Viewstamped Replication

10/19/18

MIDTERM

Next Wednesday 10/24 at 7 - 9pm in CS 104

Covers all material up to and including Monday’s lecture

Viewstamped Replication

A way to implement replicated state machines

Goal: strong consistency across replicas

Similar to Paxos and RAFT, but less popular

Viewstamped Replication

Normal operation

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

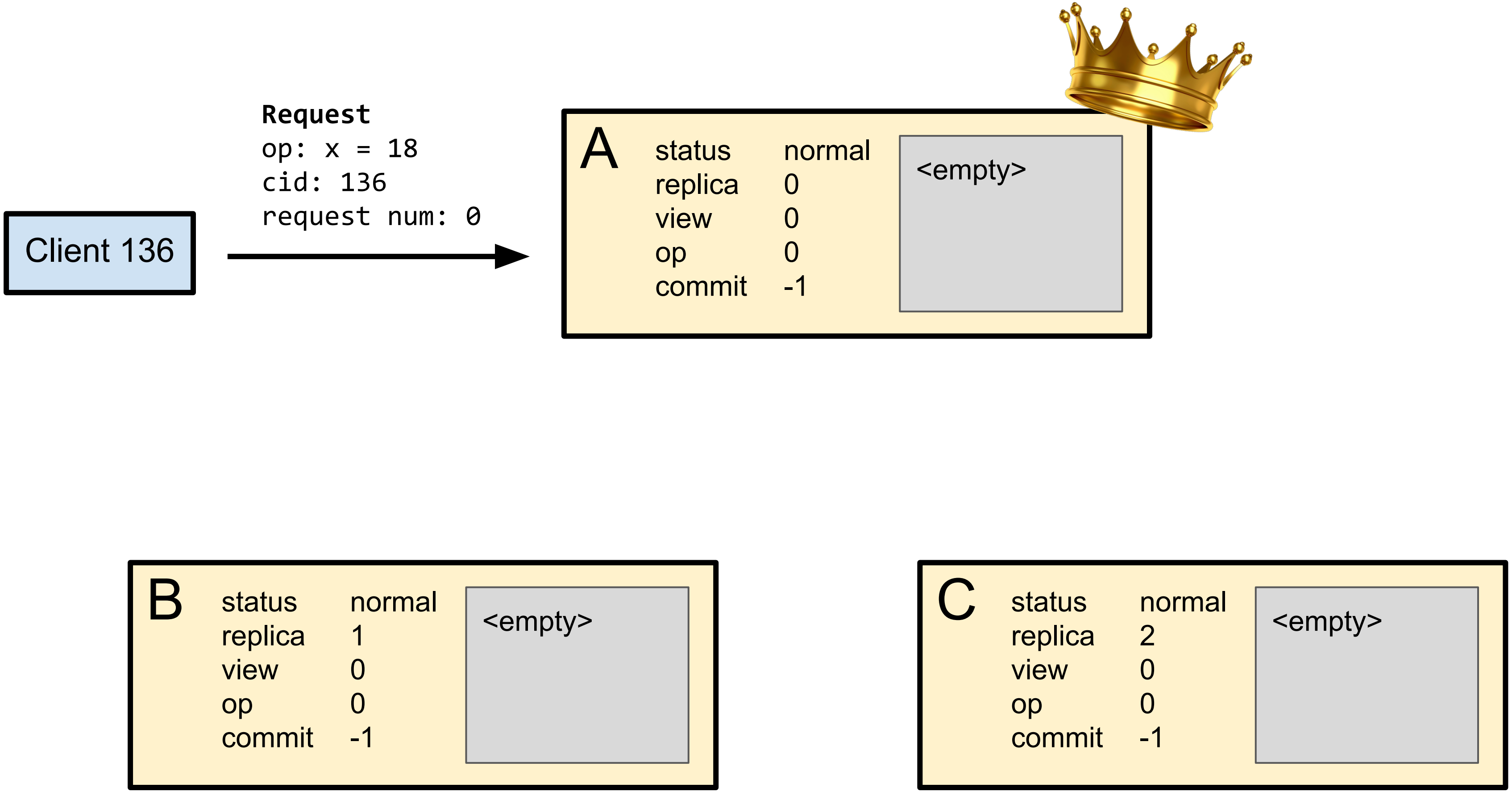
status	normal
replica	1
view	0
op	0
commit	-1

<empty>

C

status	normal
replica	2
view	0
op	0
commit	-1

<empty>



Client 136

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

A

status	normal
replica	0
view	0
op	0
commit	-1

<empty>

B

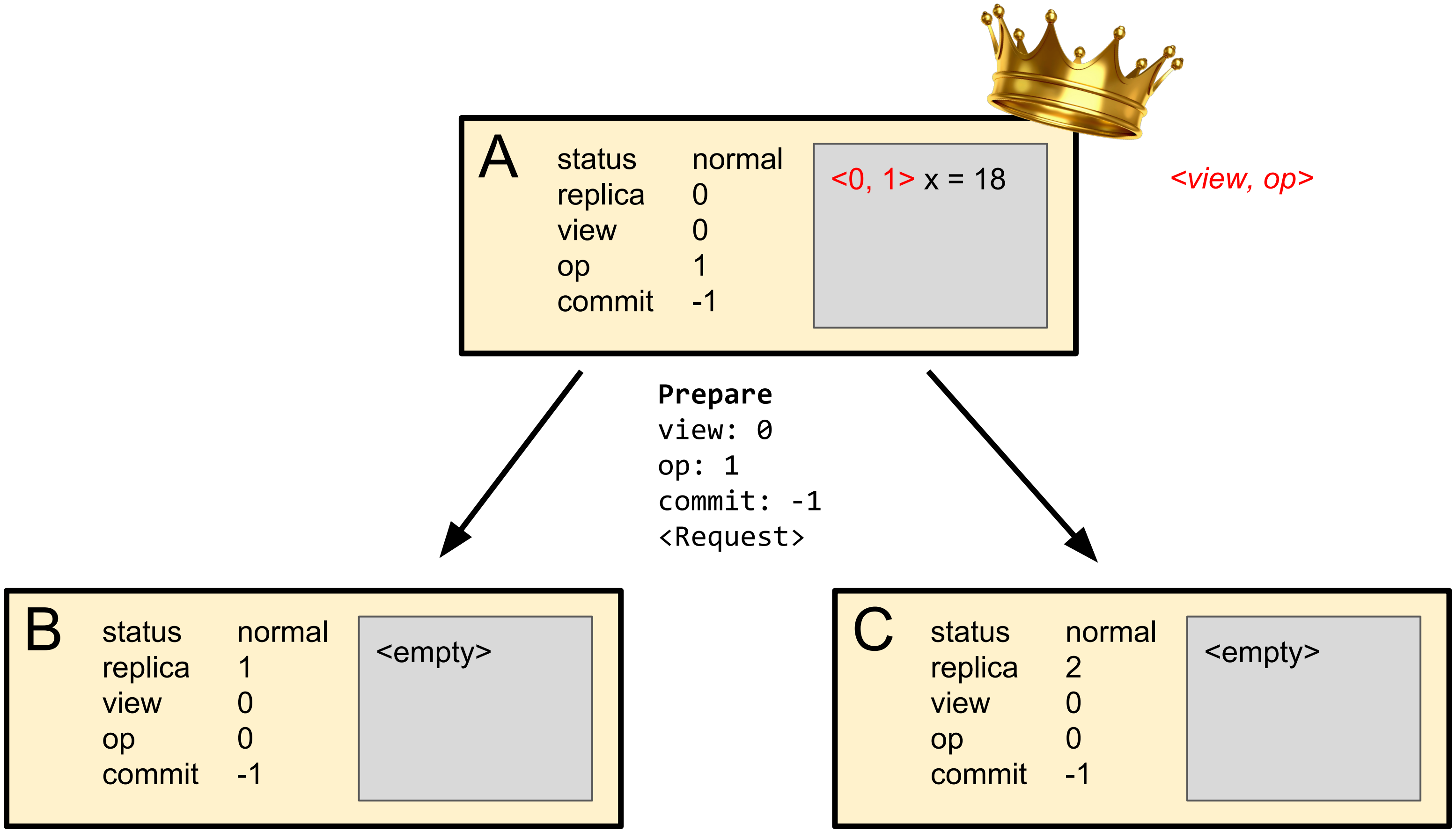
status	normal
replica	1
view	0
op	0
commit	-1

