SYSTEM ARCHITECTURE

Distributed Systems

BSc Informatics, 4th Semester

OVERVIEW

1. Architectural styles
   a. Layered architecture
   b. Object-based
   c. Service-oriented
   d. Resource-based
   e. Publish/subscribe
2. System architecture
   a. Centralized architecture
   b. Decentralized architectures (P2P)
   c. Hybrid architectures
3. Examples

OVERVIEW

1. **Architectural styles**
   a. Layered architecture
   b. Object-based
   c. Service-oriented
   d. Resource-based
   e. Publish/subscribe
2. System architecture
   a. Centralized architecture
   b. Decentralized architectures (P2P)
   c. Hybrid architectures
3. Examples

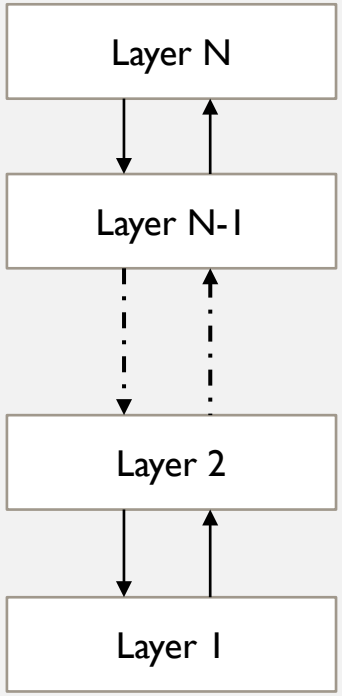
ARCHITECTURAL STYLES

- **Logical** organization of the system into software components.
- Architectural style important for sucessful development.
  - Components (and their **interfaces**).
  - How components are connected to each other (**connectors**).
  - System configuration.
- Most systems are a combination of the following architectural styles:
  - Layered architecture.
  - Object-based.
  - Service-oriented.
  - Resource-based.
  - Publish-subscribe

ARCHITECTURAL STYLES

Layered architecture

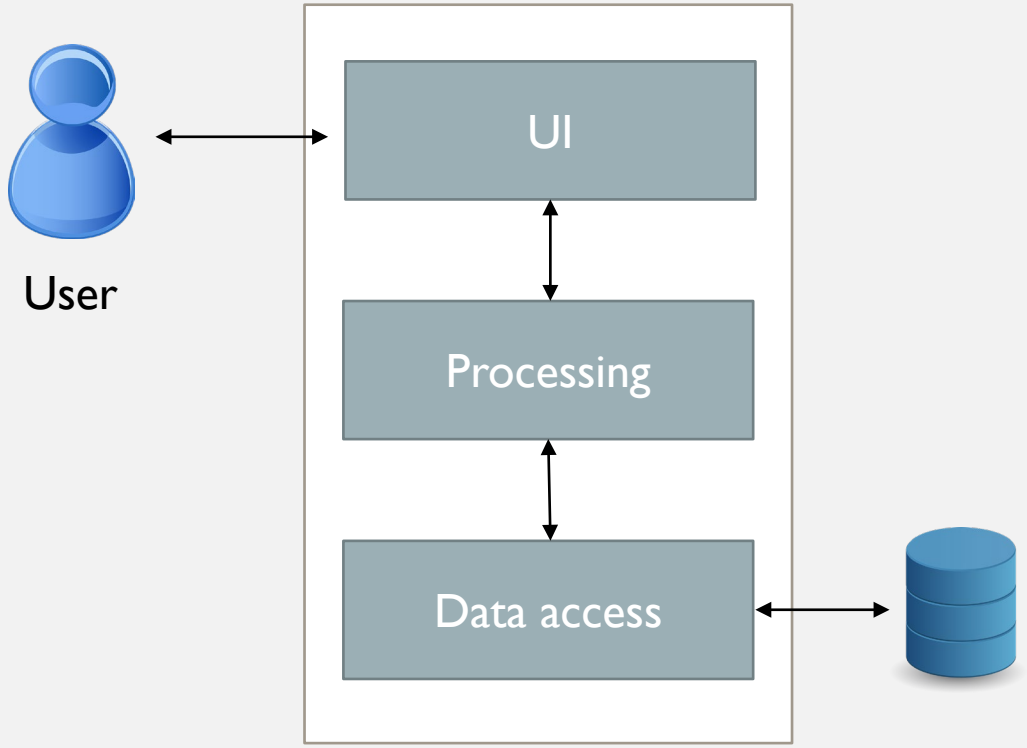
- Each layer performs a **downcall** to the layer unterneath it.
- **Results** are propagated upwards
- Typical example: network communication
  - Network protocols (OSI model)
  - Each layer represents an abstraction level.



