# SNARKs from the sum-check protocol

Justin Thaler

Georgetown University

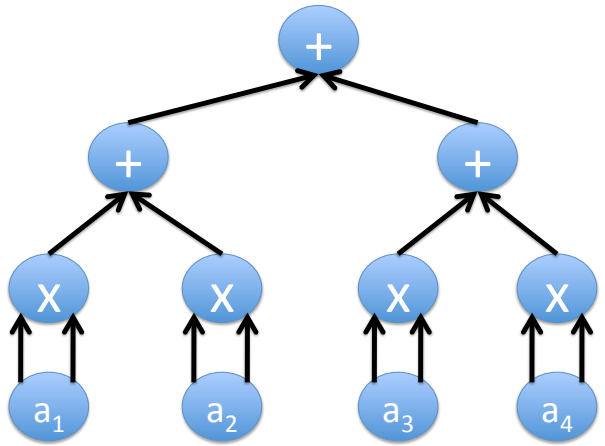
# General-Purpose SNARKs

- Prover P claims to know witness $w$ satisfying some property.

# General-Purpose SNARKs

- Prover P claims to know witness $w$ satisfying some property.
- Start with a computer program written in high-level programming language (C, Java, etc.).
  - Takes $w$ as input and checks it satisfies the claimed property.
- Step 1: Turn the program into an equivalent model amenable to probabilistic checking.
  - Typically some type of circuit-satisfiability or R1CS problem.
  - Called the **Front End** of the system.
- Step 2: Run an interactive proof on the circuit/R1CS.
  - Called the **Back End** of the system.


Front End




P and V run SNARK (back end) for R1CS or circuit-satisfiability.

# Key Paradigm for Backend Design

1. Give a “polynomial IOP” for for R1CS or circuit-satisfiability.
2. Combine with polynomial commitment scheme to get succinct interactive argument.
3. Apply Fiat-Shamir transformation to render non-interactive.

# Key Paradigm for Backend Design

1. Give a “polynomial IOP” for for R1CS or circuit-satisfiability.
2. Combine with polynomial commitment scheme to get succinct interactive argument.
3. Apply Fiat-Shamir transformation to render non-interactive.

- The sum-check protocol is a key technique for Step 1.

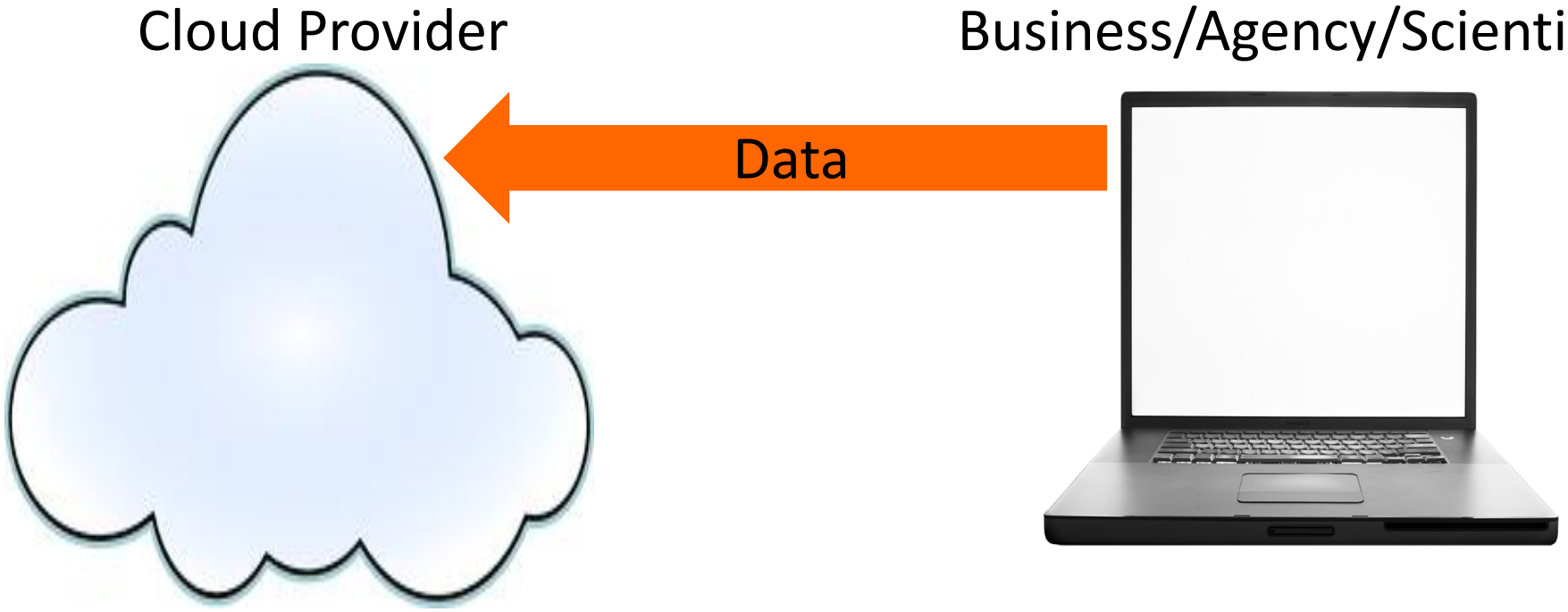
# Building Block: Interactive Proofs (IPs)

