US Census Bureau Workshop on Multi-party Computing

David W. Archer, PhD

16-Nov-2017

Census First-round Adoption Concerns

- Technology maturity
- Computational overhead
- Complexity of getting this stuff to work
- Has anybody used it, ever, for any purpose?
- A description of the security guarantees and how they are achieved

A Rough Scorecard

- Technology Readiness:

TRL		
	9	Commercialized
	8	Pre-production
	7	Field Test
	6	Prototype
	5	Bench / Lab Testing
	4	Detailed Design
	3	Preliminary Design
	2	Conceptual Design
	1	Basic Concept

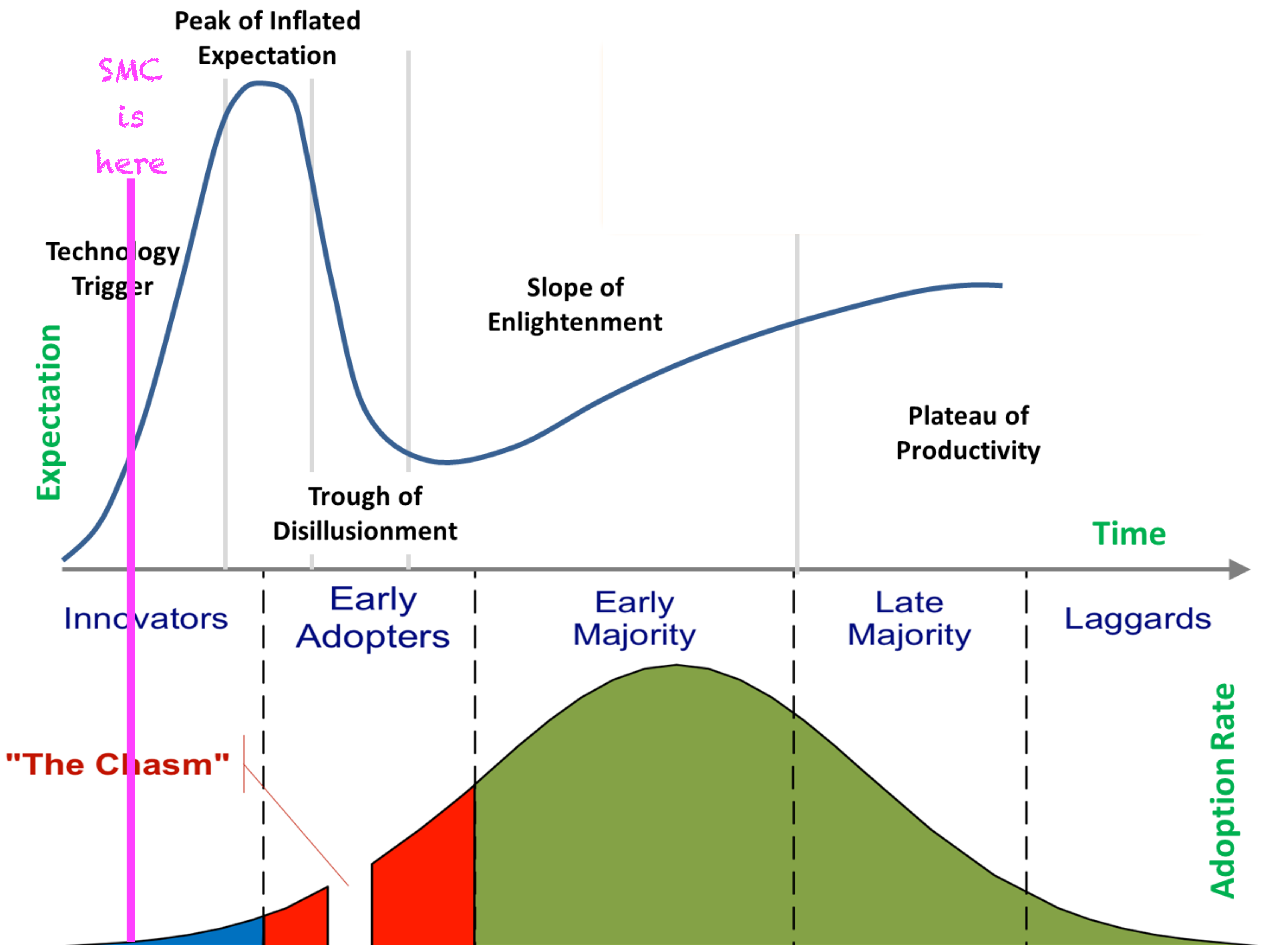
A Rough Scorecard

- Technology Readiness:
- Computational slowdown:
- Adoption Readiness:
  - Who can program it?
  - How easy to write diverse programs?
  - How easy to optimize performance?
  - How easy to deploy applications?
  - How easy to write diverse privacy policies?
- Has anybody used it, ever, for any purpose?
- What security guarantees, how achieved?
  - Privacy, Integrity, Availability, how
  - Against External user threat, Point insider threat, Distributed insider threat
  - Verifiable computation / attestation?

