Distributed Systems

Today

- Definition
- Goals and pitfalls
- Grapevine – an early example

What is a *distributed system*?

- Very broad definition
  - A collection of independent, interconnected processors that communicate and coordinate their action by exchanging messages
- Why do you want one?
  - Resource sharing – both, physical resources and information
  - Computation speedup – to solve large problems, we will need many cooperating machines
  - Reliability – machines fail frequently
  - Communication – people collaborating from remote sites
  - Many applications are by their nature distributed (ATMs, airline ticket reservation, etc)

Distributed systems challenges

- Making resources available
  - The main goal of DS – making convenient to share resources
- Security
  - Sharing, as always, introduces security issues
- Providing transparency
  - Hide the fact that the system **is** distributed
  - Types of transparency
    - Access – What's data representation?
    - Location – Where's the resource located?
    - Migration – Have the resource moved?
    - Relocation – Is the resource being move?
    - Replication – Are there multiple copies?
    - Concurrency – Is there anybody else accessing the resource now?
    - Failure – Has it been working all along?
  - Do we **really** want transparency?

Distributed systems challenges

- Openness
  - Services should follow agreed-upon rules on component syntax & semantics
- Scalability
  - In numbers (users and resources), geographic span and administration complexity
  - Some useful techniques
    - Asynchronous communication
    - Distribution
    - Caching/replication
- Adding to the challenges, common false assumptions
  - The network is unreliable / secure / homogenous
  - The topology does not change
  - Latency is zero / Bandwidth is infinite /Transport cost is zero
  - There is one administrator

Loosely-coupled systems

- Earliest systems used simple explicit network programs
  - FTP (rcp): file transfer program
  - telnet (rlogin/rsh): remote login program
  - mail (SMTP)
- Each system was a completely autonomous system, connected to others on the network
- Even today, most dist. systems are loosely-coupled
  - Each CPU runs an independent autonomous OS
  - Computers don't really trust each other
  - Some resources are shared, but most are not
  - The system may look differently from different hosts
  - Typically, communication times are long

Closely-coupled systems

- A DS becomes more “closely-coupled” as it
  - Appears more uniform in nature
  - Runs a “single” operating system
  - Has a single security domain
  - Shares all logical resources (e.g., files)
  - Shares all physical resources (CPUs, memory, disks, printers, etc.)
- In the limit, a distributed system looks to users as a centralized timesharing system, but built of a distributed set of hardware and software components

