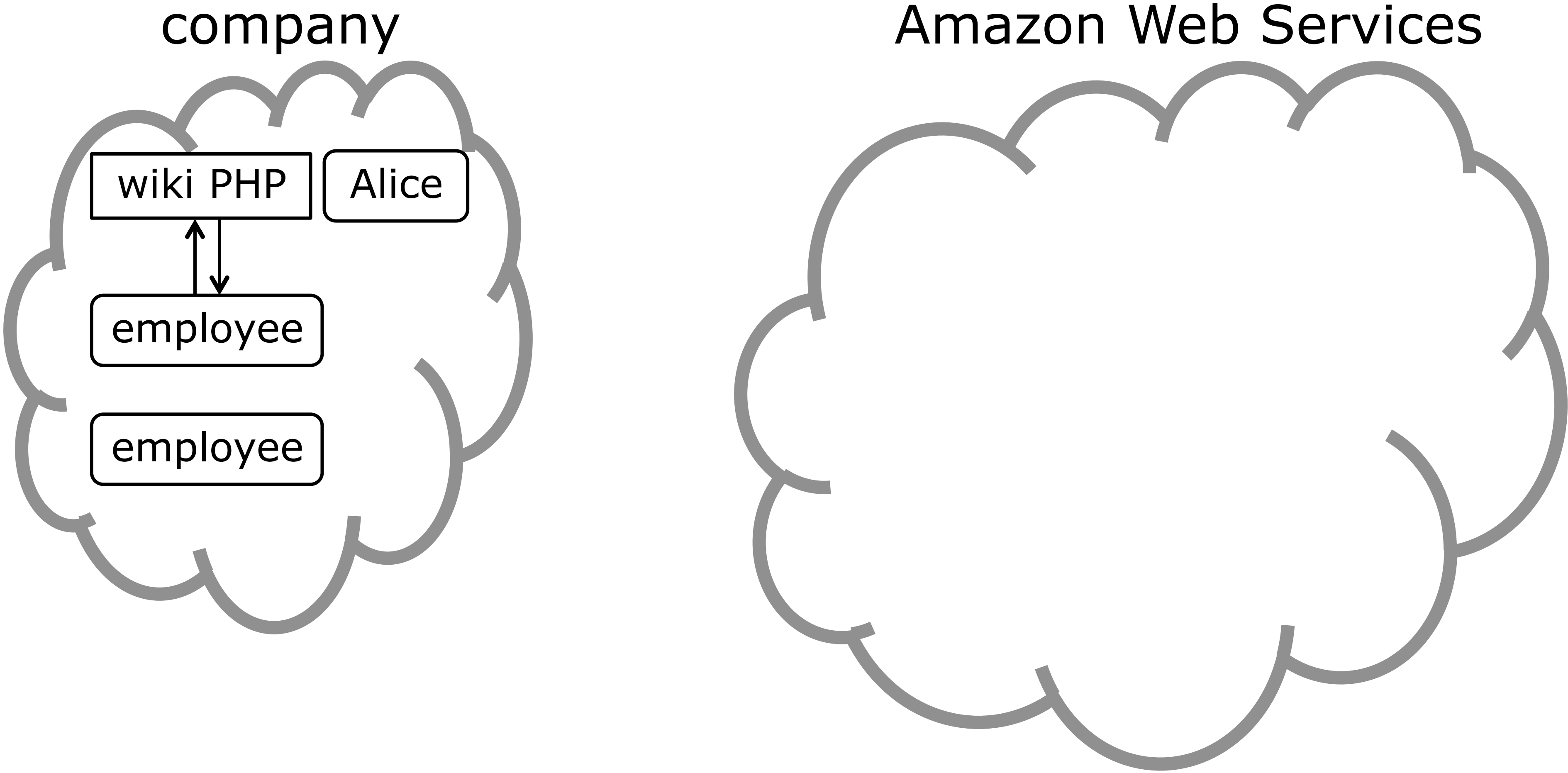
The Efficient Server Audit Problem, Deduplicated Re-execution, and the Web

Cheng Tan, Lingfan Yu,
Joshua B. Leners*, and Michael Walfish

NYU Department of Computer Science, Courant Institute
*Two Sigma Investments



company

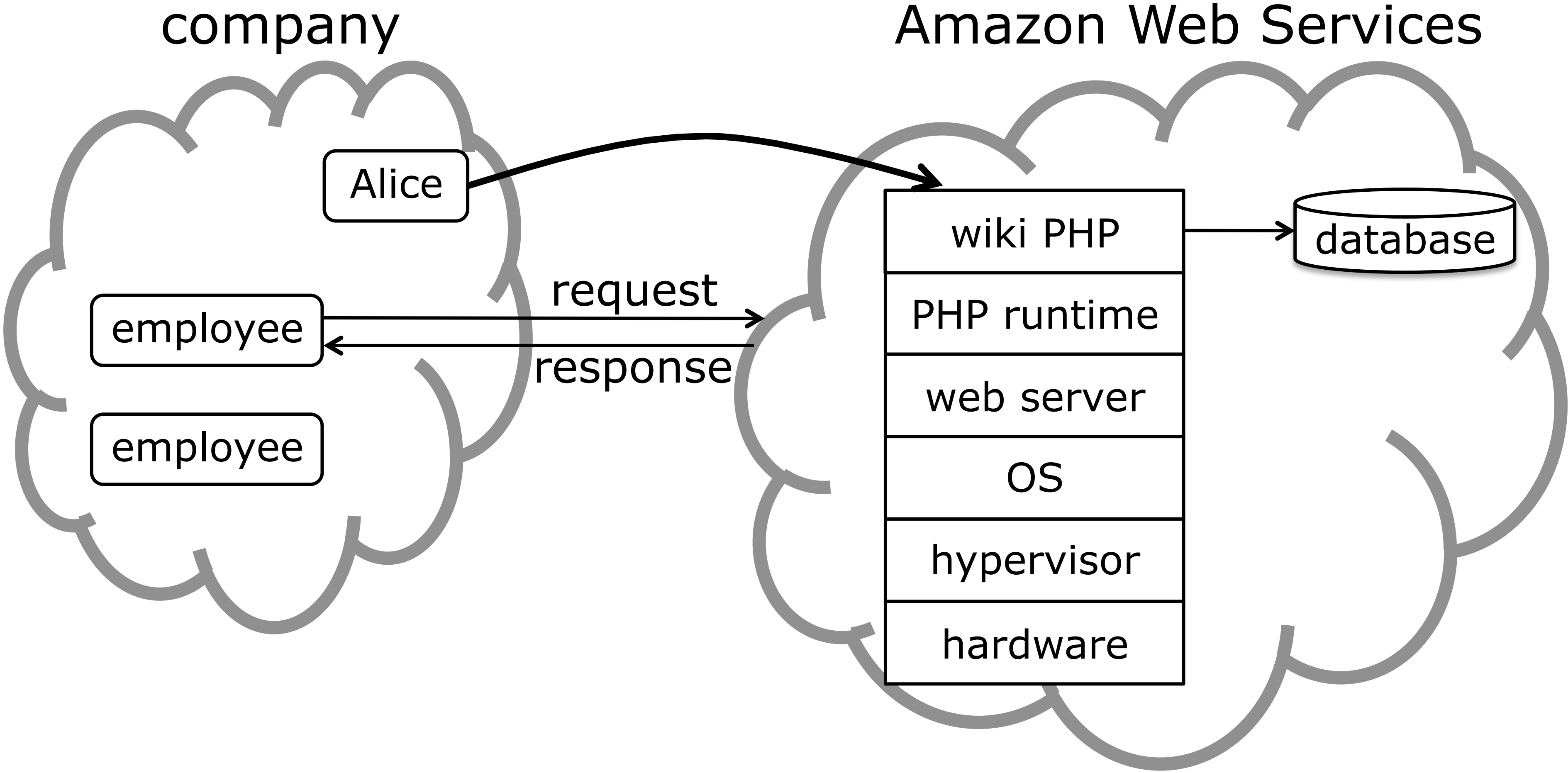
wiki PHP

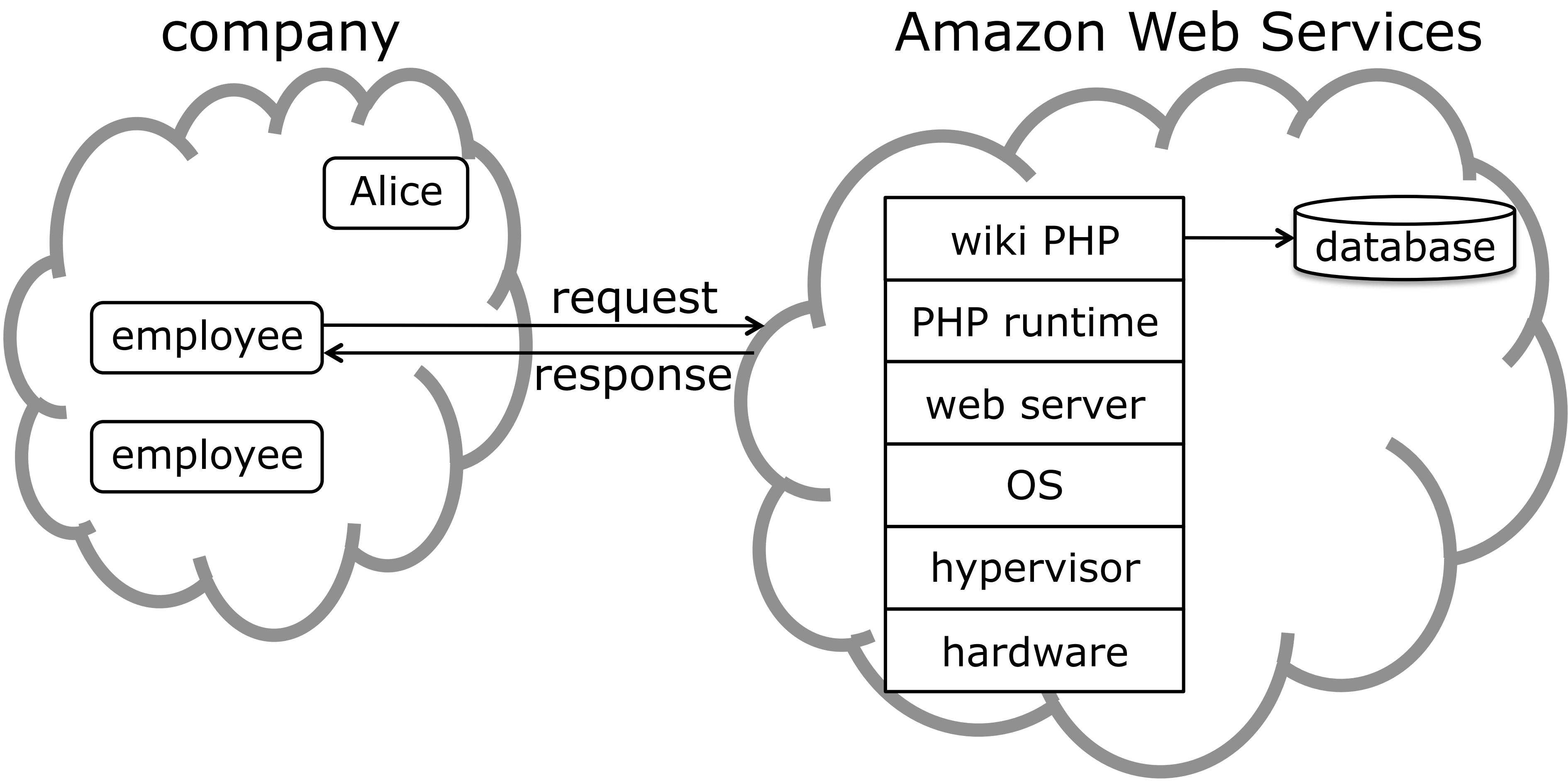
Alice

employee

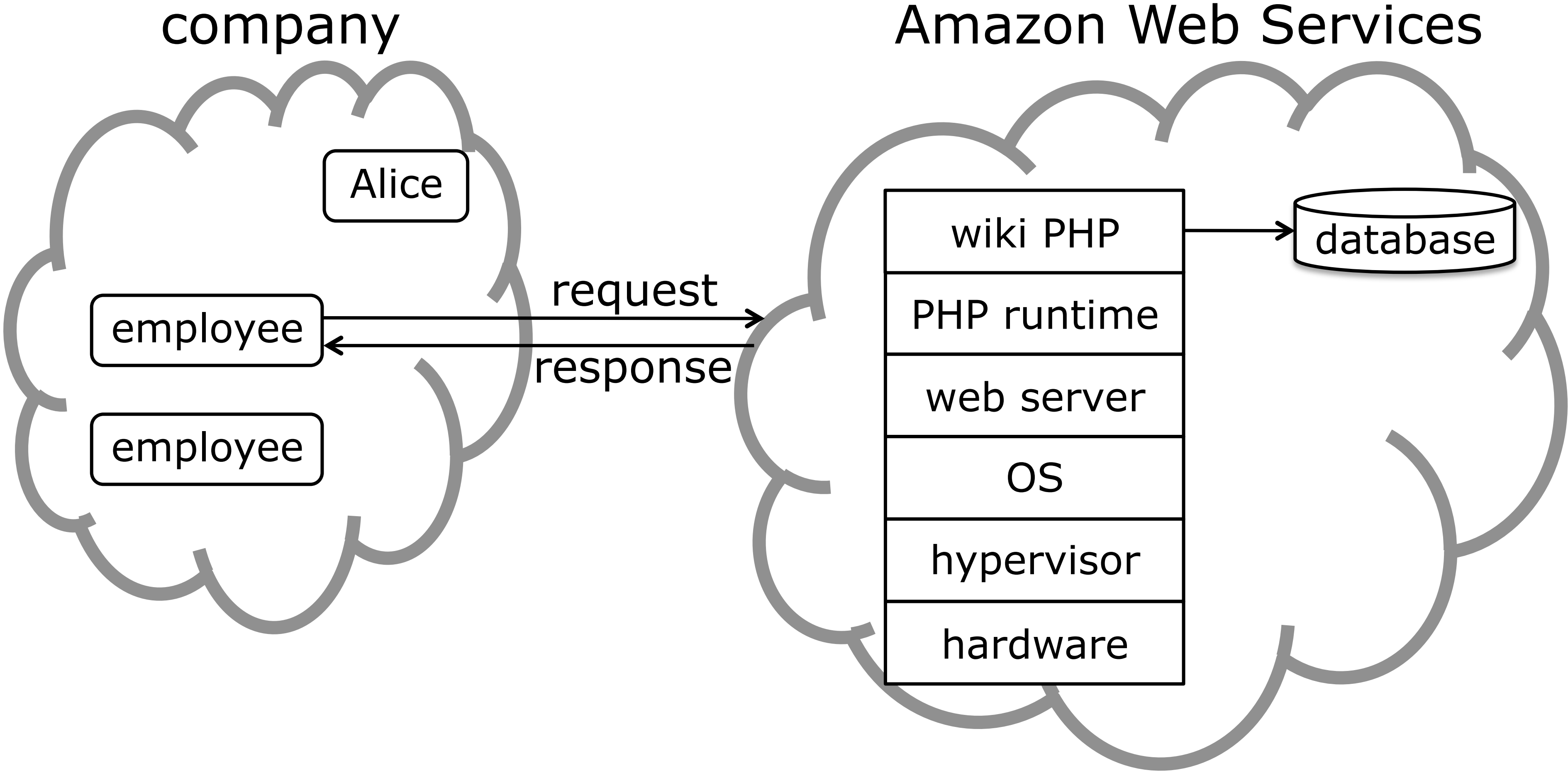
employee

Amazon Web Services

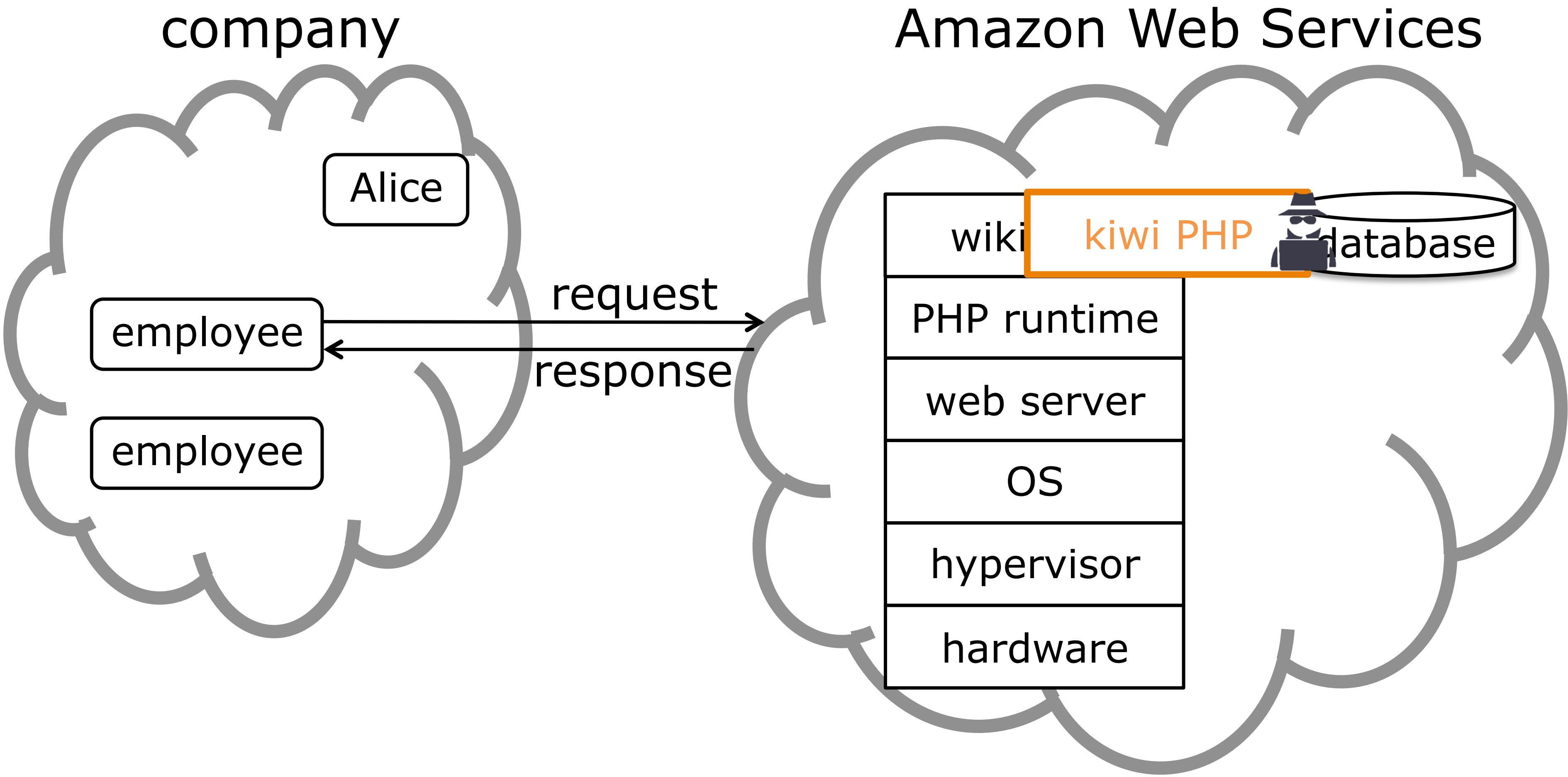

company
Amazon Web Services
Alice
employee
employee
request
response
wiki PHP
PHP runtime
web server
OS
hypervisor
hardware
database



