Auditing Outsourced Services

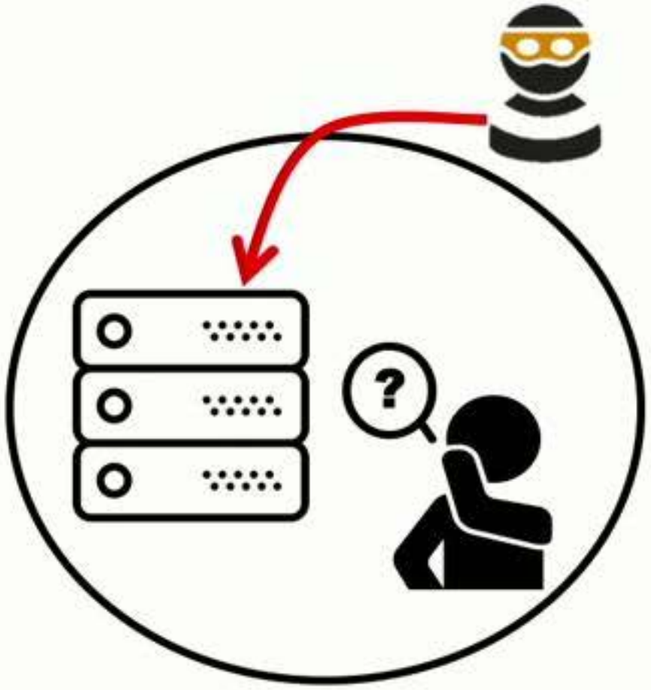
Cheng Tan

Courant Institute, New York University

chengtan@cs.nyu.edu

Does my remote service behave as promised?

Are there unexpected internal failures?



Are there external attacks?

Remote servers (clouds) make mistakes

- bugs:
- misconfigs:

...

threat post

Dangerous Kubernetes Bugs Allow Authentication Bypass, DoS

SC MEDIA

Security misconfigurations: Code for human error

Remote servers (clouds) make mistakes

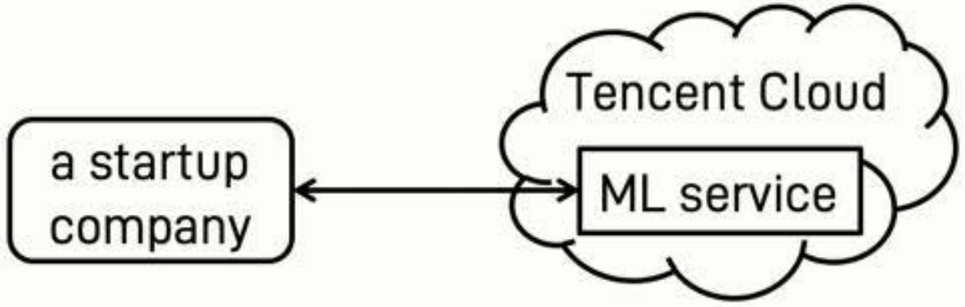
- bugs:
- misconfigs:
- untrusted cloud:

threatpost **Dangerous Kubernetes Bugs Allow Authentication Bypass, DoS**

SC MEDIA **Security misconfigurations: Code for human error**

深圳 政府在线 SHENZHEN CHINA

Tencent Cloud denies technical attack against rival



Remote servers (clouds) make mistakes

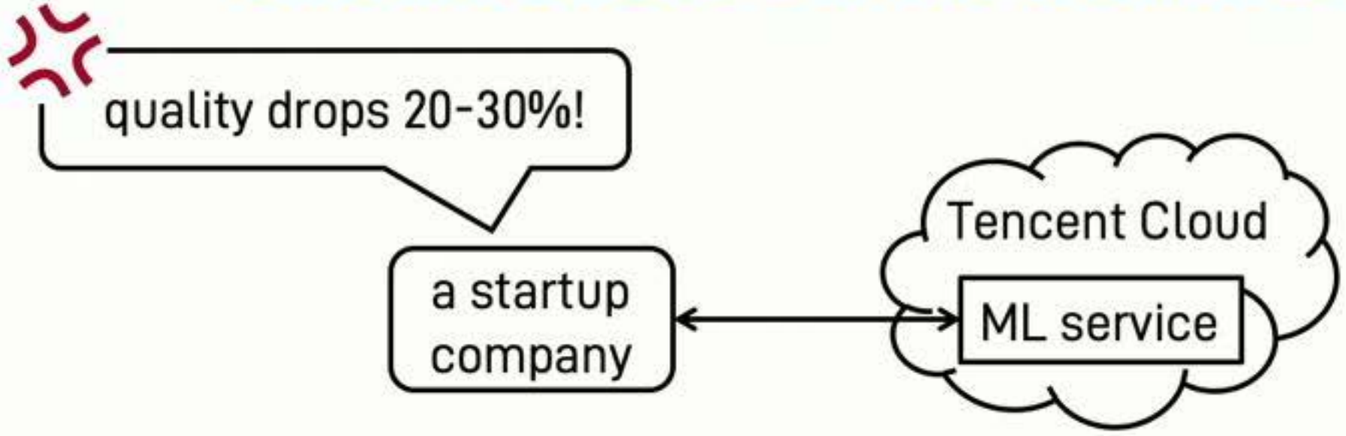
- bugs:
- misconfigs:
- untrusted cloud:

threatpost **Dangerous Kubernetes Bugs Allow Authentication Bypass, DoS**

SC MEDIA **Security misconfigurations: Code for human error**

深圳 政府在线 SHENZHEN CHINA

Tencent Cloud denies technical attack against rival



Remote servers (clouds) make mistakes

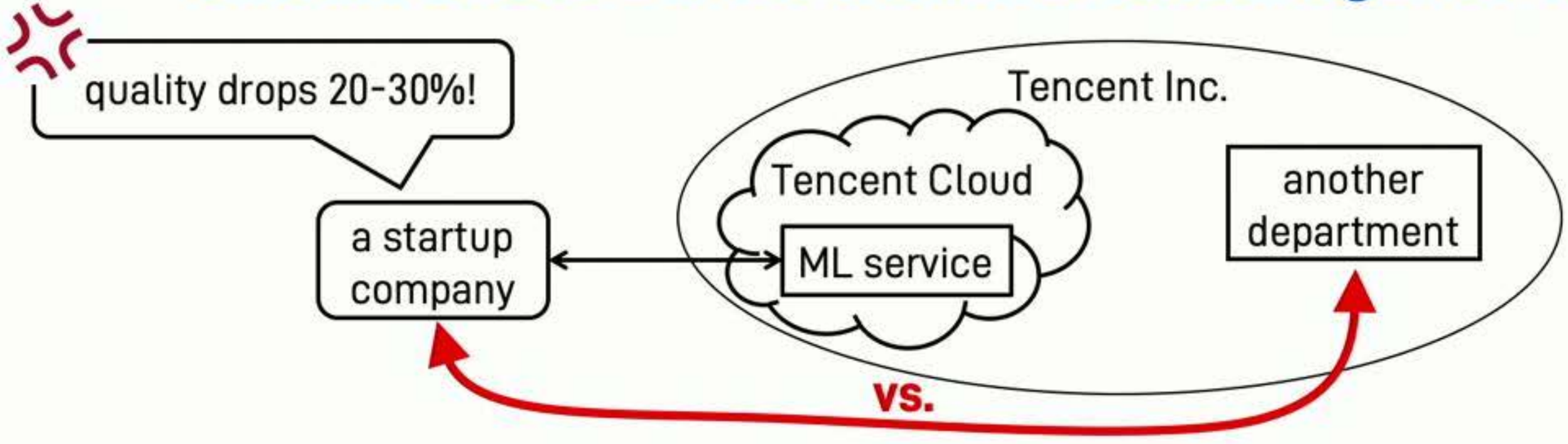
- bugs:
- misconfigs:
- untrusted cloud:

threatpost **Dangerous Kubernetes Bugs Allow Authentication Bypass, DoS**

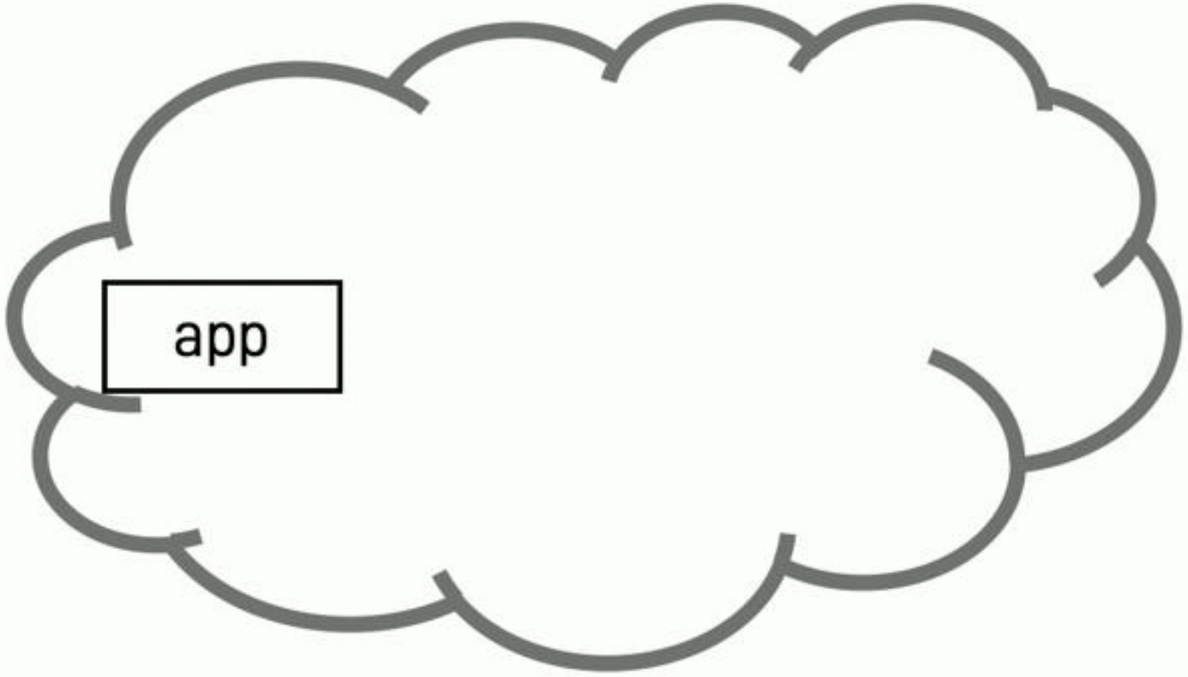
SC MEDIA **Security misconfigurations: Code for human error**

深圳 政府在线 SHENZHEN CHINA

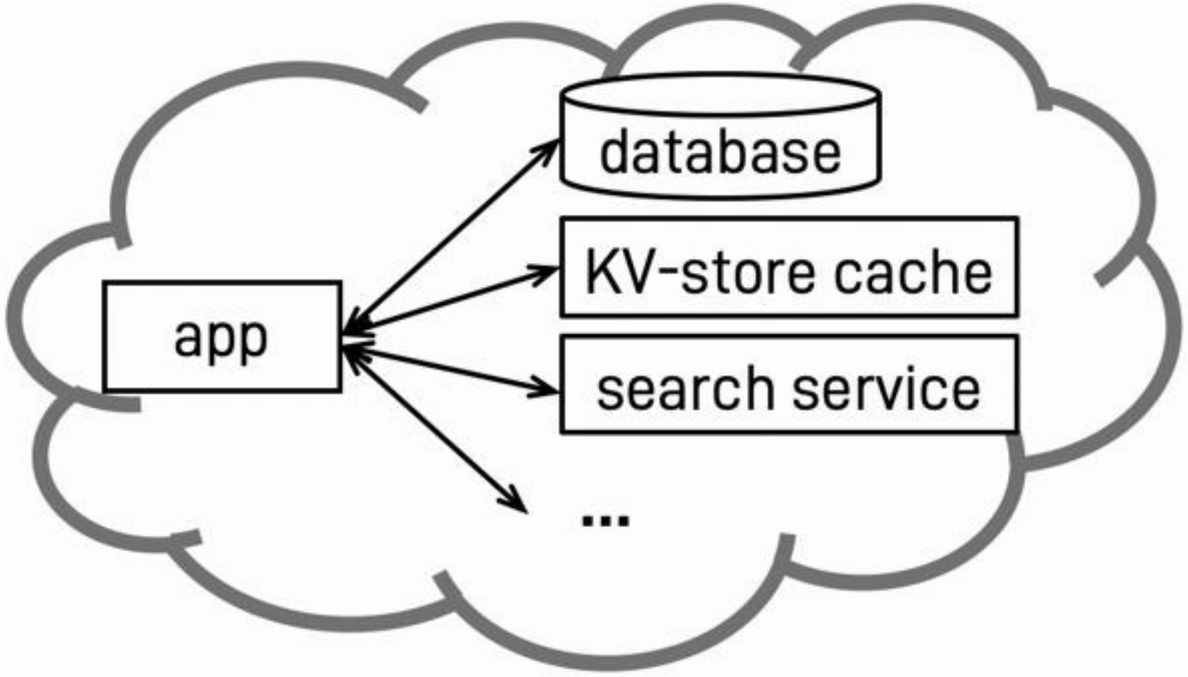
Tencent Cloud denies technical attack against rival



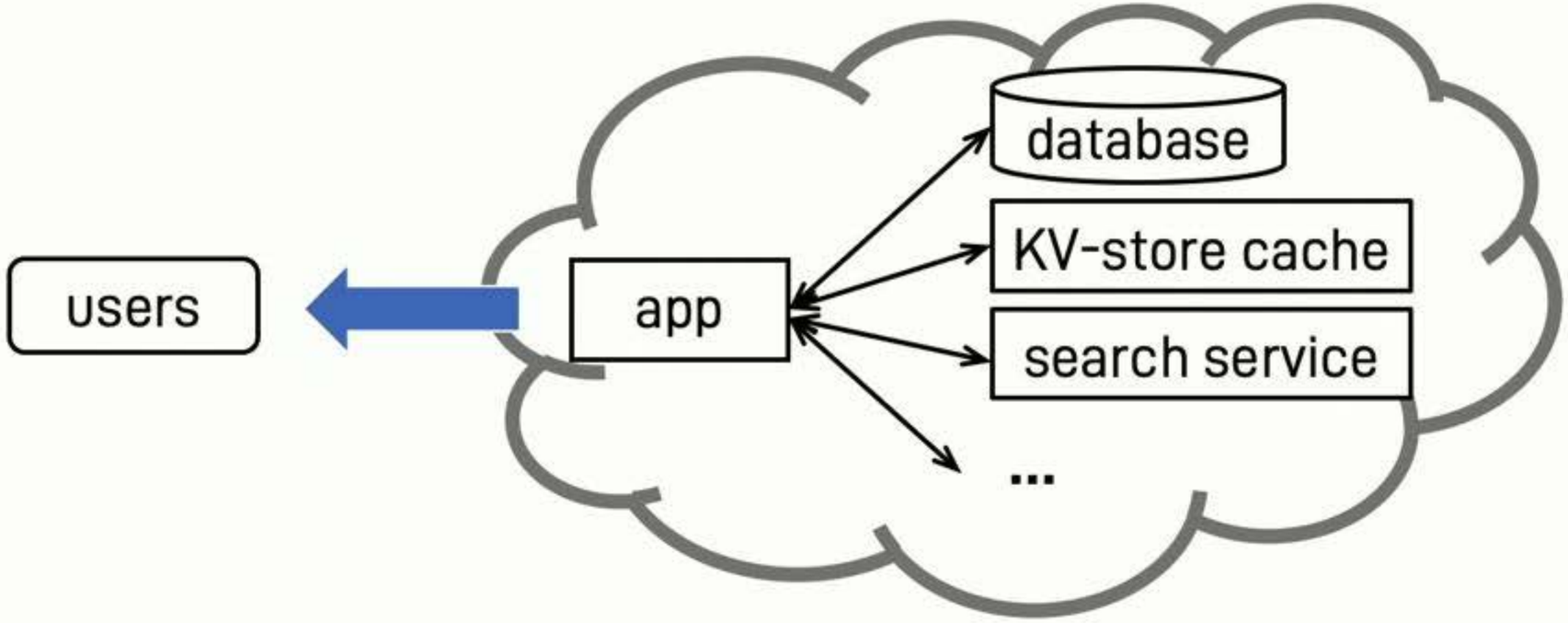
Verify outsourced services with systems



Verify outsourced services with systems



Verify outsourced services with systems

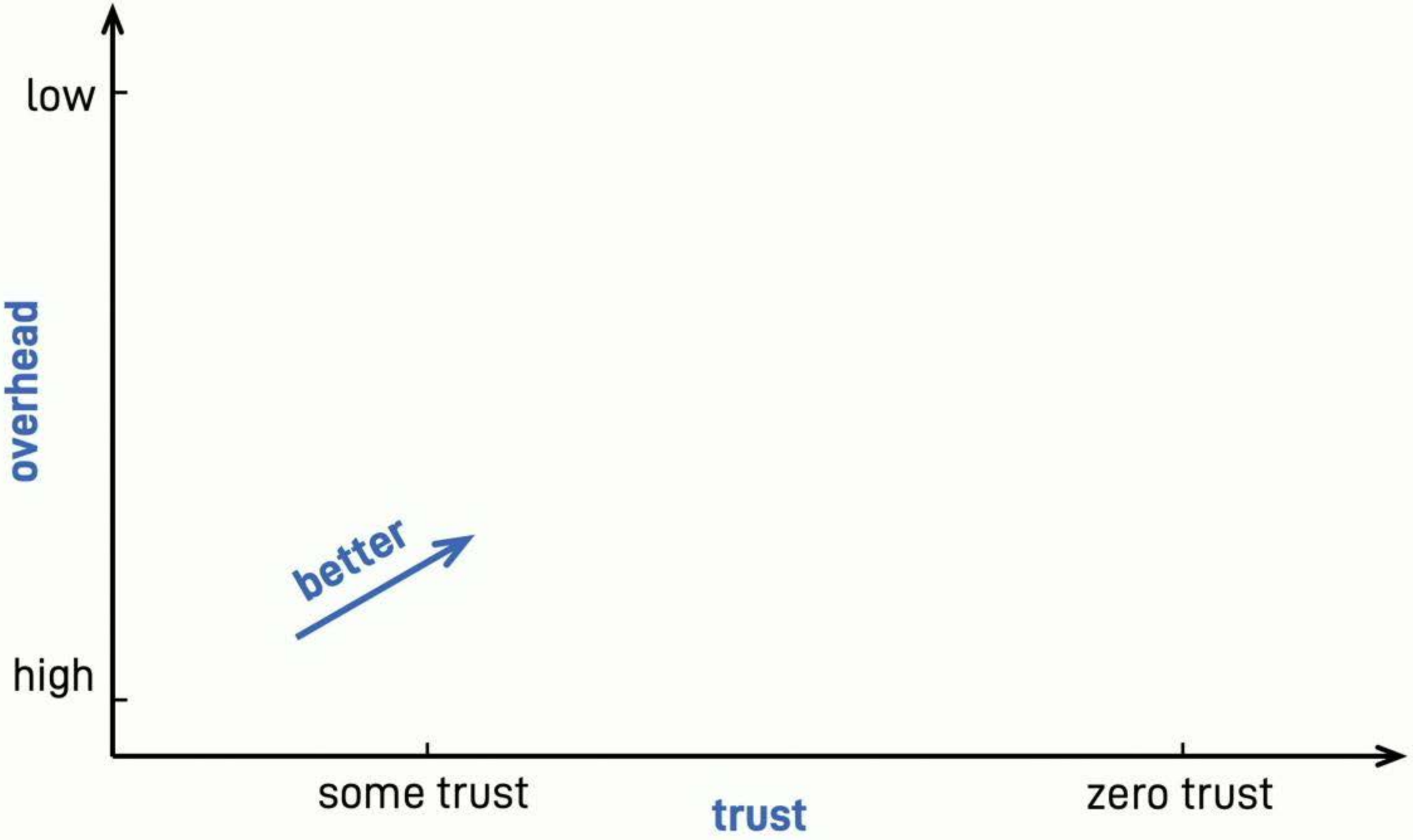


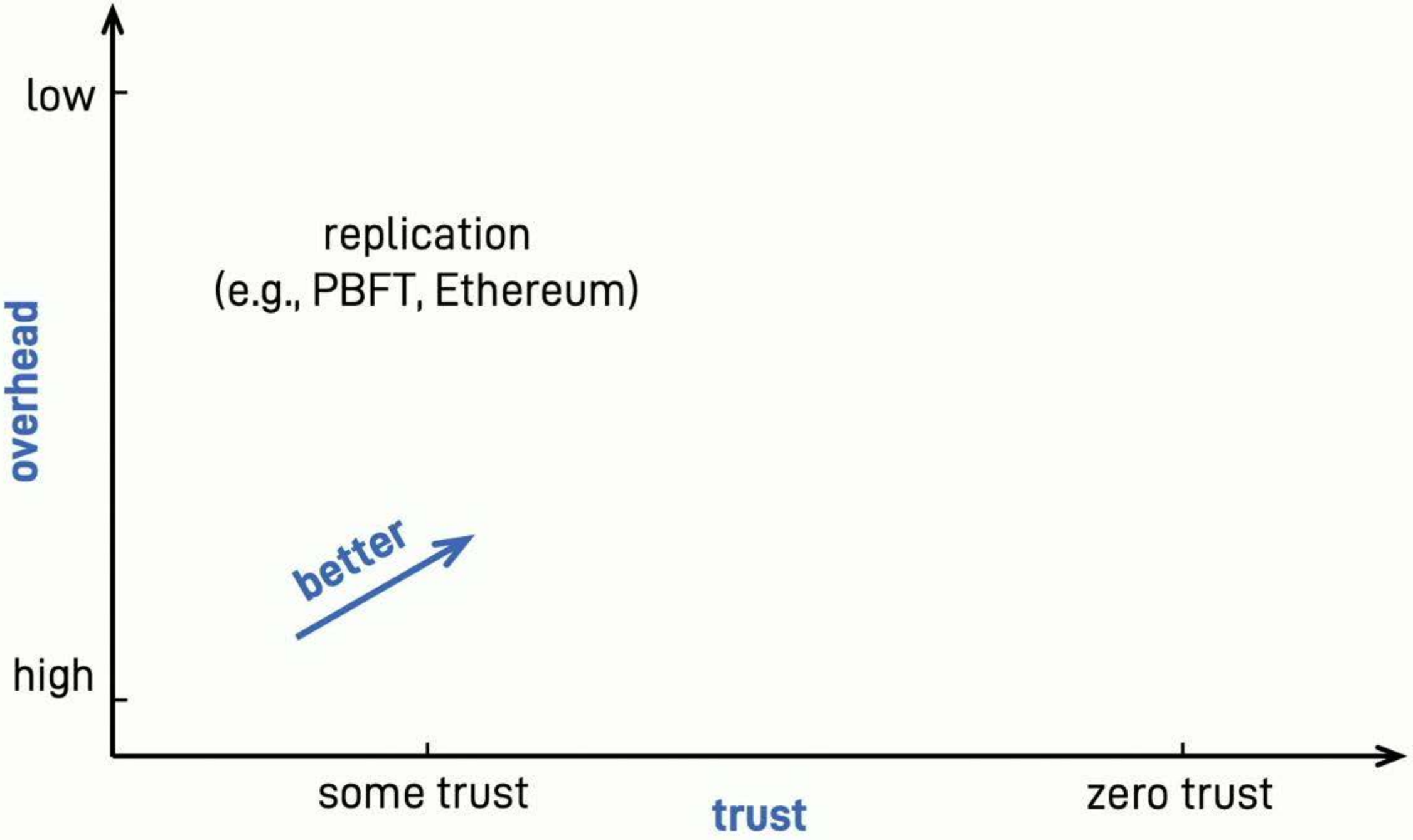
- Honest servers can convince users that app is faithfully executed.
- Buggy or malicious servers cannot pass users' checks.

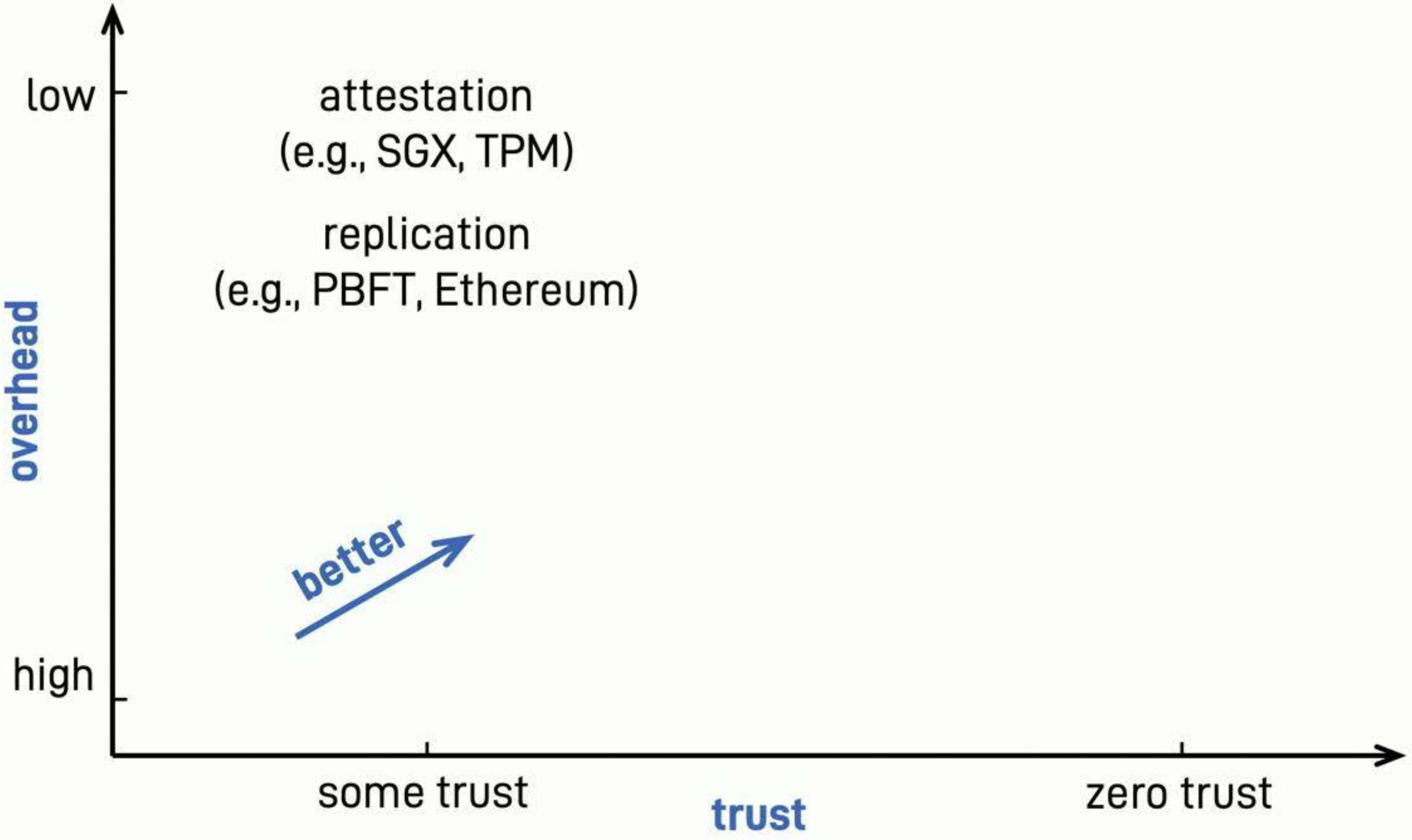
Execution integrity...

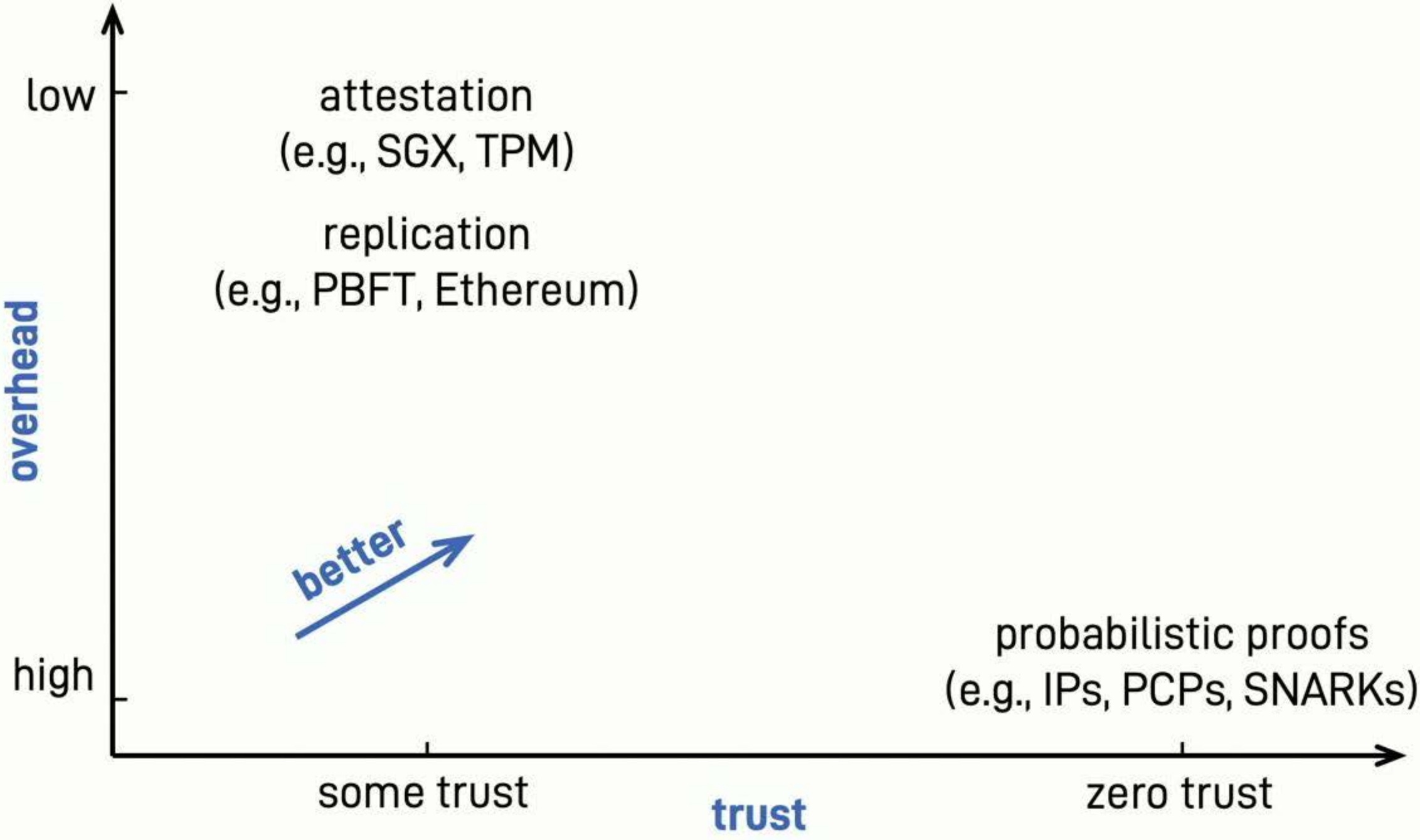
...is about executing code faithfully.

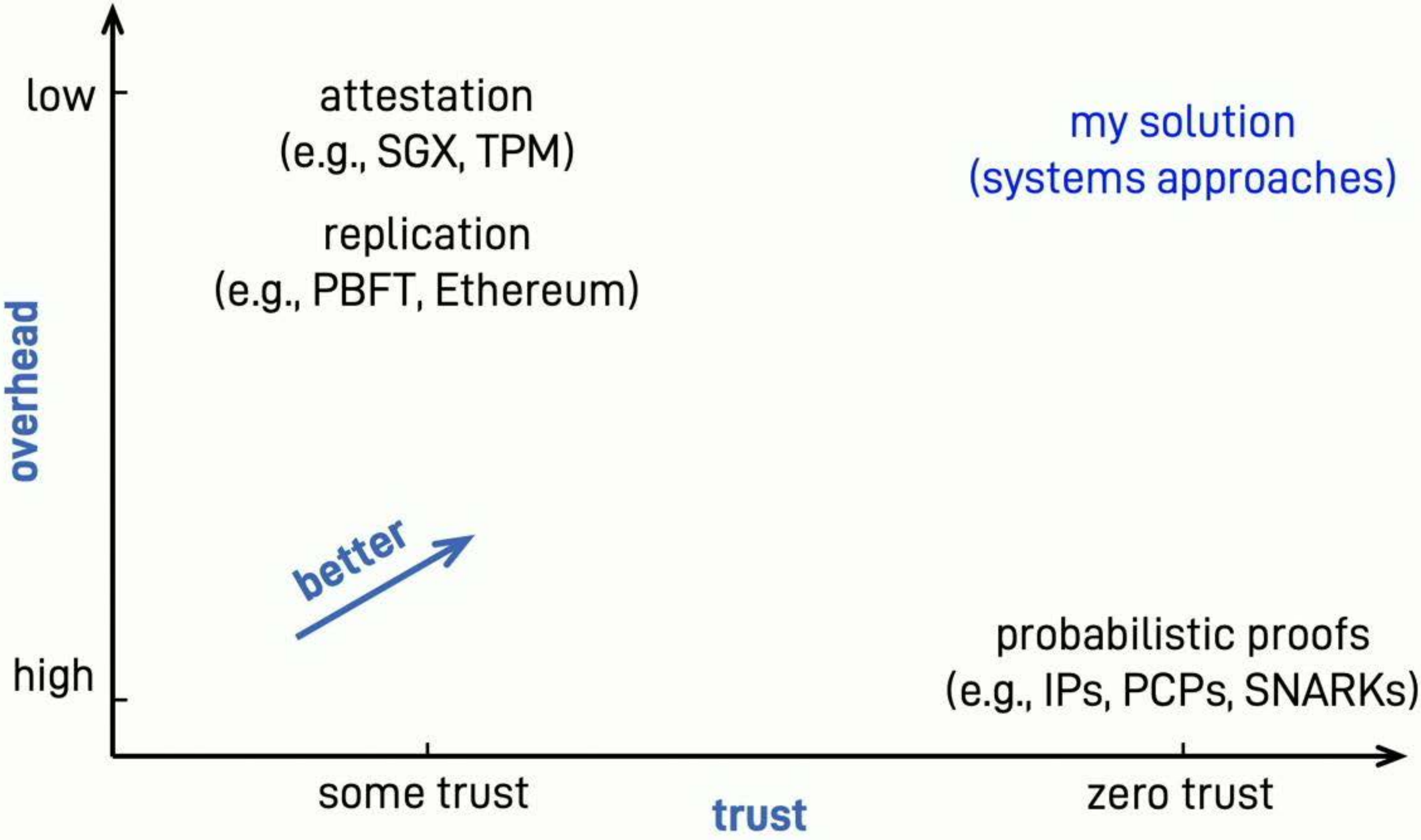
...is separate from program verification.

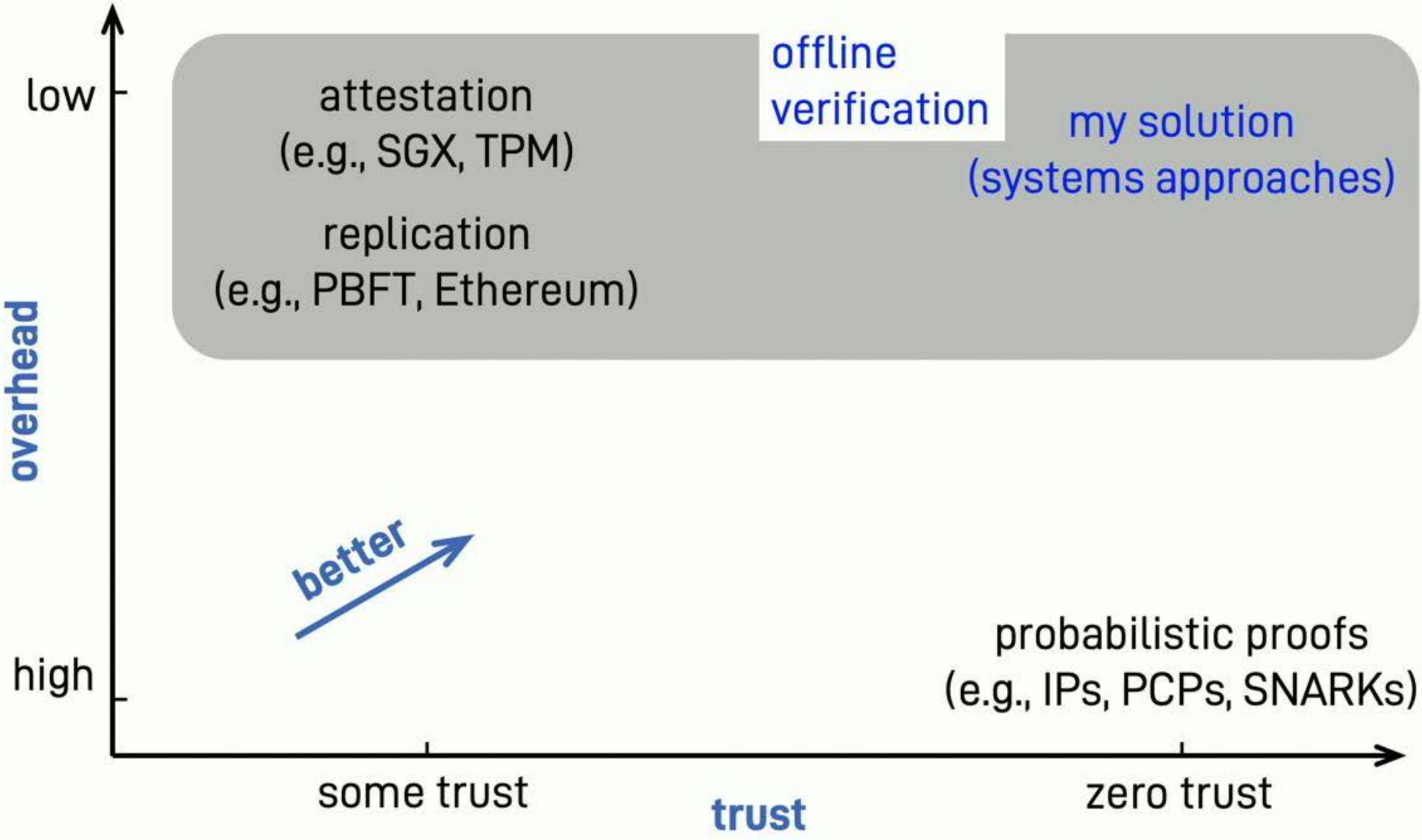

low
high
overhead
better
some trust
zero trust
trust

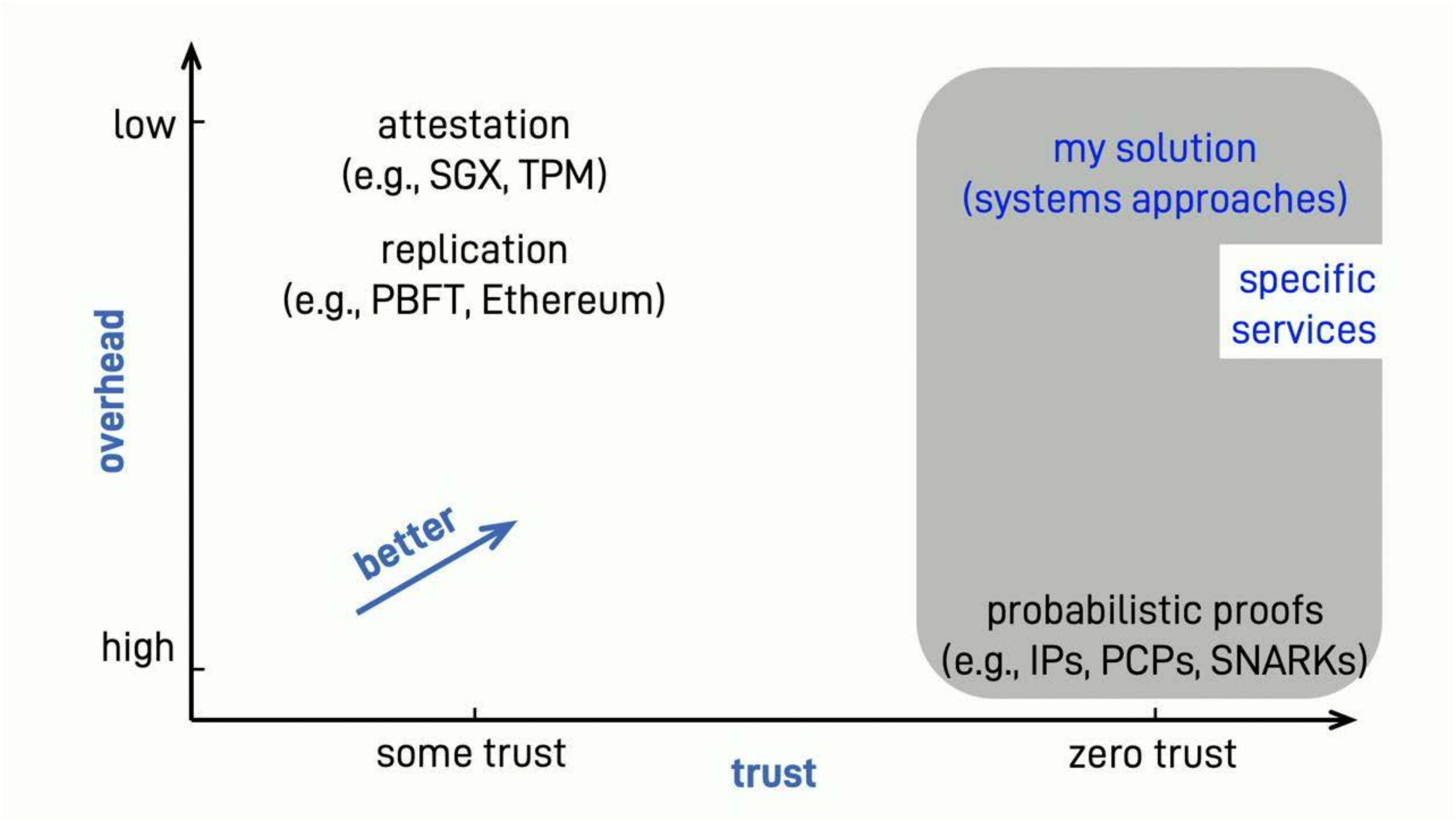

low
high
overhead
replication
(e.g., PBFT, Ethereum)
better
some trust
zero trust
trust