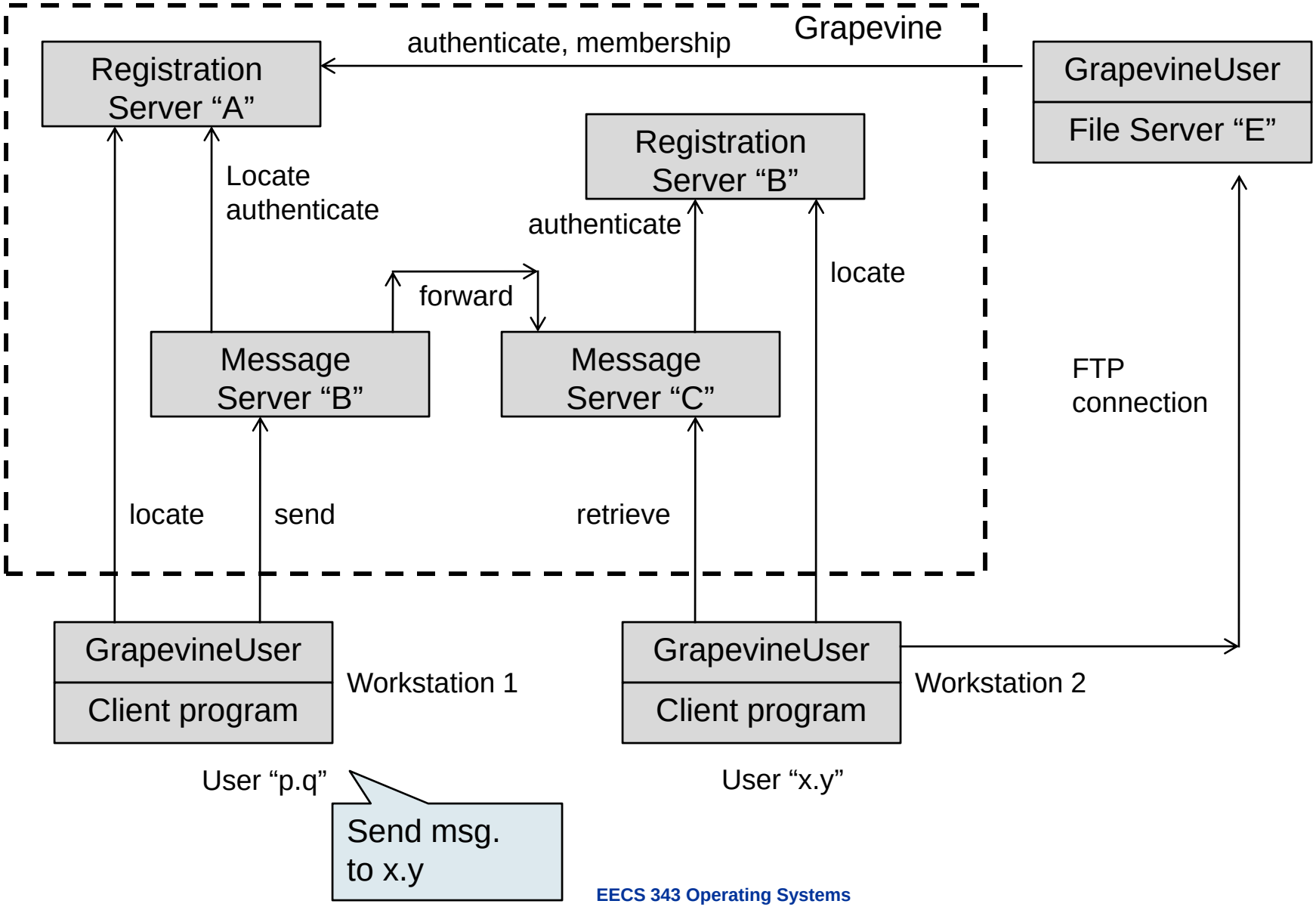
Tightly-coupled systems

- A “tightly-coupled” system usually refers to a multiprocessor
  - Runs a single copy of the OS with a single job queue
  - Has a single address space
  - Usually has a single bus or backplane to which all processors and memories are connected
  - Has very low communication latency
  - Processors communicate through shared memory

Grapevine

- Grapevine – Xerox PARC 1980
  - A loosely-coupled system
  - Provides *message delivery*, resource location, authentication, and access control
- Design goals
  - No assumptions on message content
  - Cannot rely on the integrity of clients
  - Once the system accepts mail, it will be delivered
  - Fault tolerance to single computer failures
  - Decentralized administration
- Components
  - GrapevineUser package on each client workstation
  - Registration Servers & Message Servers
  - Communication via Remote Procedure Calls

