Time-Optimal Interactive Proofs for Circuit Evaluation

Justin Thaler

Harvard University → Simons Institute for the Theory of Computing

Outsourcing

- Many applications require outsourcing computation to untrusted service providers.
  - Main motivation: commercial cloud computing services.
  - Also, weak peripheral devices; fast but faulty co-processors.
  - Volunteer Computing (SETI@home, World Community Grid, etc.)

- User requires a guarantee that the cloud performed the computation correctly.

AWS Customer Agreement

WE… MAKE NO REPRESENTATIONS OF ANY KIND … THAT THE SERVICE OR THIRD PARTY CONTENT WILL BE UNINTERRUPTED, ERROR FREE OR FREE OF HARMFUL COMPONENTS, OR THAT ANY CONTENT … WILL BE SECURE OR NOT OTHERWISE LOST OR DAMAGED.

Interactive Proofs

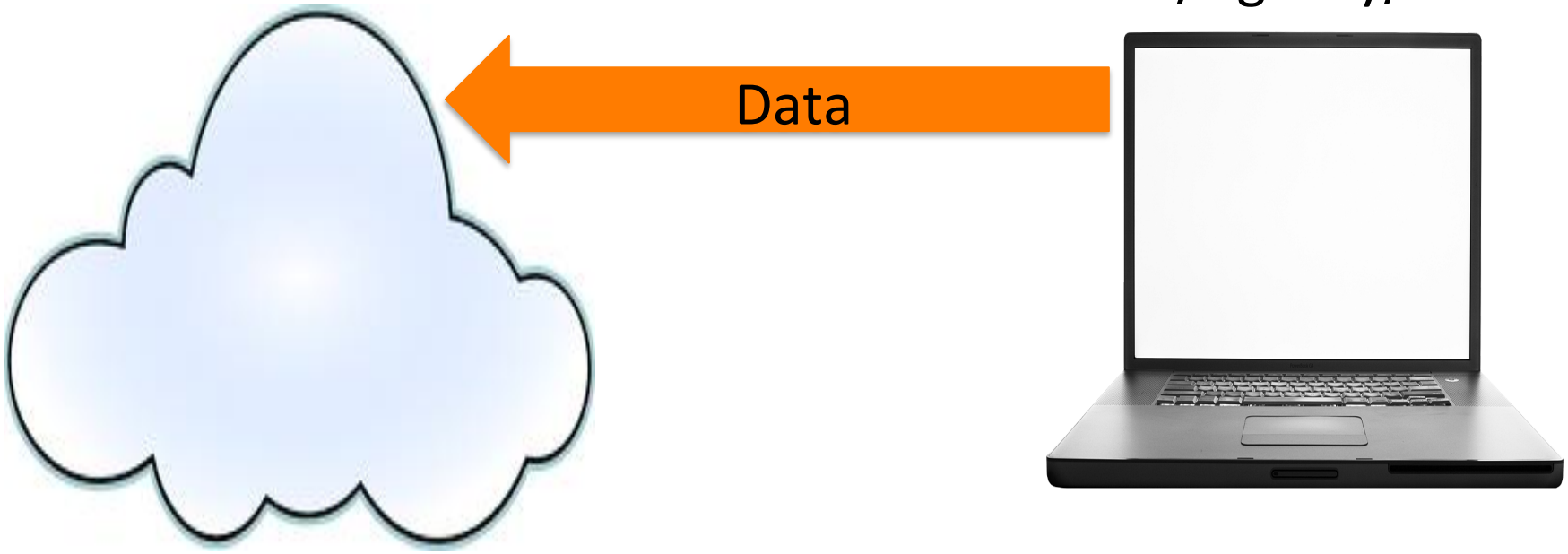
Cloud Provider

Business/Agency/Scientist

Interactive Proofs

Cloud Provider

Business/Agency/Scientist

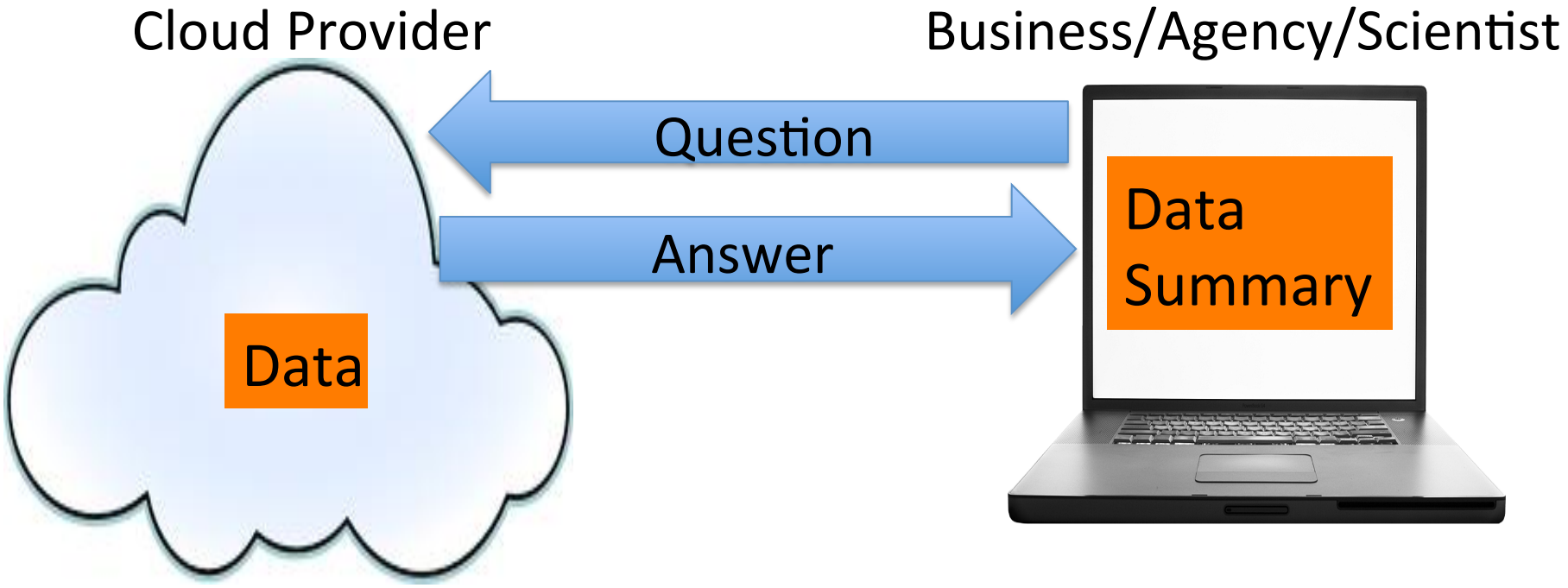


