PBS @ Twitter 6.22.12

Peter Bailis @pbailis
Shivaram Venkataraman,
Mike Franklin,
Joe Hellerstein,
Ion Stoica

UC Berkeley

PBS

Probabilistically
Bounded
Staleness

1. Fast
2. Scalable
3. Available

solution:

replicate for

1. request capacity
2. reliability

solution:

replicate for

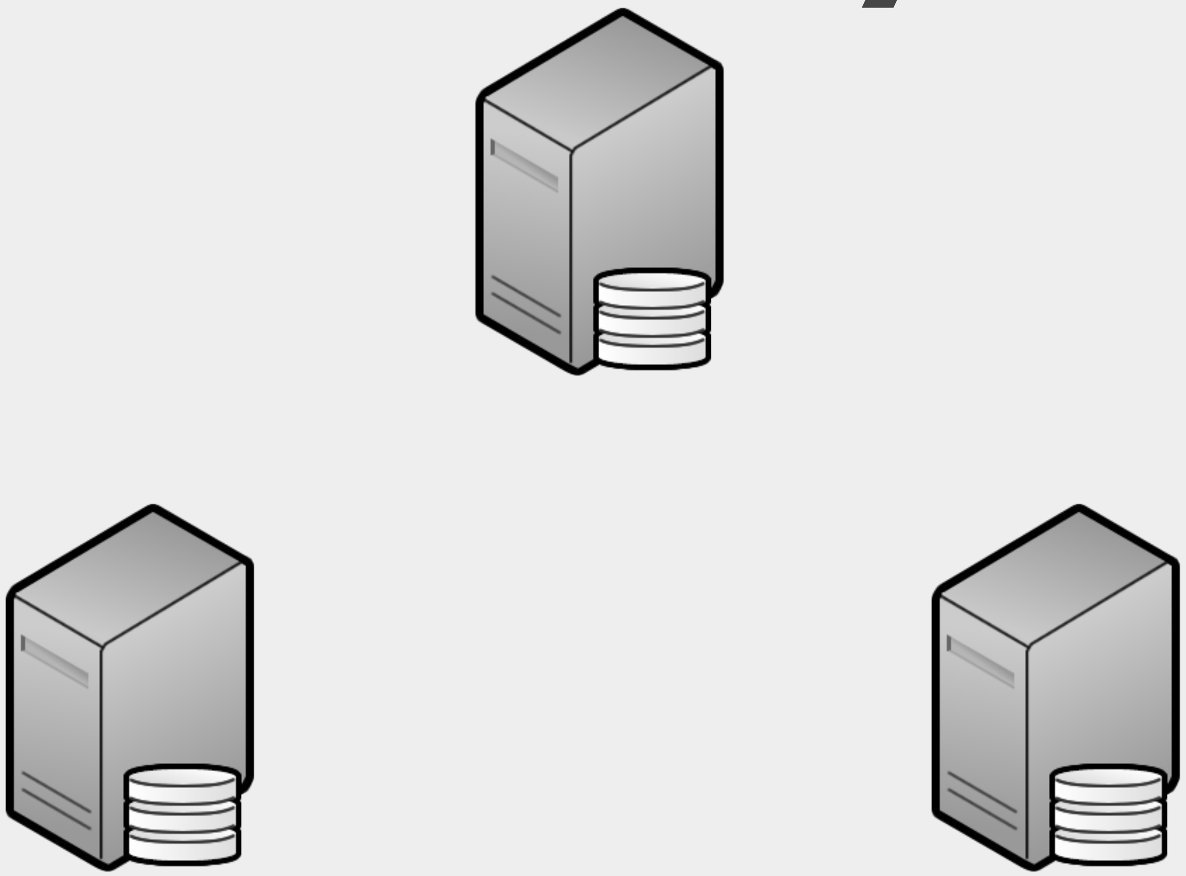
1. request capacity
2. reliability

solution:

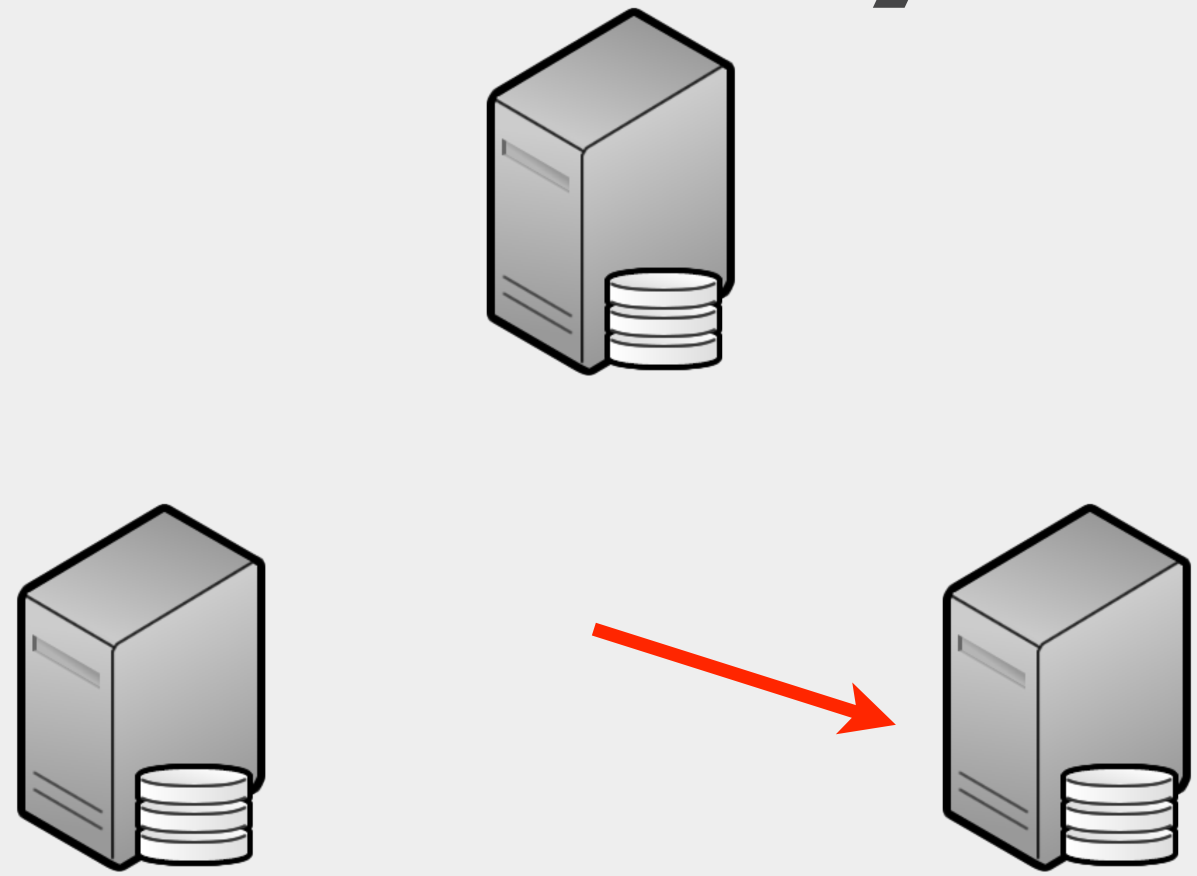
replicate for

1. request capacity
2. reliability

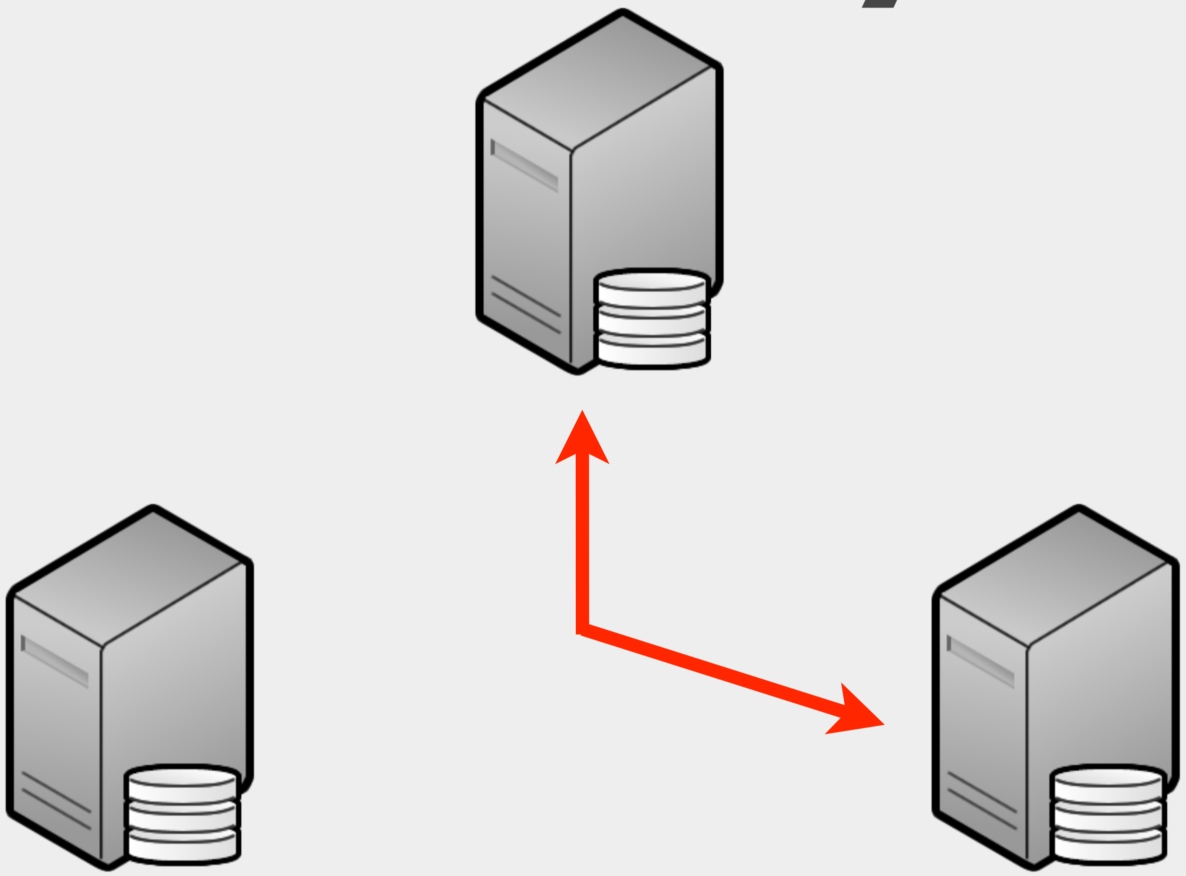
keep replicas in sync



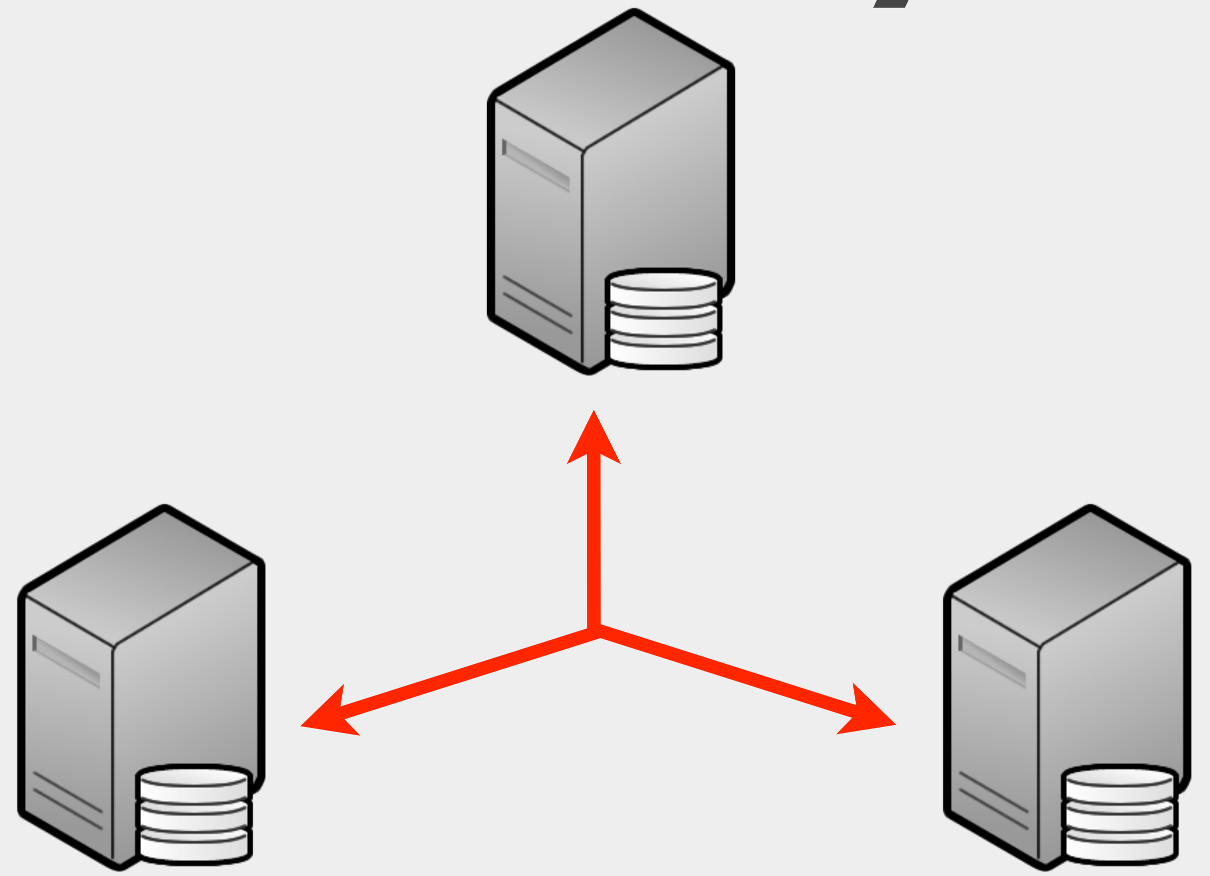
keep replicas in sync



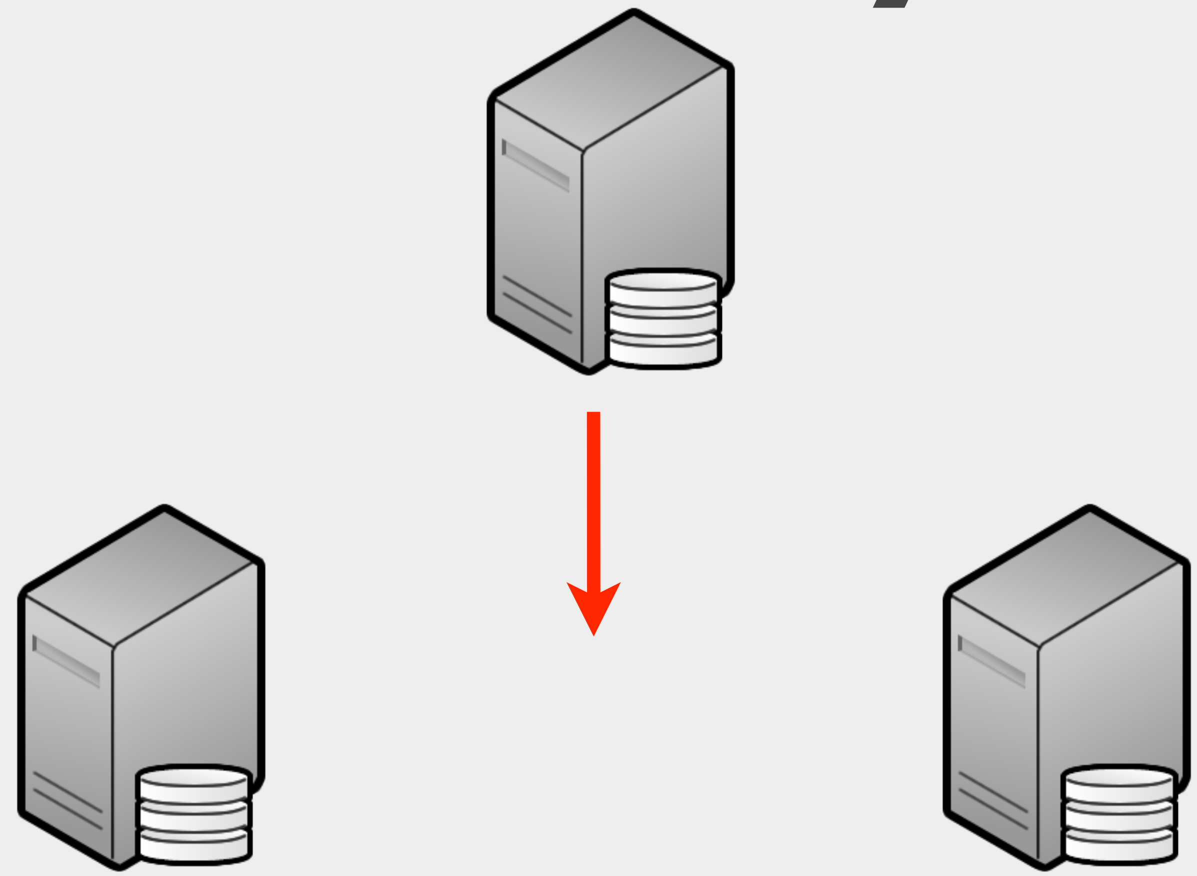
keep replicas in sync



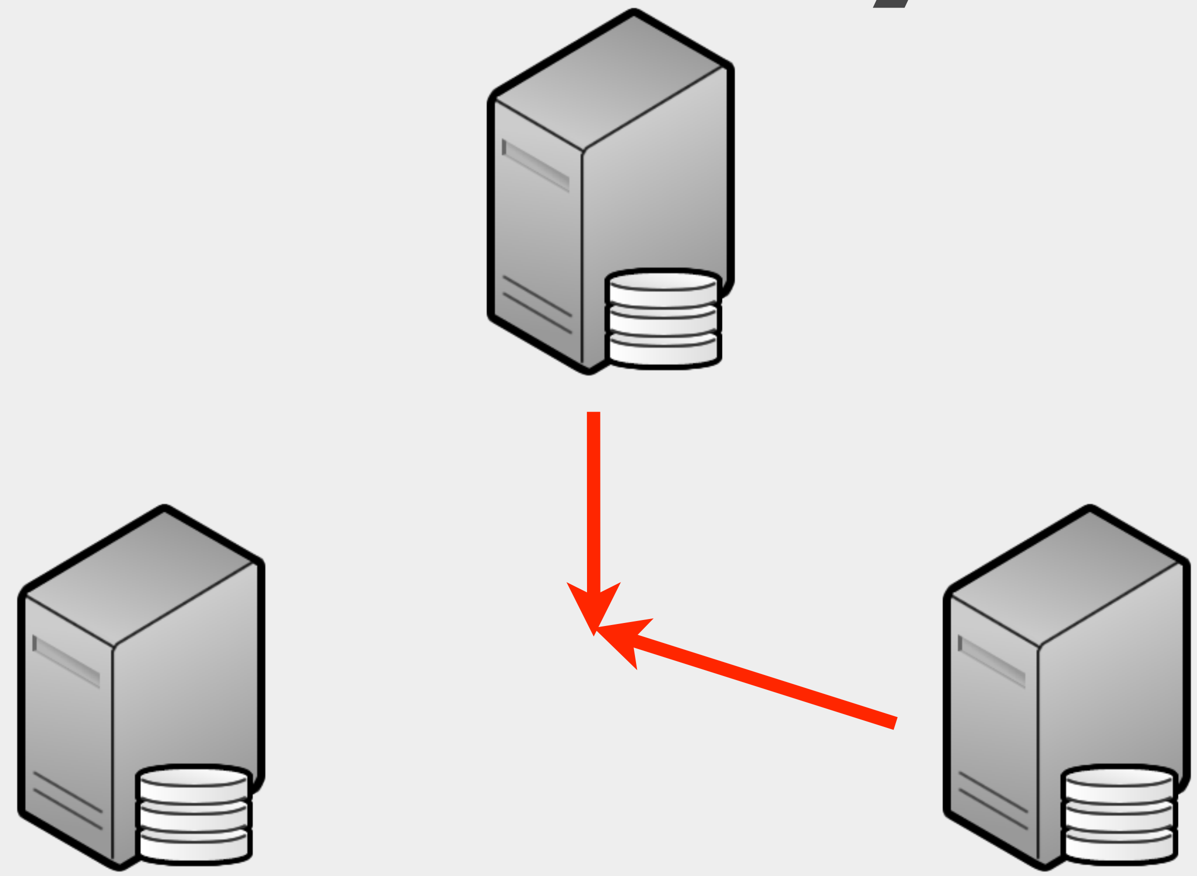
keep replicas in sync



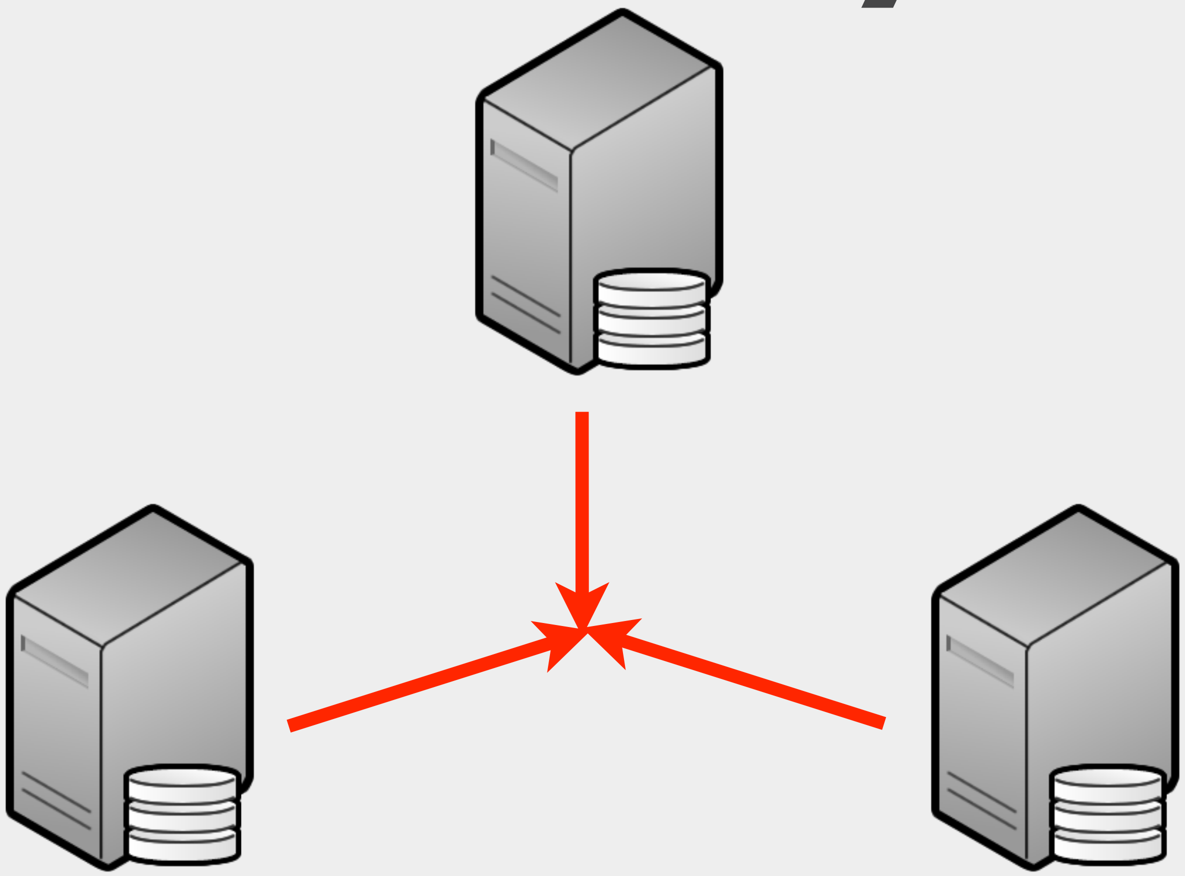
keep replicas in sync



keep replicas in sync

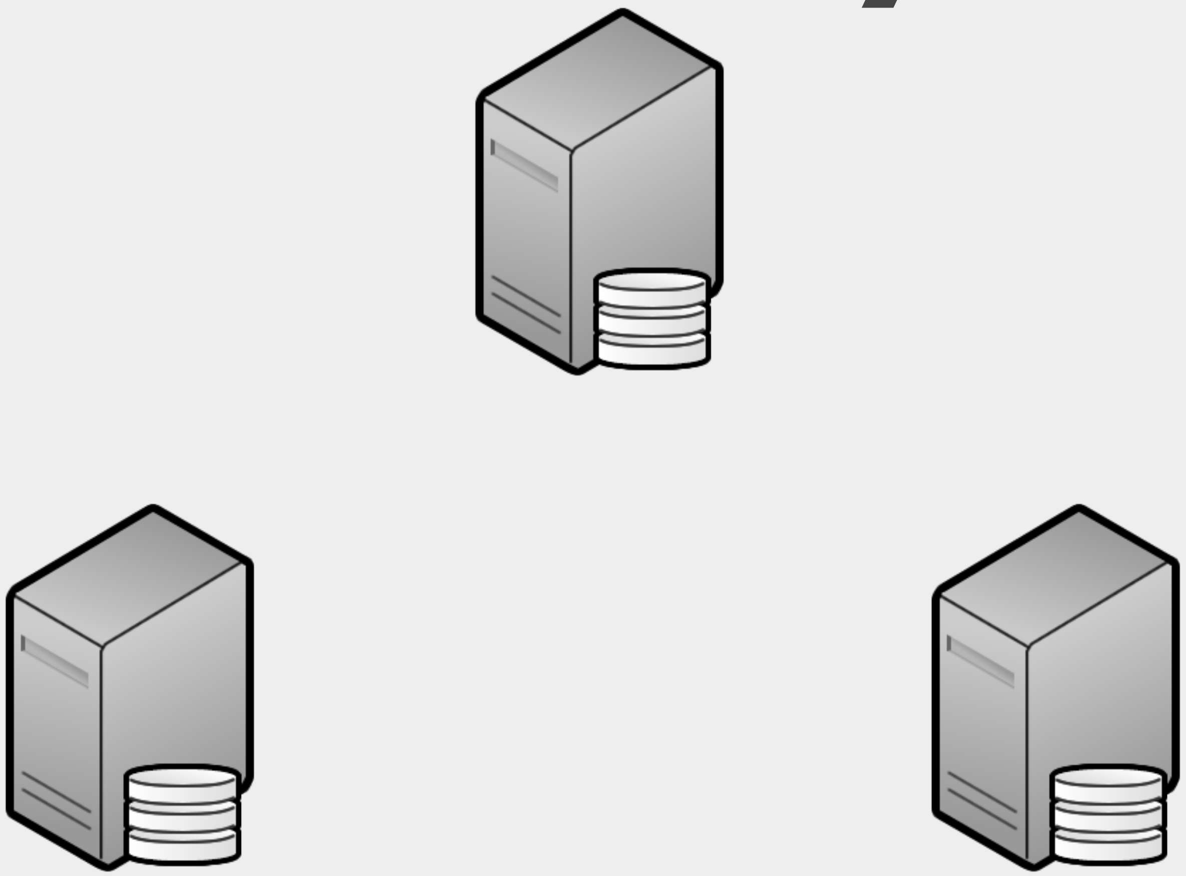


keep replicas in sync



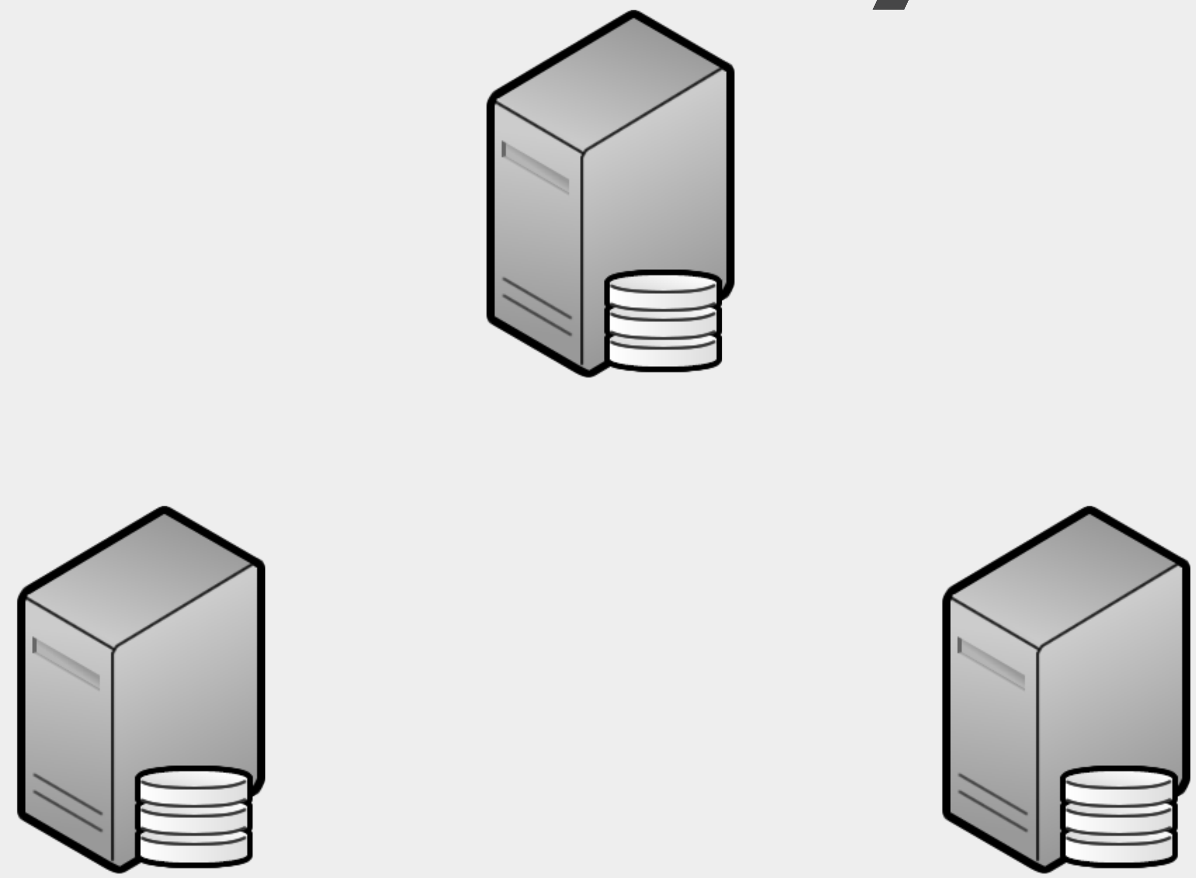
keep replicas in sync

slow



keep replicas in sync

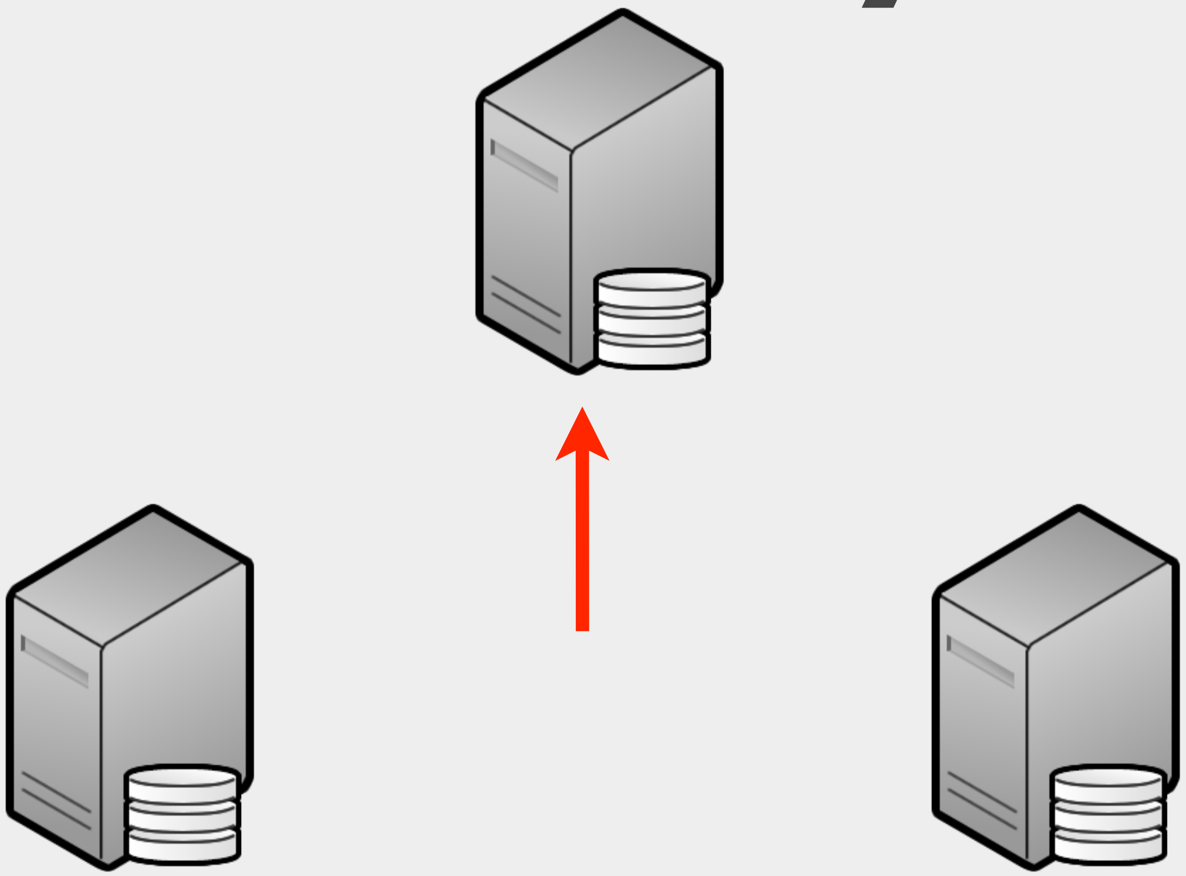
slow



alternative: sync later

keep replicas in sync

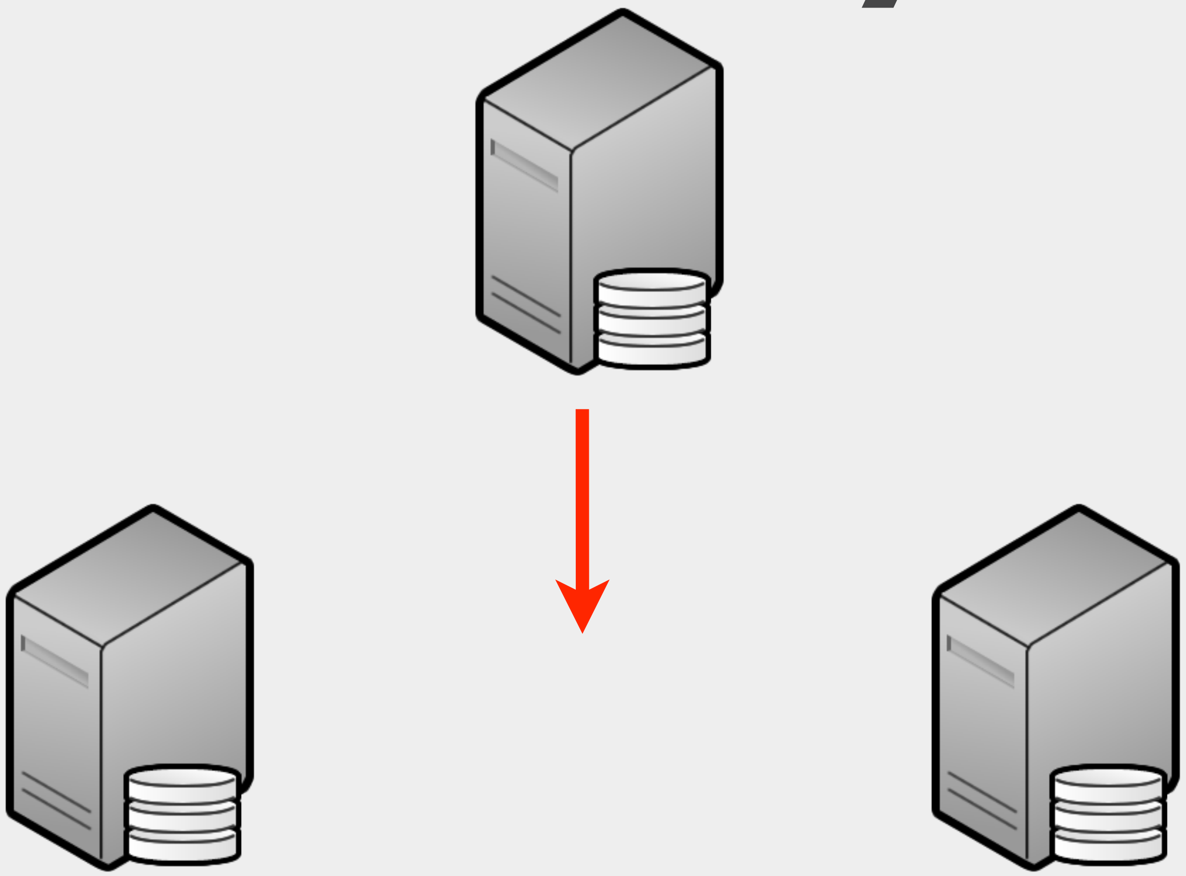
slow



alternative: sync later

keep replicas in sync

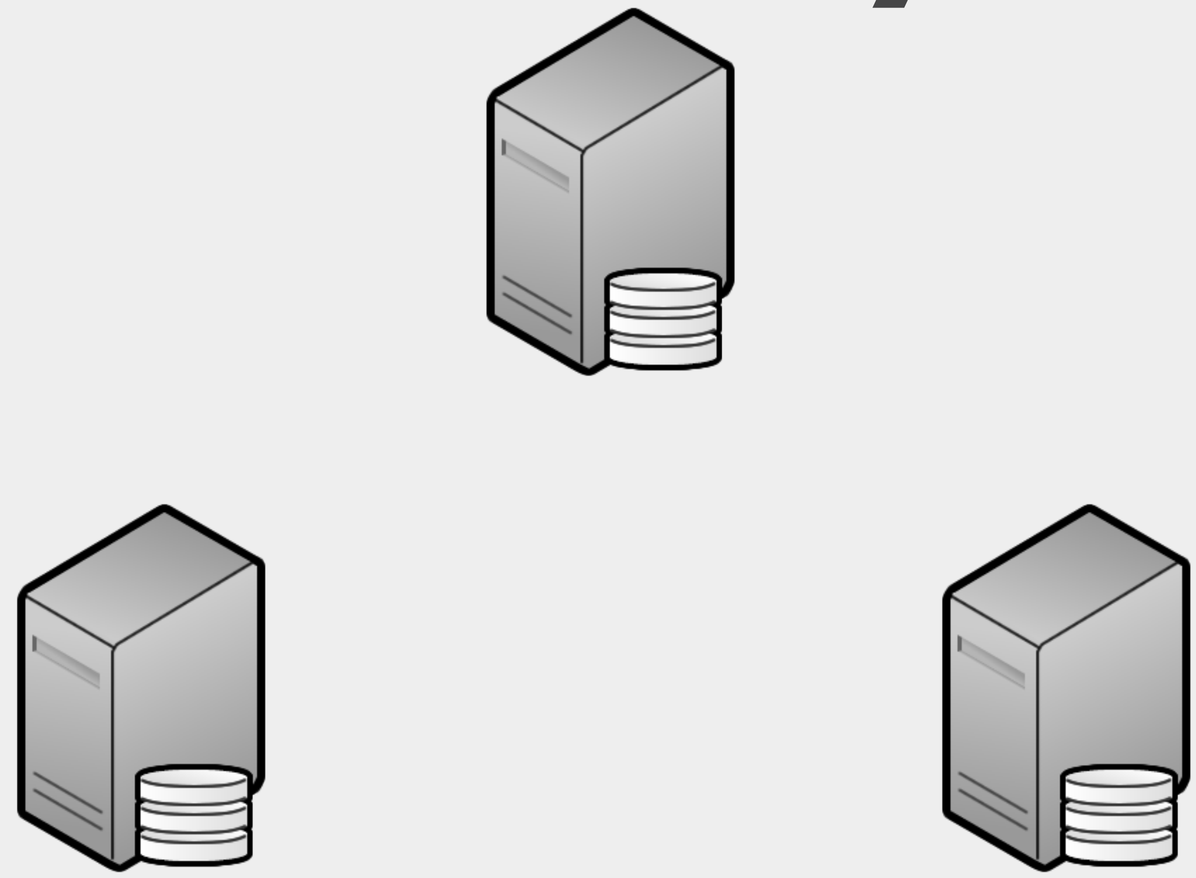
slow



alternative: sync later

keep replicas in sync

slow

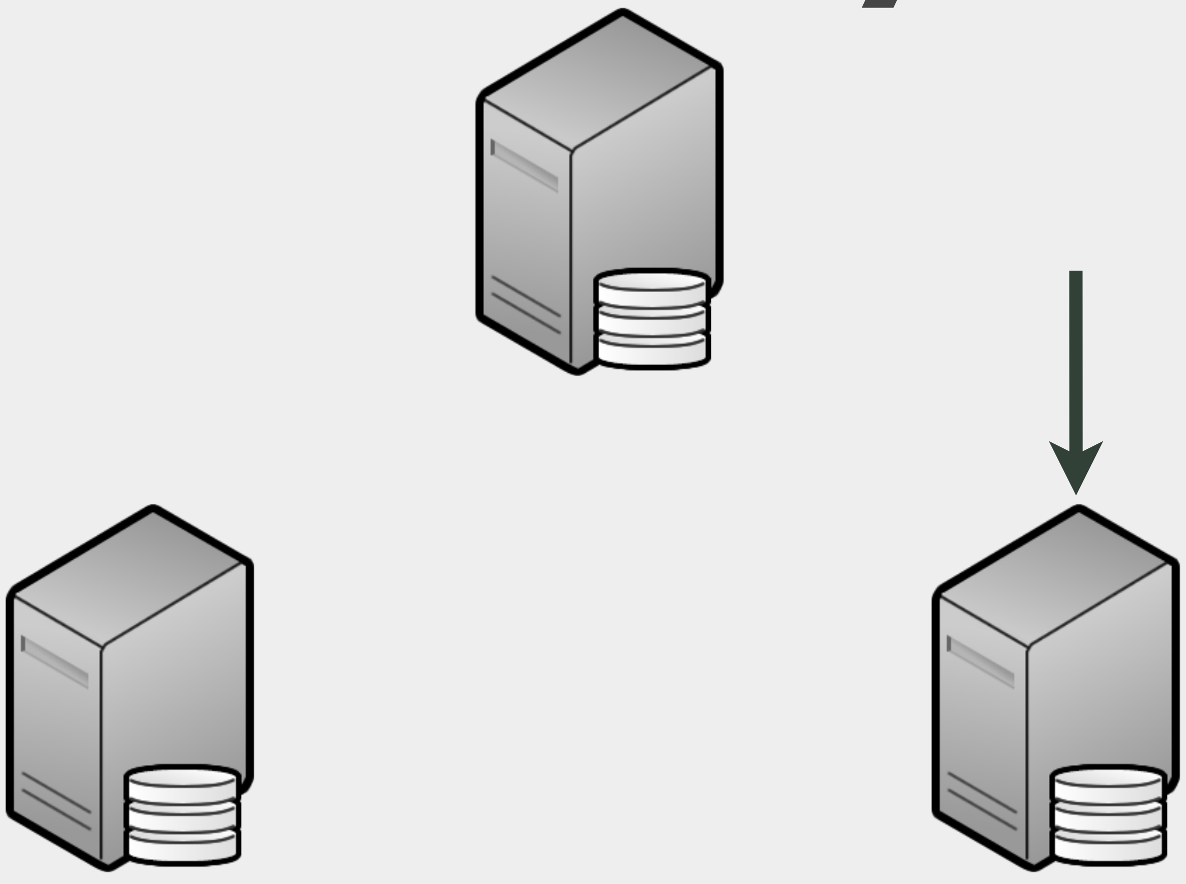


alternative: sync later

inconsistent

keep replicas in sync

slow

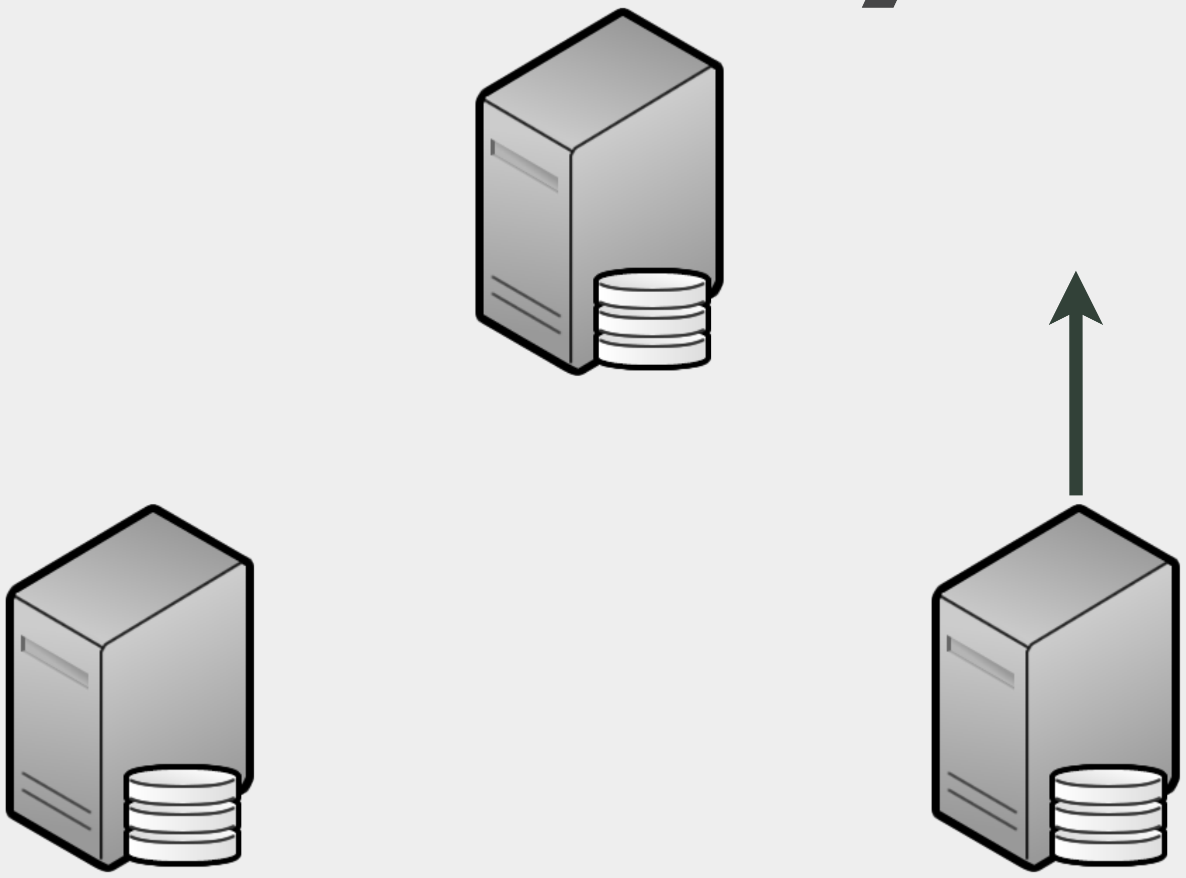


alternative: sync later

inconsistent

keep replicas in sync

slow

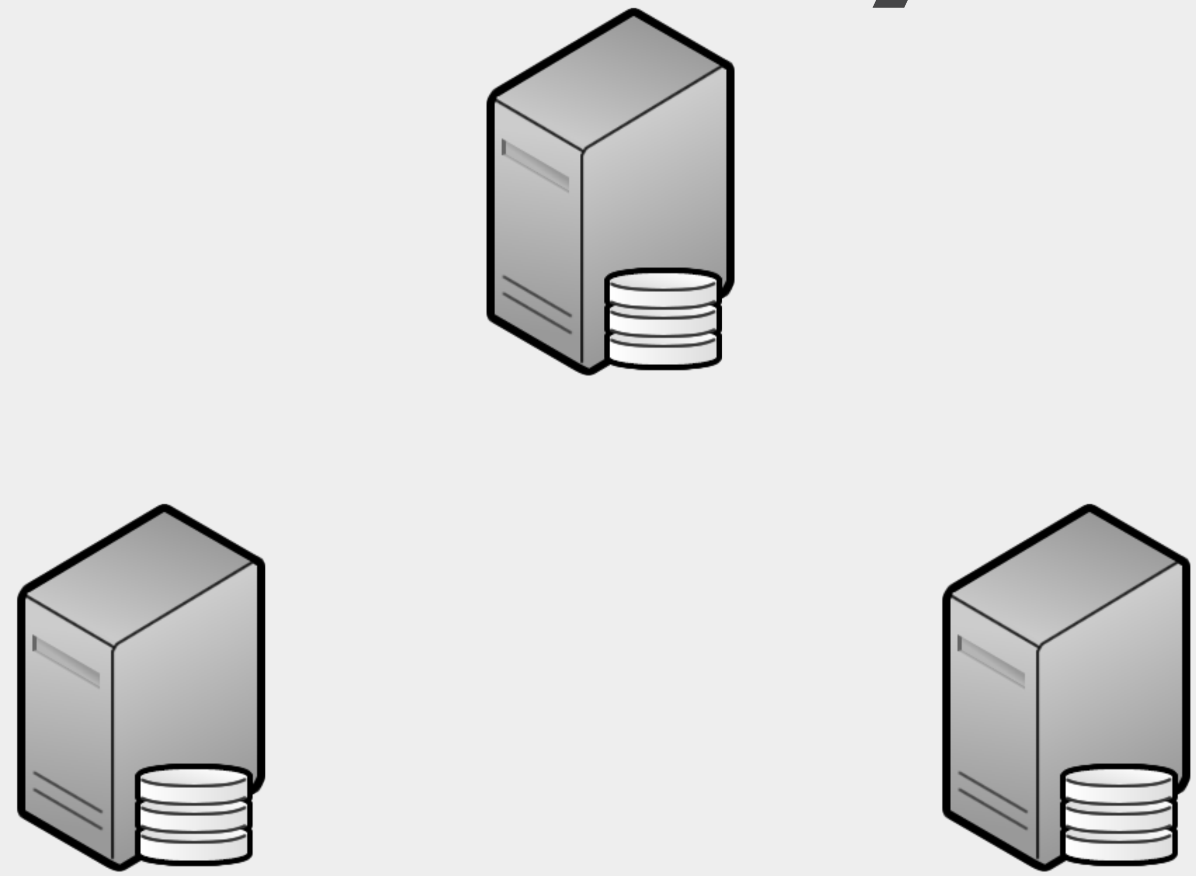


alternative: sync later

inconsistent

keep replicas in sync

slow



alternative: sync later

inconsistent

⇧ consistency, ⇧ latency

contact more replicas,
read more recent data

⇩ consistency, ⇩ latency

contact fewer replicas,
read less recent data

⇧ consistency, ⇧ latency

contact more replicas,
read more recent data

⇩ consistency, ⇩ latency

contact fewer replicas,
read less recent data

eventual consistency

"if no new updates are made to the object, eventually all accesses will return the last updated value"

W. Vogels, *CACM* 2008

How eventual?

