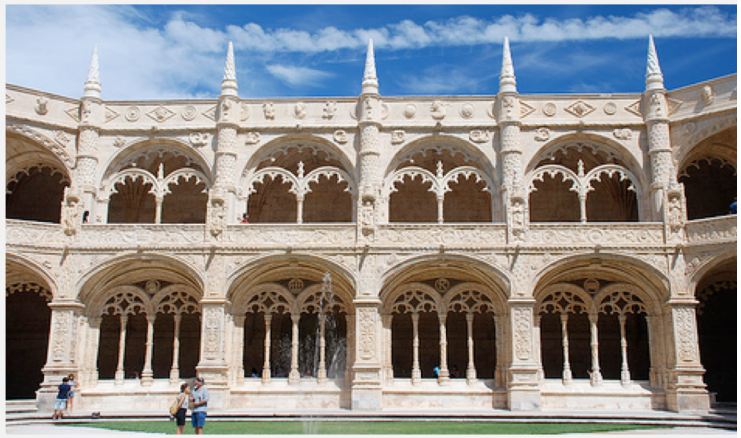
11 ARCHITECTUAL STYLES

course “software requirements and architecture”

Oct 2021

JM Fernandes

architectural style



- An architectural style in software is analogous to an architectural style in building (e.g., Manuelino).
- An architectural style consists of:
  - a set of component types (e.g., process, procedure) that perform some function at runtime
  - a topological layout of the components showing their runtime relationships
  - a set of semantic constraints
  - a set of connectors (e.g., data streams, sockets) that mediate communication among components

A system that conforms to a given style must use those types, which restricts the design space.

constraints

- Constraints can act as guide rails that point a system where you want it to go.
- One can think of a style as a prefabricated set of constraints that can be reused.
- The consistency brought about by the constraints of the style can encourage clean evolution of the system, which can make maintenance easier.
- Communication among developers is improved since the simple name of a style conveys design intent.

platonic vs. embodied

- A platonic architectural style is an idealization.
- One finds it in books and only rarely in source code.
- An embodied architectural style exists in real systems
- It often violates the strict constraints found in platonic styles.
- Example: the platonic client-server style requires that servers be unaware of clients.
- One may find embodied versions of the style where servers occasionally push data to the clients.
- Depending on how this is implemented, it may result in a server that depends on the clients.

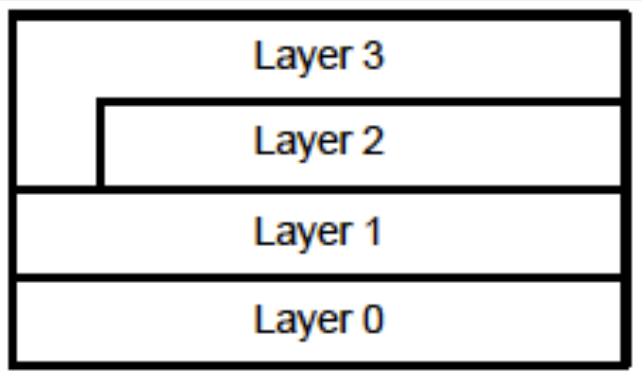
patterns vs. styles

- Design patterns are at a smaller scale than architectural styles.
- Patterns can appear anywhere in your design, and multiple patterns could appear in the same design.
- A system usually has a single dominant architectural style.
- If a system has a client-server architectural style, one expects to see client and server components in the top-level design views.
- The system could also employ architectural patterns, such as the REST pattern.

