Verifying Computations in the Cloud (and Elsewhere)

Michael Mitzenmacher, Harvard University

Work offloaded to

Justin Thaler, Harvard University

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.
  - Checking correctness should be much faster that performing the computation.

- Minimize extra effort required for cloud to provide correctness guarantee.

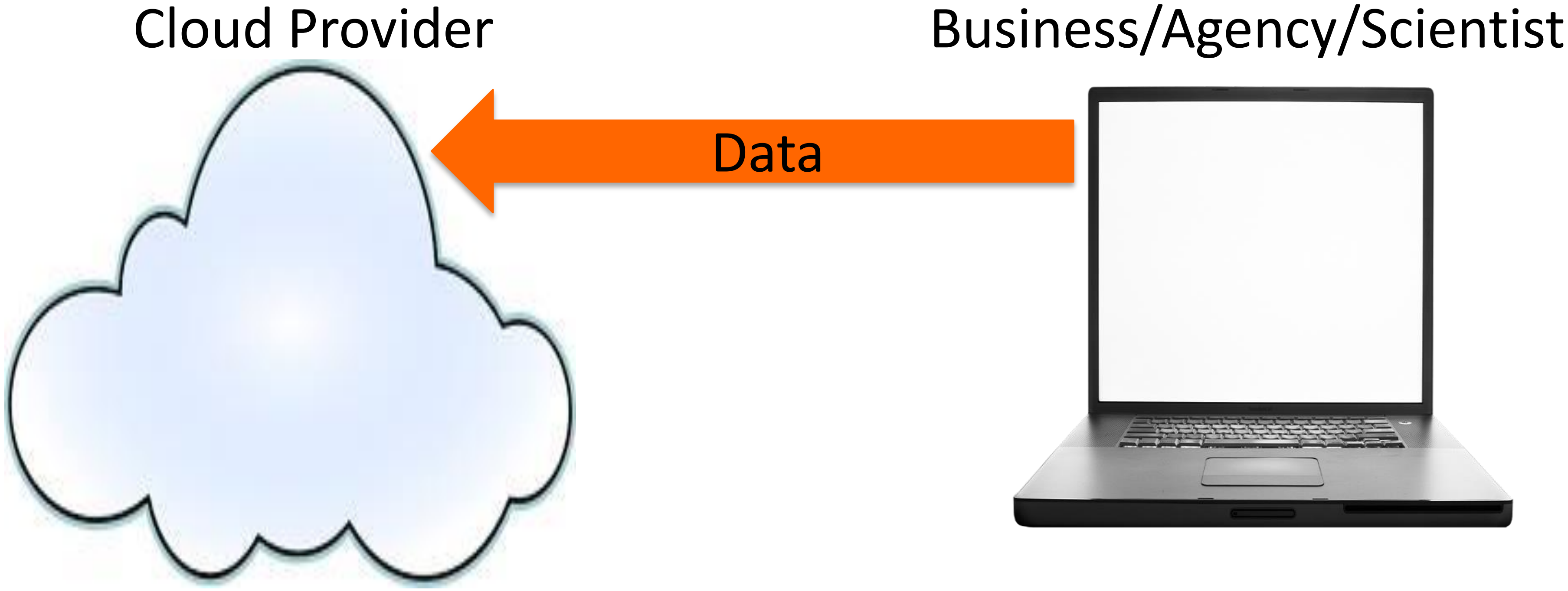
- Achieve protocols secure against malicious clouds, but lightweight for use in benign settings.

