Verifiable Computing: Between Theory and Practice

Justin Thaler

Georgetown University

Talk Outline

1. The VC Model: Interactive Proofs and Arguments
2. VC Systems: How They Work
3. Survey and Comparison of Existing VC Implementations
4. A Brief History of Interactive Proofs (IPs)
5. Techniques: IPs vs. Other Approaches

Part 1: Model and Motivation

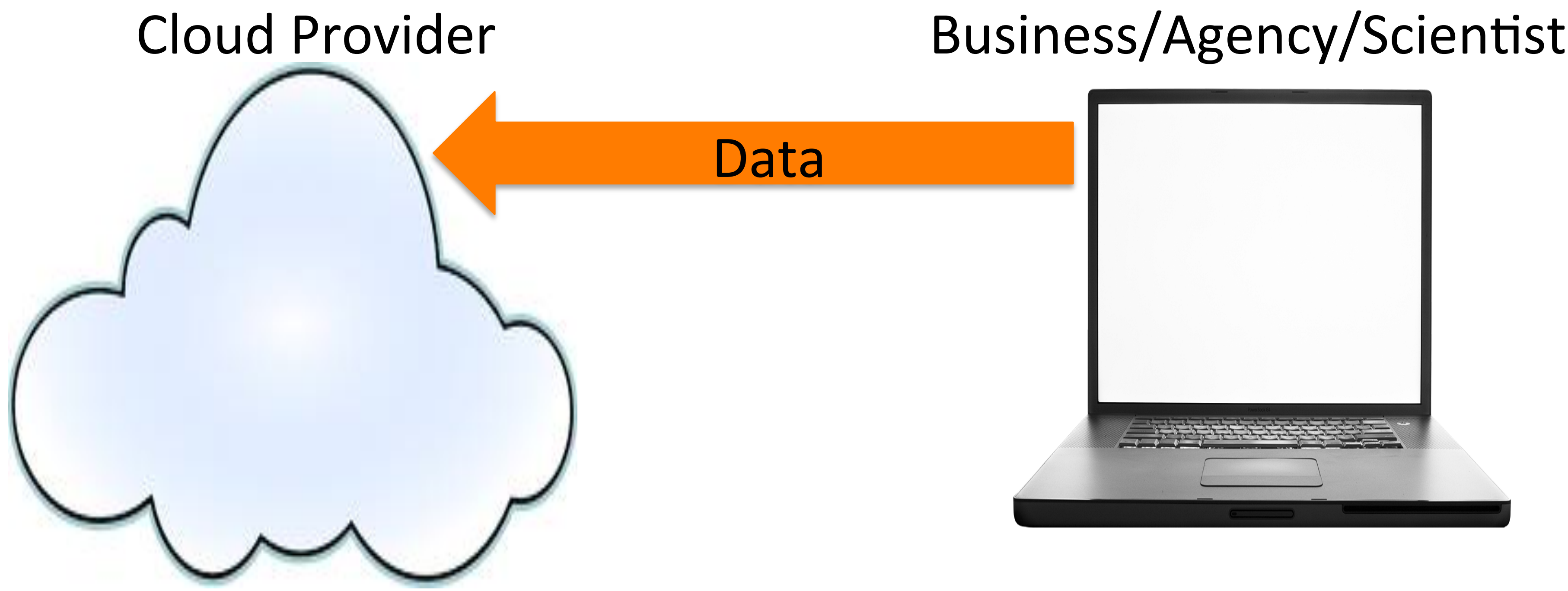
Interactive Proofs (IPs) and Arguments

- Prover P and Verifier V.

1. P solves a problem on a given input.
2. Tells V the answer.
3. Then P **proves** to V that the answer is correct.

- Requirements:
  - Completeness: an honest P can convince V to accept.
  - Soundness: V will catch a lying P with high probability.
    - IPs: information-theoretically sound [GMR1985, Babai 1985]
    - Arguments: sound against polynomial time P's. [BCC 1988]

