SSC0156 – Computação Pervasiva

Chapter 3

Smart Devices and Services

Prof. Jó Ueyama

Related Links

- Basic Distributed Computer Interaction Models in this chapter are the basis for more advanced systems in later chapters, e.g., EDA Architecture can be used for:
  - Sense & Control systems (Chapter 6)
  - Context-based Systems (Chapter 7)
  - Reflexive Intelligent Systems (Chapter 8)
- Mobile Distributed Systems (Chapter 4)
- Management of Distributed Systems (Chapter 12)
- Advances in Distributed Systems (Chapter 13)

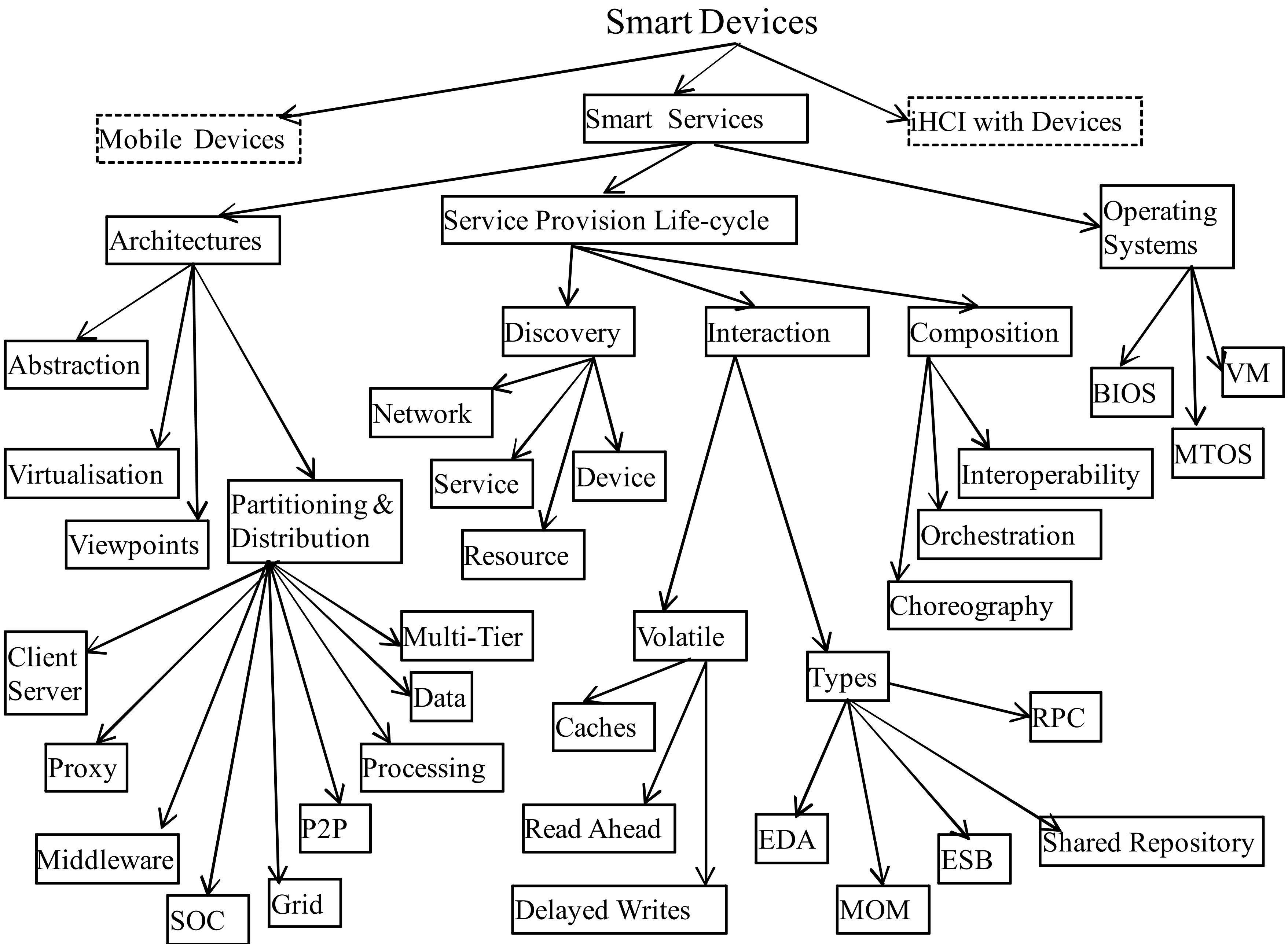
Chapter 3 Slides

The slides for this chapter are also expanded and split into several parts in the full pack

- **Part A: System Architectures**
- **Part B: Middleware, SOC & P2P**
- **Part C: Service Provision Life-cycle & Service Discovery**
- **Part D: Service Invocation**
- **Part E: Volatile Service Invocation & Service Composition**
- **Part F: MTOS, BIOS & VM √**

Overview

- **Smart Device and Service Characteristics √**
- Distributed System Viewpoints
- System Abstraction
- Partitioning and Distribution of System Components
- Proxies and Middleware
- Service Oriented Computing (SOC) & Grid Computing
- Peer-to-Peer Systems (P2P)
- Service Provision Lifecycle √
- Service Discovery
- Service Invocation
- Service Composition
- MTOS, BIOS & VM


Smart Devices
Mobile Devices
Smart Services
iHCI with Devices
Architectures
Service Provision Life-cycle
Operating Systems
Abstraction
Virtualisation
Viewpoints
Partitioning & Distribution
Discovery
Network
Service
Device
Resource
Interaction
Volatile
Caches
Read Ahead
Delayed Writes
Types
RPC
EDA
MOM
ESB
Shared Repository
Composition
Interoperability
Orchestration
Choreography
BIOS
MTOS
VM
Client Server
Proxy
Middleware
SOC
Grid
P2P
Processing
Data
Multi-Tier

Smart Device Characteristics

- Multi-purpose ICT devices, operating as a single portal to multiple remote vs. local application services
- Usually personalised devices, specified owner.
- Locus of control and user interface resides in the smart device.
- Main characteristics of smart devices: mobility, open service discovery, intermittent resource access.
- Important type of smart device is smart mobile device
- Here, we focus on design issues for the service model used by UbiCom Applications

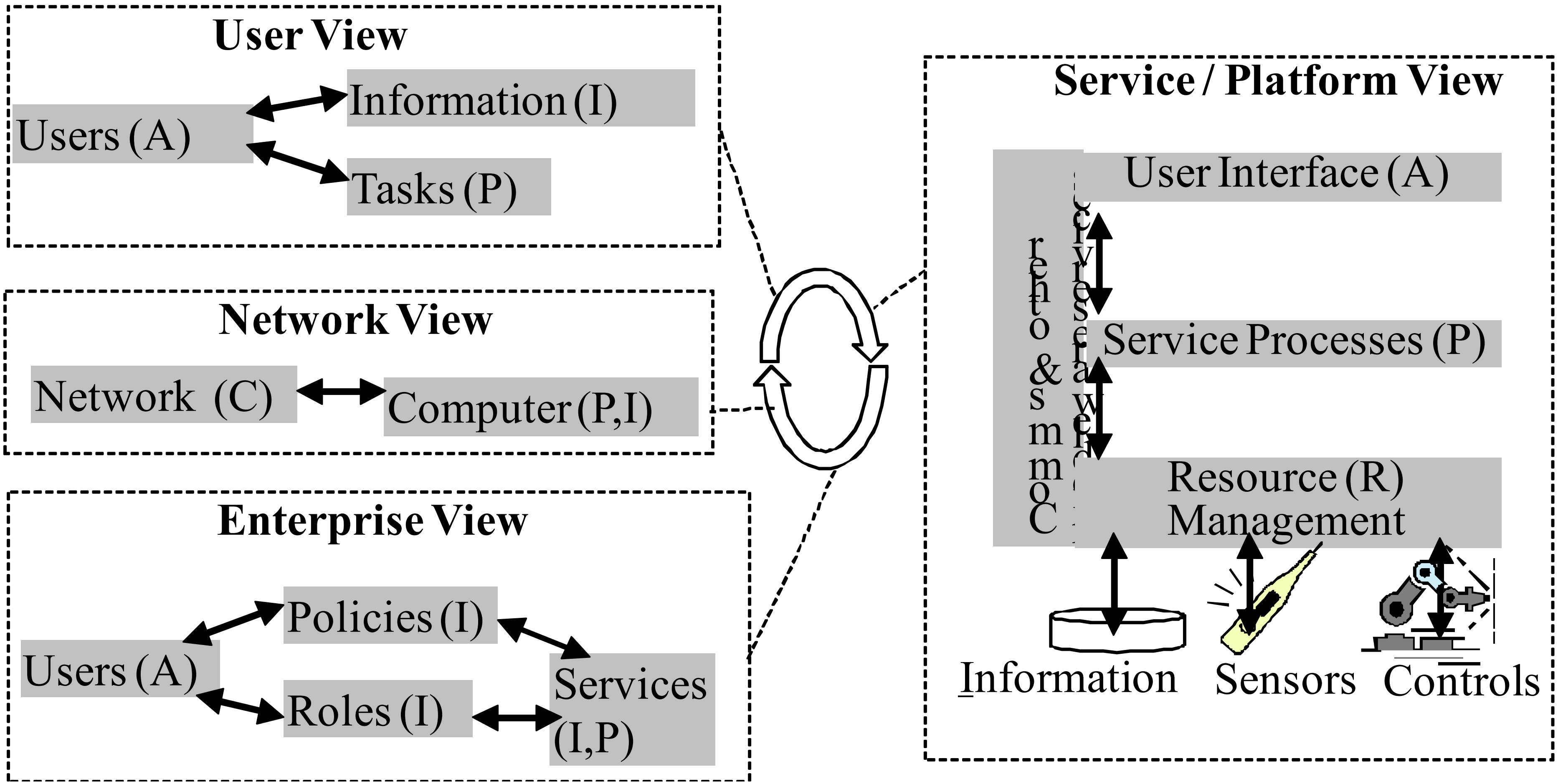
Overview

- Smart Device and Service Characteristics
- **Distributed System Viewpoints √**
- System Abstraction
- Partitioning and Distribution of System Components
- Proxies and Middleware
- Service Oriented Computing (SOC) & Grid Computing
- Peer-to-Peer Systems (P2P)
- Service Provision Lifecycle √
- Service Discovery
- Service Invocation
- Service Composition
- MTOS, BIOS & VM

Distributed System Viewpoints

- Distributed ICT Systems can be modelled from multiple complementary viewpoints with respect to:
- Viewpoints can be regarded as architectural patterns, conceptual models that capture the essential elements of an ICT system architecture and its interrelationships. Multiple viewpoints:
  - Individual user view
  - Enterprise user view:
  - Information system, service or computation platform view:
  - Network view: network elements and computer nodes
- Viewpoint model standards: RM-ODP (ISO), IEEE 1471 model

Distributed System Viewpoints



A = Access/presentation, I = Info./data, P = Processing/computation, C=Comms/ networking

Overview

- Smart Device and Service Characteristics
- Distributed System Viewpoints
- **System Abstraction √**
- Partitioning and Distribution of System Components
- Proxies & Middleware
- Service Oriented Computing (SOC) & Grid Computing
- Peer-to-Peer Systems
- Service Provision Lifecycle √
- Service Discovery
- Service Invocation
- Service Composition
- MTOS, BIOS & VM

Reducing System Complexity using Abstraction (Modularisation)

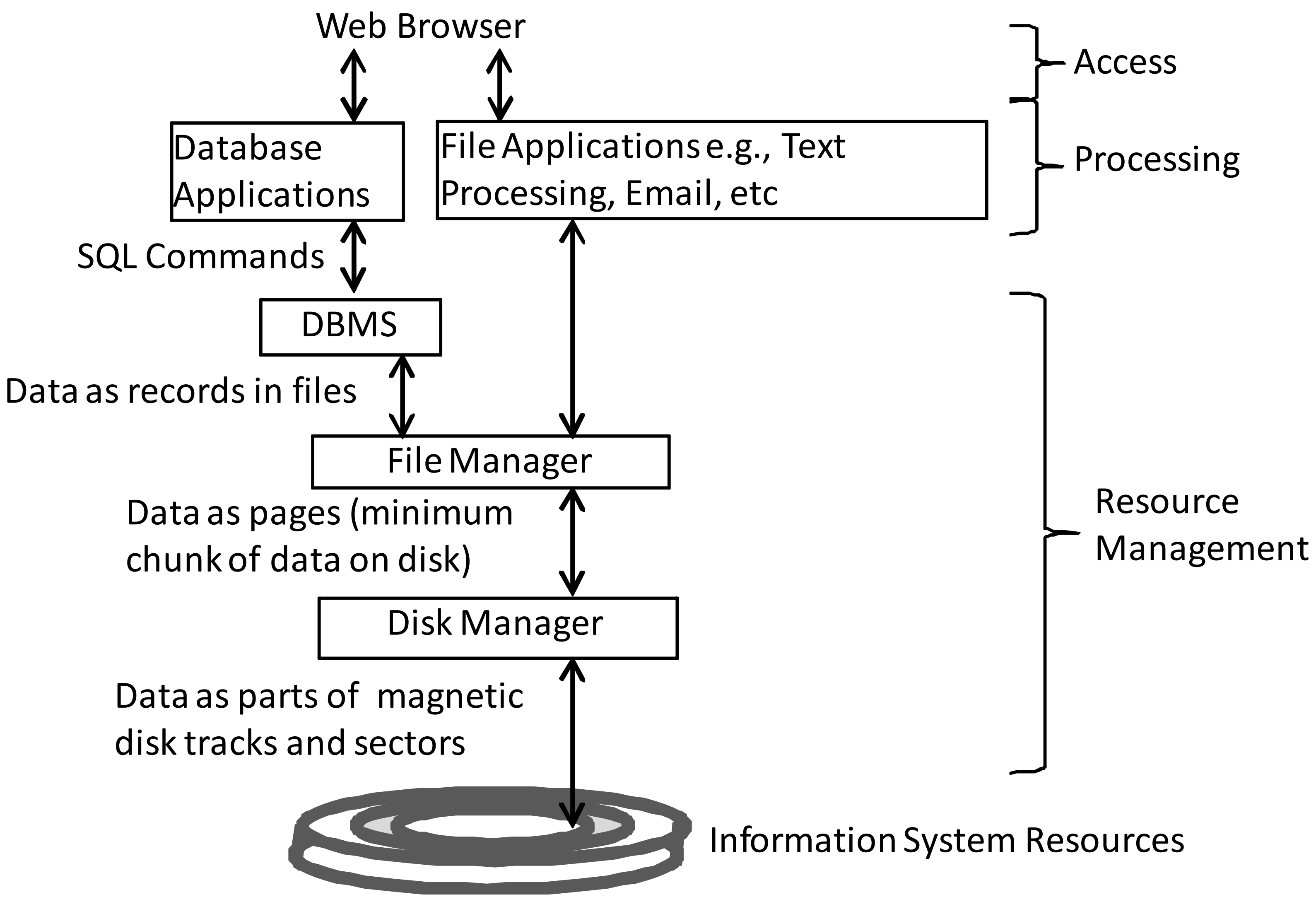
- System architectures focus on the idea of reducing complexity through both a separation of concerns using *modularisation* & *transparency*
- Two common criteria for modules:
  - *high cohesion*
  - *loose-coupling*
- Meyer (1998) uses five criteria for modularisation:
  - *Decomposability*:
  - *Composability*:
  - *Understandability:*
  - *Continuity*:
  - *Protection*:.