Interactive Proofs

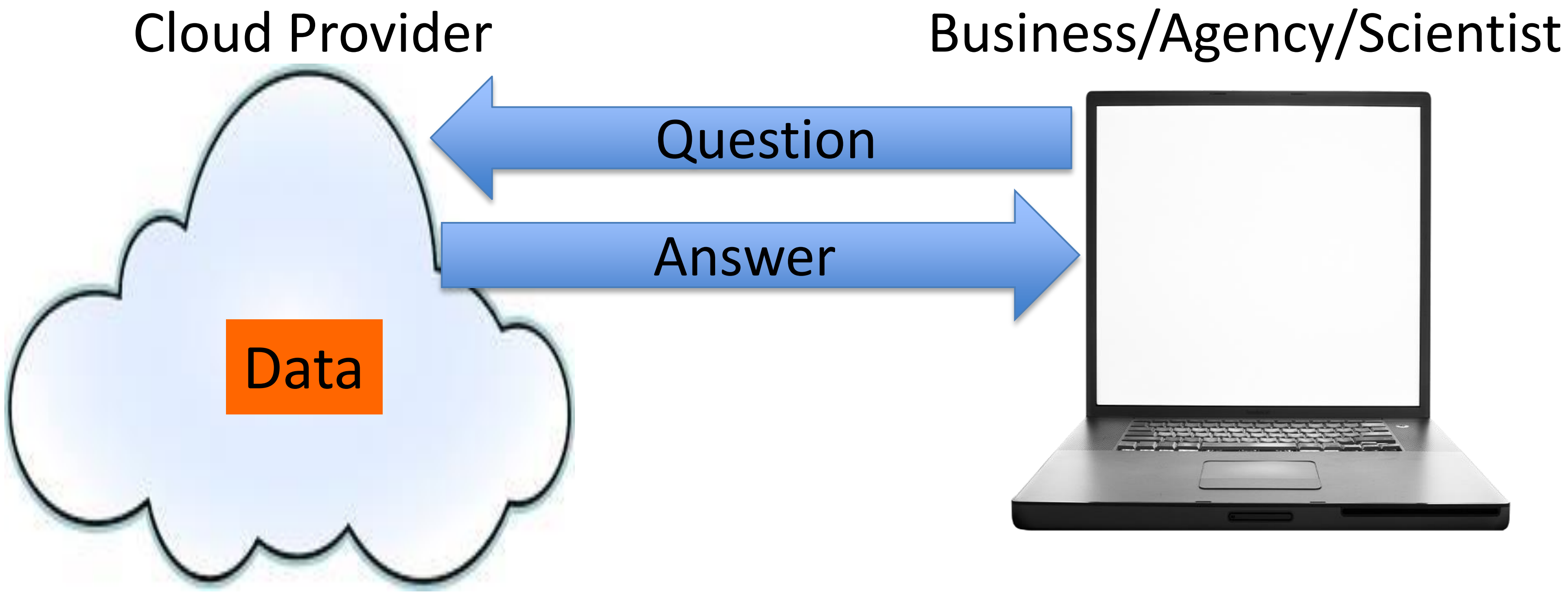
Cloud Provider

Business/Agency/Scientist

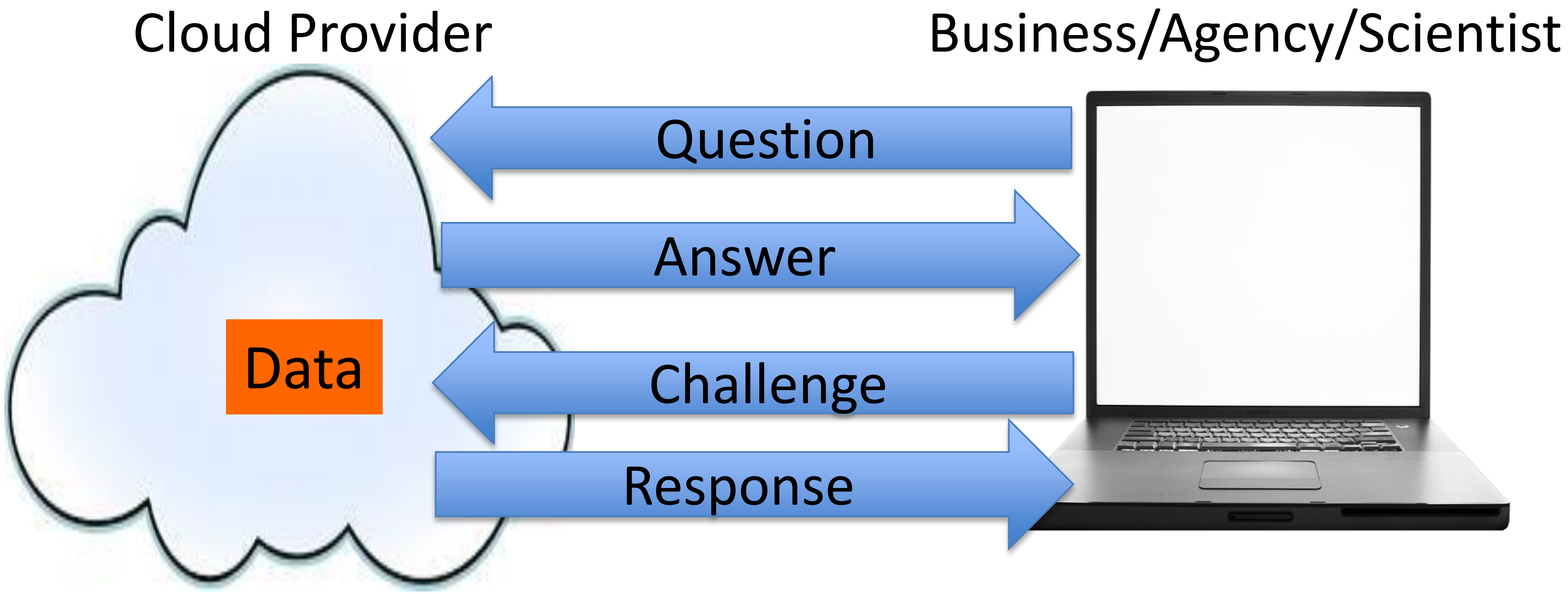
Interactive Proofs



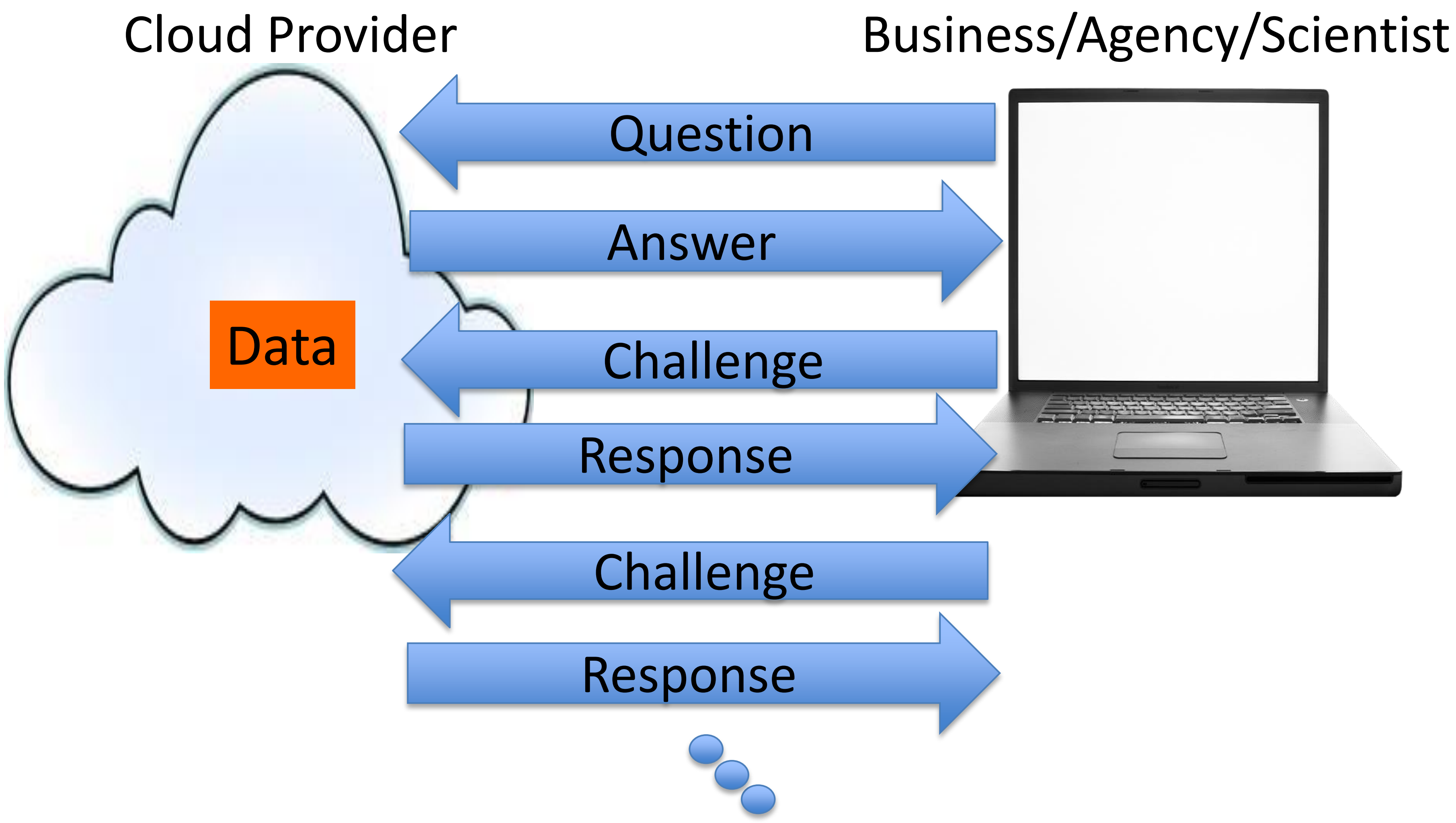
Interactive Proofs



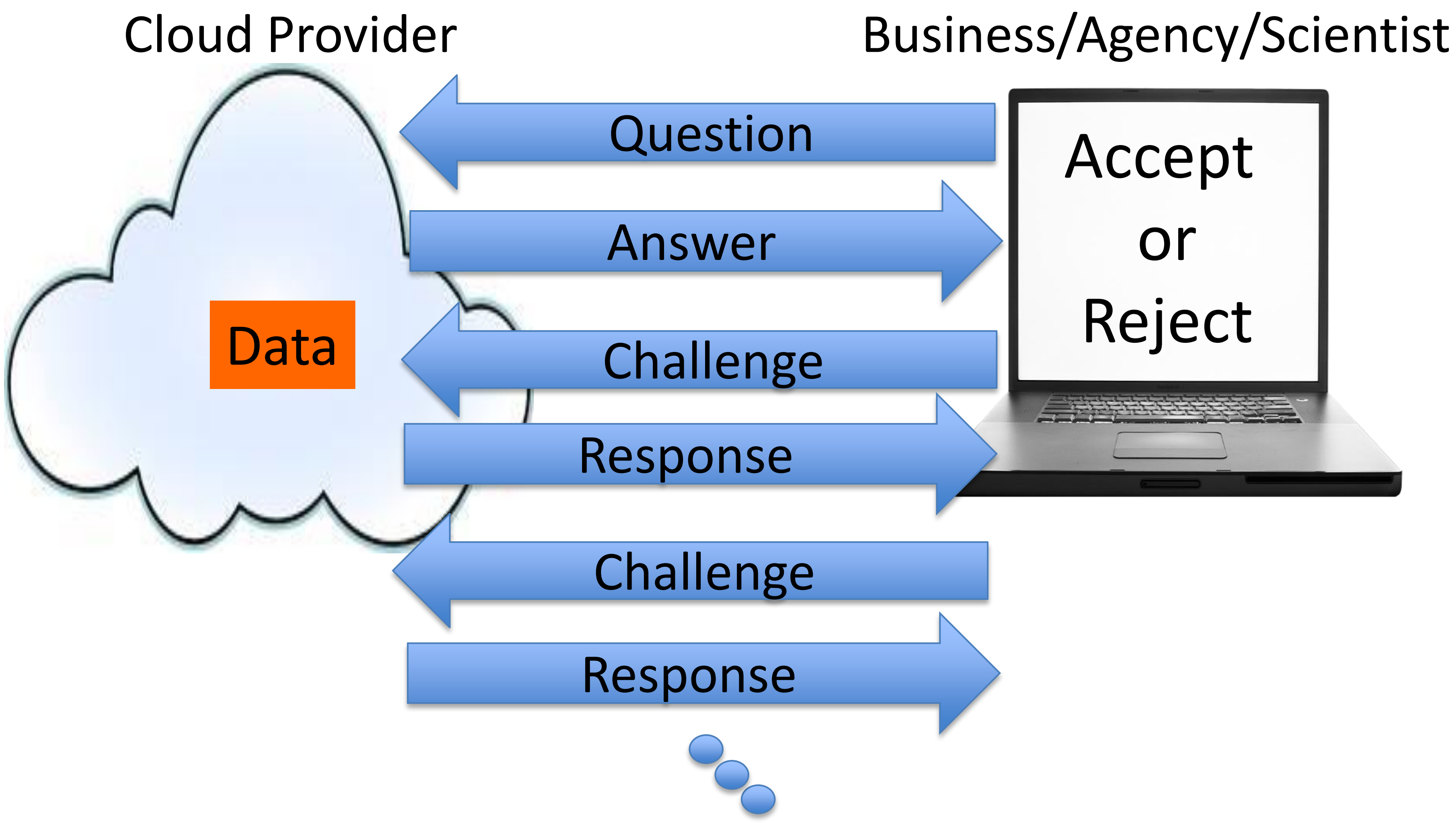
Interactive Proofs



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).

Source: http://harrypotterfans.blogg.se/2009/december/albus-dumbledore.html

Interactive Proofs

- IPs have revolutionized complexity theory in the last 25 years.
  - IP=PSPACE [LFKN90, Shamir90].
  - PCP Theorem e.g. [AS98, ALMSS98]. Hardness of approximation.
  - Zero Knowledge Proofs.

- But IPs have had very little impact in real delegation scenarios.
  - Why?
  - Not due to lack of applications!

