Syllabus

- Introduction to fault-tolerant distributed systems
- Models of distributed systems and related faults
- **Data replication**
- Distributed consensus
- State machine replication
- Database replication

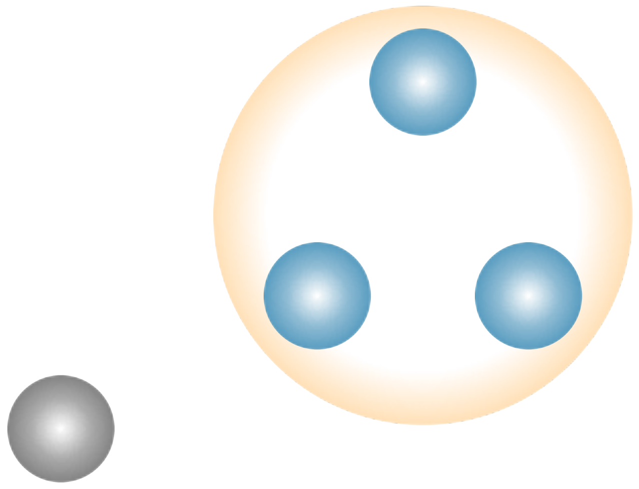
Data replication

ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes | replicas (concurrency model)
  - Crash faults (fault model)

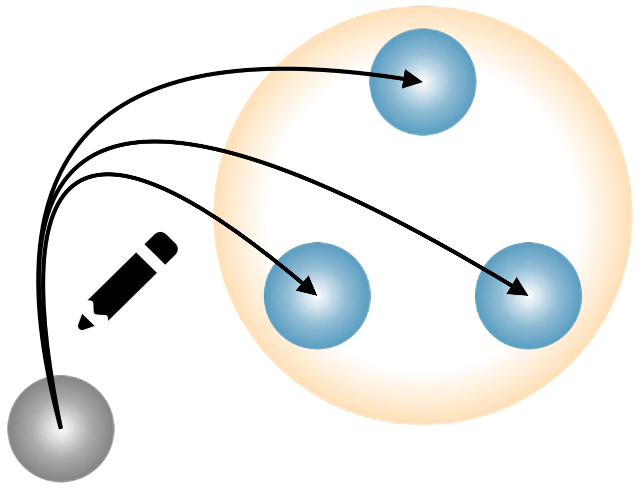
Data replication
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes | replicas (concurrency model)
  - Crash faults (fault model)
- Read-One-Write-All (ROWA)



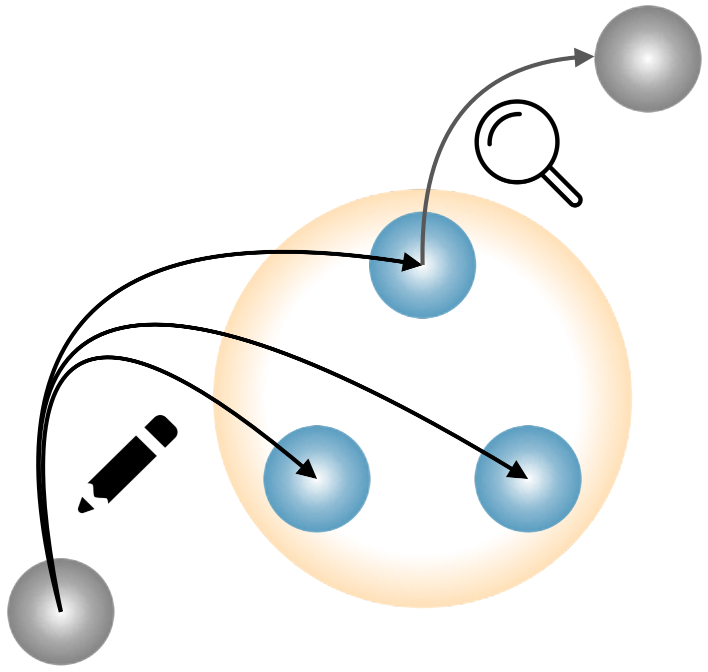
Data replication
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes | replicas (concurrency model)
  - Crash faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas



Data replication
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes | replicas (concurrency model)
  - Crash faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas
  - Read can be performed on any replica

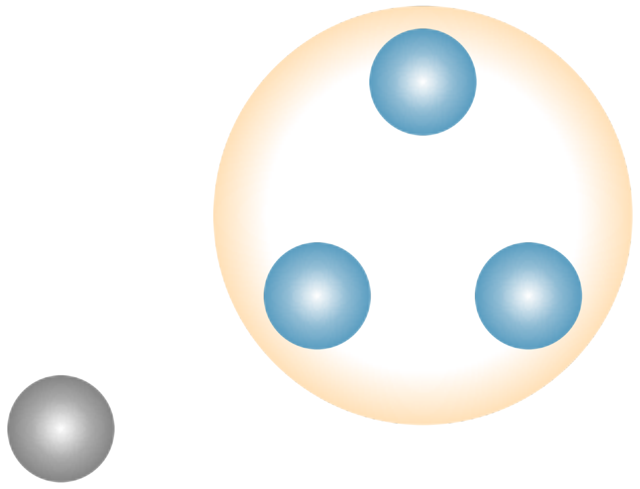


Data replication
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - No faults (fault model)

Data replication
ROWA

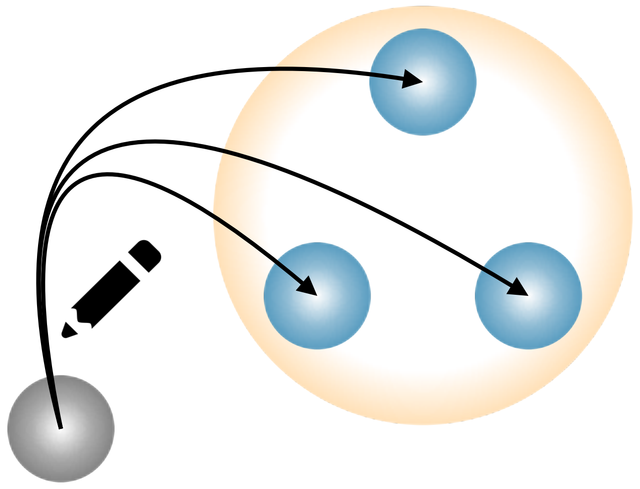
- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - No faults (fault model)
- Read-One-Write-All (ROWA)



Data replication

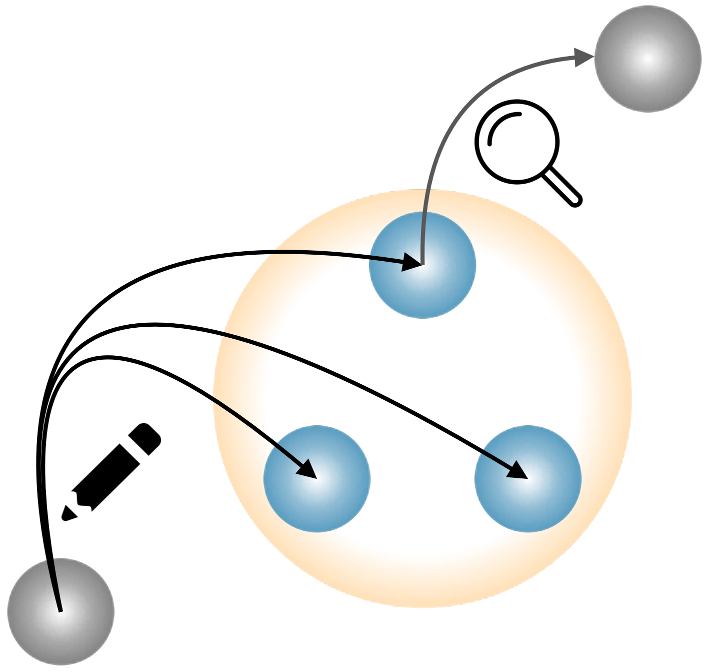
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - No faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas



Data replication
ROWA

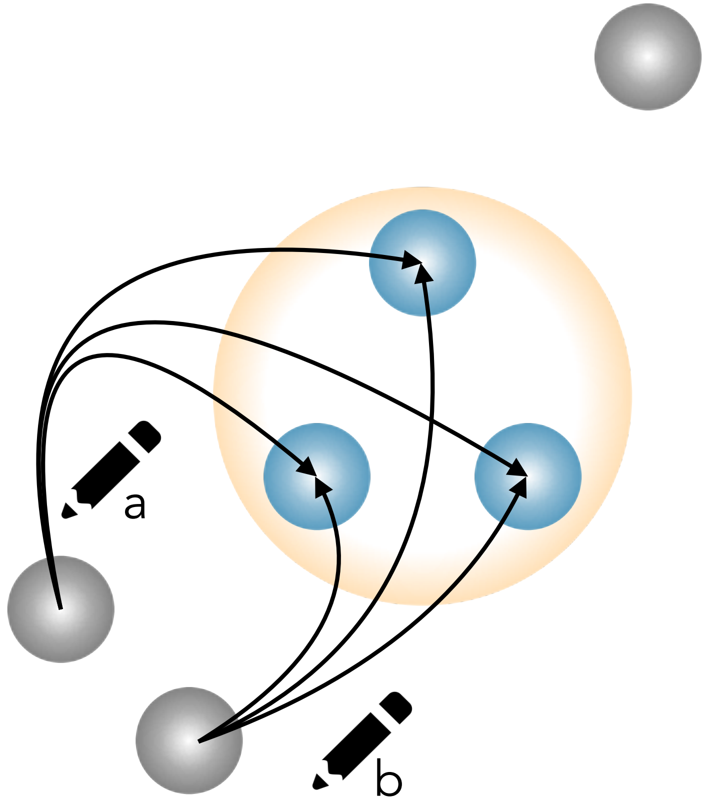
- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - No faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas
  - Read can be performed on any replica



