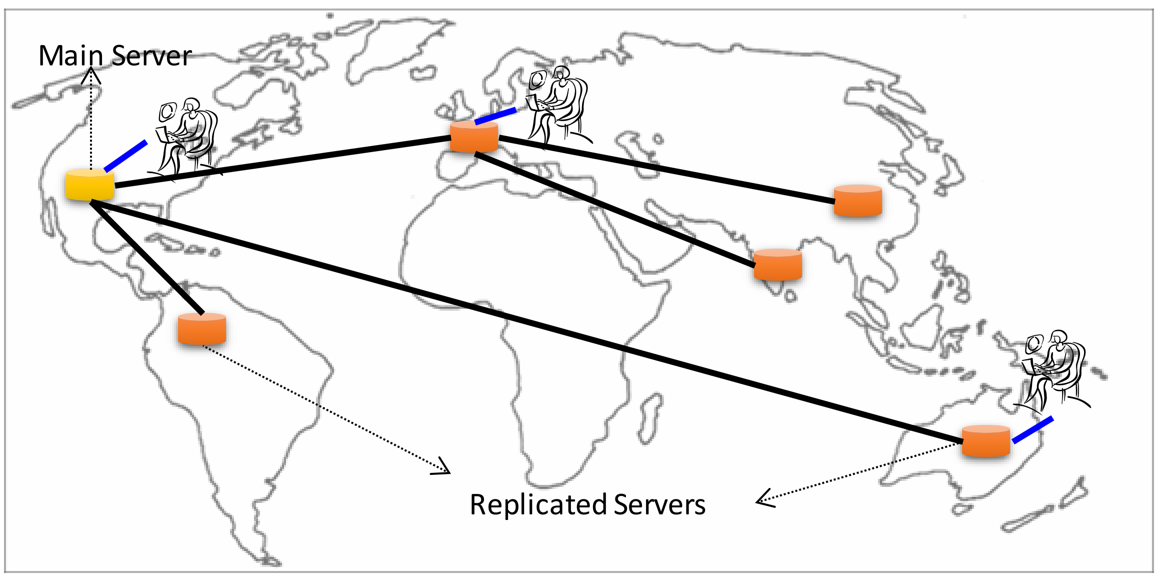
Consistency and Replication

Why Replication?

- Replication is the process of maintaining the data at multiple computers
- Replication is necessary for:
  1. Improving performance
     - A client can access the replicated copy of the data that is near to its location
  2. Increasing the availability of services
     - Replication can mask failures such as server crashes and network disconnection
  3. Enhancing the scalability of the system
     - Requests to the data can be distributed to many servers which contain replicated copies of the data
  4. Securing against malicious attacks
     - Even if some replicas are malicious, secure data can be guaranteed to the client by relying on the replicated copies at the non-compromised servers

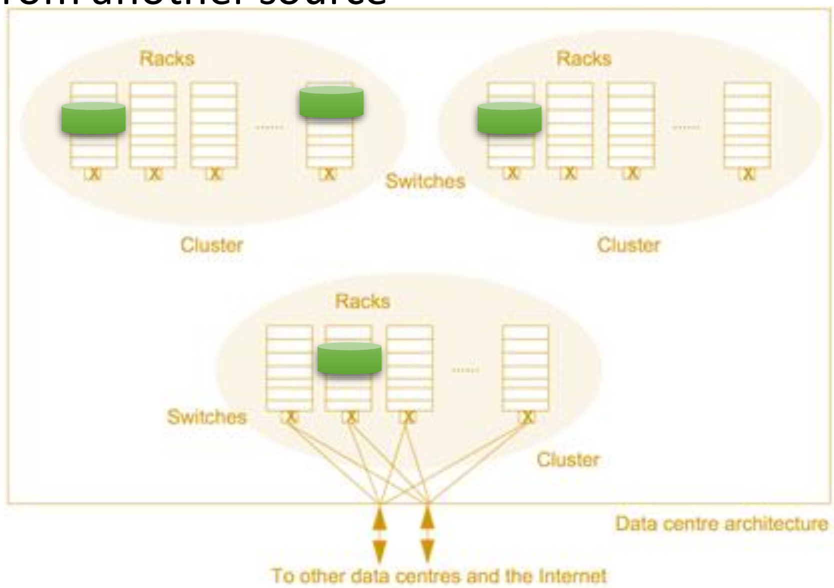
1. Replication for Improving Performance

- Example Applications
  - Caching webpages at the client browser
  - Caching IP addresses at clients and DNS Name Servers
  - Caching in Content Delivery Network (CDNs)
    - Commonly accessed contents, such as software and streaming media, are cached at various network locations



