



Проектиране и интегриране на софтуерни системи

2015-2016 учебна година

# Съдържание

- ❖ Преподавателски екип
- ❖ Цели на курса
- ❖ Умения
- ❖ Изисквания
- ❖ Оценяване
- ❖ Литература
- ❖ Структура на курса
- ❖ График на лекциите
- ❖ Теми и график на упражненията

## Преподавателски екип

### ❖ проф. д-р Силвия Илиева

- катедра Софтуерни технологии, ФМИ
- Е-mail: [sylvia@acad.bg](mailto:sylvia@acad.bg)
- Тел.: 971 04 00 или 871 71 27

### ❖ доц. д-р Десислава Петрова – Антонова

- катедра Софтуерни технологии, ФМИ
- Е-mail: [d.petrova@fmi.uni-sofia.bg](mailto:d.petrova@fmi.uni-sofia.bg)
- Тел.: 971 04 00

### ❖ Борис Величков

# Цели на курса

- ❖ Въвеждане с основните концепции, проблеми и софтуерни решения, свързани с разпределените системи
- ❖ Запознаване на различните технологии за разработване на разпределени приложения
  - отдалечно извикване на процедури (RPC)
  - брокери на обекти (Object brokers)
  - опашки със съобщения (Message queues)
  - уеб услугите и стандарти като WSDL, UDDI и SOAP
  - ...
- ❖ Представяне на практически приложения на разпределените системи

## Умения

- ❖ извършване на сравнителен анализ и избор на подходяща комуникационна парадигма за решаване на конкретен проблем;
- ❖ прилагане на конкретно технологични решение при разработването на разпределени софтуерни системи;
- ❖ реализиране на собствена софтуерна услуга, както и интегриране на софтуерни услуги в други приложения.

## Изисквания

- ❖ Присъствие на лекциите и упражненията
- ❖ Прочитане на посочена литература
- ❖ Самостоятелна подготовка за курсовите задачи
- ❖ Регистриране за курса в системата Moodle

# Оценяване

- ❖ Оценка от тест – 60%
  - Оценка от тест, част 1 – 30%
  - Оценка от тест, част 2 – 30%
- ❖ Оценка от курсов проект – 40%

## Литература

- ❖ Coulouris G, J. Dollimore, T. Kindberg, G. Blair,  
Distributed systems – concepts and design, Addison  
Wesley, 2012
- ❖ Tanenbaum A. S., Maarten van Steen, Distributed  
Systems: Principles and Paradigms, Prentice Hall,  
2006

# Структура на курса

- ❖ 1. Characterization of Distributed Systems
- ❖ 2. System Models
- ❖ 3. Interprocess Communication
- ❖ 4. Remote Invocation
- ❖ 5. Indirect Communication
- ❖ 6. Dist. Objects and Components
- ❖ 7. Web services
- ❖ 9. Peer-to-Peer Systems
- ❖ 10. Distributed Transactions, Replication
- ❖ 11. Mobile and Ubiquitous Computing
- ❖ 12. Designing Distributed Systems: Google Case Study

# График на лекциите

Дата	ТЕМА
2.10.2015, 11:00-13:00 ч.	Общо събрание на ФМИ
9.10.2015, 11:00-13:00 ч.	Въведение в курса. Characterization of Distributed Systems
16.10.2015, 11:00-13:00 ч.	System Models
23.10.2015, 11:00-13:00 ч.	Interprocess Communication
30.10.2015, 11:00-13:00 ч.	Remote Invocation
6.11.2015, 11:00-13:00 ч.	Indirect Communication
13.11.2015, 11:00-13:00 ч.	Dist. Objects and Components
20.11.2015, 11:00-13:00 ч.	Предварителен тест
27.11.2015, 11:00-13:00 ч.	Web services
04.12.2015, 11:00-13:00 ч.	Peer-to-Peer Systems
11.12.2015, 11:00-13:00 ч.	Distributed Transactions, Replication
18.12.2015, 11:00-13:00 ч.	Mobile Computing или Ubiquitous Computing
08.01.2016, 11:00-13:00 ч.	Гост лекция
15.01.2016, 11:00-13:00 ч.	Гост лекция
22.01. 2016, 11:00-13:00 ч.	Предварителен изпит

# Теми на упражненията

- ❖ 1. Въведение – график на упражненията и задание за курсова работа
- ❖ 2. RPC/Java RMI
- ❖ 3. Publish-subscribe systems, message queues
- ❖ 4. Distributed objects and components, Enterprise JavaBeans/.NET Remoting
- ❖ 5. Web services
- ❖ 6. Peer-to-peer systems, Sockets

# График на упражненията

Дата, час	Тема
12.10,17.10.2015	Въведение – график на упражненията и задание за курсов проект
19.10,24.10.2015	RPC/Java RMI
26.10,31.10.2015	Publish-subscribe systems, message queues
02.11,07.11.2015	Distributed objects and components, Enterprise JavaBeans/.NET Remoting
09.11,14.11.2015	Web services
16.11,21.11.2015	Peer-to-peer systems, Sockets
23.11,28.11.2015	Предаване на курсов проект – етап 1
30.11,05.12.2015 07.12,12.12.2015 14.12,19.12.2015	Консултация за курсов проект
04.01,09.01.2016	Консултация за курсов проект
11.01,16.01.2016 18.01,23.01.2016	Заштита на курсов проект





# Characterization of Distributed Systems

## Lecture 1

# Outline

- ❖ Introduction
- ❖ Examples of distributed systems
- ❖ Trends in distributed systems
- ❖ Focus on resource sharing
- ❖ Challenges

# Introduction

## What is a distributed system?

# Definitions

- ❖ A distributed system is a collection of independent computers that appears to its users as a single coherent system

*Tanenbaum*

- ❖ A distributed system is the one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages

*Coulouris*

# Consequences of definition

- ❖ Concurrency
- ❖ No global clock
- ❖ Independent failures
- ❖ The prime motivation for constructing and using distributed systems stems from a desire to share resources
  - Resource:
    - ✓ hardware components (disks, printers),
    - ✓ software-defined entities (files, databases),
    - ✓ stream of video frames, audio, etc

## ❖ Wide range of applications

- from relatively localized systems (in a car or aircraft) to global scale systems involving millions of nodes
- from data-centric services to processor intensive tasks
- from systems built from very small and relatively primitive sensors to those incorporating powerful computational elements
- from embedded systems to ones that support a sophisticated interactive user experience

## Selected application domains and associated networked applications

<i>Finance and commerce</i>	eCommerce e.g. Amazon and eBay, PayPal, online banking and trading
<i>The information society</i>	Web information and search engines, ebooks, Wikipedia; social networking: Facebook and MySpace.
<i>Creative industries and entertainment</i>	online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr
<i>Healthcare</i>	health informatics, on online patient records, monitoring patients
<i>Education</i>	e-learning, virtual learning environments; distance learning
<i>Transport and logistics</i>	GPS in route finding systems, map services: Google Maps, Google Earth
<i>Science</i>	The Grid as an enabling technology for collaboration between scientists
<i>Environmental management</i>	sensor technology to monitor earthquakes, floods or tsunamis

# Examples of distributed systems - Web search

- ❖ Sophisticated distributed system infrastructure to support search
  - an underlying **physical infrastructure** consisting of very large numbers of networked computers located at data centres all around the world;
  - a **distributed file system** designed to support very large files and heavily optimized for the style of usage required by search and other Google applications (especially reading from files at high and sustained rates);
  - an associated **structured distributed storage system** that offers fast access to very large datasets;
  - **lock service** that offers distributed system functions such as distributed locking and agreement;
  - **programming model** that supports the management of very large parallel and distributed computations across the underlying physical infrastructure

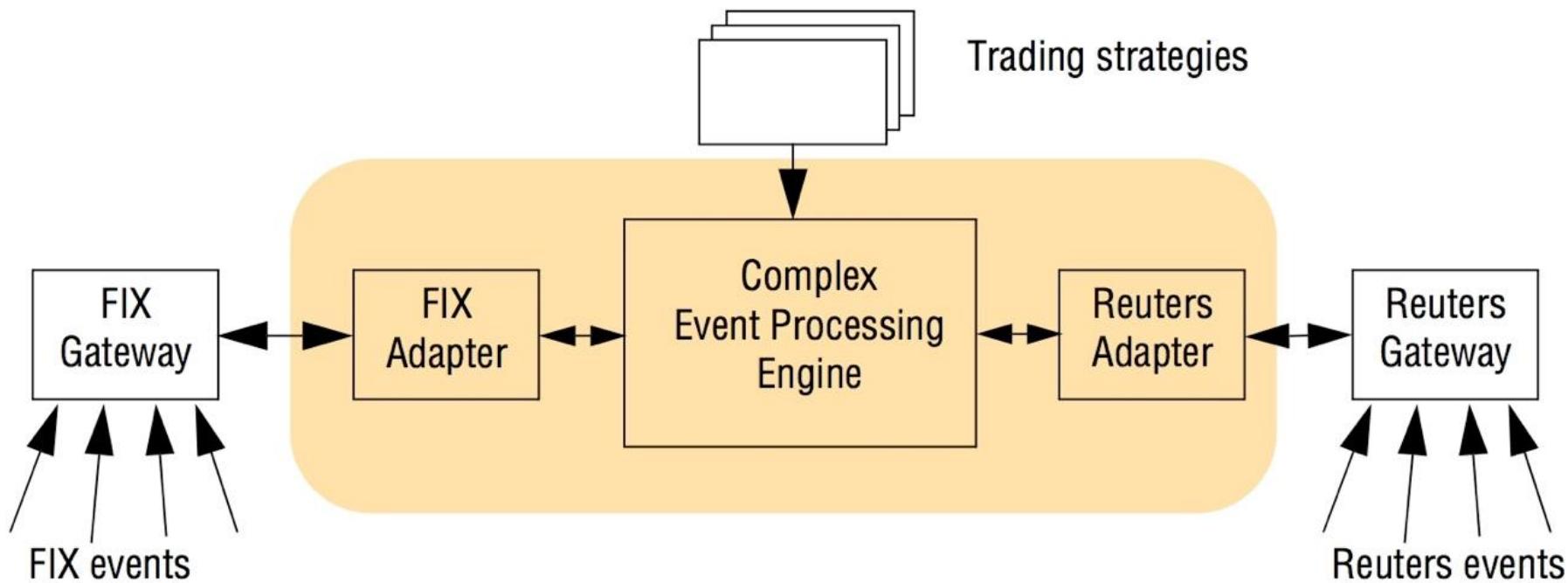
## ❖ Massively multiplayer online games (MMOGs)

- very large numbers of users interact through the Internet with a persistent virtual world
- Challenges
  - ✓ need for fast response times to preserve the user experience of the game.
  - ✓ include the real-time propagation of events to the many players and maintaining a consistent view of the shared world
- Solutions
  - ✓ Client server architecture
  - ✓ distributed architectures where the universe is partitioned across a (potentially very large) number of servers that may also be geographically distributed.
  - ✓ peer-to-peer technology

# Examples of distributed systems 3

## ❖ Financial trading

- real-time access to a wide range of information sources
- communication and processing of items of interest, known as *events* in distributed systems, with the need also to deliver events reliably and in a timely manner to potentially very large numbers of clients who have a stated interest in such information items



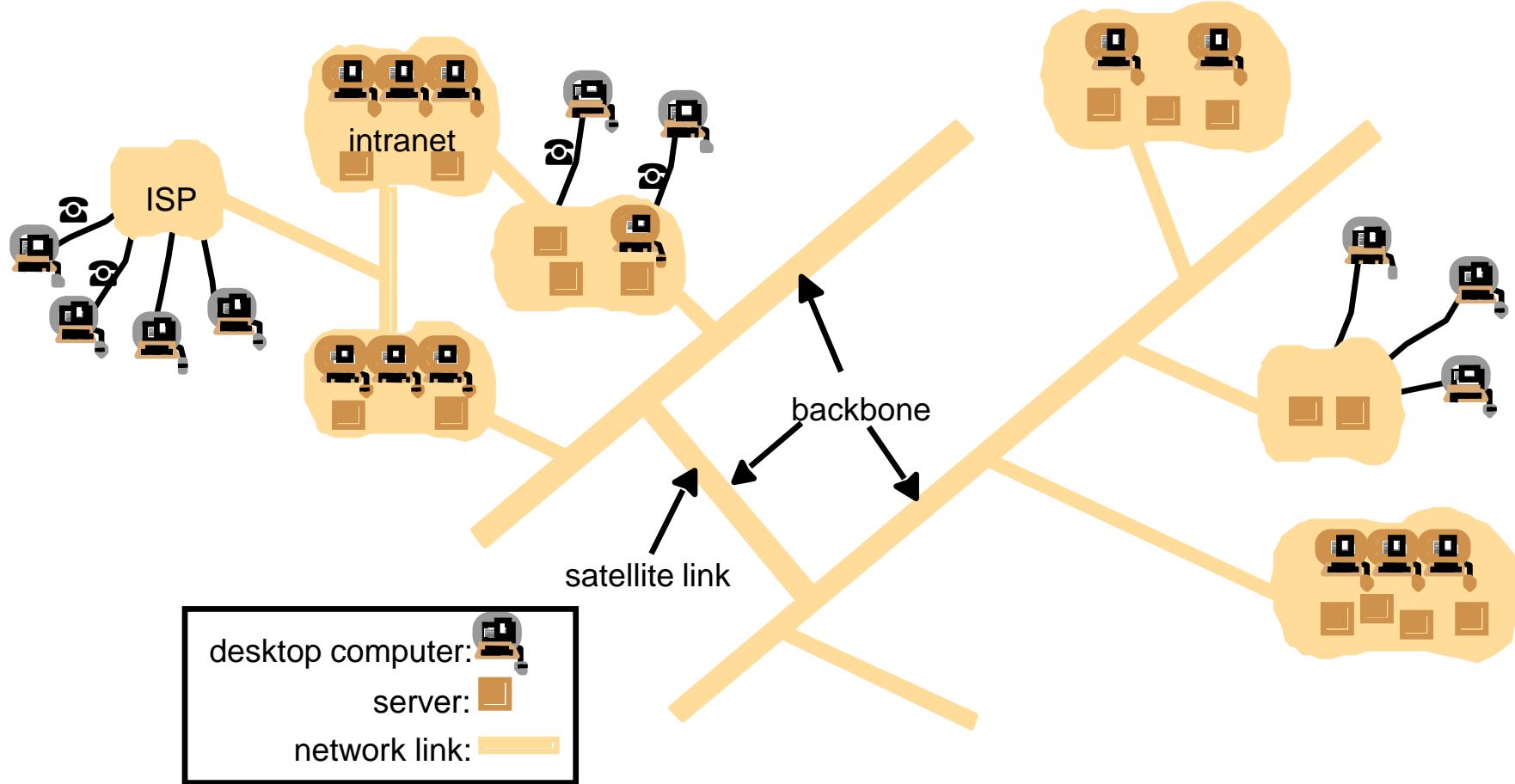
# Trends in distributed systems

- ❖ The emergence of pervasive networking technology;
- ❖ The emergence of ubiquitous computing coupled with the desire to support user mobility in distributed systems;
- ❖ The increasing demand for multimedia services;
- ❖ The view of distributed systems as a utility.

# Pervasive networking and the modern Internet

- ❖ The modern Internet is a vast interconnected collection of computer networks of many different types
  - wireless communication technologies such as WiFi, WiMAX, Bluetooth
  - third-generation mobile phone networks
- ➔ networking has become a pervasive resource and devices can be connected (if desired) at any time and in any place
- ❖ Internet protocols enabling a program running anywhere to address messages to programs anywhere else
- ❖ The Internet enables users, wherever they are, to make use of services such as the World Wide Web, email and file transfer

# Pervasive networking and the modern Internet 2



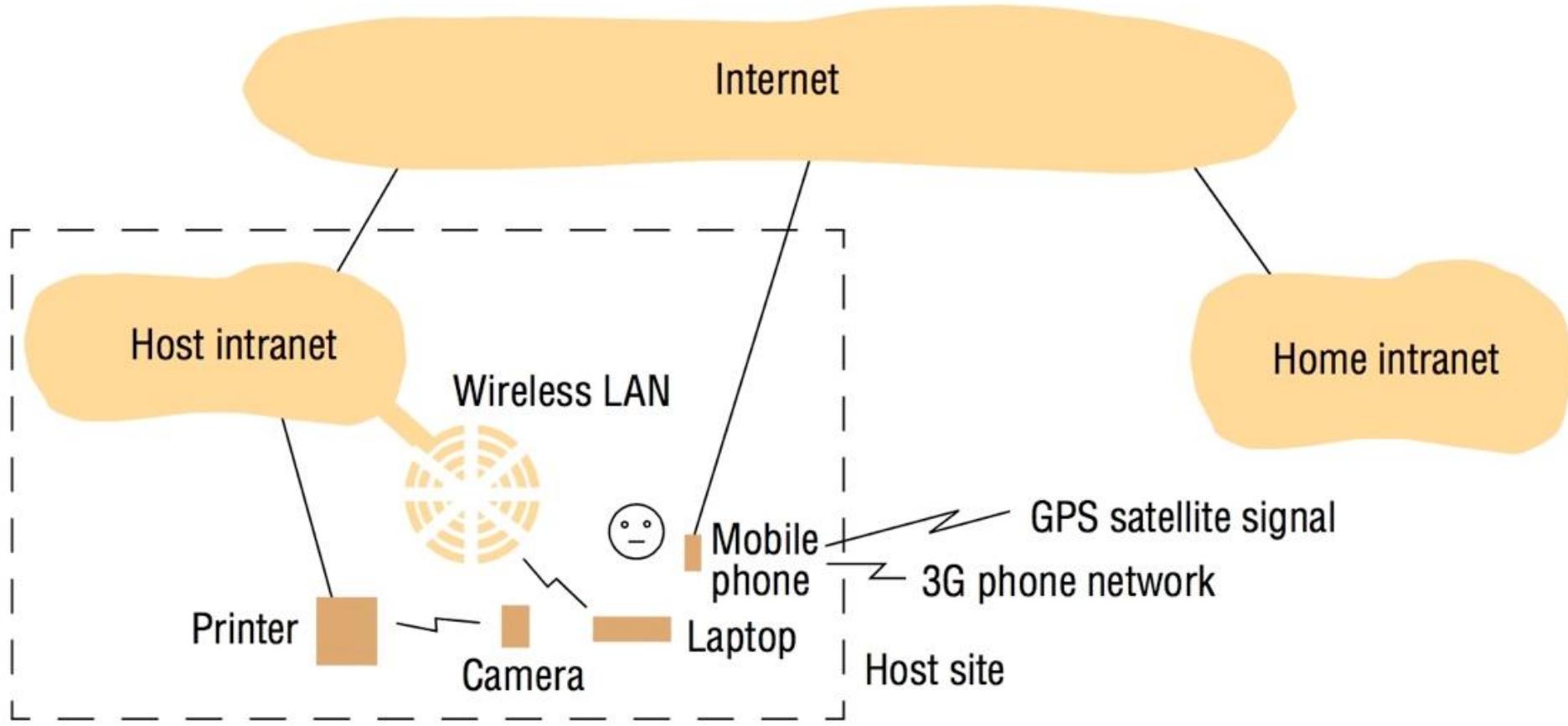
# Mobile and ubiquitous computing

- ❖ Small and portable computing devices, integrated into distributed systems, are:
  - Laptop computers.
  - Handheld devices, including mobile phones, smart phones, GPS-enabled devices, pagers, personal digital assistants (PDAs), video cameras and digital cameras.
  - Wearable devices (smart watches with functionality similar to a PDA).
  - Devices embedded in appliances (washing machines, hi-fi systems, car sand refrigerators).
- ❖ *Mobile computing* is the performance of computing tasks while the user is on the move, or visiting places other than their usual environment
  - Challenges
    - ✓ the need to deal with variable connectivity and indeed disconnection
    - ✓ the need to maintain operation in the face of device mobility

## Mobile and ubiquitous computing 2

- ❖ *Ubiquitous computing* is the harnessing of many small, cheap computational devices that are present in users' physical environments, including the home, office and even natural settings.
- ❖ Ubiquitous and mobile computing overlap, since the mobile user can in principle benefit from computers that are everywhere

# Mobile and ubiquitous computing 3



# Distributed multimedia systems

- ❖ A distributed multimedia system should be able to perform the same functions - storage, transmission and presentation - for continuous media types (audio and video)
  - to store and locate audio or video files,
  - to transmit them across the network (possibly in real time as the streams emerge from a video camera),
  - to support the presentation of the media types to the user and
  - to share the media types across a group of users
- ❖ Crucial characteristic of continuous media types is that they include a *temporal dimension*, and indeed, the integrity of the media type is fundamentally dependent on preserving real-time relationships between elements of a media type

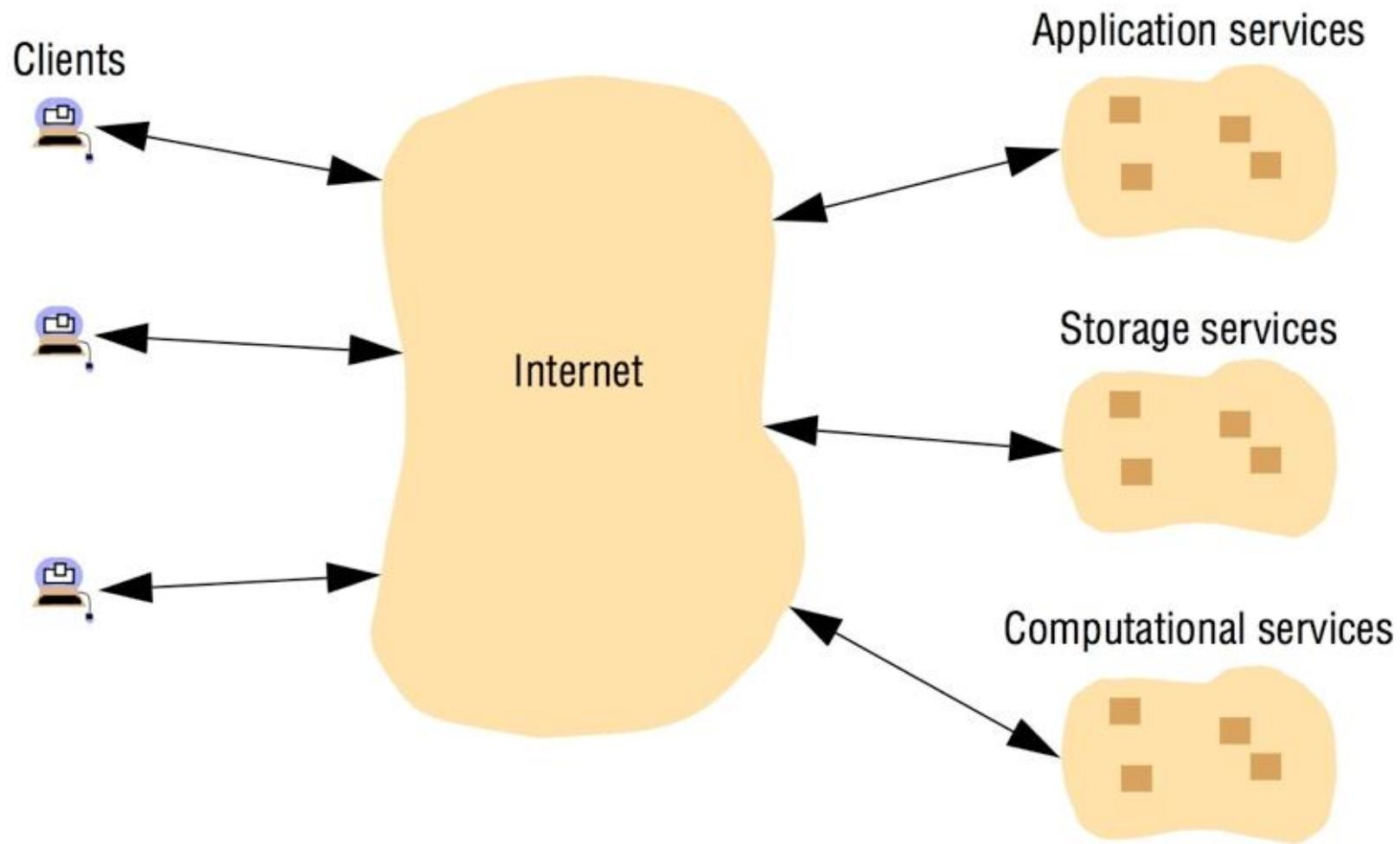
# Distributed multimedia systems 2

- ❖ The benefits of distributed multimedia computing are new (multimedia) services and applications can be provided on the desktop
  - access to live or pre-recorded television broadcasts,
  - access to film libraries
  - offering video-on-demand services,
  - access to music libraries,
  - the provision of audio and video conferencing facilities and integrated telephony features including IP telephony or related technologies such as Skype, a peer-to-peer alternative to IP telephony
- ❖ **Webcasting** is an application of distributed multimedia technology. It is the ability to broadcast continuous media, typically audio or video, over the Internet.
  - Distributed multimedia applications such as webcasting place considerable demands on the underlying distributed infrastructure

# Distributed computing as a utility

- ❖ Resources are provided by appropriate service suppliers and effectively rented rather than owned by the end user
  - Physical resources (storage and processing)
  - Software services (email and distributed calendars)
- ❖ A **cloud** is defined as 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
- ❖ A *cluster computer* is a set of interconnected computers that cooperate closely to provide a single, integrated high performance computing capability
- ❖ *Grid computing* - a precursor to the more general paradigm of cloud computing with a bias towards support for scientific applications

# Cloud computing



# Focus on resource sharing

- ❖ Patterns of resource sharing vary widely in their scope and in how closely users work together.
  - A search engine on the Web provides a facility to users throughout the world, users who need never come into contact with one another directly.
  - in *computer-supported cooperative working* (CSCW), a group of users who cooperate directly share resources such as documents in a small, closed group.
- ❖ Use the term *service* for a distinct part of a computer system that manages a collection of related resources and presents their functionality to users and applications
- ❖ Client – Server communication - remote invocation

# Challenges

What are the challenges?

# Challenges

- ❖ Heterogeneity
- ❖ Openness
- ❖ Security
- ❖ Scalability
- ❖ Failure handling
- ❖ Concurrency
- ❖ Transparency
- ❖ Quality of service

❖ Heterogeneity (that is, variety and difference) applies to all of the following:

- networks;
  - ✓ Internet protocols mask different sorts of networks
- computer hardware;
  - ✓ Different data representation (for example byte ordering)
- operating systems;
  - ✓ Different application programming interface to internet protocols
- programming languages;
  - ✓ different representations for characters and data structures
- implementations by different developers
  - ✓ Common standards

## ❖ Middleware

- software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages
  - ✓ CORBA (Common Object Request Broker), RMI
- In addition to solving the problems of heterogeneity, middleware provides a uniform computational model for use by the programmers of servers and distributed applications.
  - ✓ remote object invocation, remote event notification, remote SQL access and distributed transaction processing

## ❖ Mobile code

- program code that can be transferred from one computer to another and run at the destination
- The *virtual machine* approach provides a way of making code executable on a variety of host computers: the compiler for a particular language generates code for a virtual machine instead of a particular hardware order code

# Openness

- ❖ The openness of a computer system is the characteristic that determines whether the system can be extended and reimplemented in various ways.
- ❖ Open systems are characterized by the fact that their key interfaces are published.
- ❖ Open distributed systems are based on the provision of a uniform communication mechanism and published interfaces for access to shared resources.
- ❖ Open distributed systems can be constructed from heterogeneous hardware and software, possibly from different vendors. But the conformance of each component to the published standard must be carefully tested and verified if the system is to work correctly

- ❖ Security for information resources has three components:
  - **confidentiality** (protection against disclosure to unauthorized individuals),
  - **Integrity** (protection against alteration or corruption),
  - **availability** (protection against interference with the means to access the resources)
- ❖ The challenge is to send sensitive information in a message over a network in a secure manner including knowing for sure the identity of the user.
- ❖ Security challenges
  - Denial of service attacks
  - Security of mobile code

- ❖ A system is described as *scalable* if it will remain effective when there is a significant increase in the number of resources and the number of users
- ❖ Challenges
  - Controlling the cost of physical resources
  - Controlling the performance loss
  - Preventing software resources running out
  - Avoiding performance bottlenecks

- ❖ Failures in a distributed system are partial – that is, some components fail while others continue to function
- ❖ Failure handling techniques
  - Detecting failures
  - Masking failures
  - Tolerating failures
  - Recovery from failures
  - Redundancy
- ❖ The *availability* of a system is a measure of the proportion of time that it is available for use.

# Concurrency

- ❖ any object that represents a shared resource in a distributed system must be responsible for ensuring that it operates correctly in a concurrent environment.
- ❖ For an object to be safe in a concurrent environment, its operations must be synchronized in such a way that its data remains consistent.
  - standard techniques such as semaphores

# Transparency

- ❖ *Access transparency* enables local and remote resources to be accessed using identical operations.
- ❖ *Location transparency* enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).
- ❖ *Concurrency transparency* enables several processes to operate concurrently using shared resources without interference between them.
- ❖ *Replication transparency* enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
- ❖ *Failure transparency* enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- ❖ *Mobility transparency* allows the movement of resources and clients within a system without affecting the operation of users or programs.
- ❖ *Performance transparency* allows the system to be reconfigured to improve performance as loads vary.
- ❖ *Scaling transparency* allows the system and applications to expand in scale without change to the system structure or the application algorithms.

# Quality of service

- ❖ The main nonfunctional properties of systems that affect the quality of the Service
  - reliability,
  - security
  - performance
- ❖ Adaptability - changing system configurations and resource availability
- ❖ Time-critical data – streams of data that are required to be processed or transferred from one process to another at a fixed rate





# System Models

## Lecture 2

# Outline

- ❖ Introduction
- ❖ Physical models
- ❖ Architectural models
- ❖ Fundamental models

# Introduction

- ❖ *Physical models* are the most explicit way in which to describe a system; they capture the hardware composition of a system in terms of the computers (and other devices, such as mobile phones) and their interconnecting networks.
- ❖ *Architectural models* describe a system in terms of the computational and communication tasks performed by its computational elements; the computational elements being individual computers or aggregates of them supported by appropriate network interconnections.
- ❖ *Fundamental models* take an abstract perspective in order to examine individual aspects of a distributed system.
  - *interaction models*, which consider the structure and sequencing of the communication between the elements of the system;
  - *failure models*, which consider the ways in which a system may fail to operate correctly and;
  - *security models*, which consider how the system is protected against attempts to interfere with its correct operation or to steal its data.

- ❖ Baseline physical model
- ❖ Three generations of distributed systems
  - Early distributed systems
  - Internet-scale distributed systems
  - Contemporary distributed systems
    - ✓ Mobile computing
    - ✓ Ubiquitous computing
    - ✓ Cloud computing
- ❖ Distributed systems of systems

# Generations of distributed systems

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

## ❖ Architectural elements

- Communicating entities
- Communication paradigms
- Roles and responsibilities
- Placement

## ❖ Architectural patterns

## ❖ Associated middleware solutions

# 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?
- ❖ How are they mapped on to the physical distributed infrastructure (what is their *placement*)?

## ❖ From a system perspective

- Processes – nodes and threads

## ❖ From a programming perspective

- *Objects*
- *Components*
- *Web services*
  - ✓ a software application identified by a URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artefacts. A Web service supports direct interactions with other software agents using XML-based message exchanges via Internet-based protocols.

## ❖ interprocess communication

- message-passing primitives,
- direct access to the API offered by Internet protocols (socket programming) and
- support for multicast communication

## ❖ remote invocation

- *Request-reply protocols*
- *Remote procedure calls*
- *Remote method invocation*

## ❖ indirect communication

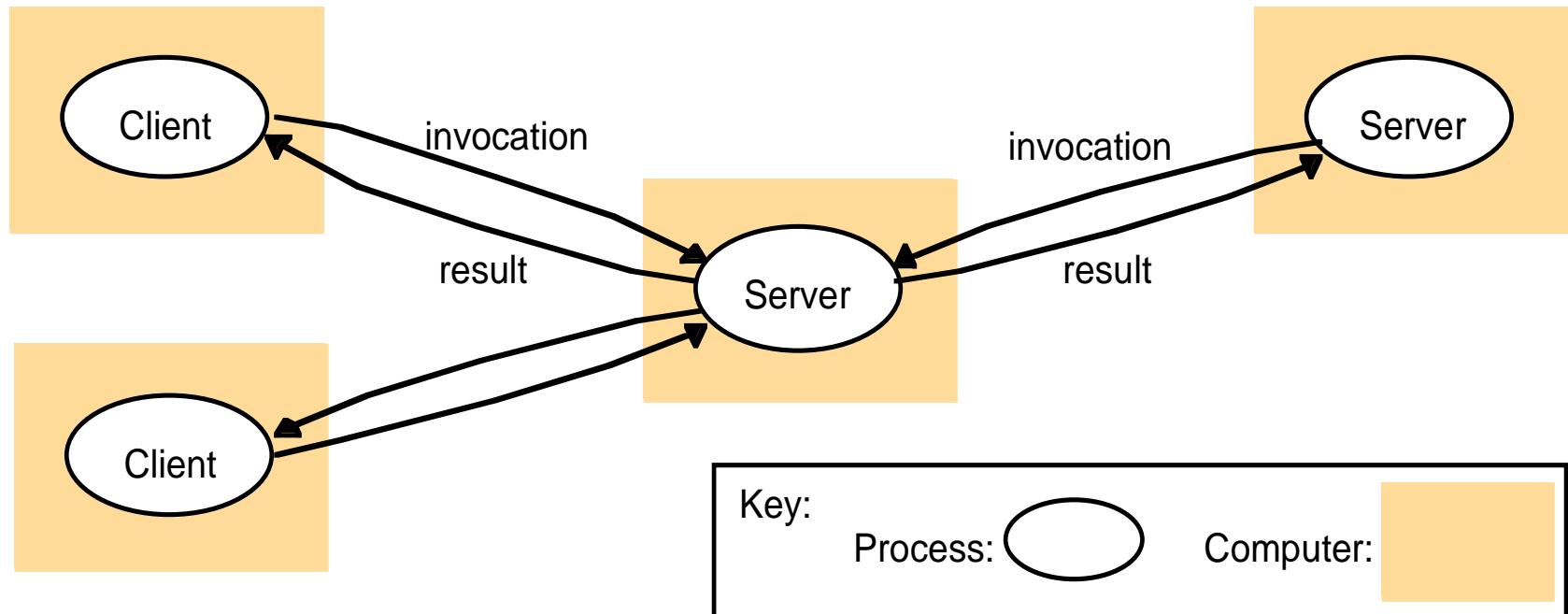
- *Group communication*
- *Publish-subscribe systems*
- *Message queues*
- *Tuple spaces*
- *Distributed shared memory*

# Communicating entities and communication paradigms

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

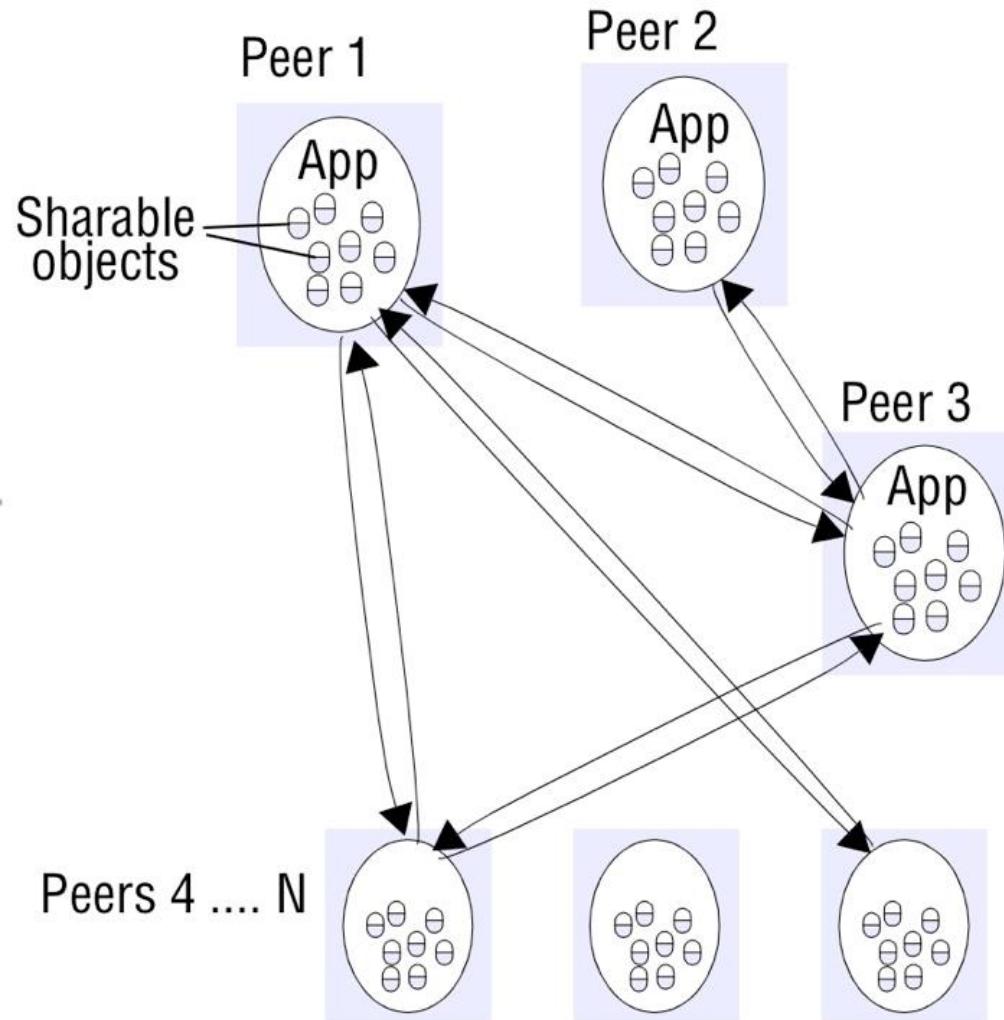
# Roles and responsibilities

## ❖ Client-server



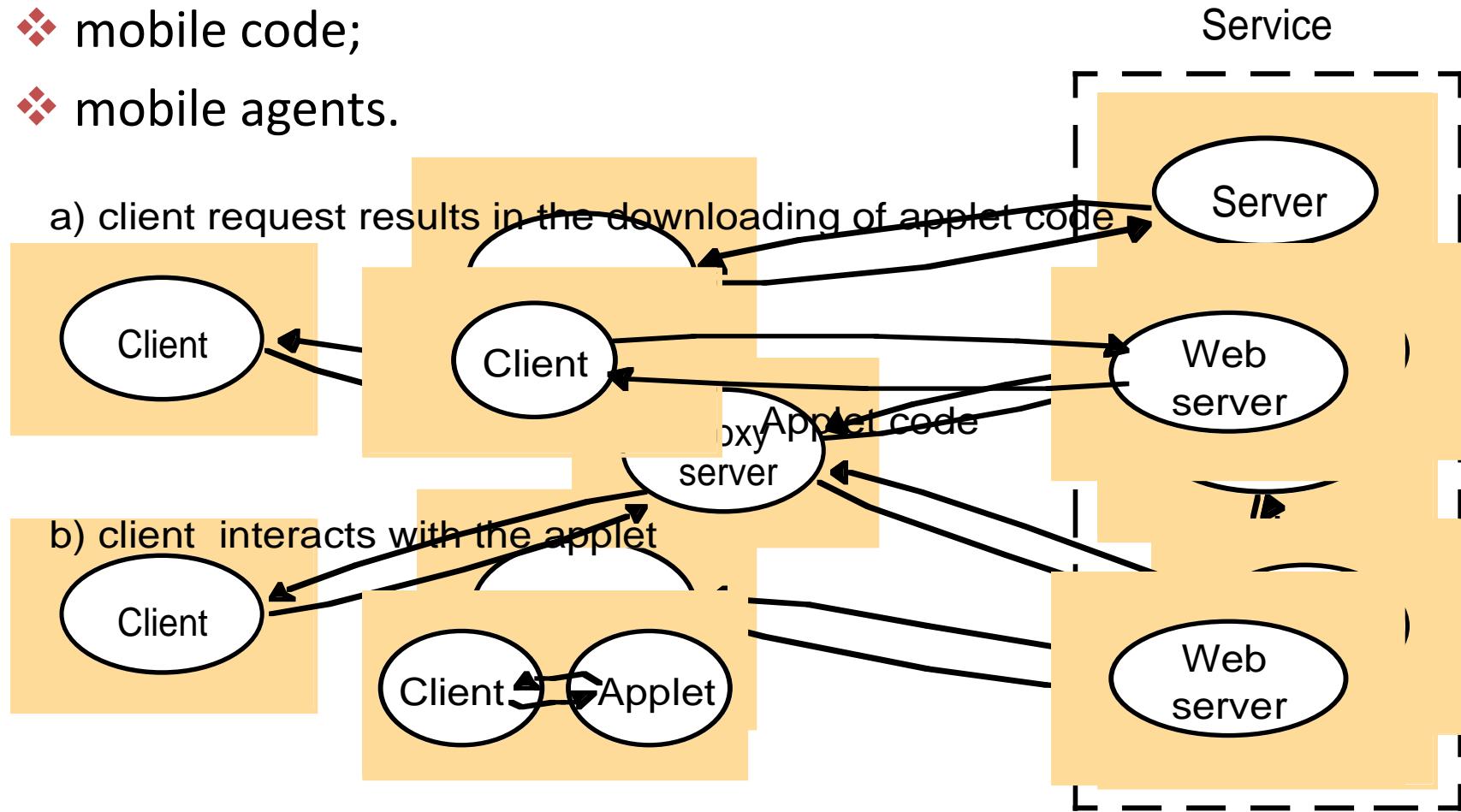
# Roles and responsibilities

## ❖ Peer-to-peer



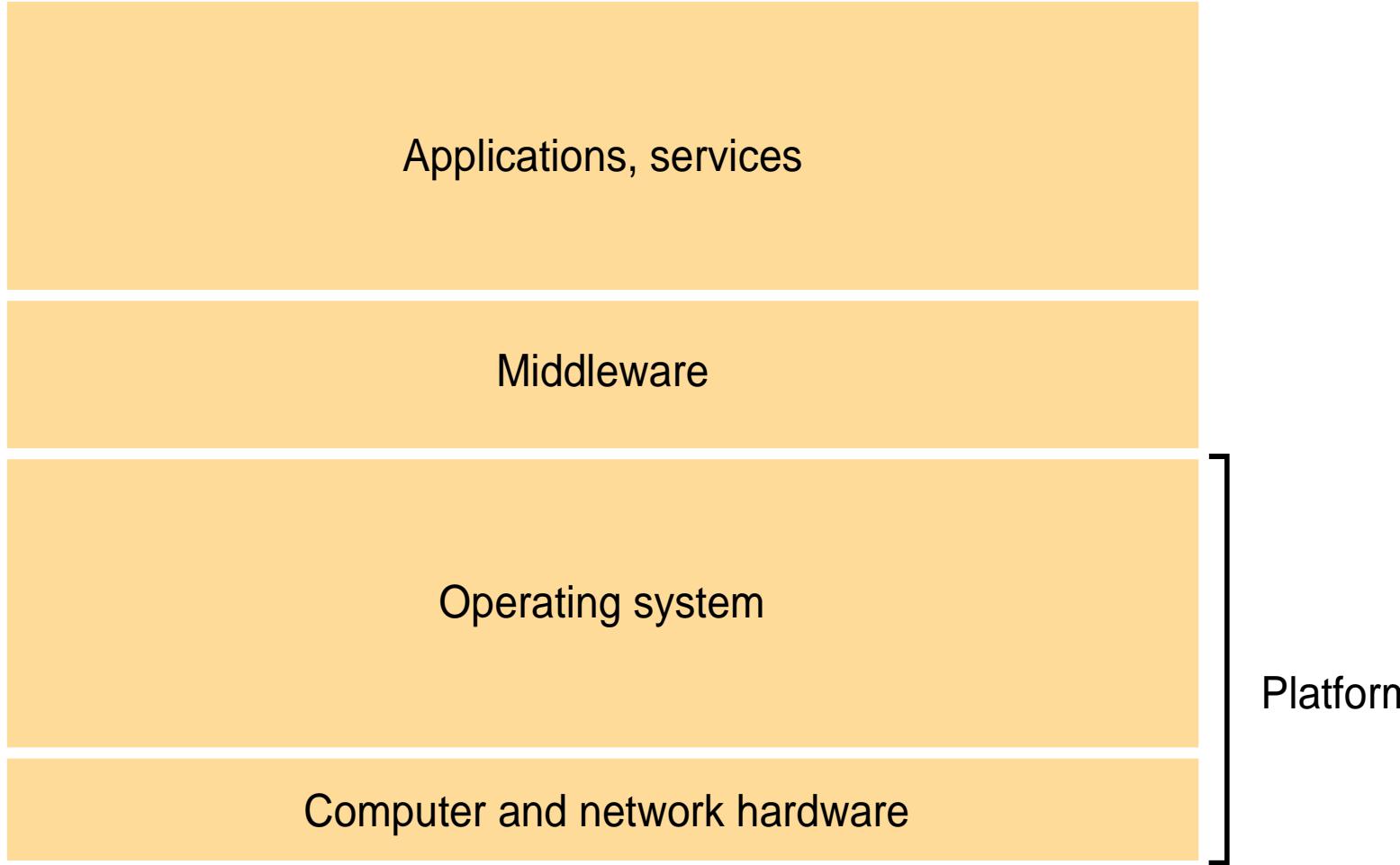
# Placement

- ❖ mapping of services to multiple servers;
- ❖ caching;
- ❖ mobile code;
- ❖ mobile agents.



- ❖ Architectural elements
- ❖ Architectural patterns
  - Layering
  - Tiered architecture
  - Other commonly occurring patterns
- ❖ Associated middleware solutions

# Layering



## Layering 2

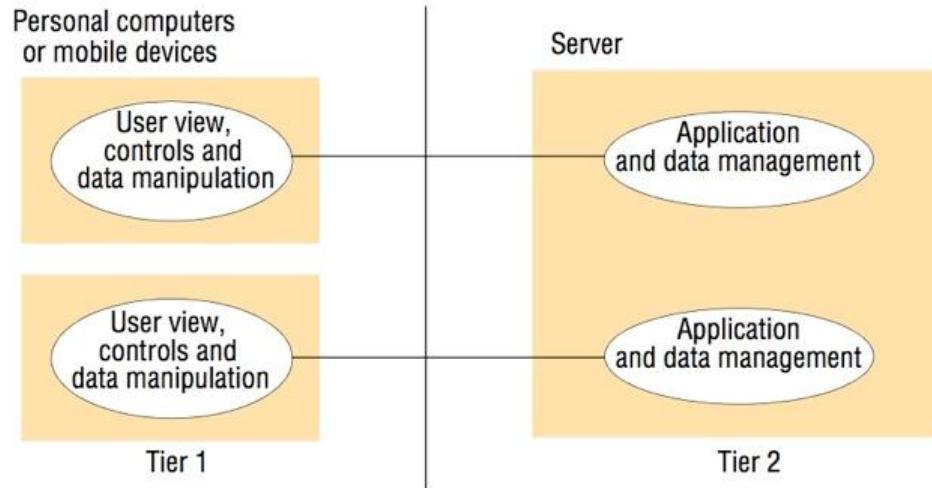
- ❖ Platform - the lowest-level hardware and software layers
  - Intel x86/Windows, Intel x86/Solaris, Intel x86/Mac OS X, Intel x86/Linux and ARM/Symbian
- ❖ Middleware - layer of software whose purpose is to mask heterogeneity and to provide a convenient programming model to application programmers

# Tiered architecture

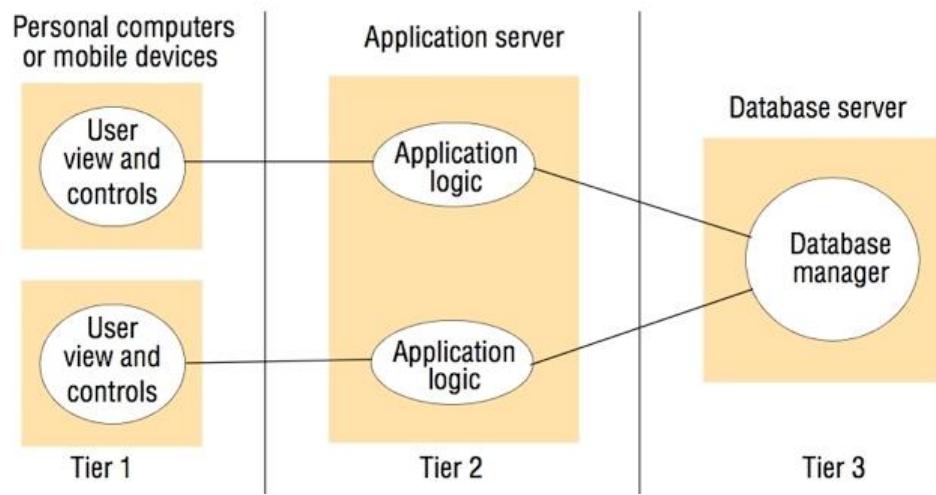
- ❖ the **presentation logic**, which is concerned with handling user interaction and updating the view of the application as presented to the user;
- ❖ the **application logic**, which is concerned with the detailed application-specific processing associated with the application (also referred to as the business logic, although the concept is not limited only to business applications);
- ❖ the **data logic**, which is concerned with the persistent storage of the application, typically in a database management system.

# Two-tier and three-tier architectures

a)



b)



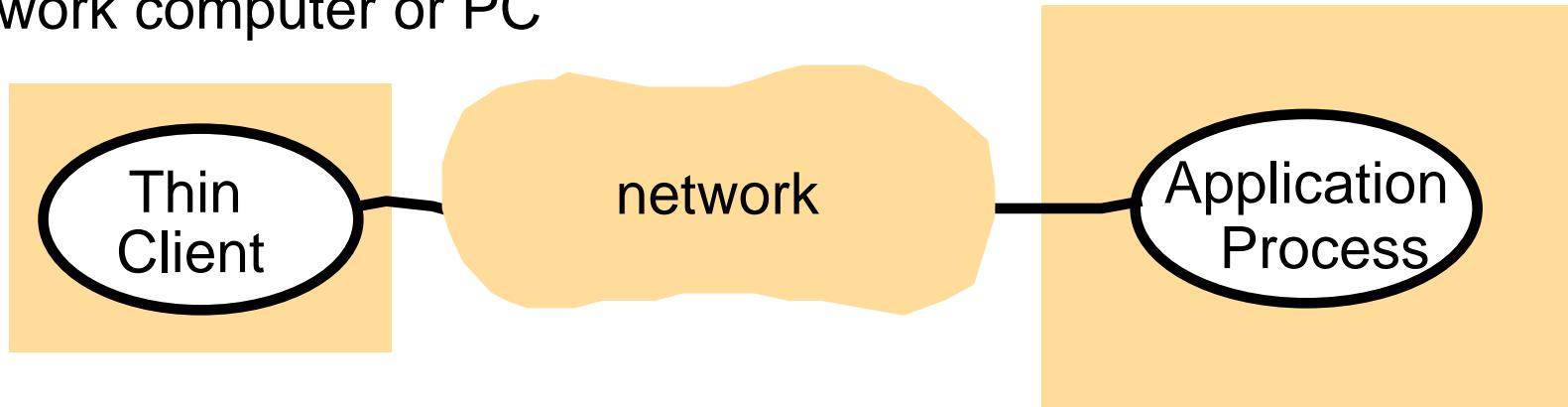
# Thin clients

## ❖ Thin clients

- a software layer that supports a window-based user interface that is local to the user while executing application programs or, more generally, accessing services on a remote computer

Network computer or PC

Compute server



## Other commonly occurring patterns

- ❖ The proxy pattern
  - a proxy is created in the local address space to represent the remote object
- ❖ The use of brokerage in web services

Further examples of architectural patterns related to distributed systems can be found in Bushmann, F., Henney, K. and Schmidt, D.C. (2007). *Pattern-Oriented Software Architecture: A Pattern for Distributed Computing*, NewYork: John Wiley & Sons.

- ❖ Architectural elements
- ❖ Architectural patterns
- ❖ Associated middleware solutions
  - Categories of middleware
  - Limitations of middleware

# Categories of middleware

<i>Major categories:</i>	<i>Subcategory</i>	<i>Example systems</i>
<i>Distributed objects (Chapters 5, 8)</i>	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
<i>Distributed components (Chapter 8)</i>	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
<i>Publish-subscribe systems (Chapter 6)</i>	-	CORBA Event Service
	-	Scribe
	-	JMS
<i>Message queues (Chapter 6)</i>	-	Websphere MQ
	-	JMS
<i>Web services (Chapter 9)</i>	Web services	Apache Axis
	Grid services	The Globus Toolkit
<i>Peer-to-peer (Chapter 10)</i>	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

# Limitations of middleware

- ❖ The right underlying middleware behaviour is a function of the requirements of a given application or set of applications and the associated environmental context, such as the state and style of the underlying network

## ❖ Content

- only the essential ingredients that we need to consider in order to understand and reason about some aspects of a system's behaviour

## ❖ Purpose

- To make explicit all the relevant assumptions about the systems we are modelling.
- To make generalizations concerning what is possible or impossible, given those assumptions. The generalizations may take the form of general-purpose algorithms or desirable properties that are guaranteed. The guarantees are dependent on logical analysis and, where appropriate, mathematical proof.

## ❖ The aspects of distributed systems that we wish to capture in our fundamental models

- Interaction
- Failure
- Security

## ❖ Performance of communication channels

- Latency
- Bandwidth
  - ✓ The *bandwidth* of a computer network is the total amount of information that can be transmitted over it in a given time. When a large number of communication channels are using the same network, they have to share the available bandwidth
- Jitter
  - ✓ *Jitter* is the variation in the time taken to deliver a series of messages. Jitter is relevant to multimedia data. For example, if consecutive samples of audio data are played with differing time intervals, the sound will be badly distorted.

## ❖ Computer clocks and timing events

- The term *clock drift rate* refers to the rate at which a computer clock deviates from a perfect reference clock

## ❖ Two variants of the interaction model

- Synchronous distributed systems
- Asynchronous distributed systems

## ❖ Event ordering

# Latency

- ❖ The delay between the start of a message's transmission from one process and the beginning of its receipt by another is referred to as *latency*. The latency includes:
  - The time taken for the first of a string of bits transmitted through a network to reach its destination. For example, the latency for the transmission of a message through a satellite link is the time for a radio signal to travel to the satellite and back.
  - The delay in accessing the network, which increases significantly when the network is heavily loaded. For example, for Ethernet transmission the sending station waits for the network to be free of traffic.
  - The time taken by the operating system communication services at both the sending and the receiving processes, which varies according to the current load on the operating systems.

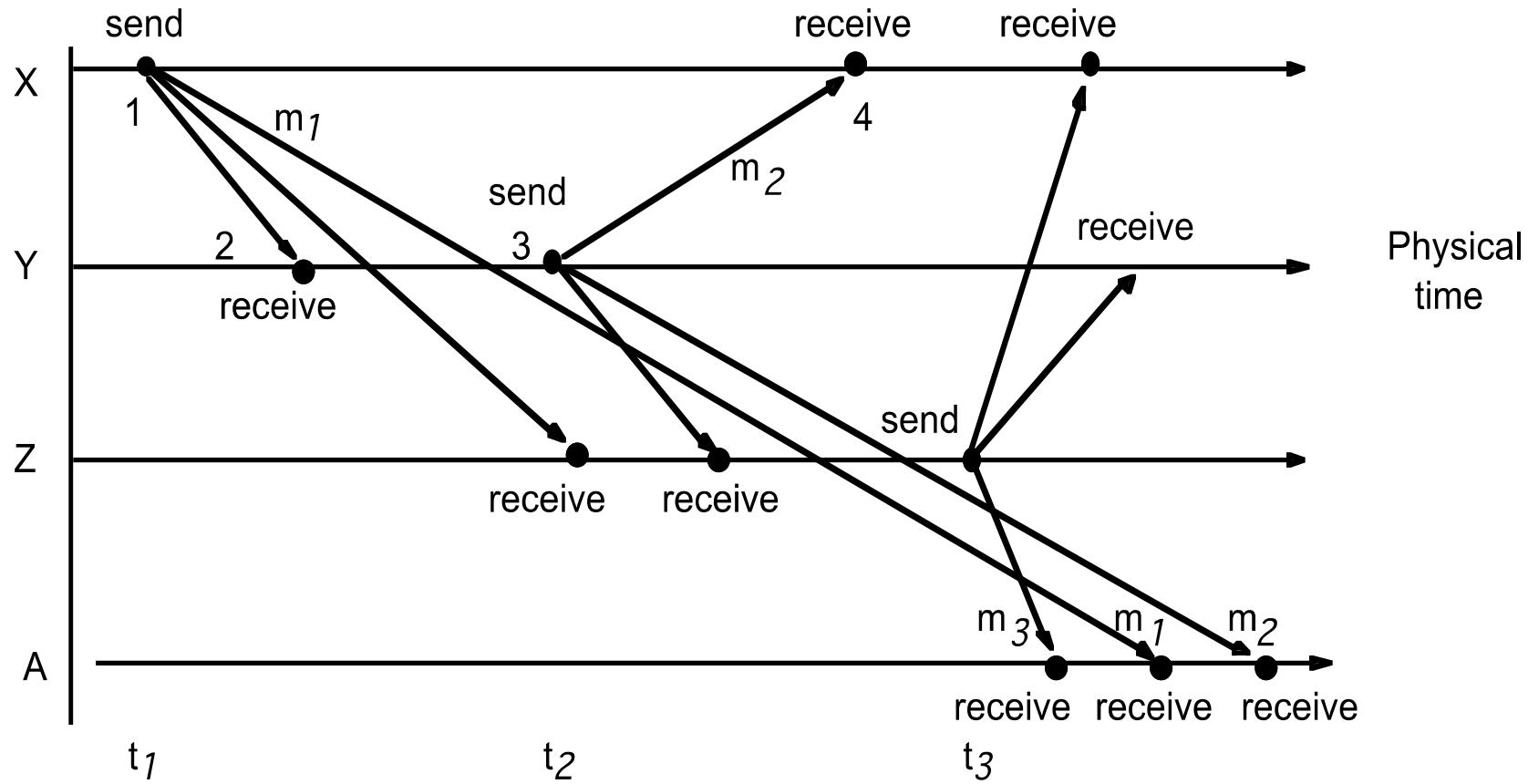
## Two variants of the interaction model

- ❖ *Synchronous distributed systems*: one in which the following bounds are defined:
  - The time to execute each step of a process has known lower and upper bounds.
  - Each message transmitted over a channel is received within a known bounded time.
- ❖ *Asynchronous distributed systems*: Many distributed systems, such as the Internet, are very useful without being able to qualify as synchronous systems. Therefore we need an alternative model. An asynchronous distributed system is one in which there are no bounds on:
  - Process execution speeds – for example, one process step may take only a picosecond and another a century; all that can be said is that each step may take an arbitrarily long time.
  - Message transmission delays – for example, one message from process A to process B may be delivered in negligible time and another may take several years. In other words, a message may be received after an arbitrarily long time.
  - Clock drift rates – again, the drift rate of a clock is arbitrary.

