Distributed systems course project:
A Distributed Messaging Application Based on SIP protocol
Design Phase

Group Members:
SeyedAlireza Sanaee
Erfan Sharafzadeh
Mahdi Bagvand
Hooman Behnejadfard

Distributed system concepts

Scalability

- Service
- Data
- Algorithm

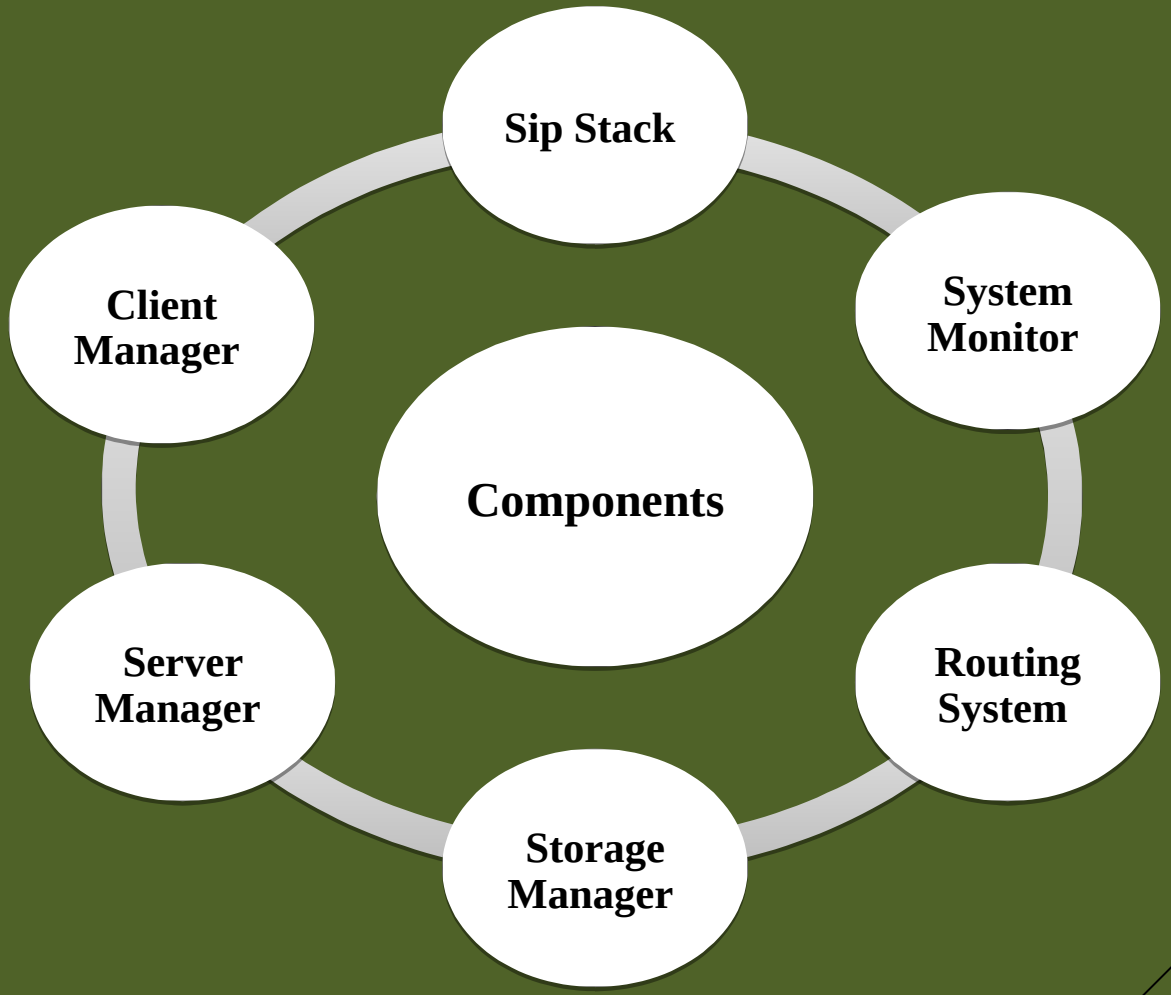
Accessibility

Openness

Transparency

- Access
- Location
- Migration
- Relocation
- Replication
- Concurrency
- Failure