Data replication

ROWA

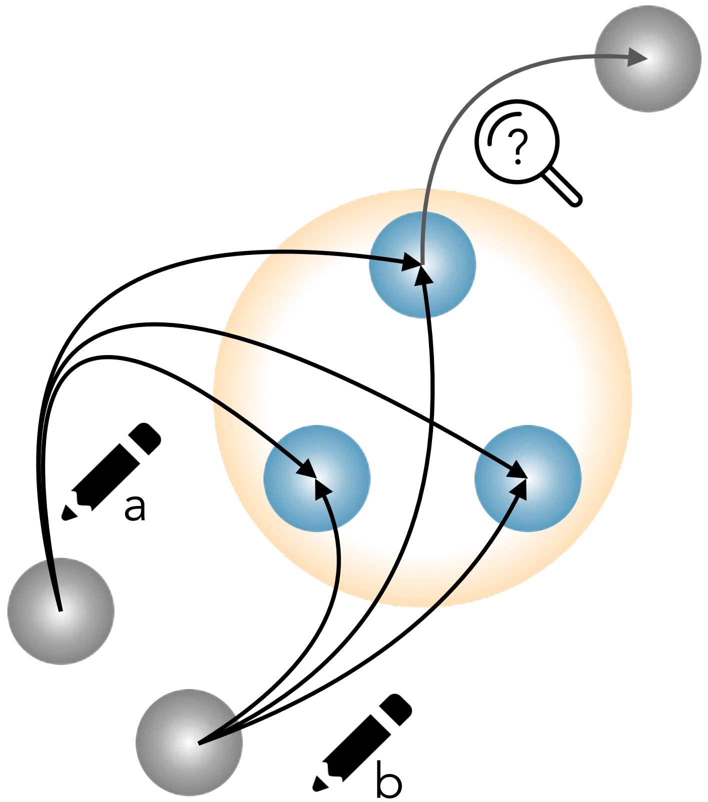
- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - No faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas
  - Read can be performed on any replica
- Concurrent writes



Data replication

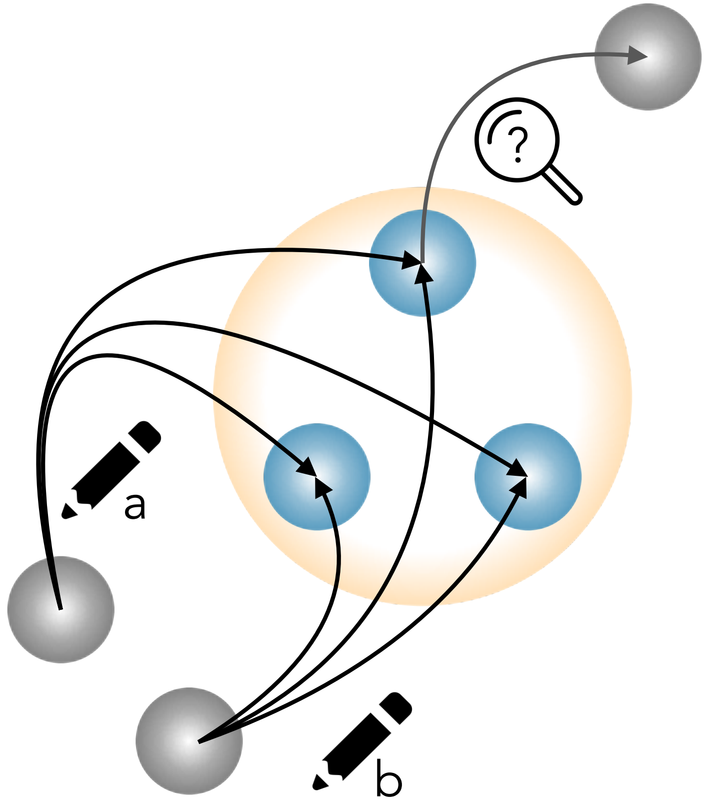
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - No faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas
  - Read can be performed on any replica
- Concurrent writes
  - What does a subsequent read return?



Data replication
ROWA

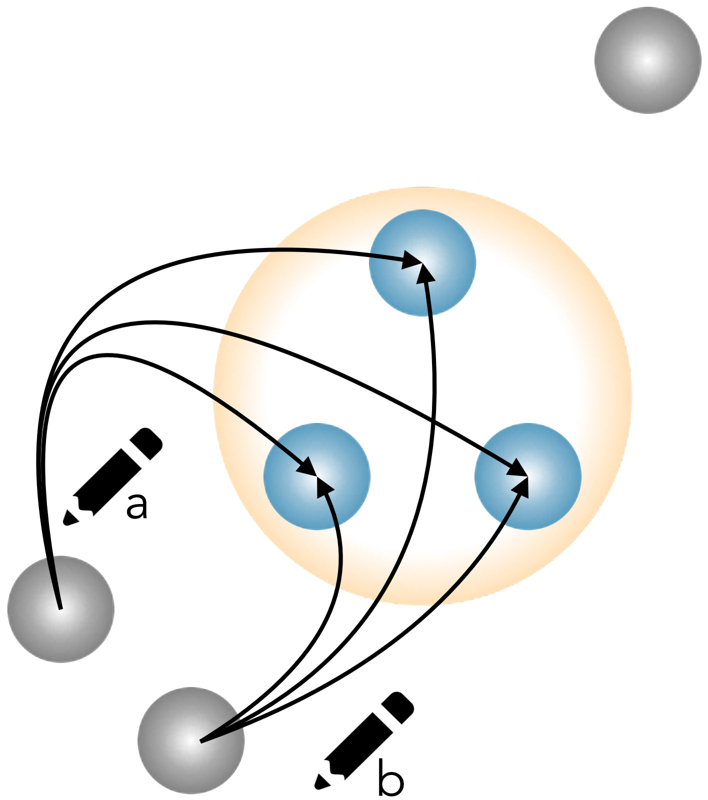
- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - No faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas
  - Read can be performed on any replica
- Concurrent writes
  - What does a subsequent read return?
  - We may end up with replicas writing a → b and others writing b → a 😳



Data replication

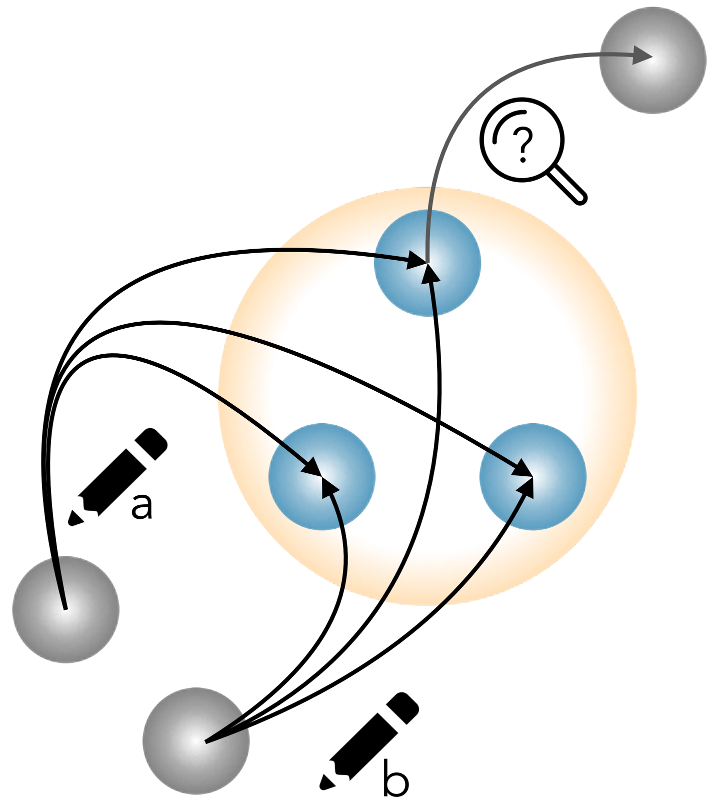
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - Crash faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas
  - Read can be performed on any replica
- Concurrent writes



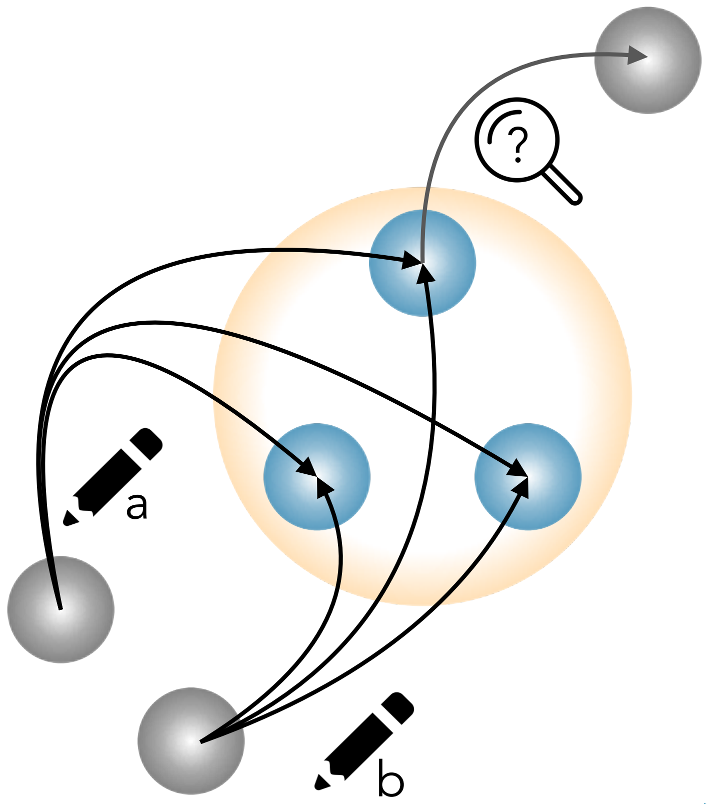
Data replication
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - Crash faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas
  - Read can be performed on any replica
- Concurrent writes
  - What does a subsequent read return?



