locker: distributed locking

Knut Nesheim
@knutin

The need

Real-time multiplayer game at Wooga

Stateful

One process per user

One process per world

The need

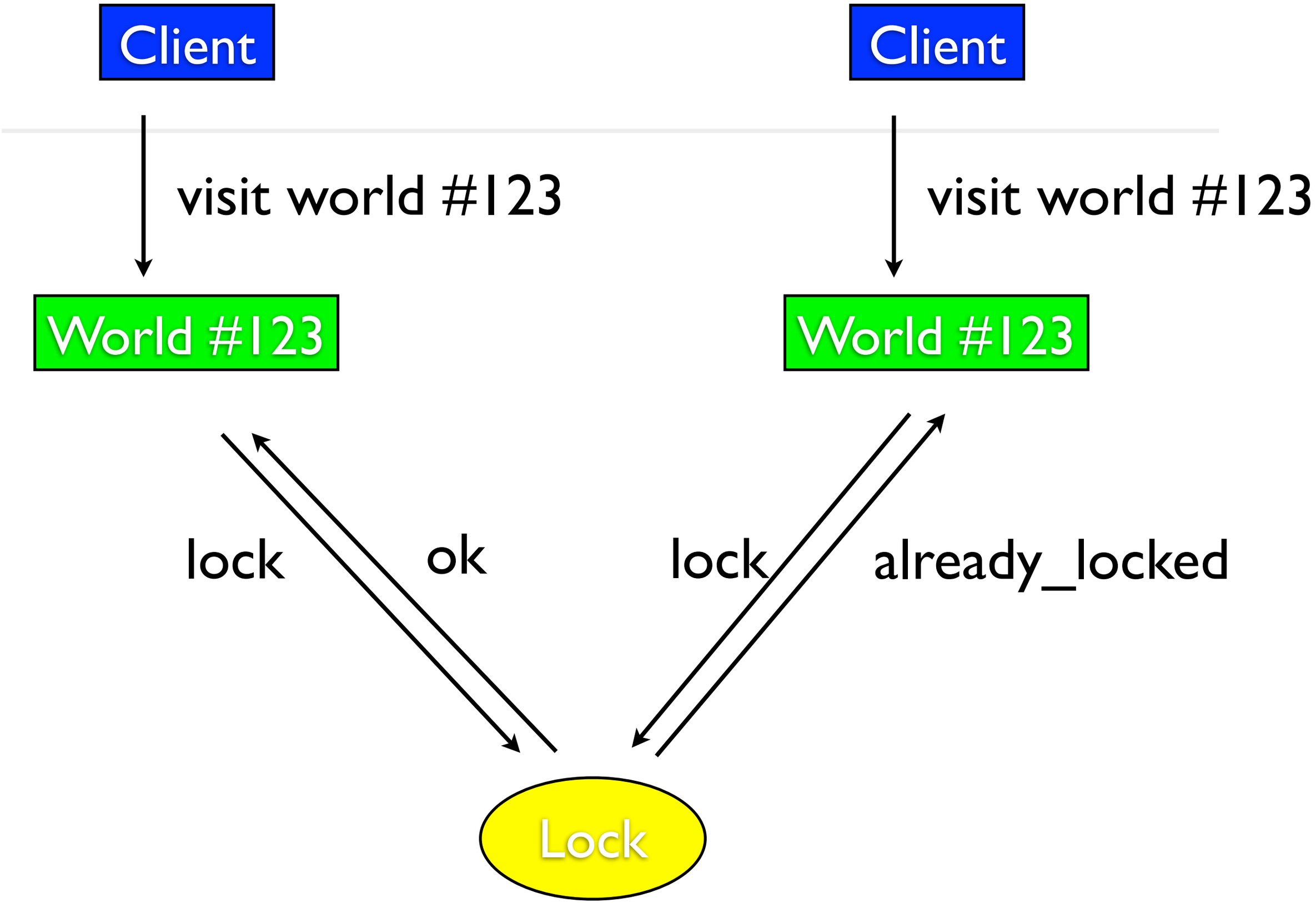
Only one process per user & world

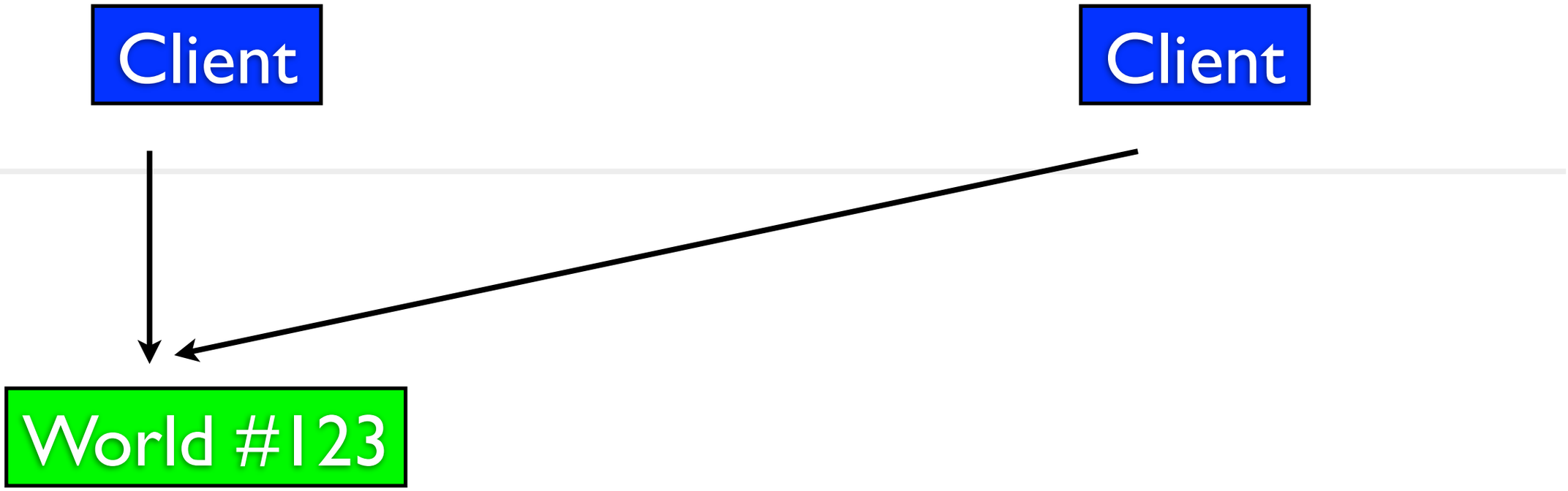
Cannot reconcile game worlds

Strong consistency

Fine-grained distributed coordination

Lookup table when sending game updates


Client
visit world #123
World #123
lock
ok
Client
visit world #123
World #123
lock
already_locked
Lock


Client
Client
World #123

Next generation

Lock implemented once already

Central serialization with atomic ops (Redis)

SPoF

Next gen want higher availability, because..

Living with failure

Hardware sometimes fails

The network is mostly ok

Software is almost bug-free

Downtime is often short

Living with a SPoF sucks..

