PBS

VLDB 2012

Probabilistically Bounded Staleness for Practical Partial Quorums

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

UC Berkeley

PBS

NOTE TO READER

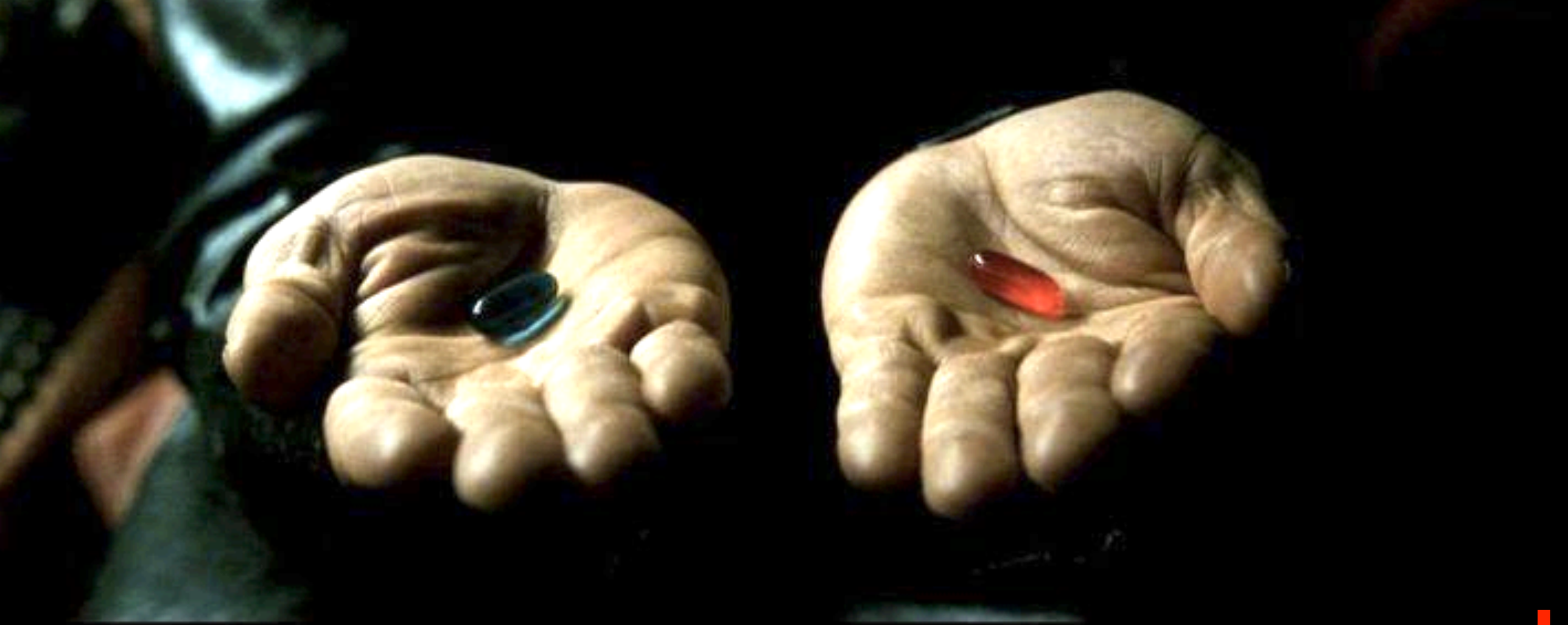
watch a recording
(of an earlier talk) at:
http://vimeo.com/37758648

PBS

NOTE TO READER

play with a live demo at:

http://pbs.cs.berkeley.edu/#demo



strong consistency

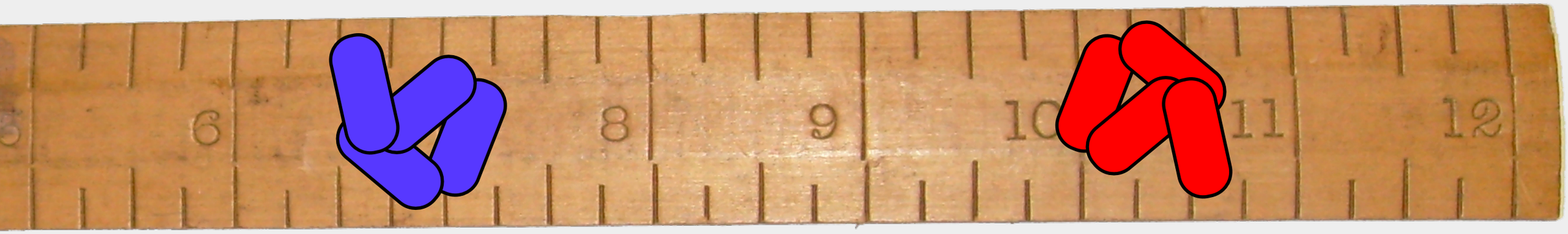
higher latency

eventual consistency

lower latency

consistency

is a binary choice

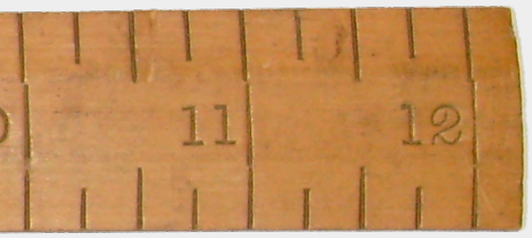


strong eventual

our focus:

latency vs. consistency

informed by practice



---

not in this talk:

availability, partitions, failures

our contributions

quantify eventual consistency:

wall-clock time ("how eventual?")

versions ("how consistent?")

analyze real-world systems:

EC is often strongly consistent

describe when and why

intro

system model

practice

metrics

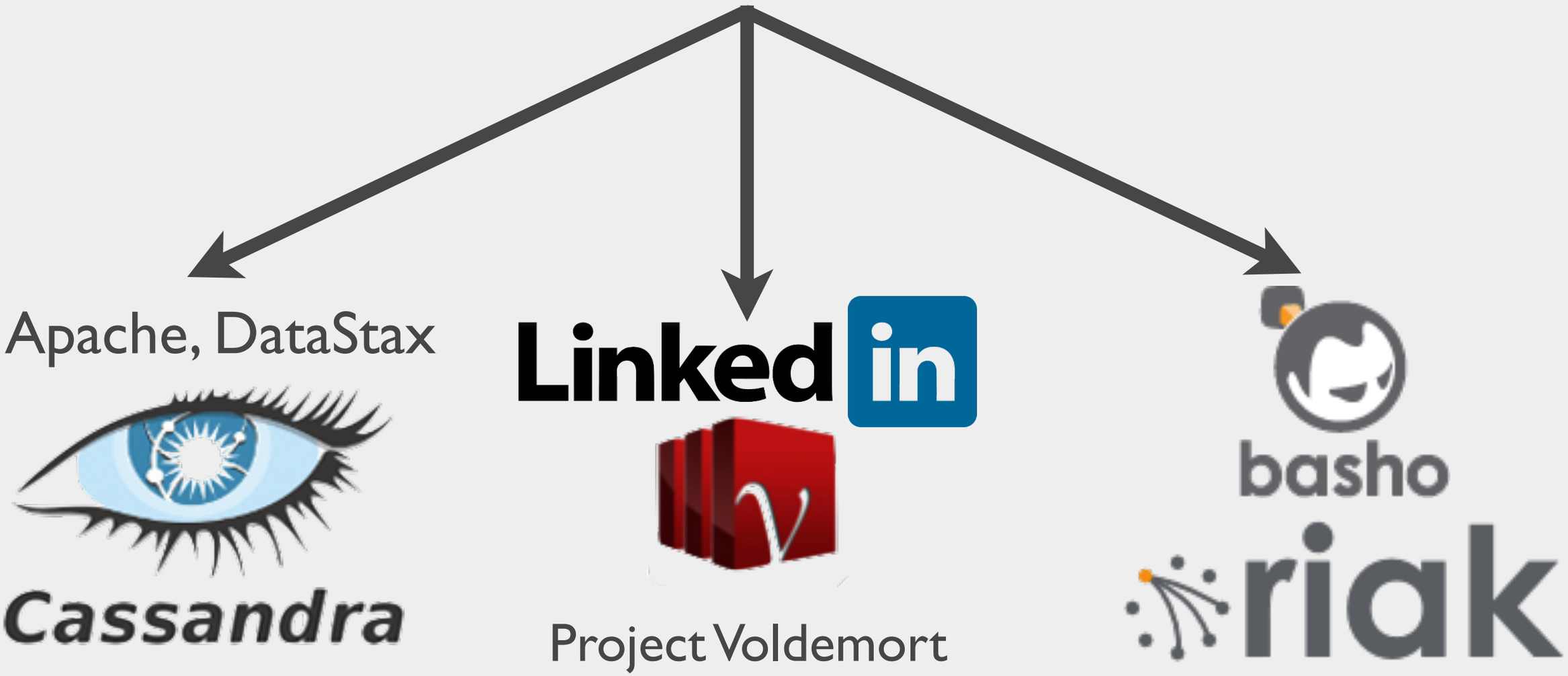
insights

integration

Dynamo:

Amazon's Highly Available Key-value Store

SOSP 2007



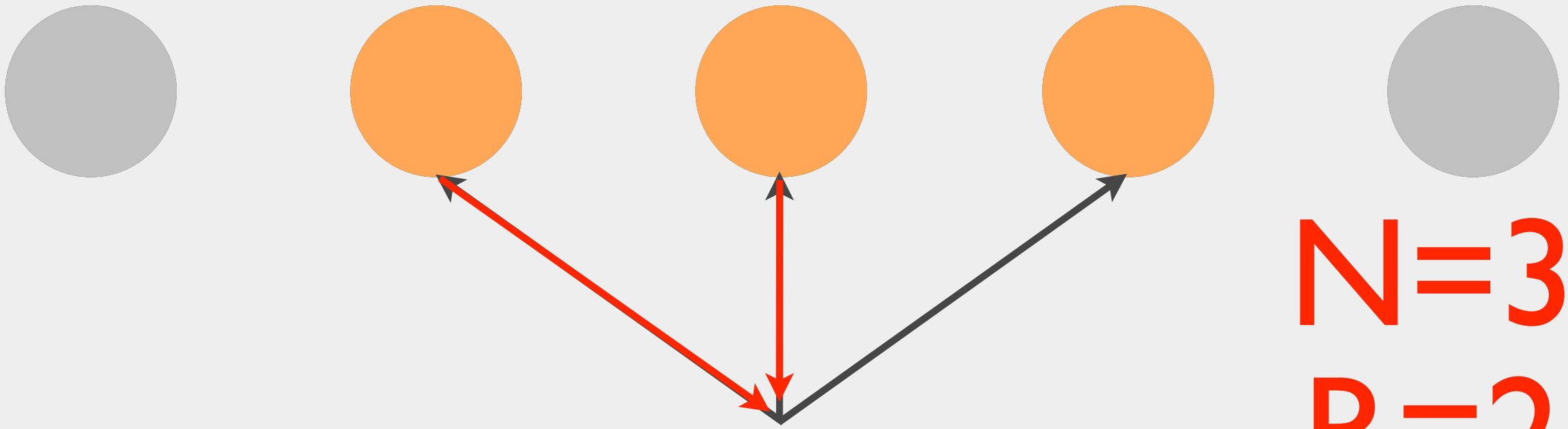
Adobe
Rackspace
IBM
Spotify
Twitter
Cisco
Netflix
Palantir
Cassandra
Reddit
Morningstar
Rhapsody
Digg
Mozilla
Shazam
Soundcloud
Gowalla

Yammer
Ask.com
Voldemort
Aol
Riak
Best Buy
GitHub
Comcast
LinkedIn
Gilt Groupe
JoyentCloud
Boeing

N replicas/key

read: wait for *R* replies

write: wait for *W* acks

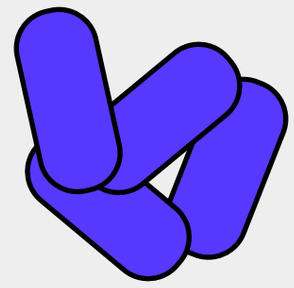


if: $R+W > N$

then: "strong" consistency

else: eventual consistency

“strong” consistency ≡ $R+W > N$



reads return the last acknowledged write or an in-flight write *(per-key)*

regular register

Latency

LinkedIn disk-based model

N=3

R

	99th	99.9th
1	1x	1x
2	1.59x	2.35x
3	4.8x	6.13x

W

	99th	99.9th
1	1x	1x
2	2.01x	1.9x
3	4.96x	14.96x

⇧ consistency, ⇧ latency

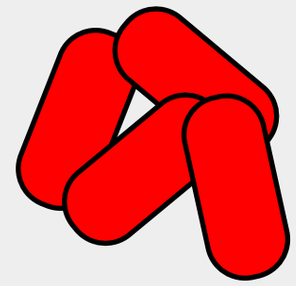
wait for more replicas,
read more recent data

⇩ consistency, ⇩ latency

wait for fewer replicas,
read less recent data

eventual consistency

$R+W \leq N$



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

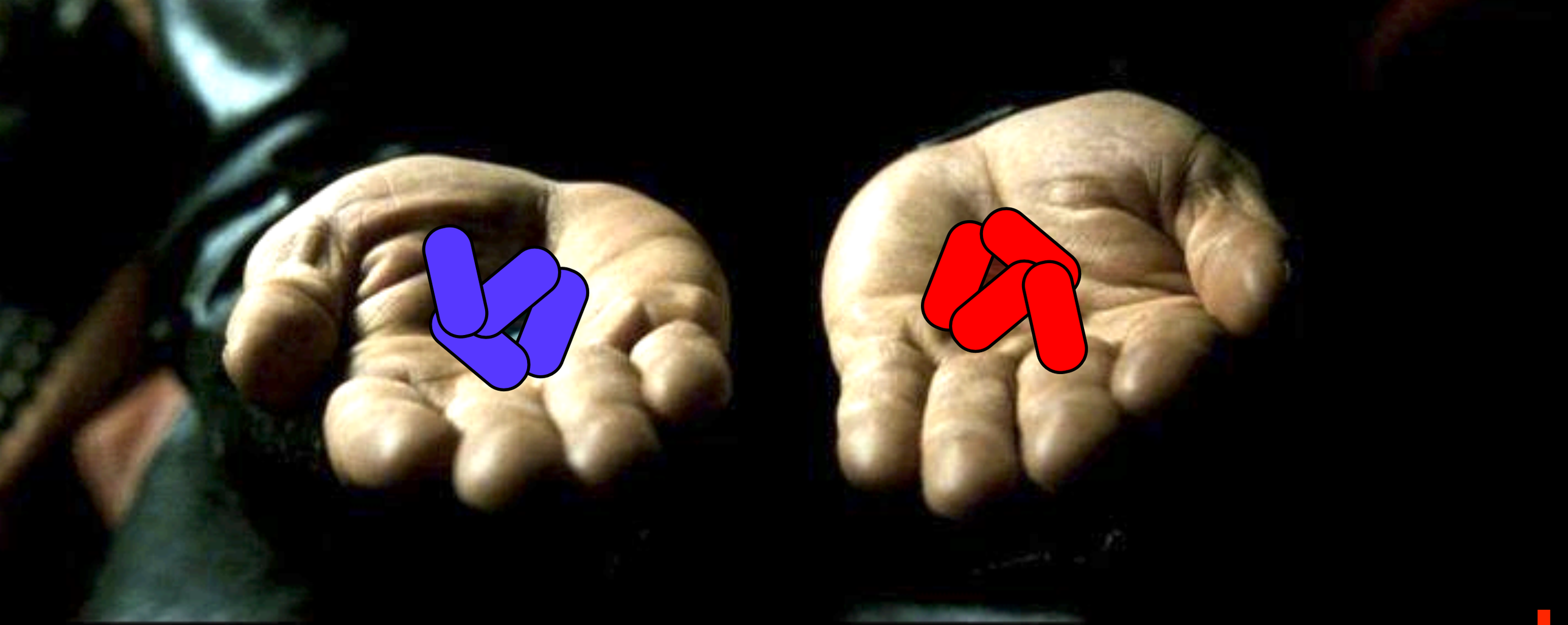
W. Vogels, *CACM* 2008

How eventual?

How long do I have to wait?

How consistent?

What happens if I don’t wait?



strong consistency

higher latency

eventual consistency

lower latency

intro

system model

practice

metrics

insights

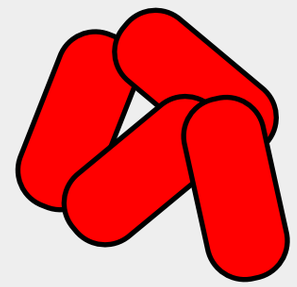
integration

Cassandra:

R=W=1, N=3

by default

(1+1 ≯ 3)



eventual consistency

in the wild

“maximum performance”

okay for “most data”

“very low latency”

“general case”

anecdotally, EC
"good enough" for
many kinds of data

How eventual?
How consistent?

"eventual and consistent enough"

Can we do better?

can't make *promises*

can give *expectations*

Probabilistically Bounded Staleness

intro

system model

practice

metrics

insights

integration

How eventual?