ARCHITECTURAL STYLES

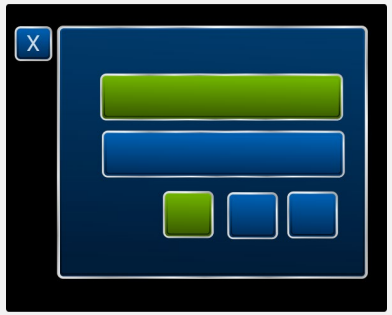
Application layered architecture

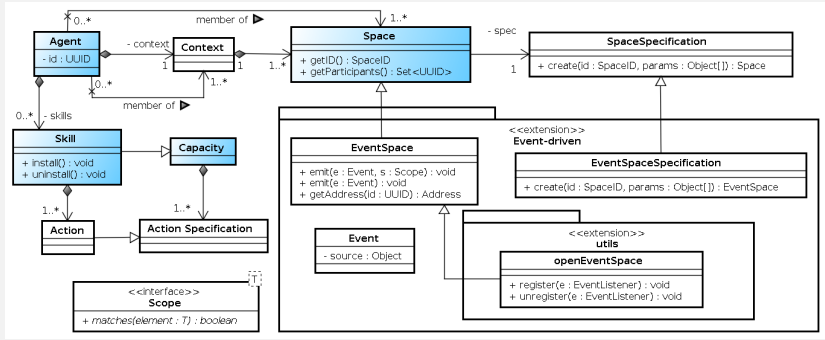
- How does a typical application look like?
  A. User interface
  B. Processing layer
  C. Data layer



ARCHITECTURAL STYLES

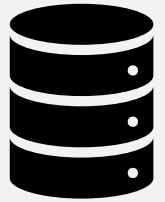
- User interface
  - Implementation of the user interaction (frontend).
  - Primitive: console interaction/echoing user input.
  - More advanced: Web UI with the newest frameworks (React, etc.).
- Processing layer
  - Commonly referred to as **business logic**.
  - Implementation of a domain model for the problem at hand.





ARCHITECTURAL STYLES

- Data access layer
  - Access and management to **persistent** storage
  - File system
  - Most commonly: full-fledged database (MySQL, MariaDB, MS-SQL, etc)
- Examples
  - Almost any ”classic” web application