2. Replication for High-Availability

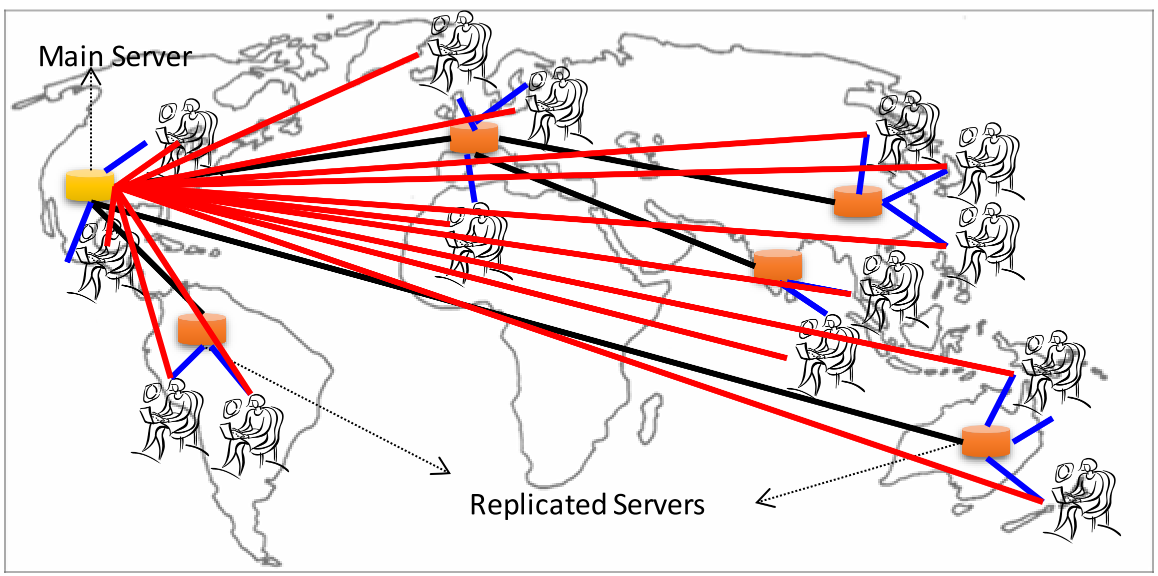
- Availability can be increased by storing the data at replicated locations (instead of storing one copy of the data at a server)
- Example: Google File-System and Chubby replicate the data at computers across different racks, clusters and data-centers
  - If one computer or a rack or a cluster crashes, then the data can still be accessed from another source



3. Replication for Enhancing Scalability

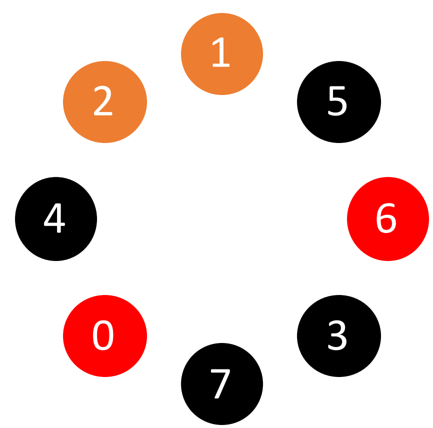
- Distributing the data across replicated servers helps in avoiding bottle-necks at the main server
  - It balances the load between the main and the replicated servers

- Example: Content Delivery Networks decrease the load on main servers of the website



4. Replication for Securing Against Malicious Attacks

- If a minority of the servers that hold the data are malicious, the non-malicious servers can outvote the malicious servers, thus providing security.
- The technique can also be used to provide fault-tolerance against non-malicious but faulty servers
- Example: In a peer-to-peer system, peers can coordinate to prevent delivering faulty data to the requester

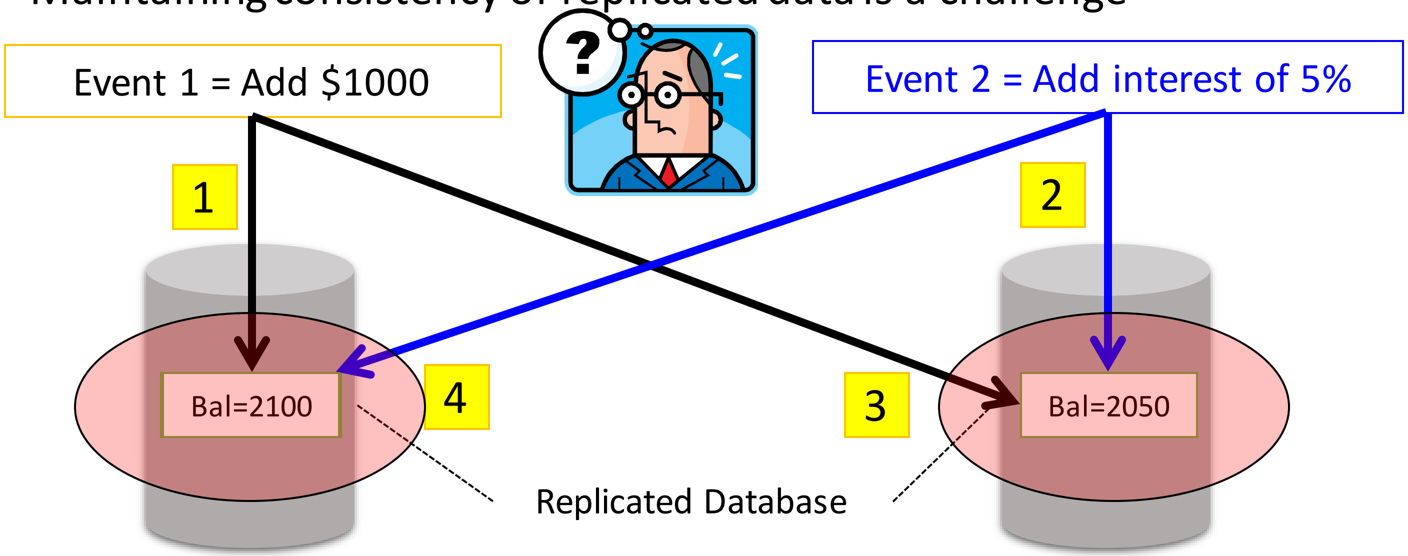


Number of servers with correct data outvote the faulty servers

n = Servers that do not have the requested data
n = Servers with correct data
n = Servers with faulty data

Why Consistency?