Interactive Proofs

- Old Answer: Most results on IPs dealt with hard problems, needed P to be too powerful.
  - But recent constructions focus on "easy" problems (e.g. Interactive Proofs for Muggles [GKR 08]).
  - Allows V to run **very** quickly, so outsourcing is useful even though problems are "easy".
  - P does not need "much" more time to prove correctness than she does to just solve the problem!

Interactive Proofs

- 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).
  - Naively, V has to retain the full input.

Streaming : New Application of IPs

- Streaming setting: data passes through V but not stored at V; V reads input and can do small amounts of computation as it passes by.

- Streaming problems: hard because V has to read input in one-pass streaming manner, but (might be) easy if V could store the whole input.

- Fits cloud computing well: streaming pass by V can occur while uploading data to cloud.

- V never needs to store entirety of data!

Data Streaming Model

- Stream: m elements from universe of size n.
  - e.g., $S = \langle x_1, x_2, \ldots, x_m \rangle = 3,5,3,7,5,4,8,7,5,4,8,6,3,2, \ldots$

- Goal: Compute a function of stream, e.g., median, frequency moments, heavy hitters.

- Challenge:

  (i) Limited working memory, i.e., sublinear(n,m).

  (ii) Sequential access to adversarially ordered data.

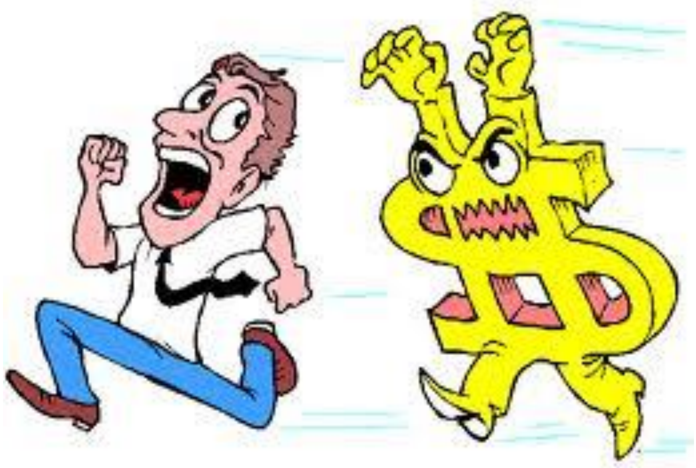
Slide derived from [McGregor 10]

One round vs. Many rounds

- Two models:
  1. One message (Non-interactive) [CCM 09/CCMT 12]: After both observe stream, P sends V an email with the answer, and a proof attached. Less interaction; more data sent.
  2. Multiple rounds of interaction [CTY 10]: P and V have a *conversation* after both observe stream.

Costs in Our Models

- Two main costs: words of communication, and V's working memory.
- Other costs: running time, number of messages.



A Two-Pronged Approach

- First Prong: General purpose implementation to verify arbitrary computation [CMT12, TRMP12, T13].
  - Building on general-purpose GKR protocol.

- Second Prong: Develop highly optimized protocols for specific important problems [CCMT12, CMT10, CTY12, CCGT13].
  - Reporting queries (what value is stored in memory location x of my database?)
  - Matrix multiplication.
  - Graph problems like perfect matching.
  - Certain kinds of linear programs.
  - Etc.

Non-Interactive Protocols with Streaming Verifiers: A Sampling

A general technique

- Arithmetization: Given function *f* defined on small domain, replace *f* with its low-degree extension, LDE(*f*), as a polynomial defined over a large field.

- Can view LDE(*f*) as error-corrected encoding of *f*. Error-correcting properties give V considerable power over P.

- If two (boolean) functions differ in one location, their LDE's will differ in almost all locations.

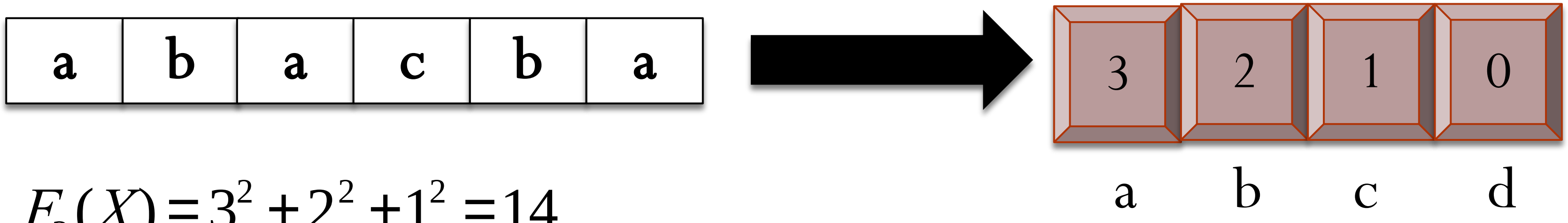
Second Frequency Moment (F_2)

- F_2 is a central streaming problem.
  - Captures sample variance, Euclidean norm, data similarity.

- Definition:
  - Let X be the frequency vector of the stream.
  - $F_2(X) = \sum_{i=1}^{n} X_i^2$

