CSC12108 Distributed Application

INTRODUCTION

Distributed System

Instructor – Msc. PHAM MINH TU

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Outline

- ❑**Introduction**
- ❑Challenges
- ❑System Model
- ❑Indirect Communication
- ❑Web Services
- ❑Distributed Transactions

Introduction

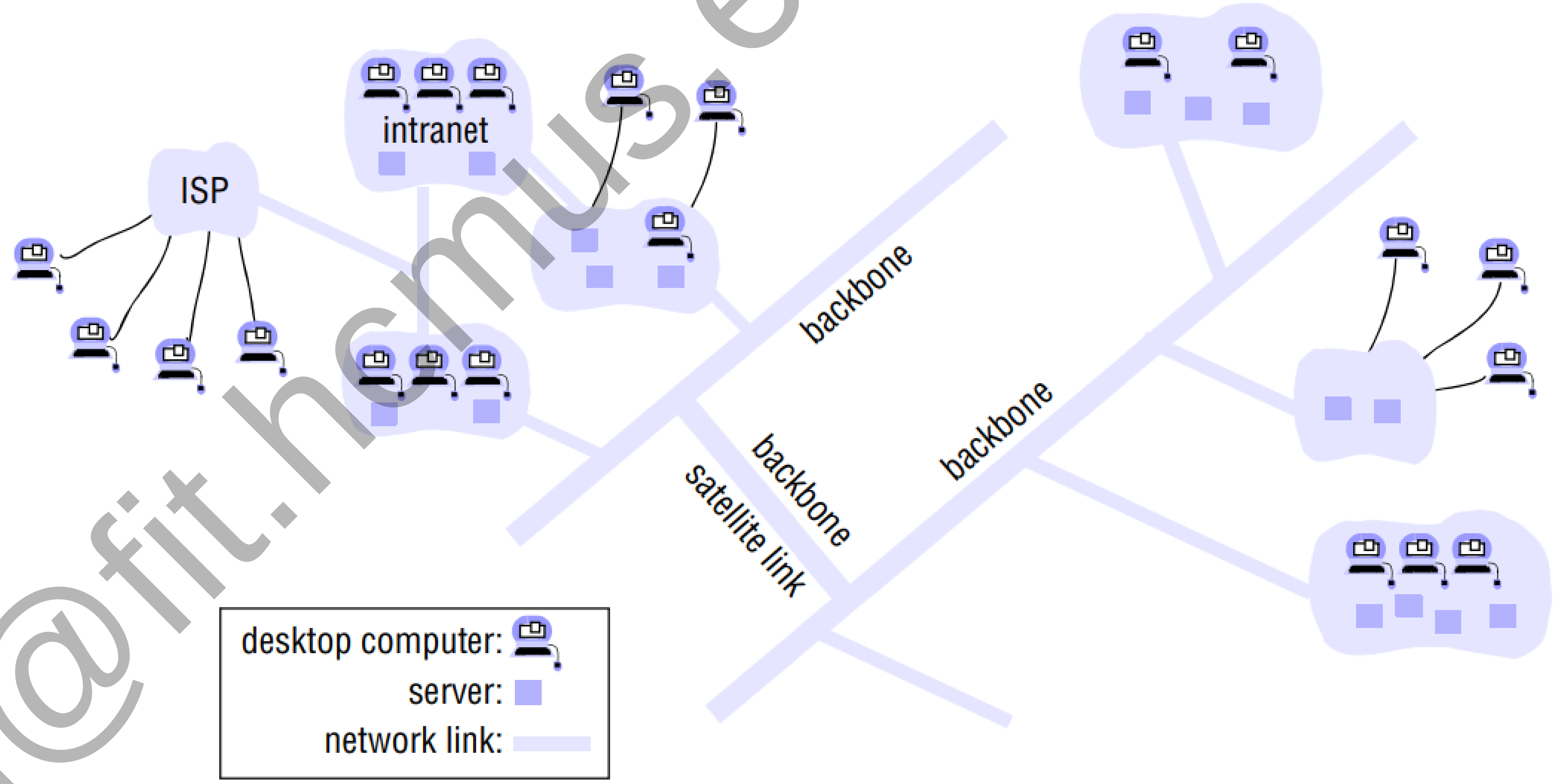
- A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. [1]
  - Concurrency of components
  - Lack of a global clock
  - Independent failures of components

Introduction

- Example
  - Let address application domains that associated networked applications.
    - Finance and commerce
    - The information society
    - Transport and logistics
    - Healthcare
    - Education
    - Creative industries and entertainment