Development easy

Operations suck

Change become scary

Operator error becomes costly

Must fix problems immediately

Requirements

~10k conditional writes per second

~3M eventually consistent reads per second

~150k reads on local replica

Lock expires if not kept alive

Dynamic cluster membership

ZooKeeper looks like the best option

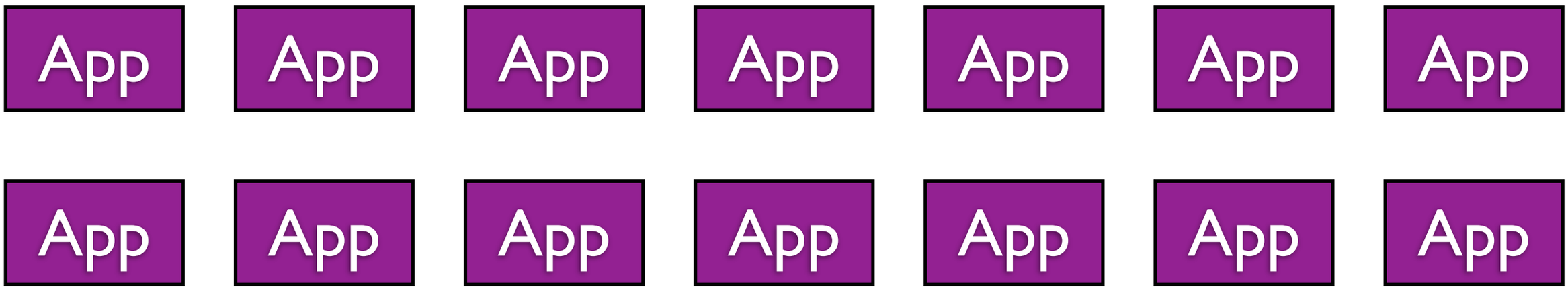
“What would the dream solution look like?”

