planet-scale leaderless consensus

Vitor Enes

Dr. **Alexey Gotsman**, Research Professor at IMDEA

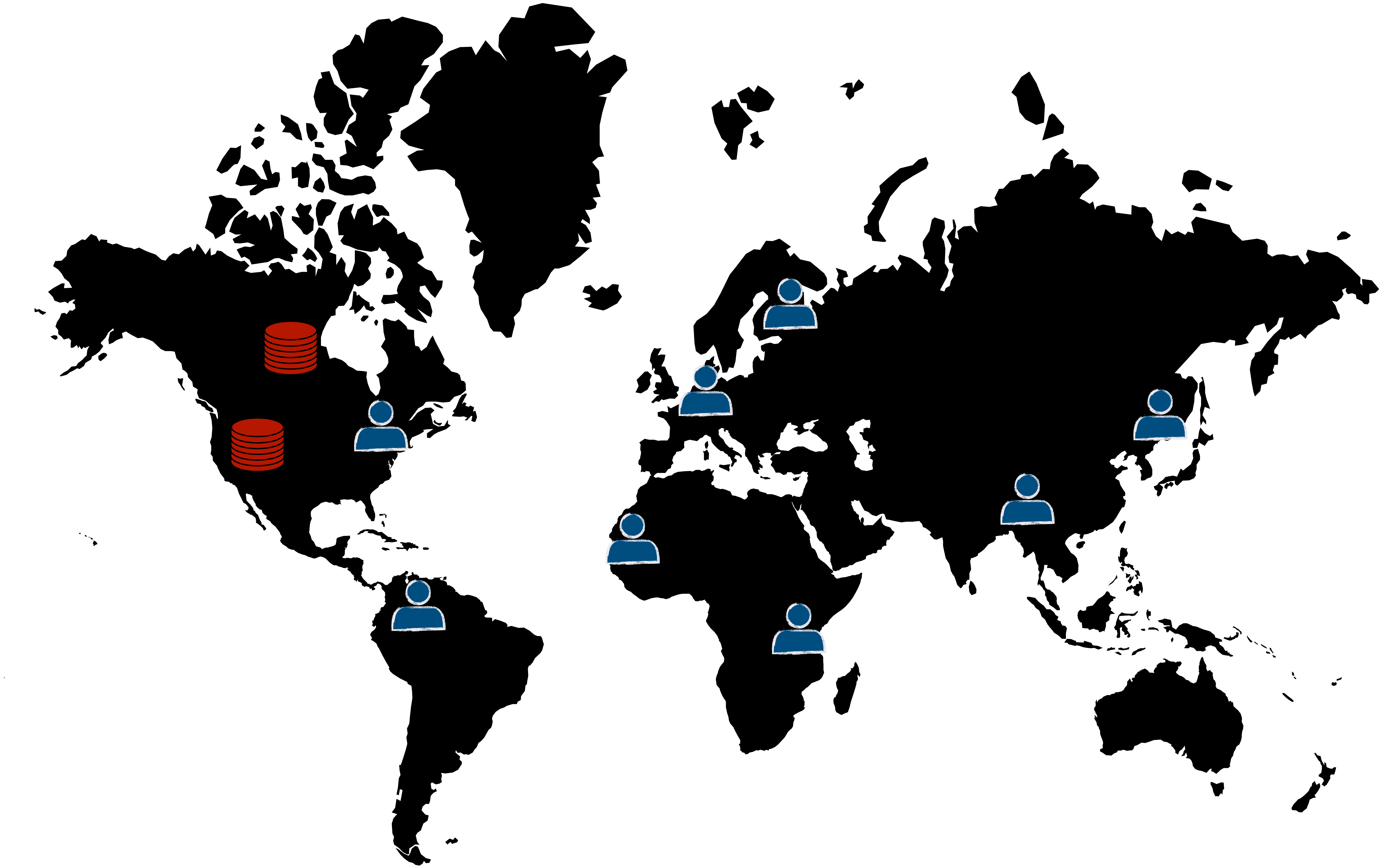
Dr. **Carlos Baquero**, Professor at FEUP

Universidade do Minho

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY

INESCTEC

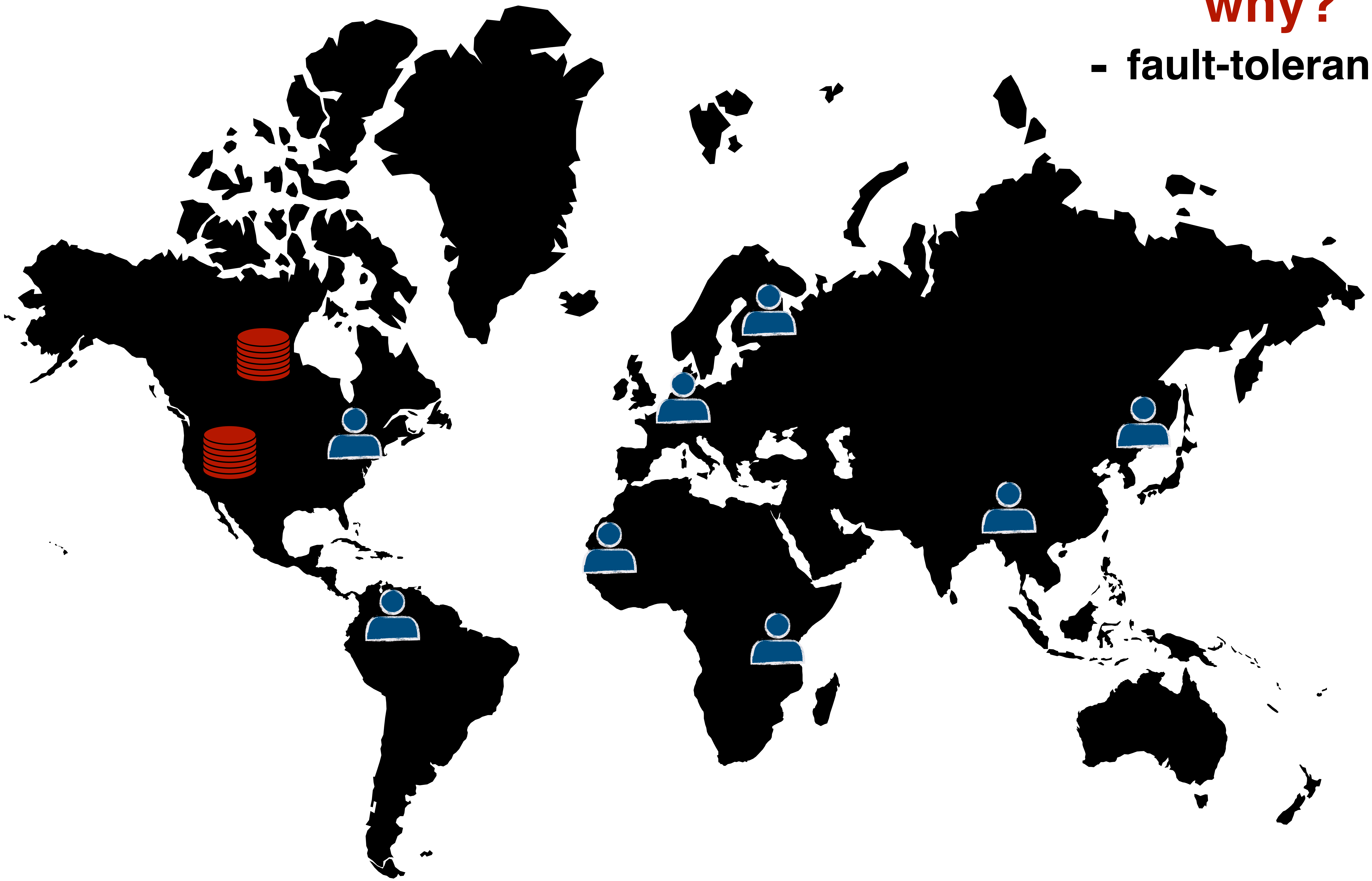
planet-scale replicated systems



planet-scale replicated systems

why?

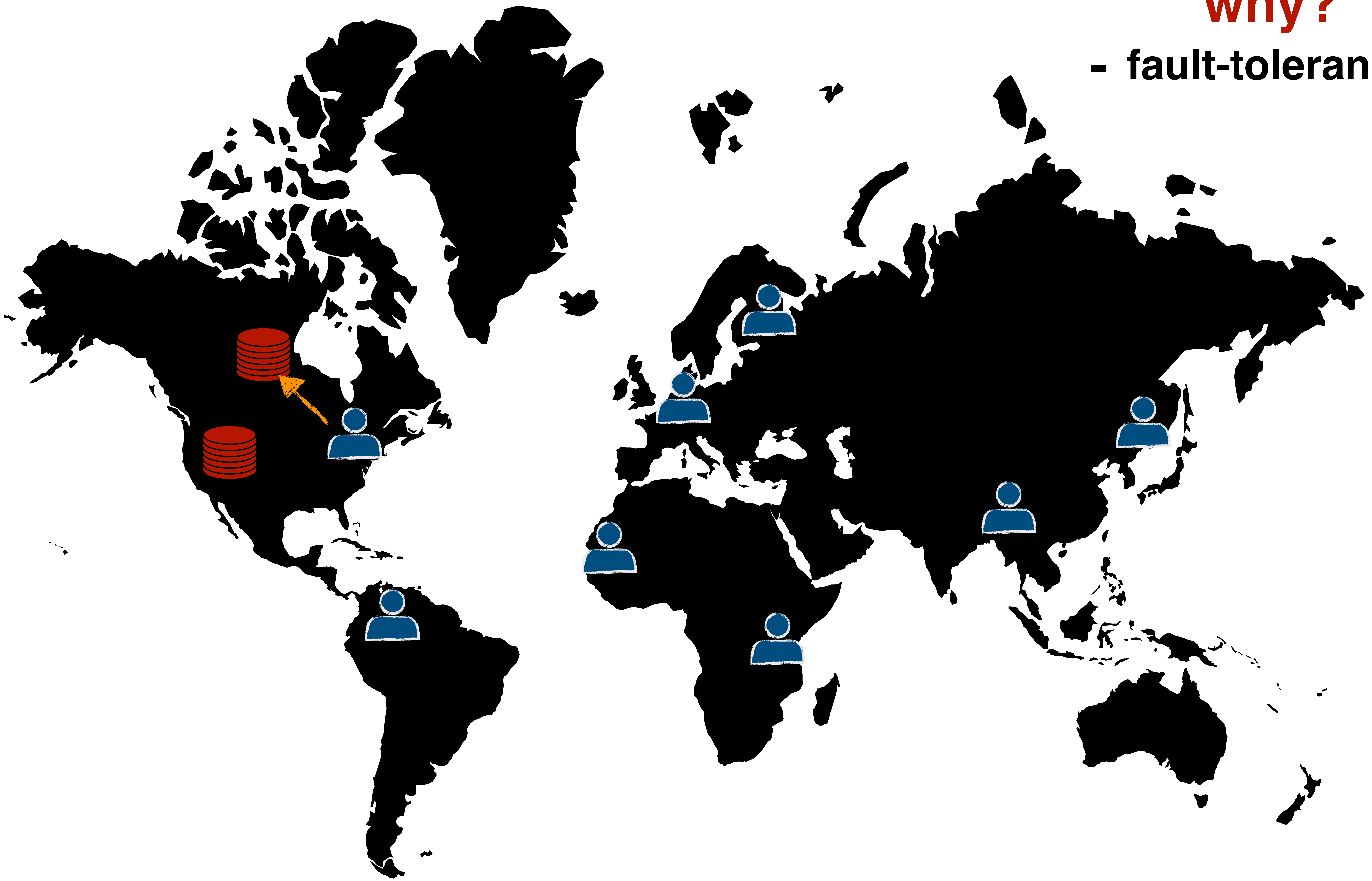
- fault-tolerance



planet-scale replicated systems

why?

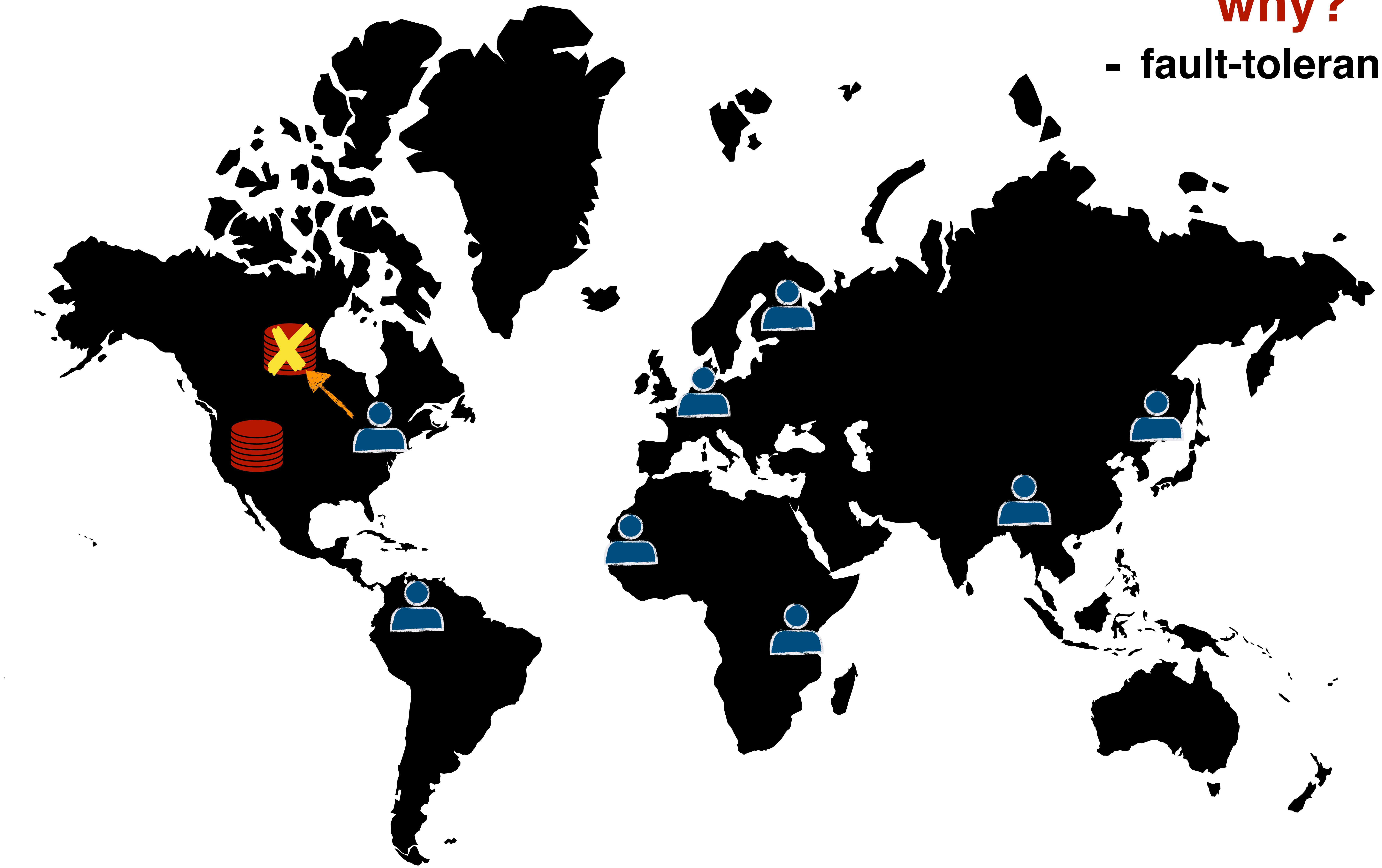
- **fault-tolerance**



planet-scale replicated systems

why?

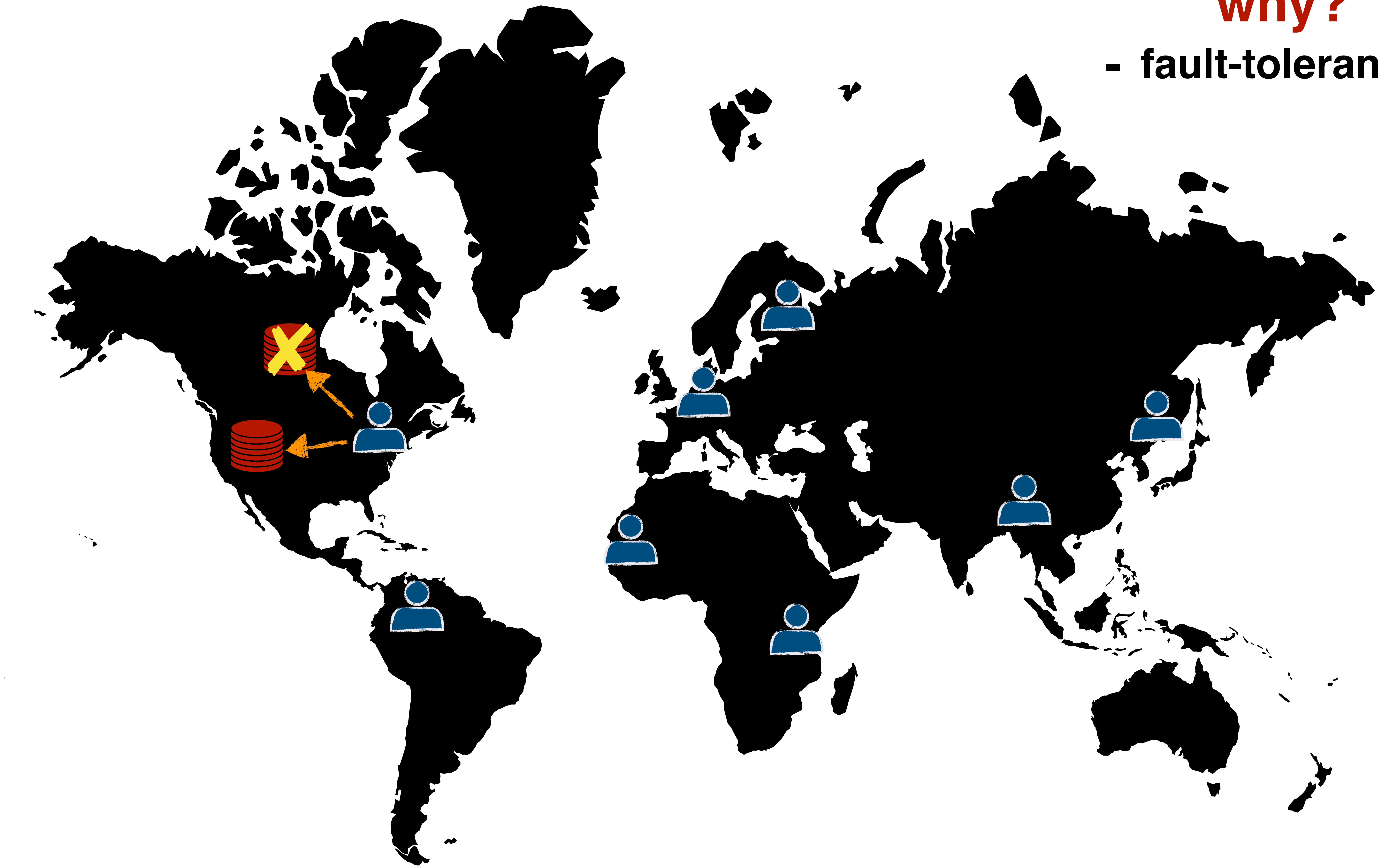
- fault-tolerance



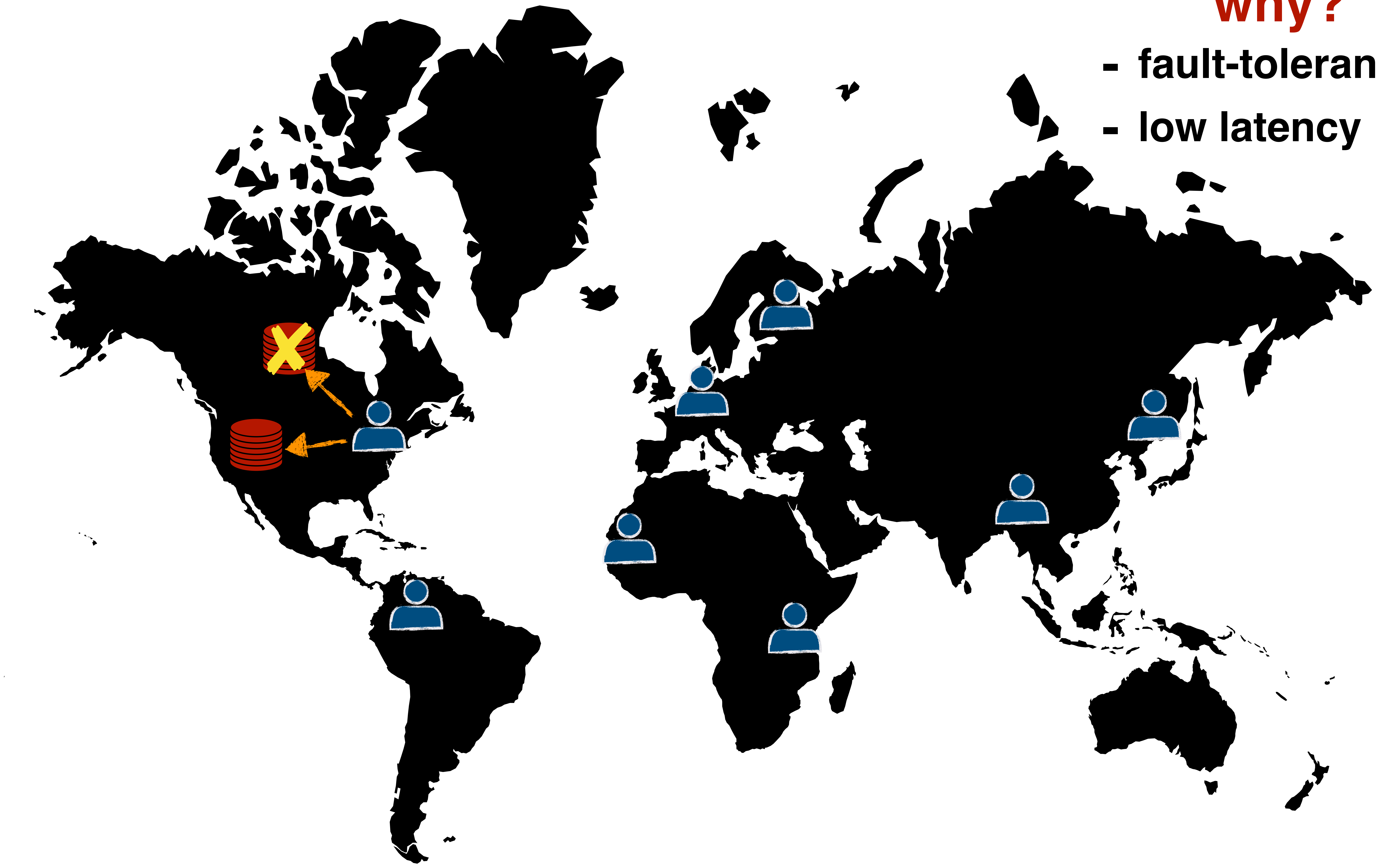
planet-scale replicated systems

why?

- **fault-tolerance**



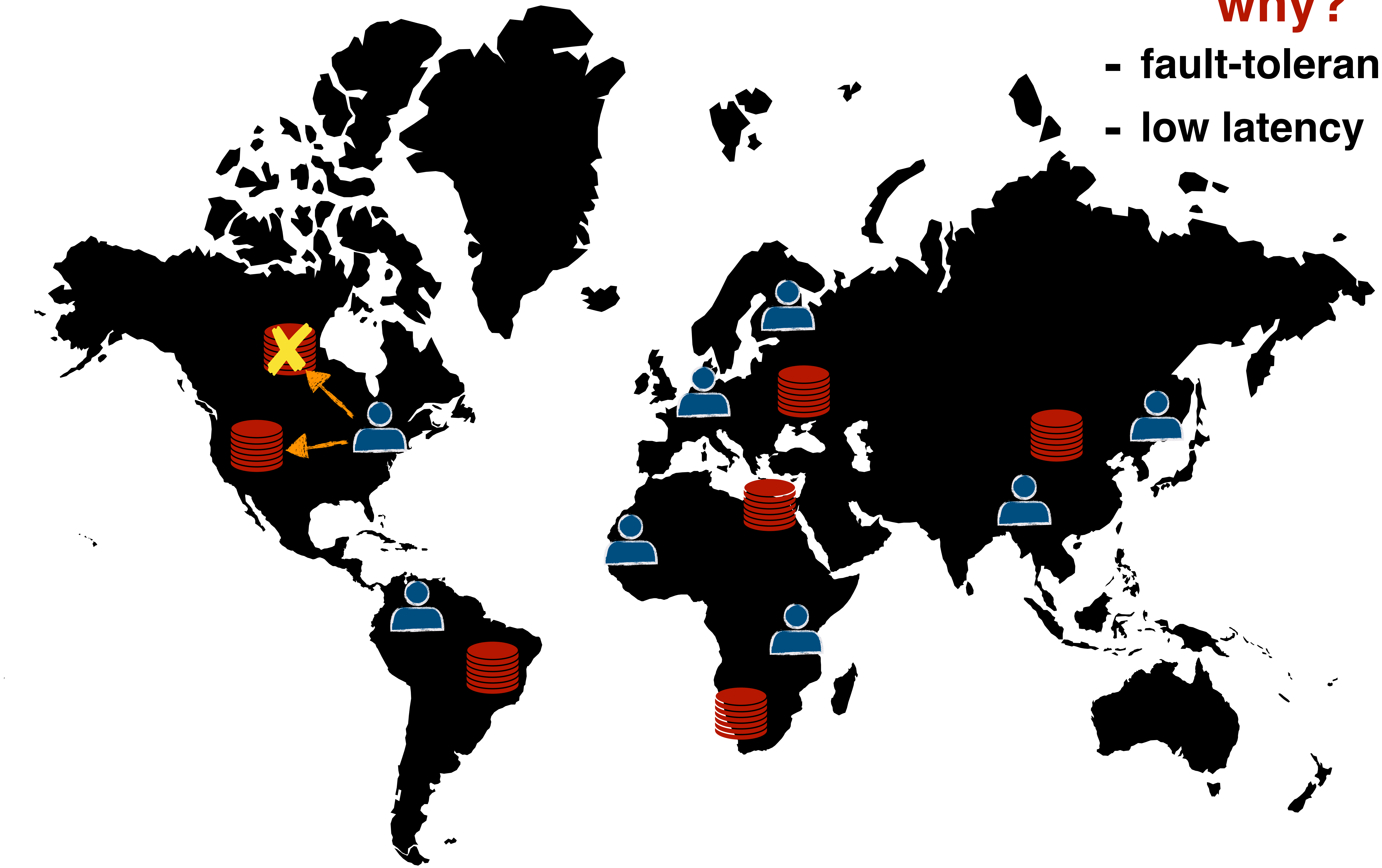
planet-scale replicated systems



why?

- fault-tolerance
- low latency

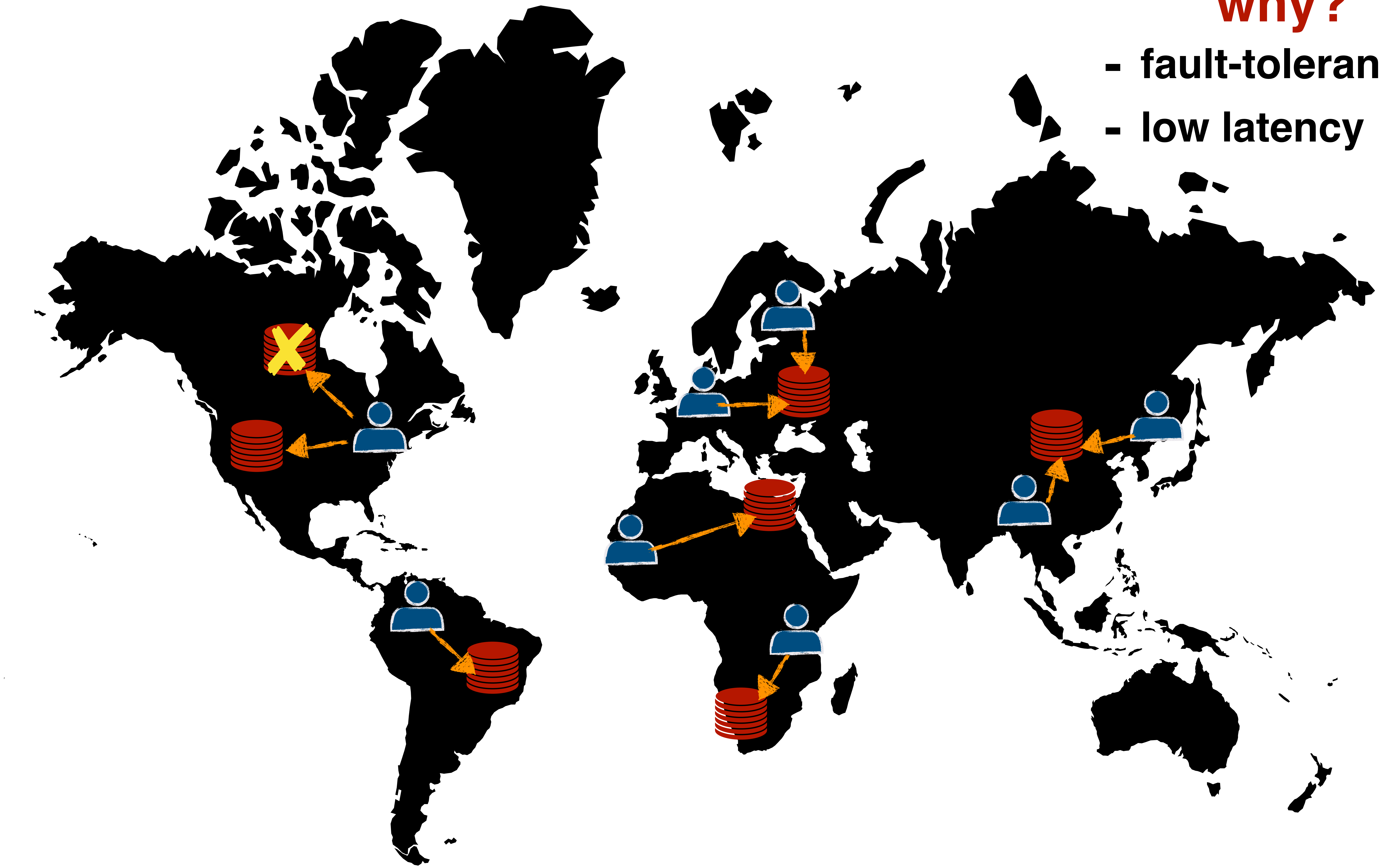
planet-scale replicated systems



why?

- fault-tolerance
- low latency

planet-scale replicated systems



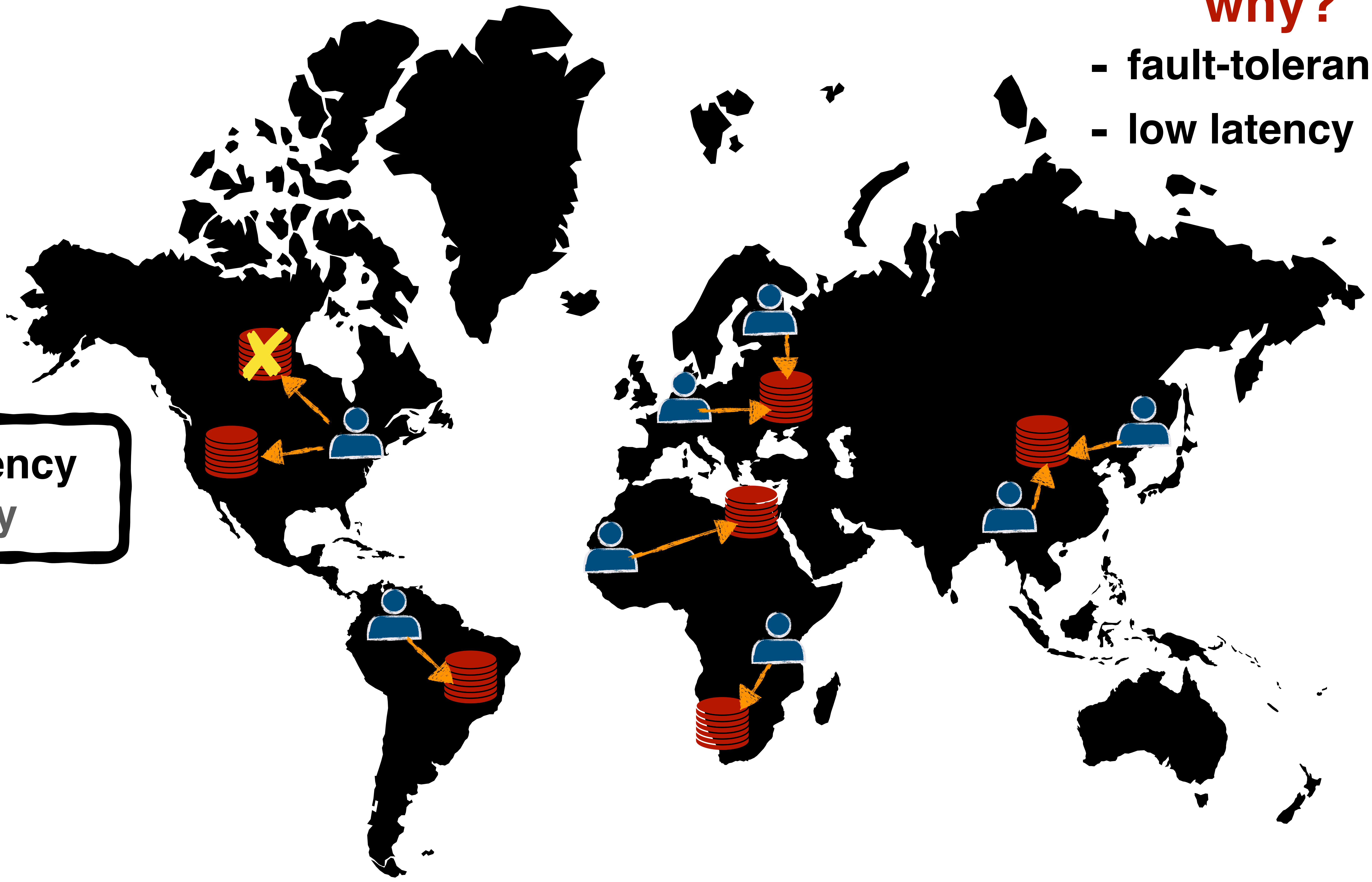
why?

- fault-tolerance
- low latency

planet-scale replicated systems

why?

- fault-tolerance
- low latency



strong consistency
linearizability

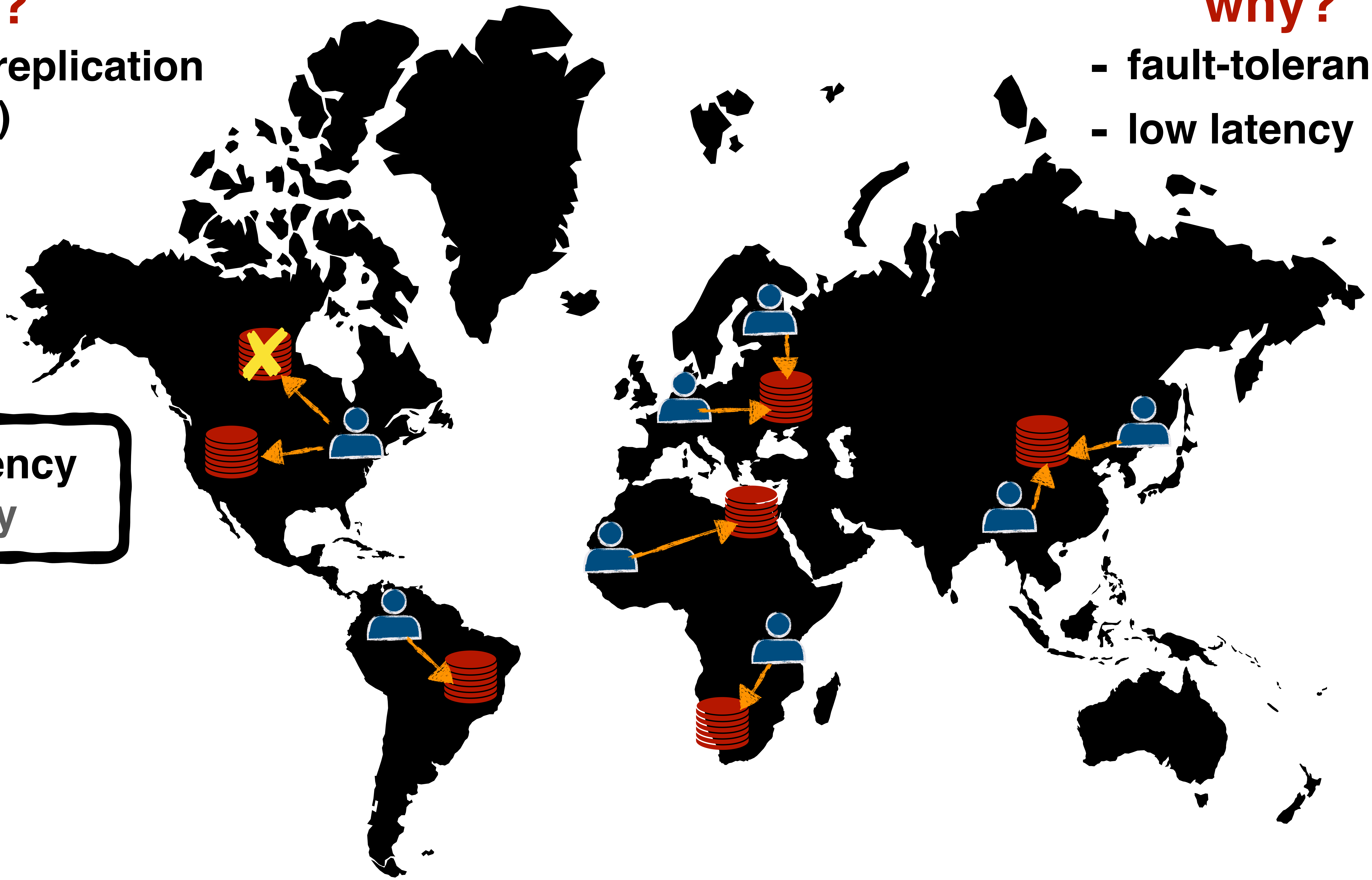
planet-scale replicated systems

how?

state-machine replication (SMR)

why?

