Distributed Systems

Lesson 7
Synchronization, Consistency & Replication

University of New York in Tirana
Master of Science in Computer Science
Prof. Dr. Marenglen Biba

Lesson 7

Synchronization

- It is important that multiple processes do not simultaneously access a shared resource, such as printer, but instead cooperate in granting each other temporary exclusive access.

- Moreover multiple processes may sometimes need to agree on the ordering of events, such as whether message *m1* from process *P* was sent before or after message *m2* from process Q.

Synchronization in distributed systems

- While communication is important, it is not everything.
  - Closely related is how processes cooperate and synchronize with one another.

- Synchronization in distributed systems is often much more difficult compared to synchronization in uniprocessor or multiprocessor systems.

CLOCK SYNCHRONIZATION

- Physical clocks
- Logical clocks
- Vector clocks

The Clock problem

- In a centralized system, time is unambiguous.
  - When a process wants to know the time, it makes a system call and the kernel tells it.
- If process *A* asks for the time, and then a little later process *B* asks for the time, the value that *B* gets will be higher than (or possibly equal to) the value *A* got.
  - It will certainly not be lower.
- In a distributed system, achieving agreement on time is not trivial.
- Is it possible to synchronize all the clocks in a distributed system?
  - The answer is surprisingly complicated.

Computer’s clock

- All computers have a circuit for keeping track of time. Despite the widespread use of the word "clock" to refer to these devices, they are not actually clocks in the usual sense.
- A computer timer is usually a precisely machined quartz crystal.
- When kept under tension, quartz crystals oscillate at a well-defined frequency that depends on the kind of crystal, how it is cut, and the amount of tension.
- Associated with each crystal are two registers, a **counter** and a **holding register**.
- Each oscillation of the crystal decrements the counter by one. When the counter gets to zero, an interrupt is generated and the counter is reloaded from the holding register.
- In this way, it is possible to program a timer to generate an interrupt 60 times a second, or at any other desired frequency.
- Each interrupt is called one clock tick.

Clocks on different machines

- With a single computer and a single clock, it does not matter much if this clock is off by a small amount.
  - Since all processes on the machine use the same clock, they will still be internally consistent.

- As soon as multiple CPUs are introduced, each with its own clock, the situation changes radically.
  - Although the frequency at which a crystal oscillator runs is usually fairly stable, it is impossible to guarantee that the crystals in different computers all run at exactly the same frequency.

- In practice, when a system has *n* computers, all n crystals will run at slightly different rates, causing the (software) clocks gradually to get out of synch and give different values when read out.
  - This difference in time values is called clock skew.

External Physical Clocks

- In some systems (e.g., real-time systems), the actual clock time is important.

- Under these circumstances, external physical clocks are needed.

- For reasons of efficiency and redundancy, multiple physical clocks are generally considered desirable, which yields two problems:

  (1) How do we synchronize them with realworld clocks and

  (2) How do we synchronize the clocks with each other?

Solar second

- Since the invention of mechanical clocks in the 17th century, time has been measured astronomically.
- Every day, the sun appears to rise on the eastern horizon, then climbs to a maximum height in the sky, and finally sinks in the west.
- The event of the sun's reaching its highest apparent point in the sky is called the transit of the sun.
- This event occurs at about noon each day.
- The interval between two consecutive transits of the sun is called the solar day.
- Since there are 24 hours in a day, each containing 3600 seconds, the solar second is defined as exactly 1/86400th of a solar day.

Mean Solar Second

- In the 1940s, it was established that the period of the earth's rotation is not constant.
- The earth is slowing down due to tidal friction and atmospheric drag.
- Based on studies of growth patterns in ancient coral, geologists now believe that 300 million years ago there were about 400 days per year.
- The length of the year (the time for one trip around the sun) is not thought to have changed; the day has simply become longer.
- In addition to this long-term trend, short-term variations in the length of the day also occur, probably caused by turbulence deep in the earth's core of molten iron.
- These revelations led astronomers to compute the length of the day by measuring a large number of days and taking the average before dividing by 86,400.
- The resulting quantity was called the mean solar second.

TAI

- With the invention of the atomic clock in 1948, it became possible to measure time much more accurately, and independent of the wiggling and wobbling of the earth, by counting transitions of the cesium 133 atom.
  - The physicists took over the job of timekeeping from the astronomers and defined the second to be the time it takes the cesium 133 atom to make exactly 9,192,631,770 transitions.
- The choice of 9,192,631,770 was made to make the atomic second equal to the mean solar second in the year of its introduction.
- Currently, several laboratories around the world have cesium 133 clocks.
- Periodically, each laboratory tells the Bureau International de l'Heure (BIR) in Paris how many times its clock has ticked. The BIR averages these to produce International Atomic Time, which is abbreviated TAI.
- Thus TAI is just the mean number of ticks of the cesium 133 clocks since midnight on Jan. 1,1958 (the beginning of time) divided by 9,192,631,770.

Time really matters

- Although TAI is highly stable and available to anyone who wants to go to the trouble of buying a cesium clock, there is a serious problem with it;
  - 86,400 TAI seconds is now about 3 msec less than a mean solar day (because the mean solar day is getting longer all the time).
- Using TAI for keeping time would mean that over the course of the years, noon would get earlier and earlier, until it would eventually occur in the first hours of the morning.

Curiosity

- *People might notice this and we could have the same kind of situation as occurred in 1582 when Pope Gregory XIII decreed that 10 days be omitted from the calendar.*
- *This event caused riots in the streets because landlords demanded a full month's rent and bankers a full month's interest, while employers refused to pay workers for the 10 days they did not work, to mention only a few of the conflicts.*
  - *The Protestant countries, as a matter of principle, refused to have anything to do with papal decrees and did not accept the Gregorian calendar for 170 years.* ☺

TAI Vs Solar Seconds and Leap Seconds

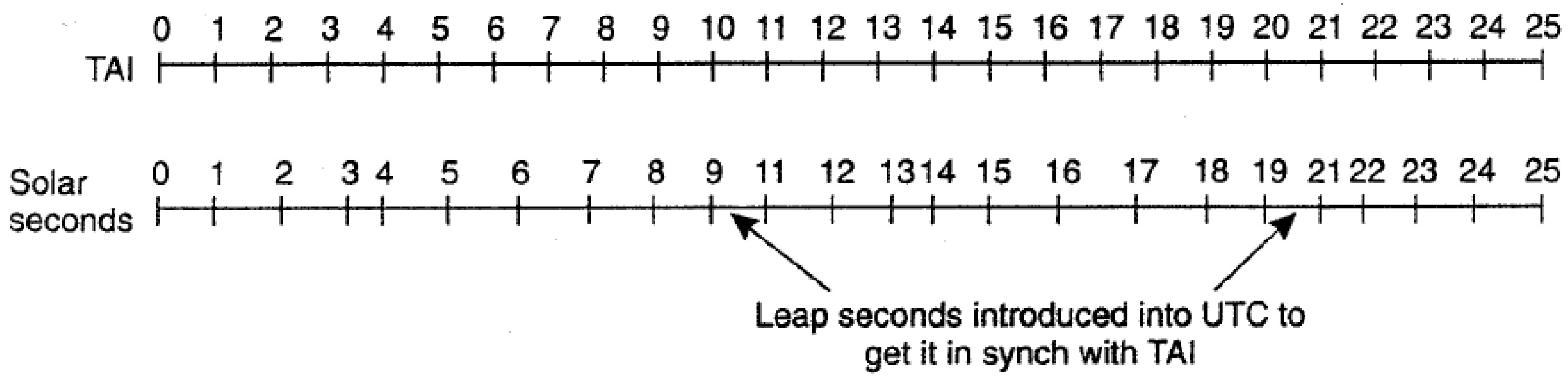


Figure 6-3. TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

BIR solves the problem by introducing leap seconds whenever the discrepancy between TAI and solar time grows to 800 msec. The use of leap seconds as a correction gives rise to a time system based on constant TAI seconds but which stays in phase with the apparent motion of the sun.
It is called Universal Coordinated Time, but is abbreviated as UTC. UTC is the basis of all modern civil timekeeping. It has essentially replaced the old standard, Greenwich Mean Time which is astronomical time.

NIST

- To provide UTC to people who need precise time, the National Institute of Standard Time (NIST) operates a shortwave radio station with call letters WWV from Fort Collins, Colorado.

- WWV broadcasts a short pulse at the start of each UTC second.

- The accuracy of WWV itself is about ±1 msec, but due to random atmospheric fluctuations that can affect the length of the signal path, in practice the accuracy is no better than ±10 msec.

- Several earth satellites also offer a UTC service. The Geostationary Environment Operational Satellite can provide UTC accurately to **0.5 msec**, and some other satellites do even better.

Global Positioning System

- As a step toward actual clock synchronization problems, we first consider a related problem, namely determining one's geographical position anywhere on Earth.
- This positioning problem is by itself solved through a highly specific dedicated distributed system, namely GPS, which is an acronym for global positioning system.
- GPS is a satellite-based distributed system that was launched in 1978.
- Although it has been used mainly for military applications, in recent years it has found its way to many civilian applications, notably for traffic navigation.

GPS

- GPS uses many satellites each circulating in an orbit at a height of approximately 20,000 km.
- Each satellite has up to four atomic clocks, which are regularly calibrated from special stations on Earth.
- A satellite continuously broadcasts its position, and time stamps each message with its local time.
- This broadcasting allows every receiver on Earth to accurately compute its own position using, in principle, only three satellites.

GPS

- Let's assume that all clocks, including the receiver's, are synchronized.
- In order to compute a position, consider first the two-dimensional case, as shown here, in which two satellites are drawn, along with the circles representing points at the same distance from each respective satellite.
- The *y-axis* represents the height, while the x-axis represents a straight line along the Earth's surface at sea level.
- Ignoring the highest point, we see that the intersection of the two circles is a unique point.