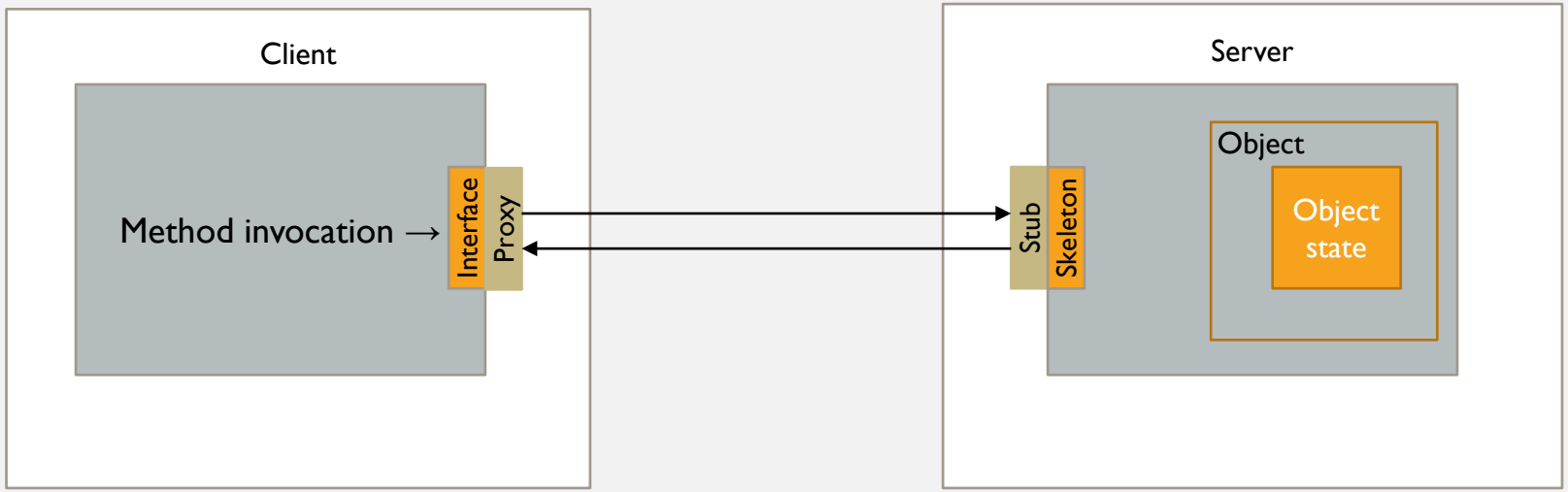
ARCHITECTURAL STYLES

Object-based architecture

- **Distributed objects** hosted on one machine can be called from another.
- The state of the object itself can be completely stored in one server or distributed across several servers.
- Client: connects (*binds*) to the distributed object and calls a method.
- Locally: proxy serializes (*marshalls*) request to the server.
- Server stub: unmarshalls object.
- Skeleton: calls object method directly on the server.
- Result is returned analogously.

ARCHITECTURAL STYLES

Object-based architecture



ARCHITECTURAL STYLES

Object-based architecture

Examples:

- Python → Pyro5
- Java → Remote Method Invocation (RMI)
- C# → .NET remote objects (Windows Communication Foundation – WCF)
- CORBA Architecture (programming language agnostic)

ARCHITECTURAL STYLES

Service-oriented architecture (SOA)

- Going one-step further beyond distributed objects.
- Encapsulate functionality as **services** in self-contained entities.
- Services operate independently.
- Distributed application = composition of services.
- Services offer a well-defined interface.
- We'll go into detail in the next lecture.

ARCHITECTURAL STYLES

Resource based

- How to best compose services in Web-based applications.
- Problem: integration nightmare if every service has its *own* interface.
- Idea: define and exchange **resources.**
- Resources can be queried created, modified, deleted.
- **RESTful architectures.**

ARCHITECTURAL STYLES

RESTful architectures

- Representational State Data Transfer
  1. Resources are identified through a single scheme (URI)
  2. All services offer the same interface (HTTP verbs).
  3. Message sent are fully self-described.
  4. **Stateless** execution: after executing a request, the callee forgets the caller.

ARCHITECTURAL STYLES

RESTful architectures

- Operations
  - GET: retrieve resource(s) in some representation.
  - PUT : create a resource.
  - DELETE: deletes a resource.
  - POST: modifies a resource.
- Example
  - https://dog.ceo/api/breeds/list/all

ARCHITECTURAL STYLES

Publish-subscribe architectures

- Processes are **referentially decoupled** but **temporally coupled.**
- Referentially decoupled: processes don't reference each other.
- Temporally coupled: typically all processes are up-and-running.
  - If not required: **shared-space** architecture.
- Processes send and receive **notifications** that are triggered by **events**.
- Processes only receive notifications they are interested in: **publish-subscribe.**

