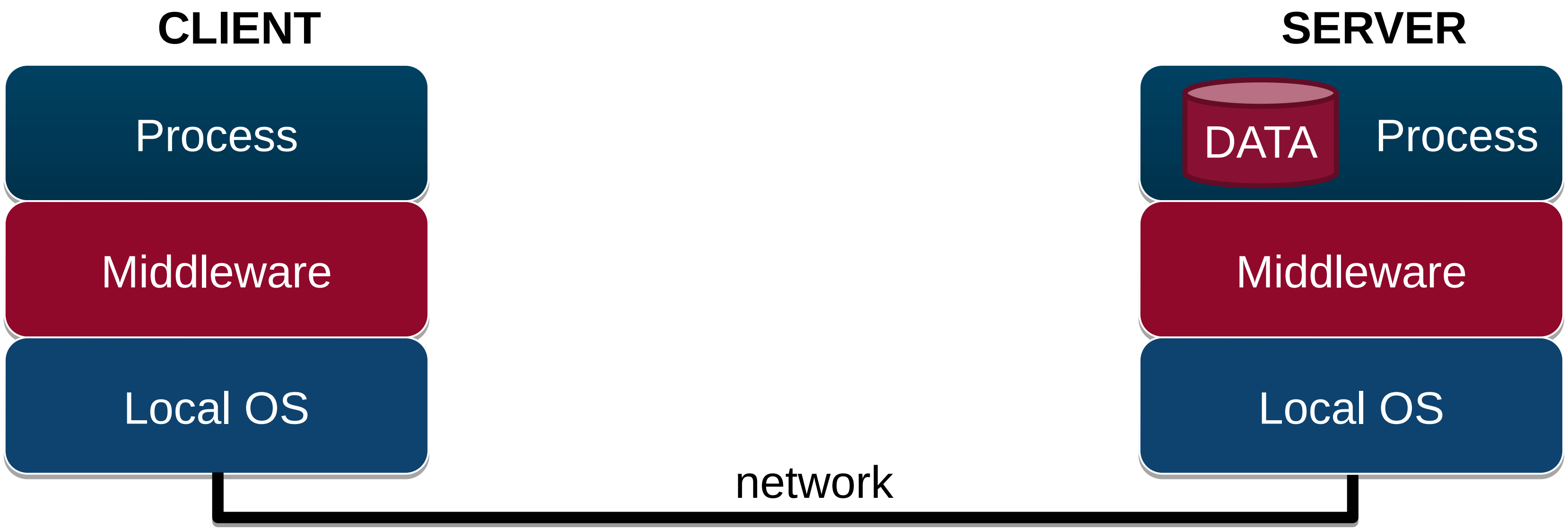
Replication

IDLab
INTERNET & DATA LAB

imec

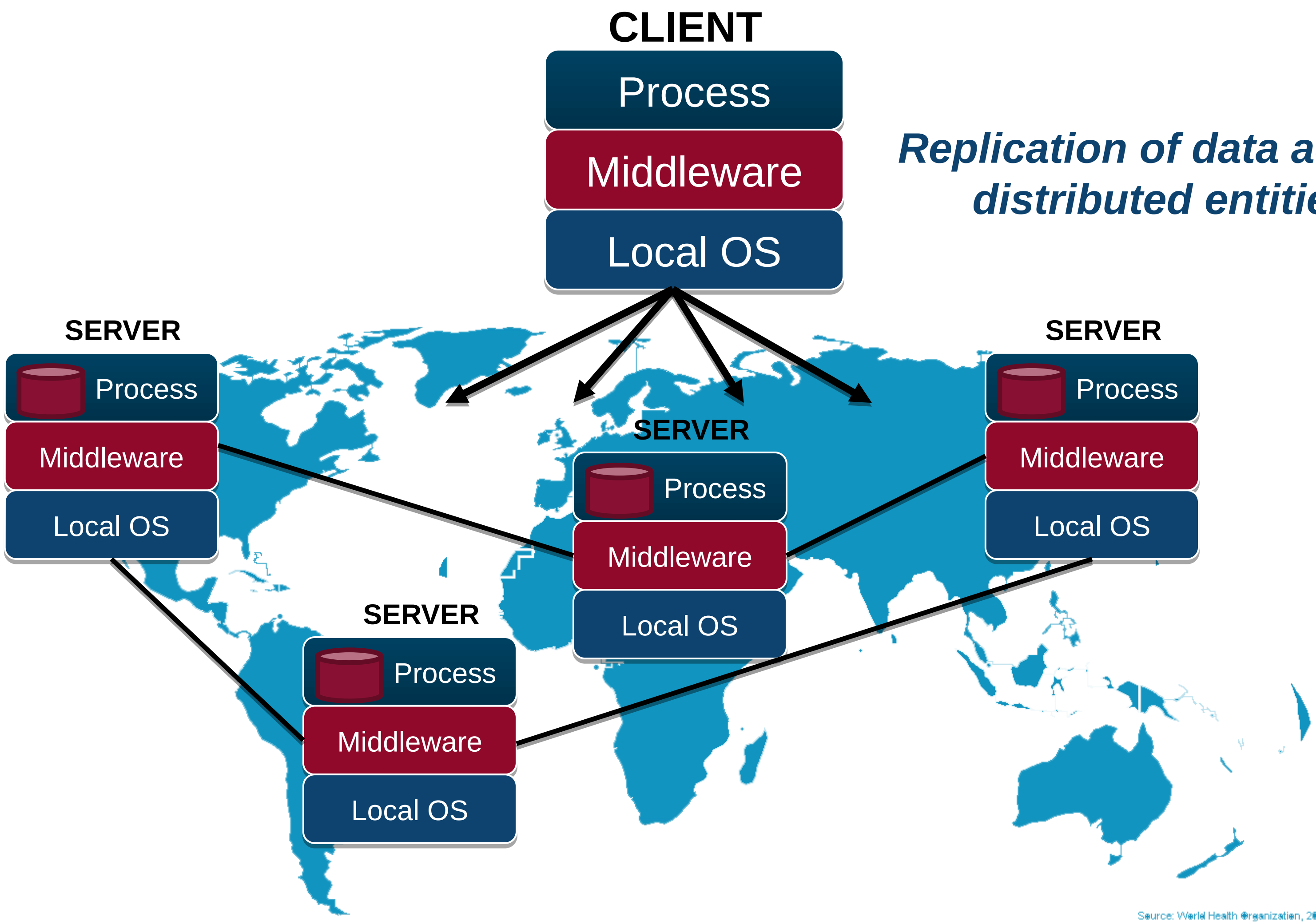

Deckare & thrillers

Replication in distributed systems



1 on 1 mapping between location and data

Replication in distributed systems



Replication of data across distributed entities

1 Why replication?

2 Replication challenges

3 Consistency models

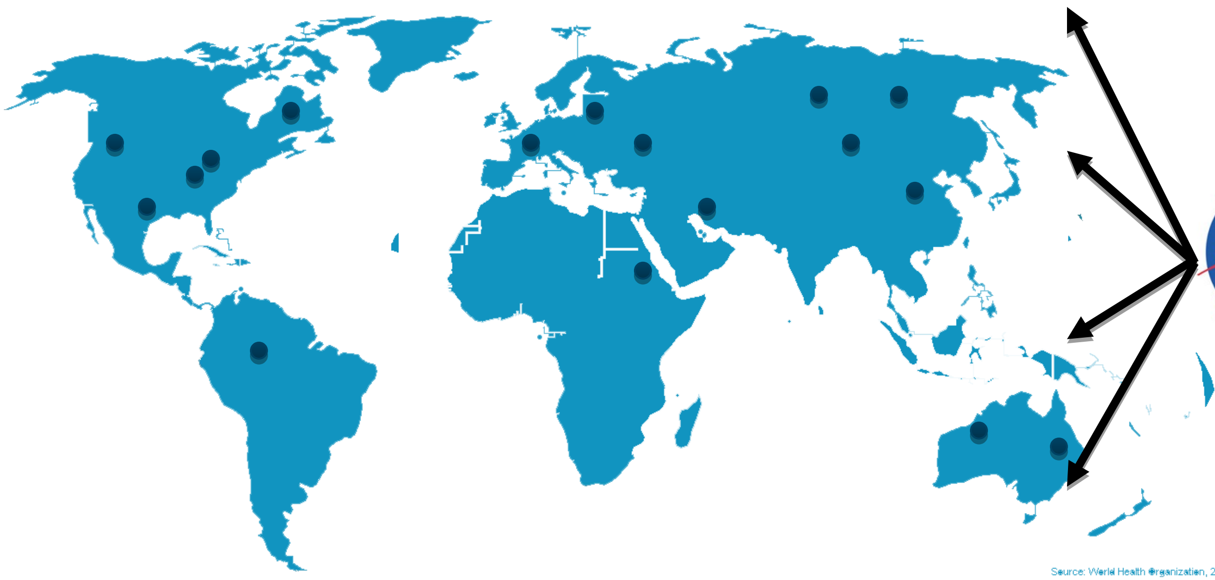
Why replication?

1 Improving performance

- Balancing the load between servers
- Geographical spreading to reduce the latency

Example: Akamai

100.000 - 250.000
content servers across the world



Why replication?

1 **Improving performance**

2 **Increased availability**