catalog of styles

	Viewtype	Elements & Relations	Constraints / guide rails	Qualities Promoted
Layered	Module	Layers, uses relationship, callback channels	Can only use adjacent lower layers	Modifiabilty, portability, reusability
Big Ball of Mud	Module	None	None	None, but many inhibited
Pipe-and-Filter	Runtime	Pipe connector, filter component, read & write ports	Independent filters, incremental processing	Reconfigurability (modifiability), reusability
Batch-Sequential	Runtime	Stages (steps), jobs (batches)	Independent stages, non-incremental processing	Reusability, modifiability
Model-Centered (Shared Data)	Runtime	Model, view, and controller components; update and notify ports	Views and controllers interact only via the model	Modifiability, extensibility, concurrency
Publish-Subscribe	Runtime	Publish and subscribe ports, event bus connector	Event producers and consumers are oblivious	Maintainability, evolvability
Client-Server & N-Tier	Runtime	Client and server components, request-reply connectors	Asymmetrical relationship, server independence	Maintainability, evolvability, legacy integration
Peer-to-Peer	Runtime	Peer components, request-reply connectors	Egalitarian peer relationship, all nodes clients and servers	Availability, resiliency, scalability, extensibility
Map-Reduce	Runtime & allocation	Master, map, and reduce workers; local and global filesystem connectors	Divisible dataset amenable to map & reduce functions, allocation topology	Scalability, performance, availability
Mirrored, Farm, & Rack	Allocation	Varies	Varies	Varies: Performance, availability

layered style



- The essential element of the layered style is a layer.
- The essential relationship is a *uses* relationship, a specialization of a dependency relationship.
- The layered style consists of a stack of layers.
- Each layer acts as a virtual machine for the layers above it and their ordering forms a DAG.
- In a simple layered style, a layer can use only the layer directly beneath it.
- Quality attributes promoted: modifiability, portability, and reusability.

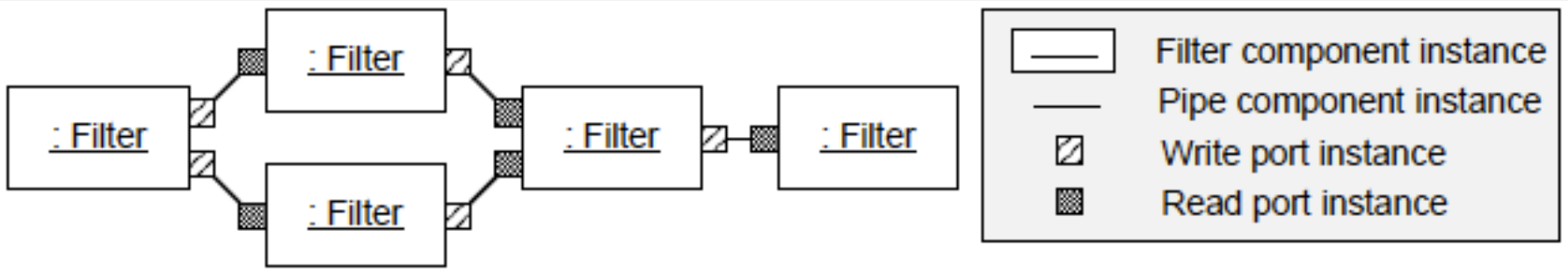
layered style

- The layered style can vary greatly from its platonic form to its embodied form.
- In practice a layered style may violate its constraint
- You may see skipping of layers or lower layers using upper layers.
- Lower layers can safely communicate with higher layers if they use a callback mechanism.
- “Lasagna code” refers to a program structure that follows the layered style.

big ball of mud style

- It has no evident structure.
- Promiscuous sharing of information is typical, to the extent that data structures become effectively global.
- Repairs and maintenance are expedient and resemble crude patches rather than elegant refactorings.
- No effort is made to enforce any conceptual integrity or consistency (“spaghetti code”).
- Such systems have poor maintainability and extensibility.
- This style is a good enough strategy of engineering.

