Applied Architectures and Styles

Software Architecture
Chapter 11

Objectives

- Describe how the concepts from the previous chapters can be used, sometimes in combination, to solve challenging design problems
- Highlight key issues in emerging application domains that have architectural implications, or where an architectural perspective is essential for system development within that domain
- Show how emerging architectures, such as P2P, can be characterized and understood through the lens of software architecture

Outline

- Distributed and networked architectures
  - Limitations
  - REST
  - Commercial Internet-scale applications
- Decentralized applications
  - Peer-to-peer
  - Web services
- Some interesting domains
  - Robotics
  - Wireless sensors

Distributed and Networked Architectures

- The phrase "distributed applications" is used to denote everything from
  - An application that is simply distributed across multiple operating system processes all running on the same physical uniprocessor, to
  - Integrated applications that run on multiple computers connected by the Internet
- A distributed model often supports *'location transparency'*
  - The developer should not necessarily know where the various processes/objects comprising the application are located in the network

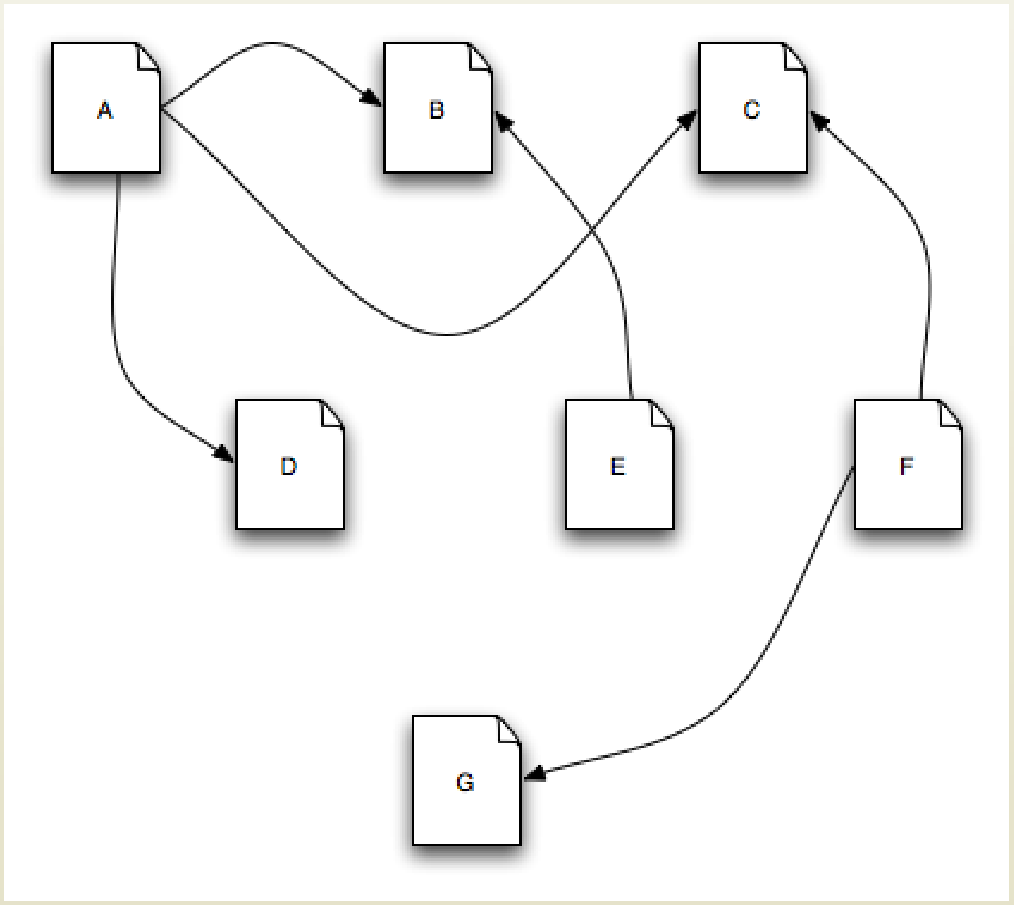
Limitations of the Distributed Systems Viewpoint

- However, masking the underlying presence of networks, becomes difficult due to the presence of the following fallacies
  - The network is reliable
  - Latency is zero
  - Bandwidth is infinite
  - The network is secure
  - Topology doesn’t change
  - There is one administrator
  - Transport cost is zero
  - The network is homogeneous