IPs and Arguments

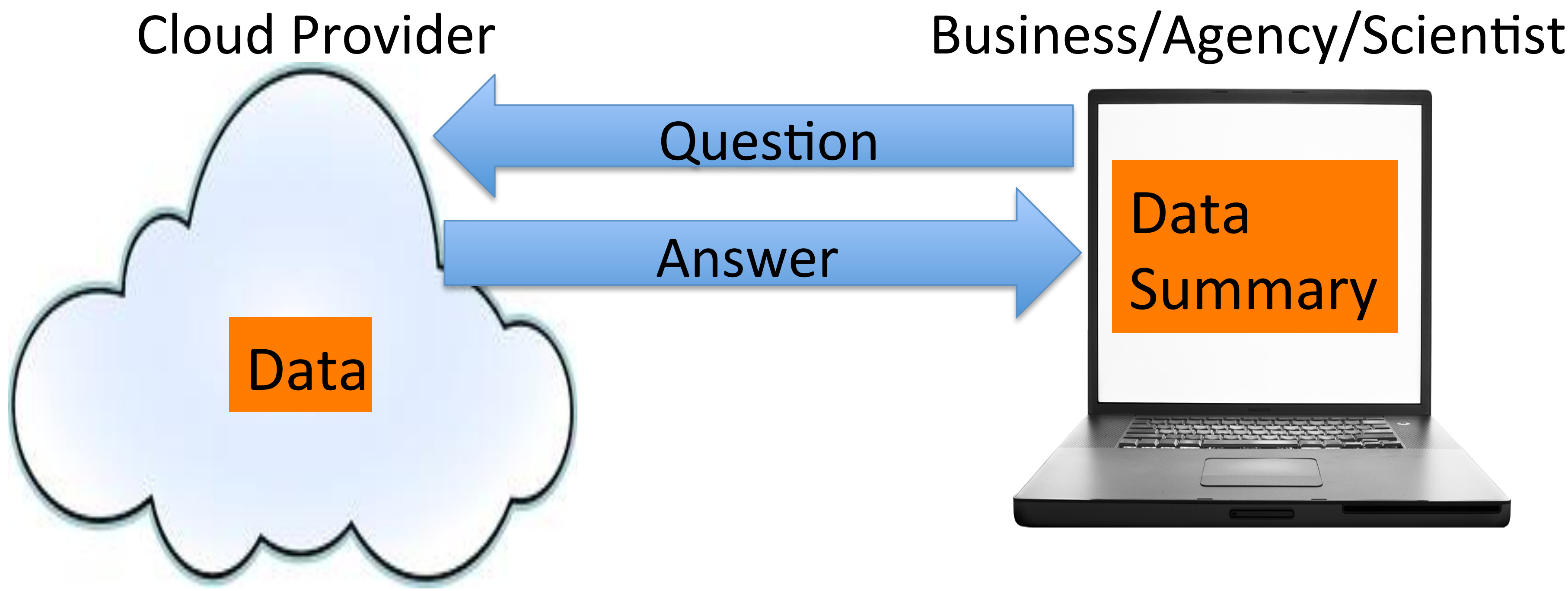


IPs and Arguments

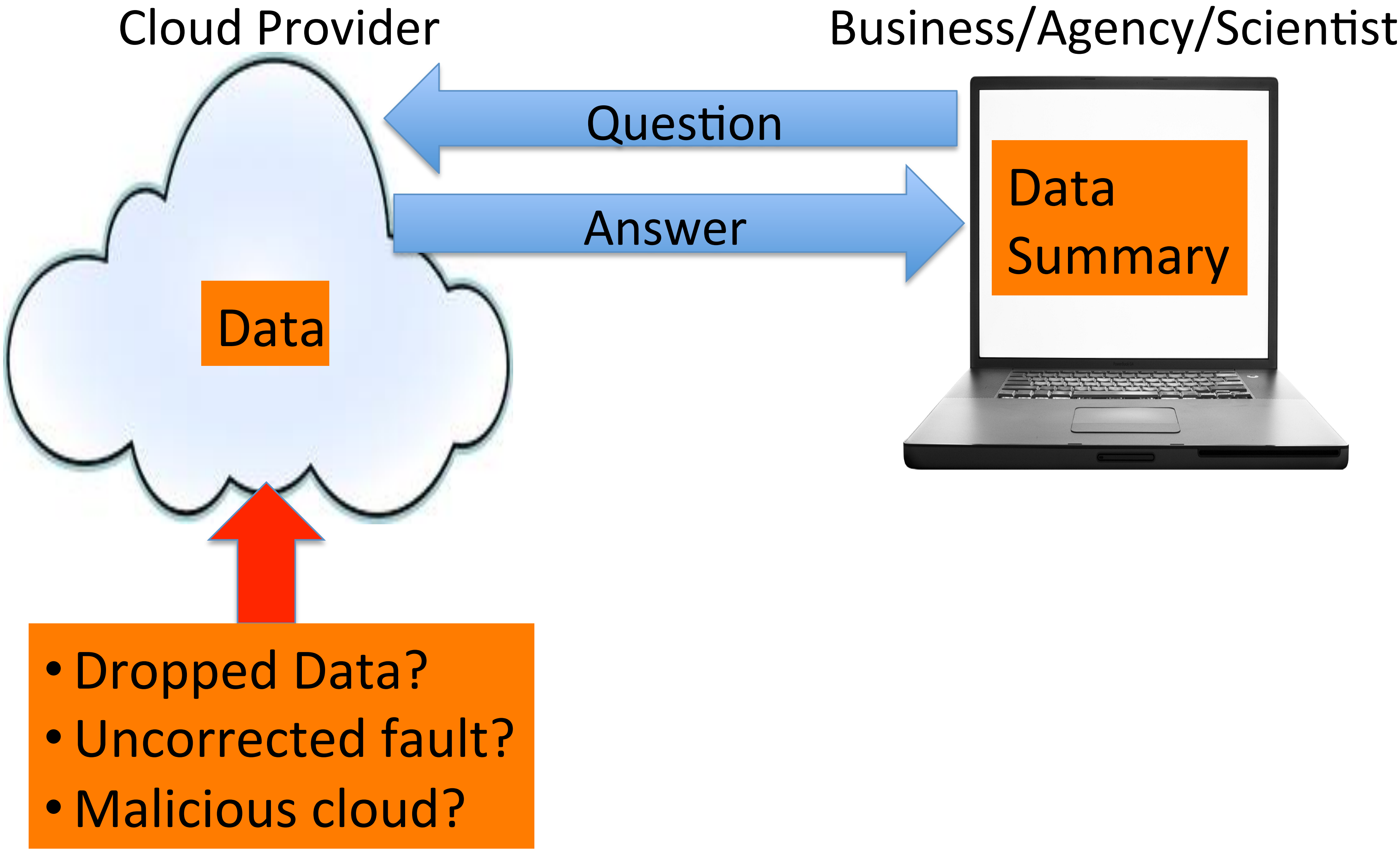
Cloud Provider

Business/Agency/Scientist

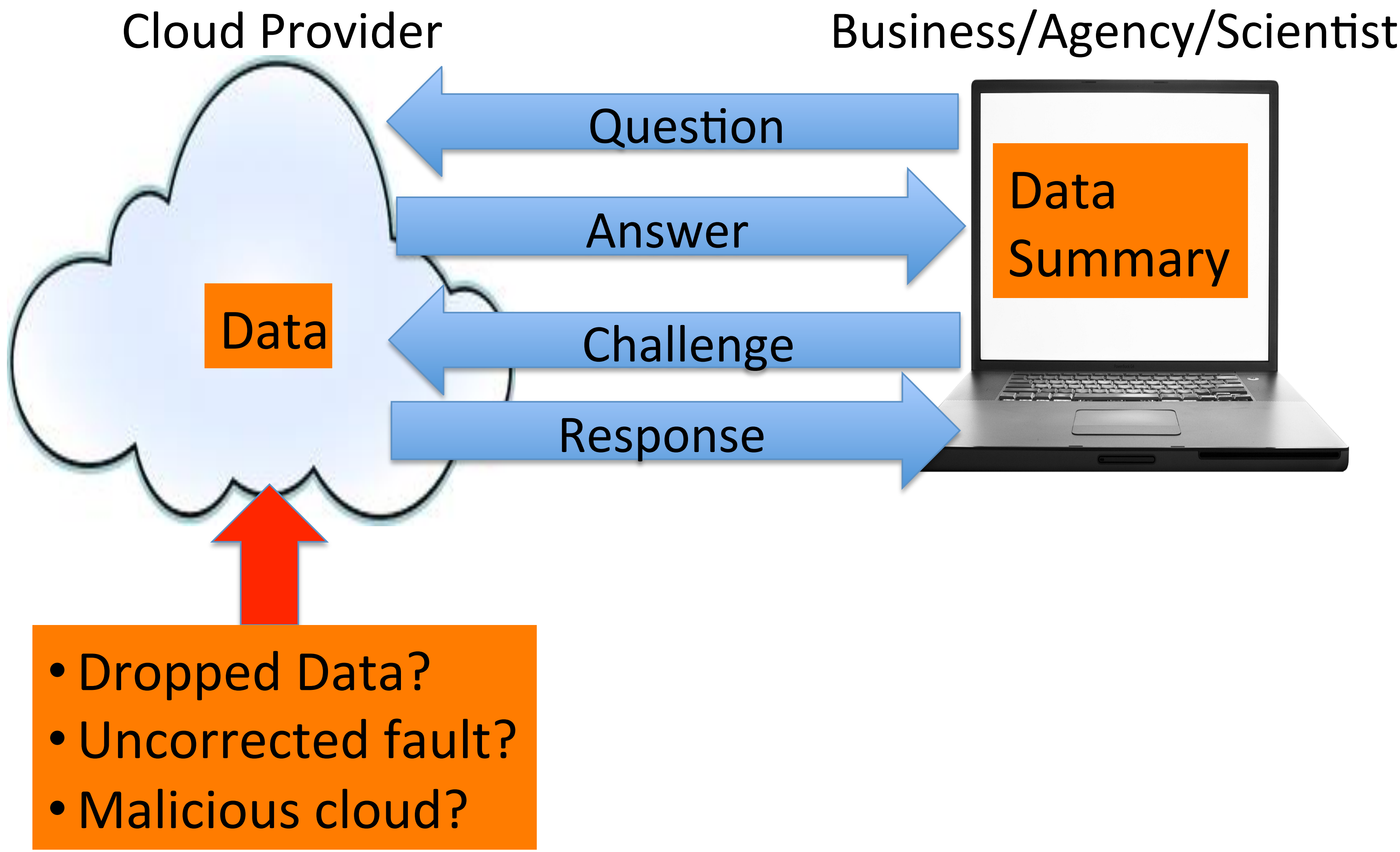
IPs and Arguments



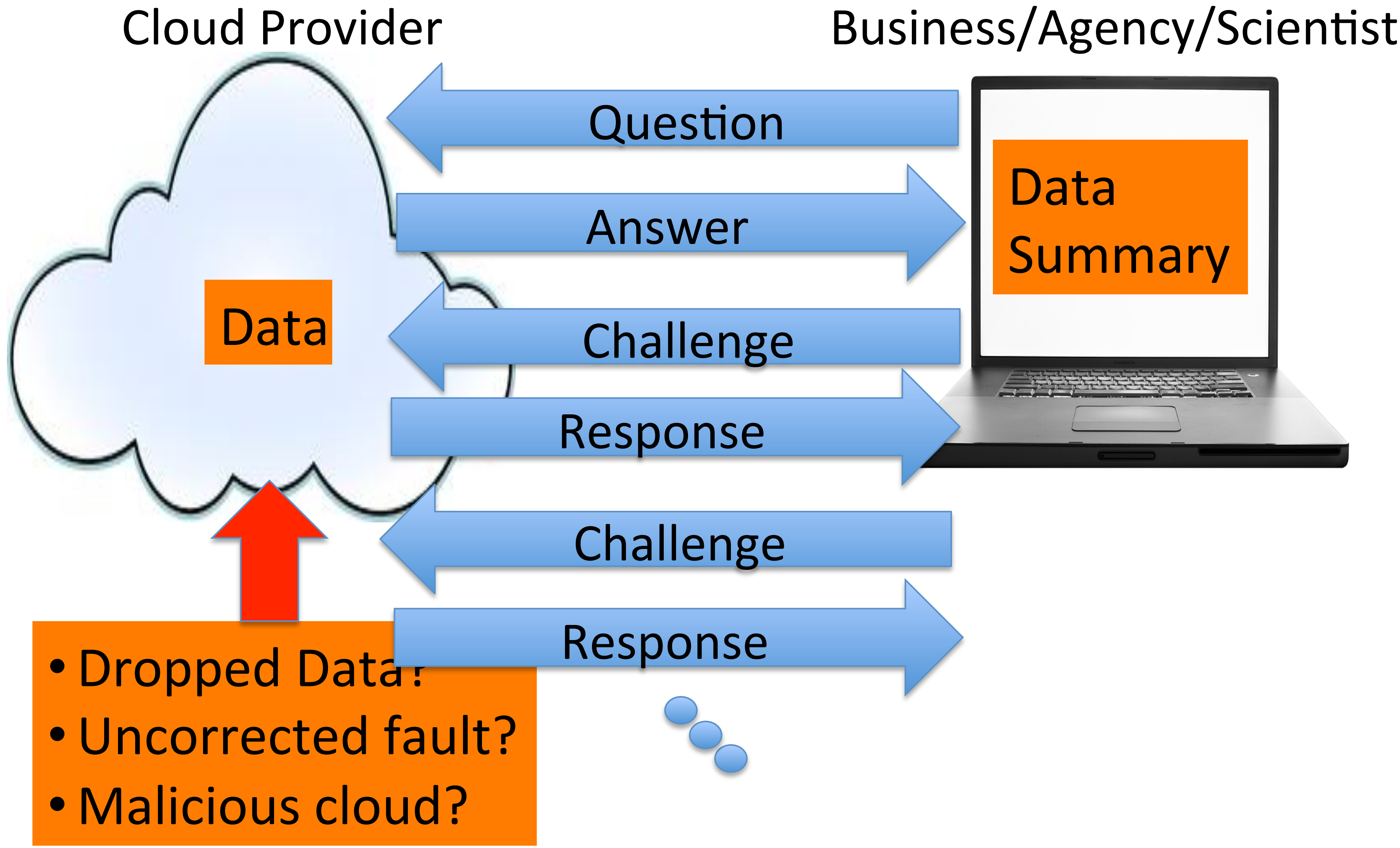
IPs and Arguments



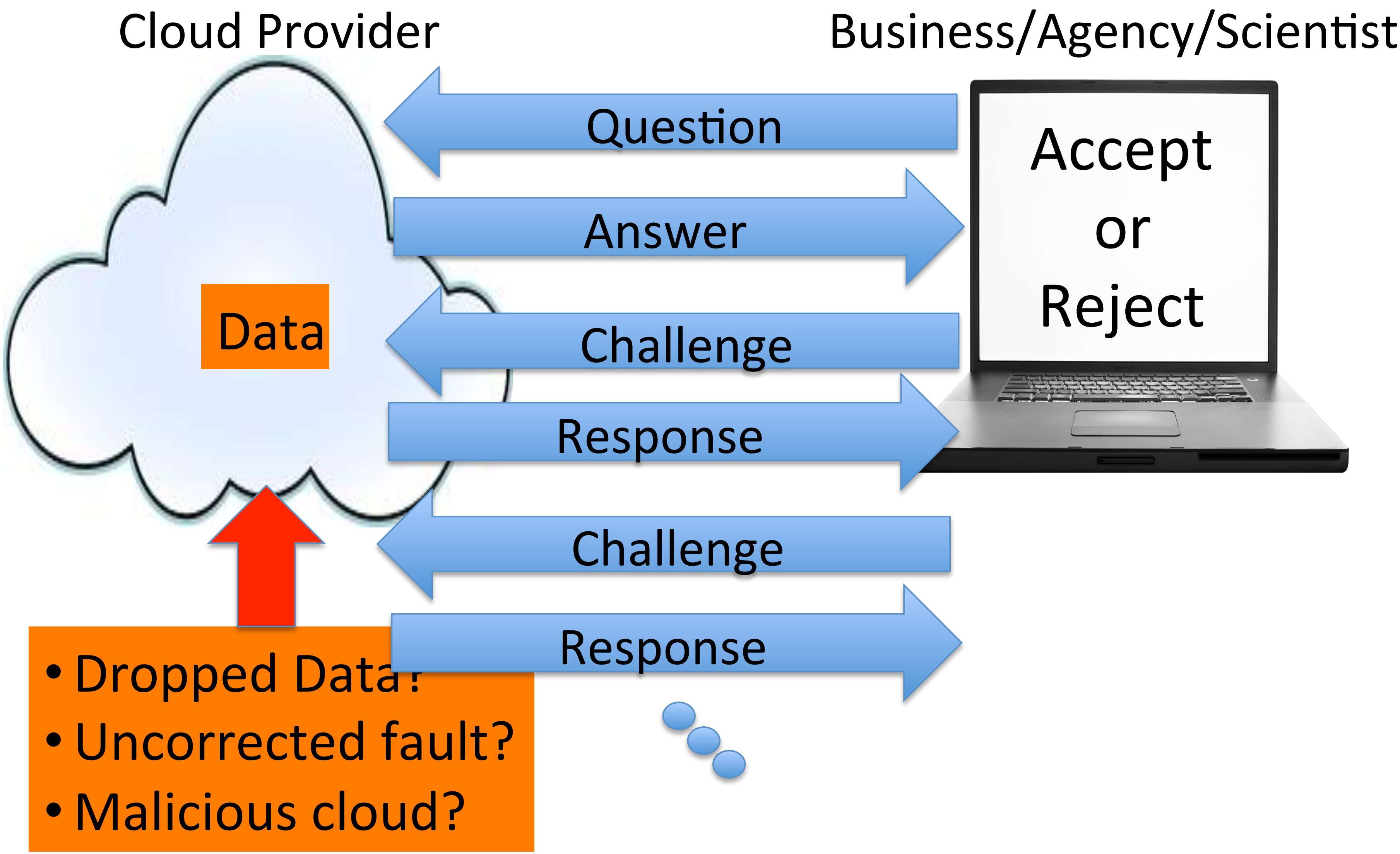
IPs and Arguments



IPs and Arguments



IPs and Arguments



Goals of Verifiable Computation

1. Provide user with guarantee of correctness.
   - Ideally user not do (much) more work than just **read the input**.
   - Ideally cloud will not do much more than just **solve the problem**.

2. Applications:
   - Cloud computing.
   - Weak peripheral devices that lack resources to perform required functionality (e.g., keycard readers).
   - Hardware manufactured in untrusted foundries.

Zero-Knowledge (ZK)

- Some IPs and arguments are also **zero-knowledge**.
  - They reveal **nothing** to V other than the validity of the statement being proven.
