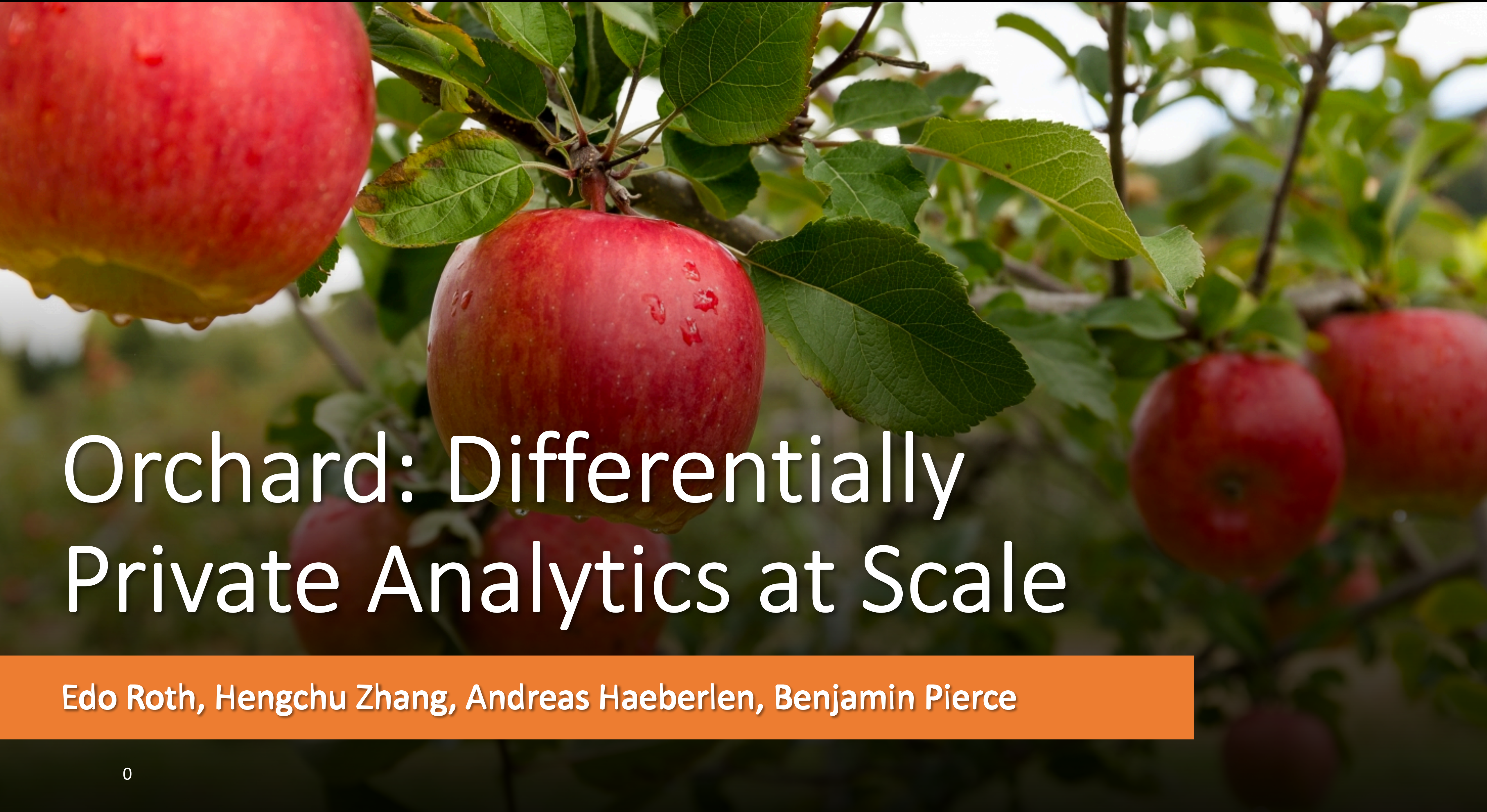
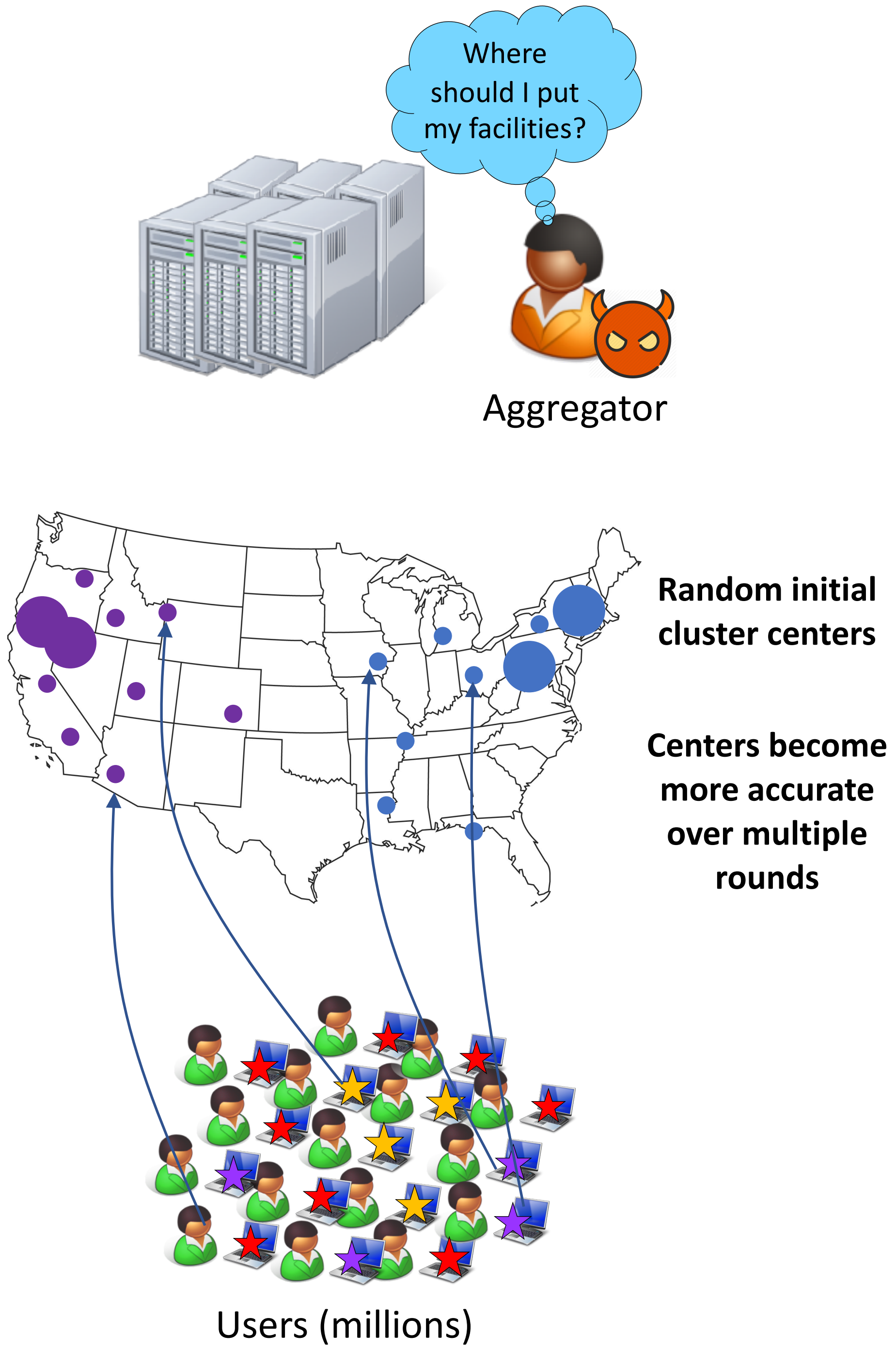


Orchard: Differentially Private Analytics at Scale

Edo Roth, Hengchu Zhang, Andreas Haeberlen, Benjamin Pierce

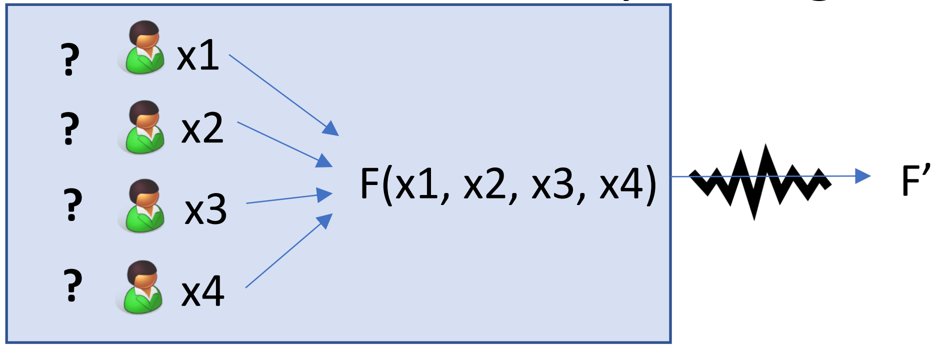
Scenario

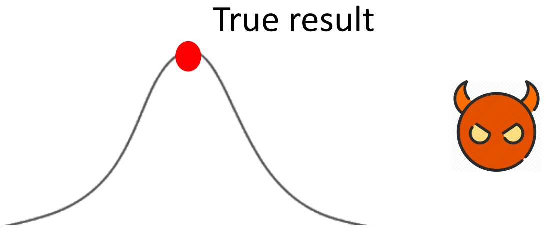
- Task: A large retailer wants to determine where to build extra shipping containers
- Users have devices (mobile or desktop) with an app that has their location
- If we had a central database, could run k-means over users' locations
- Concern: **privacy!**



Goals

- ***Leave raw data on users' devices*** and run a distributed protocol to compute the results that the aggregator wants
- **Differential Privacy** to reason about aggregative sensitive information
  - Gold standard for privacy - strong, formal guarantee
  - Hides individual contributions by adding random noise

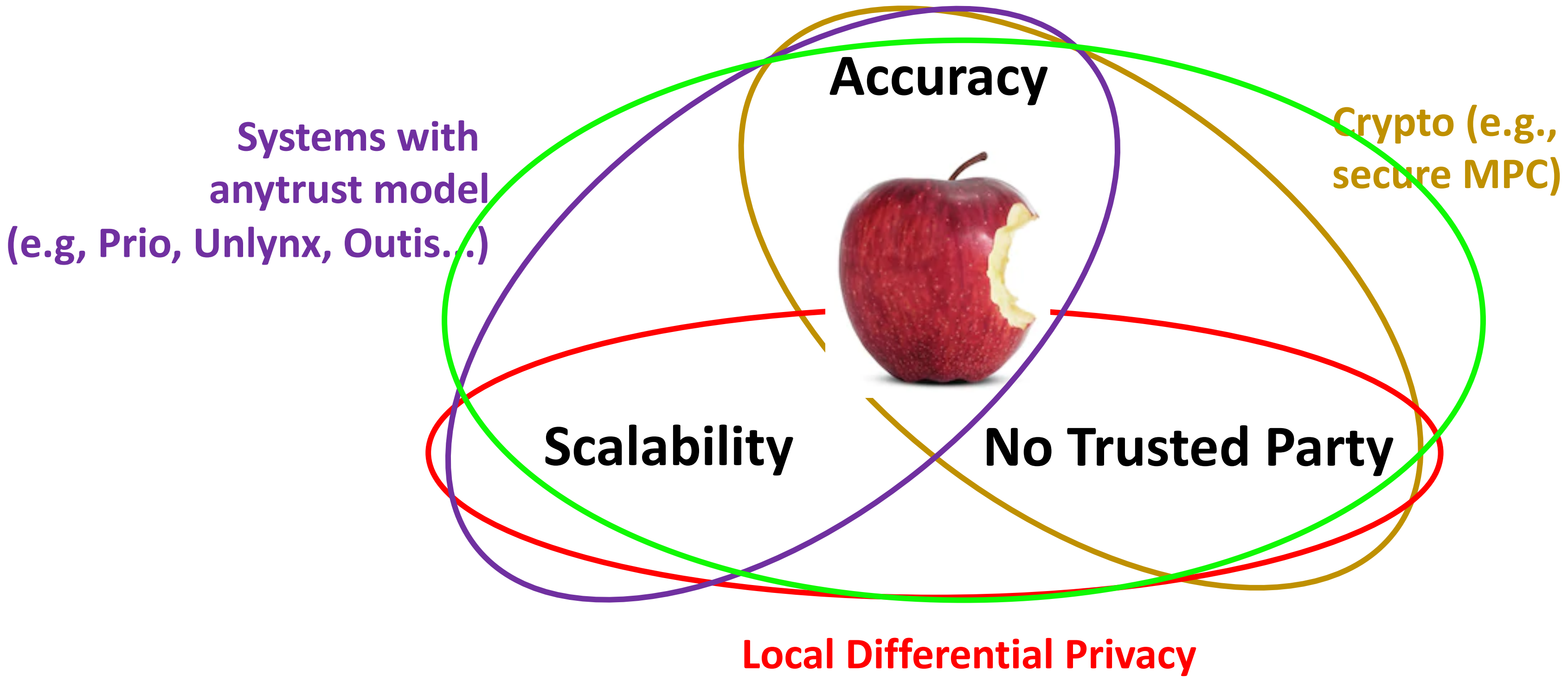




- What else might we desire out of a system like this?
  - **Scalability** to millions or even > 1 billion users!
  - **Accuracy** of query results
  - **No Trusted Party** – can be hard to find in practice

Related Work

Differential Privacy in Practice



Honeycrisp [SOSP ’19] can achieve all of this!