# Event ordering

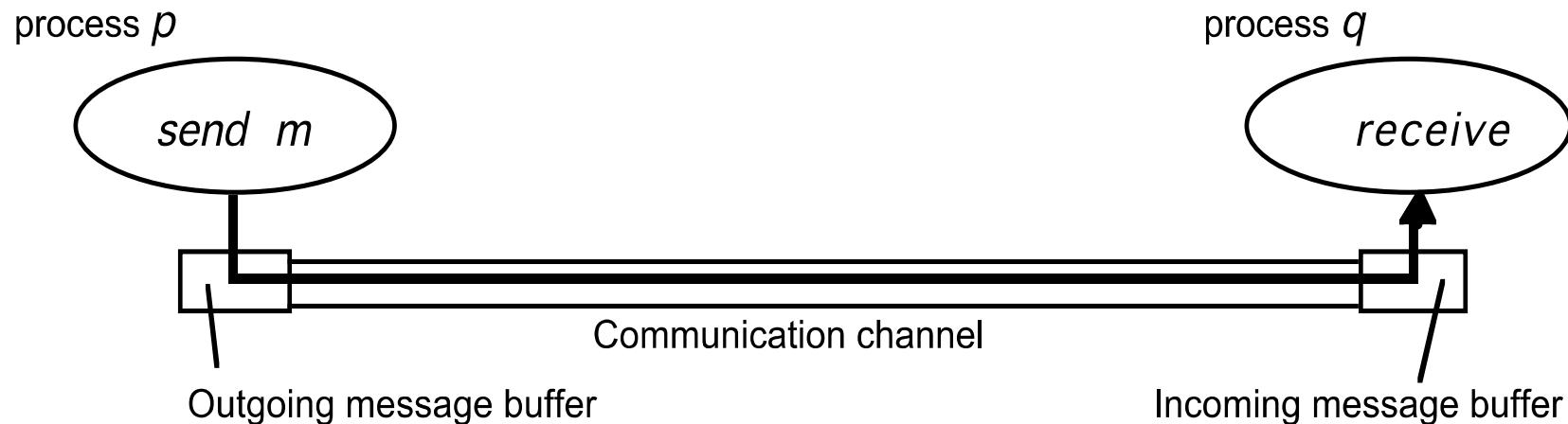
- ❖ In many cases, we are interested in knowing whether an event (sending or receiving a message) at one process occurred before, after or concurrently with another event at another process. The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks
  - User X sends a message with the subject *Meeting*.
  - Users Y and Z reply by sending a message with the subject *Re: Meeting*.
- ❖ Since clocks cannot be synchronized perfectly across a distributed system, Lamport [1978] proposed a model of *logical time* that can be used to provide an ordering among the events at processes running in different computers in a distributed system.
  - X sends  $m_1$  before Y receives  $m_1$ ; Y sends  $m_2$  before X receives  $m_2$ .
  - Y receives  $m_1$  before sending  $m_2$

# Real time ordering



# Failure model

- ❖ **Omission failures** - cases when a process or communication channel fails to perform actions that it is supposed to do
- ❖ Process omission failures – crash and fail-stop
- ❖ Communication omission failures



## Failure model 2

- ❖ **Arbitrary failures** or *Byzantine* failure is used to describe the worst possible failure semantics, in which any type of error may occur.
  - a process may set wrong values in its data items, or
  - it may return a wrong value in response to an invocation.
- ❖ Arbitrary failure of a process
  - one in which it arbitrarily omits intended processing steps or takes unintended processing steps
- ❖ Arbitrary failures of communication channels
  - Message contents may be corrupted,
  - nonexistent messages may be delivered or
  - real messages may be delivered more than once.

# Omission and arbitrary failures

<i>Class of failure</i>	<i>Affects</i>	<i>Description</i>
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a <i>send</i> , but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

## ❖ Timing failures

<i>Class of Failure</i>	<i>Affects</i>	<i>Description</i>
Clock	Process	Process' s local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message' s transmission takes longer than the stated bound.

## ❖ Masking failures

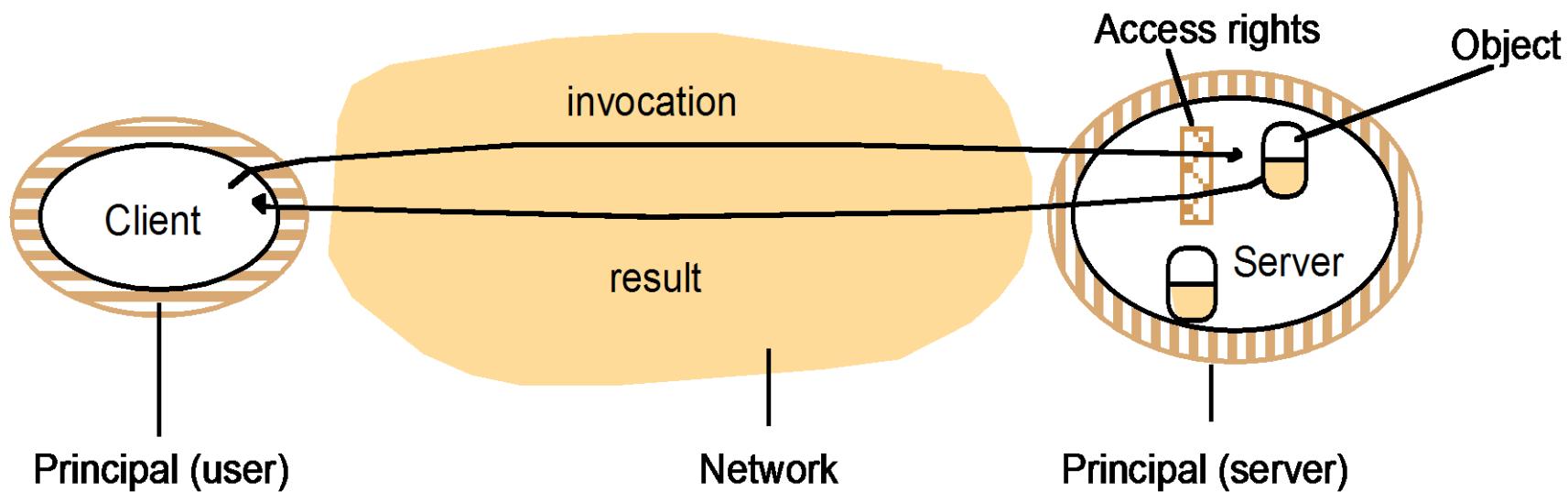
- A service *masks* a failure either by hiding it altogether or by converting it into a more acceptable type of failure.

## ❖ Reliability of one-to-one communication

- *Validity:* Any message in the outgoing message buffer is eventually delivered to the incoming message buffer
- *Integrity:* The message received is identical to one sent, and no messages are delivered twice.

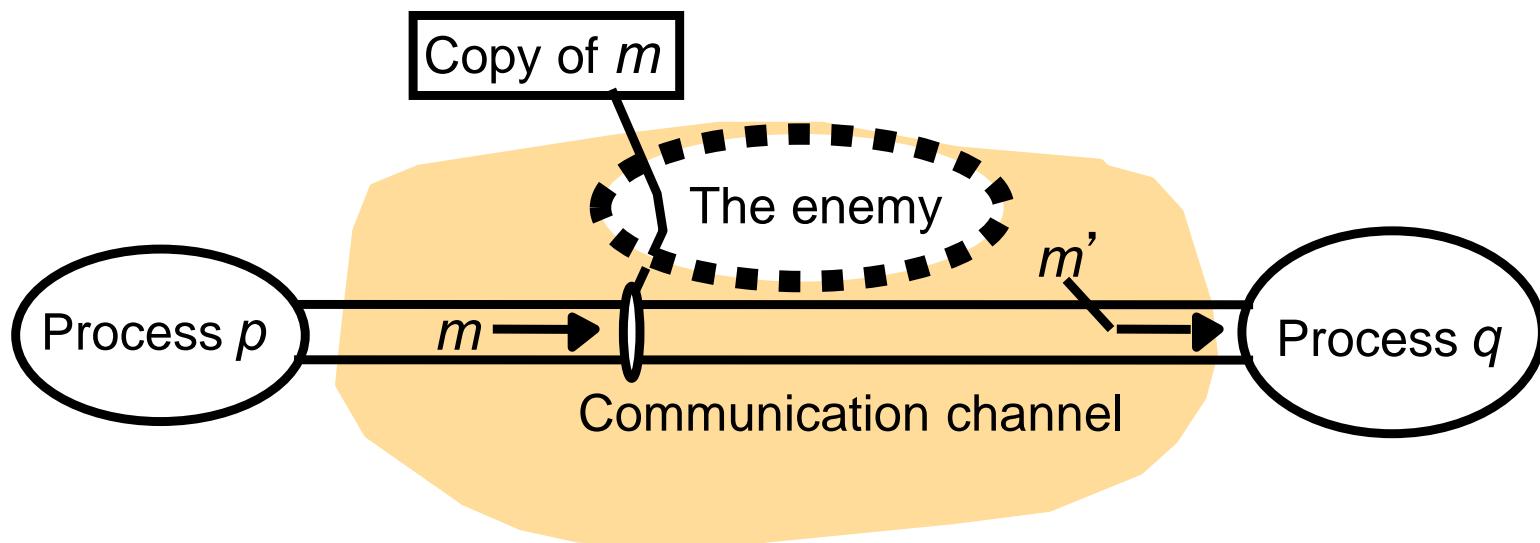
# Security model

- ❖ The security of a distributed system can be achieved by securing the processes and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorized access.
- ❖ **Protecting objects**



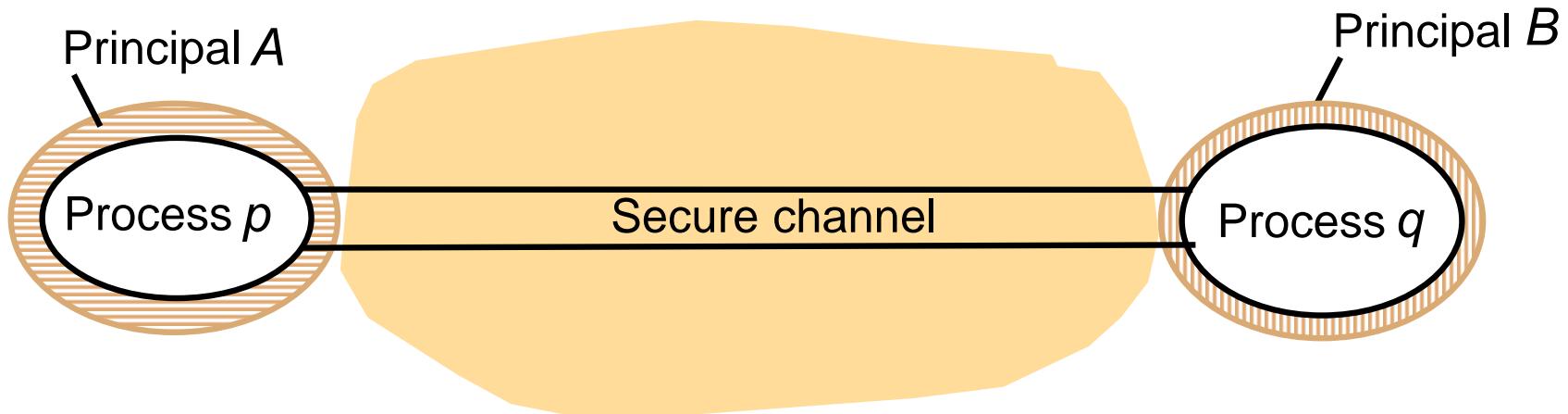
## ❖ Securing processes and their interactions

- **The enemy** - that is capable of sending any message to any process and reading or copying any message sent between a pair of processes
  - ✓ Threats to processes - A process that is designed to handle incoming requests may receive a message from any other process in the distributed system, and it cannot necessarily determine the identity of the sender
  - ✓ Threats to communication channels - An enemy can copy, alter or inject messages as they travel across the network and its intervening gateways



## ❖ Defeating security threats

- Cryptography and shared secrets
- Authentication:
- Secure channels
  - ✓ Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing
  - ✓ A secure channel ensures the privacy and integrity (protection against tampering) of the data transmitted across it.
  - ✓ Each message includes a physical or logical timestamp to prevent messages from being replayed or reordered







# Interprocess communication

Lecture 3

# Outline

- ❖ Introduction
- ❖ External data representation and marshaling
- ❖ Multicast communication
- ❖ Network virtualization

# Introduction

This chapter

Middleware layers

Applications, services

Remote invocation, indirect communication

Underlying interprocess communication primitives:  
Sockets, message passing, multicast support, overlay networks

UDP and TCP

# The characteristics of interprocess communication

## ❖ Synchronous and asynchronous communication

- Synchronous
  - ✓ sending and receiving processes synchronize at every message
  - ✓ both *send* and *receive* are *blocking* operations
- Asynchronous
  - ✓ the *send* operation is *nonblocking*
  - ✓ *receive* operation can have blocking and non-blocking variants

## ❖ Message destinations

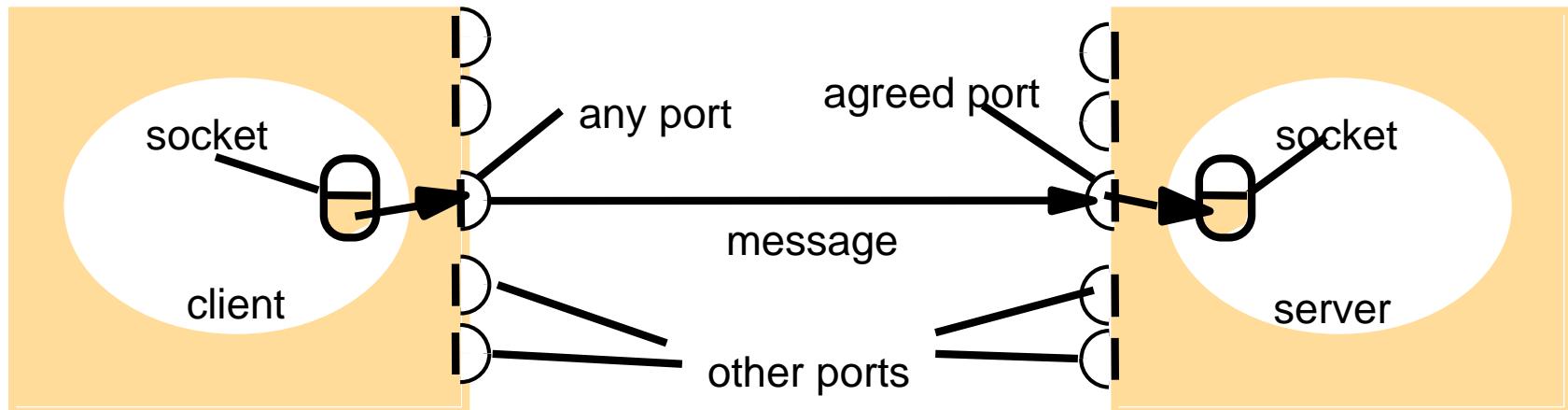
- (*Internet address, local port*)
- Client programs refer to services by name and use a name server or binder to translate their names into server locations at runtime.

## ❖ Reliability

- Validity - point-to-point message service can be described as reliable if messages are guaranteed to be delivered despite a ‘reasonable’ number of packets being dropped or lost
- Integrity – messages must arrive uncorrupted and without duplication

## ❖ Ordering

# Sockets



# External data representation

- ❖ Integer representations
  - *big-endian* order
  - *little-endian* order
- ❖ Characters representations
  - ASCII character coding (1 byte per character)
  - Unicode standard (2 bytes per character)
- ❖ Methods for two computers to exchange binary data values
  - The values are converted to an agreed external format before transmission and converted to the local form on receipt
  - The values are transmitted in the sender's format, together with an indication of the format used, and the recipient converts the values if necessary
- ❖ *External data representation* - an agreed standard for the representation of data structures and primitive values
- ❖ *Marshalling* is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message.
- ❖ *Unmarshalling* is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination

- ❖ Three alternative approaches to external data representation and marshalling:
- ❖ **CORBA's common data representation,**
  - external representation for the structured and primitive types that can be passed as the arguments and results of remote method invocations in CORBA. It can be used by a variety of programming languages.
- ❖ **Java's object serialization,**
  - the flattening and external data representation of any single object or tree of objects that may need to be transmitted in a message or stored on a disk. It is for use only by Java.
- ❖ **XML (Extensible Markup Language),**
  - defines a textual format for representing structured data. It was originally intended for documents containing textual self-describing structured data – for example documents accessible on the Web – but it is now also used to represent the data sent in messages exchanged by clients and servers in web services.

# CORBA's Common Data Representation (CDR)

## ❖ primitive types

- *short* (16-bit), *long* (32-bit), *unsigned short*, *unsigned long*, *float* (32-bit), *double* (64-bit), *char*, *boolean* (TRUE, FALSE), *octet* (8-bit), and *any* (which can represent any basic or constructed type);

## ❖ composite types

<i>Type</i>	<i>Representation</i>
<i>sequence</i>	length (unsigned long) followed by elements in order
<i>string</i>	length (unsigned long) followed by characters in order (can also can have wide characters)
<i>array</i>	array elements in order (no length specified because it is fixed)
<i>struct</i>	in the order of declaration of the components
<i>enumerated</i>	unsigned long (the values are specified by the order declared)
<i>union</i>	.type tag followed by the selected member

# CORBA CDR message

<i>index in sequence of bytes</i>	<i>4 bytes</i>	<i>notes on representation</i>
0–3	5	<i>length of string</i> <i>'Smith'</i>
4–7	"Smit"	
8–11	"h "	
12–15	6	<i>length of string</i> <i>'London'</i>
16–19	"Lond"	
20–23	"on "	
24–27	1984	<i>unsigned long</i>

The flattened form represents a *Person* struct with value: { ‘Smith’ , ‘London’ , 1984}

# Marshaling in CORBA

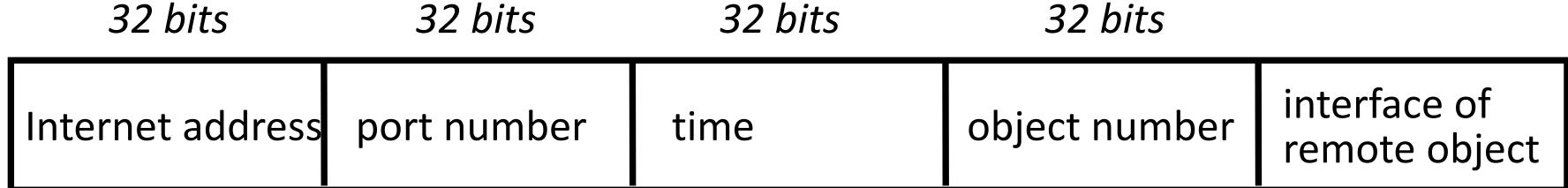
- ❖ Marshalling operations can be generated automatically from the specification of the types of data items to be transmitted in a message
- ❖ The types of the data structures and the types of the basic data items are described in CORBA IDL

```
struct Person{  
    string name;  
    string place;  
    unsigned long year;  
};
```

- ❖ The CORBA interface compiler generates appropriate marshalling and unmarshalling operations for the arguments and results of remote methods from the definitions of the types of their parameters and results.

# Remote object references

- ❖ A **remote object reference** is an identifier for a remote object that is valid throughout a distributed system
- ❖ Remote object references must be generated in a manner that ensures **uniqueness over space and time**
  - concatenate the Internet address of its host computer and the port number of the process that created it with the time of its creation and a local object number. The local object number is incremented each time an object is created in that process.



# Multicast communication

- ❖ Multicast operation sends a single message from one process to each of the members of a group of processes, usually in such a way that the membership of the group is transparent to the sender
- ❖ The simplest multicast protocol provides no guarantees about message delivery or ordering

- ❖ Multicast messages provide a useful infrastructure for constructing distributed systems with the following characteristics:
  - Fault tolerance based on replicated services
  - Discovering services in spontaneous networking
  - Better performance through replicated data
  - Propagation of event notifications

# IP multicast

- ❖ IP multicast is built on top of the Internet Protocol (IP).
- ❖ IP multicast allows the sender to transmit a single IP packet to a set of computers that form a multicast group. The sender is unaware of the identities of the individual recipients and of the size of the group.
- ❖ The membership of multicast groups is dynamic, allowing computers to join or leave at any time and to join an arbitrary number of groups. It is possible to send datagrams to a multicast group without being a member.
- ❖ Details are specific to IPv4
  - Multicast routers
    - ✓ time to live (TTL)
  - Multicast address allocation
    - ✓ Permanent and temporary multicast addresses

# Reliability and ordering of multicast

- ❖ A datagram sent from one multicast router to another may be lost
- ❖ Any one of recipients may drop the message because its buffer is full
- ❖ If a multicast router fails, the group members beyond that router will not receive the multicast message
- ❖ IP packets sent over an internetwork do not necessarily arrive in the order in which they were sent, with the possible effect that some group members receive datagrams from a single sender in a different order from other group members
- ❖ Messages sent by two different processes will not necessarily arrive in the same order at all the members of the group

# Examples of the effects of reliability and ordering

## ❖ Fault tolerance based on replicated services

- requires that either all of the replicas or none of them should receive each request to perform an operation
- require that all members receive request messages in the same order as one another.

## ❖ Discovering services in spontaneous networking

- multicast requests at periodic intervals, and for the available services to listen for those multicasts and respond.
- An occasional lost request is not an issue when discovering services

## ❖ Better performance through replicated data

- The replicated data itself, rather than operations on the data, are distributed by means of multicast messages.
- The effect of lost messages and inconsistent ordering would depend on the method of replication and the importance of all replicas being totally up-to-date

## ❖ Propagation of event notifications

# Network virtualization

- ❖ Network virtualization is concerned with the construction of many different virtual networks over an existing network such as the Internet. Each virtual network can be designed to support a particular distributed application.
- ❖ Each virtual network has its own particular addressing scheme, protocols and routing algorithms, but redefined to meet the needs of particular application classes.

# Overlay networks

- ❖ An *overlay network* is a virtual network consisting of nodes and virtual links, which sits on top of an underlying network (such as an IP network) and offers something that is not otherwise provided:
  - a service that is tailored towards the needs of a class of application or a particular higher-level service – for example, multimedia content distribution;
  - more efficient operation in a given networked environment – for example routing in an ad hoc network;
  - an additional feature – for example, multicast or secure communication.

# Overlay networks advantages

- ❖ They enable new network services to be defined without requiring changes to the underlying network, a crucial point given the level of standardization in this area and the difficulties of amending underlying router functionality.
- ❖ They encourage experimentation with network services and the customization of services to particular classes of application.
- ❖ Multiple overlays can be defined and can coexist, with the end result being a more open and extensible network architecture.
- ❖ The **disadvantages** are that overlays introduce an extra level of indirection (and hence may incur a performance penalty) and they add to the complexity of network services when compared, for example, to the relatively simple architecture of TCP/IP networks

# Types of overlay

Motivation	Type	Description
<i>Tailored for application needs</i>	Distributed hash tables	One of the most prominent classes of overlay network, offering a service that manages a mapping from keys to values across a potentially large number of nodes in a completely decentralized manner (similar to a standard hash table but in a networked environment).
	Peer-to-peer file sharing	Overlay structures that focus on constructing tailored addressing and routing mechanisms to support the cooperative discovery and use (for example, download) of files.
	Content distribution networks	Overlays that subsume a range of replication, caching and placement strategies to provide improved performance in terms of content delivery to web users; used for web acceleration and to offer the required real-time performance for video streaming [ <a href="http://www.kontiki.com">www.kontiki.com</a> ].

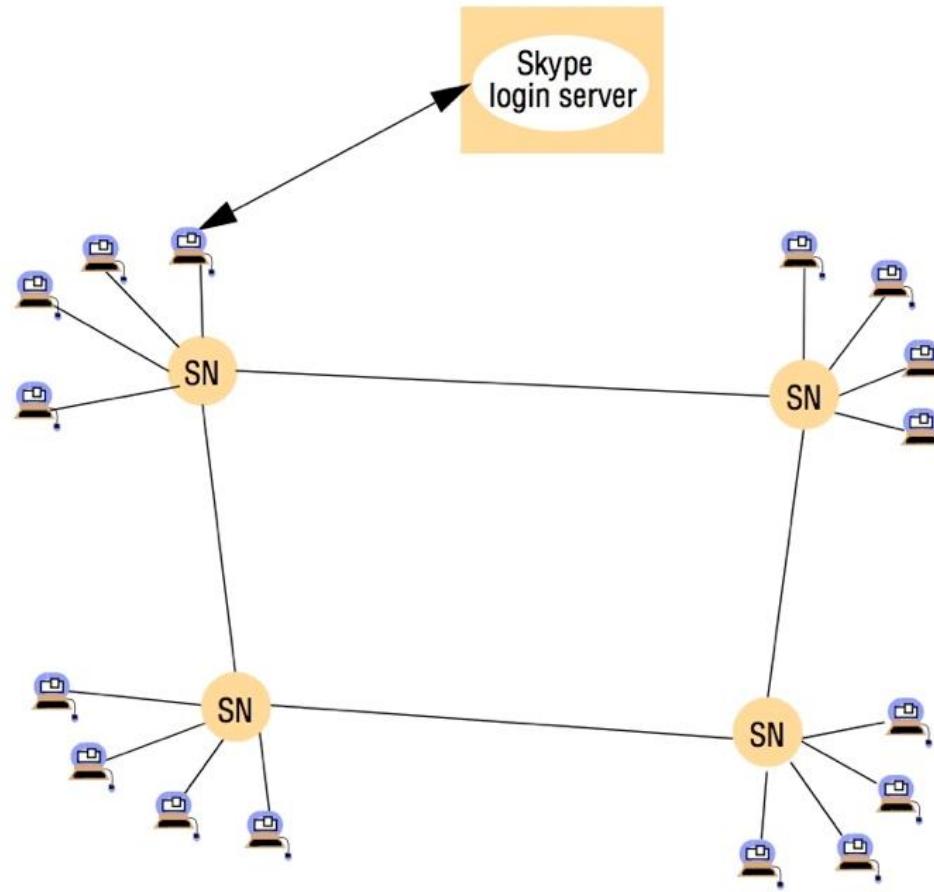
# Types of overlay 2

<i>Tailored for network style</i>	Wireless ad hoc networks	Network overlays that provide customized routing protocols for wireless ad hoc networks, including proactive schemes that effectively construct a routing topology on top of the underlying nodes and reactive schemes that establish routes on demand typically supported by flooding.
	Disruption-tolerant networks	Overlays designed to operate in hostile environments that suffer significant node or link failure and potentially high delays.
<i>Offering additional features</i>	Multicast	One of the earliest uses of overlay networks in the Internet, providing access to multicast services where multicast routers are not available; builds on the work by Van Jacobson, Deering and Casner with their implementation of the MBone (or Multicast Backbone) [ <a href="#">mbone</a> ].
	Resilience	Overlay networks that seek an order of magnitude improvement in robustness and availability of Internet paths [ <a href="#">nms.csail.mit.edu</a> ].
	Security	Overlay networks that offer enhanced security over the underling IP network, including virtual private networks, for example, as discussed in Section 3.4.8.

# Skype: An example of an overlay network

- ❖ Skype is a peer-to-peer application offering
  - Voice over IP (VoIP)
  - instant messaging,
  - video conferencing
  - interfaces to the standard telephony service through SkypeIn and SkypeOut
- ❖ Skype architecture
- ❖ User connection
- ❖ Search for users
- ❖ Voice connection

# Skype overlay architecture



**SN** Super node

 Ordinary host

# Conclusion

## ❖ Alternative styles of marshalling

- CORBA and its predecessors choose to marshal data for use by recipients that have prior knowledge of the types of its components
- when Java serializes data, it includes full information about the types of its contents, allowing the recipient to reconstruct it purely from the content.
- XML includes full type information

## ❖ Multicast messages are used in communication between the members of a group of processes

## ❖ Multicast can also be supported by overlay networks in cases where, for example, IP multicast is not supported.

## ❖ Overlay networks offer a service of virtualization of the network architecture, allowing specialist network services to be created on top of underlying networking infrastructure





## ОТДАЛЕЧЕНО ИЗВИКВАНЕ

доц. д-р Десислава Петрова-Антонова

# Съдържание

- ❖ Въведение
- ❖ Протоколи за заявка-ответ
- ❖ Отдалечно извикване на процедури
- ❖ Отдалечно извикване на методи
- ❖ Case study: Java RMI

# Въведение

Applications

Remote invocation, indirect communication

Underlying interprocess communication primitives: Sockets,  
message passing, multicast support, overlay networks

UDP and TCP

Middleware  
layers

# Комуникация със заявки и отговори

## ❖ Предназначение

- Поддръжка на роли и обмяна на съобщения в типичните клиент-сървър системи

## ❖ Типове комуникация

- Синхронна
- Асинхронна

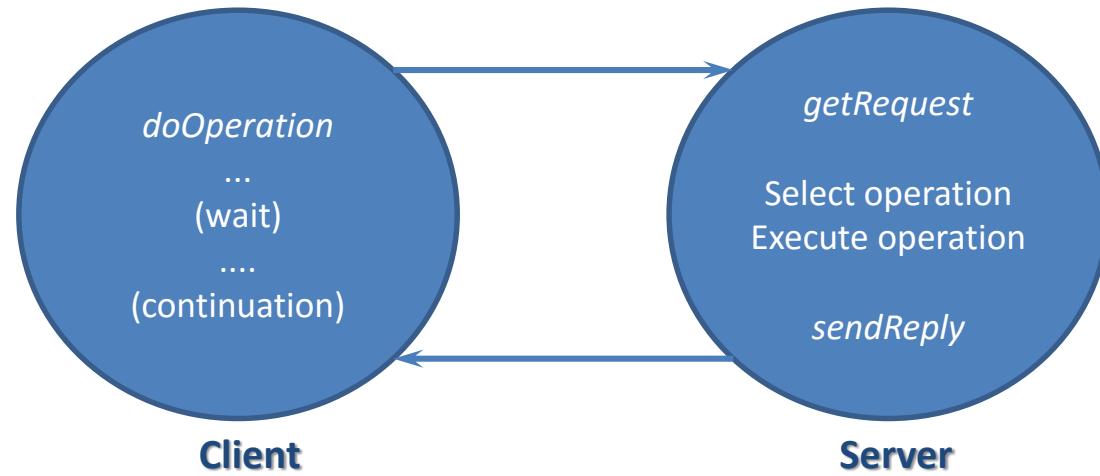
## ❖ Особености на UDP в сравнение с TCP

- Отсъствие на потвърждение
  - ✓ Отговорите се изпращат незабавно след получаването на заявките
- Установяването на връзка включва две допълнителни двойки съобщения
- Липса на необходимост от управление на потока за повечето извиквания
  - ✓ Обменят се малки аргументи и резултати

# Протокол заявка-отговор

## ❖ Примитиви на протокола (Java API с UDP дейтаграми)

- *public byte[] doOperation (RemoteRefs, int operationId, byte[] arguments)*
  - ✓ Предназначение: Извикване на отдалечена операция
  - ✓ Аргументи: референция към отдалечен сървър, операция и набор параметри за извикване на операцията
  - ✓ Резултат: масив от байтове с отговор
- *public byte[] getRequest ()*
  - ✓ Предназначение: Използва се от сървърния процес за получаване на заявка
- *public void sendReply (byte[] reply, InetAddress clientHost, int clientPort)*
  - ✓ Предназначение: изпращане на отговор от сървъра към клиента



# Структура на съобщенията за заявка и отговор

messageType

- int (0 = Request, 1 = Response)

requestId

- int

remoteReference

- RemoteRef

operationId

- int or Operation

arguments

- // array of bytes

# Идентификатори на съобщения и модел за повреди

## ❖ Идентификатор на съобщение

- Целочислена стойност от нарастваща последователност (`unsigned int,  $2^{32} - 1$` )
  - ✓ Прави идентификатора уникален при изпращача
- Идентификатор за изпращащия процес (порт и мрежов адрес)
  - ✓ Прави идентификатора уникален в разпределената система

## ❖ Модел за повреди

- Комуникационни проблеми, засягащи UDP дейтаграмите
  - ✓ Повреди, причинени от загуба на съобщения
  - ✓ Липса на гаранция за получаване на съобщенията в определен ред
- Повреди в процесите

## ❖ Специфициране на интервал за изчакване (`timeout`)

- Изпращане на индикация към клиента за неуспех на операцията `doOperation`
- Многократно изпращане на съобщения-заявки до получаване на отговор или увереност, че липсва отговор от сървъра

# Дублирани заявки, изгубени отговори и история

## ❖ Дублирани съобщения-заявка

- Предотвратяване на изпълнението на заявката повече от веднъж
- Разпознаване на последователните съобщения от един и същ клиент с еднакъв идентификатор

## ❖ Изгубени съобщения-отговор

- Поддръжка на възможност за многократно изпълнение на операция с един и същ ефект
- Съхраняване на резултата от първоначалното изпълнение на операцията в сървъра

## ❖ История

- Използва се за повторно изпращане на отговор към клиента
- Представя се със структура, която съхранява отговорите, които са изпратени от сървъра
  - ✓ Идентификатор на заявка
  - ✓ Съобщение
  - ✓ Идентификатор на клиент, към който е изпратено съобщението
- Проблеми с обема на съхранените съобщения
  - ✓ Съхраняване на последното, изпратено към клиента съобщение
  - ✓ Премахване на съобщения след изтичане на определен период от време

# Стилове на протоколите за комуникация

Name	Message send by		
	Client	Server	Client
R	Request		
RR	Request	Reply	
RRA	Request	Reply	Acknowledge reply

## ❖ R протокол

- Използва се, когато клиентът не очаква получаване на отговор
- Реализира се върху UDP дейтаграми

## ❖ RR протокол

- Отговорът от сървъра се тълкува като съобщение за потвърждение от клиента
- Следващата заявка от клиента се тълкува като съобщение за потвърждение от сървъра

## ❖ RRA протокол

- Използва съобщение за потвърждение
  - ✓ Съдържа идентификатор на заявката *requestId* от съобщението отговор
  - ✓ Потвърждава всички отговори с по-малък идентификатор *requestId*
  - ✓ Потвърдението не блокира клиента

# Реализация на протокола заявка-отговор върху TCP/HTTP

## ❖ Преимущества на протокола TCP

- Позволява предаване на аргументи и резултати с произволен размер (ограничение на дейтаграмите до 8 KB)
- Надеждност при доставянето на съобщенията
  - ✓ Отпада необходимостта от повторно изпращане и филтриране на дублирани съобщения, както и съхраняване на история
- Намаляване на работния товар поради липсата на съобщения за потвърждение

## ❖ Особености на протокола HTTP

- Специфицира съобщенията, включени в обмяната на заявки и отговори, методите, аргументите и резултатите, както и правилата за представянето им в съобщения
- Поддържа фиксирано множество от методи (GET, PUT, POST и т.н.)
- Осигурява договаряне на съдържанието
  - ✓ Клиентската заявка може да съдържа данни за начина на представяне на данните от сървъра
- Поддържа автентикация
  - ✓ Достъп до ресурс с потребителско име и парола

# Клиент сървър взаимодействие върху HTTP

## ❖ Последователност на взаимодействието

- Установяване на връзка в подразбиращ се или специфициран с URL порт на сървъра
- Изпращане на заявка от клиента
- Изпращане на отговор от сървъра
- Заваряне на връзката

## ❖ Използване на персистентни връзки

- Остават отворени при серия от заявки и отговори
- Затварят се от клиента и сървъра посредство изпращане на идикация
  - ✓ Затваряне на връзката от сървъра при липса на заявки от клиента за определен период
  - ✓ Получаване на индикация за затваряне на връзка от сървъра при изпращане на нова заявка от клиента

## ❖ Маршализиране на съобщенията

- Заявки и отговори: ASCII текст
- Ресурси: Поток от байтове

## ❖ Представяне на данни ресурси в аргументите и резултатите

- Стандартът MIME (Multipurpose Internet Mail Extensions)
  - ✓ Специфицира изпращането на съставни данни от електронна поща
  - ✓ Данните се специфицират с тип и подтип (text/plain, text/html, image/gif, image/jpeg)

# HTTP методи

## ❖ GET

- Заявка за ресурс с определен URL адрес, при който информацията за заявката се съдържа в URL адреса
  - ✓ Отговорът съдържа данни или резултат от изпълнение на програма

## ❖ HEAD

- Заявката е идентична на GET, при което се връща информация за данните, а не самите данни

## ❖ POST

- Заявка за ресурс с определен URL адрес, при който информацията се поставя в тялото ѝ

## ❖ PUT

- Заявка за съхраняване на данни като нов или съществуващ ресурс със специфичен URL адрес като идентификатор

## ❖ DELETE

- Заявка за изтриване на ресурс с определен URL адрес

## ❖ OPTIONS

- Заявка за информация, свързана с комуникационните възможности на даден URL адрес (списък от HTTP методи, които могат да се приложат върху URL адреса)

## ❖ TRACE

- Връщане на заявката от сървъра обратно към клиента с диагностични цели

# Съдържание на HTTP съобщение

1.	method	URL or pathname	HTTP version	headers	message body
	GET	http://moodle.openfmi.net/	HTTP/1.1		
2.	HTTP version	Status code	reason	headers	message body
	HTTP/1.1	200	OK		

## ❖ Съобщение заявка (1)

- Типове заявки
  - ✓ Заявка към прокси (изисква абсолютен URL адрес)
  - ✓ Заявка към сървър, хостващ ресурсите (изисква път до ресурса и специфициране на DNS име в полето Host)
- Съдържание на полетата *headers*
  - ✓ Модификатори на заявката или информация за клиента

## ❖ Съобщение отговор (2)

- Съдържание на полетата *status* и *reason*
  - ✓ Числова стойност и текстов израз, описващи резултата от обработката на заявката

## ❖ Тяло на съобщението

- Съдържа данни асоциирани с URL адреса от заявката
- Притежава собствени заглавни полета, специфициращи данните (дължина, MIME тип, кодиране и др.)

# Отдалечно извикване на процедури (RPC)



## ❖ Интерфейс

- Скрива реализацията на даден модул, предоставяйки информация за процедурите и променливите, които са достъпни чрез него

## ❖ Интерфейс при разпределени системи

- Спецификация на процедури, предлагани от даден сървър, и типовете на техните аргументи
- Пример: файлов сървър

## ❖ Преимущества при използването на интерфейси

- Липса на необходимост от познаване на програмната реализация
- Липса на необходимост от познаване на използваните програмен език и платформа
- Възможност за промяна на реализацията при запазване на интерфейса

# Особености на интерфейсите в разпределена среда

- ❖ Интерфейсът не специфицира директен достъп до променливи
  - Обикновено се използват процедури за четене и запис
- ❖ Механизми за предаване на параметри
  - Не се поддържа предаване на параметър по референция
  - Дефиниране на параметрите
    - ✓ Входни
    - ✓ Изходни
    - ✓ Двупосочни
- ❖ Адреси в процесите
  - Не могат да се подават като входни аргументи или да се връщат като резултат

# Език за дефиниране на интерфейси

- ❖ Интегриране на RPC механизма с определен програмен език
  - Осигуряване на нотация, позволяваща входните и изходните параметри да се съпоставят с типичното използване на параметрите в езика
  - Допуска се при използване на еднакъв език за реализация на частите на разпределеното приложение
- ❖ Interface Definition Language (IDL)
  - Проектиран с цел извикване на процедура, реализирана на един език, от друг език
  - Заложената в езика концепция се прилага и при други технологии
    - ✓ Sun XDR, CORBA IDL, WSDL

```
// In file Person.idl
struct Person {
    string name;
    string place;
    long year;
};

interface PersonList {
    readonly attribute string listname;
    void addPerson(in Person p);
    void getPerson(in string name, out Person p);
    long number();
};
```



# Семантика на RPC извикването (1-2)

## ❖ Реализацията на операцията *doOperation*

- Изпращане на повторна заявка
  - ✓ Определя дали ще се изпраща повторно заявка до получаване на отговор или ще се прави предположение за повреда в сървъра
- Филтриране на дублирани съобщения
  - ✓ Определя дали ще се допуска повторно изпращане на заявка и дали дублираните заявки ще се филтрират на сървъра
- Повторно изпращане на резултат
  - ✓ Определя дали ще се съхранява история на сървъра с цел изпращане на повторен отговор без повторно изпълнение на операция

Fault tolerance measures			Call semantics
Retransmit request	Duplicate filtering	Re-execute procedure or retransmit reply	
No	Not applicable	Not applicable	Maybe
Yes	No	Re-execute procedure	At-least-once
Yes	Yes	Retransmit reply	At-most-once

# Семантика на RPC извикването (2-2)

## ❖ Maybe

- Отдалечената процедура се изпълнява **еднократно или въобще не се изпълнява**
- Опастност от изгубване на съобщения за заявки или отговори
- Опастност от повреди в сървъра
  - ✓ Несигурност в изпълнението на процедурата на сървъра

## ❖ At-least-once

- Клиентът получава резултат, при което знае, че процедурата е **изпълнена поне веднъж**, или получава информация за липса на резултат
- Липса на опастност от изгубване на съобщения
- Опастност от повреда на сървъра
- Опастност от случайни повреди
  - ✓ Многократно изпълнение на операция, предизвикващо некорекна промяна в данните
  - ✓ Пример: операция за запис в банкова сметка

## ❖ At-most-once

- Клиентът получава резултат, при което знае, че процедурата е **изпълнена само веднъж**, или получава информация за липса на резултат, при което процедурата е изпълнена веднъж или изобщо не е изпълнена
- Липса на опастност от изгубване на съобщения
- Липса на случайни повреди

# Прозрачност

## ❖ Определение

- Скриване на физическото местоположение на процедурата и осигуряване на еднотипен достъп до локалните и отдалечените процедури

## ❖ Различия между локалните и отдалечените процедури

- По-голяма уязвимост на повреди
- По-голяма латентност
  - ✓ Необходимост от прекъсване на отдалечената процедура от клиента без да се оказва влияние върху сървъра
- Липса на предаване на параметър като референция
- Препоръки за **различно представянето** на локалните и отдалечените операции в интерфейса или за **използване на различен синтаксис**

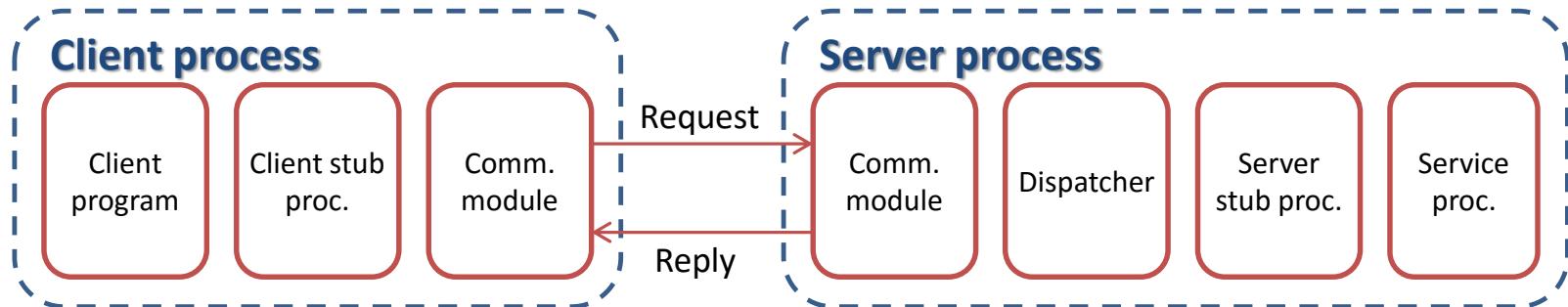
## ❖ Определяне на прозрачността при проектиране на IDL езика

- Генериране на изключения при липса на комуникация
- Специфициране на семантика на извикването

## ❖ Консенсус относно прозрачността

- Идентичен синтаксис, но различно представяне на отдалеченото и локалното извикване в интерфейса

# Реализация на RPC



## ❖ Последователност на изпълнение на отдалечена процедура

- Маршализация на процедурния идентификатор и аргументите в заявката към сървъра
- Избор на стъб процедура от диспечера
- Демаршализация на заявката от стъб процедурата на сървъра
- Извикване на отдалечената процедура
- Маршализация на отговора към клиента

## ❖ Реализация

- Използва се протокол заявка-отговор
- Реализация на семантика от тип at-least-once или at-most-once

Sun RPC

# CASE STUDY

# Характеристики на Sun RPC

- ❖ Приложение при клиент-сървър комуникация в мрежовата файлова система на Sun
- ❖ Доставя се като част от Sun и UNIX системи
- ❖ Реализацията поддържа UDP и TCP
- ❖ Размерът на съобщенията при UDP реализация е ограничен до 64 KB
- ❖ Реализацията поддържа семантика на извикването от вида at-least-once
- ❖ Осигурява се език *XDR* за описание на интерфейси
- ❖ Осигурява се интерфейсен компилатор *rpcgen*, проектиран за работа с езика C

# Особености на езика XDR

- ❖ Не позволява специфициране на имена на интерфейсите
  - Поддържа се номер на програма и номер на версия
    - ✓ Подават се като параметри на заявката
- ❖ Дефиницията на процедурата включва сигнатура и номер
- ❖ Поддържа се един входен параметър
  - При необходимост от няколко входни параметъра се създава структура
- ❖ Изходните параметри се връщат като единичен резултат
- ❖ Сигнатурата на процедурата включва тип на резултата, име на процедурата и тип на входния параметър
  - Параметрите могат да са единични стойности или структурирани данни