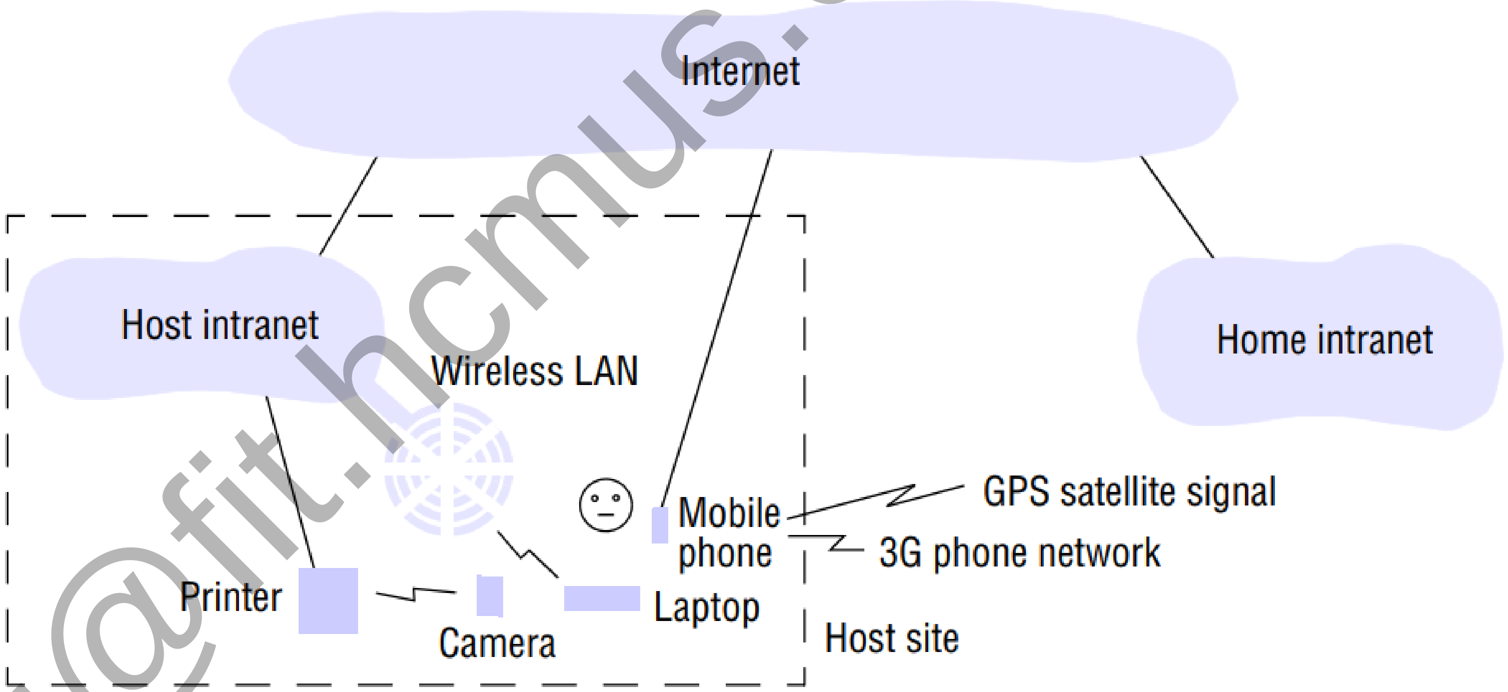
Introduction

- ❑Trends
  - ❑Pervasive networking and the modern Internet



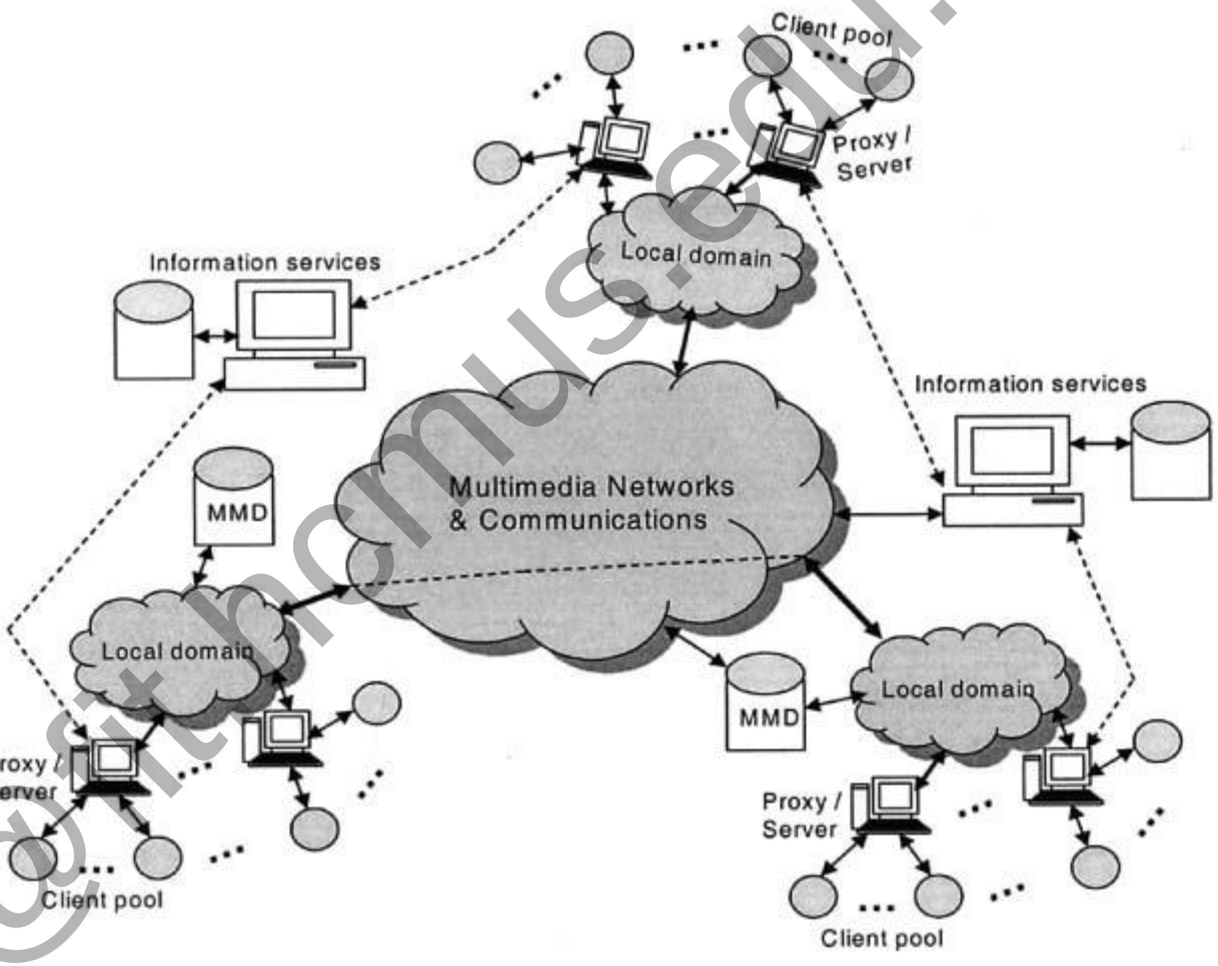
Introduction

- Trends
  - Mobile and ubiquitous computing



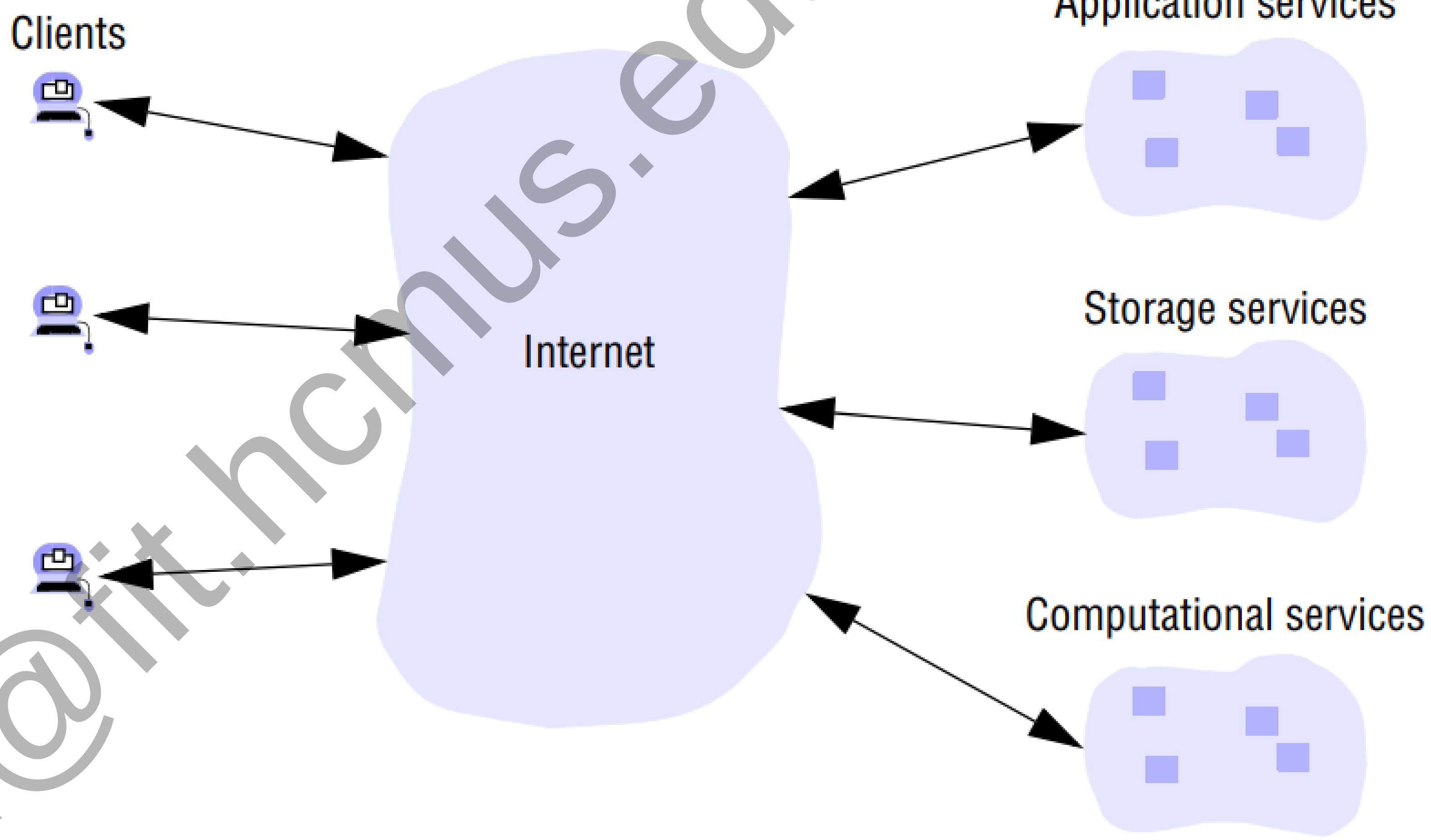
Introduction

- ❑Trends
  - ❑Distributed multimedia systems



Introduction

- Trends
  - Distributed computing as a utility



Outline

- ❑Introduction
- ❑**Challenges**
- ❑System Model
- ❑Indirect Communication
- ❑Web Services
- ❑Distributed Transactions

Challenges

What are challenges in distributed systems?

Challenges

- Heterogeneity
  - Networks
  - Computer hardware
  - Operating systems
  - Programming languages
  - Implementations by different developers

Challenges

- Openness
- Security
  - Confidentiality
  - Integrity
  - Availability

Challenges

- ❑Scalability
  - ❑Controlling the cost of physical resources
  - ❑Controlling the performance loss
  - ❑Preventing software resources running out

Challenges

- Failure handling
- Concurrency
  - The operations on the object may conflict with one another and produce inconsistent results
- Transparency

Outline

- ❑Introduction
- ❑Challenges
- ❑**System Model**
- ❑Indirect Communication
- ❑Web Services
- ❑Distributed Transactions

System Model

- Physical models
- Architectural models
- Fundamental models

System Model

❑Physical models

- ❑To describe a system that capture the hardware composition of a system in terms of the computers (and other devices, such as mobile phones) and their interconnecting networks.

System Model

- Physical models
  - Baseline physical model
    - A minimal physical model of a distributed system as an extensible set of computer nodes interconnected by a computer network for the required passing of messages
  - Early distributed systems
    - These systems typically consisted of between 10 and 100 nodes interconnected by a local area network, with limited Internet connectivity and supported a small range of services such as shared local printers and file servers as well as email and file transfer across the Internet

System Model

- ❑ Physical models
  - ❑ Internet-scale distributed systems
    - ❑ An extensible set of nodes interconnected by a network of networks
  - ❑ Contemporary distributed systems
    - ❑ Further developments in physical models
      - ❑ The emergence of mobile computing has led to physical models where nodes such as laptops or smart phones may move from location to location in a distributed system
      - ❑ The emergence of ubiquitous computing has led to a move from discrete nodes to architectures where computers are embedded in everyday objects and in the surrounding environment
      - ❑ The emergence of cloud computing and, in particular, cluster architectures has led to a move from autonomous nodes performing a given role to pools of nodes that together provide a given service.

System Model

- ❑ Architectural models
  - ❑ To describe a system in terms of the computational and communication tasks performed by its computational elements

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

System Model

- ❑Architectural models
  - ❑Architectural elements
    - ❑What are the entities that are communicating in the distributed system?
    - ❑How do they communicate, or, more specifically, what communication paradigm is used?
    - ❑What (potentially changing) roles and responsibilities do they have in the overall architecture?

System Model

- Architectural models
  - Architectural elements

Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

System Model

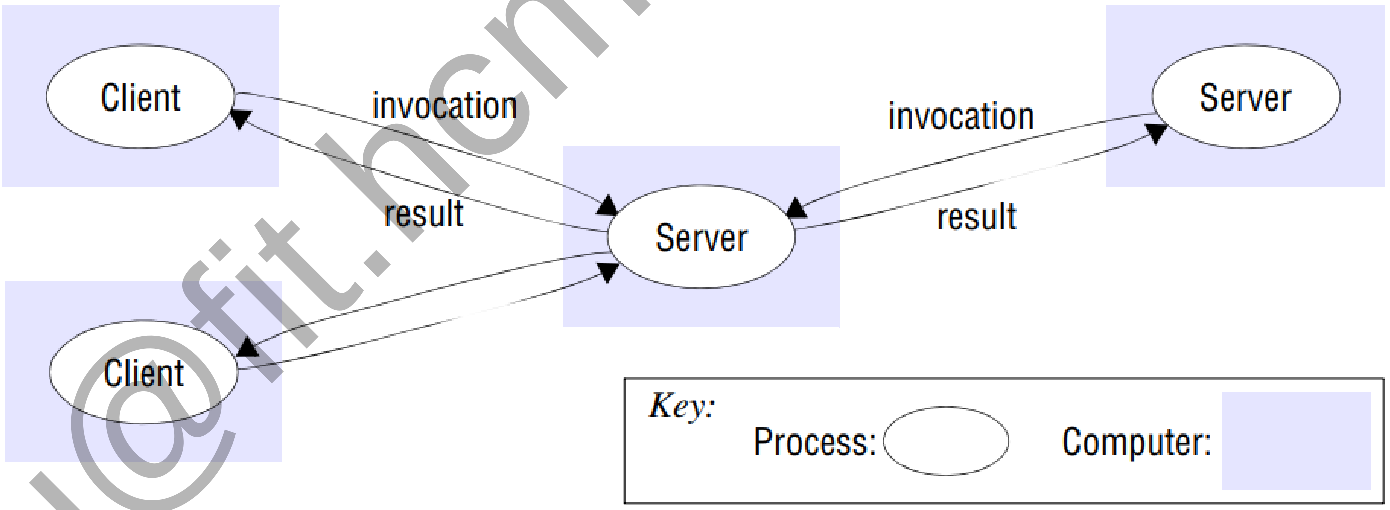
- Architectural models
  - **Interprocess communication** refers to the relatively low-level support for communication between processes in distributed systems, including message-passing primitives, direct access to the API offered by Internet protocols (socket programming) and support for multicast communication
  - **Remote invocation**
    - Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details.
    - Remote method invocation (RMI)