-- Deutsch & Gosling

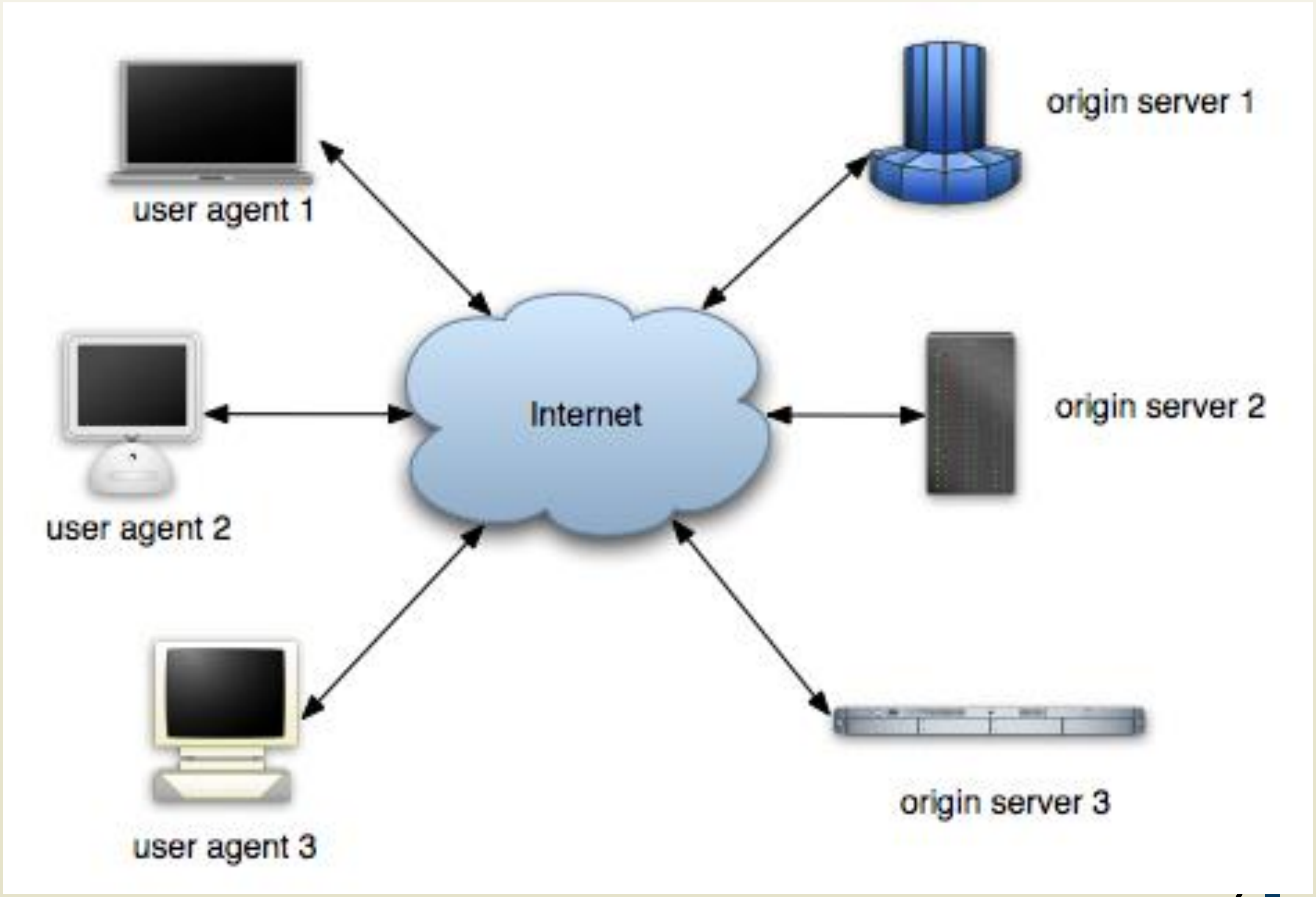
Architecture in Action: WWW

- (From lecture #1) This is the Web



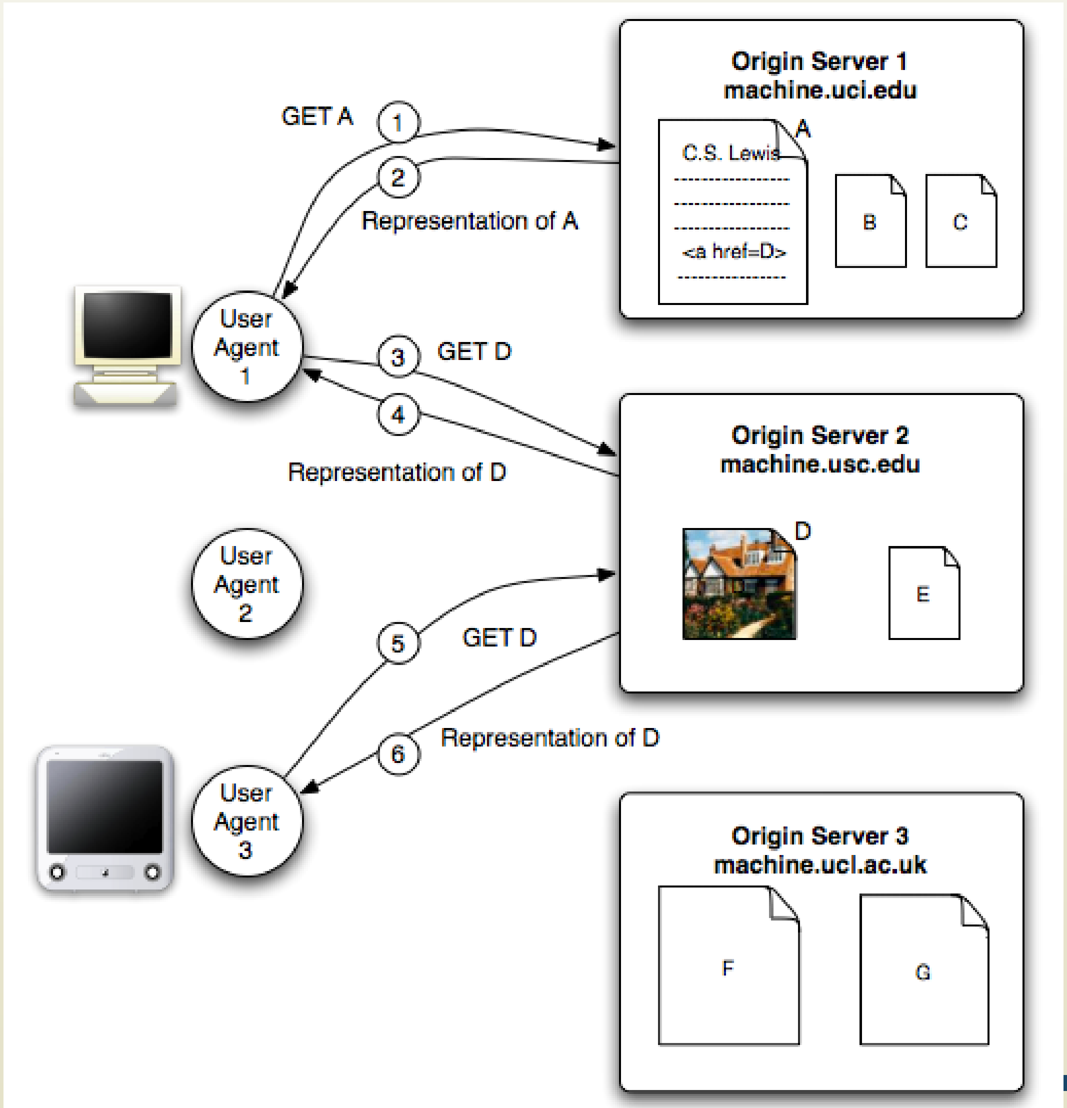
Architecture in Action: WWW (cont’d)

- So is this



Architecture in Action: WWW

- And this



WWW’s Architecture

- **The application is distributed (actually, decentralized) hypermedia**
- Architecture of the Web is wholly separate from the code
- There is no single piece of code that implements the architecture
- There are multiple pieces of code that implement the various components of the architecture
  - E.g., different Web browsers
- Stylistic constraints of the Web’s architectural style are not apparent in the code
  - The effects of the constraints are evident in the Web
- One of the world’s most successful applications is only understood adequately from an architectural vantage point

REST Principles

- [RP1] The key abstraction of information is a resource, named by an URL; any information that can be named can be a resource
- [RP2] The representation of a resource is a sequence of bytes, plus representation metadata to describe those bytes; the particular form of the representation can be negotiated between REST components
- [RP3] All interactions are context-free: each interaction contains all of the information necessary to understand the request, independent of any requests that may have preceded it

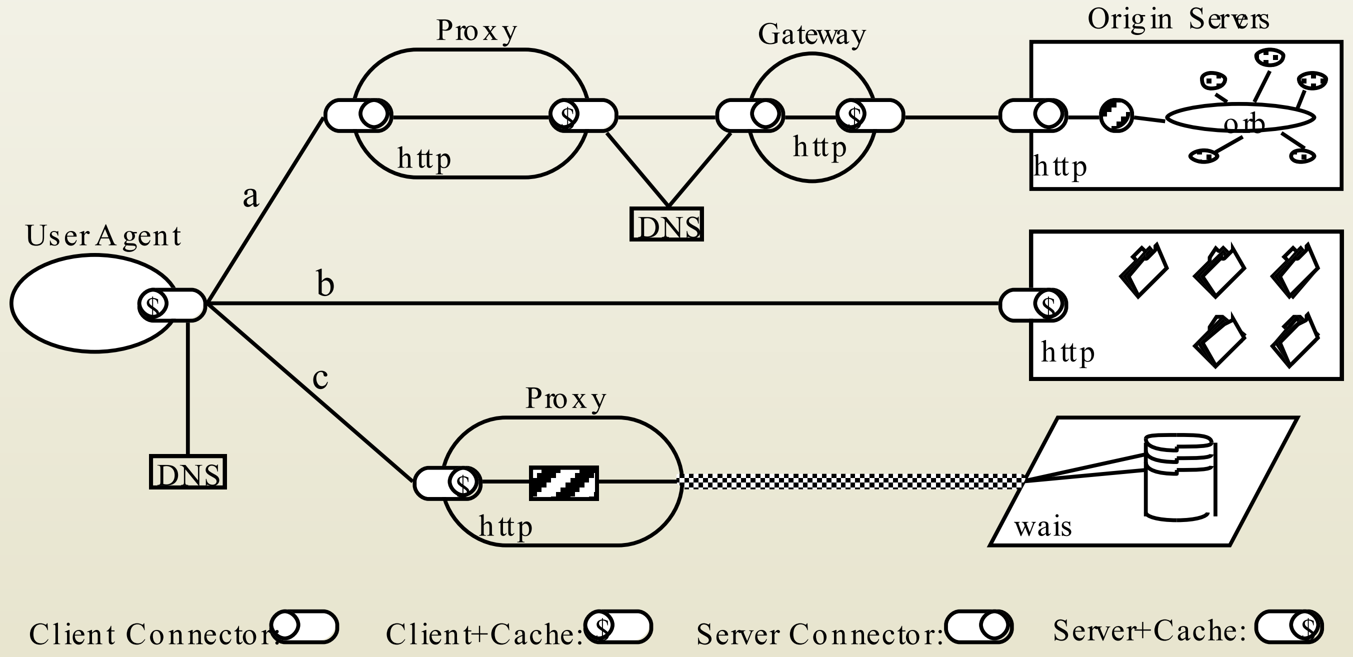
REST Principles (cont’d)

- [RP4] Components perform only a small set of well-defined methods on a resource producing a representation to capture the current or intended state of that resource and transfer that representation between components; these methods are global to the specific architectural instantiation of REST; for instance, all resources exposed via HTTP are expected to support each operation identically

REST Principles (cont’d)

- [RP5] Idempotent operations and representation metadata are encouraged in support of caching and representation reuse
- [RP6] The presence of intermediaries is promoted; filtering or redirection intermediaries may also use both the metadata and the representations within requests or responses to augment, restrict, or modify requests and responses in a manner that is transparent to both the user agent and the origin server

An Instance of REST



An Instance of REST (cont'd)

- Process view of a REST-based architecture at one instance in time
- A user agent is portrayed in the midst of three parallel interactions: `a`, `b`, and `c`
- The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector

An Instance of REST (cont'd)

- Request (`a`) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture
- Request (`b`) is sent directly to an origin server, which is able to satisfy the request from its own cache
- Request (`c`) is sent to a proxy that is capable of directly accessing `WAIS`, an information service that is separate from the Web architecture, and translating the `WAIS` response into a format recognized by the generic connector interface
- Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view

REST — Data Elements

- Resource
  - Key information abstraction
- Resource ID
- Representation
  - Data plus metadata
- Representation metadata
- Resource metadata
- Control data
  - E.g., specifies action as result of message