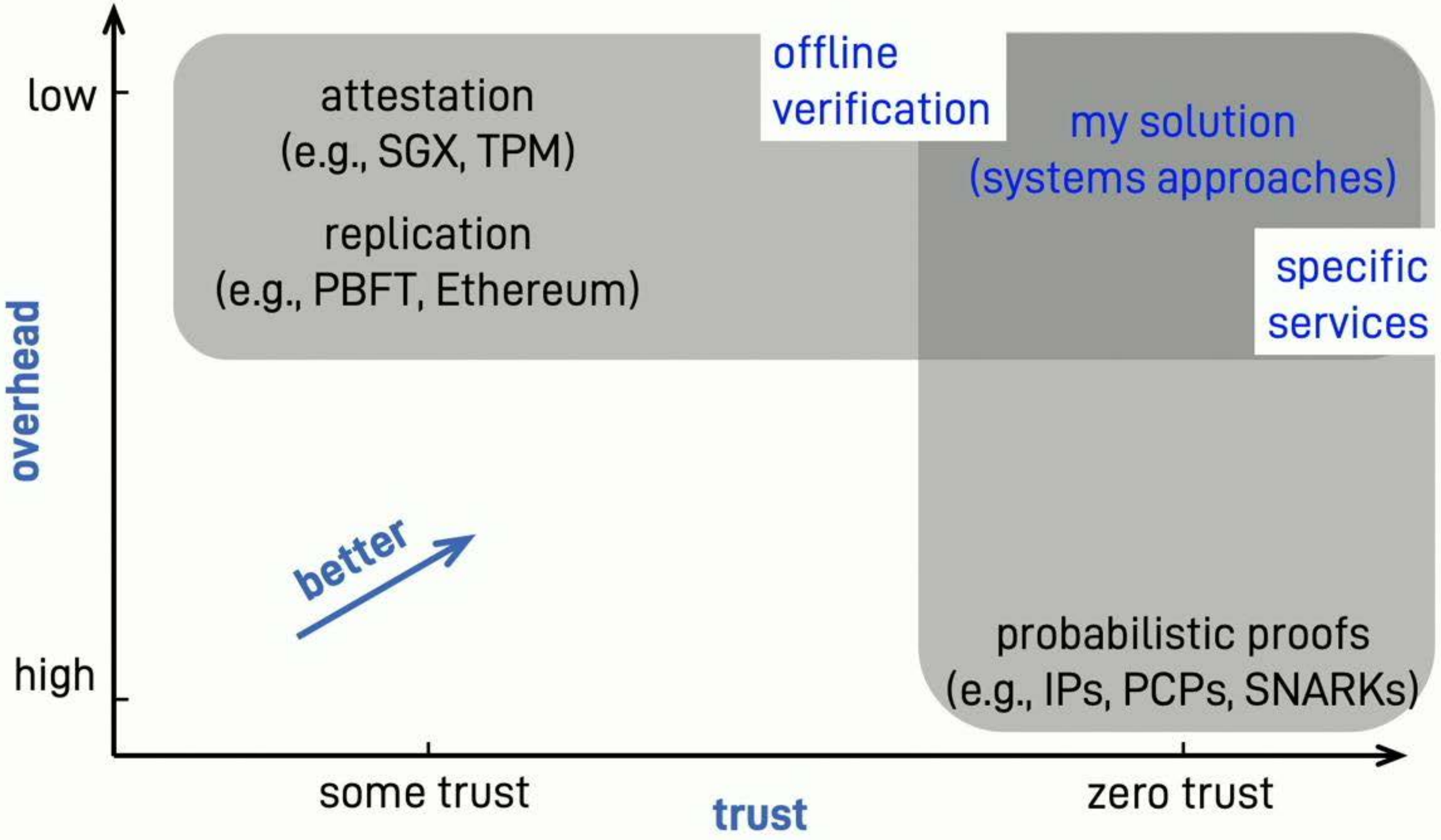

low
attestation
(e.g., SGX, TPM)
replication
(e.g., PBFT, Ethereum)
overhead
better
high
some trust
trust
zero trust


low
high
overhead
attestation
(e.g., SGX, TPM)
replication
(e.g., PBFT, Ethereum)
better
probabilistic proofs
(e.g., IPs, PCPs, SNARKs)
some trust
zero trust
trust

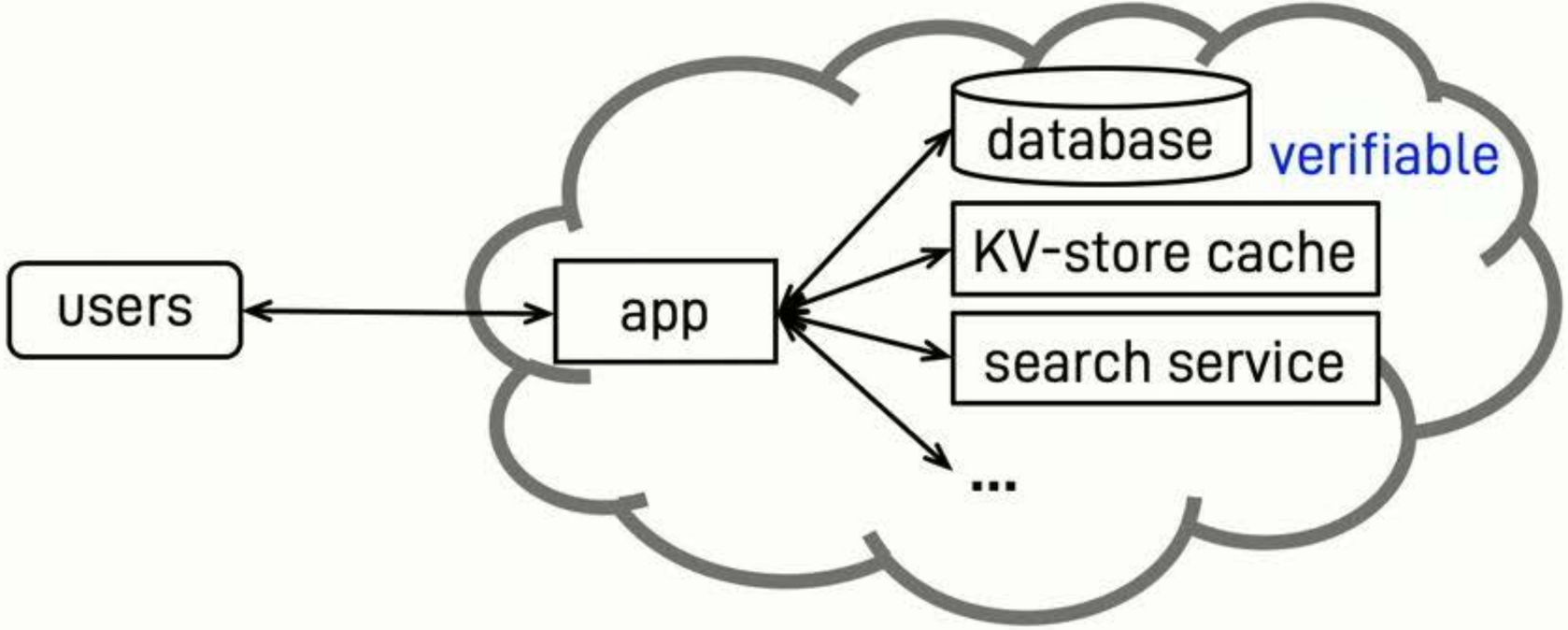

low
high
overhead
attestation
(e.g., SGX, TPM)
replication
(e.g., PBFT, Ethereum)
my solution
(systems approaches)
better
probabilistic proofs
(e.g., IPs, PCPs, SNARKs)
some trust
zero trust
trust


low
high
overhead
attestation
(e.g., SGX, TPM)
replication
(e.g., PBFT, Ethereum)
offline
verification
my solution
(systems approaches)
better
probabilistic proofs
(e.g., IPs, PCPs, SNARKs)
some trust
zero trust
trust

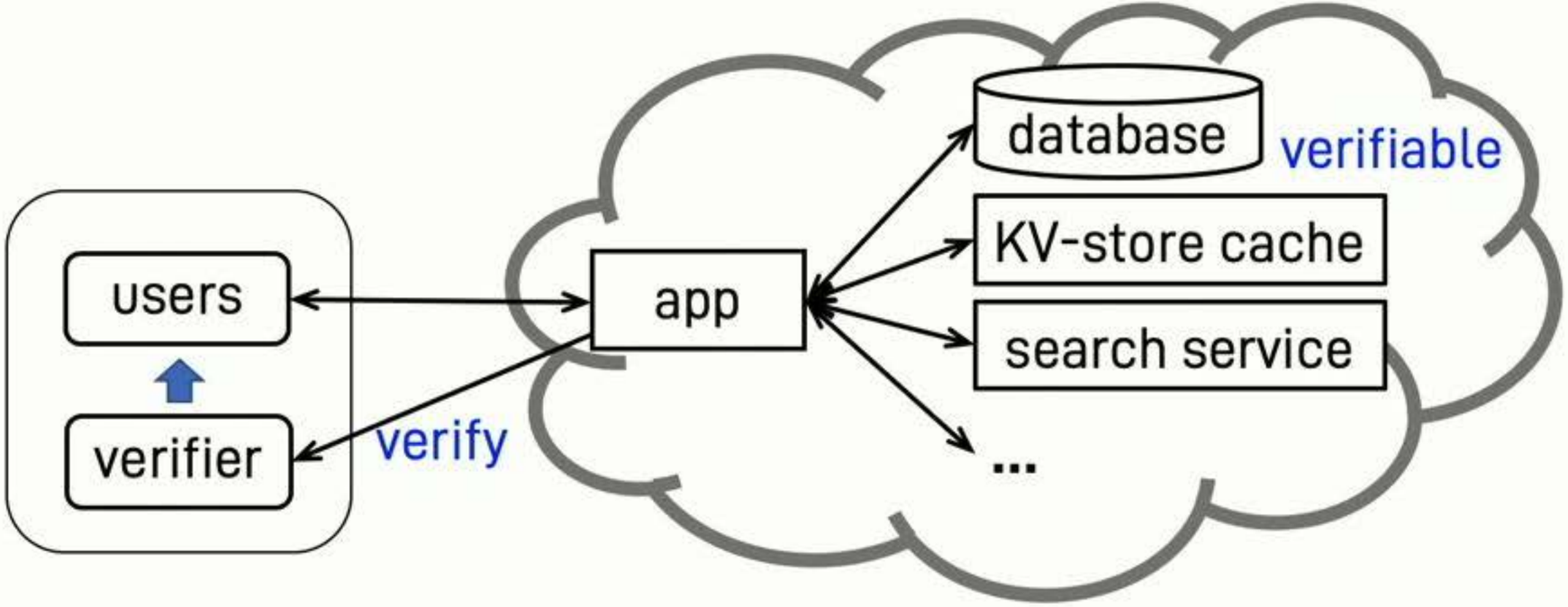

low
high
overhead
attestation
(e.g., SGX, TPM)
replication
(e.g., PBFT, Ethereum)
better
my solution
(systems approaches)
specific
services
probabilistic proofs
(e.g., IPs, PCPs, SNARKs)
some trust
zero trust
trust


low
high
overhead
attestation
(e.g., SGX, TPM)
replication
(e.g., PBFT, Ethereum)
offline
verification
my solution
(systems approaches)
specific
services
probabilistic proofs
(e.g., IPs, PCPs, SNARKs)
better
some trust
zero trust
trust

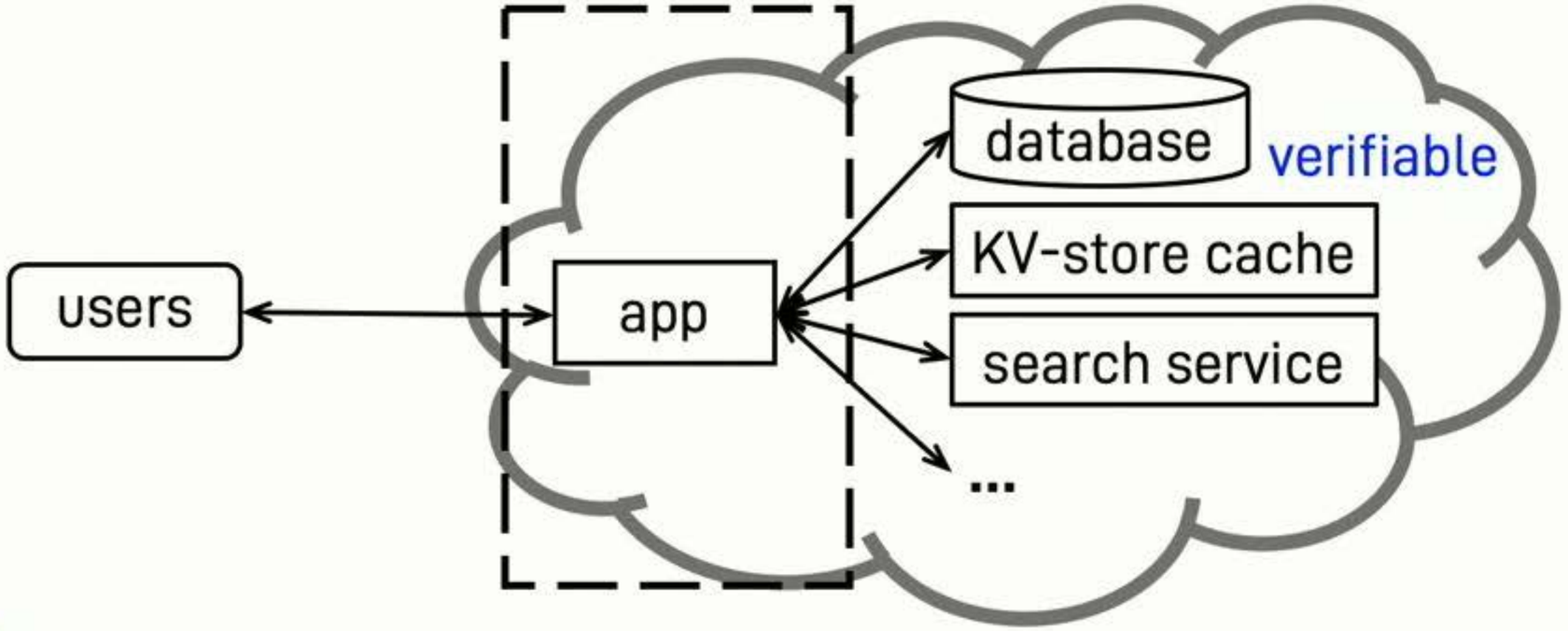
Rest of the talk: verifiable infrastructure



Rest of the talk: verifiable infrastructure



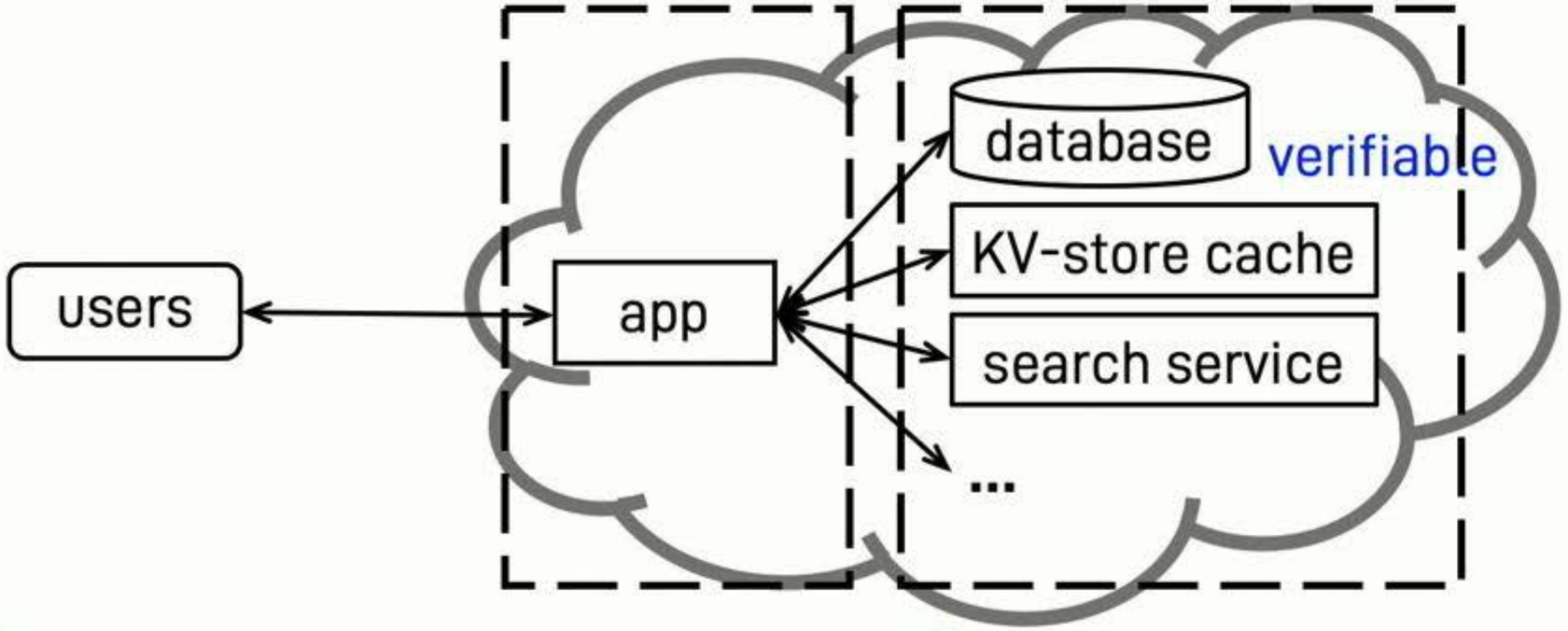
Rest of the talk: verifiable infrastructure



① verify white-box services

verifier

Rest of the talk: verifiable infrastructure

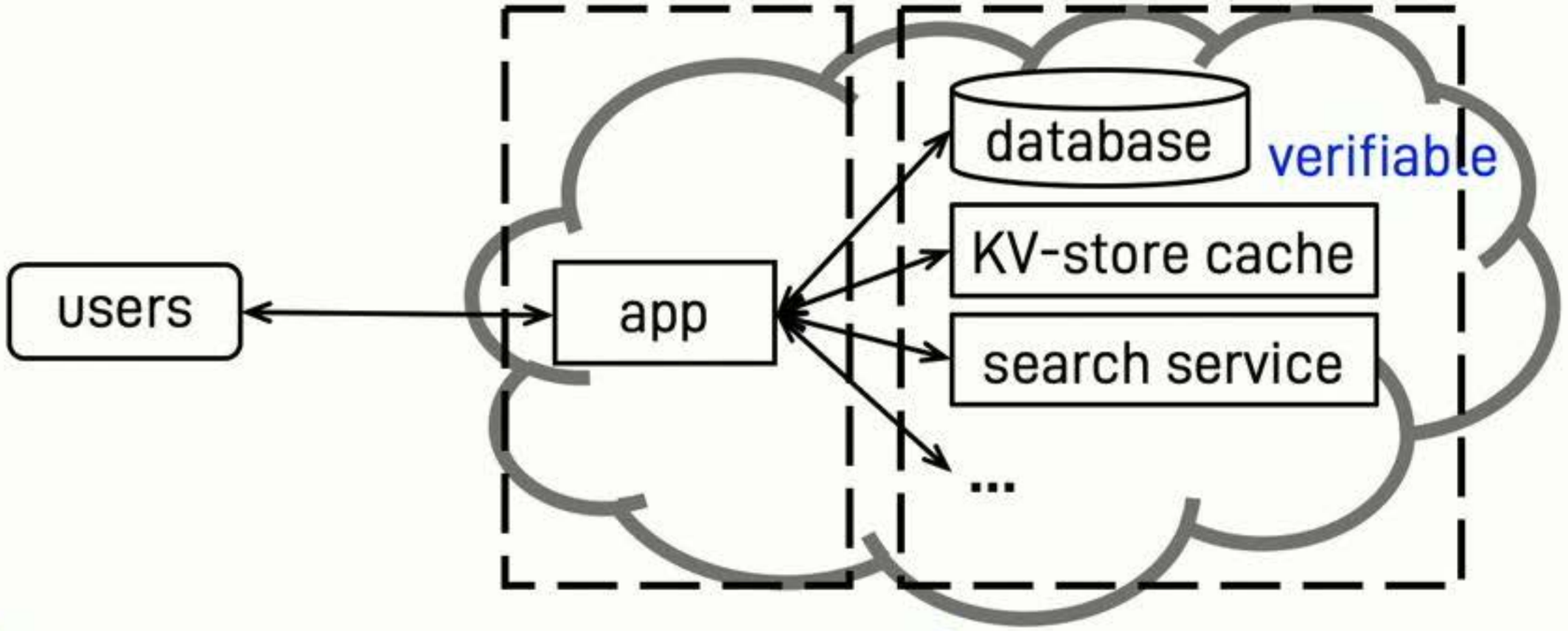


① verify white-box services

② verify black-box services

verifier

Rest of the talk: verifiable infrastructure



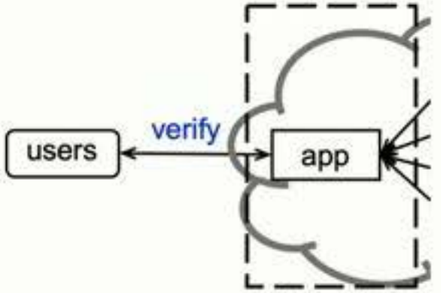
① verify white-box services

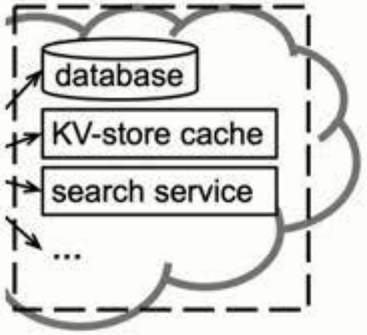
② verify black-box services

verifier

③ compose verifications of multiple services

Rest of the talk: verifiable infrastructure





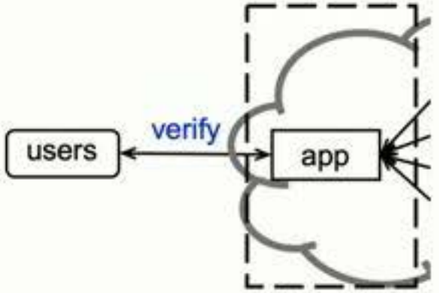
① Verify white-box services (Orochi [SOSP'17])

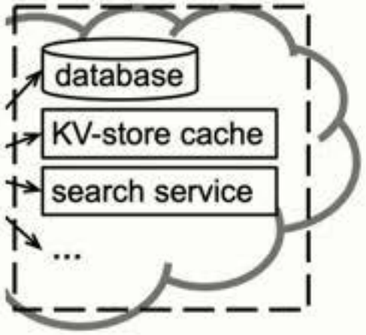
② Verify black-box services (Cobra)

③ Build composable verifiable framework (future work)

④ Other past work
- Troubleshooting data center networks [NSDI'19]
- Protecting secret via security-oriented offloading [EuroSys'15]

Rest of the talk: verifiable infrastructure



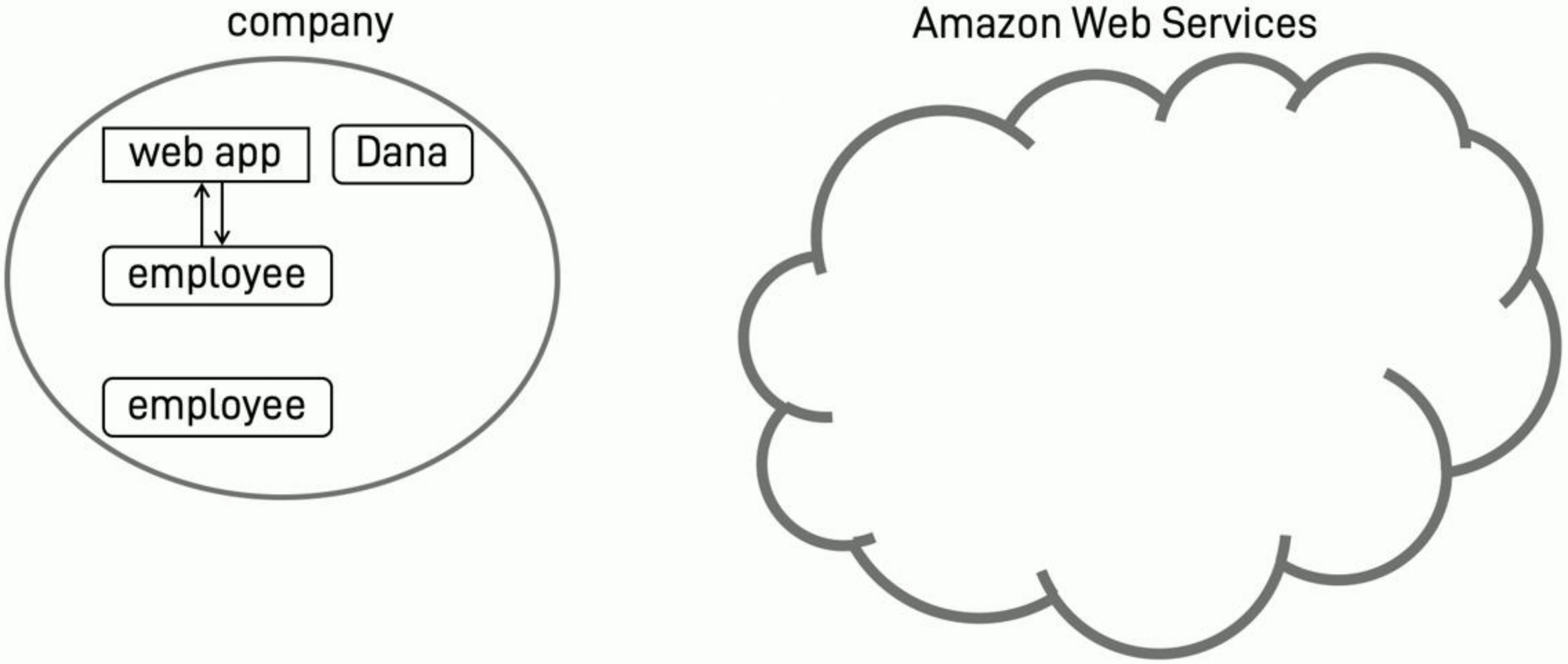


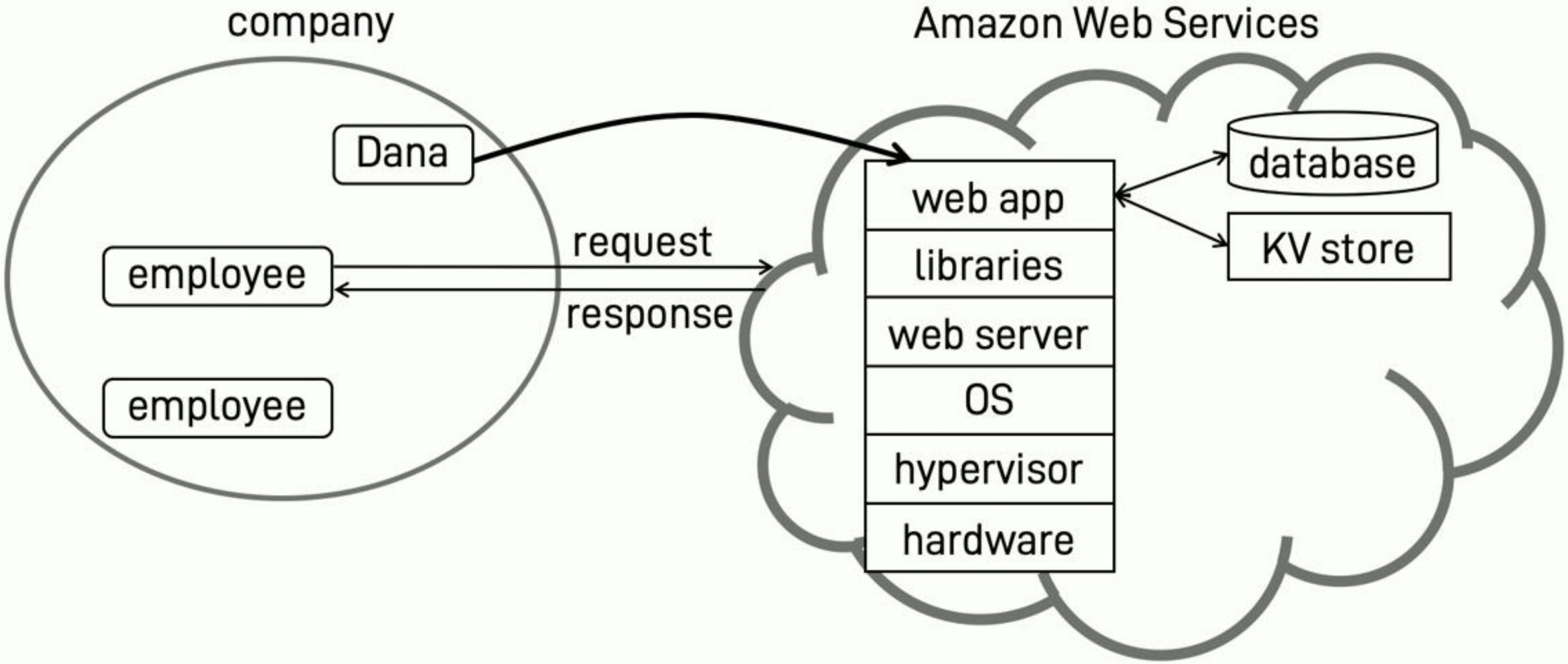
① Verify white-box services (Orochi [SOSP'17])

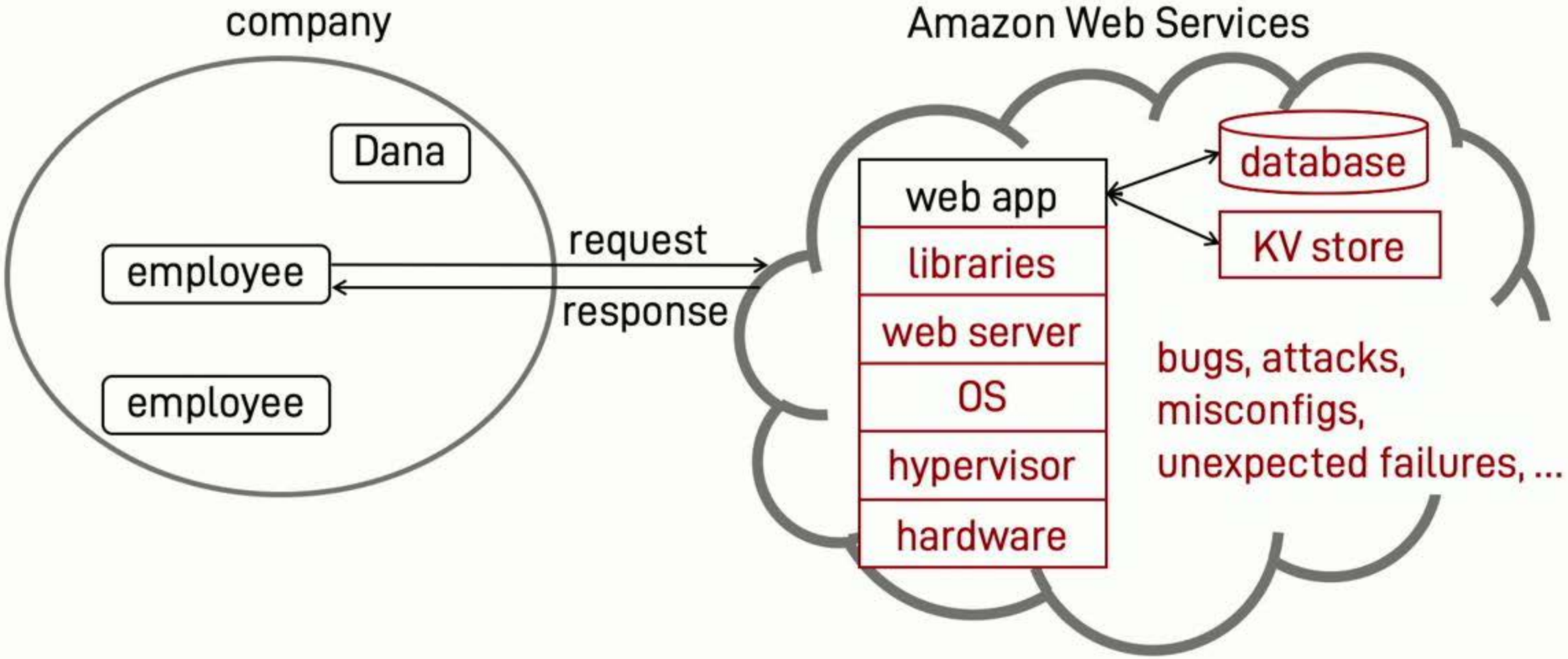
② Verify black-box services (Cobra)

③ Build composable verifiable framework (future work)

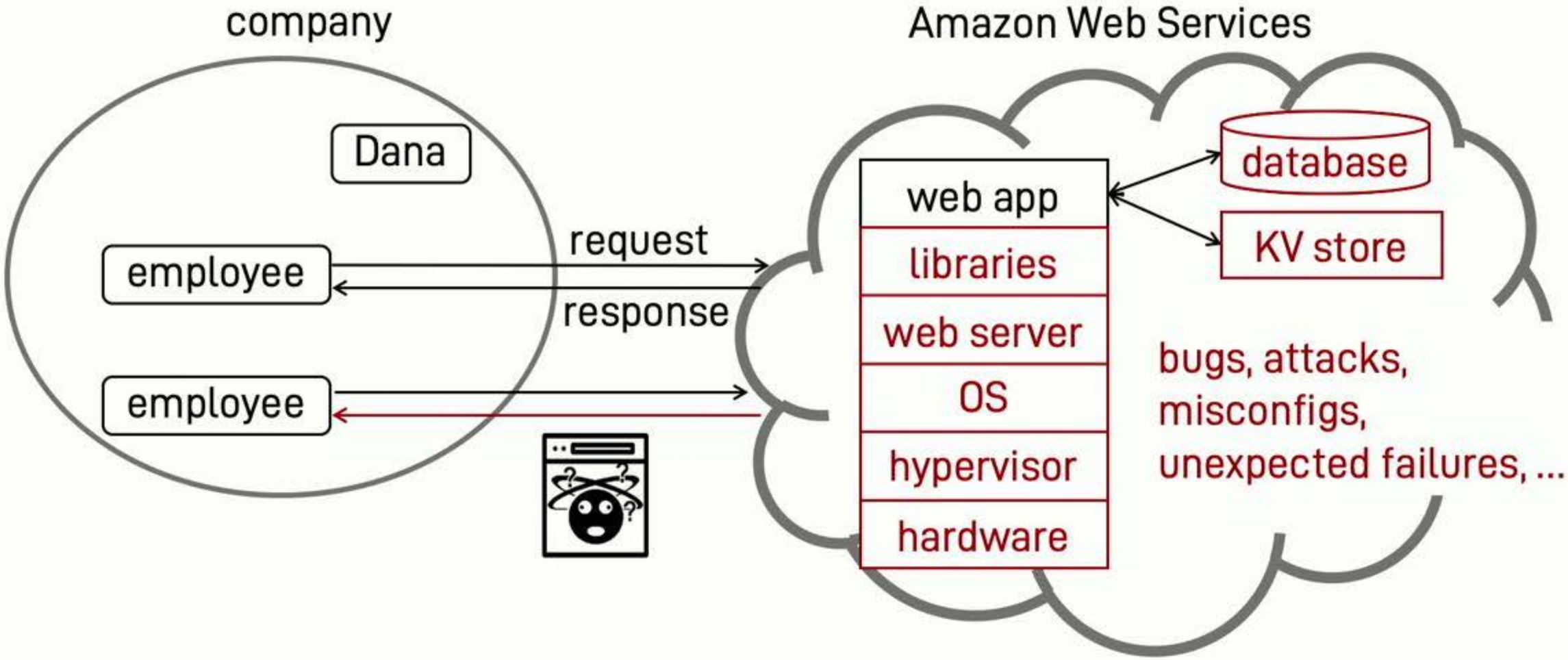
④ Other past work
- Troubleshooting data center networks [NSDI'19]
- Protecting secret via security-oriented offloading [EuroSys'15]


company
web app
Dana
employee
employee
Amazon Web Services

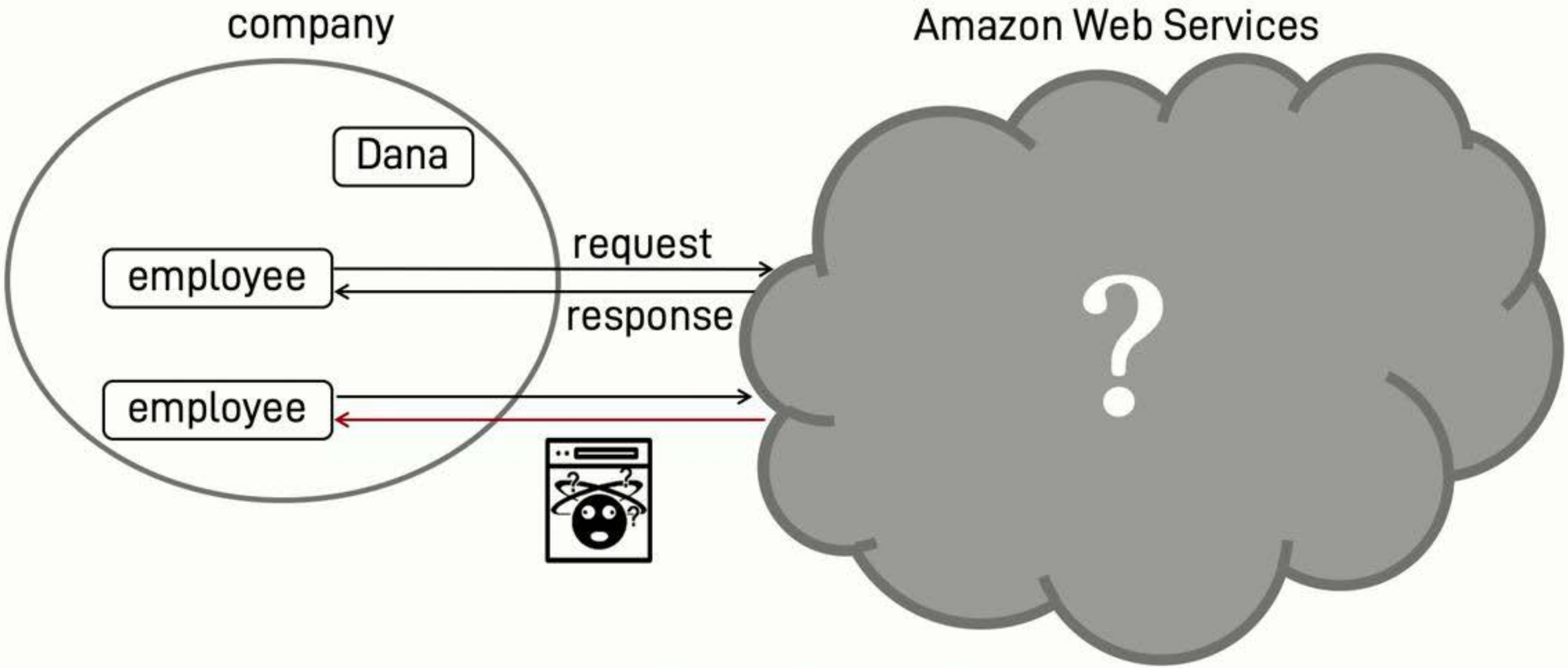

company
Amazon Web Services
Dana
employee
employee
request
response
web app
libraries
web server
OS
hypervisor
hardware
database
KV store



- A lot of things can go wrong...

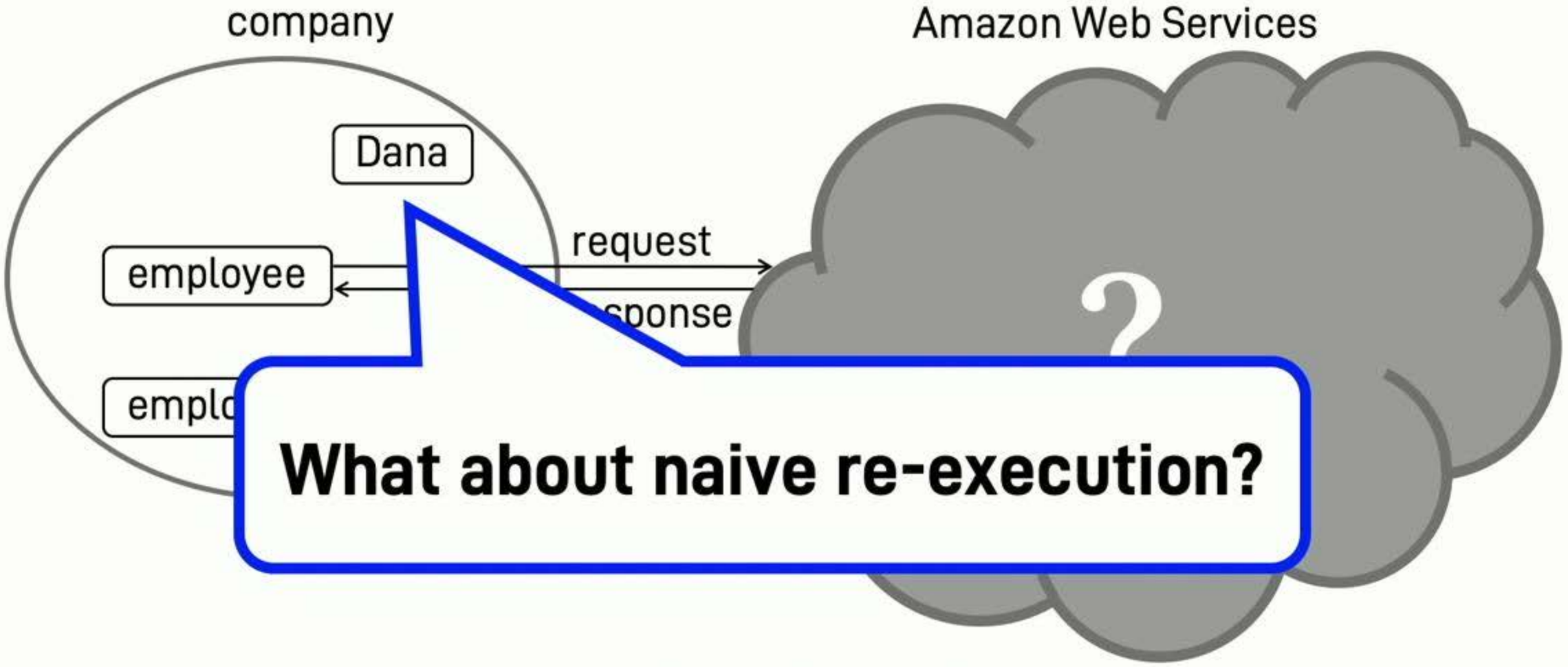


- A lot of things can go wrong...