Interactive Proofs

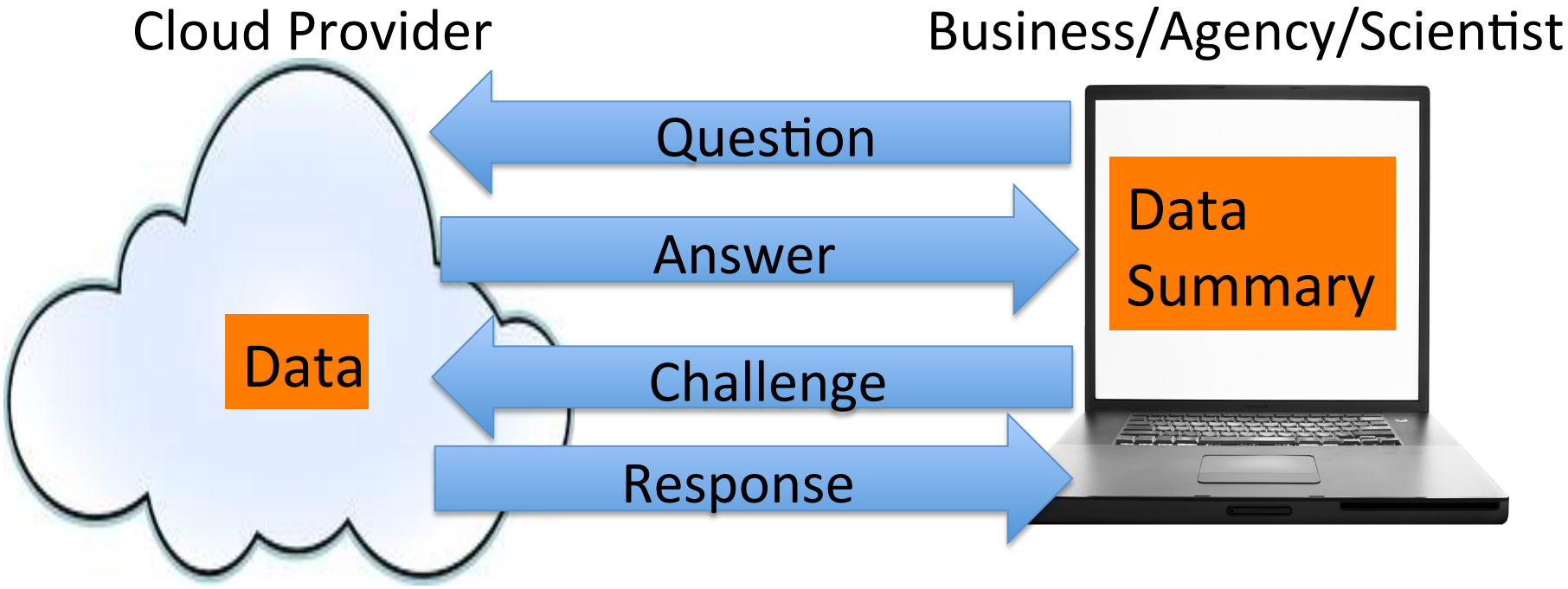
Cloud Provider

Business/Agency/Scientist

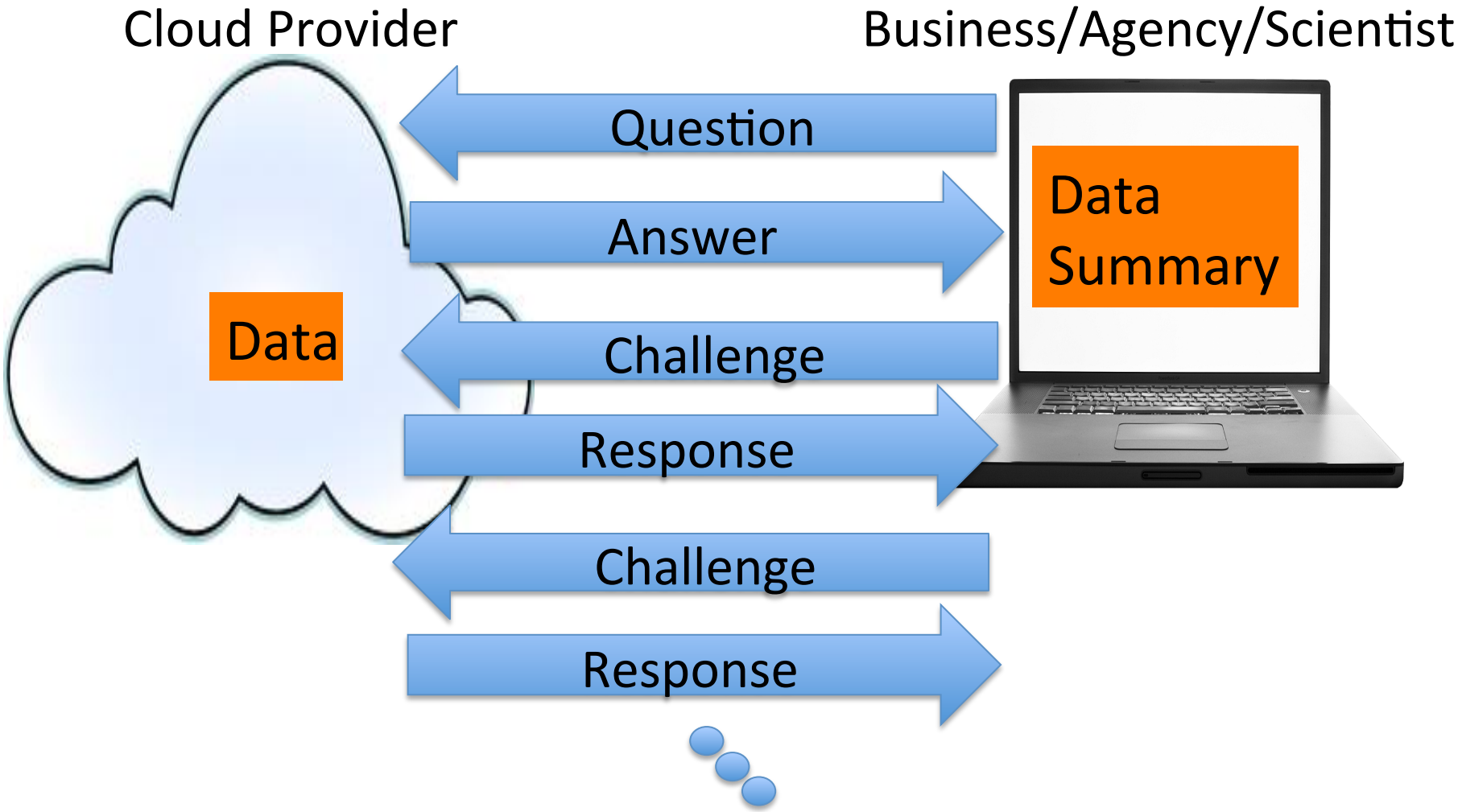
Interactive Proofs



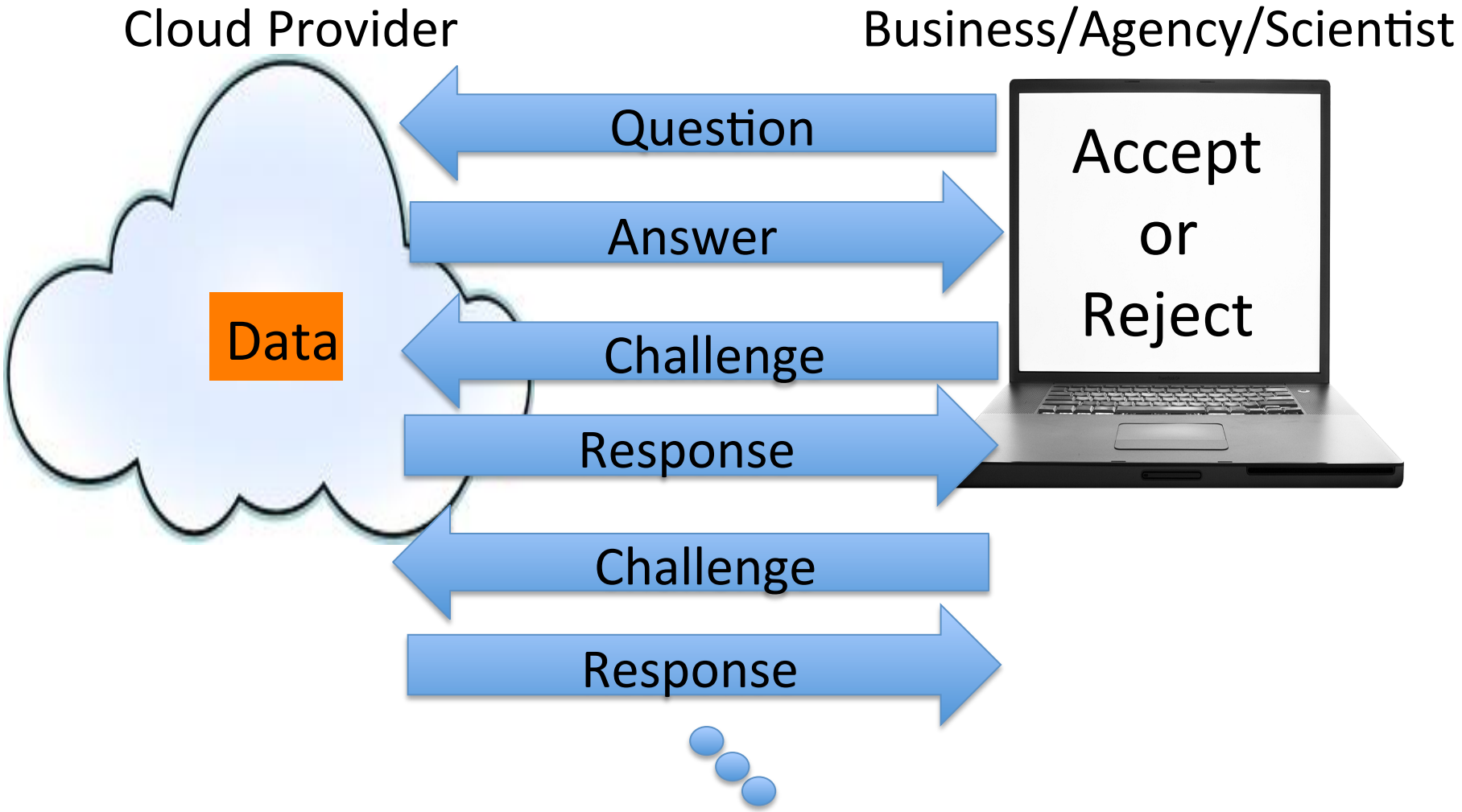
Interactive Proofs



Interactive Proofs



Interactive Proofs



Interactive Proofs

- Prover P and Verifier V.

- P solves problem, tells V the answer.
  - Then P and V have a conversation.
  - P's goal: convince V the answer is correct.