Raw data stream over universe {a, b, c, d}

Frequency Vector X



$F_2(X) = 3^2 + 2^2 + 1^2 = 14$

Second Frequency Moment

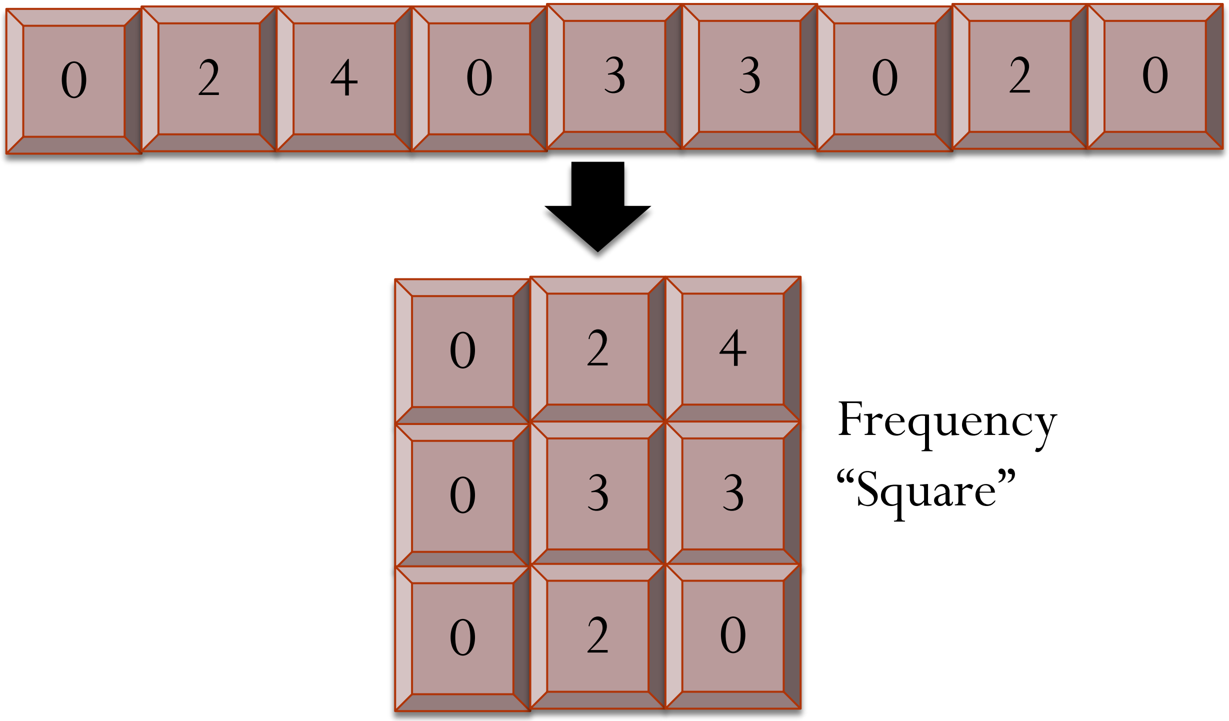
- [CCMT 12]: ($\sqrt{n}$ comm., $\sqrt{n}$ space)-protocol for F_2.
  - Terabytes of data translate to a few MBs of space and communication.

- Optimal. Lower bound of $W(n)$ on comm. * space.

F_2 Protocol

- Recall: $F_2(X) = \sum_i X_i^2$
- View universe $[n]$ as $[\sqrt{n}]$ x $[\sqrt{n}]$.

Frequency Vector X



Frequency
“Square”

- First idea: Have P send the answer "in pieces":
  - F_2(row 1). F_2(row 2). And so on. Requires $\sqrt{n}$ communication.
- V exactly tracks a row at random (denoted in yellow) so if P lies about any piece, V has a chance of catching her. Requires space $\sqrt{n}$.

Frequency Square

0	2	4
0	3	3
0	2	0

P sends

$20=2^2+4^2$

$18=3^2+3^2$

$4=2^2$

Slide derived from [McGregor 10]

- Problem: If P lies in only one place, V has small chance of catching her.

- What we'd like: if P lies about even one piece, she will have to lie about many.

- Solution: Have P commit (succinctly) to second frequency moment of rows of an **error-corrected encoding** of the input.

- Note: V can evaluate any row of the low-degree extension encoding in a streaming fashion.