System Model

- Architectural models
  - **Indirect communication**
    - Group communication
    - Publish-subscribe systems
    - Message queues
    - Tuple spaces
    - Distributed shared memory

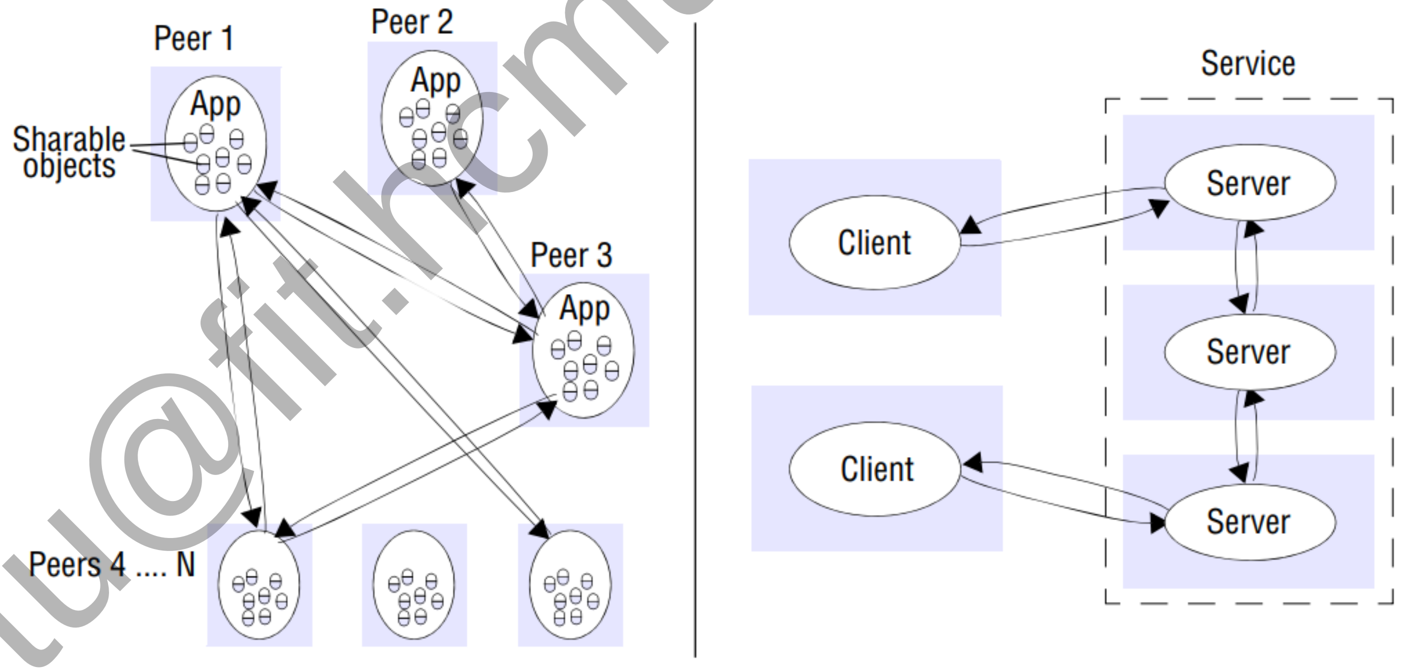
System Model

- ❑Architectural models
  - ❑**Roles and responsibilities**
    - ❑Client-server



System Model

- ❑Architectural models
  - ❑**Roles and responsibilities**
    - ❑Peer-to-peer



System Model

- Fundamental models
  - To take an abstract perspective in order to examine individual aspects of a distributed system
  - **Three important aspects** of distributed systems:
    - Interaction models
    - Failure models
    - Security models

System Model

- ❑Fundamental models
  - ❑Interaction
    - ❑Computation occurs within processes; the processes interact by passing messages, resulting in communication (information flow) and coordination (synchronization and ordering of activities) between processes
    - ❑The interaction model must reflect the facts that communication takes place with delays that are often of considerable duration, and that the accuracy with which independent processes can be coordinated is limited by these delays and by the difficulty of maintaining the same notion of time across all the computers in a distributed system.

System Model

- Fundamental models
  - Failure
    - The correct operation of a distributed system is threatened whenever a fault occurs in any of the computers on which it runs (including software faults) or in the network that connects them
    - This provides a basis for the analysis of their potential effects and for the design of systems that are able to tolerate faults of each type while continuing to run correctly

System Model

- ❑Fundamental models
  - ❑Security
    - ❑The modular nature of distributed systems and their openness exposes them to attack by both external and internal agents

Outline

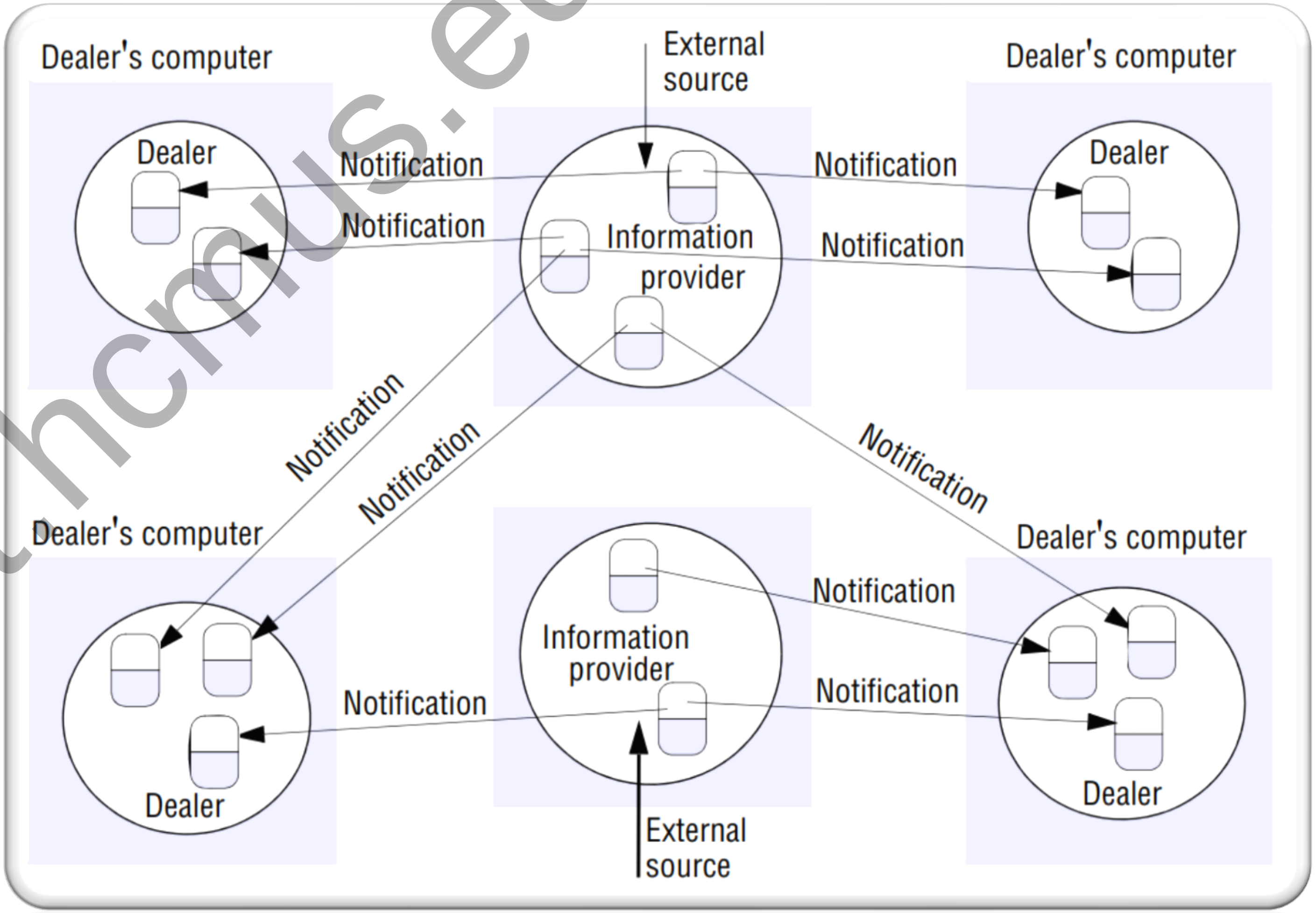
- ❑Introduction
- ❑Challenges
- ❑System Model
- ❑**Indirect Communication**
- ❑Web Services
- ❑Distributed Transactions

Indirect Communication

- Indirect communication is defined as communication between entities in a distributed system through an intermediary with no direct coupling between the sender and the receiver(s)
    - Space uncoupling in which the sender does not know or need to know the identity of the receiver(s), and vice versa. Because of this space uncoupling, the system developer has many degrees of freedom in dealing with change: participants (senders or receivers) can be replaced, updated, replicated or migrated
    - Time uncoupling, in which the sender and receiver(s) can have independent lifetimes. In other words, the sender and receiver(s) do not need to exist at the same time to communicate. This has important benefits, for example, in more volatile environments where senders and receivers may come and go