- Dana wants to audit the end-to-end execution

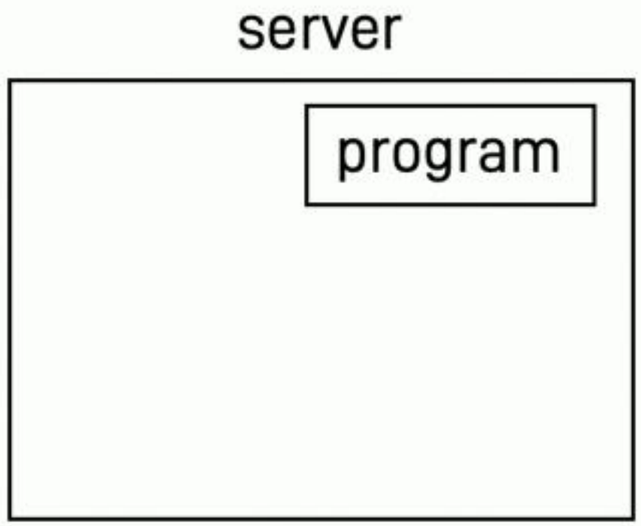
responses $\stackrel{?}{=}$ actual application + requests



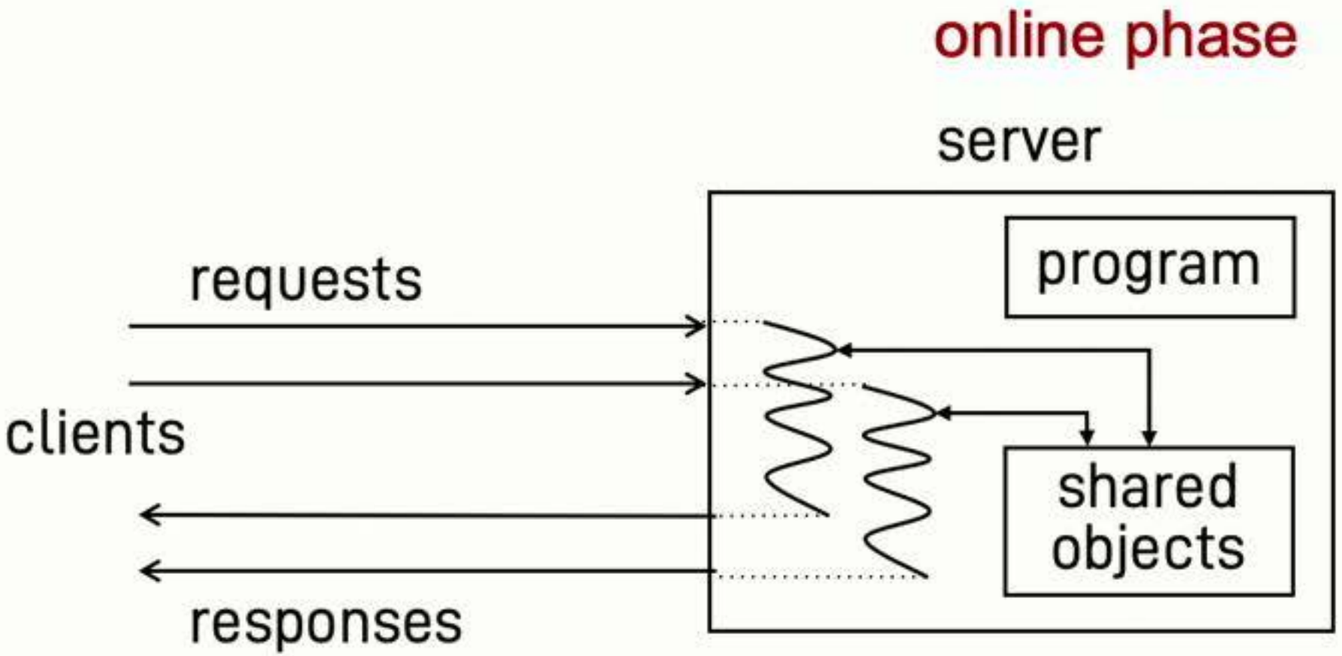
- Dana wants to audit the end-to-end execution

responses $\stackrel{?}{=}$ actual application + requests

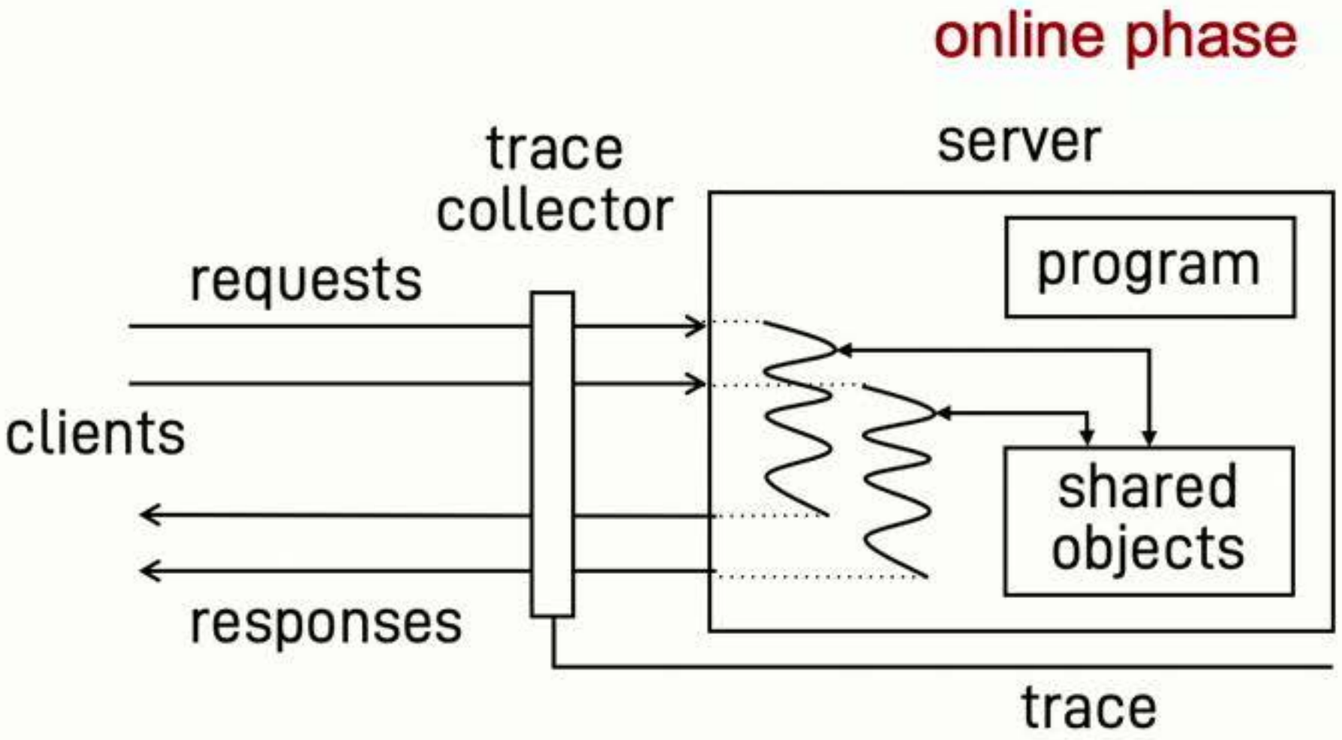
The Efficient Server Audit Problem

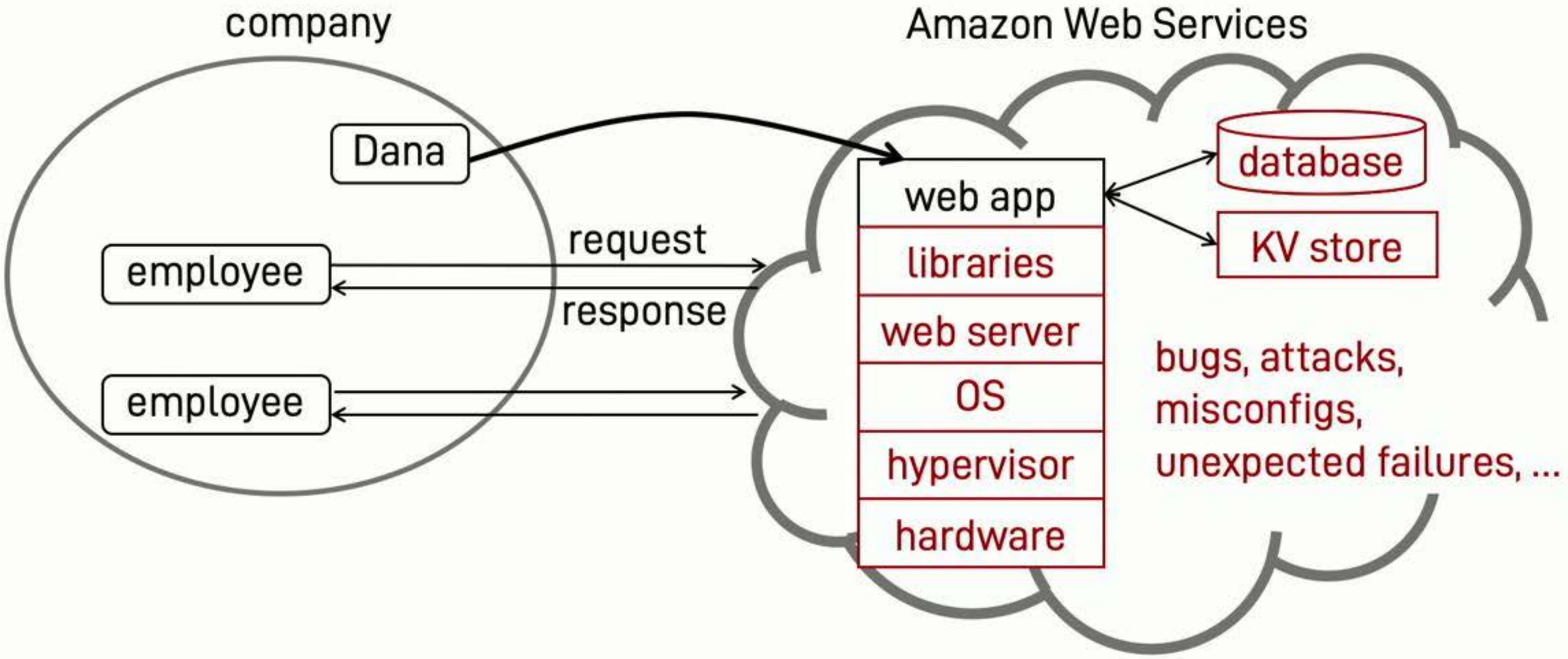


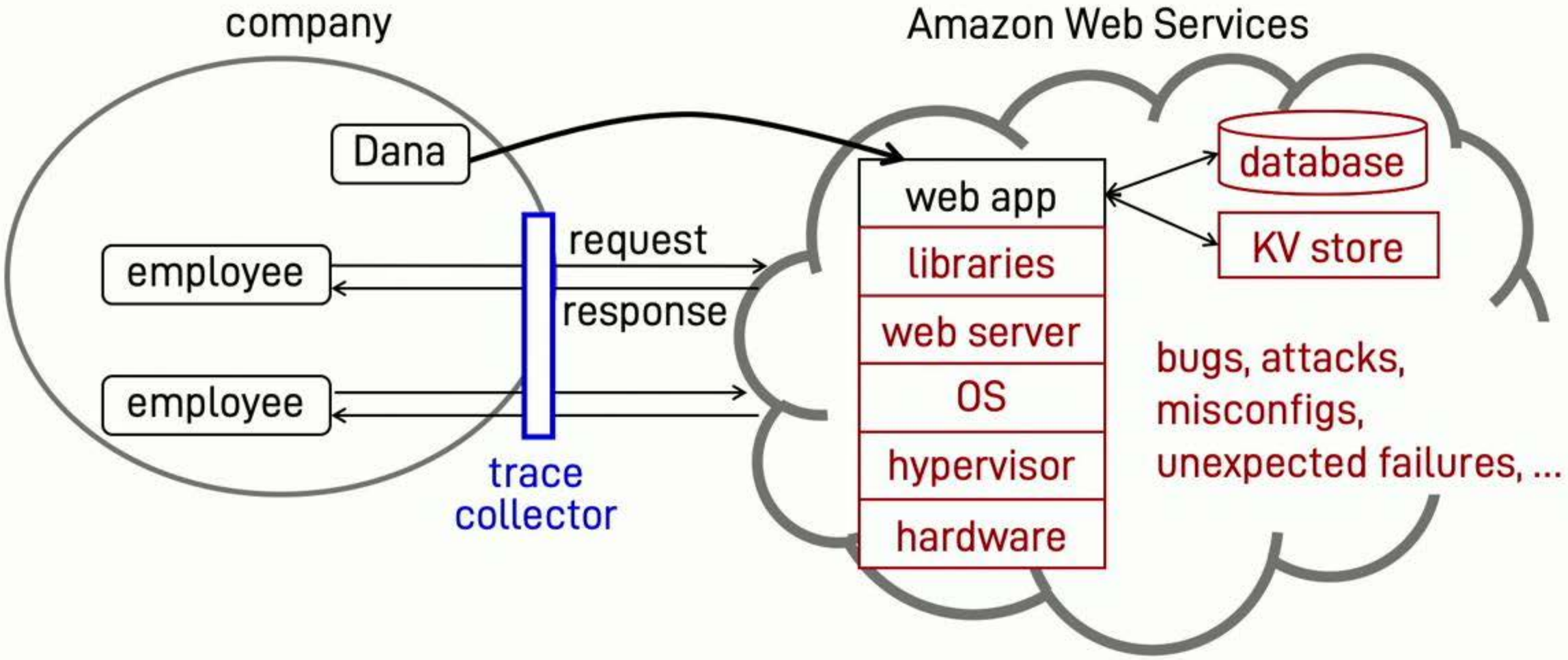
The Efficient Server Audit Problem



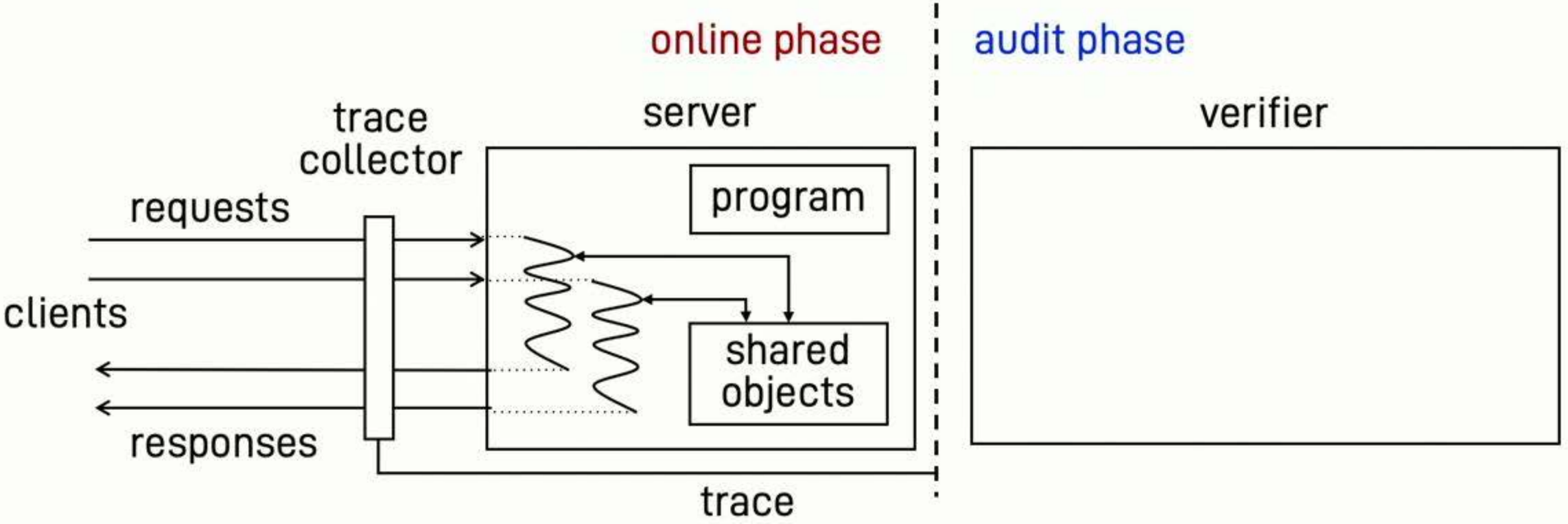
The Efficient Server Audit Problem



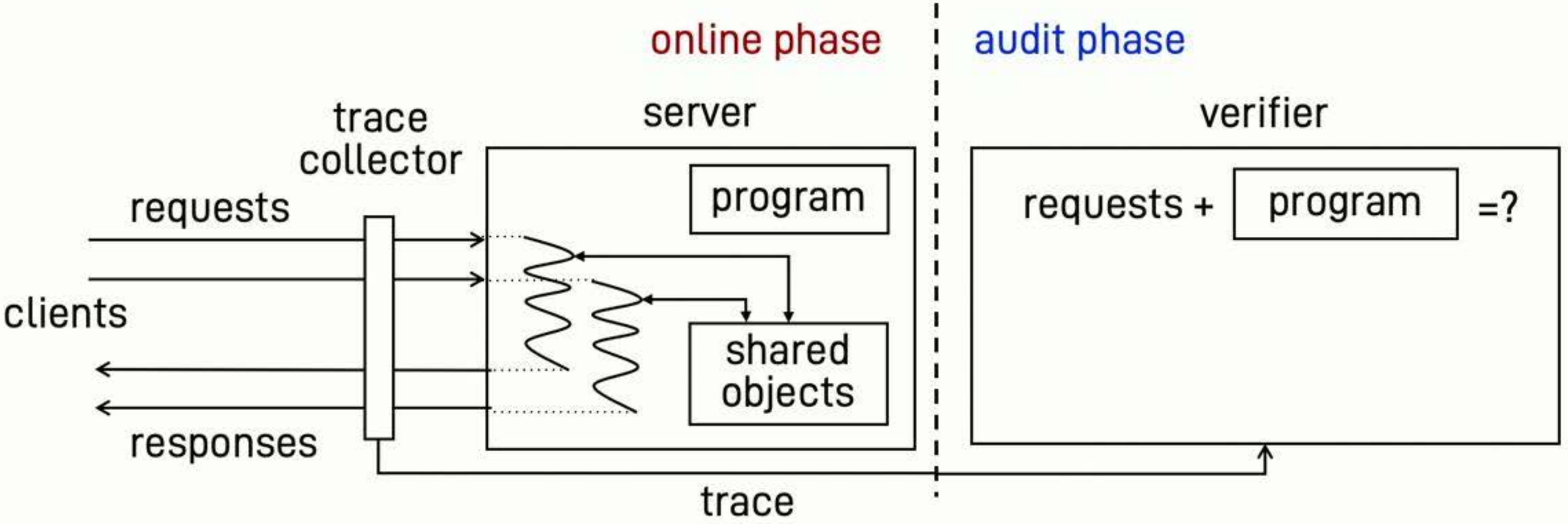

company
Amazon Web Services
Dana
employee
employee
request
response
web app
libraries
web server
OS
hypervisor
hardware
database
KV store
bugs, attacks,
misconfigs,
unexpected failures, ...


company
Amazon Web Services
Dana
employee
employee
request
response
trace
collector
web app
libraries
web server
OS
hypervisor
hardware
database
KV store
bugs, attacks,
misconfigs,
unexpected failures, ...

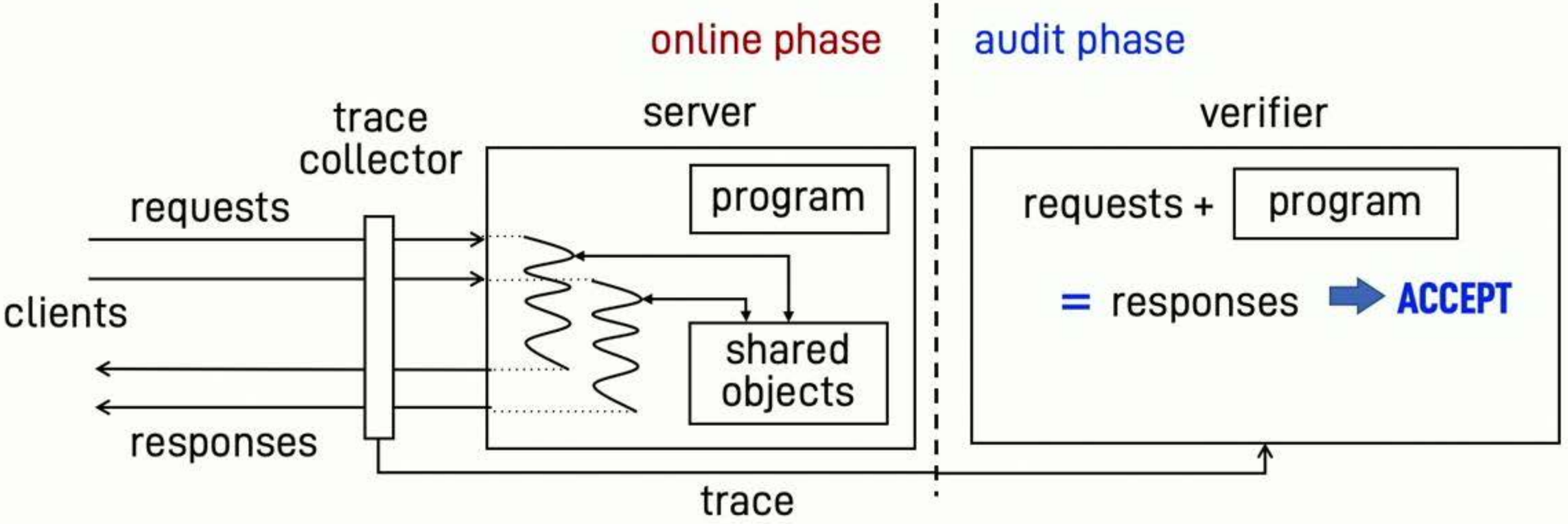
The Efficient Server Audit Problem



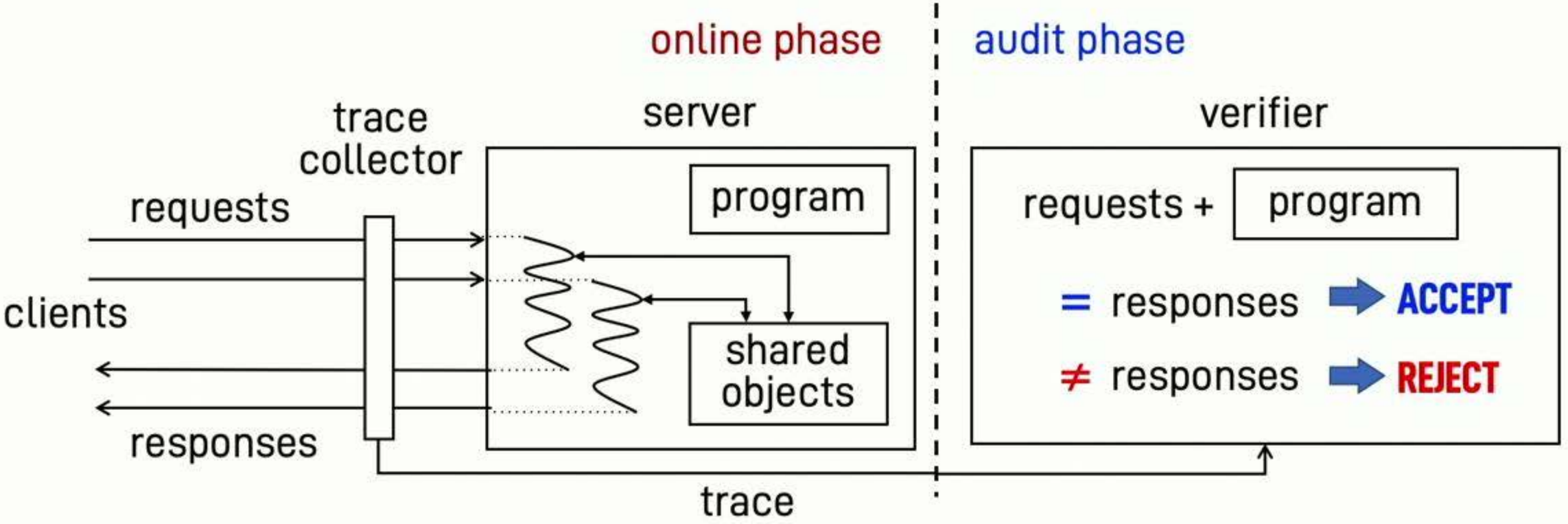
The Efficient Server Audit Problem



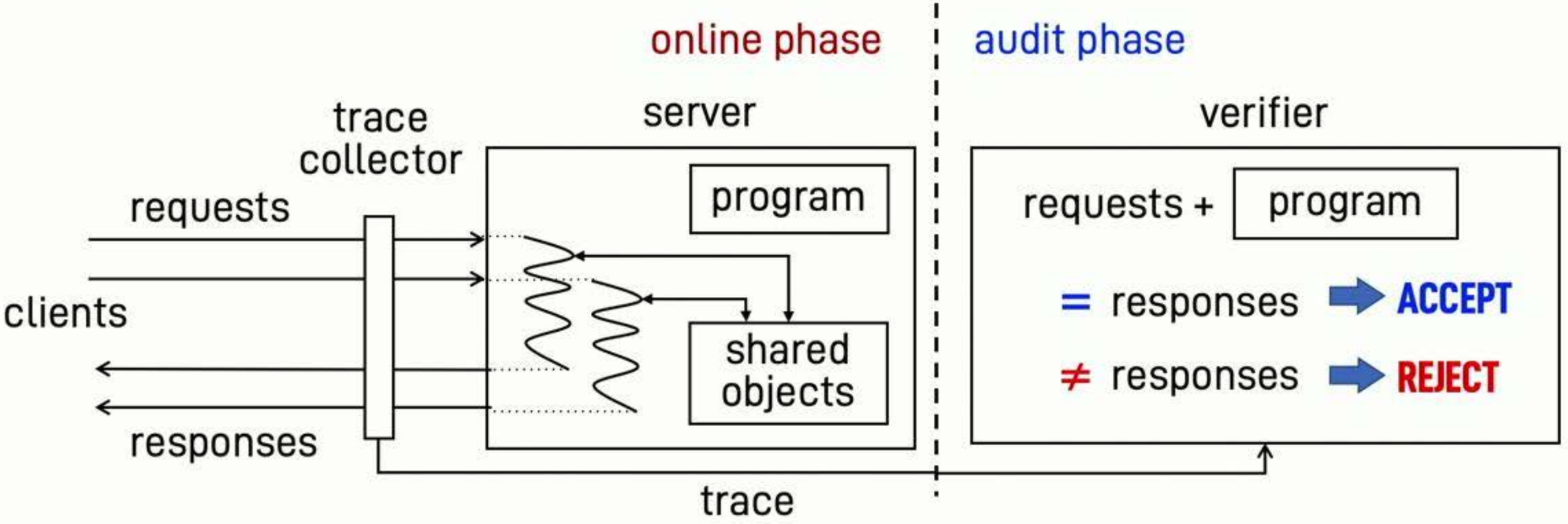
The Efficient Server Audit Problem



The Efficient Server Audit Problem

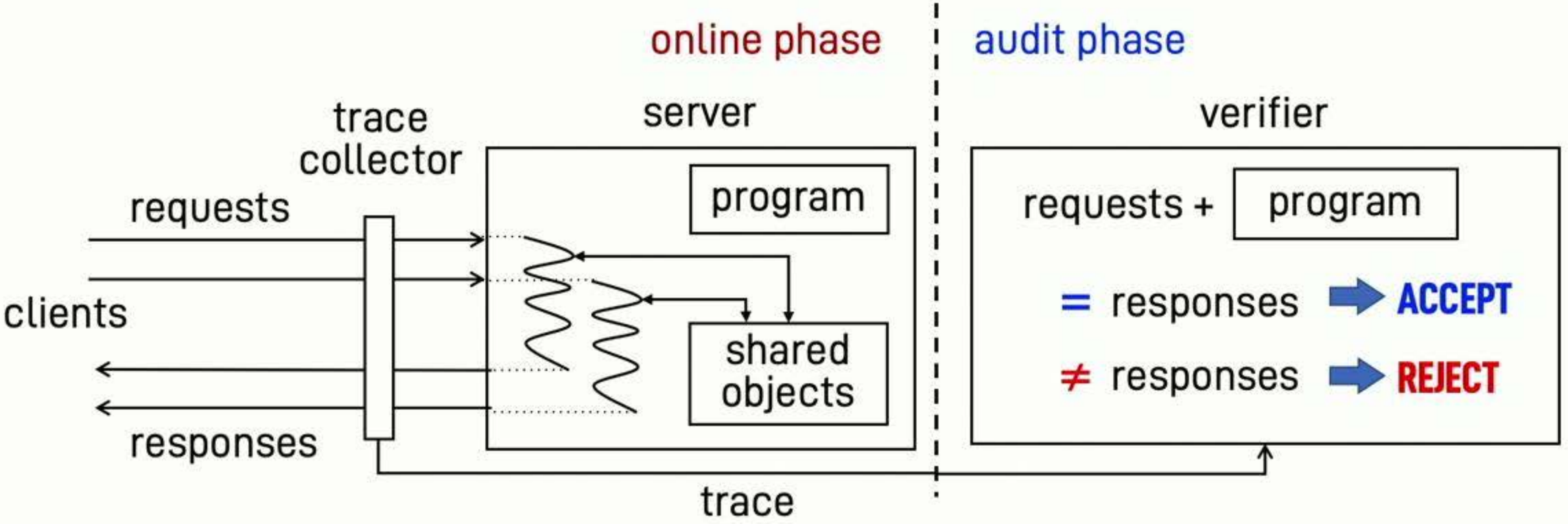


The Efficient Server Audit Problem



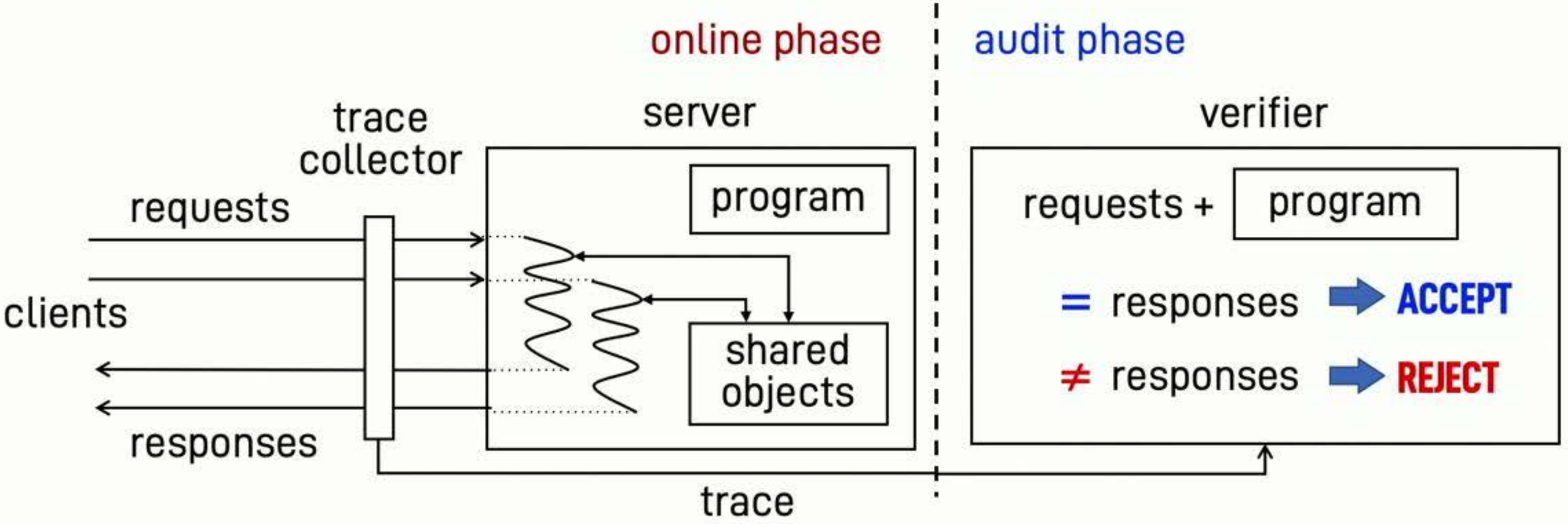
1. server can behave arbitrarily

The Efficient Server Audit Problem



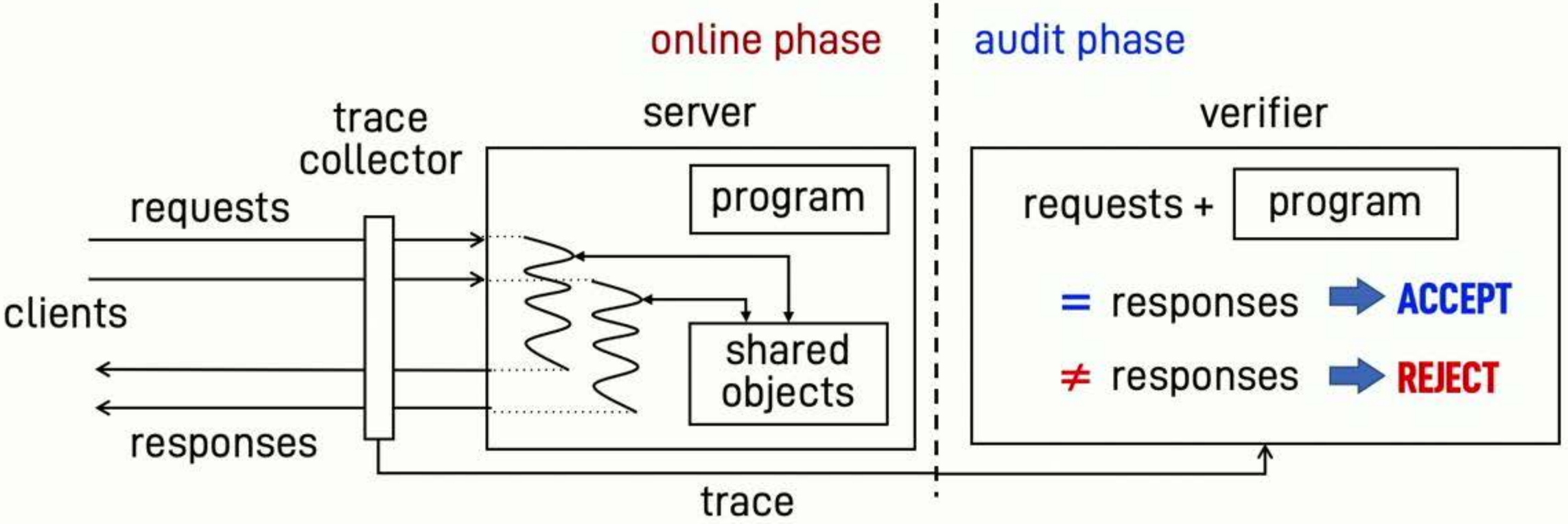
1. server can behave arbitrarily
2. server is concurrent

The Efficient Server Audit Problem



1. server can behave arbitrarily
2. server is concurrent
3. verifier requires less computation power than server

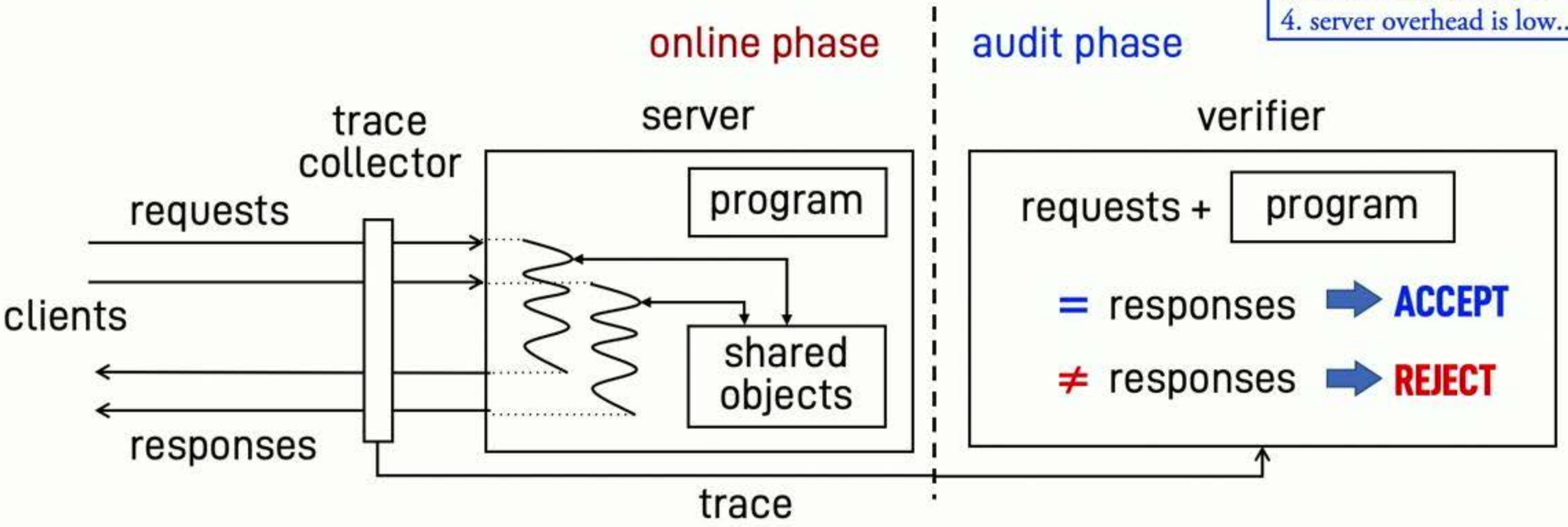
The Efficient Server Audit Problem



1. server can behave arbitrarily
2. server is concurrent
3. verifier requires less computation power than server
4. server overhead is low; legacy applications supported

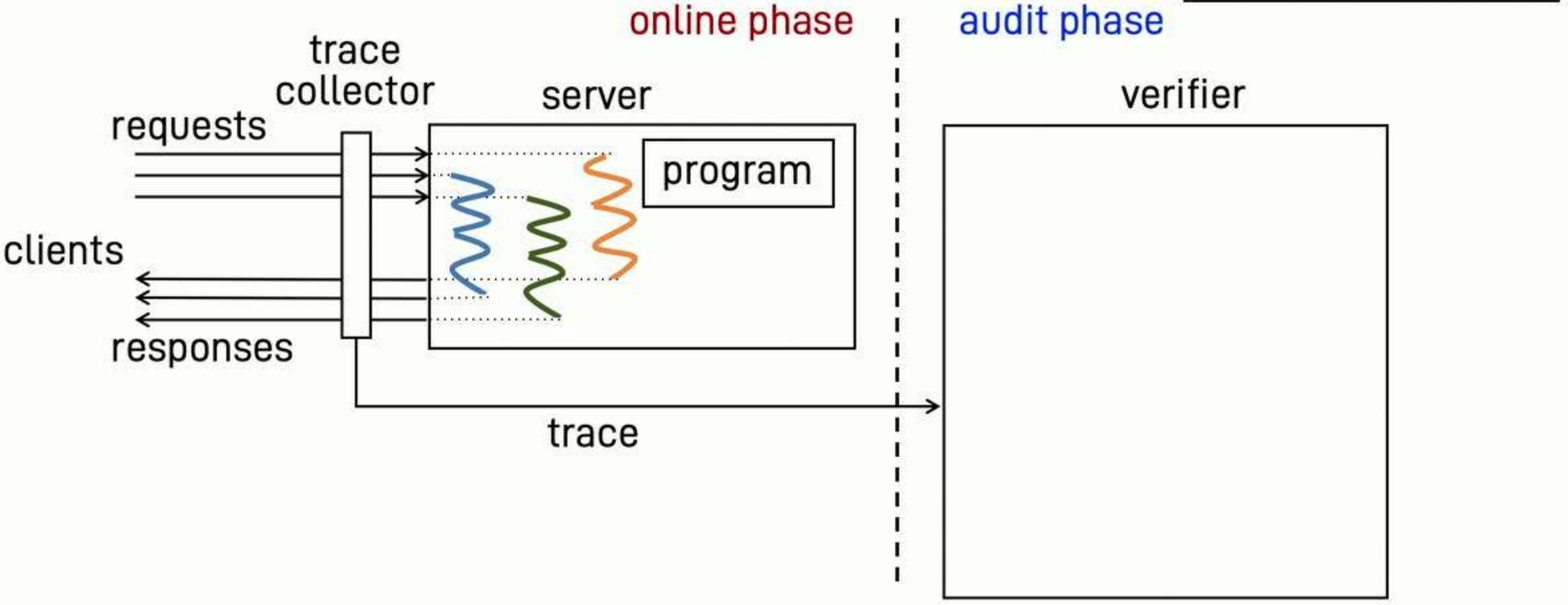
The Efficient Server Audit Problem

1. server is untrusted…
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low…



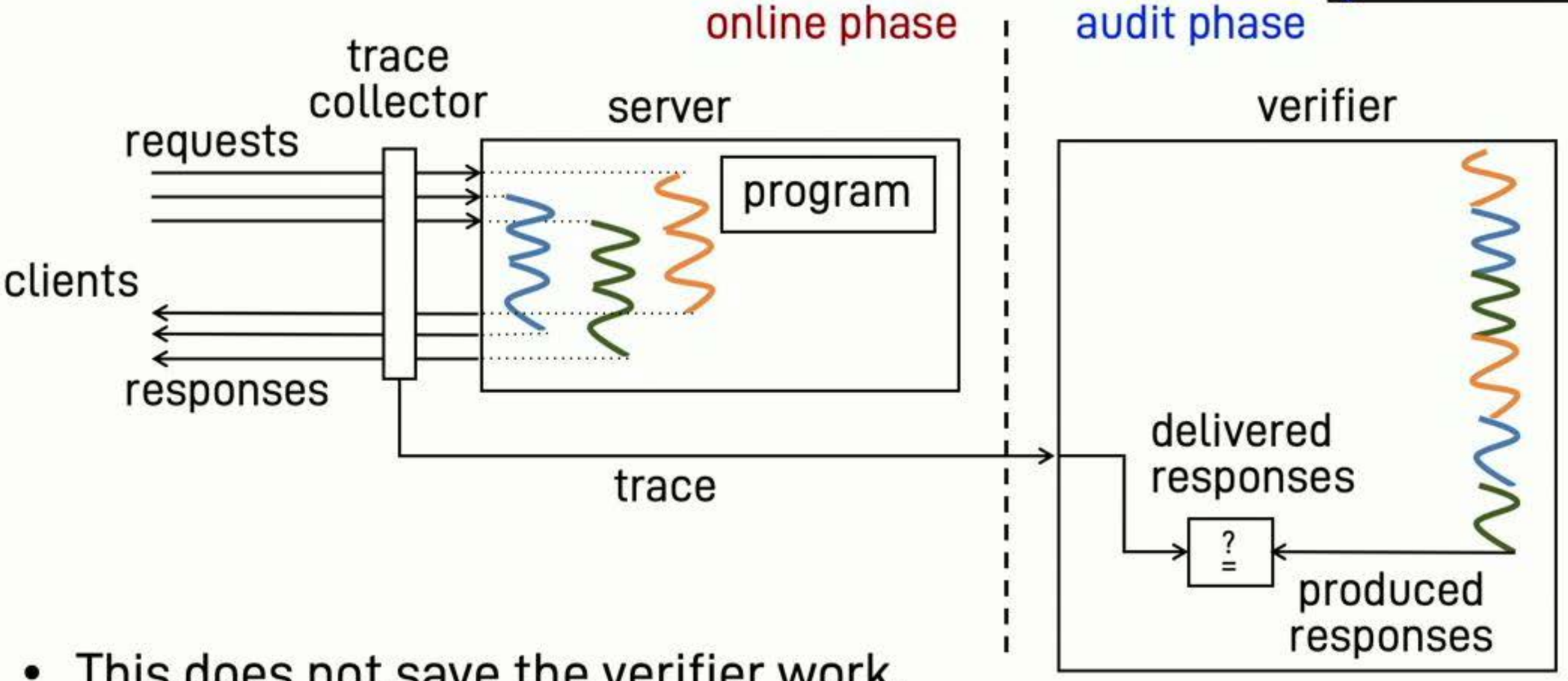
What about naive re-execution?

1. server is untrusted…
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low...



What about naive re-execution?

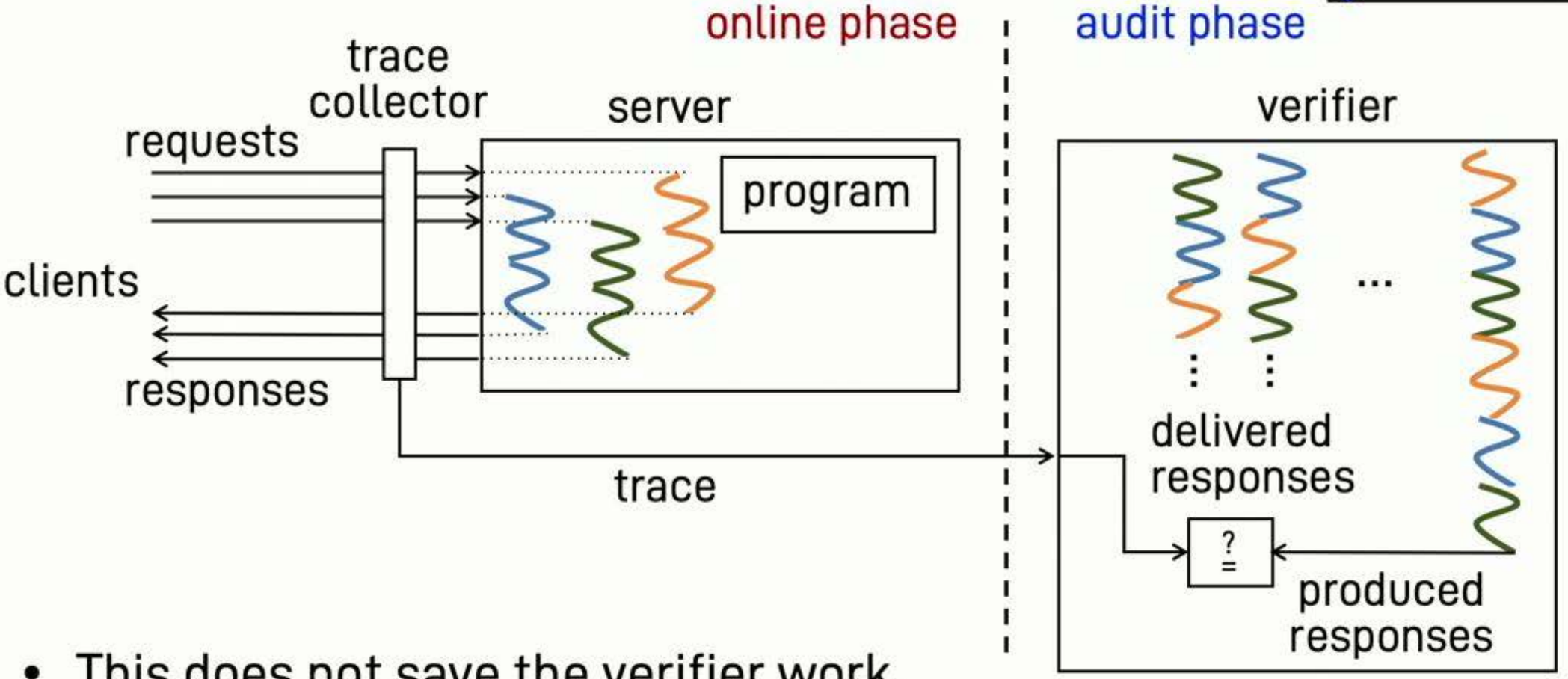
✓ server is untrusted…
? server is concurrent
✗ verifier is weaker than server
✓ server overhead is low…



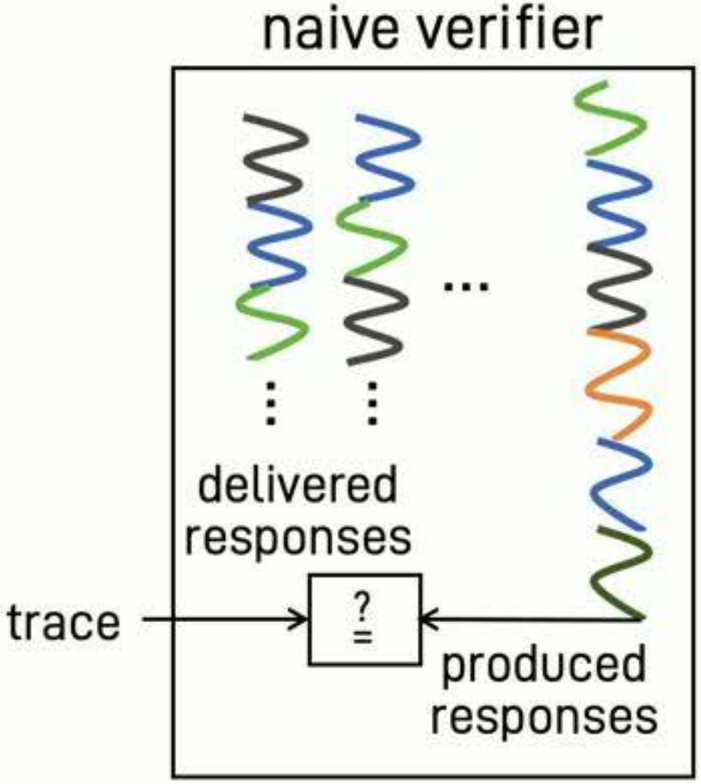
- This does not save the verifier work.

