CAP Principle

ECE 454 / 751: Distributed Computing

Instructor: Dr. Wojciech Golab

wgolab@uwaterloo.ca

Slides are derived from online materials, including DataStax documentation:
https://docs.datastax.com/en/cassandra/

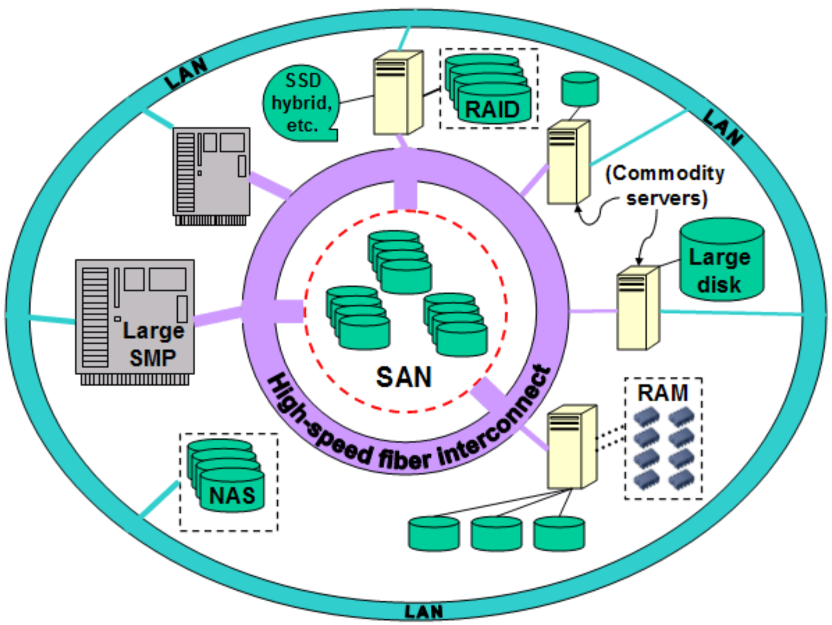
Learning objectives

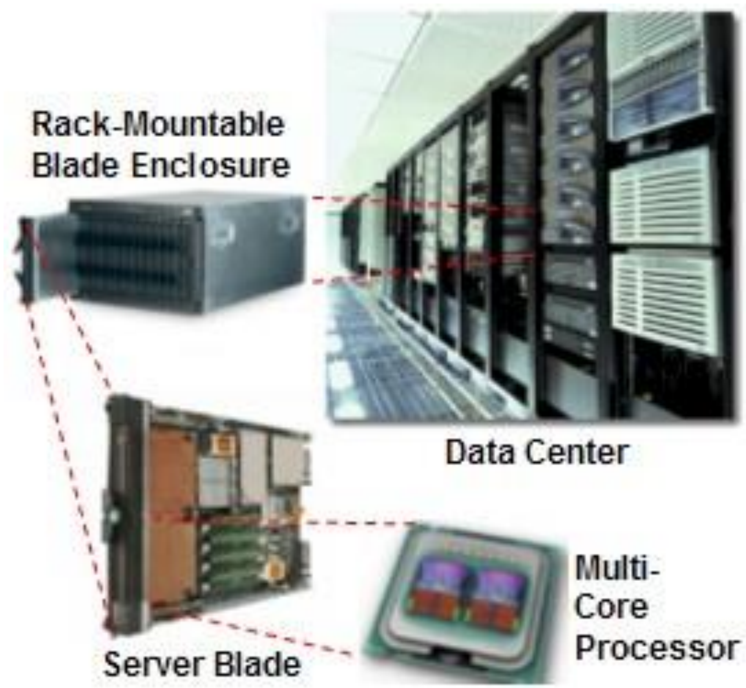
- Understand the impossibility of simultaneously achieving Consistency, Availability, and Partition tolerance.
- Understand the inherent trade-off between latency and consistency.
- Develop a deeper understanding of AP systems.

Myth 1: conventional DBs do not scale

Possible to scale <u>up</u> by adding memory, storage, cores.

Possible to scale <u>out</u> by adding replicas (for read-only queries).





source: http://www.boic.com/scalability.htm

Myth 2: transactions do not scale

True if all updates are processed by one node (e.g., the primary or master), but other designs are also possible:

- **multi-master replication** – requires careful conflict resolution (e.g., two servers try to sell the last dress shirt)
- **partitioning** or **sharding** – requires distributed transactions

Case in point: Google's Spanner.

