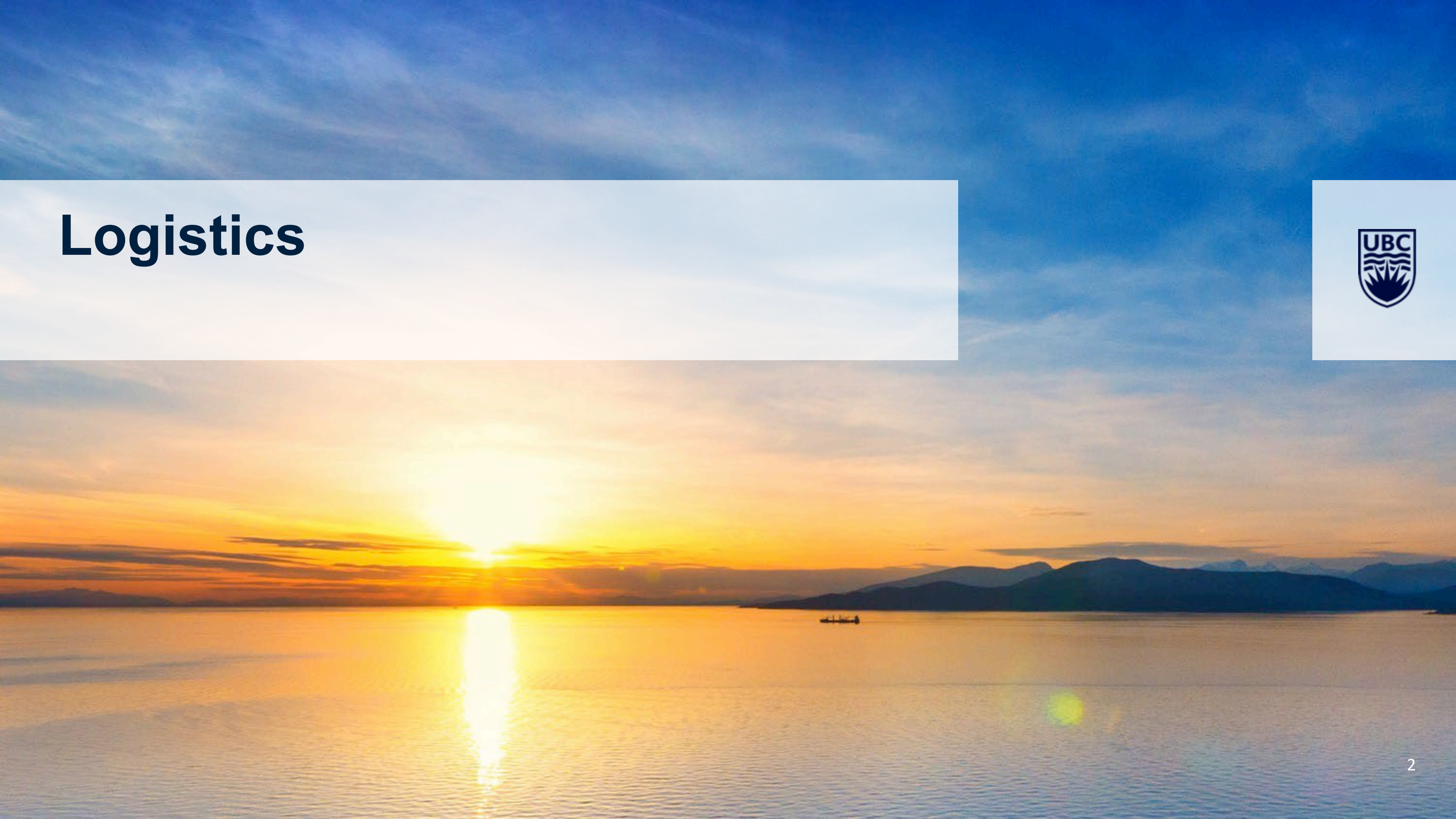
CPSC 416 Distributed Systems

Winter 2022 Term 2 (February 16, 2023)

Tony Mason (fsgeek@cs.ubc.ca), Lecturer



Logistics

Doughnuts

There is a selection of doughnuts from Brekka in the back row. Please help yourself.

Thanks for making it to class in person!

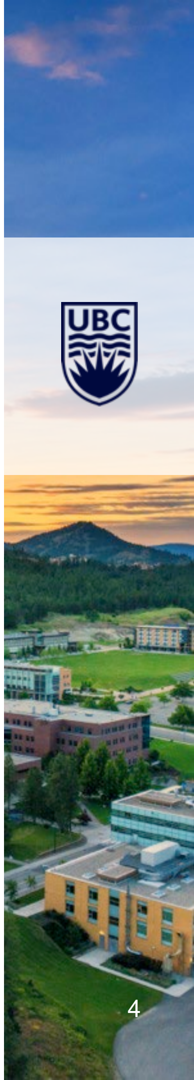
Deadlines

Project 3 Released. Late Deadline: April 13, 2023. Report Grades Pending.

Project 4 Released. Initially Due: March 13, 2023

Project 5 Released Due: April 13, 2023

All project work is due April 13, 2023. Late projects have a 75% score cap.



Deadlines

Alternate Path 1 & 2: Review in progress

- Piazza private threads need TLC
  - **Weekly updates due each Monday @ 23:59 PT**

Instructor Office Hours:

- Zoom Office Hours (Tuesday) @ 13:00-14:00
- Discord (Casual) Office Hours (Thursday) @ 14:00-15:00

TA Office Hours:

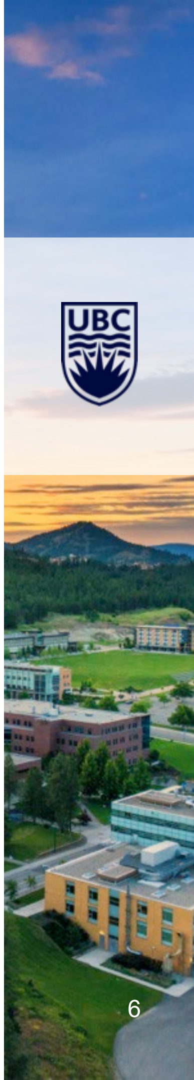
- Eric: Friday 9-11 am (in-person and Zoom)
- Japraj: Wednesday 3-5 pm (Zoom)
- Yennis: Thursday 2-4 (Zoom), Friday 2-4 (in-person)

Readings

Required:

Recommended:

- Viewstamped Replication
- Viewstamped Replication Revisited
- In Search of an Understandable Consensus Algorithm
- Paxos vs Raft: Have we reached a consensus on distributed consensus (Video)



Questions?

Questions about the class?

Questions about the previous lecture?

Funny stories to share?

Today’s Failure

Unknown Data Center

Event: Unknown

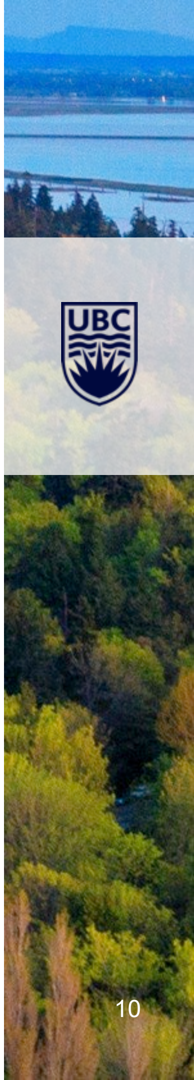
Source: Andy Warfield (UBC CS Adjunct Faculty, Amazon)

Data center design is itself a challenging field:

- Facility:
  - Space
  - Power
  - Cooling
  - Security
  - Management

Unknown Data Center

- Infrastructure
  - Servers
  - Storage
  - Networking
  - Cables and racks
  - Backup power
  - Management platforms



Unknown Data Center

What about the plumbing?

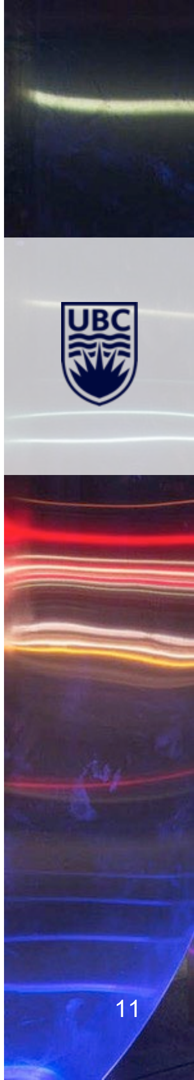
Certain equipment generates *water* as a waste product:

- Backup generator
- Air conditioning

In this case a backup generator was installed in a room, with plenty of intake air and proper venting for exhaust **above the data center*.***

When testing the backup generator system the output drain was insufficient.

Water overflowed the pan in which the generator sat and flooded the data center.



Lesson Goals

Viewstamped Replication

Viewstamped Replication

Viewstamped Replication Overview

Replicated state machines

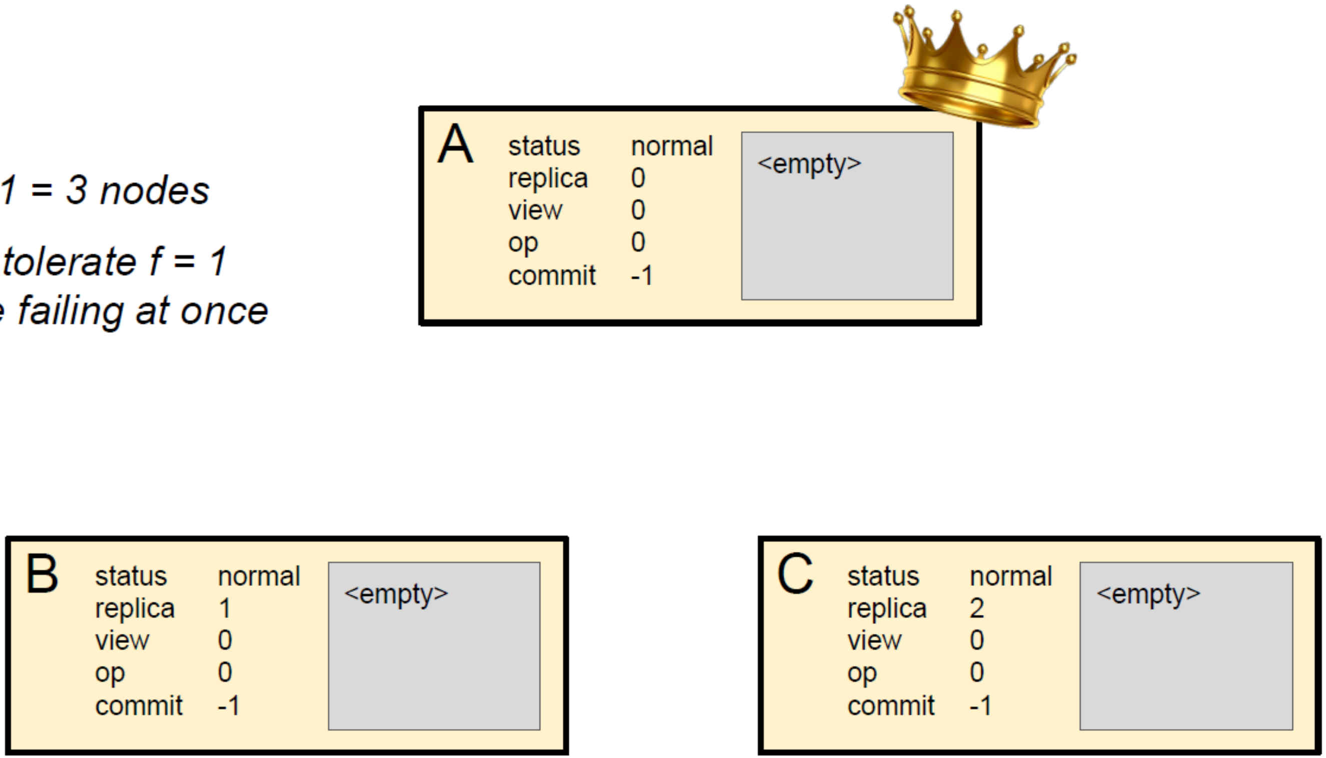
Goal: strong consistency across replicas