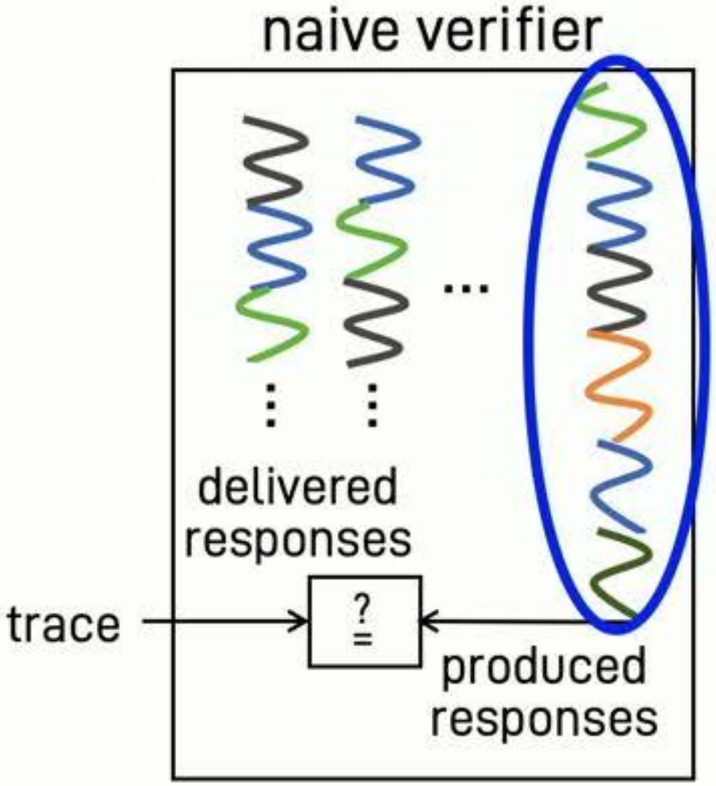
What about naive re-execution?

✓. server is untrusted…
?. server is concurrent
✗ verifier is weaker than server
✓. server overhead is low…



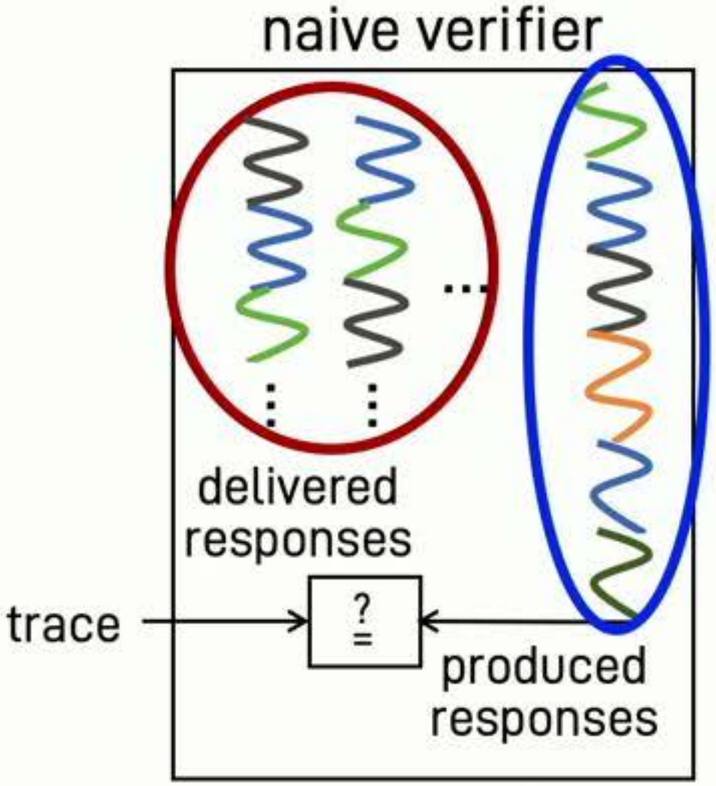
- This does not save the verifier work.
- Due to concurrency, verifier must explore many schedules.


naive verifier
delivered
responses
trace
?
=
produced
responses



Problem: naive re-execution doesn't save work

Proposal: (somehow) accelerate re-execution

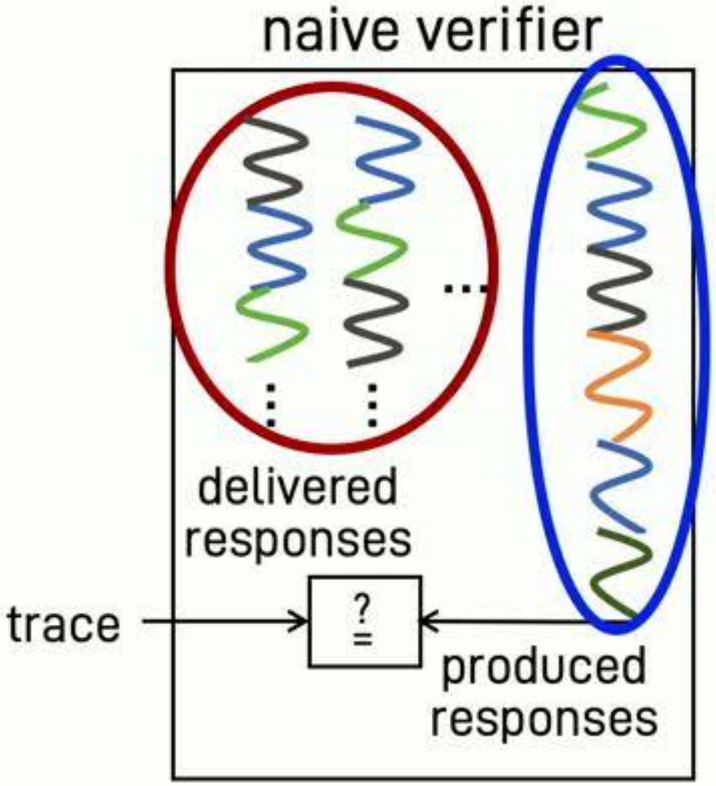


Problem: naive re-execution doesn't save work

Proposal: (somehow) accelerate re-execution

Problem: many schedules to explore

Proposal: ask server for advice, use-and-check it

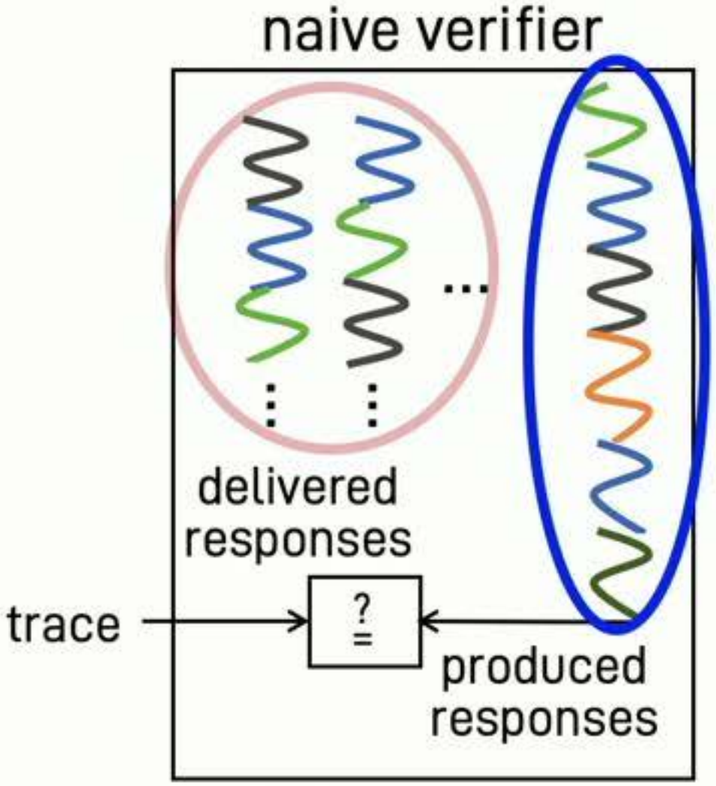


Problem: naive re-execution doesn't save work

Proposal: (somehow) accelerate re-execution

Problem: many schedules to explore

Proposal: ask server for advice, use-and-check it



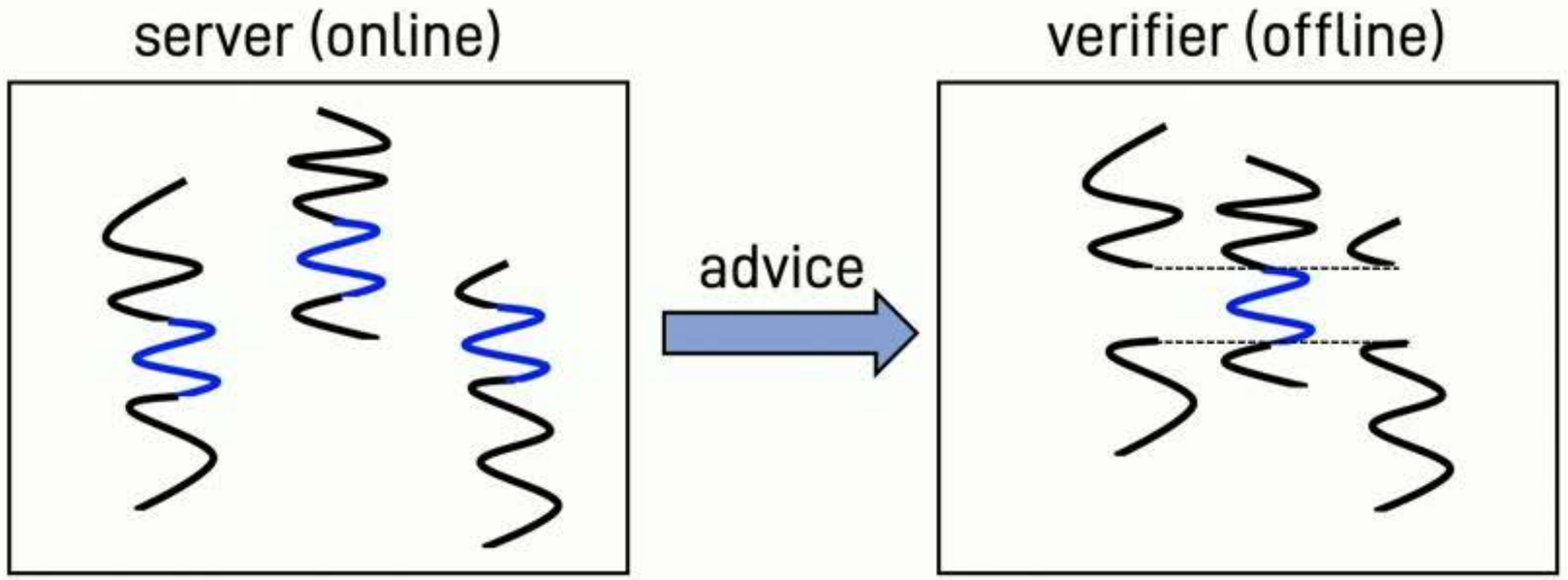
Problem: naive re-execution doesn't save work

Proposal: (somehow) accelerate re-execution

Problem: many schedules to explore

Proposal: ask server for advice, use-and-check it

Accelerating re-execution: a 30,000-foot view



deduplicate computation across requests

T. Kim, R. Chandra, and N. Zeldovich.
Efficient patch-based auditing for web applications. *OSDI*, 2012

An observation: repeated computation

web app:
HotCRP

page 1

page 2

T. Kim, R. Chandra, and N. Zeldovich.
Efficient patch-based auditing for web applications. *OSDI*, 2012

An observation: repeated computation

web app:
HotCRP

page 1

page 2

T. Kim, R. Chandra, and N. Zeldovich.
Efficient patch-based auditing for web applications. *OSDI*, 2012

An observation: repeated computation

web app:
HotCRP

page 1

page 2

have the same control flow

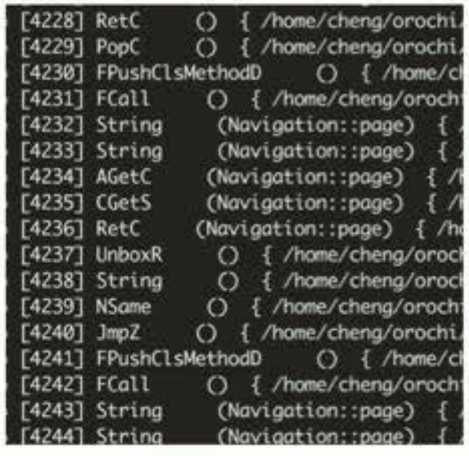
T. Kim, R. Chandra, and N. Zeldovich.
Efficient patch-based auditing for web applications. *OSDI*, 2012

An observation: repeated computation

web app:
HotCRP

page 1

page 2



opcode **operands**
e.g., ADD 1, 2

have the same control flow

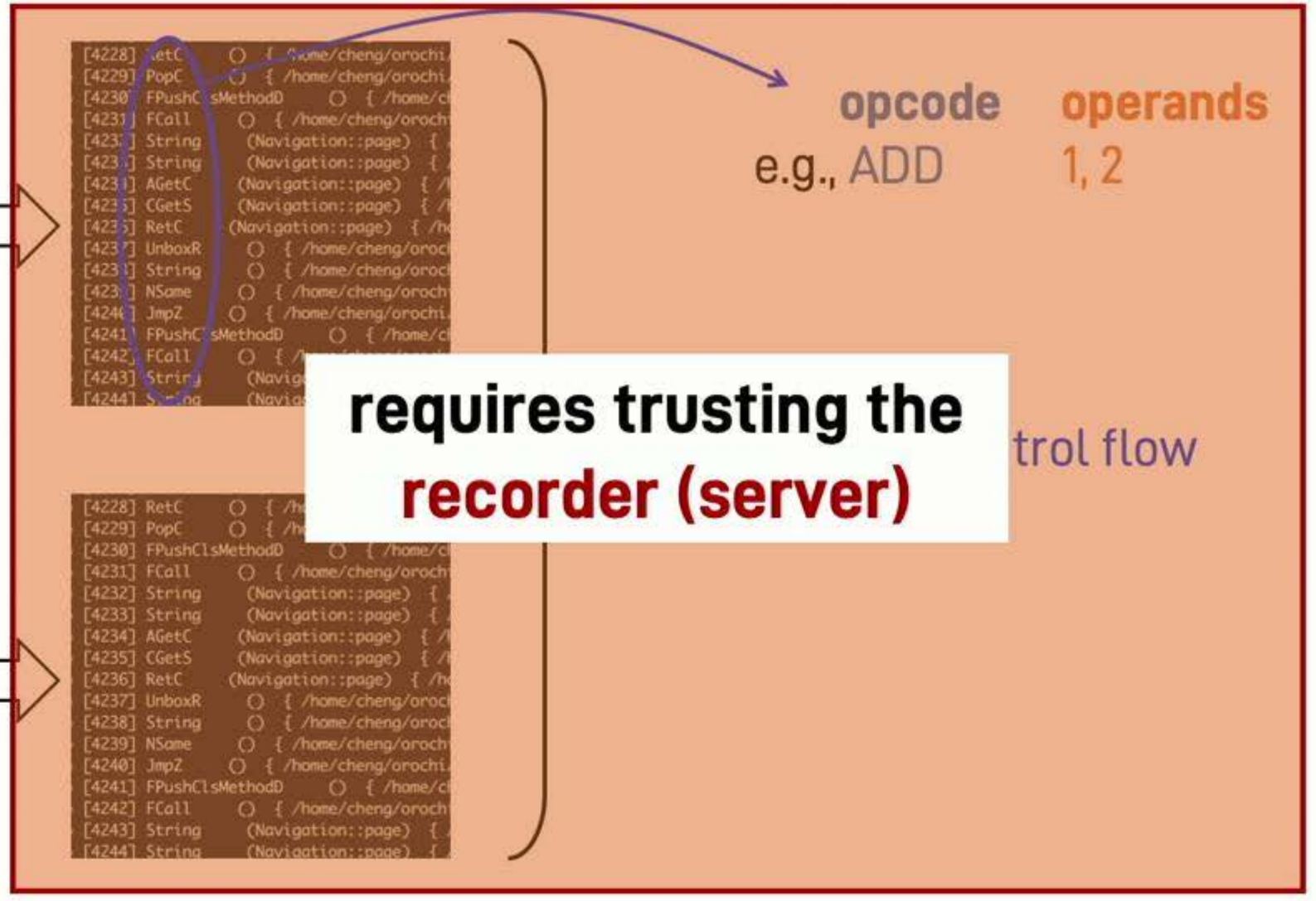
T. Kim, R. Chandra, and N. Zeldovich.
Efficient patch-based auditing for web applications. *OSDI*, 2012

An observation: repeated computation

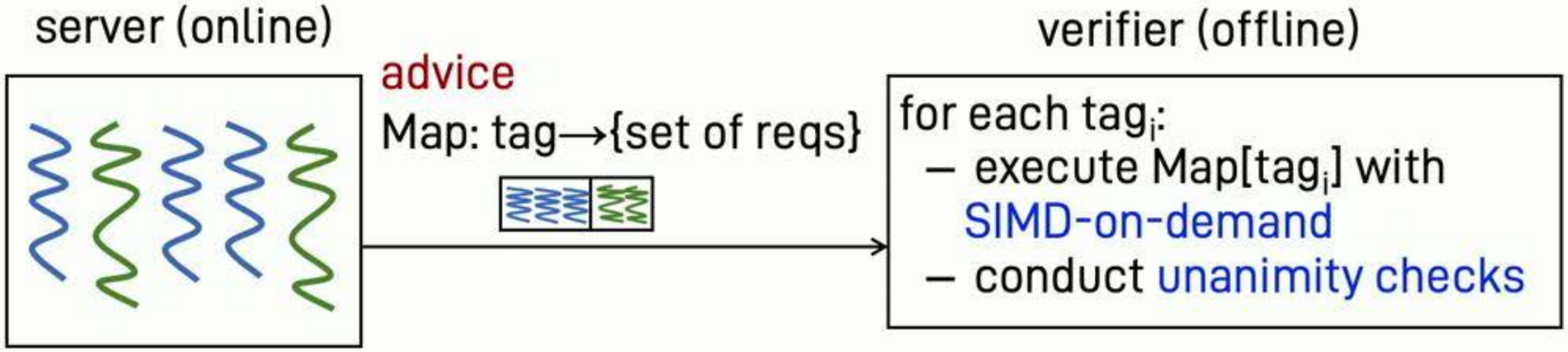
web app:
HotCRP

page 1

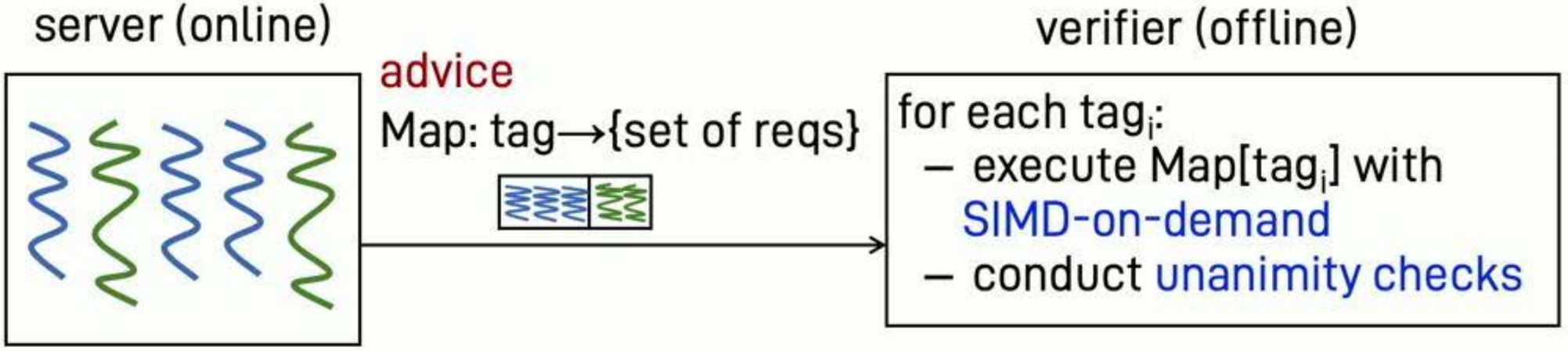
page 2



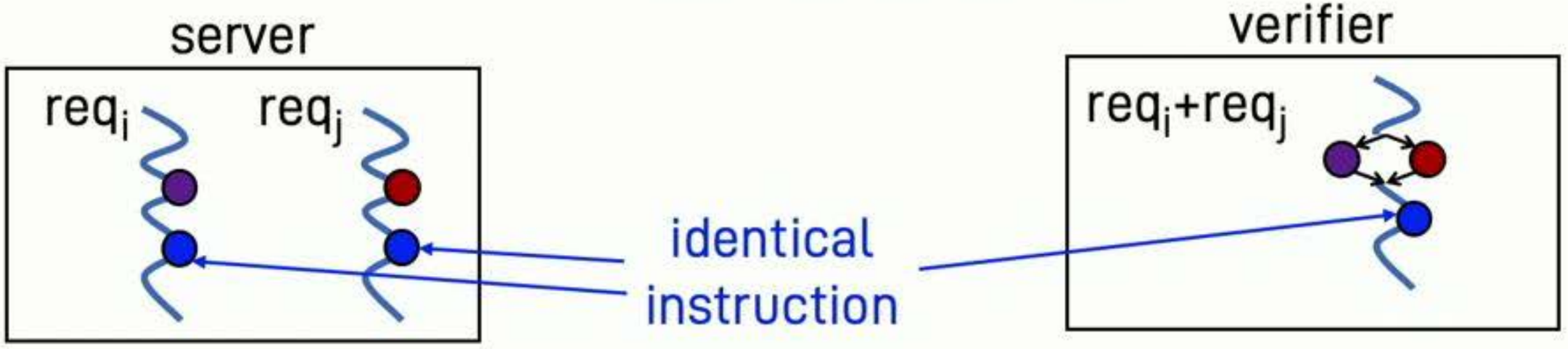
Accelerate re-execution without trusting the server



Accelerate re-execution without trusting the server



- SIMD-on-demand re-executes identical instructions once.



SIMD-on-demand eliminates redundant computation

```
main(a, b):
  c ← a * b
  c ← c + 1
```

$req_{1:}$ a=1; b=2

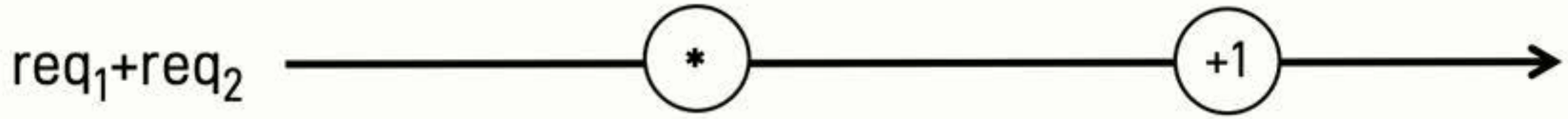
$req_{2:}$ a=2; b=1

SIMD-on-demand eliminates redundant computation

```
main(a, b):
  c ← a * b
  c ← c + 1
```

req_1: a=1; b=2

req_2: a=2; b=1

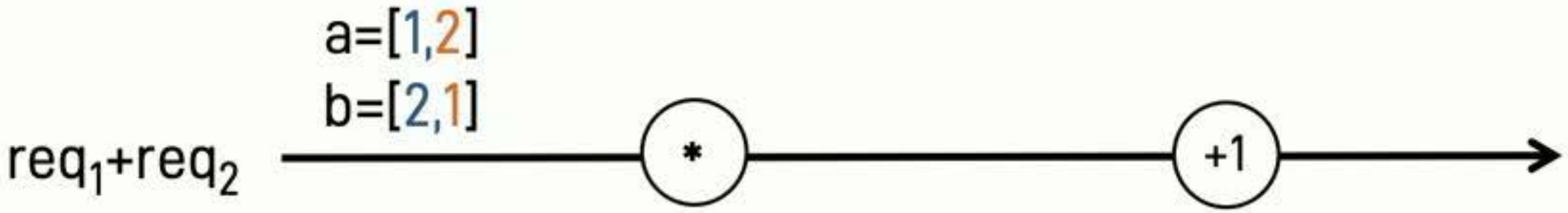


SIMD-on-demand eliminates redundant computation

```
main(a, b):
  c ← a * b
  c ← c + 1
```

req_1: a=1; b=2

req_2: a=2; b=1



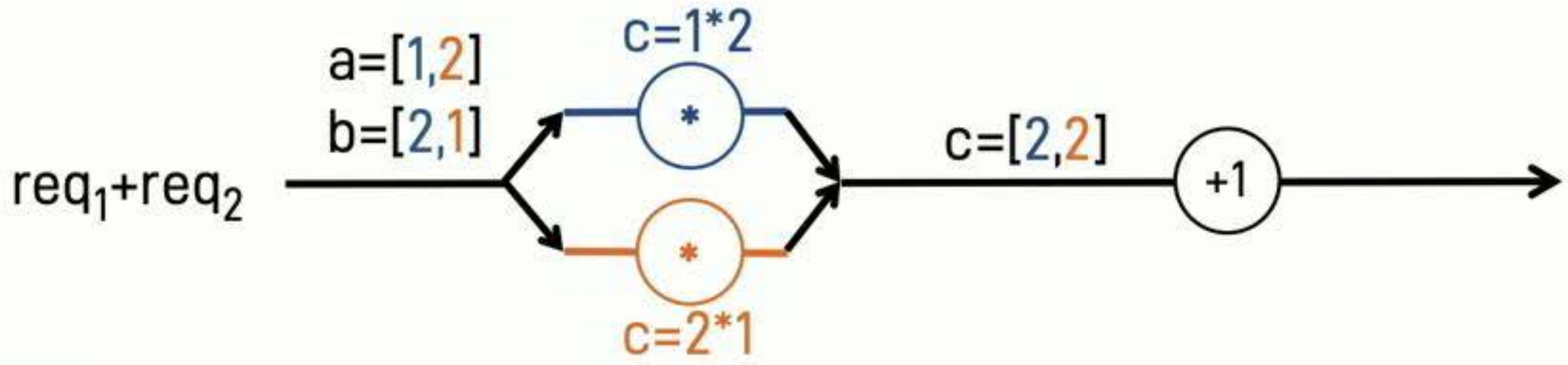
- Multi-value represents different values of the same variable.

SIMD-on-demand eliminates redundant computation

```
main(a, b):
  c ← a * b
  c ← c + 1
```

req_1: a=1; b=2

req_2: a=2; b=1



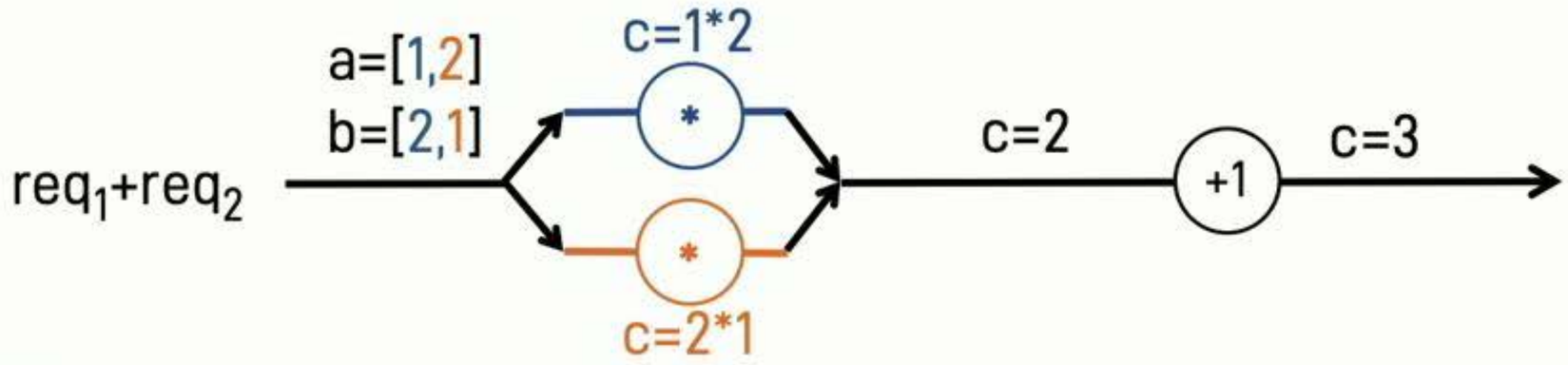
- Multi-value represents different values of the same variable.

SIMD-on-demand eliminates redundant computation

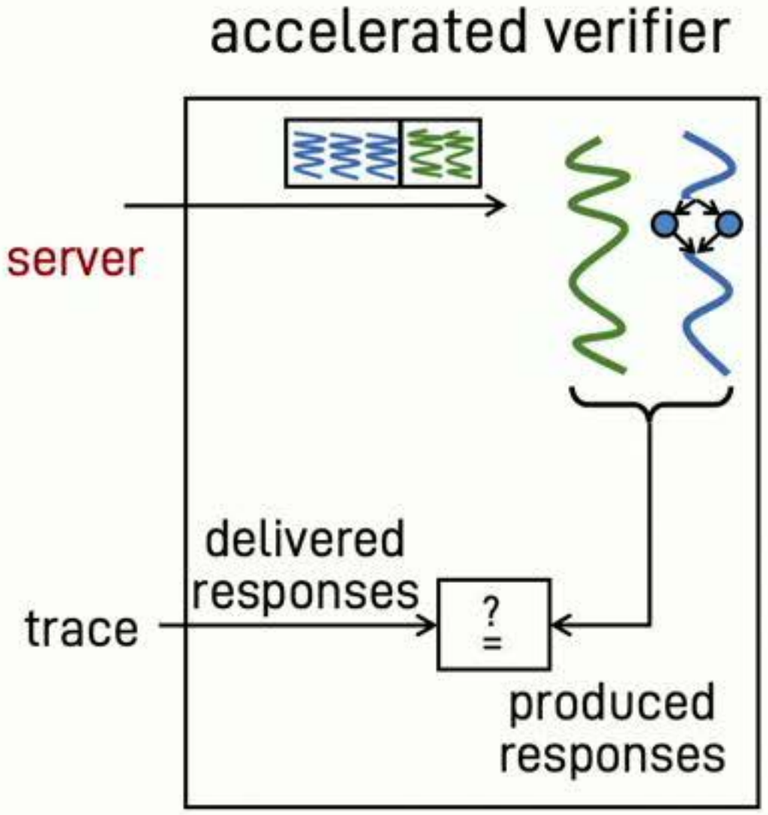
```
main(a, b):
  c ← a * b
  c ← c + 1
```

req_1: a=1; b=2

req_2: a=2; b=1

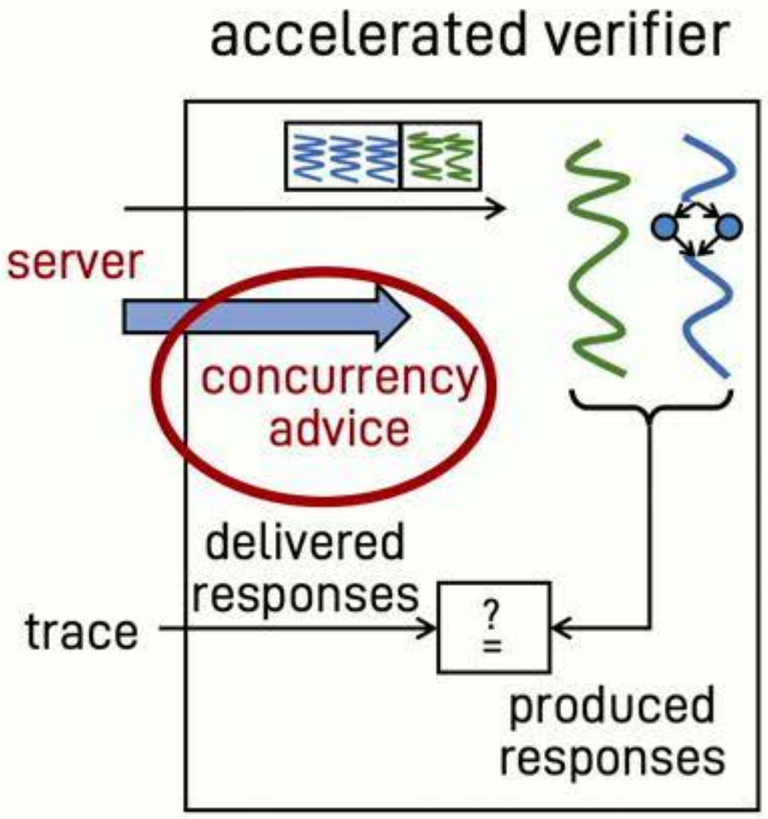


- Multi-value represents different values of the same variable.
- Verifier collapses multi-value to scalar if possible.



Problem: naive re-execution doesn't save work

Solution: deduplicated re-execution



Problem: naive re-execution doesn't save work

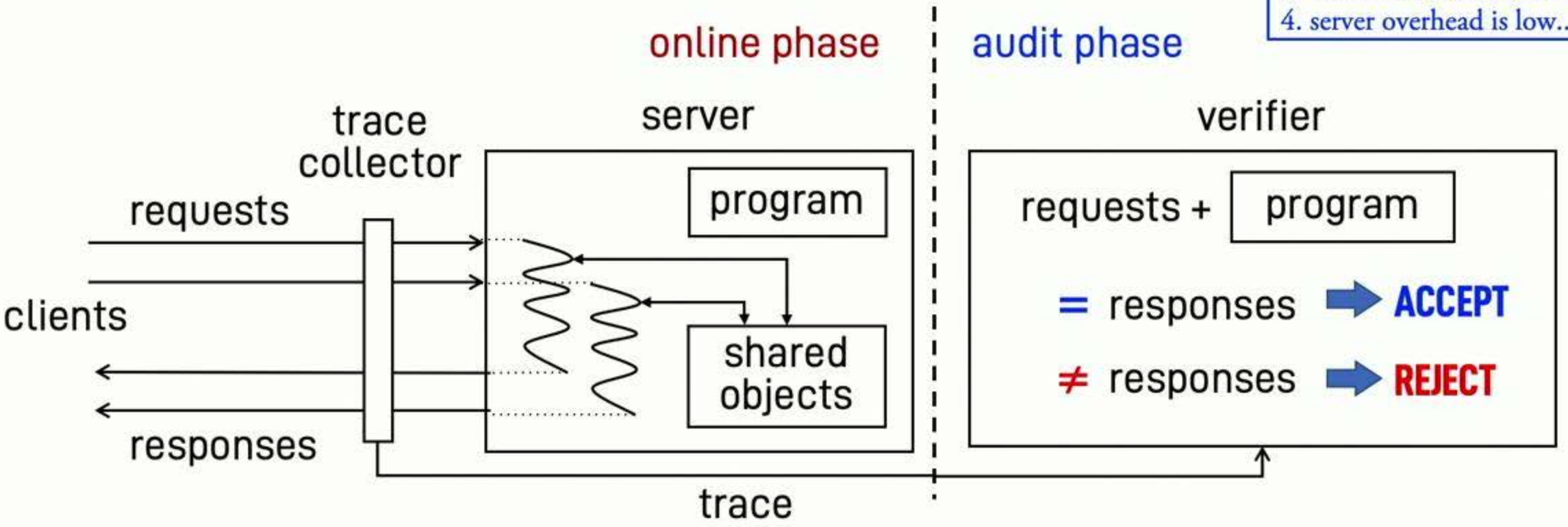
Solution: deduplicated re-execution

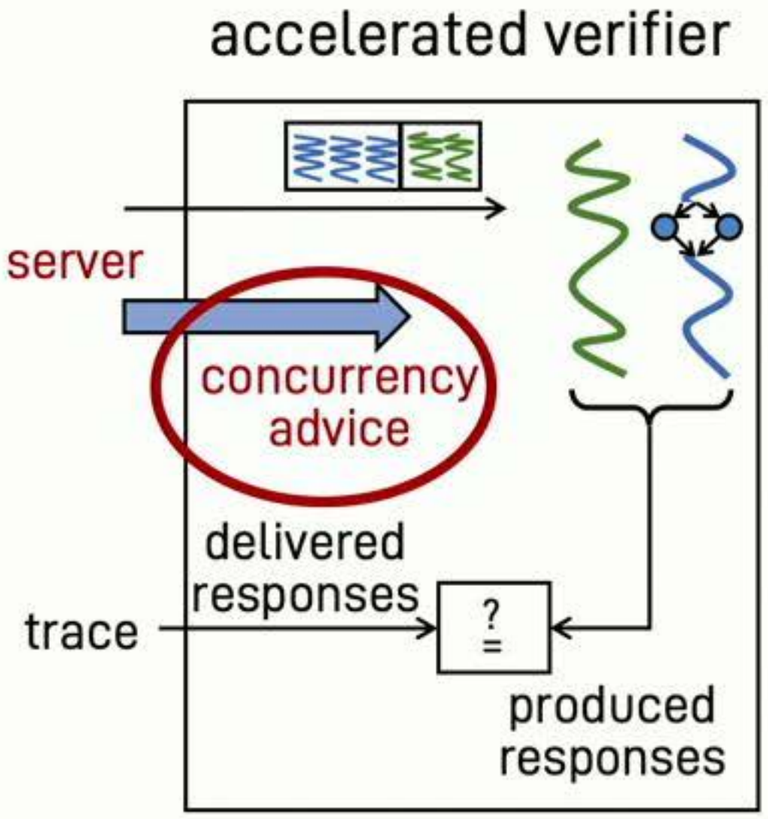
Problem: many schedules to explore

Proposal: (somehow) use but don't trust advice

The Efficient Server Audit Problem

1. server is untrusted…
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low…





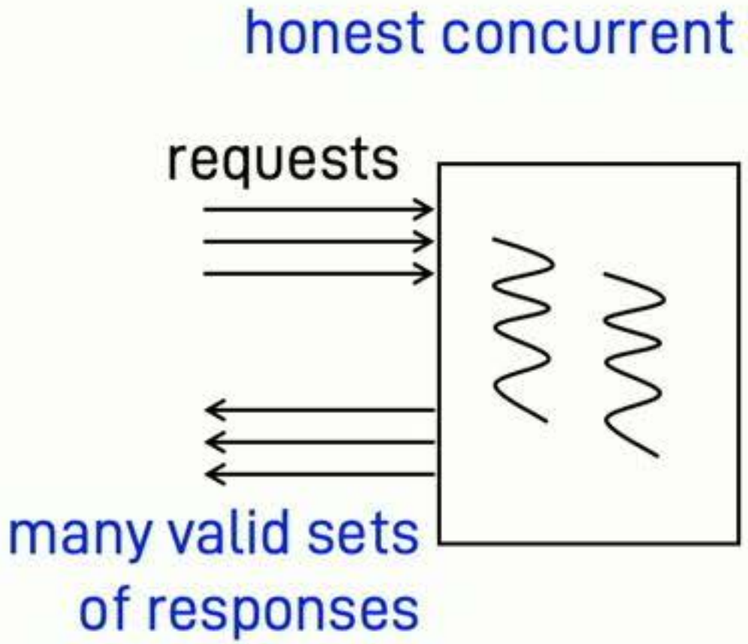
Problem: naive re-execution doesn't save work

Solution: deduplicated re-execution

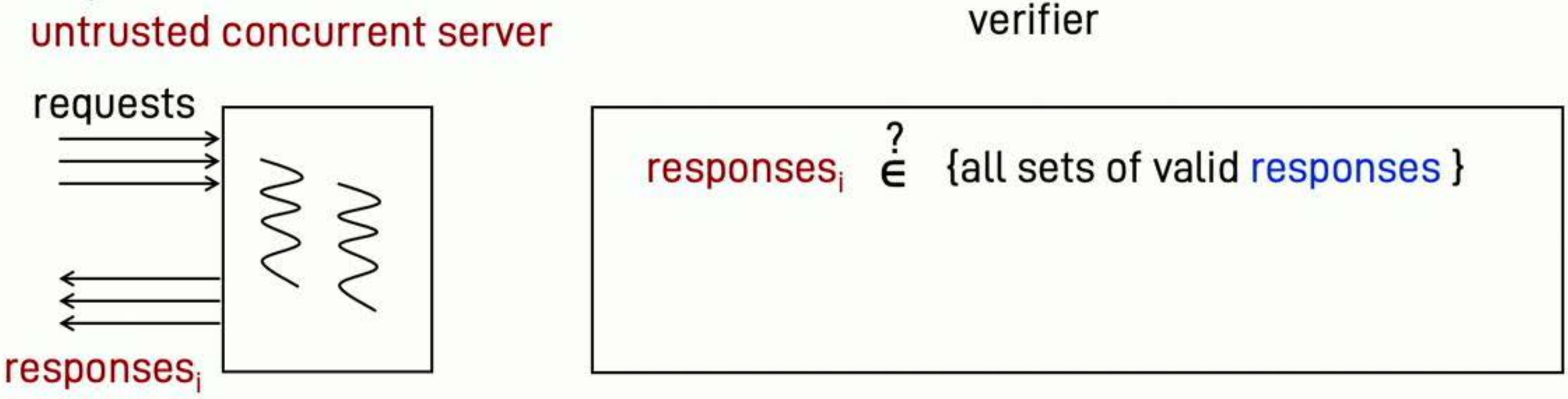
Problem: many schedules to explore

Proposal: (somehow) use but don't trust advice

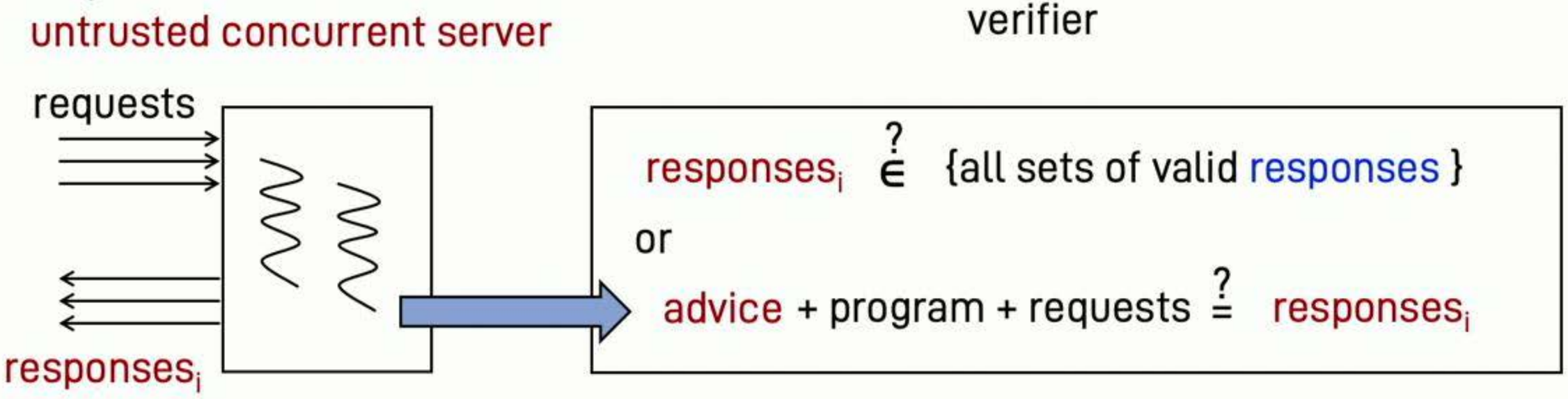
Concurrency and untrusted server are in tension