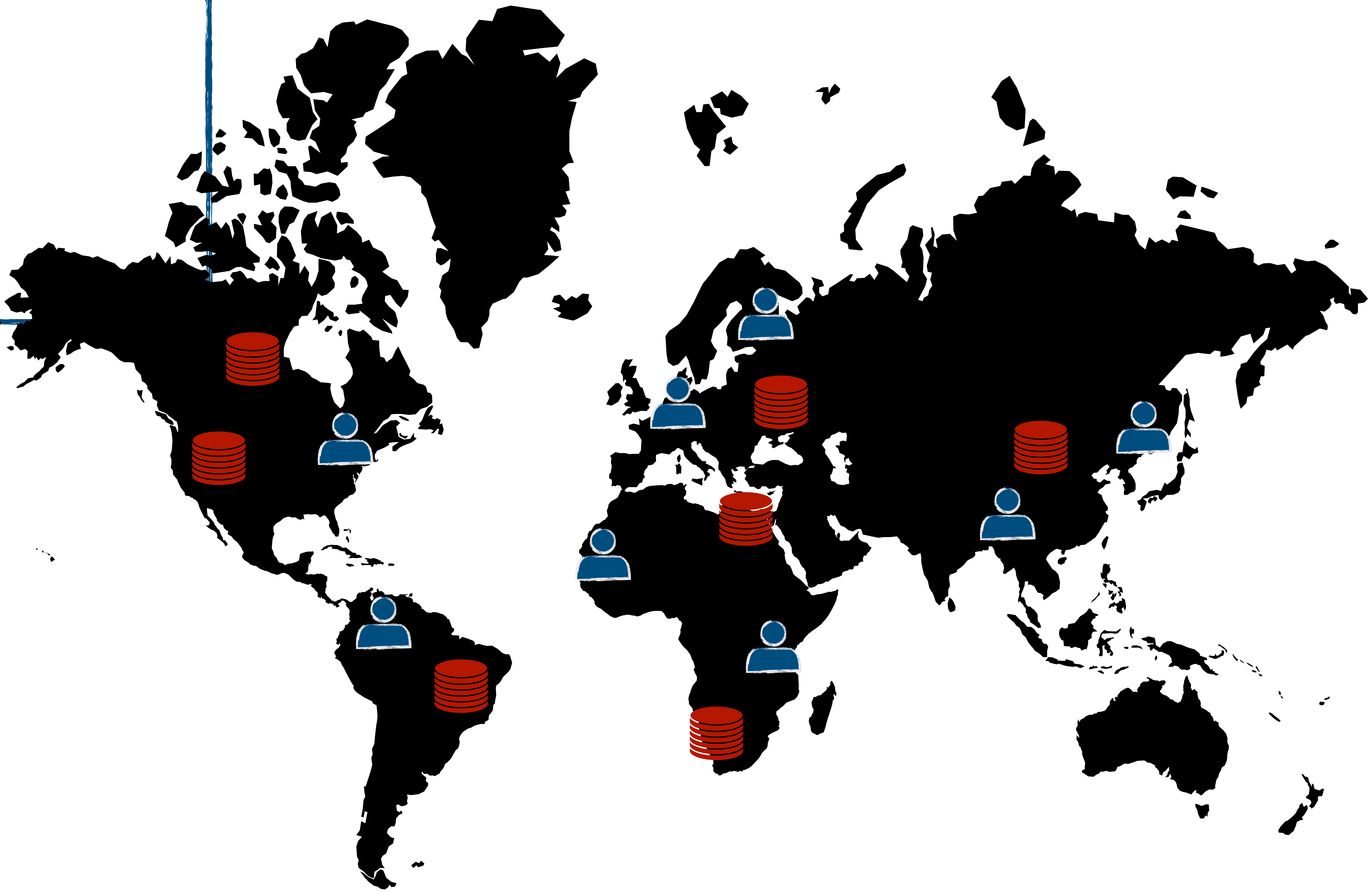
- fault-tolerance
- low latency



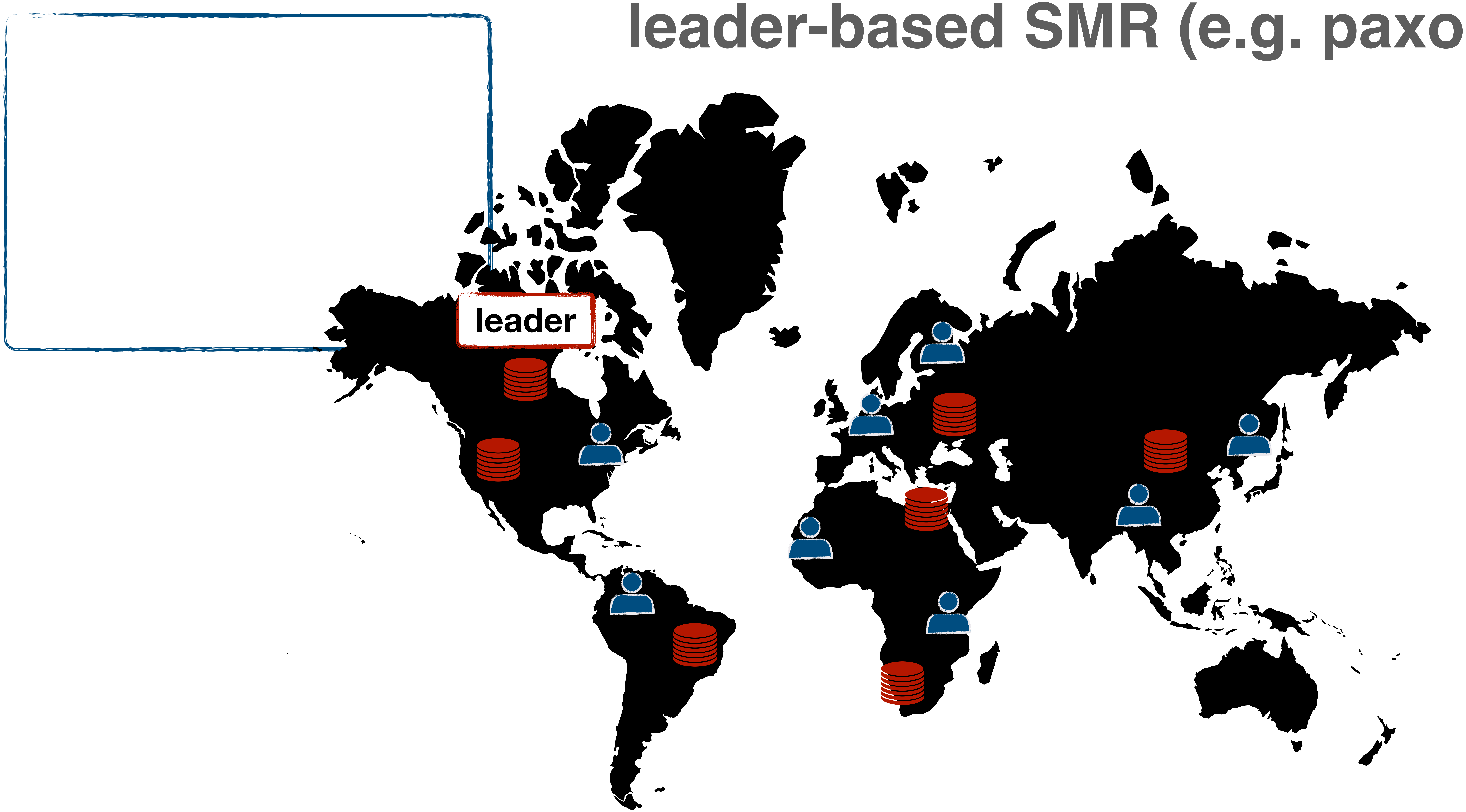
strong consistency

linearizability

leader-based SMR (e.g. paxos)

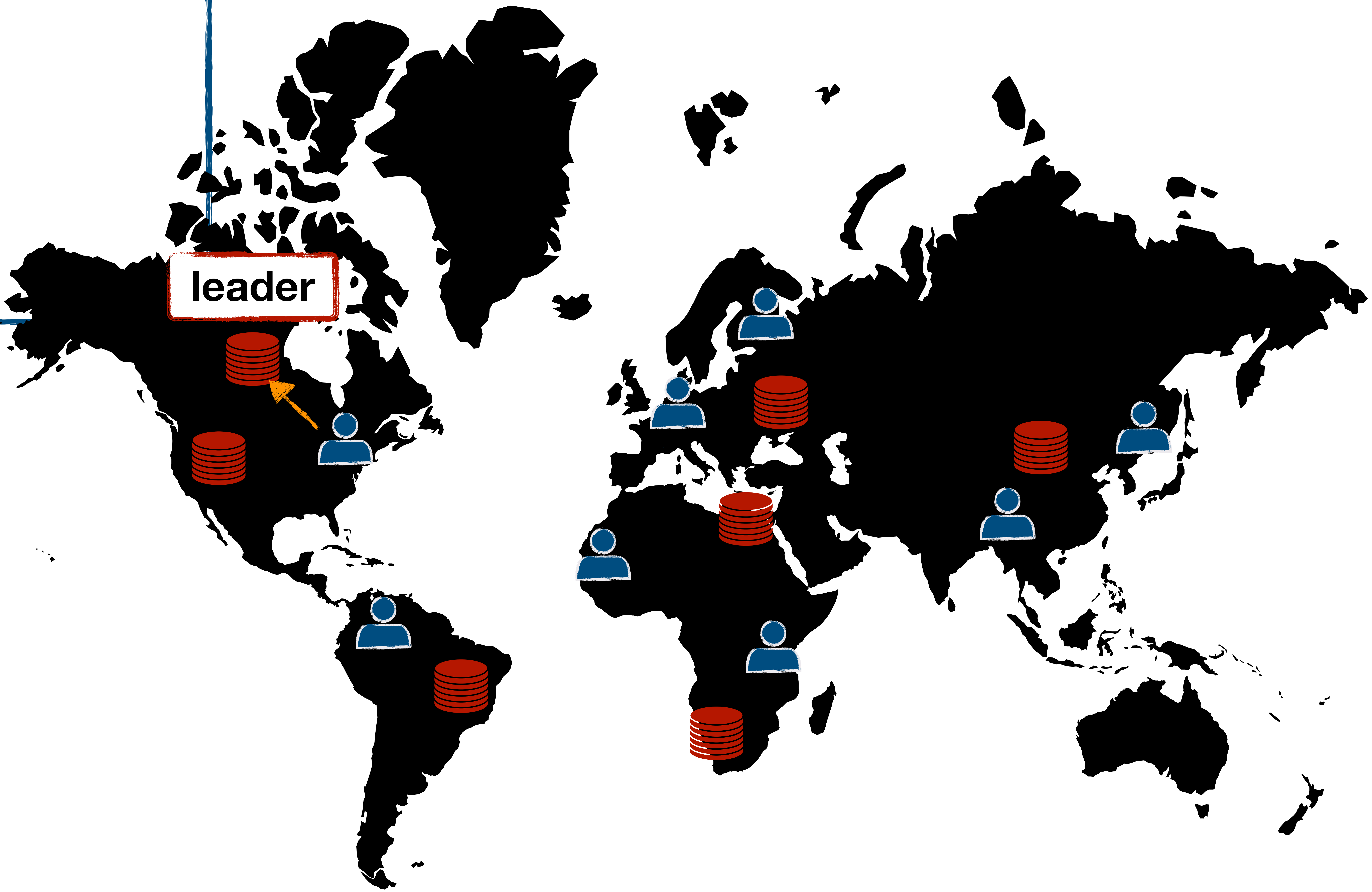


leader-based SMR (e.g. paxos)



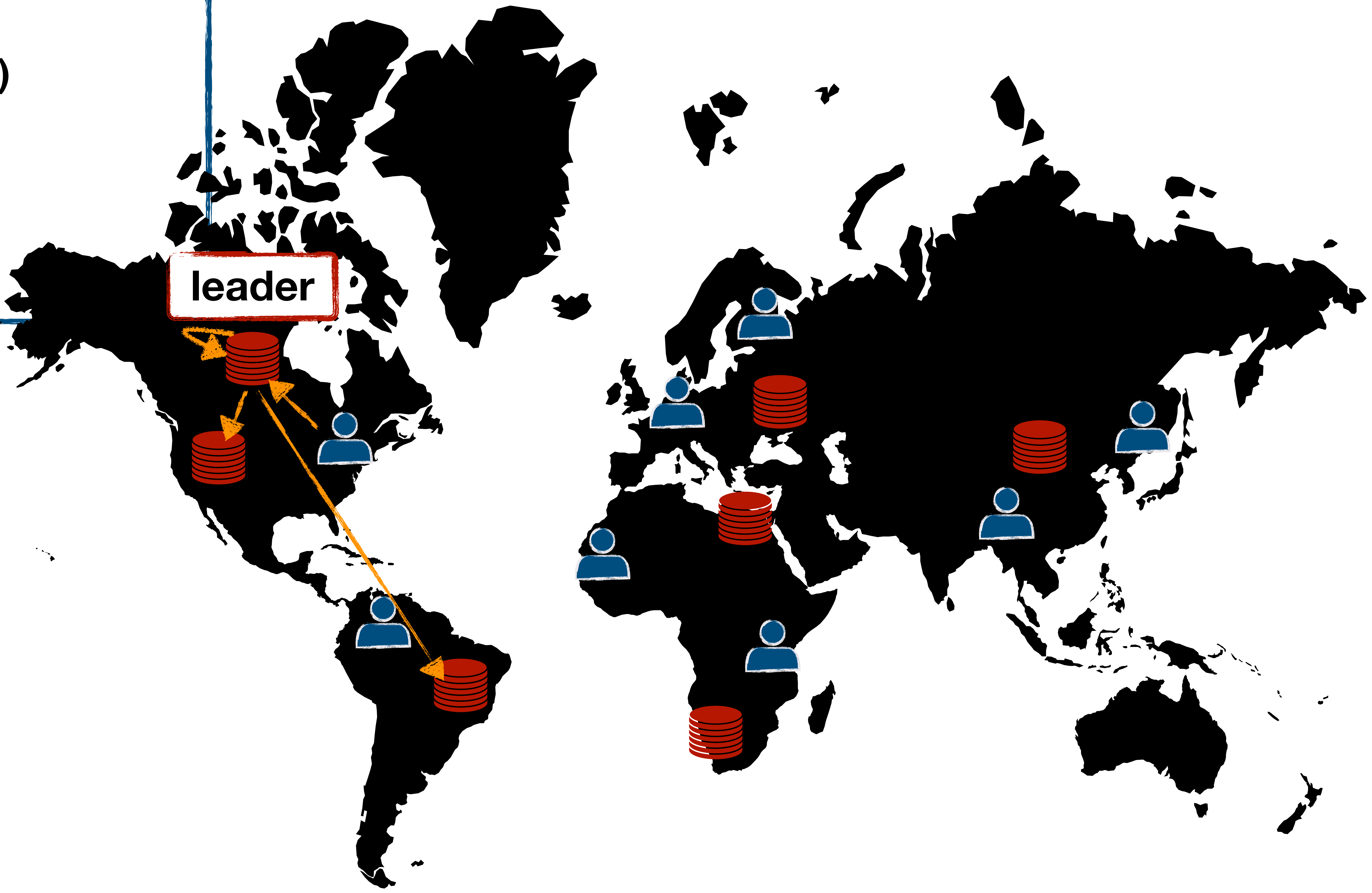
leader-based SMR (e.g. paxos)

- leader receives command



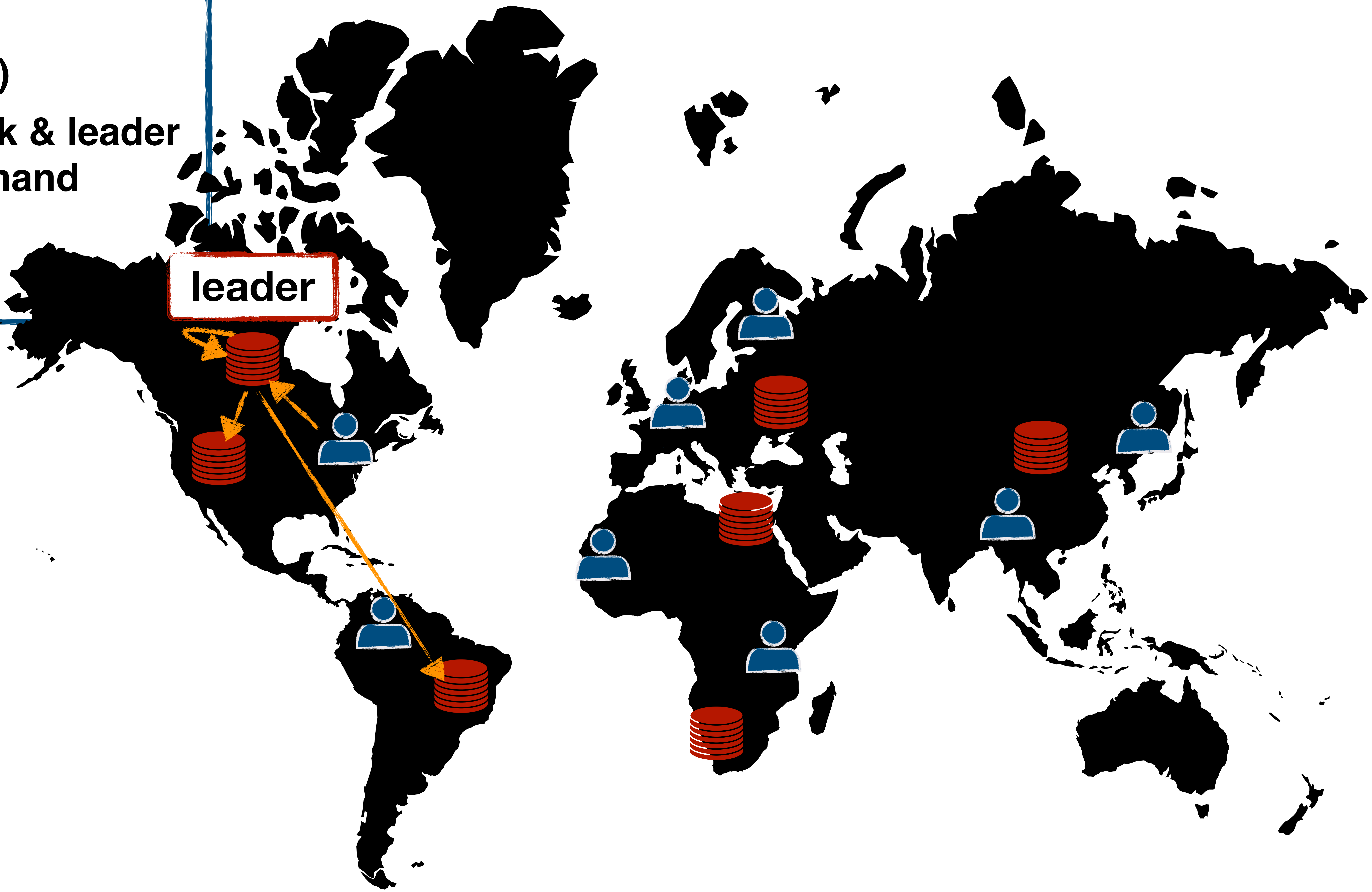
leader-based SMR (e.g. paxos)

- leader receives command
- forwards it to f+1 acceptors (say f=2)



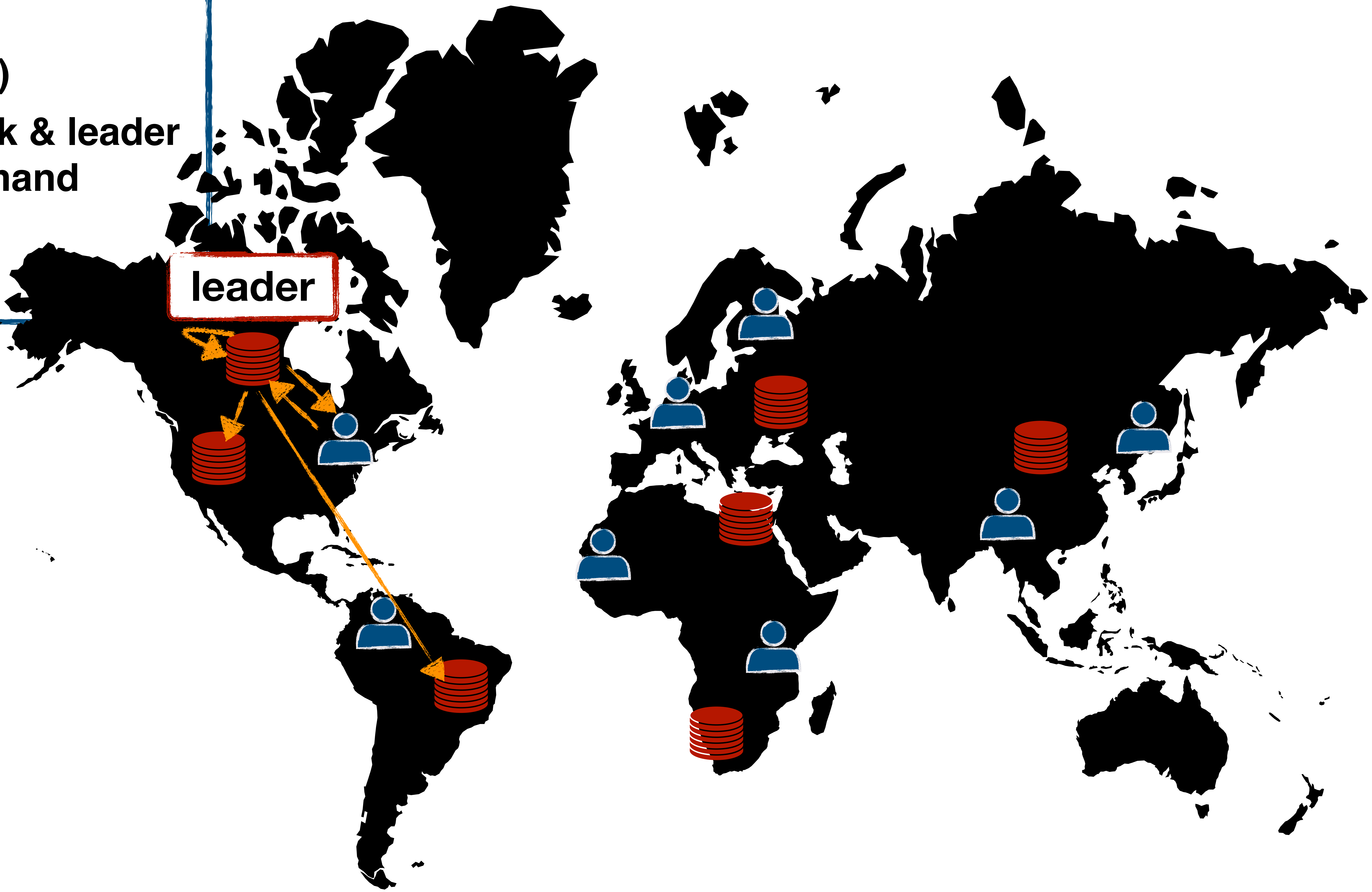
leader-based SMR (e.g. paxos)

- leader receives command
- forwards it to f+1 acceptors (say f=2)
- acceptors send ack & leader commits the command



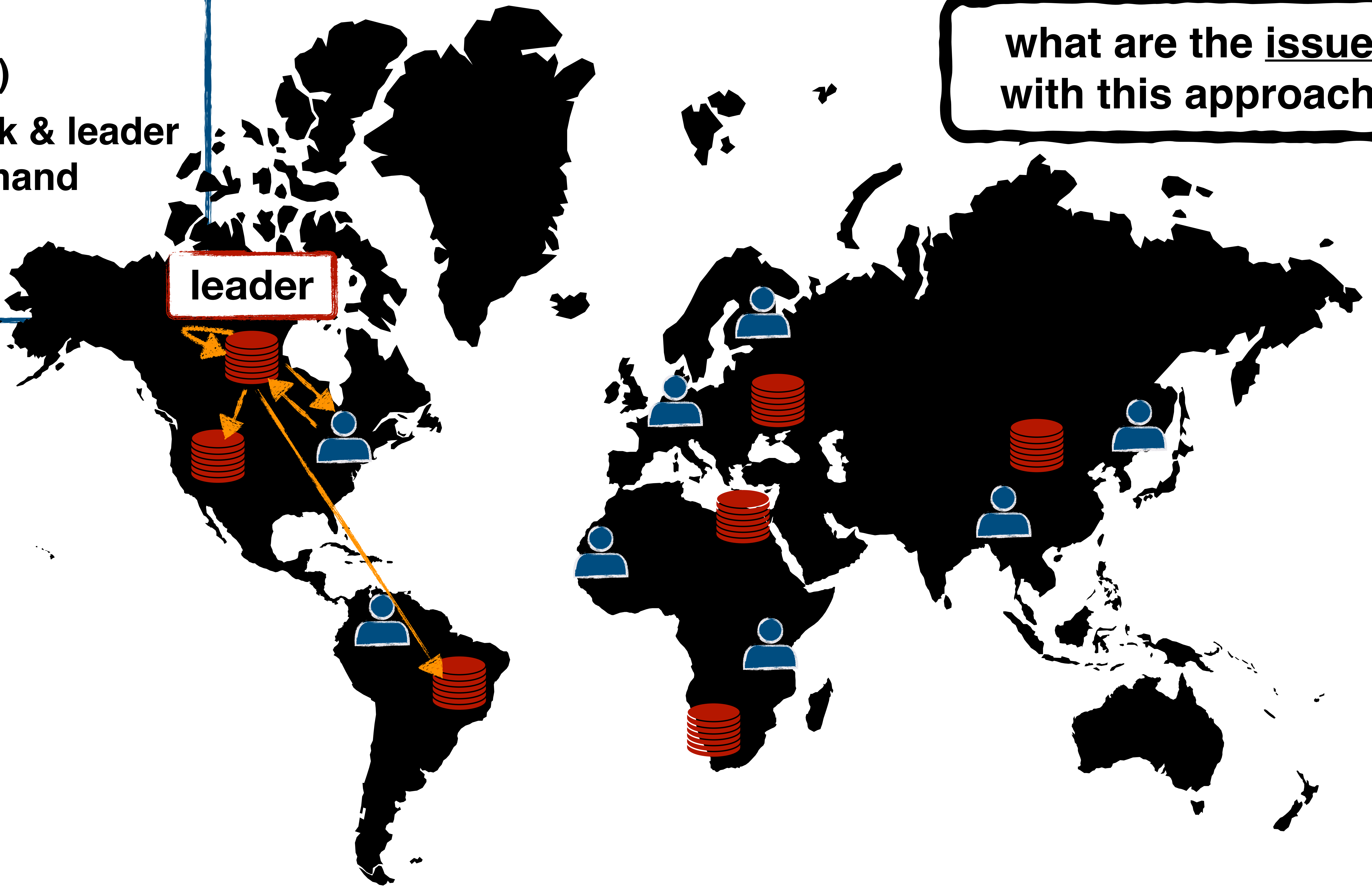
leader-based SMR (e.g. paxos)

- leader receives command
- forwards it to f+1 acceptors (say f=2)
- acceptors send ack & leader commits the command
- leader sends result to client



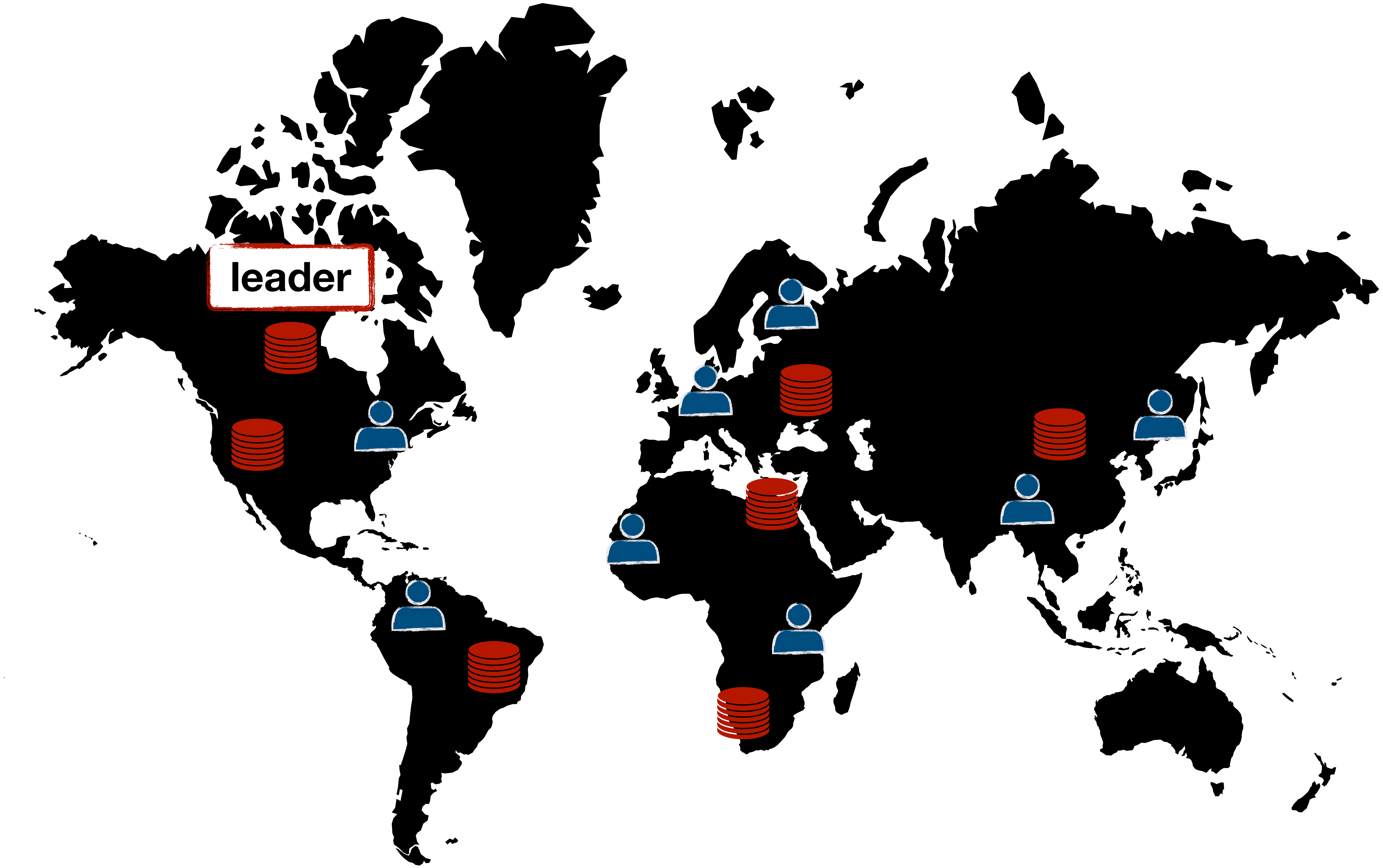
leader-based SMR (e.g. paxos)

- leader receives command
- forwards it to f+1 acceptors (say f=2)
- acceptors send ack & leader commits the command
- leader sends result to client

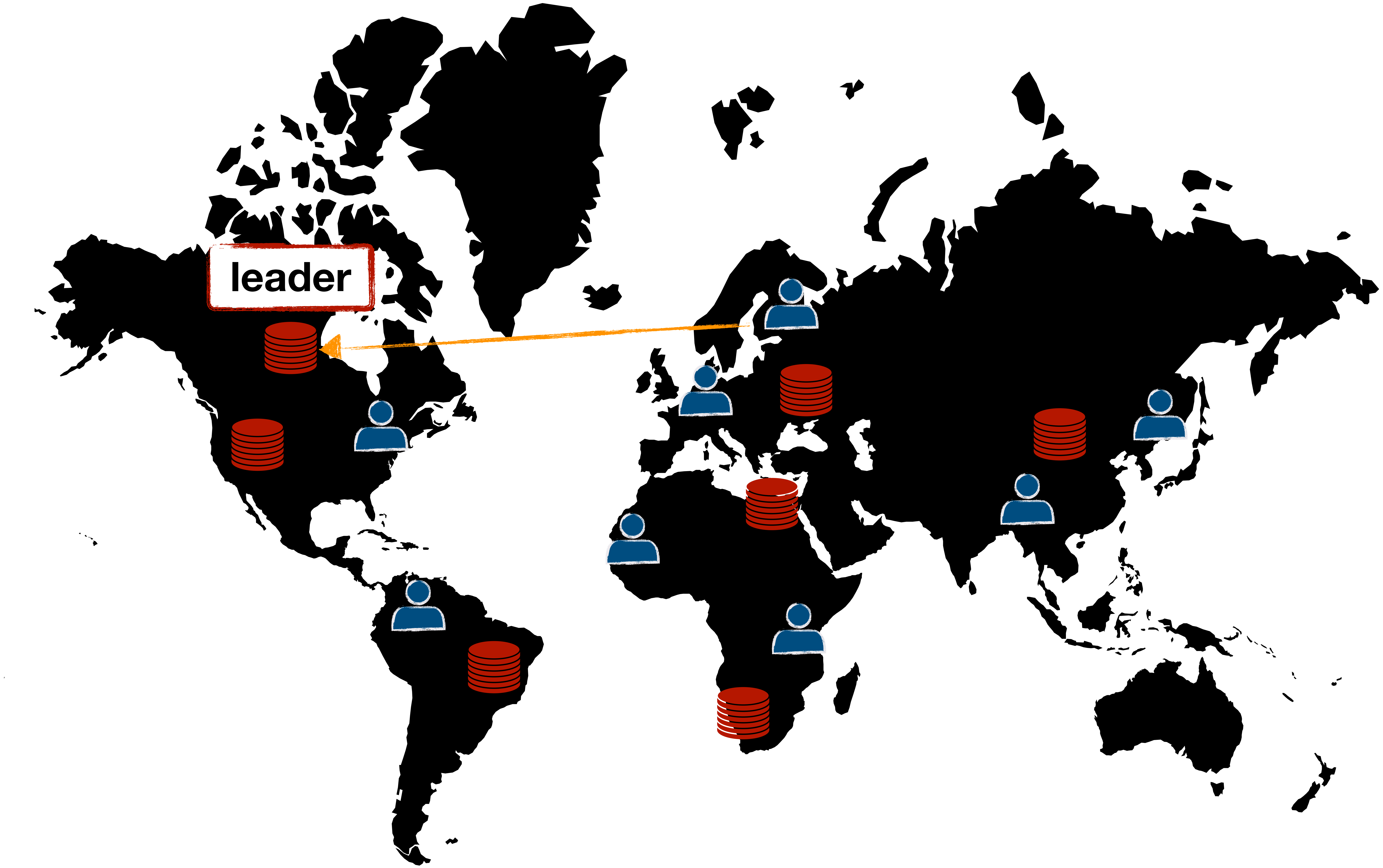


what are the issues with this approach?

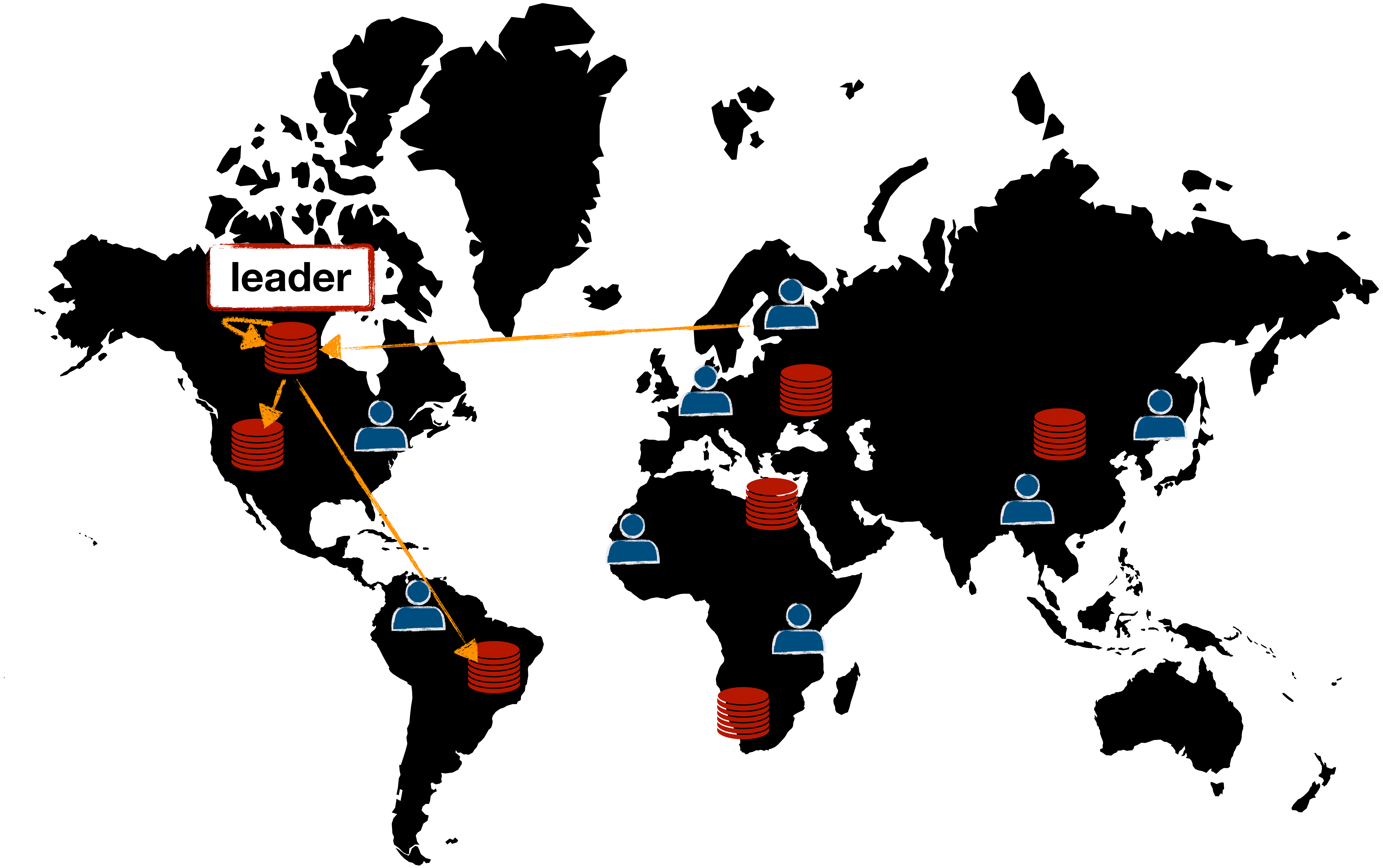
leader-based SMR pitfalls



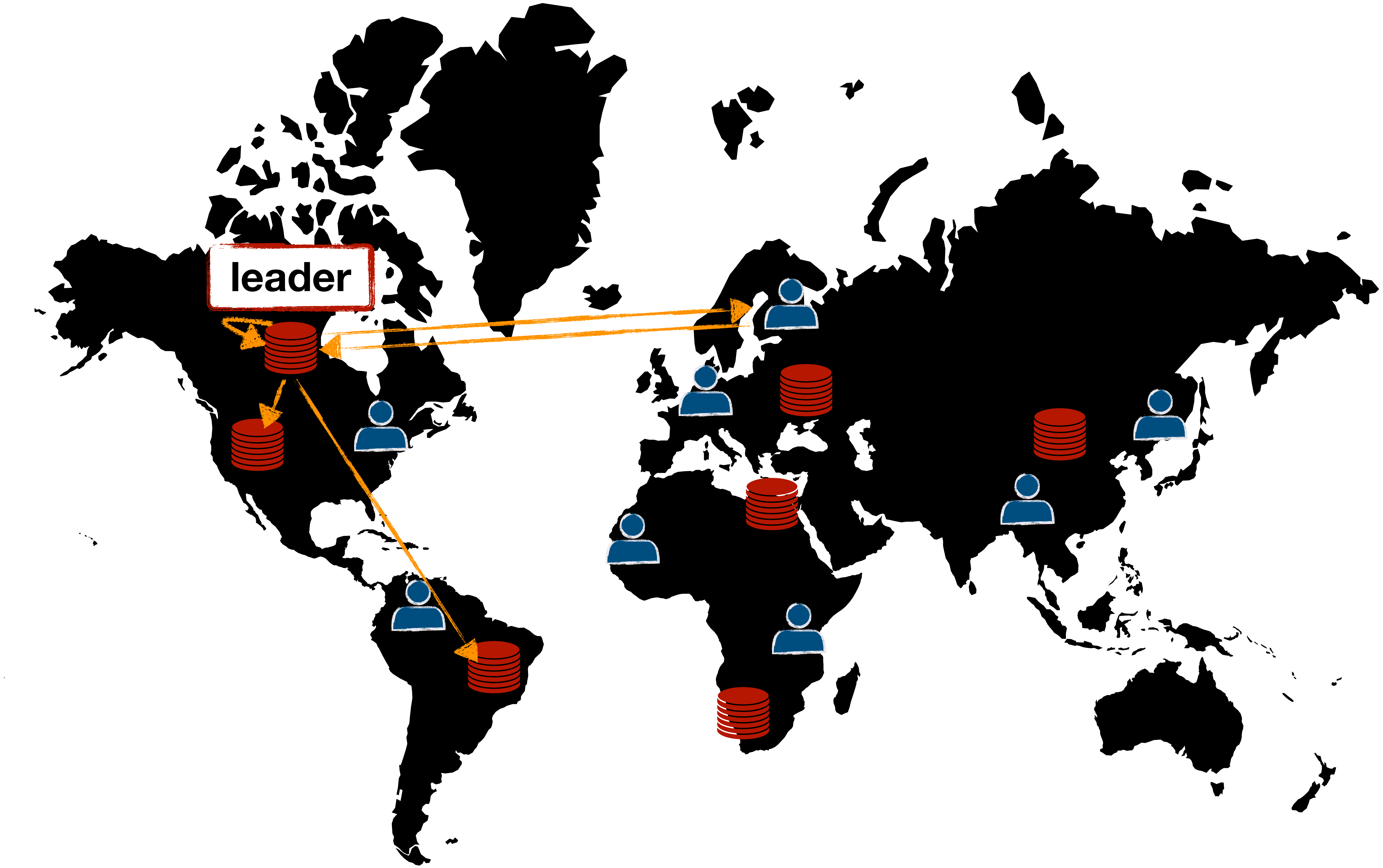
leader-based SMR pitfalls



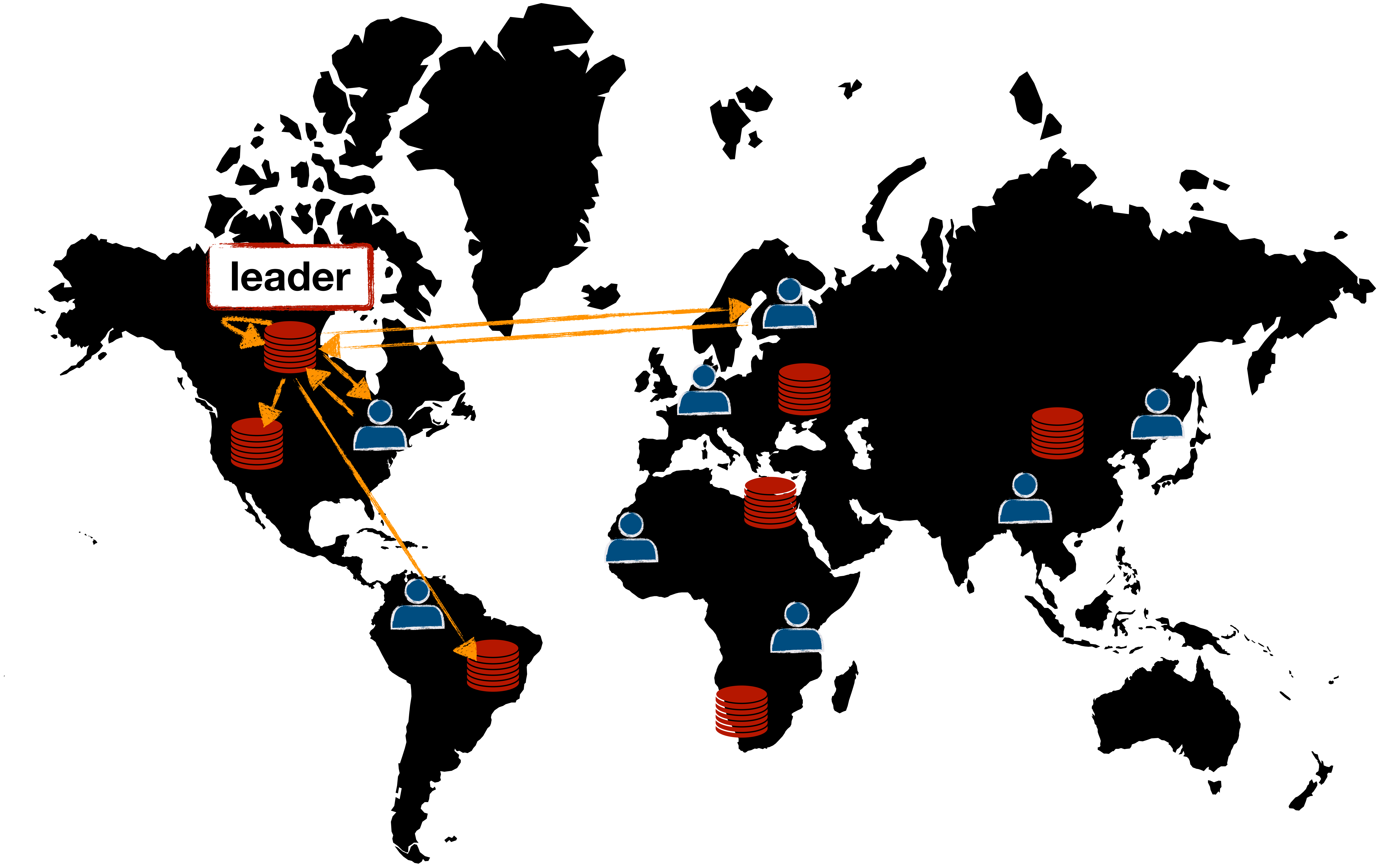
leader-based SMR pitfalls



leader-based SMR pitfalls

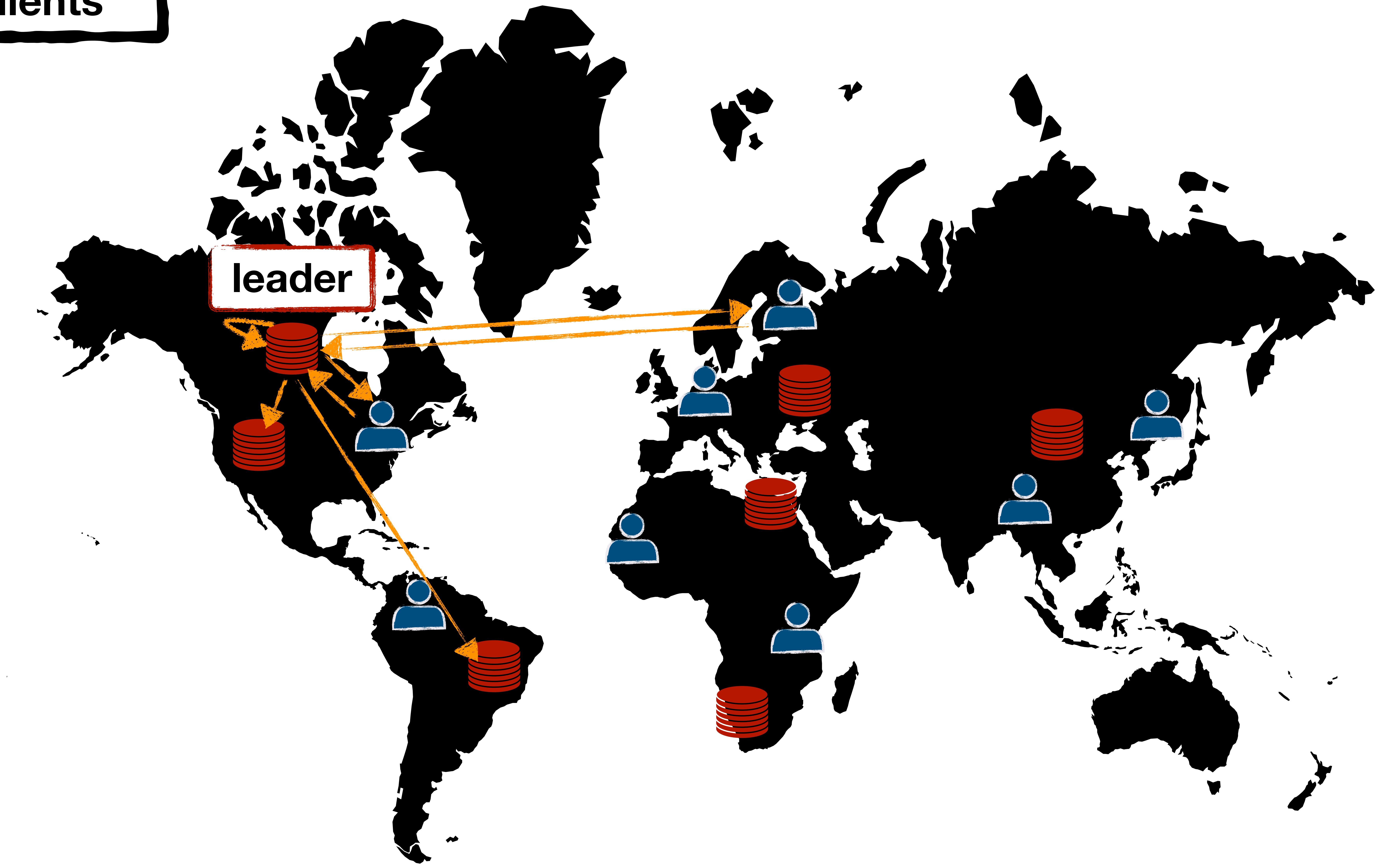


leader-based SMR pitfalls



leader-based SMR pitfalls

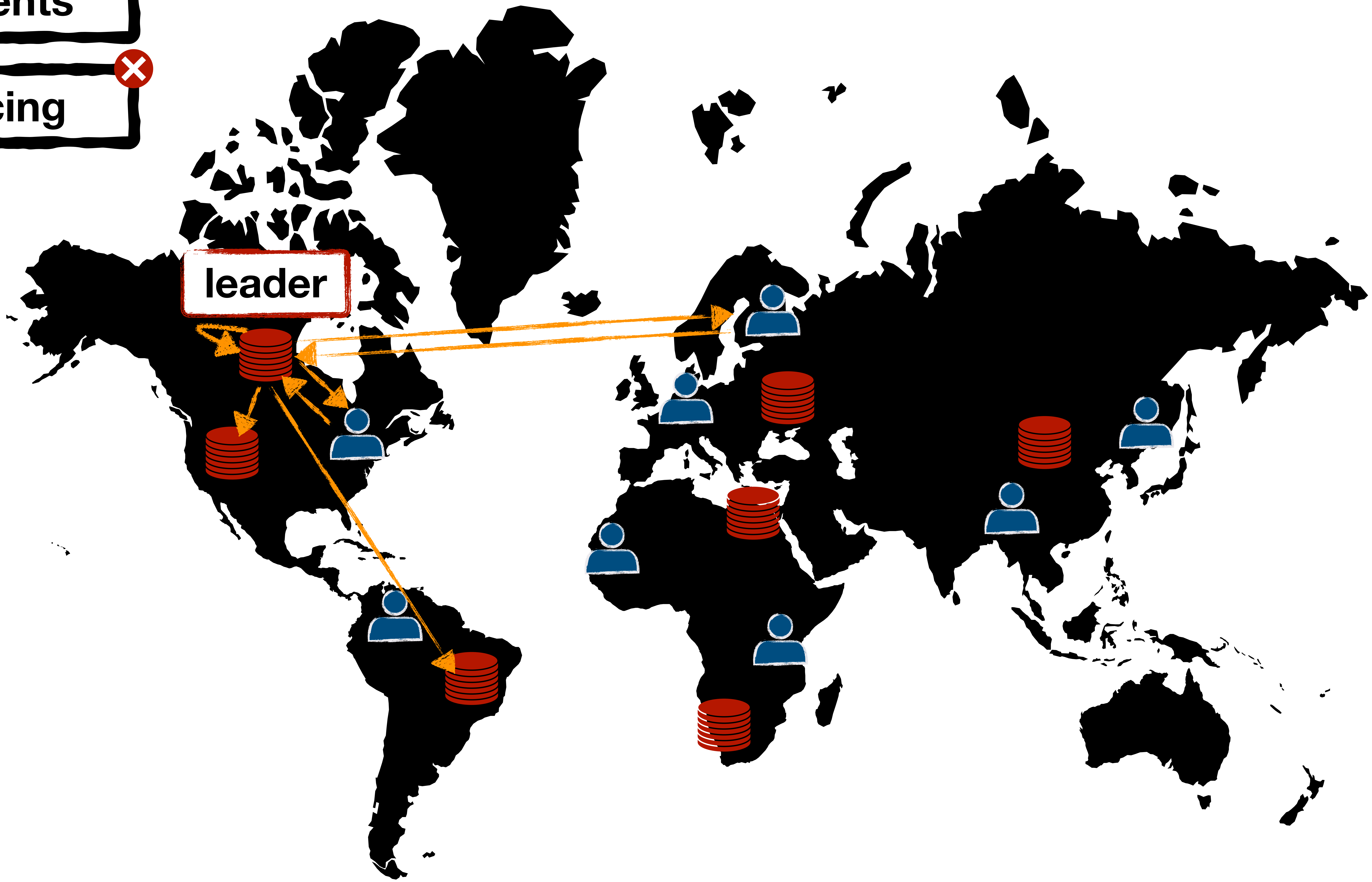
unfairness/high latency for faraway clients



