Distributed Platforms

- References:
    - A. D. Fekete and K. Ramamritham. **Consistency models for replicated data**. In B. Charron-Bost, F. Pedone, and A. Schiper, editors, Replication, volume 5959 of Lecture Notes in Computer Science, pages 1–17. Springer, 2010.
    - Any serious recent distributed systems book 😄

Scalability

- The ability of a system, network, or process, to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.

- We can measure growth in almost any terms. But there are three particularly interesting things to look at:

  - Size scalability: adding more nodes should make the system linearly faster; growing the dataset should not increase latency

  - Geographic scalability: it should be possible to use multiple data centers to reduce the time it takes to respond to user queries, while dealing with cross-data center latency in some sensible manner.

  - Administrative scalability: adding more nodes should not increase the administrative costs of the system (e.g. the administrators-to-machines ratio).

- A scalable system is one that continues to meet the needs of its users as scale increases. There are two particularly relevant aspects - performance and availability - which can be measured in various ways.

Performance (and latency)

- Characterization of the amount of useful work accomplished by a computer system compared to the time and resources used.
- Depending on the context, this may involve achieving one or more of the following:
    - Short response time/low latency for a given piece of work
    - High throughput (rate of processing work)
    - Low utilization of computing resource(s)
- Latency: the state of being latent; delay, a period between the initiation of something and the occurrence.
- Latent: From Latin *latens, latentis*, present participle of *lateo* ("lie hidden"). Existing or present but concealed or inactive.

Availability (and fault tolerance)

- the proportion of time a system is in a functioning condition. If a user cannot access the system, it is said to be unavailable.
- In formula, availability = uptime / (uptime + downtime)
- from a technical perspective, availability is mostly about being fault tolerant.
- Fault tolerance is the ability of a system to behave in a well-defined manner once faults occur

Availability	Nickname	Downtime per year
90%	one nine	more than a month
99%	two nines	less than 4 days
99.9%	three nines	less than 9 hours
99.99%	four nines	less than 1 hour
99999%	five nines	about 5 minutes
99.9999%	six nines	about 31 seconds

Examples

Examples

- One single server
- On average, one crash per week
  - Mean Time Between Failures (MTBF) 10800 mins
- Two minutes to reboot
  - Mean Time To Restart (MTTR) 2 mins

Examples

- One single server
- On a
  - Me
- Two r
  - Mean Time To Restart (MTTR) 2 mins

nins

$A = 10080/10802 = 0.\underline{9998}$

Examples

- One single server
- On a
  - Me nins
- Two r
  - Mean Time To Restart (MTTR) 2 mins

A = 10080/10802 = 0.<u>9998</u>

- 10 servers
- MTBF, MTTR as before
- All servers needed to perform operations

Examples

- One single server
- On a
  - Me mins
- Two r
  - Mean Time To Restart (MTTR) 2 mins

$$A = 10080/10802 = 0.\underline{9998}$$

- 10 se
- MTB
- All servers needed to perform operations

$$p_f = 2/10802$$

$$A = (1-p_f)^{10} = 0.\underline{998}$$

Replication

- Increase **availability**
  - Avoid single point of failure
  - Time/Space replication
- **Time** replication
  - When a replica fails, restart or replace it
  - Lower maintenance, lower availability
- **Space** replication
  - Run parallel copies, vote for output
  - High-availability, high-cost

Problem

- Whenever a copy is modified, that copy becomes different from the rest
- Modifications have to be carried out on all copies to ensure **consistency**
- **Conflicting** operations:
    - **Read–write** conflict: concurrent read and write operations
    - **Write–write** conflict: two concurrent write operations

Message from Amazon

"Whether or not inconsistencies are acceptable depends on the client application. In all cases **the developer must be aware** that consistency guarantees are provided by the storage systems and must be taken into account when developing applications."

Amazon vice-president and Chief Scientific Officer

W. Vogels. *Eventual consistent*. **Comm. of the ACM**, 52(1):40–44, 2009

Message from Amazon

"Whether or not inconsistencies are acceptable depends on the client application. In all cases

must be

consiste

are provided by the storage systems and must be taken into account when developing applications."

Se non lo sapevi, sallo!!!!

t and Chief Scientific Officer

W. Vogels. *Eventual consistent*. **Comm. of the ACM**, 52(1):40–44, 2009

Consistency

- **Consistency model** – A contract between a distributed data store and a set of processes, which specifies what the results of read/write operations are in the presence of concurrency
- **Strong consistency** models
    - Strict consistency
    - Linearizability
    - Sequential consistency
- **Weak consistency** models
    - Eventual consistency
    - Client-centric consistency models
        - Read-after-read (*monotonic read*)
        - Read-after-write (*read your writes*)
    - Causal consistency

Strict Consistency

Definition

- A read operation must return the result of the latest write operation which occurred on the data item

Implementation:

- Only possible with a global, perfectly synchronized clock
- Only possible if all writes instantaneously visible to all
- It is the model of uniprocessor systems!

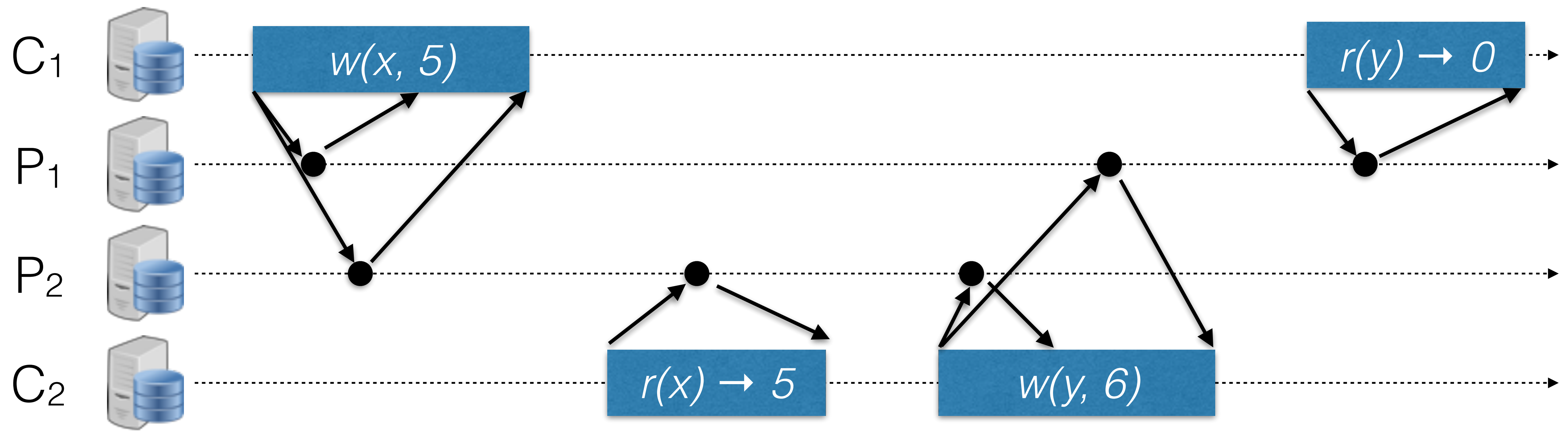
Linearizability

Definition

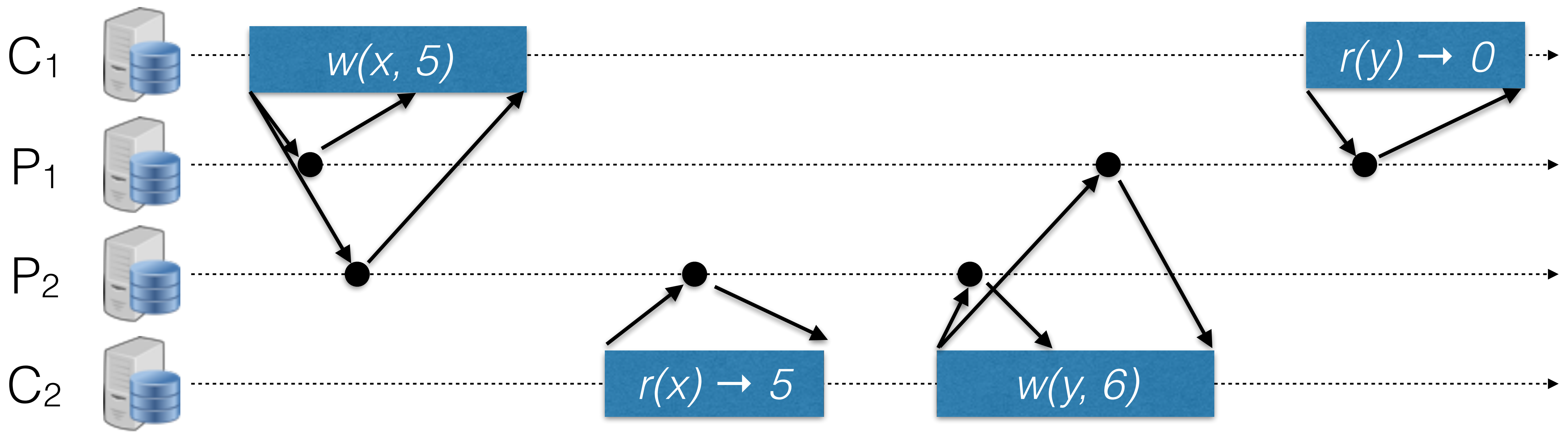
An execution E is **linearizable** provided that there exists a sequence (*linearization*) H such that:

- H *contains exactly* the same operations that occur in E, each paired with the return value received in E
- H is a *legal history* of the sequential data type that is replicated
- the total order of operations in H is *compatible with* the **real-time partial order** $<$
    - $o_1 < o_2$ means that the duration of operation o_1 (from invocation till it returns) occurs entirely before the duration of operation o_2

Example



Example



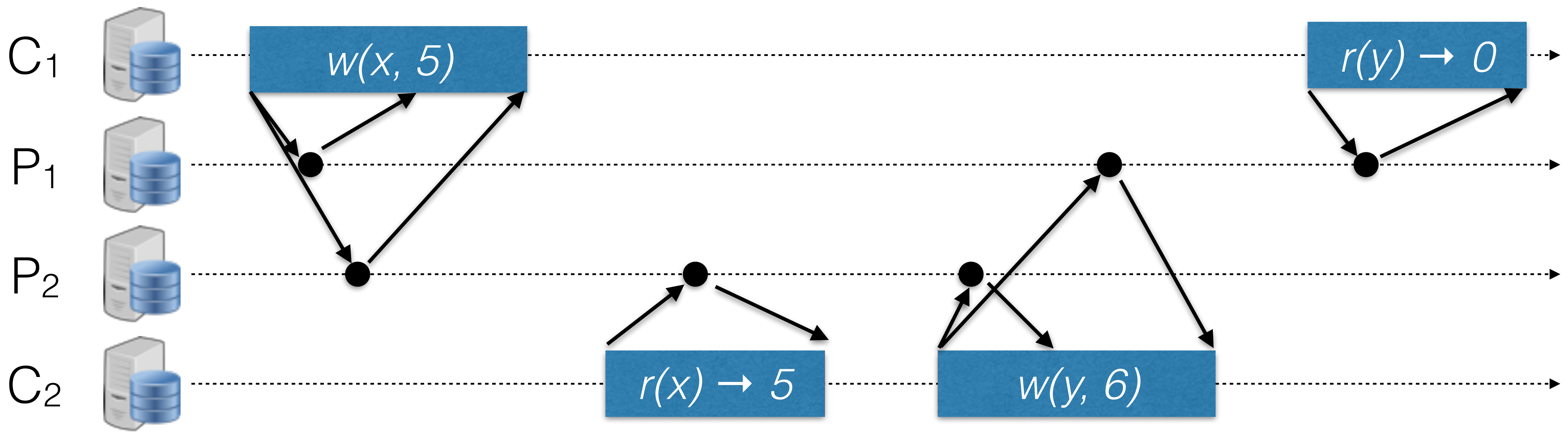
Are the following sequences possible linearizations?

w(x, 5)   r(x) → 5   w(y, 6)   r(y) → 0

w(x, 5)   r(x) →5   r(y) → 0   w(y, 6)

Is the above execution linearizable?

Example



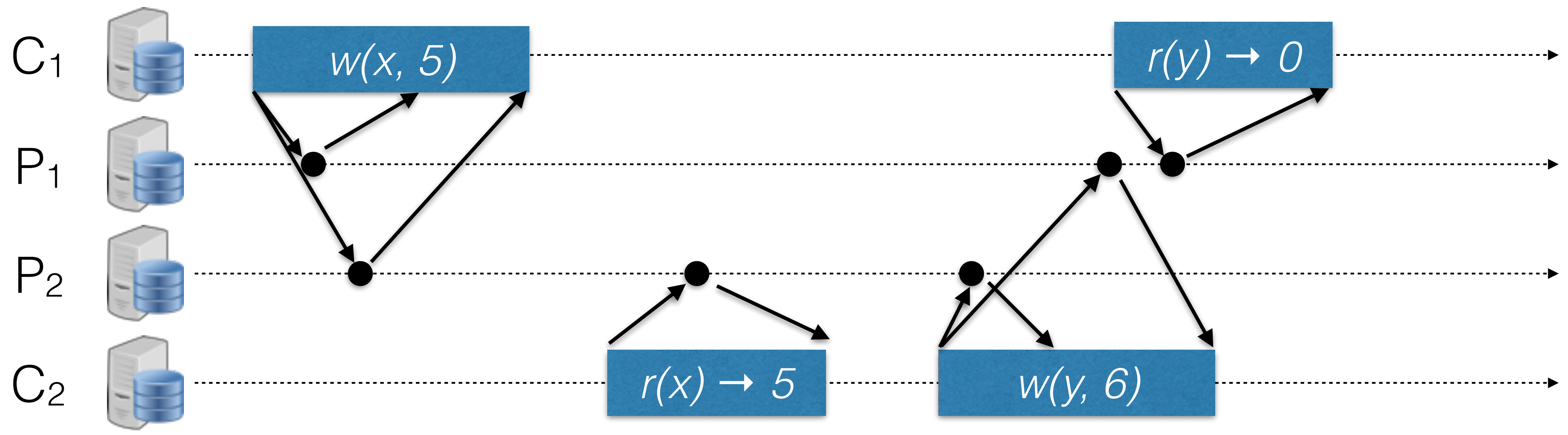
Are the following sequences possible linearizations?

w(x, 5) *r(x) → 5* *w(y, 6)* *r(y) → 0* NO

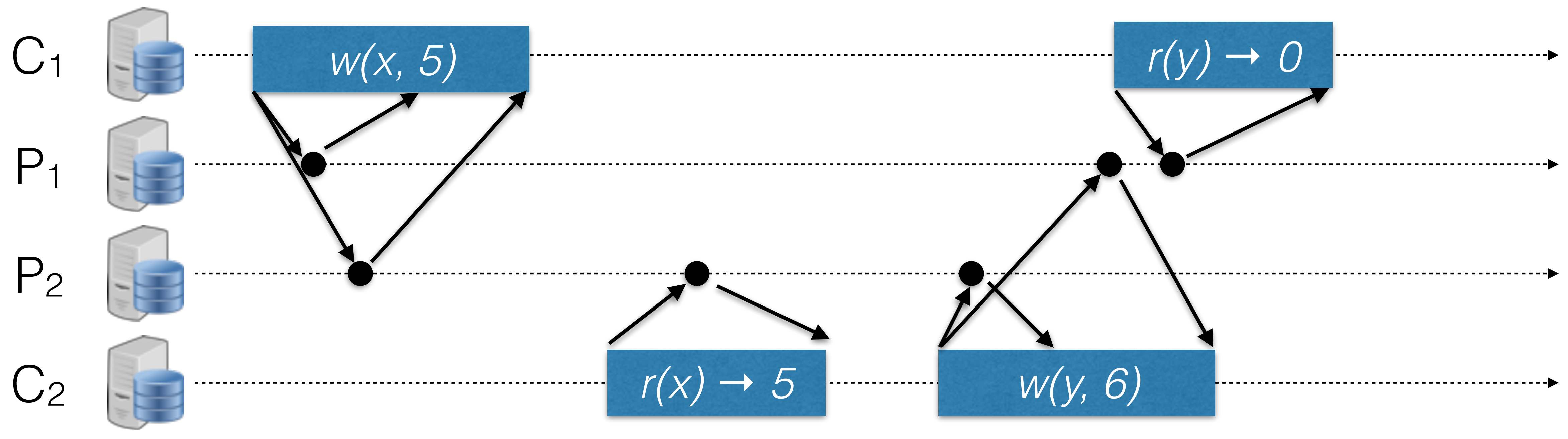
w(x, 5) *r(x) →5* *r(y) → 0* *w(y, 6)* NO

Is the above execution linearizable? NO

Example



Example



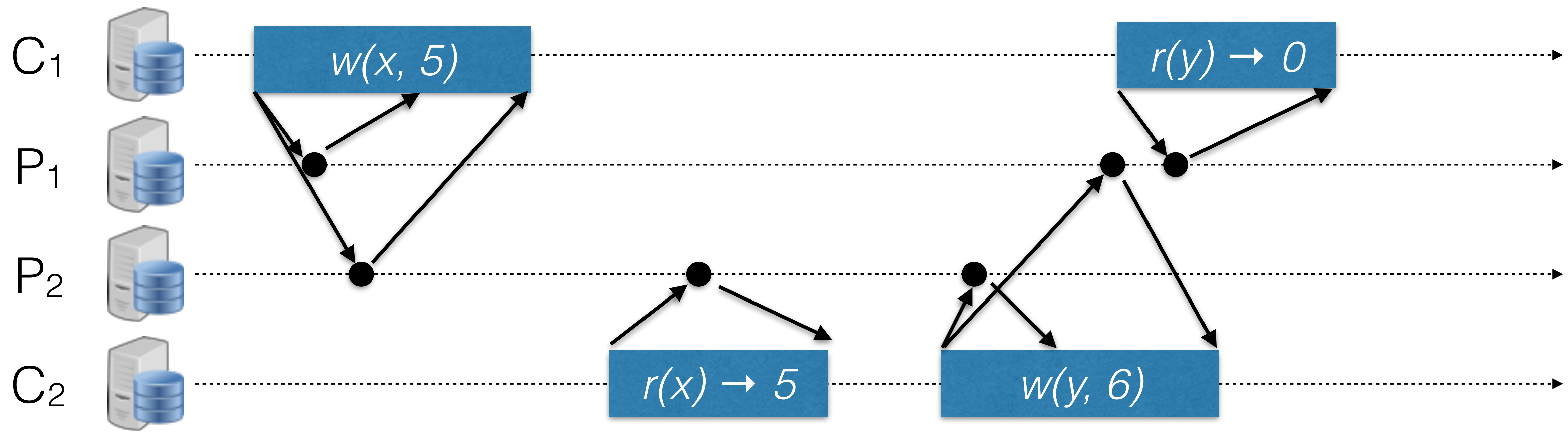
Are the following sequences possible linearizations?

w(x, 5) r(x) → 5 w(y, 6) r(y) → 0

w(x, 5) r(x) → 5 r(y) → 0 w(y, 6)

Is the above execution linearizable?