<empty>

C

status	normal
replica	2
view	0
op	0
commit	-1

<empty>



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

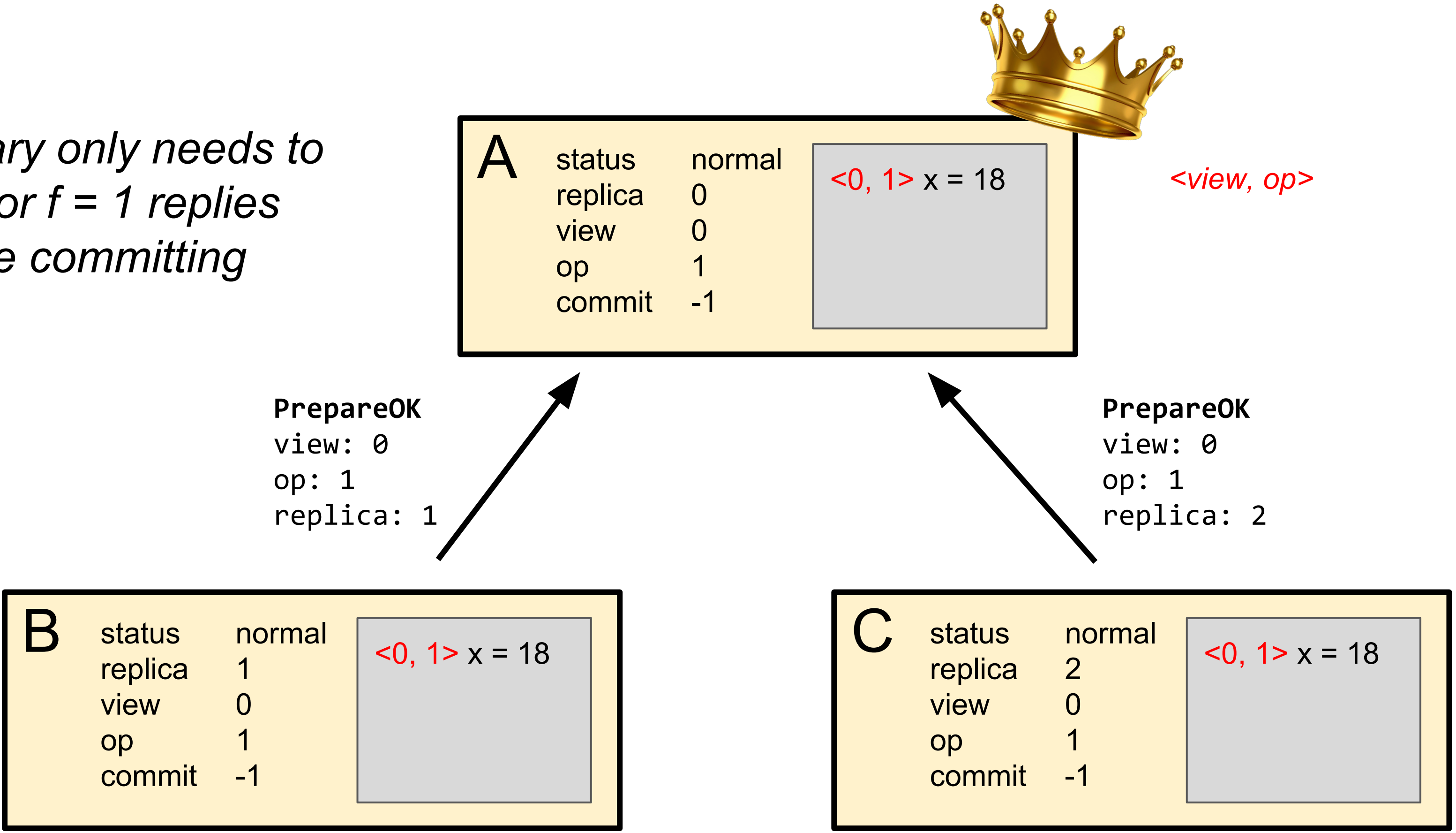
<empty>

C

status	normal
replica	2
view	0
op	0
commit	-1

<empty>

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

<view, op>

PrepareOK
view: 0
op: 1
replica: 1

PrepareOK
view: 0
op: 1
replica: 2

B

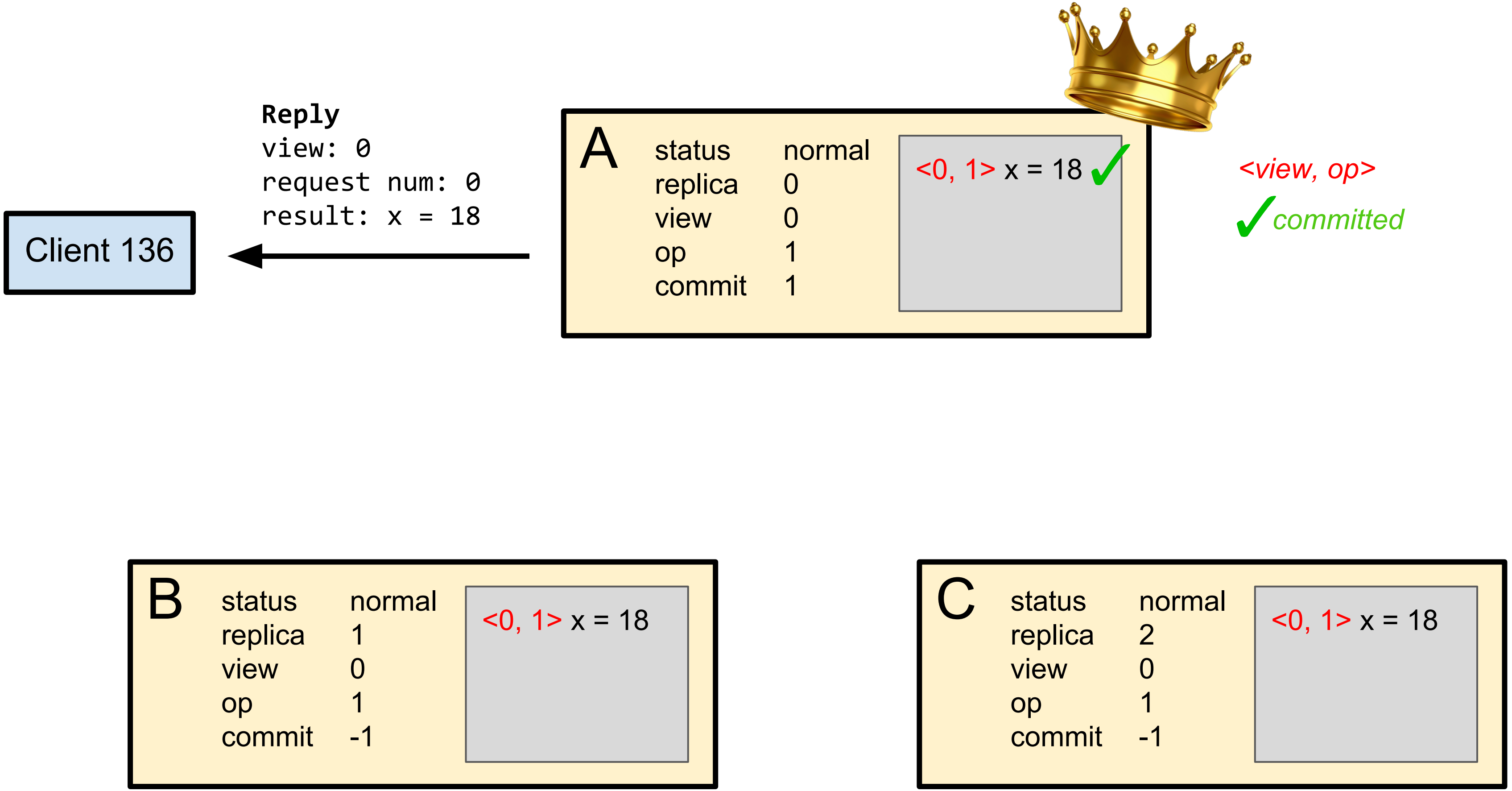
status	normal
replica	1
view	0
op	1
commit	-1

<0, 1> x = 18

C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18



Client 136

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

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

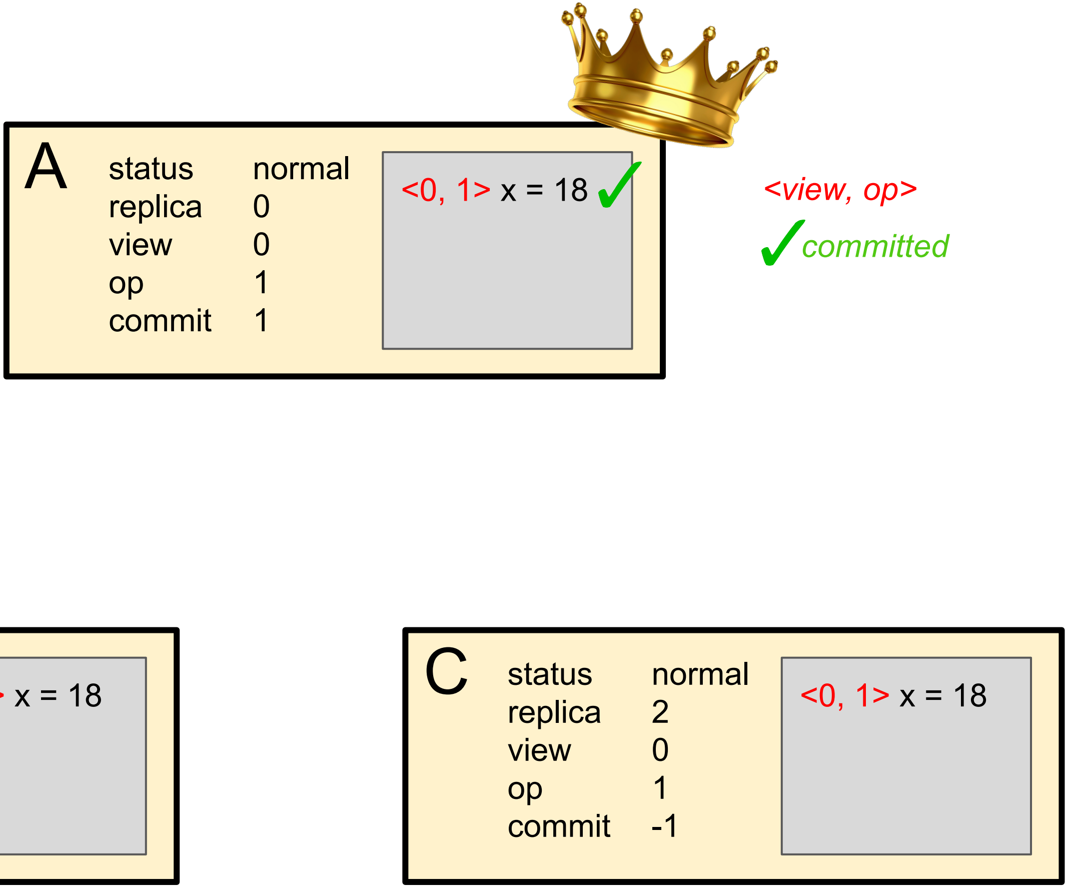
<0, 1> x = 18

C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18

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

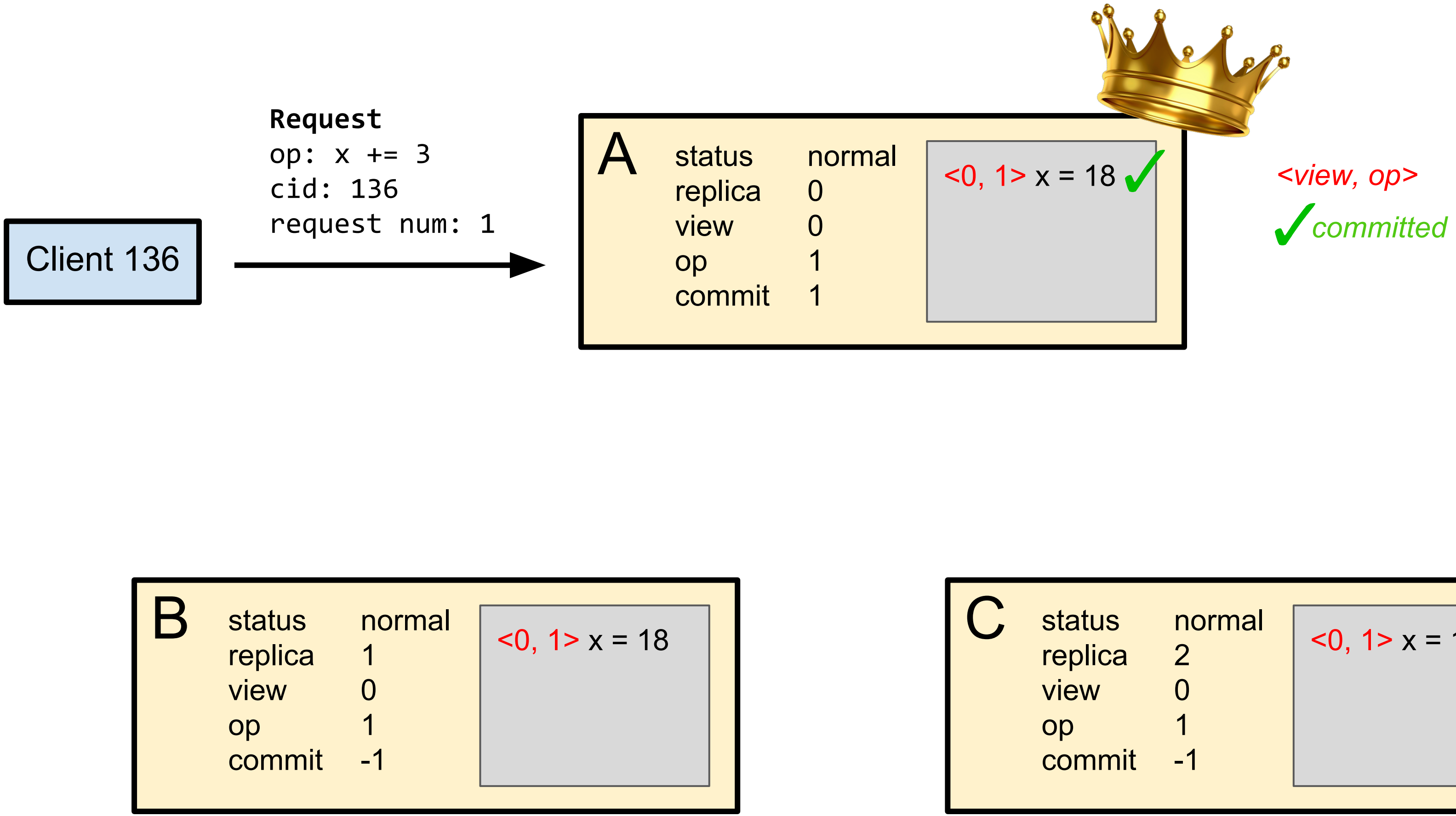
status	normal
replica	1
view	0
op	1
commit	-1

<0, 1> x = 18

C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18



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

Client 136

A

status	normal
replica	0
view	0
op	1
commit	1

<0, 1> x = 18 ✓

<view, op>

✓*committed*

B

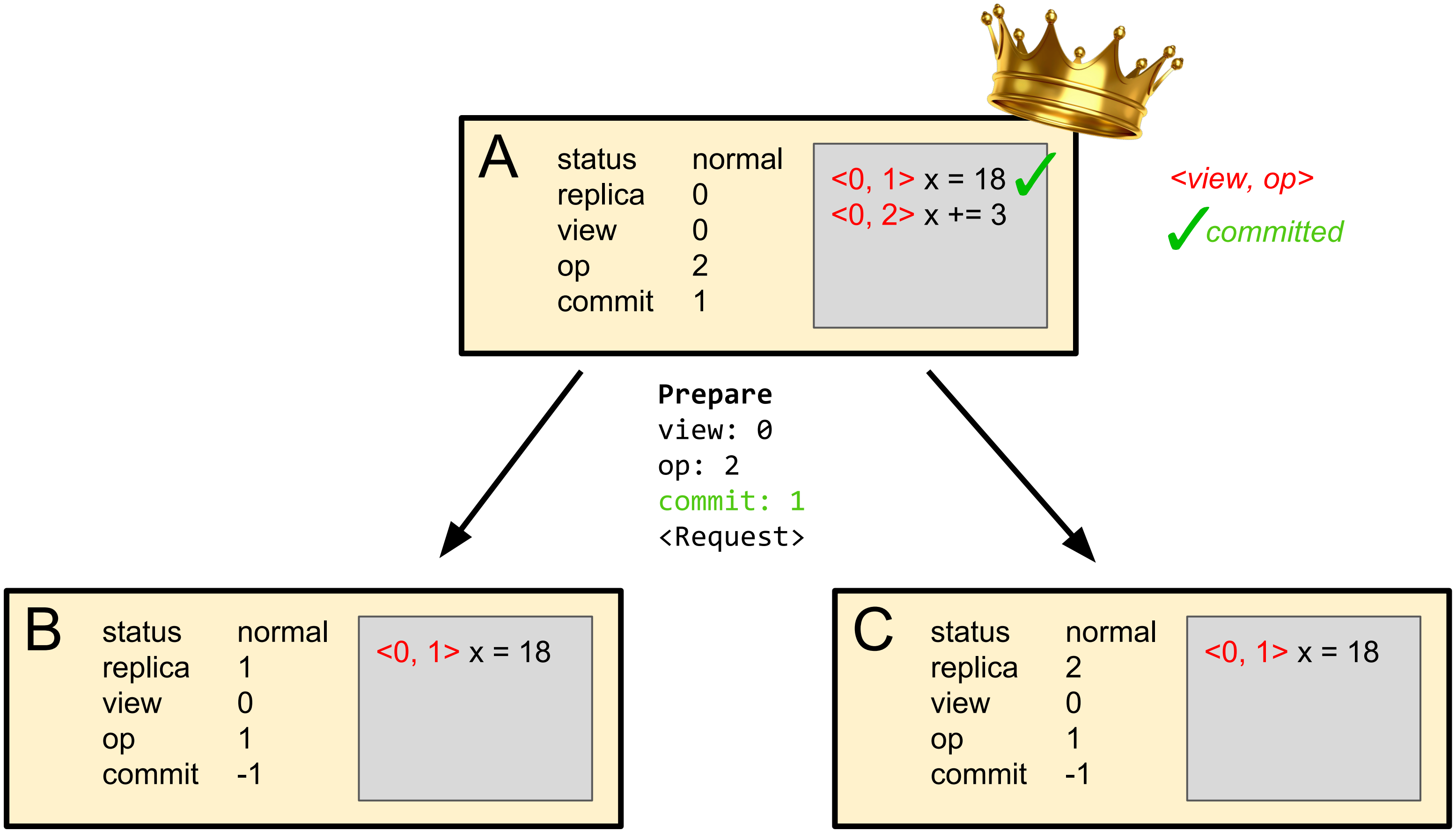
status	normal
replica	1
view	0
op	1
commit	-1