- Requirements:
  - 1. Completeness: an honest P can convince V to accept.
  - 2. Soundness: V will catch a lying P with high probability (secure even if P is computationally unbounded).

Prior Work: [GKR08, CMT12]

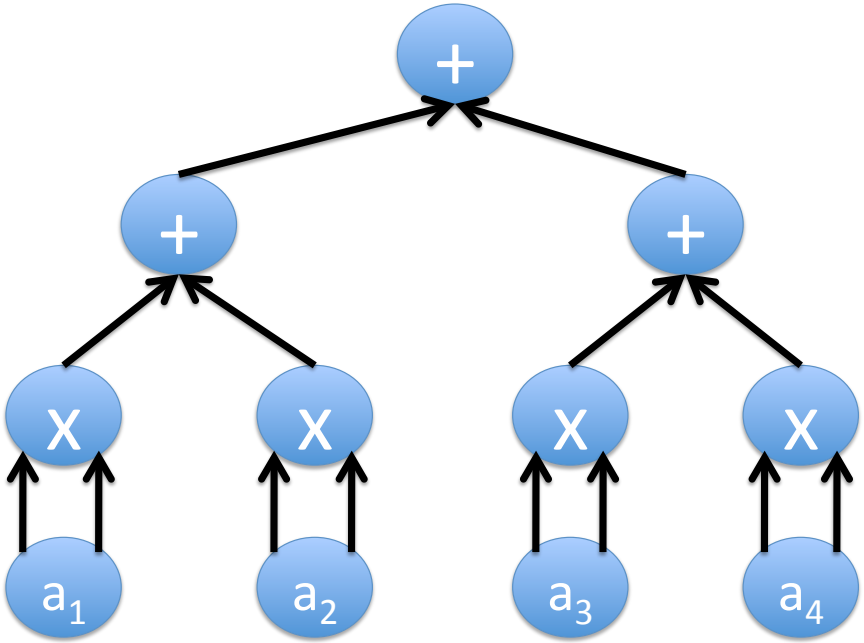
The GKR Protocol

- Interactive Proofs for Muggles [GKR 08] gives a highly efficient protocol for problems in NC.
  - Allows V to run **very** quickly, so outsourcing is useful even though problems are "easy".
  - P needs "only" polynomially more time to prove correctness than she does to just solve the problem!

The GKR Protocol

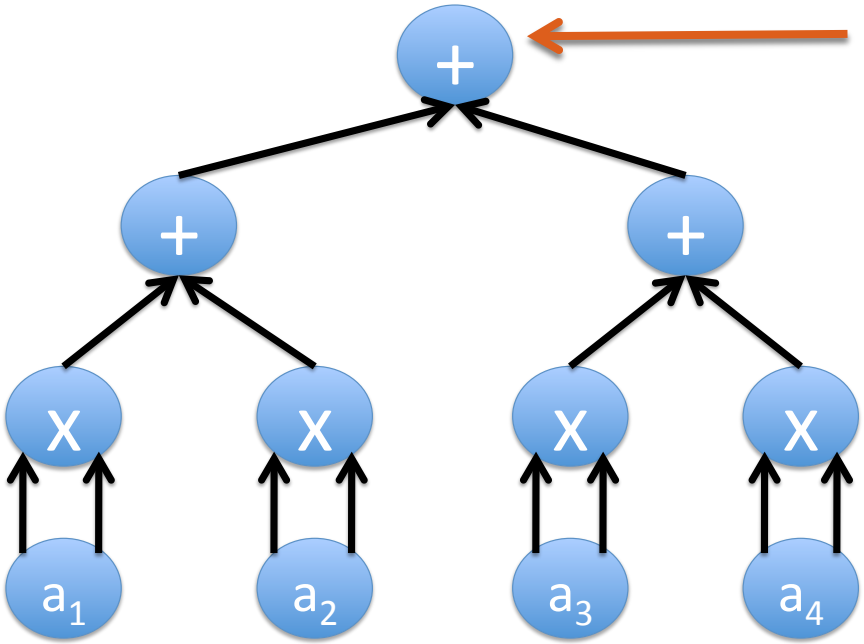
- Why does GKR **not** yield a practical protocol out of the box?
  - P has to do a lot of extra bookkeeping (**cubic** blowup in runtime).

The GKR Protocol: Overview



F_2 circuit

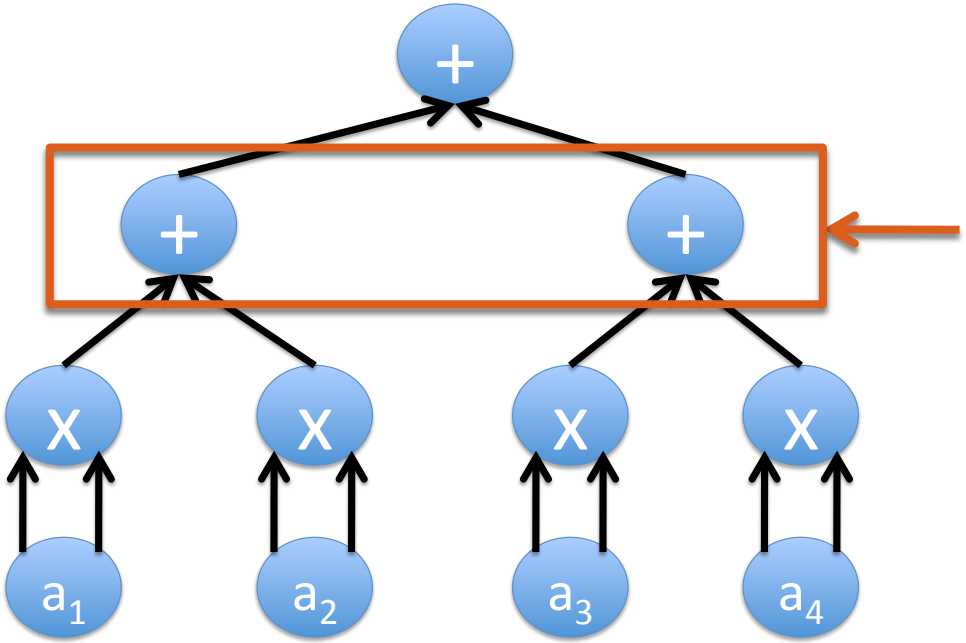
The GKR Protocol: Overview



P starts the conversation with an answer (output).

