Probabilistically Bounded Staleness

How Eventual is Eventual Consistency?

Peter Bailis, Shivaram Venkataraman,
Michael J. Franklin, Joseph M. Hellerstein, Ion Stoica (UC Berkeley)

BashoChats 002, 28 February 2012

PBS

what: consistency prediction

why: weak consistency is fast

how: measure latencies
use *WARS* model

1. Fast
2. Scalable
3. Available

solution:

replicate for

1. capacity
2. fault-tolerance

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

inconsistent

⇧ consistency, ⇧ latency

contact more replicas,

read more recent data

⇩ consistency, ⇩ latency

contact fewer replicas,

read less recent data

Dynamo:

Amazon's Highly Available Key-value Store

SOSP 2007

basho
riak

LinkedIn

Project Voldemort

Apache, DataStax

Cassandra

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)

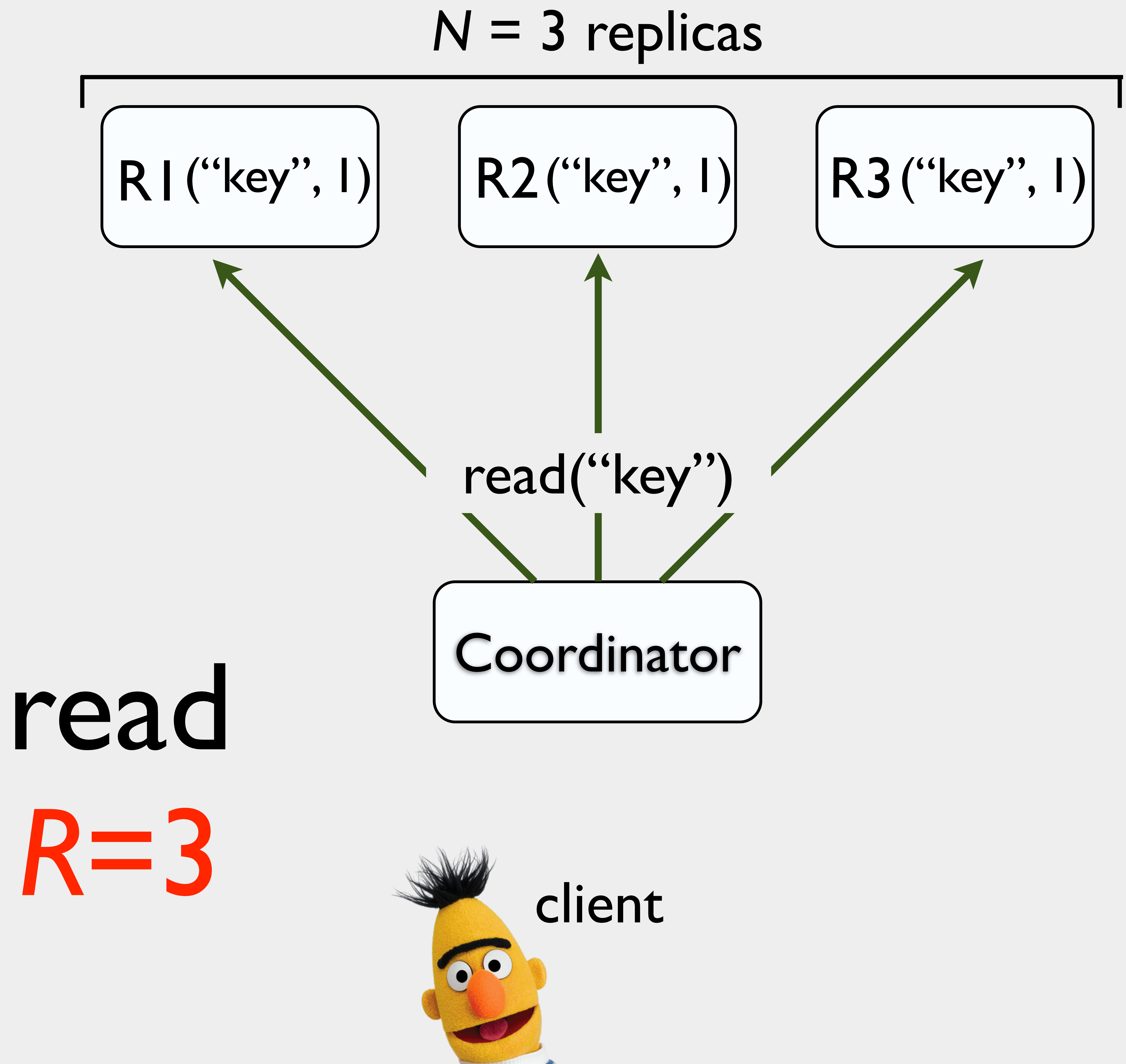
Coordinator

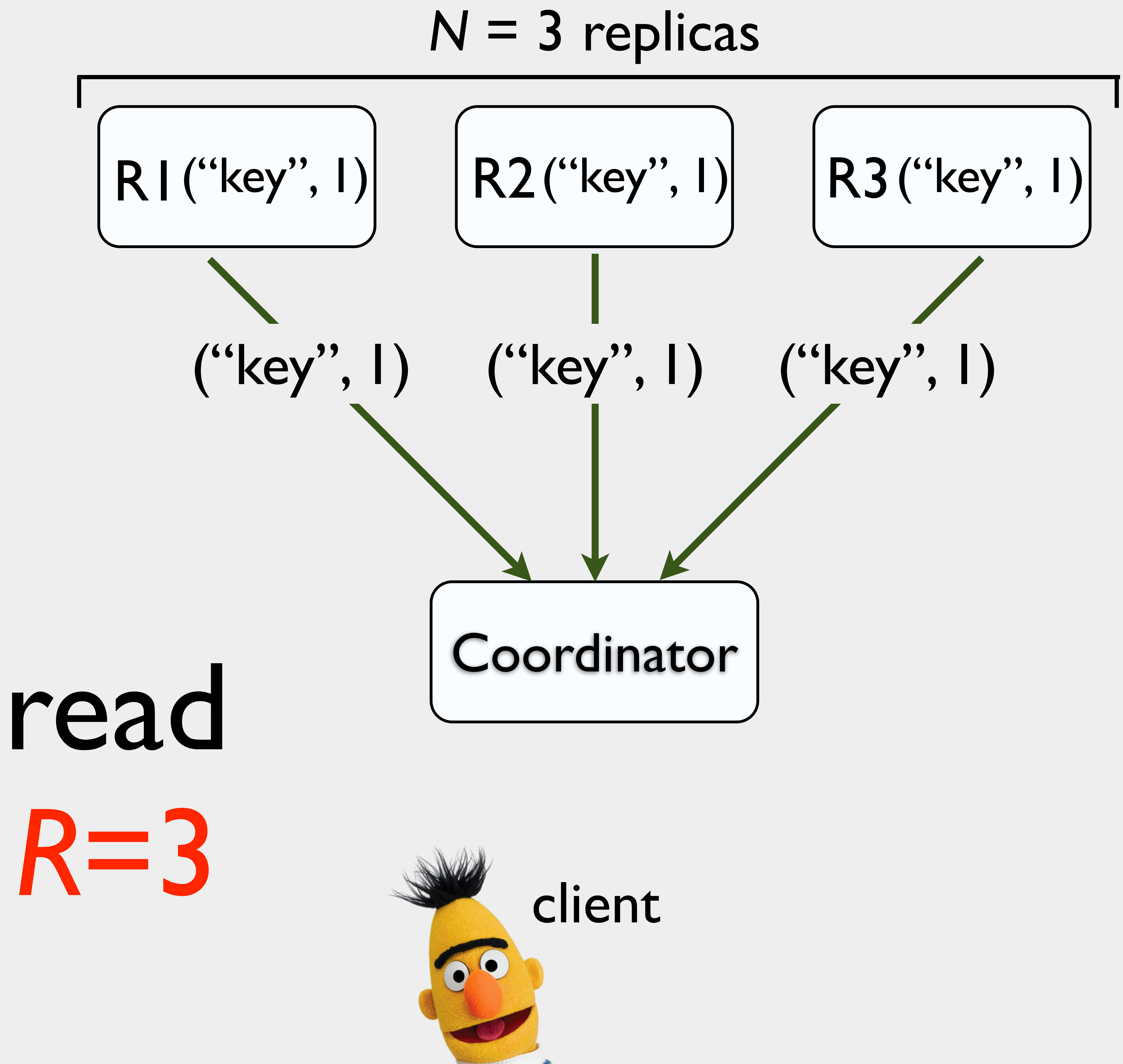
read(“key”)

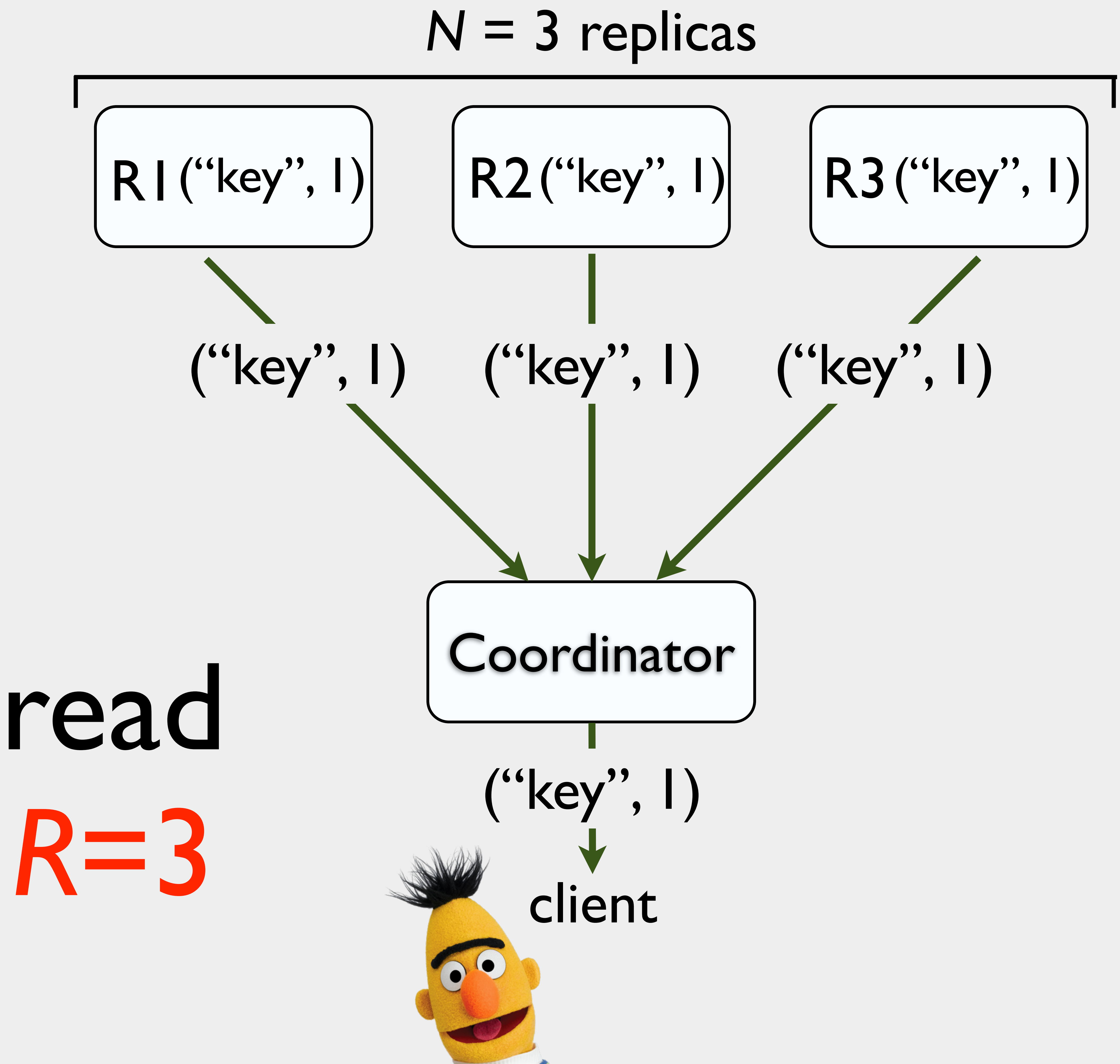
client

read

R=3


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


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


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

N = 3 replicas

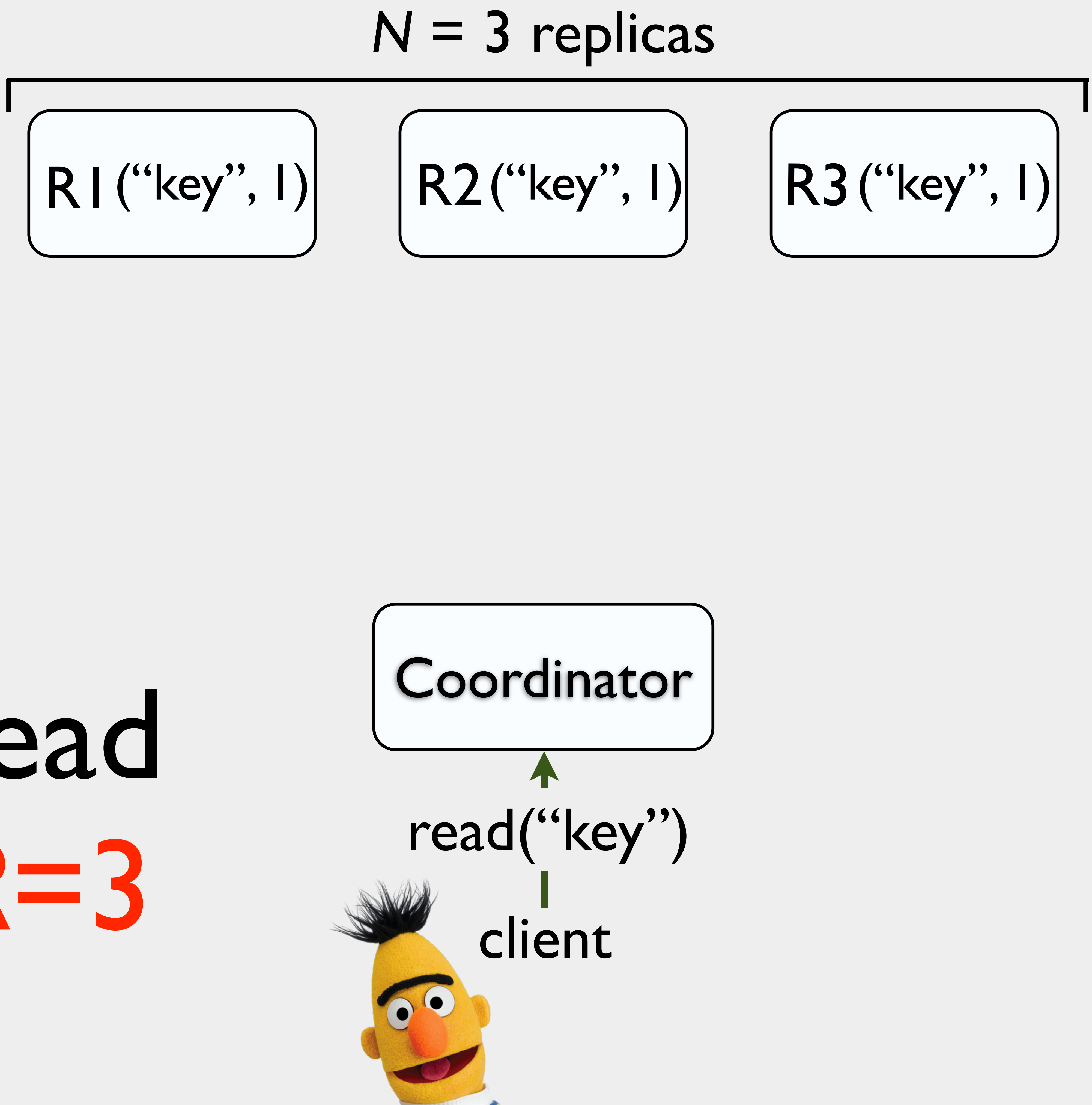
R1 (“key”, 1)   R2 (“key”, 1)   R3 (“key”, 1)

Coordinator

read

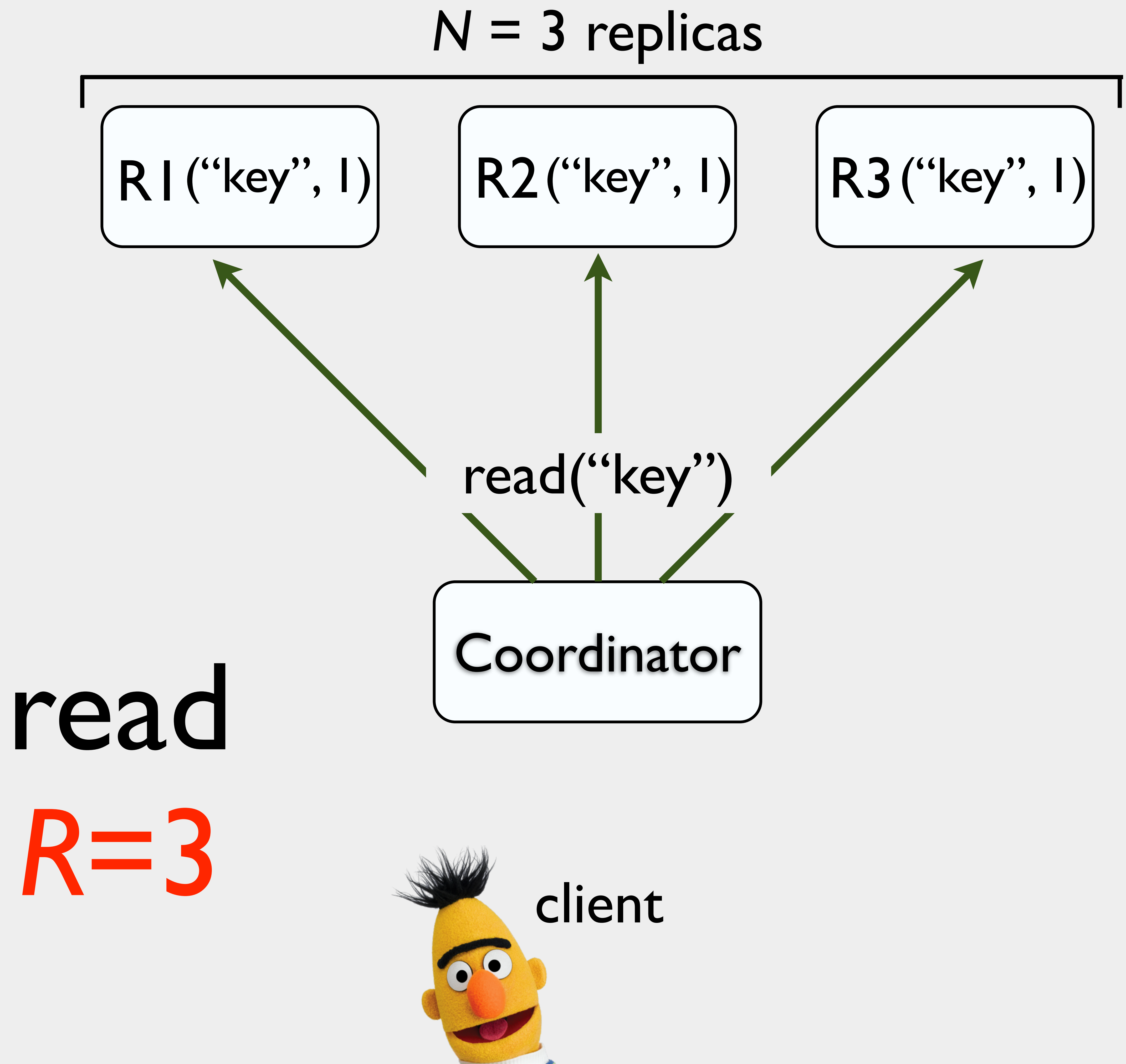
R=3

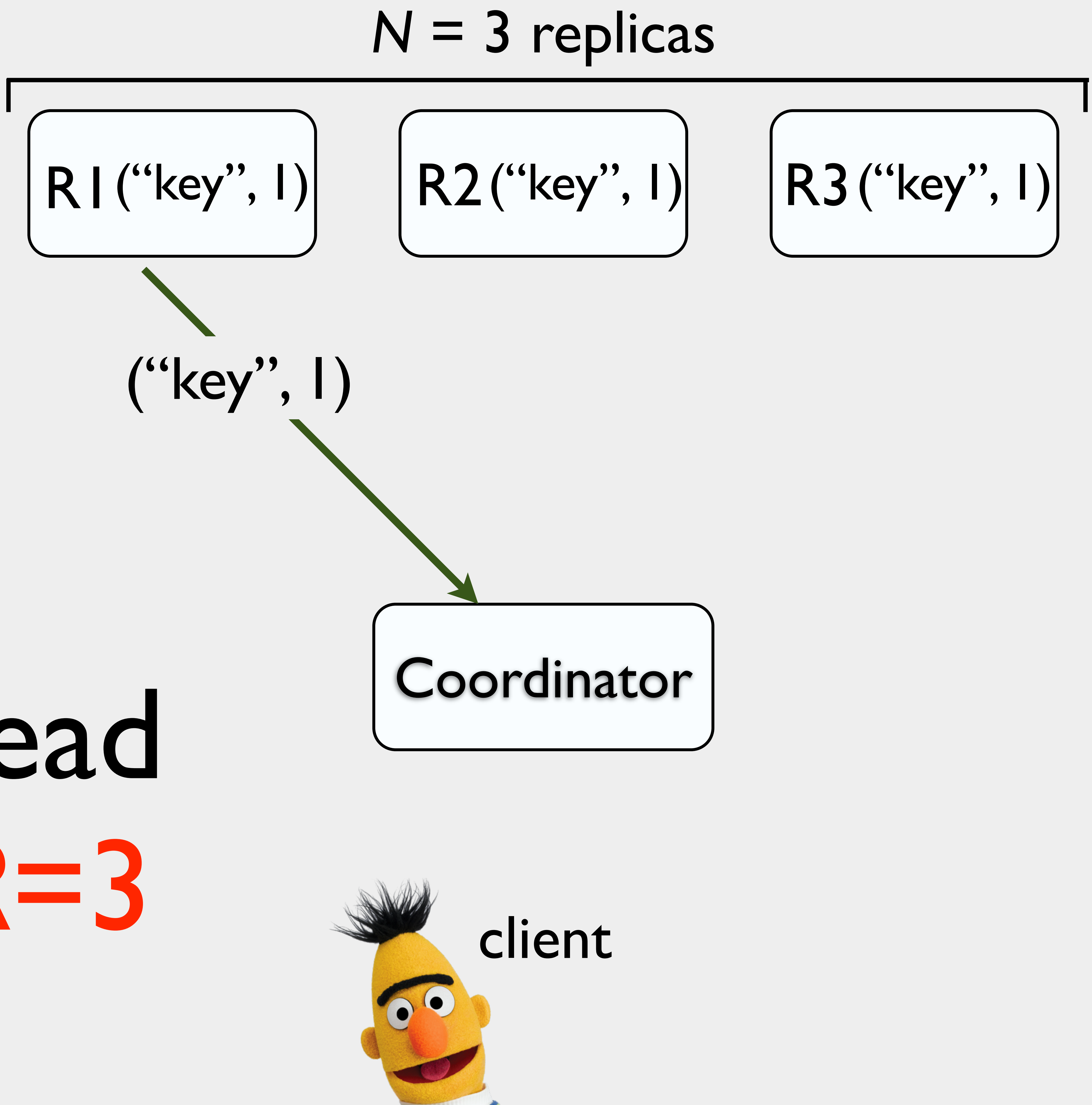
client


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

read

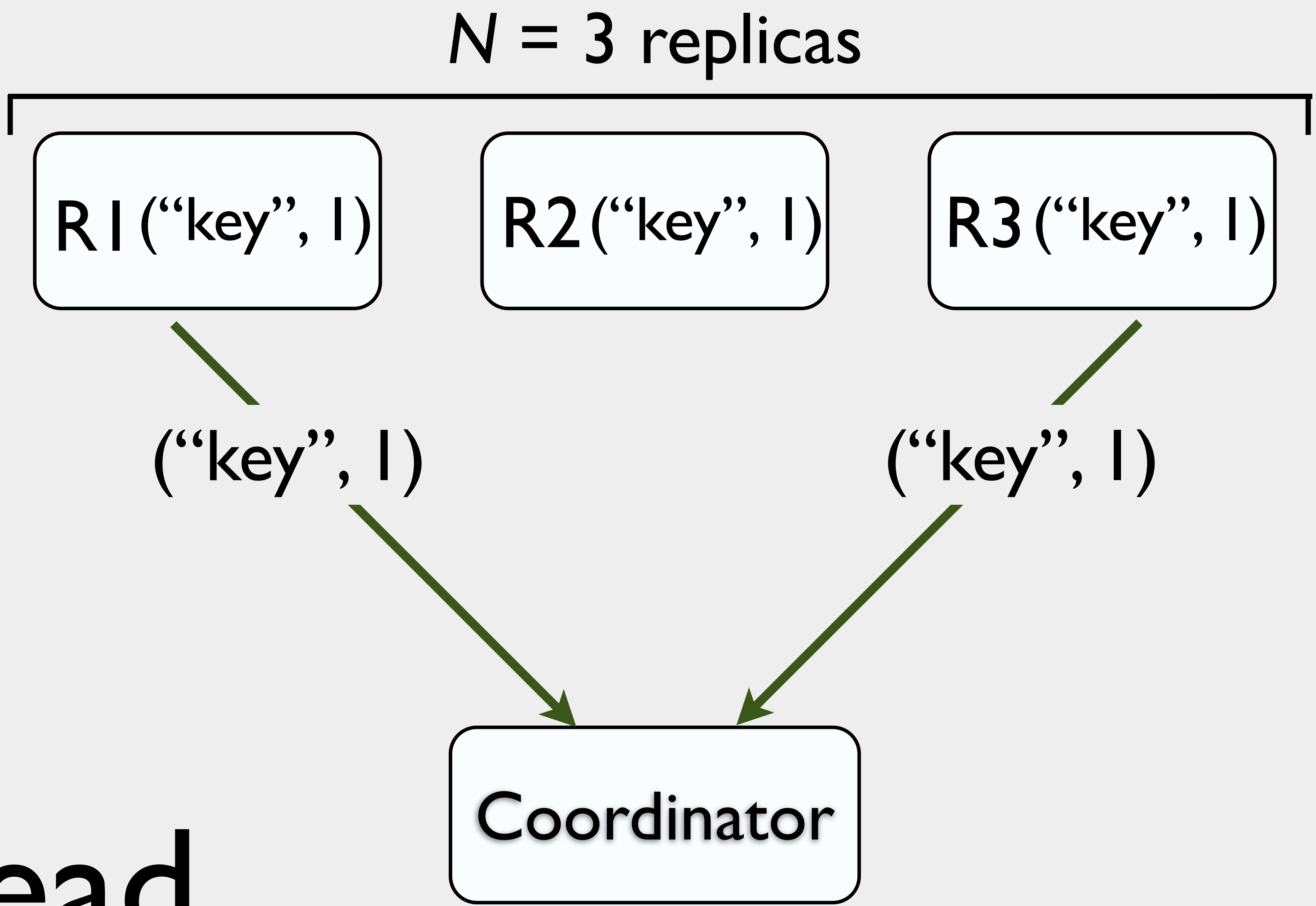
R=3


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



read

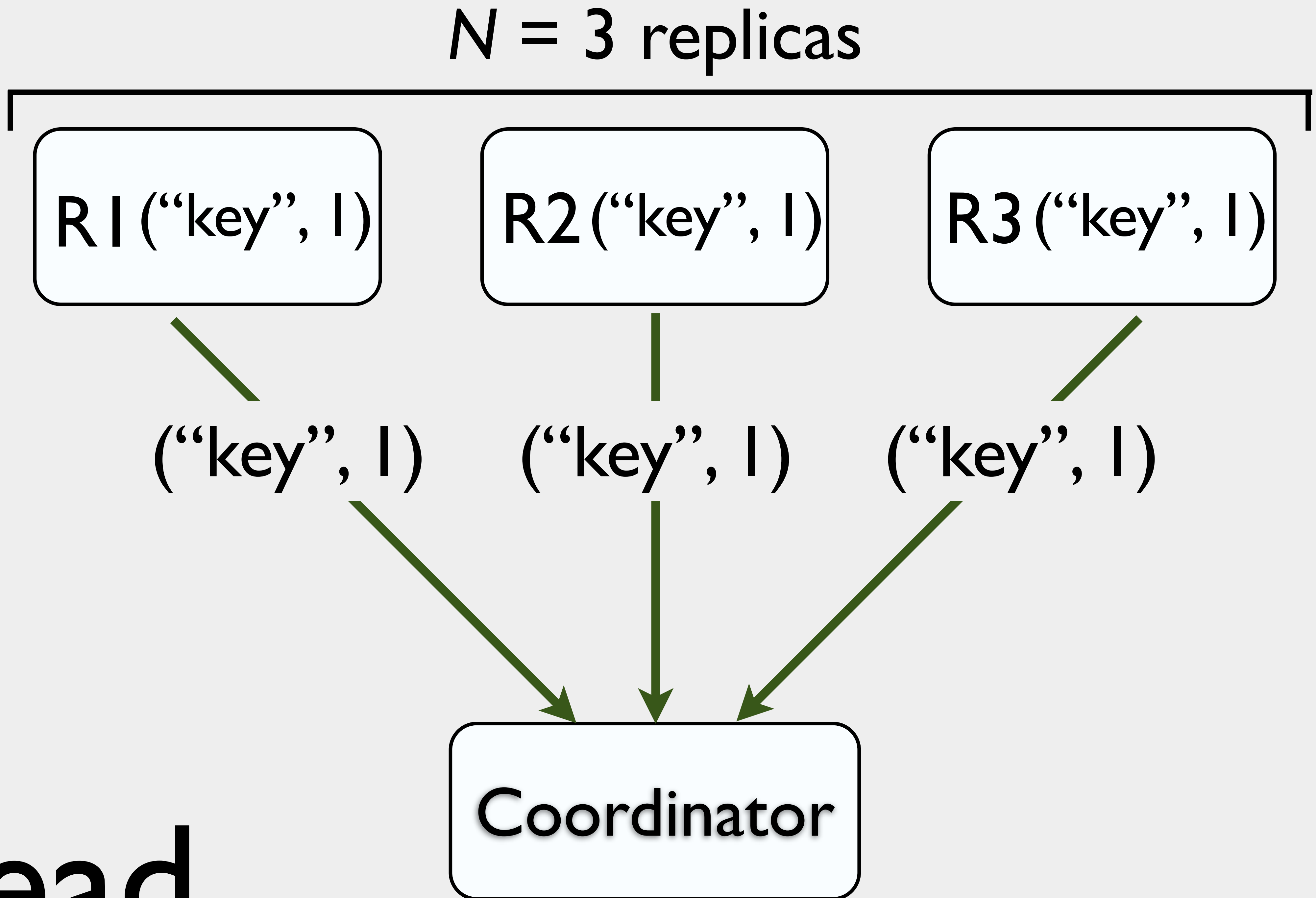
R=3



read

$R=3$

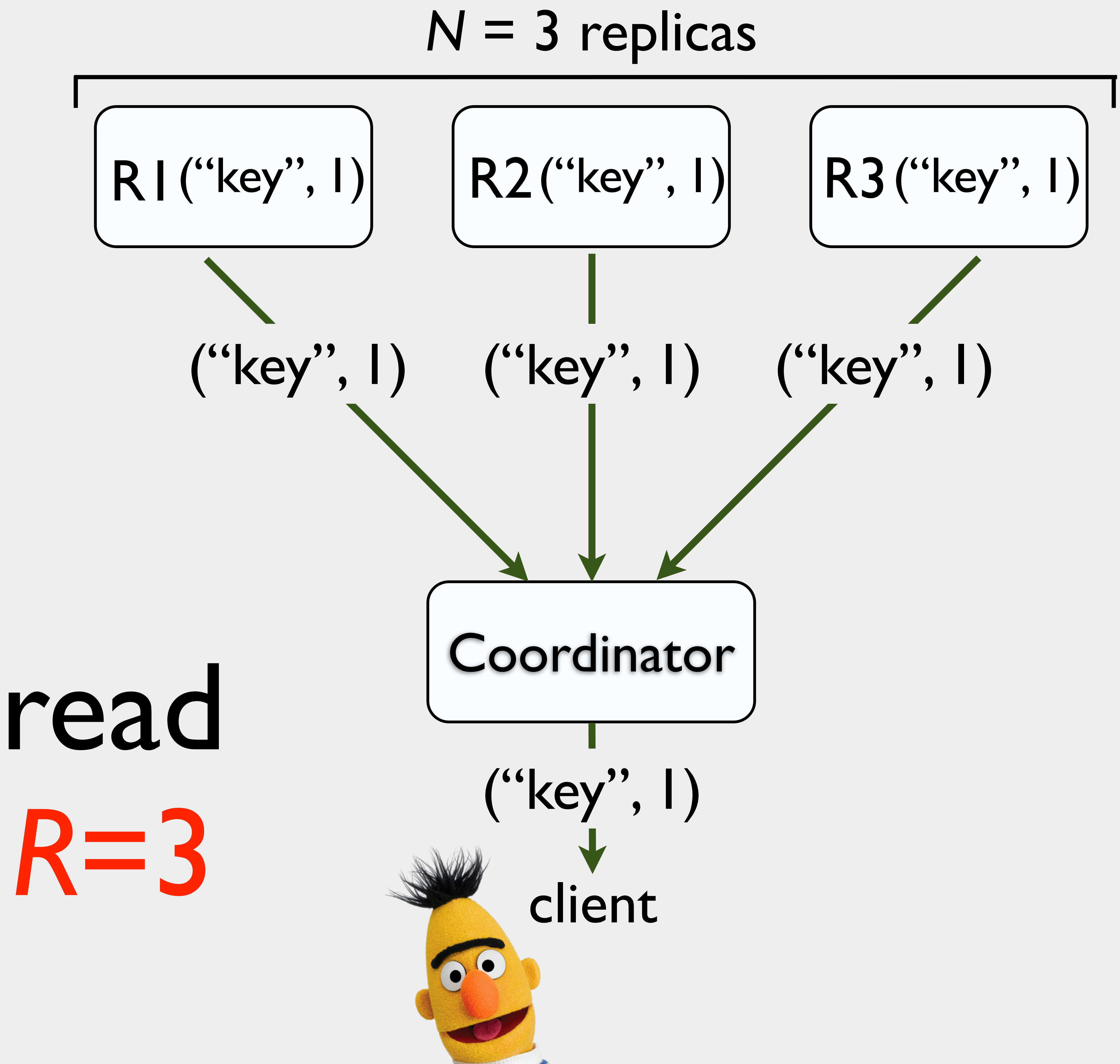
client



read

$R=3$

client


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

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)