- In a DS with replicated data, one of the main problems is keeping the data consistent
- An example:
  - In an e-commerce application, the bank database has been replicated across two servers
  - Maintaining consistency of replicated data is a challenge



Overview of Consistency and Replication

TODAY

- Consistency Models
  - Data-Centric Consistency Models
  - Client-Centric Consistency Models
- Replica Management
  - When, where and by whom replicas should be placed?
  - Which consistency model to use for keeping replicas consistent?
- Consistency Protocols
  - We study various implementations of consistency models

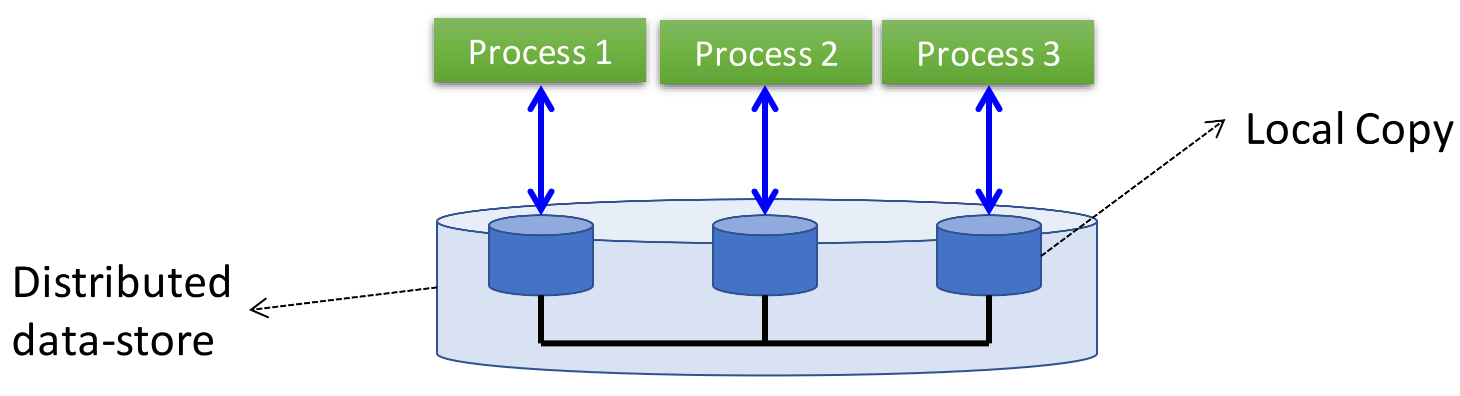
NEXT

Overview

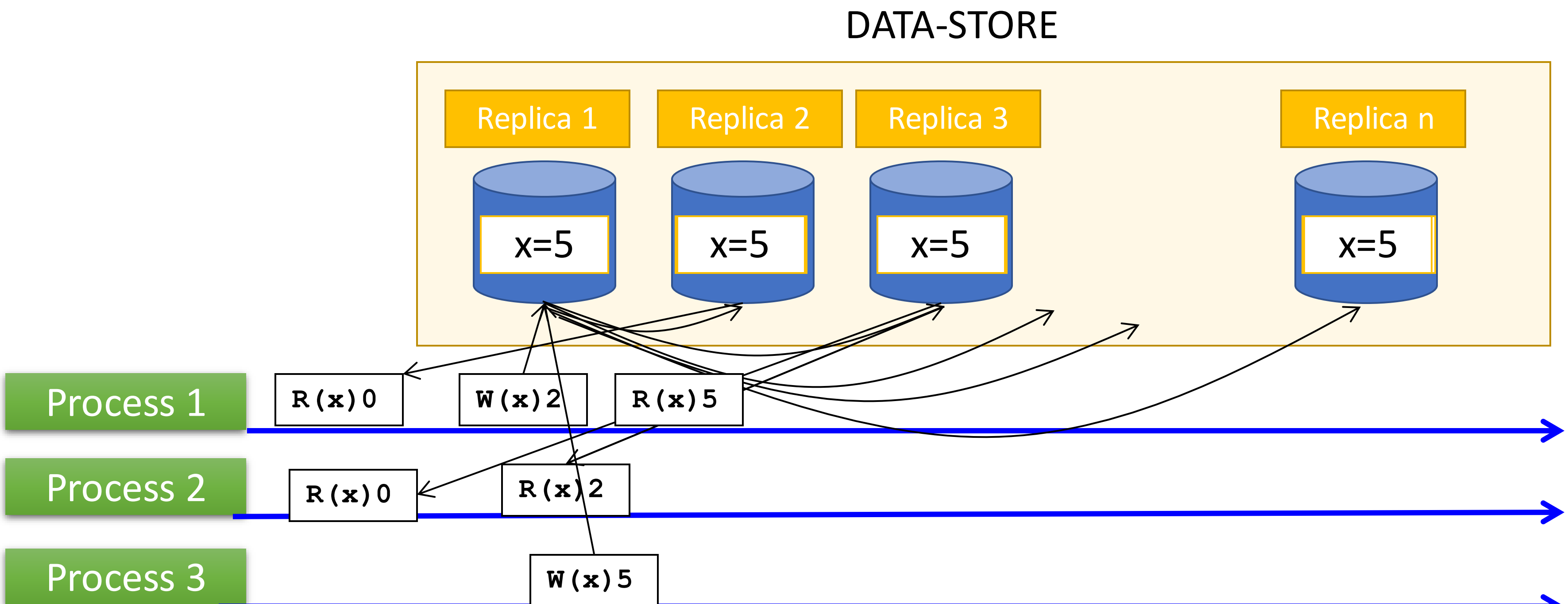
- Consistency Models
  - Data-Centric Consistency Models
  - Client-Centric Consistency Models
- Replica Management
- Consistency Protocols

Introduction to Consistency and Replication

- In a distributed system, shared data is typically stored in distributed shared memory, distributed databases or distributed file systems.
  - The storage can be distributed across multiple computers
  - Simply, we refer to a series of such data storage units as *data-stores*
- Multiple processes can access shared data by accessing any replica on the data-store
  - Processes generally perform read and write operations on the replicas