- This enables **many** additional applications.
  - E.g., Authentication. I publish a cryptographic hash of my password, and later prove I know a preimage of the hash, without revealing anything about the preimage.

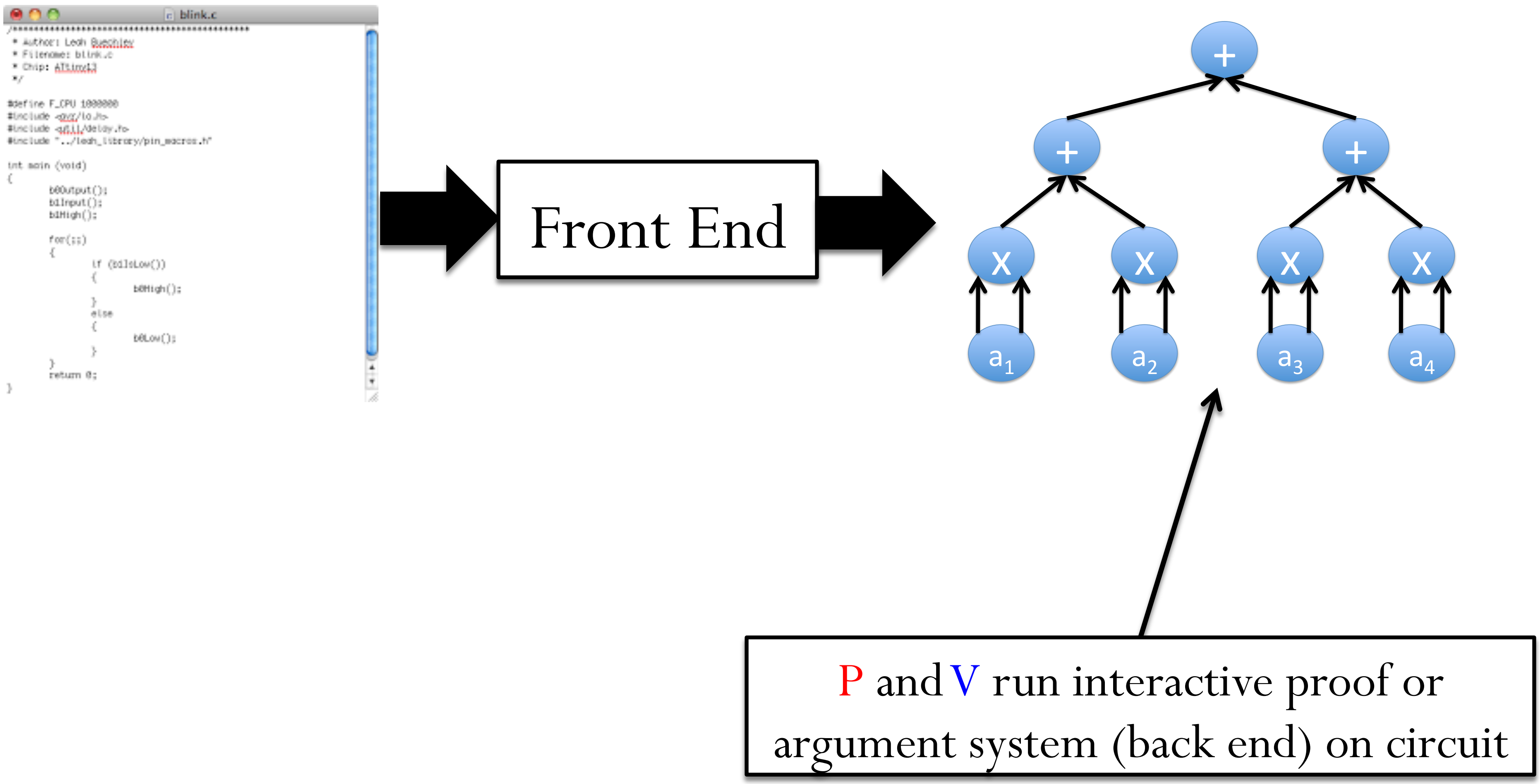
Zero-Knowledge (ZK)

- Some IPs and arguments are also **zero-knowledge**.
  - They reveal **nothing** to V other than the validity of the statement being proven.
- This enables **many** additional applications.
  - E.g., Authentication. I publish a cryptographic hash of my password, and later prove I know a preimage of the hash, without revealing anything about the preimage.
- Enables applications that are otherwise **impossible**.
  - Can justify use of a VC system even if costs are higher than desired.

Part 2: General-Purpose VC Implementations: How They Work

General-Purpose VC Implementations

- Start with a computer program written in high-level programming language (C, Java, etc.)
- Step 1: Turn the program into an equivalent model amenable to probabilistic checking.
  - Typically some type of arithmetic circuit.
  - Called the **Front End** of the system.
- Step 2: Run an interactive proof or argument on the circuit.
  - Called the **Back End** of the system.


blink.c
Front End
+
+
+
x
x
x
x
a_1
a_2
a_3
a_4
P and V run interactive proof or argument system (back end) on circuit

Sources of Prover Overhead in VC Systems

Source of Overhead	P Overhead vs. Native **(Crude Estimate)**	Slowdown Depends On…
Front End (overhead due to using a circuit representation of the computation)	(ratio of circuit size to number of machine steps of original program) **1x-10,000x**	• How amenable is the high-level computer program is to representation via circuits? • What type of circuits can the back-end handle?
Back-End	(ratio of P time to evaluating circuit gate-by-gate) **10x-1,000x**	• Varies by back-end and computation structure (e.g., data parallel?)