# Interactive Proofs (IPs)

Cloud Provider

Business/Agency/Scientist

# Interactive Proofs (IPs)

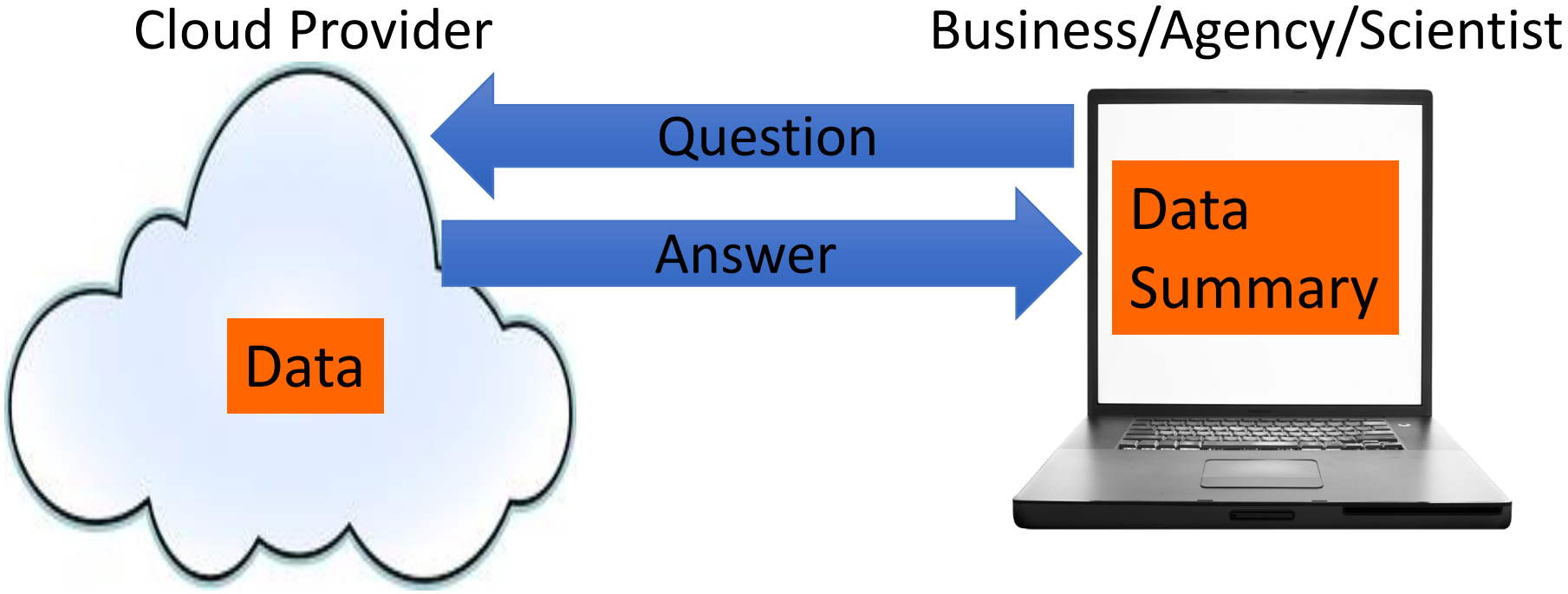


