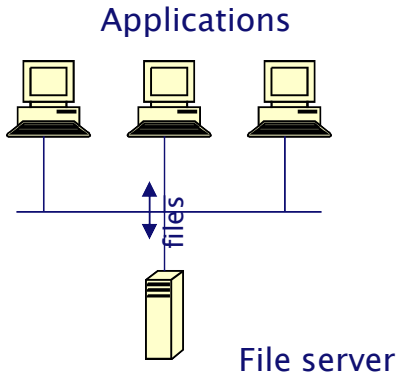
Client-Server style

- Motivation:
  - sharing some localized resource (e.g. file store, compute server)
  - protecting and managing content (e.g. a database)
  - delay binding, decrease dependencies (independent development)
  - generally: separation of concerns
- Vocabulary:
  - client, server (building blocks)
  - request, reply (interaction)
  - server discovery
- Rules:
  - Servers are passive
    - provides a service upon request from clients, does not know clients
    - handles data access, data integrity
  - Clients are active
    - initiate activity, *discover* servers
  - no connection among clients

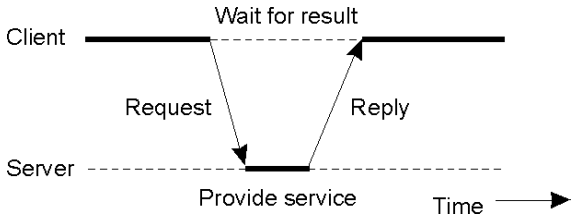
- Structure:



  - no, or limited, state on the server per client

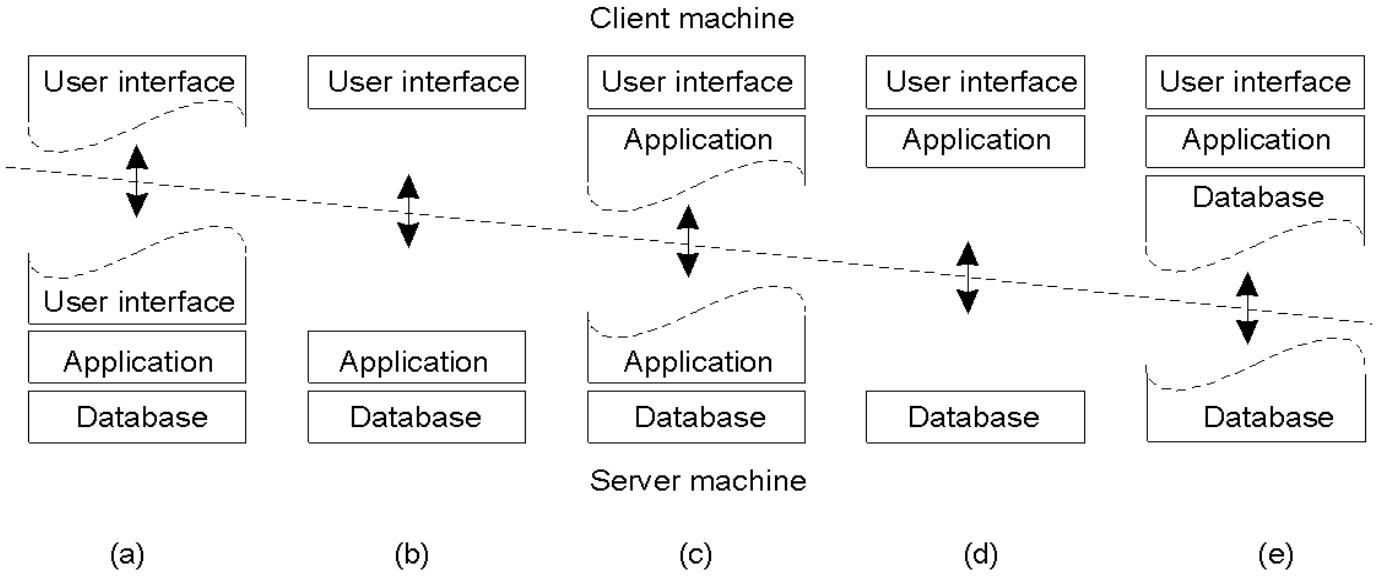
Client-Server style

- Typical behavior:
  - client finds server access pointthrough discovery
  - regular interaction:
  - possibly as part of a session