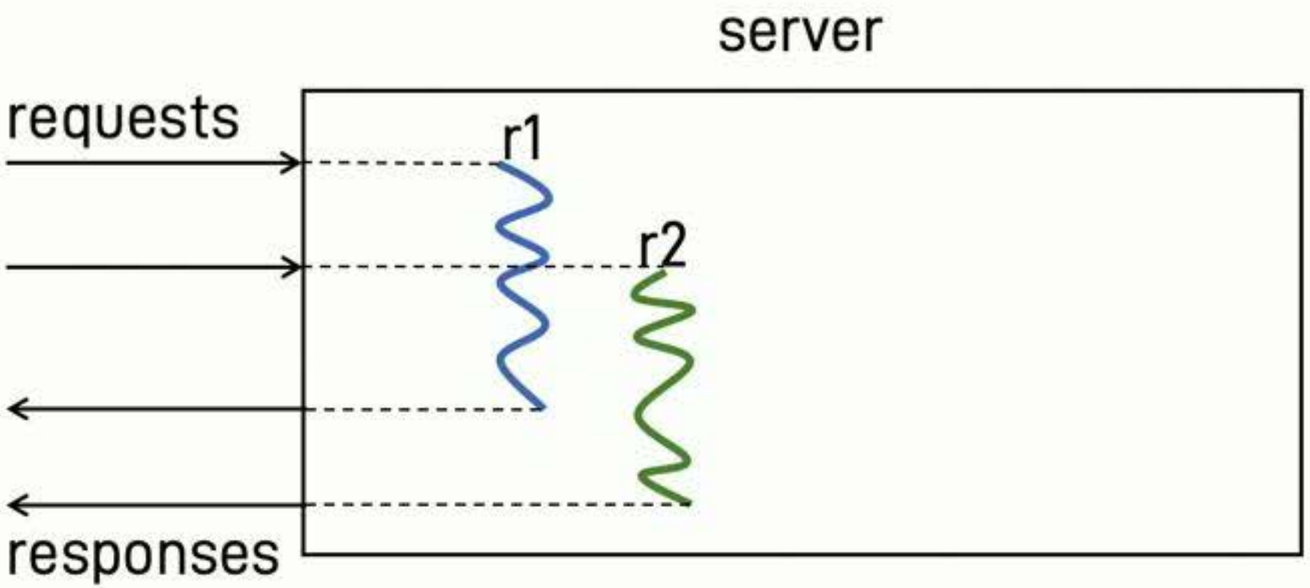
Concurrency and untrusted server are in tension



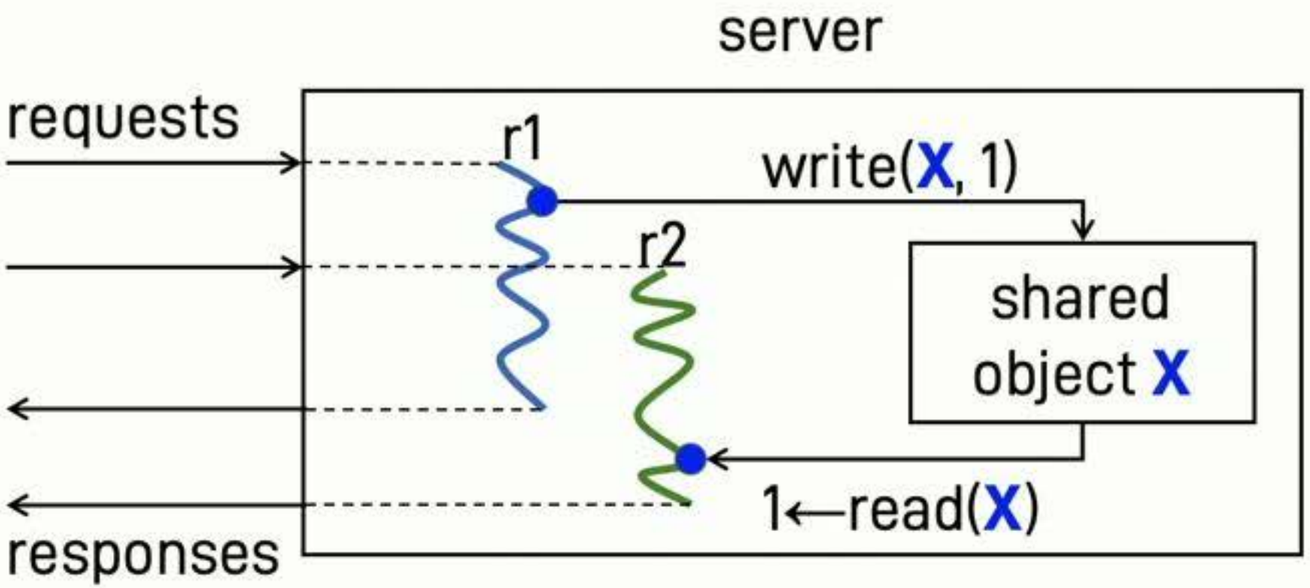
Concurrency and untrusted server are in tension



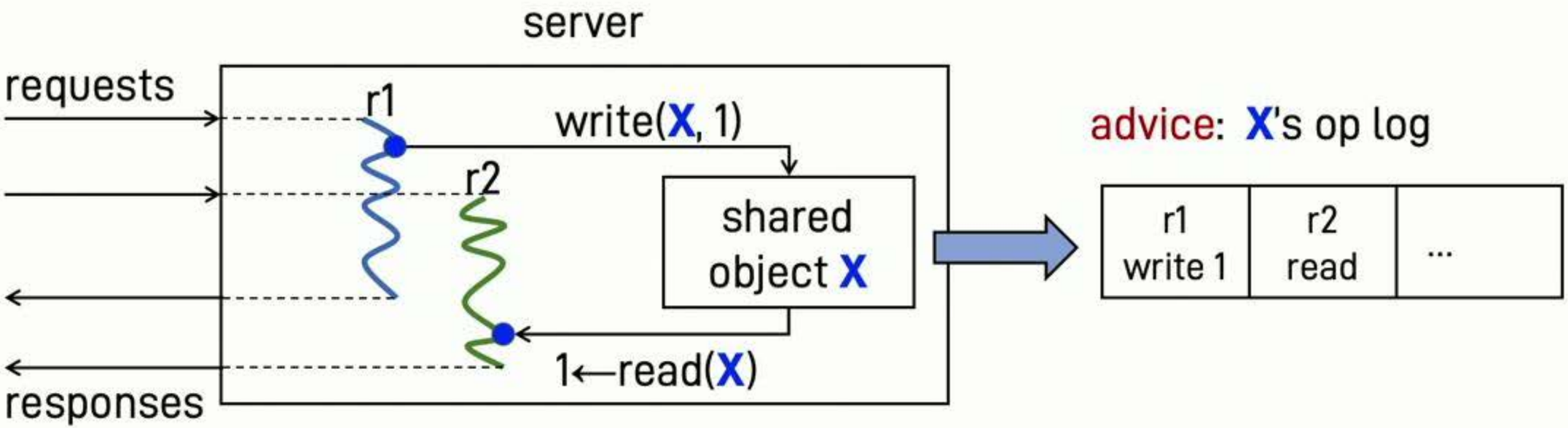
Concurrency model



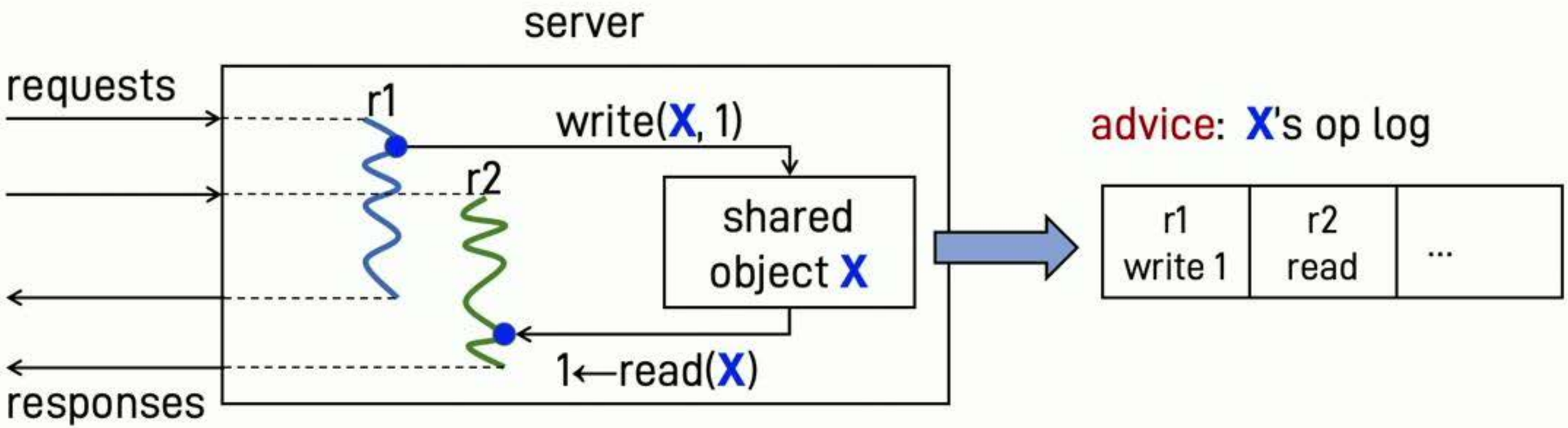
Concurrency model



Concurrency model



Concurrency model

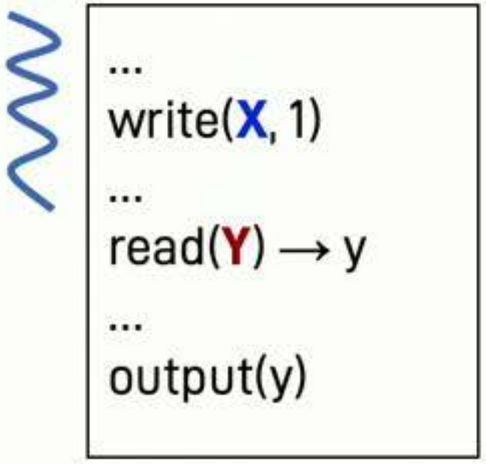


- What about simple re-execution according to op logs?

X=0, Y=0

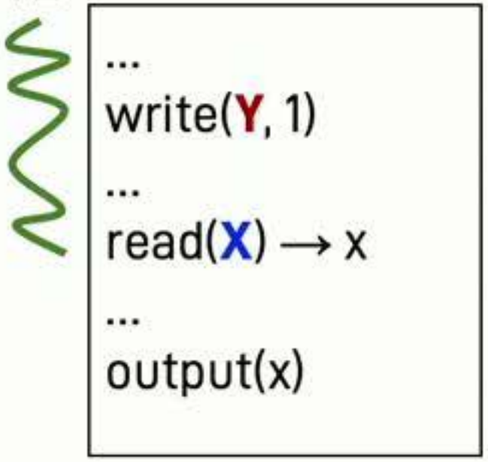
trace

r1



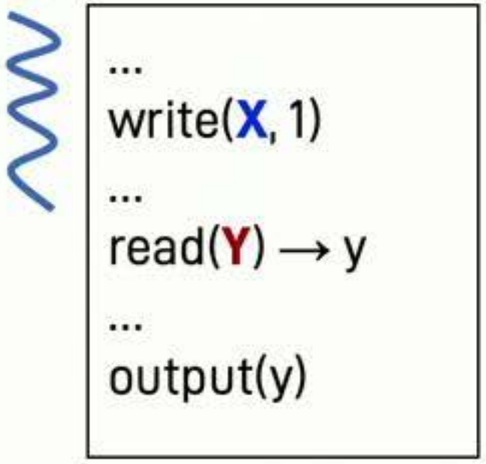
server

r2

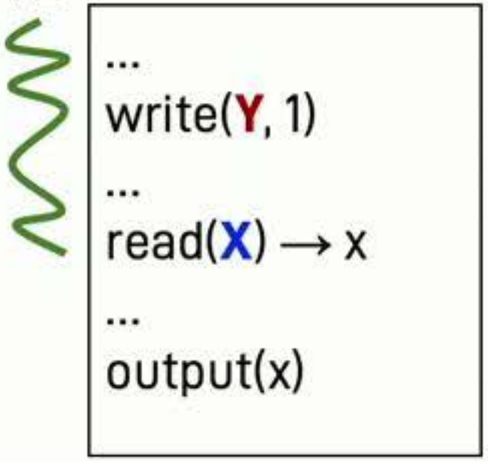


X=0, Y=0

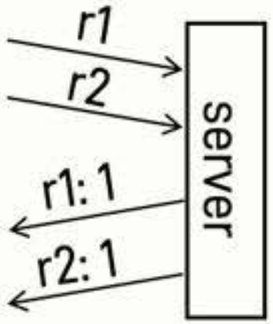
r1



r2

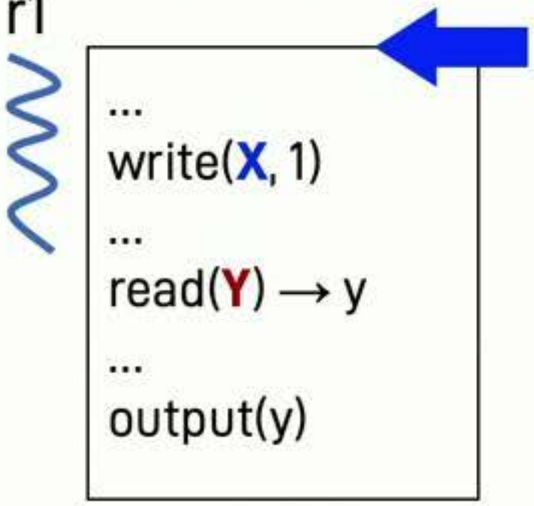


trace

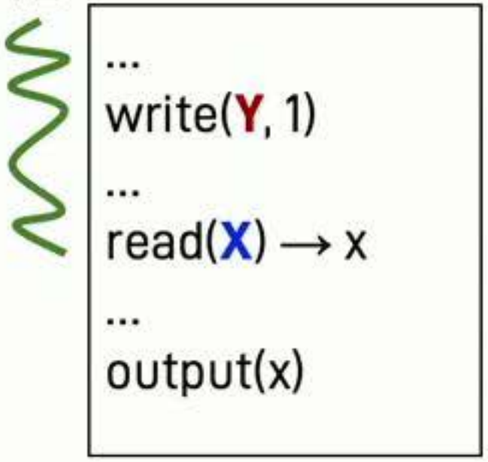


X=0, Y=0

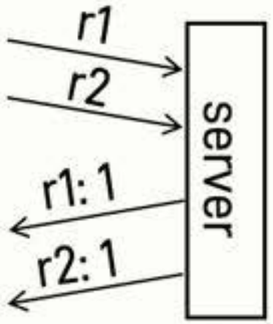
r1



r2



trace



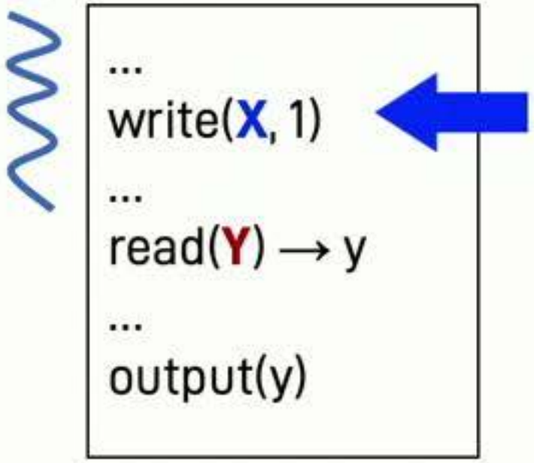
X = 0, Y = 0

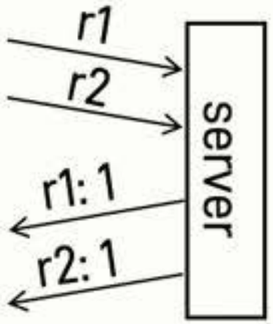
Accept

X=0, Y=0

trace

r1

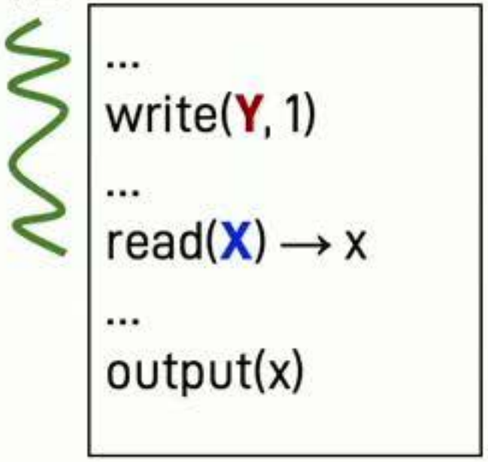




X = 1 , Y = 0

Accept

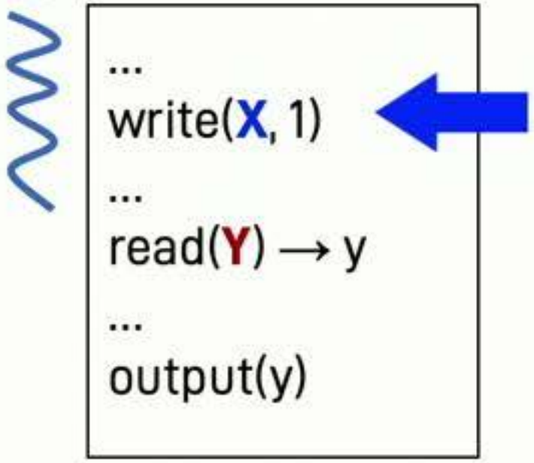
r2

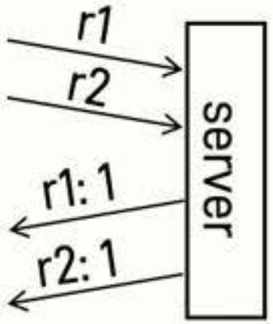


X=0, Y=0

trace

r1

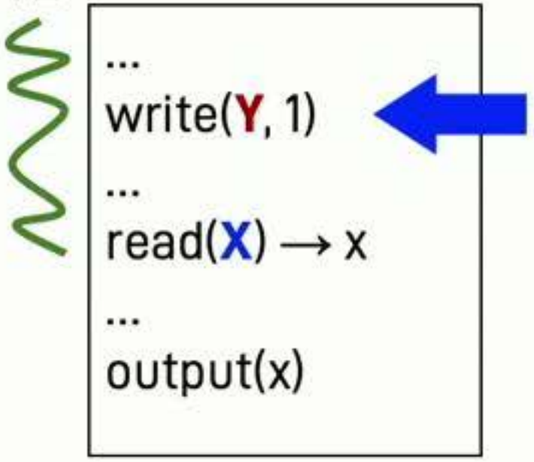


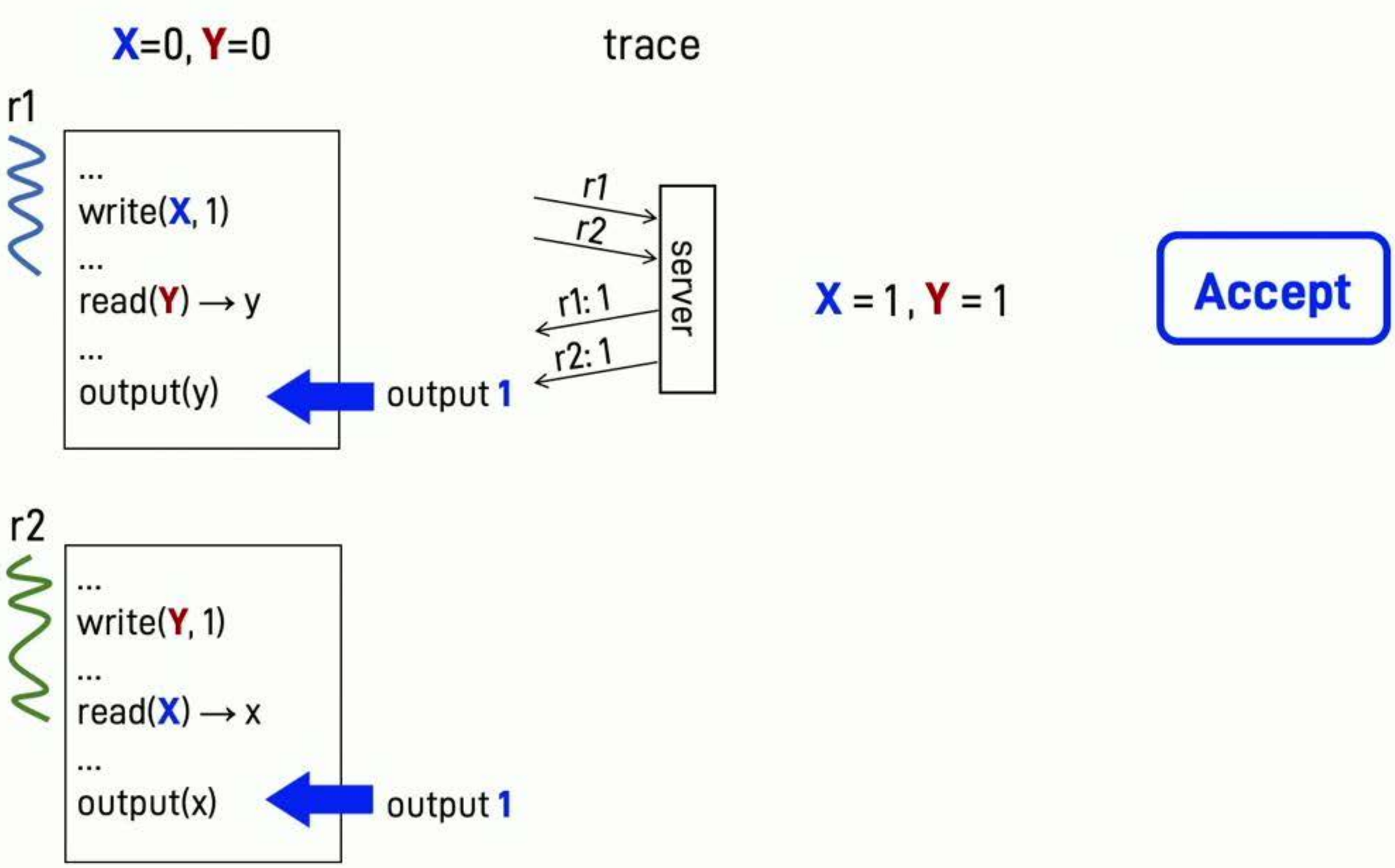


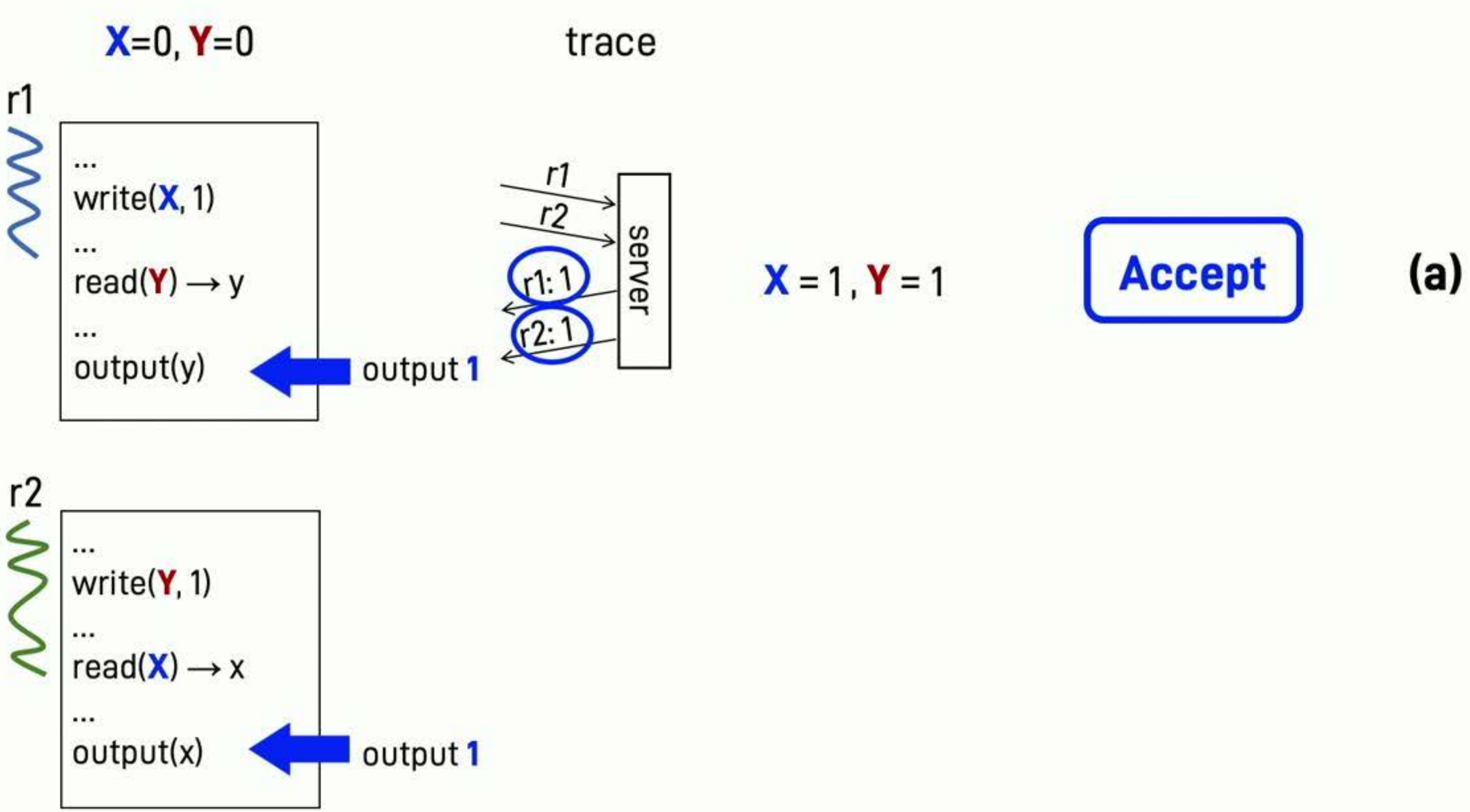
X = 1 , Y = 1

Accept

r2




X=0, Y=0
trace
r1
...
write(X, 1)
...
read(Y) → y
...
output(y)
output 1
r1
r2
r1: 1
r2: 1
server
X = 1 , Y = 1
Accept
r2
...
write(Y, 1)
...
read(X) → x
...
output(x)
output 1

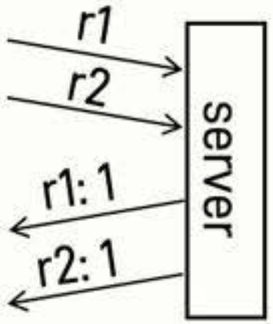

X=0, Y=0
trace
r1
...
write(X, 1)
...
read(Y) → y
...
output(y)
output 1
r2
...
write(Y, 1)
...
read(X) → x
...
output(x)
output 1
r1
r2
r1: 1
r2: 1
server
X = 1 , Y = 1
Accept
(a)

X=0, Y=0

trace

r1

```
...
write(X, 1)
...
read(Y) → y
...
output(y)
```



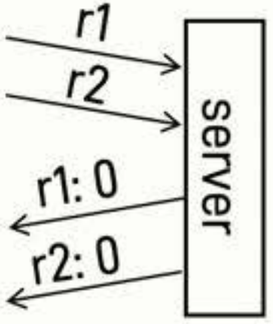
X = 1 , Y = 1

Accept

(a)

r2

```
...
write(Y, 1)
...
read(X) → x
...
output(x)
```



?

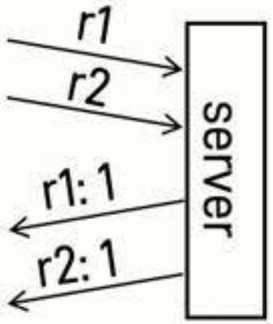
(b)

X=0, Y=0

trace

r1

```
...
write(X, 1)
...
read(Y) → y
...
output(y)
```



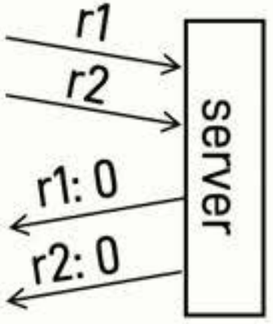
X = 1 , Y = 1

Accept

(a)

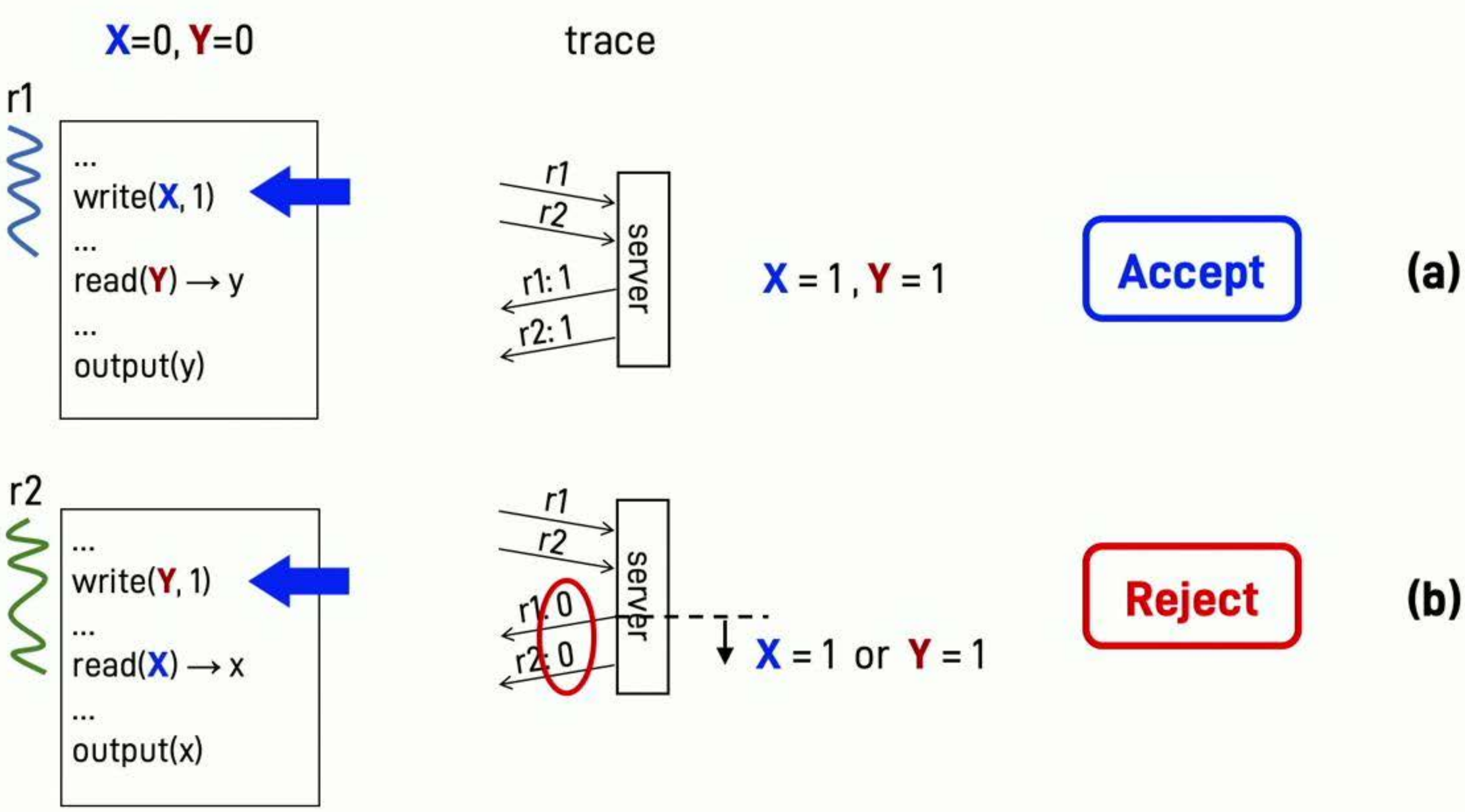
r2

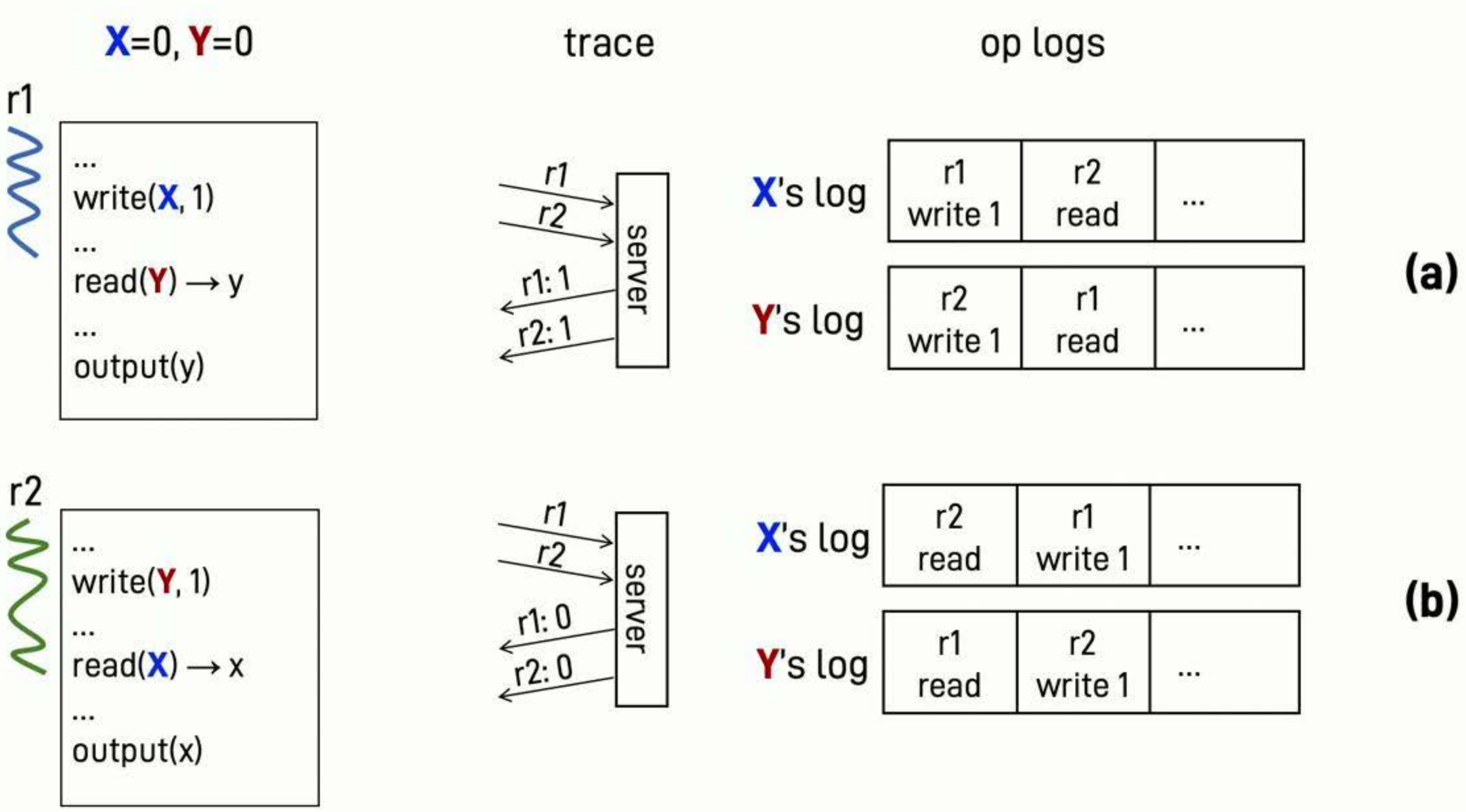
```
...
write(Y, 1)
...
read(X) → x
...
output(x)
```



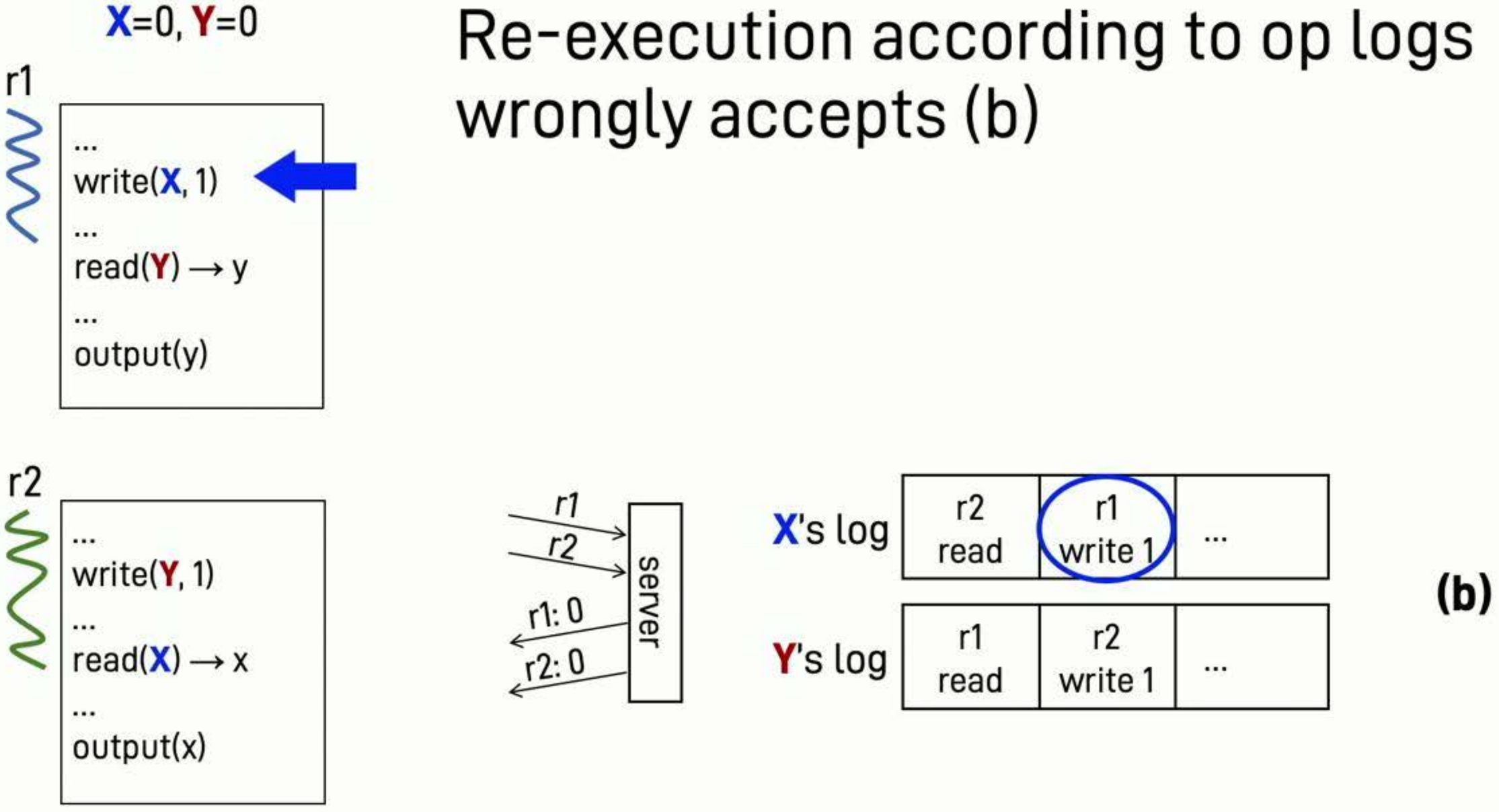
Reject

(b)

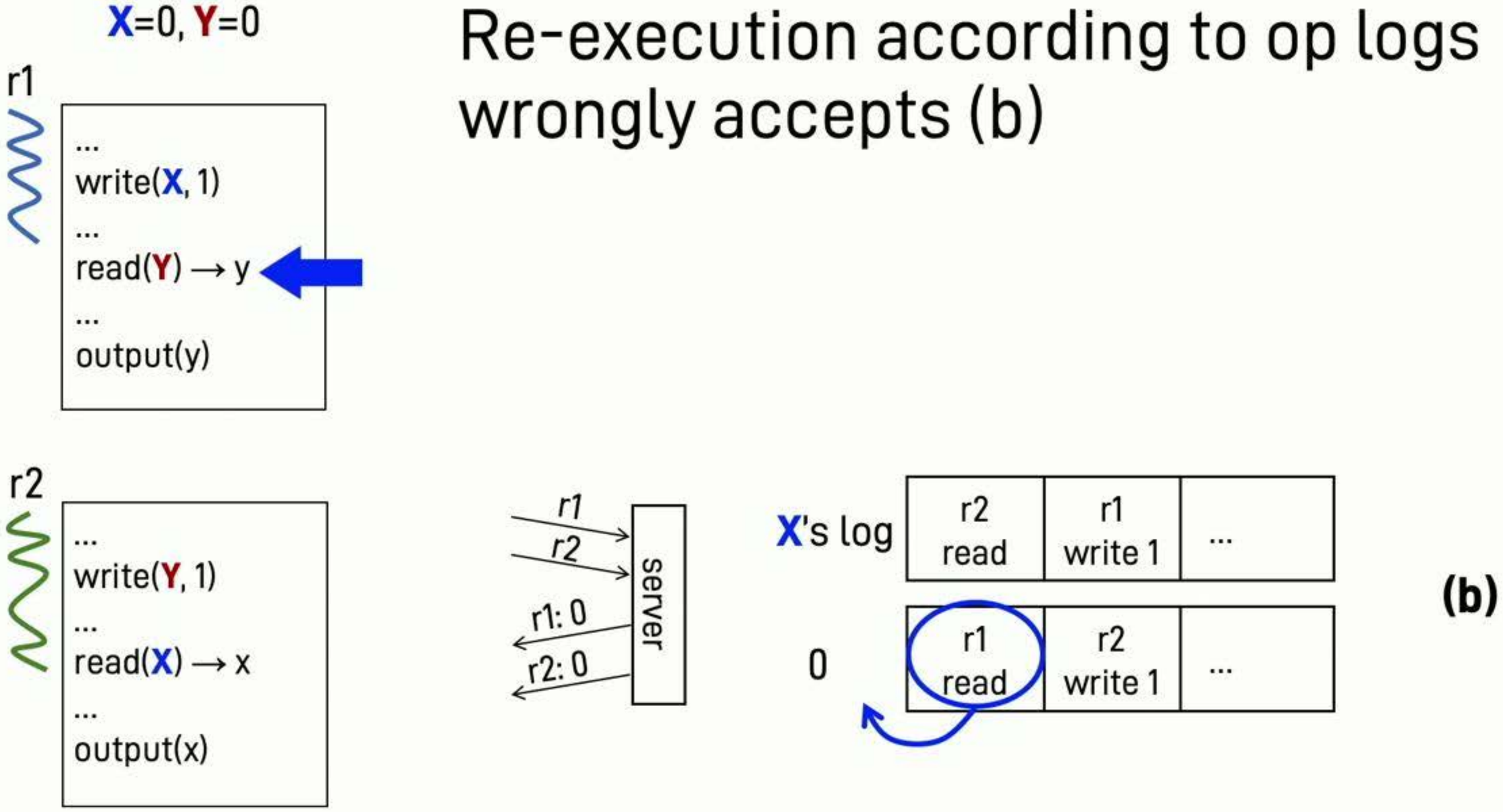

X=0, Y=0
trace
r1
...
write(X, 1)
...
read(Y) → y
...
output(y)
r1
r2
r1: 1
r2: 1
server
X = 1 , Y = 1
Accept
(a)
r2
...
write(Y, 1)
...
read(X) → x
...
output(x)
r1
r2
r1: 0
r2: 0
server
X = 1 or Y = 1
Reject
(b)