dont know each other but they run at the same time

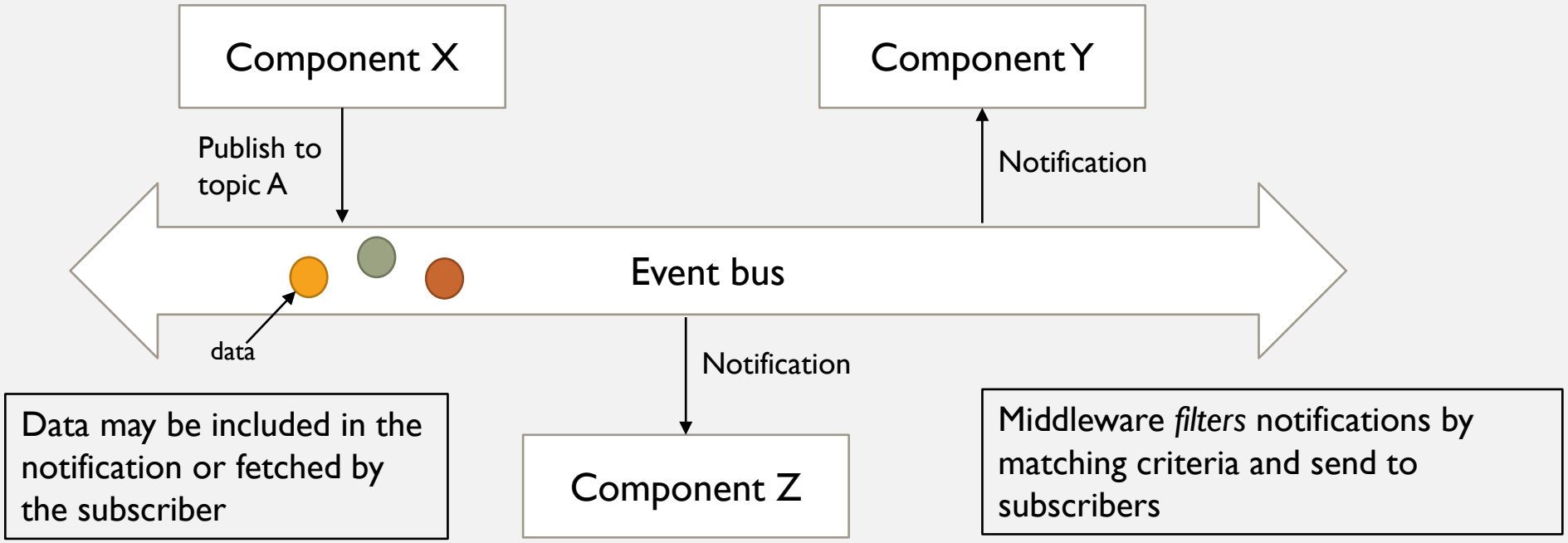
ARCHITECTURAL STYLES

Publish-subscribe architectures

- **Topic-based** event subscribe: matching by topic name.
  - Example: all notifications sent to topic with name A.
- **Content-based** event subscribe: matching by arbitrary predicates.
  - Example: all notifications satisfying predicate ”generated by usergroup X”.
- **Hybrid**: message sent to topic A but subscribed content-based.
  - Example: all notifications sent to topic A with predicate P.

ARCHITECTURAL STYLES

Publish-subscribe architectures



OVERVIEW

1. Architectural styles
   a. Layered architecture
   b. Object-based
   c. Service-oriented
   d. Resource-based
   e. Publish/subscribe
2. **System architecture**
   a. Centralized architecture
   b. Decentralized architectures (P2P)
   c. Hybrid architectures
3. Examples

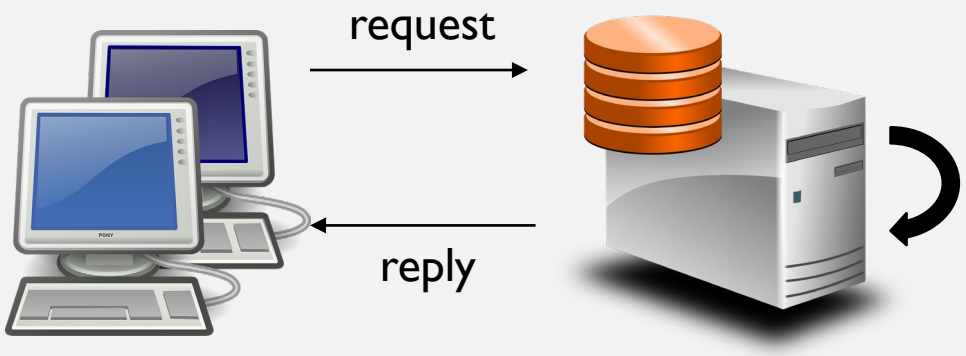
SYSTEM ARCHITECTURE

- Software components + Interactions + Placement = **System architecture**
- A **system architecture** can be understood as a concrete realization of one or several architectural styles combined, including component placement + interaction
- Possible system architectures
  - Centralized
  - Decentralized (Peer-to-peer)
  - Hybrid