Part 3: Survey and Comparison of Existing VC Implementations

Overview of Backends

- Four approaches to general-purpose VC systems have been pursued.
  - Approach 1: Arguments based on **linear PCPs**. [IKO 2007, GGPR 2013, BCIOP13]
    - Interactive variants.
    - Non-interactive variants called **SNARKs**.
  - Approach 2: Based on **IPs** [LFKN 1990, GKR 2008].
  - Approach 3: Arguments based on "**short PCPs**" [BSGHSV04, BSS05, BCGT13, BSCS16]
  - Approach 4: Arguments based on **garbled circuits** or **"MPC in the head"**. [Yao 1982, IKOS 2007, JKO2013]
    - So far, useful only for zero-knowledge applications.

Approach	VC Systems
Arguments based on linear PCPs	[SMBW 2012, SVPBBW 2012, SBVBPW 2013, BSCGTV 2013, PGHR 2013, BSCGGMTV 2014, BSCTV 2014a, BSCTV 2014b, BBFR 2015, CTV 2015, CFHKKNPZ 2015, DLFKP 2016]
IPs	[CMT 2012, TRMP 2012, VSBW 2013, Thaler 2013, WHGSW 2016, WJBSTWW 2017, ZGKPP 2017]
Arguments based on short PCPs	[BSBTCGCHPRSTV 2017]
Arguments based on garbled circuits or MPC-in-the-head	[JKO 2013, GMO 2016]

SNARKs vs. IPs: Advantages and Limitations

Advantages of SNARKs over IPs

1. **Zero-Knowledge.**
   - SNARKs are, IPs are not.
2. **Succinctness (i.e., very short proofs).**
   - Consider the arithmetic CIRCUIT-SAT problem.
   - Given: circuit C taking two inputs, first input x, and (claimed) outputs y.
     - Assume that P knows a w such that $C(x,w)=y$.
     - Goal: confirm this is the case.
   - An argument is **succinct** if the proof length is $o(|w|)$.
   - SNARKs have proof length $|y|+O(1)$ group elements.
   - IPs have proof length $|y|+|w|+O(d \cdot \log S)$ field elements.
     - d is circuit depth and S is circuit size.

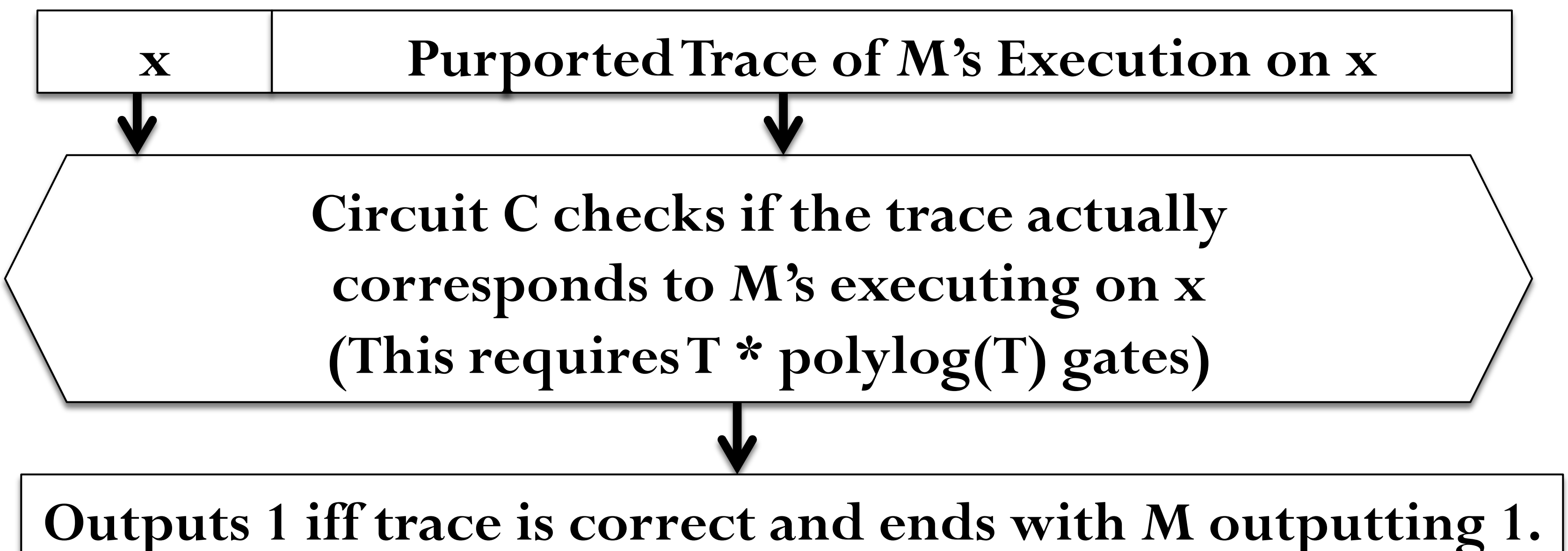
Why is Succinctness Important?

1. Shorter proofs are obviously better.
   - In blockchain applications, proofs must "live on the blockchain" forever.
2. In some zero-knowledge applications, witness is naturally large.
   - E.g., hospital publishes cryptographic hash of a massive database w of patient records, later proves it ran a specific analysis on w.
3. Enables **more efficient front ends**.
   - E.g., can turn any computer program running in time T into a CIRCUIT-SAT instance of size $T \cdot \text{poly}\log(T)$.
   - But the witness size $|w|$ is $T \cdot \text{poly}\log(T)$.
   - So need proof length $o(|w|)$ if we want V to run in time $o(T)$.

Sketch of the Transformation

[GS 1989, Robson 1991, BSCGT 2013]

- A **trace** of program M on input x is the list of the (time, configuration) pairs that arise when running M on x.
  - A configuration specifies the bits in M's program counter and registers.
- C takes x as explicit input, and takes an entire **trace** of M as non-deterministic input.
- C then checks the trace for correctness, and if so outputs whatever M outputs in the trace.



Advantages of IPs over SNARKs

1. IPs have **much faster P**.
   - SNARK prover does expensive crypto operations for each gate in C.
2. IPs have **no public parameters**.
   - In applications, SNARK parameter size is close to 1 GB or more.
3. IPs make no **crypto assumptions**.
   - SNARKs are based on strong (i.e., *non-falsifiable*) crypto assumptions.
4. IPs can **avoid expensive pre-processing** phase for V.
   - For circuits with "regular" wiring patterns.
5. IPs have much better **space costs for P**.
   - SNARK P performs FFTs on vectors of length S.
   - Limits circuits to ~20 million gates on systems with 32 GB of RAM [WSRBW 2015]
   - SNARK space and pre-processing costs can be asymptotically limited via "bootstrapping", but at very high concrete cost [BCCT 2008, BSCTV 2014].

IPs vs. SNARKs: Final Notes

- Other advantages of IPs: amenable to hardware implementations, superior parallelization.

- SNARKs are publically verifiable and non-interactive.
  - IPs can be made to satisfy these properties in the Random Oracle Model using the Fiat-Shamir heuristic.

Short PCPs, Garbled Circuits, and MPC-in-the-head

Short PCPs vs. SNARKs

- Main advantage short PCPs: they avoid an expensive pre-processing phase for V in a general-purpose manner.
- But concrete costs are currently much higher than SNARKs.
- And existing implementations of short PCPs are not zero-knowledge.