Reducing System Complexity using Abstraction (interoperability)

- Abstractions define those things that are important in a system
- Abstractions are employed to facilitate the interaction to a system

- Abstraction that simplifies the view or access to internal functionality to the outside, is also called an *interface*.

System View: Example of Abstraction



Reducing System Complexity using Abstraction (Transparency)

- Abstractions make transparent properties not needed by interactions
- Important types of transparency for distributed services include:
  - Access transparency:
  - Concurrency transparency
  - Failure transparency (Fault Tolerance)
  - Migration transparency
  - Scaling transparency
- In practice, ideal transparency of a single image for all resources, all the time, under all conditions is hard to achieve
  - Usually only when the distributed system is operating normally.

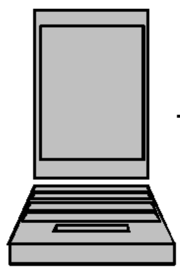
Reducing System Complexity using Abstraction (virtualisation)

- Abstractions alone do not necessarily support interoperability
  - System interfaces designed for a particular platform do not support interoperability
- *Virtualisation* provides a way to solve this limitation of abstraction

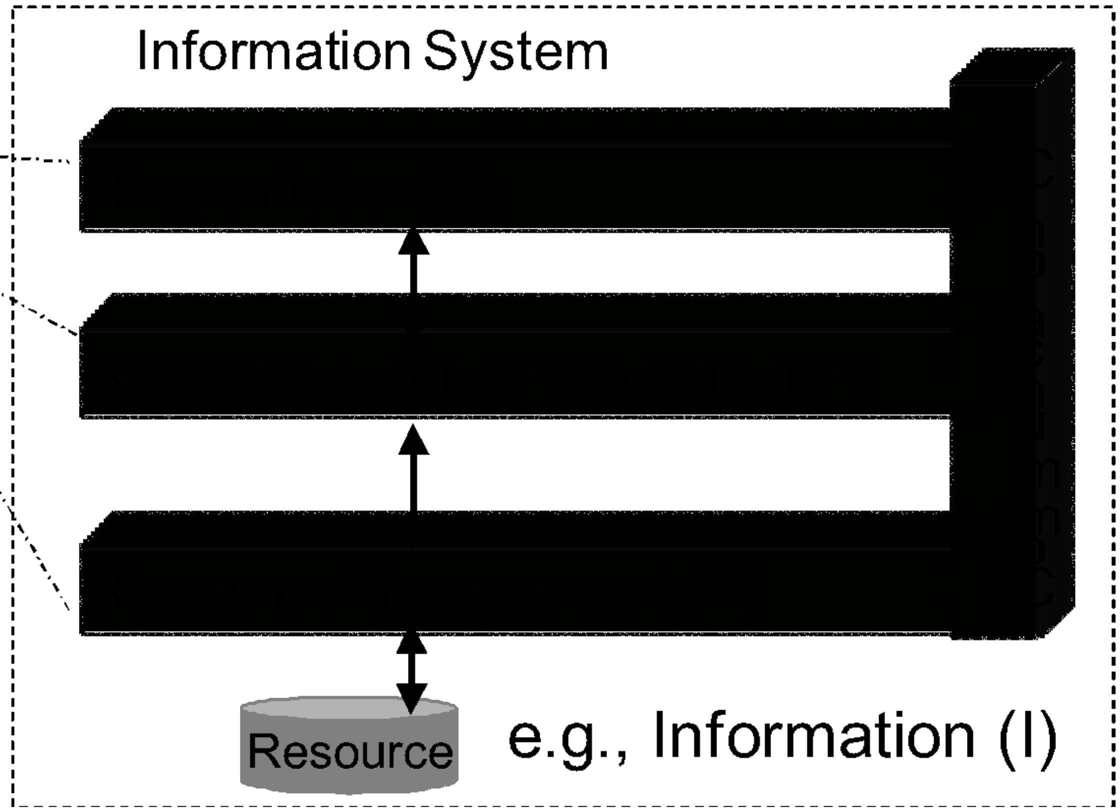
Overview

- Smart Device and Service Characteristics
- Distributed System Viewpoints
- System Abstraction
- **Partitioning and Distribution of System Components √**
- Proxies and Middleware
- Service Oriented Computing (SOC) & Grid Computing
- Peer-to-Peer Systems (P2P)
- Service Provision Lifecycle √
- Service Discovery
- Service Invocation
- Service Composition
- MTOS, BIOS & VM

Partitioning & Distribution of System Components: None



Copy of data or applications downloaded onto device, then it is used off-line



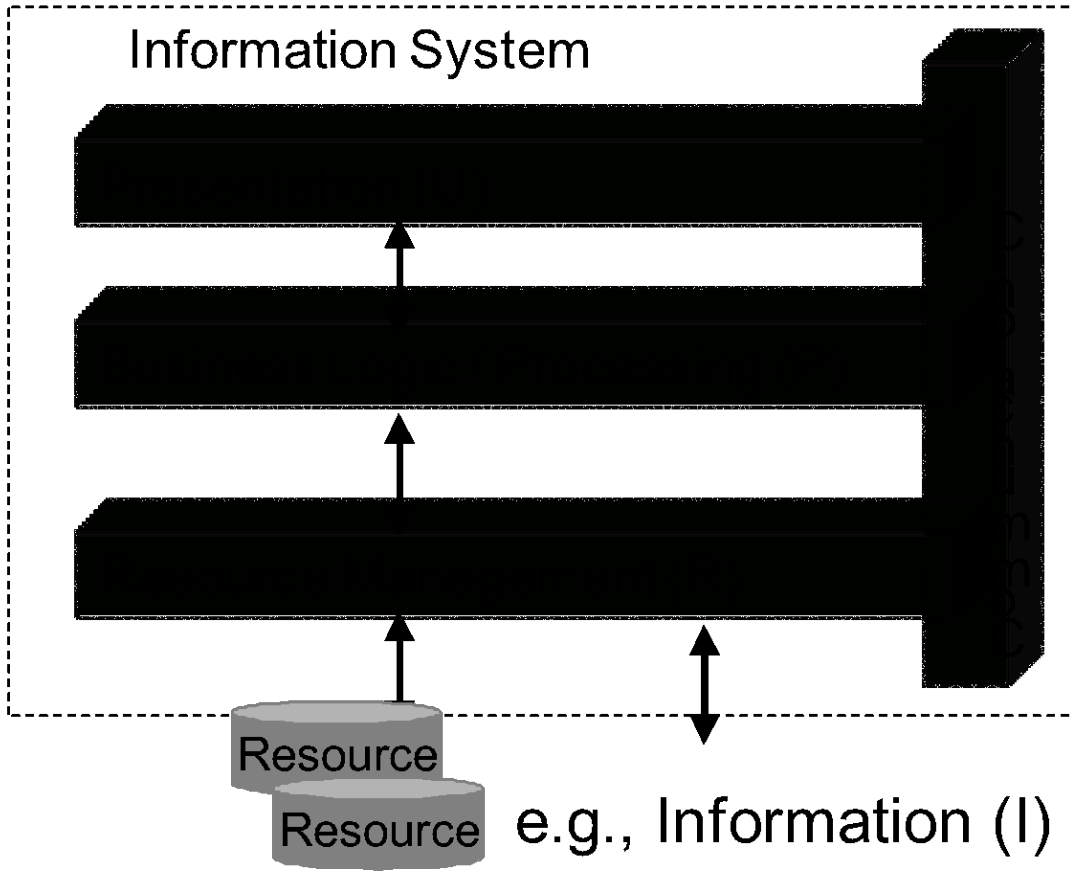
Partitioning & Distribution of System Components: None

- Advantages?

- Disadvantages?

Partitioning & Distribution of System Components

- Ex: how can we distribute these components?



Partitioning & Distributing System Components

- Range of designs for partitioning and distributing services:

- Consider type of access device, resources, communication: several ways to distribute these, e.g.,
- *High resource access devices* can act self-sufficiently,
- *Low / poor resource access devices*

System Architectures: Partitioning Example

Discuss How to partition a 2 player Person versus Machine Chess Application in terms of a client-server design / for use on a mobile device

Network Usage

Low ↔ High

CPU Usage

Low ↔ High

Data Memory Usage

Low High

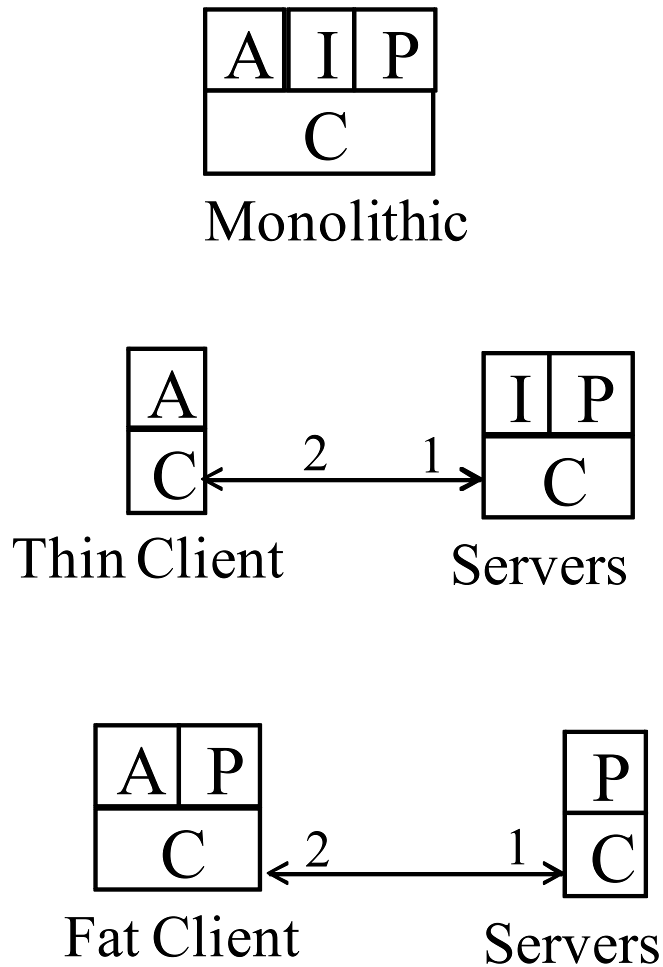
Architectures: Client Server model

- Asymmetric distributed computing model with respect to where resources reside and the direction of the interaction.

Client-server interaction is also asymmetric:

- Asymmetry benefits?
  - Synchronization between clients which starts requesting while servers start waiting for client requests

Partitioning and Distribution: Client-Server Model



Client Server Model

- System configuration (partitioning and distribution) depends upon:
  - network links;
  - local resources,
  - remote service availability;
  - type of application,
  - service maintenance model.
- Different degrees of resources on access devices (clients)
- Resource poor (thin-client server model):
  - reliance on external servers, network supports remote service access on demand

Client Server Model

- Processing needed to adapt content to different types of terminals
  –
- Thin-client server model is often considered to be easier to maintain
  –
- Thin-clients offer very limited application platform

Client Server Model

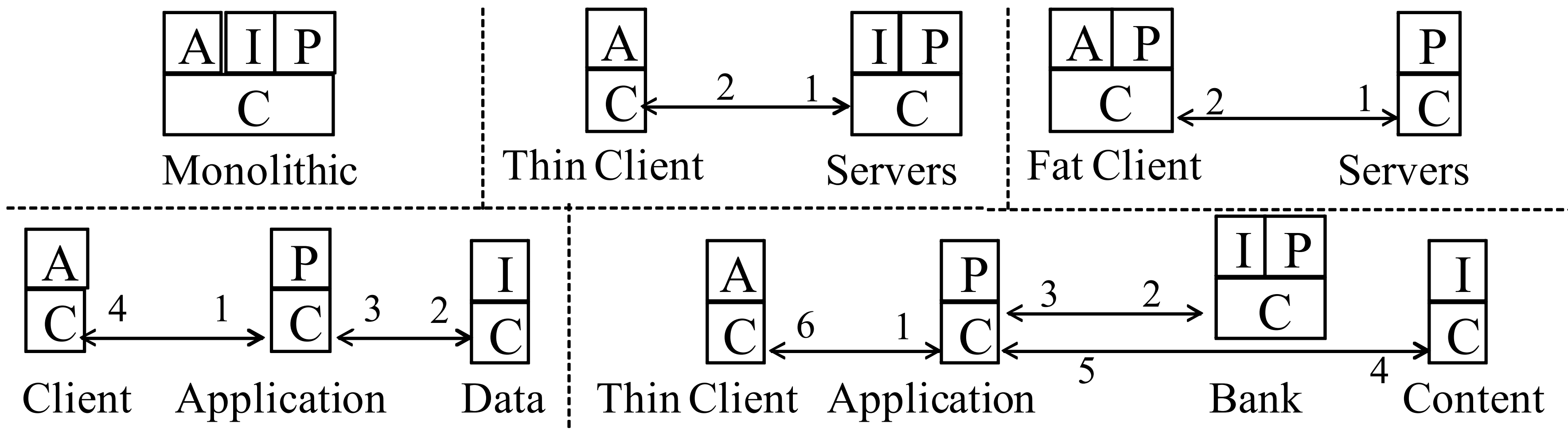
- How to cope with unreliable and low-performance networks using client-server model?
- Argues for a degree of self-reliance & use of local processing and data resources

- Fat client model is suitable when?