How long do I have to wait?

How consistent?

What happens if I don't wait?

PBS

problem: no guarantees with eventual consistency

solution: consistency prediction

technique: measure latencies
use *WARS* model

Dynamo:

Amazon’s Highly Available Key-value Store

SOSP 2007

Apache, DataStax

Cassandra

basho

riak

LinkedIn

Project Voldemort

Adobe
IBM
Rackspace
Spotify
Palantir
Twitter
Cisco
Netflix

Cassandra

Reddit
Morningstar
Digg
Mozilla
Rhapsody
Shazam
Soundcloud
Gowalla

Yammer
Ask.com

Voldemort

LinkedIn
Gilt Groupe

Aol

Riak

Best Buy
Comcast
GitHub
Boeing
JoyentCloud

N = 3 replicas

R1 (“key”, 1)

R2 (“key”, 1)

R3 (“key”, 1)

Coordinator

read

R=3

client

N = 3 replicas

R1 (“key”, 1)

R2 (“key”, 1)

R3 (“key”, 1)

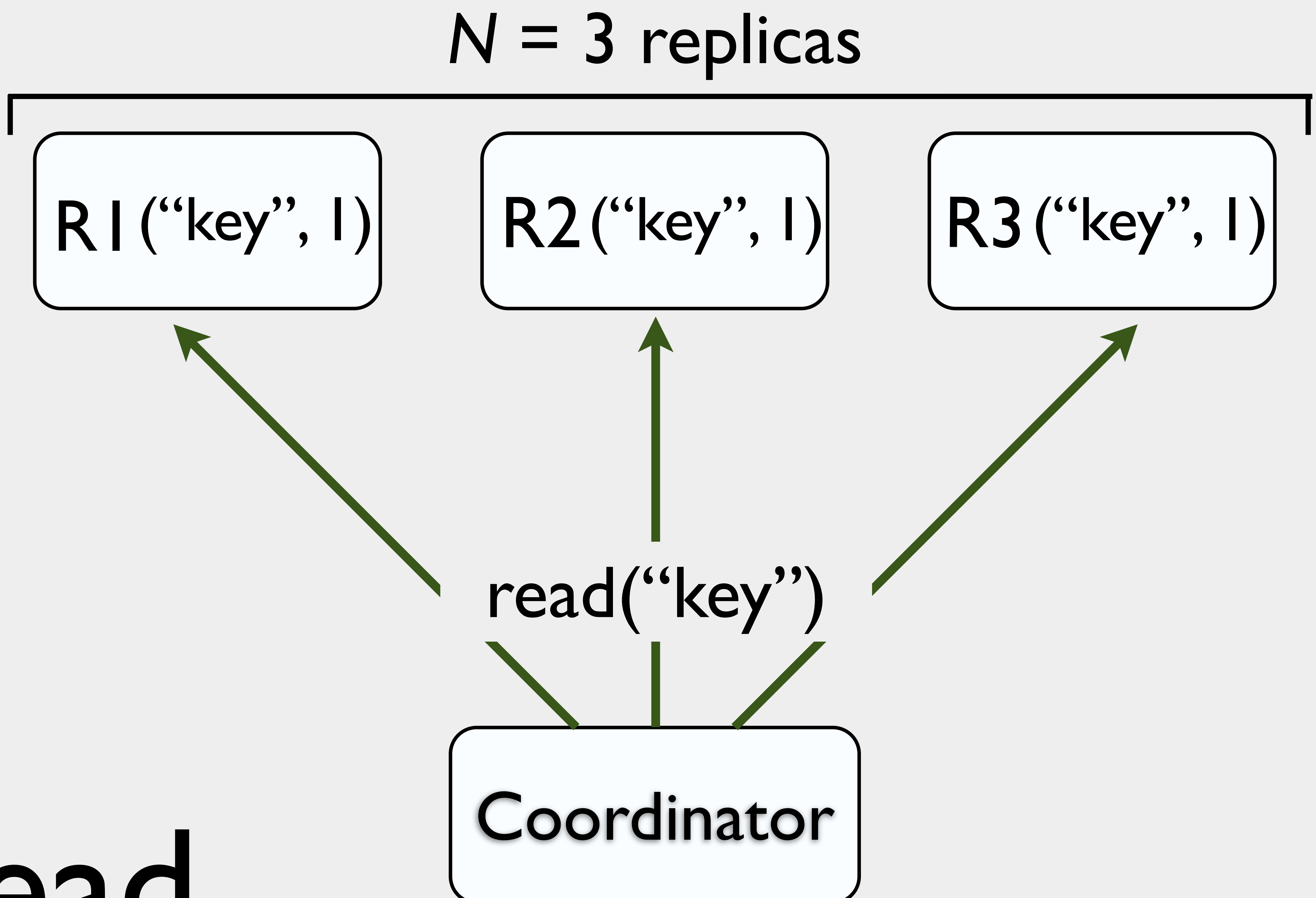
Coordinator

read(“key”)

client

read

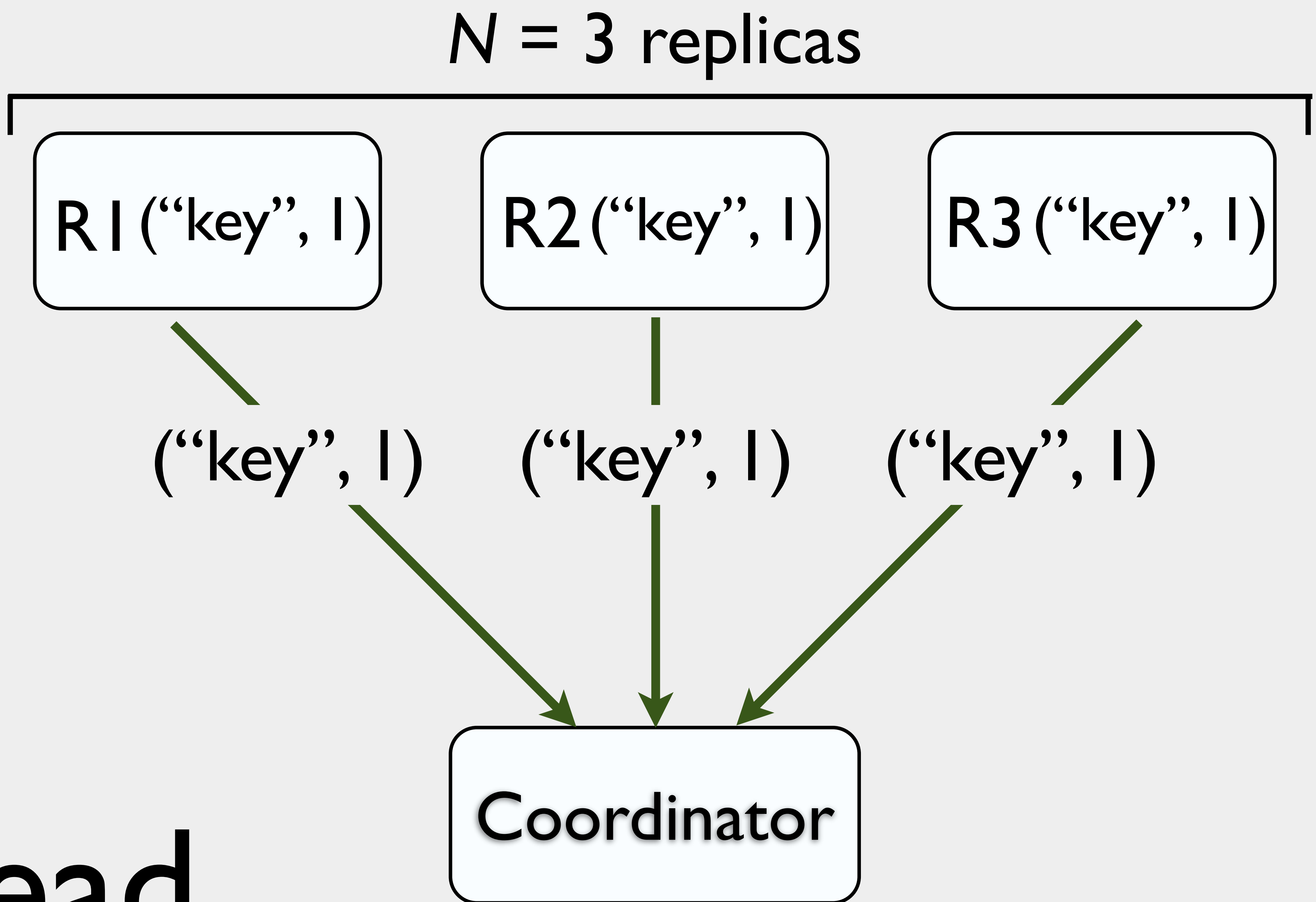
R=3


N = 3 replicas
R1 (“key”, 1)
R2 (“key”, 1)
R3 (“key”, 1)
read(“key”)
Coordinator

read

R=3

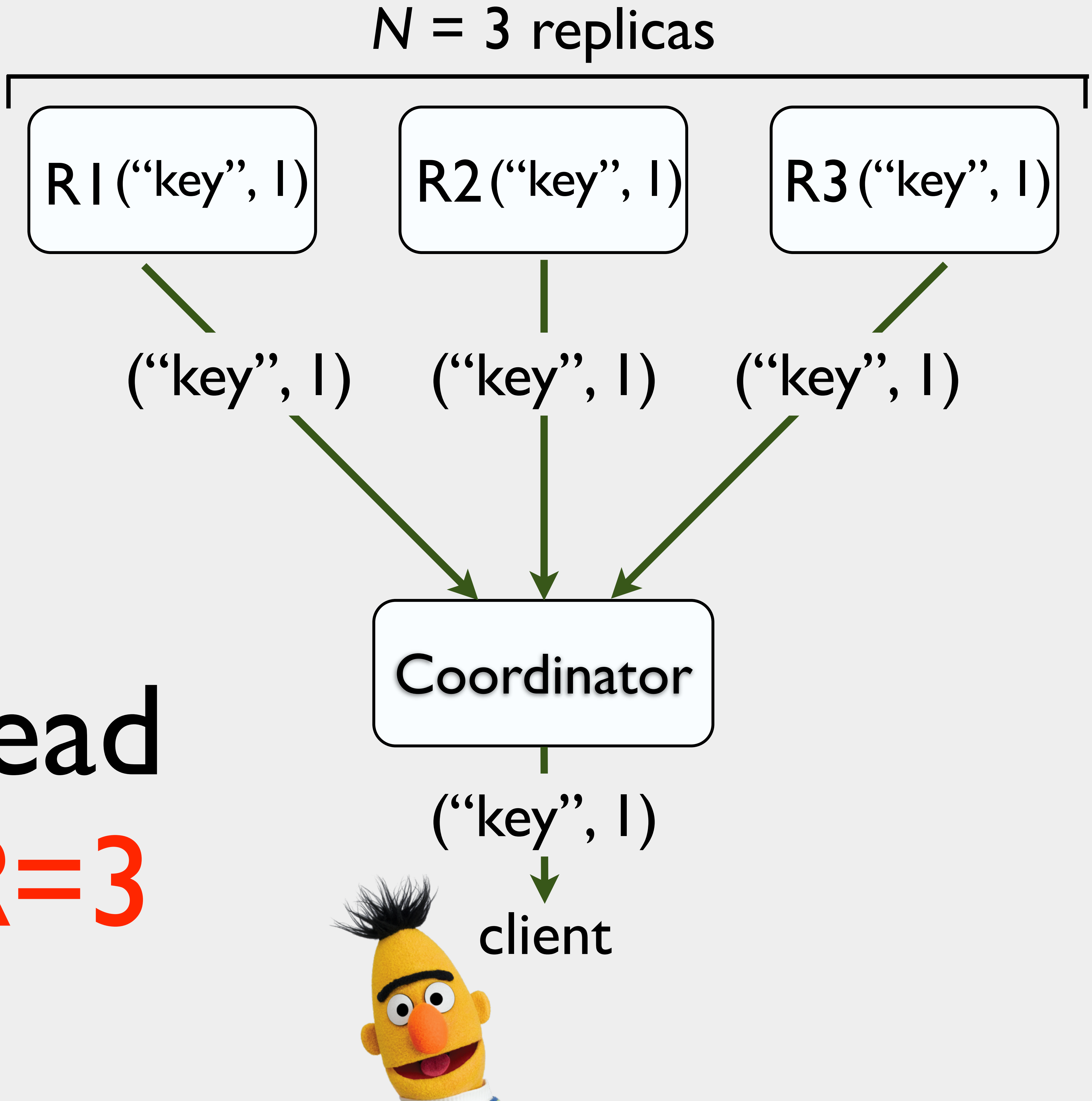

client



read

$R=3$

client



N = 3 replicas
R1 ("key", 1)
R2 ("key", 1)
R3 ("key", 1)
("key", 1)
("key", 1)
("key", 1)
Coordinator
("key", 1)
client

read

R=3

N = 3 replicas

R1 (“key”, 1)

R2 (“key”, 1)

R3 (“key”, 1)

Coordinator

read

R=3

client

N = 3 replicas

R1 (“key”, 1)

R2 (“key”, 1)

R3 (“key”, 1)

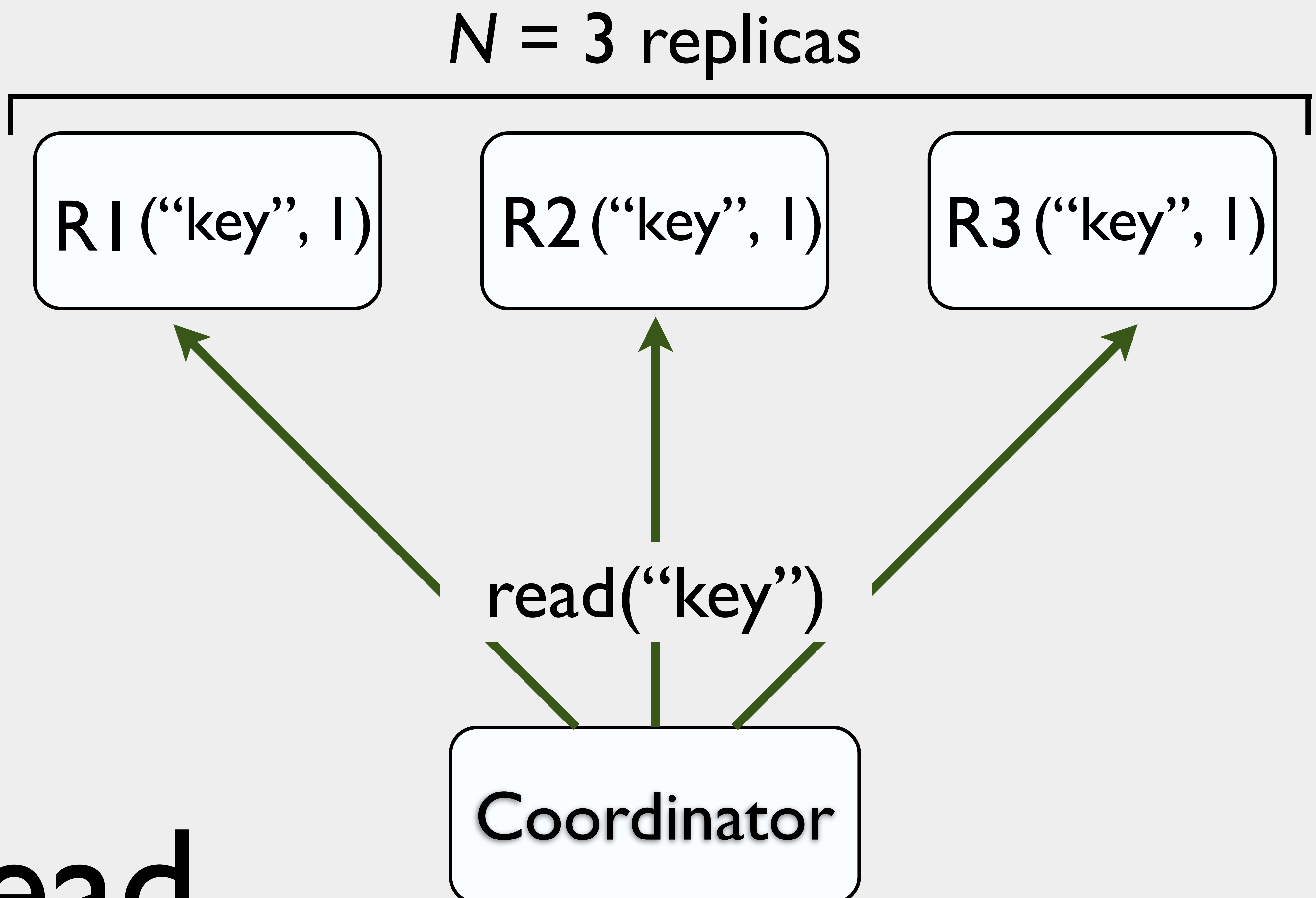
Coordinator

read(“key”)

client

read

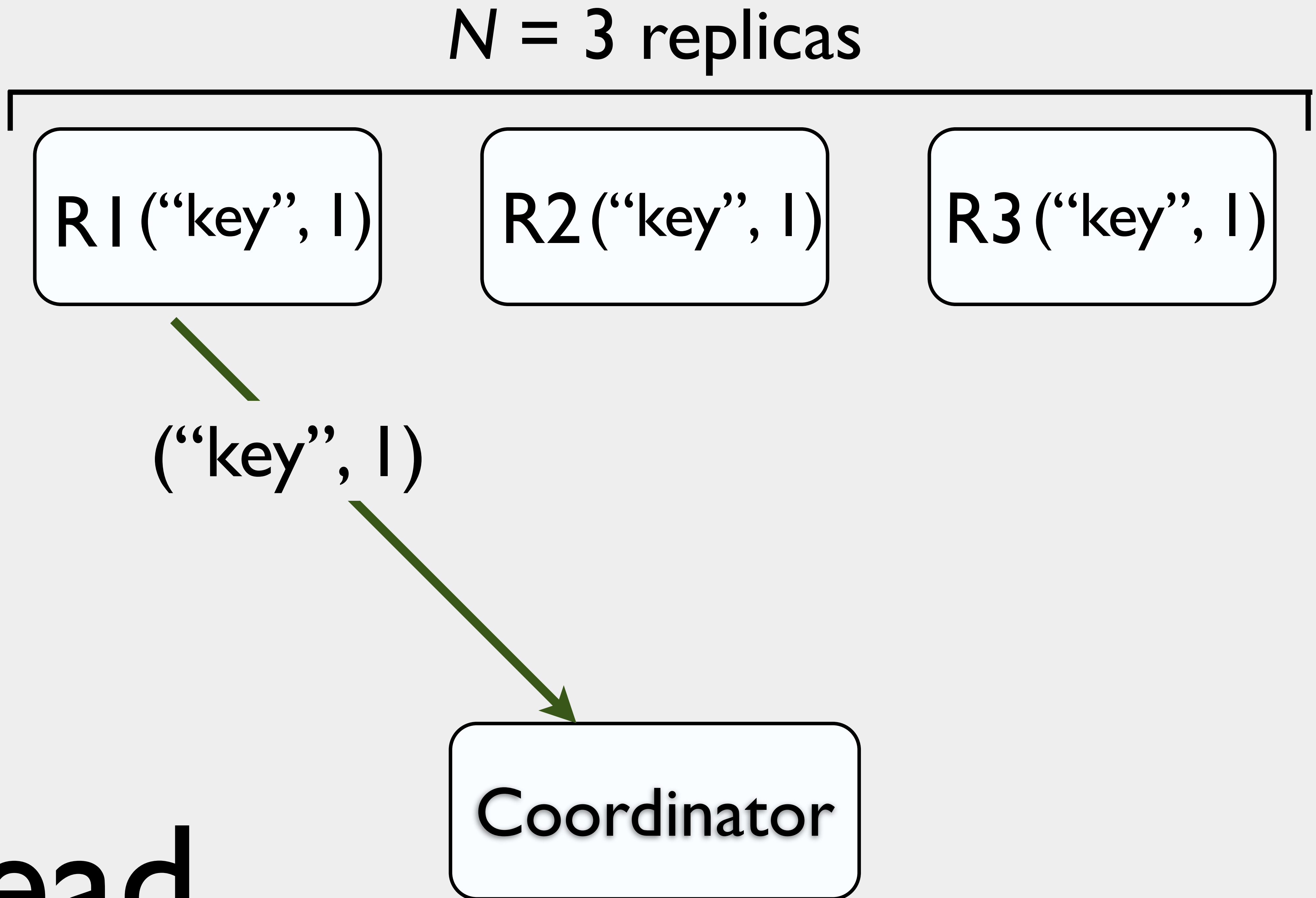
R=3



read

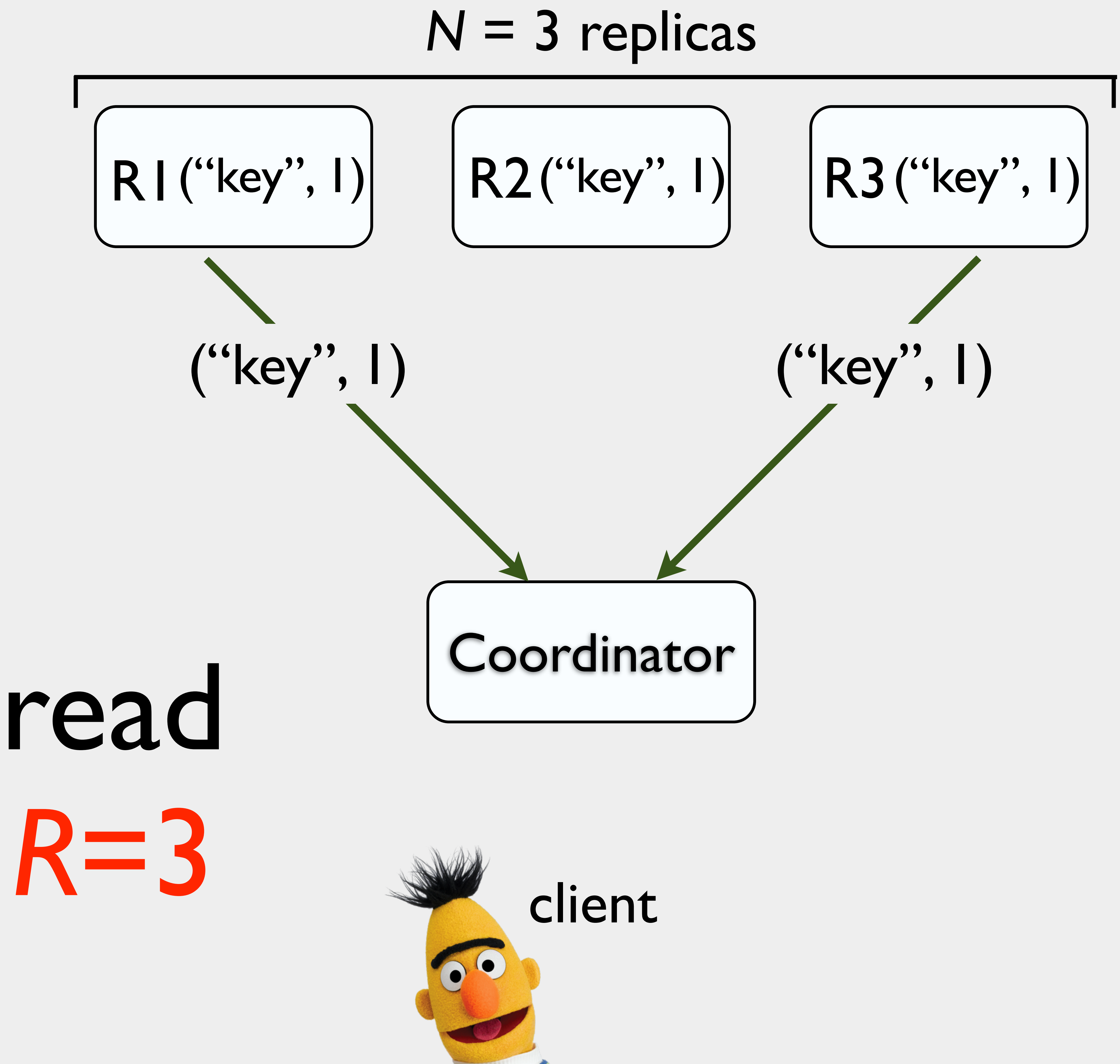
R=3

client

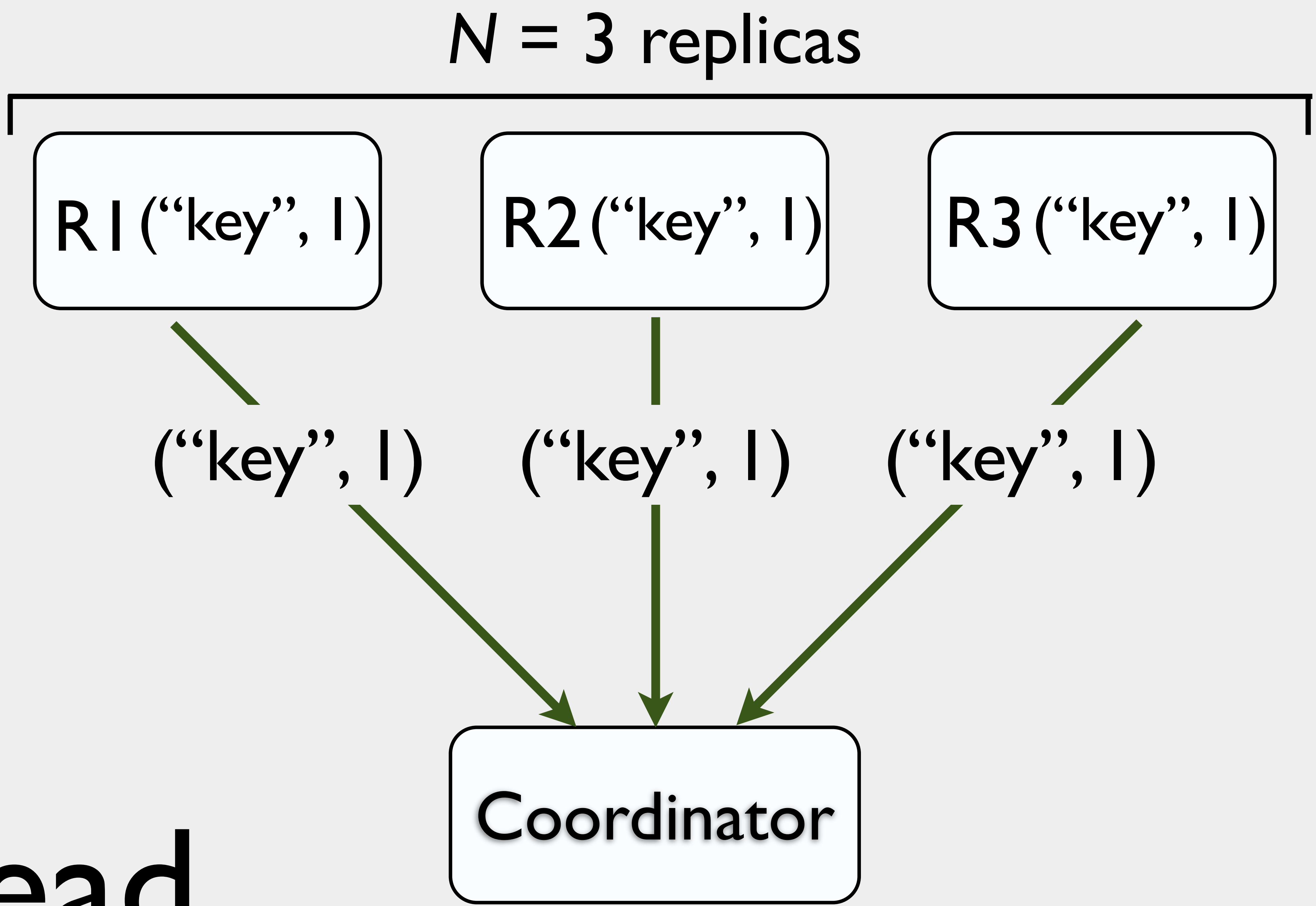


read

R=3



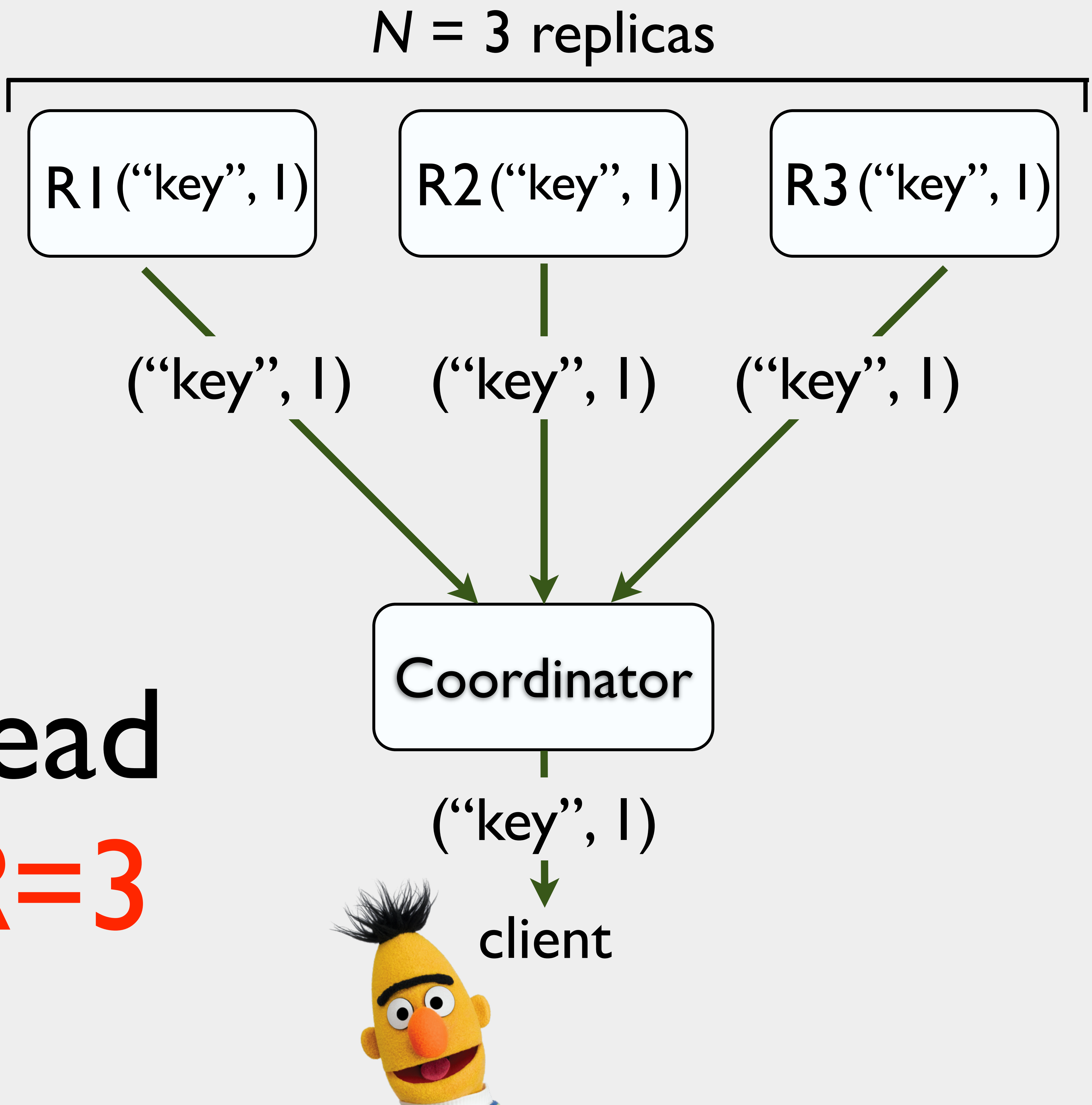
N = 3 replicas
R1 ("key", 1)
R2 ("key", 1)
R3 ("key", 1)
("key", 1)
("key", 1)
Coordinator
read
R=3
client



read

R=3

client


N = 3 replicas
R1 ("key", 1)
R2 ("key", 1)
R3 ("key", 1)
("key", 1)
("key", 1)
("key", 1)
Coordinator
("key", 1)
client

read

R=3

$N = 3$ replicas

R1 ("key", 1)

R2 ("key", 1)

R3 ("key", 1)

Coordinator

read

$R=1$

client

N = 3 replicas

R1 ("key", 1)

R2 ("key", 1)

R3 ("key", 1)

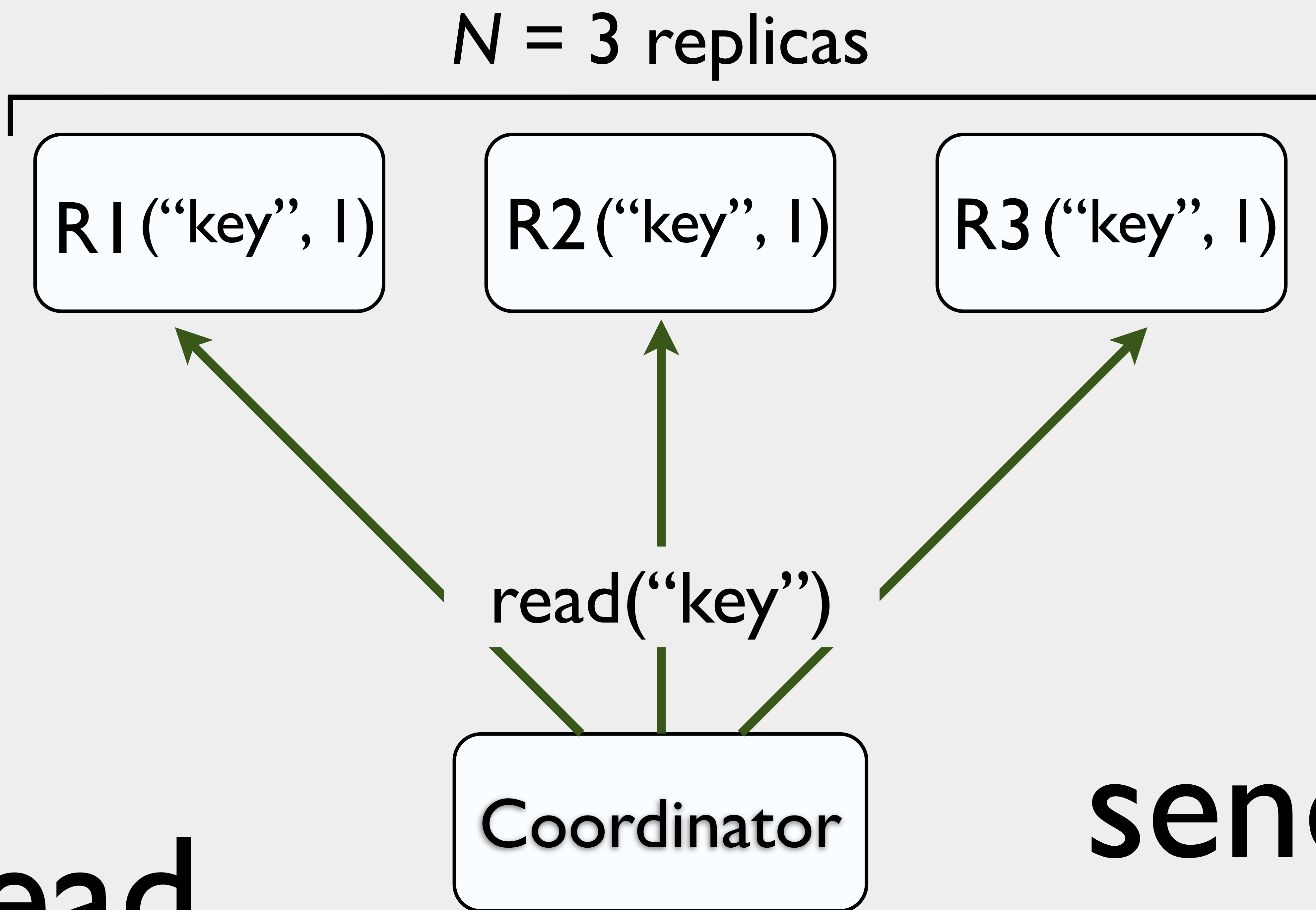
Coordinator

read

R=1

read("key")

client

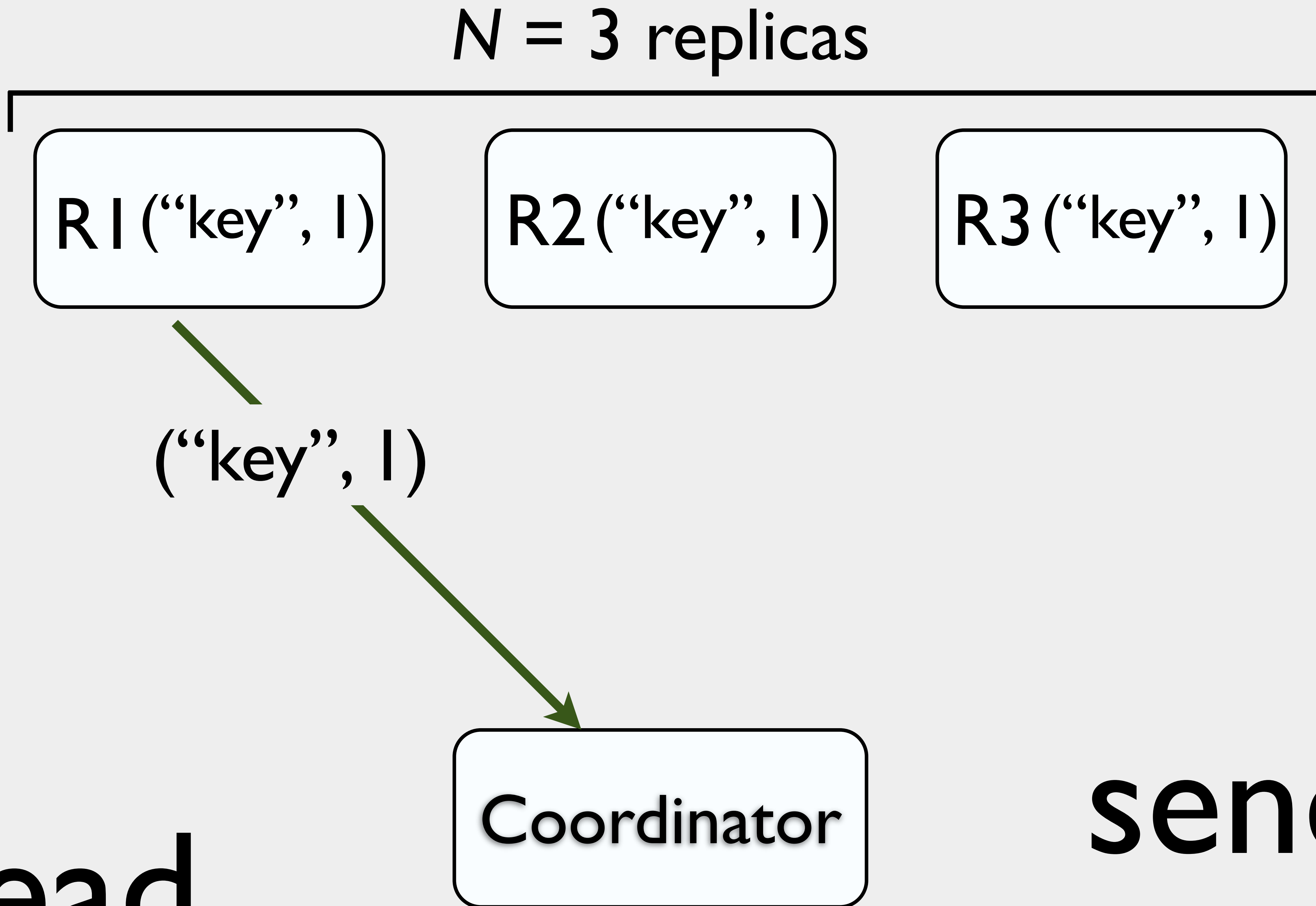


read

R=1

client

send read to all



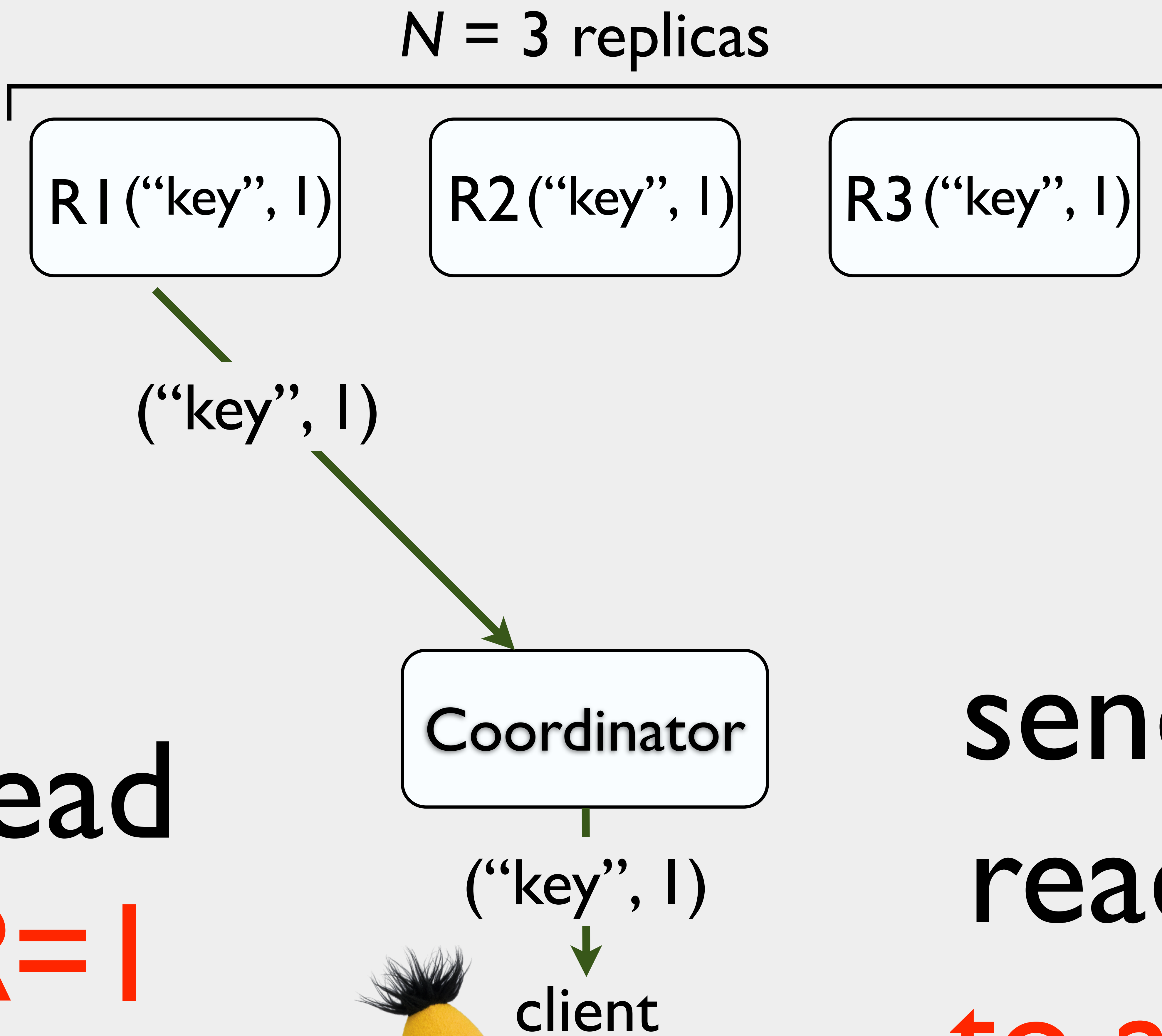
read
R=1

client

send read to all

read

R=1



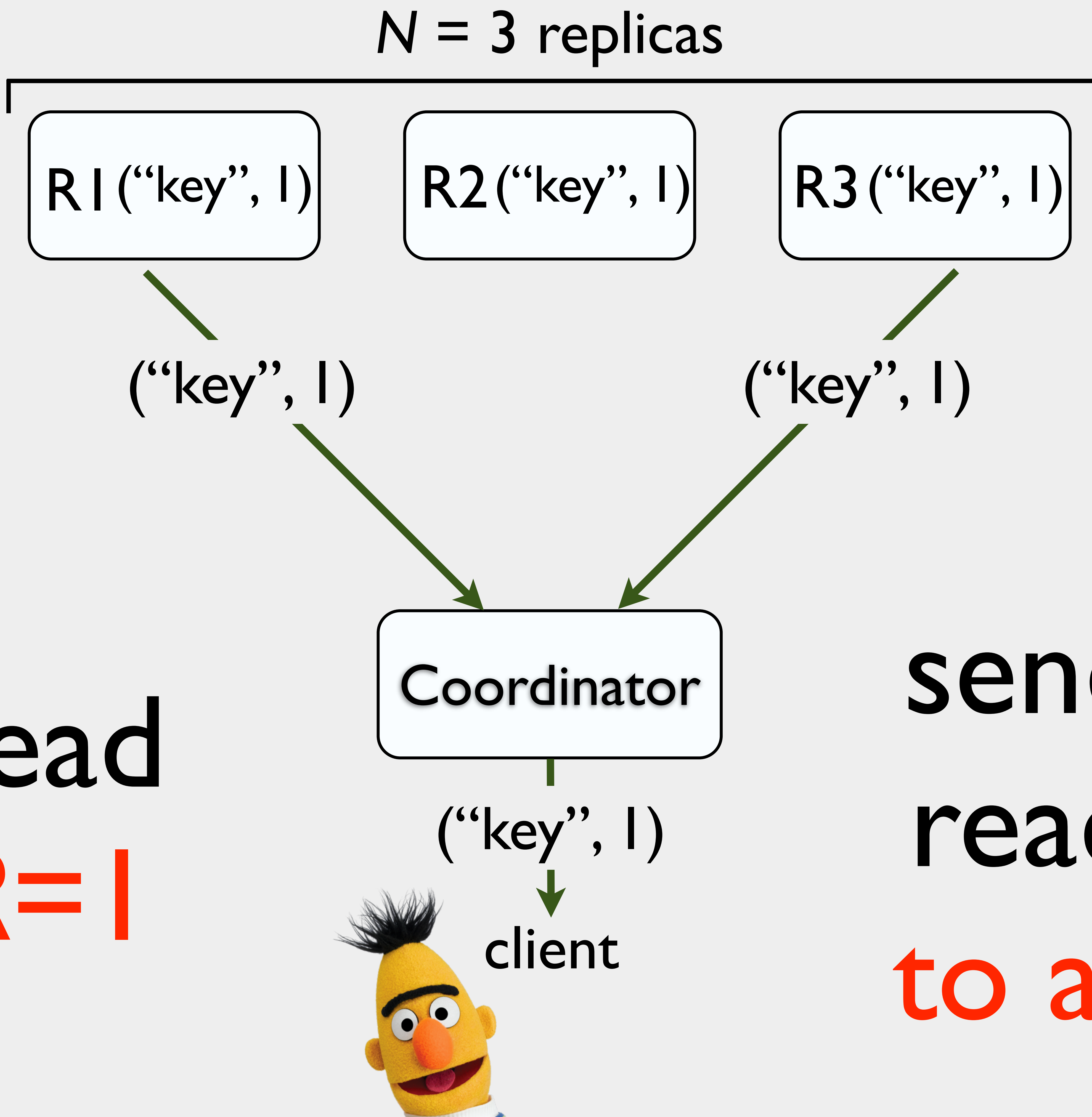
send

read

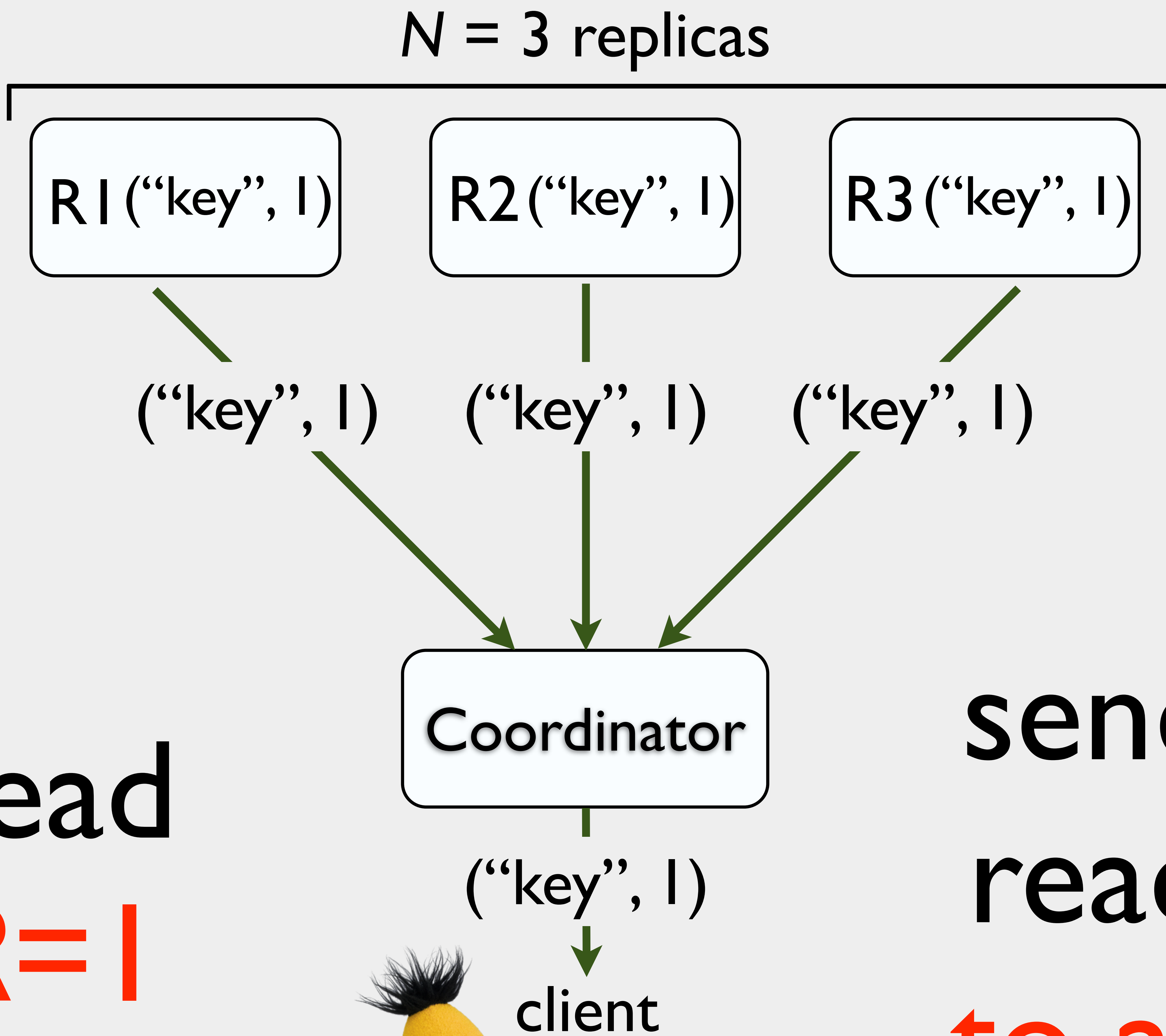
to all

read

R=1



send read to all



read

R=1

send read to all

N replicas/key

read: wait for *R* replies

write: wait for *W* acks

R1 (“key”, 1)

