Distributed Systems: **Peer to Peer Networks**

Anca Jurcut
E-mail: anca.jurcut@ucd.ie

School of Computer Science and Informatics
University College Dublin
Ireland

From Previous Lecture…

- **Andrew File System**
- **Peer to Peer Networks:** recommended reading <Chapter 10: "Peer – To- Peer Systems"> of *"Distributed Systems, Concepts and Design" by George Coulouris, Jean Dollimore and Tim Kindberg, Fourth edition, 2005*
  - What is P2P?
    - A set of networked devices that have **equal responsibility** and which **share resources and workload** as necessary
    - Each device is both a **client** and a **server**
  - Examples of P2P Systems
    - File Sharing Applications e.g. Gnutella, BitTorrent, Kazaa, …
    - Instant Messengers and Telephony e.g. Skype, Yahoo, MSN
    - Data Processing Services e.g. Seti@Home
  - Characteristics of P2P Systems
    - The key problem to be addressed when building a P2P system is **how to locate the resources** that you need

From Previous Lecture…

- Accessing (Locating) Resources in P2P Systems
- Types of P2P Systems: 3 main Types
  - **Centralized Systems:** peer connects to server which coordinates and manages communication. First generation systems addressed this problem through a **centralised server** that contained a global resource list (not scalable, legal issues).
    E.g. SETI@home
  - **Decentralized Systems**: peers run independently with no central services; discovery is decentralized and communication takes place between the peers. Second generation systems **decentralised the resource list** to improve robustness (and to sidestep the legal issues ;-)
    E.g. Gnutella, Pastry
  - **Hybrid systems**: peers connect to a server to discover other peers; peers manage the communication themselves (e.g. Napster). Third generation systems have focused on making file sharing an **anonymous activity** and on techniques for improving **resource search** speed.

- **P2P Middleware Systems**
  - Aims: to be **application independent; scalability:** to share resources, storage and data present in computers *at the edges of the internet* on a global scale
  - Provide reference architecture that allowed researchers to focus of specific P2P issues

From Previous Lecture…

- P2P Middleware systems are based on 2nd generation architectures
- **Resource discovery (searching):** algorithms used to carry out the search *within P2P middleware systems* are known as **routing overlay algorithms**
- **Routing Overlay**
  - **Basic idea**
  - **Resource GUIDS** (globally unique identifier(s))
  - **Distributed Hash Tables (DHT)**
- **Routing Overlay versus IP Routing**
- **Case Study 1: Pastry**
  - **Pastry GUIDs**
  - **Pastry Routing Overlay: The Simple Version Algorithm and The Full Routing Algorithm**
  - **Analysis of Pastry**
  - **Pastry and PAST**

Distributed Systems:
Case Study 2: BitTorrent

Introduction

- Peer to Peer File Sharing Protocol used for distributing large amounts of data
- One of the most common protocols used today - accounts for ~27% to 55% of all internet traffic (feb, 2009)
- Released in the summer of 2001 by Bram Cohen
- Basic Idea:
  - Chop file into many pieces
  - Replicate DIFFERENT pieces on different peers as soon as possible
  - As soon as a peer has a complete piece, it can trade it with other peers
  - Hopefully, we will be able to assemble the entire file at the end
- Consequence: can distribute large files without the heavy load on the source computer and network

Introduction

- BitTorrent efficient content distribution system using *file swarming (i.e. File Sharing)*
  - *- swarm = set of peers that are participating in distributing the same files*

- Usually **does not perform** all the functions of a typically P2P system such as **searching**

File Sharing – how it works

- To share a file or group of files, a peer first creates a **.torrent file**
- **.torrent file** = small file that contains:
  - meta data about file(s) to be shared
  - information about the tracker - computer that coordinates the file distribution
- Peers first obtain a **.torrent file**, and then connects to the specified **tracker** - which tells them from which peers to download the *pieces* of the file(s).

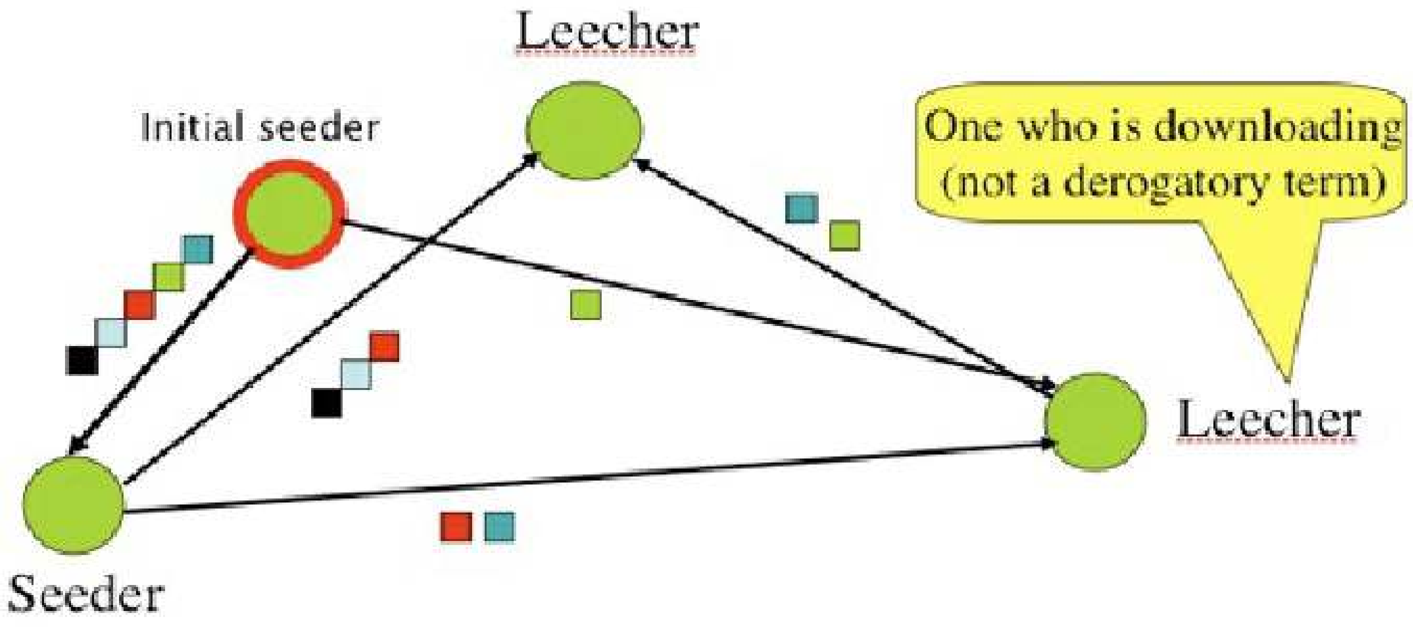
Basic Components

- **Seed**
  - Peer that has the entire file
- **Leech**
  - Peer that has an incomplete copy of the file
- **.torrent File**
  - the URL of the tracker
  - Pieces of the file: <hash1, hash2, .. , hash n>
  - piece length
  - name of the file
  - length of the file
- **A Tracker**
  - central server - keeps a list of all peers participating in the swarm
  - coordinates the file distribution
  - Allows peers to find each other
  - status information (i.e. completed or downloading)
  - Returns a random list of peers

Seeder v's Initial Seeder

Seeder = a peer that provides the complete file.

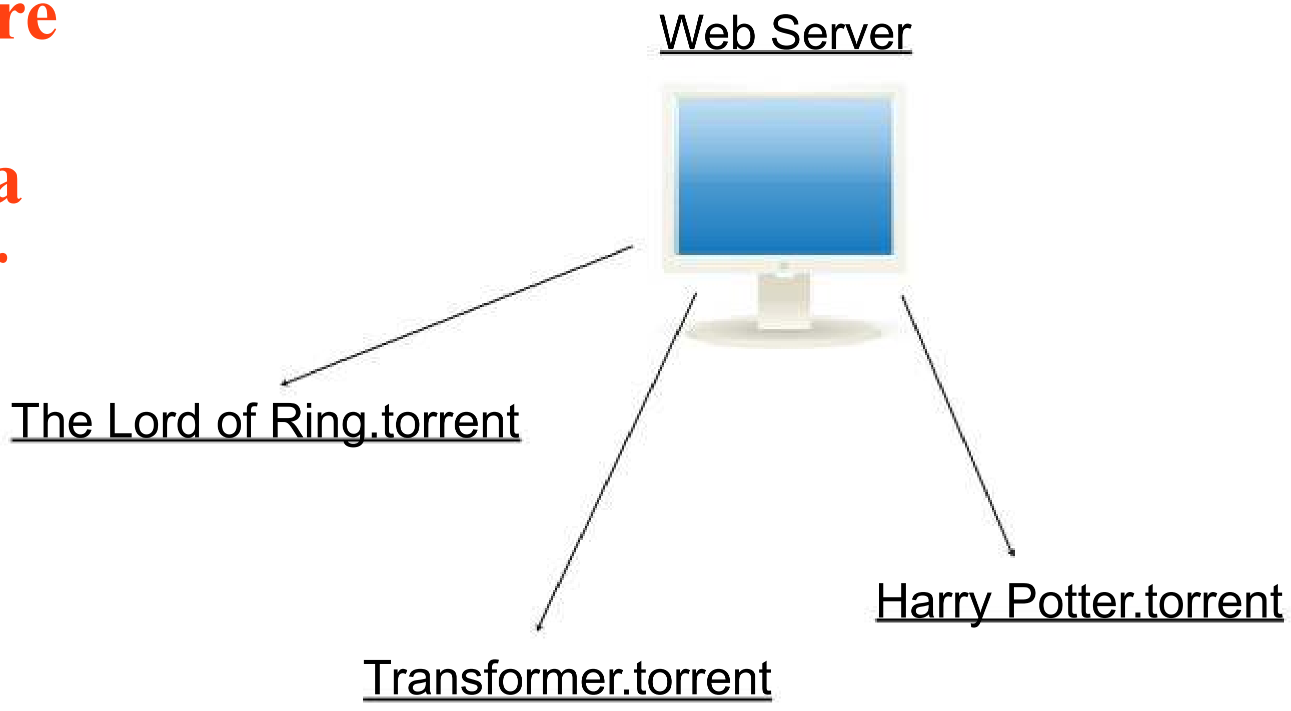
Initial seeder = a peer that provides the initial copy.



