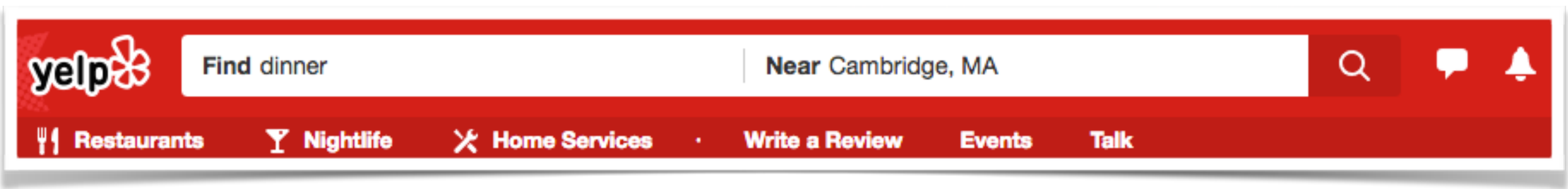
Splinter: Practical Private Queries on Public Data

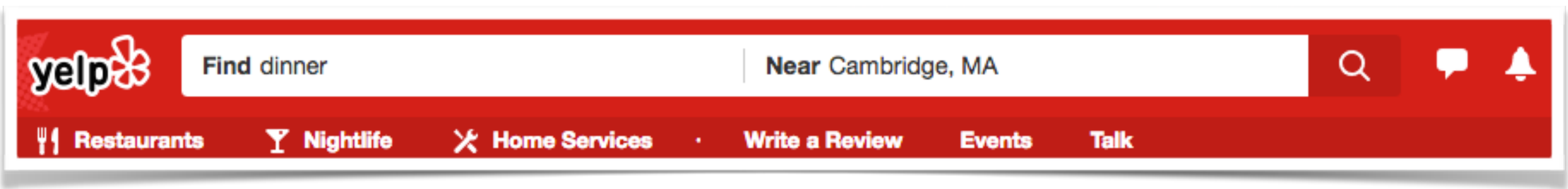
Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan (MIT CSAIL), and Matei Zaharia (Stanford)