Data replication
ROWA

- Let us consider the replication of a basic storage service
  - Elementary atomic read and write operations (mutable servers)
  - Multiple sequential client and server processes / replicas (concurrency model)
  - Crash faults (fault model)
- Read-One-Write-All (ROWA)
  - Write *value* to all replicas
  - Read can be performed on any replica
- Concurrent writes
  - What does a subsequent read return?
  - We may end up with replicas writing a and then b and others writing b and then a



Data replication

Data versioning

- How can we prevent stale writes?
- Let us add a *version* to the state of the replicas
  - v_i is replica i's version, initially null
- Write operations now become more complex

At client proxies

```
write (value)
    _read (_, vmax) from some replica
    v_write (value, vmax+1) to all replicas
```

At replica i

```
_write (value, version)
    if version > vi then
        xi = value
        vi = version
```

- How to define version? Is a simple scalar enough?
- Eg. version = (counter, pid)

```
vi > vj ::
    vi.counter > vj.counter OR
    vi.counter = vj.counter AND vi.pid > vj.pid
```

Data replication

Quorums

- ROWA is not fault tolerant
- Let us introduce Quorums
  - A quorum is a set of replicas
  - We will refer to two quorums: a write quorum Qw and a read quorum Qr
  - We will write to some Qw set of replicas and will read from some Qr set
  - ROWA is a particular case in which $|Qw| = n$ and $|Qr| = 1$
- If we make $|Qw| < n$ then the replicated system becomes fault tolerant
- Quorum replication requires that:

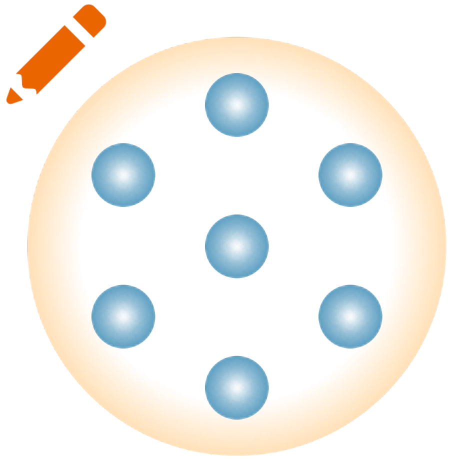
  $|Qr| + |Qw| > n$, read and write quorums always intersect

  $2 * |Qw| > n$, any two write quorums always intersect

Data replication

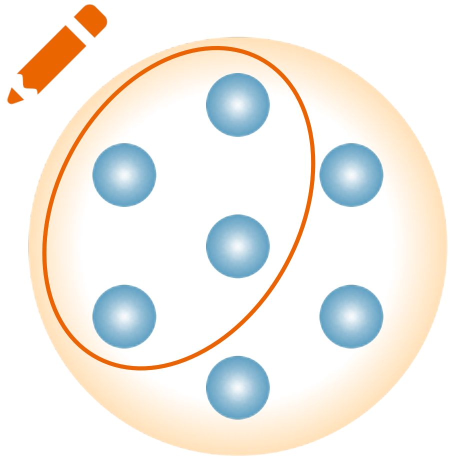
Quorums

- Read and Write operations take `Qr` and `Qw` into account now



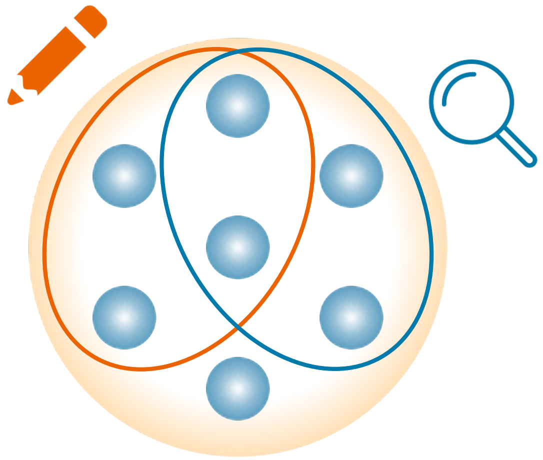
Data replication
Quorums

- Read and Write operations take `Qr` and `Qw` into account now



Data replication
Quorums

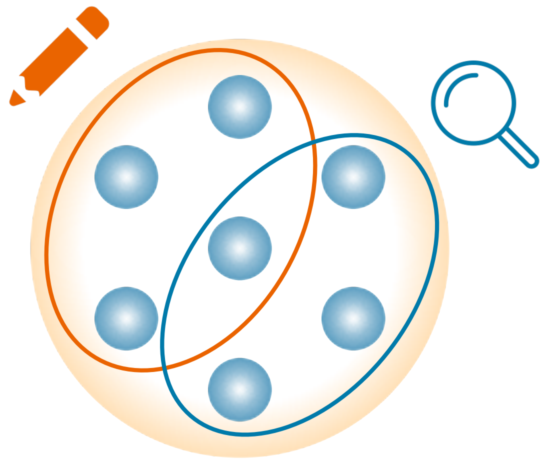
- Read and Write operations take `Qr` and `Qw` into account now



Data replication

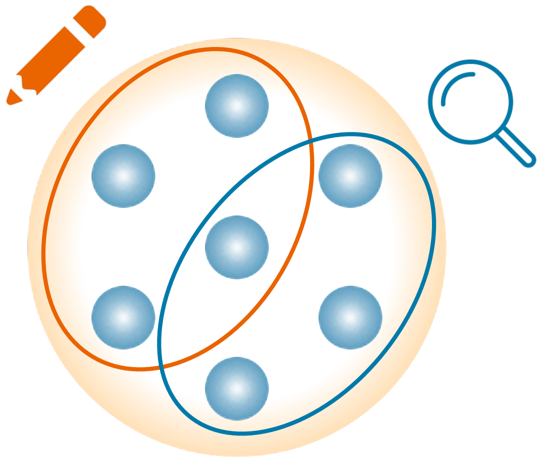
Quorums

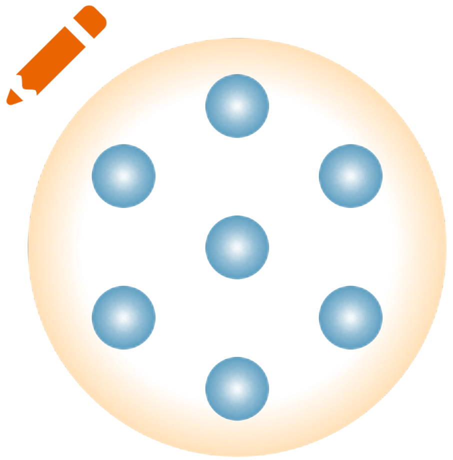
- Read and Write operations take `Qr` and `Qw` into account now