File Sharing

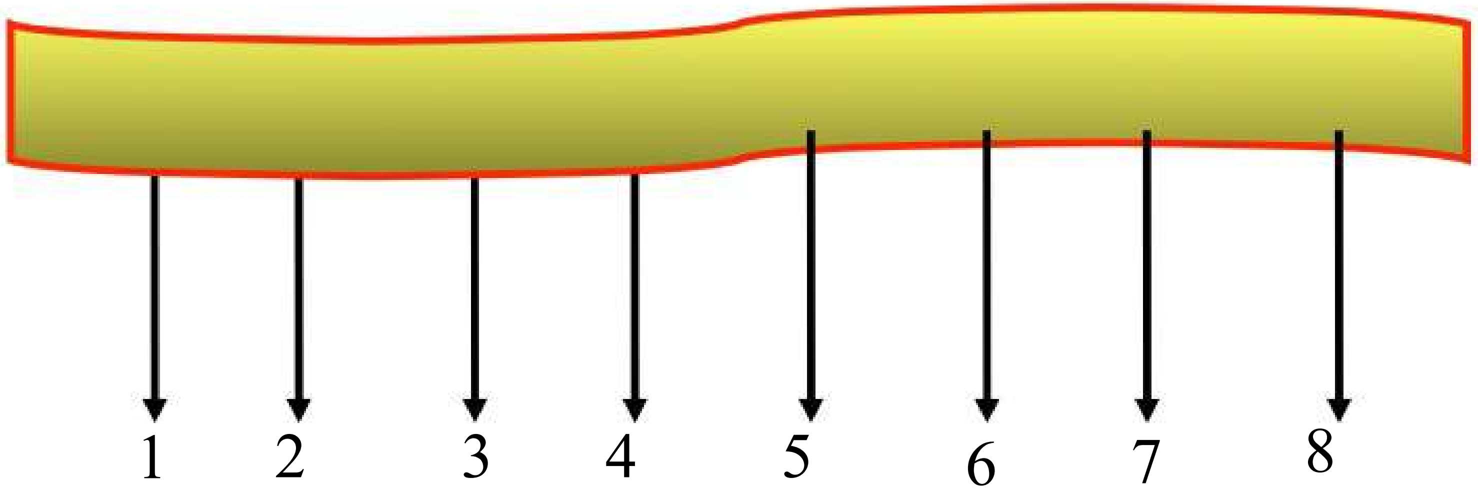
- Obtain a .torrent on a public domain site such as:
  - http://bt.LOR.net
  - http://bt.HarryPotter.com/

.torrents are typically hosted on a web server



File Sharing…

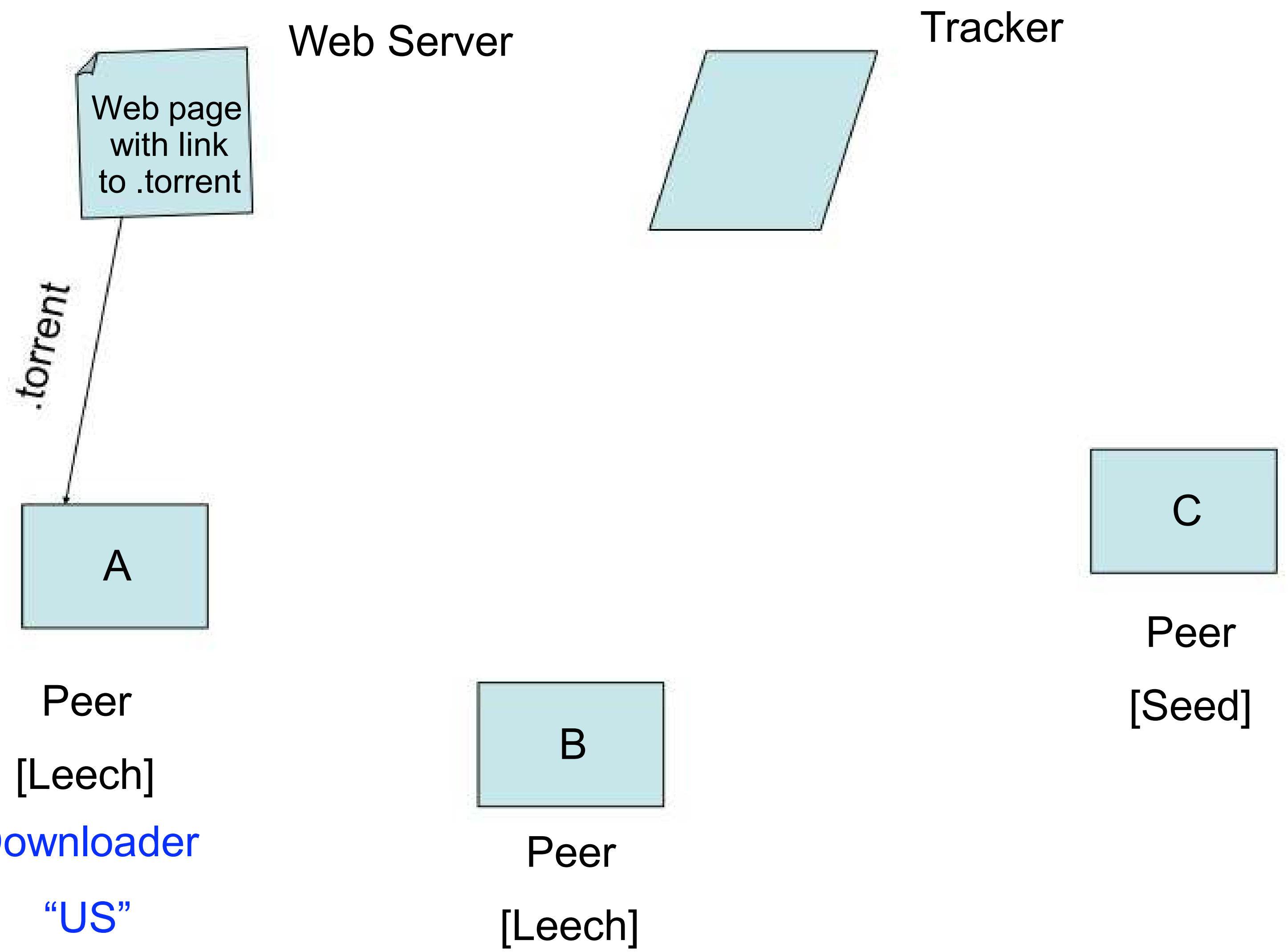
- Large files are broken up into pieces of sizes between 64KB and 1MB
  - normally 512KB segments are used



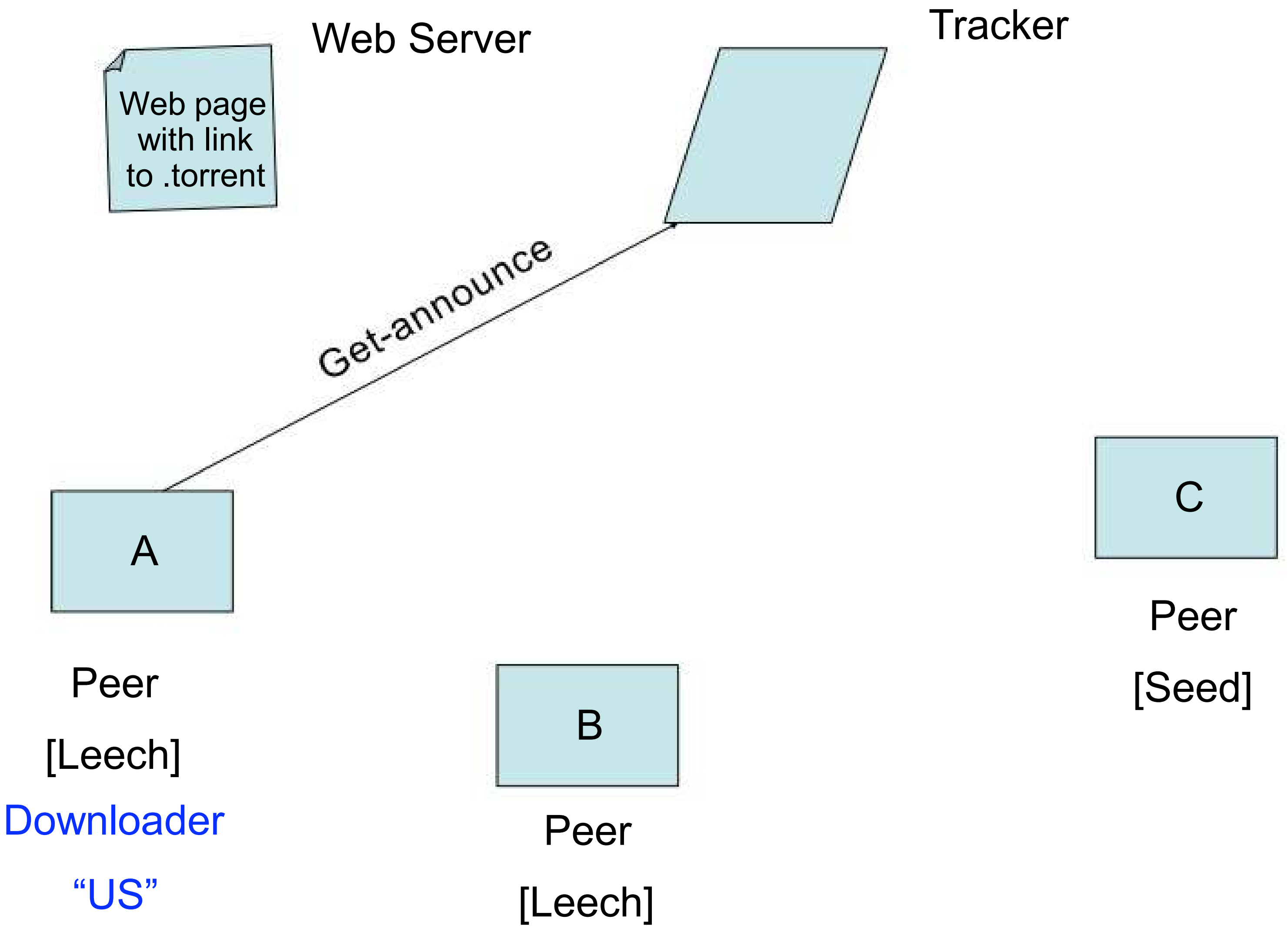
Basic Idea

- Initial seeder chops file into many pieces.
- Leecher first locates the **.torrent** file that directs it to a **tracker**, which tells which other peers are downloading that file. As a leecher downloads pieces of the file, replicas of the pieces are created. ***More downloads mean more replicas available***
- As soon as a leecher has a complete piece, it can potentially share it with other downloaders.
- Eventually each leecher becomes a seeder by obtaining all the pieces, and assembles the file. Verifies the checksum.