# Interactive Proofs (IPs)

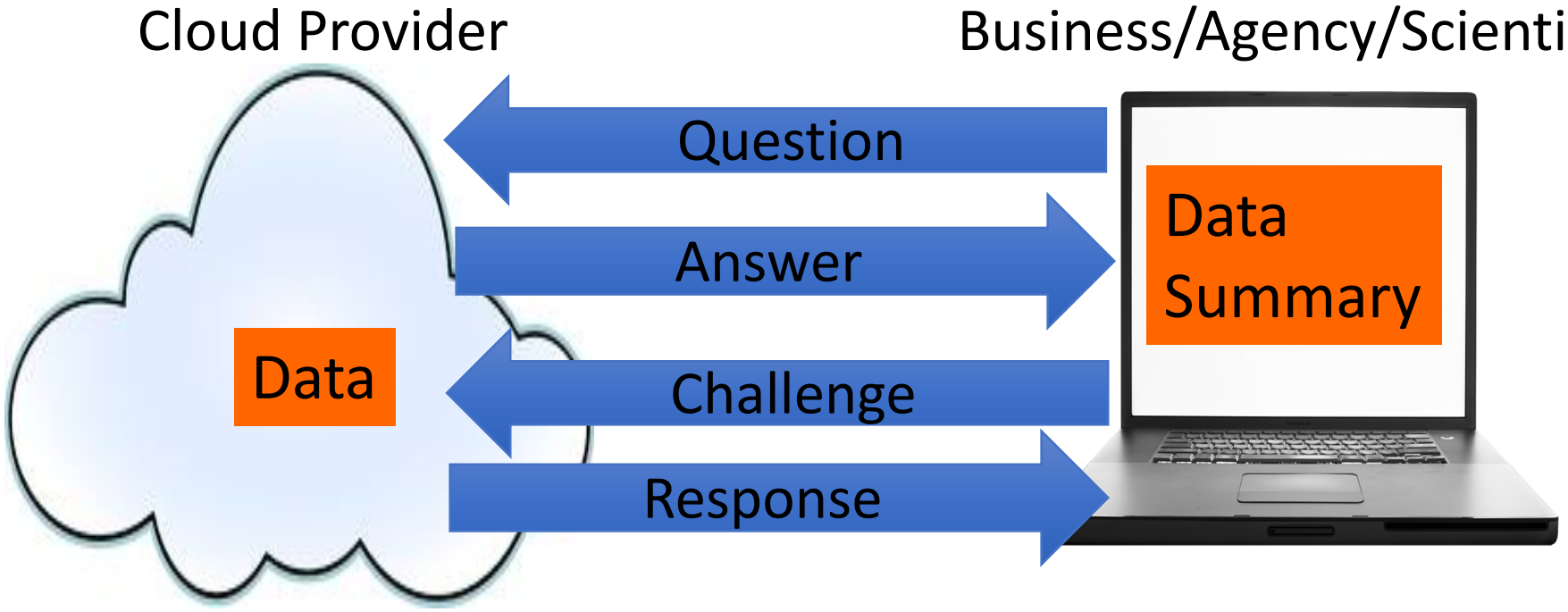
Cloud Provider


Business/Agency/Scientist

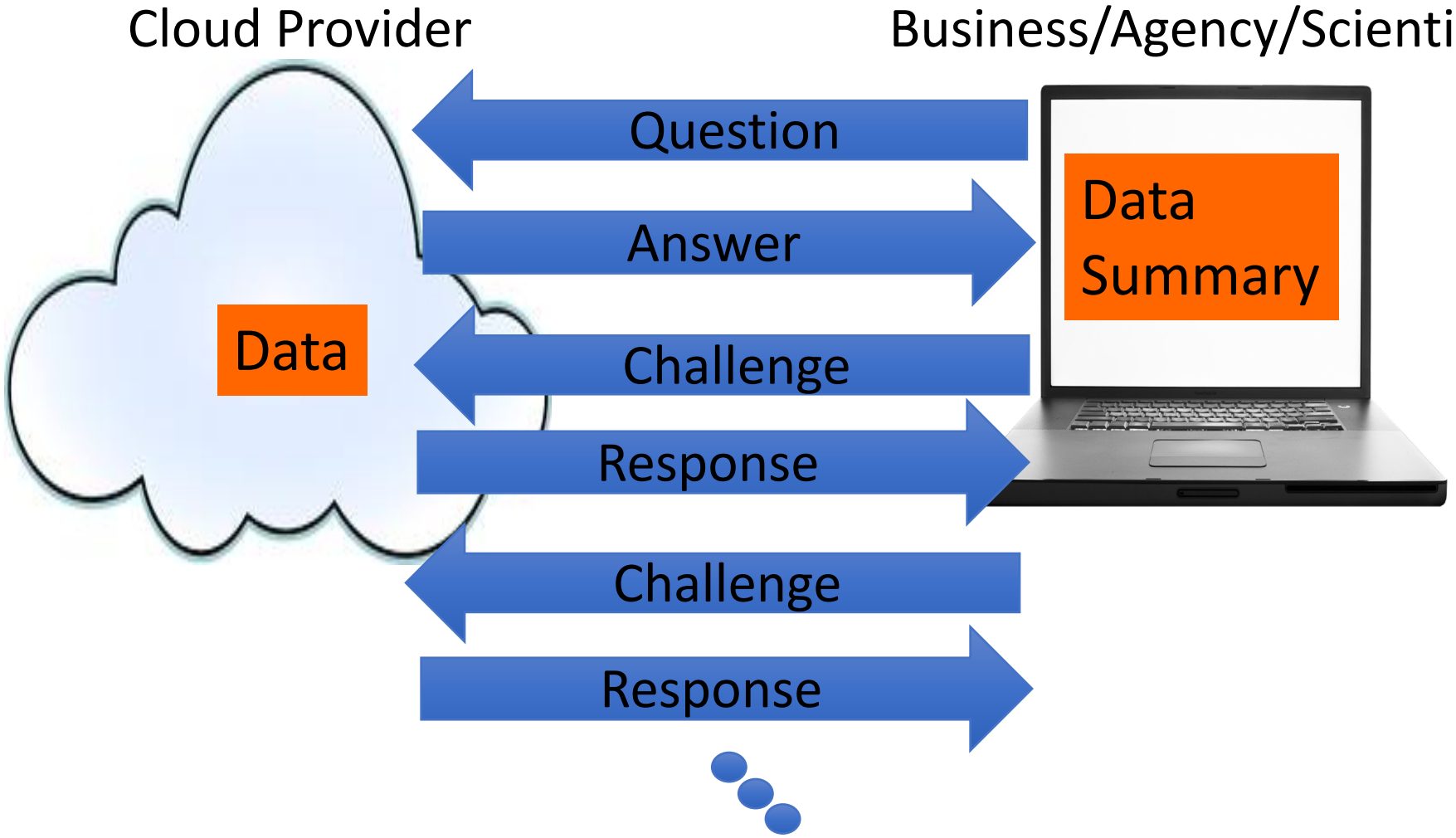
# Interactive Proofs (IPs)