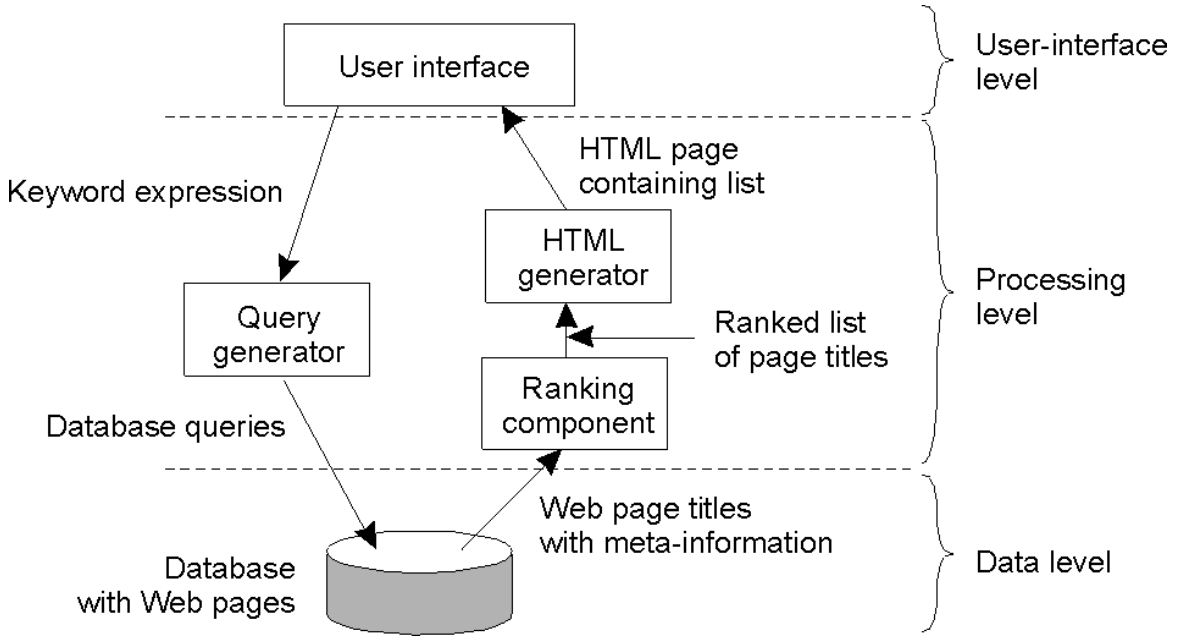


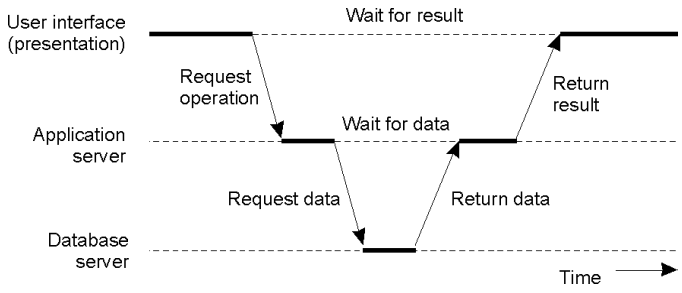
Application layers and C-S functionality

- 3-layer logical organization of data access
- cuts possible at many places: define tiers (physical layers)
  - (a,b) thin client – changing control
    - the user interface has small overhead; it typically runs on a machine that is a client to the application server
    - however, after connection, the user interface becomes a server for the application
  - (c,d) fat client
  - (e) file server



Combining layering and C/S in a distributed fashion





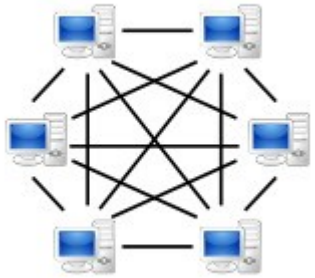
Development (logical): layering ("presentation, business and data tier (layer)")

Deployment: Client-Server interactions between the components

- also called: multi-tier client-server organization
- note: middle layer plays both client and server *roles*
  - that is actually what layering + request/reply is

