SwiftPaxos: Fast Geo-Replicated State Machines

Fedor Ryabinin[1][2], Alexey Gotsman[1], Pierre Sutra[3]

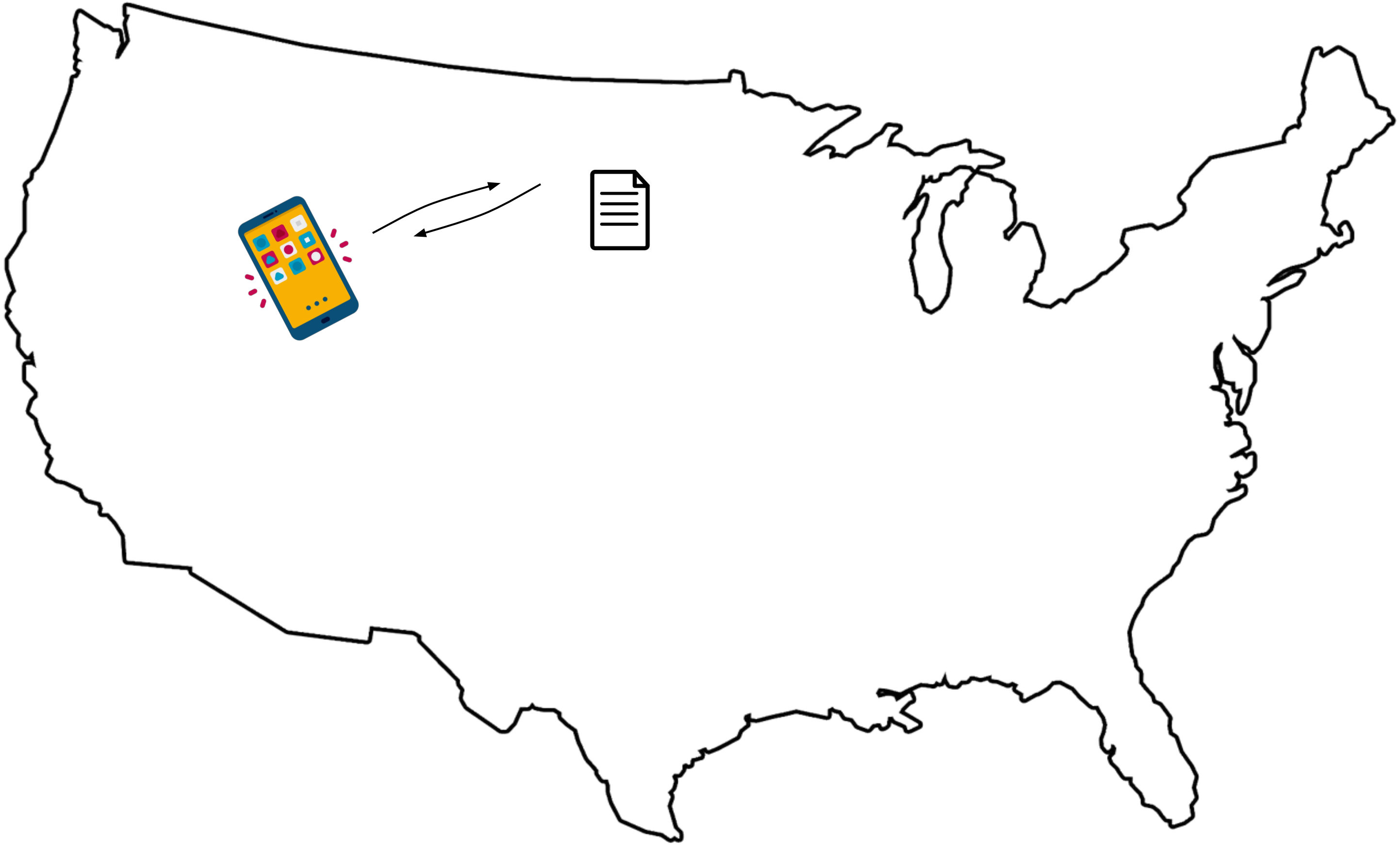
[1] *IMDEA Software*

[2] *Universidad Politécnica de Madrid*

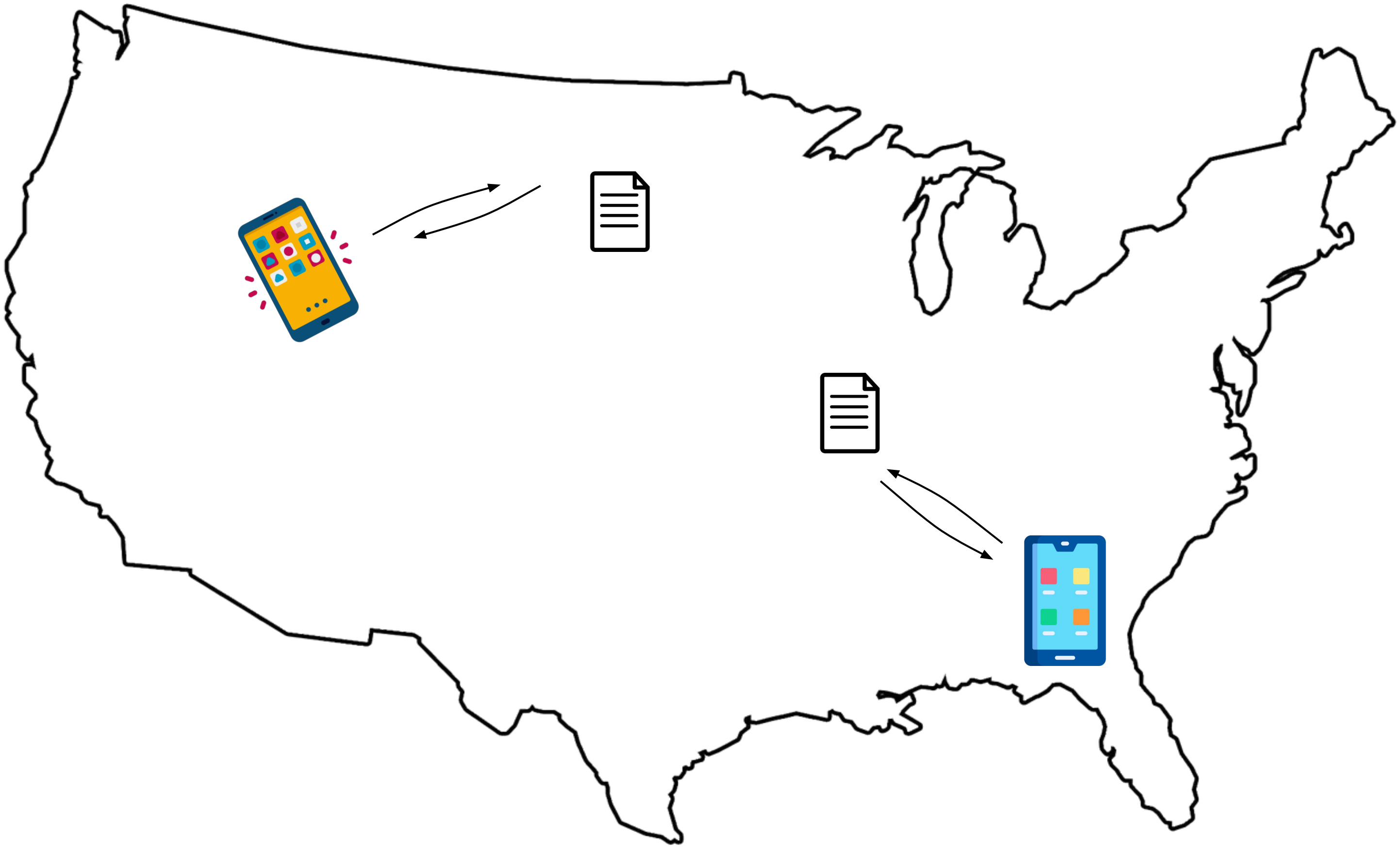
[3] *Telecom SudParis & INRIA*

nsdi '24

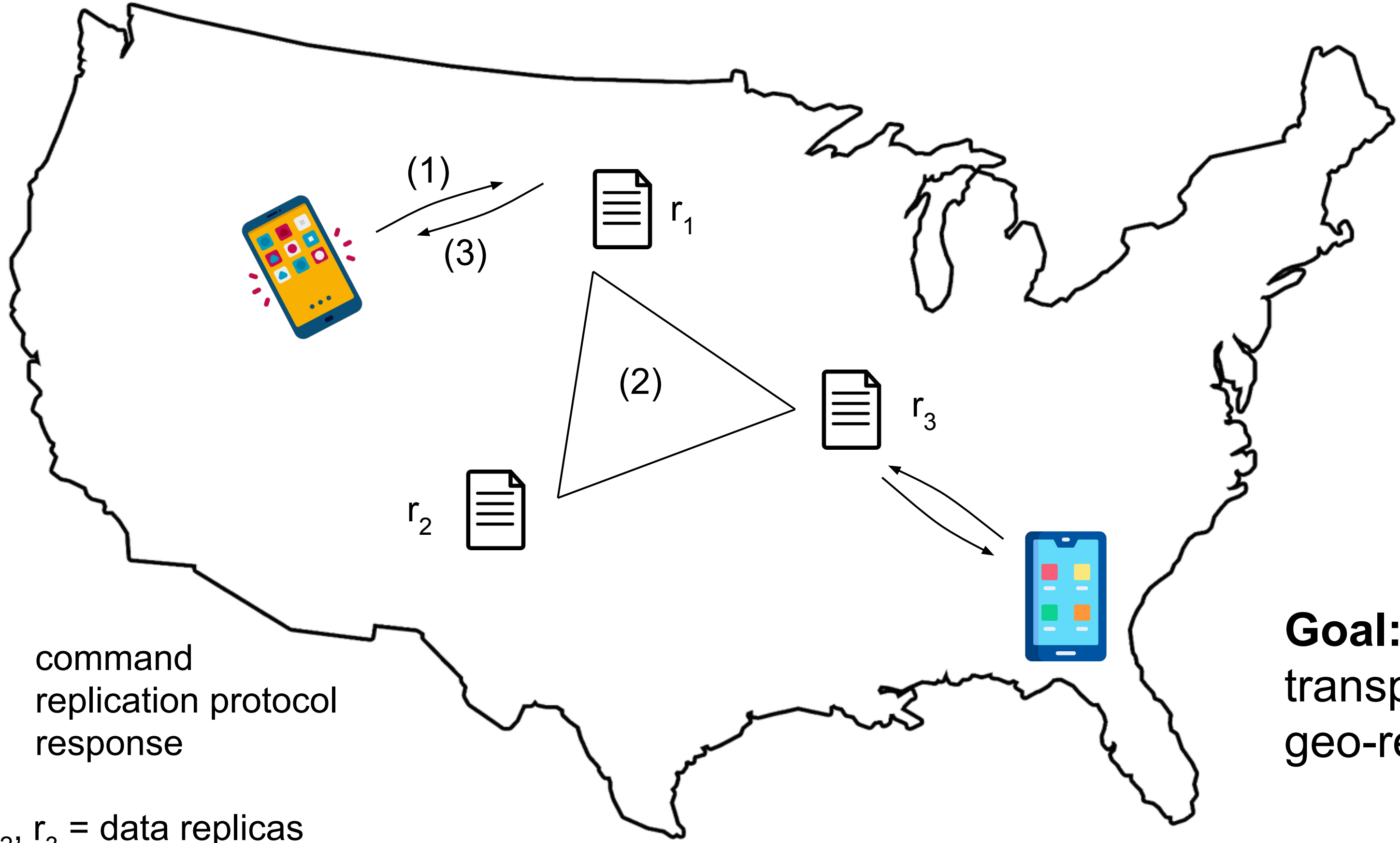
Geo-replicated data



Geo-replicated data



Geo-replicated data

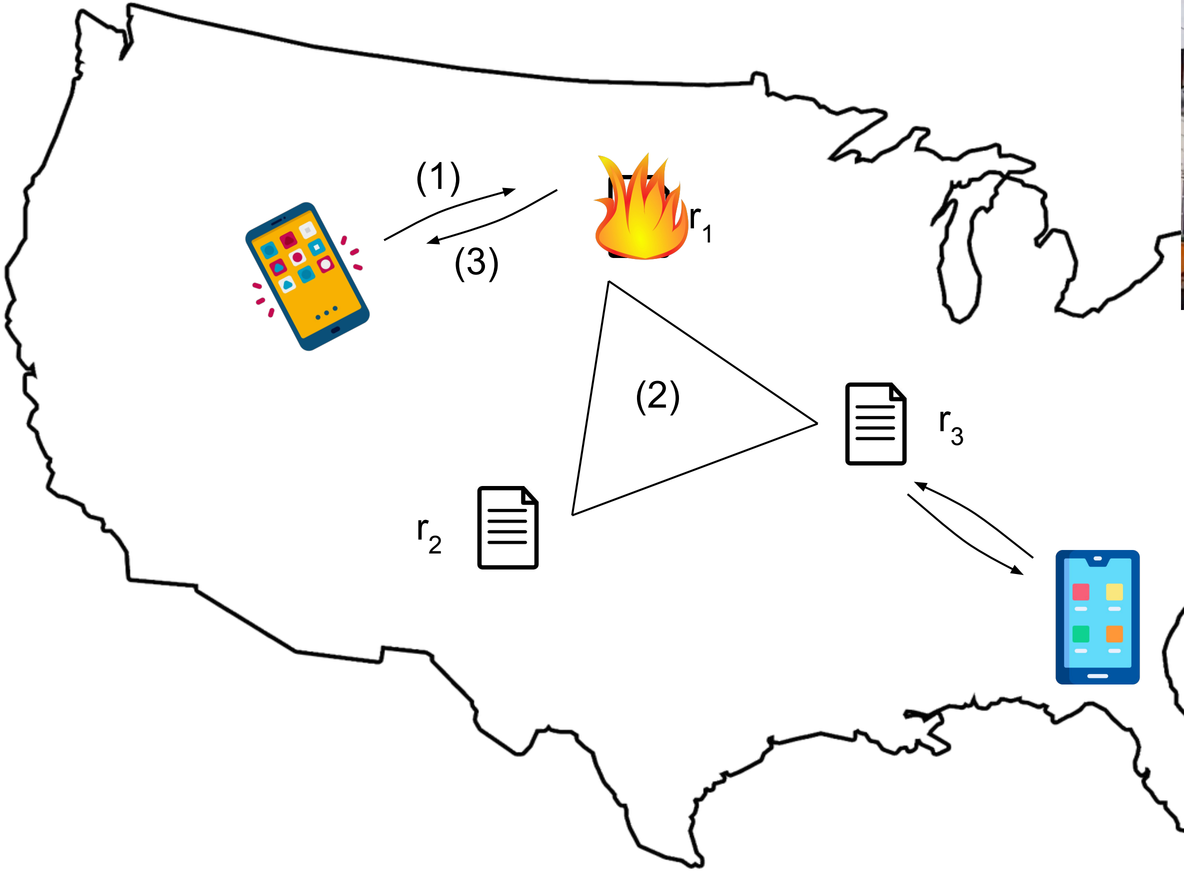


(1) command
(2) replication protocol
(3) response

r_1, r_2, r_3 = data replicas

Goal:
transparent efficient geo-replication

Geo-replicated data



Goal:
transparent efficient and *robust* geo-replication

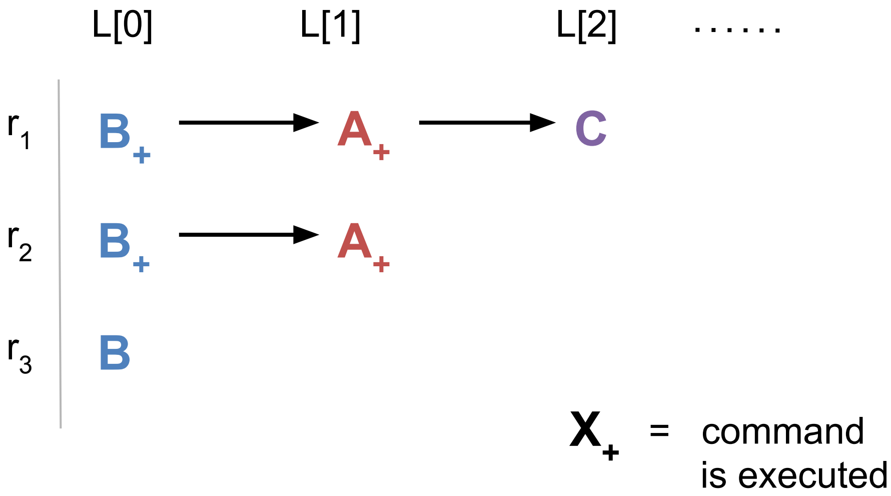
State-machine Replication

Each replica holds a log L

Execute commands in log order

To decide a command at position L[i]