- Type of processing in access device depends on type of application.
  - E.g., chess game application

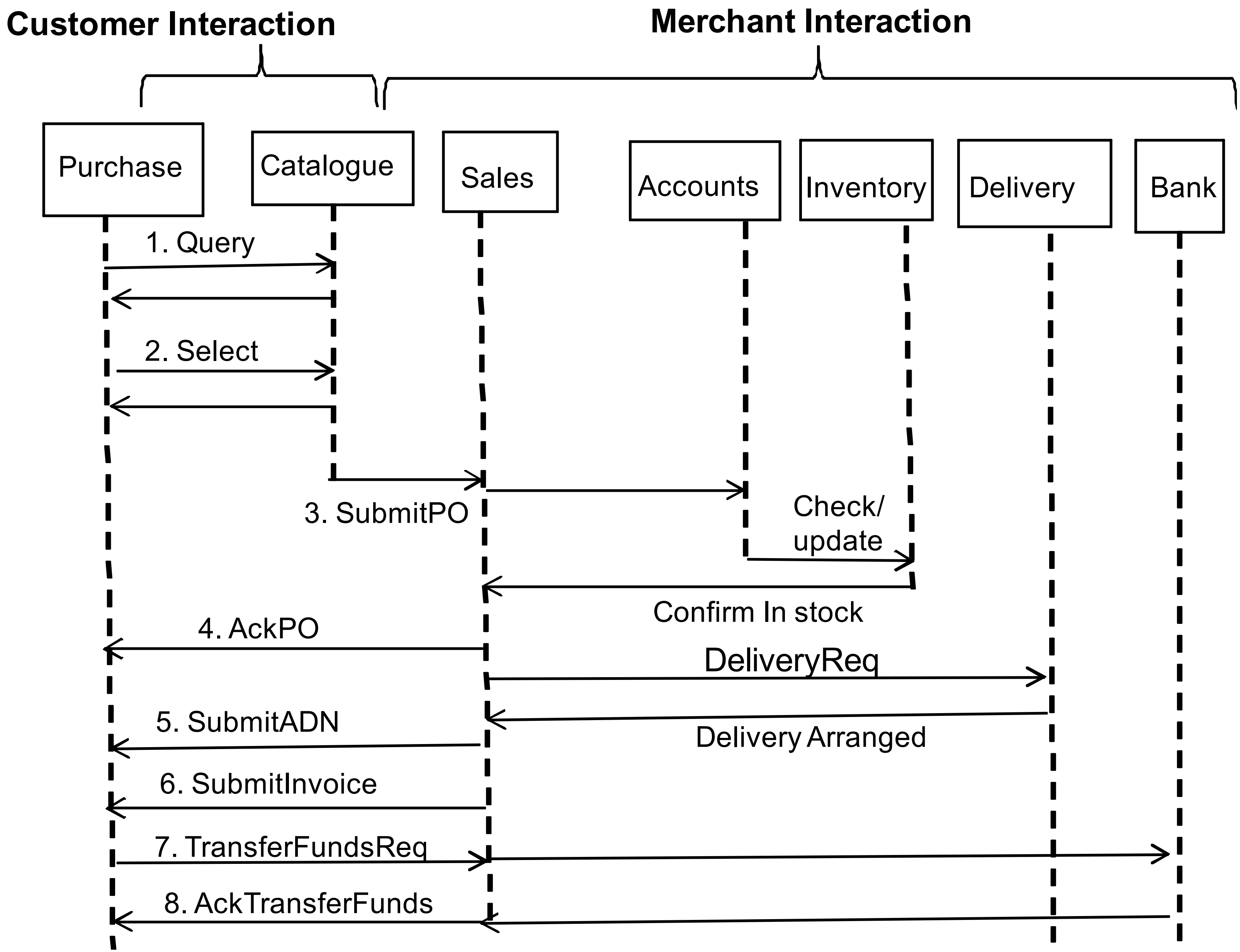
  - E.g., scientific calculation

Partitioning & Distributing System Components: Summary of Models



Partitioning & Distributing System Components: Summary of Models

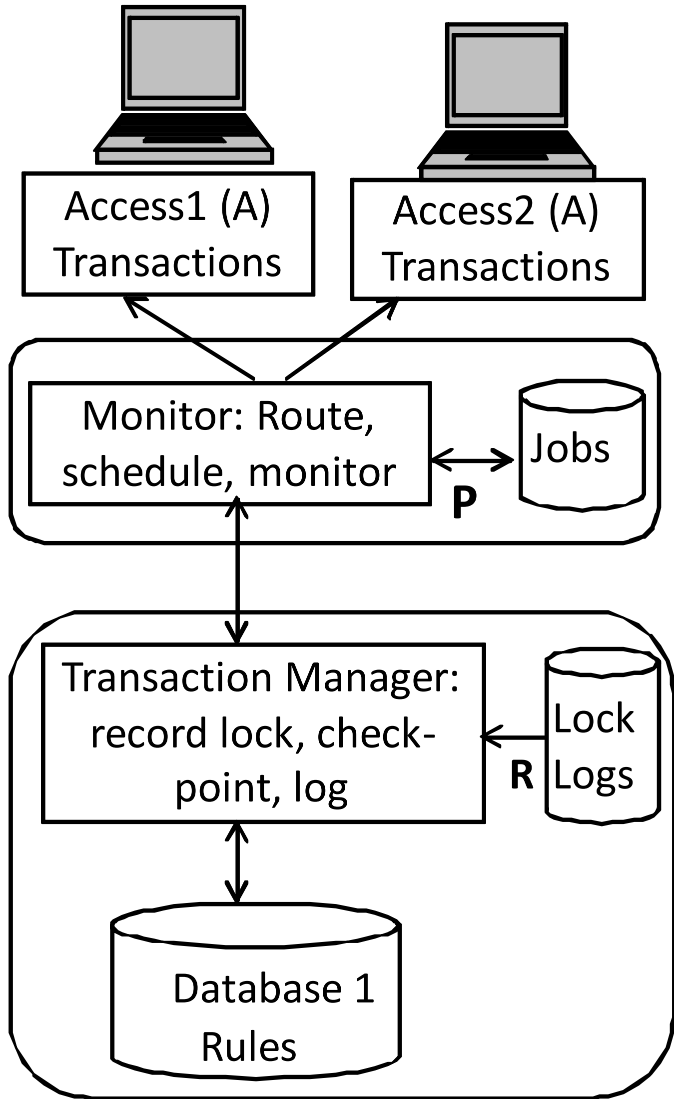
- Different designs for Information-based UbiCom systems:
  - based upon how their A, P and I components are distributed.
- Functions can be distributed over multiple different computer nodes or tiers:
  - 1-tier, monolithic system, appliance model:
  - 2-tier, thin-client server:
  - 2-tier, fat-client server model:
  - Multi-tier (3,4 ... N-Tier) systems:


Customer Interaction
Merchant Interaction
Purchase
Catalogue
Sales
Accounts
Inventory
Delivery
Bank
1. Query
2. Select
3. SubmitPO
Check/
update
Confirm In stock
4. AckPO
DeliveryReq
5. SubmitADN
Delivery Arranged
6. SubmitInvoice
7. TransferFundsReq
8. AckTransferFunds

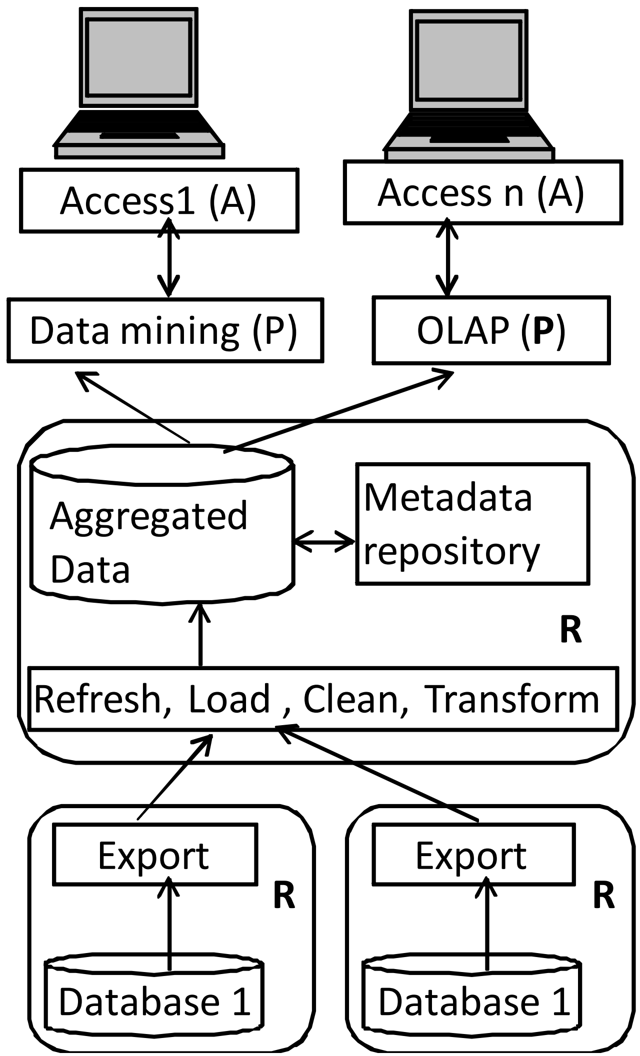
Partitioning and Distributed Data (D) Storage

- I, P, A and D can themselves be partitioned & distributed
- Examples of Partitioned & Distributed D
  - *Transaction Monitors* (TM): distributed data transactions;
  - *Data Warehouses*, centralised analysis of distributed data
  - *Distributed Databases*: distributed queries

Distributed Data (D) Storage: Transaction Processing

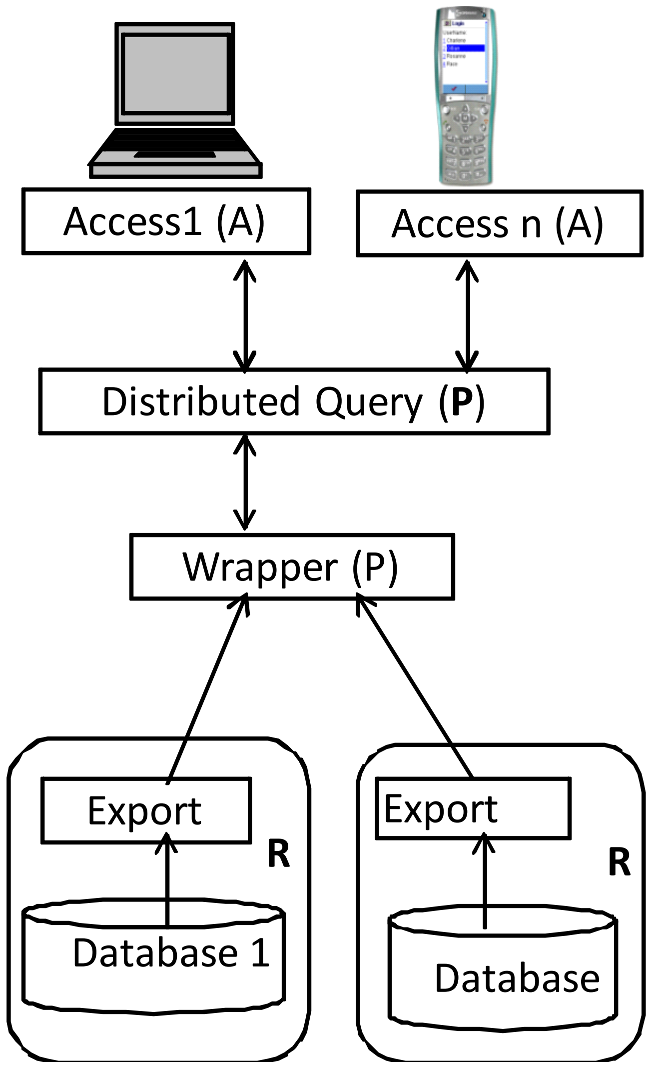


Distributed Data (D) Storage: Data Warehouse



- OLAP Online Analytical processing

Distributed Data (D) Storage: Distributed Database



Distributed Processing

- Partitioning & distributing processing onto multiple CPUs
- Use for computation intensive tasks, e.g., ??
- Time gained in ↓ processing time must be > time to partition & distribute tasks, collect individual results & combine them.

Many different architectures

- Super-computers - specialised multiple CPU systems
- Clusters of networked MTOS computers, e.g., Grids.
- Multiple CPUs in MTOS computers. e.g., multi-core processor
- P2P computing
- Cellular computing

What about

- distributed UIs?
- Distributed communication?

Distributed Processing Architectures

- Examples can be added here

Overview

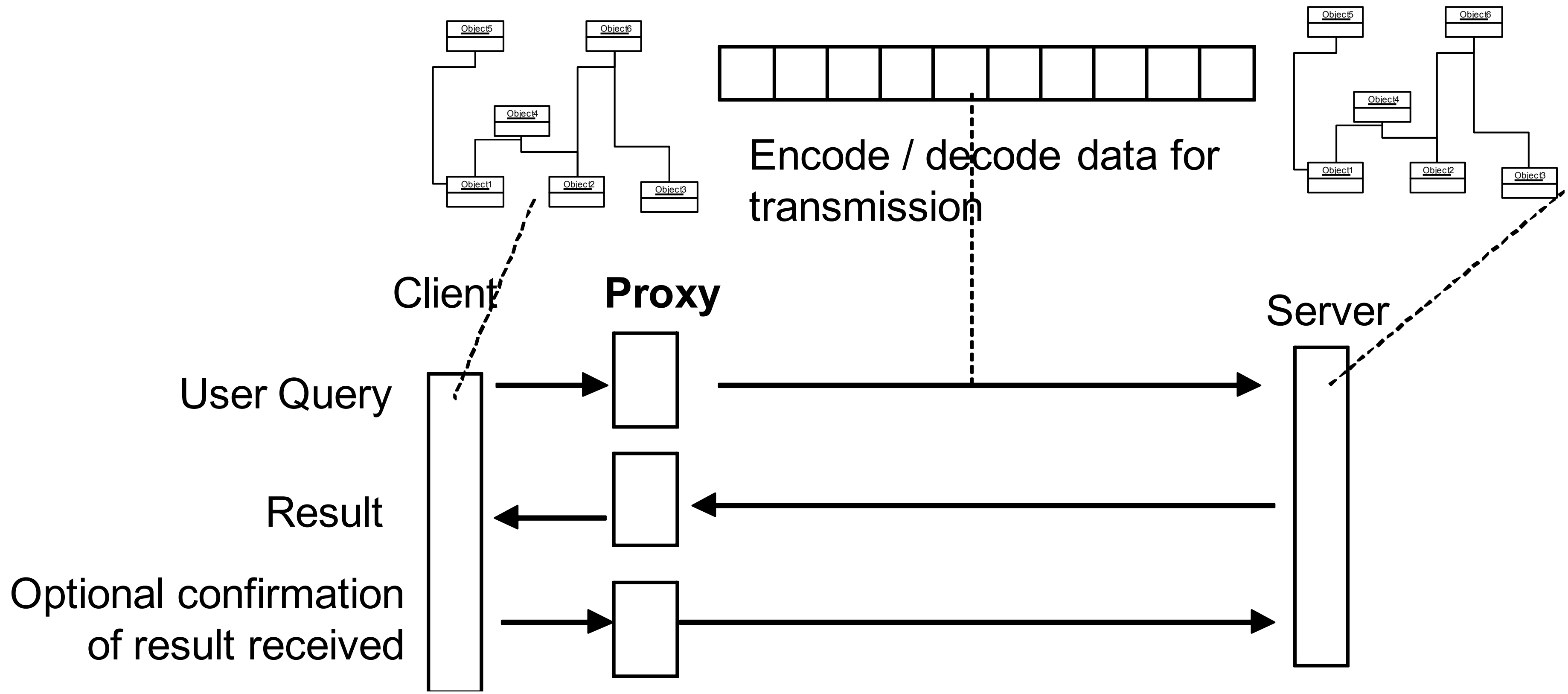
- Smart Device and Service Characteristics
- Distributed System Viewpoints
- System Abstraction
- Partitioning and Distribution of System Components
- **Proxies and Middleware √**
- Service Oriented Computing (SOC) & Grid Computing
- Peer-to-Peer Systems
- Service Provision Lifecycle √
- Service Discovery
- Service Invocation
- Service Composition
- MTOS, BIOS & VM

Proxy based Service Access

Advantages of using client proxies

- Some applications use a client proxy to simplify access processes in client, How?
- *Off-load presentation processing and network processing*
- *Hide heterogeneity* of terminal types & networks from applications
- *Simplify and compose access* to multiple service providers.
- *Reduce complexity of communication* used in access devices, e.g., ??
- *Enable devices to operate intermittently* in a disconnected state.
- *Shield network-based applications from mobility* of access devices (DTN style?)