REST — Connectors

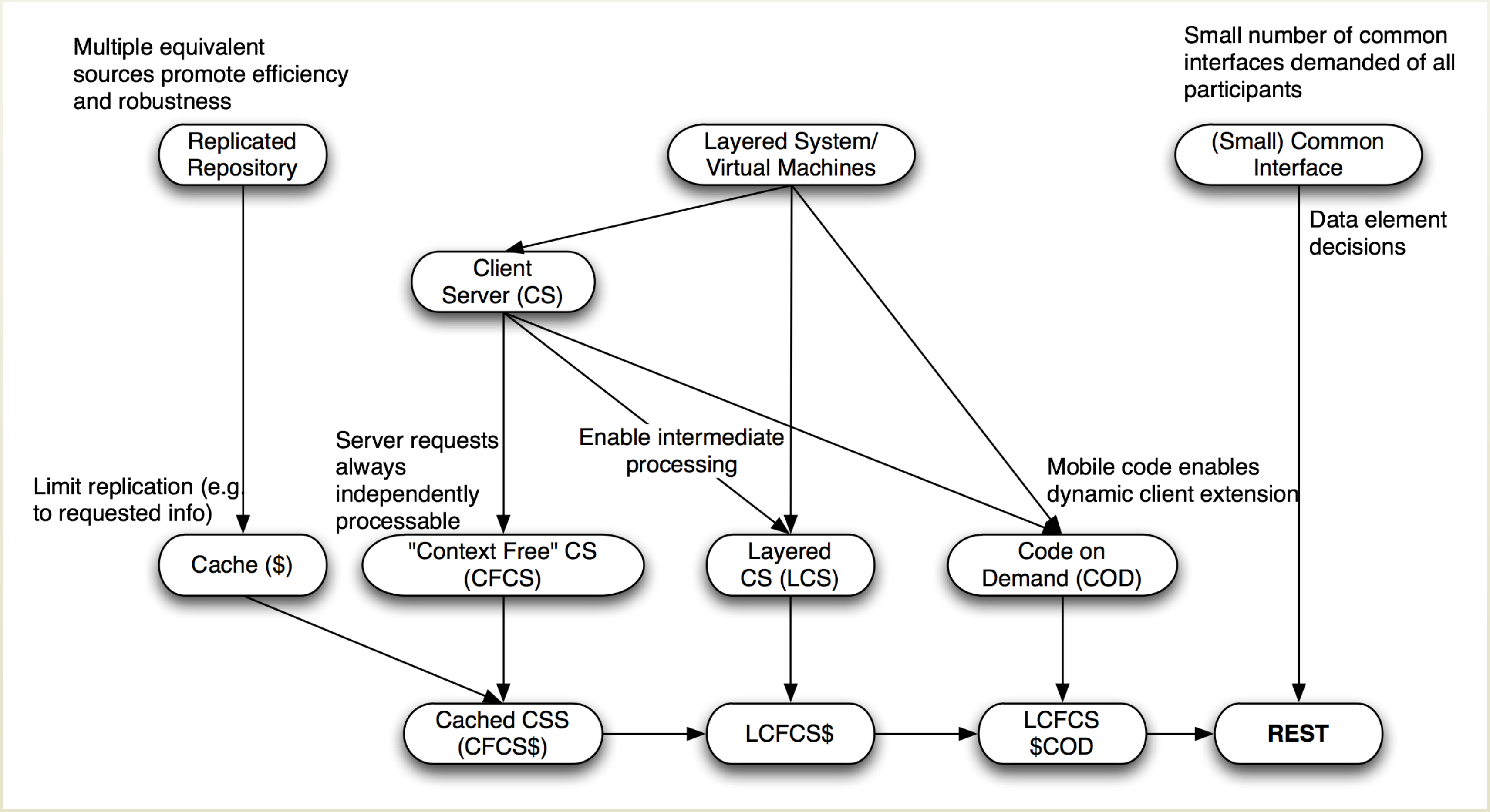
- Modern Web Examples

client	libwww, libwww-perl
server	libwww, Apache API, NSAPI
cache	browser cache, Akamai cache network
resolver	bind (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT

REST — Components

- User agent
  - E.g., browser
- Origin server
  - E.g., Apache Server, Microsoft IIS
- Proxy
  - Selected by client
- Gateway
  - Squid, CGI, Reverse proxy
  - Controlled by server

Derivation of REST


Multiple equivalent sources promote efficiency and robustness
Replicated Repository
Layered System/ Virtual Machines
Small number of common interfaces demanded of all participants
(Small) Common Interface
Data element decisions
Client Server (CS)
Server requests always independently processable
Enable intermediate processing
Mobile code enables dynamic client extension
Limit replication (e.g. to requested info)
Cache ($)
"Context Free" CS (CFCS)
Layered CS (LCS)
Code on Demand (COD)
Cached CSS (CFCS$)
LCFCS$
LCFCS $COD
REST

Derivation of REST (cont'd)

Key choices in this derivation include:

- *Layered Separation* (a theme in the middle portion of diagram) is used to increase efficiencies, enable independent evolution of elements of the system, and provide robustness
- *Replication* (left side of the diagram) is used to address latency and contention by allowing the reuse of information
- *Limited commonality* (right side) addresses the competing needs for universally understood operations with extensibility
- A fourth key issue is *dynamic extension*, i.e., by allowing clients to receive arbitrary data described by meta-data, in response to a request to a server, allows client functionality to be extended dynamically and the client is able to perform new functions and be customized to the needs of the client's user

REST: Final Thoughts

- **RE**presentational **S**tate **T**ransfer
- It combines knowledge in open network-distributed hypermedia and several simple architectural styles
- A coherent architectural approach is derived by imposing constraints on those architectural styles
- It is a powerful tool to use in network-based applications where issues of latency and authority boundaries are prominent
- It is also instructive in guiding architects in the creation of other specialized styles

Commercial Internet-Scale Applications

- Akamai
  - Caching (i.e., storing copies of files in a cache, or temporary storage location, so that they can be accessed more quickly) to the max
  - Replicates the origin server (where originally the cached content is stored) at many locations throughout the network, and directs user's requests to the replicated server closer to the user
  - The replicants of the original server are called 'edge servers' and Akamai has located thousands of them throughout the Internet, especially close to where ISPs join their networks to the Internet
  - The edge servers do have to access the origin server to update their content, but by directing users to the edge servers, demand on the origin server is kept manageable

Commercial Internet-Scale Applications (cont'd)

- Google
  - Its applications rest upon the ability to manipulate very large quantities of information, stored in inexpensive PCs running Linux, and being fault-tolerant by supporting replication
  - The storing system is called GFS (Google File System), it is simpler than a typical relational database and supports storage of files that are very large (several gigabytes)
  - On top of it runs MapReduce, a programming model in which users specify data selection and reduction operations
  - The MapReduce library parallelizes users' operations to execute on the thousands of processors available, without the user needing to deal explicitly with this issue
  - The system is designed to deal with any failures of the processors involved in the parallel executions of user queries

Architectural Lessons from Google

- **Abstraction layers abound**: GFS hides details of data distribution and failure, for instance; MapReduce hides the intricacies of parallelizing operations
- By designing, from the outset, for **living with failure** of processing, storage, and network elements, a highly robust system can be created
- **Scale is everything**: Google's business demands that everything be built with scaling issues in mind

Architectural Lessons from Google (cont'd)

- By **specializing the design to the problem domain**, rather than taking the generic "industry standard" approach, high performance and very cost-effective solutions can be developed
- By **developing a general approach** (MapReduce) to the data extraction/reduction problem, a highly reusable service was created
- Note the contrast between the two points above
  - On the one hand, a *less general* solution strategy was adopted (a specialized file system rather than a general database)
  - On the other hand, a *general programming* model and implementation was created

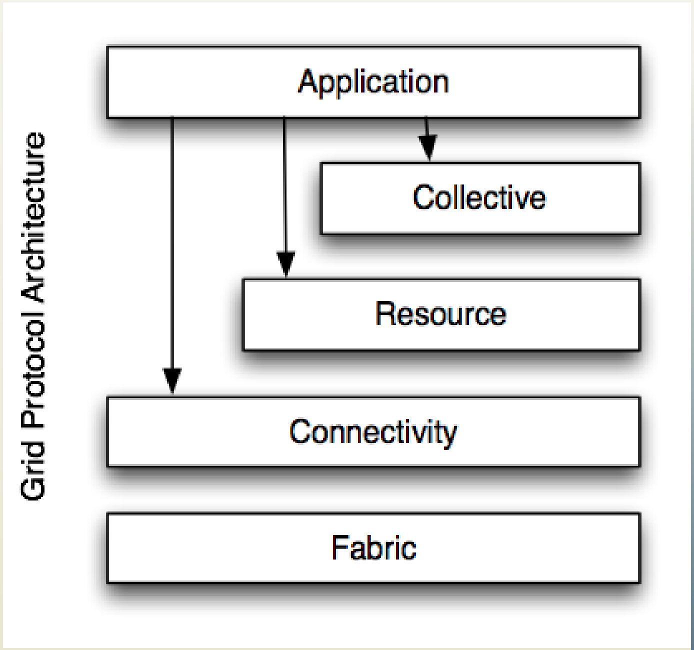
Decentralized Architectures

- Networked applications where there are multiple authorities
- In other words
  - Computation is distributed
  - Parts of the network may behave differently, and vary over time
  - E.g., web sites, e-commerce, etc.

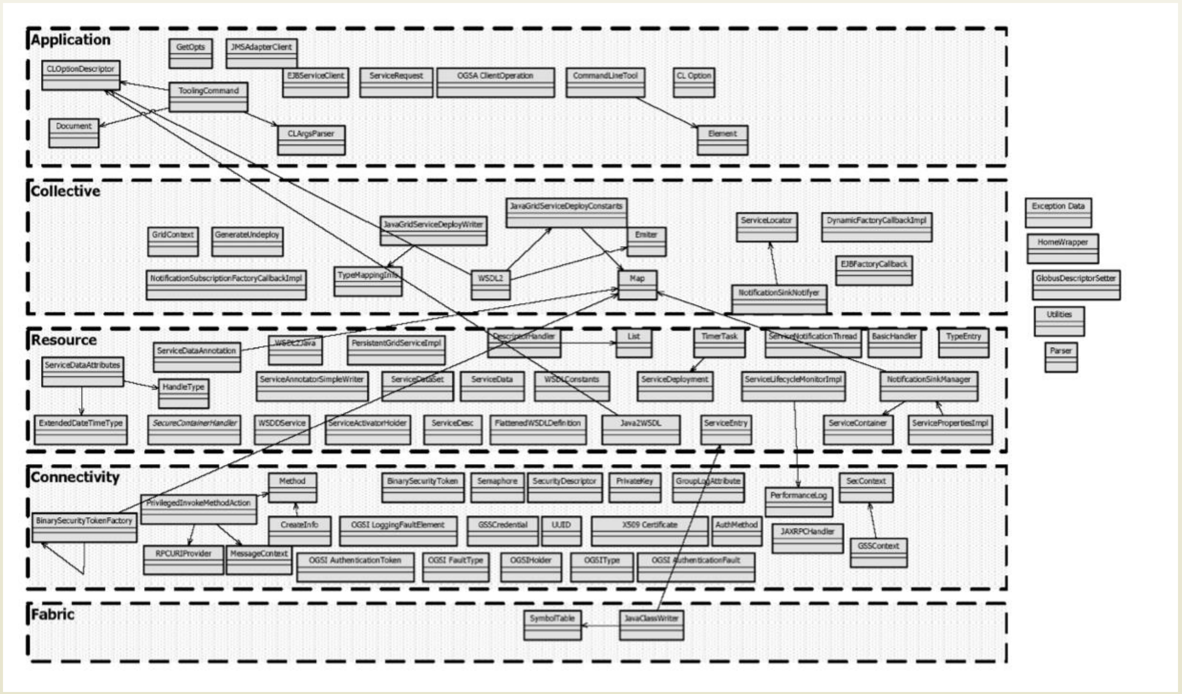
It’s just like collaboration in the real world (e.g., postal system)