First proposed in 1988 (Oki & Liskov)

Model

2f + 1 = 3 nodes

Can tolerate f = 1 node failing at once



A

status	normal
replica	0
view	0
op	0
commit	-1

<empty>

B

status	normal
replica	1
view	0
op	0
commit	-1

<empty>

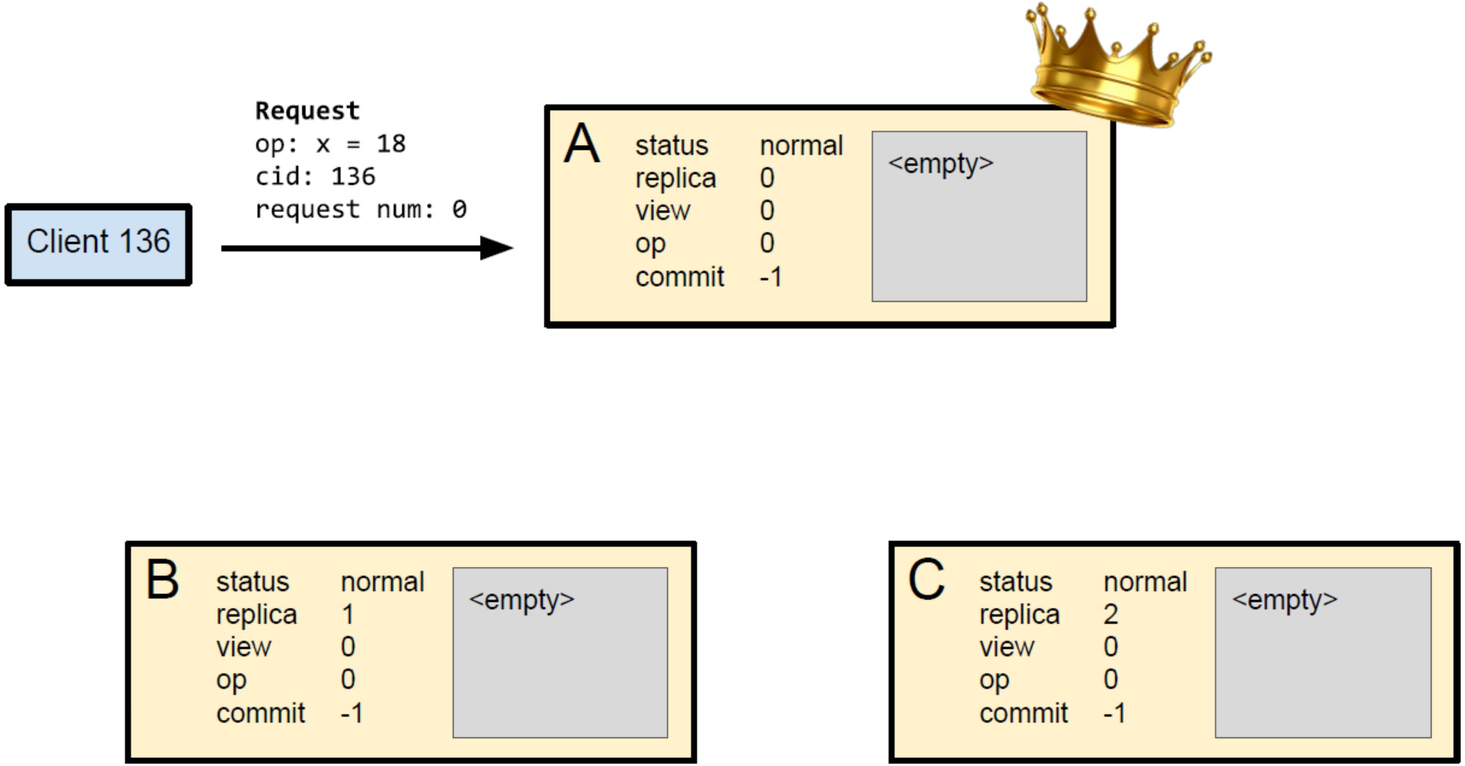
C

status	normal
replica	2
view	0
op	0
commit	-1

<empty>

Original Source

Client Request to Leader



Client 136

```
Request
op: x = 18
cid: 136
request num: 0
```

A

status	normal
replica	0
view	0
op	0
commit	-1

<empty>

B

status	normal
replica	1
view	0
op	0
commit	-1

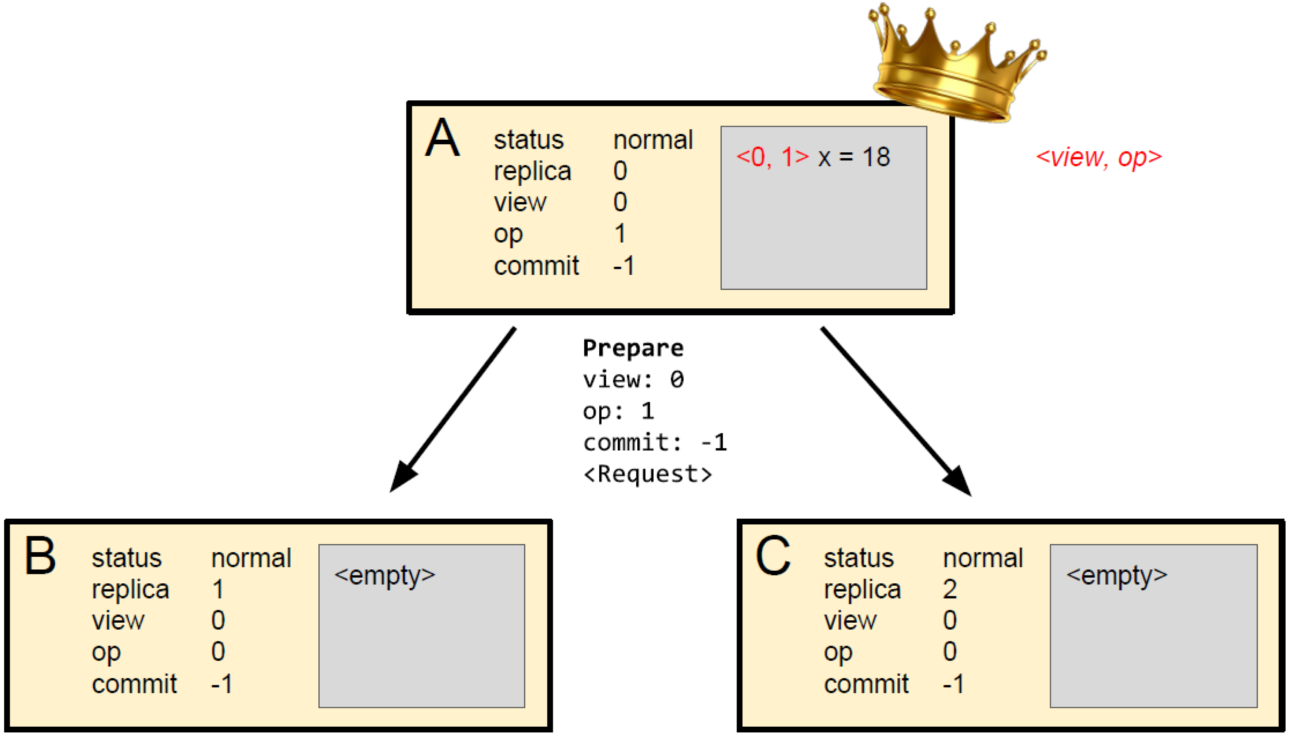
<empty>

C

status	normal
replica	2
view	0
op	0
commit	-1

<empty>

Leader Updates & Forwards



A

status	normal
replica	0
view	0
op	1
commit	-1

<0, 1> x = 18

<view, op>

```
Prepare
view: 0
op: 1
commit: -1
<Request>
```

B

status	normal
replica	1
view	0
op	0
commit	-1

<empty>

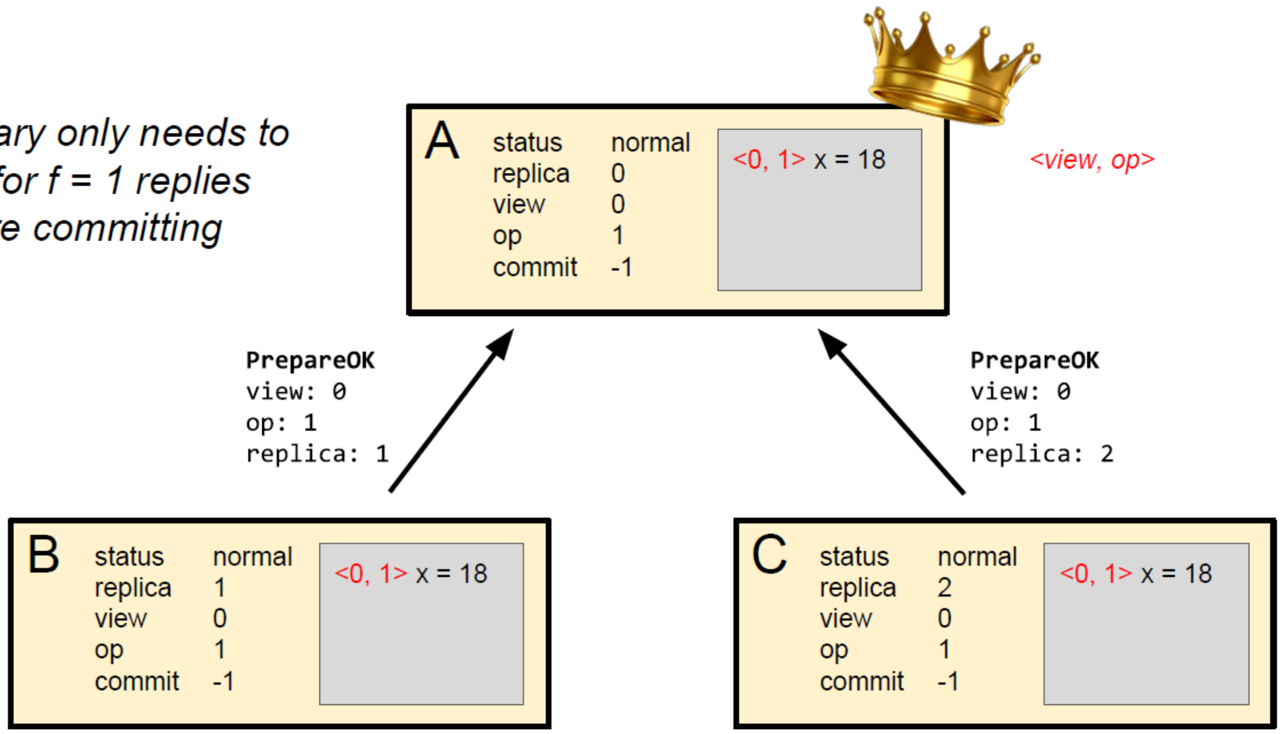
C

status	normal
replica	2
view	0
op	0
commit	-1

<empty>

Replicas acknowledge

Primary only needs to wait for f = 1 replies before committing



A

status	normal
replica	0
view	0
op	1
commit	-1

<0, 1> x = 18

```
PrepareOK
view: 0
op: 1
replica: 1
```

```
PrepareOK
view: 0
op: 1
replica: 2
```