Low-Degree Extension of Frequency Square

0	2	4
0	3	3
0	2	0
0	-1	-5
0	-6	-12
0	-13	-21

These values all lie on a low-degree polynomial

P sends

$20=2^2+4^2$

$18=3^2+3^2$

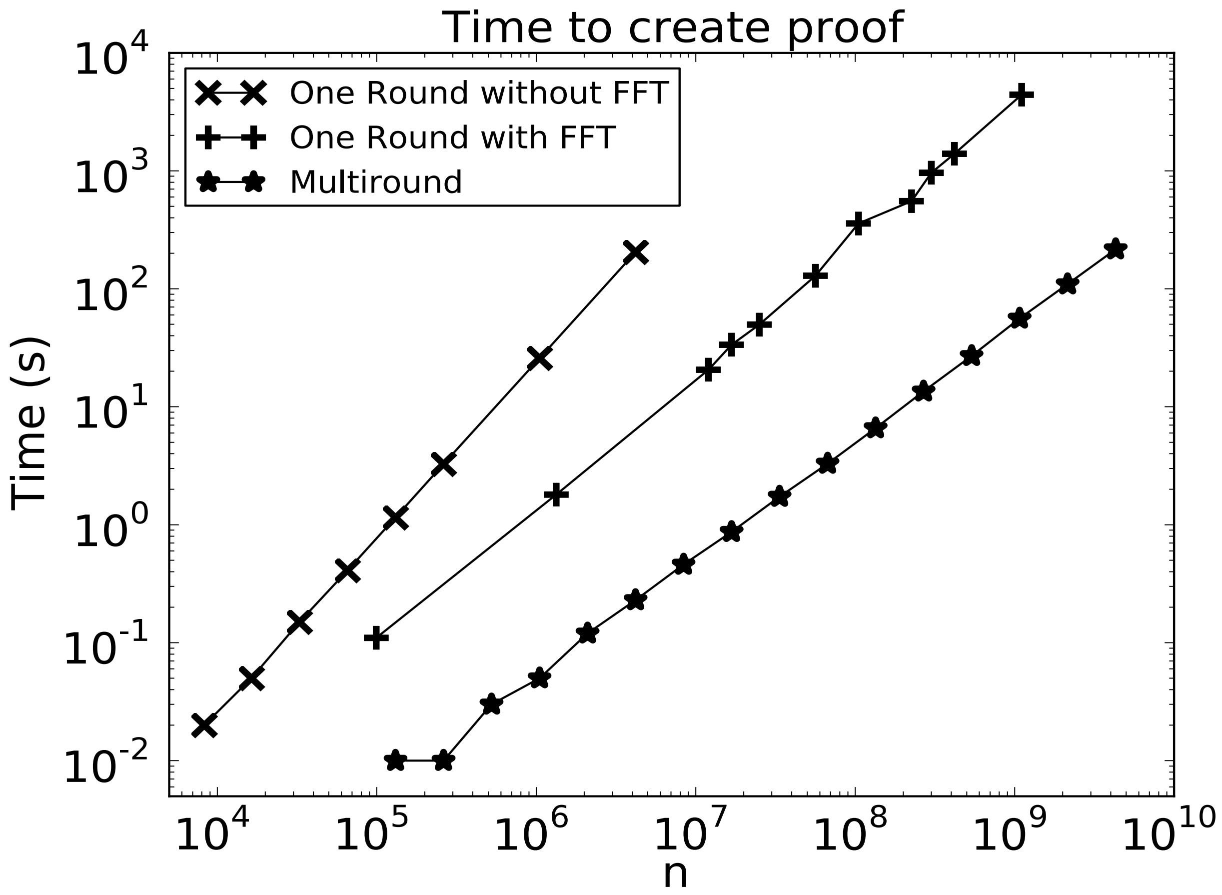
$4=2^2$

$26=(-1)^2+(-5)^2$

$180=(-6)^2+(-12)^2$

$610=(-13)^2+(-21)^2$

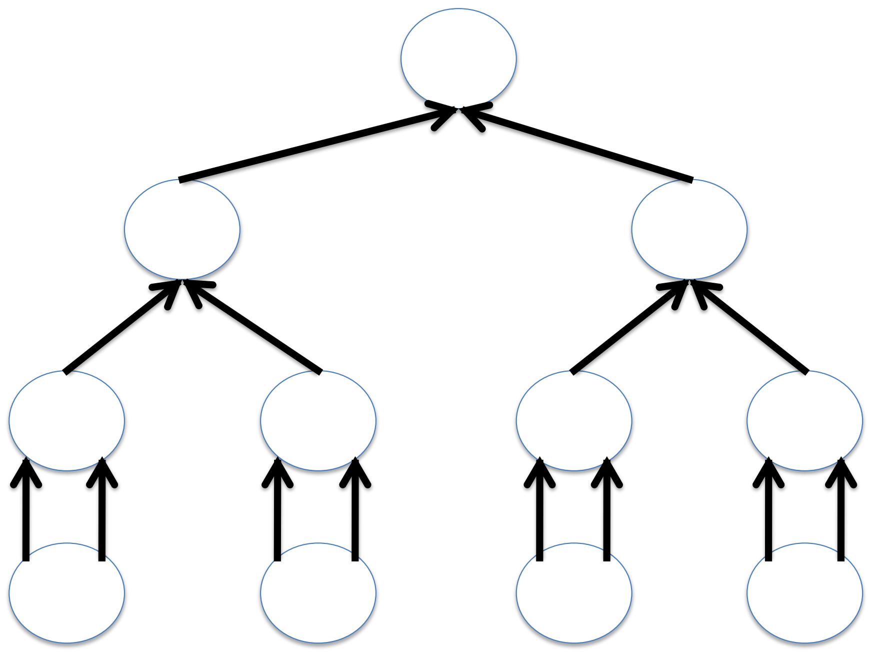
F_2 Experiments



Multi-round P from [CTY11] vs. Non-interactive P with and without FFT techniques

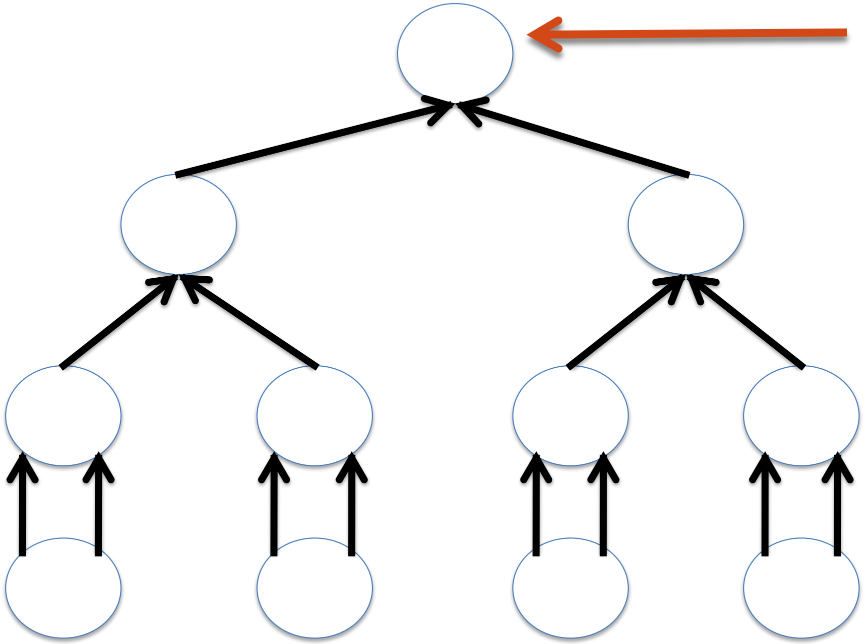
General Purpose IPs
(Extending GKR)