System Components



Client Manager Component

- Each server is in charge of managing its clients
- So it records their credentials to know them later
- Client registration and de-registration
- Client message storage

Server Manager Component

- Exactly like Client manager, Servers should have a record of their neighboring servers
- Neighbor registration and deregistration is also done here
- The Temporary Message storage is managed by this component

Storage Manager Component

- Each server have some corresponding clients,
  - Messages which are for these clients should be stored temporarily in their authoritative server
  - Until an acknowledgment comes from the receiving client

Routing System Component

- When a message received by the server,
  - If the server doesn't know about the receiver,
    - This component decides which server should the message be sent to
- Avoiding the servers that we already sent to
- Do we need broadcasting??

System Monitor Component

- Program logs
- Debugging logs
- Graphical User interface for registration
- UI for sending and receiving messages

Sip Stack Component

- The main functionality of SIP is coming from JAIN SIP library.
- We are using the library's interfaces to provide our system with SIP stack methods
- SIP is an application layer textual protocol
- Every message in the system is in SIP format

System Scenarios

Connecting client to server

1. Client sends a REGISTER request message

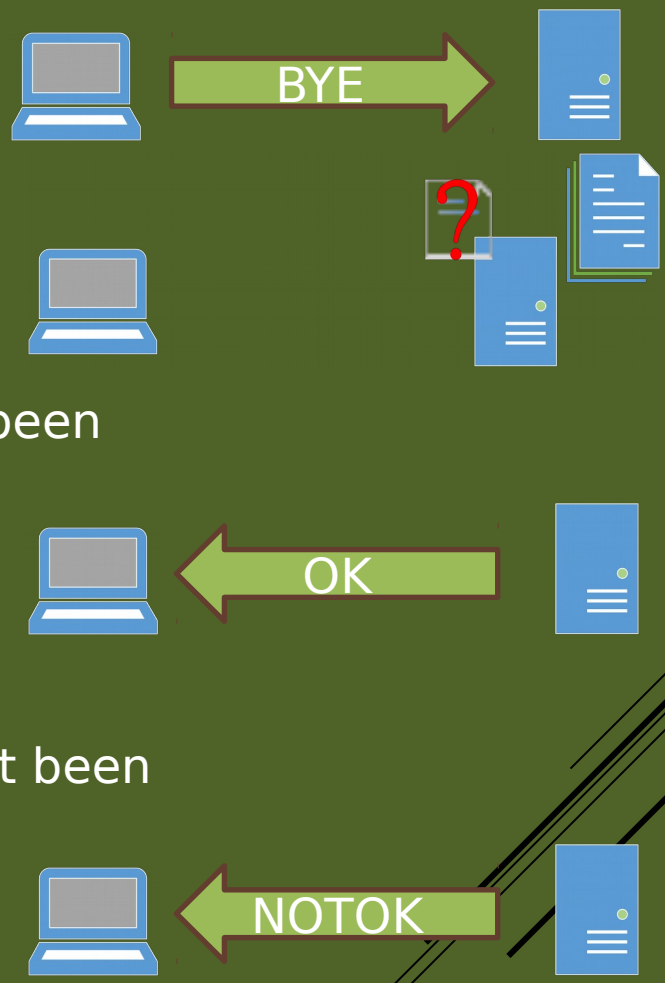
2. Server checks client credentials

3. Server returns OK response if it is the first communication with its authoritative server

4. Server returns NOTOK response if client has been already registered in the server's database

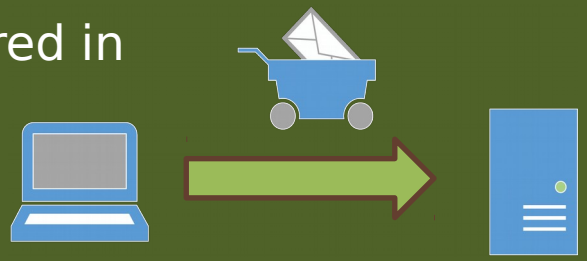
Disconnecting client to server

1. Client sends a BYE request message
2. Server checks client credentials
3. Server returns OK response if client has been registered before
4. Server returns NOTOK response if has not been registered before