Example

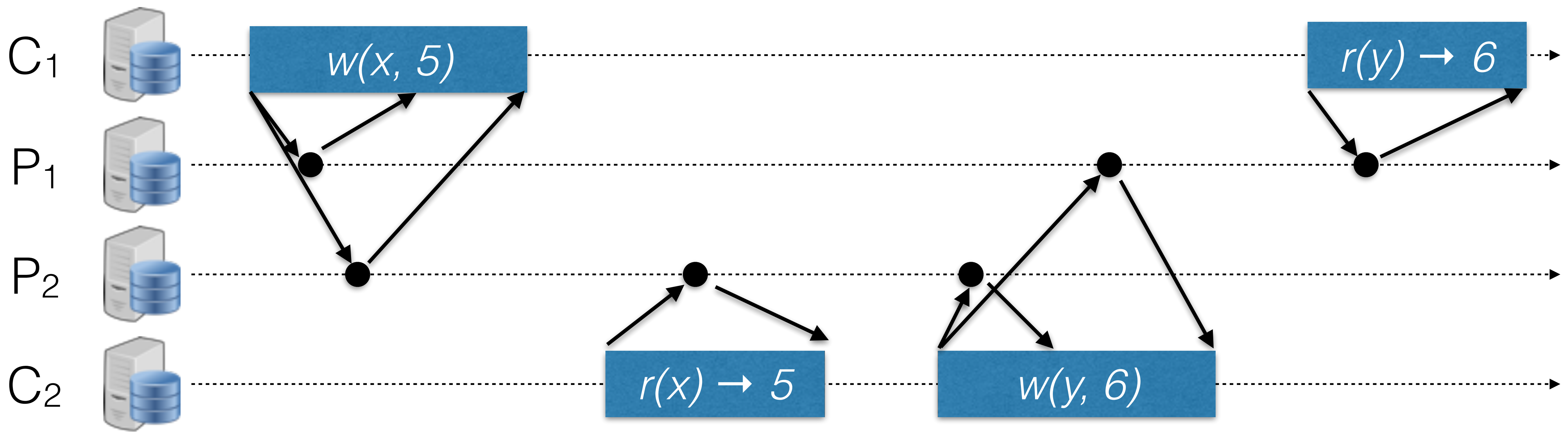


Are the following sequences possible linearizations?

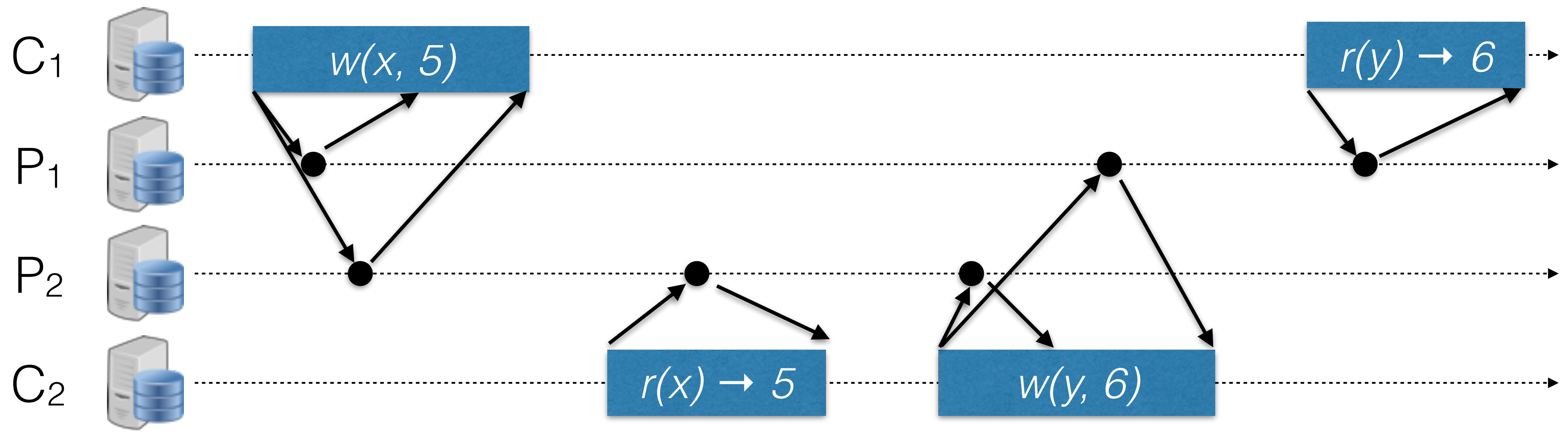
w(x, 5) *r(x) → 5* *w(y, 6)* *r(y) → 0*	NO
w(x, 5) *r(x) →5* *r(y) → 0* *w(y, 6)*	YES

Is the above execution linearizable? YES

Example



Example



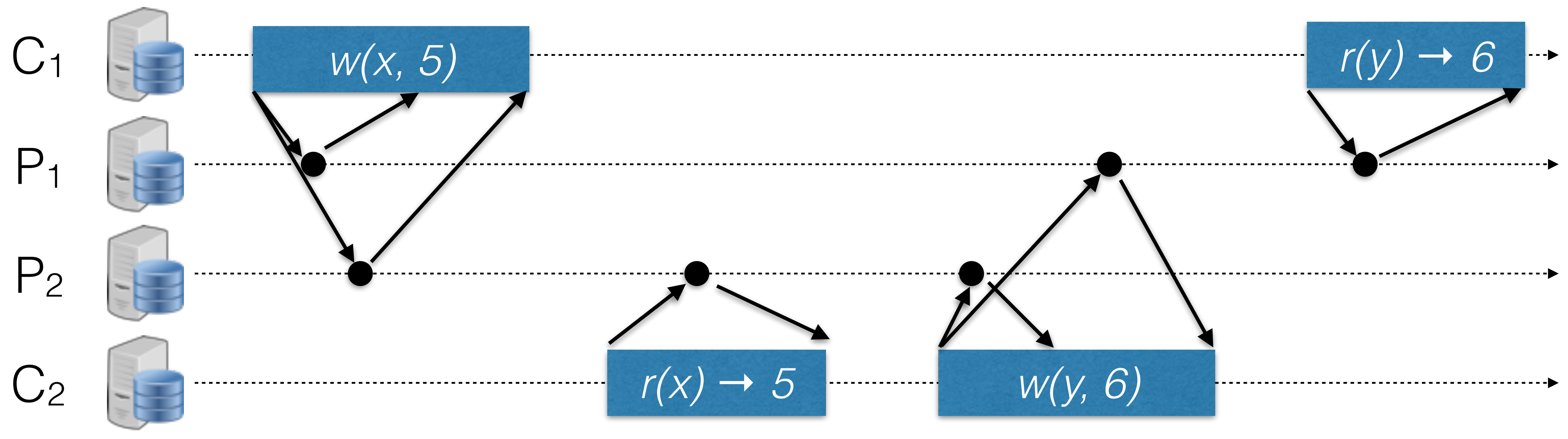
Are the following sequences possible linearizations?

w(x, 5) r(x) → 5 w(y, 6) r(y) → 6

w(x, 5) r(x) →5 r(y) → 6 w(y, 6)

Is the above execution linearizable?

Example



Are the following sequences possible linearizations?