p probability of failure of one server

$1 - p^n$ probability that system with n replicas will still be available if replicates are independent

p= 0.10

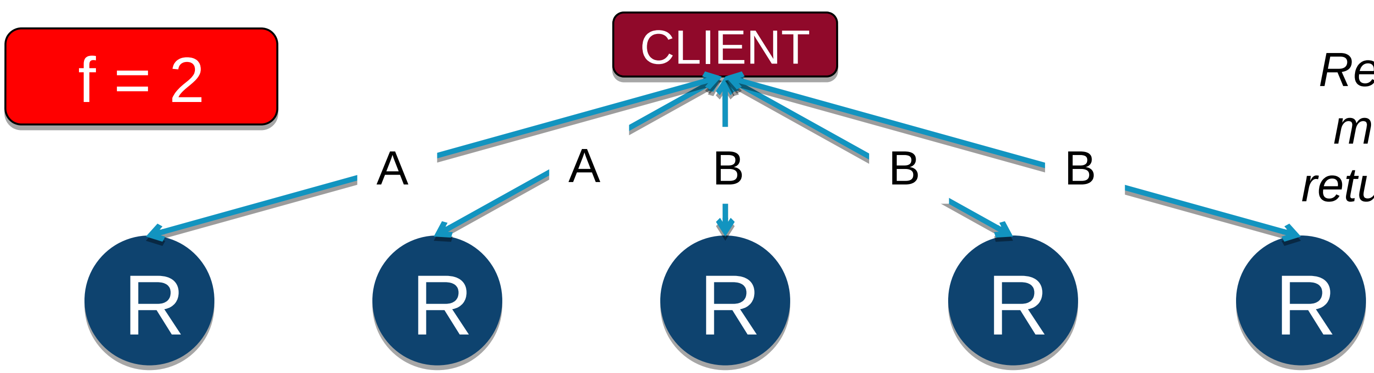
Number of replicas (n)	Availability probability ($1-p^n$)
1	90%
2	99%
3	99,9%

Why replication?

1 **Improving performance**

2 **Increased availability**

3 **Data corruption protection (fault-tolerance)**



Result is accepted if majority of replicas return the same value

- Replication can protect against corrupt data
- How? voting system: **2f +1** replicas can deal with **f** corrupt replicas

2

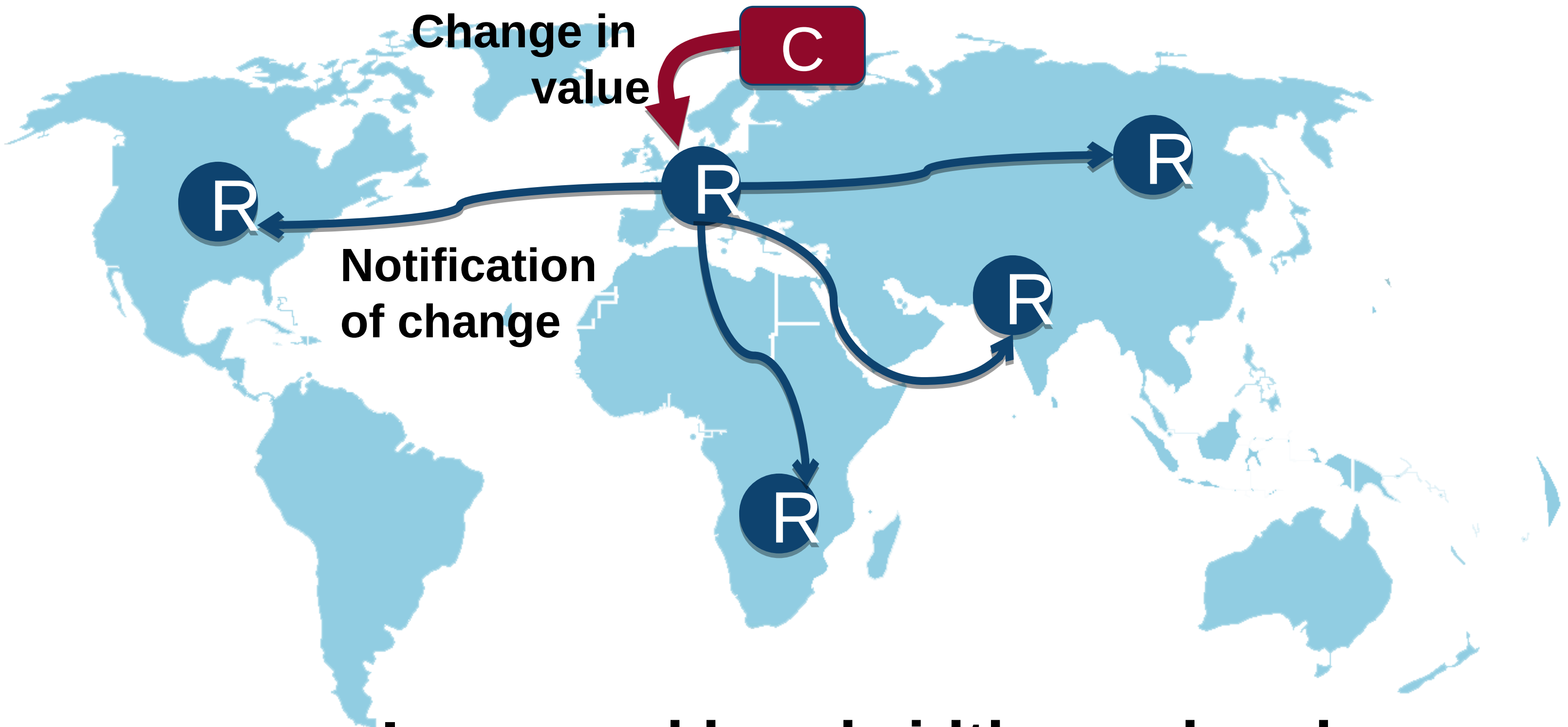
Replication challenges

Replication has it's advantages

But is it always that good?