B

status	normal
replica	1
view	0
op	1
commit	-1

<0, 1> x = 18

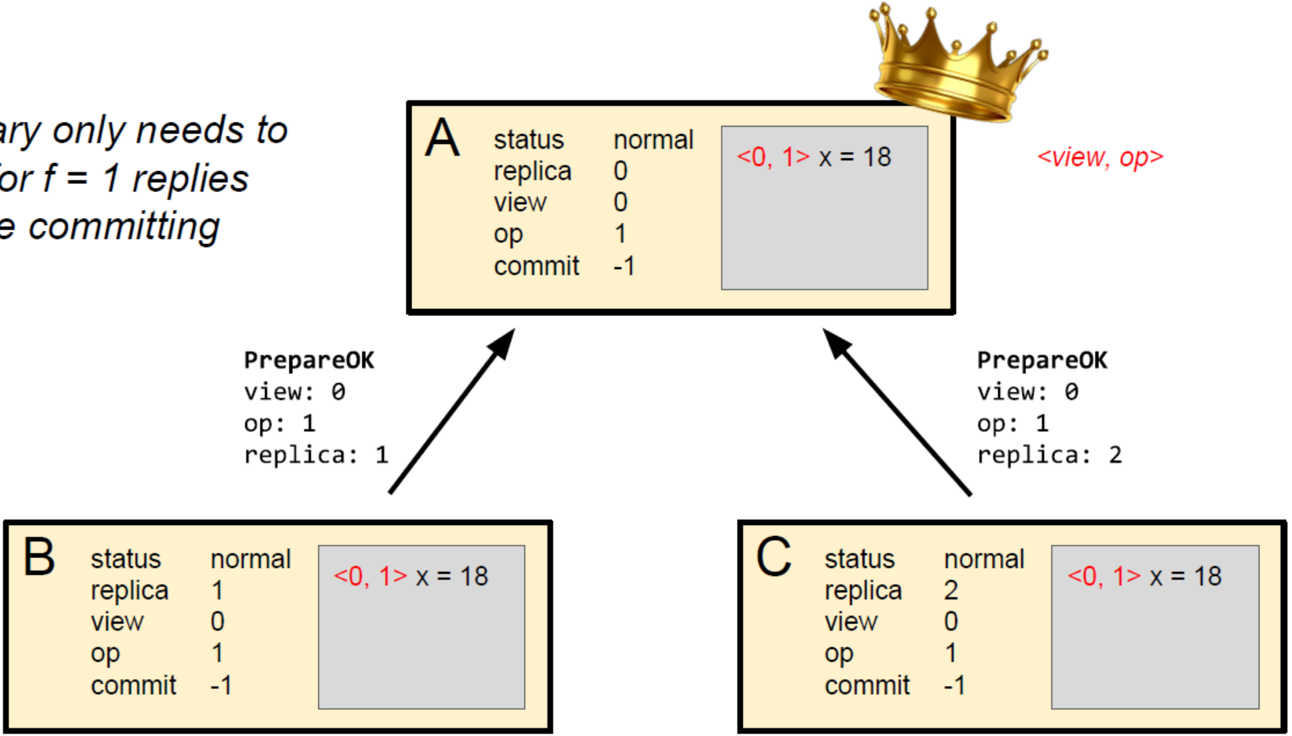
C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18

Replicas Respond & Leader Commits

Primary only needs to wait for f = 1 replies before committing

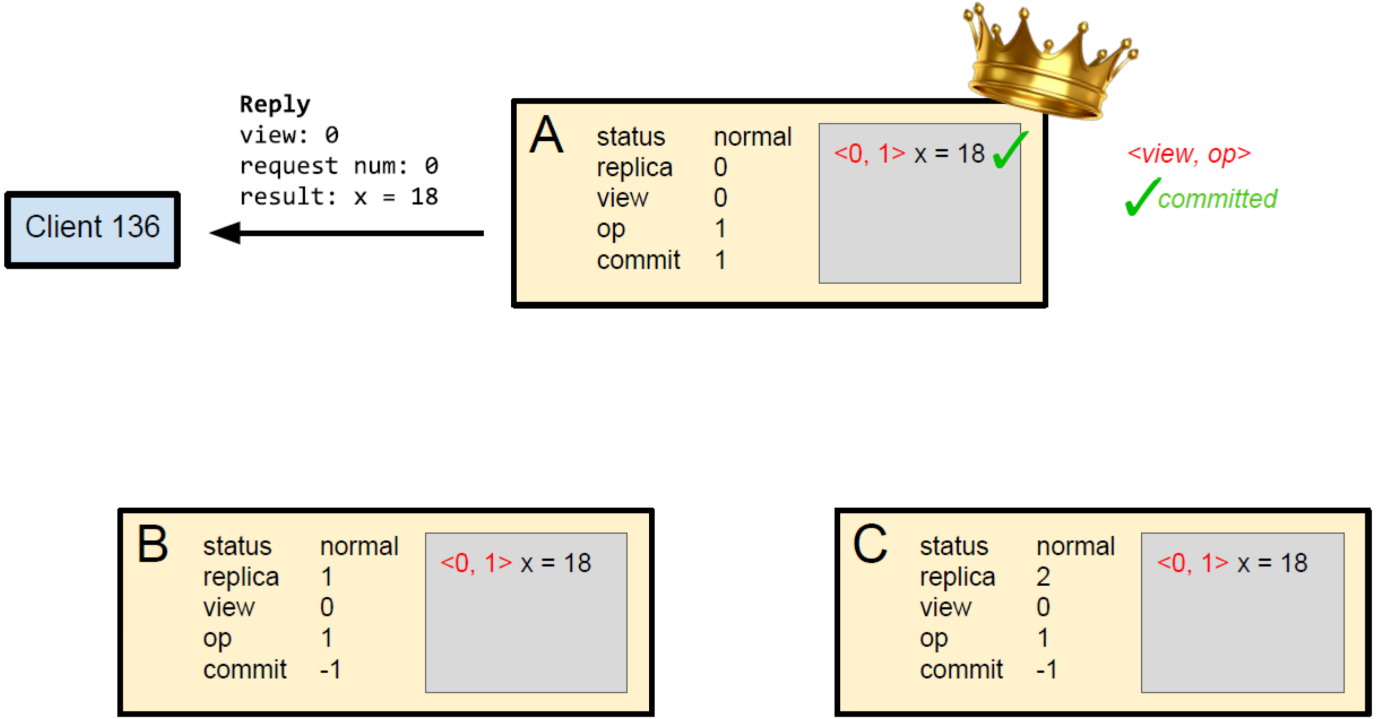


	A	B	C
status	normal	normal	normal
replica	0	1	2
view	0	0	0
op	1	1	1
commit	-1	-1	-1
log	<0, 1> x = 18	<0, 1> x = 18	<0, 1> x = 18

```
PrepareOK
view: 0
op: 1
replica: 1
```

```
PrepareOK
view: 0
op: 1
replica: 2
```

Leader Acknowledges Request to Client



```
Reply
view: 0
request num: 0
result: x = 18
```

Client 136

A

status	normal
replica	0
view	0
op	1
commit	1

<0, 1> x = 18 ✓

<view, op>

✓committed

B

status	normal
replica	1
view	0
op	1
commit	-1

<0, 1> x = 18

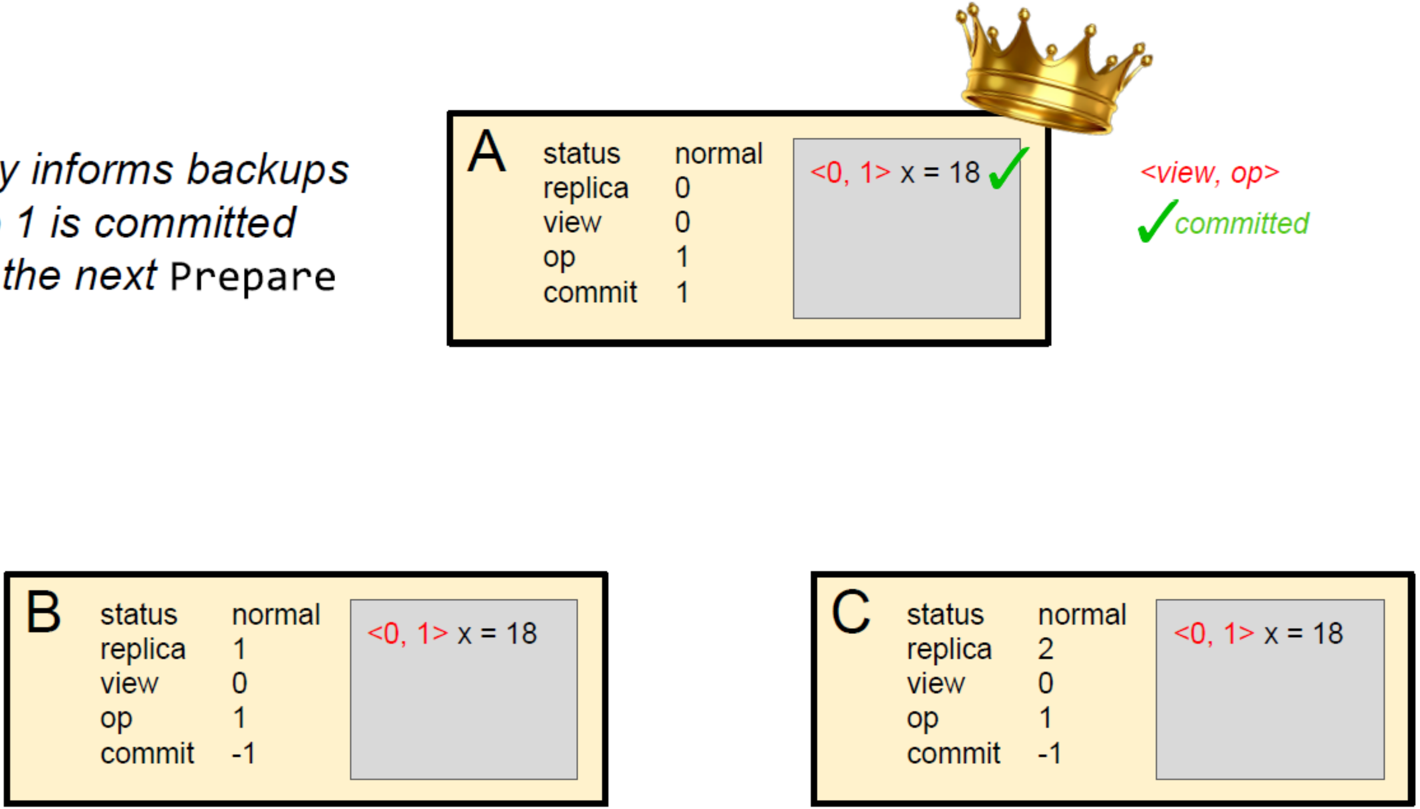
C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18

Delay Commit to Replicas

Primary informs backups that op 1 is committed during the next `Prepare`



A	
status	normal
replica	0
view	0
op	1
commit	1

<0, 1> x = 18 ✓

<view, op>

✓committed

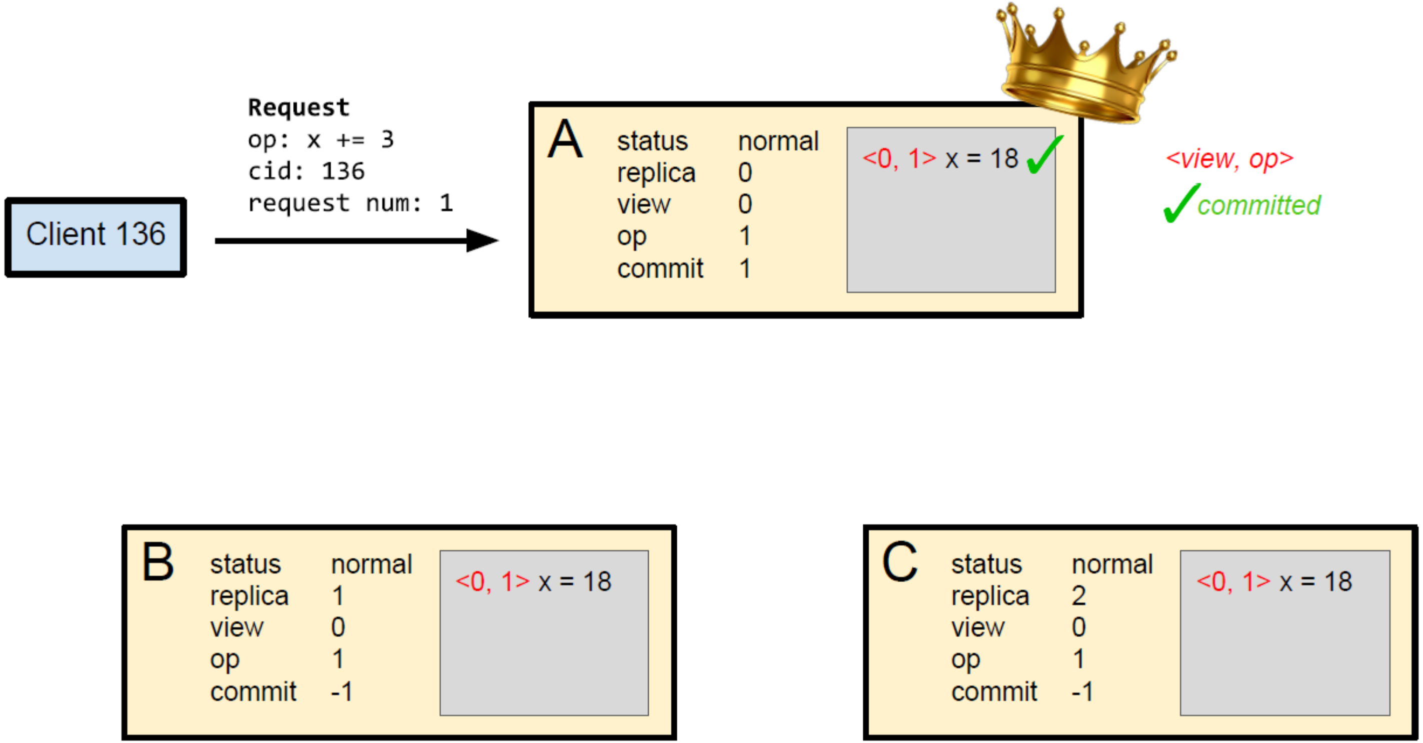
B	
status	normal
replica	1
view	0
op	1
commit	-1