Maintaining Consistency of Replicated Data

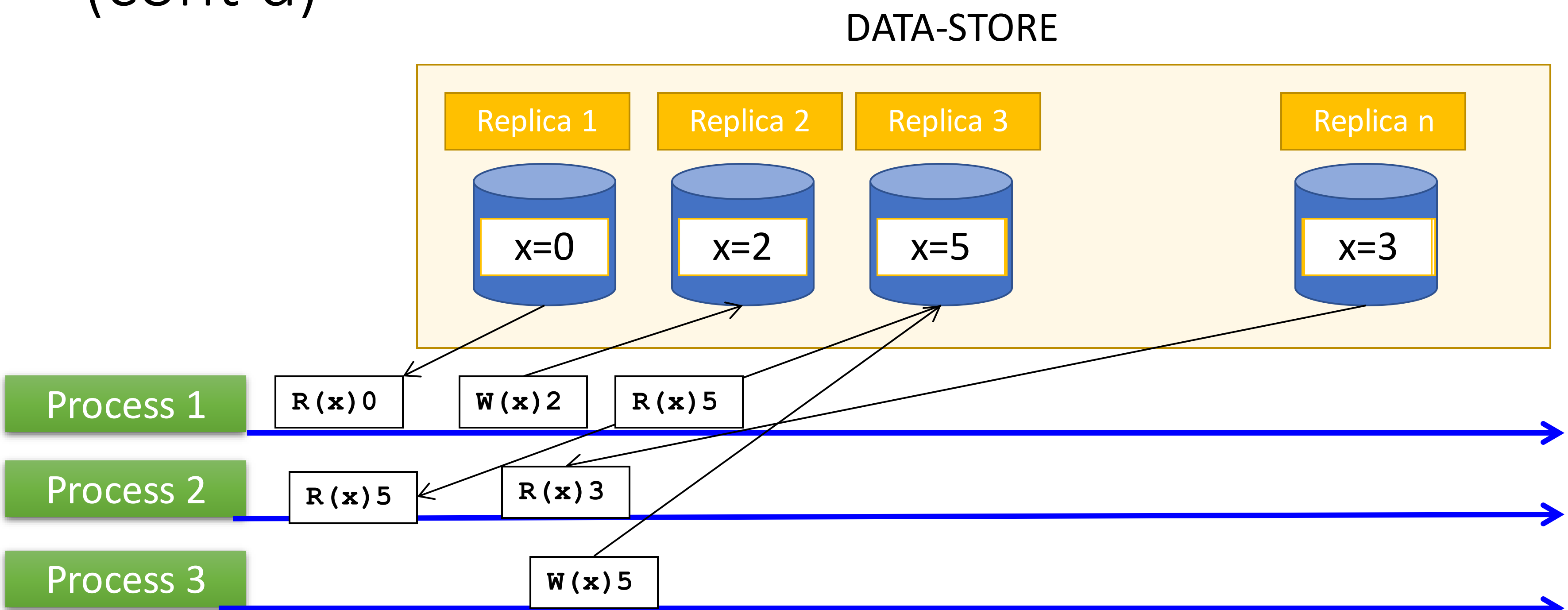


Strict Consistency

- Data is always fresh
  - After a write operation, the update is propagated to all the replicas
  - A read operation will result in reading the most recent write
- If there are occassional writes and reads, this leads to large overheads

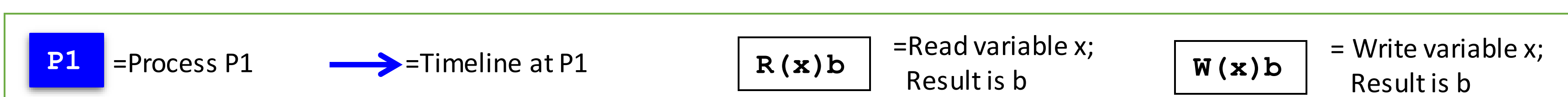
P1 =Process P1 | =Timeline at P1 | R(x)b =Read variable x; Result is b | W(x)b = Write variable x; Result is b

Maintaining Consistency of Replicated Data (cont'd)