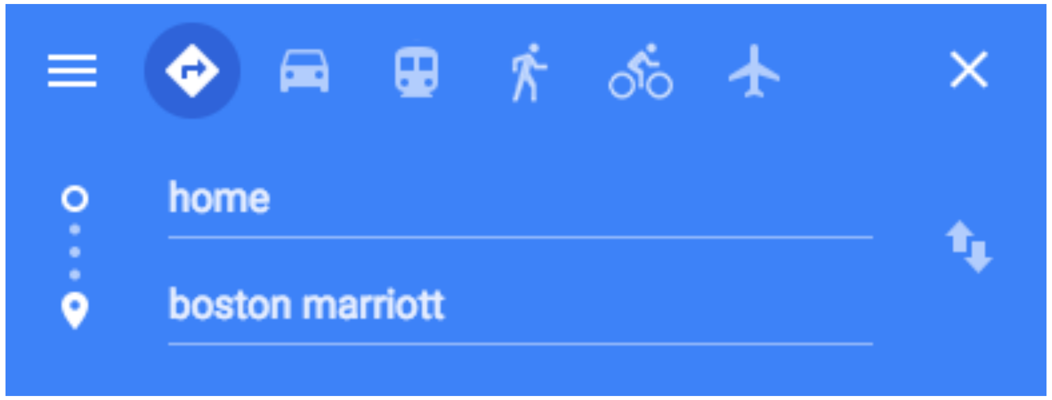
Users regularly perform online searches

Users regularly perform online searches

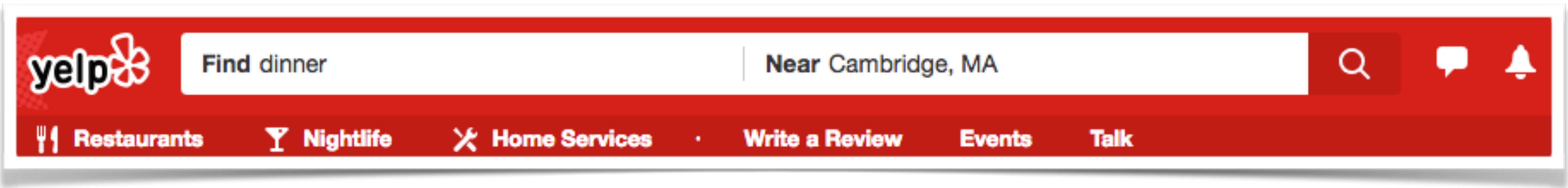


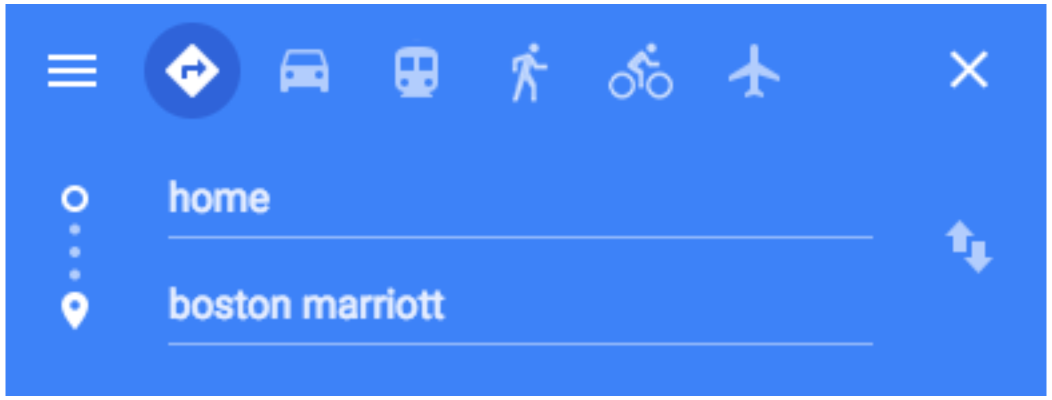
Users regularly perform online searches

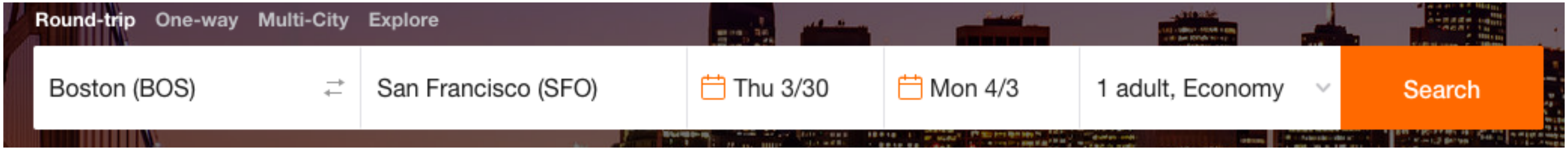




Users regularly perform online searches







These searches reveal sensitive information

These searches reveal sensitive information

Expedia is charging more for flights because you look for them!

by Drew Macomber | Jan 15, 2013 | 12 comments

These searches reveal sensitive information

Expedia is charging more for flights because you look for them!

by Drew Macomber | Jan 15, 2013 | 12 comments

NEWS

The price of free: how Apple, Facebook, Microsoft and Google sell you to advertisers

Here's what popular services like Apple, Google, Facebook, and Microsoft collect — and what you can do about it.

These searches reveal sensitive information

Expedia is charging more for flights because you look for them!

by Drew Macomber | Jan 15, 2013 | 12 comments

NEWS

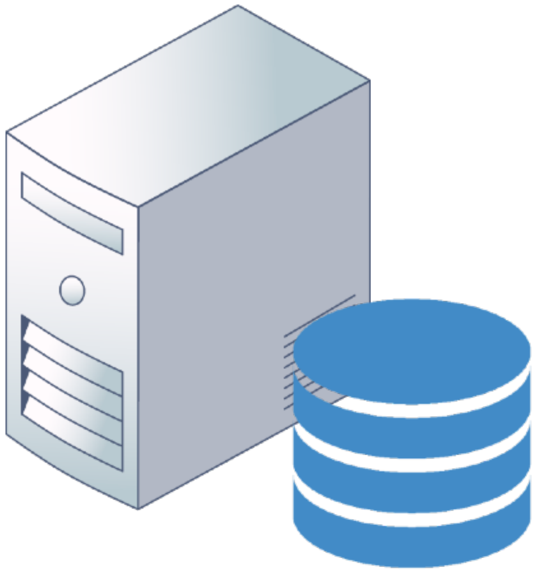
The price of free: how Apple, Facebook, Microsoft and Google sell you to advertisers

Here's what popular services like Apple, Google, Facebook, and Microsoft collect — and what you can do about it.