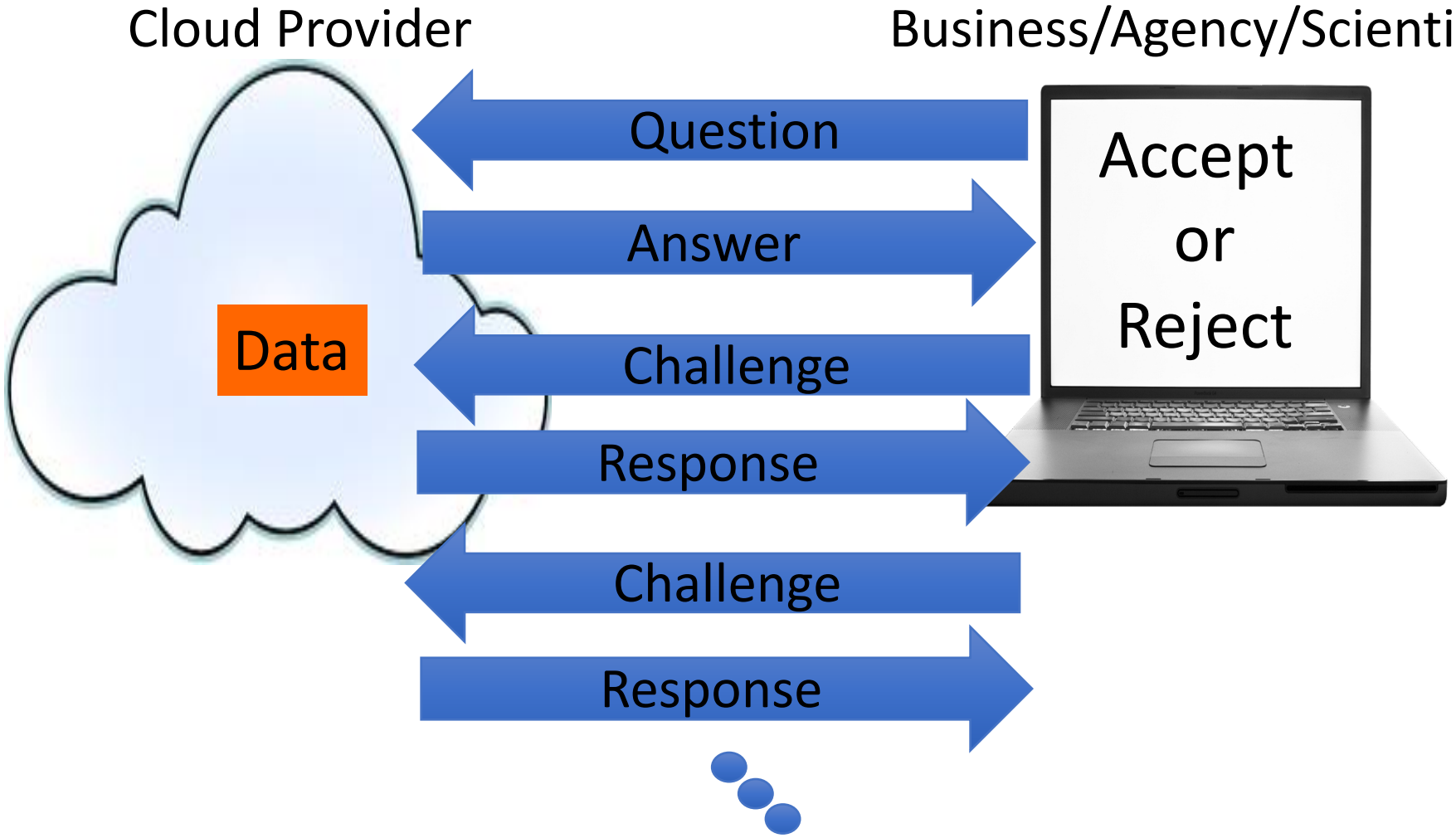
# Interactive Proofs (IPs)



# Interactive Proofs (IPs)



# Interactive Proofs (IPs)



# Doubly-Efficient Interactive Proof

- A doubly-efficient interactive proof for a problem is one where:
  - V runs in time linear in the input size.
  - P runs in polynomial time.