F_2 circuit

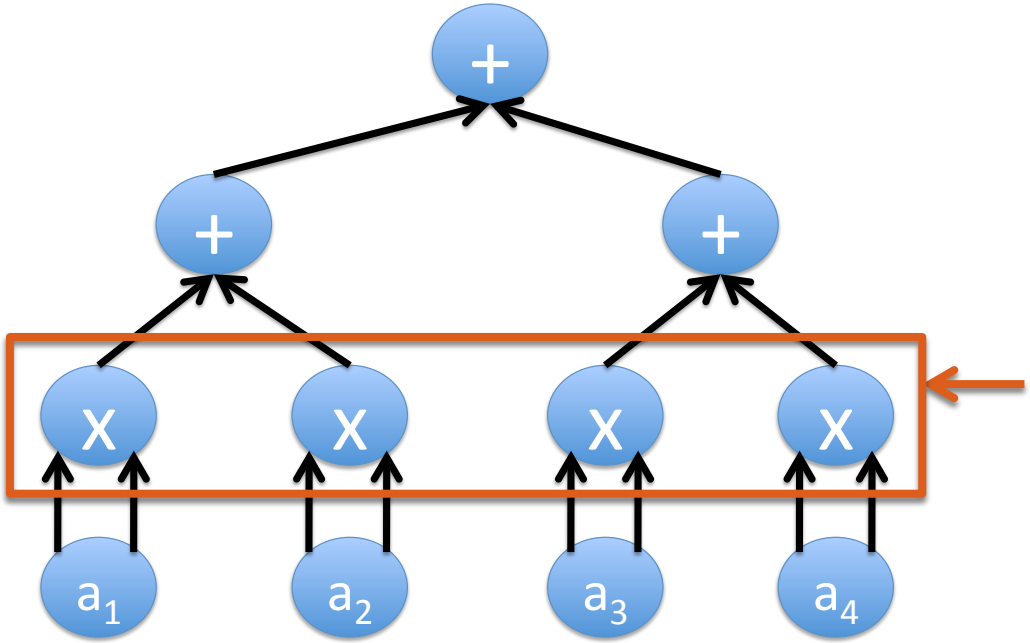
The GKR Protocol: Overview



V sends series of challenges. P responds with info about next circuit level.

F_2 circuit

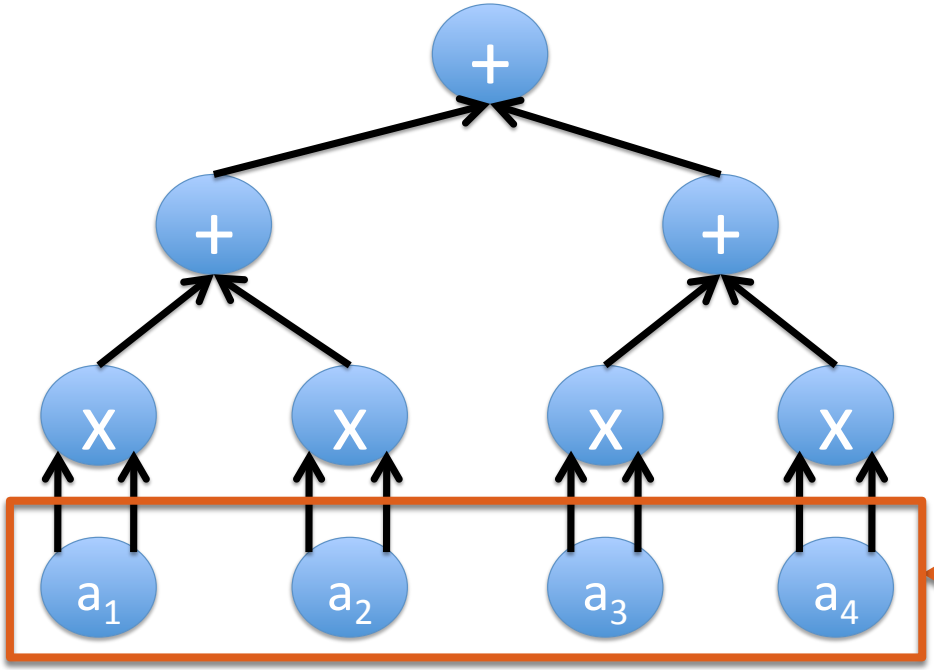
The GKR Protocol: Overview



Challenges continue, layer by layer down to the the input.