leader-based SMR pitfalls

unfairness/high latency for faraway clients ☒

no load balancing ☒

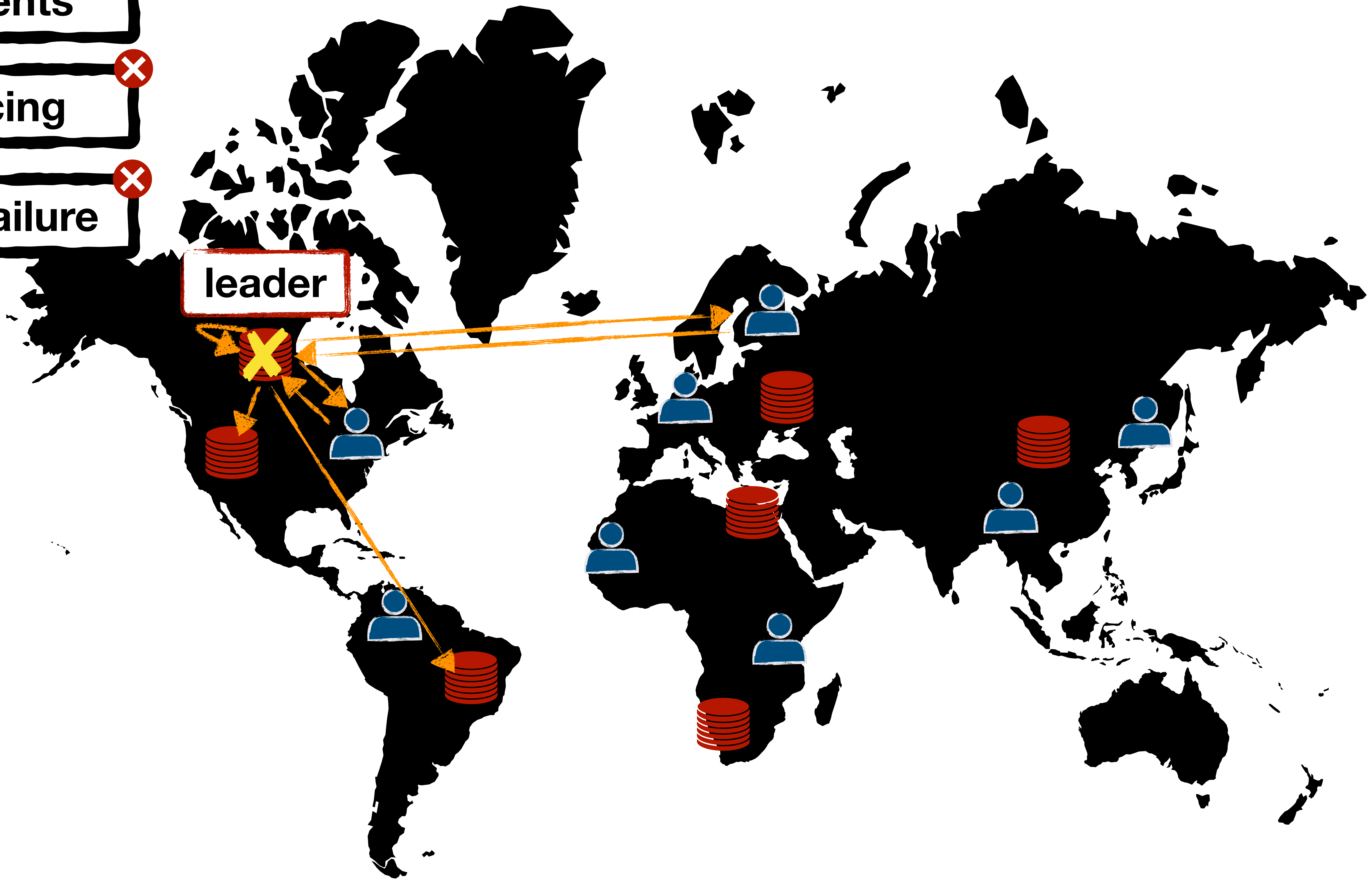


leader-based SMR pitfalls

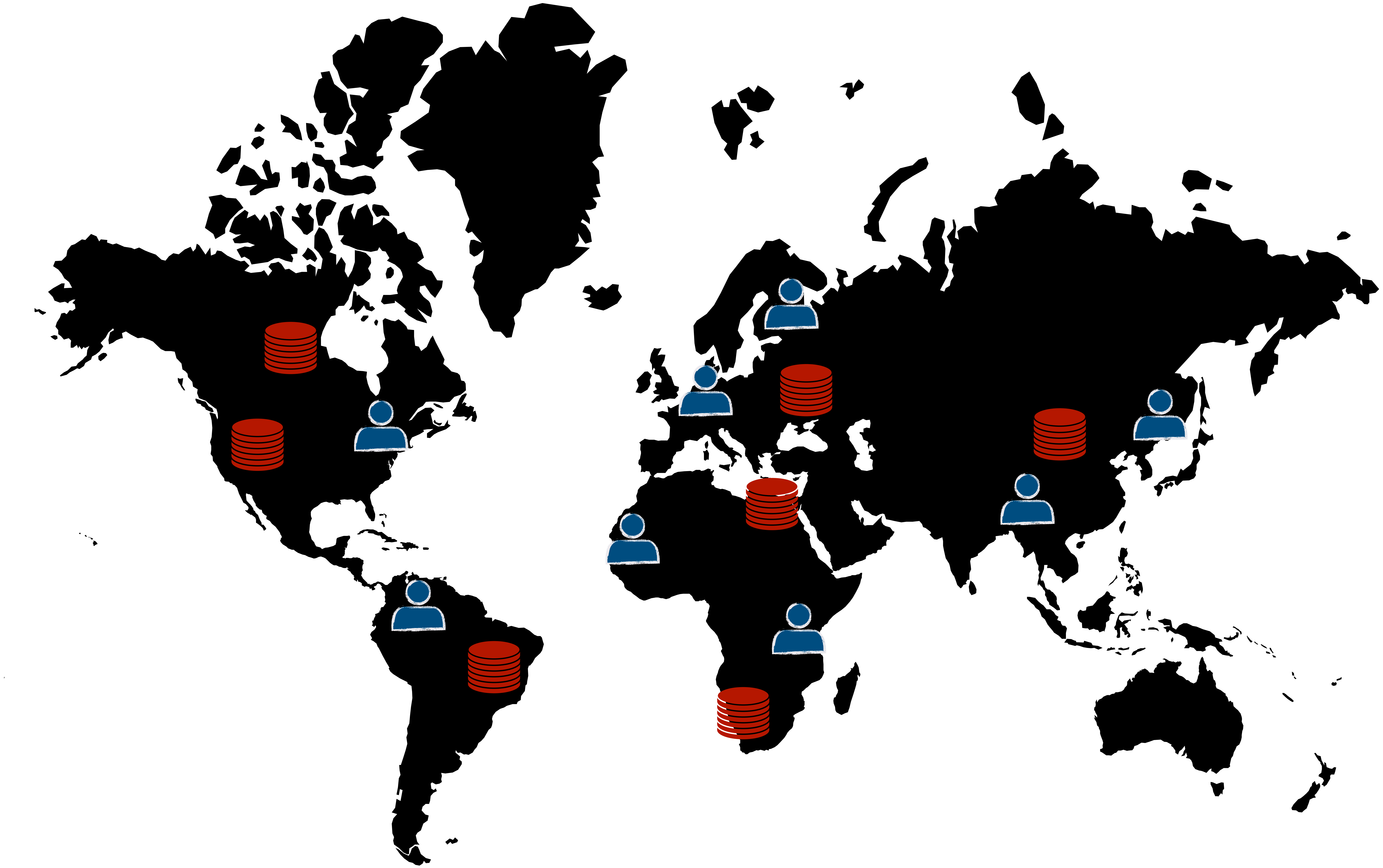
unfairness/high latency for faraway clients ☒

no load balancing ☒

single point of failure ☒

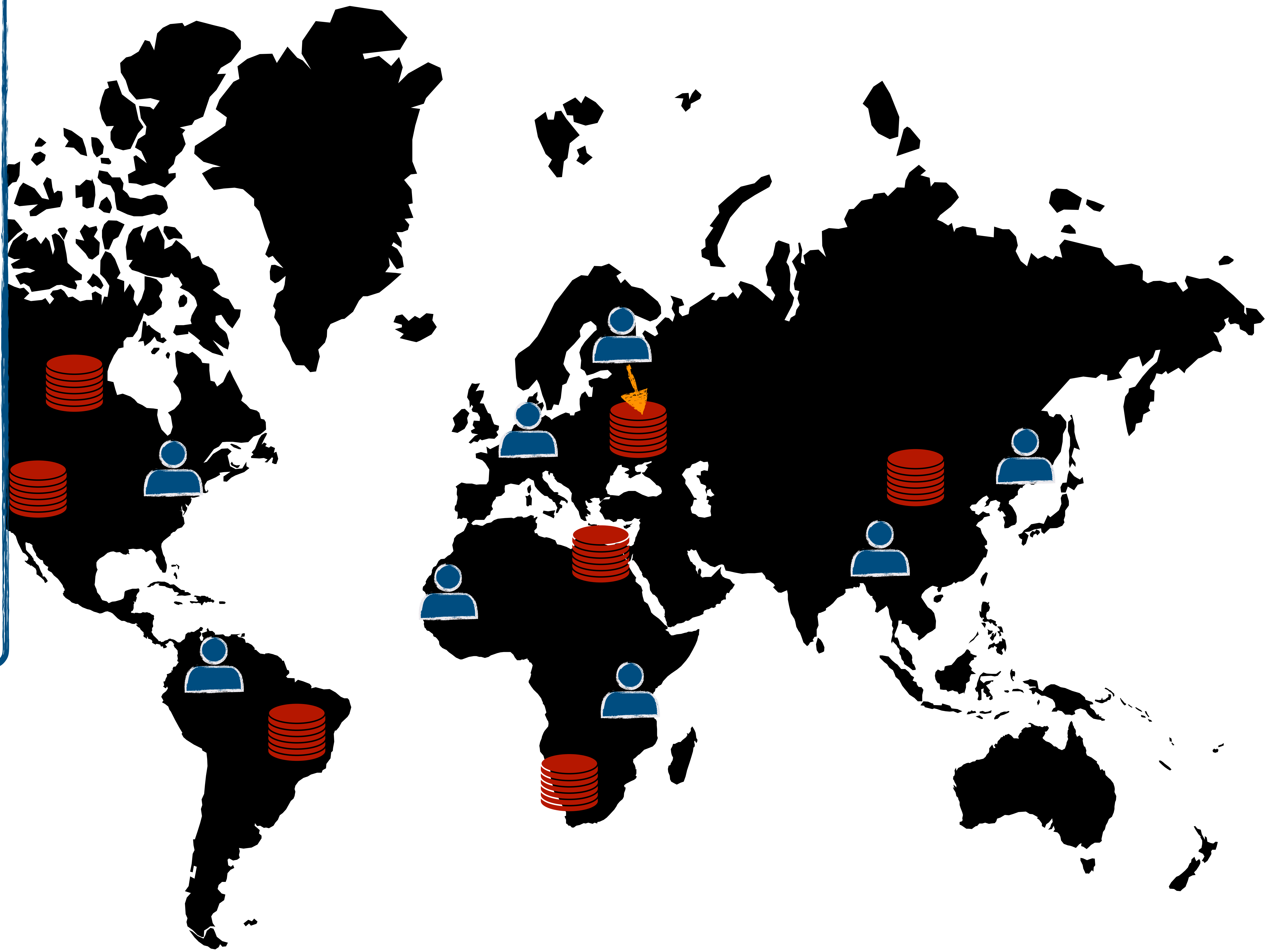


leaderless SMR (e.g. epaxos)



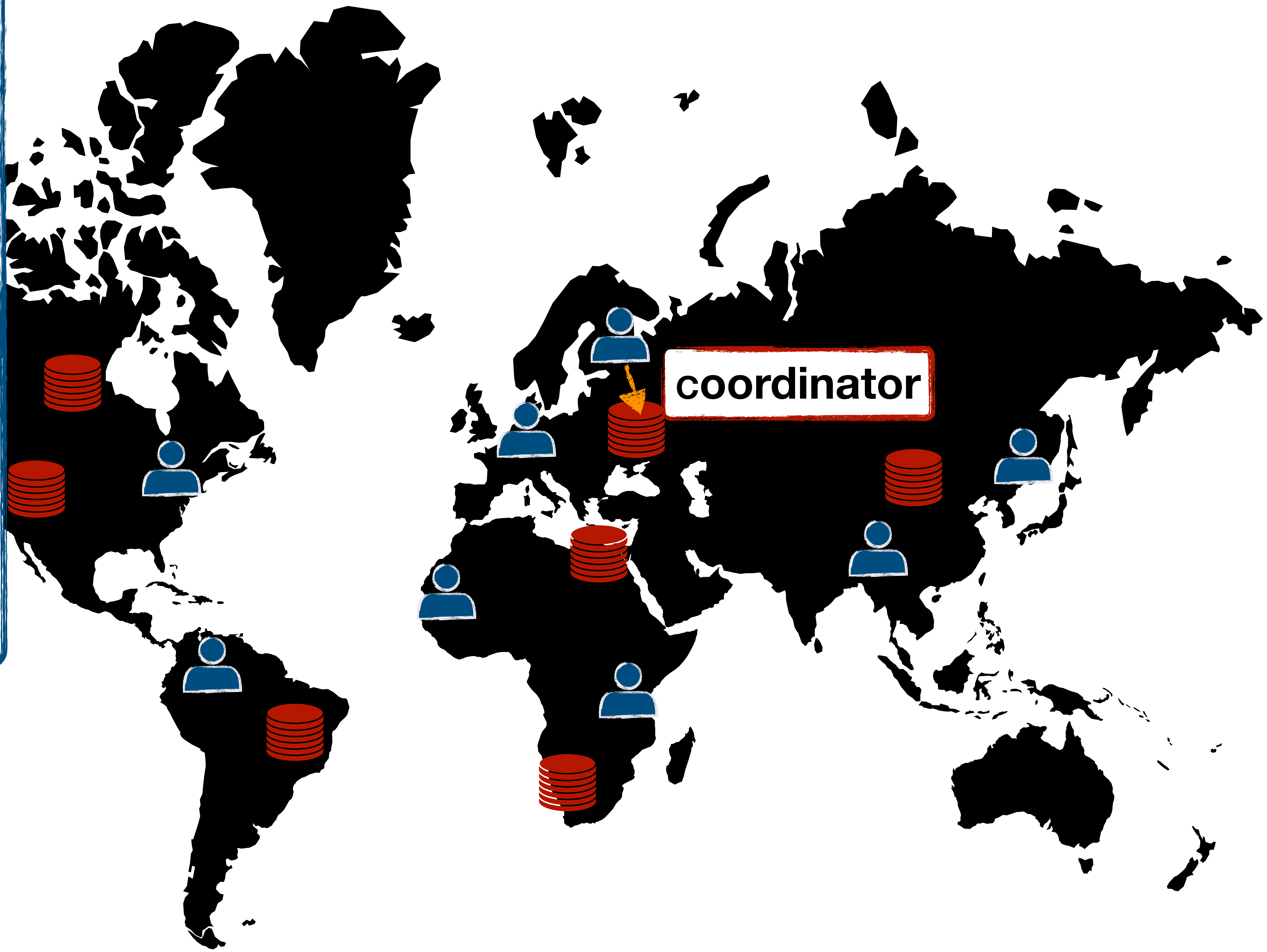
leaderless SMR (e.g. epaxos)

- replica R receives command



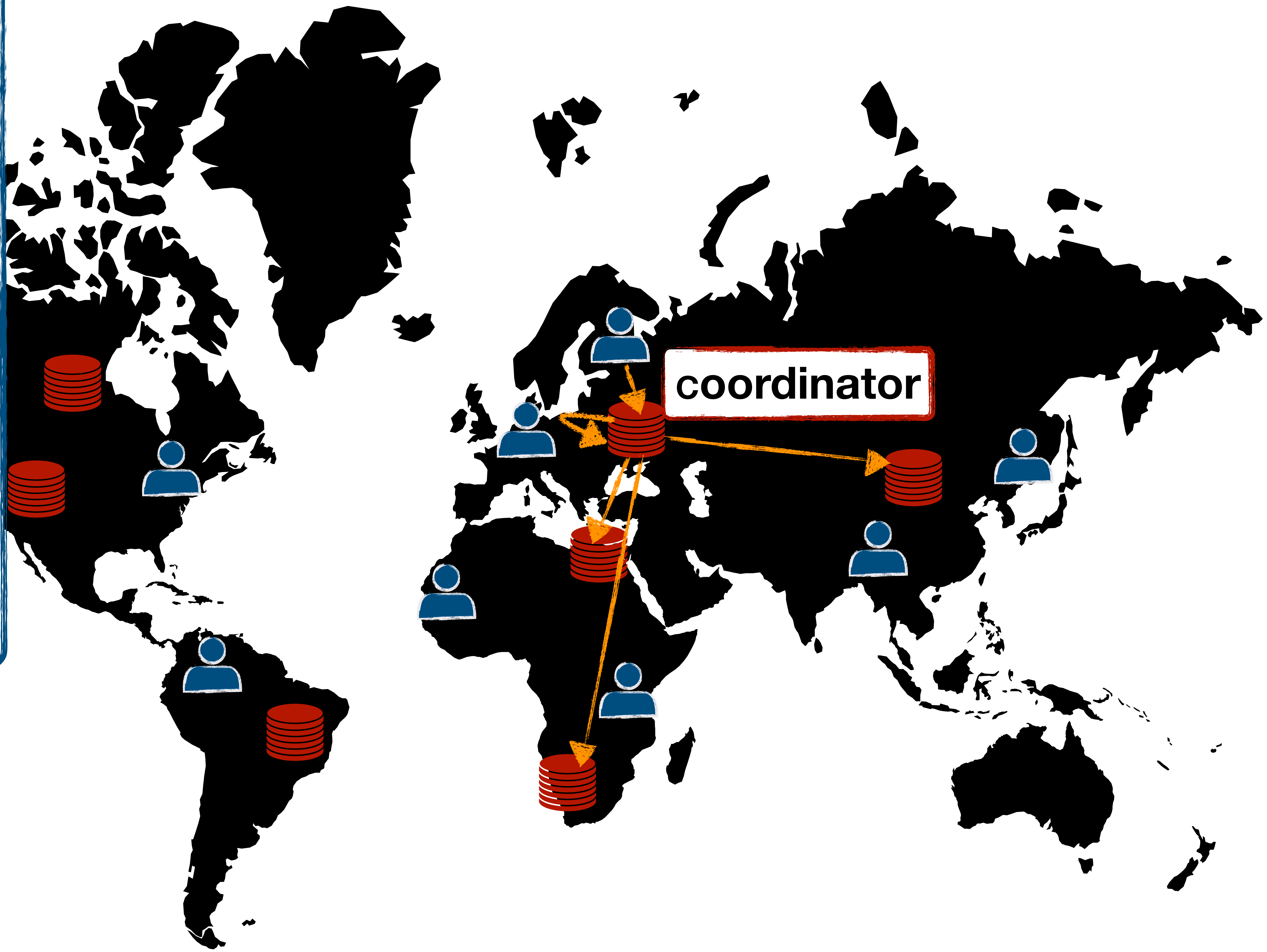
leaderless SMR (e.g. epaxos)

- replica R receives command



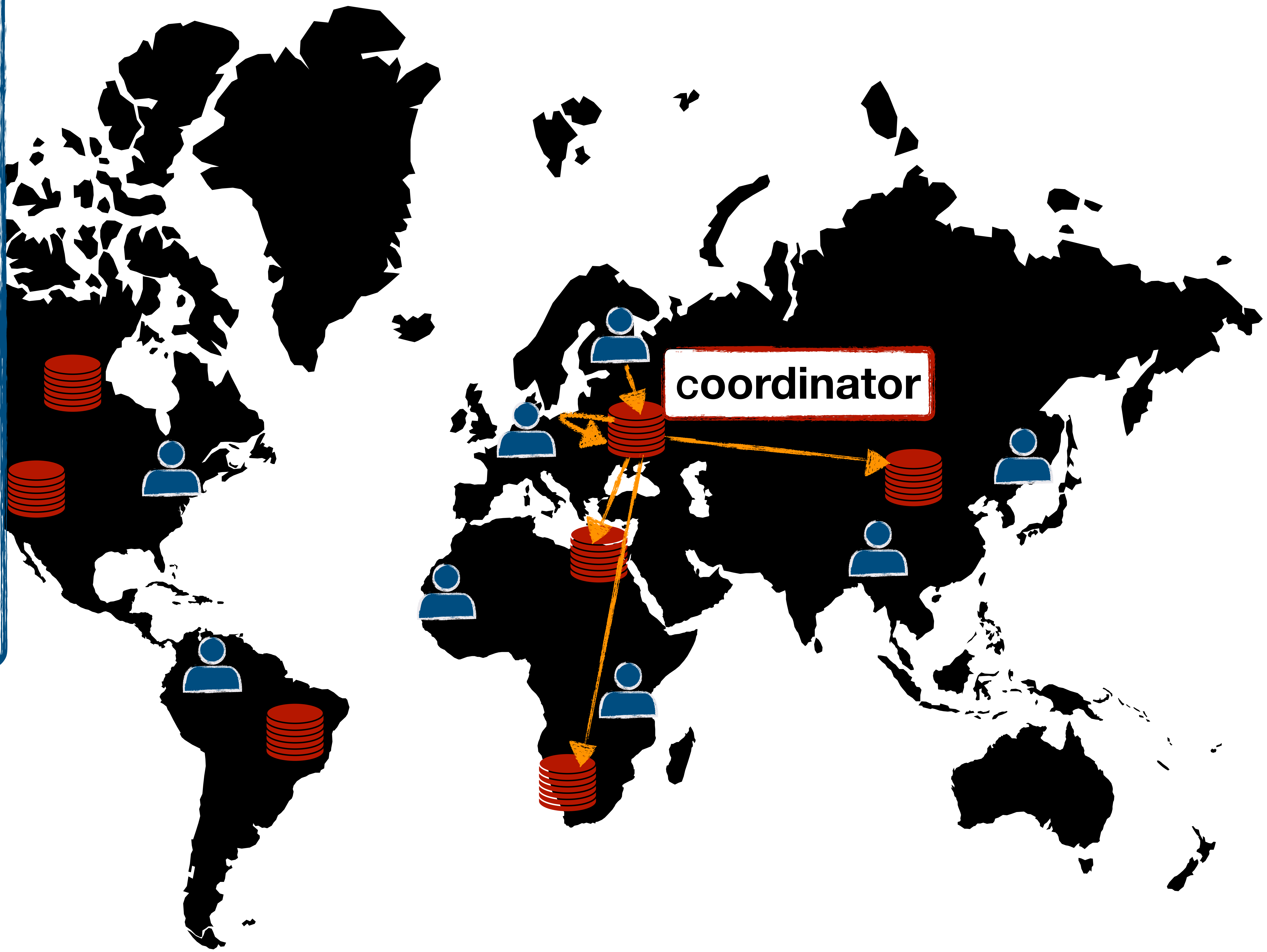
leaderless SMR (e.g. epaxos)

- replica R receives command
- forwards it to a quorum



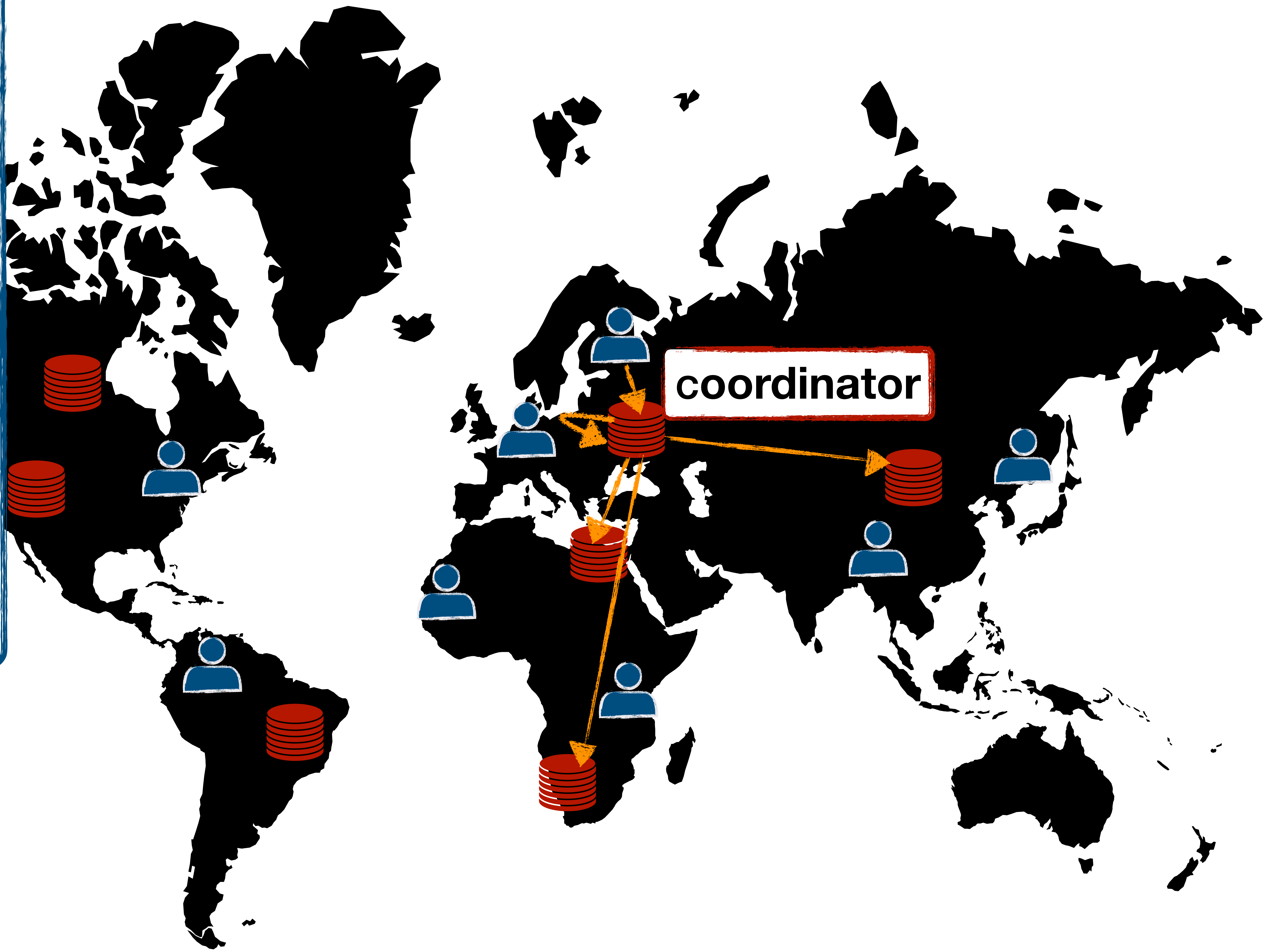
leaderless SMR (e.g. epaxos)

- replica R receives command
- forwards it to a quorum
- quorum replies



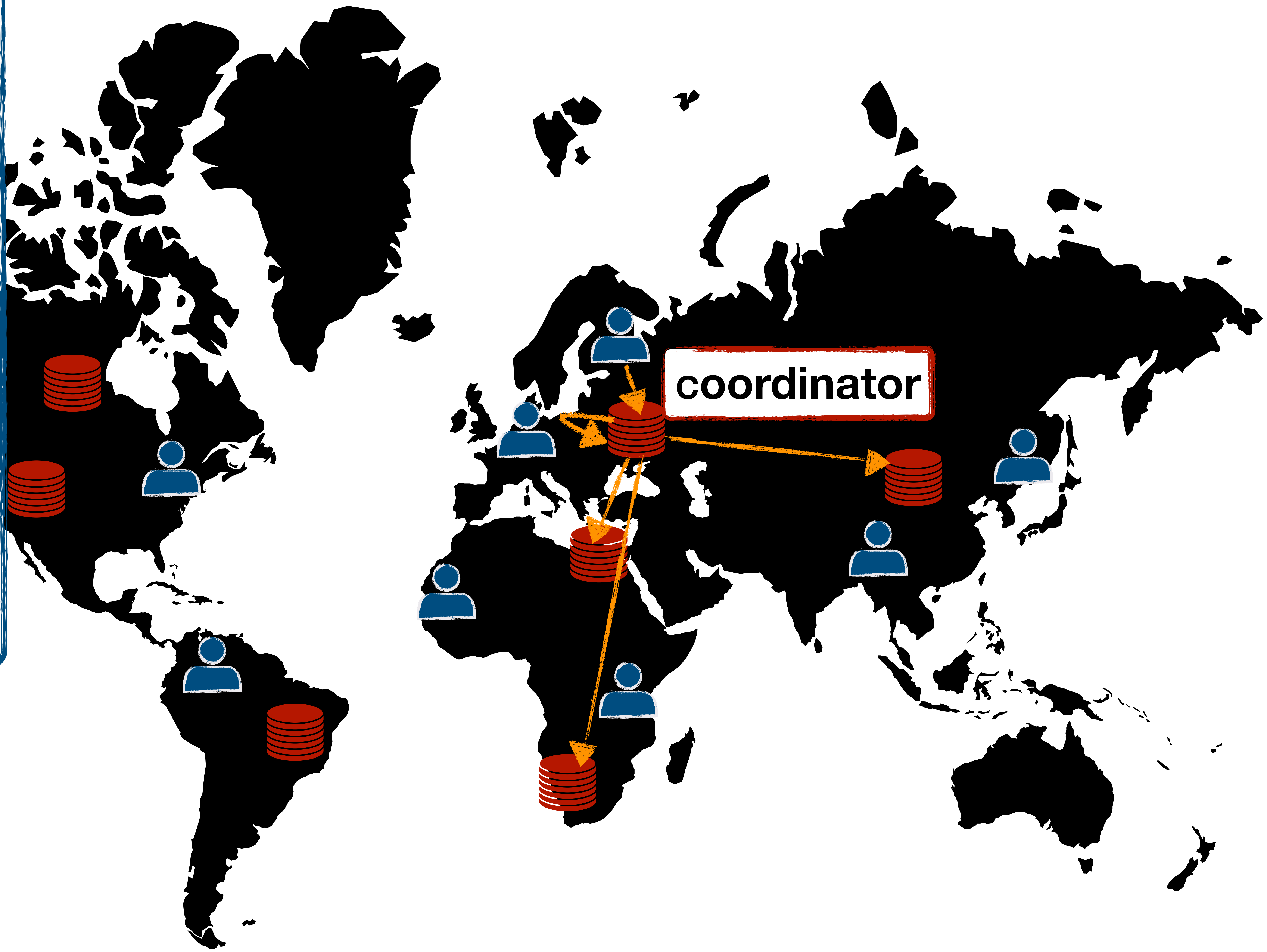
leaderless SMR (e.g. epaxos)

- replica R receives command
- forwards it to a quorum
- quorum replies
- if all replies match:
  - R commits the command



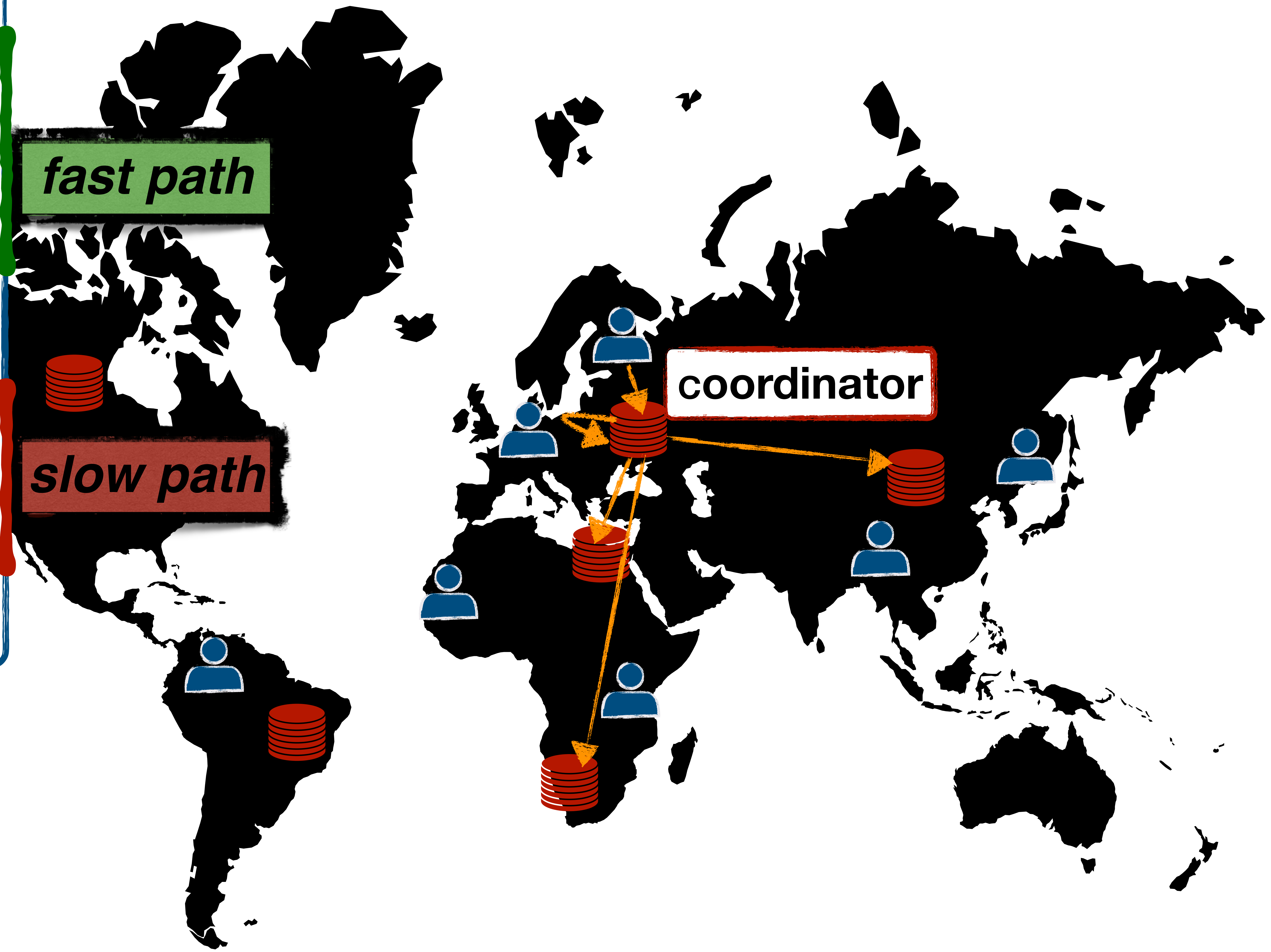
leaderless SMR (e.g. epaxos)

- replica R receives command
- forwards it to a quorum
- quorum replies
- if all replies match:
    - R commits the command
- else:
    - R contacts quorum again
    - quorum replies
    - R commits the command



leaderless SMR (e.g. epaxos)

- replica R receives command
- forwards it to a quorum
- quorum replies
- if all replies match:
  - R commits the command
- else:
  - R contacts quorum again
  - quorum replies
  - R commits the command

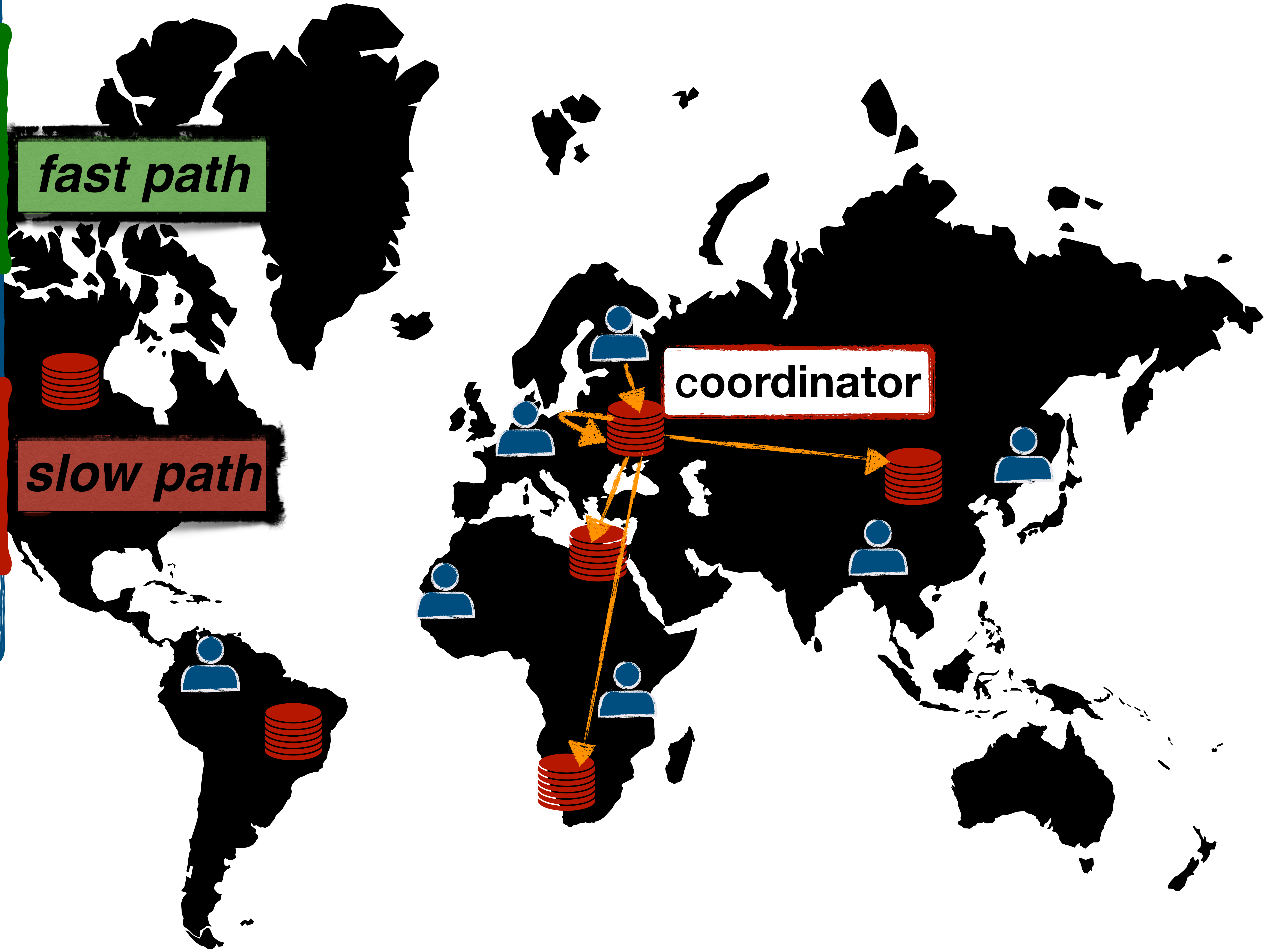


leaderless SMR (e.g. epaxos)