<0, 1> x = 18

C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18



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

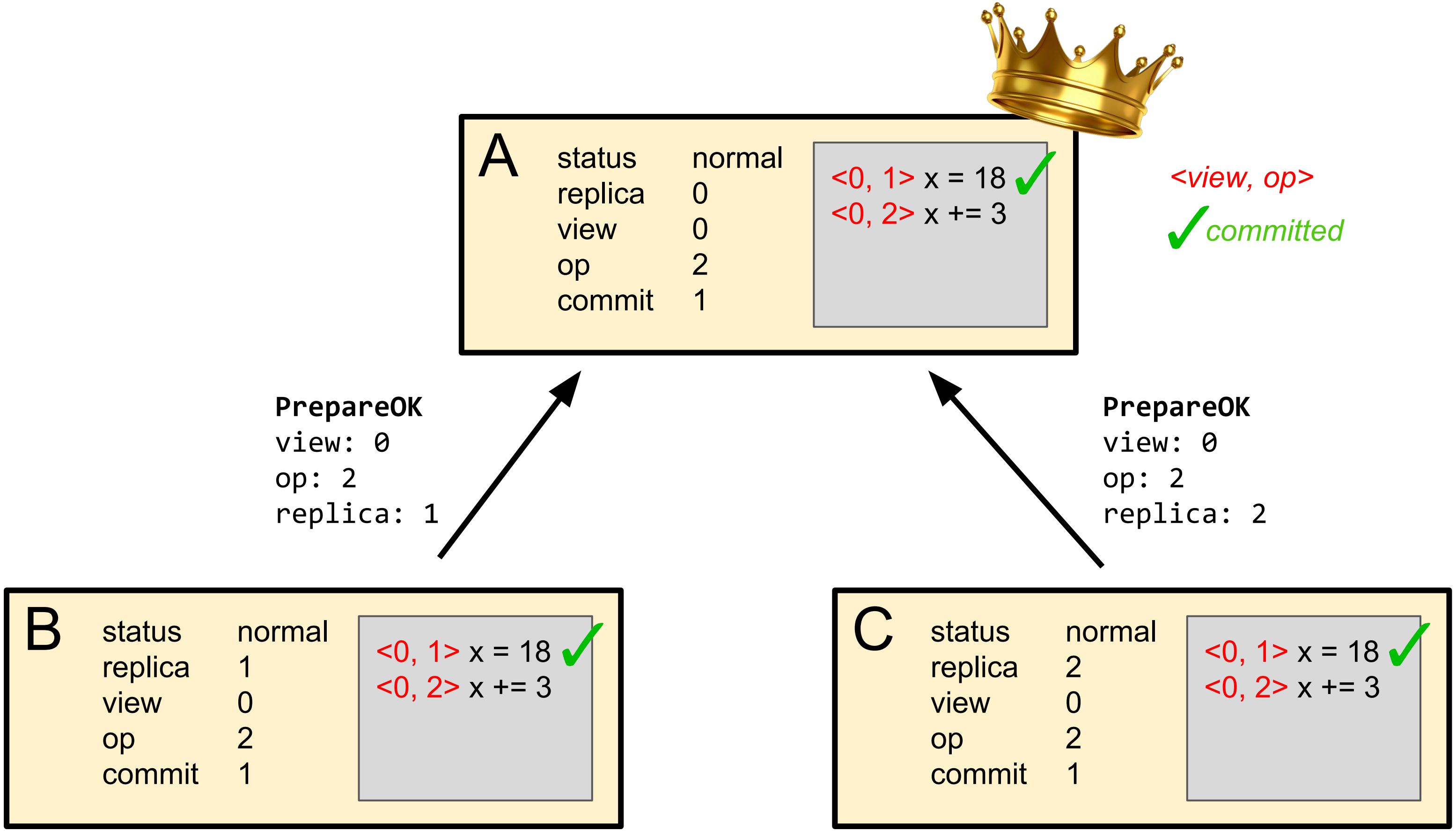
status	normal
replica	1
view	0
op	1
commit	-1

<0, 1> x = 18

C

status	normal
replica	2
view	0
op	1
commit	-1

<0, 1> x = 18



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

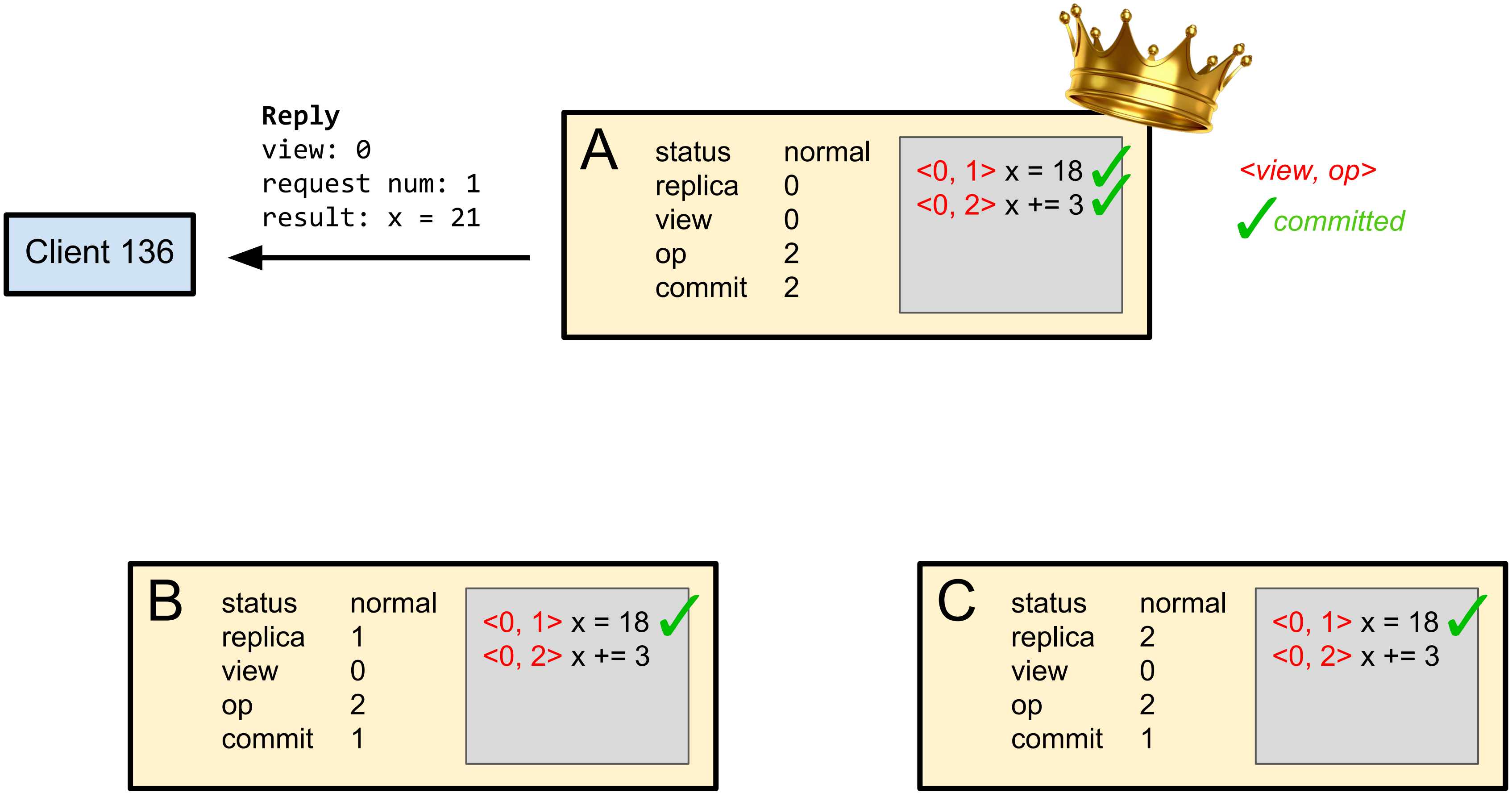
status	normal
replica	1
view	0
op	2
commit	1

<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



```
Reply
view: 0
request num: 1
result: x = 21
```

