# Lecture 1

* Distributed systems: is a collection of independent computers that appear to the users of the system as a single computer
* cloud: collection of interconnected & virtualised computers that are dynamically provisioned based on service-level agreements
* cluster: collection of interconnected stand-alone computers cooperatively working together as a single resource
* characteristics of distributed systems:
  1. parallel activities
  2. communication via message passing [no shared memory]
  3. Resource sharing
  4. No global state [no single processor can have knowledge of the current global state of the system]
  5. No global clock
* Goals of Distributed systems:
  1. connecting users & resources
  2. Transparency
  3. openness [difficulty to extend or improve an existing system]
  4. Scalability
  5. Enhanced Availability
* challenges:
  1. Heterogeneity [hardware/OS/ data representation]
  2. Distribution transparency [should be hidden from the user]
  3. concurrency [shared access]
  4. Fault tolerance
  5. Security
  6. scalability
  7. Reuse & openness

• Distributed systems enable globalization

intranet → a portion of internet & its services [ collection of computer networks of many different types and hosts various types of services]

| * Network | vs | distributed systems |
|---|---|---|
| 1. visible [IP address] | | transparent |
| 2. packets | | applications |

← de facto

# Lecture 2

## Sockets:

* Syncronous --> telephone
  Asyncronous --> whatsapp

* The characteristics of interprocess communication
  --> Reliability : guarantee to be delivered despite a number of packets being dropped
  --> ordering

* TCP/ IP Stack
  ① physical/ link layer : A stream of data from one computer to another
  ② Internet / Network layer : IP? address of remote computer
  ③ Transport layer: [delivering to a specific process]
    --> TCP : transmission control protocol
      * connection- oriented
      * Reliable flow of data (sequencing/ Retransmission)
      * speaking on phone (skype calls)
    --> UDP : user datagram protocol
      * connectionless communication
      * No guarantee about arrival or order of arrival
      * Live stream (Messenger)
  ④ Application layer

* Sockets are at transport layer

server not up ← * Connection refused exception
* unknown host exception
* IO Exception
* Security Exception

TCP

Server Soket.
① open server socket
② wait for client request
③ create I/O stream for communicating to the client
④ perform communication with client
⑤ close socket

• IP
(1) 4 byte
(2) Devices

Port
16 bit
applications

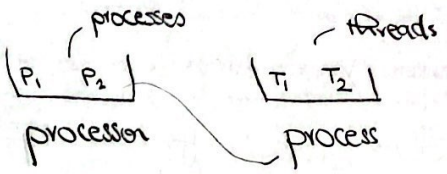
* TCP & UDP use ports to map incoming data to a particular process.

Scanned with CamScanner

# Lecture3

## Threads:

* preemptive (processor responsibility)
  ↳ time slice
* cooperative (process responsibility)
  ↳ لما واحد يخلص اللي بعده يشتغل




* Multithread: different threads within the same process [run concurrently]
  Multitasking: different processes on two different processors
* An execution environment is a collection of local kernel-managed resources to which its threads have access.
* An address space is a unit of management of a process's virtual memory.
*

| | process | Thread |
|---|---|---|
| communication | hard, message passing | easy, share resources |
| context switch | hard, storage in stack | easy, no need to store |
| Security | Better | worse |

* Threads are used to perform parallelism & concurrent execution [asynchronous]
* Threading mechanism:
  ① extends Thread [java only supports single inheritence]
  ② implements Runnable
* Life cycle of Thread:
  New → t = new Thread();
  Start → t.start();
  Ready —context switch→ running
  blocked → resource comes

* Thread operations:
  ① sleep
  ② join → waiting for response from all thread
  ③ Synchronized
    ↳ one thread tries to read data that other is trying to update at the same time

* Thread priority:
  - MIN_PRIORITY
  - NORM_PRIORITY [FCFS]
  - Max_PRIORITY

* Multitasking of two or more processes is known as process-based multitasking (→ OS-controlled)
  Multitasking of two or more threads is known as thread-based multitasking (↳ programmer-controlled)

① Extends Thread & override run() method
② MyThread T1 = new MyThread();
③ T.start();

Scanned with CamScanner

# Lecture 4 Architecture Elements

1. What entities are communicating in DS?
   - → objects:
     - They are accessed via interfaces. Example: RMI / CORBA
   - → components:
     - Example: @Dependency Injection
   - → Web services

2. What communication paradigm used?
   - → Inter process communication:
     - Low level support for communication. Example: Sockets
   - → Remote invocation:
     - calling of a remote operation, procedure or method. Example: RMI / CORBA
   - → Indirect communication:
     - Example: Message queues

online ① : API — updated ②
offline ① : Library — not updated ②, Formated ④

{ variables ١- ٢- parsing ٣- Format ٤-

3. What roles & responsibilities do entities have?
   - → client_server:
     - * Single point of failure
     - * Have the authority to make changes
   - → peer-to-peer:
     - * Example: BitTorrent & Skype
     - * No Single point of failure ⇒ Availability
     - * Don't have the authority * Scalability