- data partitioned across multiple data centers
- Paxos-based replication
- two-phase commit for distributed transactions
- SQL-based query language
(SQL supported directly in F1 DB, which is built on top of Spanner)

Reality: no free lunch

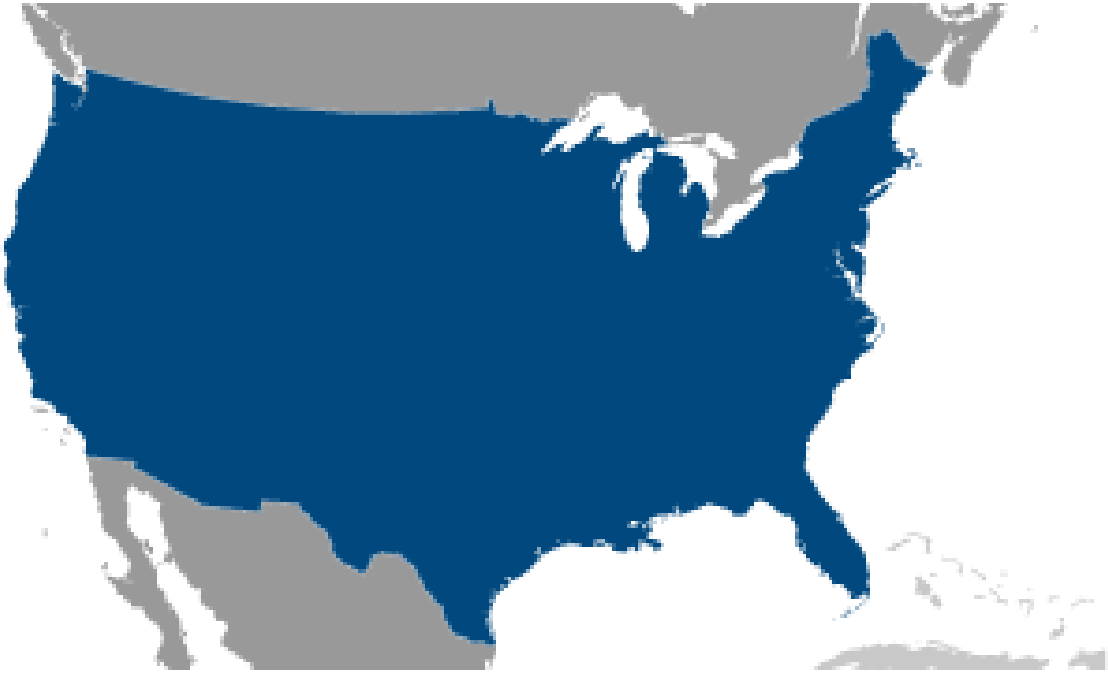
operation	latency (ms)		count
	mean	std dev	
all reads	8.7	376.4	21.5B
single-site commit	72.3	112.8	31.2M
multi-site commit	103.0	52.2	32.1M

Table 6: F1-perceived operation latencies measured over the course of 24 hours.

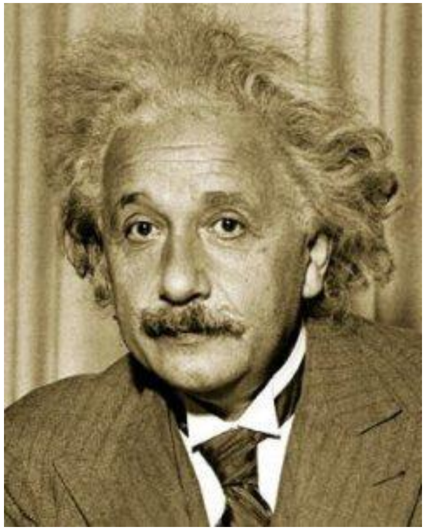
source: Corbett et al., OSDI 2012

(configuration: 5 replicas – 2 US East Coast + 3 US West Coast)

