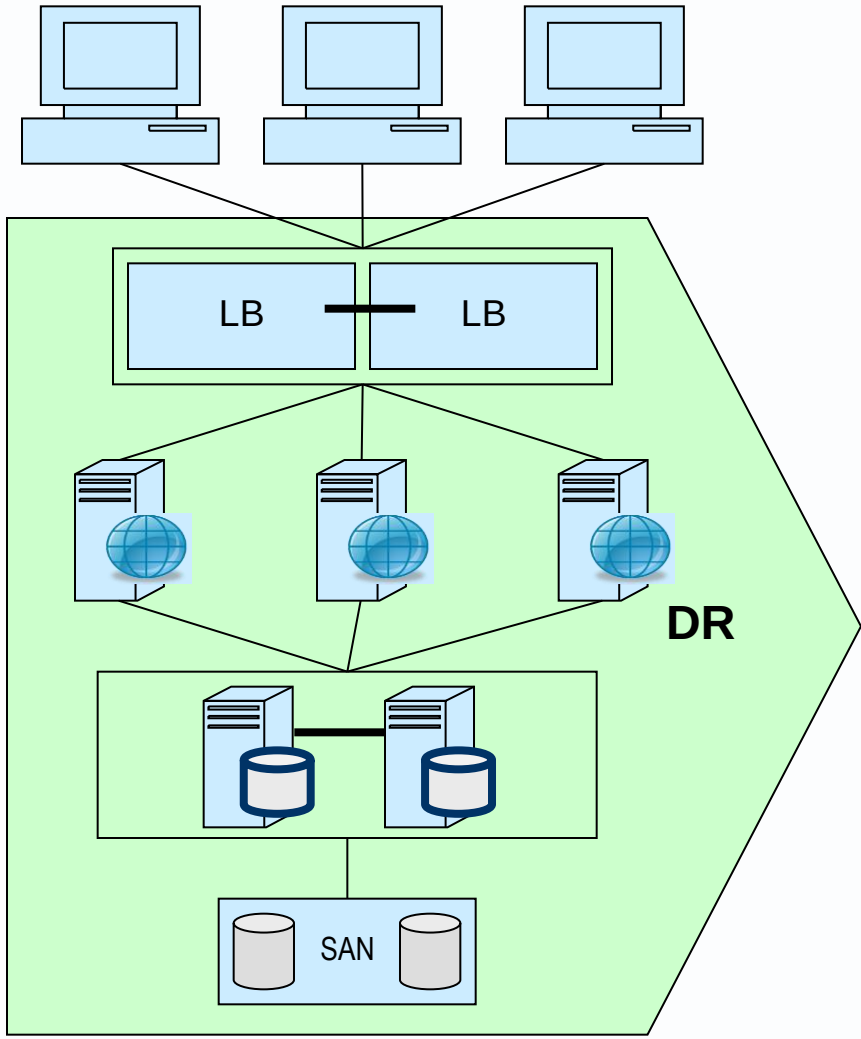


Design for serviceability

- Deploy monitoring software to track resource utilisation of all of the servers.

Design for recoverability

- Duplicate resources at a remote disaster recovery site
- The entire configuration is replicated.
- If all important state information is stored in the database, then only that tier needs regular replication across data centres.
- Also, all servers should be "backed up" off-site on a daily basis.



8 Define non-production environments

Avancier

- Remember s/w licence costs can rise with each real CPU and each virtual machine or LPAR

Environment type	Purpose	Physical platform	Hosted at Location	Contains Application Components	Contains Technology Components
Prototyping	To test/demonstrate a specific technology or design concept	1			
Development	To enable developers to write code	1			
System test	To enable system testers to the product	1			
Integration test	To test how the system integrates with others	2			
Performance test	To test how the system performs when fully loaded	2			
Data migration	To enable cleansing and migration of data	3			
User acceptance test	To enable user representatives to test to the product	4			
Production	To enable live operation of the system(s)	4			
Production support	To enable fault replication and investigation, and minor changes	5			

9 Govern deployment and transition into operations

- Monitor the progress of the deployment and refine the infrastructure design as need be

Design target infrastructure solution - END

- 1 Identify precursors (requirements and context)
- 2 Establish baseline opportunities and constraints
- 3 Define logical platform nodes
- 4 Map software to platform nodes
- 5 Map logical nodes to physical nodes
- 6 Define the network
- 7 Refine to handle NFRs
- 8 Define non-production environments
- 9 Govern deployment and transition into operations

A process for defining the technologies that will support and run one or more applications.

A process that progresses through stages from a logical application-information view through progressively more physical views up to a hardware configuration diagram.