Adoption Readiness

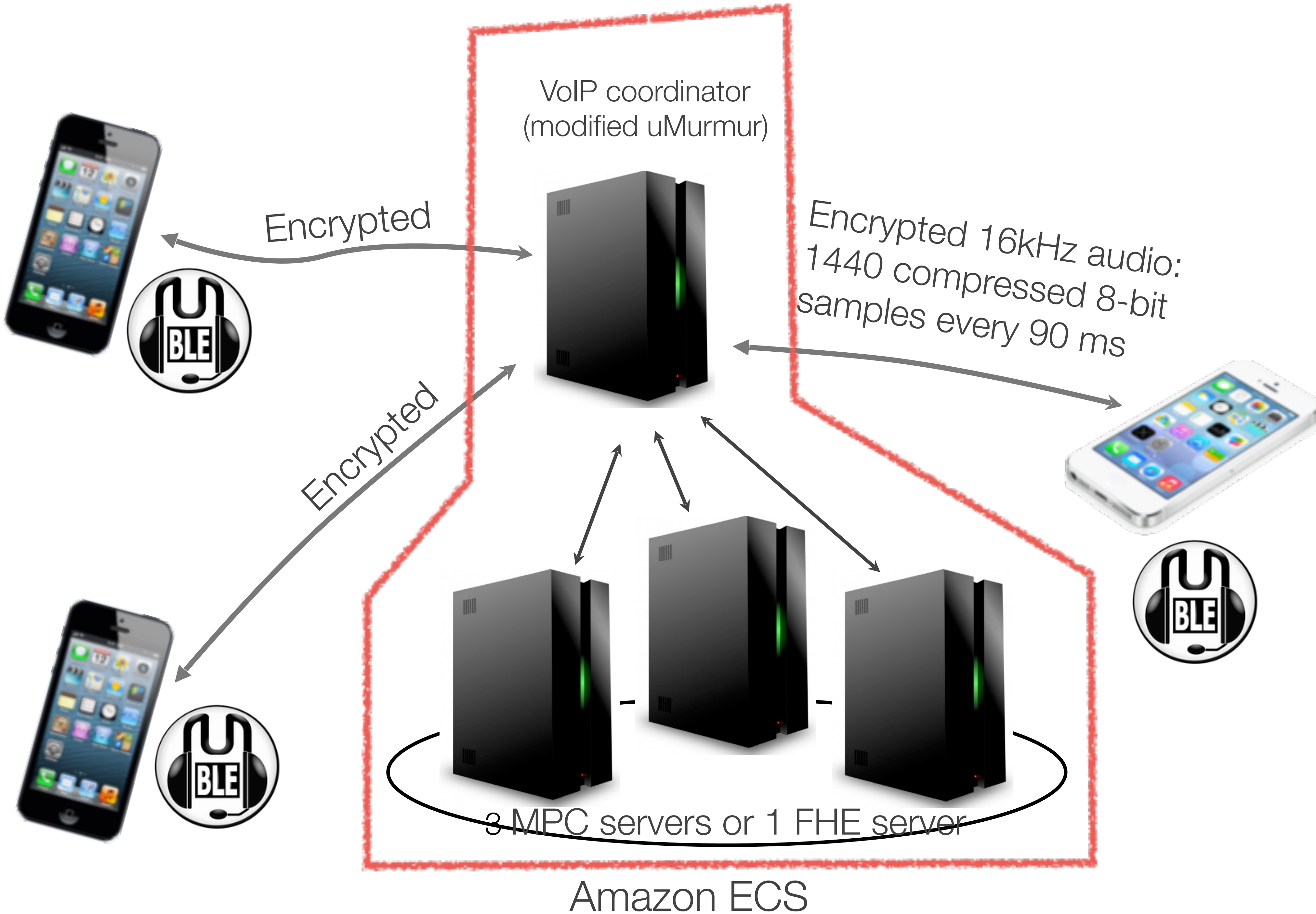
- Current Secure Computation systems resemble programming in 1950
  - When Census first moved from Hollerith tabulators to Univac-1
  - Biggest things to fix
    - Only a small handful of experts can program
    - Each system you've seen today supports only a single compute model
    - No policy flexibility or automatic compliance
    - No (or limited) *attestation* of code, nor compelling public proofs of protocols
    - No automated reasoning about feasibility or resource use
    - No "system" mindset: configuration, deployment, and clean-up
  - NOT a general programming solution for non-experts in cryptography