Reality: no free lunch



source: CIA world fact book online

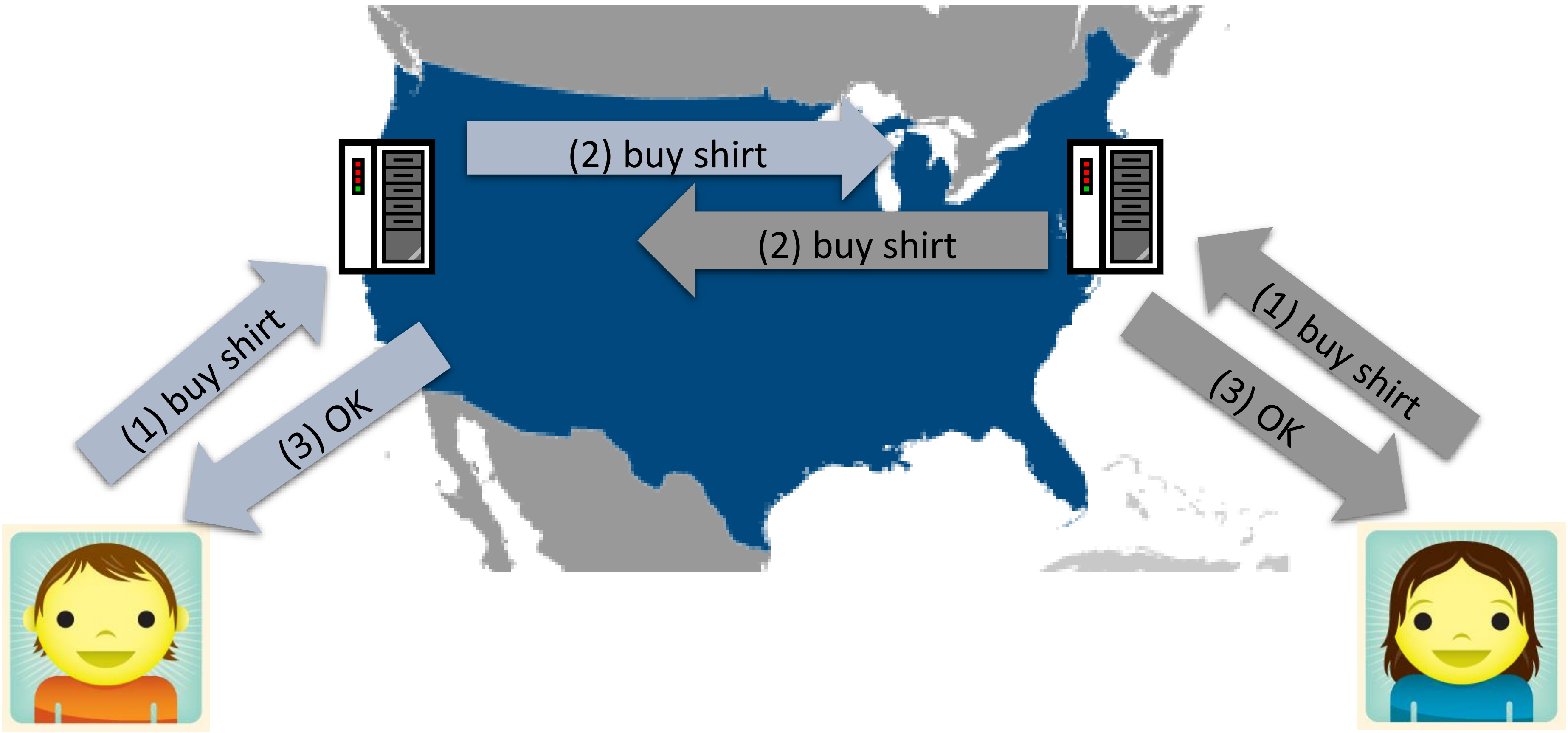


> 4000km (one way), > 20ms at speed of light in optical fibre

Users and workloads today

- Popular websites and services have a global user base.
- Users both read and update data regularly.
- What looks like a single update in the user interface may trigger multiple updates in the back end services. (Example: status change in a social networking app may show up in the newsfeeds of multiple friends.)
- Users are sensitive to latencies above a few hundred ms. Research has shown that Amazon loses 1% revenue for every 100ms increase in latency (source: Greg Linden, 2006).

Myth 3: scalability implies high latency



Consistency, Availability, Partition-Tolerance

Dr. Eric Brewer

Brewer's conjecture:

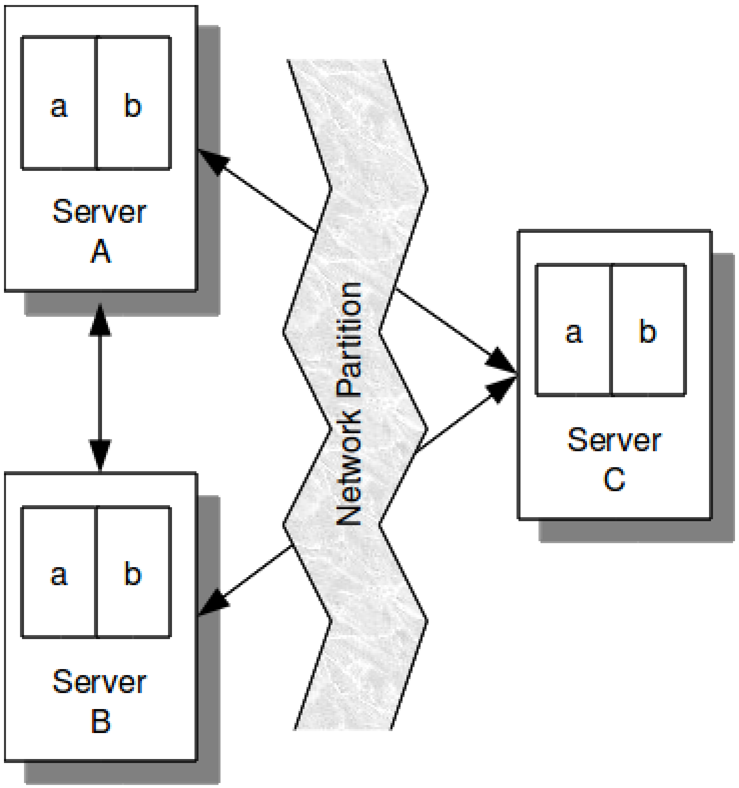
It is impossible to attain all three of the following properties simultaneously in a distributed system:

- **Consistency** – clients agree on the latest state of the data.
- **Availability** – clients able to execute both read-only queries and updates.
- **Partition tolerance** – system continues to function if the network fails and nodes are separated into disjoint sets.

What's a (wide area) network?

source: https://azure.microsoft.com/en-us/blog/how-microsoft-builds-its-fast-and-reliable-global-network/

What’s a network partition?



source: https://docs.voltdb.com/graphics/NetworkPartition.png

Consistency, Availability, Partition-Tolerance

- A more precise interpretation of the CAP principle is that **in the event of a partition (P), the system must choose either consistency (C) or availability (A), and cannot provide both simultaneously**. On the other hand, during failure-free operation a system may be simultaneously highly available and strongly consistent.
- **CP system:** in the event of a partition, choose **C** over **A.** Example: distributed ACID database.
- **AP system:** in the event of a partition, choose **A** over **C.** Example: eventually consistent system with hinted handoff (discussed later in this lecture module).

AP systems in the real world

- Appropriate for latency-sensitive, inconsistency-tolerant applications: shopping carts, news, social networking, real-time data analytics, online gaming.
- Characteristics: data accessed mostly using get/put operations, no transactions.
- Some implementations support SQL-like query languages that lack the powerful features of the relational model (e.g., joins).

What is C in CAP?

Examples of consistency models in CP systems:

- serializability
- linearizability
- sequential consistency
- $N_R + N_W > N$

Examples of consistency models in AP systems:

- eventual consistency
- causal consistency

CAP vs. PACELC

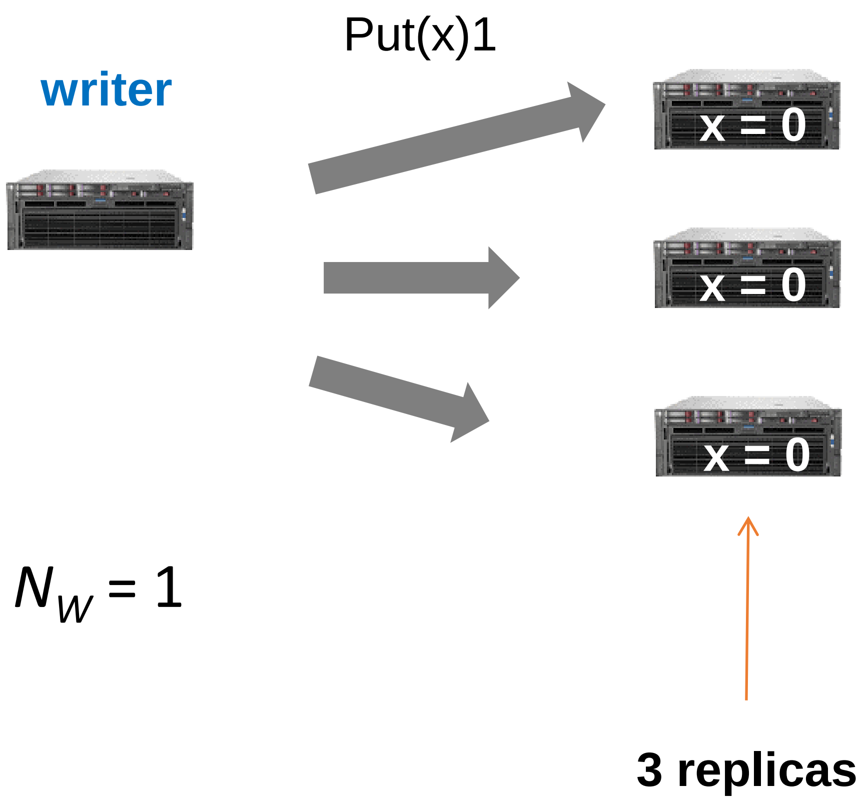
Dr. Daniel Abadi (Yale University) re-formulated CAP to give a more complete portrayal of the space of potential consistency tradeoffs. His formulation is called PACELC (pronounced "pass-elk").