Data replication
Quorums

- Read and Write operations take `Qr` and `Qw` into account now

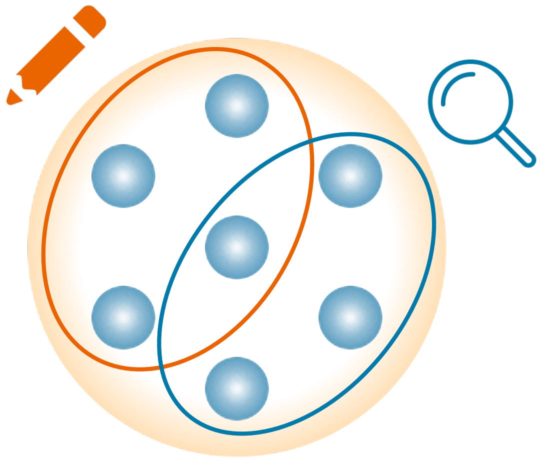


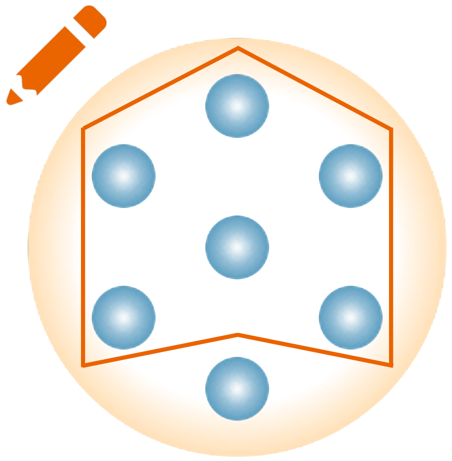


Data replication

Quorums

- Read and Write operations take `Qr` and `Qw` into account now

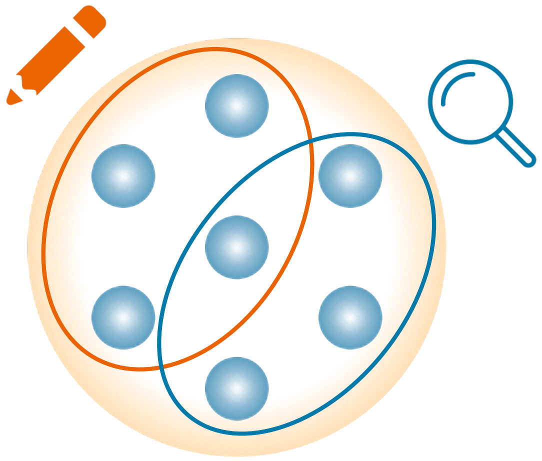


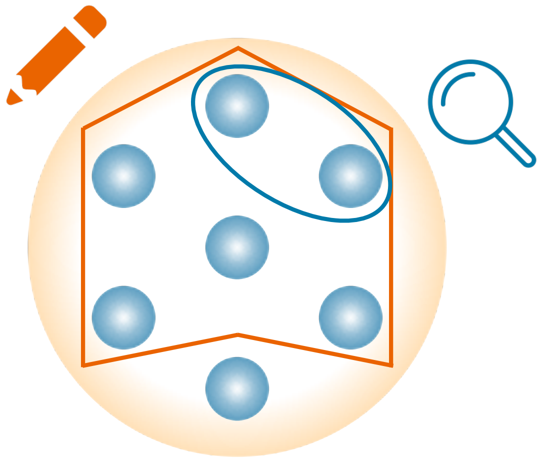


Data replication

Quorums

- Read and Write operations take `Qr` and `Qw` into account now



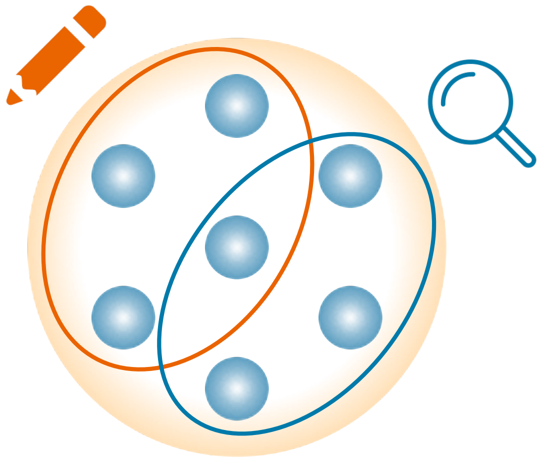


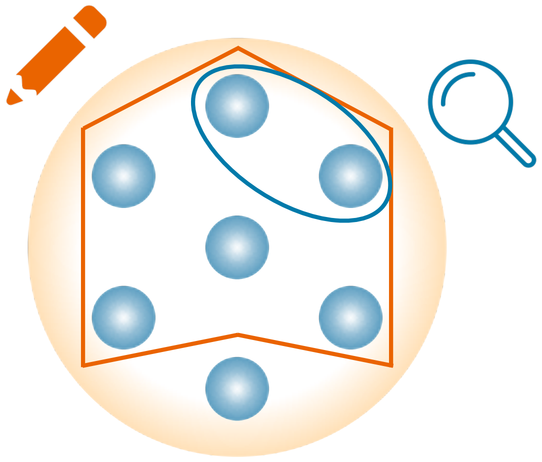
$$|Qr| + |Qw| > n$$

Data replication

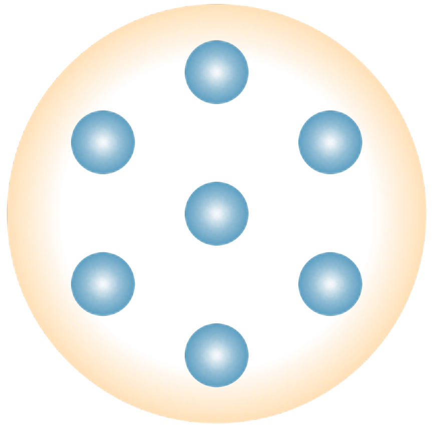
Quorums

- Read and Write operations take `Qr` and `Qw` into account now





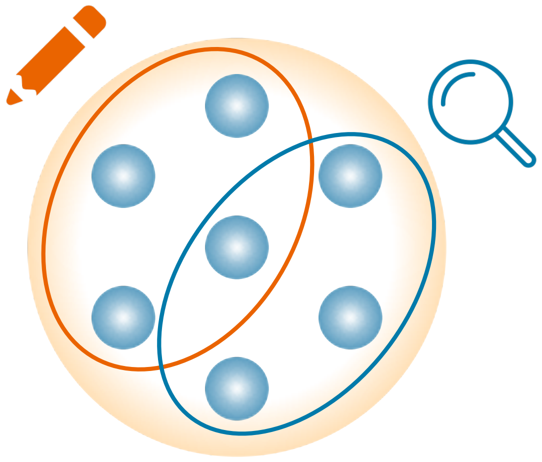
$|Qr| + |Qw| > n$

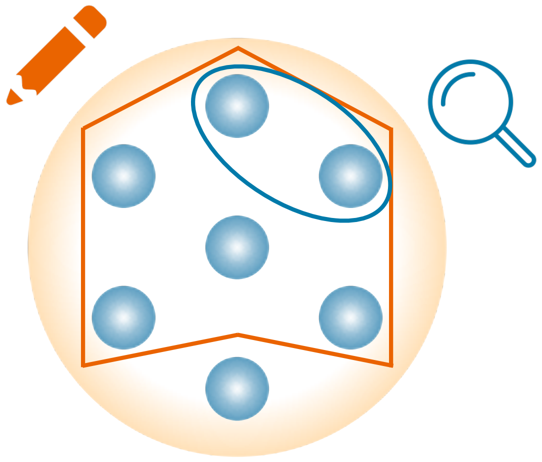


Data replication

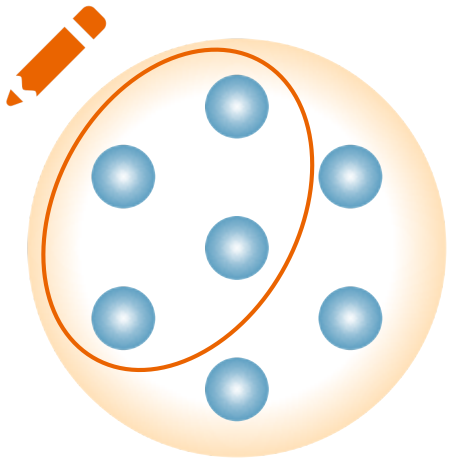
Quorums

- Read and Write operations take `Qr` and `Qw` into account now





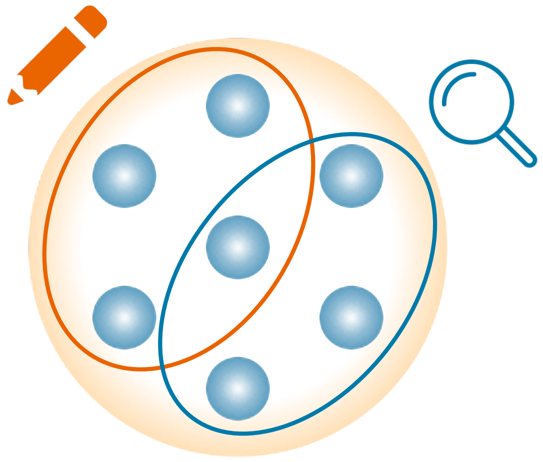
$|Qr| + |Qw| > n$

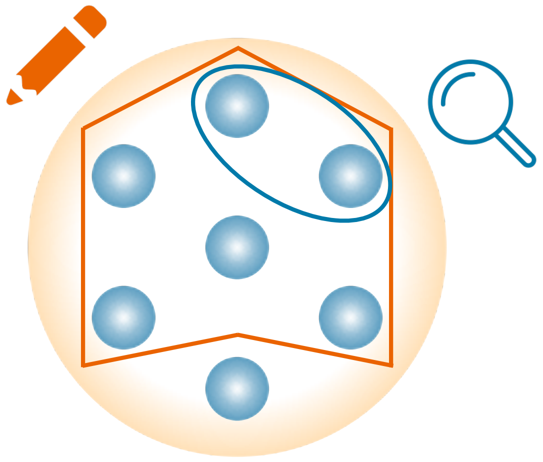


Data replication

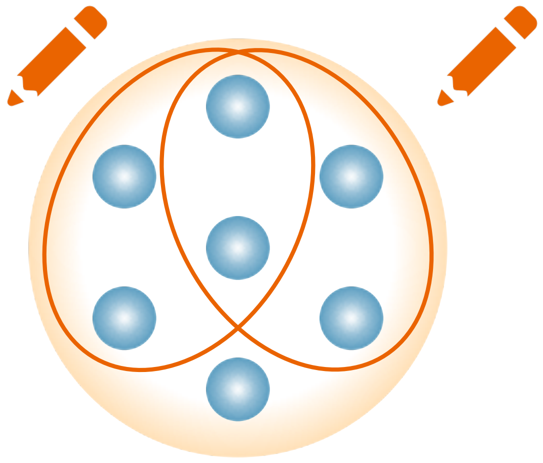
Quorums

- Read and Write operations take `Qr` and `Qw` into account now





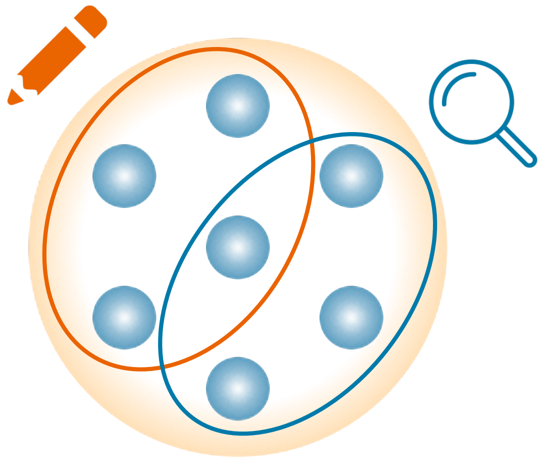
$$|Qr| + |Qw| > n$$

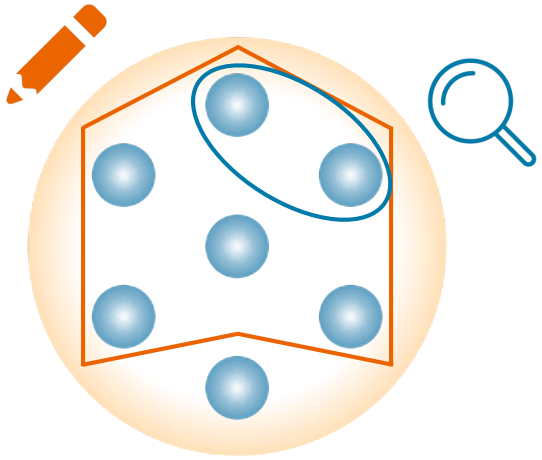


Data replication

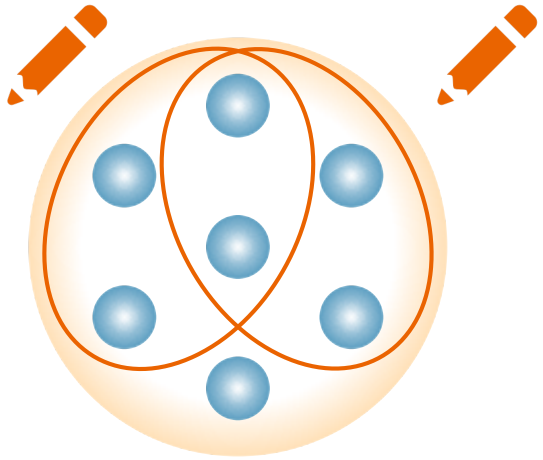
Quorums

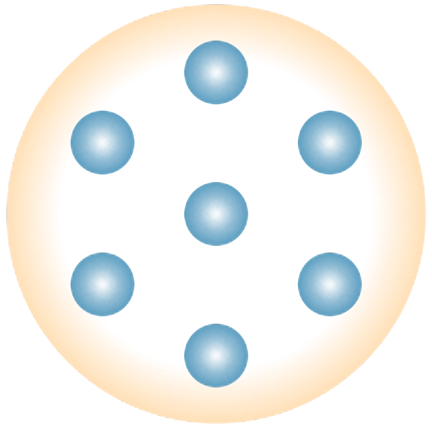
- Read and Write operations take `Qr` and `Qw` into account now