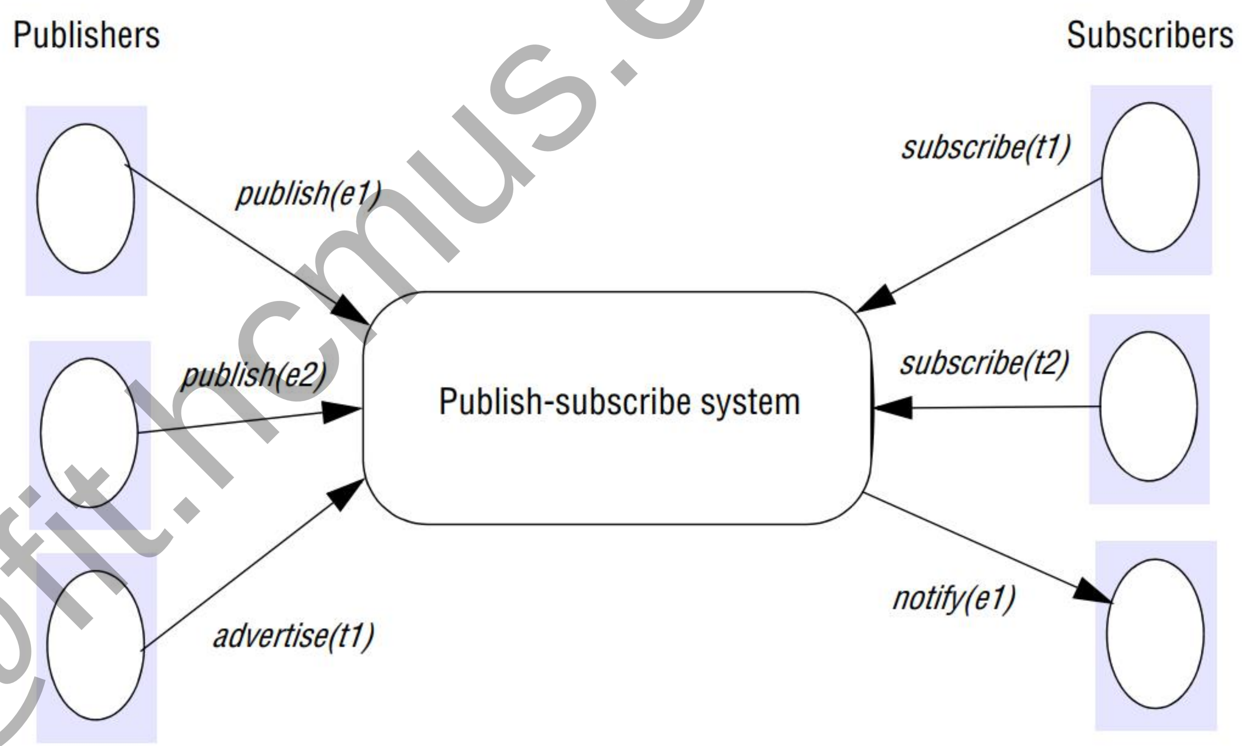
Indirect Communication

- ❑ Publish-subscribe systems - Event-based systems
  - ❑ A publish-subscribe system is a system where publishers publish structured events to an event service and subscribers express interest in particular events through subscriptions which can be arbitrary patterns over the structured events
  - ❑ Application
    - ❑ Dealing room system



Indirect Communication

- ❑Publish-subscribe systems
  - ❑Characteristics of publish-subscribe systems
    - ❑Heterogeneity
    - ❑Asynchronicity
  - ❑ The programming model



Indirect Communication

- Publish-subscribe systems
  - Implementation issues
    - The task of a publish-subscribe system is clear: to ensure that events are delivered efficiently to all subscribers that have filters defined that match the event. Added to this, there may be additional requirements in terms of security, scalability, failure handling, concurrency and quality of service
    - Centralized versus distributed implementations
      - Steps of centralized implementations
        - To centralize the implementation in a single node with a server on that node acting as an event broker
        - Publishers then publish events (and optionally send advertisements) to broker
        - Subscribers send subscriptions to the broker and receive notifications in return
        - Interaction with the broker is then through a series of point-to-point messages.

Indirect Communication

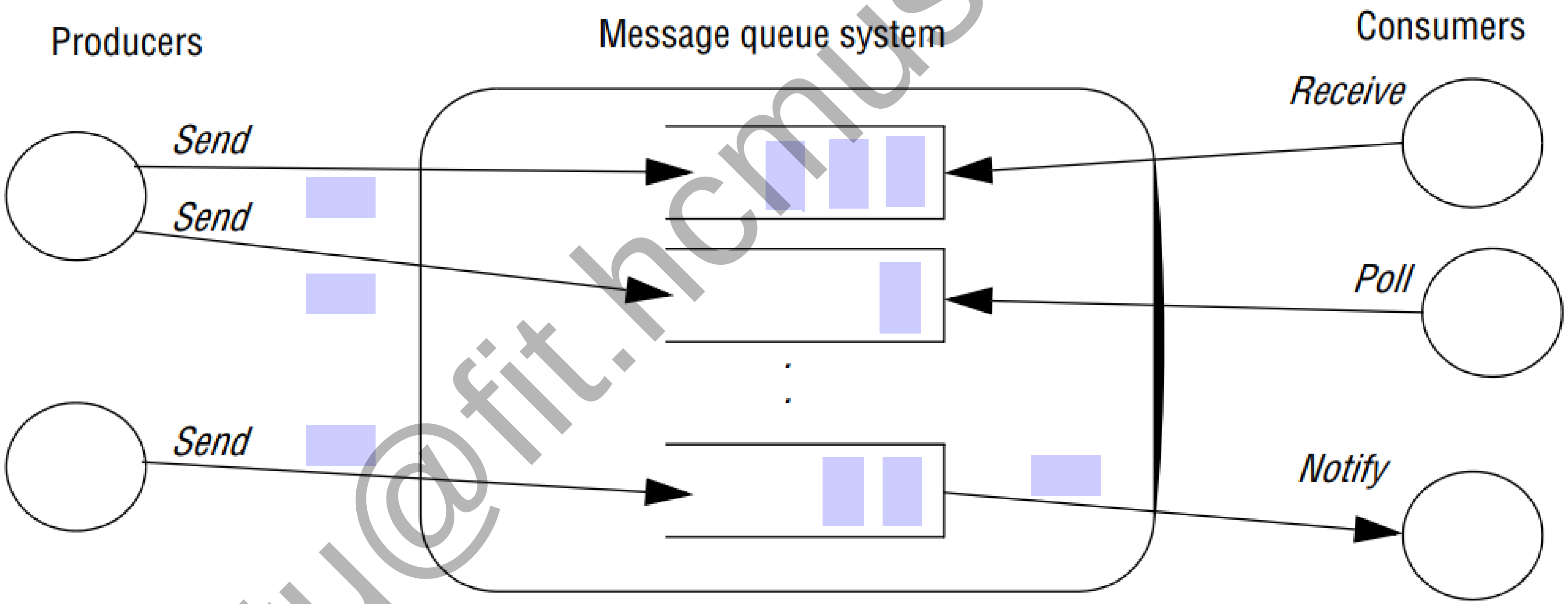
- ❑Publish-subscribe systems
  - ❑Implementation issues
    1. What is restriction of centralized implementations?
    2. Any solution?

Indirect Communication

- ❑Message queues
  - ❑Provide a point-to-point service
    - ❑Three styles of receive are generally supported
      - ❑ Blocking receive which will block until an appropriate message is available
      - ❑ Non-blocking receive (a polling operation), which will check the status of the queue and return a message if available, or a not available indication otherwise;
      - ❑ Notify operation, which will issue an event notification when a message is available in the associated queue

Indirect Communication

- Message queues



Outline

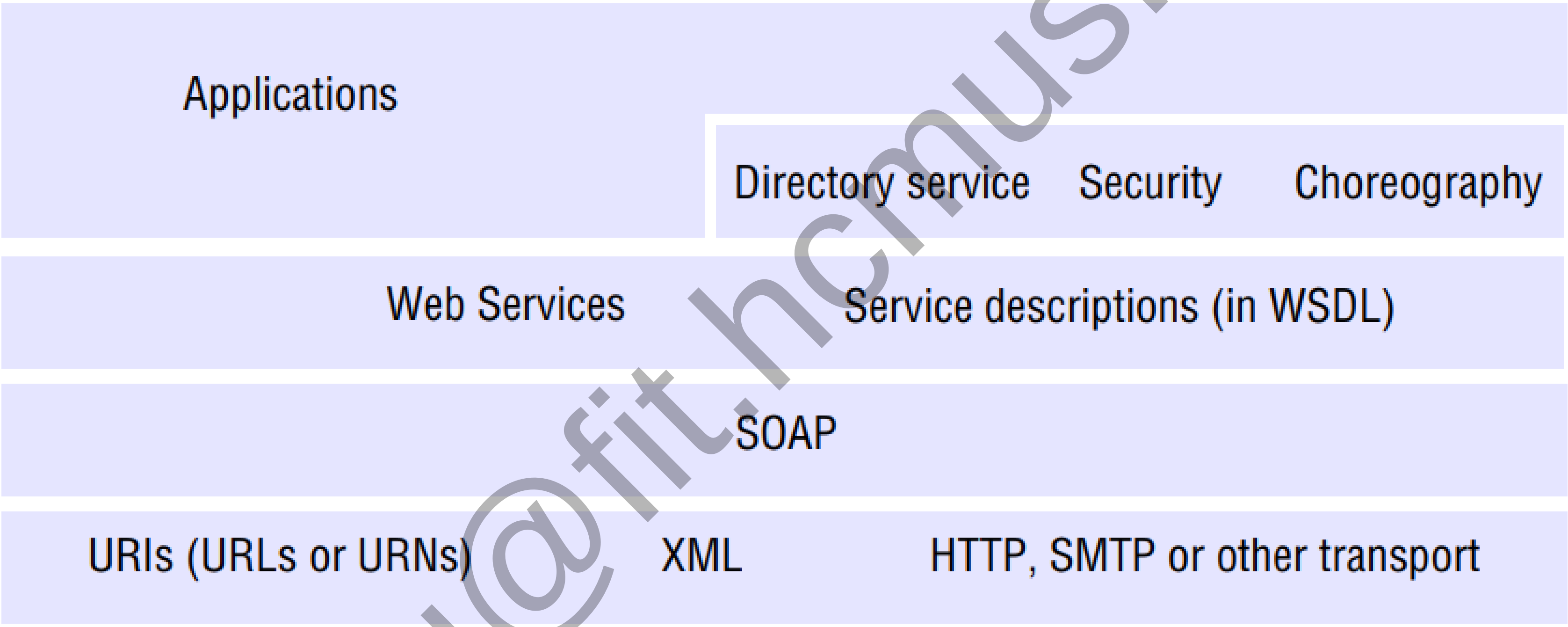
- ❑Introduction
- ❑Challenges
- ❑System Model
- ❑Indirect Communication
- ❑**Web Services**
- ❑Distributed Transactions