w(x, 5) r(x) → 5 w(y, 6) r(y) → 6	YES
w(x, 5) r(x) →5 r(y) → 6 w(y, 6)	NO

Is the above execution linearizable? YES

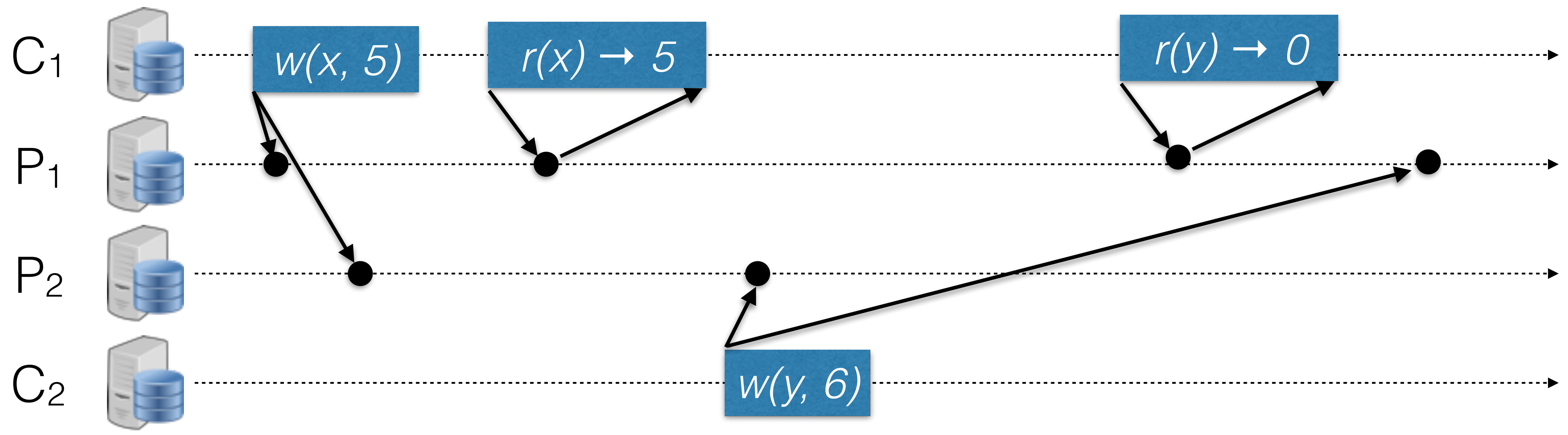
Sequential Consistency

Definition

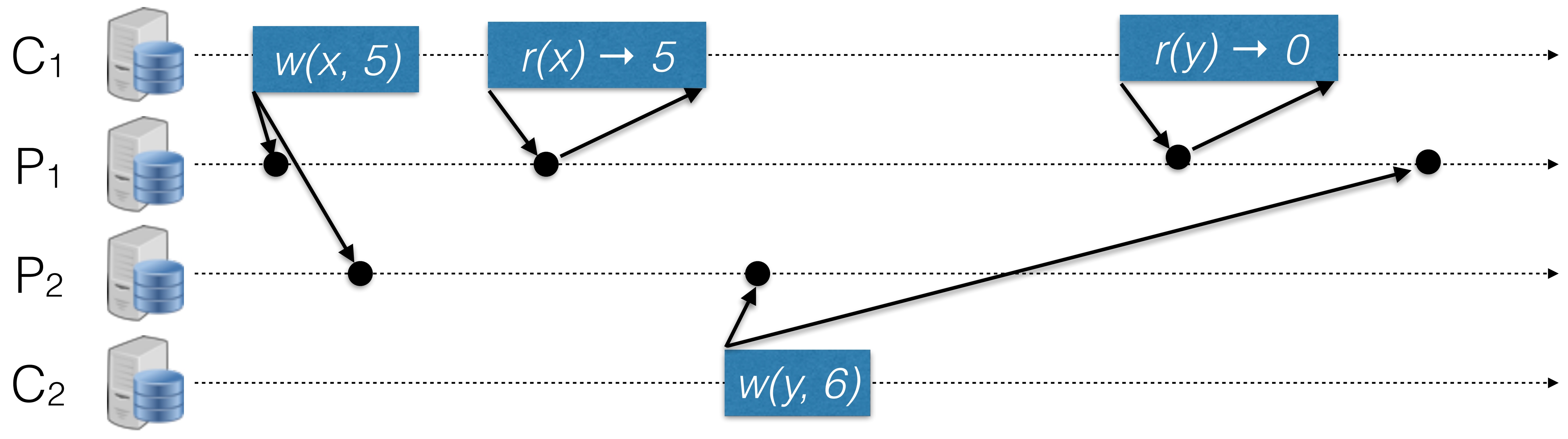
An execution E is **sequential consistent** provided that there exists a sequence H such that

- H *contains exactly* the same operations that occur in E, each paired with the return value received in E
- H is a *legal history* of the sequential data type that is replicated
- The total order of operations in H is *compatible with* the **client partial order** <
  - $o_1 < o_2$ means that the o_1 and o_2 occur at the same client and that o_1 returns before o_2 is invoked

Example

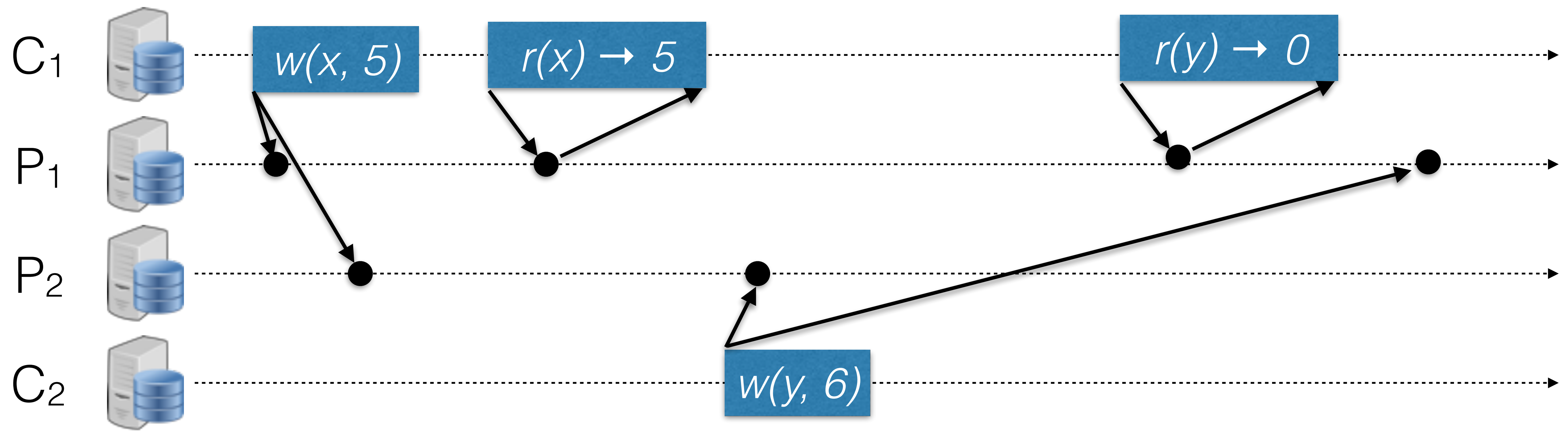


Example



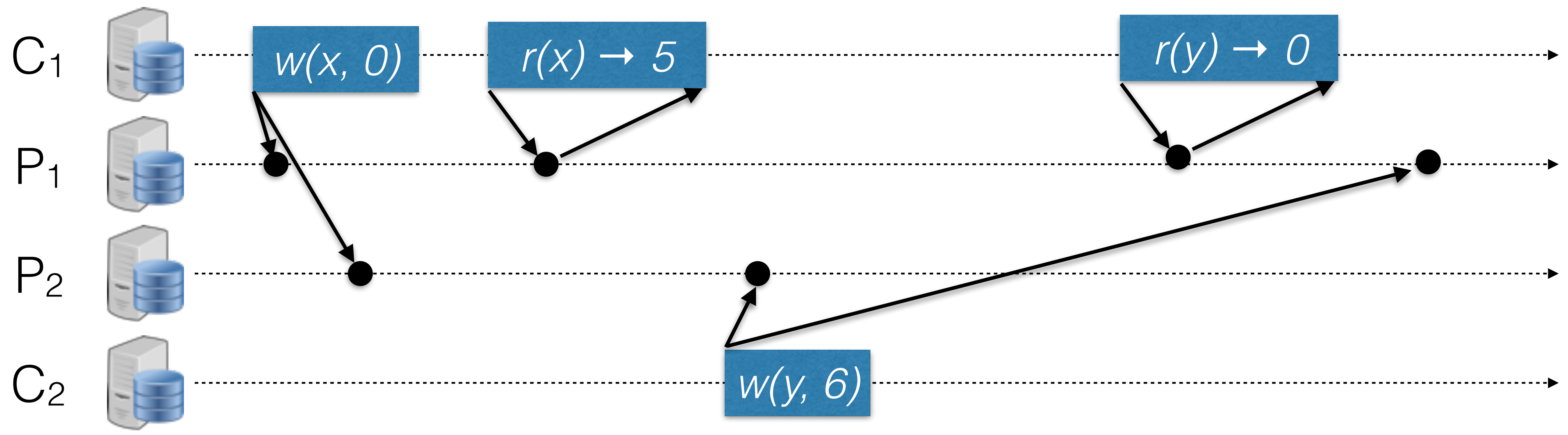
Is the execution above sequentially consistent?