R2 (“key”, 1)

R3 (“key”, 1)

Coordinator

W=1

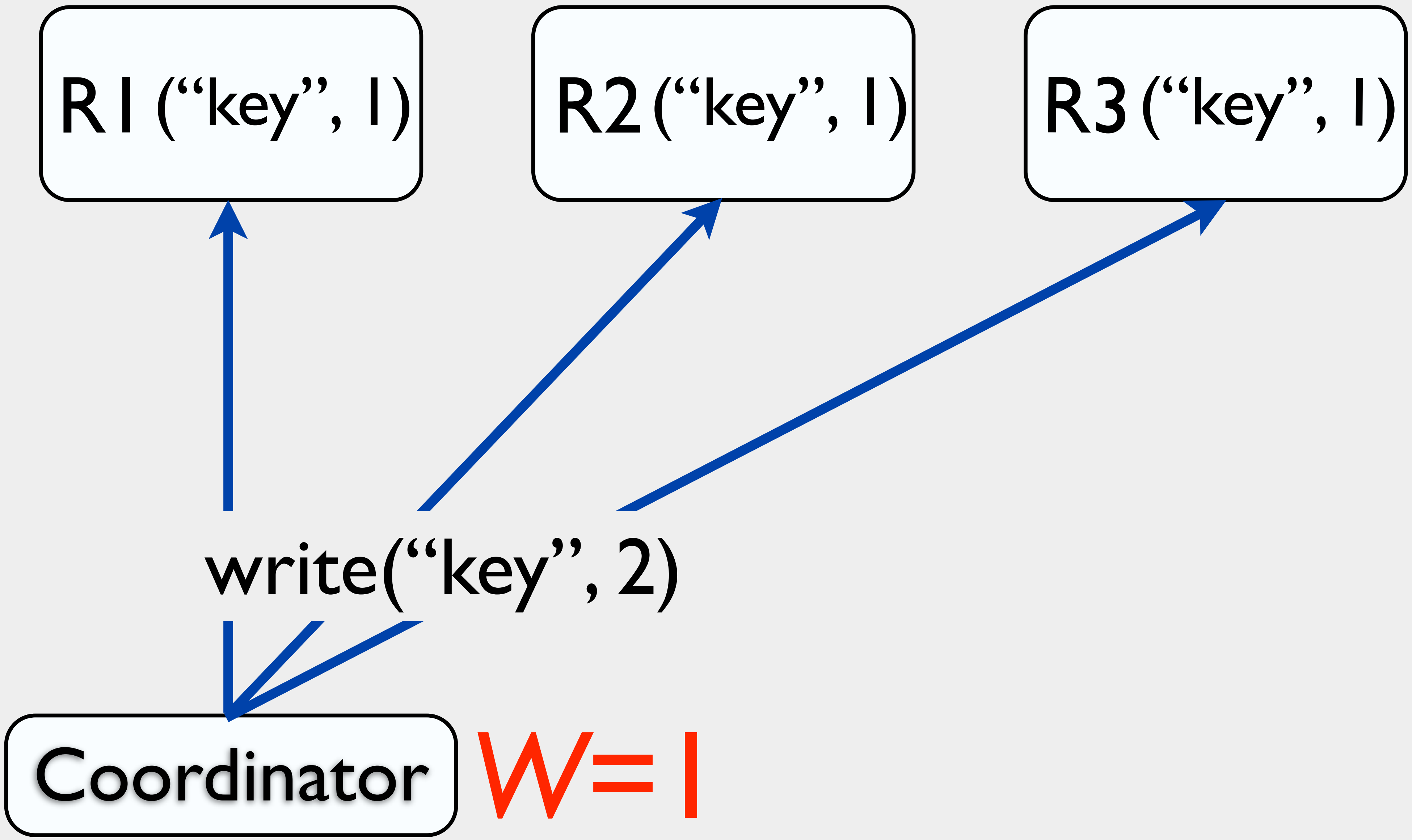
R1 ("key", 1)

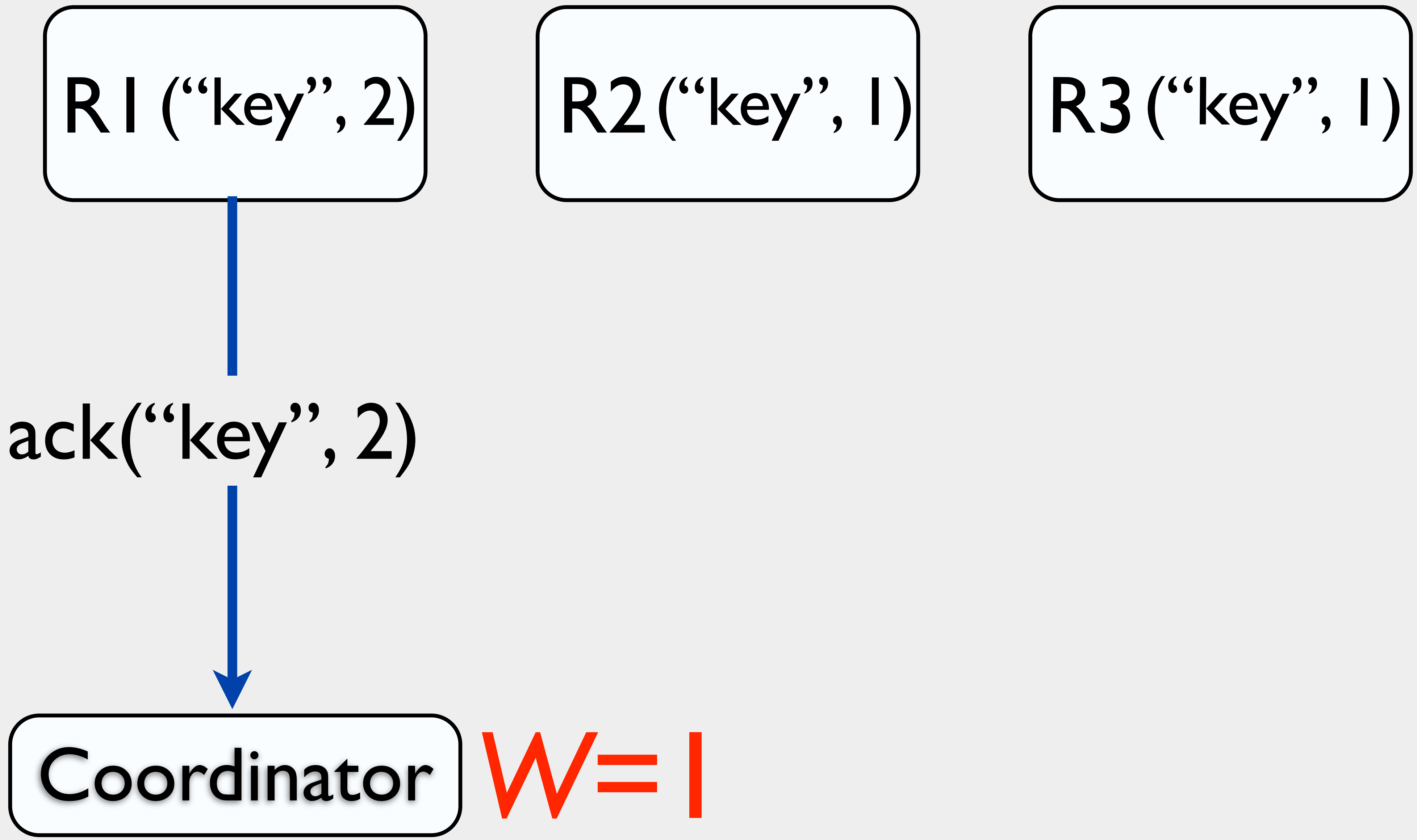
R2 ("key", 1)

R3 ("key", 1)

Coordinator W=1

write("key", 2)


R1 (“key”, 1)
R2 (“key”, 1)
R3 (“key”, 1)
write(“key”, 2)
Coordinator
W=1


R1 ("key", 2)
R2 ("key", 1)
R3 ("key", 1)
ack("key", 2)
Coordinator
W=1

R1 (“key”, 2)

R2 (“key”, 1)

R3 (“key”, 1)

ack(“key”, 2)

Coordinator

W=1

ack(“key”, 2)

R1 ("key", 2)

R2 ("key", 1)

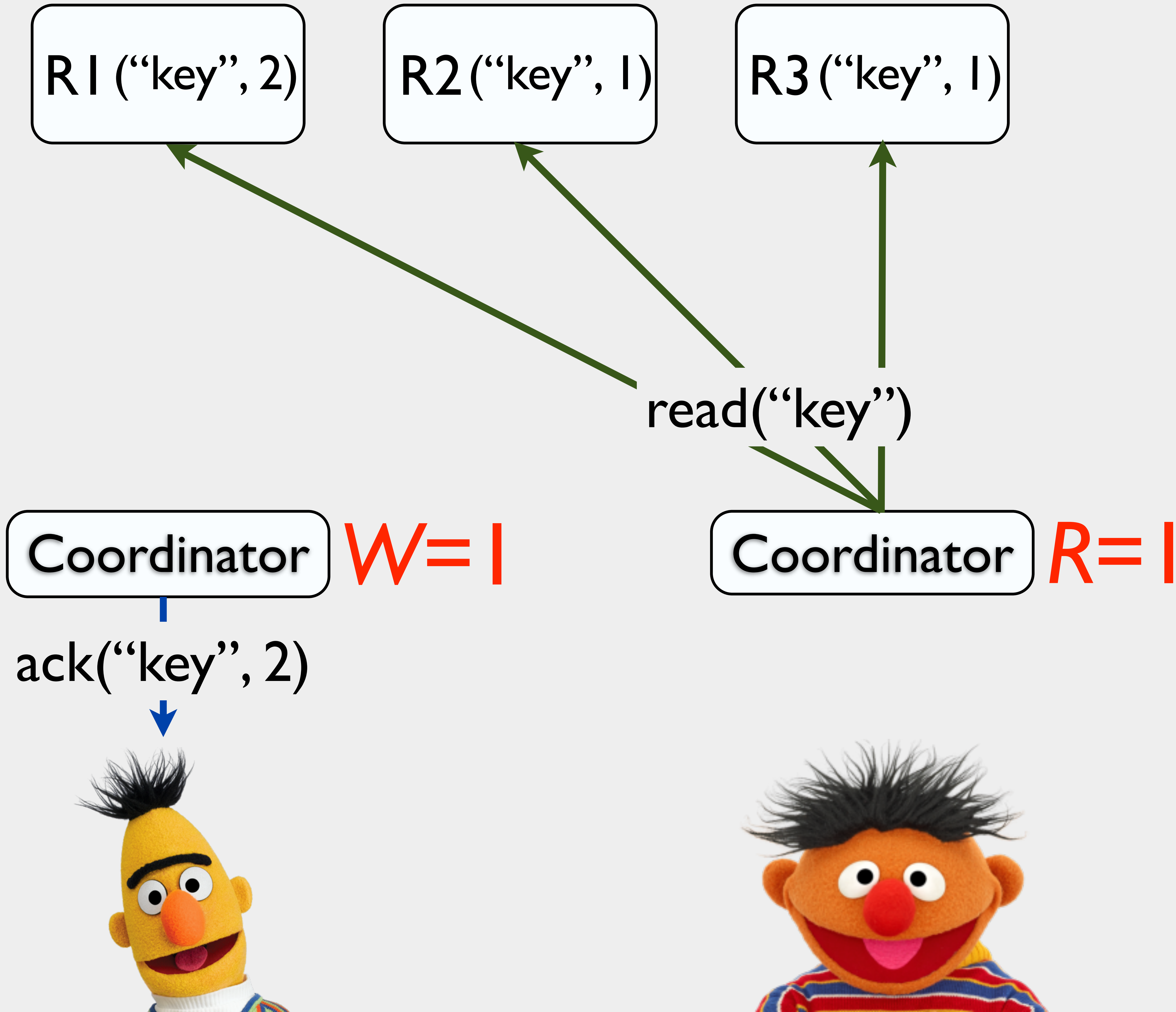
R3 ("key", 1)

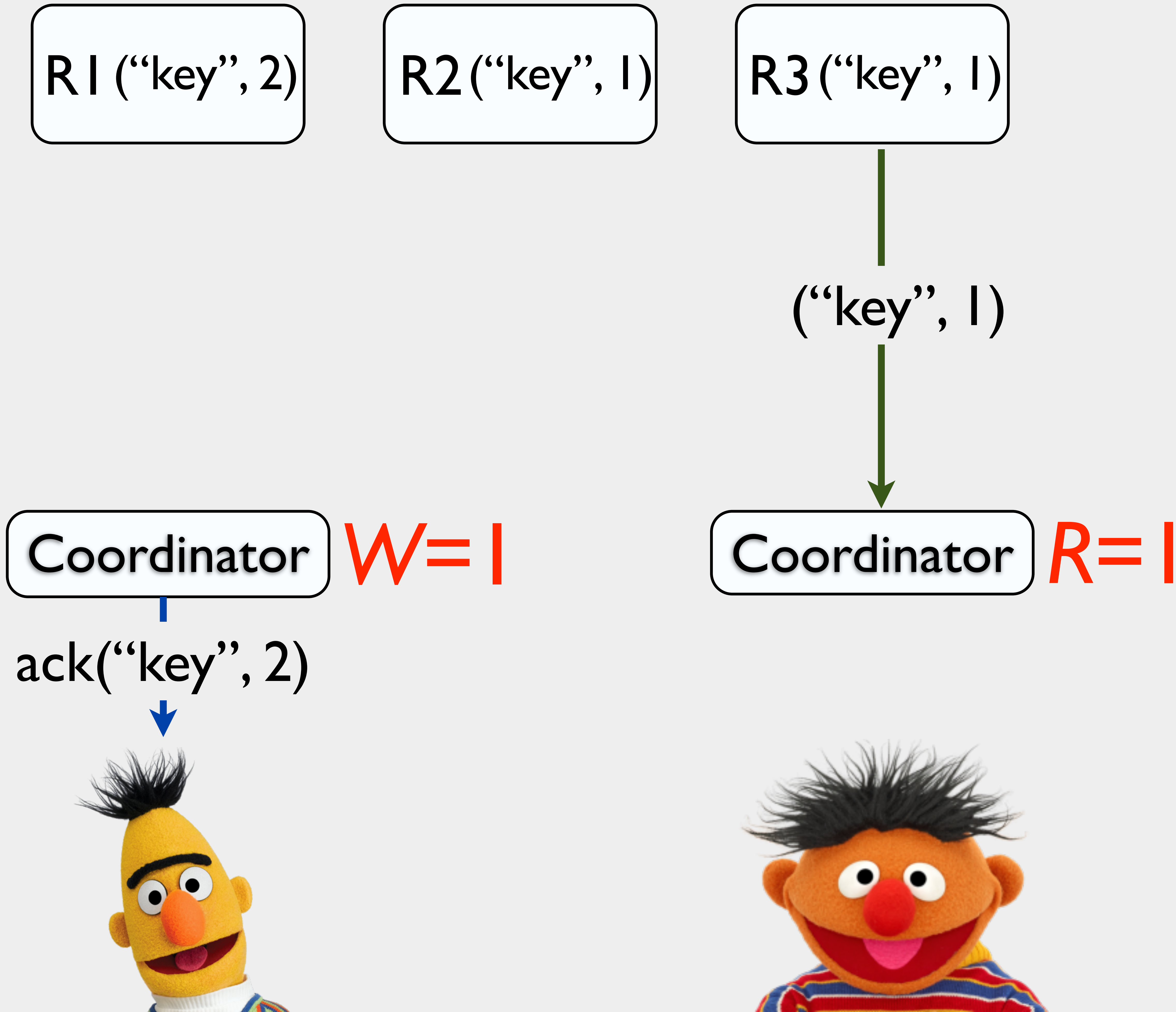
Coordinator W=1

ack("key", 2)

Coordinator R=1

read("key")


R1 ("key", 2)
R2 ("key", 1)
R3 ("key", 1)
read("key")
Coordinator
W=1
ack("key", 2)
Coordinator
R=1


R1 ("key", 2)
R2 ("key", 1)
R3 ("key", 1)
("key", 1)
Coordinator
W=1
ack("key", 2)
Coordinator
R=1

R1 (“key”, 2)

R2 (“key”, 1)

R3 (“key”, 1)

Coordinator W=1

ack(“key”, 2)

Coordinator R=1

(“key”, 1)

R1 (“key”, 2)

R2 (“key”, 1)

R3 (“key”, 1)

Coordinator W=1

ack(“key”, 2)

Coordinator R=1

(“key”, 1)

R1 (“key”, 2)

R2 (“key”, 1)

R3 (“key”, 1)

Coordinator W=1

ack(“key”, 2)

Coordinator R=1

(“key”, 1)

R1 (“key”, 2)

R2 (“key”, 1)

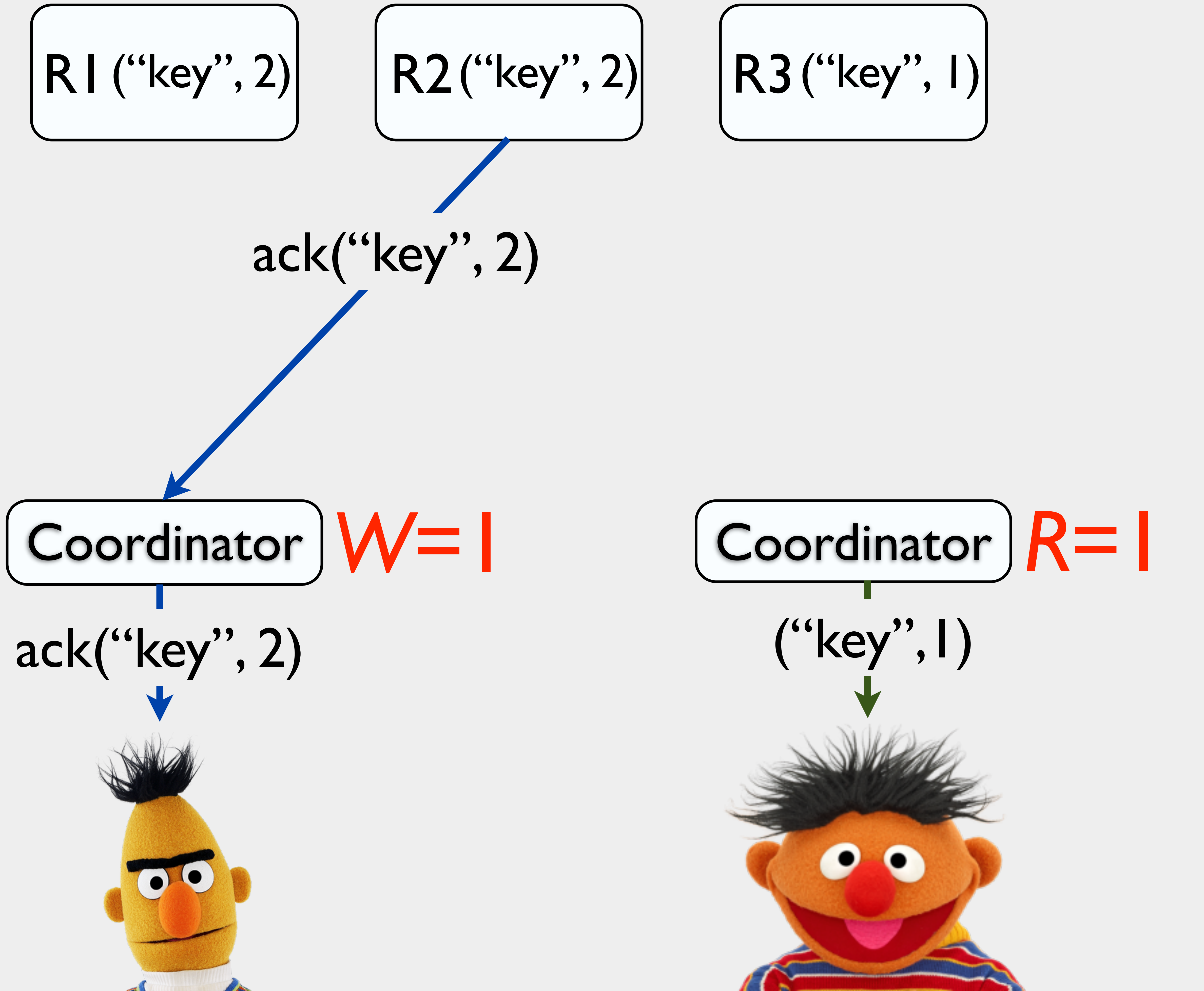
R3 (“key”, 1)

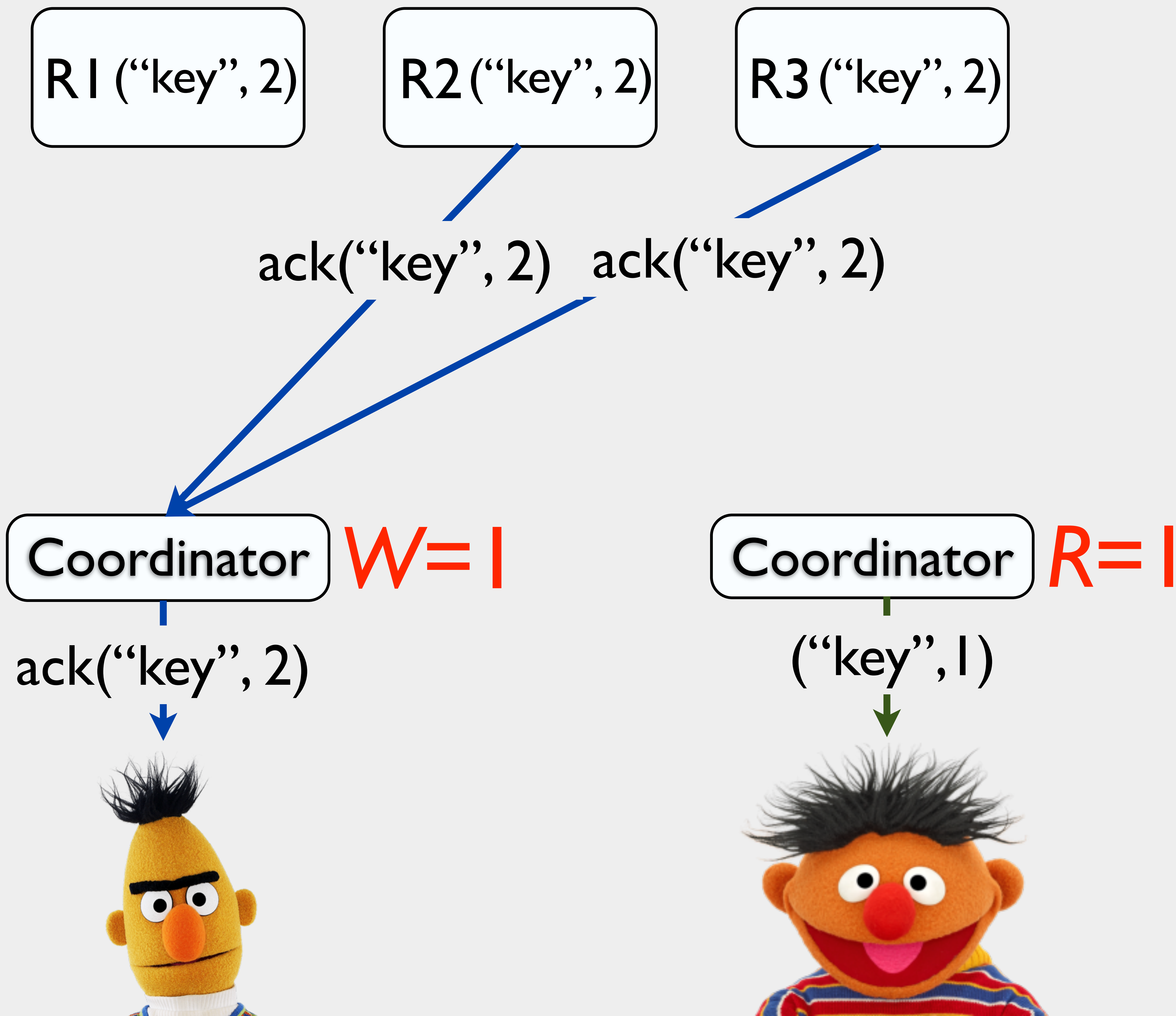
Coordinator W=1

ack(“key”, 2)

Coordinator R=1

(“key”, 1)


R1 ("key", 2)
R2 ("key", 2)
R3 ("key", 1)
ack("key", 2)
Coordinator
W=1
ack("key", 2)
Coordinator
R=1
("key", 1)


R1 ("key", 2)
R2 ("key", 2)
R3 ("key", 2)
ack("key", 2)
ack("key", 2)
Coordinator
W=1
ack("key", 2)
Coordinator
R=1
("key", 1)

R1 (“key”, 2)

R2 (“key”, 2)

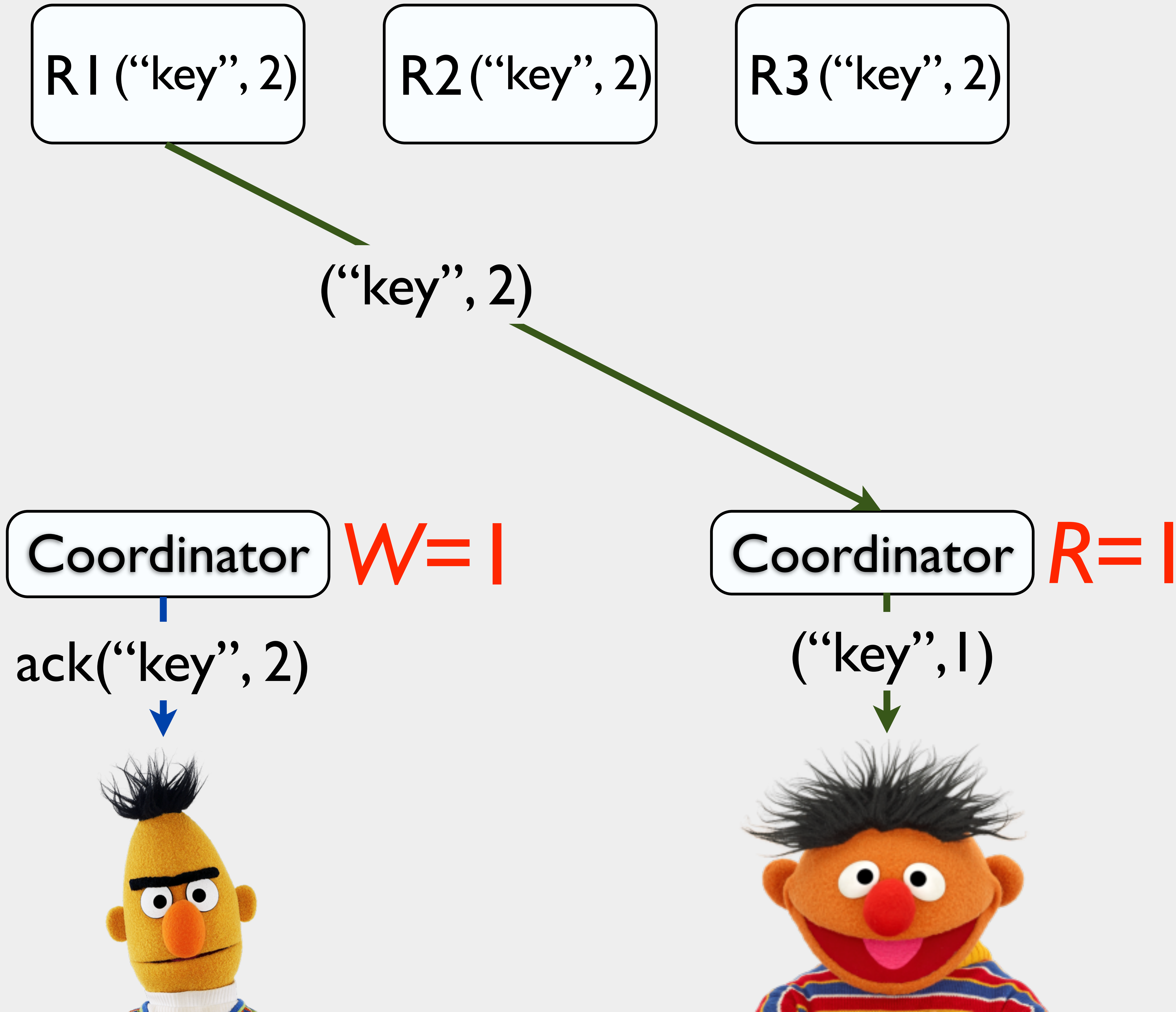
R3 (“key”, 2)

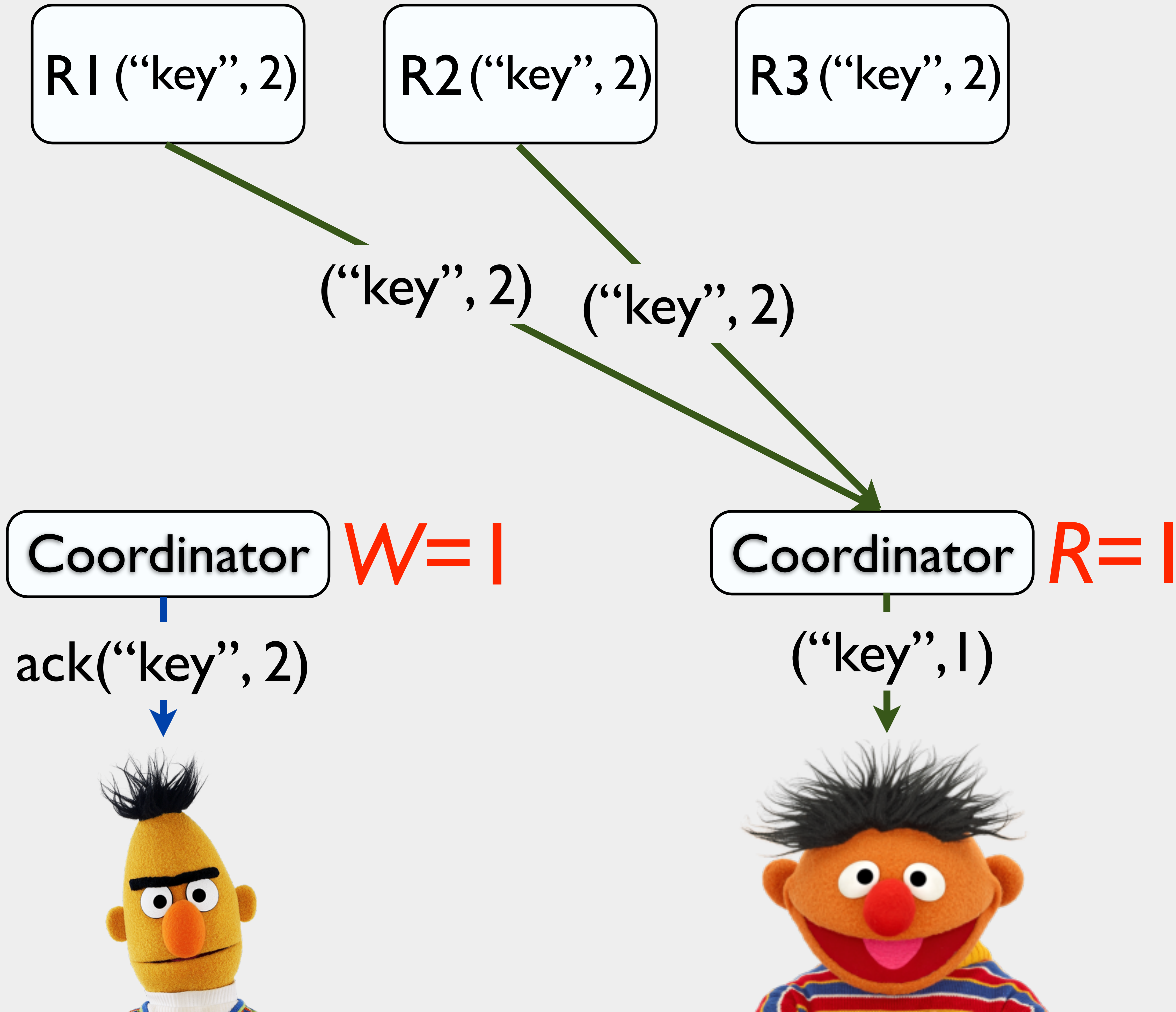
Coordinator W=1

ack(“key”, 2)

Coordinator R=1

(“key”, 1)


R1 ("key", 2)
R2 ("key", 2)
R3 ("key", 2)
("key", 2)
Coordinator
W=1
ack("key", 2)
Coordinator
R=1
("key", 1)



R1 ("key", 2)
R2 ("key", 2)
R3 ("key", 2)
("key", 2)
("key", 2)
Coordinator
W=1
ack("key", 2)
Coordinator
R=1
("key", 1)

R1 (“key”, 2)

R2 (“key”, 2)

R3 (“key”, 2)

Coordinator W=1

ack(“key”, 2)

Coordinator R=1

(“key”, 1)

if: $R+W > N$

then: “strong” consistency

else: eventual consistency

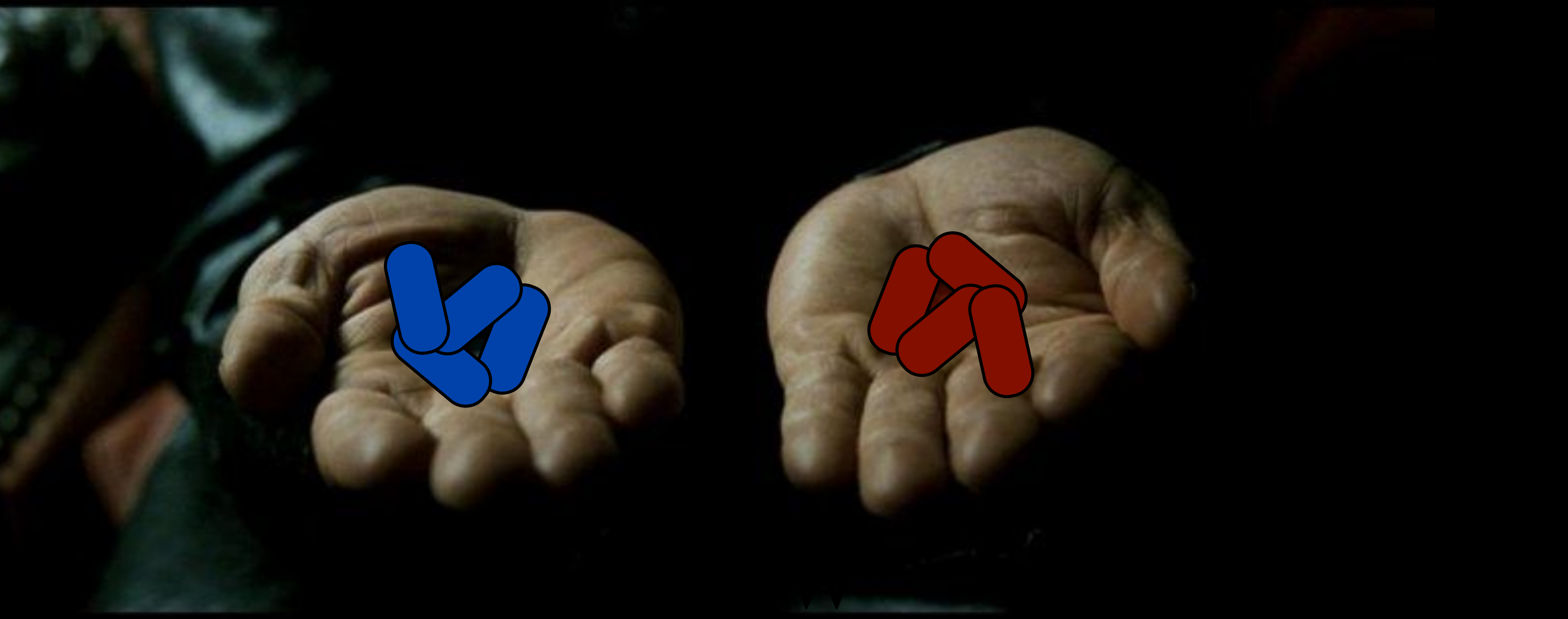
strong consistency

strong consistency

lower latency

strong consistency

lower latency



strong consistency

lower latency

Cassandra:

R=W=1, N=3

by **default**

$(1+1 \not> 3)$

"In the general case, we typically use [Cassandra's] consistency level of [R=W=1], which provides maximum performance. Nice!"

--D. Williams,
"HBase vs Cassandra: why we moved"
February 2010

http://ria101.wordpress.com/2010/02/24/hbase-vs-cassandra-why-we-moved/

reddit PROGRAMMING comments related other discussions (1)

504

reddit's now running on Cassandra (blog.reddit.com)

submitted 1 year ago by ketralnis

261 comments share

sorted by: best

you are viewing a single comment's thread.
view the rest of the comments →

[–] ketralnis [S] 13 points 1 year ago

We have a memcached (not memcachedb) in front of it which gives us the atomic operations that we need, so it can take as long as it needs to replicate behind the scenes

If we didn't, we'd use CL-ONE reads/writes for most things except the operations that needed to be atomic, where we'd do CL-QUORUM. But most of our data doesn't need atomic reads/writes.

http://www.reddit.com/r/programming/comments/bcqhi/reddits_now_running_on_cassandra/c0m3wh6 reddit PROGRAMMING comments related other discussions (1)

504 reddit's now running on Cassandra (blog.reddit.com)

submitted 1 year ago by ketralnis

261 comments share

sorted by: best

you are viewing a single comment's thread.
view the rest of the comments →

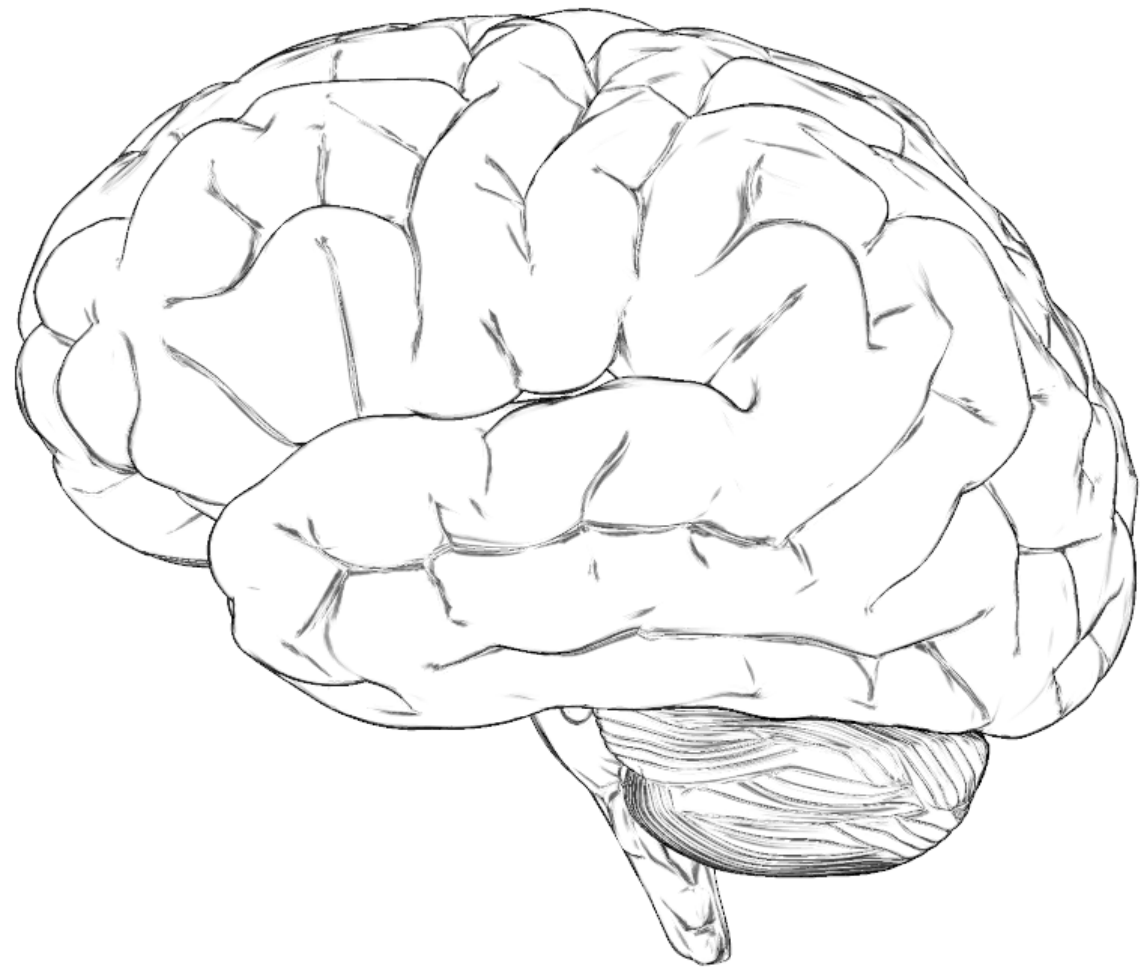
[–] ketralnis [S] 13 points 1 year ago

We have a memcached (not memcachedb) in front of it which gives us the atomic operations that we need, so it can take as long as it needs to replicate behind the scenes

If we didn't, we'd use CL-ONE reads/writes for most things except the operations that needed to be atomic, where we'd do CL-QUORUM. But most of our data doesn't need atomic reads/writes.

http://www.reddit.com/r/programming/comments/bcqhi/reddits_now_running_on_cassandra/c0m3wh6

NoSQL Primer



Low Value Data

n = 2, r = 1, w = 1

Low Value Data

$n = 2, r = 1, w = 1$

Mission Critical Data

n = 5, r = 1, w = 5, dw = 5

Mission Critical Data

$\boxed{n = 5, r = 1, w = 5,}$ $dw = 5$

Voldemort @ LinkedIn

"very low latency and high availability":

R=W=1, N=3

N=3 not required, "some consistency":

R=W=1, N=2

Alex Feinberg, personal communication

Anecdotally, EC "worthwhile" for many kinds of data

Anecdotally, EC "worthwhile" for many kinds of data

How eventual?

How consistent?

Anecdotally, EC "worthwhile" for many kinds of data

How eventual?

How consistent?

"eventual and consistent enough"

Can we do better?

Can we do better?

Probabilistically Bounded Staleness

can't make *promises*

<u>can</u> give *expectations*

PBS is:

a way to quantify latency-consistency trade-offs

what’s the latency cost of consistency?

what’s the consistency cost of latency?