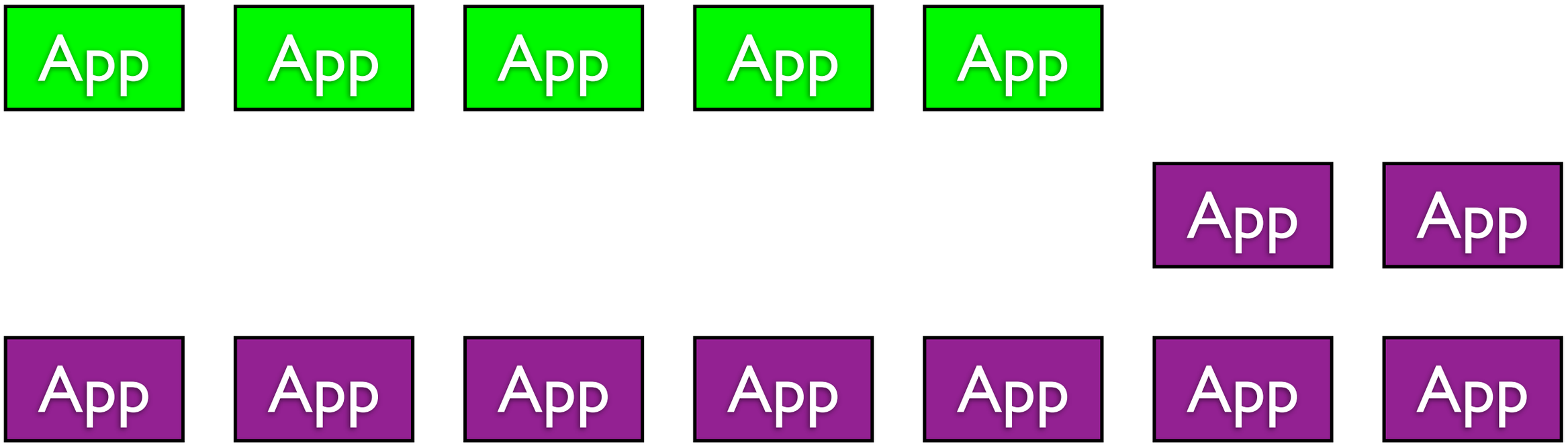
LB LB

App App App App App App App

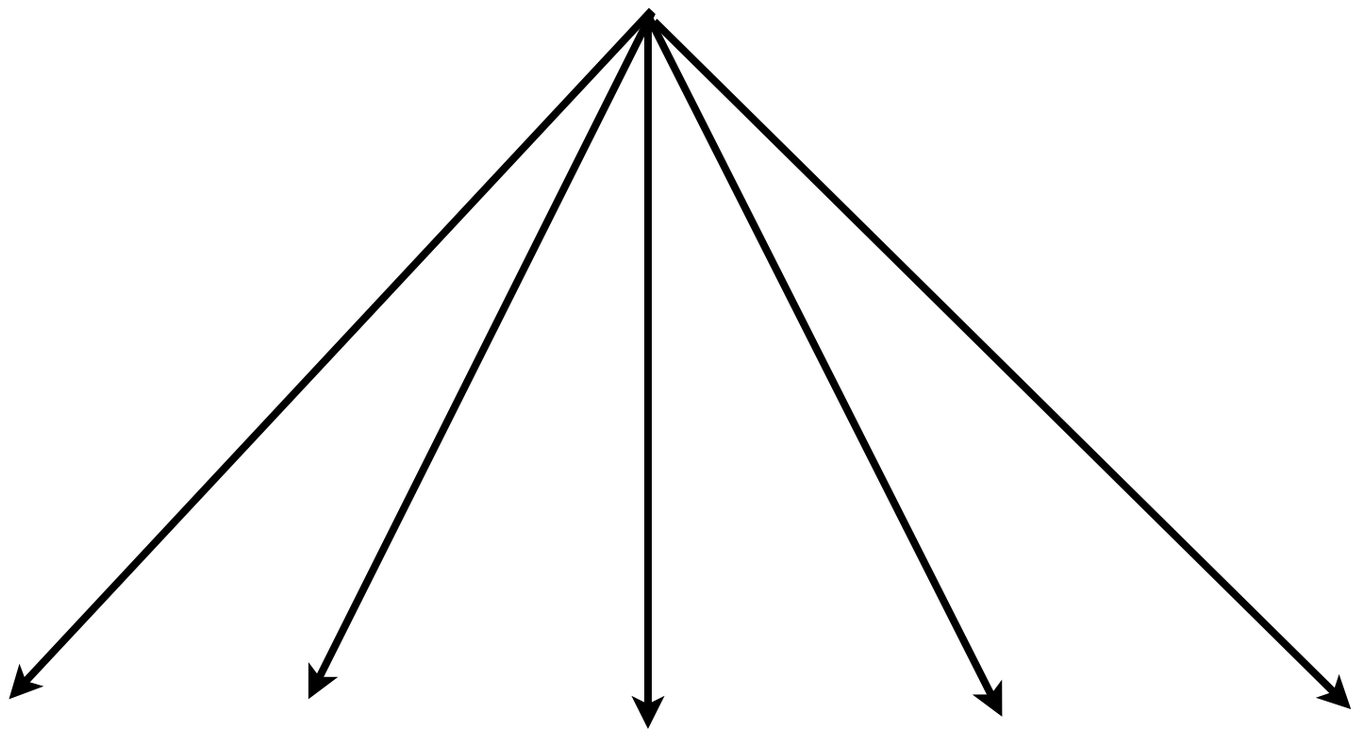
App App App App App App App

S3 DB


App
App
App
App
App
App
App
App
App
App
App
App
App
App


App
App
App
App
App
App
App
App
App
App
App
App
App
App
App

N=5, W=3

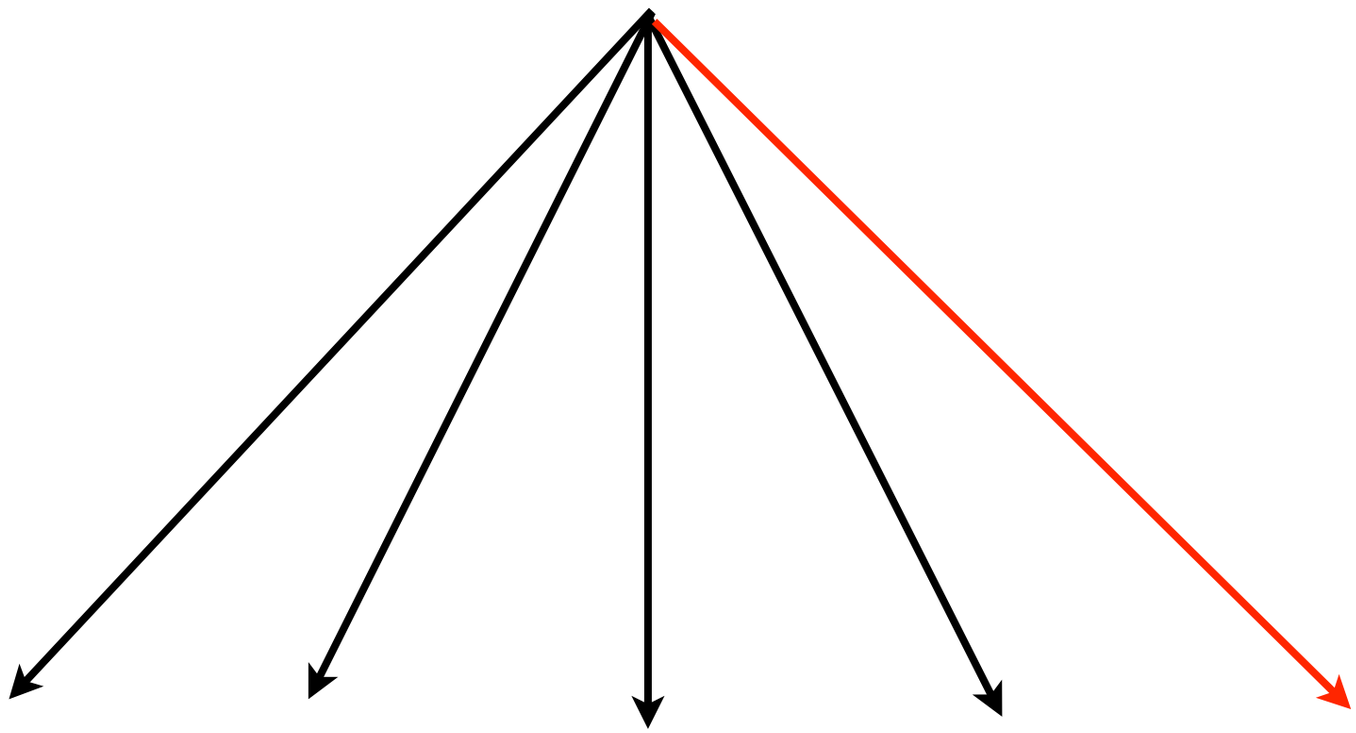


App App App App App

App App

App App App App App App App

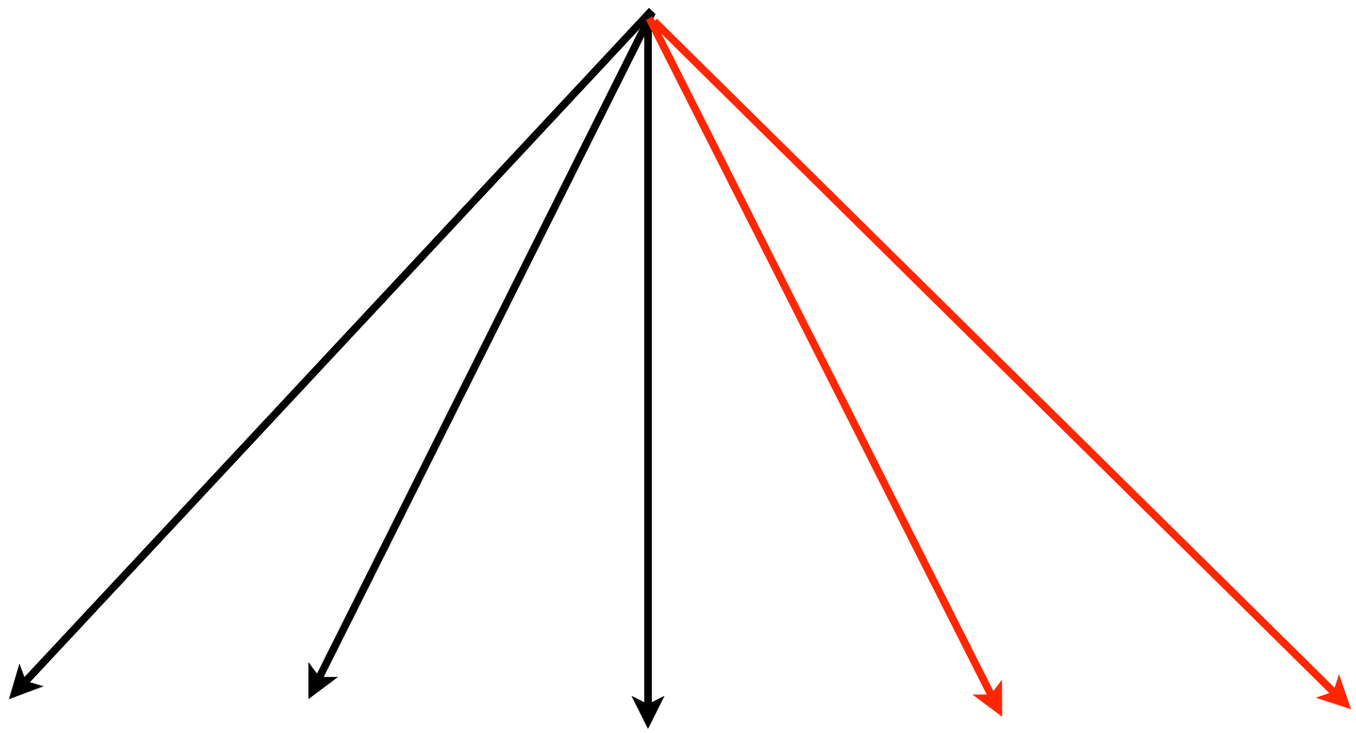
N=5, W=3



App App App App App