$|Qr| + |Qw| > n$

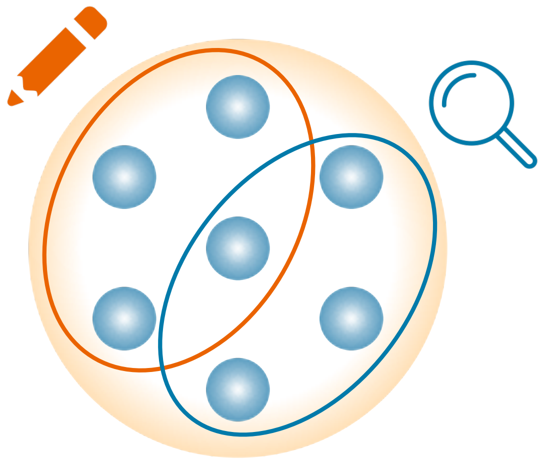


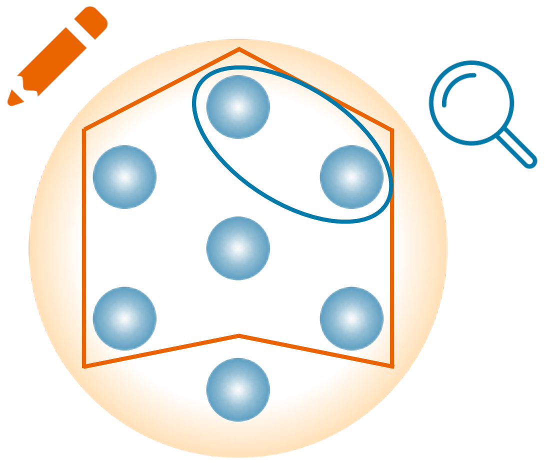


Data replication

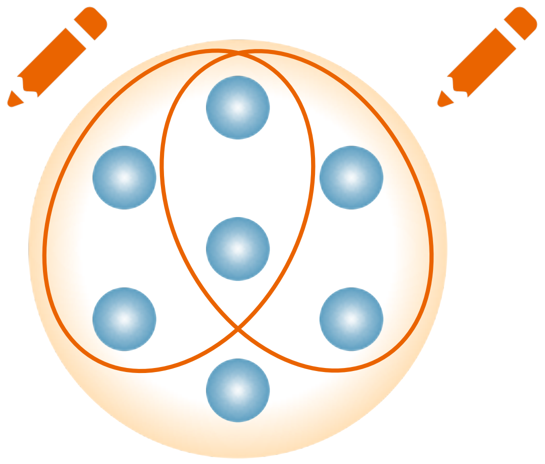
Quorums

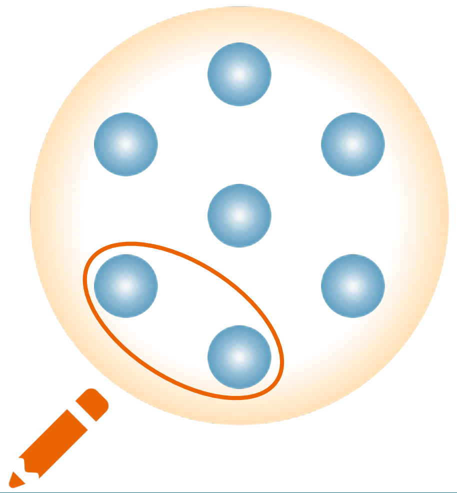
- Read and Write operations take `Qr` and `Qw` into account now





$|Qr| + |Qw| > n$

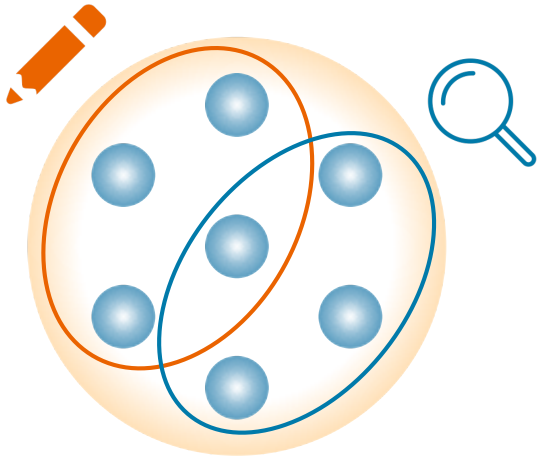


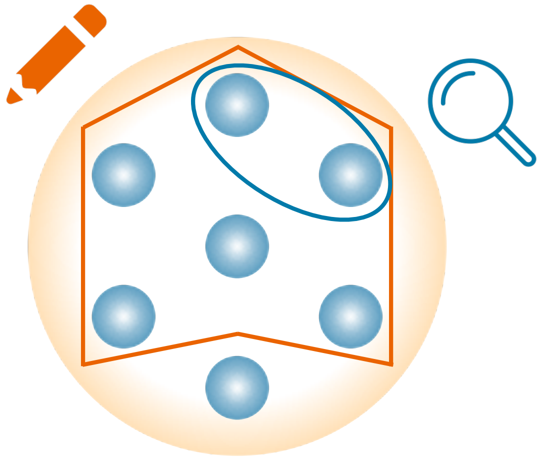


Data replication

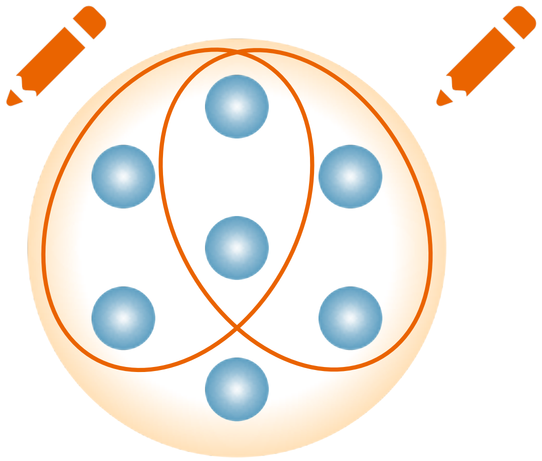
Quorums

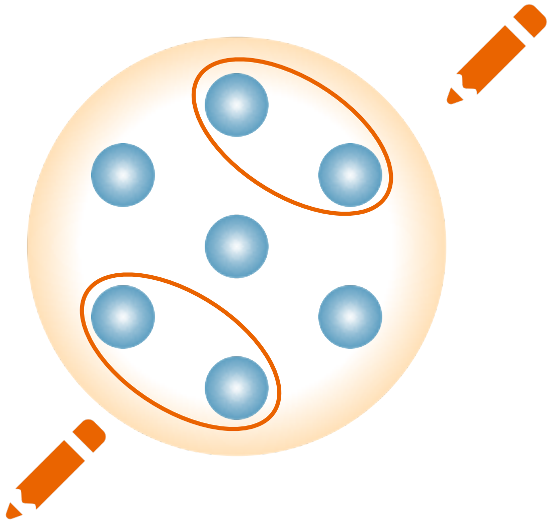
- Read and Write operations take `Qr` and `Qw` into account now





$|Qr| + |Qw| > n$

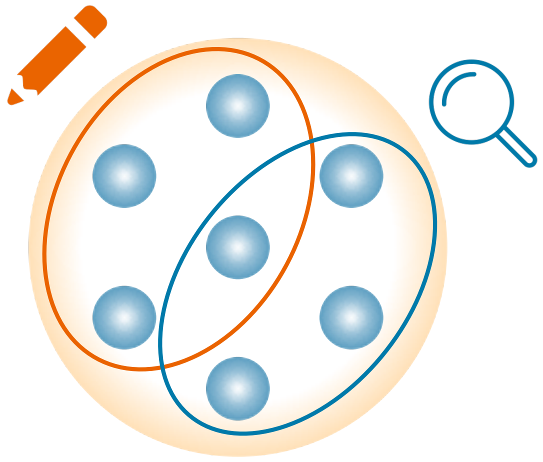


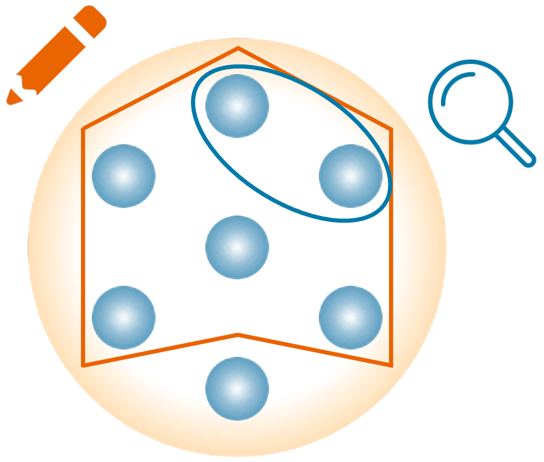


Data replication

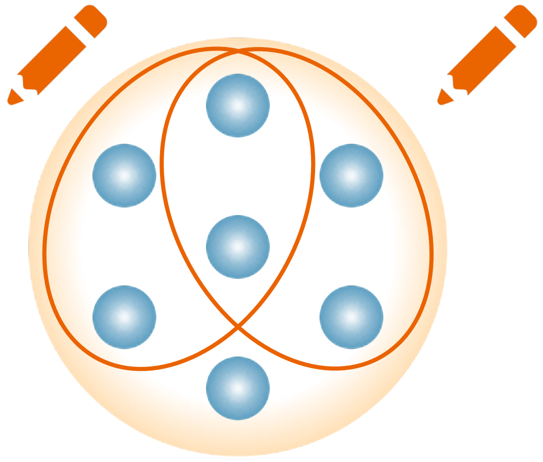
Quorums

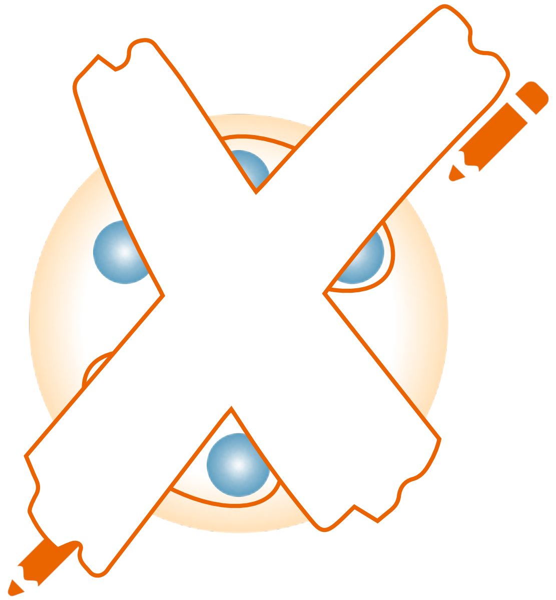
- Read and Write operations take `Qr` and `Qw` into account now