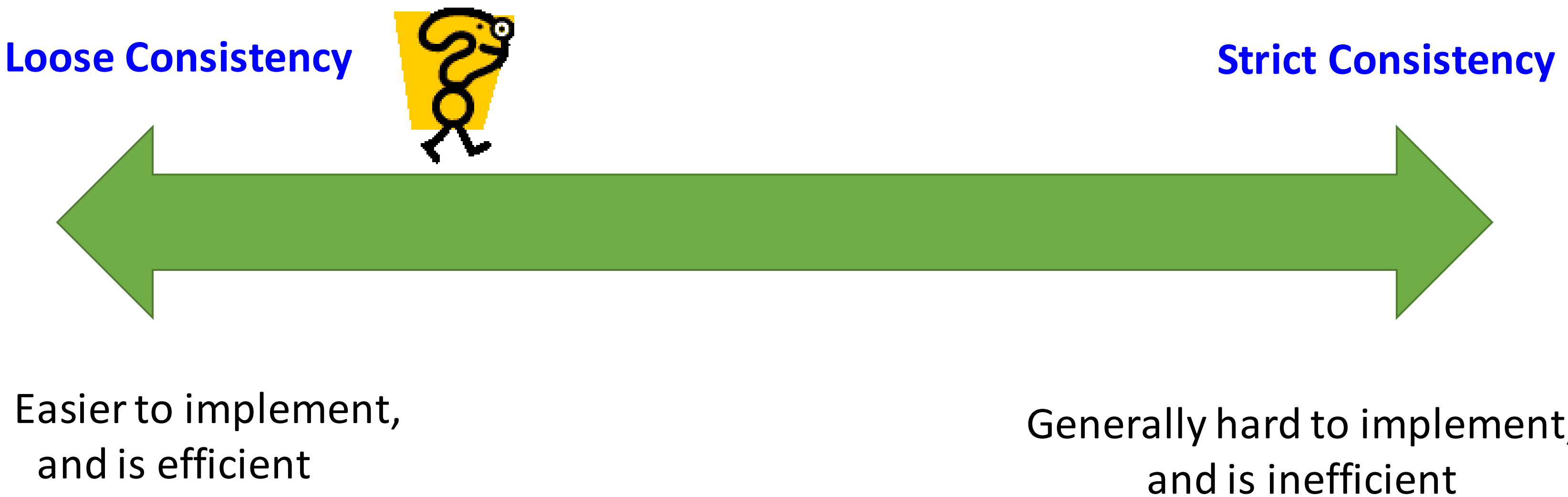
Loose Consistency

- Data might be stale
  - A read operation may result in reading a value that was written long back
  - Replicas are generally out-of-sync
- The replicas may sync at coarse grained time, thus reducing the overhead



Trade-offs in Maintaining Consistency

- Maintaining consistency should balance between the strictness of consistency versus efficiency
  - Good-enough consistency depends on your application



Consistency Model

- A consistency model is a contract between
  - the process that wants to use the data, and
  - the replicated data repository (or data-store)
- A consistency model states the level of consistency provided by the *data-store* to the processes while reading and writing the data

Types of Consistency Models

- Consistency models can be divided into two types:
  - Data-Centric Consistency Models
    - These models define how the data updates are propagated across the replicas to keep them consistent
  - Client-Centric Consistency Models
    - These models assume that clients connect to different replicas at each time
    - The models ensure that whenever a client connects to a replica, the replica is bought up to date with the replica that the client accessed previously

Overview

- Consistency Models
  - Data-Centric Consistency Models
  - Client-Centric Consistency Models
- Replica Management
- Consistency Protocols

Data-centric Consistency Models

- Data-centric Consistency Models describe how the replicated data is kept consistent, and what the process can expect

- Under Data-centric Consistency Models, we study two types of models:
  - Consistency Specification Models:
    - These models enable specifying the consistency levels that are tolerable to the application
  - Models for Consistent Ordering of Operations:
    - These models specify the order in which the data updates are propagated to different replicas

Overview

- Consistency Models
  - Data-Centric Consistency Models
    - Consistency Specification Models
    - Models for Consistent Ordering of Operations
  - Client-Centric Consistency Models
- Replica Management
- Consistency Protocols

Consistency Specification Models

- In replicated data-stores, there should be a mechanism to:
  - Measure how inconsistent the data might be on different replicas
  - How replicas and applications can specify the tolerable inconsistency levels
- Consistency Specification Models enable measuring and specifying the level of inconsistency in a replicated data-store
- We study a Consistency Specification Model called *Continuous Consistency Model*

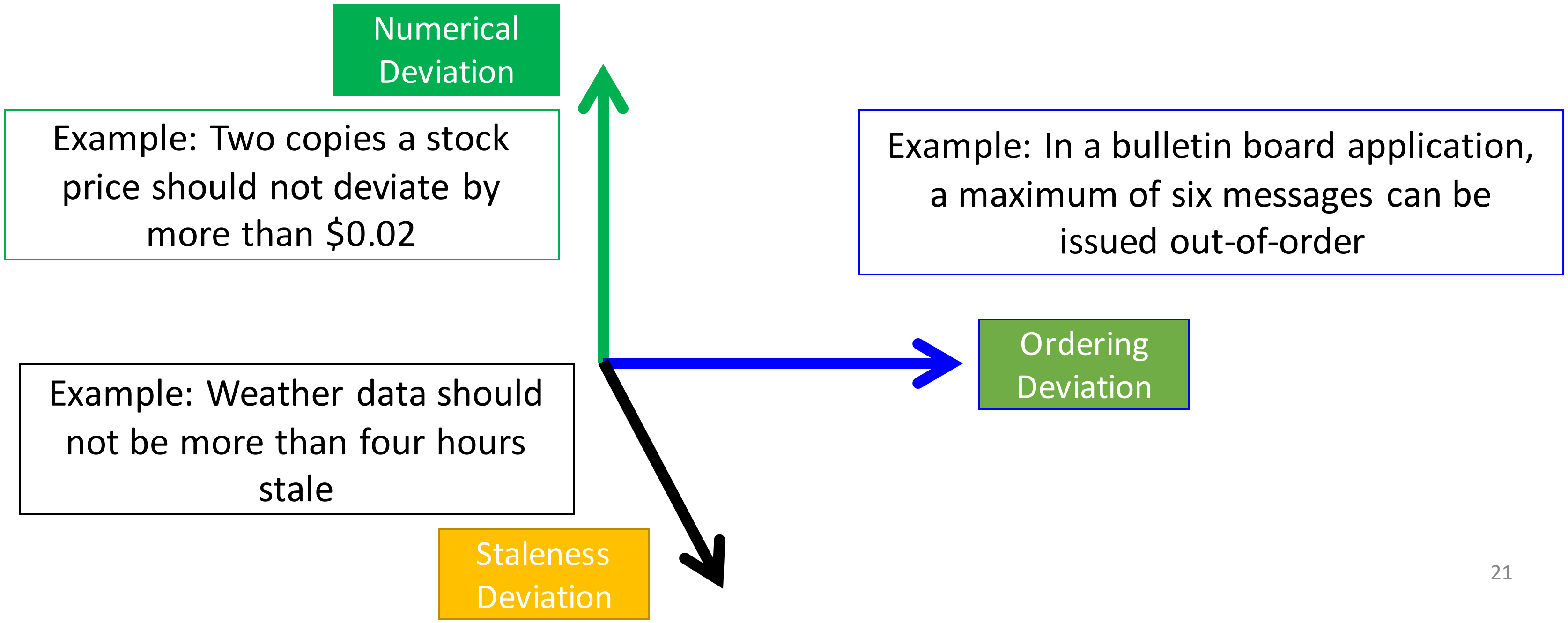
Continuous Consistency Model

- Continuous Consistency Model is used to measure inconsistencies and express what inconsistencies can be expected in the system