App App App App App App App App App

N=5, W=3

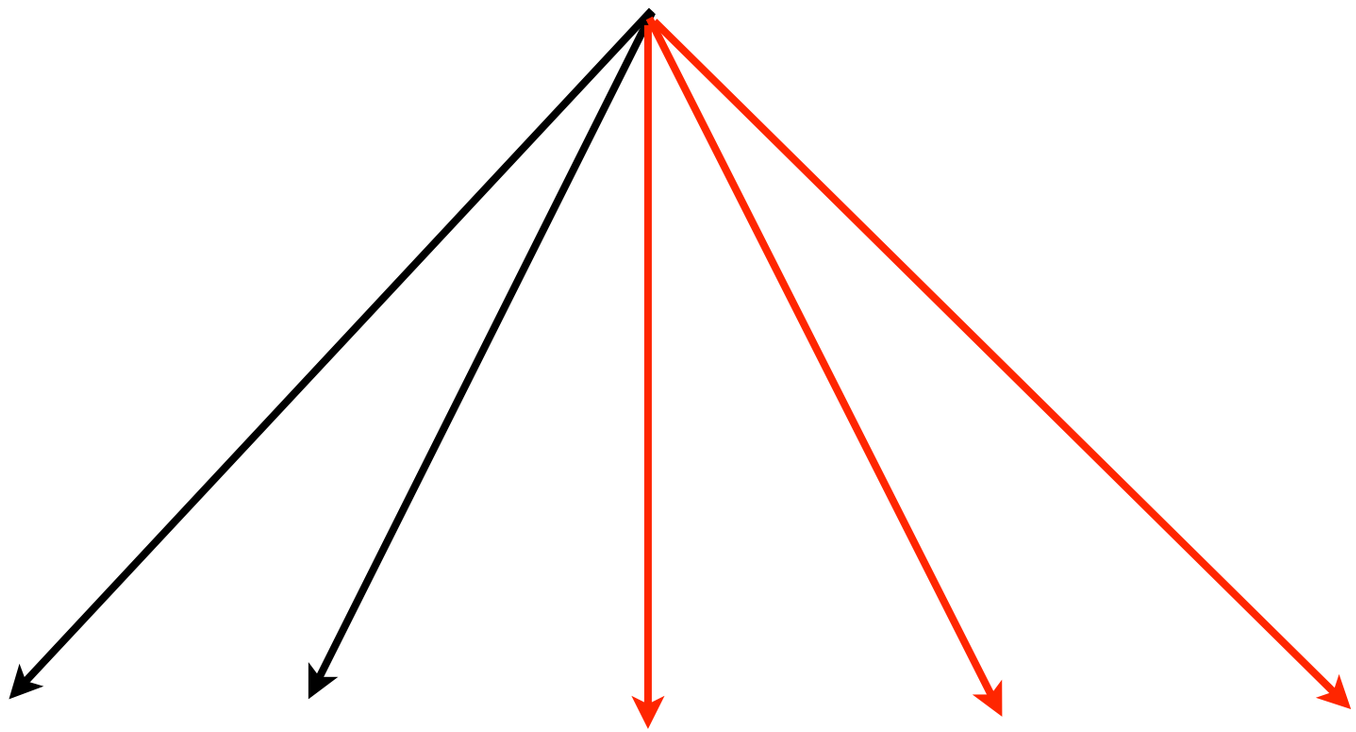


App App App App App

App App

App App App App App App App

N=5, W=3

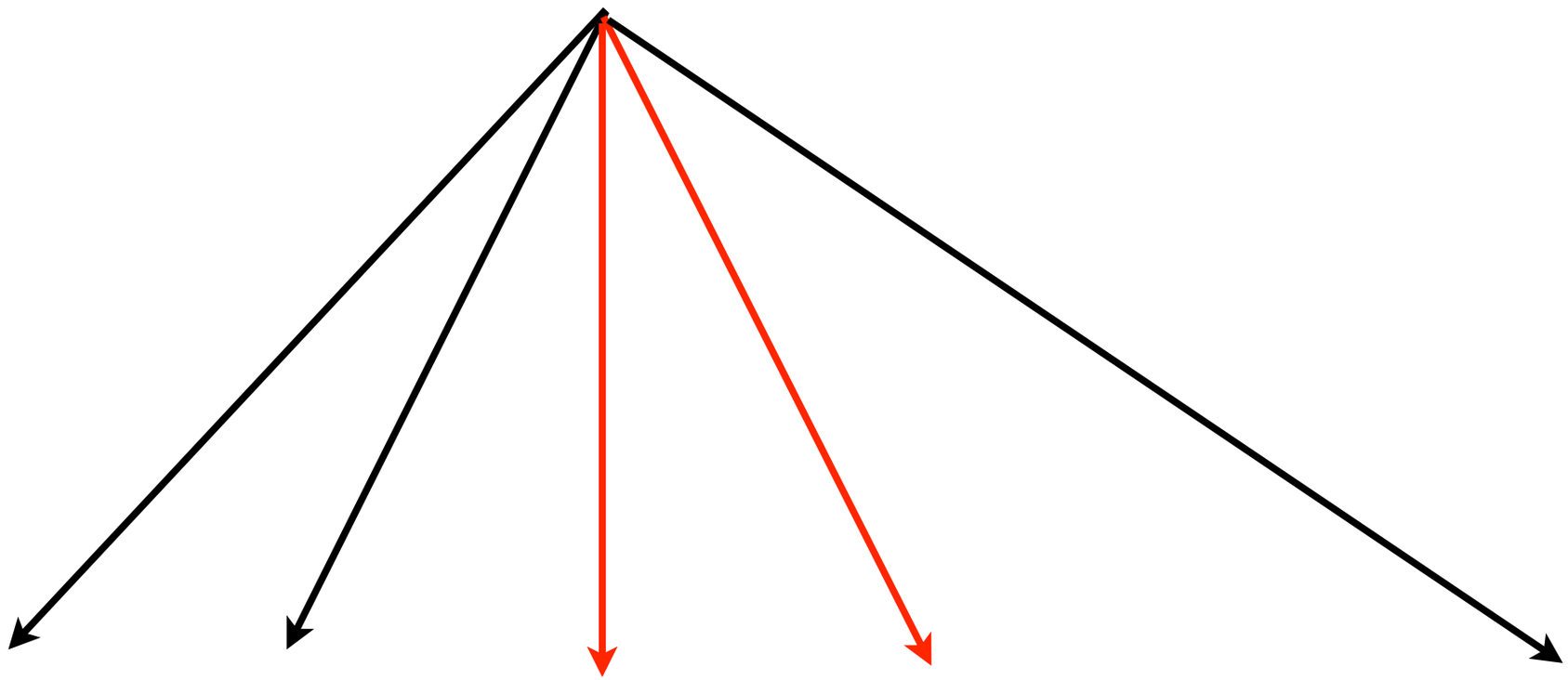


App App App App App

App App App App App App App

App App

N=5, W=3



App App App App App App App App App App App App App

Run inside our app servers

Easy to debug, instrument, monitor, deploy

Simple operations

“How hard can it be?”