Downside of replication

data must be kept consistent between replicas



Increased bandwidth overhead

CAP Theorem

aka Brewer theorem

Named after Eric Brewer
Professor at UC Berkeley
Presented in 2000, proven in 2002

Consistency — all nodes always see the same data

Availability — every request receives a response

Partition-tolerance — guarantees remain even when network failures prevent communication

“it is impossible for a distributed computer system to simultaneously provide all three of these guarantees”

Impact of CAP theorem in practice

Optimize consistency at the cost of availability
OR optimize availability at the cost of consistency

Google App Engine's Database System

High Replication Datastore

"we've been struggling with some **reliability issues** with the App Engine Datastore

…

I'm proud to announce the availability of a **new Datastore** configuration option

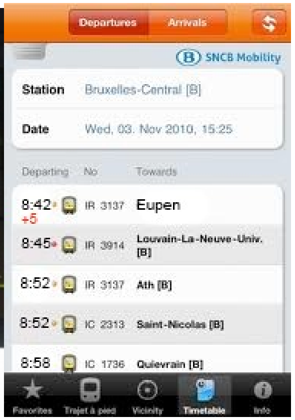
…

provides the **highest level of availability** at the **cost** of increased latency and **changes in consistency guarantees**"

Can we loosen consistency?

Different applications,
provide different consistency guarantees

Financial application
(e.g., on-line banking)



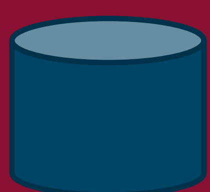
Train Information
Application

Strong consistency guarantees → Weak consistency guarantees

3 CONSISTENCY MODELS

- Clear need for loosening the consistency guarantees
- How? Through consistency models

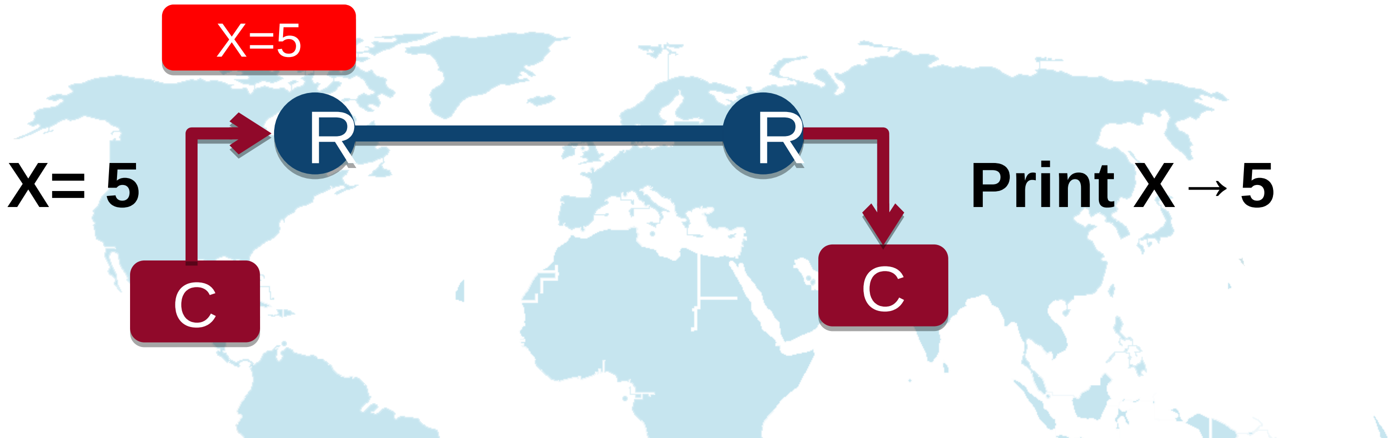
Data-Centric Consistency Models



VS

Client-Centric Consistency Models

Data-centric consistency model



Consistency model = contract between client processes (C) and datastore of replicas (R)

- If client processes follow rules in the contract…
- …the datastore will work correctly & results will be predictable
- Rules deal with **how parallel updates are received by other client processes**

Strict consistency

Identical to single machine execution

Strongest consistency guarantees

Immediate propagation of all content updates

Absolute **time ordering** of all shared access operations (global clock)

A **read returns the last write** given that time ordering

Example: money transfer within one bank
(assume `account` is initially empty)

Client 1

`account` = 100 €

time

time

Client 2

`account` =
`account` + 200 €

Value of `account`?

(1) IF 300
→ Strictly consistent

(2) IF other value
→ Not strictly consistent

Sequential consistency

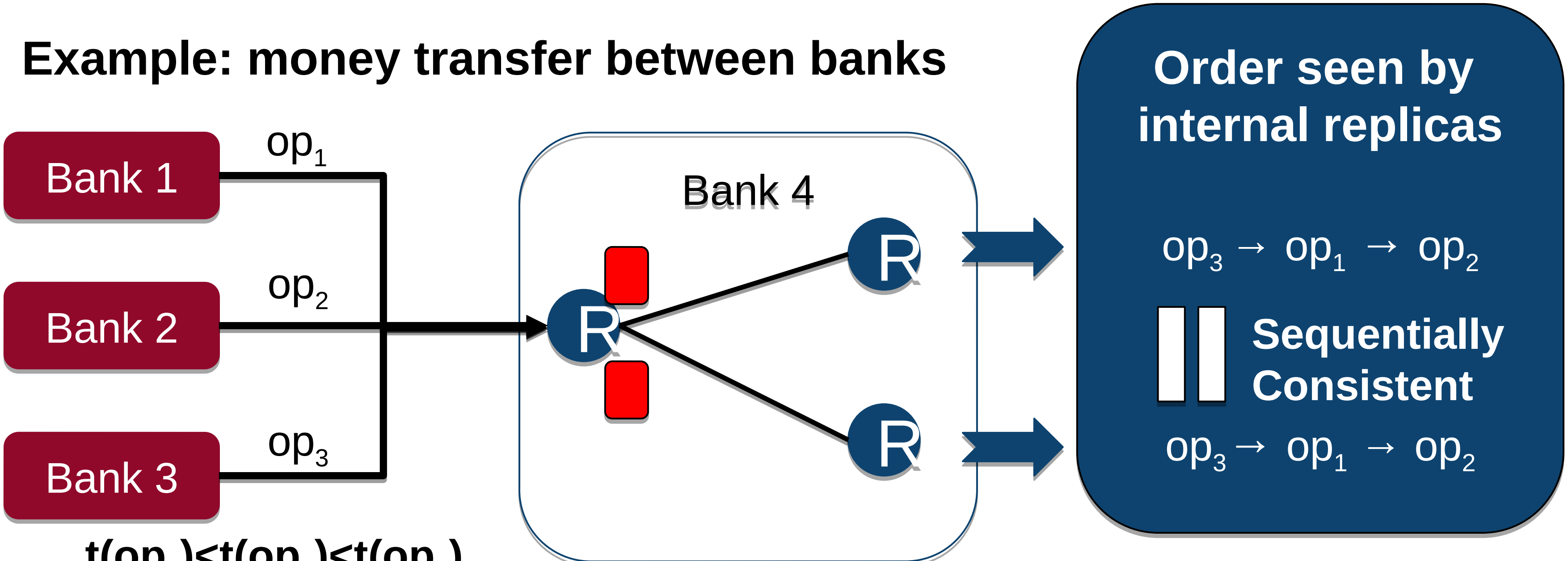
Relaxation: delay can exist in content updates

Weaker consistency guarantees than strict consistency

Operations are interleaved in some fixed **order**

All processes see the **same order**

Example: money transfer between banks



$t(op_1)<t(op_2)<t(op_3)$

Some notation

W(x)a → x = a

R(x)a → print x → a

Exercise

What Sequential order can explain these results?