- replica R receives command
- forwards it to a quorum
- quorum replies
- if all replies match:
  - R commits the command
- else:
  - R contacts quorum again
  - quorum replies
  - R commits the command

fast path

slow path



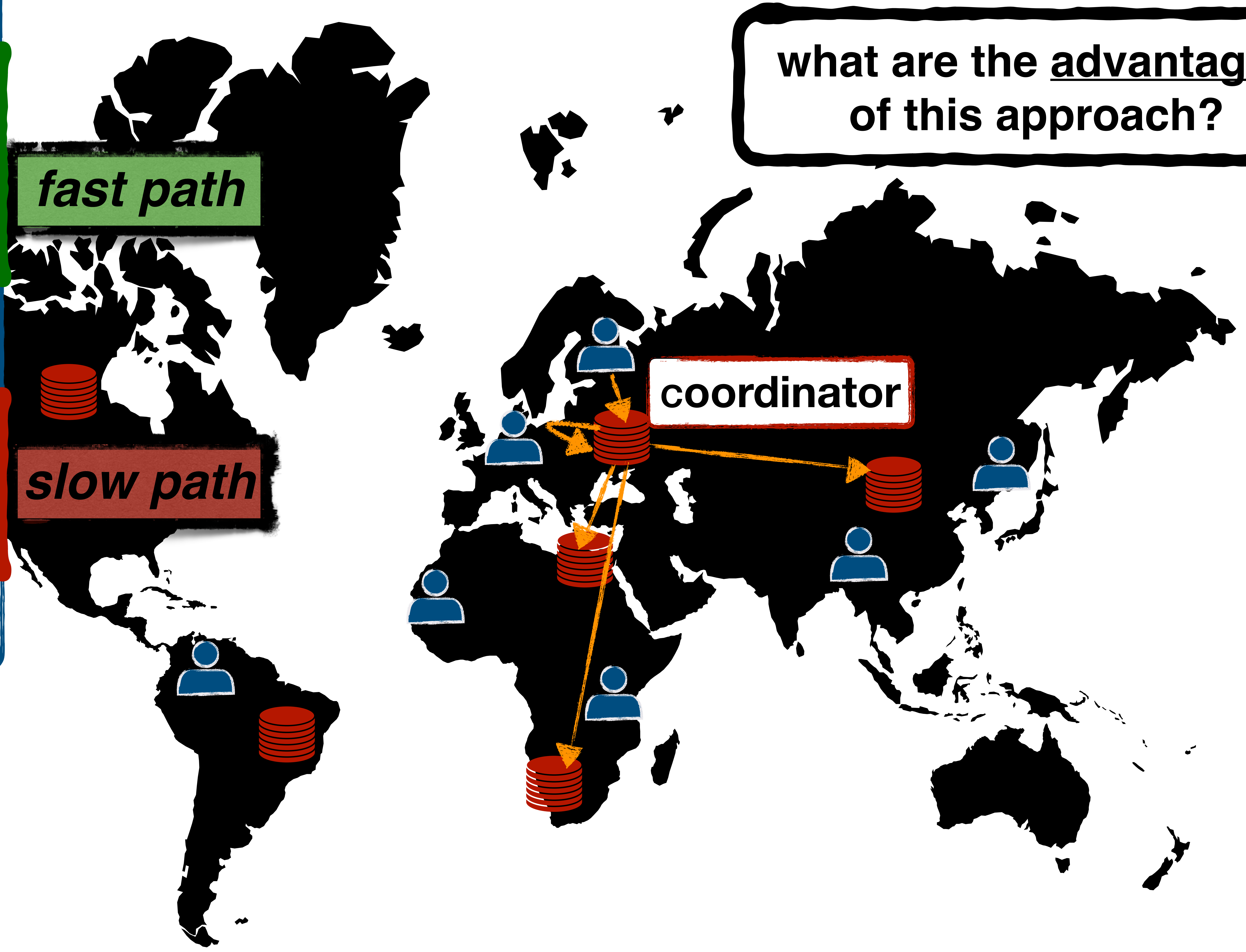
leaderless SMR (e.g. epaxos)

- replica R receives command
- forwards it to a quorum
- quorum replies
- if all replies match:
  - R commits the command
- else:
  - R contacts quorum again
  - quorum replies
  - R commits the command

fast path

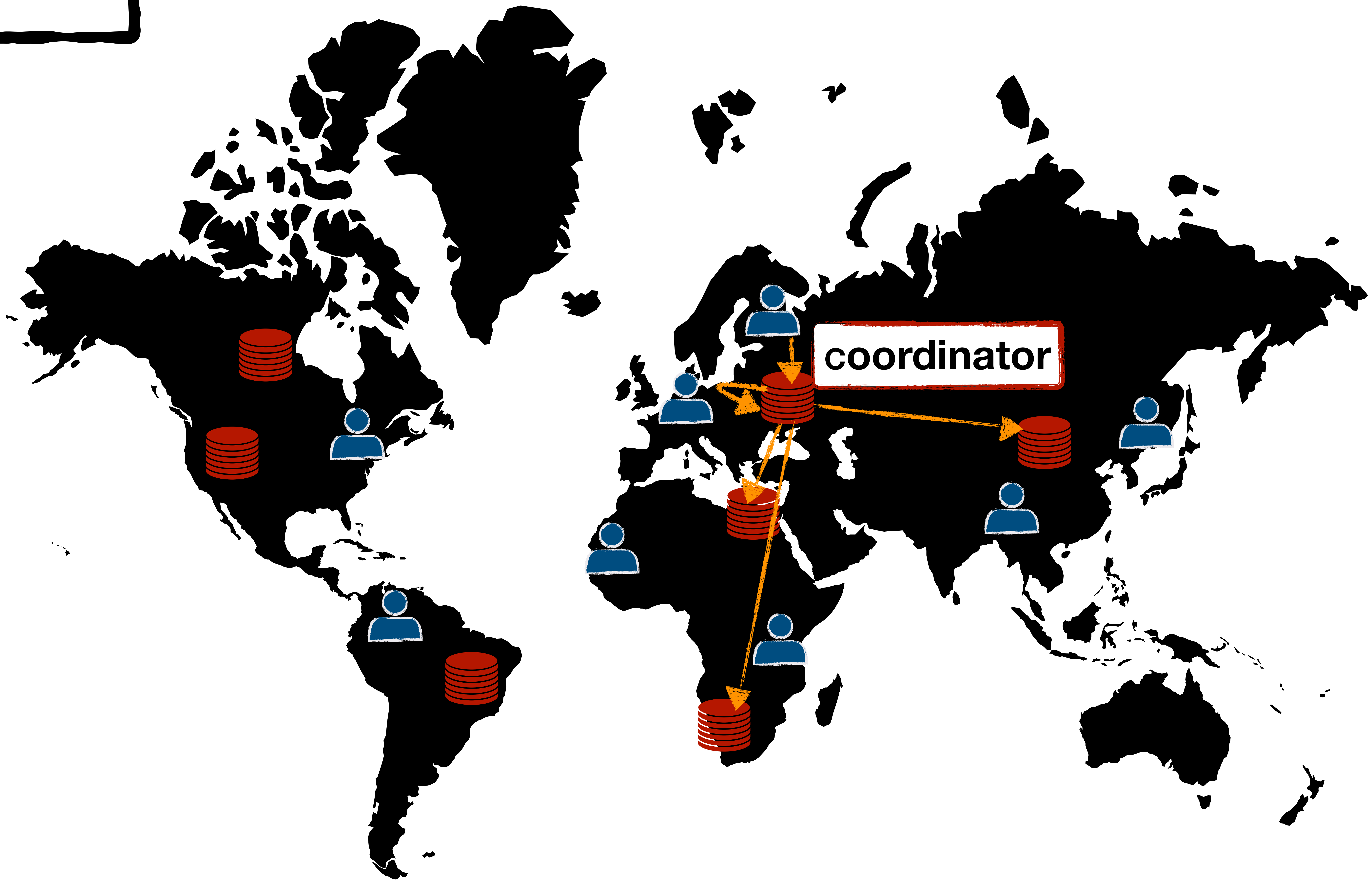
slow path

what are the advantages of this approach?



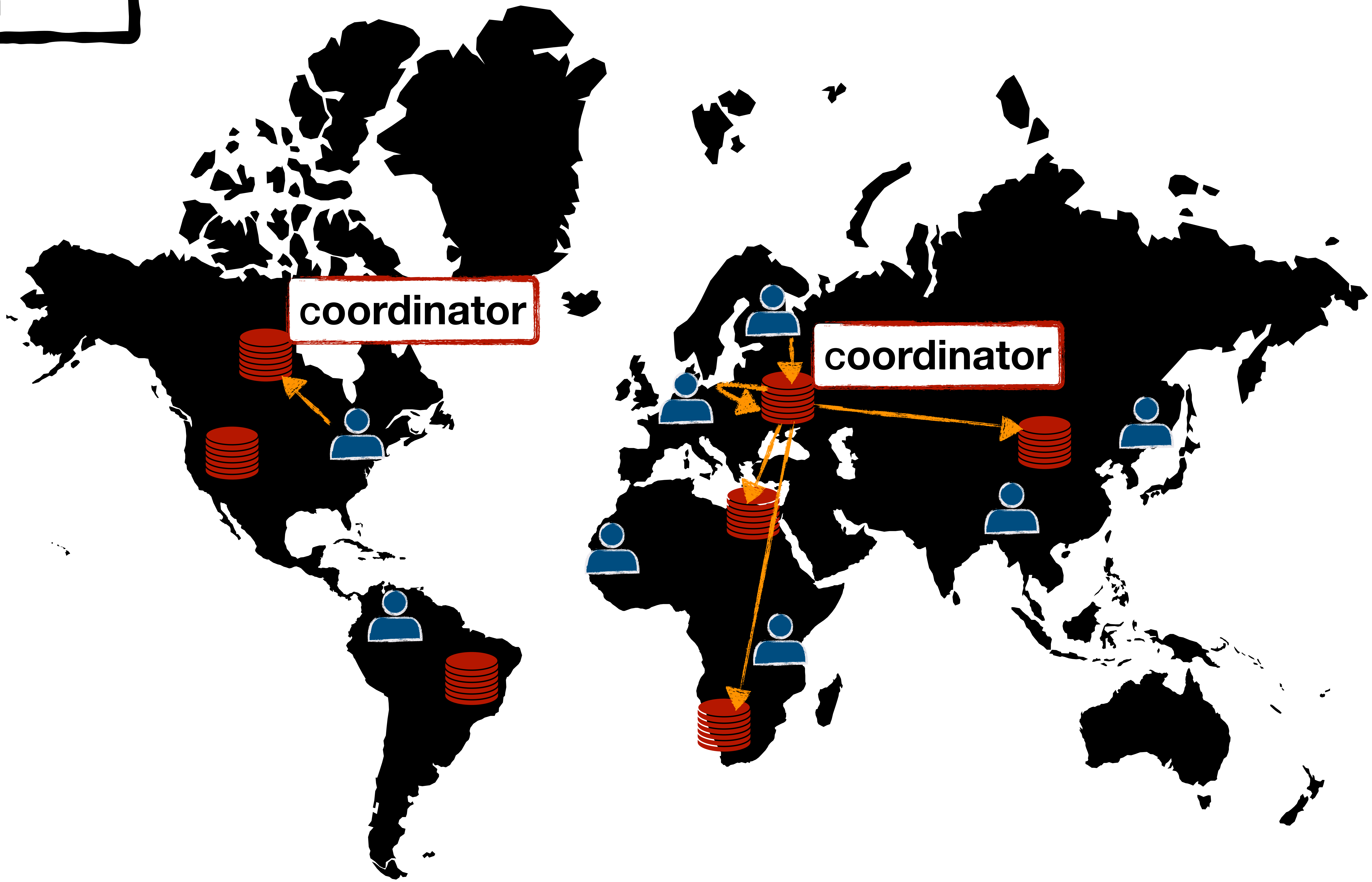
leaderless SMR advantages

fairer latency distribution ✓



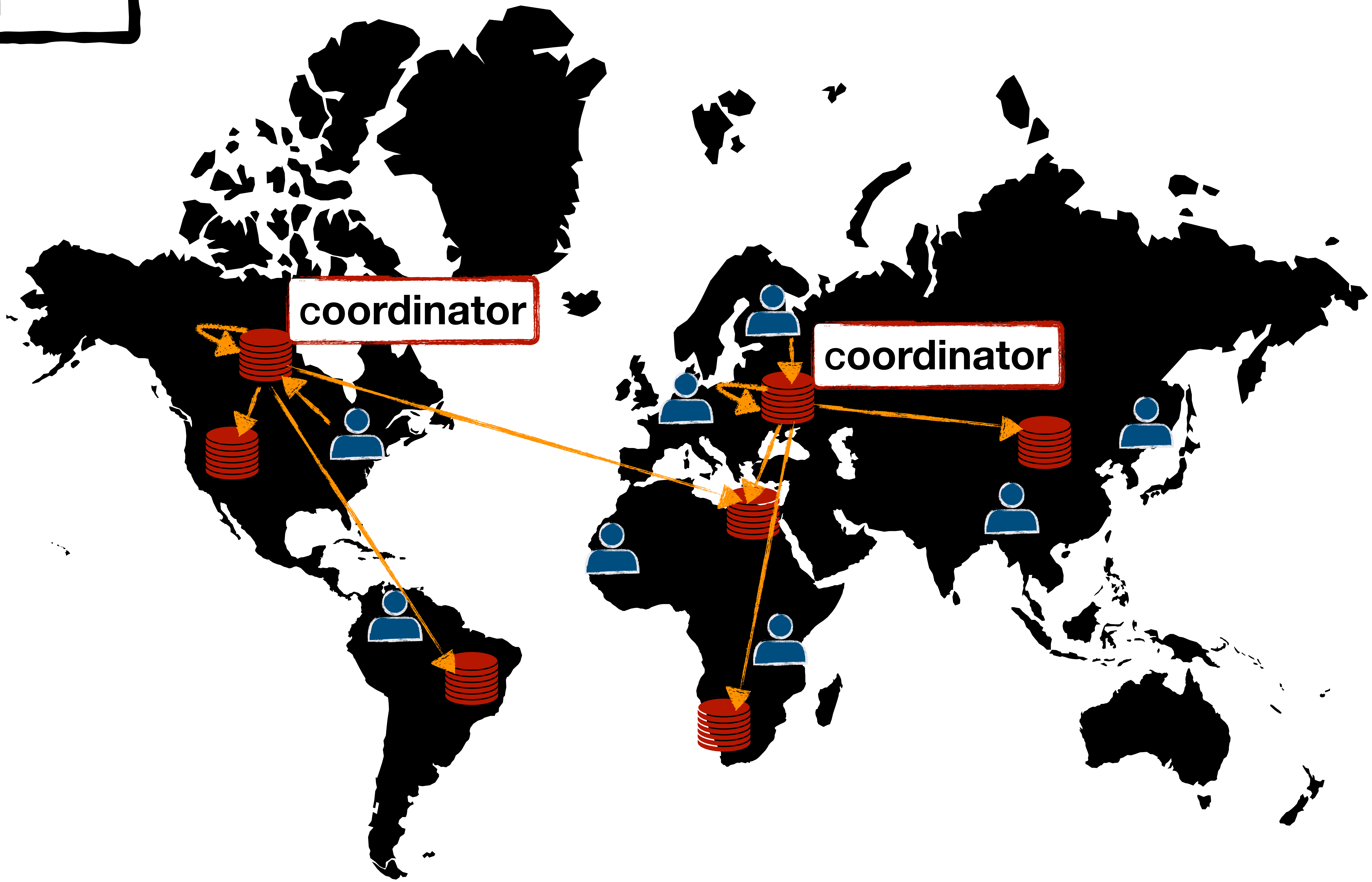
leaderless SMR advantages

fairer latency distribution ✓



leaderless SMR advantages

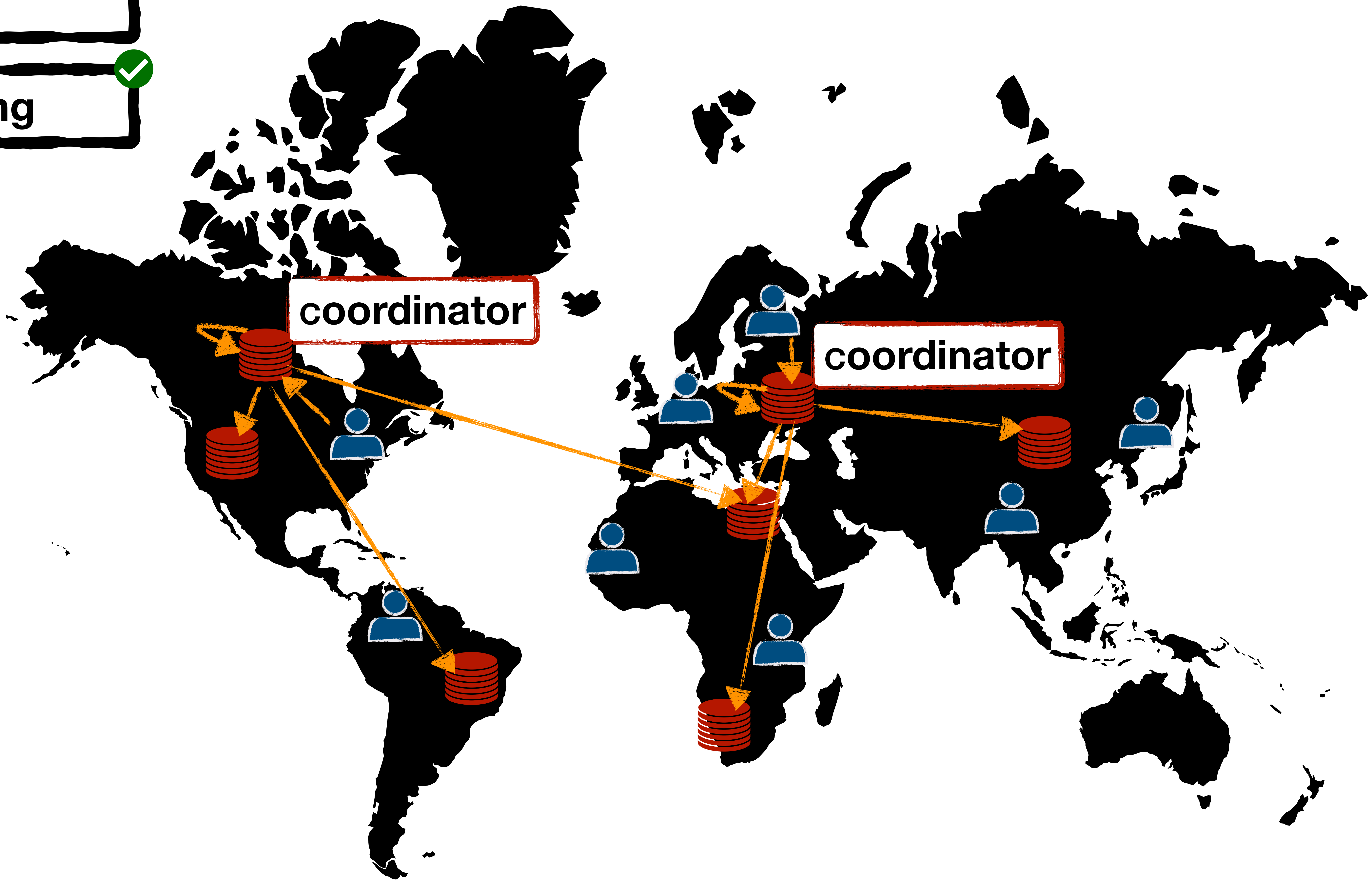
fairer latency distribution ✓



leaderless SMR advantages

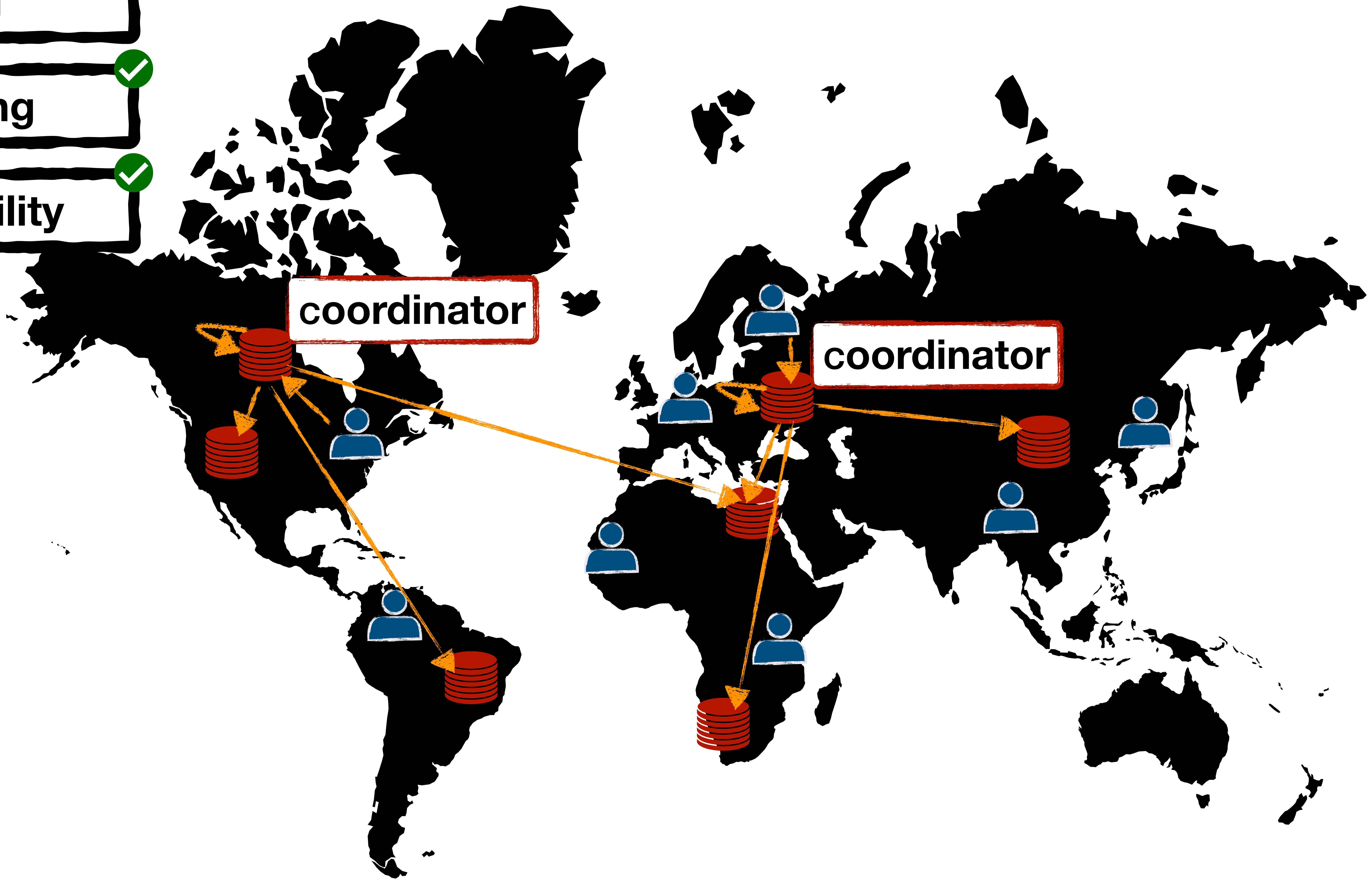
fairer latency distribution ✓

load balancing ✓



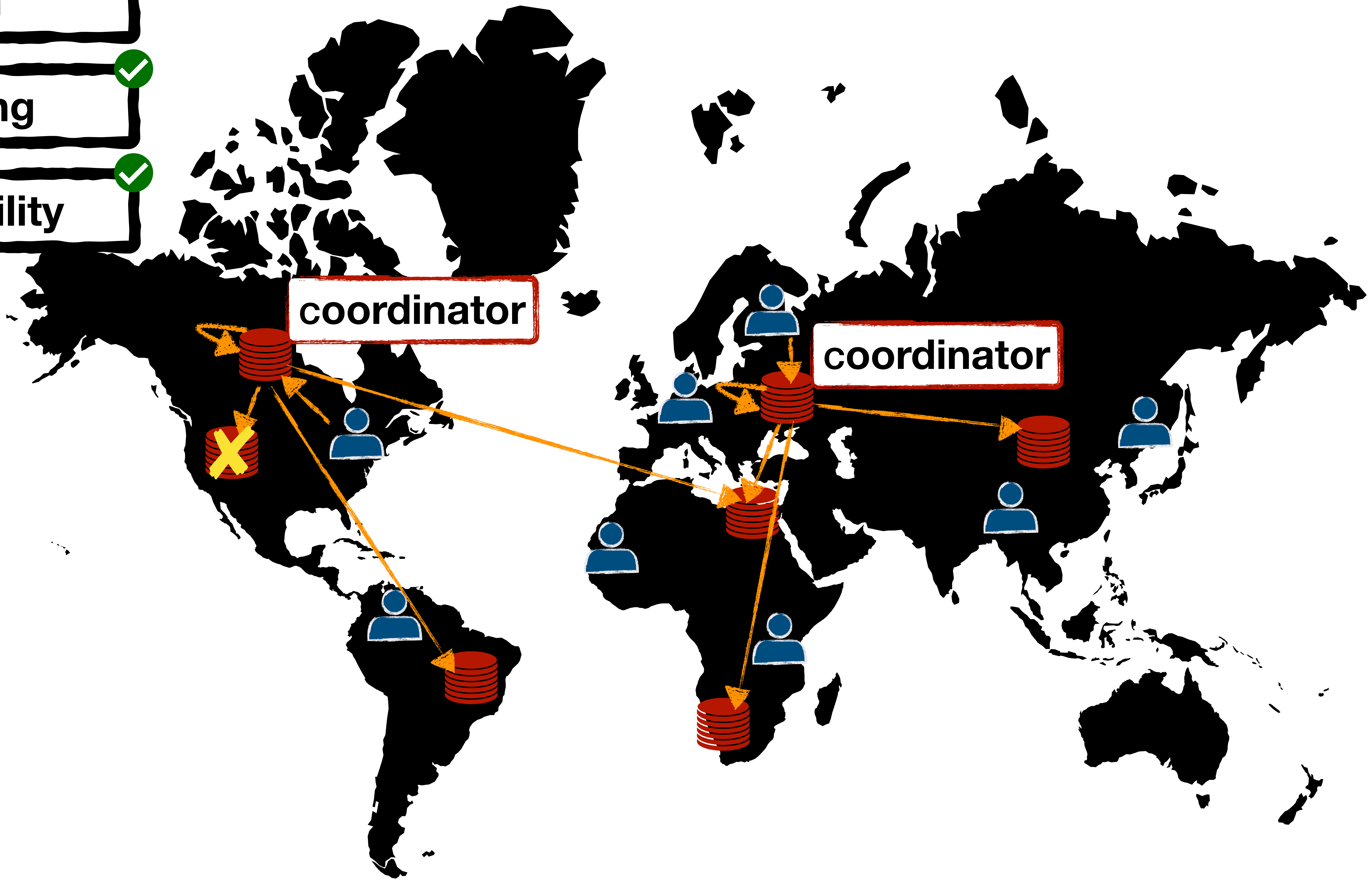
leaderless SMR advantages

- fairer latency distribution ✓
- load balancing ✓
- higher availability ✓



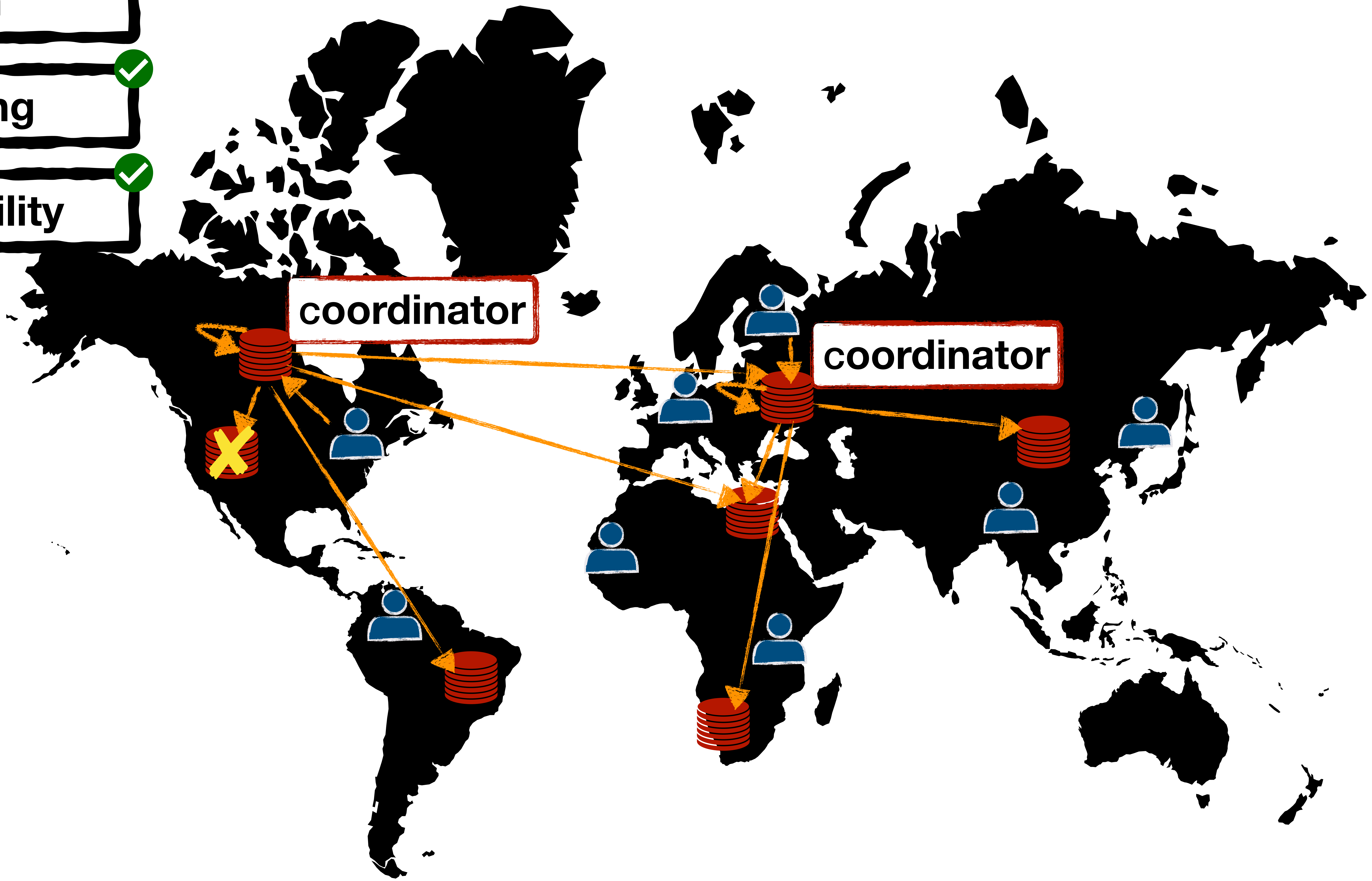
leaderless SMR advantages

- fairer latency distribution ✓
- load balancing ✓
- higher availability ✓



leaderless SMR advantages

- fairer latency distribution ✓
- load balancing ✓
- higher availability ✓



leader-based SMR	x	leaderless SMR
unfairness/high latency for faraway clients ✗		fairer latency distribution ✓
no load balancing ✗		load balancing ✓
single point of failure ✗		higher availability ✓

leader-based SMR x leaderless SMR

- unfairness/high latency for faraway clients ✗
- no load balancing ✗
- single point of failure ✗

- fairer latency distribution ✓
- load balancing ✓
- higher availability ✓
- partial replication -> higher scalability ✓

leader-based SMR x leaderless SMR

leader-based SMR

- unfairness/high latency for faraway clients ✗
- no load balancing ✗
- single point of failure ✗

leaderless SMR

- fairer latency distribution ✓
- load balancing ✓
- higher availability ✓
- partial replication -> higher scalability ✓

why haven't leaderless protocols been adopted by industry?

leader-based SMR X leaderless SMR

¡¡NOT PRACTICAL!!

- unfairness/high latency for faraway clients ✗
- no load balancing ✗
- single point of failure ✗

- fairer latency distribution ✓
- load balancing ✓
- higher availability ✓
- partial replication -> higher scalability ✓

why haven't leaderless protocols been adopted by industry?

can leaderless SMR
be practical for
planet-scale systems?

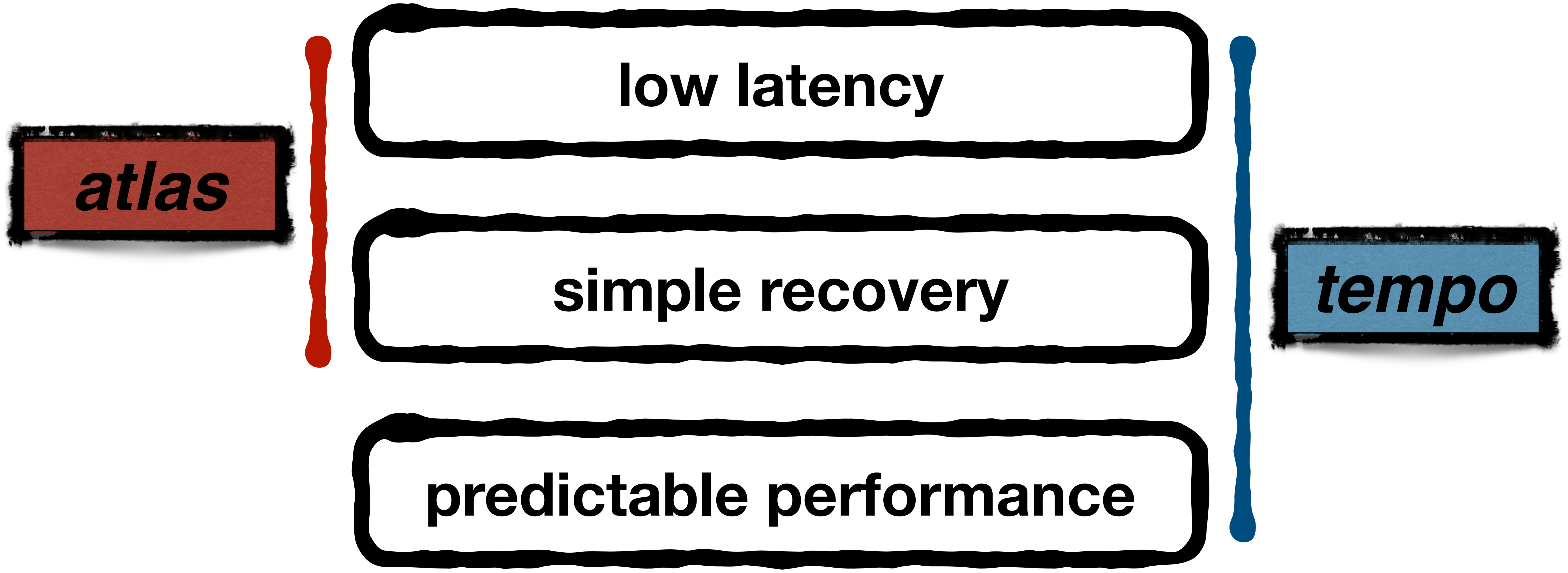
can leaderless SMR be practical for planet-scale systems?

low latency

simple recovery

predictable performance

can leaderless SMR be *practical* for planet-scale systems?



a note on commutativity

- leaderless protocols typically exploit the fact that **commands frequently commute**
  - and when they do, **commands don’t have to be ordered** (improving performance)

a note on commutativity

- leaderless protocols typically exploit the fact that **commands frequently commute**
  - and when they do, **commands don’t have to be ordered** (improving performance)

***we say that commands conflict
when they do not commute***

low latency - SMR protocols

	small quorums	single round-trip
paxos		
epaxos		
atlas		
tempo		

low latency - SMR protocols

	small quorums	single round-trip
paxos	✓ f+1	
epaxos		
atlas		
tempo		

low latency - SMR protocols

	small quorums	single round-trip
paxos	✓ f+1	
epaxos	✗ 3r/4	
atlas		
tempo		

low latency - SMR protocols

	small quorums	single round-trip
paxos	✓ f+1	
epaxos	✗ 3r/4	
atlas		
tempo		

f is always minority :(

low latency - bounds on failures

- **existing leaderless protocols assume a minority for f** (e.g. **f=3** out of **r=7**)

low latency - bounds on failures

- **existing leaderless protocols assume a minority for f** (e.g. **f=3** out of **r=7**)

observation: this is unnecessarily high

low latency - bounds on failures

- **existing leaderless protocols assume a minority for f** (e.g. **f=3** out of **r=7**)

observation: this is unnecessarily high

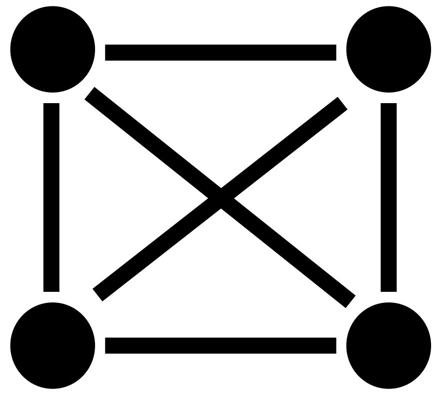
- **concurrent data center (DC) failures are rare**
- **concurrent network failures are also rare**

low latency - bounds on failures

- **existing leaderless protocols assume a minority for f** (e.g. **f=3** out of **r=7**)

<u>observation</u>: this is unnecessarily high

- **concurrent data center (DC) failures are rare**
- **concurrent network failures are also rare**
  - we ran a **link-monitoring experiment** on Google Cloud Platform
    - 17 DCs where each DC periodically **pings** the remaining DCs

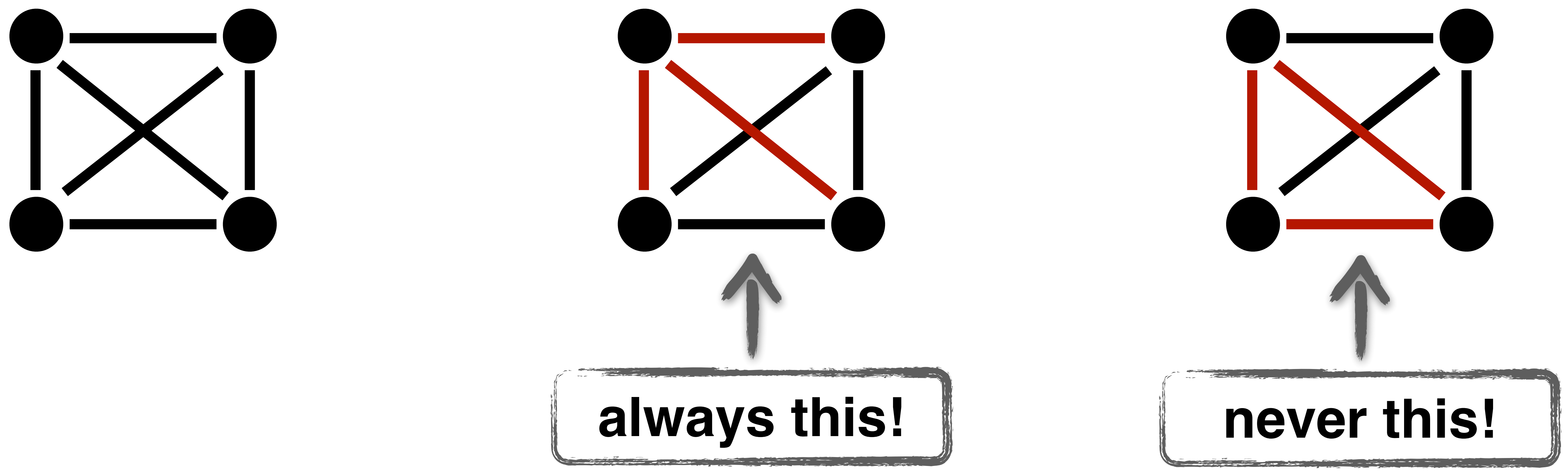


low latency - bounds on failures

- **existing leaderless protocols assume a minority for f** (e.g. **f=3** out of **r=7**)

observation: this is unnecessarily high

- **concurrent data center (DC) failures are rare**
- **concurrent network failures are also rare**
  - we ran a **link-monitoring experiment** on Google Cloud Platform
    - 17 DCs where each DC periodically **pings** the remaining DCs
  - after 3 months, we searched for **concurrent link slowdowns**
    - they were **always incident to 1 DC**



always this!

never this!