- run i-th consensus



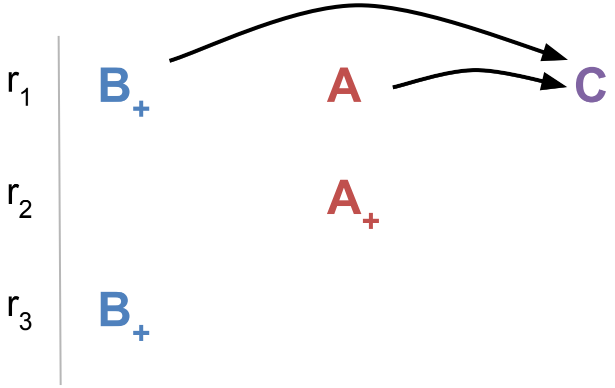
APACHE ZooKeeper™

Cloud Spanner

Apache CASSANDRA

SwiftPaxos

Execute *conflicting* commands in the same order



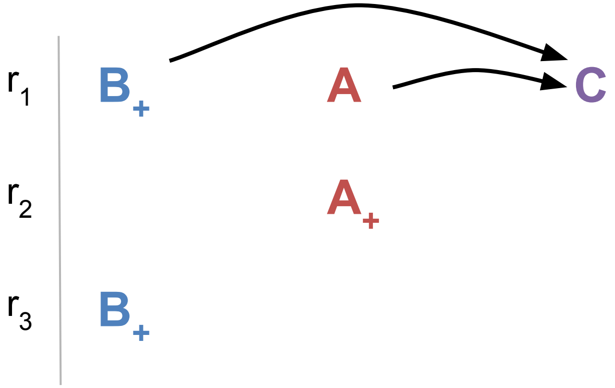
A = x ← 42

B = y ← 7

C = z ← x + y

SwiftPaxos

Execute *conflicting* commands in the same order



How:

- For each command, compute its *dependencies.*
- Execute commands wrt. dependencies.

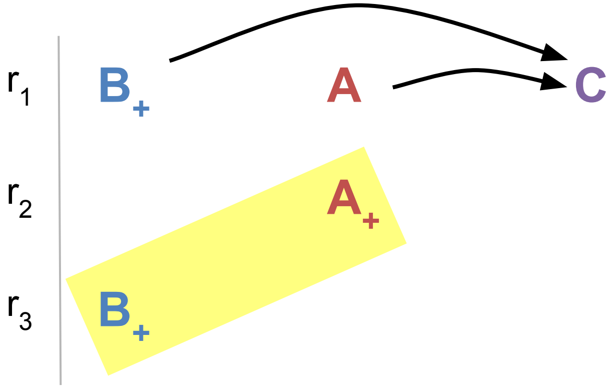
A = $x \leftarrow 42$

B = $y \leftarrow 7$

C = $z \leftarrow x + y$

SwiftPaxos

Execute *conflicting* commands in the same order



In this example,

- A can execute before or after B.

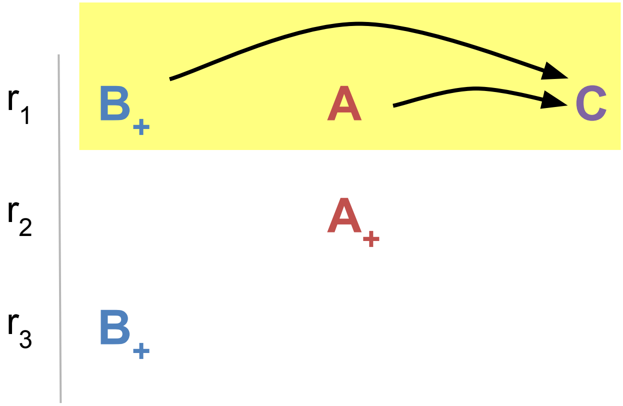
A = x ← 42

B = y ← 7

C = z ← x + y

SwiftPaxos

Execute *conflicting* commands in the same order



In this example,

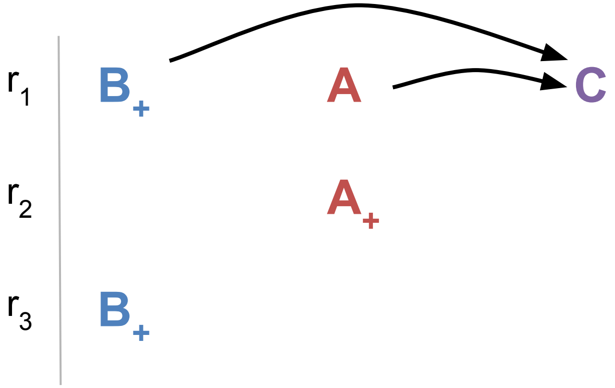
- A can execute before or after B.
- C depends on both A and B.

A = x ← 42

B = y ← 7

C = z ← x + y

Execute *conflicting* commands in the same order



Invariants:

- At each replica, dependencies are acyclic.

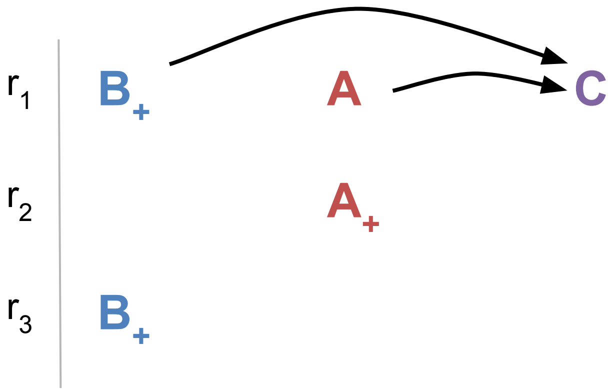
A = $x \leftarrow 42$

B = $y \leftarrow 7$

C = $z \leftarrow x + y$

SwiftPaxos

Execute *conflicting* commands in the same order



Invariants:

- At each replica, dependencies are acyclic.
- Replicas agree on dependencies.

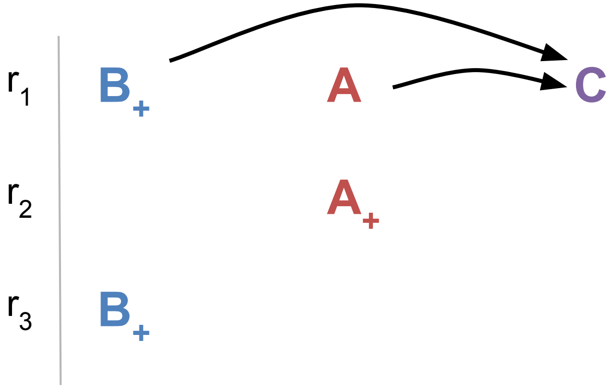
A = x ← 42

B = y ← 7

C = z ← x + y

SwiftPaxos

Execute *conflicting* commands in the same order



A = $x \leftarrow 42$

B = $y \leftarrow 7$

C = $z \leftarrow x + y$

Invariants:

- At each replica, dependencies are acyclic.
- Replicas agree on dependencies.
- For two conflicting commands **X** and **Y**, either **X** is a dependency of **Y**, or the converse is true.

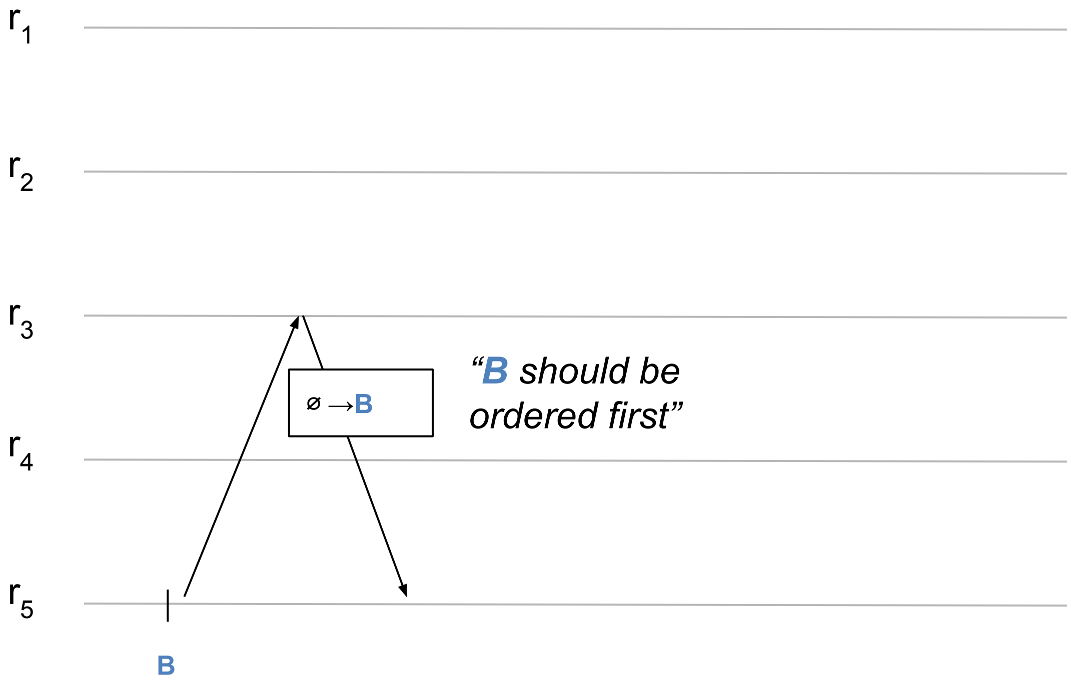
Summary

- A *new* strongly-consistent replication protocol
- Maintains *at least* Paxos latency
- Executes commands in *optimal time*:
  - 1 RTT when no contention (conflicts already solved by the network)
  - 1.5 RTT otherwise

Key novelty

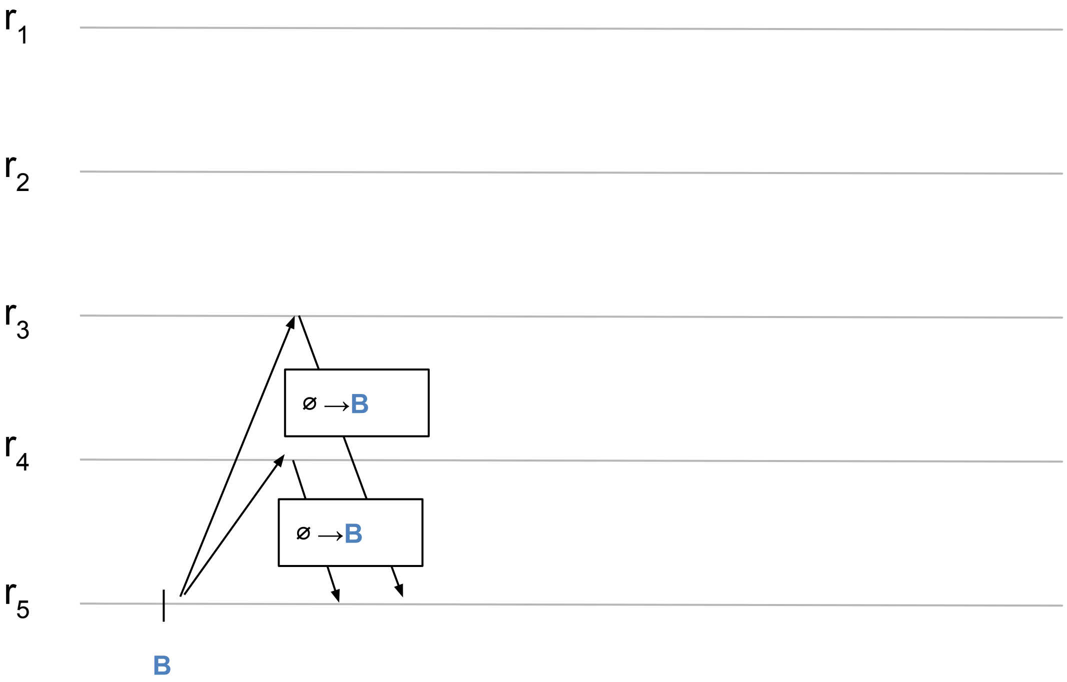
(double-voting) in consensus, a replica can vote *twice*, once for its own proposal then for the leader's.