X=0, Y=0
trace
op logs
r1
...
write(X, 1)
...
read(Y) → y
...
output(y)
r1
r2
r1: 1
r2: 1
server
X's log
r1
write 1
r2
read
...
Y's log
r2
write 1
r1
read
...
(a)
r2
...
write(Y, 1)
...
read(X) → x
...
output(x)
r1
r2
r1: 0
r2: 0
server
X's log
r2
read
r1
write 1
...
Y's log
r1
read
r2
write 1
...
(b)

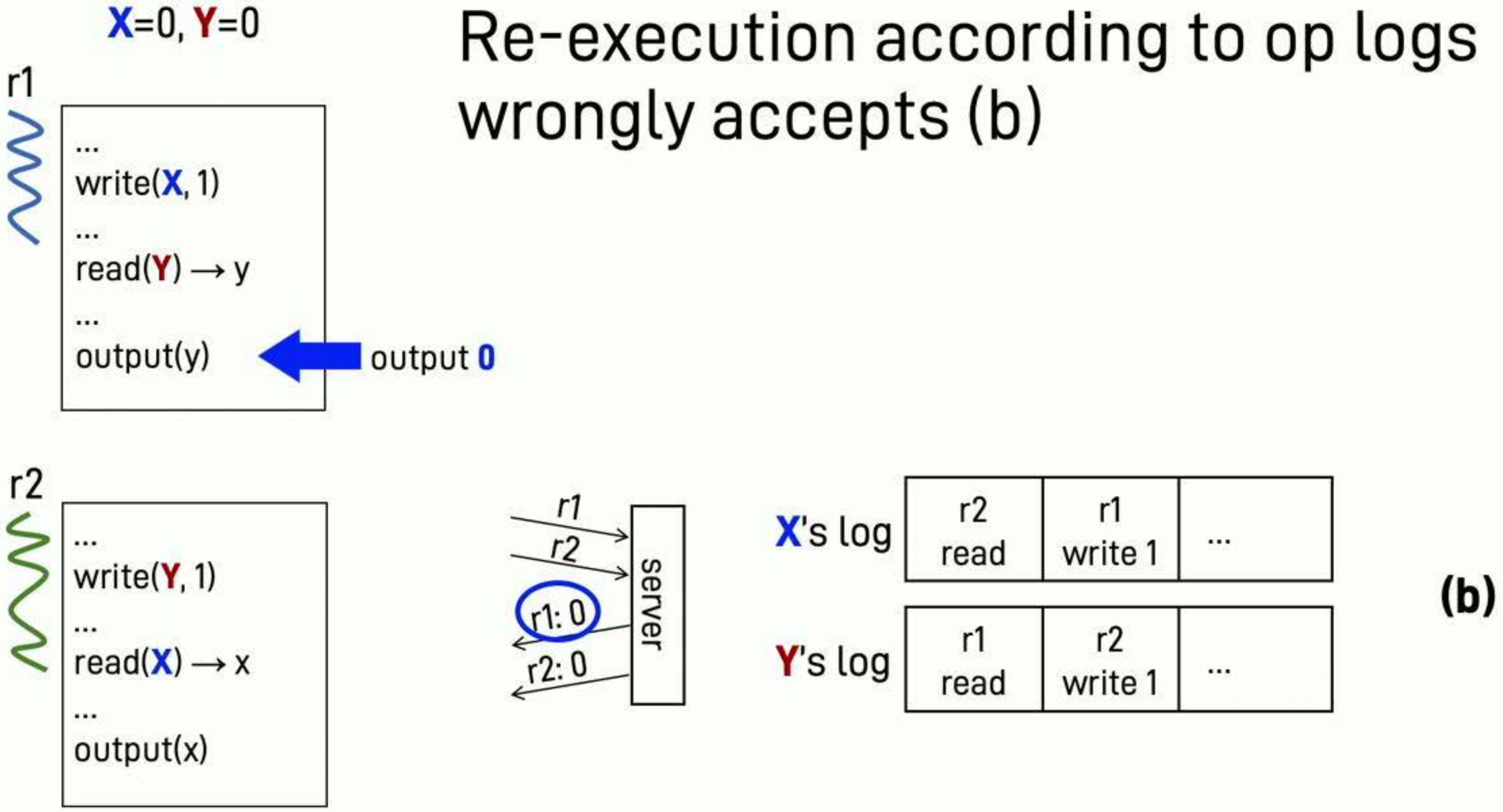
Re-execution according to op logs wrongly accepts (b)



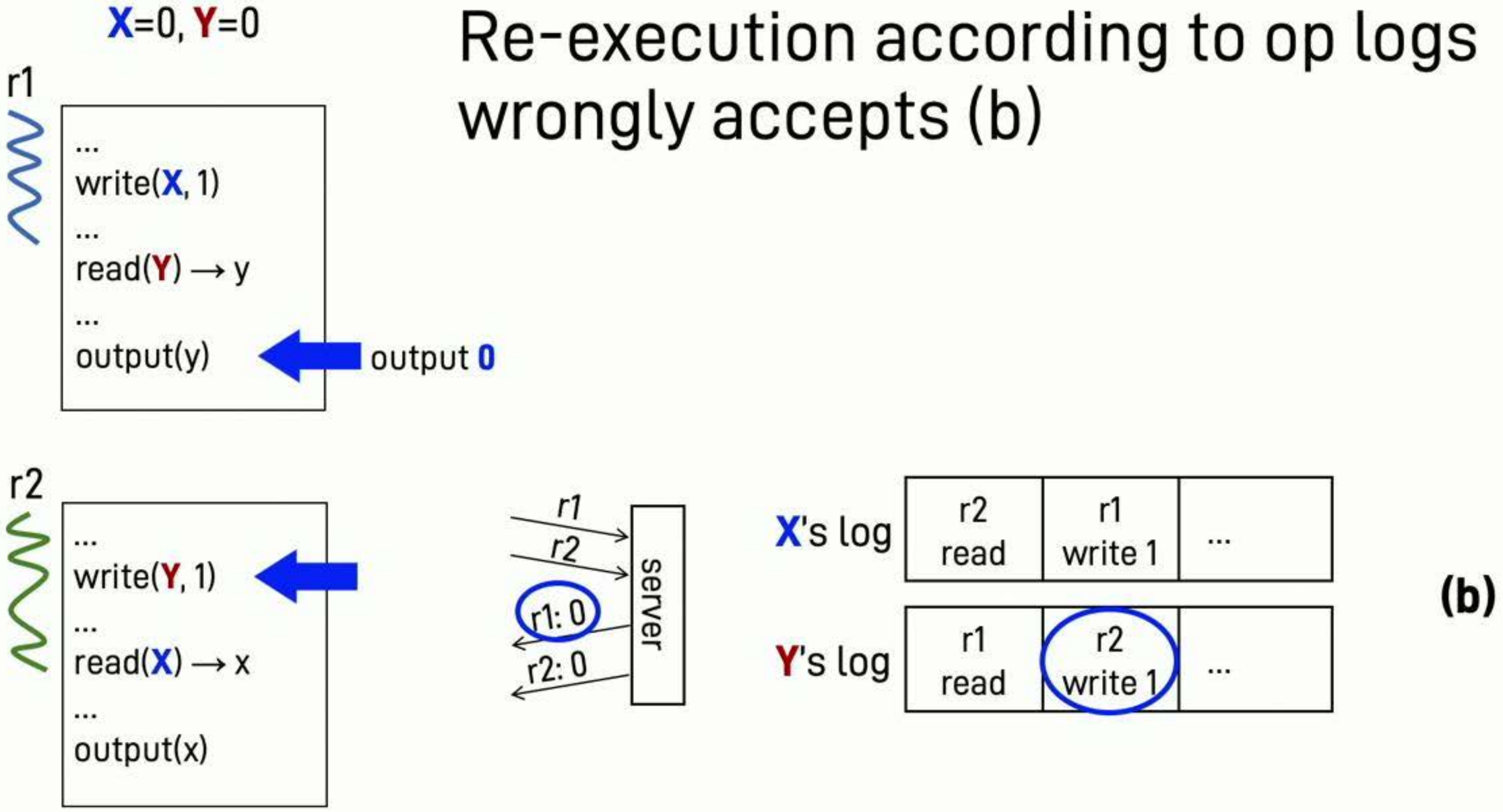
Re-execution according to op logs wrongly accepts (b)



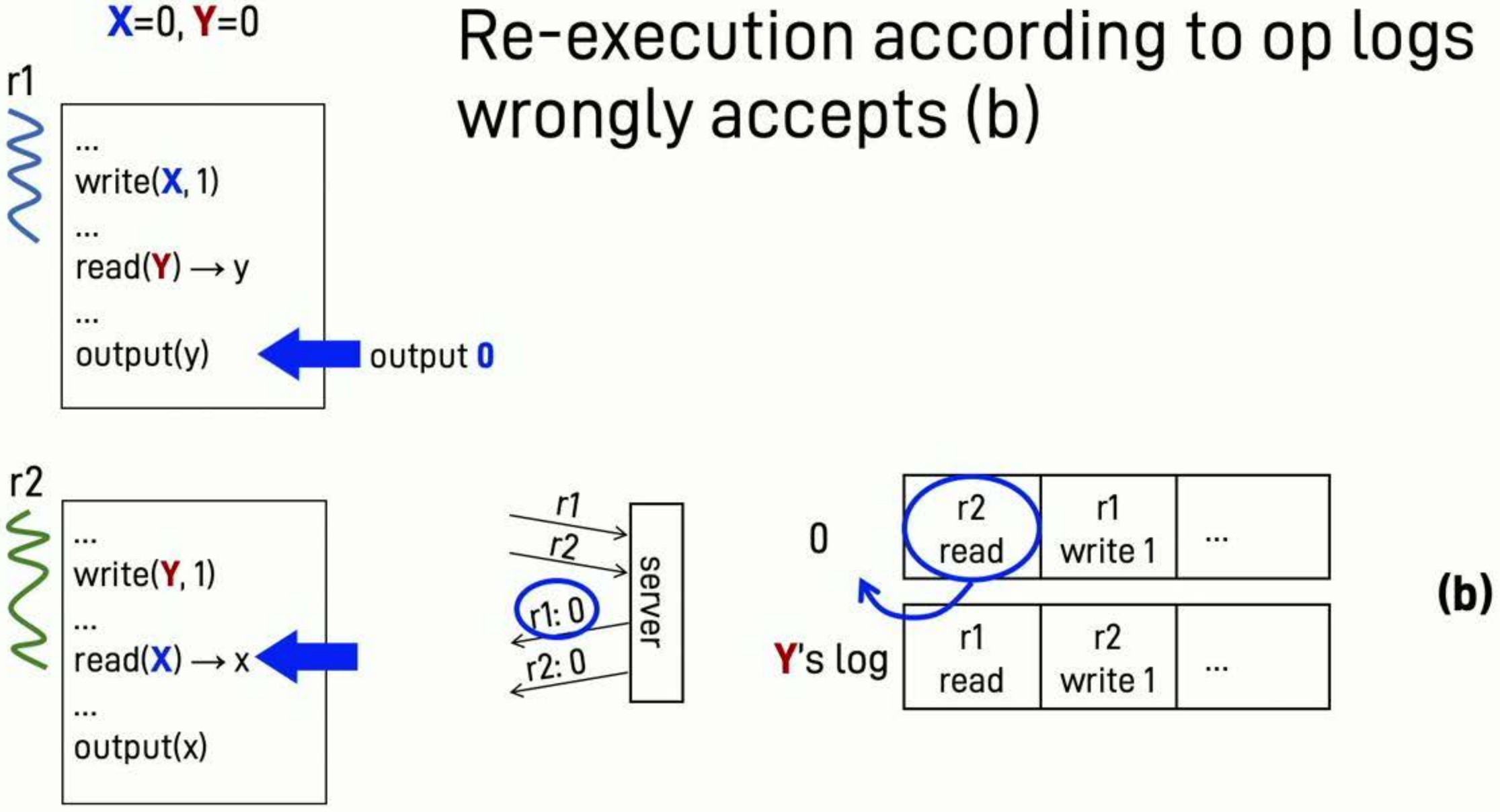
Re-execution according to op logs wrongly accepts (b)



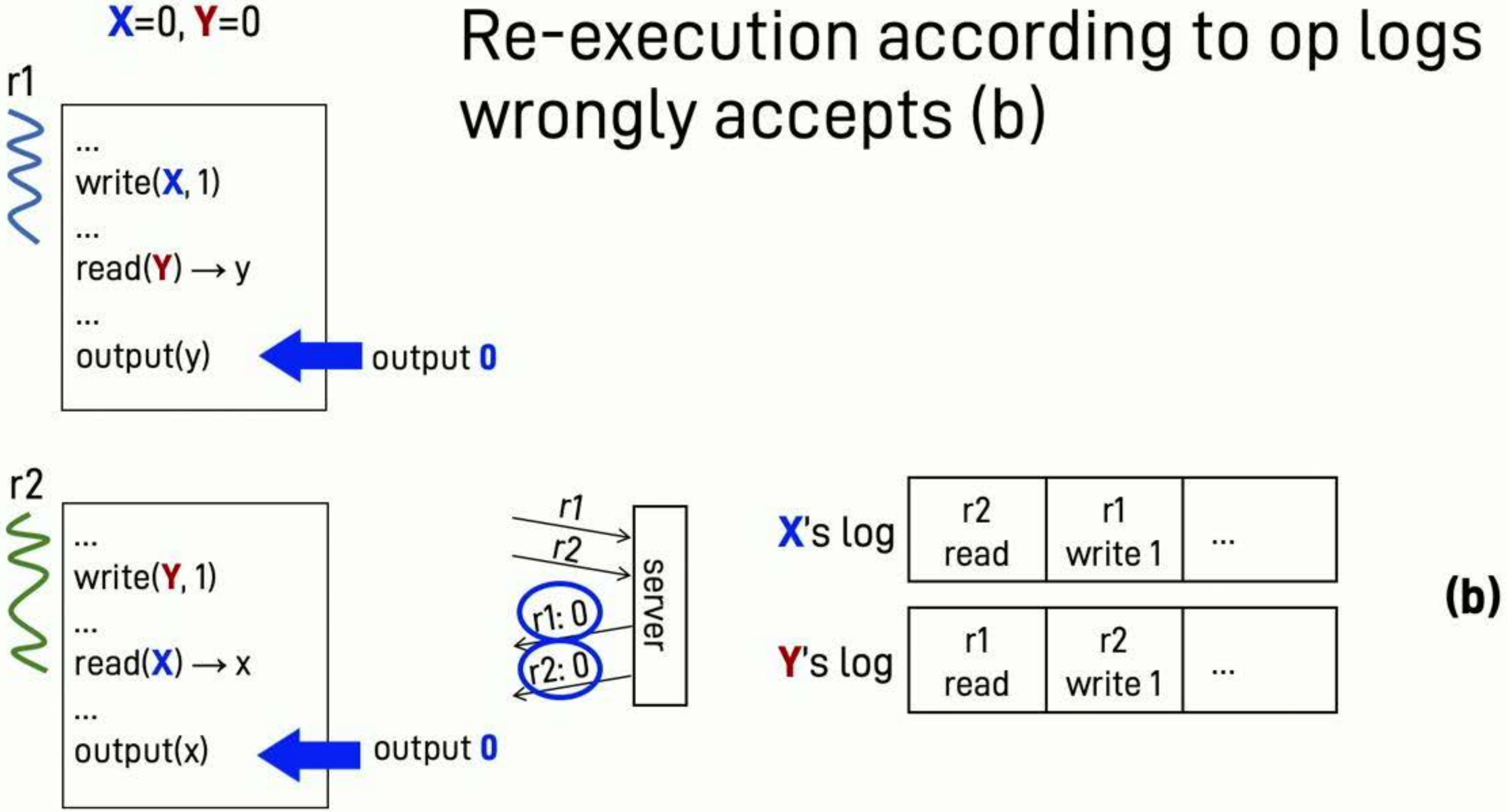
Re-execution according to op logs wrongly accepts (b)



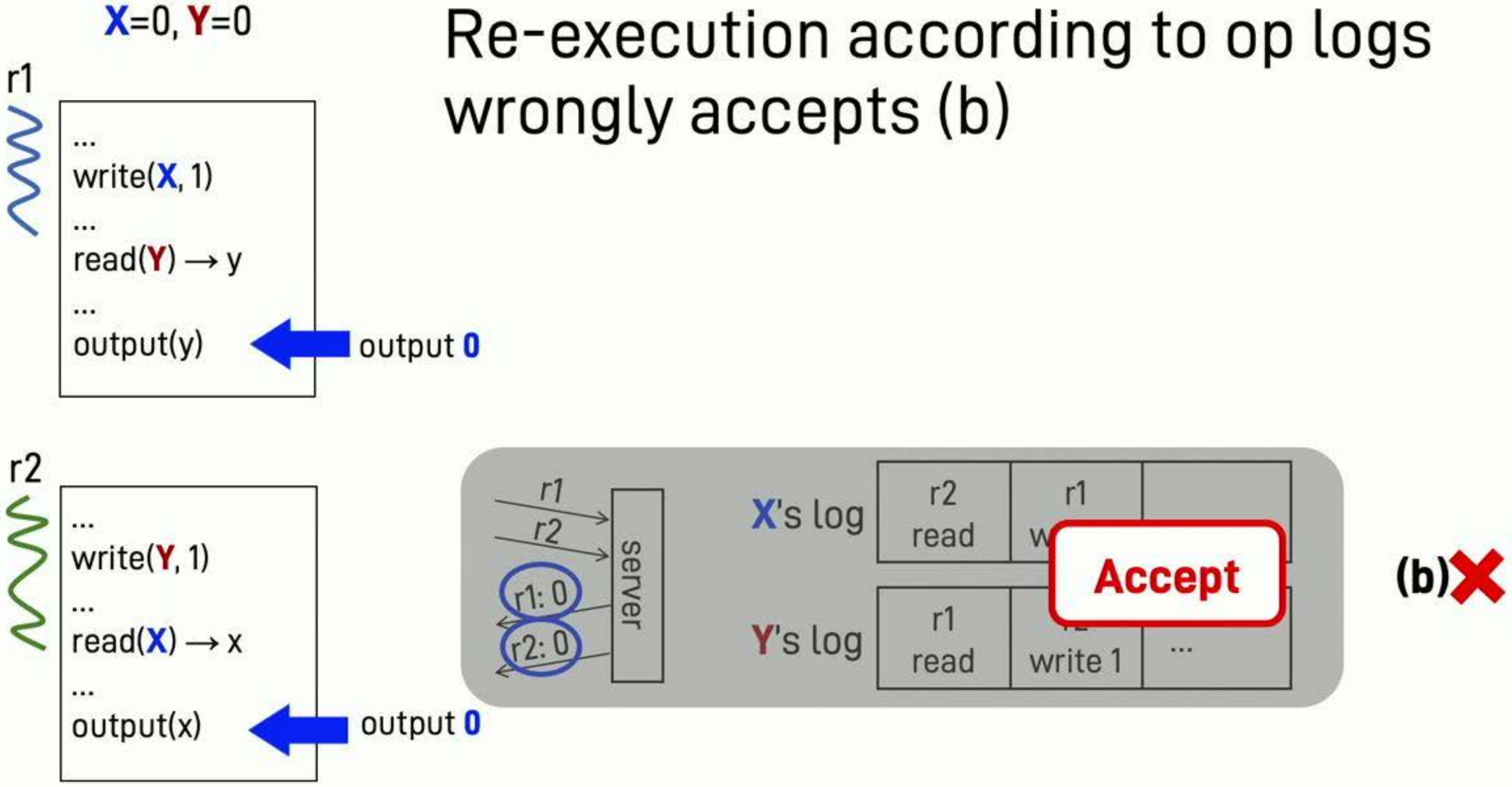
Re-execution according to op logs wrongly accepts (b)



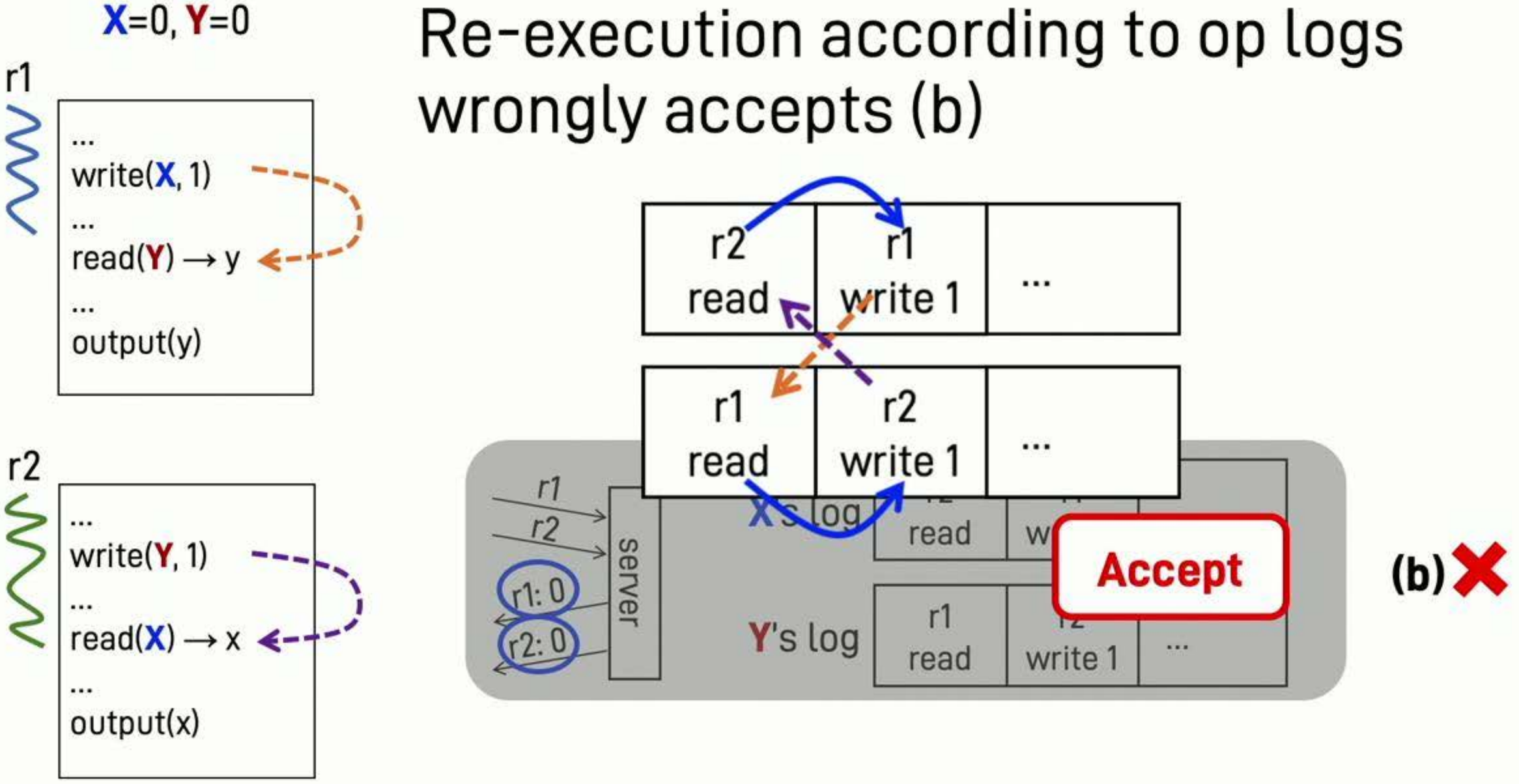
Re-execution according to op logs wrongly accepts (b)



Re-execution according to op logs wrongly accepts (b)



Re-execution according to op logs wrongly accepts (b)



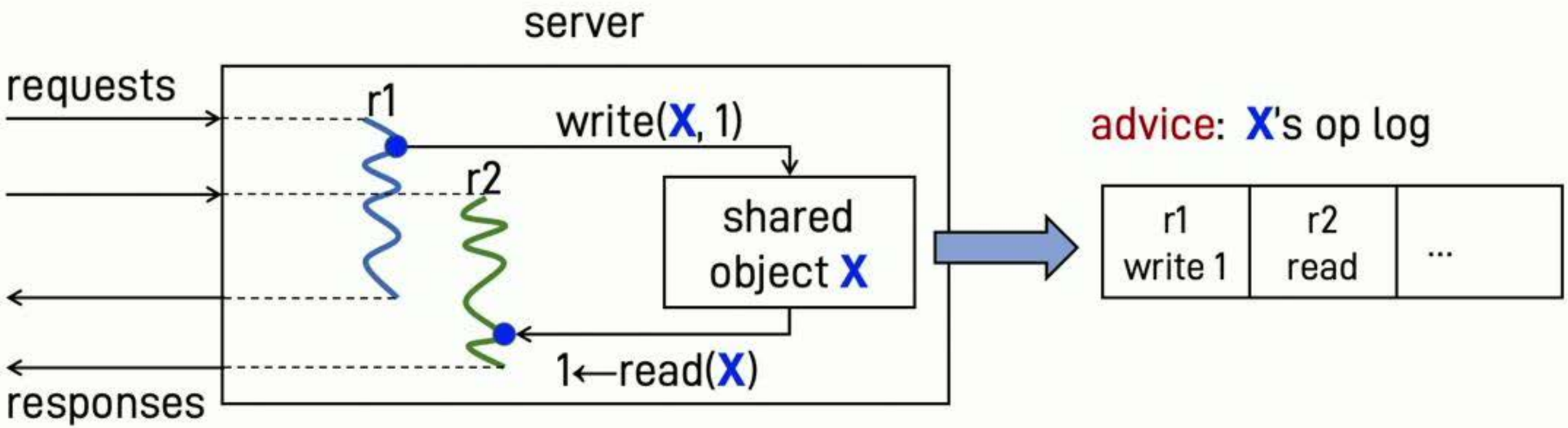
Solution must consider trace and program

- Validating the alleged op logs requires
  - request order (from trace)
  - program order (from program)
  - operation order (from op logs)

- Consistent ordering verification builds a graph…

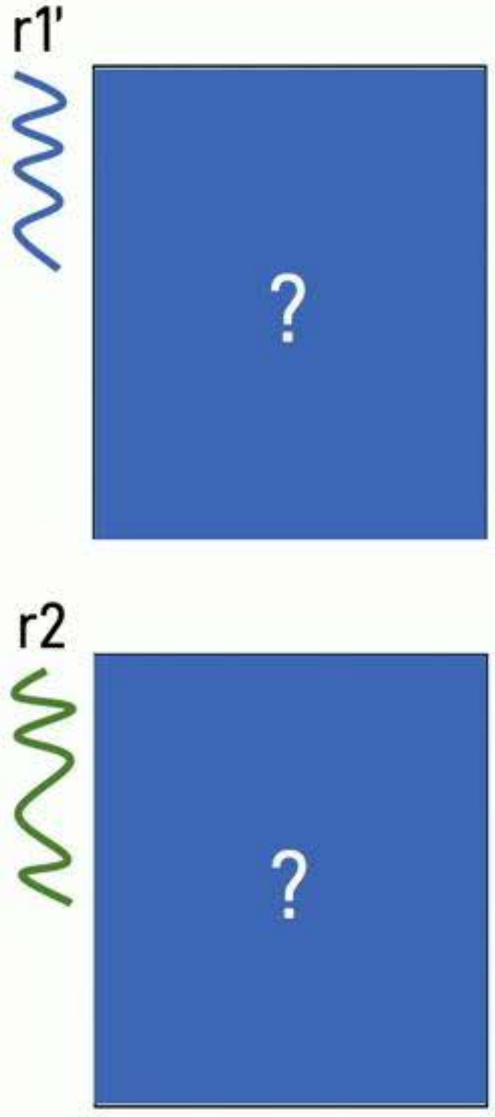
  …that includes all info above and check acyclicity

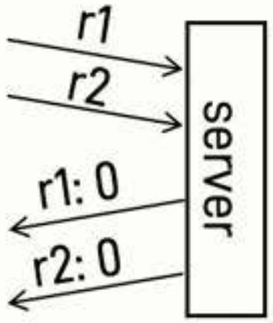
Concurrency model

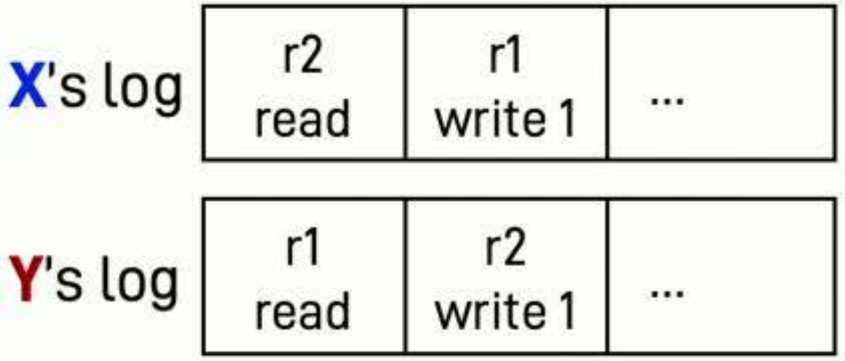


- What about simple re-execution according to op logs?

However, the actual problem is harder

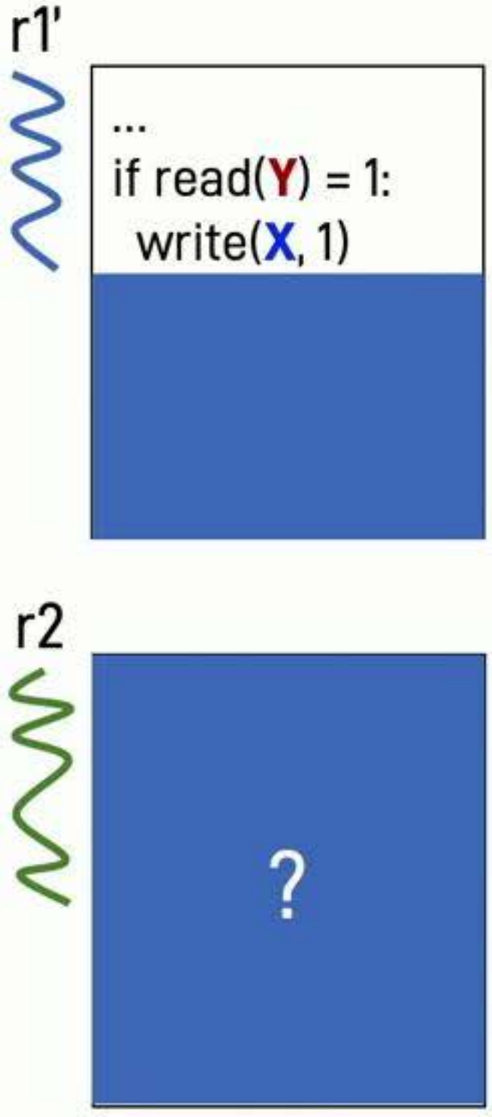


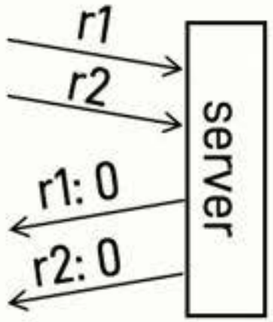


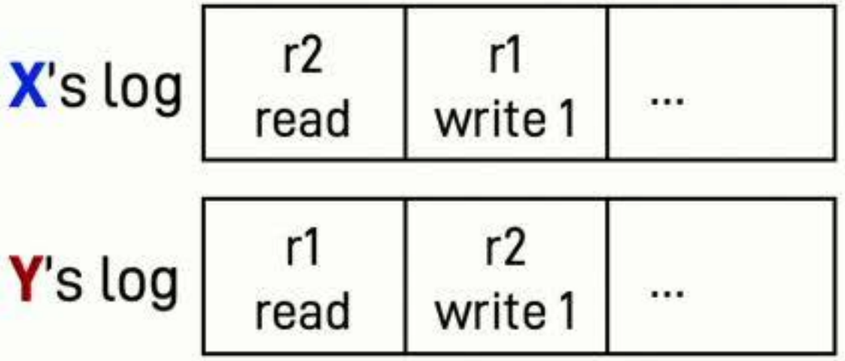


(b)

However, the actual problem is harder

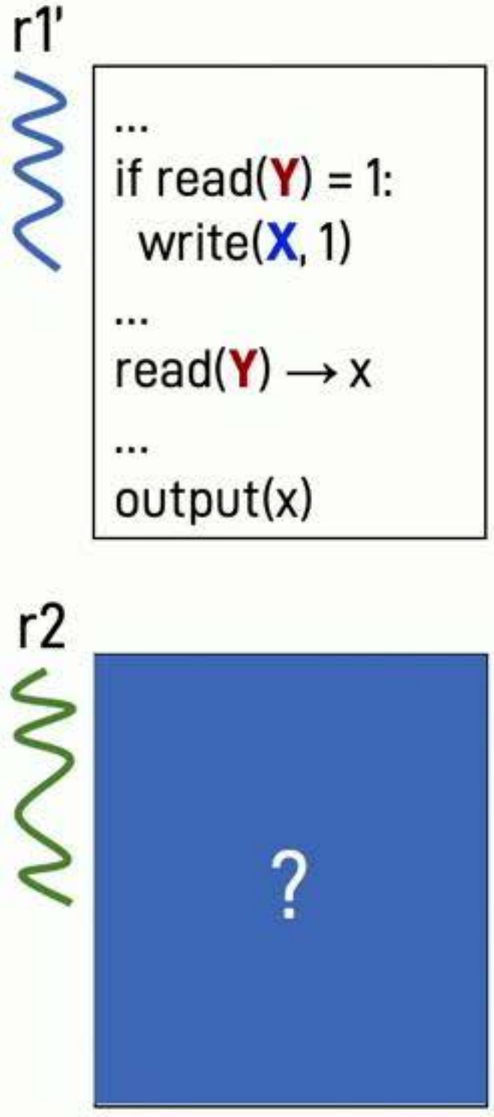


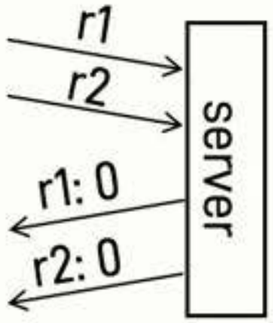


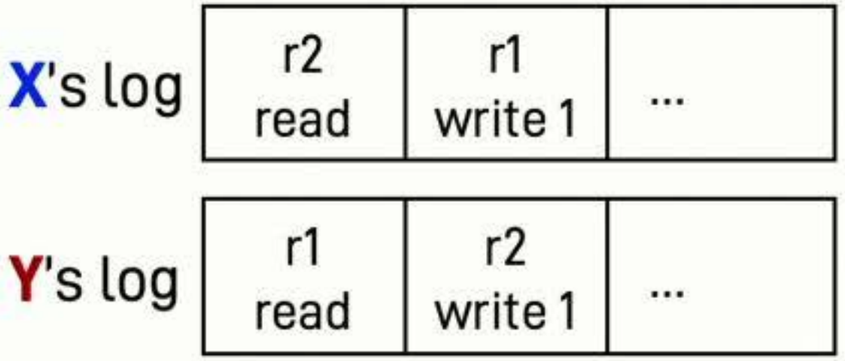


(b)

However, the actual problem is harder

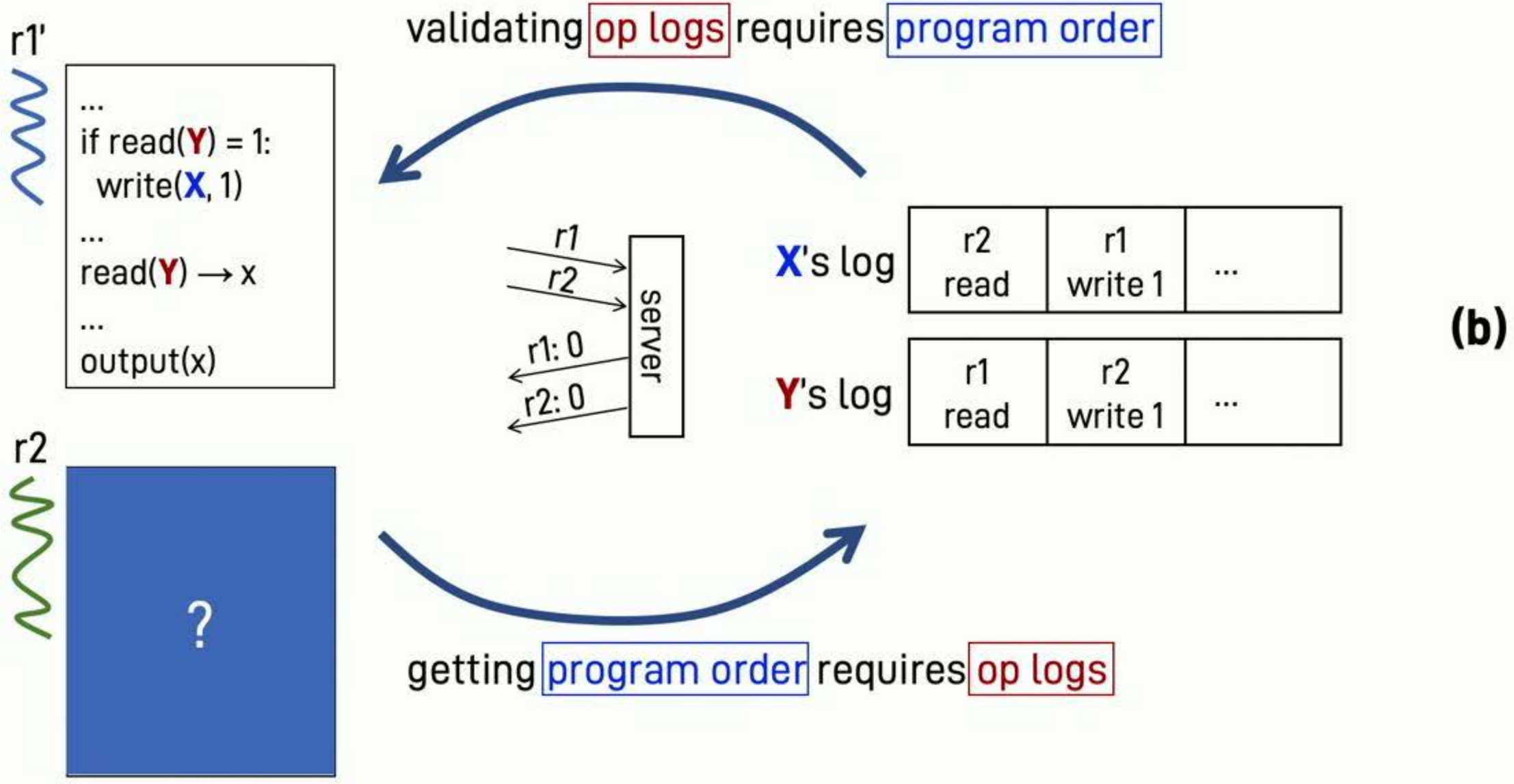






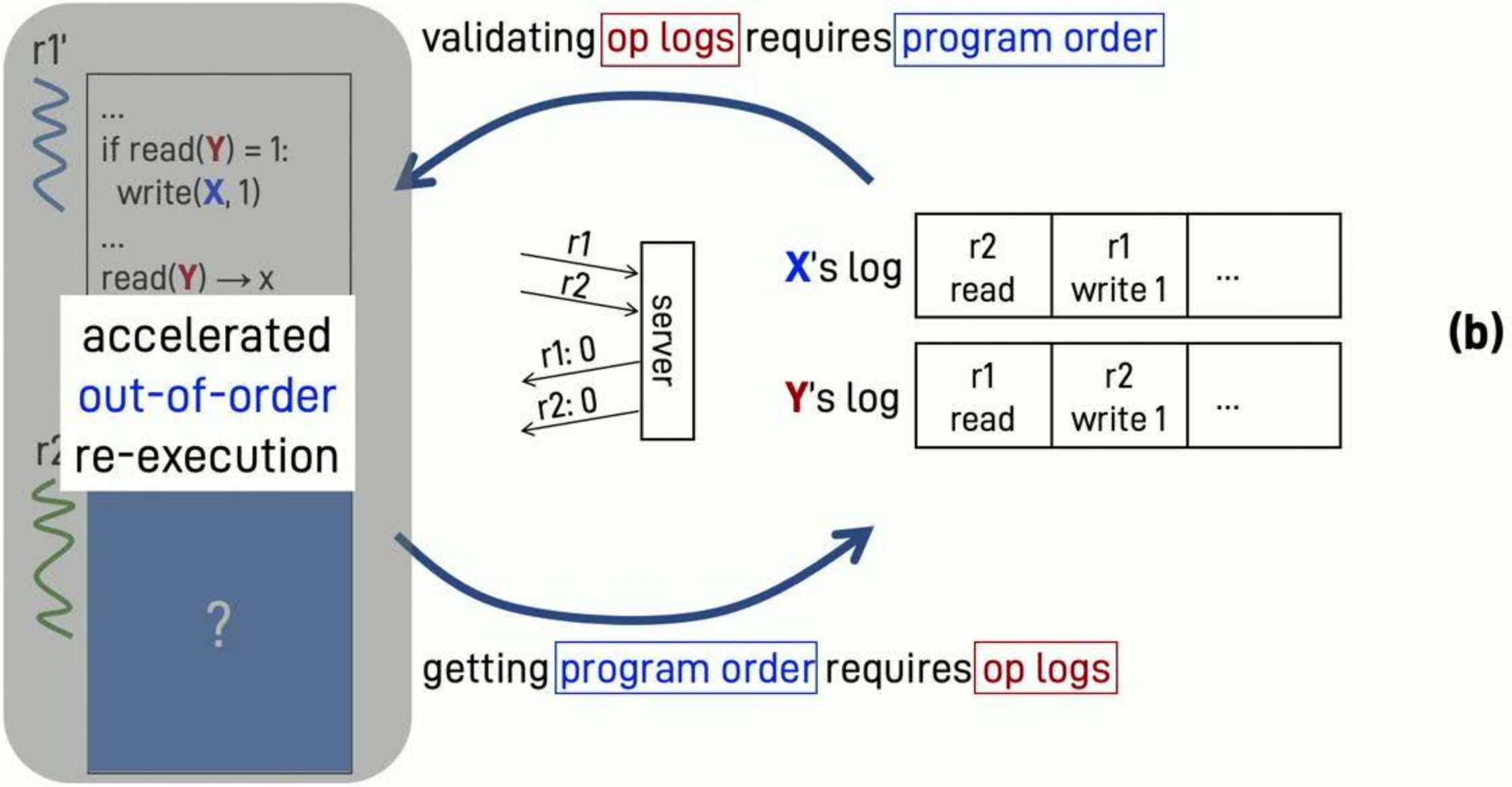
(b)

However, the actual problem is harder



(b)

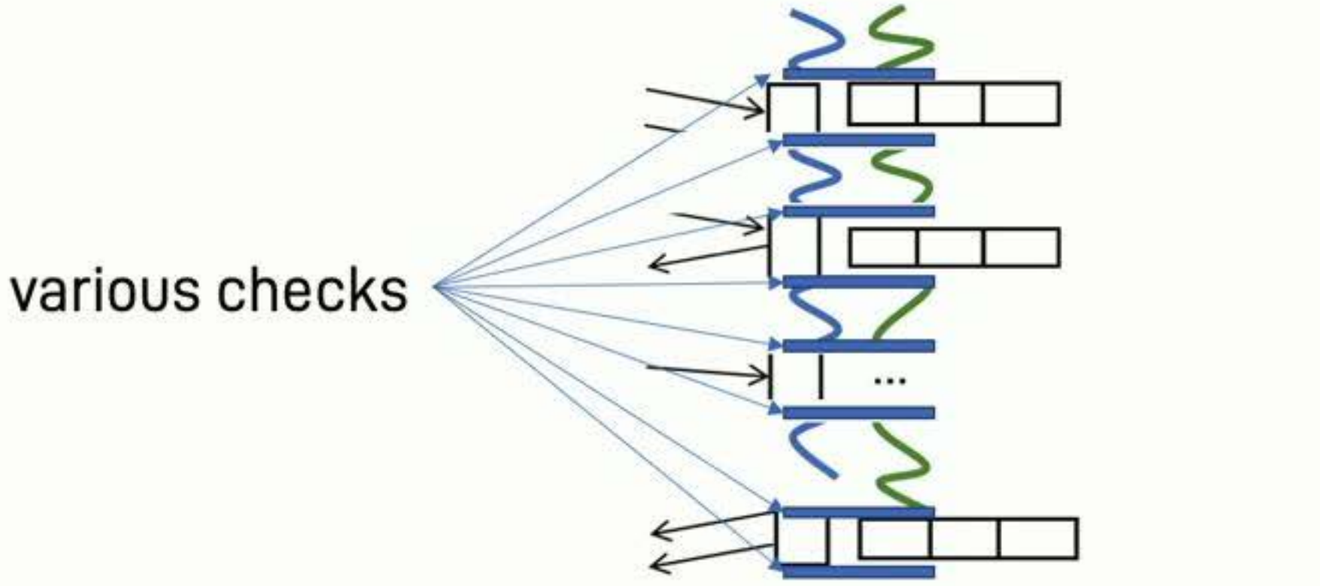
However, the actual problem is harder



(b)

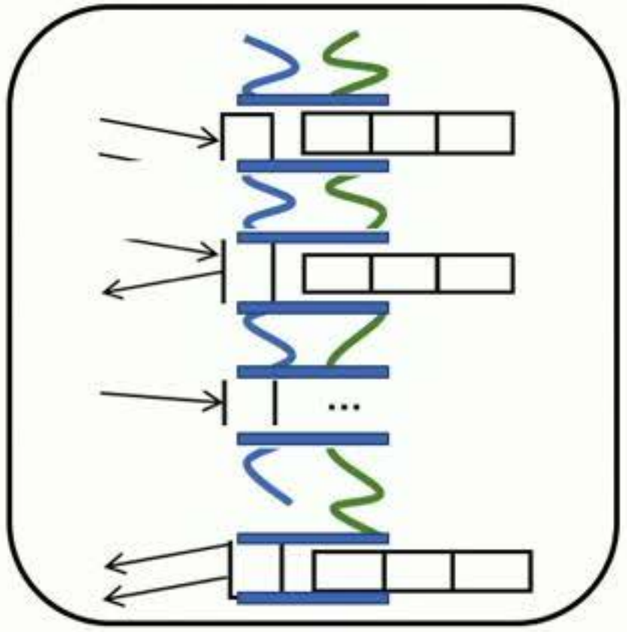
Orochi has a co-designed verification protocol

Orochi has a co-designed verification protocol



re-execution + consistent ordering

Orochi has a co-designed verification protocol



- Verification protocol is proved to be correct ...