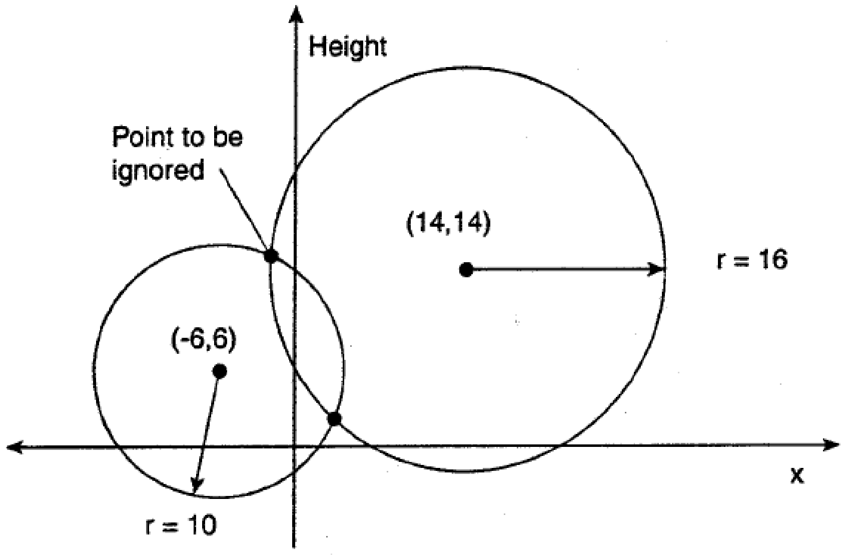


Figure 6-4. Computing a position in a two-dimensional space.

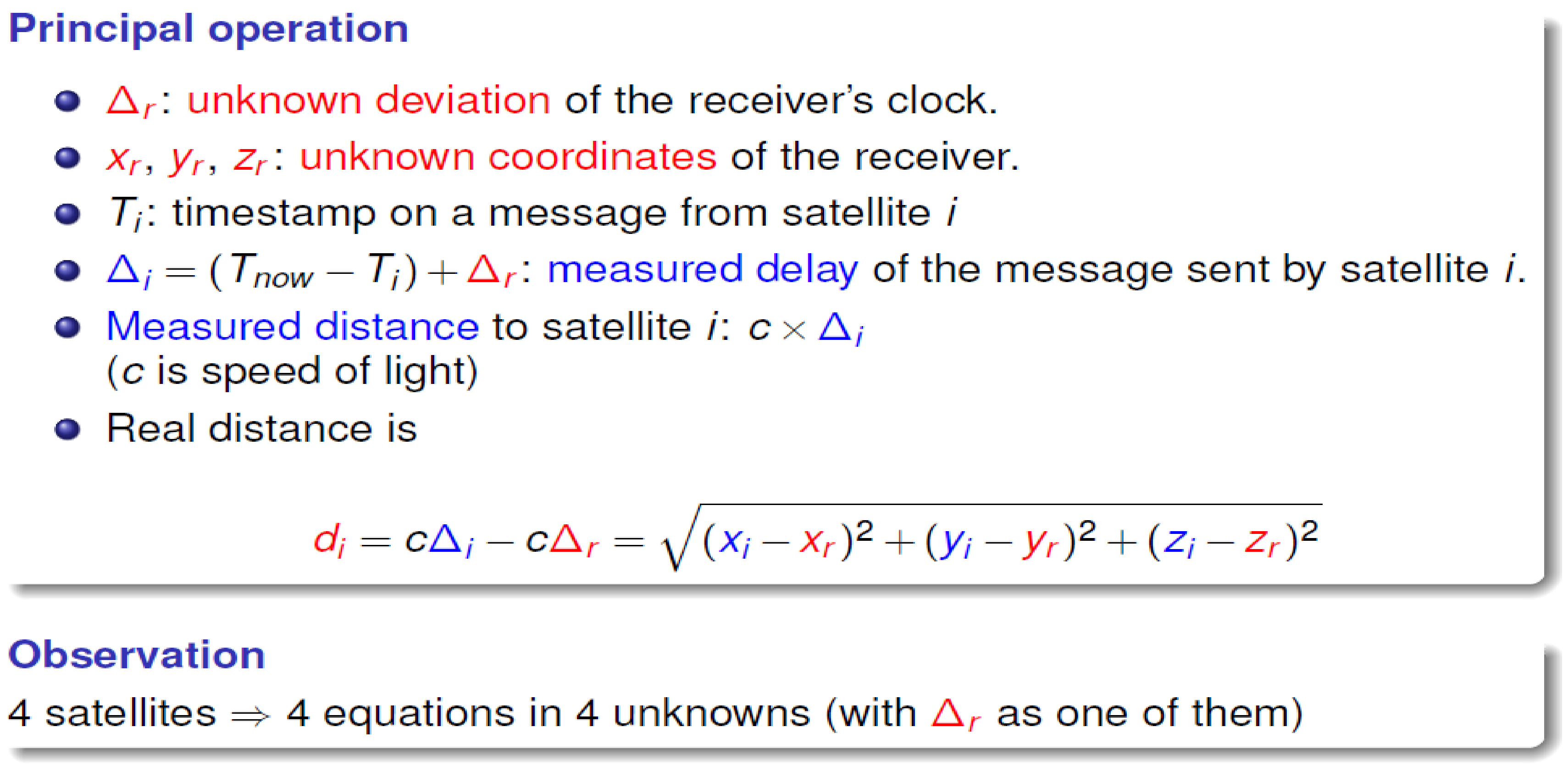
GPS and synchronization

- The principle of intersecting circles can be expanded to three dimensions, meaning that we need three satellites to determine the longitude, latitude, and altitude of a receiver on Earth.
- This positioning is all fairly straightforward, but matters become complicated when we can no longer assume that all clocks are perfectly synchronized.
- There are two important real-world facts that we need to take into account:

  1. It takes a while before data on a satellite's position reaches the receiver

  2. The receiver's clock is generally not in synch with that of a satellite.

GPS



Principal operation

- Δ_r: unknown deviation of the receiver's clock.
- x_r, y_r, z_r: unknown coordinates of the receiver.
- T_i: timestamp on a message from satellite i
- $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i.
- Measured distance to satellite i: $c \times \Delta_i$
  (c is speed of light)
- Real distance is

$$d_i = c\Delta_i - c\Delta_r = \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2}$$

Observation

4 satellites $\Rightarrow$ 4 equations in 4 unknowns (with Δ_r as one of them)

We get four equations in four unknowns, allowing us to
solve the coordinates x, y and z for the receiver, but also *the unknown deviation* Δ*.*
In other words, a GPS measurement will also give an account of the actual time.

Slide from M. Van Steen

Machine Clocks and UTC

- There is a clock in machine p that ticks on each timer interrupt.
- Denote the value of that clock by *Cp(t)*, where *t* is UTC time.
- Ideally, we have that for each machine p, *Cp(t) = t*, or, in other words, *dC/dt* = 1.
- The constant ρ (rho) is specified by the manufacturer and is known as the maximum drift rate.

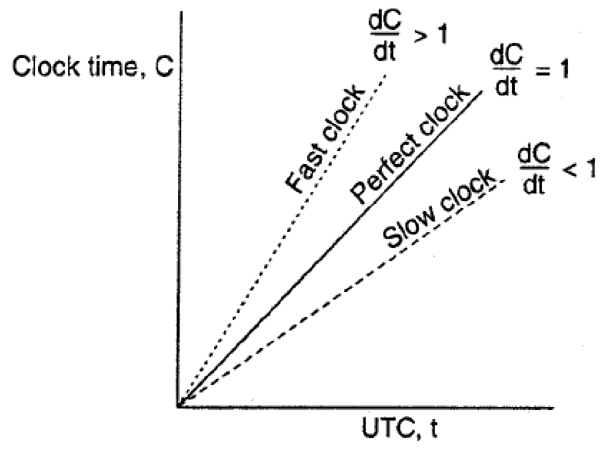


Figure 6-5. The relation between clock time and UTe when clocks tick at different rates.

In practice: $1-\rho \leq \frac{dC}{dt} \leq 1+\rho.$

Goal

Never let two clocks in any system differ by more than δ time units $\Rightarrow$ synchronize at least every $\delta/(2\rho)$ seconds.

Clock synchronization: Principle I

- **Principle I**
  - Every machine asks a time server for the accurate time at least once every δ/2ρ seconds (Network Time Protocol).
- **Note**
  - Okay, but you need an accurate measure of round trip delay, including interrupt handling and processing incoming messages.

- If the operating system designers want to guarantee that no two clocks ever differ by more than δ, clocks must be resynchronized (in software) at least every δ*/2p* seconds.

- The various algorithms differ in precisely how this resynchronization is done.

Clock synchronization: Principle II

Principle II

- Let the time server scan all machines periodically, calculate an average, and inform each machine how it should adjust its time relative to its present time.

Note

- Okay, you’ll probably get every machine in sync. You don’t even need to propagate UTC time.

Network Time Protocol

- A common approach in many protocols is to let clients contact a time server.
  - The latter can accurately provide the current time, for example, because it is equipped with a WWV receiver or an accurate clock.
- The problem, of course, is that when contacting the server, message delays will have outdated the reported time.
  - The trick is to find a good estimation for these delays.

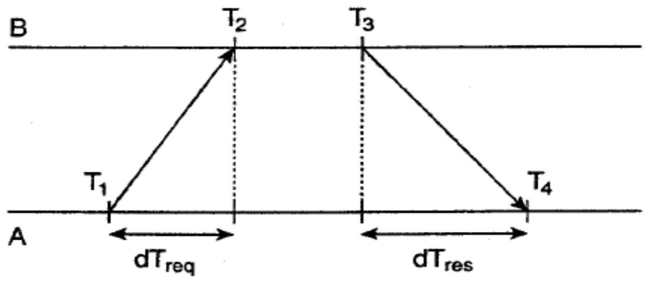


Figure 6-6. Getting the current time from a time server.

Time Corrections

You'll have to take into account that setting the time back is never allowed => smooth adjustments.

Berkeley algorithm

- In Berkeley UNIX, the time server (actually, a time daemon) is active, polling every machine from time to time to ask what time it is there.
- Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down until some specified reduction has been achieved.
- This method is suitable for a system in which no machine has a WWV receiver.
- The time daemon's time must be set manually by the operator periodically.