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	1

<0, 1> x = 18 ✓
<0, 2> x += 3

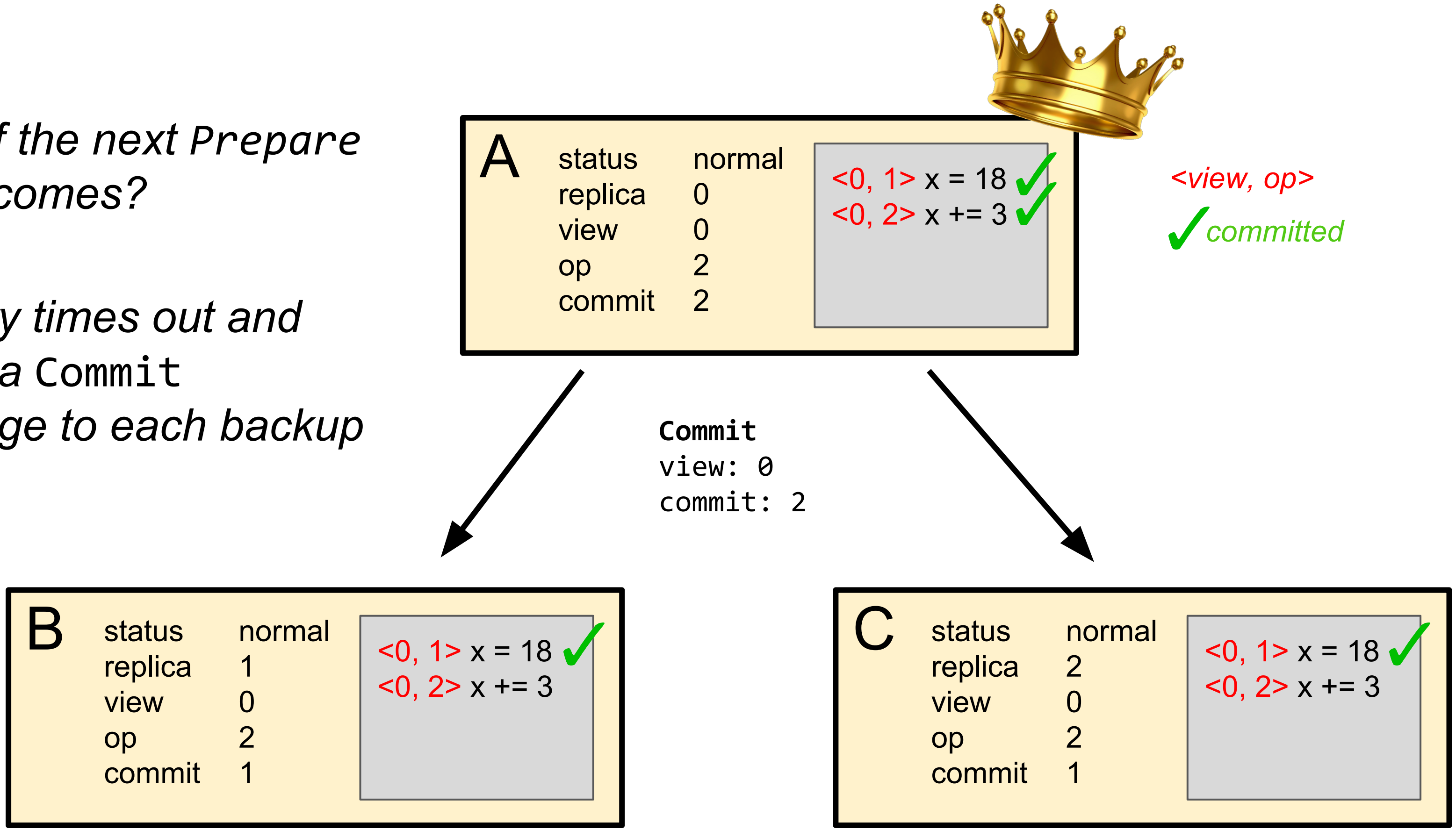
C

status	normal
replica	2
view	0
op	2
commit	1

<0, 1> x = 18 ✓
<0, 2> x += 3

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

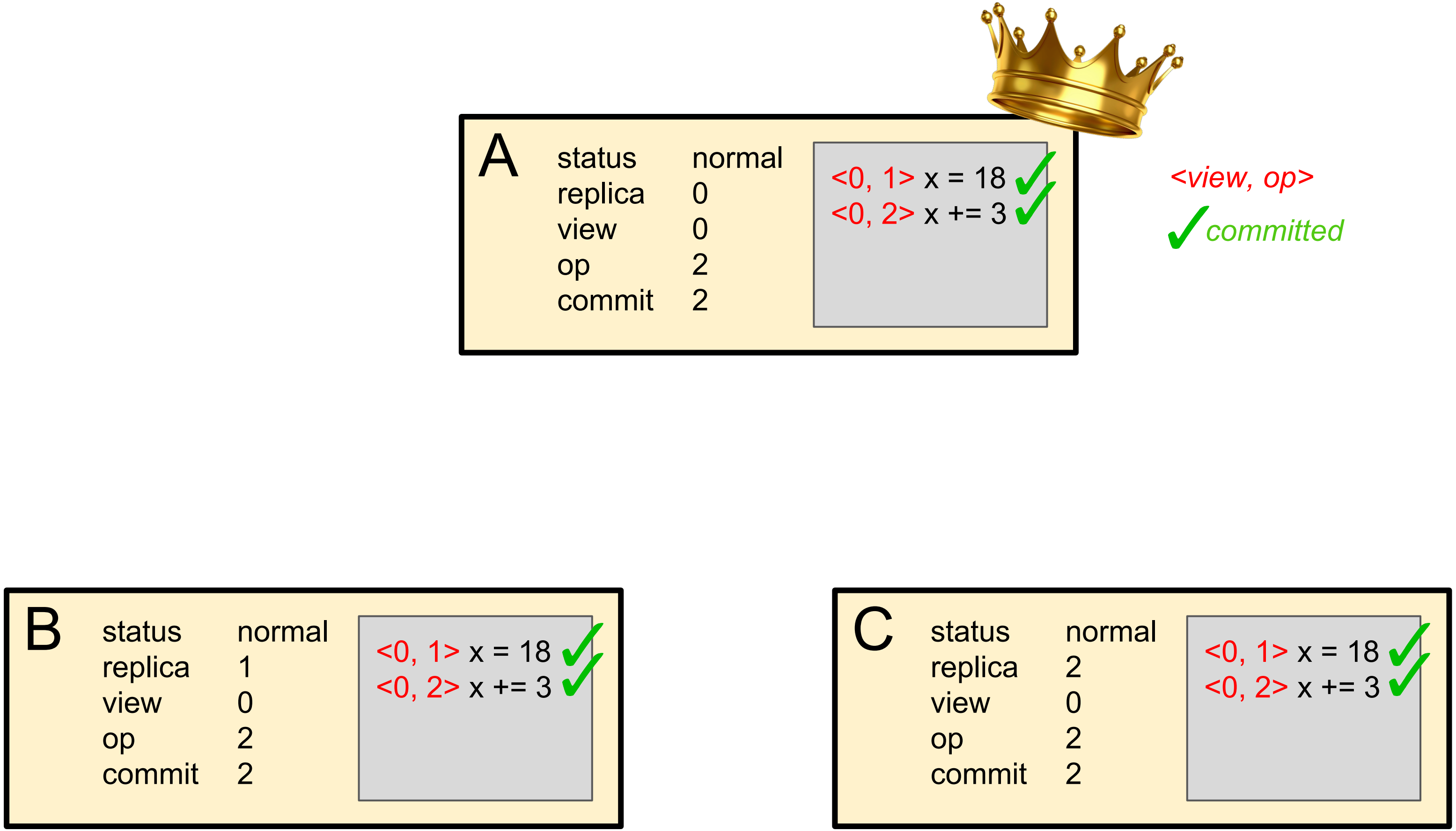
status	normal
replica	1
view	0
op	2
commit	1

<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



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

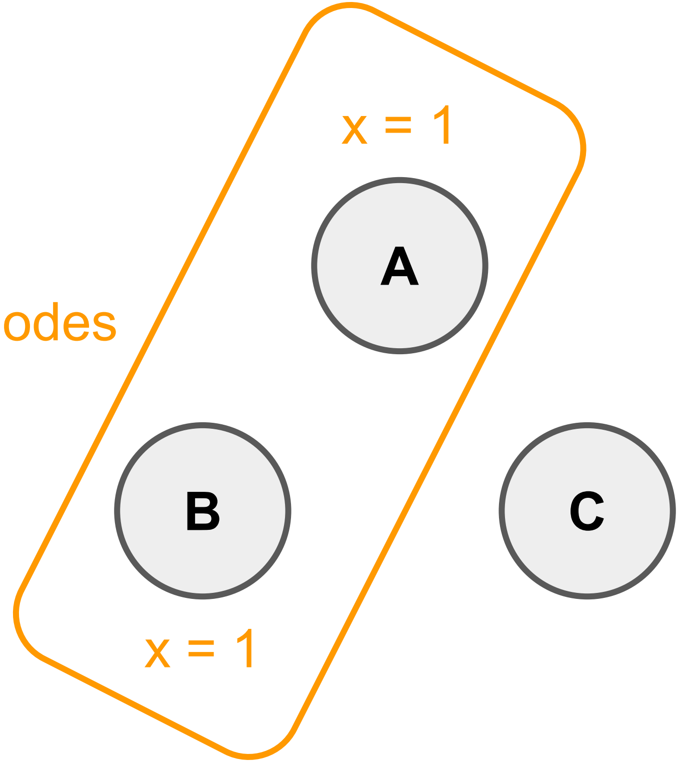
<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

Why is waiting for *f* nodes enough?

Op is guaranteed to have been executed on *f + 1* nodes (majority)