# Sun XDR файлов интерфейс

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
    version VERSION {
        void WRITE(writeargs)=1;
        Data READ(readargs)=2;
    }=2;
} = 9999;
```



# Особености при Sun RPC

## ❖ Функционалност на интерфейсния компилатор

- Генерира клиентски стъбл процедури
- Генерира *main* процедура на сървъра, диспечери и стъбл процедури
- Генерира процедури за маршализация и демаршализация

## ❖ Свързване

- Поддръжка на локална услуга за свързване, наречена “port mapper”
  - ✓ Всяка инстанция на услугата съхранява номер на програма, номер на версия и номер на порт
- Изпращане на заявка от клиента към няколко инстанции на услуга, стартирани на различни машини
  - ✓ Използва се “port mapper” услугата

## ❖ Автентикация

- Автентикуращата информация се поставя в заявката
- Сървърната програма е отговорна за контрола върху извикването
- Механизми за автентикация
  - ✓ Липса на протокол
  - ✓ UNIX стил на автентикация (uid и gid)
  - ✓ Използване на споделен ключ за подписване на RPC съобщенията
  - ✓ Kerberos

# Отдалечно извикване на методи (RMI)

## ❖ Същност

- Отдалечно извикване на метод на обект

## ❖ Прилики между RMI и RPC

- Поддръжка на програмиране с интерфейси
- Базиране на протокол заявка-отговор
- Поддръжка на идентично ниво на прозрачност
  - ✓ Еднакъв синтаксис и различно специфициране в интерфейса на локалните и отдалечените извиквания

## ❖ Разлики между RMI и RPC

- Възможност за използване на обектно-ориентираната парадигма при RMI
- Възможност за предаване на параметри като референции към обекти
  - ✓ Намалява се мрежовия трафик

# Обектен модел

## ❖ Референция на обект

- Използват се за достъп до обектите
- Mogат да се присвояват на променливи, предават като параметри, връщани като резултат от методи и т.н.

## ❖ Интерфейс

- Дефинира сигнатура на множество от методи
- Дефинира данни типове
- Не дефинира конструктори

## ❖ Действие

- Инициира се от обект, извикващ метод на друг обект
- Следствия от извикване на метода
  - ✓ Промяна на състоянието на получателя
  - ✓ Инстанциране на нов обект
  - ✓ Последващо извикване на методи на други обекти

## ❖ Изключение

- Осигурява начин за справяне със ситуации на възникване на грешка без да се усложнява кода
- Дефиниране на изключение с конструкция “*throw-catch*”

## ❖ Изчистване на паметта

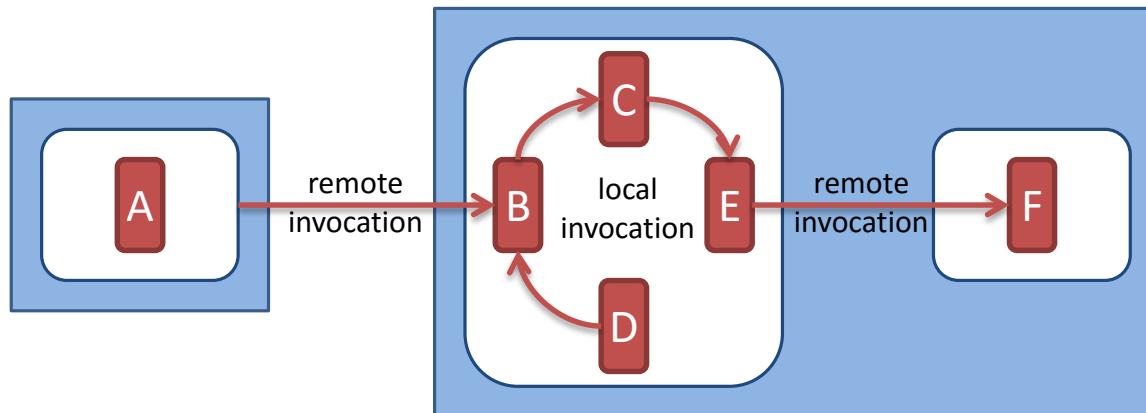
- Премахване на обектите, които не се използват, от паметта

# Отдалечени обекти

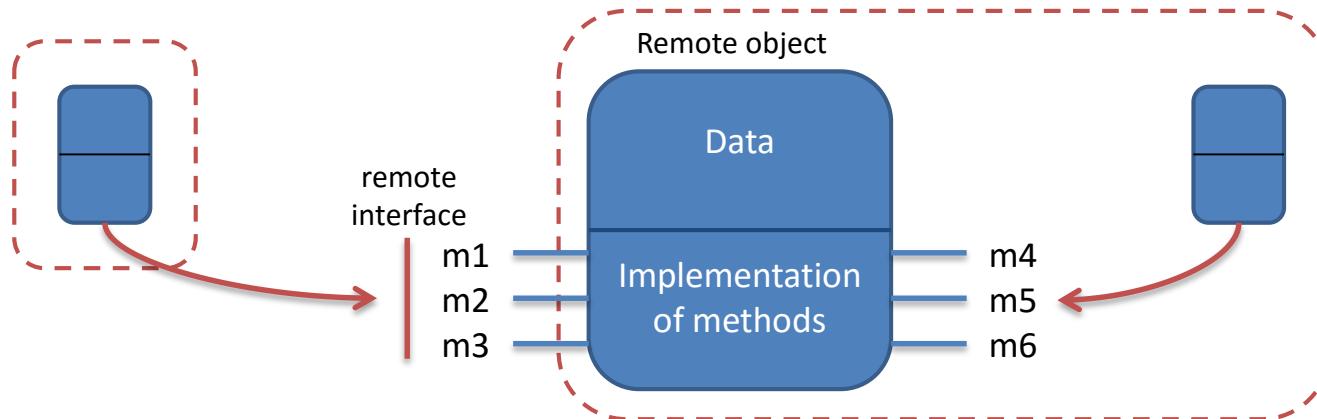
- ❖ Състояние на обект
  - Определя се от стойностите на променливите за определена инстанция
- ❖ Използване на обекти в клиент-сървър системи
  - Отдалечно извикване на метод на обект, управляем от сървър
- ❖ Извършване на репликация на обекти
  - Осигуряване на отказоустойчивост и повишаване на производителността
- ❖ Осигуряване на инкапсуляция
  - Състоянието на обекта е достъпно единствено чрез методите му
  - Разрешаване на конфликти при конкурентен достъп и осигуряване на синхронизиращи примитиви
- ❖ Представяне на състоянието на разпределена програма като колекция от обекти
  - Достъп до обектите по референция
  - Създаване на локални копия на обектите (изиска наличие на класа, реализиращ обекта, при клиента)
- ❖ Възможност за използване на различни данни формати в комуникиращите страни

# RMI обектен модел

- ❖ Отдалеченено извикване на метод
  - Извикване на метод между обекти, намиращи се в различни процеси
- ❖ Отдалечен обект
  - Обект, на който методите могат да се извикват отдалечено (B и F)
- ❖ Референция към отдалечен обект
  - Осигурява механизъм за отдалечено извикване на метод на обект
- ❖ Отдалечен интерфейс
  - Специфицира методи на обект, които могат да се извикват отдалечено



# Референция и интерфейс на отдалечен обект



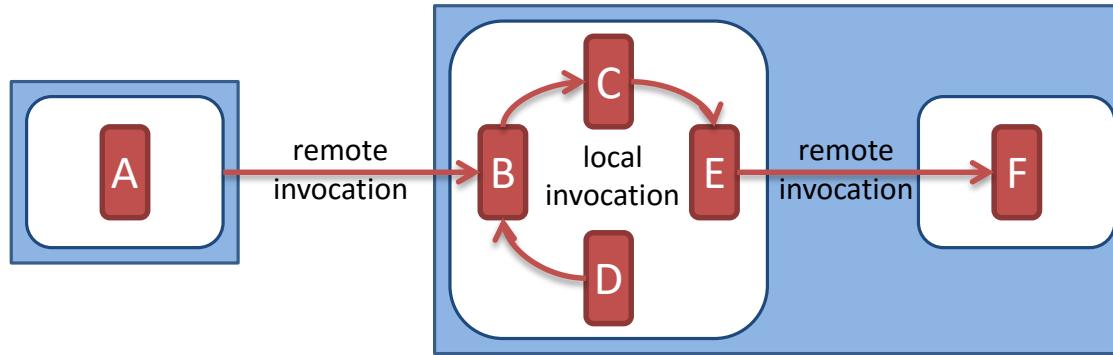
## ❖ Референция към отдалечен обект

- Идентификатор, който се използва при разпределените системи за рефериране на уникален отдалечен обект
  - ✓ Използва се при отдалечно извикване на метод
  - ✓ Може да се предава като аргумент и връща като резултат

## ❖ Отдалечен интерфейс

- Отдалечно могат да се извикват само методите в отдалечения интерфейс
- Локалните обекти могат да извикват методи както в отдалечения интерфейс, така и други методи на отдалечения обект
- Езици за дефиниране на отдалечени интерфейси
  - ✓ CORBA IDL (наличие на IDL компилатори за C++, Java и Python)
  - ✓ Java RMI (обозначаване на интерфейси като Remote)
- Поддръжка на множествено наследяване

# Действия в разпределените обектни системи

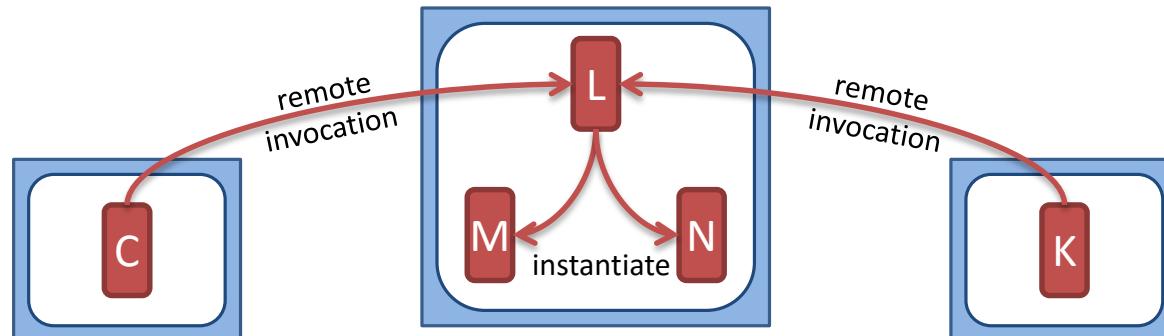


## ❖ Извикване на метод

- Извикването на отдалечен метод изиска наличие на референция към отдалечен обект (A използва отдалечена референция към B)
- Референциите могат да се получават като резултат от извикване на отдалечен метод (A може да получи отдалечена референция към F от B)

## ❖ Инстанциране на нов обект

- Новият обект се притежава от процеса, в който се заявява инстанцията
- Отдалечените обекти могат да реализират методи за инстанциране на обекти



# Изчистване на паметта и изключения

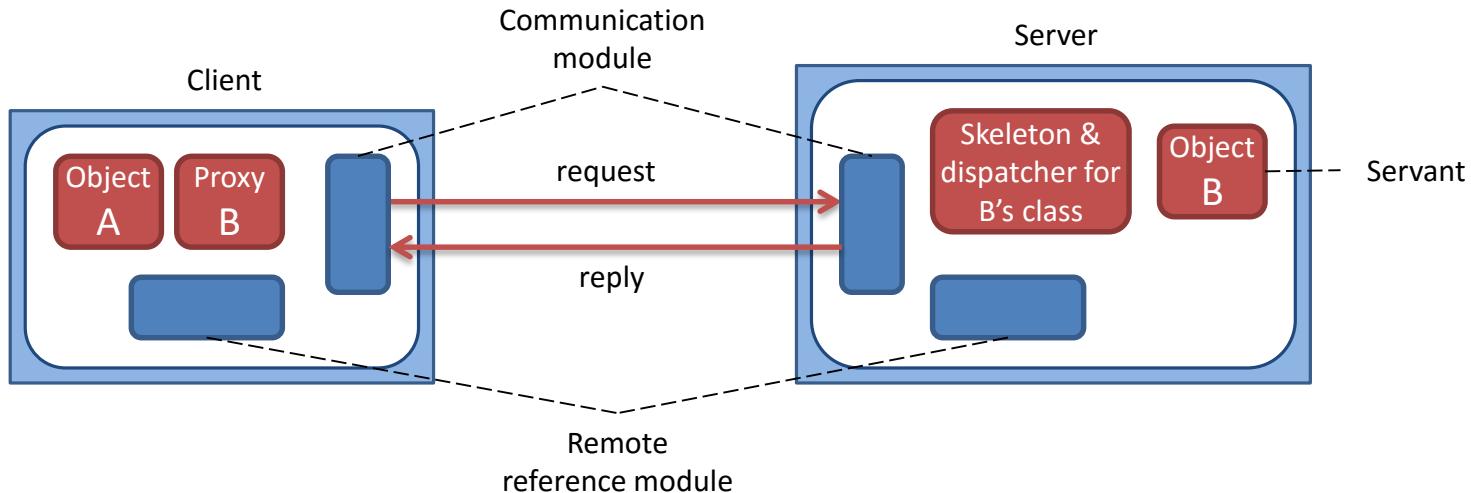
## ❖ Отдалечено изчистване на паметта

- Съчетаване на локалното изчистване на паметта с допълнителен модул, отговорен за отдалеченото изчистване на паметта
- Обикновено се използват броячи на референциите

## ❖ Изключения

- Причини за възникване на повреди
  - ✓ Сриване на отдалечения процес
  - ✓ Блокиране на отдалечения процес
  - ✓ Изгубване на извикването или резултата
- Осигуряване на механизми за предотвратяван на повреди
  - ✓ Използване на времеви интервали за изчакване
  - ✓ Генериране и прихващане на стандартни изключения
  - ✓ Пример: CORBA IDL осигурява нотация за специфициране на изключения на приложно ниво

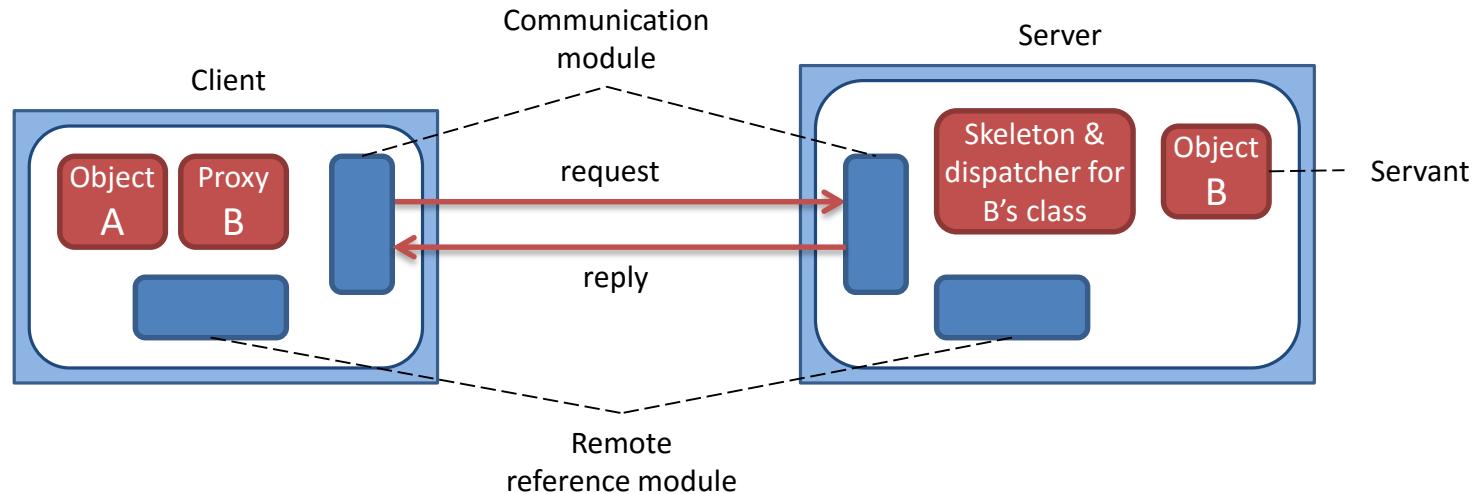
# Реализация на RMI: комуникационен модул



## ❖ Функция на комуникационния модул

- Препраща заявките и отговорите между клиента и сървъра
- Използва информация за съобщението
  - ✓ тип на съобщението
  - ✓ идентификатора на заявка requestId
  - ✓ отдалечената референция към извиквания обект
- Осигурява семантиката на извикването (*at-most-once*)
- В сървърната страна комуникира с диспечера за класа на извиквания обект
  - ✓ Подава локална референция на диспечера, получена от модула за отдалечени референции

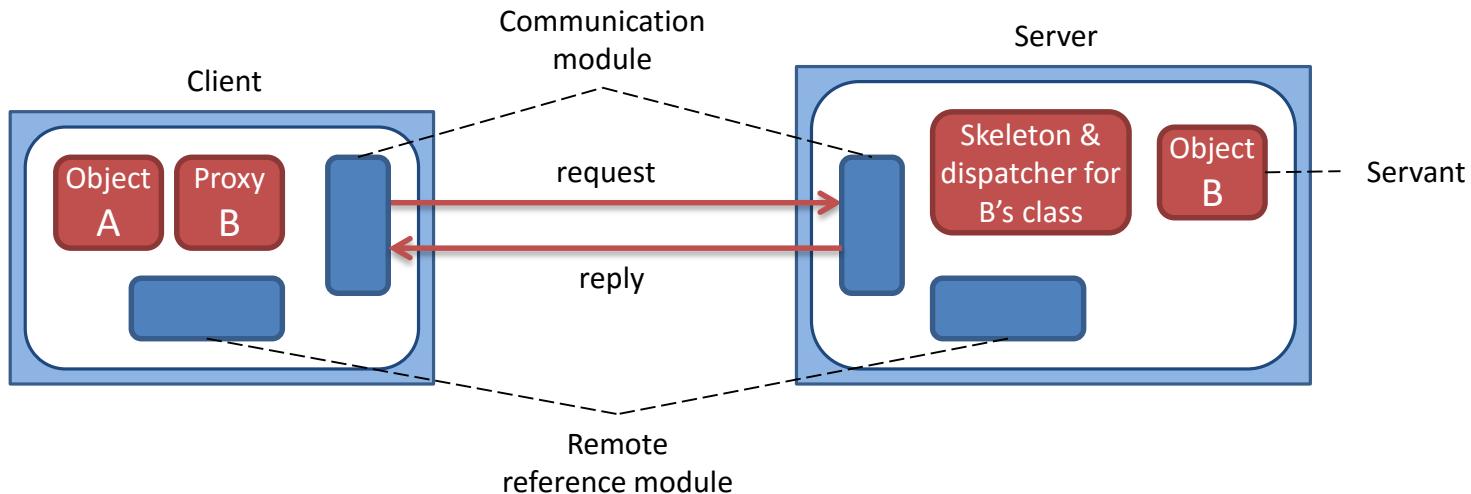
# Реализация на RMI: модул за отдалечени референции



## ❖ Функция на модула за отдалечени референции

- Съпоставя локалните и отдалечените референции на обекти (2)
- Създава отдалечени референции на обекти (1)
- Използва таблица за отдалечени обекти
  - ✓ Съхранява записи за всички отдалечени обекти в процеса (Object B)
  - ✓ Съхранява записи за локалните проксита (Proxy B)
- Случаи на употреба
  - ✓ Подаване на отдалечен обект като аргумент или резултат за пръв път (1)
  - ✓ Връщане на локална референция към обект при получаване на отдалечена такава (2)
- Извиква се от компонентите на RMI софтуера при маршализацията и демаршализацията на отдалечените референции към обекти

# Реализация на RMI: Servant



## ❖ Особености на сърванта

- Инстанция на клас, който реализира отдалечен обект
- Обработва отдалечените заявки, които получава от скелетона
- Създава се при инстанцирането на отдалечен обект
- Премахва се от системата за изчистване на паметта, когато отдалечения обект не се използва

# Реализация на RMI: комуникационни обекти

## ❖ Прокси

- Осигурява прозрачност на извикването при клиента
- Препраща заявката за извикване на метод към отдалечения обект
- Реализира методите в интерфейса на отдалечения обект
  - ✓ Методът маршилиза референция към целевия обект, идентификатора на операцията *operationID* и аргументите ѝ в заявка и я изпраща на отдалечения обект
  - ✓ Методът изчаква за отговор, демаршилиза резултата и го връща на клиента

## ❖ Диспечер

- Получава заявките от комуникационния модул
- Използва идентификатора на заявлената операция *operationID*, за да избере подходящ метод в скелетона, към който да препрати заявката

## ❖ Скелетон

- Реализира методите в отдалечения интерфейс на отдалечения обект
  - ✓ Методът демаршилиза аргументите на заявката и извика необходимия метод в сърванта
  - ✓ Методът изчаква завършването на операцията, маршилиза резултата в отговор, който се изпраща на клиента

## ❖ Генериране на класове за проксита, диспечери и скелетони

- Извършва се автоматично от интерфейсен компилатор

# Динамично извикване: прокси алтернатива

## ❖ Същност на динамичното извикване

- Свързването с отдалечения обект не е по време на компилация
- Осигурява достъп на клиента до общото представяне на отдалеченото извикване, определено от операцията *doOperation*
  - ✓ Клиентът осигурява за операцията *doOperation* **отдалечена референция, име на метод и аргументи на операцията**
- Информацията за извикването на отдалечения обект се получава от т. нар. хранилища на интерфейси
  - ✓ Interface Repository при CORBA
- Използва се, когато интерфейсите на обектите не са известни в дизайн режим

## ❖ Динамични скелетони

- Сървърът хоства обекти, чиито интерфейси не са известни по време на компилиране
- Реализации
  - ✓ Създаване на динамични скелетони при CORBA
  - ✓ Използване на общ диспечер и динамично сваляне на класове в сървъра при Java RMI

# Сървърна и клиентска програми

## ❖ Сървърната програма

- Съдържа класове на диспечери и скелетони
- Реализацията на сървантните класове
- Притежава инициализираща секция
  - ✓ Създава и инициализира най-малко един сървант, хостван от сървъра
  - ✓ Възможно е да регистрира сърванти в свързващата услуга (binder)

## ❖ Клиентска програма

- Съдържа класове на проксита
- Използва свързващата услуга при търсенето на отдалечени обекти

## ❖ “Фабрични методи”

- Използват се за създаване на сърванти, тъй като интерфейсите на отдалечените обекти не съдържат конструктори
- Реализират се от т. нар. “фабрични обекти”

# Свързваща услуга, сървърна нишка и активатор

## ❖ Свързваща услуга

- Използва се от клиентските програми за **получаване на референции** към отдалечени обекти и от сървърните програми за **регистриране** на отдалечени обекти
- Управлява таблица, съпоставяща текстови имена на референциите към отдалечени обекти

## ❖ Сървърна нишка

- Всяко извикване на отдалечен обект води до създаване на отделна нишка за него от сървъра
  - ✓ Предотвратява забавянето на изпълнението на един обект от друг обект

## ❖ Активация на отдалечен обект

- Активатор
  - ✓ Процес (услуга), който стартира сървърни процеси с цел хостване на отдалечени обекти
- Активен обект
  - ✓ Достъпен за извикване в рамките на стартиран сървърен процес
- Пасивен обект
  - ✓ Съдържа реализация на методите си и пази състоянието си в маршализирана форма
- Отговорности на активатора
  - ✓ Регистрира пасивни обекти, които са готови за активация
  - ✓ Стартира именувани сървърни процеси и активира отдалечени обекти в тях
  - ✓ Съхранява местоположенията на сървърите за отдалечените обекти, които вече е активирал

# Хранилища на персистентни обекти

## ❖ Персистентен обект

- Обект, който продължава да съществува между активациите на процесите

## ❖ Хранилище за персистентни обекти

- Съхранява персистентни обекти на диск или в база от данни
- Осигурява прозрачност на активацията от гледна точка на клиента
- Използва стратегия, за да определи кои обекти трябва да станат пасивни
  - ✓ Заявка в програмата, която активира обекта
  - ✓ Приключване на транзакция
  - ✓ Изход от програма
- Осигурява колекция от свързани персистентни обекти да притежават разбираеми за човека имена

# Местоположение на обект

## ❖ Определяне на местоположението от информацията в отдалечената референция на обекта

- Използва се, когато обекта се намира в един и същ процес през целия си жизнен цикъл
- Необходимост от използване на адрес на извикването

## ❖ Услуга за определяне на местоположението на обект

- Използва база от данни, която свързва отдалечената референция към обект с действителното му местоположение
- Използване на механизъм “cash/broadcast”
  - ✓ Съпоставянето на референции и местоположения на обектите в кеш памет на всеки компютър
  - ✓ Адресите от кеша се използват за извикване на отдалечените обекти
  - ✓ За определяне на метаположенията на обекти, които са преместени или не са в кеша се извършва разпространяване на заявката по мрежата

# Разпределено изчистване на паметта (1-2)

- ❖ Java алгоритъм за разпределено изчистване на паметта
  - Сървърният процес управлява множество от имена на процеси, които притежават отдалечени референции към обекти
    - ✓  $B.holders$ : множество от клиентски процеси, които имат проксита за обекта  $B$
  - Когато клиент  $C$  получи отдалечена референция към обект  $B$ , той извиква операция  $addRef(B)$  на сървъра и създава прокси
    - ✓ Сървърът добавя клиента  $C$  към множеството  $B.holders$
  - Когато системата за изчистване на паметта при клиента  $C$  установи, че проксито за обекта  $B$  не се достъпва, се извиква операция  $removeRef(B)$  и проксито се премахва
    - ✓ Сървърът премахва клиента  $C$  от множеството  $B.holders$
  - Когато множеството  $B.holders$  е празно, то системата за изчистване на паметта при сървъра премахва обекта  $B$

## Разпределено изчистване на паметта (2-2)

- ❖ Проблеми при изчистване на множеството с референции за даден отдалечен обект
  - Клиент А извиква *removeRef(B)* преди клиент С да извика *addRef(B)*
  - Обектът *B* се премахва преди да се добави референция към него към множеството *B.holders*
  - Към множеството *B.holders* се добавя временен запис, докато не пристигне *addRef(B)*
- ❖ Проблеми при възникване на изключение в *addRef(B)*
  - Клиентът не създава прокси и извиква *removeRef(B)*
- ❖ Проблеми при възникване на изключение в *removeRef(B)*
  - Сървърът използва политика за отдаване на обекти под наем за определен период от време
  - Клиентите са отговорни за подновяване на наема след изтичане на периода от време, за който обектът е нает

Java RMI

# CASE STUDY

# Приложението бяла дъска

- ❖ Споделяне на общ изглед на графични обекти в група от потребители
- ❖ Функционалност на приложението
  - Съхраняване на всички изрисувани форми
  - Възможност клиентите да информират сървъра за изрисувани форми
  - Възможност клиентите да получават информация за последно изрисуваните форми
  - Генериране на номер на версия при добавяне на нова форма и асоцииране с формата
  - ...



# Отдалечени интерфейси Shape и ShapeList

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject getAllState() throws RemoteException;
}
```

Отдалечен  
обект

```
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException; 2
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```

Локален  
обект



# Правила за сериализация

- ❖ Сериализиране на обект, реализиращ *Remote* интерфейс
  - Обектът се заменя с отдалечената му референция, съдържаща името на класа му
- ❖ Сериализиране на произволен обект
  - Информацията за класа се анотира с местоположението му (URL), което позволява да бъде свален
- ❖ Сваляне на класове от една виртуална машина на друга
  - Получателят не притежава клас на обект, подаден по стойност
  - Получателят не притежава клас на прокси
- ❖ Преимущества
  - Липса на необходимост от поддържане на едно и също множество от класове при всички потребители
  - Прозрачност при използване на инстанции на нови класове от клиентските и сървърните програми

# Класът Naming в RMIregistry

- ❖ *void rebind (String name, Remote obj)*
  - Използва се от сървъра за регистриране на идентификатор на отдалечен обект по име
- ❖ *void bind (String name, Remote obj)*
  - Използва се от сървъра за регистриране на отдалечен обект по име, но ако името е заето се генерира изключение
- ❖ *void unbind (String name, Remote obj)*
  - Премахва регистрацията на отдалечен обект
- ❖ *Remote lookup(String name)*
  - Използва се за търсене на отдалечен обект по име от клиента
- ❖ *String [] list()*
  - Връща масив от низове с имената на обектите в регистъра

# Класът ShapeListServer

```
import java.rmi.*;
public class ShapeListServer{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        try{
            ShapeList aShapeList = new ShapeListServant(); 1
            ShapeList stub =
                (ShapeList )UnicastRemoteObject.exportObject(aShapeList, 0); 2
            Naming.rebind("Shape List", aShapeList ); 3
            System.out.println("ShapeList server ready");
        }catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());
        }
    }
}
```



# Класът ShapeListServant

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class ShapeListServant extends UnicastRemoteObject implements ShapeList {
    private Vector theList; // contains the list of Shapes
    private int version;
    public ShapeListServant() throws RemoteException{...}
    public Shape newShape(GraphicalObject g) throws RemoteException { 1
        version++;
        Shape s = new ShapeServant( g, version); 2
        theList.addElement(s);
        return s;
    }
    public Vector allShapes() throws RemoteException{...}
    public int getVersion() throws RemoteException { ... }
}
```



# Клиент ShapeList

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        ShapeList aShapeList = null;
        try{
            aShapeList = (ShapeList) Naming.lookup("//bruno.ShapeList"); 1
            Vector sList = aShapeList.allShapes(); 2
        } catch(RemoteException e) {System.out.println(e.getMessage());}
        }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
    }
}
```







## ИНДИРЕКТНА КОМУНИКАЦИЯ

доц. д-р Десислава Петрова-Антонова

# Съдържание

- ❖ Въведение
- ❖ Групова комуникация
- ❖ “Publish-subscribe” системи
- ❖ Опашки със съобщения
- ❖ Подходи за споделяне на памет

*“All problems in computer science can be solved by another level of indirection.”*



## Titan Project at the University of Cambridge

Rodger Needham, Maurice Wilkes and David Wheeler

# Въведение

## ❖ Индиректна комуникация: определение

- Комуникация в разпределените системи посредством посредници, при което липсва директна връзка между изпращаца и получателя/получателите

## ❖ Предимства на индиректната комуникация

- Липса на свързаност в пространството
  - ✓ Изпращацът не е необходимо да идентифицира получателя и обратно
- Липса на свързаност във времето
  - ✓ Изпращацът и получателят са независими във времето

## ❖ Недостатъци на индиректната комуникация

- Намаляване на производителността поради наличие допълнително комуникационно ниво
  - ✓ Jim Gray: "*There is no performance problem that cannot be solved by eliminating a level of indirection.*"
- По-големи затруднения при управлението на комуникацията

# Свързаност във времето и пространството

Свързаност	Свързаност във времето	Несвързаност във времето
Свързаност в пространството	<b>Свойства:</b> Директна комуникация с даден получател(и); наличието на получател е задължително в даден момент от време <b>Пример:</b> изпращане на съобщения, отдалечно извикване	<b>Свойства:</b> Директна комуникация с даден получател(и); изпращащът и получателят са независими от гледна точка на времето
Несвързаност в пространството	<b>Свойства:</b> Изпращащът не е необходимо да идентифицира получателя; наличието на получател е задължително в даден момент от време <b>Пример:</b> IP multicast	<b>Свойства:</b> Изпращащът не е необходимо да идентифицира получателя; изпращащът и получателят са независими от гледна точка на времето <b>Пример:</b> Индиректна комуникация



# Взаимовръзка с асинхронната комуникация

## ❖ Асинхронна комуникация

- Изпращащът изпраща съобщение и продължава да работи без да се блокира

## ❖ Липса на свързаност във времето

- Изпращащът и получателят съществуват независимо един от друг
  - ✓ Получателят може да не съществува в момента на иницииране на комуникацията

## ❖ Комуникацията може да бъде асинхронна и да не изисква времева свързаност

## ❖ Индиректната комуникация може да се реализира и синхронно



# Групова комуникация: концепция

## ❖ Групова комуникация: определение

- Изпращане на съобщение до група от получатели и доставяне на съобщението до всички членове на групата

## ❖ Характеристики

- Изпращащът не идентифицира получателите
- Предоставя се абстракция на IP мултиicast комуникация при гарантиране на надеждност и осигуряване на възможност за откриване на повреди и групово управление

## ❖ Възможности за приложение

- Надеждно доставяне на информация до голям брой клиенти
- Поддръжка на колаборативни приложения, при които е необходимо разпространение на събития до множество потребители
- Поддръжка на стратегии за отказоустойчивост
- Поддръжка на системно управление и наблюдение, включваща стратегии за балансиране на натоварването

# Програмен модел: мултикаст комуникация

Използване на  
единична мултикаст  
операция

Намалено време за  
доставяне на  
съобщения до всички  
получатели

Гарантирана  
надеждност и ред на  
доставяне на  
съобщенията

Join



Join



Join



Group

Leave



# Процесни групи и обектни групи

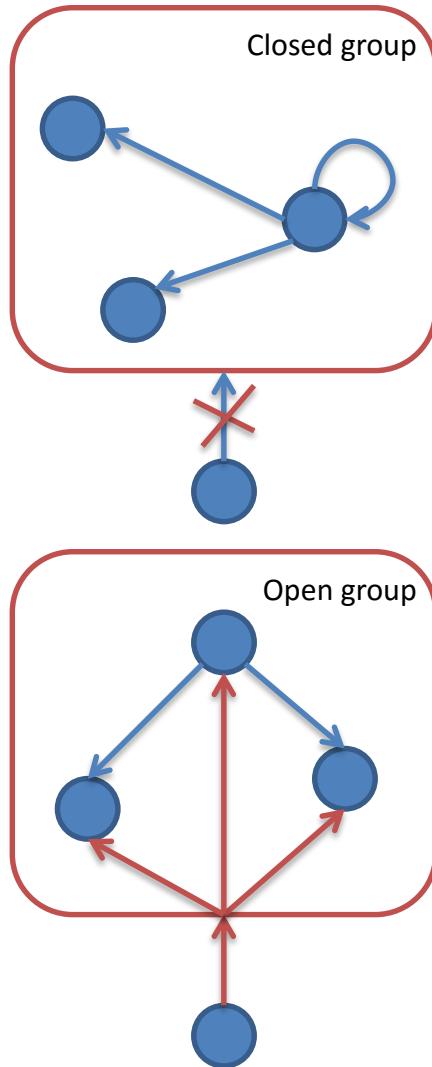
## ❖ Процесна група (подход на ниско ниво)

- Комуникиращите страни са процеси, които могат да се свързват или напускат групата
- **Съобщенията се доставят до процесите** без да се осигурява последваща поддръжка за диспечеризация
- **Съобщенията обикновено са неструктурирани масиви от байтове**, като не се осигурява маршализация на сложни данни типове

## ❖ Обектна група (подход на високо ниво)

- Колекция от обекти, които конкурентно обработват едно и също множество от извиквания, връщайки индивидуален отговор
- Клиентът използва прокси обект на групата, за да инициира извикването
- Обектните параметри и резултати се маршализират, а извикванията се диспечеризират автоматично
- Пример: Electra – CORBA базирана система, работеща в “прозрачен” и “непрозрачен” режим

## Типове групи



- ❖ **Затворени и отворени групи**
  - **Затворена група:** само членовете на групата могат да изпращат съобщения до нея
  - **Отворена група:** процес, извън групата може да изпраща съобщения до нея
- ❖ **Припокриващи се и неприпокриващи се групи**
  - **Припокриващи се групи:** процесите и обектите могат да участват в няколко групи
  - **Неприпокриващи се групи:** всеки процес принадлежи на точно определена група
- ❖ **Синхронна и асинхронна комуникация**
  - Груповата комуникация може да се реализира както синхронно, така и асинхронно

# Надеждност и ред на съобщенията

## ❖ Доставяне на съобщения при групова комуникация

- Гарантиране, че дадено множество от съобщения ще се достави до всички членове на групата
- Гарантиране, че редът на съобщенията ще се запази при доставяне до всеки член от групата

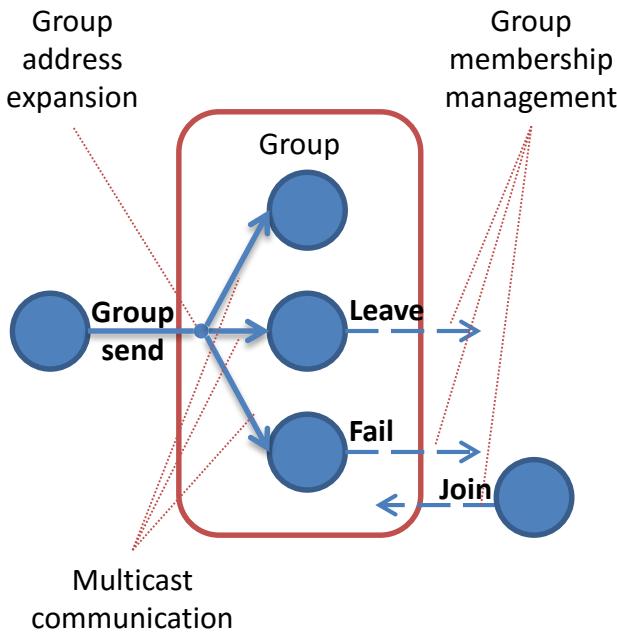
## ❖ Надеждност

- **Интегритет:** Доставянето на съобщенията е еднократно (at-most-once)
- **Валидност:** Изпратените съобщения рано или късно ще бъдат доставени
- **Договор:** Ако съобщението е доставено до един процес, то ще бъде доставено и до останалите членове на групата

## ❖ Ред на съобщенията

- **Не се гарантира от примитивите за между процесна комуникация**
- Подреждане FIFO
  - ✓ Съобщенията се доставят в реда на изпращане
- Причинно подреждане
  - ✓ Съобщенията се доставят в реда на възникването им в разпределената система
- Тотално подреждане
  - ✓ Ако съобщение е доставено на даден процес преди друго съобщение, то същия ред ще се запази и при останалите процеси

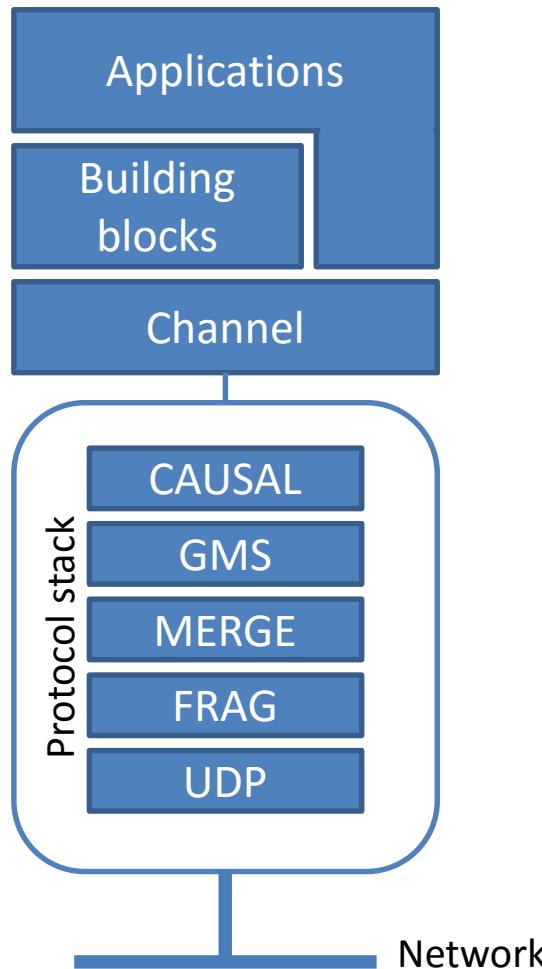
# Услуга за управление на групи



- ❖ Осигуряване на интерфейс за промяна на членството в групата
  - Операции за създаване и премахване на групи
  - Операции за добавяне или премахване на членове в групи
- ❖ Откриване на повреди
  - Наблюдение на повреди в самите процеси
  - Наблюдение на комуникационни проблеми поради наличие на недостъпни процеси
- ❖ Нотификация на членовете в групата за промени
  - Членовете на групата се уведомяват при добавяне на нов или премахване на съществуващ процес в групата
- ❖ Групов адрес
  - Изпращащият процес използва идентификатор на групата, за да изпрати съобщение
  - Услугата, управляваща групата, използва идентификатора, за да насочи съобщението към членовете на групата

JGroups toolkit

# CASE STUDY



## ❖ Предназначение

- Осигуряване на надеждна комуникация между процеси
- Присъединяване и премахване на процес към група
- Изпращане на съобщение към всички участници в групата или до единичен участник
- Получаване на съобщение от група

## ❖ Компоненти

- Канали
  - ✓ Осигуряват функционалност за присъединяване и напускане на група, изпращане и получаване на съобщения
- Градивни блокове
  - ✓ Осигуряват високо ниво на абстракция
- Протоколен стек

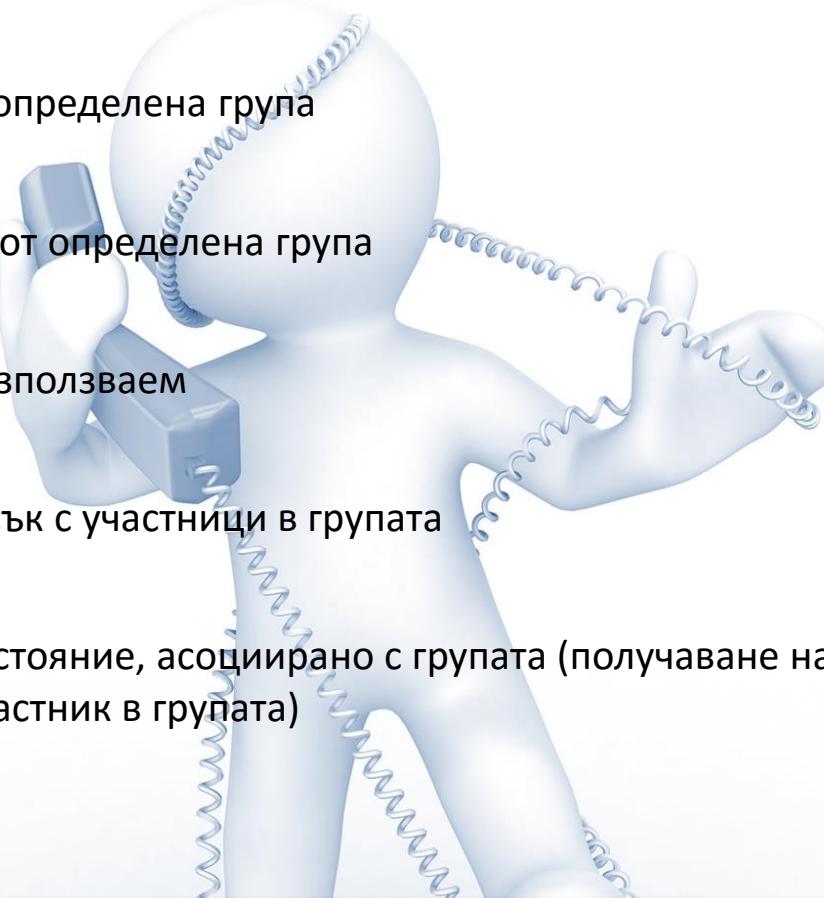


## ❖ Канал

- Обект, осигуряващ комуникация с групата

## ❖ Операции

- Connect
  - ✓ Свързва канала с определена група
- Disconnect
  - ✓ Премахва процес от определена група
- Close
  - ✓ Прави канала неизползваем
- getView
  - ✓ Връща текущ списък с участници в групата
- getState
  - ✓ Връща минало състояние, асоциирано с групата (получаване на предишни събития от нов участник в групата)



# Java клас FireAlarmJG

```
import org.jgroups.JChannel;  
public class FireAlarmJG {  
    public void raise() {  
        try {  
            JChannel channel = new JChannel();  
            channel.connect("AlarmChannel");  
            Message msg = new Message(null, null, "Fire!");  
            channel.send(msg);  
        }  
        catch(Exception e) {  
        }  
    }  
}
```



## ❖ Параметри на операцията Message

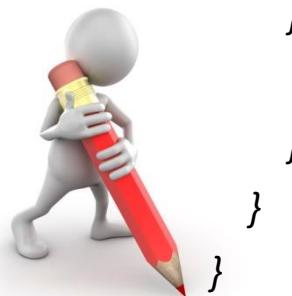
- Адрес, на който се изпраща съобщението (null изпраща до всички)
- Източник на съобщението
- Полезен товар на съобщението

## ❖ Създаване на инстанция на класа FireAlarmJG и изпращане на съобщение

```
FireAlarmJG alarm = new FireAlarmJG ();  
alarm. raise();
```

# Java клас FireAlarmConsumerJG

```
import org.jgroups.JChannel;  
  
public class FireAlarmConsumerJG {  
  
    public String await() {  
        try {  
            JChannel channel = new JChannel();  
            channel.connect("AlarmChannel");  
            Message msg = (Message) channel.receive(0);  
            return (String) msg.GetObject();  
        } catch(Exception e) {  
            return null;  
        }  
    }  
}
```



- ❖ Параметри на операцията Receive
  - Времеви интервал
- ❖ Създаване на инстанция на класа FireAlarmConsumerJG и получаване на съобщение

```
FireAlarmConsumerJG alarmCall = new FireAlarmConsumerJG();  
String msg = alarmCall.await();
```

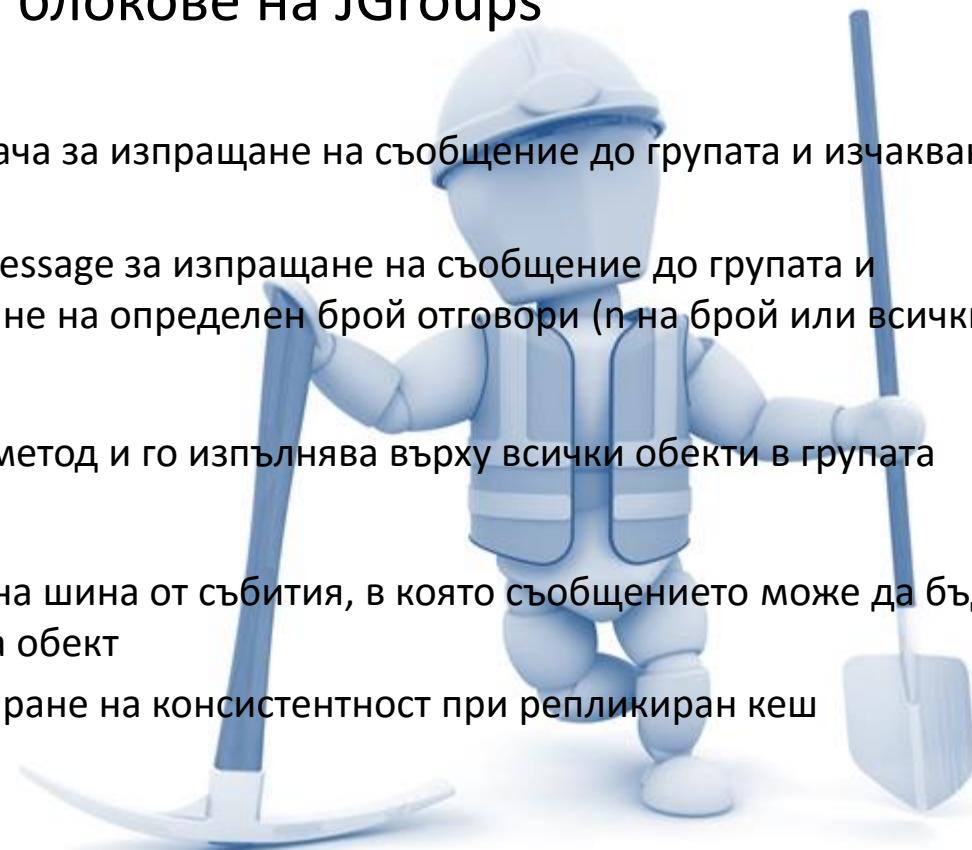
# Градивни блокове

## ❖ Градивен блок

- Абстракция от по-високо ниво върху класа за създаване на канал
- Предоставят шаблони за комуникация

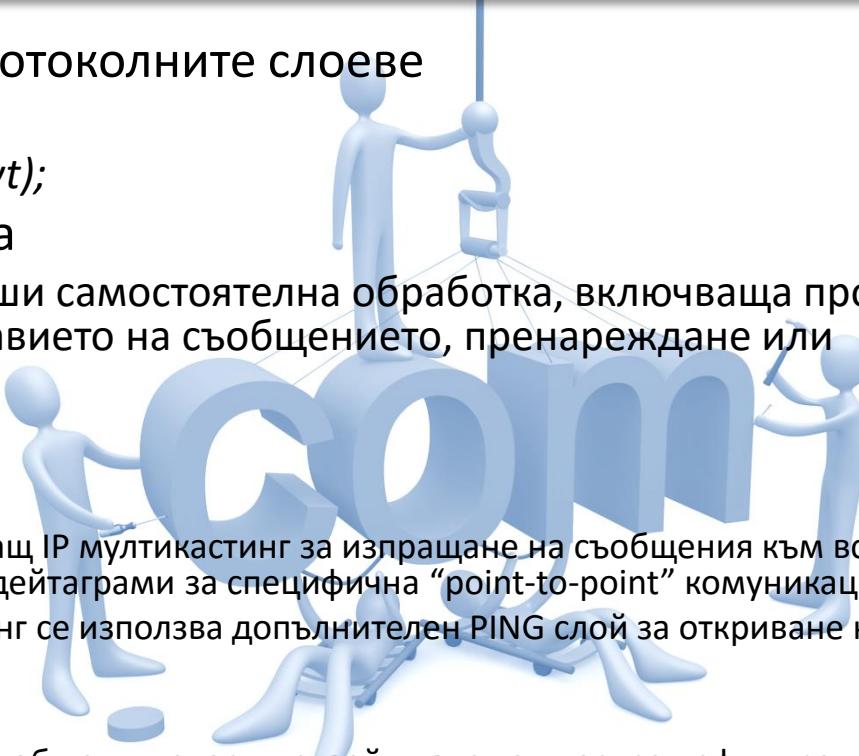
## ❖ Примери за градивни блокове на JGroups

- *MessageDispatcher*
  - ✓ Използва се от изпращаца за изпращане на съобщение до групата и изчакване на отговор(и)
  - ✓ Осигурява метод castMessage за изпращане на съобщение до групата и блокиране до получаване на определен брой отговори (n на брой или всички)
- *RpcDispatcher*
  - ✓ Получава специфичен метод и го изпълнява върху всички обекти в групата
- *NotificationBus*
  - ✓ Реализира разпределена шина от събития, в която съобщението може да бъде всеки сериализуем Java обект
  - ✓ Използва се за реализиране на консистентност при репликиран кеш



# Протоколен стек

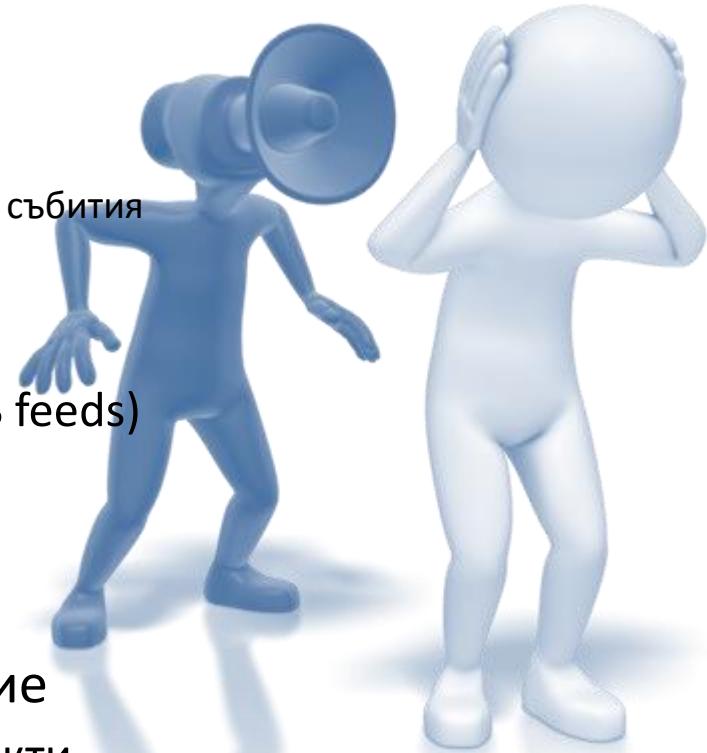
- ❖ Методи, реализирани в протоколните слоеве
  - *Public Object up (Event evt);*
  - *Public Object down (Event evt);*
- ❖ Обработка на съобщенията
  - Всеки слой може да извърши самостоятелна обработка, включваща промяна на съдържанието или заглавието на съобщението, пренареждане или премахване на съобщения
- ❖ Протоколни слоеве
  - UDP
    - ✓ Транспортен слой, използващ IP мултикастинг за изпращане на съобщения към всички участници в групата и UDP дейтаграми за специфична “point-to-point” комуникация
    - ✓ При липса на IP мултикастинг се използва допълнителен PING слой за откриване на участниците
  - FRAG
    - ✓ Реализира пакетиране на съобщенията, осигурявайки възможност за дефиниране на максимален размер (8,192 байта по подразбиране)
  - MERGE
    - ✓ Използва се при неочеквано разделяне на мрежата на подгрупи (клъстери) и последващото им сливане
  - GMS
    - ✓ Реализира протокол за членство в групата
  - CAUSAL
    - ✓ Реализира причинно подреждане на съобщенията



# Publish-subscribe системи: въведение

## ❖ Разпределени системи, базирани на събития

- Издатели
  - ✓ Публикуват структурирани събития
- Получатели
  - ✓ Абонират се за получаване на определени събития



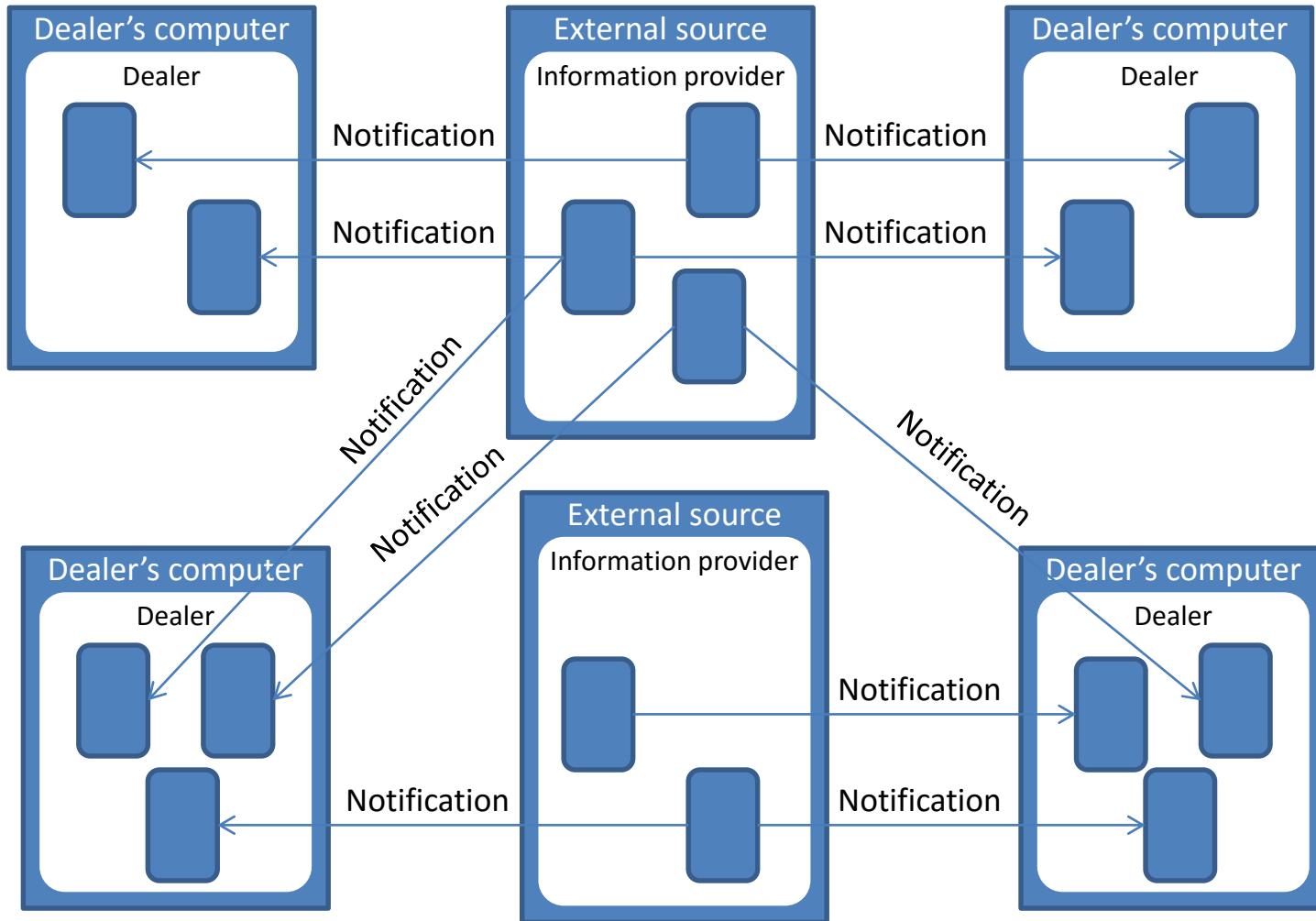
## ❖ Приложения

- Финансови информационни системи
- Даннови емисии в реално време (RSS feeds)
- Поддръжка на кооперативна работа
- Приложения за мониторинг
- Google 'ad clicks'

## ❖ Dealing Room: примерно приложение

- Информацията се представя чрез обекти
- Процес за доставяне на търговска информация (information process)
  - ✓ Публикува събития за промени в стокова борса от определен външен източник
- Дилърски процес (dealer process)
  - ✓ Абонира се за получаване на информация за събития, свързани с определени акции

# Архитектура на приложението Dealing Room



# Характеристики на publish-subscribe системите

## ❖ Хетерогенност

- Осигурява комуникация между компоненти в разпределените системи, които не са проектирани да взаимодействат

## ❖ Асинхронност

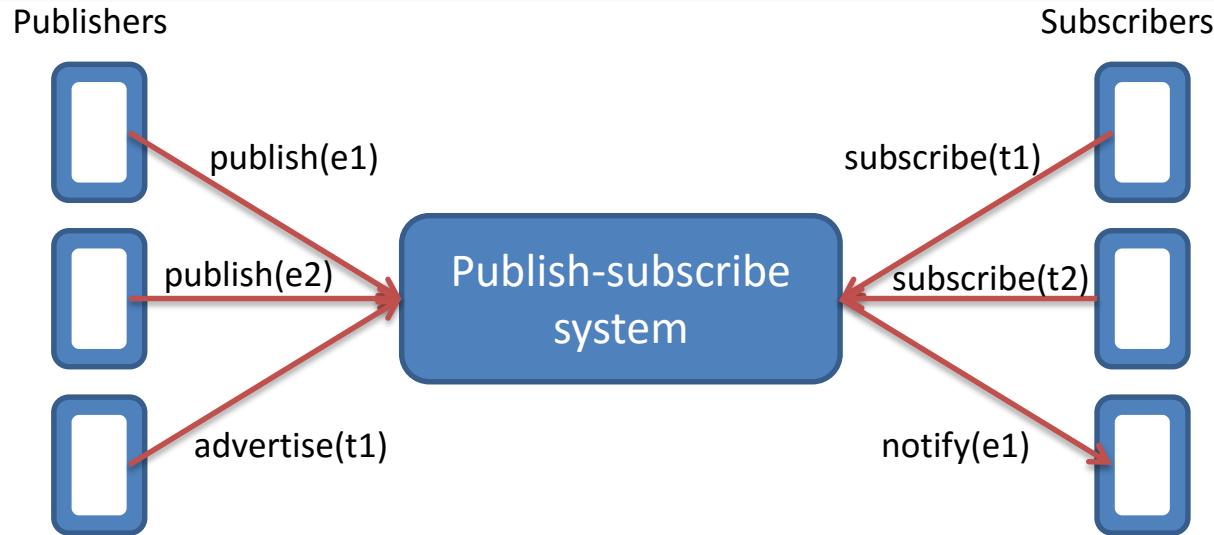
- Издателите и получателите на съобщенията не са свързани
- Системата Mushroom
  - ✓ Осигурява колаборативна работа в споделено работно пространство посредством представянето на потребители и информация като обекти
  - ✓ Промените в обектите се представлят със събития

## ❖ Възможност за реализиране на различни гаранции за доставка

- **Ненадеждно доставяне:** липса на гаранция, че всеки получател ще получи определено нотификационно съобщение
- **Надеждно доставяне:** всички получатели задължително получават нотификационните съобщения (Dealing room)
- **Комбинирано реализиране на гаранция:** осигуряване на надеждно доставяне в сървъра и ненадеждно доставяне при потребителските компютри, използваващи обектни реплики (системата Mushroom)

## ❖ Възможност за изпращане и получаване на нотификации в реално време

# Publish-subscribe парадигма



- ❖ ***publish(e)***
  - Разпространение на събитие  $e$
- ❖ ***subscribe(f)***
  - Заявяване на интерес към множество от събития, върху които е приложен филтър  $f$
- ❖ ***unsubscribe(f)***
  - Отказване за получаване на нотификации за множество от събития
- ❖ ***notify(e)***
  - Изпращане на нотификация
- ❖ ***advertise(f)***
  - Деклариране на доставка за определен тип събития от издателите
- ❖ ***unadvertise(f)***
  - Отказване на доставка за определен тип събития от издателите

# Модели за абониране

## ❖ Модел, базиран на **канал**

- Издателите публикуват в именувани канали
- Получателите се абонират именуван канал и получават всички съобщения от него
- Изискава създаване на **физически канал**

## ❖ Модел, базиран на **тема**

- Нотификациите се представят с набор от полета, едно от които представя тема
- Възможност за йерархична организация на темите

## ❖ Модел, базиран на **съдържание**

- Абонаментът се дефинира върху множество от полета на нотификационните съобщения
- Използва се език за дефиниране на заявки

## ❖ Модел, базиран на **тип**

- Основава се на обекто-ориентираните подходи, при които обектите се характеризират с тип
- Абонаментът се дефинира върху типове на събития
- Гранулярността на филтрите варира
  - ✓ Филтрите с финна гранулярност са подобни на тези при модела, базиран на съдържание

# Модели за абониране

## ❖ Модел, базиран на обект

- Аналог на модела, базиран на тип
- Абонаментът е базиран на промяна в състоянието на обектите
- Осигурява се реакция на обект при промяна в състоянието на друг обект
- Комуникацията е асинхронна

## ❖ Модел, базиран на контекст

- Контекстът е аспект на физическите обстоятелства, влияещи върху поведението на системата
- Пример за контекст: местоположение
  - ✓ Абонамент за съобщения, известяващи настъпването на авария в определена сграда

# Централизирана и разпределена реализация

## ❖ Централизирана реализация

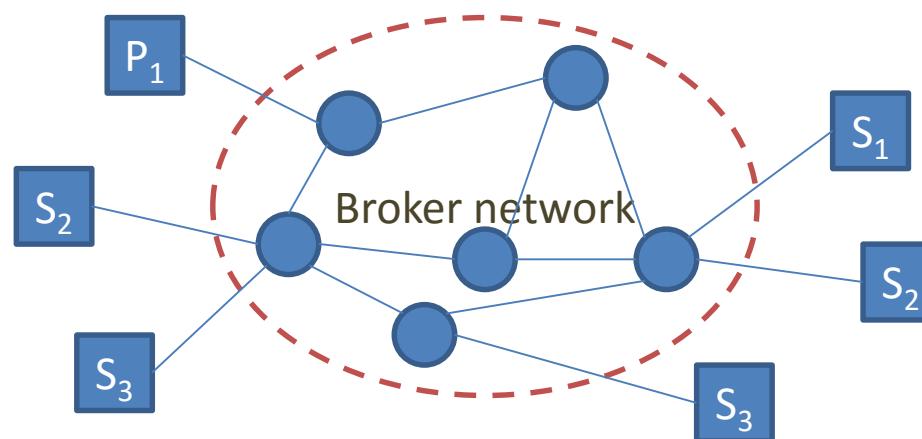
- Създаване на централизиран брокер в сървъра, който се използва от издателите за публикуване на съобщения и от получателите за абониране и получаване на нотификации
- Характеризира се с понижена гъвкавост и скалируемност
  - ✓ Брокерът представлява единична точка за потенциални проблеми

## ❖ Разпределена реализация

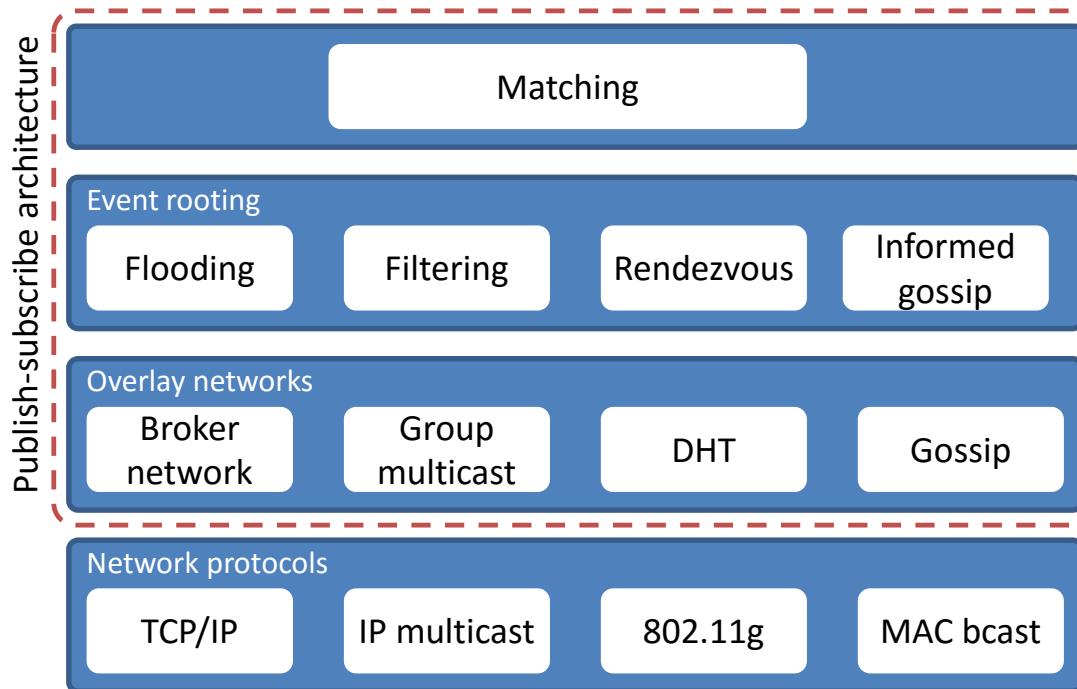
- Създаване на мрежа от брокери
- Характеризира се с по-висока устойчивост на повреди и скалируемост

## ❖ Цялостна peer-to-peer реализация

- Всички възли действат като брокери, реализиращи функционалност за маршрутизиране на съобщения



# Архитектура на publish-subscribe системите



- ❖ Слой 1: Услуги за между процесна комуникация
- ❖ Слой 2: Мрежова инфраструктура
  - Подпомага маршрутизирането на събития посредством осигуряване на мрежа от брокери или peer-to-peer структури
- ❖ Слой 3: Маршрутизация на събития
  - Осигурява ефективно маршрутизиране на събитията до съответните получатели
- ❖ Слой 4: Осигуряване на съответствие
  - Гарантира, че събитията, досатигнали до даден получател отговарят на абонамента му

# Flooding: концепция

## ❖ Алтернативи за реализация

- Изпращане на нотификация до всички възли в мрежата и извършване на подходящо съпоставяне на събитията и абонаментите при получателите
- Изпращане на абонаментите до всички издатели и съпоставяне на събитията, след което нотификациите се изпращат директно до получателите посредством комуникация от точка до точка (point-to-point)
- Организиране на брокери в ацикличен граф, при което всеки брокер препраща нотификациите за събития до всички свои съседи

## ❖ Мрежова инфраструктура

- Броудкастинг или мултикастинг

## ❖ Недостатък

- Завишен мрежов трафик

# Filtering: концепция

## ❖ Филтрация в мрежа от брокери

- Препращане на нотификация от брокер по мрежата при наличие на път до валиден получател
- Разпространяване на абонаментите по мрежата до потенциалните издатели и съхраняване на състояние в брокерите

## ❖ Съхраняване на състоянието

- Поддържане на списък със съседи във всеки възел
  - ✓ Списък със свързаните съседи в мрежата от брокери
- Поддържане на абонаментен списък
  - ✓ Списък на директно свързаните получатели, обслужвани от брокера
- Поддържане на маршрутизираща таблица
  - ✓ Съхранява списък със съседи и валидни абонаменти

## ❖ Реализиране на функция за съпоставяне (match) в брокерите

- Получава като вход нотификация за събитие и списък с възли и техните абонати
- Връща като резултат списък с възли, при които нотификационното съобщение съответства на абонамента

## ❖ Алтернативно решение: advertisement

- Разпространение на предложения с определени събития от издателите сред получателите

# Filtering: алгоритъм

