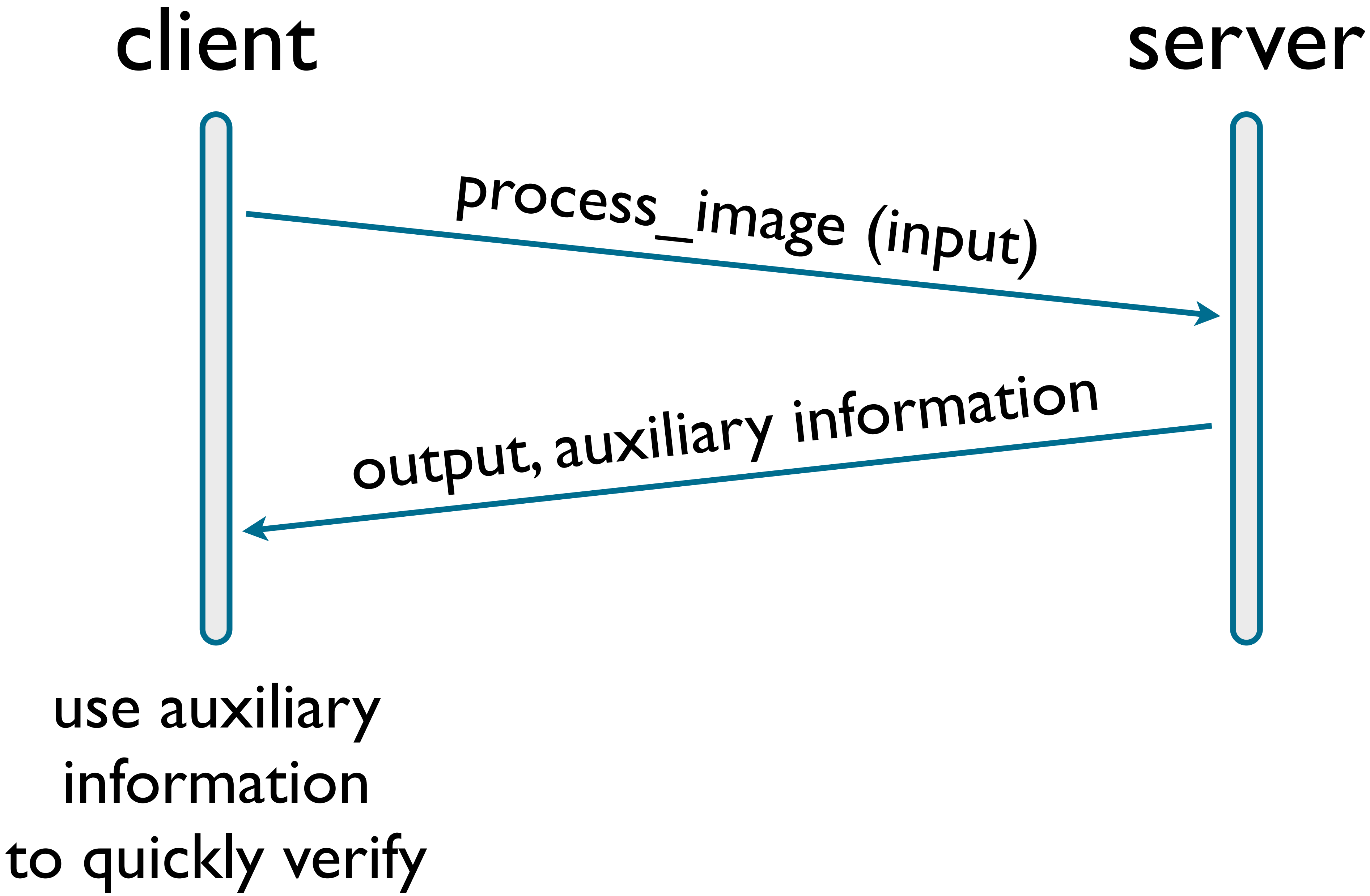
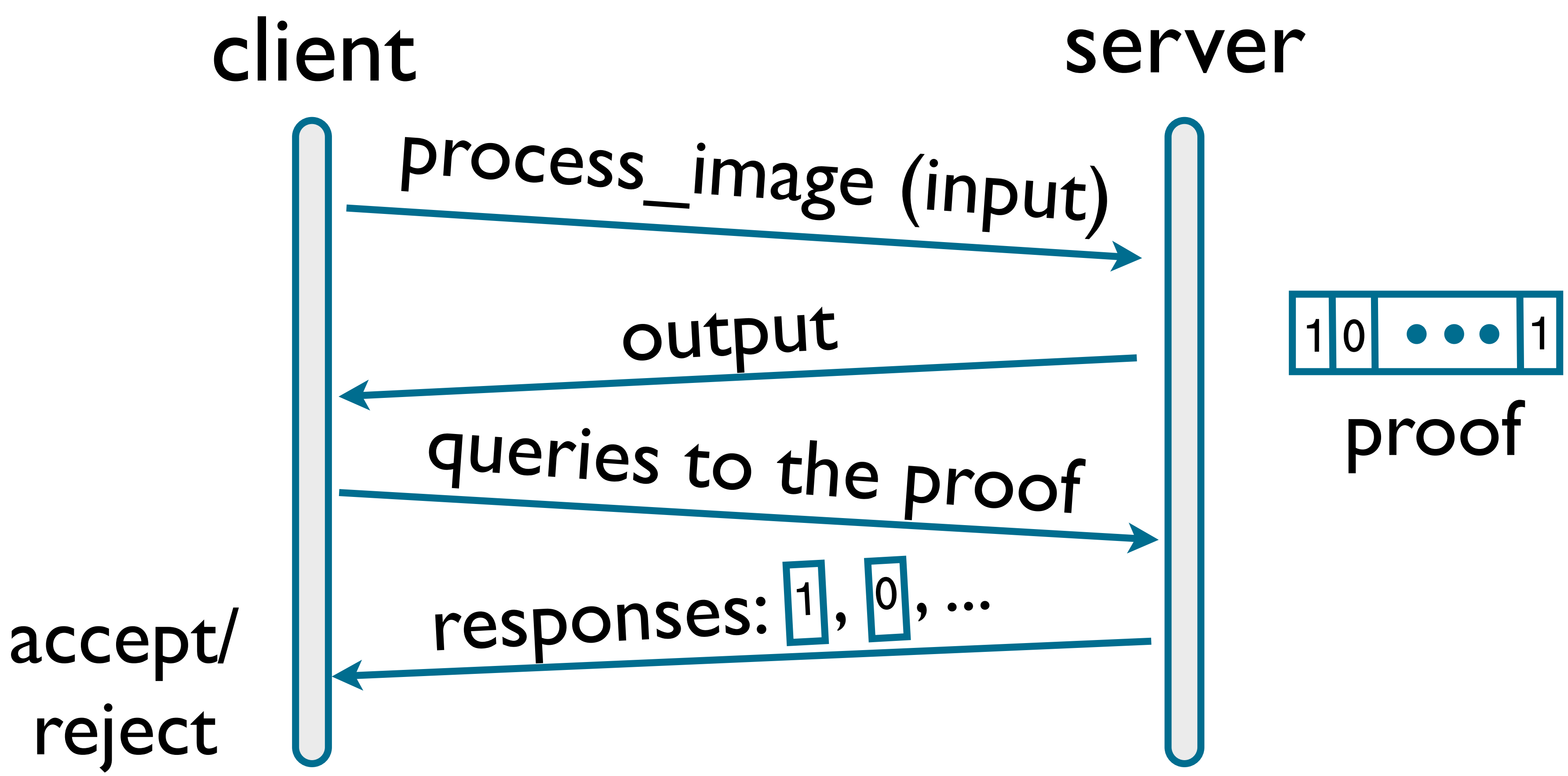
Taking proof-based verified computation a few steps closer to practicality

Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun,
Andrew J. Blumberg, and Michael Walfish

The University of Texas at Austin


client
server
process_image (input)
output, auxiliary information
use auxiliary information to quickly verify

Probabilistically checkable proofs (PCPs) can help



Fast verification: client saves work (asymptotically)

General-purpose: can outsource any computation

Unconditional: 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 a lot of renewed interest in reducing costs with built systems

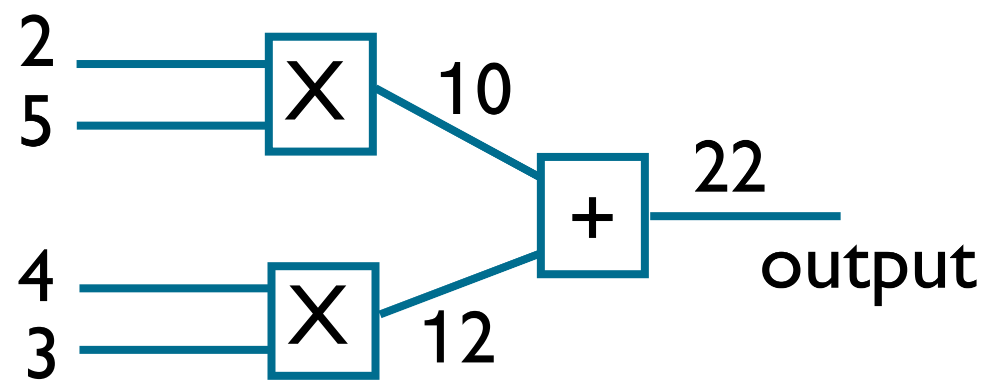
- Two efforts: PEPPER [HotOS11, NDSS12], Thaler et al. [ITCS12, HotCloud12]
- In some cases, PEPPER reduces costs by a factor of 10^{20} over a naive implementation of the theory

But all of these recent works have notable limitations

1. The client has to outsource large computations to offset verification costs

2. Their model of computation is arithmetic circuits

   Example:



   Arithmetic circuits cannot concisely express conditional control flow or comparisons

GINGER addresses some of these limitations

Reduces the client's checking work and network costs by several orders of magnitude

Includes a massively parallel GPU-based implementation

Supports a general-purpose programming model

- Concise conditionals, comparisons, efficient floating-point representation, etc.
- A compiler to go from high-level code to executables

The main takeaway

GINGER and its predecessor (PEPPER) together reduce costs by a factor of 10^{20} using theory and systems techniques

We still need a factor of $\approx 10^3$ on the server for true practicality

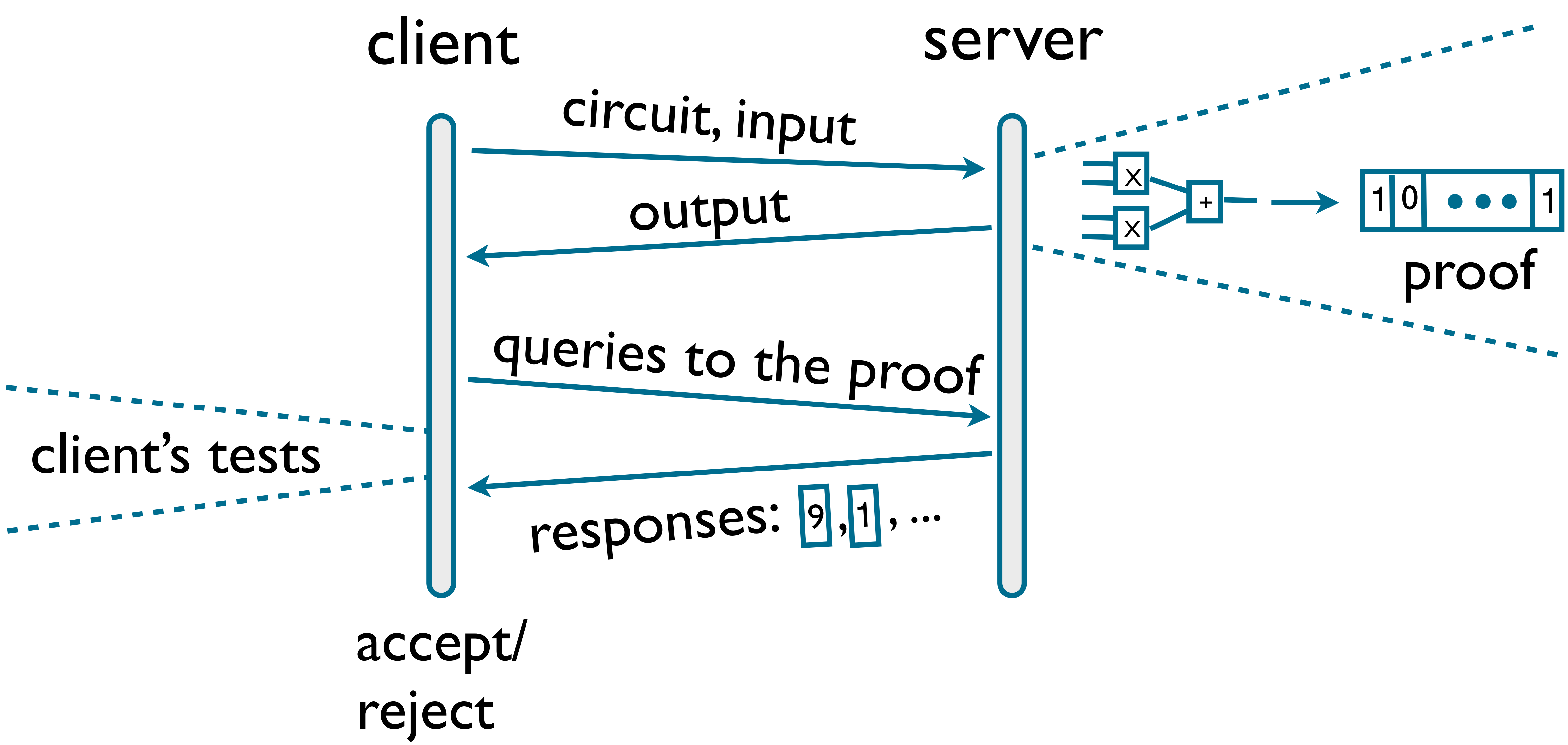
We think that proof-based verified computation could be practical in the near future