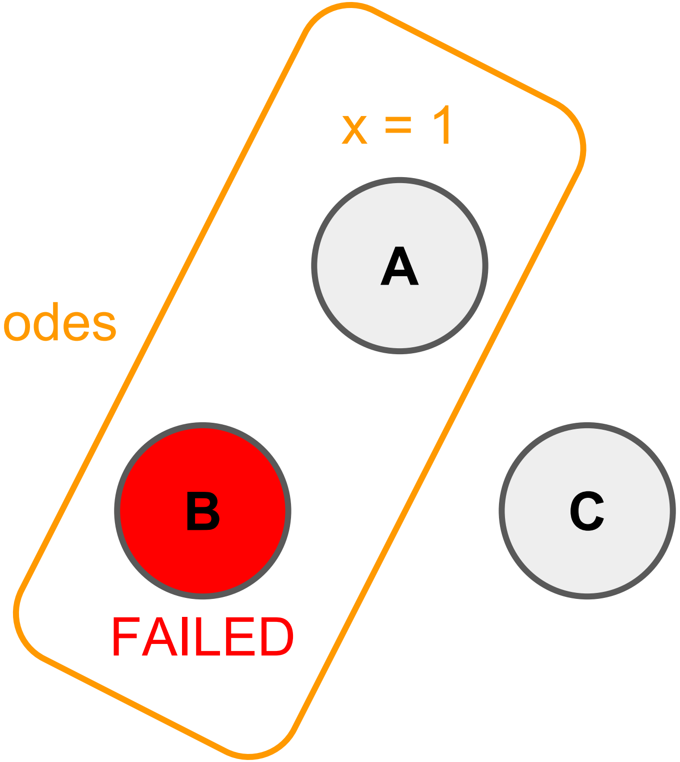
Overlapping quorums

Write quorum contains *f* + *1* nodes

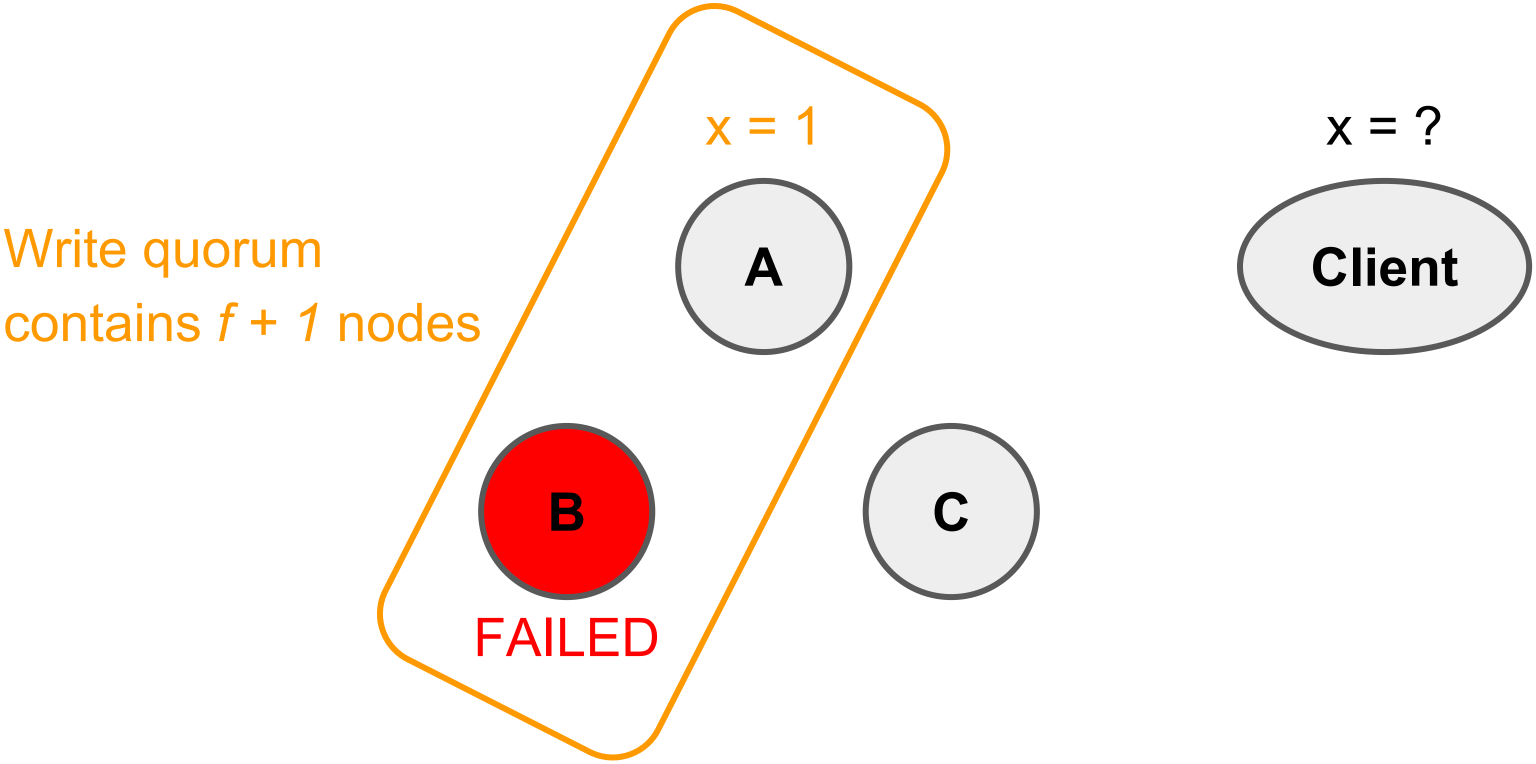


Overlapping quorums

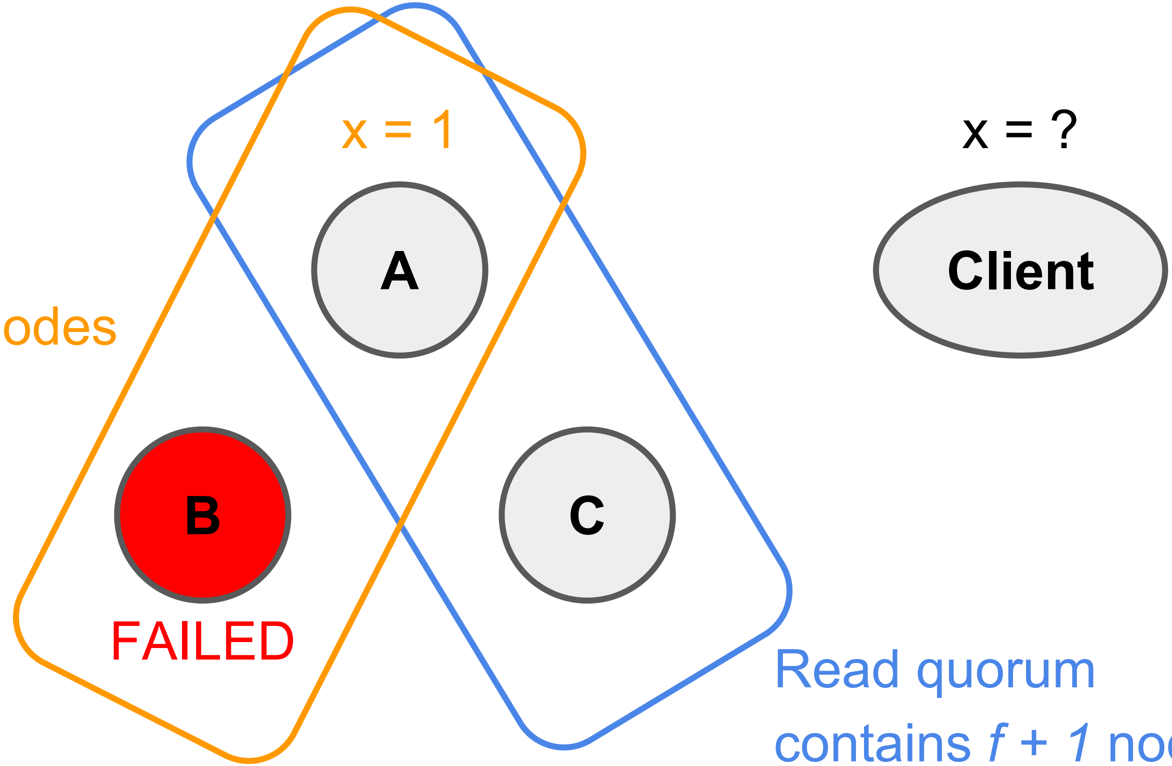
Write quorum contains *f + 1* nodes



Overlapping quorums



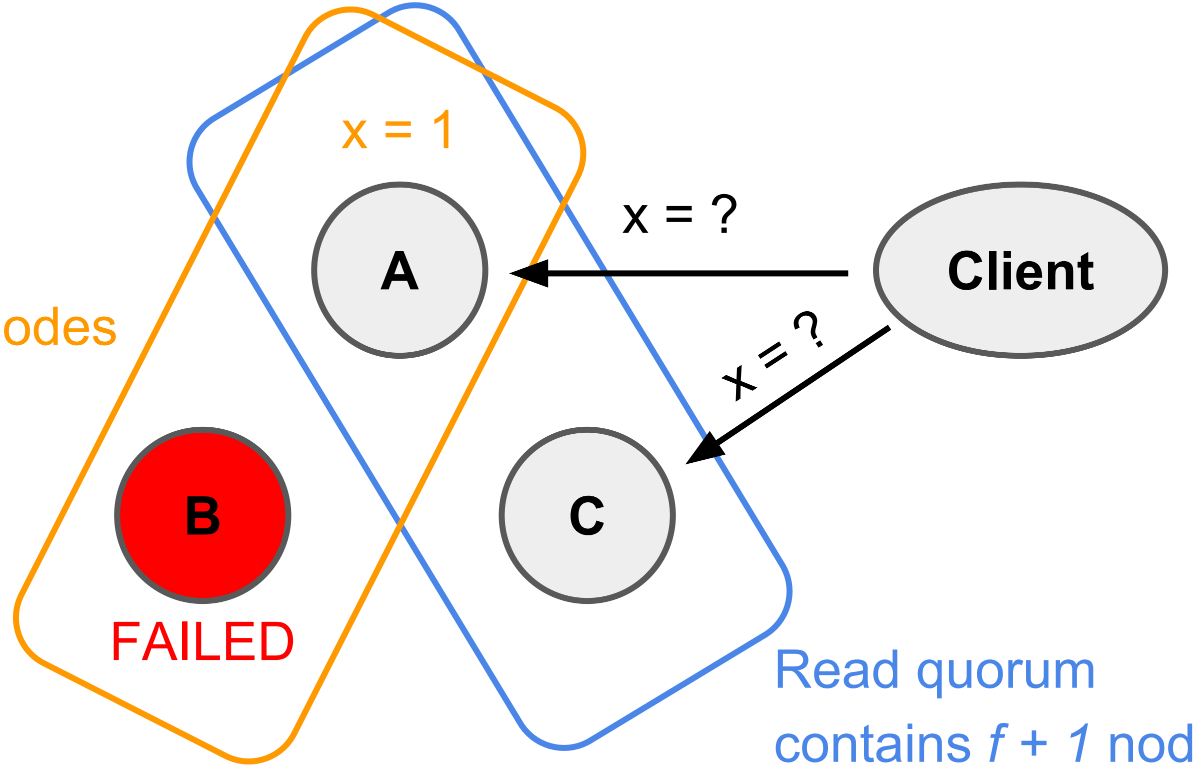
Overlapping quorums



Write quorum contains *f* + *1* nodes

Read quorum contains *f* + *1* nodes

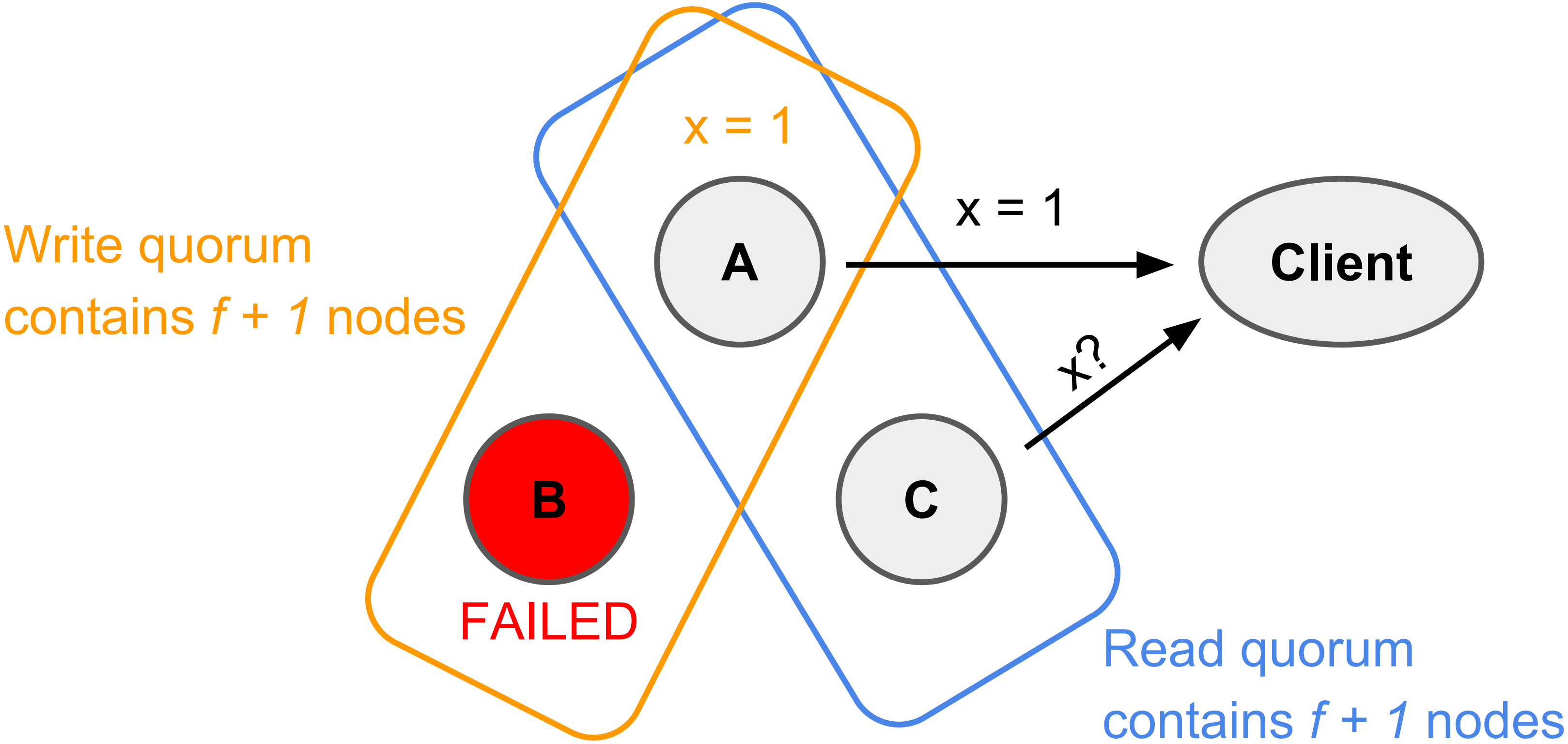
Overlapping quorums



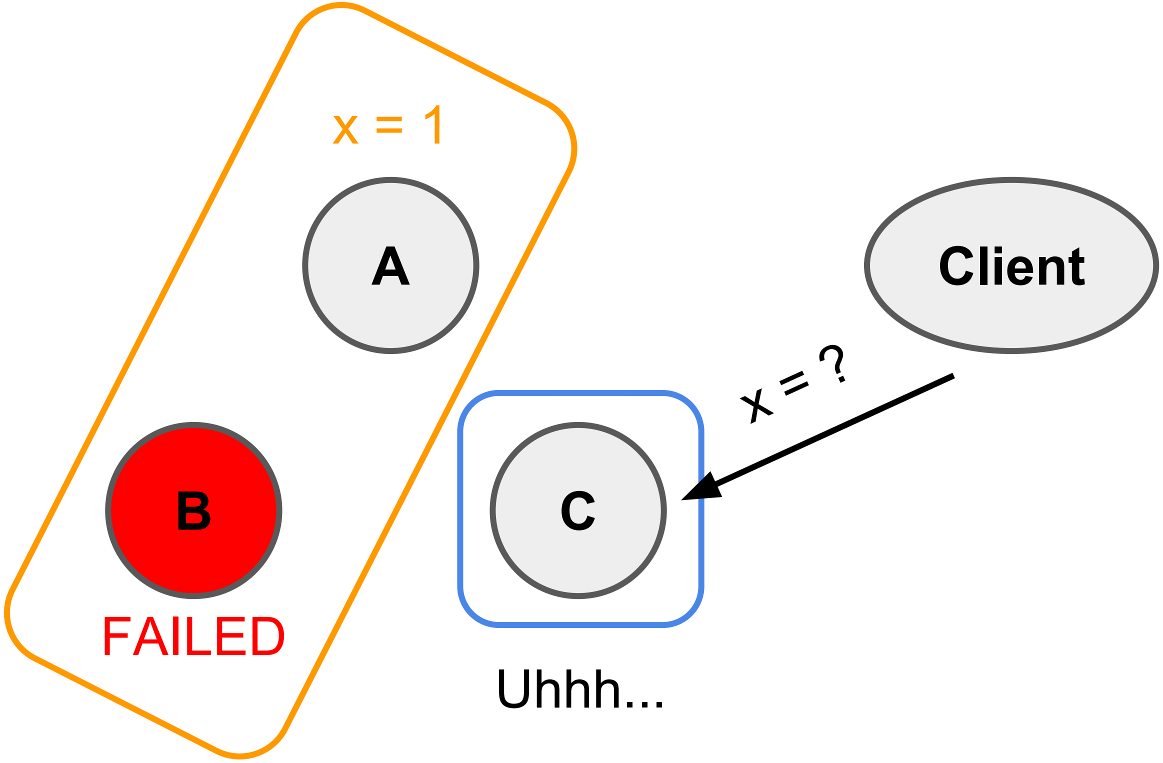
Write quorum
contains *f* + *1* nodes

Read quorum
contains *f* + *1* nodes

Overlapping quorums

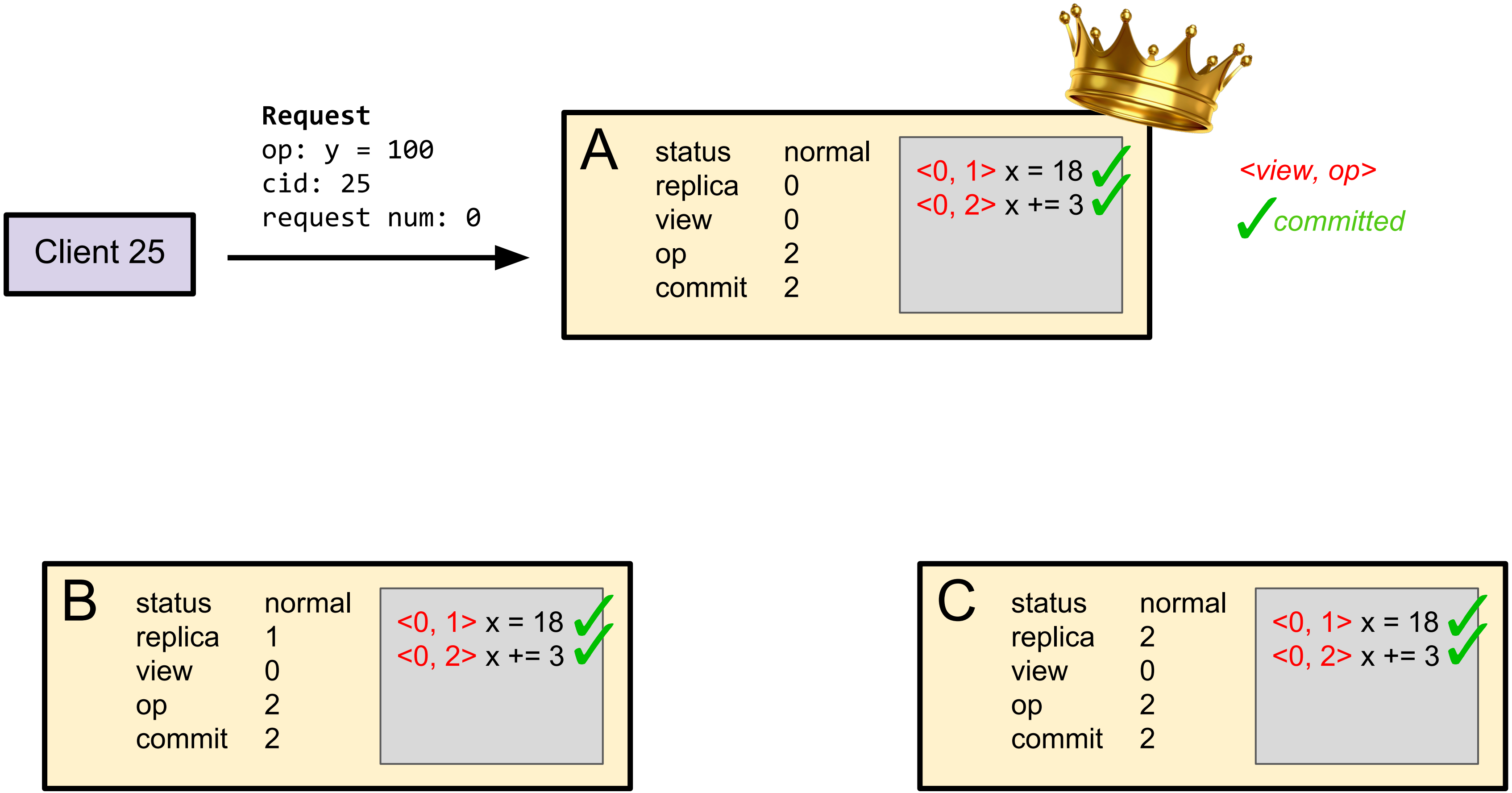

x = 1
Write quorum contains *f* + *1* nodes
A
x = 1
Client
x?
B
FAILED
C
Read quorum contains *f* + *1* nodes

Non-overlapping quorums?