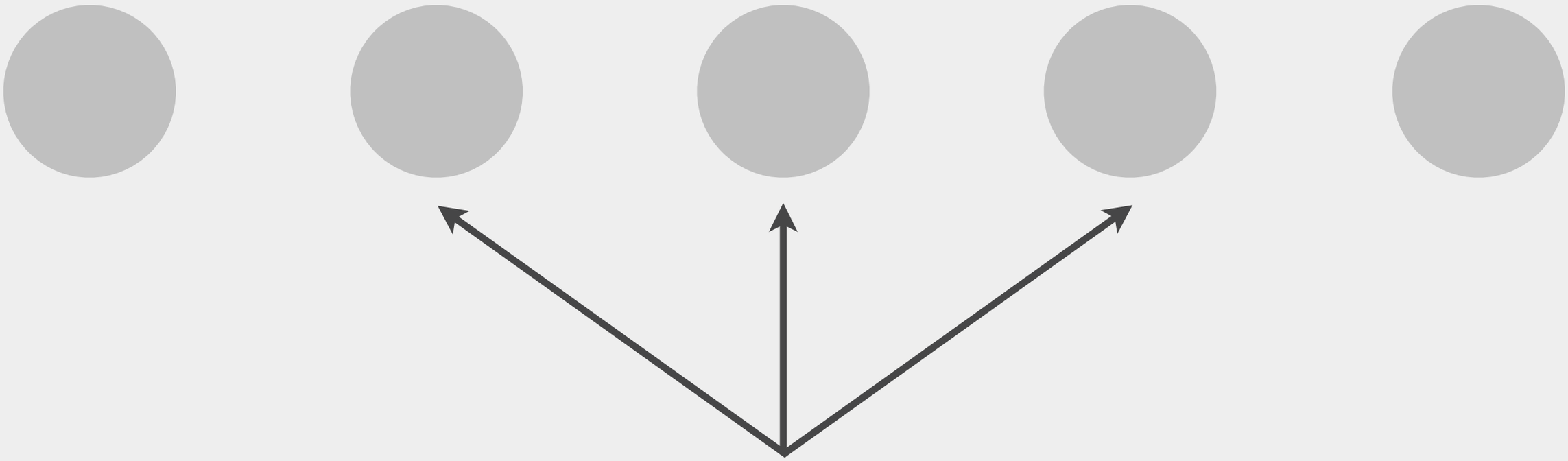
How long do I have to wait?

How eventual?

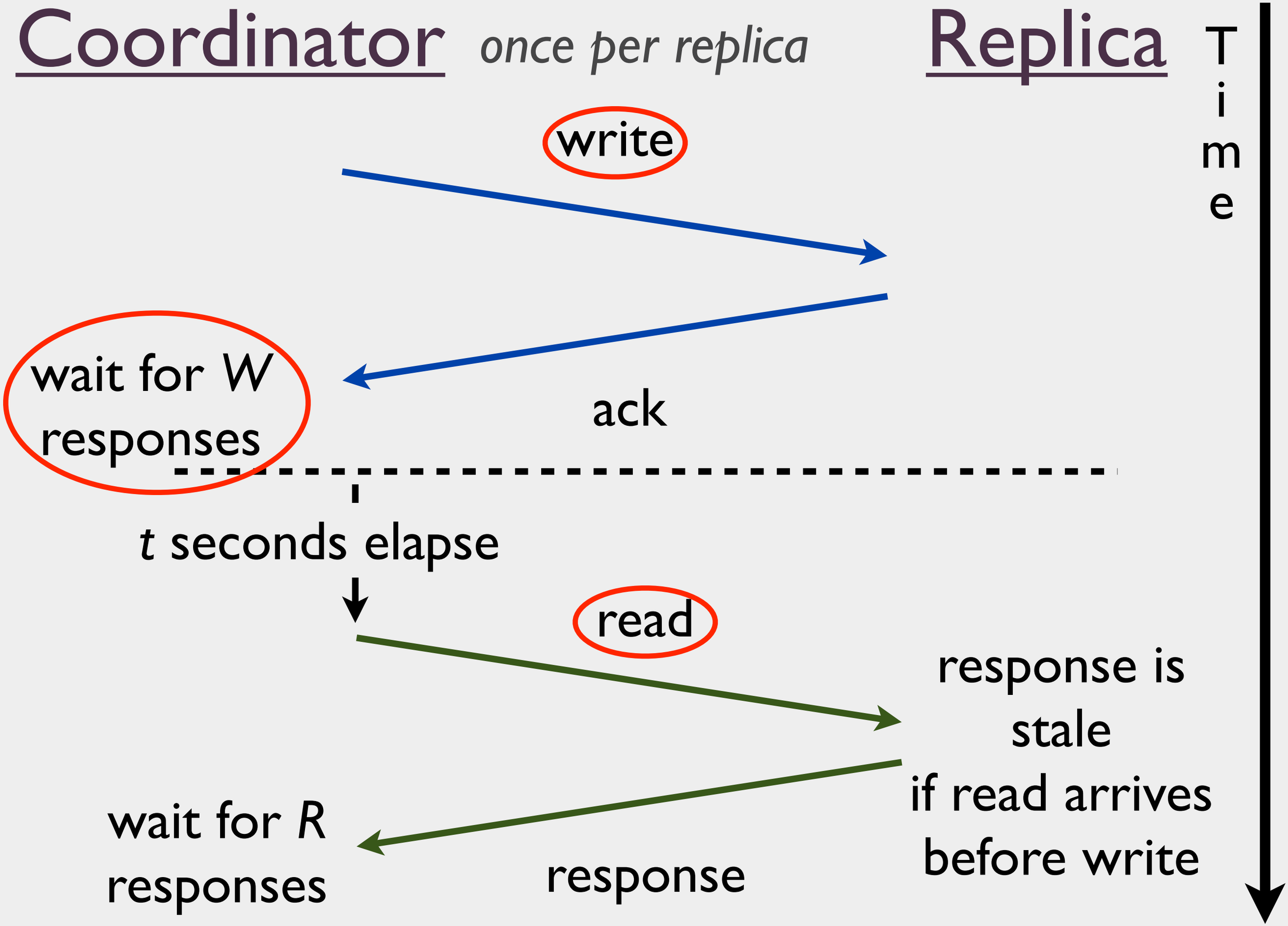
t-visibility: probability *p* of consistent reads after *t* seconds

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

t-visibility depends on



messaging and processing delays


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

Alice

write

R1

N=2

R2

Time

W=1

ack

Bob

read

R2

response

R=1

R2

inconsistent

Coordinator *once per replica* Replica

Time

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*:

order statistics dependent variables

instead:

Monte Carlo methods

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

WARS accuracy

real Cassandra cluster

varying latencies:

t-visibility RMSE: 0.28%

latency N-RMSE: 0.48%

How eventual?

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

key: *WARS* model

need: latencies

intro

system model

practice

metrics

insights

integration

LinkedIn

175M+ 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

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

N=3

LNKD-DISK

R=2, W=1, t =13.6 ms

99.9% consistent:

Latency: 12.53 ms

16.5%

faster

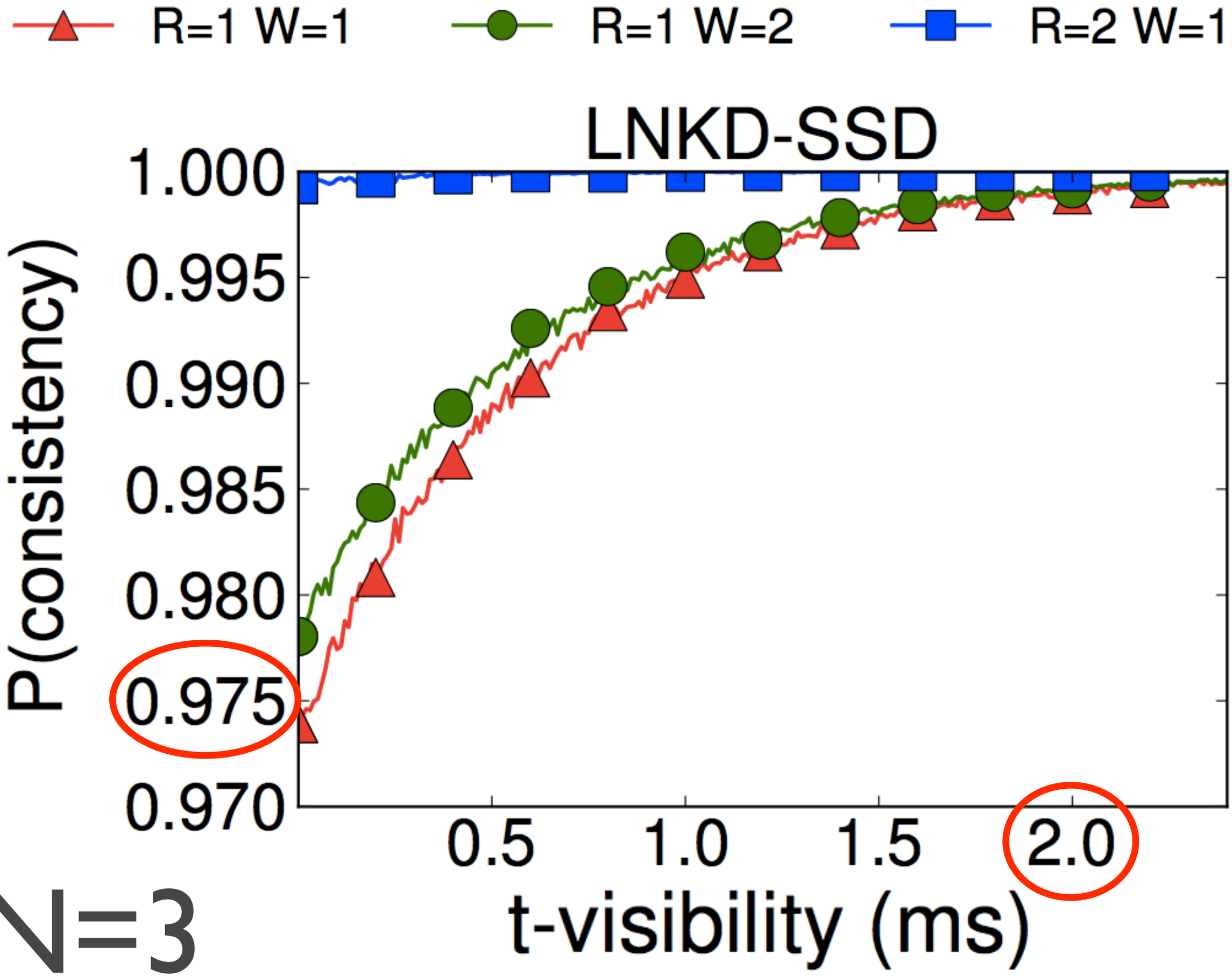
worthwhile?

R=3, W=1

100% consistent:

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
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

N=3

LNKD-SSD

R=1, W=1, t = 1.85 ms

99.9% consistent:

Latency: 1.32 ms

59.5%

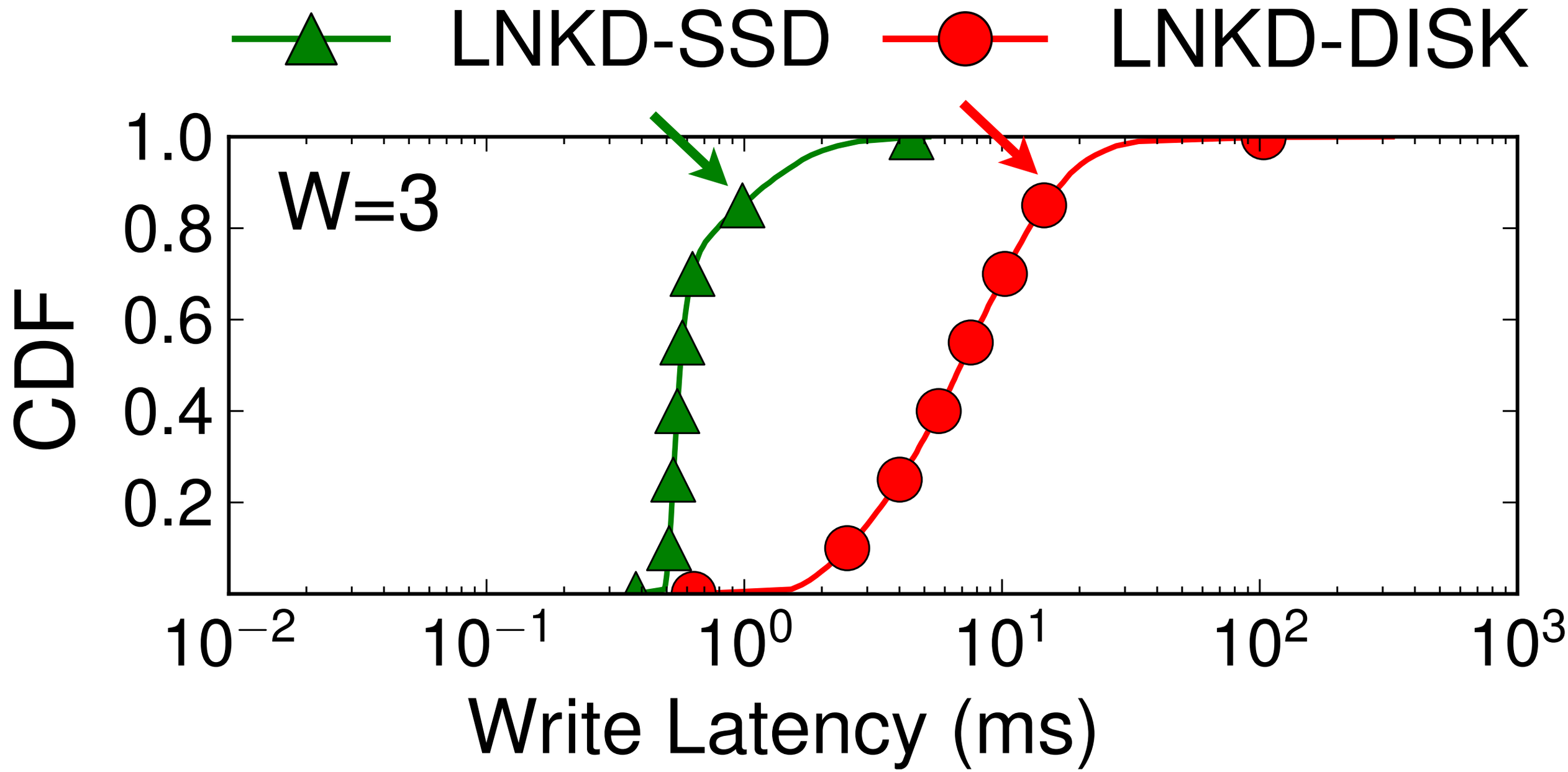
faster

R=3, W=1

100% consistent:

Latency: 4.20 ms

Latency is combined read and write latency at 99.9th percentile


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)

ack

wait for *W* responses

t seconds elapse

read

(R)

response is stale if read arrives before write

(S)

response

wait for *R* responses

N=3

Yammer

latency

⇩81.1%
(187ms)

99.9th
t-visibility
202 ms

in the paper

How consistent?

k-staleness (versions)

monotonic reads

quorum load

in the paper

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

versions and time

in the paper

latency distributions

WAN model

varying quorum sizes

staleness detection

intro

system model

practice

metrics

insights

integration

Integration

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

Cassandra

Project Voldemort

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?

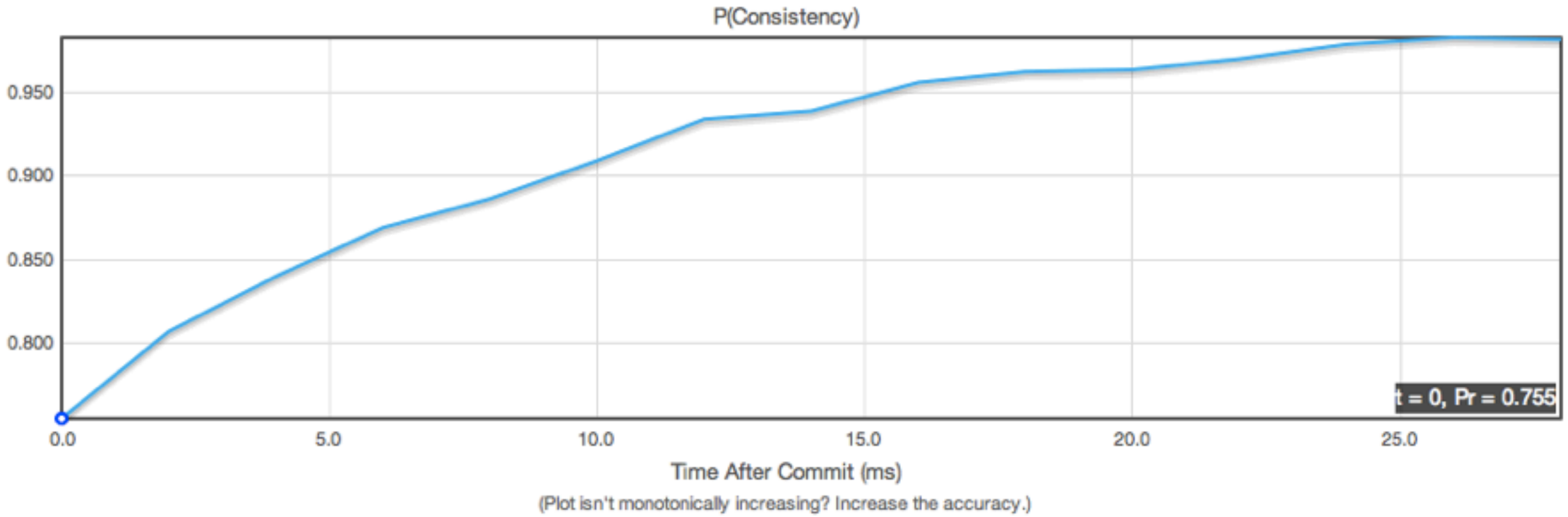
This patch provides a latency-consistency analysis within `nodetool`. Users can accurately predict Cassandra's behavior in their production environments without interfering with performance.