Example

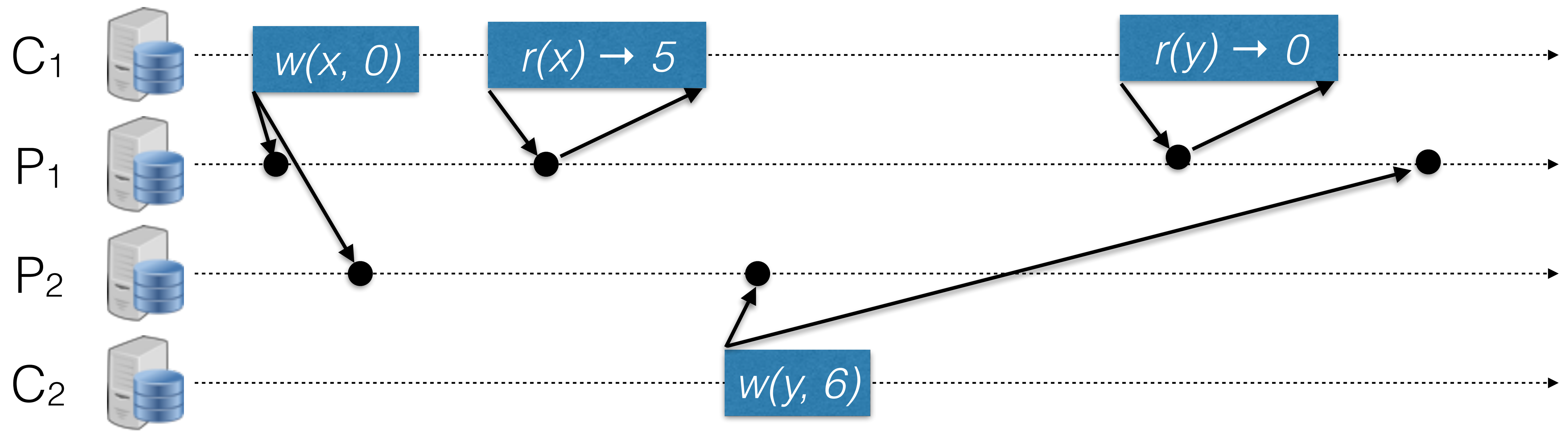


Is the execution above sequentially consistent? YES

Example

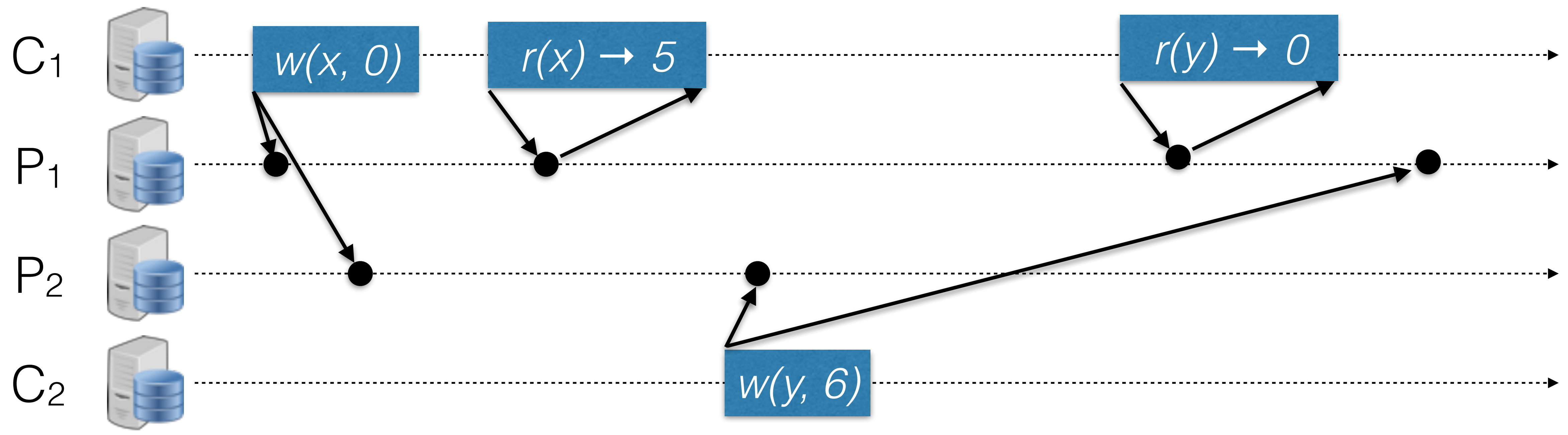


Example



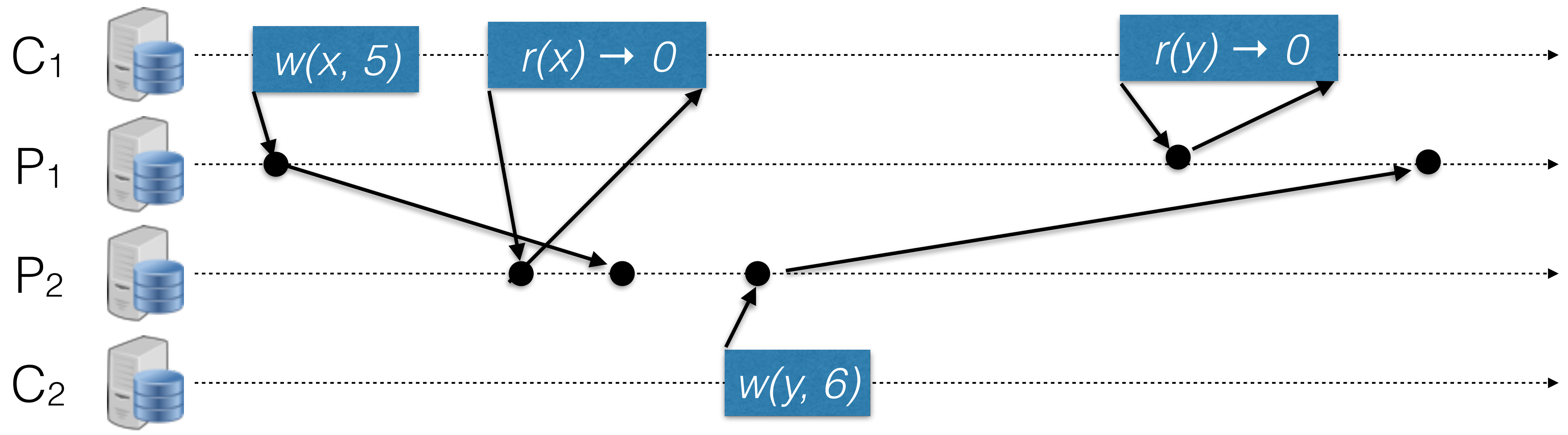
Is the execution above sequentially consistent?