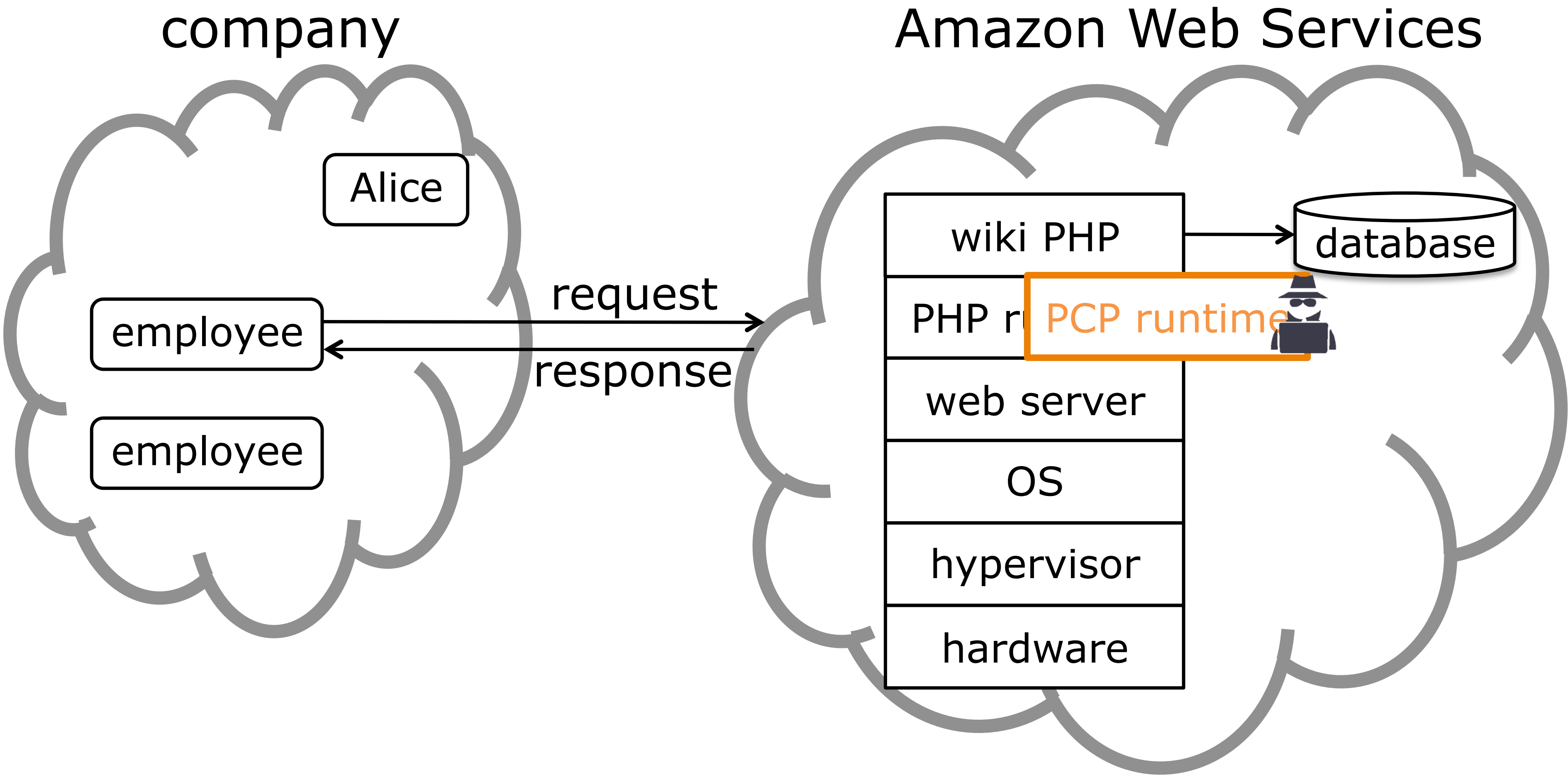
- Alice has confidence in the wiki's PHP code



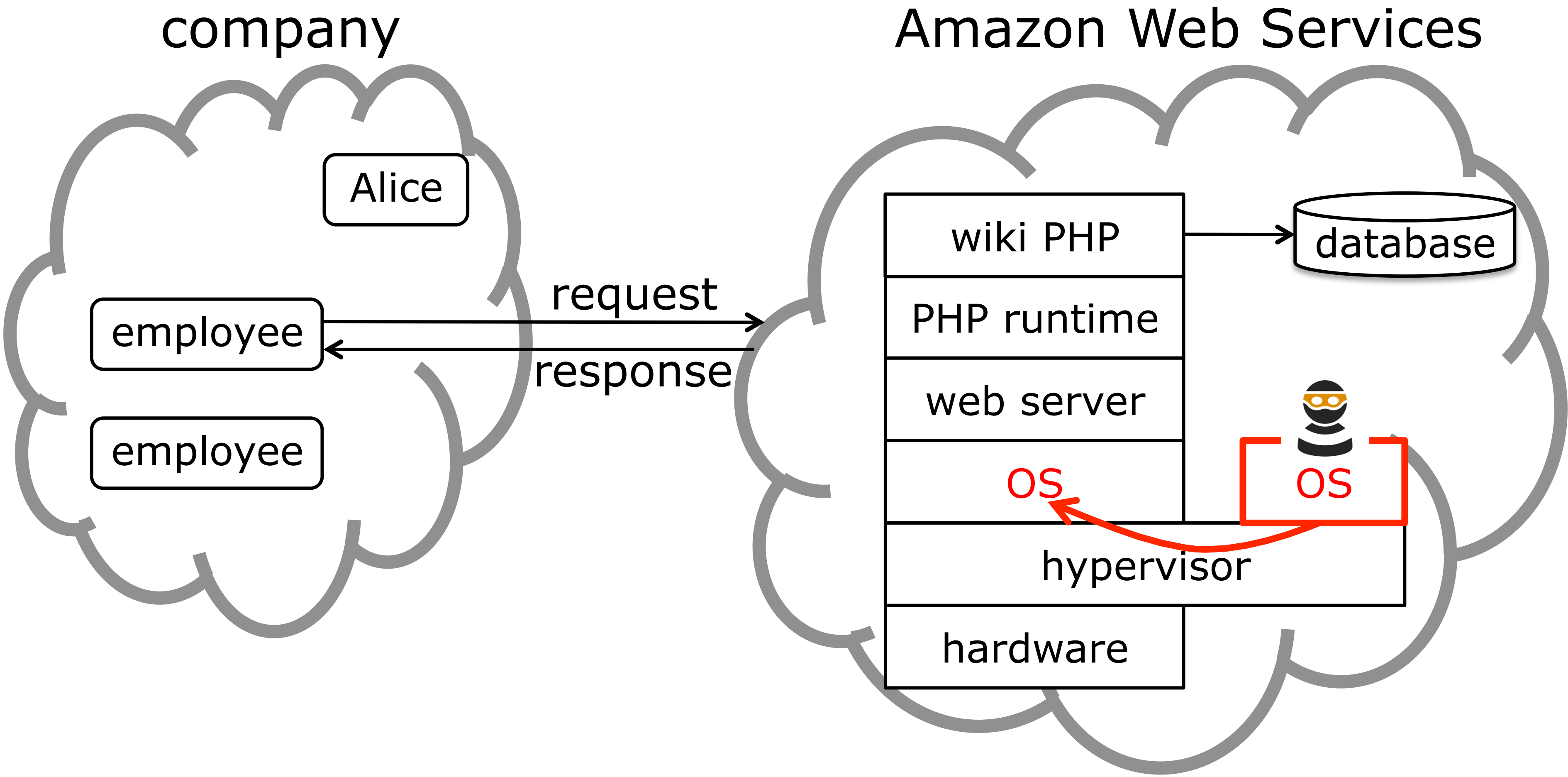
- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong ...



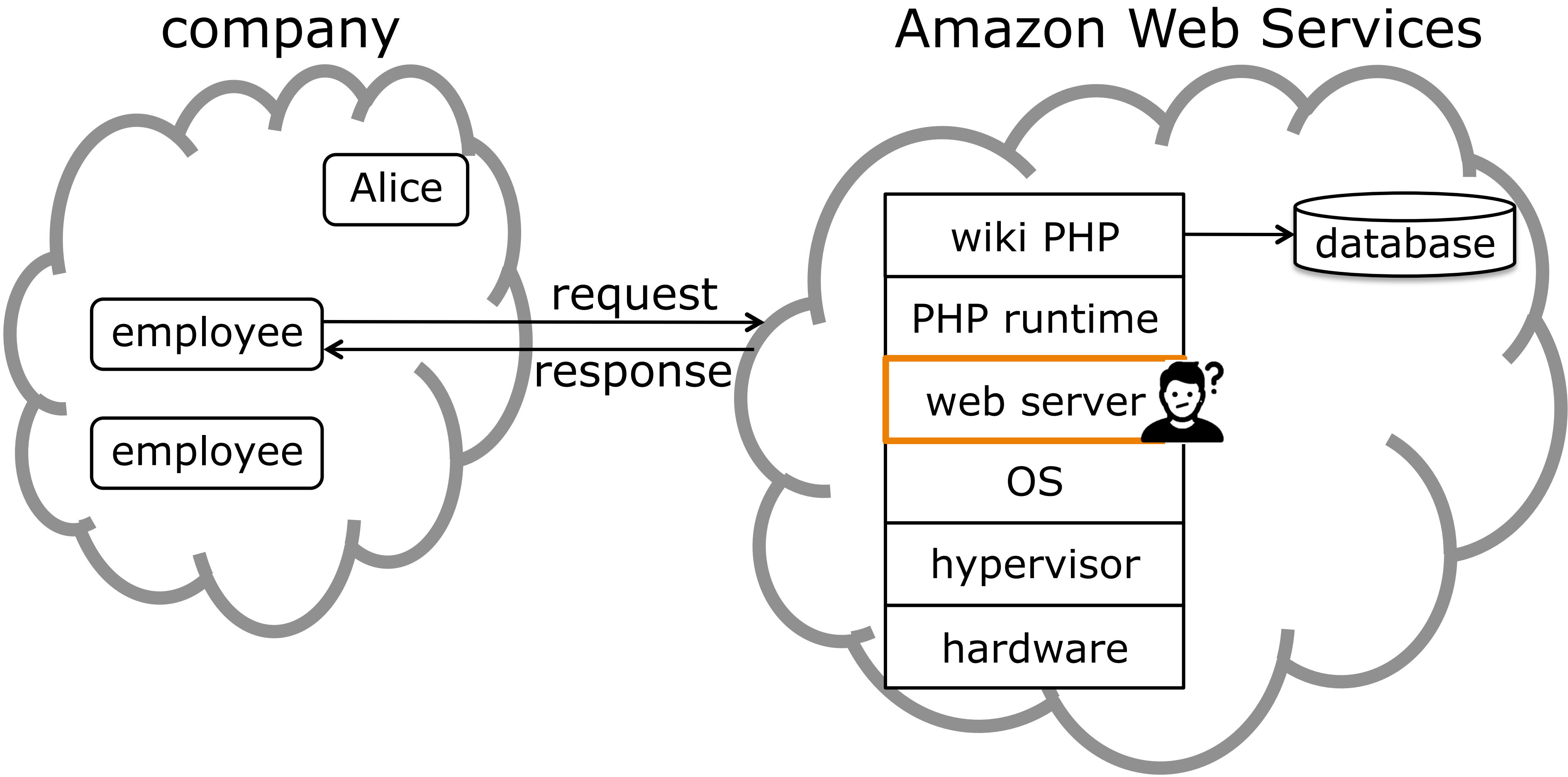
- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong ...



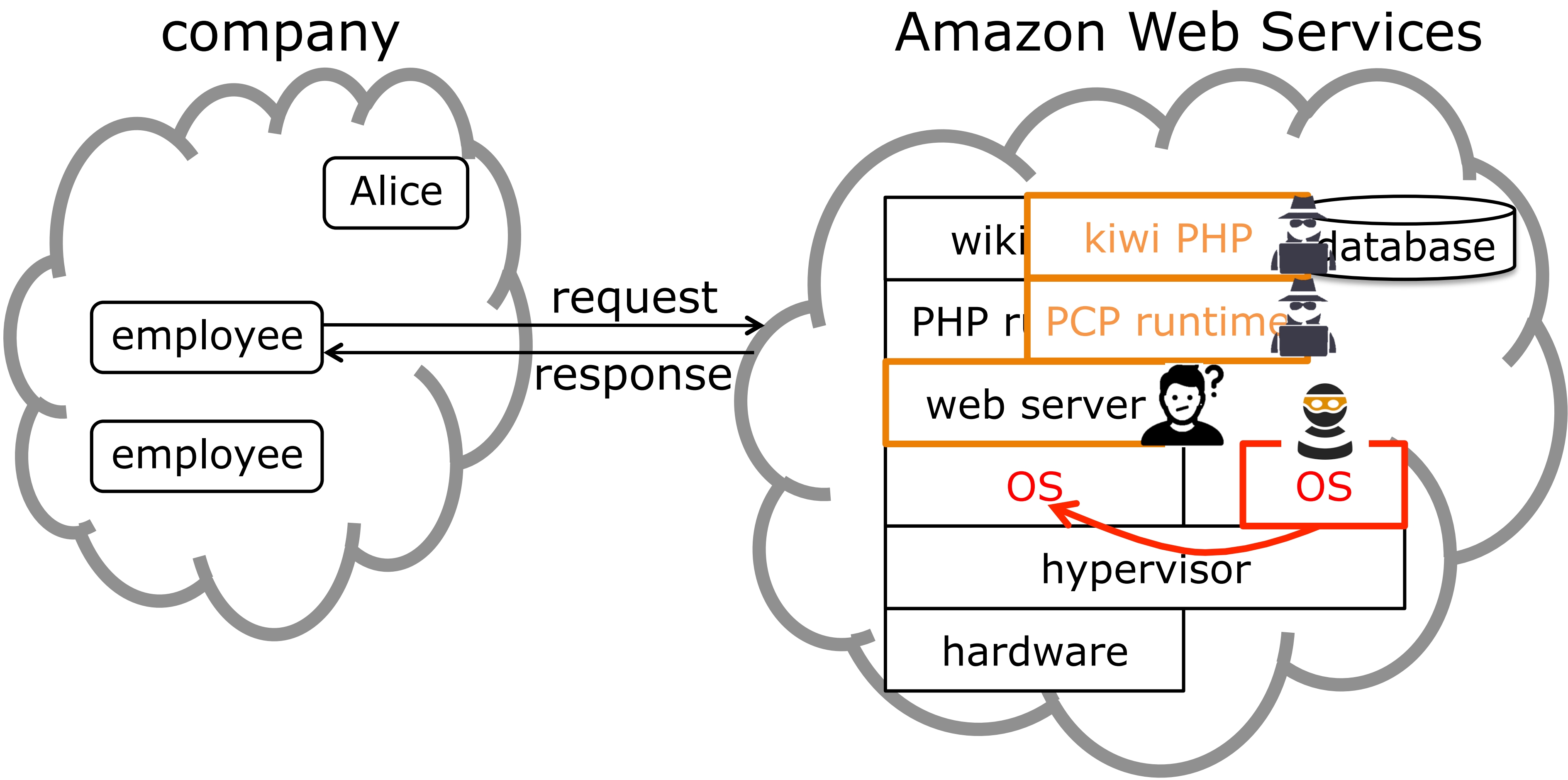
- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong ...



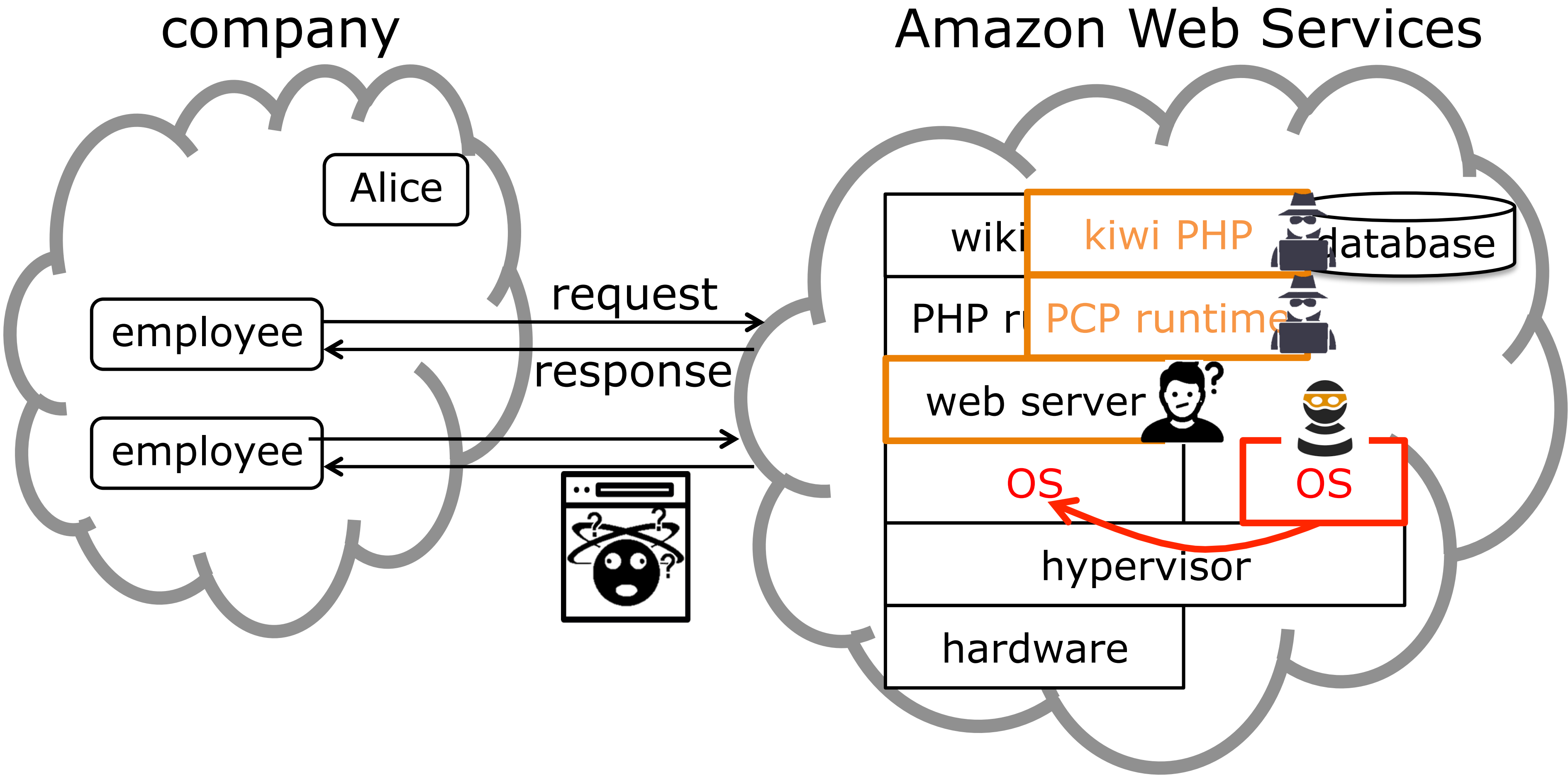
- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong ...