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: Probabilistically Bounded Staleness About Instructions Demo Questions More Info Thanks

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



(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

http://pbs.cs.berkeley.edu/#demo

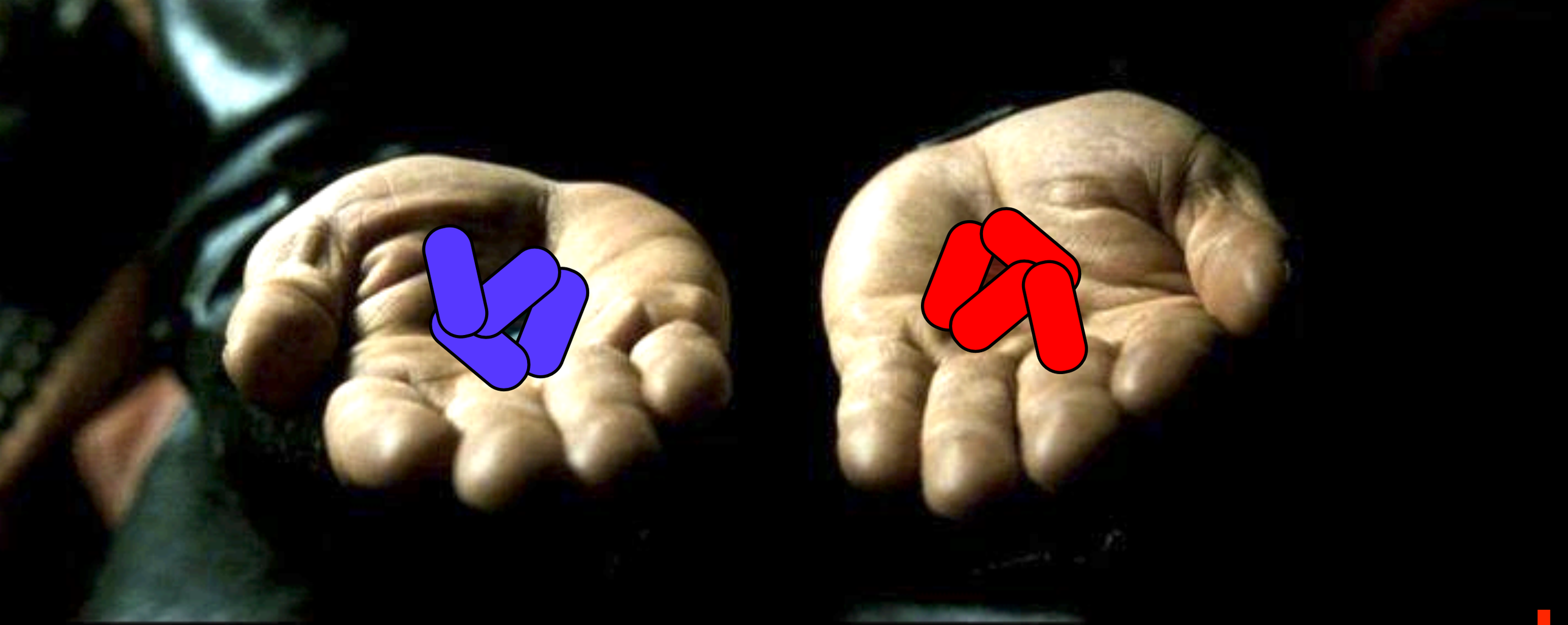
Related Work

Quorum Systems
- probabilistic quorums [PODC '97]
- deterministic k-quorums [DISC '05, '06]

Consistency Verification
- Golab et al. [PODC '11]
- Bermbach and Tai [M4WSOC '11]
- Wada et al. [CIDR '11]
- Anderson et al. [HotDep '10]
- Transactional consistency:
  Zellag and Kemme [ICDE '11],
  Fekete et al. [VLDB '09]

Bounded Staleness Guarantees
- TACT [OSDI '00]
- FRACS [ICDCS '03]
- AQuA [IEEE TPDS '03]

Latency-Consistency
- Daniel Abadi [Computer '12]
- Kraska et al. [VLDB '09]



strong consistency

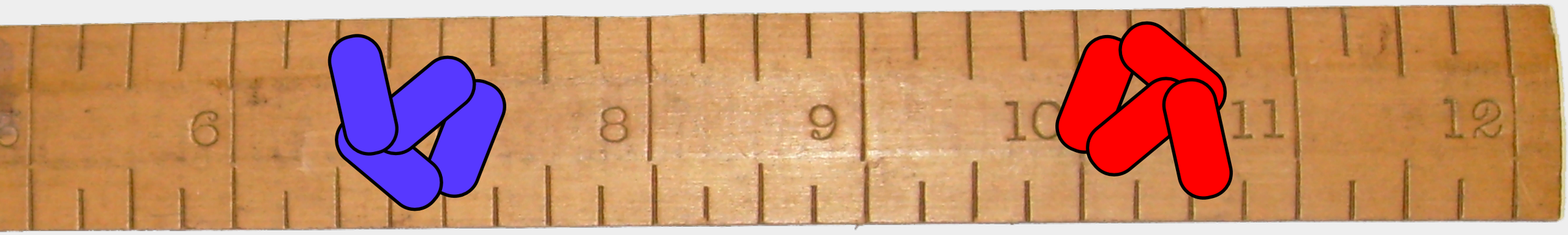
higher latency

eventual consistency

lower latency

consistency

is a continuum



strong eventual

PBS

quantify eventual consistency

model staleness in time, versions

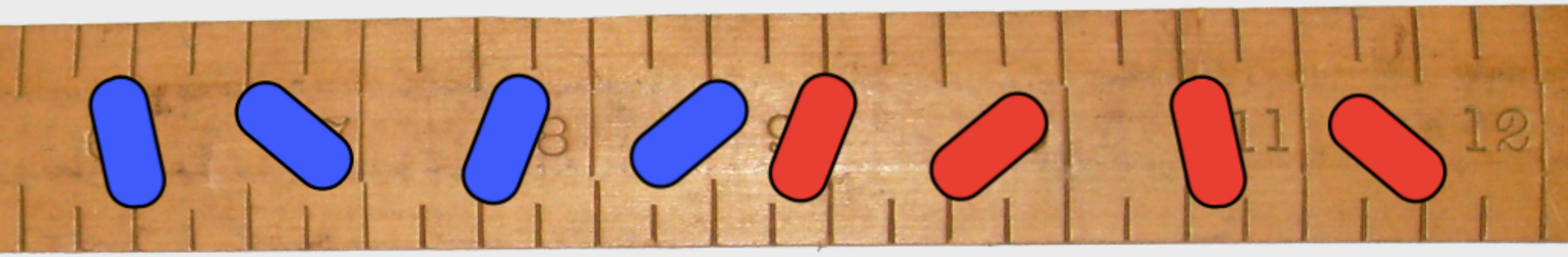
latency-consistency trade-offs

analyze real systems and hardware

pbs.cs.berkeley.edu

quantify **which** choice is best and explain

why EC is often strongly consistent



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)

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

(for a given key)

Failure?

Treat failures as

latency

spiKes

How long do partitions last?

what time interval?