- Yu and Vahdat [1] provided a framework for measuring and expressing consistency in replicated data-stores

Continuous Consistency Ranges

- Level of consistency is defined over three independent axes:
  - Numerical Deviation: Deviation in the numerical values between replicas
  - Order Deviation: Deviation with respect to the ordering of update operations
  - Staleness Deviation: Deviation in the staleness between replicas



Consistency Unit (Conit)

- Consistency unit (Conit) specifies the data unit over which consistency is measured
  - For example, conit can be defined as a record representing a single stock

- Level of consistency is measured by each replica along the three dimensions
  - Numerical Deviation
    - For a given replica R, how many updates at other replicas are not yet seen at R? What is the effect of the non-propagated updates on local Conit values?
  - Order Deviation
    - For a given replica R, how many local updates are not propagated to other replicas?
  - Staleness Deviation
    - For a given replica R, how long has it been since updates were propagated?

Example of Conit and Consistency Measures

Order Deviation at a replica R is the number of operations in R that are not present at the other replicas

Numerical Deviation at replica R is defined as n(w), where
n = # of operations at other replicas that are not yet seen by R,
w = weight of the deviation
= max(update amount of all variables in a Conit)

Replica A					Replica B				
x	**y**	**VC**	**Ord**	**Num**	**x**	**y**	**VC**	**Ord**	**Num**
0	0	(0,0)	0	0(0)	0	0	(0,0)	0	0(0)
0	0	(0,0)	0	1(2)	2	0	(0,5)	1	0(0)
2	0	(1,5)	0	0(0)	2	0	(0,5)	0	0(0)
2	1	(10,5)	1	0(0)	2	0	(0,5)	0	1(1)
2	1	(10,5)	1	1(1)	2	1	(0,16)	1	1(1)
3	1	(14,5)	2	1(1)	2	1	(0,16)	1	2(2)
3	4	(23,5)	3	1(1)	2	1	(0,16)	1	3(4)

Replica A

x; y

Operation		Result
<5,B>	x+=2	x=2
<10,A>	y+=1	y=1
<14,A>	x+=1	x=3
<23,A>	y+=3	y=4

Replica B

x; y

Operation		Result
<5,B>	x+=2	x=2
<16,B>	y+=1	y=1

<5,B> = Operation performed at **B** when the vector clock was **5**

<m,n> = Uncommitted operation

<m,n> = Committed operation

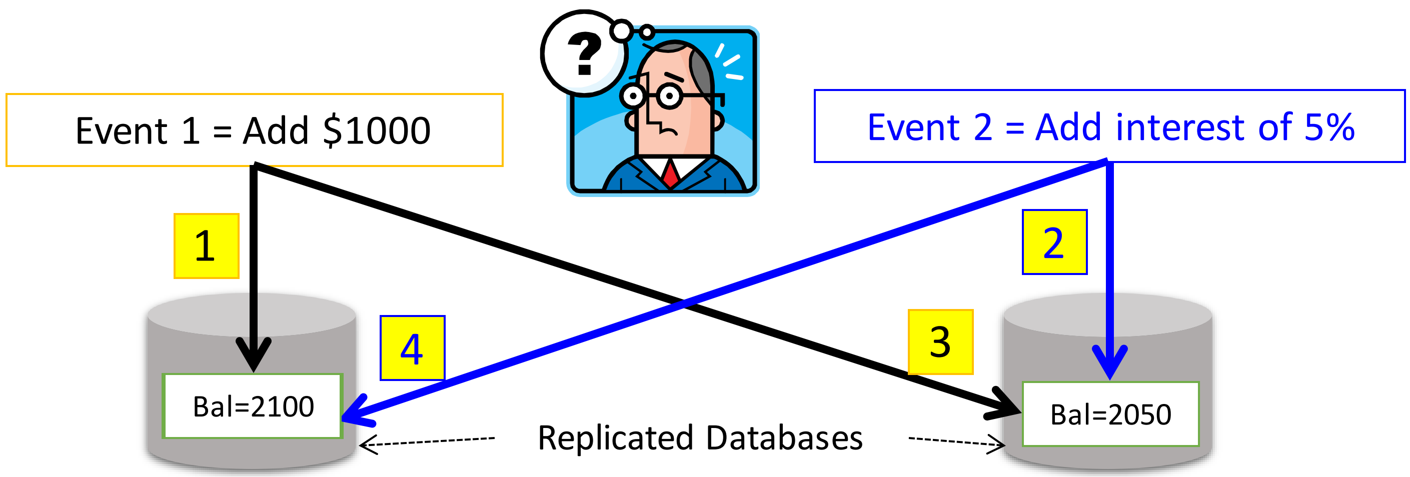
x;y = A Conit

Overview

- Consistency Models
  - Data-Centric Consistency Models
    - Continuous Specification Models
    - Models for Consistent Ordering of Operations
  - Client-Centric Consistency Models
- Replica Management
- Consistency Protocols

Why is Consistent Ordering Required in Replication?