Adoption Readiness



An Inspiration: Streaming End-to-End Secure VoIP

LSS - 4 voices @ streaming 12kb/s audio
Archer et al. - Galois, 2014

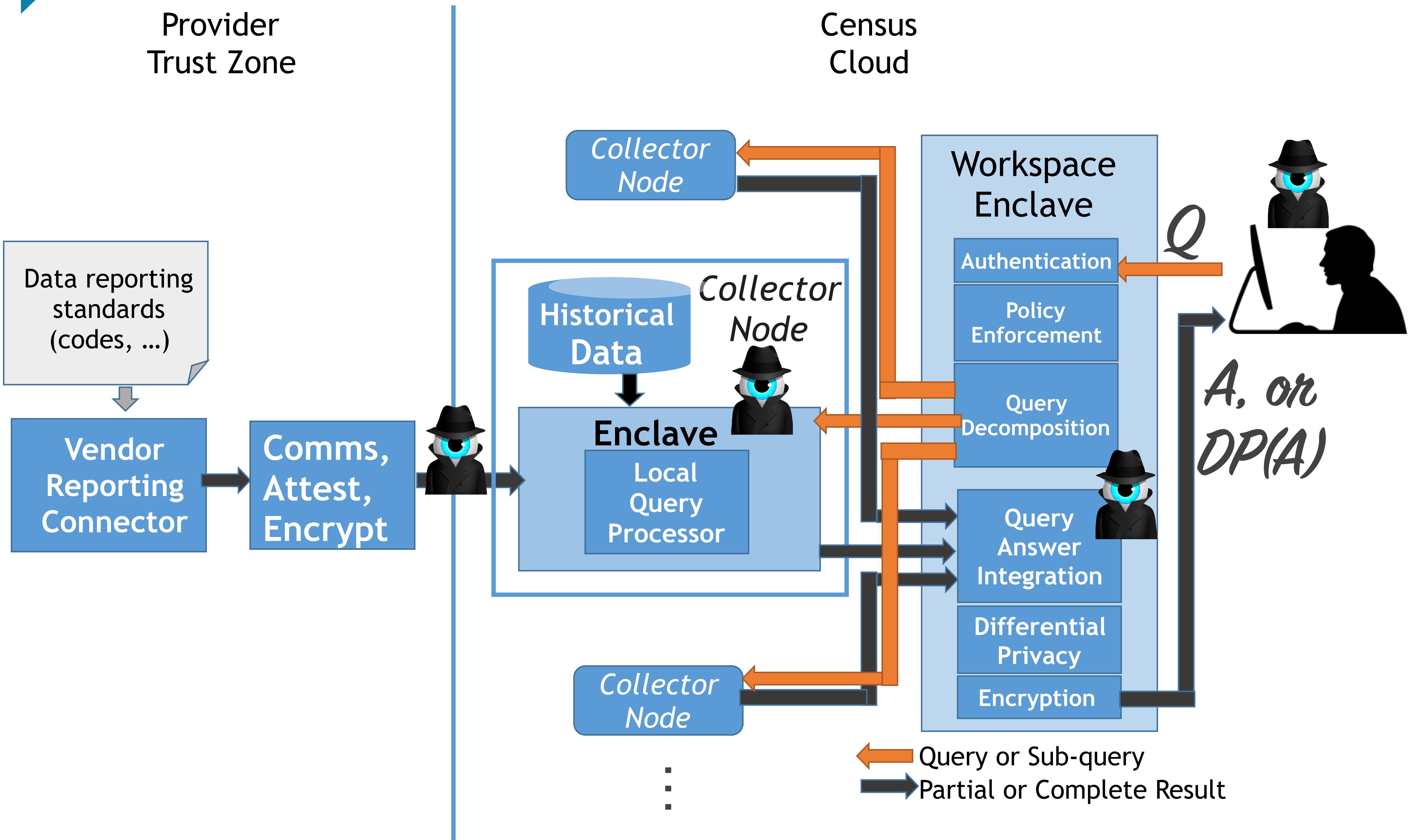


Census Use Case 2 - Requirements

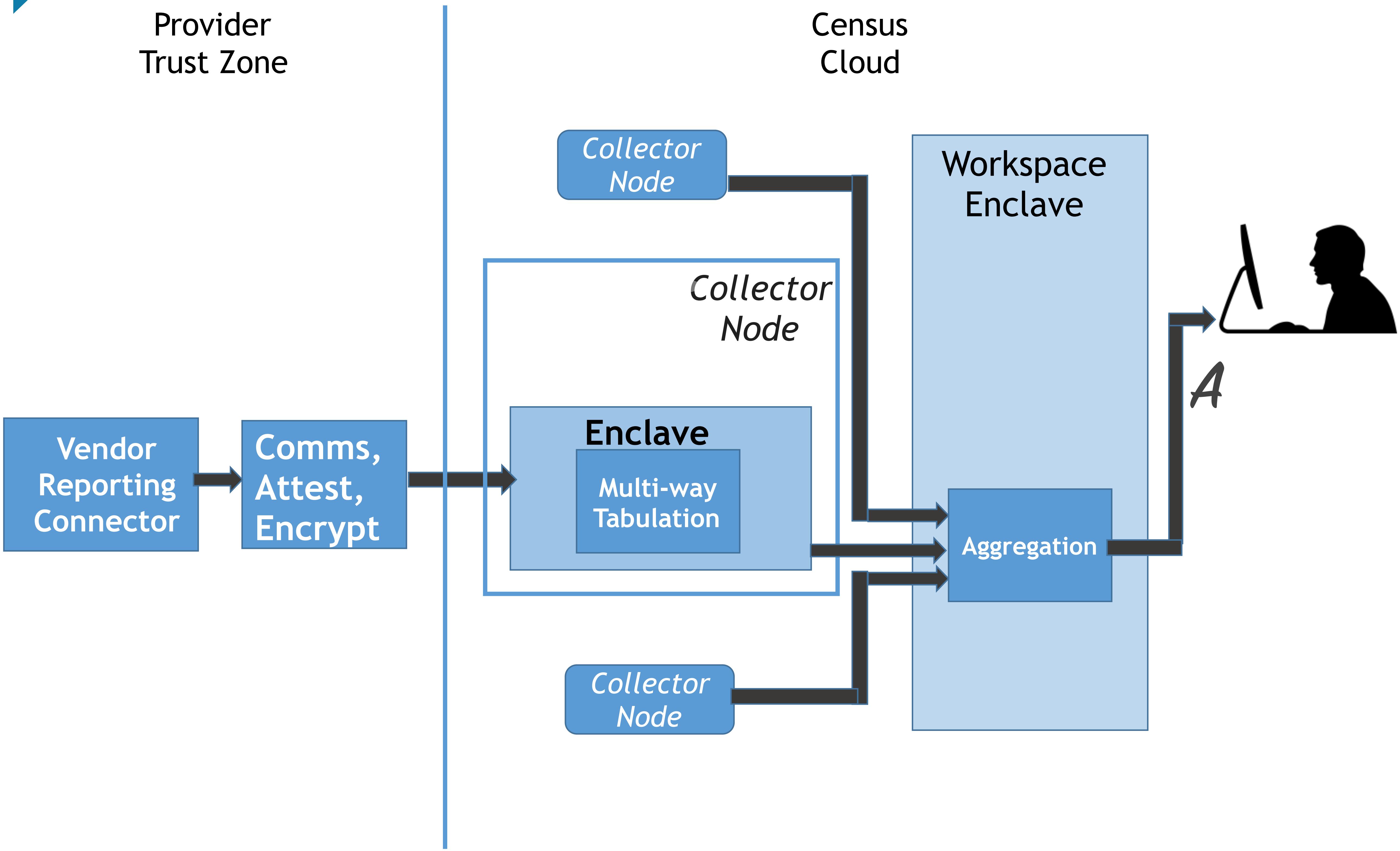
Can Census

- Track sales transactions (product, volume, price, buyer ID, seller ID)
- In ~~real~~ *streaming* time
- For multiple major companies or an entire industry
- Compute aggregate analytics: tabulations, regressions (note: requires history)
- And link to other aggregates (e.g., shipping transactions)
- While keeping all base-layer data private