Challenges

BUT:

- Honeycrisp can only do one specific query – Count Mean Sketch, motivated by Apple iOS
  - Essentially sums up local sketches

- There are LOTS of other queries people might want to ask

Query
ID3
K-means
Perceptron
PCA
Logistic Regression
Naive Bayes
Neural Network
Histogram
K-median

Query
CDF
Range queries
Bloom filters
Count Mean Sketch
Sparse Vector
DStress
PATE
Iterative Database Construction

Honey-crisp!

Do we need to build a new system for each one?

Insight

Observation: many queries can be transformed into:

node-local operations

+ sequences of sums

(w/ some public computation)

Not a coincidence – a natural consequence of DP mechanisms

We can transform many complex queries into ones that use mostly sums!

Honeycrisp can be run on most queries

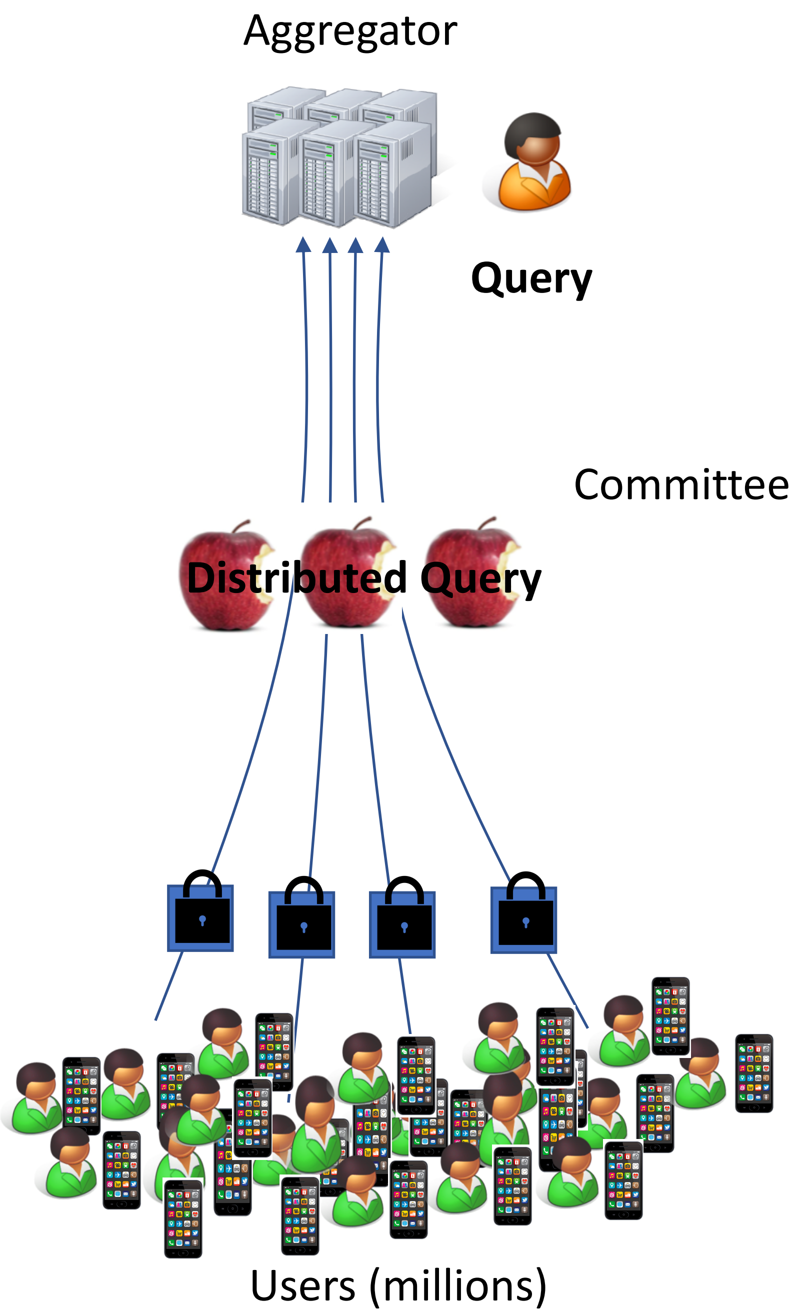
(with a small generalization)

K-means code

```
assign c1 c2 c3 pt =
    let d1 = sqdist c1 pt
    d2 = sqdist c2 pt
    d3 = sqdist c3 pt
    in if d1<d2 and d1<d3 then 0 else if d2<d1 and d2<d3
    then 1 else 2
noise totalXY size = do
    let (x, y) = totalXY
    in do x' ← lap 1.0 x
    y' ← lap 1.0 y
    size' ← lap 1.0 size
    return (x'/size', y'/size')
totalCoords pts =
    let ptxs = bmap fst pts
    ptys = bmap snd pts
    in (bsum 1.0 ptxs, bsum 1.0 ptys)
countPoints pts = bsum 1.0 (bmap (\pt → 1) pts)
step c1 c2 c3 pts =
    let [p1, p2, p3] =
        bpartition 3 (assign c1 c2 c3) pts p1TotalXY =
        totalCoords p1
        p1Size = countPoints p1
        p2TotalXY = totalCoords p2
        p2Size = countPoints p2 p3TotalXY = totalCoords p3
        p3Size = countPoints p3
    in do
        c1'  ← noise p1TotalXY p1Size
        c2'  ← noise p2TotalXY p2Size
        c3'  ← noise p3TotalXY p3Size
        num_iter++
return (c1', c2', c3')
```

Orchard Workflow

- Aggregator writes (centralized) query
- Orchard translates to a distributed query
- Users process local data and encrypt results
- Encrypted results are securely aggregated
- Committee adds noise and returns query result
- Aggregator sees only the result, but <u>never</u> any individual's data



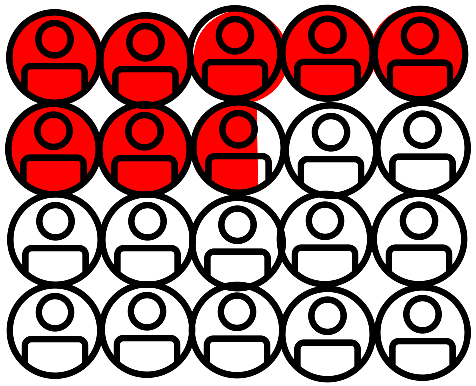
Outline (Rest of talk)

- High-Level
  - Scenario
  - Challenges
  - Insight
  - Orchard Workflow
- **Orchard Details**
  - Threat Model
  - Orchard Map
  - Orchard Walk-Through
  - Optimizations
  - Defense against Malicious Users
- Evaluation

Threat Model

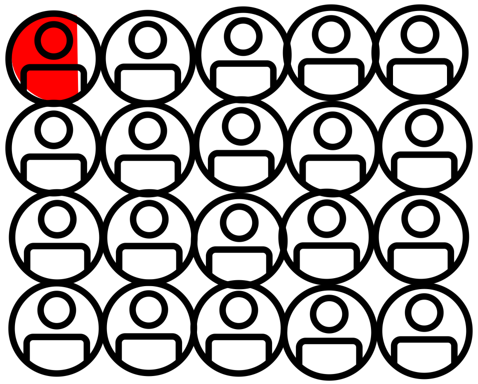
- Aggregator is Honest-but-Curious but **occasionally Byzantine**
  - For short periods of time, not at the beginning
- Users are **mostly correct**
  - A small fraction of the users (~1-5%) can be Byzantine

1/3 malicious = 430mil devices!



Byzantine Fault Tolerance Literature

3% malicious = 39mil devices



Honeycrisp + Orchard

Target: 1.3 billion iOS devices!

Orchard Map

- Programs are composed of BMCS calls
    1. Broadcast – any state required between rounds
    2. Map – local function by each user
    3. Clip – for differential privacy
    4. Sum – Aggregation of ciphertexts

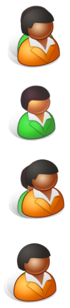
one complete round of Honeycrisp!

- Orchard transforms an original query:
    1. Recognizing boundaries of these rounds
    2. Splitting them up automatically
    3. Reducing program to multiple rounds of Honeycrisp