- In several applications, the order or the sequence in which the replicas commit to the data store is critical
- Example:



- Continuous Specification Models defined how inconsistency is measured
  - However, the models did not enforce any order in which the data is committed

Consistent Ordering of Operations (cont’d)

- Whenever a replica is updated, it propagates the updates to other replicas at some point in time

- Updating different replicas is carried out by passing messages between the replica data-stores

- We will study different types of ordering and consistency models arising from these orderings

Types of Ordering

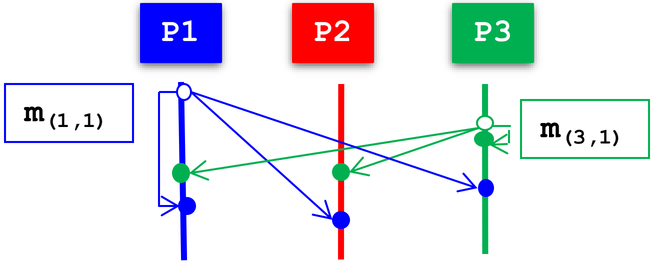
- We will study three types of ordering of messages that meet the needs of different applications:
  1. Total Ordering
  2. Sequential Ordering
     i. Sequential Consistency Model
  3. Causal Ordering
     i. Causal Consistency Model

Types of Ordering

1. Total Ordering
2. Sequential Ordering
3. Causal Ordering

Total Ordering

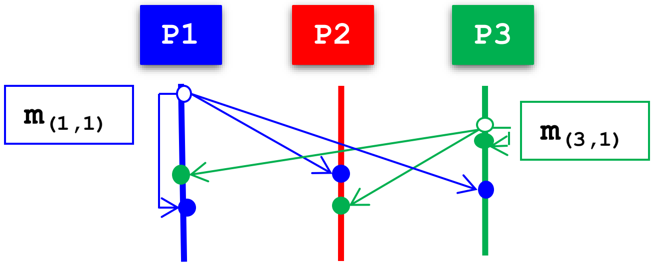
- Total Order
  - If process $\mathbf{P_i}$ sends a message $\mathbf{m_i}$ and $\mathbf{P_j}$ sends $\mathbf{m_j}$, and if one correct process delivers $\mathbf{m_i}$ before $\mathbf{m_j}$ then every correct process delivers $\mathbf{m_i}$ before $\mathbf{m_j}$



Ex1: Total Order

- Messages can contain replica updates, such as passing the read or write operation that needs to be performed at each replica
  - In the example Ex1, if $\mathbf{P_1}$ issues the operation $\mathbf{m_{(1,1)}}$**: x=x+1;** and
  - If $\mathbf{P_3}$ issues $\mathbf{m_{(3,1)}}$**: print(x);**
  - Then, at all replicas $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ the following order of operations are executed

```
print(x);
x=x+1;
```



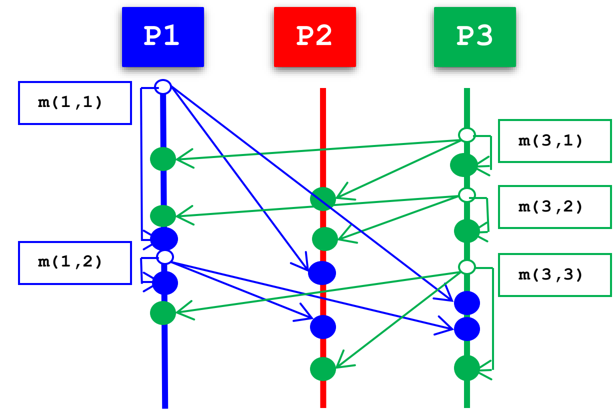
Ex2: Not in Total Order

Types of Ordering

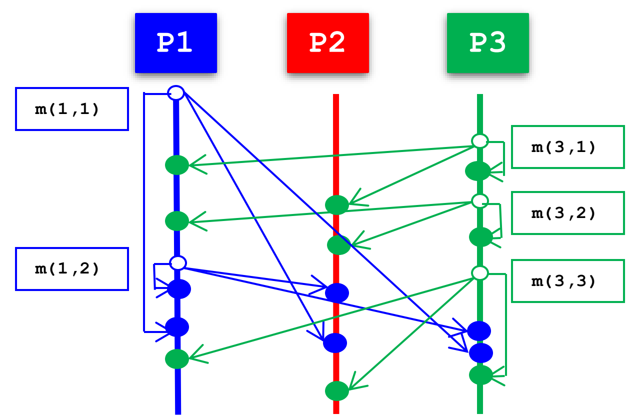
1. Total Ordering
2. Sequential Ordering
3. Causal Ordering

Sequential Ordering

+ If a process **Pi** sends a sequence of messages $\mathbf{m}_{(i,1)}, \ldots, \mathbf{m}_{(i,ni)}$, and
+ Process **Pj** sends a sequence of messages $\mathbf{m}_{(j,1)}, \ldots, \mathbf{m}_{(j,nj)}$,
+ Then, :
  + At any process, the set of messages received are in some sequential order
  + Messages from each individual process appear in this sequence in the order sent by the sender
    + At every process, $\mathbf{m}_{i,1}$ should be delivered before $\mathbf{m}_{i,2}$, which is delivered before $\mathbf{m}_{i,3}$ and so on...
    + At every process, $\mathbf{m}_{j,1}$ should be delivered before $\mathbf{m}_{j,2}$, which is delivered before $\mathbf{m}_{j,3}$ and so on...