Web Services

- A web service provides a service **interface** enabling clients to **interact** with servers in a more general way than web browsers do. Clients access the operations in the interface of a web service by means of **requests and replies** formatted in **XML** and usually transmitted over HTTP

Web Services

- Web services infrastructure and components

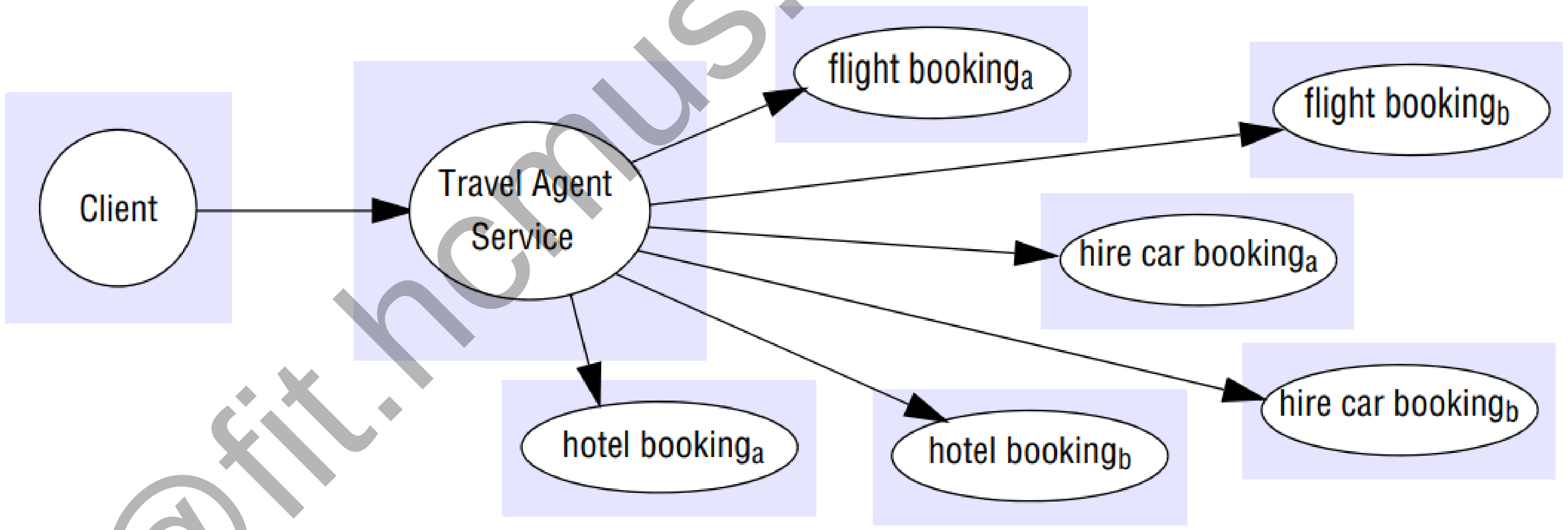


Web Services

- Commercial web servers

Web Services

- **Combination of web services**
  - Providing an interface for a web service allows its operations to be combined with those of other services to provide new functionality
  - Web services can be used with a variety of communication paradigms, including request-reply communication, asynchronous messaging



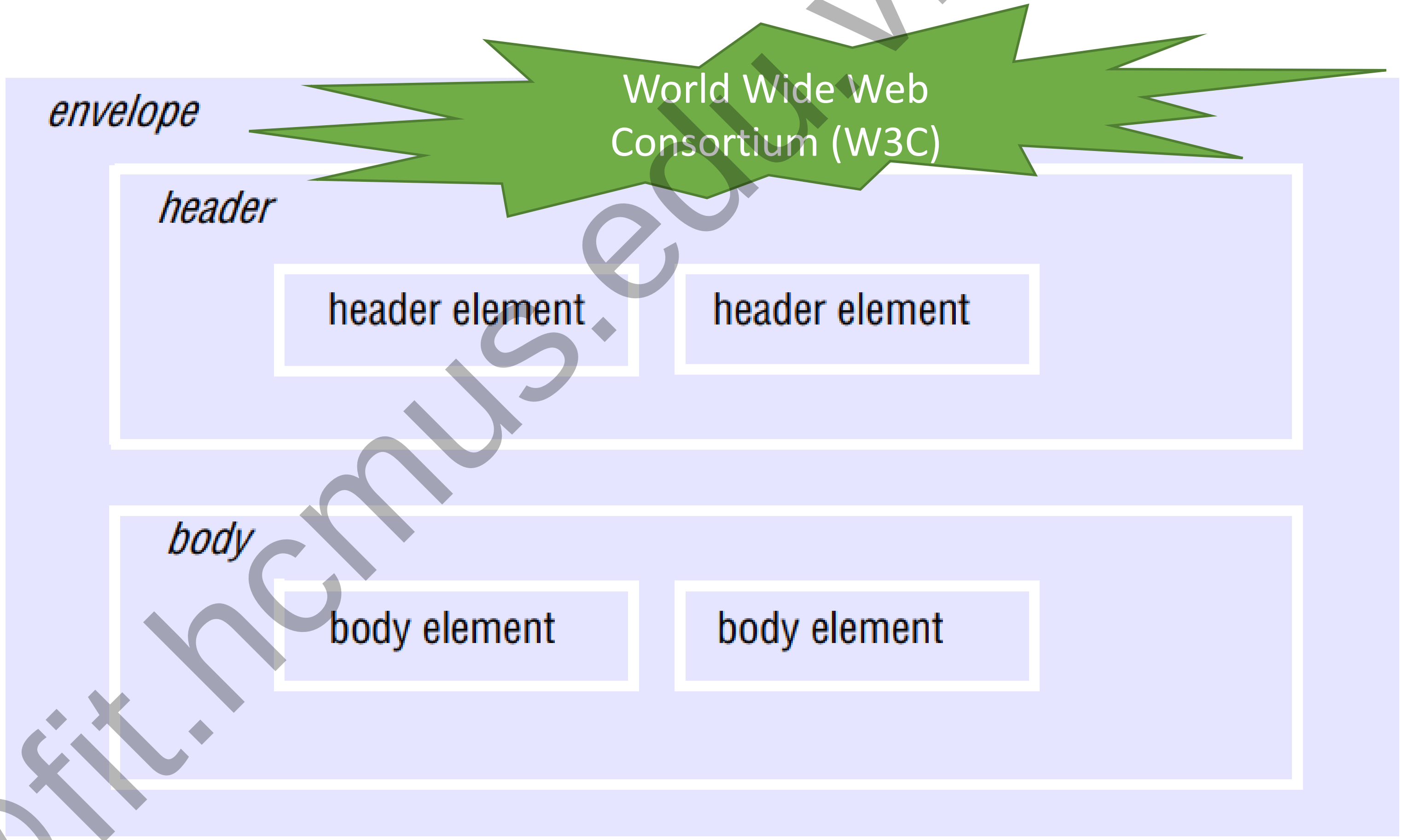
Web Services

- **Representation of messages**
  - XML
    - Both SOAP and the data it carries are represented in XML
  - REST(Representational State Transfer)
    - REST is an approach with a very constrained style of operation, in which clients use URLs and the HTTP operations GET, PUT, DELETE and POST to manipulate resources that are represented in XML

Web Services

❑SOAP

- ❑SOAP is designed to enable both client-server and asynchronous interaction c the Internet. It defines a scheme for using XML to represent the contents of request and reply message



Originally SOAP was based only on HTTP, but the current version is designed to use a variety of transport protocols including SMTP, TCP or UDP

Web Services

❑SOAP

```
POST /ROLMAN/ROLMANws.asmx HTTP/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://hostname/
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope>
  <soap:Header>
  <\soap:Head>
  <SOAP:Body>
  <app:CustomerServices xmlns:app=…>
       <Customer>
         <Name>………<\Name>
         <Address>……<\Address>
         <Phone>………<\Phone>
       <\Customer>
  <\app:CustomerServices>
<\soap:Body>
<\soap:Envelope>
```

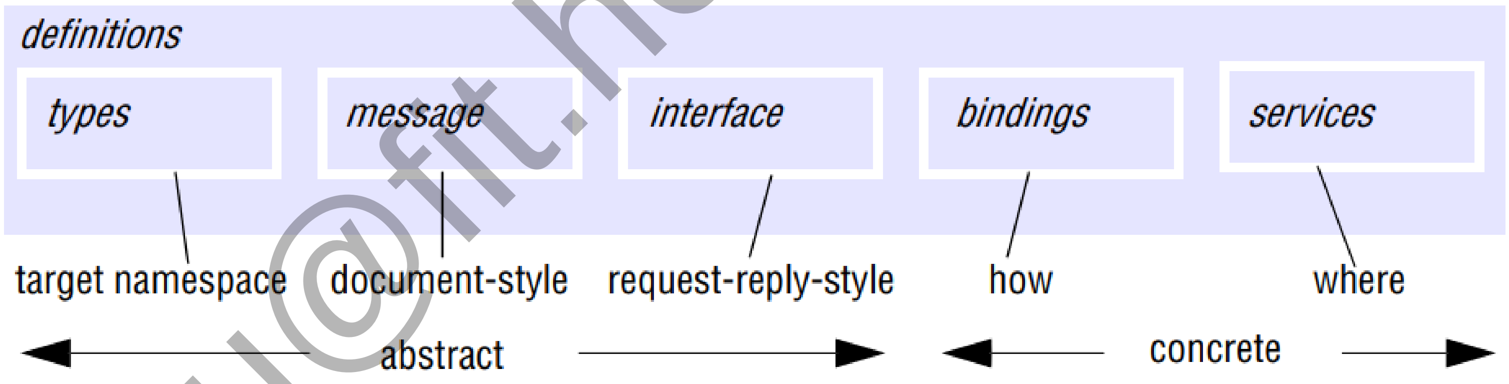
```
<soap:Envelope ...>
 <soap:Header>
  <wsse:Security>
   <wsse:BinarySecurityToken wsu:Id="myKey" ...>
... security token ...
   </wsse:BinarySecurityToken>
   <sig:Signature>
    <sig:SignedInfo>
     <sig:Reference URI="#myMsg">...digest...</sig:Reference>
    </sig:SignedInfo>
... signature ...
    <sig:KeyInfo>
     <wsse:SecurityTokenReference>
      <wsse:Reference URI="#myKey"/>
     </wsse:SecurityTokenReference>
    </sig:KeyInfo>
   </sig:Signature>
  </wsse:Security>
 </soap:Header>
 <soap:Body wsu:Id="myMsg">
  <app:StockSymbol ...>
... application sub messages ...
  </app:StockSymbol>
 </soap:Body>
</soap:Envelope>
```

