Architecture of Distributed Systems

Replication 2018-2019

Part 2: Architectural issues

R. H. Mak

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

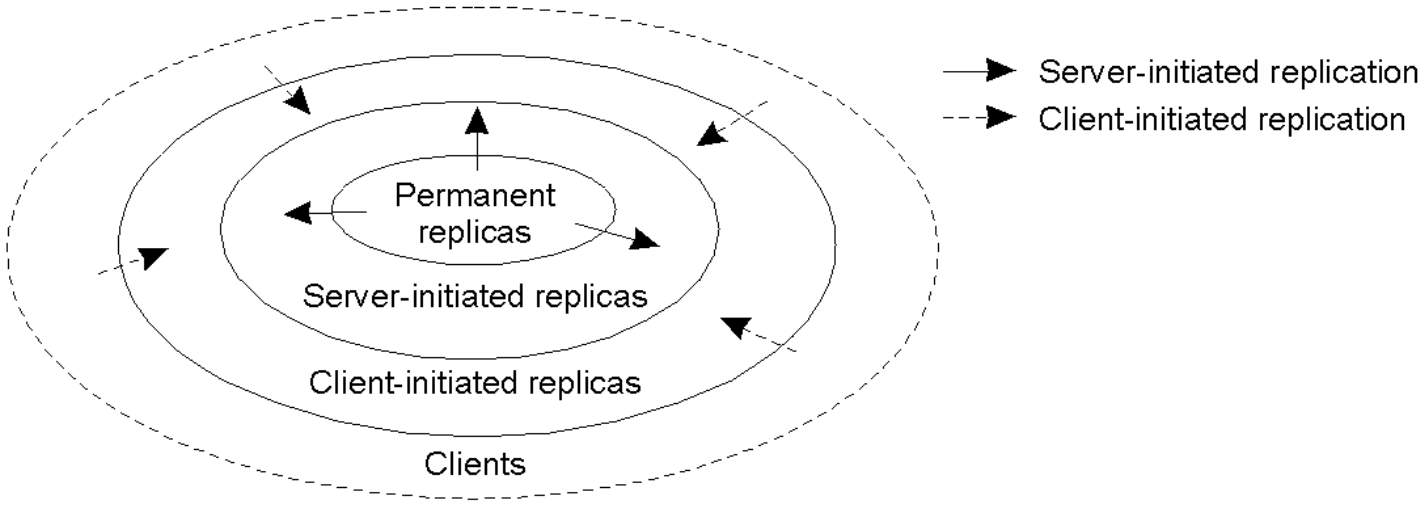
Agenda

- Introduction
- Consistency models
- Replica management
  - Placement strategies
  - Update protocols
  - Caching
  - Fault tolerance and recovery strategies
- Architectural case studies

Technische Universiteit Eindhoven University of Technology

Placement

- Server placement (the replica managers)
  - Based on geographical positions or internet topology
  - Important for data-owners, service providers
  - Organized in sites (data centers)
- Content placement (the replicas)
- Static versus dynamic



Technische Universiteit Eindhoven University of Technology

Replica placement

- Permanent replicas
  - Limited, fixed, statically determined number of replicas
  - Presumes a stable and a-priori known load distribution
  - Increased reliability and performance
- Server-initiated replicas
  - Dynamically changing number of replicas
  - To cope with server peak loads; increased availability
- Client-initiated replicas (aka client caches)
  - To improve response time by improving data proximity; to reduce network load
  - No persistent storage; data has limited life-time
  - Client is responsible for consistency; although the server may assist

Protocol classification

- What information is disseminated upon an update operation
  - The operation (active), the resulting state (passive), a notification
- Which party takes the initiative for dissemination upon an update operation
  - Push-based (aka server-based), pull-based (aka client-based)
- What communication primitives are used to disseminate an update operation
  - Unicast, multicast, gossiping
- When is completion of an operation reported to the client (i.e., the data store commits the operation)
  - Immediate on receipt, after atomic execution, quorum-based

Together these determine the supported consistency model

Pull versus Push Protocols

Issue	Push-based	Pull-based
State of server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

Leases can be used to get a mix of push and pull

- Updates are pushed only when the client has an outstanding lease
- Clients without lease either pull data or request a lease
- Server can influence size of state to be maintained by modifying lease time
- Client state at server-side limited to clients with outstanding lease

Protocol classes

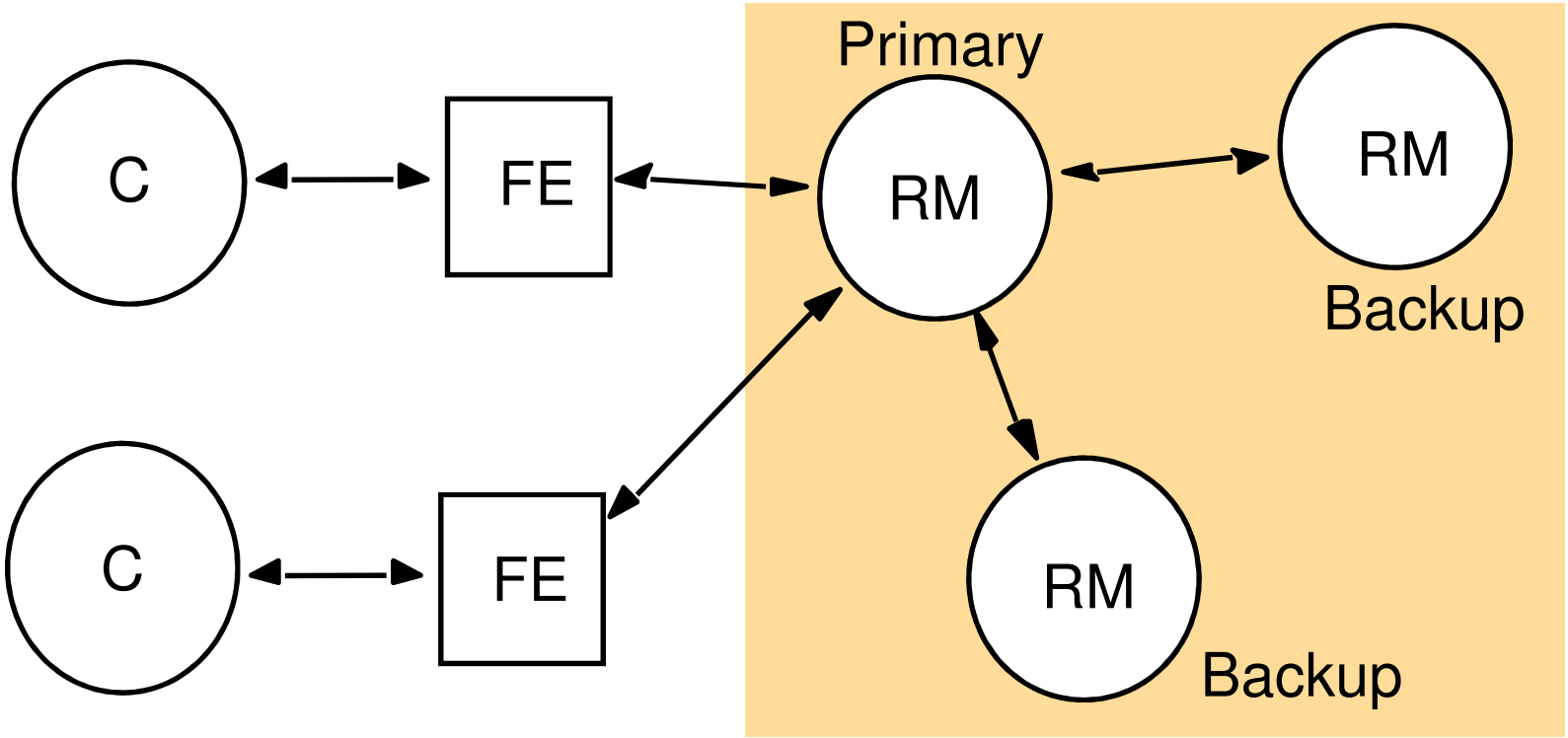
- Passive replication protocols
- Active replication protocols
- Gossip-based protocols

Technische Universiteit Eindhoven University of Technology

Passive (primary-based) replication protocols

- Primary-read, primary-write
  - seldom-used
  - supports linearizability; clients perceive atomic reads and writes.
  - blocking reads and writes, no concurrency, limited availability
- Local-read, primary-write (TvS remote-write)
  - supports sequential consistency
  - blocking writes, non-blocking concurrent reads
  - bounded staleness, at most one write missed
- Local-read, local-write (TvS local-write)
  - supports eventual consistency
  - no blocking operations, 100% availability (w.r.t. protocol not failures)
  - incomplete description in TvS
- All protocols are pushed-based w.r.t. to update values
  - local-write protocol pulls the “primary status”.