Orchard Walk-Through

B M C S

Aggregator

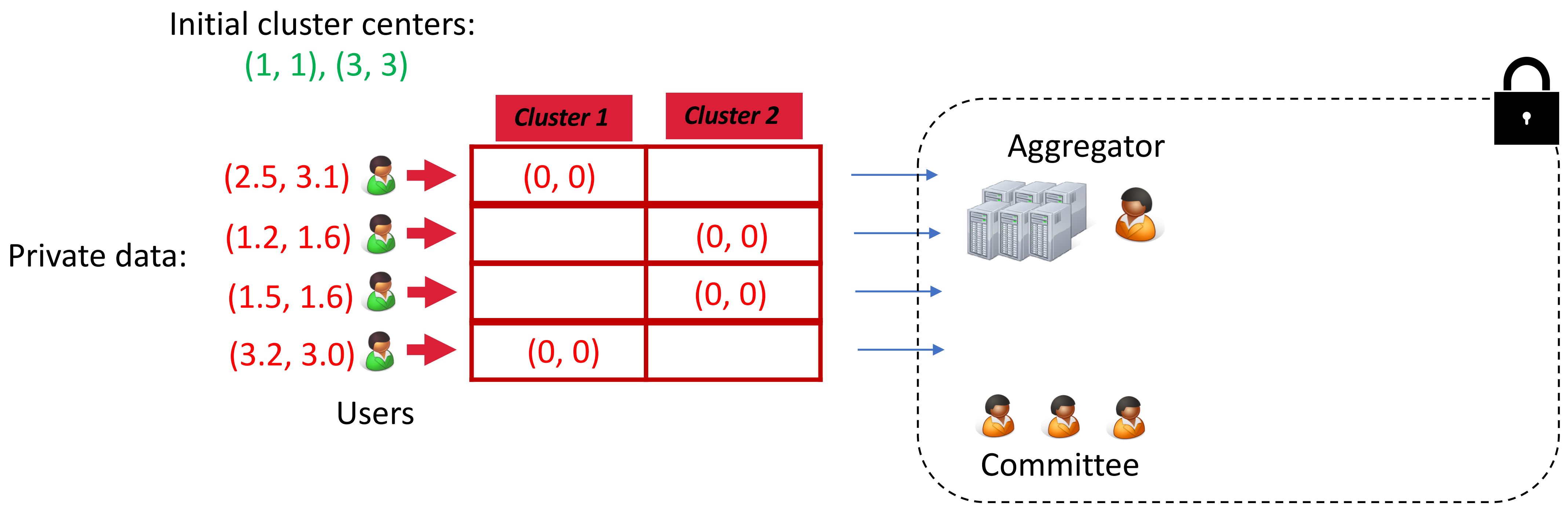


Users

Committee

Orchard Walk-Through

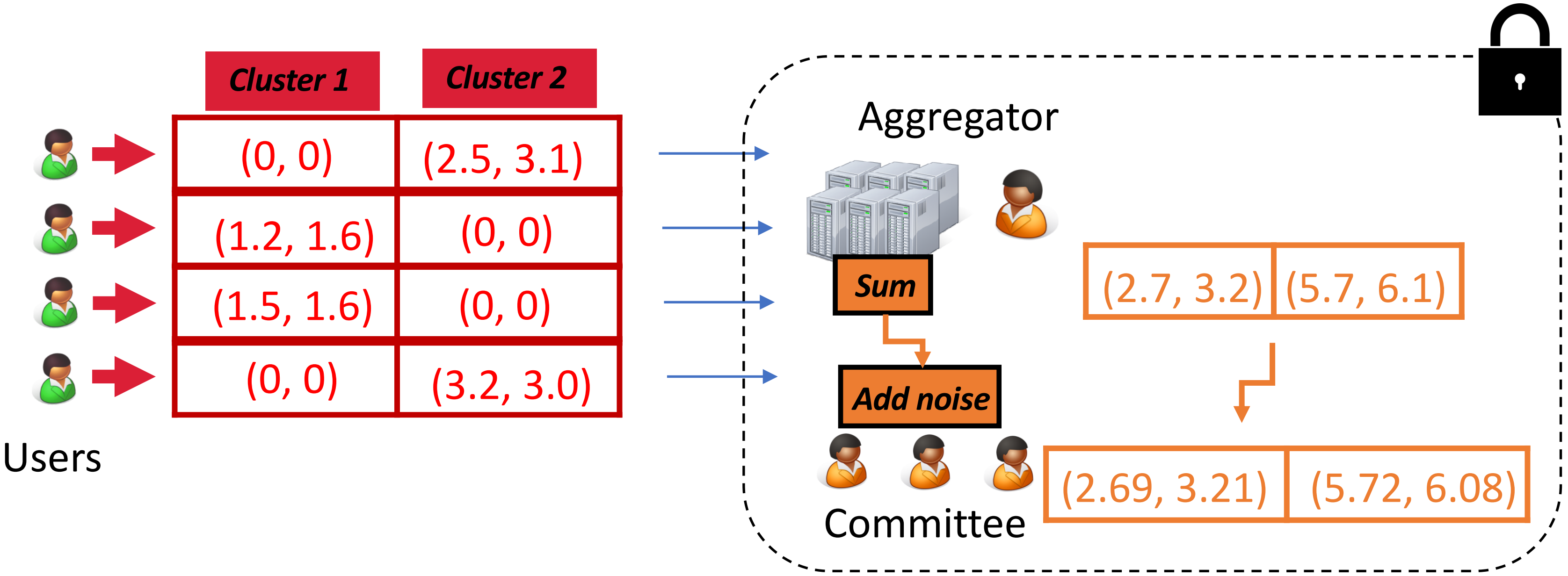
B **M** **C** S



Users **map** their data according to the query & perform **clipping** as needed, encrypting the results

Orchard Walk-Through

BMCS

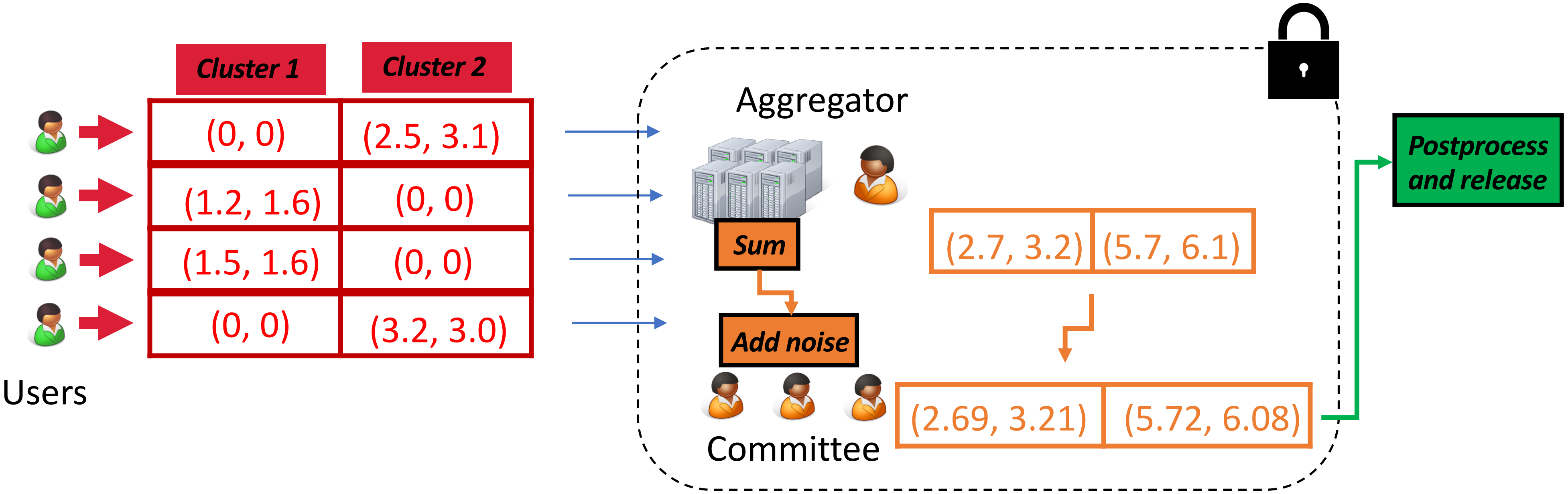


The committee adds noise to the **summed** (encrypted) result & jointly decrypts it

Orchard Walk-Through

BMCS

New cluster centers:
(1.345, 1.605), (2.86, 3.03)



The (differentially private) result is released to the aggregator, who may initiate subsequent rounds through **broadcast**

Orchard Zones

- Orchard has 3 zones (red, orange, and green)
  - Roughly correspond to:
    - users [map, clip],
    - committee members [sum], and
    - aggregator [broadcast]

- These zones naturally exist in most DP query languages, not just ours!

K-Means Example

```
while (num_iter < total_iters):
    assign c1 c2 c3 pt =
        let d1 = sqdist c1 pt
        d2 = sqdist c2 pt
        d3 = sqdist c3 pt
        in if d1<d2 and d1<d3 then 0 else
        if d2<d1 and d2<d3 then 1 else 2
    noise totalXY size = do
        let (x, y) = totalXY
        in do x’ ← lap 1.0 x
        y’ ← lap 1.0 y
        size’ ← lap 1.0 size
        return (x’/size’, y’/size’)
    totalCoords pts =
        let ptxs = bmap fst pts
        ptys = bmap snd pts
        in (bsum 1.0 ptxs, bsum 1.0 ptys)
```

```
countPoints pts =  bsum 1.0 (bmap
(\pt → 1) pts)
step c1 c2 c3 pts =
    let [p1, p2, p3] =
        bpartition 3 (assign c1 c2 c3)
        pts p1TotalXY = totalCoords p1
        p1Size = countPoints p1
        p2TotalXY = totalCoords p2
        p2Size = countPoints p2
        p3TotalXY = totalCoords p3
        p3Size = countPoints p3
    in do
        c1’  ← noise p1TotalXY p1Size
        c2’  ← noise p2TotalXY p2Size
        c3’  ← noise p3TotalXY p3Size
        num_iter++
return (c1’, c2’, c3’)
```

Red-zone code always operates on an *individual* element — data from a a single user

E.g., assign function matches closest center to each user

K-Means Example

```
while (num iter < total iters):
    assign c1 c2 c3 pt =
        let d1 = sqdist c1 pt
        d2 = sqdist c2 pt
        d3 = sqdist c3 pt
        in if d1<d2 and d1<d3 then 0 else
        if d2<d1 and d2<d3 then 1 else 2
    noise totalXY size = do
        let (x, y) = totalXY
        in do x’ ← lap 1.0 x
        y’ ← lap 1.0 y
        size’ ← lap 1.0 size
        return (x’/size’, y’/size’)
    totalCoords pts =
        let ptxs = bmap fst pts
        ptys = bmap snd pts
        in (bsum 1.0 ptxs, bsum 1.0 ptys)
```

```
countPoints pts =  bsum 1.0 (bmap
(\pt → 1) pts)
step c1 c2 c3 pts =
    let [p1, p2, p3] =
        bpartition 3 (assign c1 c2 c3)
        pts p1TotalXY = totalCoords p1
        p1Size = countPoints p1
        p2TotalXY = totalCoords p2
        p2Size = countPoints p2
        p3TotalXY = totalCoords p3
        p3Size = countPoints p3
    in do
        c1’  ← noise p1TotalXY p1Size
        c2’  ← noise p2TotalXY p2Size
        c3’  ← noise p3TotalXY p3Size
        num_iter++
return (c1’, c2’, c3’)
```

Data can only pass from red to orange by aggregation (via bsum)

K-Means Example

aggregate variables!

```
while (num_iter < total_iters):
    assign c1 c2 c3 pt =
        let d1 = sqdist c1 pt
        d2 = sqdist c2 pt
        d3 = sqdist c3 pt
        in if d1<d2 and d1<d3 then 0 else
        if d2<d1 and d2<d3 then 1 else 2
    noise totalXY size = do
        let (x, y) = totalXY
        in do x’ ← lap 1.0 x
        y’ ← lap 1.0 y
        size’ ← lap 1.0 size
        return (x’/size’, y’/size’)
    totalCoords pts =
        let ptxs = bmap fst pts
        ptys = bmap snd pts
        in (bsum 1.0 ptxs, bsum 1.0 ptys)
```

```
countPoints pts = bsum 1.0 (bmap
(\pt → 1) pts)
step c1 c2 c3 pts =
    let [p1, p2, p3] =
        bpartition 3 (assign c1 c2 c3)
        pts p1TotalXY = totalCoords p1
        p1Size = countPoints p1
        p2TotalXY = totalCoords p2
        p2Size = countPoints p2
        p3TotalXY = totalCoords p3
        p3Size = countPoints p3
    in do
        c1’  ← noise p1TotalXY p1Size
        c2’  ← noise p2TotalXY p2Size
        c3’  ← noise p3TotalXY p3Size
        num_iter++
return (c1’, c2’, c3’)
```

Data can only pass from red to orange by aggregation (via bsum)

K-Means Example

aggregate variables!

```
while (num_iter < total_iters):
    assign c1 c2 c3 pt =
        let d1 = sqdist c1 pt
        d2 = sqdist c2 pt
        d3 = sqdist c3 pt
        in if d1<d2 and d1<d3 then 0 else
        if d2<d1 and d2<d3 then 1 else 2
    noise totalXY size = do
        let (x, y) = totalXY
        in do x' ← lap 1.0 x
        y' ← lap 1.0 y
        size' ← lap 1.0 size
        return (x'/size', y'/size')
    totalCoords pts =
        let ptxs = bmap fst pts
        ptys = bmap snd pts
        in (bsum 1.0 ptxs, bsum 1.0 ptys)
```

```
countPoints pts = bsum 1.0 (bmap
(\pt → 1) pts)
step c1 c2 c3 pts =
    let [p1, p2, p3] =
        bpartition 3 (assign c1 c2 c3)
        pts p1TotalXY = totalCoords p1
        p1Size = countPoints p1
        p2TotalXY = totalCoords p2
        p2Size = countPoints p2
        p3TotalXY = totalCoords p3
        p3Size = countPoints p3
    in do
        c1'  ← noise p1TotalXY p1Size
        c2'  ← noise p2TotalXY p2Size
        c3'  ← noise p3TotalXY p3Size
        num_iter++
return (c1', c2', c3')
```

Aggregate data can only pass from orange to green by noising

K-Means Example

```
while (num_iter < total_iters):
    assign c1 c2 c3 pt =
        let d1 = sqdist c1 pt
        d2 = sqdist c2 pt
        d3 = sqdist c3 pt
        in if d1<d2 and d1<d3 then 0 else
        if d2<d1 and d2<d3 then 1 else 2
    noise totalXY size = do
        let (x, y) = totalXY
        in do x' ← lap 1.0 x
        y' ← lap 1.0 y
        size' ← lap 1.0 size
        return (x'/size', y'/size')
    totalCoords pts =
        let ptxs = bmap fst pts
        ptys = bmap snd pts
        in (bsum 1.0 ptxs, bsum 1.0 ptys)
```

```
countPoints pts =  bsum 1.0 (bmap
(\pt → 1) pts)
step c1 c2 c3 pts =
    let [p1, p2, p3] =
        bpartition 3 (assign c1 c2 c3)
        pts p1TotalXY = totalCoords p1
        p1Size = countPoints p1
        p2TotalXY = totalCoords p2
        p2Size = countPoints p2
        p3TotalXY = totalCoords p3
        p3Size = countPoints p3
    in do
        c1'  ← noise p1TotalXY p1Size
        c2'  ← noise p2TotalXY p2Size
        c3'  ← noise p3TotalXY p3Size
        num_iter++
return (c1', c2', c3')
```