Distributed systems are hard

Many books, papers on distributed systems

Riak a good example of mindset

Idea: pick the algorithms that fits best

Simplest thing that could possibly work

“good enough”

Problem	**Solution**
Consistency, conditional writes	Serialization, “2 Phase Commit”
Availability	Multiple serializers, quorum (CP)
Local replica	Replay transaction log
Dynamic cluster	Manual configuration
Anti-entropy	Lock keep alive

Implementation

Proof of concept in Friday afternoon

Looked promising, spent ~3 weeks

Turned into production quality

Test race conditions

PropEr

330 lines (!)

locker

http://github.com/wooga/locker

Beware tradeoffs!

Keep consistency, sacrifice availability during failure

No persistency, assumes enough masters stays up

No group membership, views

Manually configure masters, replicas, quorum value

Assumes perfect network during reconfiguration

Not based on a described algorithm

Usage

```
start_link(W)

set_nodes(AllNodes, Masters, Replicas)

lock(Key, Value, LeaseLength)

extend_lease(Key, Value, LeaseLength)

release(Key, Value)

dirty_read(Key)
```

`lock(Key, Value, LeaseLength)`

Two phases:

Prepare: Ask masters for votes

Commit: If majority, write on masters

Timeout counted as negative vote, CP

Asynchronous replication, `wait_for/2`

Client A

Client B

```
locker:lock(foo, pid(0,123,0))
```

Master #1

Master #2

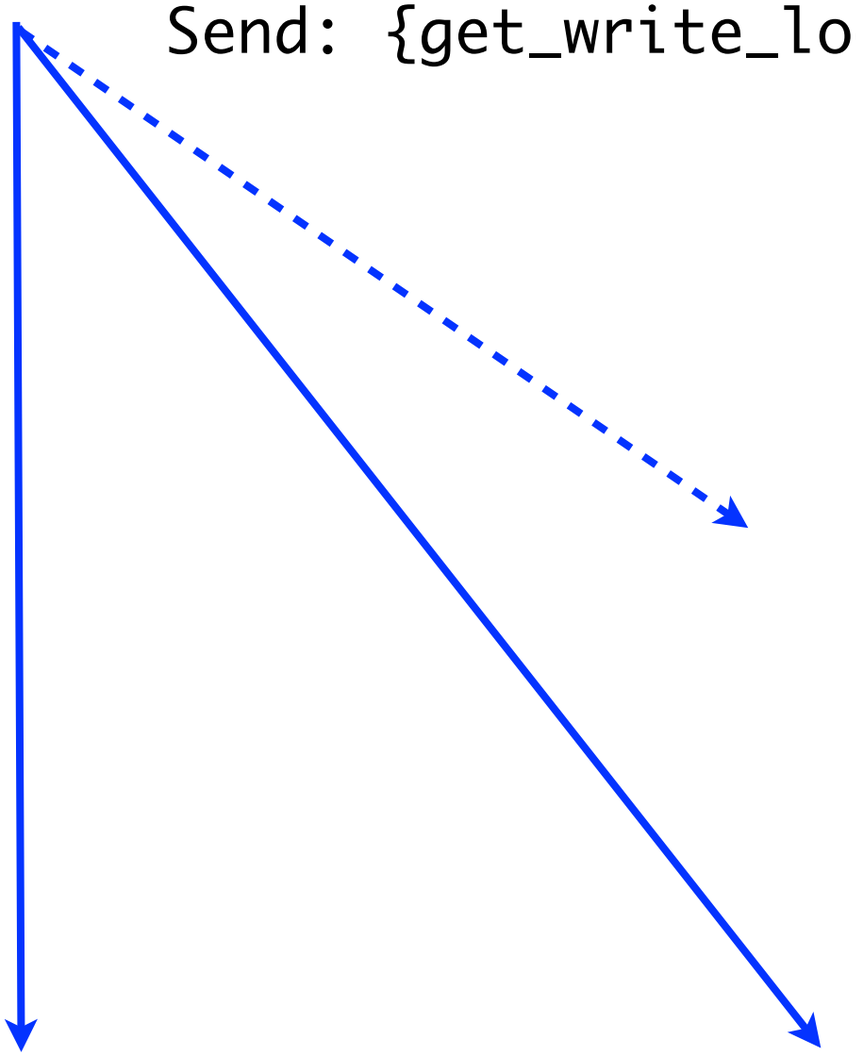
Master #3

Client A

`locker:lock(foo, pid(0,123,0))`

Client B

`Send: {get_write_lock, foo, not_found}`



Master #1

Master #2

Master #3

Write locks:[]

[]

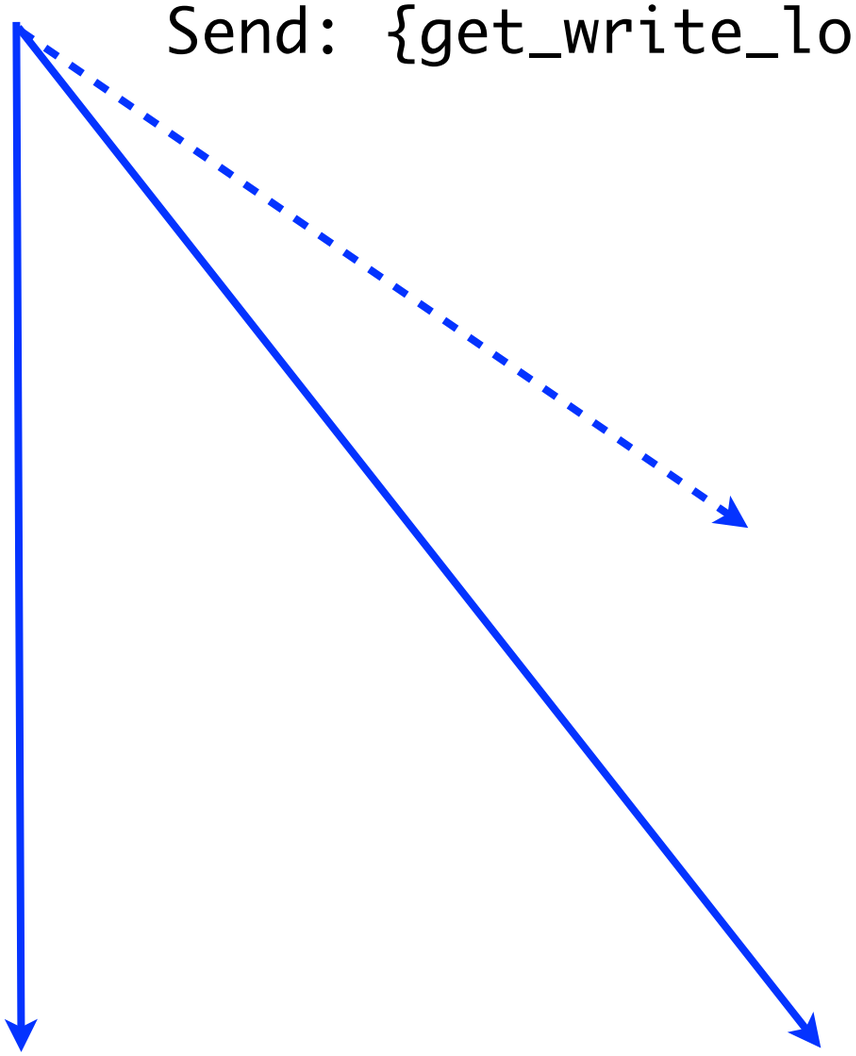
[]

