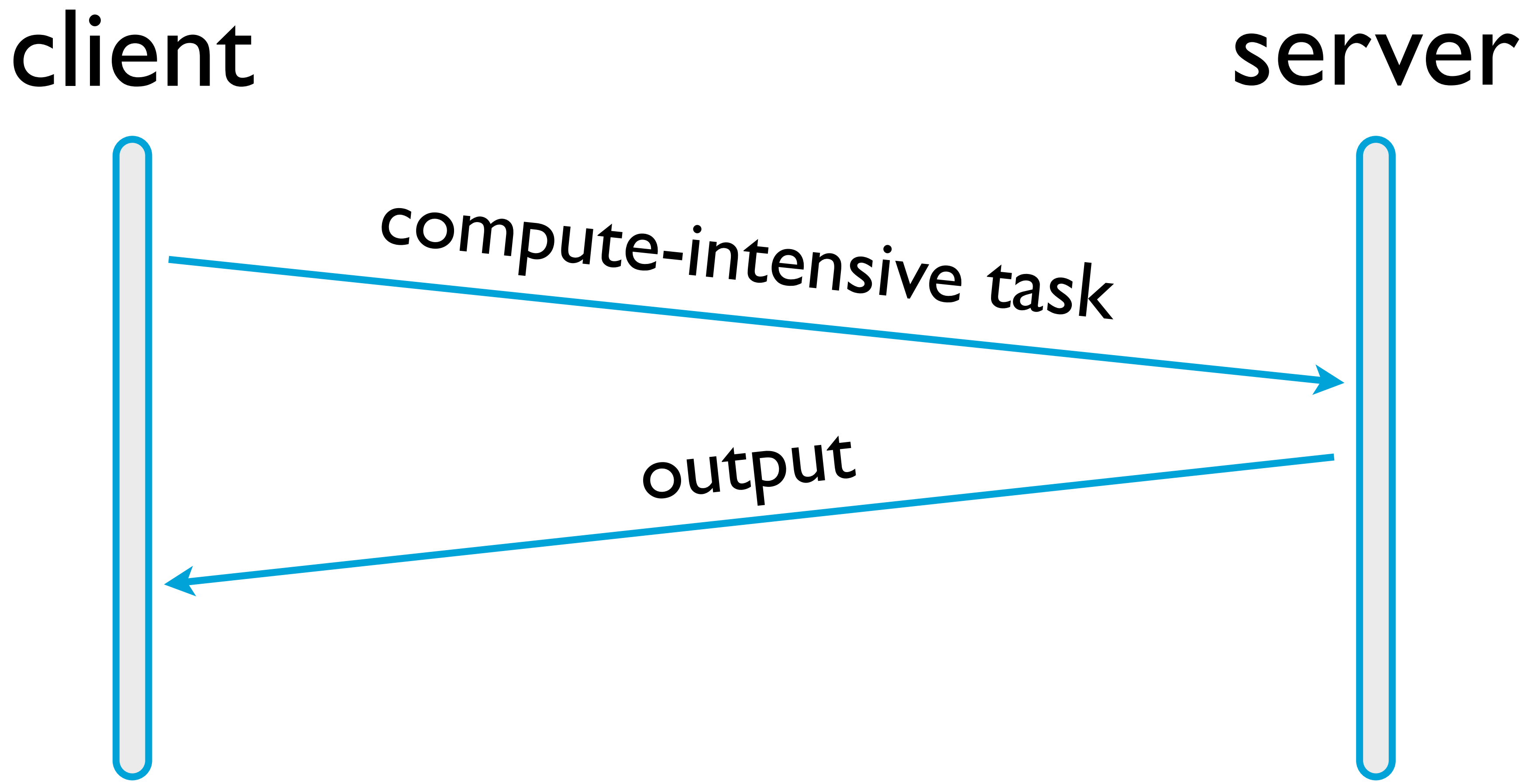
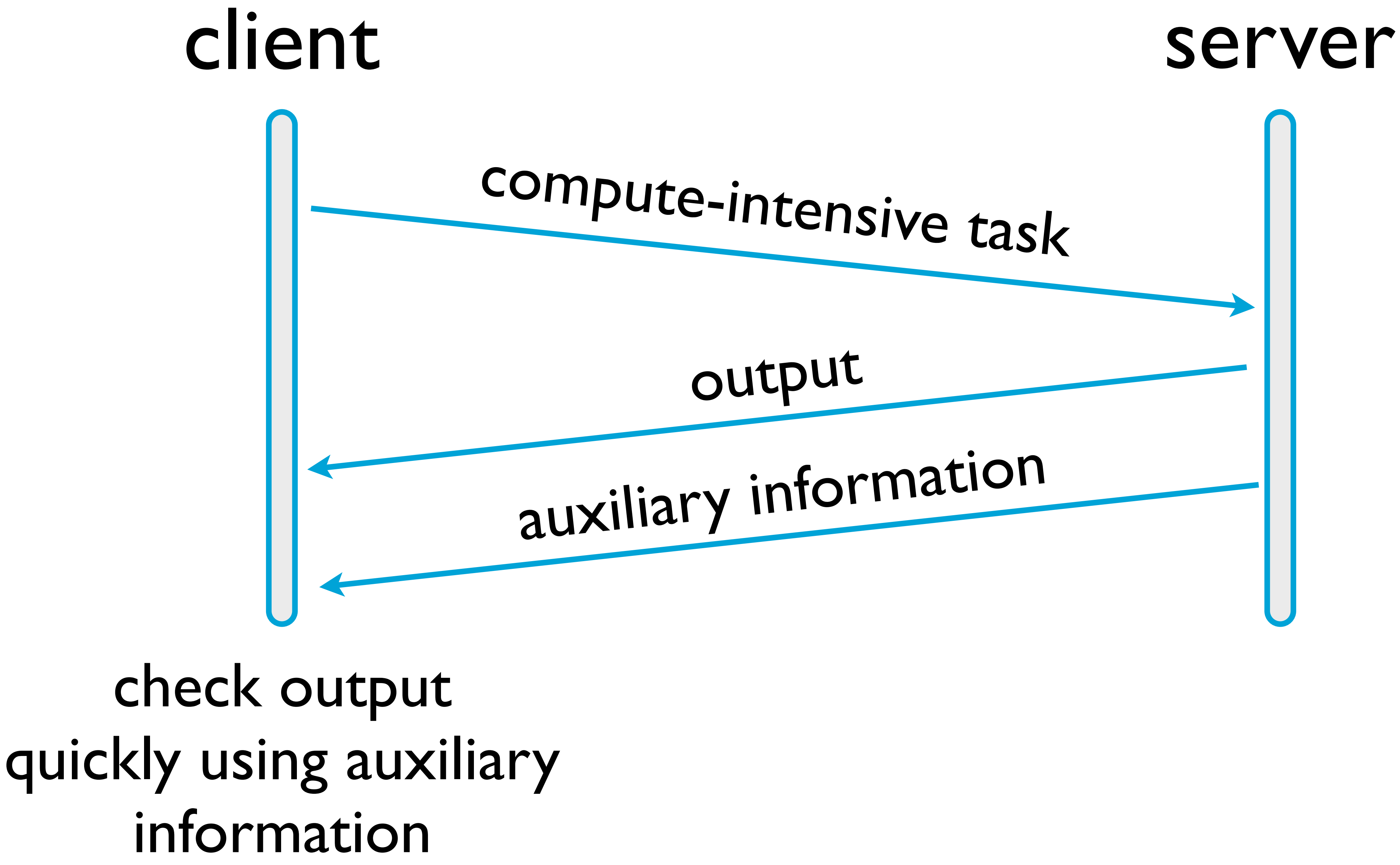
Resolving the conflict between generality and plausibility in verified computation

Srinath Setty,[★] Benjamin Braun,[★] Victor Vu,[★]
Andrew J. Blumberg,[★] Bryan Parno,[⁘] and Michael Walfish[★]