Garbled Circuits and MPC-In-The-Head vs. SNARKs

- Garbled circuits and MPC-In-The-Head have proof length $\Omega(S)$ (with large hidden constant), where S is circuit size.
  - So they don't save V time compared to native execution.
  - But are still useful in ZK applications.

- Advantages over SNARKs:
  - Lack of public parameters.
  - Much faster P for some applications.

Part 4: A Brief History of Interactive Proofs

Interactive Proofs, Pre-2008

- 1985: Introduced by [GMR, Babai].
  - IPs were believed to be just slightly more powerful than classical static (i.e., NP) proofs.
  - i.e. let **IP** denote class of problems solvable by an interactive proof with a poly-time verifier. It was believed that **IP** ≈ **NP**.

Interactive Proofs, Pre-2008

- 1985: Introduced by [GMR, Babai].
  - IPs were believed to be just slightly more powerful than classical static (i.e., NP) proofs.
  - i.e. let **IP** denote class of problems solvable by an interactive proof with a poly-time verifier. It was believed that **IP** ≈ **NP**.
- 1990: [LFKN, Shamir] proved that **IP=PSPACE**.
  - i.e., IPs with a poly-time verifier can actually solve **much** more difficult problems than can classical static proofs.

Interactive Proofs, Pre-2008

- 1985: Introduced by [GMR, Babai].
  - IPs were believed to be just slightly more powerful than classical static (i.e., NP) proofs.
  - i.e. let **IP** denote class of problems solvable by an interactive proof with a poly-time verifier. It was believed that **IP** ≈ **NP**.
- 1990: [LFKN, Shamir] proved that **IP=PSPACE**.
  - i.e., IPs with a poly-time verifier can actually solve **much** more difficult problems than can classical static proofs.
  - But IPs were still impractical.
  - Main reason: P's runtime.
    - When applying IPs of [LFKN, Shamir] even to very simple problems, the honest prover would require **superpolynomial** time.

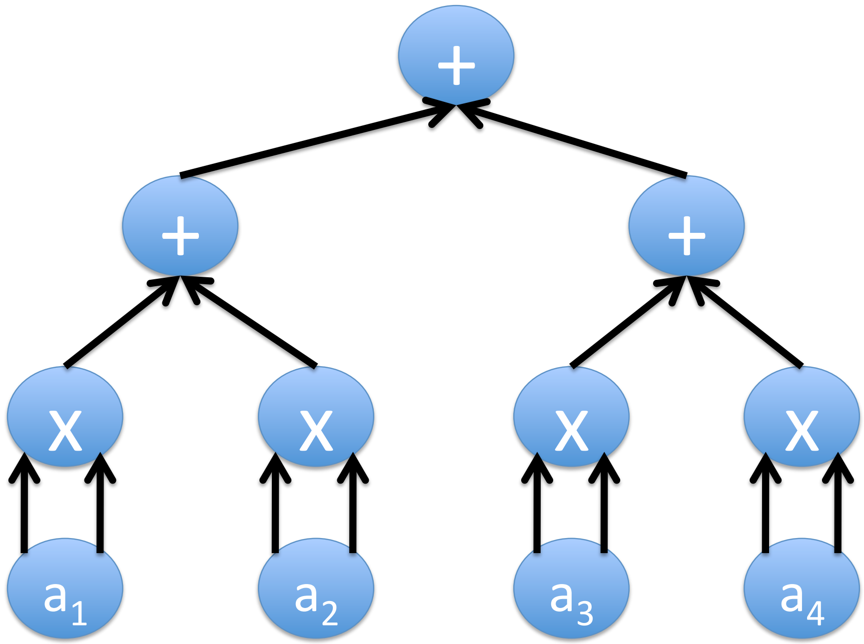
The GKR Protocol

- [GKR 2008] addressed P's runtime.
  - They gave an IP for any function computed by an efficient **parallel** algorithm.
  - P runs in polynomial time.
  - V runs in (almost) linear time, so outsourcing is useful even though problems are "easy".

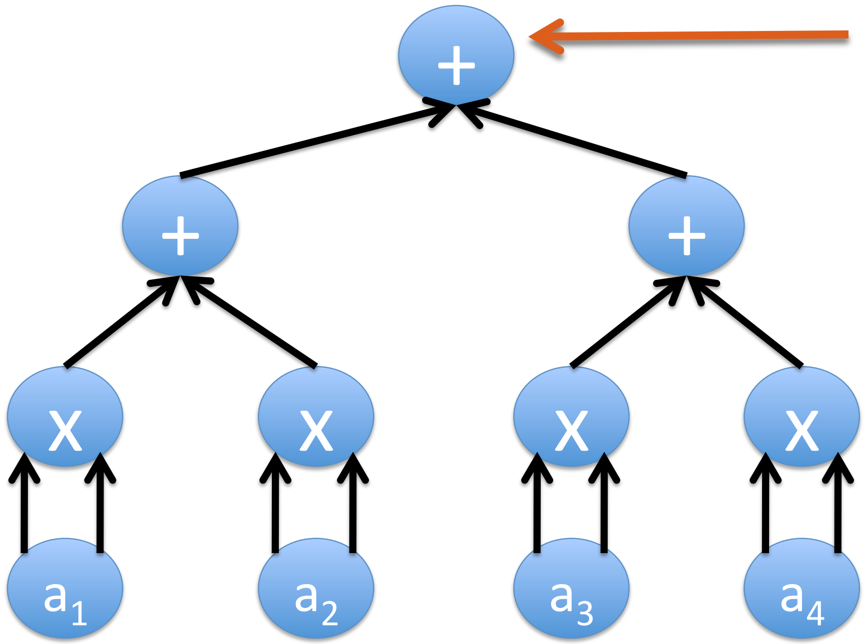
The GKR Protocol

- [GKR 2008] addressed P's runtime.
  - They gave an IP for any function computed by an efficient **parallel** algorithm.
  - P runs in polynomial time.
  - V runs in (almost) linear time, so outsourcing is useful even though problems are "easy".
- But GKR is not practical out of the box.
  - P still requires a lot of time (**quartic** blowup in runtime).