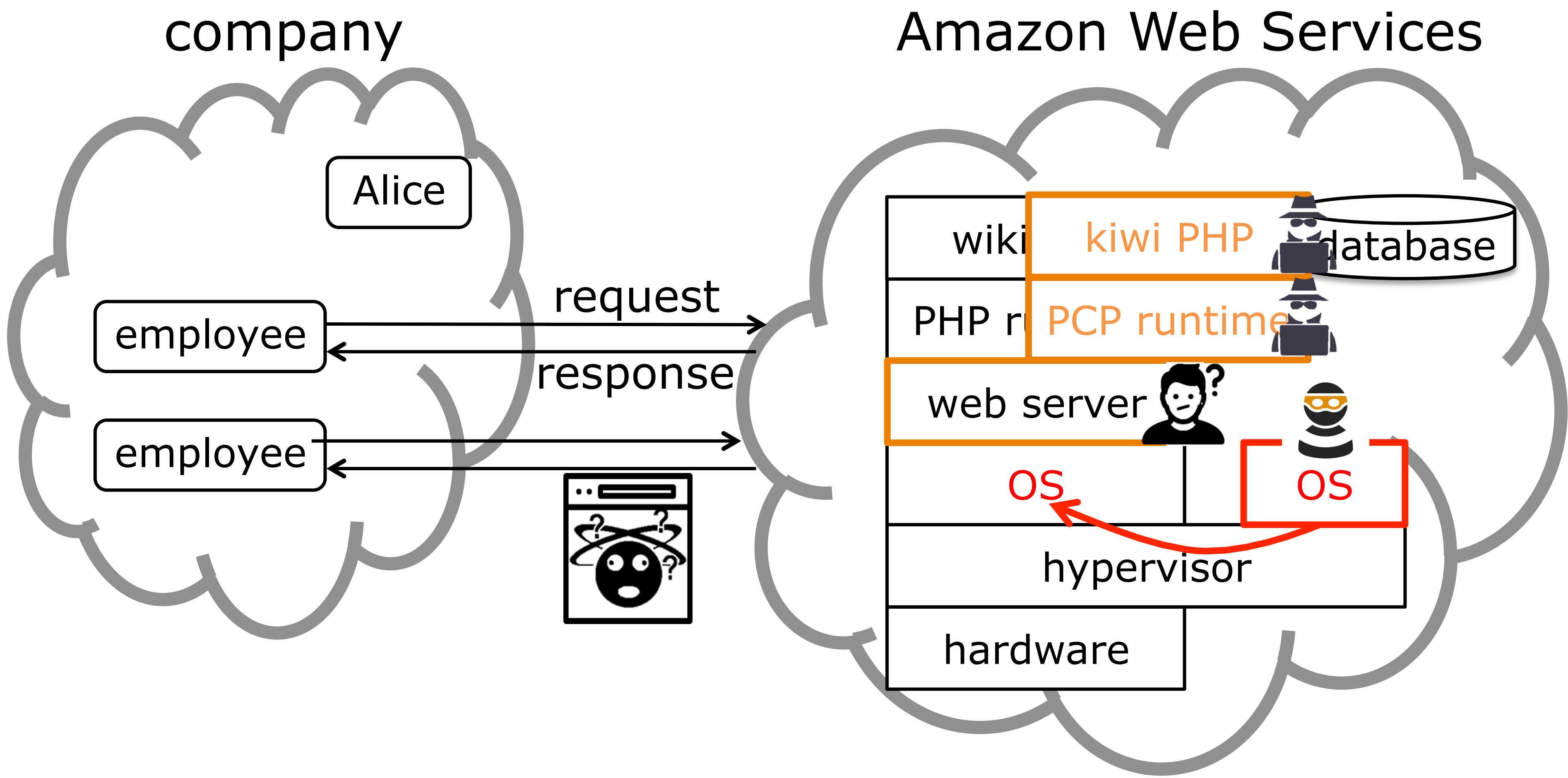
- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong ...



- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong ...

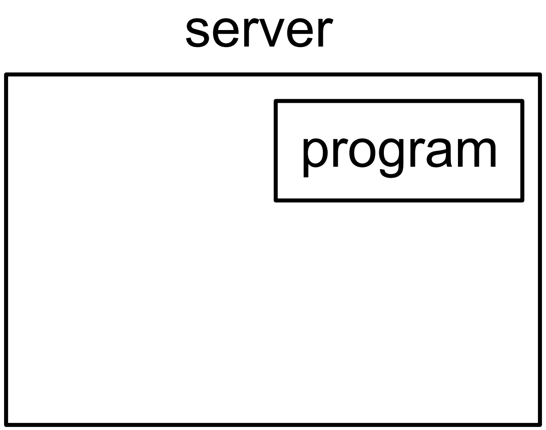


- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong …

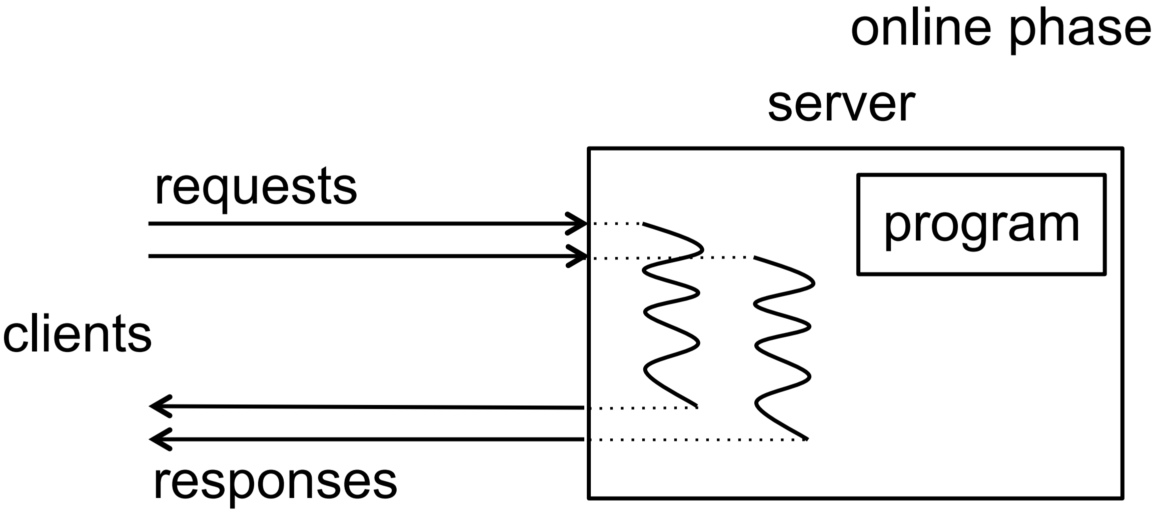


- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong ...
- Thus, Alice wants to audit the delivered responses
  - Are they derived from executing the actual application?

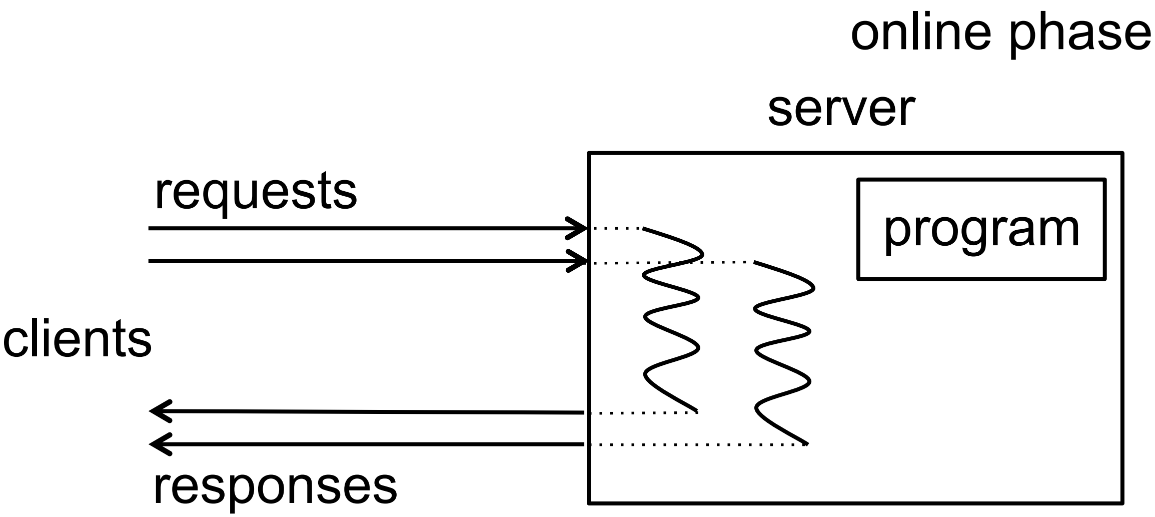
The Efficient Server Audit Problem



The Efficient Server Audit Problem

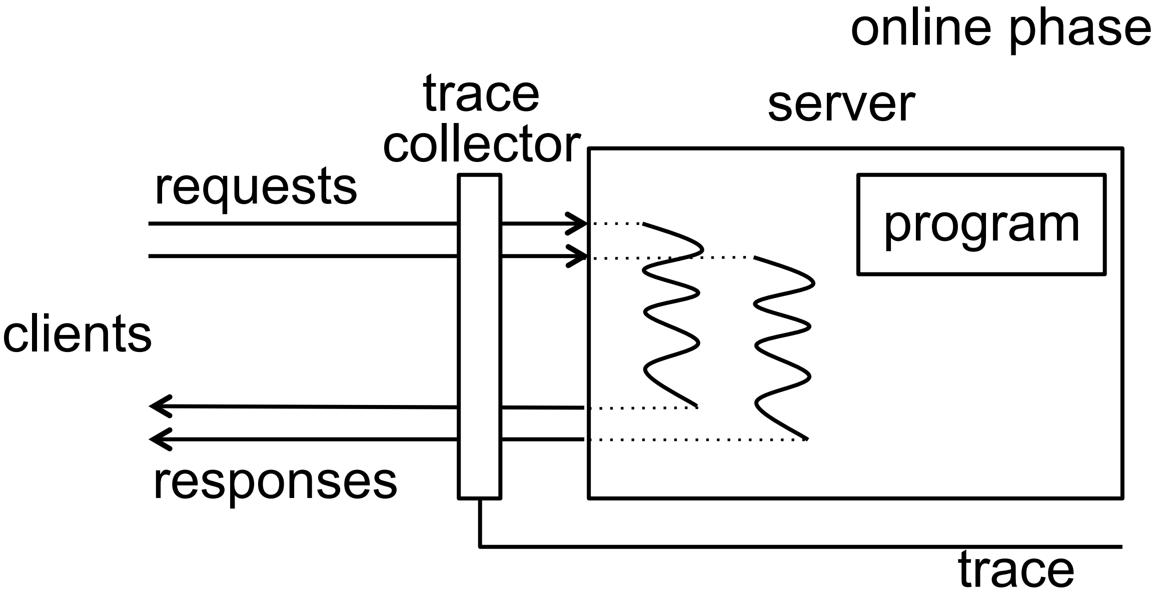


The Efficient Server Audit Problem

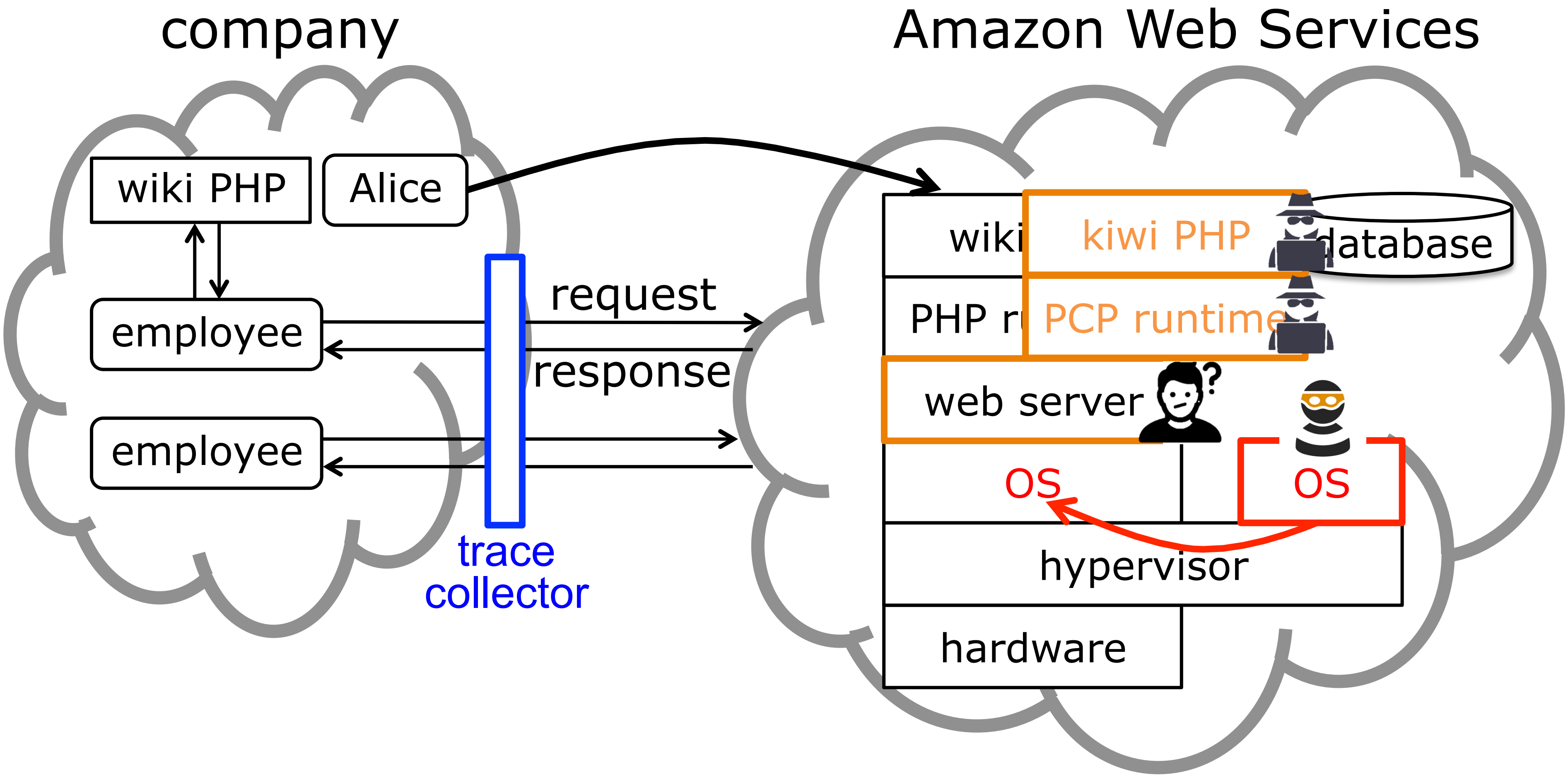


1. server is untrusted; can respond arbitrarily
2. server is concurrent

The Efficient Server Audit Problem



1. server is untrusted; can respond arbitrarily
2. server is concurrent



company

Amazon Web Services

wiki PHP

Alice

employee

employee

request

response

trace
collector

wiki

kiwi PHP

database

PHP r

PCP runtime

web server

?