K-Means Example

Aggregate data can only pass from orange to green by noising

Can broadcast new centers in the clear!

```
while (num_iter < total_iters):
    assign c1 c2 c3 pt =
        let d1 = sqdist c1 pt
        d2 = sqdist c2 pt
        d3 = sqdist c3 pt
        in if d1<d2 and d1<d3 then 0 else
        if d2<d1 and d2<d3 then 1 else 2

    noise totalXY size = do
        let (x, y) = totalXY
        in do x' ← lap 1.0 x
        y' ← lap 1.0 y
        size' ← lap 1.0 size
        return (x'/size', y'/size')

    totalCoords pts =
        let ptxs = bmap fst pts
        ptys = bmap snd pts
        in (bsum 1.0 ptxs, bsum 1.0 ptys)

countPoints pts =  bsum 1.0 (bmap
(\pt → 1) pts)
step c1 c2 c3 pts =
    let [p1, p2, p3] =
        bpartition 3 (assign c1 c2 c3)
        pts p1TotalXY = totalCoords p1
        p1Size = countPoints p1
        p2TotalXY = totalCoords p2
        p2Size = countPoints p2
        p3TotalXY = totalCoords p3
        p3Size = countPoints p3
    in do
        c1'  ← noise p1TotalXY p1Size
        c2'  ← noise p2TotalXY p2Size
        c3'  ← noise p3TotalXY p3Size
        num_iter++
return (c1', c2', c3')
```

Outline (Rest of talk)

- High-Level
  - Scenario
  - Challenges
  - Insight
  - Orchard Workflow
- Orchard Details
  - Threat Model
  - Orchard Map
  - Orchard Walk-Through
  - **Optimizations**
  - Defense against Malicious Users
- Evaluation

Optimizations

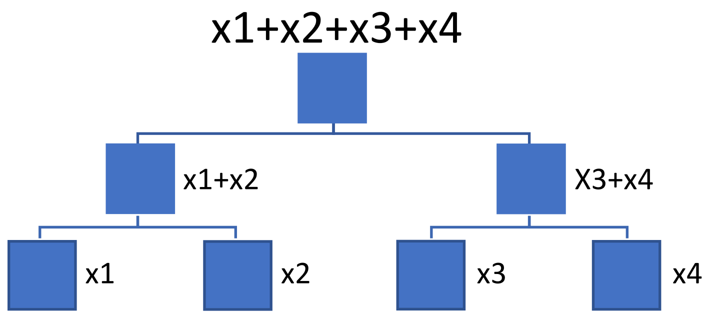
```
noise totalXY size = do
    let (x, y) = totalXY
    in do x’ ← lap 1.0 x
    y’ ← lap 1.0 y
    size’ ← lap 1.0 size
    return (x’/size’, y’/size’)

totalCoords pts =
    let ptxs = bmap fst pts
    ptys = bmap snd pts
    in  bsum 1.0 ptxs, bsum 1.0 ptys)

countPoints pts = bsum 1.0 (bmap (\pt → 1) pts)
```

Optimizations

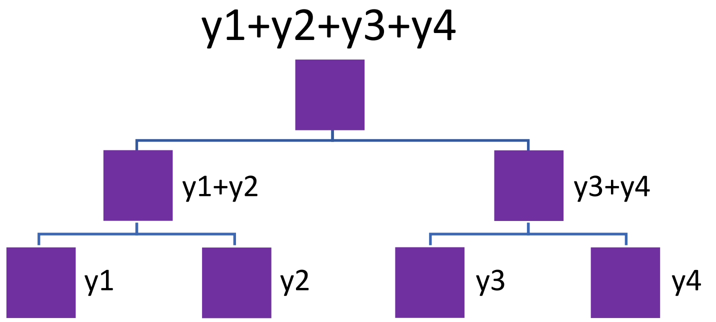
3 variables we need to aggregate and add noise to!

```
noise totalXY size = do
    let (x, y) = totalXY
    in do x' ← lap 1.0 x
    y' ← lap 1.0 y
    size' ← lap 1.0 size
    return (x'/size', y'/size')

totalCoords pts =
    let ptxs = bmap fst pts
    ptys = bmap snd pts
    in bsum 1.0 ptxs, bsum 1.0 ptys)

countPoints pts = bsum 1.0 (bmap (\pt → 1) pts)
```

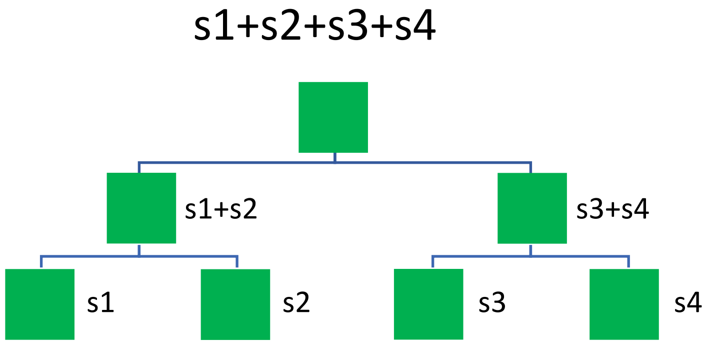
Each one of these bsum operations (naively) requires a complete round of Honeycrisp!



Aggregating 'x' across all users



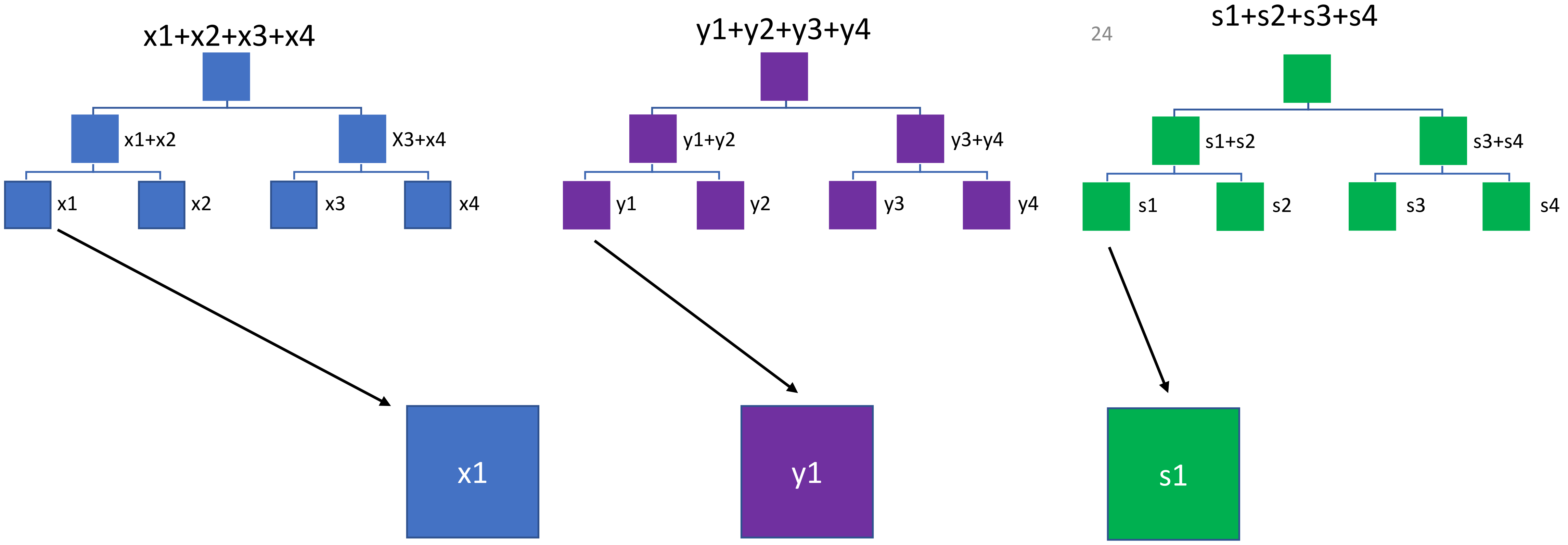
Aggregating 'y' across all users



Aggregating 'size' across all users

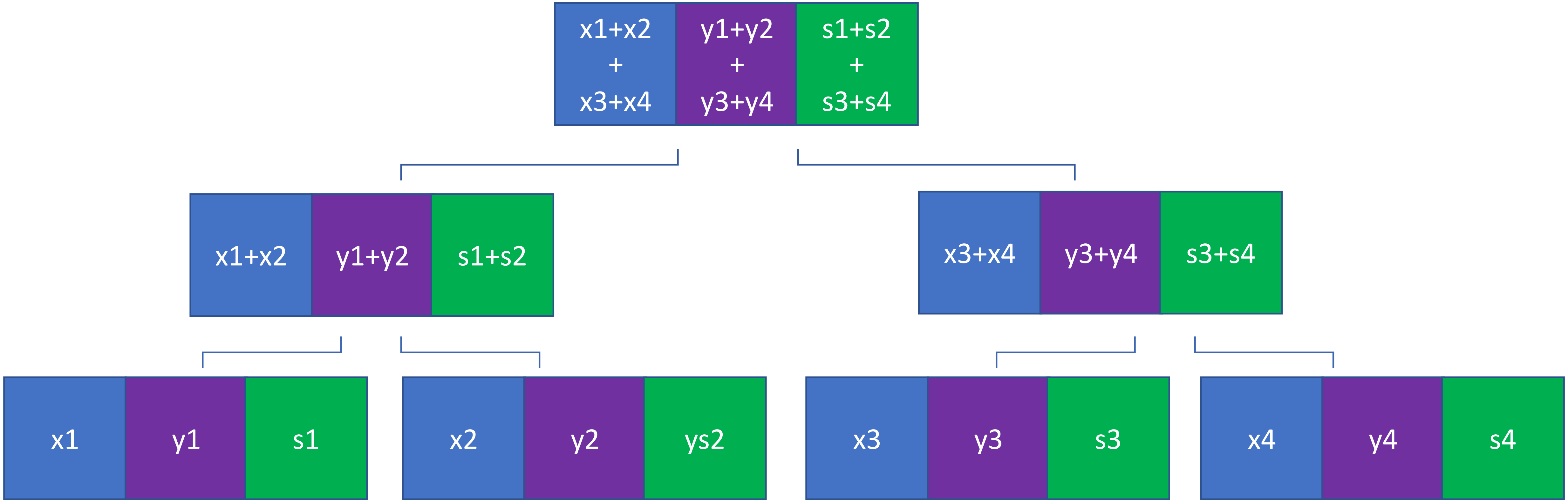
Optimizations

We can pack many ciphertexts into one vector and only aggregate once!



Optimizations

We can pack many ciphertexts into one vector and only aggregate once!



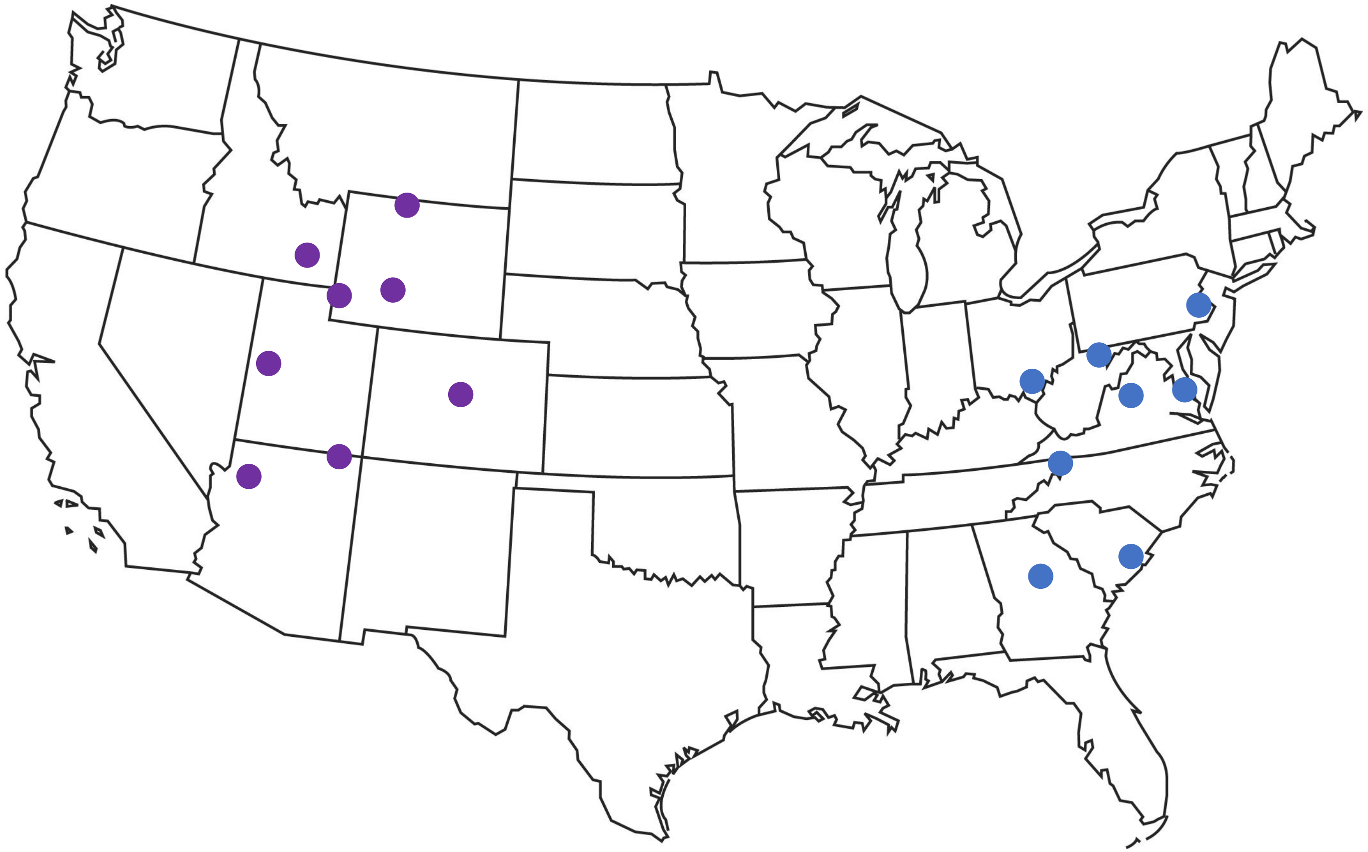
The transformation fuses BMCS calls that do not depend on each other – **full details in paper**!