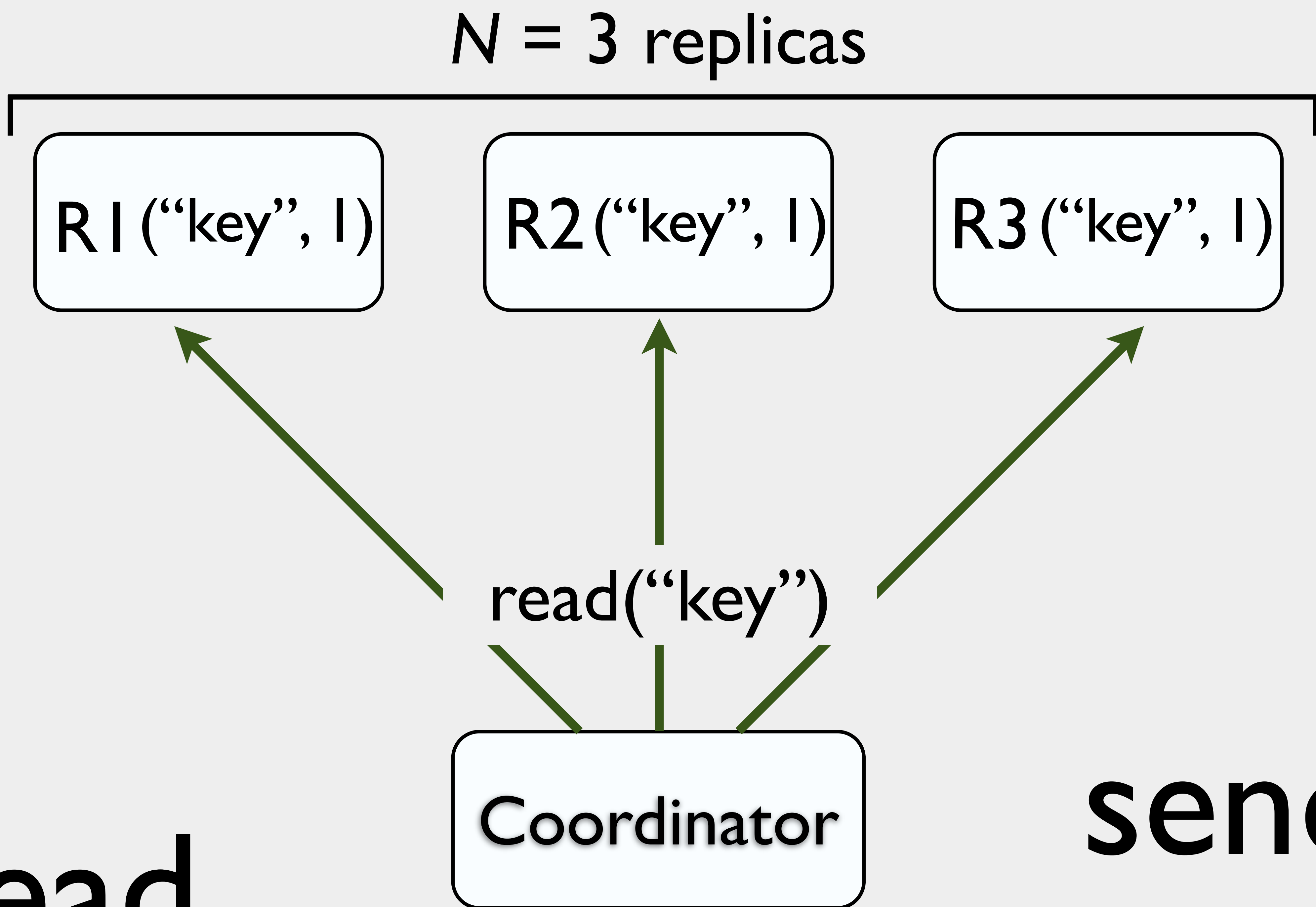
Coordinator

read("key")

client

read

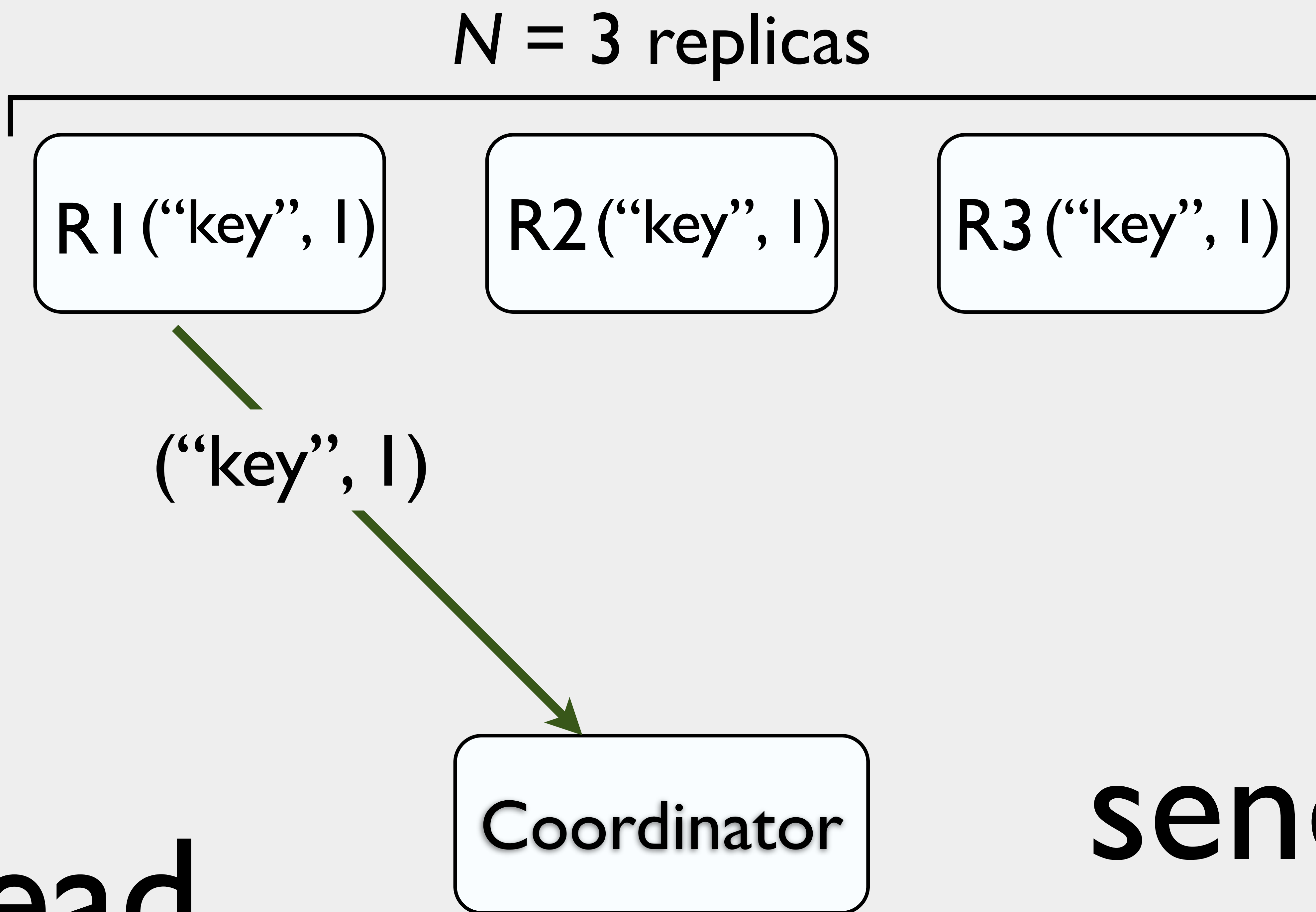
R=1



read

R=1

send read to all

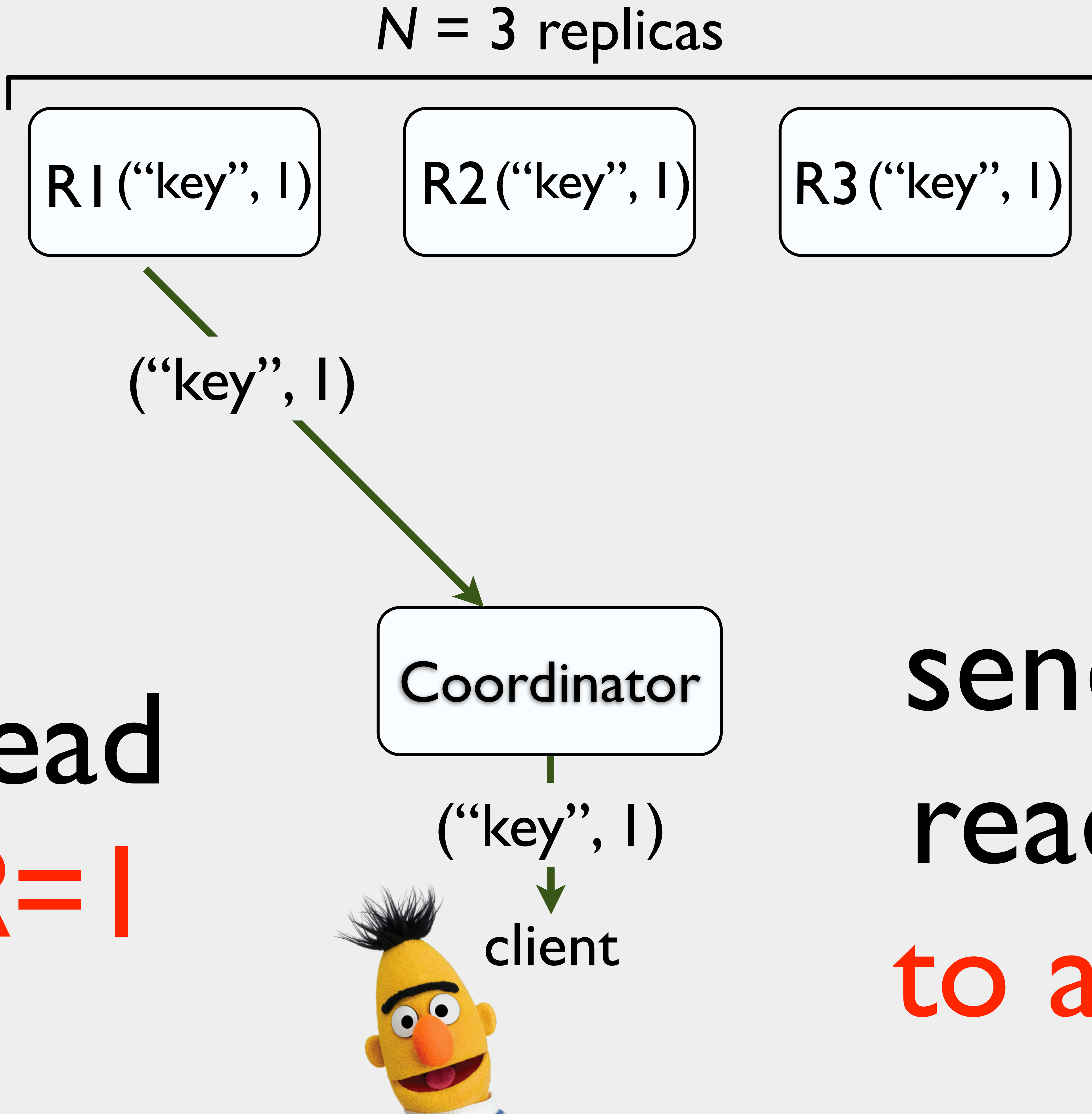


read

R=1

client

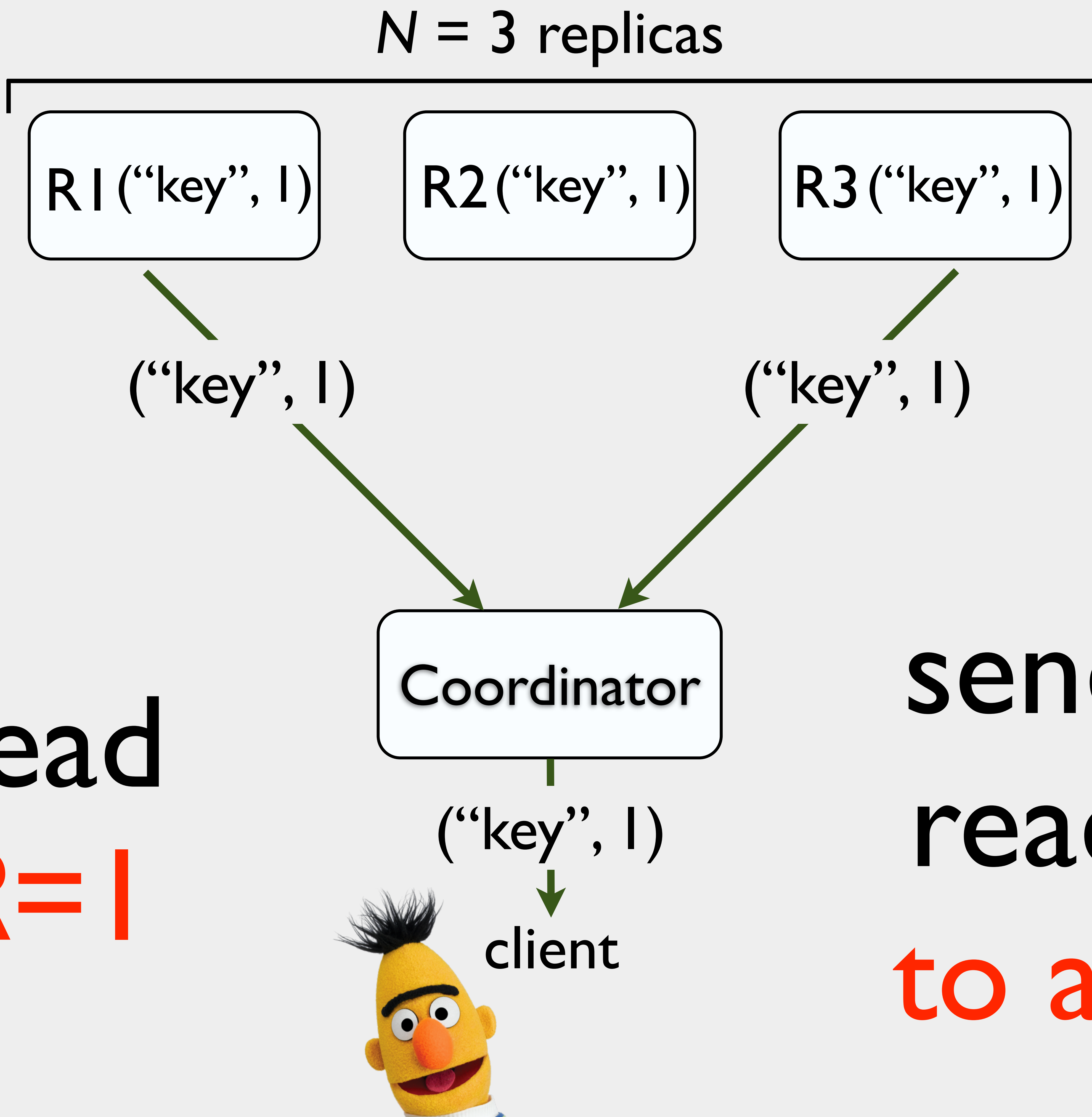
send read **to all**



read

R=1

send read to all



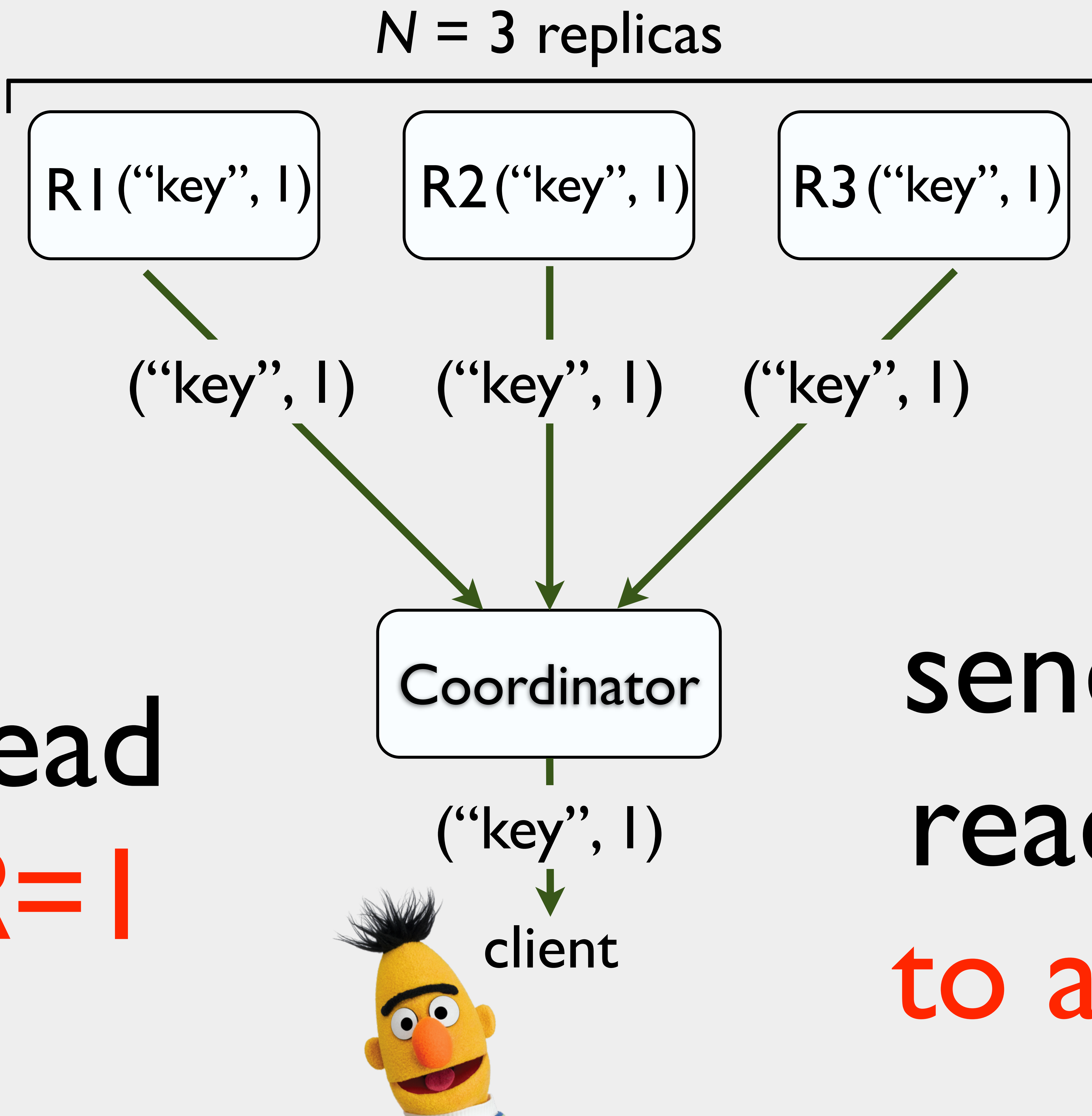
read

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

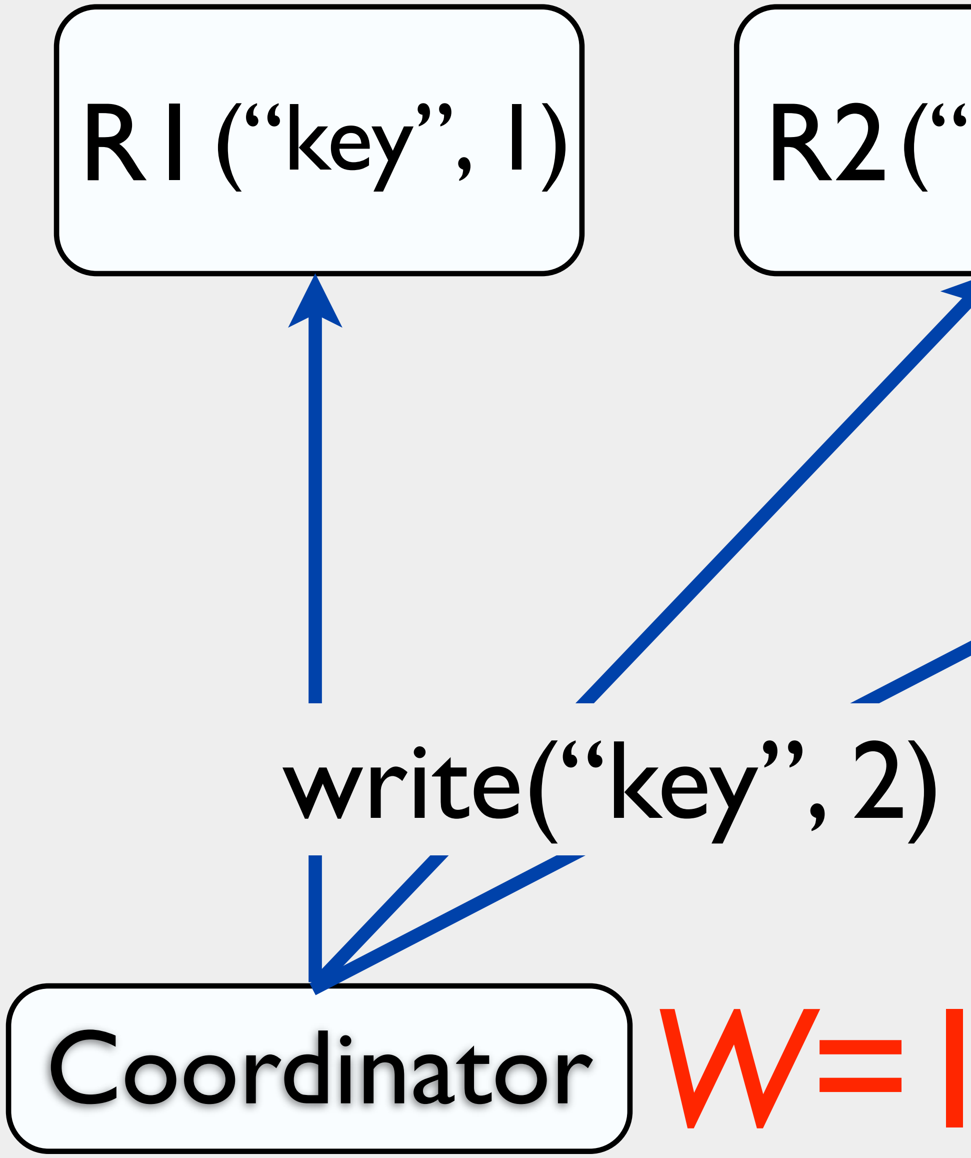
R1 (“key”, 1)

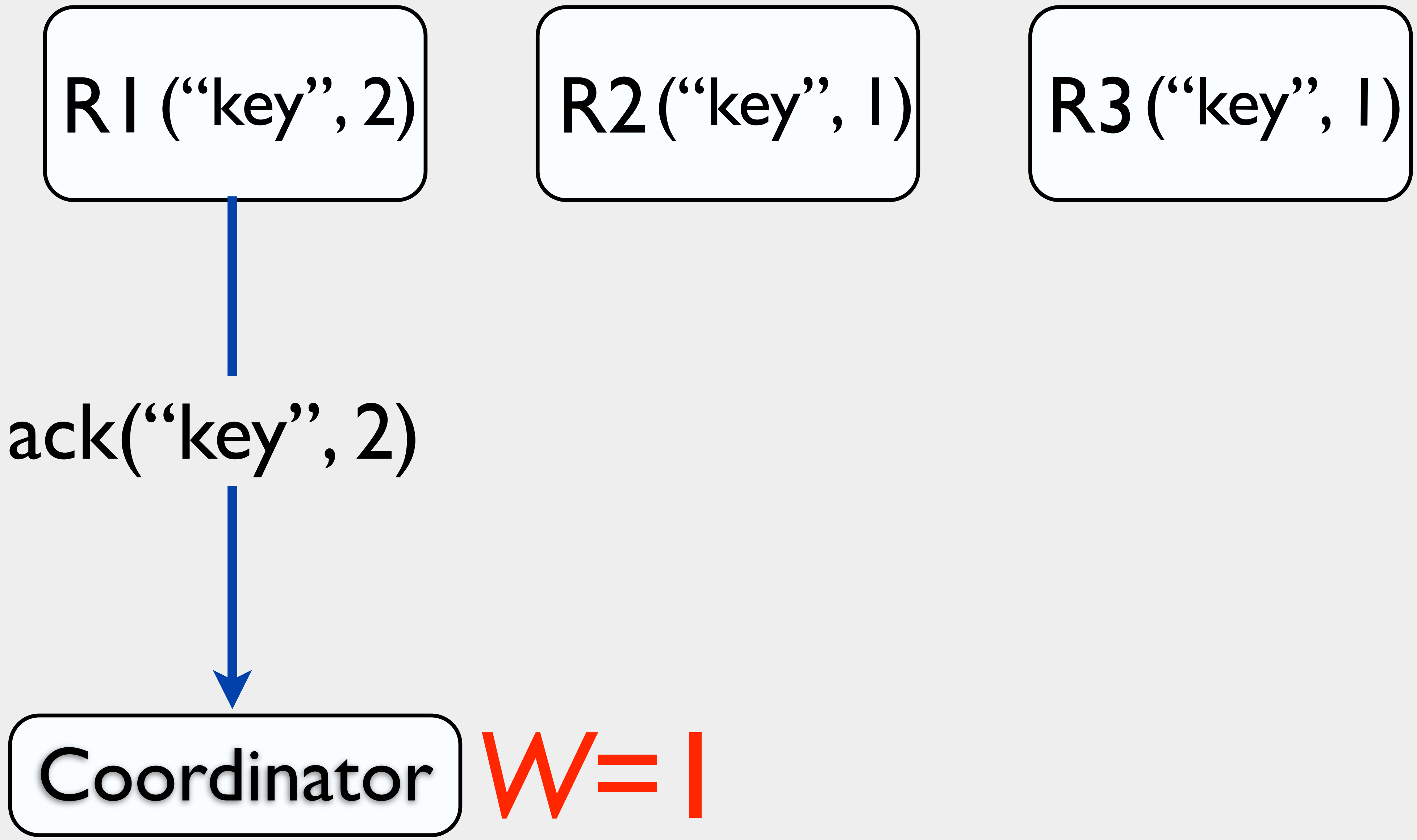
R2 (“key”, 1)

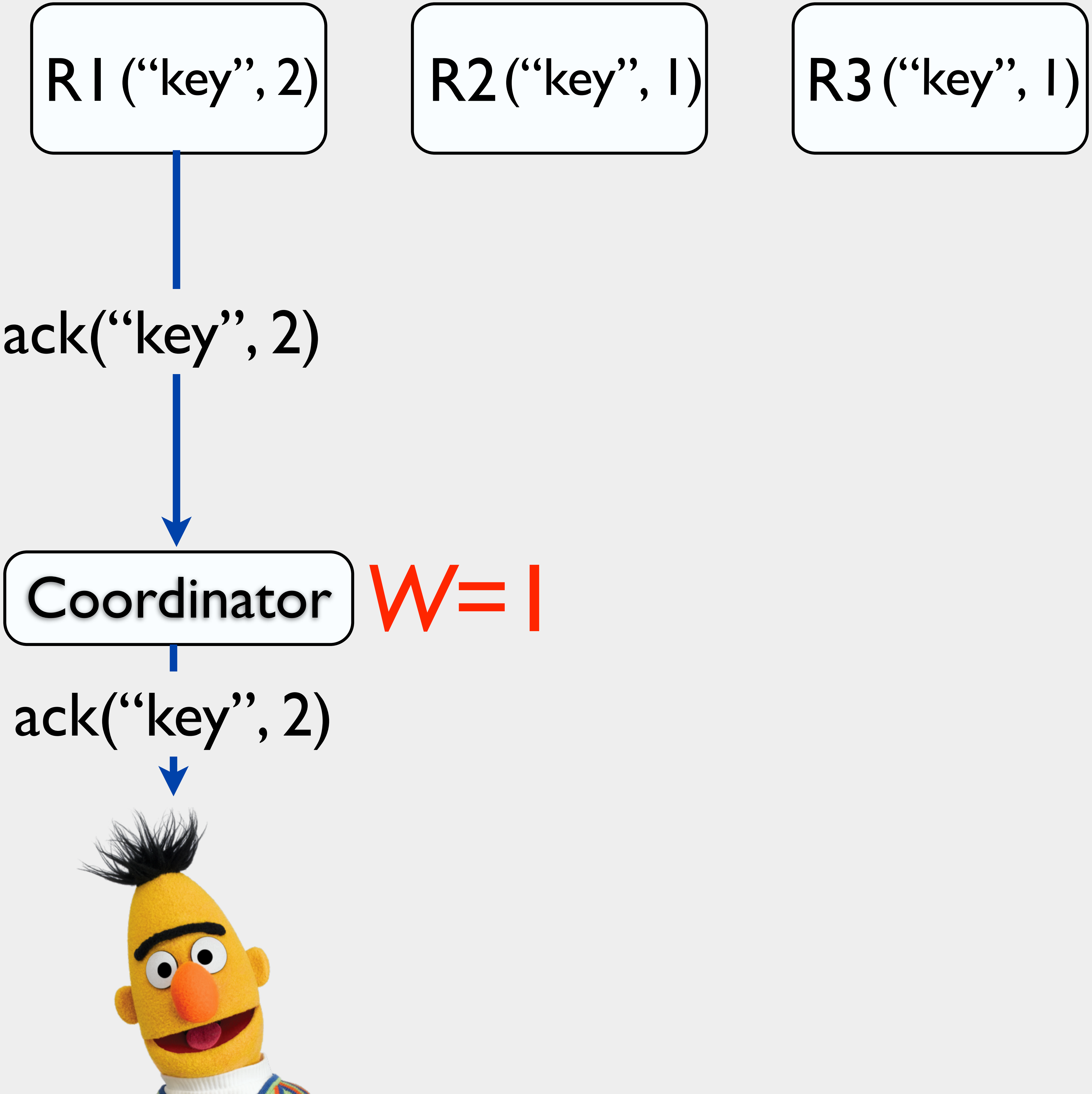
R3 (“key”, 1)

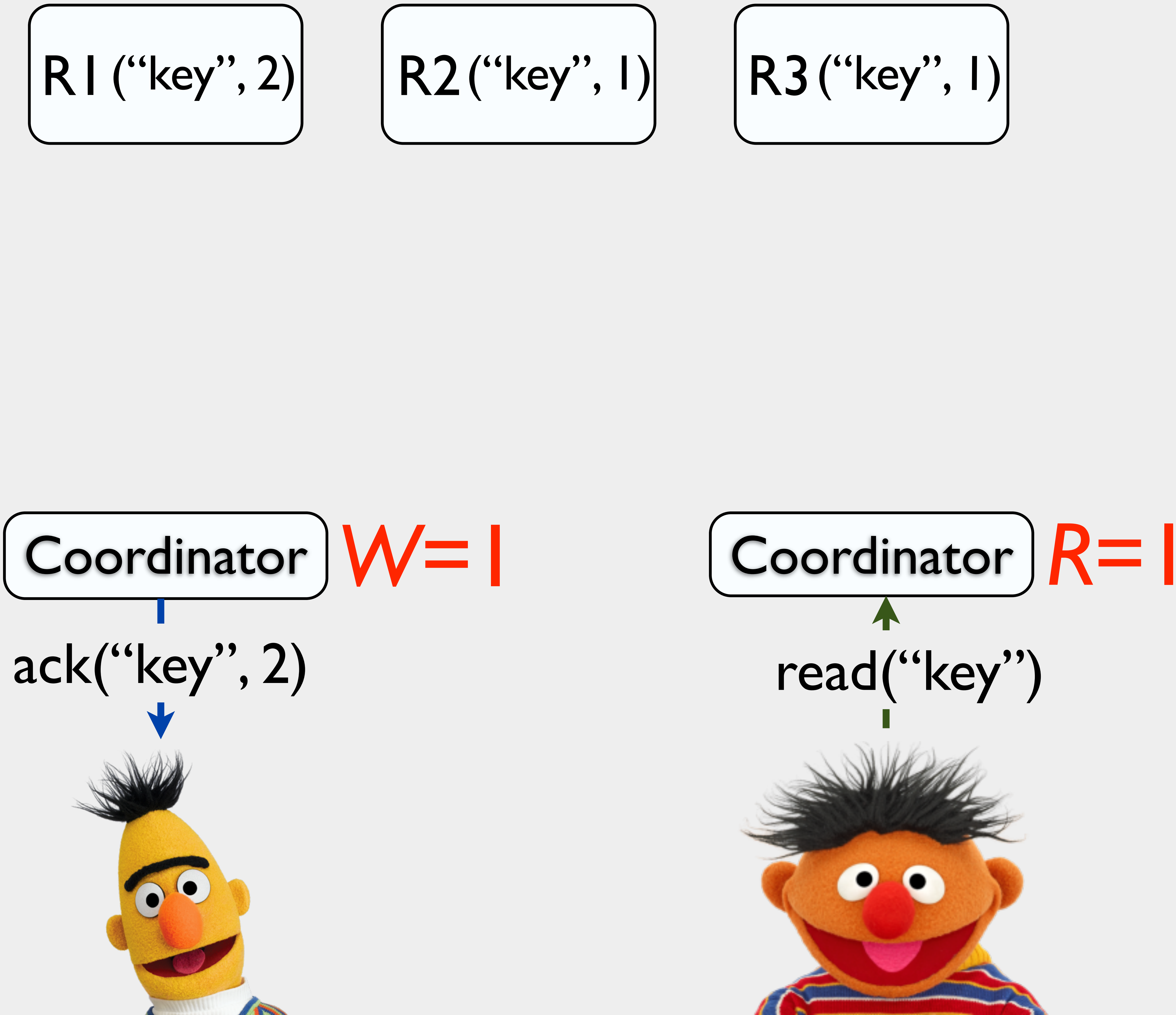
Coordinator W=1

write(“key”, 2)

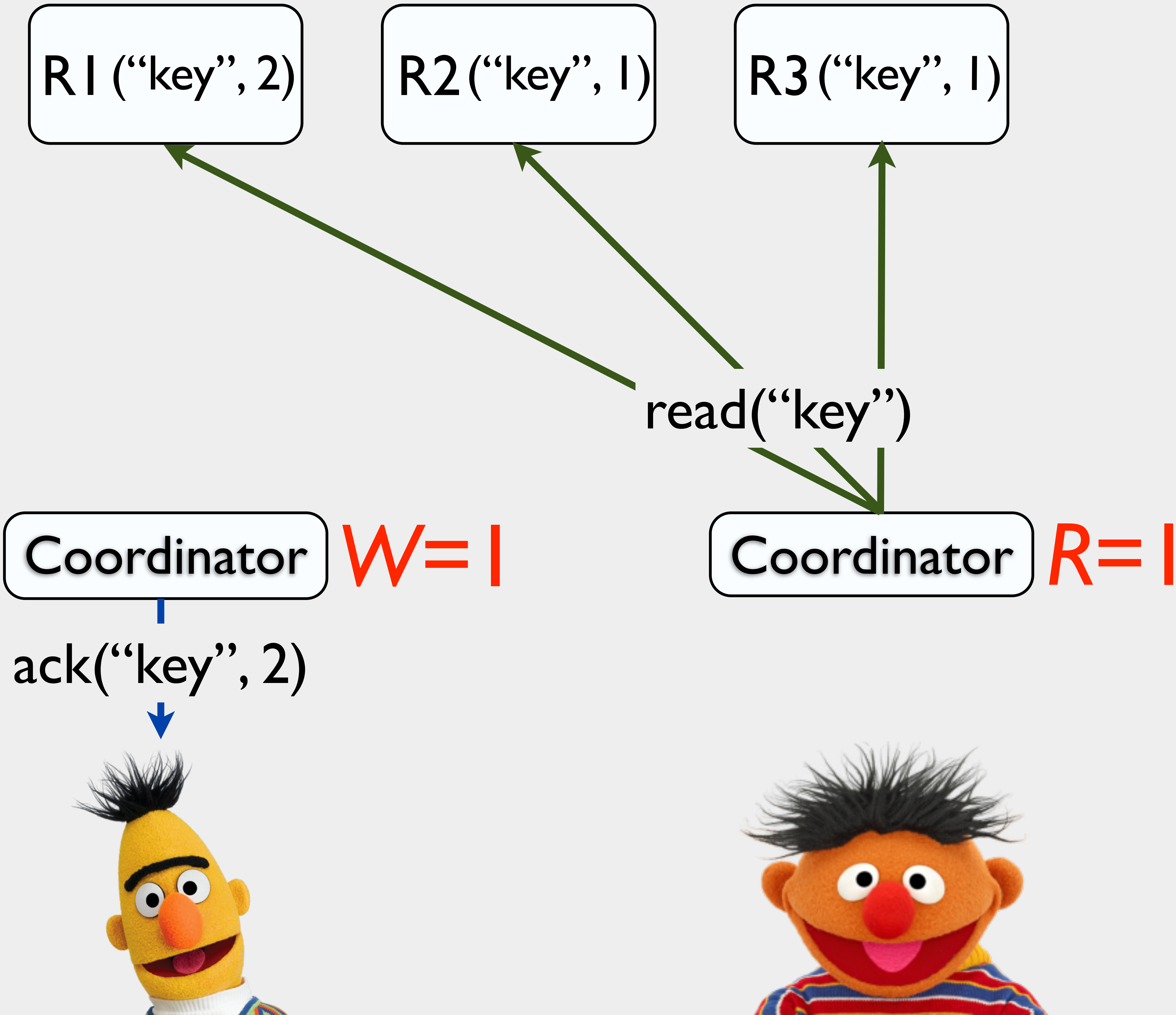

R1 ("key", 1)
R2 ("key", 1)
R3 ("key", 1)
write("key", 2)
Coordinator
W=1

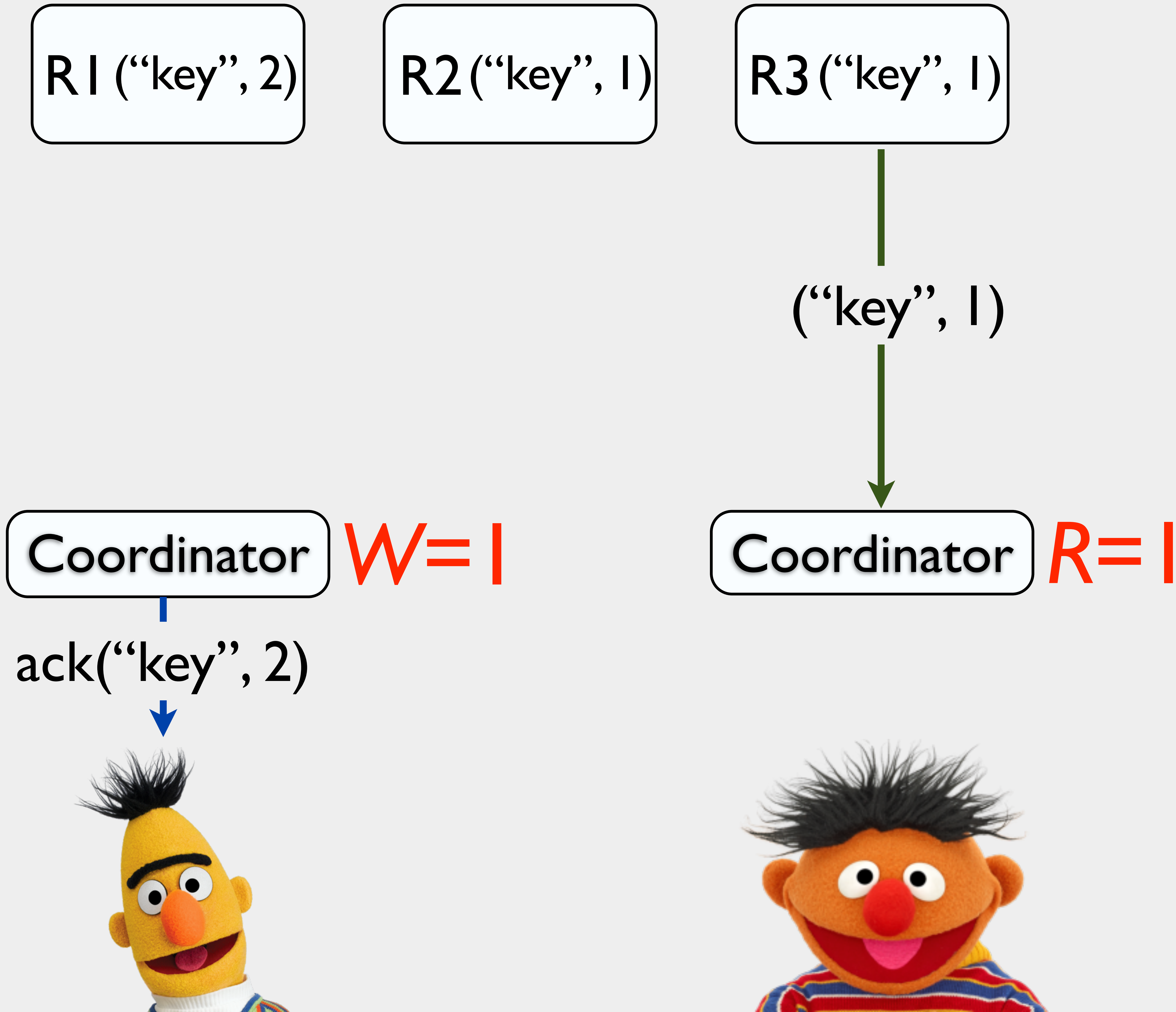

R1 (“key”, 2)
R2 (“key”, 1)
R3 (“key”, 1)
ack(“key”, 2)
Coordinator
W=1

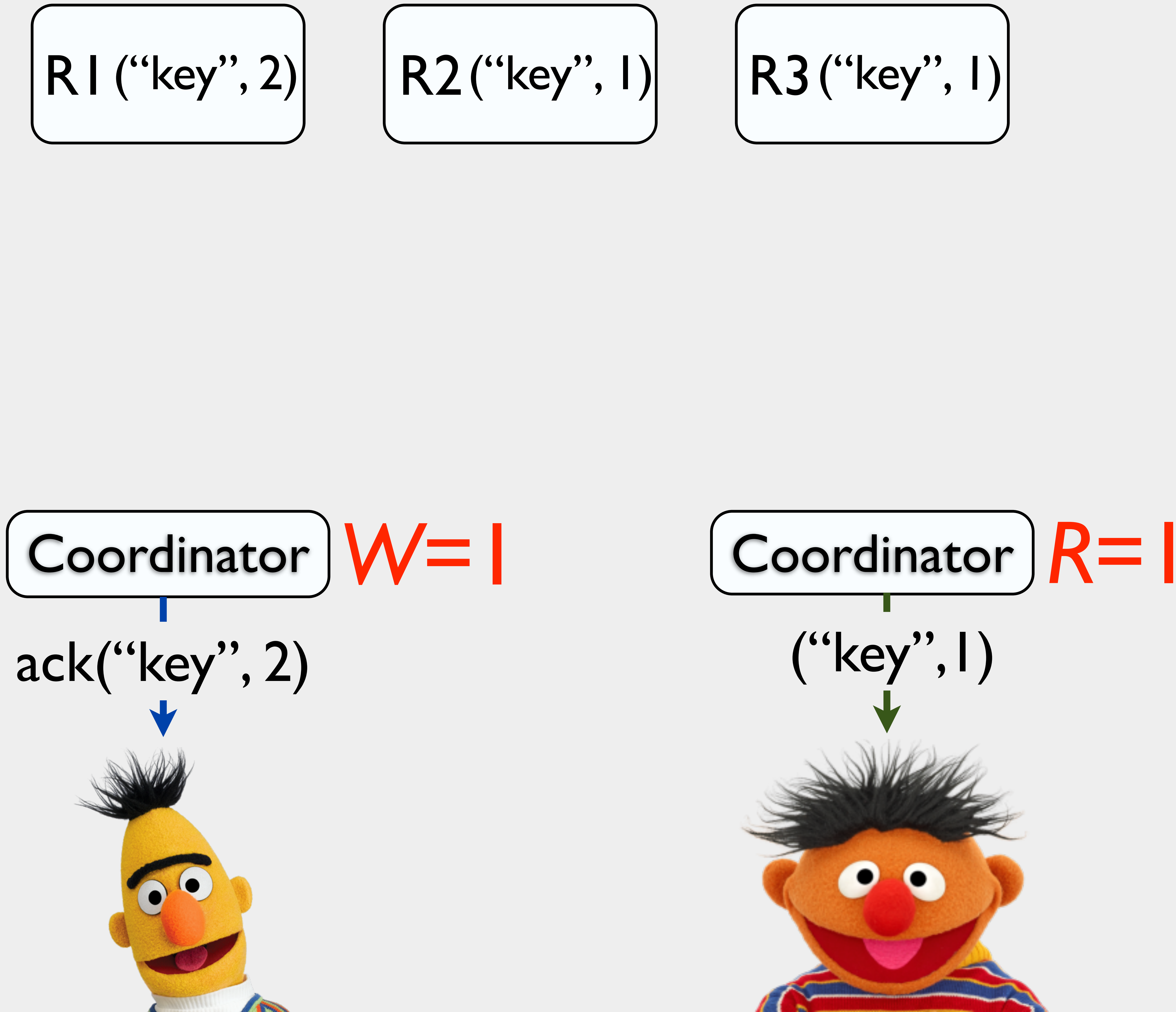

R1 (“key”, 2)
R2 (“key”, 1)
R3 (“key”, 1)
ack(“key”, 2)
Coordinator
W=1
ack(“key”, 2)



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
Coordinator
R=1
ack("key", 2)


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)