Proxy based Service Access



Use of proxies to simplify network access by transparently encoding and decoding the transmitted data on behalf of clients and / or servers

Proxy based Service Access

What are the disadvantages of Proxy-based access?

Where does the proxy reside?

Middleware

- ↑ Variety & heterogeneity & complexity of services access
- Middleware introduced in between applications & OS to simplify access to services
- Middleware factors out set of generic services, e.g., database access, file system access etc. to make them:
- Advantages for Application?
  –
- Advantages for OS?

Middleware: Design Issues

- May be useful for applications to have an awareness of lower level interaction, for resource access not to be completely hidden by middleware.
- Why?

Middleware

Application awareness of ICT Context

Full	**Partial**	**None**
Application sees full ICT system interface, no Middleware used	Middleware handles some of the complexity in interfacing to ICT system	Middleware hides complexity of ICT system from application

Overview

- Smart Device and Service Characteristics
- Distributed System Viewpoints
- System Abstraction
- Partitioning and Distribution of System Components
- Proxies and Middleware
- **Service Oriented Computing (SOC) & Grid Computing √**
- Peer-to-Peer Systems
- Service Provision Lifecycle √
- Service Discovery
- Service Invocation
- Service Composition
- MTOS, BIOS & VM

Service Oriented Computing (SOC)

- SOA (Architectures) , also referred to as SOC (Computing)
- Services as computational or information processing components
  - That are autonomous and heterogeneous
  - Can run on different platforms
  - Are possibly owned by different organizations.

SOC Standards

Several different standards for SOC

- (XML based) Web Services
- Computer Grids OGSI
- OASIS SOA RM
- Open Group SOA Working Group
- Semantic Web Services?

Service Oriented Computing (SOC)

Notion of service characterised by:

- *Descriptions*: specification of tasks; discoverable
- *Outcomes*: service is the means to achieve a defined outcome for a task
- *Offers: if an offer is made then the provider is available*
- *Competency*: to undertake the task; regulatory authority
- *Execution*: performing the service on behalf of someone
- *Composition*: Multiple services may need to be composed before they can be executed with respect to an outcome and time constraints
- *Constraints or policies*: for a service, which may be specified either by the user, e.g., for a taxi service 'don't drive too fast', or by the provider 'not exceeding the speed limit

Service Oriented Computing (SOC)

- Services in a SOA can be separated into three layers of functions: basic (lower), composition (middle) and management (higher layer) Service Management
- Service management: Services are managed by third parties, between the user and provider, based upon policies , e.g. SLAs
- Enterprise service bus(Basic function): this supports service, message, and event based interactions with appropriate service levels and manageability;

Service Management (H)
Service Composition (M)
Service Invocation (B)
Service Discovery (B)
Enterprise Service Bus (B)

SOC: Grids

- Grid computing: distributed systems that enable:
  - e large scale coordinated use and sharing of geographically distributed resources
- (Early) Grid computing system design tends to focus on high performance computing rather than fault-tolerance & dynamic ad hoc interaction,

SOC: Grids

Three main types of Grid system occur in practice:

- *Computational Grids*: they have higher aggregate computational capacity available for single applications
- *Data Grids*, provide an infrastructure for synthesising new information from data repositories such as digital libraries or data warehouses
- *Service Grids:* they provide services that are not provided by any single machine

Overview

- Smart Device and Service Characteristics
- Distributed System Viewpoints
- System Abstraction
- Partitioning and Distribution of System Components
- Middleware
- Service Oriented Computing (SOC)
- **Peer-to-Peer Systems √**
- Service Provision Lifecycle √
- Service Discovery
- Service Invocation
- Service Composition
- MTOS, BIOS & VM

Peer-to-Peer Systems (P2P)

P2P can be defined as:

- distributed systems consisting of interconnected nodes
- able to self-organize into network topologies
- with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth,
- capable of adapting to failures and accommodating transient populations of nodes
- while maintaining acceptable connectivity and performance
- without requiring the intermediation or support of a global centralized servers or authorities.

P2P Benefits?

- lower cost of ownership for content sharing
- performance enhancements: the resources of all the nodes can be used for storage, computation and data exchange rather than focusing resources mostly in the server type nodes (equality)
- ad hoc resource utilisation and sharing: as demand for particular services peaks
- autonomous control and ownership
- anonymity and privacy
- fault tolerance: there are no central servers that can be attacked or can cause complete system failure

P2P System Design Challenges?

- *More complex coordination is often needed*. In contrast, with client server interaction
- *Nodes can act as freeloaders*: nodes may be happy to play a role of service requesters but are always configured to refuse the requests
- *More complex security* may be needed as identification can be masked so access control is harder.
- *Ad hoc network routes*: need to create and discover ad hoc routes between nodes
- *Service discovery*: how to discover the selective nodes where services can be invoked from

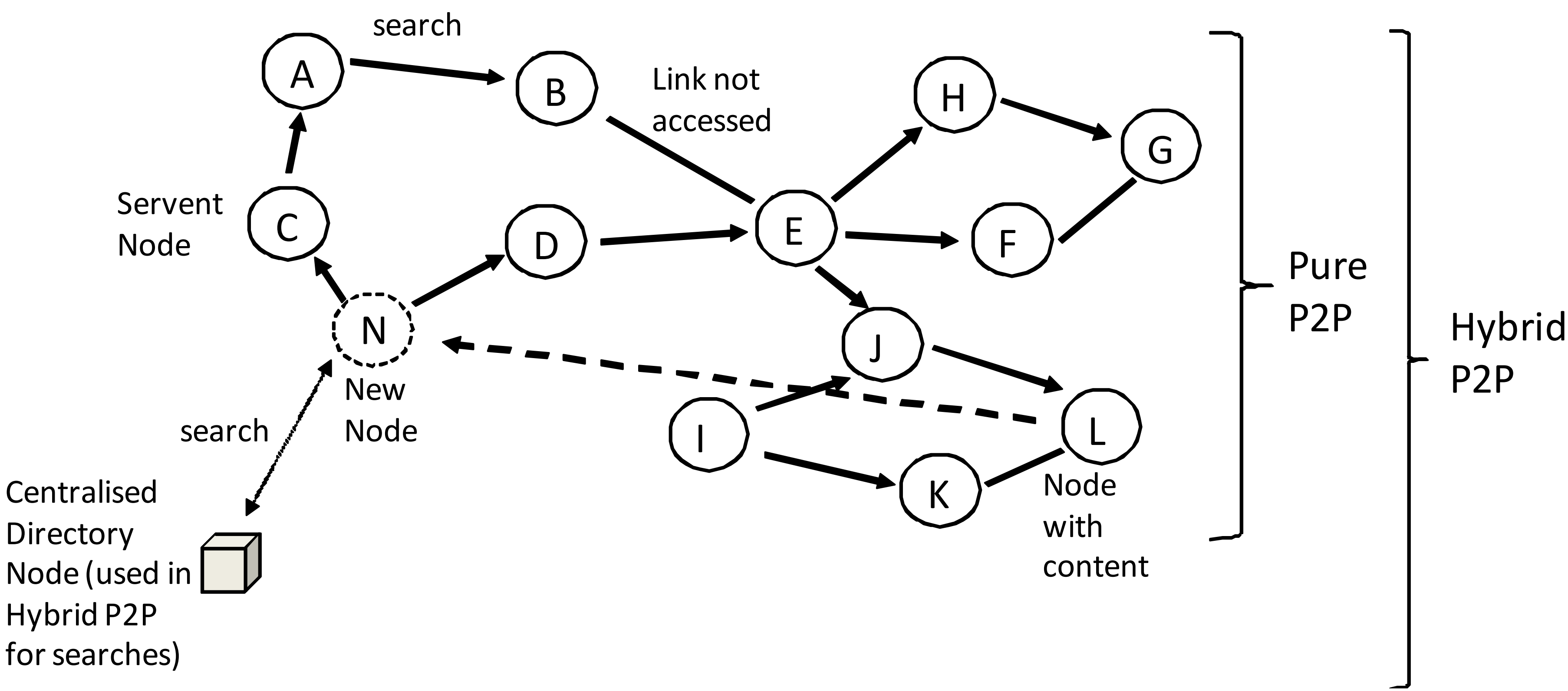
P2P System: Types

- 3 main types of P2P system depending on the types of computer nodes:
- *Pure P2P* uses no notion of fixed clients or servers, only of equal peer nodes that simultaneously function as both dynamic servers and clients (called servents)
- *Partial P2P*: all nodes are not equal, a few superpeers or supernodes are elected to operate as middleware servers, acting as network relays for other nodes
- *Hybrid P2P* networks, a client server organisation is used for specific tasks and interactions, such as searching for services and a P2P organisation is used for others such as service invocation

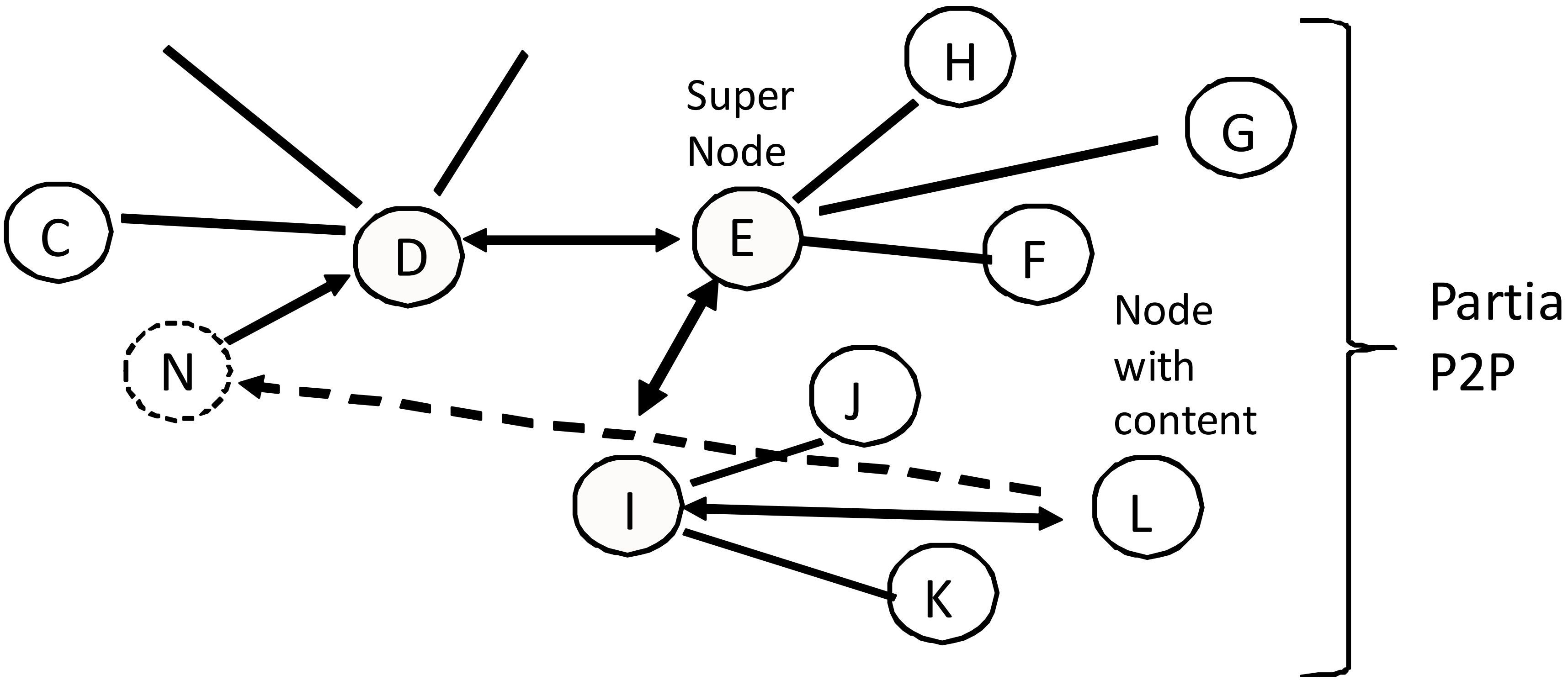
In Hybrid P2Ps

- Three basic steps for accessing content:
  - identify nodes,
  - register content provision nodes,
  - Finally, search & retrieve content

P2P System Types: Pure, Hybrid



P2P System Types : Partial P2P



P2P System Types

- Can also be grouped into two main types of topologies for P2P systems that overlay the underlying physical network:

- *Unstructured overlay networks*: they are like ad hoc networks and they are independent of any physical network topology and use decentralised and partially decentralised nodes

- *Structured overlay networks*: they are dependent on the physical network topology of nodes and use hybrid decentralised nodes