Robustness

- What to do about malicious users?
  - In Honeycrisp, we already use ZK proofs to ensure proper encryption
  - In Orchard, multiple rounds introduce a new (and powerful!) attack vector
    - Goal: malicious users <u>should not</u> be able to significantly distort the answers
- Example:
  - Submit a (false!) update to shift facility center to malicious target

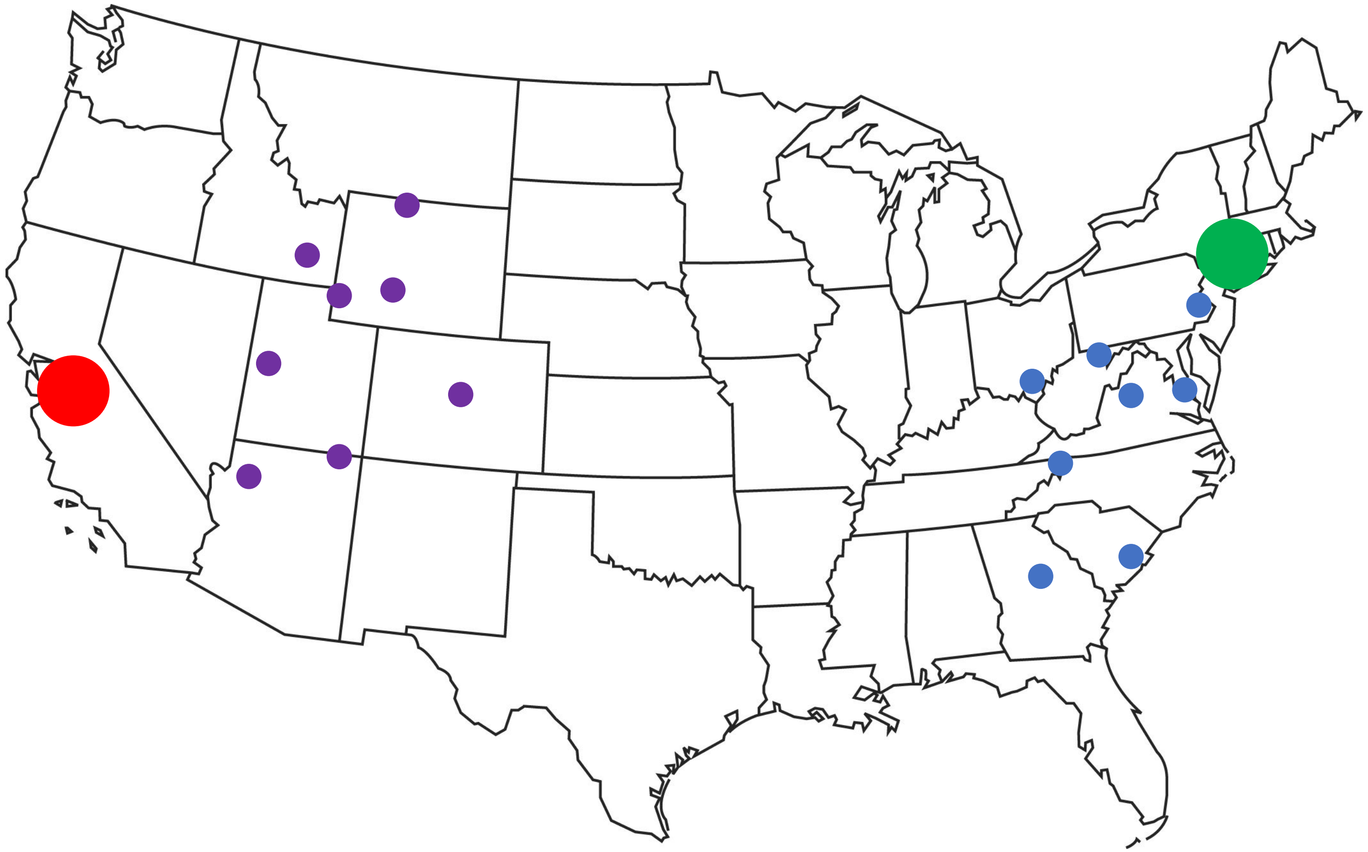
Robustness

3 rounds, 2 clusters



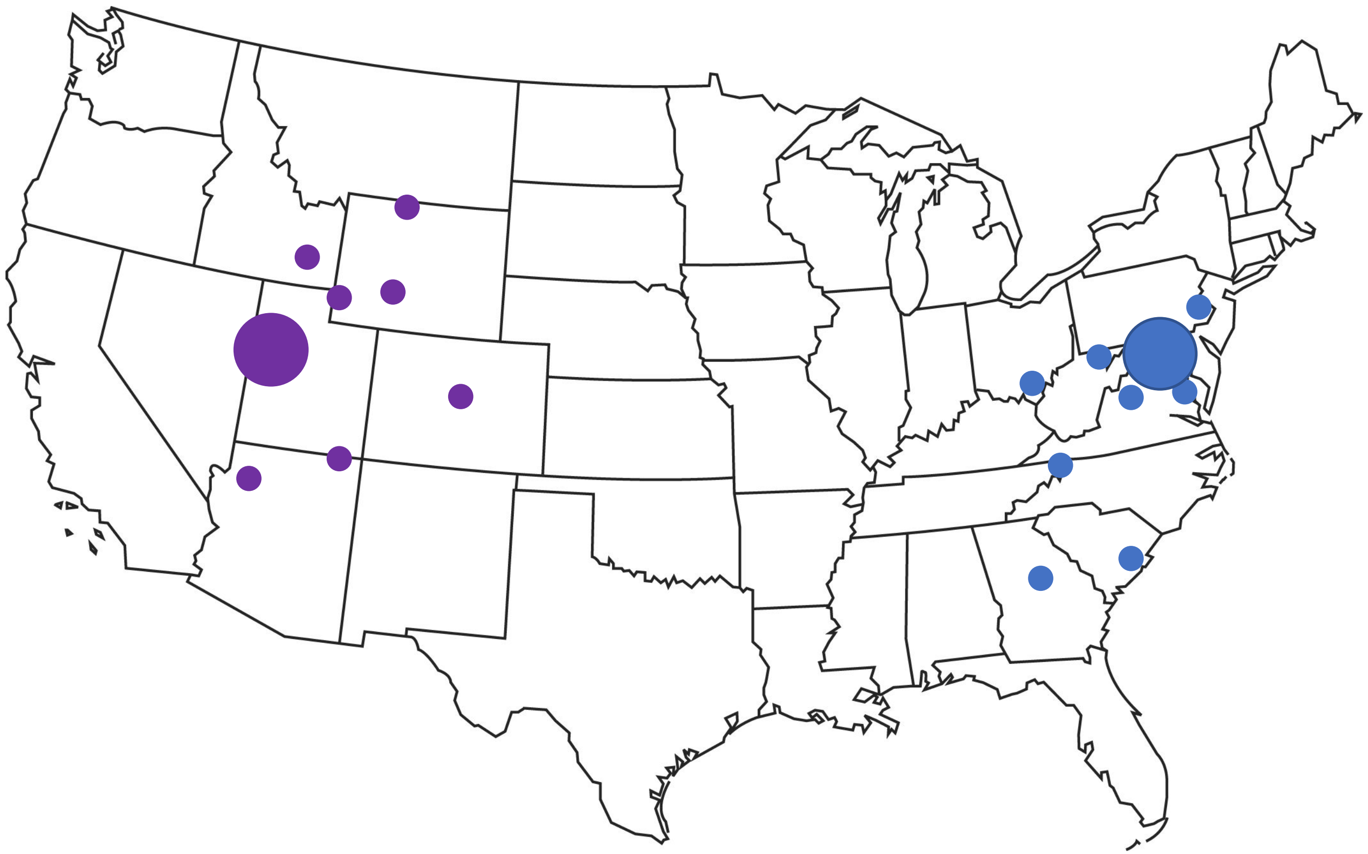
Robustness

Initial (random) clusters



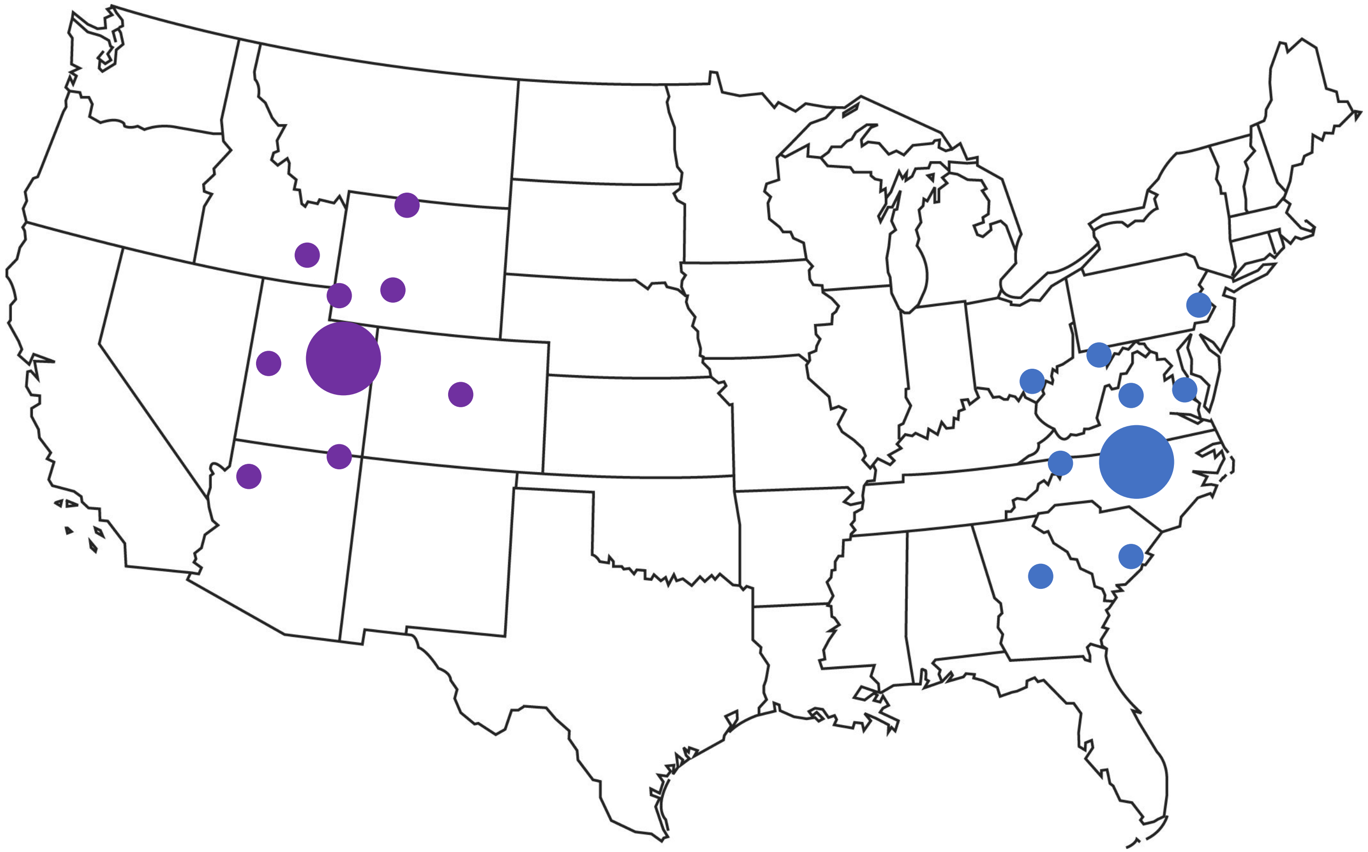
Robustness

Clusters after round 1



Robustness

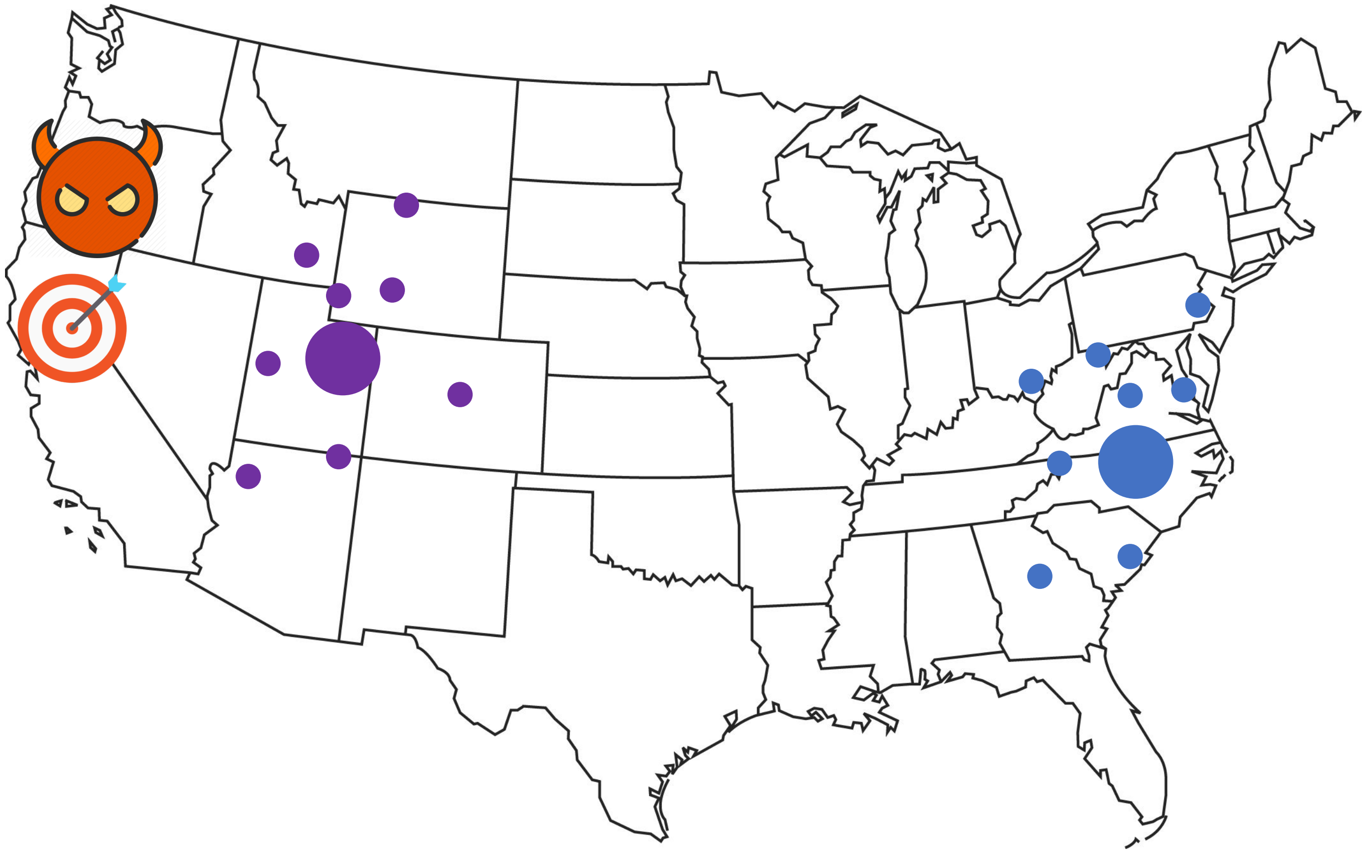
Clusters after round 2 – minor updates



Robustness

Clusters after round 2

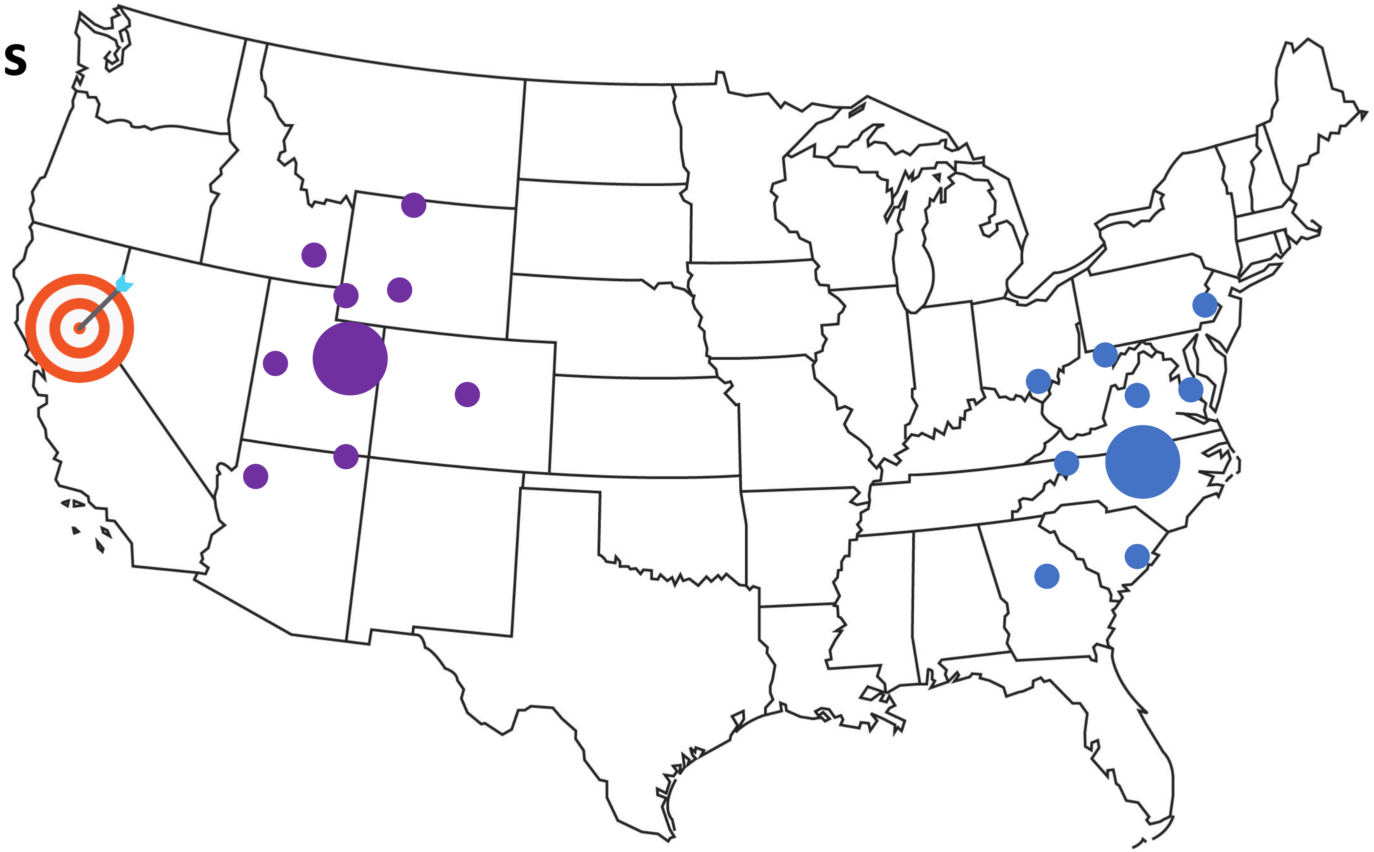
Wants to move cluster to California



Robustness

Positions themselves in different location

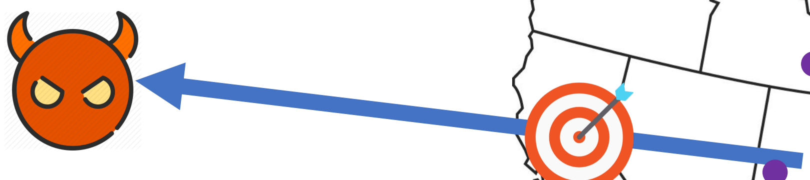
Clusters after round 2

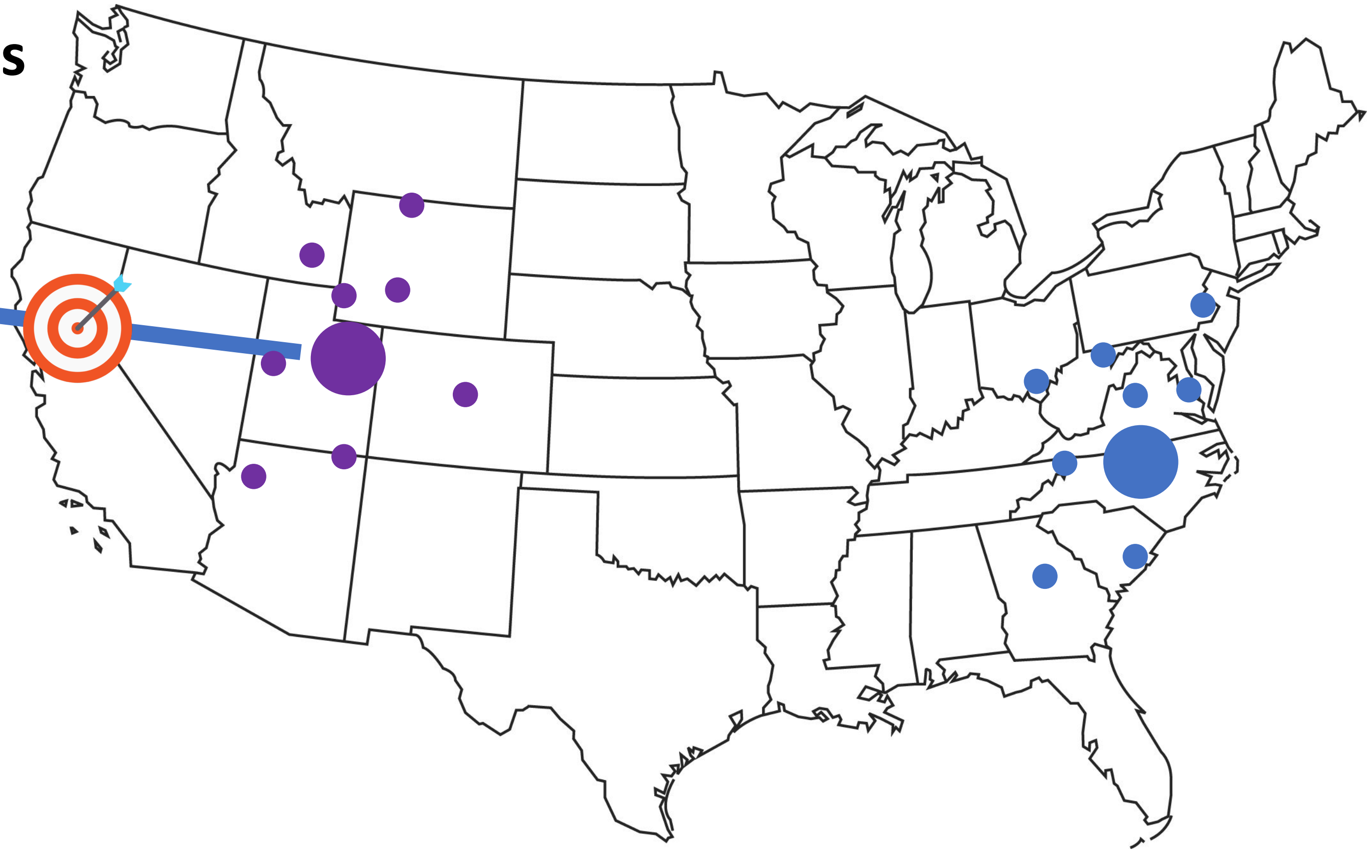


Robustness

Clusters after round 3

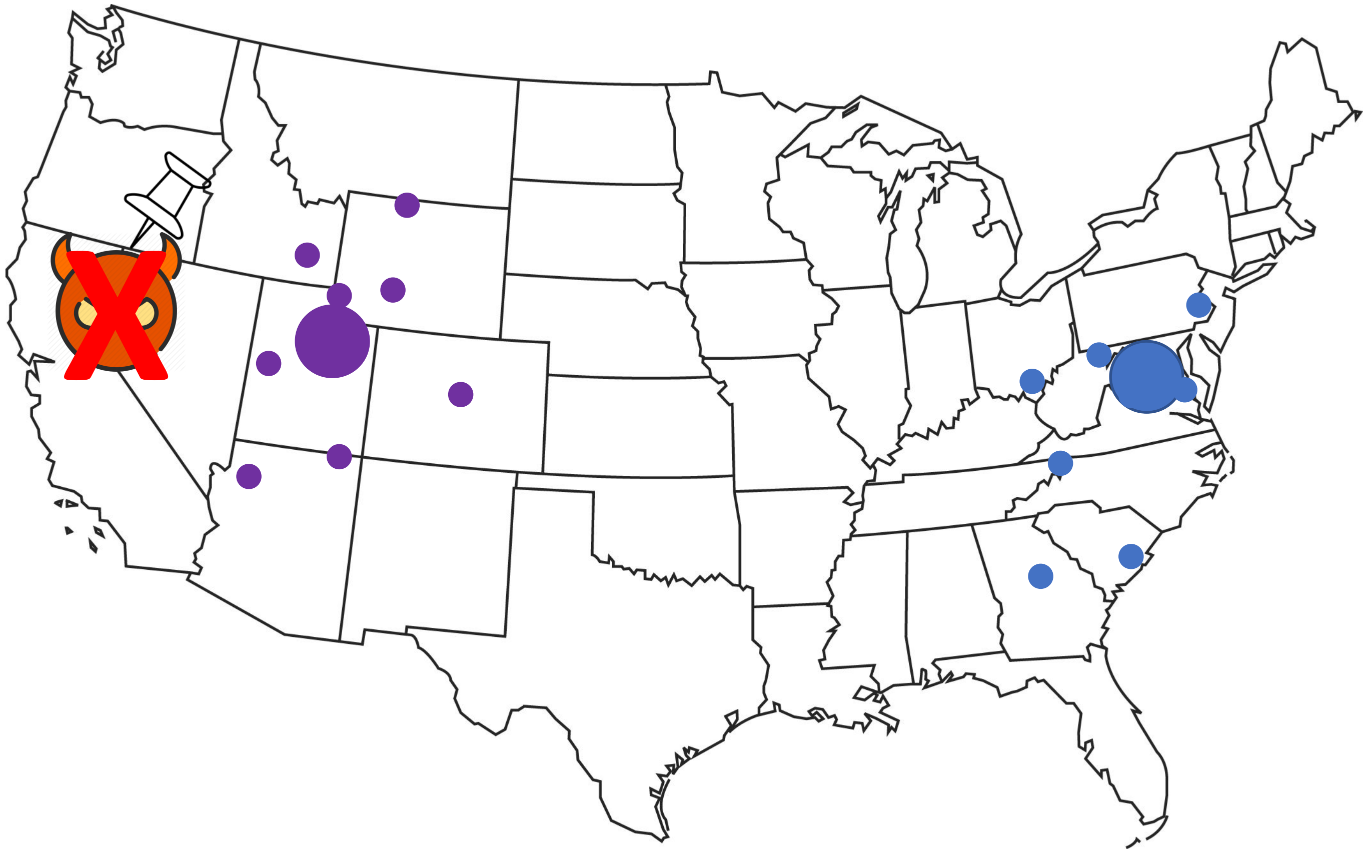
Positions themselves in different location





Robustness

Novel Defense: Use commitments & zero-knowledge proofs to ensure that the data each user uploads is consistent