Rest of this talk

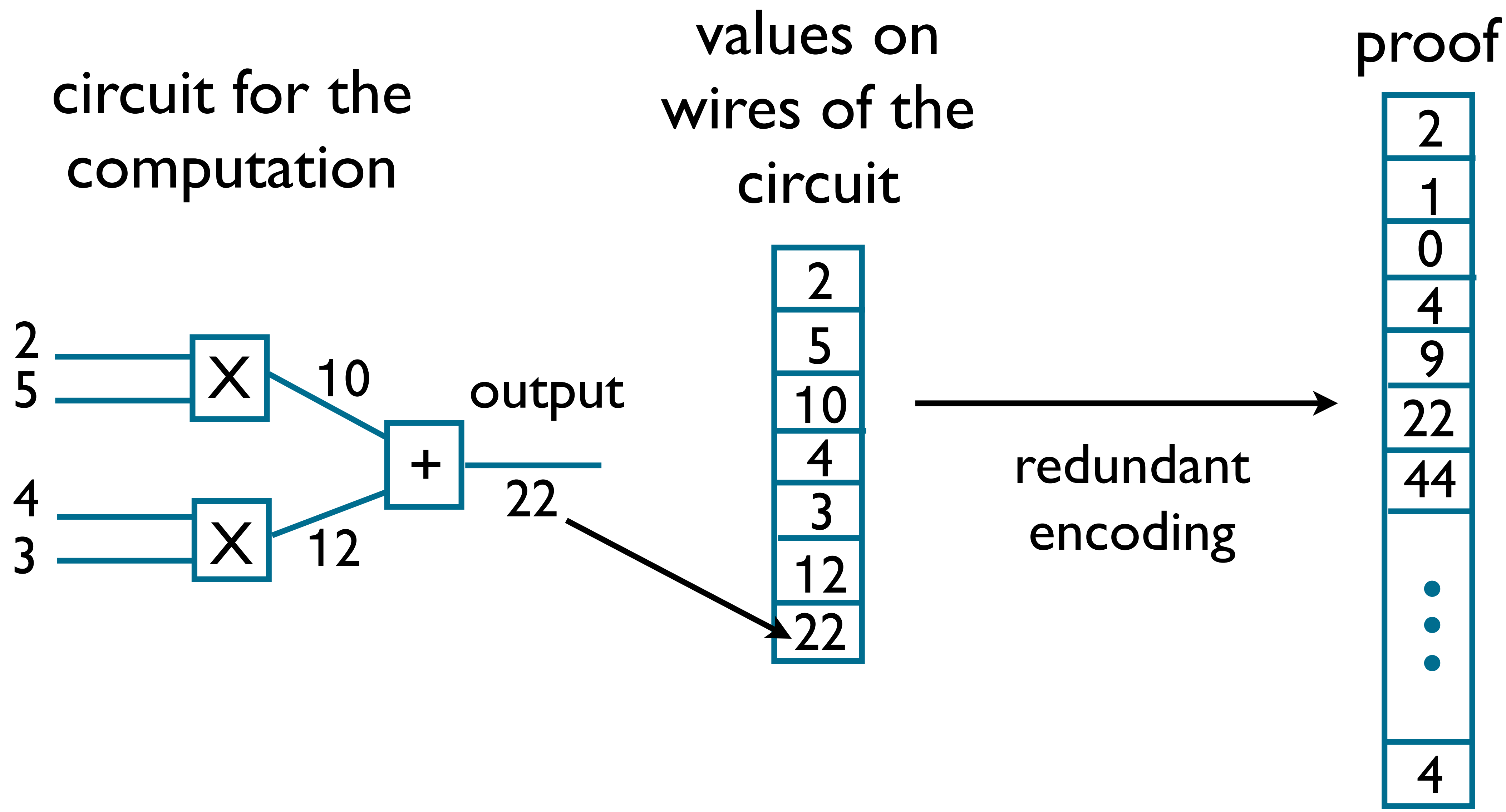
→ Design of GINGER

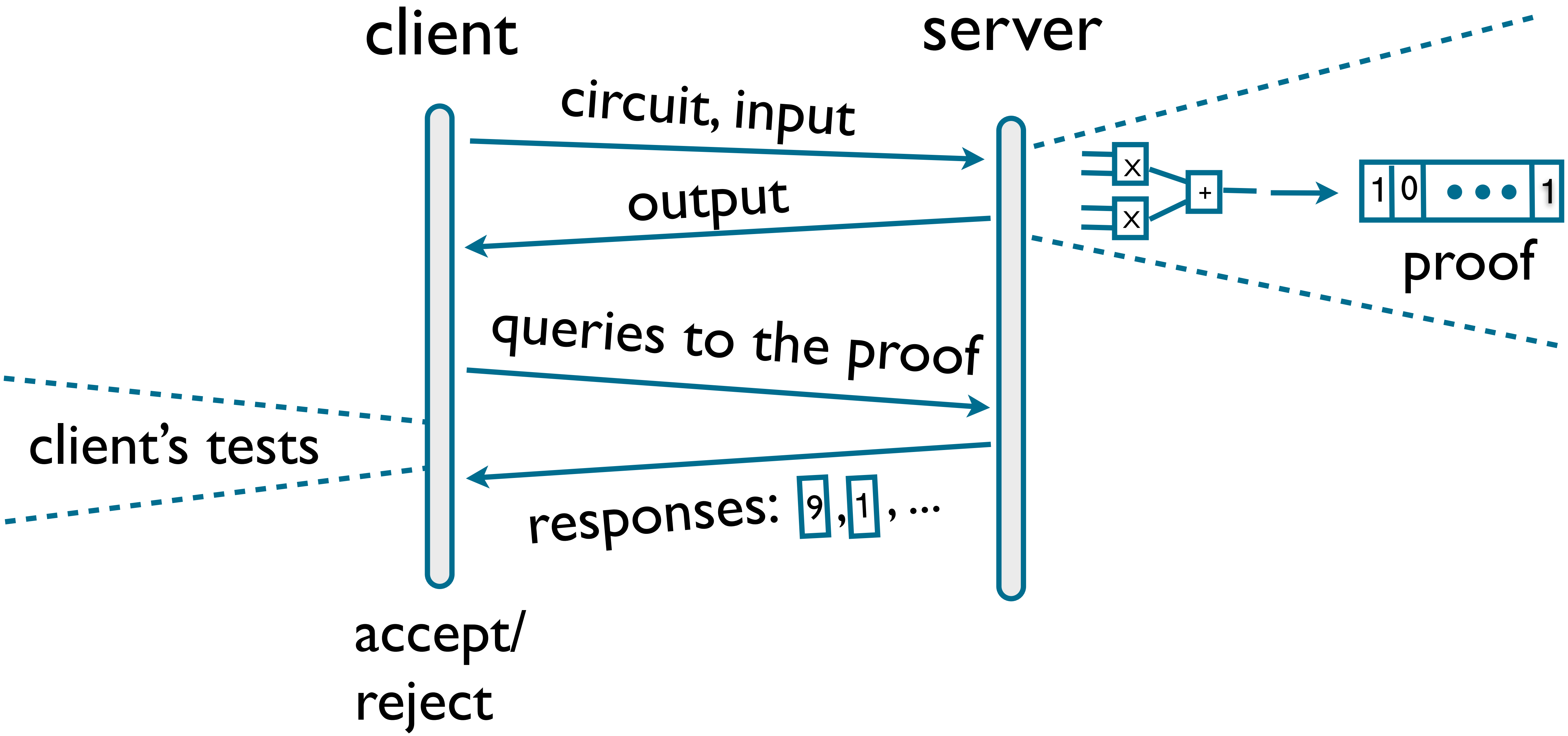
Experimental results

Theory that GINGER builds on



The server creates a proof by redundantly