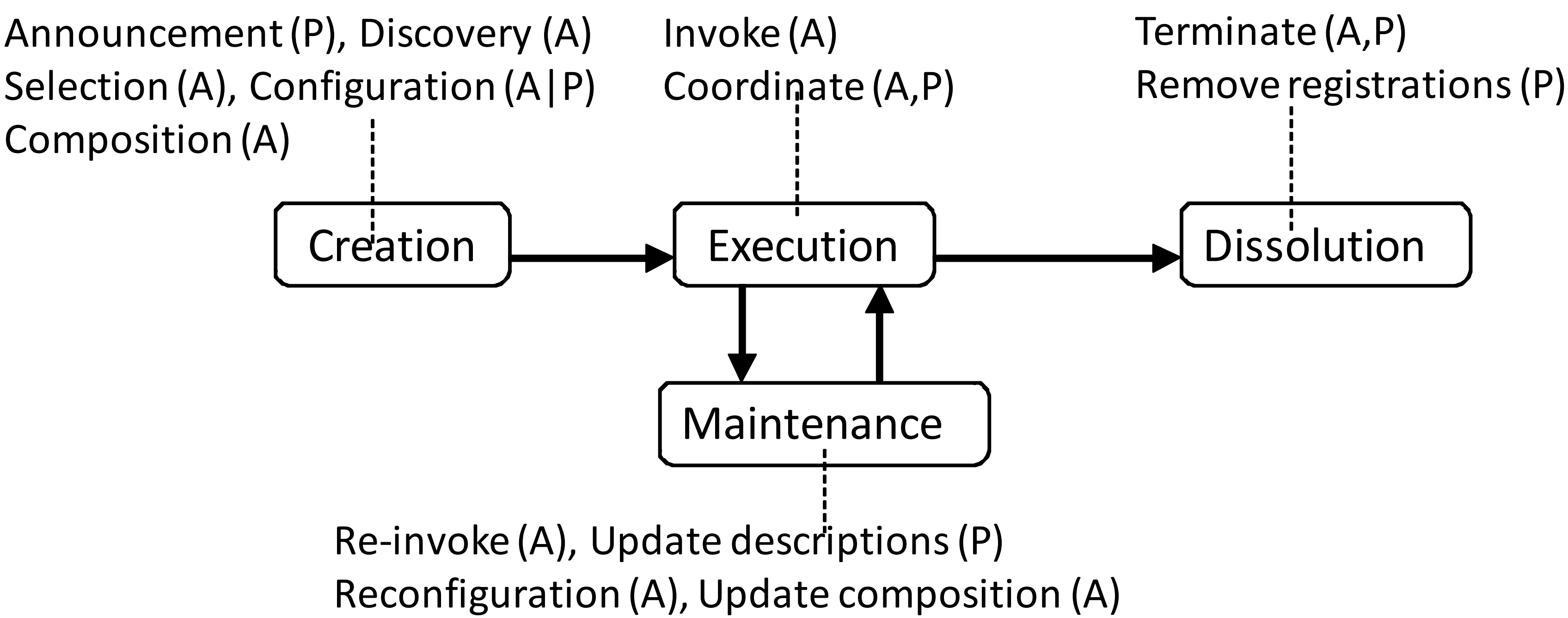
Overview

- Smart Device and Service Characteristics
- Distributed System Viewpoints
- System Abstraction
- Partitioning and Distribution of System Components
- Proxies & Middleware
- Service Oriented Computing (SOC)
- Peer-to-Peer Systems
- **Service Provision Lifecycle** √
- Service Discovery
- Service Invocation
- Service Composition
- MTOS, BIOS & VM

Service Provision Life-cycle for Smart Devices

- Service creation: service processes register themselves in service directories
- Service operation: services are invoked and multiple interlinked services may need to be coordinated
- Service maintenance phase: service processes, access configurations and service compositions can be updated
- Service dissolution: services may be put off line or terminated temporarily by the processes themselves or by requesters

Service Provision Life-cycle



- A – Access services or clients
- P – Process services or services provision

Service Provision Life-cycle

- Exercise: Consider creation, operation, maintenance, dissolution for the following types of devices & services:
- Laptop / Internet
- Set-top box audio-video receiver
- Mobile phone
- Email Service

WS SOA Support for Service Life-cycle

- Web Services (WS) support machine-to-machine interaction
- Service Interfaces are machine-processable, syntactical
- WS SOAs consist of many possible WS protocols depending on the application and service requirements.
- Core WS SOA protocols are:
  - SOAP: is a protocol for exchanging structured information in the implementation of Web Services in computer networks. It relies on XML for message format
  - WSDL: is an XML-based interface description language that is used for describing the functionality offered by a web service; provides a machine readable descriptions (e.g. parameters required)
  - UDDI : (XML)-based registry by which businesses worldwide can list themselves on the Internet, and a mechanism to register and locate web service applications

Overview

- Smart Device and Service Characteristics
- Distributed System Viewpoints
- System Abstraction
- Partitioning and Distribution of System Components
- Proxies & Middleware
- Service Oriented Computing (SOC)
- Peer-to-Peer Systems
- Service Provision Lifecycle
- **Service Discovery √**
- Service Invocation
- Service Composition
- MTOS, BIOS & VM

Service Announcement, Discovery, Selection and Configuration

- Service discovery scope & functions depends on design.
- What's involved in service discovery?
  - It could just involve asking for the list of available service providers that match a request.
  - It may or may not include service selection, service configuration, service name to address resolution and even service invocation.

- Which happens first in service discovery?
- Network discovery

Network Discovery

What Is it? Why do we need it?

- Precedes service registration and service discovery
- Dynamic network discovery, used by mobile nodes and when new nodes are introduced into a network.

Which Network Protocols support Network Discovery?

- Domain Name Service, DNS, maps IP addresses ↔ Names
- Some nodes offer long term services
  - static assigned IP addresses may be assigned
  - e.g., printers, etc
- Common approach to dynamically discover network: DHCP
  - Ask a DHCP server for an IP address
  - addresses leased for a given time.
  - Why is leasing useful?
  - Complexity in using DHCP is in setting up & managing DHCP servers. Why?
  -

Network Discovery

- Why is leasing useful?
  - enables a limited set of resources, in this case, network addresses, to be periodically renewed by active nodes and;
  - to be reused and freed from inactive computer nodes
- Complexity in using DHCP is in setting up & managing DHCP servers. Why?
  - because multiple DHCP servers may issues overlapping addresses;
  - permanent IP addresses can conflict with dynamically assigned ones;
  - inactive clients may attempt to use an address that has been reassigned

Network Discovery: Zeroconf

- Zero Configuration Networking (Zeroconf)
  - techniques that automatically creates a usable IP network without configuration or special servers
- Allows inexpert users to connect computers, networked printers etc together & expect them to work automatically.
- Without Zeroconf or something similar, need to?
  –
- Zeroconf currently solves automating three tasks
  - choosing network addresses,
  - giving oneself an address,
  - discovering names and discovering service addresses

Network Discovery: dynamically assigning IP addresses

- Both IPv4 and IPv6 have standard ways of automatically choosing / assigning IP addresses.
- IPv4 uses the 169.254.any, link-local set of addresses, see RFC 3927.
- IPv6, zeroconf, see RFC 2462 can be used.
- 2 similar ways of figuring out which network node has a certain name.
  - Apple's *Multicast DNS (mDNS)*
  - Microsoft's *Link-local Multicast Name Resolution (LLMNR)*

Dynamic Service Discovery

- Dynamic versus Static service discovery
  - If service providers and requesters are static, then there is little need for dynamic service discovery
  - Dynamic service discovery is needed to
    - allow service requesters to change providers when requesters or providers are mobile,
    - when network access is intermittent
    - when requesters or providers fail
- Allow requesters to change providers , Why? Vice-versa?
- What is involved in Dynamic Service discovery?
  - involves decoupling service provision from service requests and supporting dynamic announcements
  - dynamic discovery of service providers and service requesters

Dynamic Service Discovery: Pull versus Push

- 2 main approaches : push or pull.

Pull: How does it work?

Discovery Services: Push

Push: How does it work?

- Push uses broadcasts or multicasts to announce the available service requests or service capabilities to a number of unknown parties, e.g., Bluetooth

- Broadcasting service requests or service descriptions are a sub type of message broadcasts to unknown message receivers

Discovery services Push: Design

Broadcast / Announcements can be designed to occur:

- Periodically irrespective of whether any audience exists or not;
- Only when any kind of audience is available;
- Only when a specific type of audience is detected
  - multicast versus broadcast.

Discovery Services: Push

Pull: How does it work?

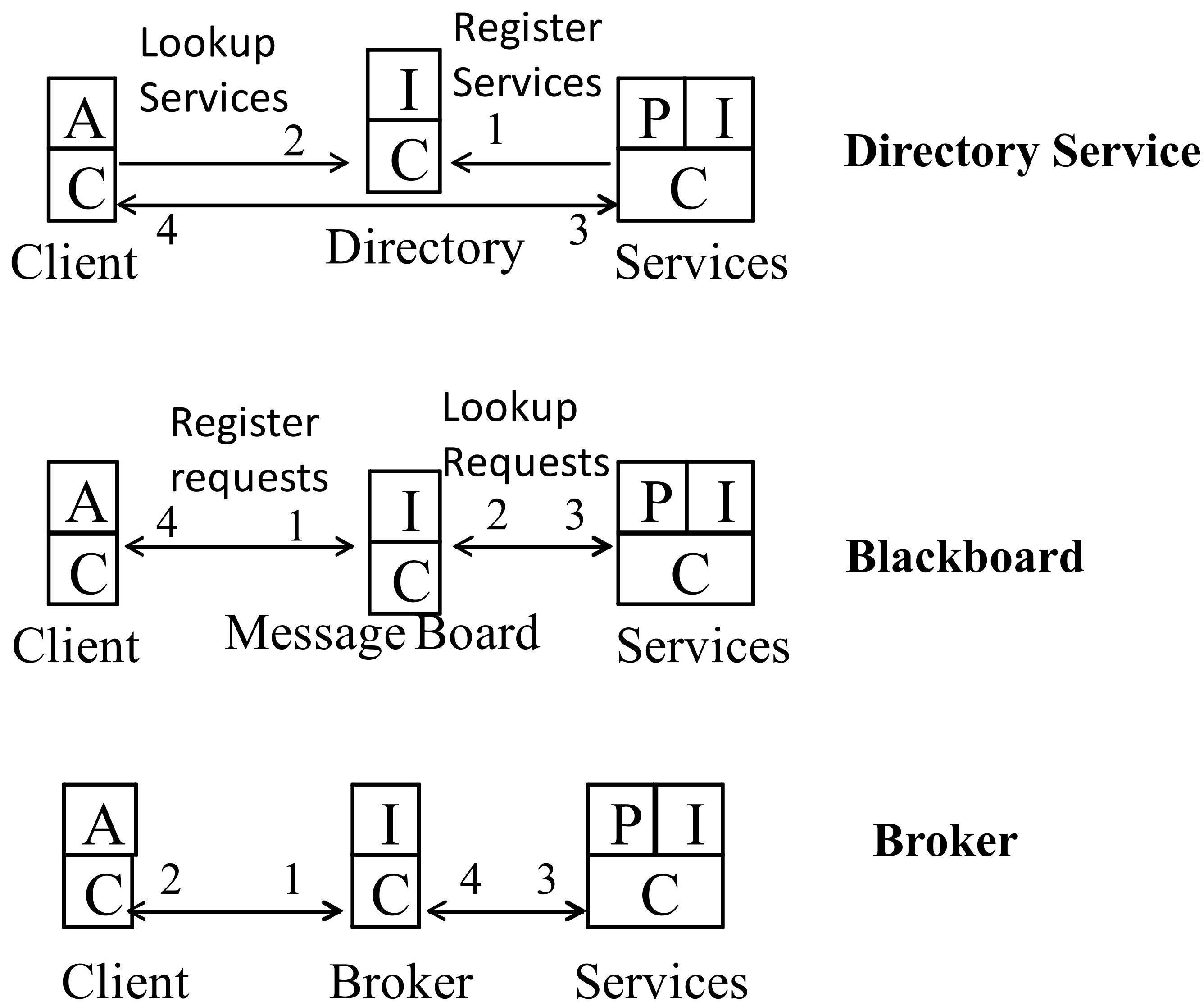
- Pull uses lookups to search or browse lists of requests or capabilities previously announced to a directory held by some known third party, e.g., Jini, UPnP, UDDI, etc.
- The third party does the matching

Discovery Services: Pull vs. Push

- Advantage of Pull?
  - Pull minimises network traffic concerning service discovery (i.e. no broadcast messages)

- Disadvantage of Pull?
  - this requires third party administration of the directory,
  - the directory to be available and
  - The directory to have a well known location for clients and servers to find it

Service Discovery Interaction Patterns



External versus Internal service selection

- External → services to satisfy request exist in virtual environment to the ICT system, rather than internally within the system itself.
- How does a requester of a service know if it needs someone else to perform the service?
  - The process of establishing whether or not a service exists internally involves self descriptions, self awareness and reflection
- How does a service requester choose between external service versus internal service invocation when both are available?
  - Exact matches or inexact wild card or even conditional matches

External versus Internal service selection

- Process of establishing whether or not a service exists internally can involve
  - Self-descriptions
  - Self-awareness
  - Reflection
- See chapter 10

Semantic Web (SW) and Semantic Resource Discovery

- Why is Syntactic level matching and discovery challenging in pervasive environments?
  - Due to the autonomy of service providers and the resulting heterogeneity of their implementations and interfaces devices
- What are benefits of Semantic matching rather than syntactic service matching?
  - leads to a service discovery performance that can give better response time and reduces network load compared to syntactic service discovery

Semantic Web (SW) and Semantic Resource Discovery

- SW represents resources using RDFS (Resource Description Framework Schema) and OWL
  - Used for semantic service descriptions
- SW defines much richer XML based data structures and relationships
  - heavier computation resources are needed to process these
- Design choices:
  - Semantic matching of service requests can enable services to be classified and grouped

Overview

- Smart Device and Service Characteristics
- Distributed System Viewpoints
- System Abstraction
- Partitioning and Distribution of System Components
- Proxies & Middleware
- Service Oriented Computing (SOC)
- Peer-to-Peer Systems
- Service Provision Lifecycle
- Service Discovery
- **Service Invocation √**
- Service Composition
- MTOS, BIOS & VM

Distributed Service Invocation

- Specifying an application protocol in terms of a set of service descriptions of service actions is often insufficient to invoke a service. Why?
  - Requesters need to have the know how to invoke the service (e.g. invoking hardware resource services such as printers may involve downloading hardware drivers into the access device)
  - Requesters may not know in which order to invoke service actions or how to handle out of order message sequences in a process without terminating service processes.
  - The interaction in the process needs to be coordinated or orchestrated?