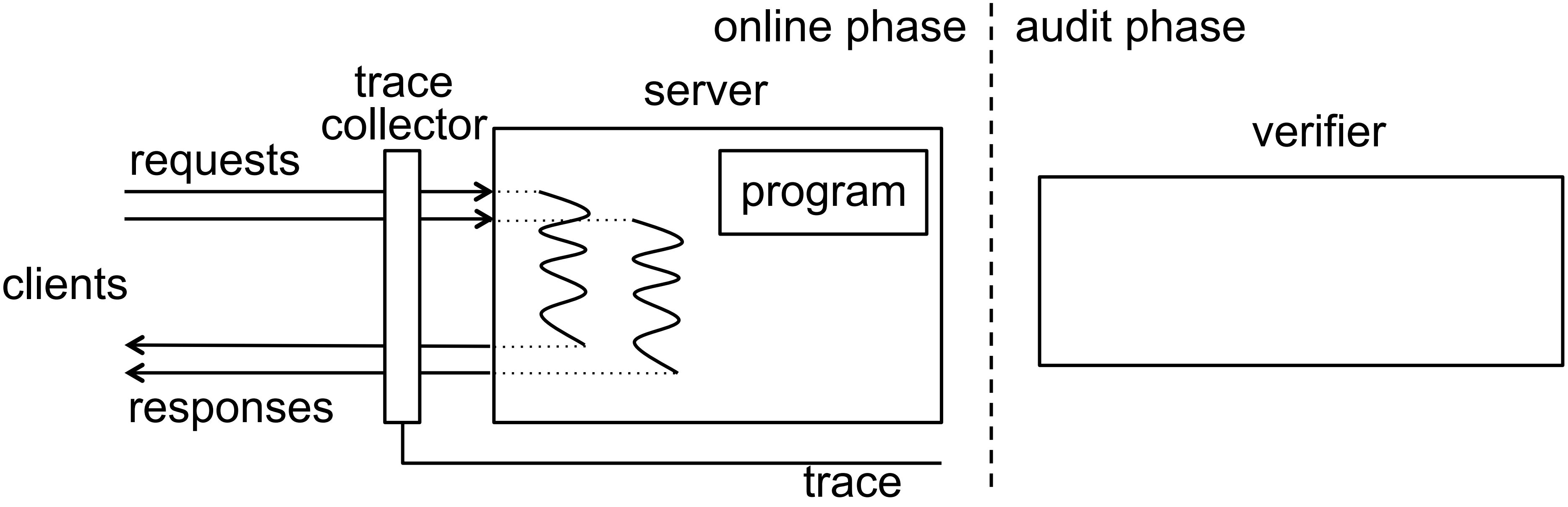
OS

OS

hypervisor

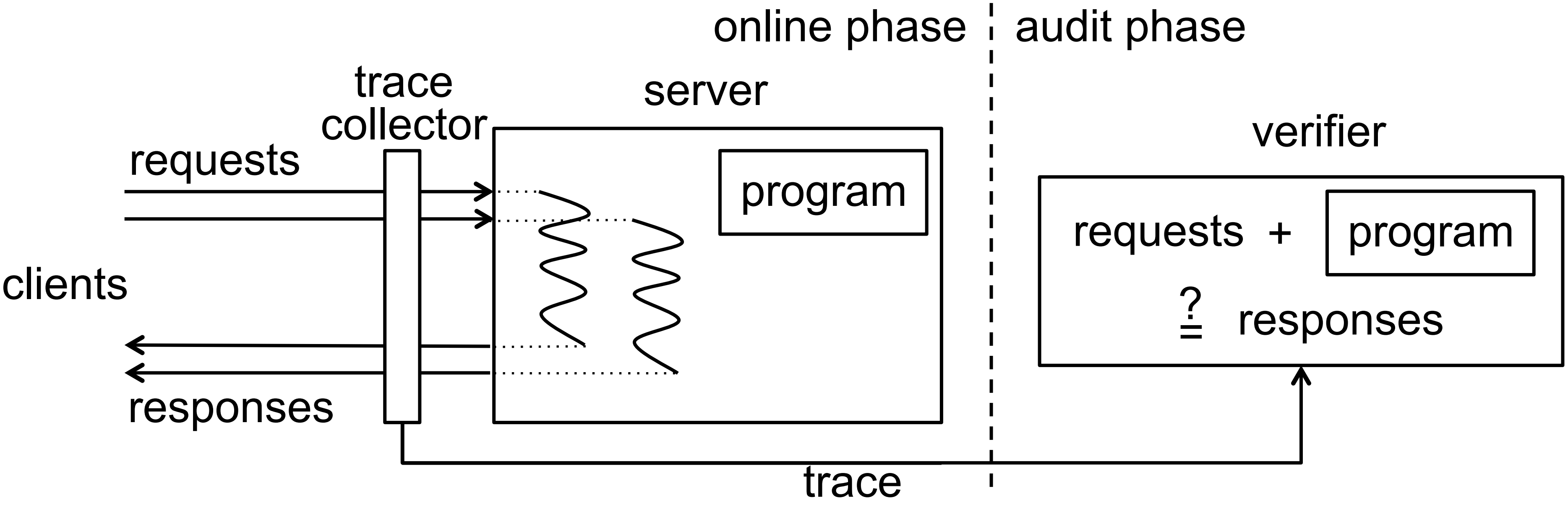
hardware

The Efficient Server Audit Problem



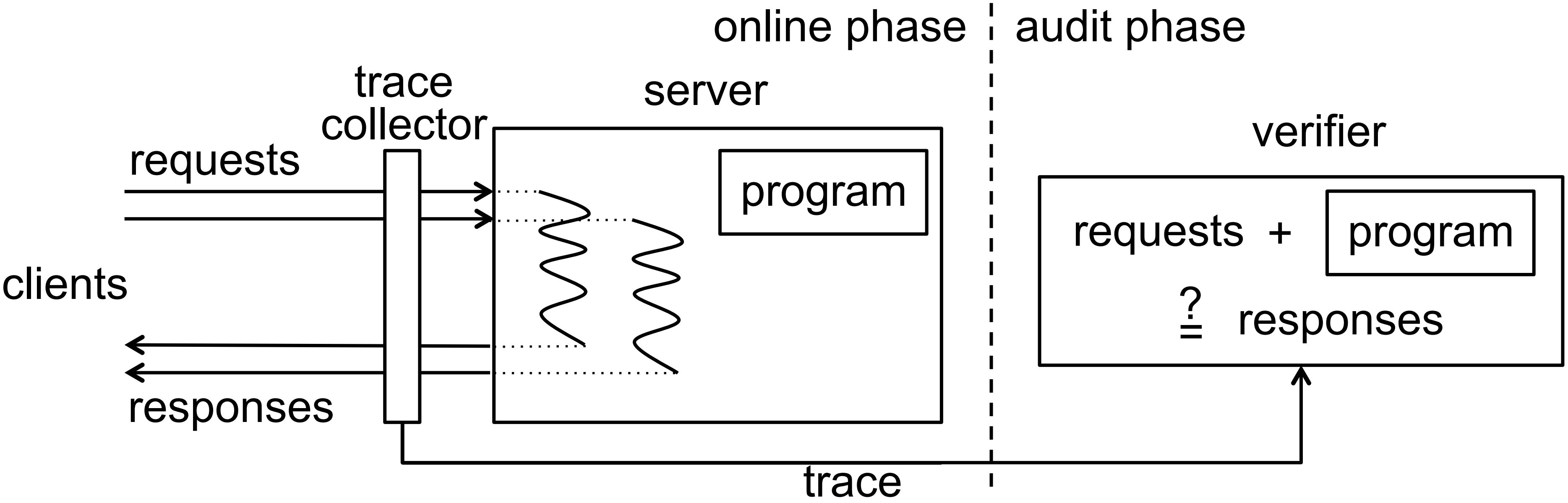
1. server is untrusted; can respond arbitrarily
2. server is concurrent

The Efficient Server Audit Problem



1. server is untrusted; can respond arbitrarily
2. server is concurrent

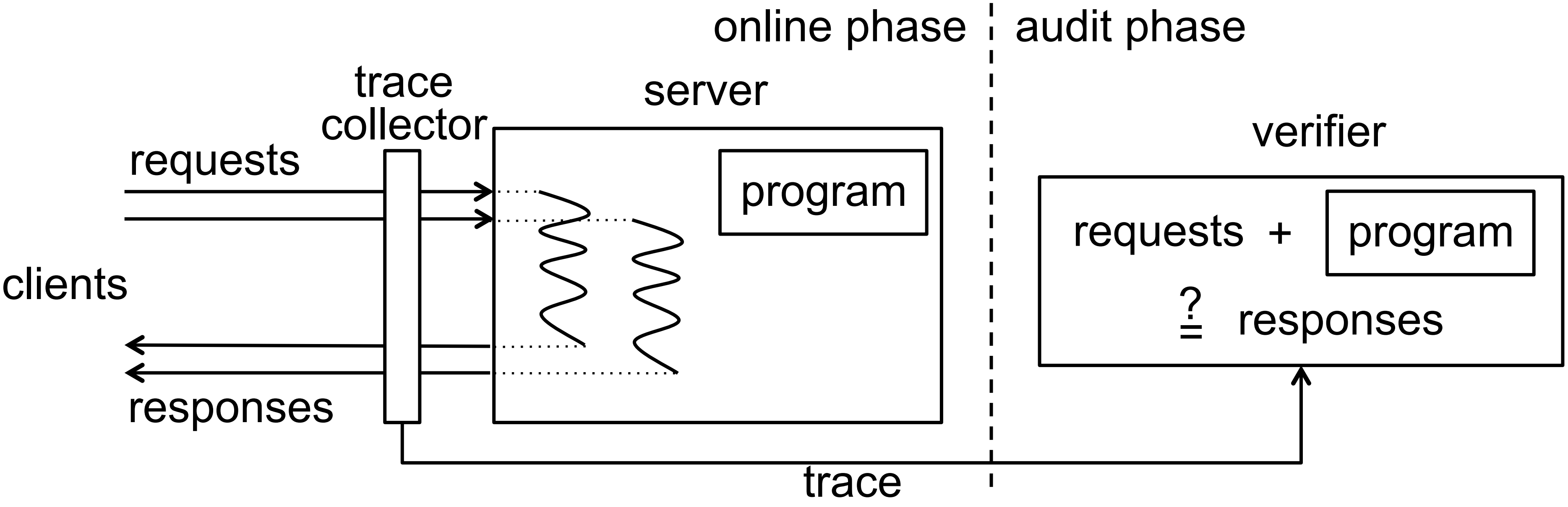
The Efficient Server Audit Problem



1. server is untrusted; can respond arbitrarily
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low; legacy applications supported

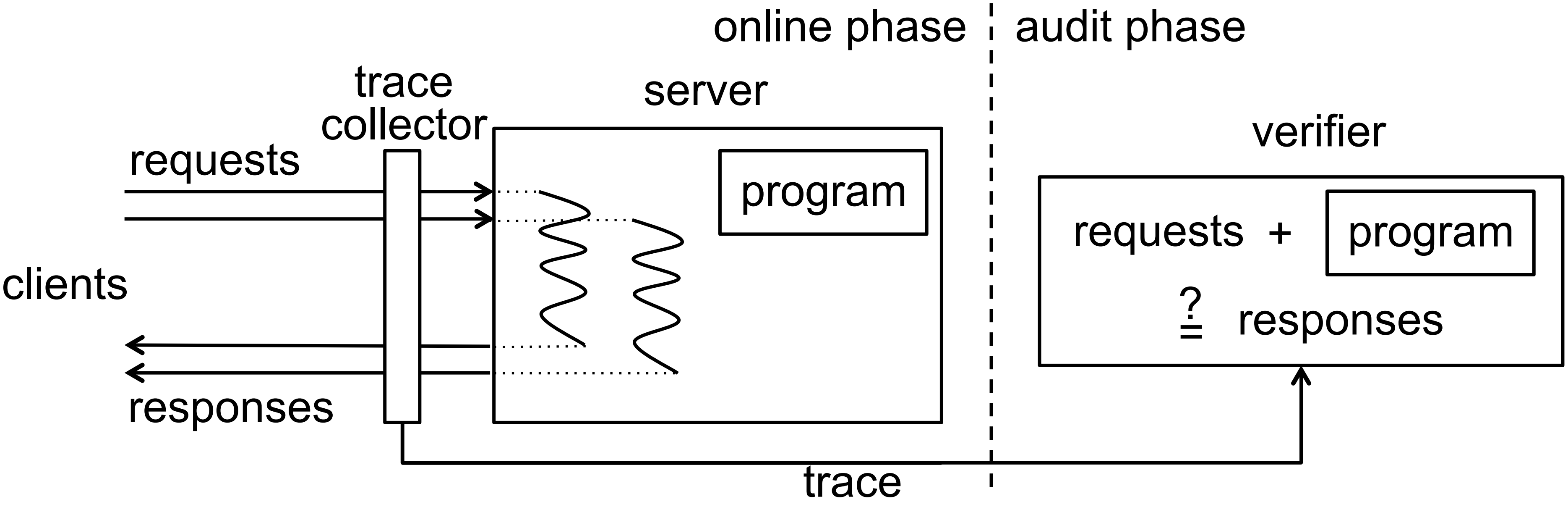
The Efficient Server Audit Problem

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



The Efficient Server Audit Problem

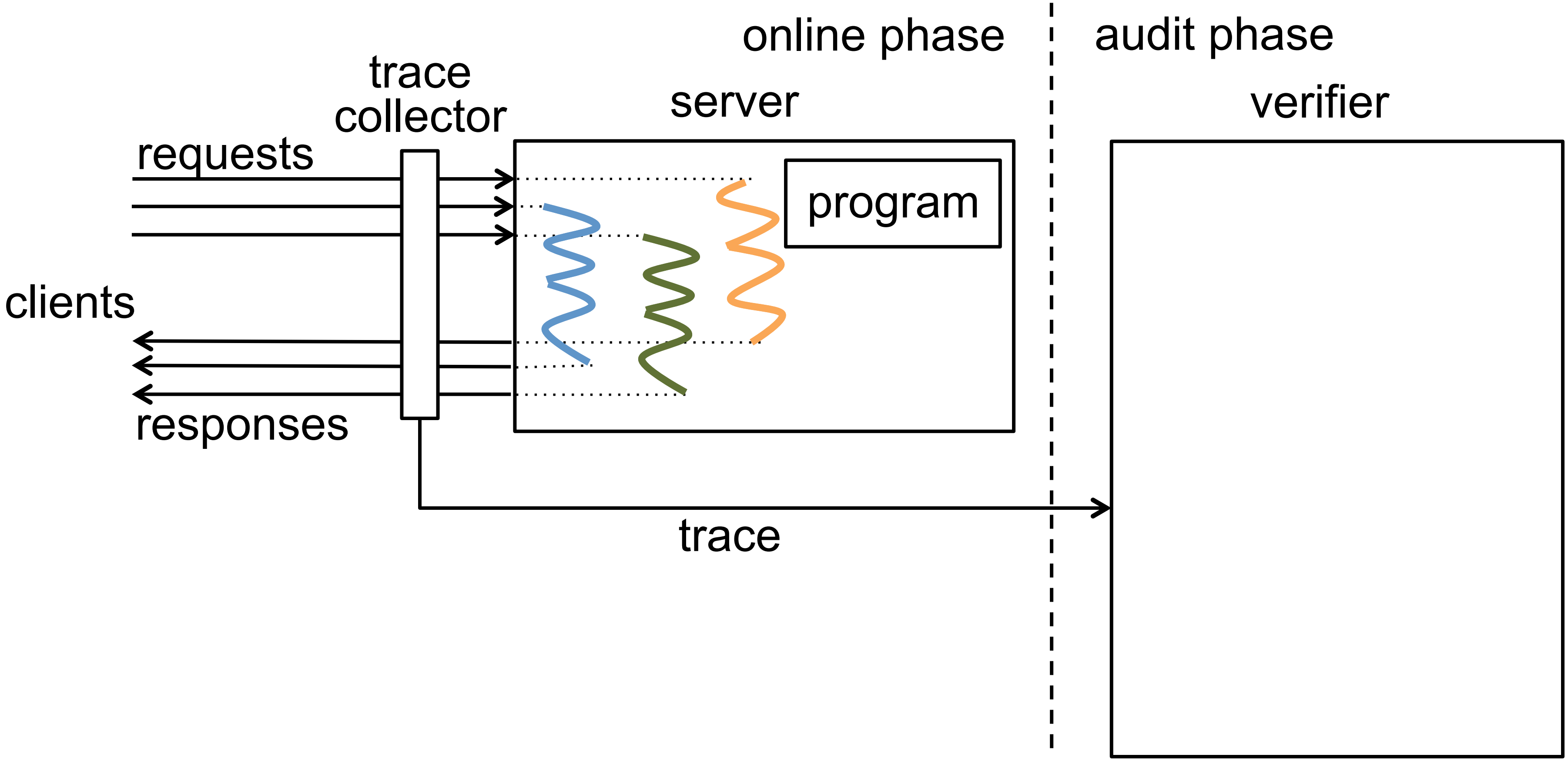
1. server is untrusted…
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low…



- Combination of these four is a new problem.
- Execution integrity is complementary to program verification.

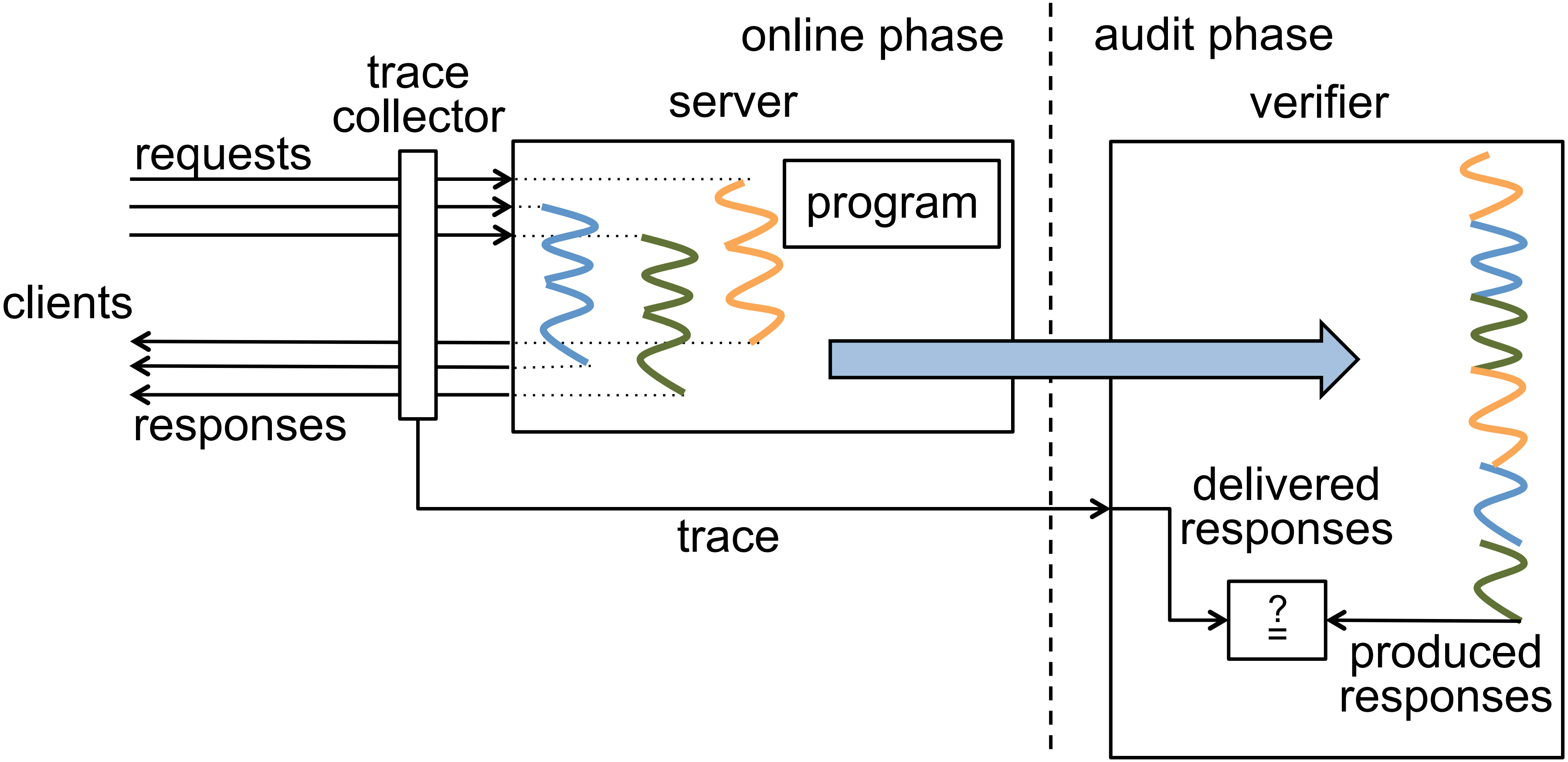
What about naive re-execution?

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



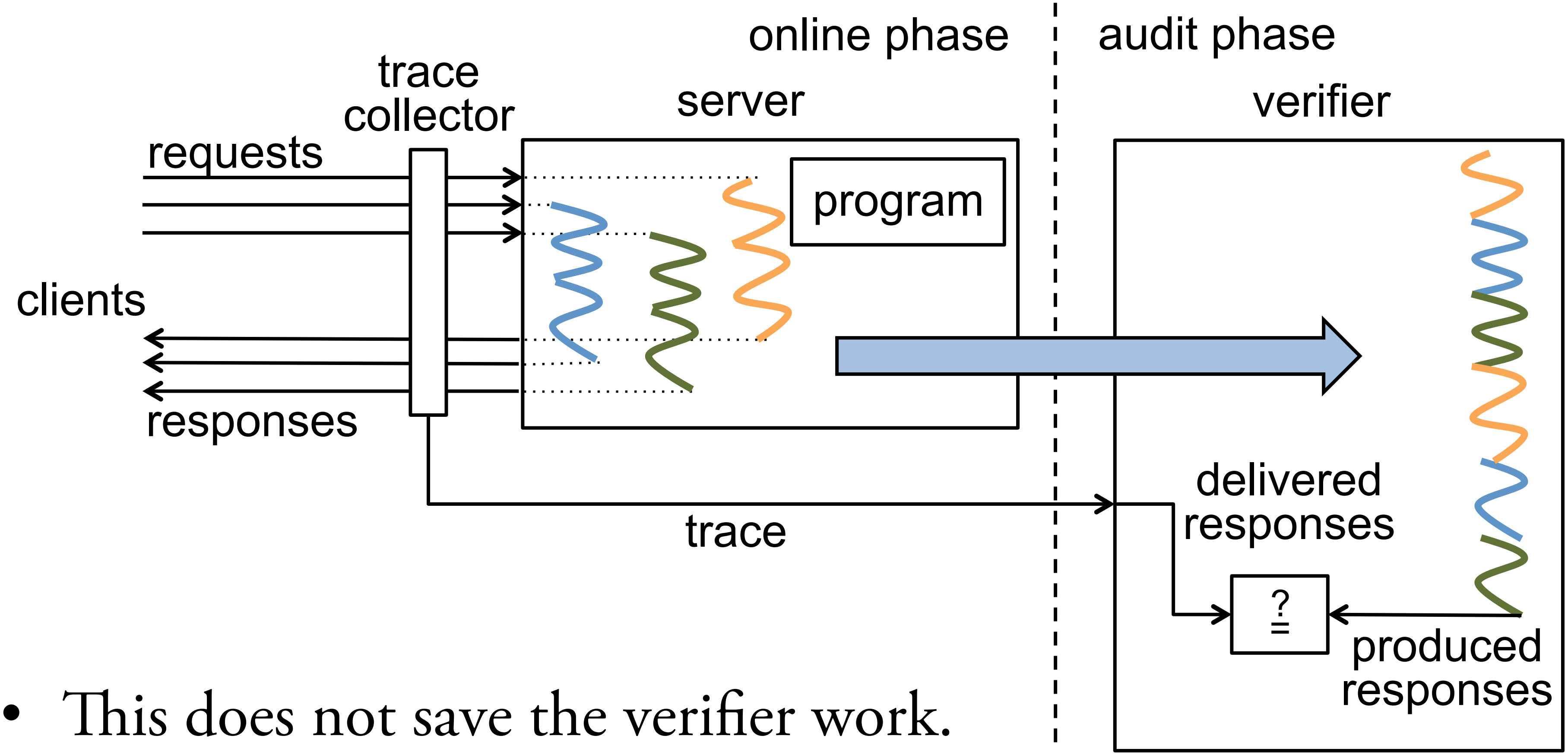
What about naive re-execution?