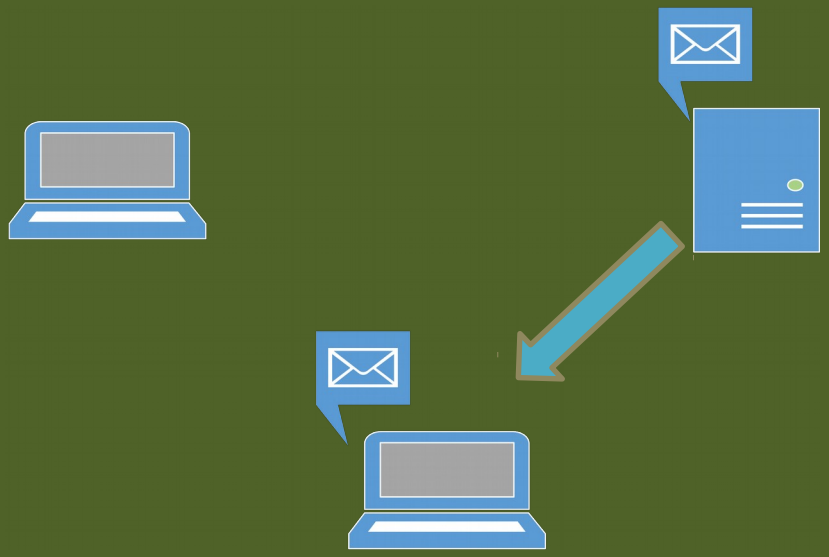


Sending message from client

1. Client sends a message to the client registered in another server other than its server



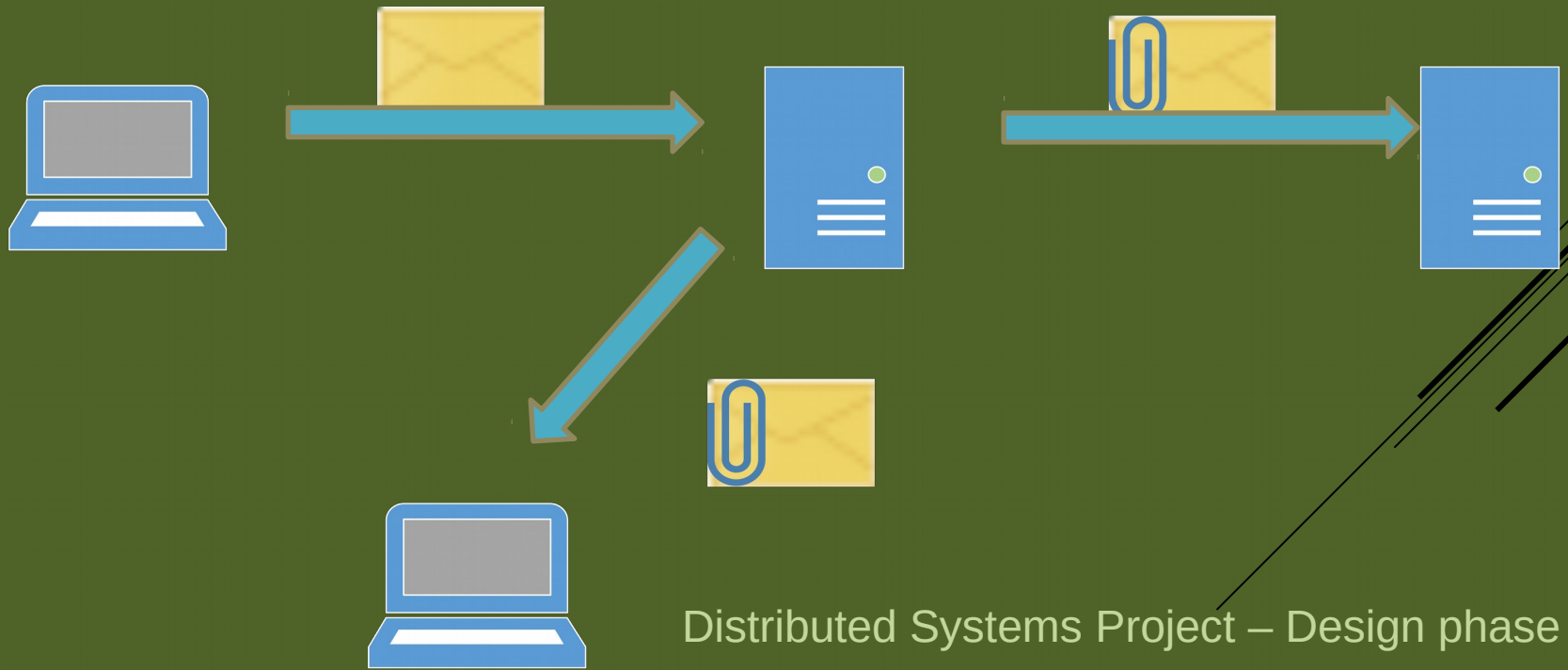
2. Client waits for two responses:
   - One for delivering message to its authoritative server
   - One for delivering message to destination