Grid Protocol Architecture

- Coordinated resource sharing in a distributed environment, using h/w and s/w resources to solve a computational problem (e.g., visualization of simulations)
- "Standard layered architecture"
  - `Application` contains the components that implement the user's particular system
  - `Collective` coordinates the use of multiple resources
  - `Resource` manages the sharing of a single resource
  - `Connectivity` handles communication and authentication
  - `Fabric` manages the details of the low-level resources that ultimately comprise the grid



Grid Protocol Architecture (cont'd)



- The elegance and clean design of this architecture is not fully maintained by a number of grid technologies
- The figure shows the recovered architecture of the Globus grid system, derived from analyzing the source code
- Several up-calls in the architecture are present, violating the layered systems principle

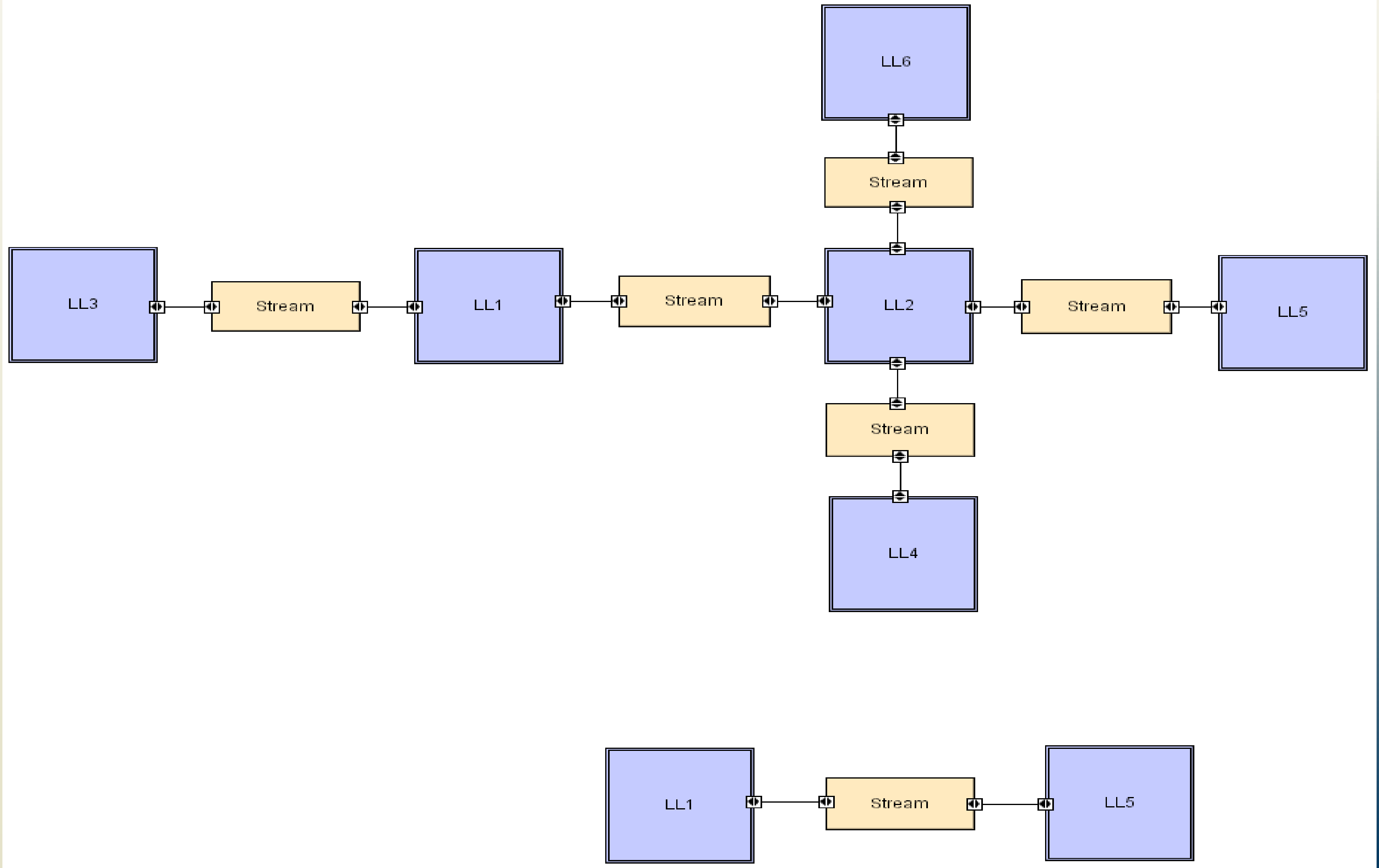
Peer-to-Peer Architectures

- Decentralized resource sharing and discovery
  - Napster
  - Gnutella
- P2P that works: Skype
  - And BitTorrent

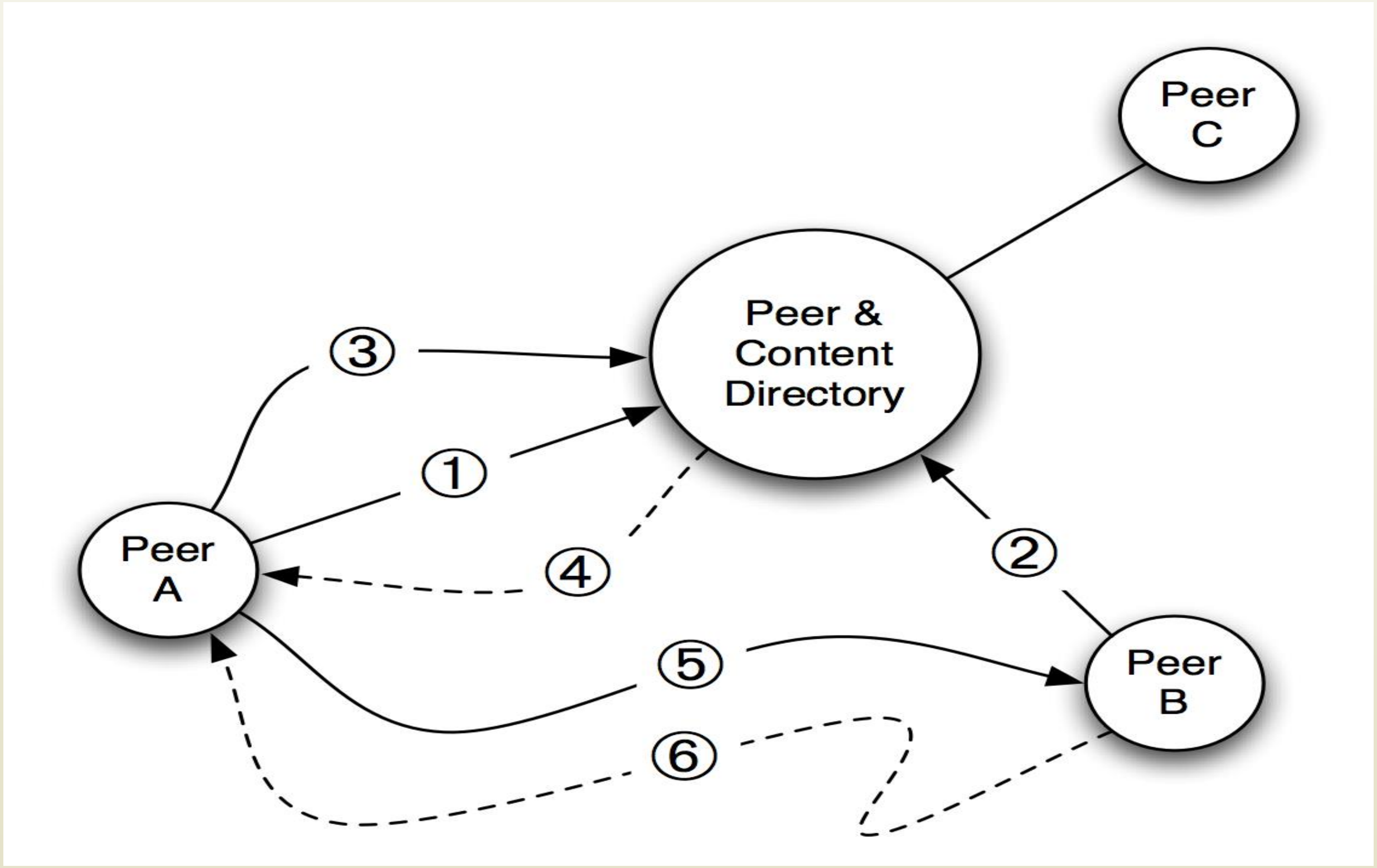
Peer-to-Peer Style

- State and behavior are distributed among peers which can act as either clients or servers
- Peers: independent components, having their own state and control thread
- Connectors: Network protocols, often custom
- Data Elements: Network messages
- Topology: Network (may have redundant connections between peers); can vary arbitrarily and dynamically
- Supports decentralized computing with flow of control and resources distributed among peers
- Highly robust in the face of failure of any given node
- Scalable in terms of access to resources and computing power