encoding the circuit's wire values



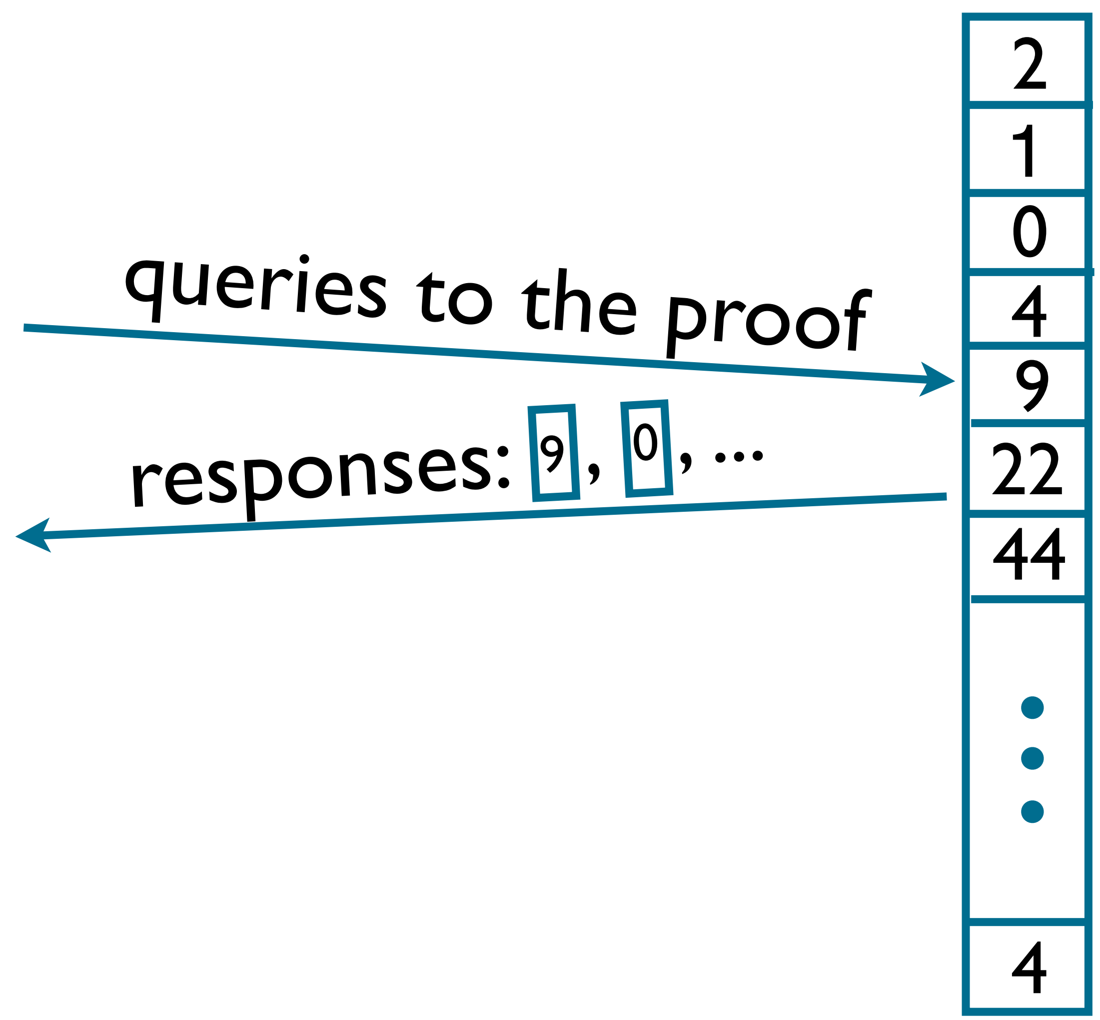

client
server
circuit, input
output
queries to the proof
responses: 9 , 1 , ...
client's tests
accept/
reject
×
×
+
1 0 1
proof