99.9% uptime/yr

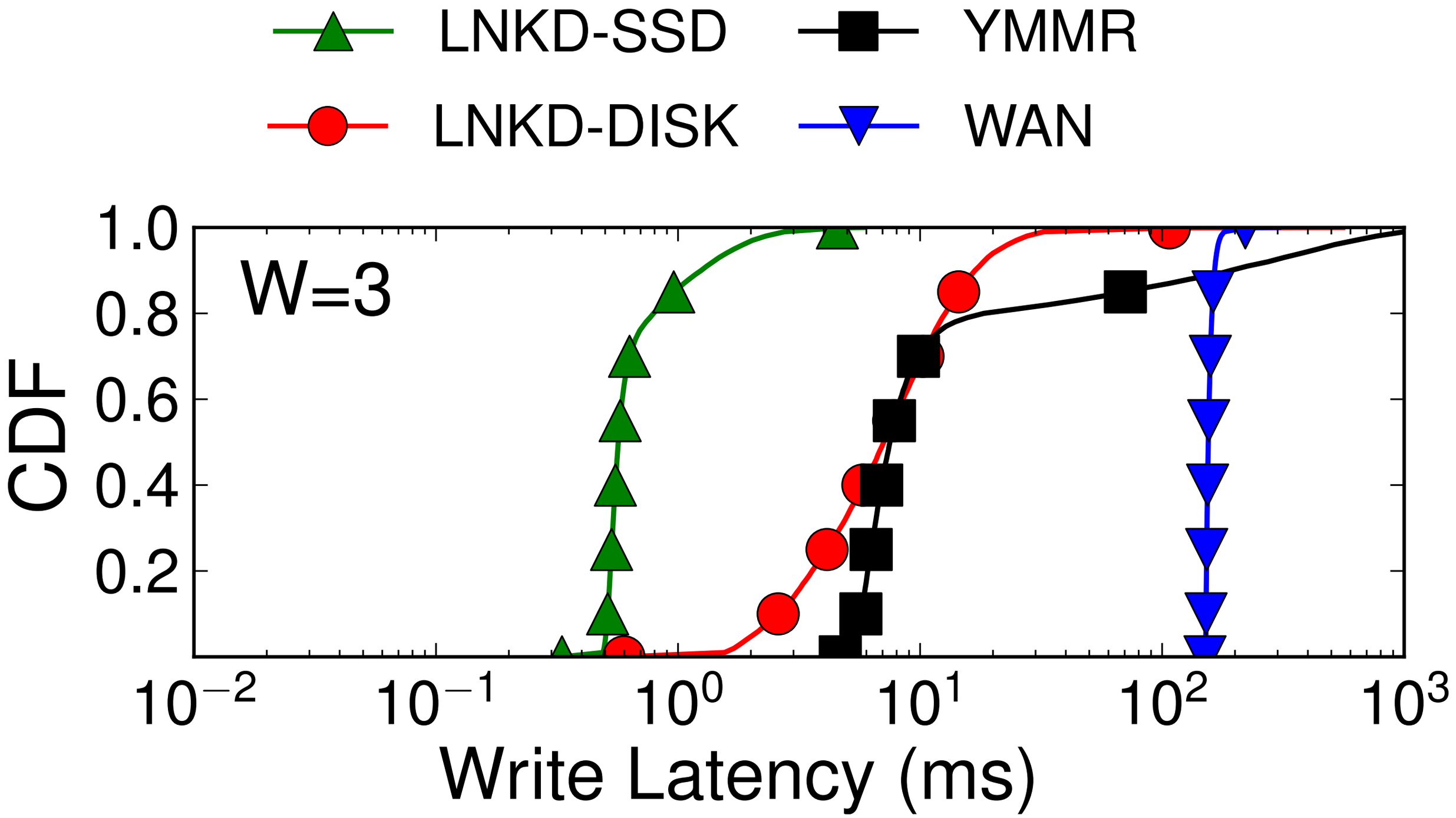
⇒ 8.76 hours downtime/yr

8.76 consecutive hours down

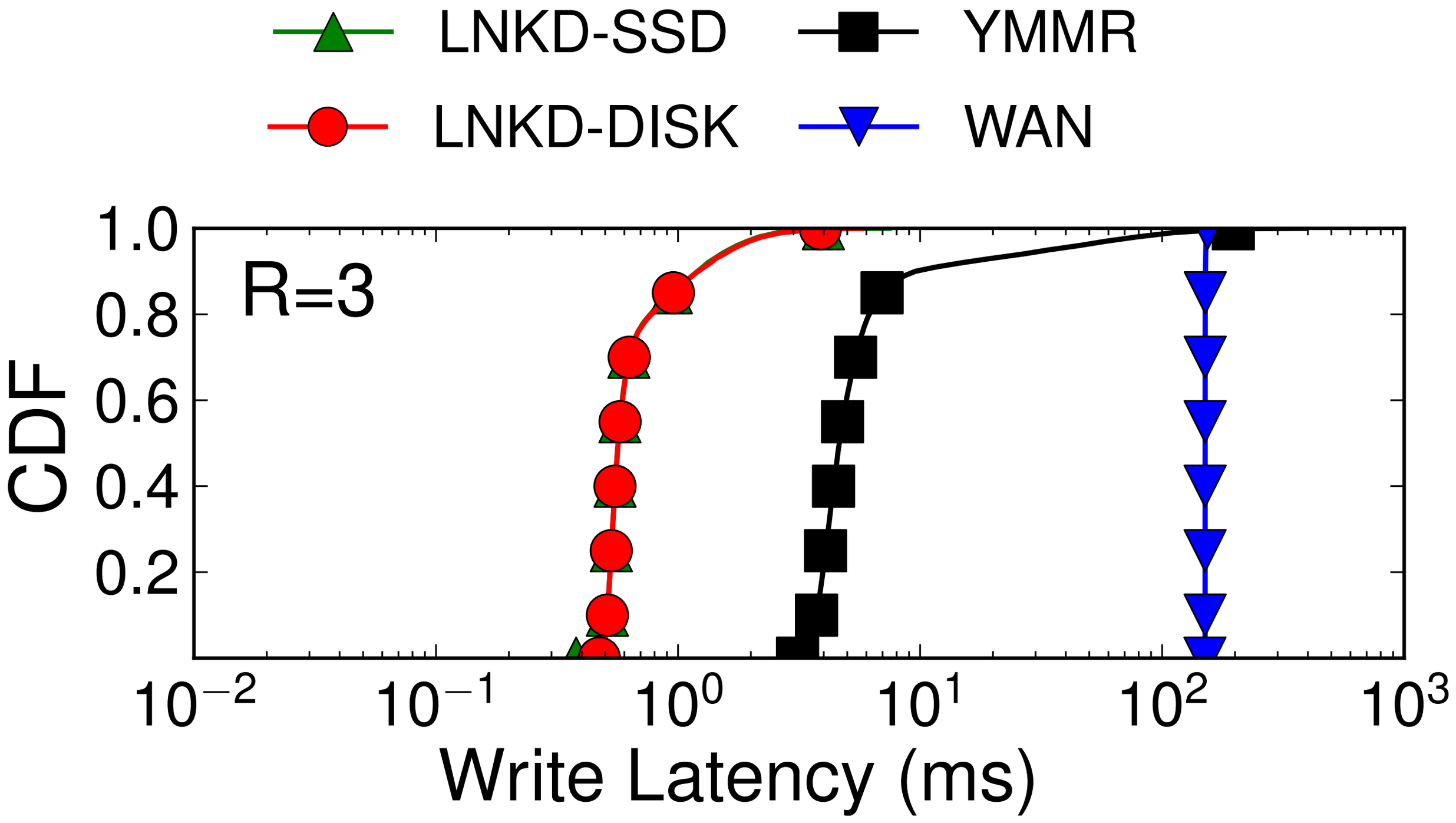
⇒ bad 8-hour rolling average

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



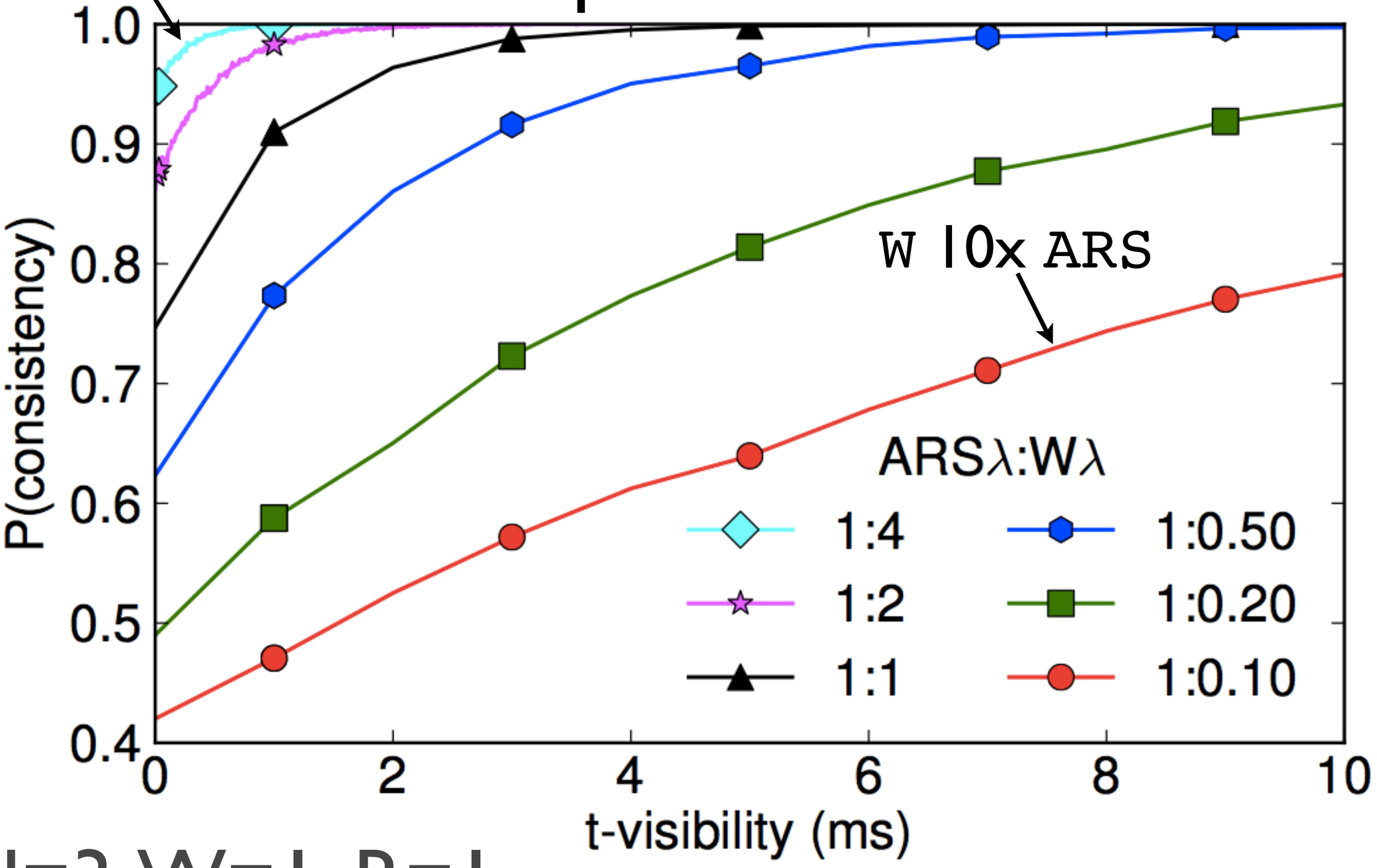
N=3

(LNKD-SSD and LNKD-DISK identical for reads)

<*k*,*t*>-staleness: versions and time

approximation:
exponentiate
t-staleness by *k*

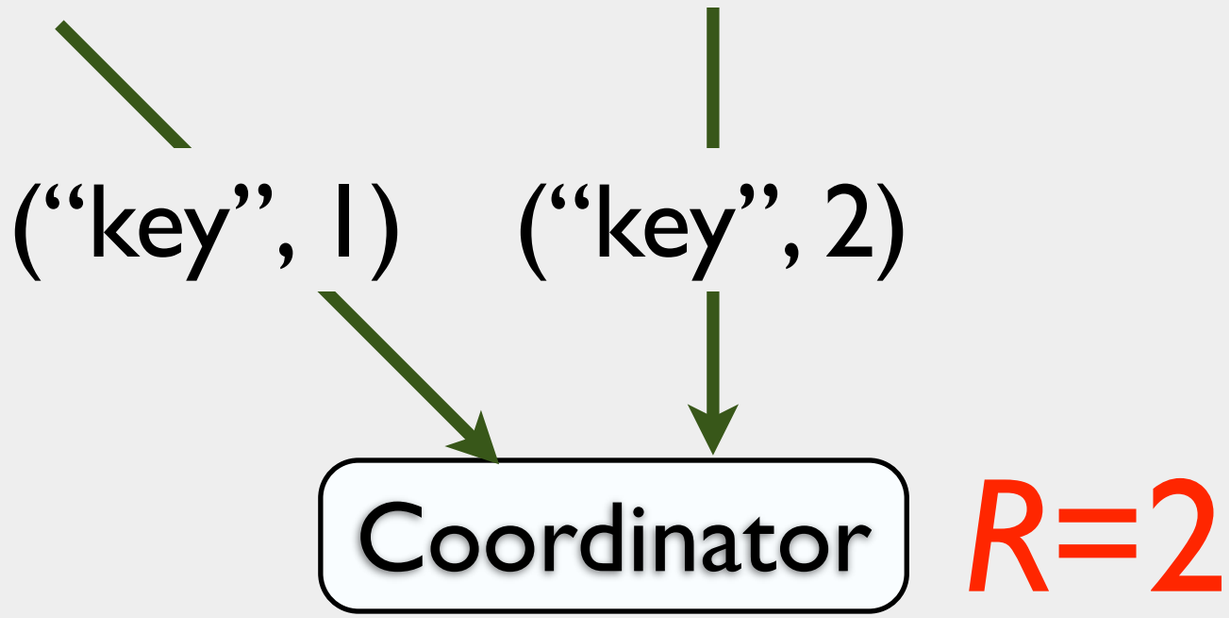
Synthetic, Exponential Distributions



N=3, W=1, R=1

concurrent writes:

deterministically choose



%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%

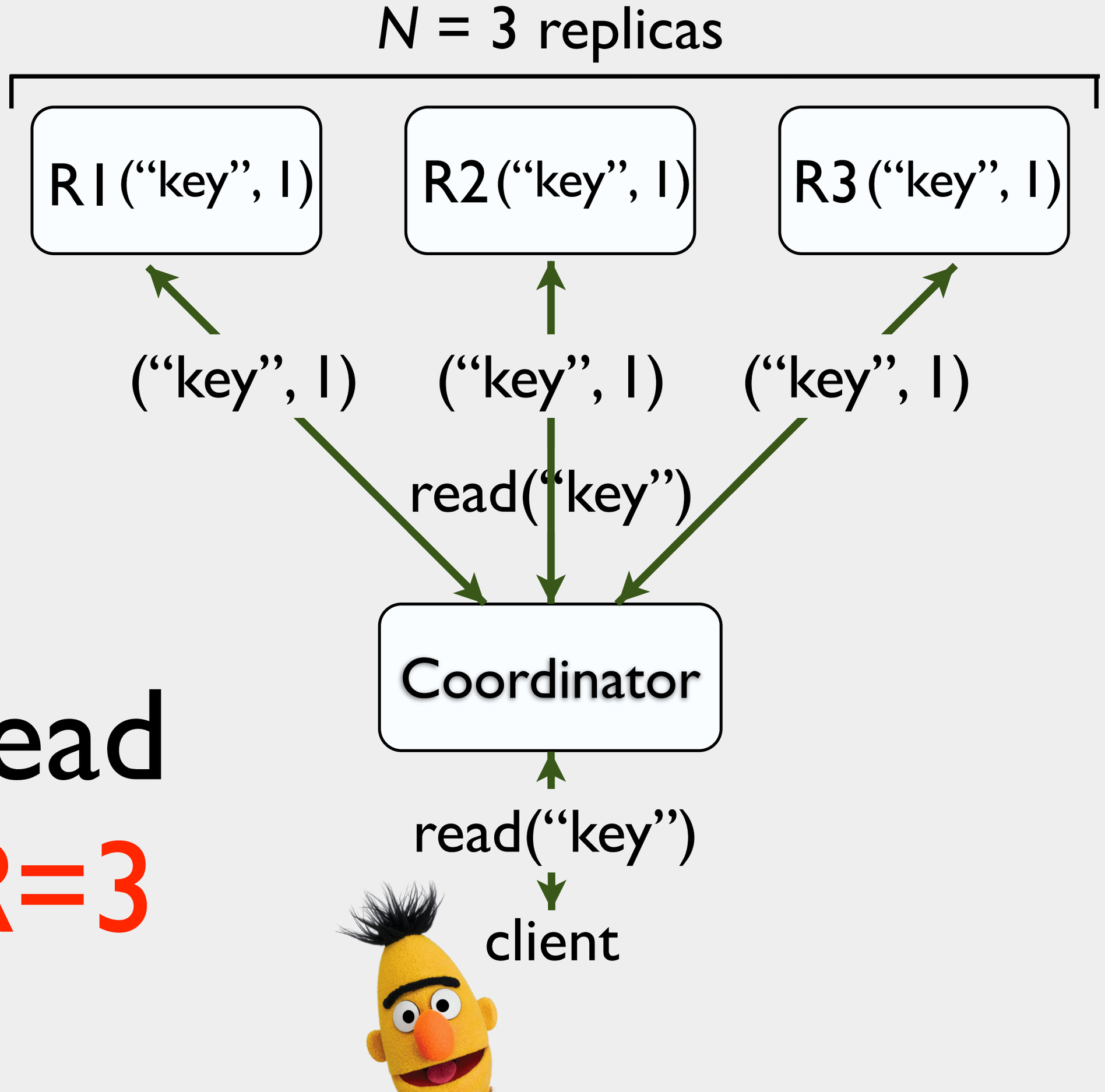
Table 3: Distribution fits for production latency distributions from LinkedIn (LNKD-*) and Yammer (YMMR).