F_2 circuit

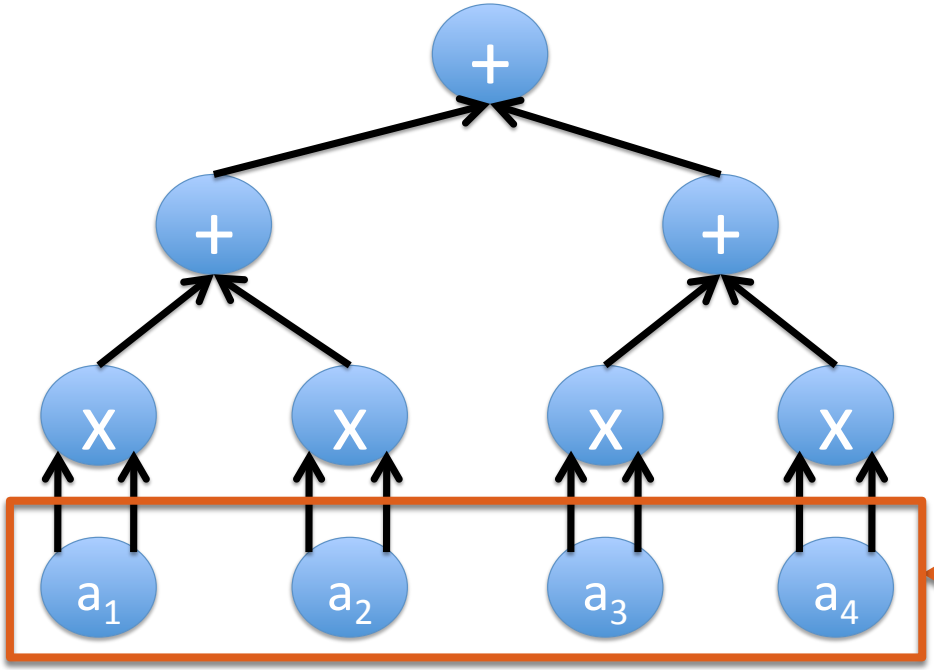
The GKR Protocol: Overview



F_2 circuit

Finally, P says something about the (multilinear extension of the) input.

The GKR Protocol: Overview



F_2 circuit

Finally, P says something about the (multilinear extension of the) input.

V sees input directly, so can check P's final statement directly.

Overview of [CMT12]

- Implemented the GKR protocol (with refinements).
- Demonstrated very low concrete costs for V.
- Brought P's runtime down from $\Omega(S^3)$, to $O(S \log S)$, where S is circuit size.
  - Key insight: use **multilinear** extension of circuit within the protocol.
  - Causes enormous cancellation in P's messages, allowing fast computation.

Overview of [CMT12]

- Implemented the GKR protocol (with refinements).
- Demonstrated very low concrete costs for V.
- Brought P's runtime down from $\Omega(S^3)$, to $O(S \log S)$, where S is circuit size.
  - Key insight: use **multilinear** extension of circuit within the protocol.
  - Causes enormous cancellation in P's messages, allowing fast computation.
- Still not good enough on its own.
  - P is $\sim 10^3$ times slower than just evaluating the circuit.
  - Naïve implementation of GKR would take trillions of times longer.

This Work: Slashing Costs for Structured Computation

Reducing Overhead Further

- Downsides to [CMT12] implementation:
  - For "regular" circuits: log S factor runtime overhead for P.
  - For "irregular" circuits: log S factor runtime overhead for P, and expensive pre-processing phase for V.

Result 1: Regular Circuits

Reducing Overhead Further

- For "regular" circuits: Reduce P's runtime from O(S log S) to O(S).
  - **Key idea: use new arithmetization of the circuit, allowing P to reuse work across rounds.**
  - Experimental results: 250x speedup over [CMT12].
  - P less than 10x slower than a C++ program that just evaluates the circuit.
  - Example applications: MatMult, DISTINCT, F_2, Pattern Matching, FFTs.