The client queries the server's proof and runs a set of tests

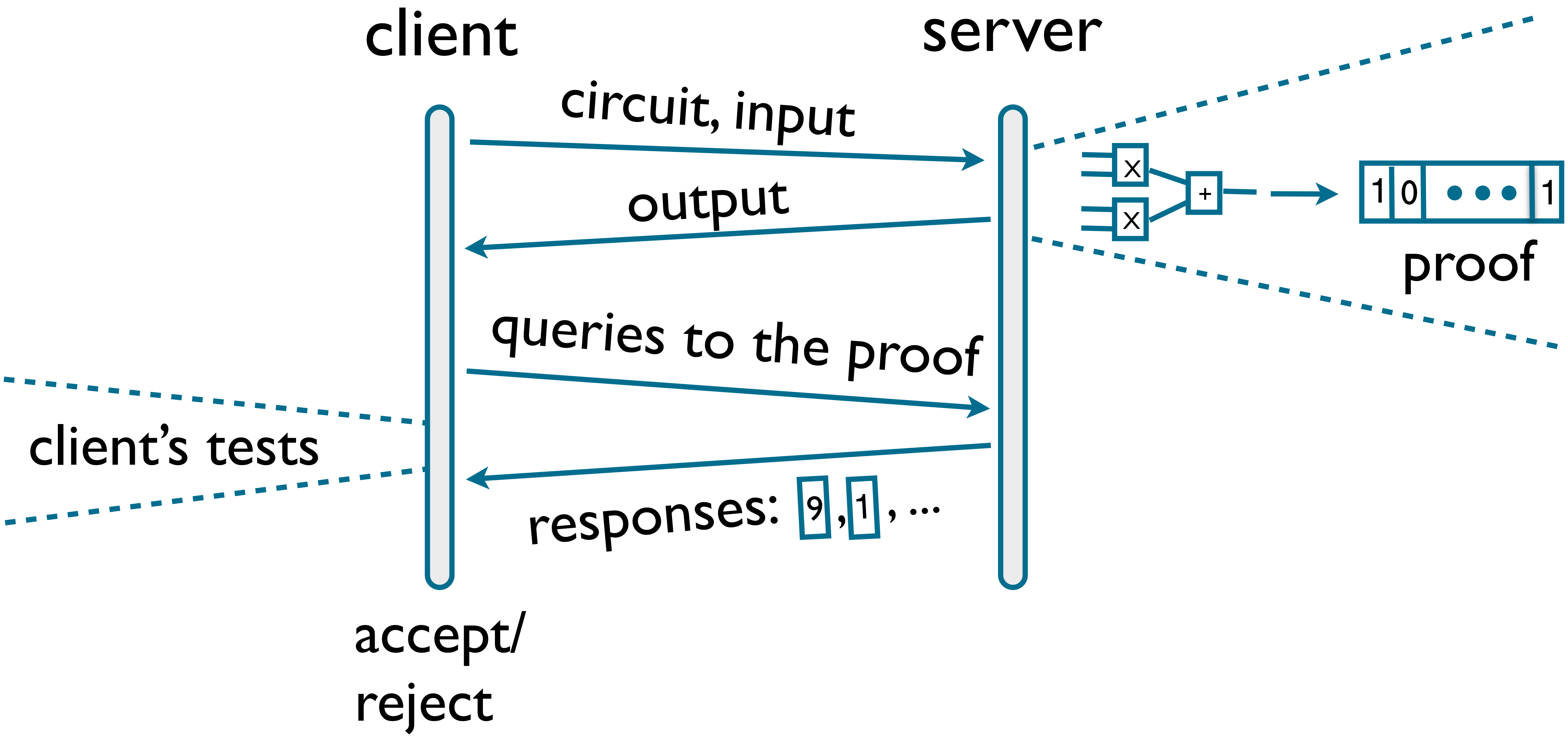
proof at
the server

tests at the client

1. consistency test
2. linearity test
3. quadratic corr. test
4. circuit test



accept/
reject



There is some probability that the client accepts an incorrect proof

The costs depend on the size of the circuit

GINGER's contributions include:

→ Reducing the costs by revisiting the client's tests

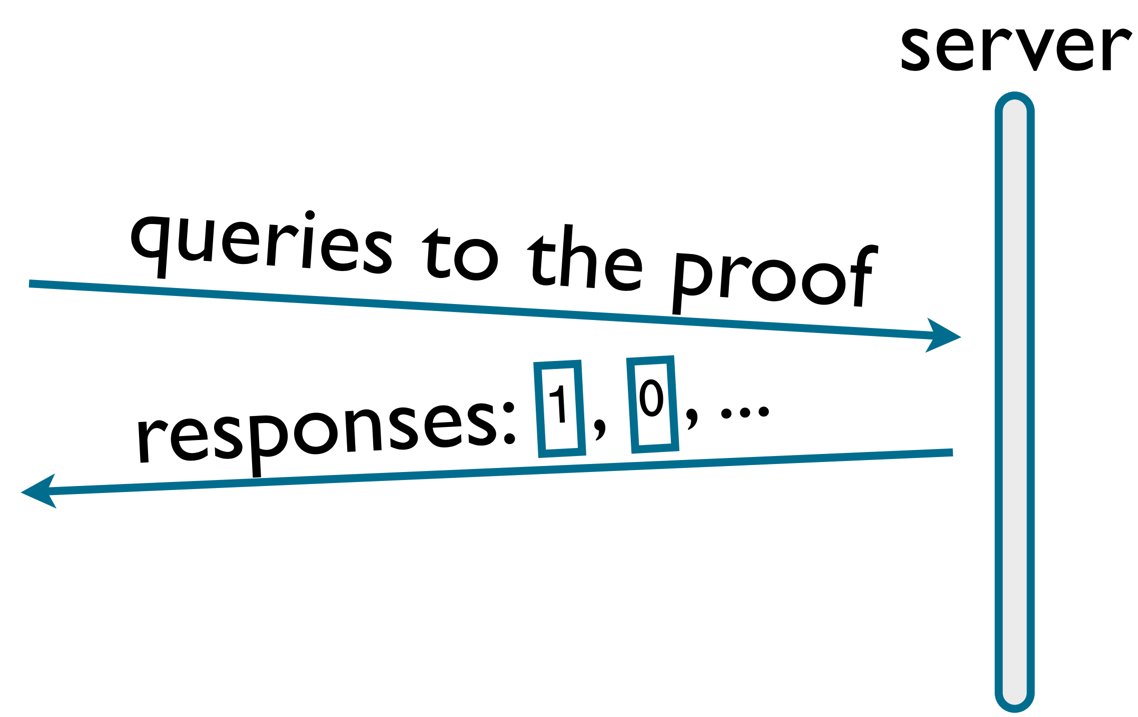
Broadening the space of computations

Incorporating primitive floating-point numbers (in the paper)