pipe & filter style

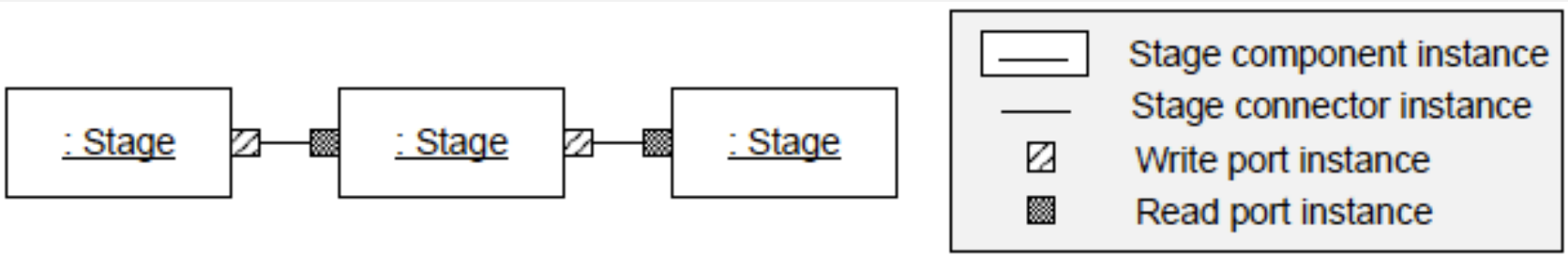


- Data flows through pipes to filters that work on the data.
- The pipe-and-filter network is continually and incrementally processing data.
- The pipe-and-filter style consist of four elements: pipes, filters, read ports, and write ports.
- A filter reads input from the input ports, does some processing, and writes output to the output ports.
- It repeats this until it is time to stop.

pipe & filter style

- Filters can enrich, refine or transform the data
- Each filter applies a function to its input.
- Pipes must only transport the data in one direction, without changes, and in order.
- Loops in the network are rare and often prohibited.
- Filters may not interact with each other, even indirectly, except through pipes.
- Filters cannot share state with each other
- A filter incrementally reads the input and, as it processes that input, incrementally write its output
- `cat "f.txt" | grep "^Braga" | cut -f 2-`

batch-sequential style



- Data flows from stage to stage.
- Each stage completes all of its processing before it writes its output.
- Data can flow between stages in a stream but is more often written to a file on disk.
- It has similar constraints to the pipe-and-filter style.
- Each stage is similarly independent.
- A stage depends on the data that it takes in.
- Stages do not interact with each other except through the input and output streams or files.

batch-sequential vs. pipe & filter

- Both decompose the task into a fixed sequence of computations, interacting only through data.

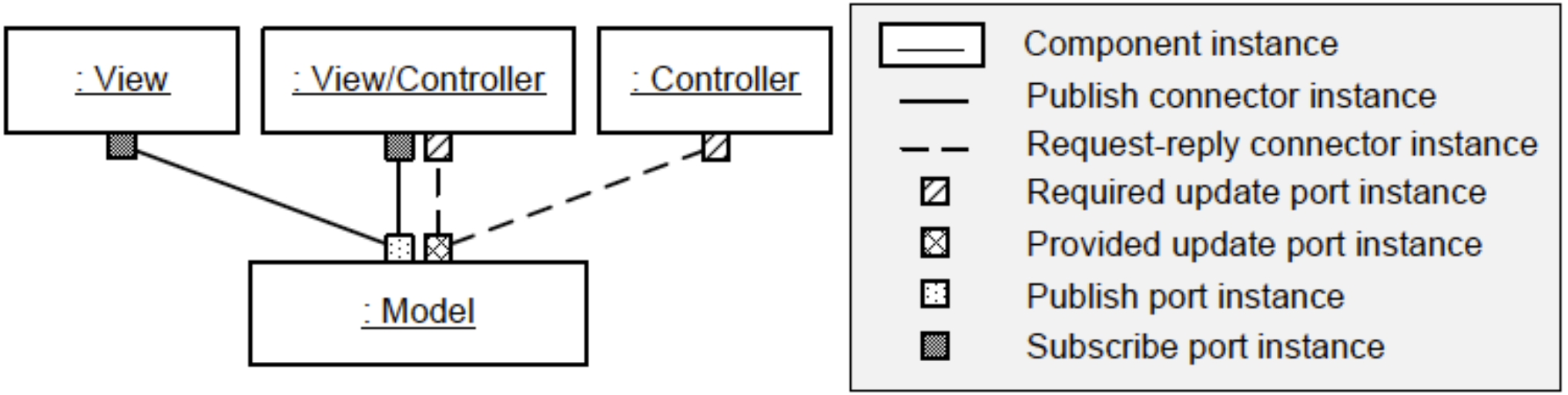
Batch sequential

- Coarse-grained
- High latency
- External access to input
- No concurrency
- No interaction