Berkeley algorithm

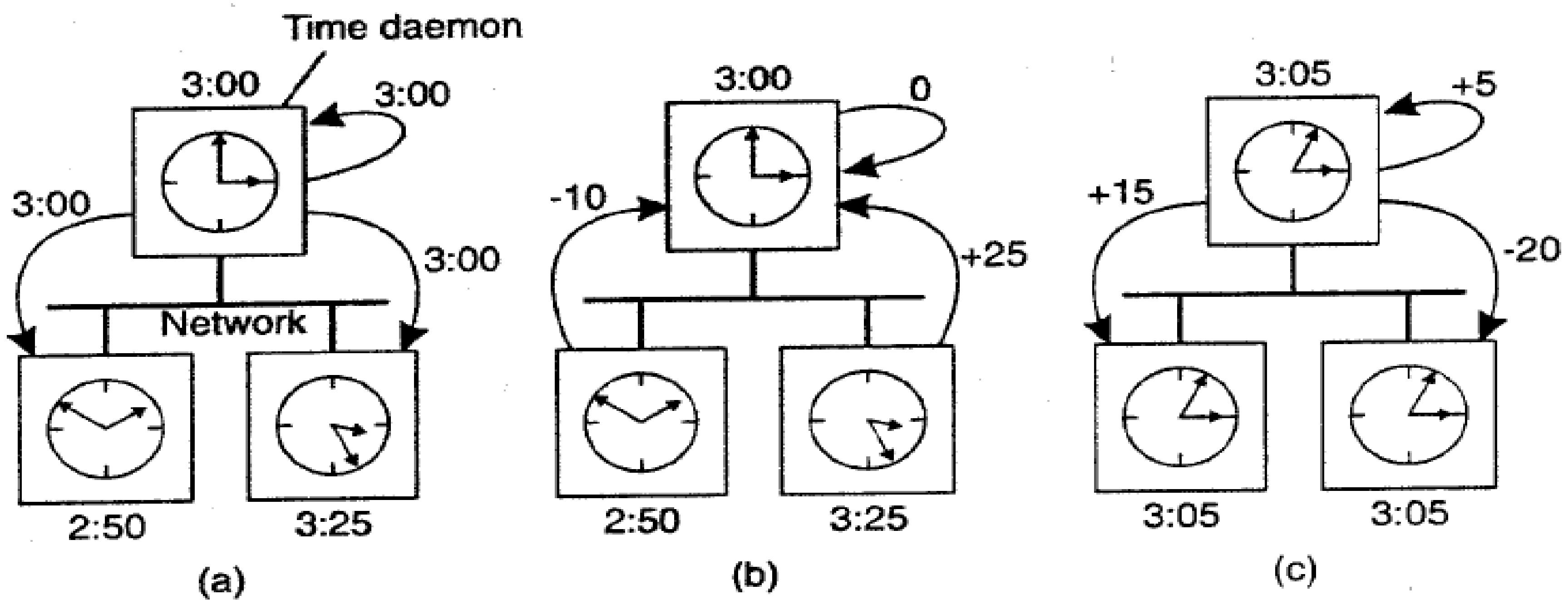


Figure 6-7. (a) The time daemon asks all the other machines for their clock values. (b) The machines answer. (c) The time daemon tells everyone how to adjust their clock.

Berkeley algorithm

- Note that for many purposes, it is sufficient that all machines agree on the same time.
- It is not essential that this time also agrees with the real time as announced on the radio every hour.
- If in the example of Fig. 6-7 the time daemon's clock would never be manually calibrated, no harm is done provided none of the other nodes communicates with external computers.
- Everyone will just happily agree on a current time, without that value having any relation with reality.

CLOCK SYNCHRONIZATION

- Physical clocks
- Logical clocks
- Vector clocks

Logical Clocks

- So far, we have assumed that clock synchronization is naturally related to real time.
- However, it may be sufficient that every node agrees on a current time, without that time necessarily being the same as the real time.
- For most algorithms, it is conventional to speak of the clocks as logical clocks.
- We showed that although clock synchronization is possible, it need not be absolute.

Agree on the order

- Lamport pointed out that what usually matters is not that all processes agree on exactly what time it is, but rather that they agree on the order in which events occur.

- The Lamport Algorithm was designed to synchronize logical clocks
- Leslie Lamport
  - http://research.microsoft.com/en-us/um/people/lamport/
- Honors
  - http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html

The Happened-before relationship

Problem

- We first need to introduce a notion of ordering before we can order anything.

The happened-before relation

- If a and b are two events in the same process, and a comes before b, then $a \to b$.
- If a is the sending of a message, and b is the receipt of that message, then $a \to b$
- If $a \to b$ and $b \to c$, then $a \to c$

Note

- This introduces a partial ordering of events in a system with concurrently operating processes.

Logical clocks

Problem

- How do we maintain a global view on the system’s behavior that is consistent with the happened-before relation?

Solution

- Attach a timestamp C(e) to each event e, satisfying the following properties:

  P1 If a and b are two events in the same process, and $a \rightarrow b$, then we demand that $C(a) < C(b)$.

  P2 If a corresponds to sending a message m, and b to the receipt of that message, then also $C(a) < C(b)$.

Problem

- How to attach a timestamp to an event when there’s no global clock => maintain a consistent set of logical clocks, one per process.

Lamport Algorithm

Solution

Each process P_i maintains a local counter C_i and adjusts this counter according to the following rules:

1: For any two successive events that take place within P_i , C_i is incremented by 1.

2: Each time a message m is sent by process P_i, the message receives a timestamp $ts(m) = C_i$.

3: Whenever a message m is received by a process *Pj* , *Pj* adjusts its local counter C_j to max{Cj ; ts(m)}; then executes step 1 before passing *m* to the application.

Notes

- Property P1 is satisfied by (1); Property P2 by (2) and (3).