[★]The University of Texas at Austin [⁘]Microsoft Research

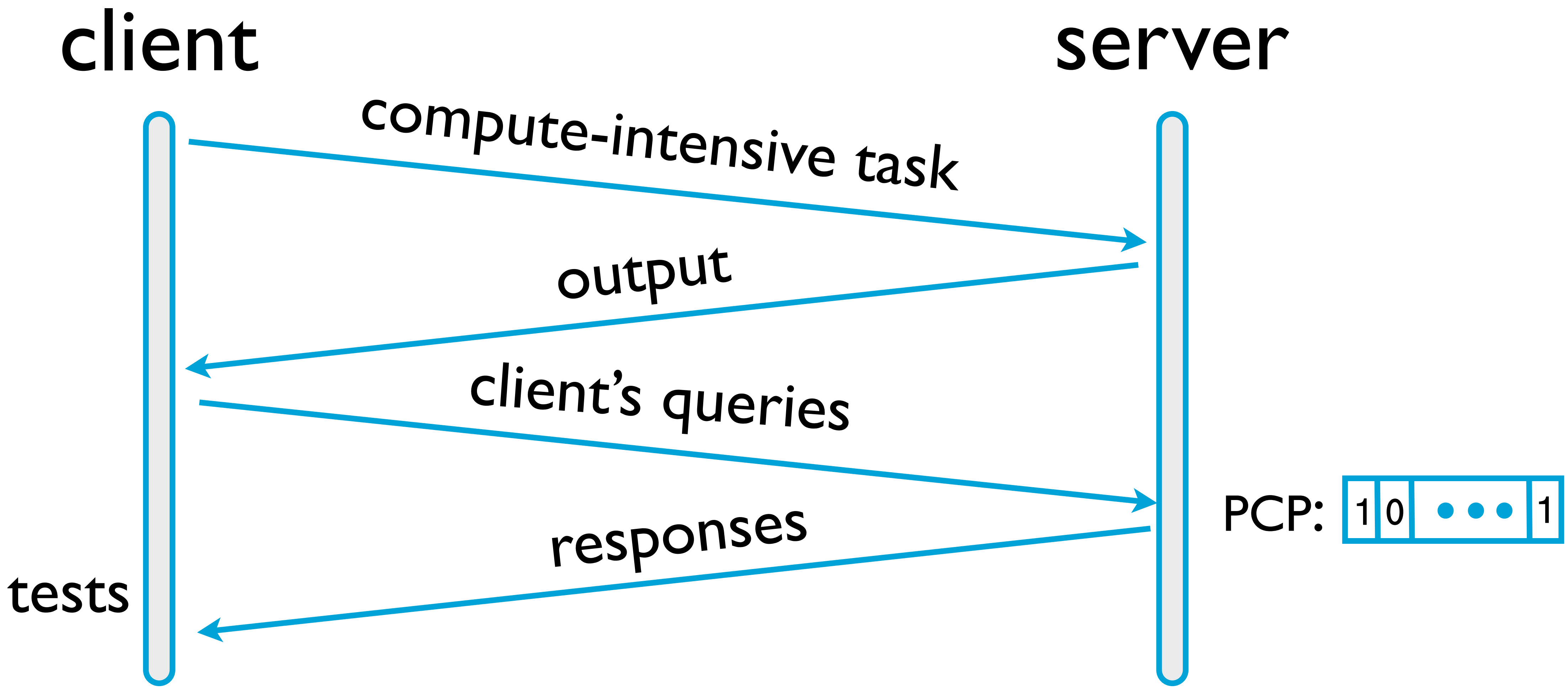

client
server
compute-intensive task
output


client
server
compute-intensive task
output
auxiliary information
check output
quickly using auxiliary
information

Probabilistically checkable proofs (PCPs) and cryptography offer a solution [ALMSS92, Kilian STOC92]

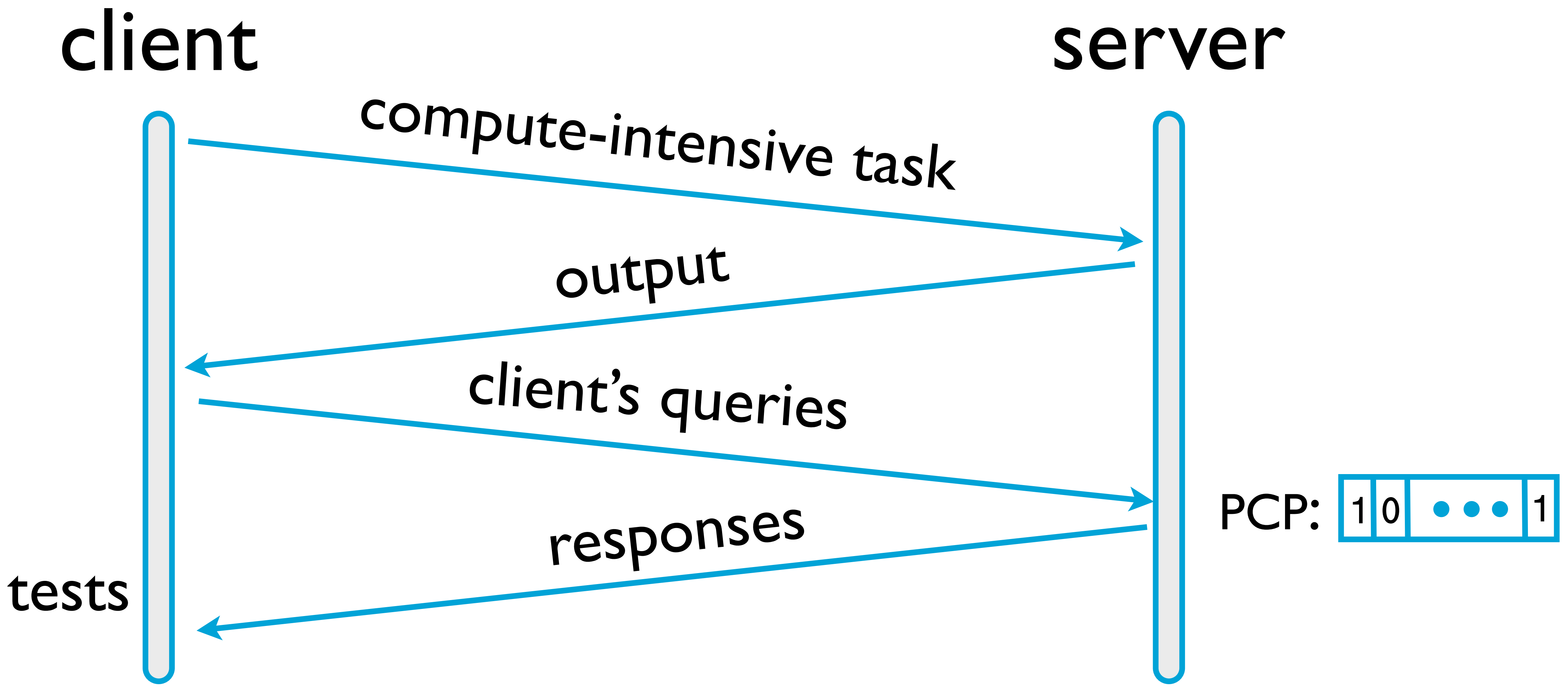
Probabilistically checkable proofs (PCPs) and cryptography offer a solution with excellent properties

[ALMSS92, Kilian STOC92]



Probabilistically checkable proofs (PCPs) and cryptography offer a solution with excellent properties

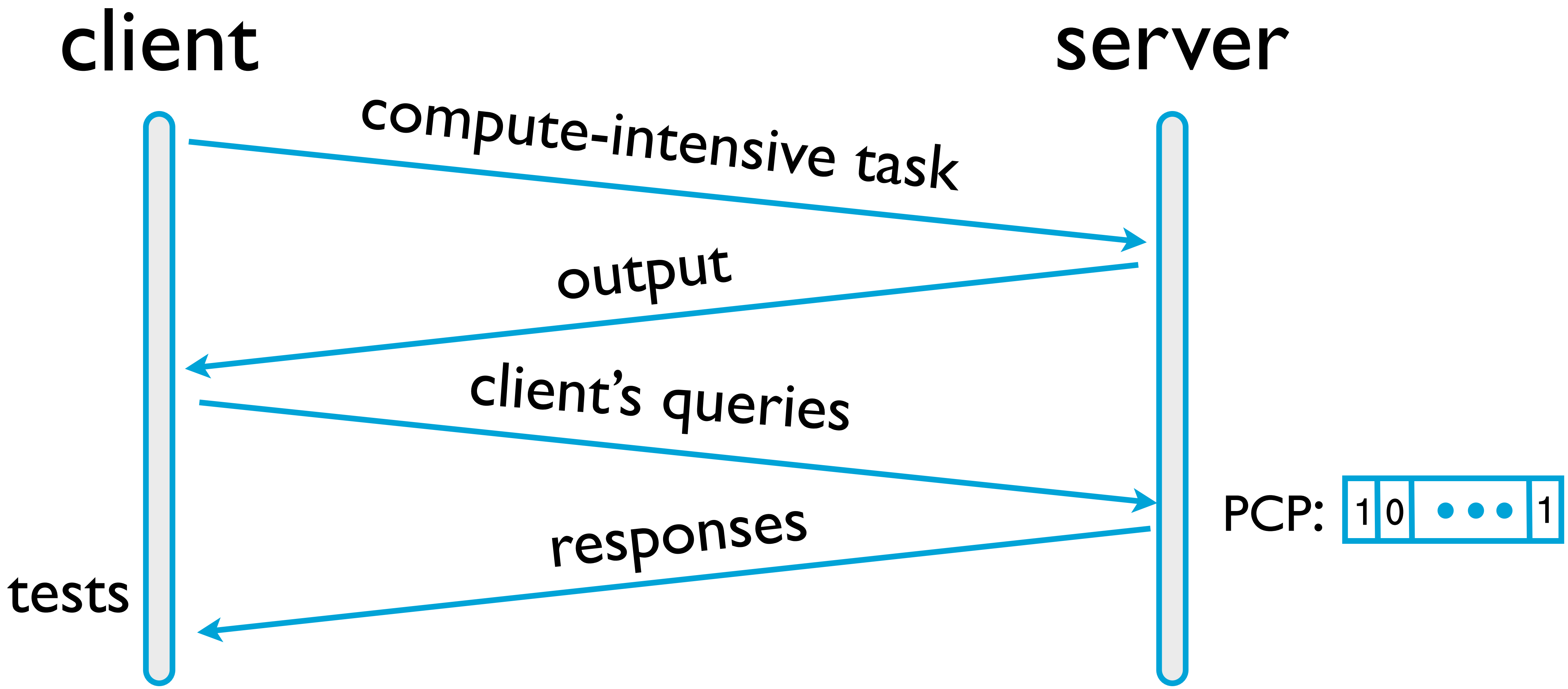
[ALMSS92, Kilian STOC92]



"Fast" verification: client saves work (asymptotically)

Probabilistically checkable proofs (PCPs) and cryptography offer a solution with excellent properties