<0, 1> x = 18

C	
status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18

Client Sends a New Request



Client 136

```
Request
op: x += 3
cid: 136
request num: 1
```

A

status	normal
replica	0
view	0
op	1
commit	1

<0, 1> x = 18 ✓

<view, op>

✓ committed

B

status	normal
replica	1
view	0
op	1
commit	-1

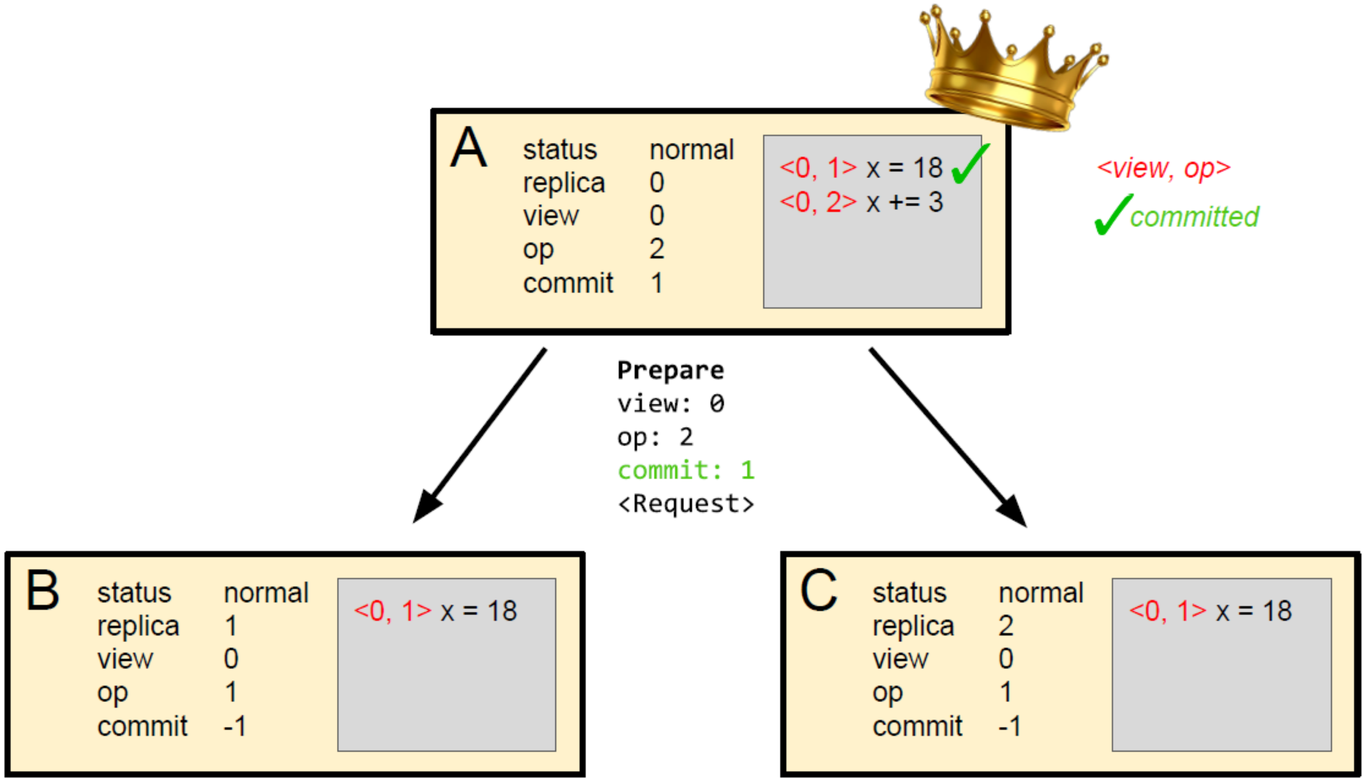
<0, 1> x = 18

C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18

Leader Updates & Forwards



A

status	normal
replica	0
view	0
op	2
commit	1

<0, 1> x = 18 ✓
<0, 2> x += 3

<view, op>
✓committed

```
Prepare
view: 0
op: 2
commit: 1
<Request>
```

B

status	normal
replica	1
view	0
op	1
commit	-1

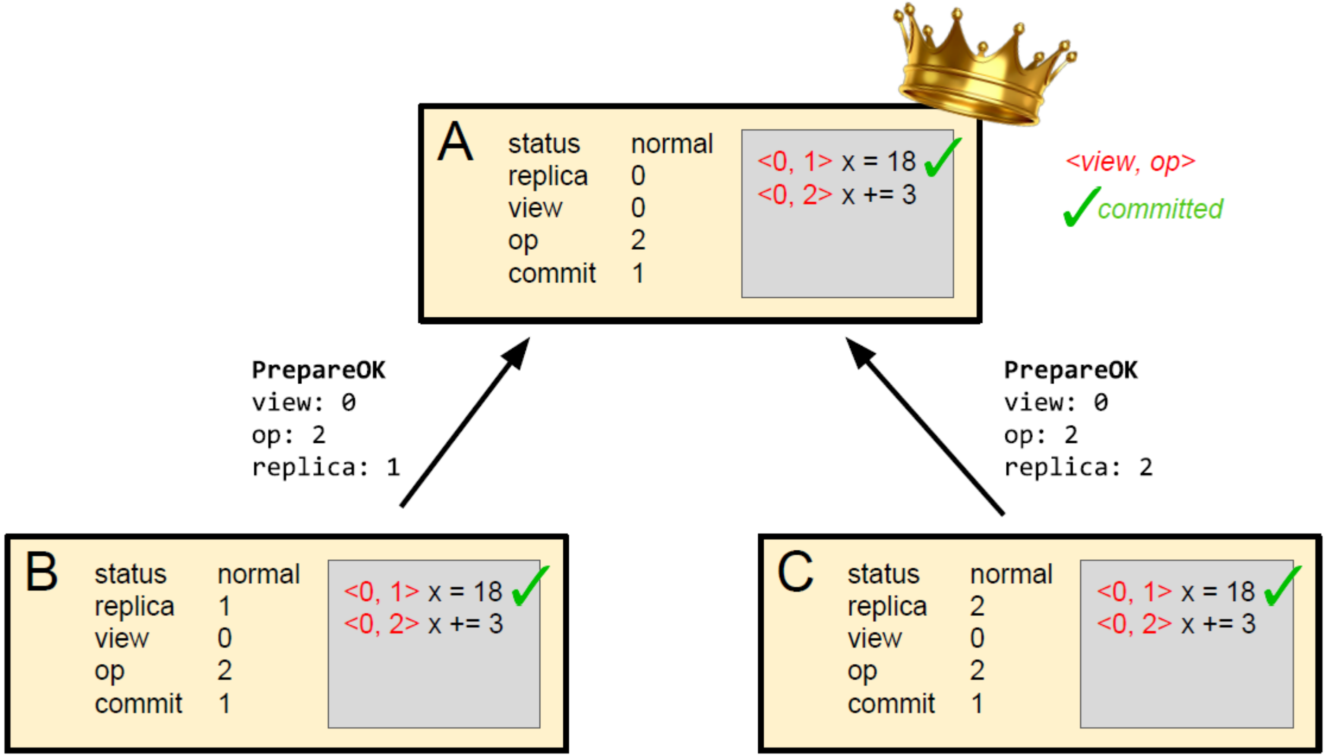
<0, 1> x = 18

C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18

Replicas Respond & Leader Commits



A

status	normal
replica	0
view	0
op	2
commit	1

<0, 1> x = 18 ✓
<0, 2> x += 3

<view, op>

✓committed

```
PrepareOK
view: 0
op: 2
replica: 1
```

```
PrepareOK
view: 0
op: 2
replica: 2
```

B

status	normal
replica	1
view	0
op	2
commit	1

<0, 1> x = 18 ✓
<0, 2> x += 3

C

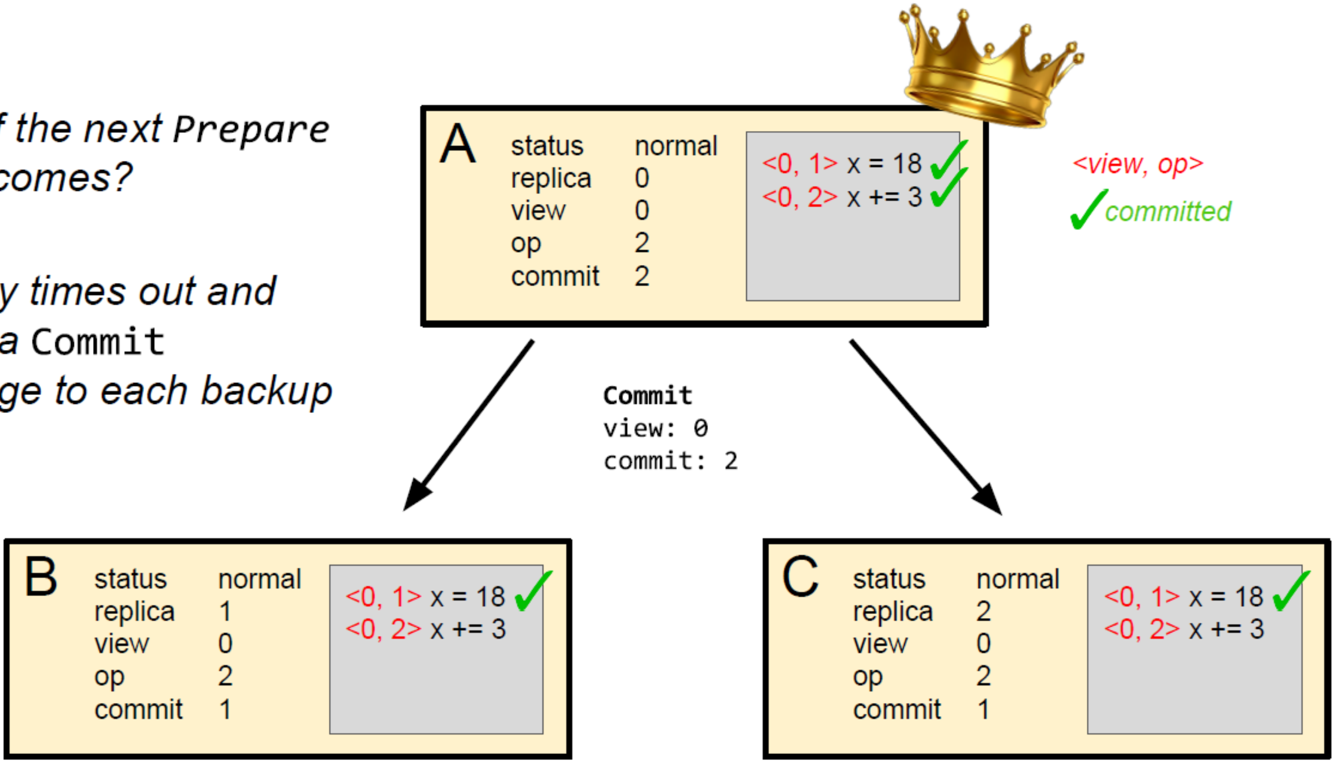
status	normal
replica	2
view	0
op	2
commit	1