P1 W(x)a

P2 W(x)b

P3 R(x)b R(x)b

P4 R(x)b R(x)b

Global clock ordering
→ strictly consistent

Note: operations of each individual process appear in this sequence in the order specified by its program

Exercise

What Sequential order can explain these results?

P1	W(x)a				
P2		W(x)b			
P3			R(x)a	R(x)b	
P4			R(x)b		R(x)b

W(x)a , **R(x)a** , W(x)b ,R(x)b

Sequentially consistent

Note: operations of each individual process appear in this sequence in the order specified by its program

Exercise: Sequential consistency

Fill in the question marks:

What should the read operations return to be sequentially consistent?

P1

x = 3

x = 8

y = 16

P2

x = 12

y = 10

P3

Print x → 12

z = 2

Print y → 16

Print y → 10

Print x → 3

z=5

Print x → 8

P4

Print x → 12

Print y → ???

Print x → 8

Print x → ???

Exercise

What Sequential order can explain these results?

P1	W(x)a				
P2		W(x)b			
P3			R(x)b	R(x)a	
P4			R(x)a		R(x)b

Not strictly consistent
Not sequentially consistent

Note: operations of each individual process appear in this sequence in the order specified by its program

Causal consistency (1)

Causal relatedness

Does the operation potentially depend on another operation?

If not causally related → concurrent

Causally related?

Earlier statements

	Read Y	Y = 10	Read X	X = 20	X = X + Y
X = 5	✗	✗	✗	✗	✗
X= B	✓	✓	✓	✓	✓
Read X	✗	✗	✗	✓	✓

Causal consistency (2)

Causal consistent writes are seen by all processes in the **same order**

Concurrent writes can be seen in **any** order

Exercise

What Sequential order can explain these results?

P1	W(x)a			W(x)c	
P2		R(x)a	W(x)b		
P3		R(x)a		R(x)b	R(x)c
P4		R(x)a		R(x)c	R(x)b

Note: operations of each individual process appear in this sequence in the order specified by its program

Exercise

What Sequential order can explain these results?

P1	W(x)a			W(x)c		
P2		R(x)a	W(x)b			
P3		**R(x)a**			**R(x)b**	**R(x)c**
P4		**R(x)a**			**R(x)c**	**R(x)b**

- Not strictly consistent
- Not sequentially consistent

Note: operations of each individual process appear in this sequence in the order specified by its program

Exercise

What Sequential order can explain these results?

P1	W(x)a			W(x)c		
P2		R(x)a	W(x)b			
P3		R(x)a			R(x)b	R(x)c
P4		R(x)a			R(x)c	R(x)b

- Not strictly consistent
- Not sequentially consistent
- W(x)a and W(x)b are causally related
- W(x)b and W(x)c are concurrent

Note: operations of each individual process appear in this sequence in the order specified by its program

PRAM / FIFO Consistency

Pipelined **R**andom **A**ccess **M**emory consistency
→ Writes from a **single** process arrive in the **same** order
→ Writes from multiple processes can arrive in any order

PRAM consistent

P1	W(x)a					
P2		R(x)a	W(x)b	W(x)c		
P3				R(x)b	R(x)a	R(x)c
P4				R(x)a	R(x)b	R(x)c

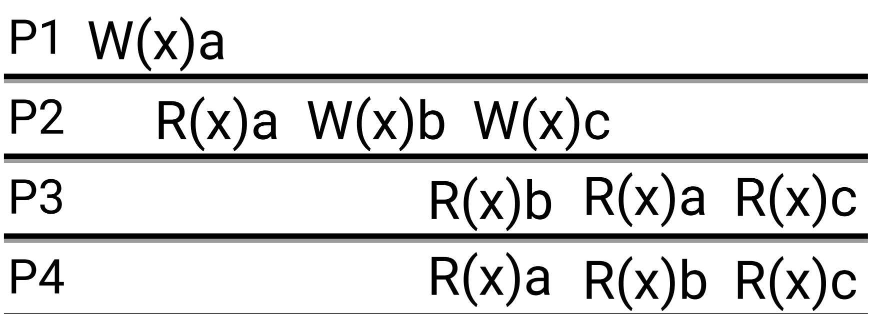
PRAM / FIFO Consistency

Pipelined **R**andom **A**ccess **M**emory consistency

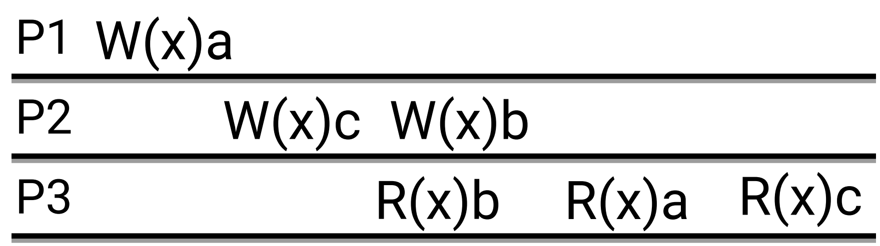
→ **Writes** from a **single** process arrive in the **same** order