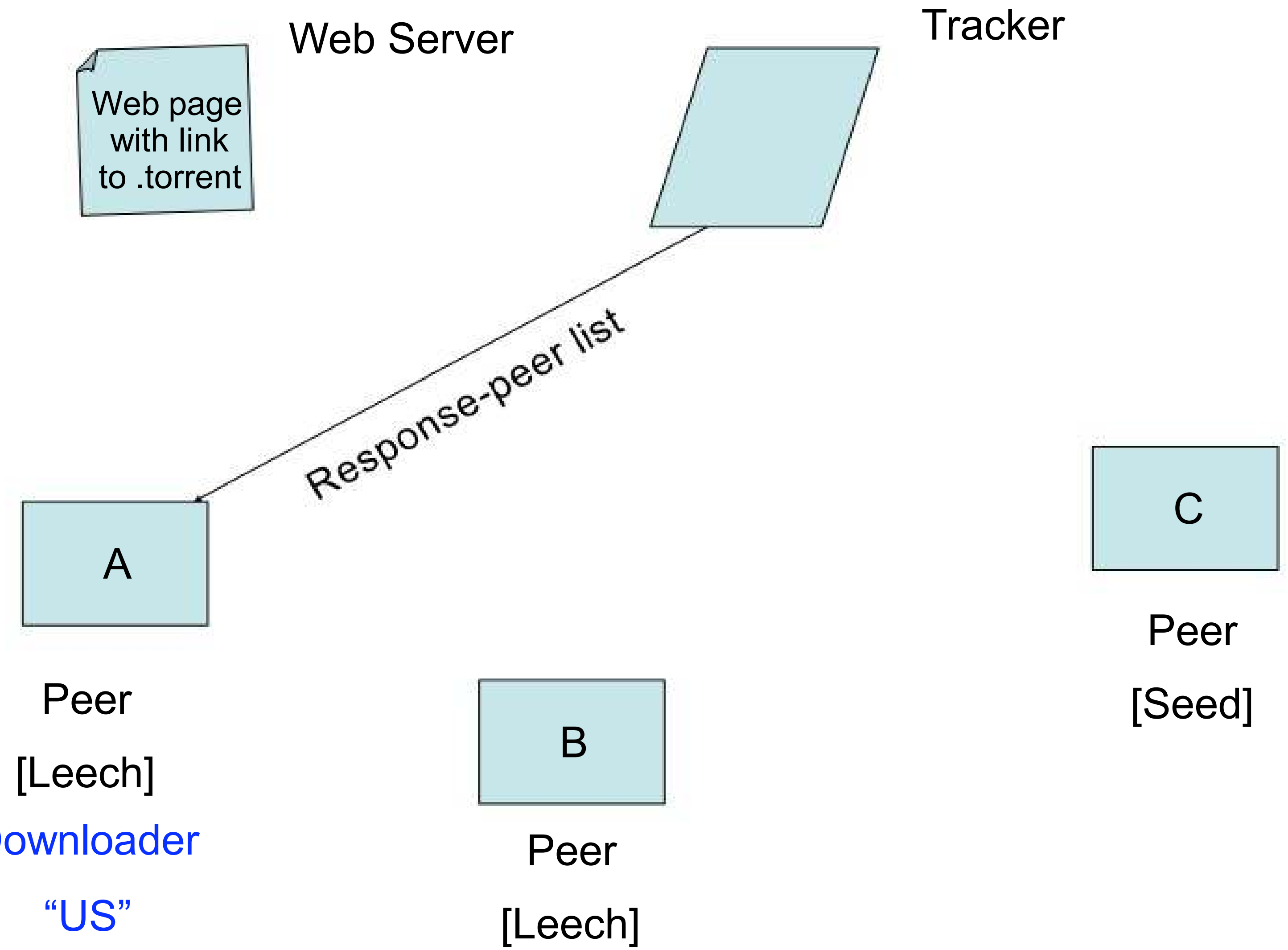
Overview – System Components



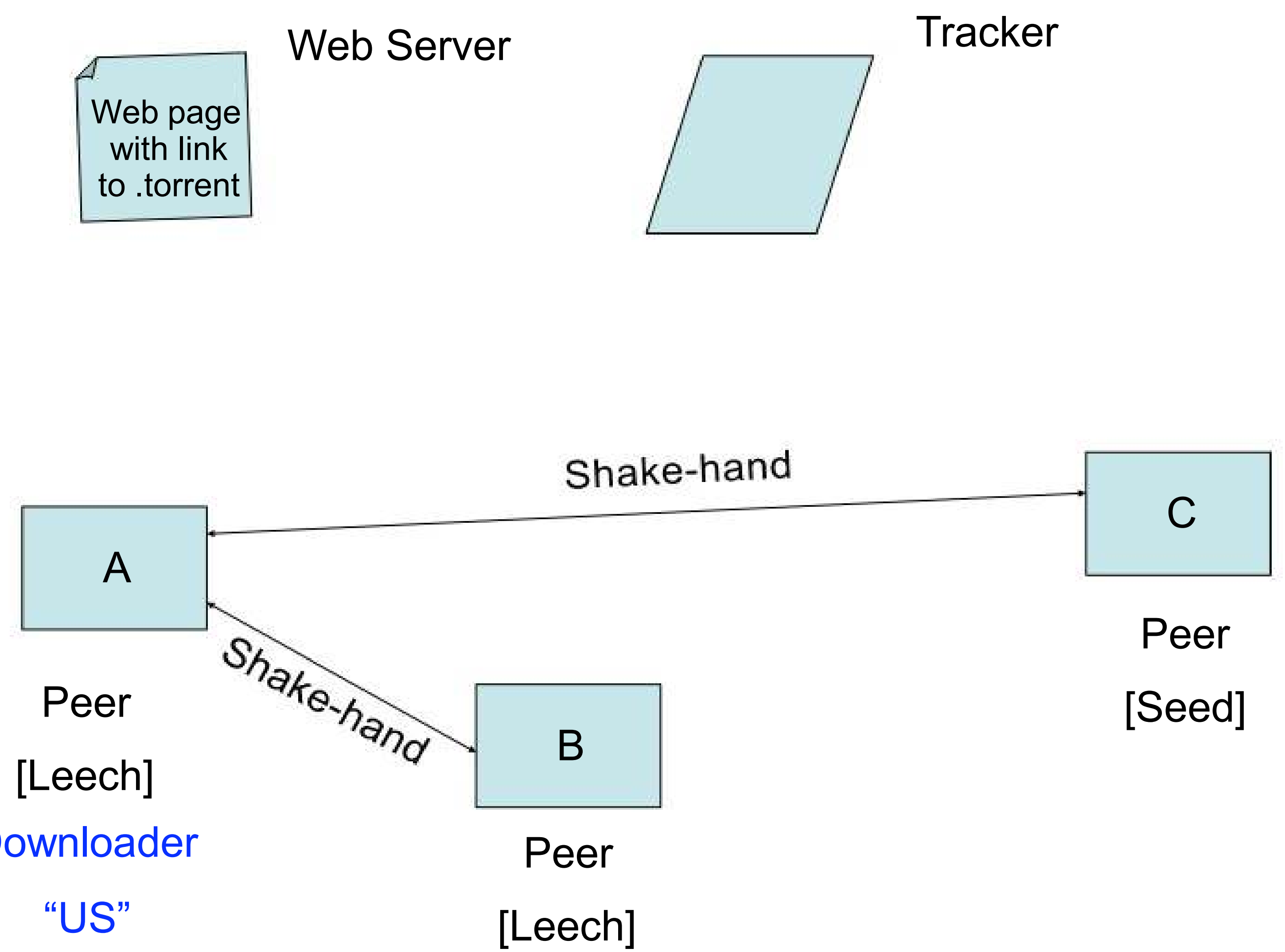
Overview – System Components(2)



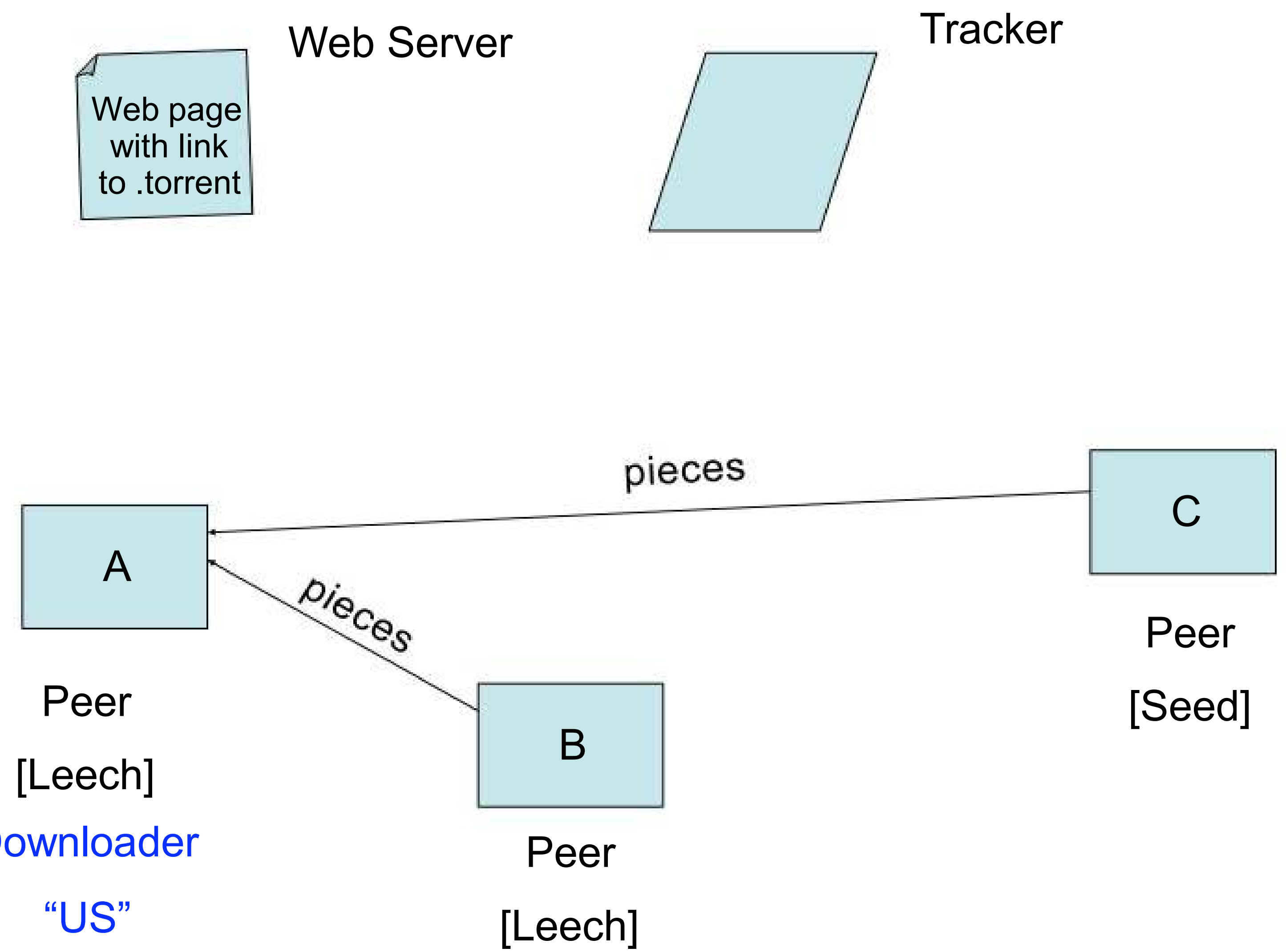
Overview – System Components(3)