Evaluation

Questions we wanted to answer in the paper:

- How many private queries can Orchard support?
- How well do Orchard's optimizations work?
- How effective are Orchard's defenses against malicious clients?
- What are the costs of Orchard?

Full results in paper!

Evaluation

Query	Orchard Support
ID3	
K-means	
Perceptron	
PCA	
Logistic Regression	
Naive Bayes	
Neural Network	
Histogram	
K-median	

Query	Orchard Support
CDF	
Range queries	
Bloom filters	
Count Mean Sketch	
Sparse Vector	
DStress	
PATE	
Iterative Database Construction	

17 queries from literature survey

Evaluation

Query	Orchard Support
ID3	X
K-means	X
Perceptron	X
PCA	X
Logistic Regression	X
Naive Bayes	X
Neural Network	X
Histogram	X
K-median	X

Query	Orchard Support
CDF	X
Range queries	X
Bloom filters	X
Count Mean Sketch	✓
Sparse Vector	X
DStress	X
PATE	X
Iterative Database Construction	X

Honeycrisp

Evaluation

Query	Orchard Support
ID3	✓
K-means	✓
Perceptron	✓
PCA	✓
Logistic Regression	✓
Naive Bayes	✓
Neural Network	✓
Histogram	✓
K-median	✓

Query	Orchard Support
CDF	✓
Range queries	✓
Bloom filters	✓
Count Mean Sketch	✓
Sparse Vector	✓
DStress	X
PATE	X
Iterative Database Construction	X

Orchard can answer **14/17** queries we looked at!