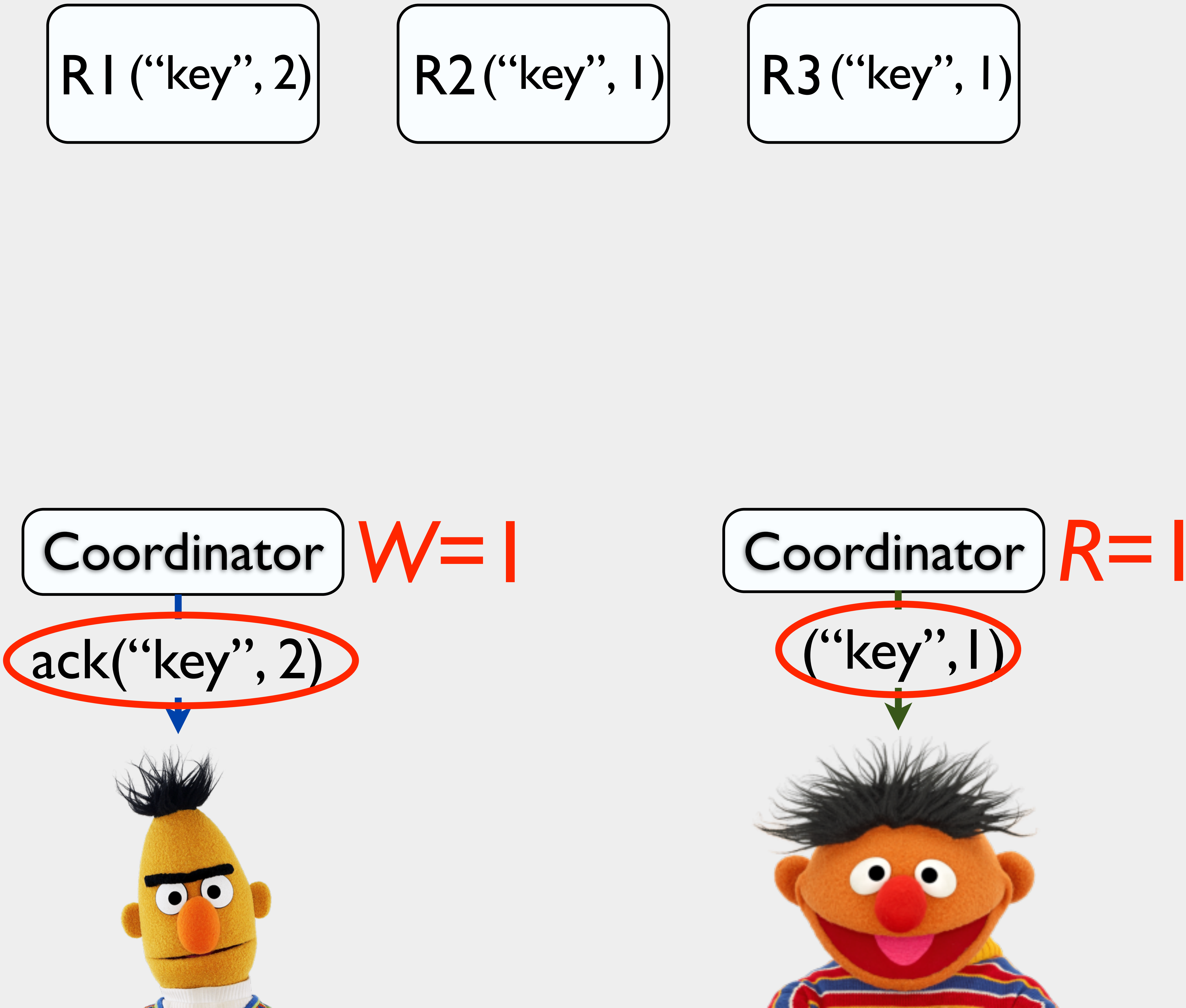
R3 ("key", 1)

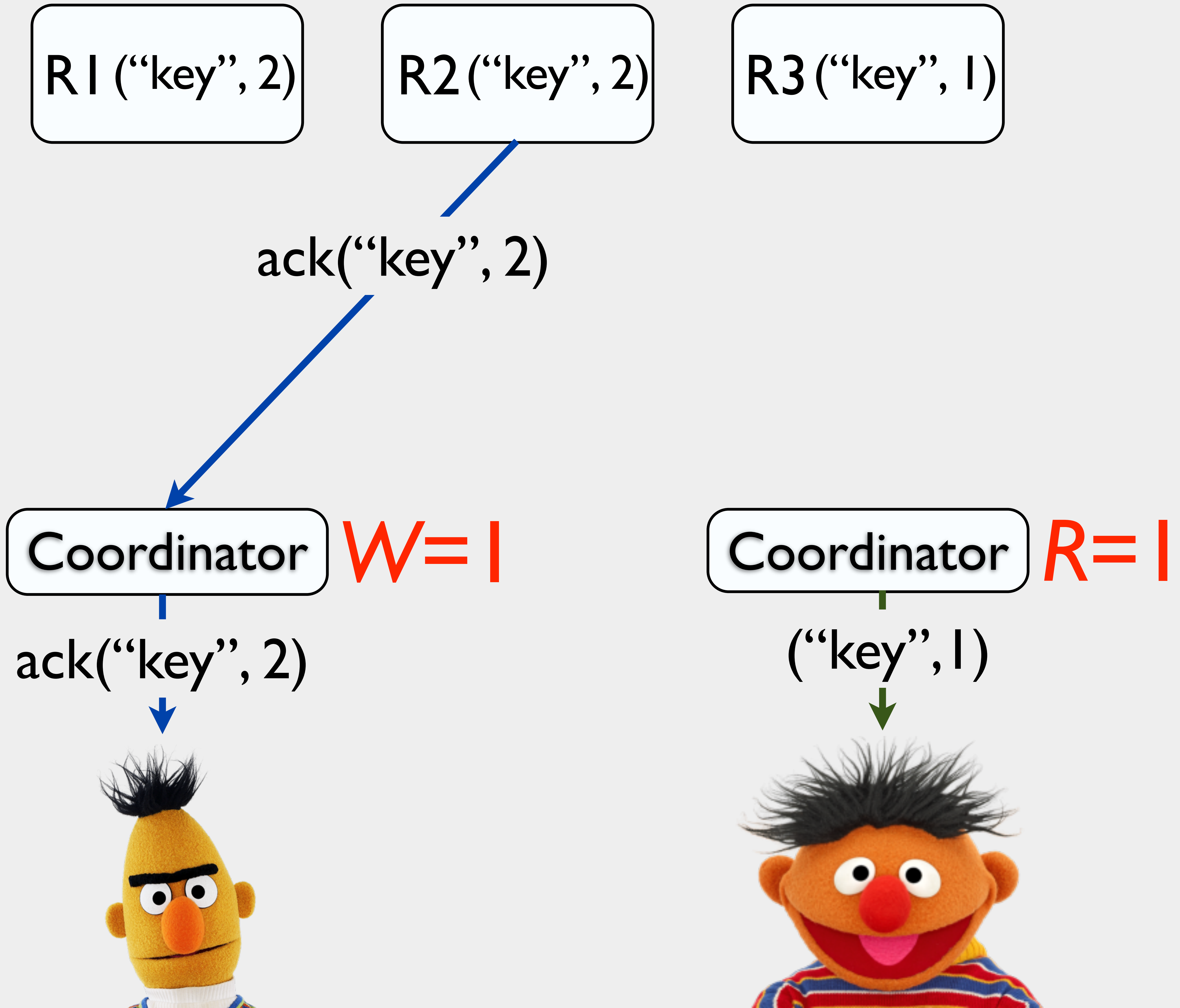
Coordinator W=1

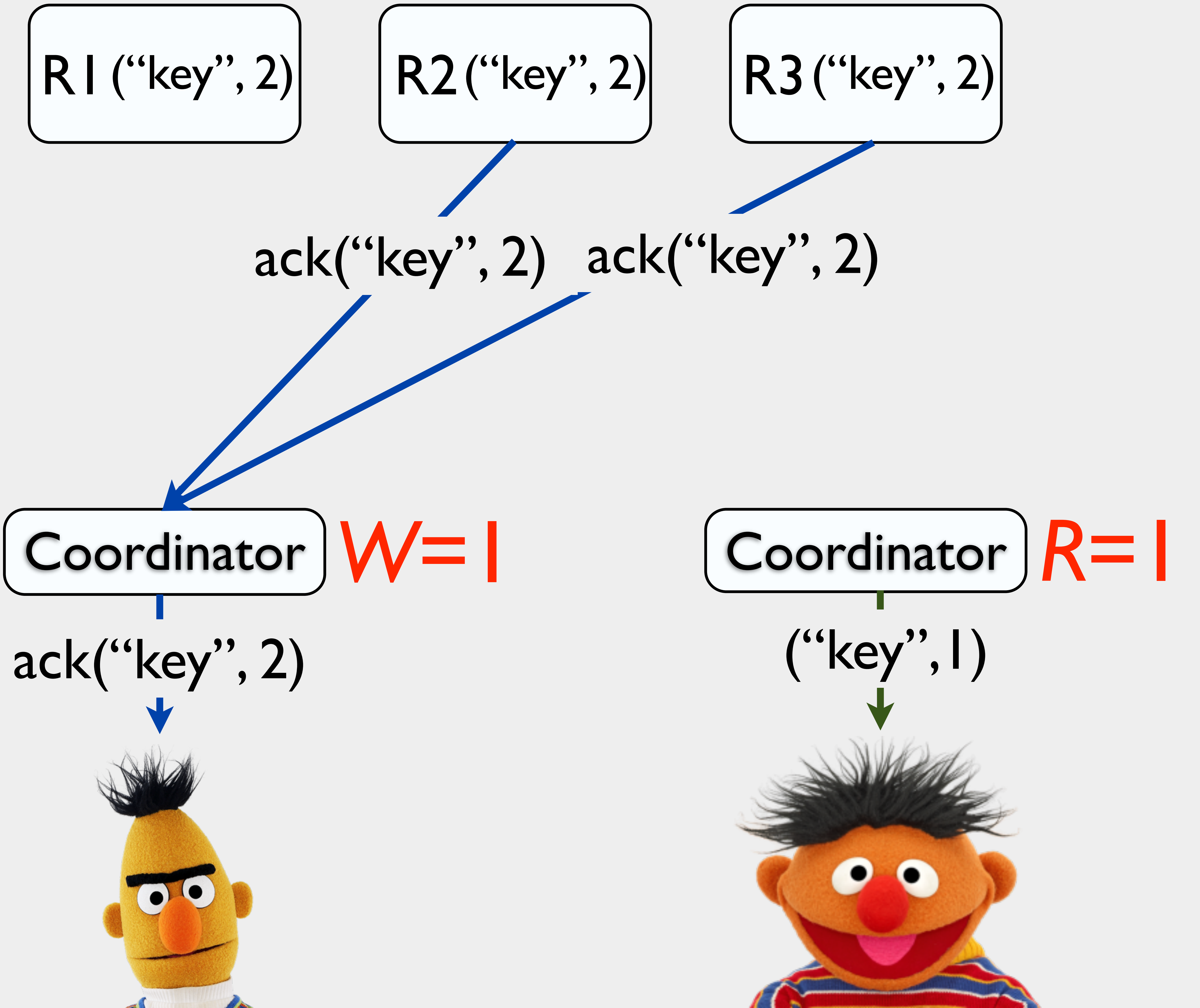
ack("key", 2)

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
Coordinator
R=1
ack(“key”, 2)
(“key”, 1)

R1 ("key", 2)

R2 ("key", 2)

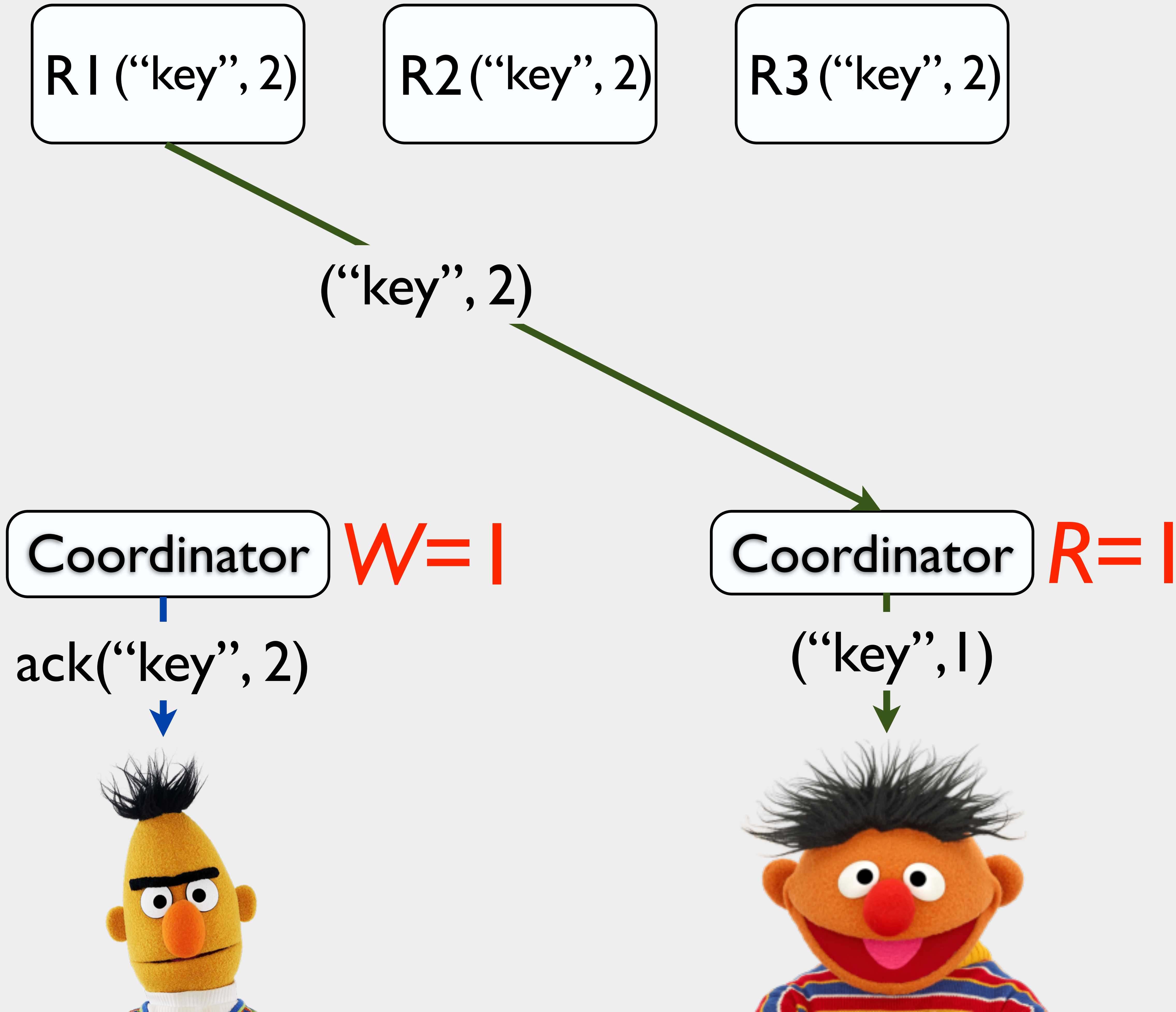
R3 ("key", 2)

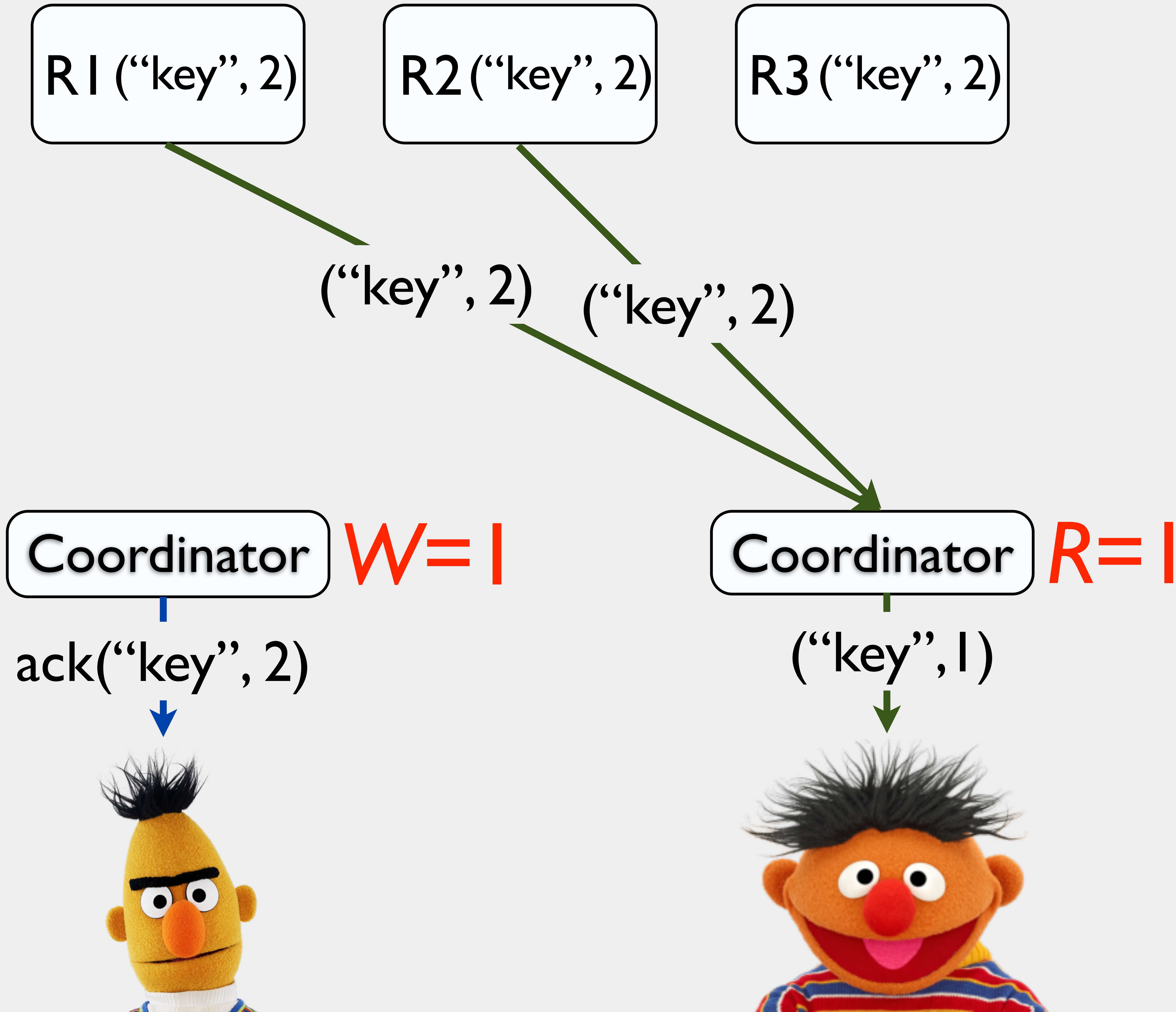
Coordinator W=1

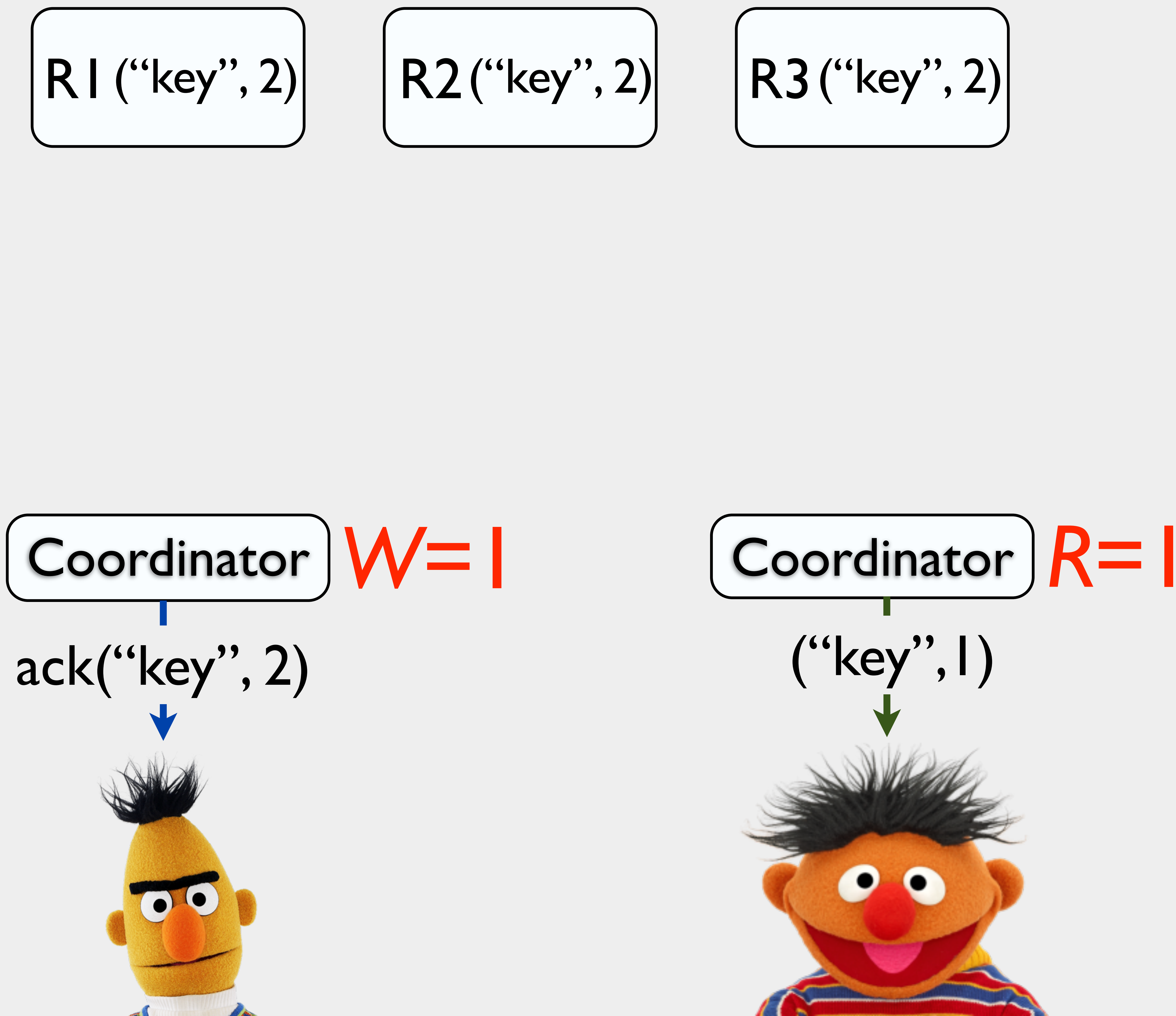
ack("key", 2)

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
Coordinator
R=1
ack("key", 2)
("key", 1)


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

if: $W > \lceil N/2 \rceil$

$R+W > N$

then: read (at least) last committed version

else: eventual consistency

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?

Riak Defaults

N=3

R=2

W=2

Riak Defaults

N=3

R=2

W=2

2+2 > 3

Riak Defaults

N=3
R=2
W=2

2+2 > 3

Phew, I'm safe!

Riak Defaults

N=3
R=2
W=2

2+2 > 3

Phew, I'm safe!

...but what's my latency cost?

Riak Defaults

N=3
R=2
W=2

2+2 > 3

Phew, I'm safe!

...but what's my latency cost?

Should I change?

strong consistency

strong consistency

low latency

Cassandra:

R=W=1, N=3

by default

$(1+1 \ngtr 3)$

"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

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

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":

$$R=W=1, N=2$$

@strlen, 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*

can 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?

a SLA for consistency

How eventual?

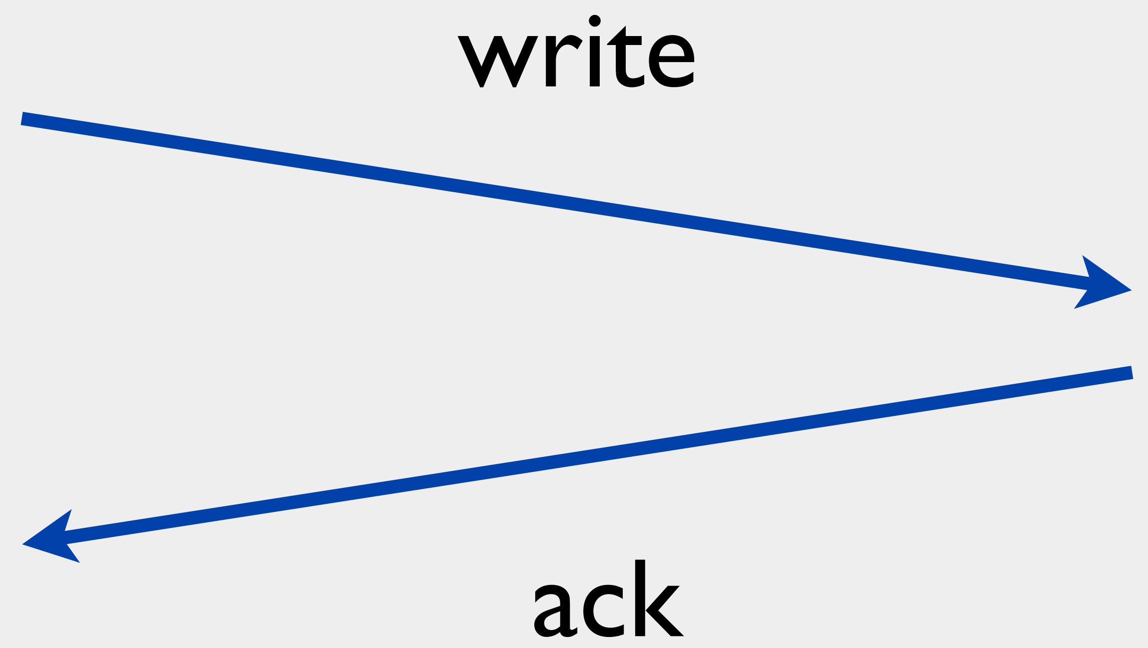
t-visibility: consistent reads with probability *p* after after *t* seconds

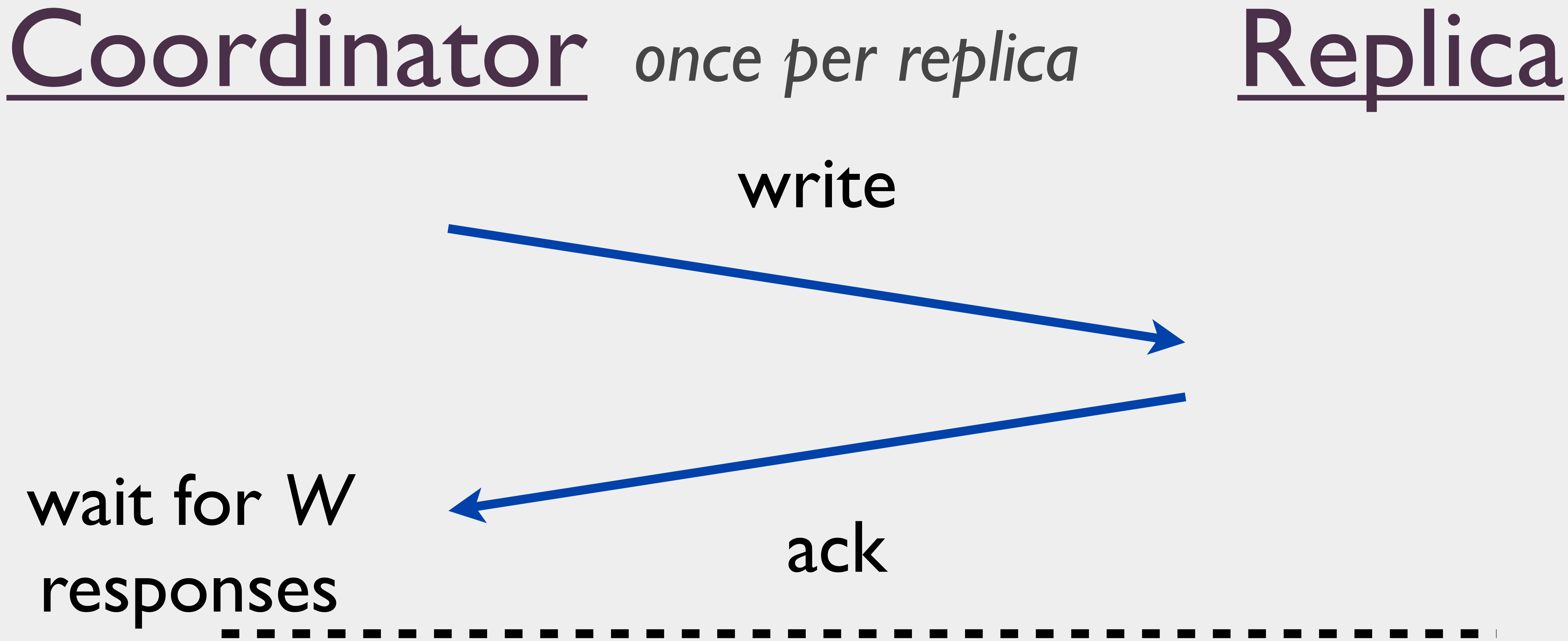
(e.g., 99.9% of reads will be consistent after 10ms)