[ALMSS92, Kilian STOC92]

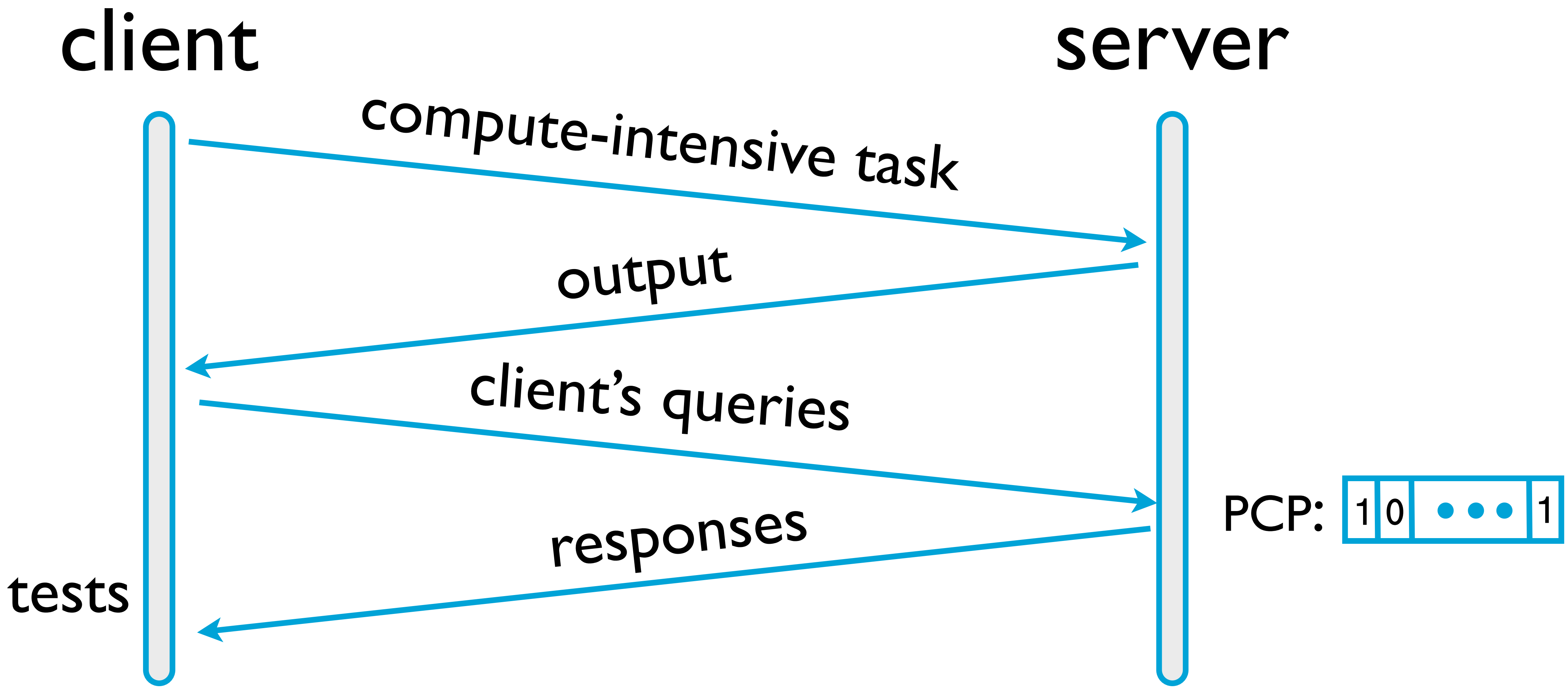


"Fast" verification: client saves work (asymptotically)

General-purpose: can outsource any computation

Probabilistically checkable proofs (PCPs) and cryptography offer a solution with excellent properties

[ALMSS92, Kilian STOC92]



"Fast" verification: client saves work (asymptotically)

General-purpose: can outsource any computation

Untrusted: no assumptions about the server

The theory provides strong security properties, but the costs are outrageous:

Verifying multiplication of 500×500 matrices would take more than 500 trillion CPU years (seriously).

There is interest in reducing costs with built systems

[HotOS11, NDSS12, USENIX SECURITY12, Cormode et al. ITCS12, Thaler et al. HotCloud12]

- Pepper and Ginger [NDSS12, USENIX SECURITY12] reduce costs by a factor of $>10^{20}$.

There is interest in reducing costs with built systems

[HotOS11, NDSS12, USENIX SECURITY12, Cormode et al. ITCS12, Thaler et al. HotCloud12]

- Pepper and Ginger [NDSS12, USENIX SECURITY12] reduce costs by a factor of $>10^{20}$.

Unfortunately, this progress comes with a tradeoff:

- Achieve generality at the cost of quadratic running time for the server [HotOS11, NDSS12, USENIX SECURITY12].
- Achieve good asymptotics at the cost of generality [Cormode et al. ITCS12, Thaler et al. HotCloud12].

Contributions of Zaatar:

Zaatar resolves the conflict between generality and expense, with a new proof encoding.

- Reduces server's work from $O(T^2)$ to $O(T \log T)$, where T is the running time of the computation.

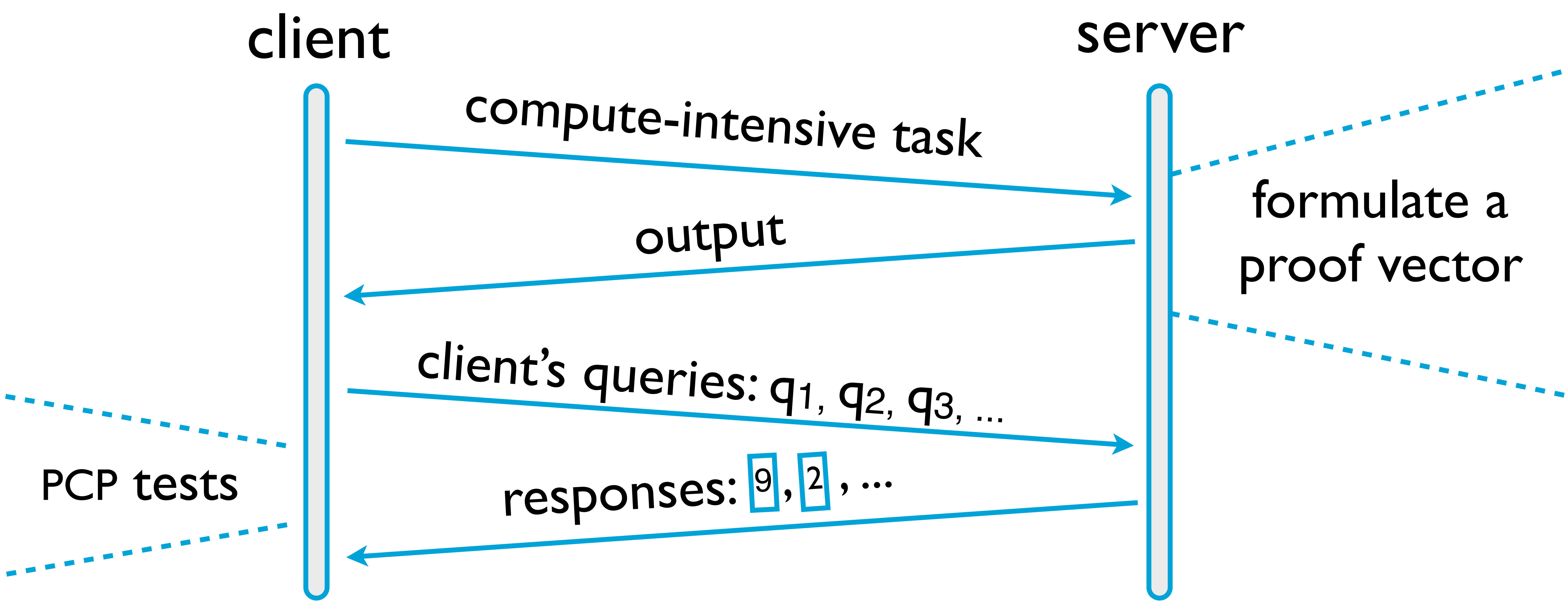
A system to compile programs in a subset of C into verifiable computations.