Lamport algorithm

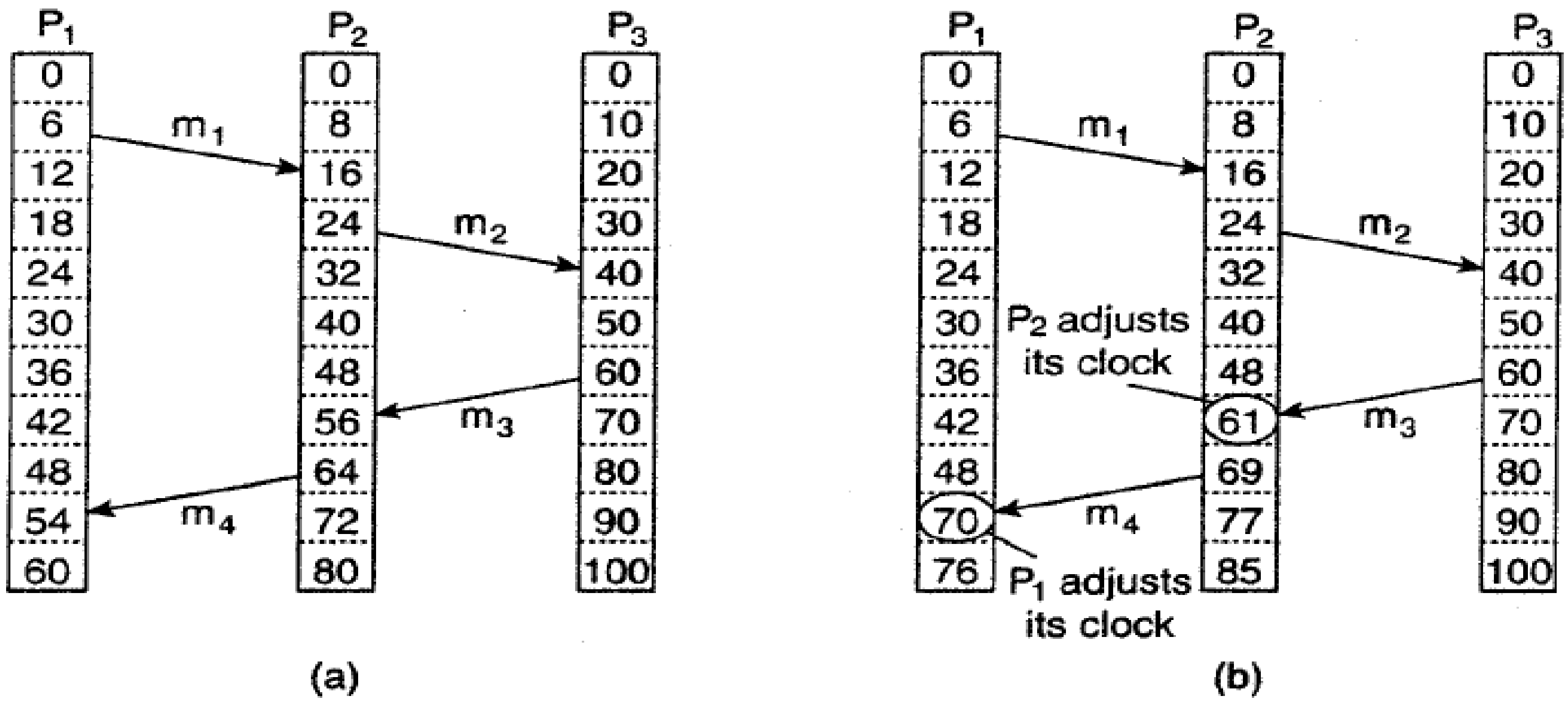


Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

Logical clocks among layers

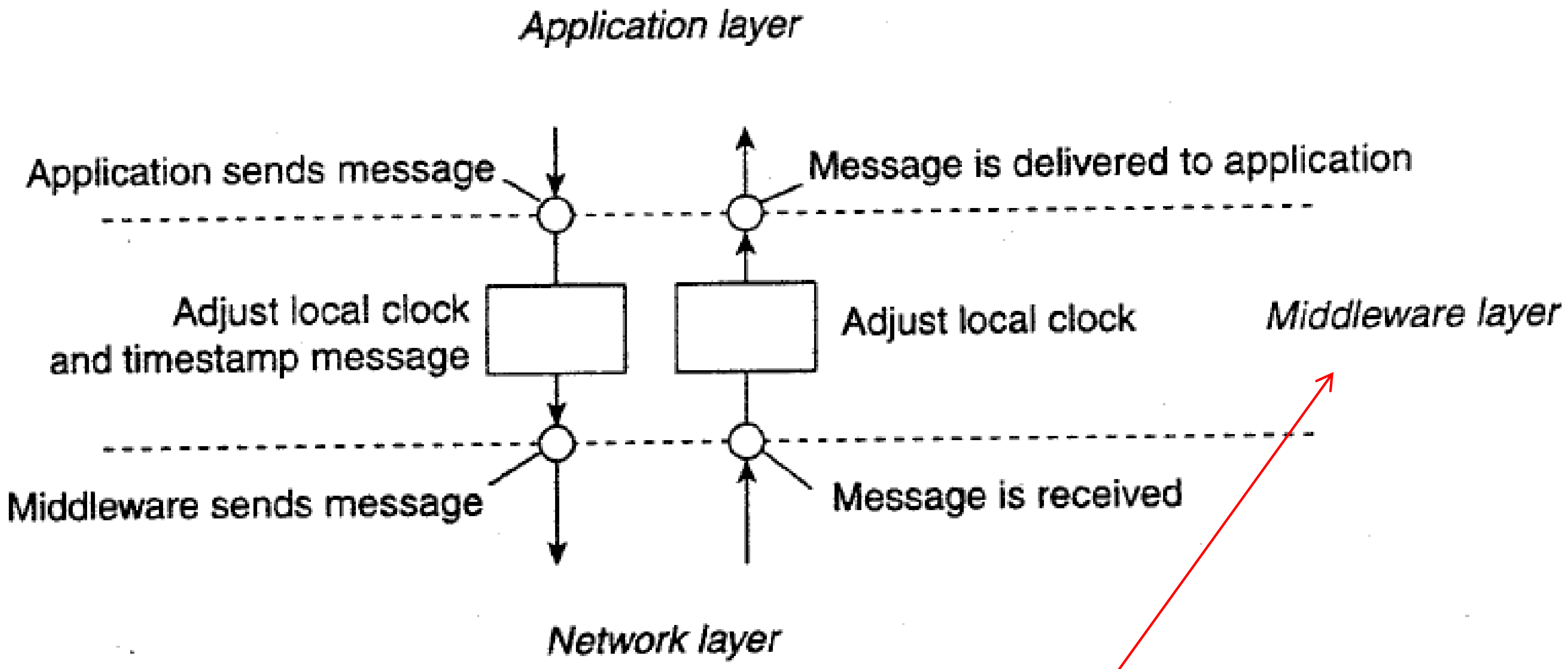


Figure 6-10. The positioning of Lamport's logical clocks in distributed systems.

Synchronization of clocks is performed in the middleware layer.

Vector Clocks

- Lamport's logical clocks lead to a situation where all events in a distributed system are totally ordered with the property that if event *a* happened before event *b,* then *a* will also be positioned in that ordering before *b,* that is, C*(a)* < C *(b).*
- However, with Lamport clocks, nothing can be said about the relationship between two events *a* and *b* by merely comparing their time values *C(a)* and *C(b),* respectively.
- In other words, if *C(a)* < *C(b),* then this does not necessarily imply that *a* indeed happened before *b.*
- This is possible with Vector Clocks.

Concurrency in Distributed Systems

Concurrency in Distributed Systems

- Fundamental to distributed systems is the concurrency and collaboration among multiple processes.

- In many cases, this also means that processes will need to simultaneously access the same resources.

- To prevent that such concurrent accesses corrupt the resource, or make it inconsistent, solutions are needed to grant mutual exclusive access by processes.

- In this section, we take a look at some of the most important distributed algorithms that have been proposed.

Token-based solutions

- In token-based solutions mutual exclusion is achieved by passing a special message between the processes, known as a token.
- There is only one token available and whoever has that token is allowed to access the shared resource.
- When finished, the token is passed on to a next process.
- If a process having the token is not interested in accessing the resource, it simply passes it on.

Token-based solutions

- Token-based solutions have a few important properties.

- First, depending on how the processes are organized, they can fairly easily ensure that every process will get a chance at accessing the resource.
  - In other words, they avoid starvation.

- Second, deadlocks by which several processes are waiting for each other to proceed, can easily be avoided, contributing to their simplicity.

Token-based solutions

- Unfortunately, the main drawback of token-based solutions is a rather serious one:
  - when the token is lost (e.g., because the process holding it crashed), an intricate distributed procedure needs to be started to ensure that a new token is created, but above all, that it is also the only token.

Permission-based approach

- As an alternative, many distributed mutual exclusion algorithms follow a permission-based approach.
- In this case a process wanting to access the resource first requires the permission of other processes.
- There are many different ways toward granting such a permission:
  - Via a centralized server.
  - Completely decentralized, using a peer-to-peer system.
  - Completely distributed, with no topology imposed.
  - Completely distributed along a (logical) ring.

Centralized Algorithm

- The most straightforward way to achieve mutual exclusion in a distributed system is to simulate how it is done in a one-processor system.

- One process is elected as the coordinator. Whenever a process wants to access a shared resource, it sends a request message to the coordinator stating which resource it wants to access and asking for permission.

- If no other process is currently accessing that resource, the coordinator sends back a reply granting permission.

Centralized Algorithm

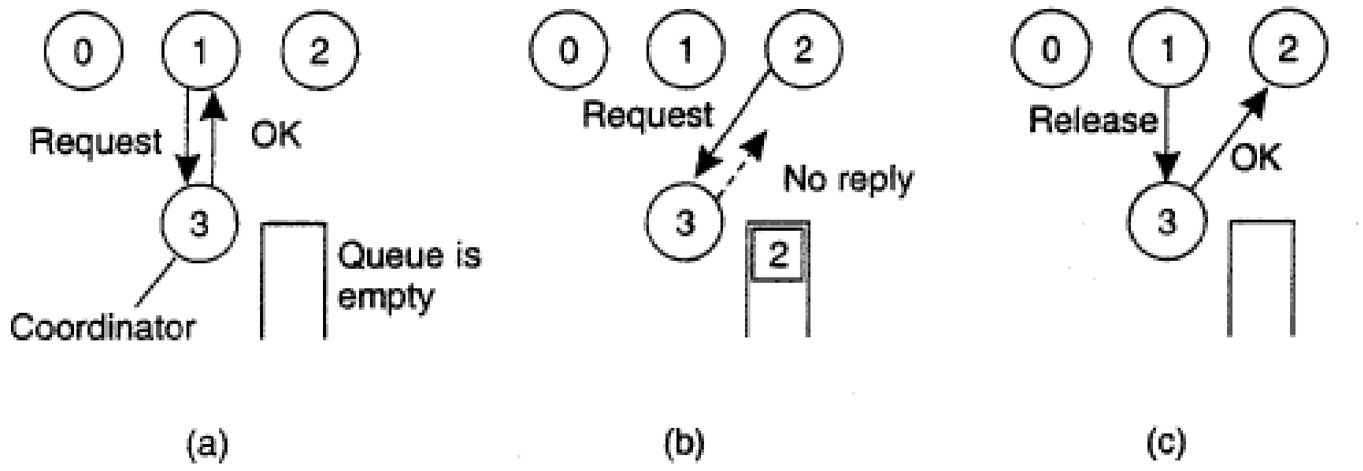


Figure 6-14. (a) Process 1 asks the coordinator for permission to access a shared resource. Permission is granted. (b) Process 2 then asks permission to access the same resource. The coordinator does not reply. (c) When process 1 releases the resource, it tells the coordinator, which then replies to 2.

Centralized Algorithm

- The centralized approach has shortcomings.
- The coordinator is a single point of failure, so if it crashes, the entire system may go down.
- If processes normally block after making a request, they cannot distinguish a dead coordinator from "permission denied" since in both cases no message comes back.
- In addition, in a large system, a single coordinator can become a performance bottleneck.
- Nevertheless, the benefits coming from its simplicity outweigh in many cases the potential drawbacks.

Decentralized Algorithm

Principle

- Assume every resource is replicated *n* times, with each replica having its own coordinator => access requires a majority vote from m > n/2 coordinators.
- A coordinator always responds immediately to a request.

Assumption

When a coordinator crashes, it will recover quickly, but will have forgotten about permissions it had granted.

- It has been shown that this algorithm is probabilistically correct: the probability of violating correctness is much smaller than the availability of any resource.

Distributed algorithm

- To many, having a probabilistically correct algorithm is just not good enough.
- So researchers have looked for deterministic distributed mutual exclusion algorithms.
  - Lamport's 1978 paper on clock synchronization presented the first one.
  - Ricart and Agrawala (1981) made it more efficient.
- This algorithm requires that there be a total ordering of all events in the system.
- That is, for any pair of events, such as messages, it must be unambiguous which one actually happened first.
- Lamport's algorithm is one way to achieve this ordering and can be used to provide timestamps for distributed mutual exclusion.

Distributed algorithm

The algorithm works as follows:

- When a process wants to access a shared resource, it builds a message containing the name of the resource, its process number, and the current (logical) time.
- It then sends the message to all other processes, conceptually including itself.
- The sending of messages is assumed to be reliable; that is, no message is lost.
- When a process receives a request message from another process, the action it takes depends on its own state with respect to the resource named in the message.

Distributed algorithm

Three different cases:

1. If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to the sender.
2. If the receiver already has access to the resource, it simply does not reply. Instead, it queues the request.
3. If the receiver wants to access the resource as well but has not yet done so, it compares the timestamp of the incoming message with the one contained in the message that it has sent everyone. The lowest one wins.

Distributed algorithm

- After sending out requests asking permission, a process sits back and waits until everyone else has given permission.
- As soon as all the permissions are in, it may go ahead.
- When it is finished with the resource, it sends *OK* messages to all processes on its queue and deletes them all from the queue.