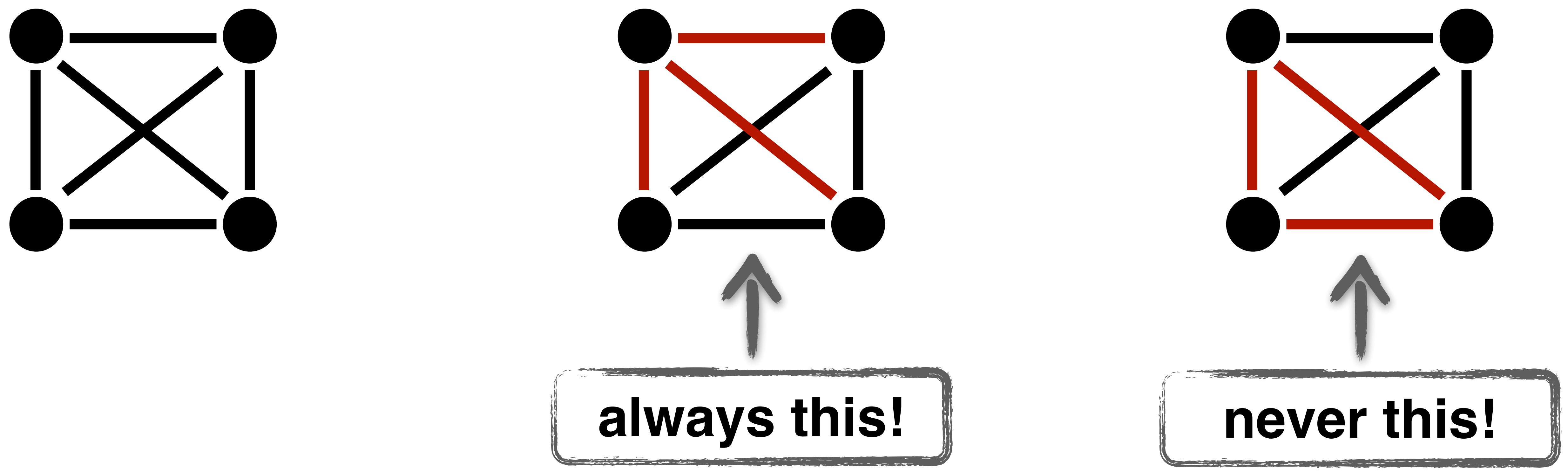
low latency - bounds on failures

- **existing leaderless protocols assume a minority for f** (e.g. **f=3** out of **r=7**)

<u>observation</u>: this is unnecessarily high

- **concurrent data center (DC) failures are rare**
- **concurrent network failures are also rare**
  - we ran a **link-monitoring experiment** on Google Cloud Platform
    - 17 DCs where each DC periodically **pings** the remaining DCs
  - after 3 months, we searched for **concurrent link slowdowns**
    - they were **always incident to 1 DC**

f=1* or *f=2
is acceptable



always this!

never this!

low latency - SMR protocols

	small quorums	single round-trip
paxos	✓ f+1	
epaxos	✗ 3r/4	
atlas		
tempo		

low latency - SMR protocols

	small quorums	single round-trip
paxos	✓ f+1	
epaxos	✗ 3r/4	
atlas	✓ r/2+f	
tempo	✓ r/2+f	

small for small values of f

low latency - SMR protocols

	small quorums	single round-trip
paxos	✓ f+1	⊖ only from the leader
epaxos	✗ 3r/4	
atlas	✓ r/2+f	
tempo	✓ r/2+f	

small for small values of f

low latency - SMR protocols

	small quorums	single round-trip
paxos	✓ f+1	⊖ only from the leader
epaxos	✗ 3r/4	⊖ only if replies match
atlas	✓ r/2+f	
tempo	✓ r/2+f	

small for small values of f

low latency - SMR protocols

	small quorums	single round-trip
paxos	✓ f+1	⊖ only from the leader
epaxos	✗ 3r/4	⊖ only if replies match
atlas	✓ r/2+f	✓
tempo	✓ r/2+f	✓

small for small values of f

flexible fast-path condition!

atlas

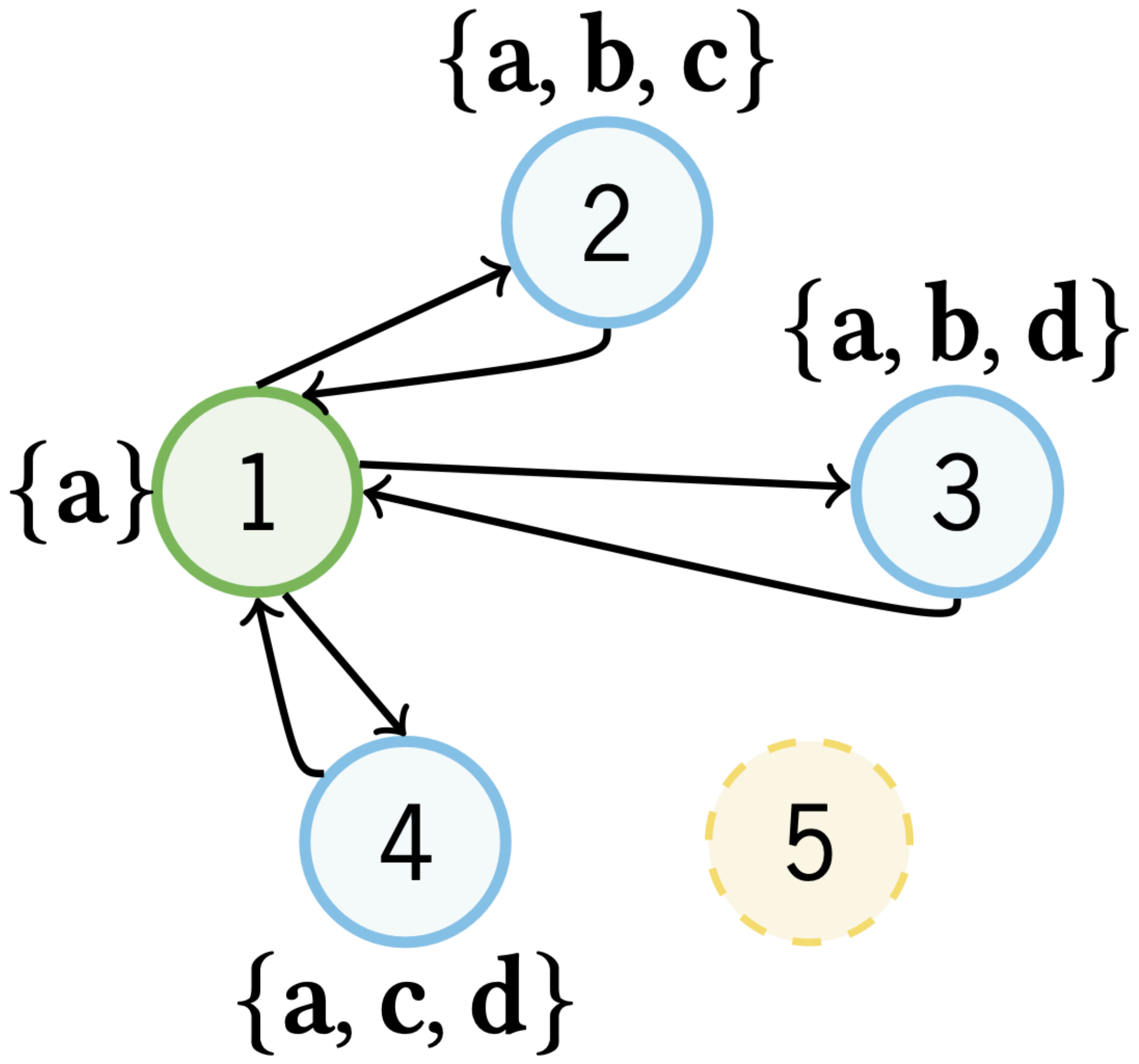
low latency - flexible fast path condition

fast path condition: each conflict was reported by at least f processes

atlas

low latency - flexible fast path condition

fast path condition: each conflict was reported by at least f processes



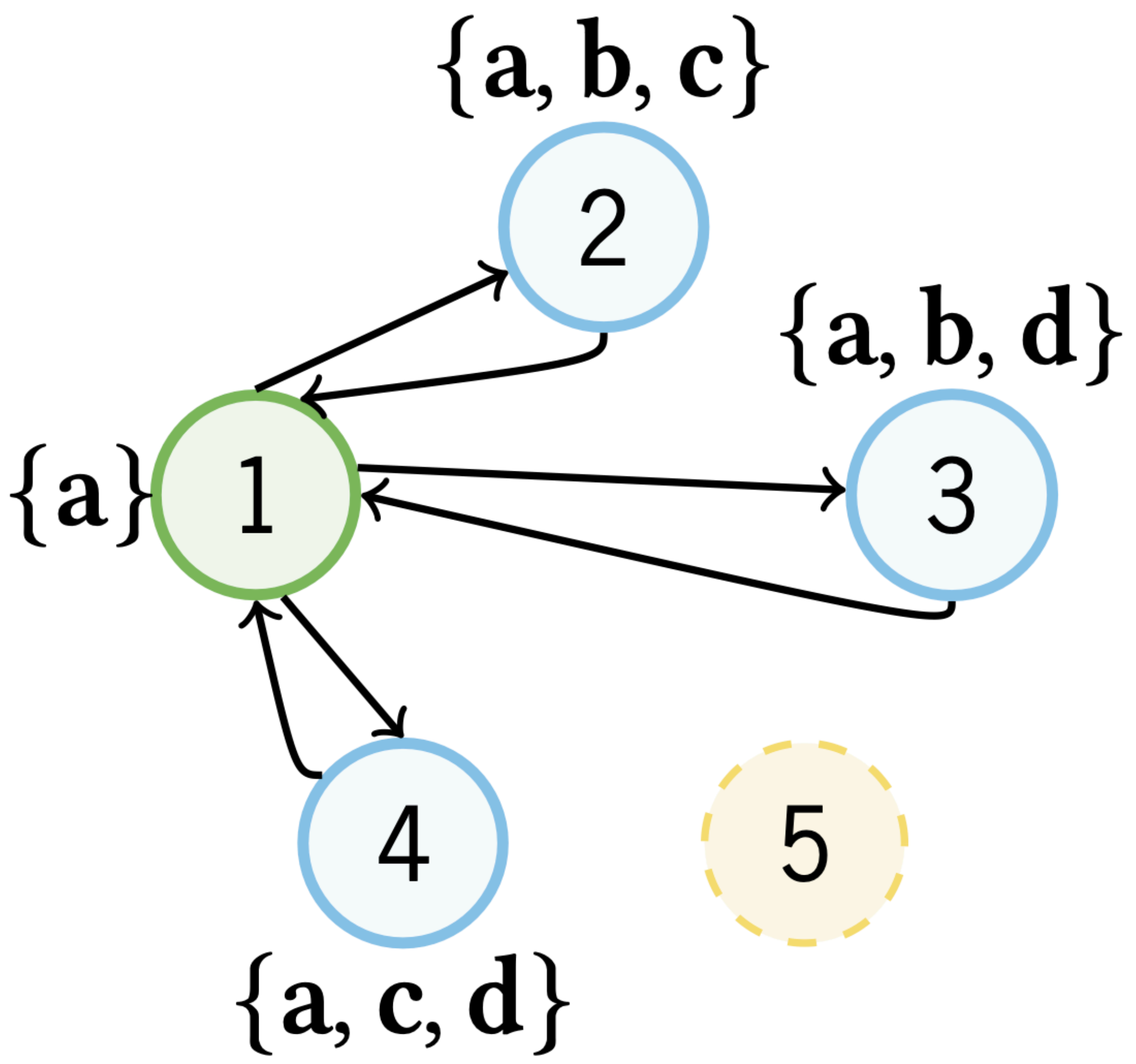
(a) ✓ `Atlas` $f = 2$

✗ *matching replies*

atlas

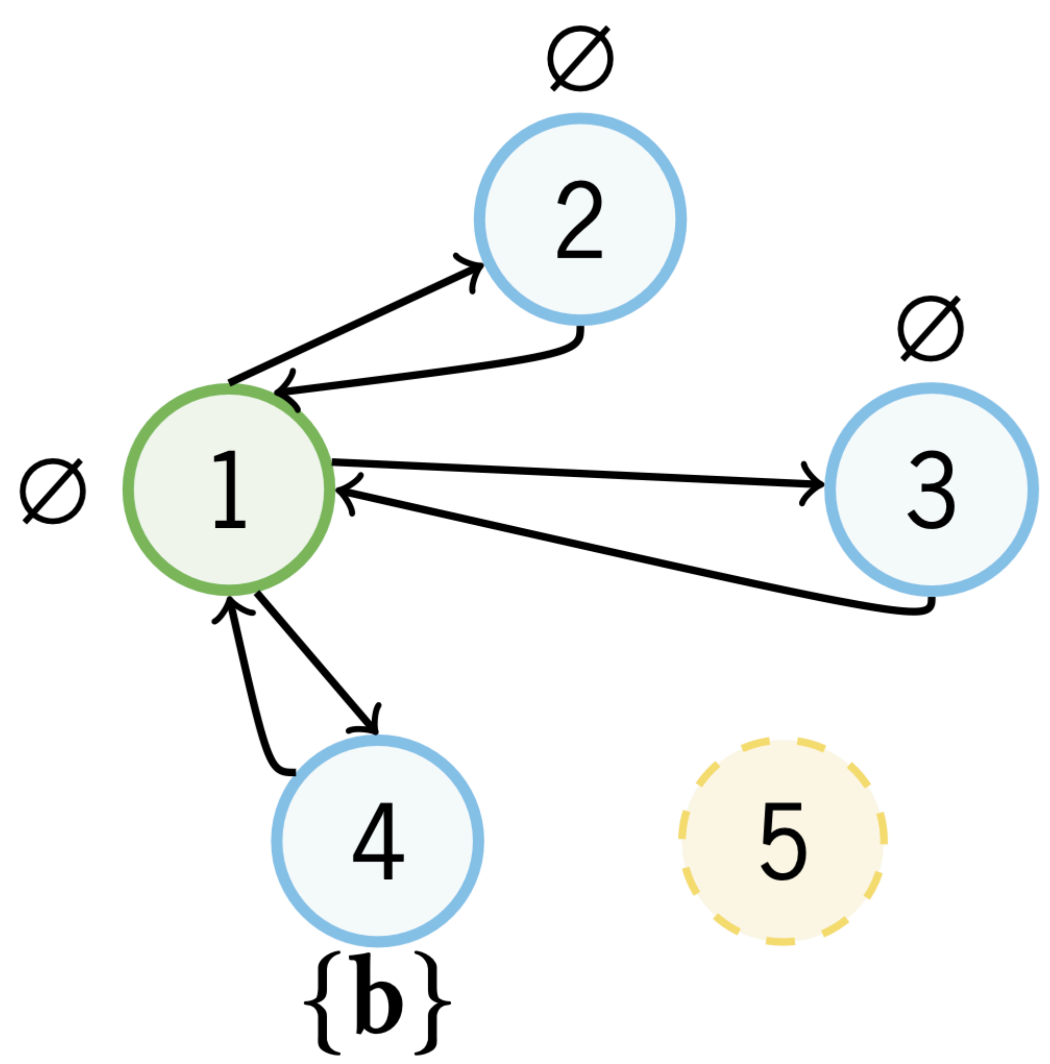
low latency - flexible fast path condition

fast path condition: each conflict was reported by at least f processes



(a) ✓ Atlas $f = 2$

✗ *matching replies*



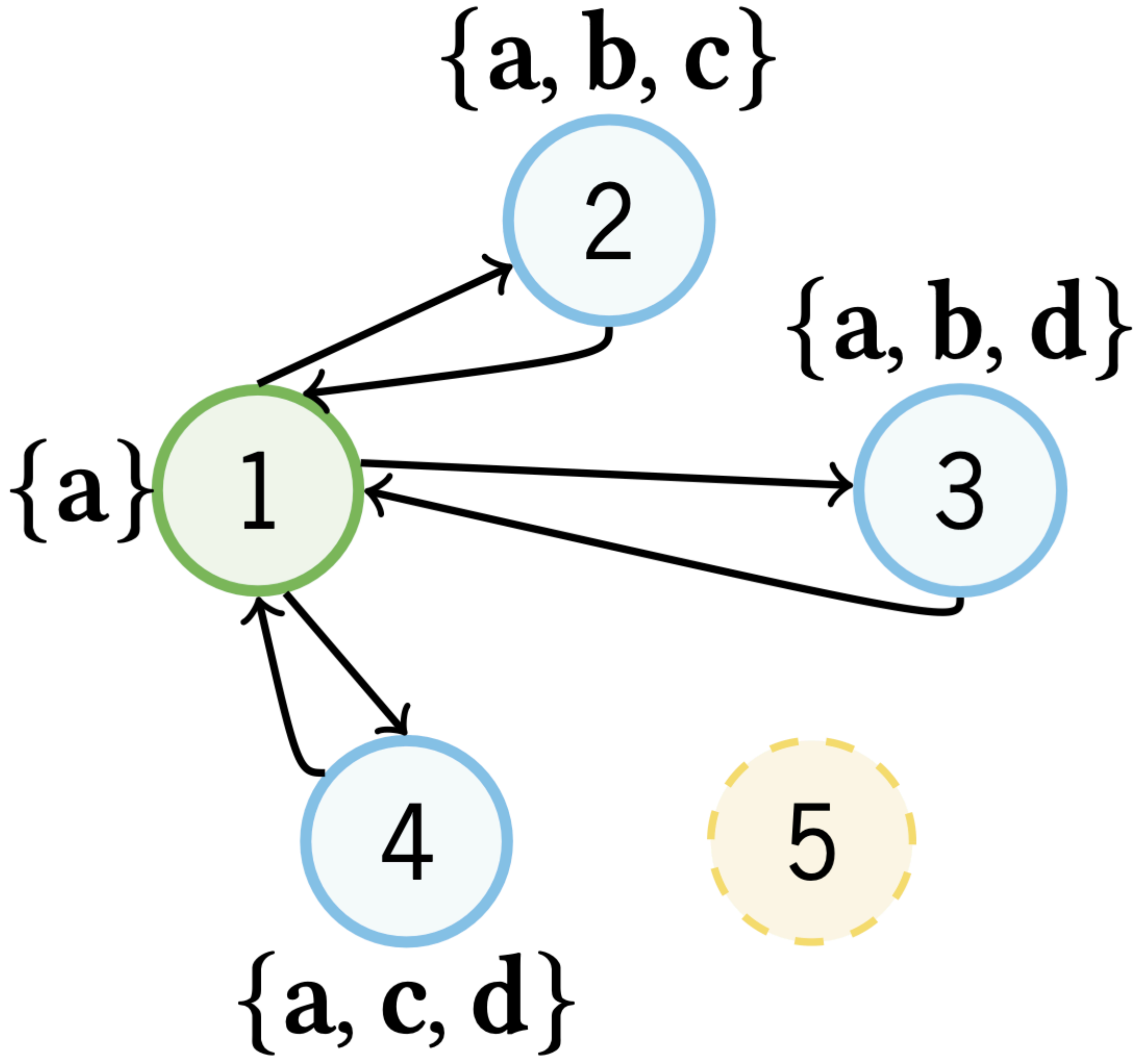
(b) ✗ Atlas $f = 2$

✗ *matching replies*

atlas

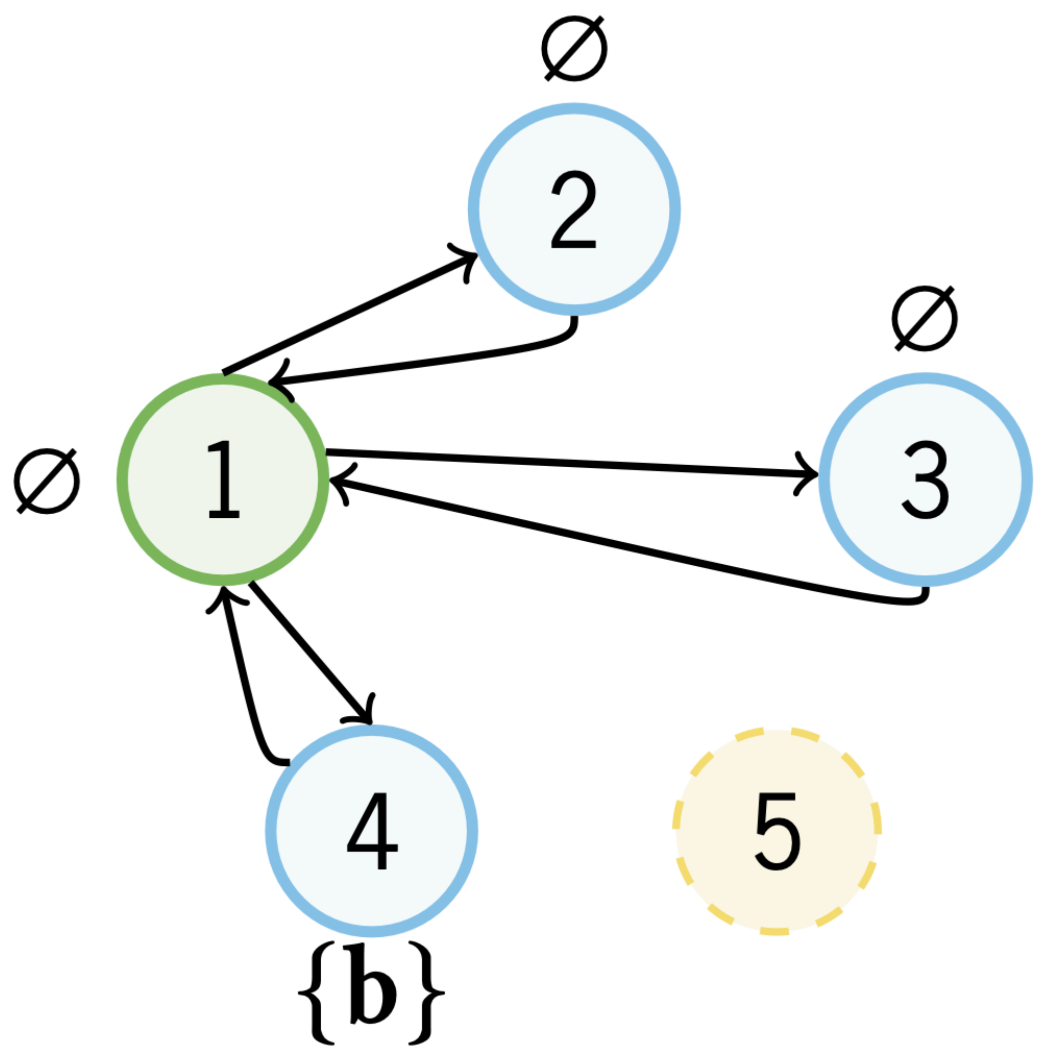
low latency - flexible fast path condition

fast path condition: each conflict was reported by at least f processes



(a) ✓ Atlas $f = 2$

✗ *matching replies*



(b) ✗ Atlas $f = 2$

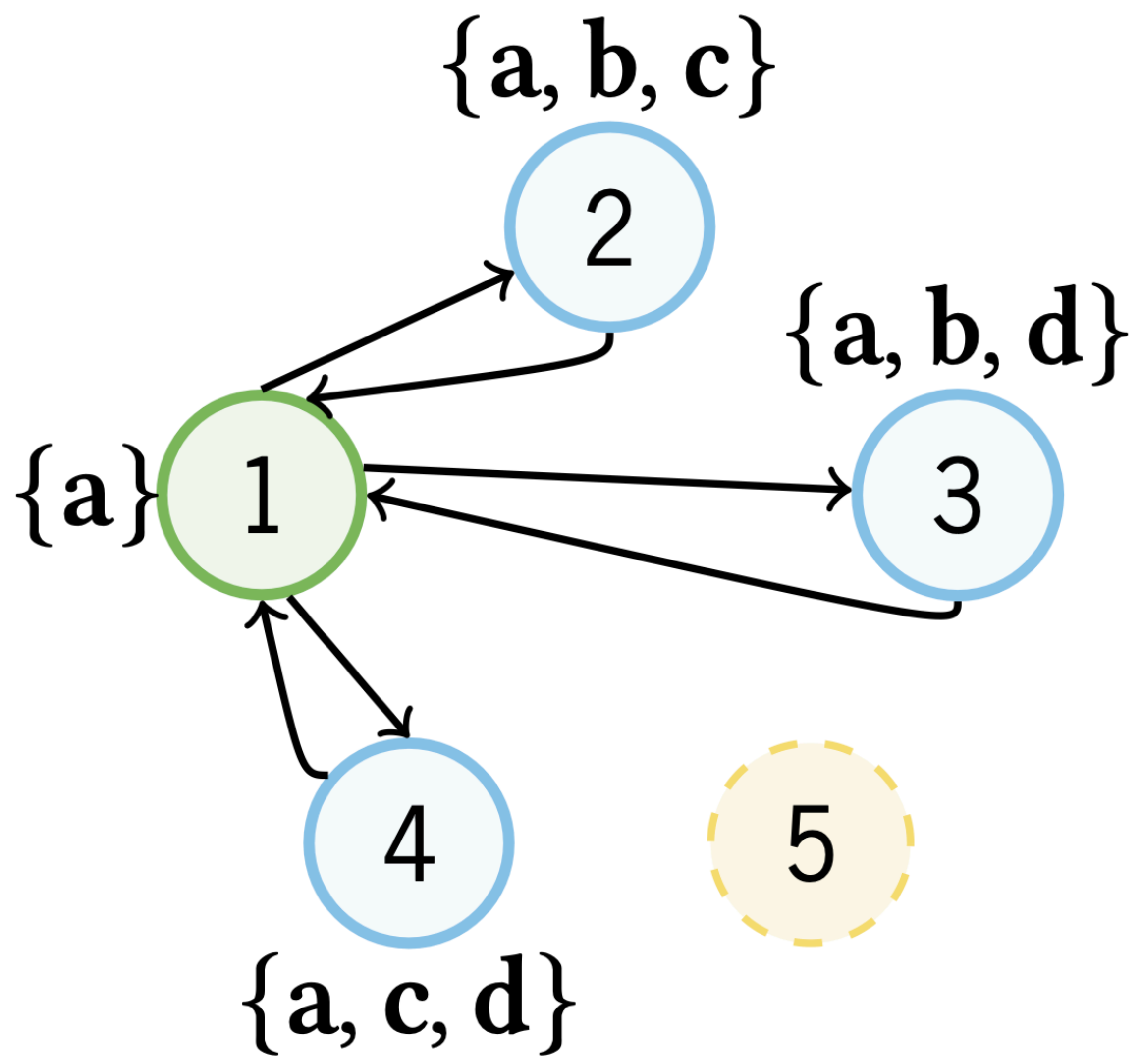
✗ *matching replies*

b reported only by 1< f process

atlas

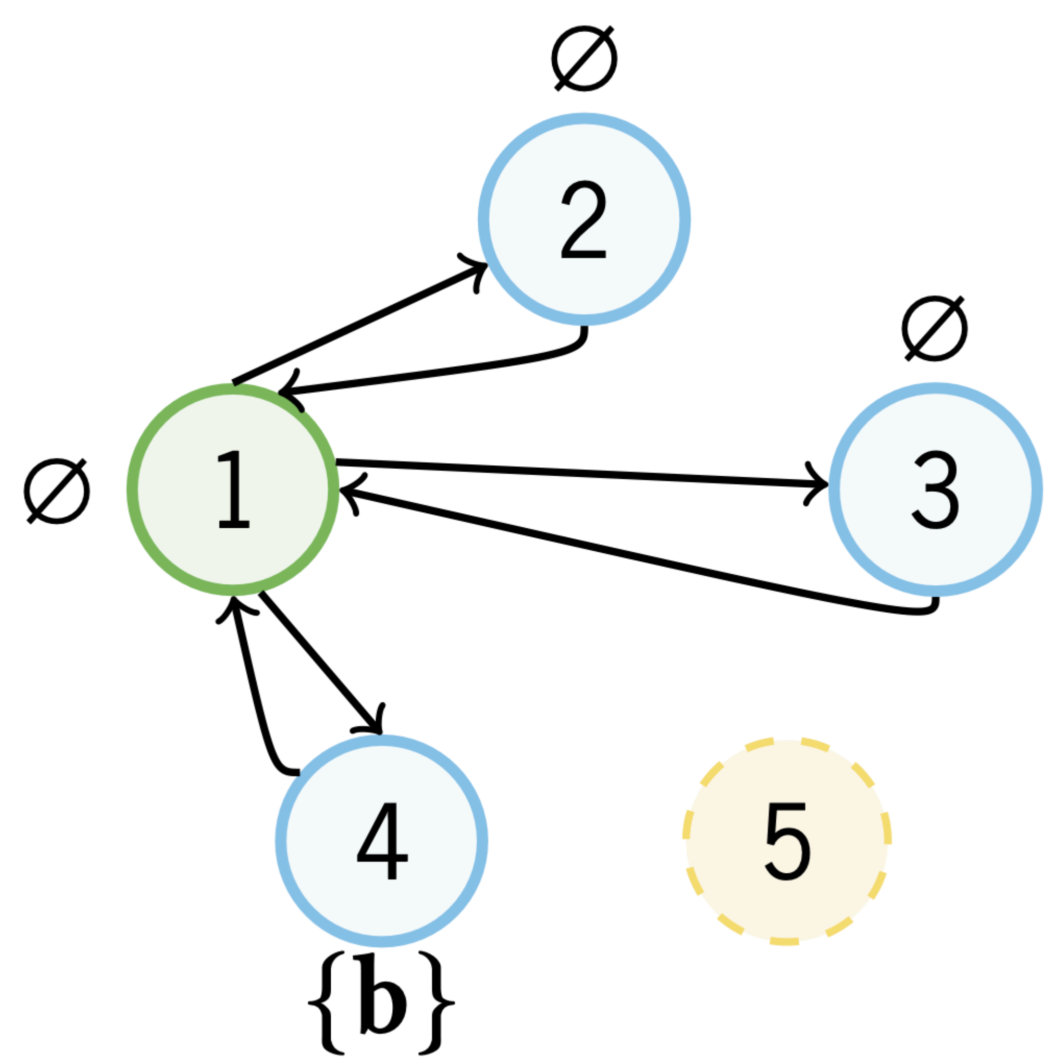
low latency - flexible fast path condition

fast path condition: each conflict was reported by at least f processes



(a) ✓ $\texttt{Atlas}\ f = 2$

✗ *matching replies*



(b) ✗ $\texttt{Atlas}\ f = 2$

✗ *matching replies*

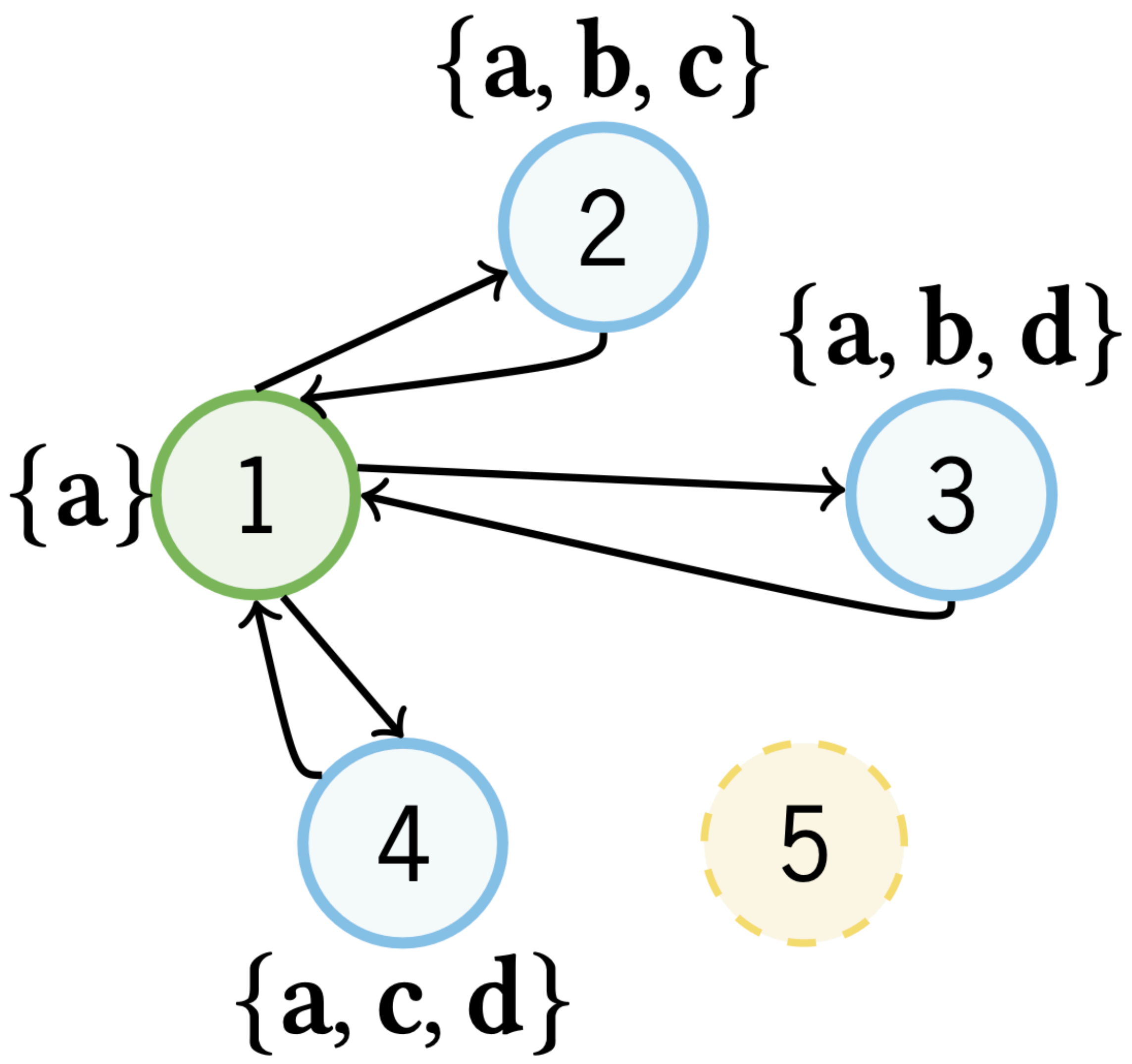
b reported only by 1< f process

epaxos would take the slow path in both examples

atlas

low latency - flexible fast path condition

fast path condition: each conflict was reported by at least f processes



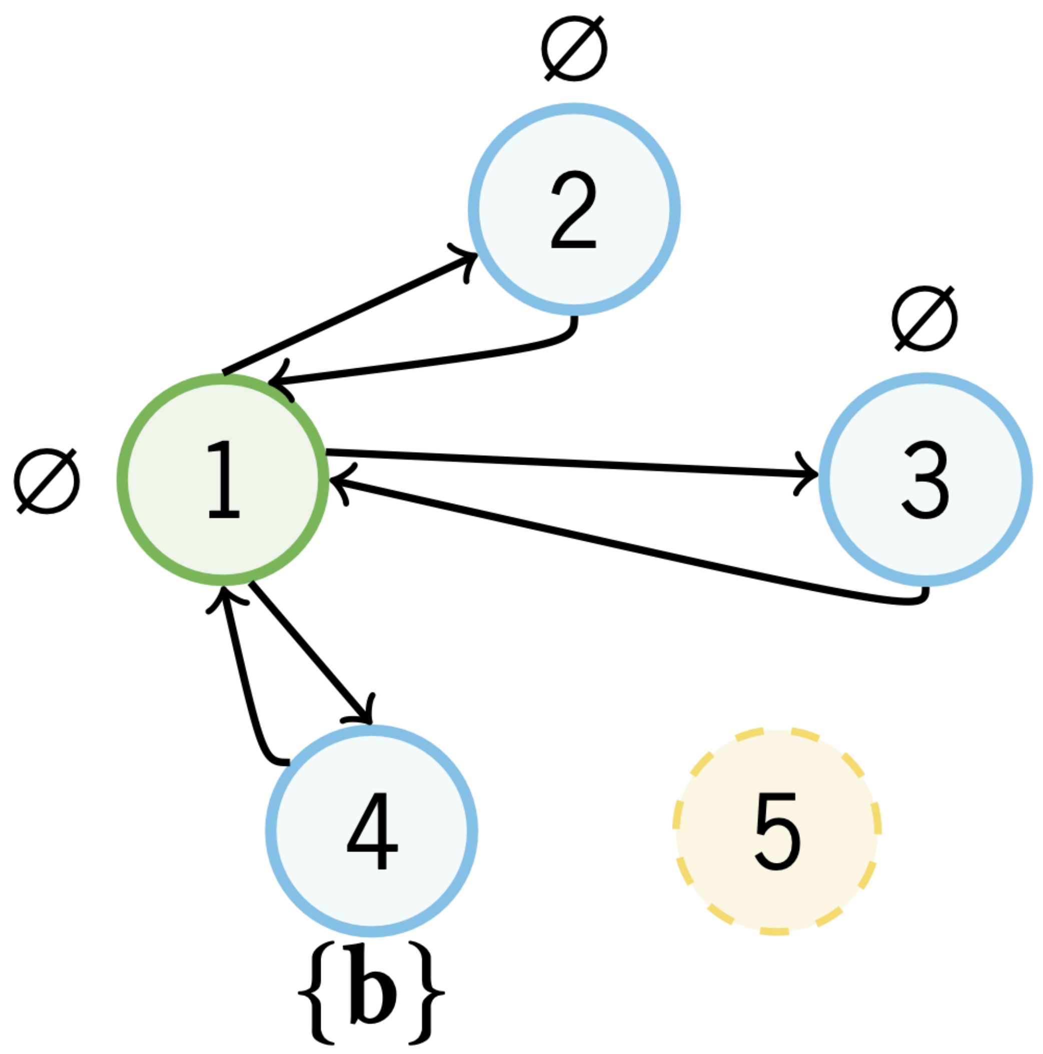
(a)

✓ Atlas $f = 2$

✗ *matching replies*

commit

dep = {a, b, c, d}



(b)

✗ Atlas $f = 2$

✗ *matching replies*

b reported only by 1< f process

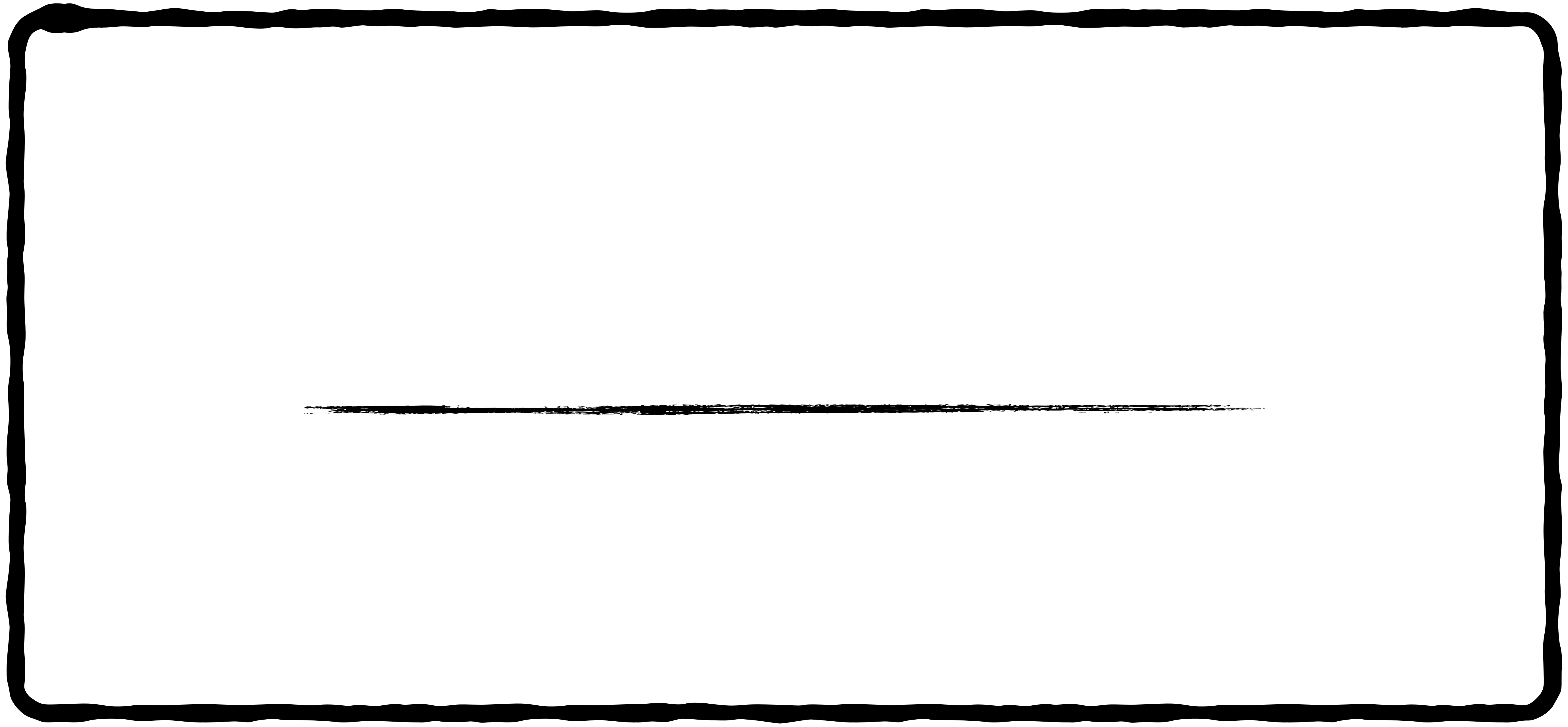
commit

dep = {b}

epaxos would take the slow path in both examples

command execution

atlas



epaxos introduced this idea

***committed dependencies** (and arbitration)*

determine command execution order

command execution

atlas

dep[a] = { }
dep[b] = {a, c}
dep[c] = {a, b}

epaxos introduced this idea

committed dependencies *(and arbitration)*
determine command execution order

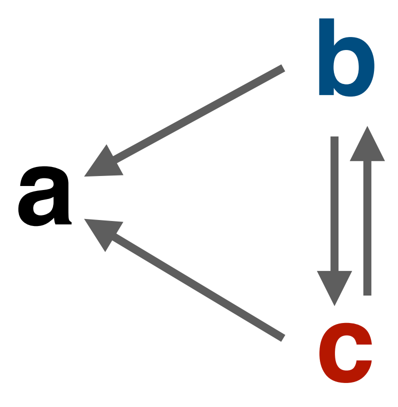
command execution

atlas

dep[a] = { }

dep[b] = {a, c}

dep[c] = {a, b}



epaxos introduced this idea

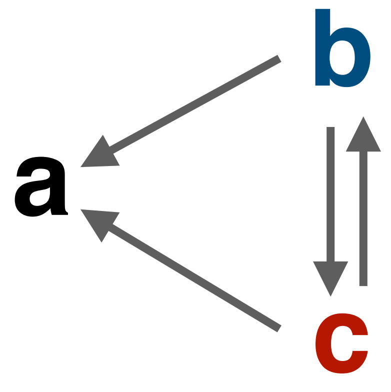
committed dependencies *(and arbitration)*

determine command execution order

command execution

atlas

dep[a] = { }
dep[b] = {a, c}
dep[c] = {a, b}



1. execute(a) ;
2. execute(a) ;

epaxos introduced this idea

committed dependencies *(and arbitration)*

determine command execution order

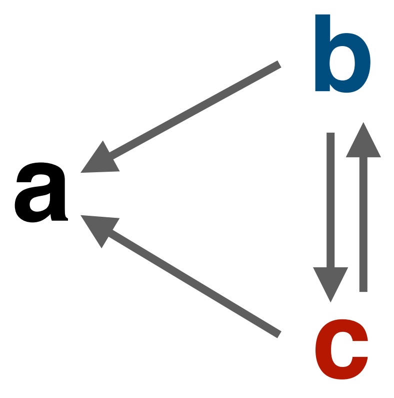
command execution

atlas

dep[a] = { }

dep[b] = {a, c}

dep[c] = {a, b}



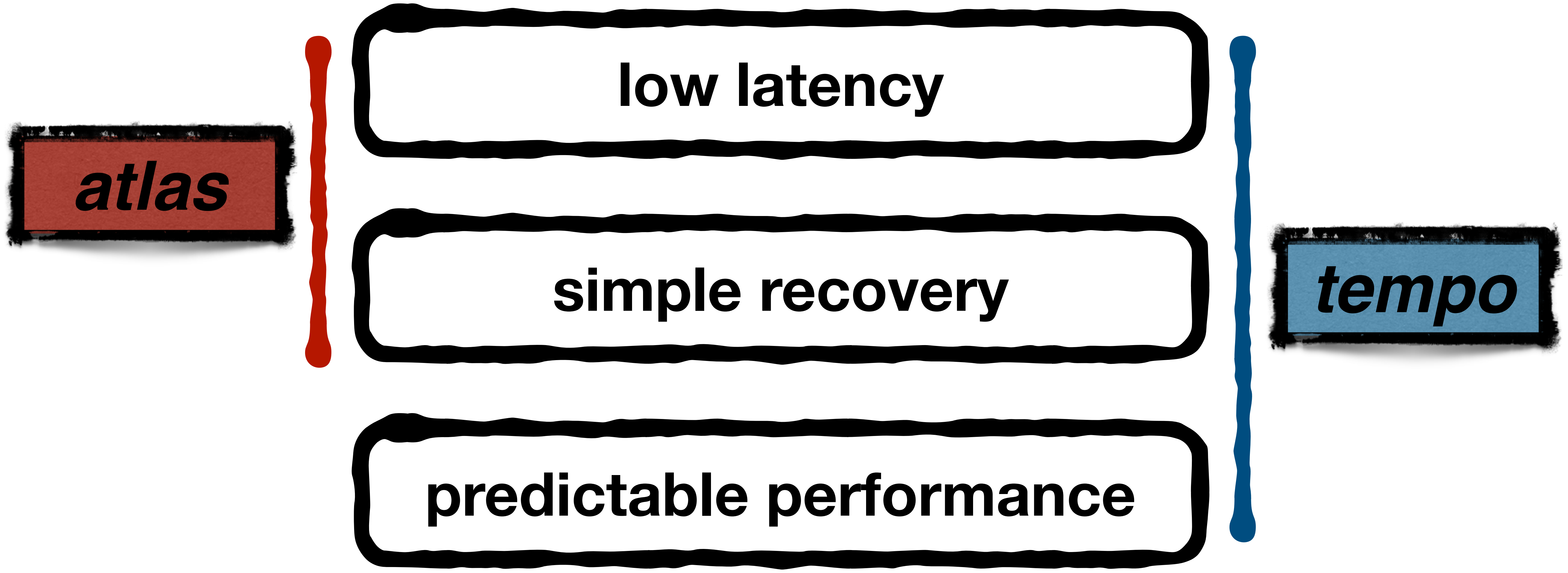
1. execute(a) ; execute(b) ; execute(c) if b < c
2. execute(a) ; execute(c) ; execute(b) if b > c

epaxos introduced this idea

***committed dependencies** (and arbitration)*

determine command execution order

can leaderless SMR be practical for planet-scale systems?