Black – SOAP elements
Red – WSS elements/attributes
Green – XML Signature elements

Web Services

❑WSDL description

- ❑Interface definitions are needed to allow clients to communicate with services.
- ❑A service description forms the basis of an agreement between a client and a server as to the service on offer
- ❑Service descriptions are generally used to generate client stubs that automatically implement the correct behaviour for the client



Web Services

❑WSDL description

❑Message exchange patterns for WSDL operations

Name	*Messages sent by*			
	Client	*Server*	*Delivery*	*Fault message*
In-Out	*Request*	*Reply*		May replace *Reply*
In-Only	*Request*			No fault message
Robust In-Only	*Request*		Guaranteed	May be sent
Out-In	*Reply*	*Request*		May replace *Reply*
Out-Only		*Request*		No fault message
Robust Out-Only		*Request*	Guaranteed	May send fault

Web Services

❑WSDL description

```
– <wsdl:definitions name="AdderService" targetNamespace="http://adder.remote.ipojo.felix.apache.org/">
  – <wsdl:types>
    – <xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
      targetNamespace="http://adder.remote.ipojo.felix.apache.org/">
        <xsd:element name="add" type="tns:add"/>
      – <xsd:complexType name="add">
        – <xsd:sequence>
            <xsd:element name="arg0" type="xsd:int"/>
            <xsd:element name="arg1" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
        <xsd:element name="addResponse" type="tns:addResponse"/>
      – <xsd:complexType name="addResponse">
        – <xsd:sequence>
            <xsd:element name="return" type="xsd:int"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:schema>
    </wsdl:types>
  – <wsdl:message name="addResponse">
      <wsdl:part element="tns:addResponse" name="parameters"> </wsdl:part>
    </wsdl:message>
  – <wsdl:message name="add">
      <wsdl:part element="tns:add" name="parameters"> </wsdl:part>
    </wsdl:message>
  – <wsdl:portType name="AdderServicePortType">
    – <wsdl:operation name="add">
        <wsdl:input message="tns:add" name="add"> </wsdl:input>
        <wsdl:output message="tns:addResponse" name="addResponse"> </wsdl:output>
      </wsdl:operation>
    </wsdl:portType>
  – <wsdl:binding name="AdderServiceSoapBinding" type="tns:AdderServicePortType">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    – <wsdl:operation name="add">
        <soap:operation soapAction="" style="document"/>
      – <wsdl:input name="add">
          <soap:body use="literal"/>
        </wsdl:input>
      – <wsdl:output name="addResponse">
          <soap:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
```

Web Services

- **Service-oriented architecture**
  - A set of design principles whereby distributed systems are developed using sets of loosely coupled services that can be dynamically discovered and then communicate with each other or are coordinated through choreography to provide enhanced services.
- **Cloud computing**
  - A set of Internet-based application, storage and computing services sufficient to support most users' needs, thus enabling them to largely or totally dispense with local data storage and application software.

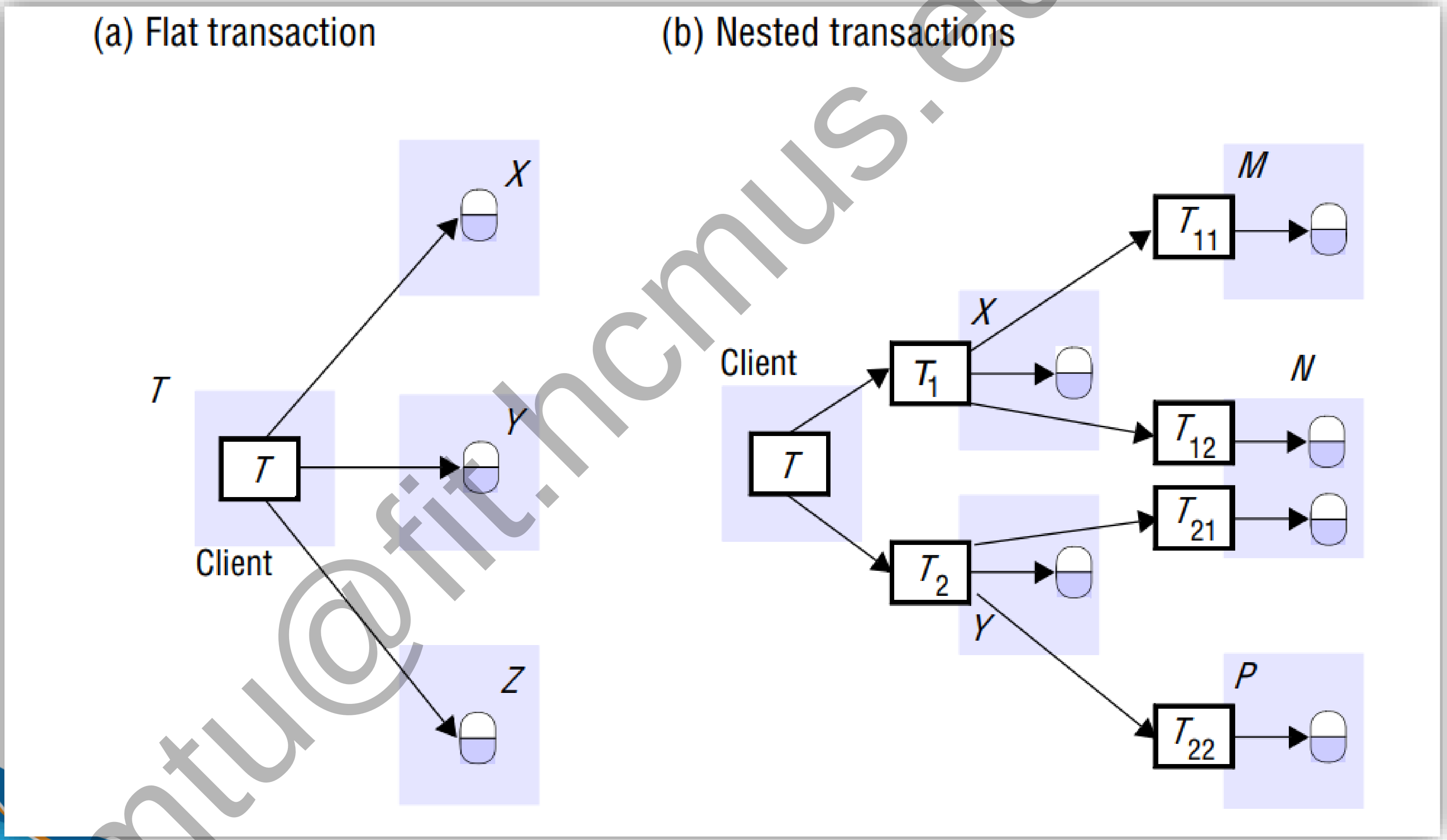
Web Services

Web service	*Description*
Amazon Elastic Compute Cloud (EC2)	Web-based service offering access to virtual machines of a given performance and storage capacity
Amazon Simple Storage Service (S3)	Web-based storage service for unstructured data
Amazon Simple DB	Web-based storage service for querying structured data
Amazon Simple Queue Service (SQS)	Hosted service supporting message queuing
Amazon Elastic MapReduce	Web-based service for distributed computation using the MapReduce model
Amazon Flexible Payments Service (FPS)	Web-based service supporting electronic payments

Outline

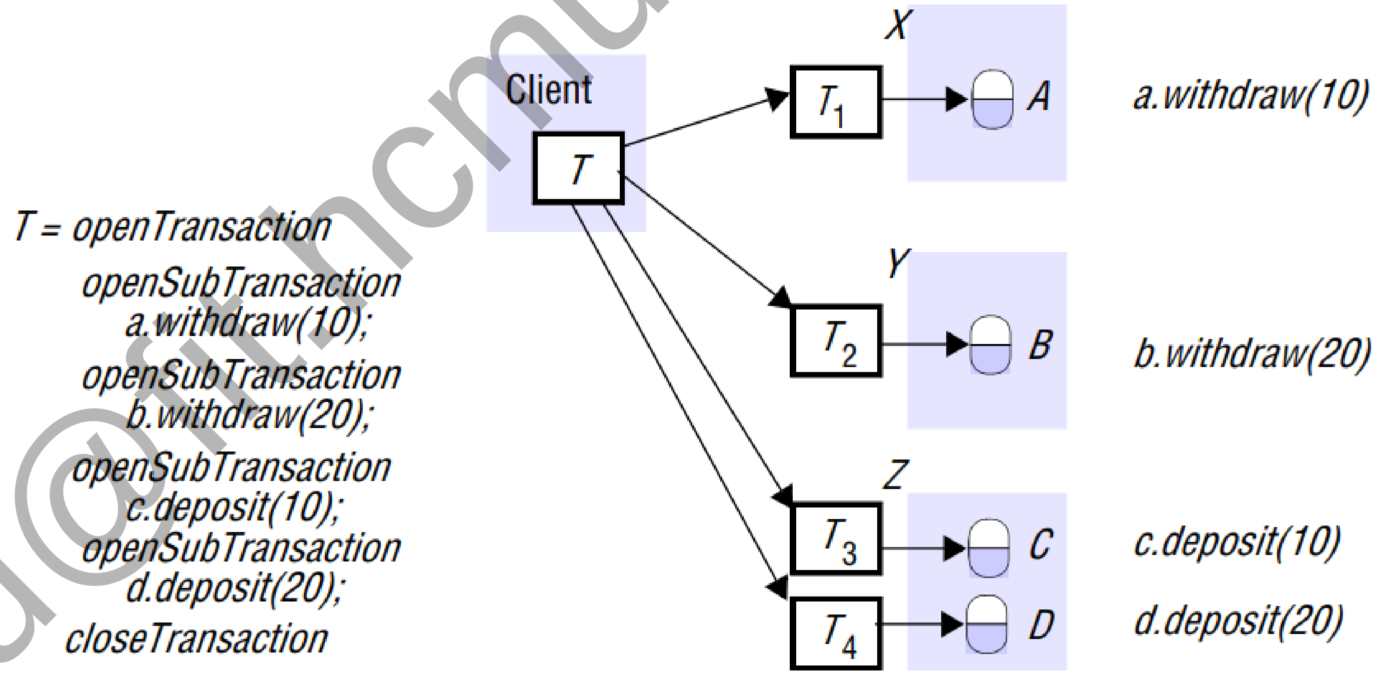
- ❑Introduction
- ❑Challenges
- ❑System Model
- ❑Indirect Communication
- ❑Web Services
- ❑**Distributed Transactions**