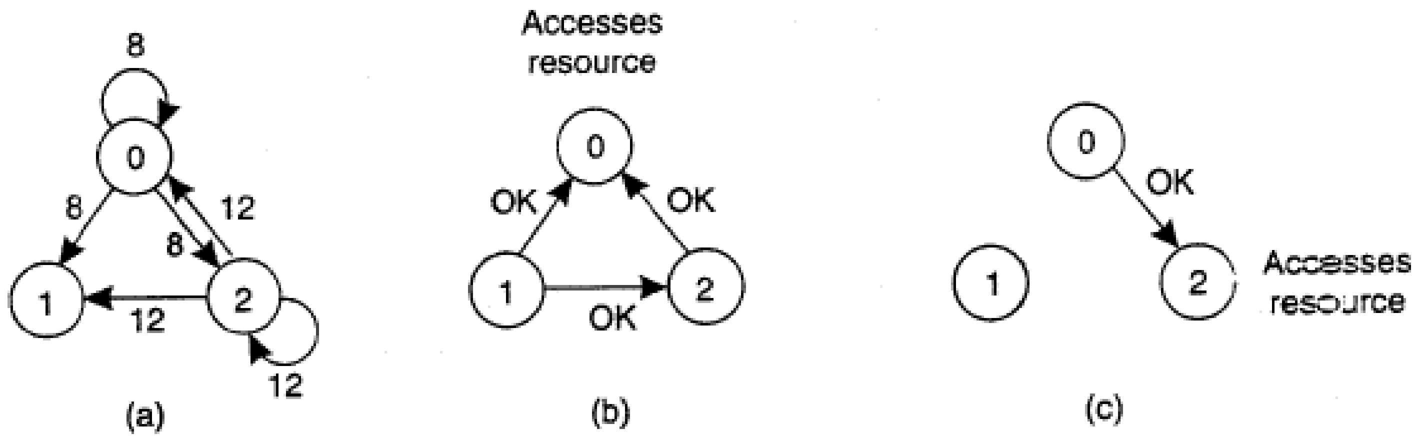


Figure 6-15. (a) Two processes want to access a shared resource at the same moment. (b) Process 0 has the lowest timestamp. so it wins. (c) When process 0 is done, it sends an *OK* also, so 2 can now go ahead.

A Token Ring Algorithm

- In software, a logical ring is constructed in which each process is assigned a position in the ring.
- The ring positions may be allocated in numerical order of network addresses or some other means.
- It does not matter what the ordering is. All that matters is that each process knows who is next in line after itself.

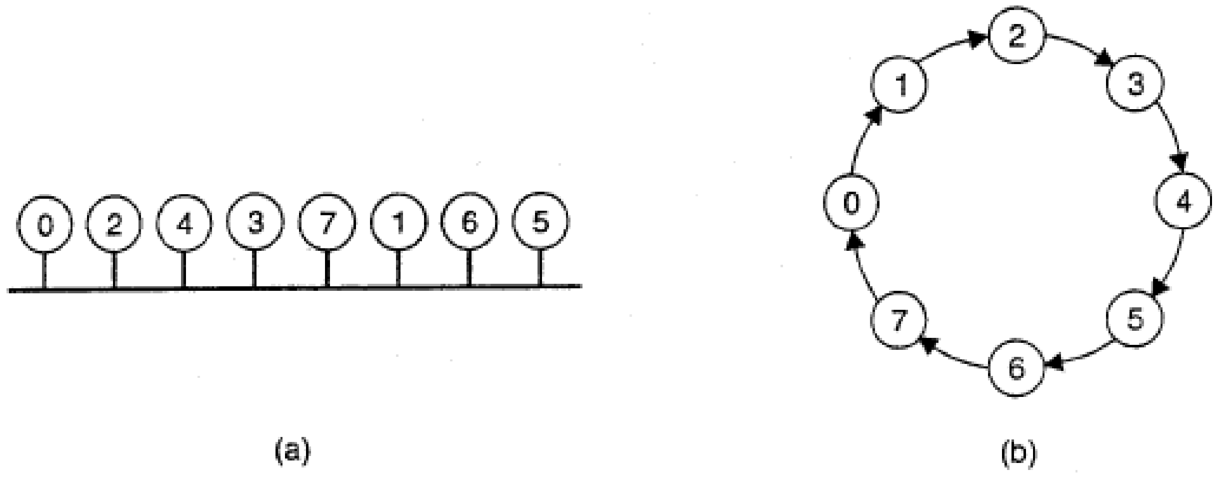


Figure 6-16. (a) An unordered group of processes on a network. (b) A logical ring constructed in software.

A Token Ring Algorithm

- When the ring is initialized, process 0 is given a token.
- The token circulates around the ring.
- It is passed from process *k* to process *k* +1 in point-to-point messages.
  - When a process acquires the token from its neighbor, it checks to see if it needs to access the shared resource.
  - If so, the process goes ahead, does all the work it needs to, and releases the resources.
  - After it has finished, it passes the token along the ring.
- It is not permitted to immediately enter the resource again using the same token.

A Token Ring Algorithm

- As usual, this algorithm has problems too. If the token is ever lost, it must be regenerated.
- In fact, detecting that it is lost is difficult, since the amount of time between successive appearances of the token on the network is unbounded.
- The fact that the token has not been spotted for an hour does not mean that it has been lost; somebody may still be using it.

A Token Ring Algorithm

- The algorithm also runs into trouble if a process crashes, but recovery is easier than in the other cases.
- If we require a process receiving the token to acknowledge receipt, a dead process will be detected when its neighbor tries to give it the token and fails.
- At that point the dead process can be removed from the group.

Election Algorithms

- Many distributed algorithms require one process to act as coordinator, initiator, or otherwise perform some special role.
- In general, it does not matter which process takes on this special responsibility, but one of them has to do it.
  – Algorithms for electing a coordinator (using this as a generic name for the special process).

- If all processes are exactly the same, with no distinguishing characteristics, there is no way to select one of them to be special .

Election Algorithms

- We will assume that each process has a unique number, for example, its network address (for simplicity, we will assume one process per machine).

- In general, election algorithms attempt to locate the process with the highest process number and designate it as coordinator.

- The algorithms differ in the way they do the location.

The Bully Algorithm

When any process notices that the coordinator is no longer responding to requests, it initiates an election.

A process *P* holds an election as follows:

1. *P* sends an *ELECTION* message to all processes with higher numbers.
2. If no one responds, *P* wins the election and becomes coordinator.
3. If one of the higher-ups answers, it takes over. *P*'s job is done.

Election by bullying

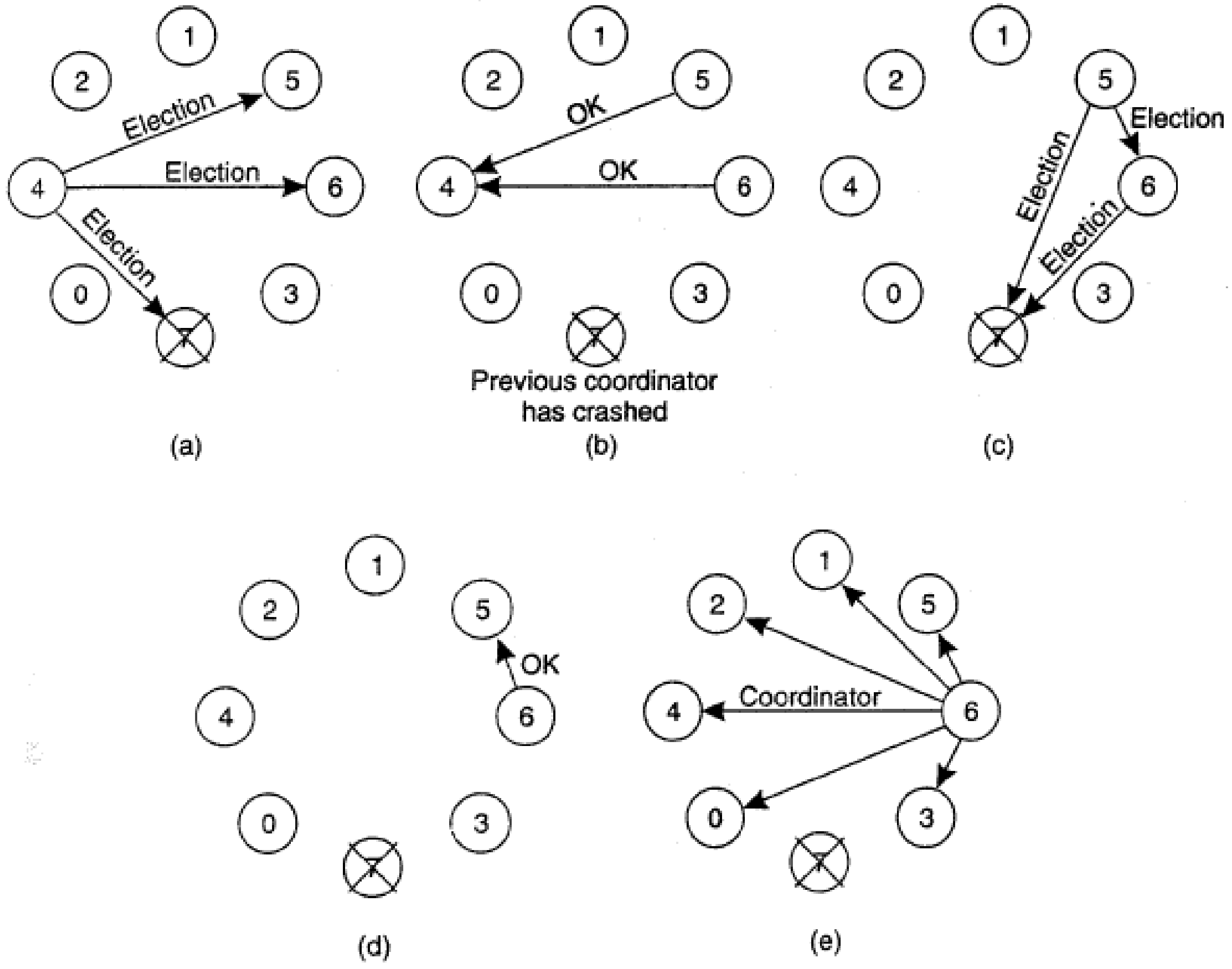


Figure 6·20. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

End of PART I

- Readings
  - Distributed Systems, Chapter 6.

PART II

Reasons for Replication

- Data are replicated to increase the reliability of a system.
- Replication for performance
  - Scaling in numbers
  - Scaling in geographical area
    - Place copies near the process that is using them
- Important features
  - Gain in performance
    - The process near the copy perceives good performance
  - Cost of increased bandwidth for maintaining replication (updating all the replicas!!!)