Distributed Service Invocation

- Design of remote interaction across different computer nodes differs from design of local process interaction within the same computer node
- Why?
- Because it occurs across the network rather than across local shared memory and because different computer nodes are autonomous and heterogeneous

Distributed Service Invocation

- Multiple heterogeneous processes often need to be interleaved: e.g., when u-commerce (ubiquitous commerce) system purchasing an item
  - select item,
  - order item,
  - receive acknowledgement and receive item,
  - need to be interleaved with a separate pay for item process.
- How to carry this out in a naturally, explicitly supported way?
- Need to synchronize multiple processes
- What are the available techniques?

Distributed Service Invocation: Ordered Service Actions

- Service requesters may not know:
  - in what order to invoke service actions
  - how to handle out of order message sequences in a process without terminating service processes.
- Interaction in the process coordinated needs to be conducted.
- Often coordination may be hard-coded into each service API and under the control of the provider.
  - Makes the coordination of multiple services inflexible.

Distributed Service Invocation: Ordered Service Actions

- Clients often need to invoke not just individual service actions in isolation but to invoke a whole series of service interactions as part of a business process
- Multiple heterogeneous processes often need to be interleaved:
- Network Transmission may or may not maintain order of action messages when these are sent
- Complex to design in order to make remote communication (remote procedure calls (RPC) or remote method invocation (RMI) look like local calls, e.g., need parameter marshalling

Distributed Service Invocation: Fully Ordered Service Actions

- Two types of design based upon service action ordering
  - Full versus Partial

1) Fully ordered system processes of actions

- Specify actions executed as fixed sequences of actions.
- Earlier actions in sequence output data used by later ones
- Control of flow may contain some flexibility in terms of branches, conditions and loops.
- Example uses method invocation such as RPC and RMI
- Suitable for synchronous executions
- Less suitable for loosely coupled infrastrcture

Distributed Service Invocation: Partially Ordered Actions

- Two types of design based upon action ordering

2) Partially or non ordered system processes of actions Non-ordered System

- Specify action triggers (events) and action (responses, handling)
- Don't fully order, but may partially order
- Example Uses
  - LooCI component model
- Suitable for open dynamic environments

Distributed Service Invocation: fully versus partially ordering

- Advantages and disadvantages of full action ordering?
- Advantages and disadvantages of partial action ordering?

Service Invocation: Separating Coordination & Computation

- Should coordination mechanisms be separated from computation mechanisms.

This supports several key benefits:

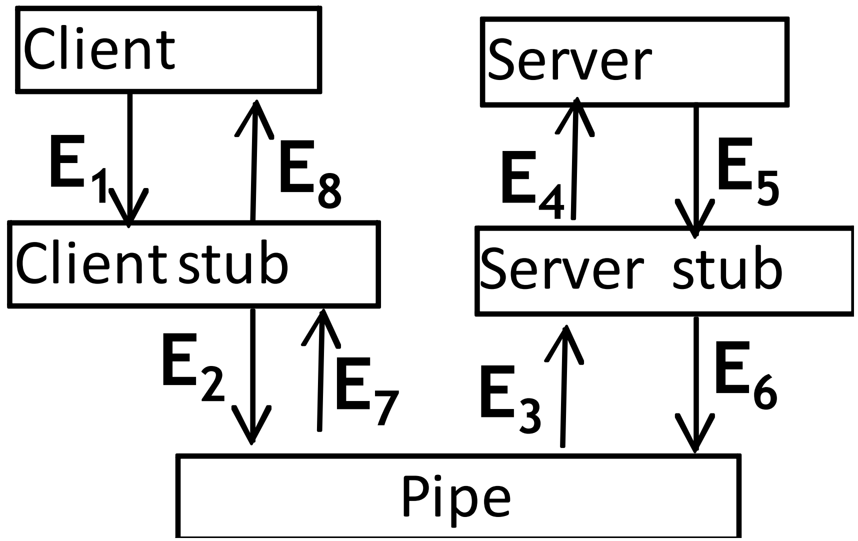
- Portability
- Heterogeneity
- Flexibility

Distributed Service Invocation Coordination Models

Designs for distributed interaction include:

- (*Remote) Procedure Calls / object-oriented Remote Method interaction*:
- *Layered model*:
- *Pipe-filter model*
- *Event-driven Action or EDA Model*:
- *Shared data repositories*:

Distributed Service Invocation Data Model: RPC model

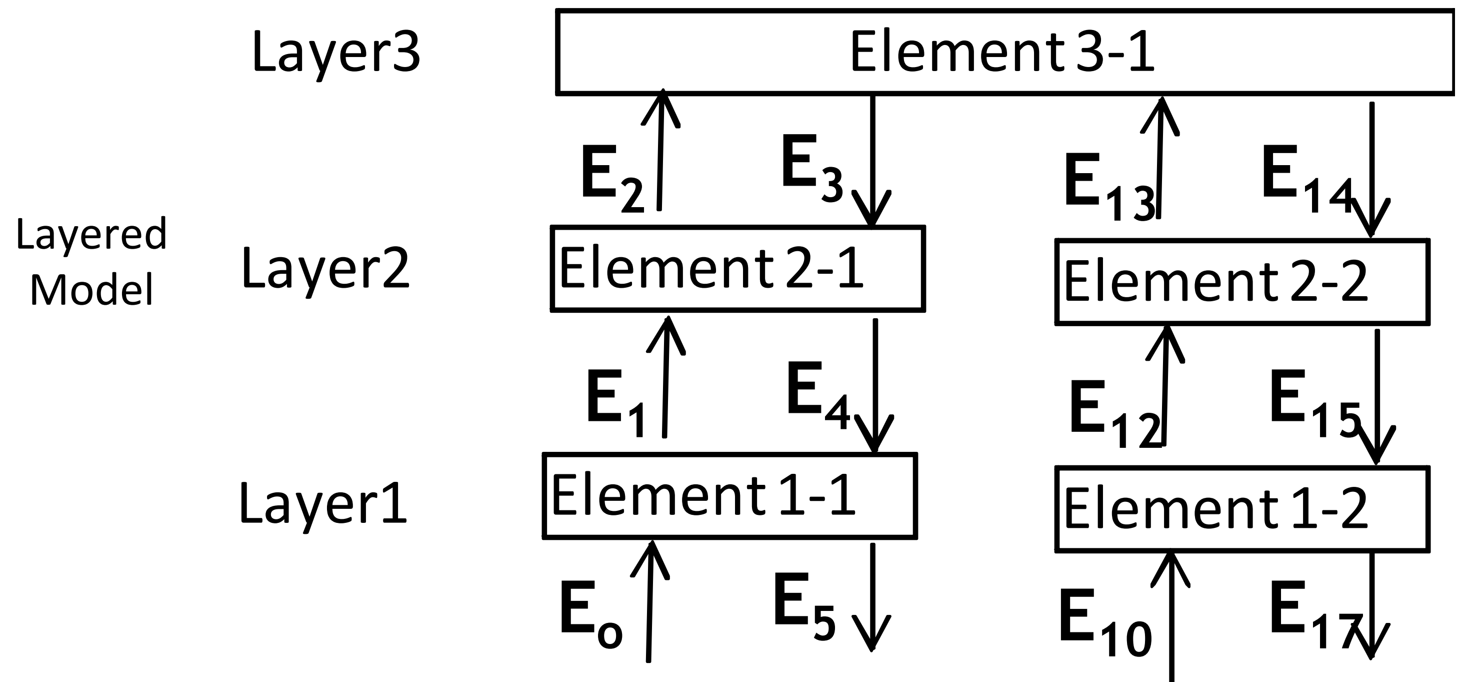


Uses?

Distributed Service Invocation Data Model: RPC model

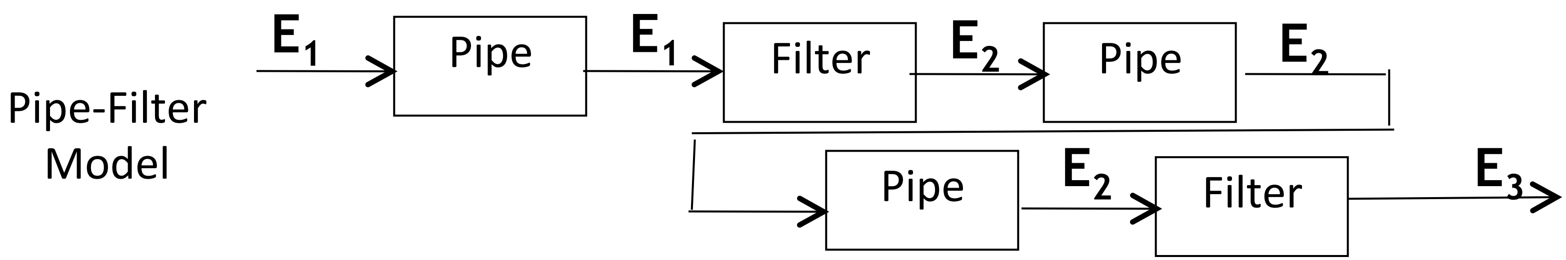
- For each type of service invocation data model we can give more detail about how the interaction model works
- (Remote) Procedure Call Model – makes remote calls look like local calls
- Remote Method Invocation – object oriented remote calls in Java

Distributed Service Invocation Data Model: Layered Model



Employed to to hide the details of lower level interaction which is often used as a design to mask and combine the use of multiple network protocols

Distributed Service Invocation Data Model: Pipe-Filter Model



often used for streaming and combining multiple media to different applications that use different kinds of content filtering

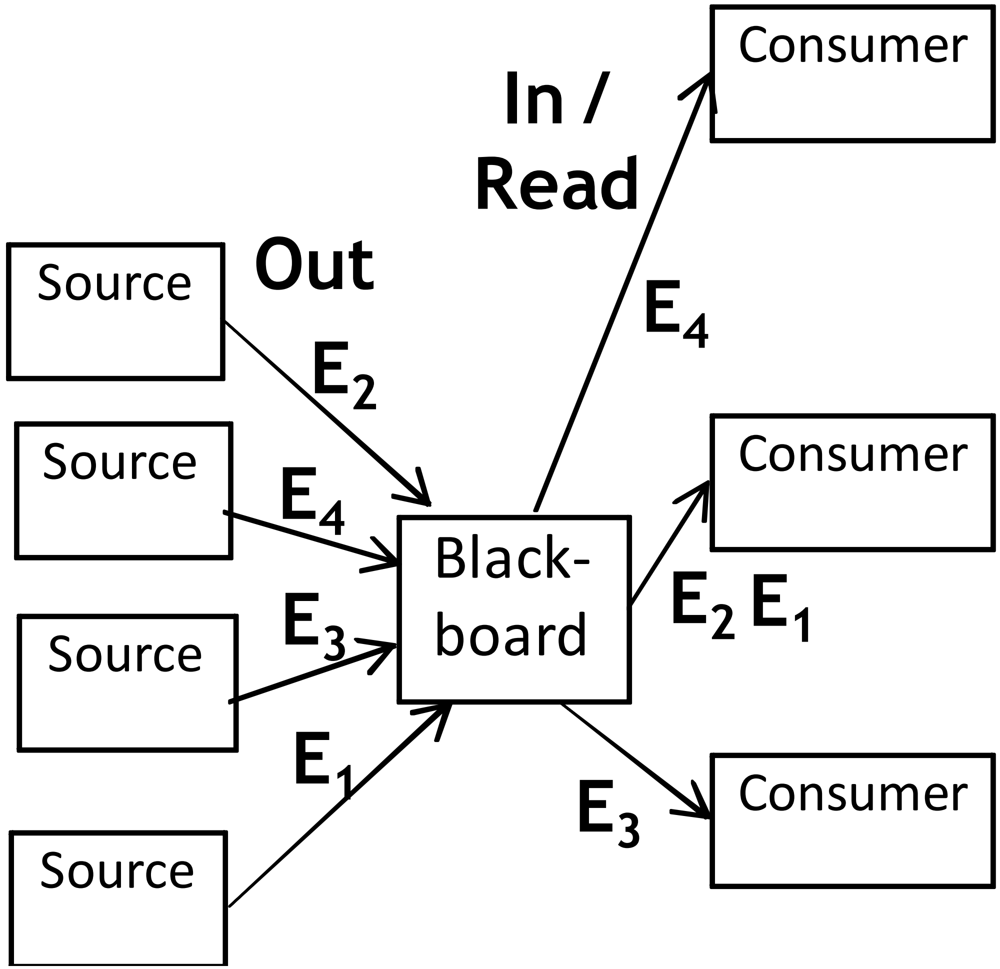
Distributed Service Invocation Data Model: Shared Data Repository

- Shared repository system consists of two types of components:
  - central data structure represents the current state (blackboard)
  - collection of independent components operate on central data store. (interacting components – consumers and providers)
- 2 major sub-types of coordination depending on:
  - if transactions in an input stream trigger the selection of executing processes, e.g., a database repository
  - if the current state of the central data structure is the main trigger of selecting processes to execute, e.g., a blackboard repository.