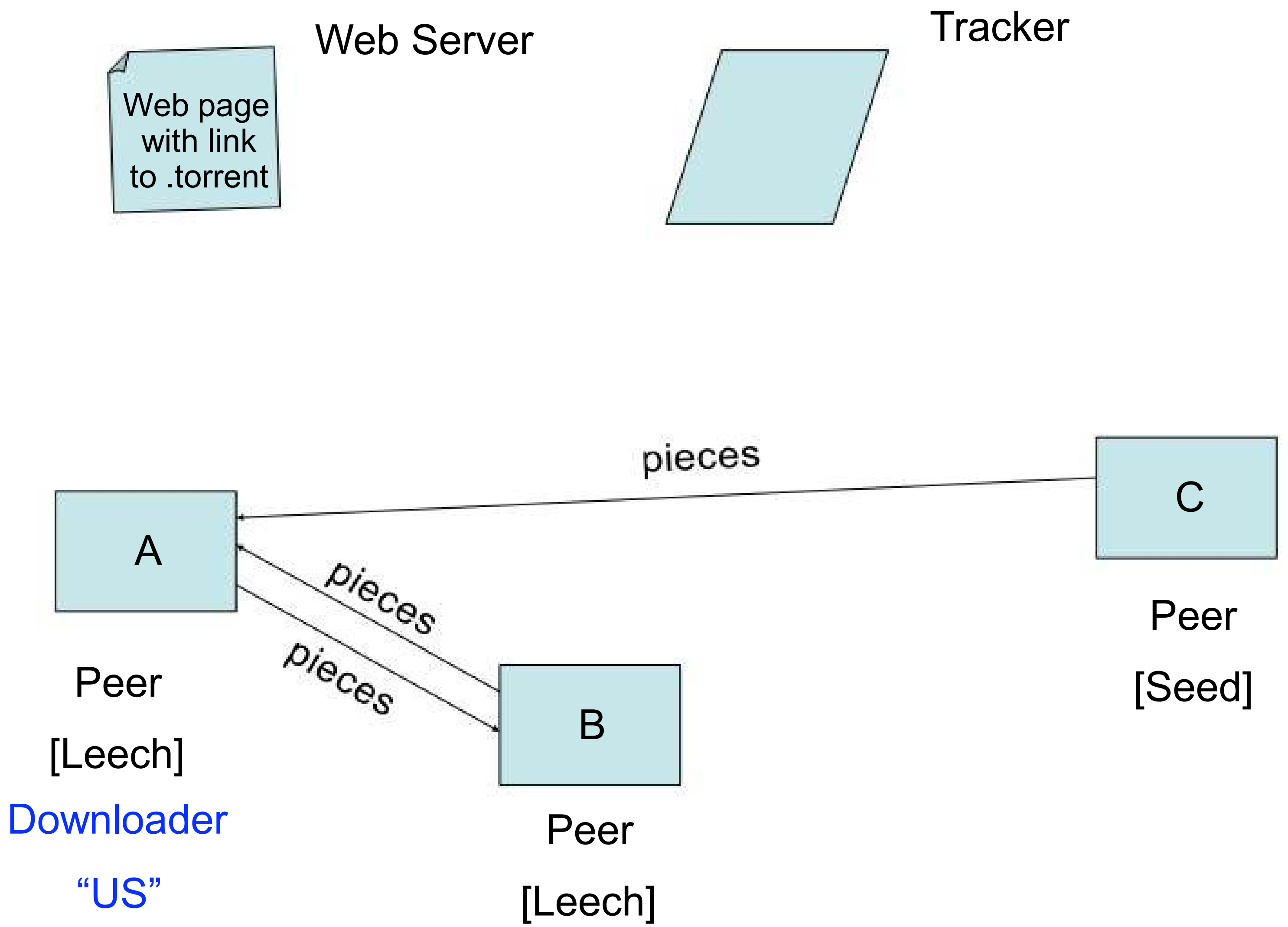
Overview – System Components (4)



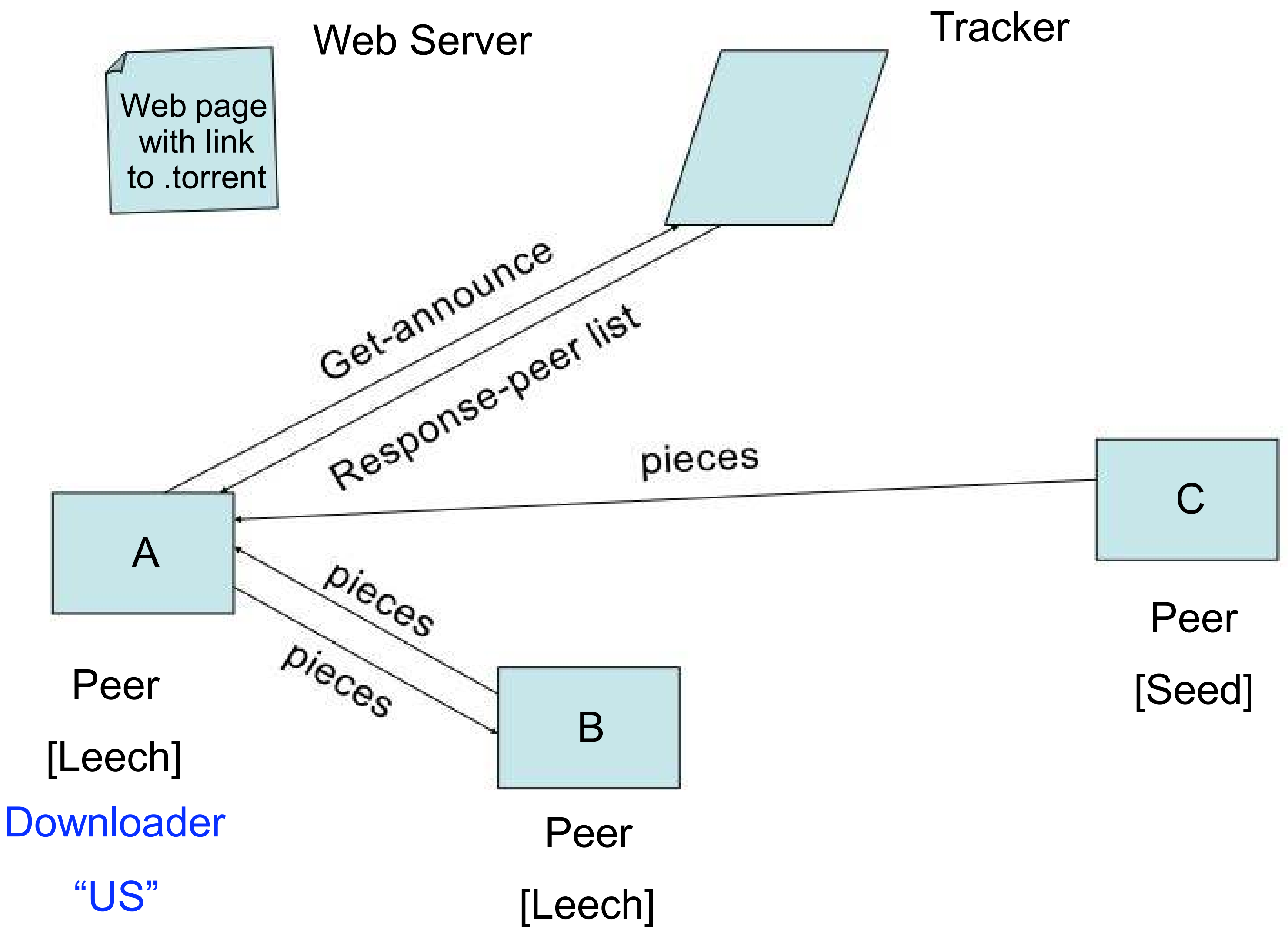
Overview – System Components (5)



Overview – System Components (6)



Overview – System Components (7)



Tracker Protocol

- communicates with clients via HTTP/HTTPS
- client GET request
  - info_hash: uniquely identifies the file
  - peer_id: chosen by and uniquely identifies the client
  - client IP and port
  - numwant: how many peers to return (defaults to 50)
  - status: bytes uploaded, downloaded, left
- tracker GET response
  - interval: how often to contact the tracker
  - list of peers, containing peer id, IP and port
  - status: complete, incomplete

Goals

- Efficiency

  - Fast downloads

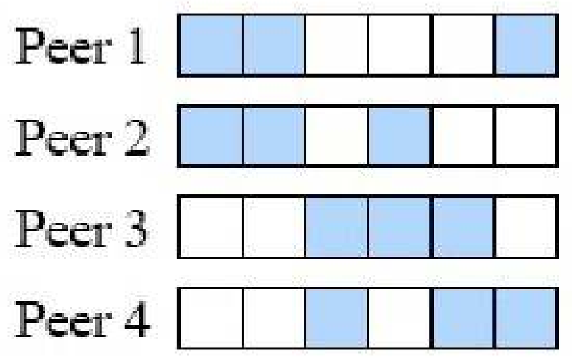
- Reliability

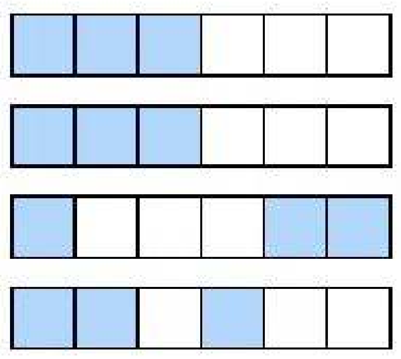
  - tolerant to dropping peers
  - ability to verify data integrity (SHA-1 Hashes)

Efficiency

- Ability to download from many peers yields to fast downloads
- Minimise **piece overlap** among peers to allow each peer to exchange with as many other peers as possible
- To minimise piece overlap:
  - download random pieces
  - prioritize the rarest pieces, aiming towards **uniform piece distribution** (all pieces are copied across peers the same number of times)

Piece Overlap





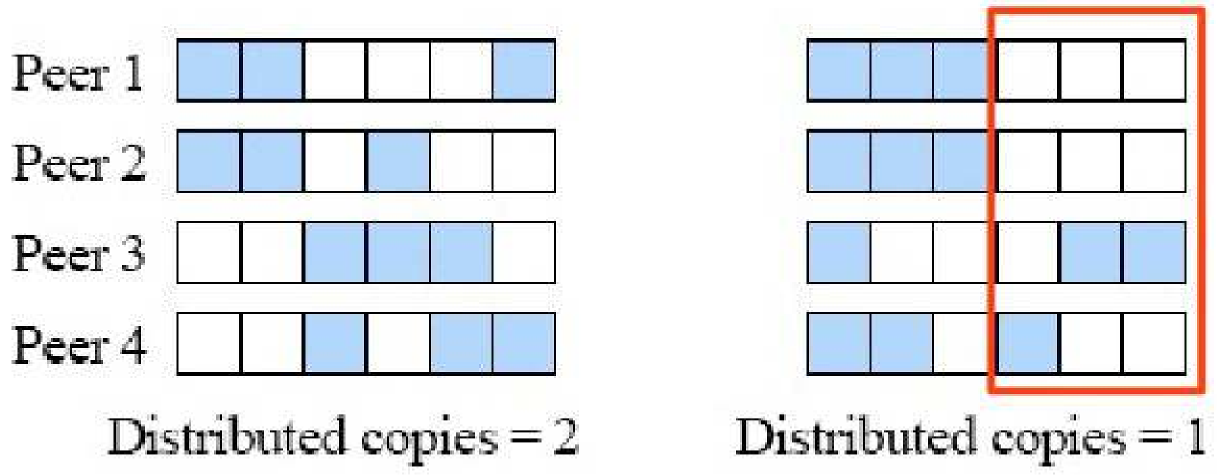
- Small overlap
  - Every peer can exchange pieces with all other peers
  - The bandwidth can be well utilised
- Big overlap
  - Only a few peers can exchange pieces
  - The bandwidth is under utilised

Reliability

- Be tolerant against dropping peers
  - each dropped peer means a decreased piece availability
- Maximise piece redundancy
  - maximise the number of distributed copies of each piece
  - ensures high availability

Distributed Copies

- The number of distributed copies is the number of copies of the rarest piece



Distributed Copies

- To maximise the distributed copies - maximise the availability of the rarest pieces
- To increase the availability of a piece - download it
- To maximise the distributed copies:
    - download the rarest piece first

Rarest First

- The piece picking algorithm used in Bittorrent is called *rarest first*
- Picks a **random** piece from the set of **rarest** pieces
- No peer has global knowledge of piece availability, it is approximated by the availibility among neighbours

Random First Piece

- Initially, a peer has nothing to trade
- Important to get a complete piece ASAP
- Rare pieces are typically available at fewer peers, so downloading a rare piece initially is not a good idea
- Policy: Select a random piece of the file and download it

Rarest Piece First

Policy: Determine the pieces that are most rare among your peers and download those first

This ensures that the most common pieces are left till the end to download

Rarest first also ensures that a large variety of pieces are downloaded from the seed

BitTorrent Summary

- BitTorrent works by using trackers to maintain lists of seeds and leechers associated with each shared file.