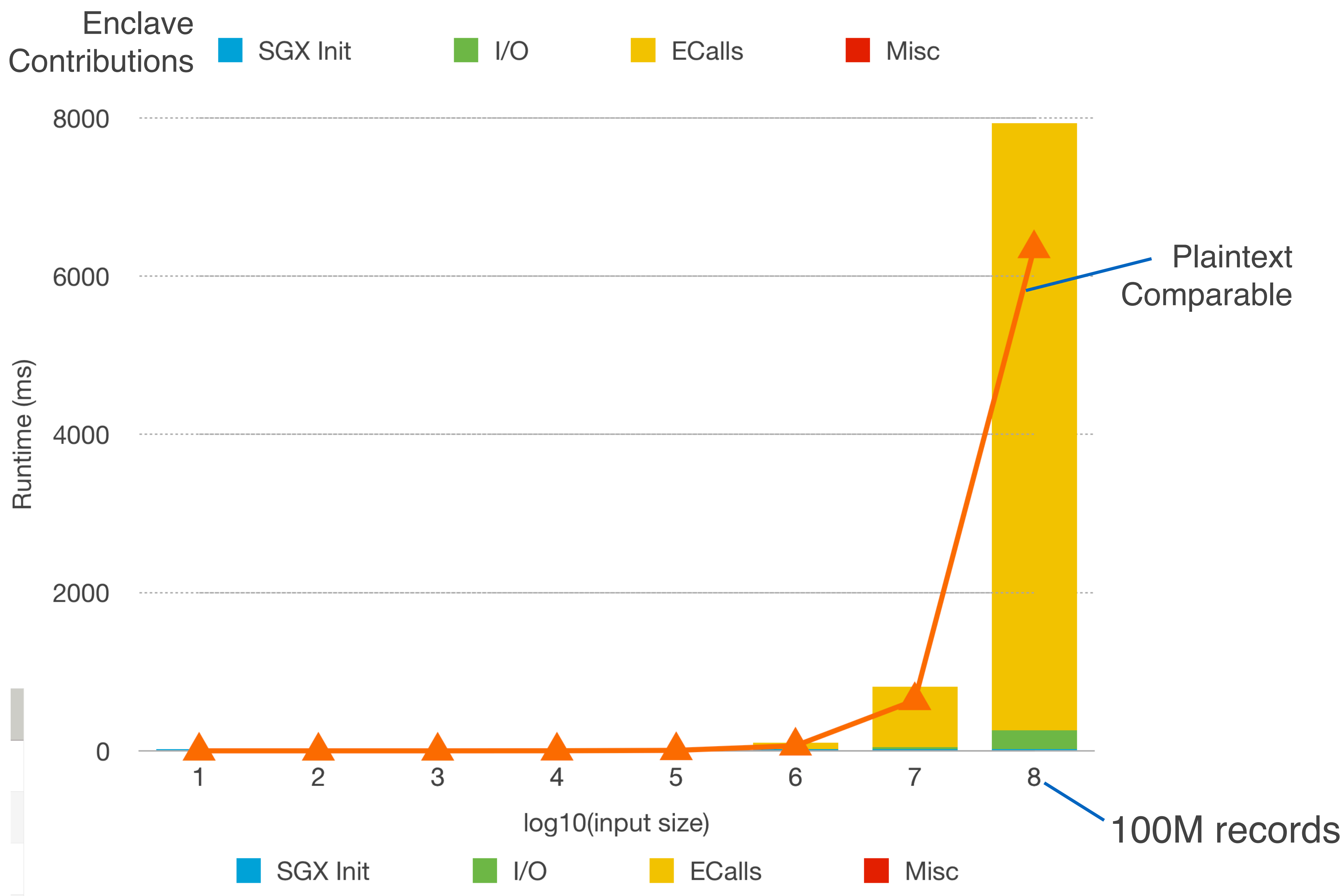
Census Use Case 2 - Goalpost



Census Use Case 2 - Today, via Intel SGX & Galois



Census Use Case 2 - Scalability and Slowdown



Use Case 2 Scorecard So Far

- Technology Readiness Level: **Intel SGX: 9, our prototype: 5**
- Computational slowdown: **~1.2 (20%)**
- Complexity of getting this stuff to work
  - Who can program it? **Anyone who knows C**
  - How easy to write diverse programs? **Easy-ish**, caveat on program size (90MB)
  - How easy to optimize performance? **Moderate** - ECalls and I/O not under app control
  - How easy to deploy applications? **On your own** (ECS offers SGX instances)
  - How easy to write diverse policies? **Coming soon in DHS-funded FIDES project**
- Has anybody used it, ever, for any purpose? **Yes, many commercial users**
- What security guarantees, how achieved?
  - **Privacy**, **Integrity**, Availability, **via hardware, soon with differential privacy**
  - Against External user threat, Point insider threat, **Distributed insider threat****
  - Verifiable computation / attestation? **Yes, through SGX remote attestation**

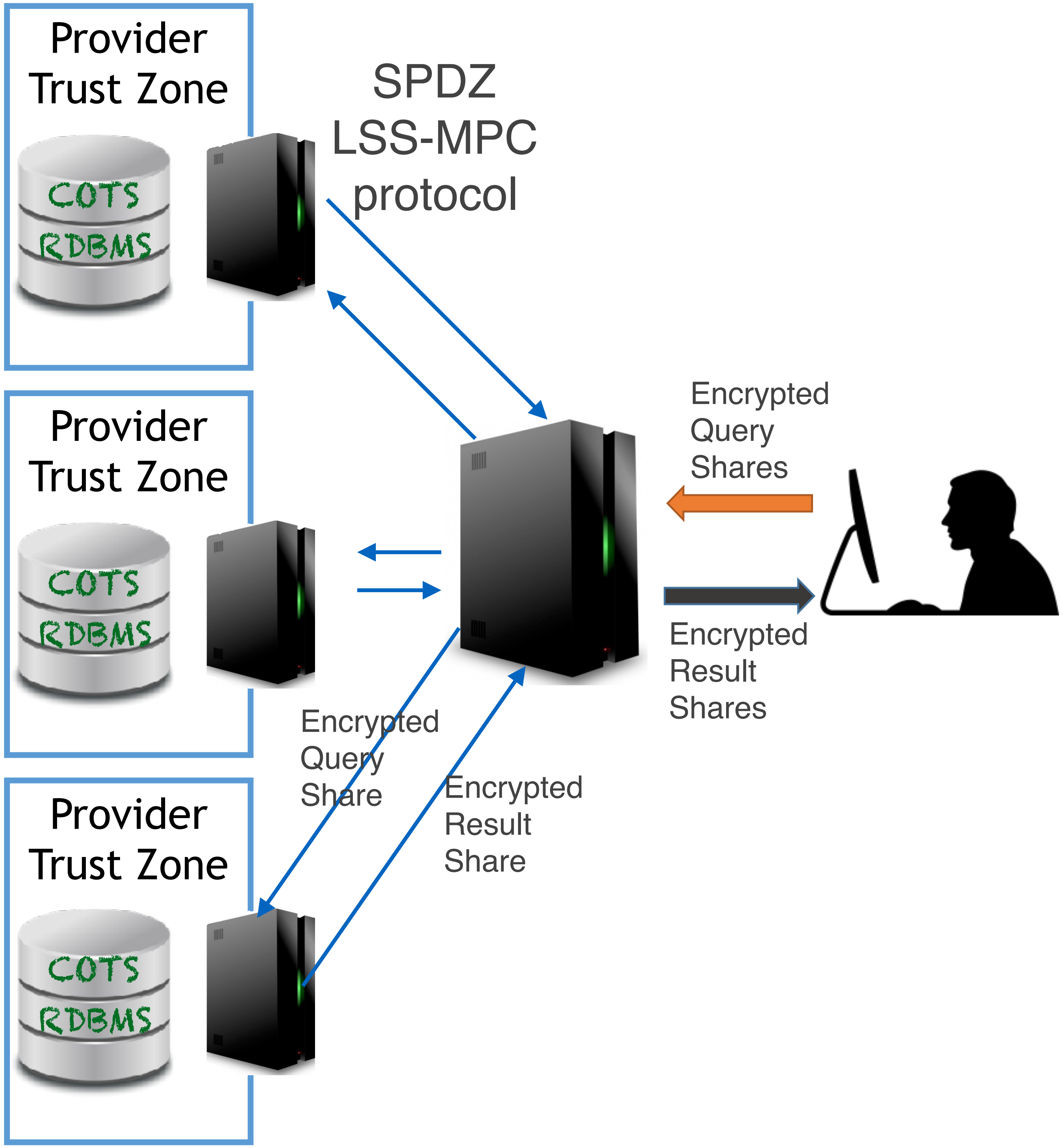
Census Use Case 1 - Requirements

Can a researcher

FERPA HIPAA XIII, XXVI

- Explore the relationship of education X health records X demographics
- Perform regression and other statistical analysis
- While data stays private to (and resident at) providing institutions