<0, 1> x = 18 ✓
<0, 2> x += 3

Leader Notifies Replicas of Commit

What if the next `Prepare` *never comes?*

Primary times out and sends a `Commit` *message to each backup*



A

status	normal
replica	0
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

<view, op>
✓committed

```
Commit
view: 0
commit: 2
```

B

status	normal
replica	1
view	0
op	2
commit	1

<0, 1> x = 18 ✓
<0, 2> x += 3

C

status	normal
replica	2
view	0
op	2
commit	1

<0, 1> x = 18 ✓
<0, 2> x += 3

Consistency Achieved

A

status	normal
replica	0
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

<view, op>
✓committed

B

status	normal
replica	1
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

C

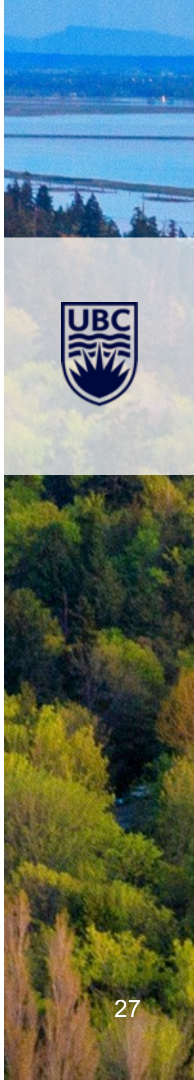
status	normal
replica	2
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

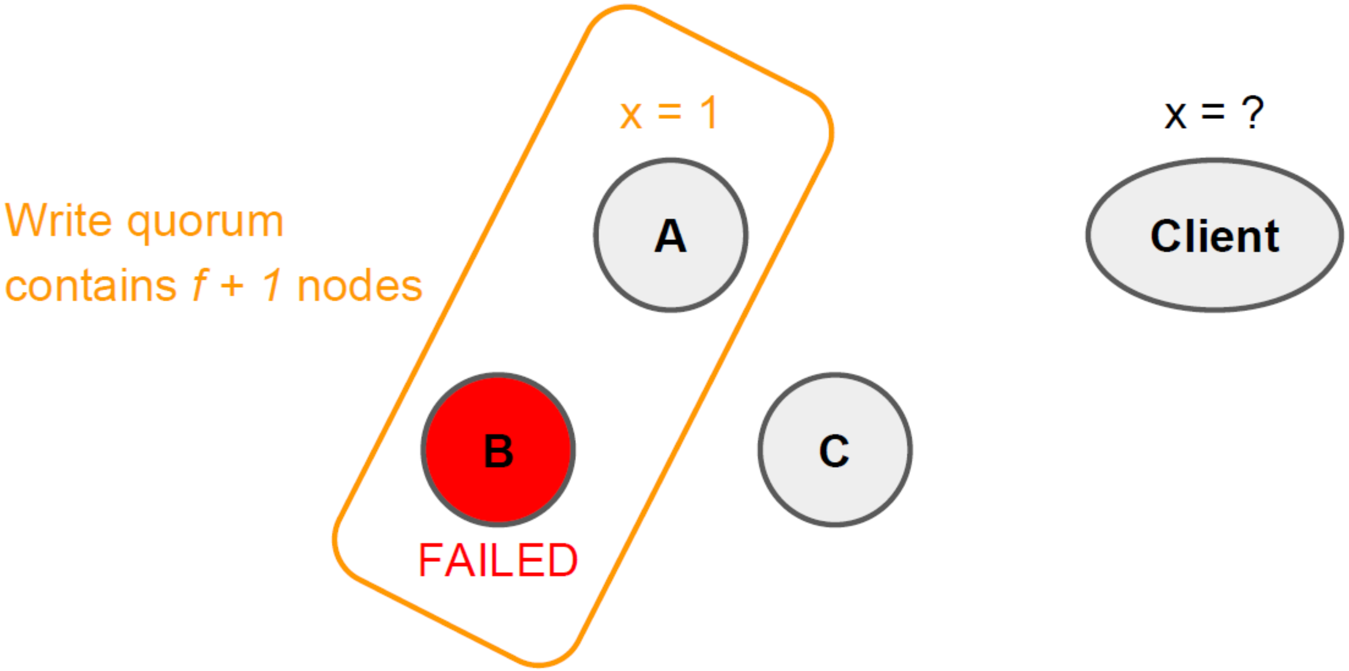
Quorum Consensus

Waiting for *f* nodes is sufficient because:

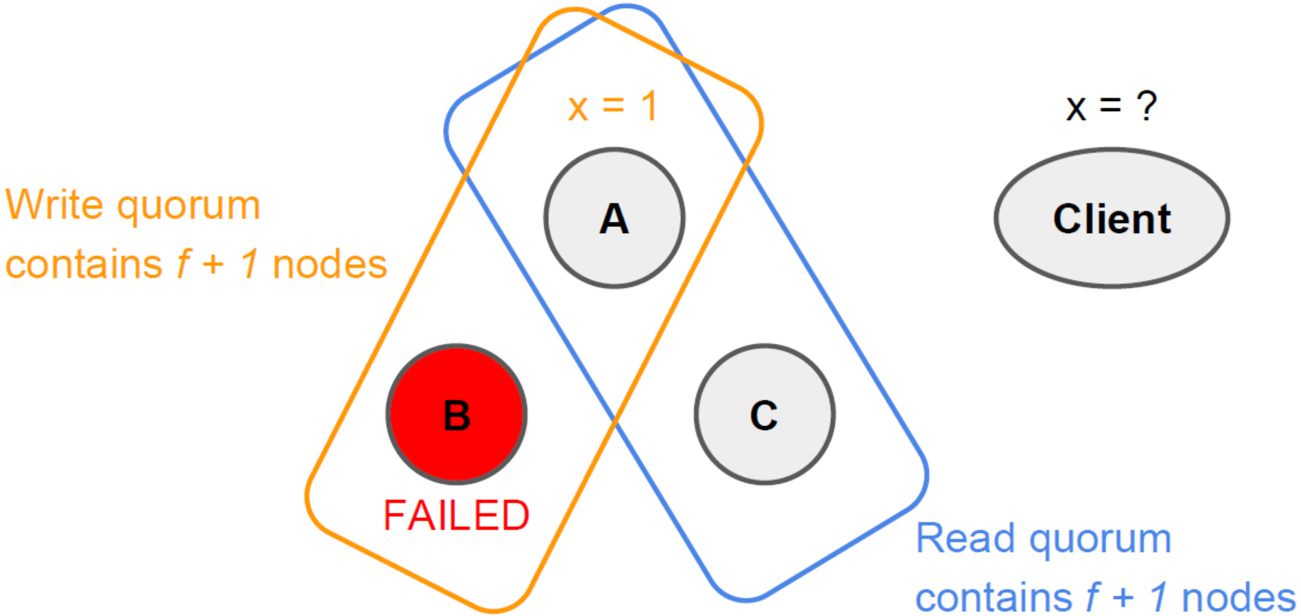
- Operation has happened on *f* + *1* nodes = quorum