The GKR Protocol: Overview

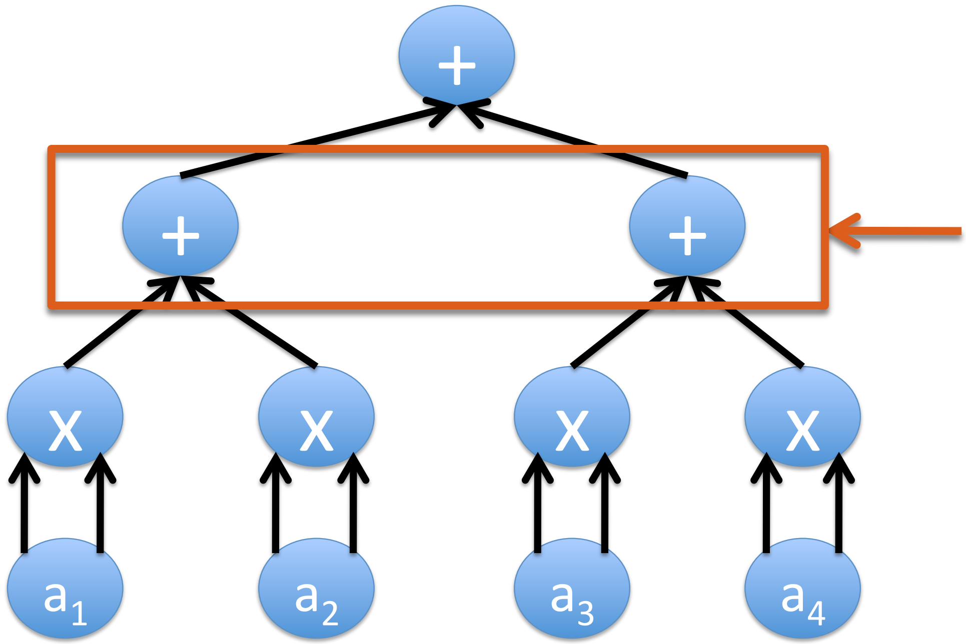


The GKR Protocol: Overview



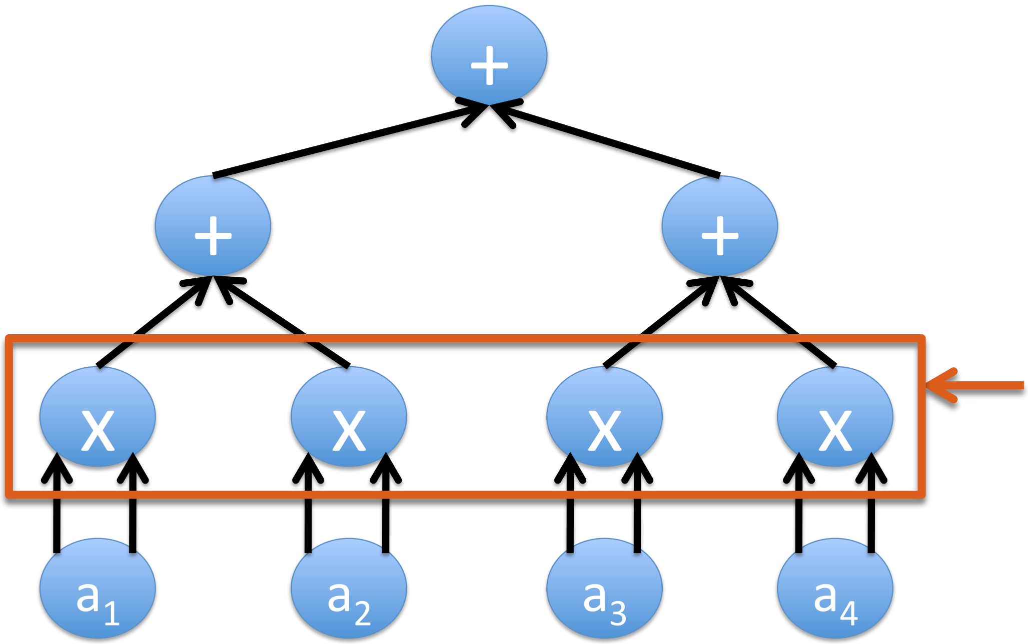
P starts the conversation with an answer (output).

The GKR Protocol: Overview



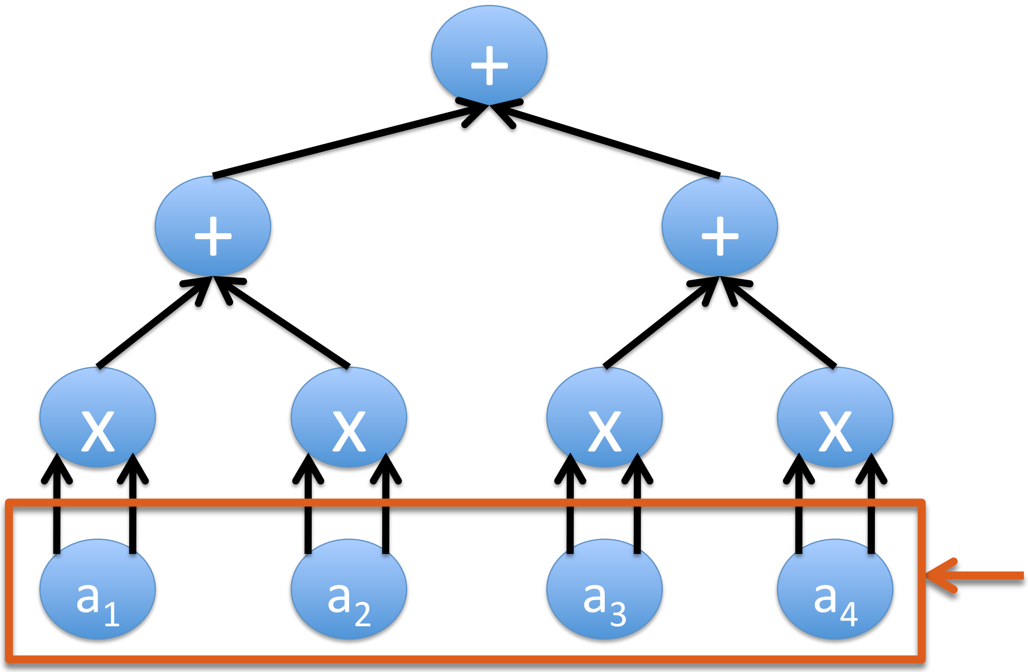
V sends series of challenges. P responds with info about next circuit level.

The GKR Protocol: Overview



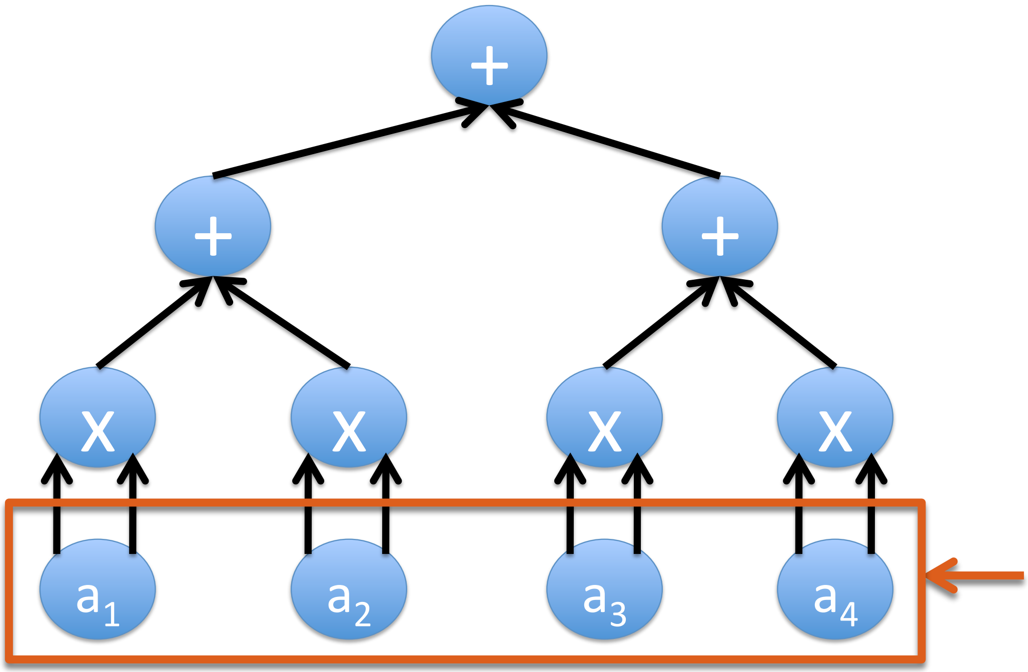
Challenges continue, layer by layer down to the the input.