# The Sum-Check Protocol: Technical Ideas

# Checking if two polynomials are equal

- Let $g$ and $h$ be two univariate degree-$d$ polynomials over field $\boldsymbol{F}$.
- Goal: quickly determine if $g$ and $h$ are the **same** polynomial.

# Checking if two polynomials are equal

- Let $g$ and $h$ be two univariate degree-$d$ polynomials over field $\boldsymbol{F}$.
- Goal: quickly determine if $g$ and $h$ are the **same** polynomial.

- **Fact:** If $g$ and $h$ are not the same polynomial, then:

$$\Pr_{r \sim \boldsymbol{F}}[g(r) \neq h(r)] \geq 1 - \frac{d}{|\boldsymbol{F}|}.$$

# The Sum-Check Protocol, Simplified

- Say we want to design an IP computing some function $f$.

# The Sum-Check Protocol, Simplified

- Say we want to design an IP computing some function $f$.
- Suppose that given any input $x$ there is some degree-$\sqrt{n}$ polynomial $g_x$ such that:

$$f(x) = \sum_{i=1}^{\sqrt{n}} g_x(i).$$

# The Sum-Check Protocol, Simplified

- Say we want to design an IP computing some function $f$.
- Suppose that given any input $x$ there is some degree-$\sqrt{n}$ polynomial $g_x$ such that:

$$f(x) = \sum_{i=1}^{\sqrt{n}} g_x(i)\,.$$

- And suppose that V can evaluate $g_x$ at any **single** input $r$ in $O(n)$ time.

# The Sum-Check Protocol, Simplified

- Say we want to design an IP computing some function $f$.
- Suppose that given any input $x$ there is some degree-$\sqrt{n}$ polynomial $g_x$ such that:

$$f(x) = \sum_{i=1}^{\sqrt{n}} g_x(i).$$

- And suppose that V can evaluate $g_x$ at any **single** input $r$ in $O(n)$ time.

- Here's the (non-interactive!) proof:
  - P sends V a polynomial $h$ claimed to equal $g_x$.
  - V picks a random input $r$ and confirms that $h(r) = g_x(r)$, rejecting if not.
  - V outputs $\sum_{i=1}^{\sqrt{n}} h(i)$. This takes $O(n)$ time.

# The Sum-Check Protocol, Simplified

- Say we want to design an IP computing some function $f$.
- Suppose that given any input $x$ there is some degree-$\sqrt{n}$ polynomial $g_x$ such that:

$$f(x) = \sum_{i=1}^{\sqrt{n}} g_x(i)\,.$$

- And suppose that V can evaluate $g_x$ at any **single** input $r$ in $O(n)$ time.

- Here's the (non-interactive!) proof:
  - P sends V a polynomial $h$ claimed to equal $g_x$.
  - V picks a random input $r$ and confirms that $h(r) = g_x(r)$, rejecting if not.
  - V outputs $\sum_{i=1}^{\sqrt{n}} h(i)$. This takes $O(n)$ time.
- [AW 2008]: MA communication complexity of Disjointness.
- [T16]: doubly-efficient MA protocol for counting triangles.

# Pros and Cons

- Pro: the protocol on the previous slide is non-interactive.
- Con: the proof gave a **complete** description of $g_x$ (cost $O(\sqrt{n})$).

# Pros and Cons

- Pro: the protocol on the previous slide is non-interactive.
- Con: the proof gave a **complete** description of $g_x$ (cost $O(\sqrt{n})$).
- If $g_x$ is multivariate, then there is an **interactive** protocol with proof length proportional to the **total degree** of $g_x$.
  - This can be logarithmic in the size of a complete description of $g_x$.

# Pros and Cons

- Pro: the protocol on the previous slide is non-interactive.
- Con: the proof gave a **complete** description of $g_x$ (cost $O(\sqrt{n})$).
- If $g_x$ is multivariate, then there is an **interactive** protocol with proof length proportional to the **total degree** of $g_x$.
  - This can be logarithmic in the size of a complete description of $g_x$.
    - E.g., $p(x_1, x_2, x_3, x_4, x_5) = 1 + x_1 + x_2 + 3x_3 + 7x_4 + 2x_1x_2 + 5x_3x_4 - x_3x_4x_5 + x_2x_3x_4 + 3x_1x_3x_4$.
    - Total degree of $p$ is only 3, but description is longer.