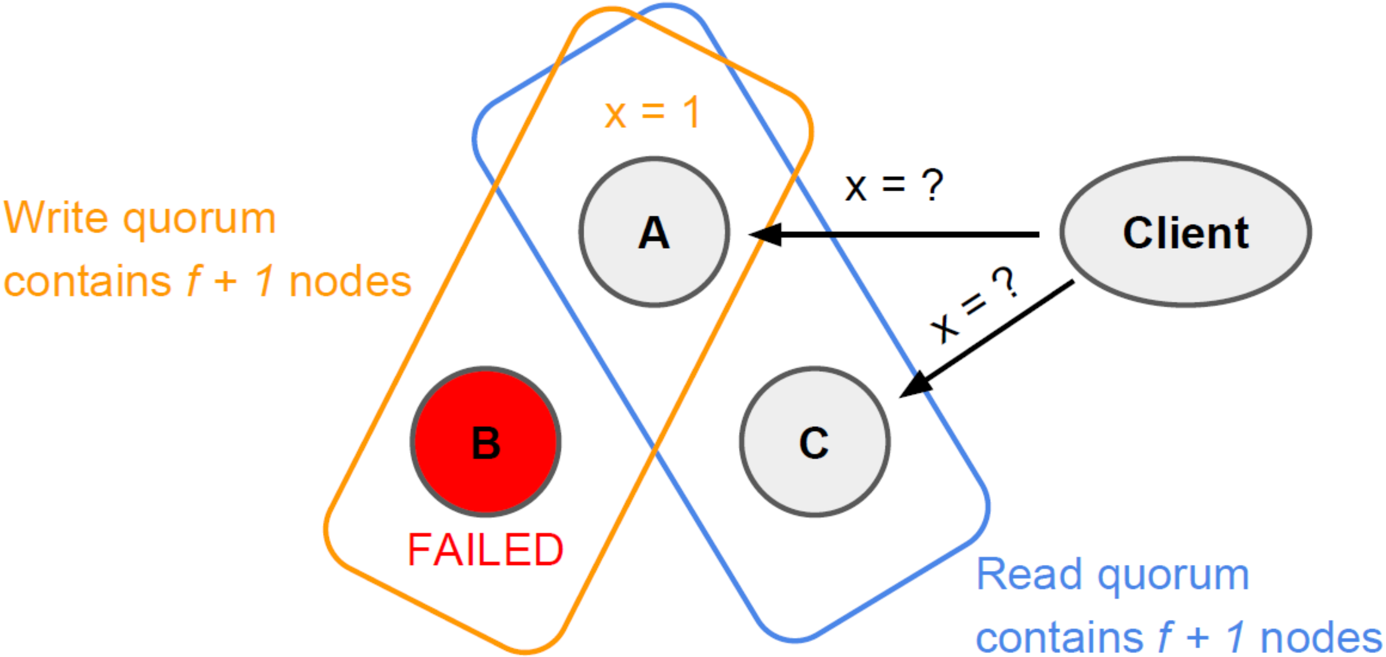
Write Quorum



Read Quorum

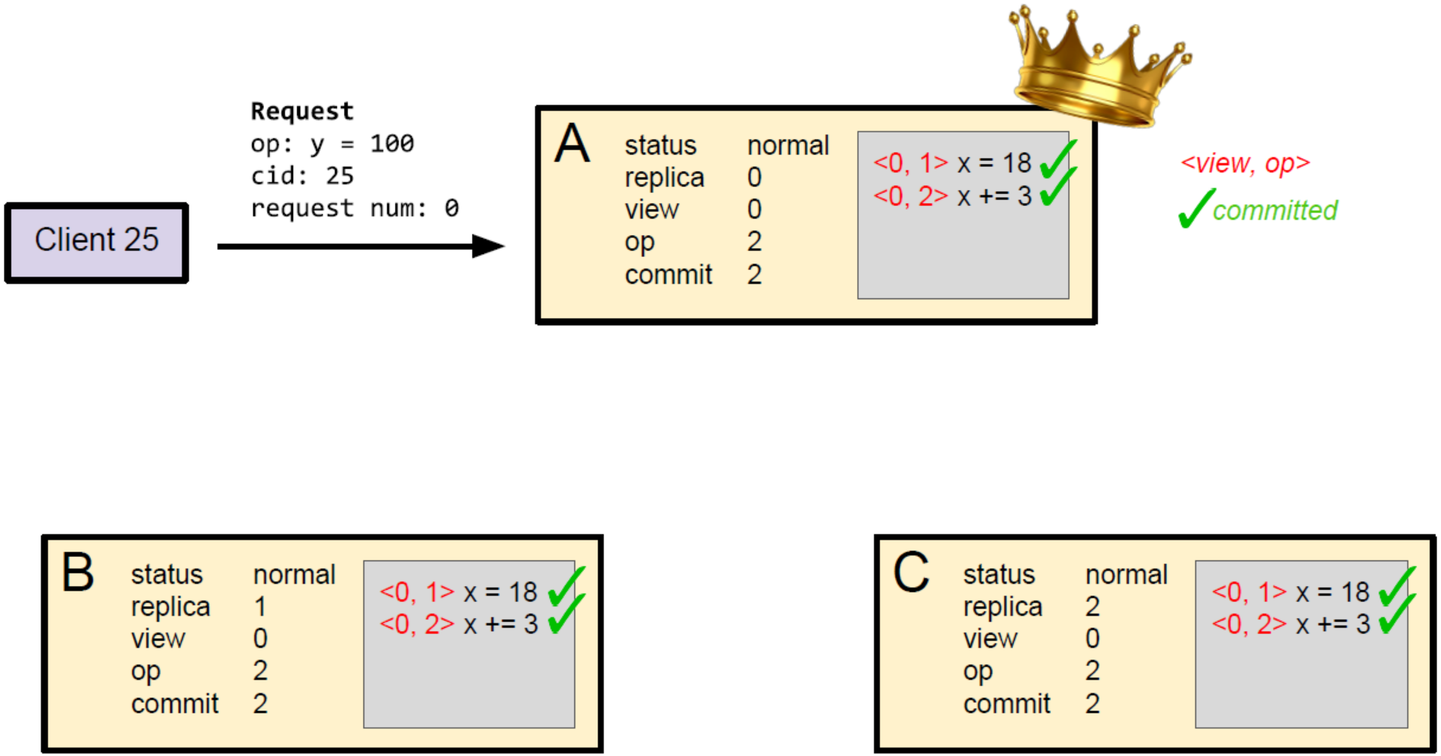

x = 1
x = ?
A
Client
Write quorum
contains *f + 1* nodes
B
FAILED
C
Read quorum
contains *f + 1* nodes

Verifying Quorum



Views

Client Sends a New Request



Client 25

```
Request
op: y = 100
cid: 25
request num: 0
```

A

status	normal
replica	0
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

<view, op>
✓committed

B

status	normal
replica	1
view	0
op	2
commit	2

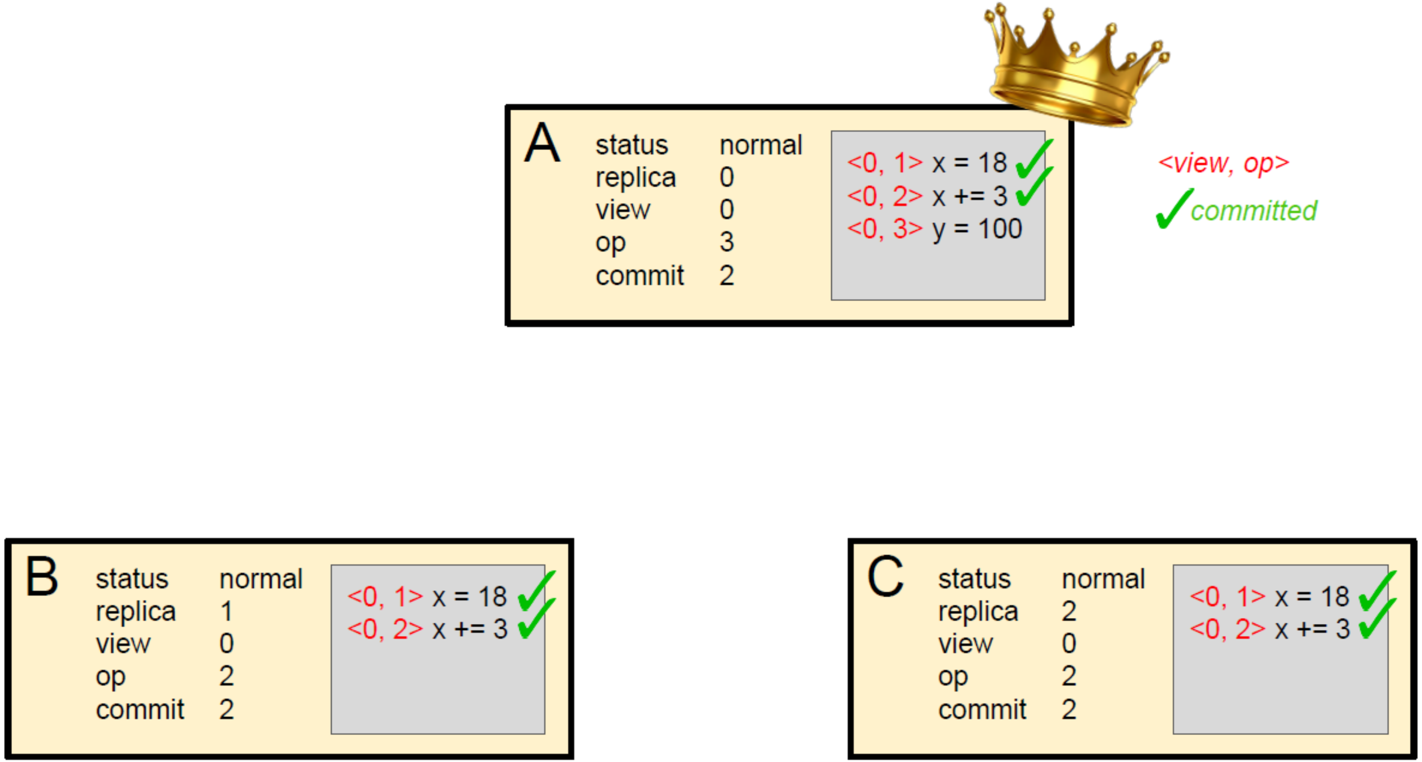
<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

C

status	normal
replica	2
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

Leader Updates



A

status	normal
replica	0
view	0
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

<view, op>
✓committed

B

status	normal
replica	1
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

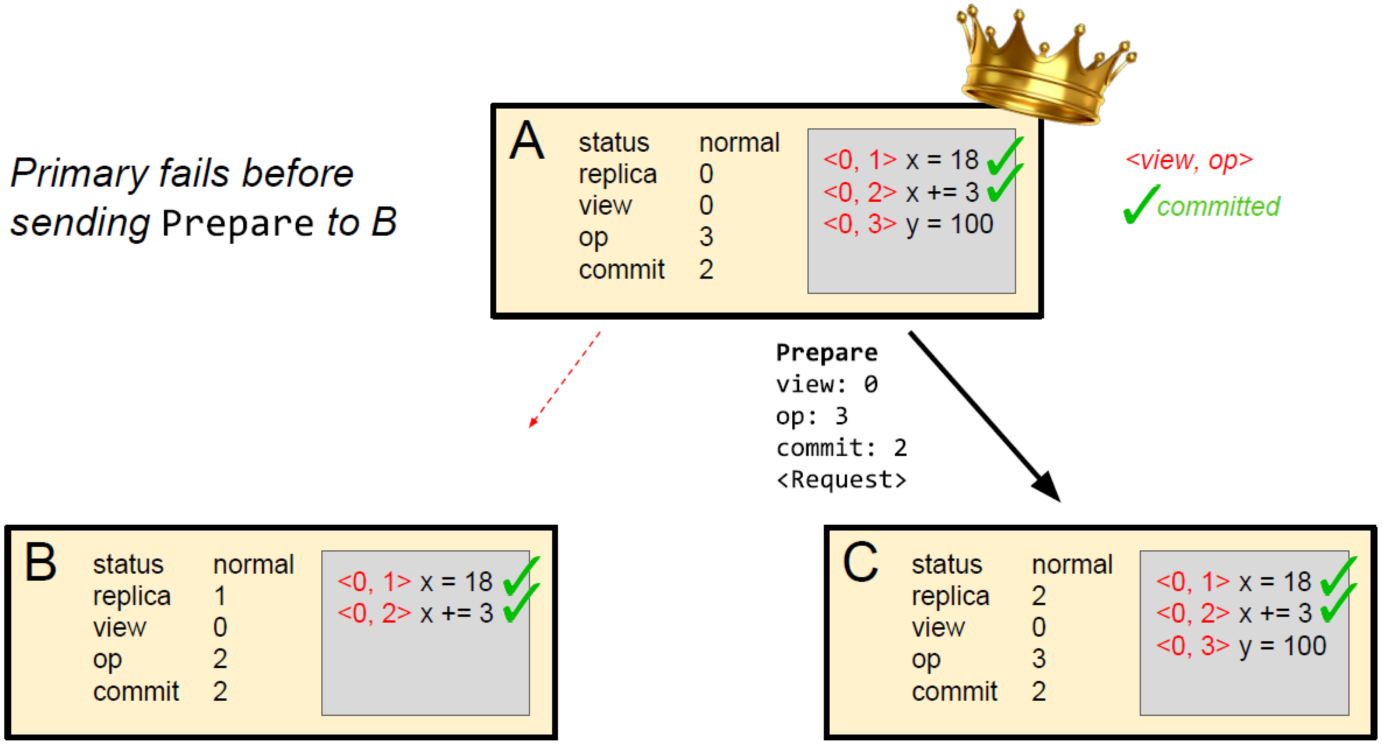
C

status	normal
replica	2
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

Leader failure

Primary fails before sending `Prepare` *to B*



A

status	normal
replica	0
view	0
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

<view, op>
✓committed

```
Prepare
view: 0
op: 3
commit: 2
<Request>
```

B

status	normal
replica	1
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

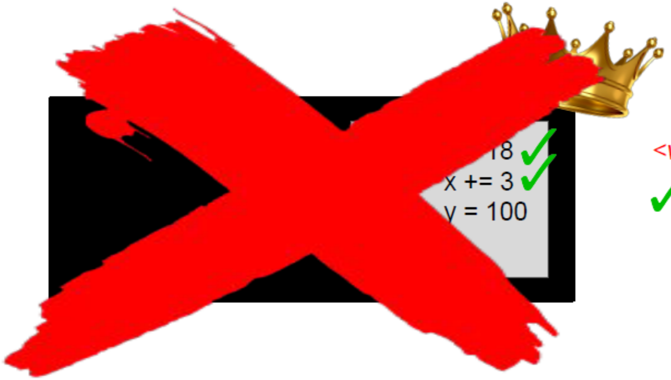
C

status	normal
replica	2
view	0
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

Failure: Inconsistent logs

Logs are out of sync



<view, op>

✓*committed*

B

status	normal
replica	1
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

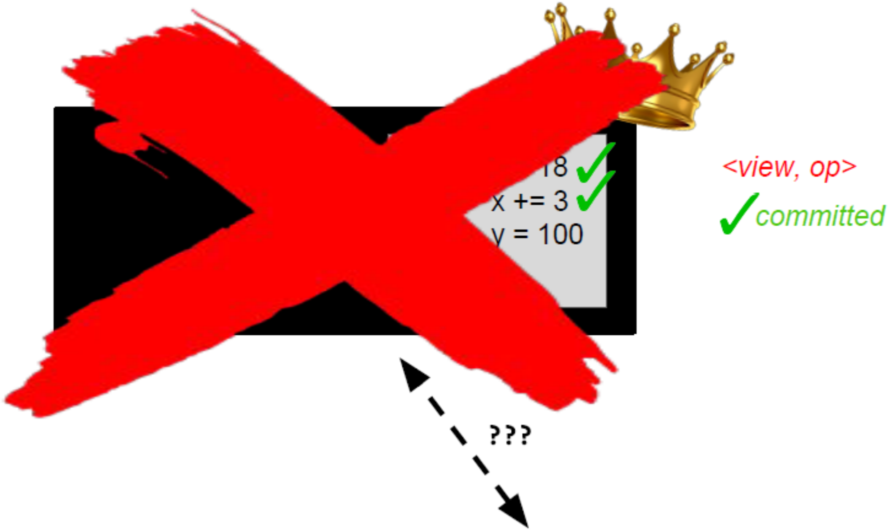
C

status	normal
replica	2
view	0
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