atlas

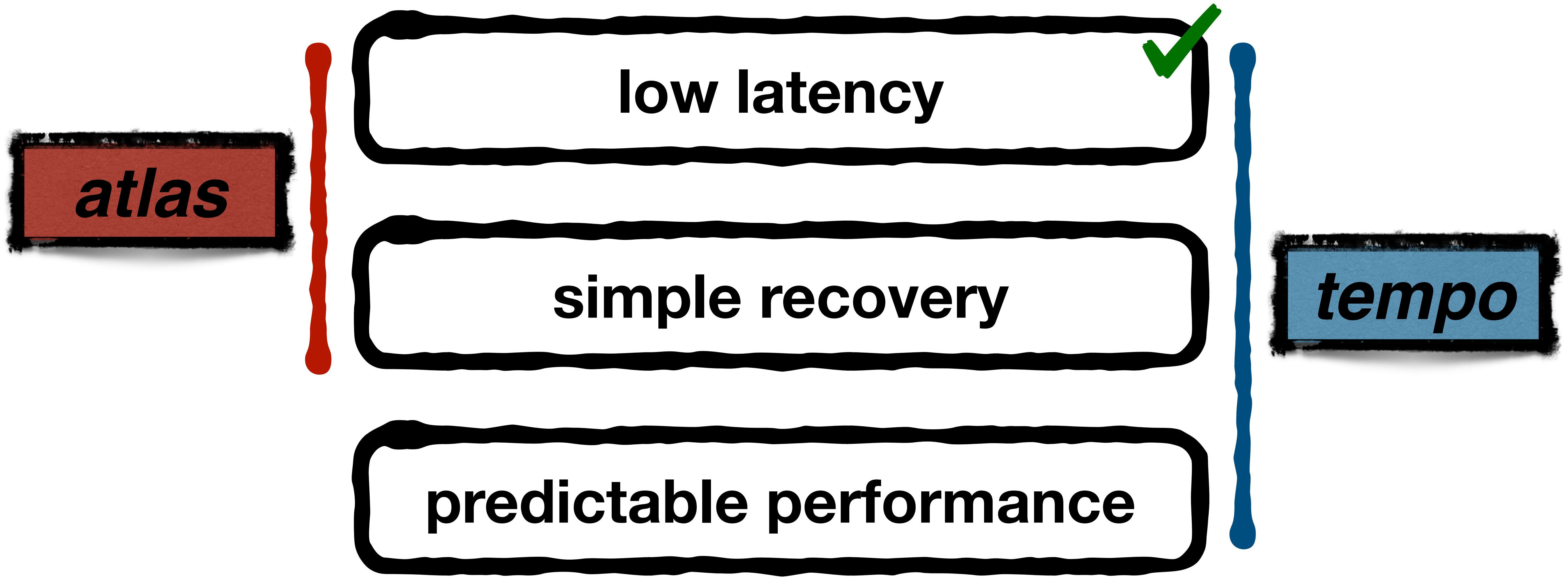
low latency

simple recovery

predictable performance

tempo

*can leaderless SMR be **practical** for planet-scale systems?*



low latency ✓

simple recovery

predictable performance

atlas

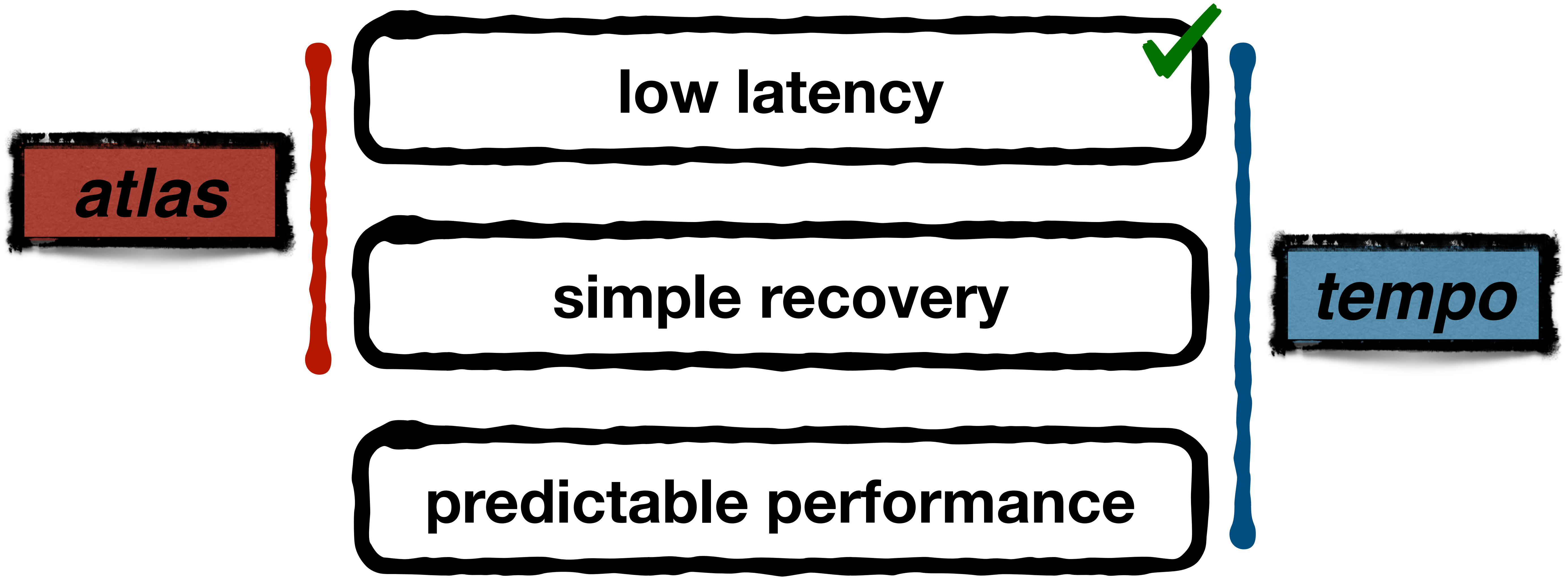
tempo

simple recovery

atlas *tempo*

- when a command is submitted, **the coordinator fixes the fast quorum**
- **recovery procedure reconstructs the committed value from within the fast quorum**
  - **epaxos** tries to recover from any quorum, which **makes recovery very complex**

can leaderless SMR be practical for planet-scale systems?

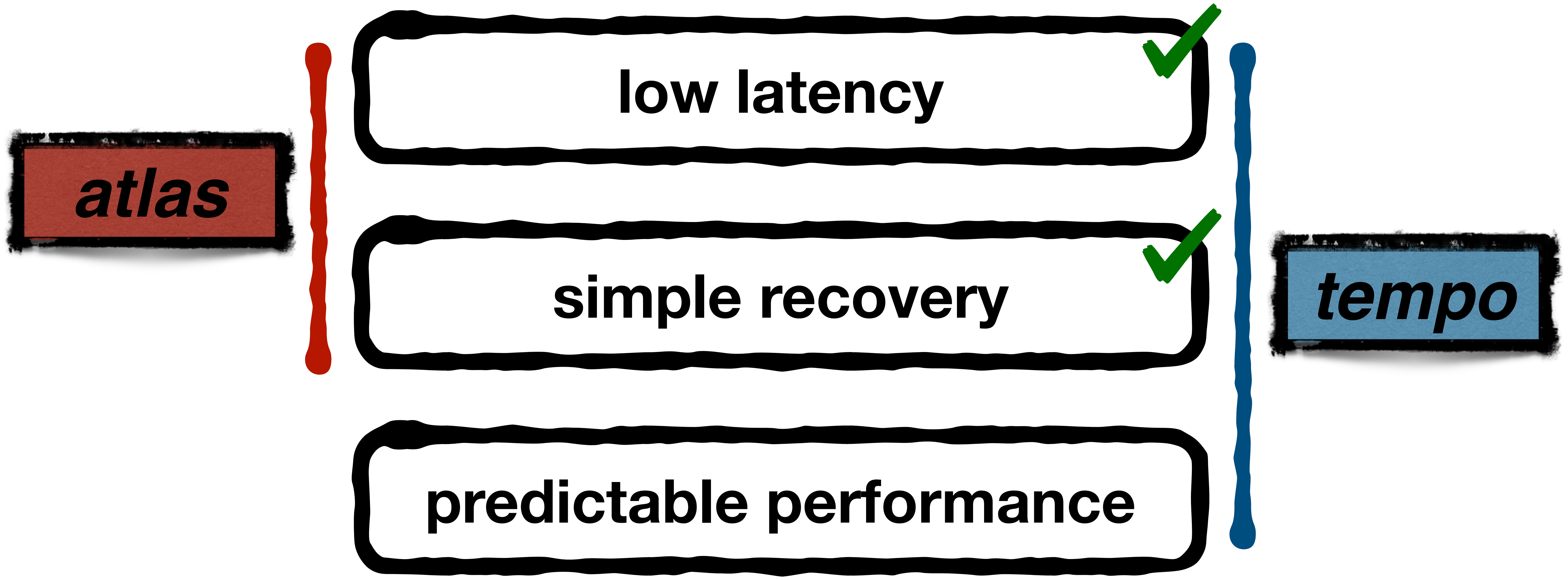


low latency ✓

simple recovery

predictable performance

can leaderless SMR be *practical* for planet-scale systems?



atlas

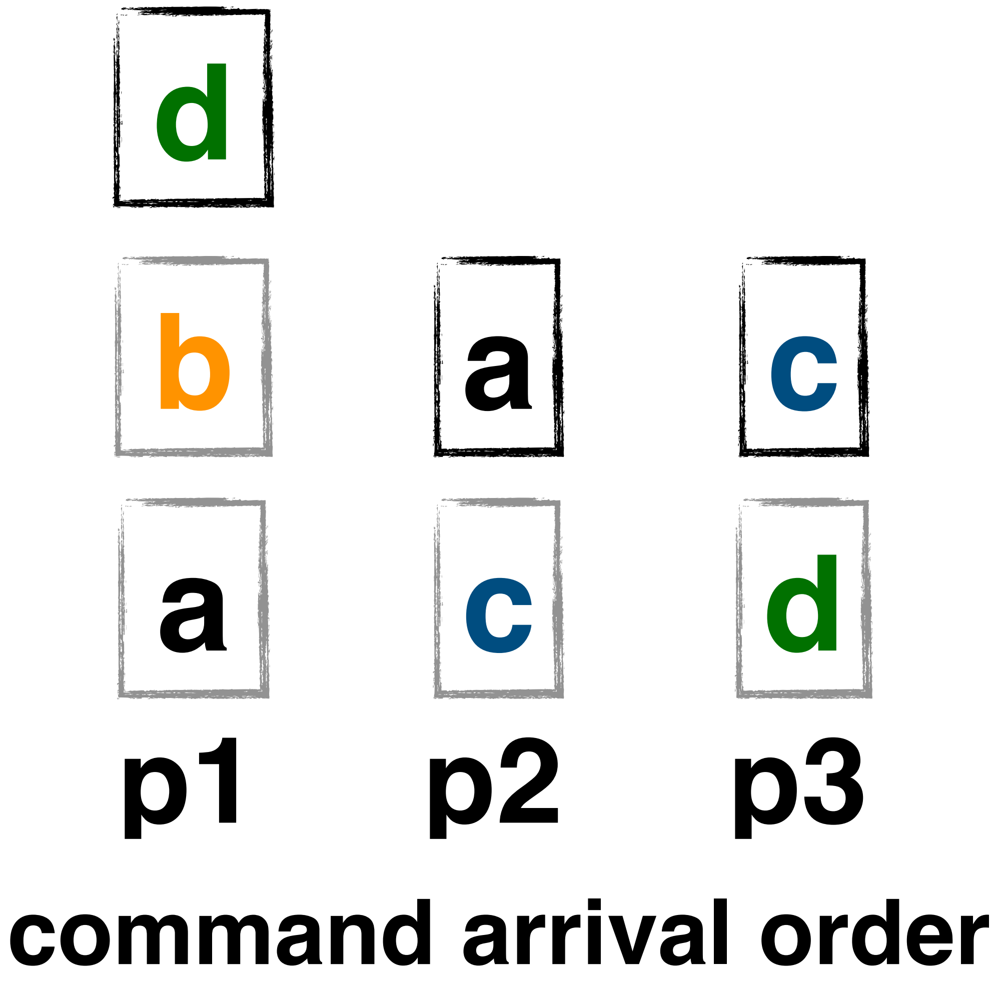
low latency ✓

simple recovery ✓

predictable performance

tempo

lack of predictable performance



d

b a c

a c d

p1 p2 p3

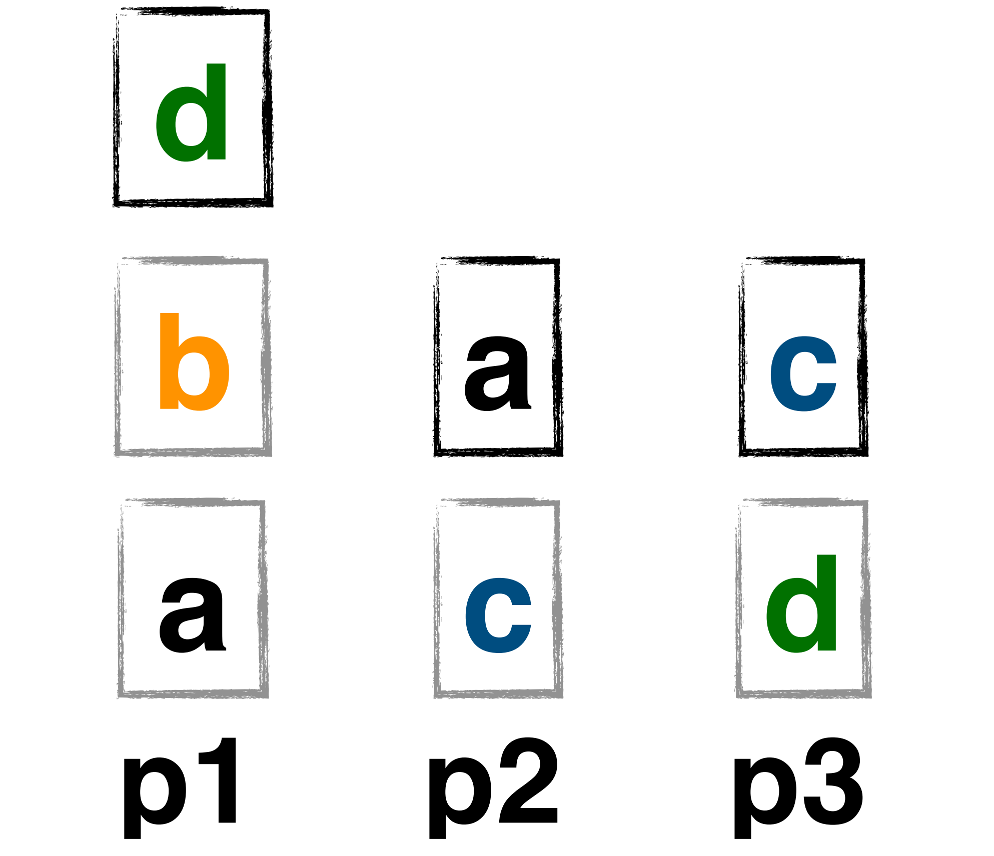
command arrival order

atlas

&

epaxos

lack of predictable performance



command arrival order

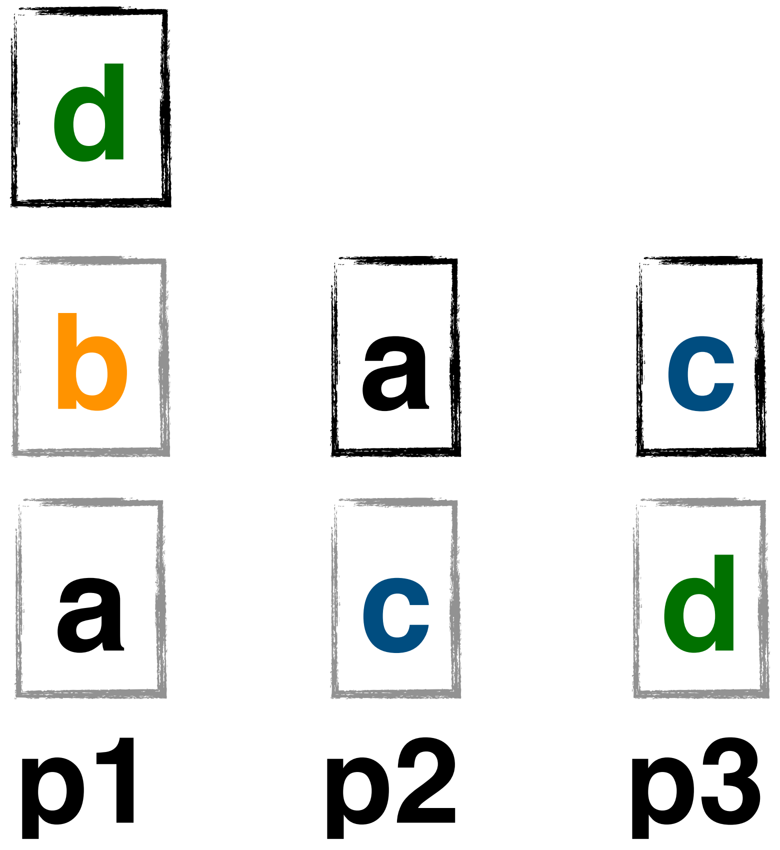
all commands but b are committed

atlas

&

epaxos

lack of predictable performance



d

b a c

a c d

p1 p2 p3

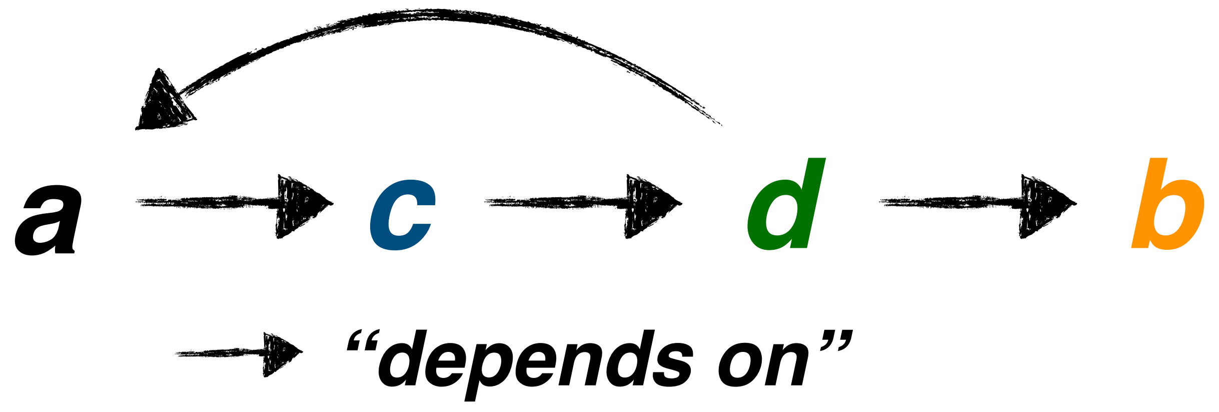
command arrival order

all commands but b are committed

atlas

&

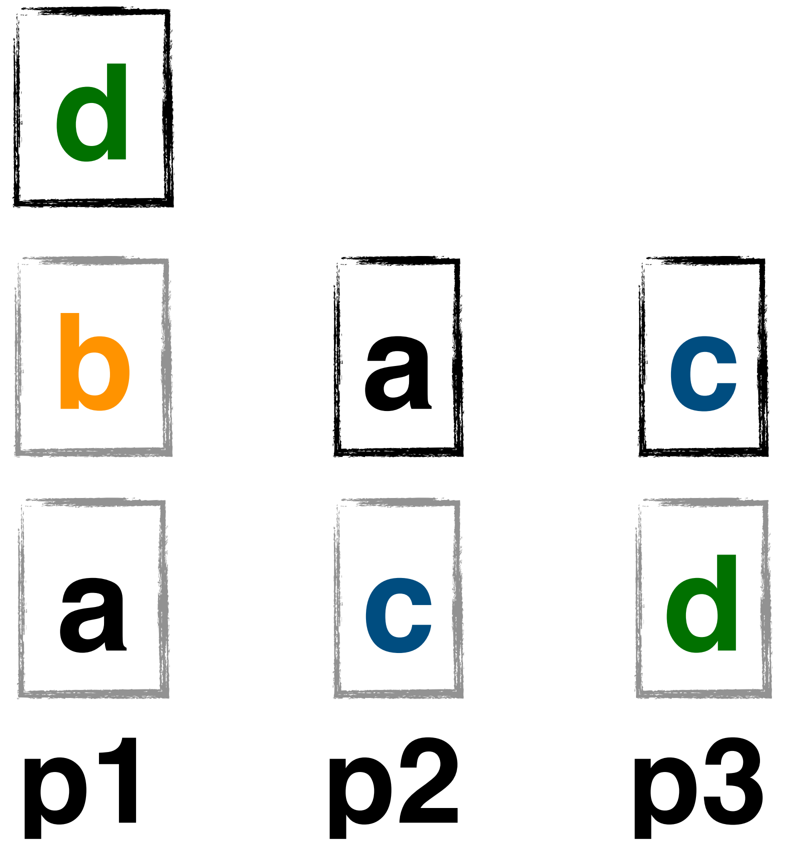
epaxos



a → *c* → *d* → *b*

→ *"depends on"*

lack of predictable performance



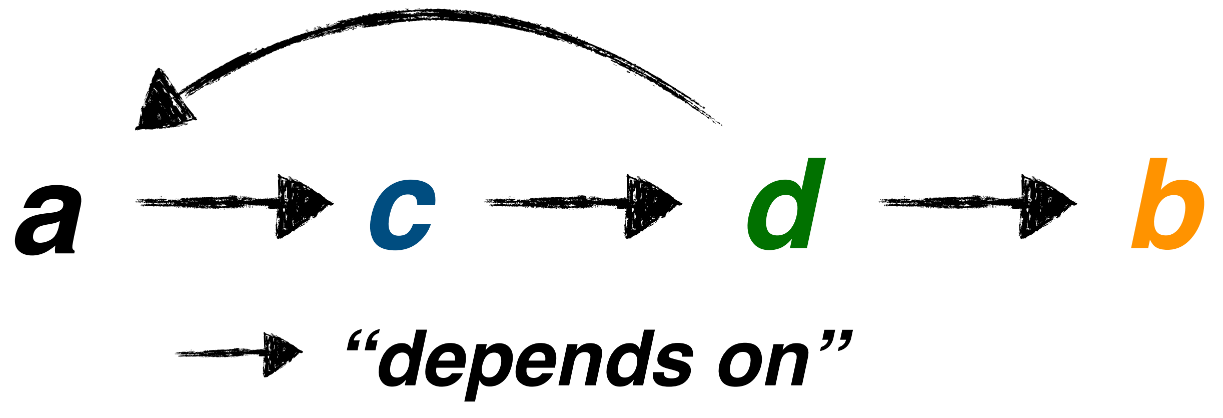
command arrival order

all commands but b are committed

atlas

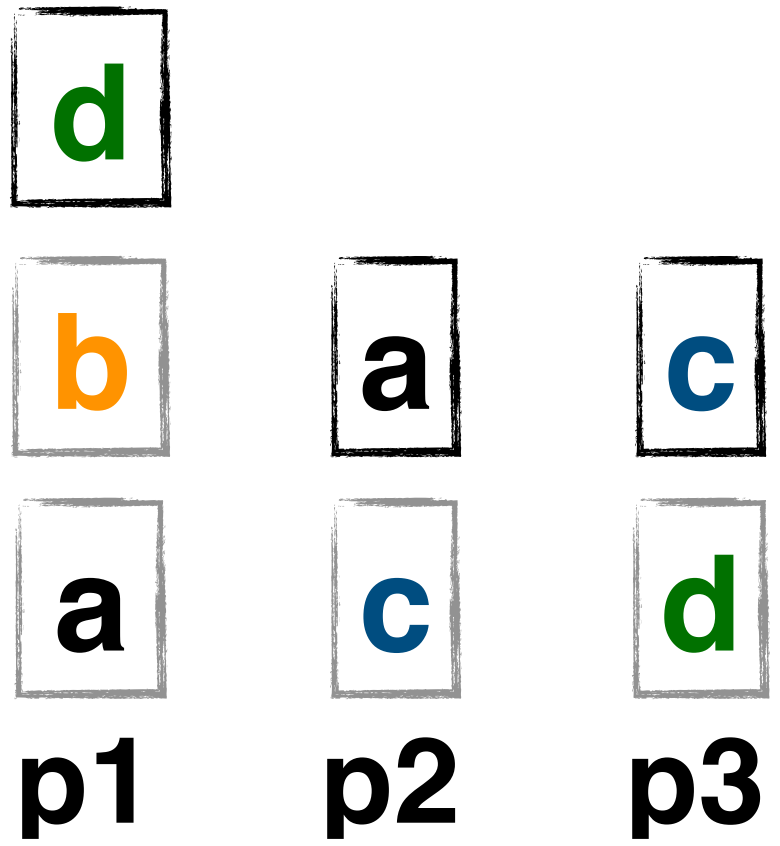
&

epaxos



no command is executed! ❌

lack of predictable performance



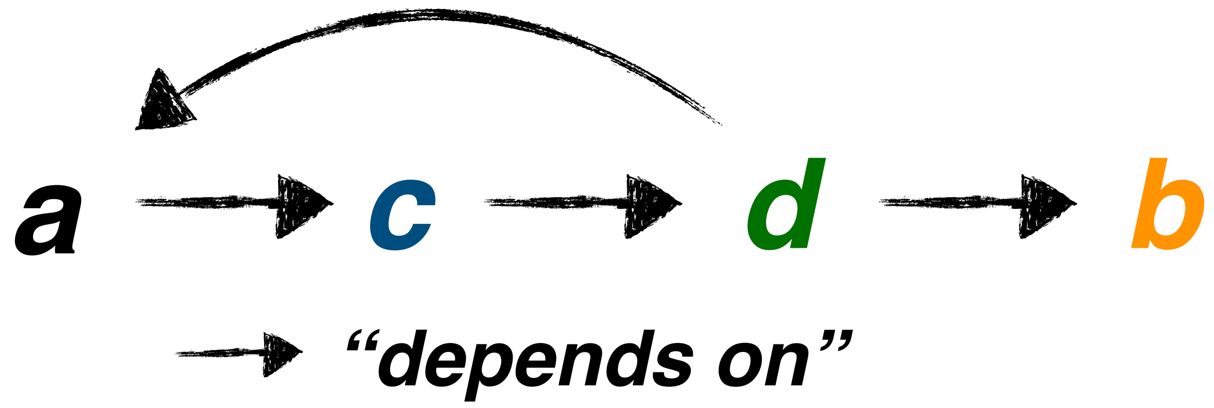
command arrival order

all commands but b are committed

atlas

&

epaxos



no command is executed!

theory: **don't terminate**
practice: **high tail latency**

lack of predictable performance

d		
b	a	c
a	c	d
p1	**p2**	**p3**

command arrival order

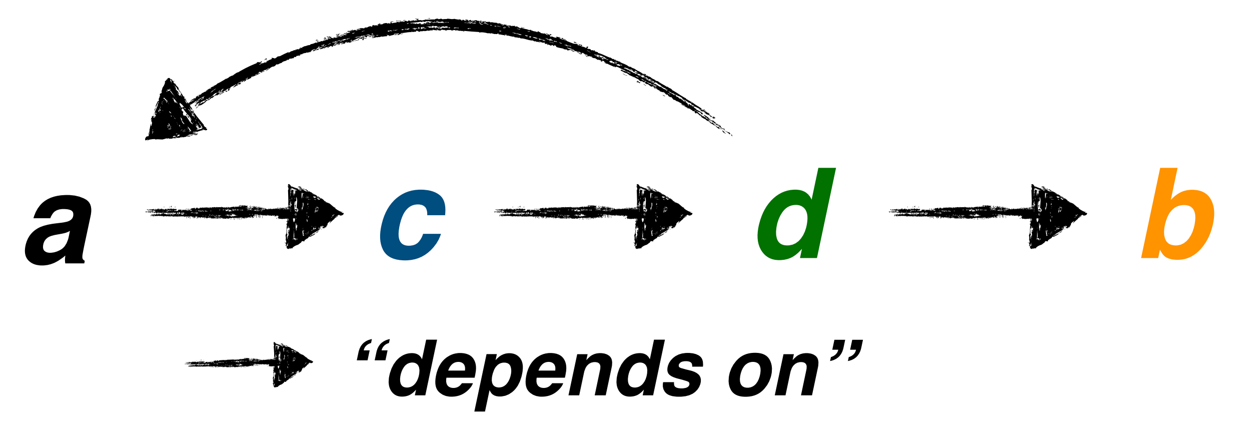
all commands but b are committed

tempo

atlas

&

epaxos



no command is executed!

theory: **don’t terminate**
practice: **high tail latency**

timestamping

tempo

- fast quorum processes propose a timestamp for the command

timestamping

tempo

- fast quorum processes propose a timestamp for the command
- the committed timestamp is the highest proposal

timestamping

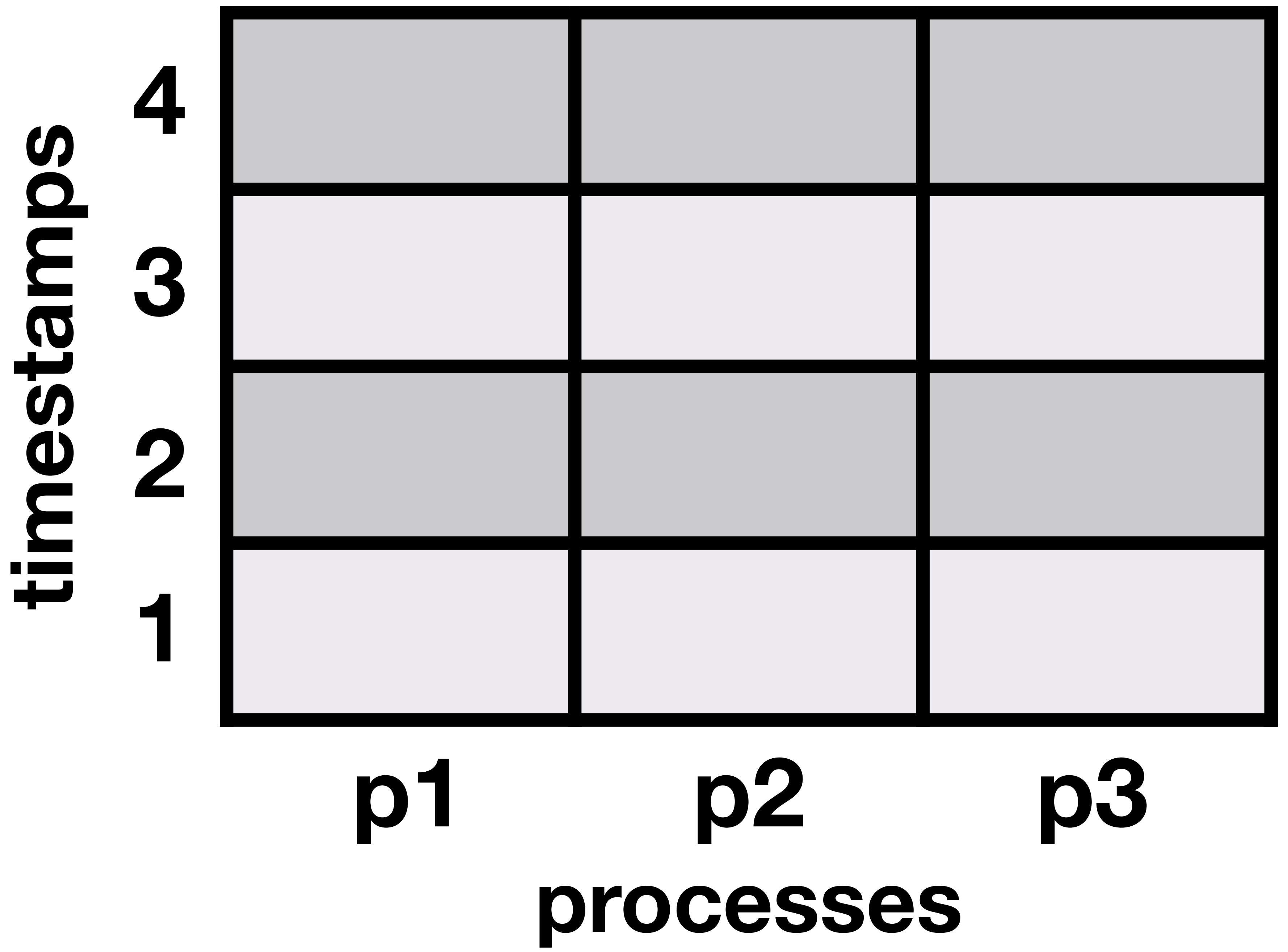
tempo

- fast quorum processes propose a timestamp for the command
- the committed timestamp is the highest proposal
- commands are executed in timestamp order

timestamping

tempo

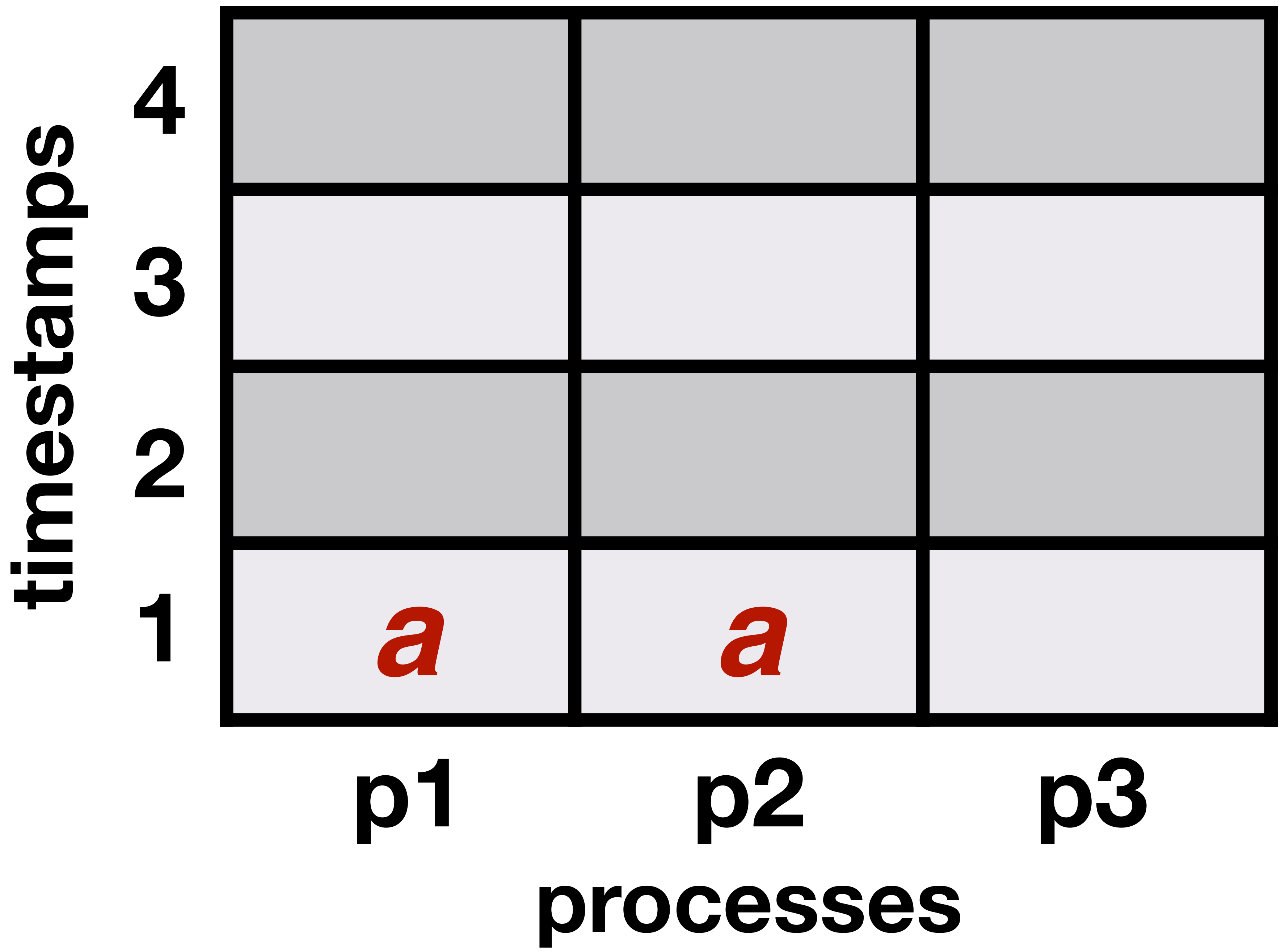
- fast quorum processes propose a timestamp for the command
- the committed timestamp is the highest proposal
- commands are executed in timestamp order



timestamping

tempo

- fast quorum processes propose a **timestamp** for the command
- the committed timestamp is the **highest** proposal
- commands are executed in timestamp order



timestamps	p1	p2	p3
4			
3			
2			
1	*a*	*a*	

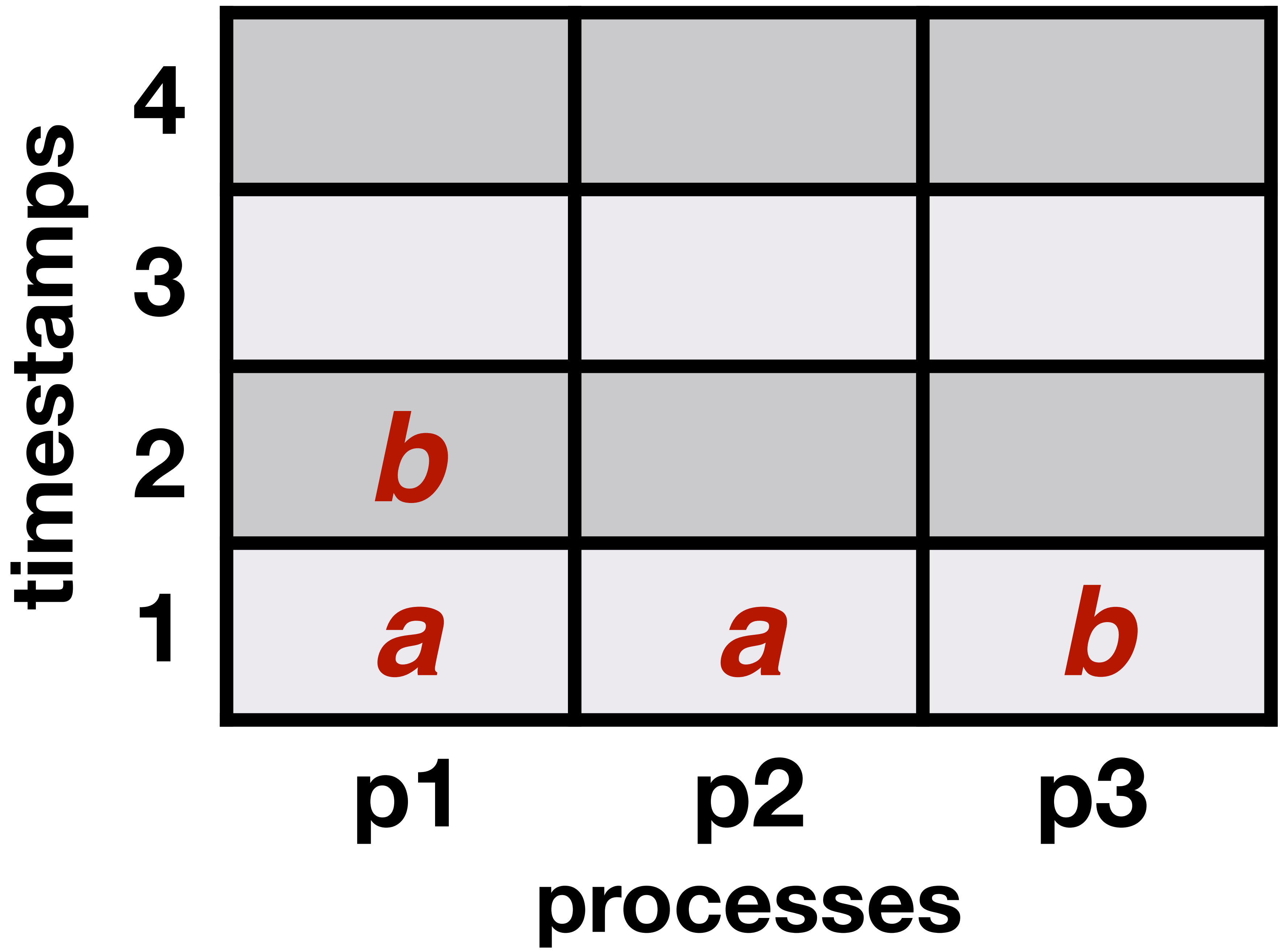
processes

ts[*a*] = 1

timestamping

tempo

- fast quorum processes propose a **timestamp** for the command
- the committed timestamp is the **highest** proposal
- commands are executed in timestamp order