Server receiving a message

Message received from a client

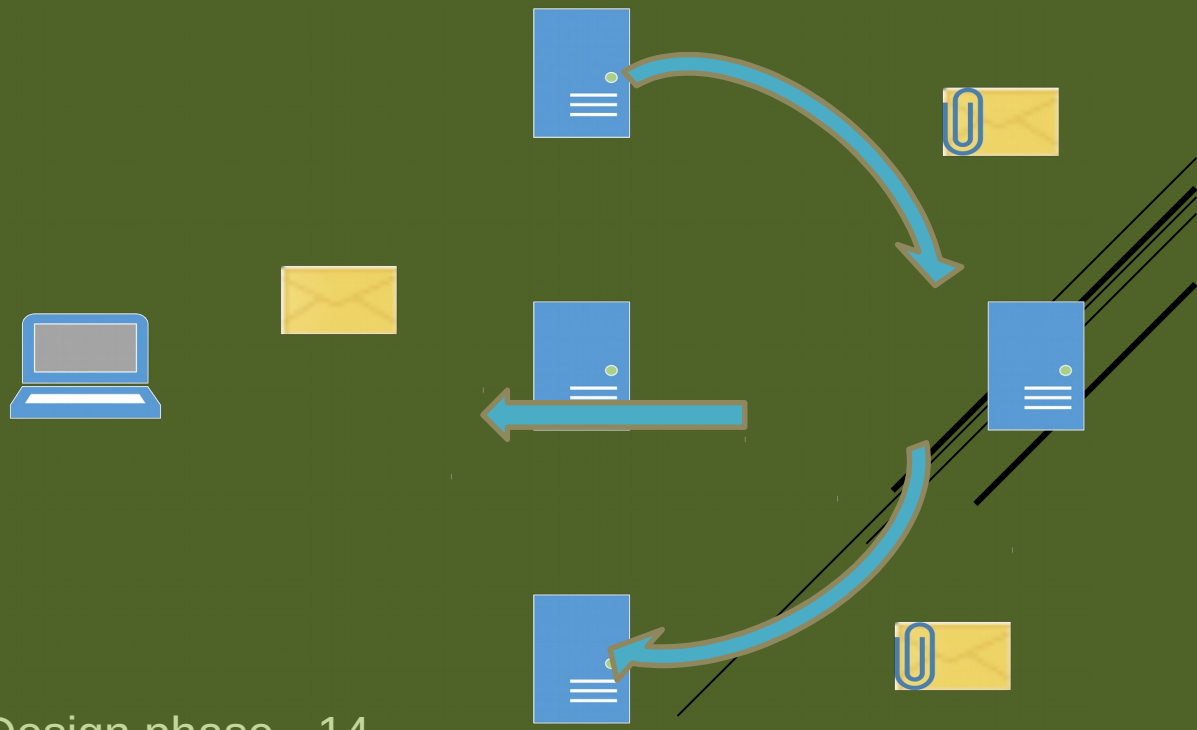
1. Server stores message then server appends its {ip:port} to the VIA list
2. Server finds a route to receiver according to the information collected from previous communications
3. Message is delivered to another server or client