Bank

Application
Middleware
OS
hardware & network
Network

platform

4. How are entities mapped onto physical infrastructure?

Architectural patterns:

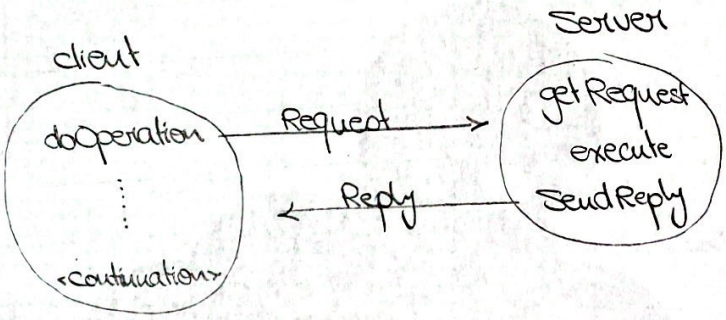
* Layer: a group of related functional components [conceptual]
  Service: functionality provided to the next layer
* Two-tier ⇒ [physical] → client_server architecture types
  Three-tier ⇒ UI → business logic → Database
  Multi-tier
* Middleware: a layer of software whose purpose is to mask heterogeneity present in distributed systems. → more portable / productive / have fewer errors
* layer → conceptual
  tier → physical

Scanned with CamScanner

# Lecture 5

## * RMI

- RPC ⇝ remote procedure call
  RMI ⇝ remote method invocation
- Request/Reply protocol
  → Synchronous & reliable
  → UDP / TCP
  → 3 primitives:
  ① doOperation
  ② getRequest
  ③ SendReply




* DoOperation (..., byte[] arguments)
  ↳ caller is blocked until server transmits reply
  ↳ ① Remote server
  ② operation
  ③ arguments of the operation

* if UDP is used instead of TCP:
  ① omission failures.
  ② no guarantee that messages will be delivered in order

* doOperation & time out: [failure model]
  - Return failure to client
  - Resend the request repeatedly [reply is received or no response]
  - Exception from server as there is no response from server.
  - to avoid the execution of an operation more than once for the same request,
    the protocol is designed to recognize successive messages (from the same client)
    with the same request identifier and to filter out duplicates

idempotent operation: العمليات التي يمكن تنفيذها عدة مرات دون تغيير النتيجة
checkBalance(); can be performed repeatedly without any effect
withdraw();

Design issues for RPC

- Styles of programming
- call semantics
- transparency

Scanned with CamScanner

## 1] Styles of Programming:

- Explicit interface
  → Tell exactly how a client can access the server.
- keep implementation separate from interface
- Difference from local procedure interface
  → can't access shared memory
  → No call by reference
  → can't pass pointers

Service interface
↳ specification of the procedures offered by a server.

IDLs
↳ interface definition lang.

## 2] RPC call semantics:

| Retransmit request? | Duplication? | Re-execute? | call semantic |
|---|---|---|---|
| X | Not applicable | Not applicable | Maybe |
| yes | NO | Re-excute | at-least once — SUN RPC |
| yes | yes | Re-transmit | at-most once — Java RMI |

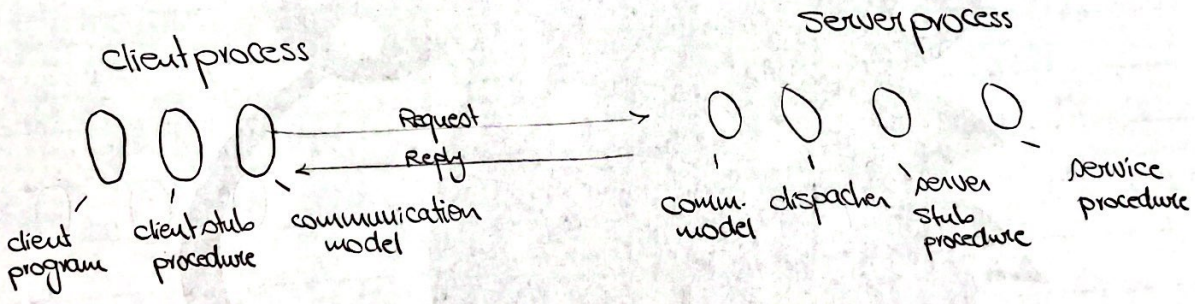
## 3] Transparency:

→ Access transparency
↳ enables local & remote resources to be accessed using identical operations.