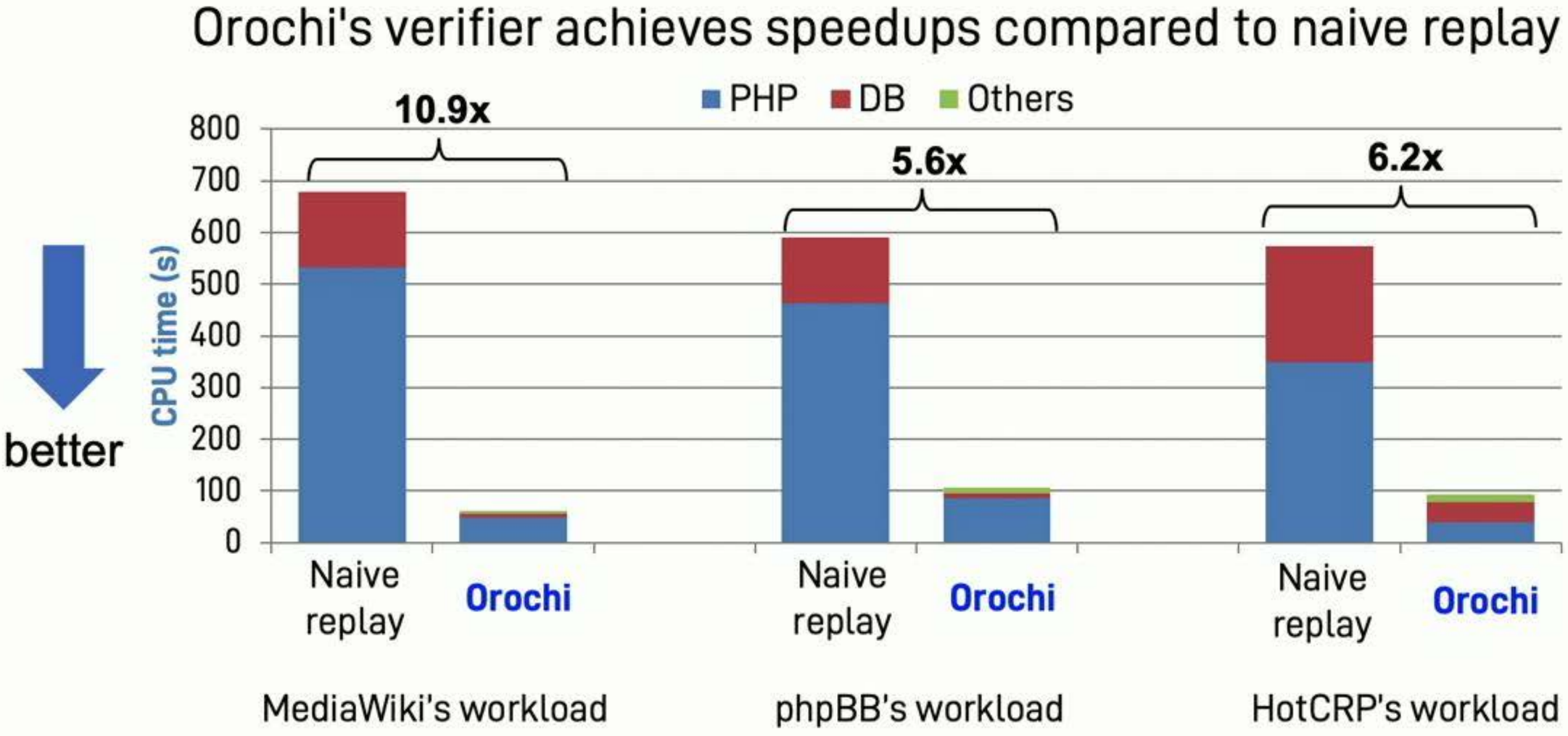
... meaning {
- Completeness: honest server ⇒ verifier accepts
- Soundness: verifier accepts ⇒ honest server

Evaluation setup

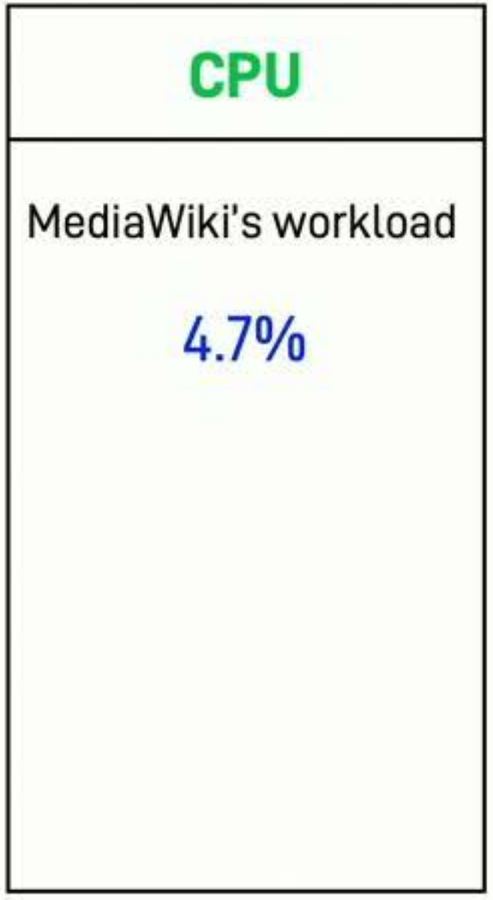
- Applications:
  - MediaWiki, phpBB and HotCRP

- Workloads:
  - MediaWiki: Wikipedia 2007 trace
  - phpBB: 7-day's posts from CentOS forum
  - HotCRP: Simulation of SIGCOMM'09

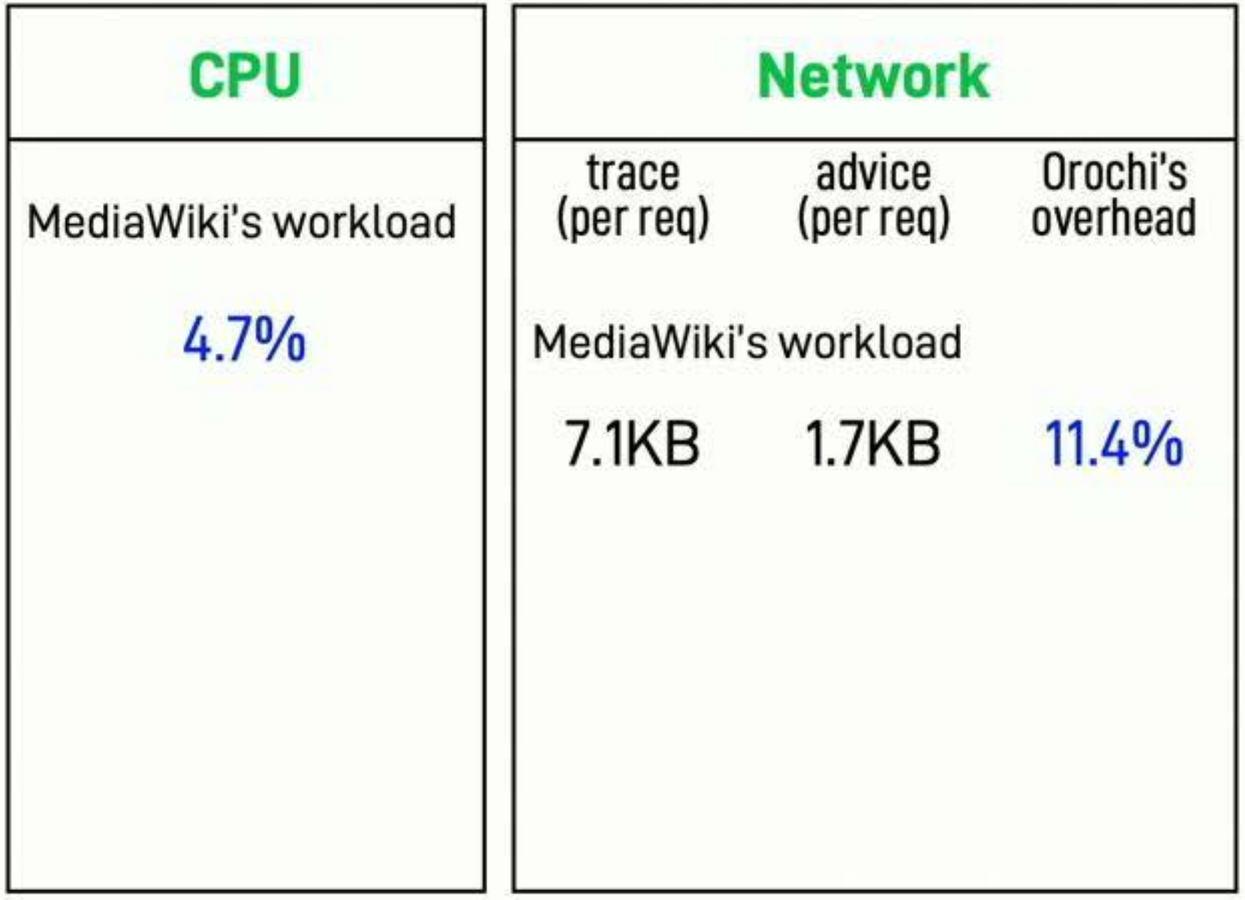
Is the verifier efficient?



What are the (server's) CPU/network/storage costs?



What are the (server's) CPU/network/storage costs?

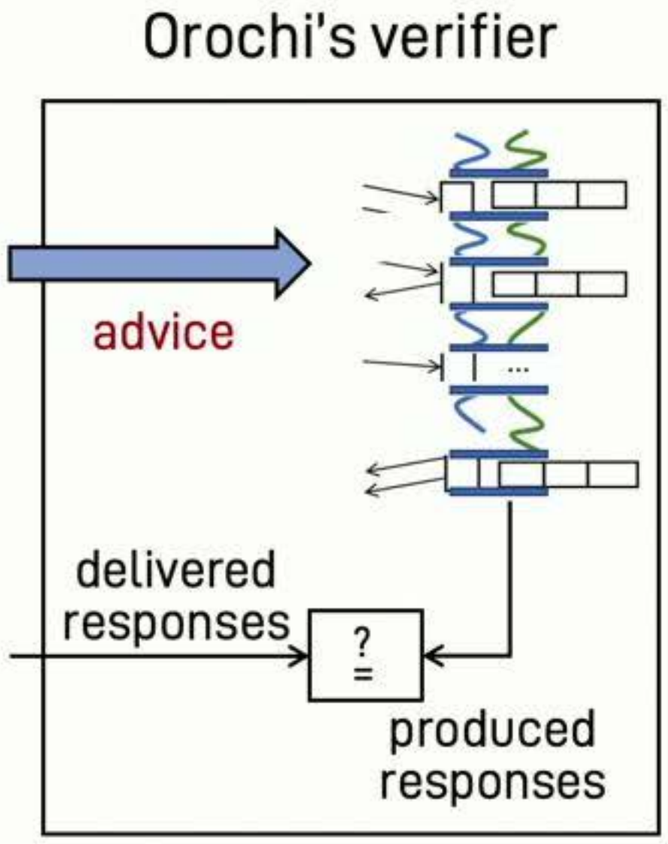


What are the (server's) CPU/network/storage costs?

CPU	Network			Storage
	trace (per req)	advice (per req)	Orochi's overhead	
MediaWiki's workload	MediaWiki's workload			MediaWiki's workload
4.7%	7.1KB	1.7KB	11.4%	1.0x

What are the (server's) CPU/network/storage costs?

CPU	Network			Storage
	trace (per req)	advice (per req)	Orochi's overhead	
MediaWiki's workload	MediaWiki's workload			MediaWiki's workload
4.7%	7.1KB	1.7KB	11.4%	1.0x
phpBB's workload	phpBB's workload			phpBB's workload
8.6%	5.7KB	0.3KB	2.7%	1.7x
HotCRP's workload	HotCRP's workload			HotCRP's workload
5.9%	3.2KB	0.4KB	10.9%	1.5x

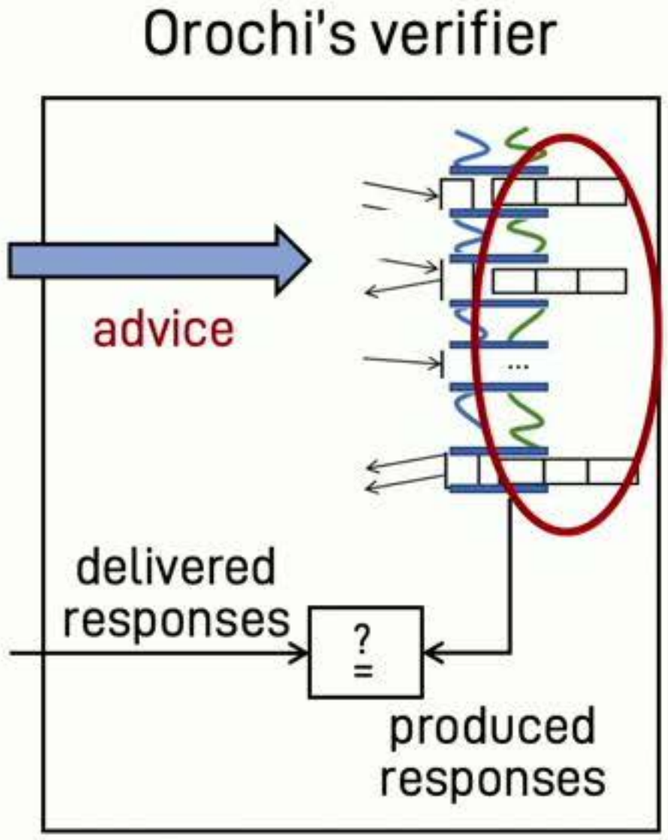


Problem: naive re-execution doesn't save work

Solution: deduplicated re-execution

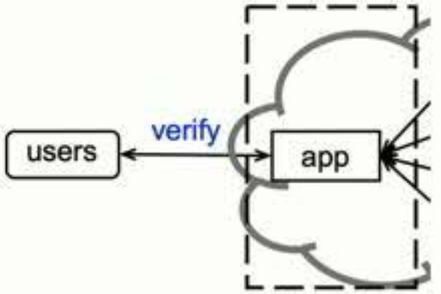
Problem: concurrent and untrusted server

Solution: co-design verification protocol

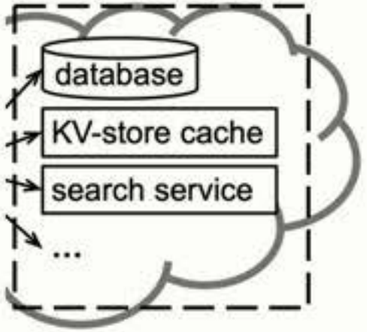


op logs may be unavailable (e.g., black-box databases)

① Verify white-box services (Orochi [SOSP'17])



② Verify black-box services (Cobra: verify Serializability)



③ Build composable verifiable framework (future work)

④ Other past work
- Troubleshooting data center networks [NSDI'19]
- Protecting secret via security-oriented offloading [EuroSys'15]

Why Serializability (SER)?

gold standard
isolation level


CockroachDB
(2015)

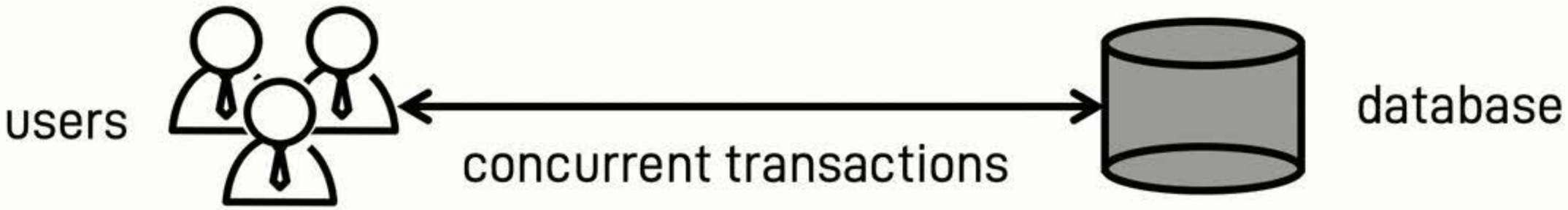

Amazon Aurora
(2015)

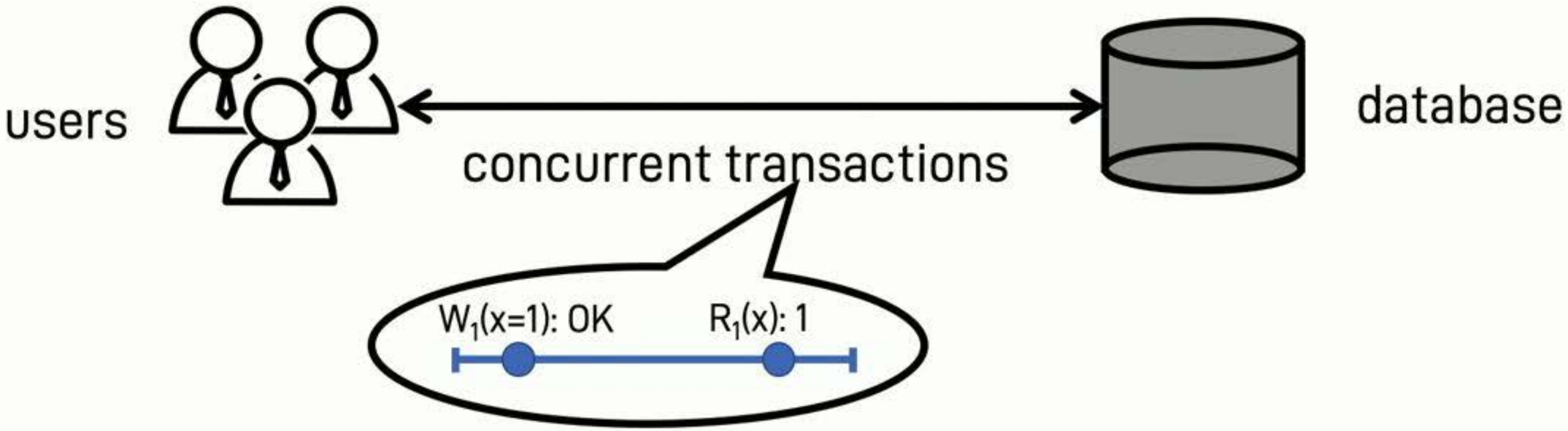

Google Spanner
(2017)

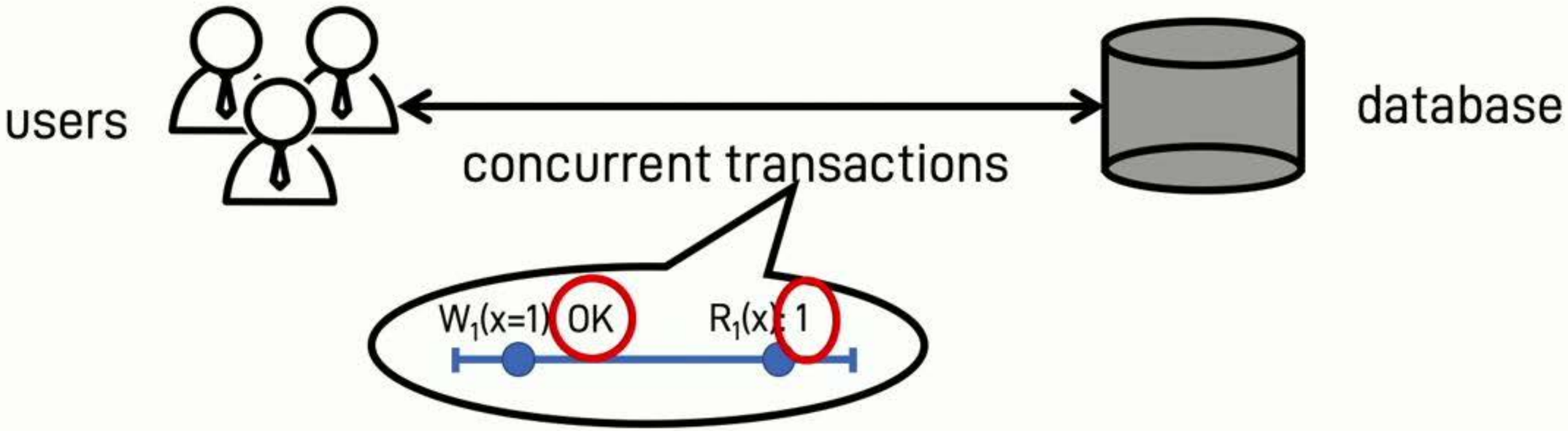

YugaByteDB
(2017)

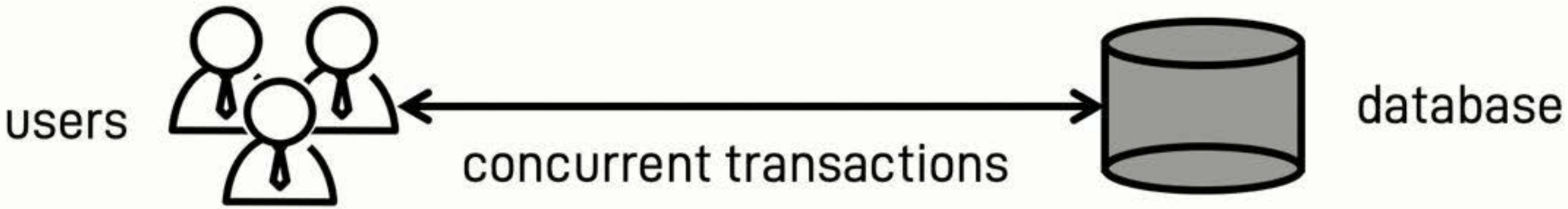
...

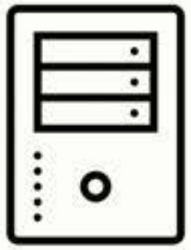
fundamental
and
challenging


users
concurrent transactions
database


users
database
concurrent transactions
W1(x=1): OK
R1(x): 1


users
database
concurrent transactions
W1(x=1) OK
R1(x) 1





Are concurrent transactions SER?

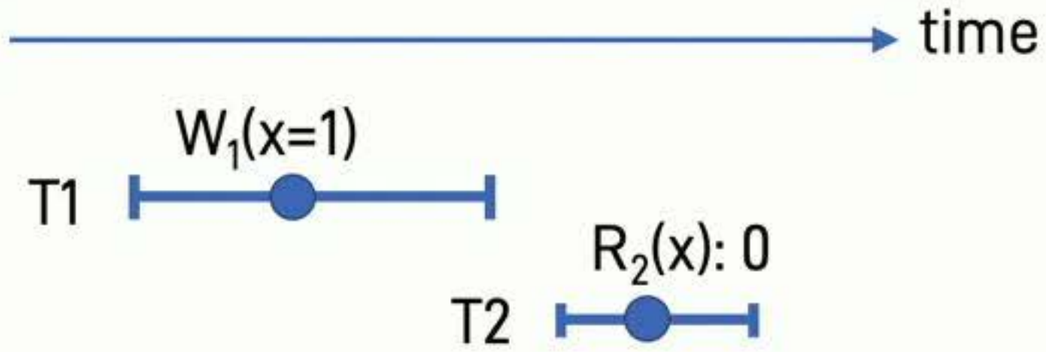
concurrent transactions $\overset{?}{=}$ sequential execution of these transactions

concurrent transactions $\overset{?}{=}$ sequential execution of these transactions

- Checking view-SER is an NP-complete problem [Papadimitriou 79]

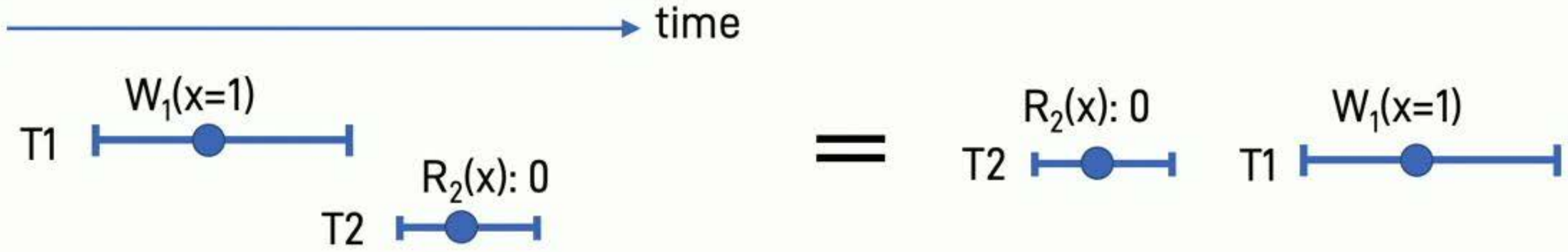
concurrent transactions $\overset{?}{=}$ sequential execution of these transactions

- Checking view-SER is an NP-complete problem [Papadimitriou 79]
- Challenge: SER doesn't respect real-time order



concurrent transactions $\overset{?}{=}$ sequential execution of these transactions

- Checking view-SER is an NP-complete problem [Papadimitriou 79]
- Challenge: SER doesn't respect real-time order



Cobra aims at real-world workloads

- Intuition: advances of SAT/SMT solvers

Cobra aims at real-world workloads

- Intuition: advances of SAT/SMT solvers
- Starting point (Papadimitriou's construction):

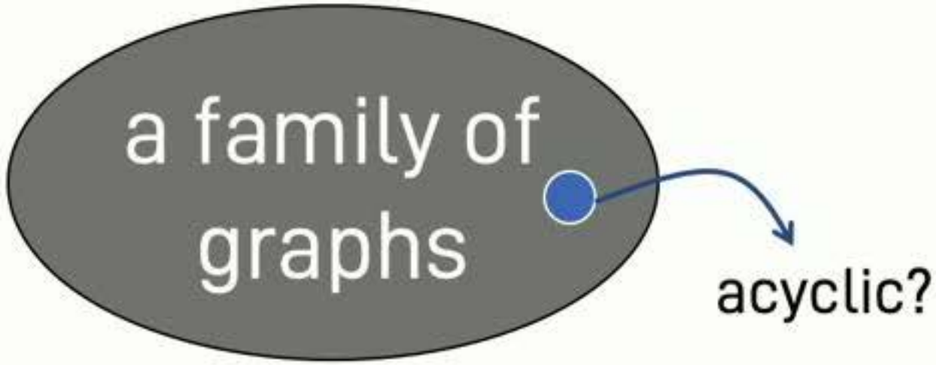
Cobra aims at real-world workloads

- Intuition: advances of SAT/SMT solvers
- Starting point (Papadimitriou's construction):

concurrent transactions

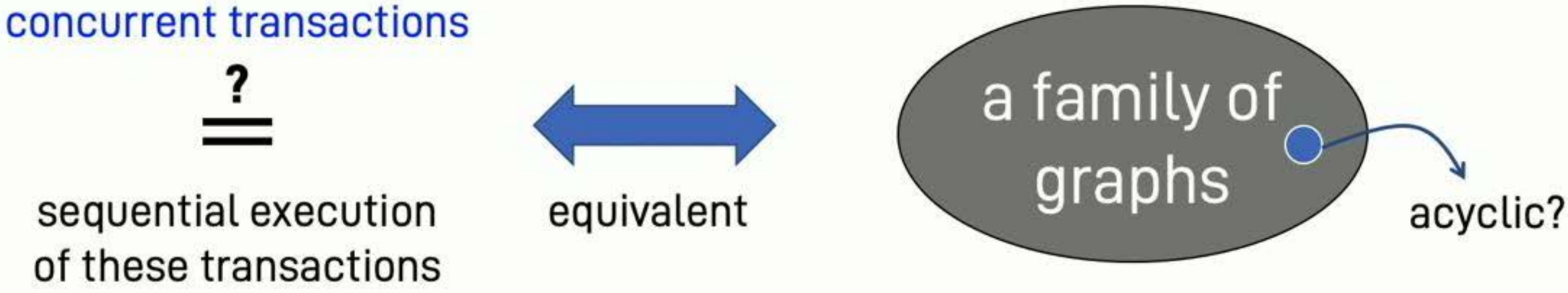
$\overset{?}{=}$

sequential execution
of these transactions



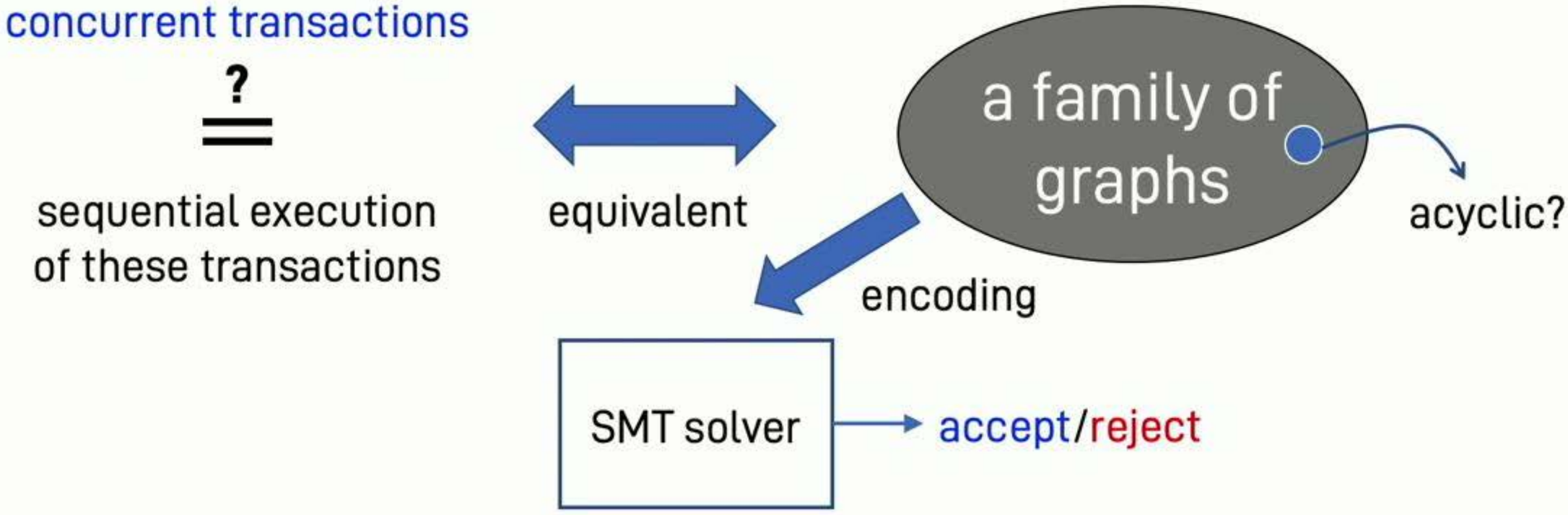
Cobra aims at real-world workloads

- Intuition: advances of SAT/SMT solvers
- Starting point (Papadimitriou's construction):

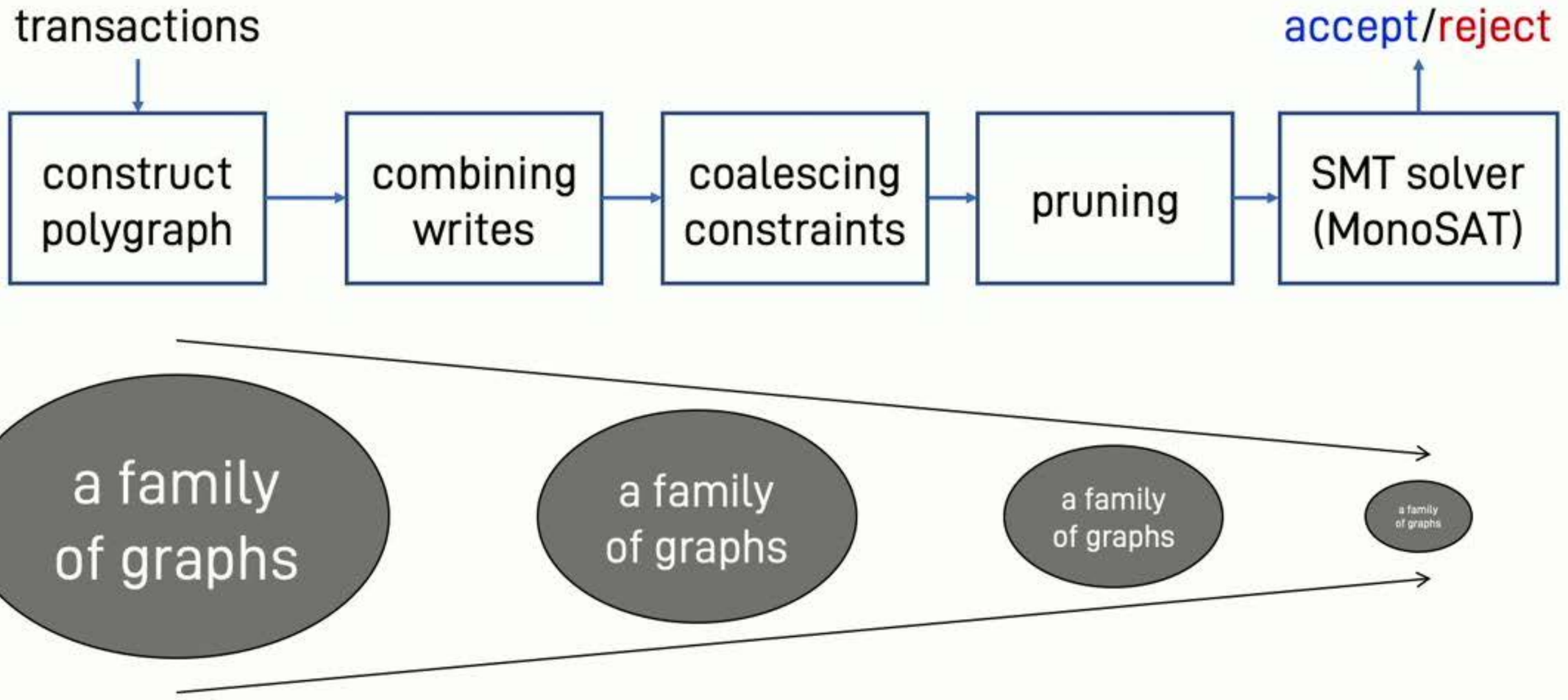


Cobra aims at real-world workloads

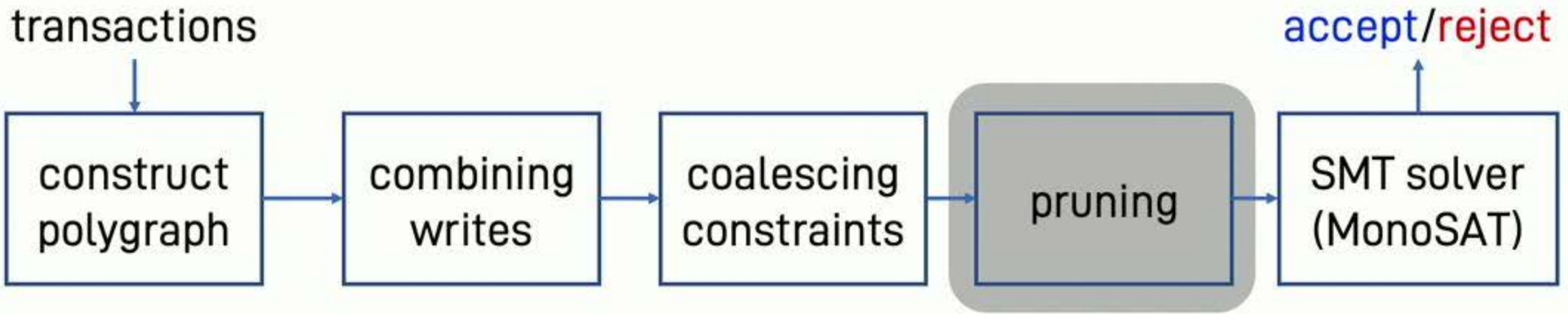
- Intuition: advances of SAT/SMT solvers
- Starting point (Papadimitriou's construction):

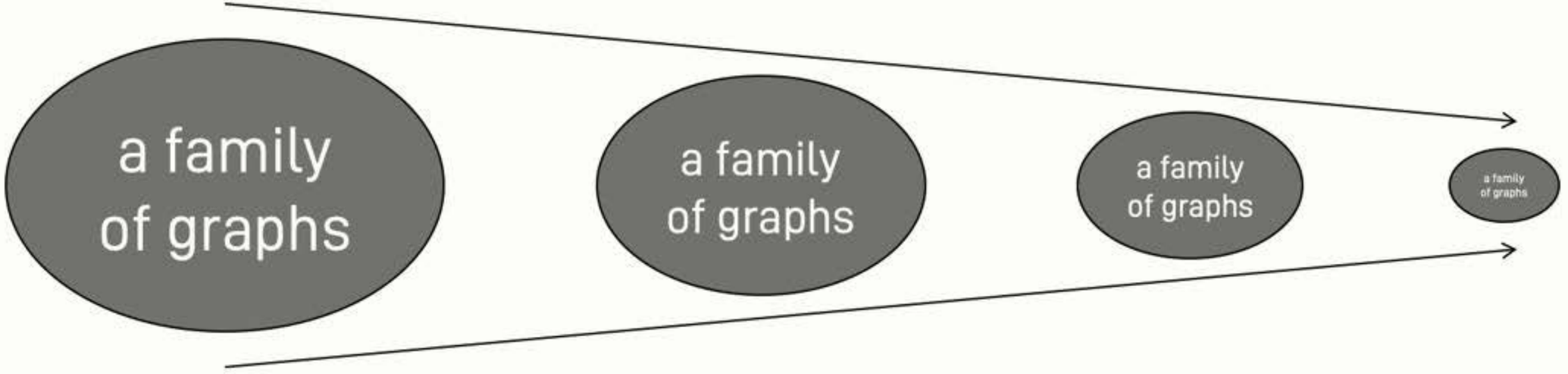


Cobra: narrowing the search space



Cobra: narrowing the search space





Pruning via graph paths (reachability)

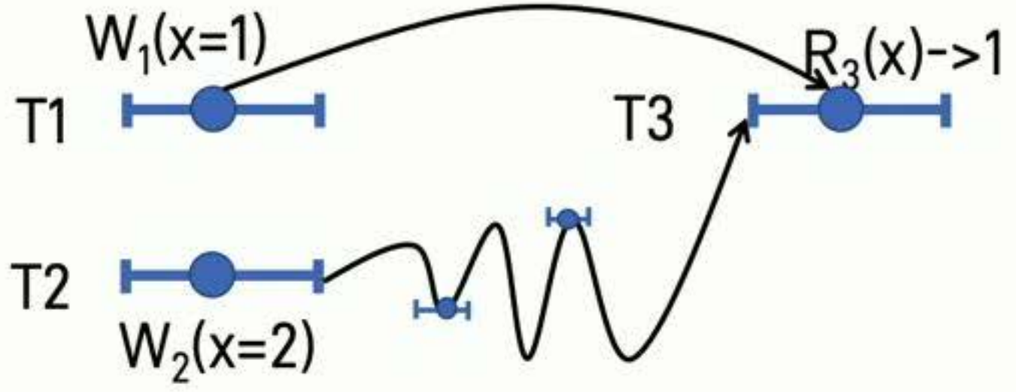
- idea: reduce #graphs by adding constraints to the family

Pruning via graph paths (reachability)

- idea: reduce #graphs by adding constraints to the family
  1) what constraints can be inferred from reachability?
  2) how to get reachability efficiently?

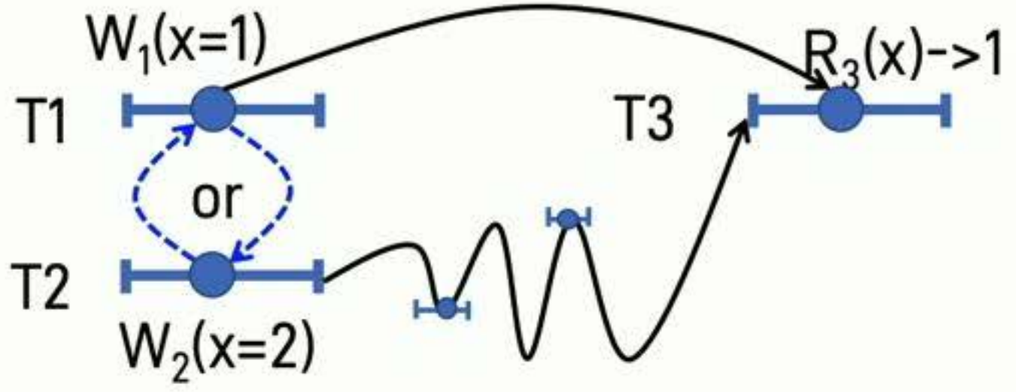
Pruning via graph paths (reachability)

- idea: reduce #graphs by adding constraints to the family
  1) what constraints can be inferred from reachability?
  2) how to get reachability efficiently?