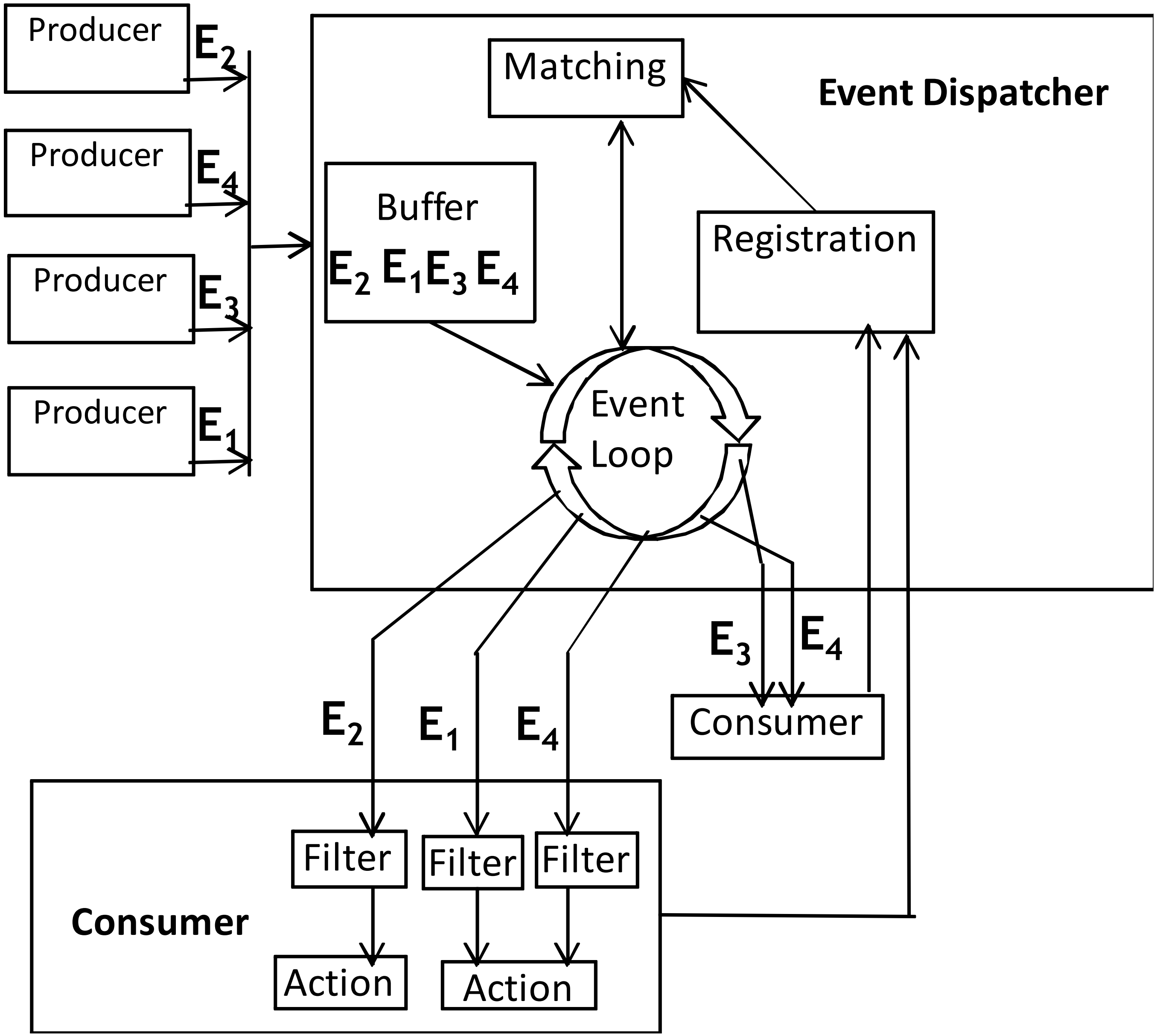
Shared Data Repository: Blackboard

- Represents & stores data created & used by other components.
- Data is input a repository from data producers.
- Data is output from a repository to data consumers.

Shared Data Repository: Blackboard



Service Invocation Data Model: EDA



Distributed Service Invocation Data Model: EDA

- Event-Driven Architectures or EDA
- EDA model is an important design for SOC and MOM (Message Oriented Middleware) Architectures
- Event is some input such as a message or procedure call of interest (e.g. if time equals 1pm)
- EDA is also known as *Publish-and-Subscribe interaction*.
- Some nodes publish events while others subscribe to being notified when specified events occur

Distributed Service Invocation Data Model: EDA

- A Few events may be significant because it may cause a significant change in state
  - e.g., a flat tyre triggers a vehicle driver to slow down
- Event may cause some predefined threshold to be crossed
  - e.g., after travelling a certain number of miles, a vehicle must be serviced to maintain it in a roadworthy state
- Event may be time-based
  - e.g., at a certain time record a certain audio video program
- External events can trigger services. Services may in turn trigger additional internal events,
  - e.g., the wheel brake pads are too worn and need to be replaced.
- Many events may not be significant

Distributed Service Invocation Data Model: EDA Challenges

- Design challenges complexity of EDA?
  - Event floods: solved by prioritising and using event expiration?
  - EDA generally have no persistence
  - Can be difficult to keep things running through a failure
- Solutions
  - Prioritising events
  - Event persistence
  - Event coordination (event coordination may be needed by applications when events can arrive in any order)
  - Highly selective event generation and transmission

Overview

- Service Provision Lifecycle
- Service Discovery
- **Service Invocation √**
    - Coordination Models
    - **On-demand Service Invocation √**
    - – Volatile Service Invocation
    - – ESB versus MOM
- Service Composition

Blackboard versus Event Driven

Key difference?

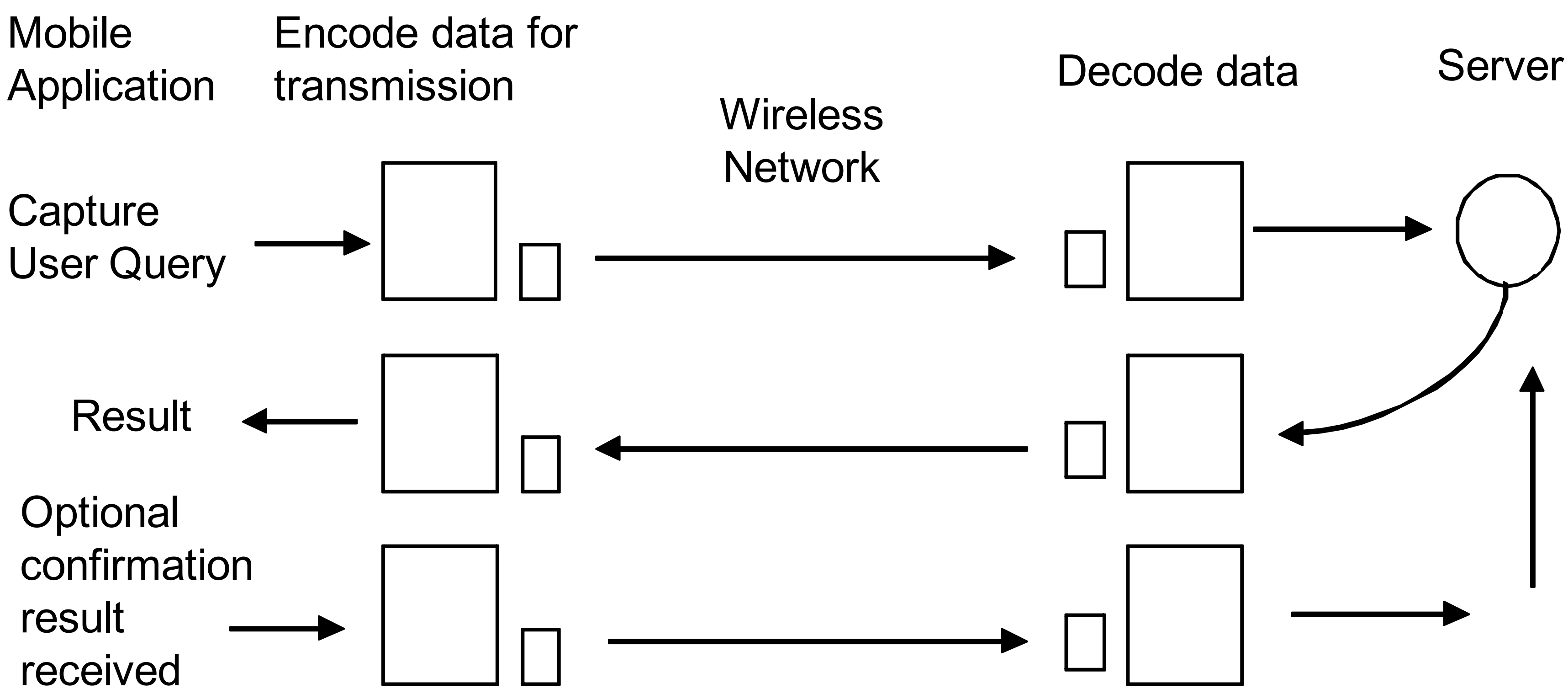
The main difference between the EDA and shared data repository is the persistent storage of the input data and data management, e.g., consistency management

On-Demand Distributed Service Invocation

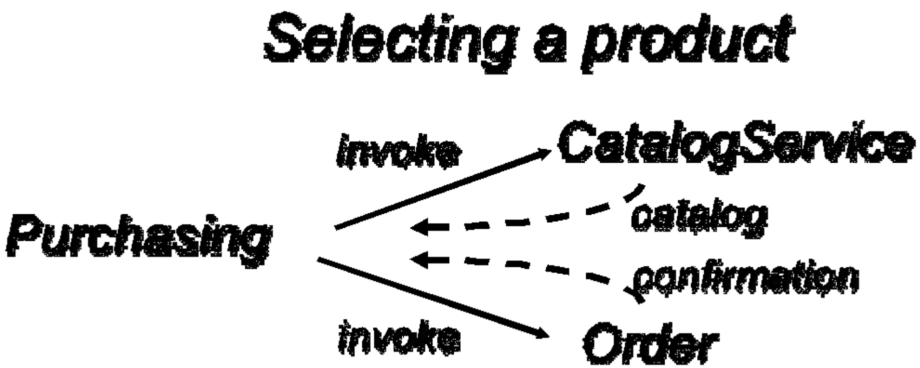
- On-demand: remote service access whenever needed
- Some data created locally & stored or processed remotely
  - E.g. e-commerce
- Some data is stored remotely & accessed locally
  - E.g., catalogue-based purchase
- Remote service invocation may involve single read or write data operations
- Remote service invocation may involve multiple read or write data operations,
  - E.g., clients issues several requests to discover suitable sevices
- Multiple service actions may be integrated into a whole
  - E.g., purchasing travel (tickets, hotels, car rent, etc.)

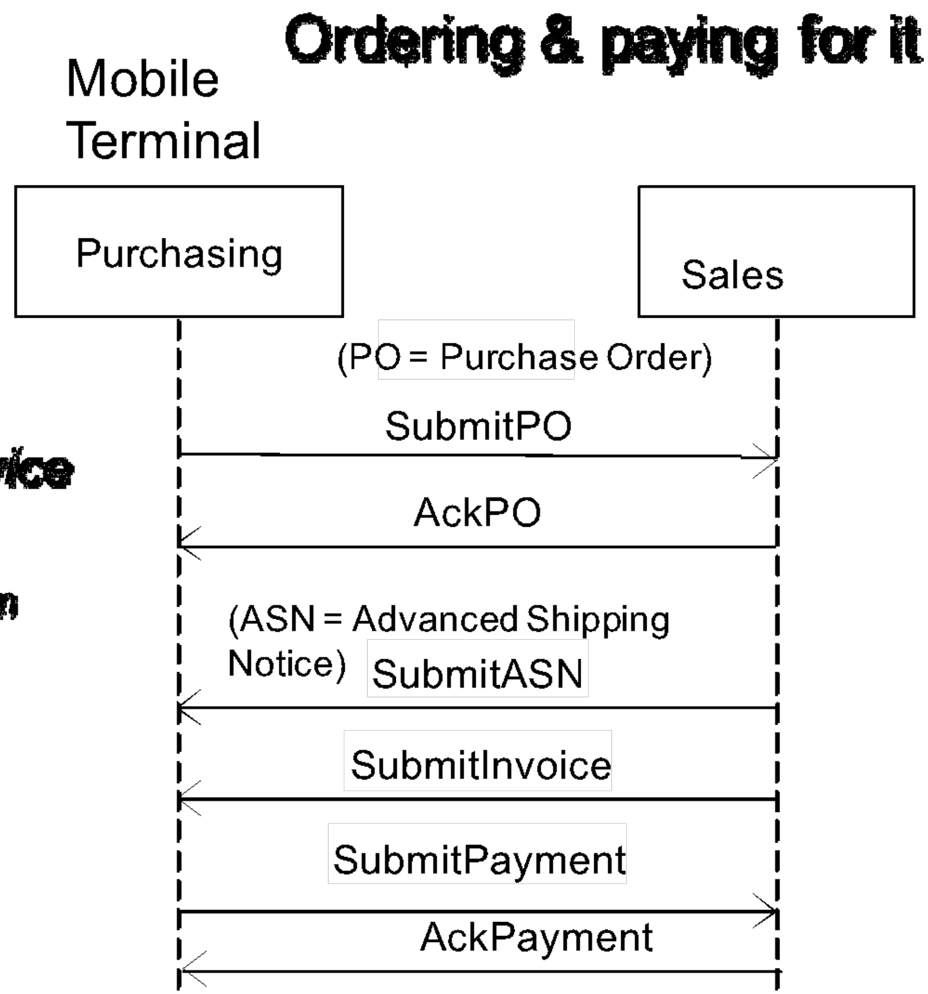
On-Demand Distributed Service Invocation

Service invocation e.g., what is the best flight given these constraints?