```
1  upon receive publish(event e) from node x
2      matchlist := match(e, subscriptions)
3      send notify(e) to matchlist;
4      fwdlist := match(e, routing);
5      send publish(e) to fwdlist - x;
6  upon receive subscribe(subscription s) from node x
7      if x is client then
8          add x to subscriptions;
9      else add(x, s) to routing;
10     send subscribe(s) to neighbours - x;
```

## ❖ Публикуване на събитие

- Изпращане на заявка за публикуване на събитие от даден възел до брокер
- Изпращане на събитието до всички свързани възли, в които е налице съответствие в абонаментния списък (ред 2 и ред 3)
- Препращане на събитието по всички пътища, за които е налице съответствие в маршрутизиращата таблица (ред 4 и ред 5)

## ❖ Заявка за абонамент

- Изпращане на заявка за абонамент от даден получател до брокер
- Ако заявката е от непосредствено свързан получател, то тя се записва в абонаментната таблица (ред 7 и ред 8)
- Ако брокерът е междинен възел, то е наличен път до съответния абонамент и се добавя запис в маршрутизиращата таблица (ред 9)
- Заявката за абонамента се препраща до всички съседи (ред 10)

# Rendezvous: концепция

```
upon receive publish(event e) from node x at node i
    rvlist := EN(e);
    if i in rvlist then begin
        matchlist :=match(e, subscriptions);
        send notify(e) to matchlist;
    end
    send publish(e) to rvlist - i;
upon receive subscribe(subscription s) from node x at
node i
    rvlist := SN(s);
    if i in rvlist then
        add s to subscriptions;
    else
        send subscribe(s) to rvlist - i;
```

- ❖ Представяне на събитията с пространство и разделяне на пространството между брокерите
- ❖ Отговорността върху подмножество от пространството на събитията се поема от брокер, представляващ възел “рандеву”
- ❖ Функции на рандеву алгоритъма
  - $SN(s)$ : Получава като вход даден абонамент  $s$  и връща всички рандеву възли, които са отговорни за абонамента
  - $EN(e)$ : Получава като вход публикувано събитие и връща един или няколко рандеву възли, които са отговорни за откриване на съответствие между събитието и абонаментите за него

## ❖ Разпределена хеш таблица

- Използва се при рандеву маршрутизацията
- Дистрибутира се сред множество от възли в peer-to-peer мрежа
- Хеш функцията се използва за свързване едновременно на събития и абонаменти със съответните рандеву възли

## ❖ Разпространение на “ключки”

- Възлите в мрежата периодично и вероятностно обменят събития със съседните си възли
- Предлага алтернативна стратегия на “flooding” концепцията
- Възможност за използване на съдържанието на събитията (informed gossip)



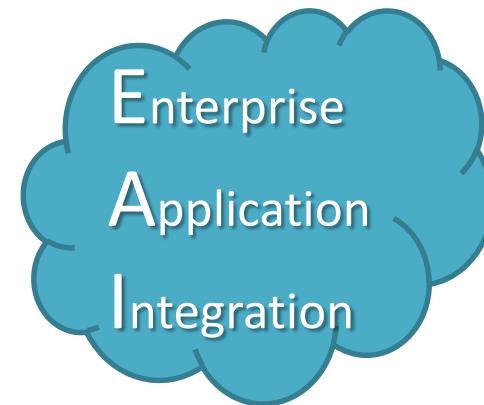
# Примерни publish-subscribe системи

System	Subscription model	Distribution model	Event routing
<b>CORBA event service</b>	Channel-based	Centralized	-
<b>TIB Rendezvous</b>	Topic-based	Distributed	Filtering
<b>Scribe</b>	Topic-based	Peer-to-peer (DHT)	Rendezvous
<b>TERA</b>	Topic-based	Peer-to-peer	Informed gossip
<b>Siena</b>	Content-based	Distributed	Filtering
<b>Gryphon</b>	Content-based	Distributed	Filtering
<b>Hermes</b>	Topic- and Content-based	Distributed	Rendezvous and Filtering
<b>MEDYМ</b>	Content-based	Distributed	Flooding
<b>Meghdoot</b>	Content-based	Peer-to-peer	Rendezvous
<b>Structure-less CBR</b>	Content-based	Peer-to-peer	Informed gossip

# Опашки със съобщения: point-to-point услуга



ORACLE®



Microsoft

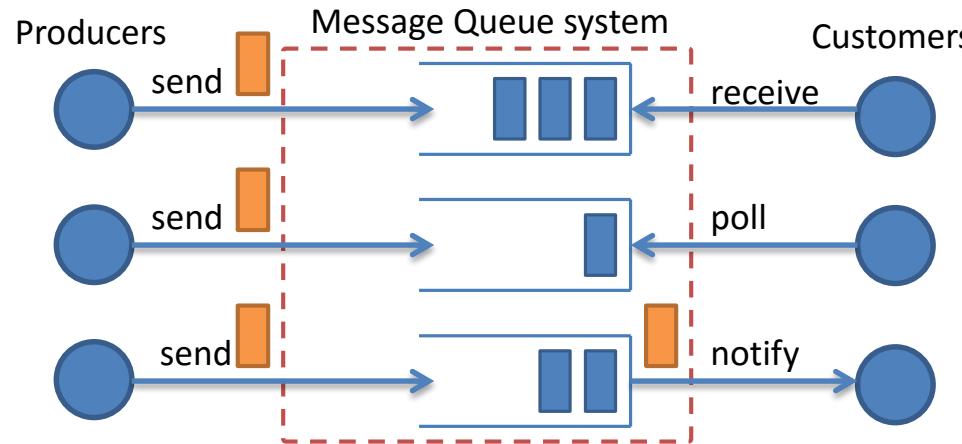
# Опашки със съобщения: парадигма

## ❖ Основни операции

- Изпращане на съобщение в опашка
- Получаване на съобщение от опашка

## ❖ Стилове за получаване на съобщения

- Блокиращо получаване
  - ✓ Блокирането продължава докато съответното съобщение е налично
- Неблокиращо получаване
  - ✓ Статусът на опашката се проверява и се връща съобщение или индикация за липса на съобщение
- Нотификация
  - ✓ Нотификация за събитие при наличие на дадено съобщение в опашката



# Опашки със съобщения: програмен модел

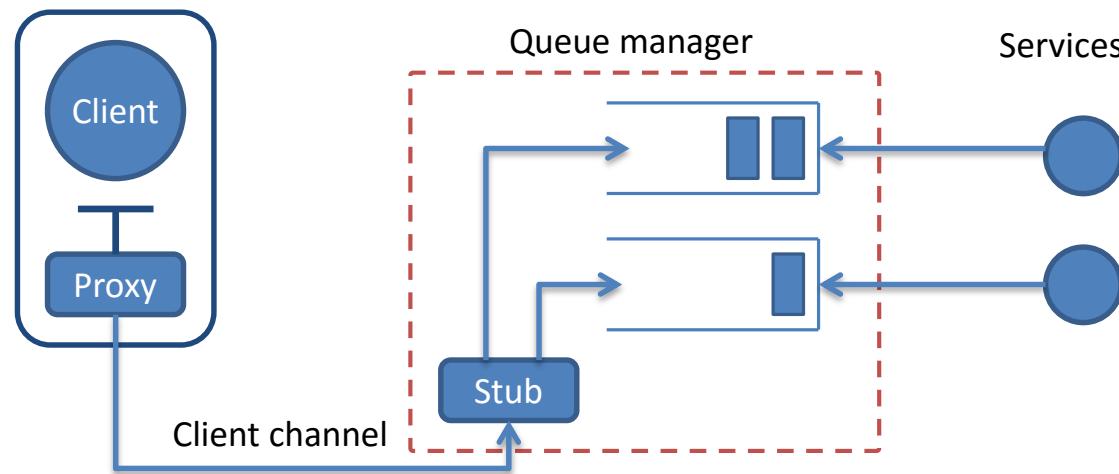
- ❖ Политики за обработка на опашки
  - Определят реда на извличане на съобщения от опашката
    - ✓ FIFO, приоритети, селекция въз основа на свойства
- ❖ Структура на съобщението
  - Идентификатор на опашка, към която се изпраща съобщението
  - Метадани (приоритет, модел на доставка и др.)
  - Тяло за съхранение на полезния товар
- ❖ Свойства на системите, базирани на опашки
  - Персистентност на съобщенията
  - Надеждна доставка на съобщения: съобщенията се доставят винаги и еднократно
- ❖ Допълнителна функционалност
  - Изпращане или получаване на съобщения в транзакция
  - Трансформиране на съобщения с цел решаване на проблеми при интеграция на хетерогенни системи
    - ✓ Често се извършва от брокерите
  - Осигуряване на механизми за сигурност, включващи конфиденциално предаване на данни с SSL, автентикация и управление на достъпа

WebSphere MQ

# CASE STUDY

# WebSphere MQ: концепция

- ❖ Мениджър на опашки
  - Хоства и управлява опашки в определен физически сървър
- ❖ Интерфейс за управление на опашки (Message Queue Interface)
  - Предоставя операции за създаване и прекъсване на връзка с опашка, изпращане и получаване на съобщения в/от опашка
- ❖ Клиентски канал
  - Осигурява комуникация между клиентското приложение и мениджъра на опашки
- ❖ Канал за съобщения
  - Осигурява асинхронна обмяна на съобщения между два мениджъра на опашки
- ❖ Прокси
  - Реализира MQI команди при клиента
- ❖ Агент на канал за съобщения
  - Осигурява установяване и управление на канал за съобщения между мениджърите на опашки
- ❖ Маршрутизиращи таблици
  - Налични са при мениджърите на опашки



# WebSphere MQ: hub-and-spoke топология

- ❖ Hub мениджър
  - Мениджър на опашки, предоставяящ набор от услуги
- ❖ Spoke мениджър
  - Мениджър на опашки, който е достъпен за клиентските приложения и предава съобщения в опашката на hub мениджъра за обработка
- ❖ Приложение
  - Машабни системи, покриващи големи географски области
- ❖ Комуникация
  - RPC стил на комуникация между клиентските приложения и spoke мениджър
    - ✓ Клиентското приложение се блокира до момента, в който съобщението се депозира в локалния мениджър
  - Асинхронна, неблокираща комуникация между мениджърите на опашки
- ❖ Недостатък
  - Наличие на единствена точка за повреда при hub мениджъра
  - Решение: създаване на клъстери от мениджъри на опашки и балансиране на натоварването между тях

Java Message Service (JMS)

# CASE STUDY

# JMS: концепция

## ❖ Същност на JMS

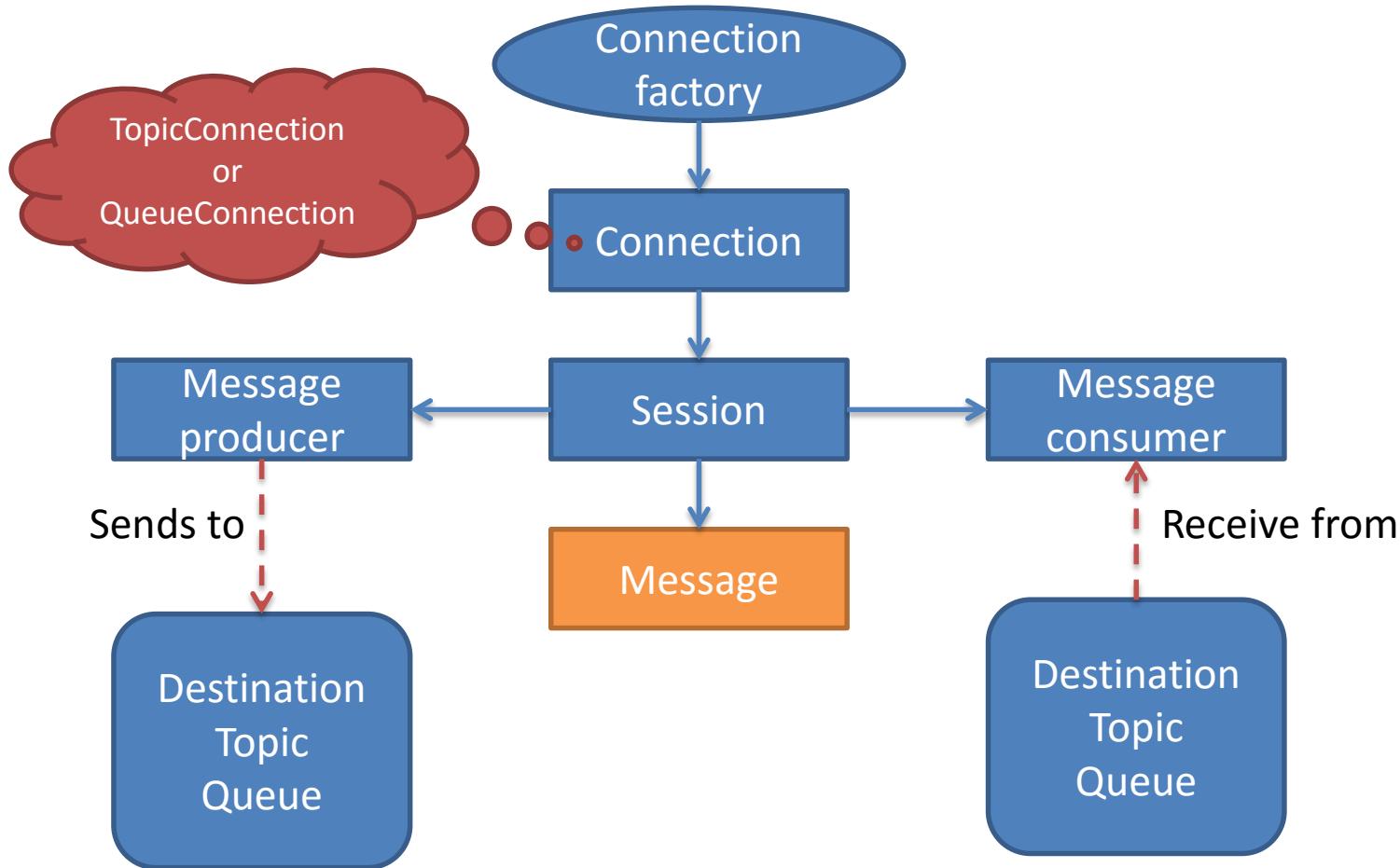
- Спецификация на стандартизиран начин за индиректна комуникация между разпределени Java приложения
- Поддържа теми и опашки като алтернативни варианти за доставяне на съобщения

## ❖ Роли в JMS

- JMS клиент
  - ✓ Java програма или компонент, които генерираят или консумират съобщения (JMS производител или JMS консуматор)
- JMS доставчик
  - ✓ Система, която реализира JMS спецификацията
- JMS съобщение
  - ✓ Обект, който се използва за комуникация между JMS клиентите
- JMS дестинация
  - ✓ Обект, поддържащ индиректна комуникация в JMS (JMS тема или JMS опашка)



# JMS: програмен модел



# JMS: елементи на програмния модел

## ❖ Връзка (Connection)

- Логически канал между клиента и доставчика

## ❖ Типове връзки

- TopicConnection
- QueueConnection

## ❖ Фабрика за връзки (Connection factory)

- Услуга, отговорна за създаване на връзка между клиента и доставчика

## ❖ Сесия (Session)

- Серия от операции, включващи създаване, изпращане и консумиране на съобщения, свързани с определена задача

## ❖ Структура на съобщението

- Заглавие: Съдържа информация за идентифициране и маршрутизиране на съобщението
  - ✓ Дестинация (тема или опашка), приоритет, дата на валидност, идентификатор и timestamp поле
- Свойства: Съхраняват специфична за приложението информация
- Тяло: Пренася полезния товар на съобщението

# JMS: елементи на програмния модел

## ❖ Производител на съобщение (Message producer)

- Обект, който публикува съобщения под определена тема или в определена опашка

## ❖ Консуматор на съобщение (Message consumer)

- Обект, осигуряващ абониране за съобщения по определена тема или в определена опашка
- Позволява филтриране на съобщенията въз основа на информацията в заглавието и свойствата

## ❖ Модели за получаване на съобщения

- Блокиращ режим с използване на операция receive
- Режим с използване на обект “слушател” (message listener), осигуряващ onMessage метод

# JMS пример: Java клас FireAlarmJMS

```
import javax.jms.*;
import javax.naming.*;
public class FireAlarmJMS {
public void raise() {
    try {
        Context ctx = new InitialContext();
        TopicConnectionFactory topicFactory = (TopicConnectionFactory)ctx.lookup ("TopicConnectionFactory");
        Topic topic = (Topic)ctx.lookup("Alarms");
        TopicConnection topicConn = topicConnectionFactory.createTopicConnection();
        TopicSession topicSess = topicConn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
        TopicPublisher topicPub = topicSess.createPublisher(topic);
        TextMessage msg = topicSess.createTextMessage();
        msg.setText("Fire!");
        topicPub.publish(message);
    } catch (Exception e) {
    }
}
```



- ❖ Създаване на инстанция на класа FireAlarmJMS и изпращане на съобщение

```
FireAlarmJMS alarm = new FireAlarmJMS ();
alarm.raise();
```

# JMS пример: Java клас FireAlarmConsumerJMS

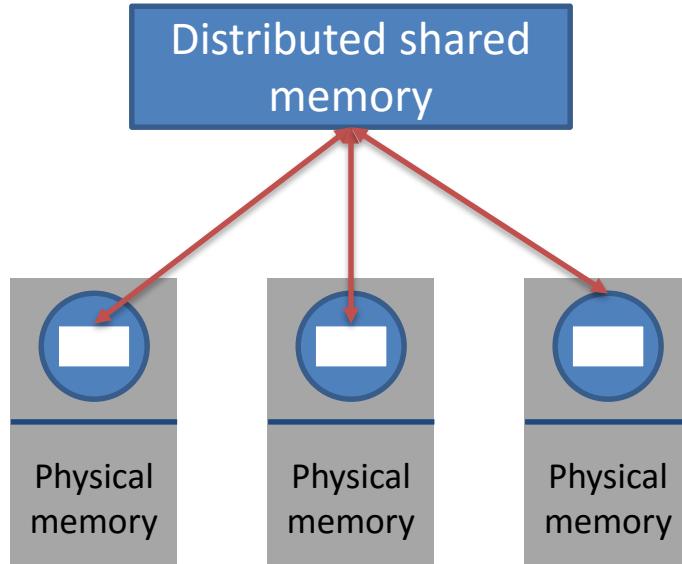
```
import javax.jms.*; import javax.naming.*;
public class FireAlarmConsumerJMS
public String await() {
    try {
        Context ctx = new InitialContext();
        TopicConnectionFactory topicFactory = (TopicConnectionFactory)ctx.lookup("TopicConnectionFactory");
        Topic topic = (Topic)ctx.lookup("Alarms");
        TopicConnection topicConn = topicConnectionFactory.createTopicConnection();
        TopicSession topicSess = topicConn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
        TopicSubscriber topicSub = topicSess.createSubscriber(topic);
        topicSub.start();
        TextMessage msg = (TextMessage) topicSub.receive();
        return msg.getText();
    } catch (Exception e) {
        return null;
    }
}
```



- ❖ Създаване на инстанция на класа FireAlarmConsumerJMS и получаване на съобщение

```
FireAlarmConsumerJMS alarmCall = new FireAlarmConsumerJMS();
String msg = alarmCall.await();
```

# Разпределена споделена памет: концепция



- ❖ Процесите **достъпват разпределената памет посредством четене и запис** по традиционния начин, както в собствено адресно пространство
- ❖ Прилежащата система за изпълнение осигурява възможност за **наблюдение на направените промени в данните** от различни процеси
- ❖ Осигурява се **директен достъп до споделената памет**, което премахва необходимостта от комуникация със съобщения в клиентските приложения
- ❖ Прилежащата система за изпълнение **управлява репликиране на данните**
- ❖ Възможност за реализация на **асинхронна комуникация и паралелна обработка**

# Изпращане на съобщения vs. разпределена споделена памет

## ❖ Предлагани услуги

- **Липса на необходимост от маршализиране на данните при използване на споделена памет**
  - ✓ Възможност за именуване на споделените променливи и използване като локални
- При комуникация със съобщения процесите притежават частно адресно пространство
  - ✓ При използване на разпределена споделена памет е възможно **верижно разпространение на повреди** между процесите при грешно записване на данни
- При реализация на разпределена споделена памет **синхронизацията се базира на известни програмни конструкции**
  - ✓ Заключване на ресурси и използване на семафори
- Комуникацията при използване на разпределена споделена памет не изисква едновременна наличност на процесите във времето

## ❖ Ефективност

- Сравнително еднаква за относително малък брой компютри

## ❖ Цена

- При комуникация със съобщения винаги е известно дали дадена операция е вътрешна за процеса или изисква разходи за комуникация
- При споделената памет наличието на комуникация зависи от фактори като наличие на предишен достъп до данните и шаблон за споделяне

# TUPLE SPACE

# Разпределена споделена памет: програмен модел

## ❖ Същност на комуникацията

- Комуникацията между процесите се базира на колекция от подредени списъци с елементи, формиращи споделено пространство
  - ✓ <“fred”, 1958>, <“sid”, 1964>, <4, 9.8, “Yes”>

## ❖ Операции

- Запис на списък (write)
- Четене на списък (read)
- Прочитане и премахване на списък (take)

## ❖ Специфициране на списък

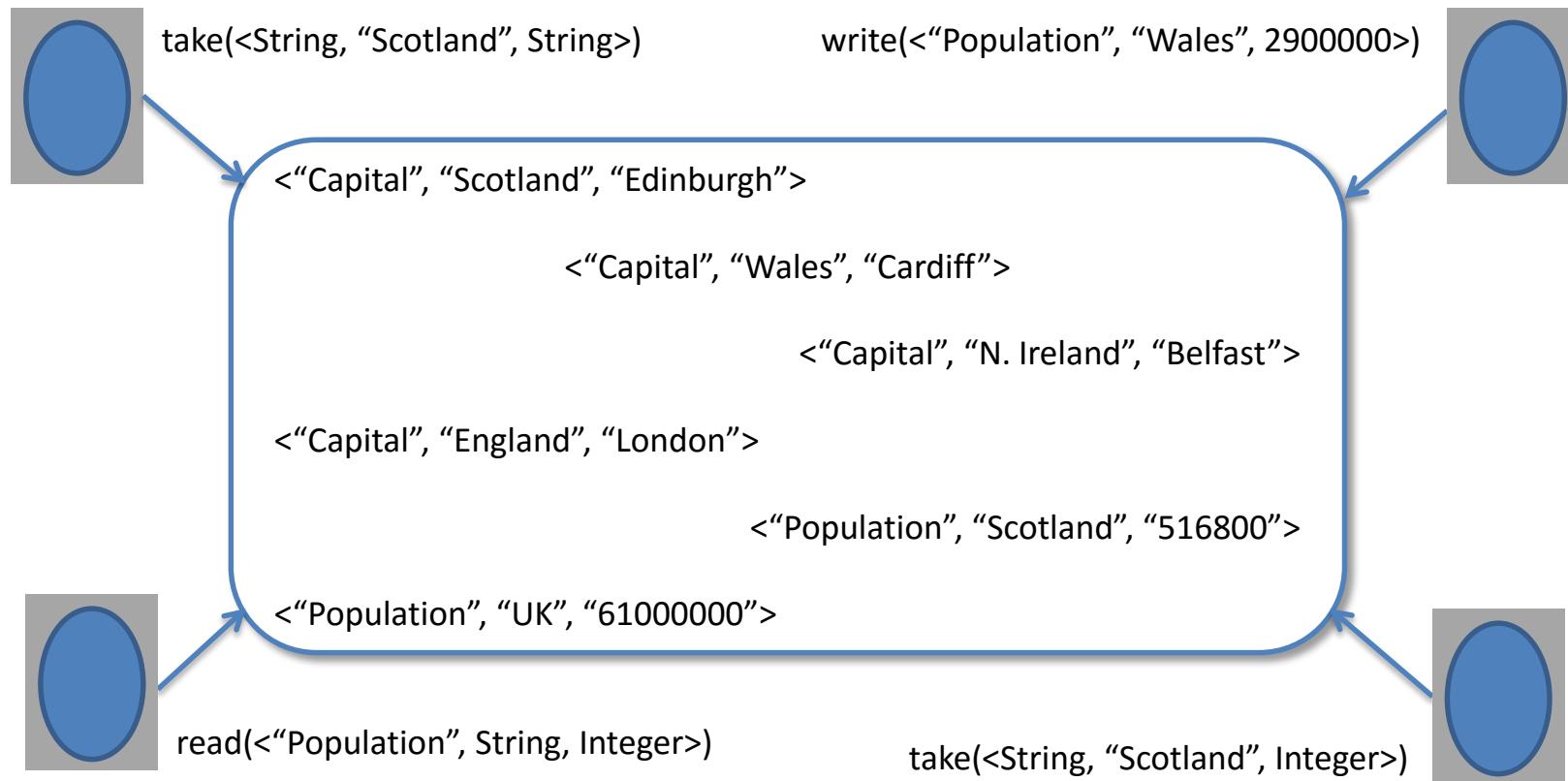
- Използва се за откриване на съответствие при четене и изтриване
  - ✓ take (<String, integer>) връща <“fred”, 1958> или <“sid”, 1964>
  - ✓ take (<String, 1958>) връща <“fred”, 1958>

## ❖ Достъп до списъците

- Липсва директен достъп, при който процесите директно променят списъците
- Пример за инкрементиране на брояч

```
<s, count> := myTS.take(<“counter”, integer>);  
myTS.take.write(<“counter”, count + 1>);
```

# Разпределена споделена памет: абстрактно представяне



## ❖ Липса на свързаност в пространството

- Списък в пространството може да се създаде от произволен брой процеси и може да се достави на произволен брой процеси

## ❖ Липса на свързаност във времето

- Списък съществува в пространството докато не бъде премахнат, при което изпращащът и получателят му не е необходимо да са налични едновременно

# Разпределена споделена памет: репликация

## ❖ Машина на състоянията

- Промяната в състоянието е отговор на събитие, получено от друга реплика или средата
- Осигуряване на консистентност
  - ✓ Репликите трябва да стартират при едно и също състояние
  - ✓ Репликите трябва да обработват събитията в еднакъв ред
  - ✓ Репликите трябва да реагират детерминистично на всяко събитие

## ❖ Репликационна стратегия на Xu&Liskov

- Изпълнява се върху определено множество от реплики
- **Списъците притежават логически имена**, определени от първия им елемент, спрямо който се групират в множества
- Системата се състои от т.нар. **работници**, извършващи изчисления над пространството със списъци, и набор от **реплики**
- Всеки мрежов възел може да съдържа произволен брой работници, реплики или и двете
- Възлите са свързани в комуникационна мрежа, допускаща **изгубване, дублиране и закъснение** на съобщенията, както **промени в реда на доставяне** на съобщенията

## ❖ Запис

1. Изпращане на съобщение за запис до всички реплики
2. Поставяне на списъка във всяка реплика и връщане на потвърждение
3. Повторение на стъпка 1 до получаване на потвърждение от всички реплики

*\*Необходимост от идентифициране на дублираните съобщения*

## ❖ Четене

1. Изпращане на съобщение за четене до всички реплики
2. Търсене на съответствие в репликите и връщане на открития списък при инициатора на операцията
3. Инициаторът връща като резултат първия получен списък от репликите, игнорирайки останалите
4. Повторение на стъпка 1 до получаване на поне един отговор

# Репликационната стратегия на Xu&Liskov: изтриване

## ❖ Фаза 1: Избор на списък за премахване

1. Изпращане на съобщение за премахване на списък до всички реплики
2. Опит за заключване на релевантните списъци в репликите и отмяна на операцията за премахване при невъзможност за заключване
3. Всяка реплика с успешно заключване връща множество от списъци, за които е открито съответствие
4. Стъпка 1 се повтаря до получаване на отговор от всички реплики и получаване на сечение от върнатите списъци, което не е празно
5. Иницииращият процес избира списък от сечението на списъците

## ❖ Фаза 2:

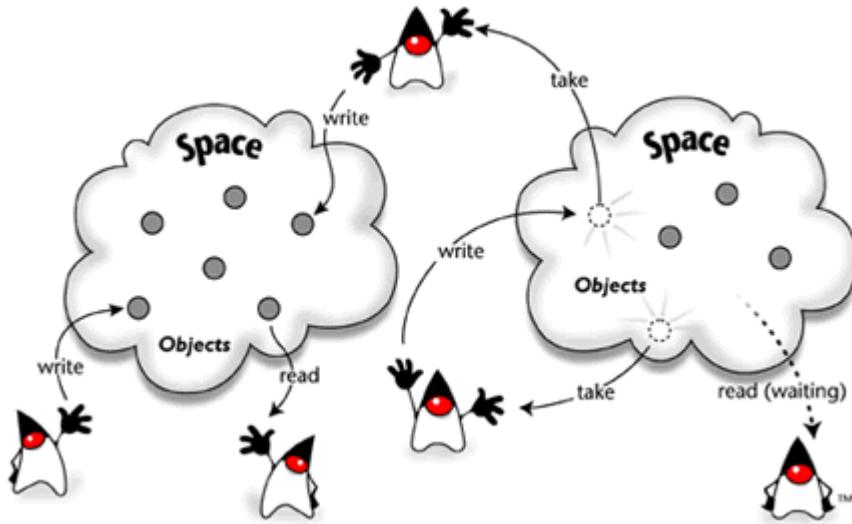
1. Изпращане на съобщение за изтриване на избрания списък до всички реплики
2. Репликите изтриват списъка, изпращат потвърждение и отключват релевантните списъци
3. Стъпка 1 се повтаря до получаване на потвърждение от всички реплики

## ❖ Минимизиране на закъснението породено от семантиката на операциите

- Операцията за четене блокира до получаване на отговор от първата реплика
- Операцията за премахване блокира до приключване на фаза 1
- Операцията за запис не е блокираща

## ❖ Допълнителни ограничения

- Изпълнението на операциите трява да се изпълняват от работниците в еднакъв ред при всяка реплика
- Операция за запис не може да бъде изпълнена в дадена реплика, ако не са приключили операциите за премахване, извършвани от даден работник във всички реплики



JavaSpaces

# CASE STUDY

# JavaSpaces: програмен модел

## ❖ Предназначение

- Инструмент за комуникация чрез пространство от списъци
- Спецификацията на JavaSpaces услугата се предоставя от Sun

## ❖ Същност

- Осигурява възможност за създаване на произволен брой инстанции на дадено пространство
- Пространството представлява споделено, персистентно хранилище на обекти

## ❖ JavaSpaces API

- *Lease write(Entry e, Transaction txn, long lease)*
  - ✓ Lease: време (милисекунди), за което обектът е достъпен
- *Entry read(Entry tmpl, Transaction txn, long timeout)*
  - ✓ Timeout: интервал за блокиране на процеса или нишката
- *Entry readIfExists(Entry tmpl, Transaction txn, long timeout)*
- *Entry take(Entry tmpl, Transaction txn, long timeout)*
- *Entry takeIfExists(Entry tmpl, Transaction txn, long timeout)*
- *EventRegistration notify(Entry tmpl, Transaction txn, RemoteEventListener listen, long lease, MarshaledObject handback)*
  - ✓ RemoteEventListener: интерфейс за нотификация

# JavaSpaces пример: Java клас AlarmTupleJS

```
import net.jini.space.JavaSpace;
public class FireAlarmJS {
public void raise() {
    try {
        JavaSpace space = SpaceAccessor.findSpace("AlarmSpace");
        AlarmTupleJS tuple = new AlarmTupleJS("Fire!");
        space.write(tuple, null, 60*60*1000);
    catch (Exception e) {
    }
}
}
```



- ❖ Създаване на инстанция на класа AlarmTupleJS и изпращане на съобщение

```
FireAlarmJS alarm = new FireAlarmJS ();
alarm.raise();
```

# JMS пример: Java клас FileAlarmConsumerJS

```
import net.jini.space.JavaSpace;
public class FireAlarmConsumerJS {
public String await() {
    try {
        JavaSpace space = SpaceAccessor.findSpace();
        AlarmTupleJS template = new AlarmTupleJS("Fire!");
        AlarmTupleJS recvd = (AlarmTupleJS) space.read(template, null, Long.MAX_VALUE);
        return recvd.alarmType;
    }
    catch (Exception e) {
        return null;
    }
}
}
```



- ❖ Създаване на инстанция на класа FireAlarmReceiverJS и получаване на съобщение

```
FireAlarmConsumerJS alarmCall = new FireAlarmConsumerJS();
String msg = alarmCall.await();
```





## РАЗПРЕДЕЛЕНИ ОБЕКТИ И КОМПОНЕНТИ

доц. д-р Десислава Петрова-Антонова

# Съдържание

- ❖ Въведение
- ❖ Разпределени обекти
- ❖ Case study: CORBA
- ❖ Разпределени компоненти
- ❖ Case studies: Enterprise JavaBeans and Fractal

# Въведение

## ❖ Разпределени обекти

- Комуникацията се осъществява посредством отдалечно извикване на методи или алтернативна парадигма (разпределени събития)
- Преимущества на подхода
  - ✓ Инкапсуляция
  - ✓ Даннова абстракция и скриване на реализацията
  - ✓ Възможност за реализиране на динамични и разширяеми решения

## ❖ Разпределени компоненти

- Преодоляват недостатъците на обектно-ориентирани разпределени платформи
  - ✓ Имплицитни зависимости
  - ✓ Програмна сложност
  - ✓ Обвързване на реализацията с механизмите за сигурност, прихващане на повреди и конкурентност
  - ✓ Липса на поддръжка при внедряване

# Предпоставки за възникване на разпределените обекти

## ❖ Разпределени системи

- Изчерпване на възможностите на клиент-сървър модела и необходимост от програмна абстракция на по-високо ниво

## ❖ Езици за програмиране

- Трудности при програмирането с първите обектно-ориентирани езици като Simula-67 и Smalltalk
- Поява на по-богати програмни абстракции (Java, C++)

## ❖ Софтуерно инженерство

- Разработване на обектно-ориентирани методи за проектиране

## ❖ Адаптиране на обектно-ориентираната парадигма в разпределена среда

- JavaRMI
  - ✓ Използва се **само** при разработване на Java приложения
- CORBA
  - ✓ Предлага независимо от програмния език решение (C++, Java, Python и др.)

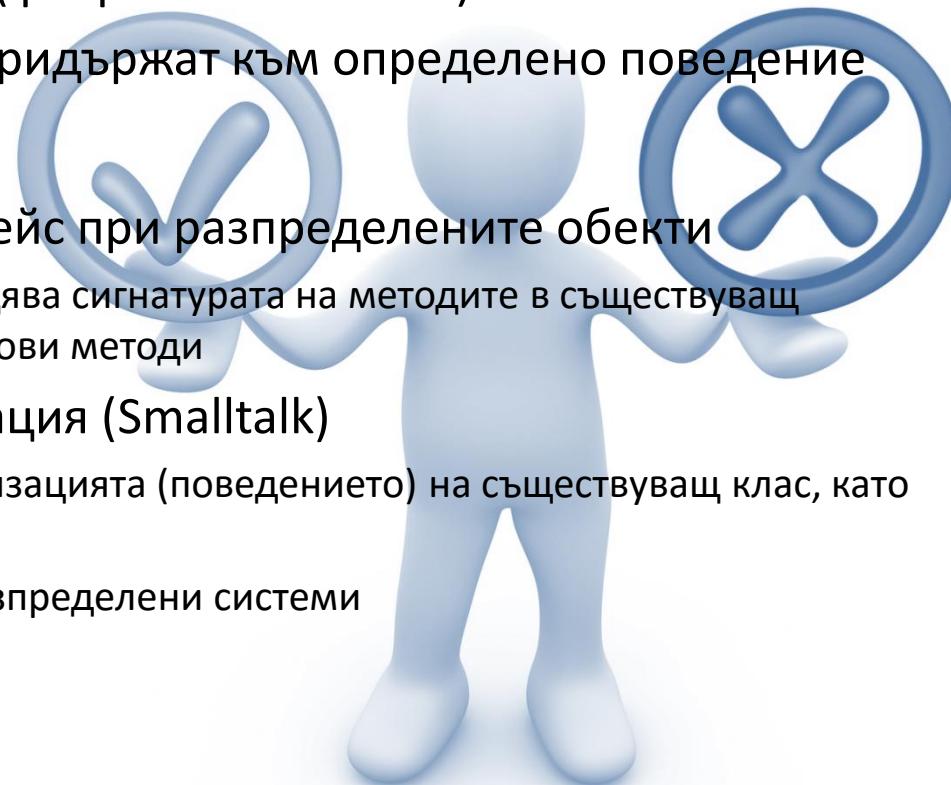
# Разлики между обектите и разпределените обекти

## ❖ Клас

- Описание на поведение, асоциирано с група от обекти (шаблон за създаване на обекти)
- Механизъм за инстанциране на обект или група от обекти с определено поведение (фабрика за класове)
- Група обекти, които се придържат към определено поведение

## ❖ Наследяване

- Наследяване на интерфейс при разпределените обекти
  - ✓ Новият интерфейс наследява сигнатурата на методите в съществуващ интерфейс, като добавя нови методи
- Наследяване на реализация (Smalltalk)
  - ✓ Нов клас наследява реализацията (поведението) на съществуващ клас, като добавя ново поведение
  - ✓ Реализира се трудно в разпределени системи



# Разпределени обекти: базови понятия

## ❖ Отдалечена референция

- Глобален уникален идентификатор, който може да се подава като параметър

## ❖ Отдалечен интерфейс

- Осигурява абстрактна спецификация на методи, които могат да бъдат извикани отдалечно

## ❖ Отдалечени действия

- Инициират се от отдалечно извикване на метод, което потенциално води до верига от извиквания

## ❖ Отдалечени изключения

- Изключения, породени от разпределения характер на системите (загуба на съобщения, сривове на процеси)

## ❖ Отдалечно изчистване на паметта

- Механизъм, който осигурява, че даден обект ще съществува докато са налични референции или отдалечени референции към него

# Допълнителна функционалност

## ❖ Комуникация между обекти в разпределена среда

- Обикновено се реализира посредством отдалечно извикване на методи
- Услуги за събития и нотификации в CORBA

## ❖ Управление на жизнения цикъл

- Създаване, миграране и изтриване на обекти в разпределена среда

## ❖ Активация и деактивация

- Активация
  - ✓ Процес на активиране на обект в разпределена среда посредством осигуряване на необходимите му ресурси за обработка на пристигащите извиквания
- Деактивация
  - ✓ Временна невъзможност на обекта да обработва извиквания

## ❖ Персистентност

- Необходимост от управление на персистентността за обектите, съхраняващи своето състояние

## ❖ Допълнителни услуги

- Услуги за именуване, осигуряване на сигурност и изпълнение на транзакции

CORBA

# CASE STUDY

# CORBA: въведение

## ❖ Цел

- Използване на отворени системи базирани на **стандартни обектно-ориентирани интерфейси**
  - ✓ Хетерогенни хардуер, мрежи, операционни системи и програмни езици
- Използване на **произволен програмен език за реализиране на разпределени обекти**, комуникиращи помежду си

## ❖ Брокер за заявки към обекти (Object Request Broker)

- Локализиране и евентуално активиране на обекта, препращане на заявки от клиента към него и връщане на отговор от изпълнението на методите му

## ❖ Компоненти на CORBA (Common ORB Architecture)

- Език за дефиниране на интерфейси (IDL)
- Архитектура
- Външно представяне на данни (CDR)
  - ✓ Дефинира формат на съобщения
- Стандартен формат за референциите към отдалечените обекти
- CORBA услуги



OBJECT MANAGEMENT GROUP

# Нови концепции в CORBA RMI

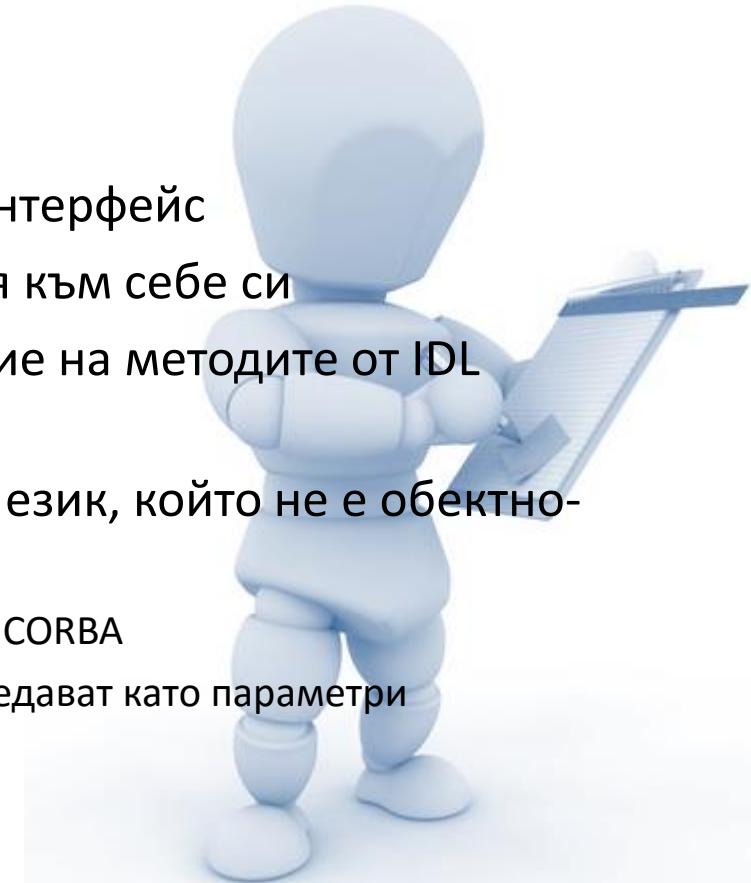


## ❖ Клиенти на CORBA обектите

- Програми, изпращащи заявки към отдалечени обекти и получаващи отговори
  - ✓ При Java RMI клиентите са обекти

## ❖ CORBA обект

- Отдалечен обект, реализиращ IDL интерфейс
- Предоставя отдалечена референция към себе си
- Притежава способност за изпълнение на методите от IDL интерфейса си
- Може да се реализира с програмен език, който не е обектно-ориентиран
  - ✓ Концепцията за клас не е представена в CORBA
  - ✓ Инстанции на класове не могат да се предават като параметри



# CORBA IDL: пример

```
struct Rectangle{  
    long width;  
    long height;  
    long x;  
    long y;  
};  
struct GraphicalObject {  
    string type;  
    Rectangle enclosing;  
    boolean isFilled;  
};  
interface Shape {  
    long getVersion();  
    GraphicalObject getAllState();  
};  
typedef sequence <Shape, 100> All;  
interface ShapeList {  
    exception FullException{ };  
    Shape newShape(in GraphicalObject g) raises (FullException);  
    All allShapes();  
    long getVersion();  
};
```



**GraphicalObject** е структура, докато при Java RMI е клас

// returns state of the GraphicalObject  
// returns sequence of remote object references

## CORBA IDL: характеристики, модули и интерфейси

```
module Whiteboard {  
    struct Rectangle{  
        ...};  
    struct GraphicalObject {  
        ...};  
    interface Shape {  
        ...};  
    typedef sequence <Shape, 100> All;  
    interface ShapeList {  
        ...};  
};
```

### ❖ Характеристики

- Осигурява средства за дефиниране на **модули, интерфейси, типове, атрибути и сигнатури на методи**
- Използва лексикалните правила на C++, добавяйки нови ключови думи като *any*, *attribute*, *in*, *out*, *inout*, *readonly* и *raises*
- Поддържа стандартната C++ предпроцесна обработка
- Използва граматика, подмножество на **ANSI C++** с допълнителни конструкции за сигнатури на методите

### ❖ IDL модули

- Позволява **логическо групиране** на интерфейси и други IDL дефиниции на типове
- Дефинира обхват на именуването

### ❖ IDL интерфейси

- Описва методите, които се предоставят CORBA обектите
- Клиентите на даден CORBA обект могат да се разработят въз основа на IDL интерфейса му

## ❖ Сигнатура на метод

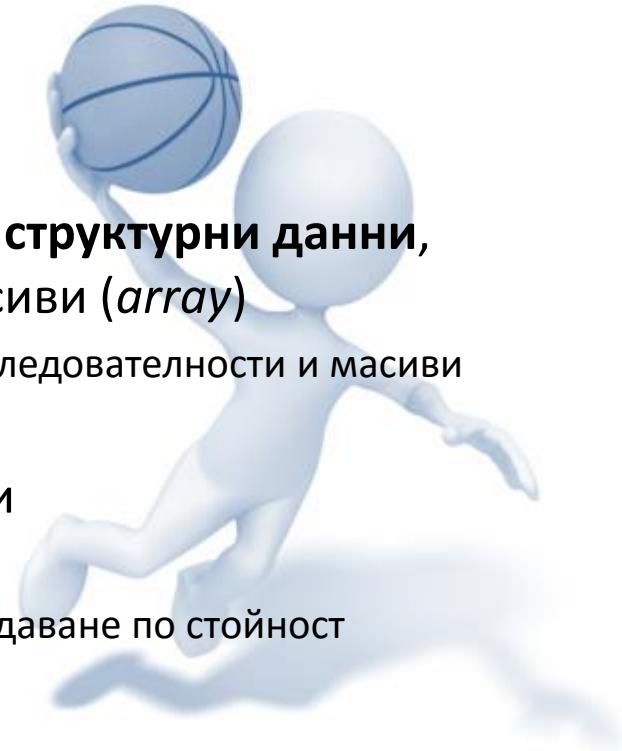
*[oneway] <return\_type> <method\_name>(parameter1, ..., parameterL)  
[raises(except1, ..., exceptN)] [context(name1, ..., nameM)];*

## ❖ Пример за сигнатура на метод

*void getPerson(in string name, out Person p);*

## ❖ Параметри на методи

- Видове параметри: *in*, *out*, *inout*
- Даннови типове: **примитивни типове и структурни данни**, дефинирани със структури (*struct*) и масиви (*array*)
  - ✓ Използване на *typedef* за дефиниране на последователности и масиви  
*typedef sequence <Shape, 100> All;*
- Семантика при предаване на параметри
  - ✓ CORBA обекти: предаване по референция
  - ✓ Примитивни типове и структурни данни: предаване по стойност
  - ✓ Пример: *allShapes()*



## ❖ Семантика на извикването

- Подразбираща се семантика е **at-most-once**
- Възможност за използване на семантика **maybe** с ключова дума **oneway**
  - ✓ Клиентското приложение не се блокира
  - ✓ Използва се при методи, които не връщат резултат

## ❖ Изключения

- Дефинират се в интерфейсите и се предизвикват от методите
- Потребителски дефинирани изключения (*raises*)
  - ✓ Дефиниране на **изключение без променливи**  
*exception FullException{ };*  
*Shape newShape(in GraphicalObject g) raises(FullException);*
  - ✓ Дефиниране на **изключение с променливи**, съхраняващи информация за изключението  
*exception FullException{GraphicalObject g};*
- Системни изключения
  - ✓ Представят проблеми със сървъра, комуникацията и клиентското приложение

# CORBA IDL: даннови типове

- ❖ Примитивни типове
  - *short* (16-bit), *long* (32-bit), *unsigned short*, *unsigned long*, *float* (32-bit),  
*double* (64-bit), *char*, *boolean* (TRUE, FALSE), *octet* (8-bit), и *any*
  - Дефиниране на константи с ключовата дума *const*
- ❖ Референции към CORBA обекти
  - Дефинират се от тип *Object*
- ❖ Структурни типове
  - Предават се по стойност
- ❖ Обекти, които не са CORBA обекти
  - Притежават атрибути и методи, които **не могат** да се извикват отдалечно
  - Предават се по стойност
    - ✓ При клиента се създава нов обект
  - Декларират с ключовата дума *valuetype*
    - ✓ Структура с допълнителни сигнатури на методи
  - Извикването на методите е локално, при което състоянието на обекта в клиентския процес е различно от това на сървъра
  - Трудни са за реализация, когато клиентът и сървърът използват различни езици

# CORBA IDL: структурни типове

Тип	Примери	Приложение
sequence	typedef sequence <Shape, 100> All; typedef sequence <Shape> All	<b>Последователност от променливи.</b> Възможно е специфициране на горна граница.
string	String name; typedef string<8> SmallString;	<b>Последователност от символи</b> , терминирана с null. Възможно е специфициране на горна граница.
array	typedef octet uniqueld[12]; typedef GraphicalObject GO[10][8]	<b>Многомерна последователност от елементи</b> с фиксирана дължина
record	struct GraphicalObject { string type; Rectangle enclosing; boolean isFilled; };	<b>Запис</b> , съдържащ група от свързани елементи
enumerated	enum Rand (Exp, Number, Name);	<b>Изброимо множество</b> , което съпоставя име на тип с малко множество от целочислени стойности
union	union Exp switch (Rand) { case Exp: string vote; case Number: long n; case Name: string s; };	<b>Обединение</b> , позволяващо подаване като аргумент на един даннов тип от множество

# CORBA IDL: атрибути и наследяване

## ❖ Атрибути

- Стойностите на атрибутите се достъпват с методи за четене и запис
- Съществуват атрибути, достъпни само за четене
- Пример за дефиниране на атрибут  
*readonly attribute string listname;*

## ❖ Наследяване

- Добавяне на нови типове, методи, атрибути и т.н.
- Предефиниране на типове, константи и изключения, но **не** и на методи
- Наследяване на повече от един интерфейс
  - ✓ Пример за множествено наследяване

```
interface A {};
interface B: A {};
interface C {};
interface Z : B, C {};
```
  - ✓ Дублираните имена на типове се използват като се задава обхватът им