Evaluation

Query	# Naïve Rounds	Optimized
ID3		
K-means		
Perceptron		
PCA		
Logistic Regression		
Naive Bayes		
Neural Network		
Histogram		
K-median		

Query	# Naïve Rounds	Optimized
CDF		
Range queries		
Bloom filters		
Count Mean Sketch		
Sparse Vector		

Measuring total number of BMCS calls (with and without our optimizations)

Evaluation

Query	# Naïve Rounds	Optimized
ID3	$2md$	
K-means	$3m$	
Perceptron	$2md$	
PCA	$d^2 + d$	
Logistic Regression	$d + 1$	
Naive Bayes	$2d$	
Neural Network	$2m(d + 1)$	
Histogram	b	
K-median	$3m$	

Query	# Naïve Rounds	Optimized
CDF	b	
Range queries	b	
Bloom filters	d	
Count Mean Sketch	d	
Sparse Vector	1	

Measuring total number of BMCS calls (with and without our optimizations)

Evaluation

Query	# Naïve Rounds	Optimized
ID3	$2md$	$m+1$
K-means	$3m$	$m+1$
Perceptron	$2md$	$m+1$
PCA	d^2+d	1
Logistic Regression	$d+1$	2
Naive Bayes	$2d$	2
Neural Network	$2m(d+1)$	$m+1$
Histogram	b	1
K-median	$3m$	m

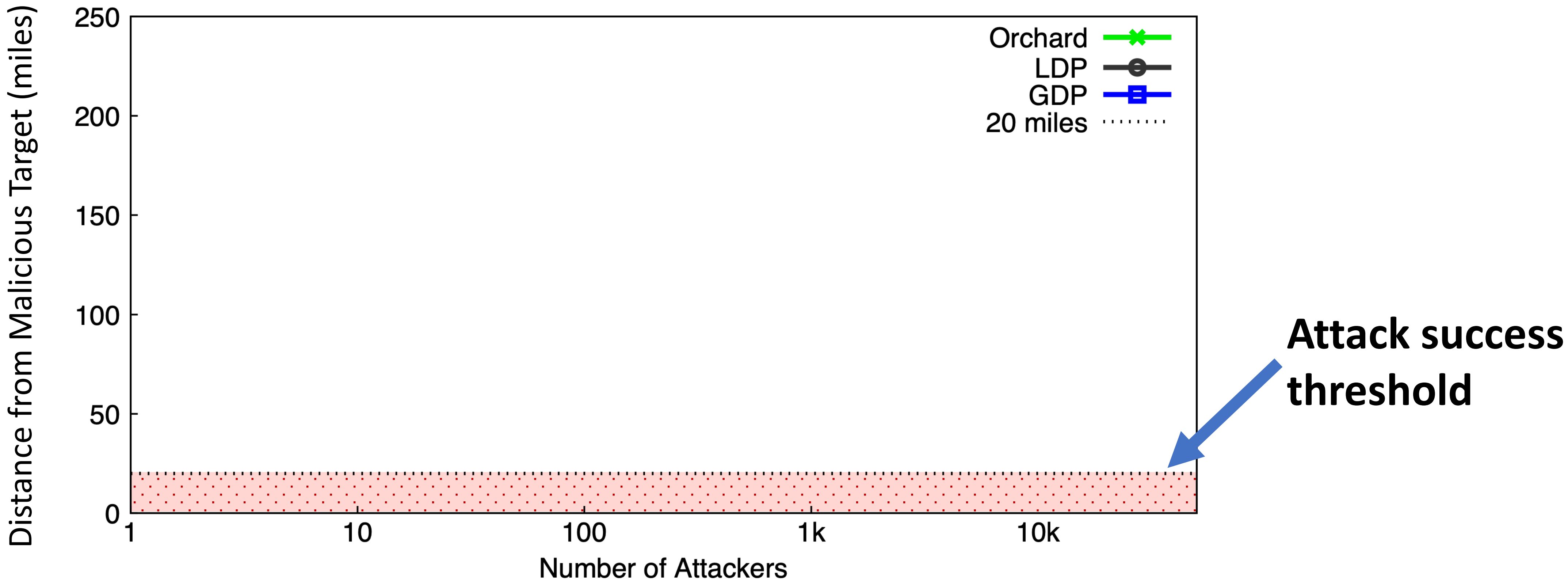
Query	# Naïve Rounds	Optimized
CDF	b	1
Range queries	b	1
Bloom filters	d	1
Count Mean Sketch	d	1
Sparse Vector	1	1

Measuring total number of BMCS calls (with and without our optimizations)

Optimizations save many total rounds!