Replica Timeout

C times out on hearing from the primary and starts view change



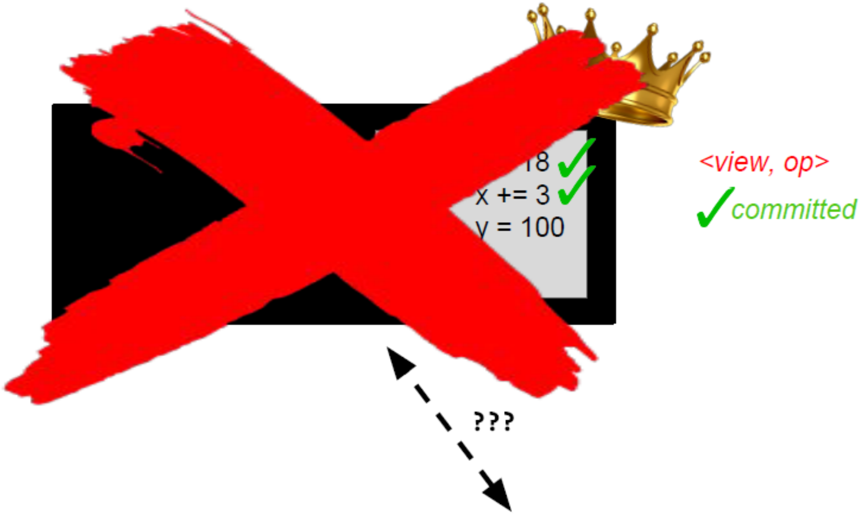
B			
status	normal	<0, 1>	x = 18 ✓
replica	1	<0, 2>	x += 3 ✓
view	0		
op	2		
commit	2		

C			
status	normal	<0, 1>	x = 18 ✓
replica	2	<0, 2>	x += 3 ✓
view	0	<0, 3>	y = 100
op	3		
commit	2		

“Elect” a New Leader

Who is the new primary?

Go through the list of sorted IP addresses and find the next one (i.e. B)



B

status	normal	<0, 1> x = 18 ✓
replica	1	<0, 2> x += 3 ✓
view	0	
op	2	
commit	2	

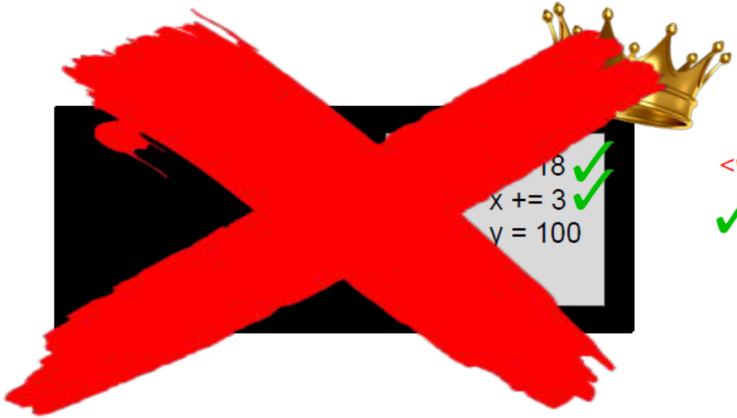
C

status	normal	<0, 1> x = 18 ✓
replica	2	<0, 2> x += 3 ✓
view	0	<0, 3> y = 100
op	3	
commit	2	

New Leader

Start view change:

Status = change
Increment local view
Send SVC to all nodes



<view, op>
✓*committed*

B

status	normal	<0, 1> x = 18 ✓
replica	1	<0, 2> x += 3 ✓
view	0	
op	2	
commit	2	

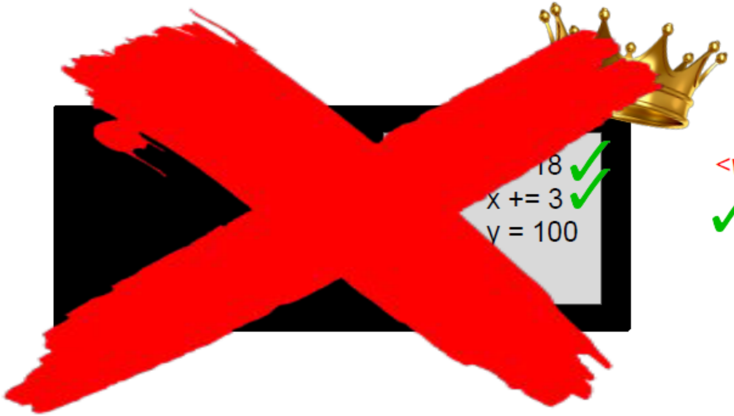
C

status	normal	<0, 1> x = 18 ✓
replica	2	<0, 2> x += 3 ✓
view	0	<0, 3> y = 100
op	3	
commit	2	

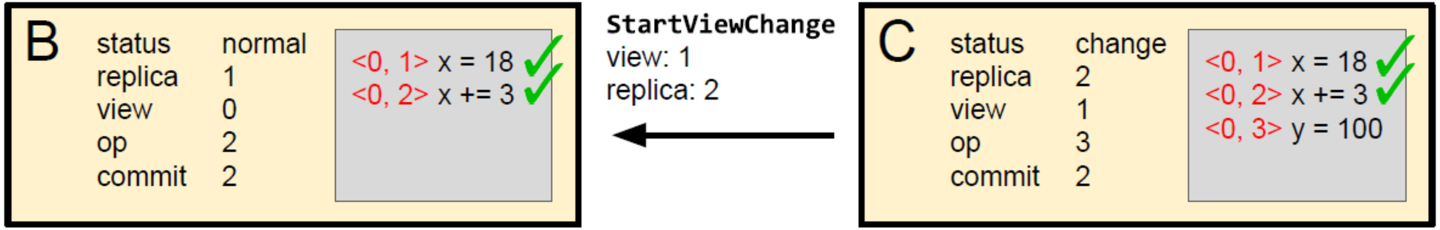
New Leader, New View

Start view change:

Status = change
Increment local view
Send SVC to all nodes



<view, op>
✓*committed*



B

status	normal
replica	1
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

`StartViewChange`
view: 1
replica: 2

C

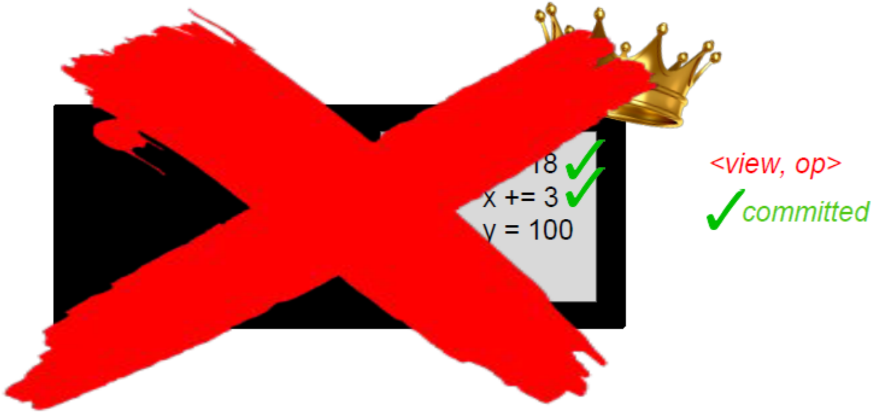
status	change
replica	2
view	1
op	3
commit	2

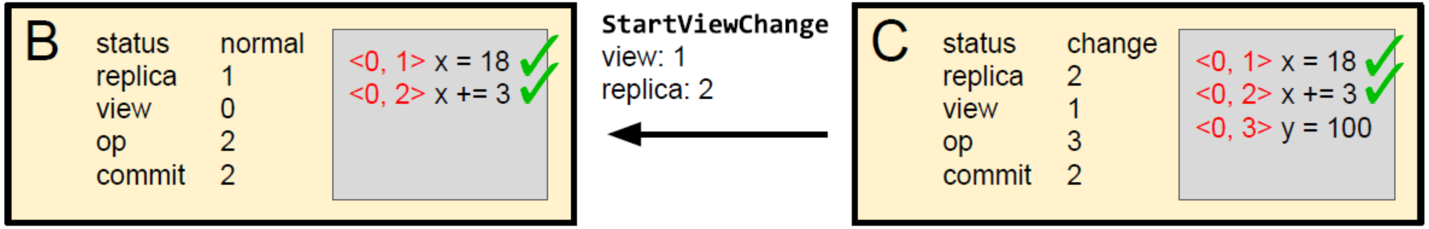
<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

Process View Change

Receive SVC where:

```
SVC.view > local view {
    Status = view change
    Advance local view
    Send SVC to other nodes
}
```





B

status	normal
replica	1
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

StartViewChange
view: 1
replica: 2

C

status	change
replica	2
view	1
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

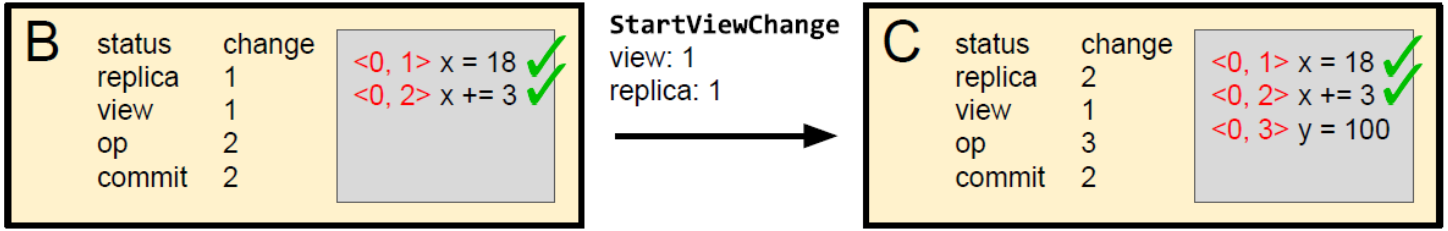
Acknowledge View Change

Receive SVC where:

```
SVC.view > local view {
    Status = view change
    Advance local view
    Send SVC to other nodes
}
```

<view, op>

✓committed



B

status	change
replica	1
view	1
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

StartViewChange
view: 1
replica: 1

C

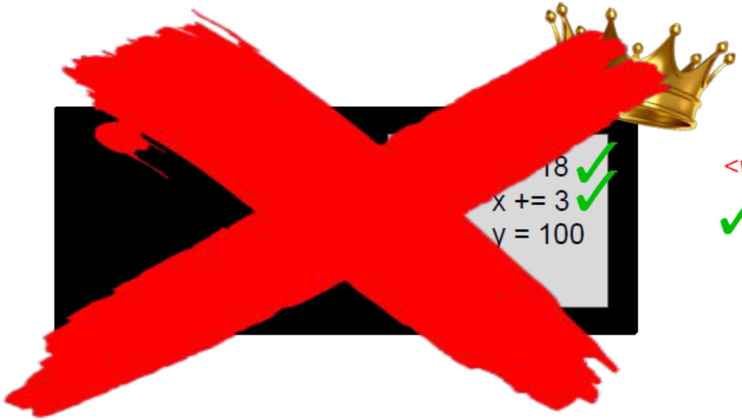
status	change
replica	2
view	1
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

Acknowledge View Change

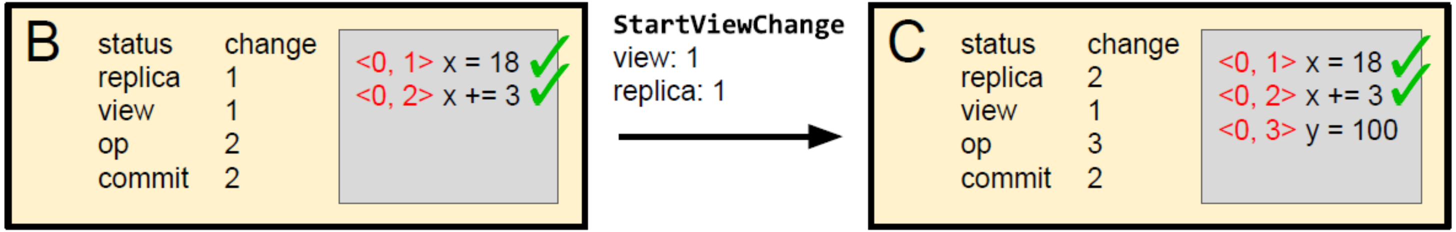
Receive f SVCs where:

```
SVC.view == local view {
    Send DVC to new primary
}
```