PBS is:

a way to quantify
latency-consistency
trade-offs

what's the latency cost of consistency?
what's the consistency cost of latency?

an "SLA" for consistency

How eventual?

t-visibility: probability *p* of consistent reads after after *t* seconds

(e.g., 99.9% of reads will be consistent after 10ms)

t-visibility depends on:

1) message delays
2) background version exchange (anti-entropy)

t-visibility depends on:

1) message delays
2) ~~background version exchange (anti-entropy)~~

anti-entropy:
- only decreases staleness
- comes in many flavors
- hard to guarantee rate

Focus on message delays

focus on

steady state

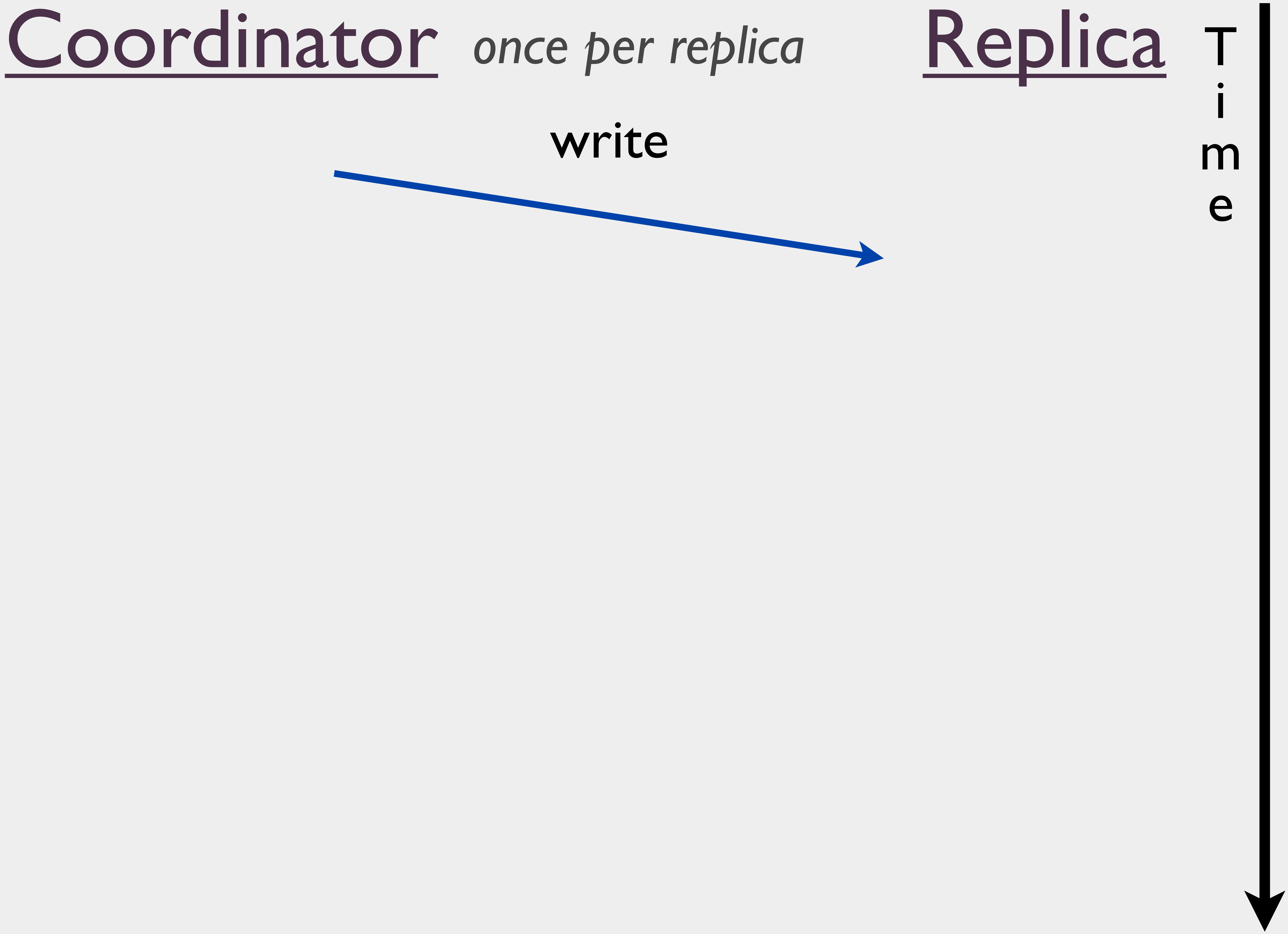
with failures:

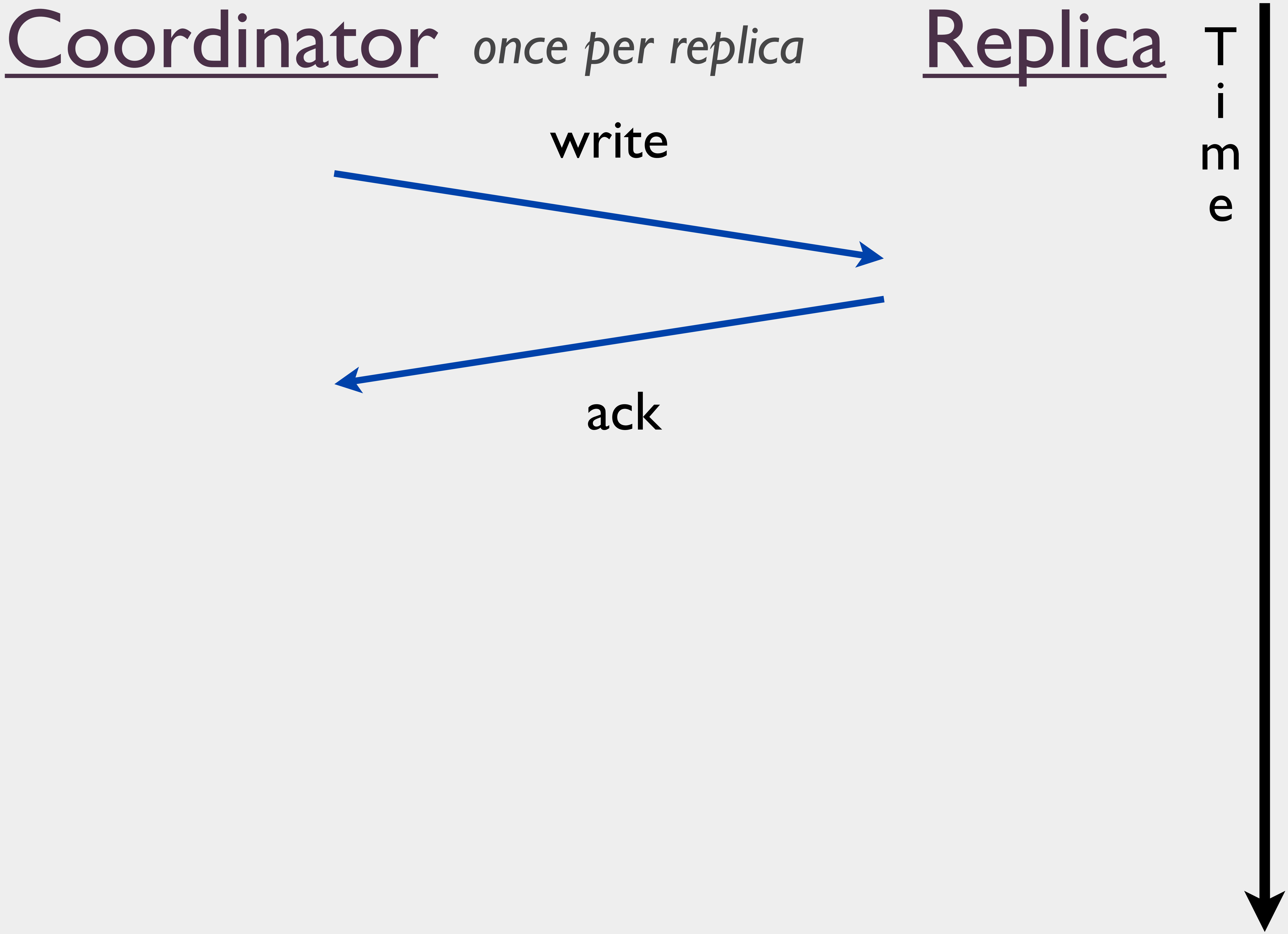
unavailable

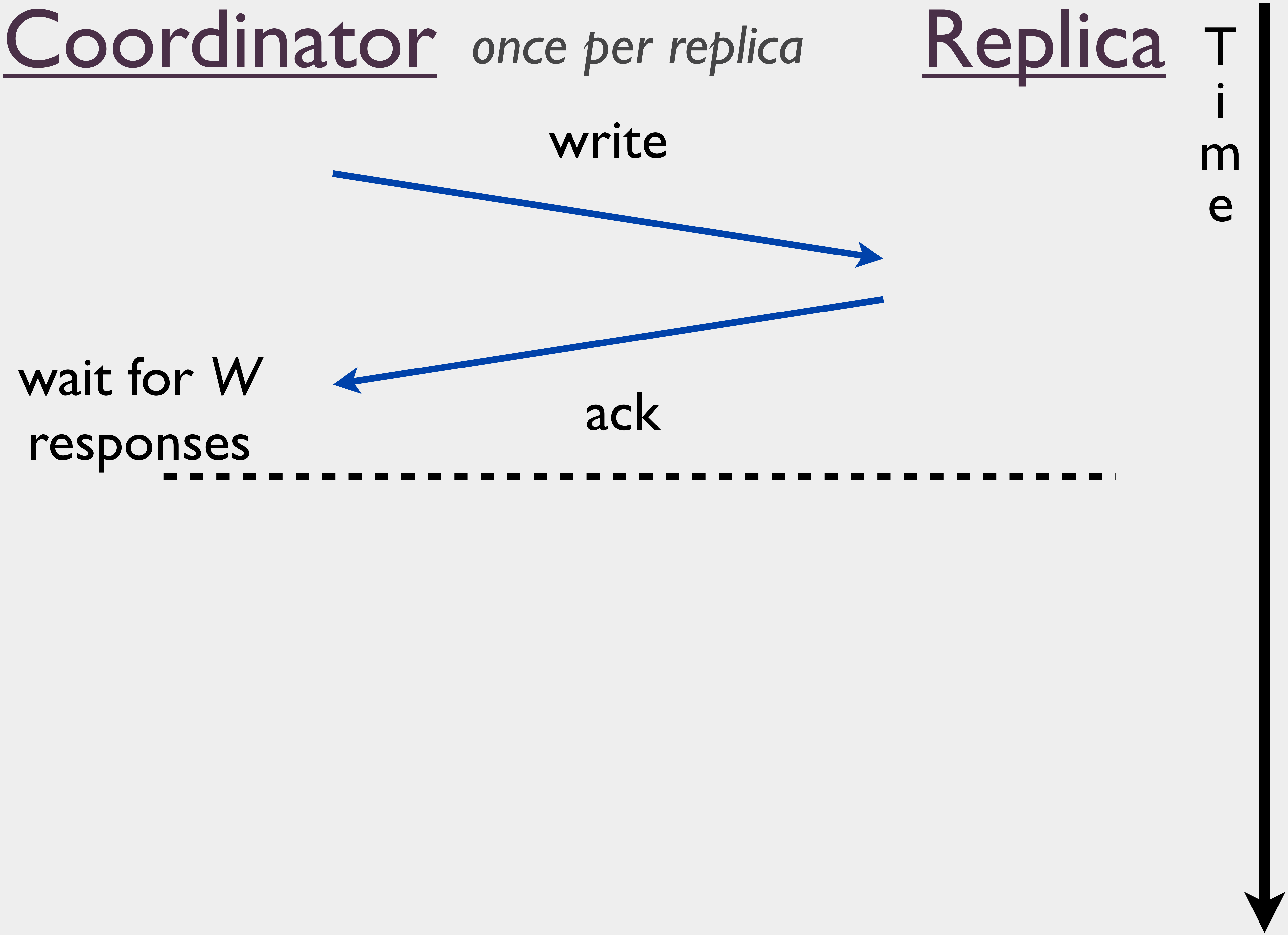
or sloppy

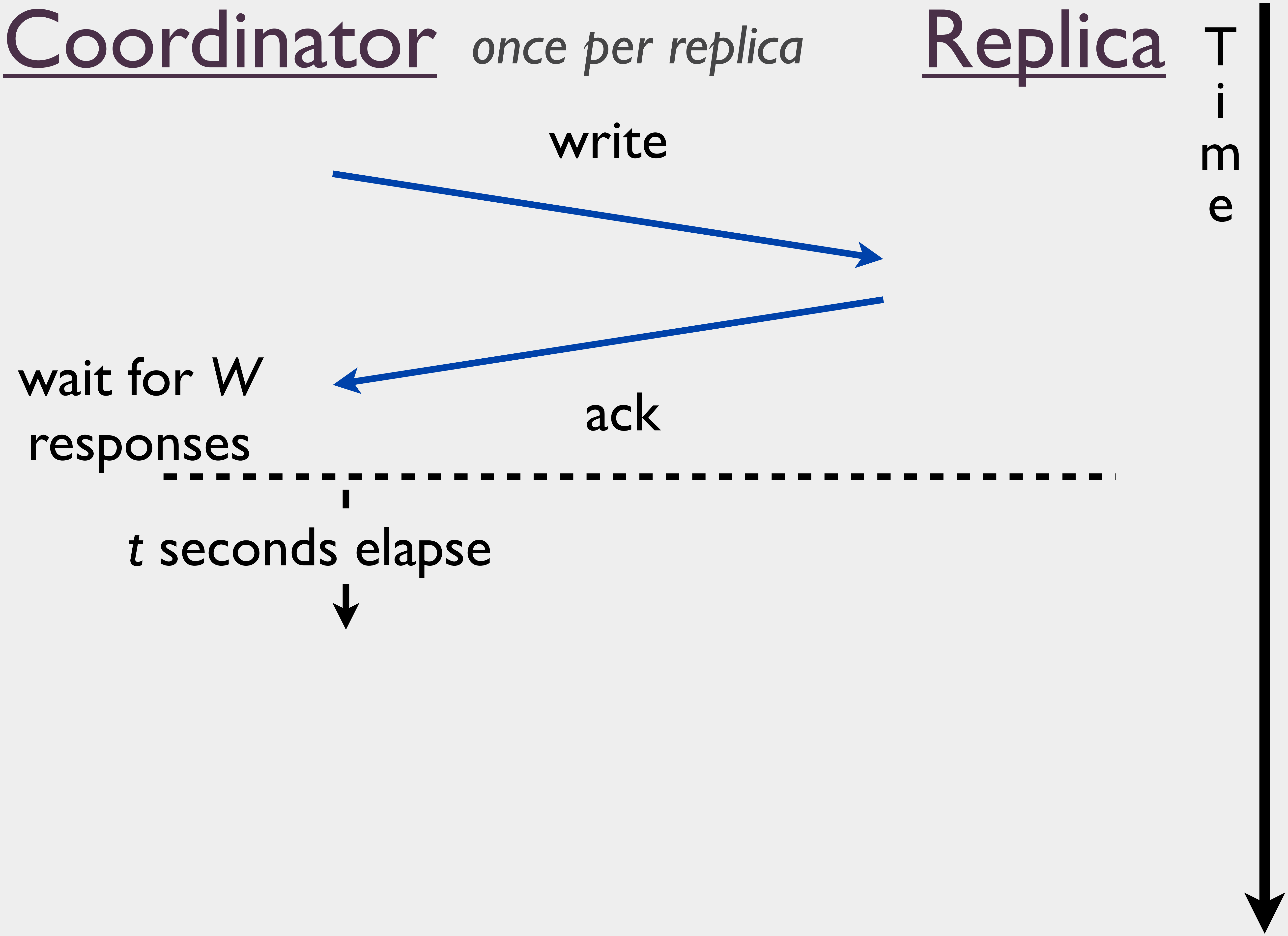
Coordinator *once per replica* Replica

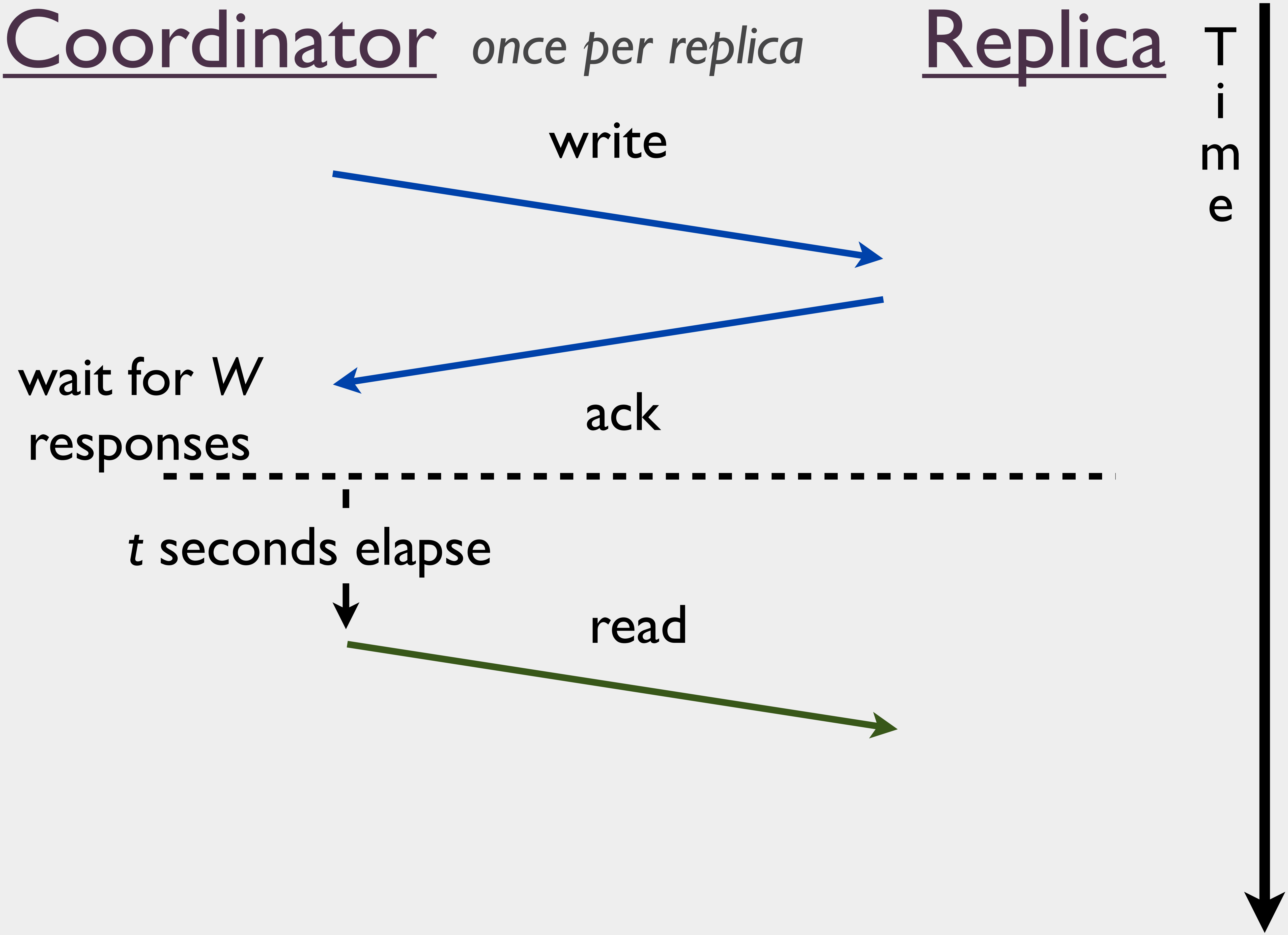
T
i
m
e

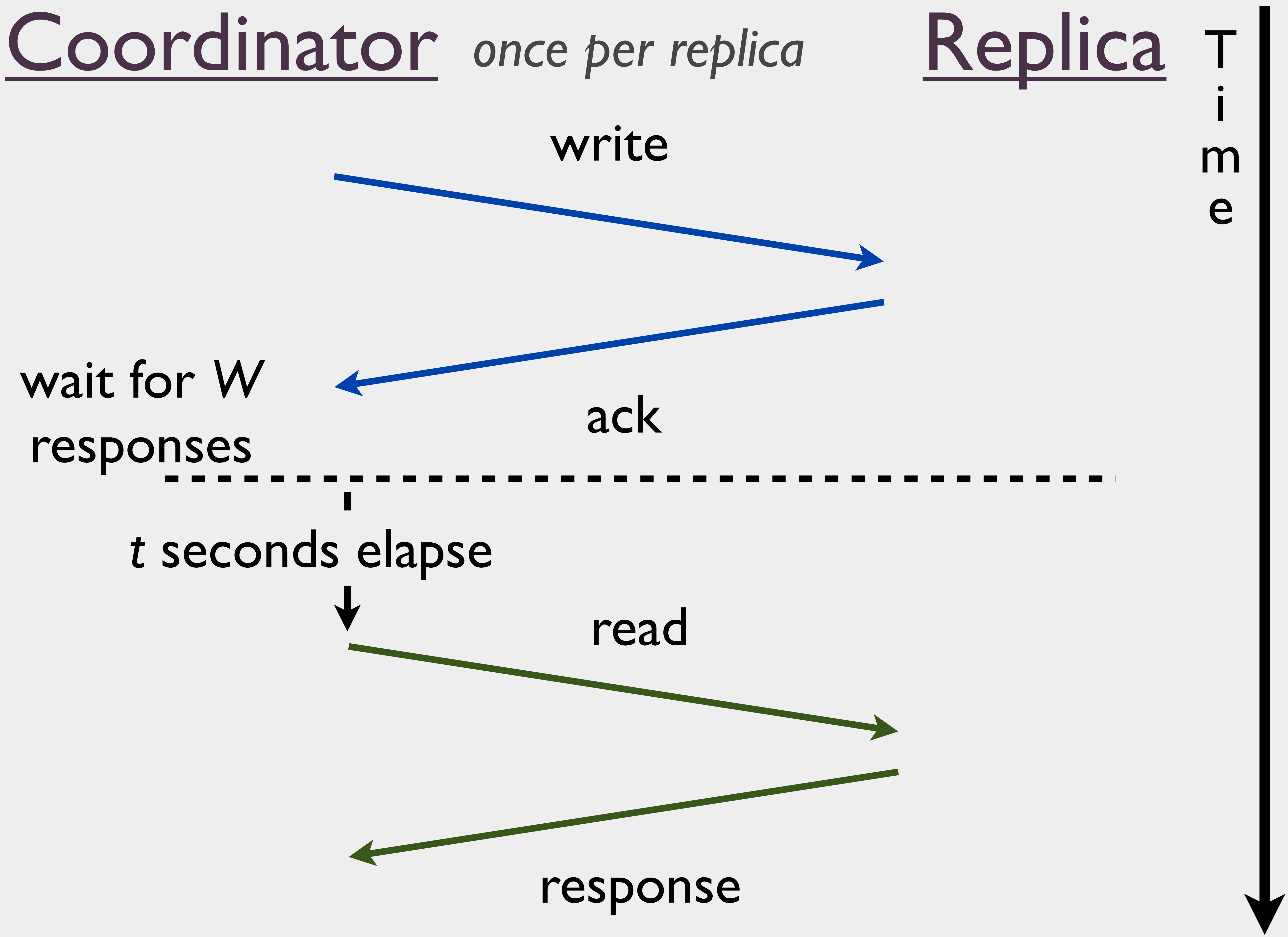

Coordinator
once per replica
Replica
write
Time

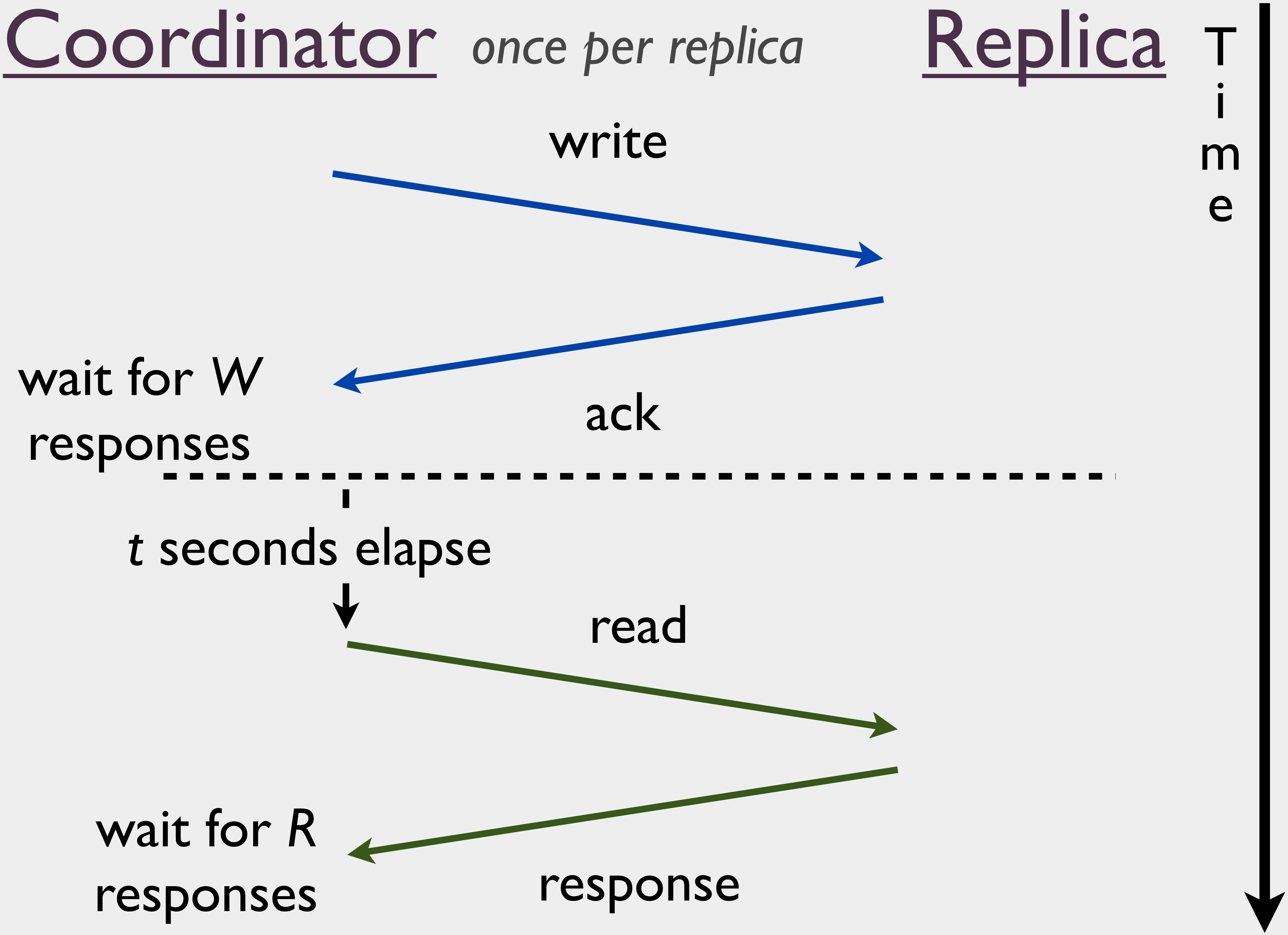

Coordinator
once per replica
Replica
Time
write
ack

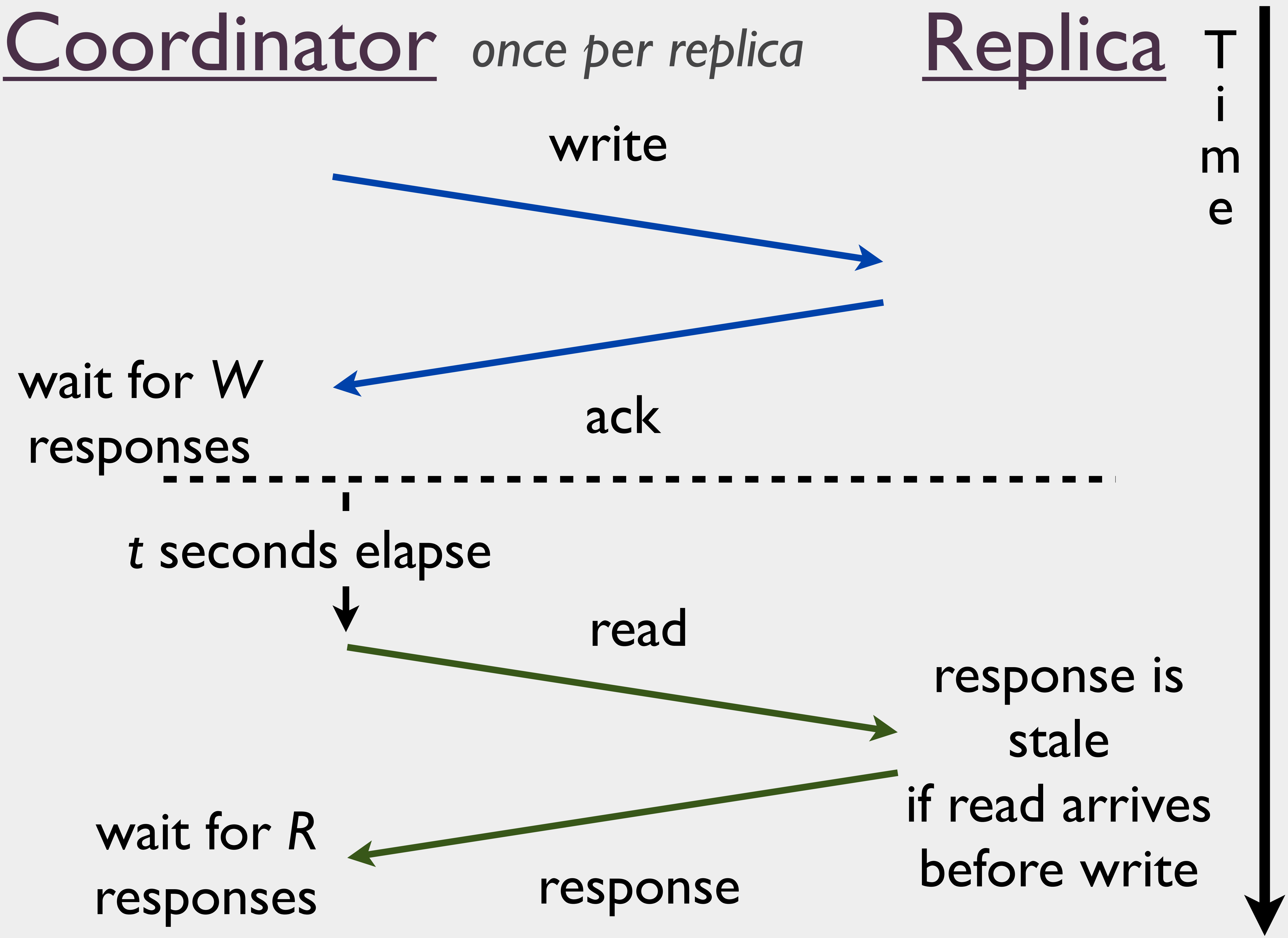

Coordinator
once per replica
Replica
Time
write
ack
wait for W responses

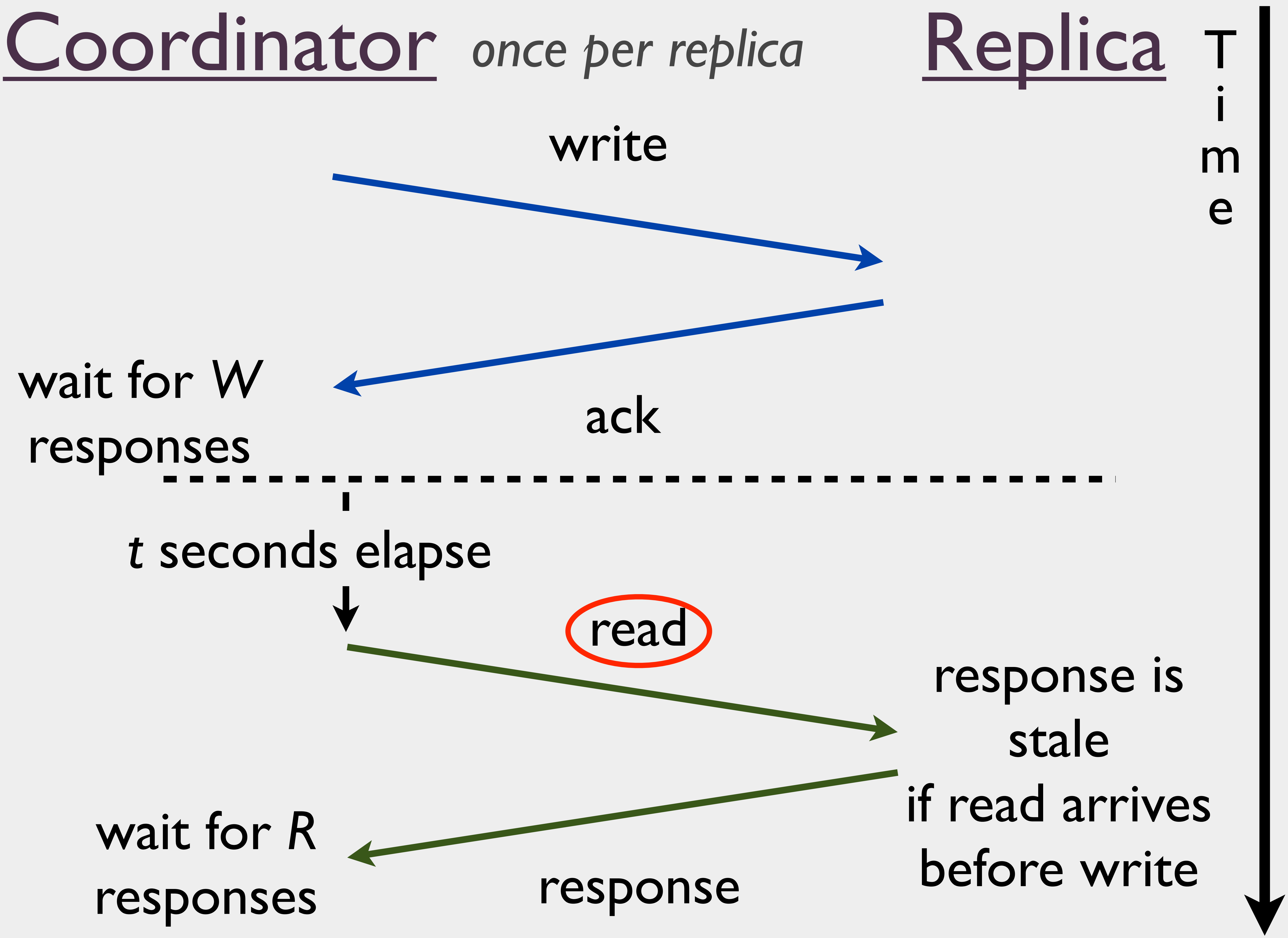

Coordinator
once per replica
Replica
Time
write
wait for W
responses
ack
t seconds elapse

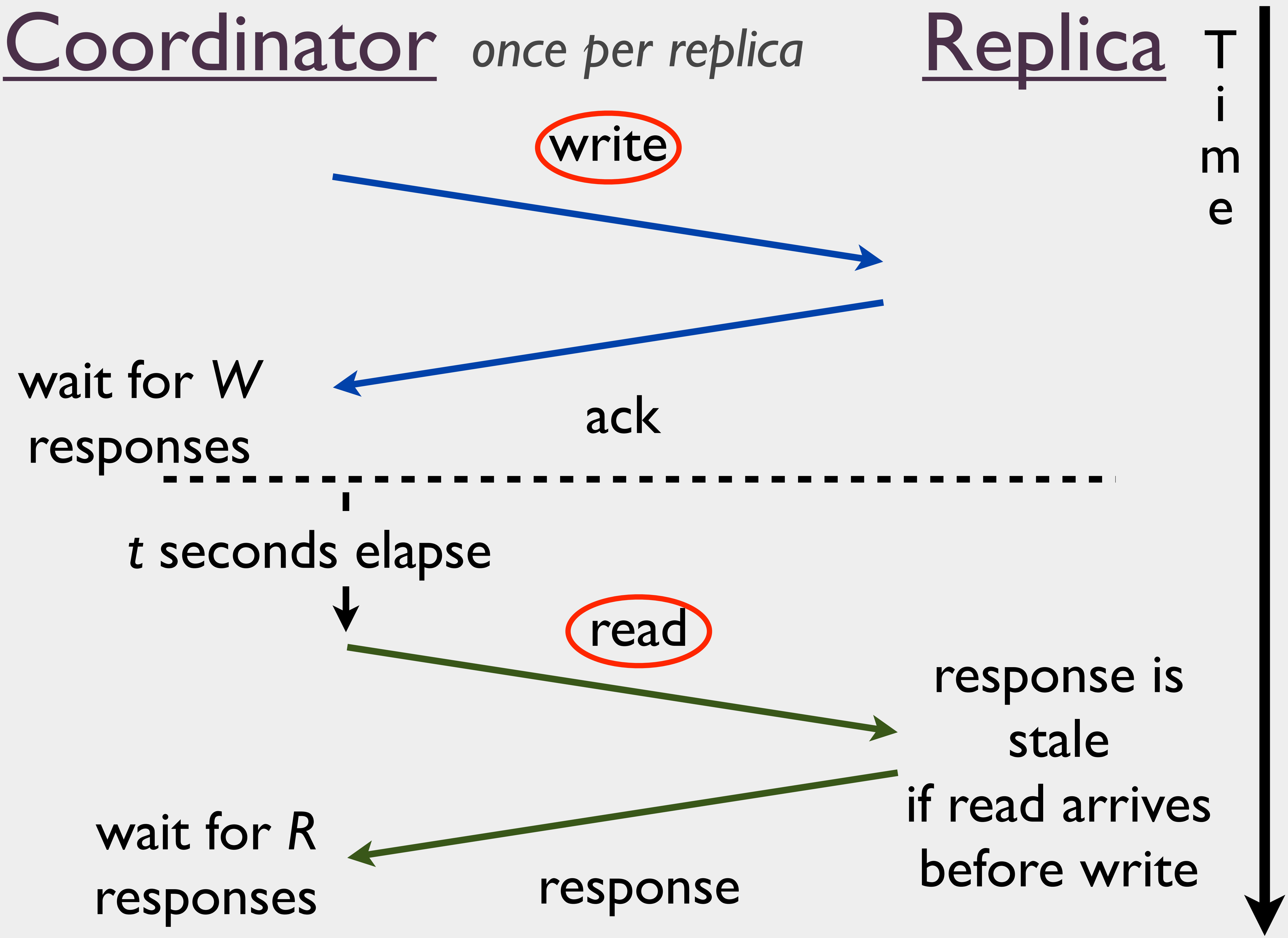

Coordinator
once per replica
Replica
Time
write
wait for W
responses
ack
t seconds elapse
read

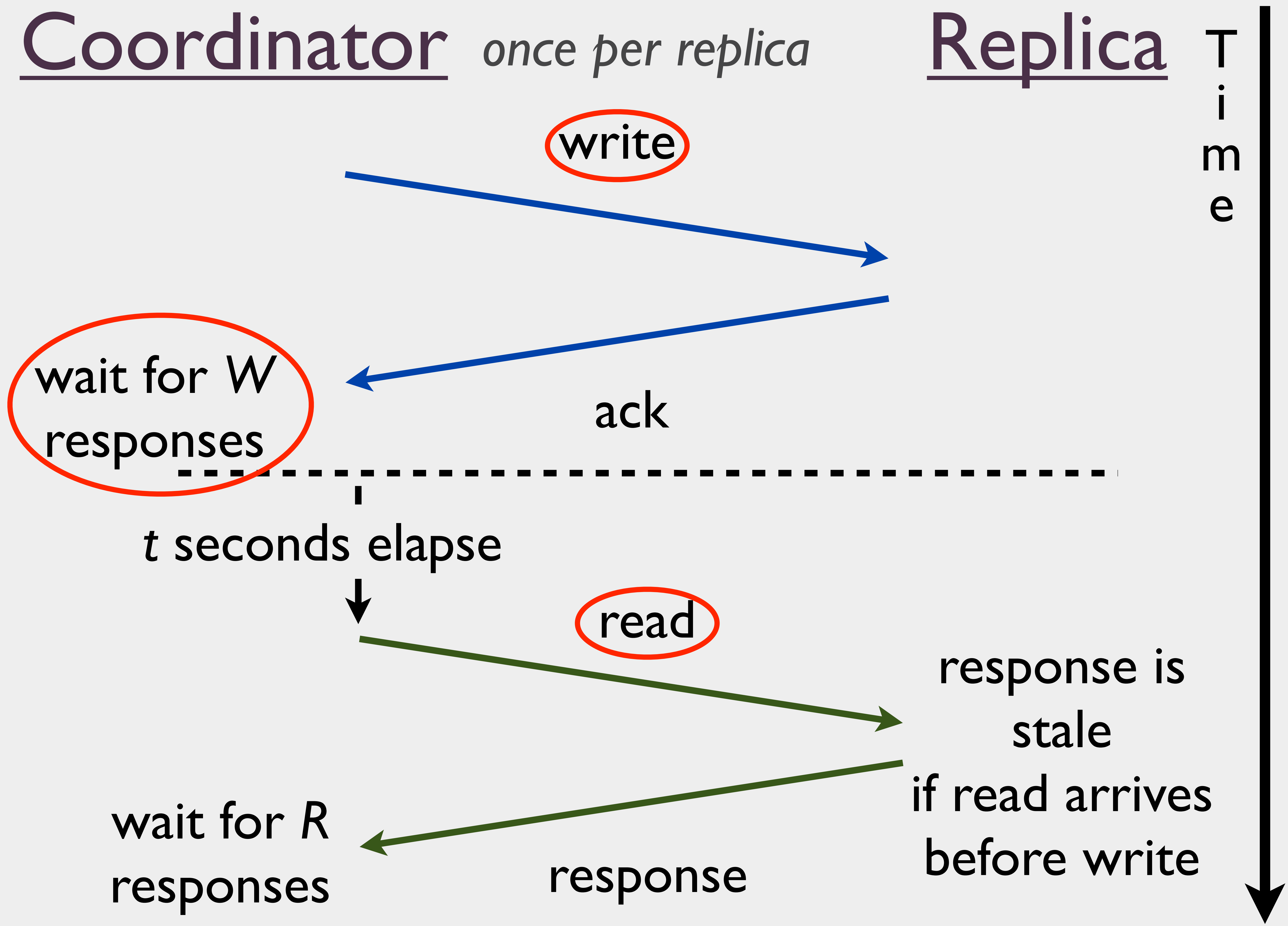

Coordinator
once per replica
Replica
Time
write
wait for W responses
ack
t seconds elapse
read
response

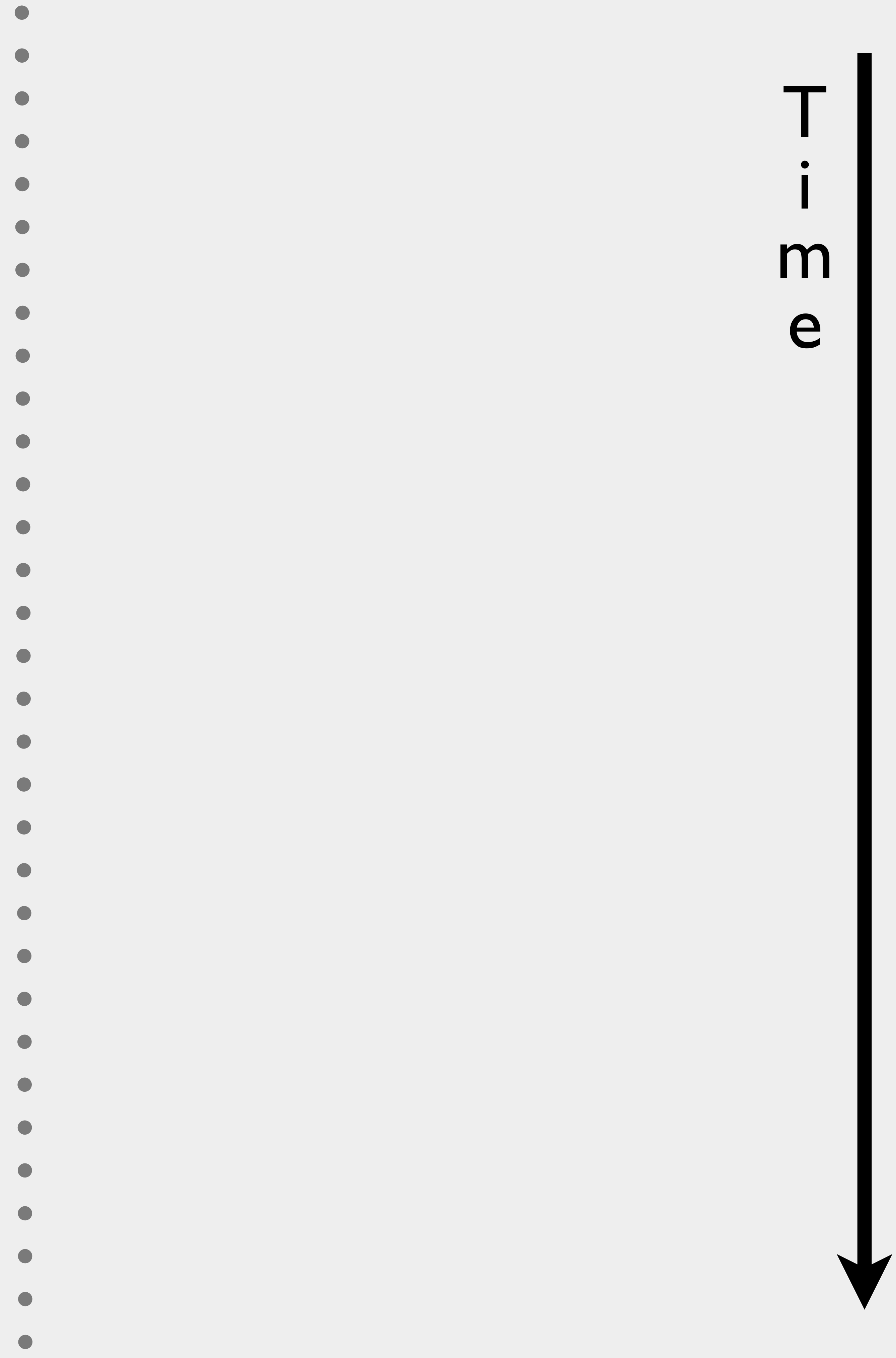

Coordinator
once per replica
Replica
Time
write
wait for W responses
ack
t seconds elapse
read
wait for R responses
response


Coordinator
once per replica
Replica
Time
write
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response

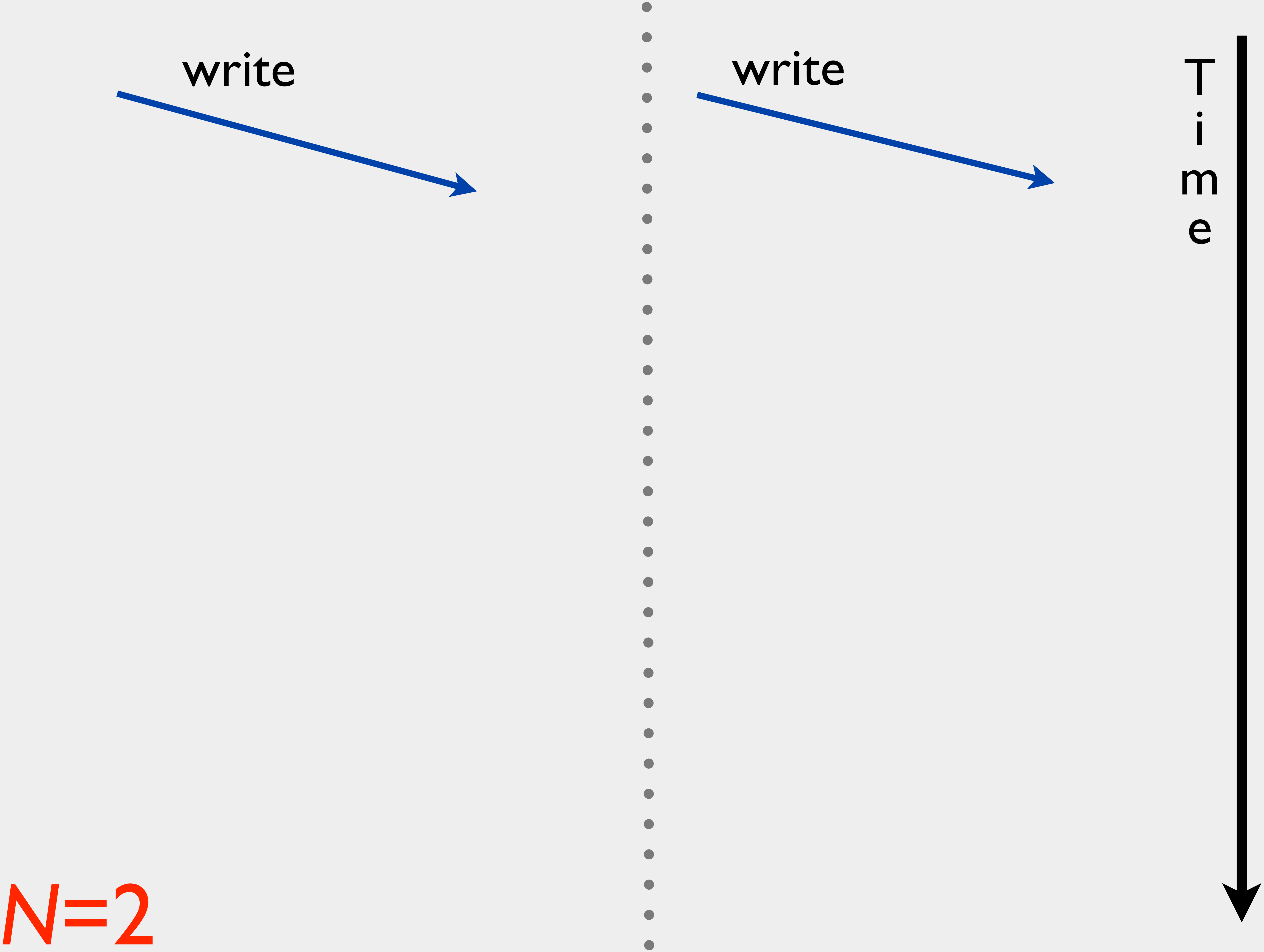

Coordinator
once per replica
Replica
Time
write
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response