- In contrast with Napster, the BitTorrent server does not contain information about the names of the files that are being shared.
  - It only uses the info_hash identifier
- Further, BitTorrent does not download whole files - it downloads only parts of files.
  - This means that it is very difficult to identify who is downloading what files…
  - At the same time, it provides significant improvements in download times!!
- Aims: Efficiency and Reliability

Distributed Systems:
Case Study 3: Plaxton Mesh

For Reading -- will not be questioned in the exam

Overview

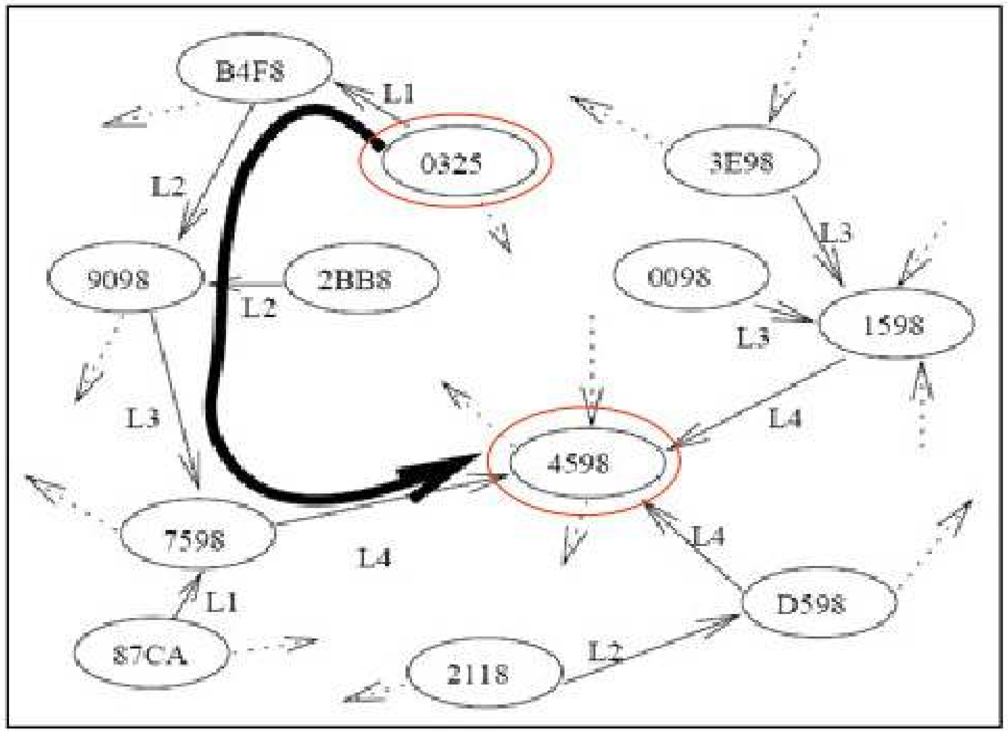
- Plaxton Mesh - earliest known proposal for a scalable P2P network (1997)
  - A distributed data structure which supports an overlay network for:
    - Locating named objects
    - Routing messages to these objects
  - Plaxton Mesh is static - no node arrival, departure or failure
- Concepts in Plaxton
  - Nodes - Act as routers, clients and servers simultaneously
  - Objects - Stored on servers, searched by clients
  - Routing -- refers to the task of forwarding messages to such objects
- Objects and nodes have **unique, location independent names**
  - Random fixed-length bit sequence in some base
  - Typically, hash of hostname or of object name
- What Plaxton does:
  - Given **message M** and **destination D**, a Plaxton mesh **routes M** to the node whose name is **numerically closest to D**

Routing

- Routing is done incrementally, digit by digit

- In each step, message sent to node with address one digit closer to destination.

- Example - node 0325 sends M to node 4598
  - Possible route:
    - 0325
    - B4F**8**
    - 90**98**
    - 7**598**
    - **4598**



Routing Tables

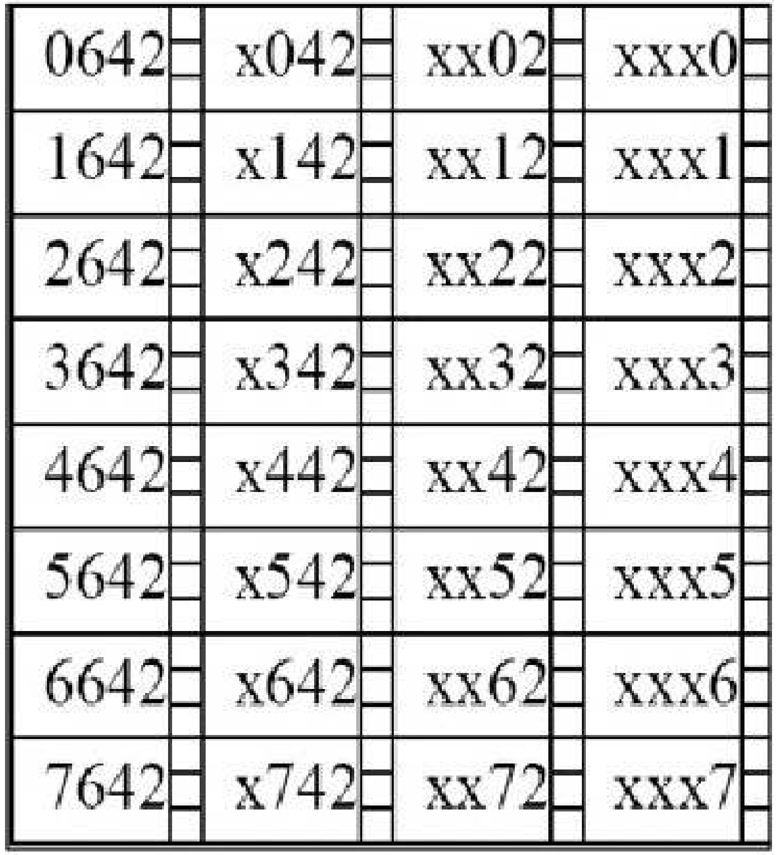
- Each node has a neighbor map
  - Used to find a node whose address has one more digit in common with the destination
- Size of the map: B*L, where:
  - B - base used for the digits of the address. Eg. 8 in octal base
  - L - length of the addresses
- 4 digit addresses in octal base -> 8*4 entries.

Routing Tables

- x are wildcards. Can be filled with any address.
  - Each slot can have several addresses. Redundancy.

- Example: node 3642 receives message for node 2342
  - Common postfix: XX**42**
    - Look into second column.
  - Must send M to a node one digit "closer" to the destination.
    - Any host with an address like X342.
  - Look in line 4

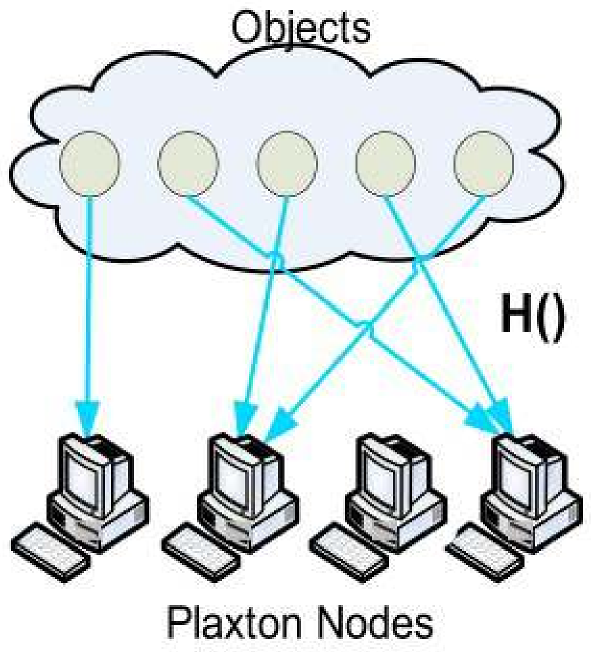
Node 3642



0642	x042	xx02	xxx0
1642	x142	xx12	xxx1
2642	x242	xx22	xxx2
3642	x342	xx32	xxx3
4642	x442	xx42	xxx4
5642	x542	xx52	xxx5
6642	x642	xx62	xxx6
7642	x742	xx72	xxx7

Publishing and Locating Objects

- Server S has an object O
  - Wants to make it known to clients
- Strength of a Plaxton Mesh
  - No central directory
  - Directory distributed between nodes
- How? Hash function
  - H: (object names) -> (node addresses)



Publishing and Locating Objects

- **Server S wants to publish object O**
  - S computes H(O), the object ID of O (hash of the name).
  - S sends a PUBLISH message to node H(O).
    - Node H(O) might not exist. No problem. Message forwarded deterministically to node X with address closest to H(O).
  - X - *root node* for object O
  - Nodes visited by the PUBLISH message associate H(O) with physical address of S (including root node)
  - These nodes are the **tree** of the object O

Publishing and Locating Objects

- A client C wants to find object O.
  - Evaluates H(O), using same algorithm as the server
  - Sends a QUERY message to H(O)
    - Forwarded to X, the object root.
    - The object root forwards the message to S.
  - Shortcut - Nodes in the tree of O also know location of O
    - If QUERY message reaches one of these nodes, it is forwarded directly to S.

Analysis of Plaxton Mesh

- **Advantages**
  - Simple fault handling - Several possible routes.
  - Scalable - No central point, routing done with local information

- **Disadvantages**
  - Global knowledge required to build routing tables
  - Static network
  - Root node vulnerability

- **More recent research overcomes most of disadvantages**
  - Dynamic networks
  - Redundant storage

Distributed Systems: Replication Systems

Introduction

- Replication is concerned with:

  “the maintenance of multiple copies of data at multiple computers”

- It is a key issue in the design of **effective** distributed systems.

- It can be used to provide:
  - Enhanced Performance
  - High Availability
  - Fault Tolerance

Key Features

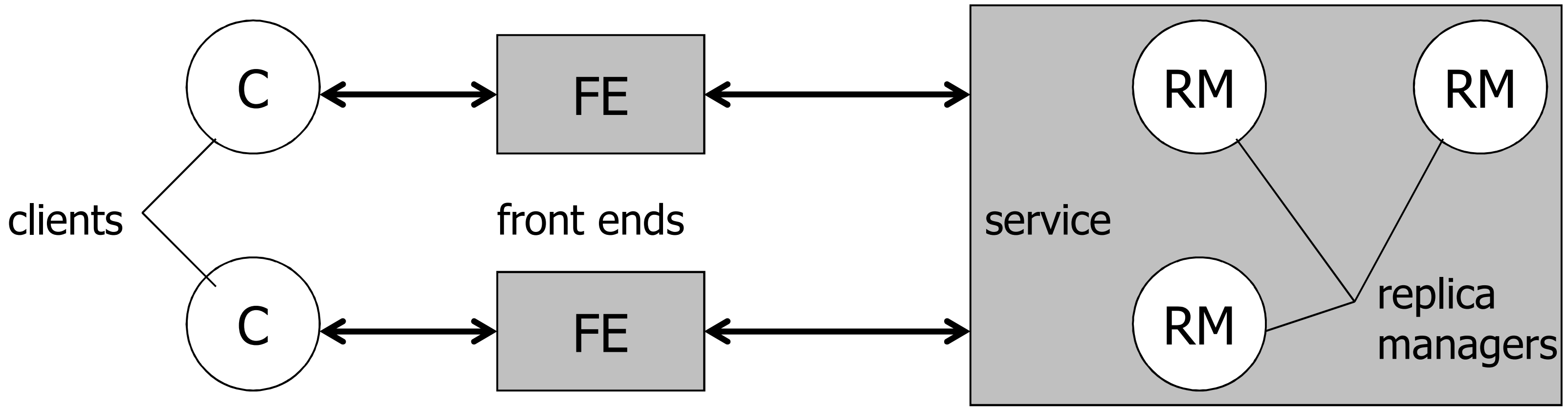
- **Replication Transparency:**
  - The replication process is hidden from the user.
    - Physical vs Logical Objects
    - Clients expect operations to return only one set of values.