Client A

`locker:lock(foo, pid(0,123,0))`

Client B

`Send: {get_write_lock, foo, not_found}`



Master #1

[{lock, foo}]

Master #2

[{lock, foo}]

Master #3

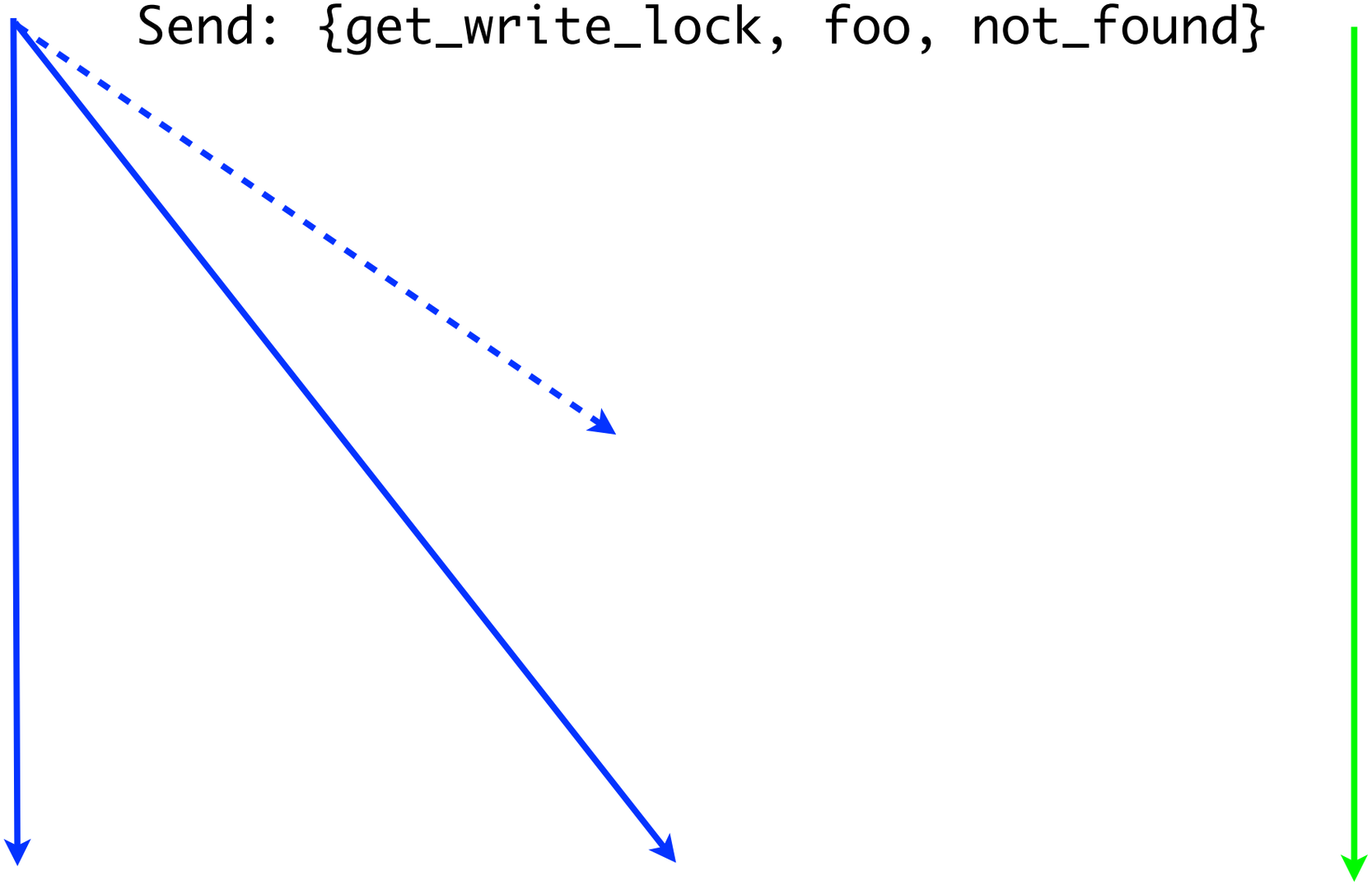
[]

Client A

`locker:lock(foo, pid(0,123,0))`

Client B

`Send: {get_write_lock, foo, not_found}`



Master #1

[{lock, foo}]

Master #2

[{lock, foo}]

Master #3

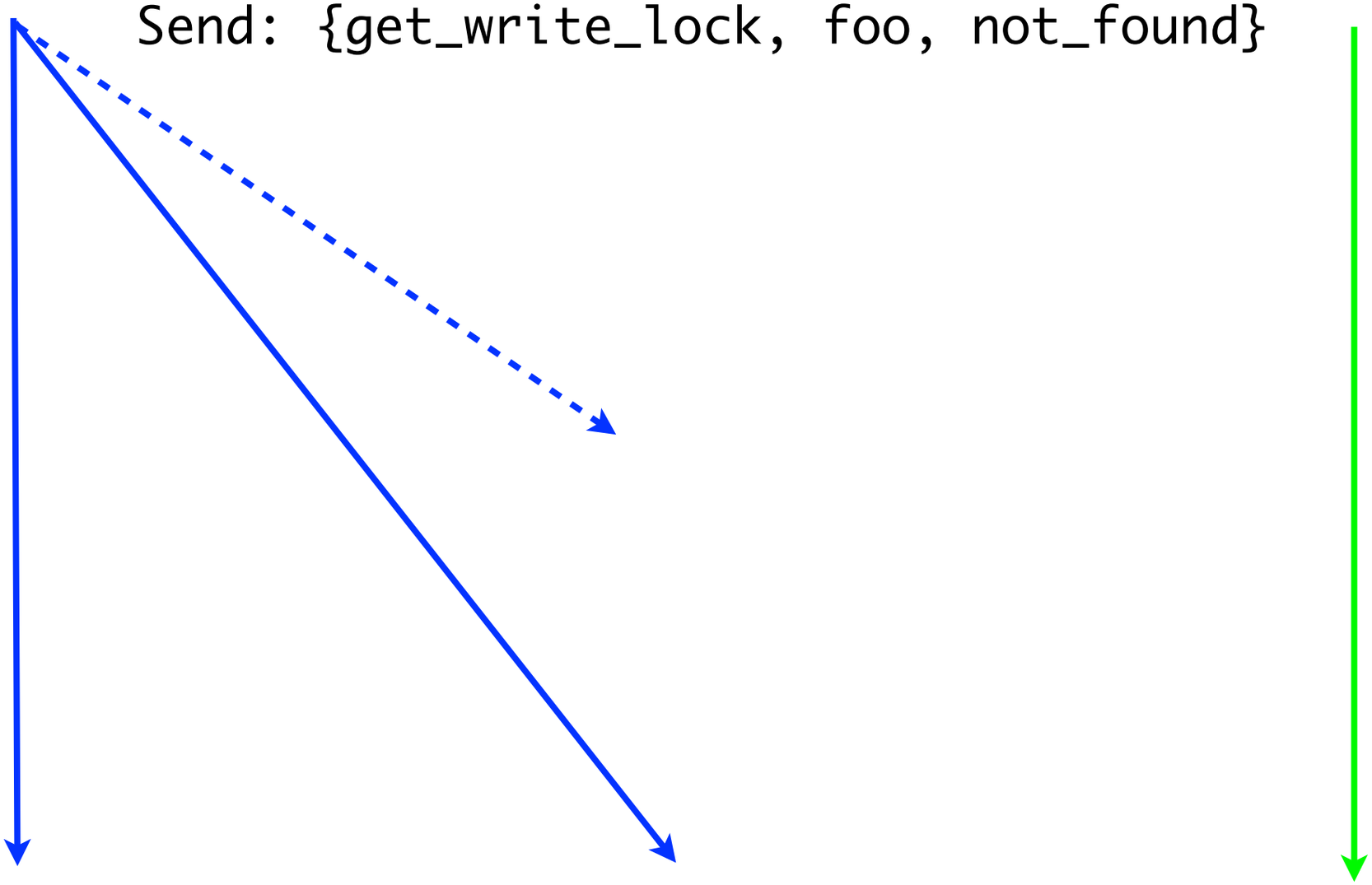
[]