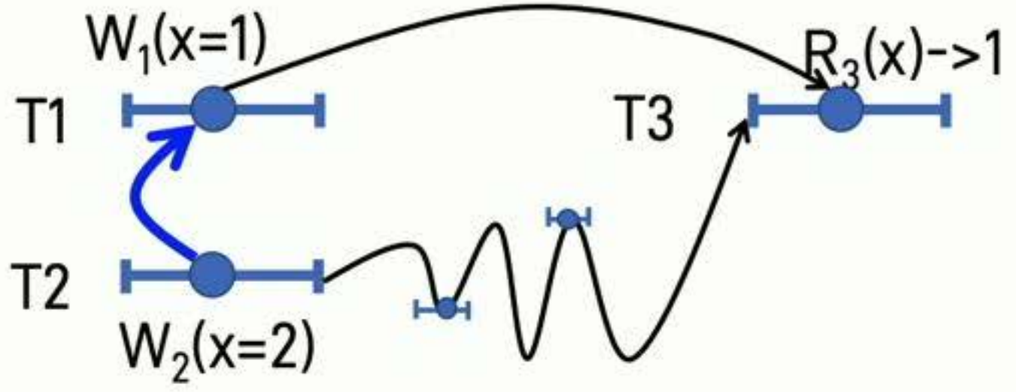
Pruning via graph paths (reachability)

- idea: reduce #graphs by adding constraints to the family
  1) what constraints can be inferred from reachability?
  2) how to get reachability efficiently?



Pruning via graph paths (reachability)

- idea: reduce #graphs by adding constraints to the family
  1) what constraints can be inferred from reachability?
  2) how to get reachability efficiently?



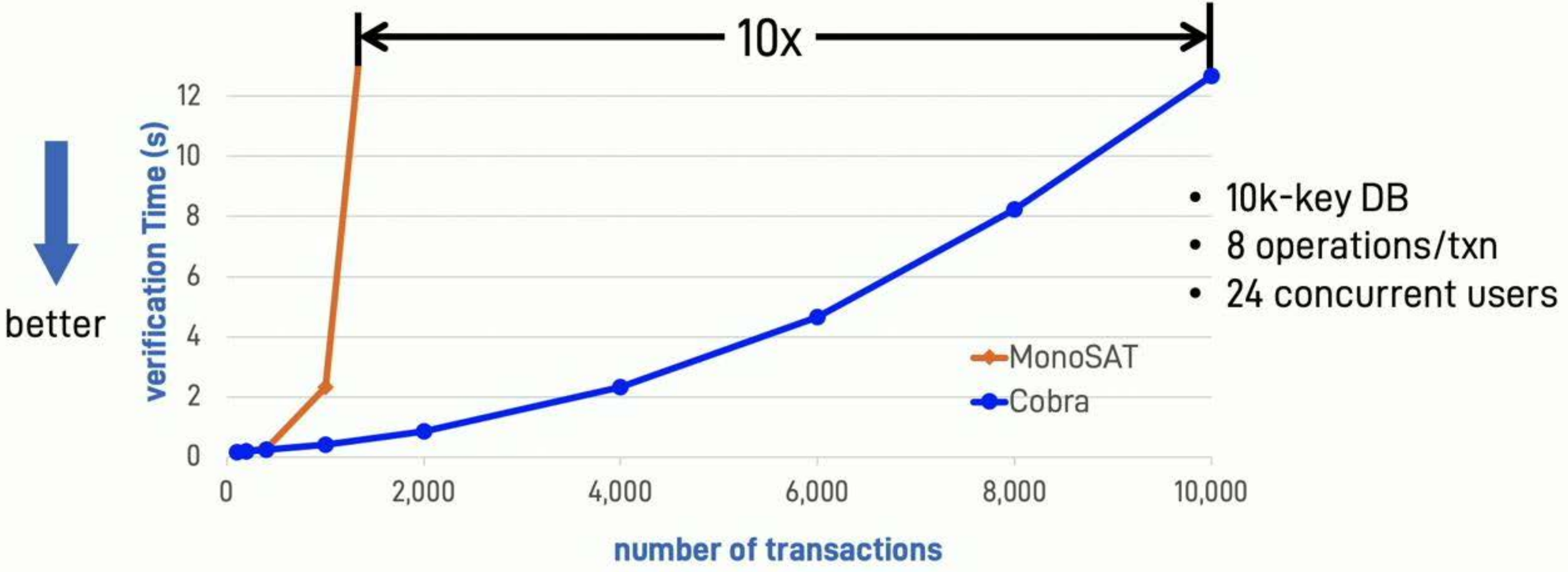
Pruning via graph paths (reachability)

- idea: reduce #graphs by adding constraints to the family
  1) what constraints can be inferred from reachability?
  2) how to get reachability efficiently?

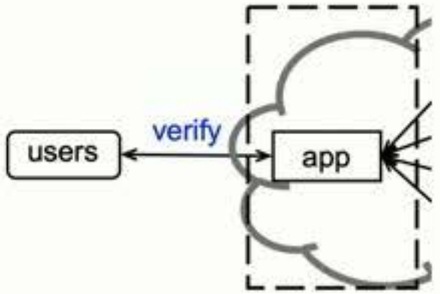
calculating reachability
using Matrix Multiplication

Cobra can handle 10x larger workloads

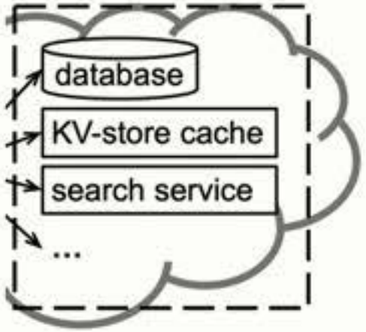
RW benchmark: read-only and write-only transactions (50:50)



① Verify white-box services (Orochi [SOSP'17])

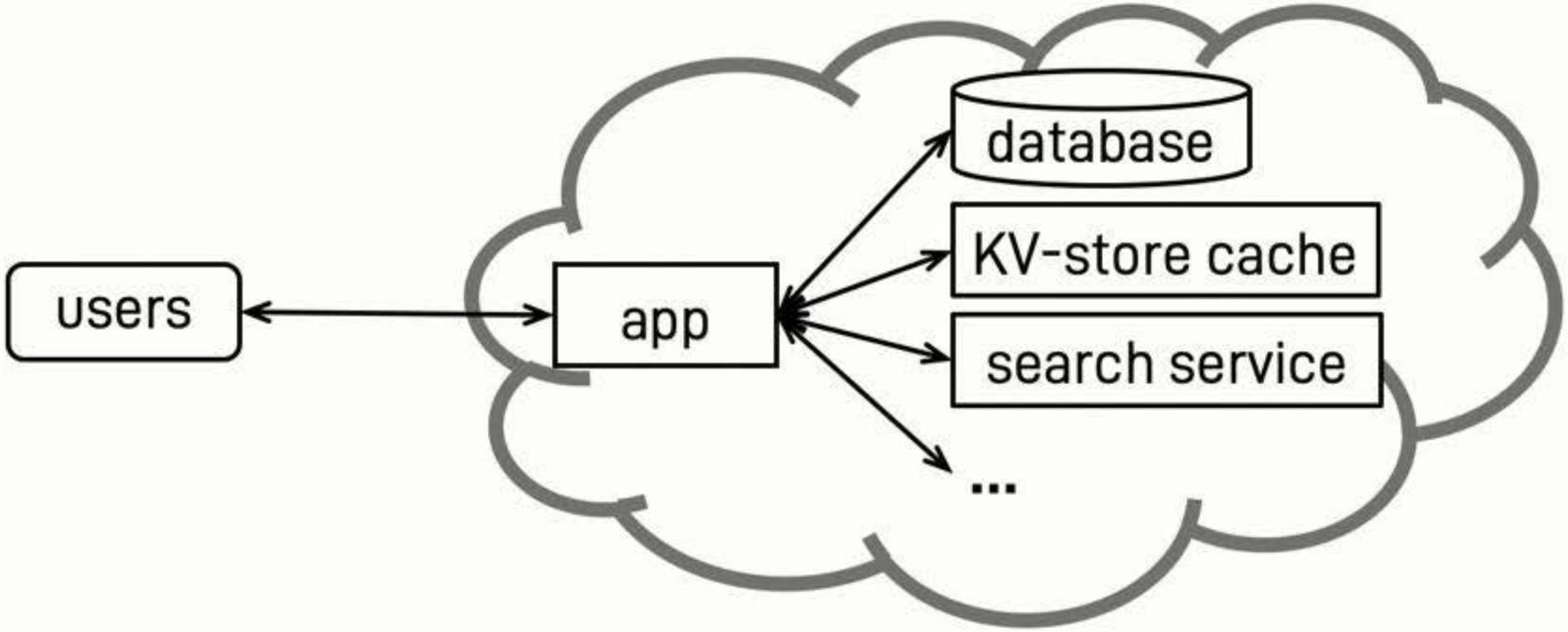


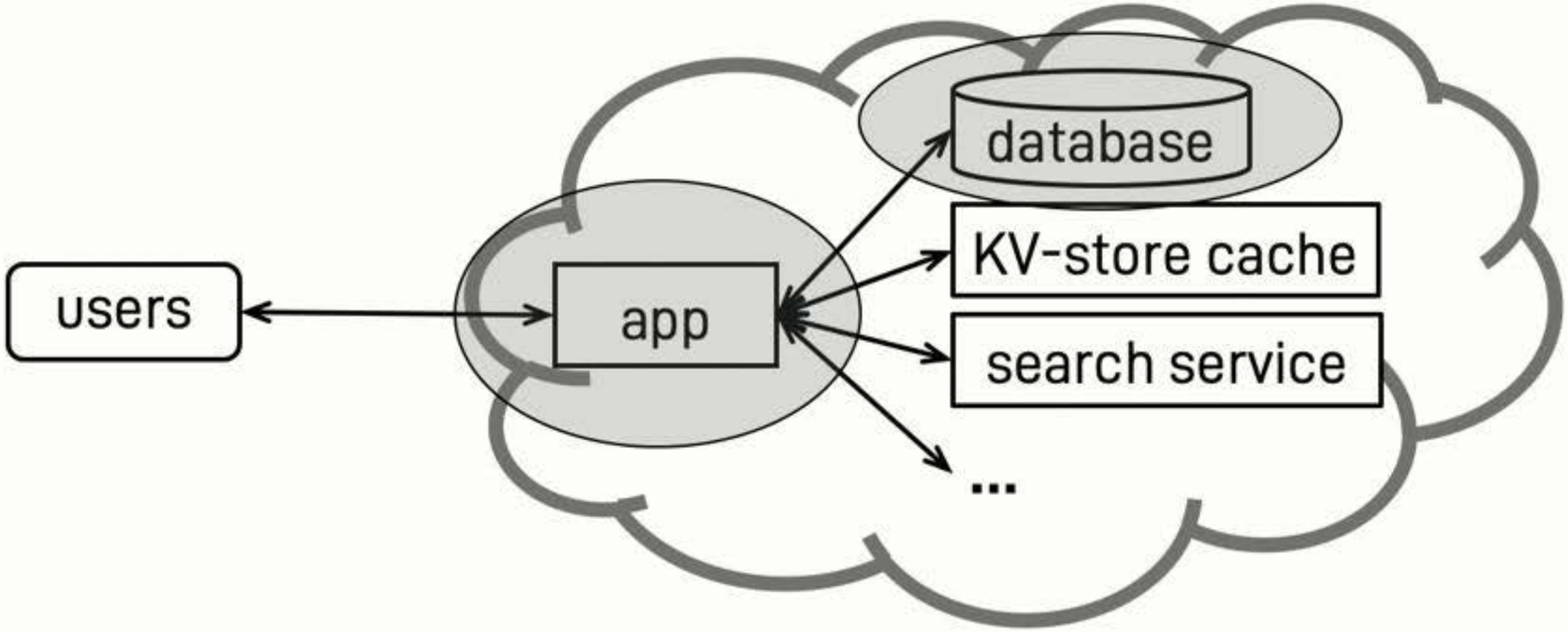
② Verify black-box services (Cobra: verify Serializability)



③ Build composable verifiable framework (future work)

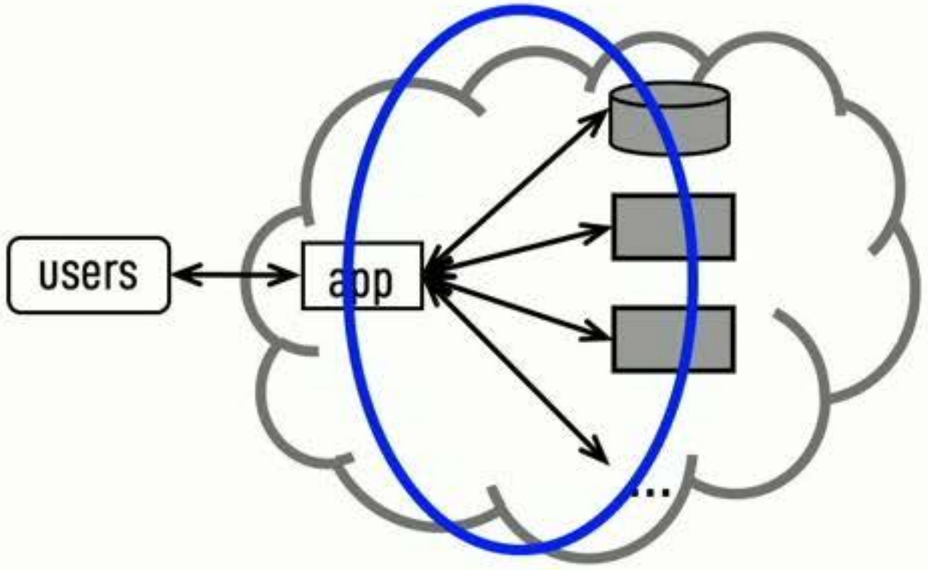
④ Other past work
- Troubleshooting data center networks [NSDI'19]
- Protecting secret via security-oriented offloading [EuroSys'15]


users
app
database
KV-store cache
search service
...

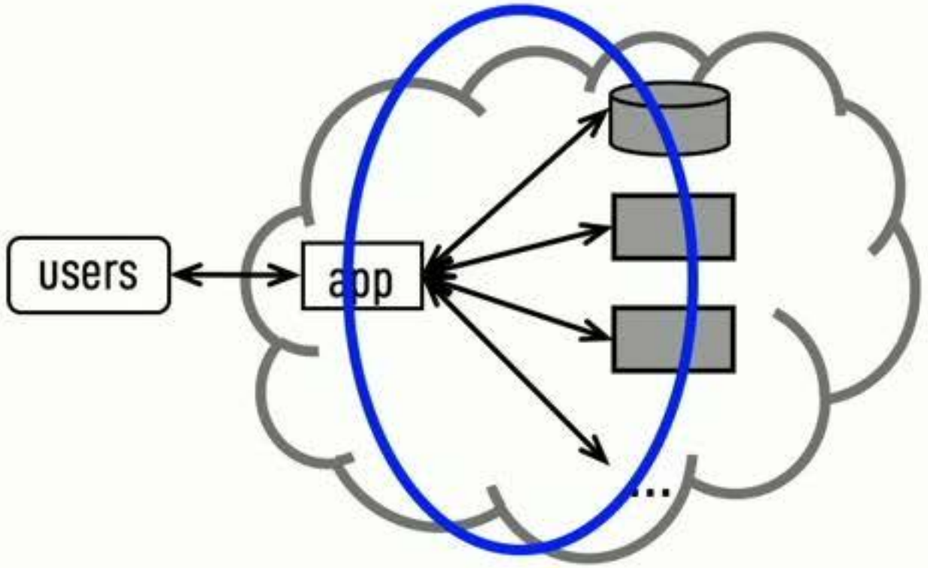

database
KV-store cache
search service
...
users
app

Question #1:

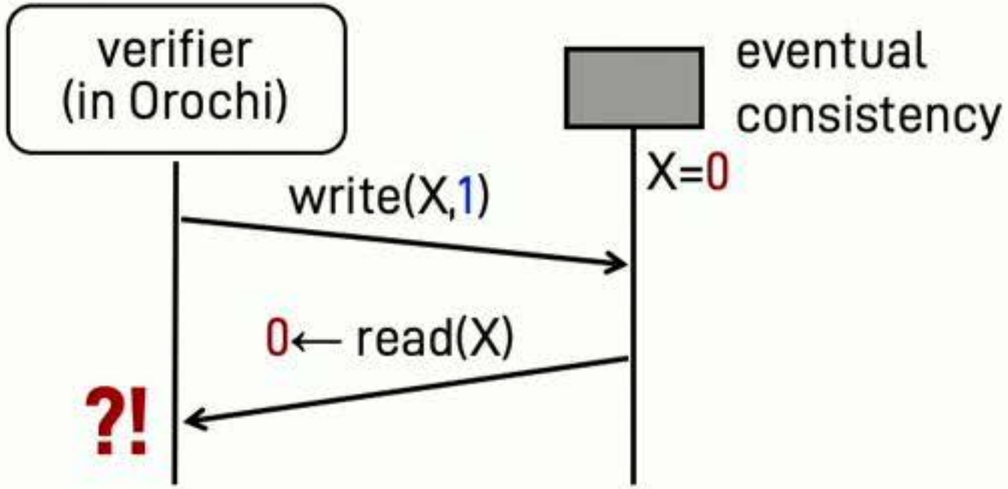
How to compose verifications of multiple services?



Question #1:
How to compose verifications of multiple services?

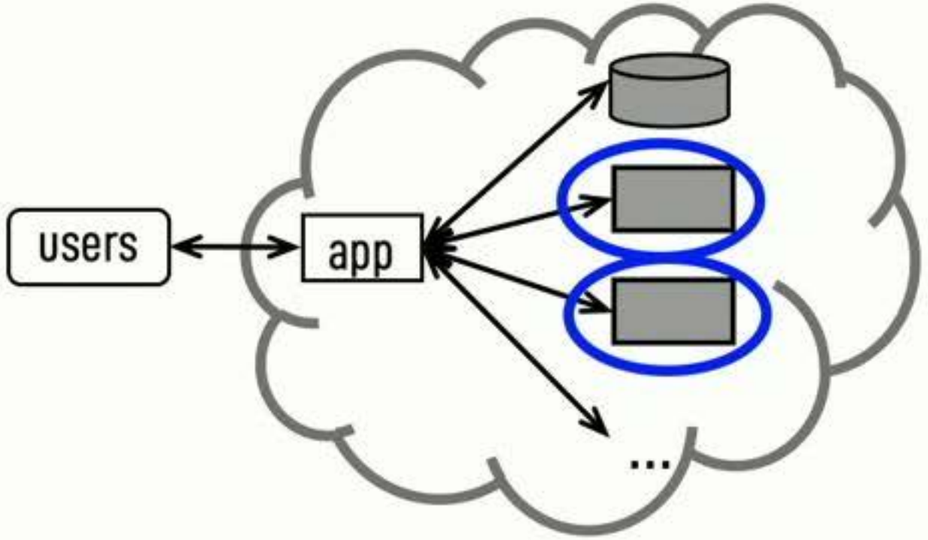


- Challenge: some services do not respect real-time order

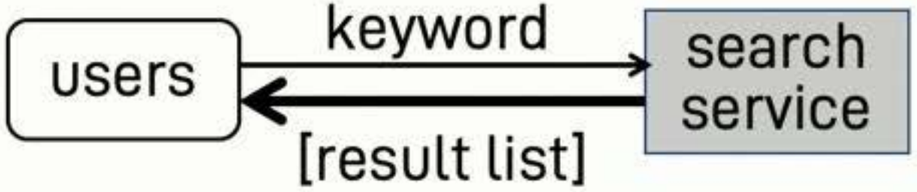


Question #2:

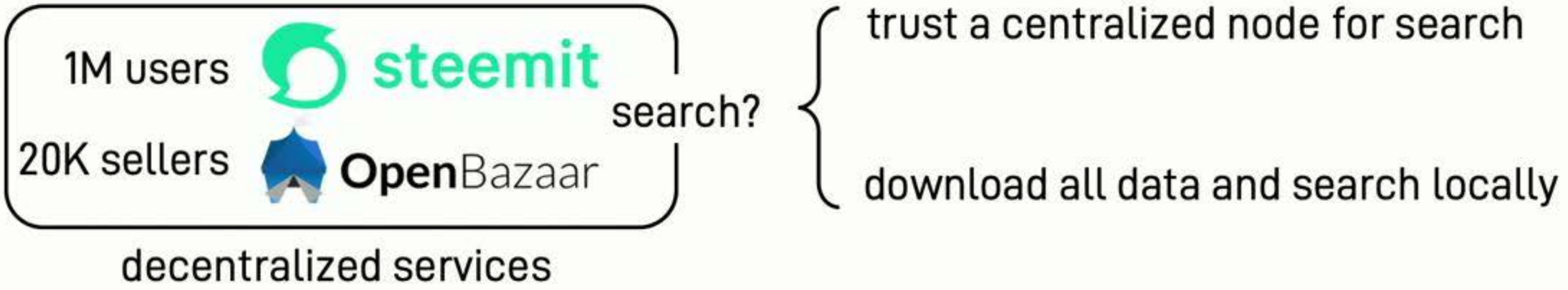
How to verify various black-box services?



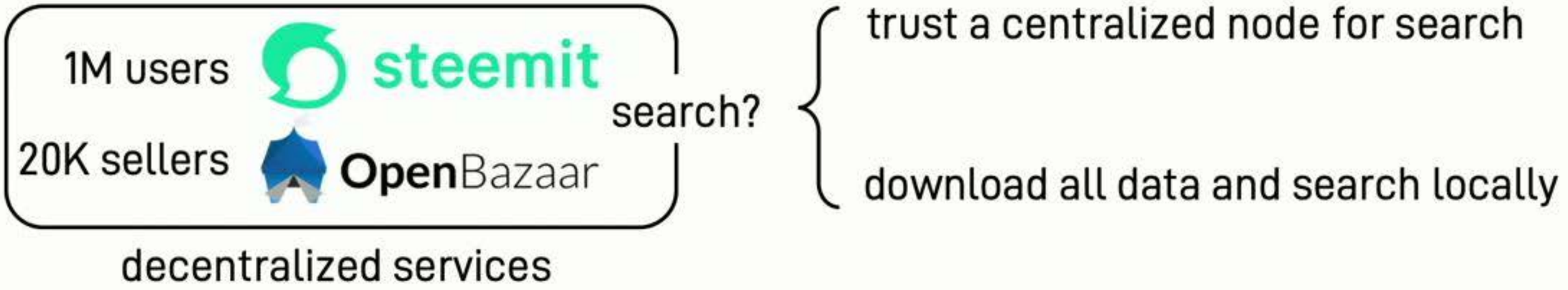
for example, a search service



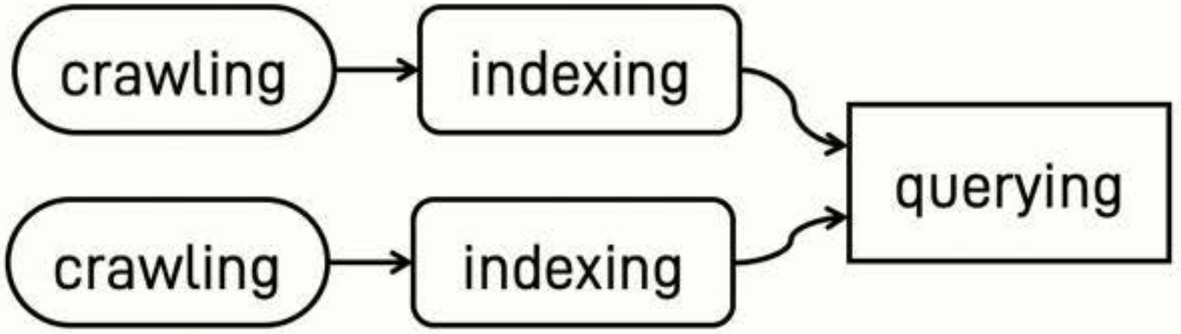
Verifiable search engine



Verifiable search engine

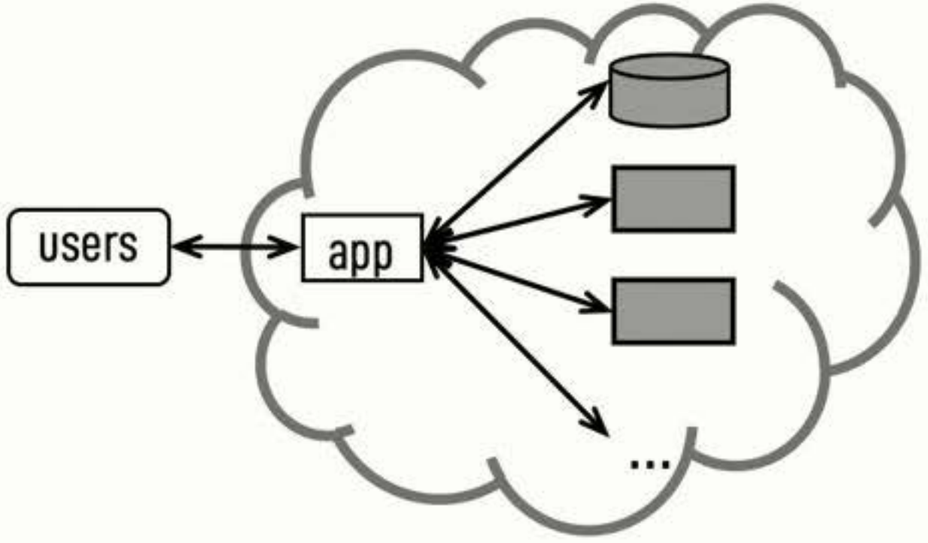


- Requirement (challenge): whole search pipeline needs to be verifiable



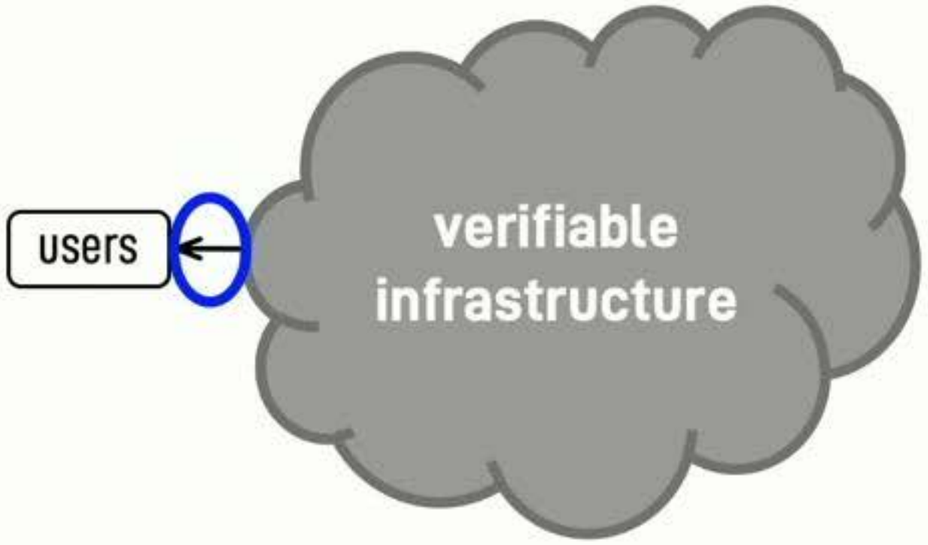
Question #3:

How to verify properties other than execution integrity?



Question #3:

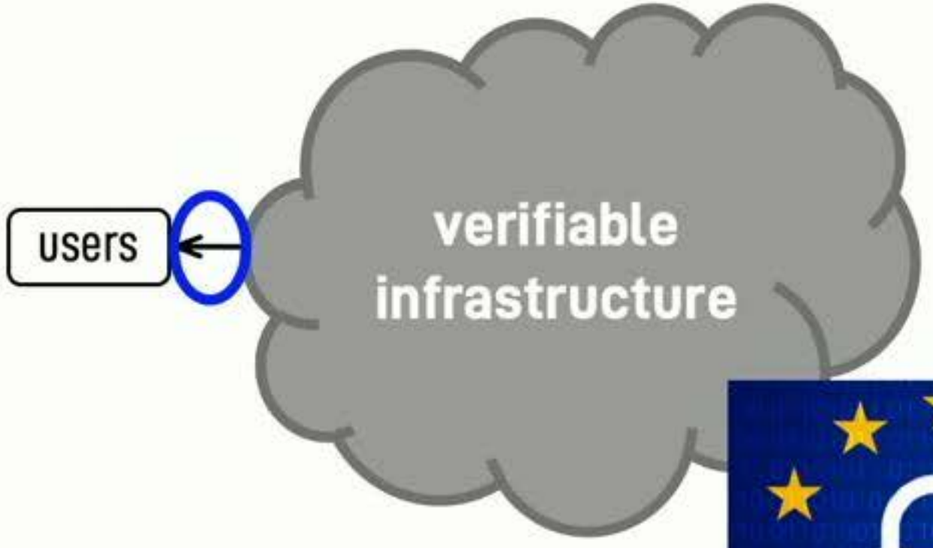
How to verify properties other than execution integrity?



- confidentiality? privacy?
- meta-data privacy?

Question #3:

How to verify properties other than execution integrity?



- confidentiality? privacy?
- meta-data privacy?
- rules (or laws)? GDPR? CCPA?

A provable GDPR framework for web apps

- Motivation

auditor

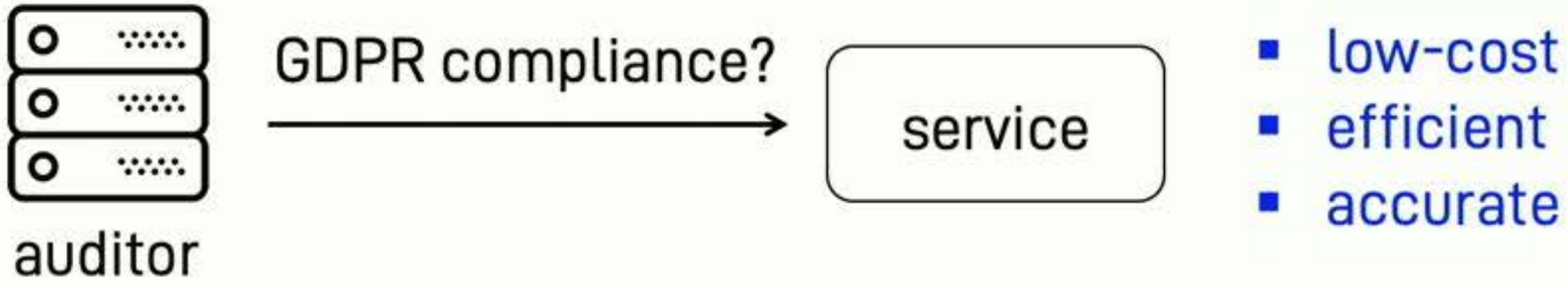
GDPR compliance? →

service

- expensive
- time consuming
- error-prone

A provable GDPR framework for web apps

- Motivation



A provable GDPR framework for web apps

- Motivation

- low-cost
- efficient
- accurate

- Requirements (challenges):
  - a machine-checkable GDPR definition
  - audit puts zero trust on the service
  - audit must be efficient

A provable GDPR framework for web apps

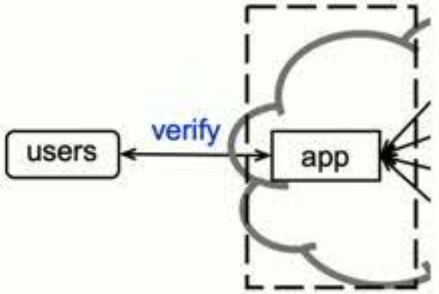
- Motivation

- low-cost
- efficient
- accurate

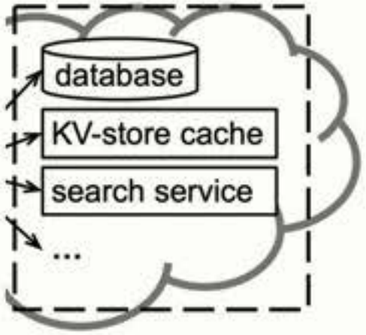
- Requirements (challenges):
  - a machine-checkable GDPR definition
  - audit puts zero trust on the service
  - audit must be efficient

} Orochi and Cobra can help.

① Verify white-box services (Orochi [SOSP'17])



② Verify black-box services (Cobra: verify Serializability)

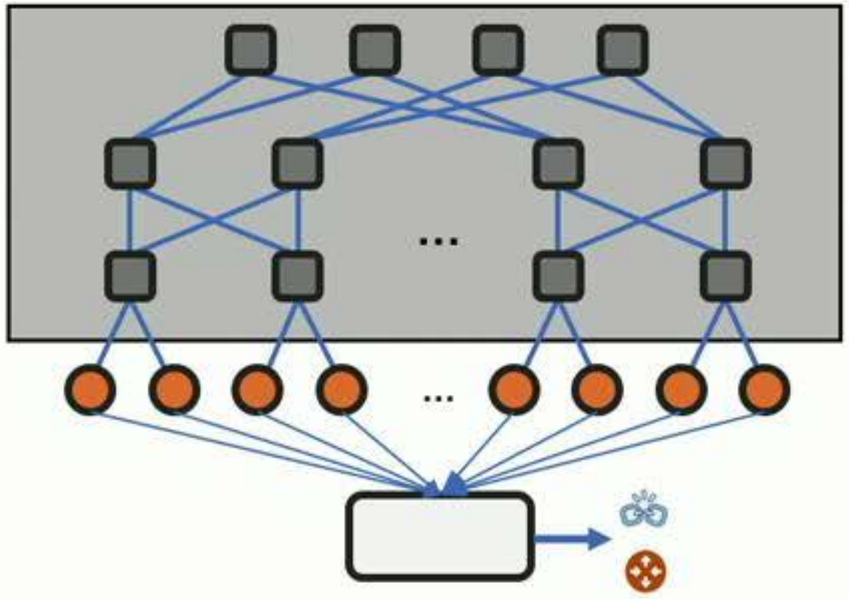


③ Build composable verifiable framework (future work)

④ Other past work
- Troubleshooting data center networks [NSDI'19]
- Protecting secret via security-oriented offloading [EuroSys'15]

NetBouncer [TJGZWDBX, NSDI'19]
localizing failures in data center networks

without trusting any
failure information from network

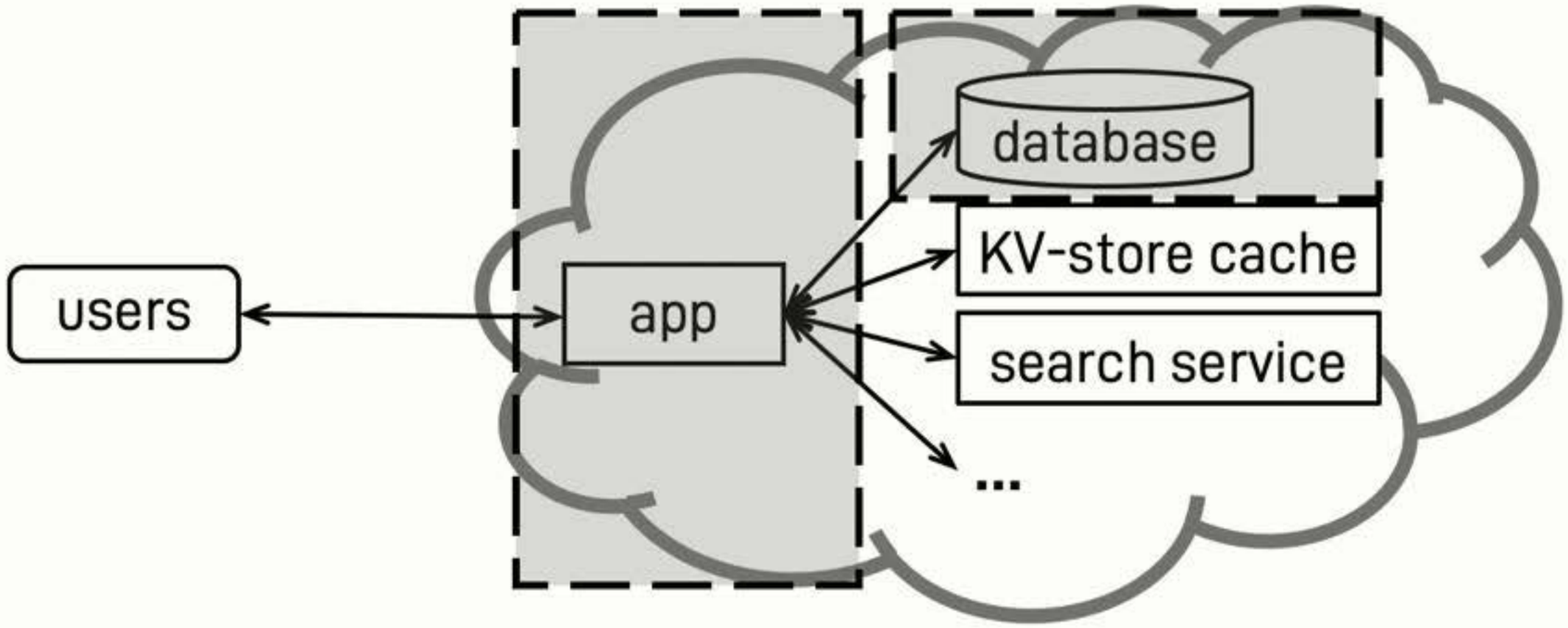


infer failures with end-to-end
observation of network packet loss

- deployed in Microsoft Azure
- detected overlooked failures

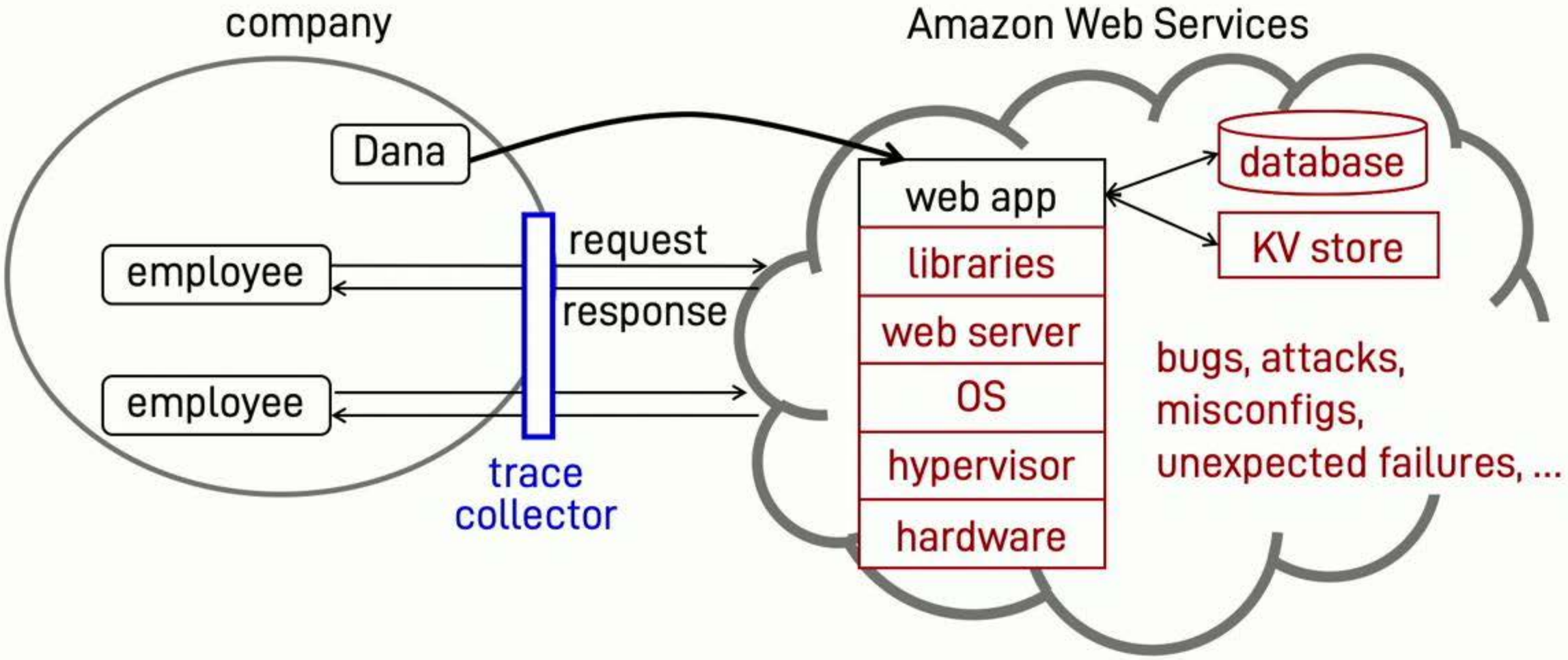
Verifiable infrastructure

... enables users to verify outsourced services.



Orochi:
verifying white-box services

Cobra:
verifying black-box databases


company
Amazon Web Services
Dana
employee
employee
request
response
trace
collector
web app
libraries
web server
OS
hypervisor
hardware
database
KV store
bugs, attacks,
misconfigs,
unexpected failures, ...

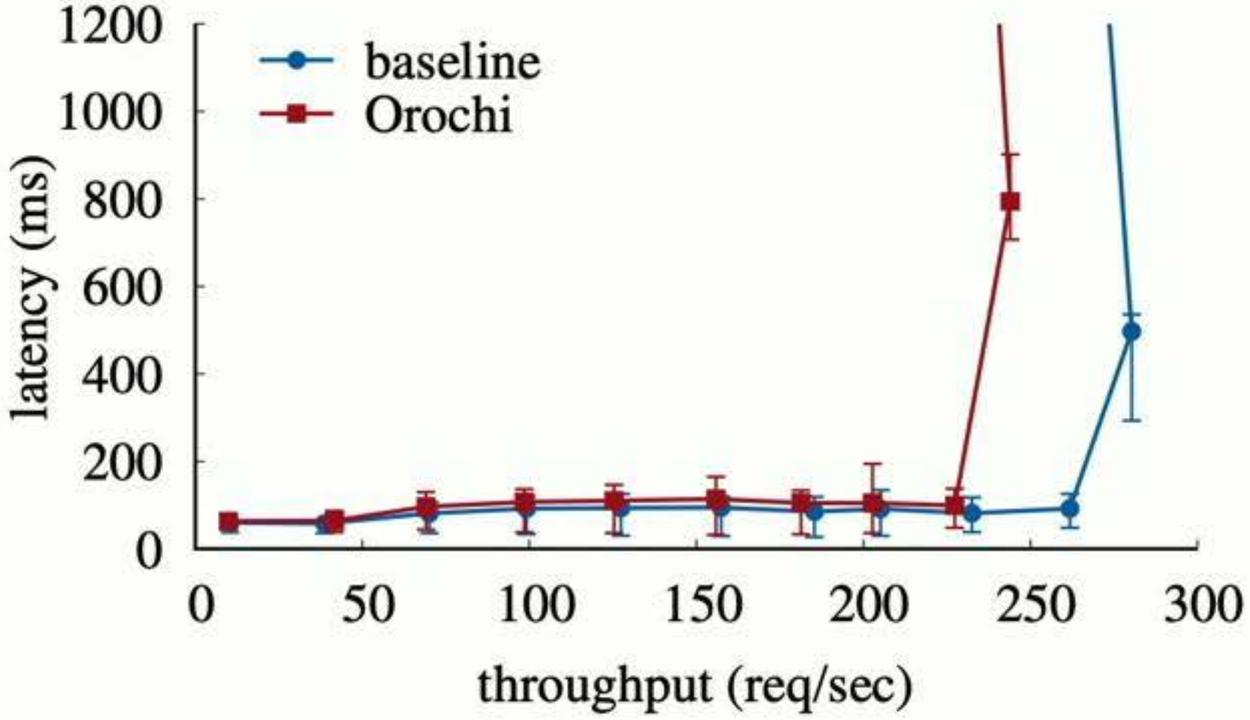
What are the (server's) CPU/network/storage costs?

CPU	Network			Storage
	trace (per req)	advice (per req)	Orochi's overhead	
MediaWiki's workload	MediaWiki's workload			MediaWiki's workload
4.7%	7.1KB	1.7KB	11.4%	1.0x

What are the (server's) CPU/network/storage costs?

CPU	Network			Storage
	trace (per req)	advice (per req)	Orochi's overhead	
MediaWiki's workload				MediaWiki's workload
4.7%	MediaWiki's workload			1.0x
phpBB's workload	7.1KB	1.7KB	11.4%	phpBB's workload
8.6%	phpBB's workload			1.7x
HotCRP's workload	5.7KB	0.3KB	2.7%	HotCRP's workload
5.9%	HotCRP's workload			1.5x
	3.2KB	0.4KB	10.9%	

Orochi imposes small overheads on throughput and latency



phpBB's workload

Why Serializability (SER)?

gold standard
isolation level


CockroachDB
(2015)


Amazon Aurora
(2015)


Google Spanner
(2017)


YugaByteDB
(2017)

...

fundamental
and
challenging