Distributed Transactions



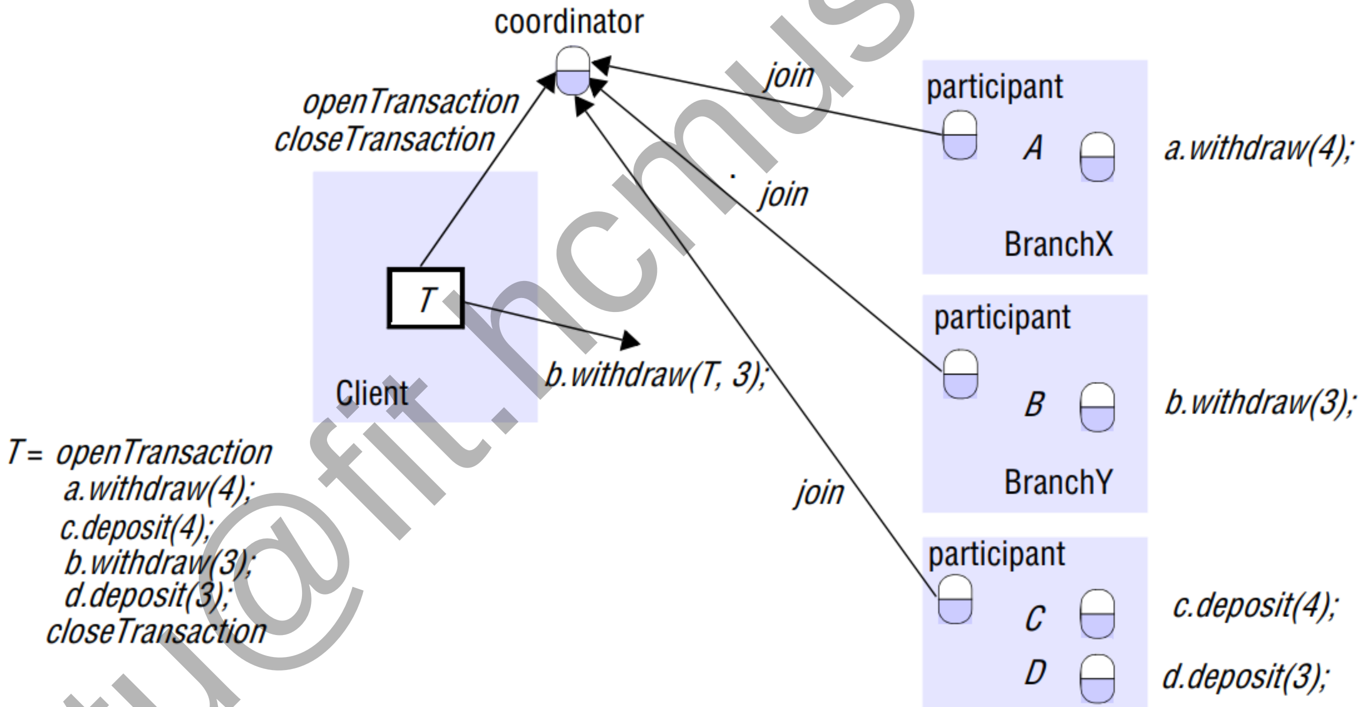
Distributed Transactions

Consider a distributed transaction in which a client transfers $10 from account A to C and then transfers $20 from B to D. Accounts A and B are at separate servers X and Y and accounts C and D are at server Z



Distributed Transactions

❑The coordinator of a distributed transaction



Note: the coordinator is in one of the servers, e.g., BranchX

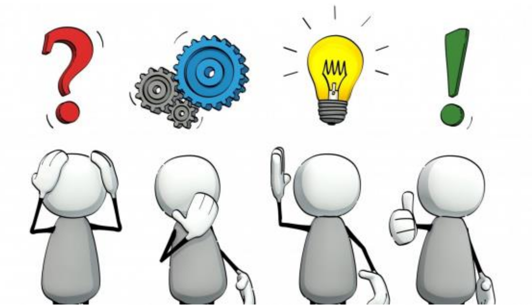
Distributed Transactions

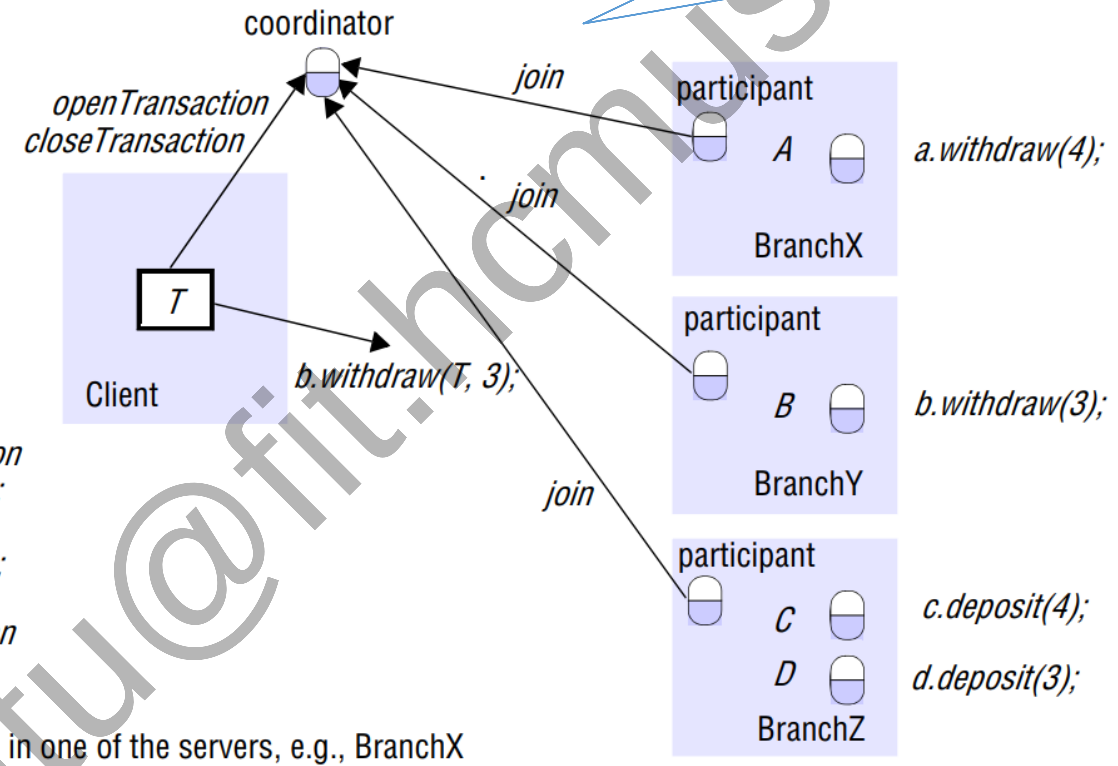
- Atomic commit protocol
  - Transaction commit protocols were devised in the early 1970s, and the two-phase commit protocol appeared in Gray [1978].
  - The atomicity property of transactions requires that when a distributed transaction comes to an end, either all of its operations are carried out or none of them

Distributed Transactions

- Atomic commit protocol

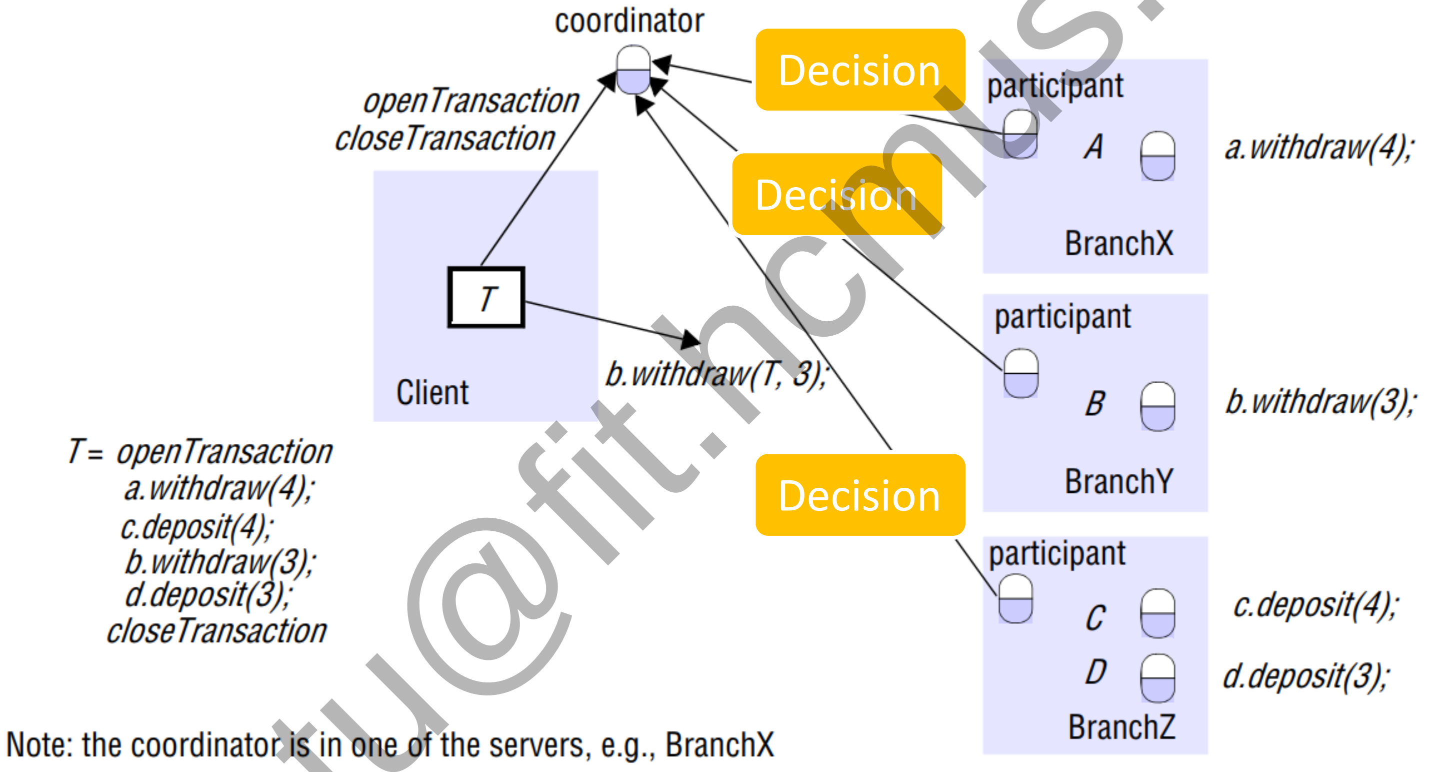
How to ensure atomicity property of transactions?