Peer-to-Peer LL



Hybrid CS/P2P: Napster



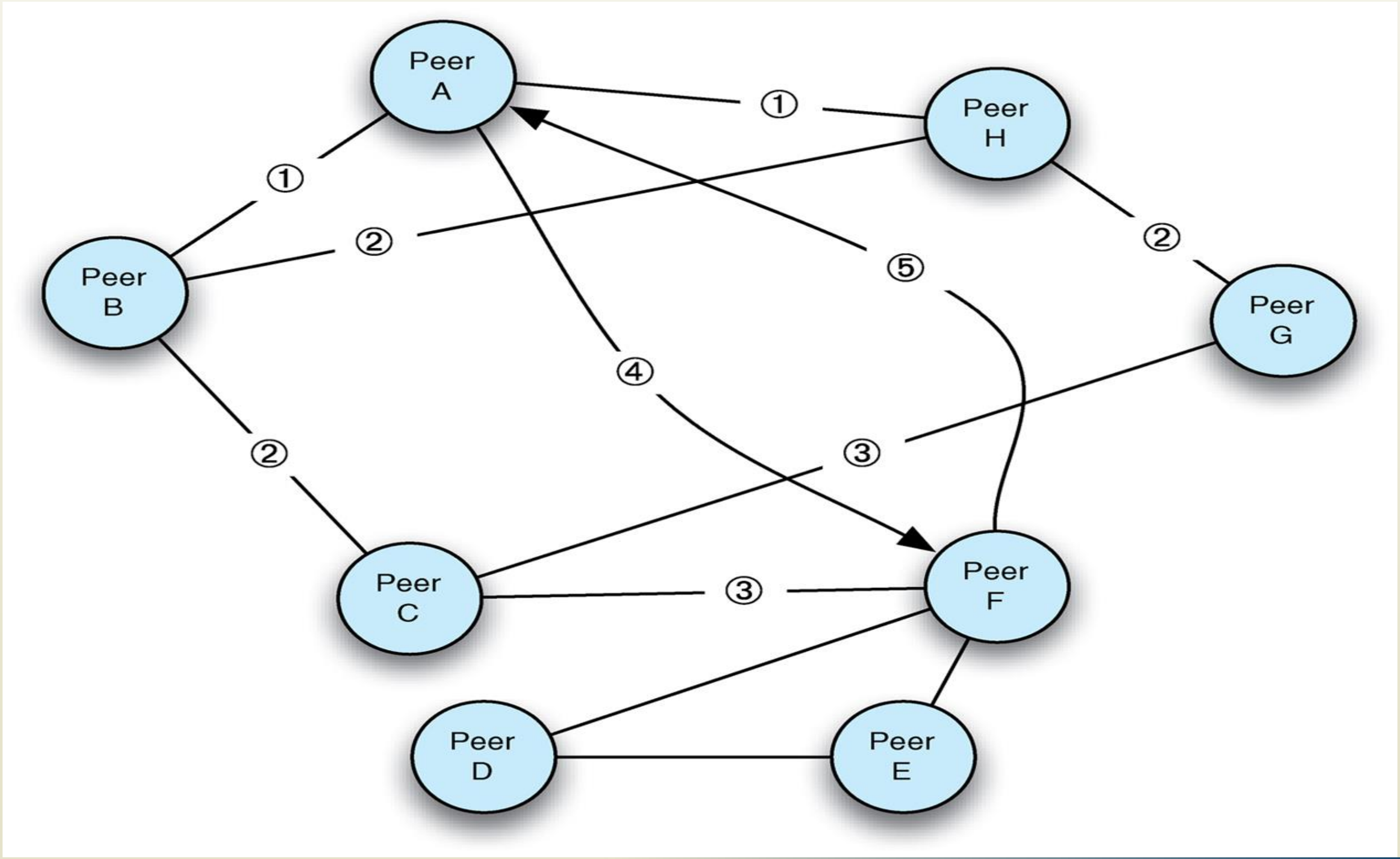
Hybrid CS/P2P: Napster (cont'd)

- Each of the peers is an independent program residing on the computers of end users
- Operation begins with the various peers registering themselves with the central server (`Peer & Content Directory`)
    - When registering or later logging in, a peer informs the server of its IP address and the music files that this peer can share (1,2)
- The server maintains a record of the music that is available, and on which peers
- Later a peer may query the server as to where on the Internet a given song can be obtained (3)
- The server responds with available locations (4)
- The peer then chooses one of those locations, makes a call directly to the peer (5), and downloads the music (6)

Hybrid CS/P2P: Napster (cont’d)

- Architecturally, this system can be seen as a hybrid of client-server and pure P2P
- The peers act as clients when registering with `Peer & Content Directory` and querying it
- Once a peer knows where to ask for a song, a P2P exchange is initiated
  - Any peer may thus sometimes act as a client (asking others for a song) or a server (delivering a song it has to another peer which has requested it)
- Napster uses a proprietary protocol for interactions between the peers and the content directory
- If a highly desired song becomes available, the server will be swamped with requests seeking its location
- And if the server goes down, peers cannot access any content

Pure Decentralized P2P: Gnutella



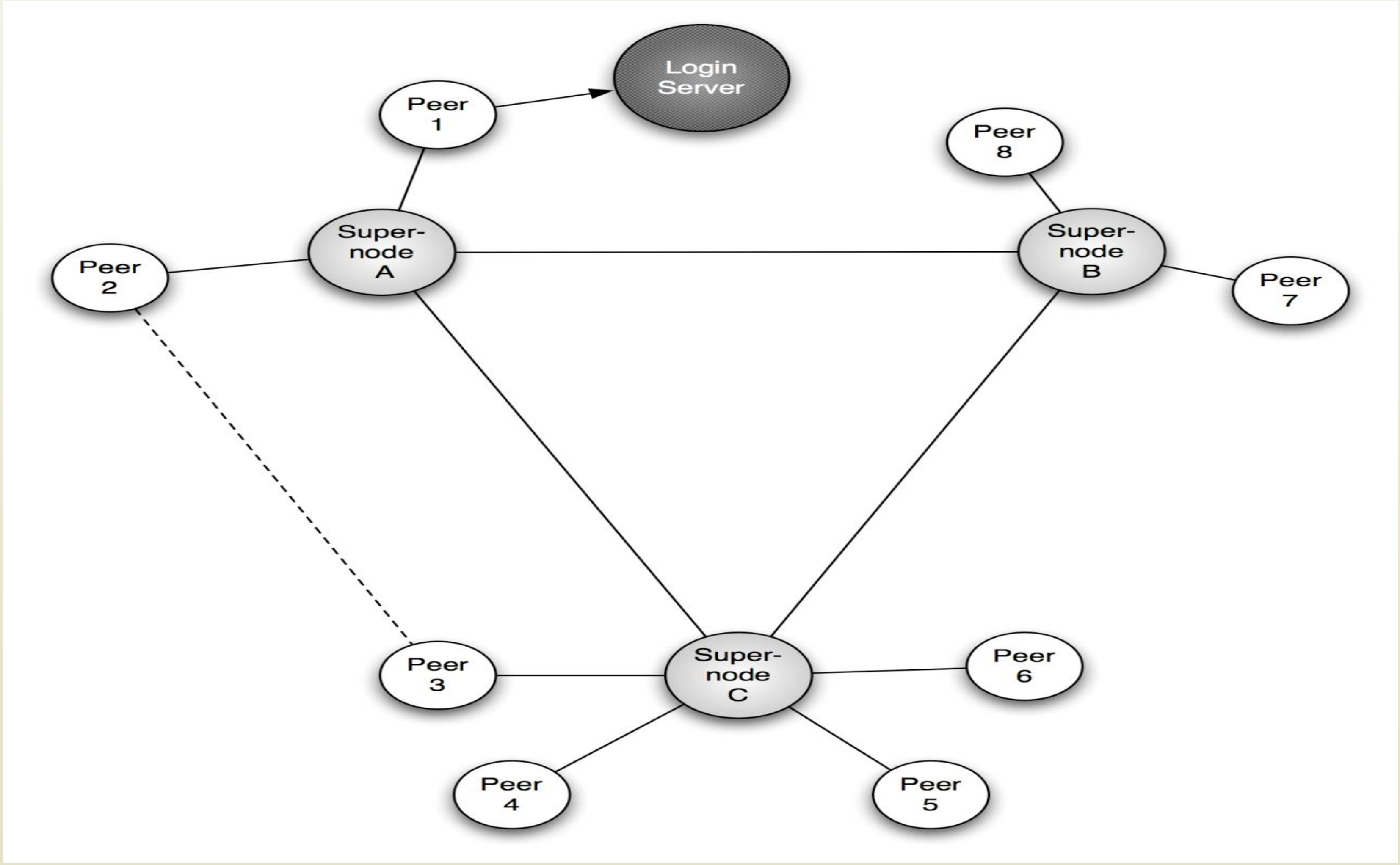
Pure Decentralized P2P: Gnutella (cont'd)

- The original version was a pure P2P system; there is no central server, and all peers are equal in capability and responsibility
- If `A` is seeking a particular song, it issues a query to the peers on the network that it knows, in this case `B` and `H` (1)
- Assuming these peers don't have the song, they pass the query to other peers they know about: each other, and `C` and `G` (2)
- Assuming that `F` has the song, it responds to the peer that asked it, which in this case is `C`, telling it that it has the song (3)
- `C` then relays this information along with `F`'s network address to the peers that asked for this information
- Eventually, `A` will obtain the address of `F` and initiates a direct request to `F` (4)
- The song is then downloaded (5)

Pure Decentralized P2P: Gnutella (cont'd)

- Gnutella is highly robust; removal of any one peer from the network, or any set of peers, does not diminish the ability of the remaining peers to continue to perform
- However, there are also a number of problems
  - How does a new peer find any other peers to send to them its queries?
  - When a query is issued, how many peers will end up being asked for the requested resource *after* some other peer has already responded and provided the information?
  - How long should a requesting peer wait to obtain a response?
  - What assurance does the requesting peer have that the information downloaded is that which was sought (and not, say, a virus)?