- **Consistency**:
  - Correctness of the operations performed on a collection of replicated objects.
    - Can we guarantee that the operations produce results that conform to the specification of the service?

Replication Systems

- A **Replication System** is the sub-system (service) of a distributed system that is concerned with replication of data.
  - Data is modelled as objects
  - A logical object implemented as a collection of physical objects known as **replicas**.

- Implemented as a set of **Replica Managers**.
  - Act as a container for the set of replicas that are stored on a given computer.
  - Can perform operations directly upon this set of replicas.

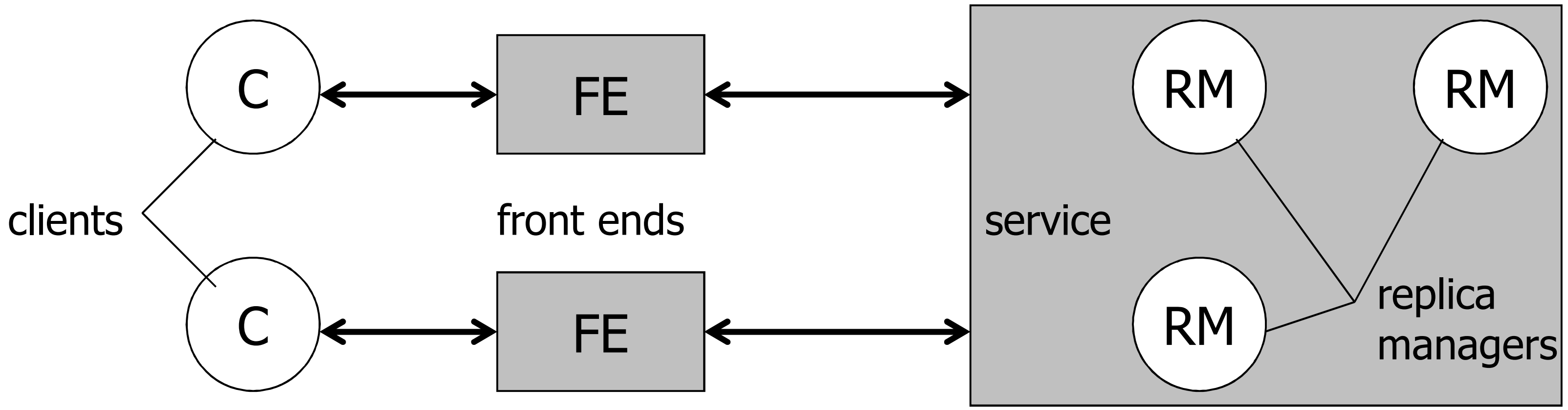


Replication Systems

- The collection of Replica Managers (RMs) provide a replication service to the clients.

- However, the clients see a service that gives them access to a set of logical objects.
  - E.g. Diaries, Bank Accounts, Music Files

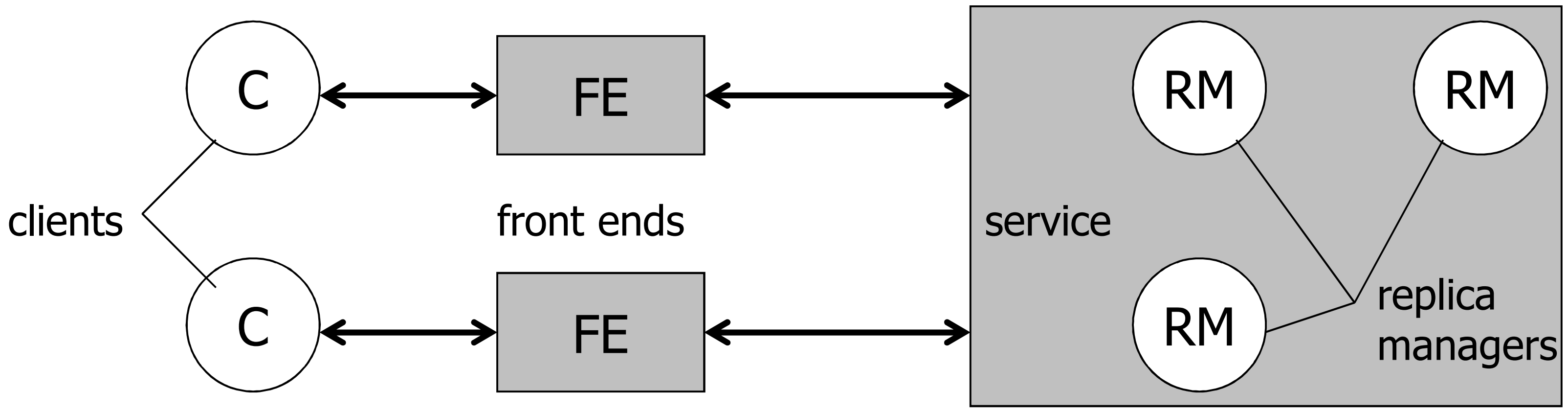
- Ideally, each RM contains a replica of **every** logical object managed by the service.



Replication Systems

- Clients request a series of operations (invocations) on one or more logical objects.
  - E.g. change time of meeting with boss, deposit 2000RMB in to John’s current account, add a meeting, download a song, …

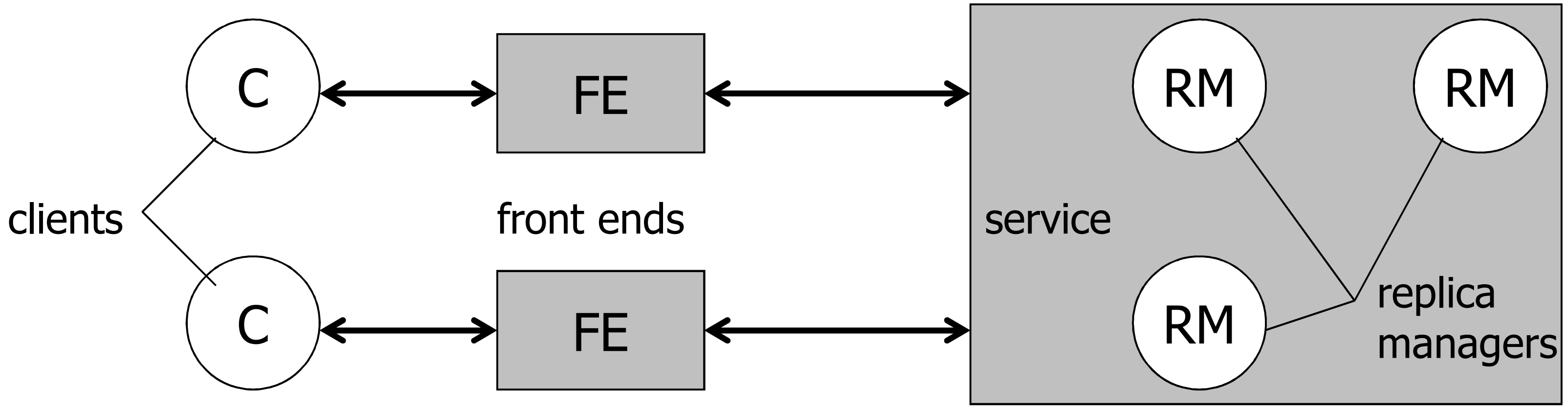
- An operation involves a combination of one or more reads of objects and updates of objects.
  - Operations that involve no updates are called **read-only requests**.
  - Operations that update an object are called **update requests**.



Replication Systems

- Client requests are first passed to a Front End component.
- This component acts as a **façade** – it hides the interactions with the RMs from the client.
  - i.e. it makes the service replication transparent.

- The façade can be implemented as:
  - Part of the client (i.e. it is a client side API, e.g. Venus in the AFS)
  - Part of a RM (i.e. it is a distinct layer in the RM – like VFS of Unix)
  - It can be an independent component of the system



Replication Systems

- Handling a request to perform an operation on a logical object normally involves 5 steps:
  - **Request**: FE issues the request to one or more RMs
    - Single message or Multicast
  - **Coordination**: RMs coordinate to execute the request consistently.
    - They agree on whether or not the request is to be applied
    - They decide on the ordering of the request relative to others
    - FIFO, Causal, Total, …
  - **Execution**: The RMs execute the request
    - This may be done tentatively (i.e. it can be undone later)
  - **Agreement**: The RMs reach consensus on the effect of the request
  - **Response**: One or more RMs respond to the FE
    - Single/Multiple Responses
    - Response Acceptance/Synthesis (where necessary)

- The ordering of these steps and the choices made when implementing them depends upon the application…
  - E.g. mobile computing / fault tolerance

Coordination Techniques

- First-In First-Out (FIFO) Ordering.
  - If a FE issues request r then request r’, then any correct RM that handles r’ handles r before it.

- Causal Ordering.
  - If the issue of request r happened-before the issue of request r’, then any correct RM that handles r’ handles r before it.

- Total Ordering.
  - If a correct RM handles r before request r’, then any correct RM that handles r’ handles r before it.

- Hybrid Orderings.
  - FIFO + Total, Causal + Total

- Most Replication Systems employ FIFO Ordering.

Recoverability

- A key requirement of a Replication System is **recoverability**.
  - That is, how well a system that has failed can recapture its state immediately prior to the failure.
  - Failure to capture the prior state can result in loss of information
    - E.g. failure of a file system while writing to a file, …

- Inability to recover from failure often arises due to the system being in an **inconsistent state**.

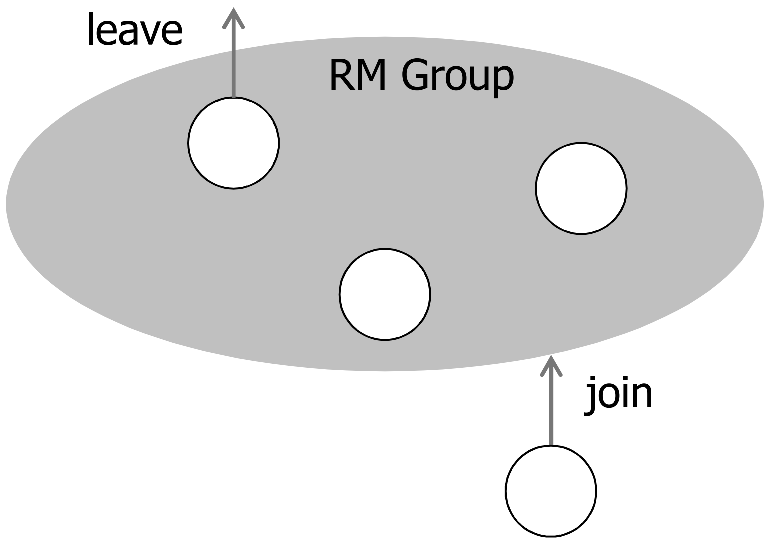
- To achieve recoverability we require that each operation carried out by a RM be **recoverable**.
  - That is, these operations must not leave inconsistent results if they fail part-way through.