→ **Writes** from multiple processes can arrive in any order

PRAM consistent



Not PRAM consistent



Break

Consistency & critical sections

Consistency models seen so far, feature per operation consistency

Read X
Write Y
Write X
Read Y

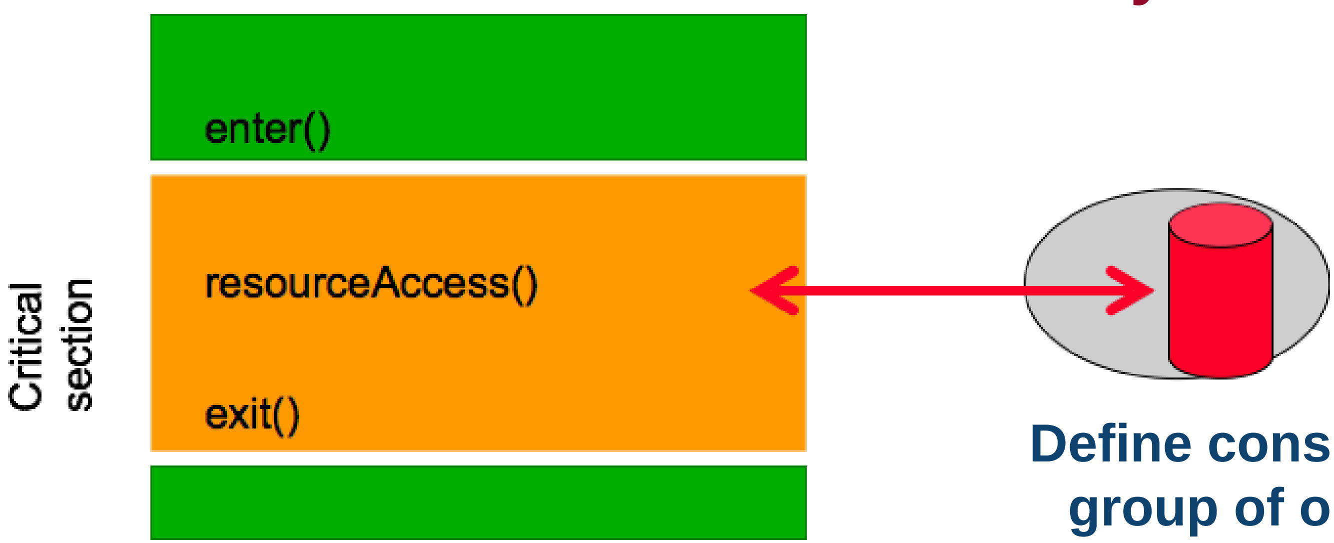
Consistent during every operation

Causal consistency

FIFO consistency

Sequential consistency

There are another set of consistency models



Define consistency on group of operations

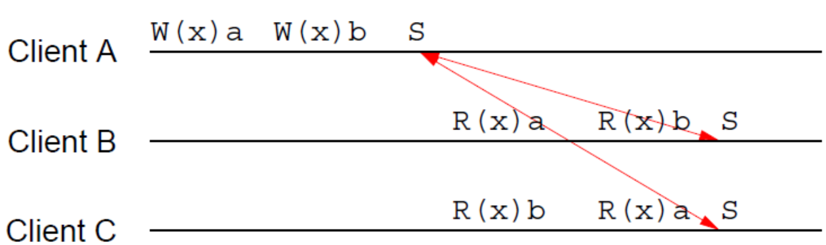
Weak consistency

Group operations to increase granularity of synchronization

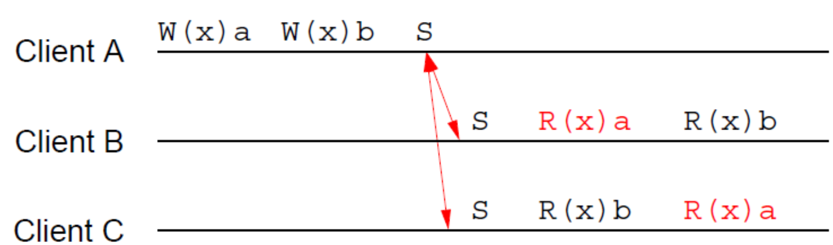
- Use of global synchronization variable S and `synchronize` operation
- Properties
  1. `synchronize` cannot be performed until all previous writes have completed everywhere
  2. `Read` or `write` operations cannot be performed until all previous synchronize operations have completed
  3. The **order of `synchronize` operations is sequentially consistent**

Example

- `synchronize(S) W(x)a W(y)b W(x)c synchronize(S)`
- Writes are performed locally, updates only propagated upon synchronization
- Only `W(y)b W(x)c` are propagated



weak consistent



not weak consistent

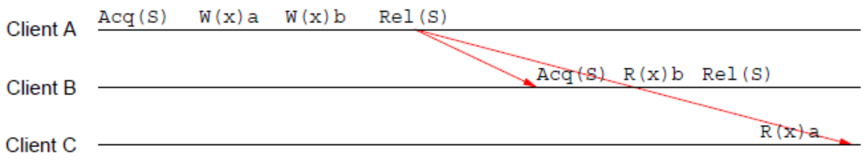
Release Consistency

Explicit synchronization operations defining critical section

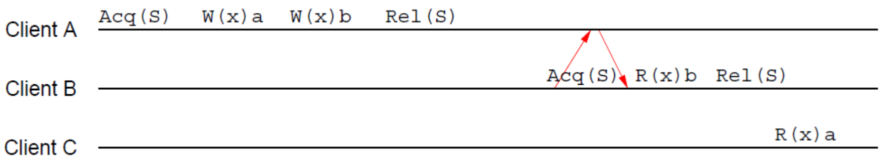
- `acquire(S)` : bring local state up to date (local updates can be propagated later)
- `release(S)` : propagate local updates (remote updates can be propagated later)
- Properties
  1. `release` cannot be performed until previous reads and writes done by the process have completed.
  2. `Read` or `write` operations cannot be performed until all previous `acquire` operations done by the process have performed
  3. The **order of synchronization operations is FIFO consistent**

Lazy Release Consistency

- `acquire` fetches newest state
- Do not send updates on `release`
- Efficiency gain if `acquire/release` done by same client



release consistent



lazy release consistent

Entry Consistency

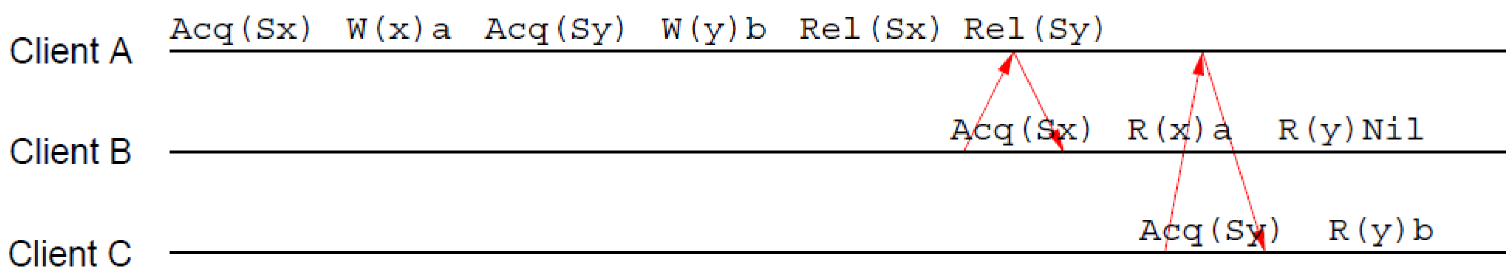
Associate specific data items (guarded data) to synchronization variables

`acquire(S)`

- Obtain exclusive (write) or non-exclusive (read) access to the associated data
- **Synchronize** by fetching associated data from the variables owner (the last client that obtained exclusive access).
- Does not complete until guarded data is made up to date locally
- Exclusive access of a client precludes any other client from accessing guarded data (no process may hold synchronization variable even in non-excl. mode)

`release(S)`

- Relinquish exclusive access



entry consistent

Type of consistency models

- **Strict**
- **Sequential**
- **Causal**
- **PRAM / FIFO**
- **Weak**
- **Release**
- **Entry**

Use synchronization operations on data

Where it is used...