Client A

`locker:lock(foo, pid(0,123,0))`

Client B

`Send: {get_write_lock, foo, not_found}`



Master #1

Master #2

Master #3

[{lock, foo}]

[{lock, foo}]

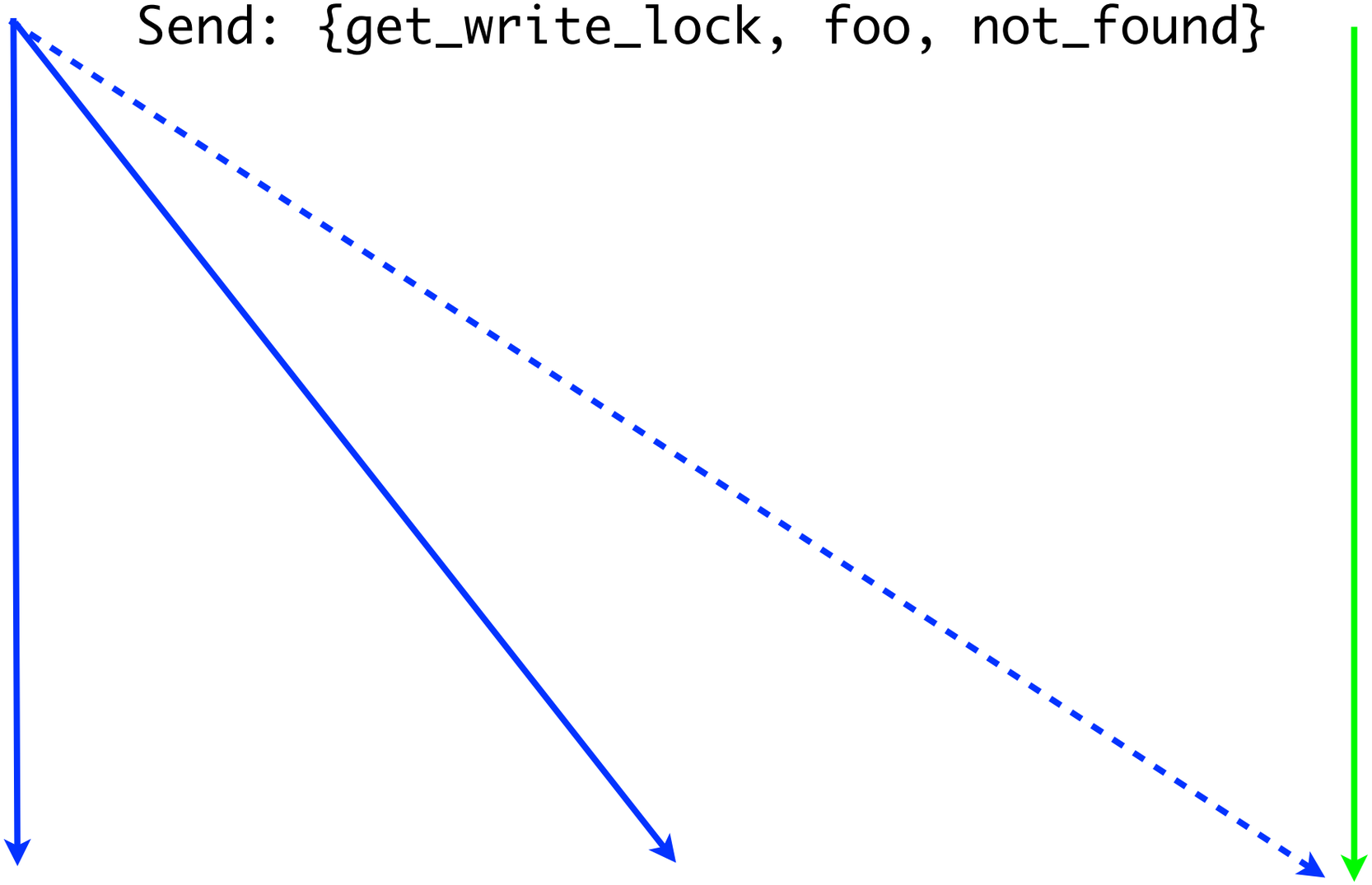
[{lock, foo}]

Client A

`locker:lock(foo, pid(0,123,0))`

Client B

`Send: {get_write_lock, foo, not_found}`



Already locked!