Rest of this talk:

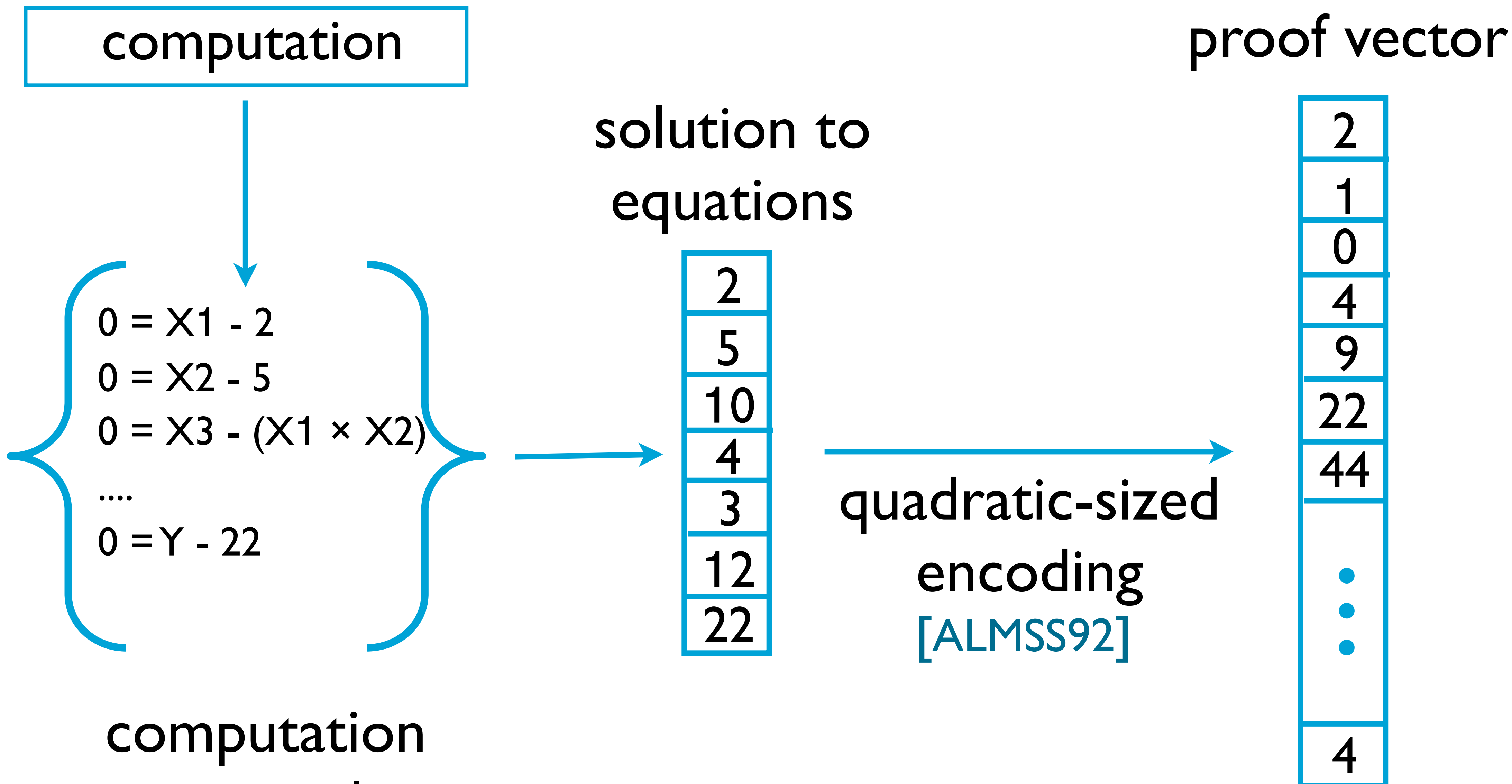
1. Design of Zaatar

2. Experimental results

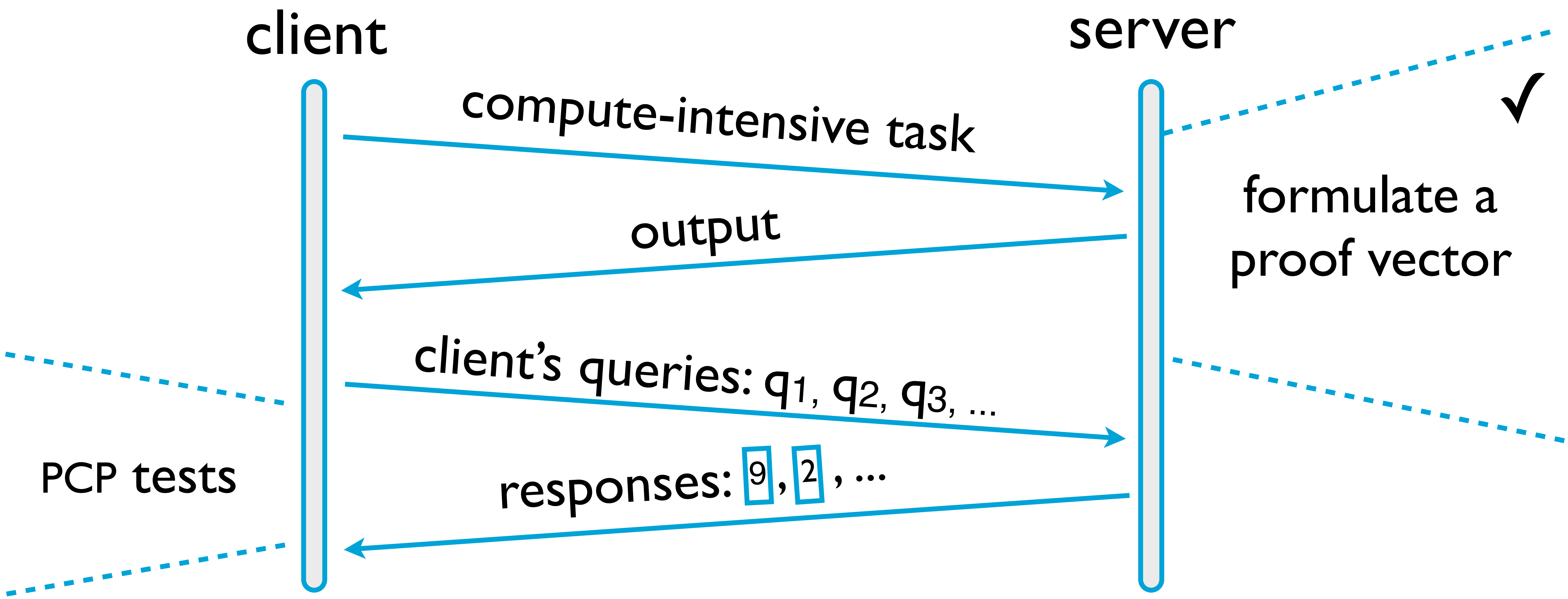
Ginger [USENIX SECURITY12] refines the protocol of [Ishai et al. CCC 07]



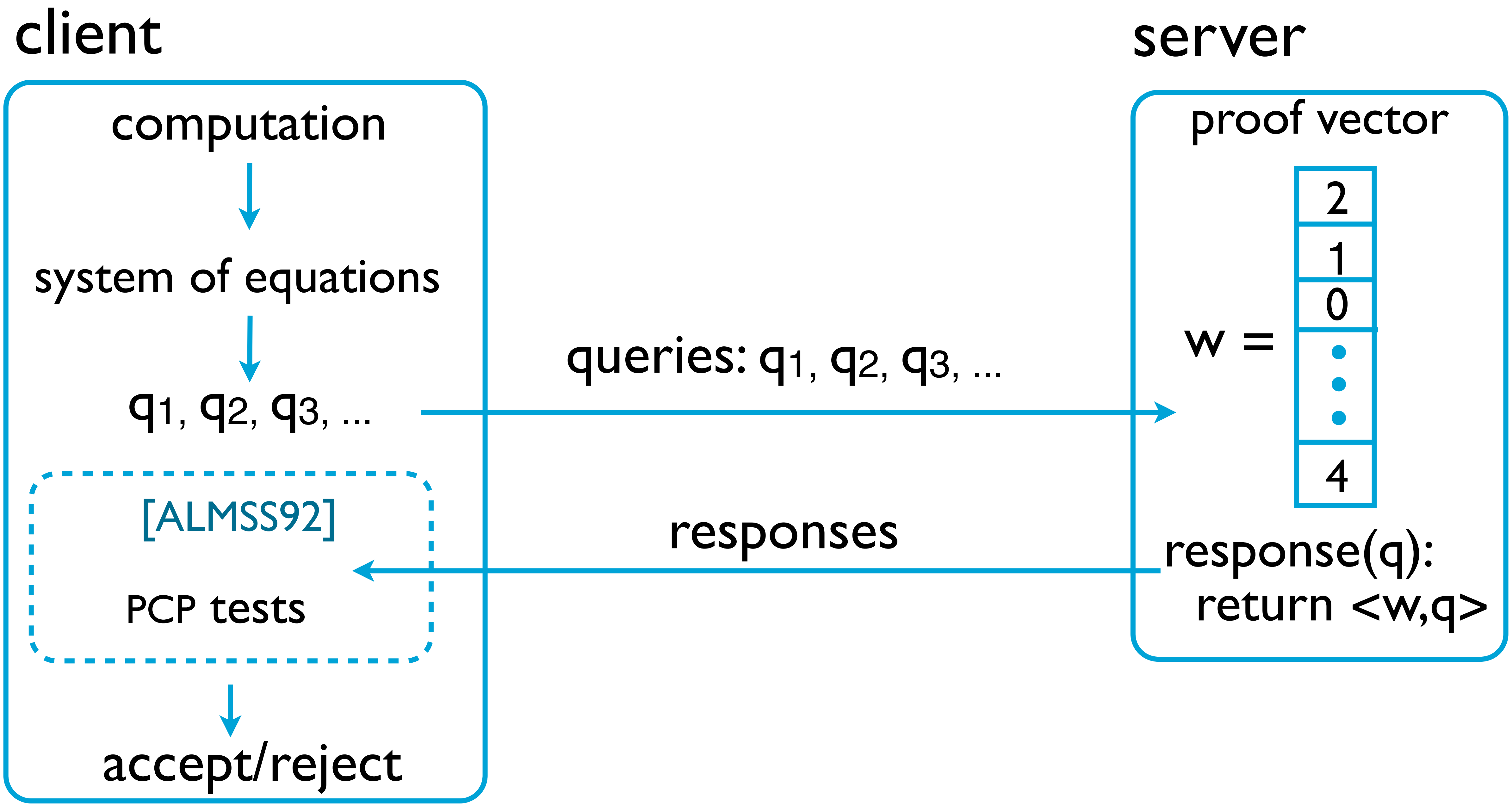
Formulating a proof vector by encoding a solution to the system of equations