$|Qr| + |Qw| > n$



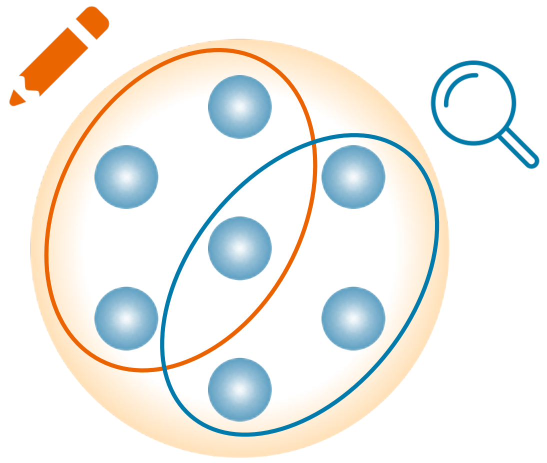


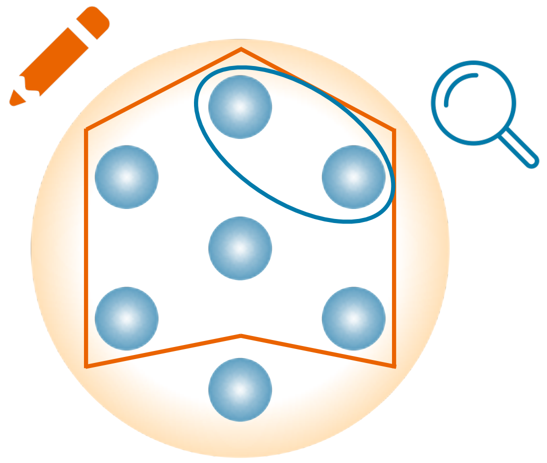
$2 * |Qw| > n$

Data replication

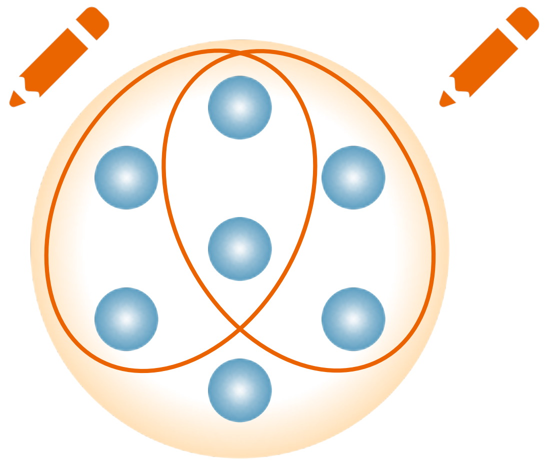
Quorums

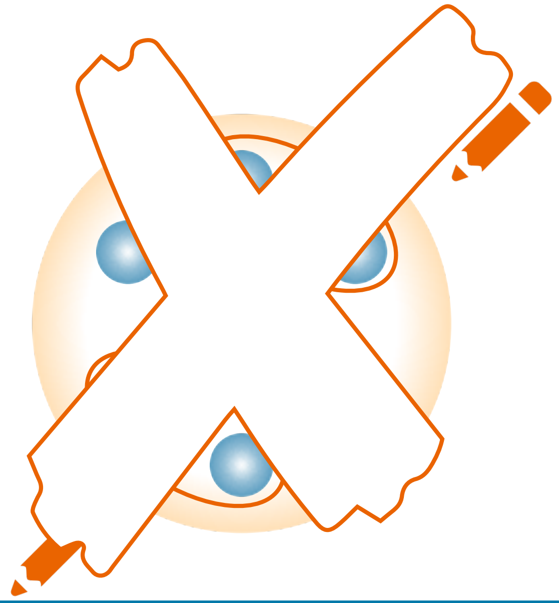
- Read and Write operations take `Qr` and `Qw` into account now





$|Qr| + |Qw| > n$





$2 * |Qw| > n$

Data replication

Quorums

- Read and Write operations take `Qr` and `Qw` into account now
- Write operations now need to read from a `Qr`

At client proxies

```
read (value)
    SetV = _read (x, v) from a Qr
    vmax = largest (_,v) from SetV
    (value, vmax) from SetV

write (value)
    SetV = _read (_, v) from a Qr
    vmax = largest (_, v) from SetV
    _write (value, vmax+1) to a Qw
```

At replica i

```
_read(value, version)
    value = xi
    version = vi

_write (value, version)
    if version > vi then
        xi = value
        vi = version
```

Data replication

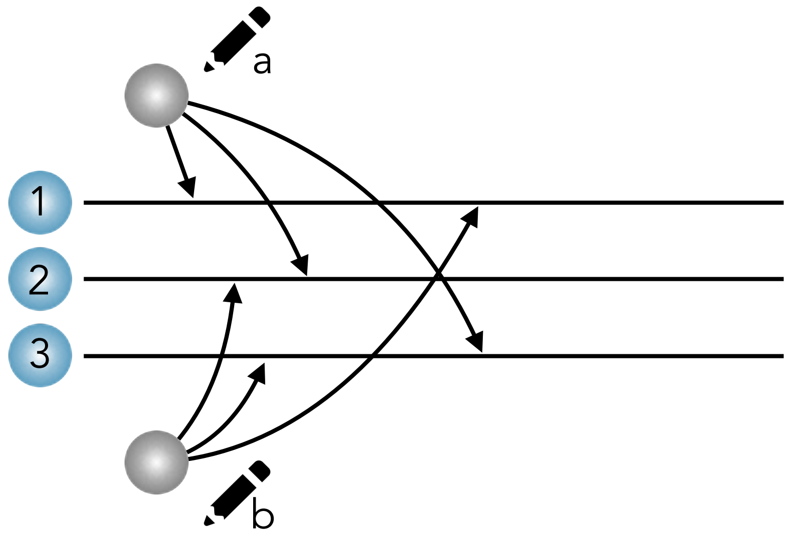
Quorums

- The use of quorums allows for trade-offs in several system aspects
- With omission faults fault tolerance can be maximised using strict majority quorums:
  - $|Qr| = |Qw| = \lceil (n+1)/2 \rceil$
  - $|Qw| = \lceil (n+1)/2 \rceil$ also leads to the least expensive write operations
- $|Qr|$ can determine workload bias; it determines the cost of reads and impacts the cost of writes
- For how it may impact throughput, latency, and network load see this chapter's reading material