- Price to be paid: Consistency

Access-to-update ratio

- Consider a process *P* that accesses a local replica *N* times per second, whereas the replica itself is updated *M* times per second.
- Assume that an update completely refreshes the previous version of the local replica.
- If $N < M$, that is, the access-to-update ratio is very low, we have the situation where many updated versions of the local replica will never be accessed by *P*, rendering the network communication for those versions useless.

Consistency

- A collection of copies is consistent when the copies are always the same.
- This means that a read operation performed at any copy will always return the same result.
- Consequently, when an update operation is performed on one copy, the update should be propagated to all copies before a subsequent operation takes place, no matter at which copy that operation is initiated or performed.
- Update as single atomic operation or transaction.

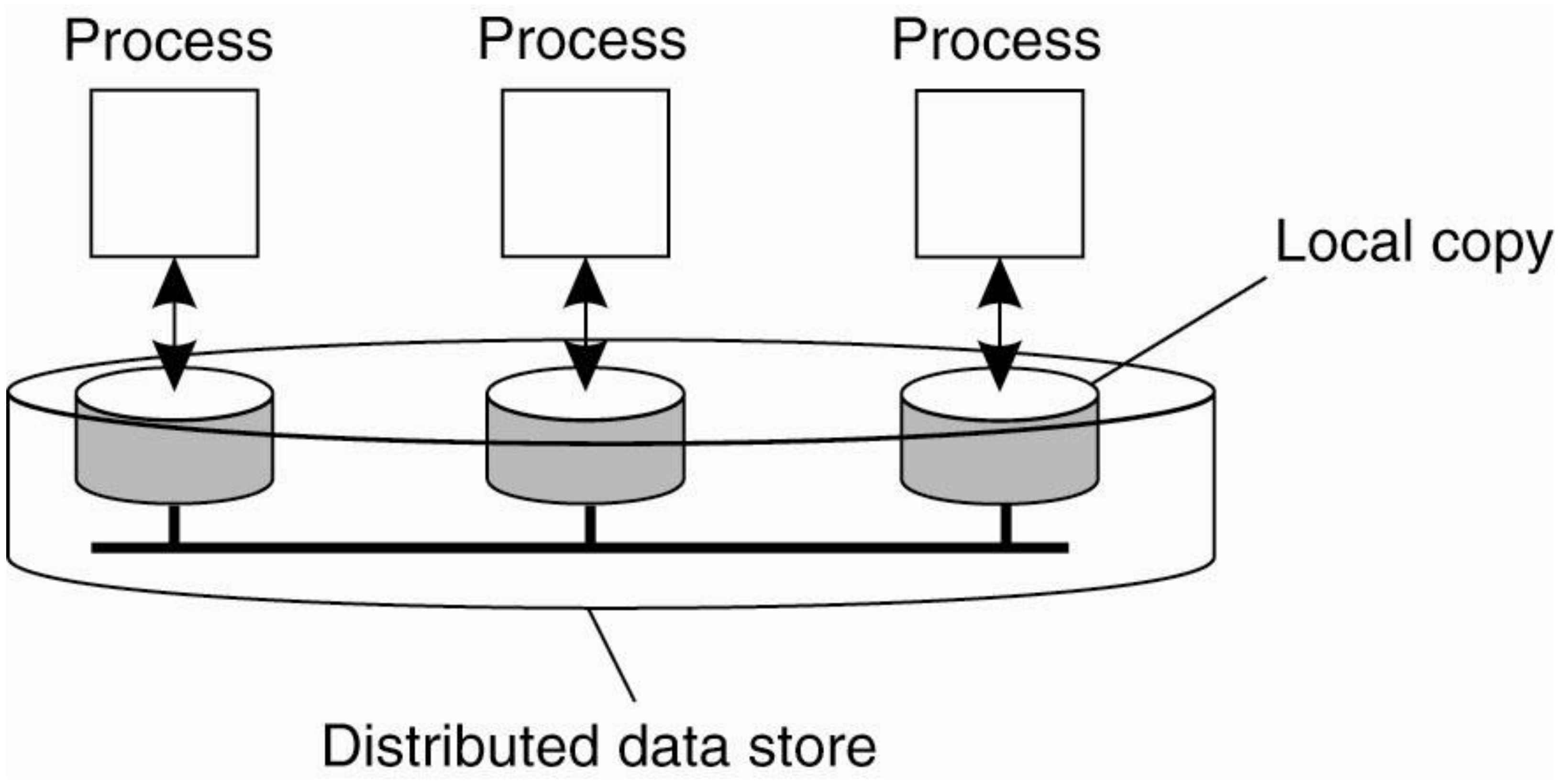
Issues related to consistency

- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols

Data-centric Consistency Models

- A data store may be physically distributed across multiple machines.
- In particular, each process that can access data from the store is assumed to have a local (or nearby) copy available of the entire store.
- Write operations are propagated to the other copies.
- A data operation is classified as a write operation when it changes the data, and is otherwise classified as a read operation.

Data-centric Consistency Models



- Figure 7-1. The general organization of a logical data store, physically distributed and replicated across multiple processes.

Consistency Model

- A consistency model is essentially a contract between processes and the data store.
- It says that if processes agree to obey certain rules, the store promises to work correctly.
- Normally, a process that performs a read operation on a data item, expects the operation to return a value that shows the results of the last write operation on that data.

Content

- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols

Client-centric consistency

Goal

- How we can perhaps avoid system wide consistency, by concentrating on what specific clients want, instead of what should be maintained by servers.

Client-centric consistency

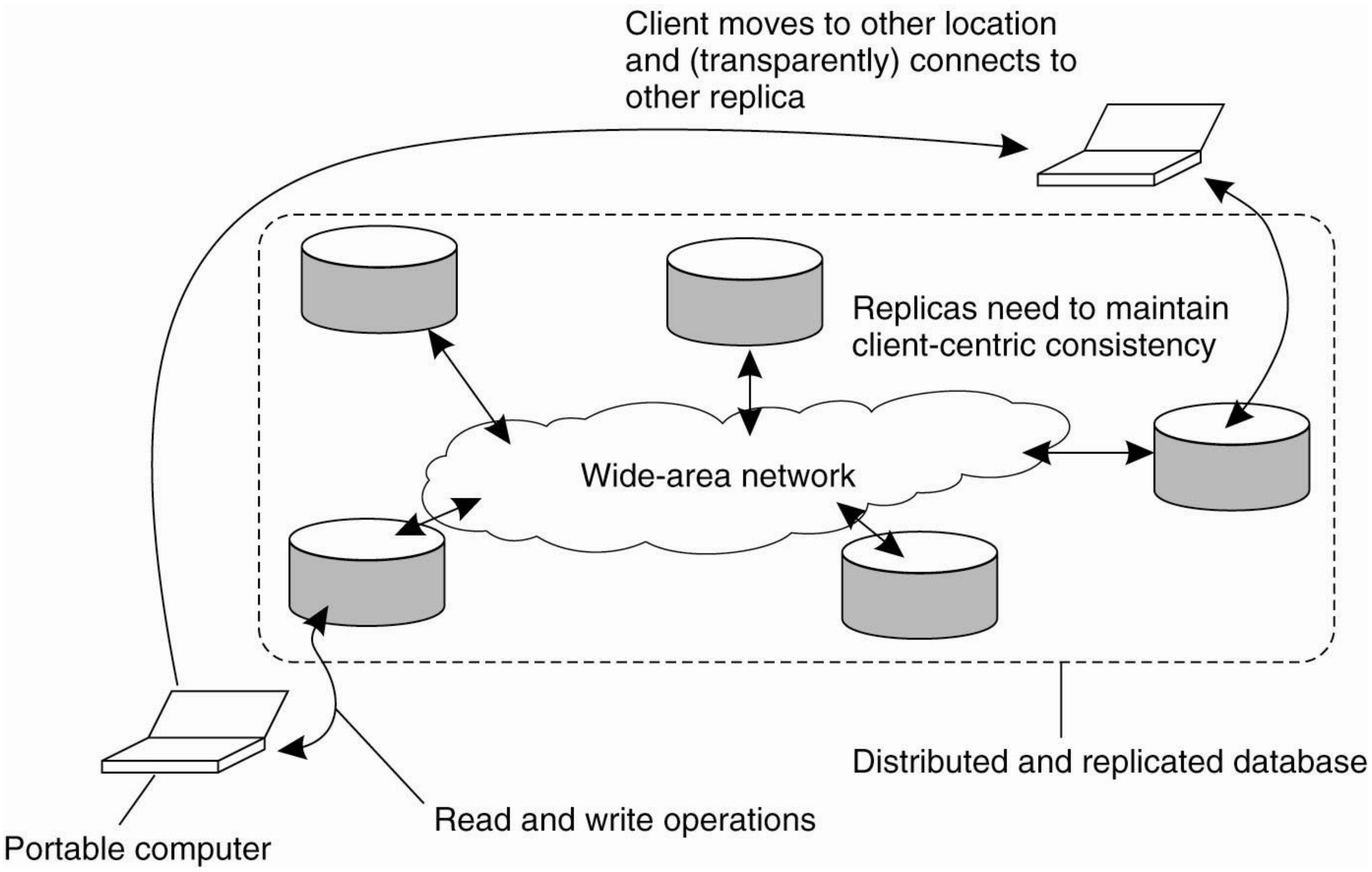

Client moves to other location and (transparently) connects to other replica
Replicas need to maintain client-centric consistency
Wide-area network
Distributed and replicated database
Read and write operations
Portable computer

Figure 7-11. The principle of a mobile user accessing different replicas of a distributed database.

Content

- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols

Replica Management

- A key issue for any distributed system that supports replication is to decide:
  - Where?
  - When? and
  - by whom replicas should be placed?
  - and subsequently which mechanisms to use for keeping the replicas consistent?

Replica Management

- Two important issues:
  - Replica-server placement is concerned with finding the best locations to place a server that can host (part of) a data store.
  - Content placement deals with finding the best servers for placing content.