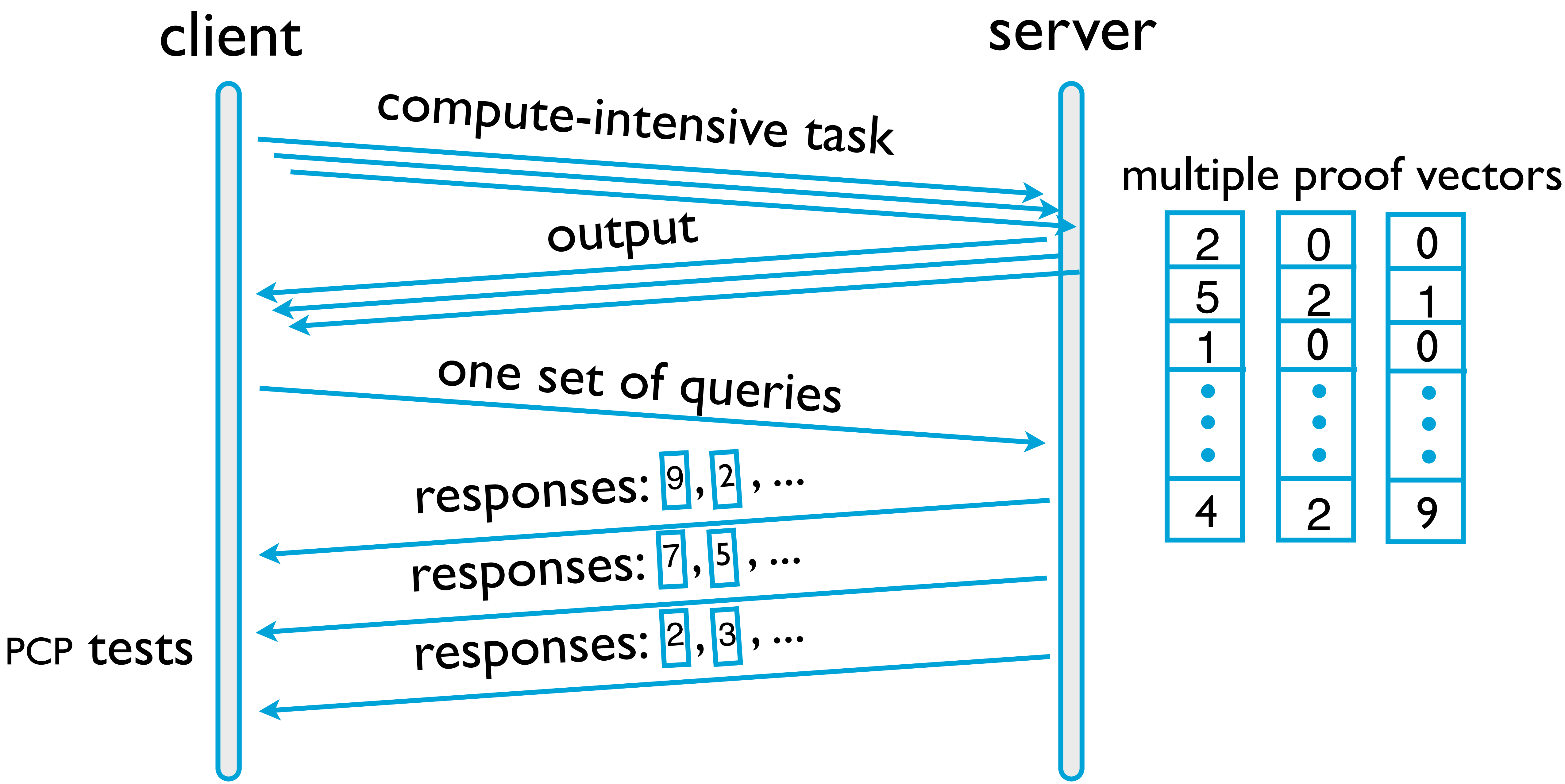
computation represented as a system of equations over a finite field

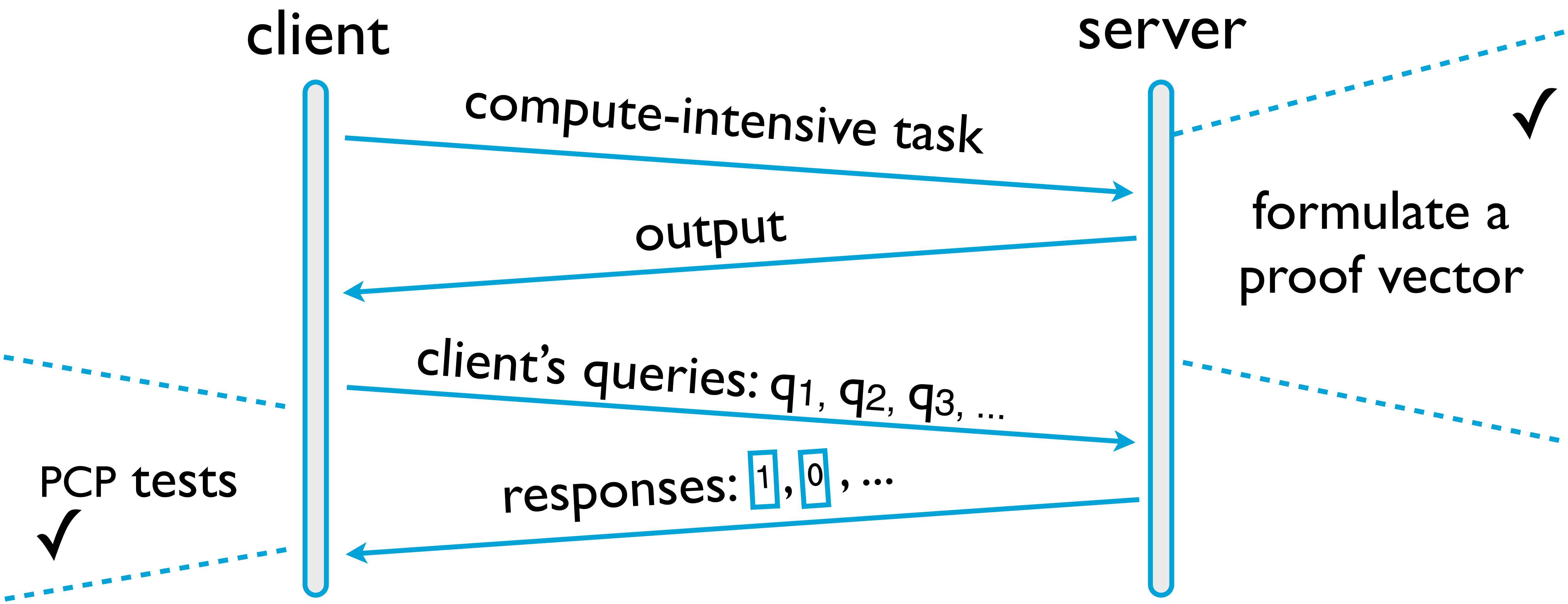

client
server
compute-intensive task
output
formulate a
proof vector
✓
client’s queries: q1, q2, q3, ...
responses: 9, 2, ...
PCP tests