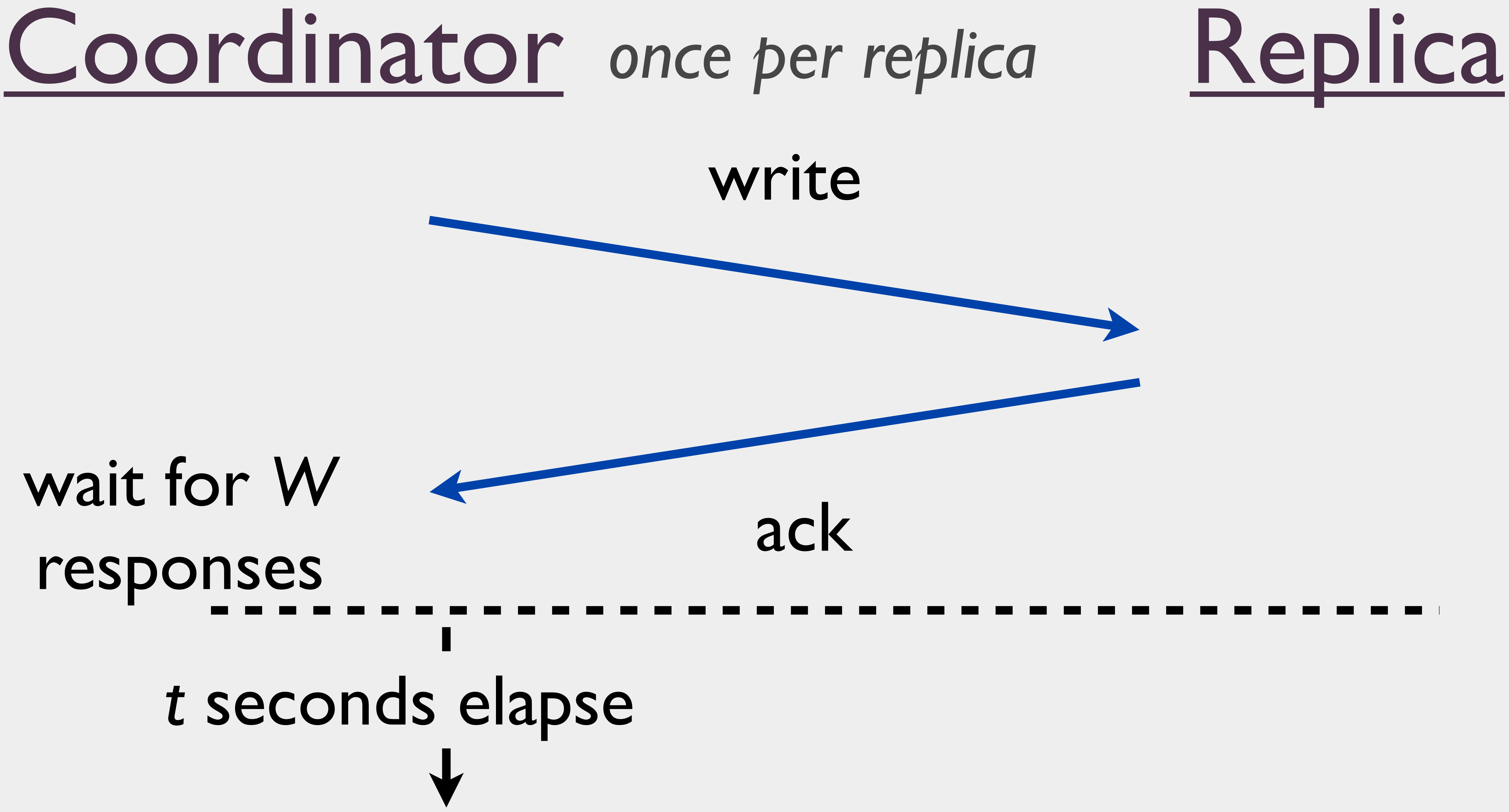
Coordinator *once per replica* Replica

<u>Coordinator</u> *once per replica* <u>Replica</u>

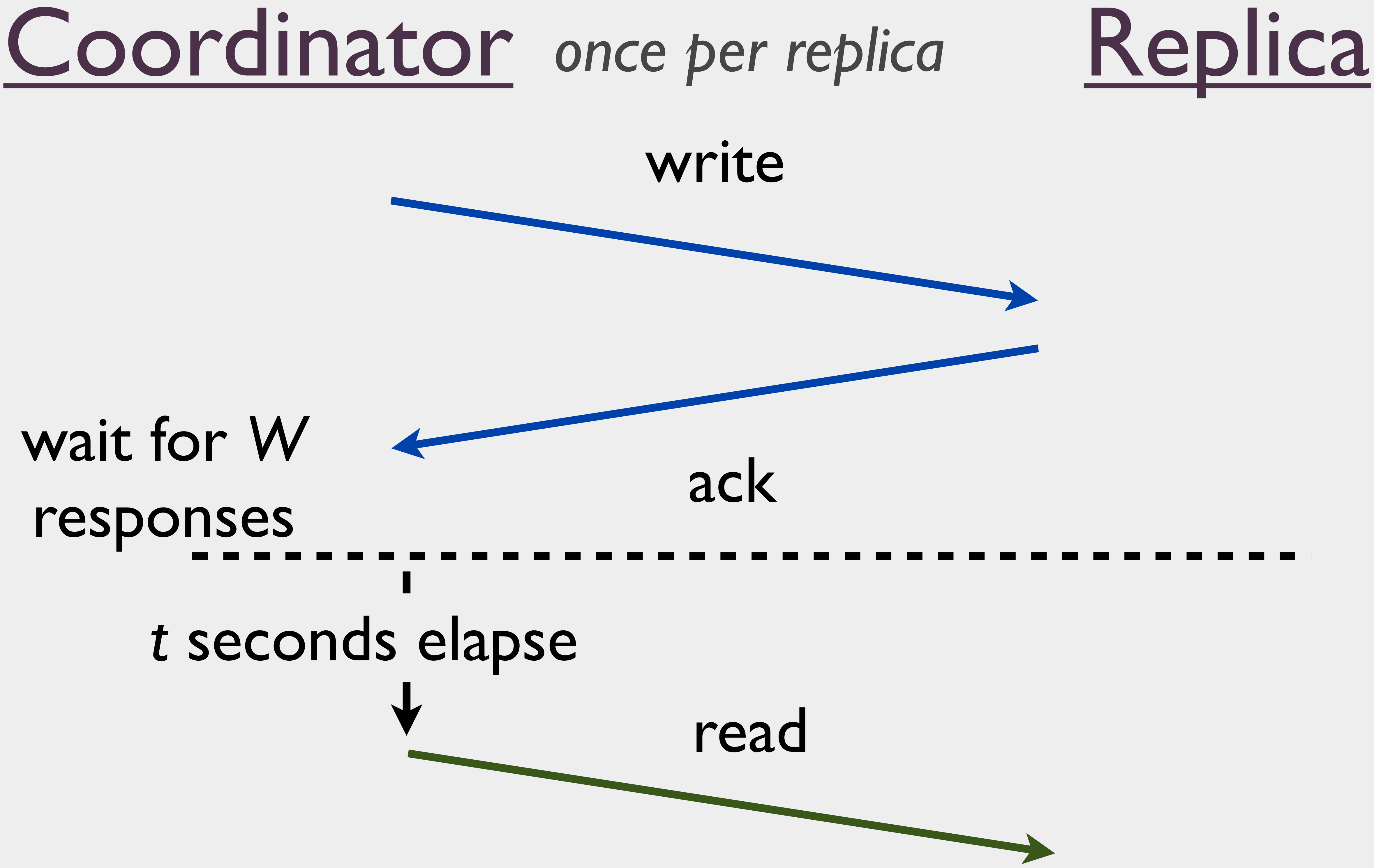
Coordinator *once per replica* Replica

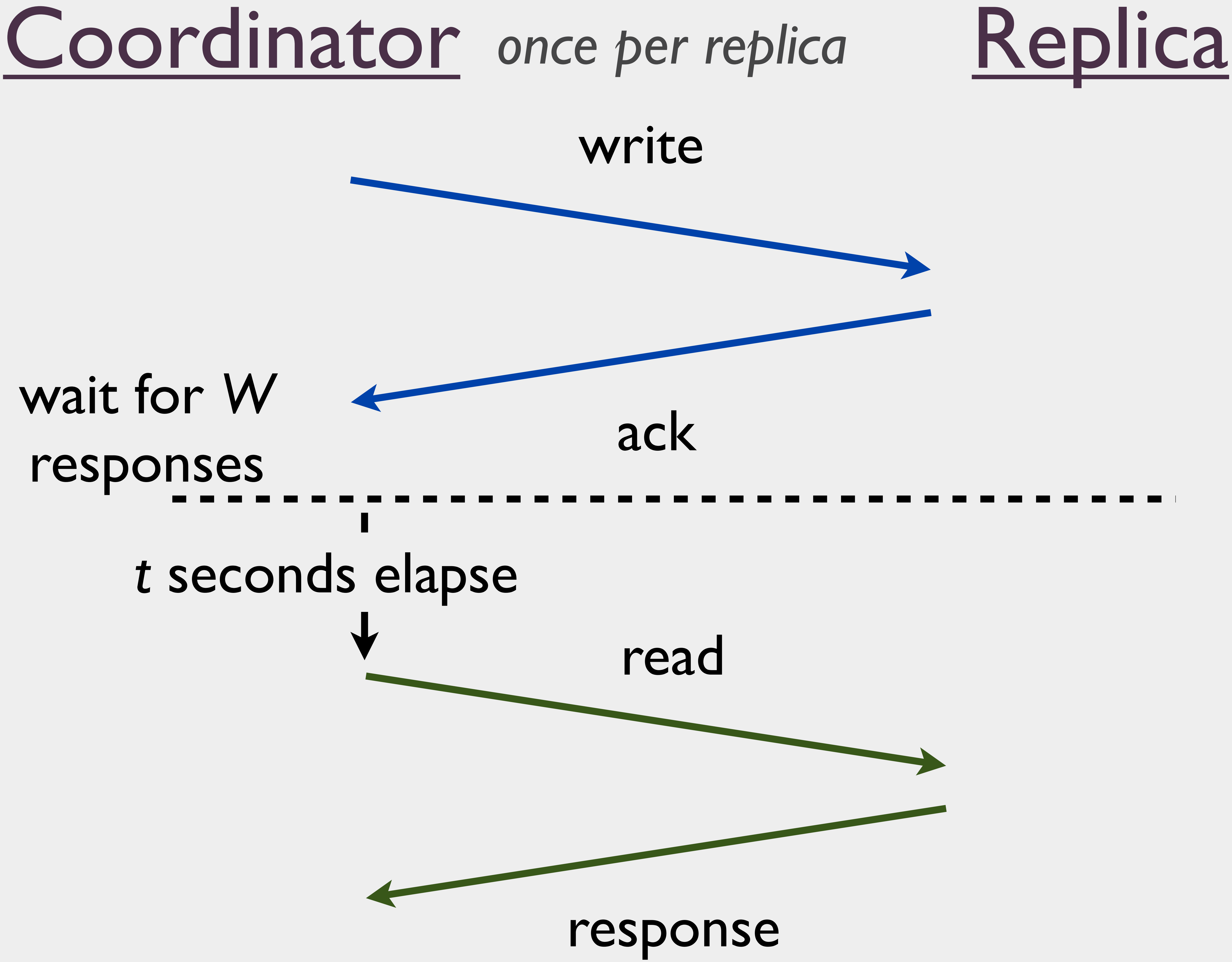


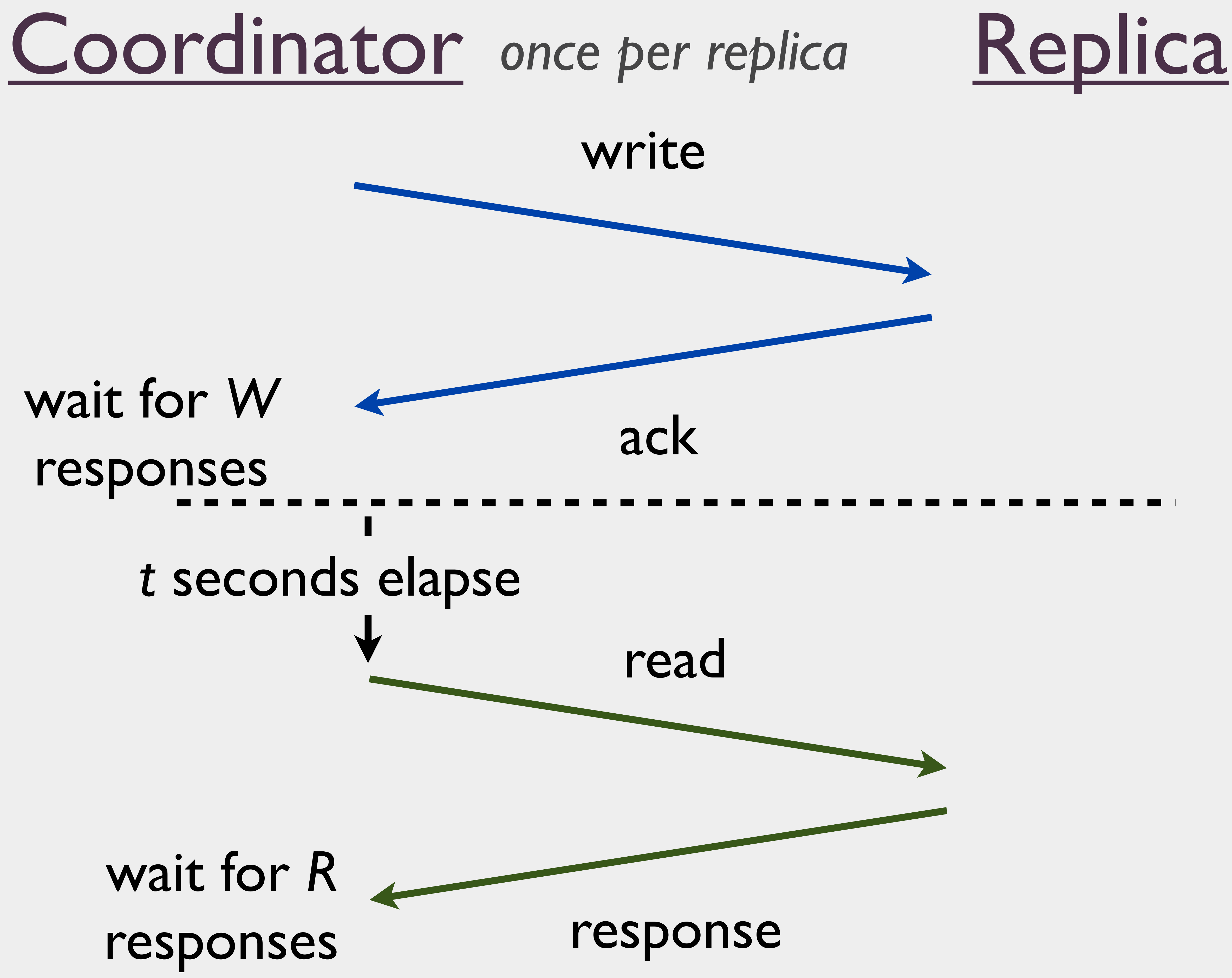

Coordinator
once per replica
Replica
write
wait for *W* responses
ack

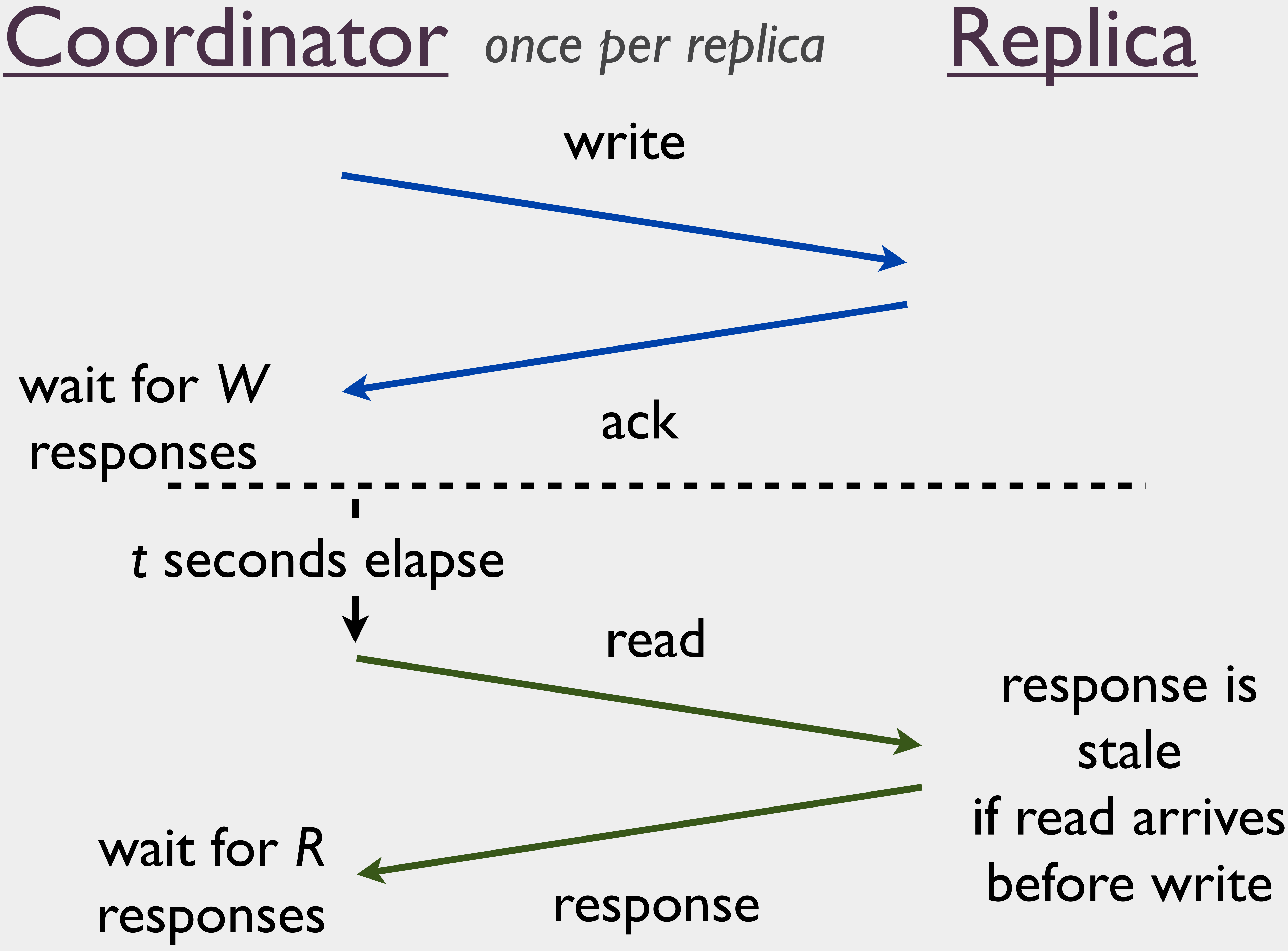


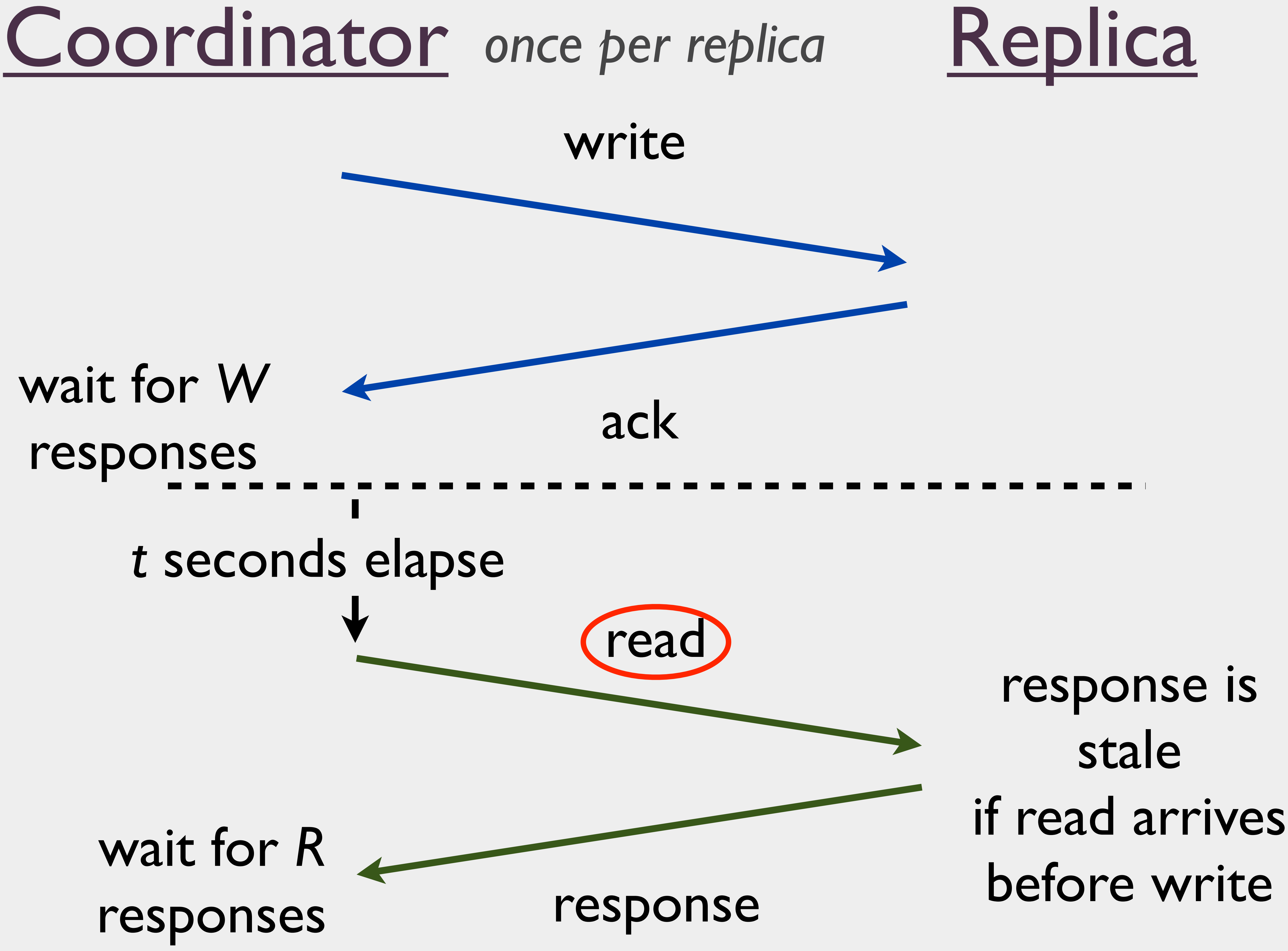
Coordinator
once per replica
Replica
write
wait for W responses
ack
t seconds elapse

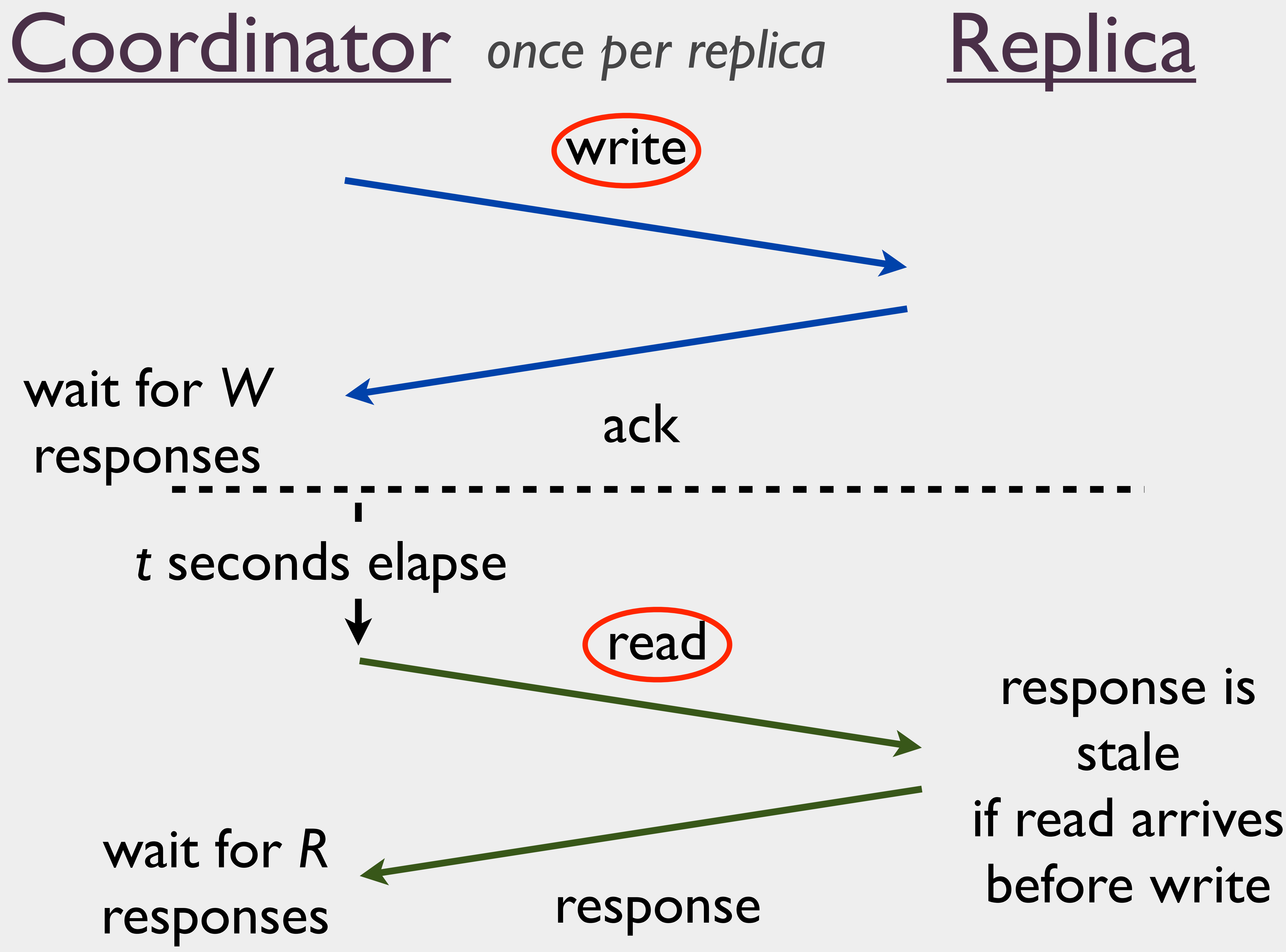

Coordinator
once per replica
Replica
write
wait for W responses
ack
t seconds elapse
read

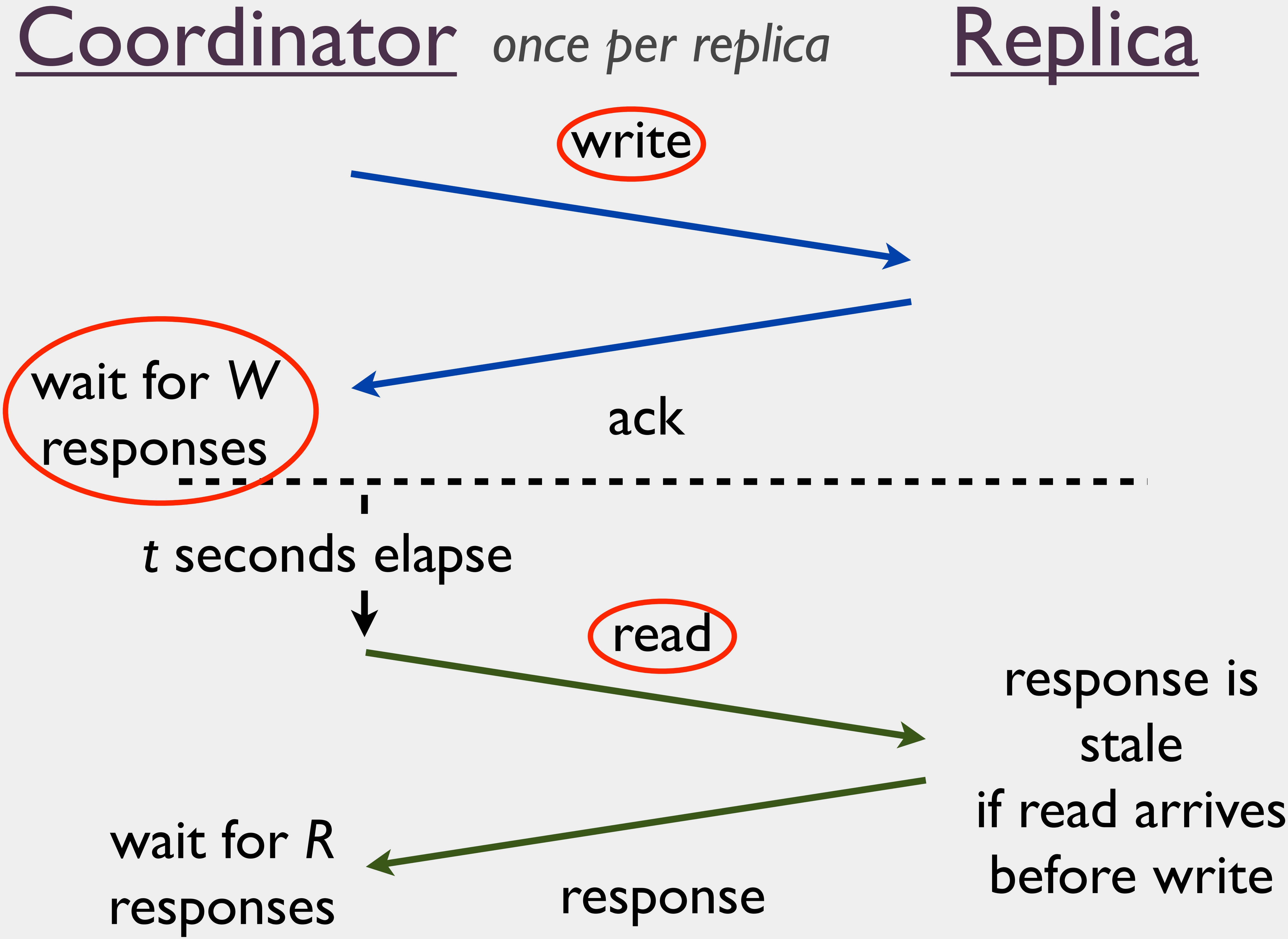

Coordinator
once per replica
Replica
write
wait for W responses
ack
t seconds elapse
read
response


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


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

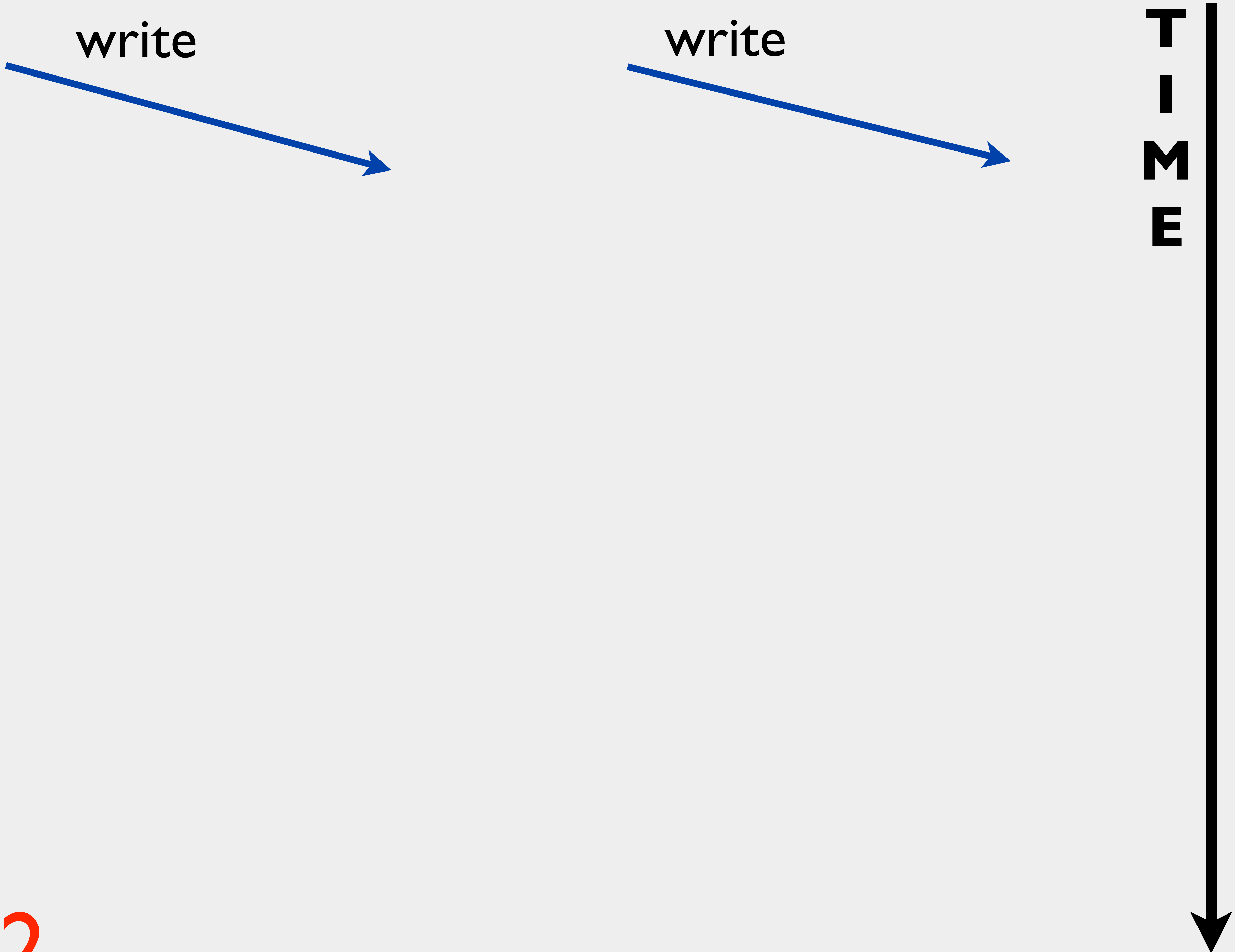

Coordinator
once per replica
Replica
write
wait for W
responses
ack
t seconds elapse
read
response is
stale
if read arrives
before write
wait for R
responses
response


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

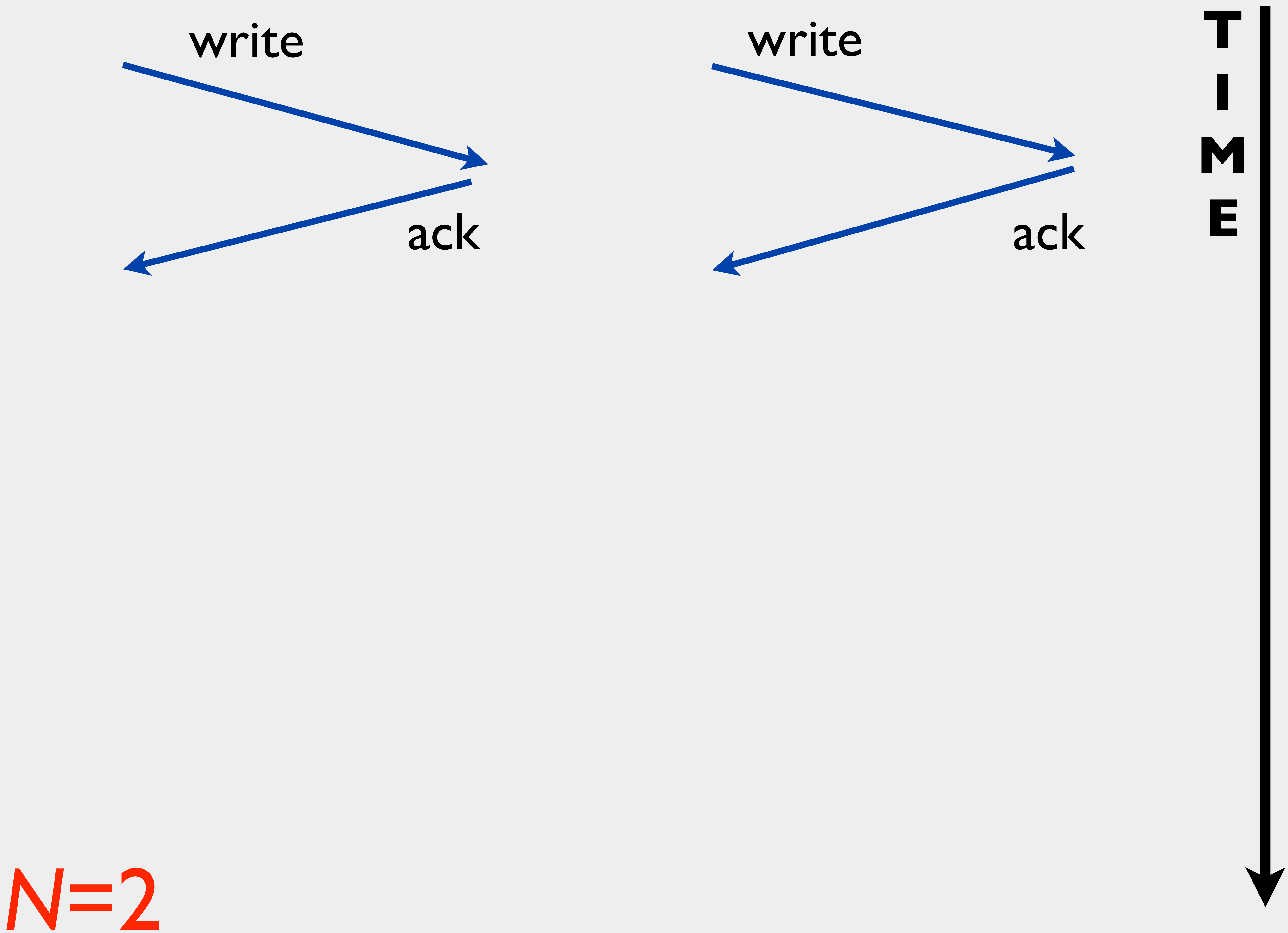

Coordinator
once per replica
Replica
write
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response