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



What about naive re-execution?

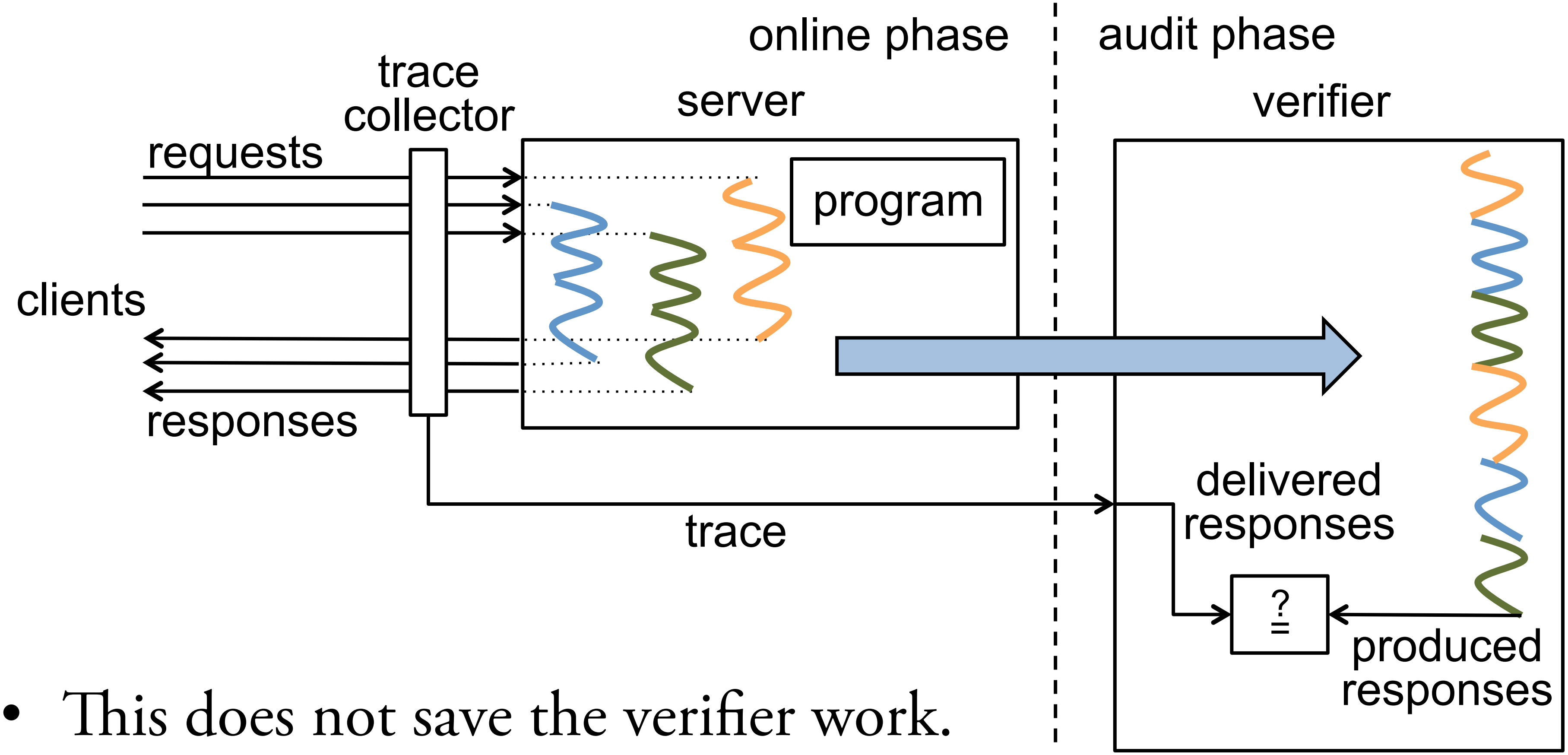
✓ 1. server is untrusted…
✓ 2. server is concurrent
✗ 3. verifier is weaker than server
✓ 4. server overhead is low…



- 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.
- Instead, we will accelerate re-execution.

Rest of the talk

1. How does the verifier accelerate re-execution?

(these two are in tension)

2. Why are shared objects (such as DBs) challenging?

3. Does our implementation for PHP perform well?

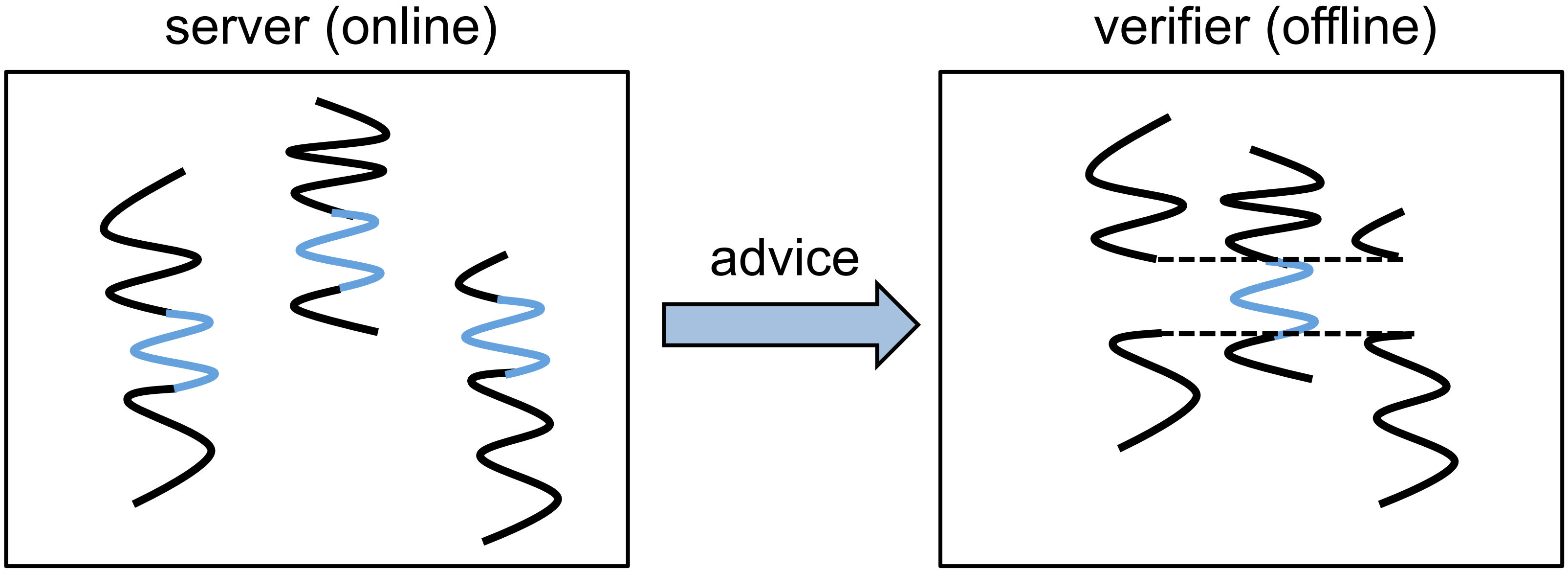
Rest of the talk

→ 1. How does the verifier accelerate re-execution?

2. Why are shared objects (such as DBs) challenging?

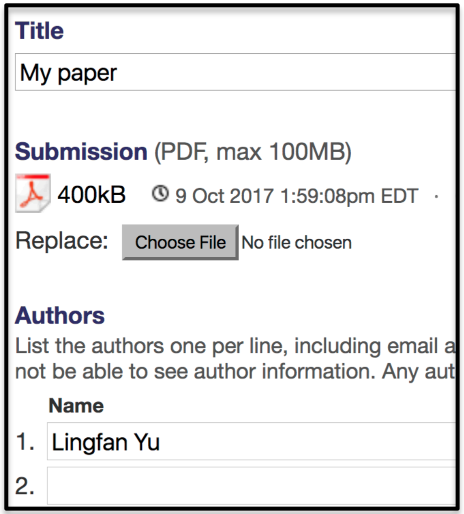
3. Does our implementation for PHP perform well?

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



- Deduplicate computation across requests

Poirot's observation: repeated computation



Title

My paper

Submission (PDF, max 100MB)

400kB 9 Oct 2017 1:59:08pm EDT ·

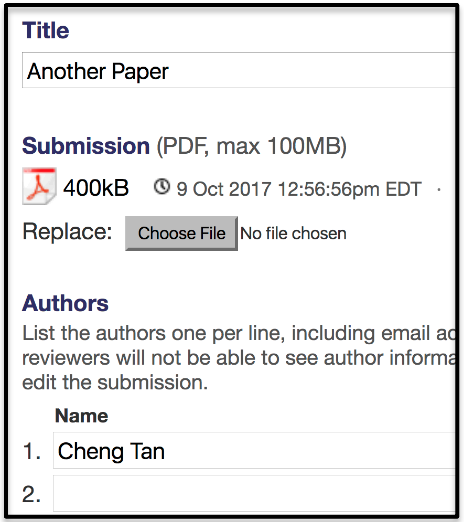
Replace: No file chosen

Authors

List the authors one per line, including email a
not be able to see author information. Any aut

Name

1. Lingfan Yu
2.



Title

Another Paper

Submission (PDF, max 100MB)

400kB 9 Oct 2017 12:56:56pm EDT ·

Replace: No file chosen

Authors

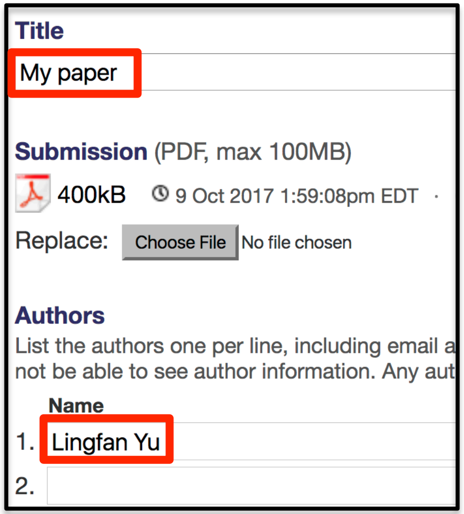
List the authors one per line, including email ac
reviewers will not be able to see author informa
edit the submission.

Name

1. Cheng Tan
2.

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

Poirot's observation: repeated computation



Title

My paper

Submission (PDF, max 100MB)

400kB 9 Oct 2017 1:59:08pm EDT ·

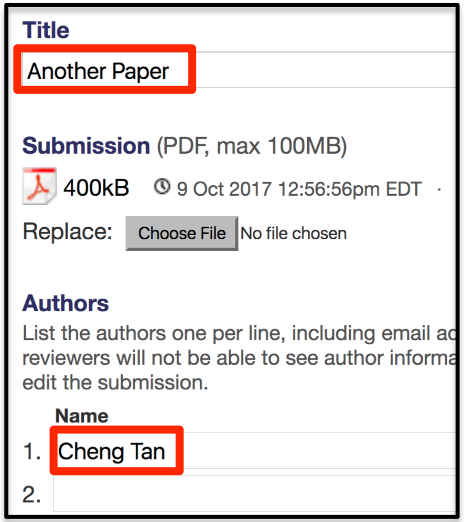
Replace: No file chosen

Authors

List the authors one per line, including email a
not be able to see author information. Any aut

Name

1. Lingfan Yu
2.



Title

Another Paper

Submission (PDF, max 100MB)

400kB 9 Oct 2017 12:56:56pm EDT ·

Replace: No file chosen

Authors

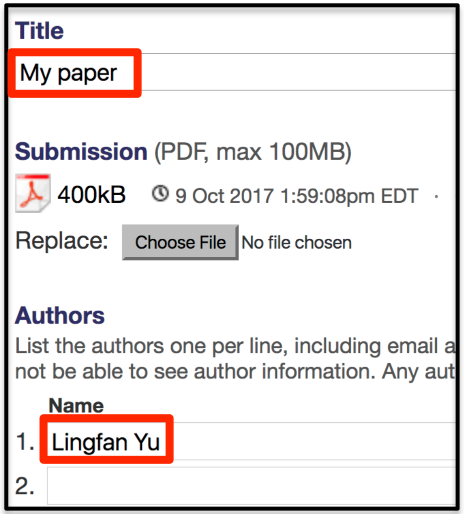
List the authors one per line, including email ac
reviewers will not be able to see author informa
edit the submission.

Name

1. Cheng Tan
2.

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

Poirot's observation: repeated computation



Title

My paper

Submission (PDF, max 100MB)

400kB 9 Oct 2017 1:59:08pm EDT ·

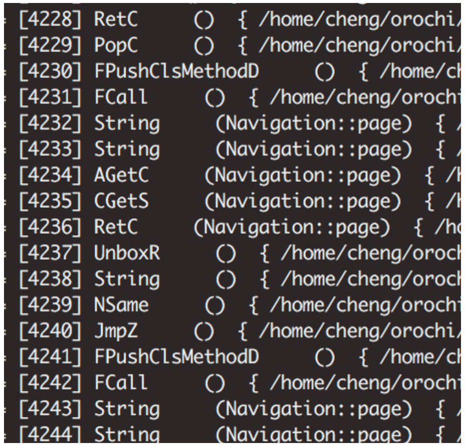
Replace: Choose File No file chosen

Authors

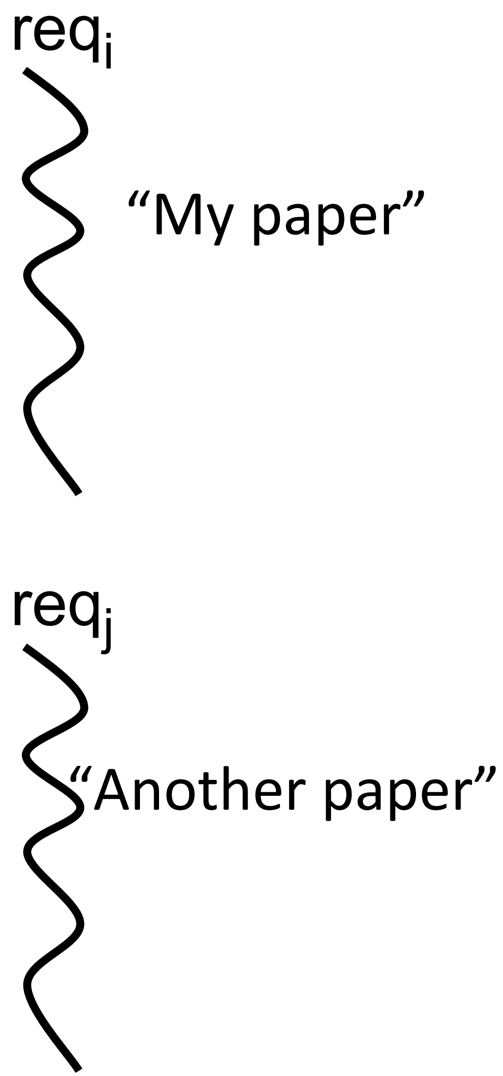
List the authors one per line, including email a
not be able to see author information. Any aut

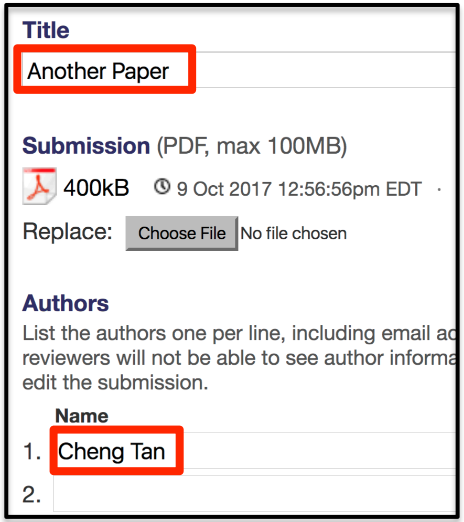
Name

1. Lingfan Yu
2.



```
[4228] RetC       ()  { /home/cheng/orochi/
[4229] PopC       ()  { /home/cheng/orochi/
[4230] FPushClsMethodD      ()  { /home/ch
[4231] FCall       ()  { /home/cheng/orochi
[4232] String       (Navigation::page)  { /
[4233] String       (Navigation::page)  { /
[4234] AGetC      (Navigation::page)  { /h
[4235] CGetS      (Navigation::page)  { /h
[4236] RetC      (Navigation::page)  { /ho
[4237] UnboxR       ()  { /home/cheng/oroch
[4238] String       ()  { /home/cheng/oroch
[4239] NSame       ()  { /home/cheng/orochi
[4240] JmpZ       ()  { /home/cheng/orochi/
[4241] FPushClsMethodD      ()  { /home/ch
[4242] FCall       ()  { /home/cheng/orochi
[4243] String       (Navigation::page)  { /
[4244] String       (Navigation::page)  { /
```





Title

Another Paper

Submission (PDF, max 100MB)

400kB 9 Oct 2017 12:56:56pm EDT ·

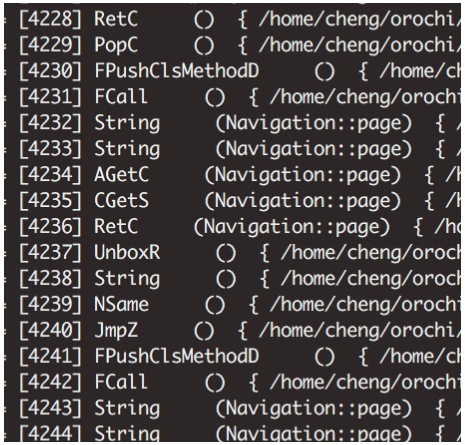
Replace: Choose File No file chosen

Authors

List the authors one per line, including email ad
reviewers will not be able to see author informa
edit the submission.