SwiftPaxos / *fast path*

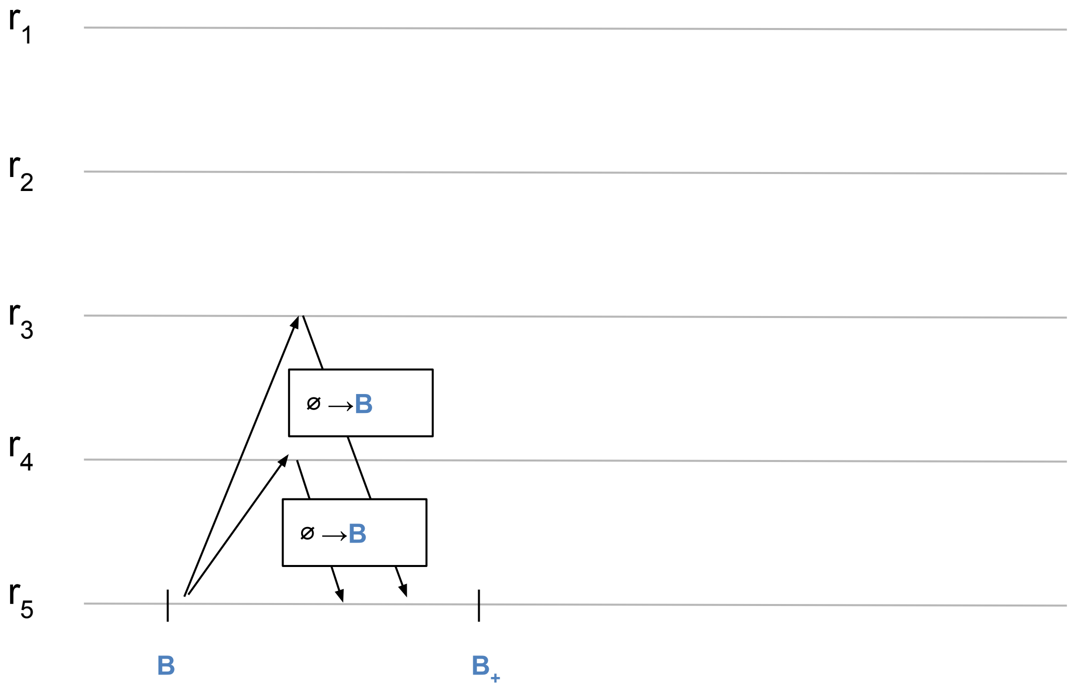


*“**B** should be ordered first”*

SwiftPaxos / *fast path*

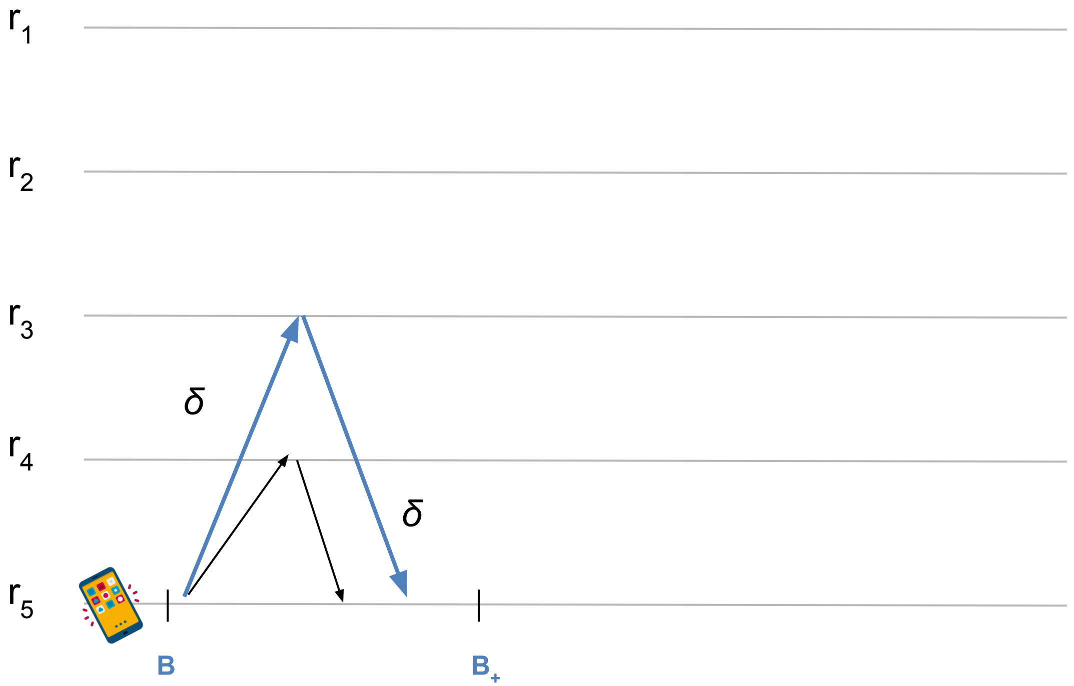


SwiftPaxos / *fast path*



$\mathbf{X_+}$ = command is executed

SwiftPaxos / *best-case latency (2δ)*

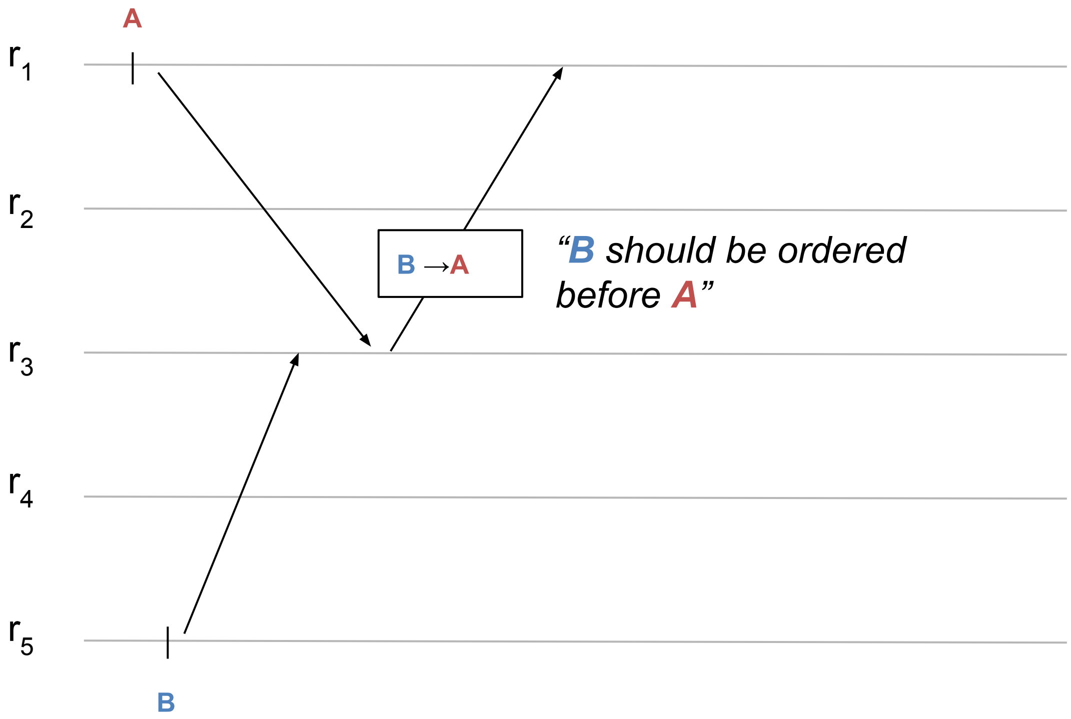


Features

- Optimal best-case latency

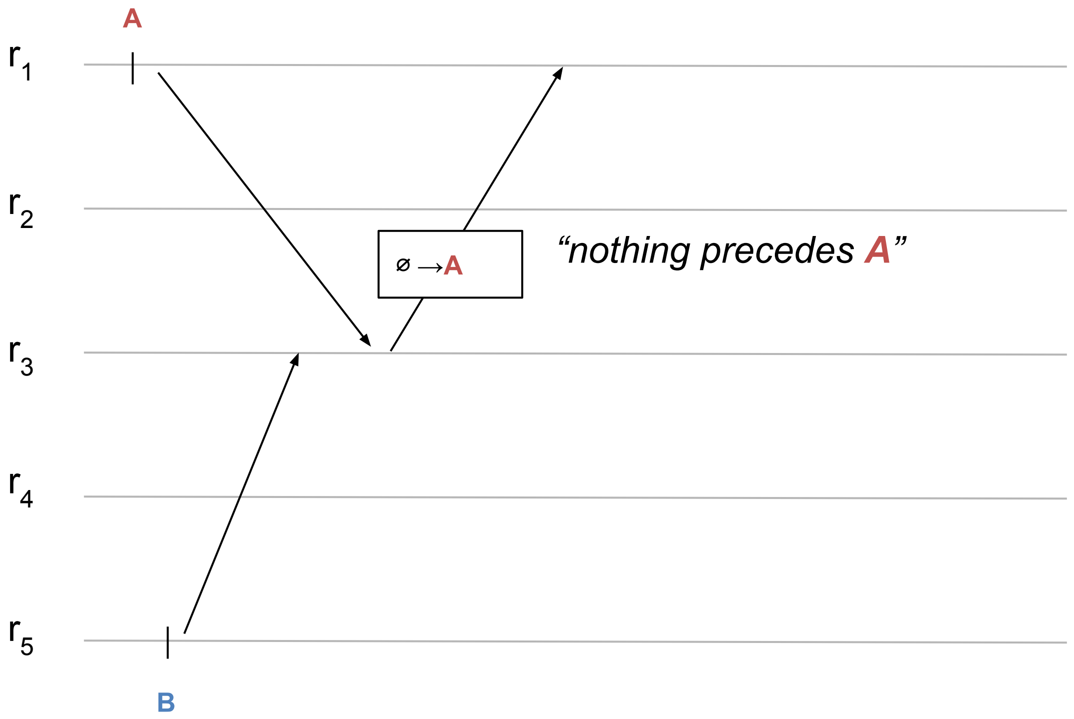
X_+ = command is executed

SwiftPaxos / *dependencies tracking*



*"**B** should be ordered before **A**"*

SwiftPaxos / *dependencies tracking*

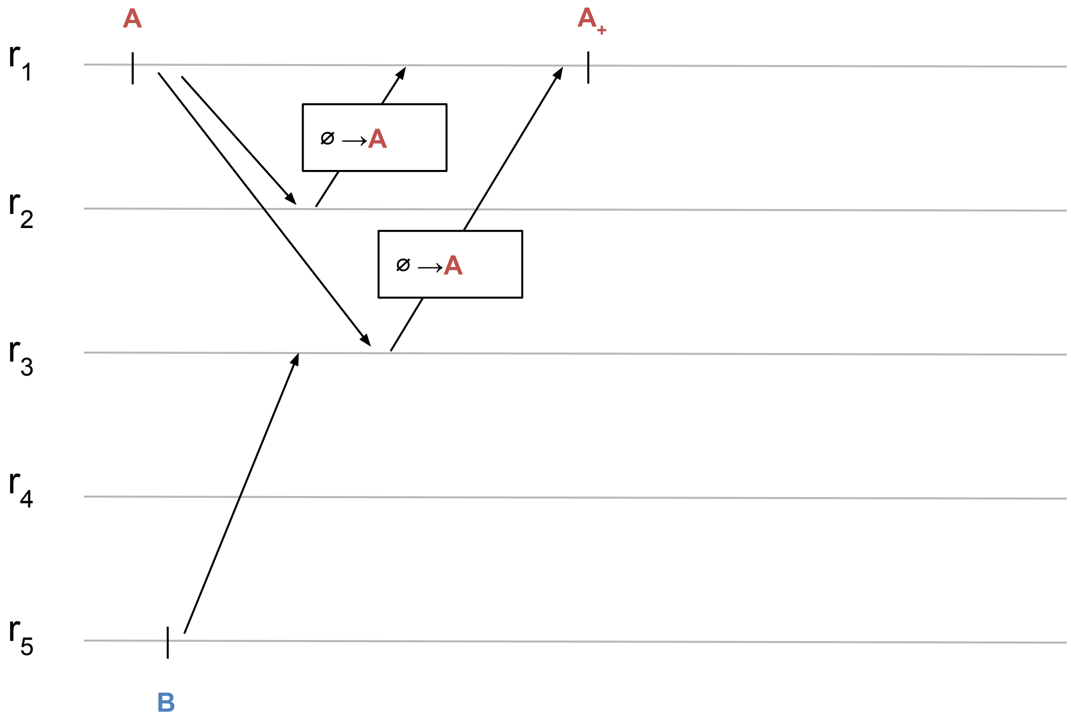


“nothing precedes A”

$$\left\{ \begin{array}{l} A = x \leftarrow 42 \\ B = y \leftarrow 7 \end{array} \right.$$