Data replication

Order of writes

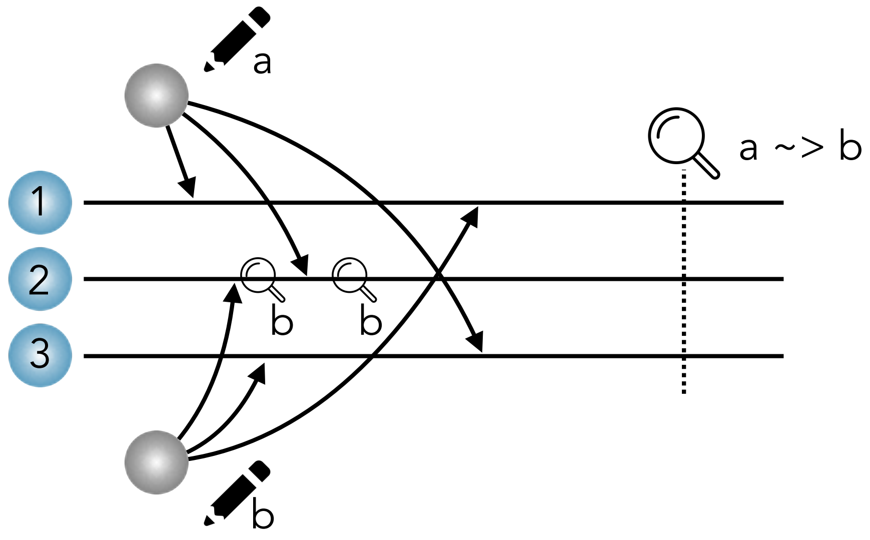
- Monotonic increasing versions + PID's



Data replication

Order of writes

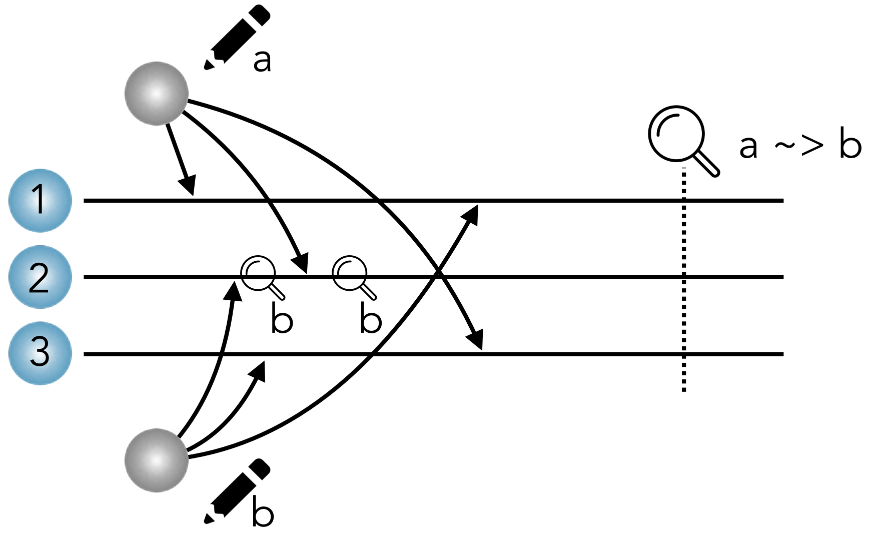
- Monotonic increasing versions + PID's

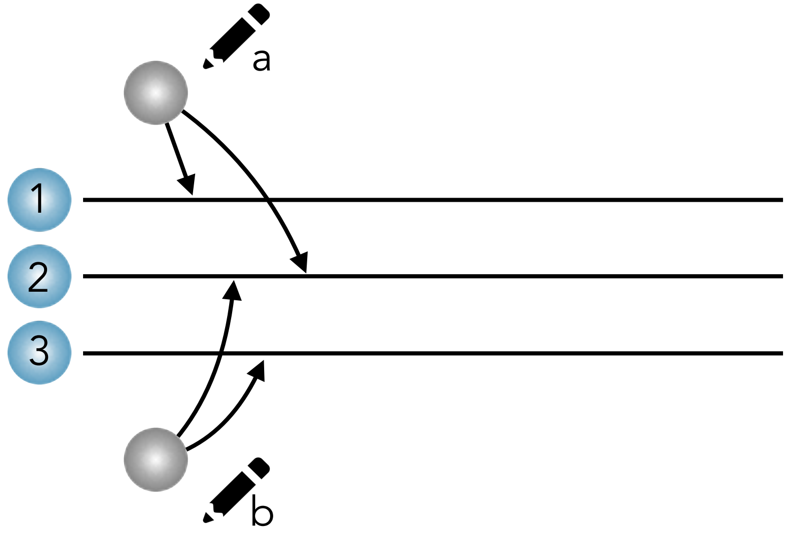


Data replication

Order of writes

- Monotonic increasing versions + PID's

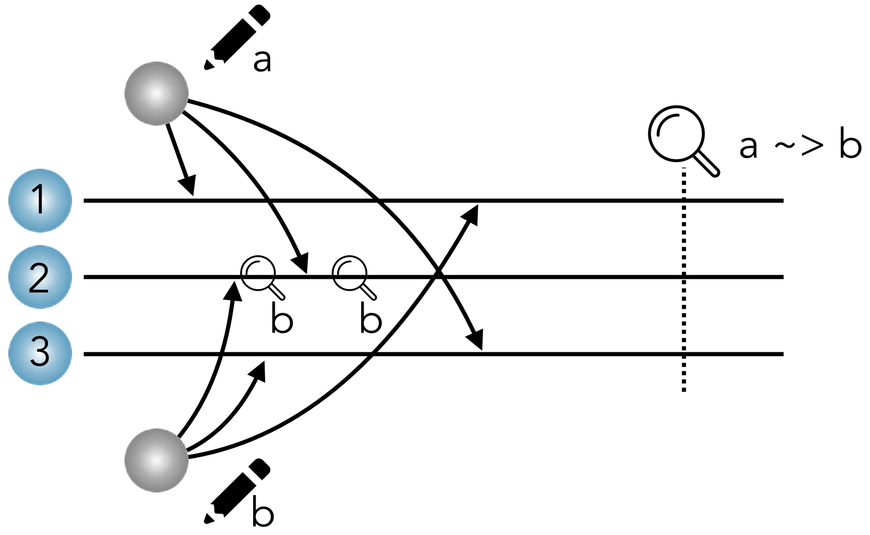


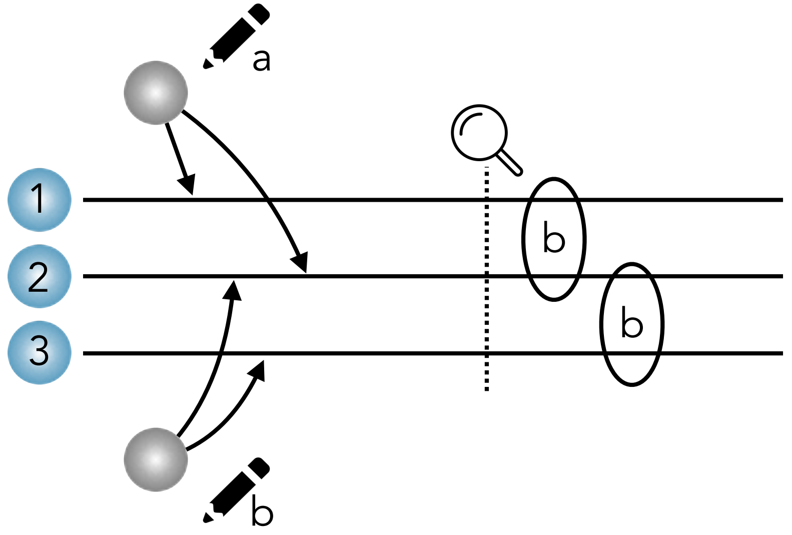


Data replication

Order of writes

- Monotonic increasing versions + PID's

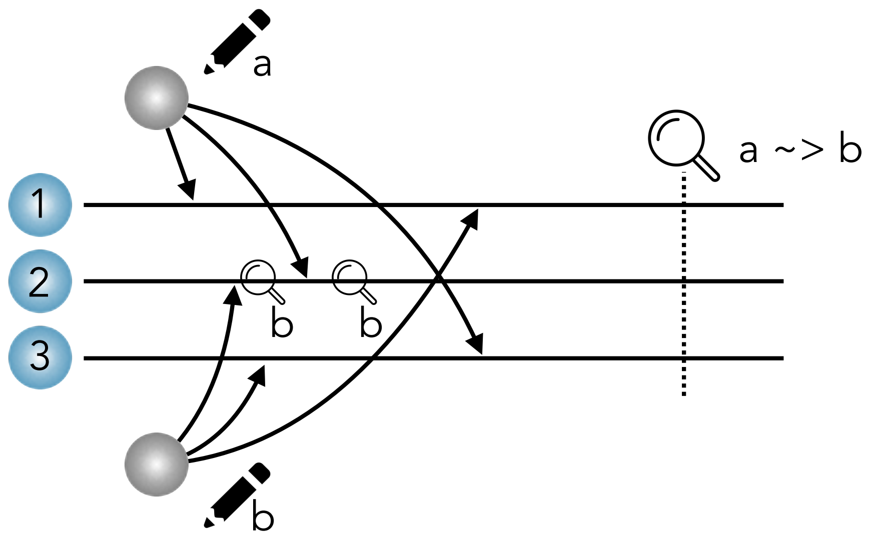


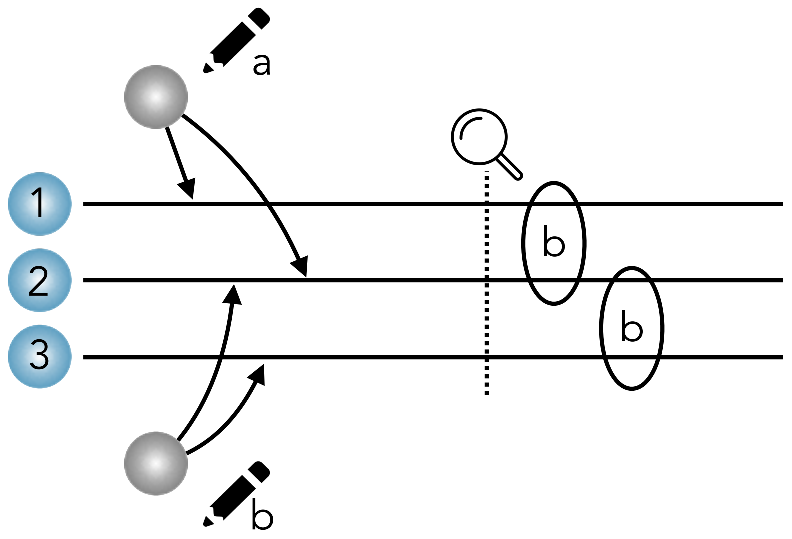


Data replication

Order of writes

- Monotonic increasing versions + PID's



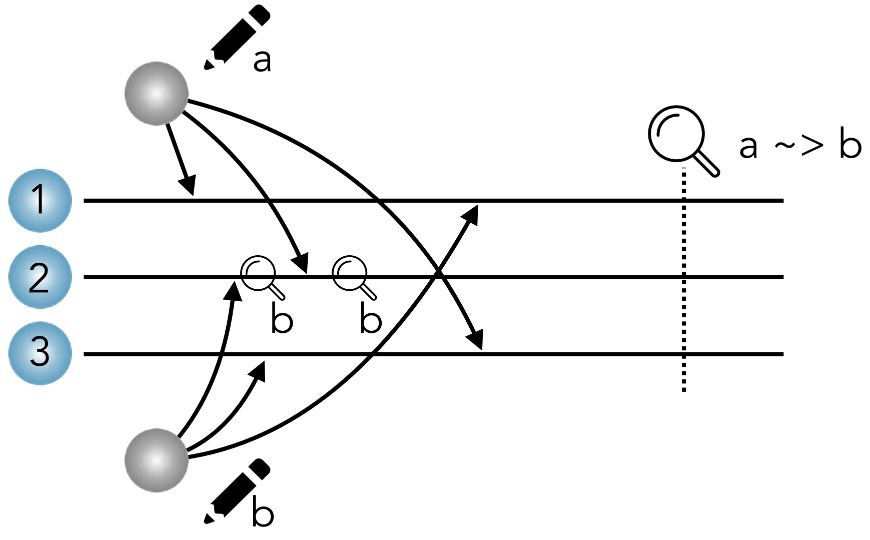


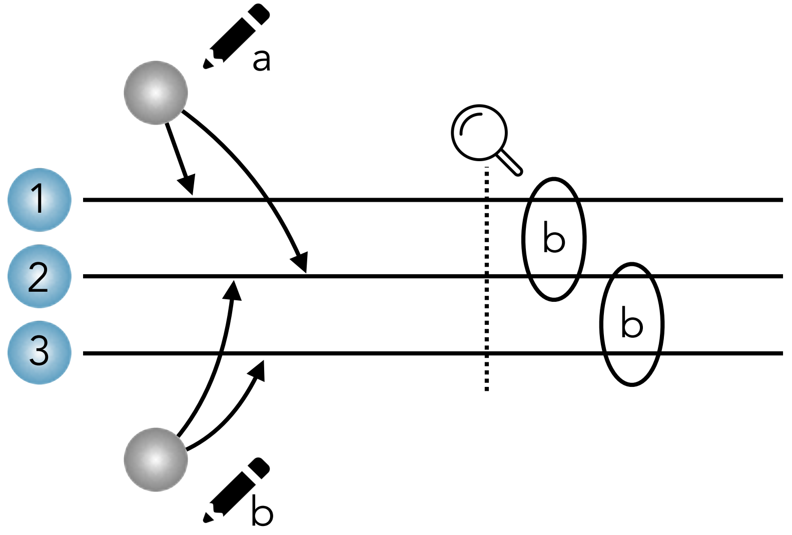
- Globally synchronised clocks

Data replication

Order of writes

- Monotonic increasing versions + PID's





- Globally synchronised clocks
- Causal ordering + PID's

Data replication

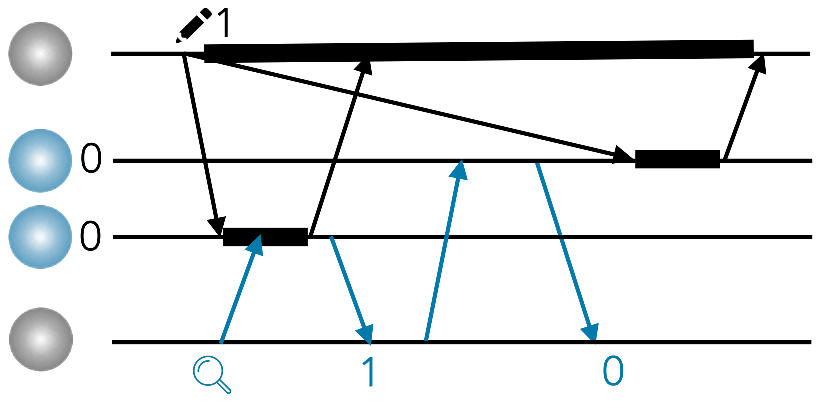
Concurrency control

- However, the lack of concurrency control of write operations may lead to basic semantics “inconsistencies”

Data replication

Concurrency control

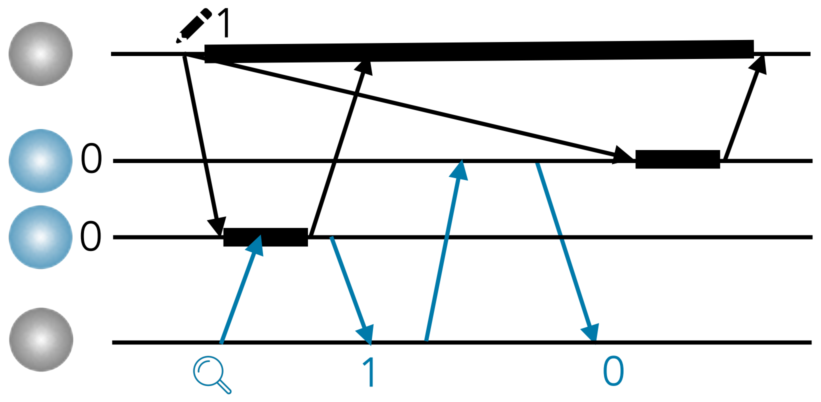
- However, the lack of concurrency control of write operations may lead to basic semantics “inconsistencies”
  - Reading **within** a concurrent write may lead to **unexplainable** results



Data replication

Concurrency control

- However, the lack of concurrency control of write operations may lead to basic semantics “inconsistencies”
  - Reading **within** a concurrent write may lead to **unexplainable** results

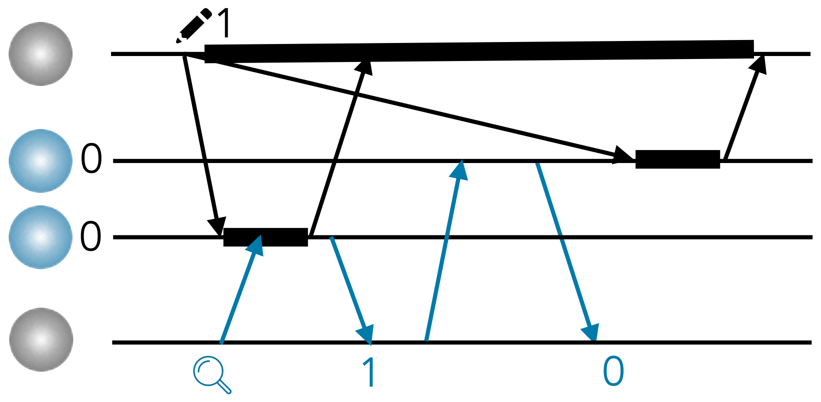


- Consider an application that manages a non-decreasing variable

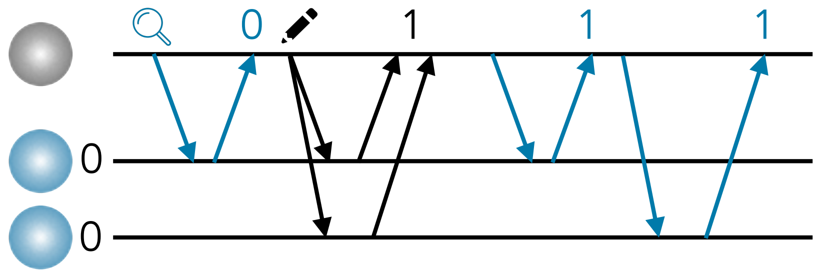
Data replication

Concurrency control

- However, the lack of concurrency control of write operations may lead to basic semantics "inconsistencies"