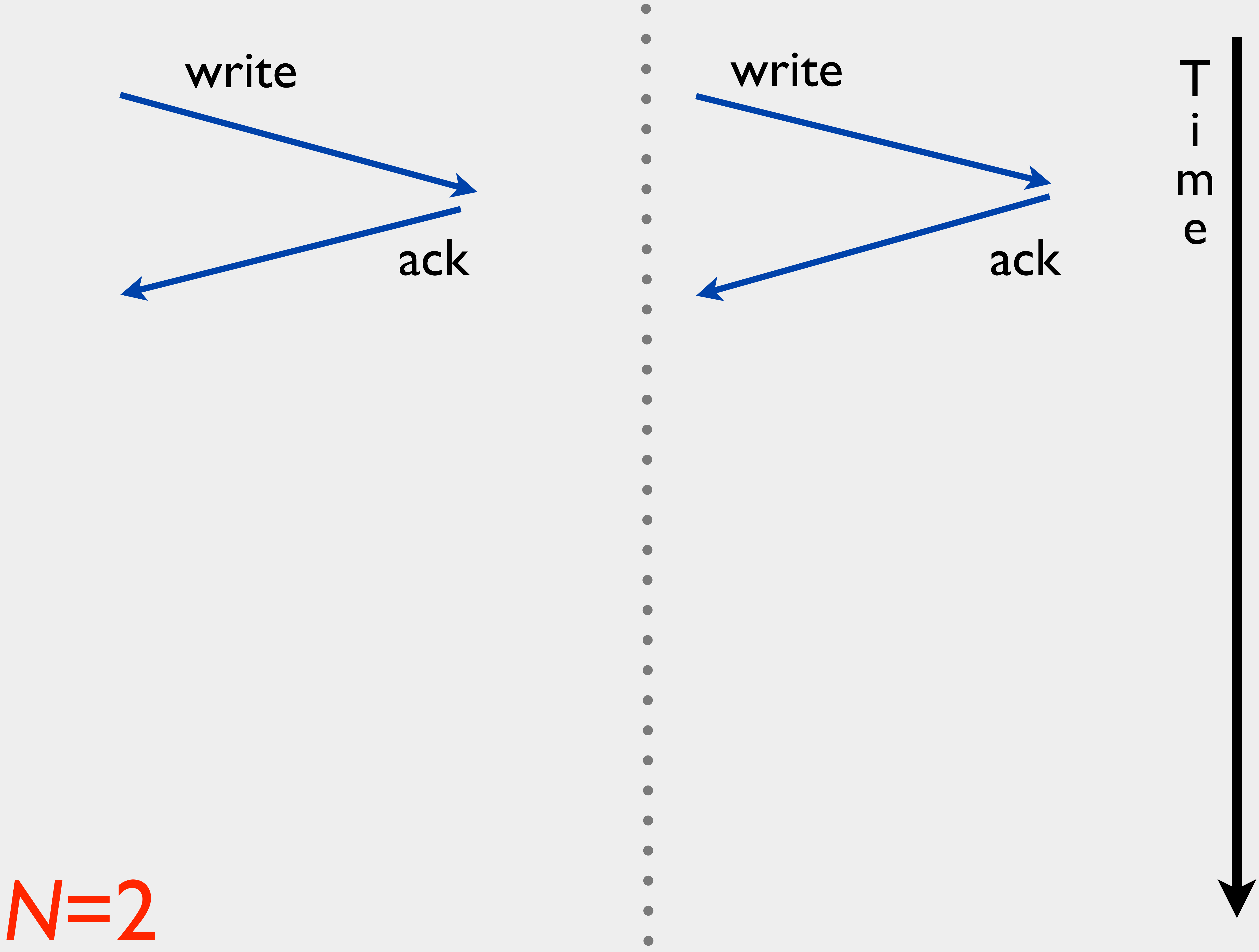

Coordinator
once per replica
Replica
Time
write
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response

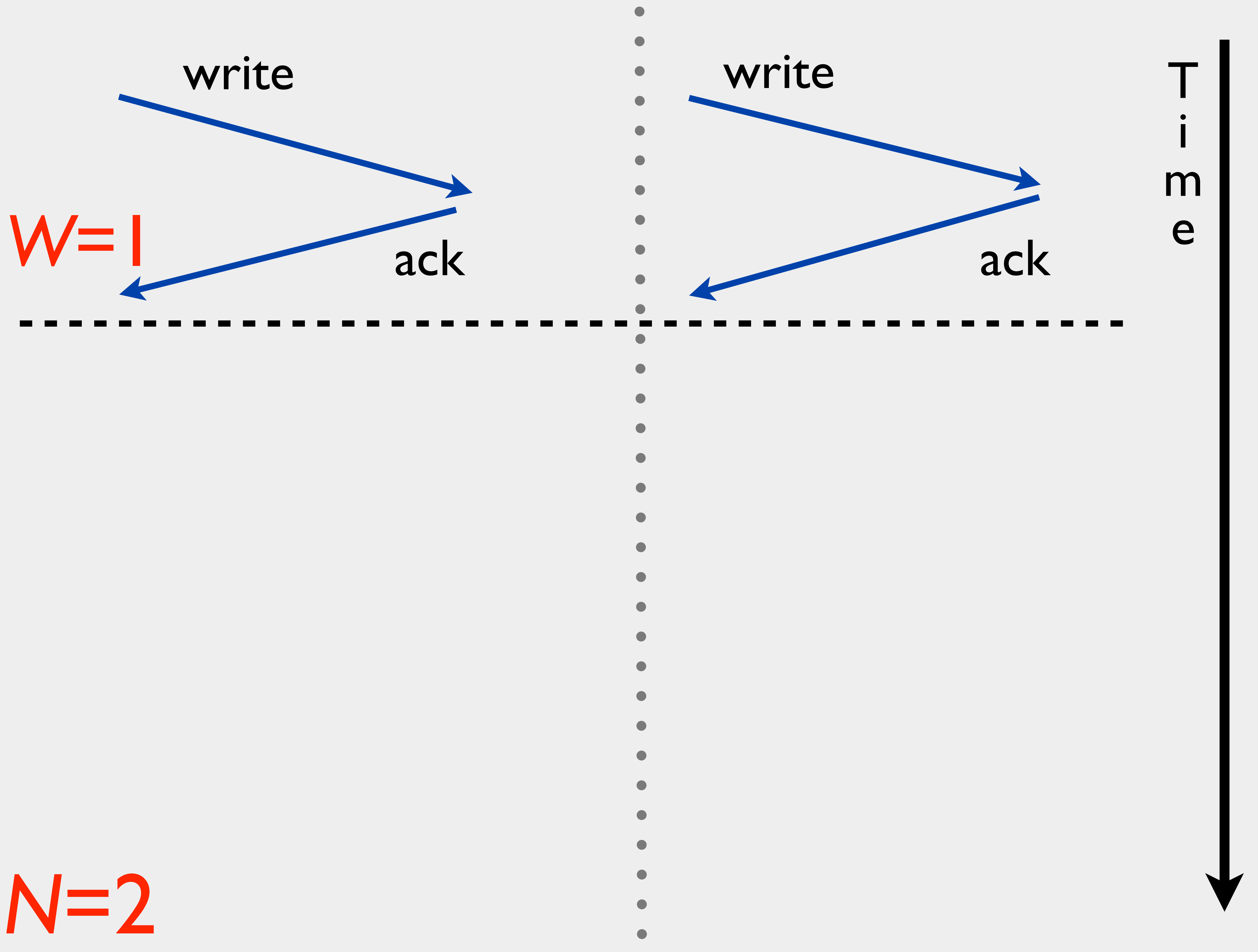

Coordinator
once per replica
Replica
Time
write
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response