Commands A and B can execute in any order

SwiftPaxos / *dependencies tracking*



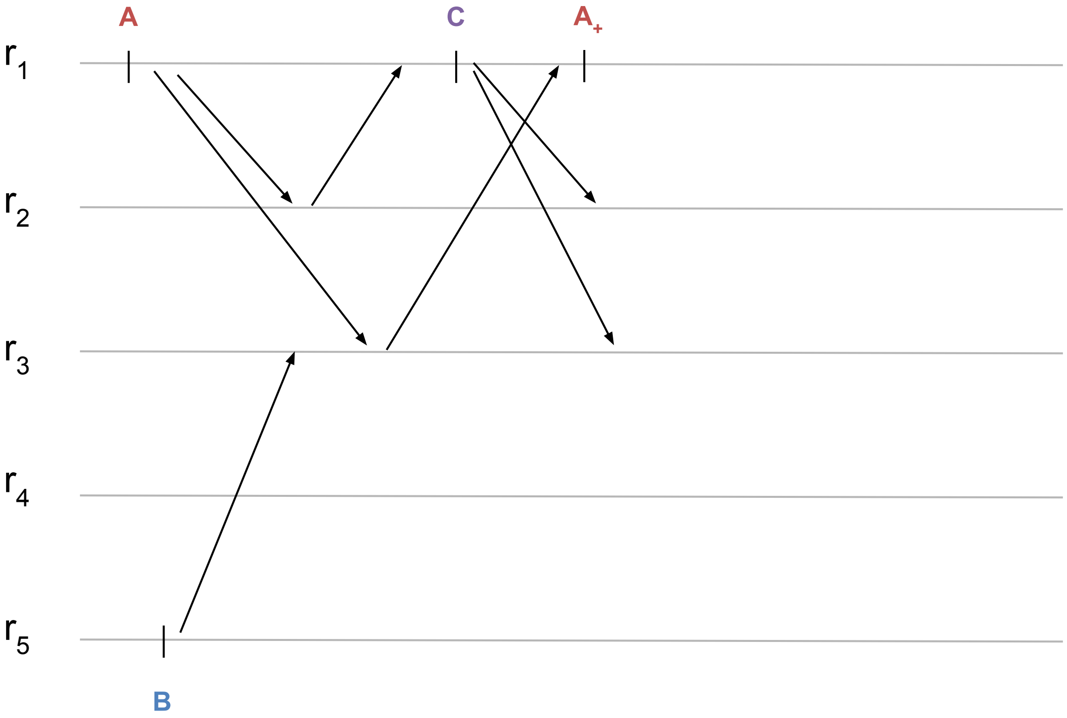
Features

- Optimal best-case latency
- Leverage commutativity

A = x ← 42

B = y ← 7

SwiftPaxos / *dependencies tracking*

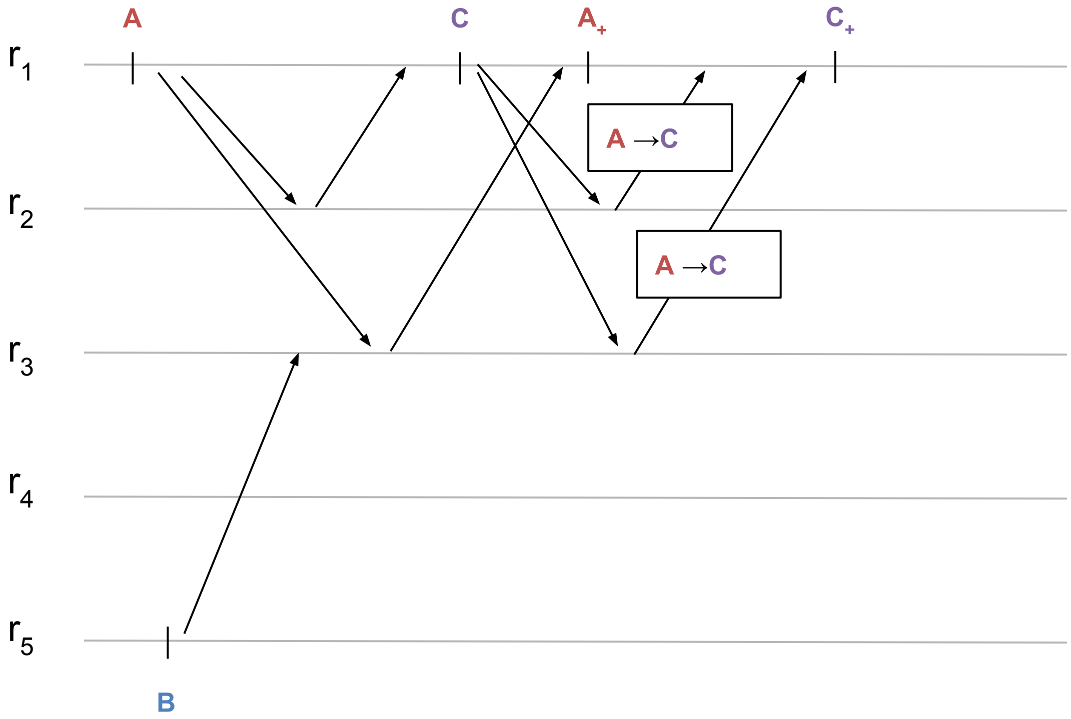


A = x ← 42

B = y ← 7

C = x ← 2x

SwiftPaxos / *dependencies tracking*

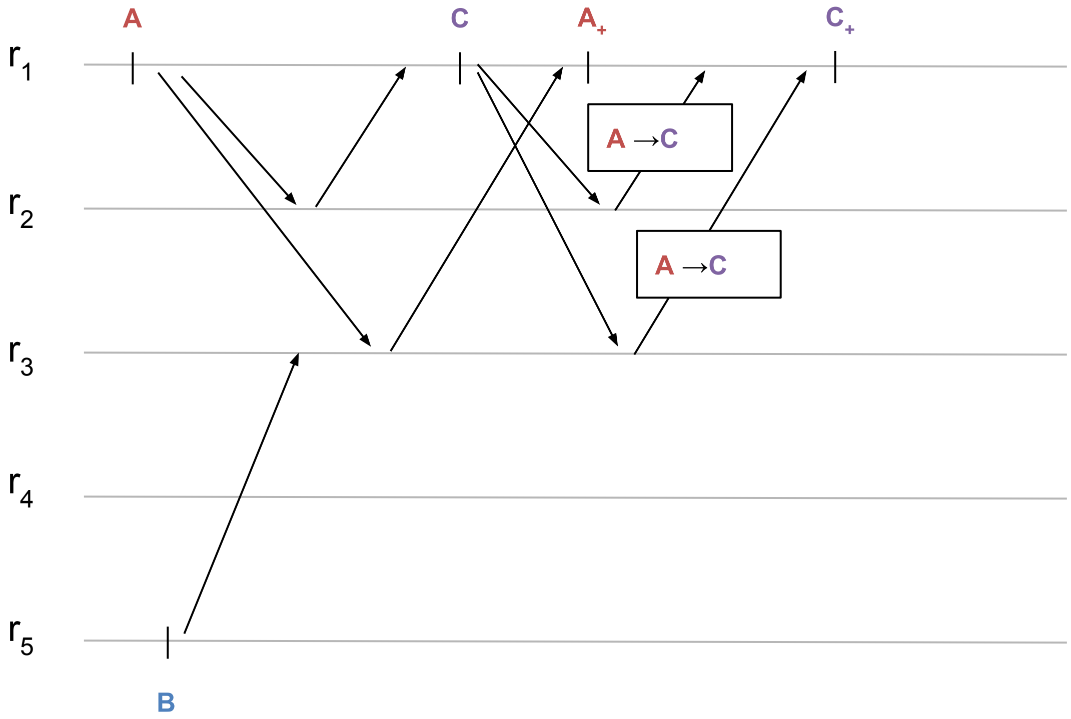


A = x ← 42

B = y ← 7

C = x ← 2x

SwiftPaxos / *dependencies tracking*



Features

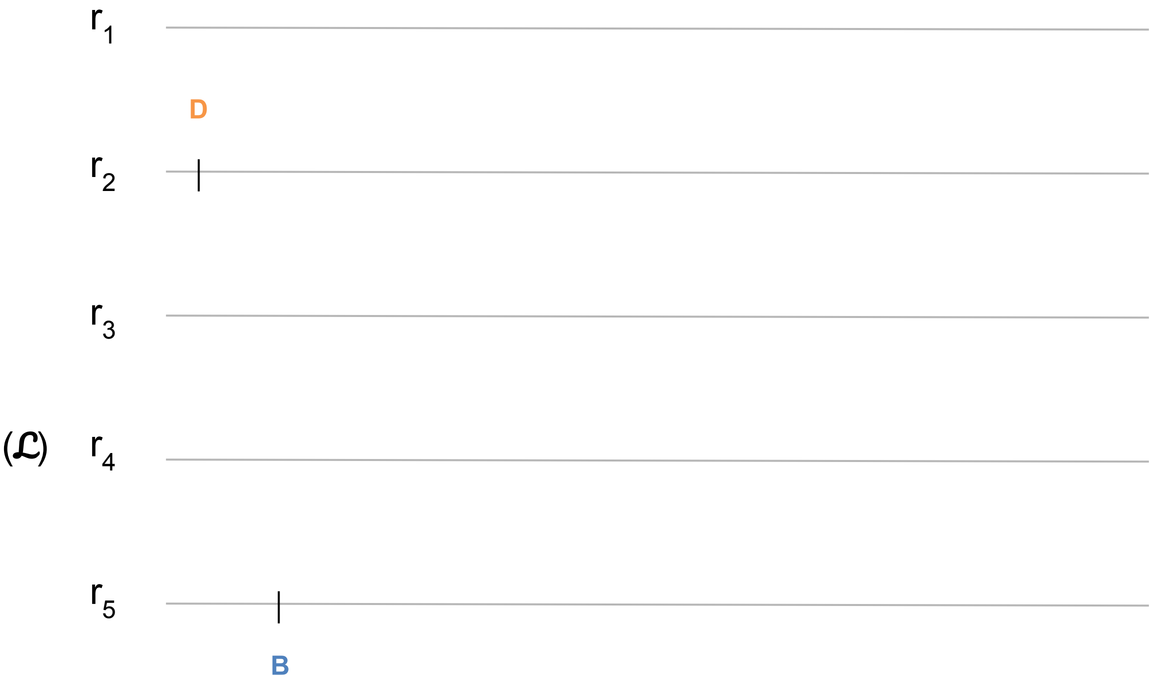
- Optimal best-case latency
- Leverage commutativity
- Use *spontaneous order* in the network