	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

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

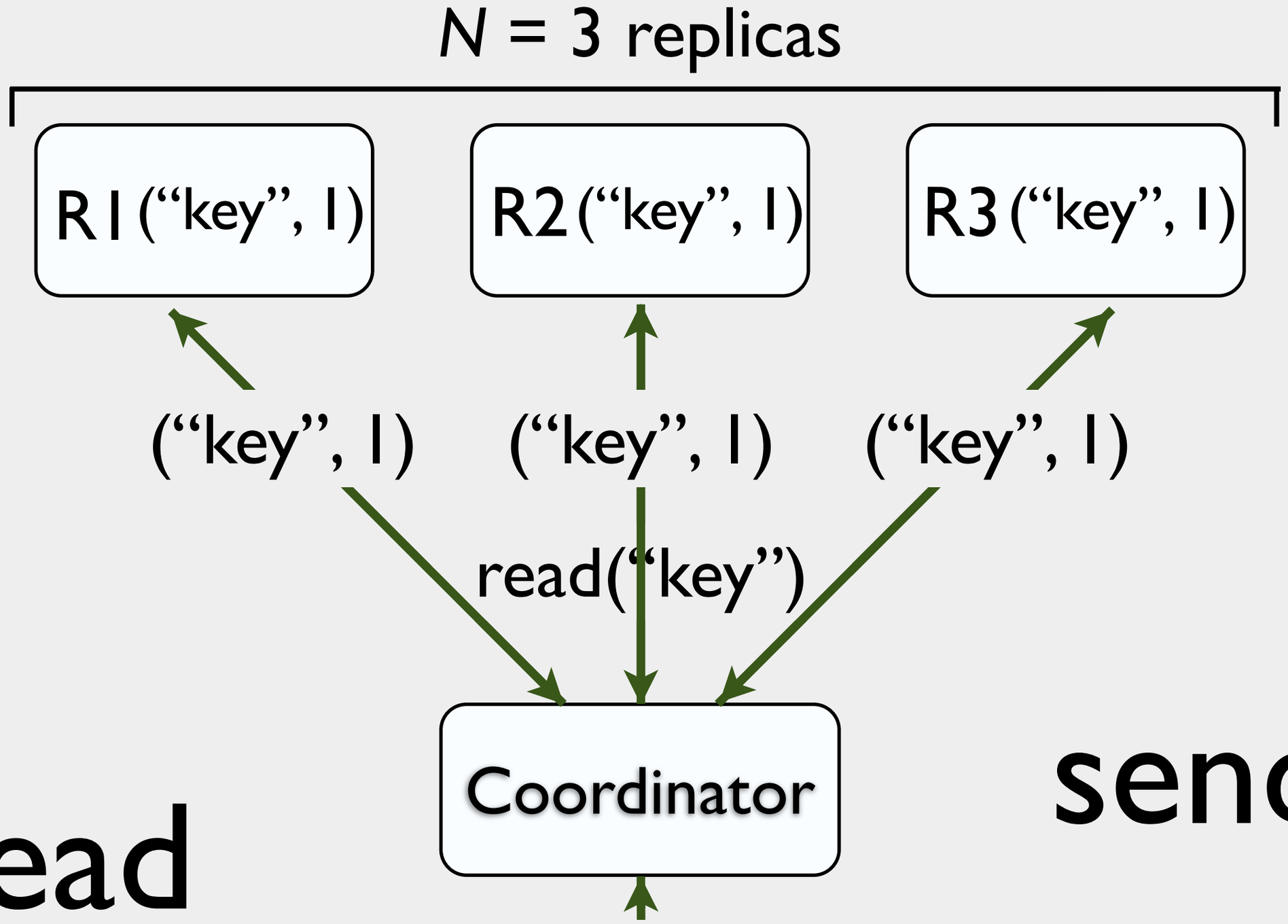
read

R=3


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

read

R=3



read

R=1

send

read

to all

R1 (“key”, 1)

R2 (“key”, 1)

R3 (“key”, 1)

ack(“key”, 2)

(“key”, 2)

(“key”, 2) “key”, 1)

write(“key”, 2)

read(“key”)

Coordinator

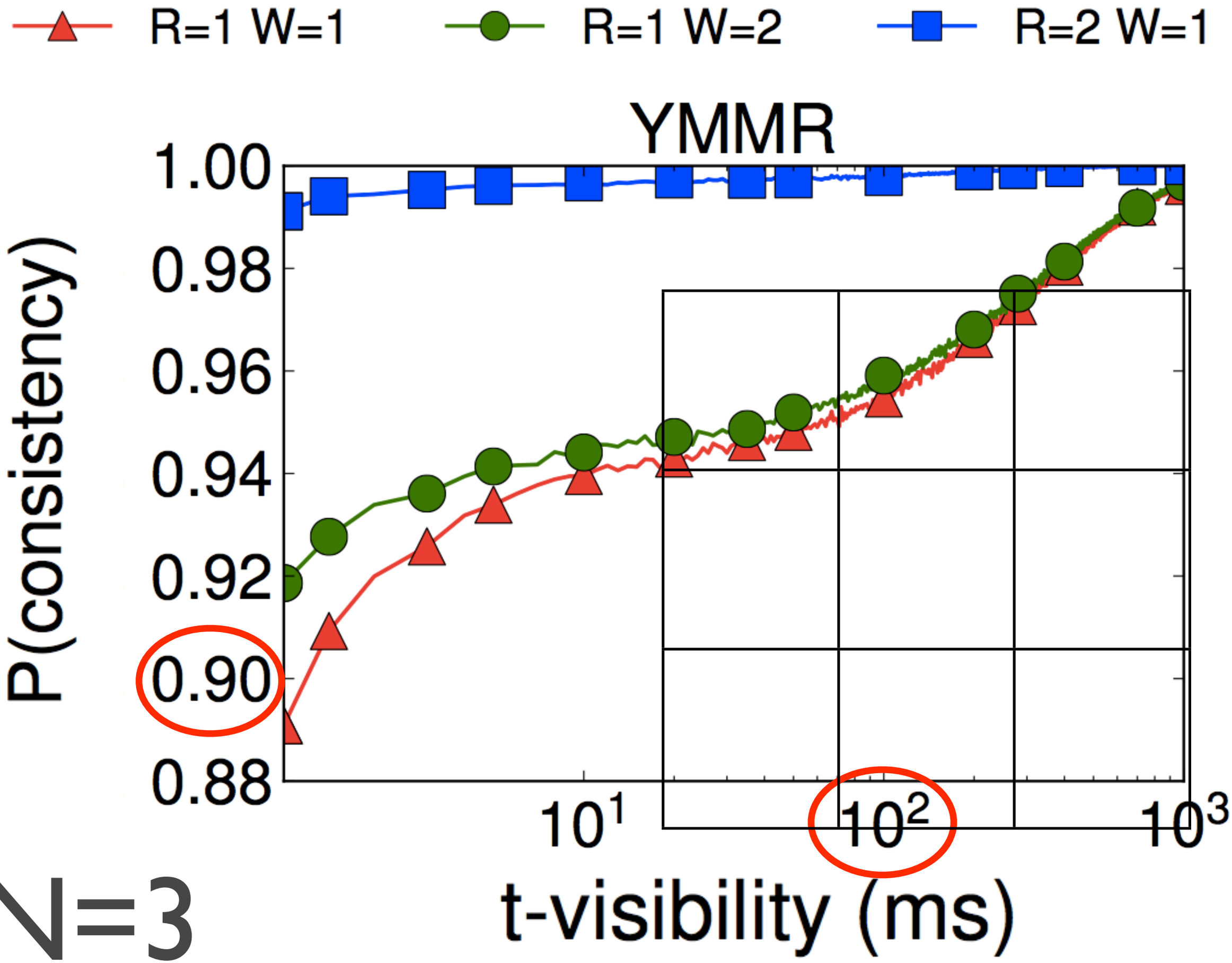
W=1

Coordinator

R=1

ack(“key”, 2)

(“key”, 1)


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