# Sum-Check Protocol [LFKN90]

- Goal: compute the quantity:

$$\sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_\ell \in \{0,1\}} g(b_1, \dots, b_\ell).$$

- Proof length is roughly the total degree of $g$.
- Number of rounds is $\ell$.
- V time is roughly the time to evaluate $g$ at a single randomly chosen input.

# Sum-Check Protocol [LFKN90]

- Goal: compute the quantity:

$$\sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).$$

- Proof length is roughly the total degree of $g$.
- Number of rounds is $\ell$.
- V time is roughly the time to evaluate $g$ at a single randomly chosen input.
- [CTY12, CMT12, T13, XZZPS19]: if $g$ is "structured" then P can compute all of its messages during the sum-check protocol quickly.
  - Almost as fast as just solving the problem in the first place.

# A Polynomial-IP for Circuit-SAT [VTBW14]

# A Polynomial-IP for Circuit-SAT [VTBW14]

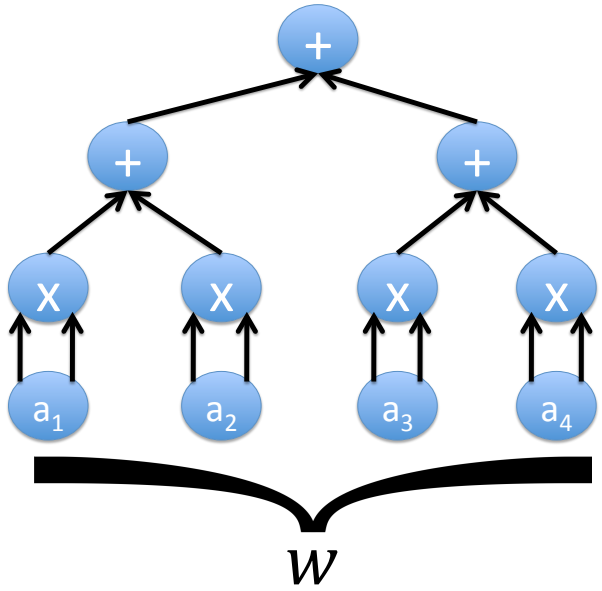
- P's first message in the protocol is a **polynomial** $h$**.**
  - V does **not** learn $h$ in full.
  - Rather, V is permitted to evaluate $h$ at only a few points.
- After that, P and V execute a standard IP.

# A Polynomial-IP for Circuit-SAT [VTBW14]

- P's first message in the protocol is a **polynomial** $h$**.**
  - V does **not** learn $h$ in full.
  - Rather, V is permitted to evaluate $h$ at only a few points.
- After that, P and V execute a standard IP.
- Turn the polynomial-IP into a SNARK by having P commit to $h$ via a polynomial commitment, then use Fiat-Shamir to remove interaction.

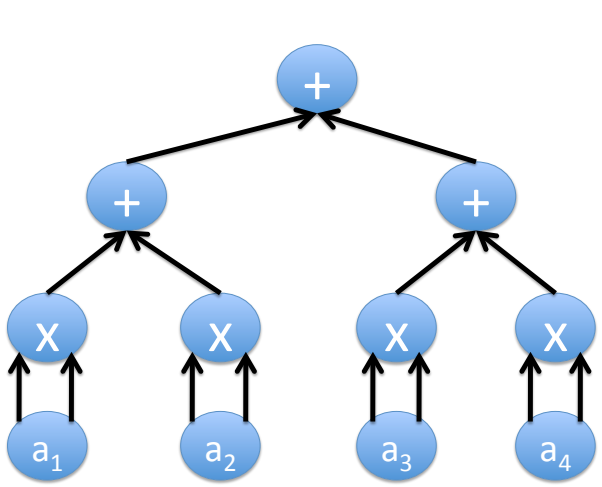
# Arithmetic Circuit Satisfiability

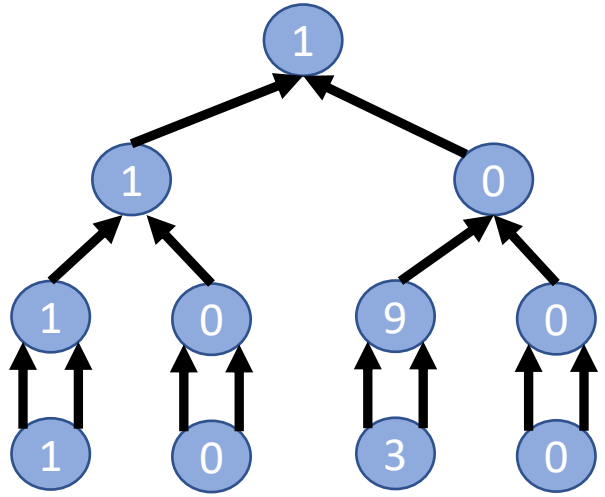
- Given: An arithmetic circuit $C$ over $\boldsymbol{F}$ of size $S$.
- Goal: Determine if there exists a $w$ such that $C(w) = 1$.