Pipe & filter

- Fine-grained
- Results start processing
- Localized input
- Concurrency possible
- Interaction possible

model-centered style

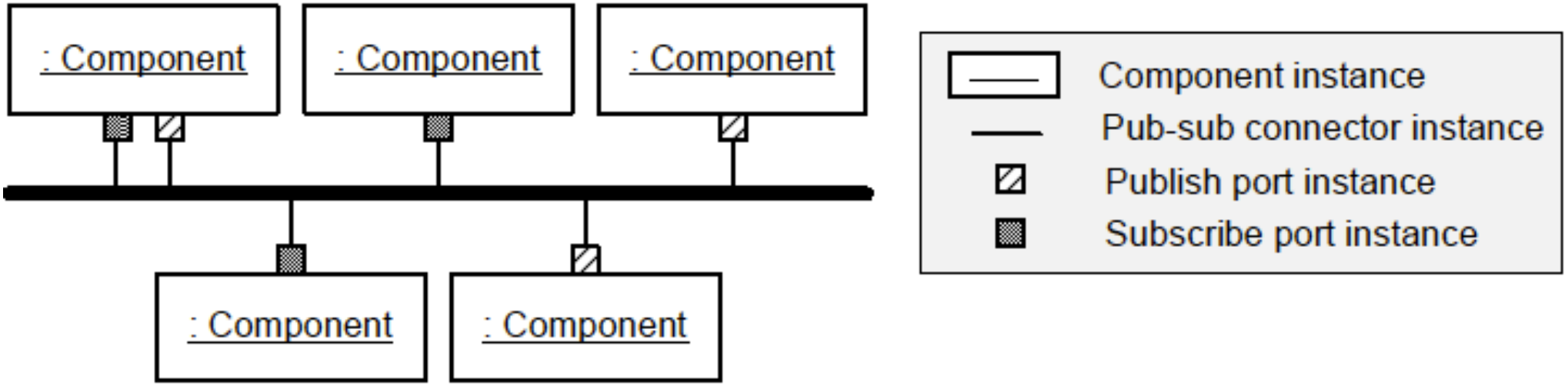


- Independent components interact with a central model (data store) instead of with each other.
- Every model-centered system has a model component and several view, controller, or view-controller components.
- This style is related to several design patterns, including the document-view, MVC, and observer.

model-centered style

- Views and controllers depend only on the model, not on each other.
- Modifiability is enhanced because the producer and consumer of information are decoupled.
- The system is extensible since unanticipated views and controllers are easily added.
- It can be easier to manage and persist state, since it is centralized in the model component.
- Concurrency may be promoted since views and controllers can run in their own threads or processes.
- This style is useful when one does not know the future configuration of the system.

publish-subscribe style

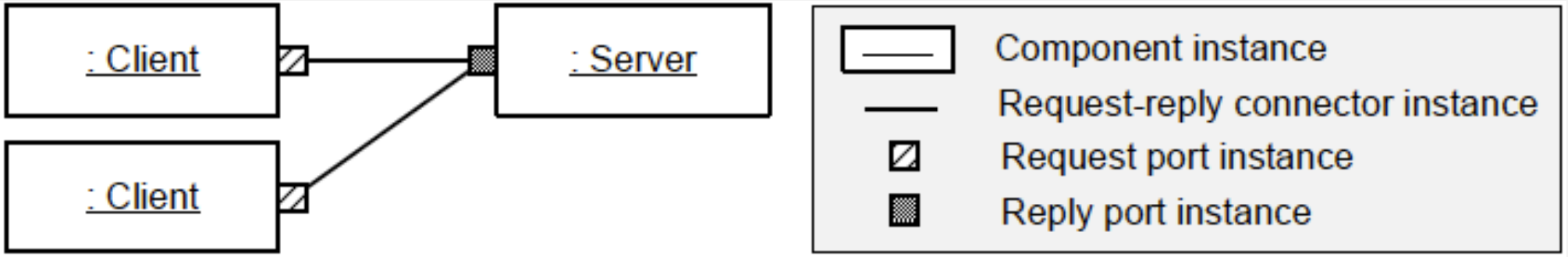


- Independent components publish events and subscribe to them.
- A publishing component is ignorant of the main reason why an event is published.
- A subscribing component does not know why or who published the event.

publish-subscribe style

- This style defines publish and subscribe ports, and one connector (an event bus connector).
- Any kind of component can publish (subscribe) events as long as it uses a publish (subscribe) port.
- One component can publish an event and many components can subscribe to it.
- The event bus connector delivers events:
  - publishers trust that events are delivered to subscribers
  - subscribers trust that they receive events they subscribe to
- Producers and consumers of events are decoupled.
- The system is more maintainable and evolvable.