Name

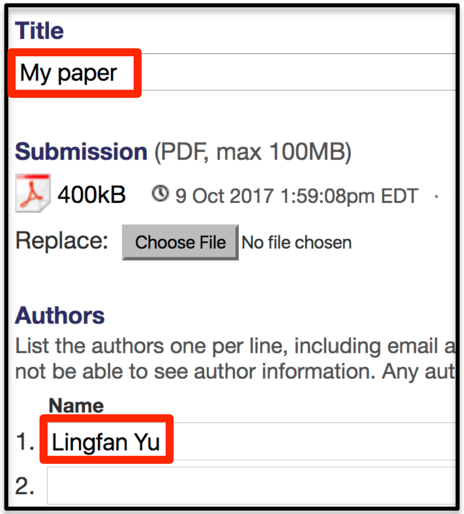
1. Cheng Tan
2.



```
[4228] RetC       ()  { /home/cheng/orochi/
[4229] PopC       ()  { /home/cheng/orochi/
[4230] FPushClsMethodD      ()  { /home/ch
[4231] FCall       ()  { /home/cheng/orochi
[4232] String       (Navigation::page)  { /
[4233] String       (Navigation::page)  { /
[4234] AGetC      (Navigation::page)  { /h
[4235] CGetS      (Navigation::page)  { /h
[4236] RetC      (Navigation::page)  { /ho
[4237] UnboxR       ()  { /home/cheng/oroch
[4238] String       ()  { /home/cheng/oroch
[4239] NSame       ()  { /home/cheng/orochi
[4240] JmpZ       ()  { /home/cheng/orochi/
[4241] FPushClsMethodD      ()  { /home/ch
[4242] FCall       ()  { /home/cheng/orochi
[4243] String       (Navigation::page)  { /
[4244] String       (Navigation::page)  { /
```

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

Poirot's observation: repeated computation



Title

My paper

Submission (PDF, max 100MB)

400kB 9 Oct 2017 1:59:08pm EDT ·

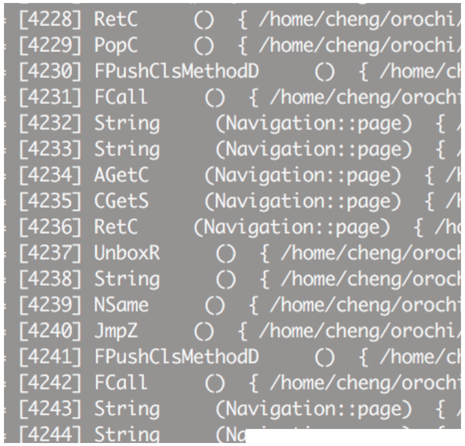
Replace: Choose File No file chosen

Authors

List the authors one per line, including email a
not be able to see author information. Any aut

Name

1. Lingfan Yu
2.

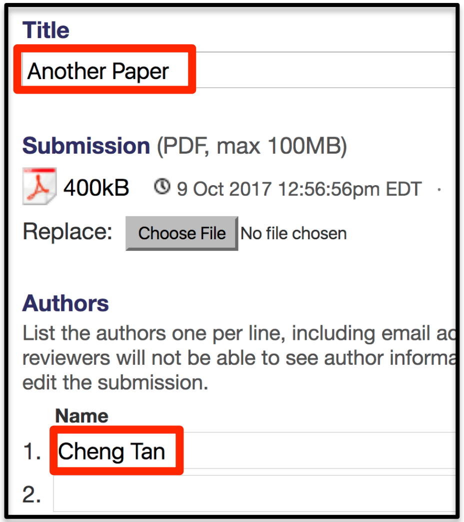


```
[4228] RetC      ()  { /home/cheng/orochi
[4229] PopC      ()  { /home/cheng/orochi
[4230] FPushClsMethodD      ()  { /home/ch
[4231] FCall      ()  { /home/cheng/orochi
[4232] String      (Navigation::page)  { /
[4233] String      (Navigation::page)  { /
[4234] AGetC     (Navigation::page)  { /h
[4235] CGetS     (Navigation::page)  { /h
[4236] RetC     (Navigation::page)  { /ho
[4237] UnboxR      ()  { /home/cheng/oroch
[4238] String      ()  { /home/cheng/oroch
[4239] NSame      ()  { /home/cheng/orochi
[4240] JmpZ      ()  { /home/cheng/orochi
[4241] FPushClsMethodD      ()  { /home/ch
[4242] FCall      ()  { /home/cheng/orochi
[4243] String      (Navigation::page)  { /
[4244] String      (Na
```

req_i

"My paper"

requires trusting the advice



Title

Another Paper

Submission (PDF, max 100MB)

400kB 9 Oct 2017 12:56:56pm EDT ·

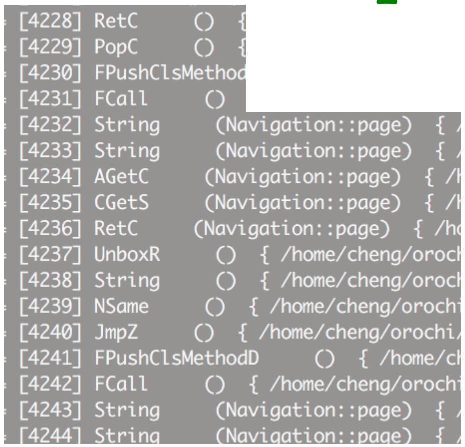
Replace: Choose File No file chosen

Authors

List the authors one per line, including email ac
reviewers will not be able to see author informa
edit the submission.

Name

1. Cheng Tan
2.

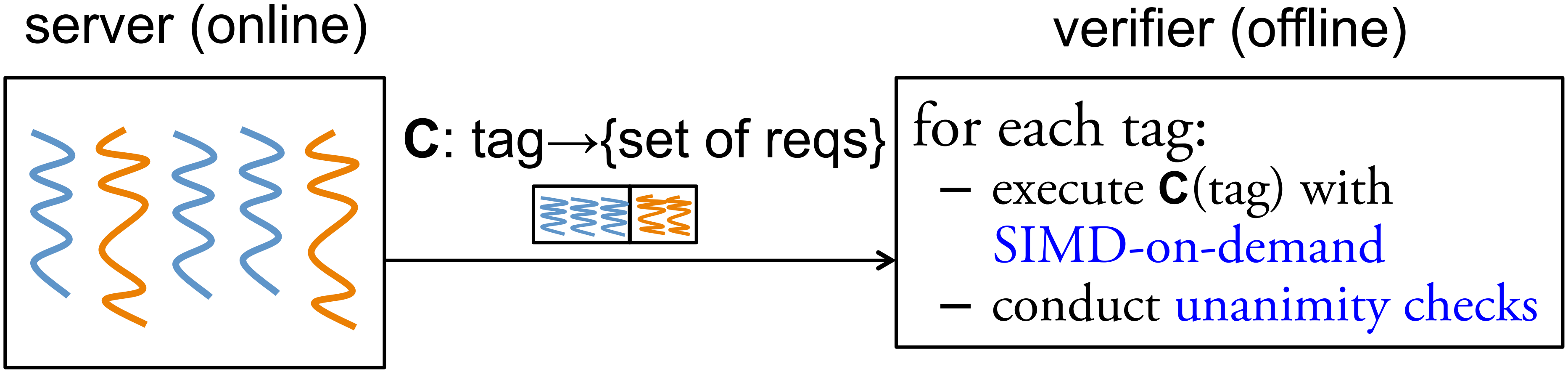


```
[4228] RetC      ()  {
[4229] PopC      ()  {
[4230] FPushClsMethod
[4231] FCall      ()
[4232] String      (Navigation::page)  { /
[4233] String      (Navigation::page)  { /
[4234] AGetC     (Navigation::page)  { /h
[4235] CGetS     (Navigation::page)  { /h
[4236] RetC     (Navigation::page)  { /ho
[4237] UnboxR      ()  { /home/cheng/oroch
[4238] String      ()  { /home/cheng/oroch
[4239] NSame      ()  { /home/cheng/orochi
[4240] JmpZ      ()  { /home/cheng/orochi
[4241] FPushClsMethodD      ()  { /home/ch
[4242] FCall      ()  { /home/cheng/orochi
[4243] String      (Navigation::page)  { /
[4244] String      (Navigation::page)  { /
```

"Another paper"

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

We accelerate re-execution without trusting the server



server (online)

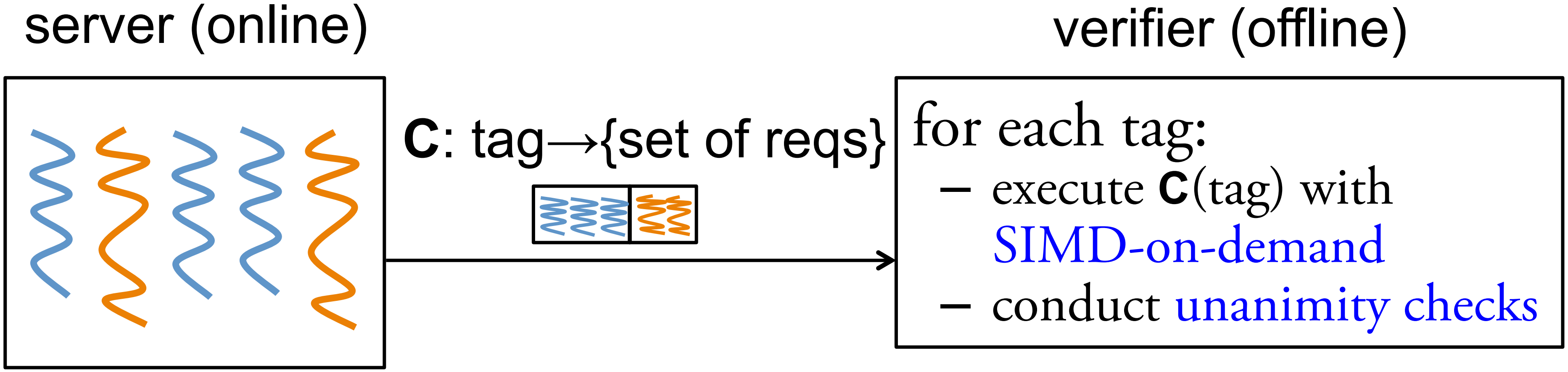
verifier (offline)

C: tag→{set of reqs}

for each tag:

- execute **C**(tag) with SIMD-on-demand
- conduct unanimity checks

We accelerate re-execution without trusting the server



server (online)

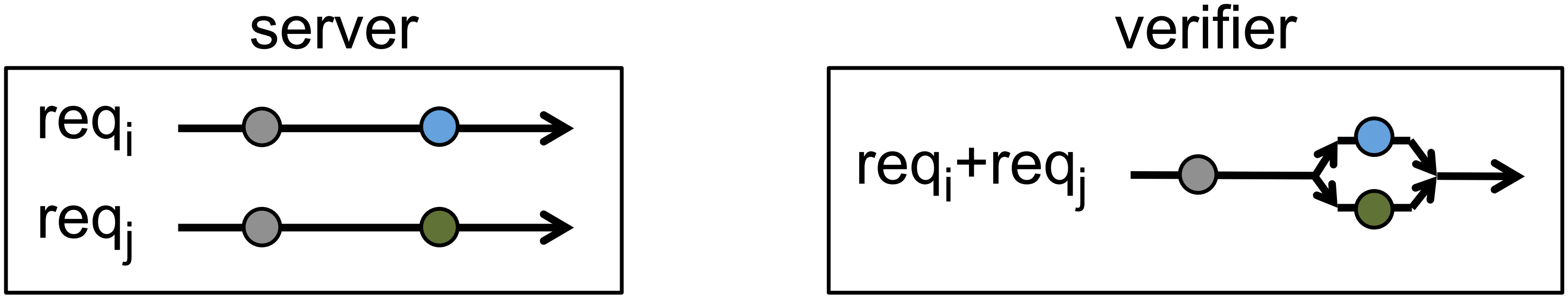
C: tag→{set of reqs}

verifier (offline)

for each tag:

- execute **C**(tag) with SIMD-on-demand
- conduct unanimity checks

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



SIMD-on-demand eliminates redundant computation

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

req$_{i:}$ a=1;b=2

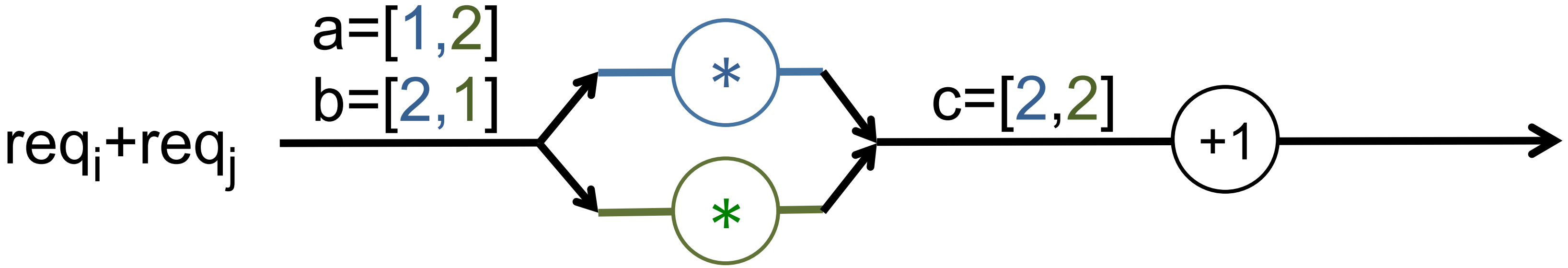
req$_{j:}$ a=2;b=1

SIMD-on-demand eliminates redundant computation

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

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

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



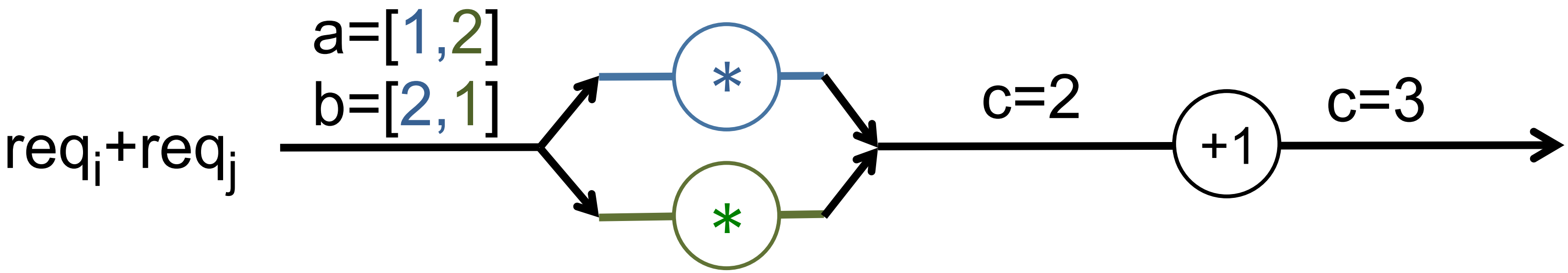
- Multi-value represents different values of the same variable.

SIMD-on-demand eliminates redundant computation

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

req$_{i:}$ a=1;b=2

req$_{j:}$ a=2;b=1



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

Recap

- Verifier re-executes in an accelerated way …

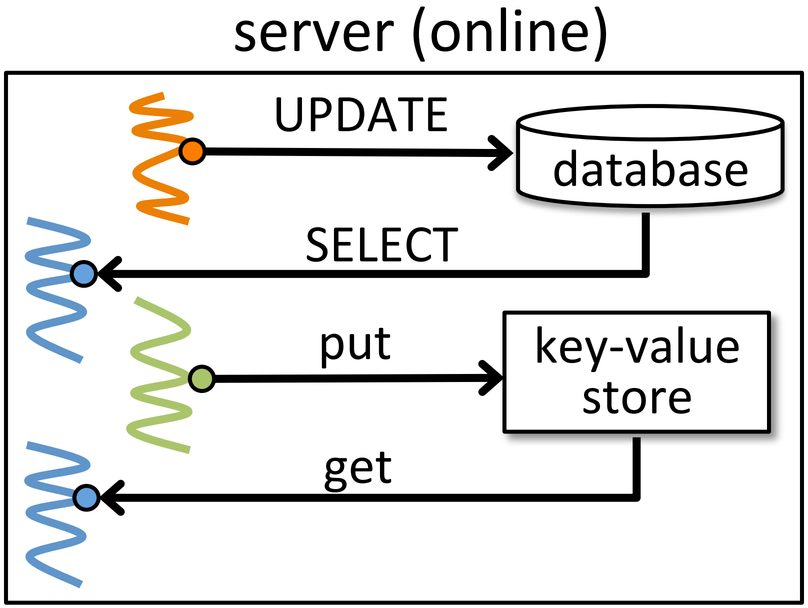
- ... by exploiting advice from the server ...

- ... without trusting that advice.

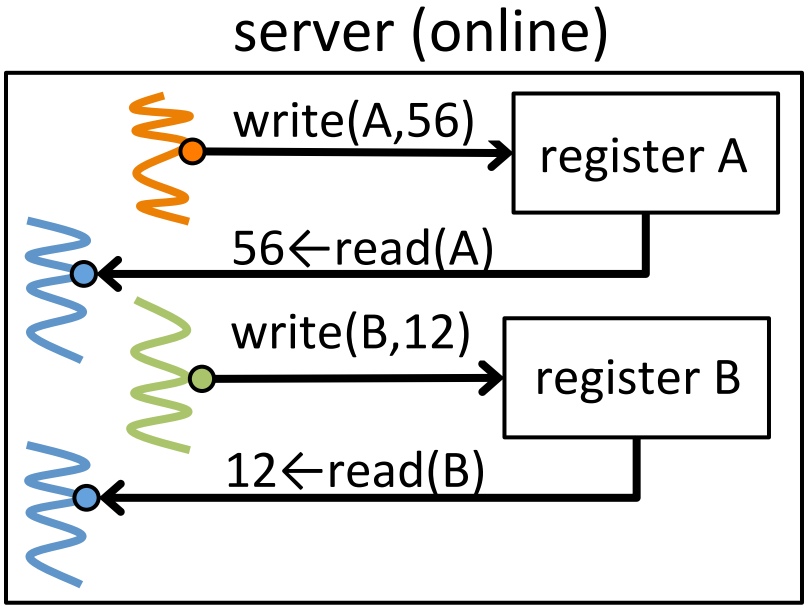
1. How does the verifier accelerate re-execution?

→ 2. Why are shared objects (such as DBs) challenging?