Figure 15.4
The passive (primary-backup) model for fault tolerance



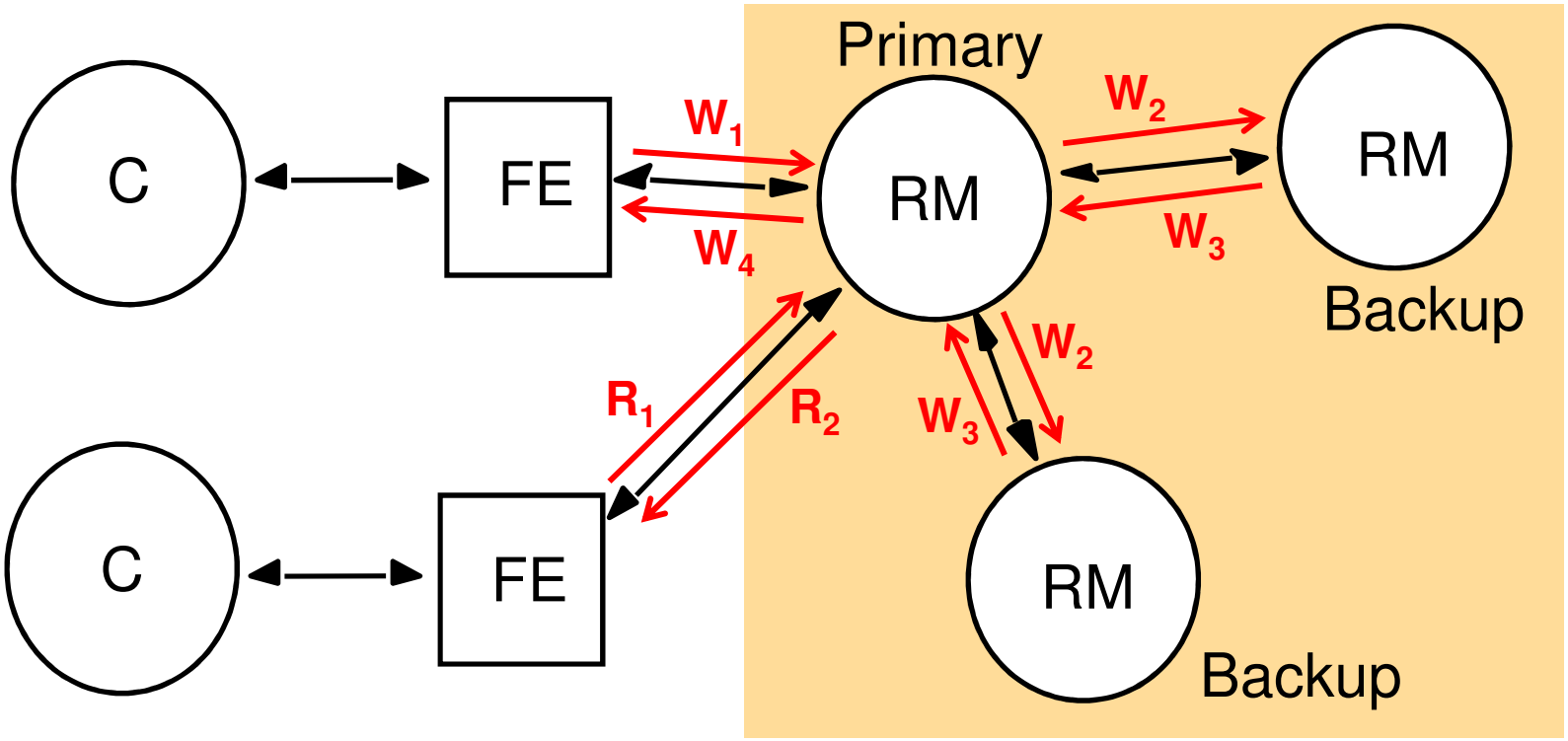
Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

Passive replication (executed in this order)

1. Request phase
   - The FE attaches a UID to the request and forwards it to the primary.
2. Coordination phase
   - The primary handles the request atomically and in order of receipt, using the UID to prevent executing an update more than once.
3. Execution phase
   - The primary executes the request and stores the response.
4. Agreement phase
   - In case of an update the primary sends the updated state, the UID and the response to the backups and waits for acknowledgements.
5. Response phase
   - The primary responds to the client's FE, which hands the response to the client.

CDK Figure 15.4 : including protocol

Primary-read, Primary-write version



Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

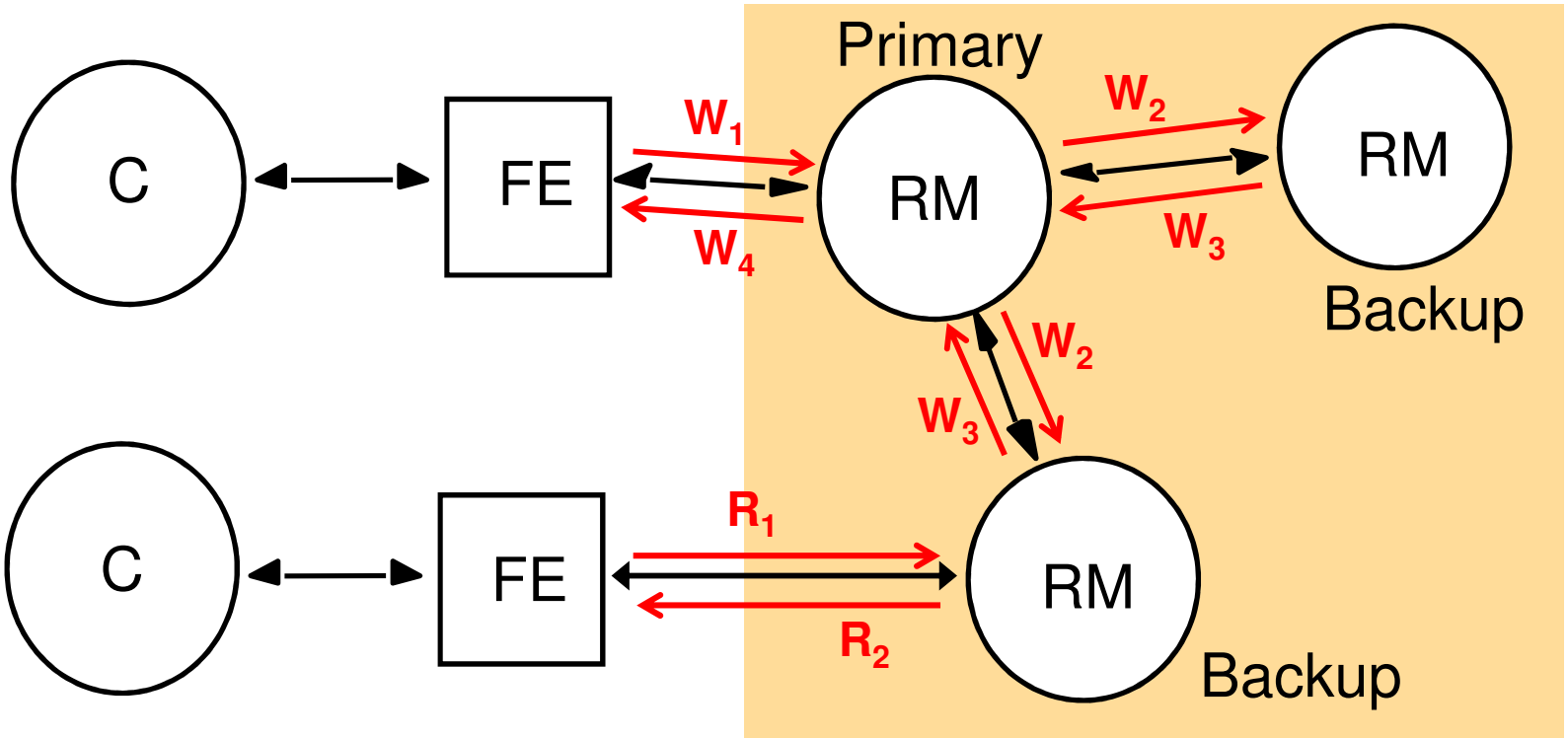
Recovery procedures

- Upon failure of the primary replica manager the system can retain linearizability when the primary is replaced by a unique backup
  - Determined by leader election algorithm or new group view
- All backups agree on the set of operations that have been performed at the moment when the replacement backup takes over.
  - This requires that writes are sent to the backups using atomic multicast, i.e., all forwarded writes are delivered in the same order to all backups.
  - FEs may use a discovery service to locate the primary