→ Location transparency

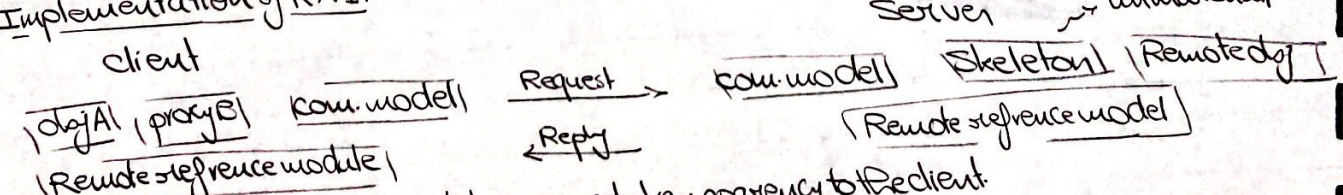
RPC → marshalling & message passing

## Implementation of RPC:




* one stub for each procedure
* communication model is used to send marshalled info to & from server
* Dispacher is used to select the corresponding server stub procedure.
* Server Stub is used to unmarshall & marshall & calls the corresponding service procedure

## Implementation of RMI:




* proxy is used to implement a stub & provide transparency to the client.
* Remote Reference module is responsible for translating between local & remote object reference

Scanned with CamScanner

# Lecture 6

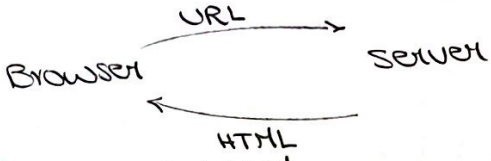
- Issues with object-oriented middleware:
  (1) Implicit dependency
  (2) Interaction with middleware [bad abstraction level]
  (3) Lack of separation of distribution concerns
  (4) No support for deployment
- Application server: middleware supporting the container pattern and the separation of concerns implied by this pattern.
  - Advantage? support for the three tier approach
  - Disadvantage? prescriptive & heavyweight
- EJB → server-side | EE → client-side
- container pattern is used to provide support for key distributed system services
- Beans:
  - Session beans:
    ① Stateless: does not maintain conversational state with clients between calls
    pro? dynamic scalability → performance↑ * → *
    ② Stateful: maintain conversational state with clients across multiple calls
    ③ Singleton: instantiated once per application and exists for the lifecycle of the application 1 → 1
    ↳ global state to all users
  - Message driven beans: enables to process messages asynchronously
- Plain old Java objects [POJO] + annotations [@] = EE
- Transactions:
  * bean managed: @ Transaction Management (BEAN)
  * container managed: @ Transaction Management (container)

Abstraction:

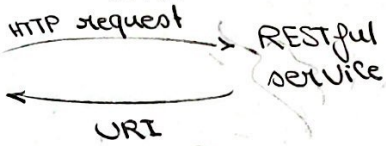

Scanned with CamScanner

# Lecture 7

- Web server:




- Web service:



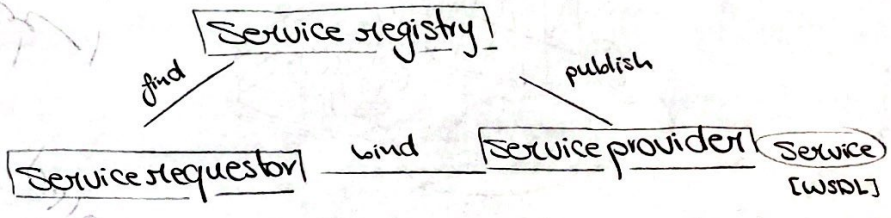

↳ is a functionality that is exposed and would be accessed through web technologies

* Response time / availability become very important
* provide modularity / extensibility / code reuse
* connection issues

* Service oriented architecture principles:

① Boundaries are explicit
② Services are autonomous
- Design-time autonomy [change in PL]
- Run-time autonomy [whatever deployment strategy i.e. replicas]
③ Share schemas and contracts, not implementation — * operation's signature
* XML file [data structure]
④ Services compatibility is based on policy

* Roles of SOA Building blocks:




- SOAP uses XML for its message format & relies on HTTP or SMTP for message negotiation
- REST uses HTTP protocol

* GET → retreive
* POST → create
* PUT → update
* Delete → delete

Scanned with CamScanner

## Scaling distributed systems:

* Scaling up: increase the application server hardware's configurations.

  pros? Simple

  cons? ① Single point of failure ② Limited capacity ③ not costly

* Scaling out: replicating a service and running multiple copies on multiple server nodes

  pros? No single point of failure ⇒ scalability

  cons? ① costly ② complex [code changes] ③ DB

* Load balancer:

  ↳ receives requests and choose a service replica to process the request

* Stateless services:

  ↳ API implementations should retain no knowledge or state associated with an individual client's session.