NoSQL: Not-Only SQL or Non-Relational SQL

- New generation of databases
- Increasing availability
- Often distributed
- Looser consistency models

Different products, with different consistency models

- Strict consistency
- Sequential consistency

Consistency models

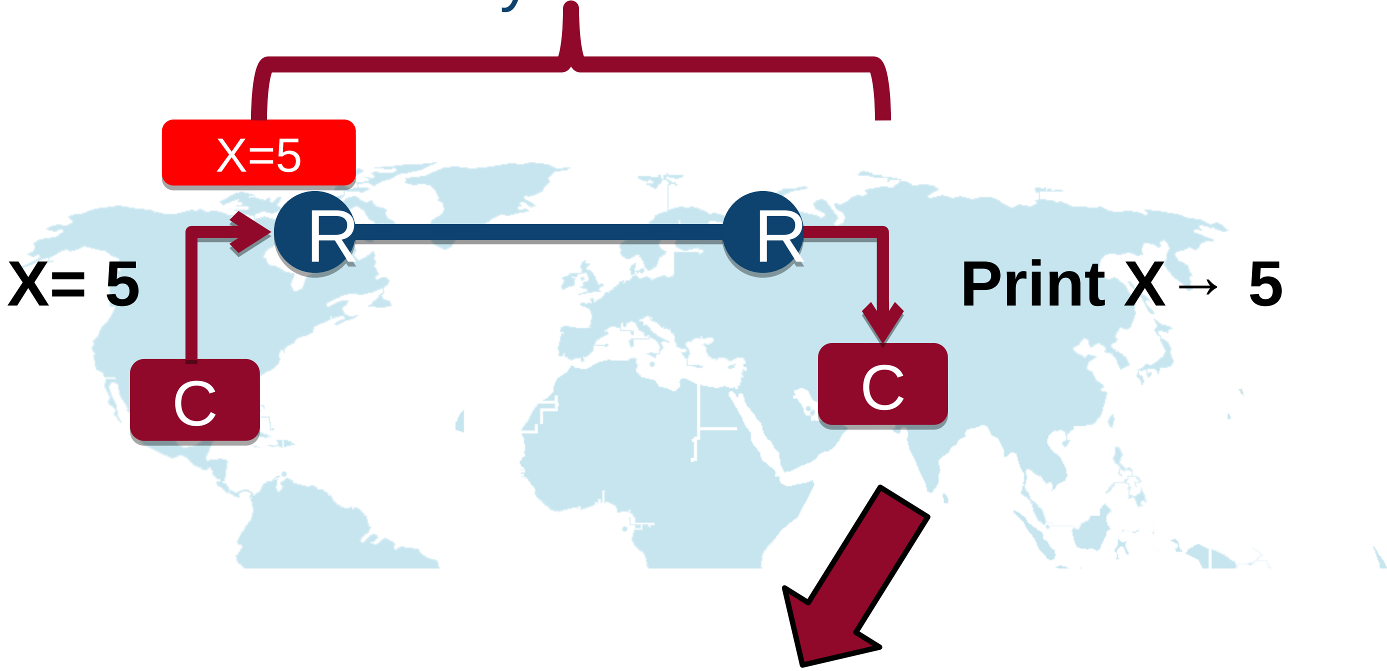
Data-Centric Consistency Models

vs

Client-Centric Consistency Models

Client-centric consistency models

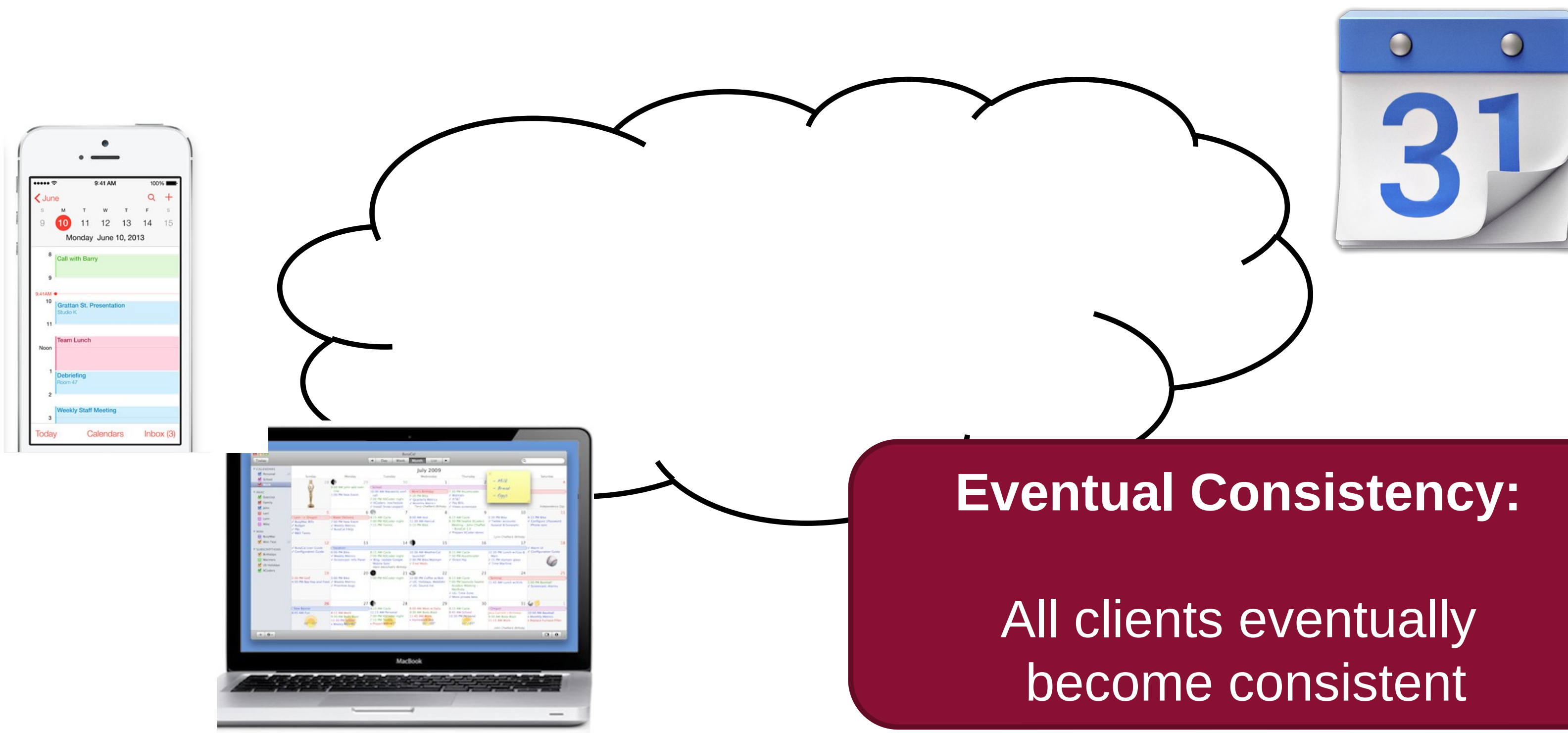
Data-centric consistency models:
Focus on consistency of distributed data store



Client-centric consistency models:
Focus on how the users see the data

Why client centric consistency models?