SYSTEM ARCHITECTURE

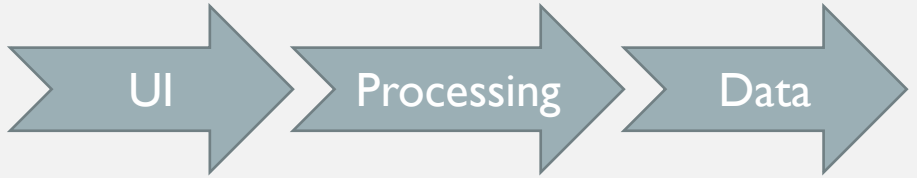
Centralized architecture

- Simple client server architecture:
  - Client requests a service and waits for reply.
  - Server provides the service.
  - **Request-reply** behaviour.
- Multitiered architecture:
  - Usually three logical layers: UI, processing, data access
  - **Two-tiered architecture**: only server/client machines
  - Which logical layer runs on which tier?



SYSTEM ARCHITECTURE

Multitiered architecture



Terminal-dependent thin-client

Frontend performs only presentation

Thin-client

when the frontend performs some actions not related to presentation

Thin-client with business logic

majority of functionality on client side,

Fat client

Fat client with data (i.e. caching)

aditional data is stored on the mechine, business processes+data processes

SYSTEM ARCHITECTURE

Centralized architecture

- Multitiered architecture
  - **Three-tiered architecture:** server also acts as a client.
  - For instance, database located in another server (most common).
  - Application server is a client of the database server .

SYSTEM ARCHITECTURE

Peer-to-peer systems

different components on different machines
in diff layers

- Multitiered applications → vertical distribution.
  - Place logical different components in different machines.
- Horizontal distribution

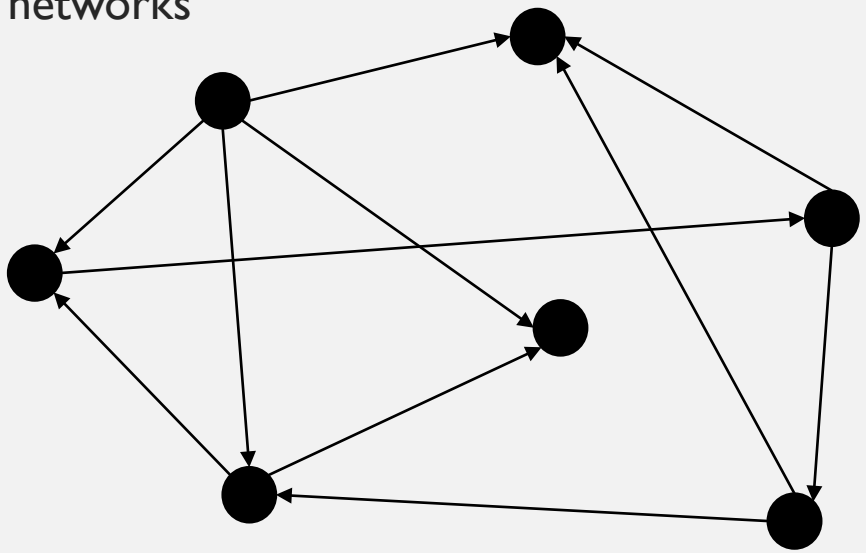
load balancing
same task
distributing work around machines

  - Each part operates on each own share of the data set.
  - Load balancing.
  - P2P systems support horizontal distribution.
  - Process can be client and server at the same time (*servant*).

SYSTEM ARCHITECTURE

Peer-to-peer systems

- Overlay networks



Nodes = processes
Arrows = communication channels

we are talking about
nodes are processes
arrows are logical connections
byproduct ofthe application

SYSTEM ARCHITECTURE

Overlay networks

- **Structured** networks: topology of the network is known in advance.
  - Structure used to efficiently look-up data.
- **Unstructured** networks: topology is not known.
  - Random graph.