Technische Universiteit Eindhoven University of Technology

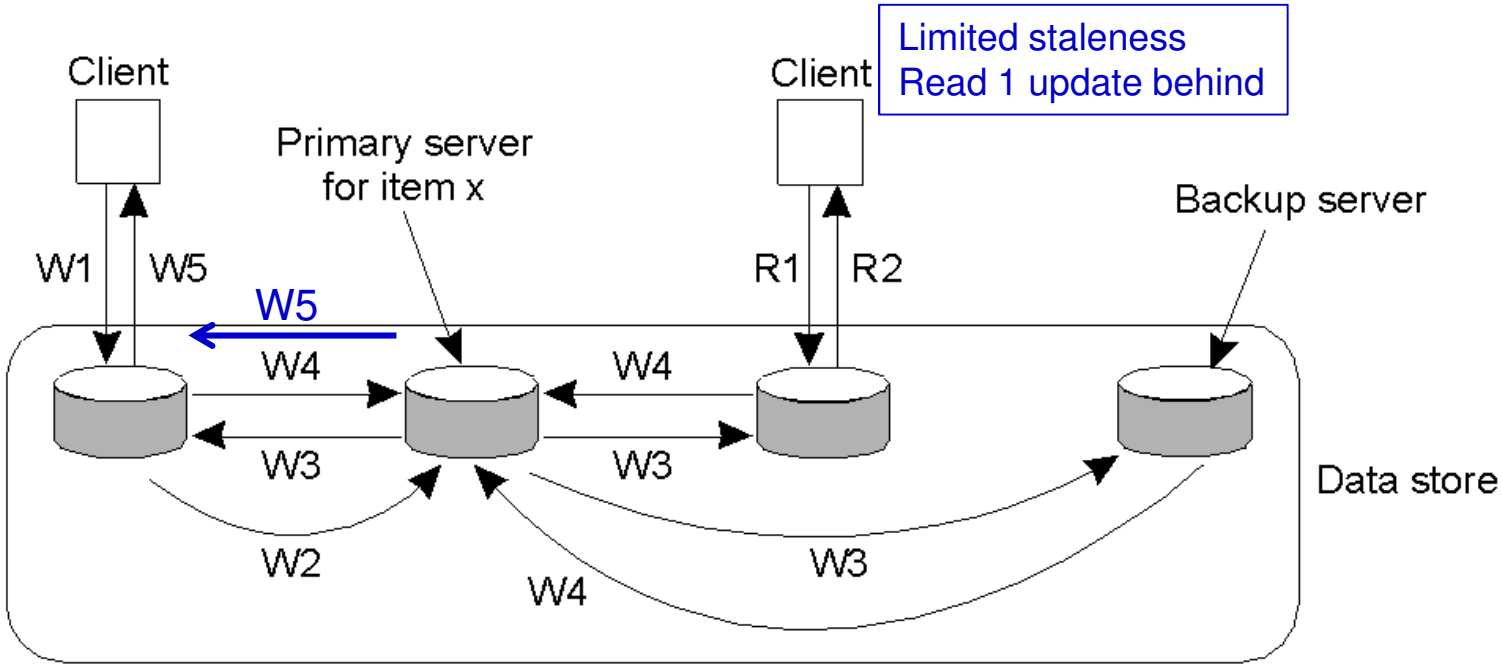
CDK Figure 15.4 : including protocol

Local-read, Primary-write version



Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

Local-read, primary-write: aka remote-write



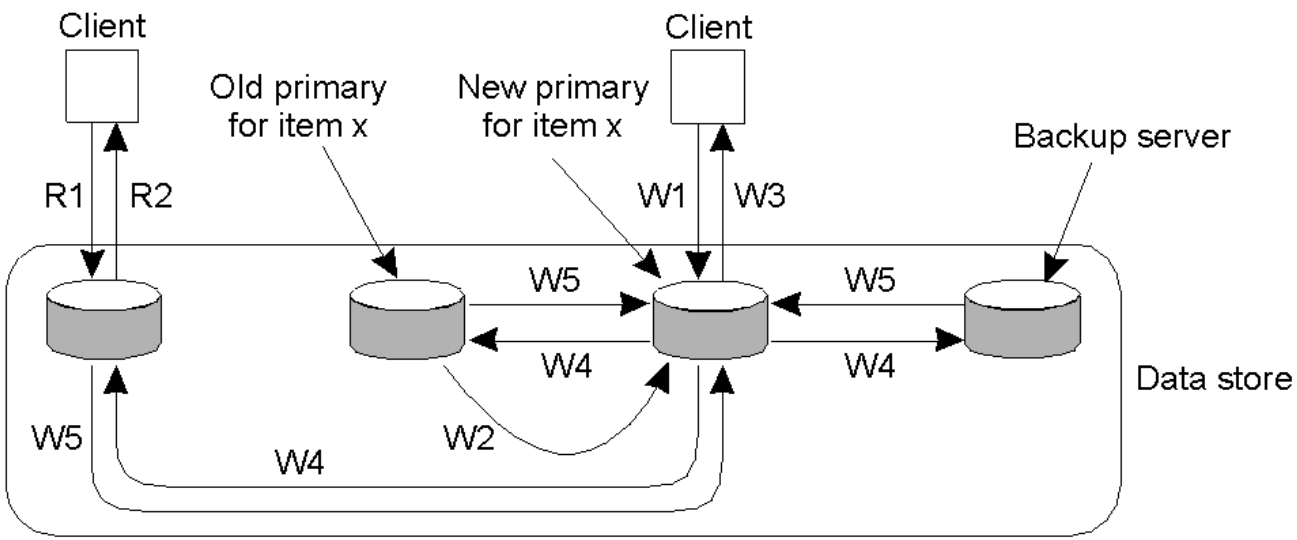
W1. Write request
W2. Forward request to primary
W3. Tell backups to update
W4. Acknowledge update
W5. Acknowledge write completed

R1. Read request
R2. Response to read

© Tanenbaum vanSteen

Local-read, local-write: aka local-write

What happens to a write request that arrives at a replica manager that has lost its primary status?

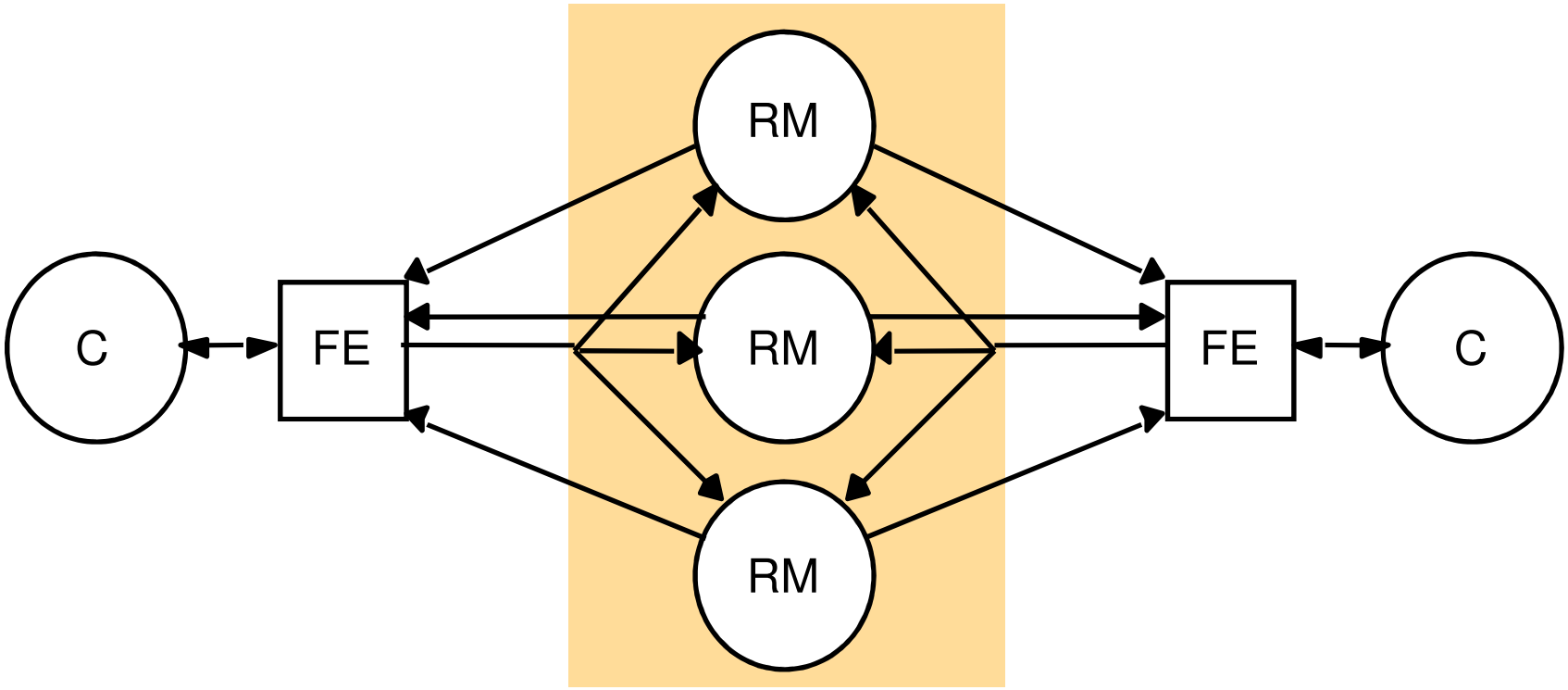


W1. Write request
W2. Move item x to new primary
W3. Acknowledge write completed
W4. Tell backups to update
W5. Acknowledge update

R1. Read request
R2. Response to read

Technische Universiteit Eindhoven University of Technology

CDK Figure 15.5
Active replication



- FE multicasts request to all RMs, which all behave the same (symmetry)
- Achieves sequential consistency, due to total order delivery of updates