- Number of reads >> number of writes
- No write-write conflicts or easy to resolve
- Data items have an owner
- Suitable for environments with unreliable network connectivity and limited network performance



Eventual Consistency:

All clients eventually become consistent

How Facebook does it…

Wall posts

Eventual consistency

Messages

Until 2010

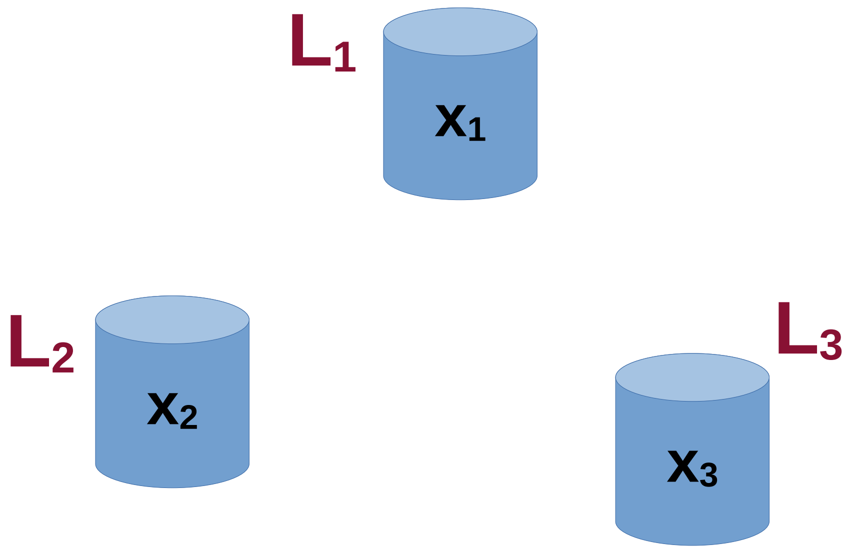
- Eventual consistency
- Using Apache Cassandra
- Developed by Facebook

From 2010 on

- Stronger consistency
- Using Apache HBase

Notation

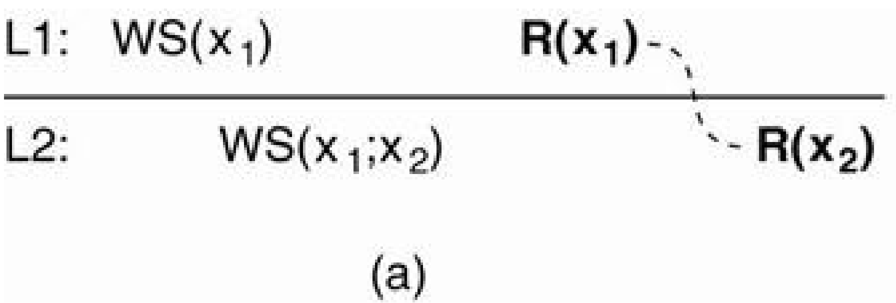
- x_i denotes the version of data item **x** at local copy L_i
- **WS(x_i)** is the set of write operations at L_i that lead to version x_i of **x**
- If operations in **WS(x_i)** have also been performed at local copy L_j, we write **WS(xi, xj)**.



Monotonic-Read Consistency

Definition: *If a process reads the value of a data item* ***x****, any successive read operation on* ***x*** *by that process will always return that* ***same or a more recent value.***

Intuition: Client **“sees”** only same or newer version of data.

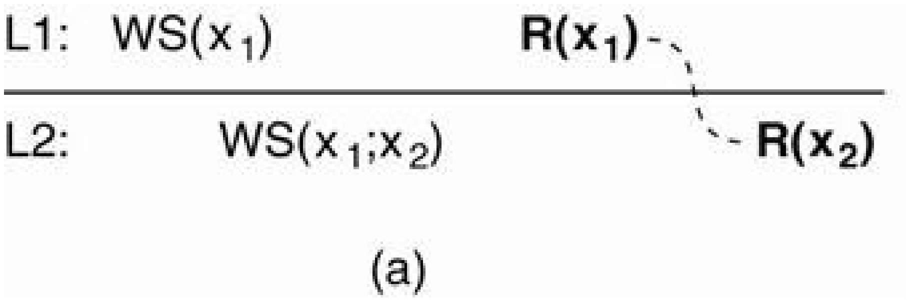


Monotonic Reads

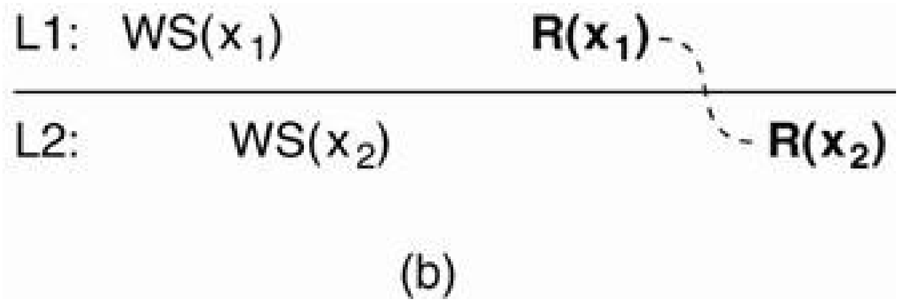
Monotonic-Read Consistency

Definition: *If a process reads the value of a data item **x**, any successive read operation on **x** by that process will always return that **same or a more recent value**.*

Intuition: Client **"sees"** only same or newer version of data.



Monotonic Reads



No Monotonic Reads

Monotonic reads – Examples

1

Automatically reading your personal calendar updates from different servers.

Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

2

Reading (not modifying) incoming e-mail while you are on the move.

Each time you connect to a different e-mail server, that server fetches (at least) all the updates from the server you previously visited.

3

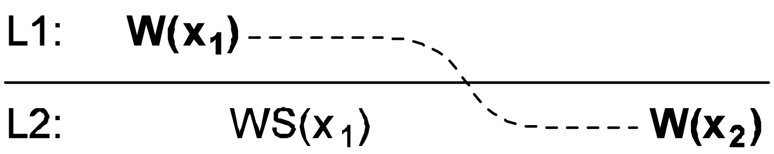
DNS System

Once you see the update of a DNS entry, you never see the old one.

Monotonic-Write Consistency

***Definition**: A write operation by a process on a data item **x** is completed before any successive write operation on **x** by the same process**.*

Intuition: Write happens on a copy only if it's brought up to date with preceding write operations on same data (but possibly at different copies)

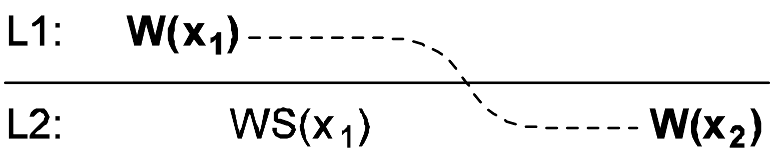


Monotonic Writes

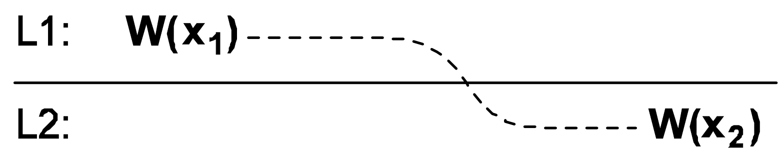
Monotonic-Write Consistency

Definition*: A write operation by a process on a data item* ***x*** *is completed before any successive write operation on* ***x*** *by the same process***.**

Intuition: Write happens on a copy only if it's brought up to date with preceding write operations on same data (but possibly at different copies)