Overlayed P2P: Skype



Overlayed P2P: Skype (cont'd)

- `Peer 1` logs into the Skype server
- Skype server tells `Peer 1` the address of `Super-node A`
- When `Peer 1` wants to see if any of its buddies are online, a query is issued to `Super-node A`
- When `Peer 1` makes a Skype call, the interaction will proceed to `Super-node A`, and then either to the receiving peer directly (such as `Peer 2`) or to another super-node and then to the receiving peer
- If both peers are on public networks and not behind firewalls, it is possible for those peers to interact directly (e.g., `Peer 2` and `Peer 3`)
- Super-nodes provide directory services and call routing
- `Login Server` is under the authority of Skype.com but any peer can become a super-node; peers get "promoted" to super-node status based on their network and machine performance

Insights from Skype

- A mixed client-server and peer-to-peer architecture addresses the discovery problem, i.e., the network is not flooded with requests in attempts to locate a buddy, as is the case with Gnutella
- Replication and distribution of the directories, in the form of super-nodes, addresses the scalability problem and robustness problem encountered in Napster
- Promotion of ordinary peers to super-nodes based upon network and processing capabilities addresses another aspect of system performance: "not just any peer" is relied upon for important services and as many nodes as are dynamically required can become super-nodes
- A proprietary protocol employing encryption provides privacy for calls that are relayed through super-node intermediaries
- Restriction of participants to clients issued by Skype, and making those clients highly resistant to inspection or modification, prevents malicious clients from entering the network, avoiding the Gnutella problem

Resource Trading P2P: BitTorrent

- The primary goal is to support the speedy replication of large files on individual peers, upon demand
- The aim is to maximize use of all available resources in the network of interested peers in order to minimize the burden of any one participant, thus promoting scalability
- Unlike Napster and Gnutella, BitTorrent distributes parts of a file to many peers, thus also distributing processing and networking loads
- A peer does no obtain a requested large file from a single resource; rather the pieces of the file are obtained from many peers and then reassembled
- Moreover, a peer is obliged to also upload the portions of the file that it has to other interested peers (failure to do so leads in the other peers deprioritizing this peer's access to the parts of the file it needs)
- This strategy prevents a peer from becoming the recipient of a very large number of requests (possibly more that the machine can support) for a popular resource, as is the case with Napster and Gnutella

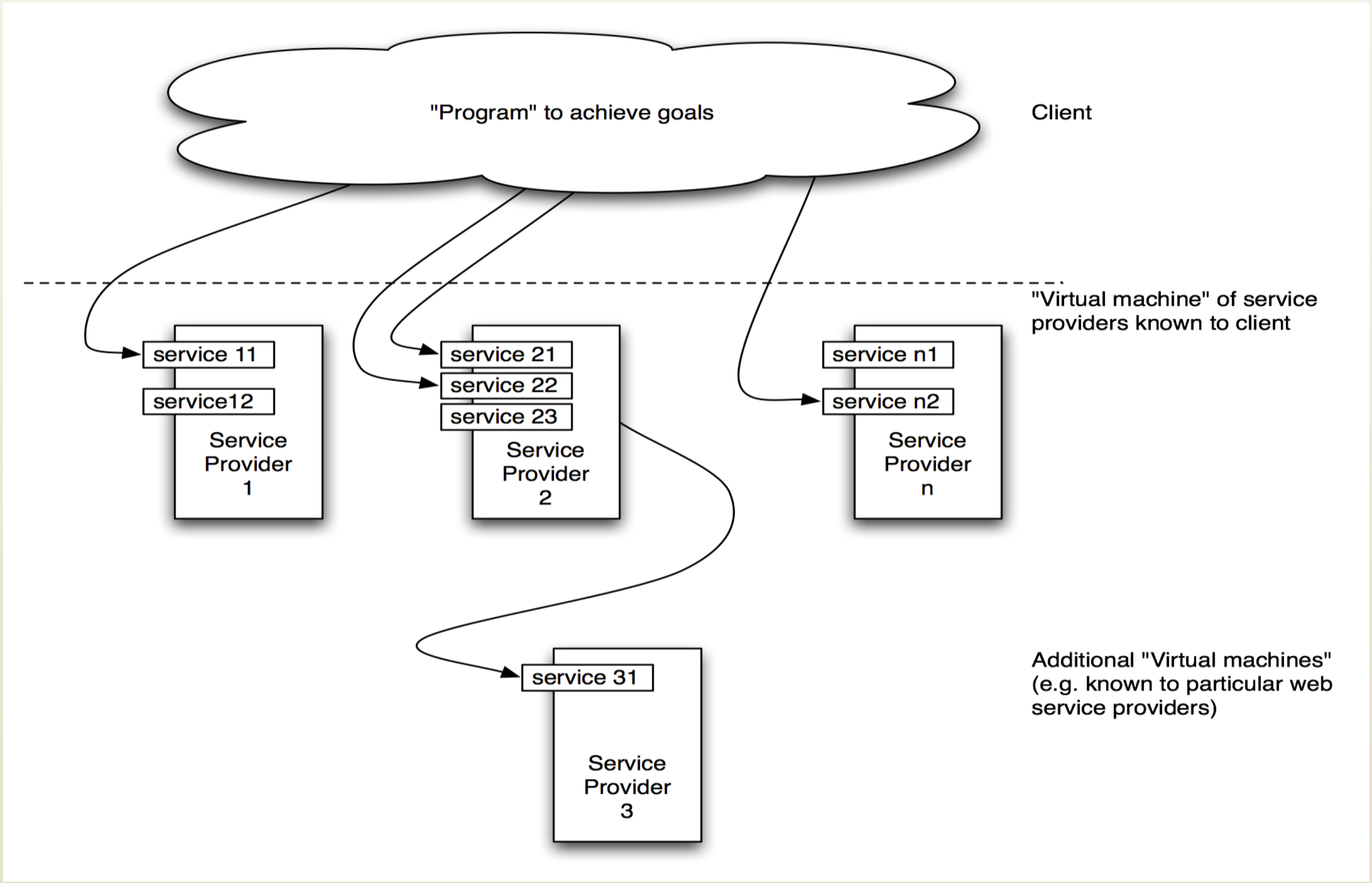
Insights from BitTorrent

- Responsibility for the discovery of content is outside the scope of BitTorrent; a user may use search engines to locate content
- A designated (centralized) machine called the *tracker* is used to oversee the process by which a file is distributed to an interested set of peers
  - The tracker does not perform any file transfers; it merely provides information to other peers so that the latter can communicate between them and initiate downloads
- Meta-data that describes how large files are "pieced", the attributes of those pieces, and the location of the tracker, are associated with a file
- Each peer participating in a file's replication, determines
  - What piece of the file to download next
  - Which peer to obtain that piece from
- All participating peers maintain knowledge of which peers have which pieces

Web Services

- The notion is that business `A` could obtain some processing service `b` from vendor `B`, service `c` from vendor `C`, service `d` from vendor `D`, and so on
  - The service that `A` obtains from `C` might be based on the result obtained from `B` in a preceding interaction
- Service-oriented architectures (SOA) must deal with all the network issues of distributed systems, plus the trust, discovery, dynamism issues present in open, decentralized systems
- From an architectural perspective, participating organizations on the Internet present a virtual machine layer of services which users (client program) can use

Web Services (cont'd)


"Program" to achieve goals
Client
"Virtual machine" of service providers known to client
service 11
service12
Service Provider 1
service 21
service 22
service 23
Service Provider 2
service n1
service n2
Service Provider n
service 31
Service Provider 3
Additional "Virtual machines" (e.g. known to particular web service providers)

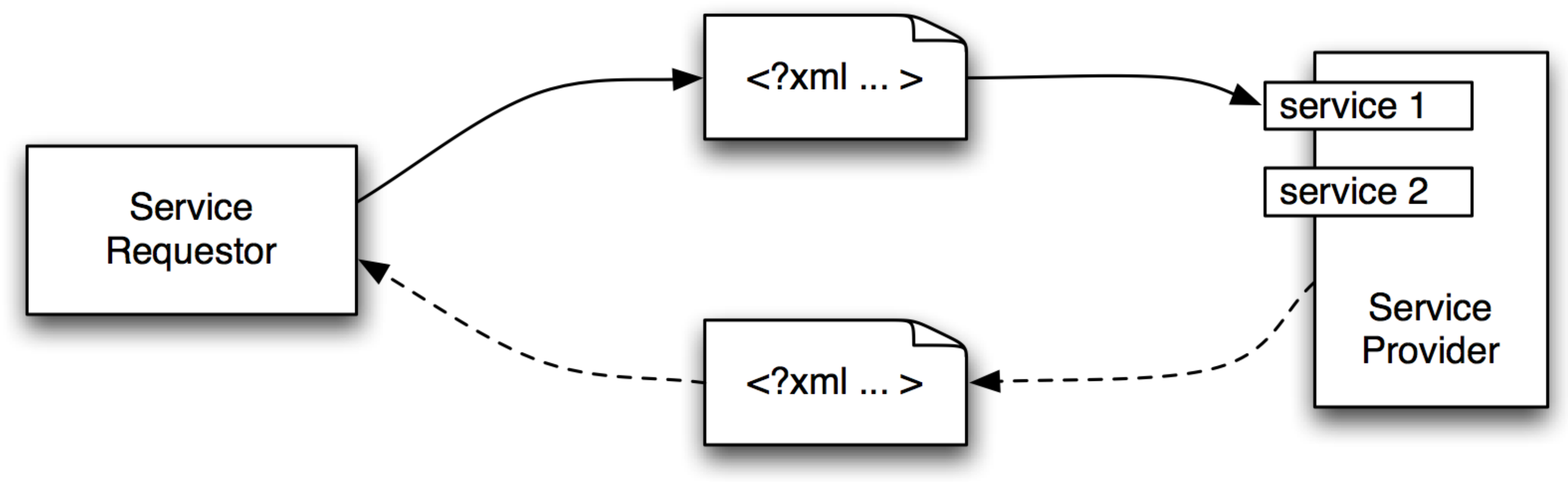
Web Services (cont'd)

- In the previous figure, a client has an objective, such as obtaining all the reservations, tickets, and travel advances associated with an itinerary, and calls upon various services (some of which may call other services)
- However, this virtual machine is different than the one we examined in chapter 4
  - Services (corresponding to functions in the classical virtual machine architecture), are offered by different agencies
  - They may be implemented in various ways, come and go over time, pose carrying risks, etc.