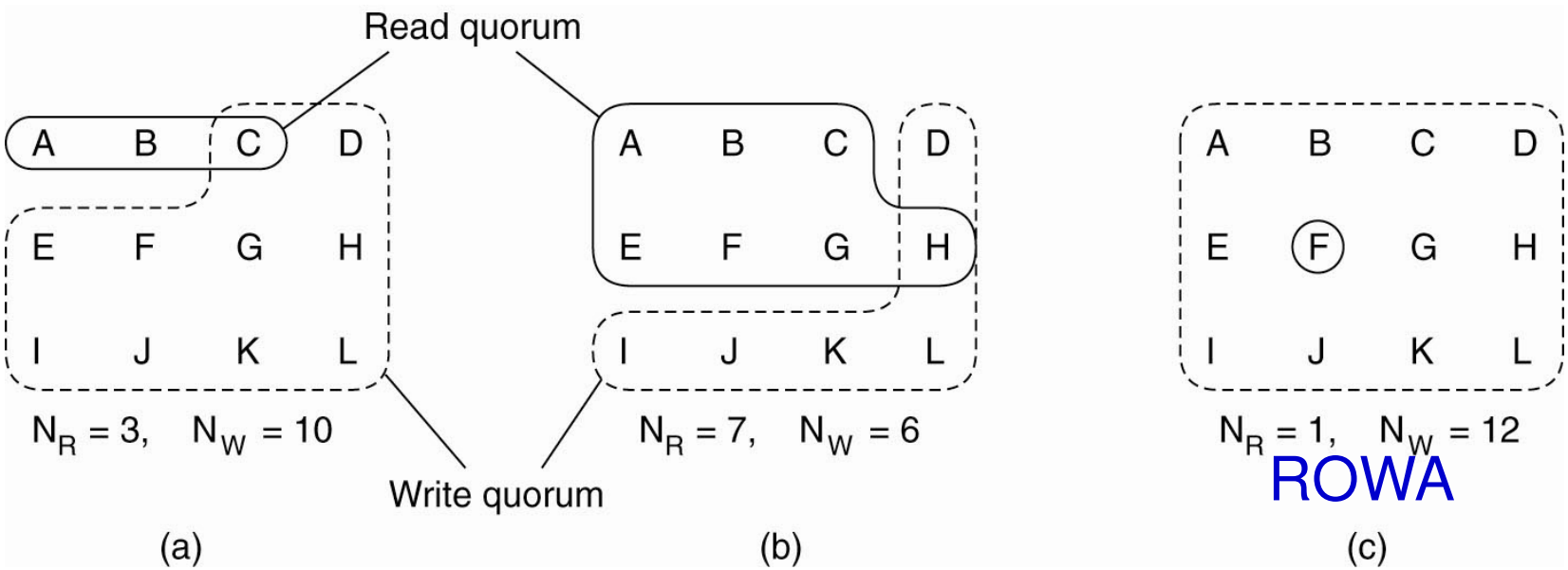
Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

Active replication

1. Request phase
   - FE attaches a UID to client operations and forwards it to all RMs using totally ordered reliable multicast. It does not issue a next request before it has received a response.
2. Coordination phase
   - Group communication delivers the request to every *correct* RM in the same total order.
3. Execution phase
   - Every RM executes the operation on its replica, resulting in the same local state and response.
4. Agreement phase absent
5. Response phase
   - RMs send their responses labeled with the request identifier to the FE which forwards it to the client after a certain number (depending on the desired fault resilience) responses have been received.

Quorum-based protocols

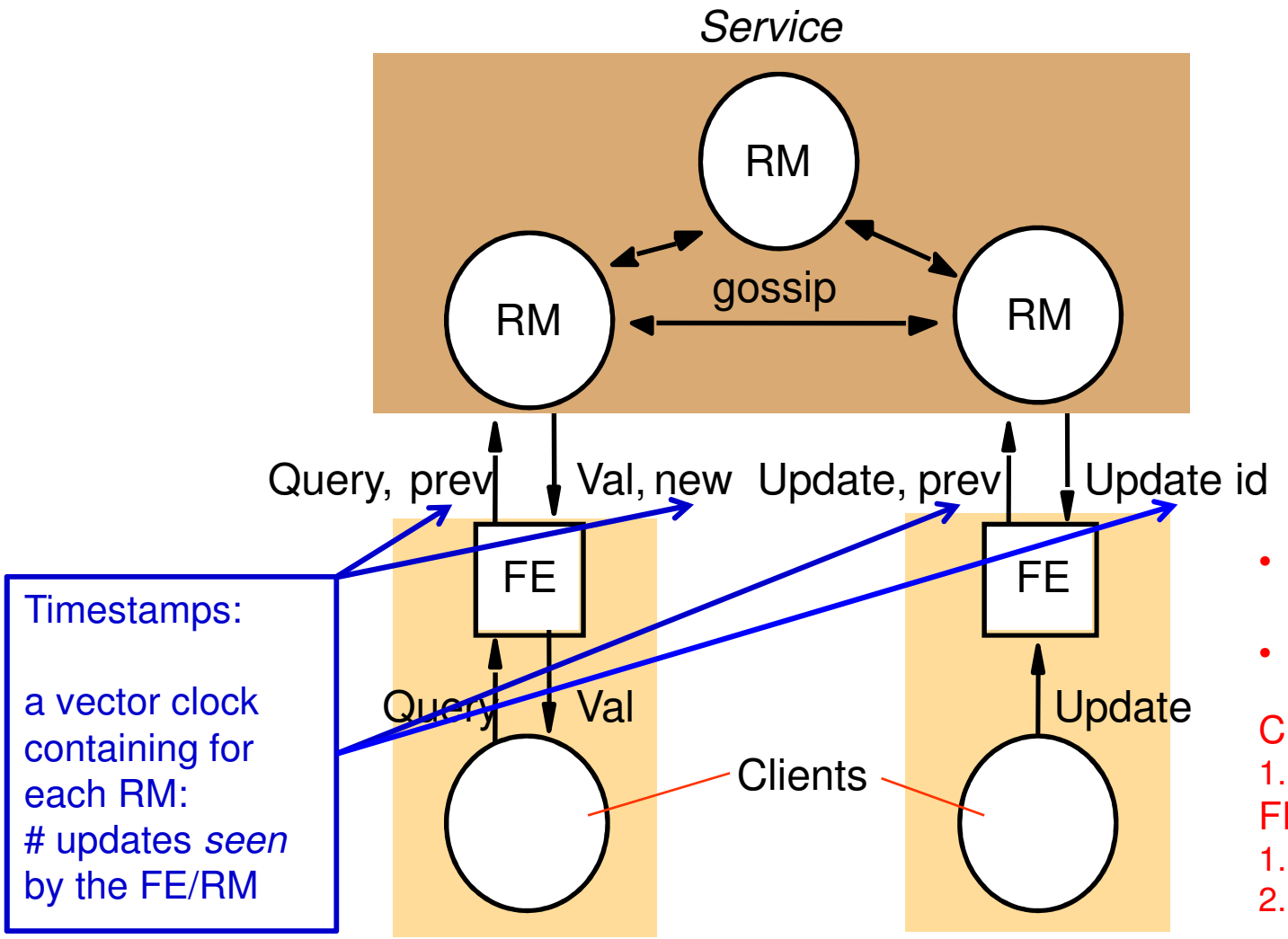
- Read operations need to inspect N_R replicas (the read quorum)
- Write operations need to update N_W replicas (the write quorum)
- With a total of N replicas
  - $N_R + N_W > N \ \wedge \ N_W > N/2$
- Replicas need a version number in order for the read to determine which replica contains the most recent value



Gossip architecture

- Aims at high availability
- Provides monotonic reads
- Customizable update delivery ordering
  - Usually causal ordering, because sequential consistency defies the goal of achieving high availability
- RMs exchange gossip messages on a periodic basis to disseminate update operations.
  - Per exchange a gossip partner chosen at random (anti-entropy)
  - Either push or pull or push-pull
- Resilient against network partitioning
  - Only one healthy RM in a partition is sufficient
- Scalability is an issue
  - # gossip messages, size of vector clocks

Figure 18.5
Query and update operations in a gossip service



- Updates should not read the state!
- Handled by FE

Client issues:

1. X := X+1

FE issues:

1. tmp := Read (X)
2. Write (X, tmp+1)

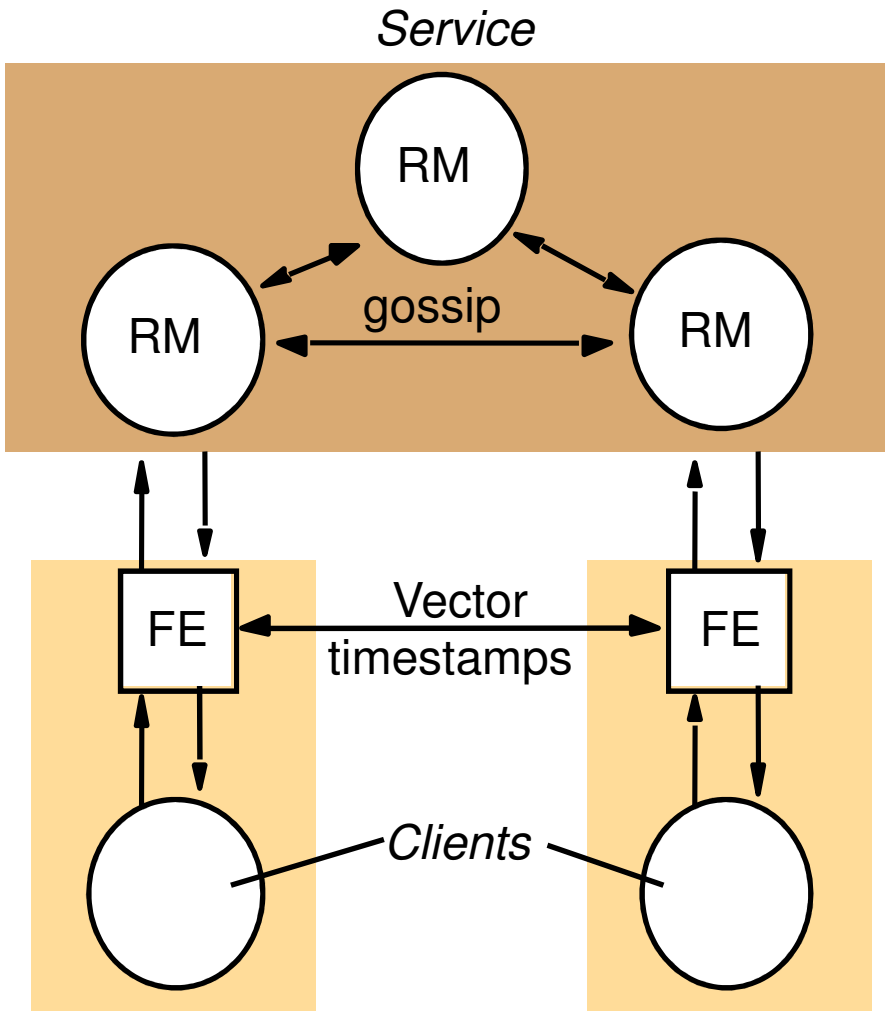
Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