- If there is a network **P**artition then choose between
  - **A**vailability and
  - **C**onsistency
- **E**lse choose between
  - **L**atency and
  - **C**onsistency

Tunable consistency

- If clients read and write overlapping sets of replicas (defined as $N_R + N_W > N$ in an earlier lecture module), then every read is guaranteed to observe the effects of all writes that finished before the read started. This is known as **strong consistency in the context of key-value storage systems**. Example: $N_R = 1$, $N_W = 3$, $N = 3$.
- The partial quorums for reads and writes can be determined in some key-value storage systems on a per-request basis using **client-side consistency** settings, leading to **tunable consistency**. Apache Cassandra supports a variety of such settings including ONE, QUORUM (i.e., majority), and ALL for reads and writes.

Review: quorum-based replication in action



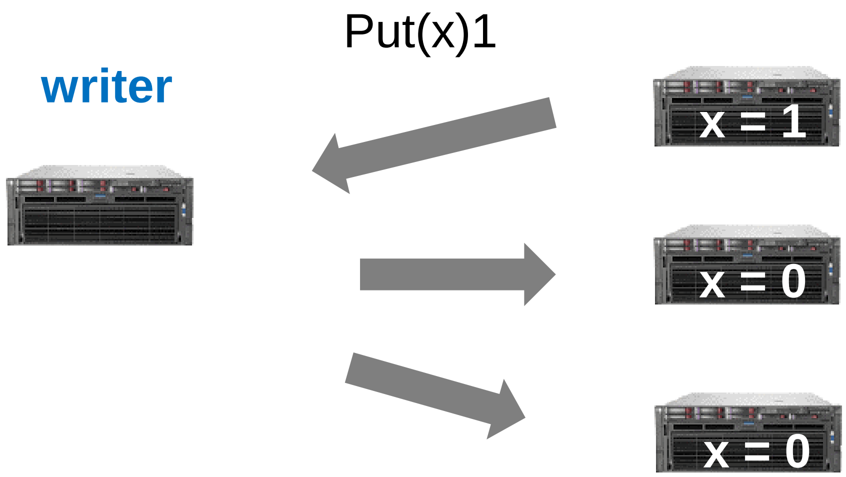
x denotes a data object

(e.g., row identified by a primary key)

initially x = 0

Note: the "writer" is a coordinator process inside the storage system that executes the storage operation on behalf of the client application, which his not shown.