Viewstamped replication

View change



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

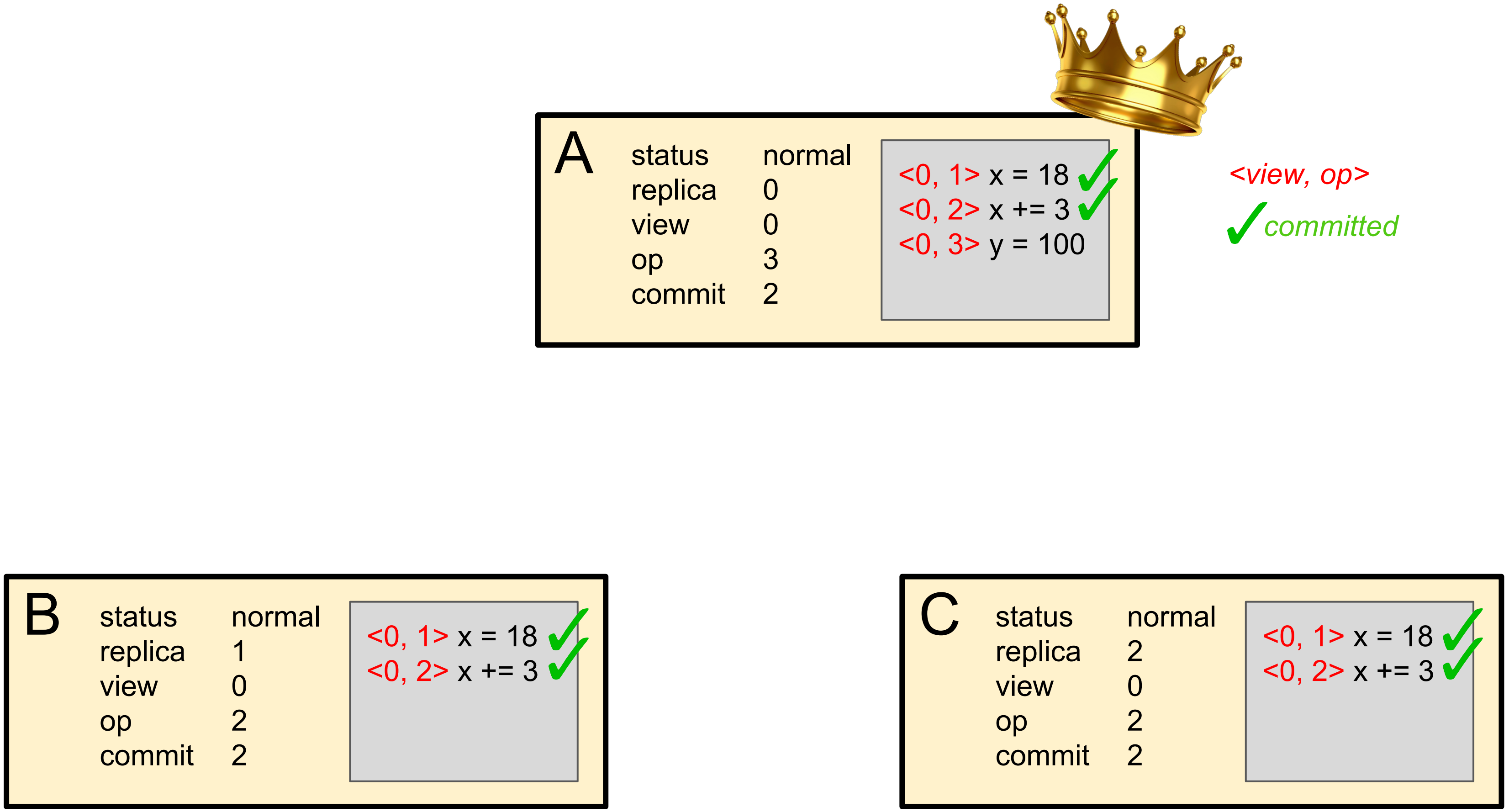
status	normal
replica	1
view	0
op	2
commit	2

<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 ✓



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

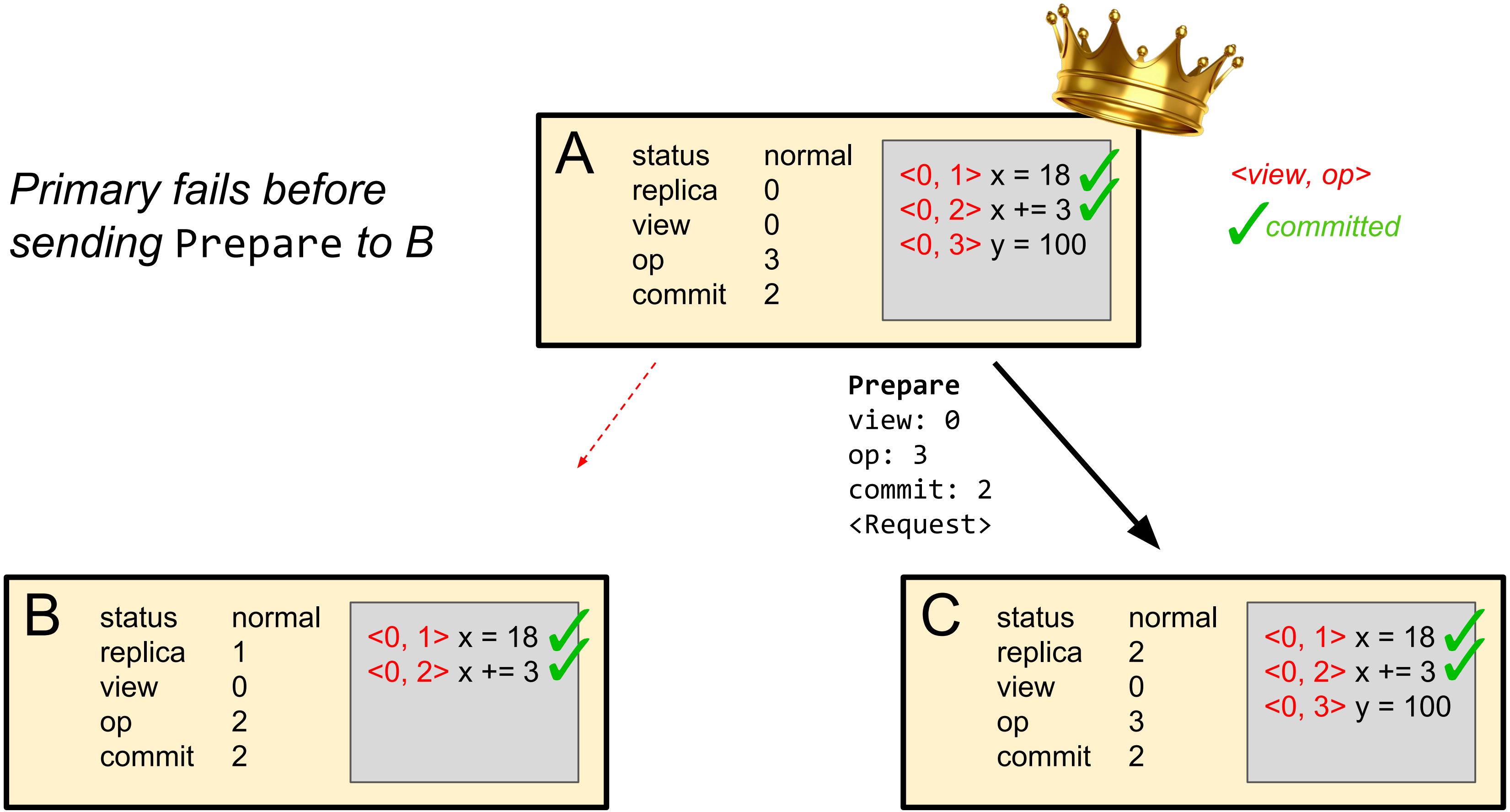
<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