Replica-Server Placement

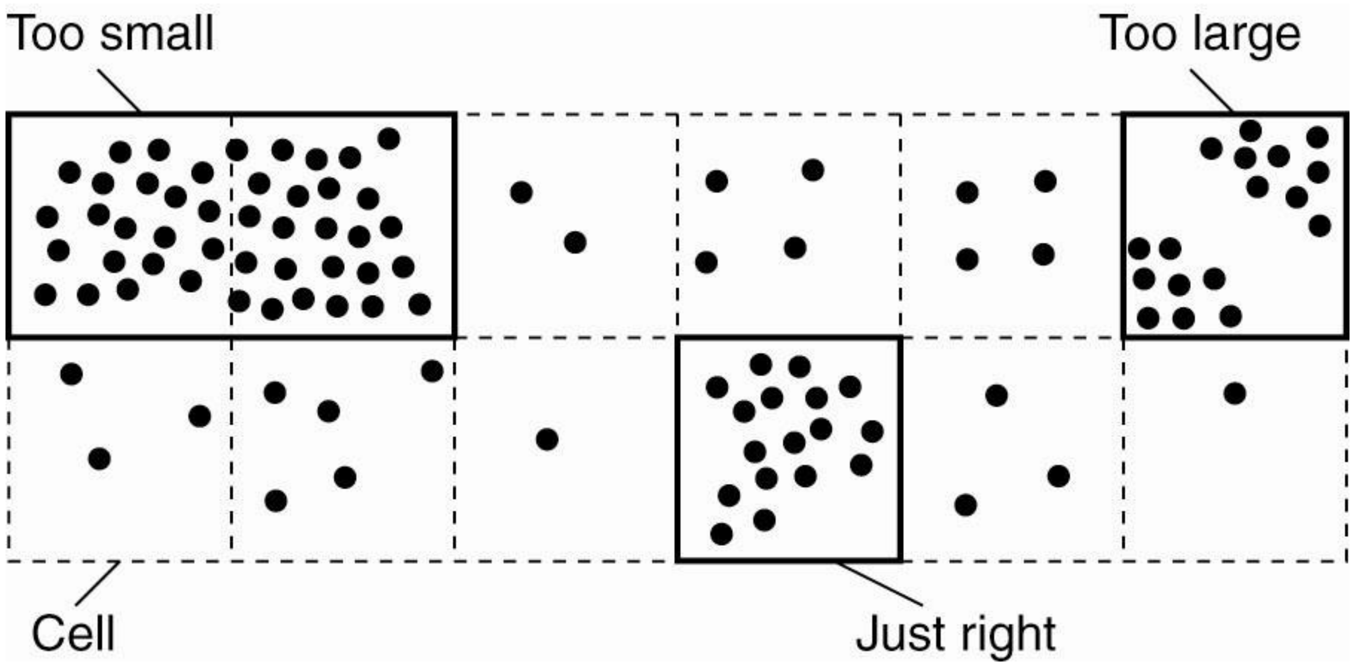


Figure 7-16. Choosing a proper cell size for server placement.

A region is identified to be a collection of nodes accessing the same content, but for which the internode latency is low.
The goal of the algorithm is first to select the most demanding regions - that is, the one with the most nodes - and then to let one of the nodes in such a region act as replica server.

Content Replication and Placement: server-initiated and client-initiated

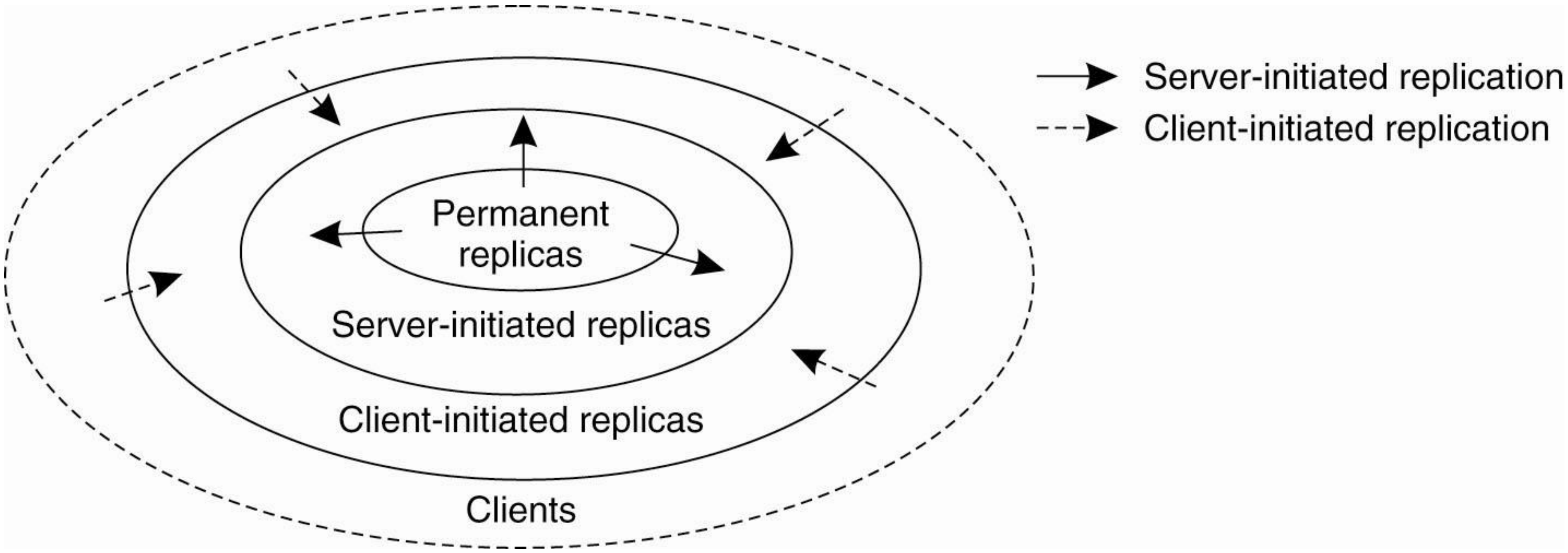

Server-initiated replication
Client-initiated replication
Permanent replicas
Server-initiated replicas
Client-initiated replicas
Clients

Figure 7-17. The logical organization of different kinds of copies of a data store into three concentric rings.

Permanent Replicas

- Permanent replicas can be considered as the initial set of replicas that constitute a distributed data store.
- In many cases, the number of permanent replicas is small.
- Consider, for example, a Web site. Distribution of a Web site generally comes in one of two forms:

1. The first kind of distribution is one in which the files that constitute a site are replicated across a limited number of servers at a single location. Whenever a request comes in, it is forwarded to one of the servers, for instance, using a round-robin strategy.
2. The second form of distributed Web sites is what is called mirroring. In this case, a Web site is copied to a limited number of servers, called mirror sites, which are geographically spread across the Internet. In most cases, clients simply choose one of the various mirror sites from a list offered to them.

Server-Initiated Replicas

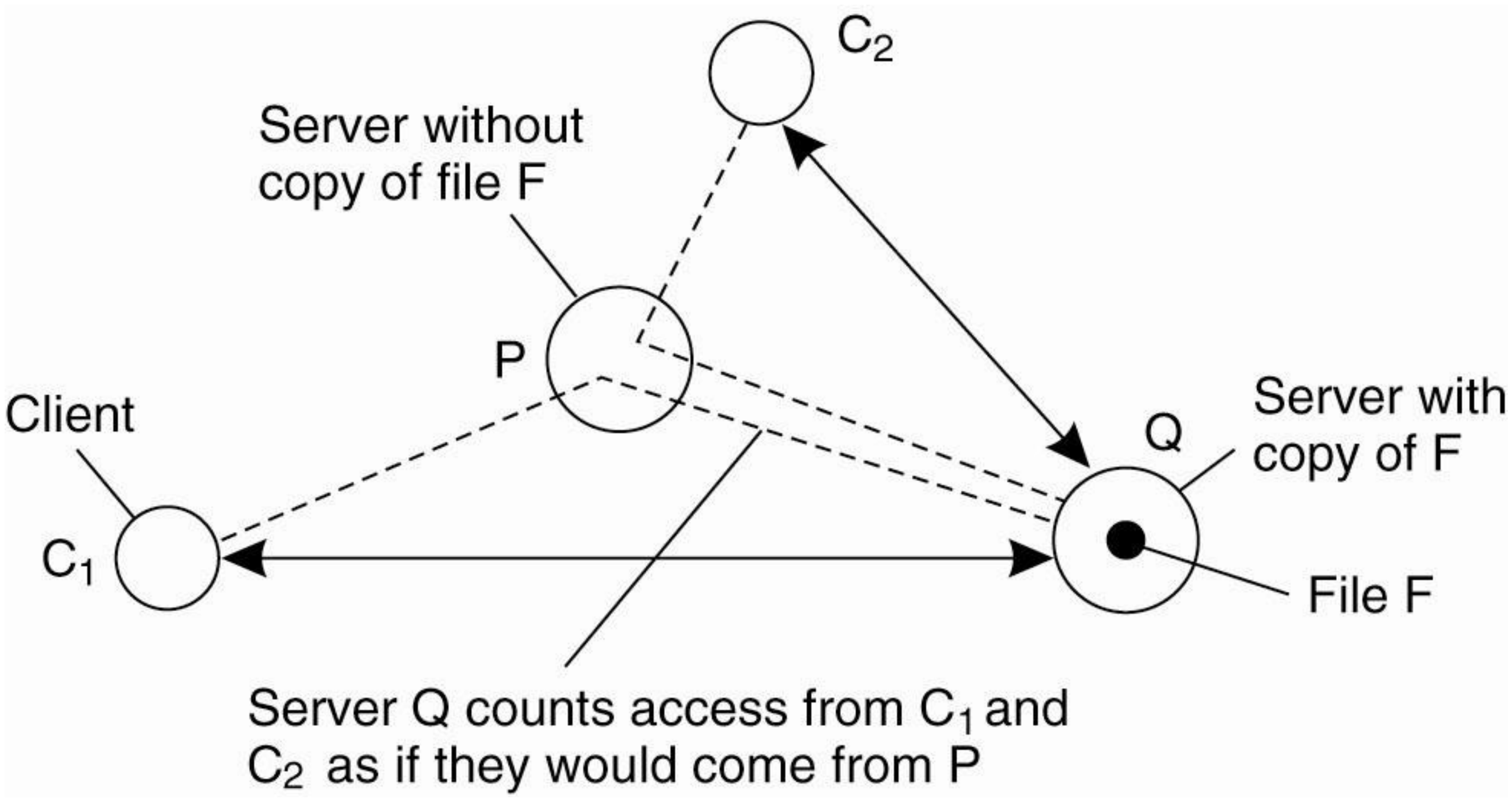

C2
Server without copy of file F
P
Client
C1
Q
Server with copy of F
File F
Server Q counts access from C1 and C2 as if they would come from P

Figure 7-18. Counting access requests from different clients.