Gossip architecture

1. Request phase
   - FE sends request to a single RM (variable per request)
   - FE blocks client on query, not on update (although it may)
2. Coordination phase
   - RM does not process a request until allowed by ordering constraints
   - Implies waiting for gossip messages from other RMs
3. Execution phase
   - Every RM executes the operation when due (has become stable)
   - Query results are returned to the FE
4. Agreement
   - Exchange of gossip messages between RMs
   - Lazy, after some updates or when knowing a peer that has the update
5. Response phase
   - FE returns outcome query to client

Technische Universiteit Eindhoven University of Technology

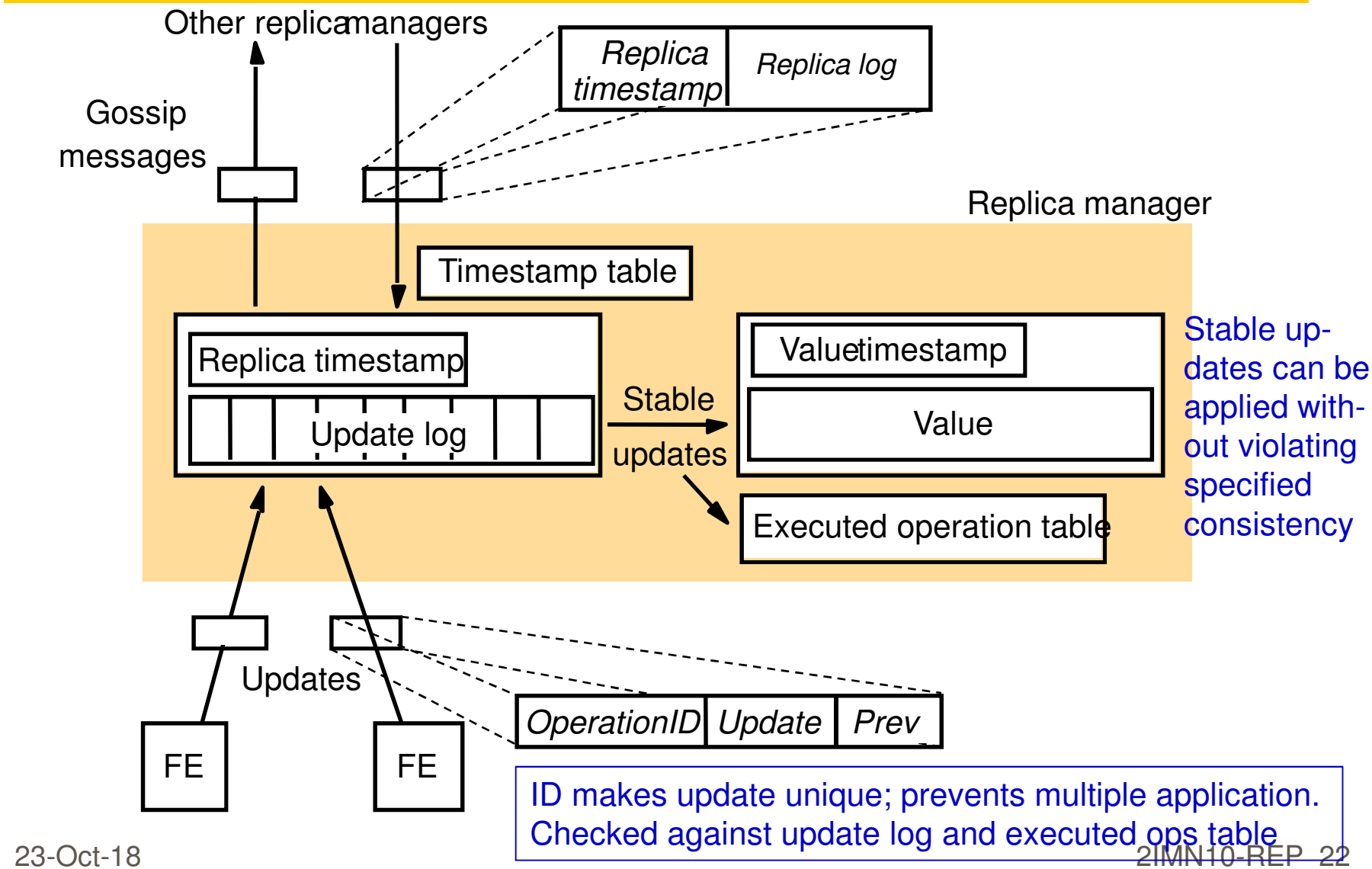
Figure 15.7
Front ends propagate their timestamps whenever clients communicate directly



ALL client-to-client messages also run via the FEs to ensure proper timestamps for causal ordering.

Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

Figure 15.8
A gossip replica manager, showing its main state components



Caching

- Caching is a specific form of data replication. Its
- distinguishing features are:
- Initiated and managed by clients
  - The server-side data store may assist by notifying the cache that its data has become stale
  - A single cache may be shared by several clients
- Introduced with the objective to decrease access time, i.e., reduce latency of client requests
  - although the system as a whole may also benefit by reduction of network traffic
- Applies to transient data
- Consistency is referred to as cache coherence

Coherence enforcement strategies

1. No caching of shared data
   - Provides only limited performance improvement
2. Server-initiated enforcement
   - Server sends invalidation notification
     - Next query pulls the fresh value from the master replica
   - Server sends update
3. Client-initiated enforcement
   - Write-through
     - Write operations to the cache are immediately forwarded to the master replica
     - Client needs to have exclusive write permission to guarantee sequential consistency
   - Write-back
     - Write operations are delayed, e.g., until the cache entry holding the replica is selected for eviction because the cache is full

Agenda

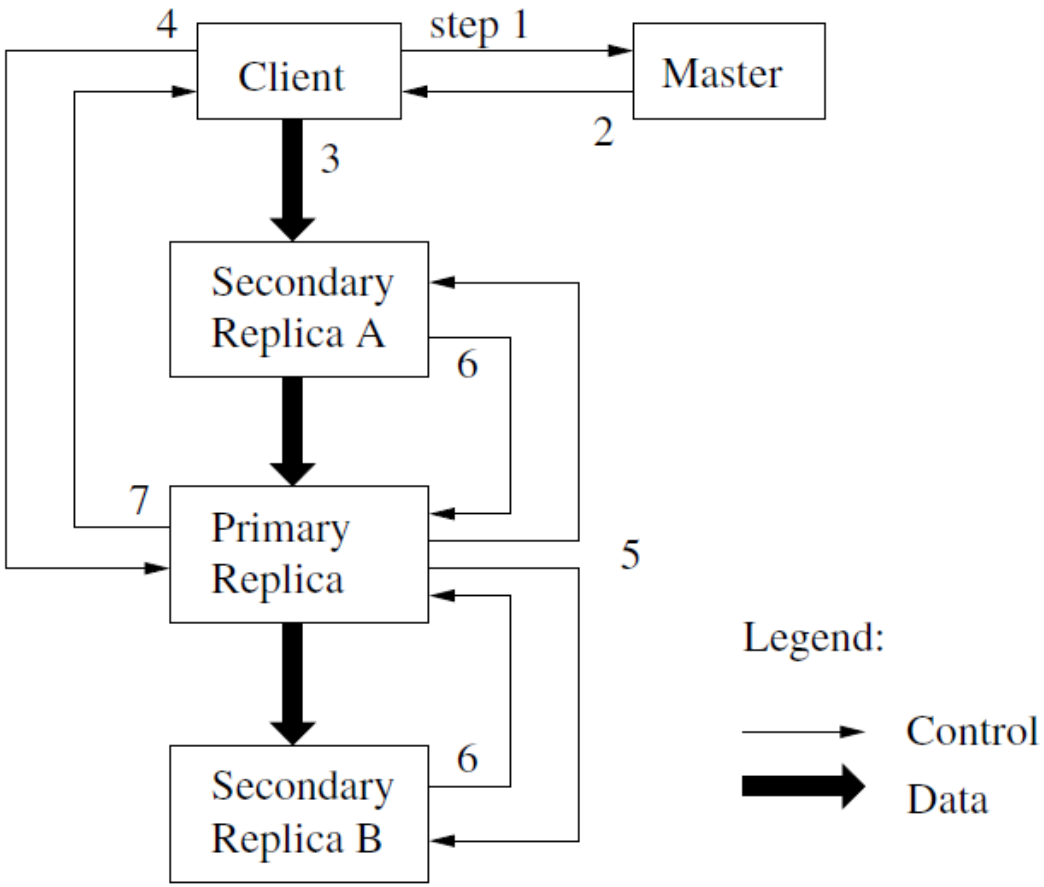
- Introduction
- Consistency models
- Replica management
- Architectural case studies
  - Google File System
  - Coda File System
  - Amazon Dynamo
  - Squirrel Web cache

Technische Universiteit Eindhoven University of Technology

Google File system

Conit	File chunk of 64KB
# Replicas	User defined (default 3)
API	Read, Append, Write (optimized for appends, not writes) Large operations are divided into series of chunk sized operations that are not considered transactions.
Placement	Chunk replicas are spread over different racks for optimal reliability and availability.
Read access	To the nearest replica (using central location service)
Update policy	Essentially primary-based. Data is pushed by clients to replicas using pipelining. Writes are committed by leased-based primary. Reads from any replica.
Consistency	Relaxed consistency. After successful updates all clients will see the same data. For concurrent updates, although all replicas are the same there may not be a sequential execution that would have produced the outcome. (undefined file regions are detected using chunk version numbers)
Fault tolerance	Crash resilient.