Census Use Case 1 - Goalpost



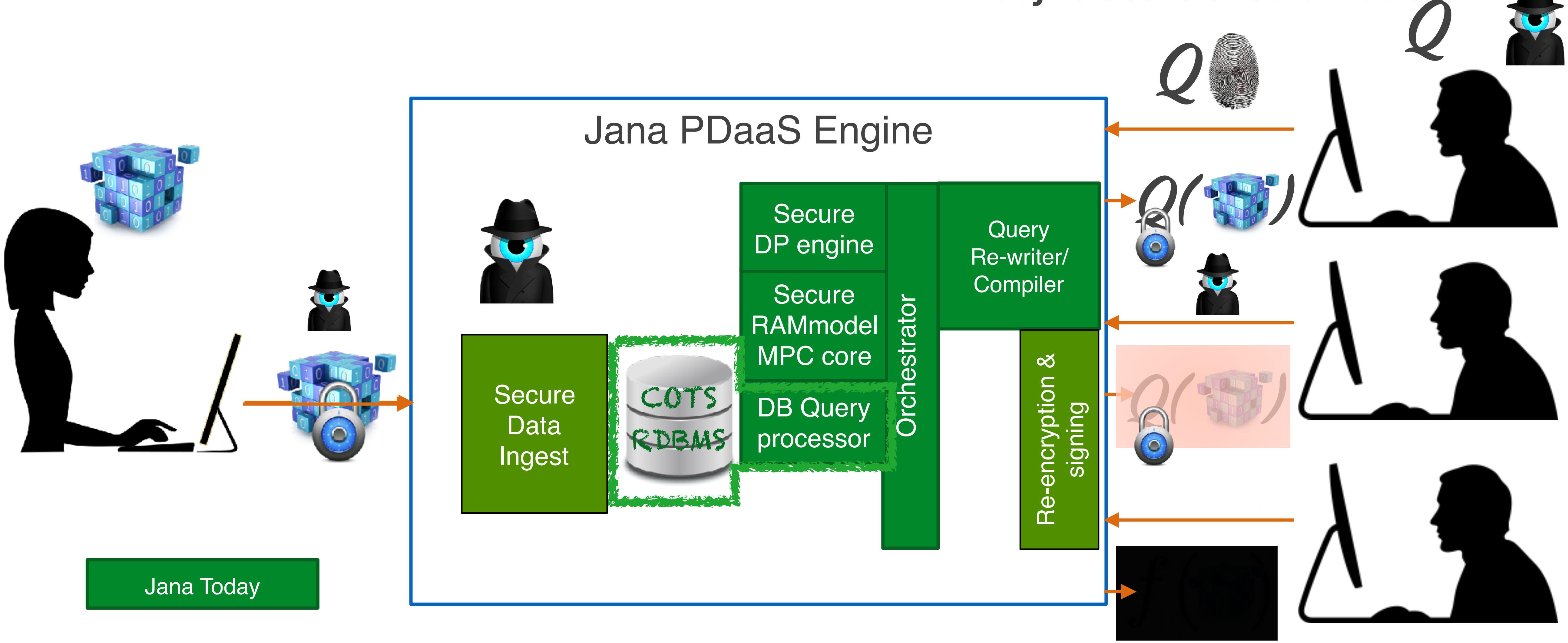
Use Case 1 Today, via Jana - Private Data as a Service

End to end++

Not pre-processed functions

Familiar, expressive: SQL + RDBMS

Easy to use: standard web serv



* - *This work funded by DARPA, by Program Manager Joshua Baron*

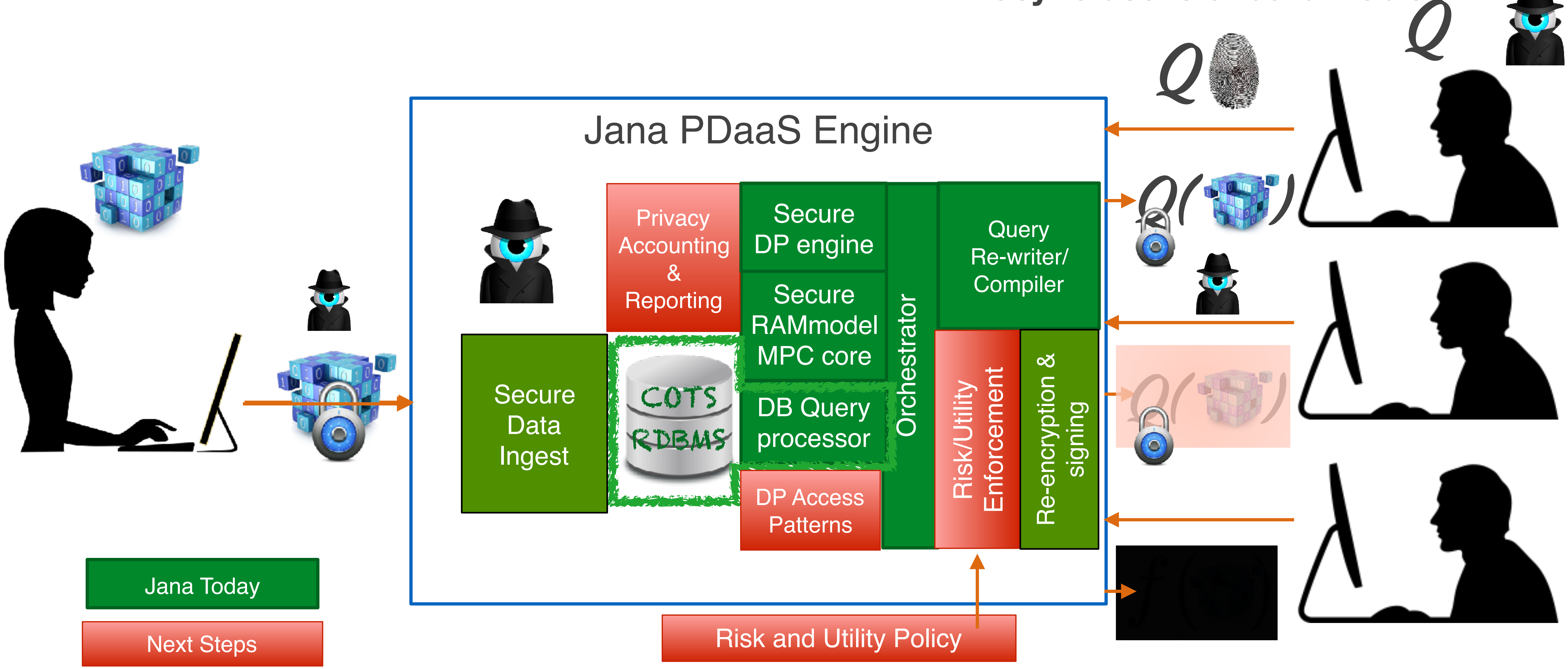
Use Case 1 Today, via Jana - Private Data as a Service