State-Based Replication

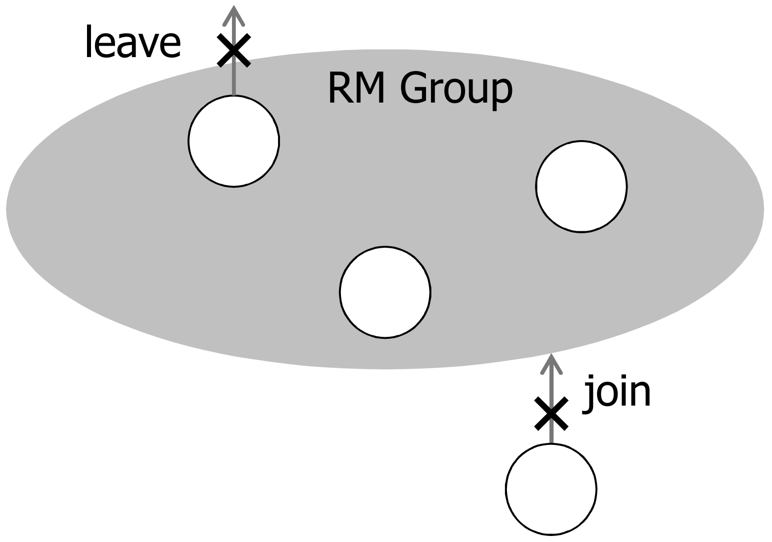
- One approach to supporting recoverability in Replication Systems is to view RMs as a **state machine**.
  - Each replica has a state and the RM applies **atomic operations** to replicas based on their current state.
- Resulting Replica state = Initial State + Operation History
- In other words, each state is a **deterministic function** of their initial state and the sequence of operations applied to it.

Static versus Dynamic Systems

- Dynamic Replication Systems:
  - These are open systems
  - When a Replica Manager crashes, it is deemed to have left the system



- Static Replication Systems:
  - The set of Replication Managers is fixed at design time.
  - Replica Managers do not crash…
    - …they simply cease operating for an indefinite period.



Example: Diary System

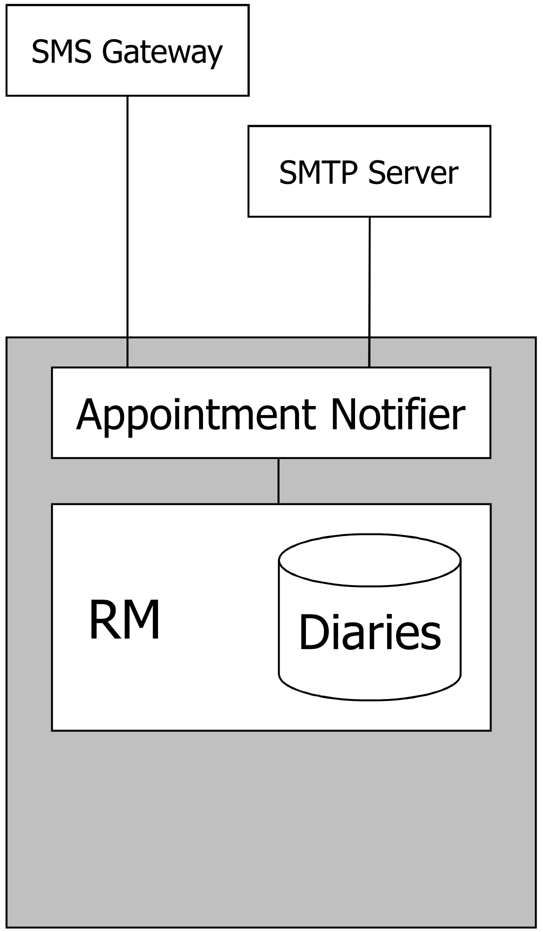
- A company employs a diary management system for scheduling meetings between staff.
  - The system maintains a separate diary for each member of staff.
  - Access to the system is via a client that is installed on every machine.

- To improve performance, availability, and fault tolerance:
  - A replication service that downloads a copy of every diary onto each client is used (supports dynamic RM)

- Key operations supported by the replication service are:
  - creating a meeting, cancelling a meeting, rescheduling a meeting, and getting meeting information.

- Meeting announcements are managed by an external (to the replication service) component that (periodically) polls the state of the diaries within the system.