timestamps	p1	p2	p3
4			
3			
2	*b*		
1	*a*	*a*	*b*

processes

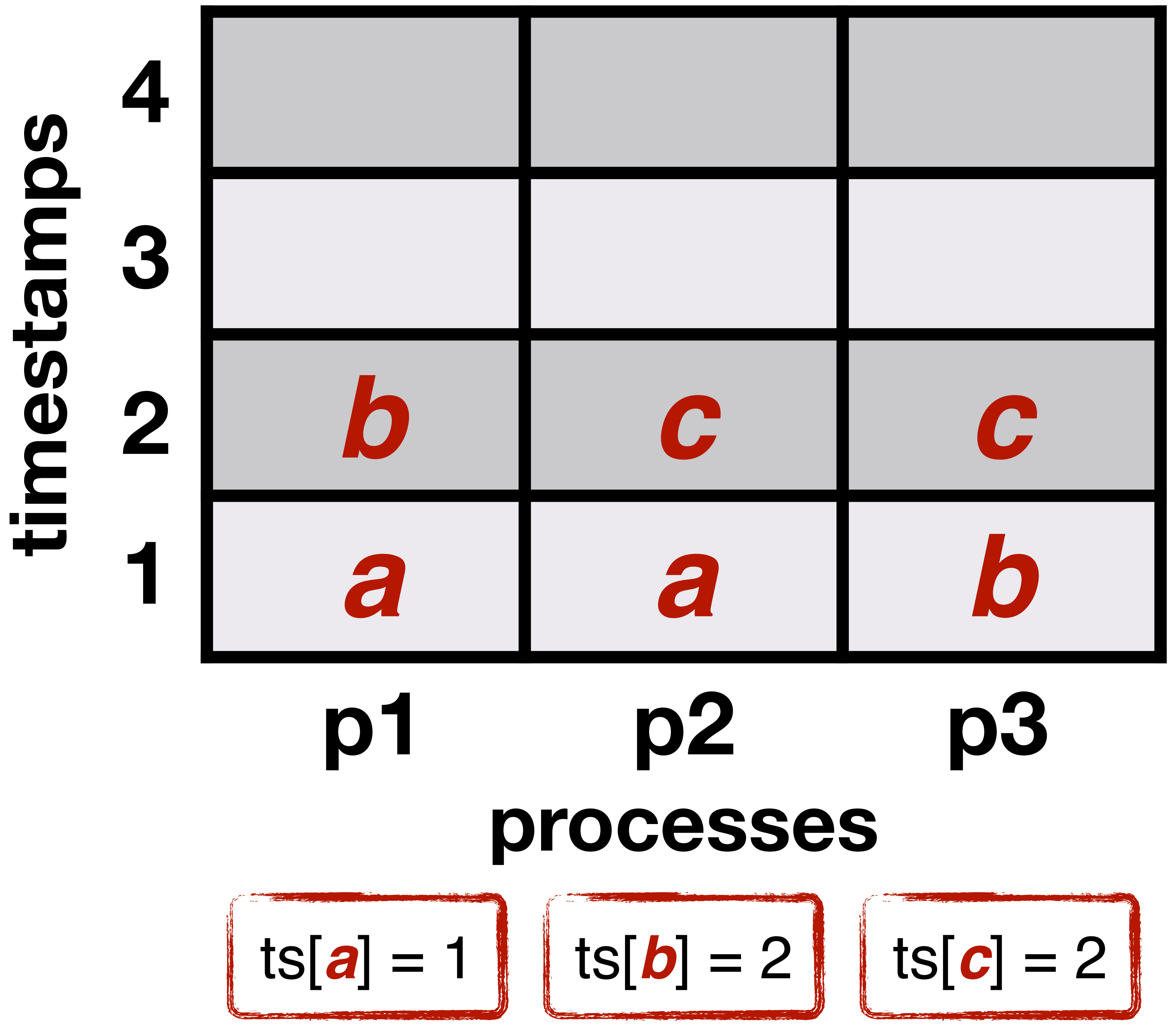
ts[*a*] = 1

ts[*b*] = 2

timestamping

tempo

- fast quorum processes propose a timestamp for the command
- the committed timestamp is the highest proposal
- commands are executed in timestamp order



timestamps	p1	p2	p3
4			
3			
2	*b*	*c*	*c*
1	*a*	*a*	*b*

ts[*a*] = 1

ts[*b*] = 2

ts[*c*] = 2

timestamping

tempo

- fast quorum processes propose a timestamp for the command
- the committed timestamp is the highest proposal
- commands are executed in timestamp order

timestamps	p1	p2	p3
4			
3			
2	*b*	*c*	*c*
1	*a*	*a*	*b*

processes

ts[*a*] = 1   ts[*b*] = 2   ts[*c*] = 2

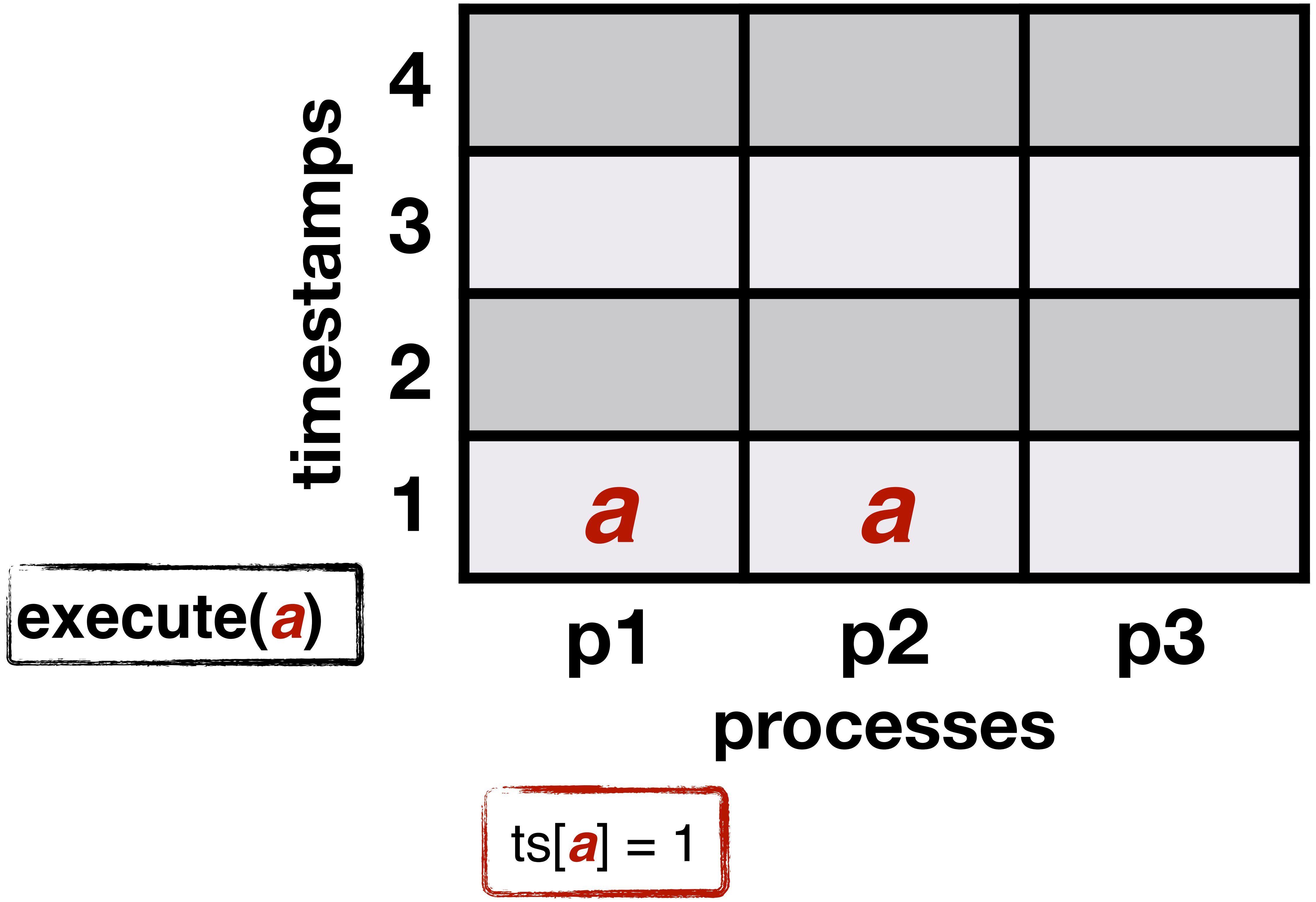
question: *when is it safe to execute a committed command?*

command execution

a process can only execute a command committed with timestamp t once it knows all proposals up to t by any majority

command execution

a process can only execute a command committed with timestamp t once it knows all proposals up to t by any majority



timestamps	p1	p2	p3
4			
3			
2			
1	*a*	*a*	

processes

execute(*a*)

ts[*a*] = 1

command execution

a process can only execute a command committed with timestamp t once it knows all proposals up to t by any majority

timestamps	p1	p2	p3
4			
3			
2	*b*		
1	*a*	*a*	*b*

processes

execute(*a*)

ts[*a*] = 1

ts[*b*] = 2

command execution

a process can only execute a command committed with **timestamp t** once it knows **all proposals up to t by any majority**

timestamps	p1	p2	p3
4			
3			
2	*b*	*c*	*c*
1	*a*	*a*	*b*

processes

execute(*a*)

execute(*b*) ; execute(*c*)
since *b* < *c*

ts[*a*] = 1

ts[*b*] = 2

ts[*c*] = 2

predictable performance

p1	p2	p3
d		
b	a	c
a	c	d

command arrival order

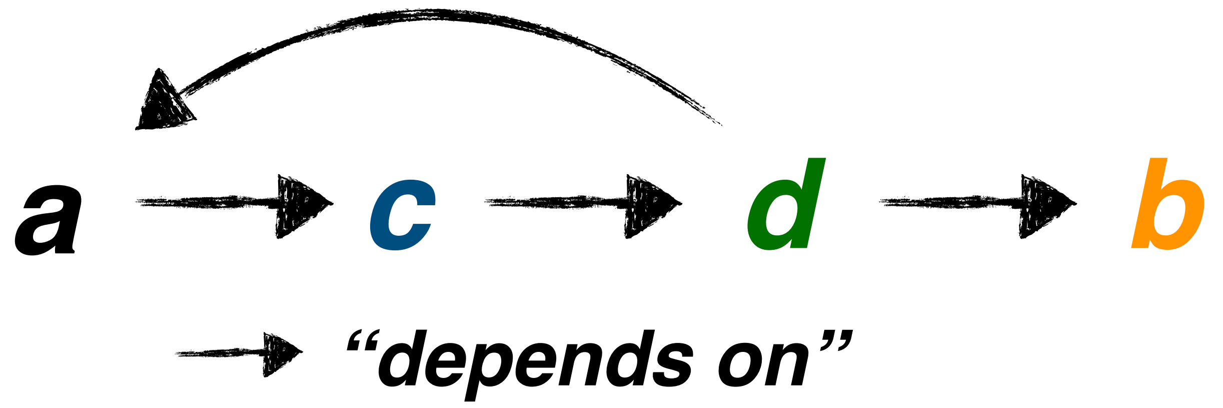
all commands but b are committed

tempo

atlas

&

epaxos



no command is executed!

theory: **don't terminate**
practice: **high tail latency**

predictable performance

p1	p2	p3
d		
b	a	c
a	c	d

command arrival order

all commands but b are committed

tempo

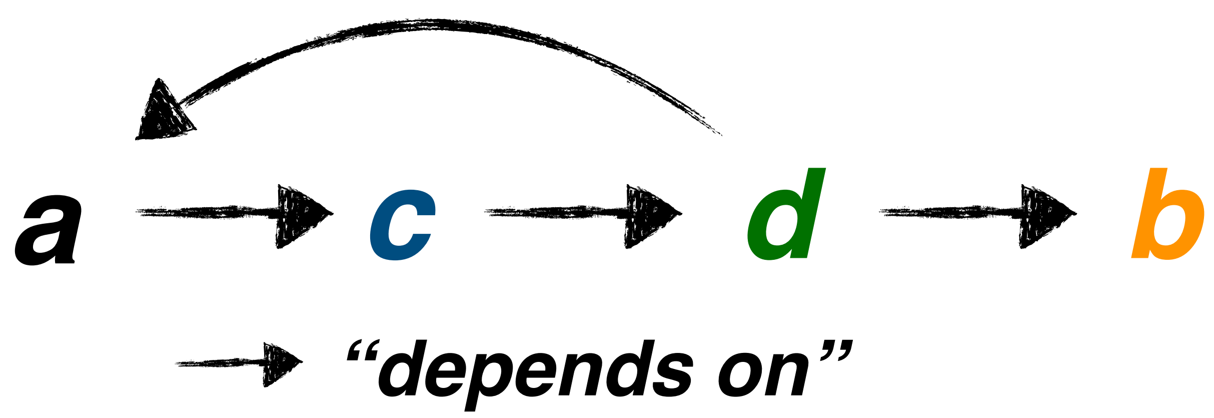
timestamps	p1	p2	p3
3	*d*		
2		*a*	*c*
1	*a*	*c*	*d*

processes

atlas

&

epaxos



no command is executed!

theory: **don’t terminate**
practice: **high tail latency**

predictable performance

d		
b	a	c
a	c	d
p1	p2	p3

command arrival order

all commands but b are committed

tempo

timestamps			
3	*d*		
2		*a*	*c*
1	*a*	*c*	*d*
	p1	p2	p3

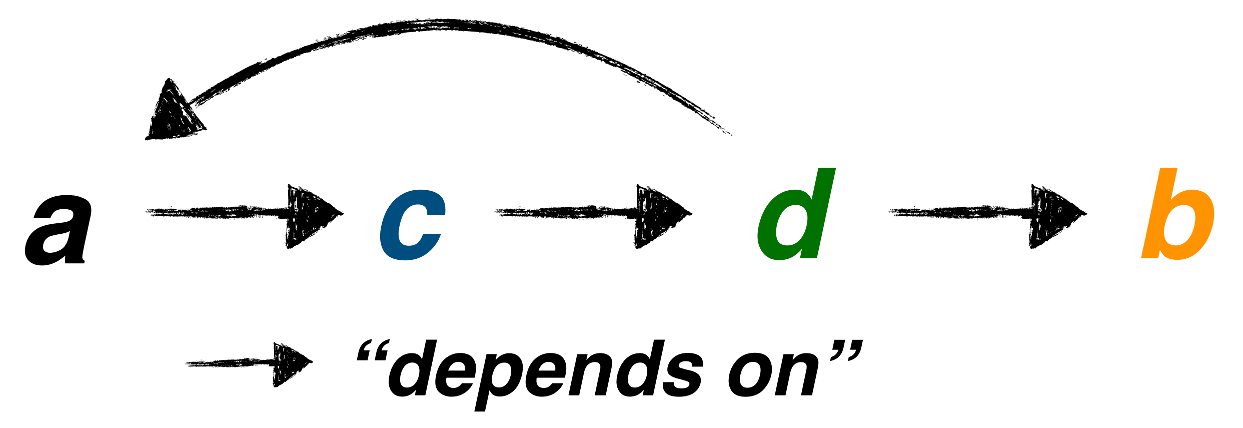
processes

have all proposals up to 2 by a majority, so tempo executes ***a*** and ***c*** ✓

atlas

&

epaxos



no command is executed! ✗

theory: **don't terminate**
practice: **high tail latency** ✗

predictable / superior performance

tempo

parallelism:

- timestamping & command execution are fully decentralized & parallel

predictable / superior performance

tempo

parallelism:

- timestamping & command execution are fully decentralized & parallel

scale vertically ↕

predictable / superior performance

tempo

scale vertically ↕

parallelism:

- timestamping & command execution are fully decentralized & parallel

in epaxos & atlas, command execution is sequential!!

predictable / superior performance

tempo

parallelism:

- timestamping & command execution are fully decentralized & parallel

scale vertically ↕

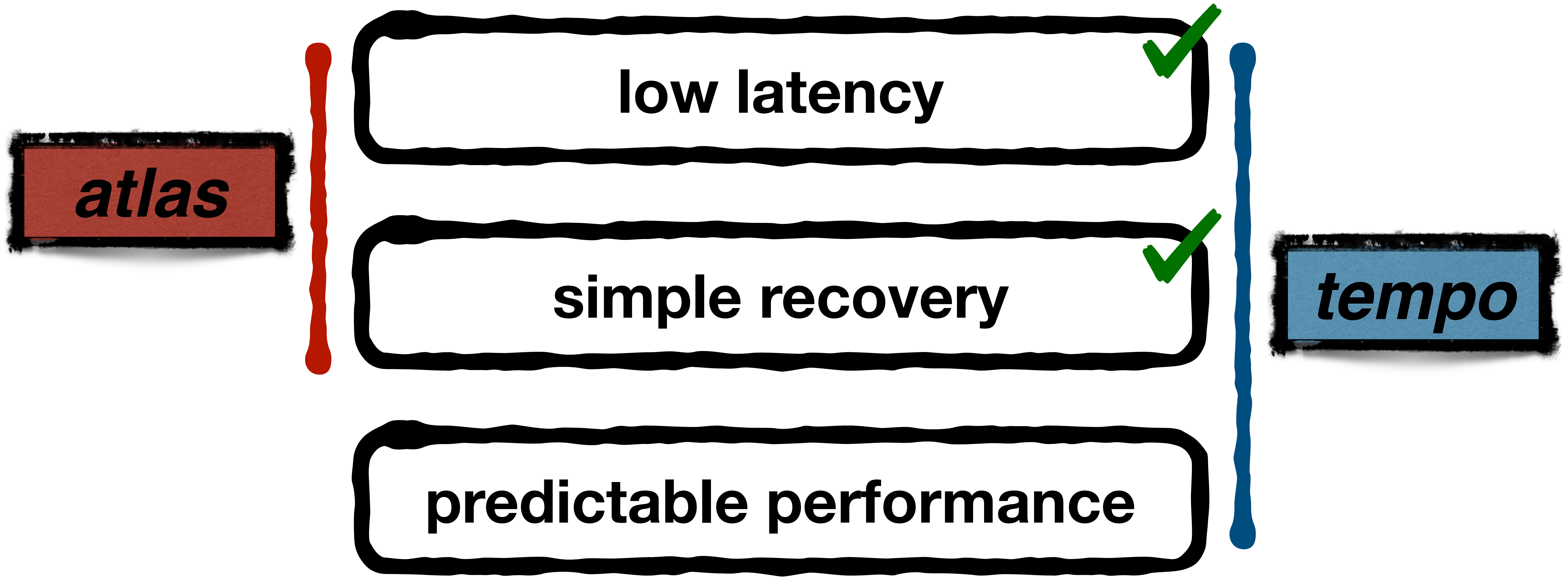
in epaxos & atlas, command execution is sequential!!

partial replication:

- the protocol easily generalizes to this setting

scale horizontally ↔

*can leaderless SMR be **practical** for planet-scale systems?*



atlas

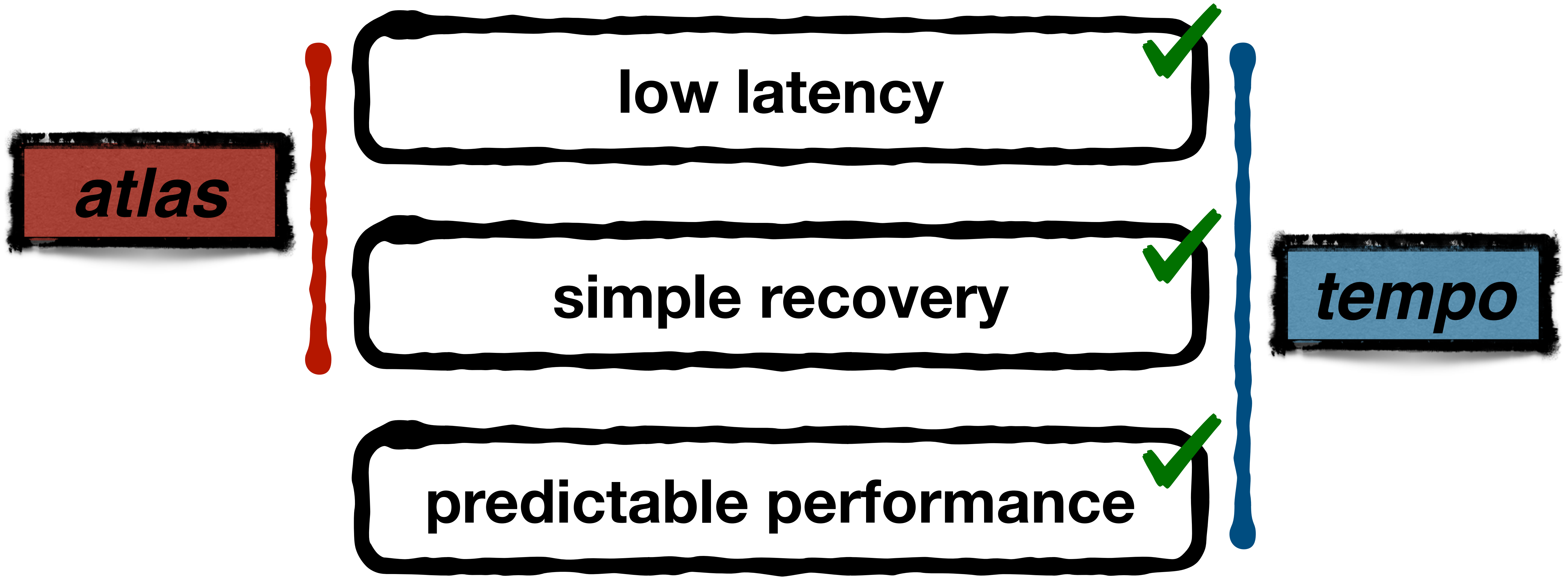
low latency ✓

simple recovery ✓

predictable performance

tempo

can leaderless SMR be practical for planet-scale systems?



atlas

low latency ✓

simple recovery ✓

predictable performance ✓

tempo

evaluation

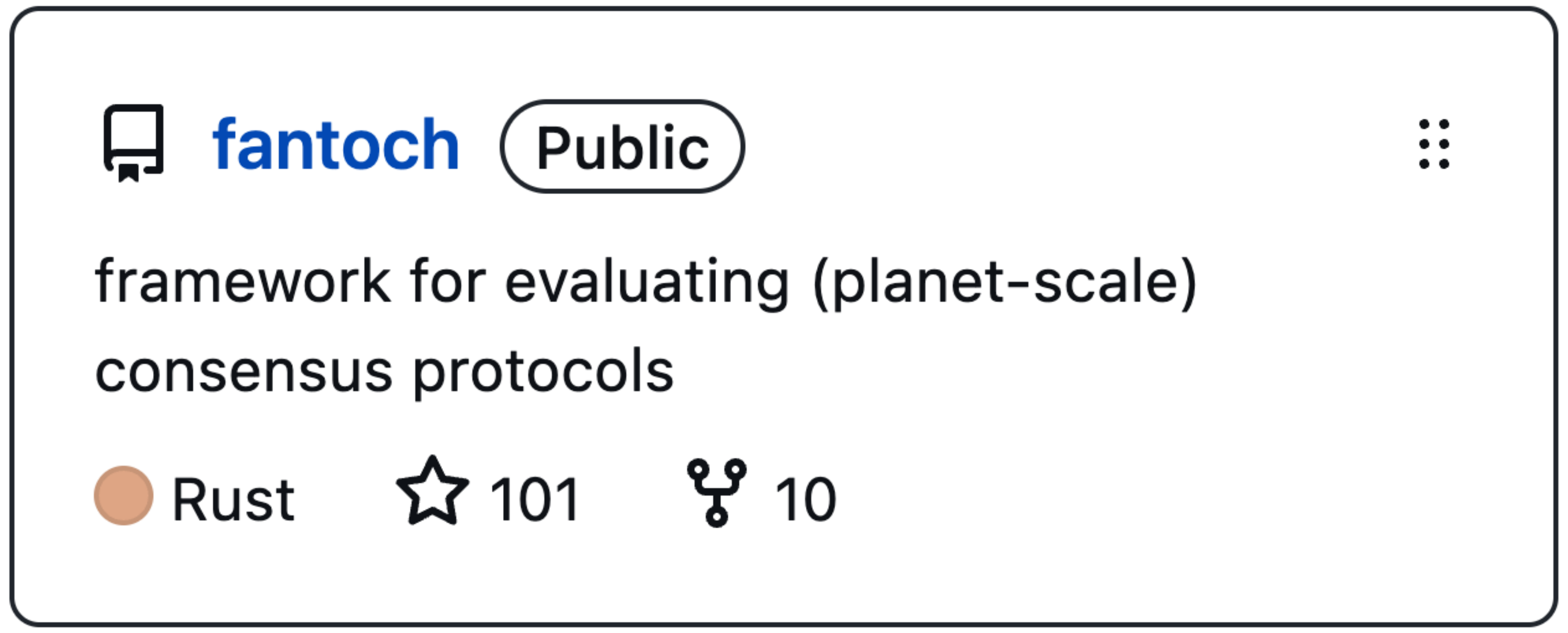
protocols considered:

- (flexible) paxos
- epaxos
- caesar (not in this presentation)
- janus (not in this presentation)
- ***atlas***
- ***tempo***

evaluation

protocols considered:

- (flexible) paxos
- epaxos
- caesar (not in this presentation)
- janus (not in this presentation)
- ***atlas***
- ***tempo***



fantoch Public

framework for evaluating (planet-scale) consensus protocols

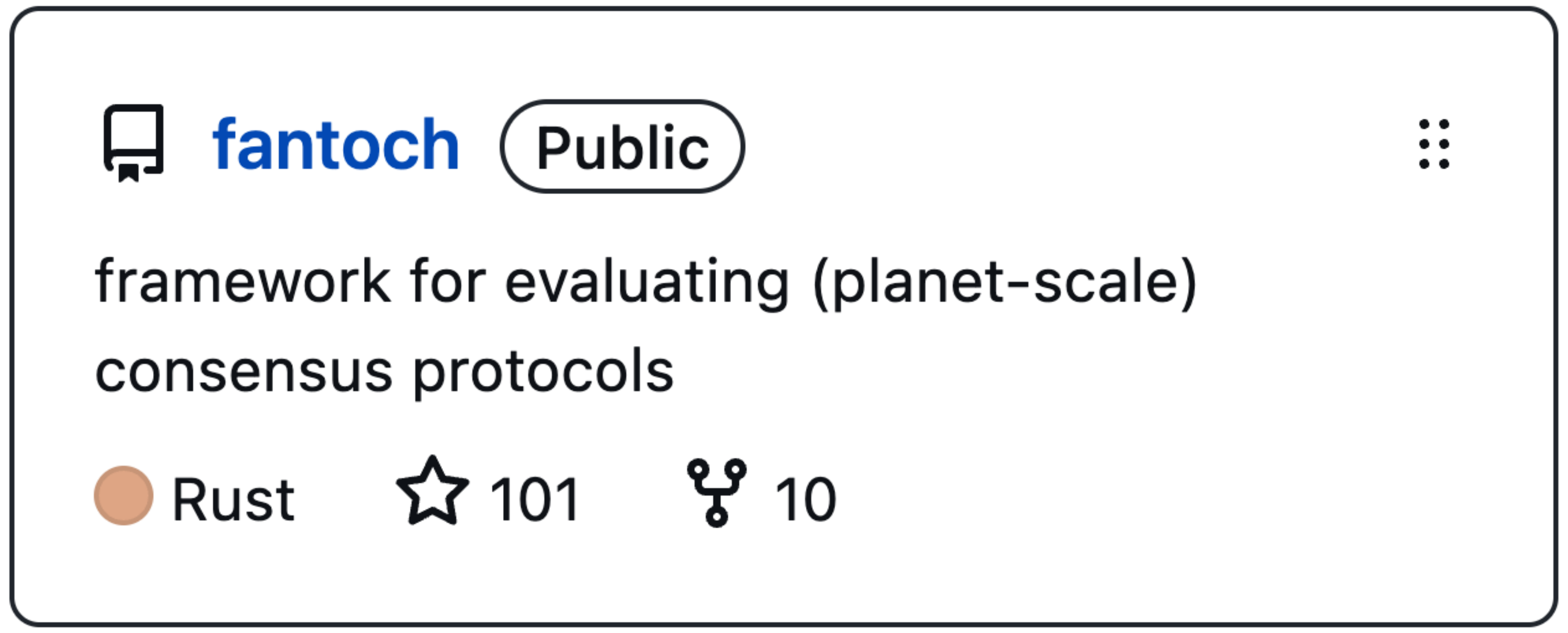
Rust ☆ 101 10

github.com/vitorenesduarte/fantoch

evaluation

protocols considered:

- (flexible) paxos
- epaxos
- caesar (not in this presentation)
- janus (not in this presentation)
- ***atlas***
- ***tempo***



fantoch Public

framework for evaluating (planet-scale) consensus protocols

● Rust ☆ 101 10

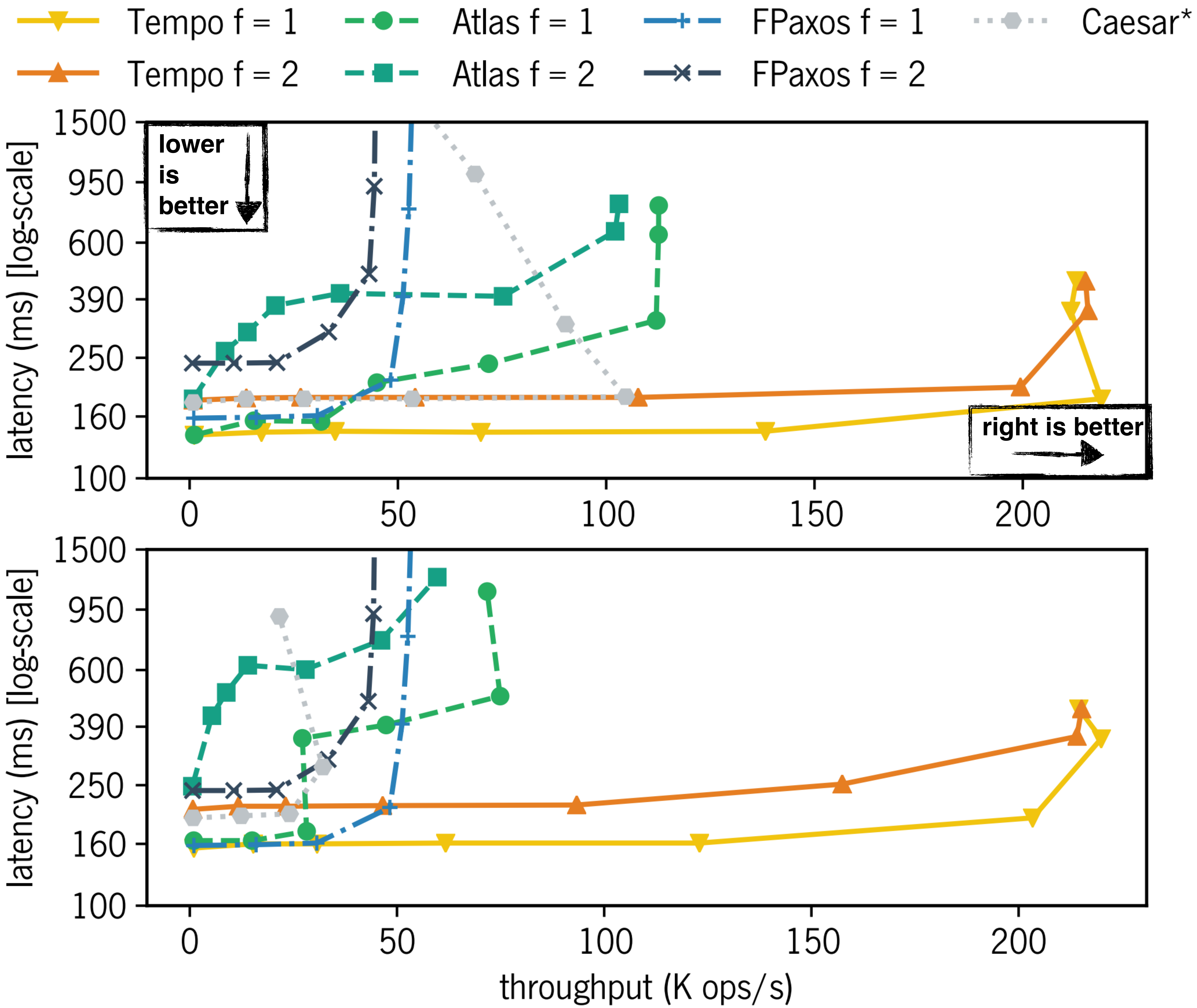
github.com/vitorenesduarte/fantoch

focus on predictable performance:

- throughput
- tail latency

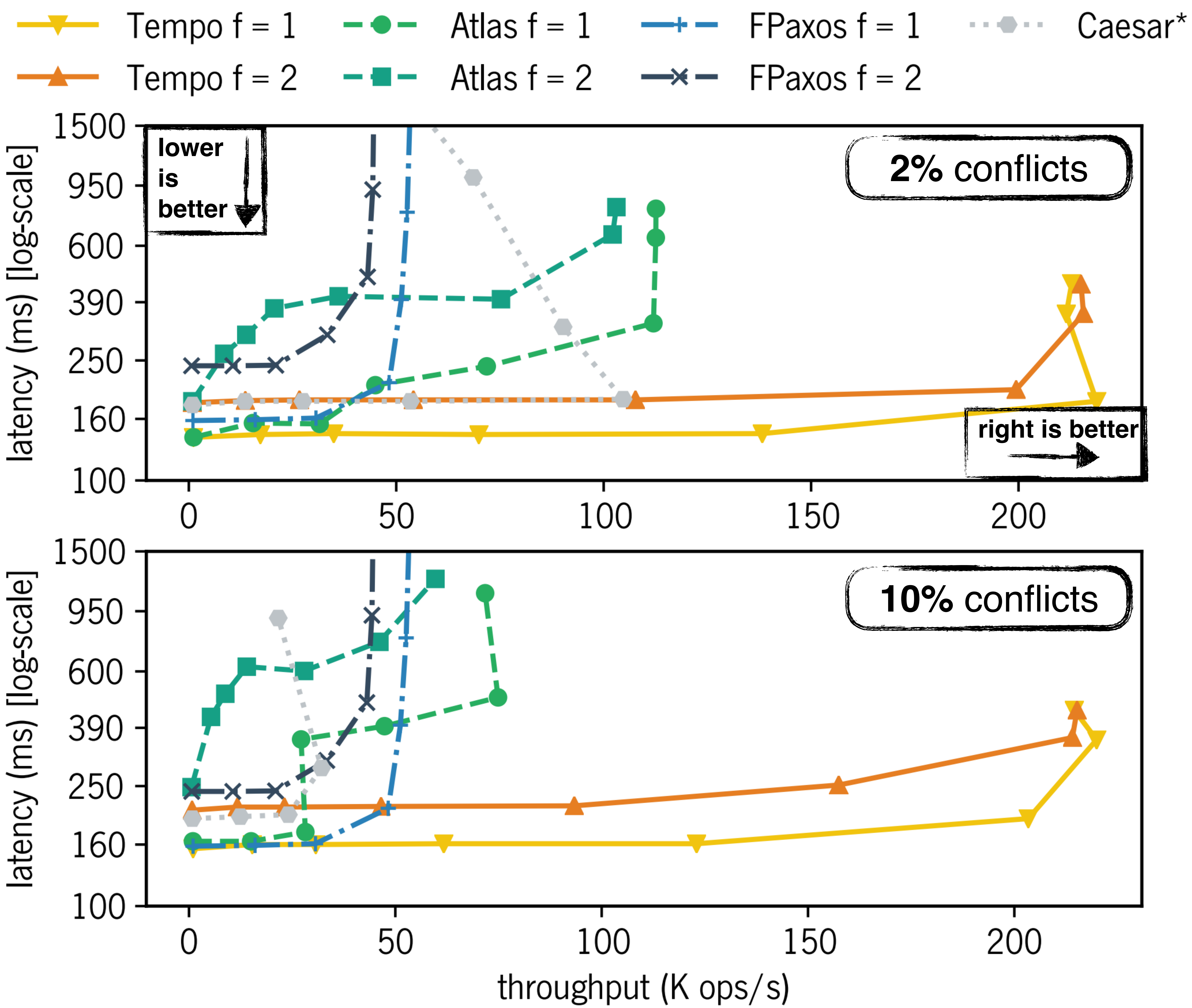
throughput

r = 5
clients per replica: 32 -> 20k



throughput

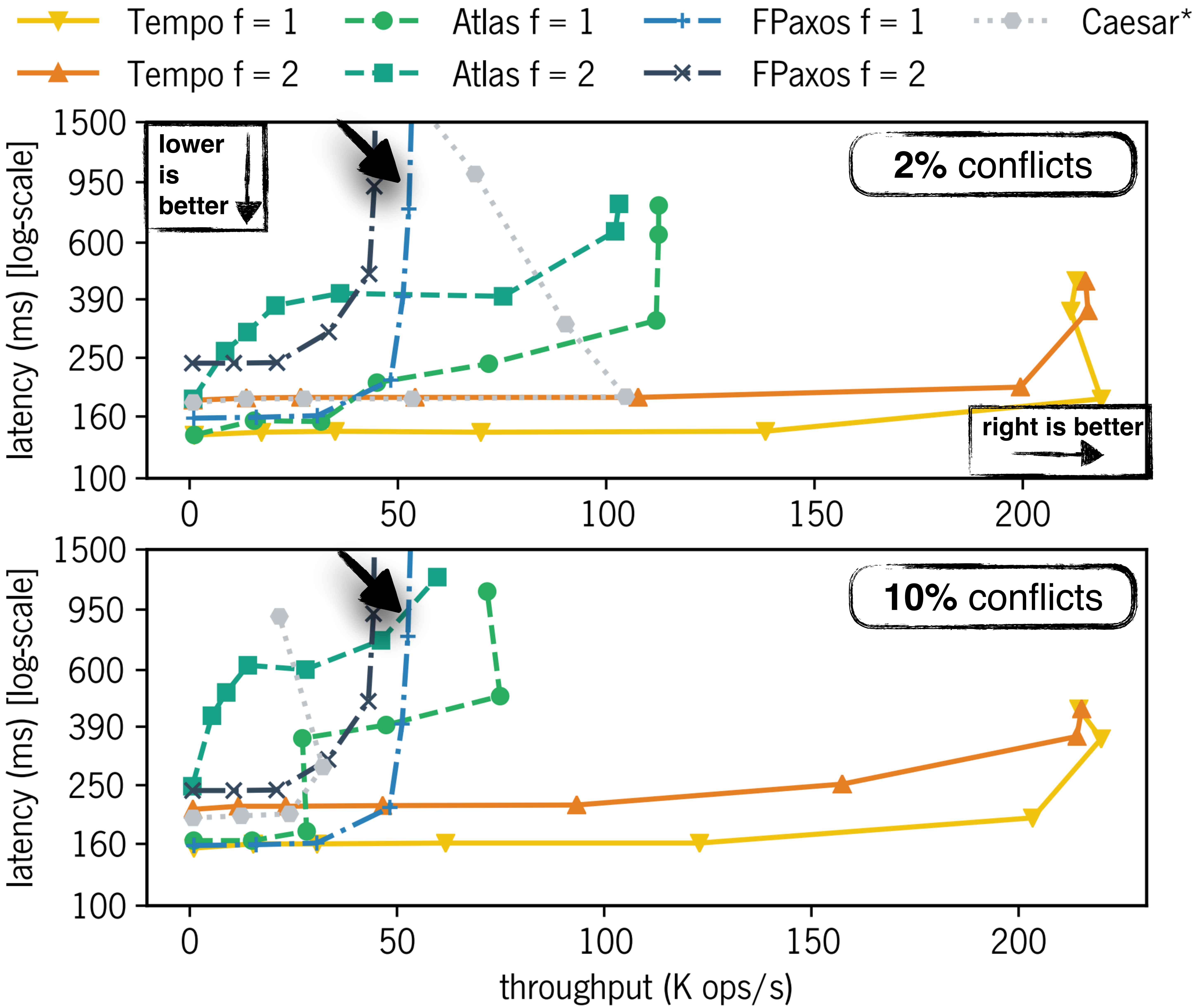
r = 5
clients per replica: 32 -> 20k



throughput

r = 5
clients per replica: 32 -> 20k

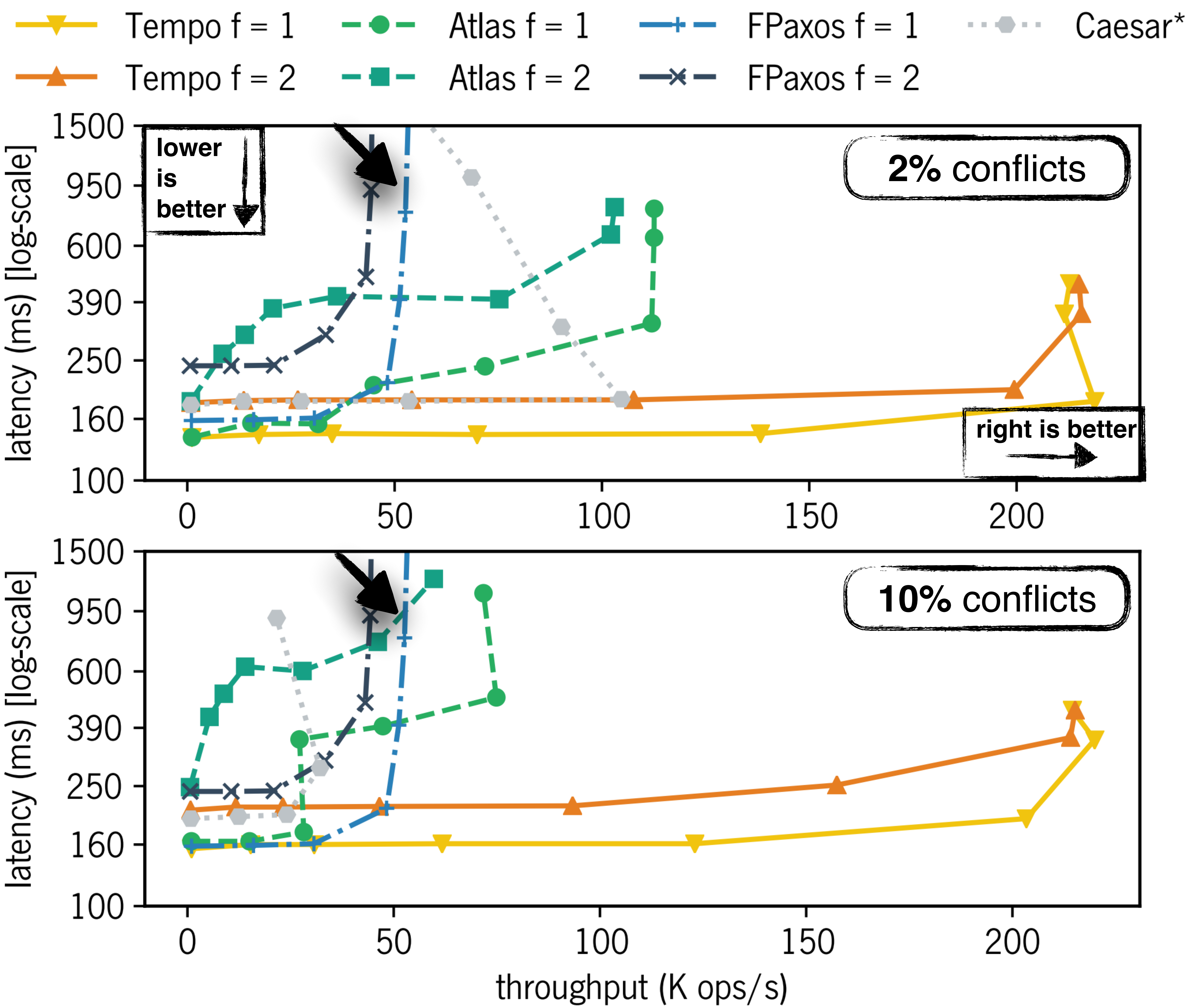
ops/s	2%	10%
fpaxos f =1	53K	53K



throughput

r = 5
clients per replica: 32 -> 20k

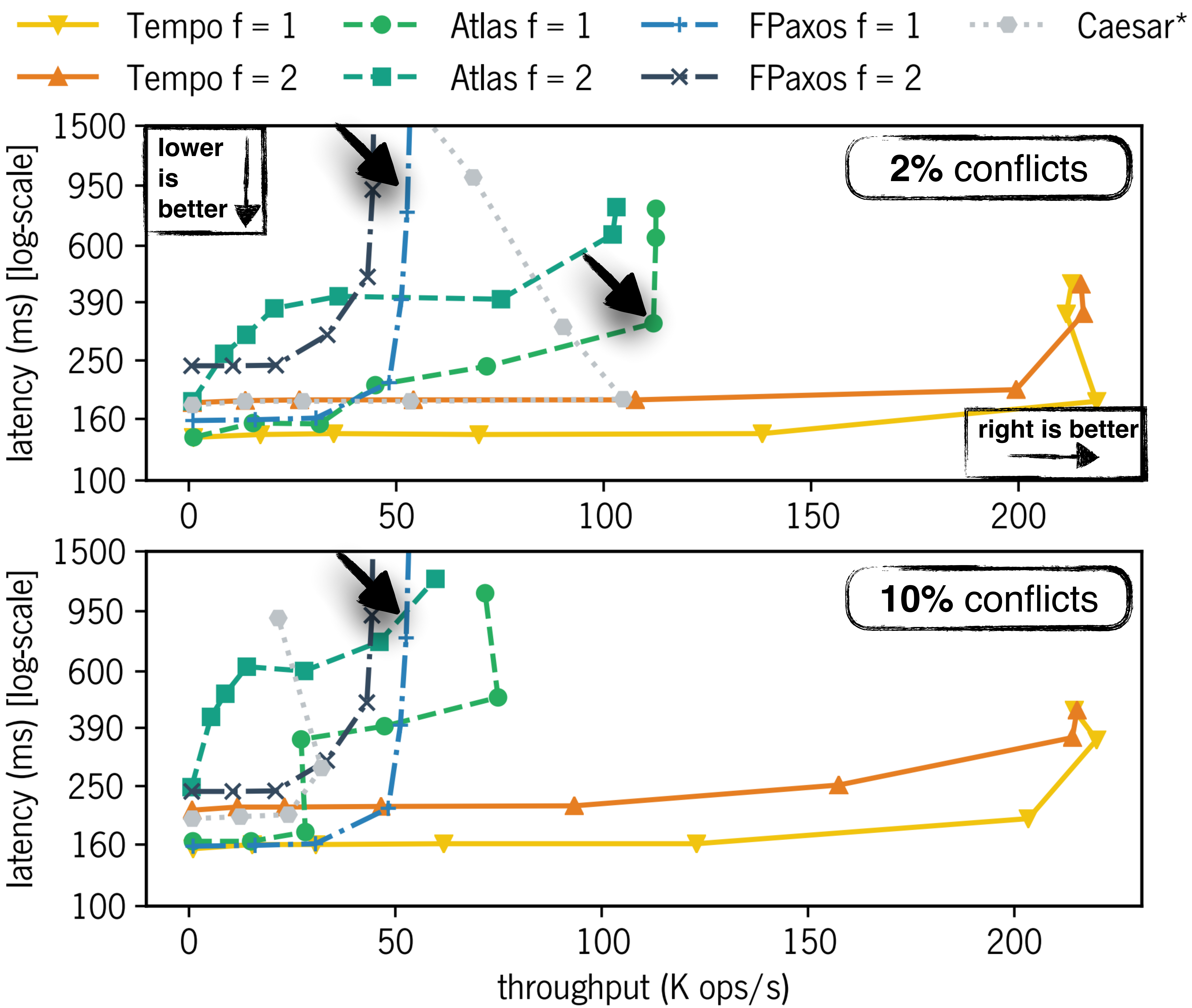
ops/s	2%	10%
fpaxos f =1	53K	53K
atlas f=1	129K	83K