  - Reading **within** a concurrent write may lead to **unexplainable** results



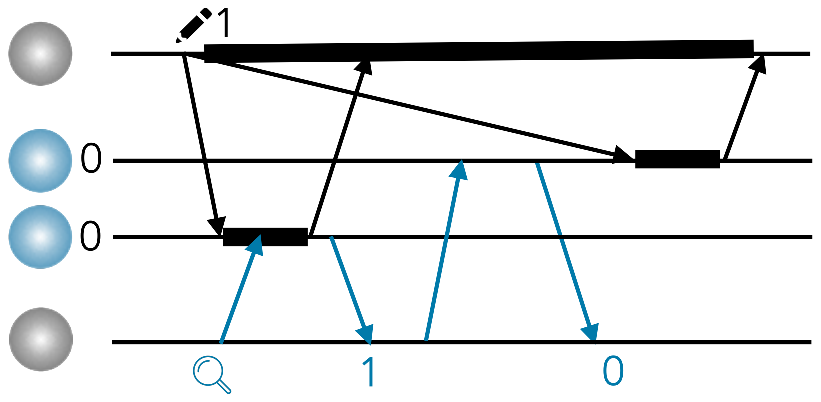
- Consider an application that manages a non-decreasing variable
  - One process (sequential) execution is OK



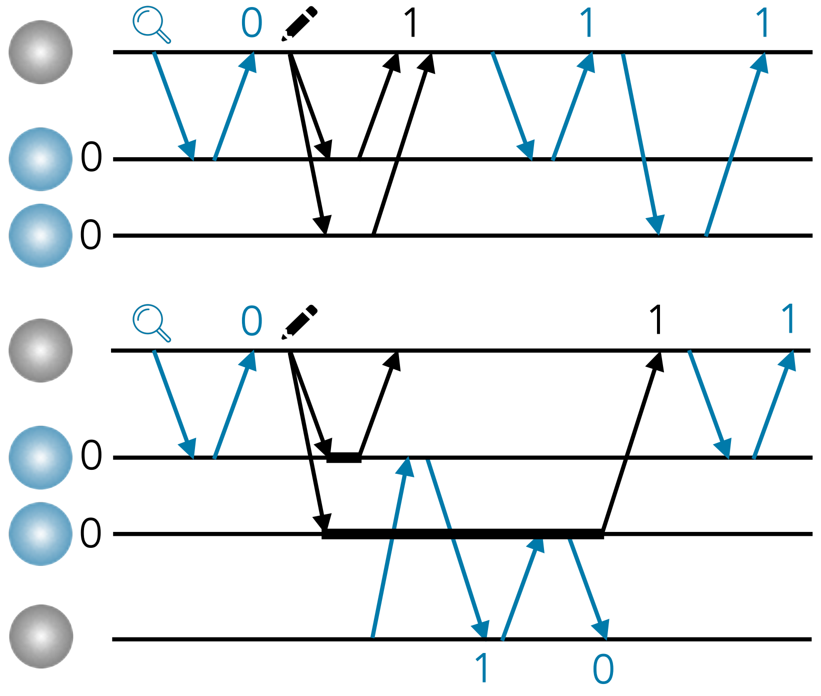
Data replication

Concurrency control

- However, the lack of concurrency control of write operations may lead to basic semantics "inconsistencies"
    - Reading **within** a concurrent write may lead to **unexplainable** results



- Consider an application that manages a non-decreasing variable
    - One process (sequential) execution is OK
    - A concurrent execution may easily violate the application semantics



Data replication

Locked quorums

- Read and Write operations take `Qr` and `Qw` into account now

At client proxies

```
read (value)
    SetL = _getlock() from a Qr
    if |SetL| == |Qr|
         SetV = _read (x, v) from a Qr
         vmax = largest (_,v) from SetV
        (value, vmax) from SetV
        res = OK
    else
        res = error
    _freelock() from SetL
    return res

write (value)
    SetL = _getlock() from a Qw
    if |SetL| == |Qw|
        SetV = _read (_, v) from a Qr
        vmax = largest (_, v) from SetV
        _write (value, vmax+1) to a Qw
        res = OK
    else
        res = error
    _freelock() from SetL
    return res
```

At replica i

```
_getlock()
    if lockedi == False
        lockedi = True
        return i
    else
        return error

_freelock()
    lockedi = False

_read(value, version)
    value = xi
    version = vi

_write (value, version)
    if version > vi then
        xi = value
        vi = version
```

Models of distributed systems and related faults

Reading material

- D. Gifford
  "Weighted voting for replicated data"
  SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 1979

- M. Whittaker, A. Charapko, J. Hellerstein, H. Howard, I. Stoica
  "Read-Write Quorum Systems Made Practical"
  PAPOC'21, 2021