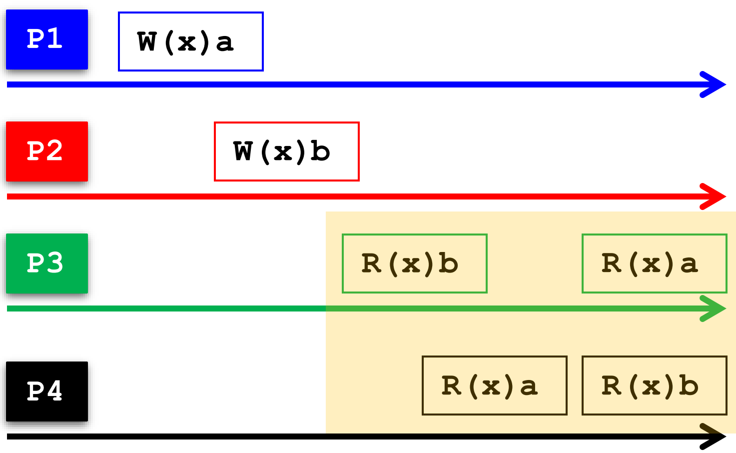
Valid Sequential Orders



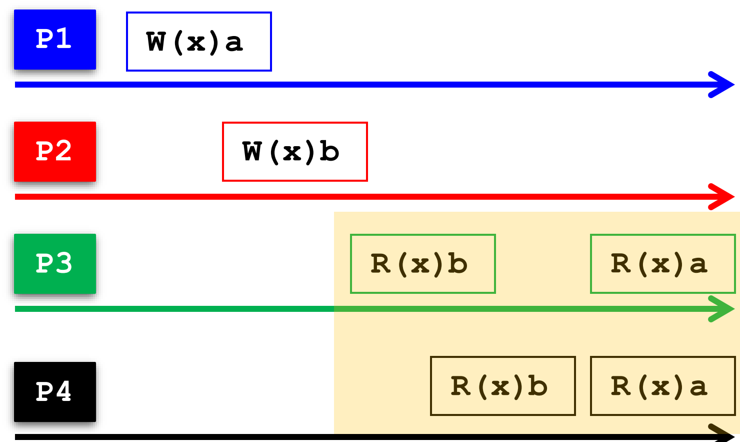
Invalid Sequential Orders, but Valid Total Order

Sequential Consistency Model

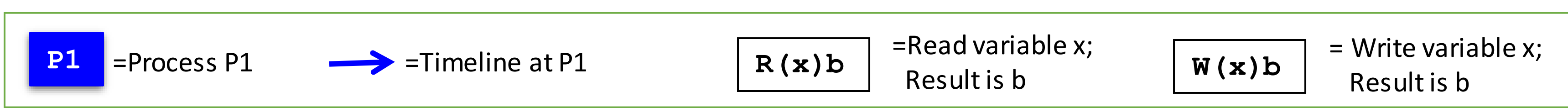
- Sequential Consistency Model enforces that all the update operations are executed at the replicas in a sequential order
- Consider a data-store with variable **x** (Initialized to **NULL**)
  - In the two data-stores below, identify the sequentially consistent data-store



Results while operating on DATA-STORE-1



( ) Results while operating on DATA-STORE-2



Sequential Consistency (cont'd)

- Consider three processes P_1, P_2 and P_3 executing multiple instructions on three shared variables **x**, **y** and **z**
  - Assume that **x**, **y** and **z** are set to zero at start

P1
```
x = 1
print (y,z)
```

P2
```
y = 1
print (x,z)
```

P3
```
z = 1
print (x,y)
```

- There are many valid sequences in which operations can be executed at the replica respecting sequential consistency
  - Identify the output

	x = 1 print (y,z) y = 1 print (x,z) z = 1 print (x,y)	x = 1 y = 1 print (x,z) print (y,z) z = 1 print (x,y)	z = 1 print (x,y) print (x,z) y = 1 x = 1 print (y,z)	y = 1 z = 1 print (x,y) print (x,z) x = 1 print (y,z)
Output	001011	101011	000111	010111

Implications of Adopting Sequential Consistency Model for Applications

- There might be several different sequentially consistent combinations of ordering
  - Number of combinations for a total of n instructions = $O(n!)$
- The contract between the process and the distributed data-store is that the process must accept all of the sequential orderings as valid results
  - A process that works for some of the sequential orderings and does not work correctly for others is INCORRECT

Summary

- Replication is necessary for improving performance, scalability and availability, and for providing fault-tolerance
- Replicated data-stores should be designed after carefully evaluating the trade-off between tolerable data inconsistency and efficiency
- Consistency Models describe the contract between the data-store and process about what form of consistency to expect from the system
- Data-centric consistency models:
  - Continuous Consistency Models provide mechanisms to measure and specify inconsistencies
  - Consistency Models can be defined based on the type of ordering of operations that the replica guarantees the applications
    - We studied Sequential Consistency Model

Next …

- Consistency Models
  - Data-Centric Consistency Model: Causal Consistency Model
  - Client-Centric Consistency Models

- Replica Management
  - Replica management studies:
    - when, where and by whom replicas should be placed
    - which consistency model to use for keeping replicas consistent

- Consistency Protocols
  - We study various implementations of consistency models

References

- [1] Haifeng Yu and Amin Vahdat, "Design and evaluation of a conit-based continuous consistency model for replicated services"
- [2] http://tech.amikelive.com/node-285/using-content-delivery-networks-cdn-to-speed-up-content-load-on-the-web/
- [3] http://en.wikipedia.org/wiki/Replication_(computer_science)
- [4] http://en.wikipedia.org/wiki/Content_delivery_network
- [5] http://www.cdk5.net
- [6] http://www.dis.uniroma1.it/~baldoni/ordered%2520communication%25202008.ppt
- [7] http://www.cs.uiuc.edu/class/fa09/cs425/L5tmp.ppt