Circuits, Fields, and All That



F_2 circuit

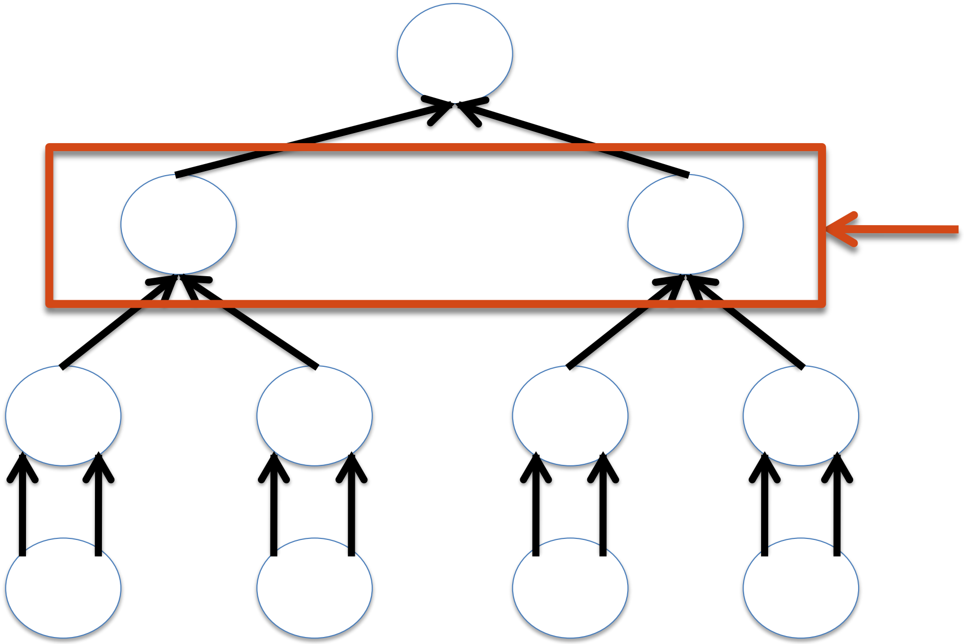
Interactive Proofs on Circuits



Prover starts the conversation with an answer (output).

F_2 circuit

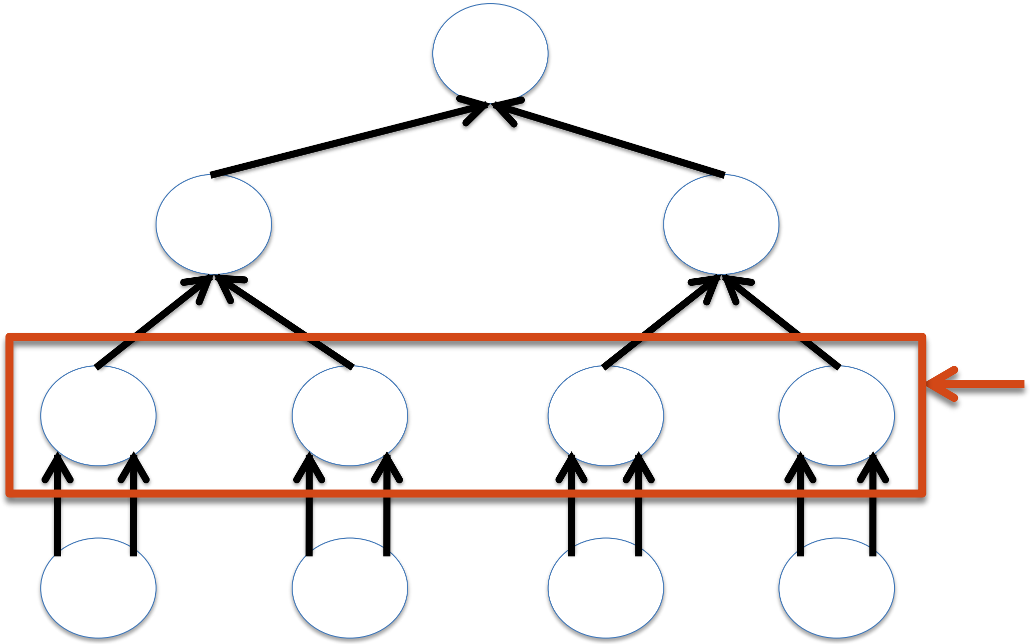
Interactive Proofs on Circuits



F_2 circuit

Verifier challenges. Prover has to respond with information about the next circuit level.

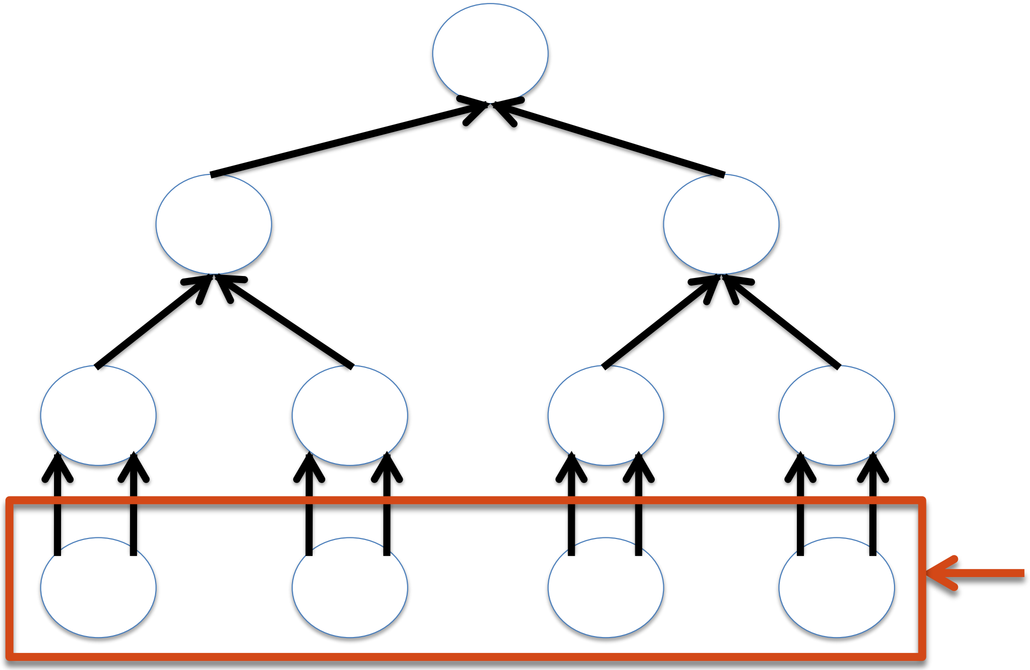
Interactive Proofs on Circuits



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

F_2 circuit

Interactive Proofs on Circuits



F_2 circuit

Finally, the Prover must say something about the input.

The verifier checks the Prover's final statement about the input, using the right kind of "fingerprint".