A = x ← 42

B = y ← 7

C = x ← 2x

SwiftPaxos / *double voting*

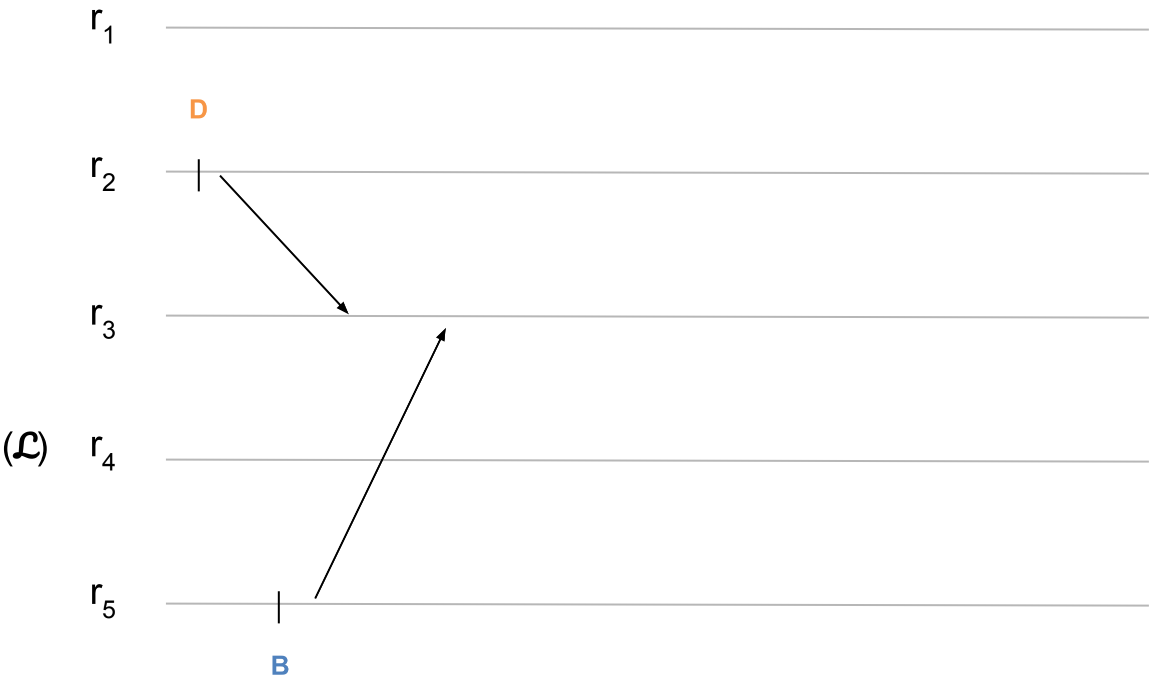


B = y ← 7

D = return x + y

ℒ = leader replica

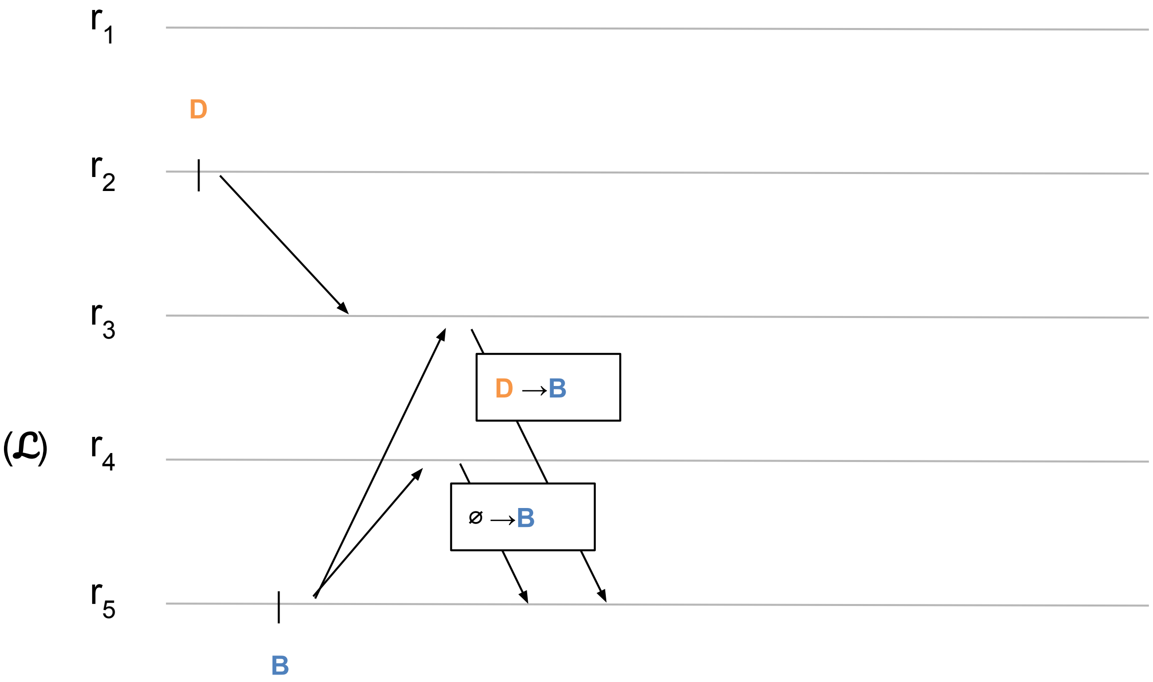
SwiftPaxos / *double voting*



B = y ← 7

D = return x + y

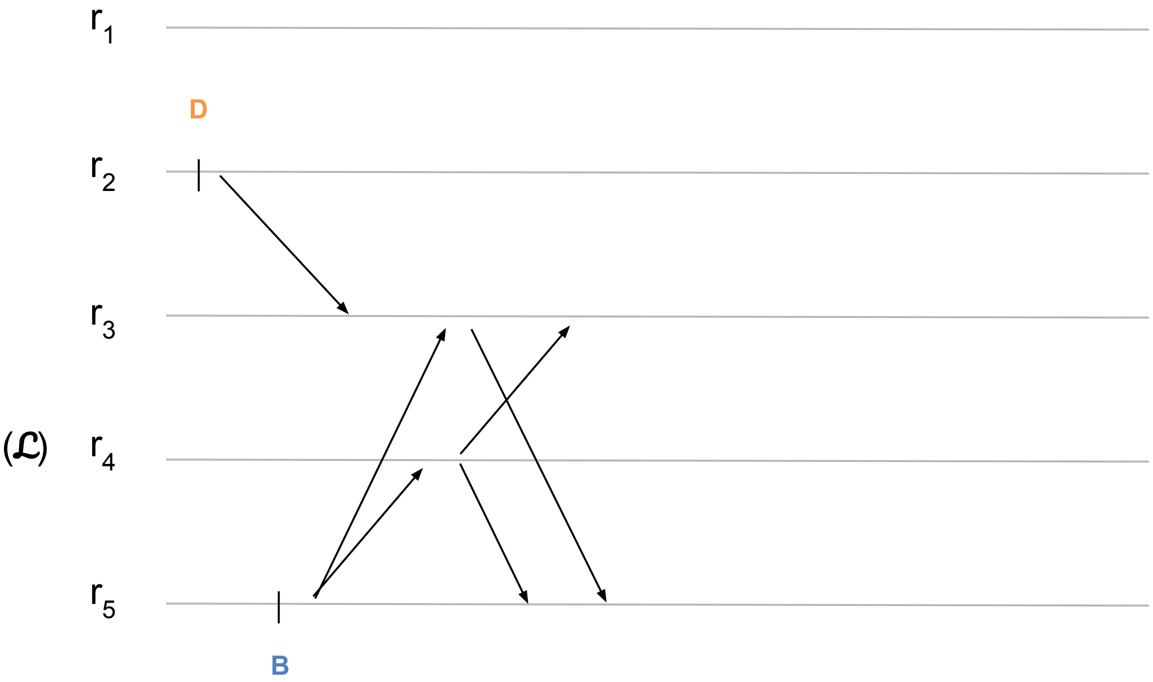
SwiftPaxos / *double voting*



B = y ← 7

D = return x + y

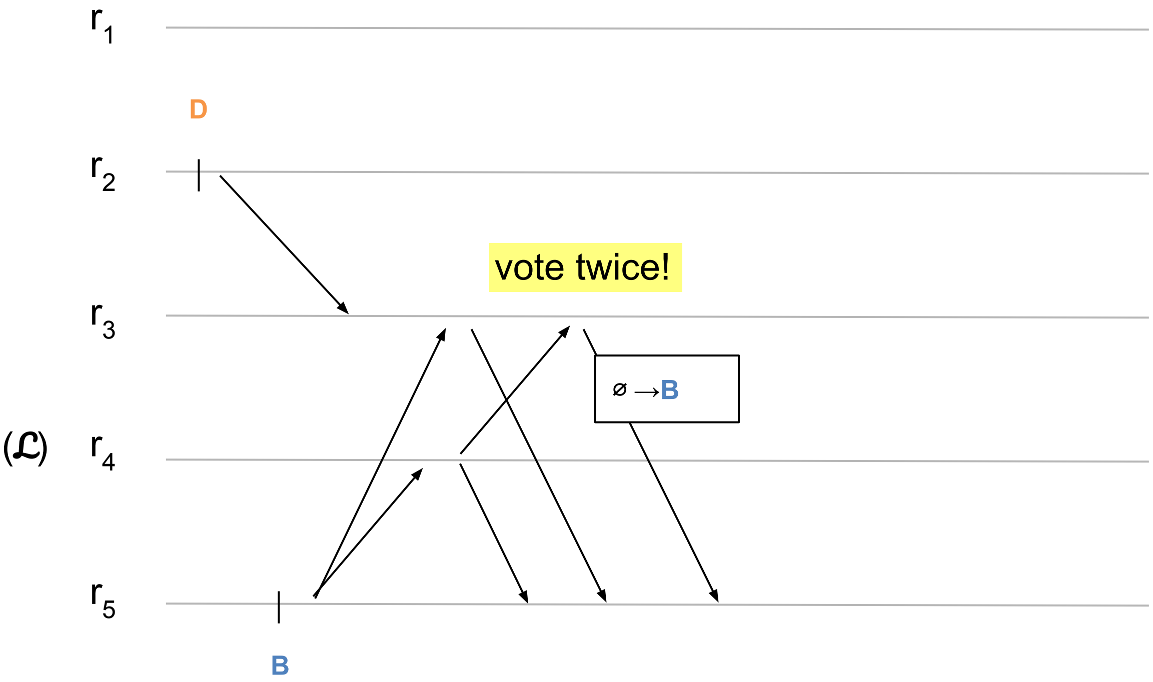
SwiftPaxos / *double voting*



B = y ← 7

D = return x + y

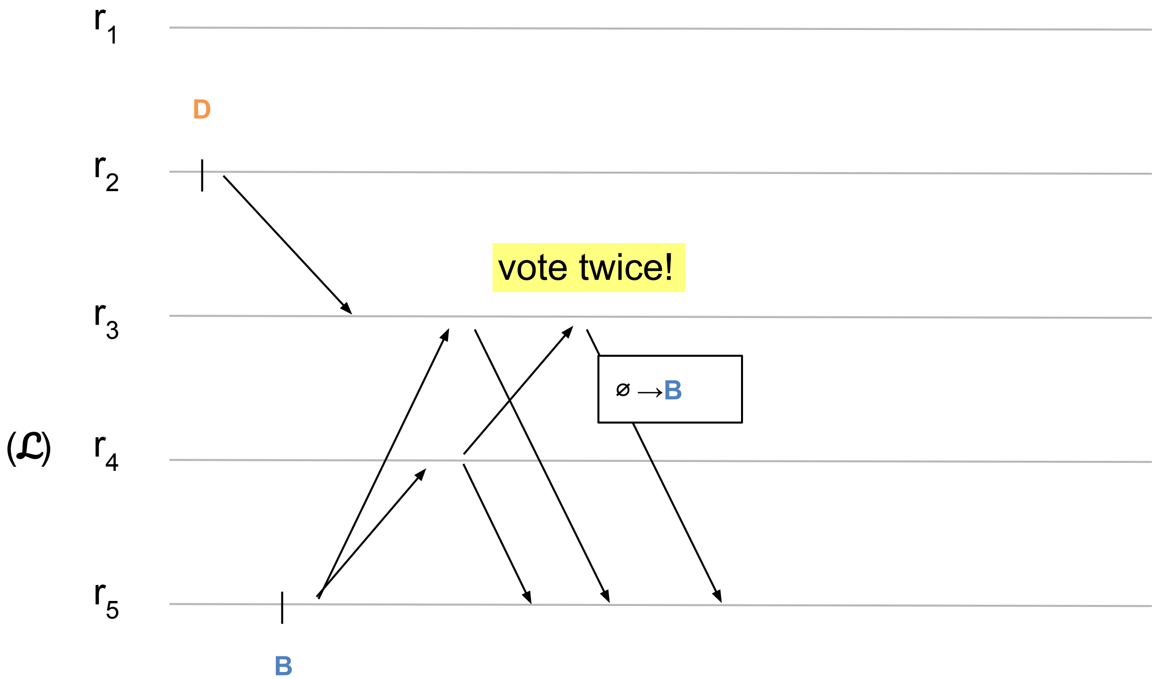
SwiftPaxos / *double voting*



B = y ← 7

D = return x + y

SwiftPaxos / *double voting*

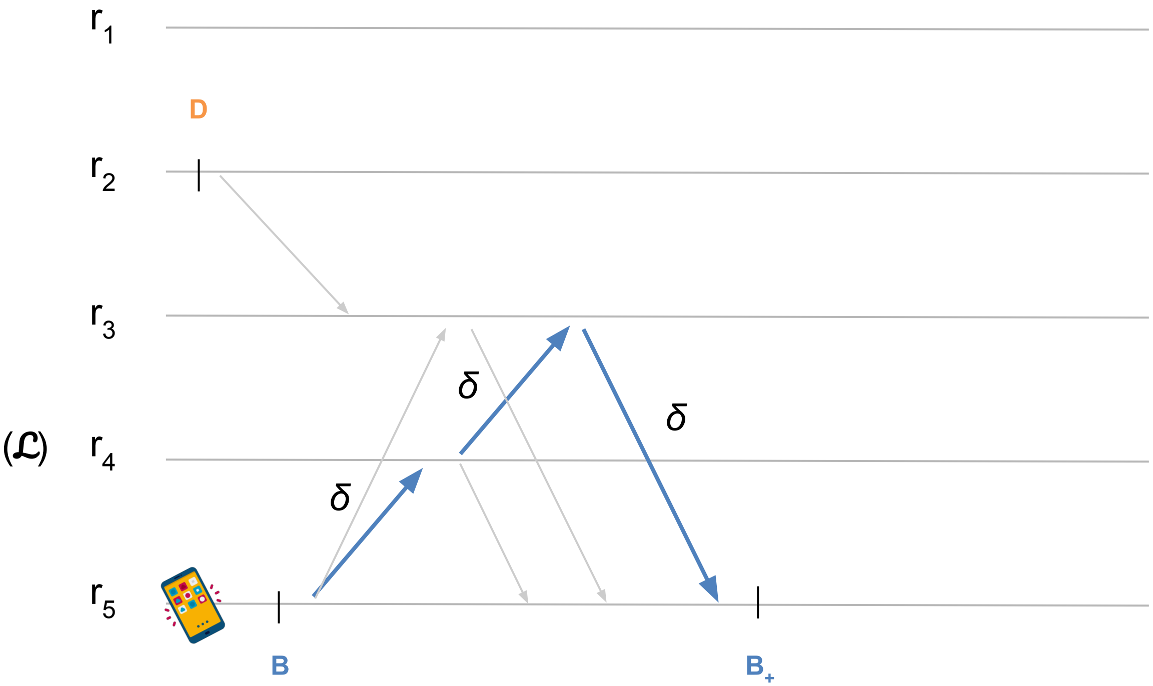


hint: the leader is part of all the quorums of the ongoing ballot

B = y ← 7

D = return x + y

SwiftPaxos / *double voting*



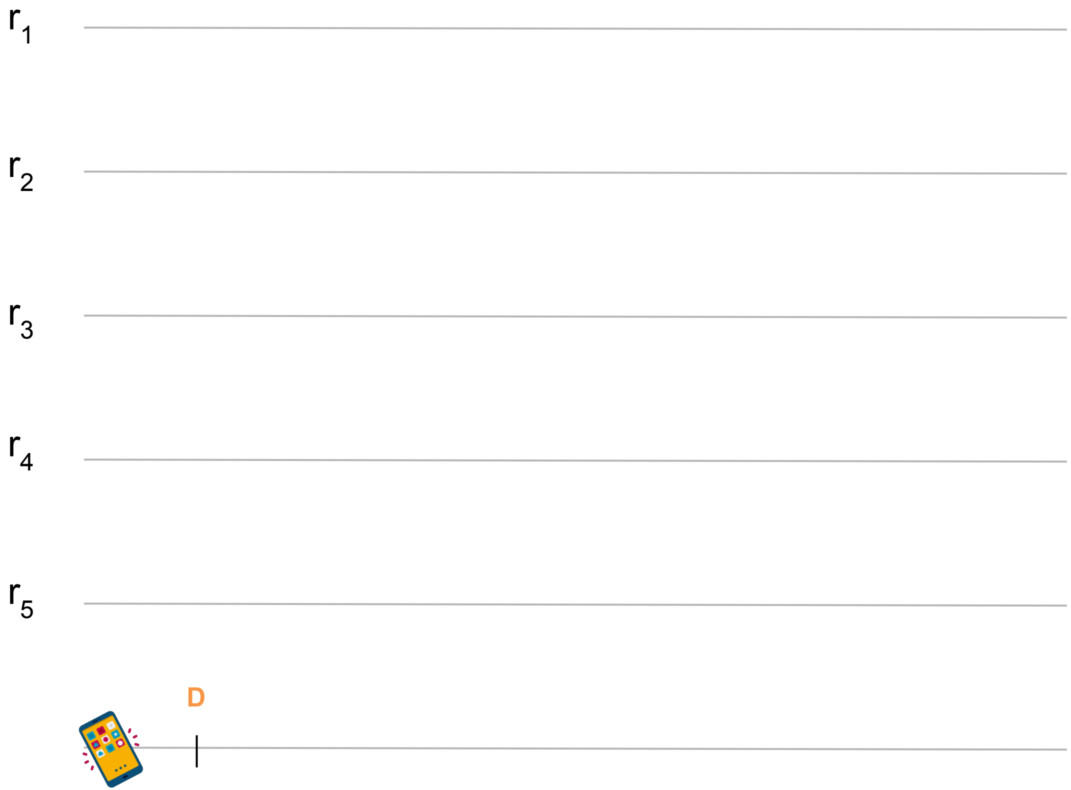
Features

- Optimal best-case latency
- Leverage commutativity
- Use *spontaneous order* in the network
- At least Paxos speed