Master #1

Master #2

Master #3

[{lock, foo}]

[{lock, foo}]

[{lock, foo}]

Client A

`locker:lock(foo, pid(0,123,0))`

Client B

`Send: {get_write_lock, foo, not_found}`

`[ok, ok, error]`

`[error,error, ok]`

Master #1

Master #2

Master #3

[{lock, foo}]

[{lock, foo}]

[{lock, foo}]

Client A

`locker:lock(foo, pid(0,123,0))`

Client B

`[ok, ok, error]`

`[error, error, ok]`

Master #1

Master #2

Master #3

[{lock, foo}]

[{lock, foo}]

[{lock, foo}]

Client A

`locker:lock(foo, pid(0,123,0))`

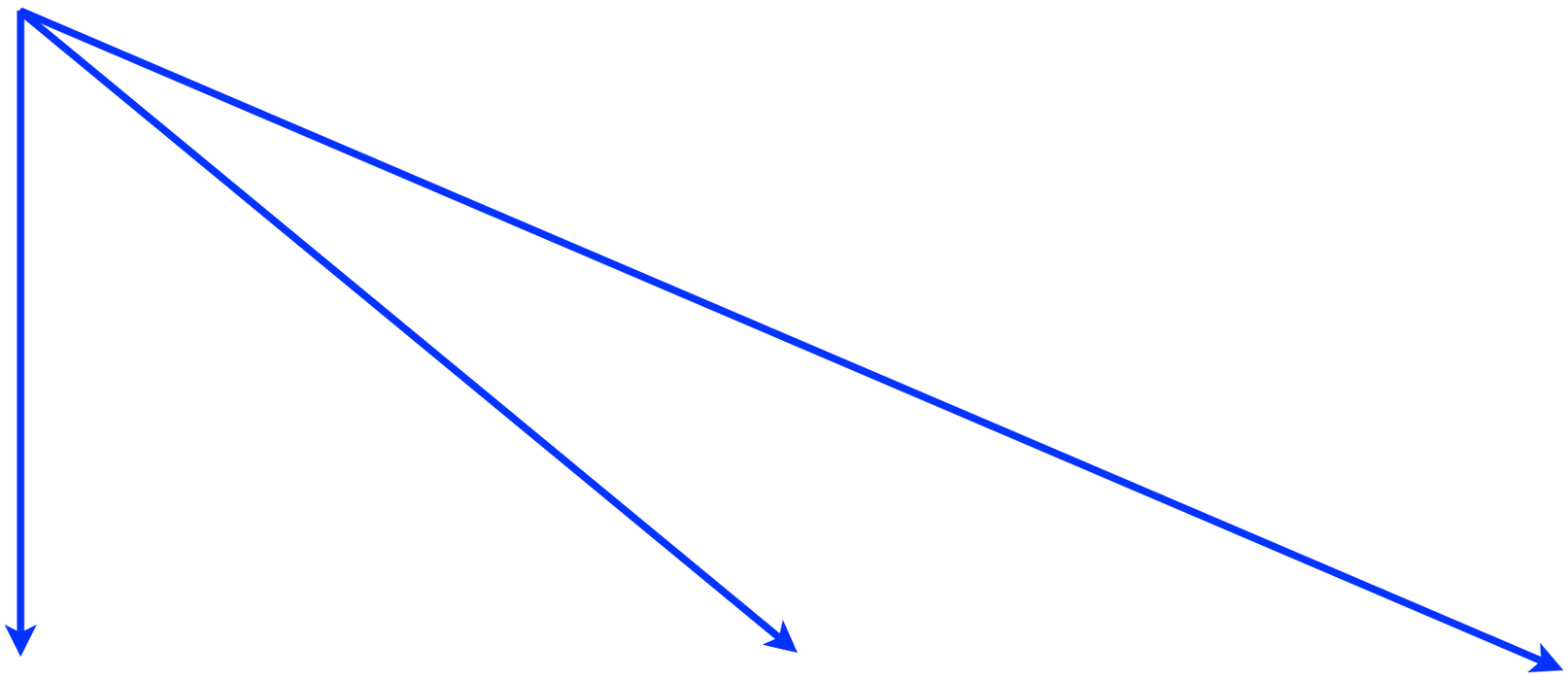
Client B

`[ok, ok, error]`

`Send: {write, foo, 123}`

`[error, error, ok]`

`Send: release_write_lock`



Master #1

Master #2

Master #3

[{lock, foo}]

[{lock, foo}]

[{lock, foo}]

Client A

`locker:lock(foo, pid(0,123,0))`

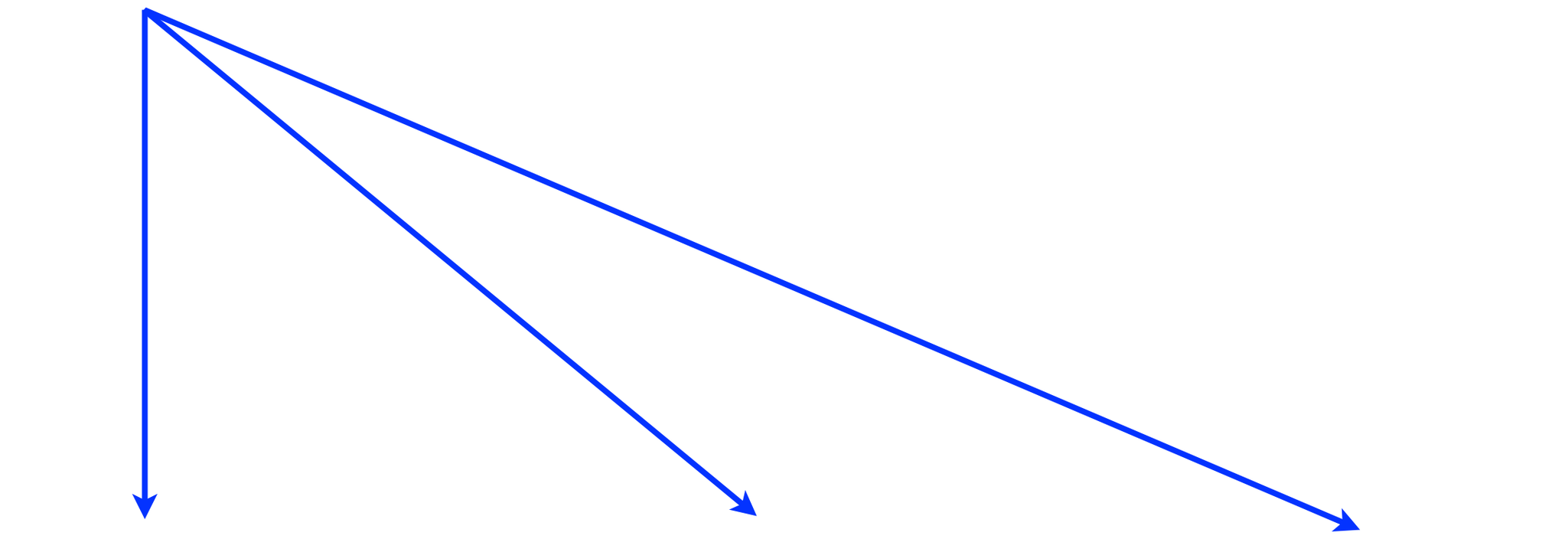
Client B

`[ok, ok, error]`

`Send: {write, foo, 123}`

`[error, error, ok]`

`Send: release_write_lock`



Master #1

Master #2

Master #3

`[]`

`[]`

`[]`

Use locker when

Strong consistency is sometimes needed

Protect resources

Leader election

Service discovery

Is it any good?

Some experts say yes,
some experts say no

Conclusion

Distributed systems are hard

We could maybe have made ZooKeeper work..

..but we now have our dream system

Ambitious, naive project led to big operational advantage

Q&A

http://github.com/wooga/locker

@knutin