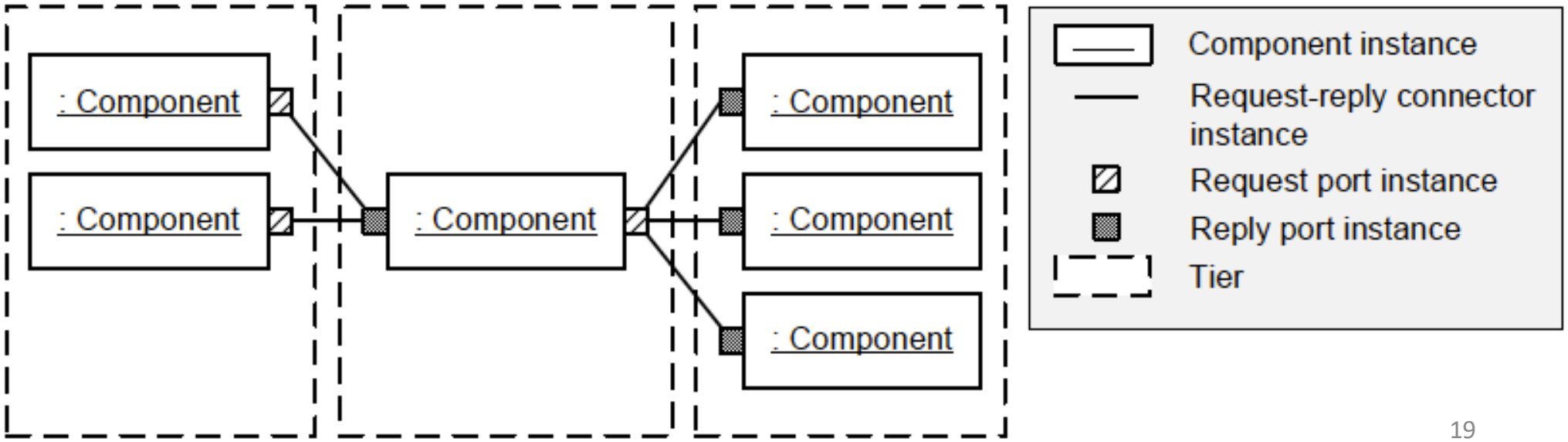
client-server style & N-tier



- Clients synchronously request services from servers.
- The client can request that the server do the work.
- Communication is initiated by clients, not the server
- The server does not know the identity of the client until it is contacted.
- Clients must either know the identity of the server or know how to look up the server.

client-server style & N-tier

- The client-server style has several variation points:
  - connectors may be synchronous or asynchronous
  - there may be limits on the number of clients or servers
  - connections may be stateless or stateful (i.e., sessions)
  - the system topology can be static or dynamic
- Another variant is the N-tier style



client-server style & N-tier

- It uses two or more instances of the client-server style to form a series of tiers
- Requests must flow in a single direction
- A common case is a 3-tier system where:
  - a user interface tier acts as a client for the business logic tier server
  - business logic tier server acts as a client for the persistence tier server
- Tiers have exclusive functional responsibilities.
- The user interface tier is exclusively responsible for user interaction
- The persistence tier exclusively saves persistent data.