throughput

r = 5
clients per replica: 32 -> 20k

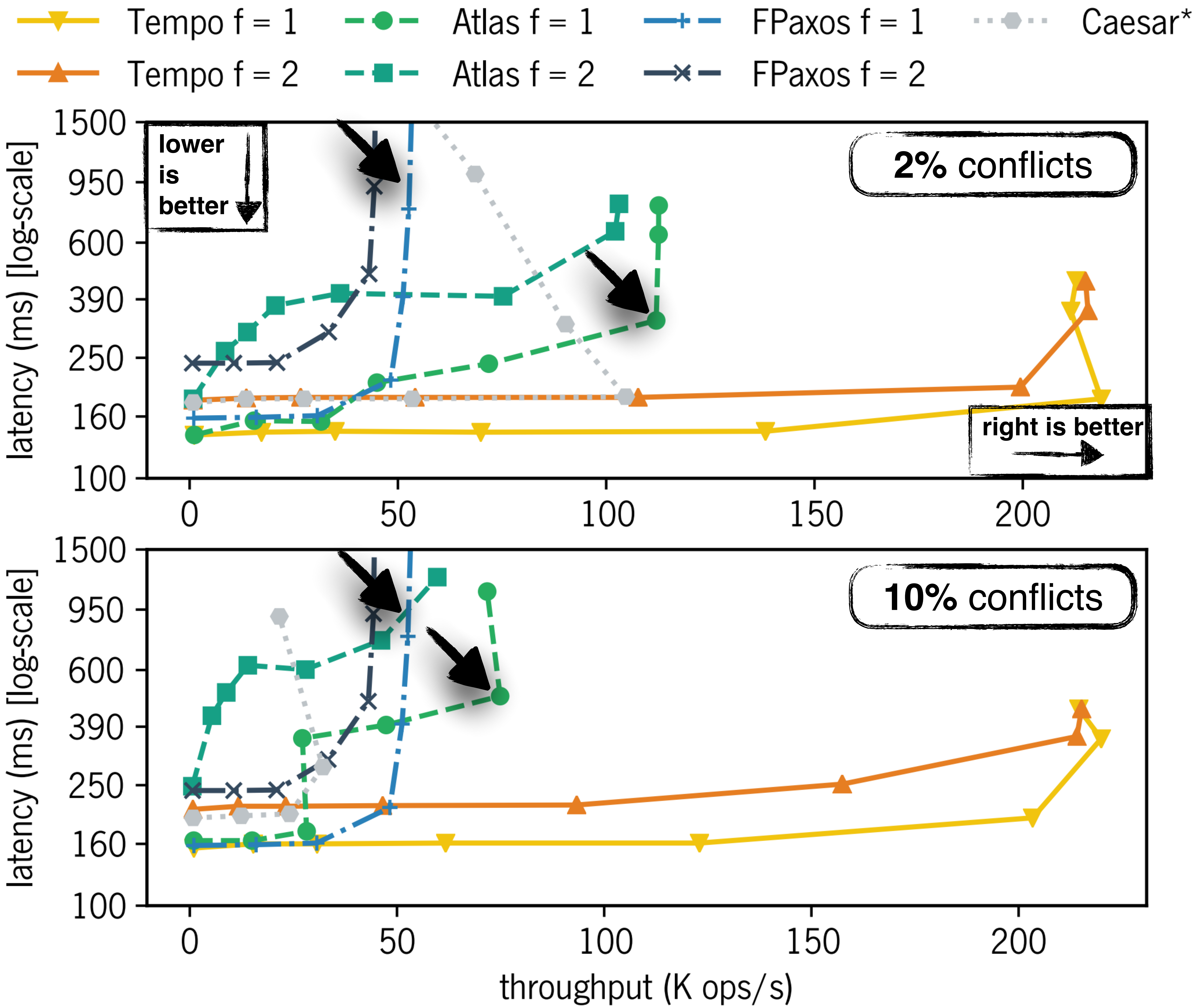
ops/s	2%	10%
fpaxos f =1	53K	53K
atlas f=1	129K	83K



throughput

r = 5
clients per replica: 32 -> 20k

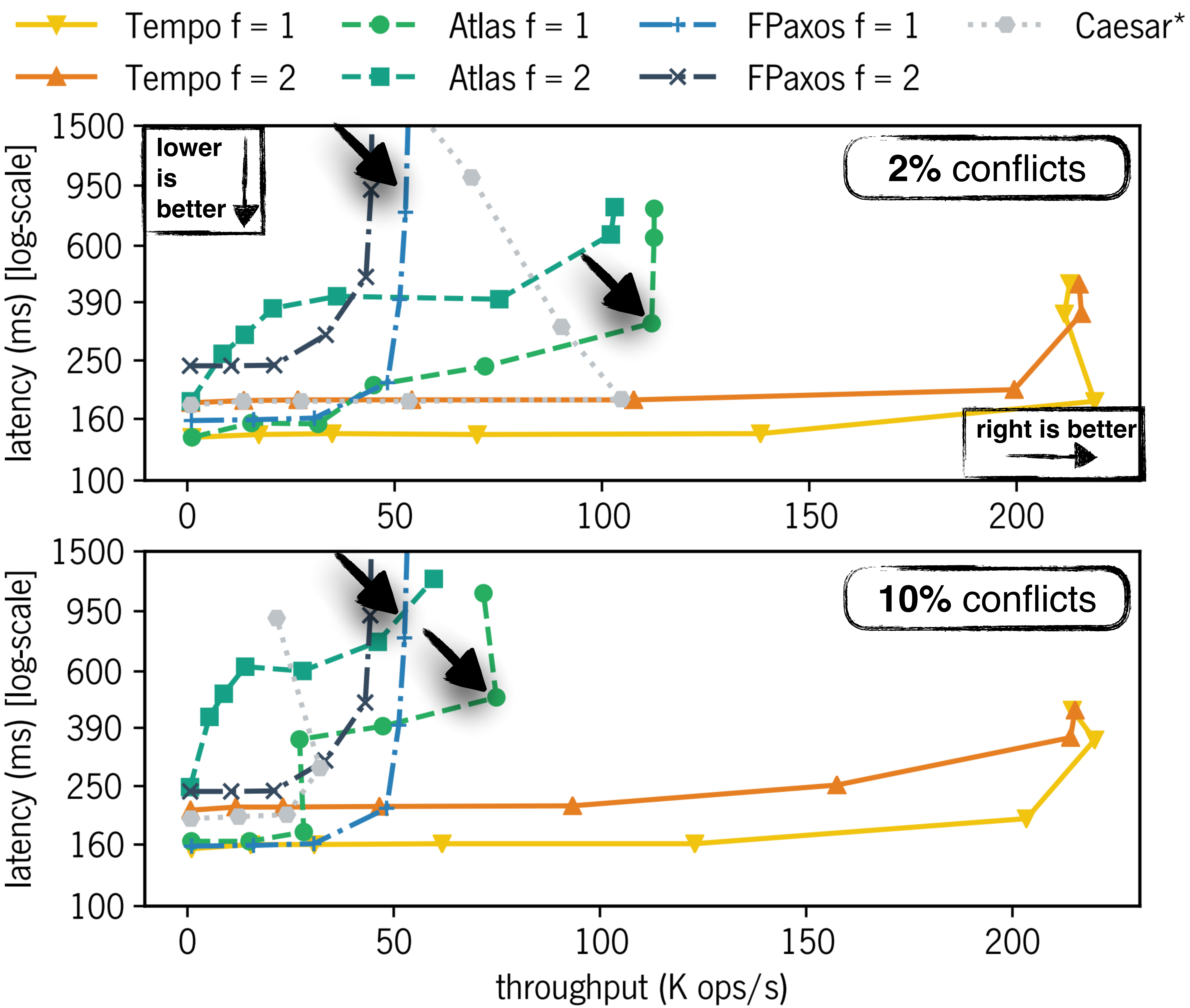
ops/s	2%	10%
fpaxos f =1	53K	53K
atlas f=1	129K	83K



throughput

r = 5
clients per replica: 32 -> 20k

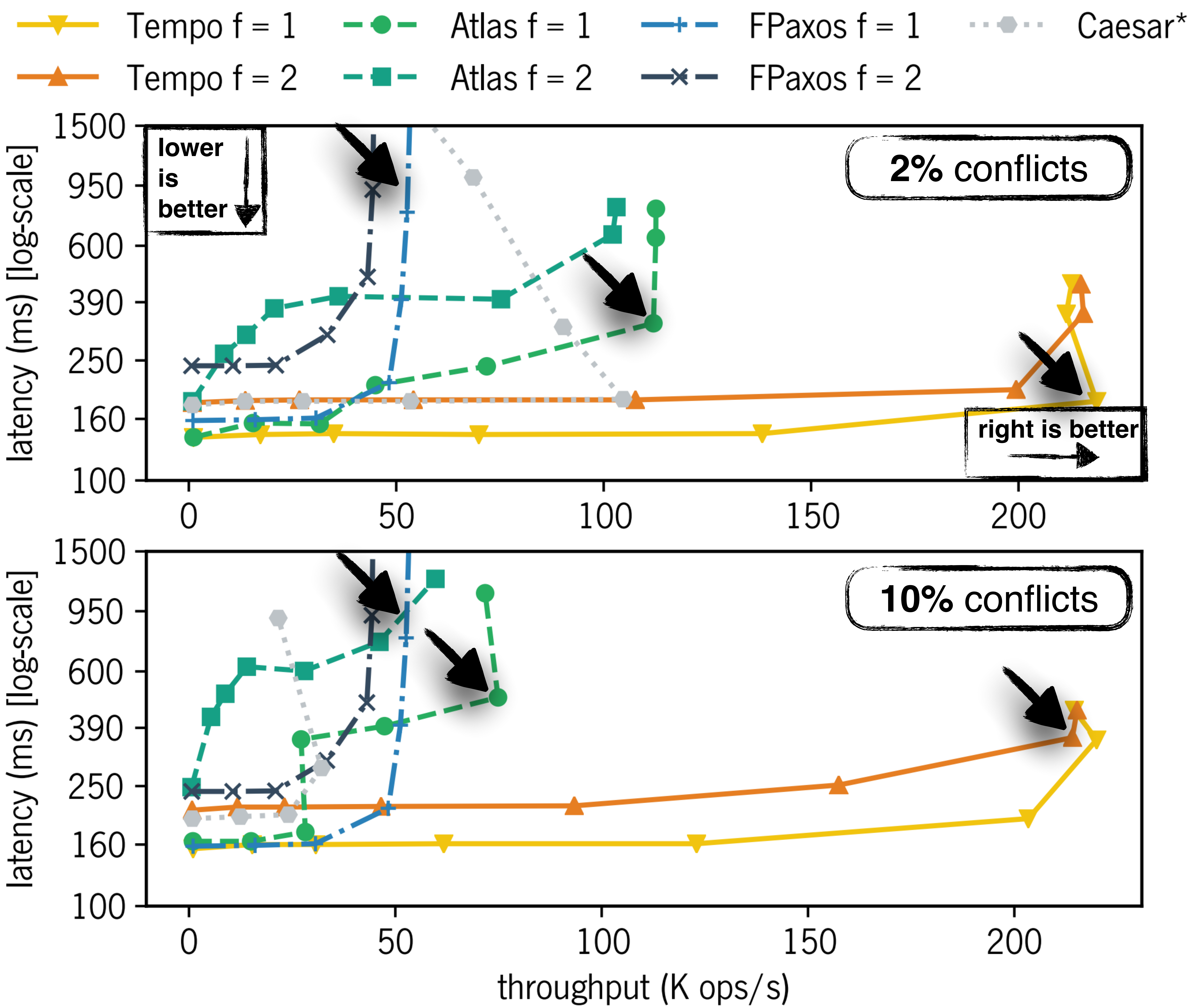
ops/s	2%	10%
fpaxos f =1	53K	53K
atlas f=1	129K	83K
tempo f=1	229K	230K



throughput

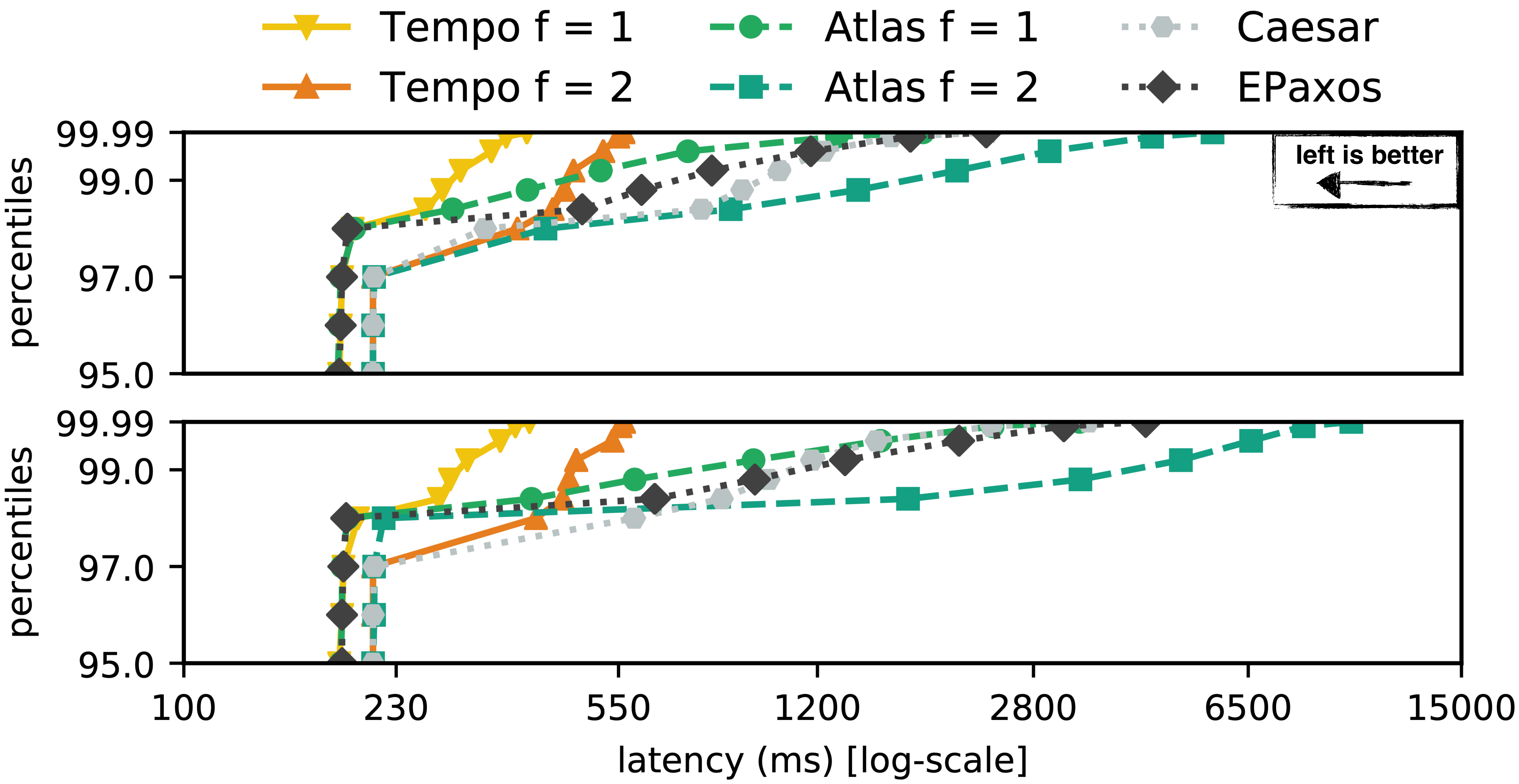
r = 5
clients per replica: 32 -> 20k

ops/s	2%	10%
fpaxos f =1	53K	53K
atlas f=1	129K	83K
tempo f=1	229K	230K



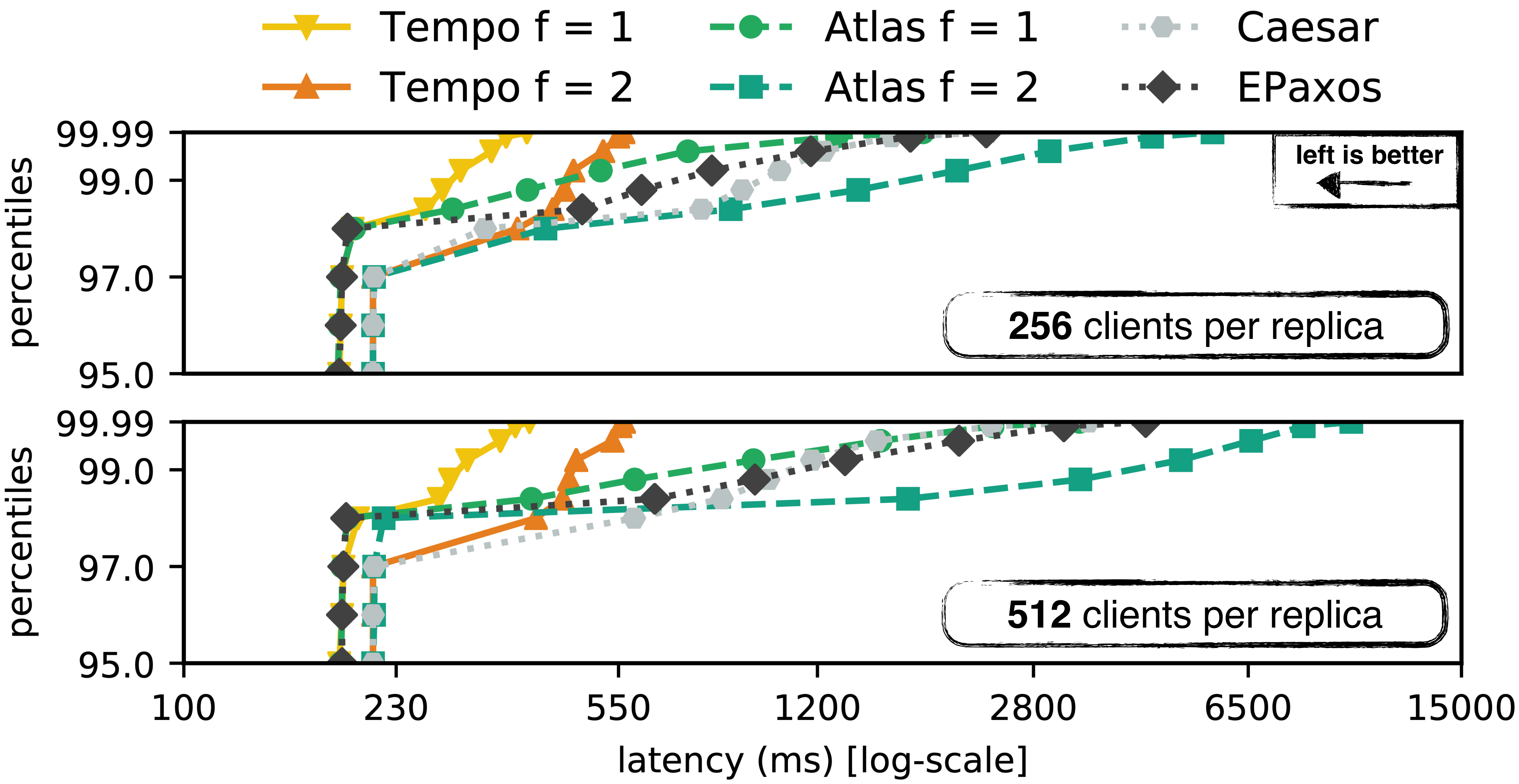
tail latency

r = 5
2% conflicts



tail latency

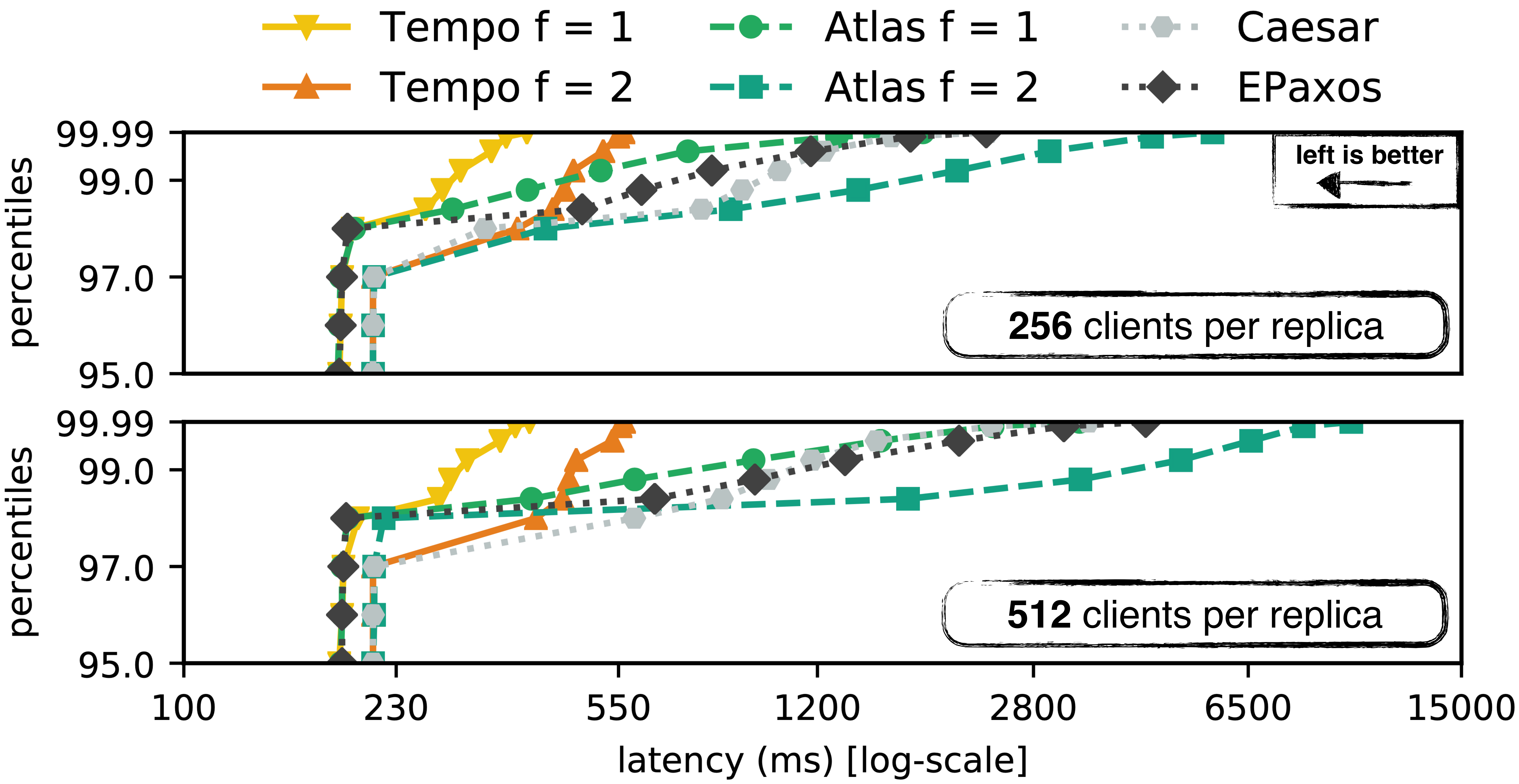
r = 5
2% conflicts



tail latency

r = 5
2% conflicts

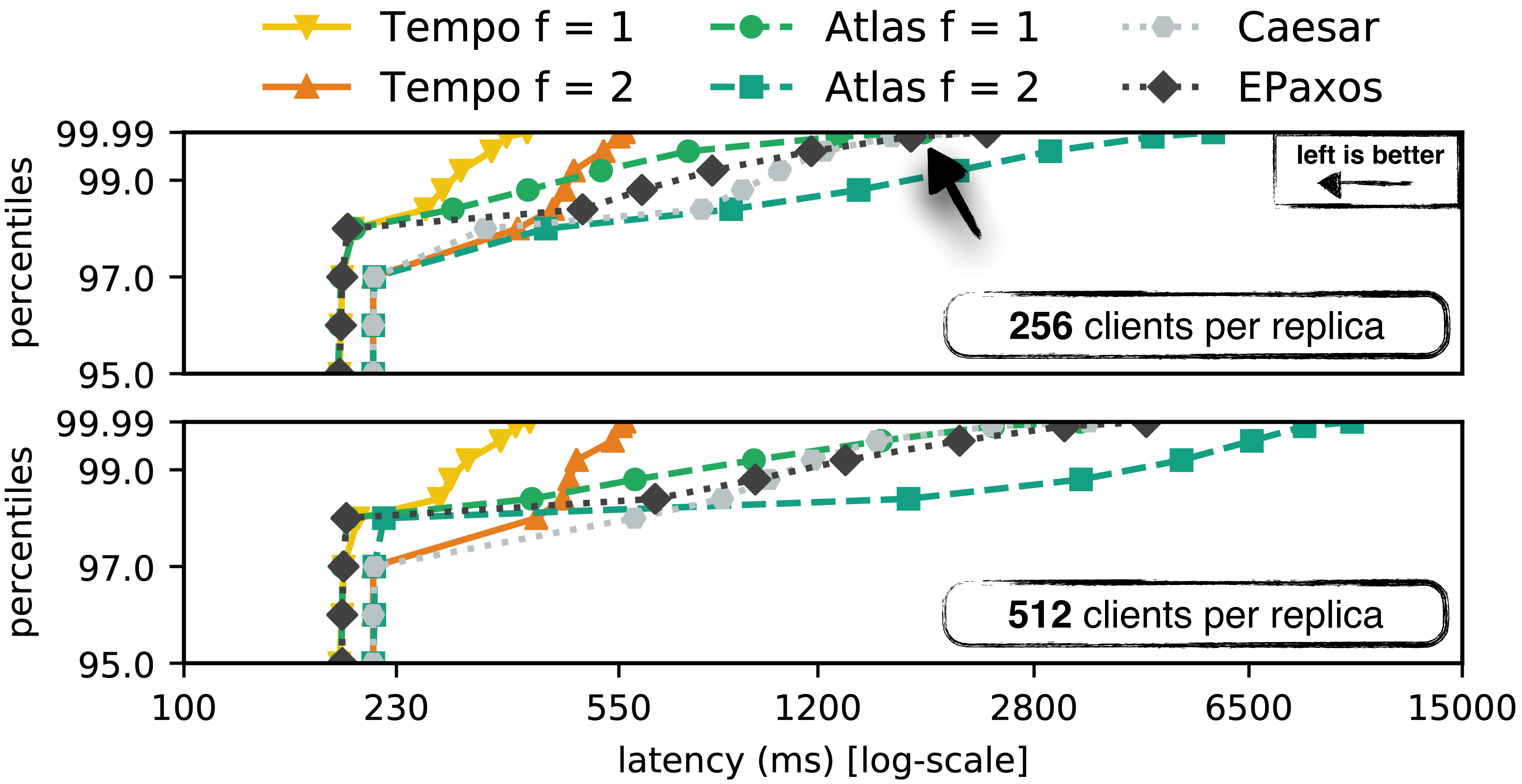
	99.9th	
	256	512
atlas f=1	1.3s	2.4s
epaxos	1.7s	3.1s



tail latency

r = 5
2% conflicts

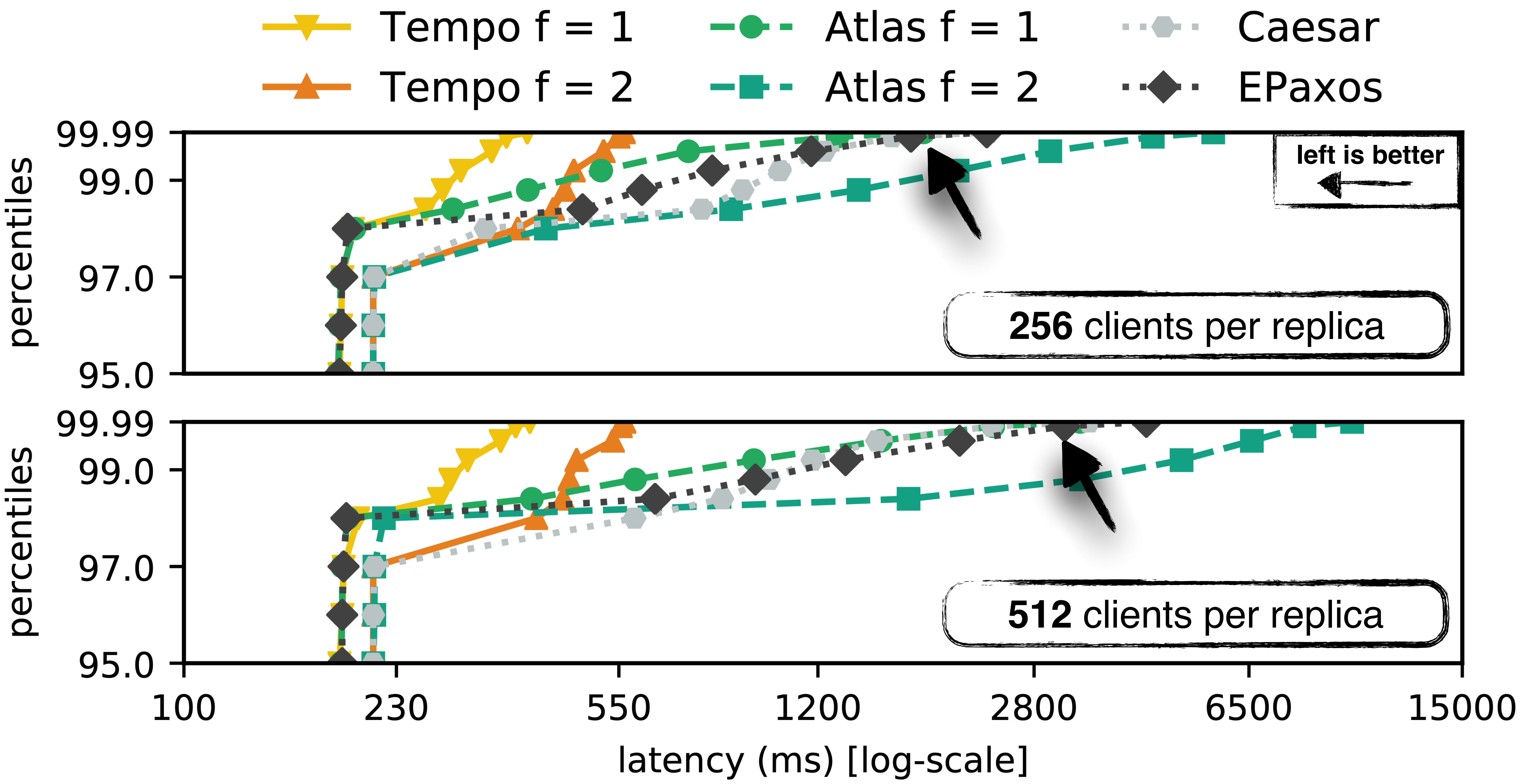
	99.9th	
	256	512
atlas f=1	1.3s	2.4s
epaxos	1.7s	3.1s



tail latency

r = 5
2% conflicts

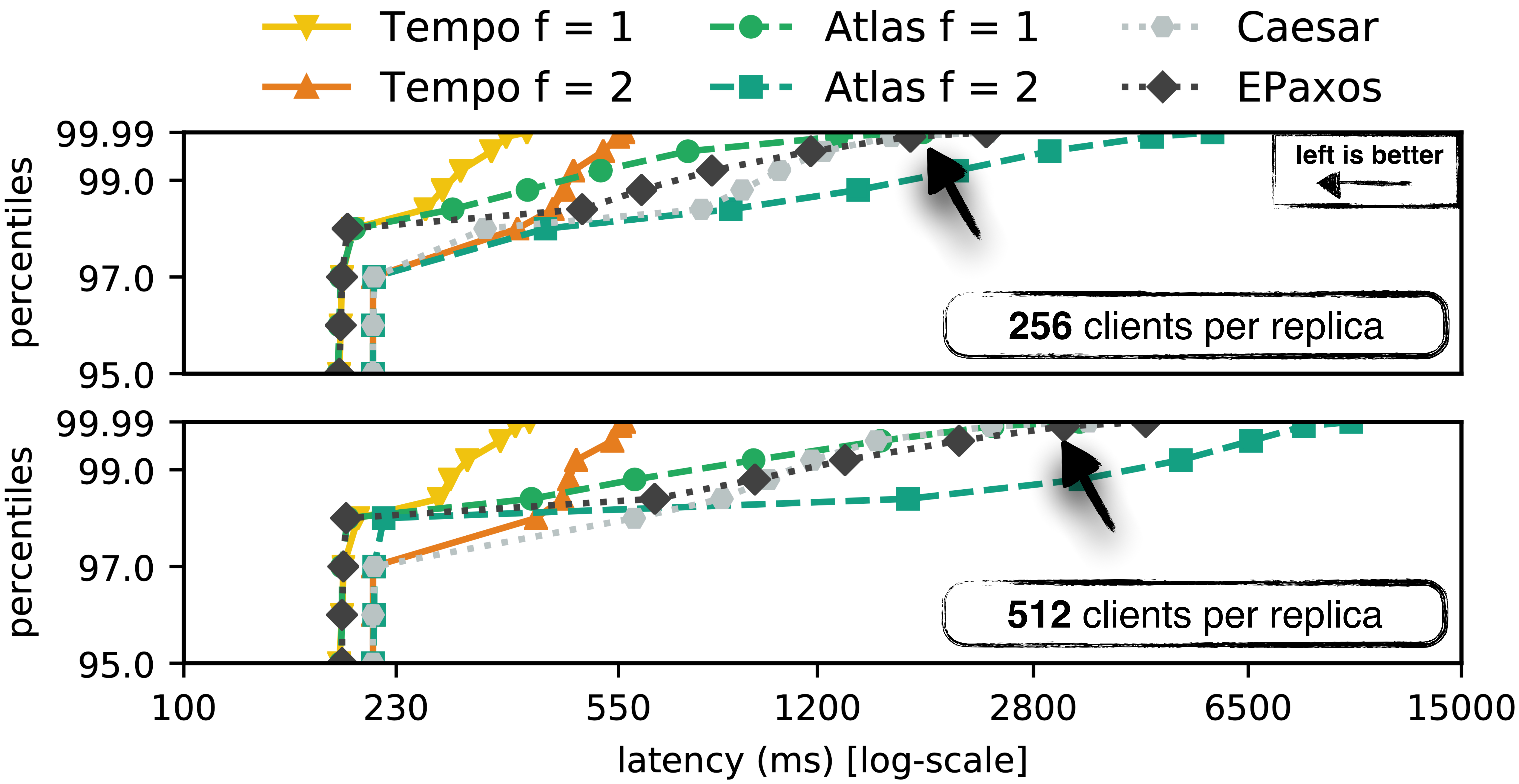
	99.9th	
	256	512
atlas f=1	1.3s	2.4s
epaxos	1.7s	3.1s



tail latency

r = 5
2% conflicts

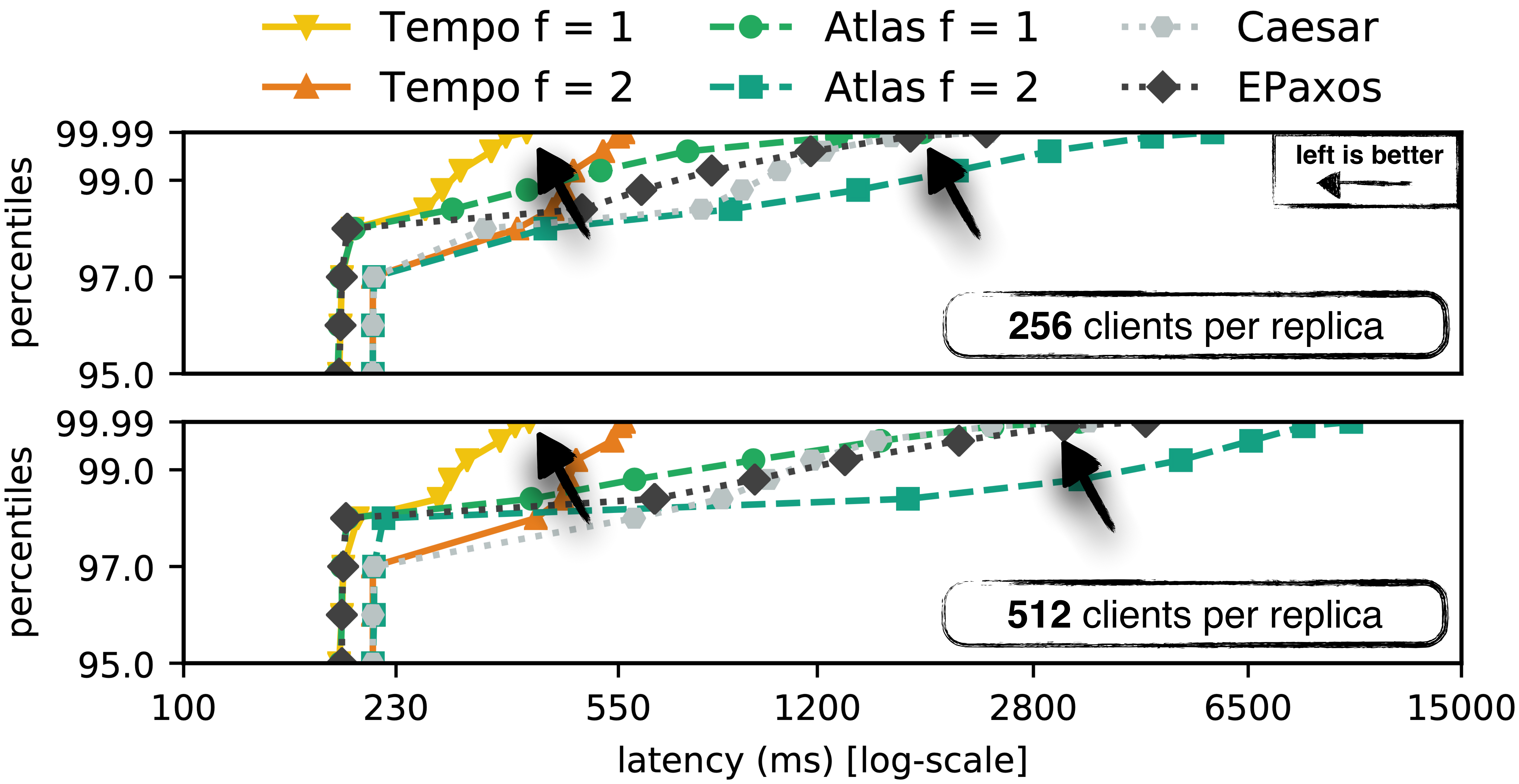
	99.9th	
	256	512
atlas f=1	1.3s	2.4s
epaxos	1.7s	3.1s
tempo f=1	354ms	367ms



tail latency

r = 5
2% conflicts

	99.9th	
	256	512
atlas f=1	1.3s	2.4s
epaxos	1.7s	3.1s
tempo f=1	354ms	367ms



summary

- atlas & tempo are the first leaderless protocols parameterized by **f**
    - **small fast quorums:** the protocols trade off higher fault tolerance for lower latency

summary

- atlas & tempo are the first leaderless protocols parameterized by **f**
    - **small fast quorums:** the protocols trade off higher fault tolerance for lower latency

- atlas & tempo fix the fast quorum, simplifying recovery

summary

- atlas & tempo are the first leaderless protocols parameterized by **f**
  - **small fast quorums:** the protocols trade off higher fault tolerance for lower latency
- atlas & tempo fix the fast quorum, simplifying recovery
- tempo provides **predictable performance** even in contended workloads
- tempo handles both **full** and **partial replication** scenarios

summary

- atlas & tempo are the first leaderless protocols parameterized by **f**
    - **small fast quorums:** the protocols trade off higher fault tolerance for lower latency
- atlas & tempo fix the fast quorum, simplifying recovery
- tempo provides **predictable performance** even in contended workloads
- tempo handles both **full** and **partial replication** scenarios

leaderless protocols are becoming practical!!
cassandra will release **accord, a new timestamp-based leaderless protocol** (like **tempo**)

publications

atlas

State-Machine Replication for Planet-Scale Systems @ **EuroSys'20**

Vitor Enes, Carlos Baquero, Tuanir França Rezende, Alexey Gotsman, Matthieu Perrin, Pierre Sutra.

tempo

Efficient Replication via Timestamp Stability @ **EuroSys'21**

Vitor Enes, Carlos Baquero, Alexey Gotsman, Pierre Sutra.

publications
& acknowledgments

atlas

State-Machine Replication for Planet-Scale Systems @ **EuroSys'20**

Vitor Enes, Carlos Baquero, Tuanir França Rezende, Alexey Gotsman, Matthieu Perrin, Pierre Sutra.

tempo

Efficient Replication via Timestamp Stability @ **EuroSys'21**

Vitor Enes, Carlos Baquero, Alexey Gotsman, Pierre Sutra.

HASLab
HIGH-ASSURANCE
SOFTWARE LABORATORY

publications
& acknowledgments

atlas

State-Machine Replication for Planet-Scale Systems @ **EuroSys'20**

Vitor Enes, Carlos Baquero, Tuanir França Rezende, Alexey Gotsman, Matthieu Perrin, Pierre Sutra.

tempo

Efficient Replication via Timestamp Stability @ **EuroSys'21**

Vitor Enes, Carlos Baquero, Alexey Gotsman, Pierre Sutra.