The client interrogates the server, to verify



The client amortizes the query generation costs via batching

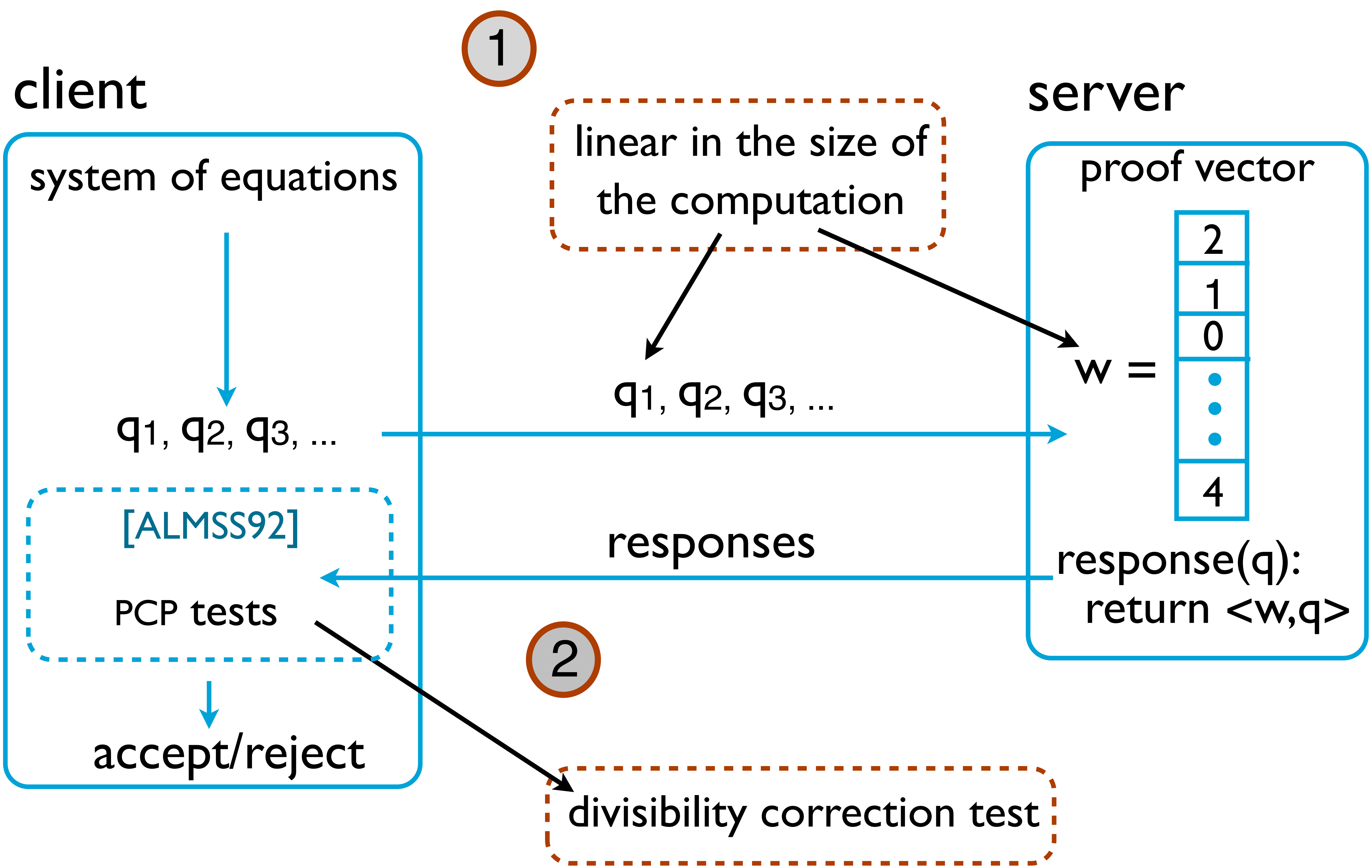



client
server
compute-intensive task
output
formulate a
proof vector
client's queries: q1, q2, q3, ...
responses: 1, 0, ...
PCP tests

Designing a new probabilistically checkable proof (PCP) encoding for linear PCPs

- The proof vector in prior works has redundancy, usually.
- [Gennaro et al. EUROCRYPT13] introduce quadratic arithmetic programs (QAPs) to represent computations.
- Our insight: QAPs can be used to design a new PCP.
  - Eliminates (most) redundancy in Ginger's proof vector, in a general-purpose way.

Verification protocol in Zaatar



Our refinement has several benefits

- Reduces costs
  - The proof vector length is linear in the running time of the computation.
  - The server's work is now O(T log T), where T is the running time of the computation.
- Has theoretical significance
  - It shows a connection between QAPs and PCPs (also shown by [Bitansky et al. TCC13], in parallel work).
  - It resolves a conjecture of [Ishai et al. CCC07].

1. Design of Zaatar

   ✓ Reducing the size of the proof vector

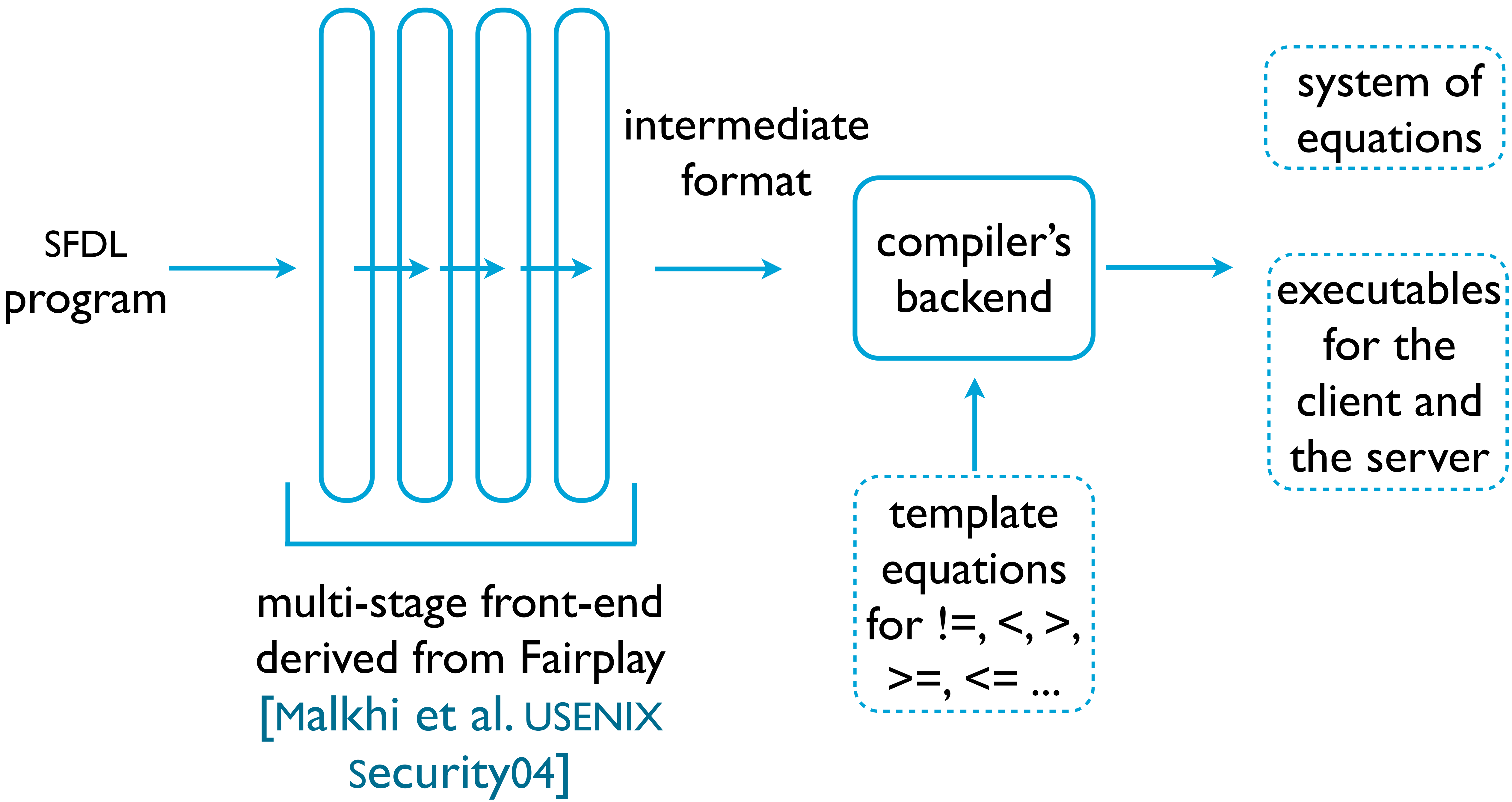
   ⟶ Integrating with Ginger [USENIX SECURITY12]

2. Experimental results

Integrating the Zaatar protocol with Ginger [USENIX SECURITY12]

- Adapt Ginger's compilation toolchain
  - Enhance the compiler to accept programs in a subset of C, following [Parno et al. Oakland13].
- Integrate with Ginger's distributed server that uses GPU acceleration for crypto operations.

Ginger's compiler works in two phases



An example

increment(X)

Y = X + 1

=

0 = X - <input>

0 = Y - (X + 1)

0 = Y - <output>

An example

increment(X)
Y = X + 1

=

0 = X - <input>
0 = Y - (X + 1)
0 = Y - <output>

Once the inputs are fixed, the system of equations has a solution if and only if the output is correct.

Suppose the input is 6

If the output is 7

0 = X - 6
0 = Y - (X + 1)
0 = Y - 7

There is a solution

If the output is 8

0 = X - 6
0 = Y - (X + 1)
0 = Y - 8

There is no solution

Encoding "Z = X != Y"

$$0 = (X - Y) \cdot M - Z$$

$$0 = (1 - Z) \cdot (X - Y)$$

Observe:

If X == Y, then Z will have to be set 0, to satisfy the first.

If X != Y, then Z will have to be set 1, to satisfy the second.