Example

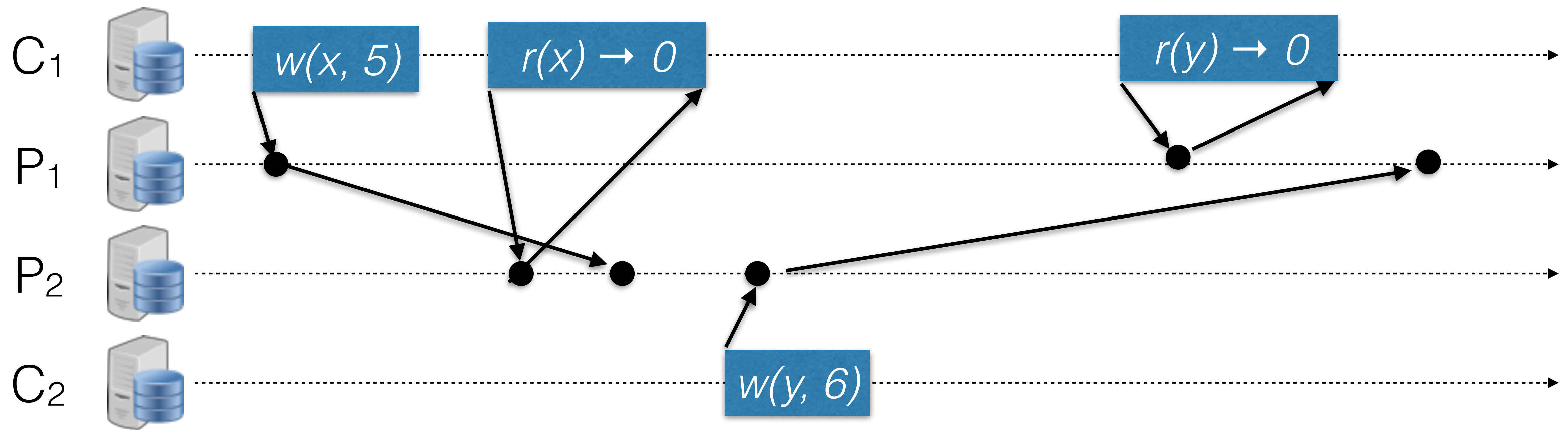


Is the execution above sequentially consistent? NO

Example

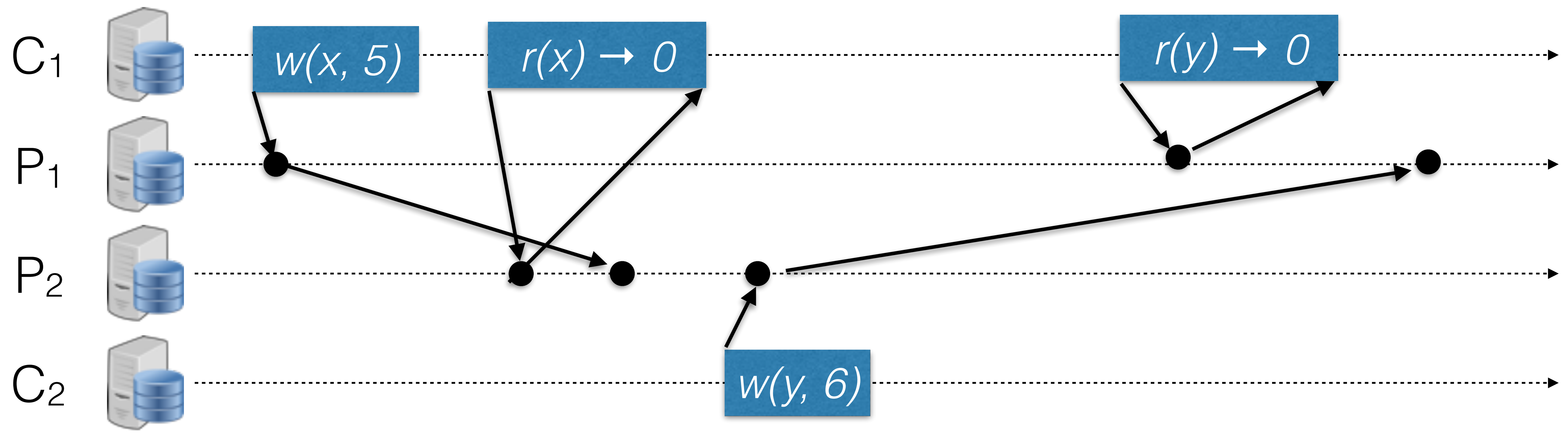


Example



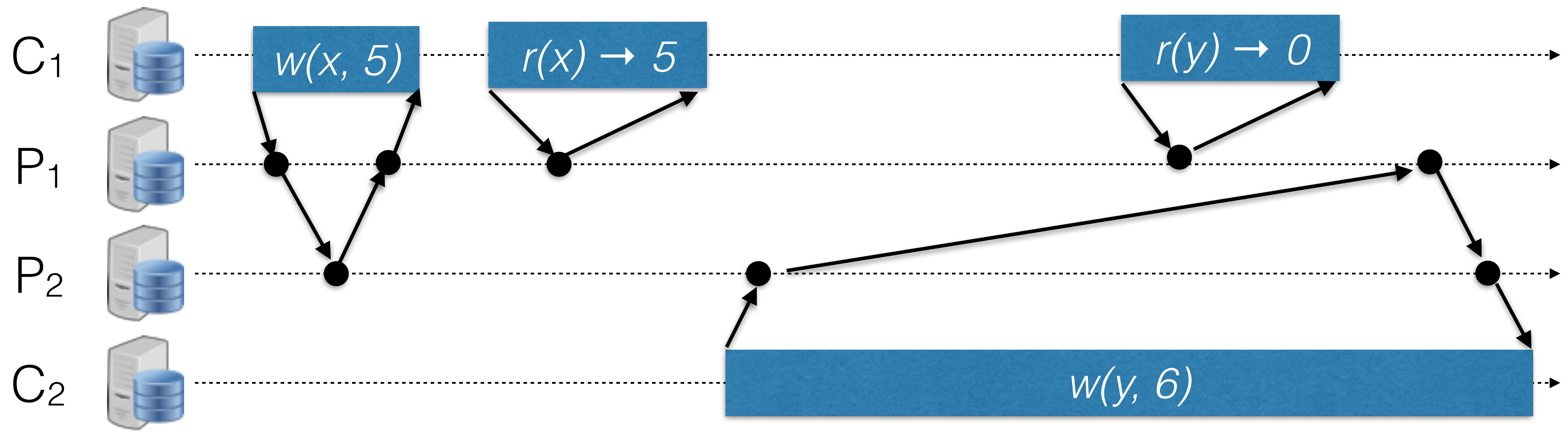
Is the execution above sequentially consistent?

Example

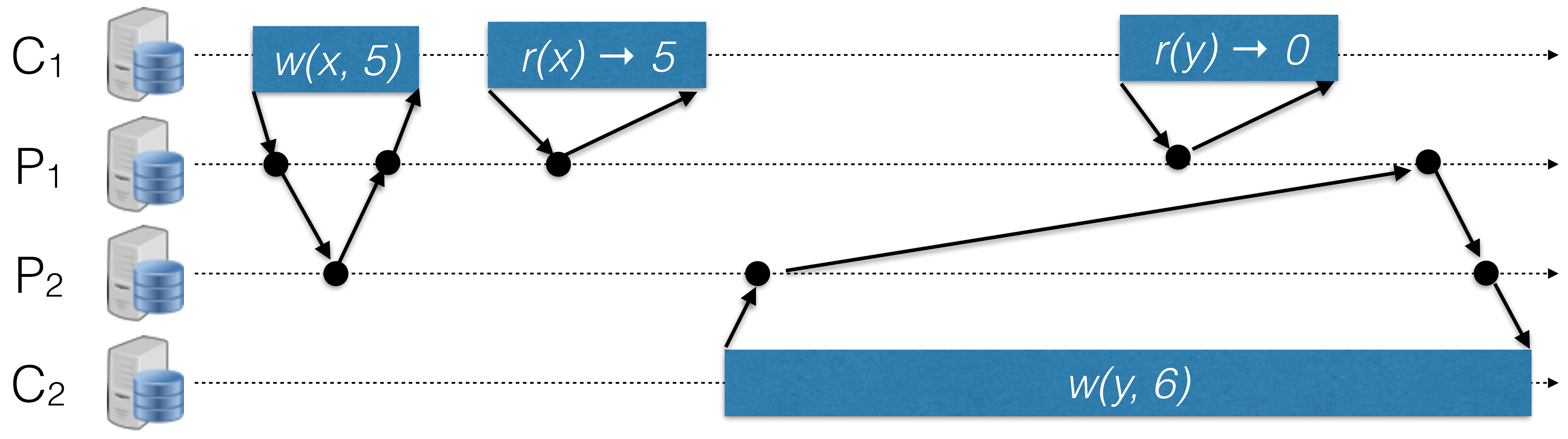


Is the execution above sequentially consistent? NO

Example

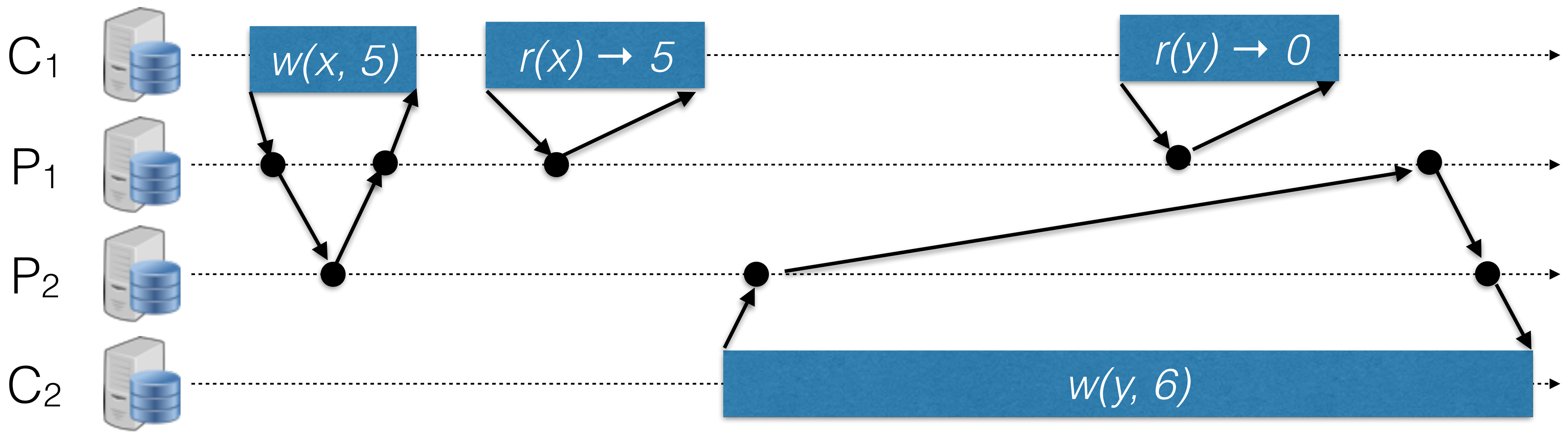


Example



Is the execution above sequentially consistent?

Example



Is the execution above sequentially consistent? YES

Issues

- It is easy to provide **strong consistency** through appropriate hardware and/or software mechanisms
    - But these are typically found to incur considerable **penalties**, in latency, availability after faults, etc.
- **Strong consistency** often implies that *message should arrive in the same order*
    - Can be implemented through a **sequencer replica**
        - Latency: the sequencer replica becomes a bottleneck
        - Availability: a new sequencer must be elected after a failure
- **Weak consistency** relaxes the *precise details* of *which reorderings are allowed*
    - Within the activity of a client
    - By whether there are any constraints at all on the information provided to different clients

Eventual Consistency

- Consider a system where
  - updates are rare
  - concurrent updates are absent, or can be easily resolved in an automatic way
  - Example: Domain Name System
- **Eventual Consistency**
  - If no updates take place for a long time, all replicas will gradually become consistent (i.e., the same)
    - The consistency policy of epidemic protocols
    - This is not a safety property, is a liveness one

Issues

- Consider a **replicated database** that you access through your **notebook**. The notebook acts as a front-end to the database
- The database is **eventually consistent**
- You **move from** location A to location B
- Unless you use the same server, you may **detect inconsistencies**:
    - your updates at A may **not have yet been propagated** to B
    - you may be **reading newer entries** than the ones available at A
    - your updates at B may **eventually conflict** with those at A
- The only thing you really care is that the entries you updated and/or read at A, are in B the way you left them in A. In that case, **the database will appear to be consistent to you**

Client-centric Consistency

- In some cases, we can avoid system-wide consistency, by concentrating on what specific clients want, instead of what should be provided by servers
- Models:
  - **Read-after-read / Monotonic reads**
  - **Write-after-write / Monotonic writes**
  - **Read-after-write / Read-your-writes**
  - **Write-after-read / Write-follows-reads**

Read-after-read / Monotonic reads

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 value or a more recent value

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

Read-after-write / Read-your-writes

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

Example: Editing of a web page. Updating your web page and guaranteeing that your web browser shows the newest version instead of its cached copy.

Write-after-write / Monotonic writes

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

Example: Concurrent software development systems. Each time you connect to a CVS server, that server updates (at least) over all the changes you previously did.