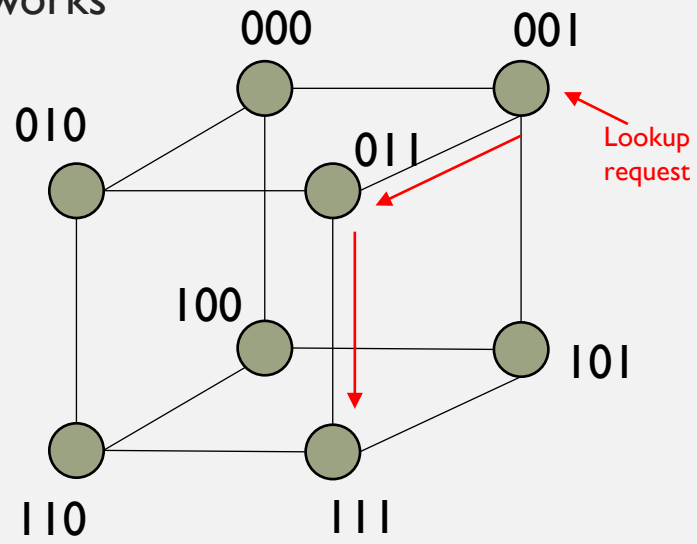
SYSTEM ARCHITECTURE

Structured networks

- Look-up data: **distributed hash table (DHT)**
- key(item) = hash(item value)
- Each node stores data for a subset of keys
- Main problem: map a key to an existing node
- Structure → efficiently *route* a lookup request to the node containing the data

SYSTEM ARCHITECTURE

Structured networks



1. Pick node 001
2. Lookup(15)
3. Hash(15)=111
4. Route request to 011
5. 011 routes request to 111

send to a neighbour that is nearer to the result

SYSTEM ARCHITECTURE

Unstructured networks

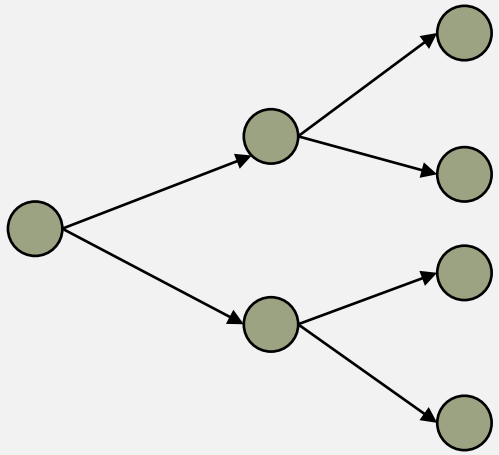
only thing you know are your neighbouts

- Topology not known.
- Each node maintains a list of **neighbours.**
- List is maintained dynamically.
  - New nodes may join.
  - Irresponsive nodes have to be replaced.
- Main problem: **searching** for data.

SYSTEM ARCHITECTURE

Unstructured networks

- How to search?
  - **Flooding**: each node passes each request to *all* of its neighbour