GFS write operation



Determination primary

- by lease (60 sec)
- renewal requests are piggy backed on keep alive messages

Technische Universiteit Eindhoven University of Technology

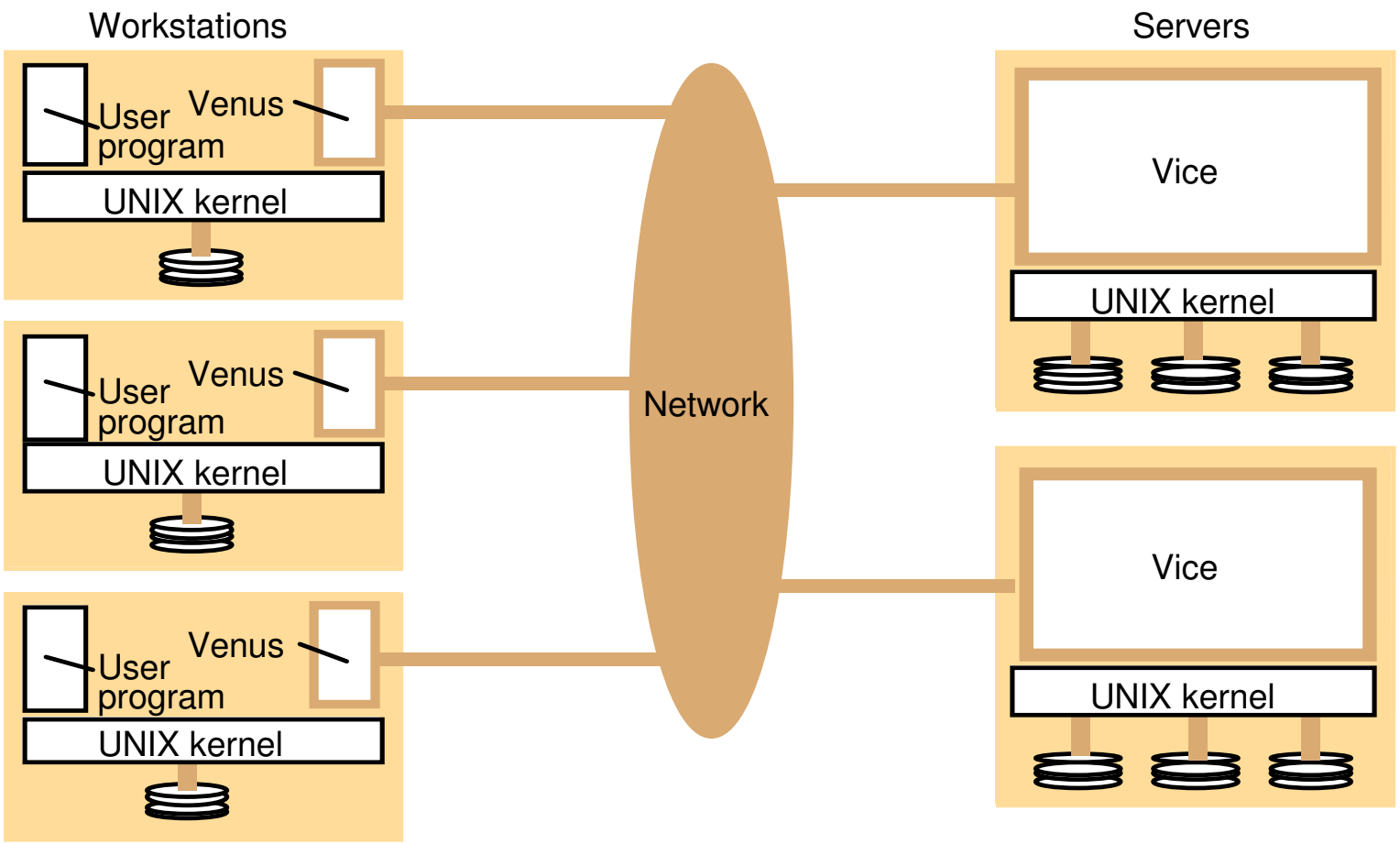
Coda

- Location transparent distributed file system
  - Descendant of the Andrew File System (AFS)
  - Caches volumes at the client side (on disk!)
  - Keeps more than one replica at the server side
- Key driver: constant availability
  - also increased resilience against failures
  - also improved scalability
- Mechanism
  - Server replication
  - Disconnected operation (using local cache)
  - Complete files are cached
- No distinction between voluntary disconnection and network or server failure

Coda file organization

- Each file and directory in shared file space has a UFID
  - All replicas have the same UFID
- Unit of replication is a volume (set of files)
  - UFIDs include volume identification
- Volume Storage Group (VSG) of a volume
  - The set of servers that have a replica of the volume
  - Each file replica has a Coda Version Vector (CVV)
- Accessible volume storage group (AVSG)
  - Subset of the corresponding VSG
  - Maintained by clients for each volume from which it has cached files
  - Each client has a preferred server per AVSG
  - When a workstation is disconnected, AVSG is empty.

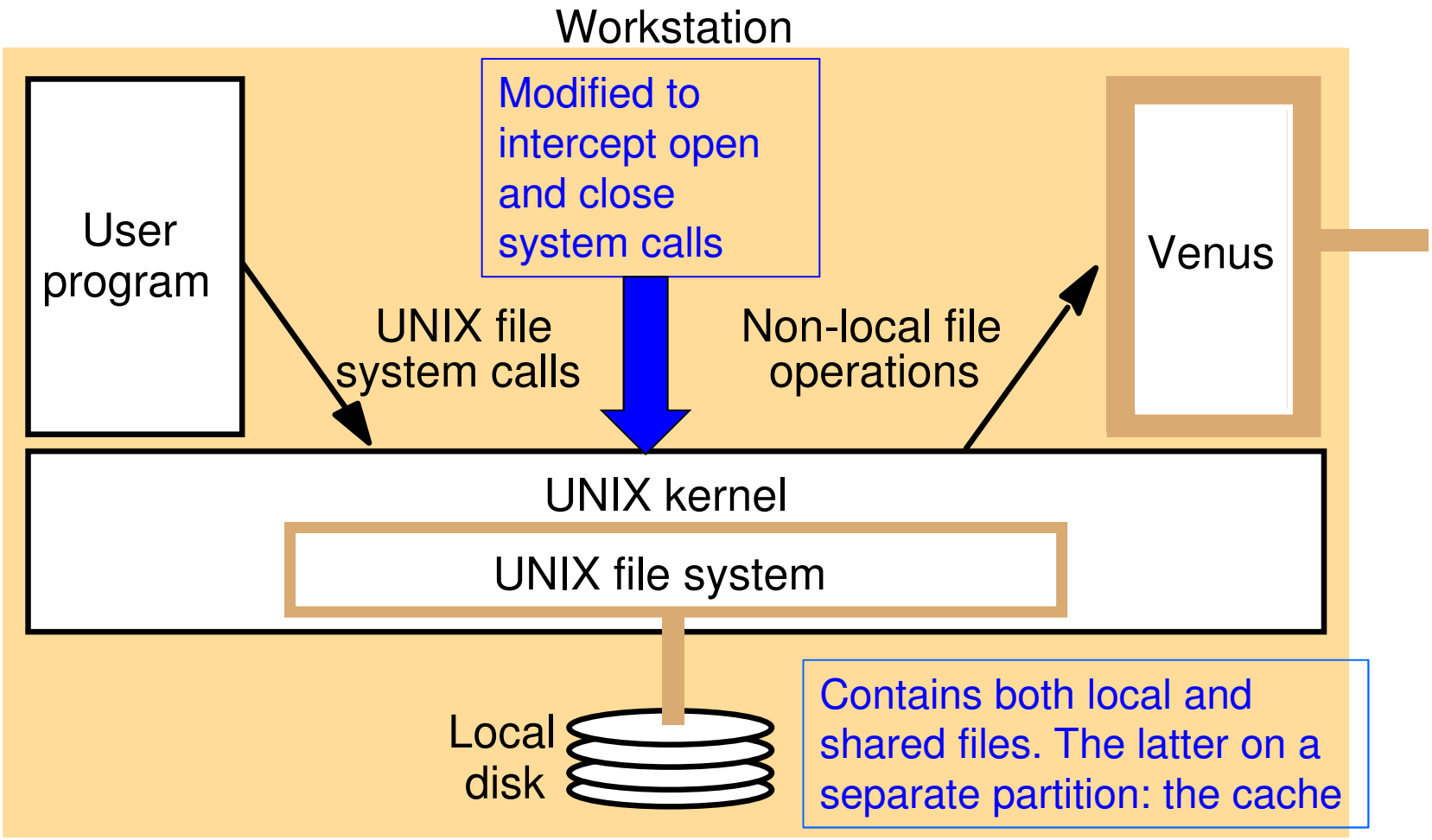
Figure 12.11
Distribution of processes in the Andrew File System