Write-after-read / Write-follows-reads

A write operation by a process on 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

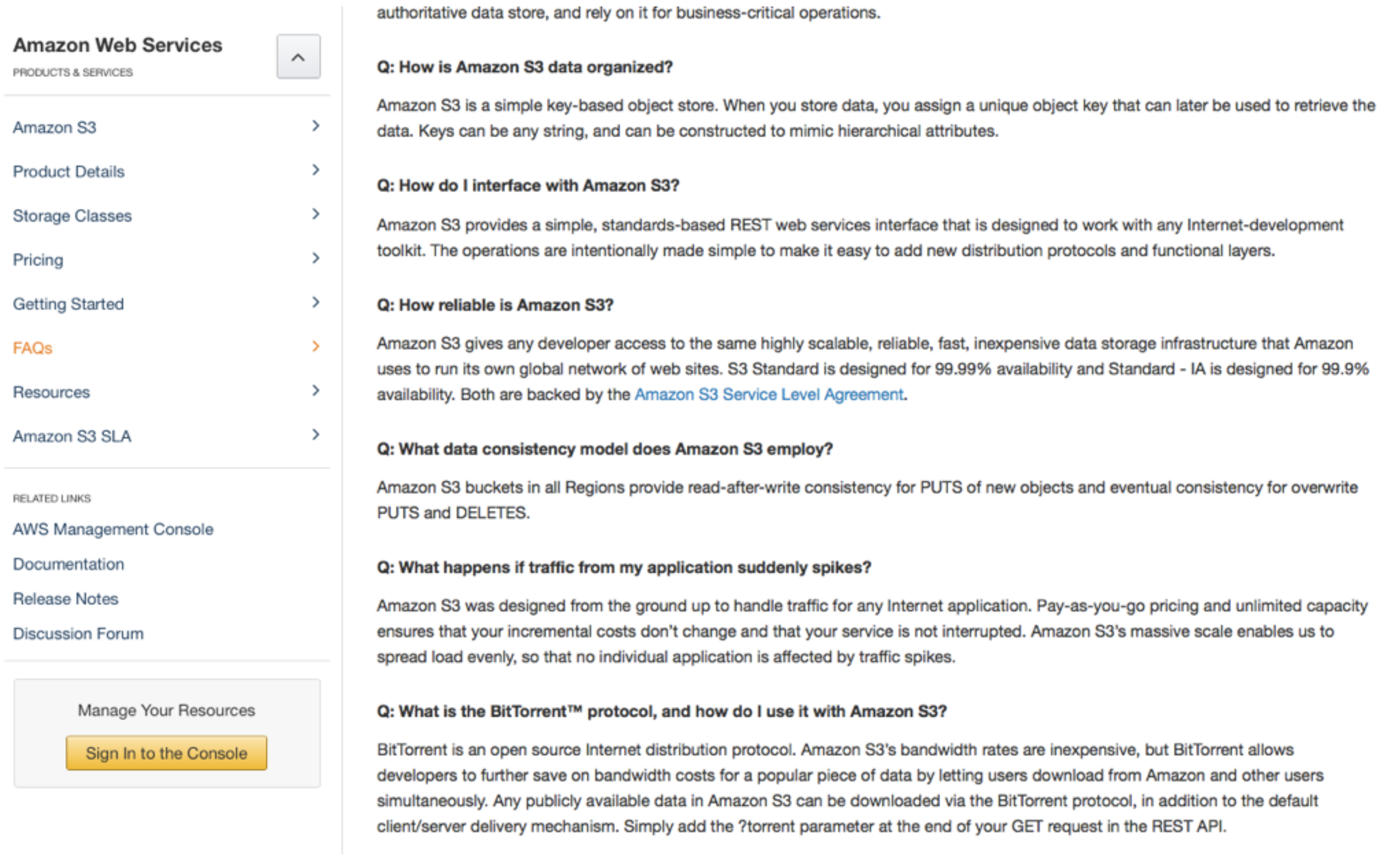
Writes affect only up-to-date data items

Example:

<u>Example</u>: Comments to posts on Facebook. Each time you write a comment to a post, the server must fetch (at least) all previous updates to that post you previously read.

Session Consistency

- A practical version of read-your-writes, where processes access a data storage in the context of a session
- As long as the session exists, the system guarantees read-your-writes
- If the session terminates because of a failure, a new session must be created
- Guarantees are limited to sessions



authoritative data store, and rely on it for business-critical operations.

Q: How is Amazon S3 data organized?

Amazon S3 is a simple key-based object store. When you store data, you assign a unique object key that can later be used to retrieve the data. Keys can be any string, and can be constructed to mimic hierarchical attributes.

Q: How do I interface with Amazon S3?

Amazon S3 provides a simple, standards-based REST web services interface that is designed to work with any Internet-development toolkit. The operations are intentionally made simple to make it easy to add new distribution protocols and functional layers.

Q: How reliable is Amazon S3?

Amazon S3 gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. S3 Standard is designed for 99.99% availability and Standard - IA is designed for 99.9% availability. Both are backed by the Amazon S3 Service Level Agreement.

Q: What data consistency model does Amazon S3 employ?

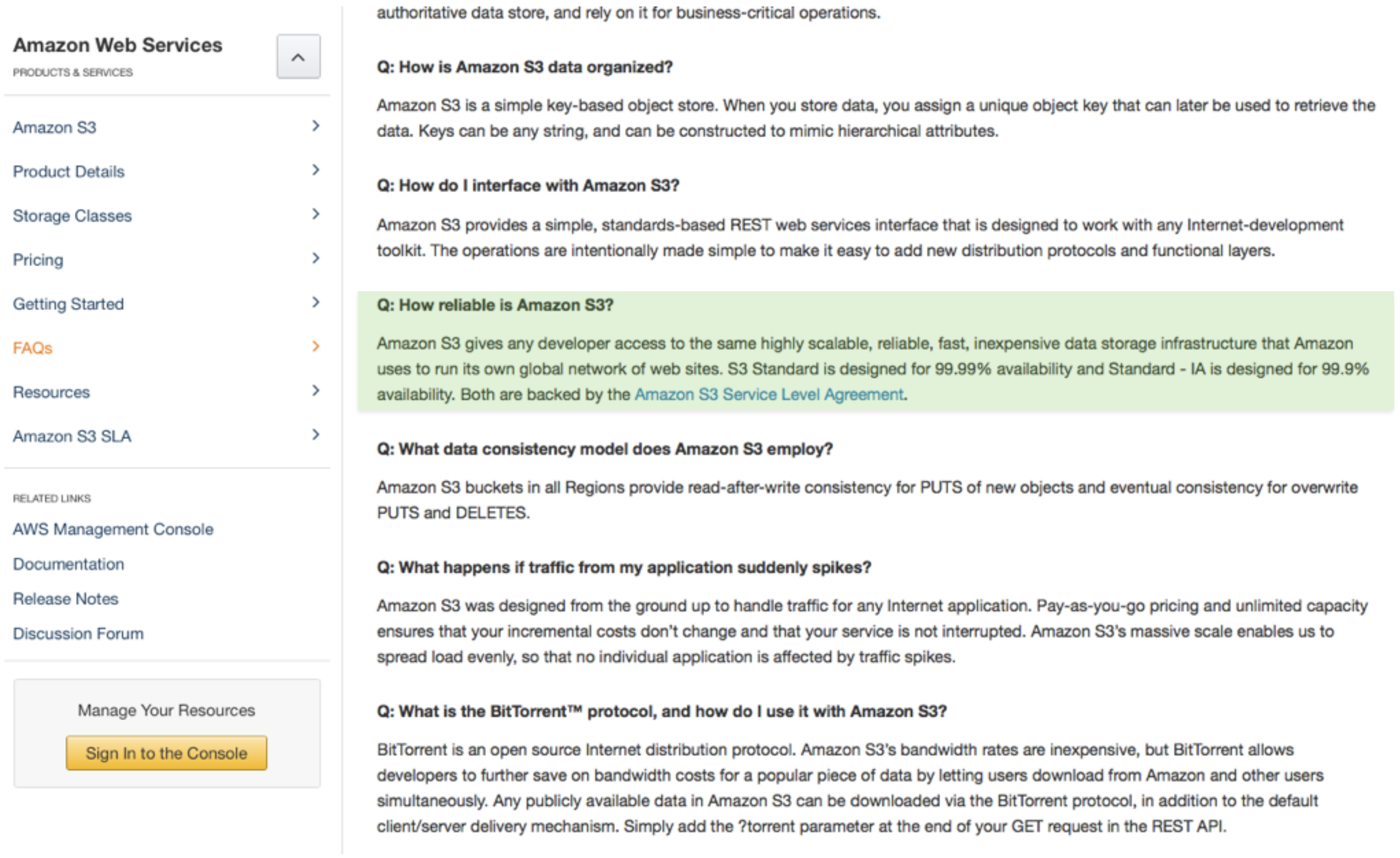
Amazon S3 buckets in all Regions provide read-after-write consistency for PUTS of new objects and eventual consistency for overwrite PUTS and DELETES.

Q: What happens if traffic from my application suddenly spikes?

Amazon S3 was designed from the ground up to handle traffic for any Internet application. Pay-as-you-go pricing and unlimited capacity ensures that your incremental costs don't change and that your service is not interrupted. Amazon S3's massive scale enables us to spread load evenly, so that no individual application is affected by traffic spikes.

Q: What is the BitTorrent™ protocol, and how do I use it with Amazon S3?

BitTorrent is an open source Internet distribution protocol. Amazon S3's bandwidth rates are inexpensive, but BitTorrent allows developers to further save on bandwidth costs for a popular piece of data by letting users download from Amazon and other users simultaneously. Any publicly available data in Amazon S3 can be downloaded via the BitTorrent protocol, in addition to the default client/server delivery mechanism. Simply add the ?torrent parameter at the end of your GET request in the REST API.



authoritative data store, and rely on it for business-critical operations.