client-server style & N-tier

Presentation tier

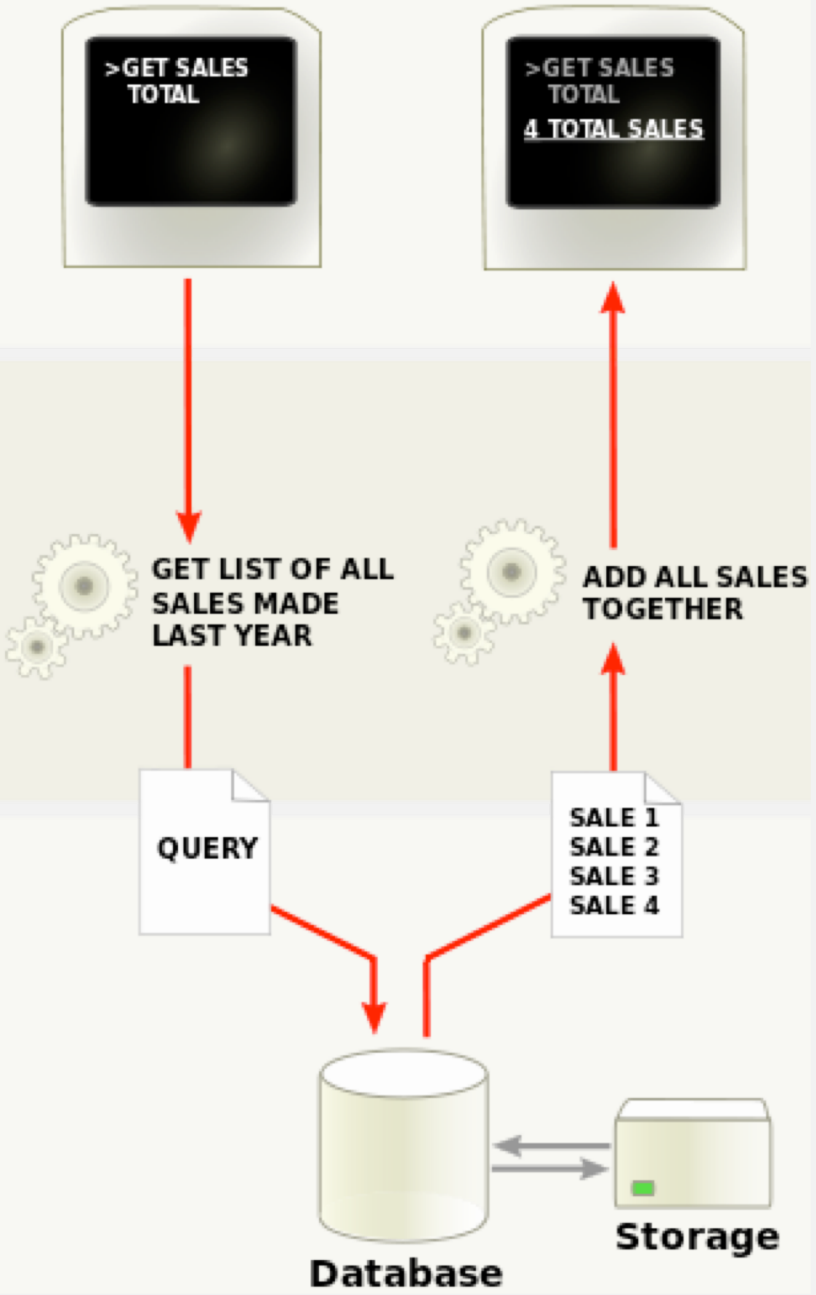
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



peer-to-peer style

- Nodes communicate with each other as peers.
- Hierarchical relationships are prohibited.
- Each node has the ability, but not obligation, to act both as a client and as a server.
- The result is a network of nodes operating as peers.
- A node can request or provide services to any node.
- The elements of the peer-to-peer style are similar to those of the client-server style.
- A peer-to-peer connector has identical roles on either end allowing both requests and responses.
- A peer-to-peer system is egalitarian where the client-server style is hierarchical.

map-reduce style

- This style is appropriate for processing large datasets (search engines and social networking sites)
- Simple programs (sorting or search) execute slowly on large datasets, if a single computer is used.
- This style enables the computation to be spread across multiple computers.
- As the number of computers used increases, the likelihood that one of them will fail also increases.
- This style enables recovery from such failures
- (more at details at the literature: Fairbanks; p.291-3)

mirrored, rack, and farm styles

- The previous architectural styles have been from the module and runtime viewtypes
- The styles from the allocation viewtype are more likely to be discussed by network engineers than software architects
- (more at details at the literature: Fairbanks; p.293-4)

further reading

- Fairbanks G; *Just-enough software architecture: A risk-driven approach,* Marshall & Brainerd, 2010 [chapter 14]