3. Does our implementation for PHP perform well?

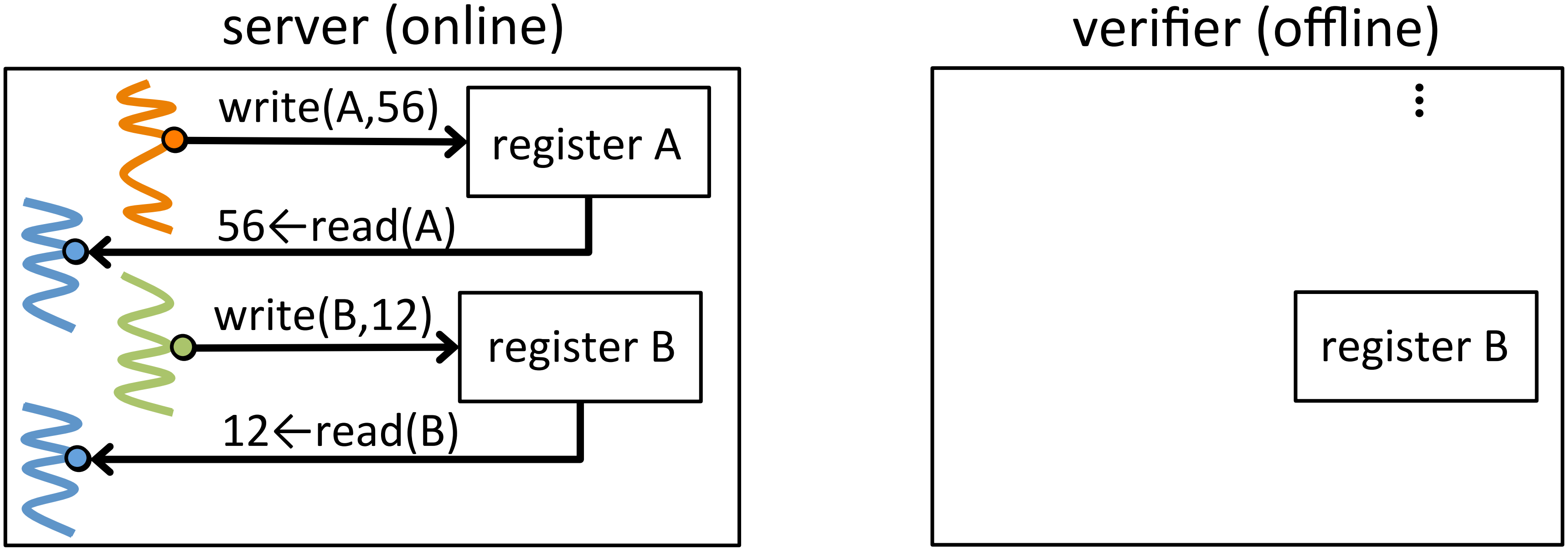


- Will try to give some intuition for the difficulties
- Solutions in the paper, rigorous proofs in tech report



- Will try to give some intuition for the difficulties
- Solutions in the paper, rigorous proofs in tech report
- For now, assume simple storage model
  - Read-write registers, named with letters

Central challenge: re-execution is out of order



server (online)

write(A,56)

register A

56←read(A)

write(B,12)

register B

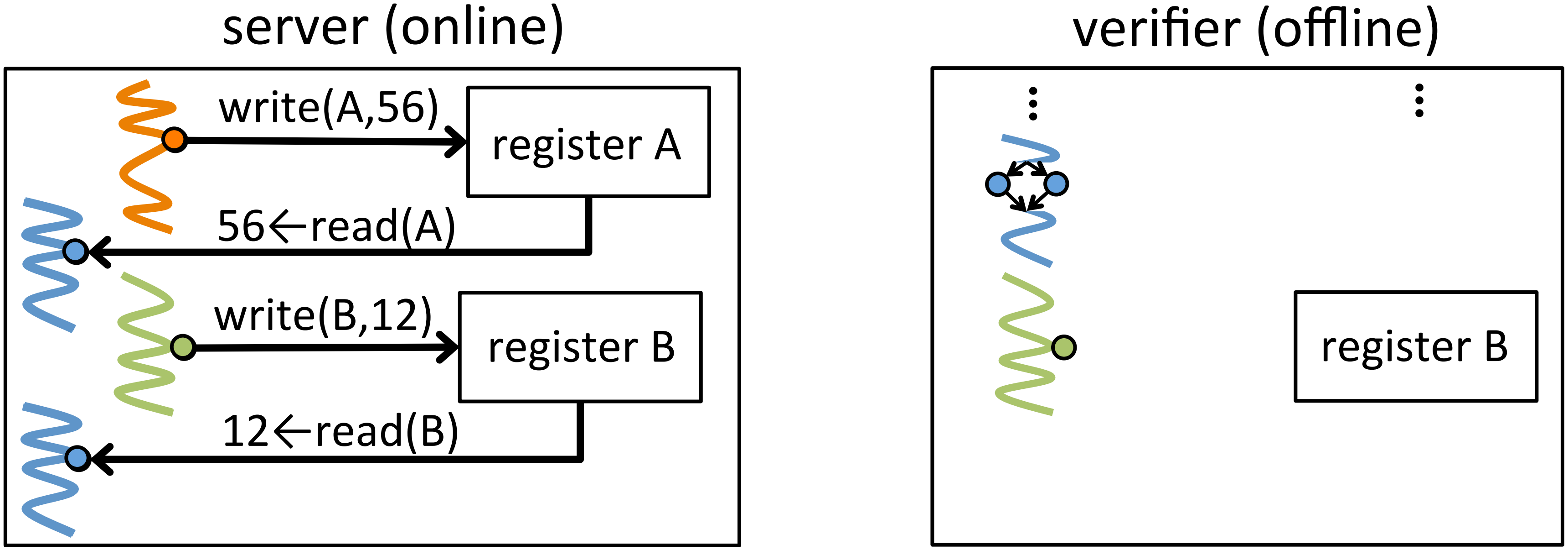
12←read(B)

verifier (offline)

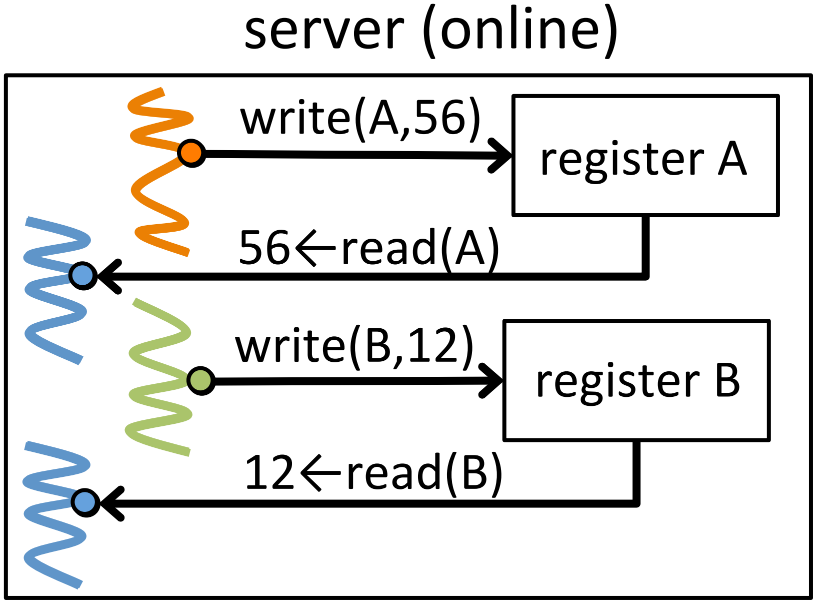
⋮

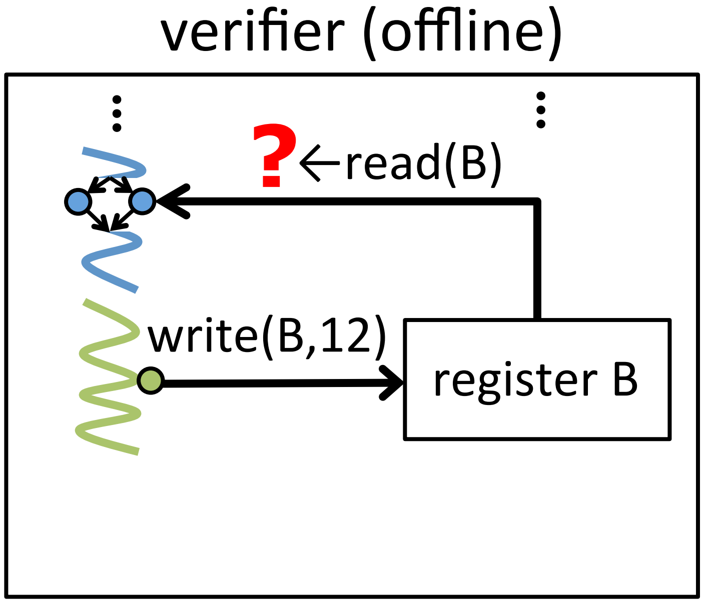
register B

Central challenge: re-execution is out of order

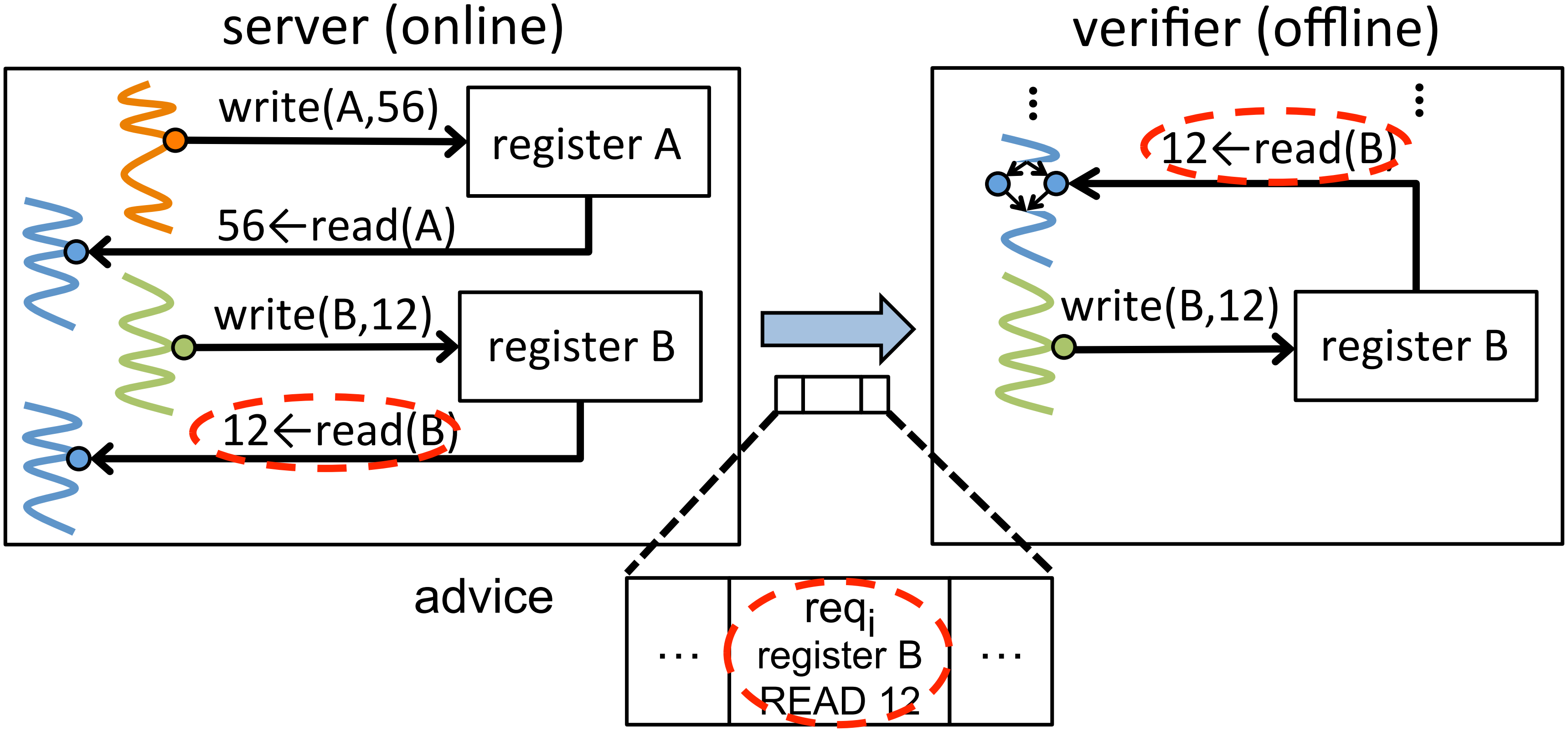


Central challenge: re-execution is out of order



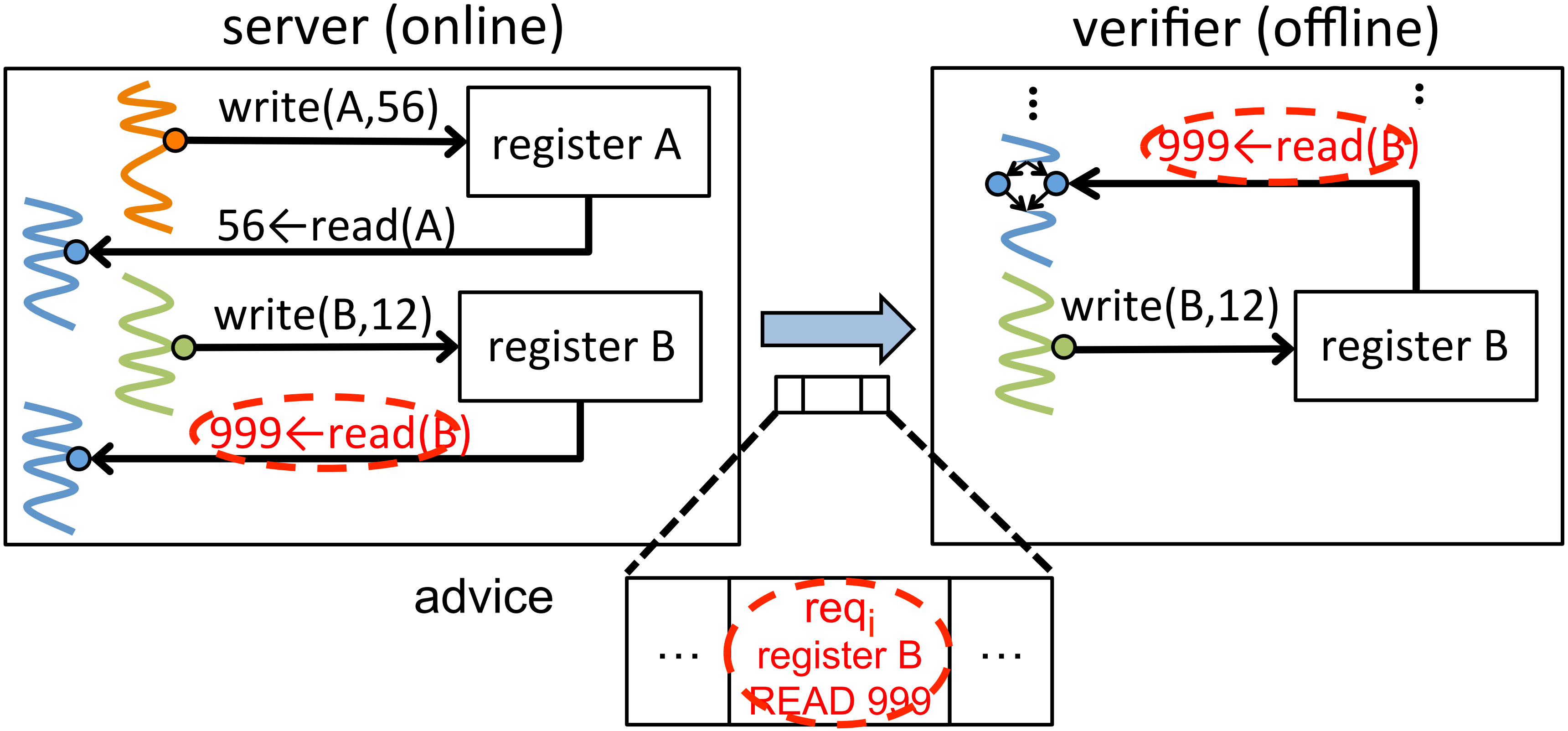


How can the verifier re-execute reads? Attempt 1:



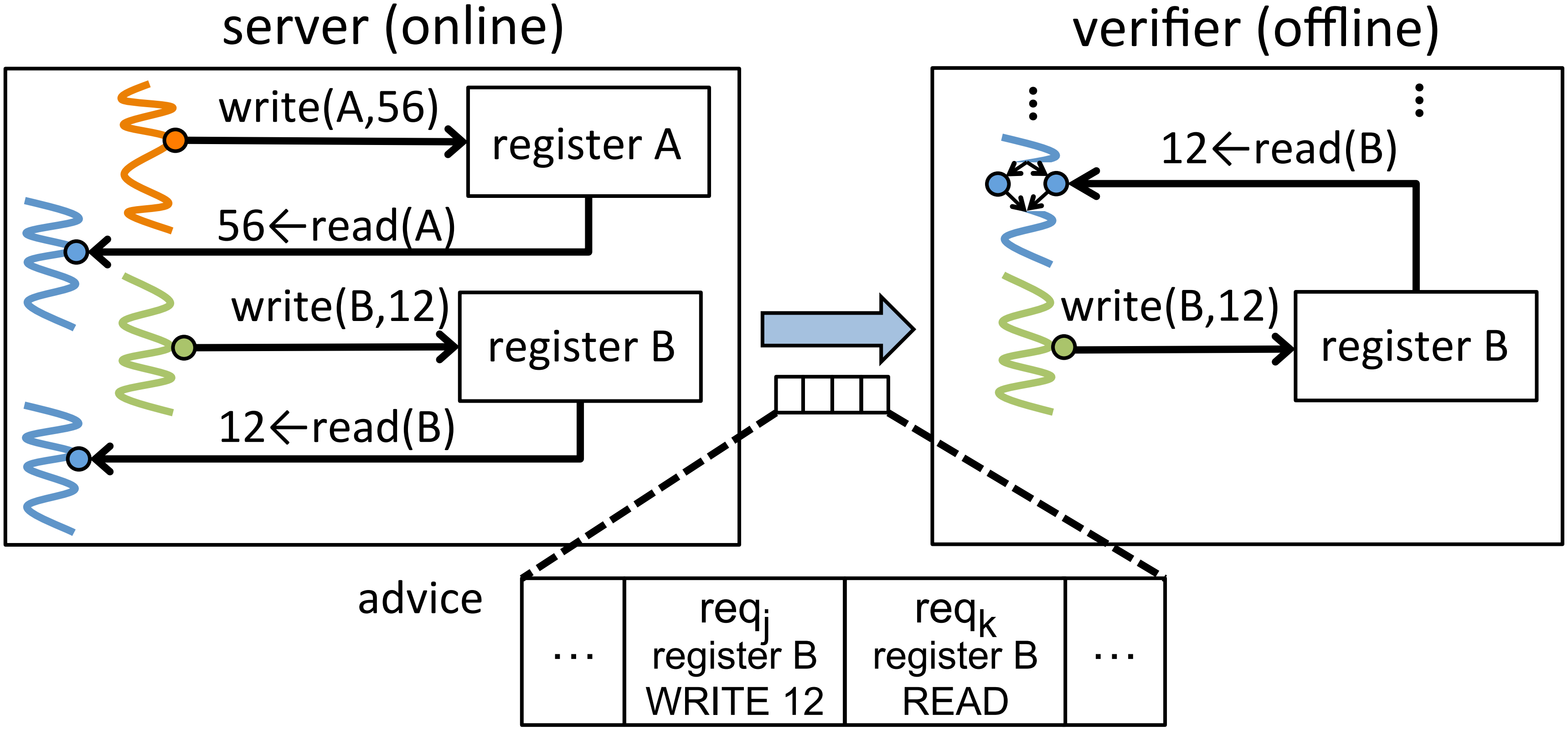
- Server logs read values; verifier supplies from log

How can the verifier re-execute reads? Attempt 1:



- Server logs read values; verifier supplies from log
- This can fool the verifier

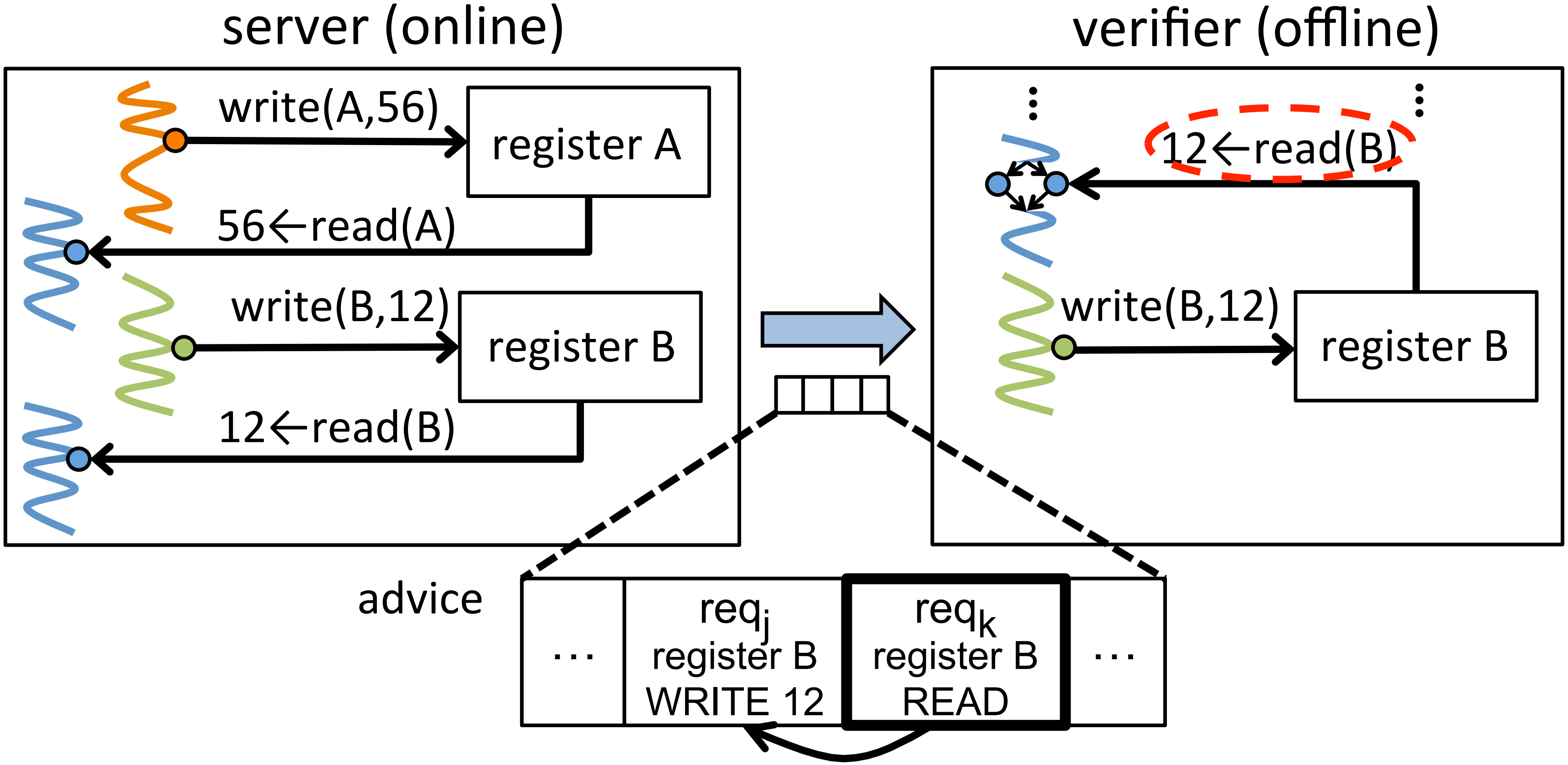
How can the verifier re-execute reads? Attempt 2:



…	req$_j$ register B WRITE 12	req$_k$ register B READ	…

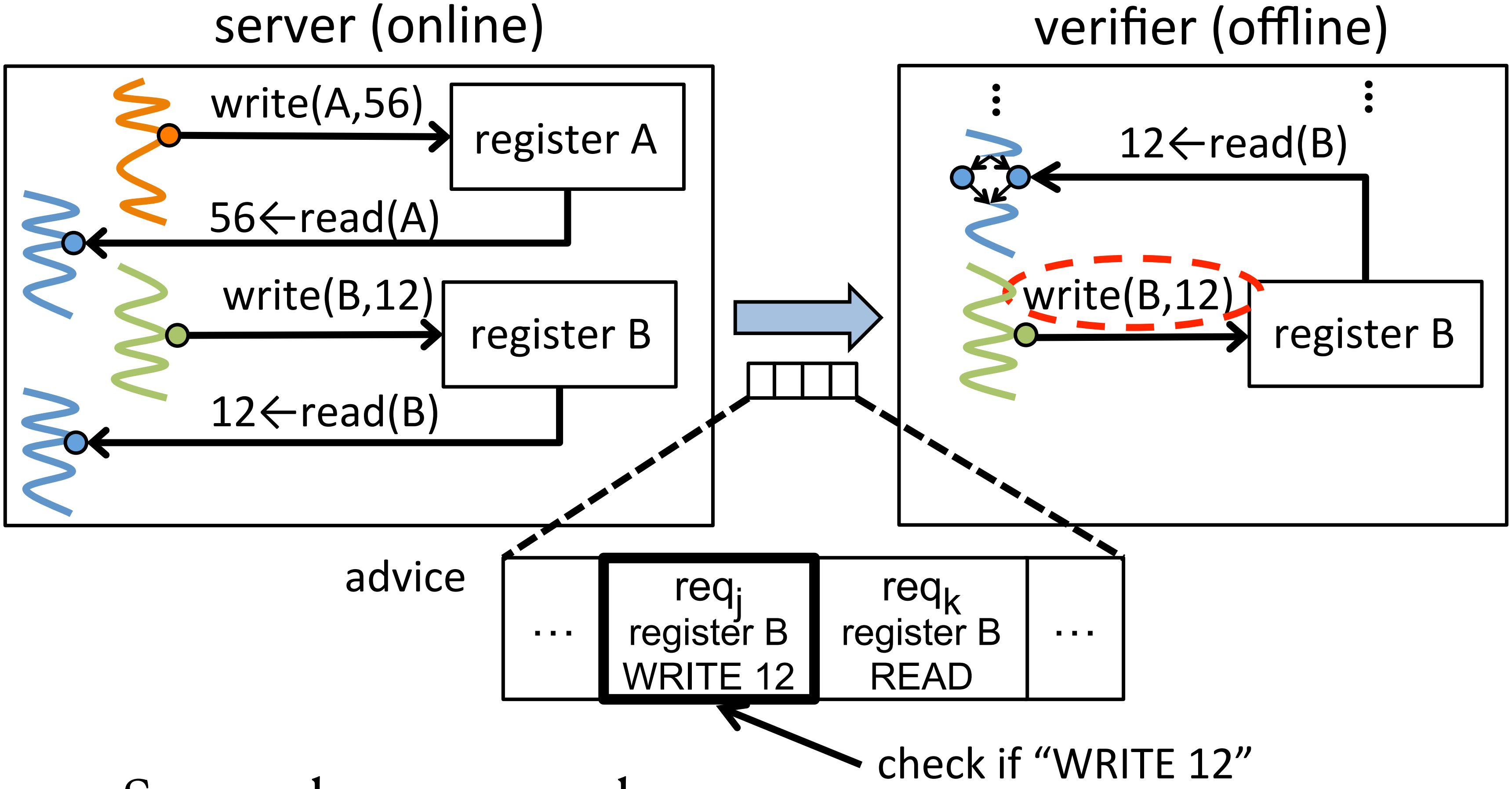
- Server: logs operands
- Verifier: simulates reads using log and checks writes

How can the verifier re-execute reads? Attempt 2:



- Server: logs operands
- Verifier: simulates reads using log and checks writes

How can the verifier re-execute reads? Attempt 2:

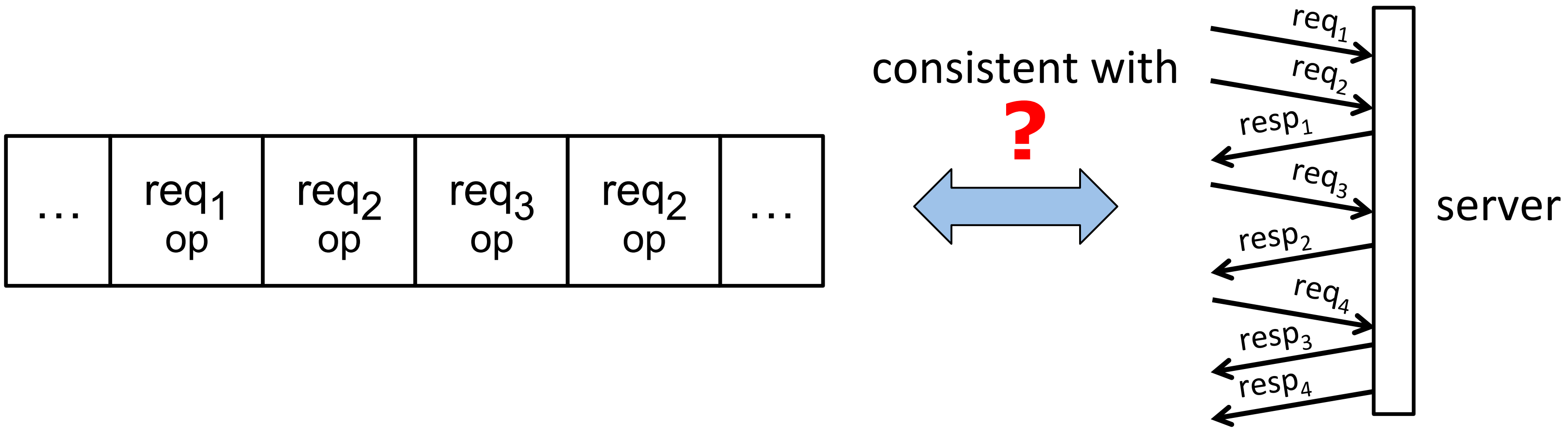


- Server: logs operands
- Verifier: simulates reads using log and checks writes

Another challenge is validating the log order

- Order in log could be nonsensical

- Verifier must check consistency of log:
  - Is log order consistent with observed request order?



…	req_1 op	req_2 op	req_3 op	req_2 op	…

consistent with

?

- This check must be efficient

1. How does the verifier accelerate re-execution?

2. Why are shared objects (such as DBs) challenging?

→ 3. Does our implementation for PHP perform well?

A built system: Orochi

- Orochi targets apps based on PHP and SQL (“LAMP”)
- Server and verifier: modified PHP runtimes
- Includes techniques for deduplicating database queries
- Details
  - Built atop HipHop VM (HHVM)
  - 20K lines of C++, PHP, Bash, Python

Evaluation questions

- Is auditing efficient for the verifier?

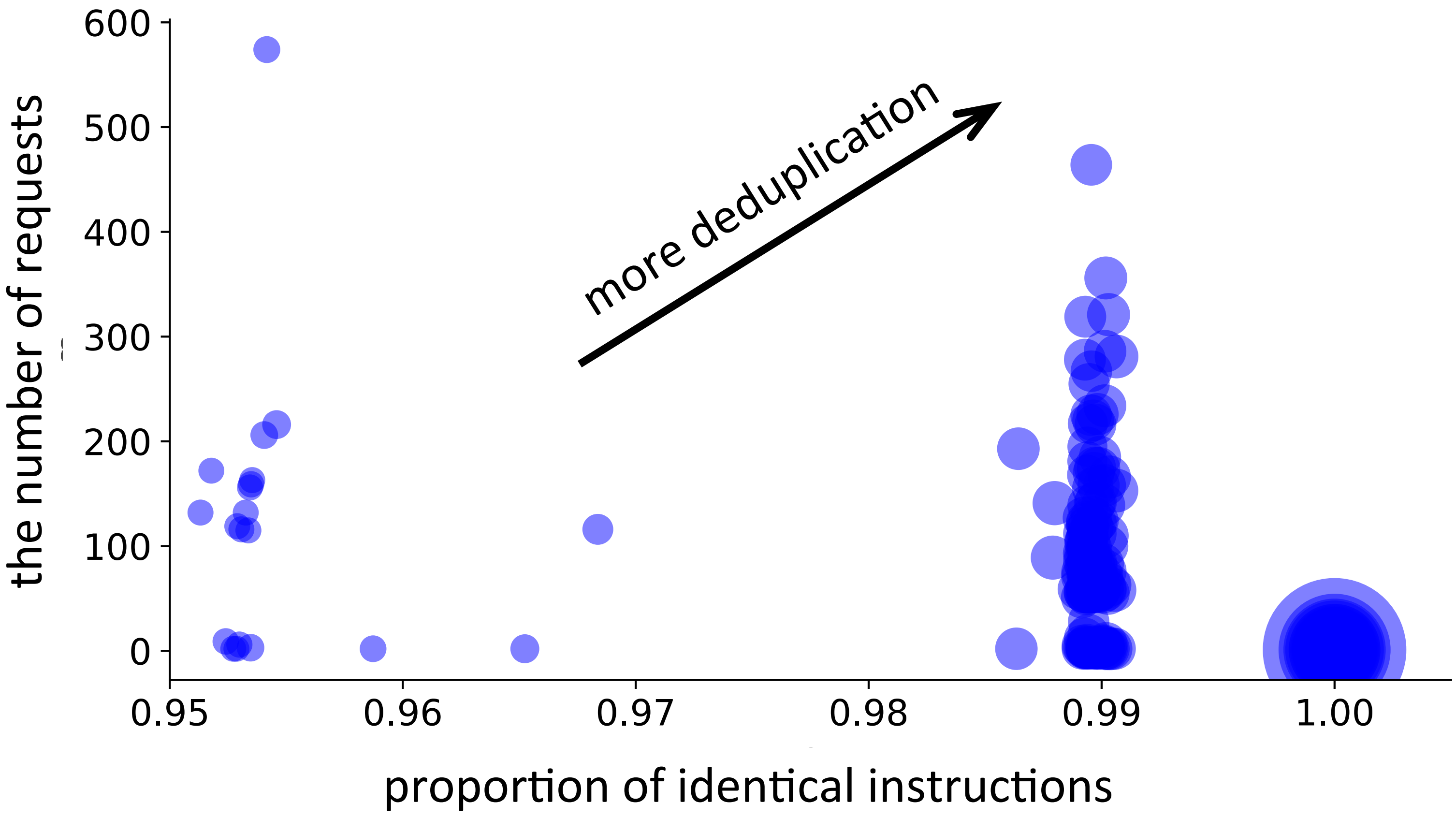
- What is the price of verifiability?

- How compatible is Orochi with legacy applications?

- Applications:
  - MediaWiki, phpBB and HotCRP

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

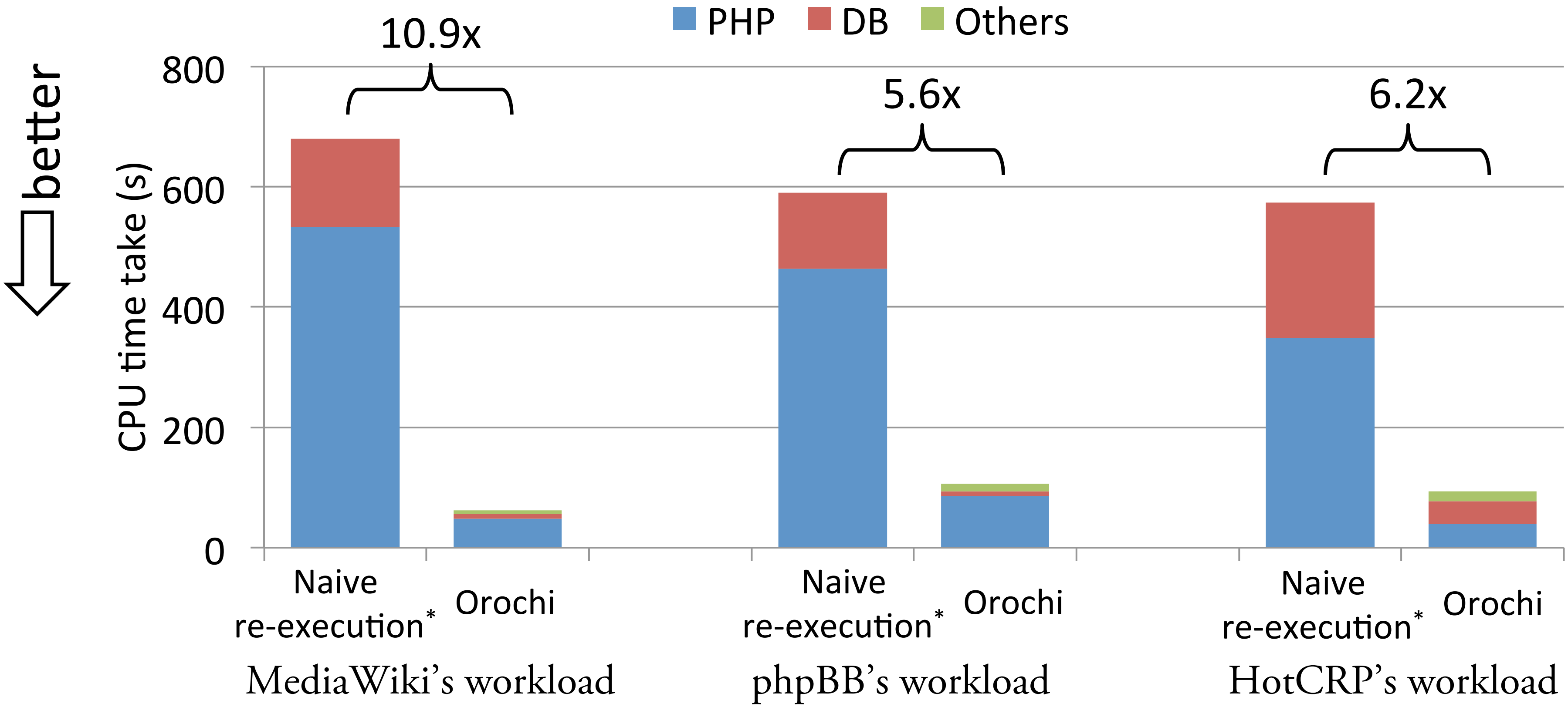
Our workloads see a lot of redundant computation



MediaWiki's workload (20K requests)

Orochi's verifier is efficient

Orochi's verifier achieves speedups compared to naive replay



* Pessimistically estimated from the original online execution

The price of verifiability is tolerable

CPU	
MediaWiki's workload	4.7%
phpBB's workload	8.6%
HotCRP's workload	5.9%

Network			
	trace (per req)	advice (per req)	Orochi's overhead
MediaWiki's workload	7.1KB	1.7KB	11.4%
phpBB's workload	5.7KB	0.3KB	2.7%
HotCRP's workload	3.2KB	0.4KB	10.9%

Storage	
MediaWiki's workload	1.0x
phpBB's workload	1.7x
HotCRP's workload	1.5x

Verifier needs to store the trace and advice for one audit epoch.

Orochi requires modest application adjustments

- Lines of code modified:
  - 346 lines of code change for MediaWiki
  - 270 lines of code change for phpBB
  - 67 lines of code change for HotCRP

- Most of the changes are due to
  - PHP features that our implementation does not support
  - Modifying the application to respect object semantics

Recap of evaluation

- Verifier: 5.6--10.9x speedup over naive re-execution

- Costs: storage at verifier, <10% overhead on server

- Compatibility: Modest application changes

Related work, future work, and wrap-up

❌ server is untrusted
2. server is concurrent
3. verifier is weaker than server
❌ server overhead is low; compatibility

Related work

- Efficient execution integrity
  - Replication: BFT
  - Attestation: TPMs, SGX
  - Probabilistic proofs: Pepper, CMT, Pinocchio, Pantry, SNARKs

- Computation deduplication (Delta execution, iThreads)

- Record-replay
  - Untrusted recorder: Accountable Virtual Machines
  - Accelerated replayer: Poirot
  - Multiprocessor: RecPlay, LEAP, DoublePlay, PRES, ODR, …

Related work

1. server is untrusted
❌ server is concurrent
3. verifier is weaker than server
❌ server overhead is low; compatibility

- Efficient execution integrity
  - Replication: BFT
  - Attestation: TPMs, SGX
  - Probabilistic proofs: Pepper, CMT, Pinocchio, Pantry, SNARKs
- Computation deduplication (Delta execution, iThreads)
- Record-replay
  - Untrusted recorder: Accountable Virtual Machines
  - Accelerated replayer: Poirot
  - Multiprocessor: RecPlay, LEAP, DoublePlay, PRES, ODR, …

Related work

- Efficient execution integrity
  - Replication: BFT
  - Attestation: TPMs, SGX
  - Probabilistic proofs: Pepper, CMT, Pinocchio, Pantry, SNARKs
- Computation deduplication (Delta execution, iThreads)
- Record-replay
  - Untrusted recorder: Accountable Virtual Machines
  - Accelerated replayer: Poirot
  - Multiprocessor: RecPlay, LEAP, DoublePlay, PRES, ODR, …

Wrap-up and future work

- Our solution to the Efficient Server Audit Problem:
  - Includes a new accelerated re-execution technique
  - Includes new algorithms for verifying concurrent executions
  - Comes with a rigorous proof of correctness
- Our instantiation for PHP, SQL web apps:
  - 5-10x speedups over a naive replay; <10% CPU overhead on server
- Future work includes:
  - SGX integration
  - Extend to multiple interacting servers
  - Accelerate other record-replay systems