C

status	normal
replica	2
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

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

```
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 ✓

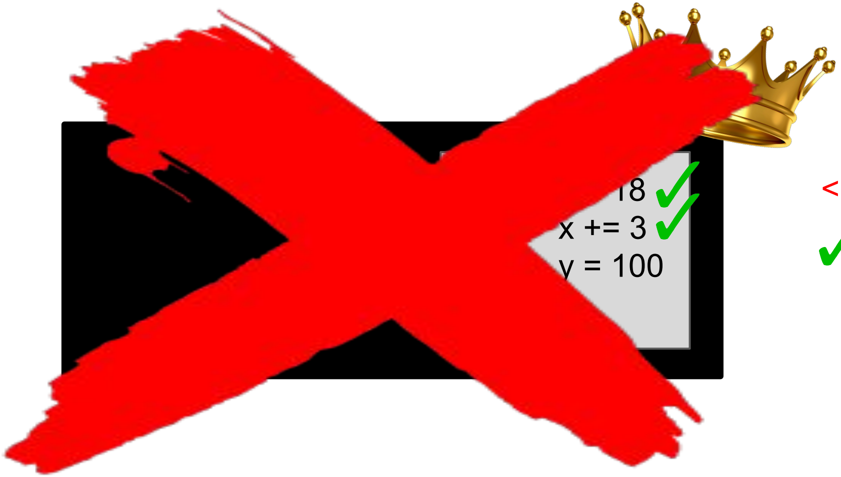
C

status	normal
replica	2
view	0
op	3
commit	2

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

<view, op>
✓ committed

Logs are out of sync



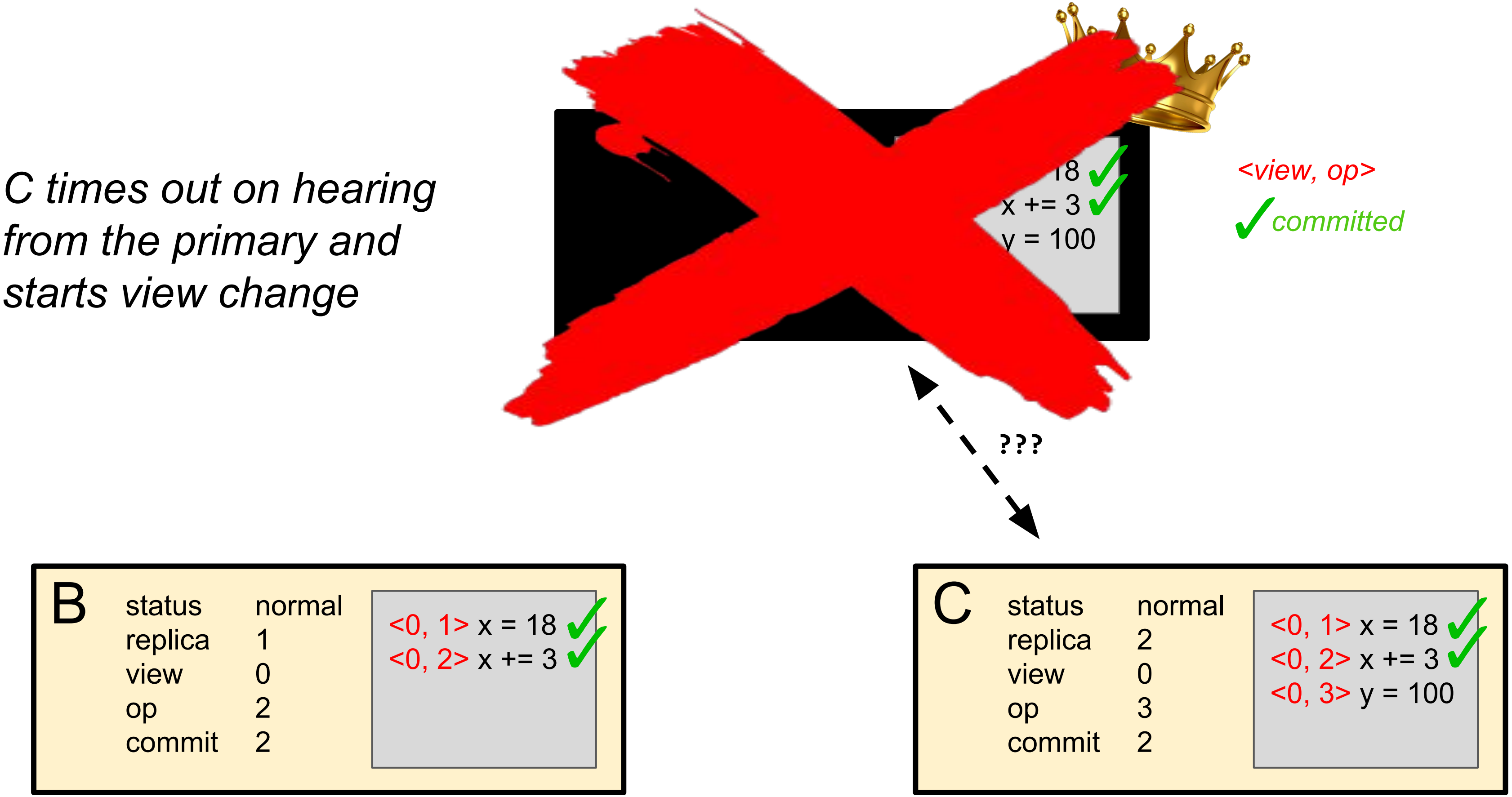
<view, op>

✓ *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	

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



B

status	normal
replica	1
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

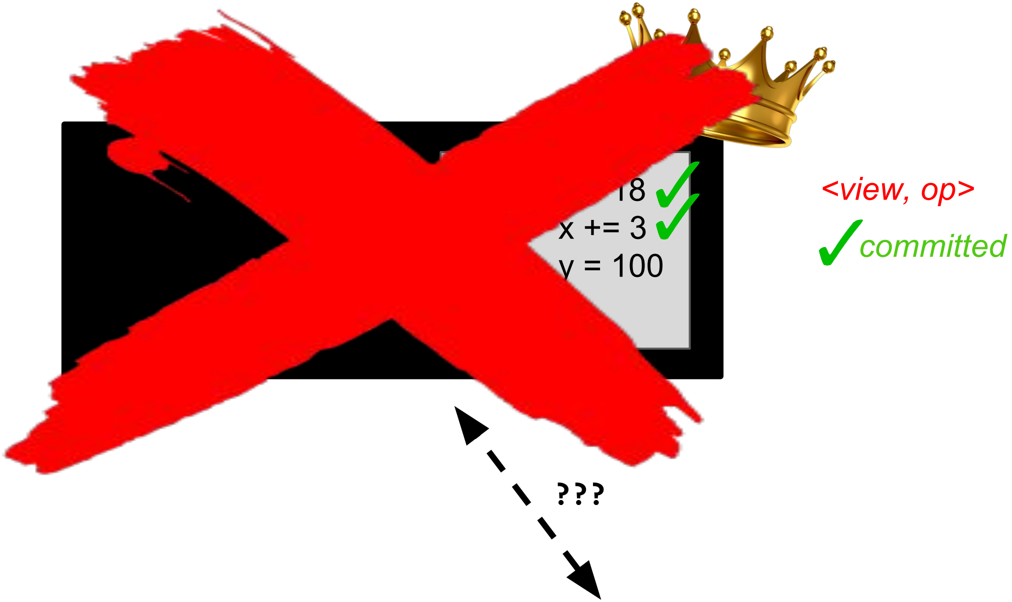
C

status	normal
replica	2
view	0
op	3
commit	2

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

Who is the new primary?

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



<view, op>

✓committed

B

status	normal
replica	1
view	0
op	2
commit	2

<0, 1> x = 18 ✓
<0, 2> x += 3 ✓

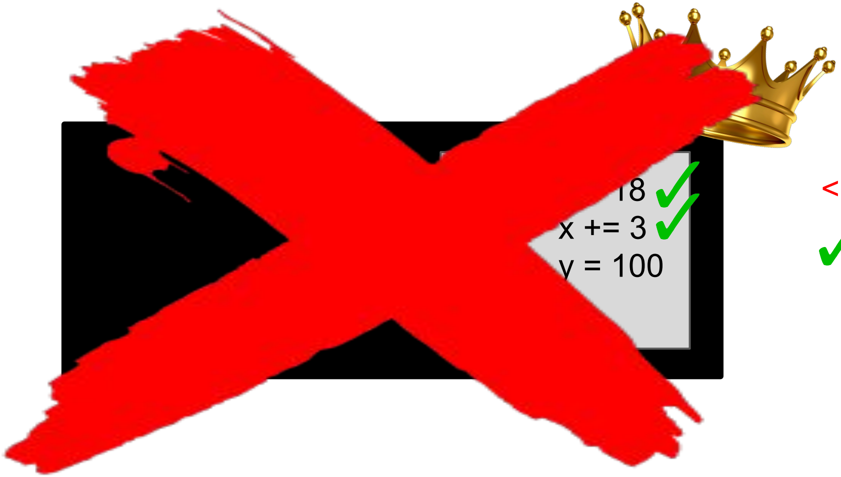
C

status	normal
replica	2
view	0
op	3
commit	2

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

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 ✓

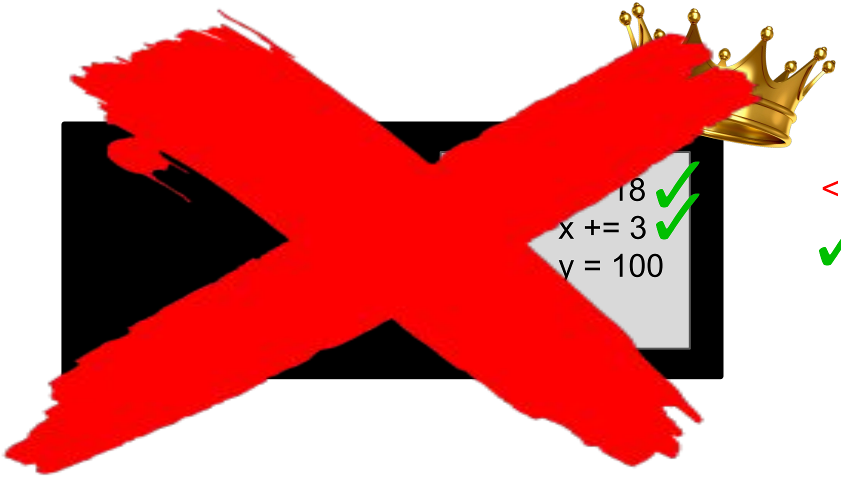
C

status	normal
replica	2
view	0
op	3
commit	2

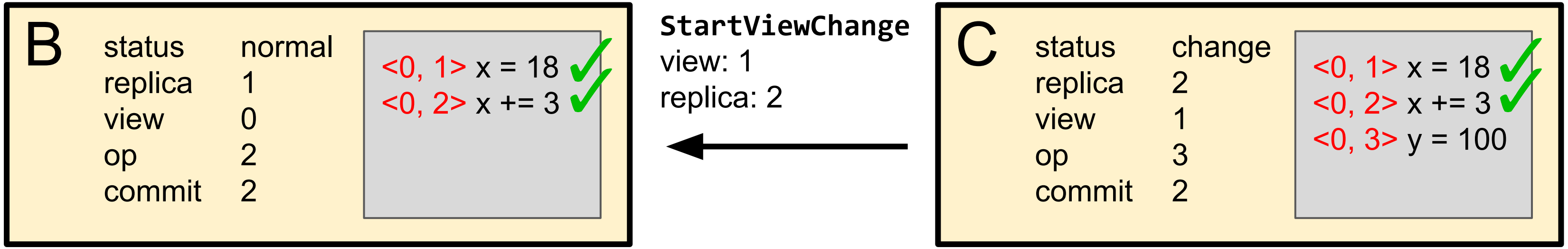
<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

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

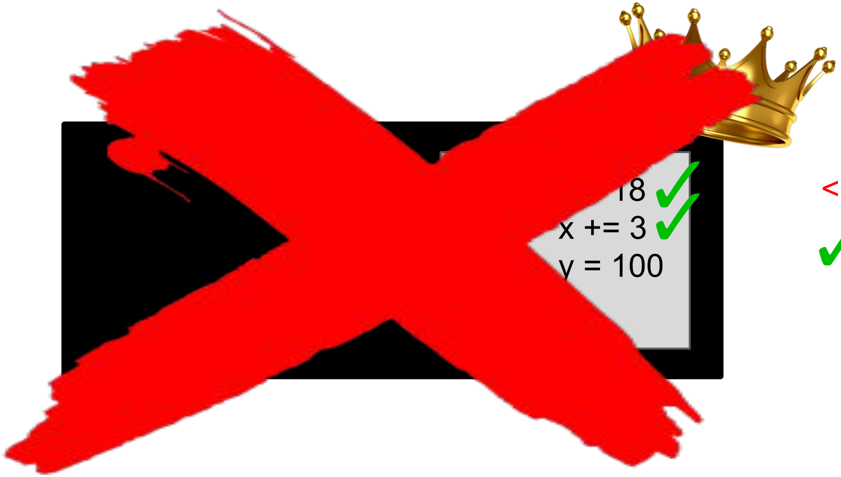
C

status	change
replica	2
view	1
op	3
commit	2

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

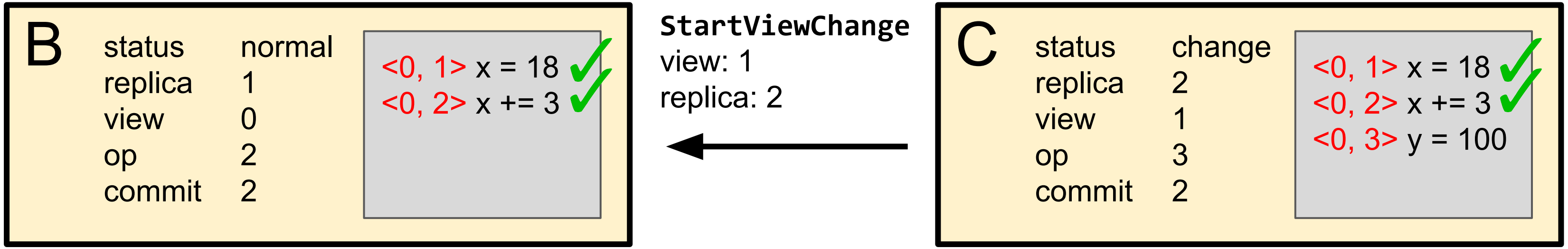
Receive SVC where:

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

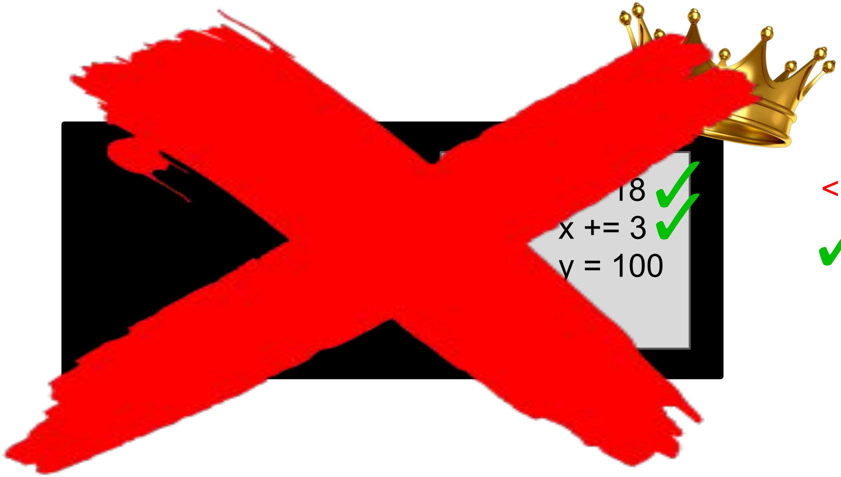
C

status	change
replica	2
view	1
op	3
commit	2

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

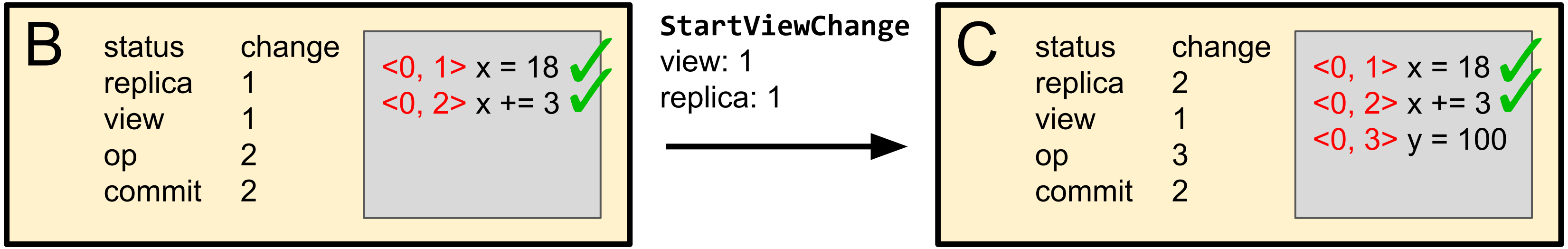
Receive SVC where:

```
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

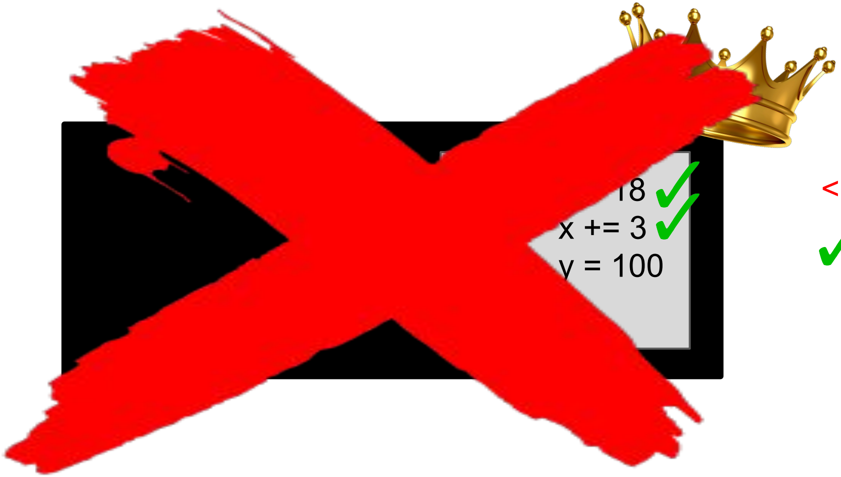
C

status	change
replica	2
view	1
op	3
commit	2

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

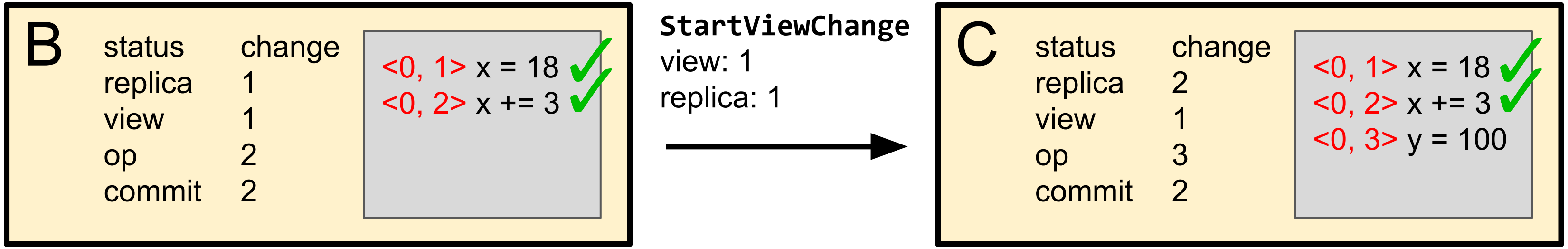
Receive* f *SVCs where:

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



<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

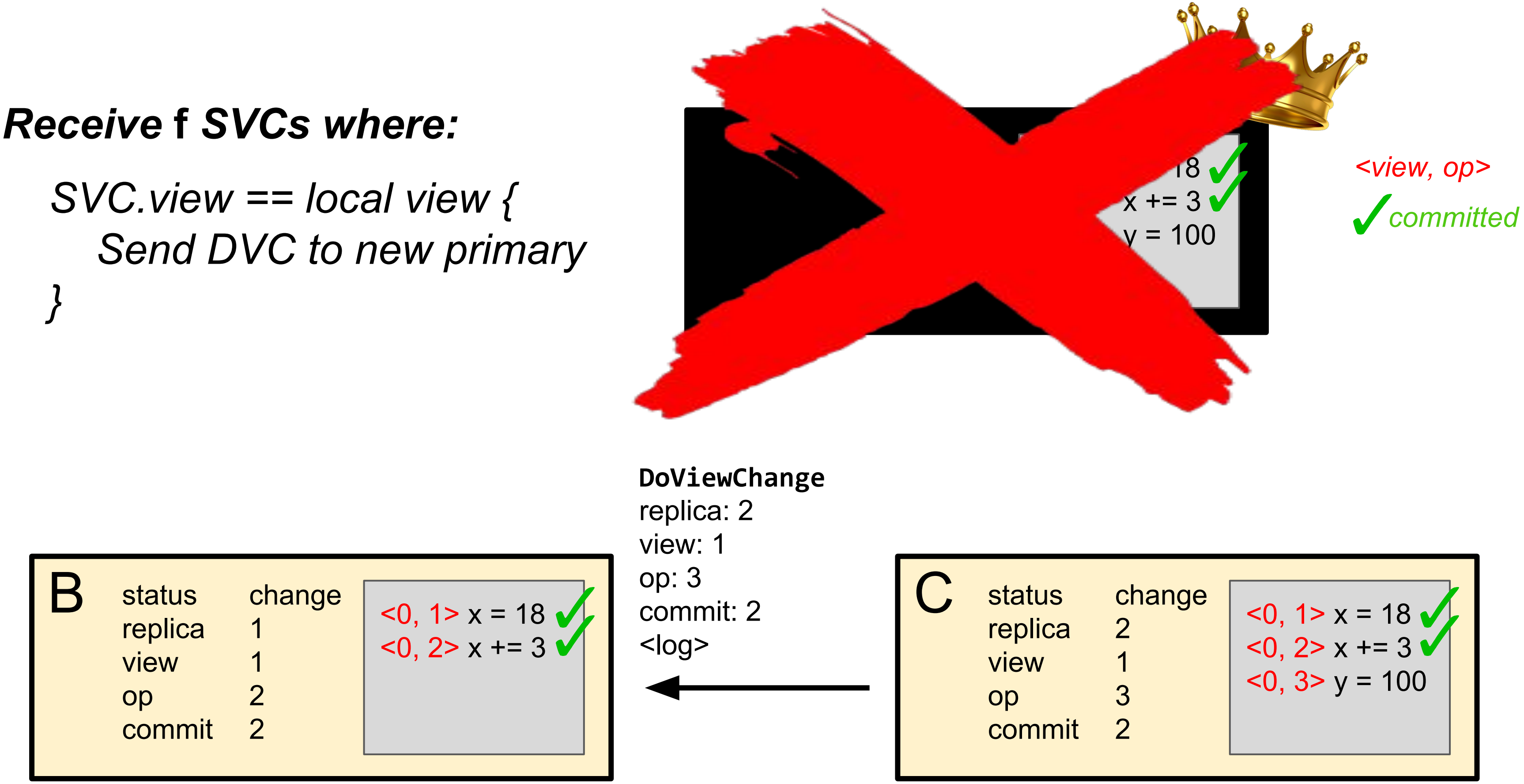
C

status	change
replica	2
view	1
op	3
commit	2

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

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>

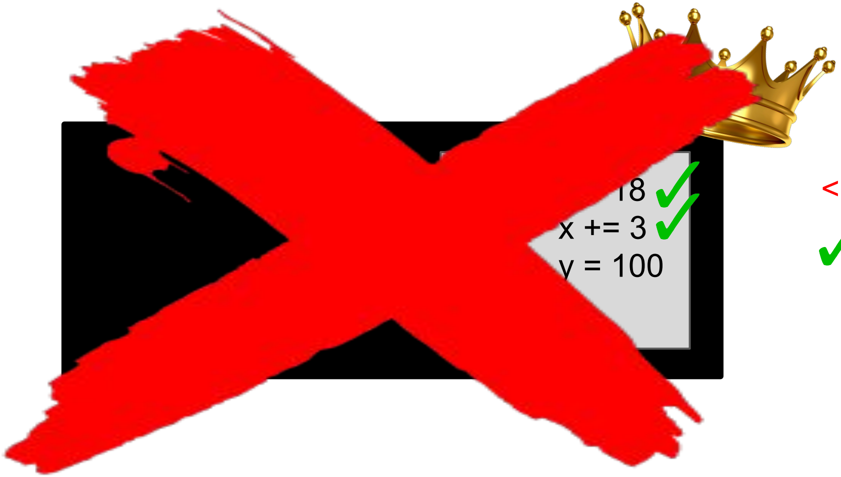
B		
status	change	<0, 1> x = 18 ✓
replica	1	<0, 2> x += 3 ✓
view	1	
op	2	
commit	2	

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

Logs 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

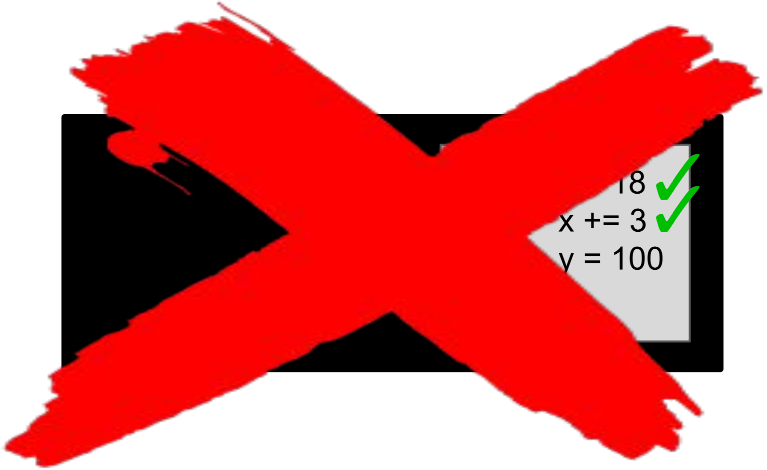
<0, 1> x = 18 ✓
<0, 2> x += 3 ✓
<0, 3> y = 100

Receive* f *DVCs:

Become new primary

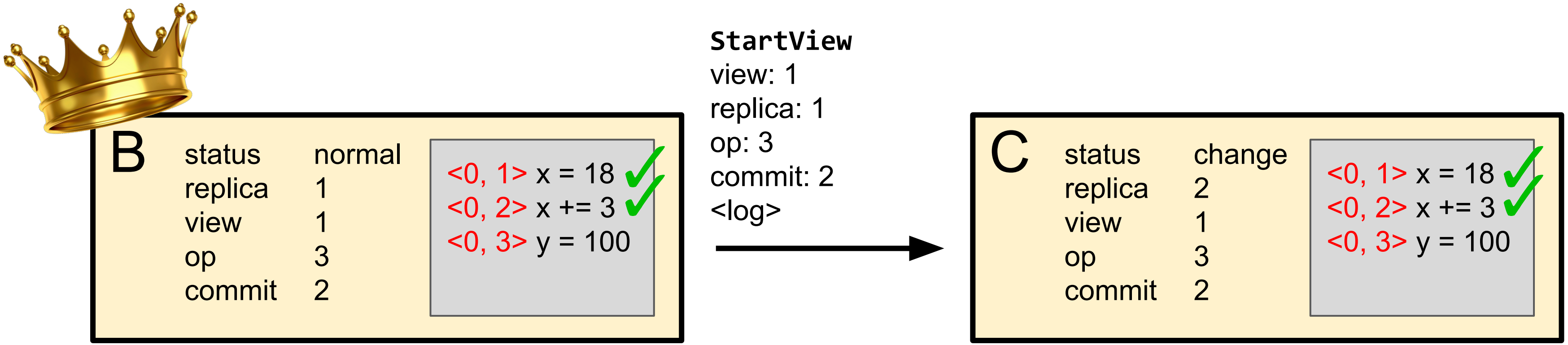
Send `StartView` *to others*

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>

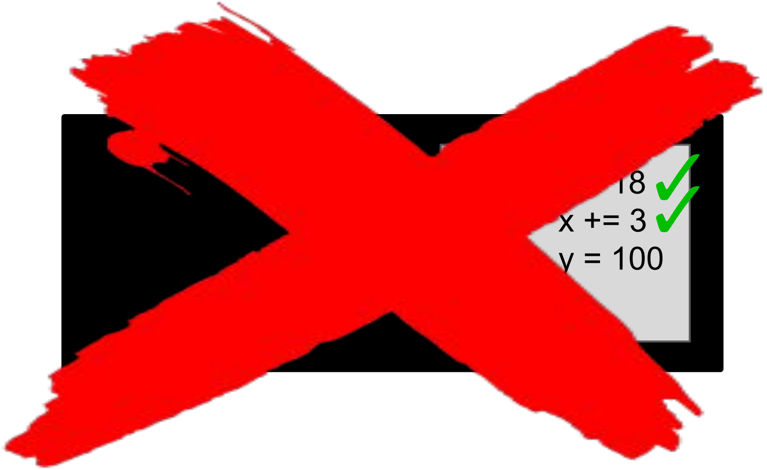
C

status	change
replica	2
view	1
op	3
commit	2

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

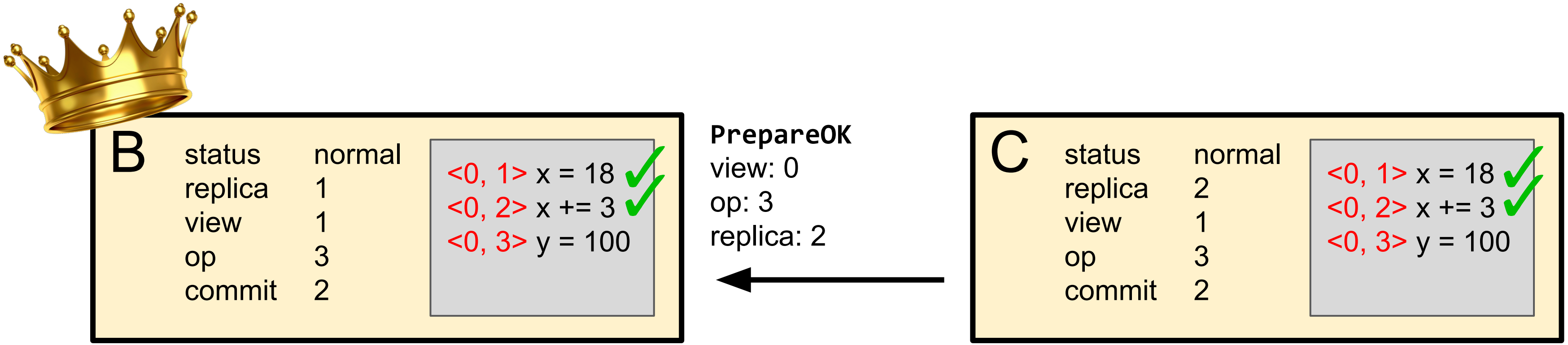
Notice <0, 3> is uncommitted and from an old view...

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

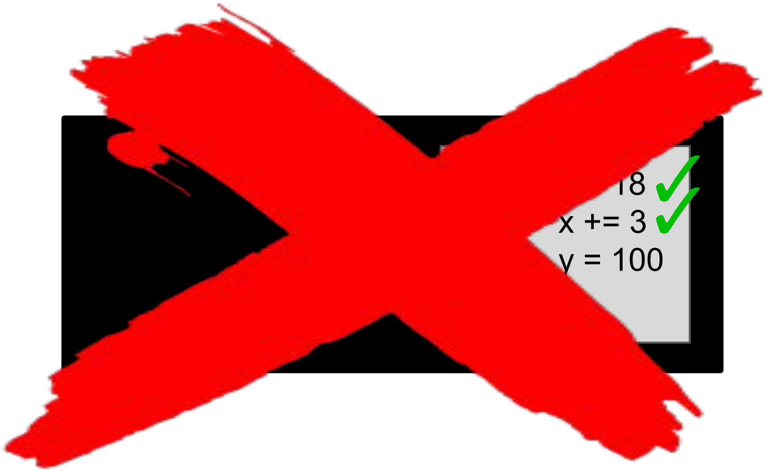
C

status normal
replica 2
view 1
op 3
commit 2

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

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 change in VR

New primary is pre-selected based on IP address (round-robin)

View change triggered by timeout, could be any node

Wait for *f* `SVC` that matches our view number before sending `DVC`

Wait for *f* `DVC` to start new view (primary)

- Why *f* in both cases?
- Provided that at most *f* servers fail, is *liveness* guaranteed?

Additional reading for viewstamped replication

http://pmg.csail.mit.edu/papers/vr-revisited.pdf

https://blog.acolyer.org/2015/03/06/viewstamped-replication-revisited/

Q & A