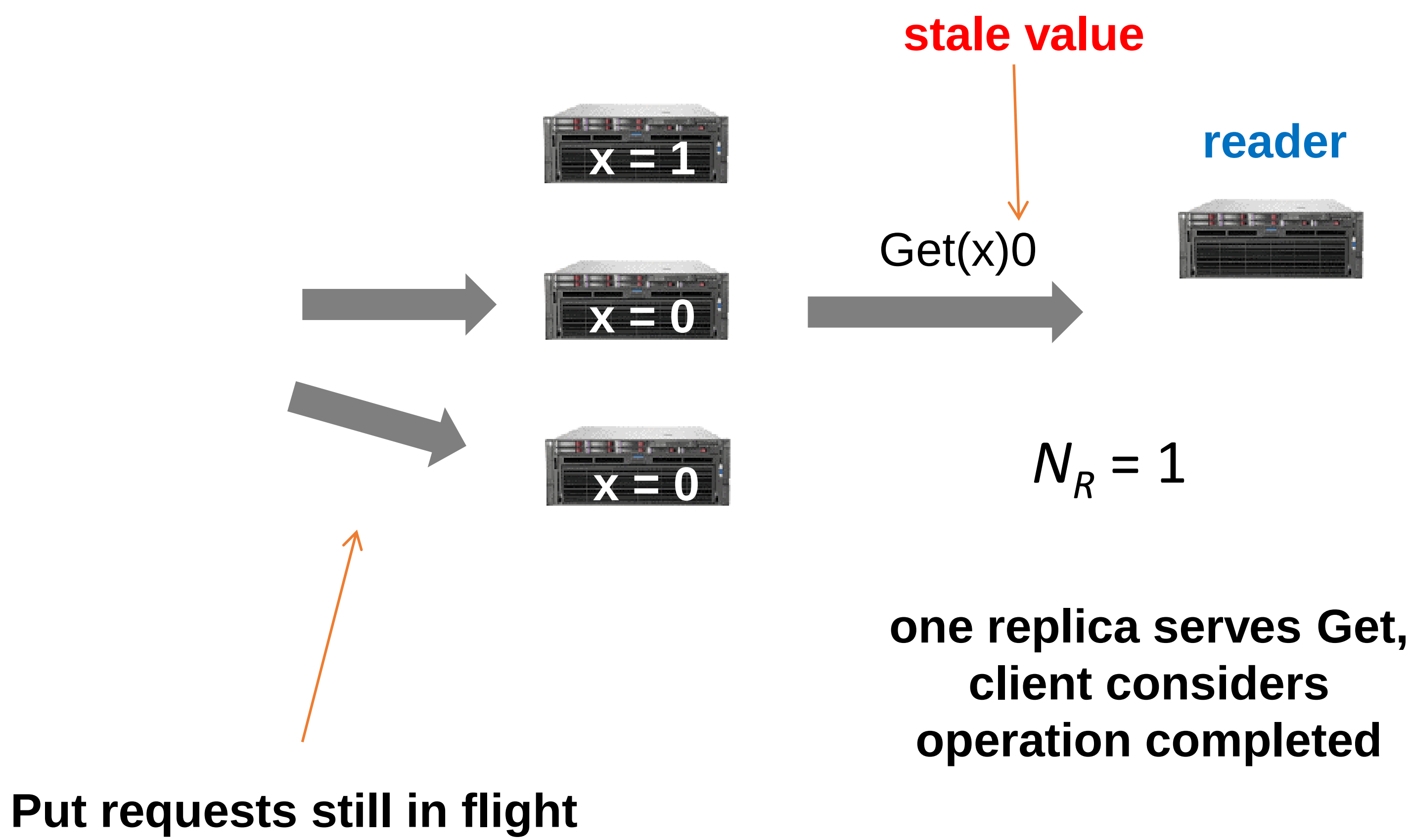
Review: quorum-based replication in action