Web Services (cont'd)

- How, architecturally, do we realize the notion of creating business applications?
  - Describe the components, i.e., the services
  - Determine the types of connectors to use, i.e., how the services will communicate and interact
  - Describe the application as a whole, i.e., how the various services are orchestrated to achieve the business goals
- Connectors are usually realized as asynchronous event notifications

Web Services (cont'd)



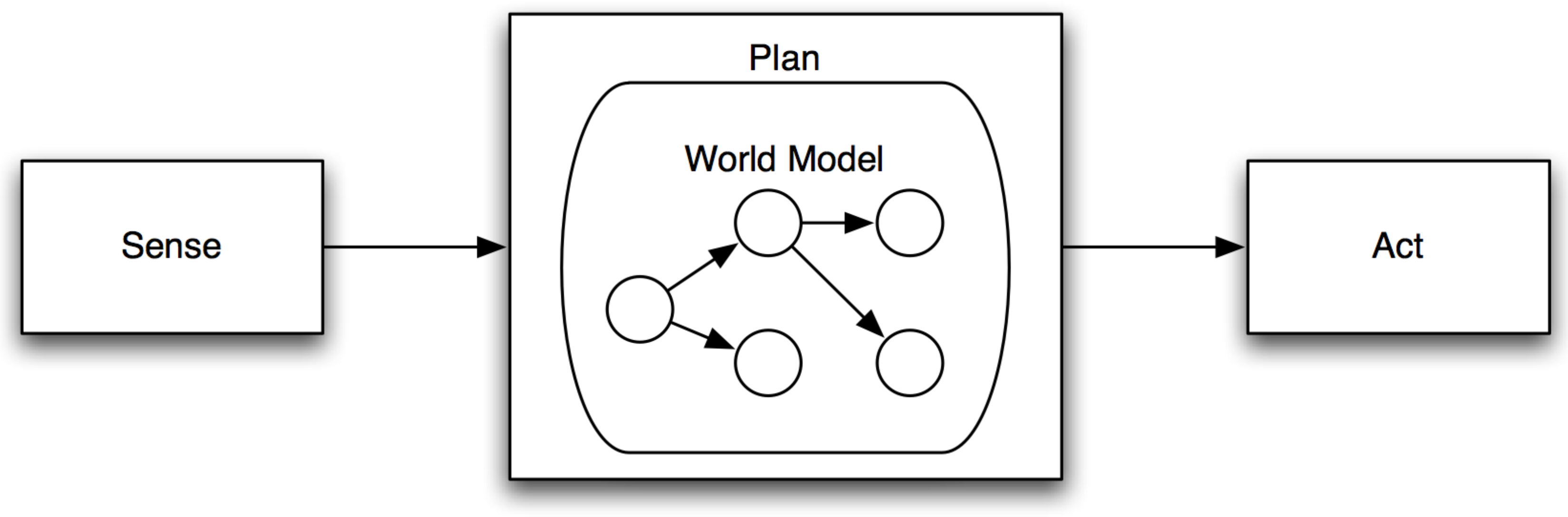
Web Services (cont'd)

- The top part of the previous figure shows a service requestor sending an XML document to a service provider across the network by any of a variety of protocols (e.g., e-mail, HTTP, etc.)
- The XML document must be structured, so that both parties understand it
- The bottom part shows the response of the provider
- However, in general, there is no obligation for the provider to return anything to the requestor

Mobile Robotics

- Manned or partially manned vehicles
- Uses
  - Space exploration
  - Hazardous waste disposal
  - Underwater exploration
- Issues
  - Interface with external sensors & actuators
  - Real-time response to stimuli
  - Response to obstacles
  - Sensor input fidelity
  - Power failures
  - Mechanical limitations
  - Unpredictable events

Robotics: Sense-Plan-Act



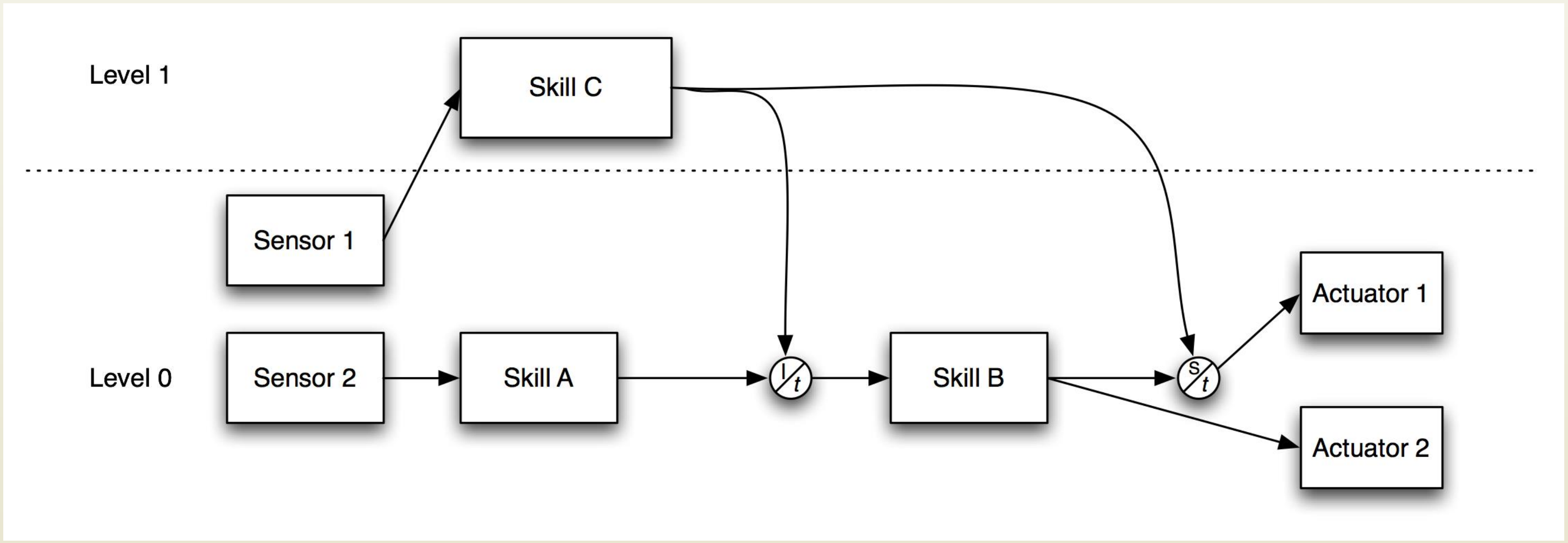
Robotics: Sense-Plan-Act (cont’d)

- The SPA architecture identifies the necessity of using continuous feedback from the environment of a robot as an explicit input to the planning of actions
- The `Sense` component is responsible for gathering sensor information from the environment
- The `Plan` component uses this input to determine which actions the robot should perform
- These actions are then communicated to the `Act` component for execution
- Planning involves the use of sensory data to reconcile the robot’s actual state with an internal model of the robot’s state in the environment; this internal model is repeatedly updated in response to newly acquired sensory inputs

Robotics: Sense-Plan-Act (cont'd)

- The main drawback of the SPA architecture is that sensor information must be integrated and incorporated into the robot's planning models in order for actions to be determined <u>at each step</u> of the architecture's iterations
  - These operations are quite time consuming and usually cannot keep up with the rate of environmental change
  - The performance of this iterative model-update-and-evaluation does not scale well as robotic system capabilities and goals expand

Robotics Subsumption Architecture



Robotics Subsumption Architecture (cont’d)

- This architecture makes the fundamental architectural decision to abandon complete world models and plans as the central element of a robotic system
- There is no explicit planning step between reading from sensors and sending commands to actuators
- Instead, there exist a number of independent components, each encapsulating a specific behavior or robot skill
- These components are arranged into successively more complex layers that communicate through two operations
    - *Inhibition*, to prevent input to a component
    - *Suppression*, to replace the output of a component