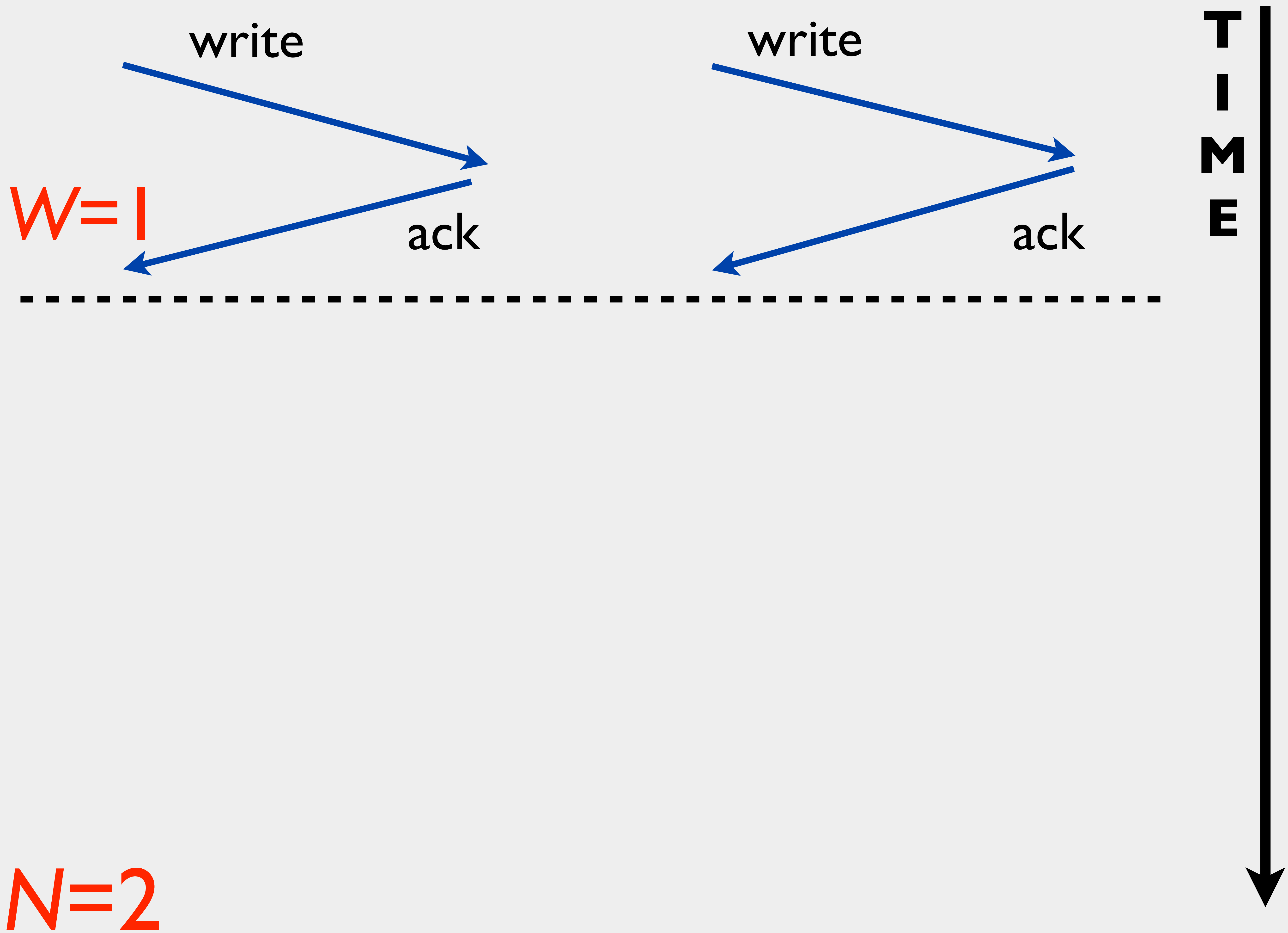
TIME

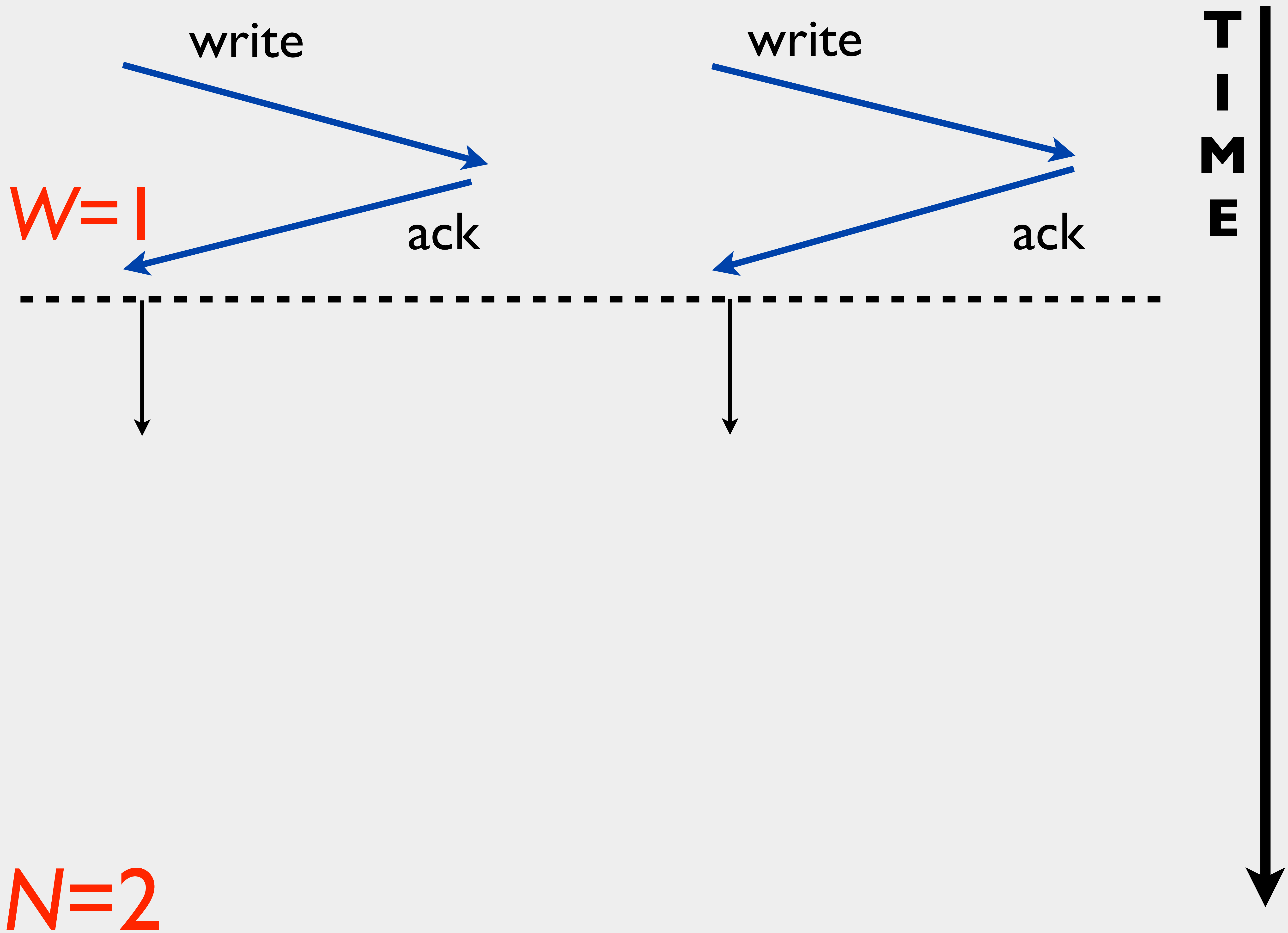
$N=2$

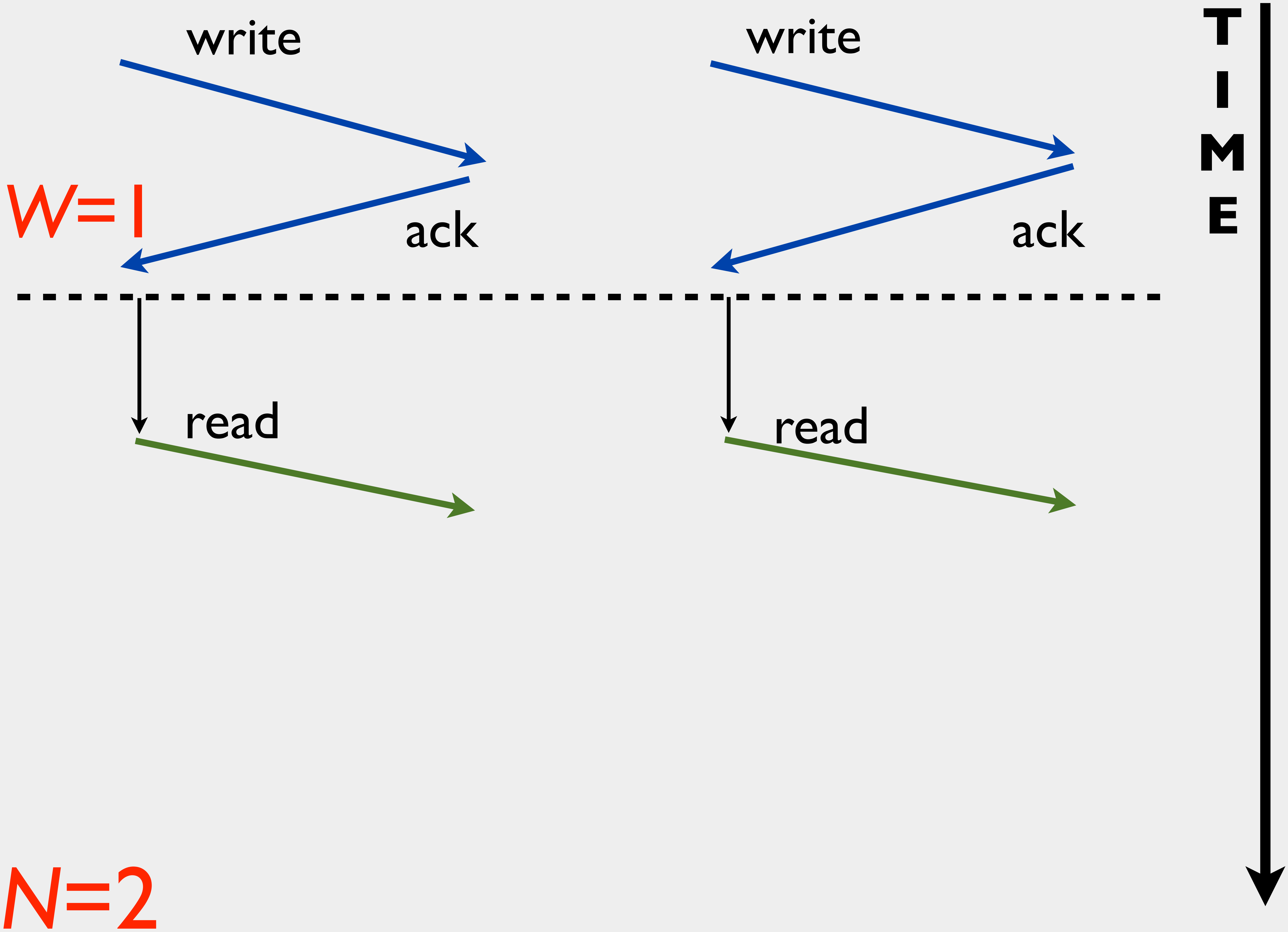


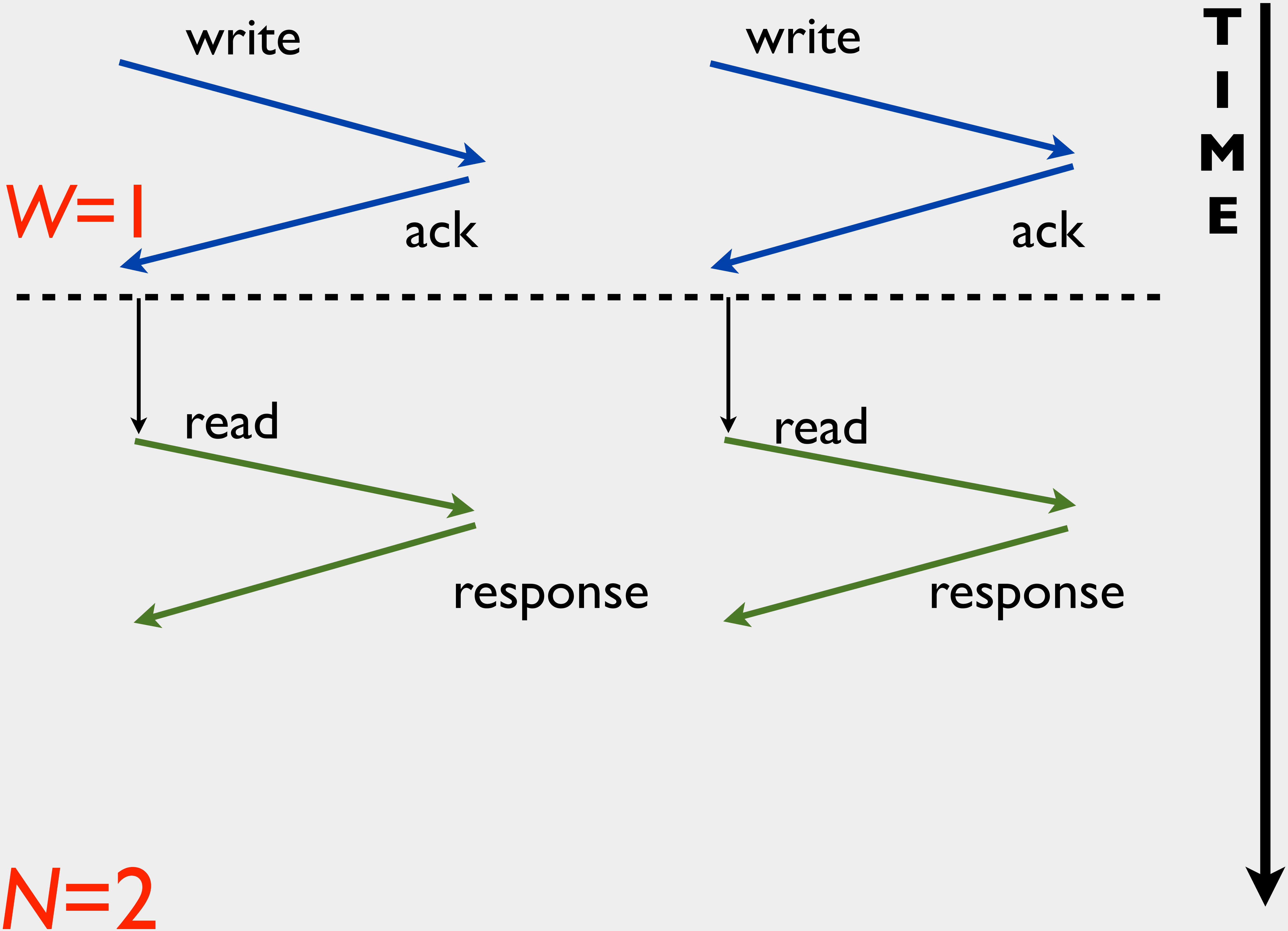
N=2

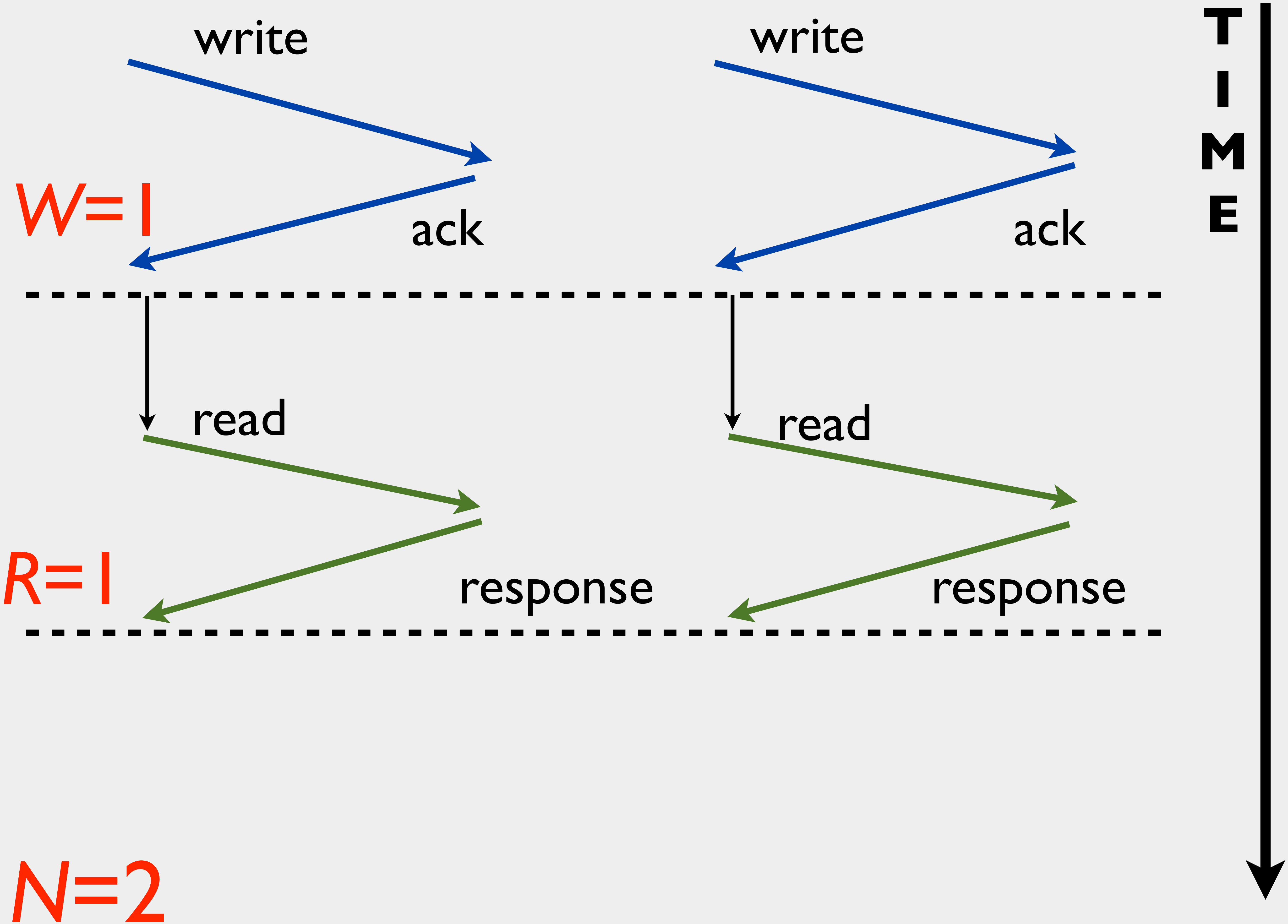

write
ack
write
ack
TIME
N=2

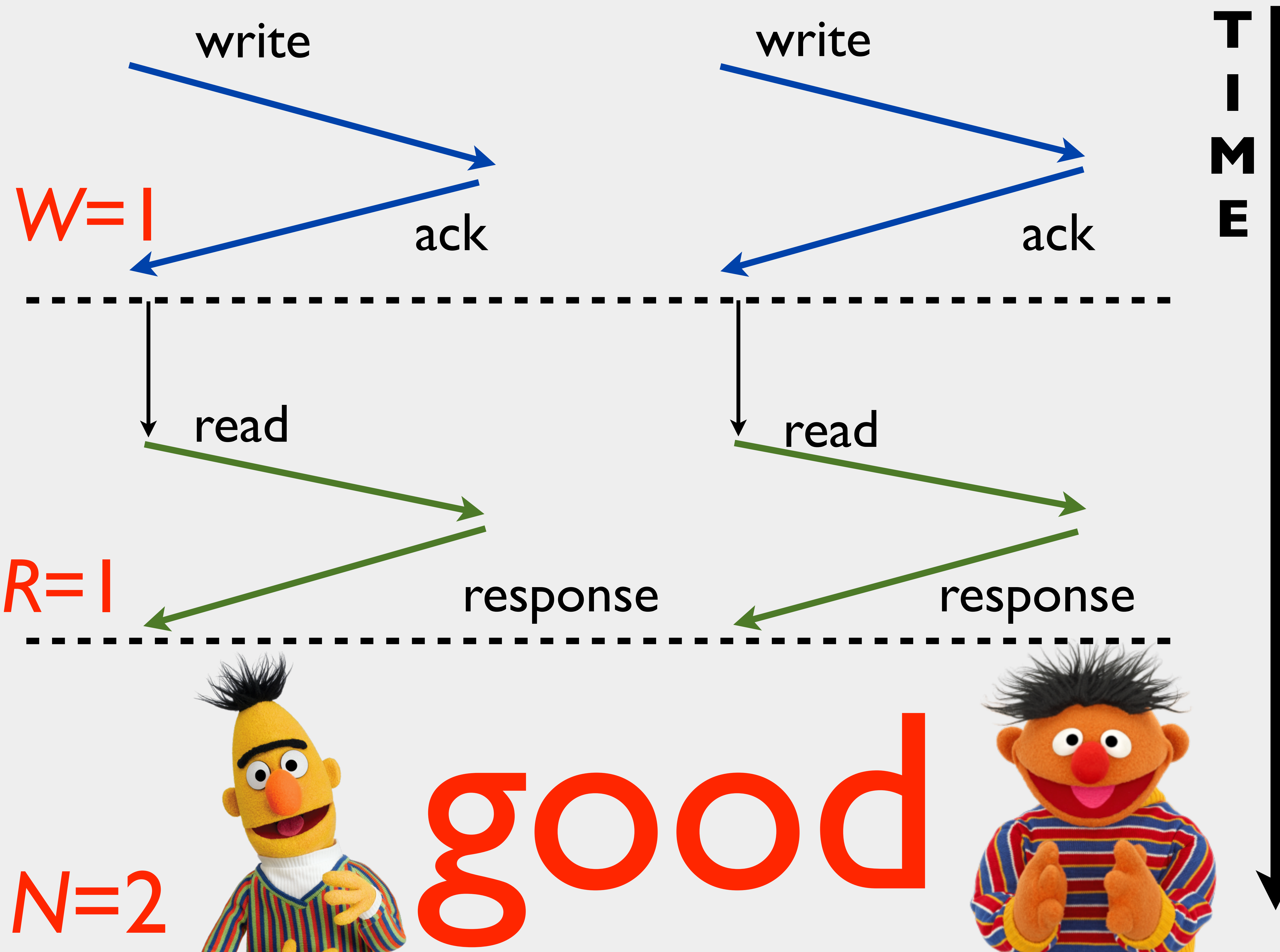

write
write
W=1
ack
ack
TIME
N=2


write
write
TIME
W=1
ack
ack
N=2


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

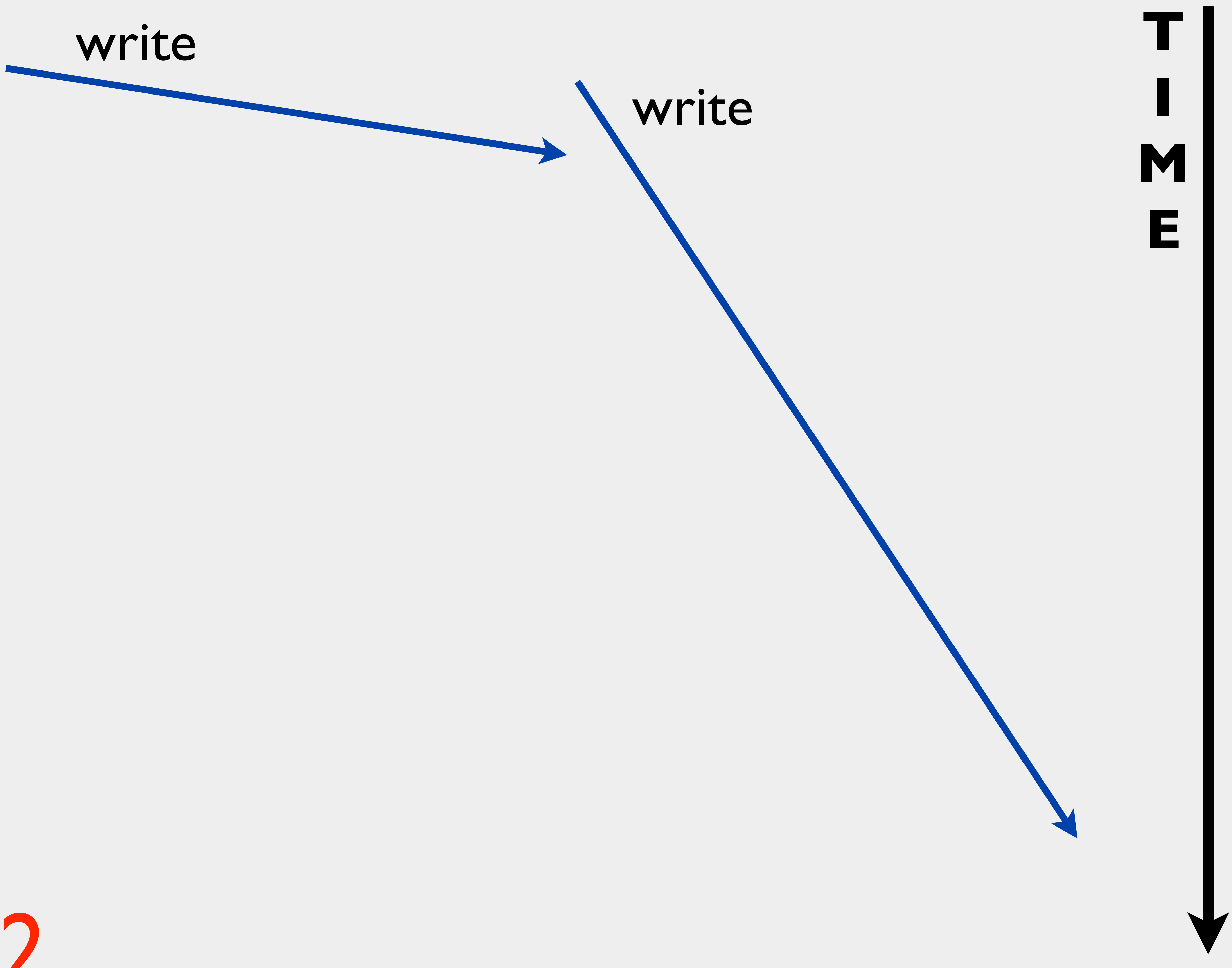

write
write
W=1
ack
ack
TIME
read
read
response
response
N=2


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

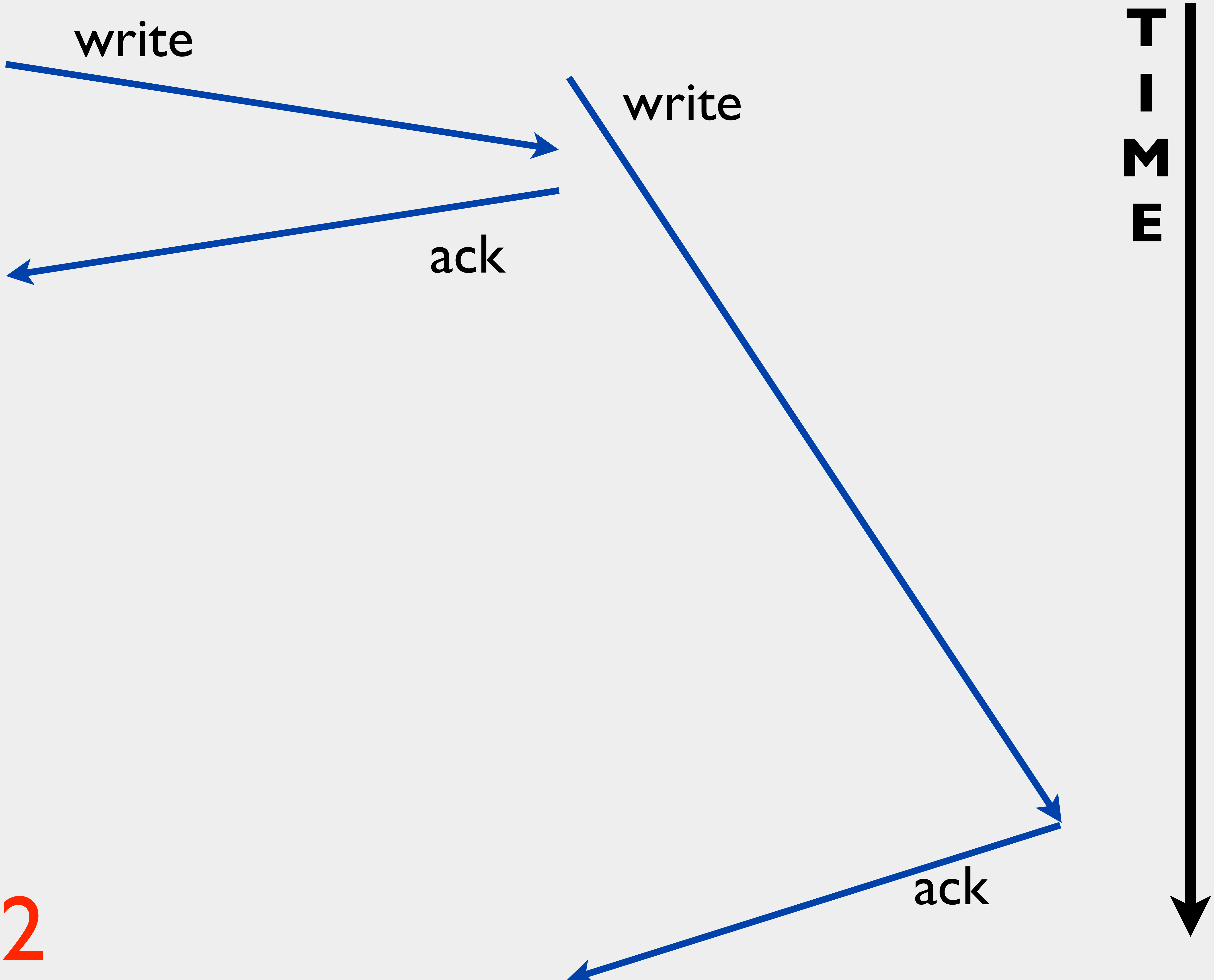

write
write
ack
ack
W=1
read
read
response
response
R=1
N=2
good
TIME

TIME

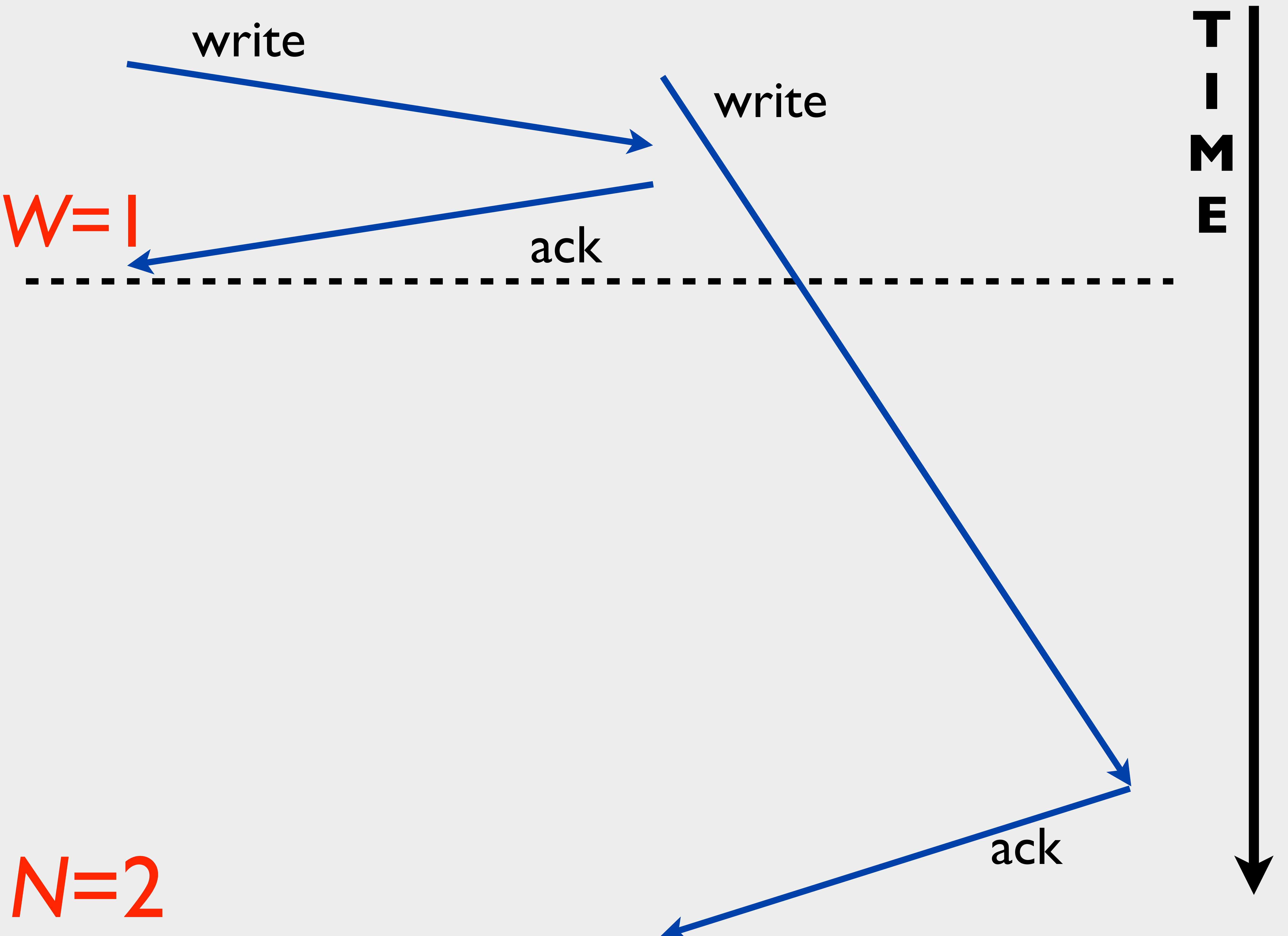
$N=2$

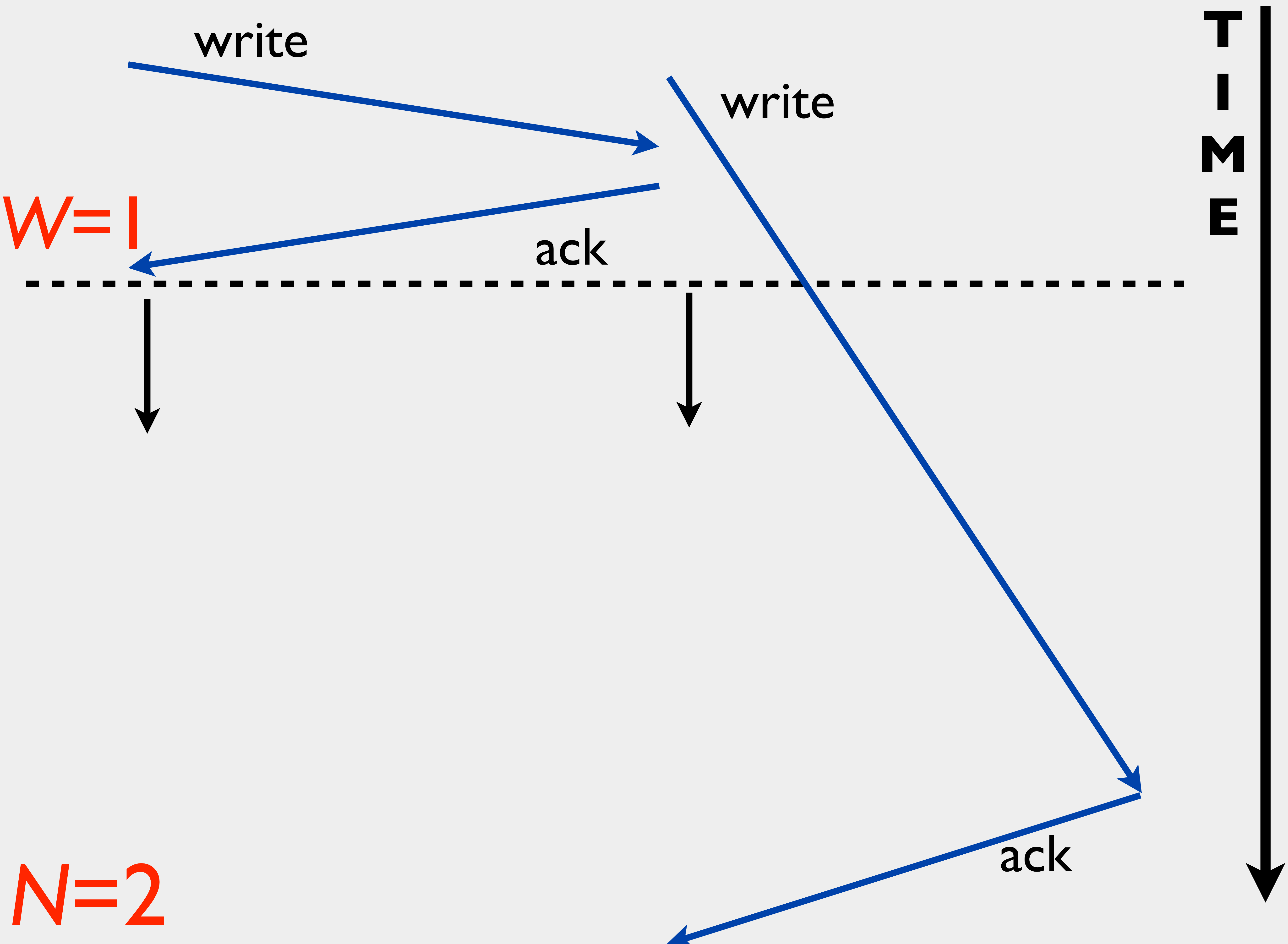


N=2

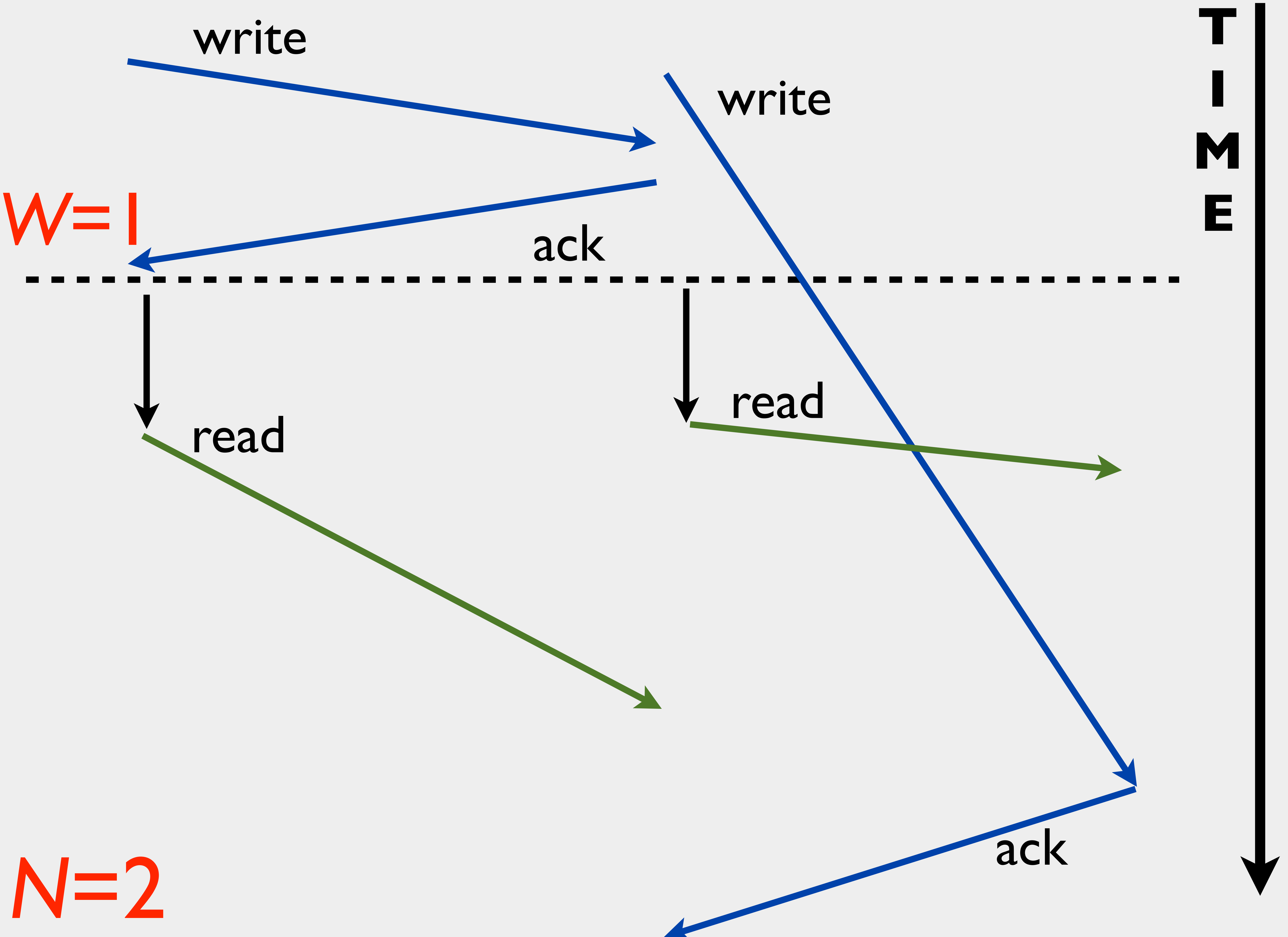


N=2

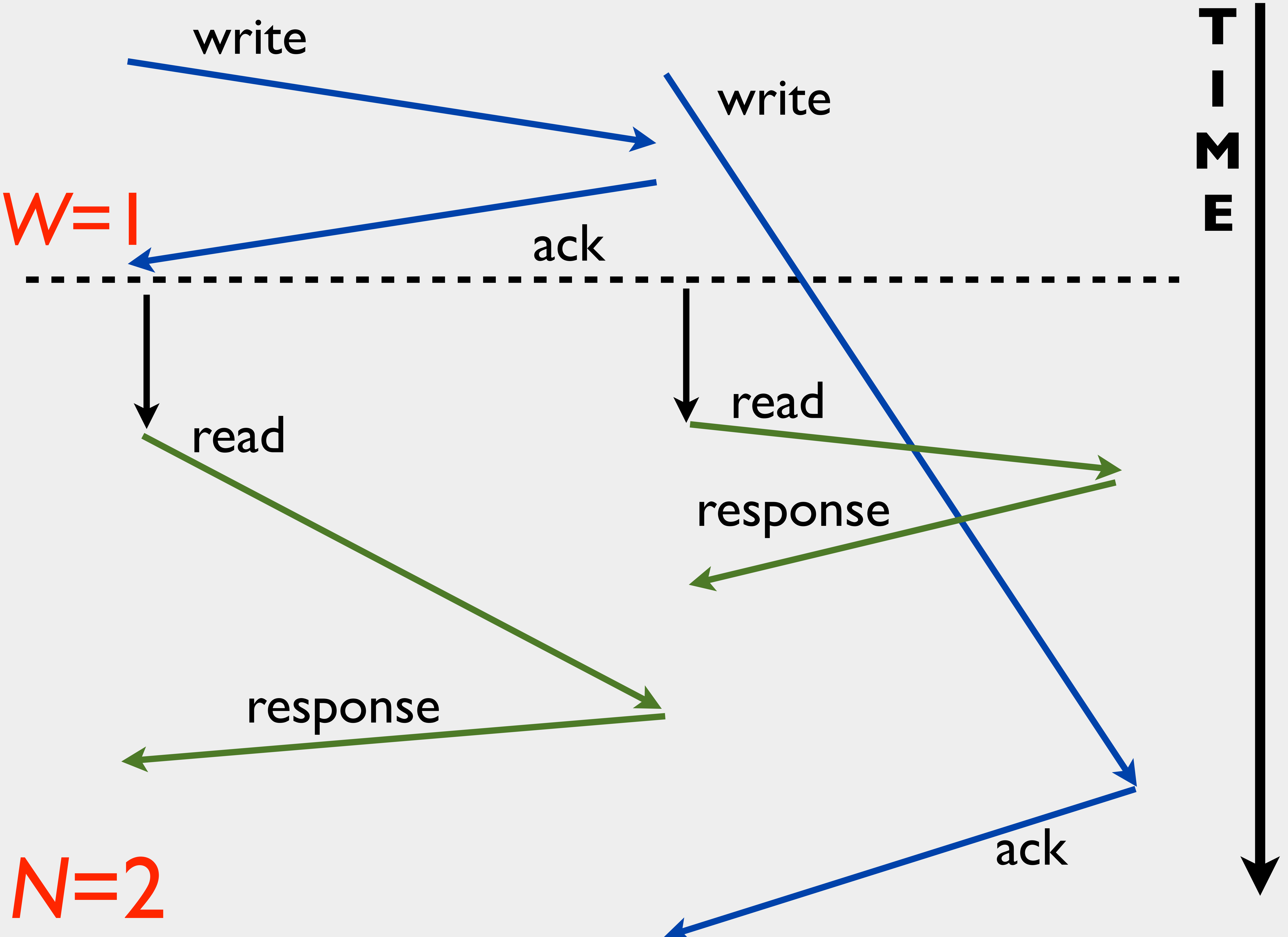

write
write
W=1
ack
TIME
ack
N=2

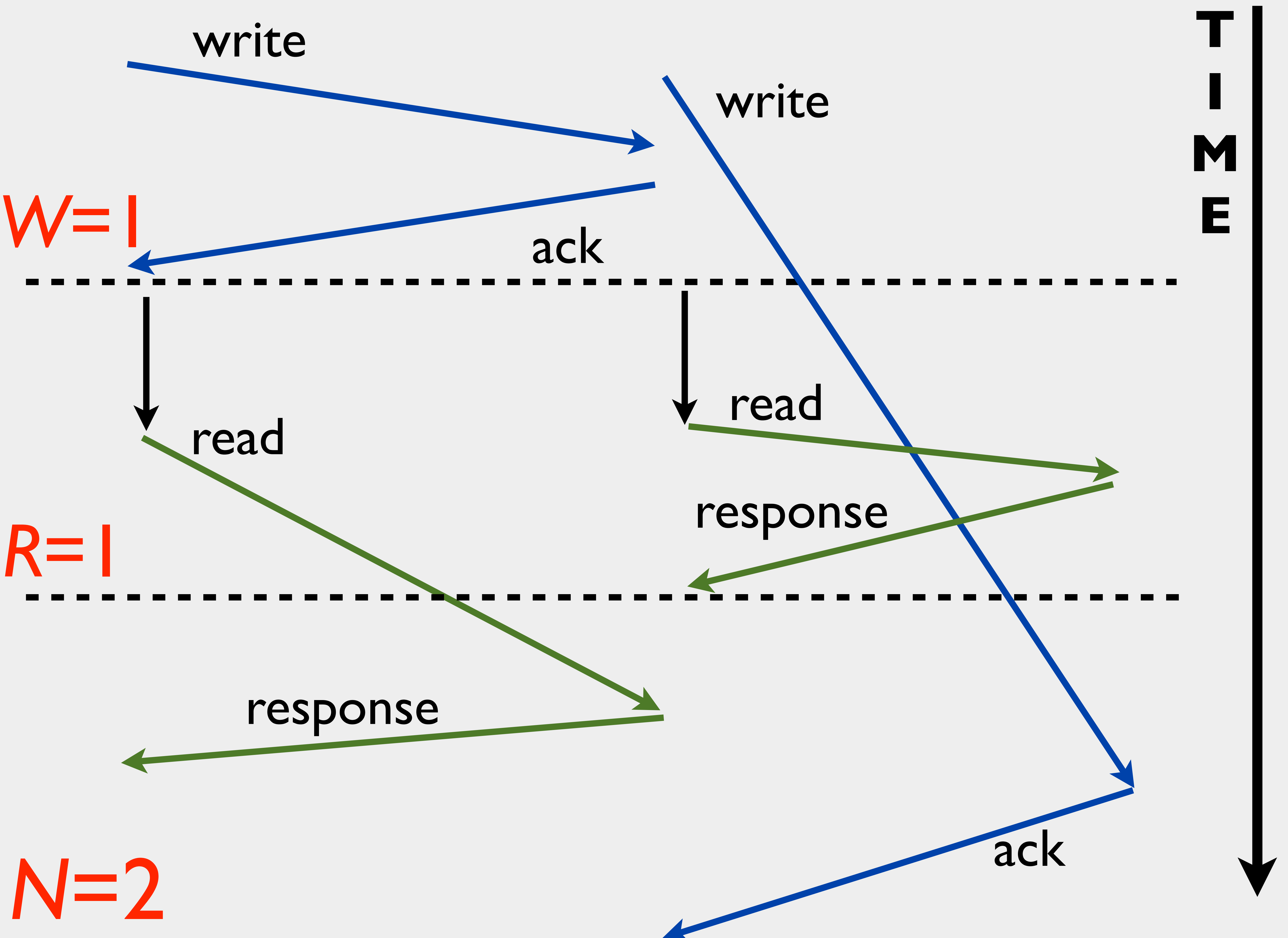


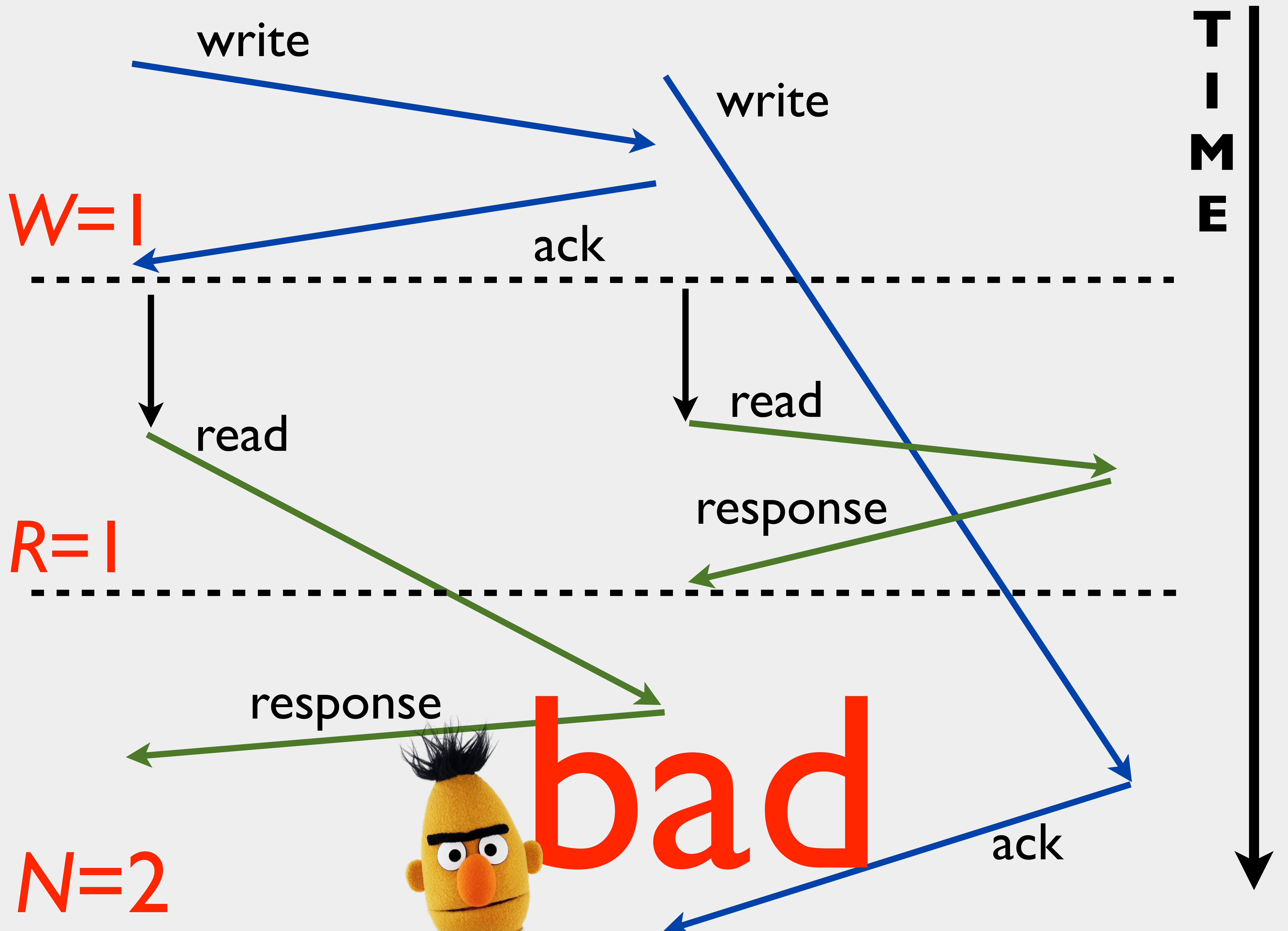
write
write
W=1
ack
TIME
ack
N=2

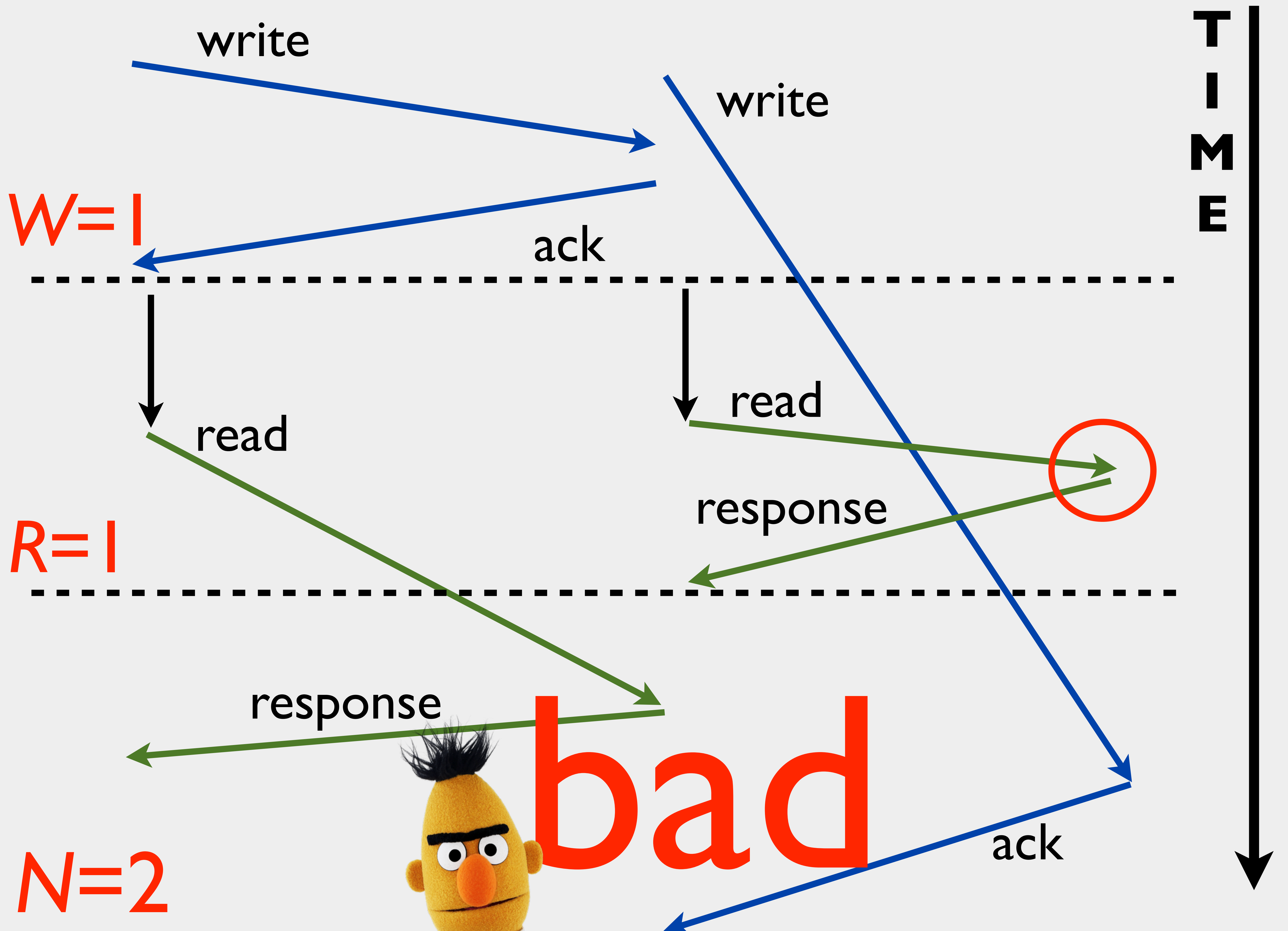


write
write
W=1
ack
read
read
ack
N=2
TIME

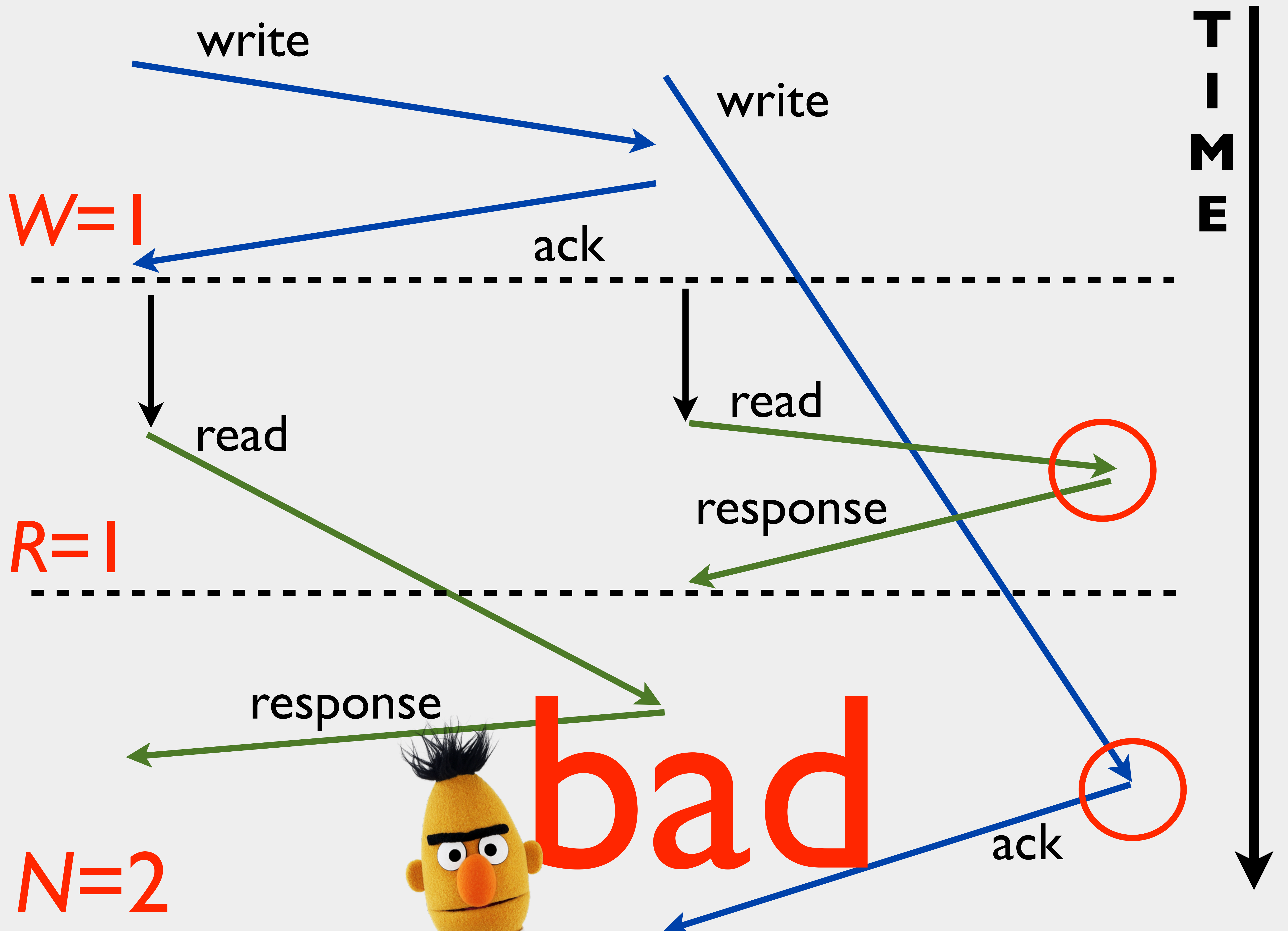

write
write
W=1
ack
TIME
read
read
response
response
ack
N=2


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

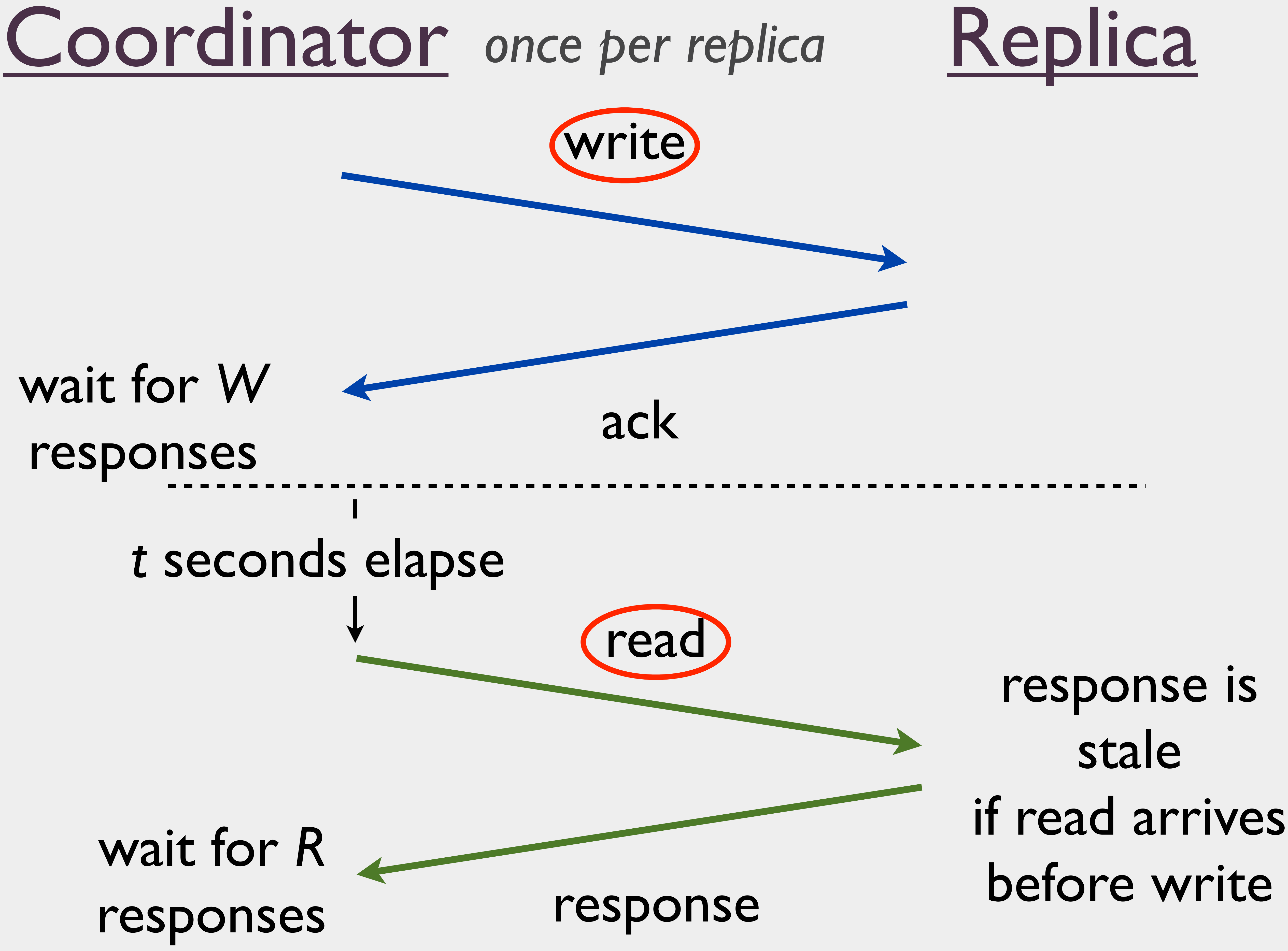