Reducing the costs by revisiting the client's tests

client's tests

1. consistency test
2. linearity test
3. quadratic corr. test
4. circuit test



Modifications:

Trade off more queries that are cheap for fewer of a more expensive type

Reuse queries across tests, and compress queries

Benefit: savings in client's checking costs and network costs

GINGER's contributions include:

✓ Reducing the costs by revisiting the client's tests

⟶ Broadening the space of computations

Incorporating primitive floating-point numbers (in the paper)

We change the model of computation from arithmetic circuits to systems of equations

The new model can represent general-purpose programming constructs concisely

End-to-end costs decrease by many orders of magnitude

An example

increment(X)
$Y = X + 1$

=

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

Once the inputs are fixed, an incorrect output will result in an inconsistent system of equations

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

We can encode many program constructs

For example, consider "X != Y":

Our equation is 1 = (X - Y) * M

Observe: no solution if X = = Y

Another example with conditional control flow

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

=

$$0 = X - M$$

$$Y = M * 3 + (1-M) * 4$$

Compiling code into a system of equations

equations for <, >, !=, IF/ELSE, ...

(in the paper)

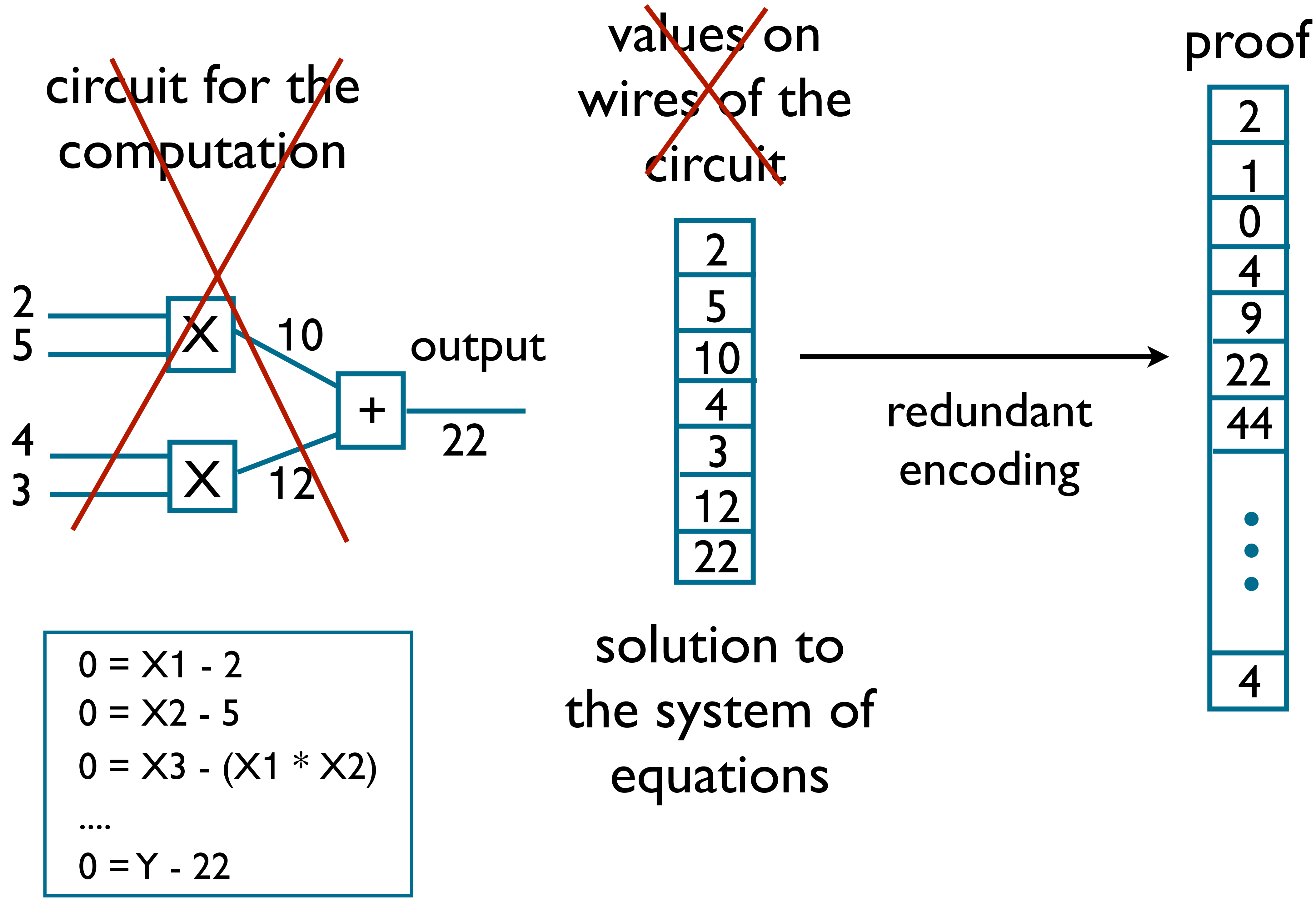
```
function(bool X)
  if (X)
   Y = 3
  else
   Y = 4
```

compiler based on Fairplay [Malkhi et al. USENIX Security04]

$0 = X - M$

$Y = M * 3 + (1-M) * 4$

The server creates a proof by redundantly encoding a solution to the system of equations



GINGER's contributions include:

- ✓ Reducing the costs by revisiting the client's tests

- ✓ Broadening the space of computations

- Incorporating primitive floating-point numbers (in the paper)