```
interface B {}; → B::Q
interface C {}; → C::Q
```
- Всички IDL интерфейси наследяват типа Object
  - ✓ IDL операциите могат да приемат като аргумент и връщат като резултат референция към отдалечен обект от произволен тип

## ❖ Типови идентификатори

- Генерираат се от IDL компилатора за всеки тип в IDL интерфейса
- Пример: IDL тип за интерфейса *Shape*  
***IDL:Whiteboard/Shape:1.0***
- Структура на IDL типово име
  - ✓ IDL префикс
  - ✓ Име на тип
  - ✓ Номер на версия

## ❖ Директивата `pragma`

- Осигурява специфициране на допълнителни (не IDL) свойства за компонентите в IDL интерфейса
  - ✓ Пример: Използване на даден интерфейс само локално
- Пример за използване на директивата  
**#pragma version Whiteboard 2.3**

## ❖ Езикови съответствия между CORBA IDL и Java

- Съпоставяне на **примитивни типове** в CORBA IDL и Java
- Съпоставяне на **структурните типове struct, enum и union** в CORBA IDL на Java класове
- Съпоставяне на **последователности и масиви** в CORBA IDL на масиви в Java
- Съпоставяне на **изключения** в CORBA IDL на Java клас, осигуряващ променливи за полетата на изключенията и конструктори

## ❖ Съответствие в **семантиката при предаване на параметри** при CORBA IDL и Java

- CORBA IDL позволява връщане на няколко параметъра като резултат, а Java – само един
- Използване на *Holder* класове
  - ✓ IDL: `void getPerson(in string name, out Person p);`
  - ✓ Java: `void getPerson(String name, PersonHolder p);`

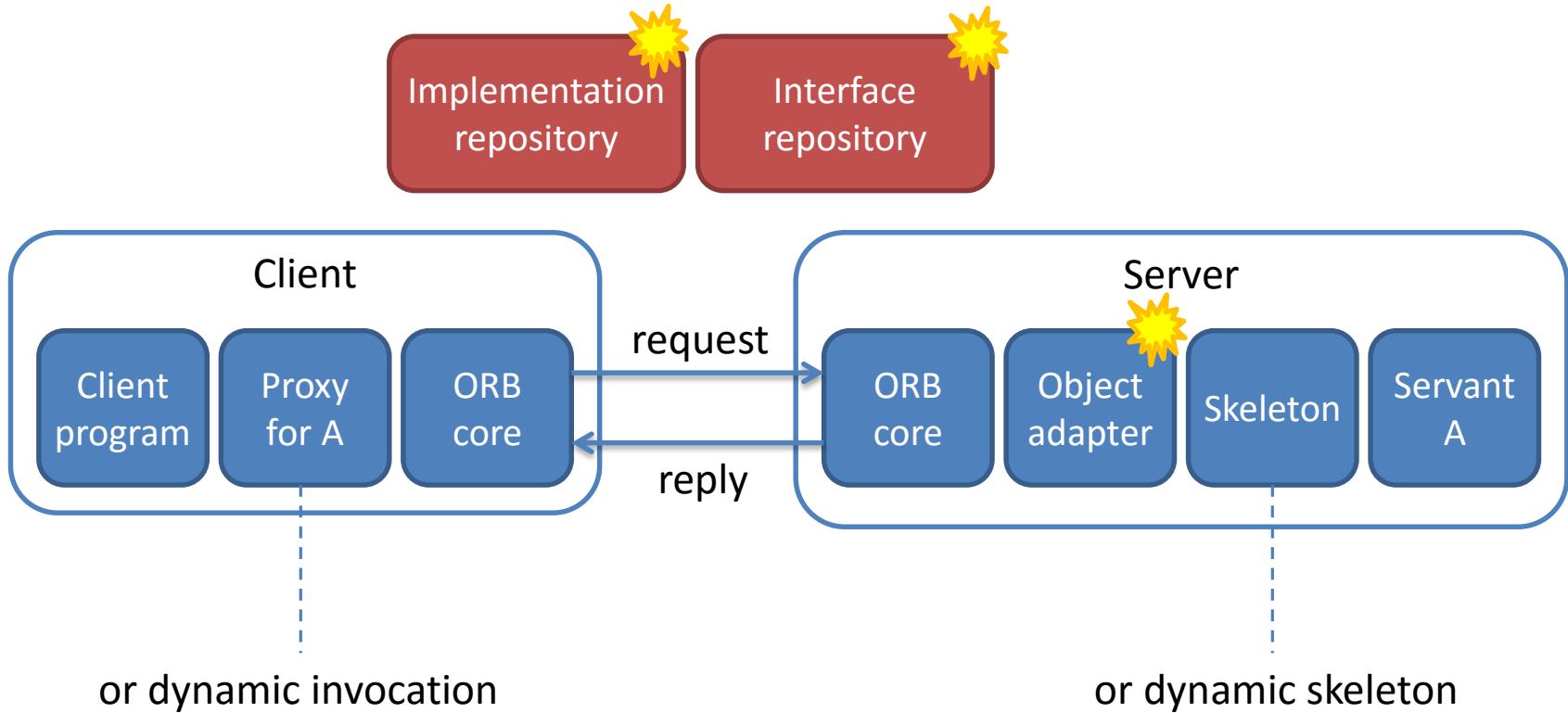
## ❖ Особености

- Обикновено се реализира при клиента
- В повечето случаи сървърът няма информация за типа на извикването – асинхронно или синхронно

## ❖ Възможности за реализация

- callback
  - ✓ Клиентът използва допълнителен параметър при извикване на метод, за да подаде референция на callback функция, която се извиква от сървъра при връщане на резултат
- polling
  - ✓ Сървърът връща обект от тип *valuetype*, който може да се използва за проверка или изчакване на резултат

# CORBA: архитектура



**Нови компоненти в архитектурата**

**Поддържа се статично и динамично извикване**

# CORBA компоненти: ORB core и Object adapter

## ❖ ORB core

- Изпълнява роля на комуникационен модул
- Операции, с които се **стартира и спира**
- Операции за **преобразуване** на отдалечени обектни референции в символни низове и обратно
- Операции за **осигуряване на списъци с аргументи** за заявки при динамично извикване

## ❖ Обектен адаптер (Object adapter)

- Създава отдалечени обектни референции за CORBA обекти
- Диспечеризира отдалечените извиквания към скелетона на релевантния сървант
- Активира и деактивира сърванти

## ❖ Име на обект

- Специфицира се от приложната програма или се генерира от обектния адаптер
- Участва във формирането на отдалечената референция заедно с името на обектния адаптер
- Използва се за регистриране на CORBA обект в таблица на отдалечените обекти
  - ✓ Таблицата съпоставя имената на CORBA обектите на техните сърванти

## ❖ Име на обектен адаптер

# CORBA: Portable Object Adapter (POA)

## ❖ Същност на POA стандарта

- Осигурява приложенията и сървантите да използват ORBs, разработени от различни разработчици
- **Стандартизира** класовете на скелетоните и взаимодействието между адаптерите на обекти и сървантите

## ❖ Типове време на живот за CORBA обекти

- Ограничено време на живот в рамките на процеса, в който се инстанциира сърванта (временни референции)
- Време на живот, обхващащо инстанцирането на сърванти в множество процеси (перsistентни референции)

## ❖ Предназначение

- Осигурява **прозрачност при инстанцирането** на CORBA обекти
- **Разделя създаването** на CORBA обекта и сърванта
- Дефиниране на **политики** за POA
  - ✓ Пример: Осигуряване на отделни нишки за всяко извикване, създаване на постоянни или временни референции към обекти, на отделен сървант за всеки CORBA обект

## ❖ Реализация на ORB core и POA в CORBA

- Предоставят функционалност под формата на “псевдо-обекти”
- Притежават IDL интерфейси и са реализирани като библиотеки

# CORBA: Skeleton, proxy/stub, implementation repository

## ❖ Skeleton

- Клас, който се генерира от IDL компилатора, като се използва програмния език на сървъра

## ❖ Proxy/stub

- Прокси клас (при обектно-ориентираните езици) или множество от стъб процедури (при процедурните езици), които се генерират от IDL компилатора, като се използва програмния език на клиента

## ❖ Implementation repository

- Съхранява **съответствие** между имената на обектните адаптери и местоположението на файловете, съхраняващи реализациите на обектите
- **Регистрацията** в хранилището става при инсталација на сървърната програма
- При **активиране** на обектната реализация в хранилището се добавя информация за име на хост и номер на порт
- Допуска съхраняване на допълнителна информация

Implementation repository entry:

Object adapter name	Pathname of object implementation	Host name and port number of the server
---------------------	-----------------------------------	---

# CORBA: Interface repository и Dynamic invocation

## ❖ Interface repository

- Осигурява информация на клиентите и сървърите за регистрираните IDL интерфейси
  - ✓ Имена на методи, имена и типове на аргументи и изключения
- Съхранява съответствие между типовите идентификатори (repositoryID) и интерфейсите
  - ✓ Типовите идентификатори се генерира от IDL компилатора
- Използва се при динамично свързване с CORBA обекти
  - ✓ Отдалечената референция включва информация за типовия идентификатор

## ❖ Dynamic invocation interface

- Осигурява на клиентите динамично да извикват методи на CORBA обекти
- Използва се, когато създаването на прокси не е подходящо
- Клиентът получава информация за методите от хранилището за интерфейси

## ❖ Dynamic skeleton

- Използва се, когато даден интерфейс на CORBA обект не е известен преди компилирането на сървъра

## ❖ Наследен софтуер

- Възможност за предоставяне на функционалност чрез интерфейс на CORBA обект

# CORBA: отдалечени референции към обекти

## ❖ Формат на отдалечена референция към обект (Interoperable Object Reference)

- Идентификатор на IDL интерфейсен тип
- Транспортен протокол и информация за идентифициране на сървър
- Обектен ключ, състоящ се от име на обектен адаптер в сървъра и име на CORBA обект

## ❖ Типове референции

- Временни
  - ✓ Съществува в рамките на процеса с инстанция на обекта
  - ✓ Съдържа адреса на сървъра, хостващ обекта
- Постоянни
  - ✓ Съществува между активациите на обекта
  - ✓ Съдържа адреса на хранилището за реализации, в който е регистриран обекта

IDL interface type ID

Protocol and address detail

Object key

Interface repository identifier or type	IIOP	Host domain name	Port number	Adapter name	Object name
---	------	------------------	-------------	--------------	-------------

# CORBA: локализиране на сървант

## Временна отдалечена референция към обект

Сървърният брокер на обекти получава заявка, съдържаща име на обектен адаптер и име на обект

Името на обектния адаптер се използва от брокера за откриването му

Обектният адаптер използва името на обекта за откриване на сървANTA

## Постоянна отдалечена референция към обект

Хранилището за реализации получава заявка, съдържаща име на обектен адаптер

CORBA обектът се активира на хост адреса, записан в хранилището за реализации

Хранилището за реализации връща на клиентския брокер информация за адреса на обекта, включваща име на обектен адаптер и име на обект

Обектният адаптер използва името на обекта за откриване на сървANTA

Сървърният брокер използва името на адаптера за откриването му

## ❖ Naming service

- Осигурява откриване на CORBA обект по име

## ❖ Trading service

- Осигурява откриване на CORBA обект по атрибут

## ❖ Event service

- Осигурява изпращане на нотификации от обекти до абонати с помощта на CORBA RMI

## ❖ Notification service

- Осигурява допълнителни възможности при нотифициране (дефиниране на филтри, надеждност и подреждане в канала за събития)

## ❖ Security service

- Осигурява механизми за сигурност като автентикация, контрол на достъпа, защитена комуникация и др.

## ❖ Transaction service

- Осигурява създаване на плоски и вложени транзакции

## ❖ Concurrency control service

- Управлява конкурентния достъп до обектите, използвайки заключвания

## ❖ Persistent state service

- Осигурява персистентно хранилище за CORBA обекти, от което се възстановява състоянието им

## ❖ Lifecycle service

- Дефинира конвенции за създаване, изтриване, копиране и преместване на CORBA обекти
  - ✓ Пример: използване на фабрики за създаване на обекти

# CORBA пример: функционалност и елементи

## ❖ Функционалност на примера

- Създаване на клиентска и сървърна програми, използващи IDL интерфейсите *Shape* и *ShapeList*

## ❖ Елементи, генериирани от *idl/j* компилатора

- 2 Java интерфейси за всеки един IDL интерфейс
- Скелетони за всеки IDL интерфейс (завършват с POA)
- Прокси класове или клиентски стъбове за всеки IDL интерфейс (завършват с Stub)
- Java класове, съответстващи на структурните данни в IDL интерфейсите
- Класове, наречени Helper и Holder, за всеки тип, дефиниран в IDL интерфейс

```
public interface ShapeListOperations {  
    Shape newShape(GraphicalObject g) throws ShapeListPackage.FullException;  
    Shape[] allShapes();  
    int getVersion();  
}  
  
public interface ShapeList extends ShapeListOperations, org.omg.CORBA.Object,  
    org.omg.CORBA.portable.IDLEntity { }
```



# CORBA пример: Сървърна програма

## ❖ Функционалност

- Създава инстанция на сървантен клас и го регистрира заедно с обектния адаптер
- Обектният адаптер създава CORBA обект от инстанцията, асоциирайки го с отдалечена референция

## ❖ Реализация

- Реализация на интерфейсите *Shape* и *ShapeList* като сървантни класове *ShapeServant* и *ShapeListServant*
  - ✓ Всеки сървантен клас реализира методи на IDL интерфейс, използвайки сигнатурите, дефинирани в еквивалентния Java интерфейс
- Реализация на сървърен клас
  - ✓ В метода *main* се създава и инициализира обектен брокер с име *orb*, получава се референция към обектен адаптер с име *rootpoa* и се активира мениджъра на адаптера
  - ✓ Създава се инстанция на класа *ShapeListServant* с име *SLSRef*, използвайки референция към обектния адаптер *rootpoa*
  - ✓ Създава се CORBA обект посредством регистриране на инстанцията *SLSRef* в обектния адаптер *rootpoa*
  - ✓ Сървърът се регистрира с услугата за именуване (*Naming service*)
  - ✓ Получава се контекст за именуване с *NamingContextHelper*
  - ✓ Създава се обект от тип *NameComponent* за представяне на името на CORBA обекта
  - ✓ Името и отдалечената референция на CORBA обекта се регистрират в контекста с помощта на метода *rebind*

# Java клас ShapeListServant

```
import org.omg.CORBA.*;           import org.omg.PortableServer.POA;
class ShapeListServant extends ShapeListPOA {                                //extends the skeleton class
    private POA theRootpoa;
    private Shape theList[];
    private int version;
    private static int n=0;
    public ShapeListServant(POA rootpoa){
        theRootpoa = rootpoa;
    }
    public Shape newShape(GraphicalObject g)
        throws ShapeListPackage.FullException {                                //factory method (creates objects)
        version++;
        Shape s = null;
        ShapeServant shapeRef = new ShapeServant( g, version);
        try {
            org.omg.CORBA.Object ref = theRootpoa.servant_to_reference(shapeRef); //registration with POA
            s = ShapeHelper.narrow(ref);
        } catch (Exception e) {}
        if(n >=100) throw new ShapeListPackage.FullException();
        theList[n++] = s;
        return s;
    }
    public Shape[] allShapes(){ ... }
    public int getVersion() { ... }
}
```



# Java клас ShapeListServer

```
import org.omg.CosNaming.*; import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
public class ShapeListServer {
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null); //creates and idealizes ORB
            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate(); //activate POAManager
            //servant instance (Java object)
            ShapeListServant SLSRef = new ShapeListServant(rootpoa);
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(SLSRef); //CORBA object
            ShapeList SLRef = ShapeListHelper.narrow(ref);
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef); //get naming context
            //create object nc representing a name in CORBA passing name and kind parameters
            NameComponent nc = new NameComponent("ShapeList", "");
            NameComponent path[] = {nc}; //defines a path consisting of a single name
            ncRef.rebind(path, SLRef); //register the name and the remote object reference in the context
            orb.run();
        } catch (Exception e) { ... }
    }
}
```



# CORBA пример: Клиентска програма

## ❖ Реализация

- Създаване и инициализиране на обектен брокер
- Получаване на референция към отдалечения *ShapeList* обект от услугат за именуване
- Извикване на метод *allShapes* за получаване на последователност от референции към отдалечени обекти
- Извикване на метод *getAllState* и получаване на инстанция на класа *GraphicalObject*
- Прихващане на CORBA изключения, породени от разпределената обработка

# Java клас ShapeListClient

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class ShapeListClient{
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null); //creates and idealizes ORB
            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent("ShapeList", "");
            NameComponent path [] = { nc };
            //get reference to remote ShapeList object using resolve method
            ShapeList shapeListRef = ShapeListHelper.narrow(ncRef.resolve(path));
            Shape[] sList = shapeListRef.allShapes(); //get all Shapes on the server
            GraphicalObject g = sList[0].getAllState(); //get the state of the first Shape
        } catch(org.omg.CORBA.SystemException e) {...} //catch exceptions
    }
}
```



# Ограничения на отдалечените обекти (1-2)

## ❖ Имплицитни зависимости

- Скриване на вътрешното поведение на обекта и липса на информация за разпределените системни услуги в интерфейса му
- **Изискване:** Необходимост от специфициране в интерфейса на зависимостите, които обекта има по отношение на разпределената архитектура

## ❖ Взаимодействие с разпределената платформа

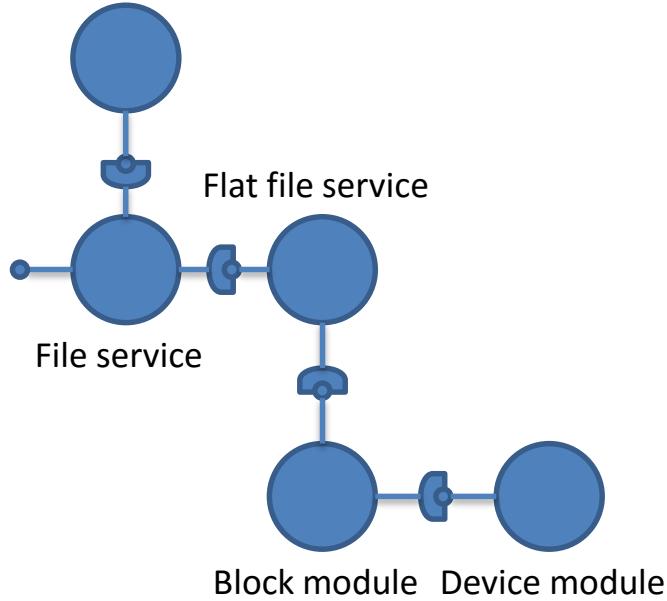
- Необходимост от допълнителни действия, свързани със създаване на обектен адаптер и обектен брокер, управление на обектни референции, реализация на политики за активация и деактивация, и др.
- **Изискване:** Отделяне на реализацията на обекта от операциите, свързани с прилежащата разпределена платформа

## Ограничения на отдалечените обекти (2-2)

- ❖ Липса на поддръжка от гледна точка на разпределената обработка
  - Необходимост от познаване на разпределените системни услуги
  - Смесване на приложния код с извикване на разпределените услуги на прилежащата платформа
  - **Изискване:** Скриване на взаимодействието със системните услуги от програмиста
- ❖ Липса на поддръжка за внедряване
  - Ръчно инсталиране на обектите, водещо до възможност за допускане на грешки
  - Необходимост от активиране и свързване с други обекти
  - **Изискване:** Осигуряване на поддръжка за внедряване от разпределената платформа така, че разпределеният софтуер да се инсталира по начин, аналогичен на локалния за дадена машина софтуер

## Компоненти: концепция

Directory service



### ❖ Компонент: дефиниция

- Композитен модул със специфициран под формата на договор **интерфейс** и наличие единствено на **експлицитни контекстни зависимости**

### ❖ Елементи на договора

- Множество от осигурени интерфейси
- Множество от необходими интерфейси

### ❖ Софтуерна архитектура

- Компонентна конфигурация, при която всеки **необходим** интерфейс е свързан с **осигурен** от друг компонент интерфейс

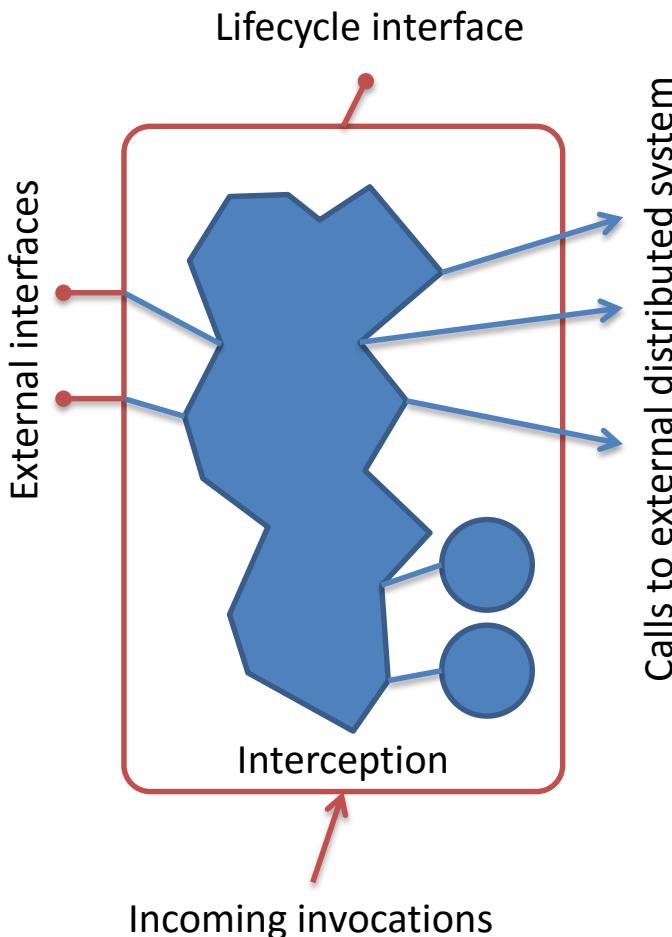
### ❖ Типове интерфейси

- Интерфейси, поддържащи **отдалечено извикване на методи**
- Интерфейси, поддържащи **разпределени събития**

### ❖ Композиция

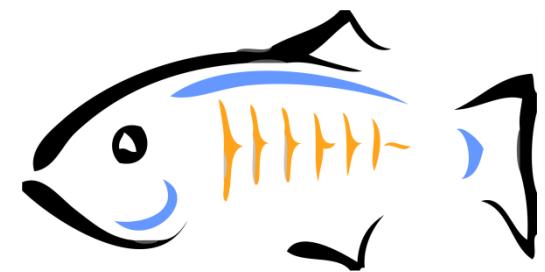
- Комбиниране на функционалност от няколко компонента с цел предоставяне на комплексна софтуерна услуга
- Интегриране на компоненти, разработени от трети страни

# Компоненти и разпределени системи



- ❖ Общ шаблон за изграждане на разпределени приложения
  - Клиент (front-end)
  - Контейнер, хостващ един или няколко компонента, реализиращи приложение или бизнес логика
  - Системни услуги, управляващи данни в персистентно хранилище
- ❖ Контейнер
  - Осигурява управляема хостваща среда за компонентите от страна на сървъра
  - Гарантира разделяне на приложната логика на компонента от функциите на разпределената среда
  - Скрива директния достъп до компонента и приема подходящи действия, за да осигури поддръжка на разпределеното приложение
- ❖ Приложен сървър
  - Изпълнява ролята на контейнер

# Приложни сървъри



# Поддръжка при внедряване

## ❖ Внедряване на компонентни конфигурации

- Пакетиране на компонентите и взаимовръзките между тях с **дескриптори за внедряване**, описващи конфигурационните параметри
- Интерпретиране на дескрипторите за внедряване от контейнерите с цел **определяне на политиките**, свързани с разпределените системни услуги

## ❖ Предназначение на дескрипторите за внедряване

- Осигуряват коректно свързване на компонентите при използване на подходящи протоколи
- Коректно конфигуриране на разпределената платформа с цел поддръжка на компонентната конфигурация
- Инициализиране на разпределените системни услуги с цел осигуряване на определено ниво на сигурност, поддръжка на транзакции и т.н.

Enterprise Java Beans

# CASE STUDY



ENTERPRISE  
Java Beans

# Enterprise Java Beans (EJB)

## ❖ Същност на EJB

- Спецификация на сървърна компонентна архитектура и базов елемент на Java EE, множество от спецификации за клиент-сървър програмиране
- Поддържа разработване на приложения, при които **голям брой клиенти комуникират с набор от услуги**, реализирани като компоненти или конфигурация от компоненти

## ❖ Компоненти на EJB (beans)

- Реализират приложна (бизнес) логика
- Отделяне на бизнес логиката от съхраняване на състоянието

## ❖ Управление в EJB

- Контейнерът поддържа ключови разпределени услуги като транзакции, сигурност и др.
- Container-managed EJB
  - ✓ Извикванията към разпределените услуги са от контейнера
- Bean-managed EJB
  - ✓ Разработчикът управлява извикването на операциите на разпределените услуги

# Роли в EJB спецификацията

- ❖ Bean provider
  - Разработва приложната логика в компоненти
- ❖ Application assembler
  - Асемблира компонентите в приложни конфигурации
- ❖ Deployer
  - Подпомага внедряването в определена оперативна среда
- ❖ Service provider
  - Специалист в областта на базовите разпределени системни услуги, който осигурява определено ниво на тяхната поддръжка
- ❖ Persistent provider
  - Специалист, управляващ асоциирането на персистентни данни с прилежащите бази от данни
- ❖ Container provider
  - Конфигурира контейнерите с необходимото ниво на системна поддръжка за транзакции, сигурност и др.
- ❖ System administrator
  - Наблюдава системата в режим на изпълнение и извършва настройки

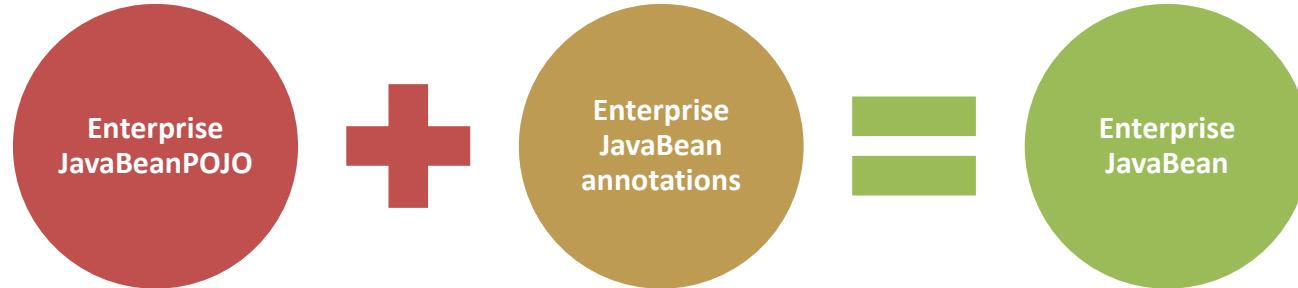
## ❖ EJB bean

- Компонент, предлагащ един или повече бизнес интерфейси, които са достъпни отдалечно или локално
- Състои се от множество интерфейси и клас, който ги реализира

## ❖ Типове компоненти в EJB 3.0

- Session bean
  - ✓ Компонент, реализиращ определена задача в приложната логика на дадена услуга
  - ✓ Съществува в рамките на услугата и управлява комуникацията с клиента в рамките на дадена сесия
  - ✓ Състоянието може да се запазва (stateful bean) или да не се запазва (stateless bean)
- Message-driven bean
  - ✓ Поддържа индиректна комуникация
  - ✓ Използва опашки със съобщения или теми, реализирани върху JMS

# Реализация на EJB



- ❖ **Enterprise JavaBeanPOJO (Plain Old Java Object)**
  - Реализира единствено приложна логика на Java
- ❖ **Enterprise JavaBean annotations**
  - Осигуряват коректно поведение на компонентите в EJB контекста
  - Механизъм за асоцииране на метаданни с пакети, класове, методи, параметри и променливи
  - Примери за анотации, обозначаващи компонент с определен тип

```
@Stateful public class eShop implements Orders { ... }
```

```
@Stateless public class CalculatorBean implements Calculator { ... }
```

```
@Message-driven public class SharePrice implements MessageListener { ... }
```
  - Примери за анотации, обозначаващи интерфейс от определен тип

```
@Remote public interface Orders { ... }
```

```
@Local public interface Calculator { ... }
```

# EJB транзакции

## ❖ Същност на транзакциите

- Осигуряват, че всички обекти, управлявани от даден сървър ще се запазят в консистентно състояние при конкурентен достъп или срив на сървъра
- Гарантират атомарно изпълнение на последователност от операции
- Пример: eShop

## ❖ Управление на транзакции в EJB

- Необходимост от ясна спецификация на последователността от операции в транзакцията
- Прилагат се и при двата типа компоненти – session bean и message-driven bean
- Транзакции, управлявани от компонента
  - ✓ Експлицитно идентифициране от разработчика на последователността от операции в транзакцията  
*@TransactionManagement(BEAN)*
- Транзакции управлявани от контейнера
  - ✓ Използване на подходяща анотация за метод, участващ в транзакция  
*@TransactionManagement(CONTAINER)*

# EJB транзакции: примери

## ❖ Транзакция, управляема от компонент

```
@Stateful  
@TransactionManagement(BEAN)  
public class eShop implements Orders {  
    @Resource javax.transaction.UserTransaction ut;  
    Public void MakeOrder (...) {  
        ut.begin();  
        ...  
        ut.end();  
    }  
}
```

## ❖ Транзакция, управляема от контейнер

```
@Stateful public eShop implements Orders {  
    ...  
    @TransactionAttribute(REQUIRED)  
    public void MakeOrder (...) {  
        ...  
    }  
}
```

## ❖ Инжектиране на зависимости (EJB dependency injection)

- Компонентът заявява зависимост от системна услуга посредством анотация, а контейнерът я осигурява
- **@Resource:** зависимост от обект, реализиращ интерфейс **UserTransaction**

# Атрибути за дефиниране на транзакции

Атрибут	Политика
REQUIRED	Ако клиентът е асоцииран със стартирана транзакция, то изпълнението е в нея; в противен случай се стартира нова транзакция
REQUIRES_NEW	Винаги се стартира нова транзакция
SUPPORTS	Ако клиентът е асоцииран със стартирана транзакция, то изпълнението е в нея; в противен случай изпълнението протича без поддръжка на транзакция
NOT_SUPPORTED	Ако клиентът извиква метод, участващ в транзакция, то транзакцията се преустановява преди извикването на метода и се възстановява след това, т.е. извикваният метод се изключва от транзакцията
MANDATORY	Методът трябва да се извика в рамките на клиентска транзакция; в противен случай се генерира изключение
NEVER	Методът не трябва да се извика в клиентска транзакция; в противен случай се генерира изключение

# Промяна на подразбиращо се поведение

Извикване на методи  
от бизнес интерфейса

Събития, свързани с  
жизнения цикъл

# EJB interception: методи

## ❖ Същност

- Асоцииране на действие или група от действия с извикване на метод (session bean) или настъпване на събитие (message-driven bean) от бизнес интерфейса
- Осигурява имплицитно управление от страна на разработчика
- Пример: осигуряване на лог за операциите от бизнес интерфейс

## ❖ Реализация

- `@Interceptors`
  - ✓ Реализира се на ниво клас
- `@AroundInvoke`
  - ✓ Реализира се на ниво метод

## ❖ Пример

```
@Stateful
public class eShop implements Orders {
    public void MakeOrder (...) { ... }

    @AroundInvoke
    public Object log(invocationContext ctx) throws Exception {
        System.out.println("Invoked method: " + ctx.getMethod().getName());
        return invocationContext .proceed();
    }
}
```



# EJB: контекст на извикване

- ❖ Синтаксис на интерсепторен метод  
*Object <methodName>(javax.ejb.InvocationContext)*
- ❖ Предназначение на **InvocationContext**
  - ❖ Осигурява метаданни за извикването и ограничени възможности за промяна на параметрите на метода преди изпълнението му

Сигнатура на метод	Приложение
public Object getTarget()	Връща инстанция на компонента, асоцииран с входящото извикване или събитие
public Method getMethod()	Връща извикания метод
public Object[] getParameters()	Връща множество от параметри, асоциирани с извиквания метод
public void setParameters(Object[] params)	Позволява множеството от параметри да бъде променено от метода, реализиращ интерсепцията
public Object proceed() throws Exception	Осигурява продължаване на изпълнението със следващия интерсептен метод или с текущо извиквания метод

# EJB interception: събития, свързани с жизнения цикъл

## ❖ Същност

- Асоцииране на интерсептен метод със събитията за създаване или изтриване на компоненти

## ❖ Реализация

- *@PostConstruct* и *@PreDestroy*
- *@PostActive* и *@PrePassivate*

## ❖ Пример

```
@Stateful
public class eShop implements Orders {
    public void MakeOrder (...) { ... }

    ...
    @PreDestroy void TidyUp() { ... }
}
```



Оsvобождаване  
на ресурси и  
запис на данни в  
база от данни



*The open source community for infrastructure software*

Fractal

# CASE STUDY

# Fractal: концепция

## ❖ Същност на програмния модел Fractal

- Компонентно базиран модел за **програмиране с интерфейси**
- Позволява **разделяне на интерфейса от реализацията**
- Поддържа **експлицитно представяне** на софтуерната архитектура без да допуска имплицитни зависимости
- Осигурява реализация на приложения, които са **конфигурируеми и реконфигурируеми** в режим на изпълнение по отношение на текущата оперативна среда и изисквания

## ❖ Езици, поддържащи програмния модел на Fractal

- Julia и AOKell (Java базирани)
- Cecilia и Think (C базирани)
- FracNet (.NET базиран)
- FracTalk (SmallTalk базиран)
- Julio (Python базиран)

# Fractal: компонентен модел

## ❖ Типове интерфейси

- Сървърни интерфейси
  - ✓ Поддържат входящи оперативни извиквания
- Клиентски интерфейси
  - ✓ Поддържат изходящи извиквания (еквивалентни са на необходимите интерфейси)

## ❖ Типове свързване за осигуряване на композиция

- Примитивно свързване (primitive binding)
  - ✓ Директно съответствие между клиентски и сървърен интерфейс в рамките на едно и също адресно пространство
- Съставно свързване (composite binding)
  - ✓ Реализация на комуникация между множество интерфейси, обикновено разположени на различни машини

## ❖ Предимства, осигурени от съставното свързване

- Постигане на цялостна конфигурируемост на системата
  - ✓ Взаимодействие с произволна комуникационна парадигма, включително разработване и използване на нови
- Възможност за реконфигуриране на системата в режим на изпълнение
  - ✓ Добавяне на механизми за сигурност или осигуряване на скалируемост

# Fractal: йерархична композиция

## ❖ Същност

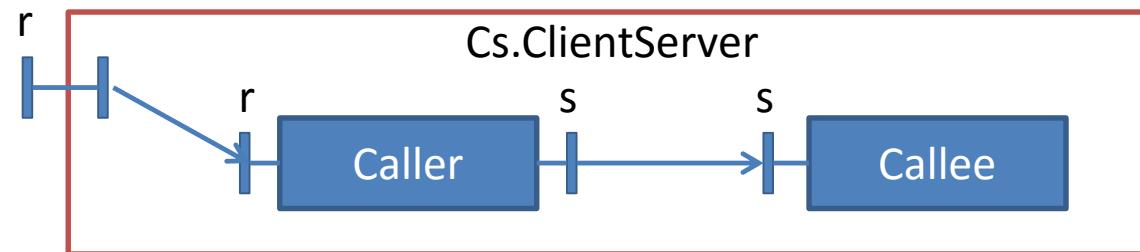
- Изграждане на приложения, при които компонентите са композирани в подкомпоненти, а подкомпонентите от своя страна също могат да са съставни

## ❖ Език за описание на Fractal архитектура

- Fractal Architectural Description Language (FADL)

## ❖ Пример: компонент с два подкомпонента

```
<definition name="cs.ClientServer">
    <interface name="r" role="server" signature="java.lang.Runnable"/>
    <component name="caller" definition="hw.CallerImpl"/>
    <component name="callee" definition="hw.CalleeImpl"/>
    <binding client="this.r" server="caller.r"/>
    <binding client="caller.s" server="callee.s"/>
</definition>
```



# Fractal: структура на компонент

## ❖ Мембрана

- Дефинира управлението на компонента посредством контролери и съдържанието му

## ❖ Съдържание

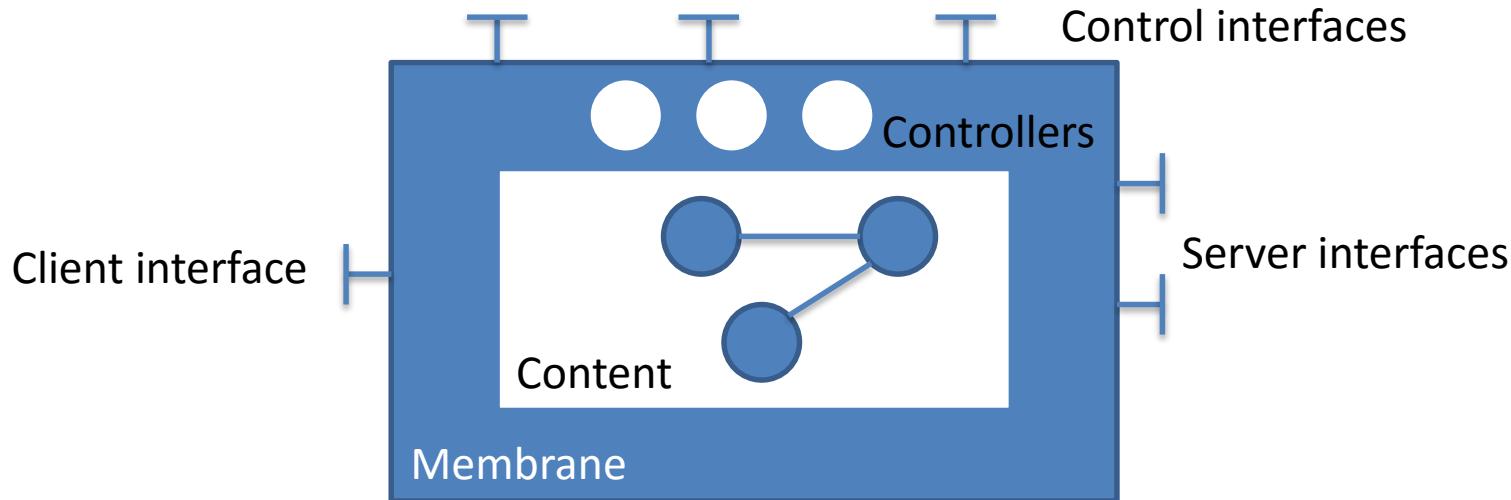
- Множество от подкомпоненти, изграждащи архитектурата

## ❖ Контролер

- Осигурява свойства за управление на компонента

## ❖ Типове интерфейси

- Вътрешни, които са видими само за компонентите от съдържанието
- Външни, които са видими за всички компоненти



# Приложения на контролерите

## ❖ Управление на жизнения цикъл на компонента

- Операции за активация и деактивация: *suspend*, *resume* и *checkpoint*
  - ✓ Динамична подмяна на сървър по време на изпълнение (извикване на *suspend*, замяна на callee компонента с нов и извикване на *resume*)
- Пример: *LifeCycleController* с методи *startFc*, *stopFc* и *getFcState*

## ❖ Осигуряване на рефлексия

- Динамично откриване на интерфейси, асоциирани с компонента, и обхождане на архитектурата на съставен компонент
- Реализира се от интерфейсите Component и ContentController

## ❖ Осигуряване на интерсепция

- Функционалността е подобна на предоставената в EJB

# Fractal: интерфейсите Component и ContentController

## ❖ Предназначение

- *Component* : Динамично откриване на интерфейси
- *ContentController* : Обхождане на архитектурата на съставен компонент и реконфигурация

```
public interface Component {  
    Object[] getFcInterfaces();  
    Object getFcInterface(String iftName);  
    Type getFcType();  
}  
  
public interface ContentController {  
    Object[] getFcInternalInterfaces();  
    Object getFcInterfaceInterface(String iftName);  
    Component[] getFcSubComponents();  
    void addFcSubComponent(Component c);  
    void removeFcSubComponent(Component c);  
}
```







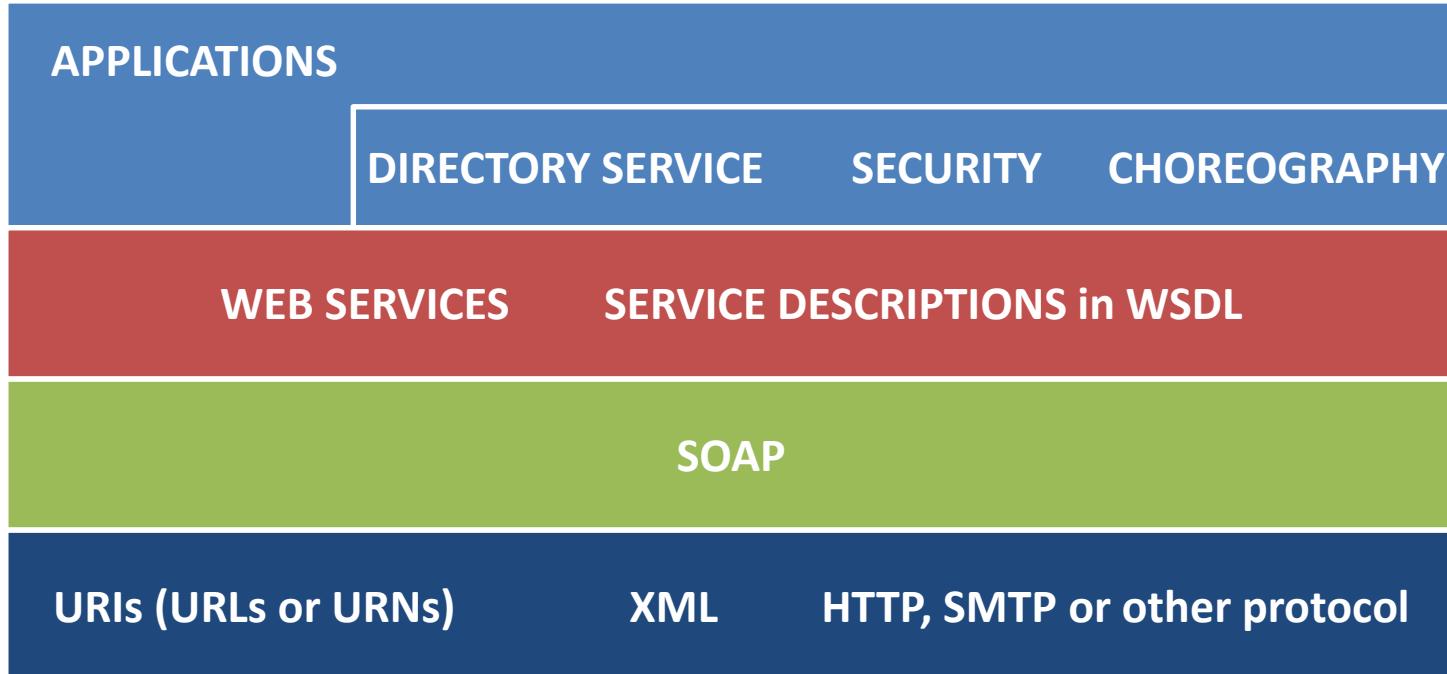
## УЕБ УСЛУГИ

доц. д-р Десислава Петрова-Антонова

# Съдържание

- ❖ Инфраструктура и компоненти на уеб услугите
- ❖ Комуникация с уеб услуги
- ❖ Описание на уеб услугите
- ❖ Директорийна услуга за използване на уеб услуги
- ❖ XML сигурност
- ❖ Координация на уеб услуги
- ❖ Приложение на уеб услугите

# Инфраструктура и компоненти на уеб услуги



# Инфраструктура и компоненти на уеб услугите

## ❖ XML

- Осигурява механизъм за представяне на данните и маршализиране на съобщенията, обменяни между клиентите и уеб услугите

## ❖ SOAP протокол

- Специфицира правилата за пакетиране на съобщенията с XML и изпращането им с HTTP или друг транспортен протокол

## ❖ URI

- Идентифицира уеб услуга

## ❖ WSDL

- Специфицира интерфейса на уеб услугата и допълнителна информация за начина на комуникация с нея и местоположението ѝ

## ❖ Директорийна услуга

- Осигурява възможност за търсене на уеб услуги от клиентите

## ❖ Сигурност

- Документи или части от документи могат да се подписват и криптират

## ❖ Хореография

- Осигурява координиране на изпълнението на операциите на няколко уеб услуги

# Уеб услуги: обща концепция

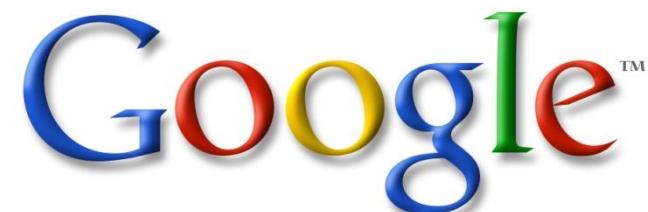
## ❖ Интерфейс на уеб услуга

- Колекция от операции, които могат да бъдат използвани от клиенти през интернет

## ❖ Комуникация

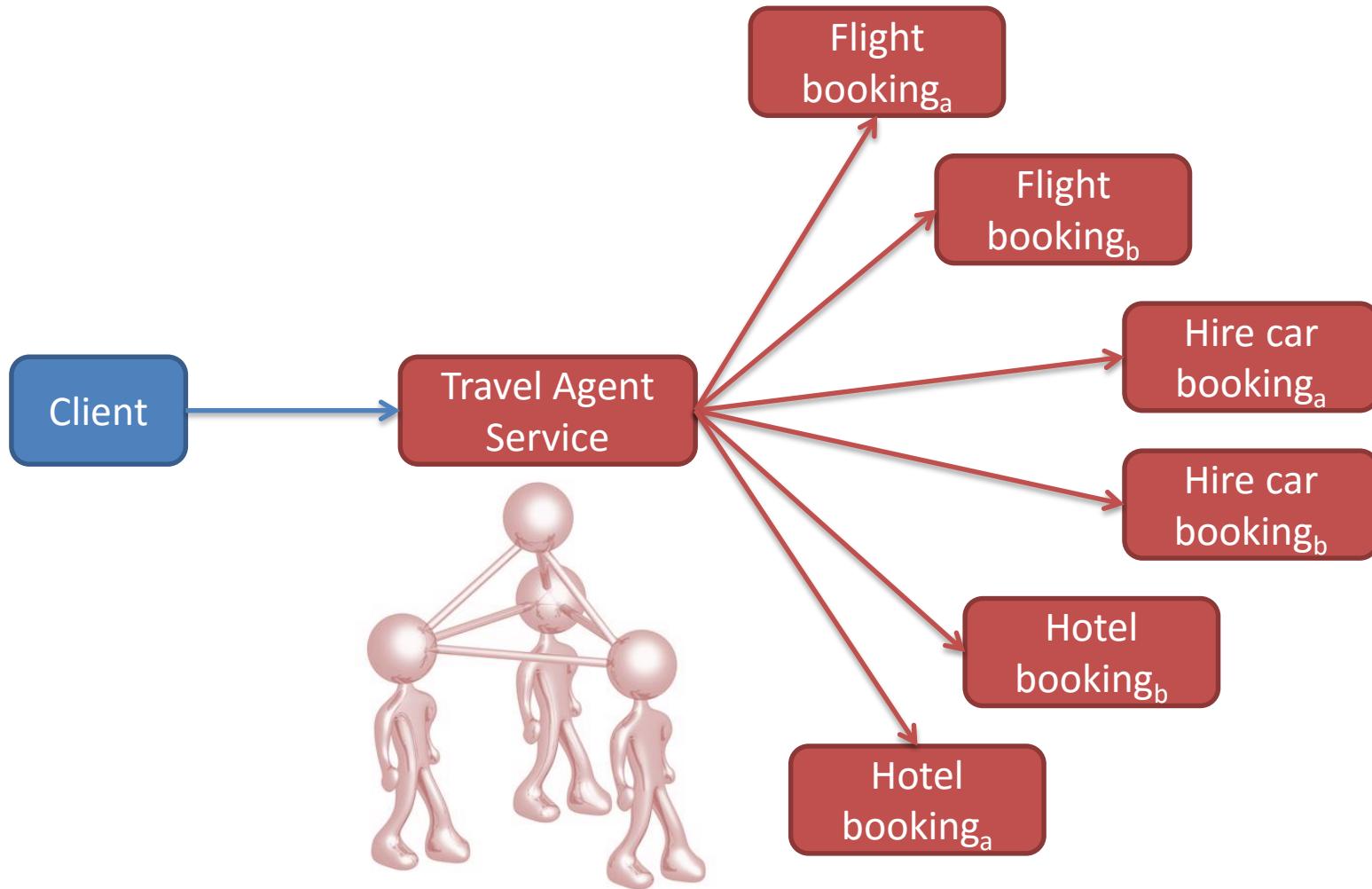
- SOAP подход
- REST подход

## ❖ Комерсиални уеб услуги



# Комбинация на уеб услуги

- ❖ Осигуряване на нова функционалност чрез комбиниране на операциите на няколко уеб услуги



# Комуникационни шаблони

## ❖ Асинхронна комуникация

- Обмяна на съобщения, при което изпращането на заявка не изисква незабавен отговор
- Пример: Извършване на резервация посредством обмяна на документи

## ❖ Синхронна комуникация

- Използване на протокол от тип заявка-отговор
- Пример: проверка на кредитна карта

## ❖ Комуникация, базирана на събития

- Получаване на съобщения при настъпване на събития
- Пример: Клиентите на директорийната услуга могат да се абонират за получаване на съобщения при добавяне на нова услуга

## ❖ Независимост на уеб услугите от програмната парадигма

# Слаба свързаност

## ❖ Слаба свързаност в контекста на уеб услугите

- Минимизиране на зависимостите между уеб услугите с цел осигуряване на гъвкава прилежаща архитектура
- Цел: комбиниране на функционалността на уеб услугите

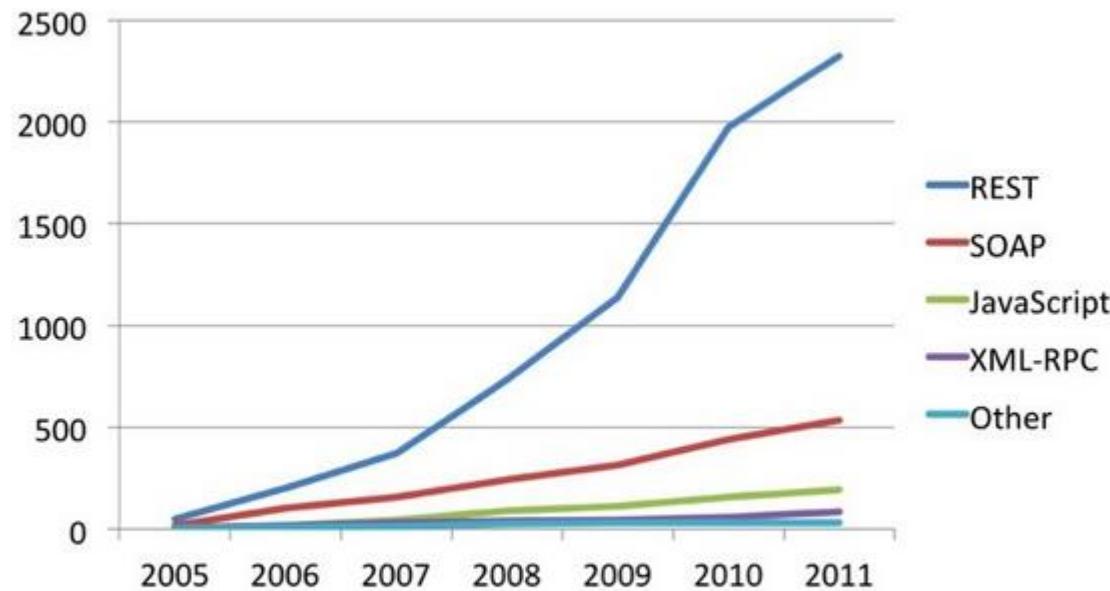
## ❖ Осигуряване на слаба свързаност

- Разделяне на интерфейса от реализацията на уеб услугата (програмиране с интерфейси)
  - ✓ Осигуряване на хетерогенност от гледна точка на програмен език и платформа
- Тенденция за създаване на опростени интерфейси в разпределена среда (REST уеб услуги)
  - ✓ Акцентиране върху данните, а не върху операциите
- Възможност за използване на различни комуникационни парадигми
  - ✓ Комуникационната парадигма (синхронна, асинхронна, индиректна) определя степента на свързаност



# Предпочтение за опростен интерфейс

REST vs. SOAP: Simplicity wins again



*Distribution of API protocols and styles*

*Based on directory of 3,200 web APIs listed at ProgrammableWeb, May 2011*

# Съобщения и референции към услуги

## ❖ Представяне на съобщенията

- Текстово представяне, базирано на XML
- Предимства
  - ✓ Осигуряване на формат, разбираем от човек
  - ✓ Разширяемост и транспортиране на данни от произволен тип
- Недостатъци
  - ✓ Намалена скорост на обработка
  - ✓ По-голям размер на обменяните съобщения
  - ✓ Проблем при интерпретирането на неоправдано сложни документи

## ❖ Референции към услуги (endpoint)

- Дефинират се посредством URI
  - ✓ URL (Uniform Resource Locator): съдържа домейн името на компютъра, хостващ услугата
  - ✓ URN (Uniform Resource Name): зависи от контекста и изисква услуга за търсене

# Активиране на услуга и прозрачност

## ❖ Активиране на услуга

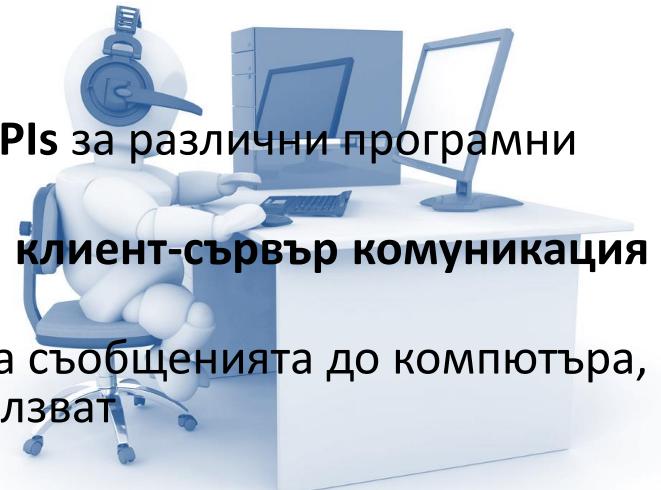
- Достъпът до услугата е от компютъра, чието домейн име се съдържа в нейния URL адрес
- Услугата може да е стартирана постоянно или да се стартира при поискване
- URL адреса на услугата е постоянна референция
  - ✓ Референцията е валидна докато сървърът, който указва, е стартиран

## ❖ Прозрачност

- Скриване на детайлите за конструиране на SOAP съобщенията в локалния приложен програмен интерфейс на програмния език
  - ✓ Интерфейсът на услугата се използва за автоматизиране на процеса по маршализация и демаршализация
- Използване на прокси или стъб процедура
  - ✓ Платформата за обработка на извикванията и процедурата за маршализиране се създават преди самите извиквания
- Динамично извикване
  - ✓ Локализирането на услугата е по време на изпълнение

# SOAP спецификация

- ❖ Цел
  - Дефинира схема за използване на XML при **представянето на заявки и отговори**, както и схема за **обмяна на документи**
- ❖ Същност (SOAP version 1.2)
  - Дефинира **начина за представяне на съдържанието** на съобщенията в XML
  - Определя начина, по който двойка съобщения могат да се комбинират за **комуникация от тип заявка-отговор**
  - Дефинира **правилата за обработка на XML елементите** в съобщенията от получателите
  - Определя **начина за използване на HTTP и SMTP** за комуникация със SOAP съобщения
- ❖ Особености
  - Наличие на **имплементации на SOAP APIs** за различни програмни езици
  - Дефиниране на стандартен протокол за **клиент-сървър комуникация в интернет** (XML + HTTP)
  - Наличие на **междинни възли** по пътя на съобщенията до компютъра, чиито ресурси е необходимо да се използват



# SOAP съобщения

## Schema in SOAP XML namespace

envelop

header

header element

header element

body

body element

body element

### ❖ Обвивка

- Съдържа заглавието и тялото на съобщението

### ❖ Заглавие

- Съхранява информация, която може да бъде обработвана от посредници на съобщението

### ❖ Тяло

- Съхранява същинската информация на съобщението в XML формат

### ❖ Стилове за комуникация

- Документен стил
  - ✓ В тялото на съобщението се поставя XML документ с референция към XML схемата, която го описва
- Клиент-сървър стил
  - ✓ В тялото на съобщението се поставя информация за заявка или отговор

# Примерни съобщения за заявка и отговор

env:envelop xmlns:env = namespace URI for SOAP envelopes

env:body

m:exchange

xmlns:m = namespace URI of the service description

m:arg1

Hello

m:arg2

World

env:envelop xmlns:env = namespace URI for SOAP envelopes

env:body

m:exchangeResponse

xmlns:m = namespace URI of the service description

m:res1

World

m:res2

Hello

*fault element*



## ❖ Незасегнати аспекти в SOAP спецификацията

- Как заглавните блокове могат да се използват от услугите на по-високо ниво?
  - ✓ Начин на използване на идентификатор на транзакция от транзакционна услуга
  - ✓ Начин на използване на идентификатора на съобщението за осигуряване на надеждност при доставяне на последователност от съобщения
  - ✓ Начин на използване на потребителско име, цифрова сигнатура или публичен ключ
- Как съобщенията се маршрутизират между множество от посредници до крайния им получател?