The GKR Protocol: Overview



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

The GKR Protocol: Overview



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

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

From Theory to Practice

- [CMT 2012] implemented the GKR protocol (with refinements).
- Demonstrated low concrete costs for V.
- Brought P's runtime down from $\Omega(S^4)$, 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.

From Theory to Practice

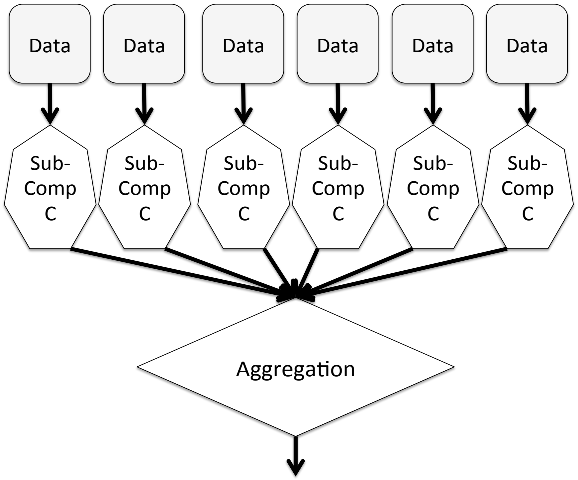
- [CMT 2012] implemented the GKR protocol (with refinements).
- Demonstrated low concrete costs for V.
- Brought P's runtime down from $\Omega(S^4)$, to $O(S \log S)$, where S is circuit size.
  - Key insight: use **multilinear** extension of circuit within the protocol.
  - Causes enormous cancellation in P's messages, allowing fast computation.
- Still not good enough on its own.
  - P is ~10^3 times slower than just evaluating the circuit.
  - Naïve implementation of GKR would take trillions of times longer.

From Theory to Practice

- [CMT 2012] implemented the GKR protocol (with refinements).
- Demonstrated low concrete costs for V.
- Brought P's runtime down from $\Omega(S^4)$, 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.
  - Both P and V can be sped up 40x-100x using GPUs [TRMP12].

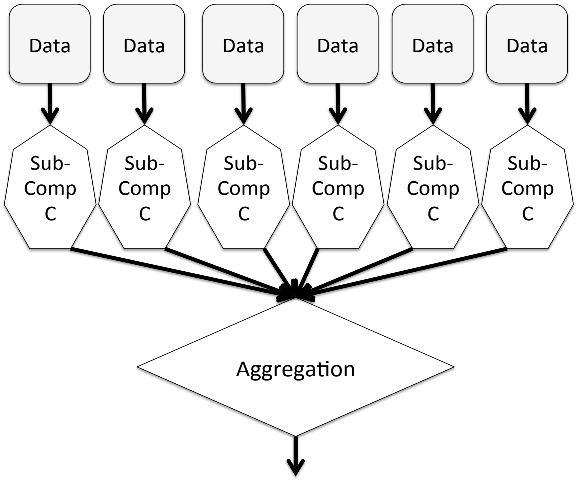
Improvements for "Structured Computation"

- [Thaler 2013] brought P's runtime down further for any circuit that exhibits **repeated structure.**
  - Includes any **data parallel** computation.
  - P runs in time $O(S \log B)$, where B is size of the **sub**-computation.



Improvements for "Structured Computation"

- [Thaler 2013] brought P's runtime down further for any circuit that exhibits **repeated structure.**
  - Includes any **data parallel** computation.
  - P runs in time $O(S \log B)$, where B is size of the **sub**-computation.



- [WJBSTWW 2017] brings this down even further, to $O(S + B \cdot \log B)$.
  - For "sufficient levels of data parallelism", this is $O(S)$.
  - The hidden constant is ≈10.

Verifiable ASICs

- [WHGSW 2016, WJBSTWW 2017] implement these IPs in hardware.
  - Motivation: verifiable ASICS.
  - Produce fast, special-purpose hardware in an (untrusted) country's advanced foundry.
  - Make the hardware act as P.
  - Implement V using **much** slower, domestically-manufactured hardware.

Making IPs Succinct

- [ZGKPP 2017] renders IPs succinct.
  - By combining IPs with a cryptographic primitive called a **polynomial commitment scheme** [KGG 2010, PST 2013]
  - Reduces proof length for CIRCUIT-SAT from $|y|+|w|+O(d \cdot \log S)$ to $|y|+O(\log|w|)+O(d \cdot \log S)$.
  - Applies techniques to database applications.
  - Downsides: introduces strong cryptographic assumptions, utilizes public parameters of size proportional to $|w|$.

Open Questions

- One VC System to rule them all?
  - Endow IPs with **zero-knowledge** and **succinctness** without sacrificing **any** of IPs' advantages over SNARKs?
- Understand the power of IPs in communication complexity.
  - Proving lower bounds on the communication analog of **AM** is a notorious open problem.
  - Even open for the communication analogs of **NISZK** and **SZK.**

Comparison of Techniques: IPs vs. Other Approaches

Overview of Argument Systems

- Most arguments work by:
  1. "Starting" with an information-theoretically secure protocol in a model where P is assumed to behave in a restricted manner.
     - E.g., a linear PCP, "short" PCP, etc.
     - These models assume P is non-adapative (i.e., P's answer to each query from V does not depend on earlier queries).
  2. Then using cryptography to "force" a computationally bounded P to behave in the restricted manner.

SNARKs, Short PCPs

- Whereas GKR checks the circuit layer by layer, all other approaches check the circuit **all at once**.
- They crucially exploit non-adaptivity of P to do this.

- Recall: C is arithmetic circuit (over $\mathbf{F}$) of size S and we want to check that $C(x,w) = y$.

SNARKs, Short PCPs, MIPs, etc.

- Let H be a set of size S. Assign each gate in C a label from H.
- A **transcript** $W : H \rightarrow \mathbf{F}$ is an assignment of **values** to each gate.
- Call W **valid** if it is consistent with C's execution on input (x, w).
- Let $\widetilde{W}$ be **a low-degree extension** of W.
  - i.e., a low degree polynomial such that
    $$\widetilde{W}(a) = W(a) \text{ for all } a \in H.$$
- Somehow define a polynomial $g_{\widetilde{W}}$ derived from $\widetilde{W}$ such that:
  $$g_{\widetilde{W}}(a) = 0 \text{ for all } a \in H \Leftrightarrow W \text{ is a valid transcript.}$$
- The "proof" can be regarded as having two parts:
  - Part 1: $\widetilde{W}$
  - Part 2: some extra info certifying that $g_{\widetilde{W}}(a) = 0$ for all $a \in H$.

Thank you!