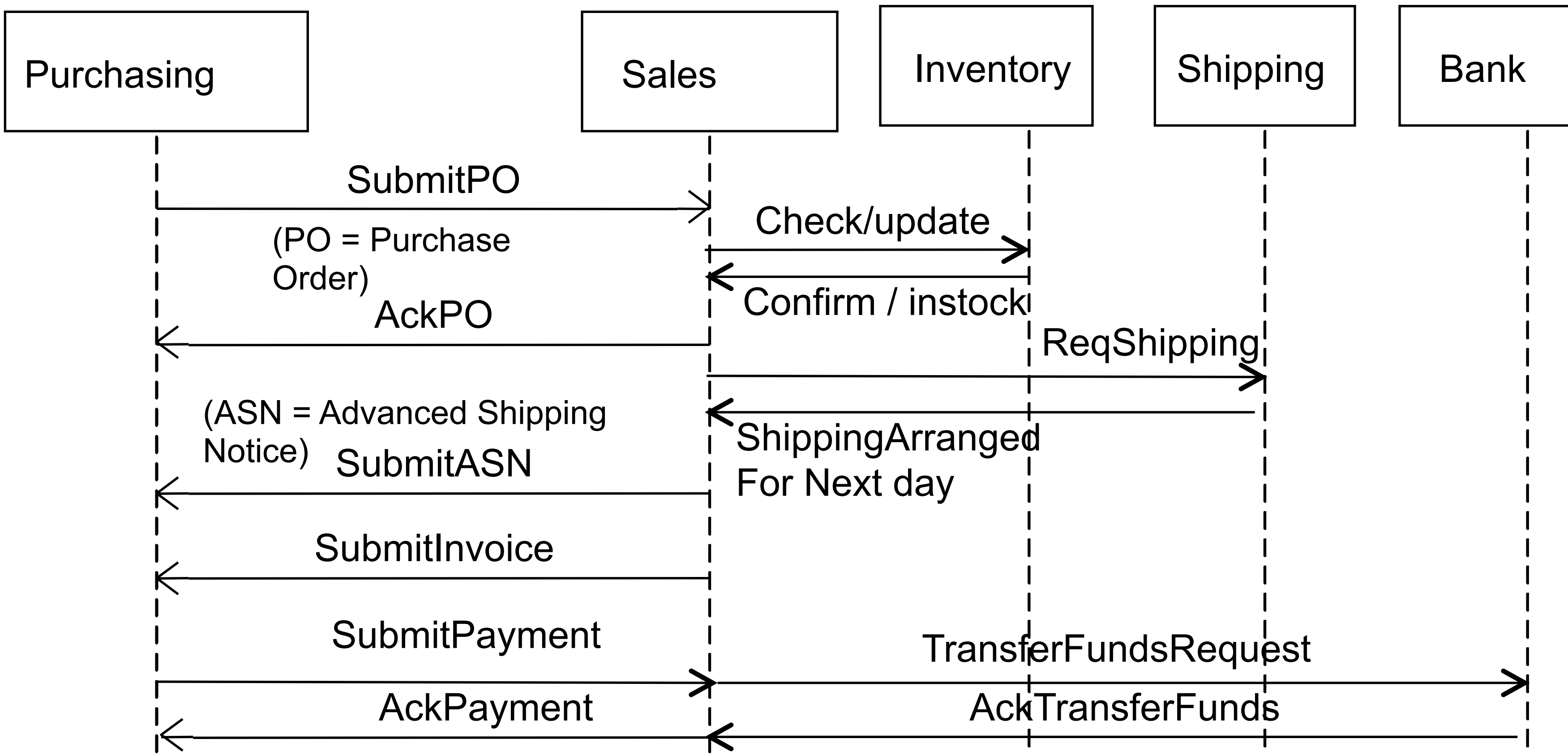
On-Demand Distributed Service Invocation can be Complex





On-Demand Distributed Service Invocation can be Complex

Ordering & paying for it



On-demand Service access

- On-demand service interaction suited to thin client interaction. Why?
  - It ensures server based processing in the network
  - Look up in the catalogue
- Suitable for small foot-print / low resource / devices or terminals
- Back-end server required continually throughout the transaction
- Server-side processing used to
  - Process transaction
  - Dynamic Server-side device profiling of clients

On-Demand Services: Request-reply, Pull Design

- Request/response is “pull-based” approach
- Client, requires instantaneous updates of information
- Need highly available network connection
- Clients continuously poll service providers
- In many mobile applications energy is a scarce resource
- Pure pull-based solution may not support the high dynamicity of information resources, that change and move
- Pull works if directory service is up to date
- Publish/subscribe or EDA can be used so that timely notification of events are sent to interested subscribers.

Overview

- Service Provision Lifecycle
- Service Discovery
- **Service Invocation √**
  - Coordination Models
  - On demand Service invocation
  - **Volatile Service Invocation √**
  - ESB versus MOM
- Service Composition

Volatile Service Invocation: Networks

Sometimes service access may be quite intermittent because of intermitted network, Causes?

- Limited network area converage
- Intermittent low bandwidth access via some networks
- Hand offs
- Interference
- Variable signal reception

Volatile Service Invocation: services

- Intermittent service access causes?

- Designs of the application and middleware must take volatile into account Why?
  - Because requests will block or terminate and may need to be repeated and restarted
- Basic designs to handle volatile service access?
  - use of asynchronous communication
  - handling unreliable communication (ACK/NACK?)and
  - message caching

Volatile Service Invocation: Designs

Designs to support volatile service invocation? By Satyanarayanan (1996)

- They provide concurrent access to shared data occurs. This concurrency may occur at two different
- levels: remote versus local and write versus read

Volatile Service invocation: Over Unreliable Networks

- Networks may offer a QoS to deliver messages without loss or delay and in order
- Service access over wireless networks often more unreliable than wired networks.
- Applications can assume no network guarantee about delivery - need to detect & handle message corruption & message loss. How?
  - Check message integrity
  - ACK/NACK
  - Caches

Volatile Service invocation Design: Stateful Senders & Receivers

- Senders and receivers can be aware of states (stateful)
  - They buffer sent messages or retain some intermediate states about the messages
- Stateful senders don’t need to create message replacements from scratch
- Stateful communication may be more complex to synchronise than stateless communication because the equivalence of intermediate states may need to be compared.

Volatile Service Invocation: Repeating Service Requests

- Before repeating message transmission is to consider the consequences of doing this.
- Messages that can be repeated, at least once, without side-effects are called *idempotent messages*,
  - e.g. pressing an elevator call button again because the response has not yet been completed
- Other messages may be non-idempotent,
  - e.g. a message request that withdraws funds from one bank account

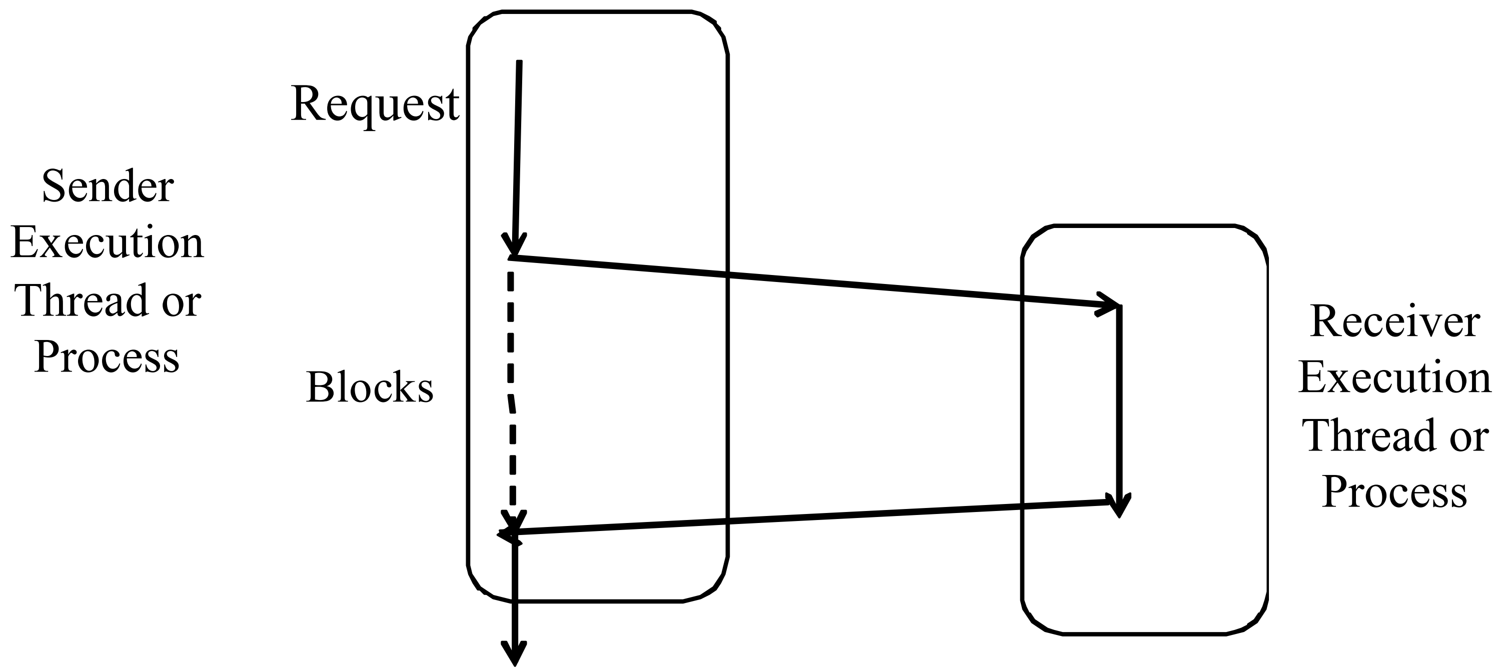
Volatile Service Invocation: Repeating Service Requests

- Partial observability at sender / requester ↑ complexity. Why?
  - the sender may not be able to distinguish between a sender crash before the message is sent,
  - a sent message being lost,
  - a remote server crash and the received message being lost

Volatile Service Invocation: Asynchronous (MOM) I/O

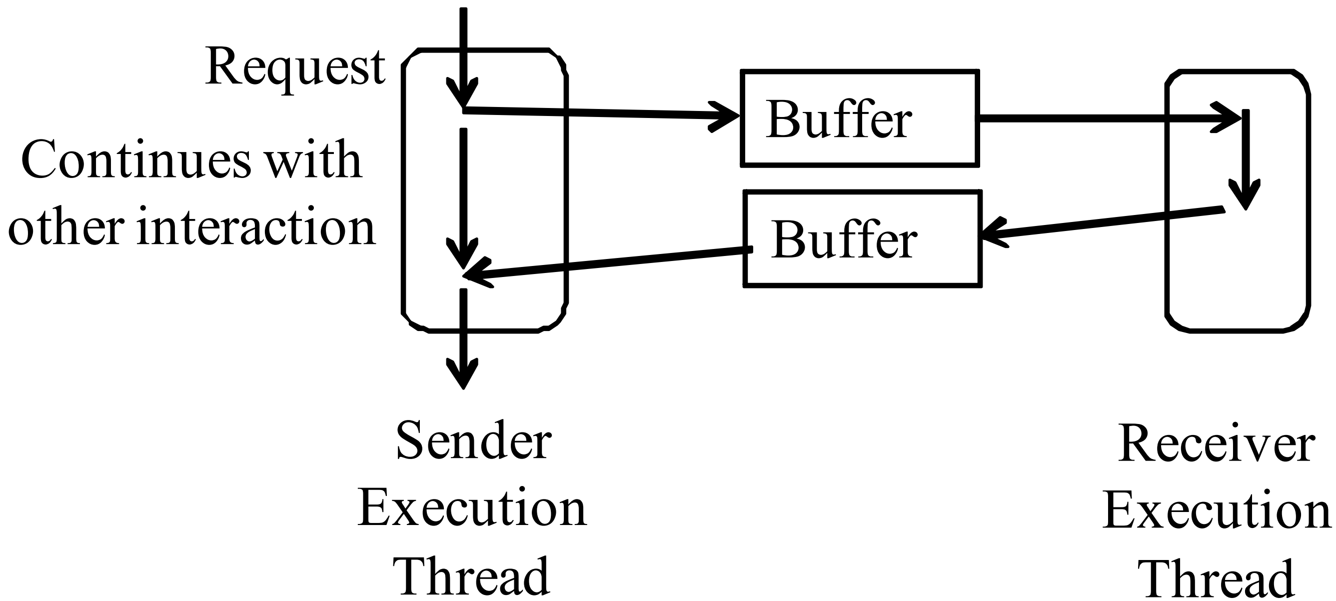
- Problems when senders issues requests to receivers
  - receivers (e.g. servers) need to be ready before clients start to make requests to them
- Asynchronous messaging can solve this issue. Asynchronous messaging applications such as email over the Internet, SMS over mobile voice networks are often regarded as the first important data applications over these networks respectively.
- Two basic variants of asynchronous messaging exist:
  - Sender side asynchronous requests
  - Receiver side asynchronous requests

Volatile Service Invocation: Synchronous I/O



- Advantages?
  - Normally, no buffer required
- Disadvantages
  - It Blocks processes

Asynchronous I/O: design based upon buffering

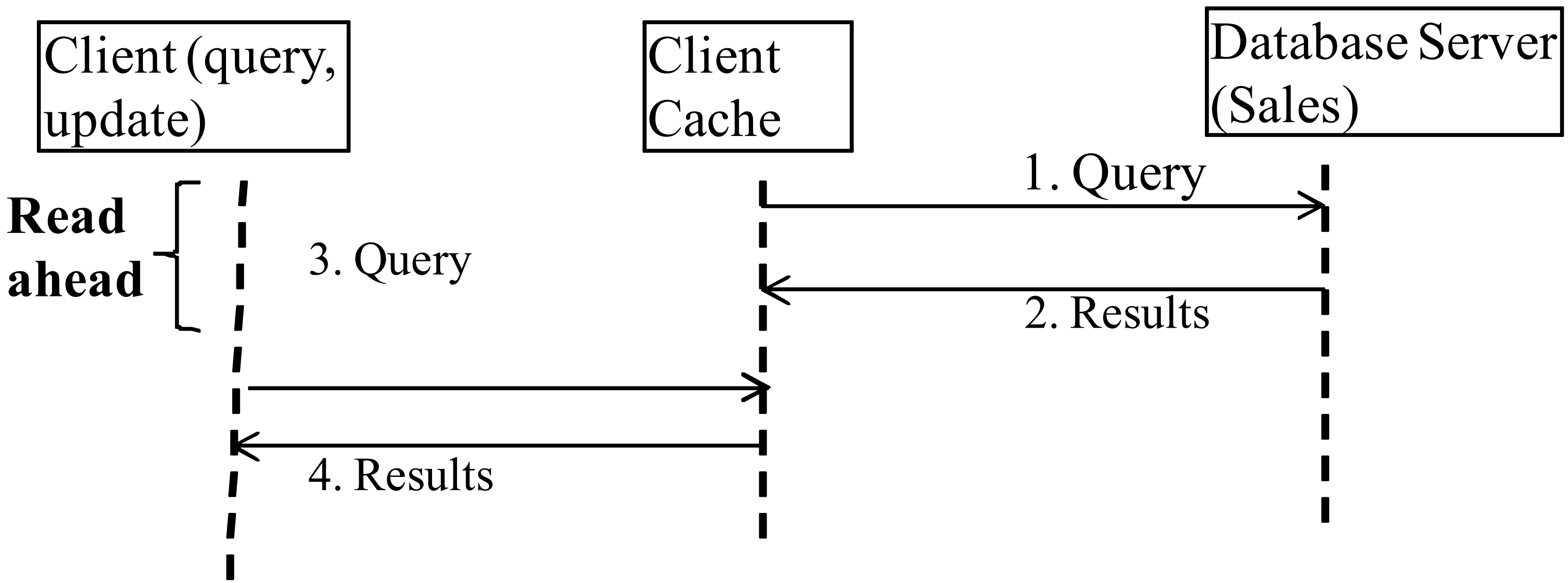


Volatile Service invocation: read ahead caches

- Read ahead is one design option to deal with volatile service invocation
- Information is pre-cached in devices when the network is available.
- Cache-hit
- Cache-miss
- Design decisions?

- Support frequent caching?
  - .
- Support less frequent caching?

Volatile Service Invocation: Read Ahead



Volatile Service Invocation: Delayed Writes

- With delayed writes, updates are made to the local cache whilst services are unreachable which must be later reintegrated upon reconnection.
- Concurrent local and remote updates may need to be synchronised.
- Write conflicts need to be detected when the same data has been modified locally and remotely.
- Techniques to handle cache misses & cache resynchronisation?

Volatile Service Invocation: Delayed Writes