B = `y ← 7`

D = `return x + y`

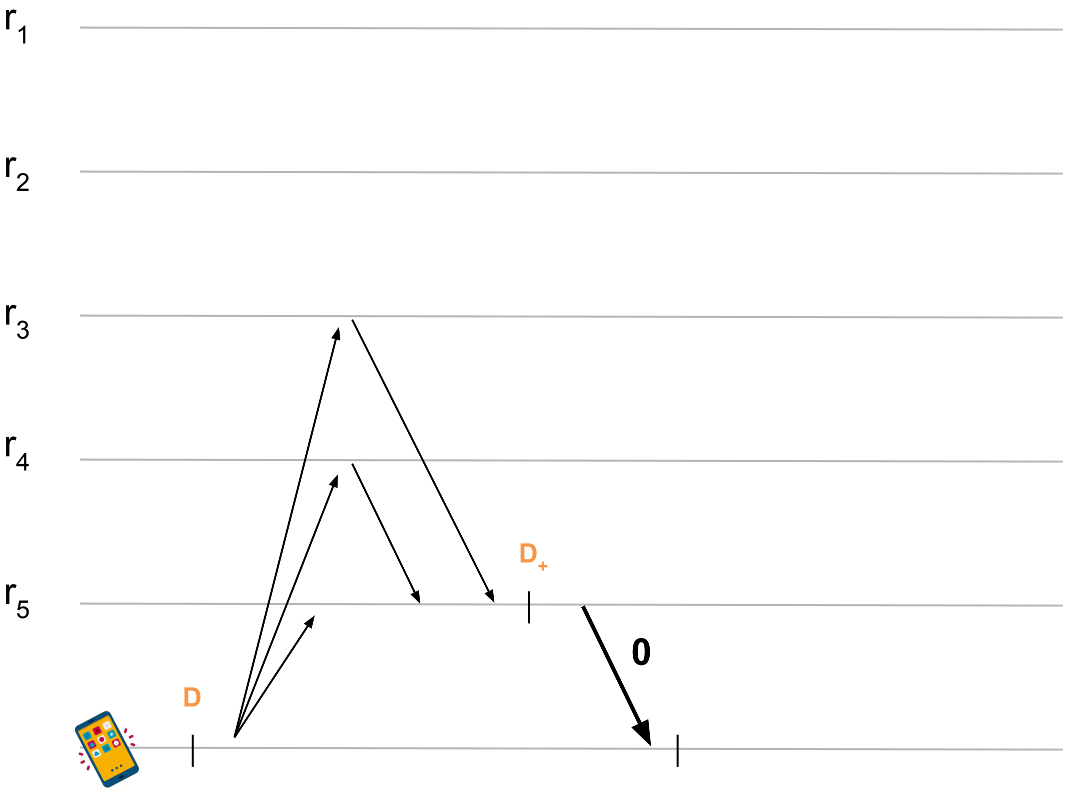
SwiftPaxos / *non-collocated clients*



B = `y ← 7`

D = `return x + y`

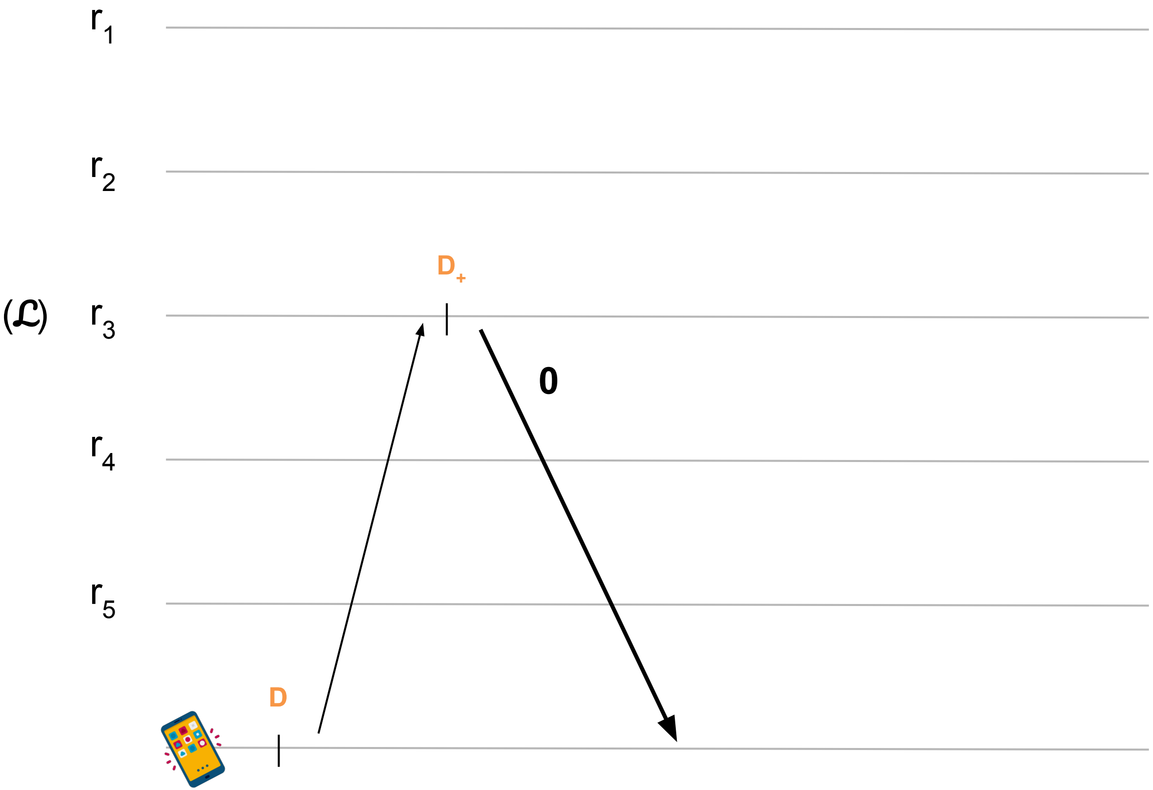
SwiftPaxos / *non-collocated clients*



B = y ← 7

D = return x + y

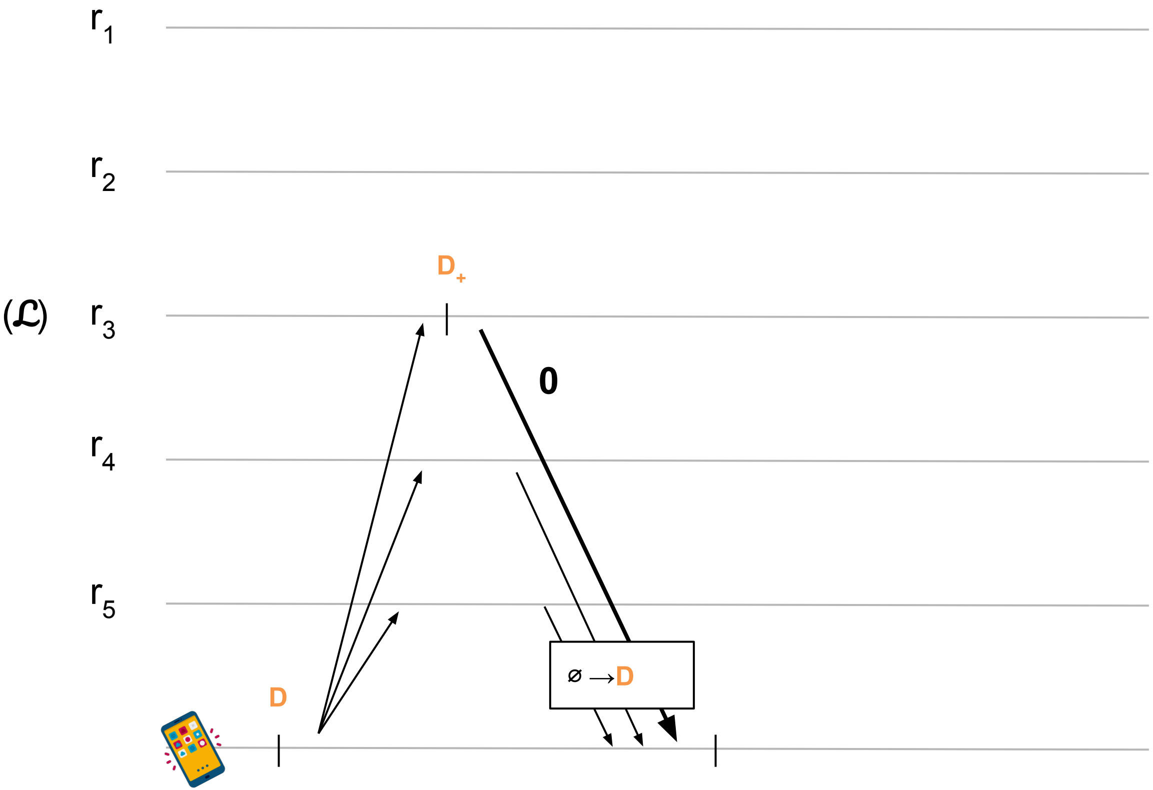
SwiftPaxos / *optimistic execution*



B = y ← 7

D = return x + y

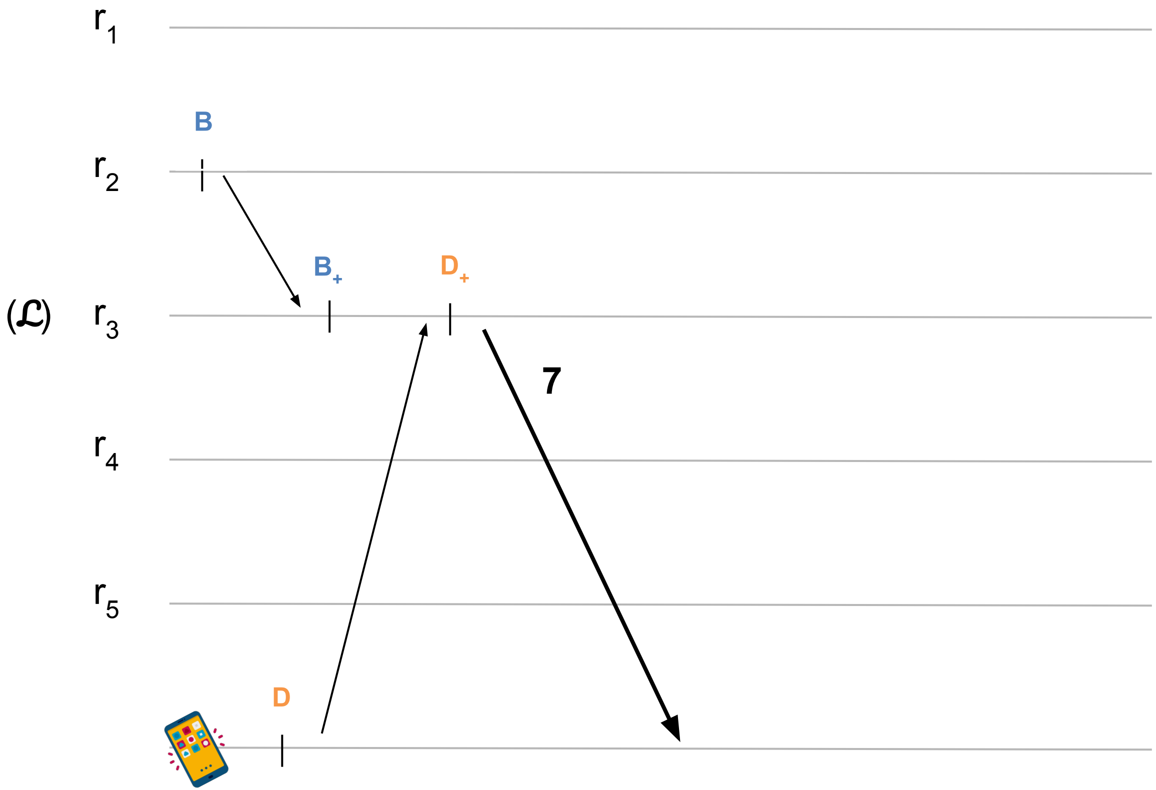
SwiftPaxos / *optimistic execution*



B = y ← 7

D = return x + y

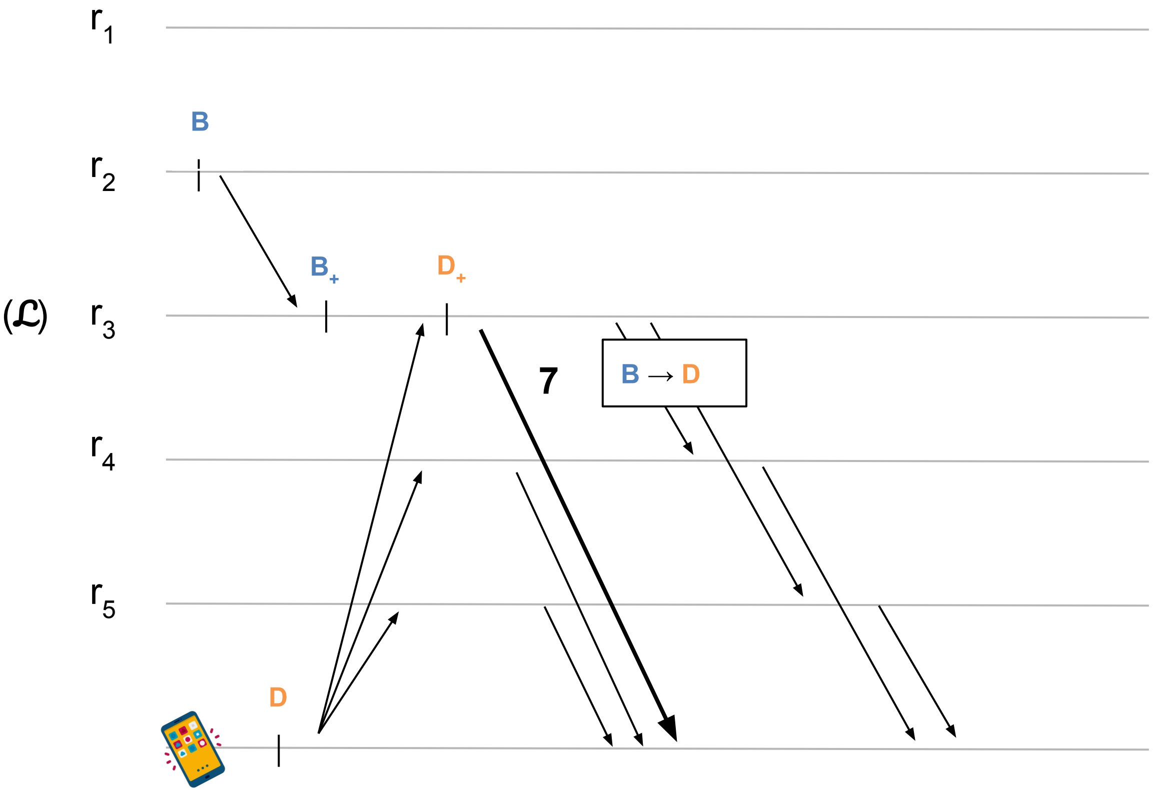
SwiftPaxos / *optimistic execution*



B = `y ← 7`

D = `return x + y`

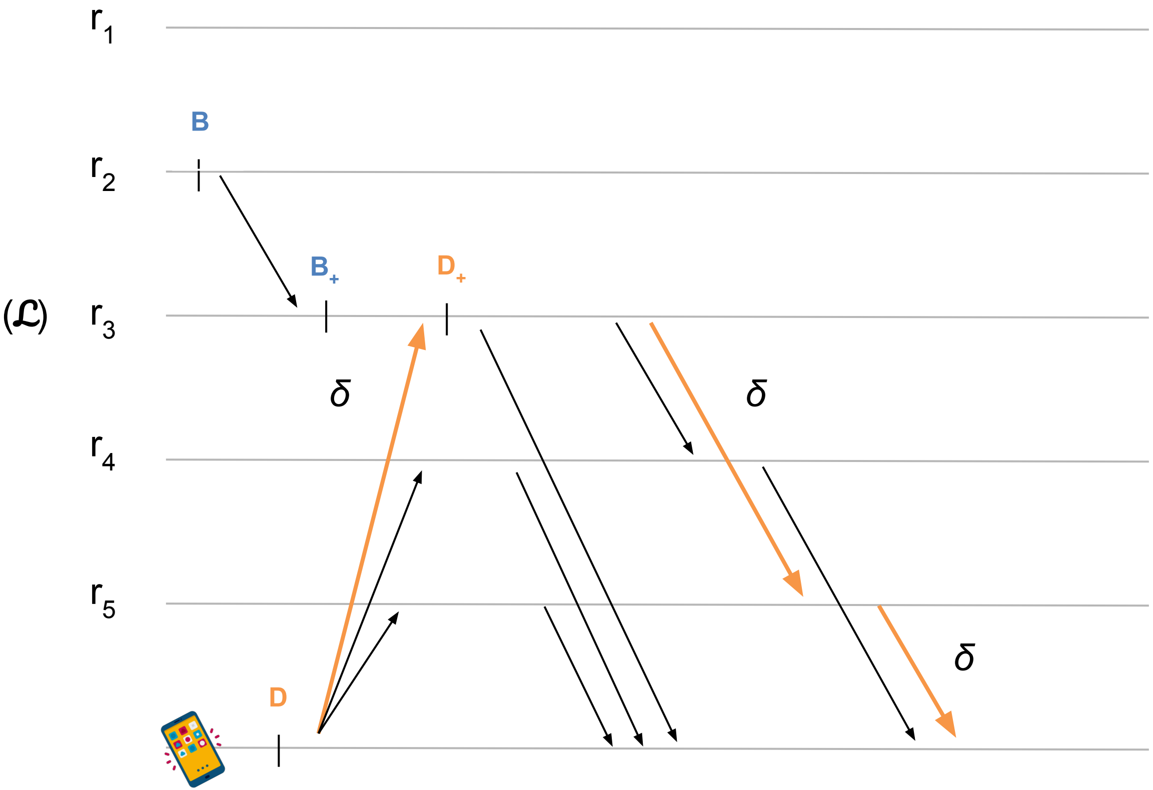
SwiftPaxos / *optimistic execution*



B = `y ← 7`

D = `return x + y`

SwiftPaxos / *latency (2δ, 3δ)*



Features

- Optimal best-case latency
- Leverage commutativity
- Use *spontaneous order* in the network
- At least Paxos speed
- Quick reply to non-collocated clients

B = `y ← 7`

D = `return x + y`

SwiftPaxos / *related work*

	sequential	*conflict-free*	*contention-free*	*general*
Paxos	4δ			
FastPaxos+	$2\delta+1$	$3\delta+1$		
Generalized Paxos	$2\delta+1$			$6\delta+1$
Egalitarian Paxos	$2\delta+1$			$O(n\delta)$
CURP	2δ	$3\delta+1$		
SwiftPaxos	2δ			3δ

(*sequential*) no concurrent commands

(*conflict-free*) concurrent commands do not conflict

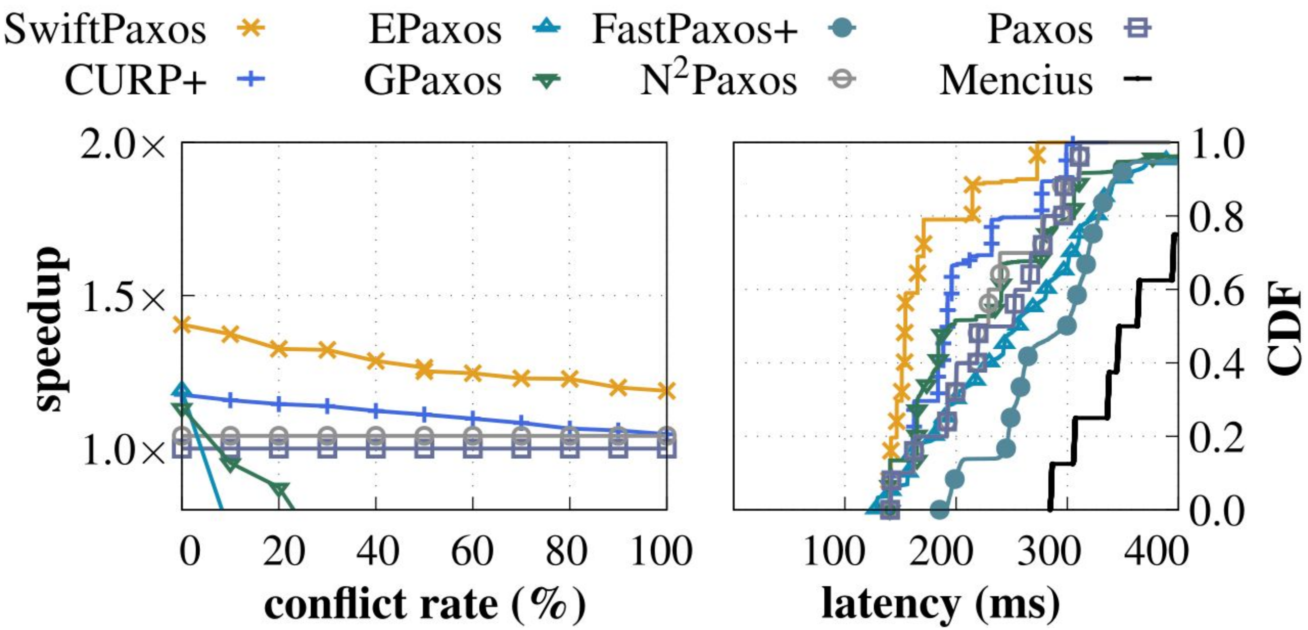
(*contention-free*) concurrent conflicting commands are received in the same order everywhere

SwiftPaxos / *related work*

	fast path	*dependencies tracking*	*optimistic execution*	*double-voting*
Paxos	✗	✗	✗	✗
FastPaxos+	✔	✗	✗	✗
Generalized Paxos	✔	✓	✗	✗
Egalitarian Paxos	✔	✔	✗	✗
CURP	✔	✓	✔	✗
SwiftPaxos	✔	✔	✔	✔