End to end++

Not pre-processed functions

Familiar, expressive: SQL + RDBMS

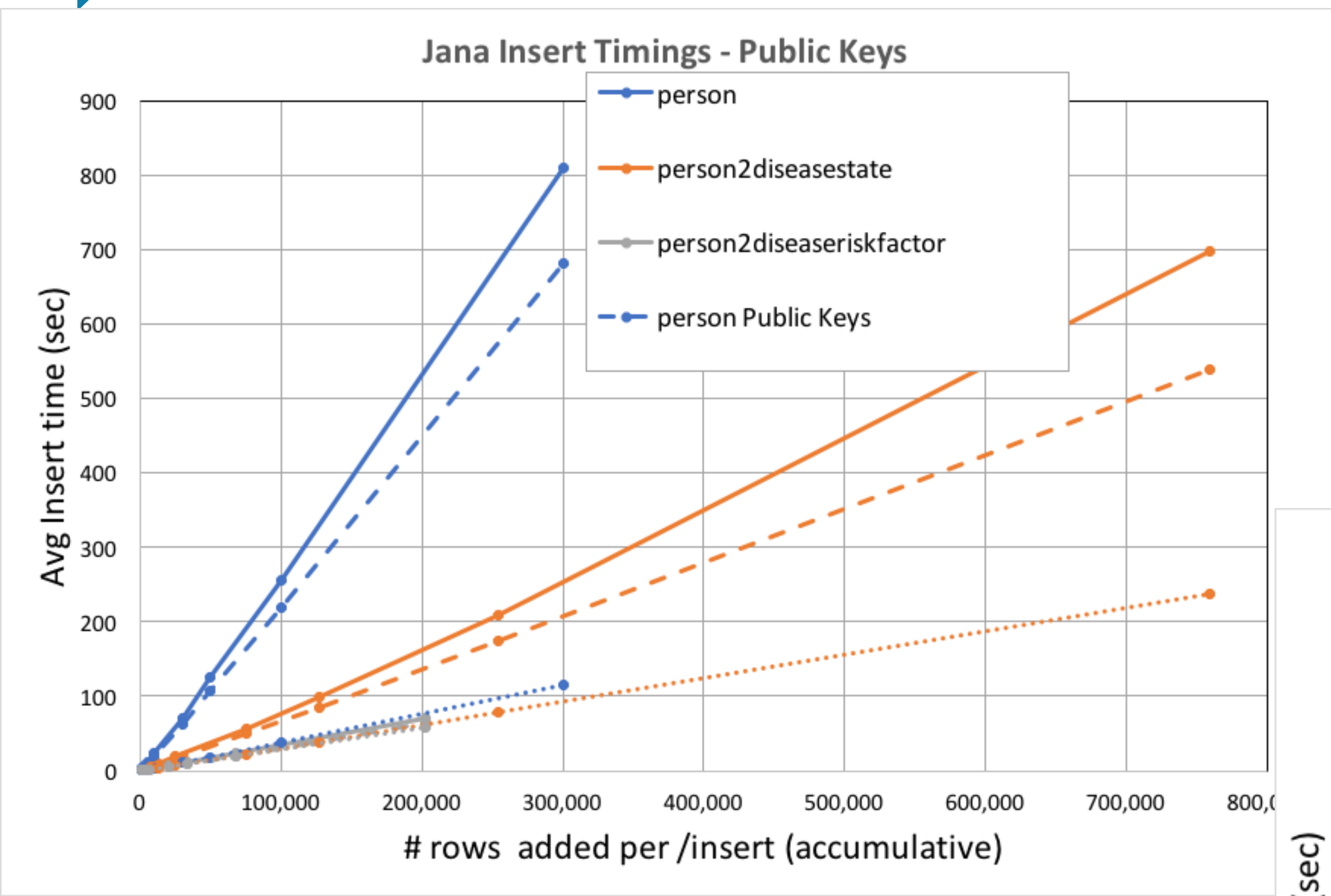
Easy to use: standard web serv



Today: Data to 100,000s of records (however, YMMV)

** - This work funded by DARPA, by Program Manager Joshua Baron*

Census Use Case 1 - Scalability and Slowdown



Use Case 1 Scorecard So Far

- Technology Readiness Level: **6 (full system, relevant env)**
- Computational slowdown: **2-1000, depending on workload**
- Complexity of getting this stuff to work
  - Who can program it? **Anyone who knows SQL (but *only* SQL)**
  - How easy to write diverse ***queries***? **Easy - supports normalized multiple relations**
  - How easy to optimize performance? **Hard - similar to relational databases**
  - How easy to deploy ***a database***? **Easy** (Available today as an appliance)
  - How easy to write diverse policies? **mid-2018, via Coull/Kenneally framework**
- Has anybody used it, ever, for any purpose? **Yes, in complex demo systems**
- What security guarantees, how achieved?

  Note: All LSS fails the Franklin test
  - **Privacy**, **Integrity**, Availability, **via LSS-MPC, searchable encryption, AES**
  - Against External user threat, Point insider threat, **Distributed insider threat****
  - Verifiable computation / attestation? **Partial, via SPDZ malicious security**

Adoption Readiness, Again

- Biggest things to fix
  - Only a small handful of experts can program, especially with generality
  - Each system supports only a single compute model
  - No privacy policy flexibility or automatic compliance
  - No (or limited) *attestation* of code, nor compelling public proofs of protocols
  - No automated reasoning about feasibility or resource use
  - No “system” mindset: configuration, deployment, and clean-up

- NOT a general programming solution for non-experts in cryptography

One Step: RAMPARTS*

- Assess feasibility of
  - General SMC programming without deep crypto expertise

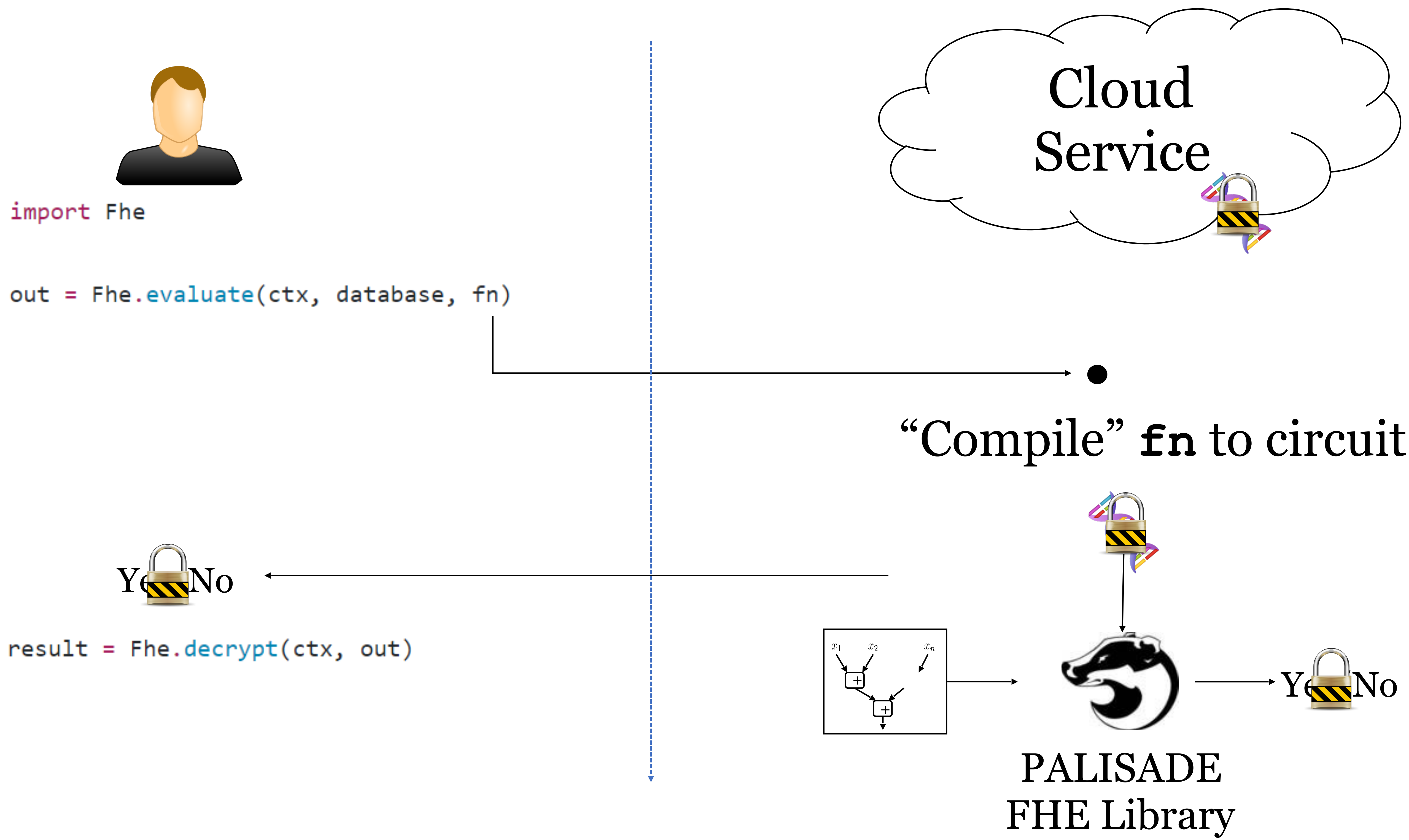
```
import Fhe

ctx = Fhe.FheContext()
Fhe.keygen(ctx)
ciphertexts = Fhe.encrypt(ctx, [1, 2, 3])
result = Fhe.evaluate(ctx, ciphertexts, sum)
println("Result: ", Fhe.decrypt(ctx, result))
```