write
write
W=1
ack
T
I
M
E
read
read
response
R=1
response
bad
N=2
ack



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



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


Coordinator
once per replica
Replica
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)

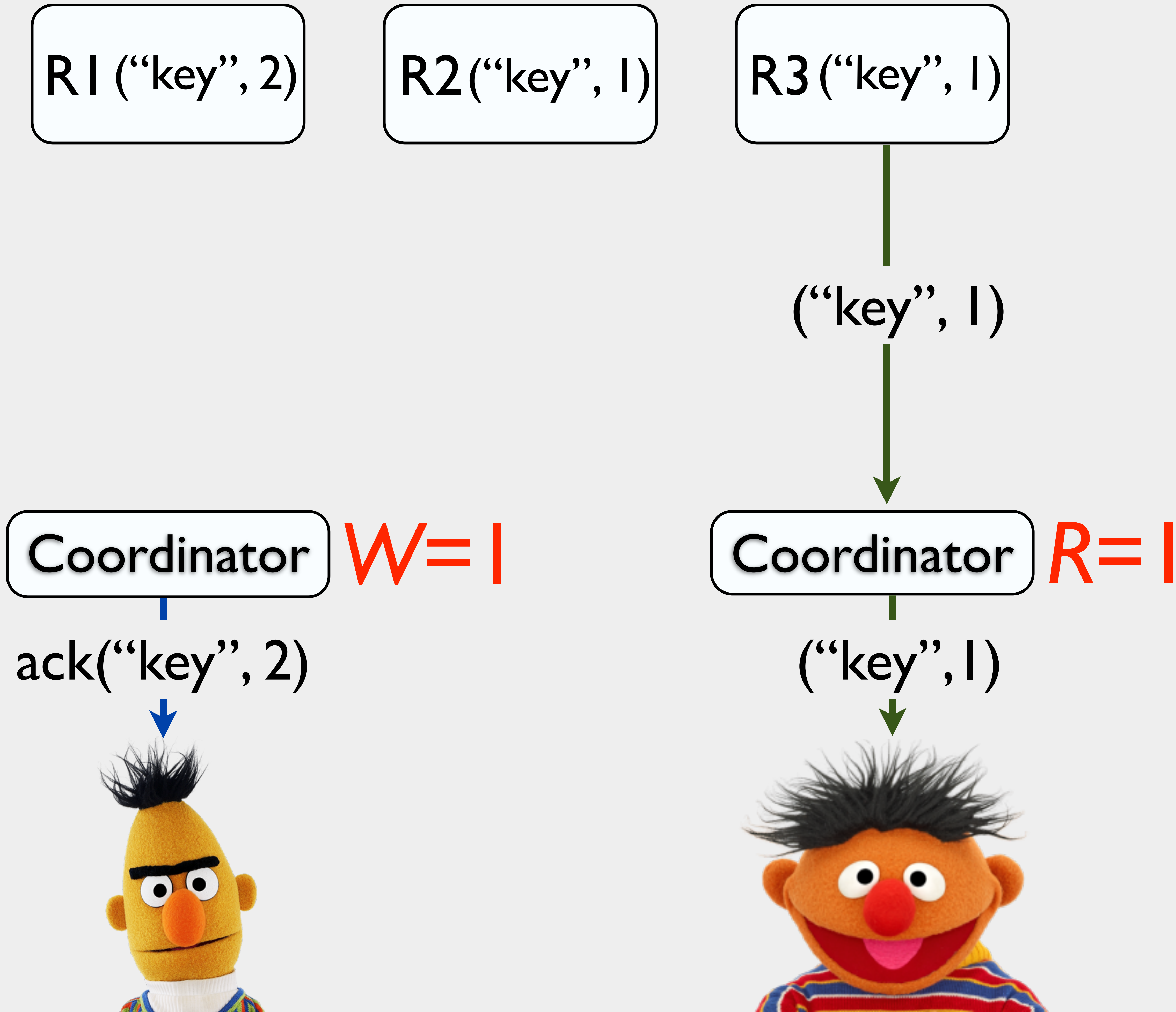
R2 (“key”, 1)

R3 (“key”, 1)

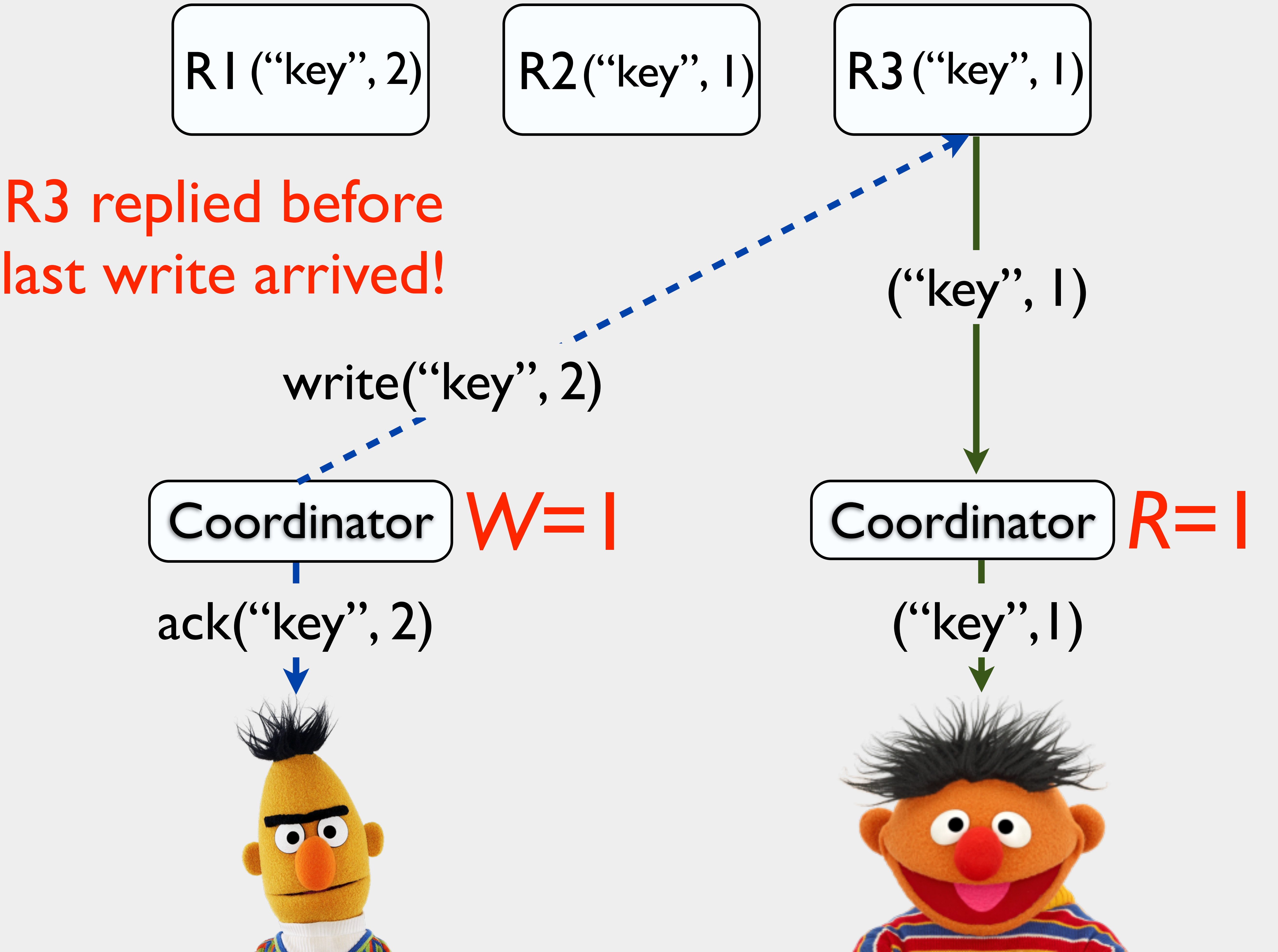
Coordinator W=1

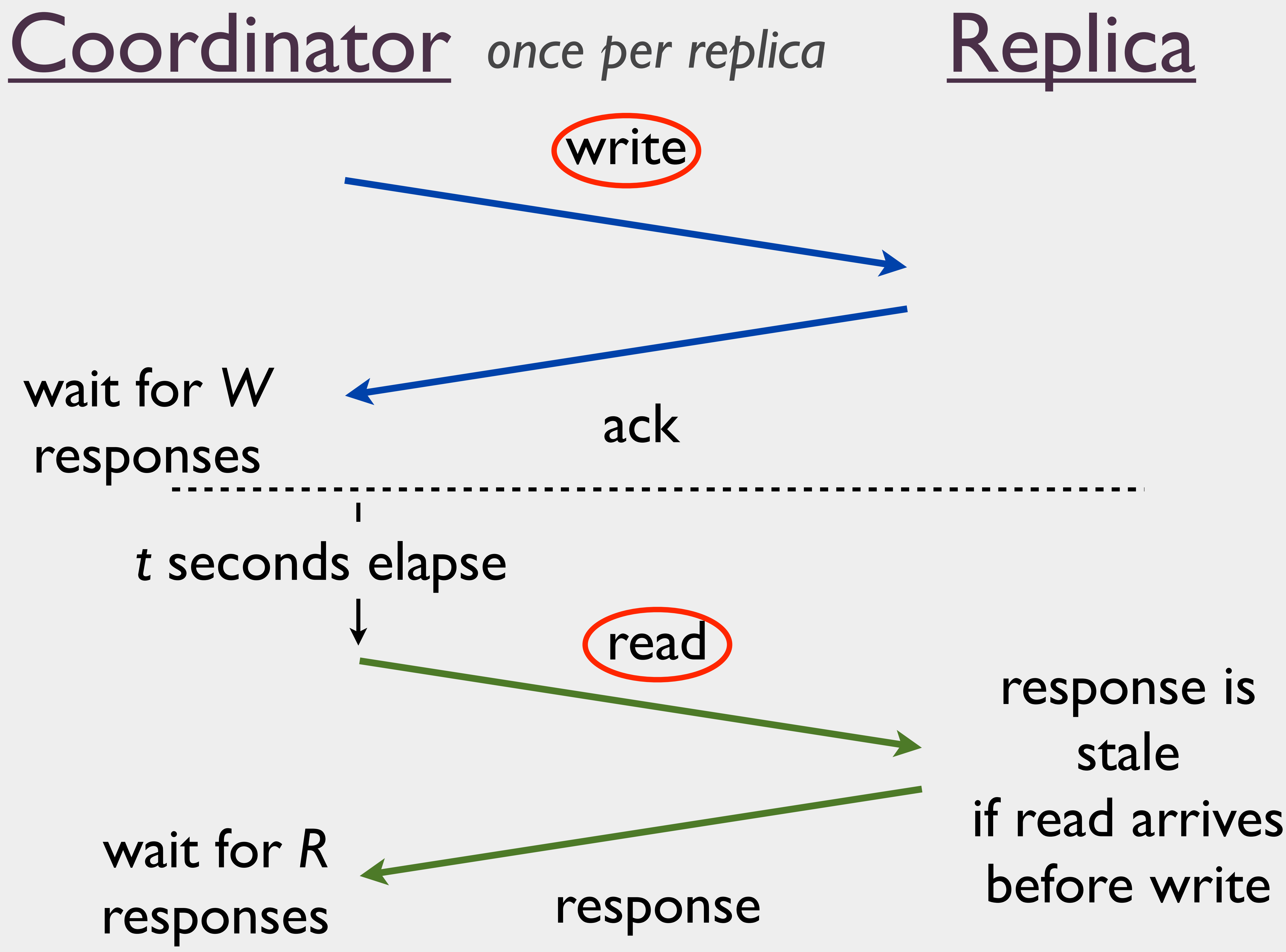
ack(“key”, 2)

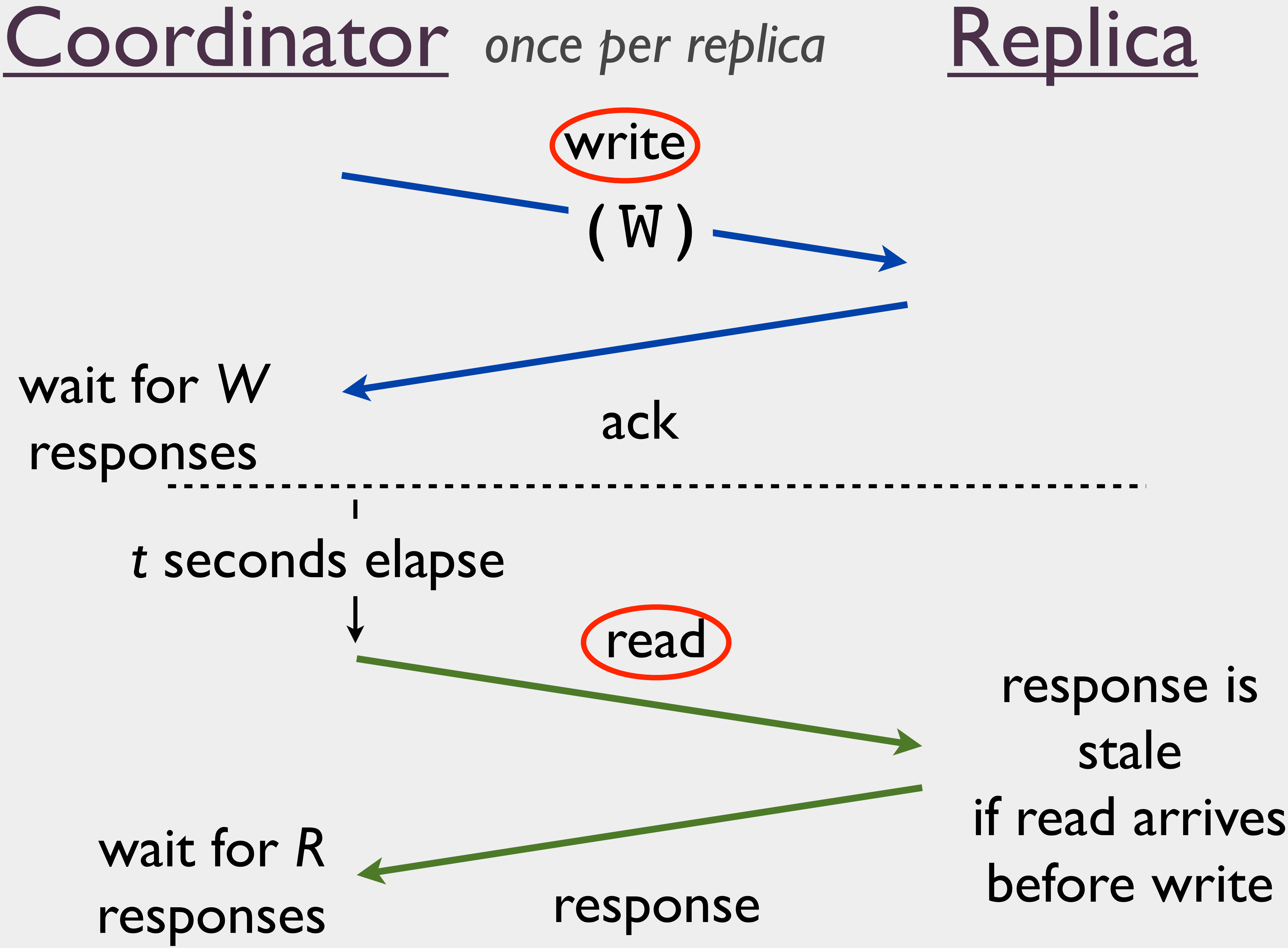
Coordinator R=1

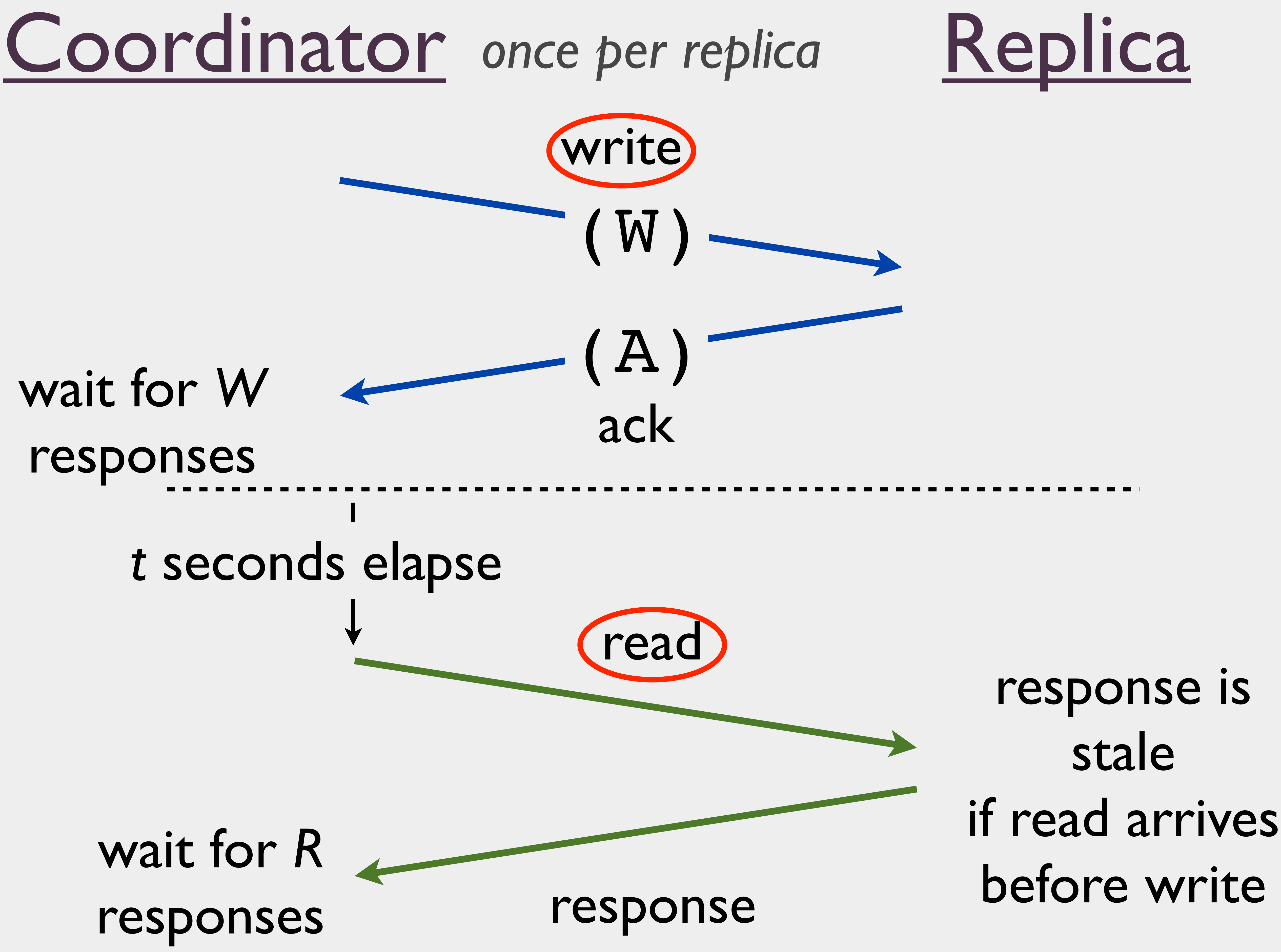


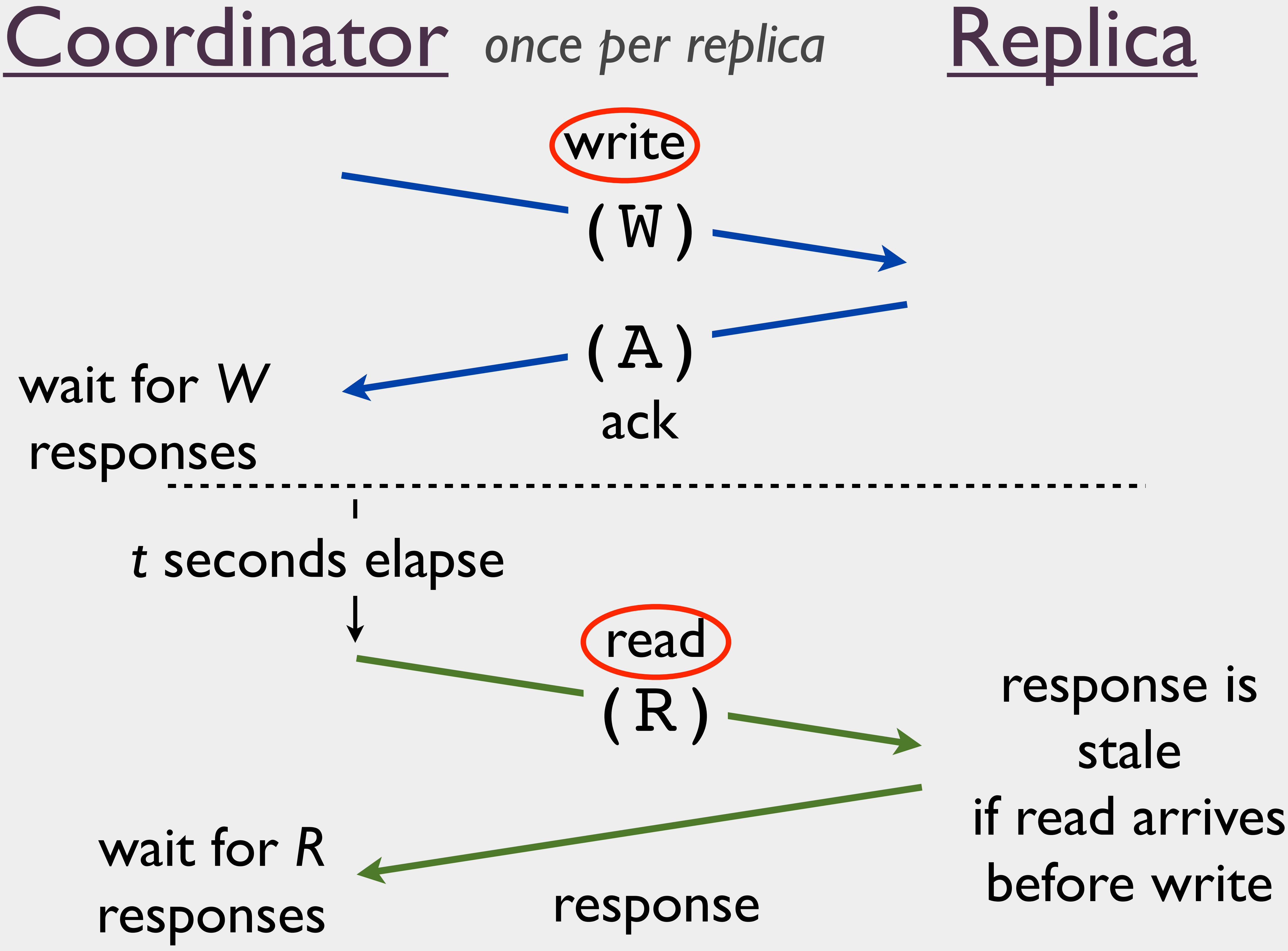
R1 ("key", 2)
R2 ("key", 1)
R3 ("key", 1)
("key", 1)
Coordinator
W=1
ack("key", 2)
Coordinator
R=1
("key",1)


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

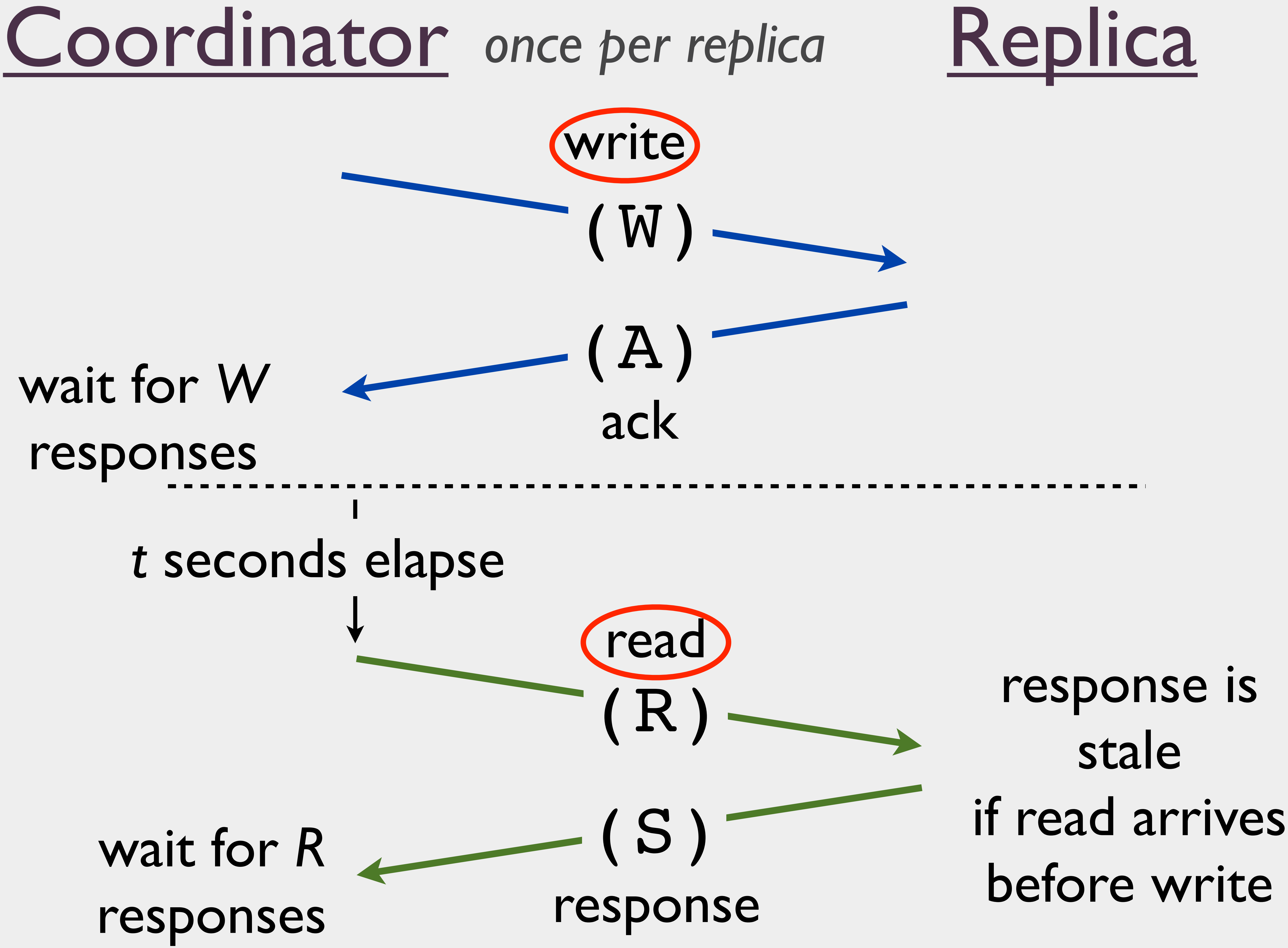

Coordinator
once per replica
Replica
write
wait for W responses
ack
t seconds elapse
read
response is stale if read arrives before write
wait for R responses
response


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

Solving *WARS*: hard

Monte Carlo methods: easy

To use *WARS*:

gather latency data

run simulation

Cassandra implementation validated simulations; available on Github

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%

Riak Defaults

N=3

R=2

W=2

2+2 > 3

Phew, I'm safe!