* Stateful uses session store to store the last state of user's activity

## MicroServices

* Monolithic application puts all its functionality into a single process _ [independent deployable]
* Microservice architecture puts each element of functionality into a separate service where it communicates with light weight mechanisms, often an HTTP resource API

* Limitations of monolithic applications

  ① change cycles
  ② Modular structure
  ③ Scaling
  ④ Start time
  ⑤ Reliability

* The scale cube:

  ① x axis - horizontal duplication [monolithic]
  ↳ scale by cloning

  ② y-axis - functional decomposition
  ↳ scale by splitting different things [microservice]

  ③ z-axis - data partitioning
  ↳ scale by splitting similar things

* characteristics of microservice architecture:

  ① componentization via services
  ② organized around business capabilities not technologies
  ③ Decentralized governance
  ④ Decentralized data management
  ⑤ Design for failure

Scanned with CamScanner

* Microservice trade-offs:

① Strong modular boundaries [Pro]
② Distribution [con]
- Low performance — remote calls — latency
solutions? ① coarse-grained calls
② Asynchrony [debugging ↓]
- Reliability ↓
③ Eventual consistency
④ Independent deployment [PRO]
⑤ Technology diversity [PRO]

* API gateway encapsulates the internal system architecture & provides tailored API to each client
↳ coarse-grained APIs are used
Pros? encapsulates the internal structure of the application
cons? at some point, it becomes a bottle neck that must be deployed, development&manag

* Design considerations:
① performance & scalability
② service invocation
③ service discovery

* Inter-process communication
① Request/response
② Notification [publish/subscribe]
③ Message queues [Request/async]

Scanned with CamScanner

# Distributed caching:

* Caching benefits
  - Heavy read traffic reduction
  - Computation costs reduction
* Cache hit [maximize] ⇝ cache موجود
* Cache miss [minimize] ⇝ [DB retrieval] ⇝ cache مش موجود فالـ
* When items are updated regularly, the cost of cache misses can negate the benefits of cache

⟹ if TTL ↓ ⇝ update faster than DB
TTL ↑ ⇝ out dated data

* Application caching_[cache-aside]
  1. Read-through ⇝ 🧍 → cache ⟷ (miss) loader ↔ DB
  2. Write-through ⇝ 🧍 → cache ⟷ (write) writer ↔ DB [write updates]
     ↳ only when the DB is updated, the application can complete the request
  3. Write-behind ⇝ like write-through, except the application does not wait for the value to be written to DB from the cache. Data inconsistency

* Web caching
  1. Web browser caches are also known as private caches [for single user] — personalized cache
  2. Organizational web proxy (ISP proxy) are shared caches that support requests from multiple users
  3. Edge caches lives at various strategic geographic locations globally

* Consistency
  - Strong consistency: all replicas must exhibit the same value to clients all the time.
  - Eventual consistency: consistency achieved with some latency [availability over consistency]
    ↳ affected by the number of replicas

Cons of read through cache
1. cost
2. Added new responsibility on cache
3. No failure handling ⇝ no access directly to DB

⟹ change in business logic

Scanned with CamScanner

# Blockchain:

* Definition: is a peer-to-peer distributed ledger that is cryptographically secured, append-only, immutable, and updatealde only via agreement among peers.

* Byzantine node: exhibits irrational behavior

* CAP theorem:
  ① consistency
  ② Availability
  ③ Partition tolerance
  ] Distributed system cannot have all three of the desired properties

* Types of faults in distributed systems:
  ① fail-stop faults (crash faults) ⇒ simpler to deal with
  ② Byzantine faults (untrustworthy) ⇒ difficult to deal with

* The generic structure of a block:
  ① Address ⇝ unique identifiers used to denote senders & recipient
  ② Transaction ⇝ transfer of value from one address to another
  ③ Block ⇝ block header & transaction header
  ④ Nonce ⇝ generated and used once to provide replay protection
  ⑤ Timestamp ⇝ creation time of the block
  ⑥ Refrence to a previous block ⇝ hash of the header of the previous block.
  - Genesis block is the first block in the blockchain

* Hash:
  ① General properties:
    - maps input ∝ of any size to an output [hash]
    - Deterministic
    - Efficiently computed
  ② cryptographic properties
    - one way
    - collision resistant
    - Avalanche effect
    - puzzle friendliness

* Binary data tree with hashes
  ↳ smaller in size + no fraud

* centralized
  ↳ client & server (single authority)

* Distributed
  ↳ data & computations are spread across multiple nodes in the network

* Decentralized
  ↳ every node makes its own decision

Scanned with CamScanner