## ❖ Роли и задължения на посредниците

- Атрибутът *role*
  - ✓ Определя дали всеки посредник, нито един от тях или само крайният получател могат да обработват съответния елемент

# Транспортиране на SOAP съобщения

HTTP headers  
SOAP message

*POST /examples/stringer  
Host: www.cdk4.net  
Content-Type: application/soap+xml  
Action: http://www.cdk4.net/examples/stringer#exchange*

*<env:envelope xmlns:env= namespace URI for SOAP envelope  
<env:header> </env:header>  
<env:body> </env:body>  
</env:Envelope>*

## ❖ Комуникация върху HTTP

- HTTP заглавните части специфицират **URI на крайния получател** (endpoint address), **действието**, което трябва да се извърши (action) и **типа на съдържанието** *application/soap+xml*
  - ✓ Поставянето на информация за действието оптимизира диспечеризирането

- HTTP тялото пренася **SOAP съобщението**

## ❖ Предимства от разделянето на SOAP съобщението от информацията за начина на изпращането му

- Възможност за избор на подходящ модул, който да обработи съобщението без преглеждане на SOAP съобщението (Action header)
- Възможност за използване на различни комуникационни протоколи

## ❖ Проблеми

- Независимост на SOAP от транспортния протокол
- Специфициране на път за SOAP съобщенията

## ❖ Предназначение на WS-Addressing

- Отделяне на диспачеризиращата информация от транспортния протокол

## ❖ Реализация

- Специфициране на данни за маршрутизирането на съобщенията в SOAP заглавията
- Осигуряване на информация за следващия посредник на съобщението в прилежащата SOAP инфраструктура
- *Endpoint Reference*
  - ✓ XML структура, съдържаща крайния адрес и информация за маршрутизиране
  - ✓ Възможност за дефиниране на адрес за връщане на отговор и идентификатори на съобщенията

- ❖ Недостатък на TCP комуникацията
  - Липса на гаранции за доставяне и получаване на съобщенията
- ❖ Семантика при **доставяне на съобщенията**, дефинирана от Oasis в спецификацията WS-ReliableMessaging
  - At-least-once
    - ✓ Съобщенията се доставят най-малко веднъж и се докладва грешка, ако не се доставят
  - At-most-once
    - ✓ Съобщенията се доставят най-много веднъж и не се докладва грешка, ако не се доставят
  - Exactly-once
    - ✓ Съобщенията се доставят точно веднъж и се докладва грешка, ако не се доставят
  - In-order
    - ✓ Съобщенията се доставят в реда, в който са изпратени

# Уеб услуги vs. отдалечени обекти

## ❖ Отдалечени референции към обекти

- Невъзможност за използване при уеб услугите
  - ✓ Наличие на „factory“ метод при JavaRMI, който създава отдалечени обекти и връща референции към тях
  - ✓ Уеб услугите не могат да създават инстанции на отдалечени обекти

## ❖ Програмен модел за уеб услуги

- Пример: методът *newShape*
  - ✓ Отдалечен обект: Инстанциите на обекта *Shape* се създават в сървъра, при което се получава отдалечена референция към тях
  - ✓ Уеб услуга: Премахване на интерфейса *Shape* и добавяне на операциите му към интерфейса *ShapeList*
- Пример: вектор от обекти *Shape*
  - ✓ Отдалечен обект: в сървъра се съхранява вектор от обекти *Shape*
  - ✓ Уеб услуга: създава се вектор от обекти *GraphicalObject*, при което методът *NewShape* връща отместване на обекта във вектора *GraphicalObject*

## ❖ Сърванти

- Отдалечен обект: създава един сървант за *ShapeList* и по един сървант за всеки *Shape* обект
- Уеб услуга: не поддържа сърванти, не предоставя конструкути и не реализира *main* метод

# Уеб услуги vs. отдалечени обекти: пример

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject getAllState() throws RemoteException;
}
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException;
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```



```
import java.rmi.*;
public interface ShapeList extends Remote {
    int newShape(GraphicalObject g) throws RemoteException;
    int numberOfShapes() throws RemoteException;
    int getVersion() throws RemoteException;
    int getGOVersion(int i) throws RemoteException;
    GraphicalObject getAllState(int i) throws RemoteException;
}
```

Реализация на уеб услуга с JAX-RPC

# CASE STUDY

# Използване на SOAP с Java

## ❖ JAX-RPC

- Java API за разработване на уеб услуги и клиенти върху SOAP
- Дефинира **съответствие между данновите типове в Java и XML дефинициите** в SOAP съобщенията и интерфейсите на уеб услугите

## ❖ Допустими типове

- Integer, String, Date, Calendar и java.net.uri

## ❖ Изисквания за подаване на инстанции на класове като аргументи и резулти от отдалечени извиквания

- Използване на допустимите типове за променливите на инстанцията
- Необходимост от публичен конструктор по подразбиране
- Рестрикция относно реализацията на Remote интерфейс

## ❖ Java интерфейс за уеб услуга

- Наследяване на *Remote* интерфейс
- Рестрикция относно декларации на константи (*public final static*)
- Генериране на изключения от клас *java.rmi.RemoteException* (или негов под клас) от методите на интерфейса
- Използване на допустимите в JAX-RPC типове за параметрите и резултатите на методите в интерфейса

# Java реализация на ShapeList сървър

```
import java.util.Vector;  
  
public class ShapeListImpl implements ShapeList {  
    private Vector theList = new Vector();  
    private int version = 0;  
    private Vector theVersions = new Vector();  
  
    public int newShape(GraphicalObject g) throws RemoteException{  
        version++;  
        theList.addElement(g);  
        theVersions.addElement(new Integer(version));  
        return theList.size();  
    }  
    public int numberOfShapes(){}  
    public int getVersion() {}  
    public int getGOVersion(int i){ }  
    public GraphicalObject getAllState(int i) {}  
}
```

Липсва  
конструктор и  
метод main



- ❖ Компилиране на интерфейса и реализацията на услугата
  - Използване на `wscompile` и `wsdeploy`
  - Генериране на скелетон клас и описание на услугата (WSDL)

# Контейнер на сървлет

## ❖ Роля на контейнера

- Зарежда, инициализира и изпълнява сървлета
- Предоставя диспечер и скелетон

## ❖ Диспечер

- Свързва входящата SOAP заявка с подходящ скелетон

## ❖ Скелетон

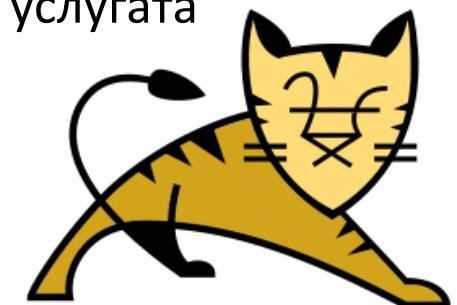
- Трансформира входящата заявка в Java и я препраща към метод на сървлета
- Преобразува резултата, получен от метода в SOAP отговор

## ❖ URL на услугата

- Включва URL на контейнера, категория и име на услугата
  - ✓ <http://localhost:8080/ShapeList-jaxrpc/ShapeList>

## ❖ Примерен контейнер

- Apache Tomcat



# Apache Tomcat

Интерфейс за управление на сървлетите

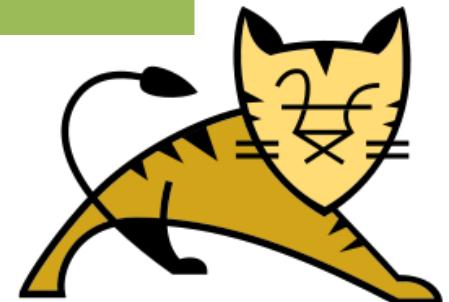
Интерфейсът за управление е достъпен на определен URL адрес през браузър

Показва имената на внедрените сървлети

Осигурява операции за управление на сървлетите

Достъп до описанието на всяка услуга

Описанията на услугите са представени в XML формат



# Java реализация на ShapeList клиент

```
package staticstub;
import javax.xml.rpc.Stub;
public class ShapeListClient {
    public static void main(String[] args) /* pass URL of service */
        try {
            Stub proxy = createProxy();
            proxy._setProperty
                (javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY, args[0]);
            ShapeList aShapeList = (ShapeList)proxy;
            GraphicalObject g = aShapeList.getAllState(0);
        } catch (Exception ex) { ex.printStackTrace(); }
    }
    private static Stub createProxy() {
        return
            (Stub) (new MyShapeListService_Impl().getShapeListPort());
    }
}
```



# Механизми за свързване с услугата

## ❖ Статично прокси

- Локален обект, който подава съобщения на отдалечена услуга
- Генерира се при компилиране на описанието на услугата с *wscompile*
- Създава се с метод *createProxy* и се инициализира с адреса на услугата, взет от командния ред
- Типа на проксито се ограничава до типа на интерфейса на услугата
- Проксито се използва за извикване на метода *getAllState*

## ❖ Динамично прокси

- Класът на проксито се генерира в режим на изпълнение от описанието и интерфейса на услугата

## ❖ Интерфейс за динамично извикване

- Клиентът използва серия от операции, за да получи информация за услугата

# Реализация на Java SOAP

- ❖ Комуникационни модули
  - Използва се двойка от HTTP модули
  - HTTP модула в сървъра **избира диспачер** въз основа на URL адреса в *Action* заглавната част на POST заявката
- ❖ Клиентско прокси
  - **Маршализира** името на метода и аргументите, заедно с референция към XML схемата за услугата в SOAP заявка
  - **Демаршализира** резултата от SOAP отговора
- ❖ Диспачер и скелетон
  - Диспачерът получава името на операцията от *Action* заглавната част на HTTP заявката и **извиква подходящ метод в скелетона**
  - Скелетонът извлича информация за метода от SOAP заявката, **извиква метода** и пакетира резултата в SOAP отговор
- ❖ Грешки, генериирани от уеб услугата
  - Представят се с върната стойност от изпълнение на метод или параметри за грешка в описанието на услугата
- ❖ Проверка за коректност от страна на скелетона (клиентското прокси)
  - **Проверява** дали SOAP обвивката съдържа **правилно форматирана с XML заявка**
  - **Използва XML схемата от SOAP обвивката**, за да провери дали заявката отговаря на услугата и дали операцията и нейните аргументи са подходящи

# Уеб услуги vs. CORBA

## ❖ Именуване

- Отдалечените обекти в CORBA се **реферират по име**, управлявано от CORBA услуга за именуване
- CORBA обектите се споделят в рамките на **дадена организация или малък кръг от организации**
  - ✓ Всеки сървър управляет собствен граф с имена, който е независим от тези на останалите сървъри

## ❖ Референции

- IOR съдържа **типов идентификатор на интерфейса**, който е разбираем само за интерфейсното хранилище
- Услугите се идентифицират с **URL**, който осигурява отдалечено извикване в интернет

## ❖ Отделяне на активирането и определянето на местоположението при услугите

- Постоянните референции в CORBA указват хранилище за реализации

## ❖ Удобство на използване

- CORBA платформата изисква инсталация и поддръжка

## ❖ Ефективност

- **По-висока ефективност при CORBA** поради двоичното представяне на данните (CORBA CDR) в съобщенията
- Възможността за включване на двоични данни в XML елементи се дискутира от W3C
  - ✓ XML поддържа шестнайсетично и base64 представяне на двоични данни
  - ✓ SOAP + attachments (използване на MIME документи при транспортиране)

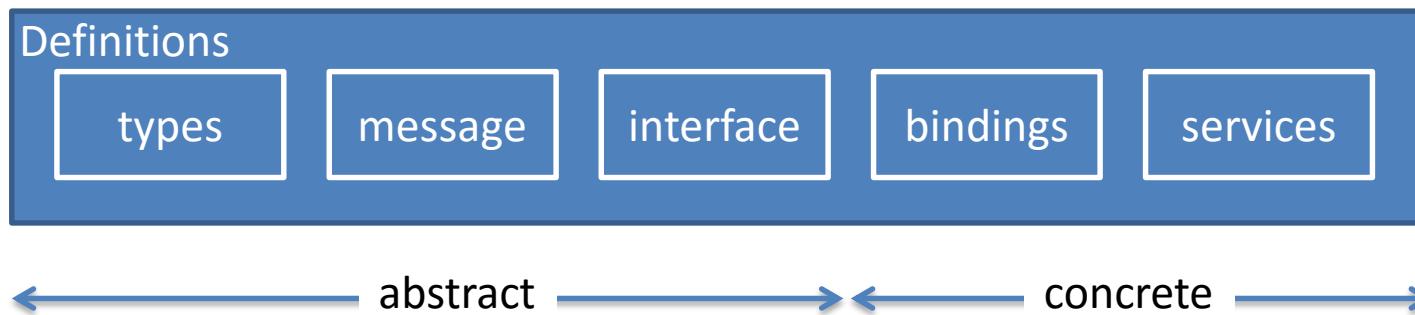
# Описание на уеб услугите

## ❖ Същност

- Специфицира обмяната на съобщенията при комуникация с услугата и нейния URI
- Определя стила на комуникация: документен или клиент-сървър
- Дефинира метода на комуникация между клиента и услугата
- Премахва необходимостта от използване на услуга за определяне на местоположението

## ❖ Web Service Description Language (WSDL)

- Дефинира XML схема за представяне на компонентите от описанието на услугата



# Структура на WSDL описанието

## ❖ Абстрактна част

- Дефиниция на типове за стойностите, обменяни със съобщения
  - ✓ Комплексните типове се представят като последователност от именувани елементи

```
<element name="isFilled" type="boolean"/>
<element name="originx" type="int"/>
```
- Дефиниция на съобщенията, обменяни с услугата
- Дефиниция на интерфейс като множество от предоставяни операции

## ❖ Конкретна част

- Специфицира начина за комуникация и местоположението на услугата

# WSDL съобщения за операцията newShape

message name = “ShapeList\_newShape”

part name=“GraphicalObject\_1”  
type= “ns:GraphicalObject”



message name = “ShapeList\_newShapeResponse”

part name=“result”  
type= “xsd:int”

# WSDL: дефиниция на интерфейс

- ❖ Предназначение
    - Дефинира множество от операции с описание на шаблон за обмяна на съобщения
  - ❖ Шаблони за обмяна на съобщения
    - In-Out
      - ✓ Комуникация от тип заявка-отговор, инициирана от клиента
    - In-Only
      - ✓ Еднопосочко съобщение от клиента към услугата (не се поддржат fault съобщения)
    - Out-Only
      - ✓ Еднопосочко съобщение от услугата към клиента (не се поддржат fault съобщения)
    - Out-In
      - ✓ Комуникация от тип заявка-отговор, инициирана от сървъра
    - Robust In-Only
      - ✓ Еднопосочко съобщение от клиента към услугата с гарантирана доставка
    - Robust Out-Only
      - ✓ Еднопосочко съобщение от услугата към клиента с гарантирана доставка
  - ❖ Наследяване
- ```
operation name = "newShape"
          pattern = In-Out
          input message = "tns:ShapeList_newShape"
          output message = "tns:ShapeList_newShapeResponse"
```

## WSDL: елементът binding

```
binding
  name = "ShapeListBinding"
  type="tns:ShapeList"

  soap: binding transport = URI
  for schemas for soap/http
  style="rpc"
```

```
operation name = "newShape"
```

```
  input
    soap:body
    encoding, namespace
```

```
  output
    soap:body
    encoding, namespace
```

```
  soap:operation
    soapAction
```

### ❖ Предназначение

- Определя формата на съобщенията и външното представяне на данните

### ❖ *soap:binding*

- Специфицира URL на протокол за транспортиране на SOAP съобщения
- Атрибути
  - ✓ Стил за обмяна на съобщения
  - ✓ XML схема за формат на съобщенията (пр. SOAP envelope)
  - ✓ XML схема за външно представяне на данните (пр. SOAP encoding)

### ❖ *wsdl:operation*

- Специфицира входящото и изходящото съобщение за дадена операция

### ❖ *soap:operation*

- Използва атрибут *soapAction*, за да дефинира URI за крайната точка на заявката към услугата

## WSDL: елементът service

```
service name =  
"MyShapeListService"
```

```
endpoint  
name = "ShapeListPort"  
binding =  
"tns:ShapeListBinding"
```

```
soap:address  
location = service URI
```

### ❖ wsdl:service

- Специфицира името на услугата и крайни точки за свързване с нея

### ❖ wsdl:port

- Дефинира крайна точка като комбинация от обвързване и мрежов адрес

### ❖ soap:address

- Специфицира URI за местоположението на услугата

### ❖ Употреба на WSDL

- Описанията на услугите са достъпни за клиентите чрез URI или директорийна услуга (UDDI)
- Наличие на инструменти за автоматично генериране на WSDL от програмен код
  - ✓ Пример WSDL4J

# Universal Description Discovery and Integration (UDDI)

## ❖ Предназначение

- Описание на организации и предлаганите от тях услуги
- Откриване на информация за компаниите и осигурените от тях услуги
- Взаимодействие с компаниите и консумиране на предлаганите услуги

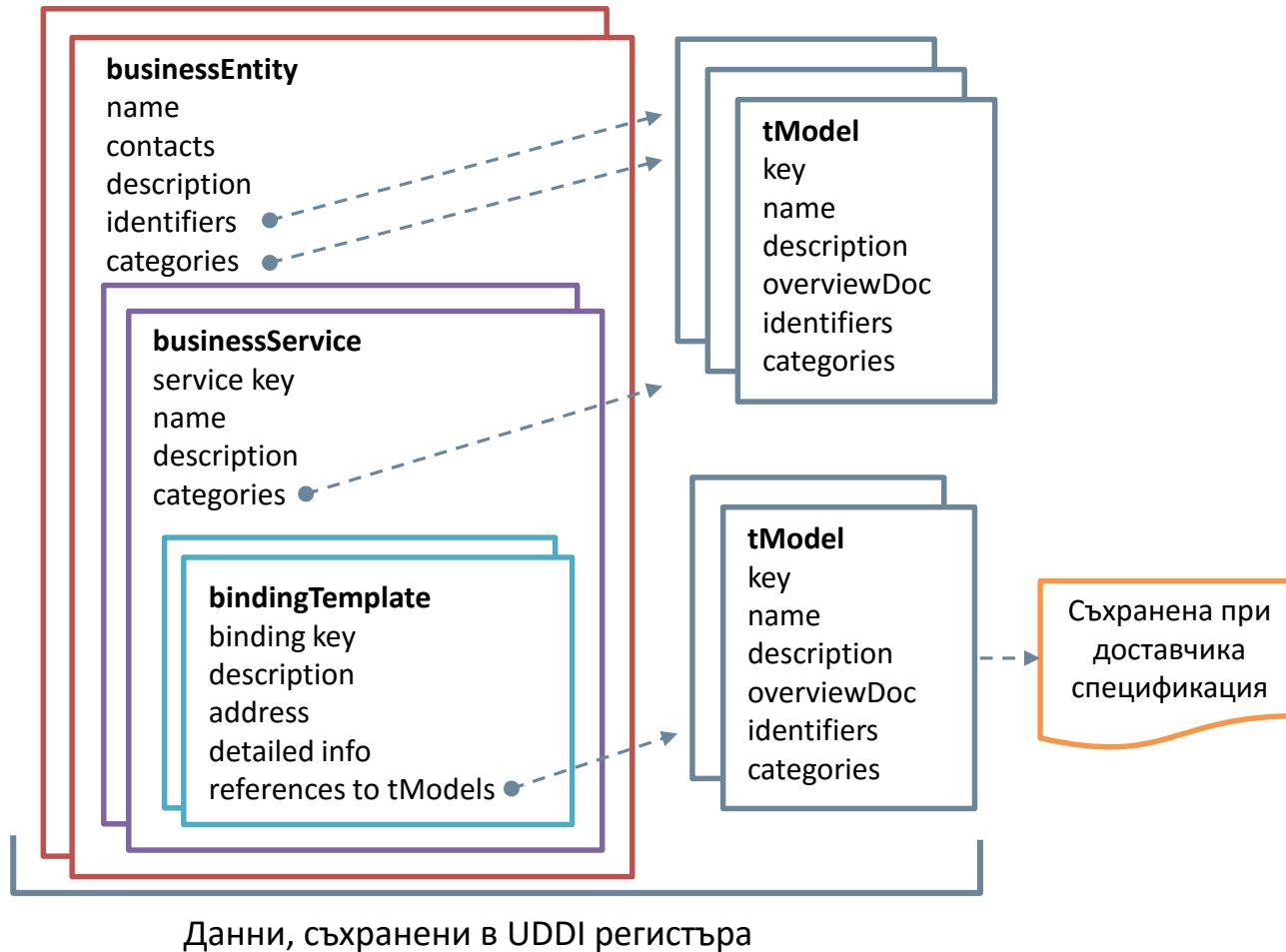
## ❖ Структура

- Бели страници
  - ✓ Информация за доставчиците на услуги
- Жълти страници
  - ✓ Класифициране на информацията според индустриални класификации, базирани на стандартни таксономии
- Зелени страници
  - ✓ Техническа информация за уеб услугите

# Даннови структури на UDDI

- ❖ UDDI дефинира стандарт за представяне на данновите структури в регистъра
- ❖ businessEntity
  - Описва организация, предоставяща уеб услуги
- ❖ businessService
  - Съхранява информация за множество от инстанции на уеб услуга
- ❖ bindingTemplate
  - Съхранява адрес на инстанция на уеб услуга и референция към описането ѝ
- ❖ tModel
  - Съхранява информация за местоположението на описането на услугата

# Схема на данновите структури на UDDI



## ❖ Интерфейс за търсене и получаване на информация

- Find операции: Откриват информация, отговаряща на определени критерии
  - ✓ find\_business, find\_relatedBusinesses, find\_service, find\_binding, find\_tModel
- Get операции: Извличат специфична информация след реализиране на търсене
  - ✓ get\_businessDetail, get\_businessDetailExt, get\_serviceDetail, get\_bindingDetail, get\_tModelDetail

## ❖ Интерфейс за нотификация

- Използва се от клиентите за получаване на информация относно промени в регистъра

## ❖ Интерфейс за публикуване и обновяване

- Публикуването на информация е свързано с генериране на ключ за идентификация
- Услугите принадлежат на сървъра, в който са публикувани

# Репликация и взаимодействие между UDDI сървърите

## ❖ Структура на UDDI регистъра

- Един или няколко сървъра, всеки от които притежава копие на данните

## ❖ Репликация

- Промяната на данните се прави в сървъра, където са публикувани
- Сървърът с промяна нотифицира останалите сървъри в регистъра, които от своя страна изпращат заявка за промяна
- Промените от даден сървър се получават в последователен ред от останалите сървъри
- Промените от различни сървъри не спазват определена подредба

## ❖ Взаимодействие

- Преотстъпване на собственост върху даннова структура от един сървър на друг
- Търсенето в регистъра не изиска взаимодействие между UDDI сървърите

## ❖ Същност

- Дефинира набор от W3C изисквания за подписване на документи, управление на ключове и криптиране

## ❖ Базови изисквания

- Възможност за криптиране на цял документ или част от него
  - ✓ Пример: Финансова транзакция
- Възможност за подписване на цял документ или част от него
  - ✓ Пример: Корпоративна работа на група от хора върху документ

## ❖ Допълнителни изисквания

- Възможност за допълване на подписан документ и подписване на резултат
  - ✓ Пример: Удостоверяване на подpis на един служител от друг
- Възможност за оторизиране на различни потребители да виждат различни части от документа
  - ✓ Пример: Данни от медицински преглед
- Възможност за допълване на документ, който има криптиирани секции, и криптиране на части от новата версия

## ❖ Изисквания по отношение на алгоритмите

- Стандартът трябва да специфицира **пакет от алгоритми** за всяка реализация на XML сигурността
- **Алгоритъмът**, използван за криптиране и автентикация на даден документ, **трябва да се избира от специфицирания пакет**
- **Имената на използваните алгоритми** трябва да се реферират в рамките на самия XML документ

## ❖ Изисквания по отношение на ключовете за търсене

- Подпомагане на потребителите на даден защищен документ в откриването на необходимите ключове
- Възможност за коопериране на потребителите при използване на ключове

## ❖ Елементът KeyInfo

- Ключ, който се използва за валидиране на подpis или декриптиране на данни
- Може да съдържа сертификати, имена на ключове или алгоритми

# Канонична форма на XML

## ❖ Предназначение

- Гарантира липса на промяна в информационното съдържание на документа

## ❖ Приложение

- Канонизиране на XML елементите преди подписване на документа и съхраняване на името на канонизиращия алгоритъм с подписа

## ❖ Същност на канонизацията

- Сериализиране на XML като поток от байтове
  - ✓ Добавяне на подразбиращи се атрибути
  - ✓ Премахване на излишни схеми
  - ✓ Поставя атриутите и декларациите на схемите в лексикографичен ред във всеки елемент
  - ✓ Използване на стандартна форма за нов ред
  - ✓ Използване на UTF-8 кодиране

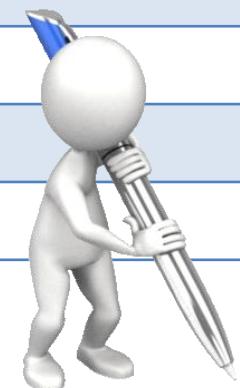
# Цифрови подписи в XML

## ❖ Спецификация на цифровите подписи в XML

- W3C препоръка, която дефинира нови типове XML елементи за представяне на подписи, имена на алгоритми, ключове и референции към подписанна информация
- Използване на XML Signature schema
  - ✓ Signature, SignatureValue, SignedInfo, KeyInfo

## ❖ Алгоритми, необходими за реализация на XML подпис

| Type of algorithm         | Name of algorithm | Required    |
|---------------------------|-------------------|-------------|
| Message digest            | SHA-1             | Required    |
| Encoding                  | base64            | Required    |
| Signature (asymmetric)    | DSA with SHA-1    | Required    |
| Signature (asymmetric)    | RSA with SHA-1    | Recommended |
| MAC signature (symmetric) | HMAC-SHA-1        | Required    |
| Canonicalization          | Canonical XML     | Required    |



# Услуга за управление на ключове и XML криптиране

- ❖ Спецификация на услуга за управление на ключове
  - Съдържа протоколи за дистрибуция и регистриране на публични ключове в XML подписите
  - Съвместимост със съществуващите инфраструктури за публични ключове, като X.509, SPKI (Simple Public Key Infrastructure), PGP (Pretty Good privacy)
- ❖ XML стандарт за криптиране
  - W3C препоръка, специфицираща начин за представяне на криптирани данни в XML и процеса на криптиране и декриптиране
  - Дефинира елемент *Encrypted Data*
- ❖ Алгоритми за XML криптиране

| Type of algorithm                            | Name of algorithm                                         | Required |
|----------------------------------------------|-----------------------------------------------------------|----------|
| Block cipher                                 | TRIPLEDES, AES 128, AES-256                               | Required |
| Block cipher                                 | AES-192                                                   | Optional |
| Encoding                                     | base64                                                    | Required |
| Key transport                                | RSA-v1.5, RSA-OAEP                                        | Required |
| Symmetric key wrap (signature by shared key) | TRIPLEDES                                                 | Required |
| Symmetric key wrap (signature by shared key) | KeyWrap, AES-128 KeyWrap, AES 256KeyWrap, AES-192 KeyWrap | Optional |
| Key agreement                                | Diffie-Hellman                                            | Optional |

# Необходимост от координация: туристически агент

1. The client asks the travel agent service for information about a set of services; for example, flights, car hire and hotel bookings.
2. The travel agent service collects prices and availability information and sends it to the client, which chooses one of the following on behalf of the user:
  - a) refine the query, to get more information, then repeat step 2;
  - b) make reservations;
  - c) quit.
3. The client requests availability.
4. Either all are available or for services that either alternative or the client goes back to step 3;
5. Take deposit.
6. Give the client a reservation confirmation.
7. During the period until the final payment, the client may modify or cancel reservations

Уеб услугите, които са клиенти на други услуги се нуждаят от описание на протокол, който да следват при взаимодействието си с тях, както и механизъм за запазване на консистентността на данните.

# Хореография на услуги

## ❖ Предназначение

- Поддръжка на взаимодействие между множество услуги, управявани от различни организации

## ❖ Изисквания към езика за хореография

- Йерархична и рекурсивна композиция на хореографии
- Възможност за добавяне на нови инстанции на услуги или нови услуги
- Конкурентни пътища, алтернативни пътища и възможност за повторение на секции в хореографията
- Времеви ограничения за променливи
  - ✓ Пример: Ограничаване на резервация за период
- Изключения
  - ✓ Пример: Прихващане на грешен ред на съобщения; Отказ на резервация
- Асинхронно взаимодействие
- Предаване на референции
  - ✓ Пример: Проверка на платежоспособност от страна на туристическа агенция
- Отбелязване на границите за отделните транзакции
- Възможност за включване на документация, разбираема от човек

## ❖ Web Service Choreography Language

- XML базиран език за описание на хореографии

# Приложение на уеб услугите: SOA

## ❖ Service-Oriented Architecture (SOA)

- Дефинира набор от принципи за разработване на разпределени приложения с използване на слабо свързани услуги
- Абстрактна концепция, която може да се реализира с различни технологии

## ❖ Цел

- Осигуряване на гъвкава софтуерна архитектура за организациите
- Постигане на оперативна съвместимост
- Бизнес към бизнес интеграция

## ❖ Mashup подход за реализация на софтуерни приложения

- Създаване на нова услуга от трети страни посредством комбиниране на функционалността на две или повече услуги с всеобщо установен интерфейс и предоставянето й за използване в други приложения

✓ Amazon, Flickr, eBay

# Приложение на уеб услугите: Grid

## ❖ Grid

- Платформа за споделяне на ресурси между група от потребители в различни организации за постигане на обща цел
  - ✓ Споделяне на данни или изчислителен ресурс
- Ресурсите се поддържат от хетерогенни компютърен хардуер, операционни системи, програмни езици и приложения
- Необходимост от механизми за управление на достъпа и сигурност

## ❖ Пример: World-Wide Telescope by Microsoft Research

- Осигурява споделяне на даннови ресурси в областта на астрономията
- Състои се от приложения, използващи интензивно данни
  - ✓ Събиране на данни от **научни инструменти**
  - ✓ Съхраняване на данни в **архиви на различни места**
  - ✓ Управление на данни от екипи с учени, работещи в **различни организации**
  - ✓ Генериране на **огромно количество “сирови” данни** от инструментите
  - ✓ Използване на компютърни програми за **анализ на данните и генериране на обобщения**

# Приложение на уеб услугите: Grid

## ❖ Изисквания към Grid приложението

- **Отдалечен достъп до ресурси**
- **Обработка на данните, където се управяват и съхраняват**
- Възможност мениджъра на ресурси да създава динамично инстанции на услуги, опериращи върху части от данните
- Използване на метаданни
  - ✓ Характеристика на данните в архива
  - ✓ Характеристики на услугите, управляващи данните (цена, доставчик и т.н.)
- Осигуряване на директорийни услуги въз основа на метаданните
- Осигуряване на софтуер за управление на заявки, трансфер на данни и резервиране на ресурси

## ❖ Grid приложения, извършващи интензивни изчисления

- Обработка на данните от CMS ускорителя на частици в CERN
- Тестване на молекули, които са кандидати за лекарствени препарати
- Поддръжка на мащабни онлайн игри

# Приложение на уеб услугите: Cloud computing

## ❖ Същност

- Нов бизнес модел, при който облачните доставчици предлагат услуги на клиентите в зависимост от ежедневните им нужди
- Услугите се предлагат на платформено, инфраструктурно и софтуерно ниво
- Наследява характеристики на Grid технологията и уеб услугите

## ❖ Пример: Amazon Web Services

- Множество от облачни услуги, реализирани върху широка физическа инфраструктура, притежавана от Amazon.com
- Предоставяните услуги са базирани на стандартите за уеб услуги



## ❖ Amazon Elastic Compute Cloud (EC2)

- Уеб базирана услуга за достъп до виртуални машини с определена производителност и дисков капацитет

## ❖ Amazon Simple Storage Service (S3)

- Уеб базирана услуга за съхраняване на неструктурирани данни

## ❖ Amazon Simple DB

- Уеб базирана услуга за генериране на заявки към структурирани данни

## ❖ Amazon Simple Queue Service (SQS)

- Хоствана услуга, поддържаща опашки със съобщения

## ❖ Amazon Elastic MapReduce

- Уеб базирана услуга за разпределени изчисления, базирани на модела MapReduce

## ❖ Amazon Flexible Payments Service (FPS)

- Уеб базирана услуга за електронни разплащания

# Amazon Elastic Compute Cloud

## ❖ Предназначение

- Осигурява виртуални машини с променлив изчислителен капацитет

## ❖ Типове виртуални машини (инстанции)

- standard instance
  - ✓ Подходяща за широк кръг от приложения
- high-memory instance
  - ✓ Осигурява допълнителна оперативна памет
- high-CPU
  - ✓ Осигурява изпълнение на задачи с висока изчислителна сложност
- cluster compute instance
  - ✓ Осигурява кълстер от виртуални процесори с високочестотна връзка за извършване на изчисления с висока производителност

## ❖ Конфигурация

- Xen hypervisor
- Поддръжка на разнообразие от операционни системи
- Поддръжка на конфигурации с разнообразен софтуер

## ❖ “Еластичен” IP адрес

- Асоциира се с клиентския акаунт, а не с виртуалната машина





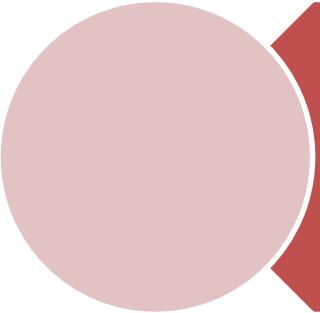
## P2P системи

доц. д-р Десислава Петрова-Антонова

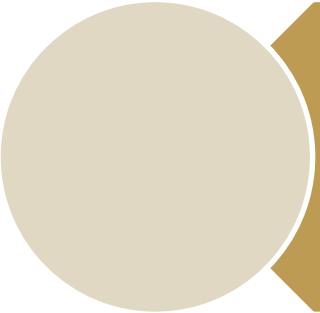
# Съдържание

- ❖ P2P системи: същност и характеристики
- ❖ Приложението Napster
- ❖ P2P платформи
- ❖ Маршрутизация
- ❖ Case study: Pastry, Tapestry
- ❖ Case study: Squirrel, OceanStore

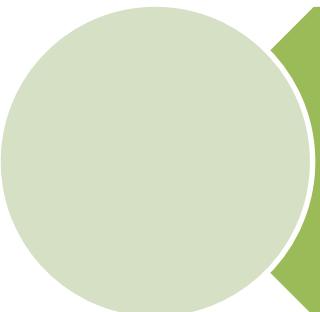
# Ограничения при доставчиците на услуги



Необходими ресурси за администриране на инфраструктурата



Разходи за възстановяване от повреди



Ограничения, свързани с капацитета на единични сървъри

# Цел на P2P системите

- ❖ Споделяне на данни и ресурси в много голям мащаб
  - Премахване на изискването за отделно управление на сървъри и свързаната с тях инфраструктура
- ❖ Поддръжка на разпределени услуги и приложения
  - Използване на данни и изчислителни ресурси от персоналните компютри и работни станции в интернет
- ❖ Създаване на приложения, използващи ресурси в периферията на интернет
  - Споделените ресурси се осигуряват от разпределени приложения, използвани при повечето персонални компютри
- ❖ Преодоляване на ограниченията на клиент-сървър системите
  - Скалируемостта се ограничава от хардуерния капацитет и мрежовата свързаност

## ❖ Същност

- Достъп до информационни ресурси, съхранявани на компютри в мрежа
- Алгоритми за разпределение и извличане на информационни обекти
- Напълно децентрализирана и самоорганизираща се услуга за динамично балансиране на паметта и натоварването

## ❖ Характеристики

- Доставяне на ресурси от всеки потребител на системата
- Еднакви функционални възможности и отговорности на възлите в системата
- Липса на зависимост от централизирано администрирана система
- Ограничена анонимност на доставчиците и потребителите на ресурси
- Необходимост от ефективен алгоритъм за разпределение и достъп до данните при балансиране на натоварването

## Недостатъци

- Непостоянство на ресурсите
- Непредвидимост на процесите и компютрите в системата
- Липса на гарантиран достъп до определени ресурси

## Преодоляване на недостатъците

- Репликация на ресурсите

# Поколения P2P системи

- ❖ DNS
- ❖ Netnews/Usenet
- ❖ Xerox Grapevine
- ❖ Napster
- ❖ Freenet
- ❖ Gnutella
- ❖ Kazaa
- ❖ BitTorrent
- ❖ Pastry
- ❖ Tapestry
- ❖ CAN
- ❖ Chord
- ❖ Kademlia



Скалируемост,  
анонимност,  
отказоустойчивост



Независимо от  
приложението  
управление на  
ресурсите



# Характеристики на P2P системи от трето поколение

## ❖ Особености

- Съхраняване на ресурси в множество от компютри, разпределени в интернет, и маршрутизиране на съобщения между тях
- **Гарантирана доставка за заявки при ограничен брой мрежови преходи**
- Поддръжка на репликирани ресурси, организирани по структуриран начин

## ❖ Ресурси

- Идентифицират се с глобален уникален идентификатор (GUID)
  - ✓ Прилагане на хеш функция върху състоянието на ресурса (“самосертифициране” на ресурсите)
  - ✓ Защита от съхраняване в ненадеждни възли
- Рискове при съхраняване на обекти в компютри, принадлежащи на една организация, географски район, администрация, държава и др.
  - ✓ Случайно разпределение на репликите в мрежата

# Маршрутизация

|                                | IP маршрутизация                                                                                            | Маршрутизация на приложно ниво                                                                                         |
|--------------------------------|-------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| Машаб                          | IPv4 ( $2^{32}$ адресирами възли) и IPv6 ( $2^{128}$ адресирами възли); йерархично структуриране на адреси  | GUID пространството е с по-голям машаб ( $>2^{128}$ ) и не е йерархично структурирано                                  |
| Балансиране на натоварването   | Натоварването върху рутерите се определя от мрежовата топология и асоциираните шаблони за трафик            | Местоположението на обектите може да е случайно, поради което шаблоните за движение са отделени от мрежовата топология |
| Мрежова динамика               | Асинхронно обновяване на IP маршрутизиращите таблици                                                        | Синхронно или асинхронно обновяване на маршрутизиращите таблици със закъсление, което е части от секундата             |
| Отказоустойчивост              | Отказоустойчивостта може да се гарантира за единични рутери или мрежови повреди                             | Маршрутите и обектите референции могат да се реплицират                                                                |
| Идентификация на целевия обект | Всеки IP адрес съответства на точно един целеви възел                                                       | Съобщенията могат да се маршрутизират до най-близката реплика на целевия обект                                         |
| Сигурност и анонимност         | Адресирането е сигурно, ако всички възли са надеждни. Не се осигурява анонимност за собствениците на адреси | Сигурността е достижима дори и в среда с ограничена надеждност. Осигурява се ограничено ниво на анонимност             |

# Развитие на разпределени изчисления

## ❖ Xerox PARC [Shoch and Hupp 1892]

- Извършване на слабо свързани, интезивни изчисления
- Стартиране на изчислителни процеси върху ~100 персонални компютри, свързани в локална мрежа

## ❖ SETI@home [Anderson et. Al. 2002]

- Разделя поток от цифровизирани радио телескопични данни в работни единици с продължителност 107 секунди и размер 350 KB
  - ✓ Репликиране на работните единици върху 3-4 компютъра
- Разделянето на работни единици и координирането на резултатите се извършва от единичен сървър, комуникиращ с клиентите
- Количествени показатели, свързани с проекта, за 1 година
  - ✓ 3,91 млн. персонални компютри
  - ✓ 221 млн. работни единици
  - ✓ 27,36 терафлопа изчислителна мощ

# Napster: функционалност

## ❖ Предназначение

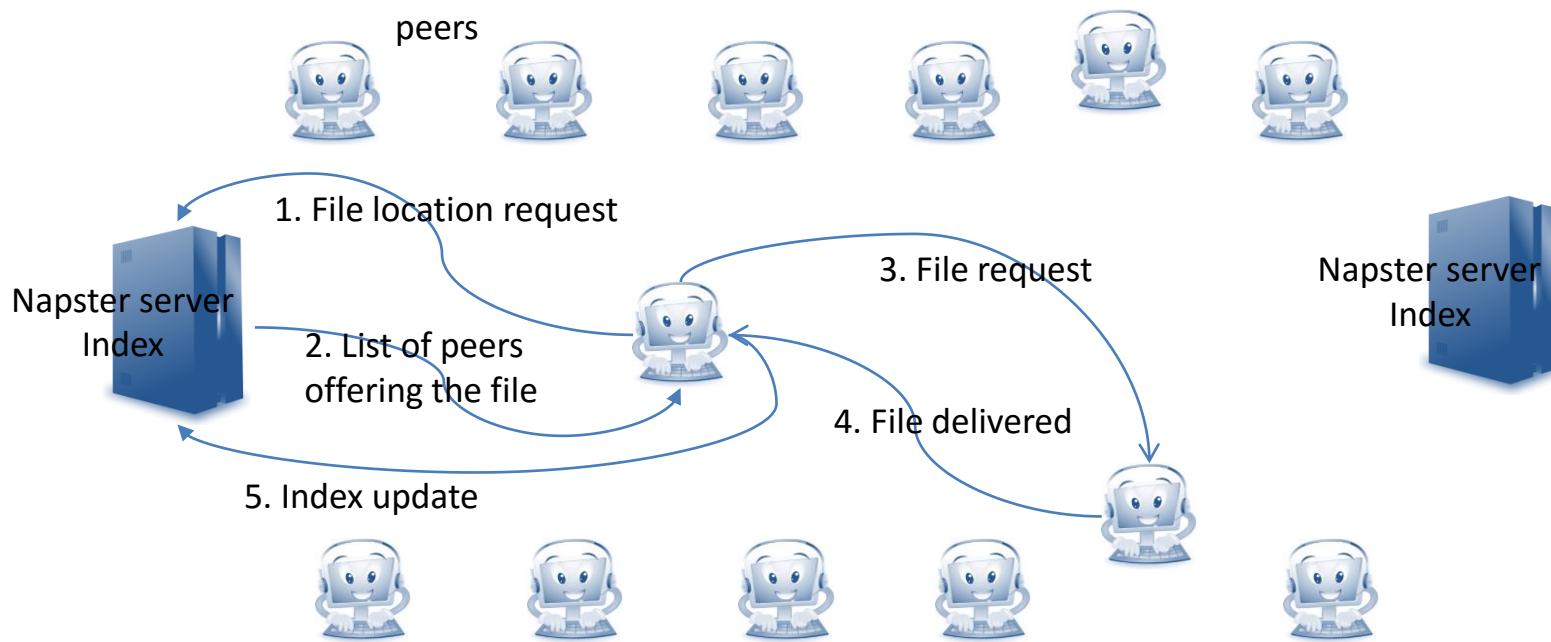
- Споделяне на файлове

## ❖ Архитектура

- Съхраняването и доставяне на файлове от персоналните компютри на потребителите
- Осигуряване на услуга за индексиране

## ❖ Развитие

- Сваляне на системата от употреба поради нарушаване на авторски права



# Napster: Изводи

## ❖ Ползи

- Възможност за изграждане на полезна, скалируема в широк мащаб услуга, зависеща почти изцяло от данните и компютрите на регулярните интернет потребители
- Използване на механизъм за разпределение на натоварването

## ❖ Ограничения

- Липса на консистентност на репликите за индексът на музикалните файлове
- Трудности при откриване и адресиране на обектите

## ❖ Зависимост от практическото приложение

- Липса на промени в музикалните файлове, поради което проверката на консистентността на репликите не е необходима
- Липса на необходимост от гаранции за наличността на файловете

# Функционални изисквания към P2P платформите

- ❖ Опростяване на конструирането на **услуги, които са реализирани върху множество хостове** в мрежова среда
- ❖ Възможност **клиентите да откриват и комуникират с** всеки ресурс, достъпен за услугата
- ❖ Възможност за **добавяне на нови и премахване на съществуващи ресурси**
- ❖ Възможност за **добавяне на нови и премахване на съществуващи хостове**
- ❖ Осигуряване на **опростен програмен интерфейс**, който е независим от типа на разпределените ресурси



# Нефункционални изисквания към P2P платформите

## ❖ Глобална скалируемост

- Възможност за достъп до огромен брой обекти, осигурени от множество хостове

## ❖ Балансирано натоварване

- Случайно разпределение на ресурси и използване на реплики за ресурсите с висока използваемост

## ❖ Оптимизация на локалните взаимодействия между съседните възли

- Зависимост на взаимодействието от мрежовата дистанция на възлите
- Поставяне на ресурсите максимално близко до възлите, от които се достъпват най-много

## ❖ Динамична наличност на хостовете

- Динамичност на възлите в мрежата и липса на централизирано управление
- Осигуряване на надеждни услуги независимо от спецификата на P2P системите
  - ✓ Балансиране на натоварването при добавяне и премахване на хостове

## ❖ Сигурност на данните в среда с хетерогенна надеждност

- Използване на механизми за автентикация и криптиране

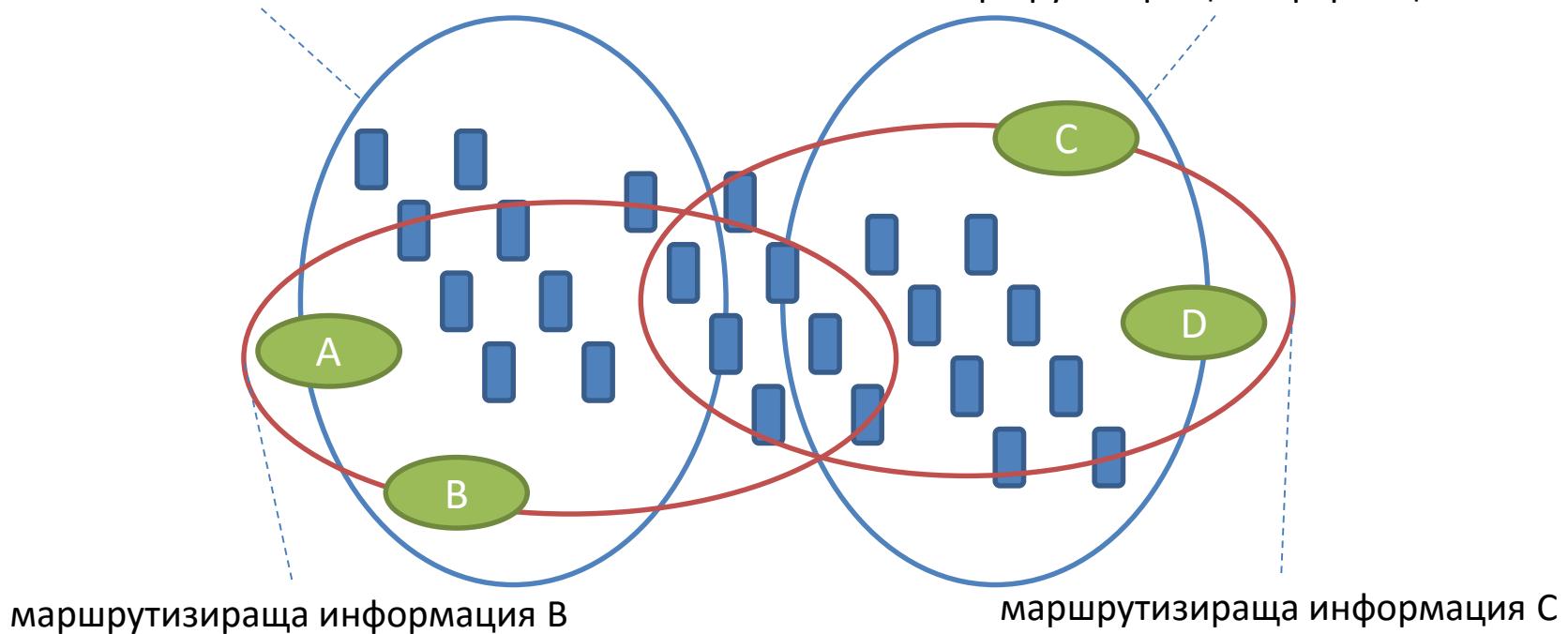
## ❖ Анонимност, възможност за отказ и устойчивост на цензура

# Местоположение на обектите

- ❖ Информацията за разпределението на обектите се разделя и разпределя в мрежата
- ❖ Всеки възел поддържа информация за местоположението на обектите в част от пространството, както и информация за топологията на пространството като цяло
- ❖ За осигуряване на висока надеждност се извършва репликация

маршрутизираща информация А

маршрутизираща информация D



# Маршрутизиращ слой: концепция

## ❖ Същност

- Разпределен алгоритъм за откриване на възли и обекти
- Маршрутизиране на заявки от произволен клиент към хост, който съхранява заявения обект
- Обектите се поставят и преразпределят във възлите без намеса на клиента
- Реализира се на **приложно ниво**
- Достъп от всеки възел до произволен обект
  - ✓ Изпращане на заявката до най-близкия активен възел при репликация
- Идентификацията на обектите е с GUID

## ❖ Задачи

- Маршрутизиране на заявки до обекти
  - ✓ Клиентът изпраща заявка с GUID на обекта
- Добавяне на обекти
  - ✓ Генериране на GUID от възела, съхраняващ обекта
- Изтриване на обекти
- Добавяне и премахване на възли
  - ✓ Преразпределяне на отговорностите между възлите

# Маршрутизиращ слой: Pastry

## ❖ Генериране на GUID

- Хеш функция, използваща част или цялото състояние на обекта
- Уникалността се проверява посредством търсене на друг обект със същия GUID

## ❖ Представяне на рутиращия слой като разпределена хеш таблица в Pastry

- Използване на идентификаторите за определяне на местоположението на обектите
- $\text{put}(\text{GUID}, \text{data})$ 
  - ✓ Съхранява данни в реплики при всички възли, отговорни за обект с определен GUID
- $\text{remove}(\text{GUID})$ 
  - ✓ Изтрива всички референции към определен GUID и асоциираните с него данни
- $\text{value} = \text{get}(\text{GUID})$ 
  - ✓ Предоставя данни, асоцииирани с определен GUID

# Маршрутизиращ слой: Tapestry

- ❖ Слой за разпределено локализиране на обекти и рутиране (DOLR) в Tapestry
  - Поддържа съответствие между идентификаторите на обектите и възлите, в които се съхраняват техните реплики
  - Репликиране на обекти и съхраняване с един и същ идентификатор в различни хостове
- ❖ Съхраняване на обект с GUID X при **DHT модел**
  - Съхранява се във възел, чийто GUID е числово най-близък до  $X$  и в  $r$  на брой хостове, чийто GUID са числово следващи по-близост
- ❖ Съхраняване на обект при **DOLR модел**
  - Местоположенията на репликите за обектите се определят извън рутиращия слой
    - ✓ Маршрутизиращият слой се уведомява за хост адреса на всяка реплика посредством операцията *publish*
  - *publish(GUID)*
    - ✓ Възелът, изпълняващ операцията става хост за обекта с определен GUID
  - *unpublish(GUID)*
    - ✓ Обектът с определен GUID става недостъпен
  - *sendToObj(msg, GUID, [n])*
    - ✓ Изпращане на съобщение до обект (или  $n$  реплики) с цел достъп до него

# Маршрутизиращи схеми

## ❖ Префикс маршрутизиране

- Пътят на съобщенията се базира на GUID идентификатори
- Определянето на следващ възел в пътя става посредством прилагане на **двоична маска**, която селектира **нарастващ брой шестнадесетични цифри** от GUID при всеки преход

## ❖ Маршрутизираща схема при Chord

- Използва **числова разлика** между GUID идентификаторите на текущия възел и крайния възел

## ❖ Маршрутизираща схема при CAN

- Използва разстояние в **d-мерно хиперпространство**, в което са поставени възлите

## ❖ Маршрутизираща схема при Kademlia

- Използва **XOR на двойки GUID идентификатори** като метрика за разстояние между възли

## ❖ Получаване на GUID

- Използване на **индексираща услуга и достъп по име**
- **Стъб файл** (BitTorrent)
  - ✓ Съхранява GUID и URL на хоста, съхраняващ списък с мрежови адреси, на които е достъпен обекта

Pastry, Tapestry

# CASE STUDY

# Pastry: базова концепция

## ❖ Идентификатори

- 128-битов GUID ( $0 \div 2^{128} - 1$ )
- При **възлите** се прилага хеш функция (SHA-1) върху публичния им ключ
- При **обектите** се прилага хеш функция върху името на обекта или част от състоянието му

## ❖ Маршрутизиране на съобщение

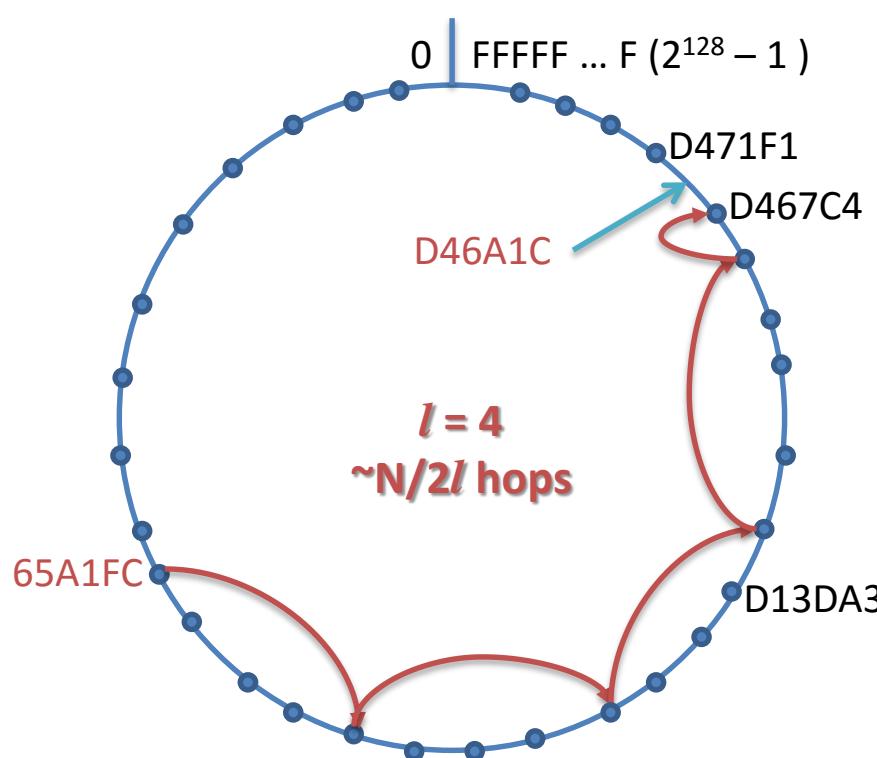
- При  $N$  възела маршрутизирането на съобщение до обект с определен GUID изисква  $O(\log N)$  стъпки
- Ако възелът, идентифициран от GUID не е активен, съобщението се препраща към активен възел с най-близкия по стойност GUID
- Маршрутизиращите стъпки използват прилежащия транспортен протокол (UDP)

## ❖ Реконфигурация

- Новите възли получават данни за конструиране на маршрутизиращи таблици от наличните възли, обменяйки  $O(\log N)$  съобщения
- При повреда или премахване на възел наличните възли го идентифицират и обновяват маршрутизиращите си таблици

# Pastry маршрутизиращ алгоритъм: фаза 1

- ❖ Използва вектор  $L$  с размер  $2l$ , съдържащ GUID идентификатори и IP адреси на възлите, чиито GUID са числово най-близки до GUID идентификатора на текущия възел ( $\pm l$ )
- ❖ GUID пространство
  - Представя се с кръг, при който възли с GUID 0 и GUID  $2^{128} - 1$  са съседи



# Pastry маршрутизиращ алгоритъм: фаза 2

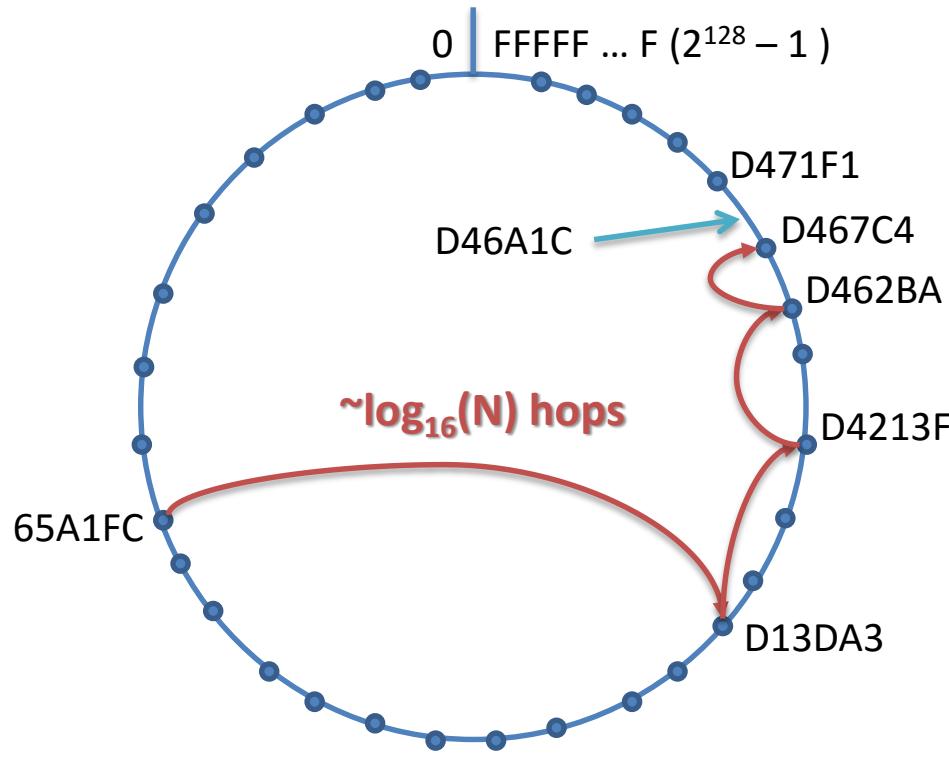
## ❖ Използва дървовидно структурирана таблица

- Съхранява GUID идентификатори и IP адреси на множество от възли, разпределени в целия диапазон от възможни GUID стойности ( $2^{128}$  на брой)
- Броят на редовете съответства на броя на шестнадесетичните цифри в GUID ( $128/4 = 32$ )
- Всеки ред има 15 записи – един за всяка възможна стойност на n-тата шестнадесетична цифра, като се изключва стойността за GUID на локалния възел

| p = | GUID prefixes and corresponding node handles n |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-----|------------------------------------------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 0   | 0                                              | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | A    | B    | C    | D    | E    | F    |
|     | n                                              | n    | n    | n    | n    | n    |      | n    | n    | n    | n    | n    | n    | n    | n    | n    |
| 1   | 60                                             | 61   | 62   | 63   | 64   | 65   | 66   | 67   | 68   | 69   | 6A   | 6B   | 6C   | 6D   | 6E   | 6F   |
|     | n                                              | n    | n    | n    | n    |      | n    | n    | n    | n    | n    | n    | n    | n    | n    | n    |
| 2   | 650                                            | 651  | 652  | 653  | 654  | 655  | 656  | 657  | 658  | 659  | 65A  | 65B  | 65C  | 65D  | 65E  | 65F  |
|     | n                                              | n    | n    | n    | n    | n    | n    | n    | n    | n    |      | n    | n    | n    | n    | n    |
| 3   | 65A0                                           | 65A1 | 65A2 | 65A3 | 65A4 | 65A5 | 65A6 | 65A7 | 65A8 | 65A9 | 65AA | 65AB | 65AC | 65AD | 65AE | 65AF |
|     | n                                              |      | n    | n    | n    | n    | n    | n    | n    | n    |      | n    | n    | n    | n    | n    |

# Pastry: Маршрутизиране на съобщение при фаза 2

- ❖ Маршрутизиране на съобщение от възел 65A1FC до възел D46A1C



# Pastry: Стъпки на маршрутизиращия алгоритъм

## ❖ Изпращане на съобщение M до възел D

- R[p,i] е елемент в колона i на ред p в маршрутизиращата таблица
- 1. *If ( $L_i < D < L_j$ ) { // дестинацията е в рамките на вектора или в текущия възел*
- 2. Препращане на M до елемент  $L_i$  във вектора, който е най-близък до D или до текущия възел A
- 3. *} else { // използва се маршрутизиращата таблица, за да се препрати M до възел с най-близък GUID*
- 4. Намиране на p, дължината на най-дългия общ префикс на D и A
- 5. *If ( $R[p,i] \neq null$ ) препращане на M до  $R[p,i]$  // изпращане на M до възел с най-голям общ префикс*
- 6. *else { // липсва запис в маршрутизиращата таблица*
- 7. Препращане на M до произволен възел в L или R с общ префикс с дължина p и с GUID, който е числено по-близък
- }