Monotonic Writes

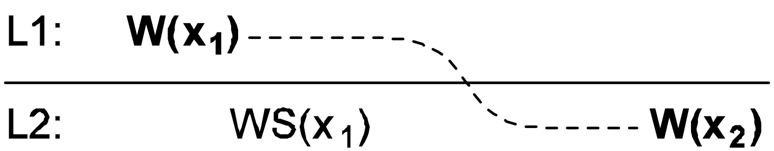


No Monotonic Writes

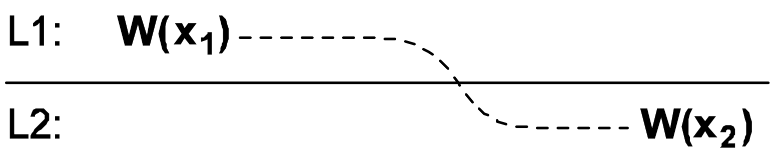
Monotonic-Write Consistency

***Definition**: A write operation by a process on a data item **x** is completed before any successive write operation on **x** by the same process**.*

Intuition: Write happens on a copy only if it's brought up to date with preceding write operations on same data (but possibly at different copies)



Monotonic Writes



No Monotonic Writes

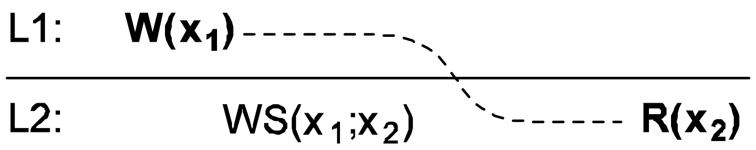
Examples

- Software version control: always update before committing changes.

Read-Your-Writes Consistency

Definition: The effect of a write operation by a process on data item **x**, will always be seen by a successive read operation on **x** by the same process.

Intuition: All previous writes are always completed before any successive read

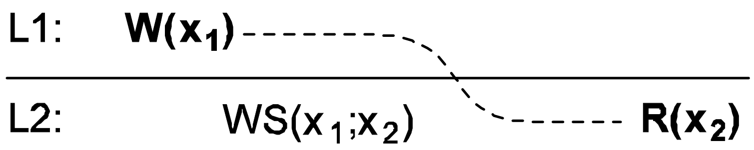


Read your writes

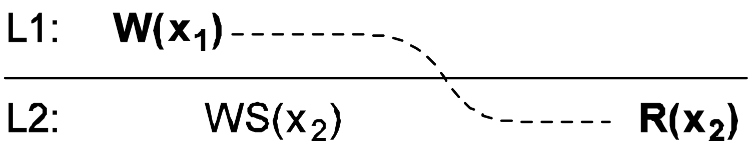
Read-Your-Writes Consistency

Definition: The effect of a write operation by a process on data item **x**, will always be seen by a successive read operation on **x** by the same process.

Intuition: All previous writes are always completed before any successive read



Read your writes

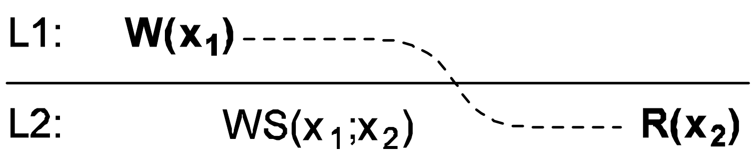


No Read your writes

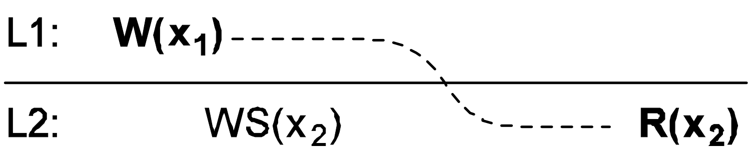
Read-Your-Writes Consistency

Definition: The effect of a write operation by a process on data item **x**, will always be seen by a successive read operation on **x** by the same process.

Intuition: All previous writes are always completed before any successive read



Read your writes



No Read your writes

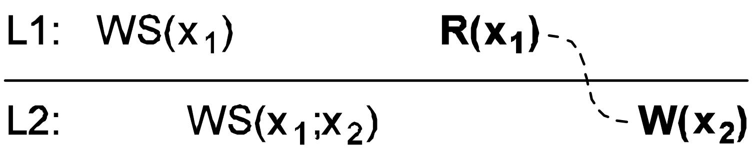
Examples

- Password databases
- Update of web page immediately pushes new version to caches

Writes-Follow-Reads Consistency

Definition: A write operation by a process on a data item **x** following a previous read operation on **x** by the same process, is guaranteed to take place on the same or a more recent value of **x** that was read**.**

Intuition: Any successive write operation on **x** will be performed on a copy of **x** that is same or more recent than the last read.

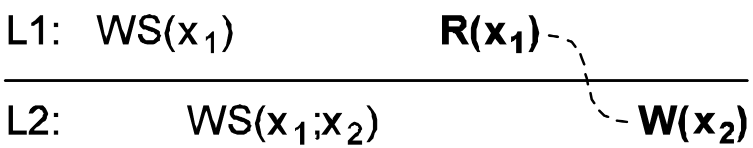


Writes follows reads

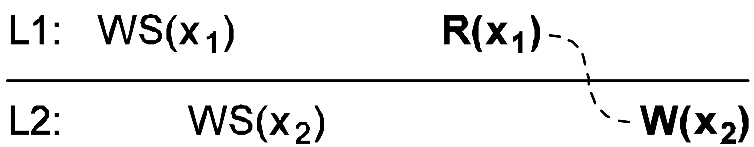
Writes-Follow-Reads Consistency

Definition: A write operation by a process on a data item **x** following a previous read operation on **x** by the same process, is guaranteed to take place on the same or a more recent value of **x** that was read**.**

Intuition: Any successive write operation on **x** will be performed on a copy of **x** that is same or more recent than the last read.



Writes follows reads

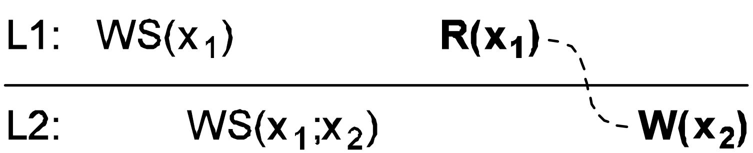


No Writes follows reads

Writes-Follow-Reads Consistency

Definition: A write operation by a process on a data item **x** following a previous read operation on **x** by the same process, is guaranteed to take place on the same or a more recent value of **x** that was read.

Intuition: Any successive write operation on **x** will be performed on a copy of **x** that is same or more recent than the last read.



Writes follows reads

No Writes follows reads

Examples

- Comments on a web page, Forum…

Trade-offs in choosing a consistency model

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

1 **Consistency and redundancy**

- All copies must be strongly consistent
- All copies must contain full state
- Reduced consistency leads to reduced redundancy

2 **Consistency and performance**

- Consistency induces extra computation and communication
- Increased consistency leads to decreased performance

3 **Consistency and scalability**

- Scalability depends on the implementation of a consistency model
  - Avoid centralized approaches
  - Avoid strong increase in communication

Further Reading

- **Replication and Consistency - Jussi Kangasharju**
  https://www.cs.helsinki.fi/webfm_send/1256

- **Tannenbaum and Van Steen – Chapter 7**
  http://csis.pace.edu/~marchese/CS865/Lectures/Chap7/Chapter7fin.htm

- **Consistency models in modern distributed Systems,**
  https://riunet.upv.es/bitstream/handle/10251/54786/TFMLeticiaPascual.pdf;sequence=1

- **Consistency in Distributed Systems , Burckhardt (2016)**
  https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/printversion.pdf

- **David Mosberger - Memory consistency models**
  https://dl.acm.org/doi/10.1145/160551.160553

Questions?

Replication