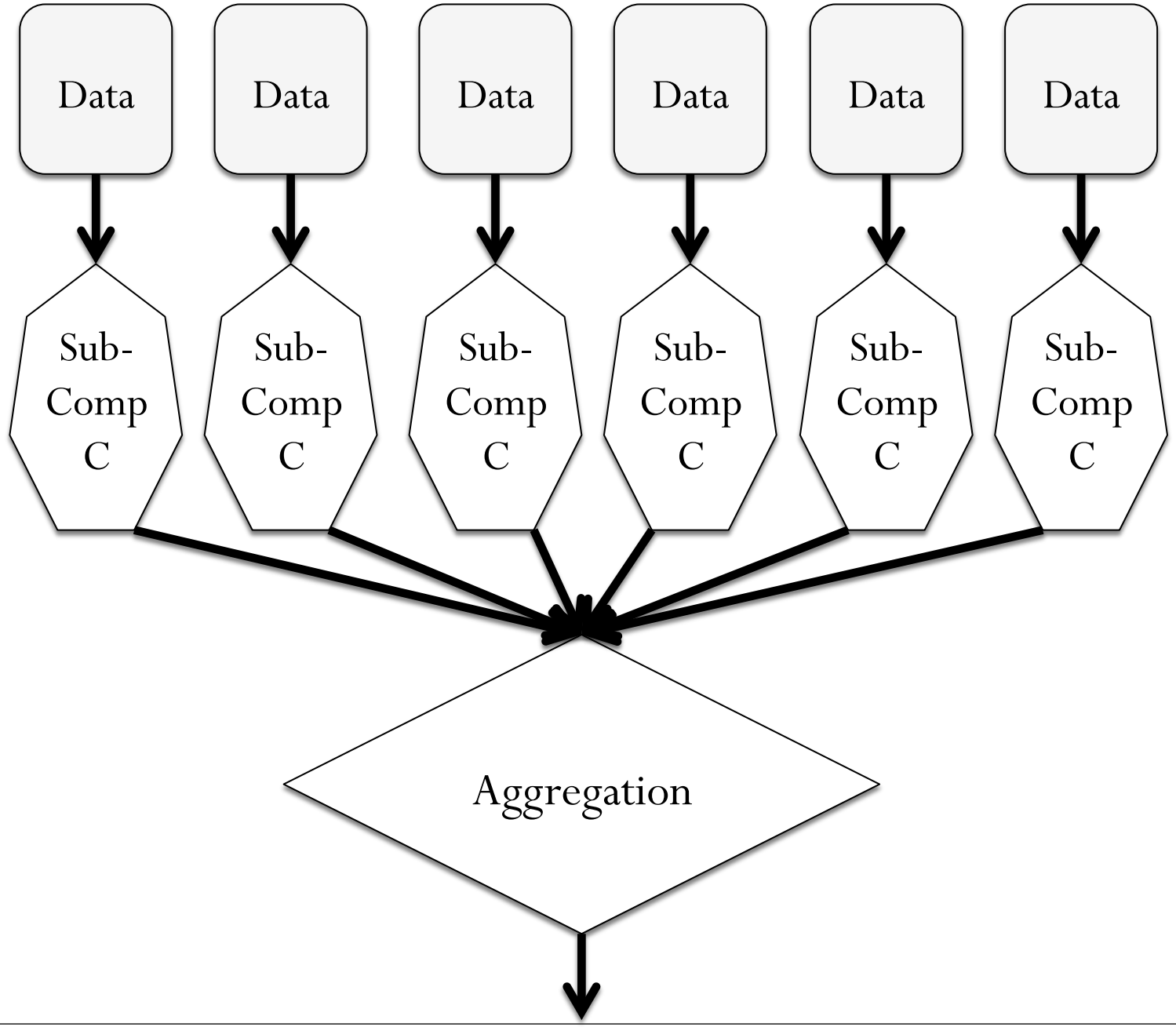
Results for Regular Circuits

Problem	P time [CMT12]	P time [T13]	Circuit Eval Time	Rounds [T13]	Protocol Comm [T13]	V time [Both]
DISTINCT ($n=2^{20}$)	56.6 minutes	17.2 s	1.88 s	236	40.7 KB	.2 s
MatMult (512 x 512)	2.7 hours	37.8 s	6.07 s	1361	5.4 KB	.1 s

Result 2: Data Parallel Computation

Dealing with Irregular Circuits

- No magic bullet for dealing with irregular wiring patterns.
  - Need *some* assumption about the computation being outsourced.
  - Is there structure in real-world computations?
- Yes: Data Parallel computation.
  - Any setting where a sub-computation C is applied to many pieces of data.
  - Make no assumptions about C itself.
  - These are the sort of problems getting outsourced!


Data
Data
Data
Data
Data
Data
Sub-
Comp
C
Sub-
Comp
C
Sub-
Comp
C
Sub-
Comp
C
Sub-
Comp
C
Sub-
Comp
C
Aggregation

Leveraging Parallelism

- Directly applying existing results to data parallel computations has big overhead.
  - Costs depend on number of data pieces.
- Our approach: take advantage of parallelism.
  - Reduce V's effort to proportional to size of C.
  - Reduce P's overhead to log size of C.
  - No dependence on number of data pieces.
- **Key insight: C may be irregular internally, but the computation is maximally regular between copies of C.**

Result 3: Matrix Multiplication

A Final Result: n x n MatMult

- P simply sends V the "right answer", and then P does $O(n^2)$ extra work to prove its correctness.
- Doesn't matter how P obtains the right answer!
- Optimal runtime up to leading constant assuming no $O(n^2)$ time algorithm for MatMult.

A Final Result: n x n MatMult

- P simply sends V the "right answer", and then P does $O(n^2)$ extra work to prove its correctness.
- Doesn't matter how P obtains the right answer!
- Optimal runtime up to leading constant assuming no $O(n^2)$ time algorithm for MatMult.

Problem Size	Naïve MatMult Time	Additional P time	V Time	Rounds	Protocol Comm
1024 x 1024	2.17 s	0.03 s	0.67 s	11	264 bytes
2048 x 2048	18.23 s	0.13 s	2.89 s	12	288 bytes

Comparison to Freivalds' Algorithm

- Freivalds (MFCS, 1979) gave the following protocol for MatMult. To check AB=C:
  - V picks random vector x.
  - Accepts if A*(Bx) = Cx.
  - **No** extra work for P, $O(n^2)$ time for V.
- Our big win: verifying algorithms that invoke MatMult, but aren't really interested in matrices.
  - E.g. Best-known graph diameter algorithms square the adjacency matrix, but are only interested in a single number.
  - Total communication for us is $O(\log^2 n)$, Freivalds' is $\Omega(n^2)$.

Summary

- [CMT12] gives a general-purpose interactive proof protocol with near-practical costs for verifying any small-depth computation.
- We slash costs for more structured computations (regular circuits, data parallel, matrix multiplication)
- Major message: the more structure in computation, the faster it can be verified.
  - And this structure exists in real-world computations!

Our Results in Context: Related Work

Work on Argument Systems

- Substantial body of recent work implements argument systems with pre-processing for circuit evaluation.
  - [SMBW12, SVP+12, B-SCGT13, GGPR13, SVB+13, PHGR13, BFR+13, B-SCGT+13]
- Advantages of our approach:
  - Secure against computationally unbounded provers.
  - No or minimal pre-processing for large classes of computation.
  - Unmatched prover efficiency when applicable.
- Disadvantages of our approach:
  - Only applicable to small-depth circuits.
  - No support for "non-deterministic circuits" (see next talk).
  - Logarithmically many rounds of interaction.

Other Work on Argument Systems

- [B-SCGT13a, B-SCGTb] develops argument systems based on short PCPs.
  - This approach **never** requires pre-processing for verifier.
  - But likely introduces substantial additional concrete overhead for prover.

Future Directions

1. Identify IPs that **avoid circuit representation** for a larger class of computational primitives.

2. Upcoming work [TVWB13]: a two-prover MIP implementation extending GKR techniques.
   - High-level message: disadvantages of GKR approach (no deep circuits, no non-deterministic circuits) go away in two-prover setting!
   - In turn, MIPs can be compiled into single-prover argument systems [BC12]
   - Disclaimer: current transformations are based on impractical primitives (FHE).

Thank you!

Sum-Check Protocol

- Given: a *d*-variate polynomial g over field **F**
- Sum-check protocol computes the quantity:

$$\sum_{x \in \{0,1\}^d} g(x_1, \ldots, x_d)$$

- Costs:
  - # of rounds: d
  - Time cost for V: d + [time to evaluate g at a point]
  - Time cost for P: at most $O(2^d)$ * [time to evaluate g at a point]

Set-Up

- Given $n \times n$ input matrices A, B over field **F**, interpret A and B as functions mapping $\{0,1\}^{\log n} \times \{0,1\}^{\log n}$ to **F** via:

$$A(i_1,\ldots,i_{\log n},j_1,\ldots,j_{\log n}) = A(i,j)$$

- Let $\tilde{A}, \tilde{B}$: $\mathbf{F}^{\log n} \times \mathbf{F}^{\log n} \rightarrow \mathbf{F}$ denote the multilinear extensions of the functions A and B.