# Arithmetic Circuit Satisfiability

- A **transcript** $T$ for $C$ is an assignment of a value to every gate.
  - $T$ is a **correct** transcript if assigns the gate values obtained by evaluating $C$ on a valid witness $w$.




Circuit-SAT instance $C$

Correct transcript for $C$

# The Start of the Polynomial-IP

- Assign each gate in $C$ a $(\log S)$-bit label and view $T$ as a function mapping gate labels to $\boldsymbol{F}$.
- P's first message is a $(\log S)$-variate polynomial $h$ claimed to **extend** a correct transcript $T$, which means:

$$h(x) = T(x) \; \forall \; x \in \{0,1\}^{\log S}.$$

# The Start of the Polynomial-IP

- Assign each gate in $C$ a $(\log S)$-bit label and view $T$ as a function mapping gate labels to $\boldsymbol{F}$.
- P's first message is a $(\log S)$-variate polynomial $h$ claimed to **extend** a correct transcript $T$, which means:

$$h(x) = T(x) \; \forall \; x \in \{0,1\}^{\log S}.$$

- V needs to check this, but is only able to learn a few evaluations of $h$.

**Key Lemma:** Given any $(\log S)$-variate polynomial $h$, there is a related $(3\log S)$-variate polynomial $g_h$ such that:

$h$ **extends** a correct transcript $T \Leftrightarrow g_h(a,b,c) = 0\ \forall (a,b,c) \in \{0,1\}^{3\log S}$.

Moreover, to evaluate $g_h(r)$ at any input $r$, suffices to evaluate $h$ at only 3 inputs.

**Key Lemma:** Given any $(\log S)$-variate polynomial $h$, there is a related $(3\log S)$-variate polynomial $g_h$ such that:

$$h \textbf{ extends} \text{ a correct transcript } T \Leftrightarrow g_h(a,b,c) = 0 \ \forall (a,b,c) \in \{0,1\}^{3\log S}.$$

Moreover, to evaluate $g_h(r)$ at any input $r$, suffices to evaluate $h$ at only 3 inputs.

- Proof sketch (simplification):

$$g_h(a,b,c) := \widetilde{add}(a,b,c) \cdot \Big(h(a) - \big(h(b) + h(c)\big)\Big) + \widetilde{mult}(a,b,c) \cdot \big(h(a) - h(b) \cdot h(c)\big).$$

  - Ensures that:
    1. $g_h(a,b,c) = h(a) - \big(h(b) + h(c)\big)$ if $a$ is the label of a gate that computes the **sum** of gates $b$ and $c$.
    2. $g_h(a,b,c) = h(a) - h(b) \cdot h(c)$ if $a$ is the label of a gate that computes the **product** of gates $b$ and $c$.
    3. $g_h(a,b,c) = 0$ otherwise.

# The Polynomial IP for Circuit-SAT

- Recall: P*'s* first message is a polynomial $h$ claimed to **extend** a correct transcript T.
- By key lemma: $h$ **extends** a correct transcript $\Leftrightarrow g_h(a,b,c) = 0\ \forall (a,b,c) \in \{0,1\}^{3\log S}$.

# The Polynomial IP for Circuit-SAT

- Recall: P's first message is a polynomial $h$ claimed to **extend** a correct transcript T.
- By key lemma: $h$ **extends** a correct transcript $\Leftrightarrow g_h(a,b,c) = 0\ \forall (a,b,c) \in \{0,1\}^{3\log S}$.
- V checks this by running sum-check protocol with P to compute:

$$\sum_{a,b,c\in\{0,1\}^{\log S}} g_h(a,b,c)^2.$$

  - If all terms in the sum are 0, the sum is 0.
  - If working over the integers, any non-zero term in the sum will cause the sum to be strictly positive.

# The Polynomial IP for Circuit-SAT

- Recall: P's first message is a polynomial $h$ claimed to **extend** a correct transcript T.
- By key lemma: $h$ **extends** a correct transcript $\Leftrightarrow g_h(a,b,c) = 0\ \forall (a,b,c) \in \{0,1\}^{3\log S}$.
- V checks this by running sum-check protocol with P to compute:

$$\sum_{a,b,c\in\{0,1\}^{\log S}} g_h(a,b,c)^2.$$

  - If all terms in the sum are 0, the sum is 0.
  - If working over the integers, any non-zero term in the sum will cause the sum to be strictly positive.

- At end of sum-check protocol, V needs to evaluate $g_h(r_1, r_2, r_3)$.
  - To do so, suffices to evaluate $h(r_1), h(r_2), h(r_3)$.