Encoding conditional control flow

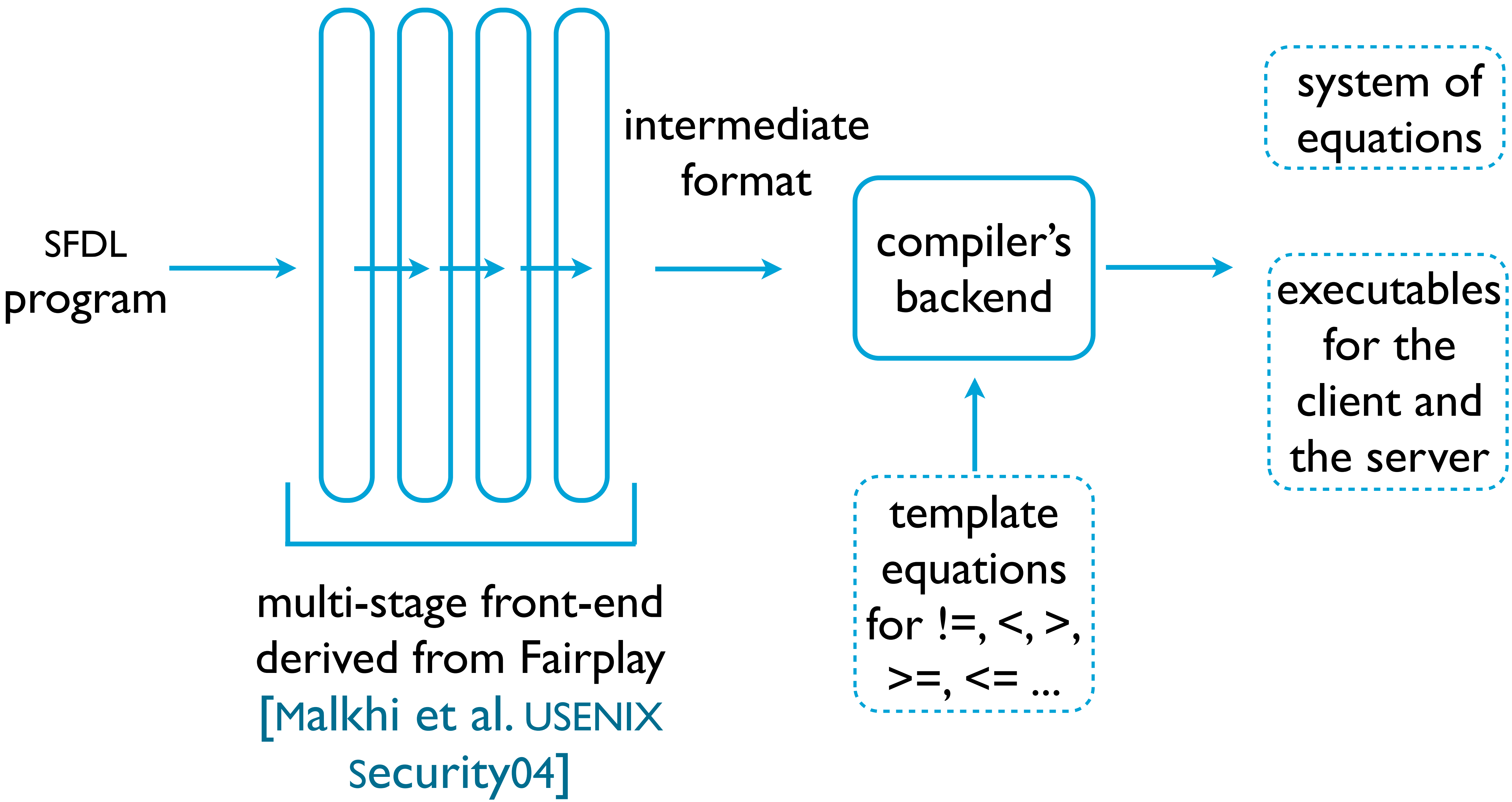
```
function(bool X)
  if (X)
   Y = 3
  else
   Y = 4
```

=

$$0 = X - M$$

$$Y = M \cdot 3 + (1-M) \cdot 4$$

Ginger’s compiler works in two phases



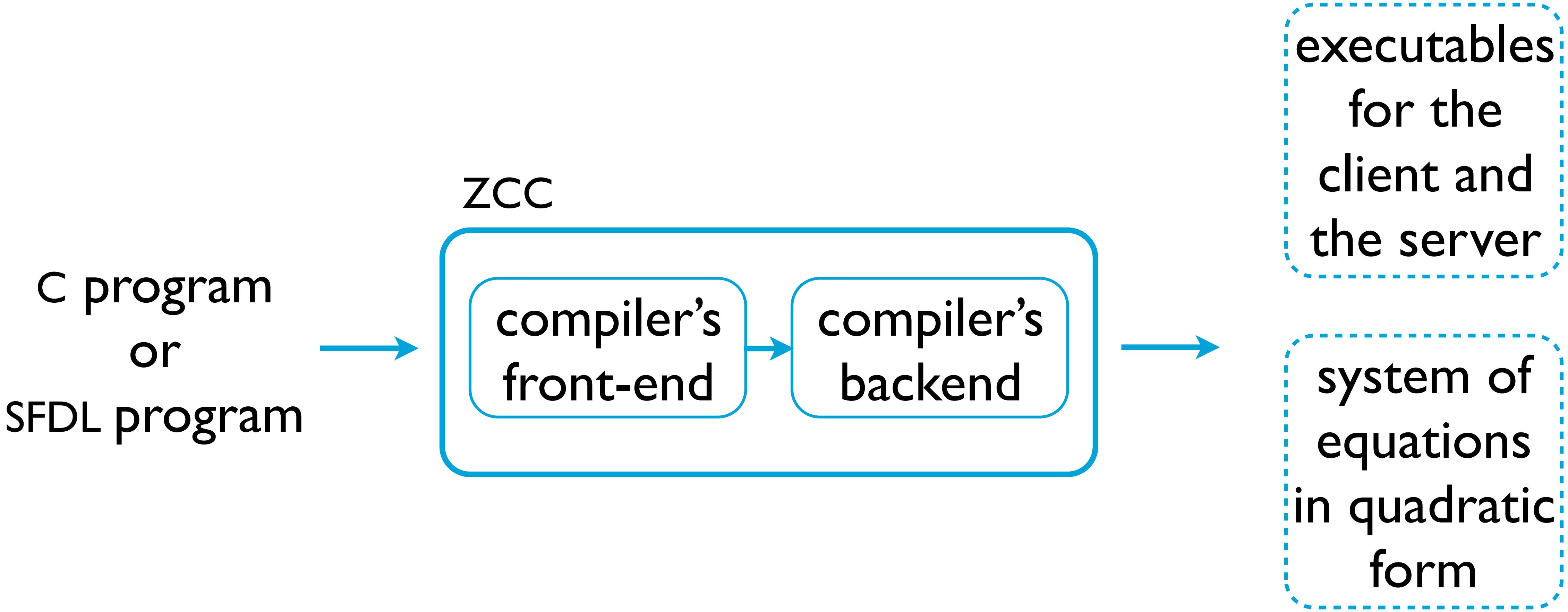
Adapting Ginger's compiler toolchain for Zaatar

- Zaatar's protocol requires equations in *quadratic form*
  - $A \cdot B = C$, where A, B, and C are degree-1 polynomials.
- Ginger's compiler outputs degree-2 equations, and our modification transforms them into quadratic form.
- $Z_1 \cdot Z_2 = Z_3 \cdot Z_4 + Z_5$ is automatically split into:

  $Z_3 \cdot Z_4 = Z_6$

  $Z_1 \cdot Z_2 = Z_6 + Z_5$

Zaatar's compiler adapted from Ginger



Recently, we enhanced the compiler to accept programs in a subset of C, following [Parno et al. Oakland13].

✓ (1) Design of Zaatar

→ (2) Experimental results

Benchmarks and implementation

Benchmarks:

- all-pairs shortest paths, with m nodes in a graph
- longest common subsequence with strings of length m
- PAM clustering m samples with d dimensions each
- root finding for polynomials with m variables
- Fannkuch benchmark

Distributed implementation, to handle batching

C++ code; HTTP/Open MPI to distribute server's work

CUDA to offload cryptographic work to GPUs

Evaluation method

Measure Zaatar's performance from experiments.

Estimate Ginger's performance from microbenchmarks, since it's infeasible to run.

Evaluation testbed

A cluster at Texas Advanced Computing Center (TACC)

Each machine runs Linux on an Intel Xeon 2.53 GHz with 48GB of RAM.

Evaluation questions

1. What are the costs of Zaatar's server, relative to simply executing the computation?

2. What are the costs of Zaatar's server, relative to the costs of Ginger's server?

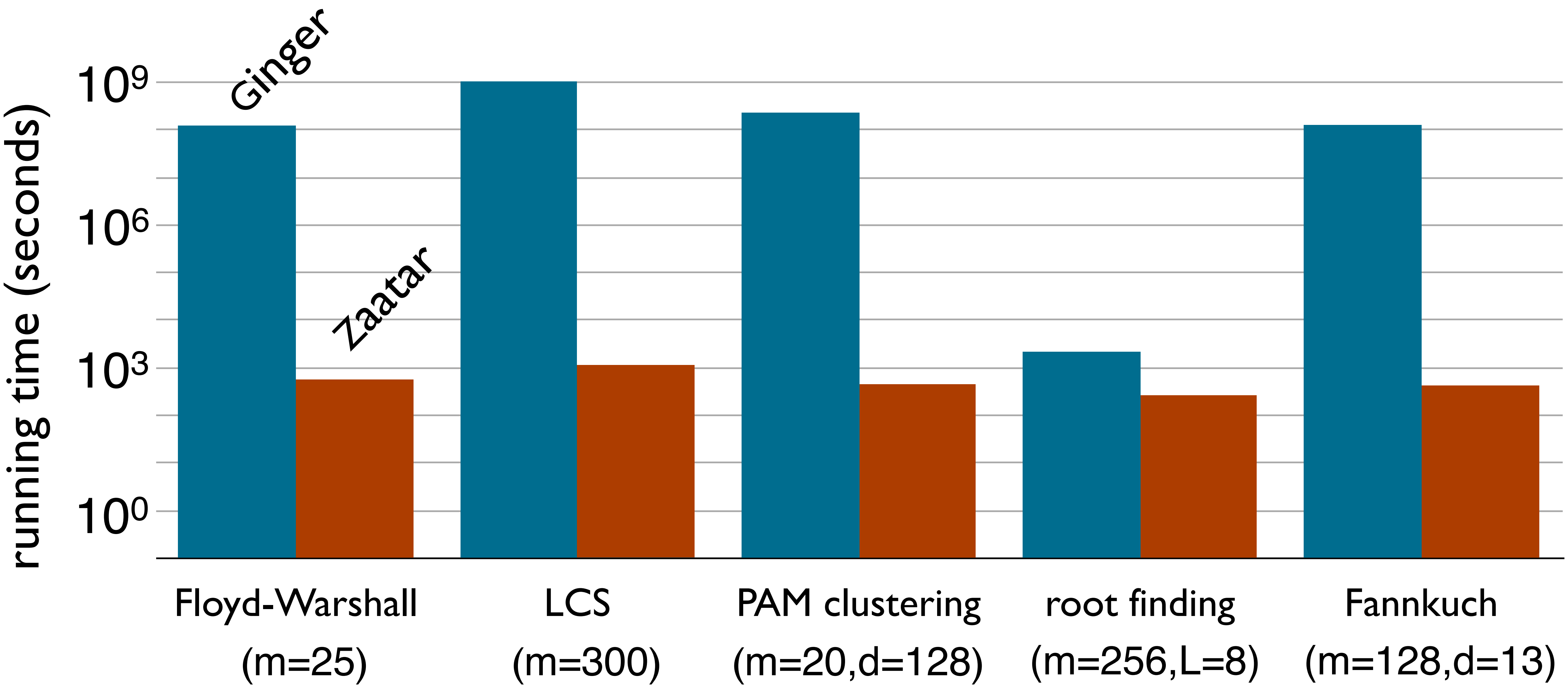
3. What are the break-even points under Zaatar?