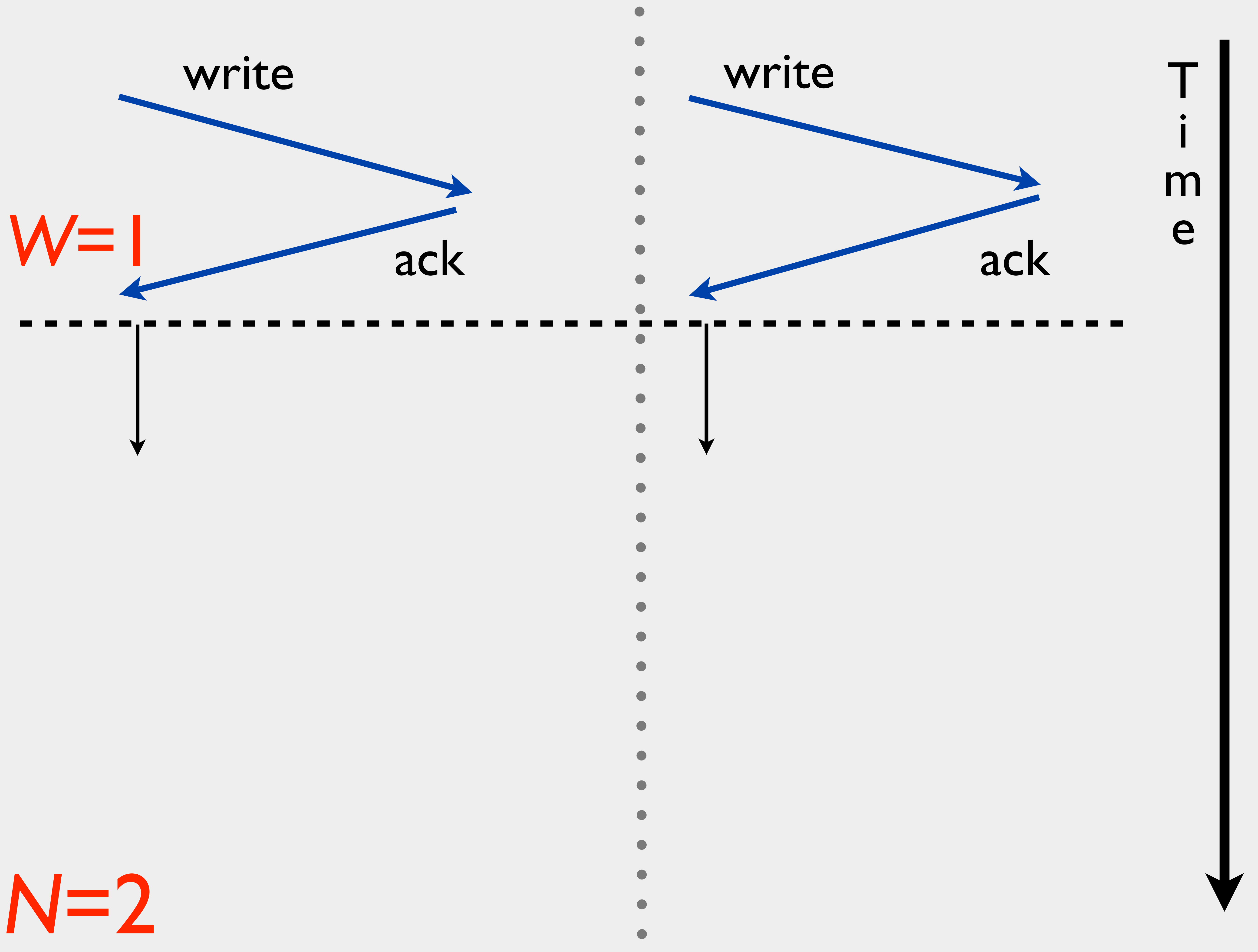


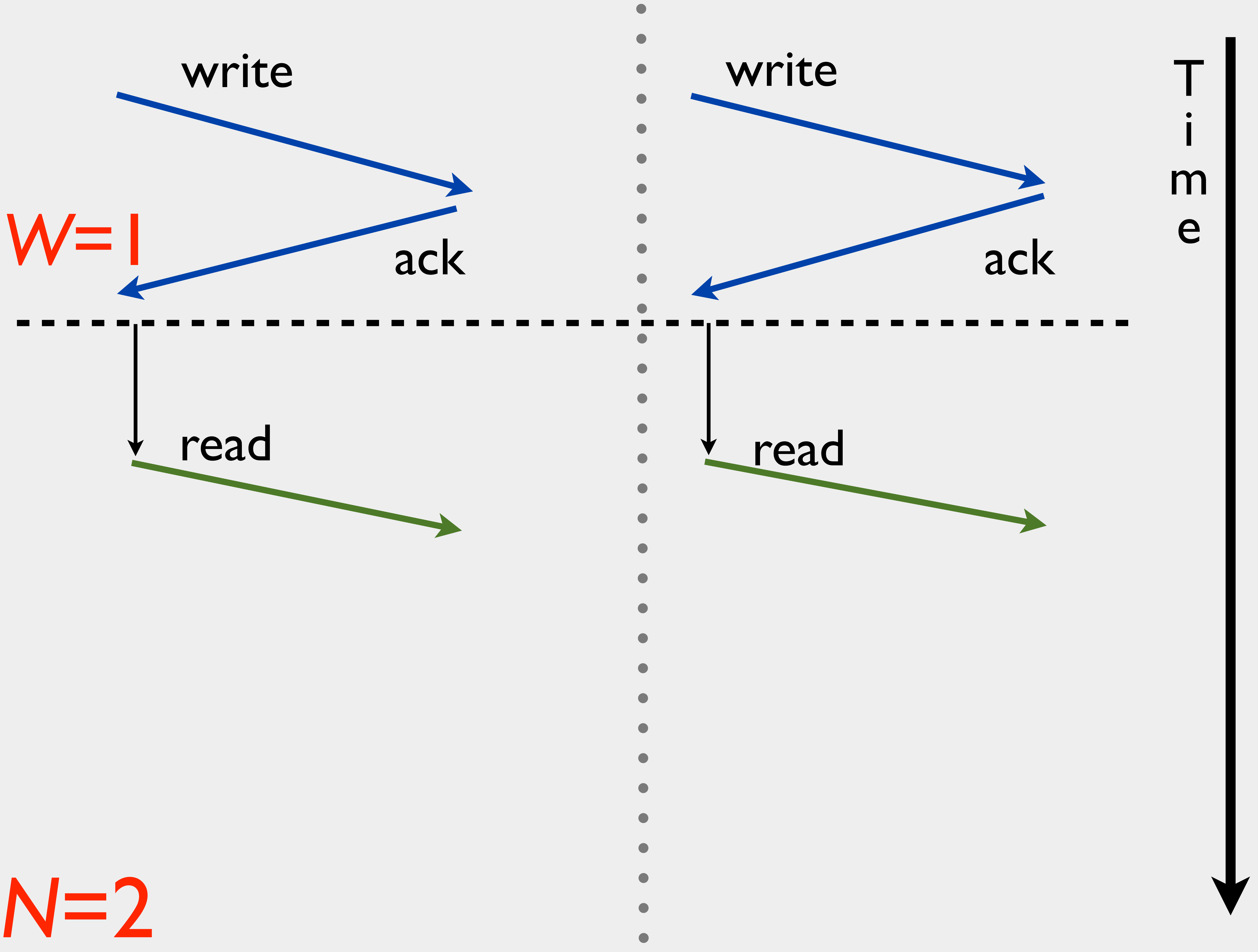
$N=2$


write
write
Time
N=2

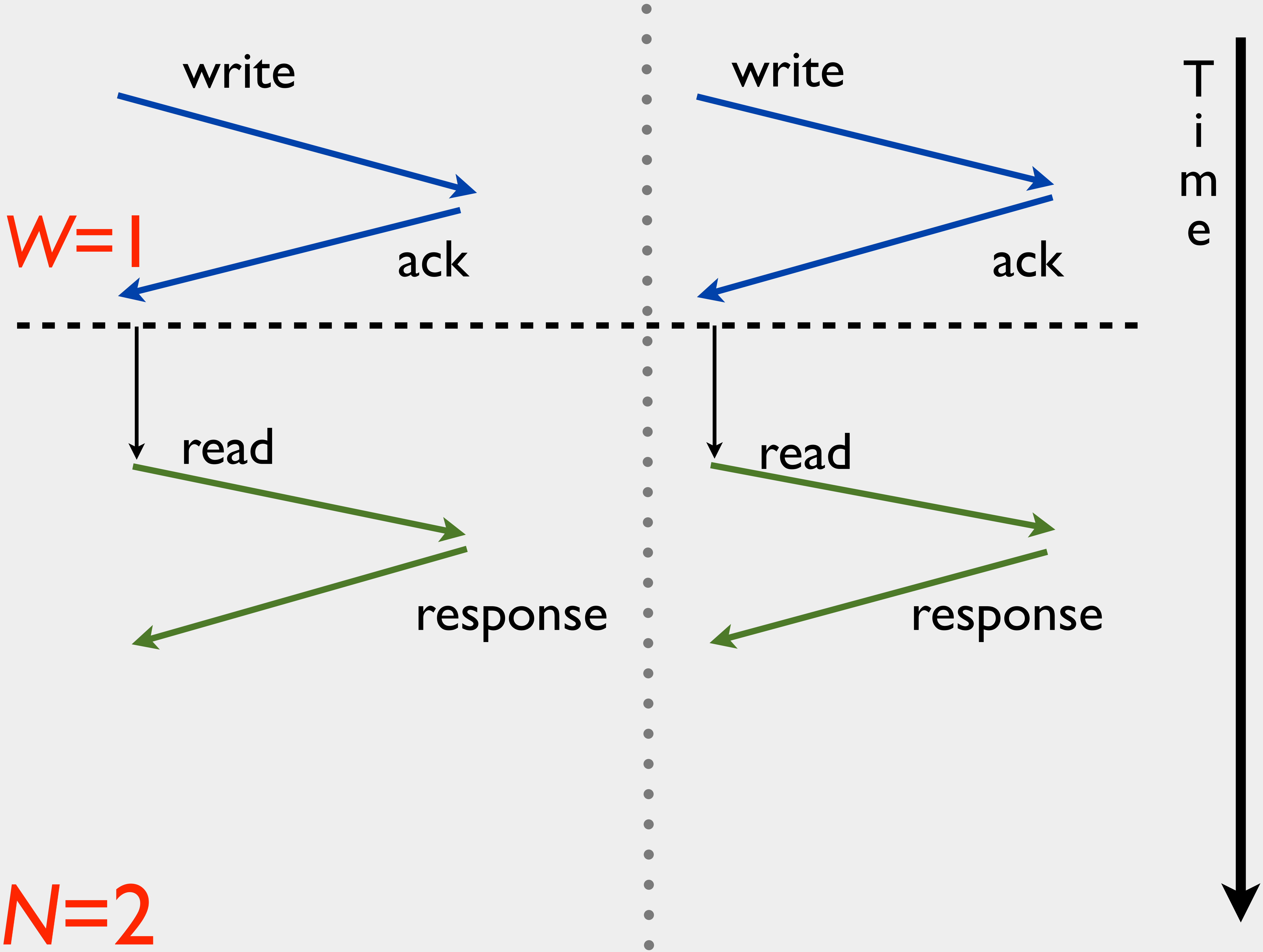

write
ack
write
ack
Time
N=2

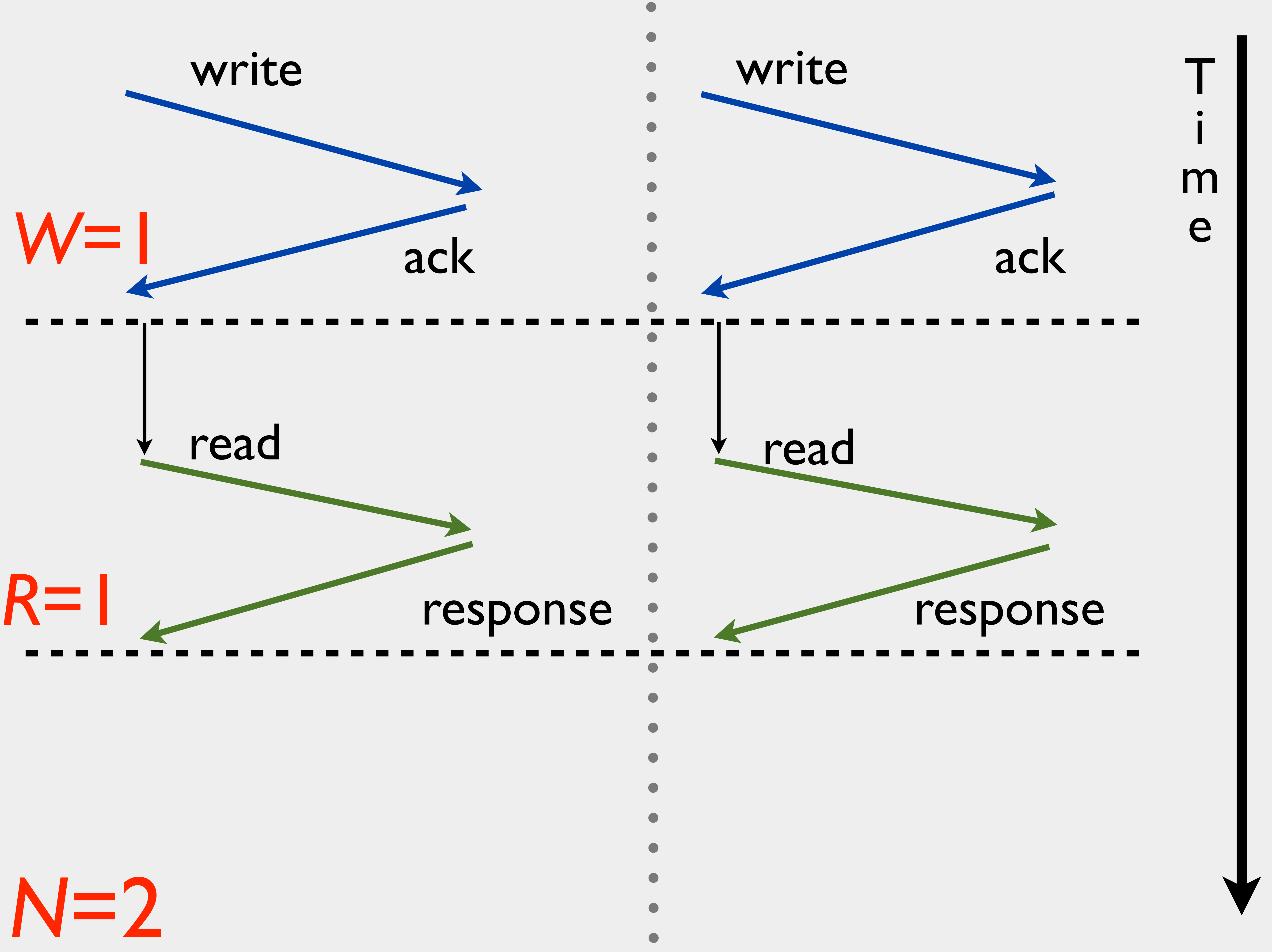

write
write
W=1
ack
ack
Time
N=2

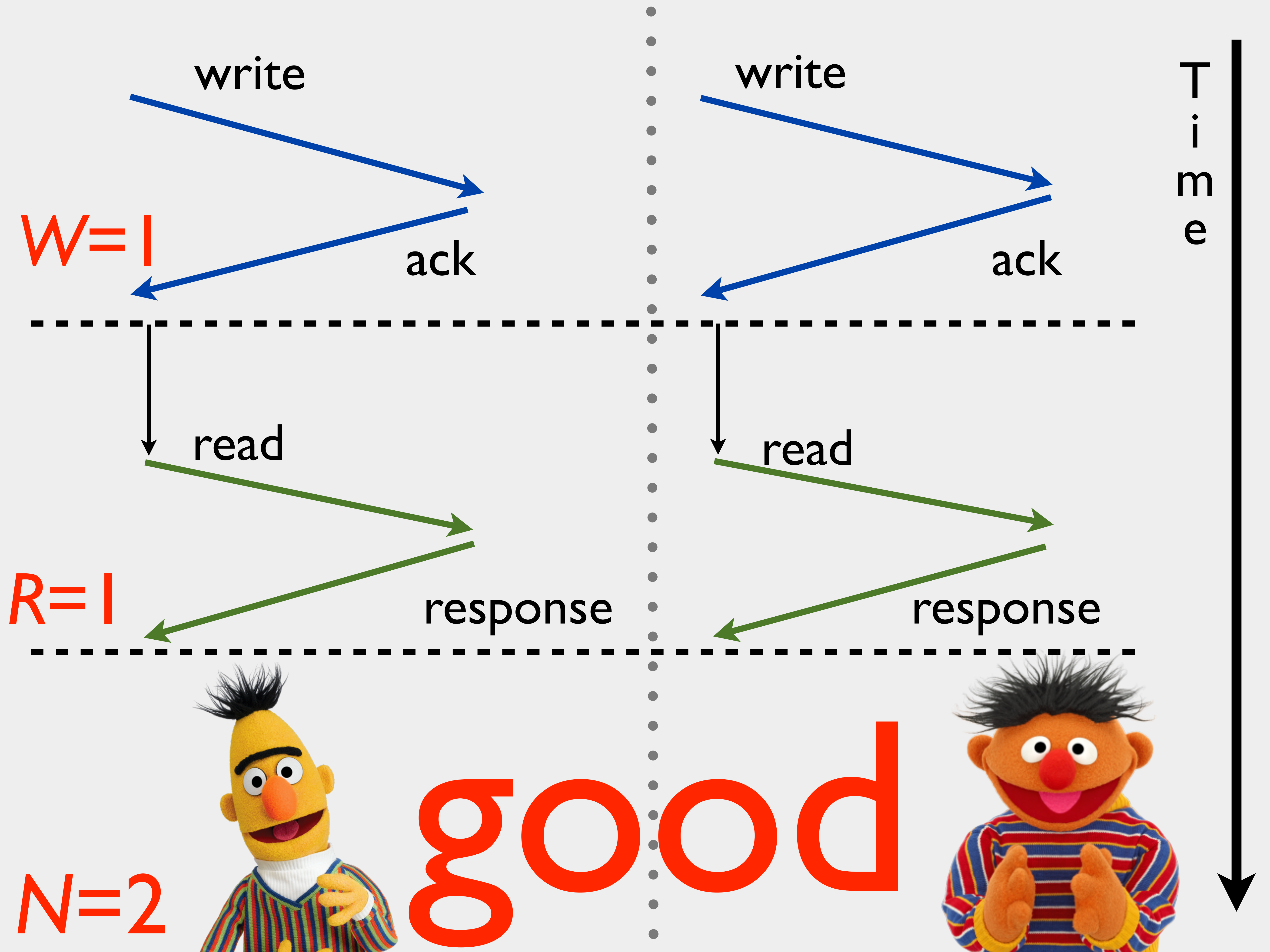

write
write
W=1
ack
ack
Time
N=2

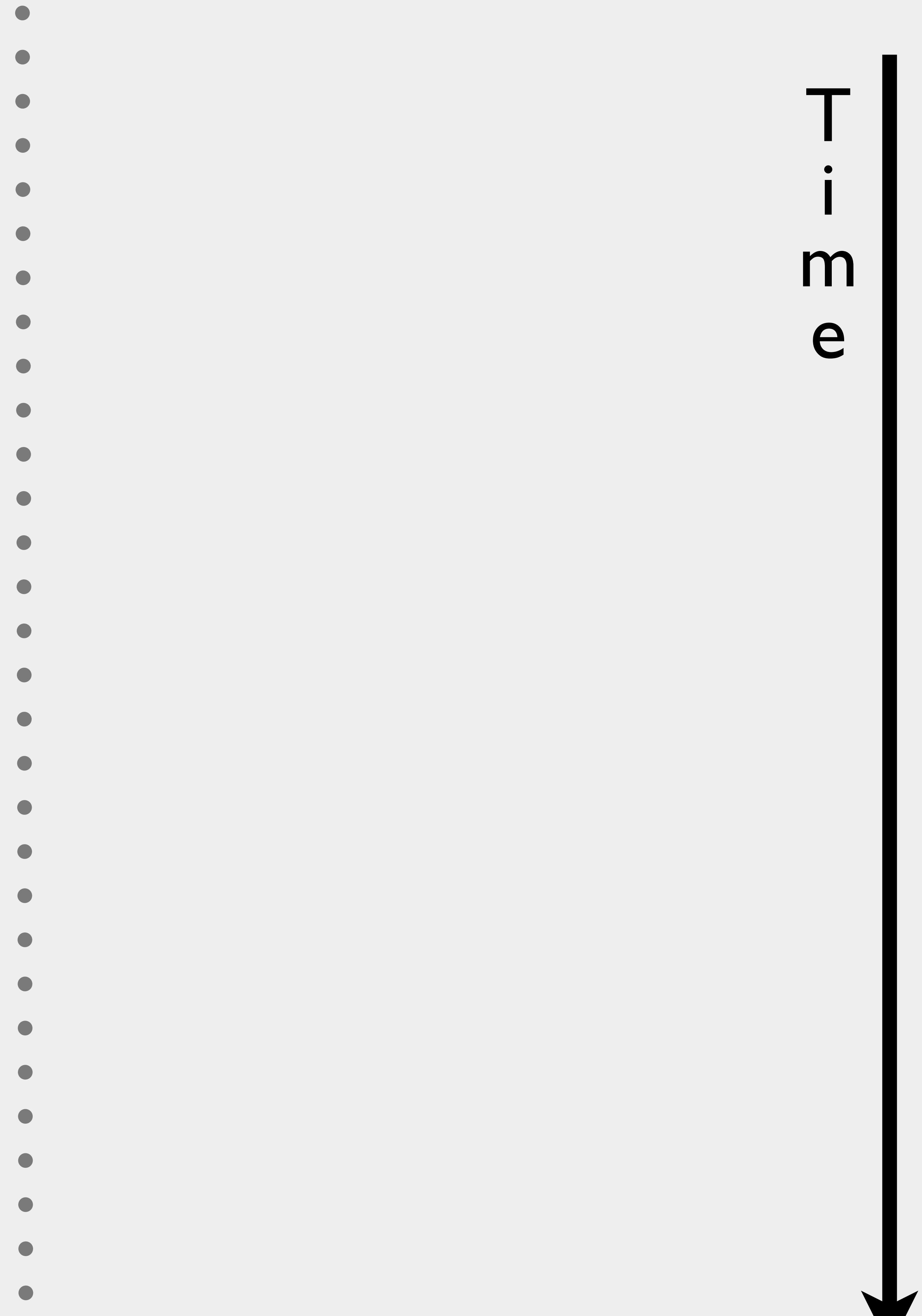


write
write
W=1
ack
ack
Time
read
read
N=2

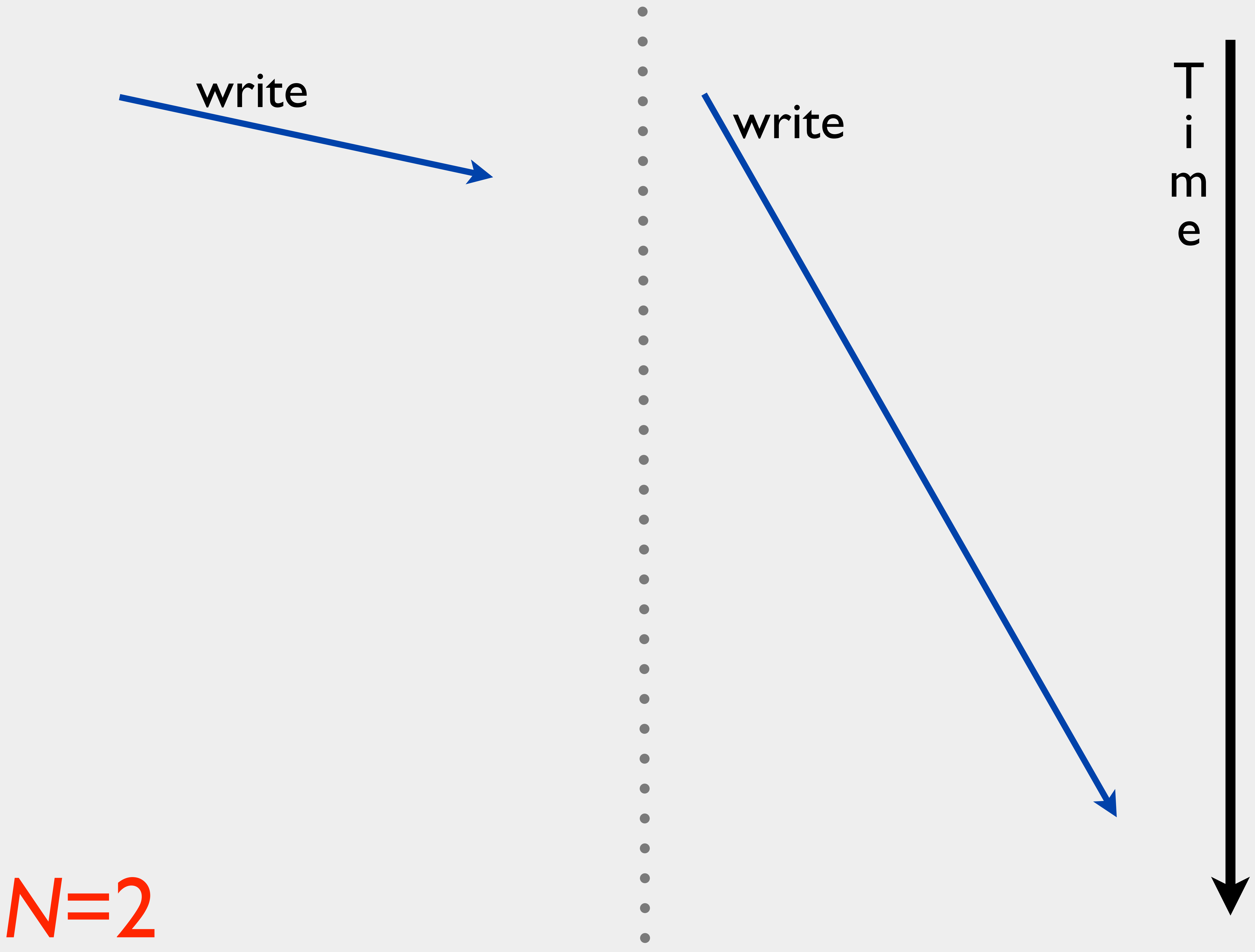

write
write
Time
W=1
ack
ack
read
read
response
response
N=2

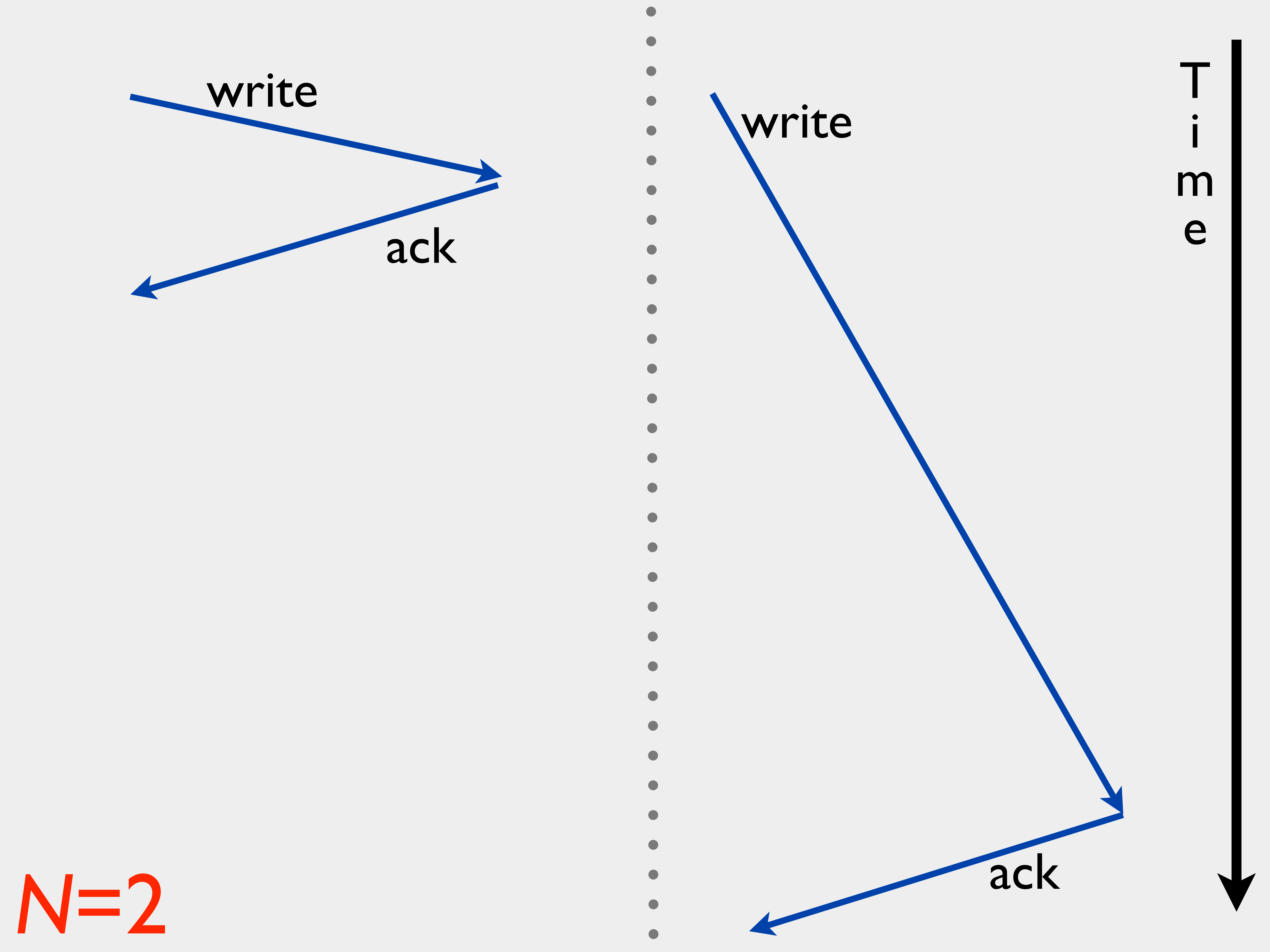

write
ack
W=1
write
ack
Time
read
response
R=1
read
response
N=2

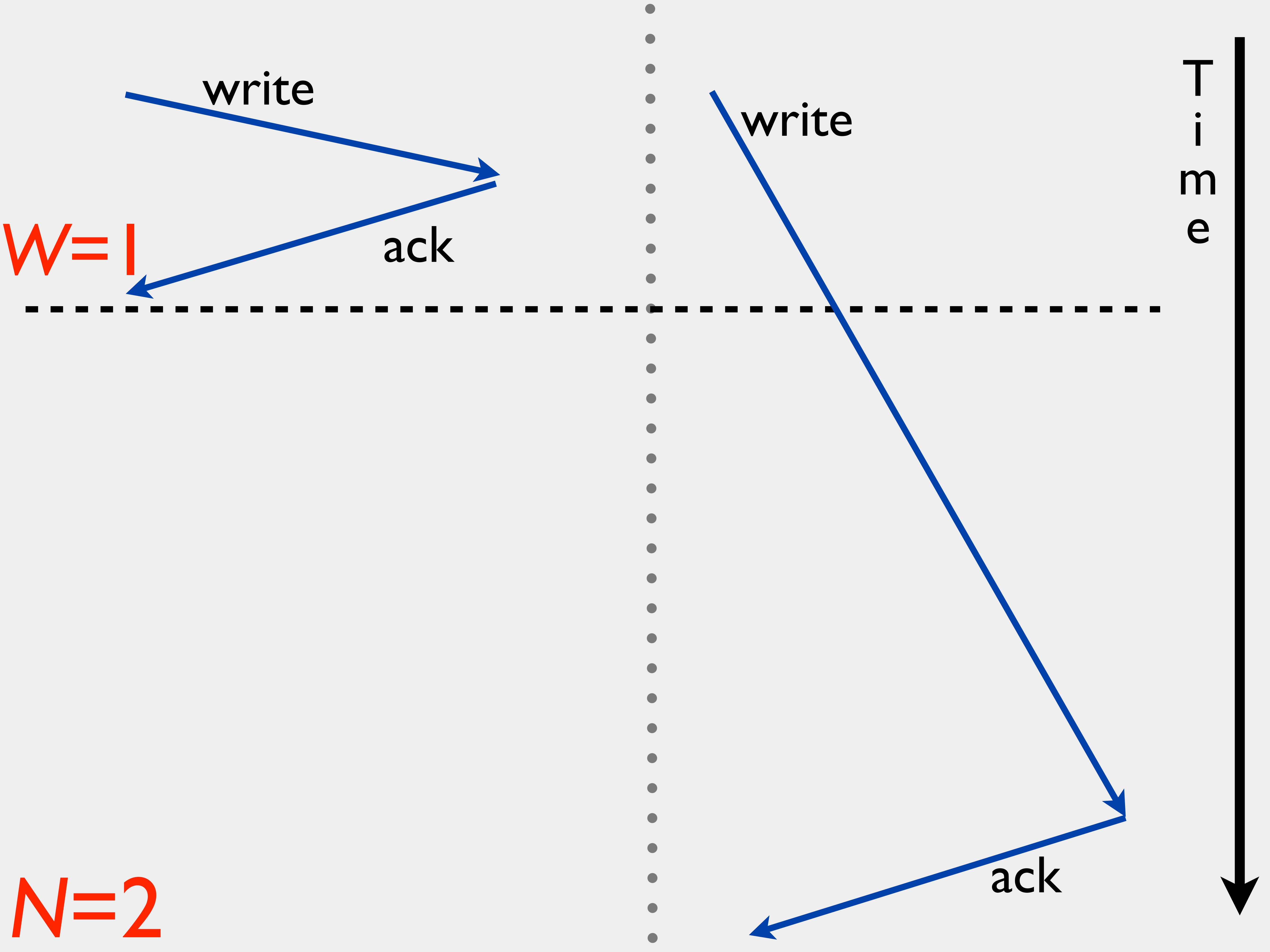

write
ack
W=1
read
response
R=1
write
ack
read
response
Time
N=2
good

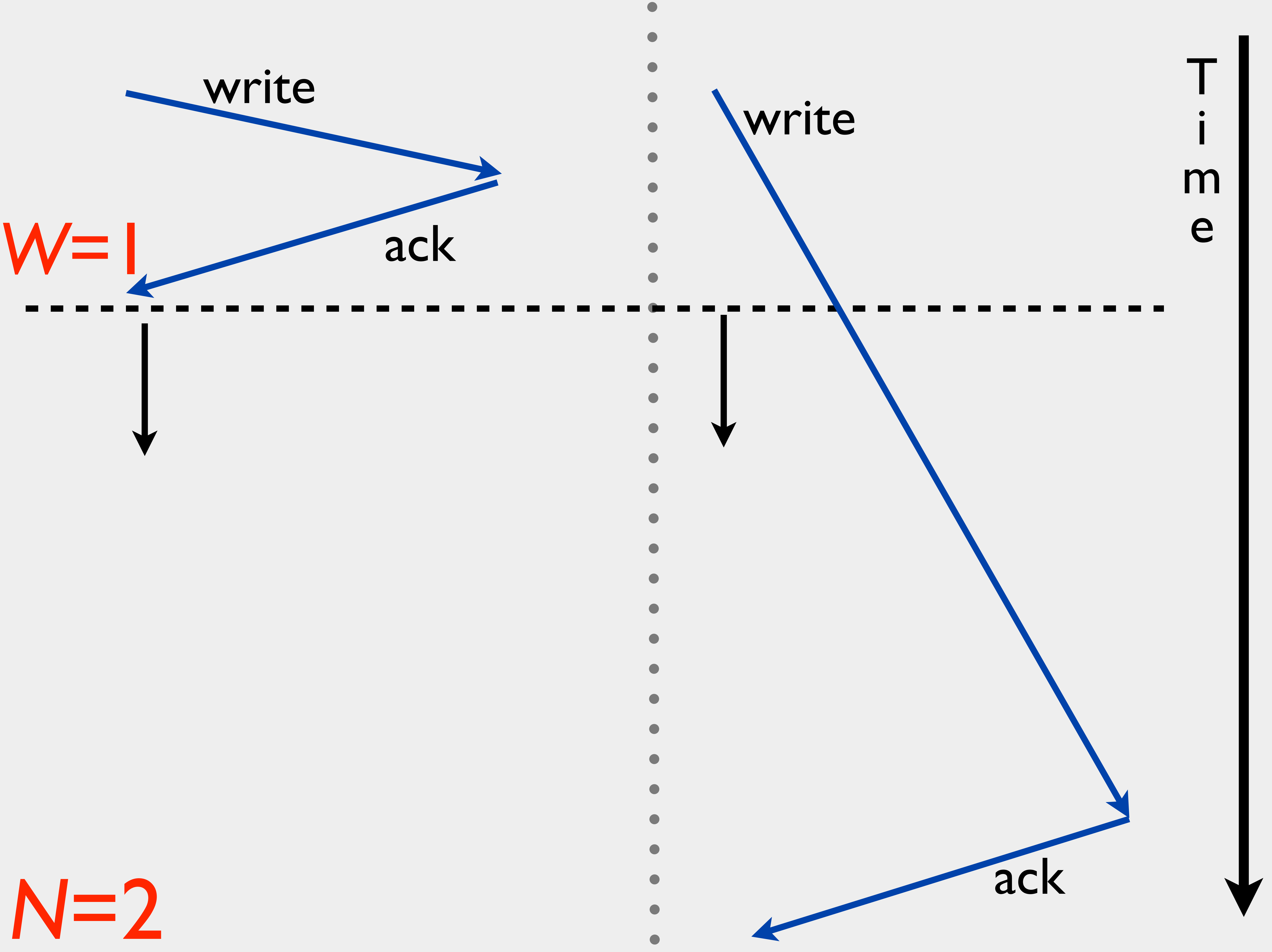


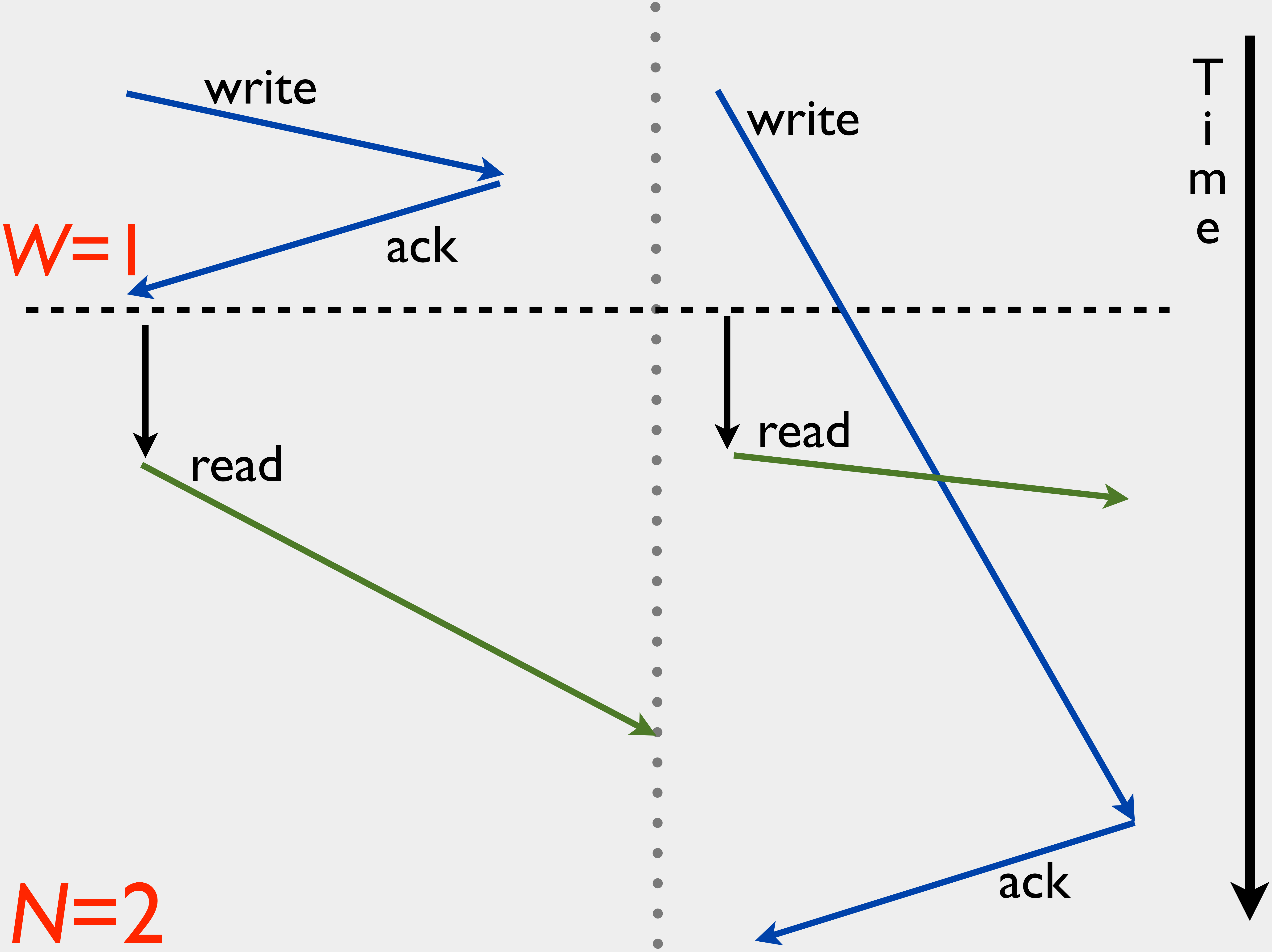
$N=2$

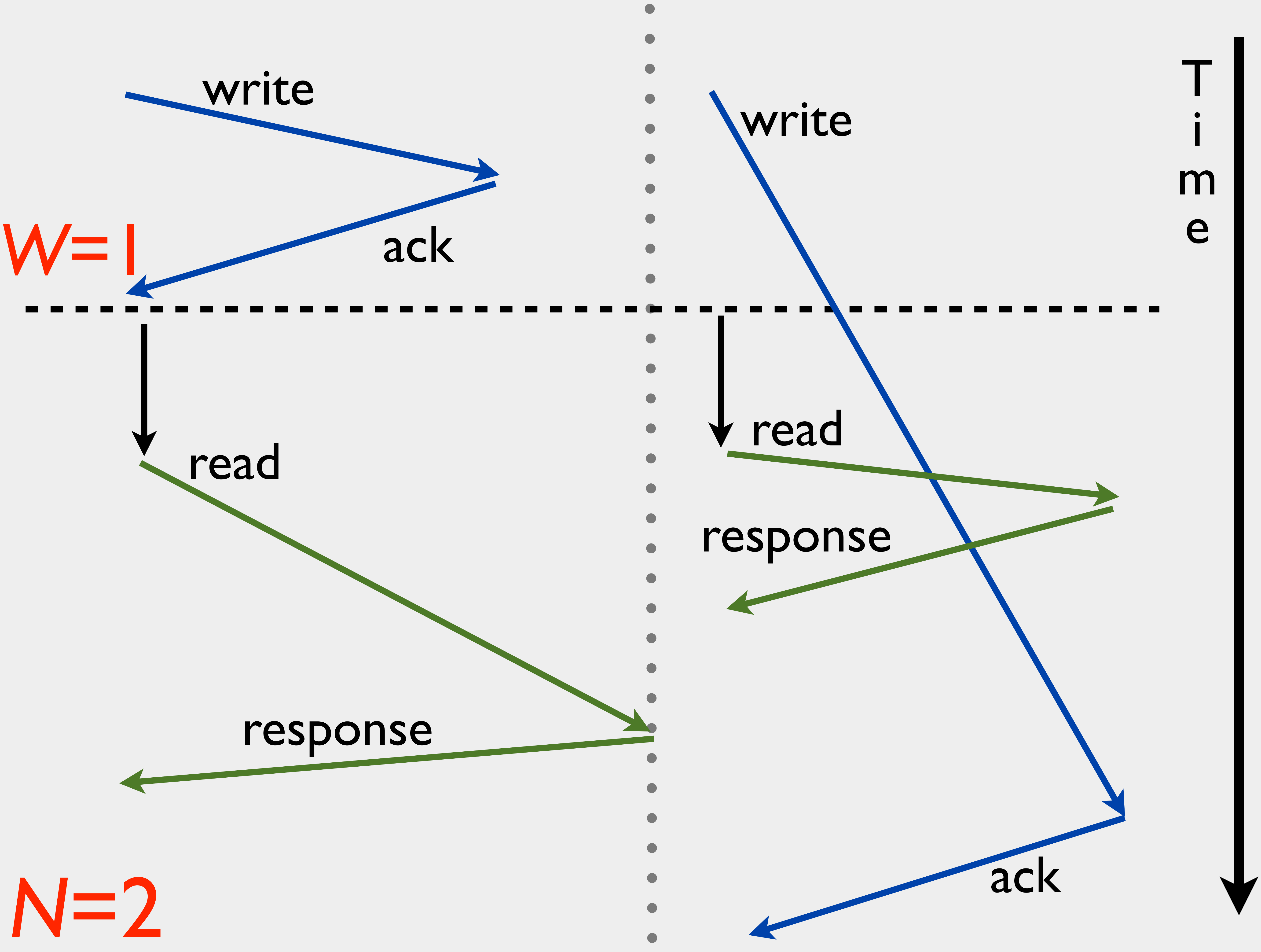

write
write
Time
N=2

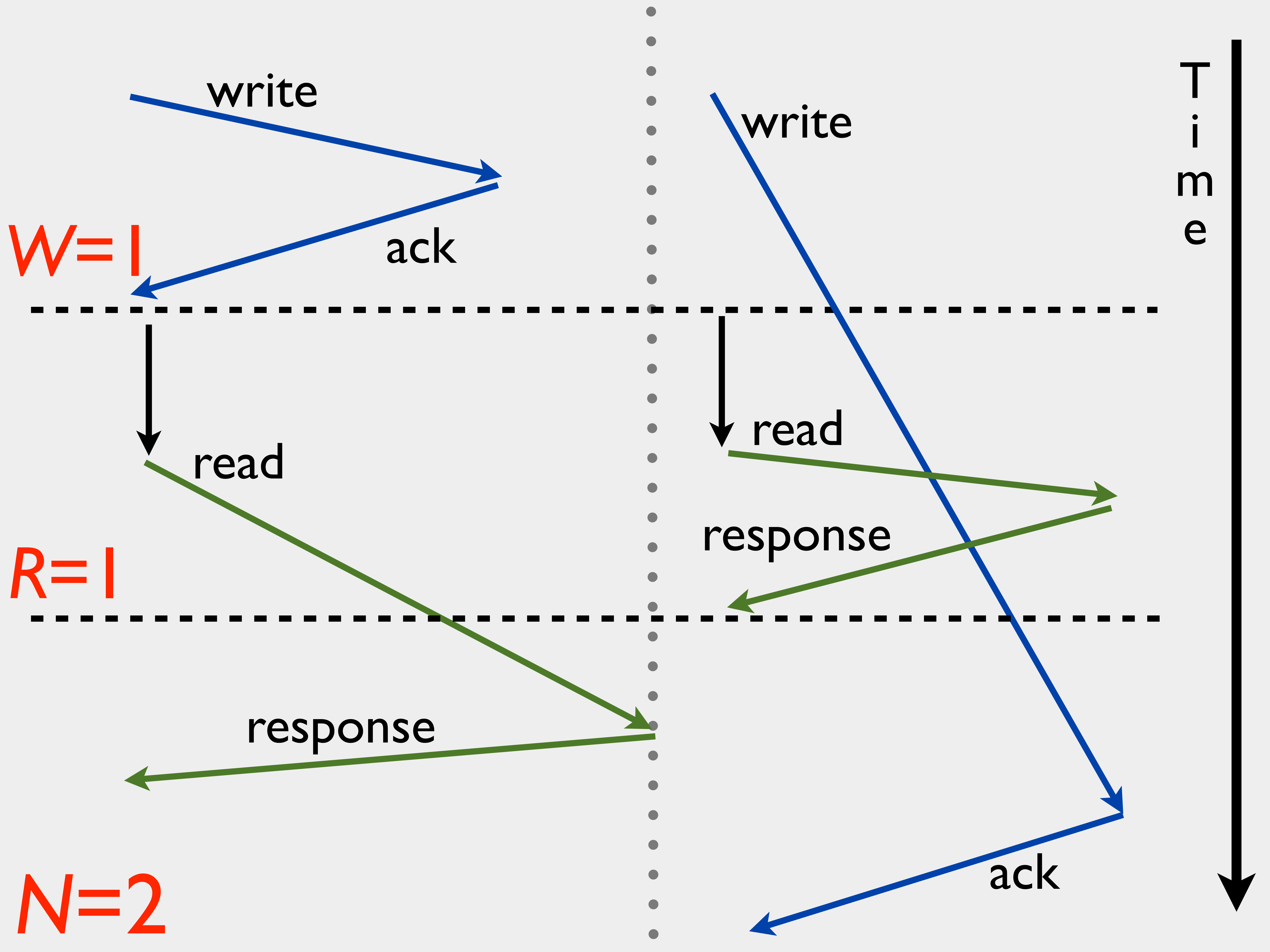

write
ack
write
ack
Time
N=2

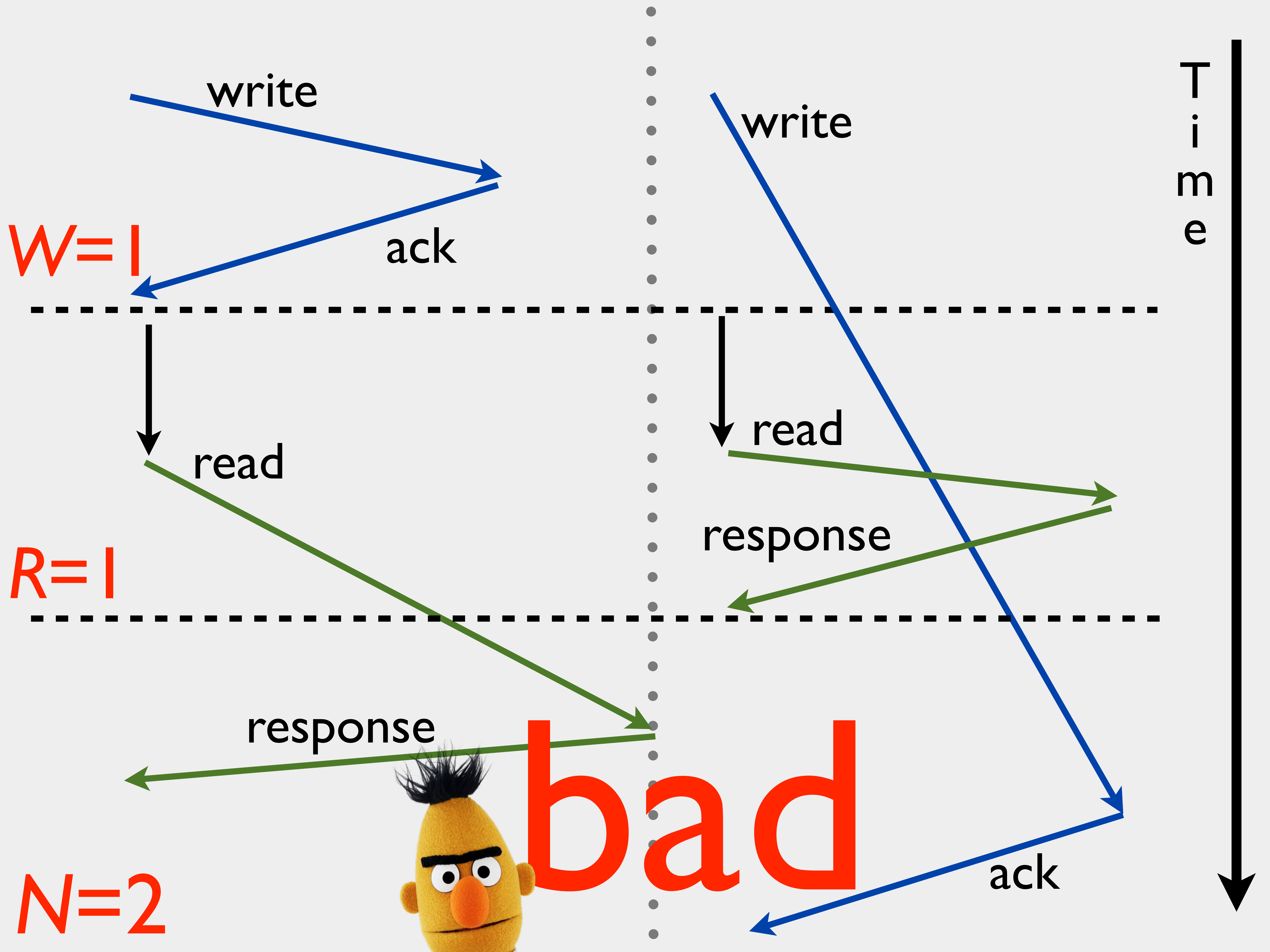

write
ack
W=1
write
Time
ack
N=2

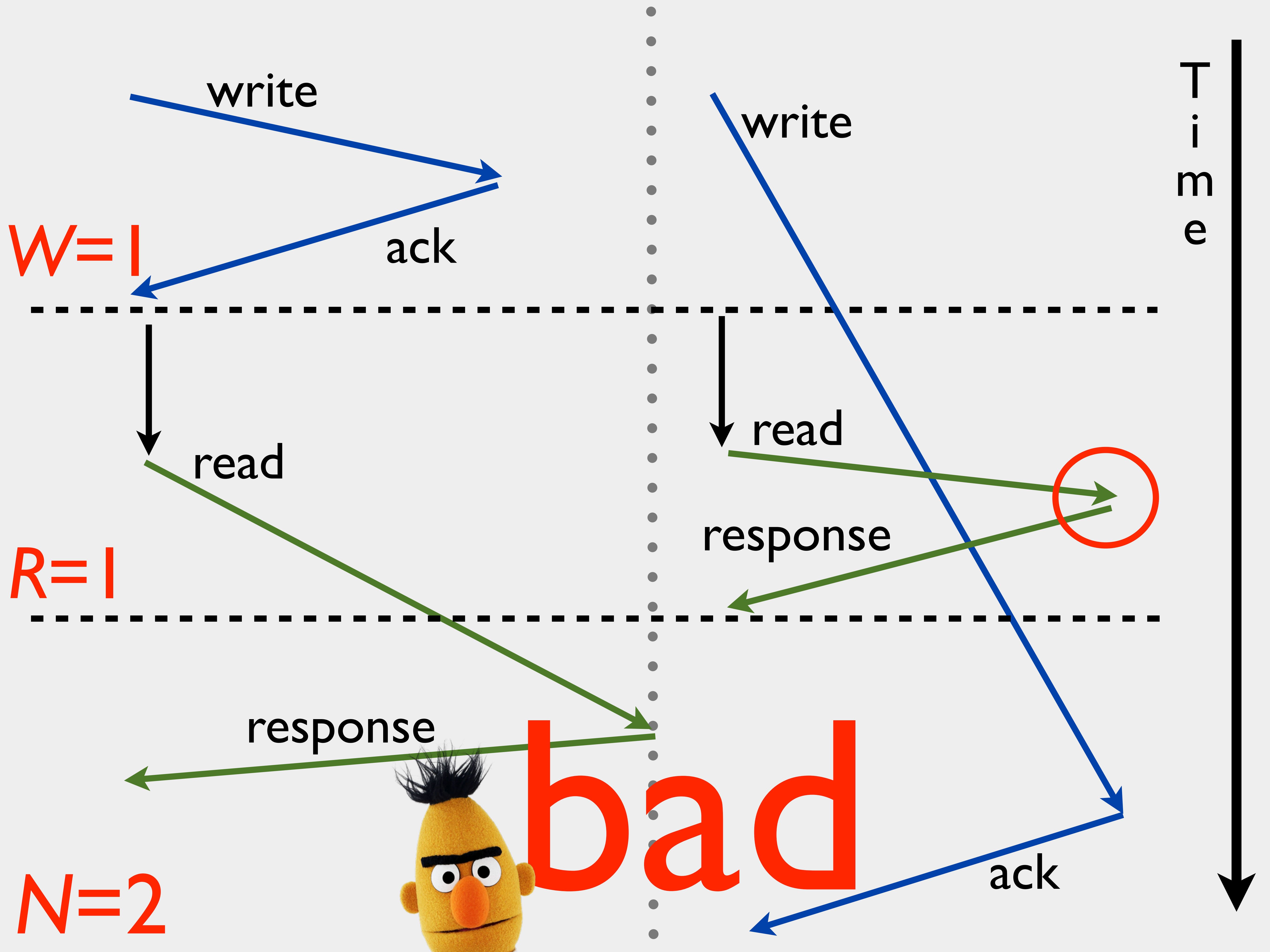

write
ack
W=1
write
ack
N=2
Time

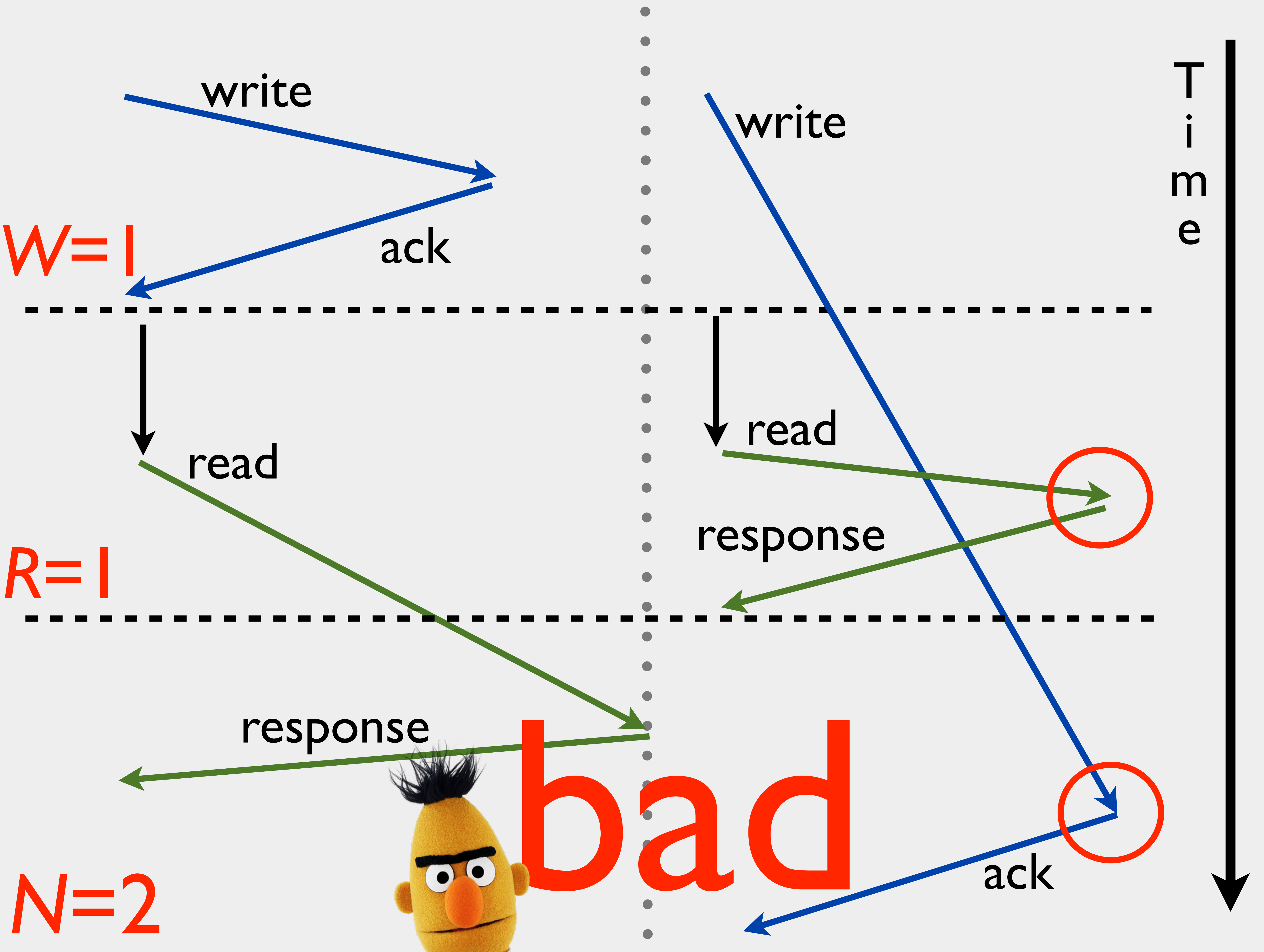

write
ack
W=1
write
T
i
m
e
read
read
ack
N=2


write
ack
W=1
write
Time
read
read
response
response
ack
N=2

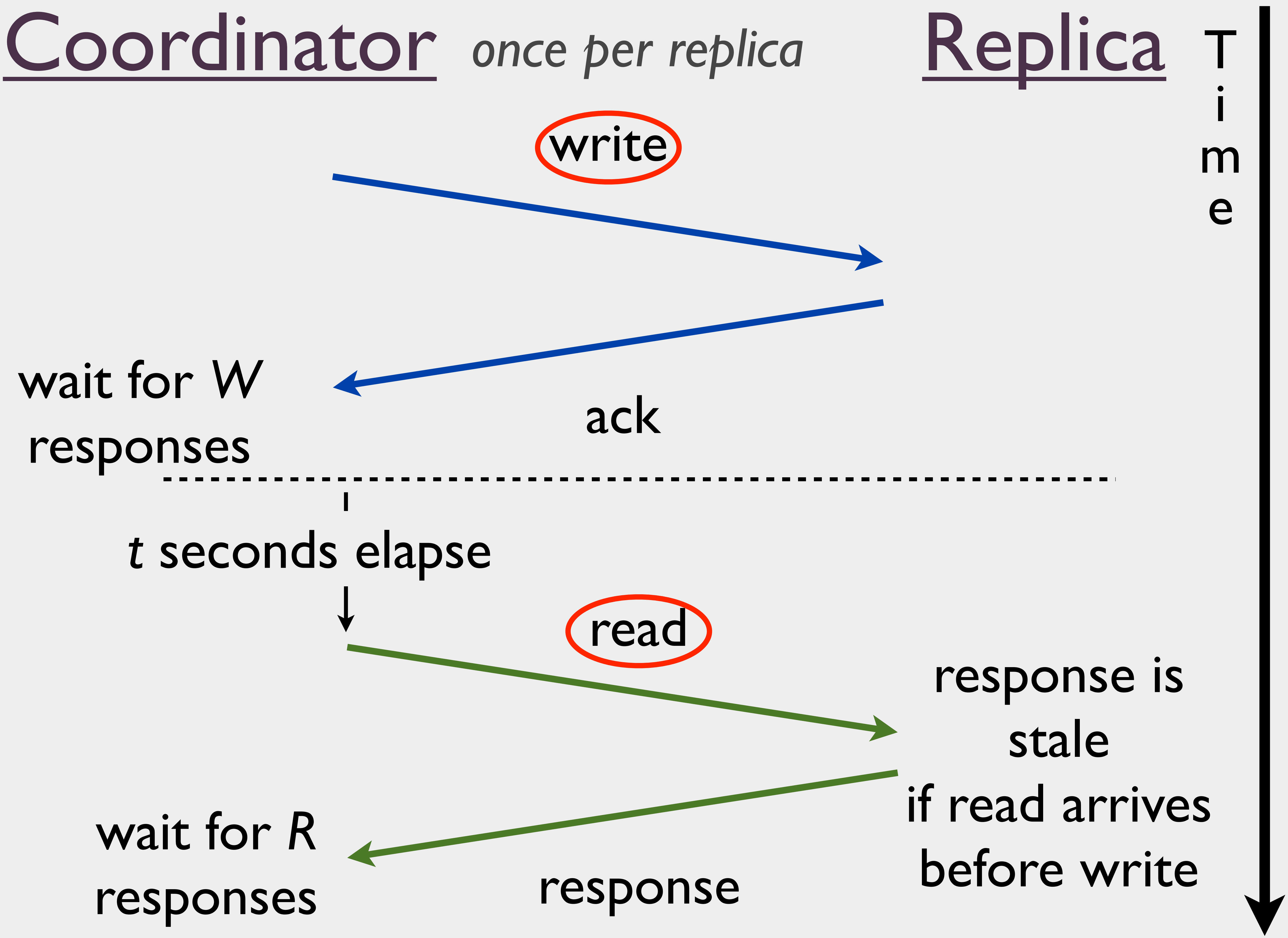

write
ack
W=1
write
Time
read
read
response
R=1
response
ack
N=2


write
ack
W=1
write
Time
read
read
response
R=1
response
bad
ack
N=2


write
ack
W=1
write
Time
read
read
response
R=1
response
bad
ack
N=2



write
ack
W=1
write
Time
read
read
response
R=1
response
bad
ack
N=2


Coordinator
once per replica
Replica
Time
write
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response

R1 (“key”, 2)

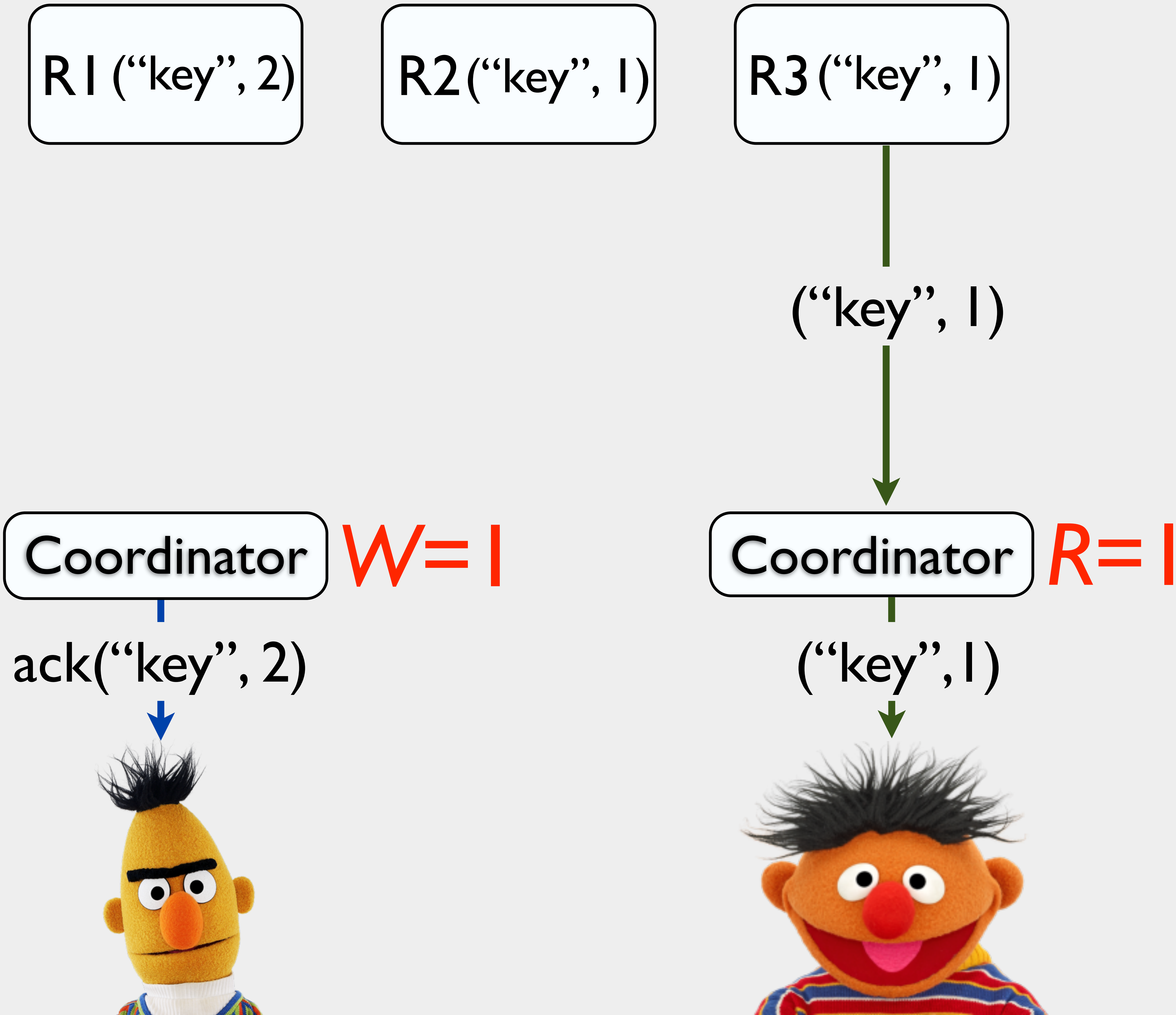
R2 (“key”, 1)

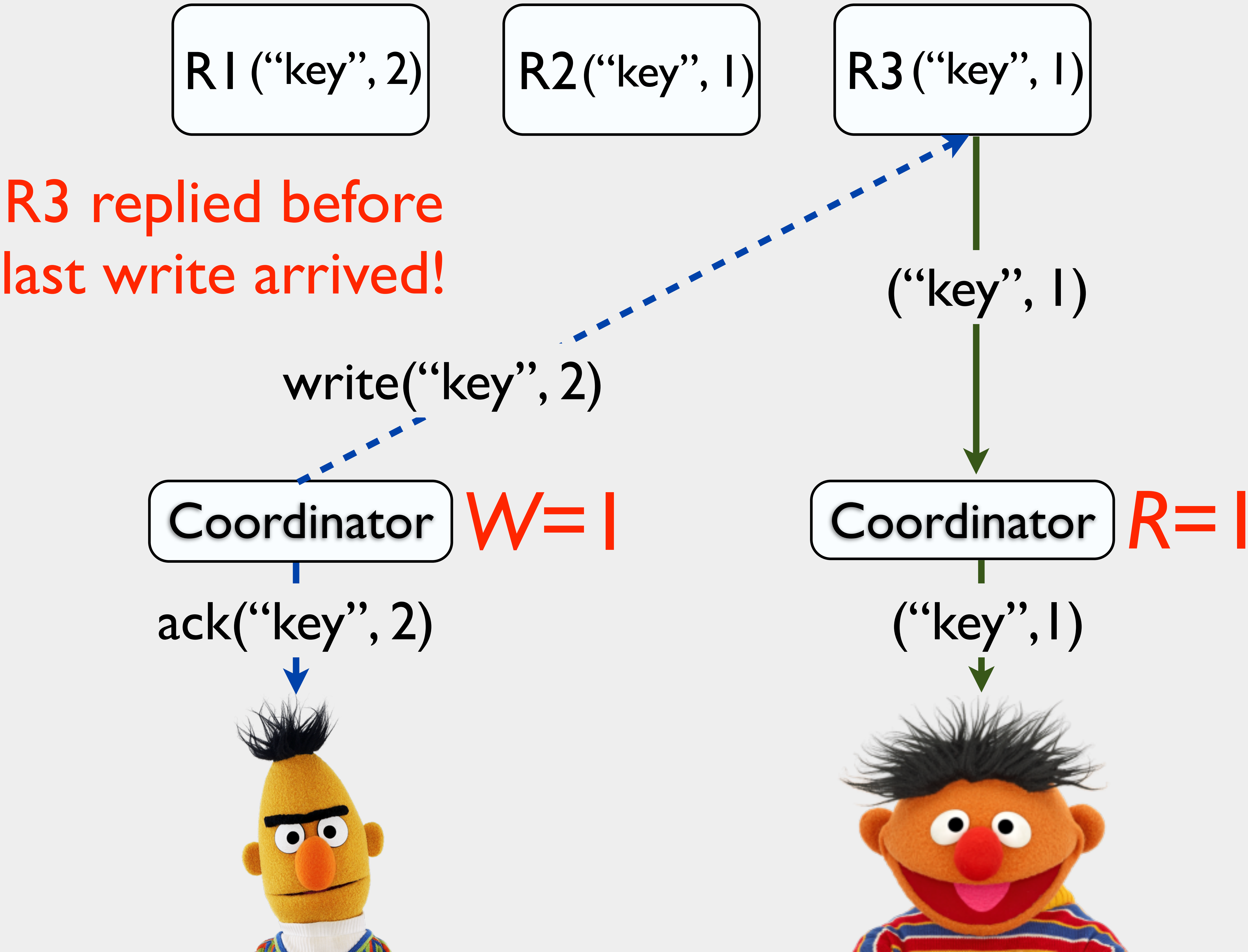
R3 (“key”, 1)

Coordinator W=1

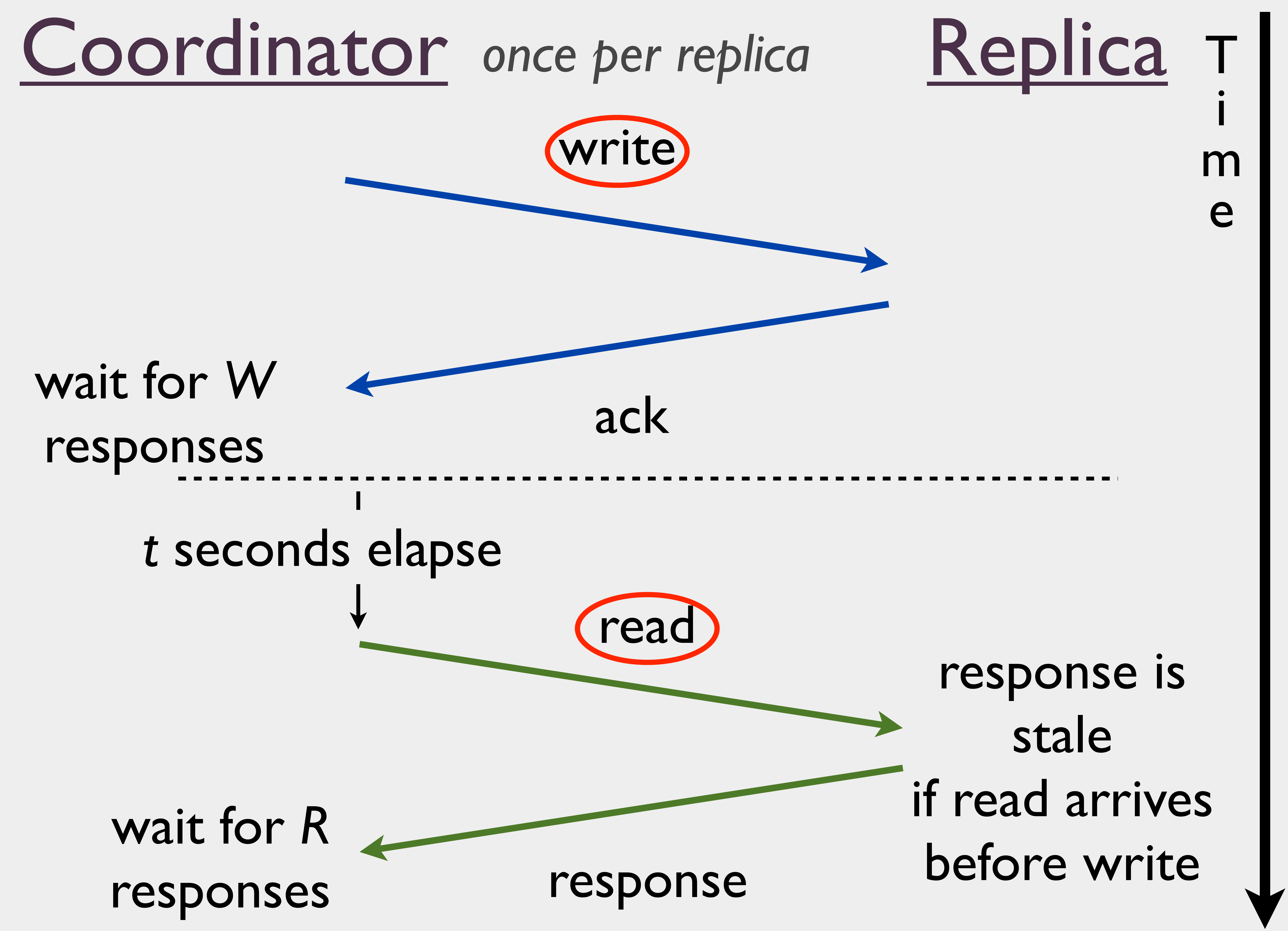
ack(“key”, 2)

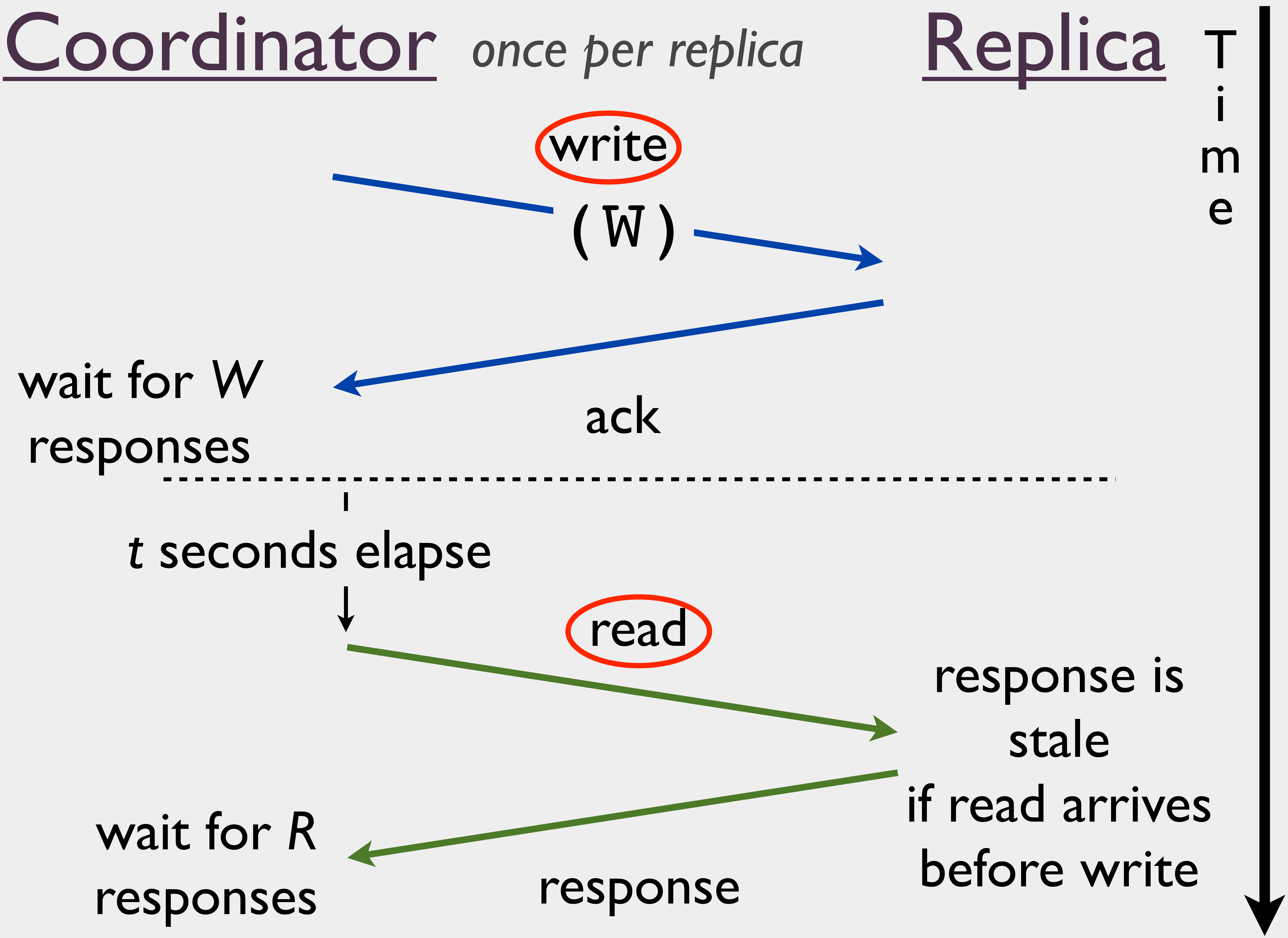
Coordinator R=1

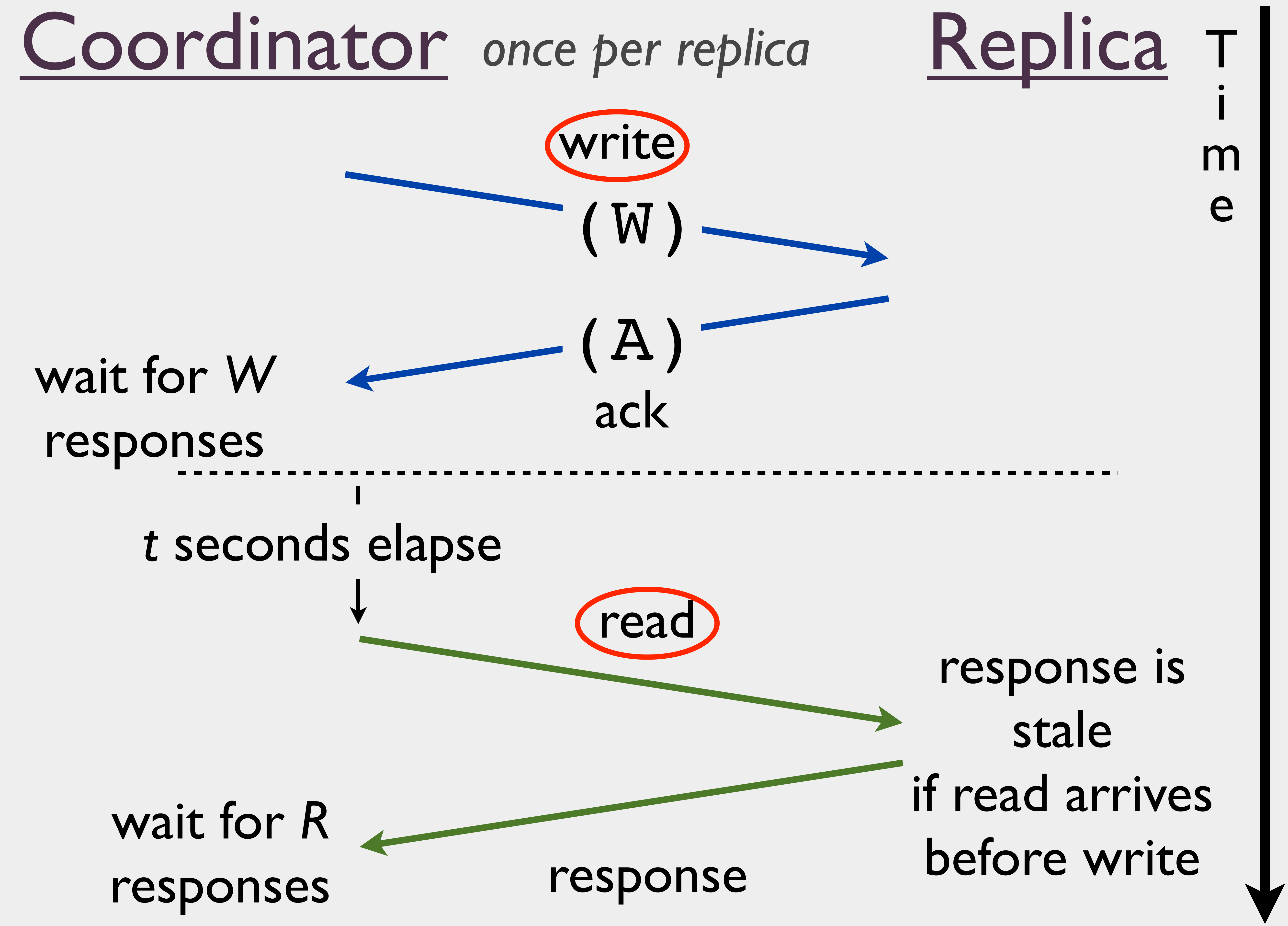

R1 ("key", 2)
R2 ("key", 1)
R3 ("key", 1)
("key", 1)
Coordinator
W=1
ack("key", 2)
Coordinator
R=1
("key", 1)

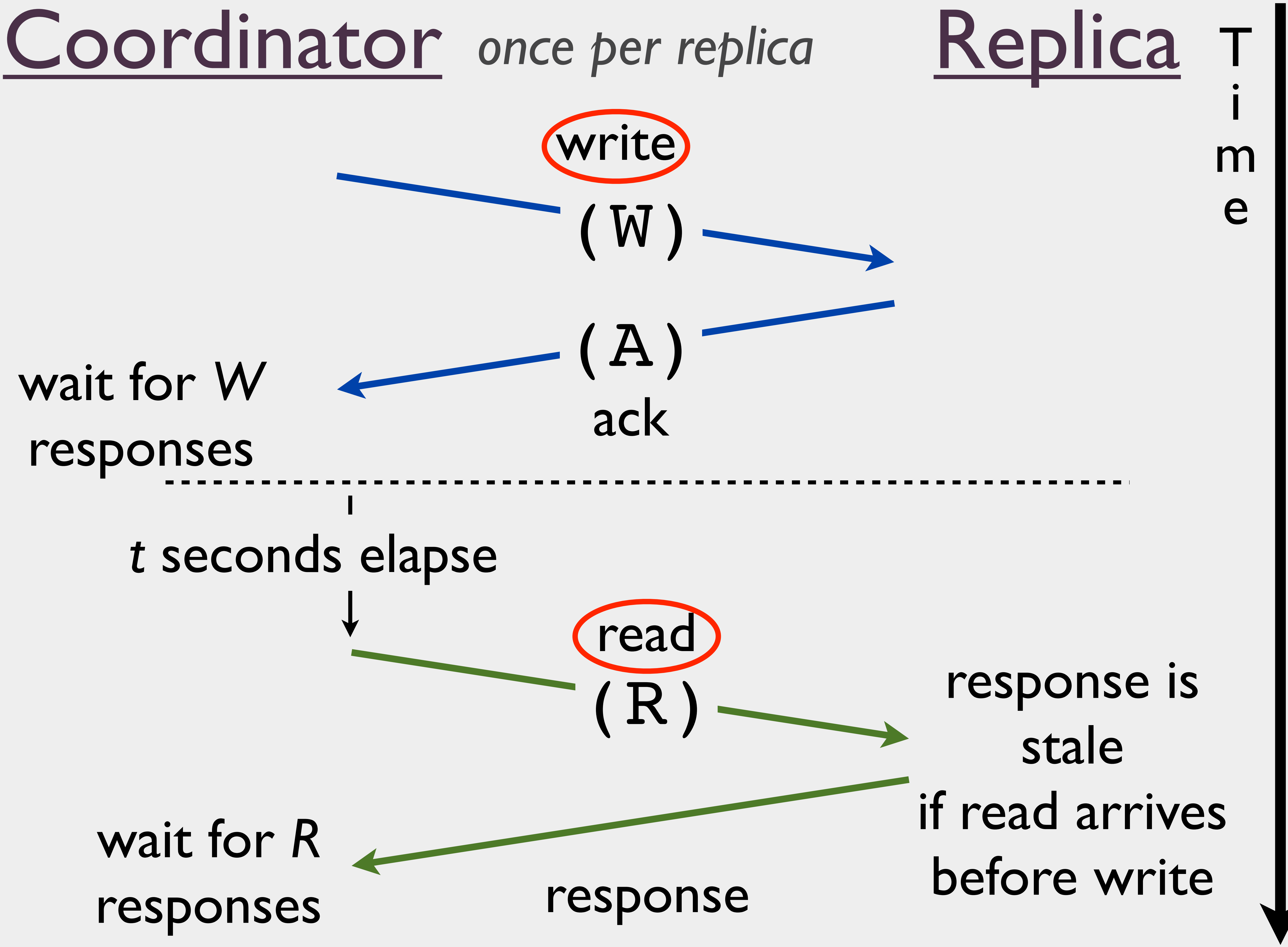


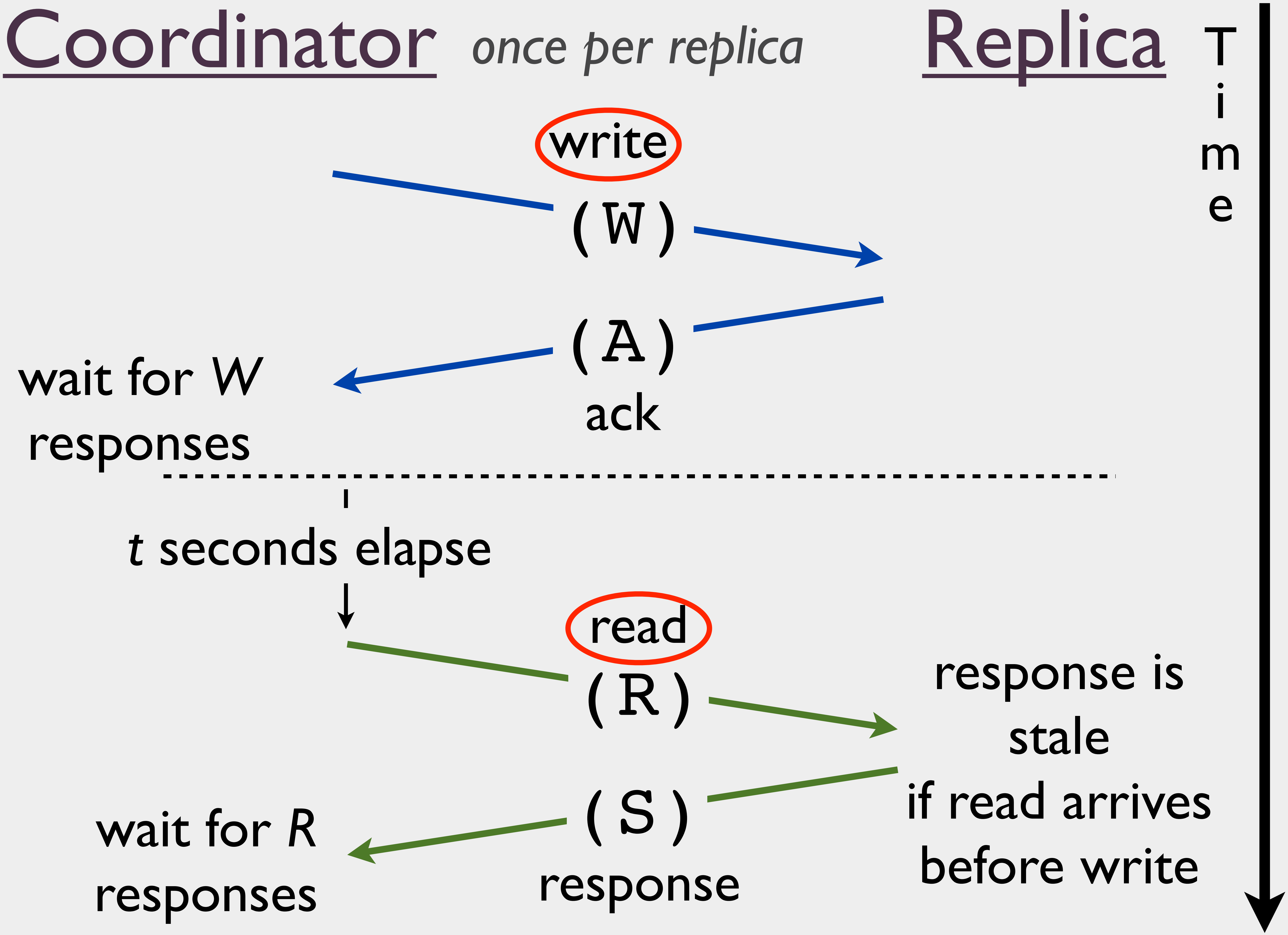
R1 ("key", 2)
R2 ("key", 1)
R3 ("key", 1)
R3 replied before
last write arrived!
write("key", 2)
("key", 1)
Coordinator
W=1
Coordinator
R=1
ack("key", 2)
("key",1)


Coordinator
once per replica
Replica
Time
write
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response


Coordinator
once per replica
Replica
Time
write
(W)
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response


Coordinator
once per replica
Replica
Time
write
(W)
(A)
ack
wait for W
responses
t seconds elapse
read
response is
stale
if read arrives
before write
wait for R
responses
response


Coordinator
once per replica
Replica
Time
write
(W)
(A)
ack
wait for W responses
t seconds elapse
read
(R)
response is stale if read arrives before write
wait for R responses
response


Coordinator
once per replica
Replica
Time
write
(W)
(A)
ack
wait for W responses
t seconds elapse
read
(R)
(S)
response
wait for R responses
response is stale if read arrives before write

Solving *WARS*: hard

Monte Carlo methods: easier

To use *WARS*:

gather latency data

W	A	R	S
53.2	10.3	15.3	9.6
44.5	8.2	22.4	14.2
101.1	11.3	19.8	6.7
...	...	...	...

run simulation

Monte Carlo, sampling

How eventual?

t-visibility: consistent reads with probability *p* after after *t* seconds

key: *WARS* model

need: latencies

How consistent?