- Let C=A*B. Then the multilinear extension of C satisfies:

$$\tilde{C}(i_1,\ldots,i_{\log n},j_1,\ldots,j_{\log n}) = \sum_{k \in \{0,1\}^{\log n}} \tilde{A}(i_1,\ldots,i_{\log n},k_1,\ldots,k_{\log n}) * \tilde{B}(k_1,\ldots,k_{\log n},j_1,\ldots,j_{\log n})$$

Matrix Multiplication Protocol

- P sends a matrix D claimed to equal A*B.
- V evaluates $\tilde{D}$ at a random point $(r_1,\ldots,r_{\log n},r'_1,\ldots,r'_{\log n}) \in \mathbf{F}^{2\log n}$
- Schwartz-Zippel lemma implies that it is safe for V to believe that D equals the correct answer C as long as

$$\tilde{D}(r_1,\ldots,r_{\log n},r'_1,\ldots,r'_{\log n}) = \tilde{C}(r_1,\ldots,r_{\log n},r'_1,\ldots,r'_{\log n})$$

- So V applies the sum-check protocol to compute:

$$\sum_{k\in\{0,1\}^{\log n}} g(k_1,\ldots,k_{\log n}),$$

where

$$g(k_1,\ldots,k_{\log n}) = \tilde{A}(r_1,\ldots,r_{\log n},k_1,\ldots,k_{\log n}) * \tilde{B}(k_1,\ldots,k_{\log n},r'_1,\ldots,r'_{\log n})$$

Matrix Multiplication Protocol

- In final round of sum-check protocol, V must evaluate g at a random point $(r''_1,\ldots,r''_{\log n}) \in \mathbf{F}^{\log n}$.
- Crucial observation: V can do this in time $O(n^2)$ by evaluating $\tilde{A}(r_1,\ldots,r_{\log n},r''_1,\ldots,r''_{\log n})$ and $\tilde{B}(r''_1,\ldots,r''_{\log n},r'_1,\ldots,r'_{\log n})$ and using the identity:

$$g(r''_1,\ldots,r''_{\log n}) = \tilde{A}(r_1,\ldots,r_{\log n},r''_1,\ldots,r''_{\log n}) * \tilde{B}(r''_1,\ldots,r''_{\log n},r'_1,\ldots,r'_{\log n})$$

- Using our "reuse of work" technique, P has to do $O(n^2)$ work on top of finding the right answer C in order to run the sum-check protocol.

Goals of Verifiable Computation

- Provide user with correctness guarantee, without requiring her to perform full computation herself.
  - Ideally user will not even maintain a local copy of the data (all of our protocols allow verifier to make single streaming pass over input).
- Minimize extra effort required for cloud to provide correctness guarantee.
- Achieve protocols secure against malicious clouds, but lightweight for use in benign settings.

Independent Results

- Recent efforts to build practical *argument systems*.
  - Setty, McPherson, Blumberg, Vu, Braun, Parno, Walfish (NDSS12, Security12, EuroSys13)
- Vu et al. (Oakland13) build a system that:
  1. Starts with a high-level programming language.
  2. Automatically compiles any program in the language into an arithmetic circuit.
  3. Decides whether GKR implementation from [CMT12] (plus refinements) or state-of-the-art argument system is more efficient, and runs the better of the two.
- Experimental comparison in [Oakland13] shows [CMT12] significantly faster except for programs with complicated control flow or that are highly sequential.

More Independent Results

- Other argument systems that avoid short PCPs [PGHR13].
- Work towards practical (short) PCPs [B-SCGT13a, B-SCGT13b].
- Refereed games and arguments [CRR11].

More Independent Results

Approach	Pros/Cons
Interactive Proofs (this talk)	Pros: **Most efficient when applicable** (no crypto, minimal pre-processing for V, least overhead for P). Cons: Applies only to parallel computation, does not support 'non-deterministic reductions' to circuits – important for sorting, comparisons, etc.
Argument Systems avoiding short PCPs [Setty et al., Parno et al.]	Pros: **General**. Public verifiability/zero-knowledge properties. Cons: **Big pre-preprocessing costs**, more overhead for P (crypto expensive)
Argument Systems based on short PCPs [Ben-Sasson et al.]	Pros: General. **No pre-preprocessing costs.** Cons: **Much more overhead for P.**

What About "Sparse" Streams? [CCGT13]

- Many streams are over enormous domain sizes (e.g. IPv6 flows)
  - Existing results depend explicitly (though optimally) on n.
  - Want costs to depend on number of data items m, not domain size n.
- Idea: Domain reduction.
  - Ask P to provide 'perfect' hash function g mapping huge domain to small one.
  - Challenges: ensuring that collisions in remapping do not cause errors (need a way for V to 'detect' collisions under g).
  - New protocols that allow P to 'correct' collisions online.
- Bottom line [CCGT13, to be submitted]: near-optimal tradeoffs in terms of m for frequency moments, graph problems, etc.

References

- Cormode, Mitzenmacher, T. (ESA 2010)
- Cormode, T., Yi (VLDB 2012)
- Cormode, Mitzenmacher, T. (ITCS 2012)
- T., Roberts, Mitzenmacher, Pfister (HotCloud 2012)
- Chakrabarti, Cormode, McGregor, T. (ICALP 2009, in submission 2012)
- T. (in submission, 2013)
- Chakrabarti, Cormode, Goyal, T. (ongoing, 2013)

Further Leveraging Parallelism [TRMP12, T13]

- In our protocols, P and V themselves can be parallelized (although V runs quickly even without parallelization).
- Using a GPU, achieved 40x-100x speedups for P, 100x speedups for V.