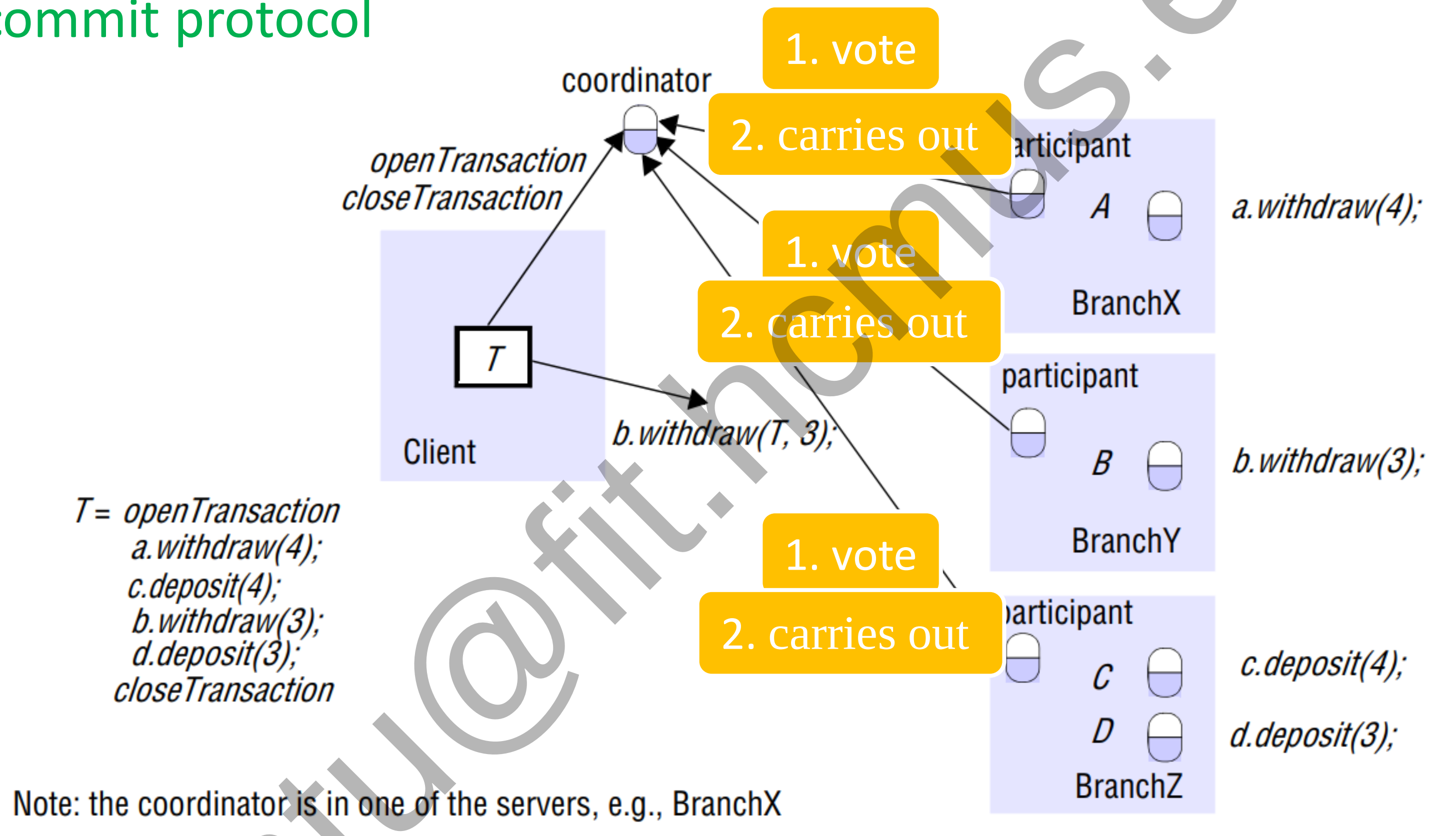
Distributed Transactions

❑Atomic commit protocol



Distributed Transactions

- Atomic commit protocol



Distributed Transactions

- The coordinator to communicate the commit or abort request to all of the participants in the transaction and to keep on repeating the request until all of them have acknowledged that they have carried it out => **one-phase atomic commit protocol**
- The **two-phase commit protocol** is designed to allow any participant to abort its part of a transaction.

Distributed Transactions

❑Operations for two-phase commit protocol

canCommit?(trans)→ Yes / No
Call from coordinator to participant to ask whether it can commit a transaction. Participant replies with its vote.

doCommit(trans)
Call from coordinator to participant to tell participant to commit its part of a transaction.

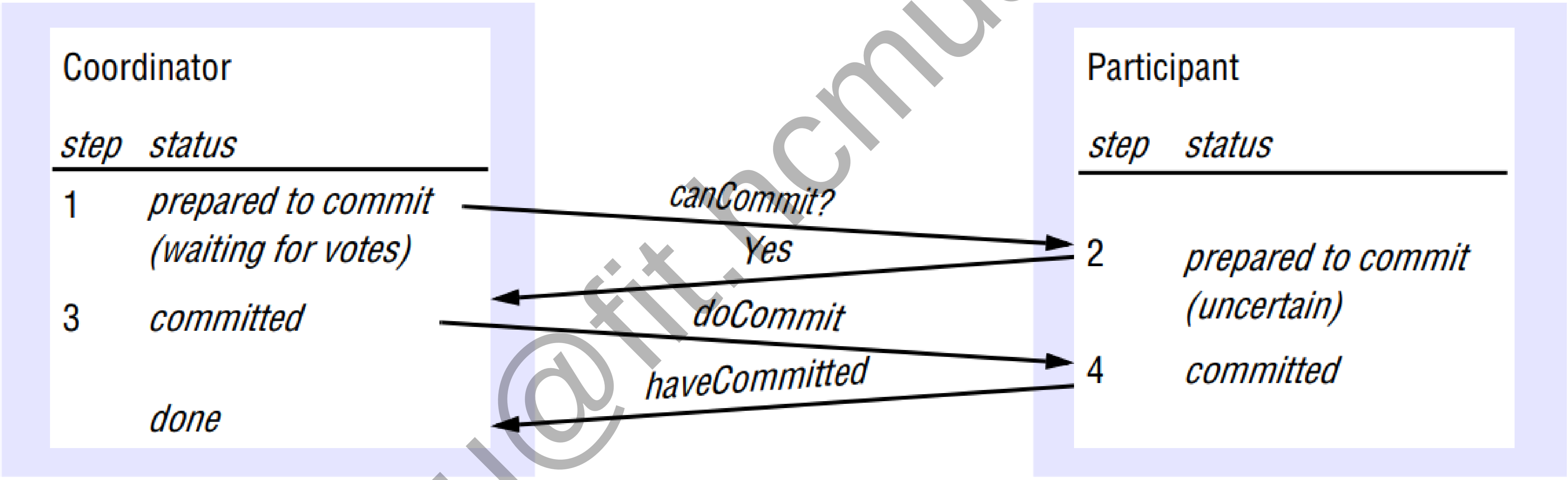
doAbort(trans)
Call from coordinator to participant to tell participant to abort its part of a transaction.

haveCommitted(trans, participant)
Call from participant to coordinator to confirm that it has committed the transaction.

getDecision(trans) → Yes / No
Call from participant to coordinator to ask for the decision on a transaction when it has voted *Yes* but has still had no reply after some delay.

Distributed Transactions

❑Communication in two-phase commit protocol



Distributed Transactions

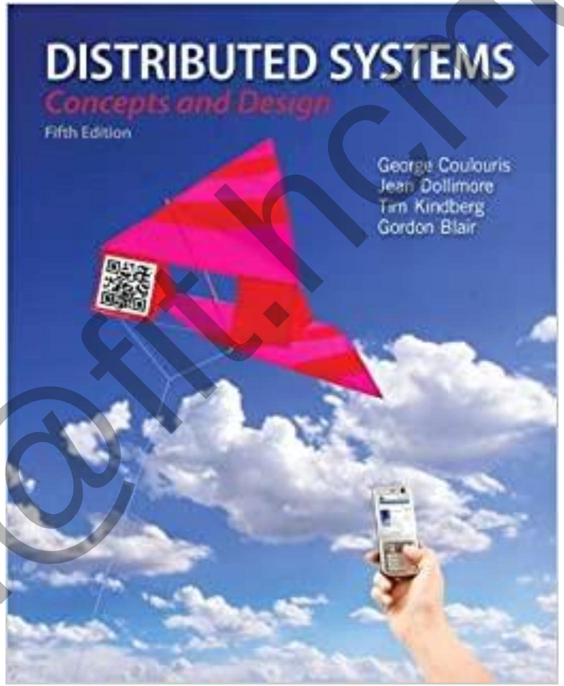
- ❑ **Have any problems in two-phase commit protocol?**

Solutions

References

- Distributed Systems: Concepts and Design - 2012

  https://www.amazon.com/Distributed-Systems-Concepts-Design-5th/dp/0132143011




Q
A