Implementation and experimental testbed

Massively parallel implementation

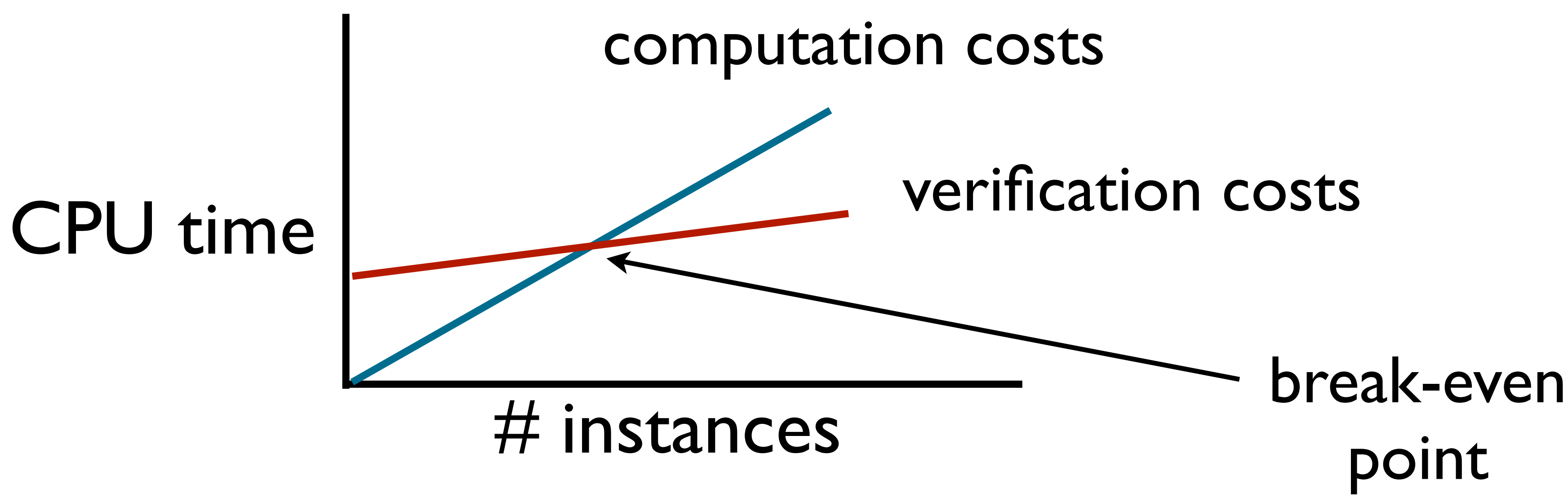
- C++ code with OpenMP threads; HTTP/Open MPI to distribute server's work
- CUDA to offload work to GPUs

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.
- For GPU experiments, we use NVIDIA Tesla M2070 GPUs (448 CUDA cores and 6GB of memory)

Evaluation questions

1. What are the break-even points under GINGER?



2. What is the result of parallelizing the server?
3. What are the savings from using systems of equations as opposed to circuits?

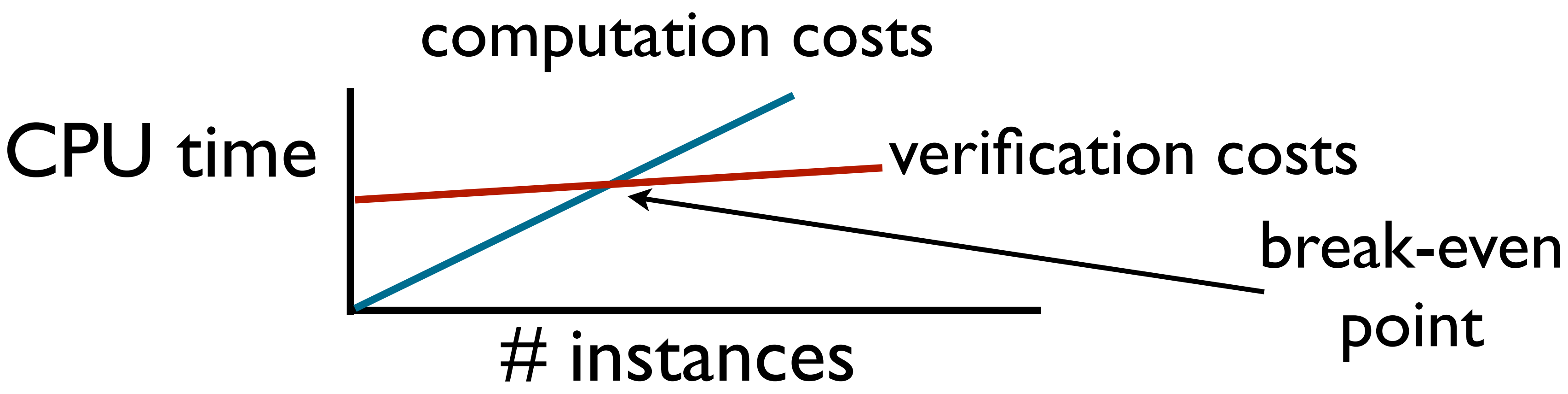
The break-even points decrease significantly

Consider outsourcing many instances of 400×400 matrix multiplication

		PEPPER	GINGER
CPU	break-even # instances	4500	1800
CPU	client verification time	5.3 hours	2.1 hours
GPU for crypto	break-even # instances	3600	1300
GPU for crypto	client verification time	4.3 hours	1.5 hours