Evaluate *existing* Julia built-ins or any user-defined function

  - Automatic parameterization
  - Automatic resource use estimation
  - Automatic DevOps-style deployment and result integration

** - This work funded by IARPA, by Program Manager Mark Heiligman*


Cloud
Service
import Fhe
out = Fhe.evaluate(ctx, database, fn)
"Compile" fn to circuit
Yes No
result = Fhe.decrypt(ctx, out)
PALISADE
FHE Library
Yes No

Why Symbolic Execution?

- FHE uses circuits statically configured *before* execution

But…

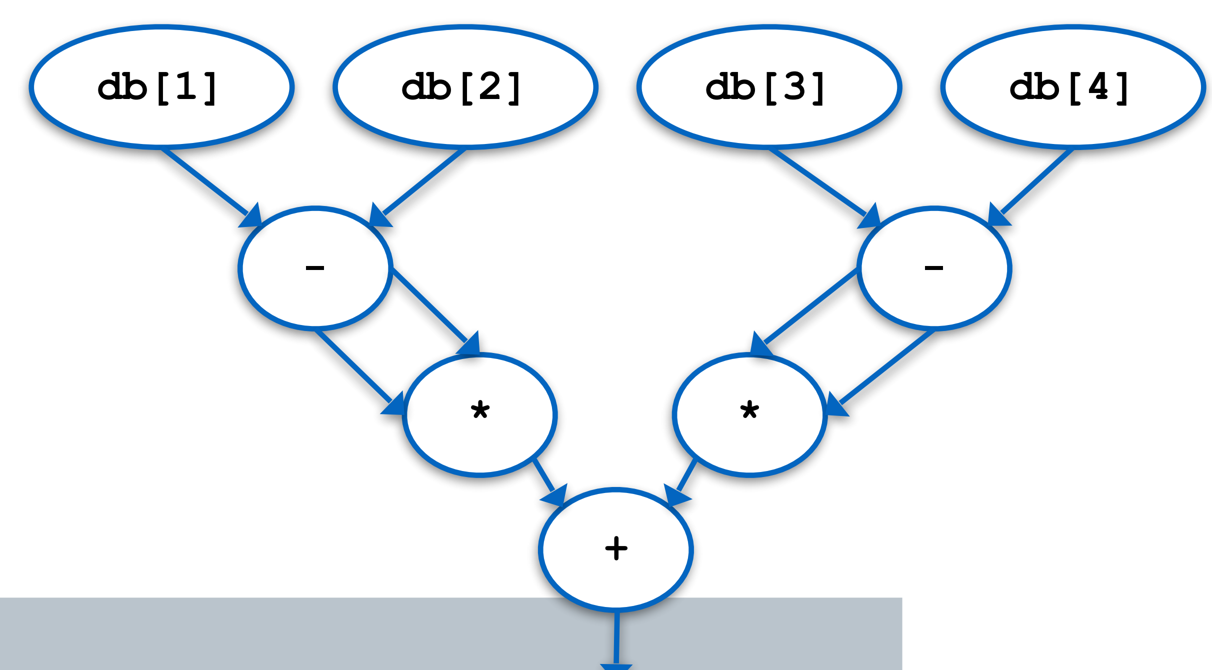
- (Imperative) programs dynamically configured *during* execution

To cross evaluation gap, use *symbolic execution*

  - ***Interpret*** (almost) all execution paths in the program
  - ***Express*** program values symbolically rather than concretely
  - ***Encode*** terminal expressions for values as logic or arithmetic circuits

```
a = db[1] – db[2]
b = db[3] – db[4]
return a*a + b*b
```