What happens if I don’t wait?

Probability of reading later older than *k* versions is exponentially reduced by *k*

Pr(reading latest write) = 99%

Pr(reading one of last two writes) = 99.9%

Pr(reading one of last three writes) = 99.99%

Cassandra / CASSANDRA-4261

[patch] Support consistency-latency prediction in nodetool

Log In

Details

Type:	New Feature	Status:	Patch Available
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	1.2	Fix Version/s:	None
Component/s:	Tools		
Labels:	None		

People

Assignee:
Reporter:
Vote (0)

Dates

Created:
Updated:

Description

Introduction

Cassandra supports a variety of replication configurations: ReplicationFactor is set per-ColumnFamily and ConsistencyLevel is set per-request. Setting `ConsistencyLevel` to `QUORUM` for reads and writes ensures strong consistency, but `QUORUM` is often slower than `ONE`, `TWO`, or `THREE`. What should users choose?

This patch provides a latency-consistency analysis within `nodetool`. Users can accurately predict Cassandra's behavior in their production environments without interfering with performance.

https://issues.apache.org/jira/browse/CASSANDRA-4261

Cassandra cluster, injected latencies:

WARS Simulation accuracy

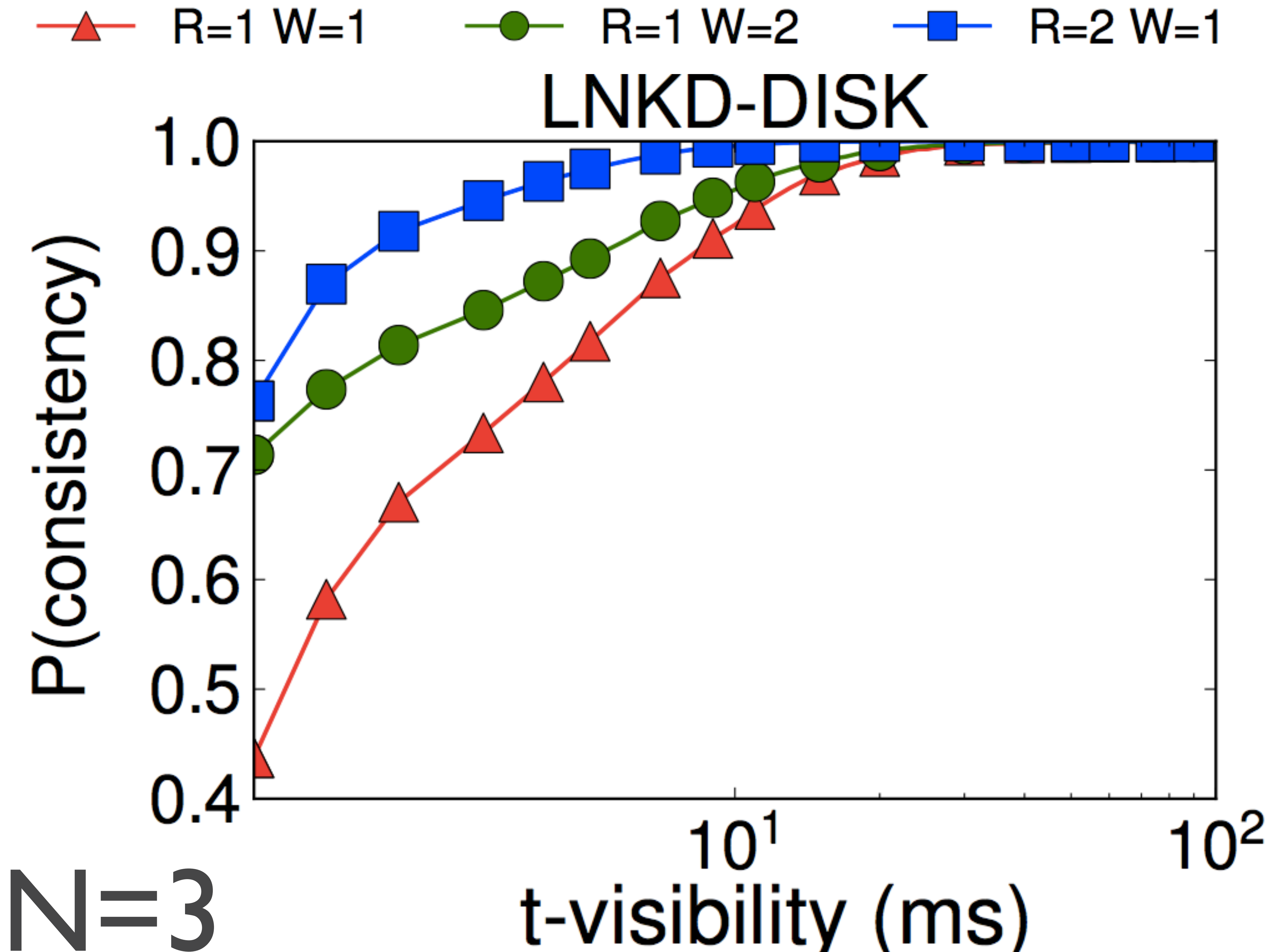
t-staleness RMSE: 0.28%

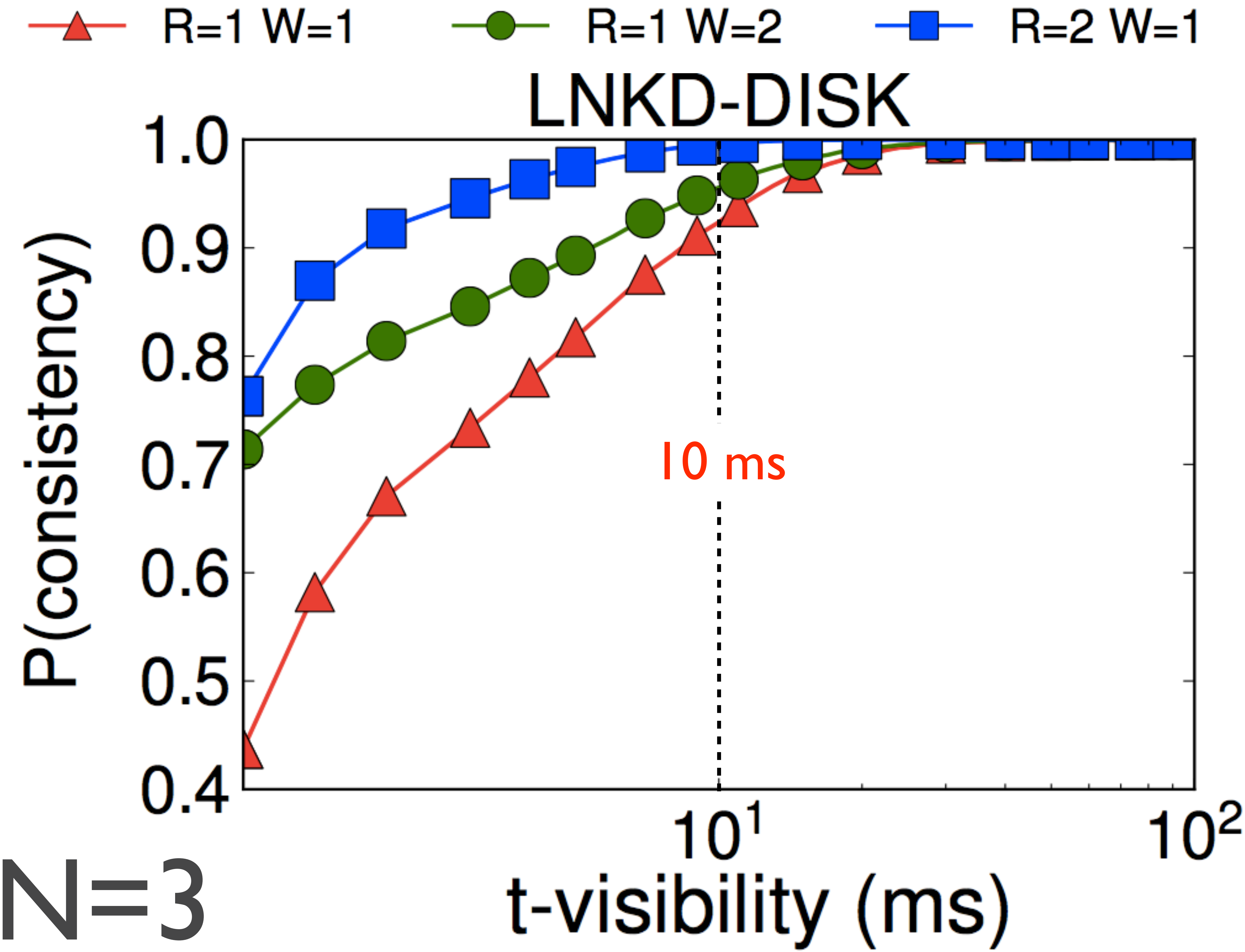
latency N-RMSE: 0.48%

LinkedIn
150M+ users
built and uses Voldemort

Yammer
100K+ companies
uses Riak

production latencies
fit gaussian mixtures


R=1 W=1
R=1 W=2
R=2 W=1
LNKD-DISK
1.0
0.9
0.8
0.7
0.6
0.5
0.4
P(consistency)
10^1
10^2
t-visibility (ms)
N=3



R=1 W=1
R=1 W=2
R=2 W=1
LNKD-DISK
P(consistency)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
10 ms
10^1
10^2
t-visibility (ms)

N=3

LNKD-DISK

99.9% consistent reads:

R=2, W=1

***t* = 13.6 ms**

Latency: 12.53 ms

100% consistent reads:

R=3, W=1

Latency: 15.01 ms

N=3

Latency is combined read and write latency at 99.9th percentile

LNKD-DISK

16.5% faster

99.9% consistent reads:
R=2, W=1

t = 13.6 ms

Latency: 12.53 ms

100% consistent reads:
R=3, W=1

Latency: 15.01 ms

N=3

Latency is combined read and write latency at 99.9th percentile

LNKD-DISK

16.5% faster

worthwhile?

N=3

99.9% consistent reads:

R=2, W=1

t = 13.6 ms

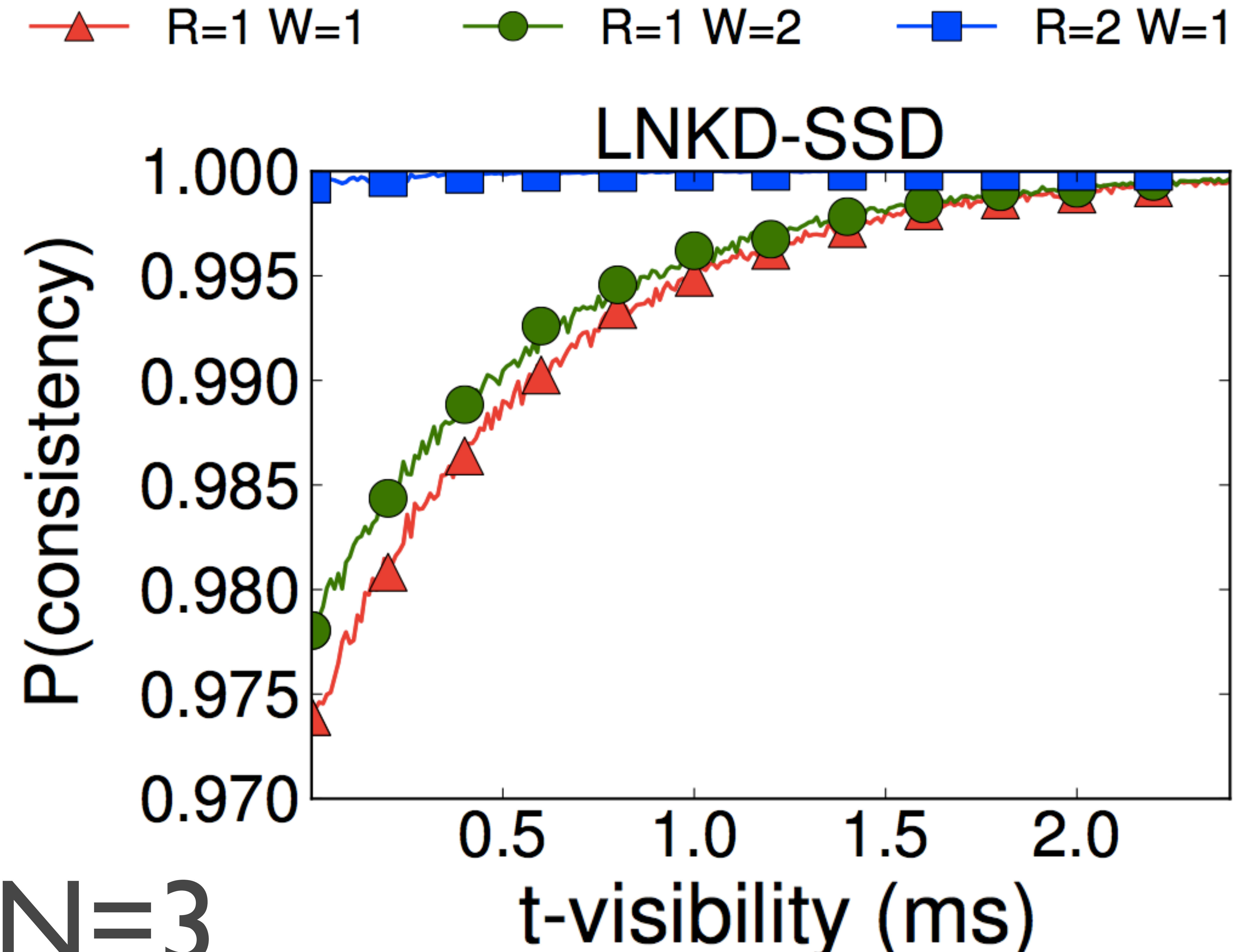
Latency: 12.53 ms

100% consistent reads:

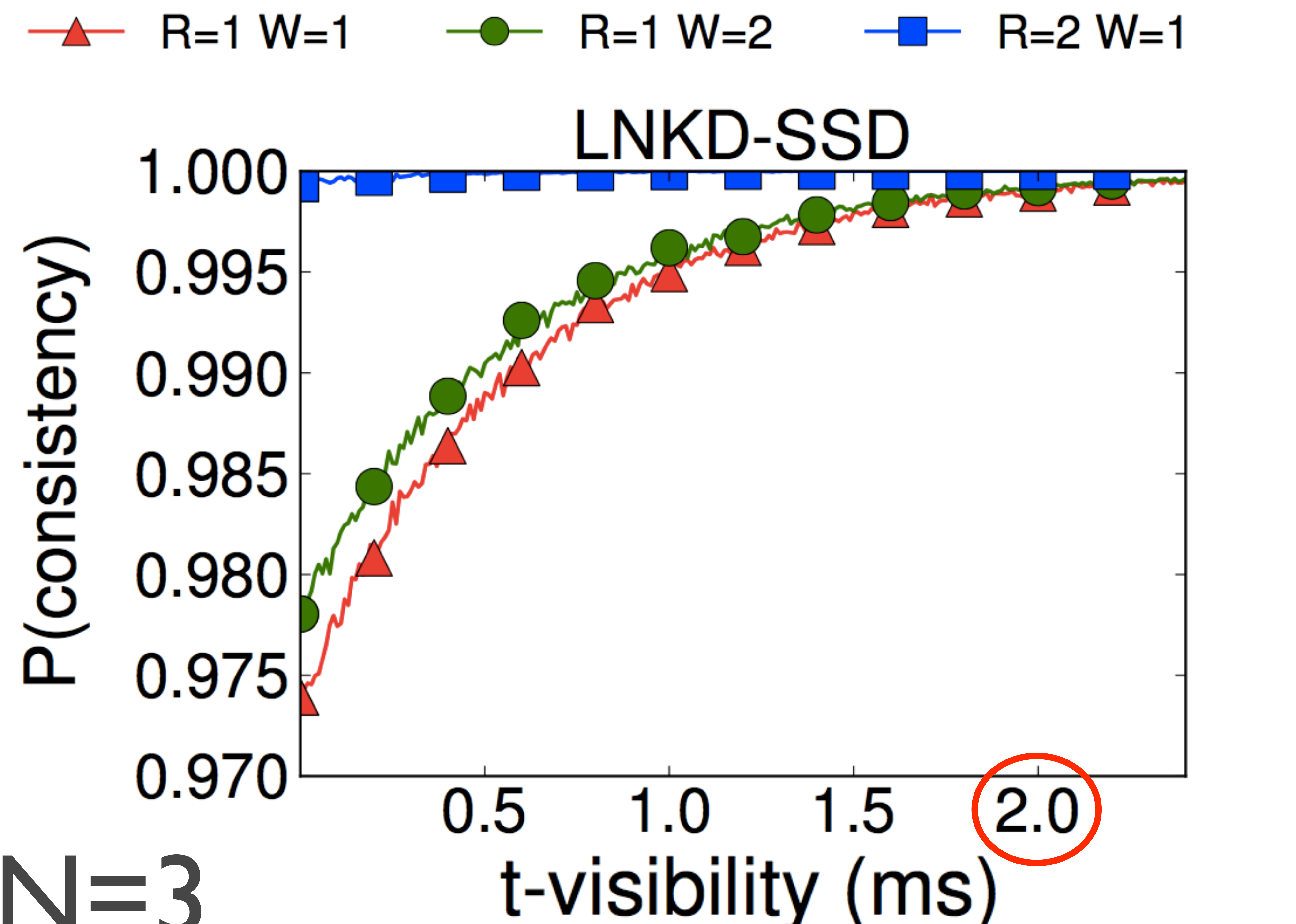
R=3, W=1

Latency: 15.01 ms

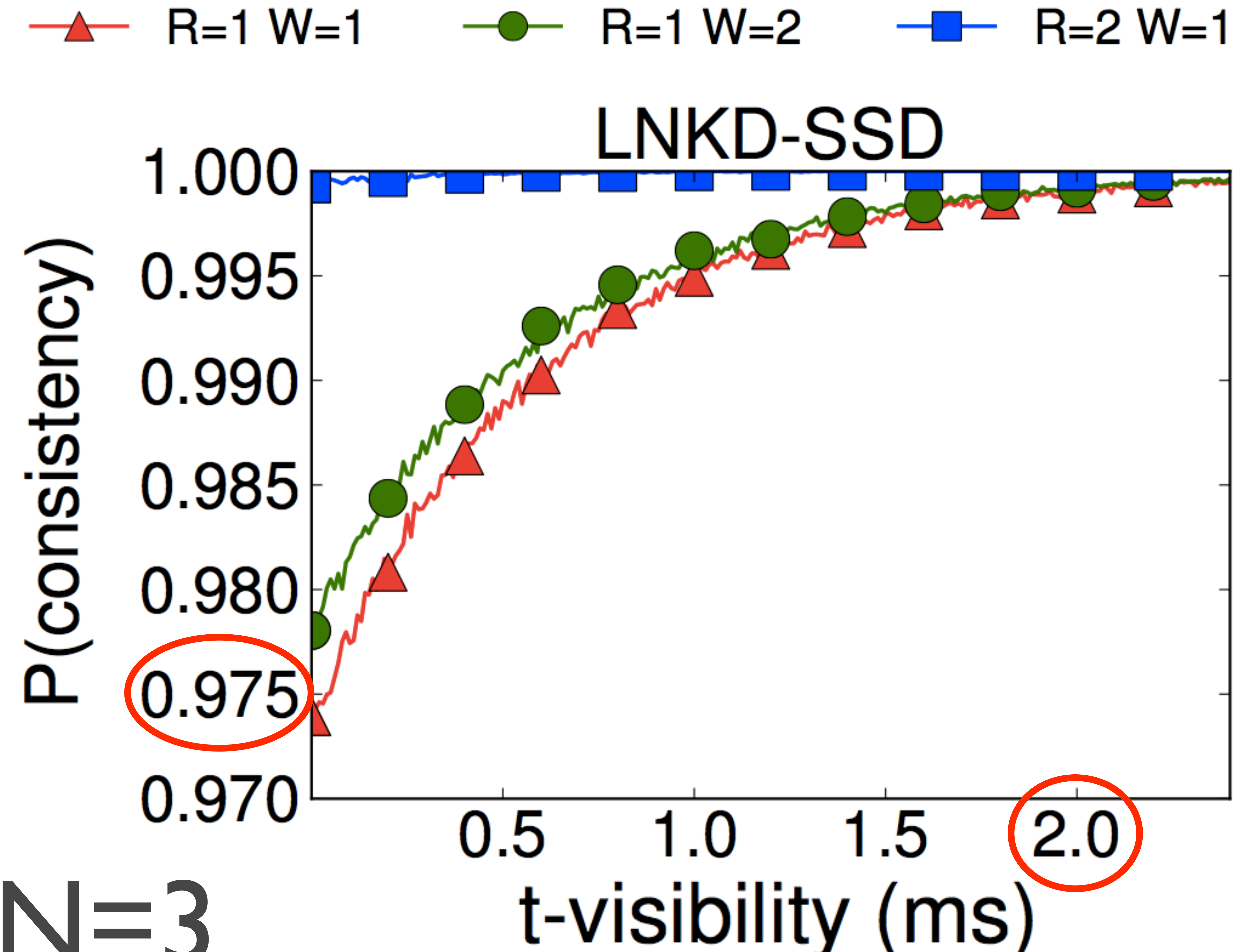
Latency is combined read and write latency at 99.9th percentile


R=1 W=1
R=1 W=2
R=2 W=1
LNKD-SSD
P(consistency)
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.5
1.0
1.5
2.0
t-visibility (ms)

N=3


R=1 W=1
R=1 W=2
R=2 W=1
LNKD-SSD
P(consistency)
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.5
1.0
1.5
2.0
t-visibility (ms)

N=3


R=1 W=1
R=1 W=2
R=2 W=1
LNKD-SSD
P(consistency)
1.000
0.995
0.990
0.985
0.980
0.975
0.970
0.5
1.0
1.5
2.0
t-visibility (ms)

N=3

LNKD-SSD

99.9% consistent reads:

R=1, W=1

t = 1.85 ms

Latency: 1.32 ms

100% consistent reads:

R=3, W=1

Latency: 4.20 ms

N=3

Latency is combined read and write latency at 99.9th percentile

LNKD-SSD

59.5% faster

99.9% consistent reads:

R=1, W=1

t = 1.85 ms

Latency: 1.32 ms

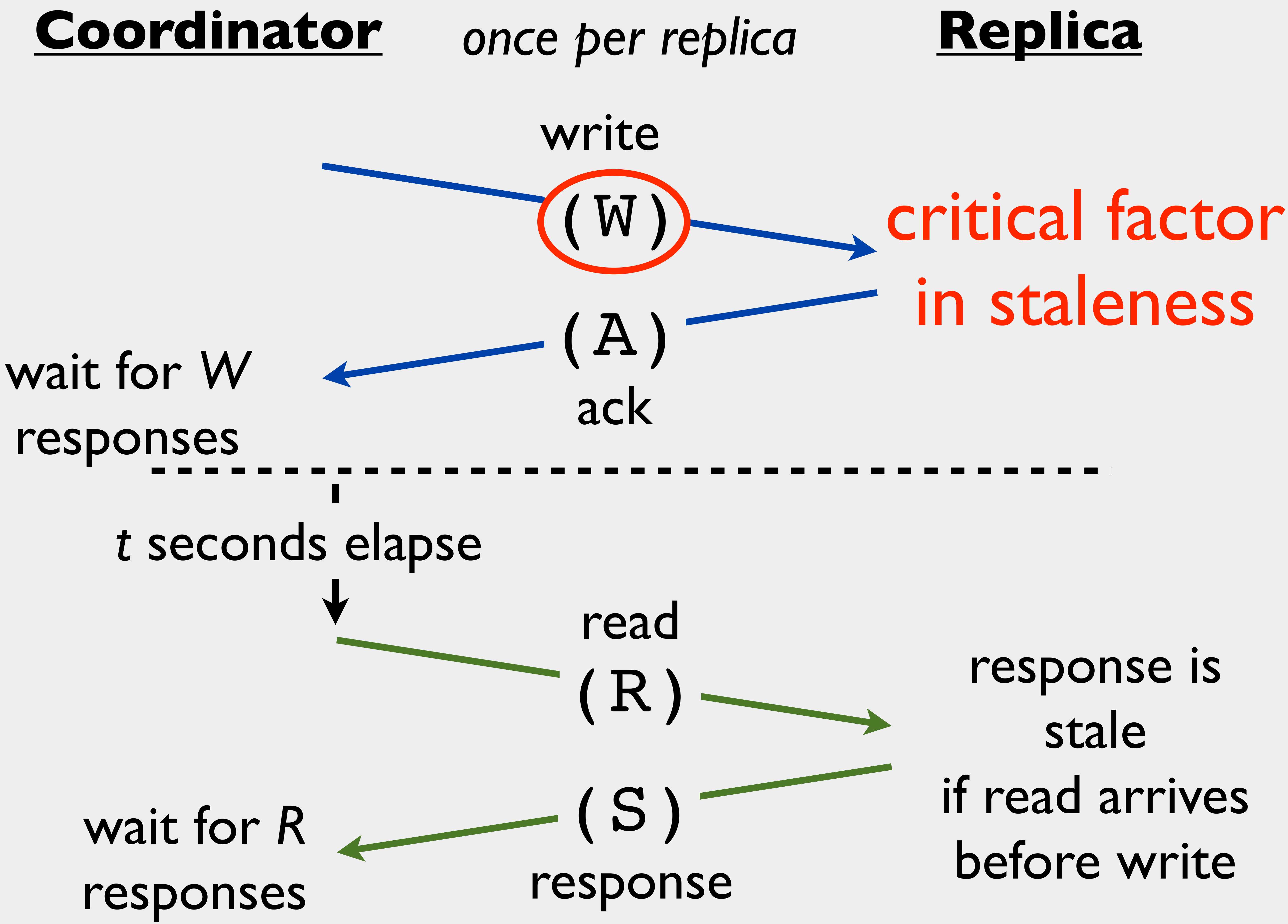
100% consistent reads:

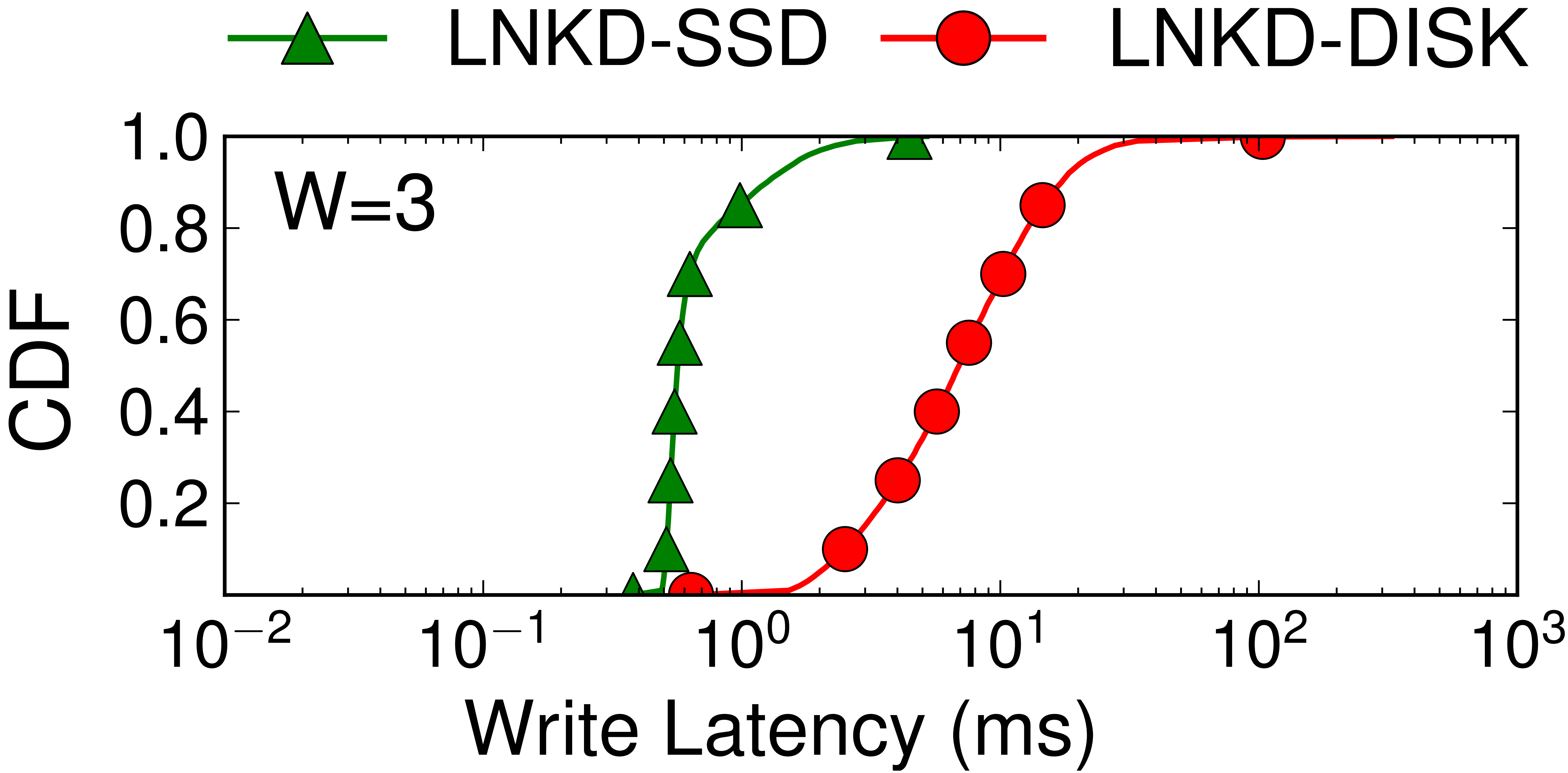
R=3, W=1

Latency: 4.20 ms

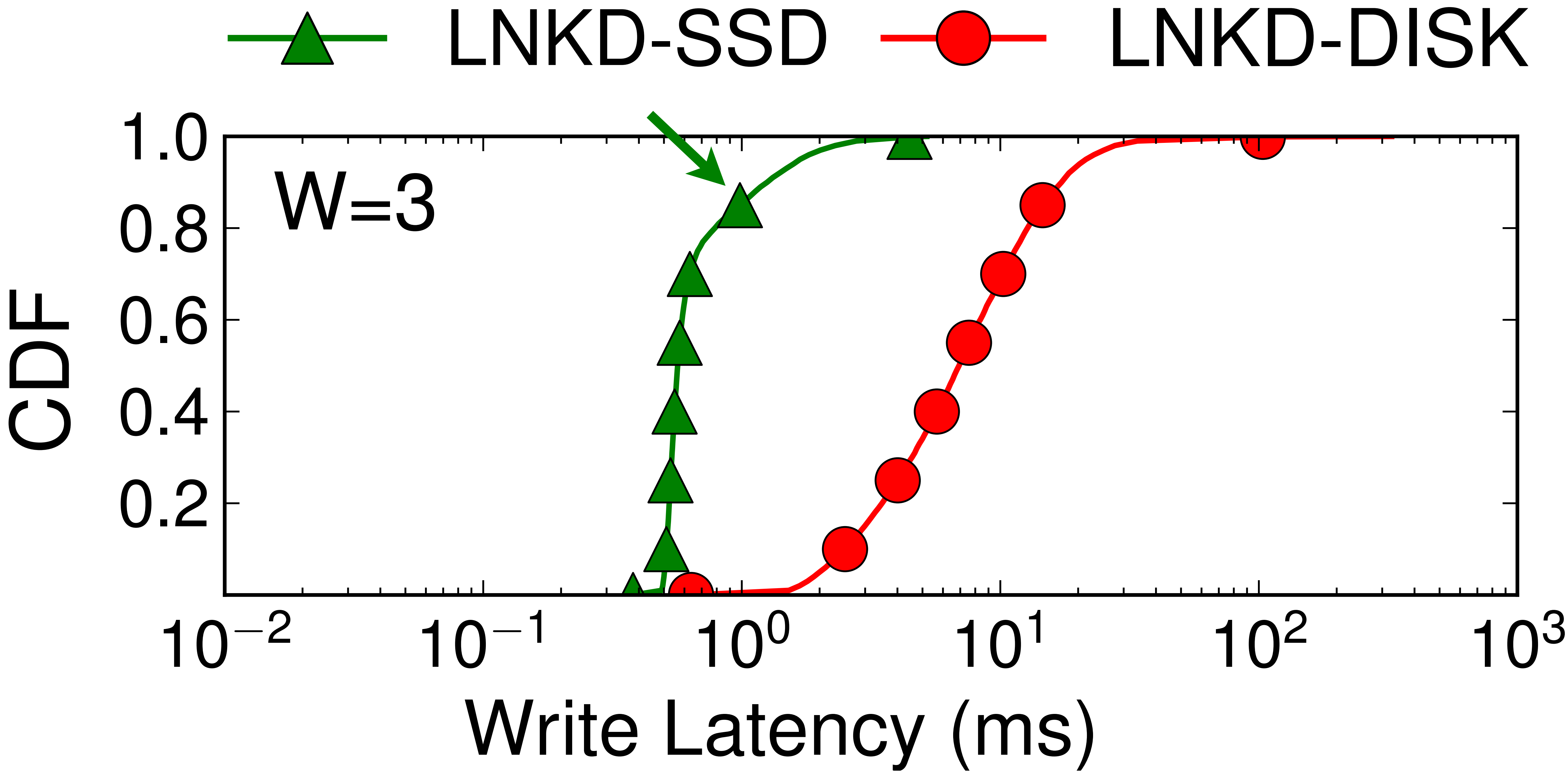
N=3

Latency is combined read and write latency at 99.9th percentile


Coordinator
once per replica
Replica
write
(W)
critical factor in staleness
(A)
ack
wait for W responses
t seconds elapse
read
(R)
response is stale if read arrives before write
(S)
response
wait for R responses

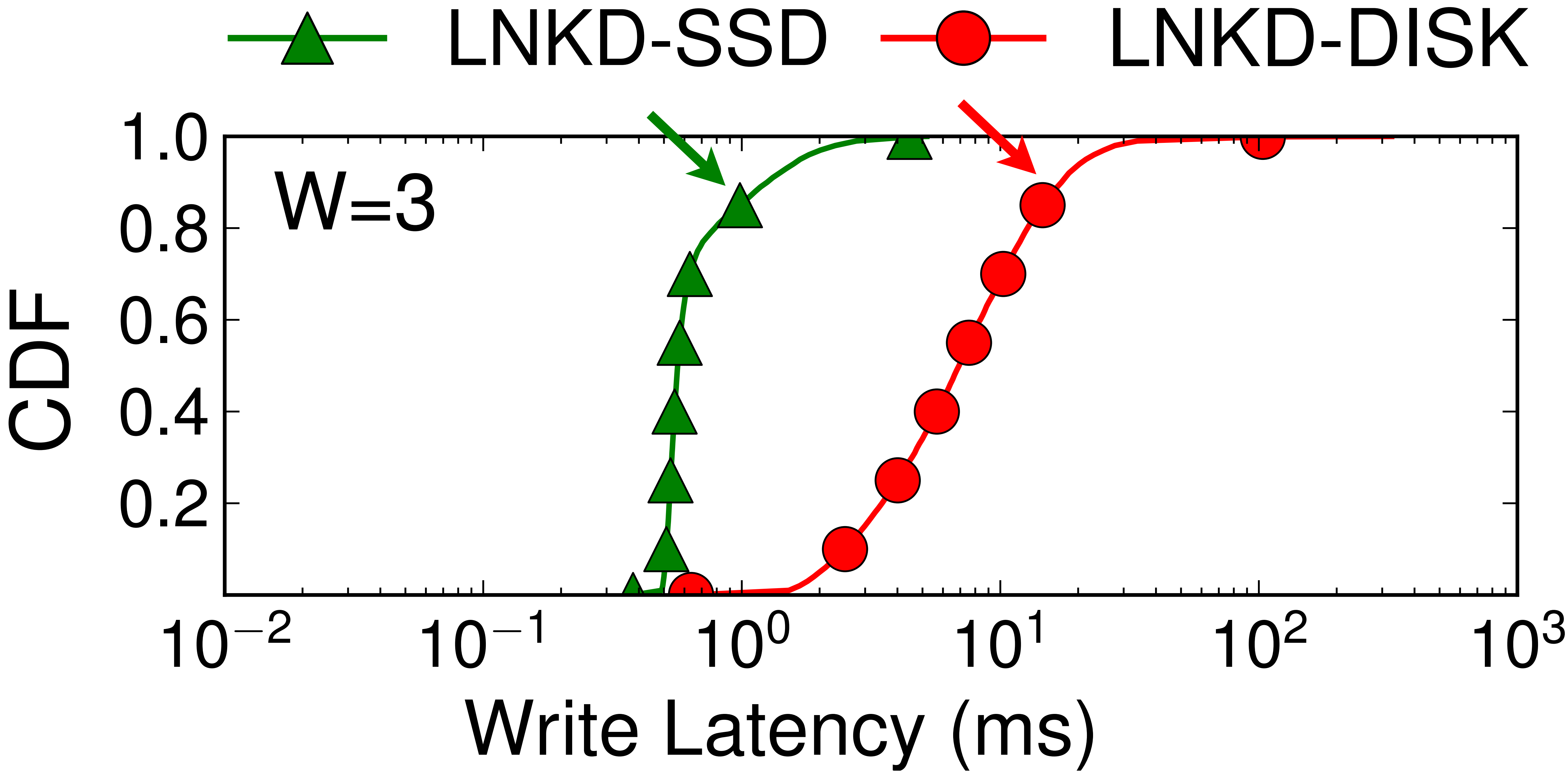

LNKD-SSD
LNKD-DISK
W=3
CDF
1.0
0.8
0.6
0.4
0.2
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Write Latency (ms)

N=3

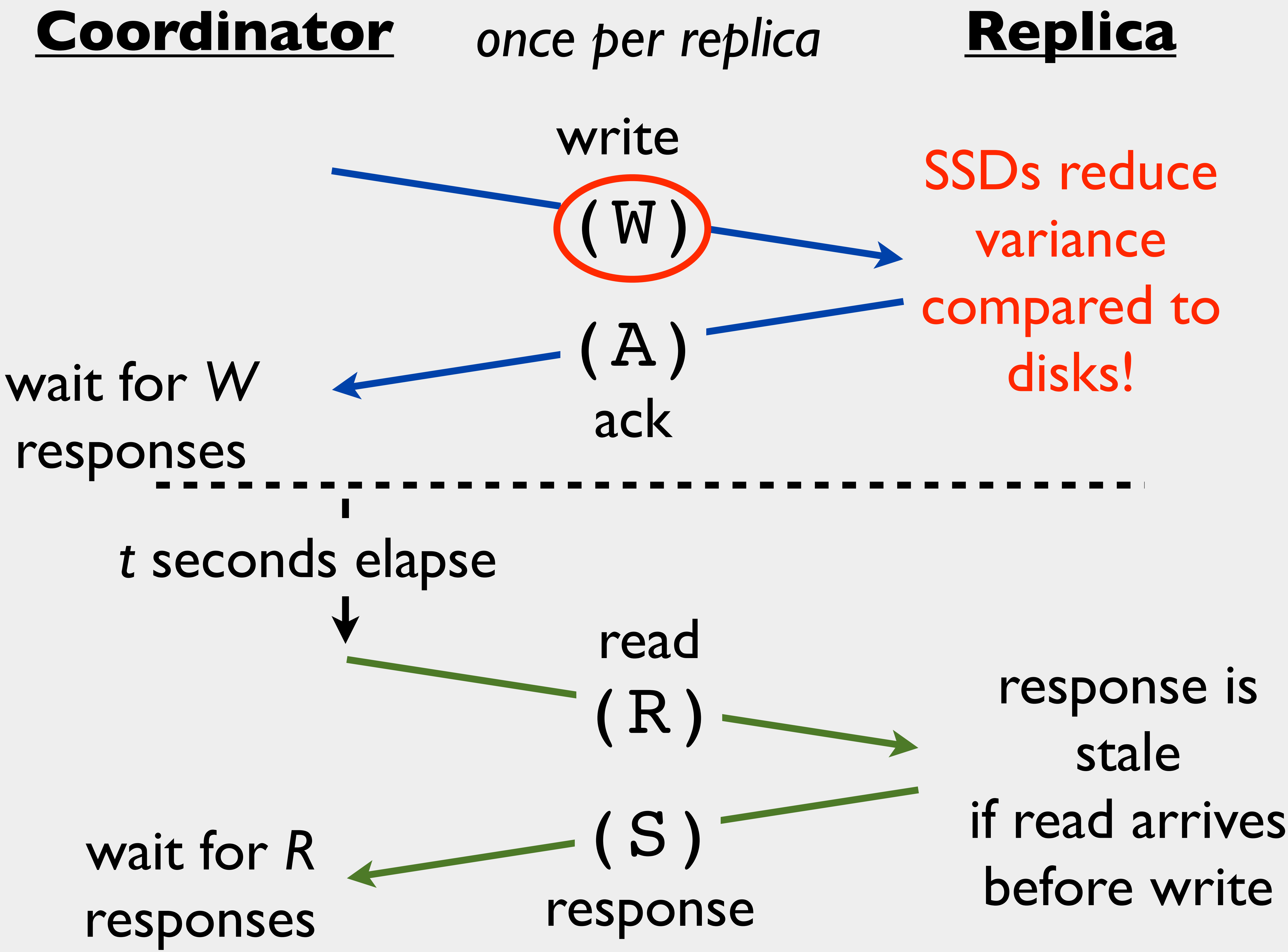


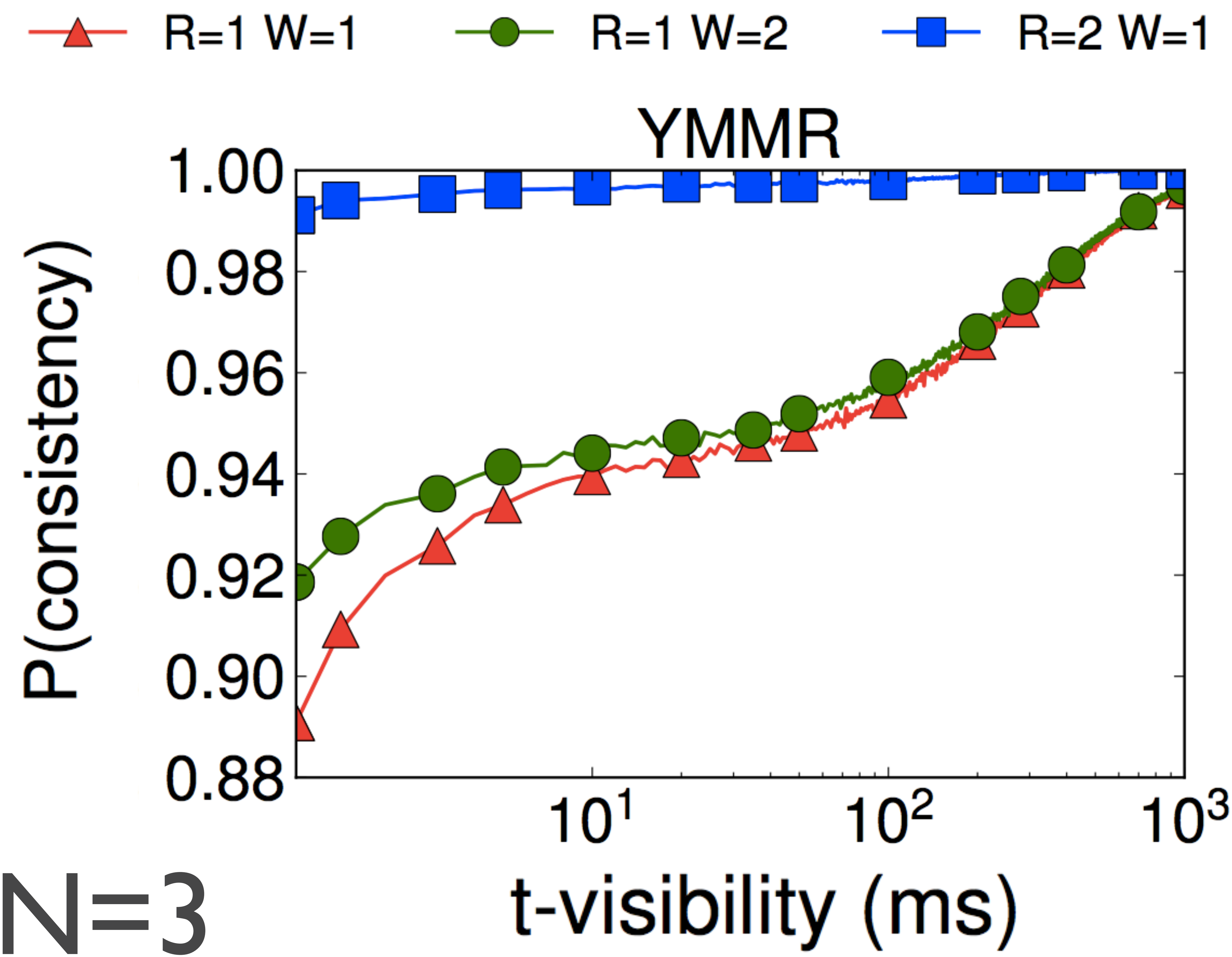
LNKD-SSD
LNKD-DISK
W=3
CDF
1.0
0.8
0.6
0.4
0.2
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Write Latency (ms)

N=3

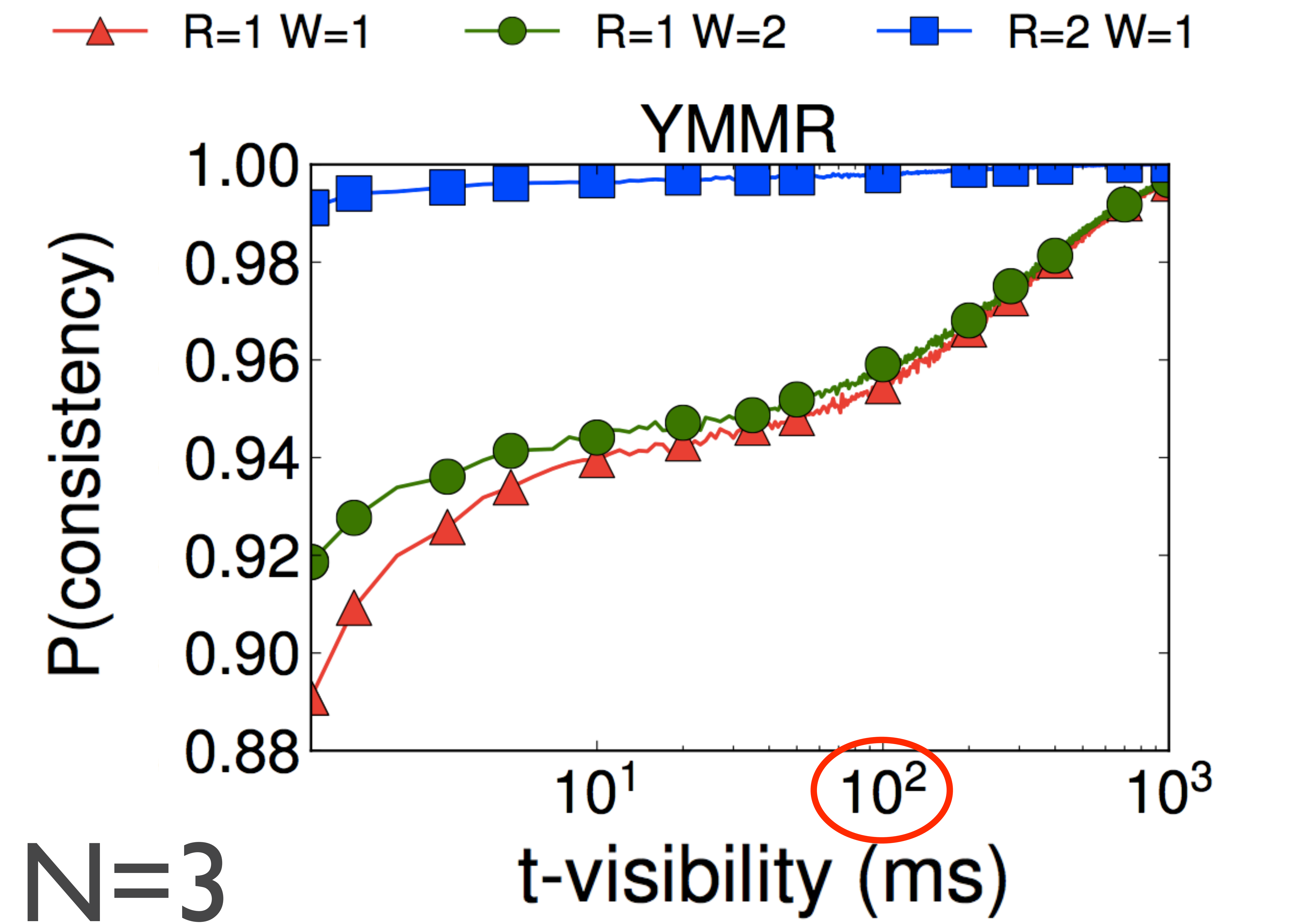

LNKD-SSD
LNKD-DISK
W=3
CDF
1.0
0.8
0.6
0.4
0.2
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Write Latency (ms)