Deletion and replication thresholds

Algorithm:

Keep track of access counts per file, aggregated by considering server closest to requesting clients

- Number of accesses drops below threshold D => drop file
- Number of accesses exceeds threshold R => replicate file
- Number of accesses between D and R => migrate file

Client-Initiated Replicas

- Client-initiated replicas are more commonly known as (client) caches.
- Data are generally kept in a cache for a limited amount of time
- Placement of client caches is relatively simple: a cache is normally placed on the same machine as its client, or otherwise on a machine shared by clients on the same local-area network.
- However, in some cases, extra levels of caching are introduced by system administrators by placing a shared cache between a number of departments or organizations, or even placing a shared cache for an entire region such as a province or country.

Push and pull-based protocol

- Push-based (or server-based protocol)
  - Updates are propagated to other replicas without those replicas asking for the updates.
  - Applied when replicas need to maintain high degree of consistency

- Pull-based (or client-based protocol)
  - A server or client requests another server or client to send it any updates it has at that moment.
  - Efficient when the read-to-update ratio is relatively low.

Content

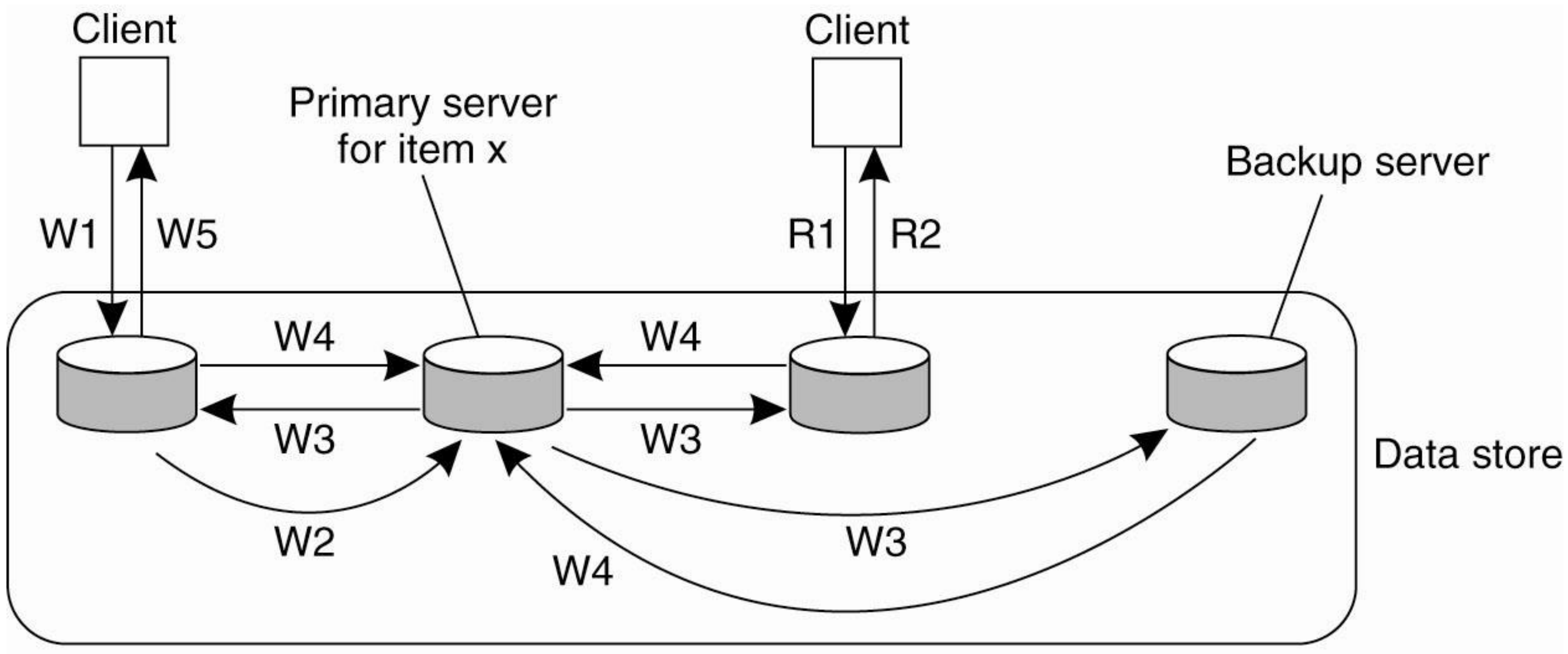
- Data-centric consistency
- Client-centric consistency
- Replica management
- Consistency protocols

Consistency Protocols

A consistency protocol describes an implementation of a specific consistency model.

- Primary-based protocols
- Replicated-write protocols

Remote-Write Protocols



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

All write operations need to be forwarded to a **fixed single server**.

Read operations can be carried out locally. Such schemes are also known as **primary-backup protocols**

Local-Write Protocols

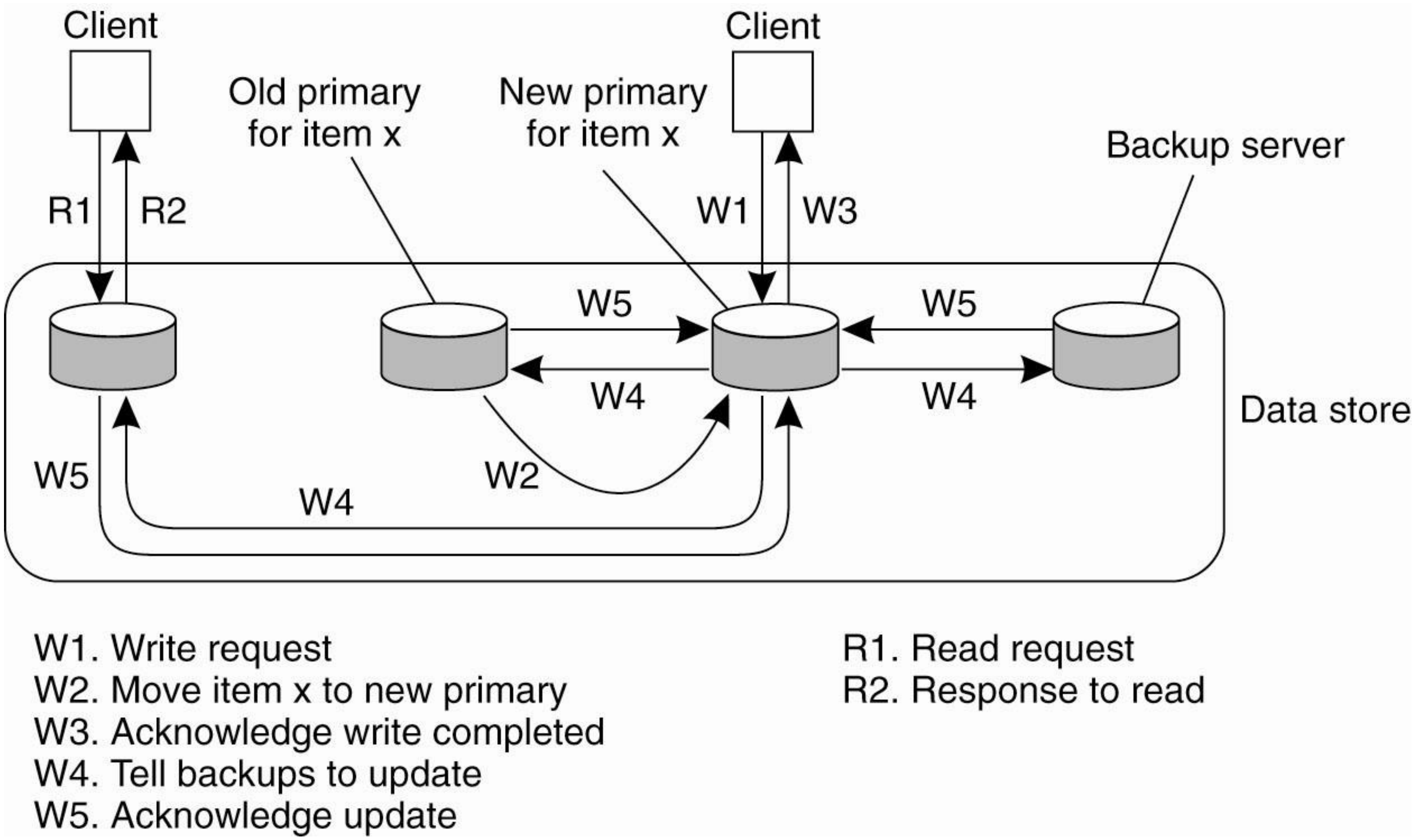


Figure 7-21. Primary-backup protocol in which the primary migrates to the process wanting to perform an update.

Consistency Protocols

A consistency protocol describes an implementation of a specific consistency model.

- Primary-based protocols
- Replicated-write protocols

Active Replication

- In replicated-write protocols, write operations can be carried out at multiple replicas instead of only one.
- In active replication, each replica has an associated process that carries out update operations.
- In other words, the operation is sent to each replica.

Quorum-Based Protocols

- Clients have to request and acquire the permission of multiple servers before either reading or writing a replicated data item.
- Majority scheme
- Gifford’s scheme, two constraints:

1. $N_R + N_W > N$
2. $N_W > N / 2$

Where N is the number of replicas, N_R is the read quorum, N_W is the write quorum.

- To read or write a file, N_R and N_W must be achieved.

End of PART II

- Readings
  - Distributed Systems, Chapter 7