Server receiving a message

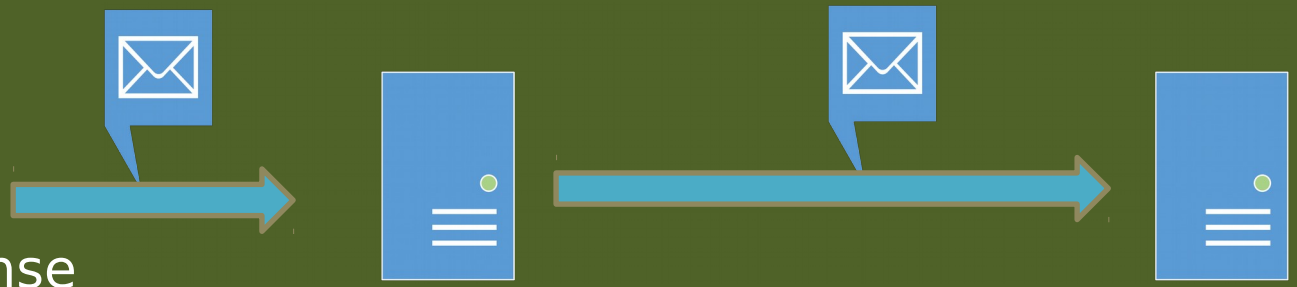
Message received from a server

1. Delivering message to client if target is already registered in current server
2. Forwarding a message with a extra Via tag to next proper server
   - Using Round Robin algorithm



Receiving response from another server

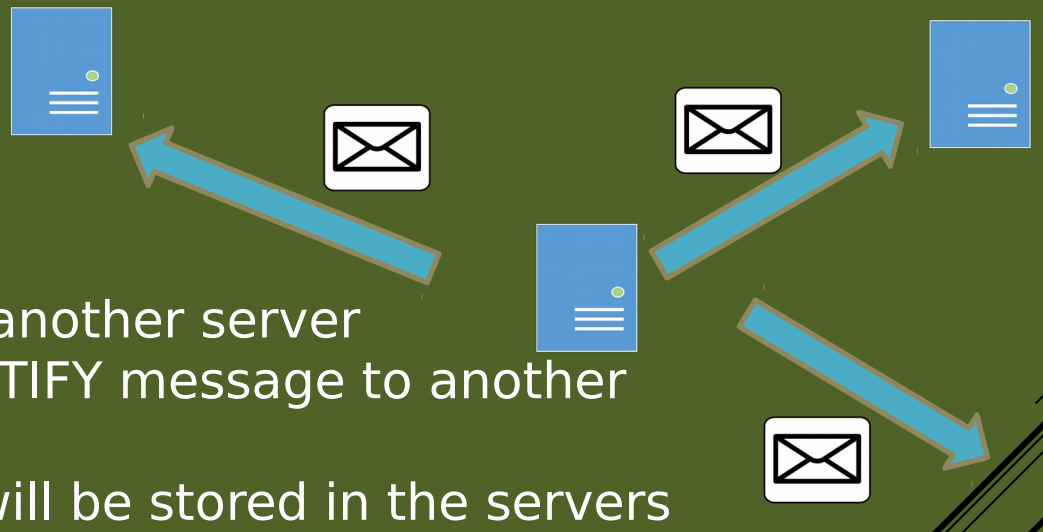
1. OK response
    1. Server pops a Via tag from message
    2. Forward message to last Via address



2. NOTOK response
    1. Server finds a proper route to target according to the records collected from previous communications
    2. Server forwards message a new Via tag to next server

Servers connections

1. Connecting to another server
    1. Server sends NOTIFY message to another server
    2. New credentials will be stored in the servers
    3. New server will be responded with an Ack message



2. Disconnecting from another server
    1. Server sends NOTIFY message to another server
    2. Old credentials will be stored in the servers
    3. Response with a Ack message
    4. Disconnection Proc is finished and client is free to
3. Makes any decision he wants