Zaatar's server is many orders of magnitude more expensive than simply executing the computation

benchmark computation	local	Zaatar
all-pairs shortest paths (m=25)	8.1 ms	9.0 min
longest common subsequence (m=300)	1.4 ms	18.0 min
PAM clustering (m=20,d=128)	52 ms	8.7 min
root finding by bisection (m=256,L=8)	800 ms	6.5 min
Fannkuch benchmark (m=128,d=13)	0.8 ms	8.8 min

Performance from a recent, improved implementation.

Zaatar's server is 1-6 orders of magnitude faster than Ginger's server

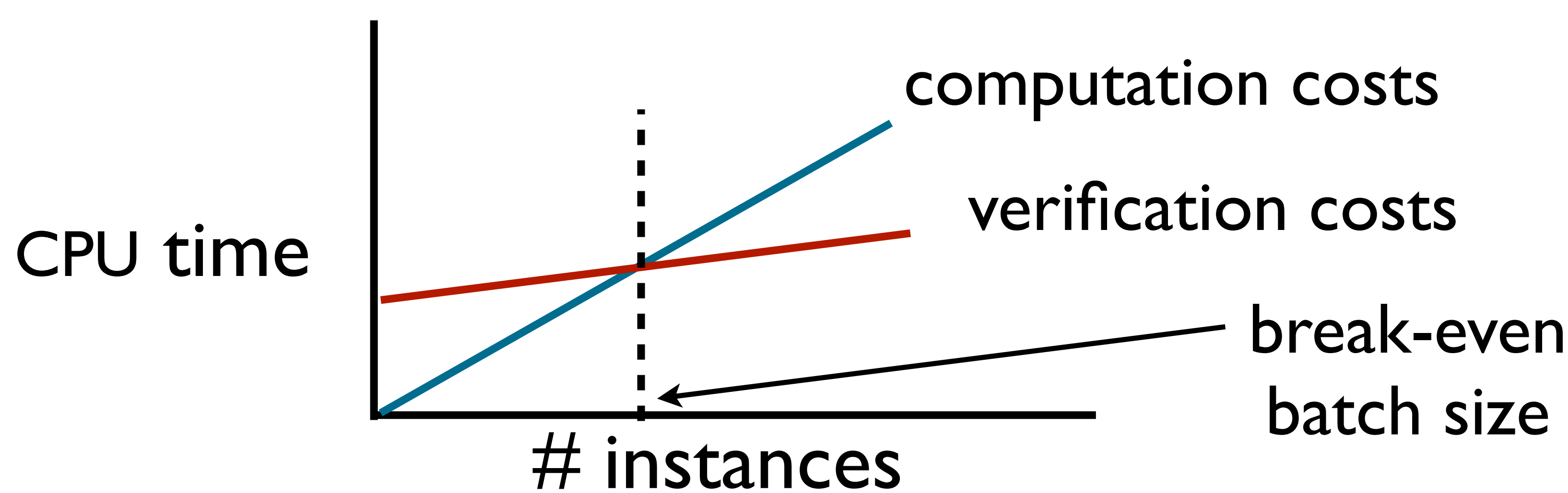


Evaluation questions

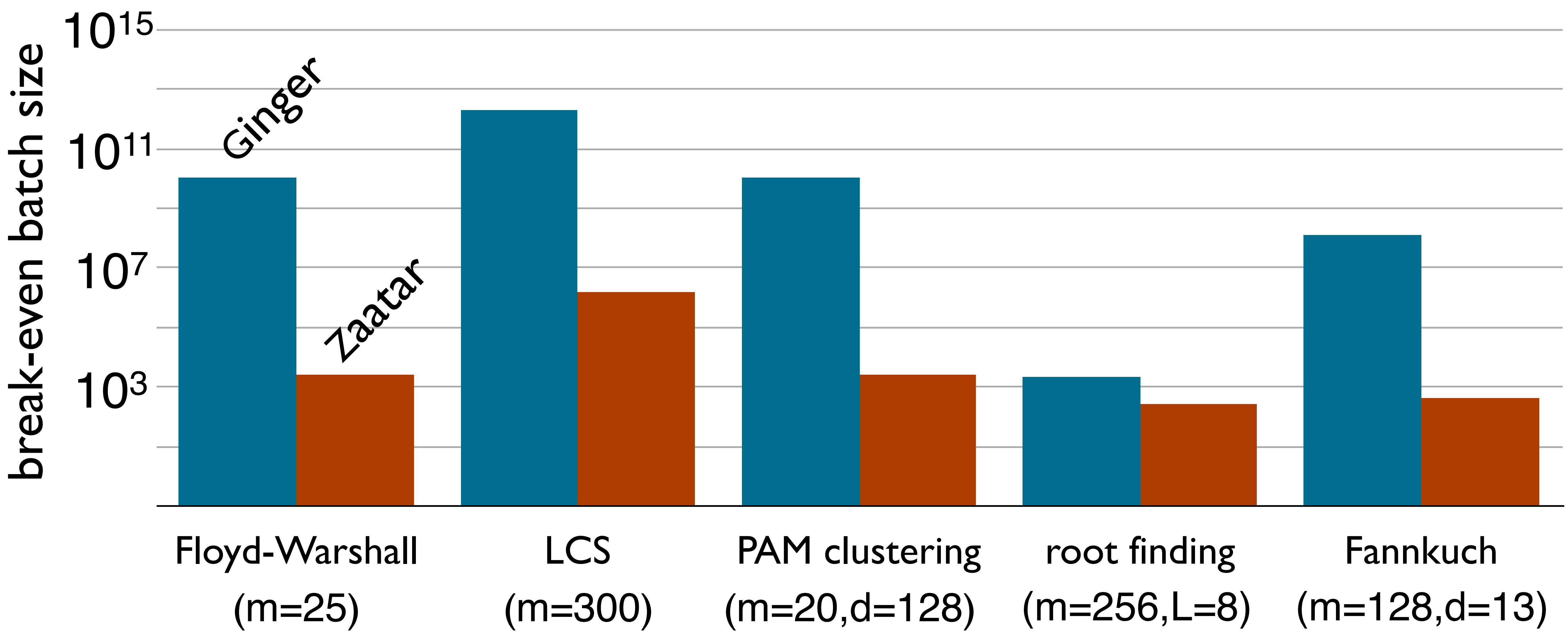
✓ ① What are the costs of Zaatar's server, relative to simply executing the computation?

✓ ② What are the costs of Zaatar's server, relative to the costs of Ginger's server?

→ ③ What are the break-even points under Zaatar?



The break-even batch sizes are 1-6 orders of magnitude smaller in Zaatar compared to Ginger



✓ ① Design of Zaatar

✓ ② Experimental results

→ ③ Limitations, prior work, and summary

Limitations of Zaatar

- Zaatar's model of computation is general-purpose, but transformation into this model may not be efficient for all program constructs.

- The server's asymptotic running time is good, but the constant is still large.

- The client has to batch-verify computations to gain from outsourcing.

Related work on verified computation

Make strong trust assumptions or give up being general-purpose:

- Replication [Castro & Liskov TOCS02], trusted hardware [Chiesa & Tromer ICS10, Sadeghi et al. TRUST10], and auditing [Monrose et al. NDSS99, Haeberlen et al. SOSP07]
- Special-purpose [Freivalds MFCS79, Golle & Mironov RSA01, Sion VLDB05, Benabbas et al. CRYPTO11, Boneh & Freeman EUROCRYPT11]

Proof-based verified computation

- Theory of Probabilistically checkable proofs [Ben-Or STOC88, Babai STOC91, Kilian STOC92, Blum et al. JACM95, ALMSS92]
- Via fully homomorphic encryption [Gennaro et al. CRYPTO10, Chung et al. CRYPTO10]
- Theory that can be a foundation for systems [Ben-Sasson et al. STOC13, ITCS13]

Built systems that refine and evaluate proof-based verified computation

- Pepper and Ginger [HotOS11, NDSS12, USENIX SECURITY12] based on [Ishai et al. CCC07]
- Interactive proofs [Thaler et al. ITCS12, HotCloud12] based on [Goldwasser et al. STOC08]
- Pinocchio [Parno et al. Oakland13] based on [Gennaro et al. EUROCRYPT13]

Summary of Zaatar

Zaatar resolves the conflict between generality and expense, with a new proof encoding.

- Reduces server's work from $O(T^2)$ to $O(T \log T)$, where T is the running time of the computation.

A system to compile programs in a subset of C into verifiable computations.

Verified computation can be almost practical, especially when the server is inexpensive and powerful.