Peer-to-peer style

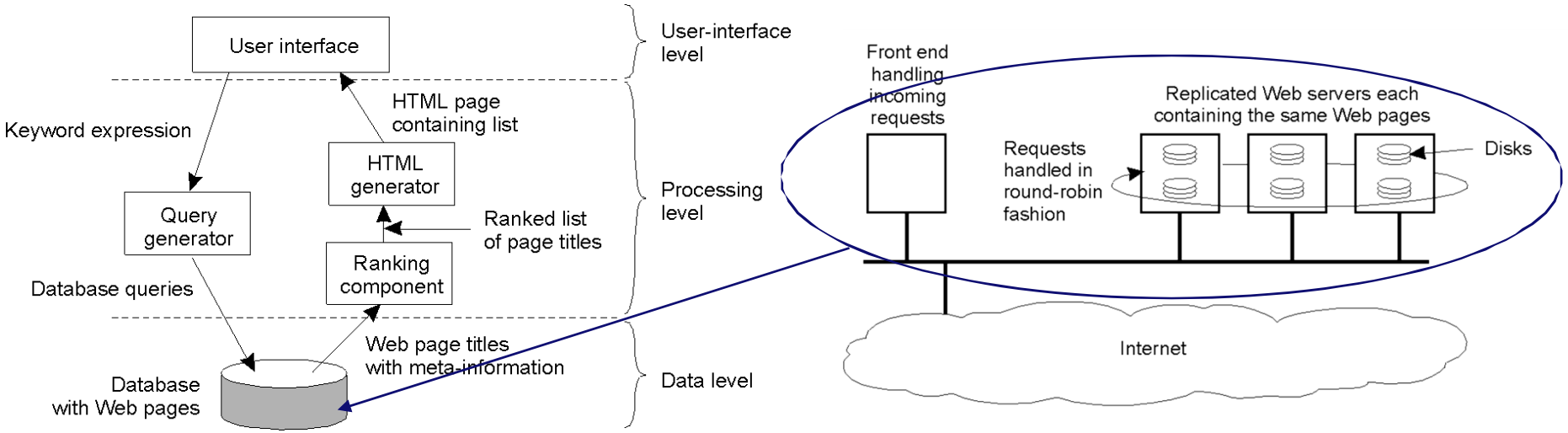
- Motivation:
  - sharing resources and content
  - cooperation in communities
  - symmetry in roles
  - increase concurrency
  - horizontal scalability
- Vocabulary:
  - peer, super peer, distributed hash table (building blocks)
  - community (of peers)
  - overlay, application level multicast, peer discovery
- Rules:
  - Peer:
    - symmetric in functionality, and contribution
    - can discover, reach and cooperate with all other peers, in principle
    - can be both passive and active
  - Super peer
    - contributes extra resources to the community (extra services, like directory, discovery)
- Structure::
  - overlay, on top of network technology (from deployment view)



- Typical behavior:
  - peer finds community access point
  - joins community
  - provides passively services to other peers
  - actively uses services from peers in the community
  - the collective services of peers result in new functionality
    - e.g. file sharing
- Example: Skype VOIP

Combine peer-to-peer with application layering

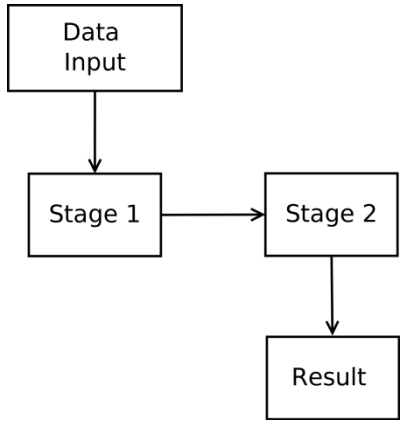
- Vertical distribution: map entire *layers* to machines
- Horizontal distribution:
  - divide a *layer* across a collection of symmetric *peers* (picture)
    - e.g. a distributed database, distribution of business logic
  - divide different functions inside a layer across (asymmetric) peers



Batch sequential style

- Motivation:
  - Processing in several stages
  - Monolithic component would be too complex
  - Reconfigurable system: stages inserted/removed/reordered easily
- Vocabulary:
  - Processing steps (stages), batch of data (single input set), sequential processing
- Rules:
  - Every stage is stand alone in processing. There is no shared state.
  - Only subsequent stages exchange data.
  - System handles input sets one by one. No stream of data. No concurrent work of stages on different data.
- Weak point
  - Not interactive and slow due to sequential processing