Grapevine: Functional diagram



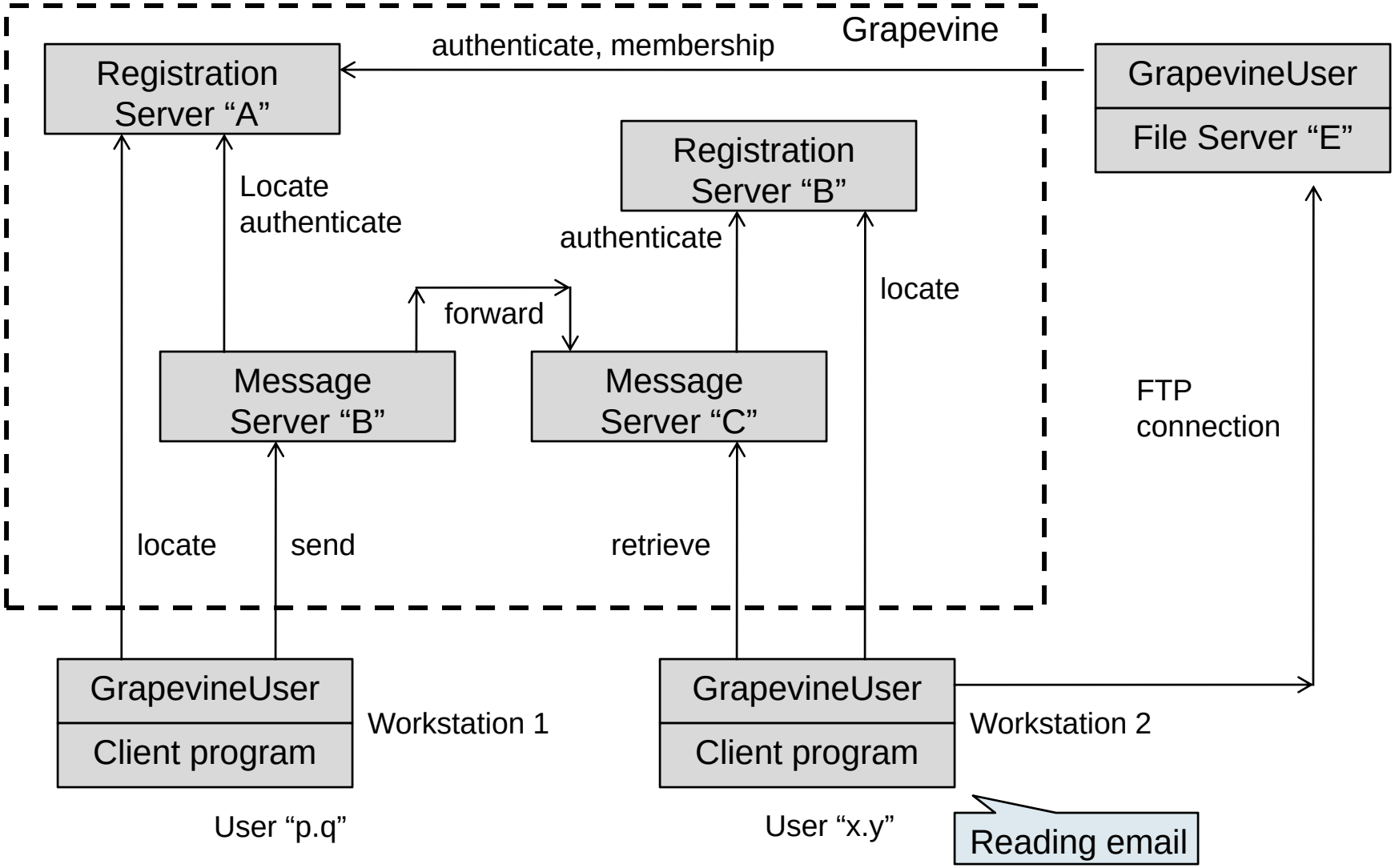
Grapevine: Sending a message

- User prepares message using mail client
- Mail client contacts GrapevineUser package on same workstation to send message
- GrapevineUser package
  - Contacts *any* Registration Server to get a list of Message Servers
  - Contacts *any* Message Server to transmit message
    - Presents source and destination userids, and source password, for authentication
      - Message Server uses *any* Registration Server to authenticate
    - Sends message body to Message Server
      - Message Server places it in stable storage & acknowledges receipt

Grapevine: Transport and buffering

- For each recipient, Message Server contacts *any* Registration Server to obtain list of Message Servers holding mail for that recipient
- Sends copy of the message to one of those Message Servers for that recipient (preferred site)
  - Hints for speeding up mappings
- If message cannot be deliver to some, reports this back to sender with an explanation

Grapevine: Functional diagram



Grapevine: Retrieving mail

- User uses mail client to contact GrapevineUser package on same workstation to retrieve mail
- GrapevineUser package
  - Contacts *any* Registration Server to get a list of each Message Server holding mail for the user (“inbox site”)
  - Contacts each of these Message Servers to retrieve mail
    - Presents user credentials
      - Message Server uses any Registration Server to authenticate
    - Acknowledges receipt of messages so that the server can delete them from its storage

Grapevine: Scalability

- Can add more Registration Servers
  - Eventual consistency in registration data base
  - Long term inconsistency solved with periodic anti-entropy mechanism
- Can add more Message Servers
- One thing didn't scale – handling of distribution lists
  - The accepting Message Server was responsible for expanding the list (recursively if necessary) and delivering to an appropriate Message Server for each recipient
  - Some distribution lists contained essentially the entire user community
- Different classes of transparency
  - Location, migration, replication