✔ *supported*
✓ *partially supported*
✗ *not supported*

Experiments / *average latency*



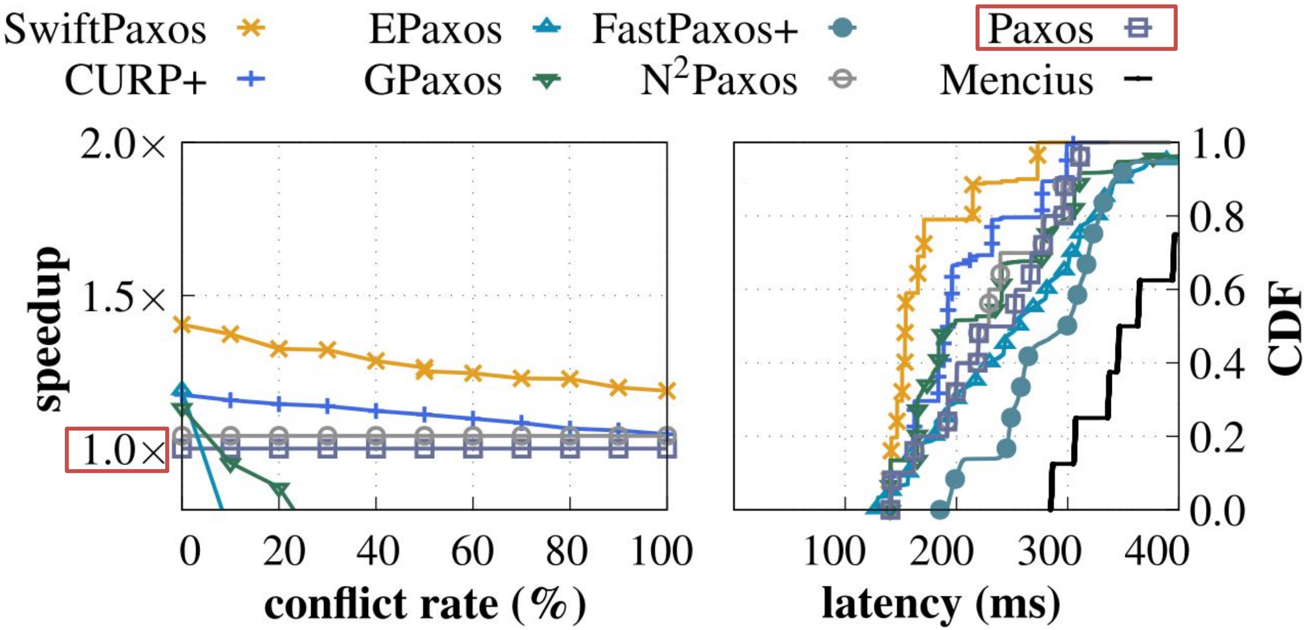
13 AWS EC2 sites
- 5 replica sites
- 10 client sites

no-op service (1KB)

varying conflict rate

fixed 2% conflict rate

Experiments / *average latency*

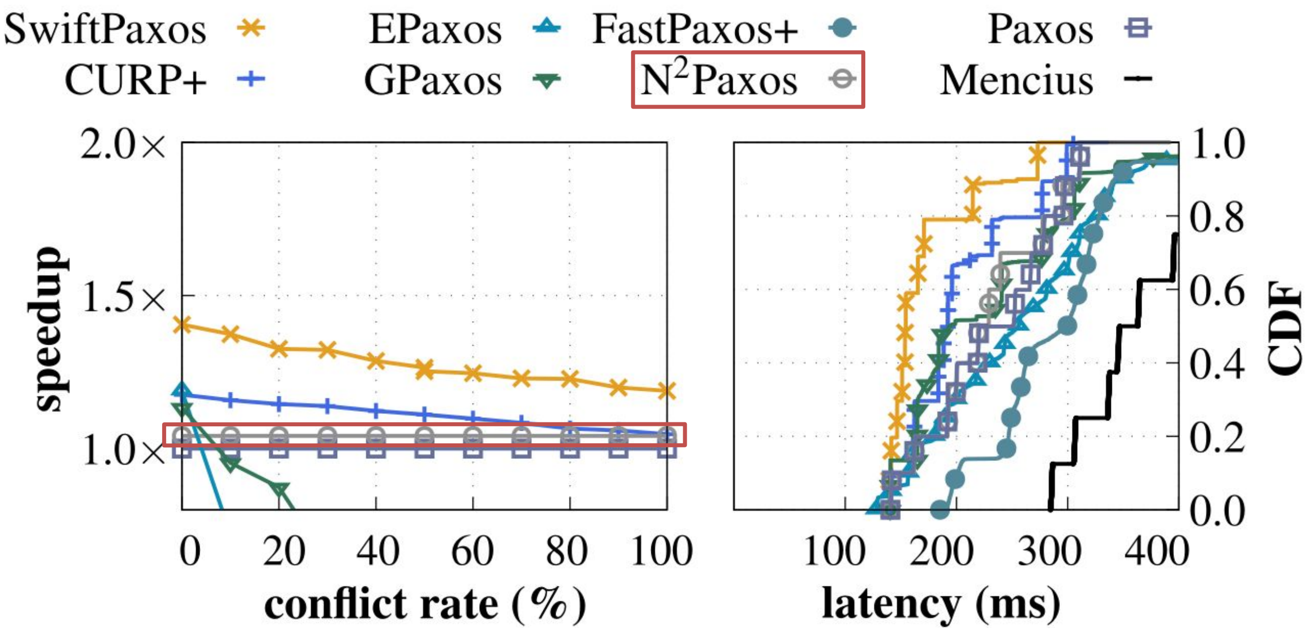


13 AWS EC2 sites
- 5 replica sites
- 10 client sites

no-op service (1KB)

- baseline

Experiments / *average latency*



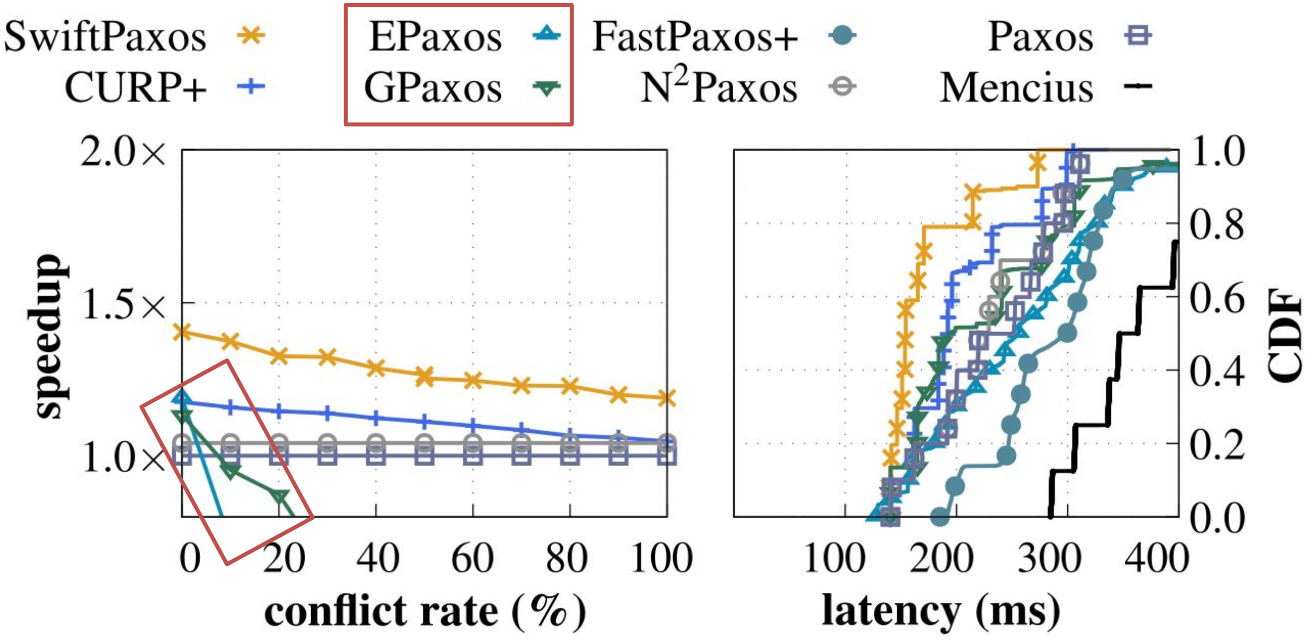
13 AWS EC2 sites

- 5 replica sites
- 10 client sites

no-op service (1KB)

- around 5% faster

Experiments / *average latency*



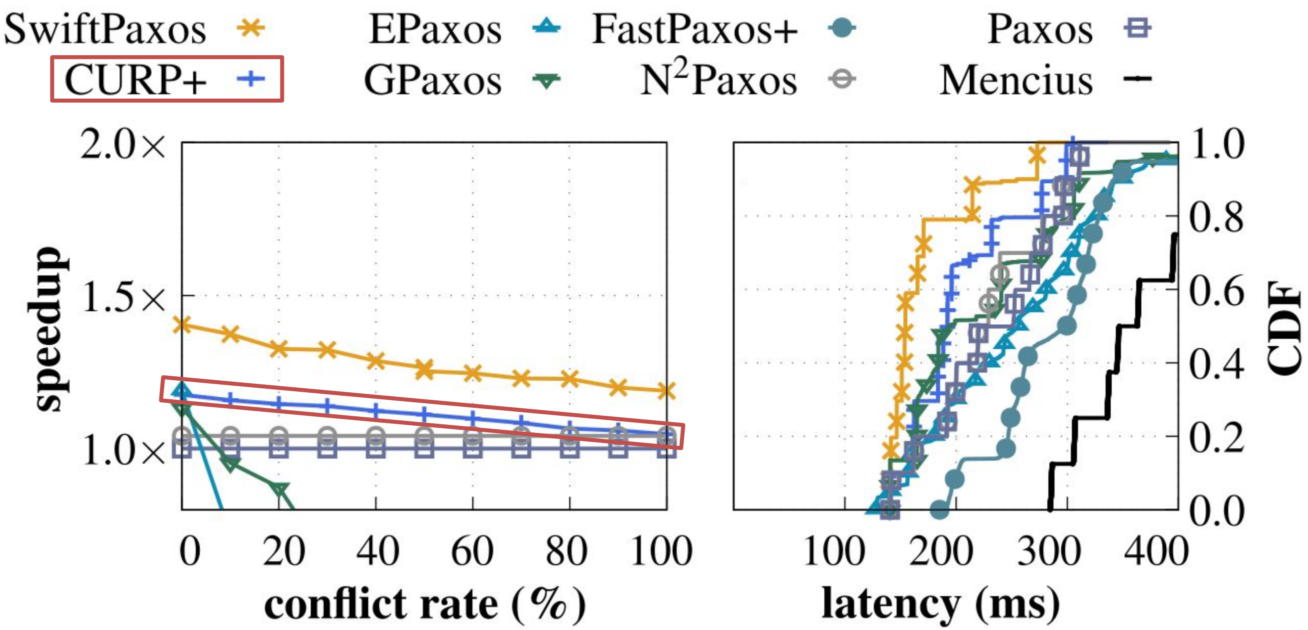
13 AWS EC2 sites

- 5 replica sites
- 10 client sites

no-op service (1KB)

- performance quickly drops w. conflicts

Experiments / *average latency*

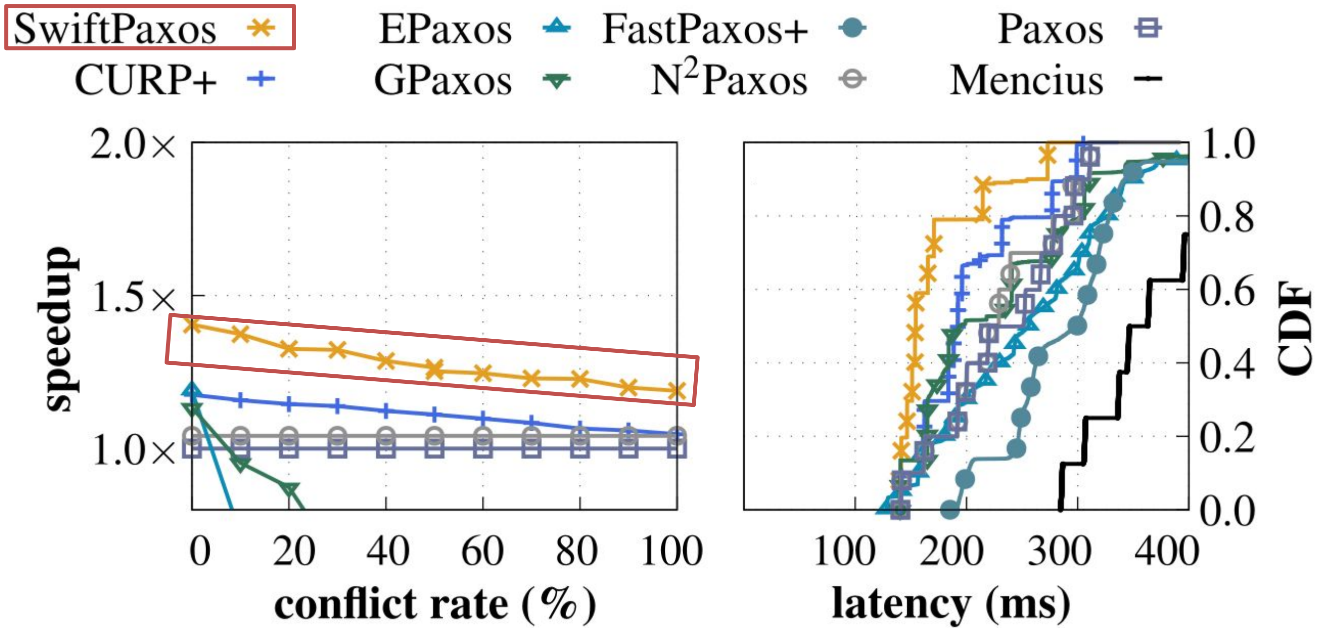


13 AWS EC2 sites
- 5 replica sites
- 10 client sites

no-op service (1KB)

- 18% faster than Paxos on average

Experiments / *average latency*



13 AWS EC2 sites

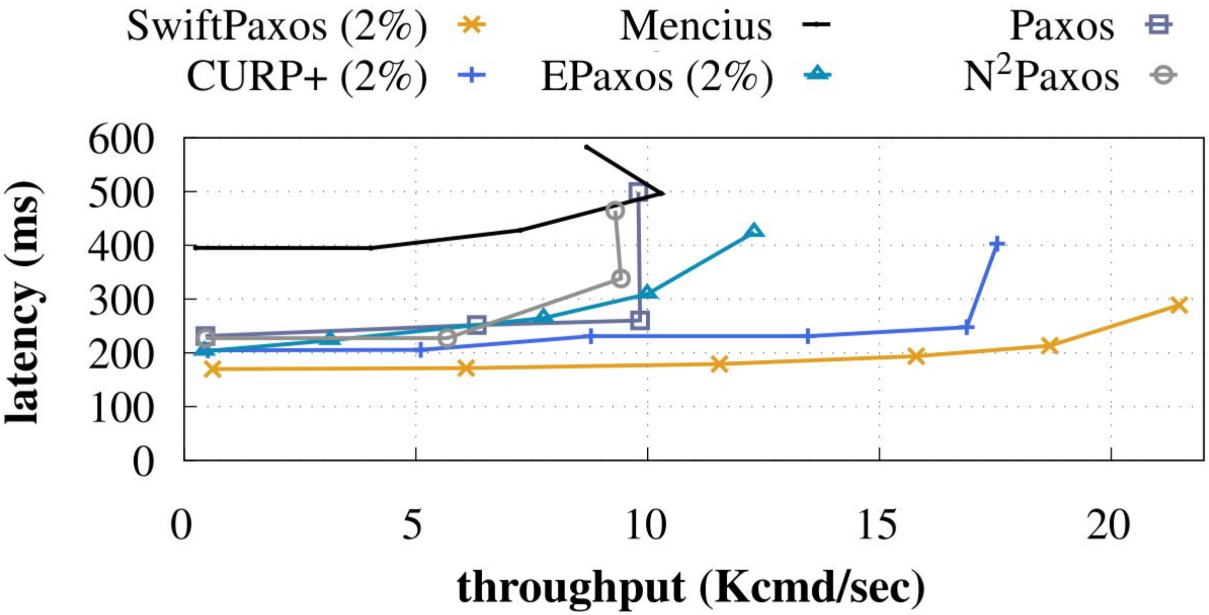
- 5 replica sites
- 10 client sites

no-op service (1KB)

Takeaways:

- fastest protocol of all (up to 1.4x)
- always *at least* Paxos speed

Experiments / *scalability*

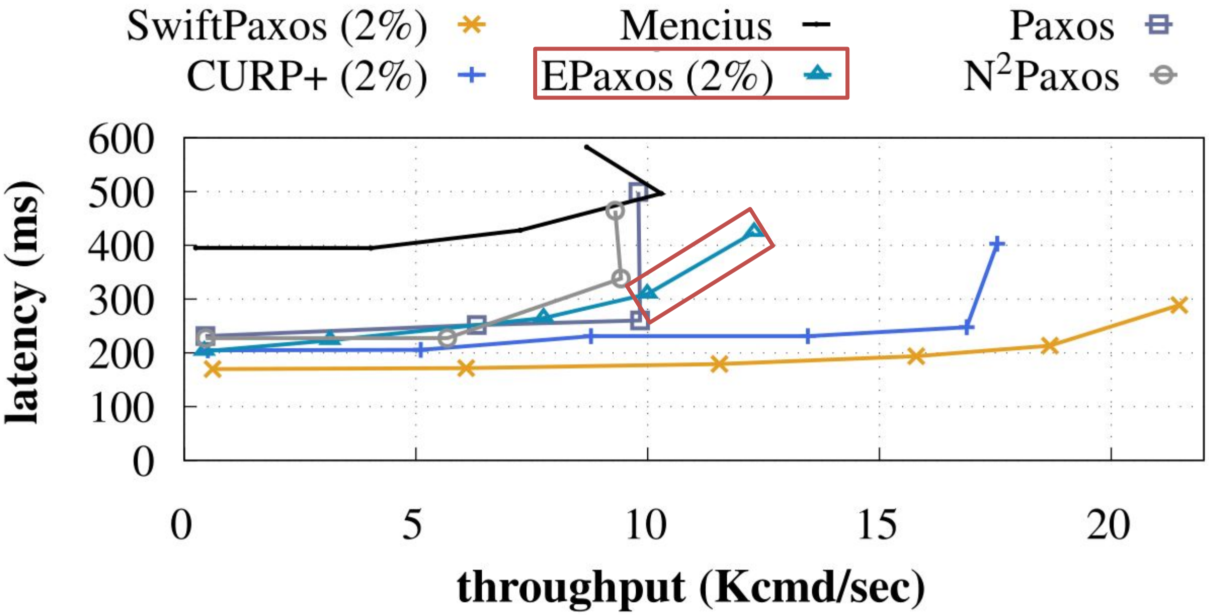


13 AWS EC2 sites
- 5 replica sites
- 10 client sites

no-op service (3KB)

progressively add new clients (up to 5,000)
at a fixed 2% conflict rate

Experiments / *scalability*



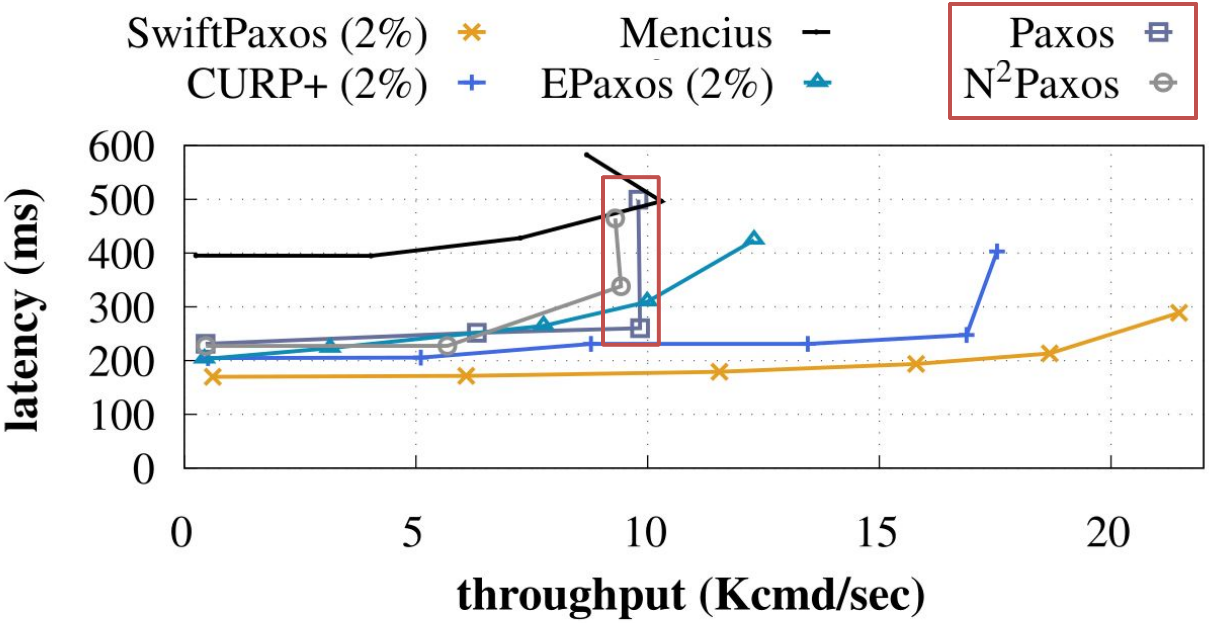
13 AWS EC2 sites

- 5 replica sites
- 10 client sites

no-op service (3KB)

- performance saturates due to convoy effect (long chains)

Experiments / *scalability*



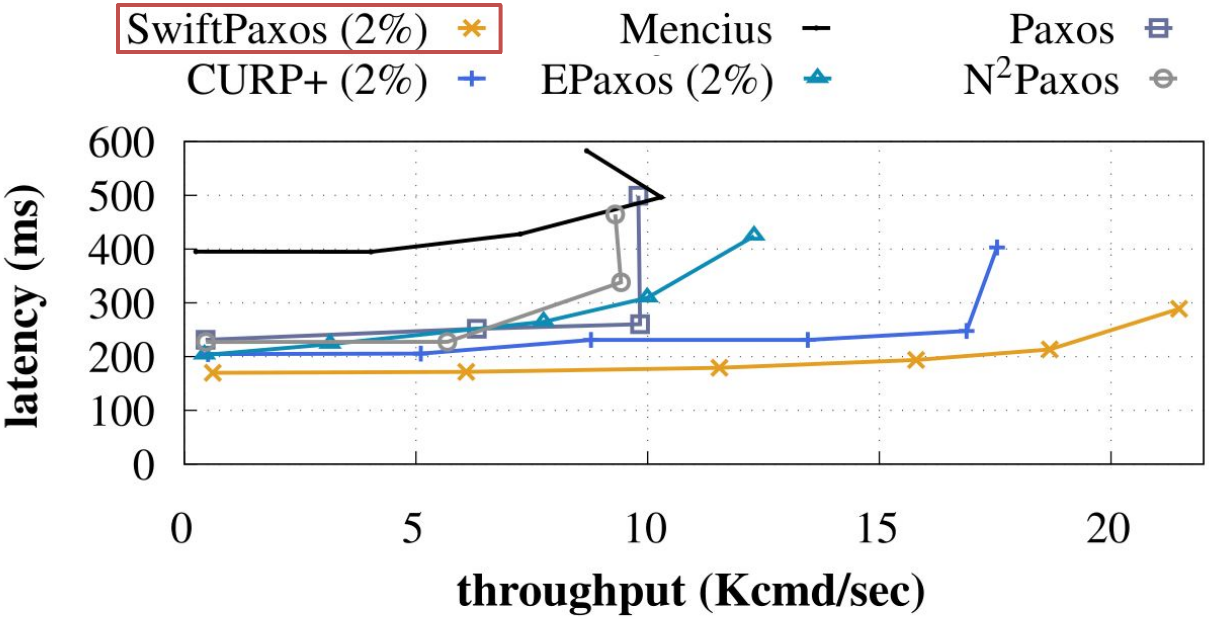
13 AWS EC2 sites

- 5 replica sites
- 10 client sites

no-op service (3KB)

- the leader is bottlenecking (because it disseminates all commands)

Experiments / *scalability*



13 AWS EC2 sites

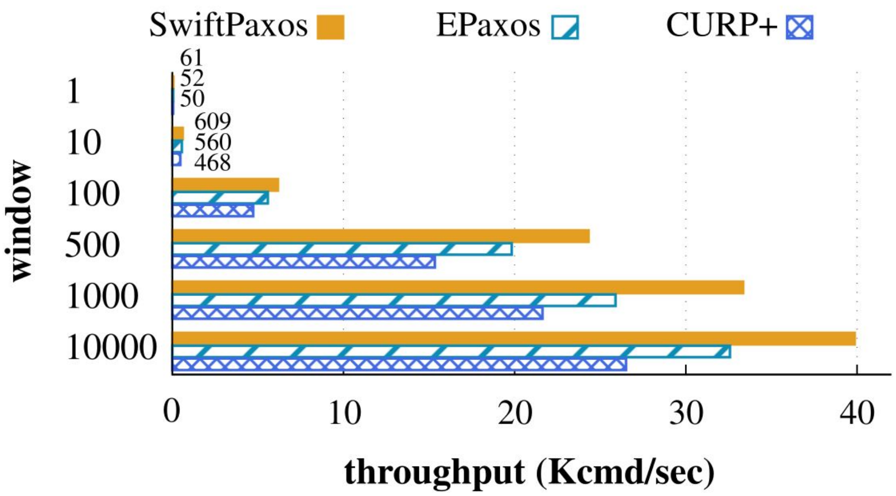
- 5 replica sites
- 10 client sites

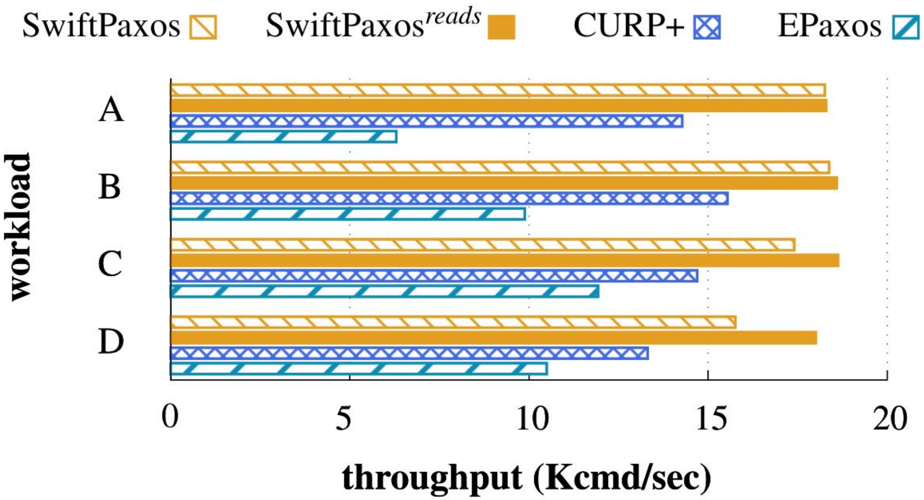
no-op service (3KB)

Takeaways:

- stable performance
- low overhead at the leader

Experiments / *applications*

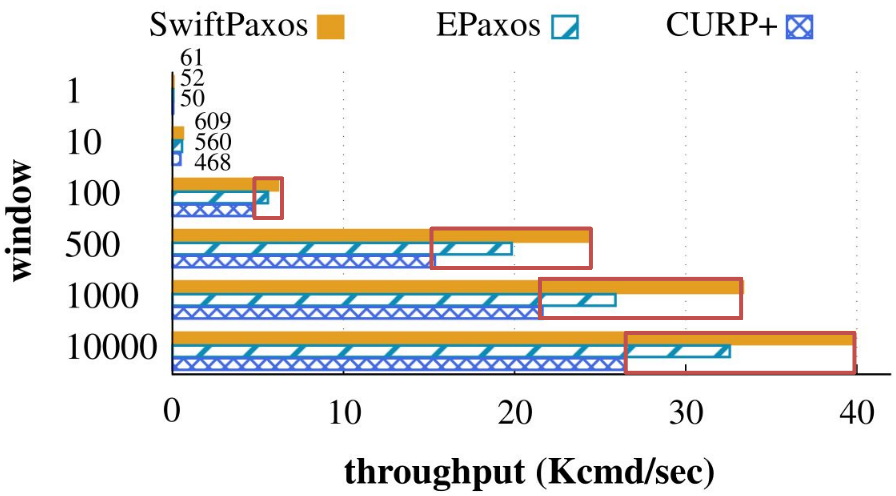


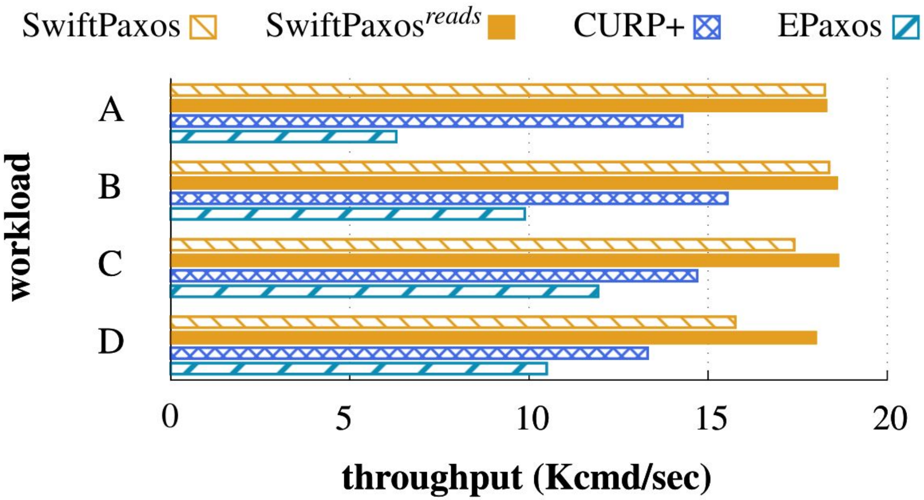


pipelining

YCSB

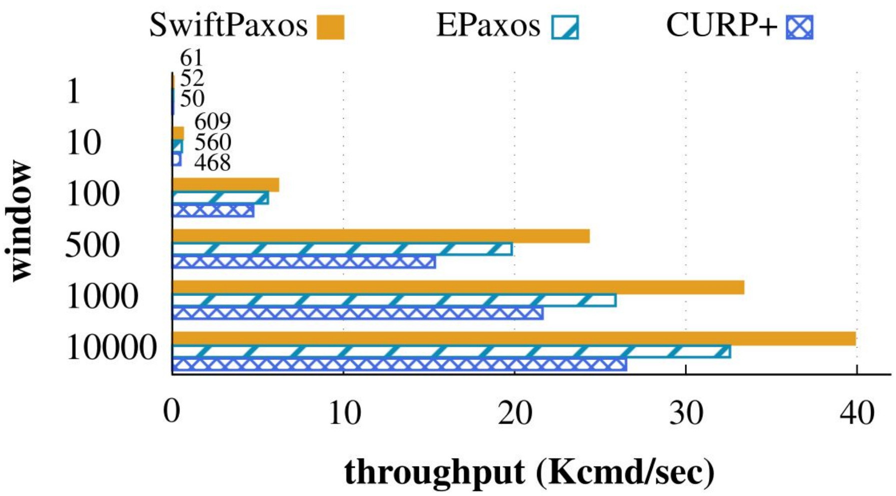
Experiments / *applications*

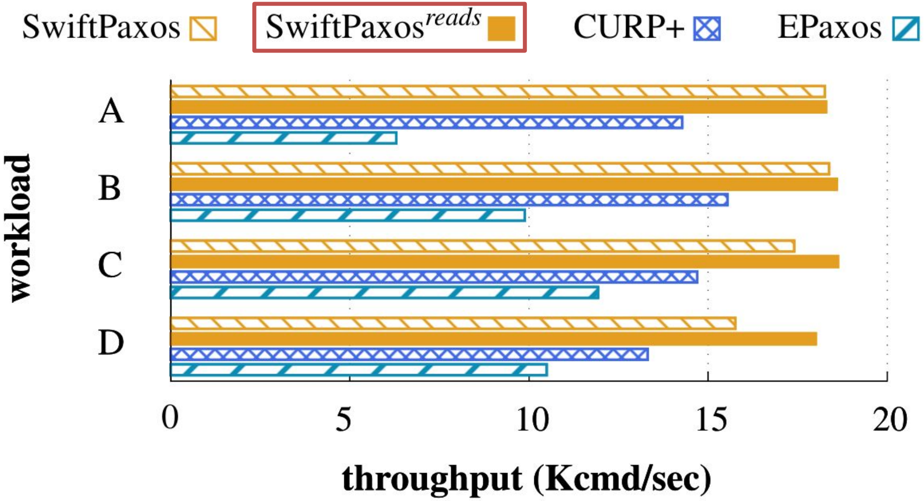




- up to 49% improvement over CURP [NSDI '19]

Experiments / *applications*





Takeaways:

- consistently *better performance* than competitors
- can execute *fast linearizable reads* at any replica

Conclusion

SwiftPaxos

- A *new* (leader-driven) state-machine replication protocol
- Executes commands in optimal time:
  - 1 RTT when no contention
  - 1.5 RTT otherwise.

In practice,

- *always* faster than Paxos (16-29% better)
- up to 2.9x higher throughput than alternatives
- low metadata usage

try it!