- Structure:



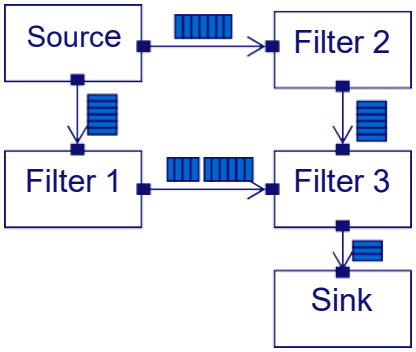
- Typical behavior:
  - Stages designed to be reusable (e.g. have configurable parameters)
  - Different stages can be on different processors, but their execution is still sequential.
- Metaphore
  - Assembly line that can start processing next product only after previous one has been completely processed

Examples: Compilers, classical ETL systems

Pipes & filters style

- Motivation:
  - Systems that work with streams of data (e.g. video processing multimedia systems)
  - Streams of data are naturally processed in several stages
  - Reconfigurable system: stages inserted/removed/reordered easily
- Vocabulary:
  - Pipes, filters, data stream, data source, data sink, ports, buffers, pipeline
  - Data stream is processed by filters
  - Filters are connected via pipes (explicit connectors) .
  - Forks and joins are allowed, but pipeline is sequence of filters from data source to sink.
- Rules:
  - Every filter is stand alone stage in processing. There is no shared state.
  - Pipes store data (state) and preserve or ordering (FIFO)
  - Only adjacent filters exchange data
- Weak points:
  - Difficult to share global data
  - Difficult to handle control (e.g. filter crashes)
  - Not convenient for interaction

- Structure:

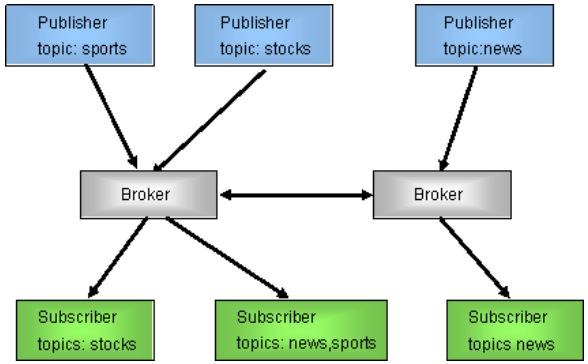


- Typical behavior:
  - Filters are designed to be reusable (e.g. have configurable parameters)
  - Filters can execute concurrently (on different processors) and asynchronously
  - Pipes have finite capacity; filters block on read and possibly on writes (otherwise data loss). Capacity usually chosen to prevent this
- Metaphore:
  - Stream of products being processed along assembly line
  - Example: Unix shell commands, GStreamer

Publish/subscribe style

- Motivation:
  - Decoupling data producer from data consumer
  - Sending data when it is available. Avoiding need to poll for data.
  - Allow multiple producers/consumers
  - Allow runtime changes of set of producers/consumers
- Vocabulary:
  - Publisher, subscriber, subscription, notification, topics, broker
  - Subscriber registers with publisher to receive notifications for chosen topic.
- Rules:
  - Every subscription relates a topic to 1 subscriber.
  - Publisher can have multiple subscribers for same topic.
  - Notification goes to all subscribers; however, subscribers may specify receive policies
  - When existent, broker decouples the publishers and subscribers

- Structure::



- Typical behavior:
  - Subscribers find brokers and send subscriptions for topics of interest
  - Publisher register with brokers, or are discovered
  - Notification may be only about events, or it can also contain data.
  - Broker can be omitted
- Metaphore:
  - Newspaper/magazine subscription
  - Observer design pattern is an example of publish/subscribe style

Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures, Chapters 3 and 5, The definition of REST, the WWW architecture*, 2000.

Paris Avgeriou, Uwe Zdun , *Architectural Patterns Revisited – A Pattern Language*, EPLOP, 2005.

J. Newmarch, *A RESTful approach: clean UPnP without SOAP, Proceedings CCNC 2005, p134-138*.