AFS: 1 server (custodian) per shared file;
CODA: multiple servers (VSG) per shared file

Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Sstems: Concepts and Design Edn. 5
© Pearson Education 2012

Figure 12.13
System call interception in AFS

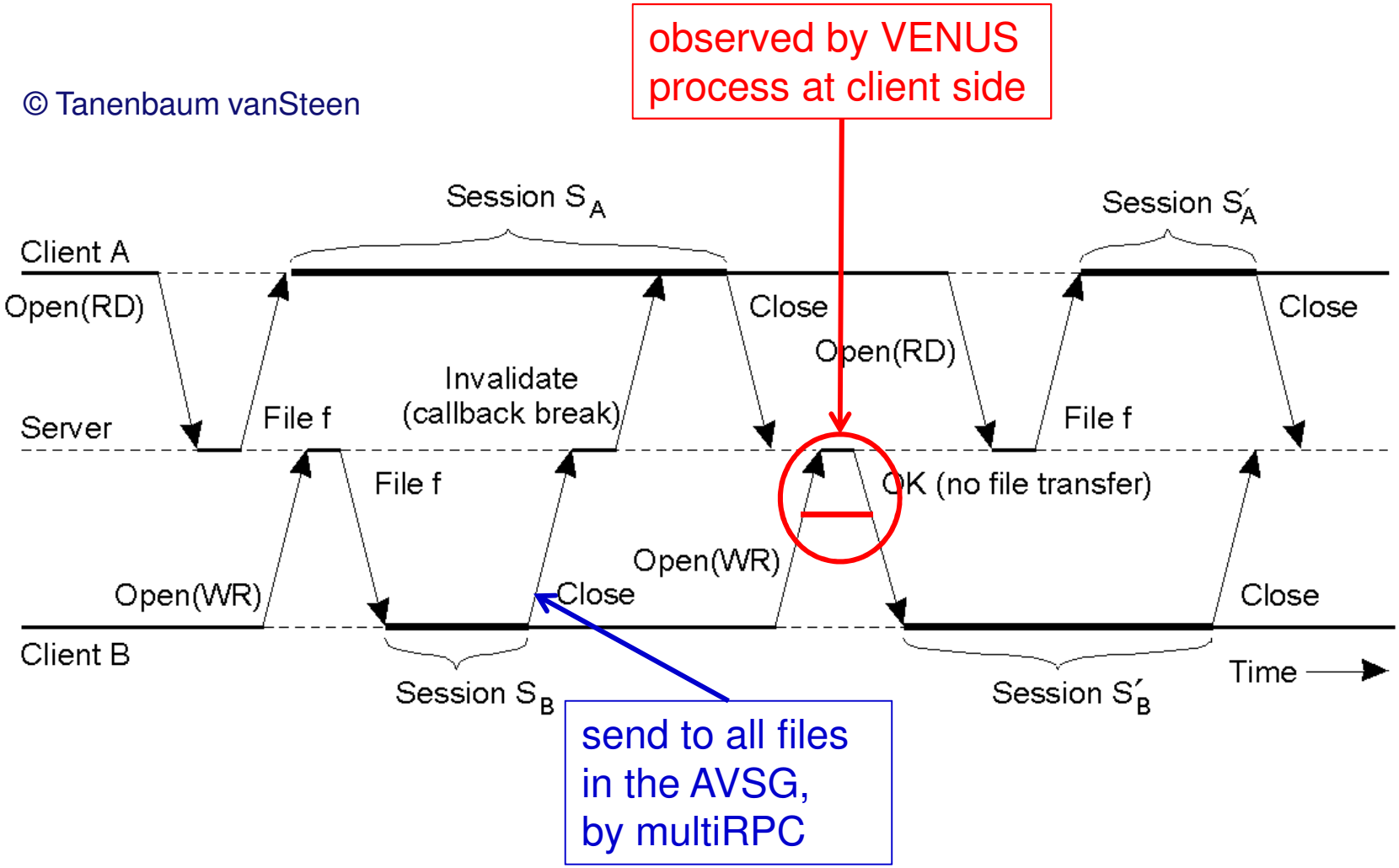


Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

Callback mechanism

- When a file f is copied from Vice to Venus
  - A callback is registered at Vice servers (state-full server!)
  - A valid callback promise for f is set at Venus client
    - Promises are either valid or cancelled (cf. staleness bit)
  - When a Vice server becomes aware that another client has modified f it sends an invalidation message (*break*) to all clients that hold a promise changing the state to cancelled
    - So servers maintain state for each client
  - Callback promises are checked when a file is opened
    - Opening a file for the first time or opening a file for which a callback break has been received results in downloading a fresh copy of the file
  - Breaks are sent as a result of files being closed
  - Callbacks are leases; valid for a period of time after which they have to be renewed.

Sharing Files in Coda



Technische Universiteit Eindhoven University of Technology

Consistency

- Variant of ROWA
- On “open file” and cache miss or stale copy.
  - the server holding the most recent copy is made preferred server, callback is established and a copy is fetched
- On “close file” after modification the new file is transferred in parallel to AVSG.
- Venus regularly polls status AVSG both for loss of members or for newly available members.
  - On loss of preferred server callback promise is dropped
- Inconsistencies, arising from partitioning, are resolved through Coda Version Vectors (vector clocks) or user intervention.
- Further details, see Coda paper and TvS (2nd ed, chp 11).

Amazon Dynamo

- Dynamo is a key-value store
- Availability is the key driver
  - achieved at the cost of reduced consistency
- Number of replicas N
- Placement strategy:
  - on the first N healthy nodes of a preference list , containing > N physical nodes, spread across various data centers.
  - physical nodes host virtual nodes (tokens) organized as a DHT.
  - in principle, the replicas are on N consecutive virtual nodes starting at the one indicated by the hash of the object key using consistent hashing.
  - zero-hop DHT.
    - i.e. enough routing information to reach the destination in 1 step

Technische Universiteit Eindhoven University of Technology

Amazon Dynamo (cntd)

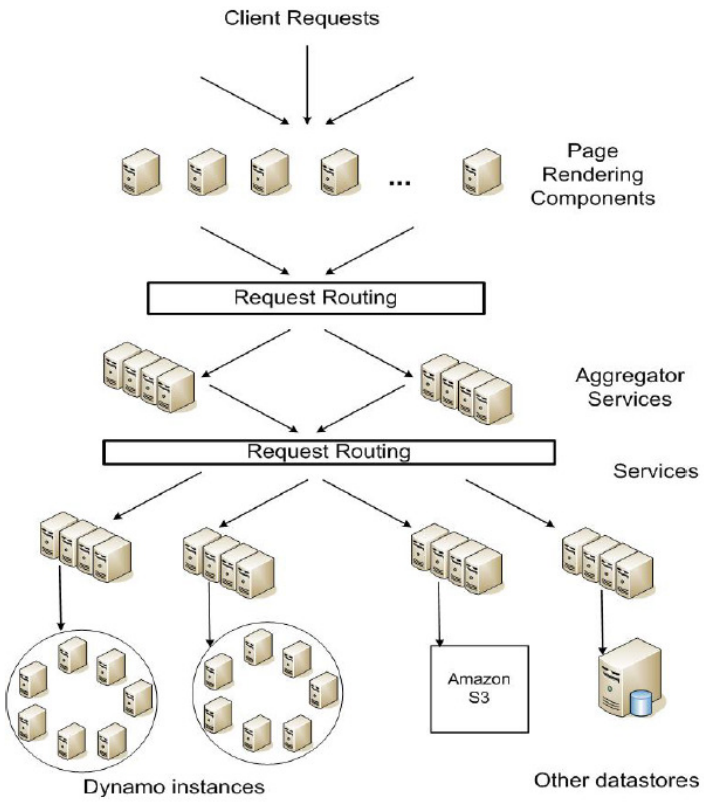


Figure 1: Service-oriented architecture of Amazon's platform

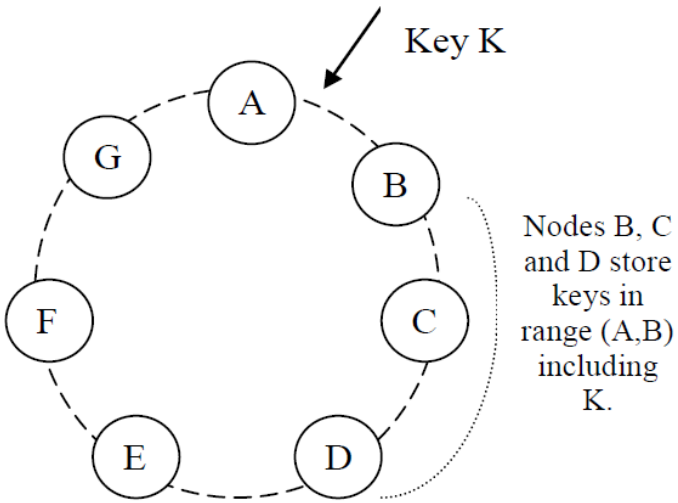


Figure 2: Partitioning and replication of keys in Dynamo ring.