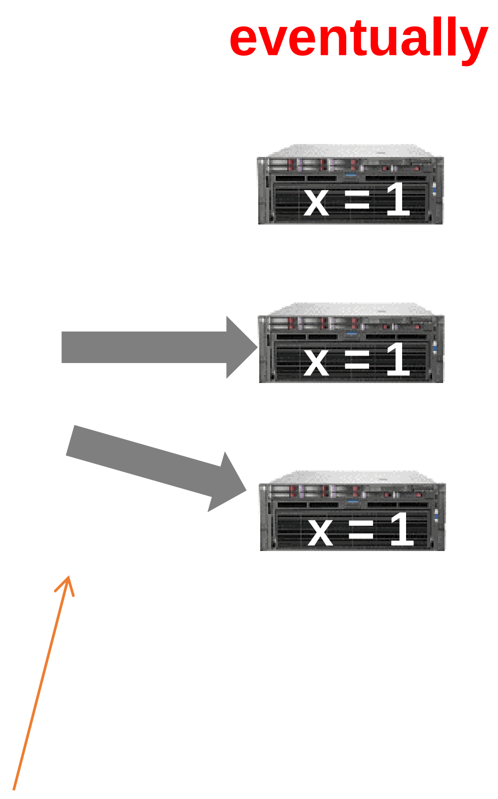
$N_W = 1$

<u>one</u> replica acknowledges Put, client considers operation completed

Review: quorum-based replication in action



Review: quorum-based replication in action

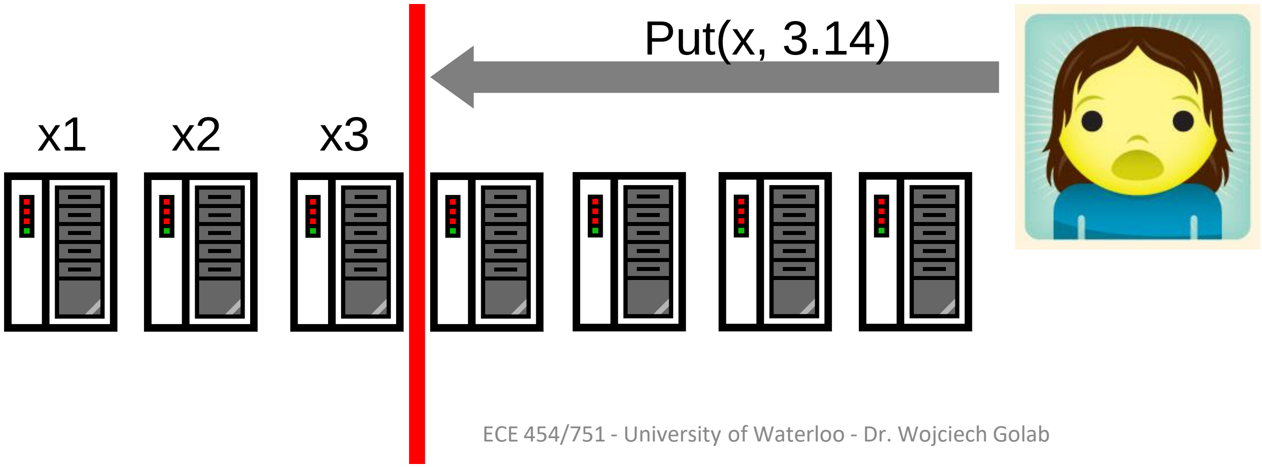


Recall:
Write-write conflicts are resolved using a concurrency control mechanism not shown in these slides. For example, updates can be tagged with timestamps obtained from a global clock.

Put requests applied at remaining replicas

Client-side consistency settings vs. CAP

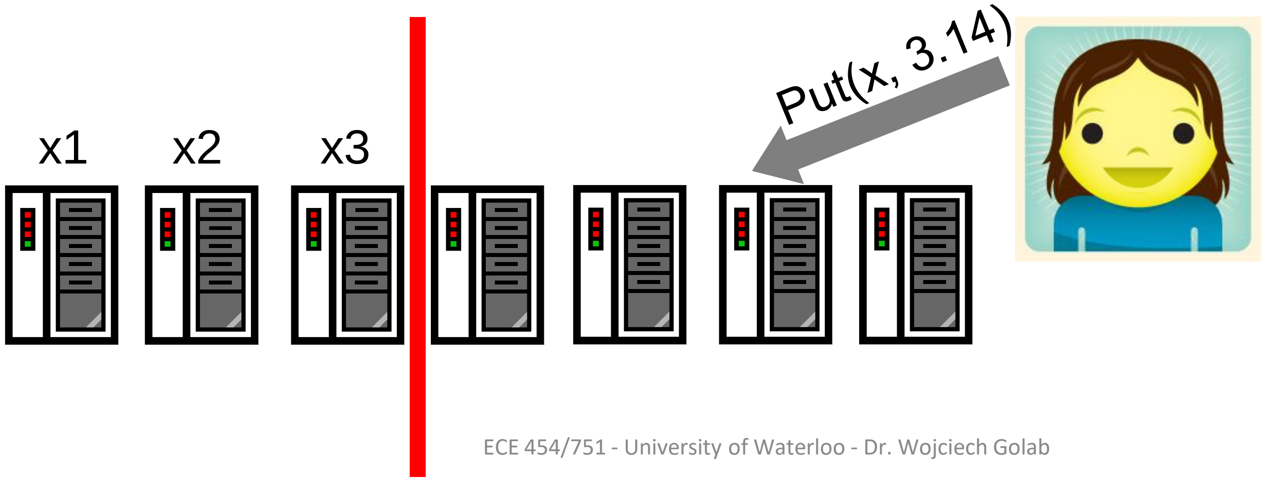
- "Strong consistency" ($N_R + N_W > N$) is considered C in the context of the CAP principle.
- What about when clients read and write only one out of N>1 replicas? Do we obtain a CP system or an AP system?
- Assumption: the N replicas are fixed for a given data object.



Alice is on the majority side of a network partition but cannot access any replica of data object x.

Client-side consistency settings vs. CAP

- In general, read ONE / write ONE settings <u>do not provide AP</u> !
- Solution: use **sloppy quorums**, in which the set of replicas (i.e., partial quorum) can change dynamically.
- Example: In Apache Cassandra, **hinted handoff** allows an arbitrary node to accept an update for a given key. This surrogate node holds the update until one of the replicas becomes available. Hinted handoff is enabled in Cassandra via **write ANY** consistency.



Alice hands off the update temporarily to a surrogate node.

Client-side consistency settings vs. CAP

- **Note 1:** The example in the previous slide assumes partial replication. If a quorum-replicated storage system uses full replication then hinted handoff is not necessary.
- **Note 2:** Hinted handoff ensures write availability, but not read availability.

Case study: Cassandra - overview

- Quorum-replicated key-value store supporting tunable consistency with (optional) full write availability. Initially developed at Facebook, later open-sourced under Apache license.
- Schema:
  - **keyspace:** a name space for column families
  - **column family:** similar to a database table, each column has a name/value/timestamp
  - each row identified uniquely by a **row key** (think primary key)
  - **sparse-column storage** engine: for a given row, only the columns present are stored (no need to store NULL values)
  - columns can be indexed using hash-like structures
- SQL-like language called **CQL** in recent versions, but no joins or foreign keys.