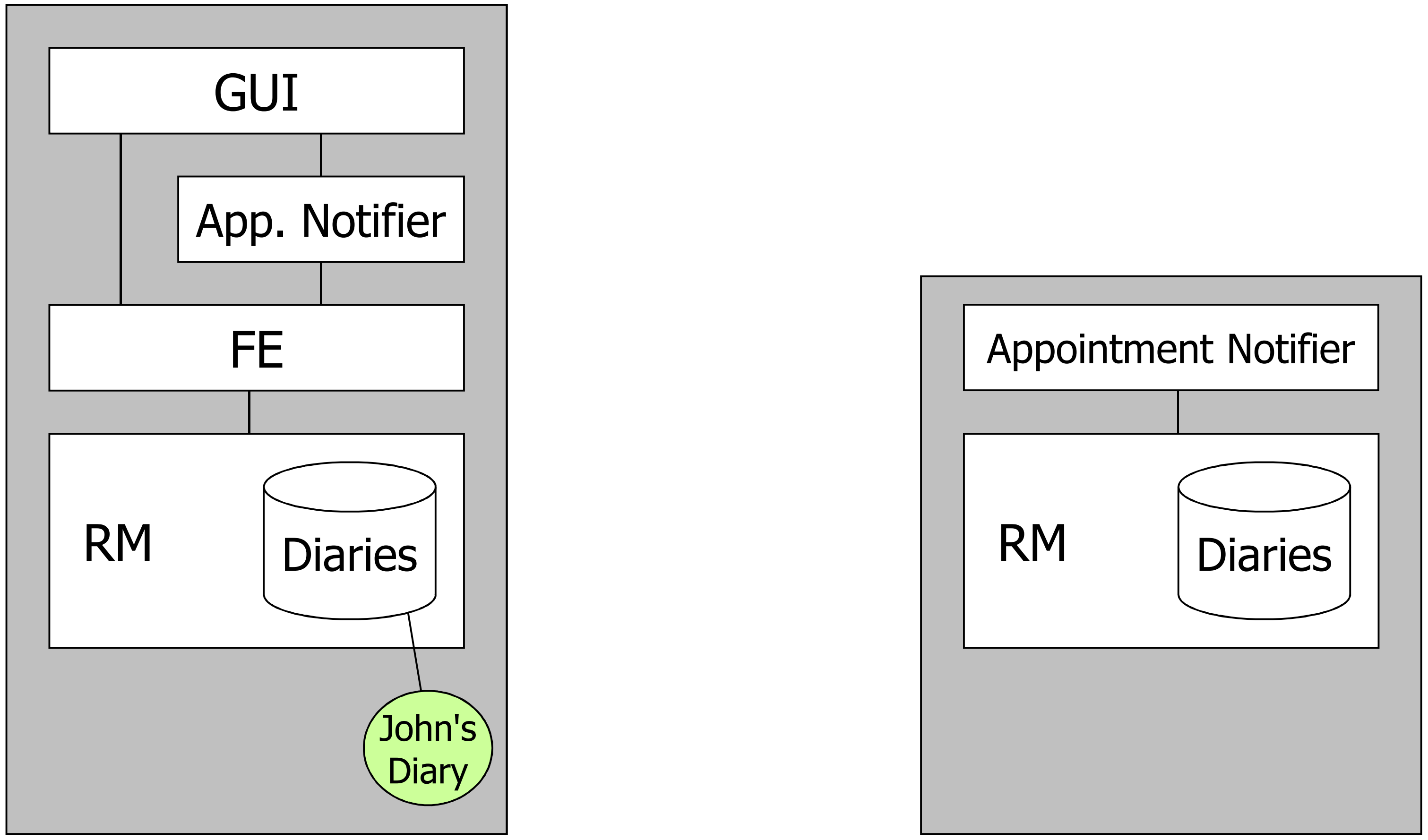
Example: Diary System



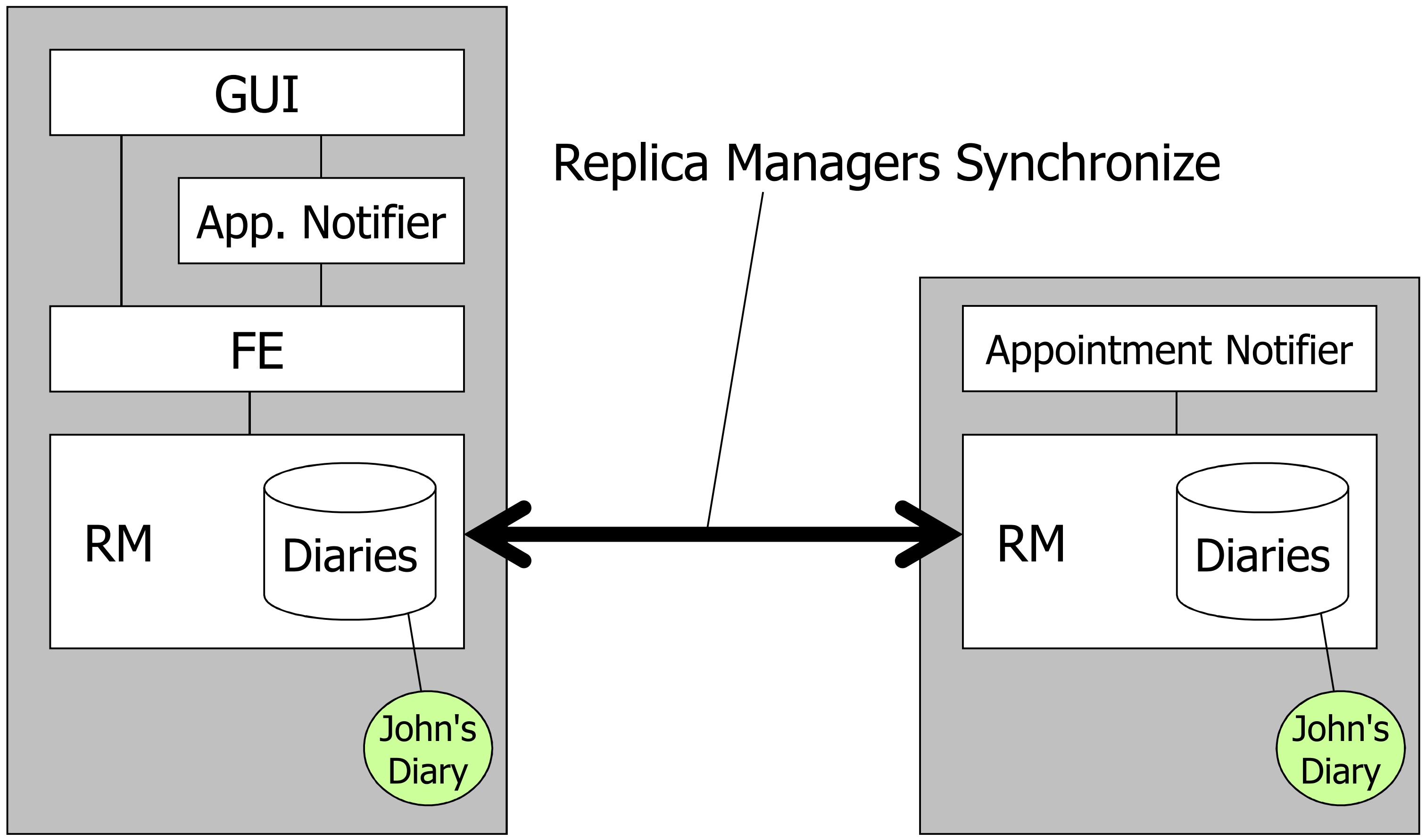
Central Server Instance Created

Example: Diary System

Initial Instance on “John's” Laptop

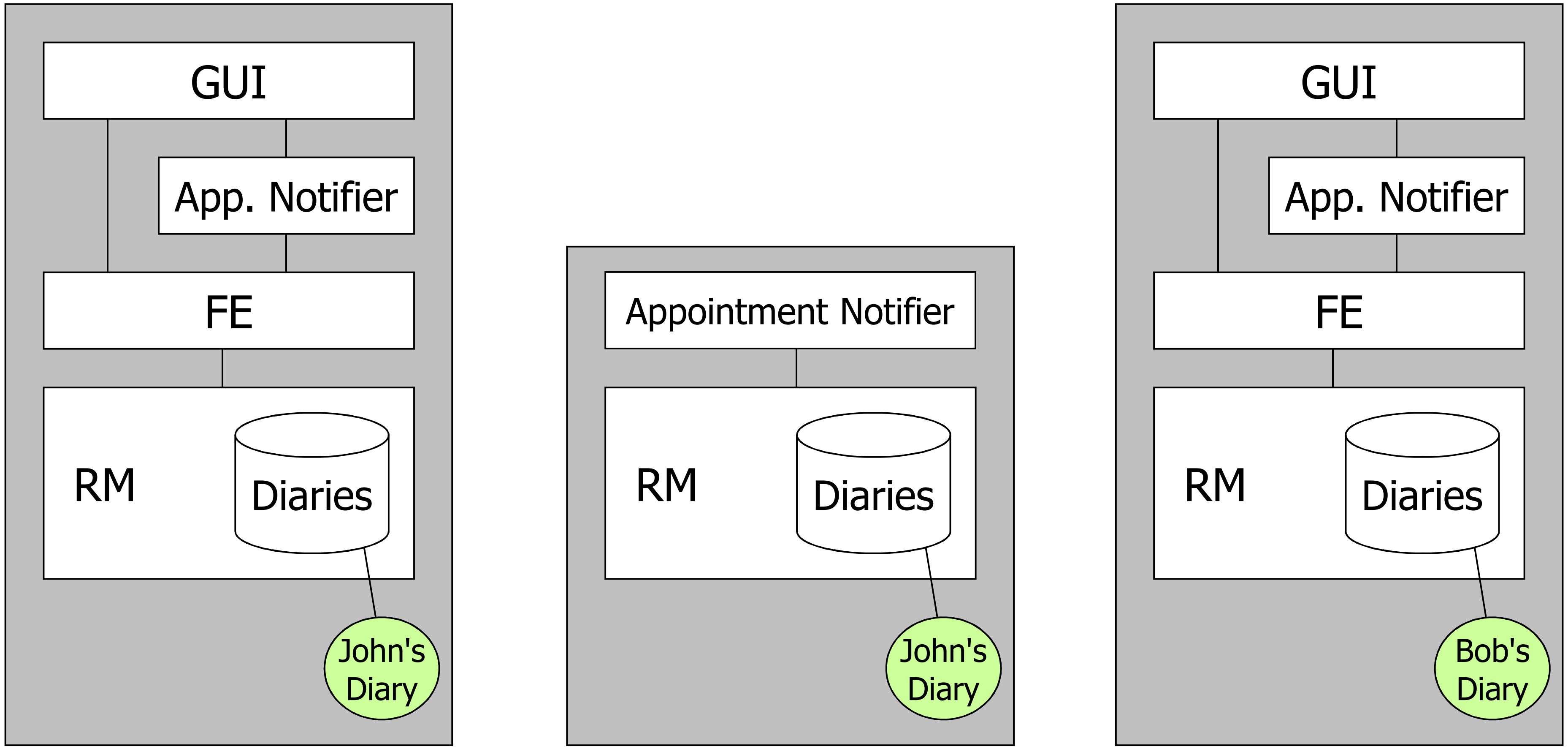


Example: Diary System

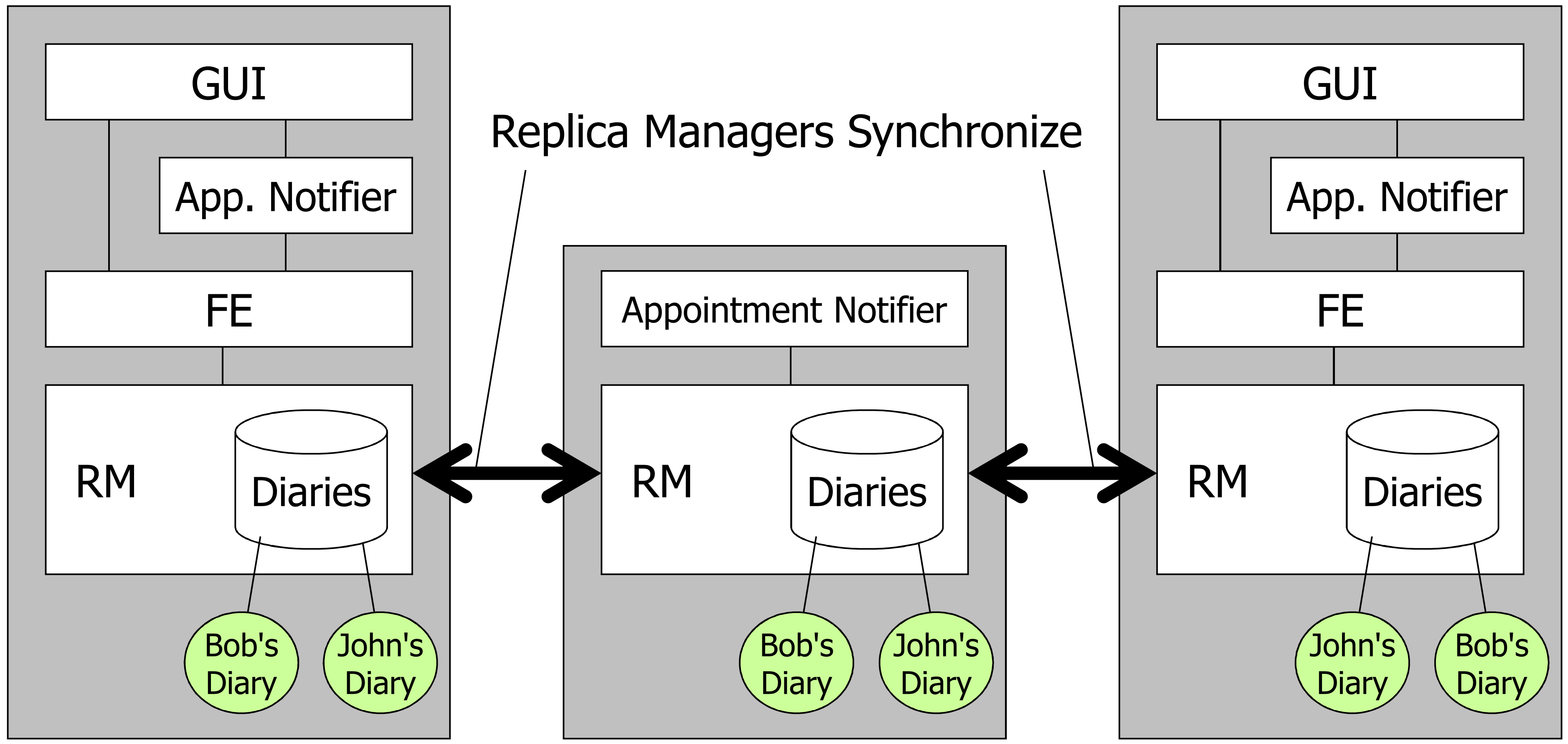


Example: Diary System

Second Instance on “Bobs” PC

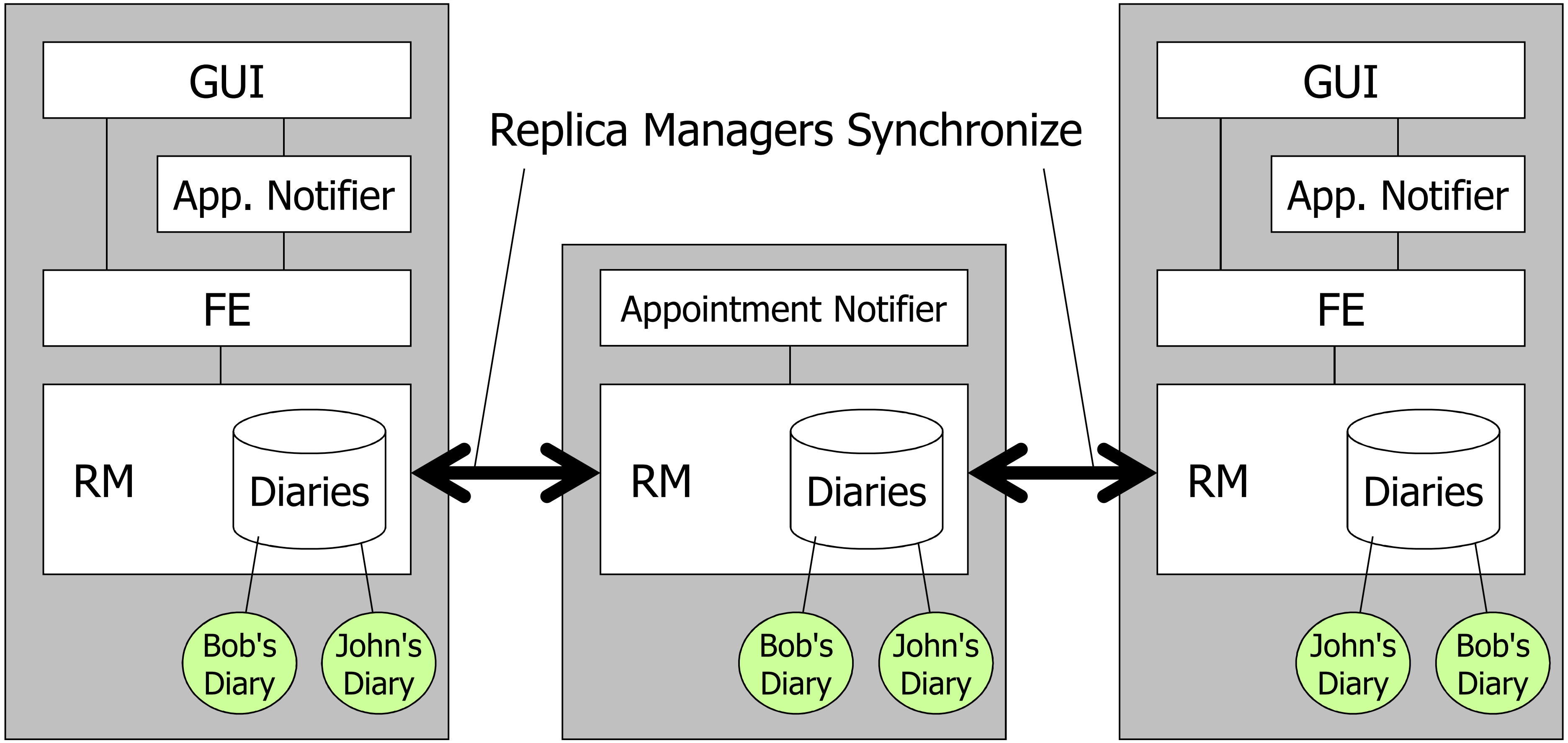


Example: Diary System



Two Questions...

- How do the RMs communicate?
- When and how does the synchronization/replication take place?



Next Lecture...Group Communication