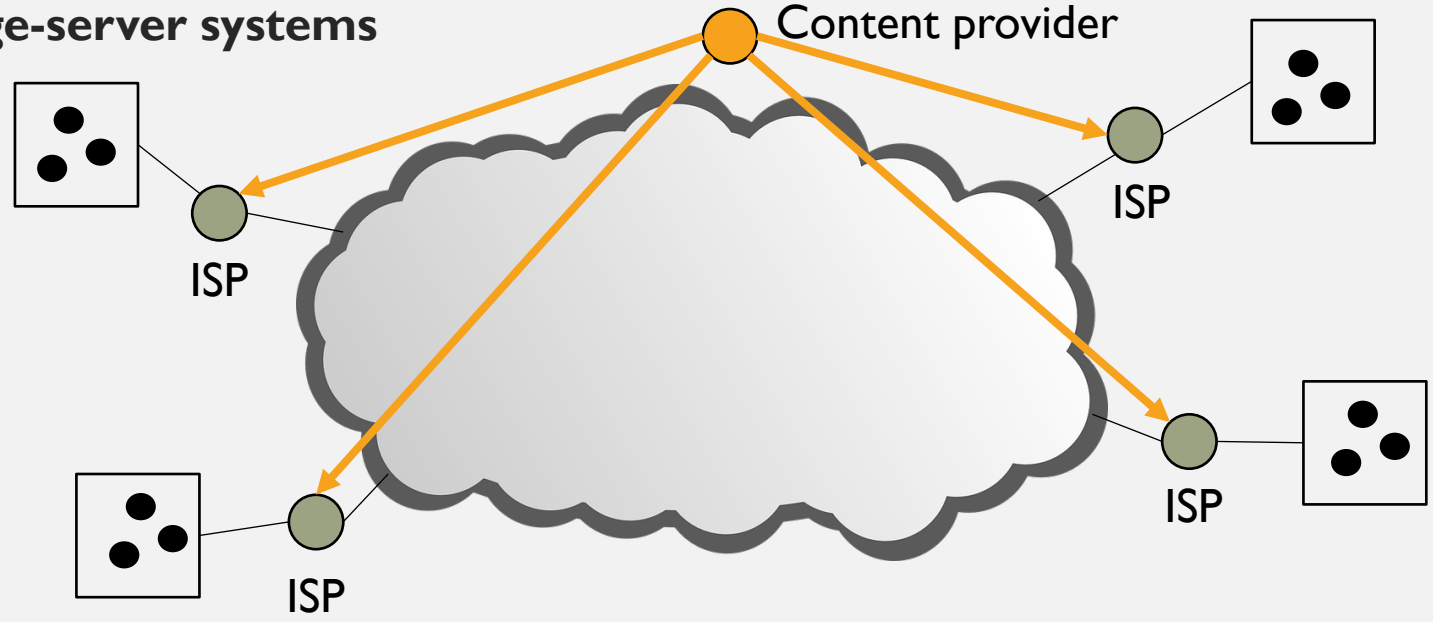
- Flooding is expensive!
- TTL = time-to-live: maximum number of hops.
- Avoid cycles: if request was already seen, ignore.
- Progressive variant: TTL=1, then TTL=2, etc. until item found.

SYSTEM ARCHITECTURE

Hybrid architectures

CDN - content delivering network

- Many distributed systems combine client-server with decentralized architectures
- **Edge-server systems**



- Edge-server serves content.
- Filtering.
- Transcoding.

OVERVIEW

1. Architectural styles
   a. Layered architecture
   b. Object-based
   c. Service-oriented
   d. Resource-based
   e. Publish/subscribe
2. System architecture
   a. Centralized architecture
   b. Decentralized architectures (P2P)
   c. Hybrid architectures
3. **Examples**

EXAMPLES

World Wide Web

- Initially a way of sharing documents over the Internet (Berners-Lee, 1989).
- **info.cern.ch**: first world's ever website.
- Way of sharing documents between researchers.
- Communication by means of the Hyper Text Transfer Protocol (HTTP).
- Documents written in Hyper Text Markup Language (HTML).
- Possibility to combine different media types.
- Naming a resource by its Uniform Resource Locator (URL).

EXAMPLES

World Wide Web

- Two-tiered client/serve architecture
  - Client: Browser
  - Server: HTTP server
- Transformation from **static** content to **dynamic** content
- **Common Gateway Interface (CGI)**
  - Client sends input data
  - Server executes the program + input data given in the request
  - Server fetches result document and serves it to the client
  - Client's browser displays result

EXAMPLES

World Wide Web

- WWW as base for service oriented architectures.
- Communication via HTTP protocol.
- Embedding programs in web documents (Javascript).
- Web applications.

EXAMPLES

Skype

- Peer-to-peer network with super peers.
- Centralized Skype login server.
  - Accesible by both super peers and weak peers.
- Weak peers: host cache of super peers.
- Connection with super peer via TCP connection.
- User search: weak peers ask super peer → list of other peers to ask.
- If not found → super peer returns longer list.
- Until user found.
- **Policy based** search.

EXAMPLES

Skype

- Then, VOIP connection is established.
- VOIP call via UDP connection.
  - Special cases if caller or callee are behind a firewall.
- What are the advantages of using a central login server?