Case study: Cassandra - consistency

- ONE: N_R or N_W = 1
- ANY: for writes only, like ONE but uses hinted handoff if needed
- TWO: N_R or N_W = 2
- THREE: N_R or N_W = 3
- QUORUM: N_R or N_W = ceiling[(N+1)/2]
- ALL: N_R or N_W = N
- LOCAL_ONE/LOCAL_QUORUM: like ONE/QUORUM but the subset of replicas is chosen from the local data center only
- EACH_QUORUM: for writes only, writes to a quorum in each data center

Note: The Cassandra replication strategy defines the replication factor (N) separately for each data center.

Case study: Cassandra - CQL

- Example: keyspace creation

```
CREATE KEYSPACE demodb WITH REPLICATION = {'class'
: 'SimpleStrategy', 'replication_factor': 3};
```

- Example: table creation

```
CREATE TABLE emp (
    empID int,
    deptID int,
    first_name varchar,
    last_name varchar,
    PRIMARY KEY (empID, deptID));
```

- Note: The size is not specified for varchar columns. The first attribute of primary key (in this case empID) is used for partitioning.

source: http://www.datastax.com/documentation/cql/3.0

Case study: Cassandra - CQL

- Example: insert data into a table

```
INSERT INTO emp (empID, deptID, first_name,
last_name) VALUES (104, 15, 'jane', 'smith');
```

- Example: query data

```
USE demodb;
SELECT * FROM emp WHERE empID IN (130,104) ORDER
BY deptID DESC USING CONSISTENCY QUORUM;
```

- Example: create secondary index

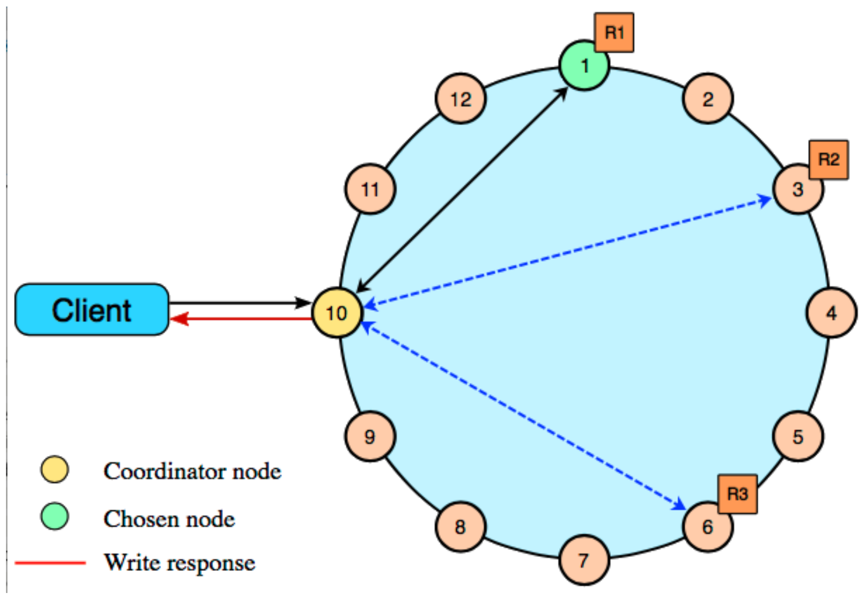
```
CREATE INDEX last_name_index ON emp (last_name);
```

  Note: specify ALLOW FILTERING to take advantage of secondary indices.

source: http://www.datastax.com/documentation/cql/3.0/

Case study: Cassandra - Put

A Put operation is executed on behalf of client by a **coordinator**, which is the node to which the client connects. The coordinator sends the update to all replicas of a row. The consistency level determines only how many acknowledgments the coordinator waits for.



source:
https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dmlClientRequestsRead.html

Case study: Cassandra - Get

- Get operations are also executed by a coordinator.
- The coordinator contacts all replicas of a row using two types of requests:
  - **direct read request** – retrieves data from the closest replica
  - **digest request** – retrieves a hash of the data from the remaining replicas (coordinator waits for at least N_R - 1 of these to respond)
  - **background read repair request** – sent if a discrepancy is detected among the hashes reported by different replicas, tells the replica to obtain the latest value
- **Note 1:** The coordinator uses timestamps to determine which replica has the latest data.
- **Note 2:** The coordinator replies to the client after receiving N_R replies from the storage nodes, and determining the latest value. Read repair occurs in the background.