# The Polynomial IP for Circuit-SAT

- Recall: P's first message is a polynomial $h$ claimed to **extend** a correct transcript T.
- By key lemma: $h$ **extends** a correct transcript $\Leftrightarrow g_h(a,b,c) = 0\ \forall (a,b,c) \in \{0,1\}^{3\log S}$.
- V checks this by running sum-check protocol with P to compute:

$$\sum_{a,b,c\in\{0,1\}^{\log S}} g_h(a,b,c)^2.$$

  - If all terms in the sum are 0, the sum is 0.
  - If working over the integers, any non-zero term in the sum will cause the sum to be strictly positive.
- At end of sum-check protocol, V needs to evaluate $g_h(r_1,r_2,r_3)$.
  - To do so, suffices to evaluate $h(r_1), h(r_2), h(r_3)$.
  - Outside of these evaluations, V runs in time $O(\log S)$.
  - P performs $O(S)$ field operations given a witness $w$.

# Trustless Setup SNARKs in One Slide

# Trustless Setup SNARKs

1. Spartan [Setty 20] extends the polynomial IP from circuit-SAT to R1CS, among other improvements.
   - See Srinath's talk later today!
   - Other recent papers (Quarks [SL21], Cerberus [LSTW21]) plug in different polynomial commitment schemes to get SNARKs with different performance profiles.

# Trustless Setup SNARKs

1. Spartan [Setty 20] extends the polynomial IP from circuit-SAT to R1CS, among other improvements.
   - See Srinath's talk later today!
   - Other recent papers (Quarks [SL21], Cerberus [LSTW21]) plug in different polynomial commitment schemes to get SNARKs with different performance profiles.
2. vSQL/Hyrax/Libra/Virgo also use multivariate sum-check techniques.
   - Based on [GKR08] IP for circuit-evaluation.
   - But SNARK proofs are only short for small-depth circuits.

# Trustless Setup SNARKs

1. Spartan [Setty 20] extends the polynomial IP from circuit-SAT to R1CS, among other improvements.
   - See Srinath's talk later today!
   - Other recent papers (Quarks [SL21], Cerberus [LSTW21]) plug in different polynomial commitment schemes to get SNARKs with different performance profiles.
2. vSQL/Hyrax/Libra/Virgo also use multivariate sum-check techniques.
   - Based on [GKR08] IP for circuit-evaluation.
   - But SNARK proofs are only short for small-depth circuits.
3. Marlin/Aurora/Fractal: use univariate rather than multivariate polynomials.

# Trustless Setup SNARKs

1. Spartan [Setty 20] extends the polynomial IP from circuit-SAT to R1CS, among other improvements.
   - See Srinath's talk later today!
   - Other recent papers (Quarks [SL21], Cerberus [LSTW21]) plug in different polynomial commitment schemes to get SNARKs with different performance profiles.
2. vSQL/Hyrax/Libra/Virgo also use multivariate sum-check techniques.
   - Based on [GKR08] IP for circuit-evaluation.
   - But SNARK proofs are only short for small-depth circuits.
3. Marlin/Aurora/Fractal: use univariate rather than multivariate polynomials.
   - They give a "univariate sum-check protocol".

# Trustless Setup SNARKs

1. Spartan [Setty 20] extends the polynomial IP from circuit-SAT to R1CS, among other improvements.
   - See Srinath's talk later today!
   - Other recent papers (Quarks [SL21], Cerberus [LSTW21]) plug in different polynomial commitment schemes to get SNARKs with different performance profiles.
2. vSQL/Hyrax/Libra/Virgo also use multivariate sum-check techniques.
   - Based on [GKR08] IP for circuit-evaluation.
   - But SNARK proofs are only short for small-depth circuits.
3. Marlin/Aurora/Fractal: use univariate rather than multivariate polynomials.
   - They give a "univariate sum-check protocol".
   - Pro: univariate sum-check is only one round, rather than logarithmically many.
     - So if compiled to a SNARK using a polynomial commitment scheme with constant proof size, the resulting proofs are constant-sized (see Marlin, Plonk, etc.)
     - But such polynomial commitments (e.g., KZG) require a trusted setup, aren't post-quantum.
   - Con: P costs tend to be higher.

Thank you!

# Univariate Sum-Check

- Univariate sum-check:
  - Let $\mathrm{H}$ be a multiplicative subgroup of $\mathbf{F}$, $g$ be a low-degree univariate polynomial, and $\boldsymbol{Z}_H$ is the "vanishing polynomial" of $\mathrm{H}$.
  - Lemma: $\sum_{a \in H} g(a) = 0$ if and only if there exists polynomials $p$ and $q$ of degree $d - n$ and $n - 1$ such that:

$$g(X) = p(X) \cdot \boldsymbol{Z}_H(X) - X \cdot q(X).$$

  - To prove that $\sum_{a \in H} g(a) = 0$, P can send two low-degree polynomials $p$ and $q$ such that $g(X) = p(X) \cdot \boldsymbol{Z}_H(X) - X \cdot q(X)$.
    - V can pick a random $r$ and check that $g(r) = p(r) \cdot \boldsymbol{Z}_H(r) - r \cdot q(r)$.