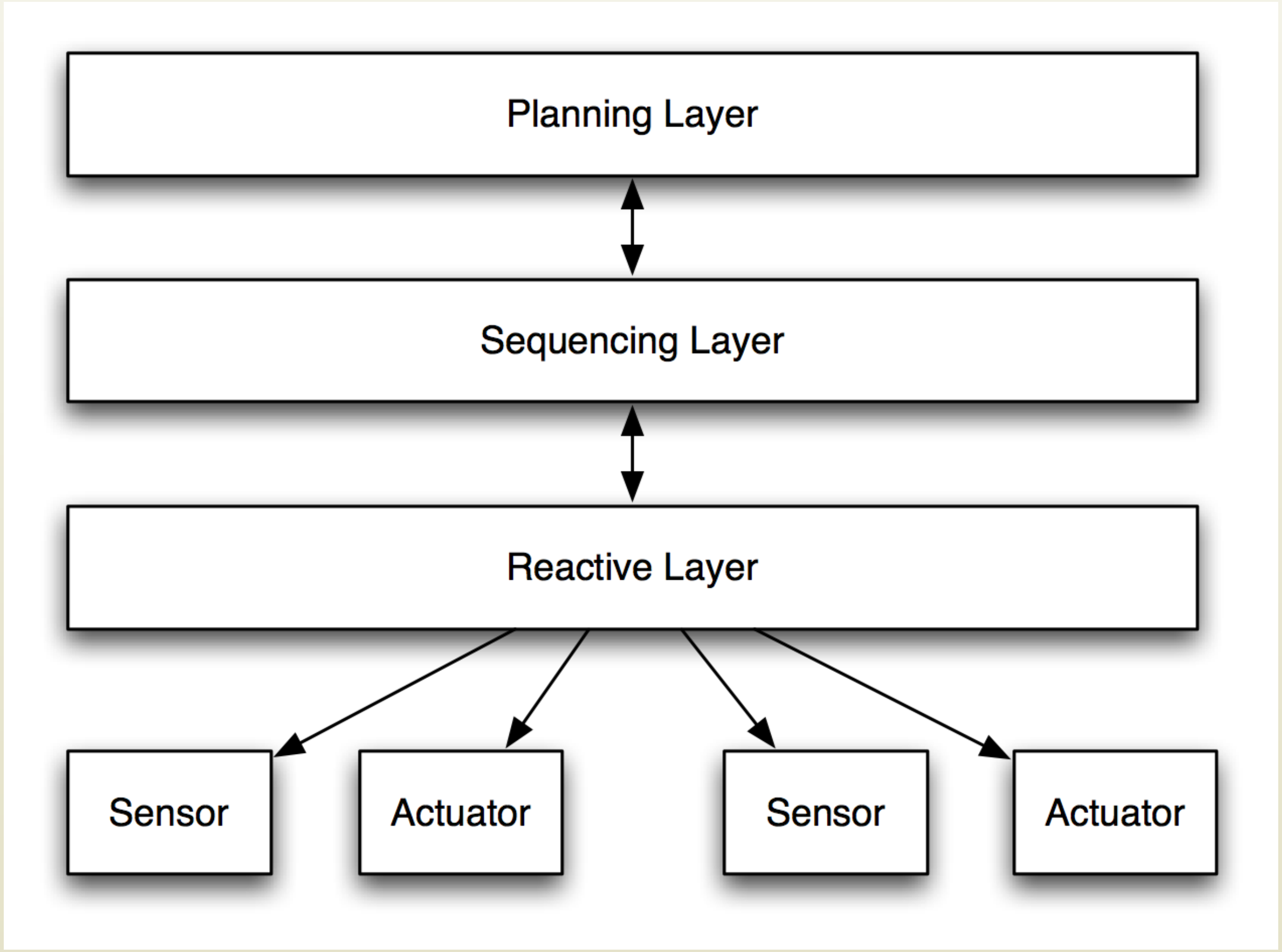
Robotics Subsumption Architecture (cont'd)

- Output from `Skill C` may cause the input from `Skill A` to `Skill B` to be delayed by time t
- Output from `Skill C` may override the output of `Skill B` to `Actuator 1` for time t
- The subsumption architecture modularizes robot behavior and functionality
    - Instead of behavior being represented in a single model (as in SPA), independent components arranged in layers each capture one facet of the overall behavior
    - Each of these components independently relies on sensory inputs in order to trigger the actions it is supposed to perform
    - Overall robot behavior doesn't depend on a central plan

Robotics Subsumption Architecture (cont’d)

- Consequently, subsumption architectures are more reactive in nature, which means they have better performance than SPA
- However, they also have the drawback that there is no coherent architectural plan for layering
- Specifically, there is no explicit guidance or support on how to define different layers
  - Components are inserted into the data flow depending on their specific task, without their position necessarily being related to the layer within which they are positioned
  - Also, components belonging to the same layer are not inserted in similar manners and positions

Robotics: Three-Layer



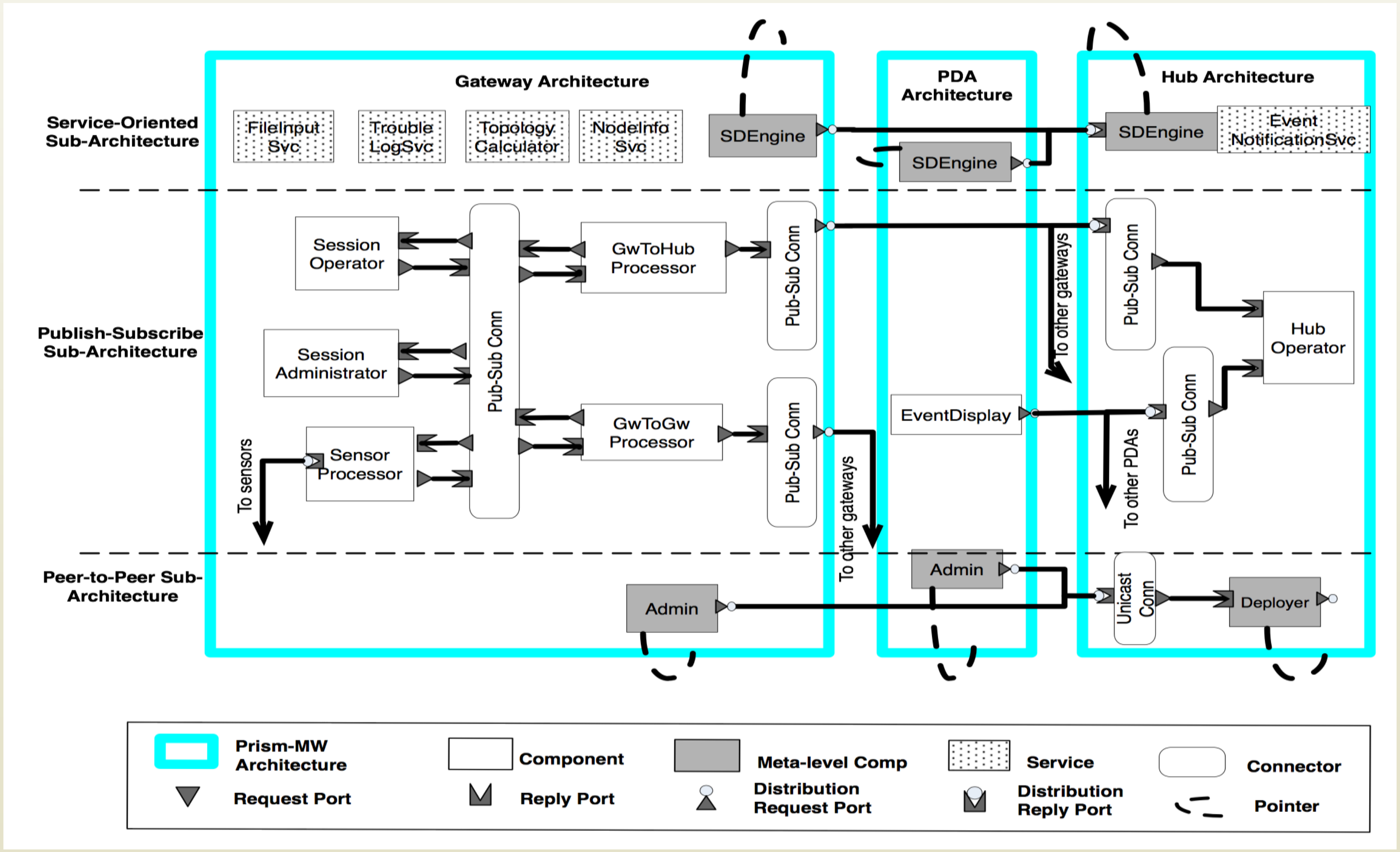
Robotics: Three-Layer (cont'd)

- 3L architectures are characterized by the separation of robot functionality into three layers
  - The `Reactive Layer` quickly reacts to events in the environment with quick actions
  - The `Sequencing Layer` is responsible for linking functionalities present in the reactive layer into more complex behaviors
  - The `Planning Layer` performs slower long-term planning
- 3L combine both reactive operations (as in subsumption architectures) and long-term planning (as in SPA)
- A challenge in 3L is understanding how to separate functionality into the three layers
  - This separation depends on the robot's intended role

Wireless Sensor Networks

- WSN are largely reactive applications
  - Their first task is to monitor the environment and report on its state
- They are used in a variety of domains
  - Medical systems, navigation, industrial automation, etc.
- Benefits
  - Low installation costs, inexpensive maintenance, easy reconfiguration
- MIDAS WSN architecture applies three different architectural styles
  - The P2P portion is responsible for deployment activities, including exchanging application-level components
  - The publish-subscribe portion is responsible for the routing and processing of sensor data
  - The service-oriented portion supports generic services

MIDAS Wireless Sensor Networks Architecture



Summary

- Architectures for complex applications result from
  - Deep understanding of the application domain
  - Careful choice of constituent elements and styles based upon experience and their known properties
  - Hybridization of these elements into a coherent solution
- REST is a typical example of an architecture that creates a complex style from the combination of elements from simpler styles as well as coping with issues of latency, security, etc.
- P2P architectures highlighted problems related to discovering peers, searching for resources, and dealing with performance issues
- Successful commercial architectures, such as Google, Akamai or Skype, demonstrate that scalability need not be a serious problem