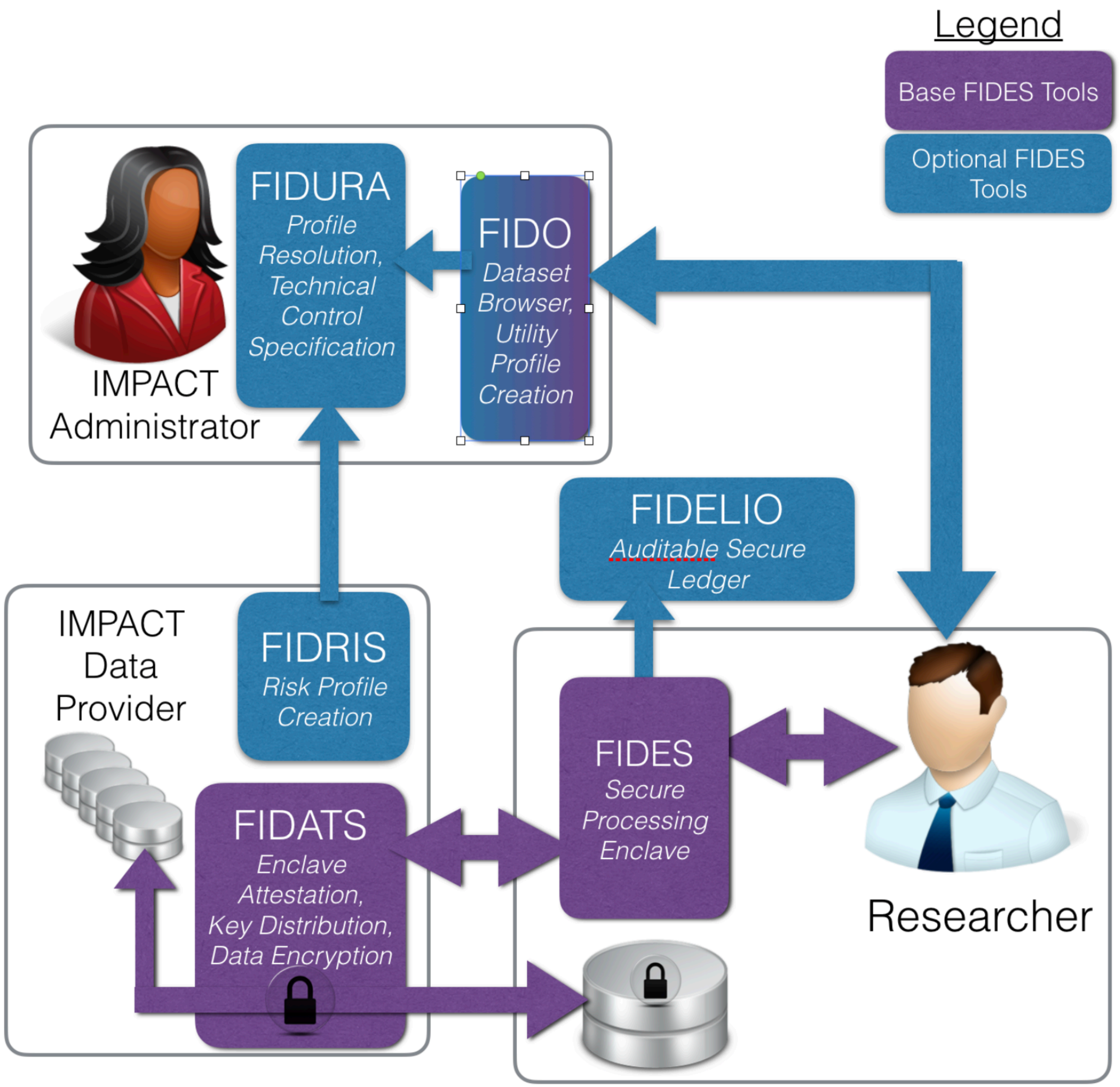
Symbolic simulation



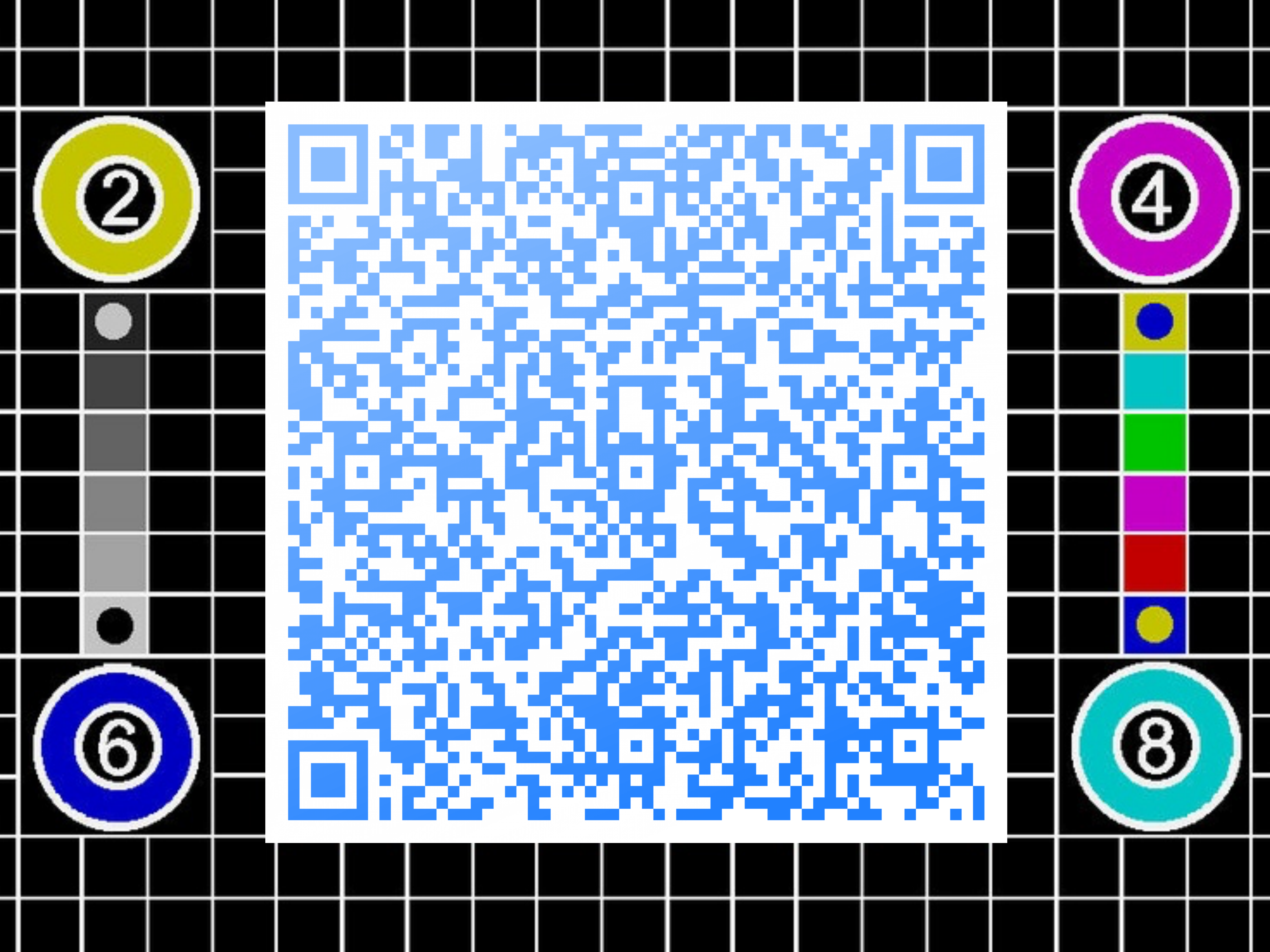
RAMPARTS Scorecard So Far

- Technology Readiness Level: **6**
- Computational slowdown: **Consistent with PALISADE FHE backend**
- Complexity of getting this stuff to work
  - Who can program it? **Anyone who knows Julia**
  - How easy to write diverse programs? **Easy, via symbolic simulation**
  - How easy to optimize performance? **Easy-ish: circuit optimization built in**
  - How easy to deploy applications? **Easy: automatic**
  - How easy to write diverse policies? **Not implemented - hand-parameterized**
- Has anybody used it, ever, for any purpose? **In demonstrations**
- What security guarantees, how achieved?
  - **Privacy**, **Integrity**, Availability, **via FHE**
  - Against External user threat, Point insider threat, **Distributed insider threat****
  - Verifiable computation / attestation? **Not yet - unsolved research problem**

DHS S&T IMPACT: FIDES project



** - This work funded by DHS S&T, by Program Manager Erin Kenneally*


2
4
6
8