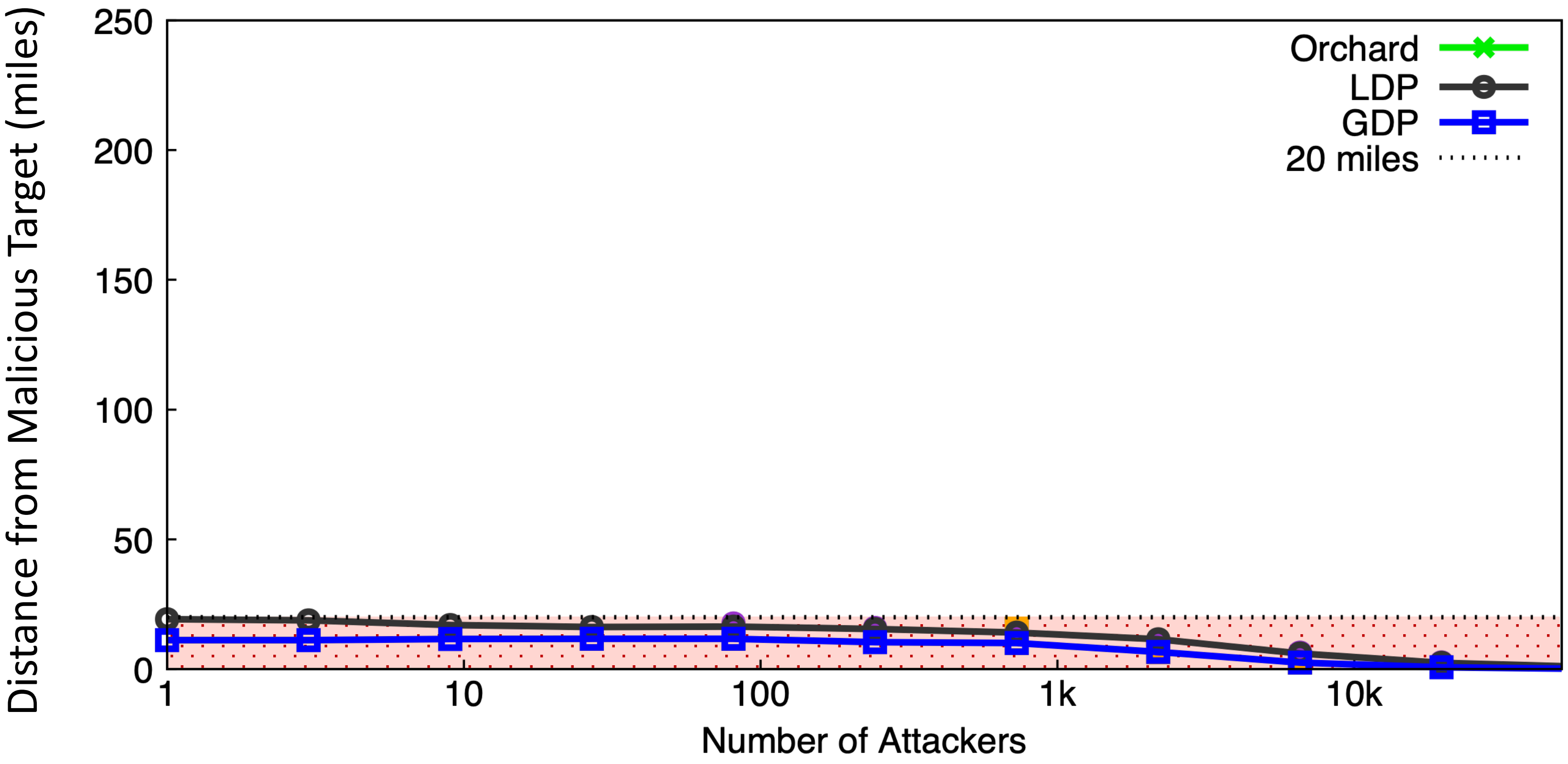
Evaluation - Robustness

Simulation of attack with 10K total users



Evaluation - Robustness

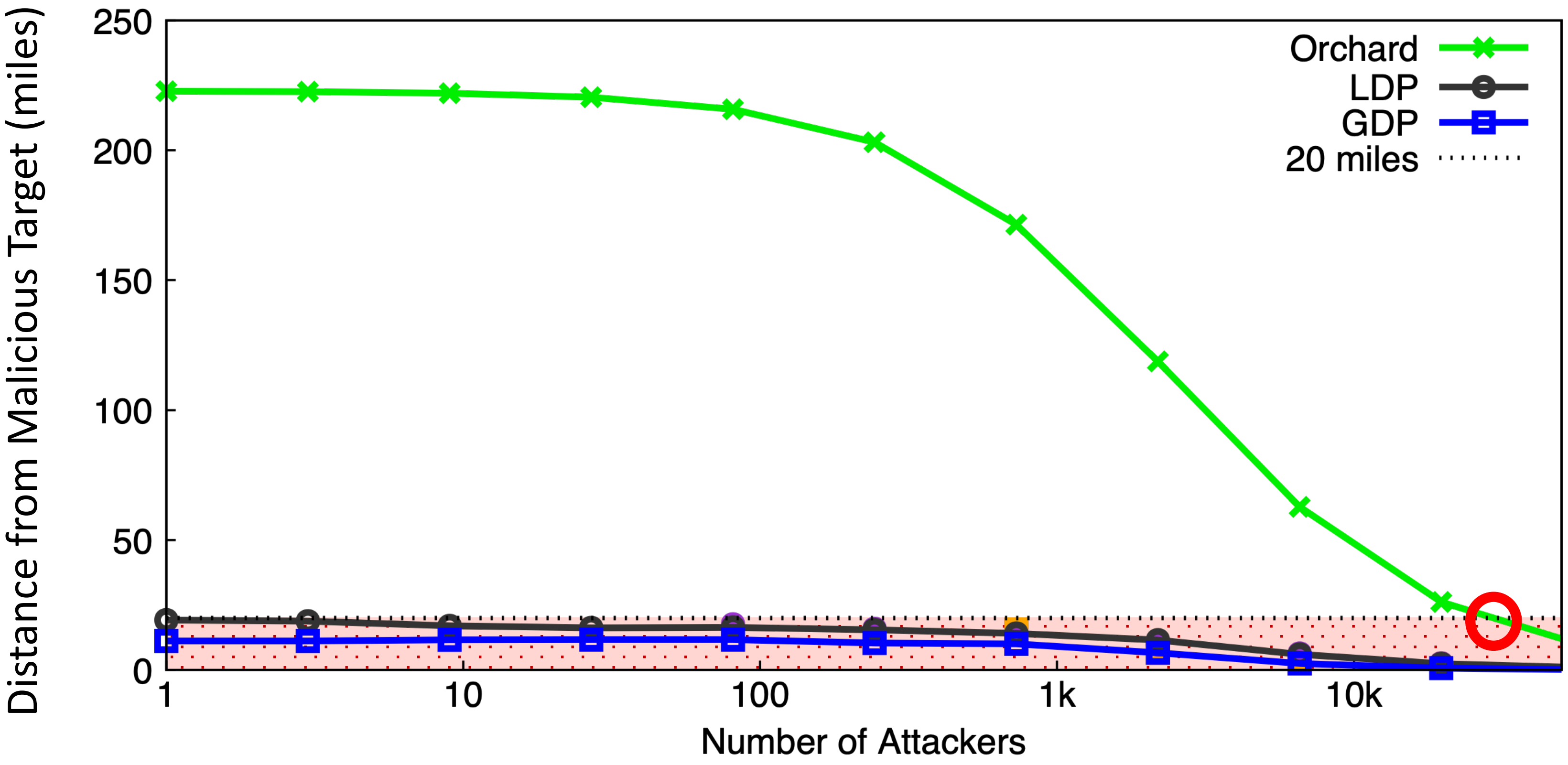
Simulation of attack with 10K total users



- With LDP or GDP, a single 'bad apple' can spoil the whole result

Evaluation - Robustness

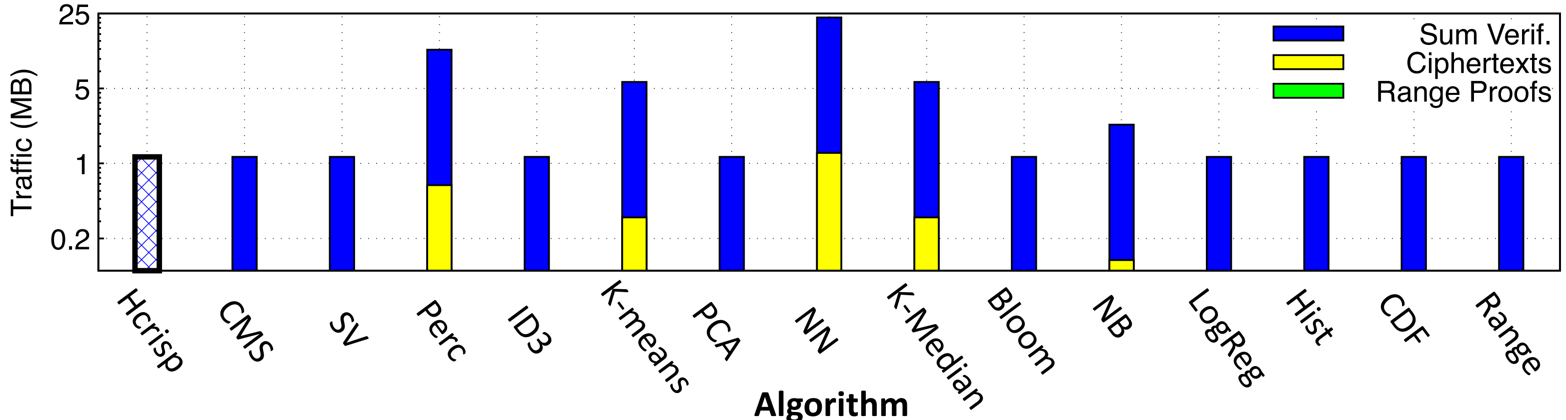
Simulation of attack with 10K total users

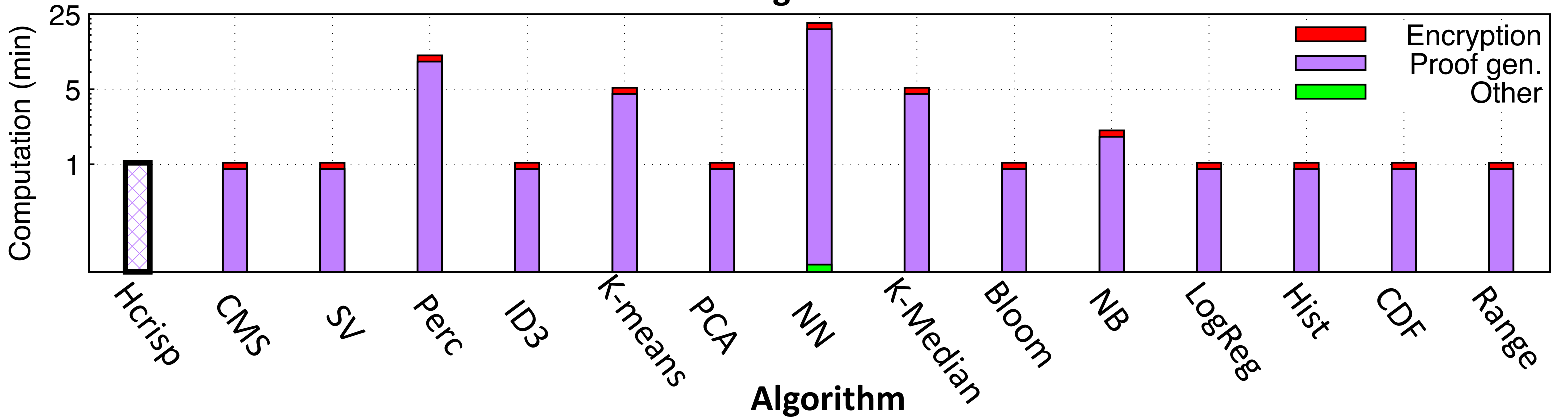


- With LDP or GDP, a single 'bad apple' can spoil the whole result
- With Orchard, the malicious users would have to be in the majority!

Evaluation - Users

Costs for all (non-committee) participants





Cost varies a bit with query, but is generally fairly low

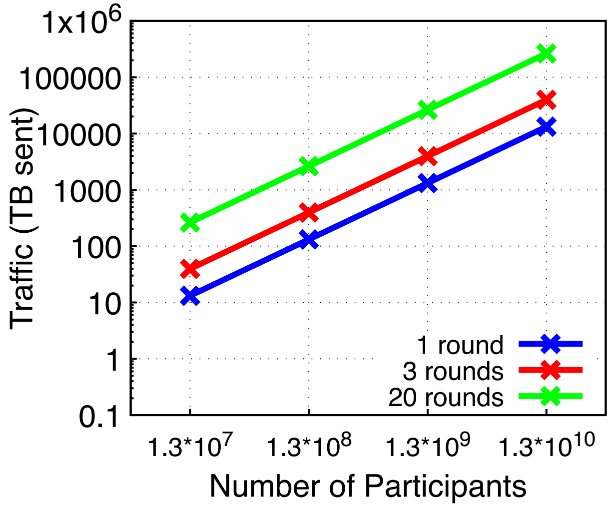
Most users:

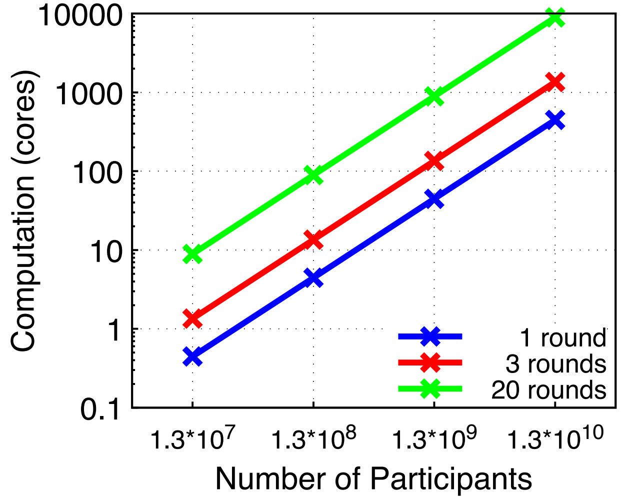
- send less than 25MB of traffic
- spend up to 25 minutes of computation time

If elected to committee, requires substantially more

Most users' costs (>99.99%) are low!

Evaluation - Aggregator





- Both bandwidth and computation scale linearly with number of rounds and participants

- MAX costs:
  - 892 cores, or 74 machines with two CPUs each.
  - 13,180 TB → 10 MB per user (~5 average webpages!)
  - Much of this can be offloaded to CDNs

Absolute costs are within reach of a data center

Summary

- Goal: federated analytics at massive scale, with strong privacy guarantees

- Challenges:
  - Many different queries, no general-purpose solution
  - Small groups of malicious users can manipulate results

- Idea: transform queries to expose internal sums
  - Can be done for most queries we found
  - Enables Honeycrisp-style aggregation (w/ some generalizations)
  - ZKP's can be used to prevent manipulation

- Our solution: Orchard
  - Automatic query transformation, with optimizations
  - Scales almost linearly, to billions of users
  - Good accuracy, even if some users are malicious

Thank You!

Contact: edoroth@seas.upenn.edu