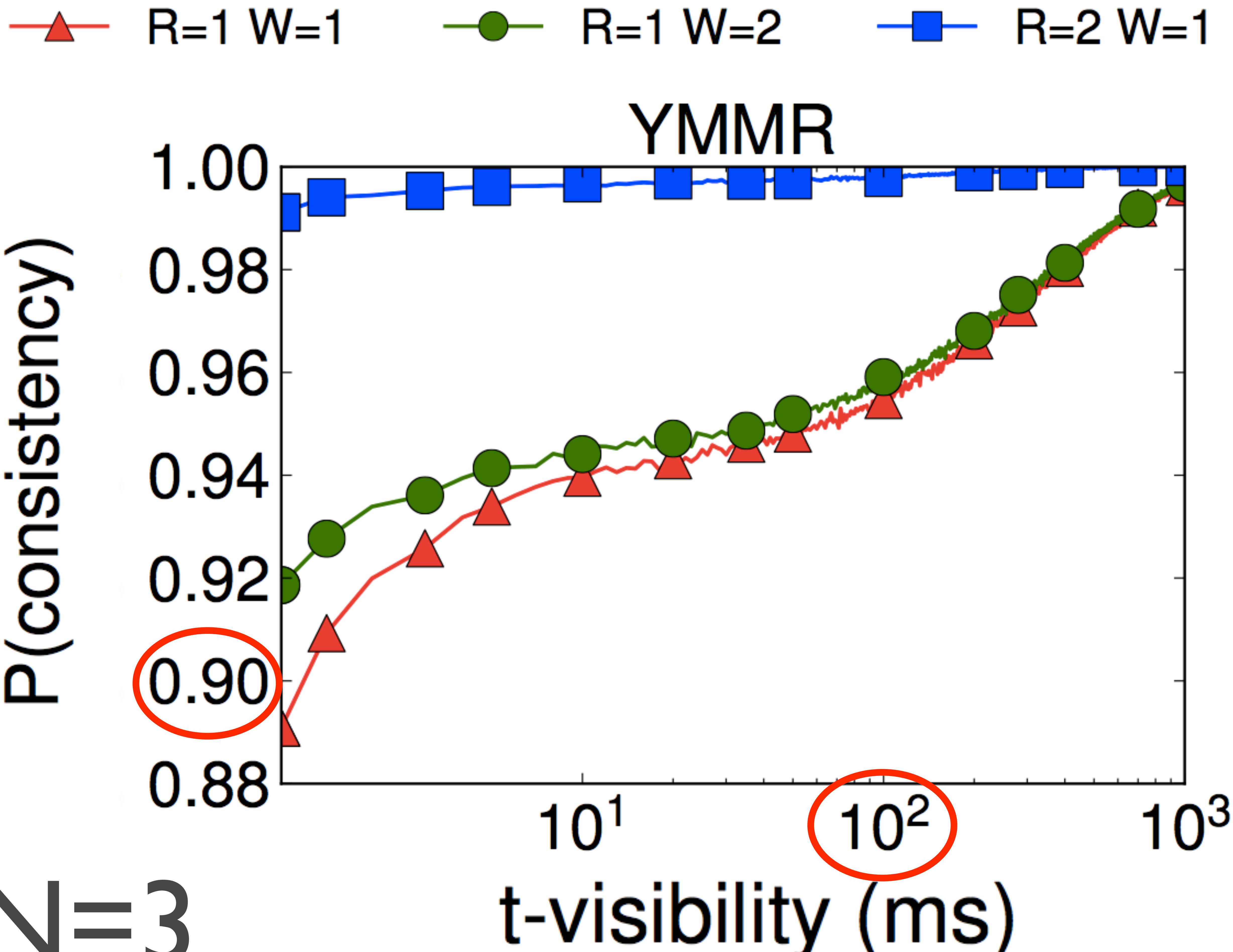
N=3


Coordinator
once per replica
Replica
write
(W)
SSDs reduce variance compared to disks!
(A)
wait for W responses
ack
t seconds elapse
read
(R)
response is stale if read arrives before write
(S)
wait for R responses
response


R=1 W=1
R=1 W=2
R=2 W=1
YMMR
P(consistency)
1.00
0.98
0.96
0.94
0.92
0.90
0.88
10^1
10^2
10^3
t-visibility (ms)

N=3


R=1 W=1
R=1 W=2
R=2 W=1
YMMR
P(consistency)
1.00
0.98
0.96
0.94
0.92
0.90
0.88
10^1
10^2
10^3
t-visibility (ms)
N=3


R=1 W=1
R=1 W=2
R=2 W=1
YMMR
P(consistency)
1.00
0.98
0.96
0.94
0.92
0.90
0.88
10^1
10^2
10^3
t-visibility (ms)

N=3

YMMR

99.9% consistent reads:

R=1, W=1

***t* = 202.0 ms**

Latency: 43.3 ms

100% consistent reads:

R=3, W=1

Latency: 230.06 ms

N=3

Latency is combined read and write latency at 99.9th percentile

YMMR

99.9% consistent reads:
R=1, W=1

t = 202.0 ms

Latency: 43.3 ms

100% consistent reads:
R=3, W=1

Latency: 230.06 ms

Latency is combined read and write latency at 99.9th percentile

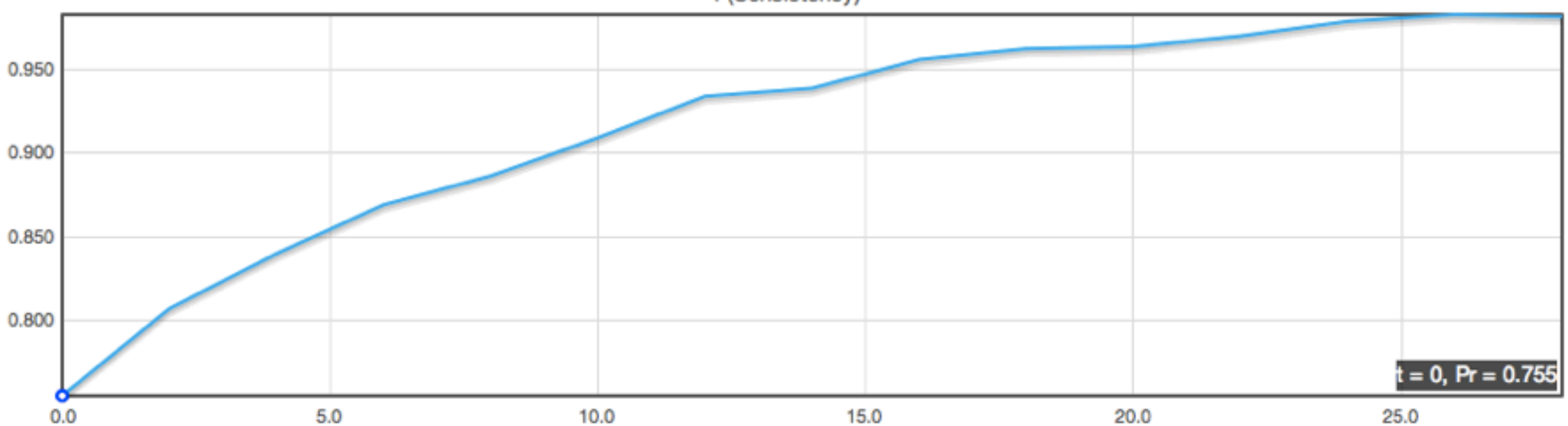
81.1% faster

N=3

PBS: Probabilistically Bounded Staleness About Instructions Demo Questions More Info Thanks

How Eventual is Eventual Consistency? PBS in action under Dynamo-style quorums

P(Consistency)



Time After Commit (ms)

(Plot isn't monotonically increasing? Increase the accuracy.)

You have at least a 74.8 percent chance of reading the last written version 0 ms after it commits.
You have at least a 92.2 percent chance of reading the last written version 10 ms after it commits.
You have at least a 99.96 percent chance of reading the last written version 100 ms after it commits.

Replica Configuration

N: 3
R: 1
W: 1

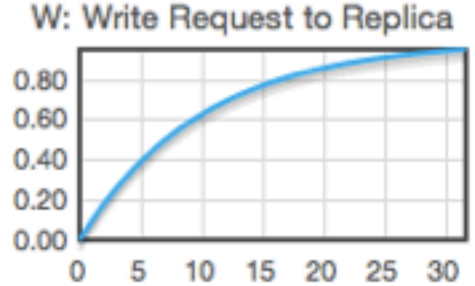
Read Latency: Median 8.43 ms, 99.9th %ile 36.97 ms
Write Latency: Median 8.38 ms, 99.9th %ile 38.28 ms

Tolerable Staleness: 1 version

Accuracy: 2500 iterations/point

Operation Latency: Exponentially Distributed CDFs

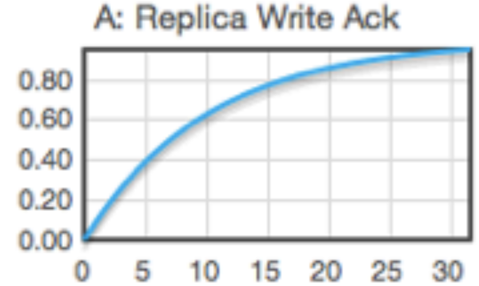
W: Write Request to Replica



Latency (ms)

λ 0.100

A: Replica Write Ack



Latency (ms)

λ 0.100

R: Read Request to Replica

Latency (ms)

λ 0.100

S: Replica Read Response

Latency (ms)

λ 0.100

Workflow

1. Tracing
2. Simulation
3. Tune *N*, *R*, *W*
4. Profit

Cassandra / CASSANDRA-4261

[patch] Support consistency-latency prediction in nodetool

Log In

Details

Type:	New Feature	Status:	Patch Available
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	1.2	Fix Version/s:	None
Component/s:	Tools		
Labels:	None		

People

Assignee:

Reporter:

Vote (0)

Dates

Created:

Updated:

Description

Introduction

Cassandra supports a variety of replication configurations: ReplicationFactor is set per-ColumnFamily and ConsistencyLevel is set per-request. Setting `ConsistencyLevel` to `QUORUM` for reads and writes ensures strong consistency, but `QUORUM` is often slower than `ONE`, `TWO`, or `THREE`. What should users choose?

This patch provides a latency-consistency analysis within `nodetool`. Users can accurately predict Cassandra's behavior in their production environments without interfering with performance.

https://issues.apache.org/jira/browse/CASSANDRA-4261

```
ubuntu@ip-10-46-87-156:~/cassandra-pbs$ bin/nodetool -h ec2-23-2
0-168-89.compute-1.amazonaws.com predictconsistency 3 75 1
75ms after a given write, with maximum version staleness of k=1
N=3, R=1, W=1
Probability of consistent reads: 0.716500
Average read latency: 31.170300ms (99.900th %ile 193ms)
Average write latency: 42.873798ms (99.900th %ile 192ms)

N=3, R=1, W=2
Probability of consistent reads: 0.902400
Average read latency: 30.958000ms (99.900th %ile 189ms)
Average write latency: 106.877098ms (99.900th %ile 240ms)

N=3, R=1, W=3
Probability of consistent reads: 1.000000
Average read latency: 30.104000ms (99.900th %ile 192ms)
Average write latency: 171.652298ms (99.900th %ile 341ms)

N=3, R=2, W=1
Probability of consistent reads: 0.934200
Average read latency: 84.446602ms (99.900th %ile 231ms)
Average write latency: 42.800301ms (99.900th %ile 194ms)

N=3, R=2, W=2
Probability of consistent reads: 1.000000
Average read latency: 82.663902ms (99.900th %ile 238ms)
Average write latency: 106.141296ms (99.900th %ile 236ms)
```

PBS

problem: no guarantees with eventual consistency

solution: consistency prediction

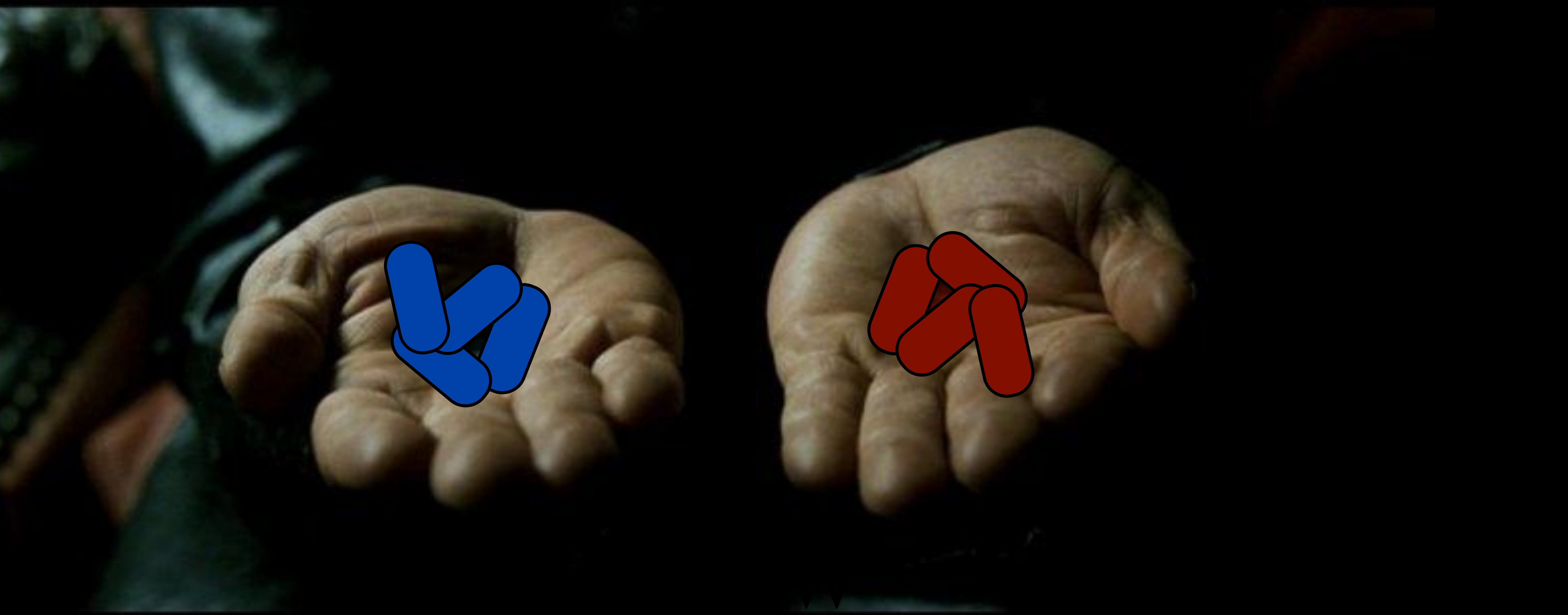
technique: measure latencies
use *WARS* model

consistency

is a metric

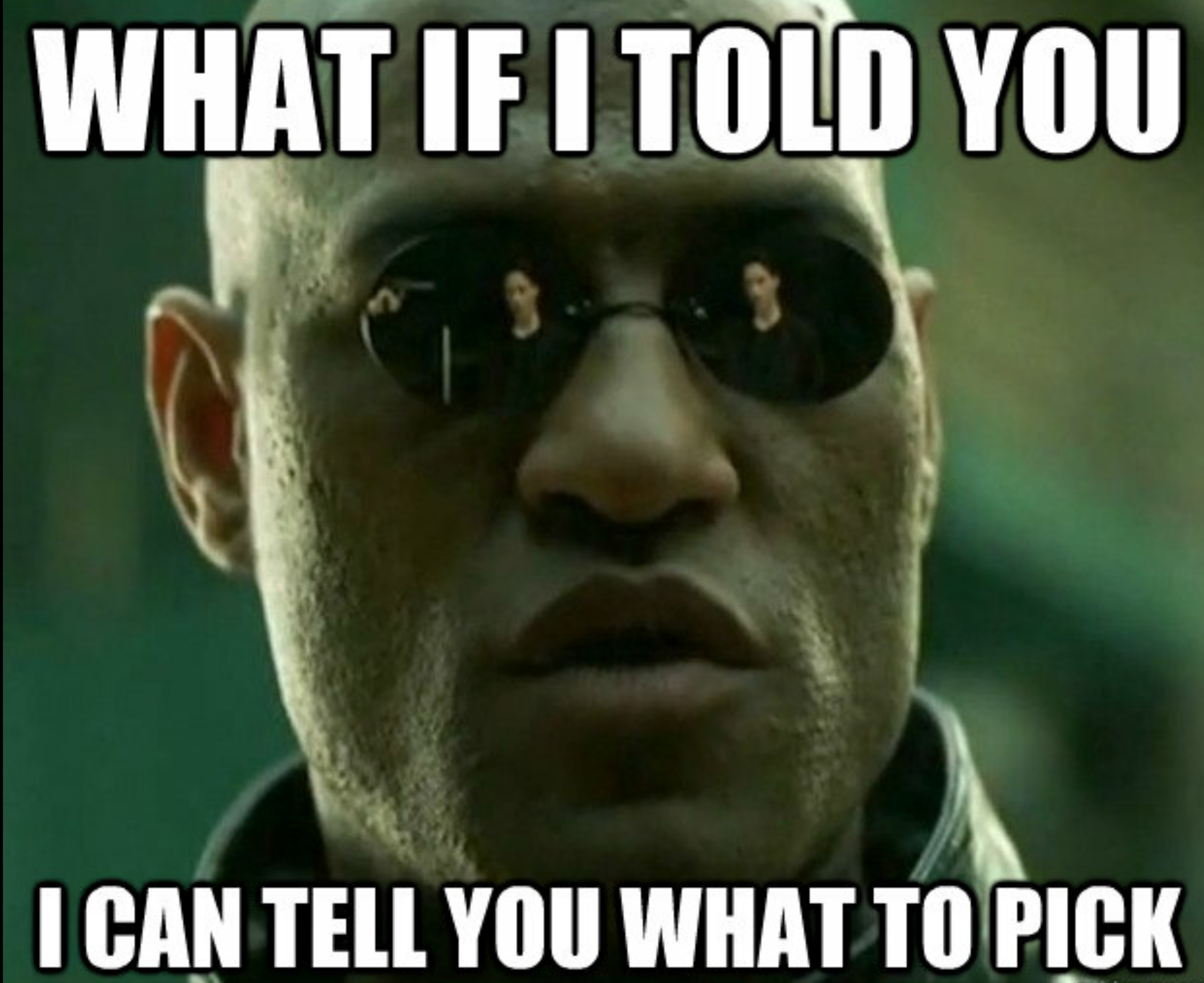
to measure

to predict



strong consistency

lower latency


WHAT IF I TOLD YOU
I CAN TELL YOU WHAT TO PICK

PBS

latency vs. consistency trade-offs

simple modeling with *WARS*

model staleness in time, versions

PBS

latency vs. consistency trade-offs

simple modeling with *WARS*

model staleness in time, versions

eventual consistency

often fast

often consistent

PBS helps explain **when** and **why**

PBS

latency vs. consistency trade-offs

simple modeling with *WARS*

model staleness in time, versions

eventual consistency

often fast

often consistent

PBS helps explain **when** and **why**

pbs.cs.berkeley.edu/#demo

@pbailis

VLDB 2012 early print

tinyurl.com/pbsvldb

cassandra patch

tinyurl.com/pbspatch

Extra Slides

Related Work

Quorum System Theory

e.g., Probabilistic Quorums

k-quorums

Deterministic Staleness

e.g., TACT/conits

FRACS

Consistency Verification

e.g., Golab et al. (PODC '11), Bermbach and Tai (M4WSOC '11)

PBS
and
apps

staleness requires either:

staleness-tolerant data structures

timelines, logs

cf. commutative data structures

logical monotonicity

asynchronous compensation code

detect violations after data is returned; see paper

write code to fix any errors

cf. "Building on Quicksand"

memories, guesses, apologies

asynchronous compensation

minimize:

(compensation cost)×(# of expected anomalies)

Read only newer data?

(monotonic reads session guarantee)

$$\frac{\text{\# versions tolerable staleness}}{} = \frac{\text{client's read rate}}{\text{global write rate}}$$

(for a given key)

Failure?

Treat failures as

latency

spiKes

How l o n g

do partitions last?

what time interval?

99.9% uptime/yr

⇒ 8.76 hours downtime/yr

8.76 consecutive hours down

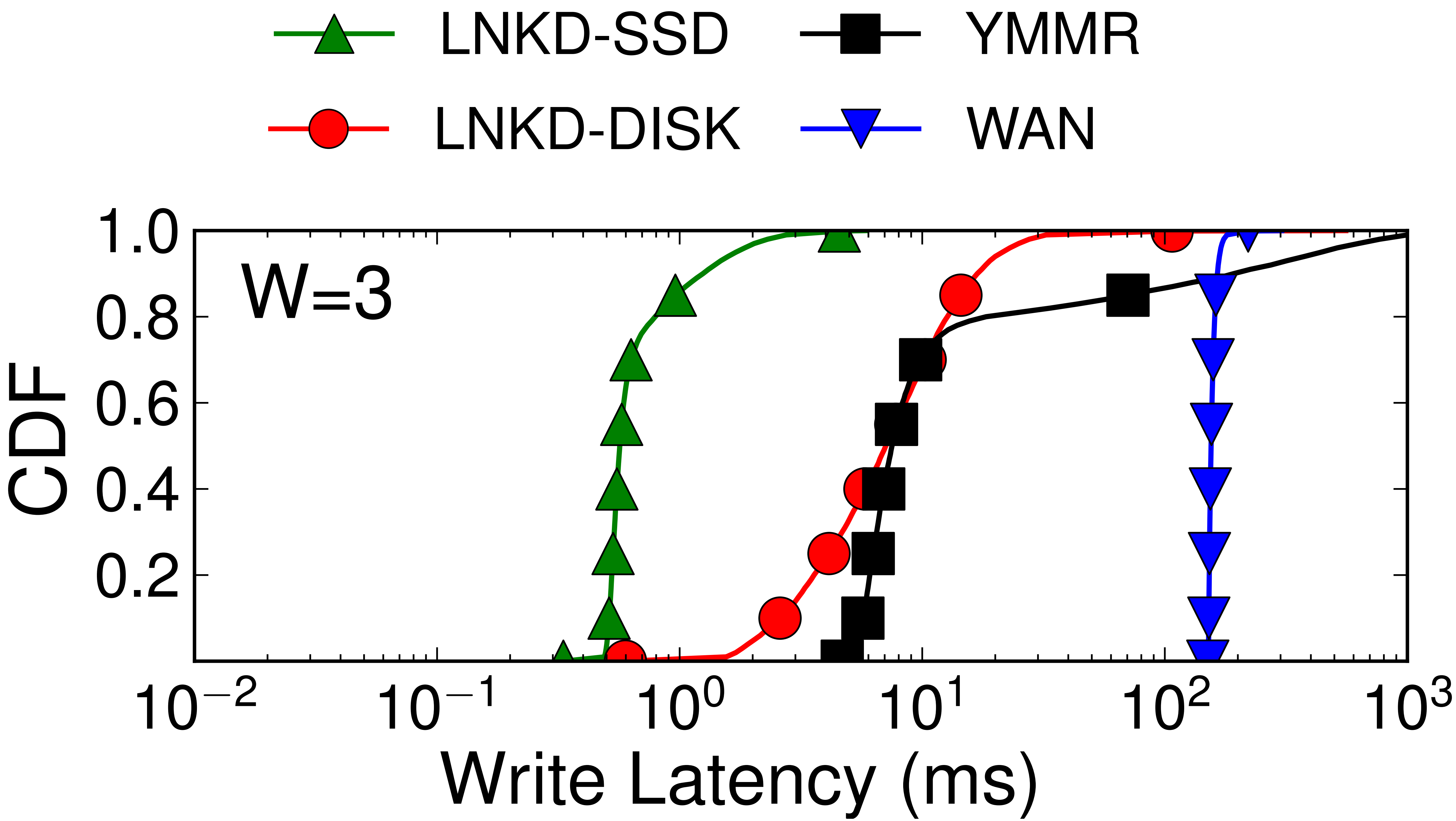
⇒ bad 8-hour rolling average

what time interval?

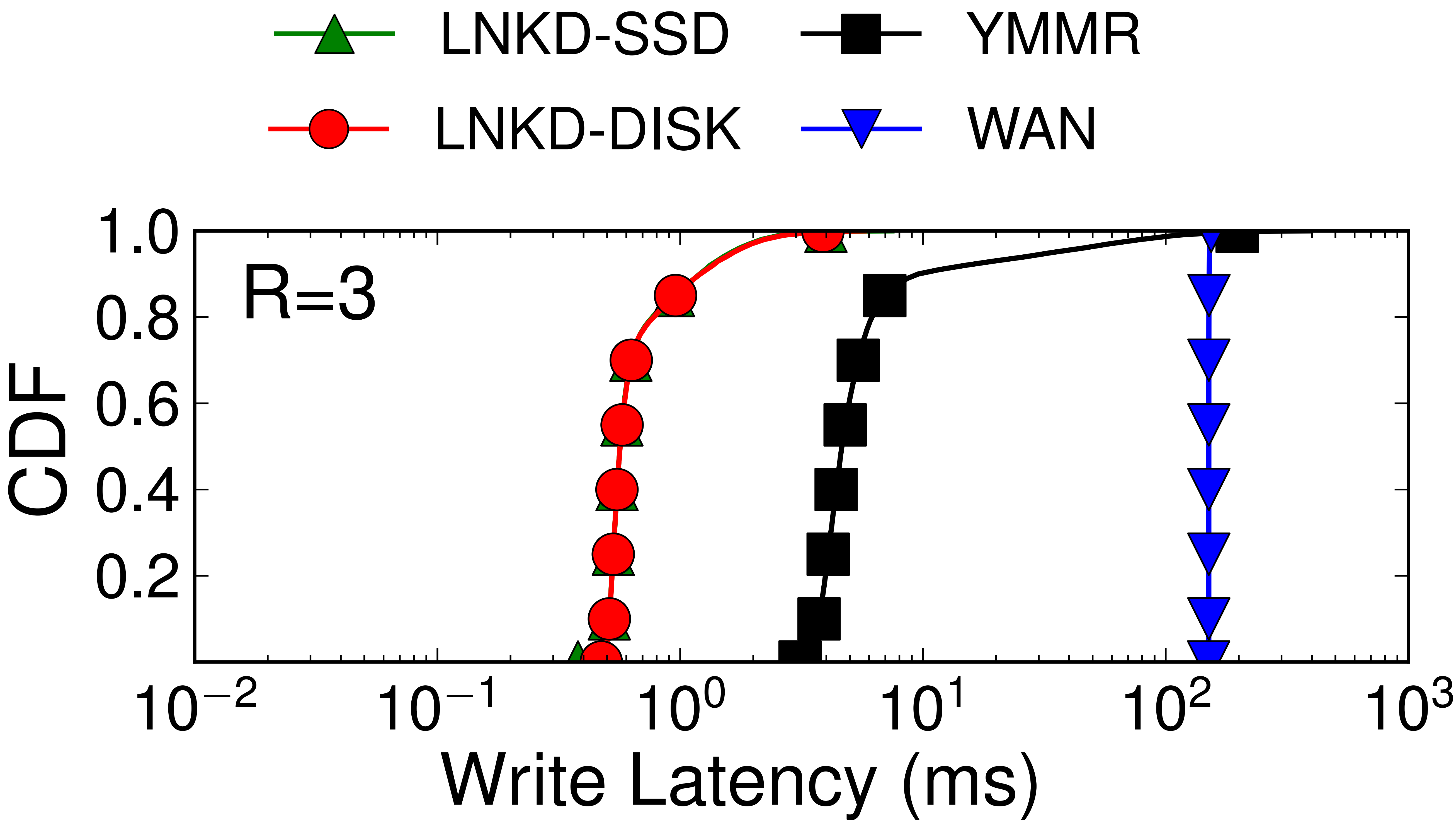
99.9% uptime/yr
⇒ 8.76 hours downtime/yr

8.76 consecutive hours down
⇒ bad 8-hour rolling average

hide in tail of distribution OR
continuously evaluate SLA, adjust


LNKD-SSD
YMMR
LNKD-DISK
WAN
W=3
CDF
1.0
0.8
0.6
0.4
0.2
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Write Latency (ms)

N=3



N=3 *(LNKD-SSD and LNKD-DISK identical for reads)*

Probabilistic quorum systems

$$p_{\text{inconsistent}} = \frac{\binom{N-W}{R}}{\binom{N}{R}}$$

How consistent?

k-staleness: probability *p* of reading one of last *k* versions

How consistent?

$$1-\left(\frac{\left(\frac{N-W}{R}\right)}{\left(\frac{N}{R}\right)}\right)^{K}$$

How consistent?

$$1 - \left(\frac{\left(\frac{N-W}{R} \right)}{\left(\frac{N}{R} \right)} \right)^{K}$$

$\langle k, t \rangle$-staleness:

versions and time

<*k*,*t*>-staleness:

versions and time

approximation:

exponentiate

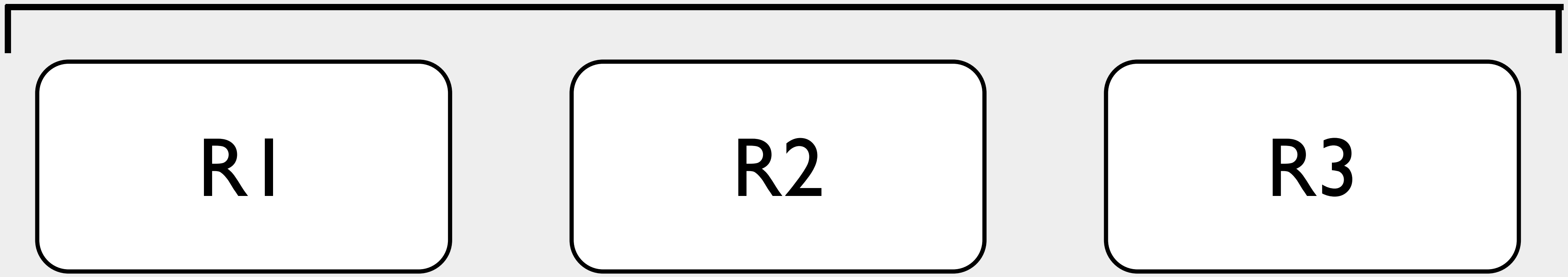
t-staleness by *k*

“strong” consistency ≡

reads return the last written value or newer

(defined w.r.t. real time, when the read started)

N = 3 replicas



Write to W, read from R replicas

N = 3 replicas

R1 R2 R3

Write to W, read from R replicas

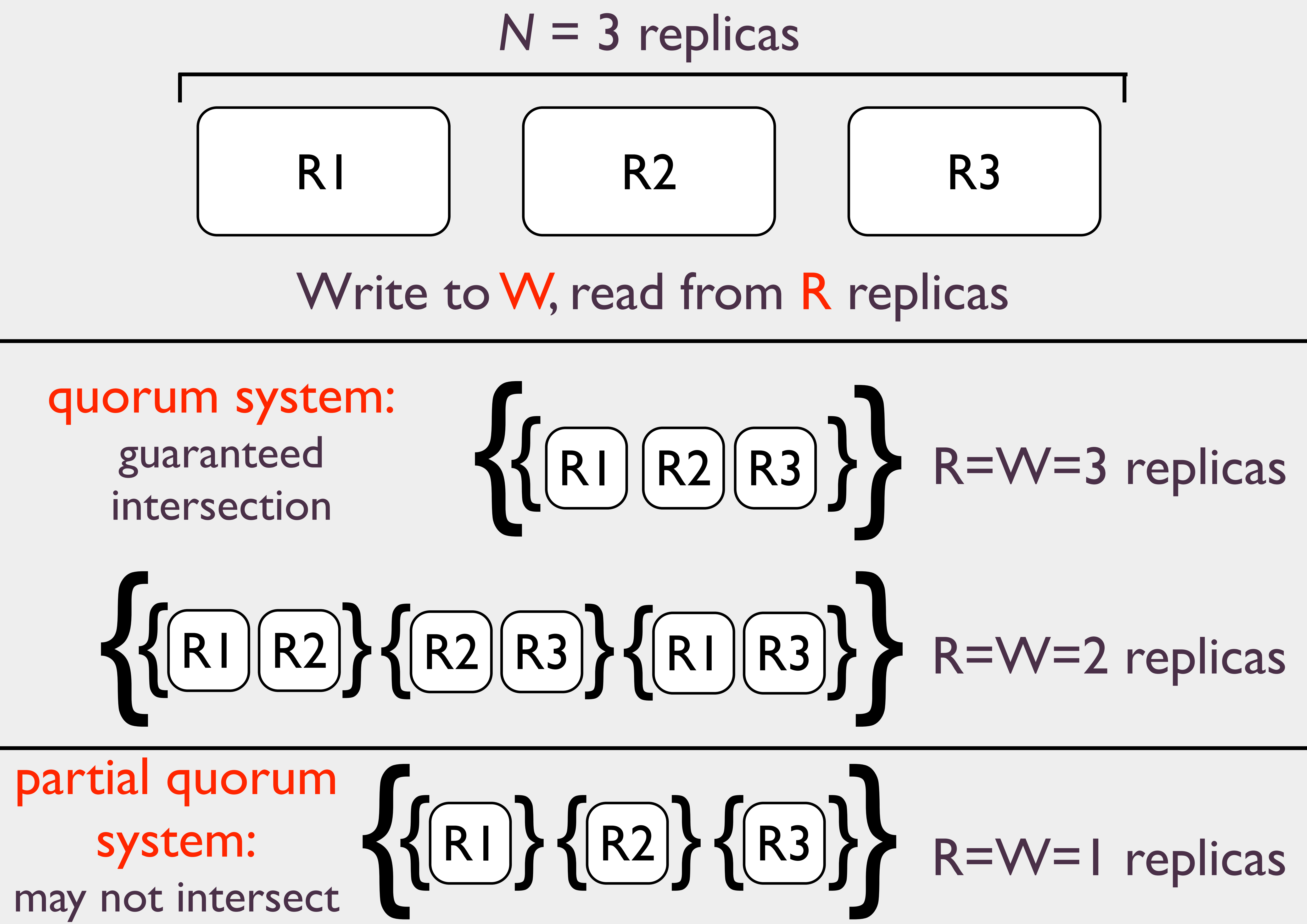
---

quorum system:

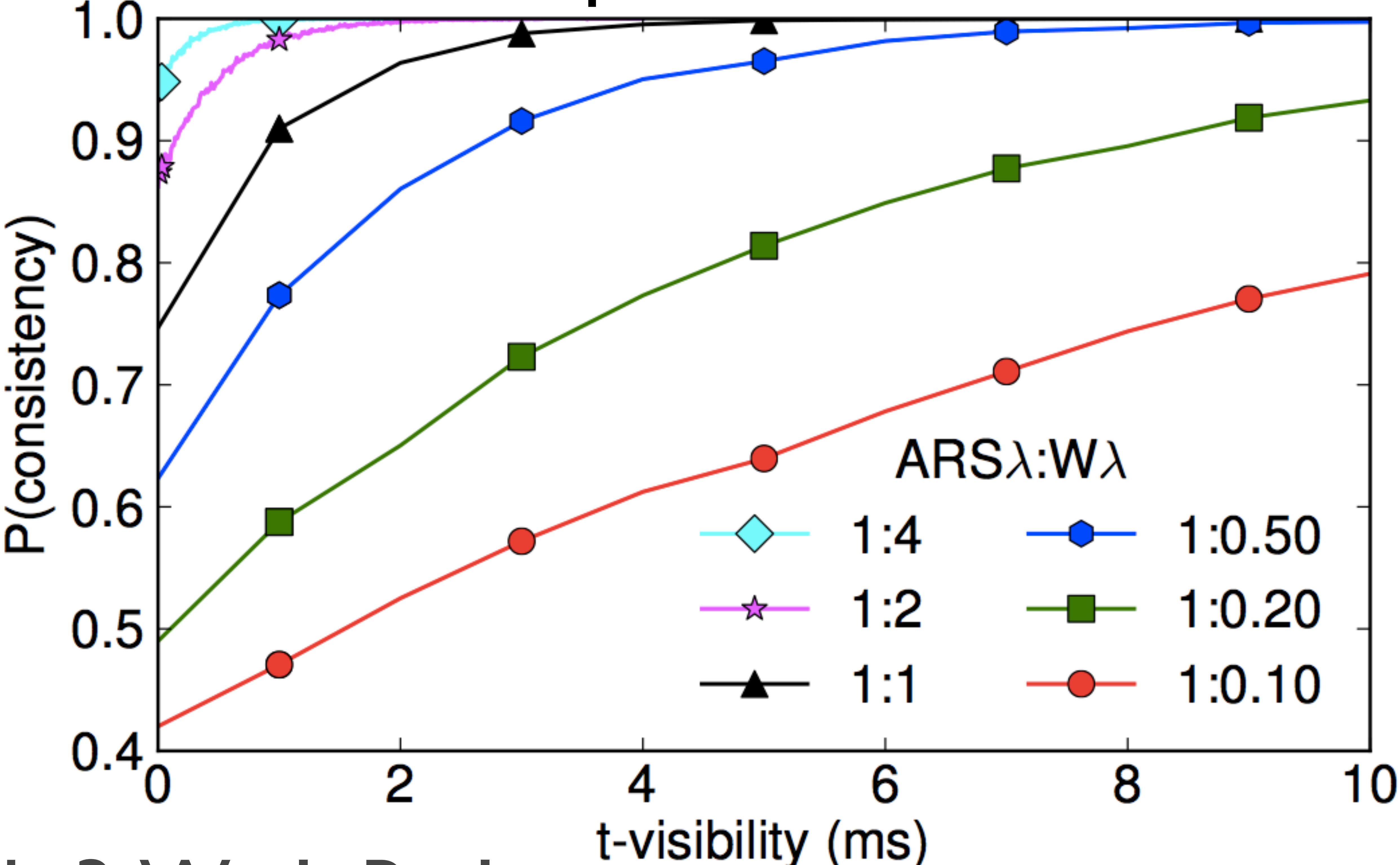
guaranteed intersection

{{R1 R2 R3}} R=W=3 replicas

{{R1 R2} {R2 R3} {R1 R3}} R=W=2 replicas

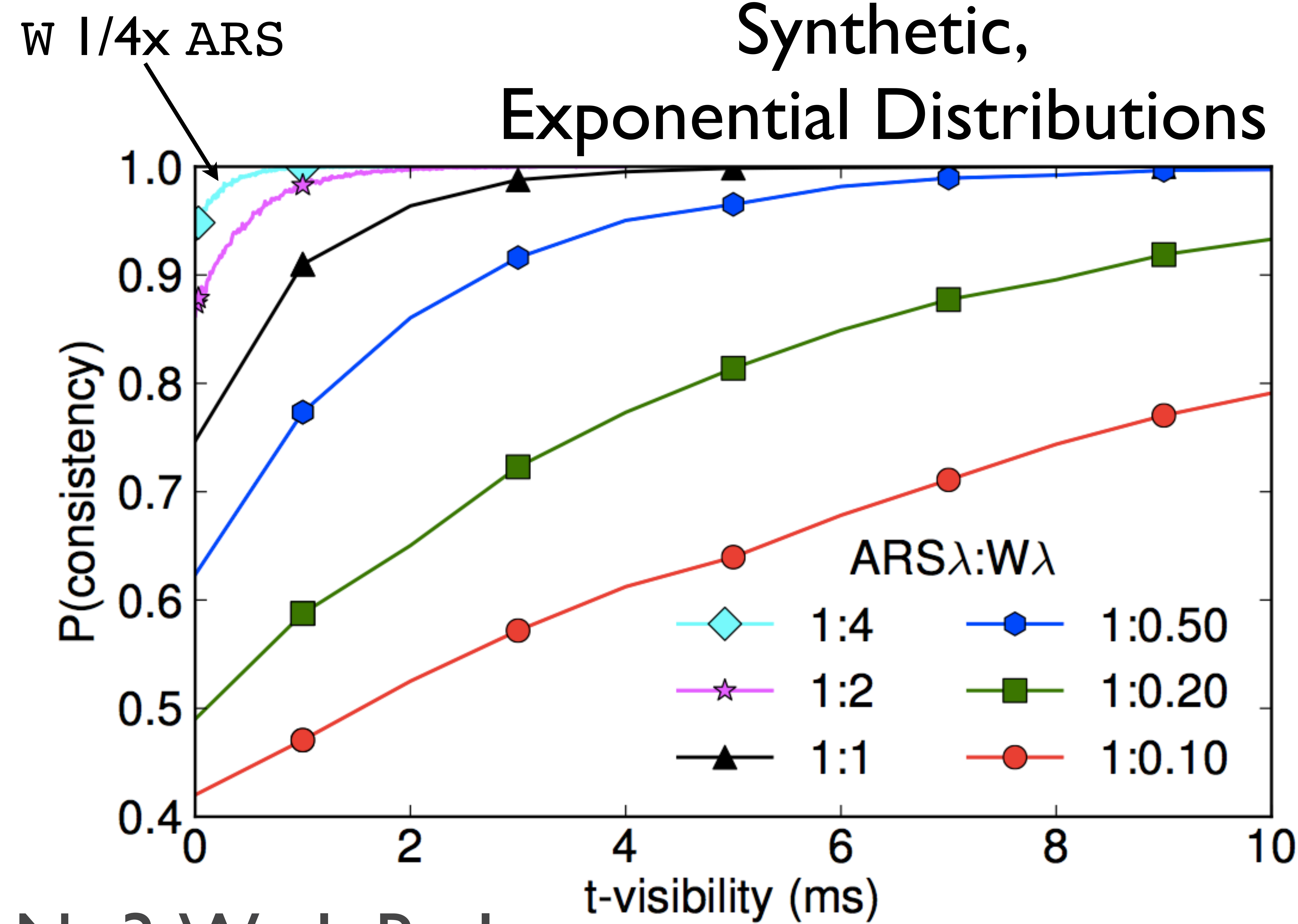

N = 3 replicas
R1
R2
R3
Write to W, read from R replicas
quorum system:
guaranteed intersection
R1 R2 R3
R=W=3 replicas
R1 R2 R2 R3 R1 R3
R=W=2 replicas
partial quorum system:
may not intersect
R1 R2 R3
R=W=1 replicas

Synthetic, Exponential Distributions



N=3, W=1, R=1

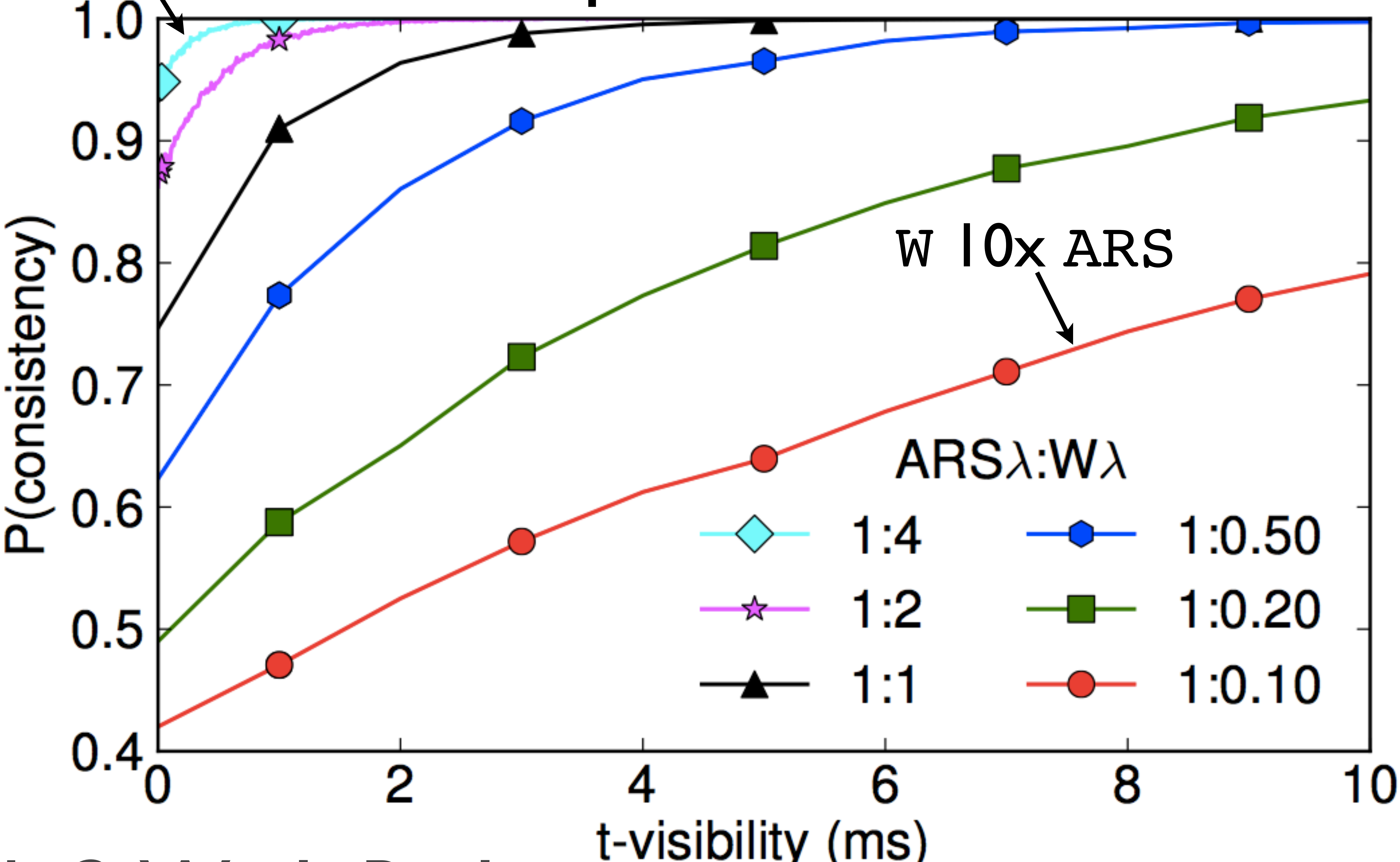
Synthetic,
Exponential Distributions



N=3, W=1, R=1

Synthetic, Exponential Distributions

W 1/4x ARS



N=3, W=1, R=1

concurrent writes:

deterministically choose

("key", 1) ("key", 2)

Coordinator *R*=2

%ile	Latency (ms)
15,000 RPM SAS Disk	
Average	4.85
95	15
99	25
Commodity SSD	
Average	0.58
95	1
99	2

Table 1: LinkedIn Voldemort single-node production latencies.

%ile	Read Latency (ms)	Write Latency (ms)
Min	1.55	1.68
50	3.75	5.73
75	4.17	6.50
95	5.2	8.48
98	6.045	10.36
99	6.59	131.73
99.9	32.89	435.83
Max	2979.85	4465.28
Mean	9.23	8.62
Std. Dev.	83.93	26.10
Mean Rate	718.18 gets/s	45.65 puts/s

Table 2: Yammer Riak N=3, R=2, W=2 production latencies.

LNKD-SSD	W = A = R = S : 91.22%: Pareto, $x_m = .235, \alpha = 10$ 8.78%: Exponential, $\lambda = 1.66$ N-RMSE: .55%
LNKD-DISK	W: 38%: Pareto, $x_m = 1.05, \alpha = 1.51$ 62%: Exponential, $\lambda = .183$ N-RMSE: .26%
	A = R = S : LNKD-SSD
YMMR	W: 93.9%: Pareto, $x_m = 3, \alpha = 3.35$ 6.1%: Exponential, $\lambda = .0028$ N-RMSE: 1.84%
	A = R = S : 98.2%: Pareto, $x_m = 1.5, \alpha = 3.8$ 1.8%: Exponential, $\lambda = .0217$ N-RMSE: .06%

Table 3: Distribution fits for production latency distributions from LinkedIn (LNKD-*) and Yammer (YMMR).

	LNKD-SSD			LNKD-DISK			YMMR			WAN		
	L_r	L_w	t	L_r	L_w	t	L_r	L_w	t	L_r	L_w	t
R=1, W=1	**0.66**	**0.66**	**1.85**	0.66	10.99	45.5	5.58	10.83	1364.0	**3.4**	**55.12**	**113.0**
R=1, W=2	0.66	1.63	1.79	0.65	20.97	43.3	5.61	427.12	1352.0	3.4	167.64	0
R=2, W=1	**1.63**	**0.65**	**0**	**1.63**	**10.9**	**13.6**	**32.6**	**10.73**	**202.0**	151.3	56.36	30.2
R=2, W=2	**1.62**	**1.64**	**0**	1.64	20.96	0	33.18	428.11	0	151.31	167.72	0
R=3, W=1	4.14	0.65	0	**4.12**	**10.89**	**0**	**219.27**	**10.79**	**0**	**153.86**	**55.19**	**0**
R=1, W=3	0.65	4.09	0	0.65	112.65	0	5.63	1870.86	0	3.44	241.55	0