<view, op>

✓committed



B		
status	change	<0, 1> x = 18 ✓
replica	1	<0, 2> x += 3 ✓
view	1	
op	2	
commit	2	

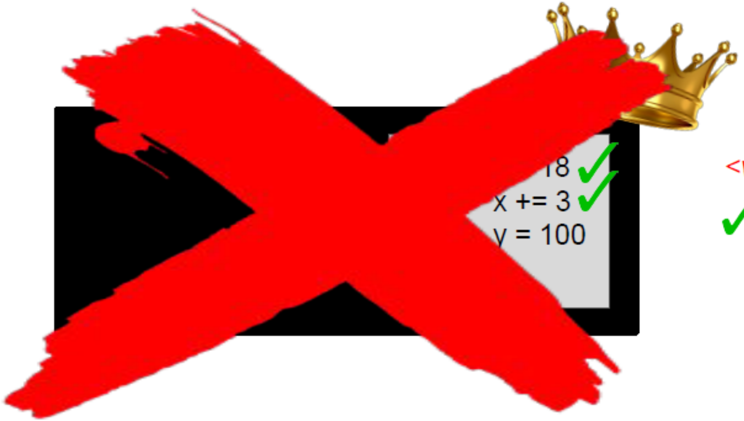
StartViewChange
view: 1
replica: 1

C		
status	change	<0, 1> x = 18 ✓
replica	2	<0, 2> x += 3 ✓
view	1	<0, 3> y = 100
op	3	
commit	2	

Complete View Change

Receive f SVCs where:

```
SVC.view == local view {
    Send DVC to new primary
}
```



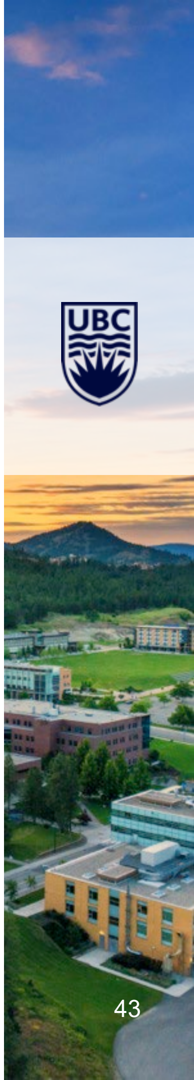
<view, op>
✓committed

DoViewChange
replica: 2
view: 1
op: 3
commit: 2
<log>

←

B		
status	change	<0, 1> x = 18 ✓
replica	1	<0, 2> x += 3 ✓
view	1	
op	2	
commit	2	

C		
status	change	<0, 1> x = 18 ✓
replica	2	<0, 2> x += 3 ✓
view	1	<0, 3> y = 100
op	3	
commit	2	



Logs: In Sync

Logs are no longer out of sync!

With more nodes, we may receive multiple different logs

Pick the one with highest view and op number

<view, op>

✓committed

B

status	change
replica	1
view	1
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

C

status	change
replica	2
view	1
op	3
commit	2

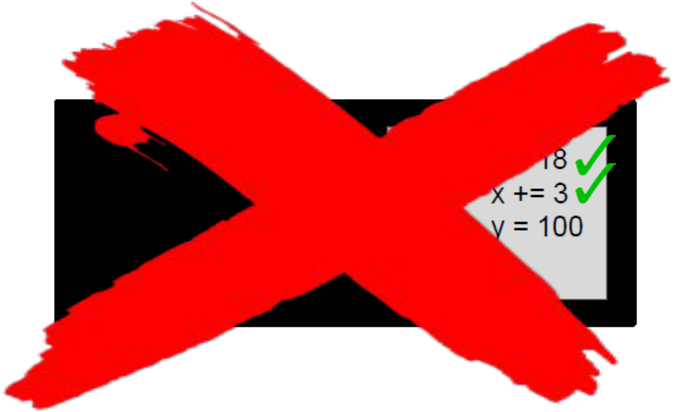
<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

Start New Primary

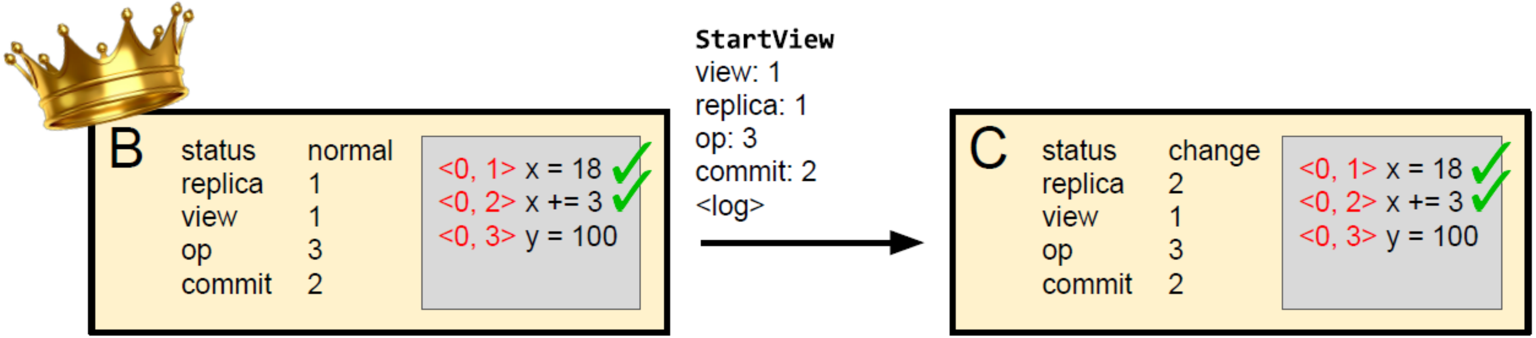
Receive f DVCs:

Become new primary
Send `StartView` *to others*

Why do we send the log here?



<view, op>
✓committed



B
status normal
replica 1
view 1
op 3
commit 2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

`StartView`
view: 1
replica: 1
op: 3
commit: 2
<log>

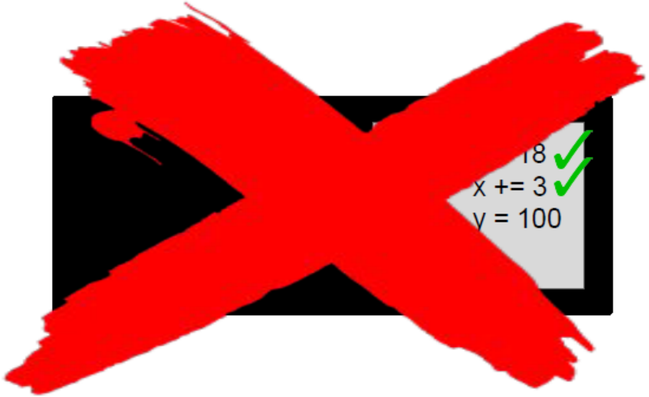
C
status change
replica 2
view 1
op 3
commit 2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

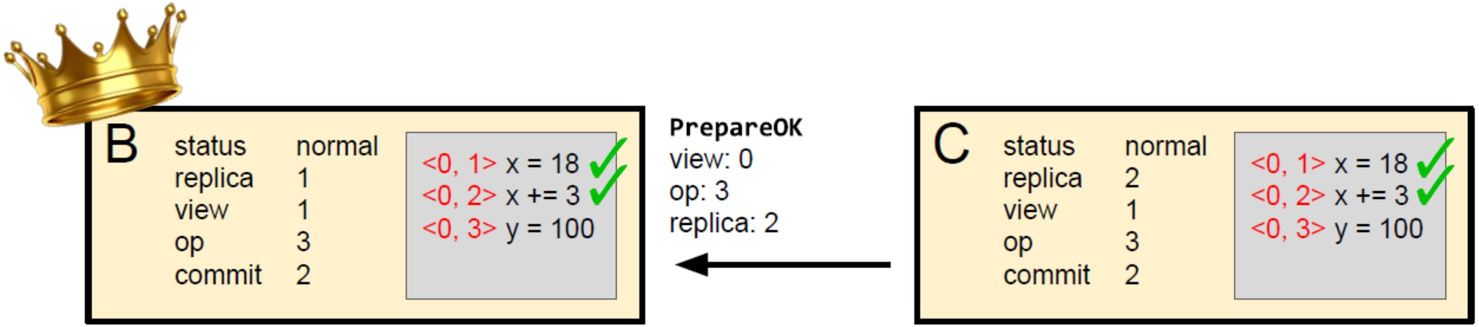
Reconcile

Notice <0, 3> is uncommitted and from an old view...

Do we commit it?



<view, op>
✓committed



B

status	normal
replica	1
view	1
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

PrepareOK
view: 0
op: 3
replica: 2

C

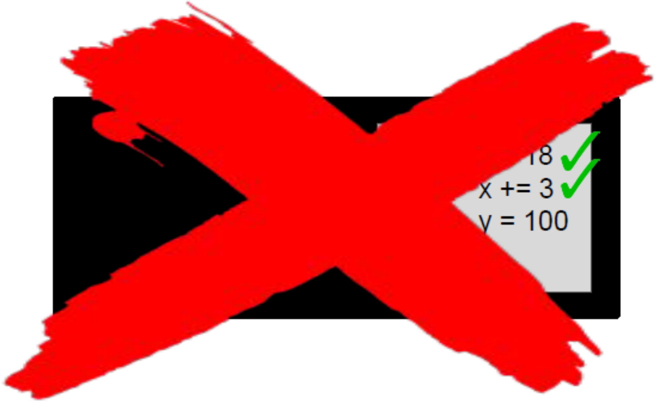
status	normal
replica	2
view	1
op	3
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

Reconciliation

Are uncommitted ops like <0, 3> guaranteed to survive into the new view?

What about committed ops? (e.g. <0, 1> and <0, 2>)

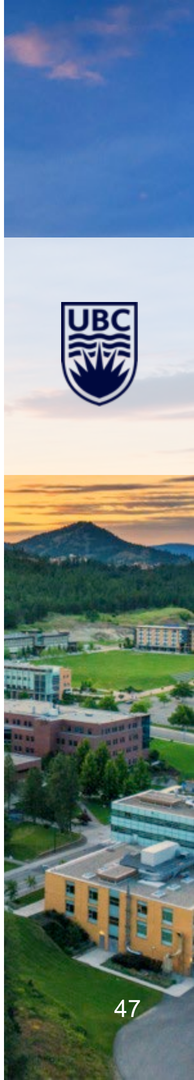


<view, op>

✓ *committed*

B			
status	normal	<0, 1>	x = 18 ✓
replica	1	<0, 2>	x += 3 ✓
view	1	<0, 3>	y = 100 ✓
op	3		
commit	3		

C			
status	normal	<0, 1>	x = 18 ✓
replica	2	<0, 2>	x += 3 ✓
view	1	<0, 3>	y = 100
op	3		
commit	2		



Summary: View Changes

New primary: picked based upon IP address (*any tie breaker will do*)

View change triggered by timeout – *initiated by any node*

Wait for *f* StartViewChange ops matching new primary's view number

Send DoViewChange op after *f* StartViewChange ops are received

Viewstamped replication guarantees liveness if no more than *f* replicas fail.

Lesson Review

Viewstamped Replication

Leader/Primary

- Receives client requests
- Maintains database
- Forwards requests to replicas

Replicas

- Can fail *and* restart
- Implement a read/write quorum consensus

Views

- Used to determine a new leader after leader failure
- Enables replicas resynchronizing/rejoining

Questions?

THE UNIVERSITY OF BRITISH COLUMBIA