Saving V Space and Time [CMT12]

- Saves V substantial amounts of space (works for streaming).
- Save V substantial amounts of time.
- E.g. when multiplying two 512x512 matrices, V requires .12s, while naive matrix multiplication takes .70s.
- Savings for V will be much larger at larger input sizes, and for more time-intensive computations.

Minimizing P’s Overhead [CMT12]

- Brought P’s runtime down from $\Omega(S^3)$, to $O(S \log S)$, where S is circuit size.
- Lots of additional engineering.
  - Choosing the “right” finite field to work over.
  - Using the “right” circuits.
  - Etc.
- Practically speaking, still not good enough on its own.
  - 256 x 256 matrix multiplication takes P 27 minutes.
  - Naïve implementation of GKR would take trillions of times longer.

Reducing Overhead Further [T13]

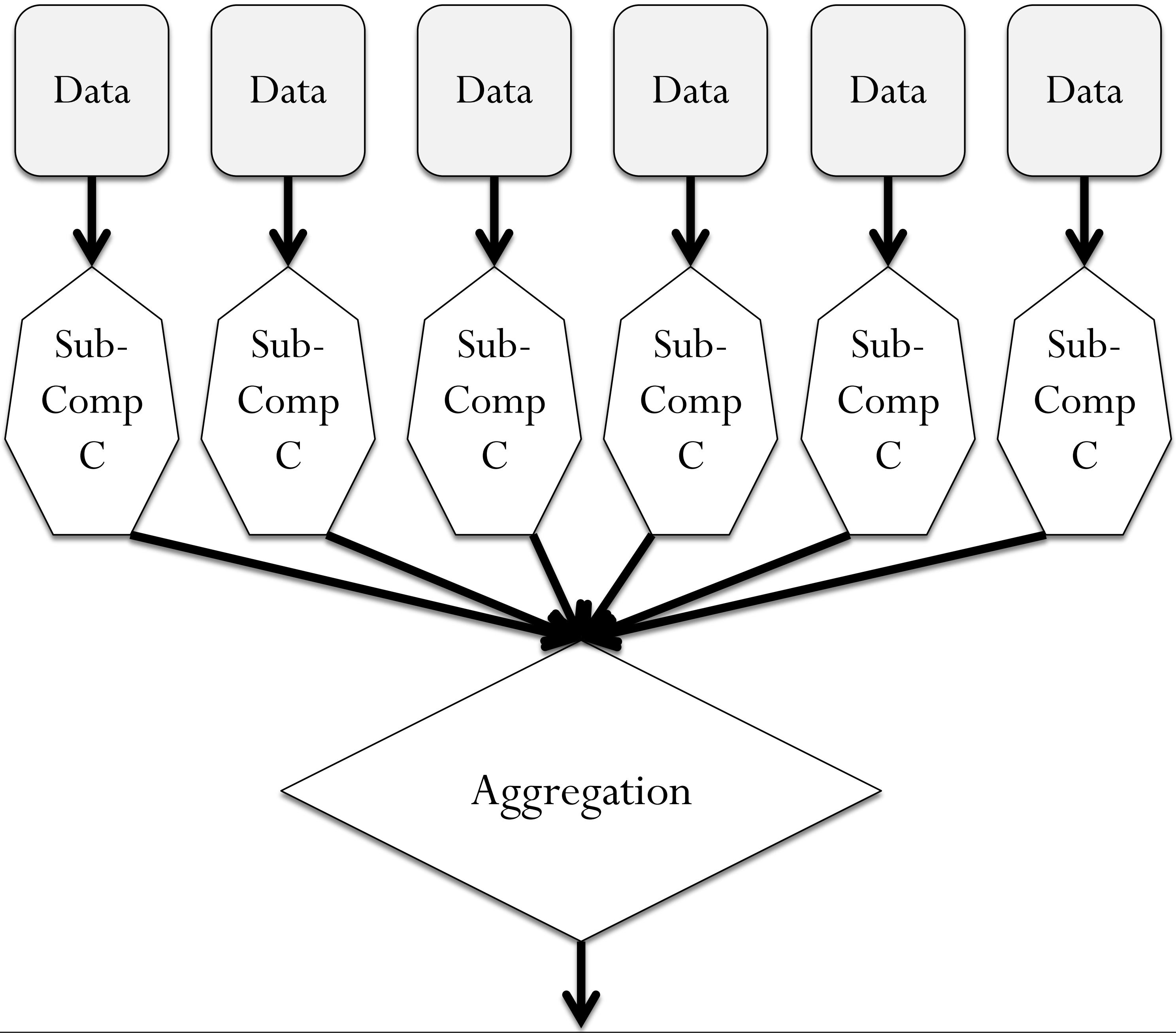
- Improvements for "regular" circuits: Reduce P's runtime to O(S).
  - Experimental results: 250x speedup over [CMT12].
  - P less than 10x slower than a C++ program that just evaluates the circuit for example applications: MatMult, DISTINCT, F_2, Pattern Matching, FFTs.

Results for Regular Circuits [T13]

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

Dealing with Irregular Circuits [T13]

- 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 [T13]

- Problem: Verify massive parallel computations.
  - Directly applying existing results has big overhead.
  - Costs depend on number of data pieces.
- 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.

A Final Result: MatMult [T13]

- Let A be **any** time t, space s algorithm for n x n MatMult.
- New MatMult protocol:
  - P takes time $t + O(n^2)$ and space $s + o(n^2)$.
  - 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

Future Directions

- Build a system that avoids the circuit model.
  - Writing computations as circuits is limiting, can blow up time for verification.
  - Can we design systems that work with general C programs?
    - In theory, mostly yes; currently prover time is impractically large.
  - Can we design systems that work with MapReduce?
- Continue pushing speed, functionality, of current systems
  - More room for improvement
- From the big data cloud to small attachable devices.
  - Imagine special purpose high-speed attachable devices for special purposes – e.g., decrypting messages, custom calculations.
    - Special ASICs, or GPUs, or…
  - These devices should be able to verify their work.