Evaluation questions

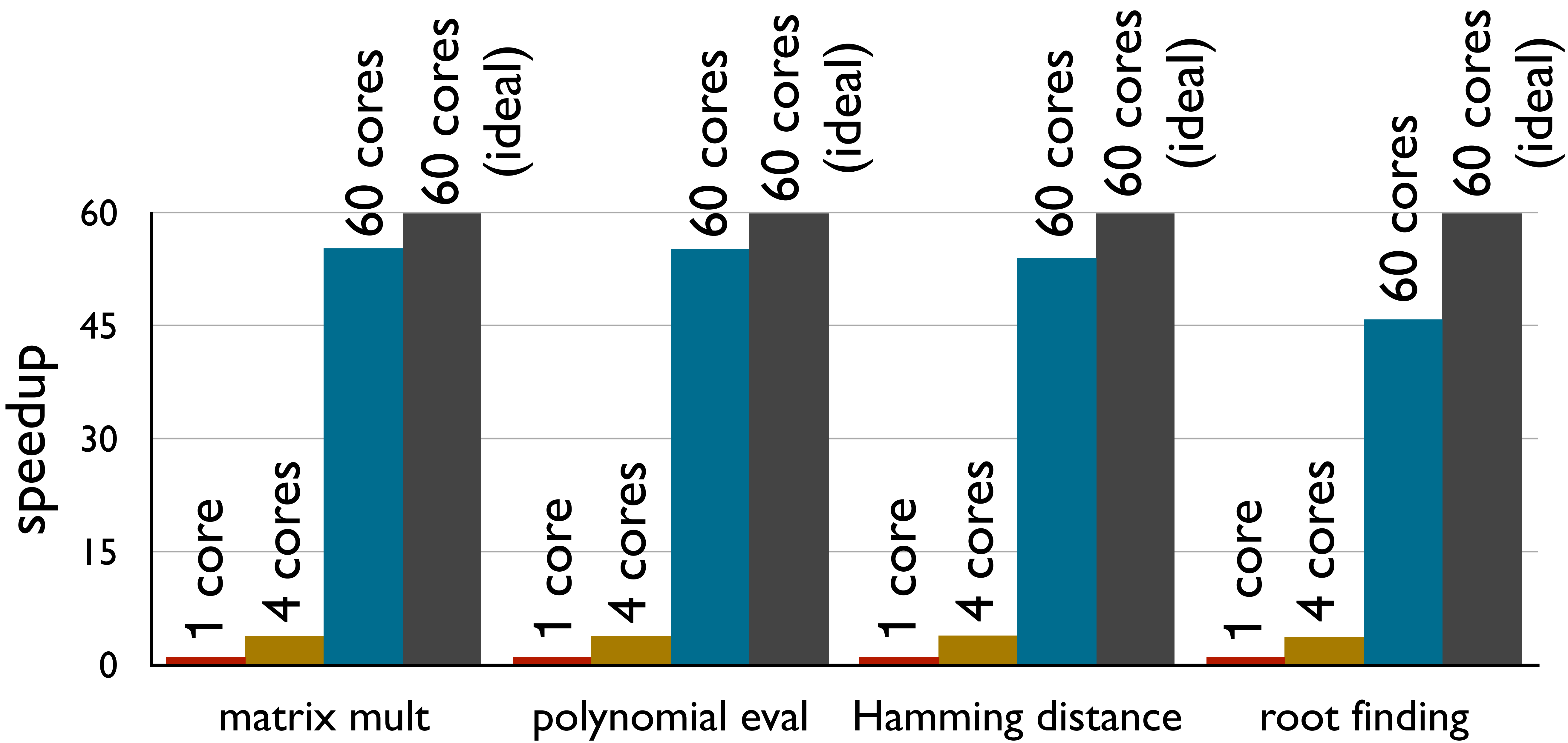
1. What are the break-even points under GINGER?



2. What is the result of parallelizing the server?

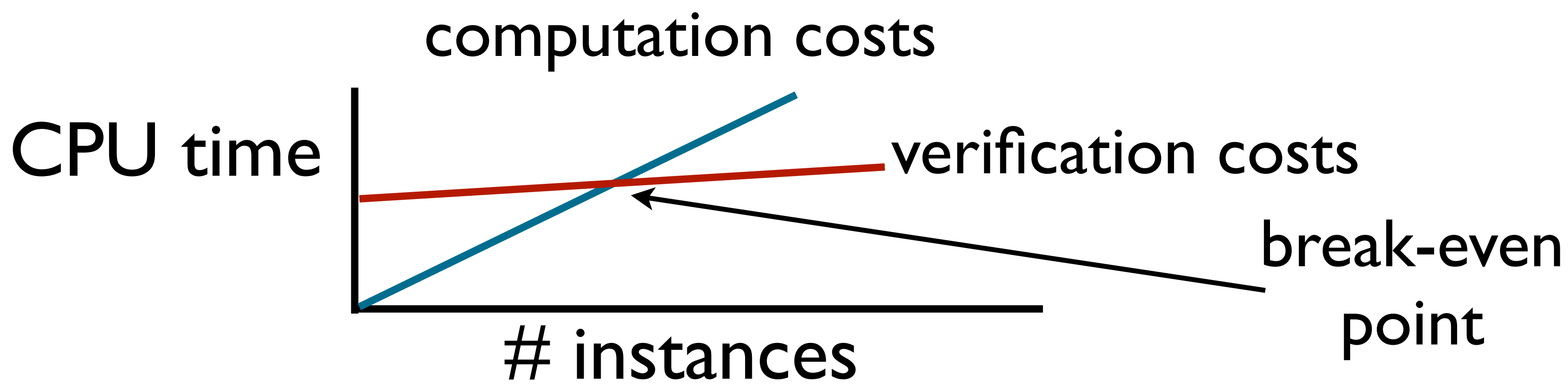
3. What are the savings from using systems of equations as opposed to circuits?

Parallelizing the server results in a near-linear speedup in most cases



Evaluation questions

1. What are the break-even points under GINGER?



2. What is the result of parallelizing the server?

3. What are the savings from using systems of equations as opposed to circuits?

GINGER's representation is many orders of magnitude shorter compared to Boolean circuits

Benchmark	# gates in Boolean circuit	# variables in GINGER's representation
root finding via bisection	3×10^8	2×10^3
Hamming distance	10^6	2×10^4

Rest of this talk

✓ Design of GINGER

✓ Experimental results

⟶ Limitations, related work, and outlook

Limitations of GINGER

The client needs to outsource many instances to gain

The server's resource costs are still high

- Also, the efficiency of the server sometimes relies on reducing the redundancy in the proof's encoding

The number of iterations in a loop should be known at compile time

Prior work on verifying computations

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]

Use fully homomorphic encryption [Gennaro et al. CRYPTO10, Chung et al. CRYPTO10]

Proof-based verified computation [Ben-Or STOC88, Babai STOC91, Kilian STOC92, Blum et al. JACM95, Arora et al. JACM98, Ben-Sasson et al. 12, Gennaro et al. 12]

Built systems:

- Toward practical interactive proofs [Cormode ITCS12, Thaler et al. HotCloud12] based on [Goldwasser et al. STOC08]

- Our prior work: PEPPER [HotOS11, NDSS12] based on [Ishai et al. CCC07]

Summary of GINGER

Reduces the client’s checking work and network costs by several orders of magnitude

Includes a massively parallel GPU-based implementation

Supports a general-purpose programming model

Looking back

About two years ago, we set out to build a system for proof-based verified computation

Then, the estimated costs were on the order of trillions of CPU years

Main takeaway

We combined theory and systems techniques to reduce costs by a factor of 10^{20}

We still need a factor of $\approx 10^3$ on the server for true practicality

But we think that proof-based verified computation could be practical for real in the near future