## ❖ Нов възел

- Използва протокол, за да получи съдържание за маршрутизиращата си таблица и вектор, и да нотифицира останалите възли за промените в техните таблици
  - ✓ Изчисляване на GUID (прилагане на SHA-1 върху публичен ключ)
  - ✓ Осъществяване на връзка с най-близък съседен възел (минимален брой мрежови преходи или минимално закъснение за предаване)

## ❖ Добавяне на възел с GUID X, осъществил връзка с възел A

- X изпраща на *join* заявка до A
- A диспечеризира *join* съобщението чрез Pastry до възел Z с GUID, който числово е най-близък до X
- Всички възли от A до Z изпращат релевантни части от съдържанието на маршрутизиращите си таблици и вектори до X
- X конструира своя маршрутизираща таблица и вектор и ги изпраща до останалите възли, за да реконфигурират своите маршрутизиращи таблици и възли

# Pastry: Изграждане на маршрутизираща таблица

- ❖ А е съсед на X, първият ред на таблицата за А се използва за първи ред на таблицата за X ( $X_0$ )
  - Таблицата на А не е релевантна за втория ред на таблицата за X ( $X_1$ )
- ❖ GUID идентификаторите на X и В споделят първата си шестнадесетична цифра
  - Вторият ред от таблицата на В се използва за втори ред на таблицата за X ( $X_1$ )
- ❖ По аналогичен начин се обработват и останалите възли
  - ...
- ❖ Идентификатора на Z е най-близък до идентификатора на X
  - Векторът на X се различава от вектора на Z по едно число
  - Векторът на Z се взима като първоначална апроксимация, която евентуално ще бъде оптимизирана
- ❖ Брой на съобщенията за добавяне на нов възел:  $O(\log N)$

# Pastry: Повреда или премахване на възел

## ❖ Повреден възел

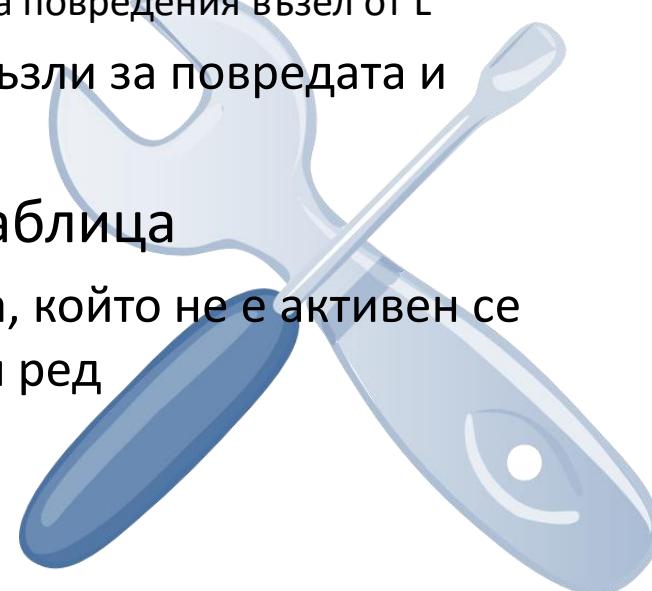
- Възел, чиито непосредствени съседи в GUID пространството не могат да комуникират с него

## ❖ Обновяване на вектор

- Възелът, идентифицирал повредата, открива активен възел, който е най-близо до повредения и изисква копие на вектора му  $L'$ 
  - ✓  $L'$  съдържа последователност от GUID, които частично покриват тези в  $L$
  - ✓ Възел от  $L'$  се използва за възстановяване на повредения възел от  $L$
- Уведомяване на останалите съседни възли за повредата и актуализиране на векторите им

## ❖ Обновяване на маршрутизираща таблица

- При откриване на елемент в таблицата, който не е активен се преминава към друг елемент от същия ред



## ❖ Характеристики на маршрутизиращата таблица

- Всеки ред притежава 16 клетки
- Стойностите в ред  $i$  представляват адресите на 15 възела с GUID идентификатори, при които първите  $i - 1$  шестнайсетични цифри съвпадат с тези на текущия възел

## ❖ Конструиране на маршрутизираща таблица

- Броят на възлите често надвишава размера на маршрутизиращата таблица
- За всяка позиция се избира един възел от множество кандидати
- Сравнението на кандидатите се извършва въз основа на локална метрика (алгоритъм за избор на близък съсед)
  - ✓ Брой IP преходи
  - ✓ Латентност

## ❖ Не се осигурява глобално оптимизиране на маршрутизацията

- Получените пътища са средно 30-50% по-дълги от оптималните

## ❖ Проверка за активни възли

- Изпращане на съобщения през определен интервал от време до съседните възли за удостоверяване на активност (heartbeat съобщения)

## ❖ Предотвратяване на проблеми

- Използване на механизъм за доставка at-least-once
  - ✓ Клиентите изпращат заявки многократно при липса на отговор
  - ✓ Осигуряване на по-дълъг времеви интервал за откриване на повреди

## ❖ Промяна на маршрутизиращия алгоритъм

- Случаен избор между малка част от случаи, които водят до общ префикс, който е по-малък от максималната дължина (стъпка 5)
- Повишаването на надеждността е за сметка на намалена оптималност

- ❖ Включване на потвърждения в маршрутизиращия алгоритъм
  - При получаване на съобщение текущия възел изпраща потвърждение
  - При липса на потвърждение изпращащия възел избира алтернативен път за съобщението
- ❖ Проверка за активни възли
  - Възстановяване на вектор
    - ✓ Всеки възел изпраща съобщение с индикация за активност до своя ляв съсед във вектора си (*heartbeat* съобщение)
    - ✓ Всеки възел записва времето за получаване на последното съобщение с индикация за активност от своя десен съсед
    - ✓ Ако текущият интервал от време надхвърли определен праг, се стартира процедура за възстановяване
  - Възстановяване на маршрутизираща таблица
    - ✓ Възлите, за които се предполага, че са повредени се определят по аналогичен начин на вече разгледания (*probe* съобщение)
    - ✓ Стартиране на gossip протокол на всеки 20 мин. за периодична обмяна на информация за рутиращите таблици

## ❖ Тестова среда

- Симулиране на мрежа от хостове на единична машина и закъснения при получаване на съобщения

## ❖ Надеждност

- Процент на изгубените IP съобщения: 0%
  - ✓ Неуспешно доставени 1.5 заявки от 100 000
  - ✓ Всички заявки са доставени до правилните възлите
- Процент на изгубените IP съобщения: 5%
  - ✓ Неуспешно доставени 3.3 заявки от 100 000
  - ✓ Некоректно доставени 1.6 заявки от 100 000

## ❖ Производителност

- Relative Delay Penalty: отношение между средните закъснения за доставяне на заявка посредством маршрутизиращия слой и за доставяне на подобно съобщение посредством UDP/IP
  - ✓ ~ 1.8 при 0% на изгубените IP съобщения
  - ✓ ~ 2.2 при 5% на изгубените IP съобщения

## ❖ Допълнително натоварване

- Допълнителното натоварване на мрежата генерирано от управляващия трафик е по-малко от 2 съобщения на минута за възел

# Tapestry: концепция

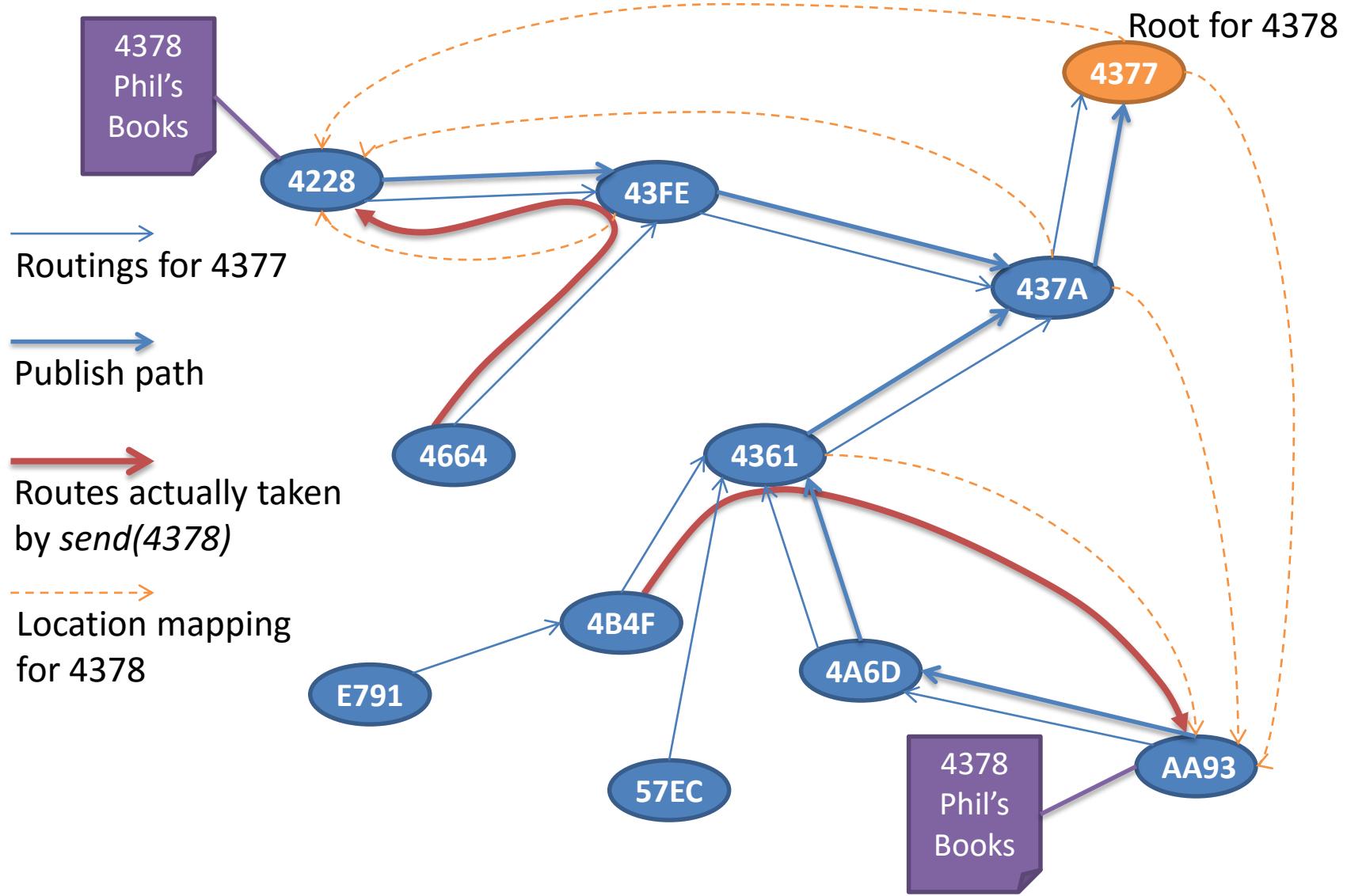
## ❖ Ключови характеристики

- Използване на разпределена хеш таблица
- Скриване на хеш таблицата зад DOLR приложен програмен интерфейс
- Репликиране на ресурсите посредством публикуване на ресурси с еднакъв GUID в няколко възела
- Поставяне на реплики в близост (мрежова) до потребителите, които най-често ги използват

## ❖ Идентификатори

- 160-битови идентификатори за ресурсите и възлите
  - ✓ Nodeld – идентификатор на компютър, извършващ рутиране
  - ✓ GUID – идентификатор на обект
- За всеки ресурс с GUID  $G$  съществува уникален коренов възел с GUID  $R_G$ , който числово е най-близо до  $G$
- Хост  $H$ , който съдържа реплики на  $G$ , периодично извиква  $publish(G)$ , за да гарантира, че новите хостове са осведомени за наличието на  $G$
- Съответствието  $(G, IP_H)$  се добавя в маршрутизиращите таблици на всички възли между  $H$  и  $R_G$  включително

# Tapestry: маршрутизация



# Структурирани и неструктурни P2P системи

## ❖ Структурирани P2P системи

- Наличие на глобална политика за управление на топологията на мрежата, добавянето и търсенето на обекти
  - ✓ Хеш таблици и кръгови структури при Pastry и Tapestry

## ❖ Неструктурирани P2P системи

- Липса на единен контрол върху топологията и добавянето на обектите
- Маршрутизиращият слой се създава динамично
- Установяване на контакт с множество от съседи при добавяне на възел
- Създадената мрежа от възли е децентрализирана, самоорганизираща се и гъвкава по отношение на повреди
- Подходът доминира в интернет при споделяне на файлове
  - ✓ Gnutella, FreeNet, BitTorrent

# Структурирани vs. неструктурни P2P системи

|              | Структурирани P2P системи                                                                                                     | Неструктурирани P2P системи                                                                                                                |
|--------------|-------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Преимущества | Гарантирано откриване на обекти; сравнително малко претоварване с допълнителни съобщения                                      | Самоорганизиращи се и гъвкави по отношение на повреди                                                                                      |
| Недостатъци  | Необходимост от поддържане на сложни маршрутизиращи слоеве, които могат да бъдат трудно достигими при среди с голяма динамика | Не осигуряват гаранции за откриване на обекти; предразположени към претоварване с допълнителни съобщения, което влияе върху скалируемостта |



# Стратегии за ефективно търсене

- ❖ Стратегия, прилагана при ранните версии на Gnutella (v. 0.4)
  - Всеки възел изпраща заявка до своите съседи, които я препращат до своите съседи и т.н. (flooding)
  - Времево ограничаване на търсенето и средна свързаност от 5 възела
- ❖ Разширяващо се кръгово търсене
  - Извършва се серия от търсения с нарастващо време на препращанията
- ❖ Случайно търсене
  - Създава се множество от “пешеходци”, които следват свои случаини пътища във взаимосвързан граф на неструктурирания маршрутизиращ слой
- ❖ Епидемичен подход
  - Изпращане на заявка до съседен възел с определена вероятност, след което заявките се разпространяват по мрежата под формата на вирус
    - ✓ Вероятността може да бъде фиксирана за мрежата или изчислявана динамично въз основа на предишен опит
- ❖ Поддръжка на **репликиращи техники** с цел повишаване на ефективността на търсенето
  - Репликация на цели файлове и разпределение на фрагменти от файлове в интернет (BitTorrent)

Gnutella

# CASE STUDY

❖ Промяна в чистата P2P концепция (v. 0.6): **редуциране на преходите**

- Създаване на специални възли с допълнителни ресурси (ultrapeer)
- Създаване на възли листа, свързани със специалните възли
- Свързване на специалните възли

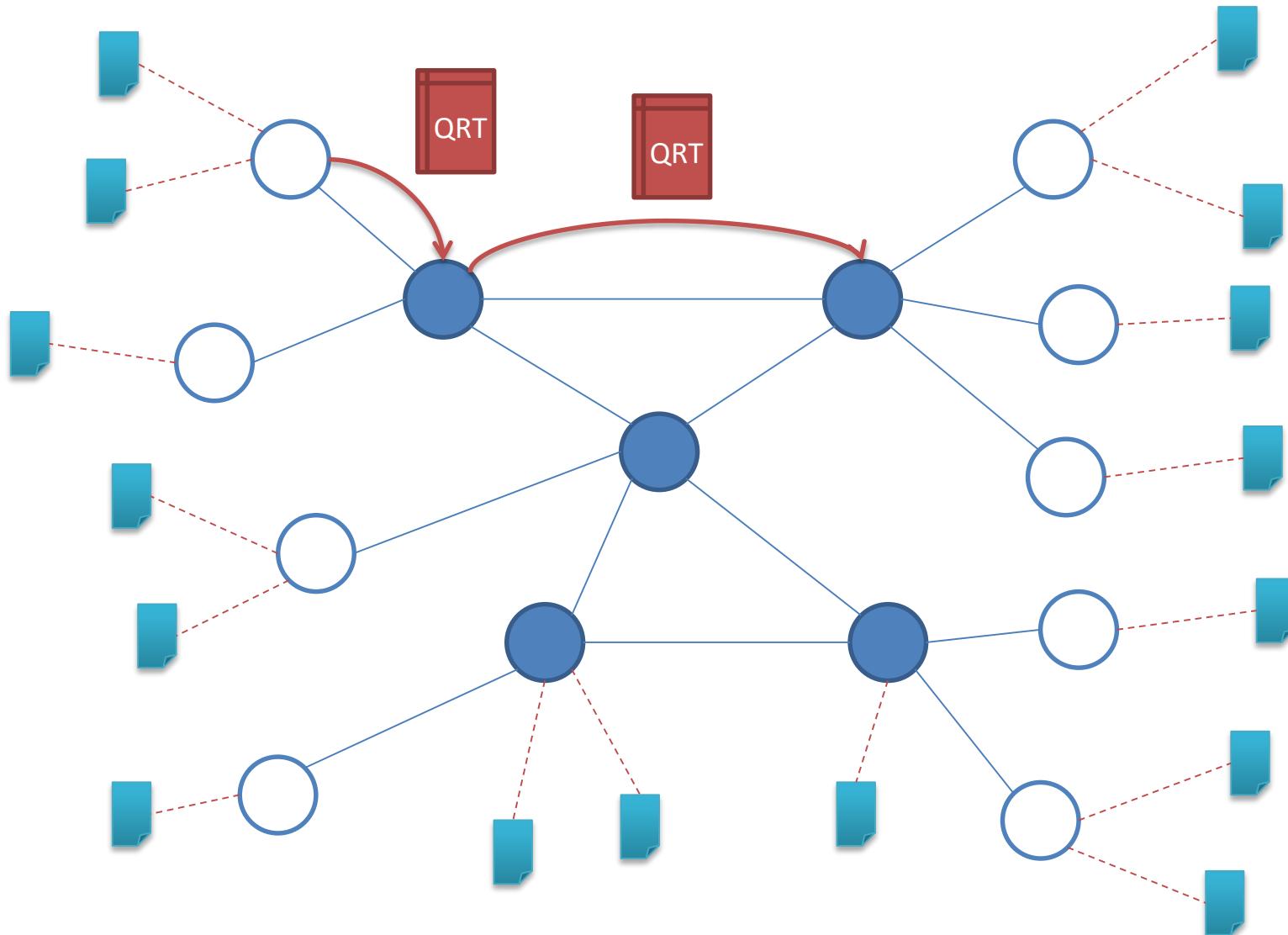
❖ Използване на протокол за маршрутизиране на заявки (Query Routing Protocol) : **редуциране на заявките**

- Липса на директно споделяне на информация за файловете
- Използване на множество от числа, получени при хеширане на думите във файловото име
  - ✓ “Chapter ten on P2P” → <65, 47, 09, 76>

## ❖ Използване на таблици за маршрутизиране на заявки (Query Routing Table)

- Създаване на QRT таблици от възлите листа и изпращане към специалите възли
- Създаване на QRT таблици в специалните възли посредством обединяване на QRT таблиците от възлите листа и собствените QRT таблици
- Обмяна на QRT таблиците между специалните възли
- Изпращане на единична заявка от възел към специален възел в даден момент
- Запазване на мрежовия адрес на иницииращия специален възел в заявката и връщане на директен отговор към него
  - ✓ Избягва се обратното обхождане на графа

# Gnutella: ключови елементи на рутирация протокол



Squirrel peer-to-peer web caching service

# CASE STUDY

## ❖ Нива на кеширане

- Кеш на браузъра в клиентската машина
- Прокси уеб кеш
  - ✓ Услуга стартирана на друг компютър в същата локална мрежа или съседен възел в интернет
- Кеш на сървъра, към който е изпратена заявка от клиентския браузър

## ❖ Получаване на GET заявка в кеша на браузъра или прокси уеб кеша

- Заявеният обект не подлежи на кеширане (препращане на заявката към сървъра)
- Липсва запис на обекта в кеша (препращане на заявката към сървъра)
- Обектът е записан в кеша (проверка за актуалност)

# Съхраняване на уеб обекти

- ❖ Метаданни в уеб сървърите и кеш сървърите
  - **timestamp**: дата на последна модификация (T)
  - **time-to-live**: време на живот (t)
  - **eTag**: хеш стойност, изчислена върху съдържанието на уеб страница
- 1. Връщане на обект от кеша към клиента
  - Актуален обект:  $T + t >$  текущо време
- 2. Условна GET заявка (cGET)
  - If-Modified-Since: съдържа дата на последна модификация
  - If-None-Match: съдържа eTag
  - Обслужва се от друг уеб кеш или от оригиналния сървър
  - Отговорът съдържа обект или съобщение за липса на модификация
- ❖ Актуализиране на кеша при получаване на обект от оригиналния сървър
  - Замяна на старите обекти с нови и актуализиране на метаданните

# Squirrel: концепция

## ❖ Предназначение

- Услуга за уеб кеширане, използваща малка част от ресурсите на клиентските компютри в локална мрежа

## ❖ Характеристики

- Използване на 128-битов Pastry GUID
  - ✓ Прилагане на SHA-1 хеш функция върху URL адреса на кешираните обекти
- Съхраняването на кешираните копия на обектите във възел с GUID, който е най-близък до GUID на обекта (*home node*)
- Клиентските възли притежават локален Squirrel прокси процес, който извършва локалното и отдалеченото кеширане на обекти

## ❖ Функциониране

- Липса на актуално копие в локалния кеш
  - ✓ Изпращане на GET или cGET заявка до “*home*” възел
- Липса или наличие на неактуално копие в “*home*” възела
  - ✓ Изпращане на GET или cGET заявка до оригиналния сървър

# Squirrel: оценка

## ❖ Цел на изследването

- Сравняване на Squirrel уеб кеш с централизиран кеш
- Използване на реални работни среди в Кембридж (105 клиенти) и Редмонд (повече от 36 000 клиенти)

## ❖ Параметри на изследването

- Намаляване на външния трафик
  - ✓ Обратно пропорционален на броя на изпълнените заявки от кеша
  - ✓ Ниско понижаване на външния трафик (~1%)
- Латентност при достъп на обектите в кеша
  - ✓ Наличие на допълнителен трансфер на съобщения в локалната мрежа между клиента и хоста, кеширал обекта
  - ✓ Локалният трафик се компенсира от модерния етернет хардуер
- Допълнително натоварване на клиентските възли
  - ✓ Ниски нива на допълнително консумирани системни ресурси

OceanStore

# CASE STUDY

# OceanStore: концепция

## ❖ Предназначение

- Съхраняване и споделяне голям брой данни обекти при постоянно променяща се мрежа и изчислителни ресурси
- Репликирано съхраняване на променливи и постоянни обекти

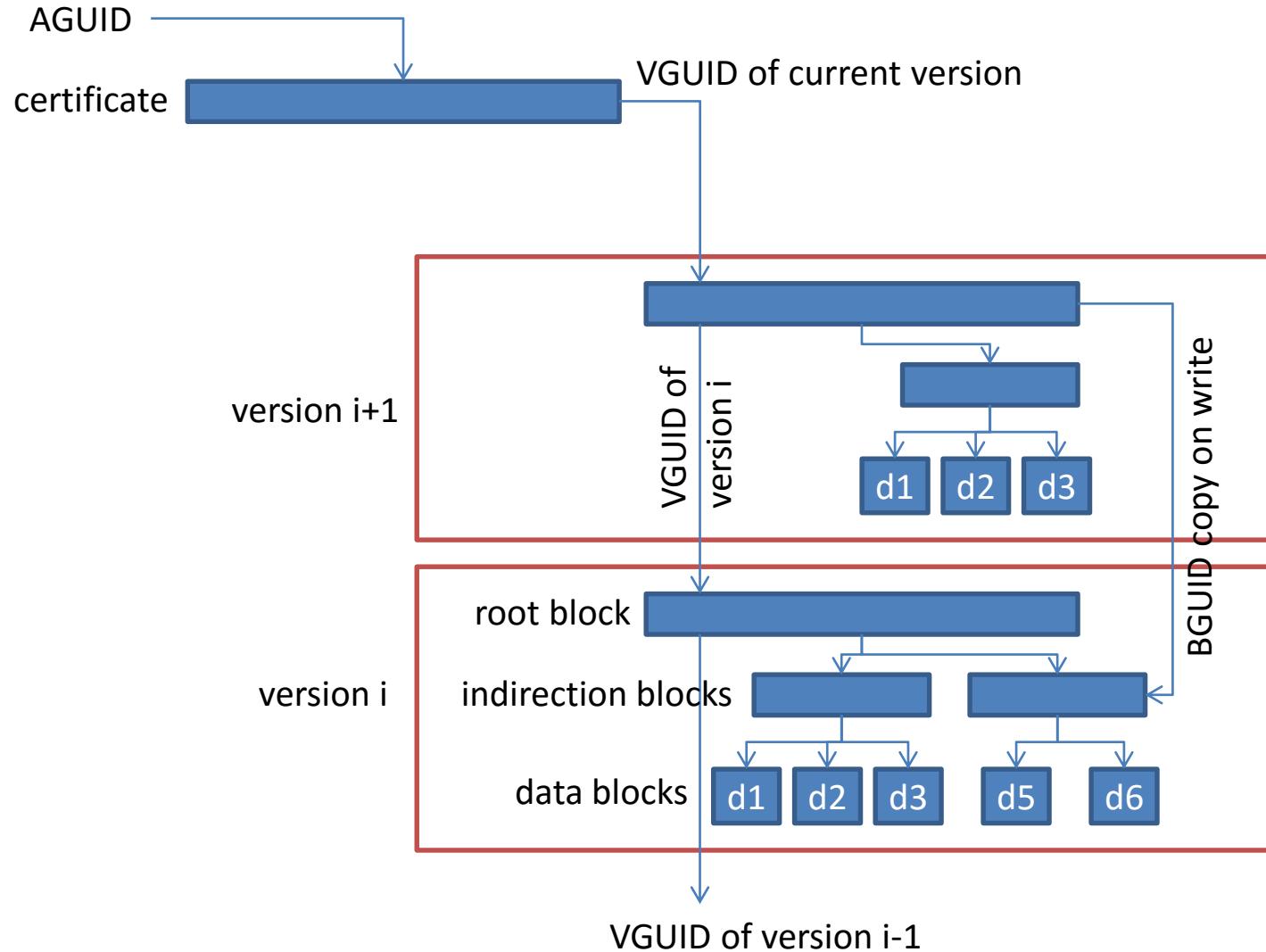
## ❖ Данни обекти

- Множество от данни блокове
  - ✓ Съхраняват данни на даден обект
- Обектът се представя с последователност от непроменливи версии, които се съхраняват постоянно
  - ✓ Версии са споделят непроменяемите блокове
- Коренов блок
  - ✓ Блок с метаданни, който организира достъпа до данните блокове
- Индиректен блок
  - ✓ Асоциира текстово или друго външно видимо име с последователност от версии на даден обект

## ❖ Типове идентификатори

- BGUID: Идентификатор на блок
- VGUID: Идентификатор на версия
- AGUID: Идентификатор на всички версии за даден обект

# OceanStore: организация на хранилището



# OceanStore: идентификатори

## ❖ BGUID и VGUID

- Изчисляват се от съдържанието на релевантния блок с хеш функция
- Използва се за автентикация и проверка на интегритета на съдържанието

## ❖ AGUID

- Изчислява се посредством прилагане на хеш функция върху специфично за приложението име, доставено от клиента, и публичен ключ, представящ собственика на обекта

## ❖ Сертификат

- Съхранява съответствие между AGUID и версии на обекта
- Съдържа VGUID на текущата версия

## ❖ Коренов блок за дадена версия

- Съдържа VGUID на предходната версия

## ❖ При създаване на нова версия на обекта се генерира нов сертификат

- Съдържа VGUID на новата версия

# OceanStore: оценка

| Phase | LAN       |      | WAN       |      | Predominant operation in benchmark |
|-------|-----------|------|-----------|------|------------------------------------|
|       | Linux NFS | Pond | Linux NFS | Pond |                                    |
| 1     | 0.0       | 1.9  | 0.9       | 2.8  | Read and write                     |
| 2     | 0.3       | 11.0 | 9.4       | 16.8 | Read and write                     |
| 3     | 1.1       | 1.8  | 8.3       | 1.8  | Read                               |
| 4     | 0.5       | 1.5  | 6.9       | 1.5  | Read                               |
| 5     | 2.6       | 21.0 | 21.5      | 32.0 | Read and write                     |
| Total | 4.5       | 37.2 | 47.0      | 54.9 |                                    |

- ❖ Фази (емуляция на натоварването при разработка на софтуер)
  - Фаза 1: рекурсивно създаване на поддиректории
  - Фаза 2: копиране на дърво с файлове
  - Фаза 3: проверка на статуса на всички файлове в дървото без да се обработват данните им
  - Фаза 4: обработка на всеки байт от данни във всички файлове
  - Фаза 5: компилиране и свързване на файлове
- ❖ Извод: Интернет базираната файлова услуга на OceanStore предлага ефективно решение за дистрибуция на файлове, които не се променят много често (кеширани копия на уеб страници)





## РАЗПРЕДЕЛЕНИ ТРАНЗАКЦИИ

доц. д-р Десислава Петрова-Антонова

# Съдържание

- ❖ Въведение
- ❖ Плоски и вложени транзакции
- ❖ Атомарни протоколи
- ❖ Управление на конкурентността в разпределените транзакции
- ❖ Разпределени мъртви хватки
- ❖ Възстановяване на транзакции

# Разпределени транзакции: същност

## ❖ Дефиниция

- Транзакции, които достъпват обекти, управявани от множество сървъри

## ❖ Изискване за атомарност

- Транзакцията се изпълнява върху всички сървъри или не се изпълнява върху нито един от тях

## ❖ Координатор на разпределена транзакция

- Изпълнява се от сървър, участващ в транзакцията

## ❖ Управление на конкурентността

- Локално управление на конкурентността в сървърите
- Глобално сериализиране на транзакцията

## ❖ Възстановяване от транзакция

- Всички обекти, участващи в транзакцията трябва да са възстановяни
- Всички промени при успешна транзакция се отразяват върху обектите и обратно

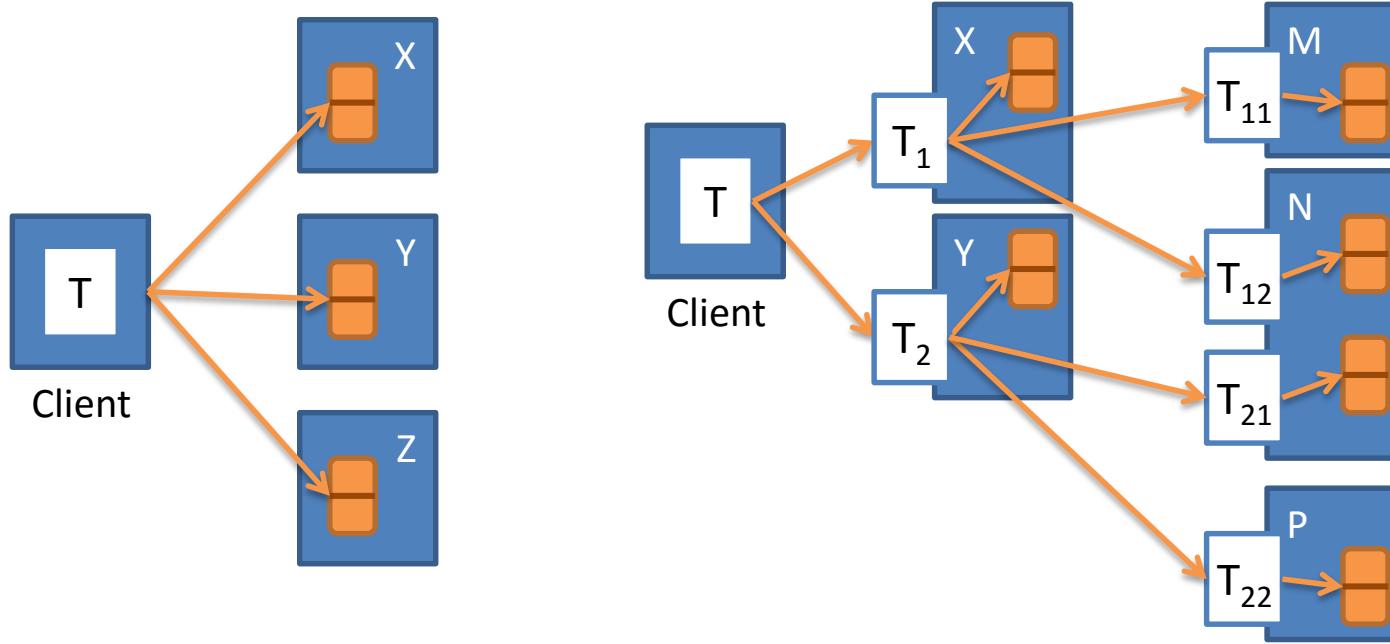
# Плоски и вложени транзакции

Плоски  
транзакции

Вложени  
транзакции

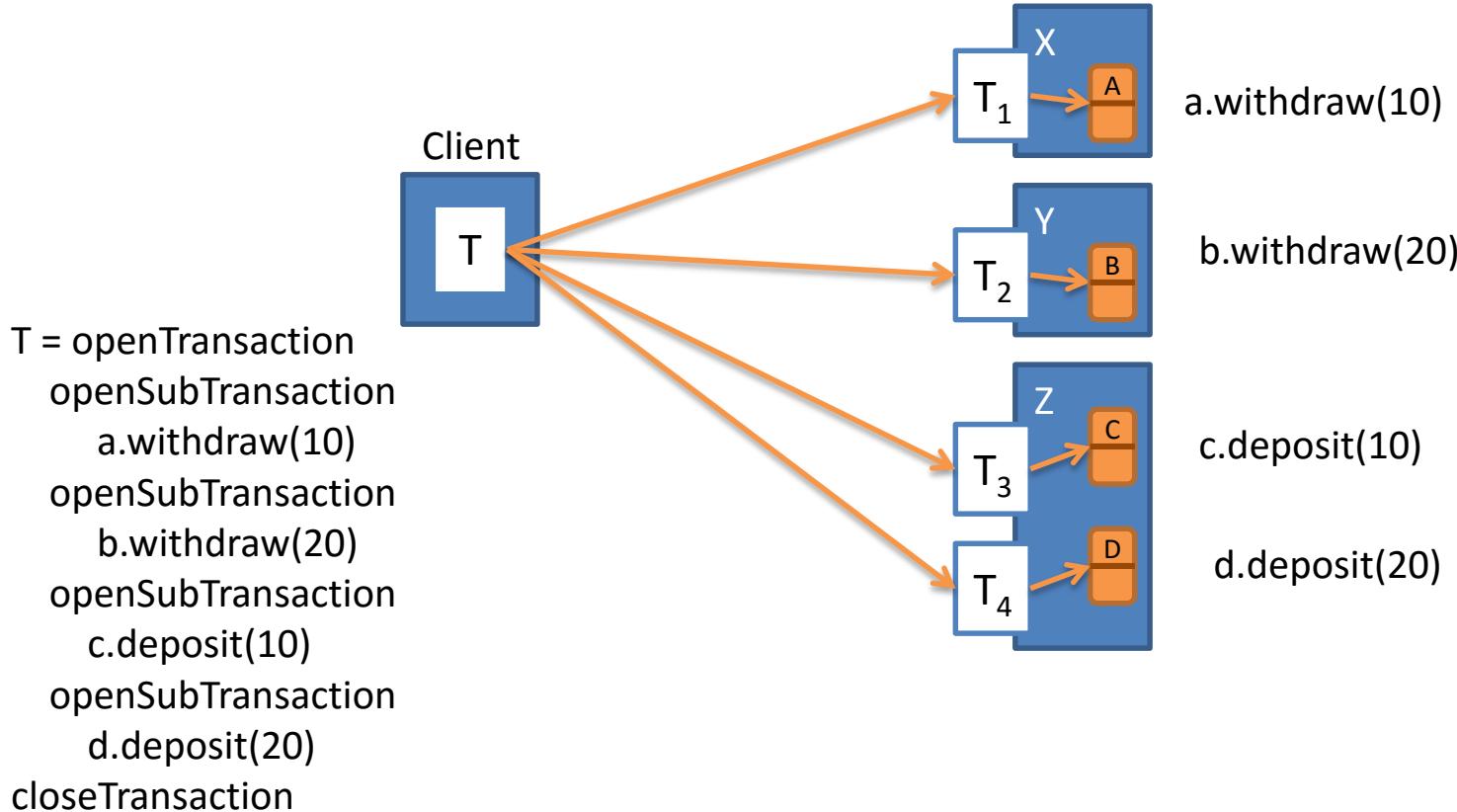
Разпределени  
транзакции

# Плоски и вложени транзакции



- ❖ Клиентът изпраща заявка до повече от един сървър
  - Последователно изпълнение на заявките от транзакцията в сървърите
- ❖ Транзакциите от по-високо ниво могат да отварят подтранзакции
  - Подтранзакциите с еднакво ниво могат да се изпълняват паралелно

# Вложени транзакции: Производителност



## ❖ Вложени транзакции: превод по сметка

- А, В, С и D са акаунти съответно в сървъри X, Y и Z
- Трансфер на \$10 от A в C
- Трансфер на \$20 от B в D

# Координатор на разпределена транзакция

## ❖ Отваряне на транзакцията

- Клиентът изпраща заявка *openTransaction* до произволен координатор
- Координаторът отваря транзакцията и връща нейн идентификатор (TID)
  - ✓ TID включва идентификатор на сървър, създал транзакцията, и номер уникален за сървъра

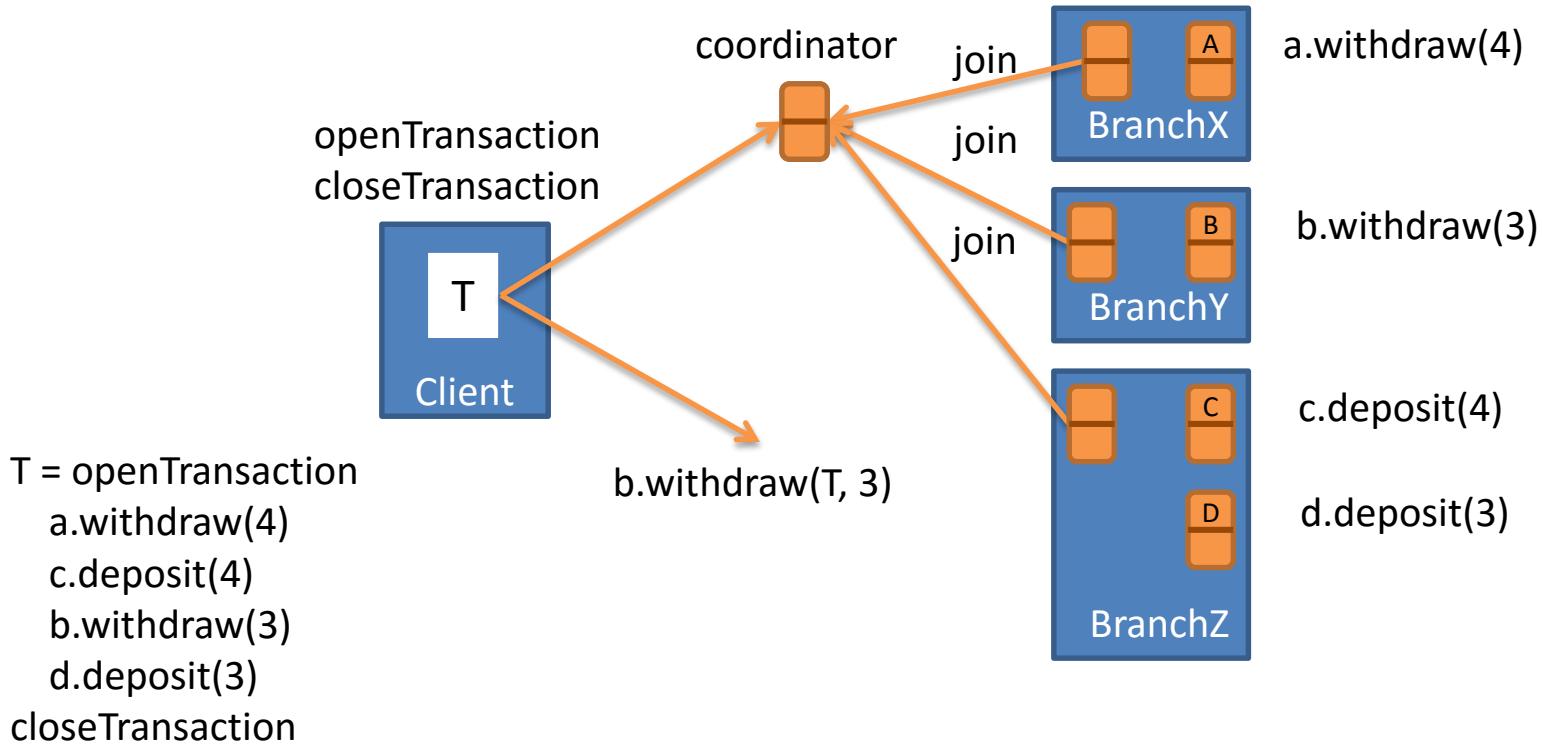
## ❖ Роли при разпределена транзакция

- Координатор: сървър, който отваря транзакцията
  - ✓ Съхранява референции към участниците
- Участници: сървъри, предоставящи обекти в транзакцията
  - ✓ Съхраняват референция към координатора

## ❖ Присъединяване към транзакция

- Метод *join*: информира координатора за нов участник в транзакция *Trans*
  - ✓ *join(Trans, reference to participant)*

# Плоски транзакции: Пример



## ❖ Плоска транзакция: превод по сметка

- А, В, С и D са акаунти съответно в сървъри BranchX, BranchY и BranchZ
- Трансфер на \$4 от А в С
- Трансфер на \$3 от В в D

# Атомарни протоколи: видове

## ❖ Еднофазов протокол

- Клиентът изпраща заявка за приключване или отхвърляне на транзакция
- Координаторът препраща заявката до всички участници докато не получи потвърждение от тях за нейното изпълнение
- Недостатък: сървърите не могат да взимат самостоятелно решение за отхвърляне на своите части от транзакцията

## ❖ Двуфазов протокол

- Всеки сървър може да охвърли своята част от транзакцията
- Фаза 1: Събиране на информация от сървърите за изхода от транзакцията и подготовка за приключване, ако транзакцията е успешна
- Фаза 2: Взимане на решение за изхода от транзакцията

## ❖ Проблеми

- Срив на сървър или загубана съобщения
  - ✓ Решения на ниво протокол за заявка-ответ
- Срив на процес
  - ✓ Съхраняване на състоянието на процесите

# Двуфазов протокол: същност

## ❖ Отхвърляне на транзакция

- Заявява се от клиента или от участник в транзакцията
- Координаторът информира останалите участници за отхвърлената транзакция

## ❖ Приключване на транзакция

- Изпълнява се двуфазовият протокол
- `canCommit?(trans) → Yes / No`
  - ✓ Запитване за приключване от координатора към участник
- `doCommit(trans)`
  - ✓ Заявка за приключване на транзакция от координатора към участник
- `doAbort(trans)`
  - ✓ Заявка за отхвърляне на транзакция от координатора към участник
- `haveCommitted(trans, participant)`
  - ✓ Запитване за потвърждение на приключена транзакция от координатора към участник
- `getDecision(trans) → Yes / No`
  - ✓ Запитване за изход на транзакция от участник към координатора в случай на сървърна повреда или закъснение на съобщения

# Двуфазов протокол: последователност на изпълнение

## ❖ Фаза 1 (гласуване)

1. Координаторът изпращат заявка *canCommit?* до участниците
2. Участниците отговарят с Yes или No на координатора
  - ✓ При отговор Yes обектите се съхраняват
  - ✓ При отговор No участникът отхвърля транзакцията незабавно

## ❖ Фаза 2 (изход въз основа на резултата от гласуването)

3. Координаторът анализира гласуването
  - a) Успешно приключване на транзакцията при единодушно гласуване с Yes
  - b) Отхвърляне на транзакцията при наличие на поне един глас No
4. Участниците, гласували с Yes, очакват заявка *doCommit* или *doAbort*
  - ✓ При успешна транзакция участниците изпращат заявка *haveCommitted* като потвърждение на координатора

## ❖ Възстановяване от срив

- Съхраняване на информацията за двуфазовия протокол
- Използване на интервали за изчакване

# Закъснения при изпълнение на двуфазовия протокол

## ❖ Състояние на неопределеност

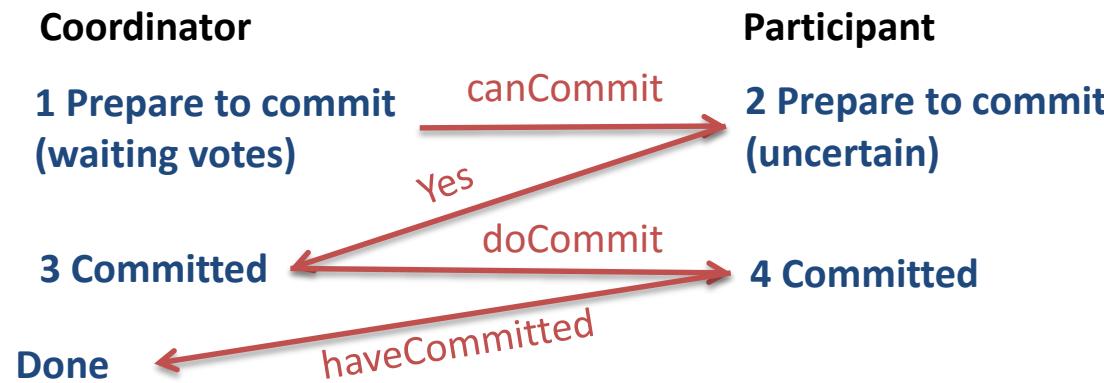
- Участник гласува с Yes и **изчаква** координатора да вземе решение за изхода на транзакцията
- Изпращане на заявка *getDecision* от участника към сървъра
- При срив на координатора участникът не може да вземе решение и изпада в **състояние неопределеност**

## ❖ Състояние на неопределеност при липса на заявка *canCommit?*

- Отхвърляне на транзакцията от участник след определен интервал от време

## ❖ Състояние на неопределеност при изчакване на резултата от гласуването от страна на координатора

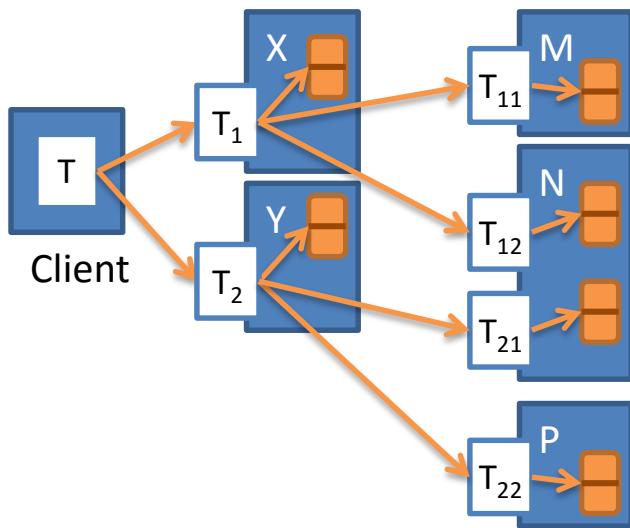
- Отхвърляне на транзакцията от координатора



# Производителност на двуфазовия протокол

- ❖ Производителност при N участници и успешна транзакция
  - Брой на съобщенията:  $3N$ 
    - ✓ N заявки canCommit
    - ✓ N отговора
    - ✓ N заявки doCommit
  - Брой на циклите: 3
- ❖ Намаляване на производителността при състояние на неопределеност
  - Закъснение при срив в координатора
    - ✓ Липса на отговор за заявката getDecision към координатора
  - Закъснение при срив в участник
    - ✓ Липса на отговор за заявката getDecision към участник
- ❖ Закъсненията се преодоляват при трифазовия протокол за сметка на повишен брой на съобщенията и циклите

## Вложени транзакции



### ❖ Транзакция от високо ниво

- Най-външната транзакция при вложени транзакции

### ❖ Подтранзакция

- Поддържа временно приключване на транзакция, при което състоянието не се съхранява
- Временно приключилите транзакции участват в двуфазовия протокол

### ❖ Операции на координатора за вложени транзакции

- `openSubTransaction(trans) → subTrans`
  - ✓ Отваря нова подтранзакция с родител trans и връща уникален идентификатор на подтранзакция
- `getStatus(trans) → committed, aborted, provisional`
  - ✓ Заявка към координатора за статус на транзакция trans

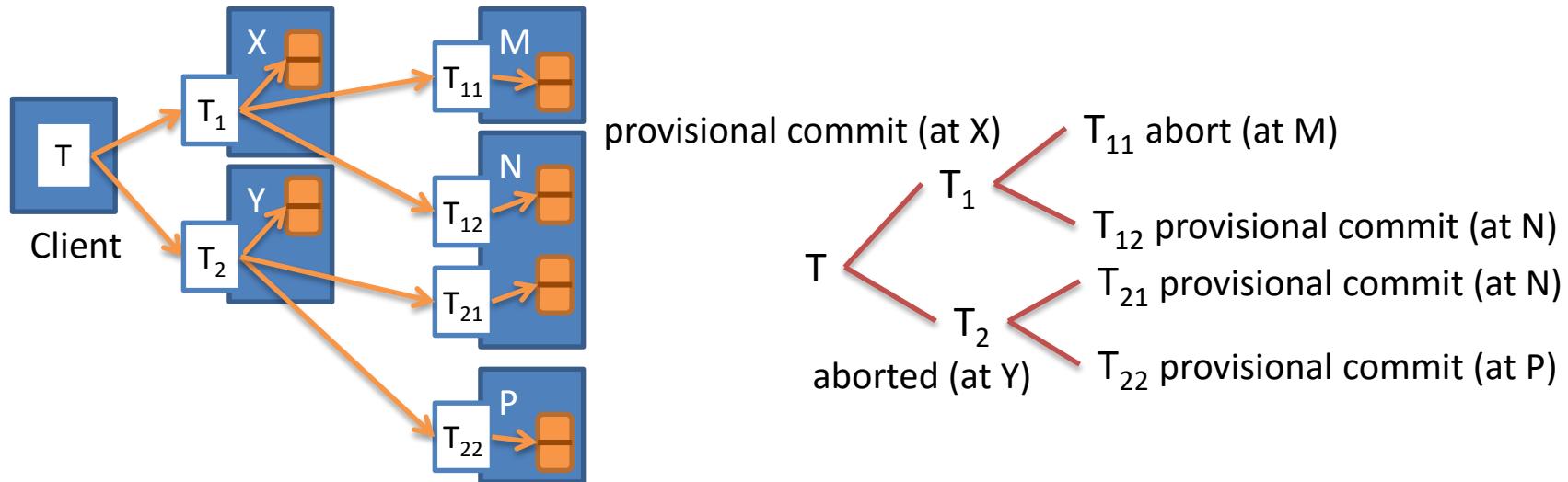
# Изпълнение на вложени транзакции (1-2)

- ❖ Клиентът отваря транзакция от високо ниво с *openTransaction*
  - Получаване на TID
- ❖ Клиентът отваря подтранзакция с *openSubTransaction*
- ❖ Подтранзакцията се присъединява към родителската транзакция с метода *join*
  - Получаване на subTID
- ❖ Клиентът извиква *closeTransaction* или *abortTransaction* при координатора на транзакцията от високо ниво
- ❖ Ако родителска транзакция е неуспешна, то дъщерните транзакции също се считат за неуспешни
- ❖ Транзакция от високо ниво може да приключи успешно при неуспешно приключили подтранзакции

# Изпълнение на вложени транзакции (2-2)

## ❖ Изход от родителката транзакция при неуспешна дъщерна транзакция

- Пример: Вложени транзакции при банкови трансфери
  - ✓ Подтранзакции Transfer с подтранзакции deposit и withdraw
  - ✓ Неуспех на една Transfer подтранзакция може да **не** доведе до неуспех на останалите
- Пример: Транзакция T с подтранзакция T<sub>1</sub>, която е временно приключила и подтранзакция T<sub>2</sub>, която е неуспешна



# Изпълнение на двуфазов протокол при вложена транзакция

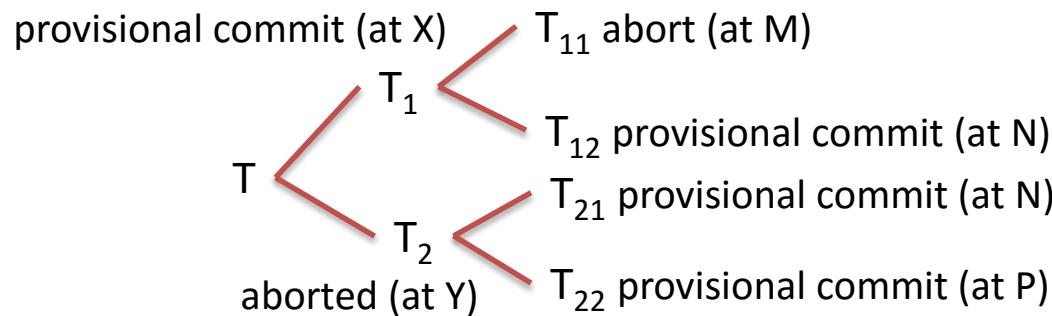
## ❖ Роли на координаторите

- Координаторът на транзакцията от високо ниво управлява изпълнението на двуфазовия протокол
- Координаторът на всяка родителска транзакция притежава списък на дъщерните си подтранзакции
  - ✓ При **успех** дадена подтранзакция рапортува статуса си на родителската транзакция заедно със статуса на подтранзакциите си
  - ✓ При **неуспех** дадена подтранзакция рапортува само своя статус на родителската транзакция

| Coordinator of transaction | Child transaction | Participant               | Provisional commit list | Abort list    |
|----------------------------|-------------------|---------------------------|-------------------------|---------------|
| T                          | $T_1, T_2$        | Yes                       | $T_1, T_{12}$           | $T_{11}, T_2$ |
| $T_1$                      | $T_{11}, T_{12}$  | Yes                       | $T_1, T_{12}$           | $T_{11}$      |
| $T_2$                      | $T_{21}, T_{22}$  | No (aborted)              |                         | $T_2$         |
| $T_{11}$                   |                   | No (aborted)              |                         | $T_{11}$      |
| $T_{12}, T_{21}$           |                   | $T_{12}$ but not $T_{21}$ | $T_{21}, T_{12}$        |               |
| $T_{22}$                   |                   | No (parent aborted)       | $T_{22}$                |               |

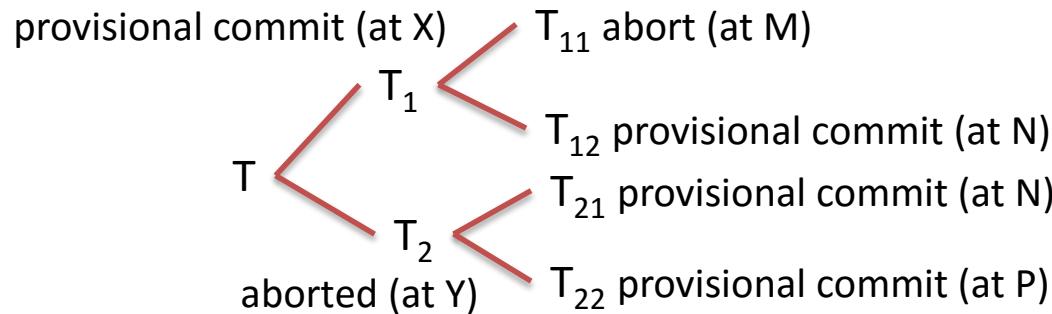
# Двуфазов протокол при вложена транзакция

- ❖ Координатор на двуфазовия протокол
  - Транзакция от най-високо ниво
- ❖ Участници в двуфазовия протокол
  - Всички подтранзакции, които са временно успешно приключили и нямат неуспешно приключили родителски транзакции
  - Пример: Изпълнение на двуфазовия протокол за  $T$ ,  $T_1$  и  $T_{12}$
- ❖ Особености на изпълнение на фаза 1
  - Подтранзакциите съхраняват състоянието на своите обекти като част от транзакцията на високо ниво