Taken from: DeCandia et al, Dynamo: Amazon's Highly Available Key-value Store, SOSP 07, ACM, 2007

Technische Universiteit Eindhoven University of Technology

Amazon Dynamo (cntd)

- Very simple API
  - Get (key)
    - returns list of (object, context) pairs (usually 1 pair)
  - Put (key, context, object)
    - Writes an immutable new version. Older version are subsumed.
    - Due to concurrency and failures conflicting versions may result
- Access
  - Via coordinator node, usually the one determined by hashing the key. Using R+W quorums on the first N healthy nodes
    - Typically $(N, W, R) = (3,2,2)$
  - Coordinators generate vector clocks for versioning
  - Requests are blocked until the quorum is reached
    - Taking W=1 avoids blocking writes

Technische Universiteit Eindhoven University of Technology

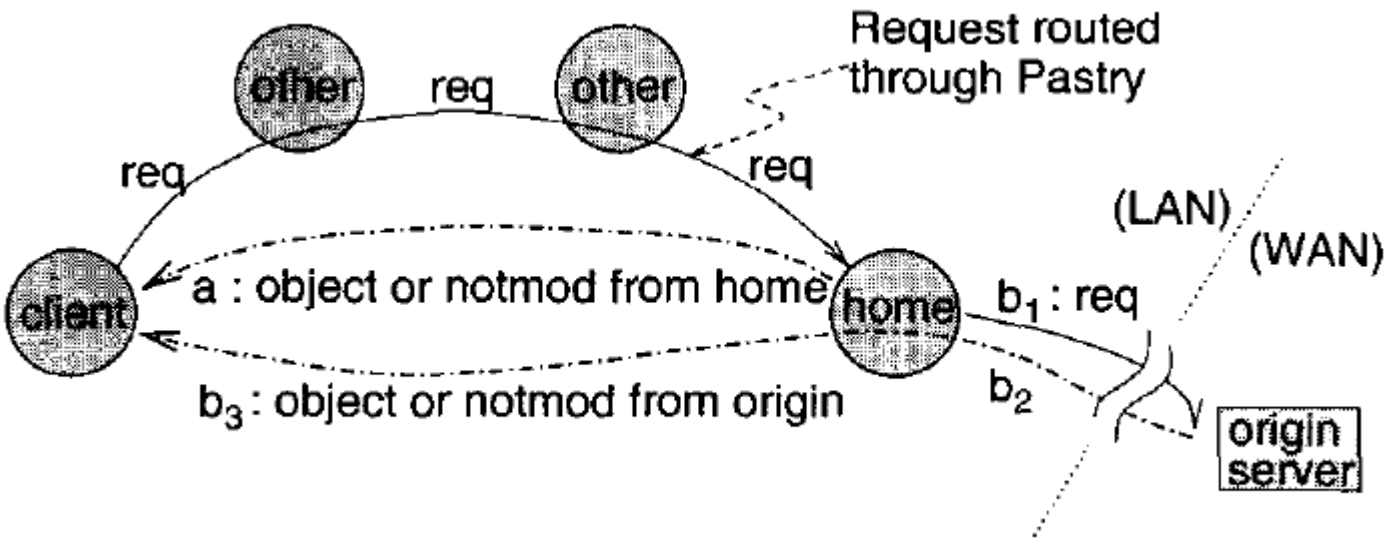
Amazon Dynamo

- Eventual consistency, sloppy quorum + some add-ons
  - Inconsistent values are stored side-by-side
    - Updates are time-stamped using a vector clock; updates arriving at a replica that are concurrent according to the vector clock are both stored
    - Vector clocks allow detection of causal order
    - Clients need to resolve remaining inconsistency; applications are such that customer's are willing to do this, e.g., cleaning up their shopping cart that contains too many items, because deletes have not been seen.
    - Only 0.06% inconsistency
  - Committed values cannot be lost (durability), for this a (sloppy) quorum-based approach is used.
    - Nodes other than the first N of the preference list may assist to reach a quorum
- Fault tolerance (details see paper)
  - Transient node failures are covered by hinted handoff.
    - Values temporarily stored elsewhere are returned their home nodes, upon recovery of those. (original home is provided as hint)
  - Permanent failures are covered by replica synchronization

Squirrel web cache

- Local caches in web browsers of clients in a single LAN are used to implement a distributed LAN-based web-server proxy.
  - Can handle corporate networks from 100-100,000 nodes or more.
  - Alternative for (a cluster of) web-proxy servers
- Uses a peer-to-peer architecture (overlay) on the LAN
  - Built on top of Pastry, but any P2P routing overlay will do.
  - Inserts a Squirrel proxy between the browser and the local web cache.
- Scales horizontally and automatically
  - As the number of browsers increases so does the cache capacity, whereas cache response time stays almost constant.
  - No administrative effort required.
- Makes use of HTTP headers to implement a conditional Get
  - If-Modified-Since <timestamp>
  - If-None-Match <ETag>

Home-store scheme



Items are stored in caches both at the client and at the home node

Each Web-resource has a home node on the LAN

- determined by a hash of its URL

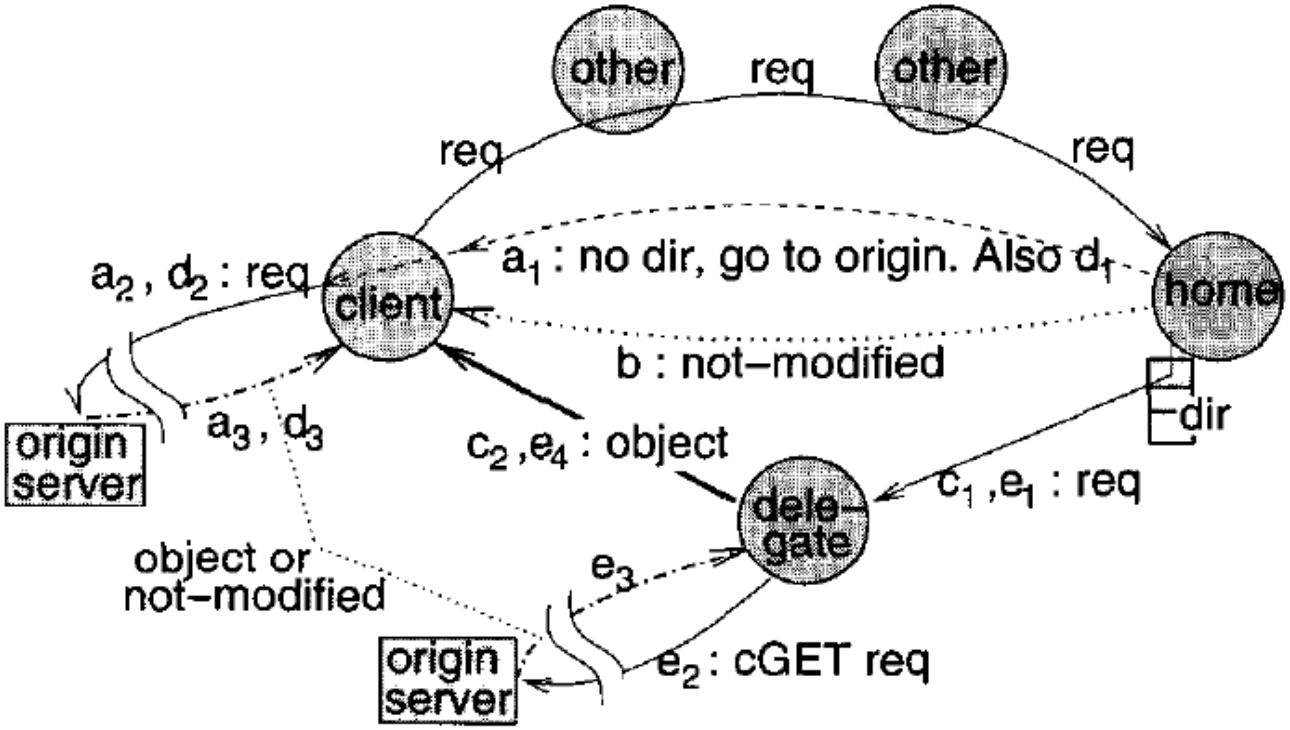
Seeking access to a cacheable object into its home node may result in

1. Miss (issue a GET and return object from origin))
2. Hit and fresh (return from home)
3. Hit and stale (issue a cGET and get an object or notmod in return)

Directory scheme

The home node contains a (small) directory of delegates that

- are local nodes that have recently accessed the object (resource)
- are expected to have a replica of the object in their cache
- share the same version (ETag) of the object (so, all fresh, or all stale)



Literature

- For the preparation of this slide set we have used material from various sources.
- Tanenbaum, van Steen: *Distributed Systems: Principles and Paradigms,*
  - Chapter 7, Sec 11.5.3, 11.6.1, 11.6.2.
- Coulouris, Dollimore, Kindberg, *Distributed Systems: Concepts and Design,*
  - Chapter 18 (basic model + gossip architecture)
- M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, *Understanding Replication in Databases and Distributed Systems*,
  - basic model
- R. Ladin, B Liskov, L. Shrira, and S. Ghemawat

  Providing high availability using lazy replication
  - gossiping

Technische Universiteit Eindhoven University of Technology

Literature

- Sitaram Iyer, Antony Rowstron, and Peter Druschel, *Squirrel: A decentralized peer-to-peer web cache*, *PODC 2002.*

- Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere, *Coda: A Highly Available File System for a Distributed Workstation Environment* , IEEE Trans on Comp , Vol 39, Issue 4,1990.

- Giuseppe DeCandia et. al., *Dynamo: Amazon's Highly Available Key-value Store*, *SOSP 2007.*

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, *The Google File System*, *SOSP 2003*