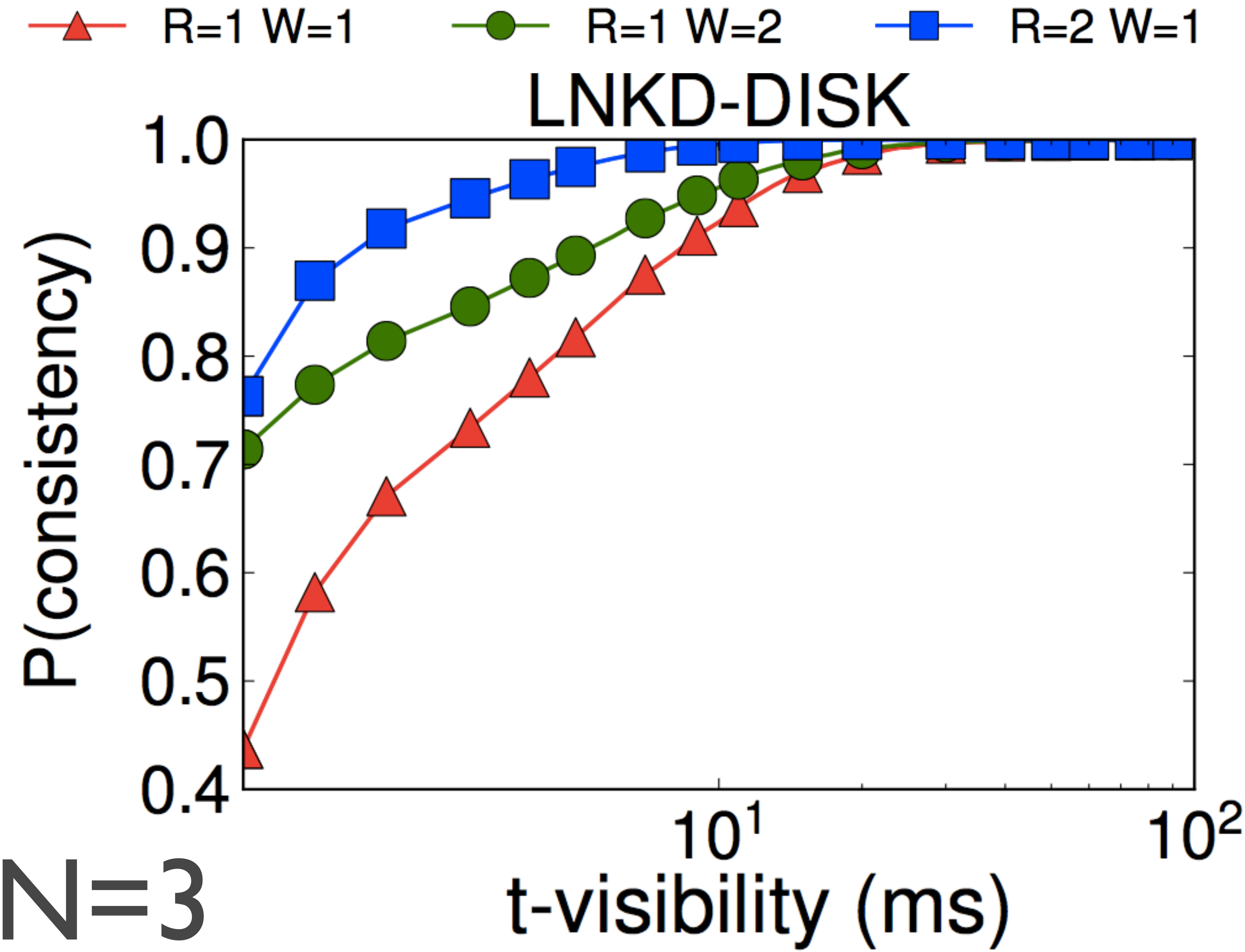
...but what's my latency cost?

Should I change?

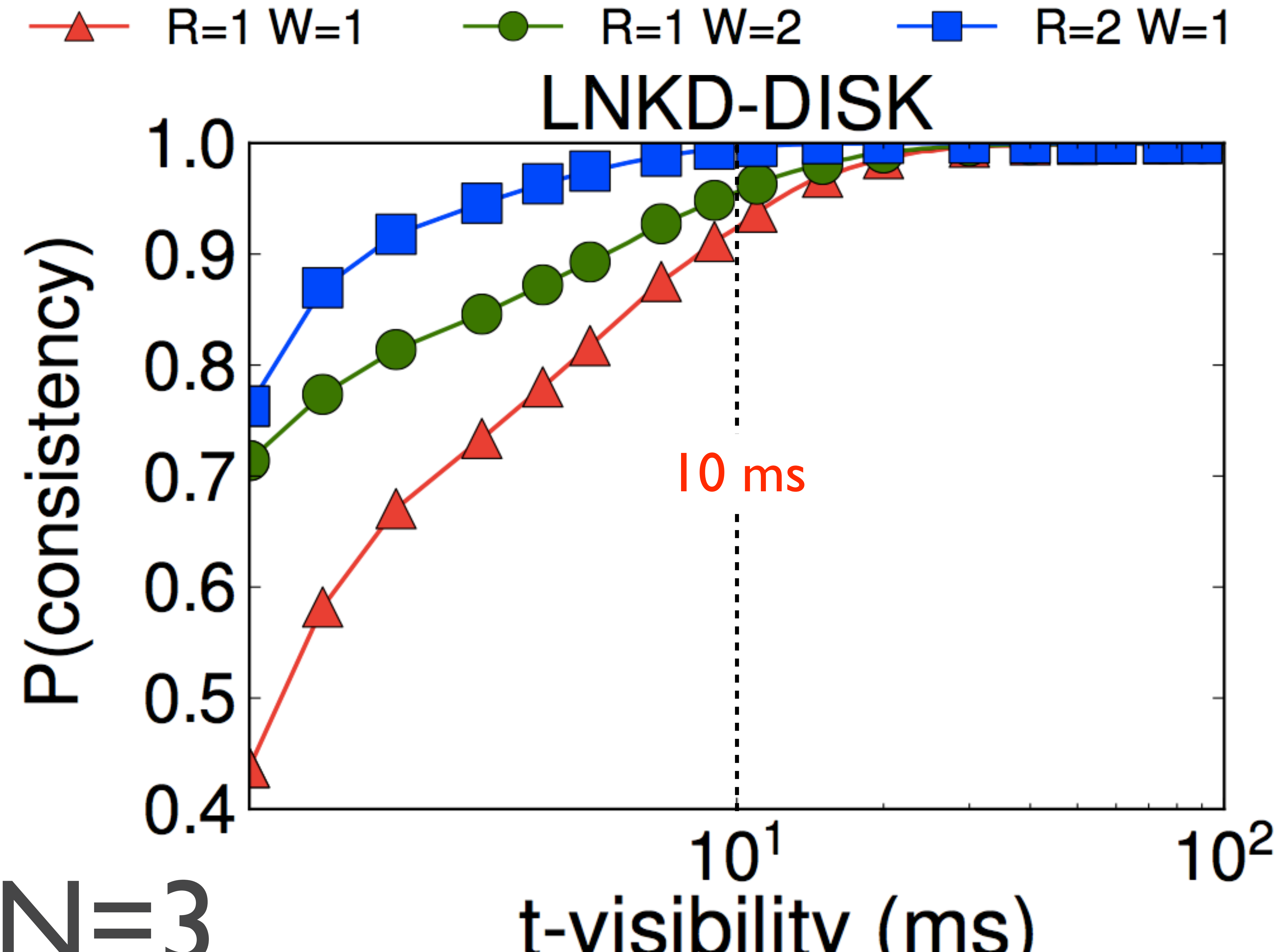
LinkedIn
150M+ users
built and uses Voldemort

Yammer
100K+ companies
uses Riak

Thanks to @strlen and @coda:
production latencies


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
10^1
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?

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

t = 13.6 ms

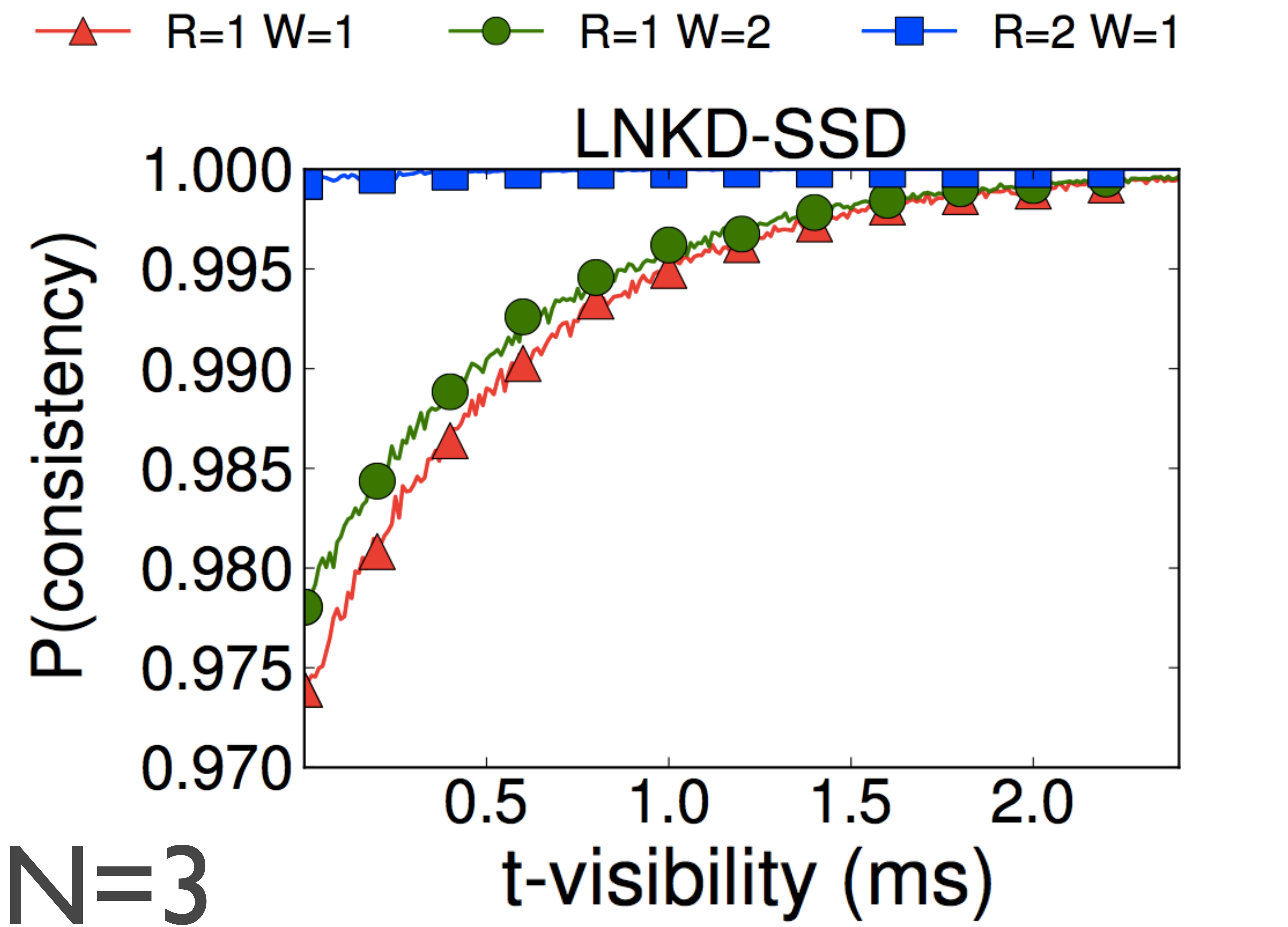
Latency: 12.53 ms

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

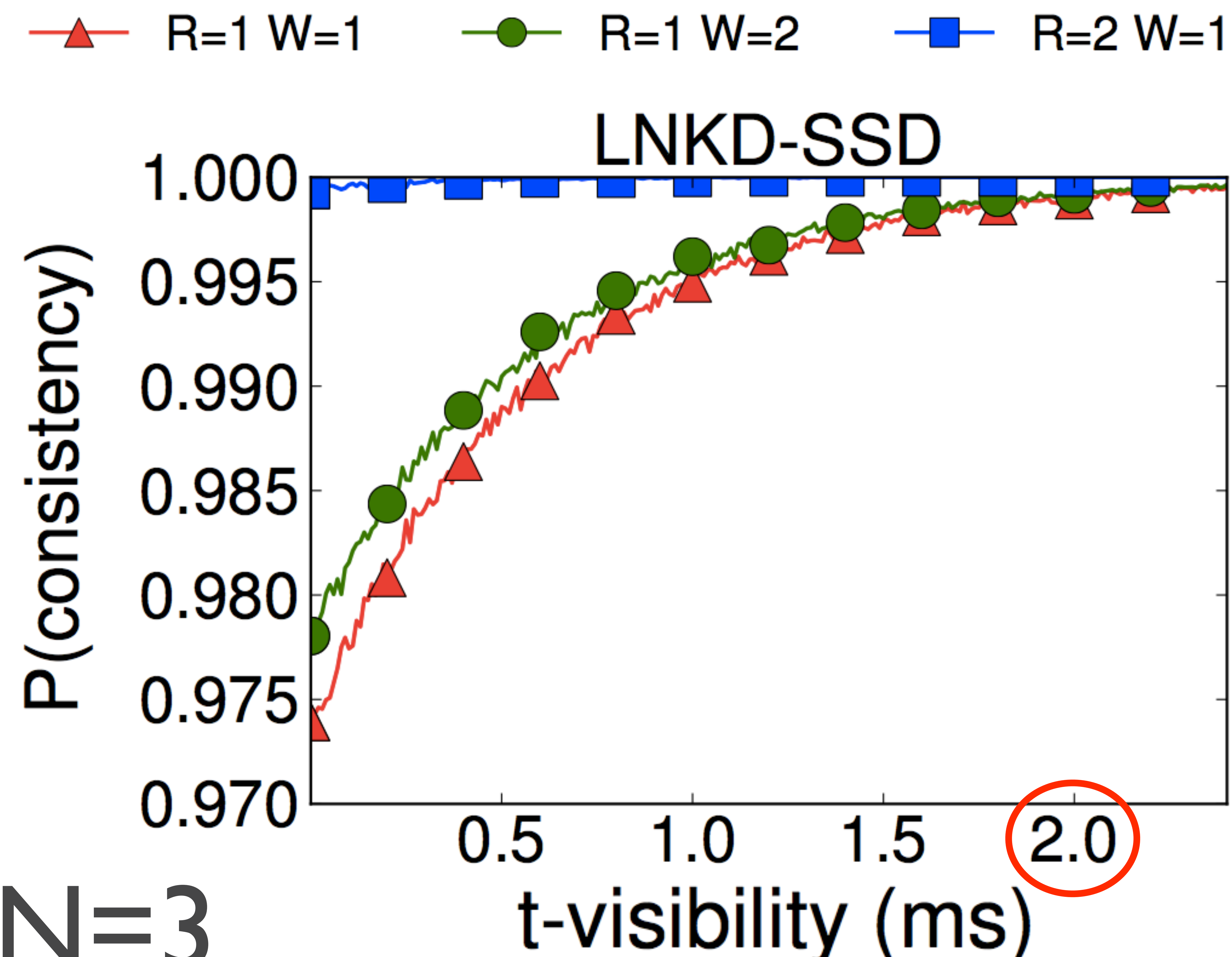
Latency: 15.01 ms

N=3

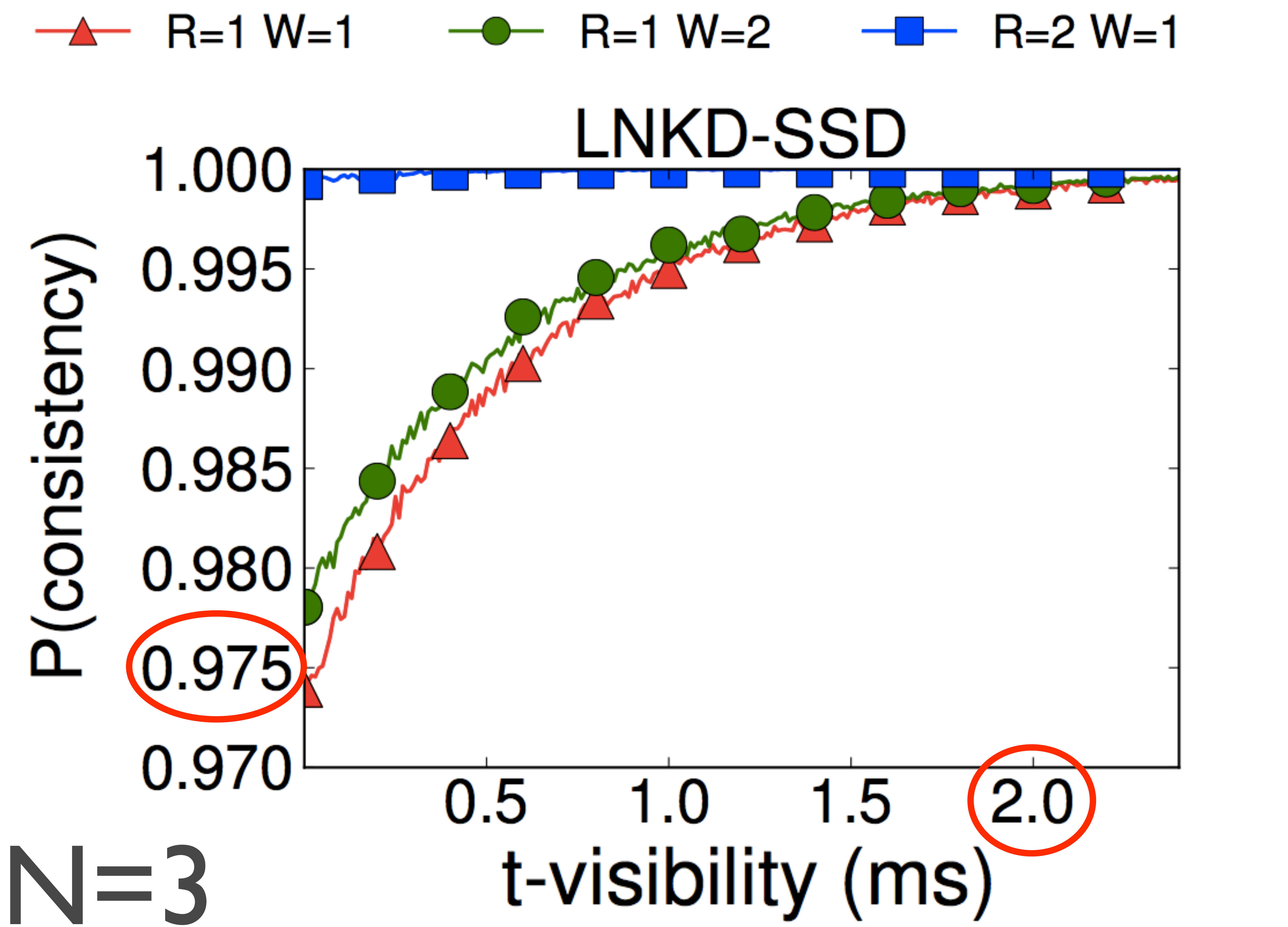
Latency is combined read and write latency at 99.9th percentile



R=1 W=1
R=1 W=2
R=2 W=1
LNKD-SSD
1.000
0.995
0.990
0.985
0.980
0.975
0.970
P(consistency)
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
1.000
0.995
0.990
0.985
0.980
0.975
0.970
P(consistency)
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

LNKD-SSD

59.5% faster

better payoff!

N=3

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

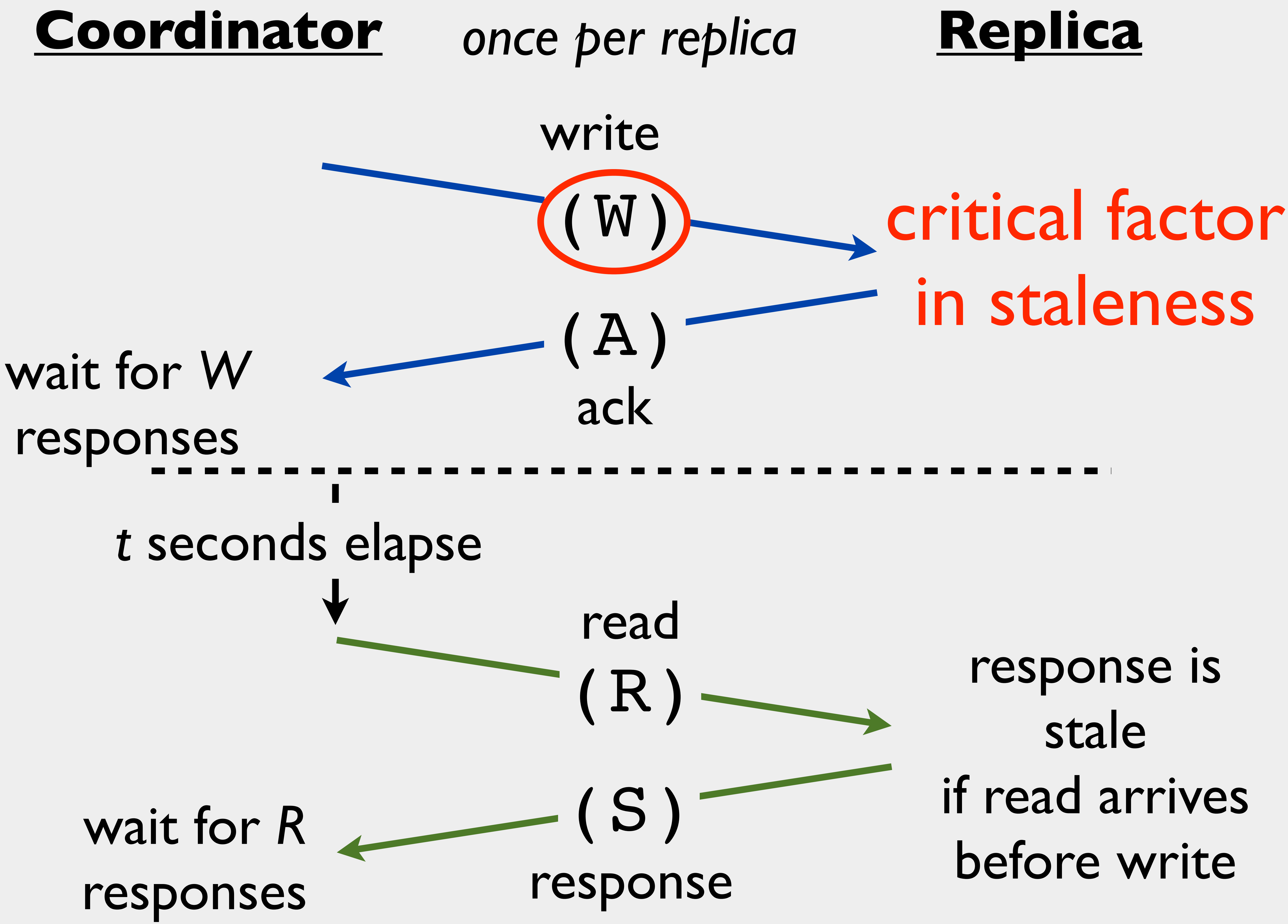
t = 1.85 ms

Latency: 1.32 ms

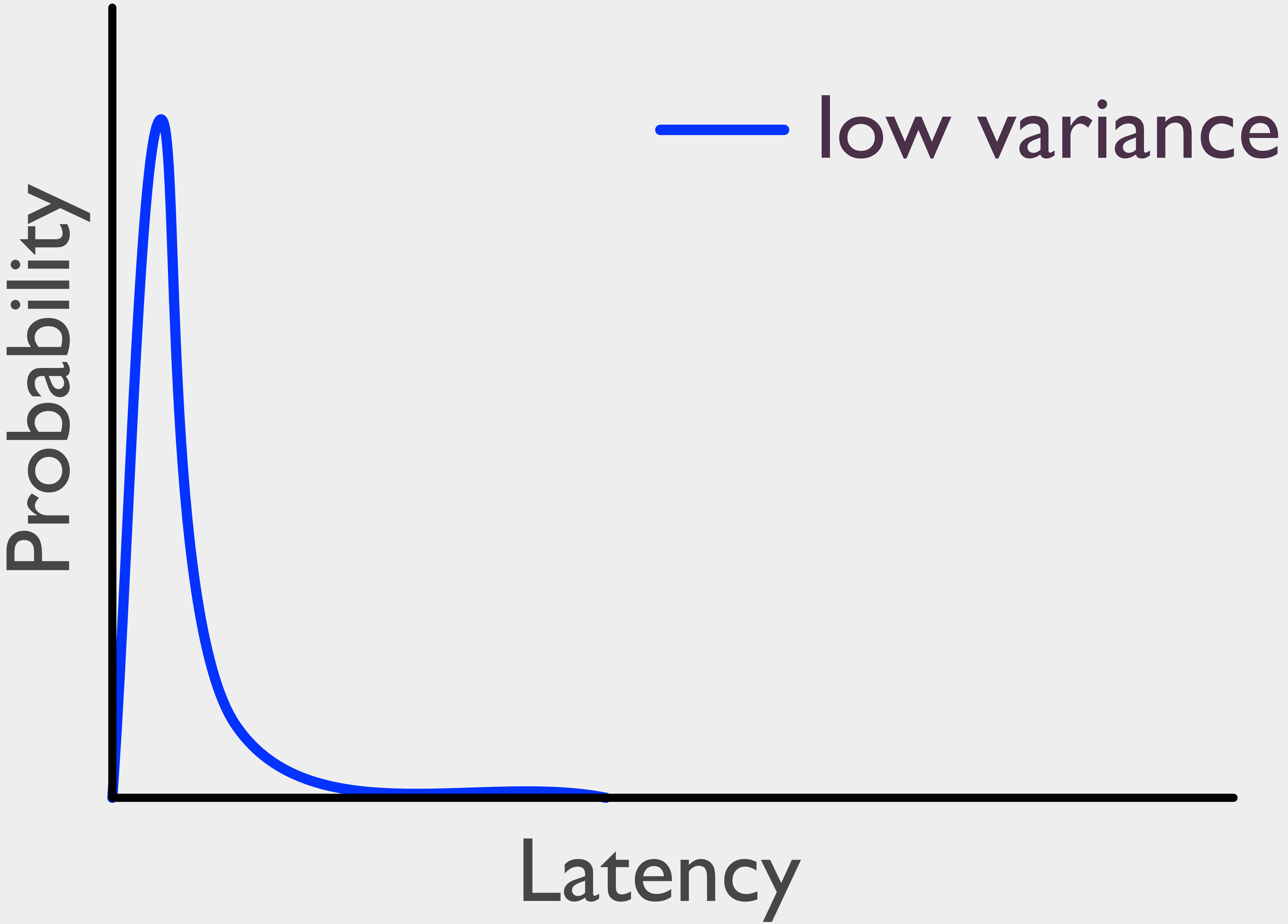
100% consistent reads:
R=3, W=1

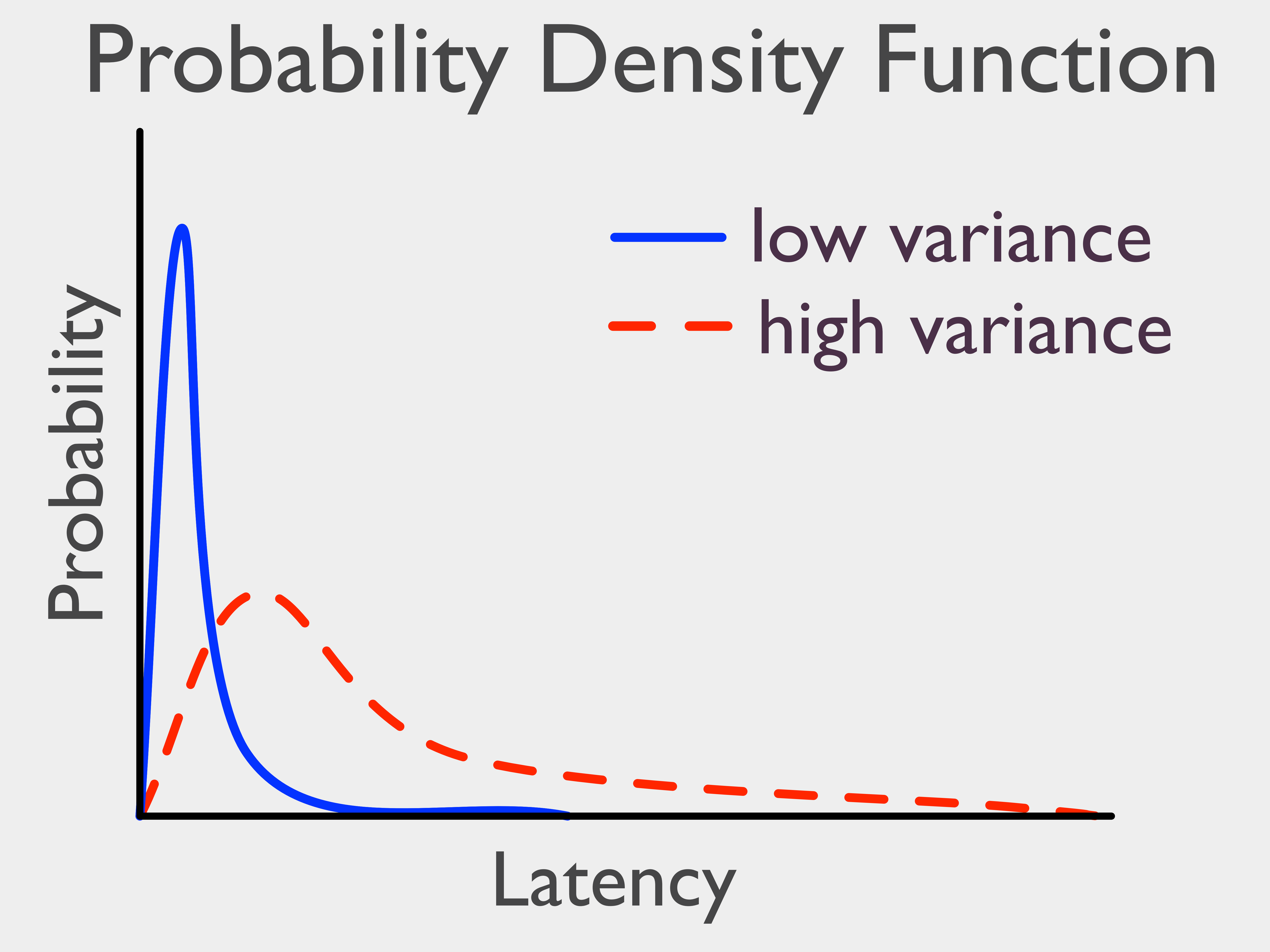
Latency: 4.20 ms

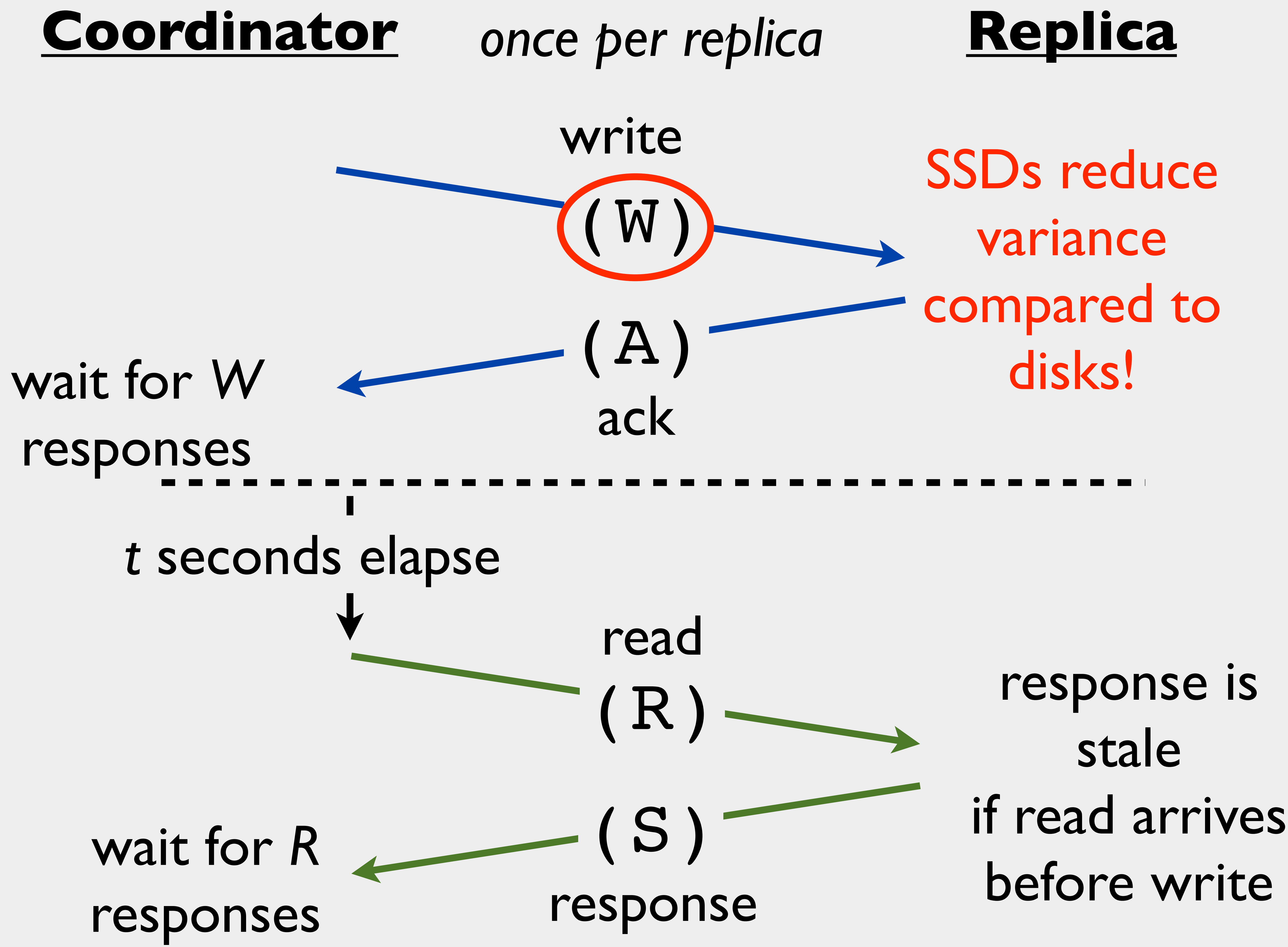
Latency is combined read and write latency at 99.9th percentile


Coordinator
once per replica
Replica
write
(W)
critical factor
in staleness
(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

Probability Density Function




Probability Density Function
low variance
high variance
Probability
Latency



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

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

81.1% faster

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

81.1% faster

even better payoff!

N=3

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

Riak Defaults

N=3
R=2
W=2

2+2 > 3

Phew, I'm safe!

...but what's my latency cost?

<u>Should I change?</u>

Is low latency worth it?

Is low latency worth it?

PBS can tell you.

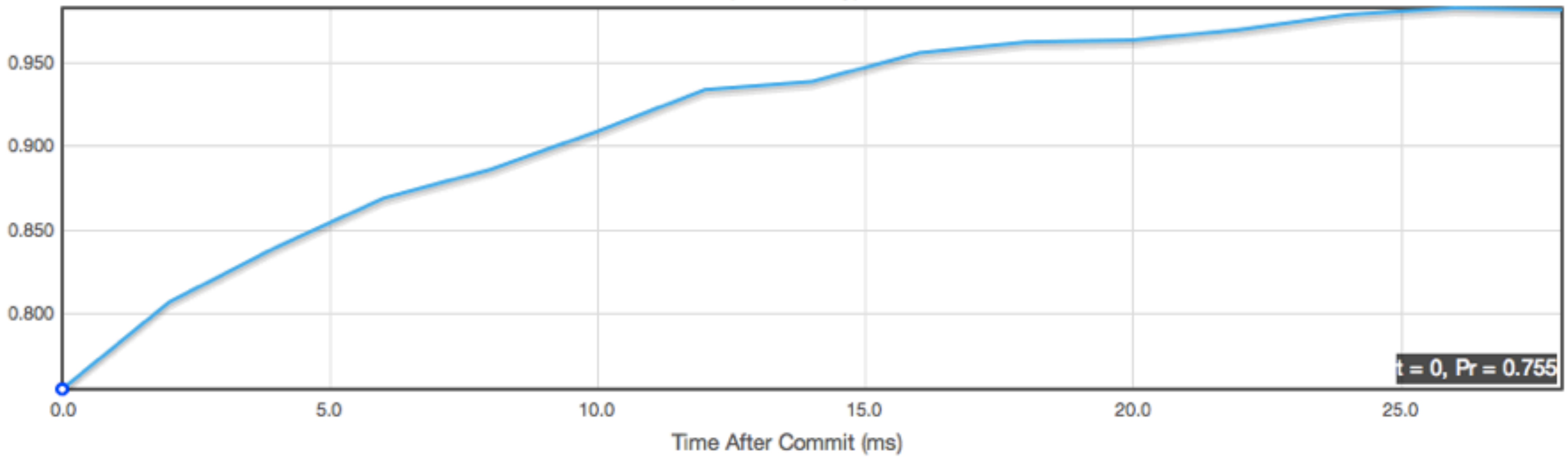
Is low latency worth it?

PBS can tell you.

(and PBS is easy)

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

P(Consistency)



(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

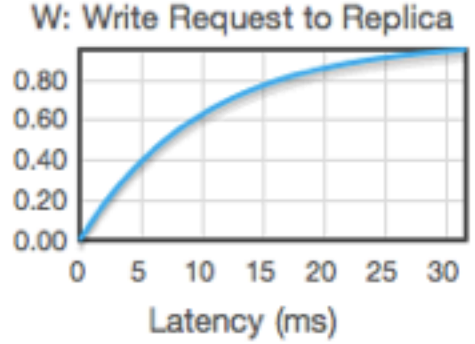
Read Latency: Median 8.43 ms, 99.9th %ile 36.97 ms
Write Latency: Median 8.38 ms, 99.9th %ile 38.28 ms

Tolerable Staleness: 1 version

Accuracy: 2500 iterations/point

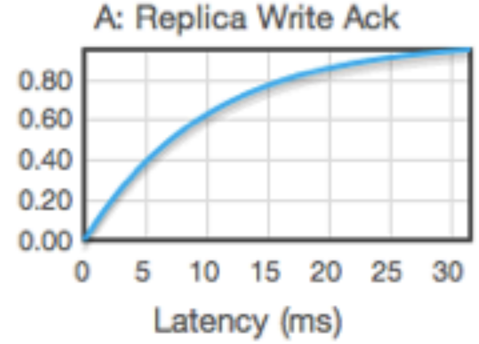
Operation Latency: Exponentially Distributed CDFs

W: Write Request to Replica



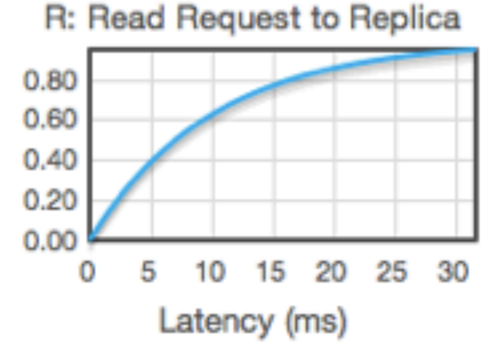
λ 0.100

A: Replica Write Ack



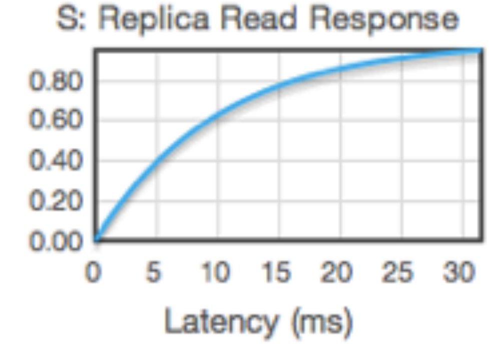
λ 0.100

R: Read Request to Replica



λ 0.100

S: Replica Read Response



λ 0.100

Workflow

1. Metrics
2. Simulation
3. Set *N*, *R*, *W*
4. Profit

PBS

what: consistency prediction

why: weak consistency is fast

how: measure latencies
use *WARS* model

strong consistency

low latency


SUIT
Be more
PBS

PBS

latency vs. consistency trade-offs

fast and simple modeling

large benefits

be more

bailis.org/projects/pbs/#demo

@pbailis

VLDB 2012 early print

tinyurl.com/pbspaper

cassandra patch

github.com/pbailis/cassandra-pbs

Extra Slides

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

Give me

(and academia)

failure data!