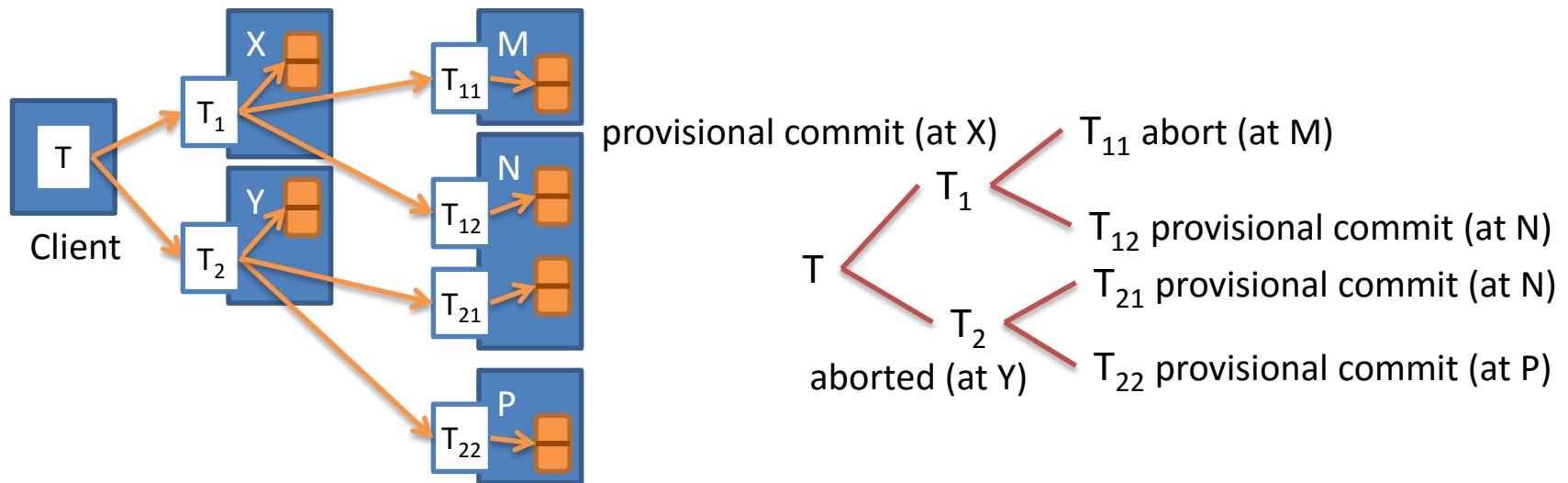
# Йерархичен двуфазов протокол

- ❖ Координаторът от високо ниво изпраща *canCommit?* съобщения само на преките си наследници
- ❖ Родителските координатори изпращат на координаторите наследници *canCommit?* съобщения
  - *canCommit(trans, subTrans)* → Yes / No
- ❖ Всеки координатор събира резултатите, от своите наследници
- ❖ Параметри на *canCommit?*
  - trans: TID на транзакцията от високо ниво
  - subTrans: TID на участник
  - Ако участник открие подтранзакция със subTrans TID, то той отговаря с Yes, в противен случай – с No



# Плосък двуфазов протокол (1-2)

- ❖ Координаторът от високо ниво изпраща *canCommit?* съобщения на всички координатори на временно приключили транзакции ( $T_1$  и  $T_{12}$ )
- ❖ Участниците използват TID на транзакцията от високо ниво
- ❖ Недостатък
  - Опасност от некоректни действия на участниците
    - ✓ Координаторът на N може да поиска успешно приключване на подтранзакциите  $T_{12}$  и  $T_{21}$



# Плосък двуфазов протокол (2-2)

## ❖ Реализация на *canCommit?* за плосък двуфазов протокол

- *canCommit?(trans, abortList) → Yes / No*
  - ✓ Параметър *abortList*: списък с неуспешни подтранзакции

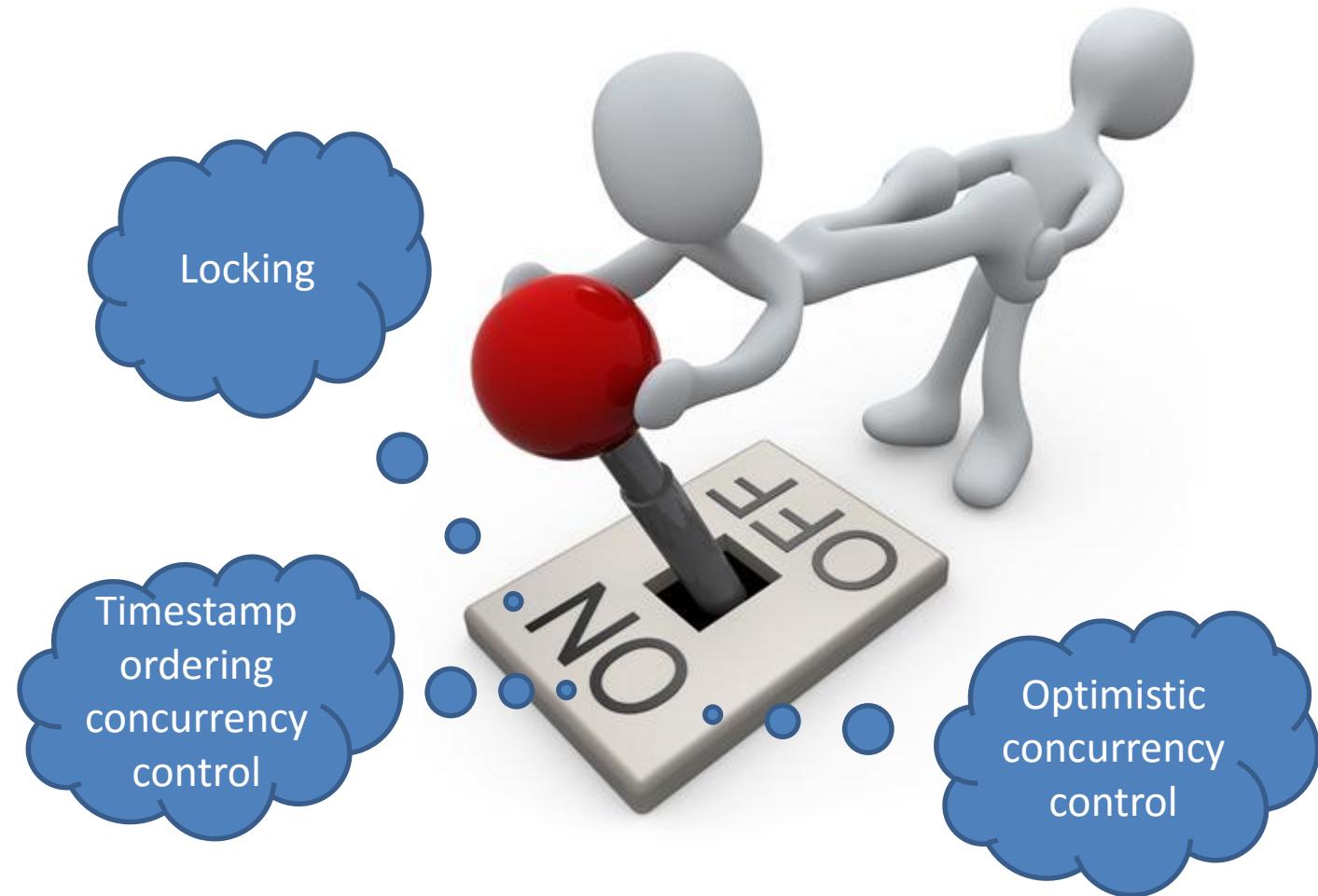
## ❖ Отговор на участник

- Участникът **има** временно приключили транзакции, които са наследници на транзакцията от високо ниво
  - ✓ Проверява за неуспешни родителски транзакции в списъка *abortList* и инициира подготовка за приключване
- Участникът **няма** временно приключили транзакции, които са наследници на транзакцията от високо ниво
  - ✓ Изпраща No на координатора поради наличие на повреда

## ❖ Задаване на интервали за изчакване

- Всяка подтранзакция, която не е получила съобщение *canCommit?* може да изпрати заявка *getStatus* на своя родител
  - ✓ Ако временно успешно приключила подтранзакция не може да установи контакт със своя родител за определен период от време, то тя трябва да бъде отхвърлена
  - ✓ Пример:  $T_{22}$  е временно успешно приключила, но има неуспешна родителска транзакция  $T_2$

# Управление на конкурентен достъп до обекти



# Заключване

- ❖ Заключването се извършва локално
- ❖ Обектите се освобождават при приключване на транзакцията във всички сървъри
  - Обектите на даден сървър остават заключени при изпълнението на атомарния двуфазов протокол
- ❖ Заключването в отделните сървъри е независимо
  - Опасност от възникване на мъртва хватка
  - Пример: Транзакции T и U в сървъри X и Y

| Transaction T            | Transaction U            |
|--------------------------|--------------------------|
| Write(A) at X locks A    |                          |
|                          | Write(B) at Y lock B     |
| Read(B) at Y waits for U |                          |
|                          | Read(A) at X waits for T |

# Управление на конкурентността чрез времеви индикатор

- ❖ Използване на глобален уникален времеви индикатор (timestamp)
  - Получава се от клиента на първия координатор достъпен чрез транзакция
- ❖ Взаимна отговорност на сървърите в разпределената транзакция по отношение на конкурентния достъп
  - Транзакции T и U в даден сървър достъпват даден обект последователно
  - Същата последователност за достъп до обектите се спазва и в останалите сървъри
- ❖ Съдържание на времения индикатор, използван от координаторите за осигуряване на ред в изпълнението на транзакциите
  - <local timestamp, server-id>
- ❖ Не се изисква задължително синхронизиране на локалните часовници
  - Повишаване на ефективността при наличие на синхронизация

## ❖ Фази в изпълнението на транзакциите

- Работна фаза
  - ✓ Транзакциите разполагат с временни обекти, върху които изпълняват операции
- Фаза на валидация
  - ✓ Установява се наличие на конфликти между операциите изпълнявани върху едни и същи обекти от различни транзакции
  - ✓ Използват се транзакционни номера
- Фаза на обновяване
  - ✓ При успешна валидация временните промени в обектите стават постоянни

## ❖ Валидация при разпределените транзакции

- Извършва се през първата фаза на двуфазовия протокол

| Transaction T | Transaction U |
|---------------|---------------|
| Read(A) at X  | Read(B) at Y  |
| Write(A)      | Write(B)      |
| Read(B) at Y  | Read(A) at X  |
| Write(B)      | Write(A)      |

### Commitment deadlock

T before U at X  
U before T at Y  
X validates T first  
Y validates U first

# Оптимистично управление на конкурентността 2-3

## ❖ Правила за валидация при транзакции в един сървър

| $T_v$ | $T_i$ | Правило | Описание                                                                                                     |
|-------|-------|---------|--------------------------------------------------------------------------------------------------------------|
| Write | Read  | 1       | $T_i$ не трябва да чете обекти записани от $T_v$                                                             |
| Read  | Write | 2       | $T_v$ не трябва да чете обекти записани от $T_i$                                                             |
| Write | Write | 3       | $T_i$ не трябва да записва обекти записани от $T_v$<br>и $T_v$ не трябва да записва обекти записани от $T_i$ |

- ❖ Прилагане на паралелна валидация при разпределени транзакции
- ❖ Правило 3 и правило 2 се подлагат на обратна валидация
  - Множеството от write операции на транзакцията, която се валидира, се проверяват за припокриване с множеството от write операции на по-ранните транзакции

# Оптимистично управление на конкурентността 3-3

## ❖ Опастност от различно сериализиране на едни и същи транзакции в различните сървъри

- T before U at X
- U before T at Y

## ❖ Техники за сериализация на транзакции

- Извършване на глобална валидация след локалната
  - ✓ Проверка, че валидираните транзакции не участват в цикли
- Използване на глобален уникален транзакционен номер
  - ✓ Генерира се от координатора на двуфазовия протокол
  - ✓ Изпраща се към участниците с *canCommit?* съобщение
  - ✓ Изисква се споразумяване на сървърите относно генерираните от тях транзакционни номера

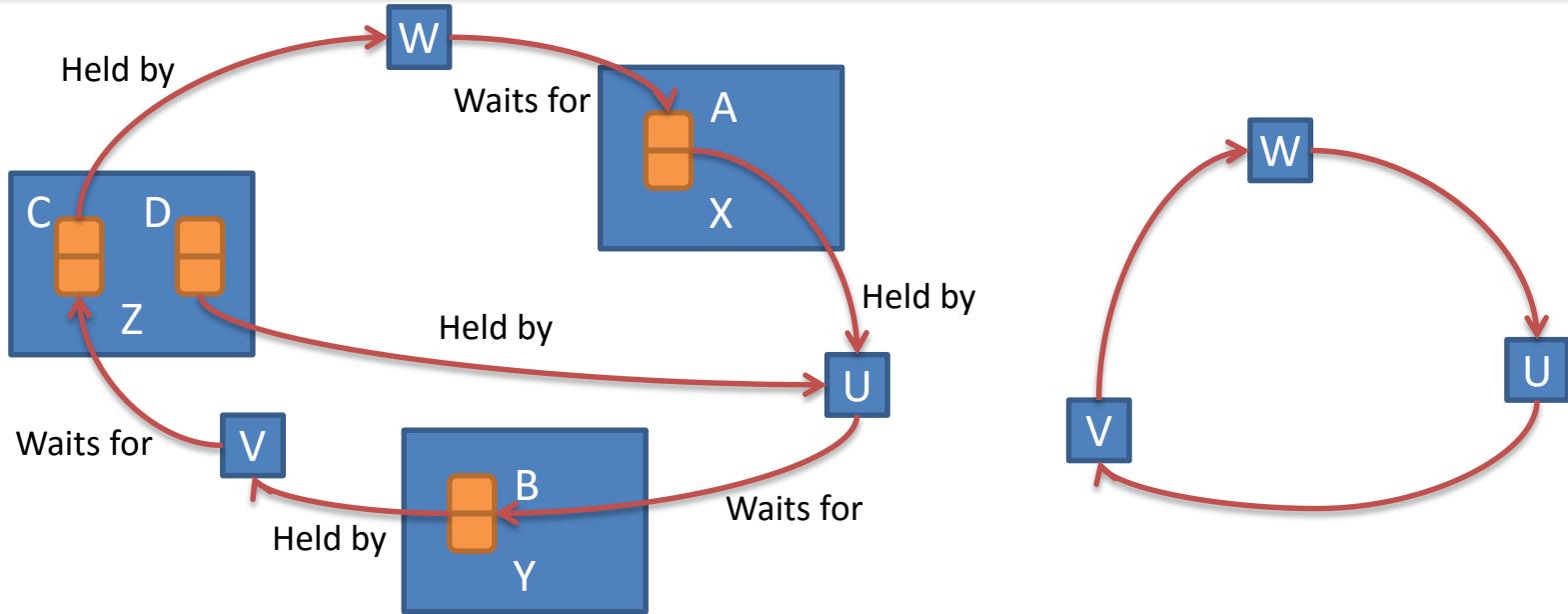


# Разпределени мъртви хватки: концепция 1-2

- ❖ Откриване на разпределени мъртви хватки
  - Конструиране на глобален транзакционен wait-for граф от локалните такива
- ❖ Wait-for граф
  - Възлите представлят транзакции и обекти
  - Дъгите представлят обект, използван в транзакция, или транзакция, изчакваща обект
- ❖ Пример за припокриване на транзакции
  - Транзакции U, V и W
  - Обекти A и B, управлявани от сървъри X и Y
  - Обекти C и D, управлявани от сървър Z

| U                         | V                         | W                         |
|---------------------------|---------------------------|---------------------------|
| d.Deposit(10) lock D      |                           |                           |
|                           | b.Deposit(10) lock B at Y |                           |
| a.Deposit(20) lock A at X |                           |                           |
|                           |                           | c.Deposit(30) lock C at Z |
| b.Withdraw(30) wait at Y  |                           |                           |
|                           | c.Withdraw(20) wait at Z  |                           |
|                           |                           | a.Withdraw(20) wait at X  |

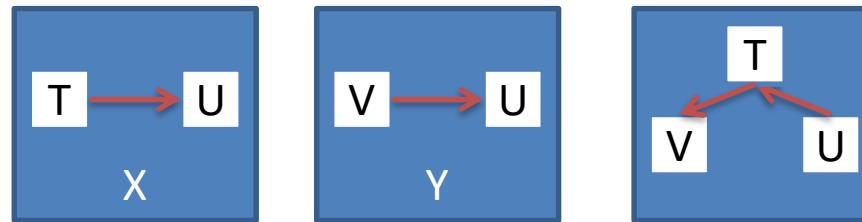
# Разпределени мъртви хватки: концепция 2-2



- ❖ Откриване на цикли в локалните wait-for графи
  - Server Y: U → V (added when U requests b.withdraw(30))
  - Server Z: V → W (added when V requests c.withdraw(20))
  - Server X: W → U (added when W requests a.withdraw(20))
- ❖ Централизирано откриване на мъртви хватки в глобалния транзакционен wait-for граф
  - Недостатъци: Слаба наличност, липса на отказоустойчивост, липса на скалируемост, разходи за трансфер на локалните wait-for графи

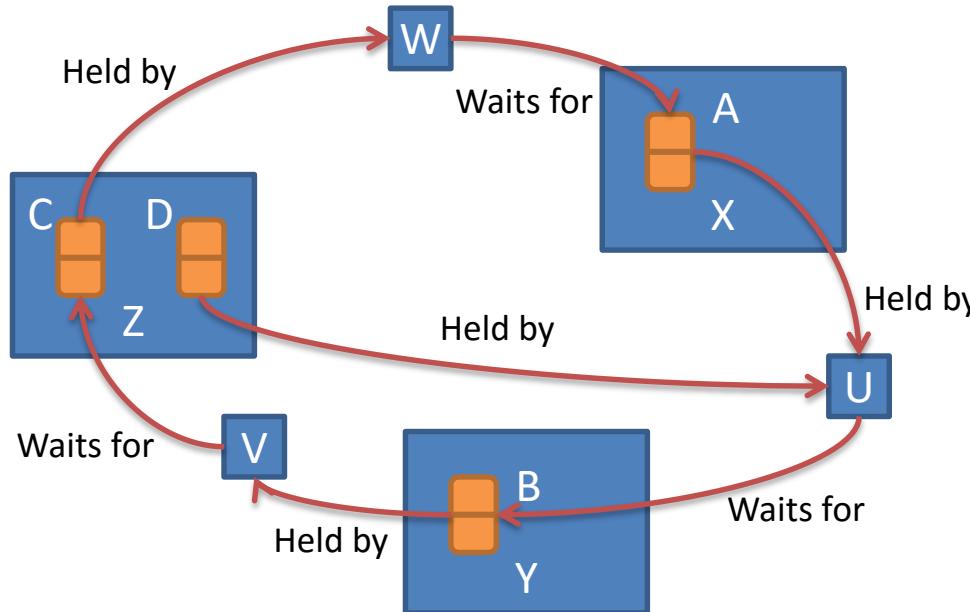
# Мъртви хватки “фентоми”

- ❖ Мъртва хватка фентом: Идентифицирана, но несъществуваща мъртва хватка
- ❖ Пример
  - Наличие на глобален детектор за мъртви хватки
  - Транзакция U освобождава обект в сървър X и заявява обект, използван от транзакция V в сървър Y
  - Глобаленият детектор получава първо локалния граф на Y
  - Наличие на цикъл  $T \rightarrow U \rightarrow V \rightarrow T$
- ❖ Възможност за откриване на мъртва хватка фентом
  - Изчакваща транзакция в мъртва хватка приключва неуспешно по време на процедурата за откриване на мъртва хватка



# Разпределен подход за откриване на мъртви хватки

- ❖ Откриването на цикли използва проучвания посредством изпращане на съобщения (probes) между сървърите
  - Сървър X добавя дъга W → U към локалния си граф
  - Транзакция U чака за обект В в сървър Y
  - Необходимост от изпращане на съобщение за проучване към Y
  - Липса на необходимост от изпращане на съобщение на по-ранна фаза, когато Z добавя дъга V → W в локалния си граф



# Алгоритъм за откриване на мъртви хватки

## ❖ Иницииране

- Т изчаква U, която изчаква обект на друг сървър
- Изпращане на съобщение  $\langle T \rightarrow U \rangle$  към сървъра, в който U е блокирана
- Ако U споделя заключване, то съобщението се изпраща до всички притежатели на заключването

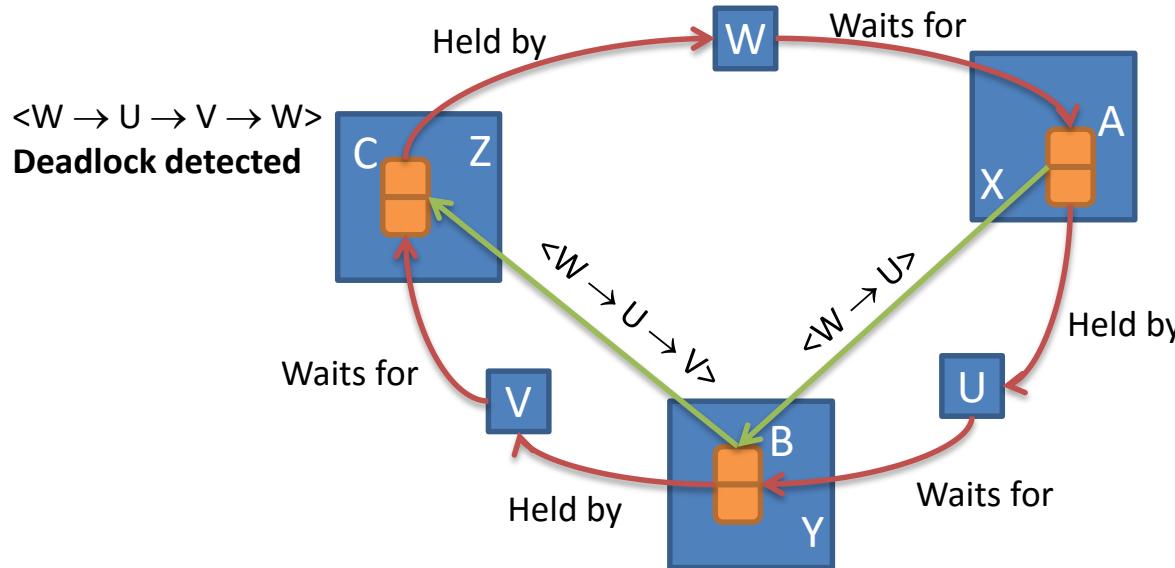
## ❖ Откриване

- Сървър получава съобщение  $\langle T \rightarrow U \rangle$  и проверява дали U е изчакваща
- Ако това е изпълнено, то в съобщението се актуализира  $\langle T \rightarrow U \rightarrow V \rangle$
- Ако V е изчакваща, то съобщението се препраща
- Преди изпращането на съобщението се проверява за наличие на цикли

## ❖ Разрешаване

- Прекъсване на транзакция в цикъл

# Пример за откриване на мъртви хватки



- ❖ Сървър X изпраща съобщение  $\langle W \rightarrow U \rangle$  към сървър Y
- ❖ Сървър Y получава съобщение  $\langle W \rightarrow U \rangle$ , установява, че В се използва от V и добавя V в съобщението  $\langle W \rightarrow U \rightarrow V \rangle$
- ❖ Сървър Y установява, че V изчаква за С и препраща съобщението до Z
- ❖ Сървър Z получава съобщение  $\langle W \rightarrow U \rightarrow V \rangle$ , установява, че С се използва от W и добавя W в съобщението  $\langle W \rightarrow U \rightarrow V \rightarrow W \rangle$
- ❖ В повечето случаи **сървърите изпращат съобщения до координаторите на транзакциите**, които след това ги препращат към съответните сървъри

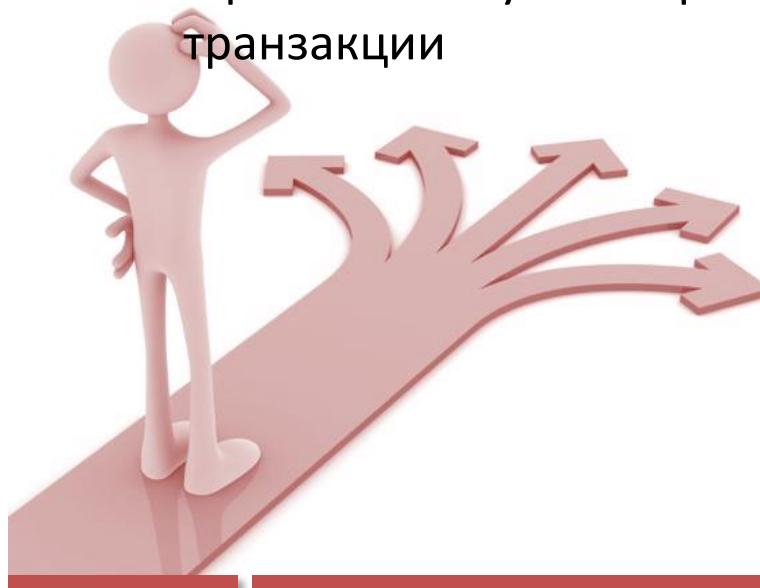
# Алгоритъм за откриване на мъртви хватки: анализ

## ❖ Ефективност на алгоритъма

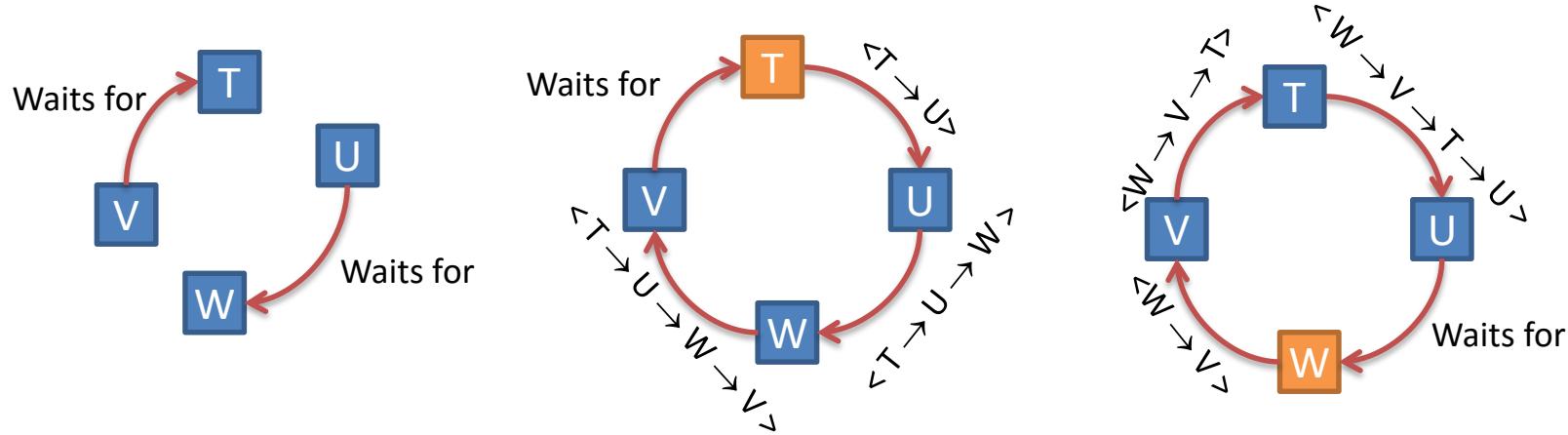
- Открива мъртви хватки при условие, че няма неуспешно приключили транзакции и повреди (загуба на съобщения и сривове в сървърите)

## ❖ Недостатък

- Изпращане на голям брой съобщения
  - ✓ При N транзакции са необходими  $2(N-1)$  съобщения
- В реалните случаи мъртвите хватки се определят от цикли с две транзакции



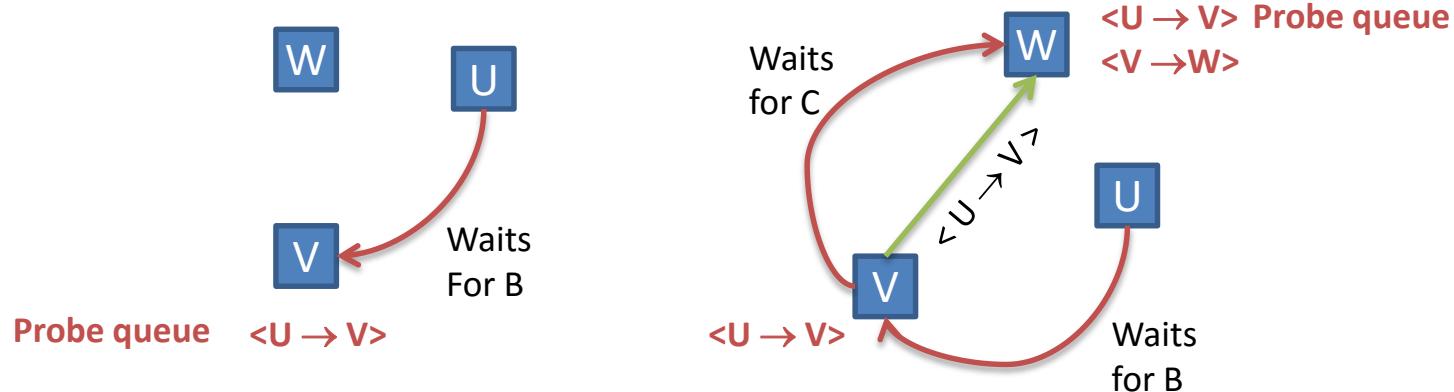
# Приоритети на транзакции 1-2



## ❖ Необходимост от приоритети

- Прекъсване на една единствена транзакция при установяване на цикъл
  - ✓ Приоритетите могат да се определят от времеви индикатори (timestamps)
  - ✓  $T > U > V > W, <T \rightarrow U \rightarrow W \rightarrow V \rightarrow T>$  или  $<W \rightarrow V \rightarrow T \rightarrow U \rightarrow W>$
- Редуциране на броя на ситуацията, иницииращи откриване на мъртва хватка
  - ✓ Правило: установяване на мъртва хватка само, ако транзакция с по-висок приоритет изчаква транзакция с по-нисък приоритет
  - ✓  $T > U$ , изпращане на  $<T \rightarrow U>$
  - ✓  $W < V, <W \rightarrow V>$  не се изпраща

# Приоритети на транзакции 2-2



## ❖ Приложение на приоритетите

- Намаляване на броя на препращаните съобщения (probes)
- Изпращане на съобщения от транзакции с по-висок приоритет към транзакции с по-нисък приоритет

## ❖ Невъзможност за откриване на мъртва хватка

- U изчаква V и V изчаква W, когато W започва изчакване за U
- W не изпраща  $<W \rightarrow U>$ ,  $W < U$

## ❖ Решение

- Съхраняване на копия на съобщенията (probes) в опашки при координаторите
- U изчаква V и координаторът на V съхранява  $<U \rightarrow V>$
- V изчаква W и координаторът на W съхранява  $<V \rightarrow W>$  и V препраща  $<U \rightarrow V>$  към W
- W започва изчакване за обект A и препраща  $<U \rightarrow V \rightarrow W>$  към сървъра на A, който установява зависимост  $W \rightarrow U$  и открива мъртва хватка  $<U \rightarrow V \rightarrow W \rightarrow U>$

# Възстановяване на транзакции: същност

## ❖ Атомарност на транзакциите

- Стабилност: съхраняване на обектите в перманентно хранилище
- Атомарност при повреди: запазване на атомарността при повреди

## ❖ Възстановяване на транзакция

- Възстановяване на сървъра с обектите, които са обновени от последните успешно приключили транзакции

## ❖ Задачи на мениджъра за възстановяване

- **Съхраняване на обекти** от успешните транзакции в перманентно хранилище (recovery file)
- **Възстановяване на обекти** след повреда на сървър
- **Преорганизиране** на хранилището с цел повишаване на производителността
- **Поддръжка** на капацитета на хранилището

## ❖ Установяване на устойчивост на хранилището

- Поддържане на копия на перманентното хранилище на огледални дискове или с различно местоположение

# Транзакционни списъци

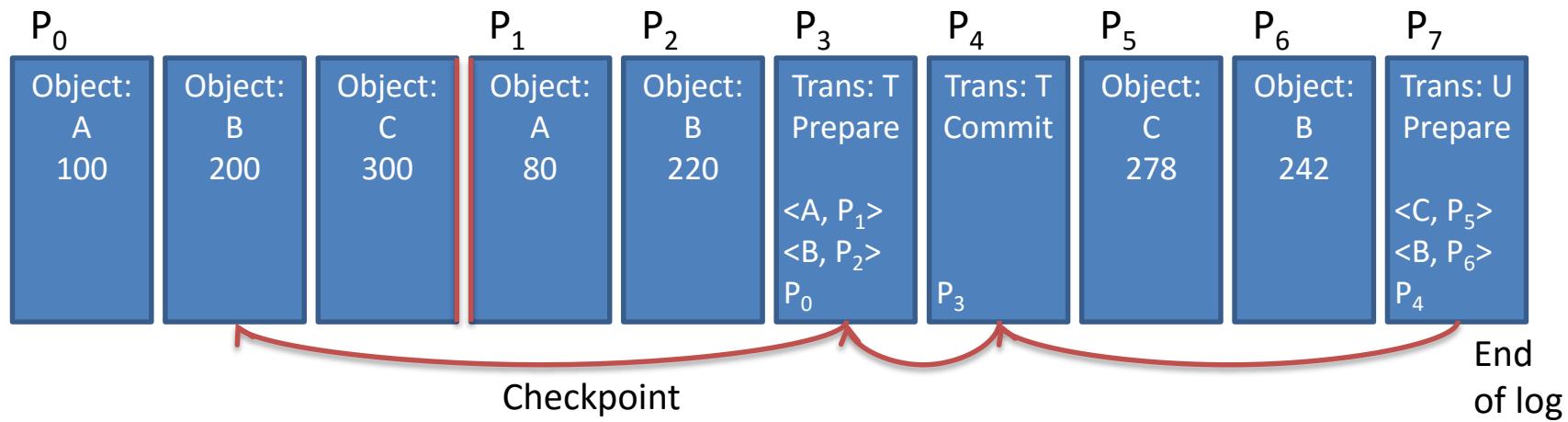
- ❖ Списък на сървър
  - Съхранява текущите активни транзакции
- ❖ Списък на транзакция
  - Съхранява референциите и стойностите на обектите от транзакцията
- ❖ Изпълнение на двуфазовия протокол
  - **Фаза 1:** Мениджърът за възстановяване съхранява списъка на транзакцията и обектите от нея във файл за възстановяване
  - **Фаза 2:** Файловете за възстановяване на сървърите трябва да осигурят достатъчно информация за успешното приключване на транзакцията дори и при повреди
- ❖ Структура на файловете за възстановяване
  - Обект
  - Статус на транзакция
    - ✓ Идентификатор, статус (prepared, committed, aborted) и др. стойности за 2PC
  - Транзакционен списък
    - ✓ Идентификатор и последователност от двойки  $\langle \text{objectID}, P_i \rangle$ , където  $P_i$  е позиция във файла

# Логове

## ❖ Структура на логовете

- Стойности на обекти, статуси и транзакционни списъци
- Редът на записите в лога се определя от последователността на изпълнение на транзакциите (prepared, committed, aborted)
  - ✓ Prepared: добавяне на обектите от транзакционния списък и текущия статус на транзакцията във файла за възстановяване
  - ✓ Committed, aborted: добавяне на текущия статус на транзакцията във файла за възстановяване
- Буфериране и едновременно записване на последователни промени във файла за възстановяване

## ❖ Лог механизъм за банкови транзакции T и U



# Възстановяване на обекти

- ❖ Възстановяване на сървър от срив
  - Мениджърът за възстановяване използва файл за възстановяване
- ❖ Подходи
  - Обхождане на файла за възстановяване в права посока
    - ✓ Последователна замяна на стойностите на обектите за успешните транзакции
  - Обхождане на файла за възстановяване в обратна посока
    - ✓ Обектите се възстановяват само веднъж
- ❖ Пример за възстановяване\*
  - Установяване, че U не е приключила и отмяна на промените от нея
  - Установяване, че T е приключила ( $P_4$ )
  - Переход към предходен запис във файла ( $P_3$ ) и откриване на транзакционни списъци за T
  - Възстановяване на обекти A и B от  $P_1$  и  $P_2$
  - Переход към контролна точка  $P_0$  и възстановяване на C

\*Вж. схемата на предходния слайд

- ❖ Особености
  - Преорганизиране на файла за възстановяване от мениджъра
    - ✓ Задаване на статус aborted за транзакции със статус prepared
  - Възстановяването трябва да има един и същ ефект при повторение
    - ✓ Постига се лесно при възстановяване на обектите от независима памет
    - ✓ Среща трудности при възстановяване на обектите от перманентно хранилище с кеш памет

# Преорганизиране на файла за възстановяване

## ❖ Цел

- Бързодействие при възстановяване и освобождаване на място
- Във файла за възстановяване са необходими само обектите с потвърдени промени

## ❖ Установяване на контролна точка (checkpointing)

- Процес, при който стойностите на текущите успешно променени обекти се записват в нов файл заедно със статусите и списъците на транзакциите, за които не е взето решение

## ❖ Установяване на контролна точка във времето

- Незабавно след възстановяване и преди стартиране на нови транзакции
- Периодично установяване

## ❖ Създаване на файл за бъдещо възстановяване

- Добавяне на маркер в текущия файла за възстановяване, запис на сървърните обекти и копиране в нов файл на:
  1. Записи преди маркировката, отнасящи се до транзакции, за които липсва решение
  2. Записи след маркировката

# Възстановяване със склад за версии 1-2

## ❖ Същност на подхода

- Използване на карта за асоцииране на идентификаторите на обектите с позициите на техните текущи версии в склада
- Версии записани от транзакциите се наричат “сенки” на предишно успешно записаните обекти

## ❖ Механизъм за изпълнение

- Prepared: добавяне на обектите, променени от транзакцията, в склада
- Commit: създаване на нова карта посредством копиране на старата и запис на позициите на версии “сенки”; замяна на старата карта с нова

## ❖ Възстановяване от срив

|                  | Map at start       |                     | Map when T commits   |                    |                    |                      |     |
|------------------|--------------------|---------------------|----------------------|--------------------|--------------------|----------------------|-----|
|                  | A → P <sub>0</sub> | B → P <sub>0'</sub> | C → P <sub>0''</sub> | A → P <sub>1</sub> | B → P <sub>2</sub> | C → P <sub>0''</sub> |     |
| P <sub>0</sub>   |                    |                     |                      | A → P <sub>1</sub> |                    |                      |     |
| P <sub>0'</sub>  |                    | B → P <sub>0'</sub> |                      |                    | B → P <sub>2</sub> |                      |     |
| P <sub>0''</sub> |                    |                     | C → P <sub>0''</sub> |                    |                    | C → P <sub>0''</sub> |     |
|                  |                    |                     |                      |                    |                    |                      |     |
| P <sub>1</sub>   |                    |                     |                      | A → P <sub>1</sub> |                    |                      |     |
| P <sub>2</sub>   |                    |                     |                      |                    | B → P <sub>2</sub> |                      |     |
| P <sub>3</sub>   |                    |                     |                      |                    |                    | C → P <sub>0''</sub> |     |
| P <sub>4</sub>   |                    |                     |                      |                    |                    |                      |     |
| Version store    | 100                | 200                 | 300                  | 80                 | 220                | 278                  | 242 |
|                  |                    |                     |                      |                    |                    |                      |     |
|                  | Checkpoint         |                     |                      |                    |                    |                      |     |

# Възстановяване със склад за версии 2-2

## ❖ Предимство

- По-бързо възстановяване в сравнение с използването на лог

## ❖ Недостатък

- По-бавно изпълнение при нормално функциониране на системата

## ❖ Използване на транзакционен статусен файл

- Съхранява транзакционни статуси и списъци
  - ✓ Списъците представлят части от картата, които са променени от транзакциите при успешното им приключване

## ❖ Статус на сървъра във времето между промяна на статуса на транзакцията в транзакционния статусен файл и времето, когато картата се обновява

- Мениджърът за възстановяване проверява в картата за промени, свързани с последната успешна транзакция в транзакционния статусен файл

| Map                  | T                  | T         | U                  |
|----------------------|--------------------|-----------|--------------------|
| A → P <sub>1</sub>   | prepared           | committed | prepared           |
| B → P <sub>2</sub>   | A → P <sub>1</sub> |           | B → P <sub>3</sub> |
| C → P <sub>0''</sub> | B → P <sub>2</sub> |           | C → P <sub>4</sub> |

# Възстановяване при разпределени транзакции 1-3

## ❖ Нови стойности за статус на транзакция

- Done: индикация от координатора за приключване на двуфазовия протокол
- Uncertain: индикация от участник гласувал с Yes за липса на информация относно резултата от гласуването

## ❖ Нови типове записи

- Coordinator: транзакционен идентификатор, списък от участници
- Participant: транзакционен идентификатор, координатор

## ❖ Фаза 1 на двуфазовия протокол

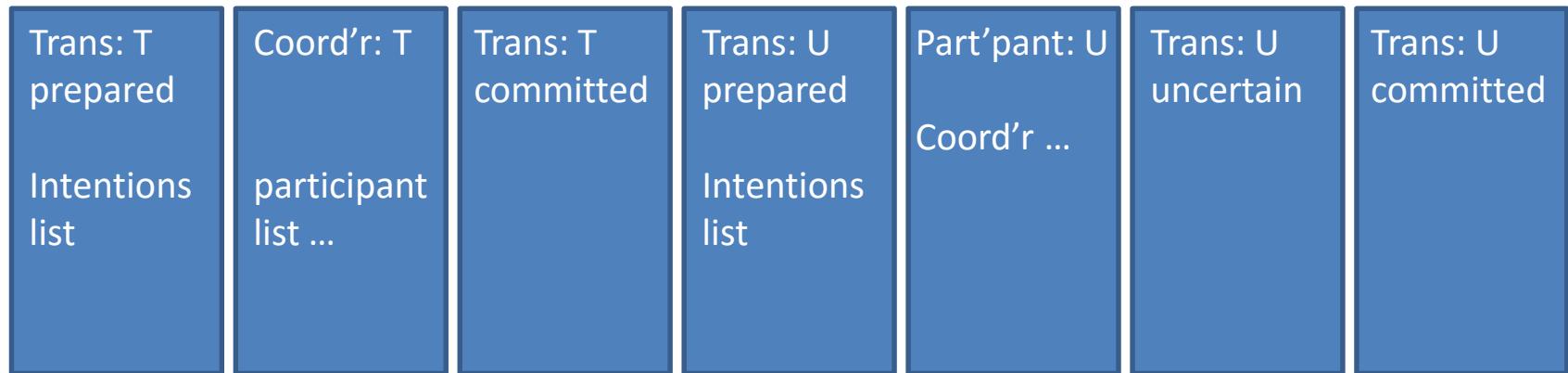
- Добавяне на статус **prepared** от координатора
- Добавяне на запис **coordinator** от мениджъра
- Добавяне на статус **prepared** от участник преди гласуване с Yes
- Добавяне на запис **participant** от мениджъра при гласуване с Yes
- Добавяне на статус **uncertain** от участник при гласуване с Yes
- Добавяне на статус **abort** за транзакция при гласуване на участник с No

# Възстановяване при разпределени транзакции 2-3

## ❖ Фаза 2 на двуфазовия протокол

- Мениджърът на координатора добавя статус committed или aborted
- Мениджърите на участниците добавя статус committed или aborted
- При потвърждение от всички участници мениджърът на координатора добавя статус done

## ❖ Лог за транзакции T (координатор) и U (участник)



# Възстановяване при двуфазен протокол

| Роля        | Статус    | Действие на мениджъра                                                                                                                               |
|-------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Coordinator | Prepared  | Липса на решение преди повреда. Изпращане на <i>abortTransaction</i> до всички сървъри в списъка с участниците и добавяне на статус <i>aborted</i>  |
| Coordinator | Committed | Наличие на решение при повреда. Изпращане на <i>doCommit</i> до всички участници в списъка и възстановяване от стъпка 4 на протокола                |
| Participant | Committed | Участникът изпраща <i>haveCommitted</i> до координатора, който премахва информацията за транзакцията при следващото установяване на контролна точка |
| Participant | Uncertain | Участникът се срива преди да получи информация за решението от гласуването. Изпращане на <i>getDecision</i> към координатора                        |
| Participant | Prepared  | Участникът не е гласувал и може да откаже транзакцията                                                                                              |
| Coordinator | Done      | Не се изисква действие                                                                                                                              |





# Mobile and Ubiquitous Computing

## Lecture 11

# Outline

- ❖ Introduction
- ❖ Association

# Introduction

- ❖ Mobile and hand-held computing
- ❖ Ubiquitous computing
- ❖ Wearable computing
- ❖ Volatile systems
  - Smart spaces
  - Device model
  - Volatile connectivity
  - Spontaneous interoperation
  - Lowered trust and privacy

# Mobile and hand-held computing

## ❖ Mobile computing

- users could carry their personal computers and retain some connectivity to other machines
- Laptop, netbook, tablet with wireless connectivity (cellular networks, WiFi and Bluetooth)

## ❖ Hand-held computing

- the use of devices that fit in the hand
- PDAs, Smartphones and purpose-built handheld devices (cameras and GPS-based navigation units)

## ❖ Problems in wireless communication

- how to provide continuous connectivity for mobile devices that pass in and out of range of *base stations*
- how to enable collections of devices to wirelessly communicate with one another in places where there is no infrastructure

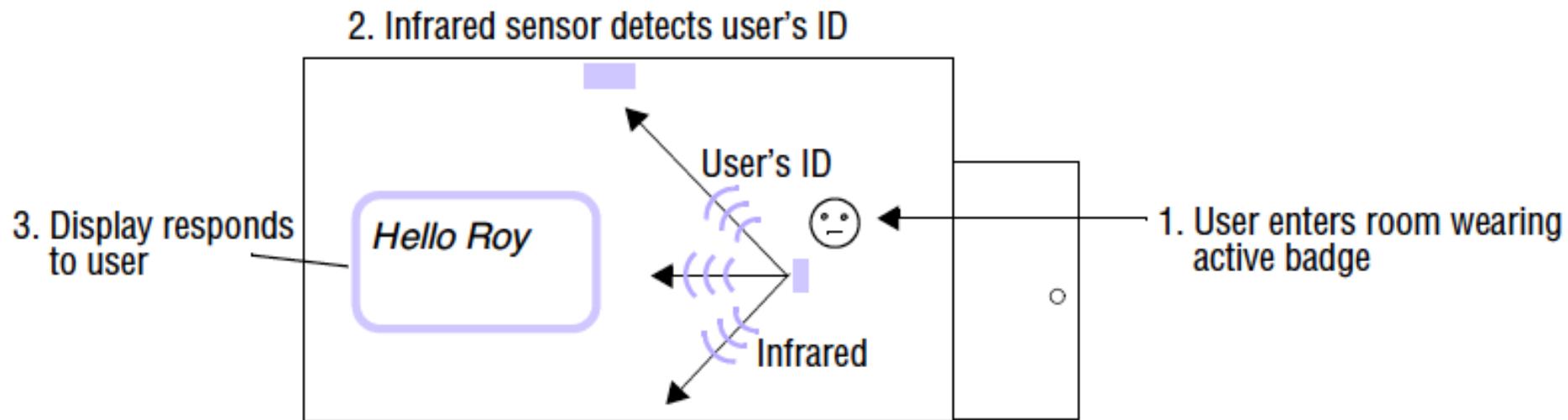
# Ubiquitous computing

- ❖ ‘Ubiquitous’ means ‘to be found everywhere’
- ❖ *ubiquitous computing - pervasive computing*
  - ‘one person, many computers’ - computers multiply in form and function to suit different tasks.
  - computers would ‘disappear’ - they would ‘weave themselves into the fabric of everyday life until they are indistinguishable from it’

# Wearable computing

## Wearable computing devices

- operate without the user having to manipulate them
- have specialized functionality



## Context-aware computing

- computer systems automatically adapt their behaviour according to physical circumstances

# Volatile systems

## ❖ Volatile systems

- ‘volatile’ - certain changes are common rather than exceptional
- The set of users, hardware and software in mobile and ubiquitous systems is highly dynamic and changes unpredictably
- spontaneous systems or spontaneous networking
- Forms of volatility
  - ✓ failures of devices and communication links;
  - ✓ changes in the characteristics of communication such as bandwidth;
  - ✓ the creation and destruction of *associations* – logical communication relationships– between software components resident on the devices

## ❖ Component - any software unit, such as objects or processes, regardless of whether it interoperates as a client or server or peer

# Smart spaces

- ❖ A *smart space* is any physical place with embedded services
  - services provided only or principally within that physical space.
- ❖ Types of movements in smart spaces
  - physical mobility
  - logical mobility
  - users may add relatively static devices (media players) as longer-term additions to the space, and correspondingly withdraw older devices from it.
  - devices may fail and thus ‘disappear’ from a space

# Device model

❖ Our device model must take into account the following characteristics:

- Limited energy
- Resource constraints
- Sensors and actuators
  - ✓ Sensors - devices that measure physical parameters and supply their values to software (sensors that measure position, orientation, load, and light and sound levels)
  - ✓ Actuators - software-controllable devices that affect the physical world (programmable air conditioning controllers)

❖ Devices

- Motes
  - ✓ small devices intended for autonomous operation in applications such as environmental sensing
- Smart phones

## ❖ Factors

- Connection technologies vary in their nominal bandwidth and latency, in their energy costs and in whether there are financial costs to communication
- volatility of connectivity – the variability at runtime of the state of connection or disconnection between devices, and the quality of service between them – also has impact on system properties

## ❖ Connectivity issues:

- Disconnection
  - ✓ Ad-hoc routing - a collection of devices communicate with one another without reliance on any other device: they collaborate to route all packets between themselves
- Variable bandwidth and latency

# Spontaneous interoperation 1

❖ Components routinely change the set of components they communicate with.

- ***association*** - the logical relationship formed when at least one of a given pair of components communicates with the other over some well-defined period of time,
- ***interoperation*** - their interactions during their association

❖ Types of associations

- Preconfigured associations
- Spontaneous associations

# Spontaneous interoperation 2

## Preconfigured

Service-driven:

*email client and server*

## Spontaneous

Human-driven:

*web browser and web servers*

Data-driven:

*P2P file-sharing applications*

Physically driven:

*mobile and ubiquitous systems*

Lowered trust and privacy

# Outline

- ❖ Introduction
- ❖ Association

- ❖ Network bootstrapping
- ❖ Association
- ❖ The association problem and the boundary principle
  - Association problem: how to associate appropriately within the smart space.
  - Solution must address two aspects – scale and scope
- ❖ The boundary principle
  - smart spaces need to have system boundaries that correspond accurately to meaningful spaces as they are normally defined territorially and administratively
  - Those ‘system boundaries’ are system-defined criteria that scope but do not necessarily constrain association.

# Discovery services

- ❖ Discovery service - directory service in which services in a smart space are registered and looked up by their attributes, but one whose implementation takes account of volatile system properties
  - the directory data required by a particular client is determined at runtime.
  - there may be no infrastructure in the smart space to host a directory server.
  - services registered in the directory may spontaneously disappear.
  - the protocols used for accessing the directory need to be sensitive to the energy and bandwidth they consume.
- ❖ Device discovery - clients discover the names and addresses of co-present devices

# Interface to discovery service

## Methods for service registration and deregistration

*lease := register(address, attributes)*

## Explanation

Register the service at the given address with the given attributes; a lease is returned

*refresh(lease)*

Refresh the lease returned at registration

*deregister(lease)*

Remove the service record registered under the given lease

## Method to look up a service

*serviceSet := query(attributeSpecification)*

Return a set of registered services whose attributes match the given specification

# Developments in discovery services

- ❖ Jini discovery service
- ❖ the service location protocol
- ❖ the Intentional Naming System
- ❖ the simple service discovery protocol (Universal Plug and Play initiative - [www.upnp.org](http://www.upnp.org))
- ❖ Secure Service Discovery Service

# Design of a discovery service

❖ The issues to be dealt with in the design of a discovery service are

- Low-effort, appropriate association
  - ✓ the set of services returned by the *query* operation would be appropriate
  - ✓ Service selection could be made programatically or with small human input
- Service description and query language
  - ✓ The query and description languages have to agree (or be translatable), and their expressiveness has to keep pace with the development of new devices and services
- Smart-space-specific discovery
  - ✓ We require a mechanism for devices to access an instance (or scope) of the discovery service that is appropriate to their current physical circumstances – a mechanism that doesn't rely on the device knowing the particular name or address for that service *a priori*.
- Directory implementation
- Service volatility

# Implementing a discovery service

## ❖ Design choices

- whether the discovery service should be implemented by a *directory server*, or be serverless?

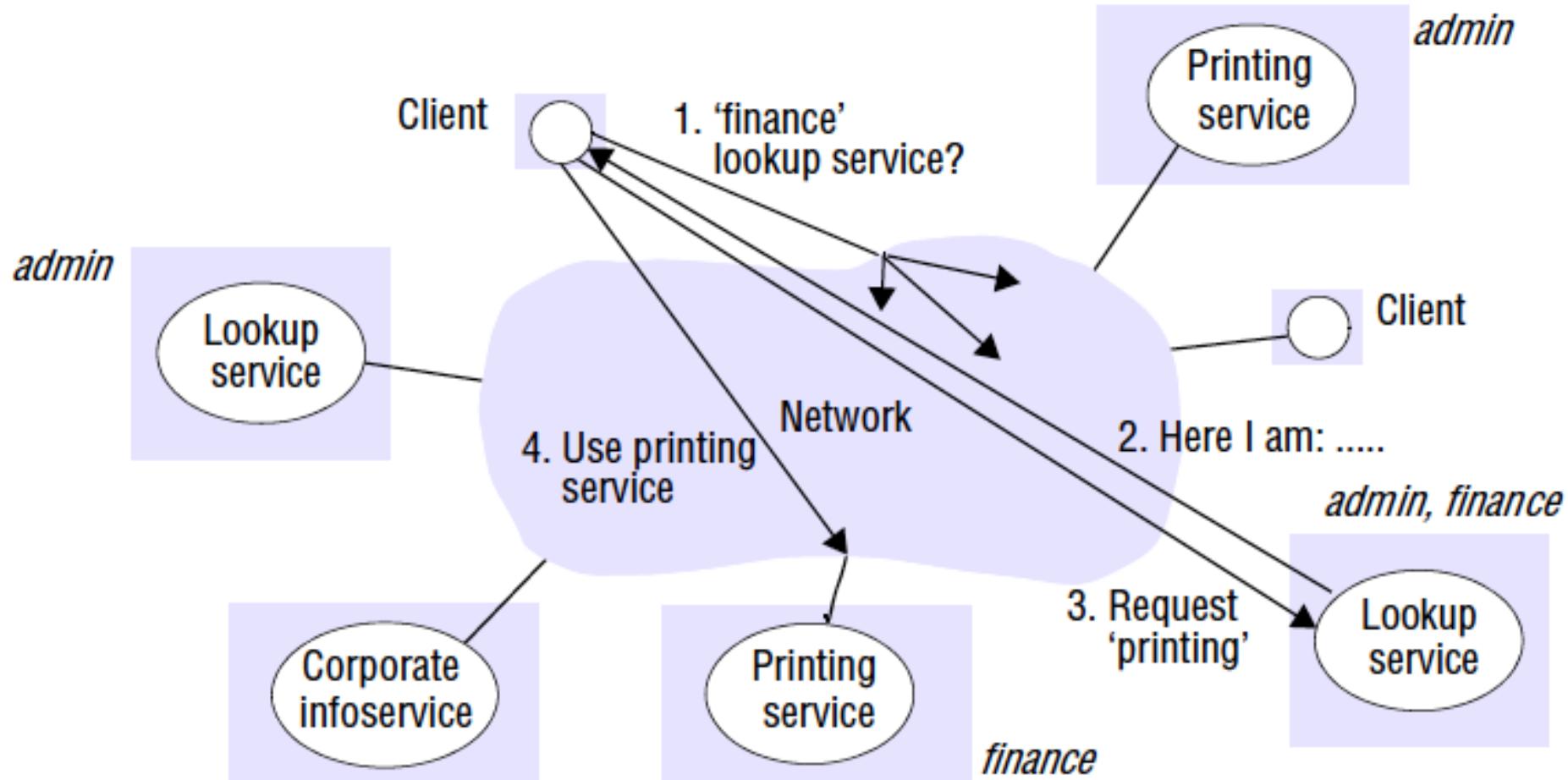
## ❖ Directory server

- holds a set of descriptions of services that have registered with it, and responds to clients issuing queries for services
- location
  - ✓ can often be run from mains power on a robust machine;
  - ✓ Election a server from whatever devices happened to be present

## ❖ *Serverless discovery*

- the participating devices collaborate to implement a distributed discovery service, in lieu of a directory server.
- two main implementation variants
  - ✓ *push* model, services multicast ('advertise') their descriptions regularly.
  - ✓ *pull* model, clients multicast their queries

# Jini's discovery system



# Discussion of network discovery services

- ❖ Network discovery services raise two difficulties when looked at from the perspective of the boundary principle:
  - the use of a subnet
    - ✓ Network discovery may mistakenly include services that are not in the smart space
    - ✓ network discovery may mistakenly discount services that are ‘in’ the smart space in the sense of being eligible for discovery there, but are hosted beyond its subnet
  - inadequacies in the way services are described
    - ✓ discovery may be *brittle*: even slight variations in the service-description vocabulary used by disparate organizations could cause it to fail
    - ✓ there may be *lost opportunities* for service access

- ❖ Human input to scope discovery
  - a human provides input to the device to set the scope of discovery
- ❖ Sensing and physically constrained channels to scope discovery
- ❖ Direct association
  - Address-sensing
  - Physical stimulus
  - Temporal or physical correlation

# Summary

- ❖ the main challenges raised by mobile and ubiquitous computer systems, and rather fewer solutions
- ❖ Volatile systems - the set of users, hardware and software components in a given smart space is subject to unpredictable change