Q: How is Amazon S3 data organized?

Amazon S3 is a simple key-based object store. When you store data, you assign a unique object key that can later be used to retrieve the data. Keys can be any string, and can be constructed to mimic hierarchical attributes.

Q: How do I interface with Amazon S3?

Amazon S3 provides a simple, standards-based REST web services interface that is designed to work with any Internet-development toolkit. The operations are intentionally made simple to make it easy to add new distribution protocols and functional layers.

Q: How reliable is Amazon S3?

Amazon S3 gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. S3 Standard is designed for 99.99% availability and Standard - IA is designed for 99.9% availability. Both are backed by the Amazon S3 Service Level Agreement.

Q: What data consistency model does Amazon S3 employ?

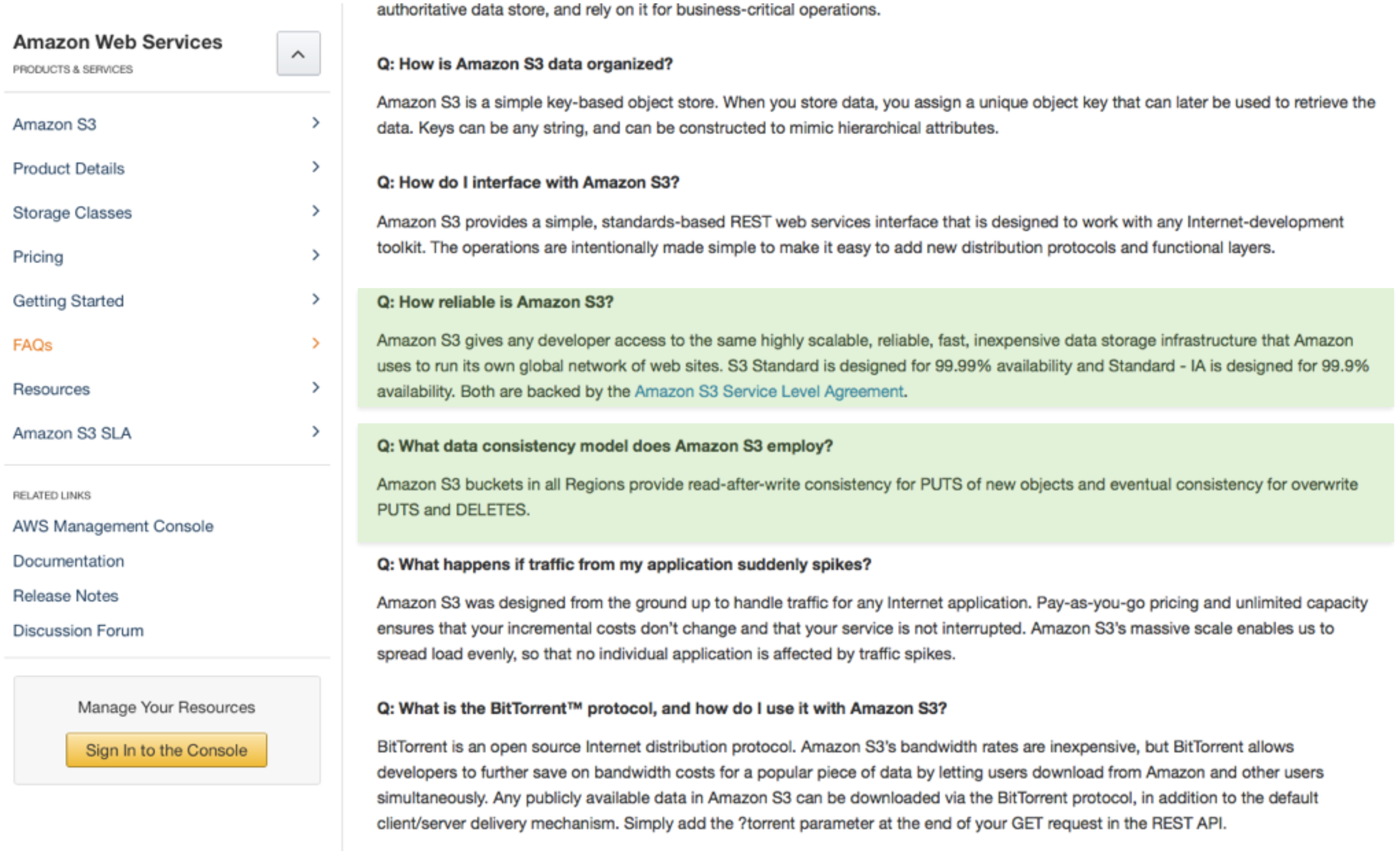
Amazon S3 buckets in all Regions provide read-after-write consistency for PUTS of new objects and eventual consistency for overwrite PUTS and DELETES.

Q: What happens if traffic from my application suddenly spikes?

Amazon S3 was designed from the ground up to handle traffic for any Internet application. Pay-as-you-go pricing and unlimited capacity ensures that your incremental costs don't change and that your service is not interrupted. Amazon S3's massive scale enables us to spread load evenly, so that no individual application is affected by traffic spikes.

Q: What is the BitTorrent™ protocol, and how do I use it with Amazon S3?

BitTorrent is an open source Internet distribution protocol. Amazon S3's bandwidth rates are inexpensive, but BitTorrent allows developers to further save on bandwidth costs for a popular piece of data by letting users download from Amazon and other users simultaneously. Any publicly available data in Amazon S3 can be downloaded via the BitTorrent protocol, in addition to the default client/server delivery mechanism. Simply add the ?torrent parameter at the end of your GET request in the REST API.



authoritative data store, and rely on it for business-critical operations.

Q: How is Amazon S3 data organized?

Amazon S3 is a simple key-based object store. When you store data, you assign a unique object key that can later be used to retrieve the data. Keys can be any string, and can be constructed to mimic hierarchical attributes.

Q: How do I interface with Amazon S3?

Amazon S3 provides a simple, standards-based REST web services interface that is designed to work with any Internet-development toolkit. The operations are intentionally made simple to make it easy to add new distribution protocols and functional layers.

Q: How reliable is Amazon S3?

Amazon S3 gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. S3 Standard is designed for 99.99% availability and Standard - IA is designed for 99.9% availability. Both are backed by the Amazon S3 Service Level Agreement.

Q: What data consistency model does Amazon S3 employ?

Amazon S3 buckets in all Regions provide read-after-write consistency for PUTS of new objects and eventual consistency for overwrite PUTS and DELETES.

Q: What happens if traffic from my application suddenly spikes?

Amazon S3 was designed from the ground up to handle traffic for any Internet application. Pay-as-you-go pricing and unlimited capacity ensures that your incremental costs don't change and that your service is not interrupted. Amazon S3's massive scale enables us to spread load evenly, so that no individual application is affected by traffic spikes.

Q: What is the BitTorrent™ protocol, and how do I use it with Amazon S3?

BitTorrent is an open source Internet distribution protocol. Amazon S3's bandwidth rates are inexpensive, but BitTorrent allows developers to further save on bandwidth costs for a popular piece of data by letting users download from Amazon and other users simultaneously. Any publicly available data in Amazon S3 can be downloaded via the BitTorrent protocol, in addition to the default client/server delivery mechanism. Simply add the ?torrent parameter at the end of your GET request in the REST API.

Read your writes consistency

Some applications require the ability to read replicated data at a client site, and determine whether it is consistent with data that has been written previously at the master site.

For example, a web application may be backed by multiple database environments, linked to form a replication group, in order to share the workload. Web requests that update data must be served by the replication master, but any site in the group may serve a read-only request. Consider a work flow of a series of web requests from one specific user at a web browser: the first request generates a database update, but the second request merely reads data. If the read-only request is served by a replication client database environment, it may be important to make sure that the updated data has been replicated to the client before performing the read (or to wait until it has been replicated) in order to show this user a consistent view of the data.

Berkeley DB supports this requirement through the use of transaction "tokens". A token is a form of identification for a transaction within the scope of the replication group. The application may request a copy of the transaction's token at the master site during the execution of the transaction. Later, the application running on a client site can use a copy of the token to determine whether the transaction has been applied at that site.

It is the application's responsibility to keep track of the token during the interim. In the web example, the token might be sent to the browser as a "cookie", or stored on the application server in the user's session context.

The operations described here are supported both for Replication Manager applications and for applications that use the replication Base API.

Read your writes consistency

Some applications require the ability to read replicated data at a client site, and determine whether it is consistent with data that has been written previously at the master site.

For example, a web application may be backed by multiple database environments, linked to form a replication group, in order to share the workload. Web requests that update data must be served by the replication master, but any site in the group may serve a read-only request. Consider a work flow of a series of web requests from one specific user at a web browser: the first request generates a database update, but the second request merely reads data. If the read-only request is served by a replication client database environment, it may be important to make sure that the updated data has been replicated to the client before performing the read (or to wait until it has been replicated) in order to show this user a consistent view of the data.

Berkeley DB supports this requirement through the use of transaction "tokens". A token is a form of identification for a transaction within the scope of the replication group. The application may request a copy of the transaction's token at the master site during the execution of the transaction. Later, the application running on a client site can use a copy of the token to determine whether the transaction has been applied at that site.

It is the application's responsibility to keep track of the token during the interim. In the web example, the token might be sent to the browser as a "cookie", or stored on the application server in the user's session context.

The operations described here are supported both for Replication Manager applications and for applications that use the replication Base API.