In paper:

- Closed-form analysis
- Monotonic reads
- Staleness detection
- Varying *N*
- WAN model
- Production latency data

tinyurl.com/pbspaper

t-visibility depends on:

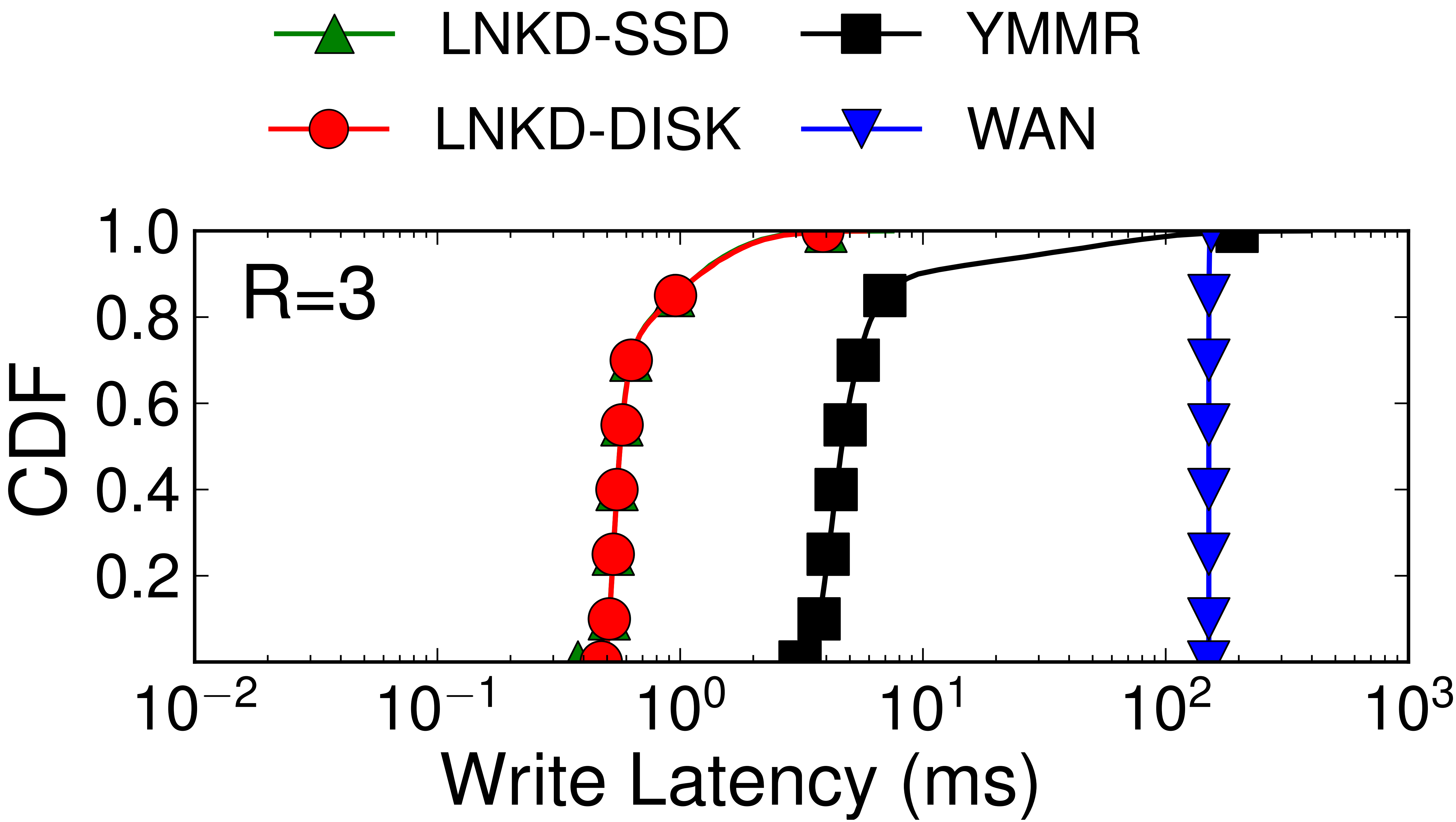
1) message delays
2) background version exchange (anti-entropy)

t-visibility depends on:

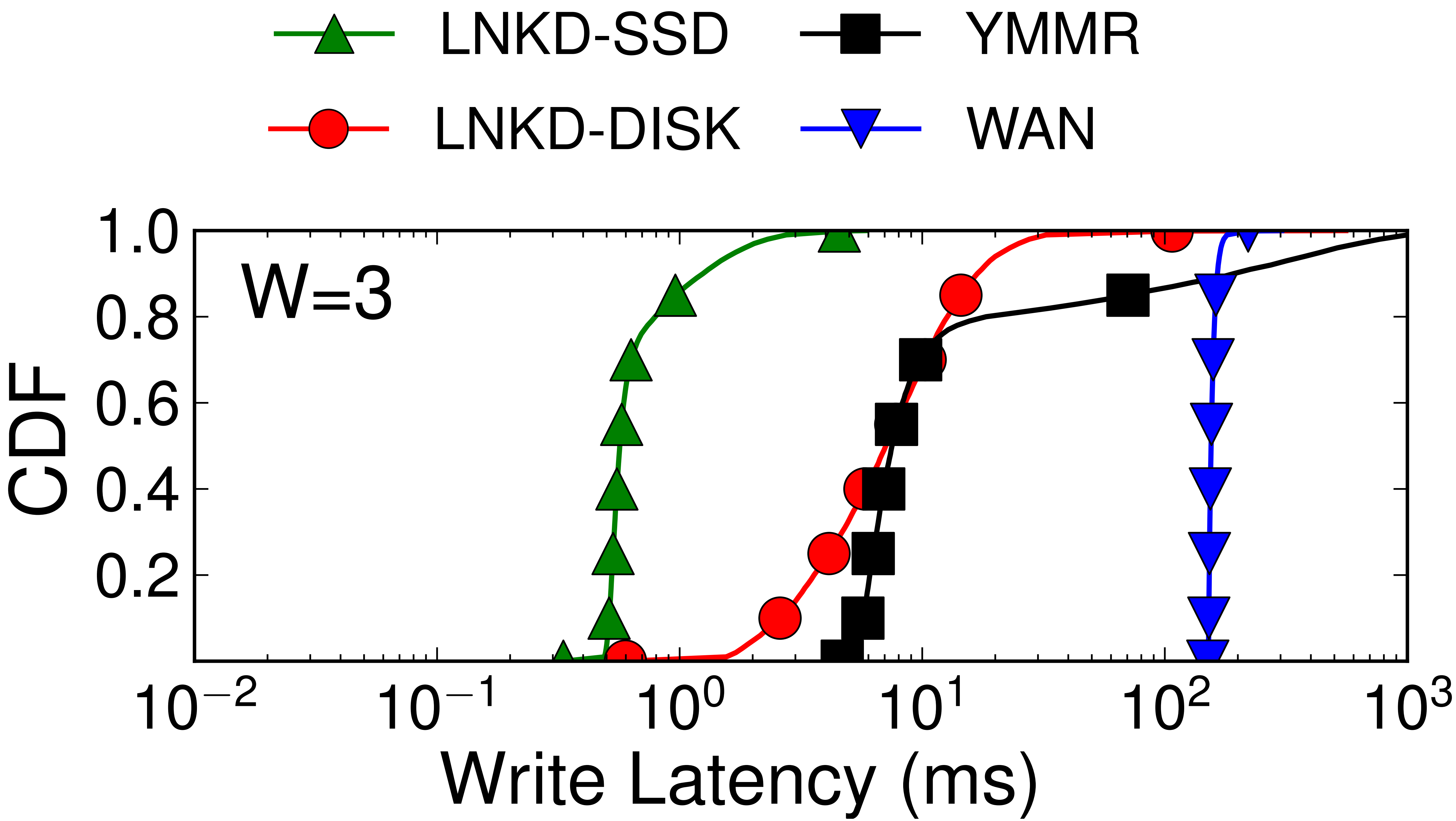
1) message delays
2) ~~background version exchange (anti-entropy)~~

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

Focus on message delays first

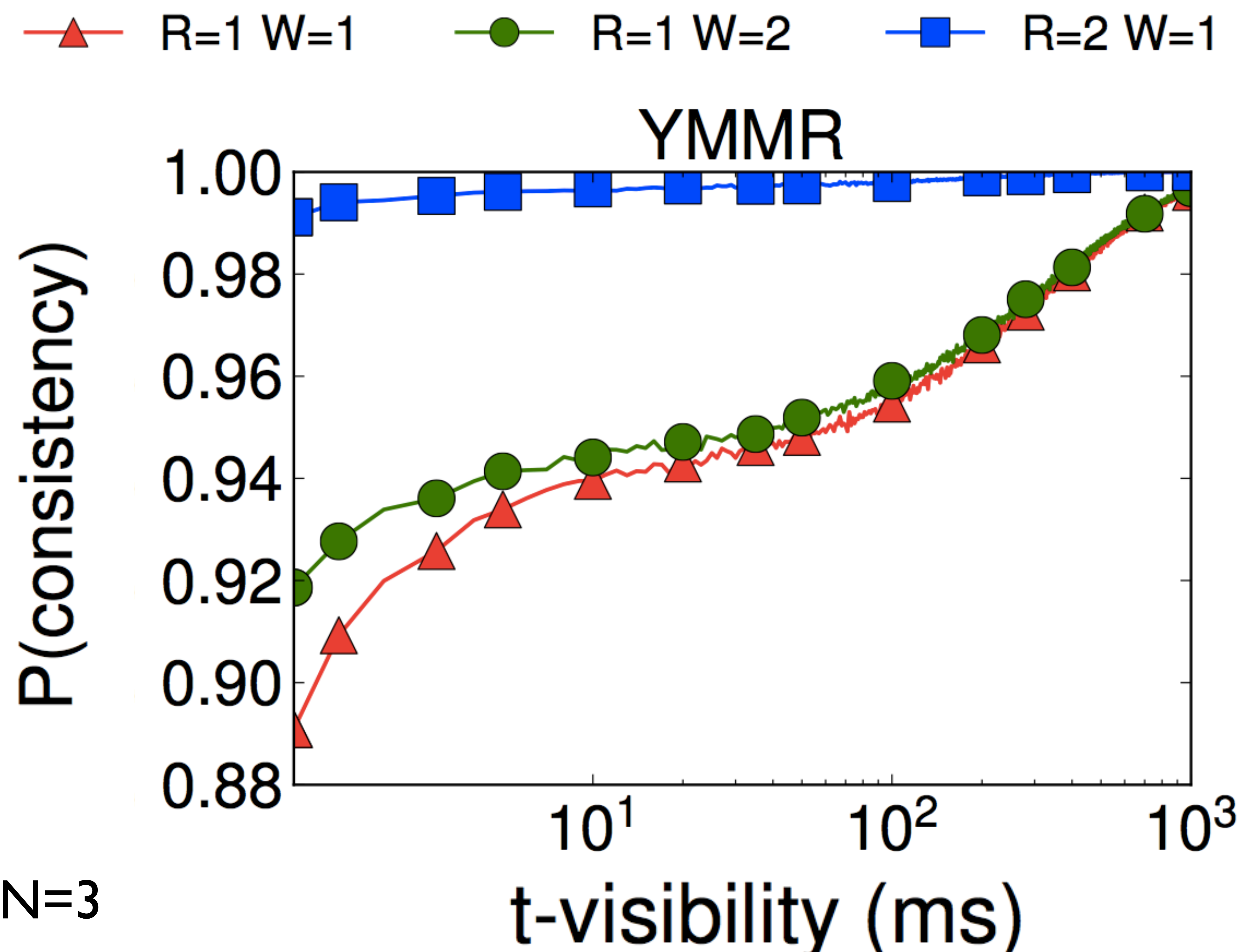


N=3 *(LNKD-SSD and LNKD-DISK identical for reads)*



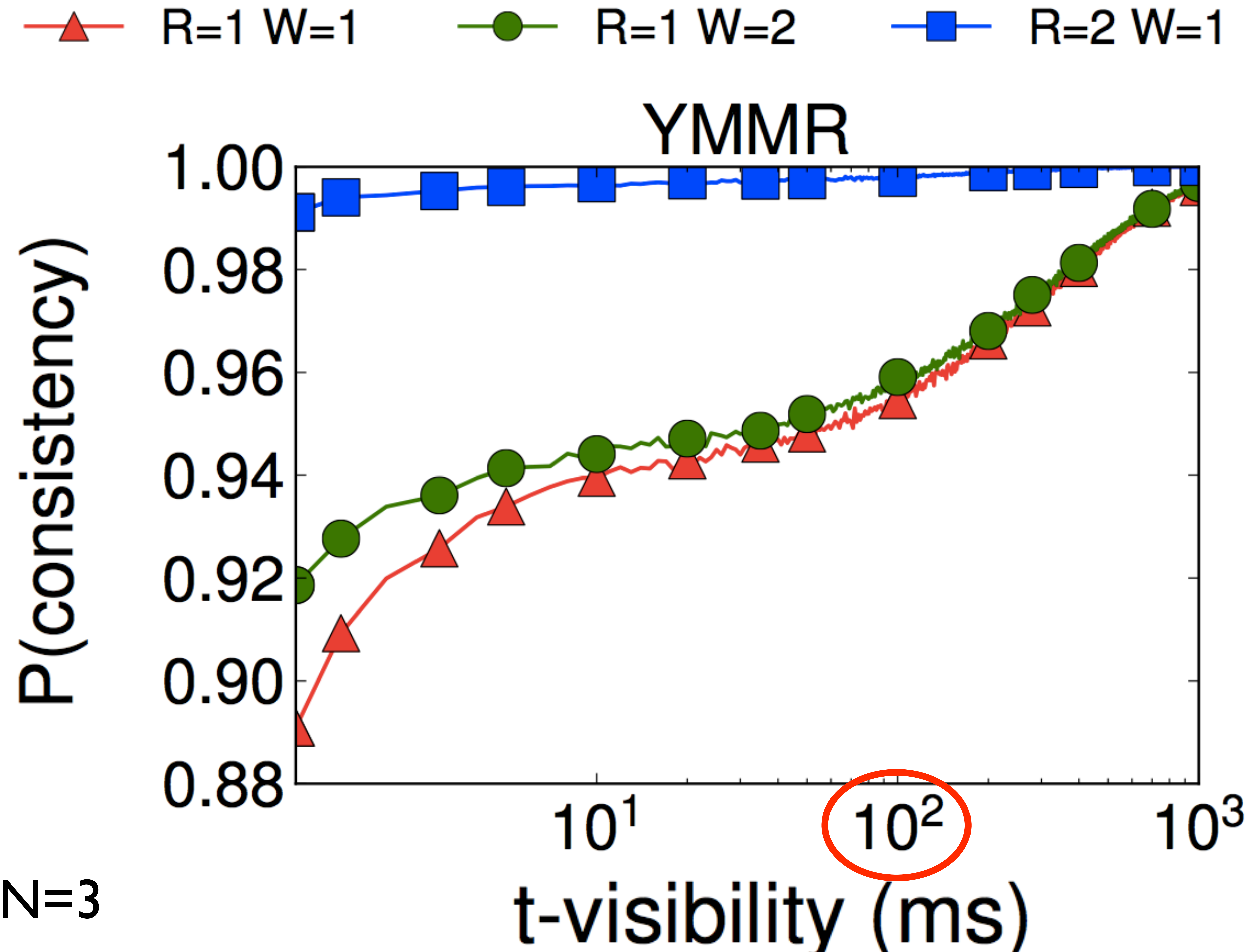
LNKD-SSD
YMMR
LNKD-DISK
WAN
W=3
CDF
1.0
0.8
0.6
0.4
0.2
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
Write Latency (ms)

N=3

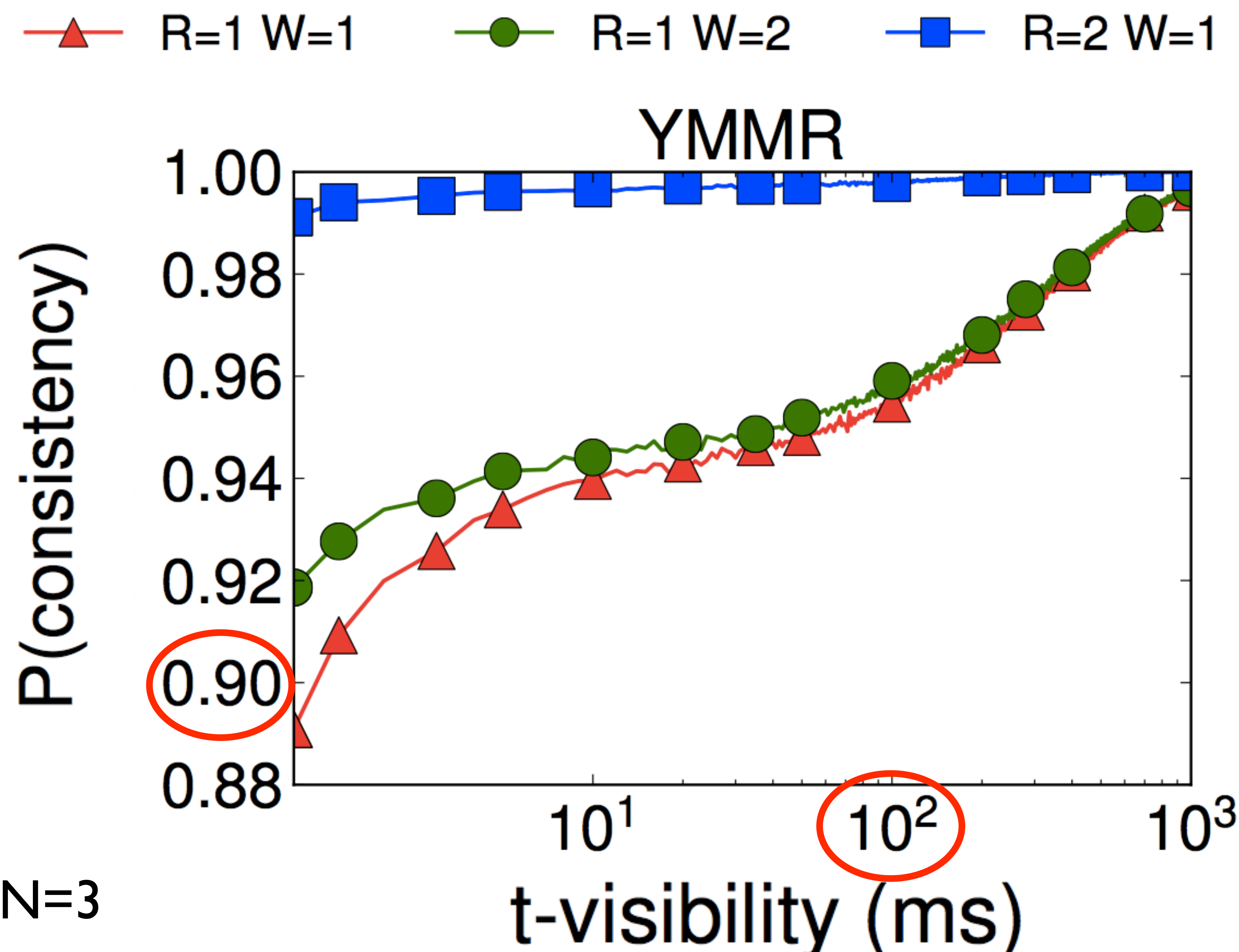


R=1 W=1
R=1 W=2
R=2 W=1
YMMR
P(consistency)
1.00
0.98
0.96
0.94
0.92
0.90
0.88
10^1
10^2
10^3
t-visibility (ms)

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)

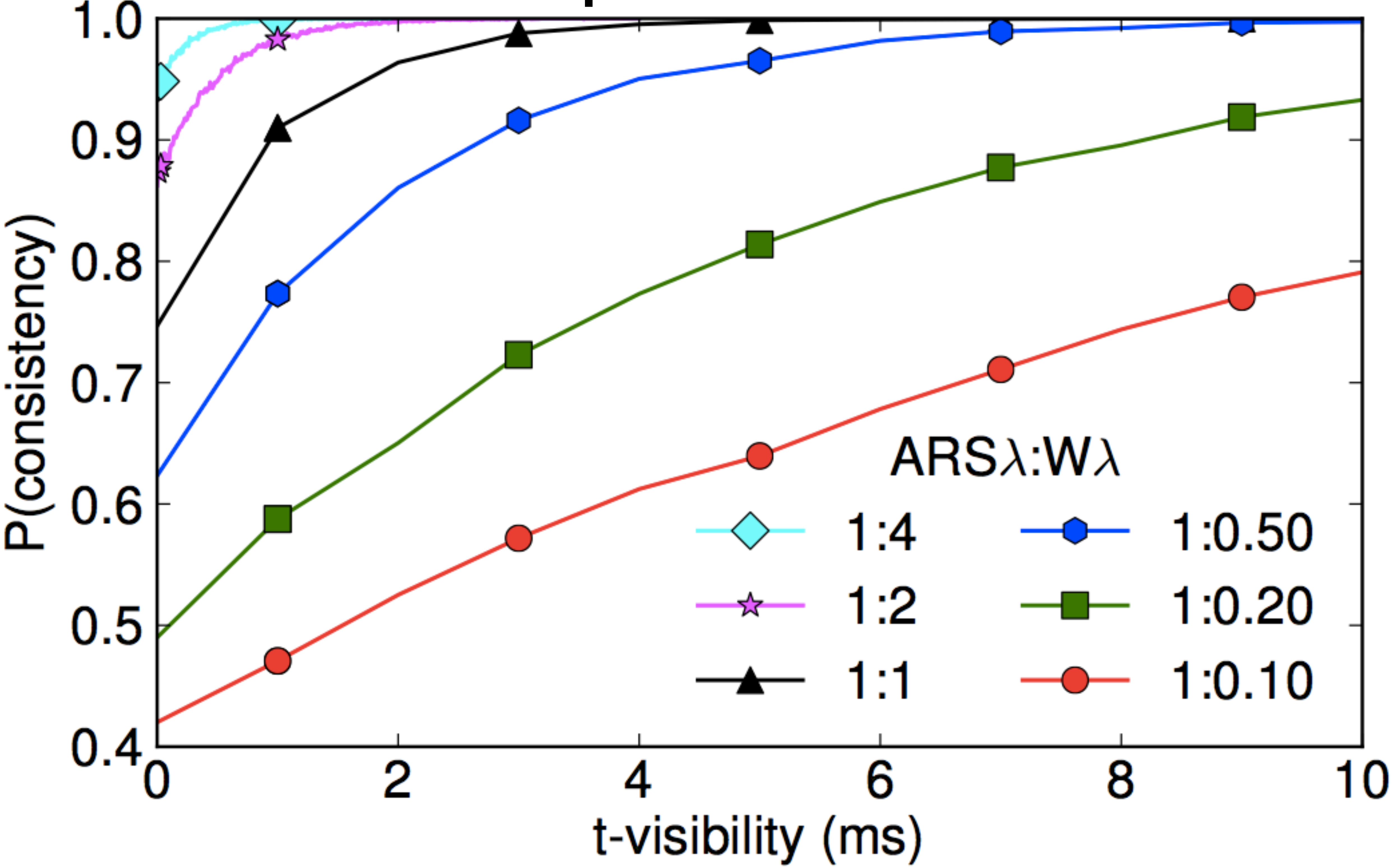
N=3



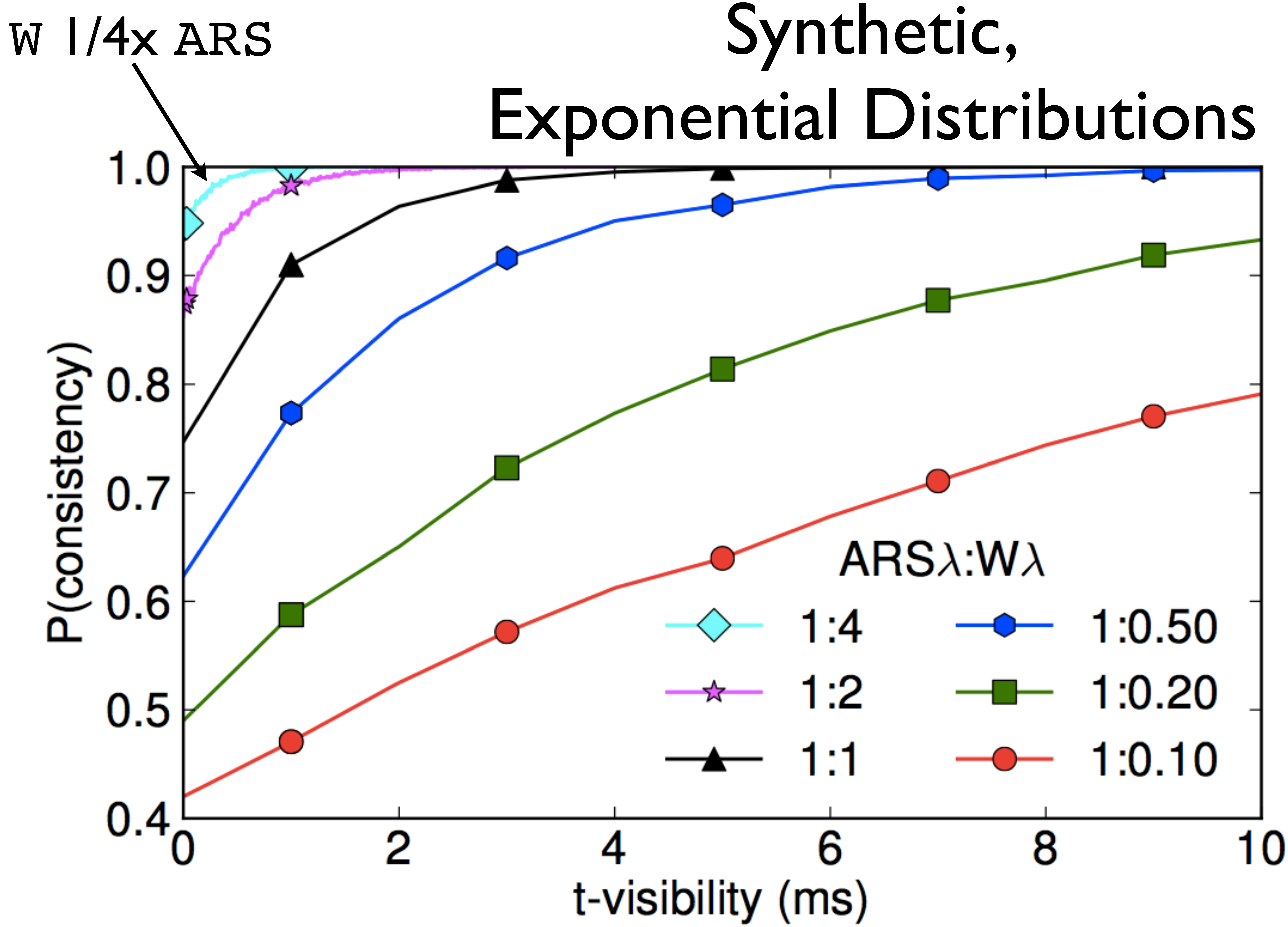
R=1 W=1
R=1 W=2
R=2 W=1
YMMR
P(consistency)
1.00
0.98
0.96
0.94
0.92
0.90
0.88
10^1
10^2
10^3
t-visibility (ms)

N=3

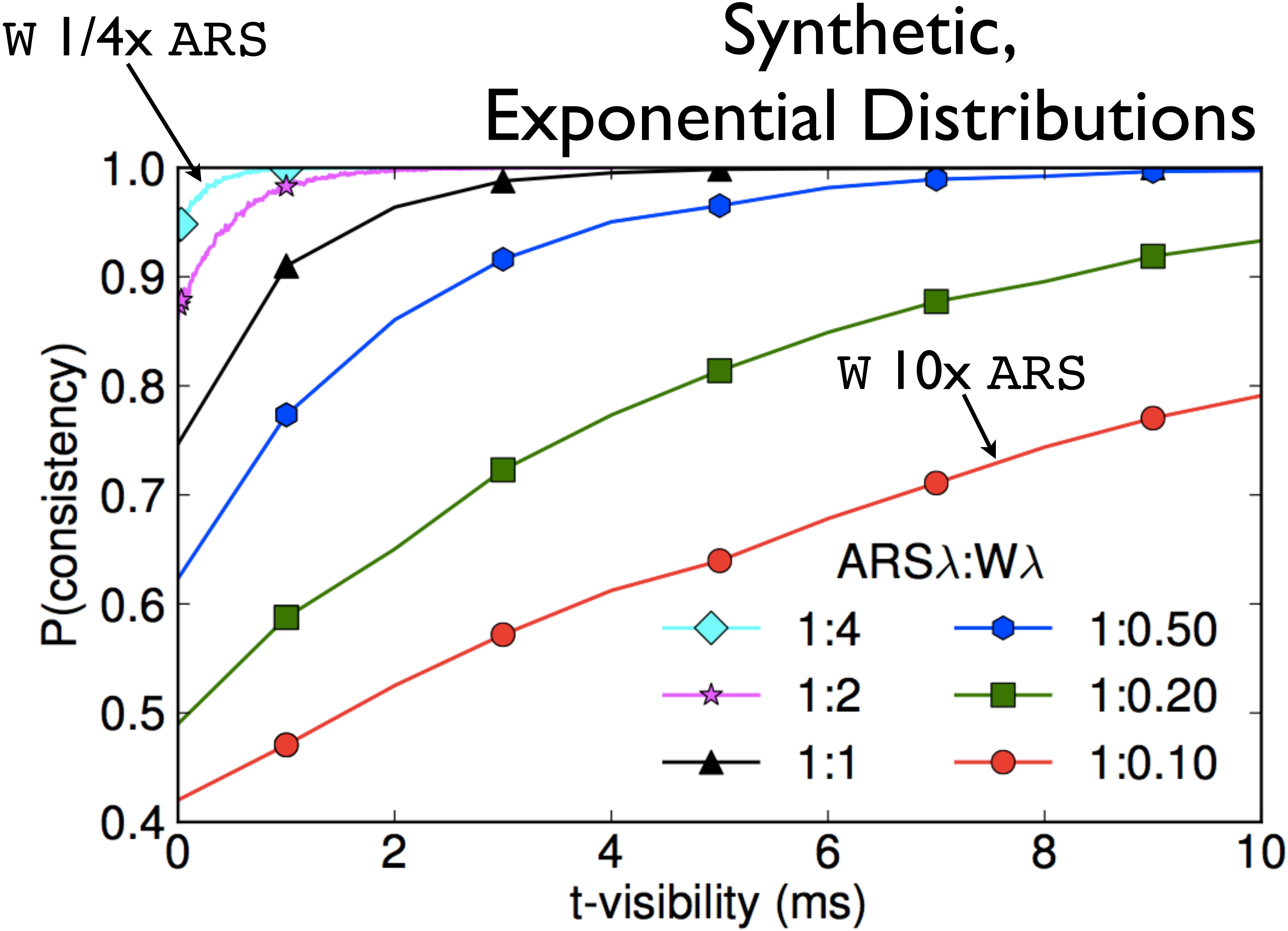
Synthetic, Exponential Distributions

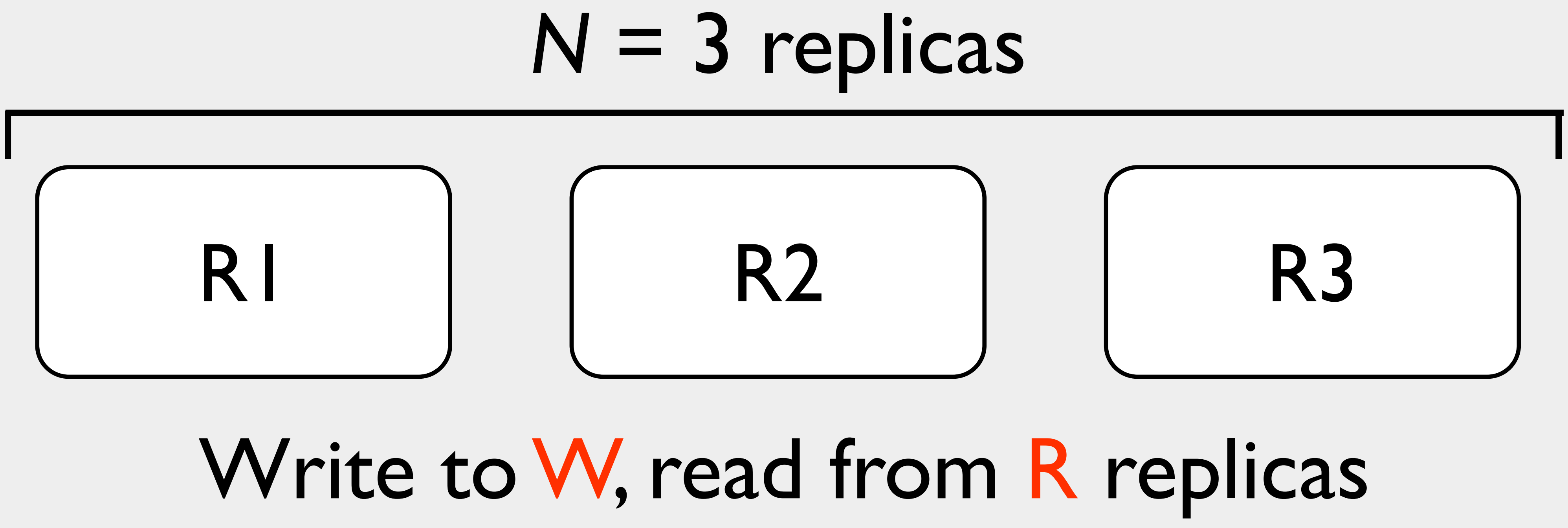


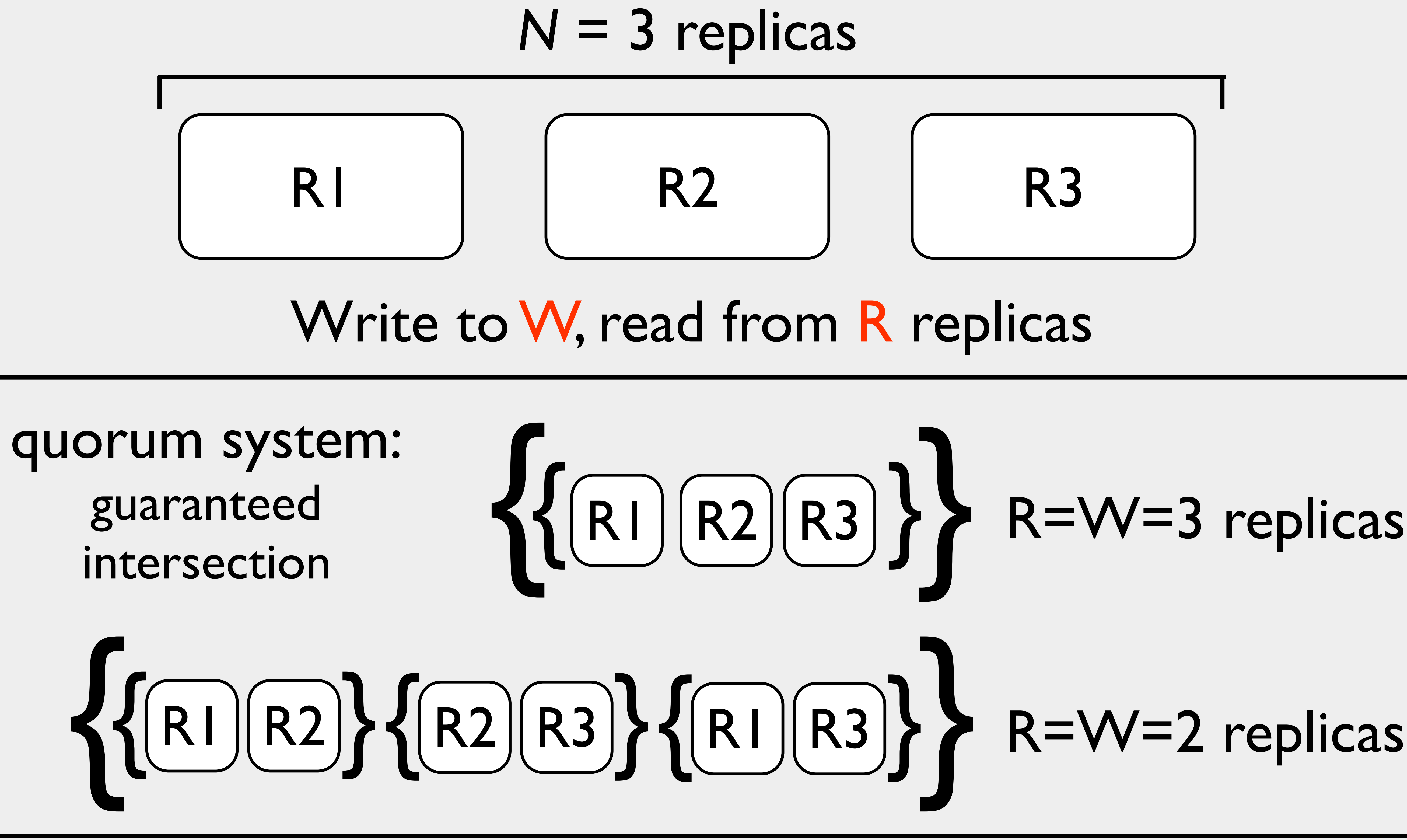
Synthetic, Exponential Distributions

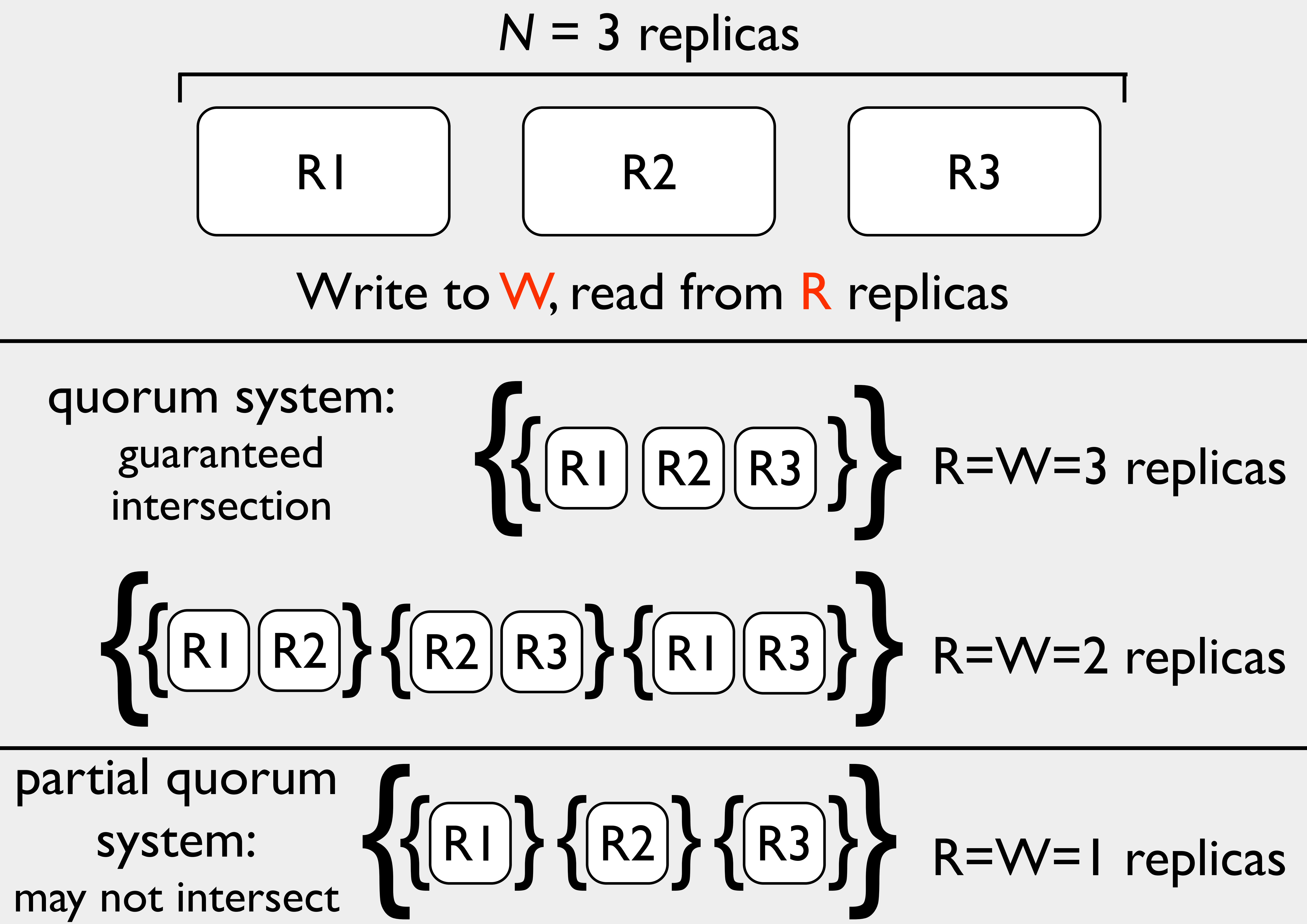


Synthetic, Exponential Distributions




N = 3 replicas
R1
R2
R3
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