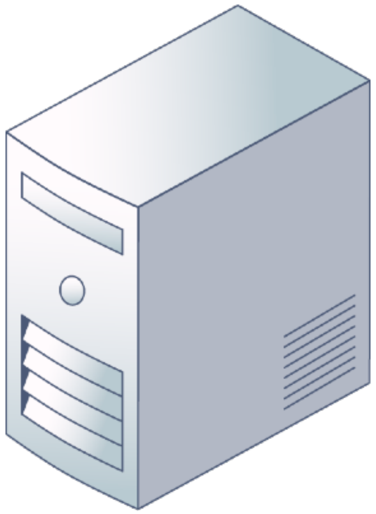
LILY HAY NEWMAN SECURITY 12.31.16 7:00 AM

THE YEAR'S BIGGEST HACKS, FROM YAHOO TO THE DNC

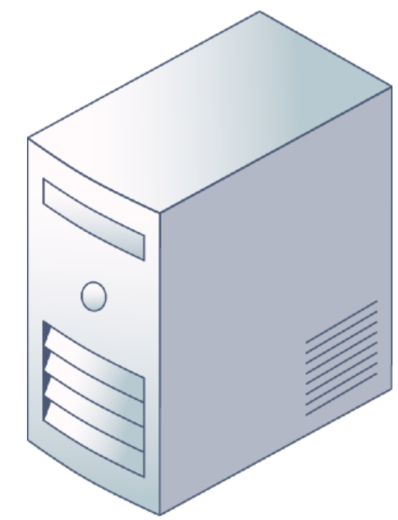
Naive Approach



Naive Approach

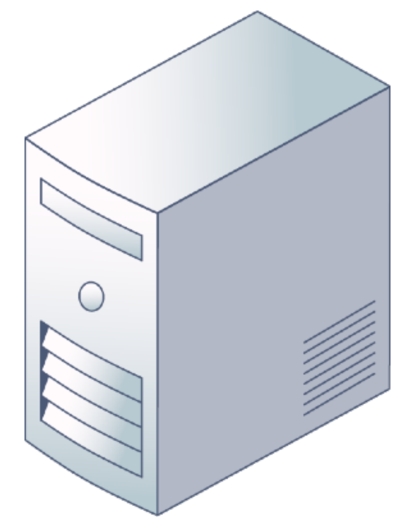


Naive Approach



Problem: Large databases and user has to re-download on updates.

Naive Approach



Problem: Large databases and user has to re-download on updates.

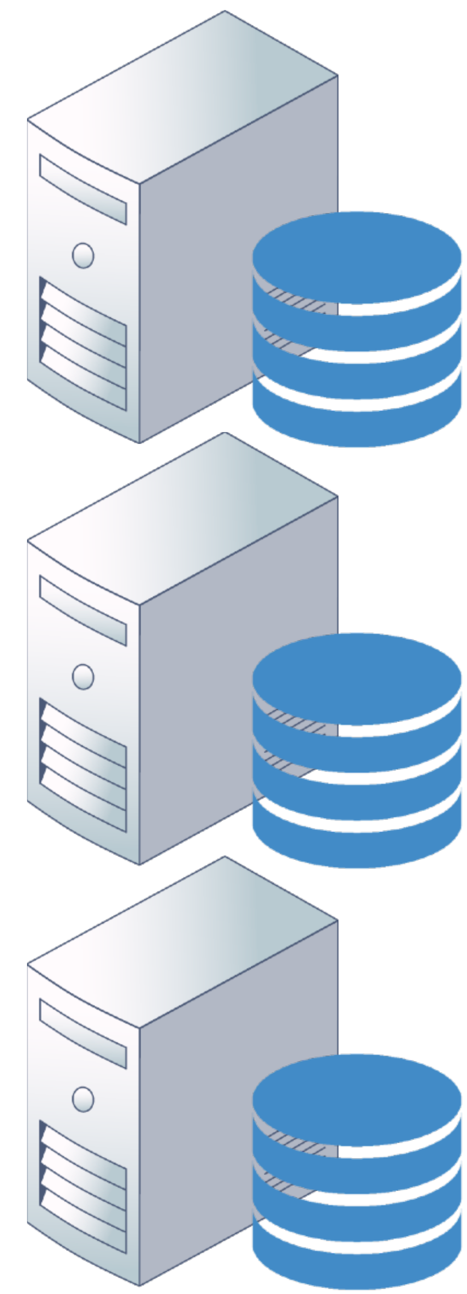
How do we build a practical system that keeps user queries private?

Splinter Design

Splinter Design

User

Providers

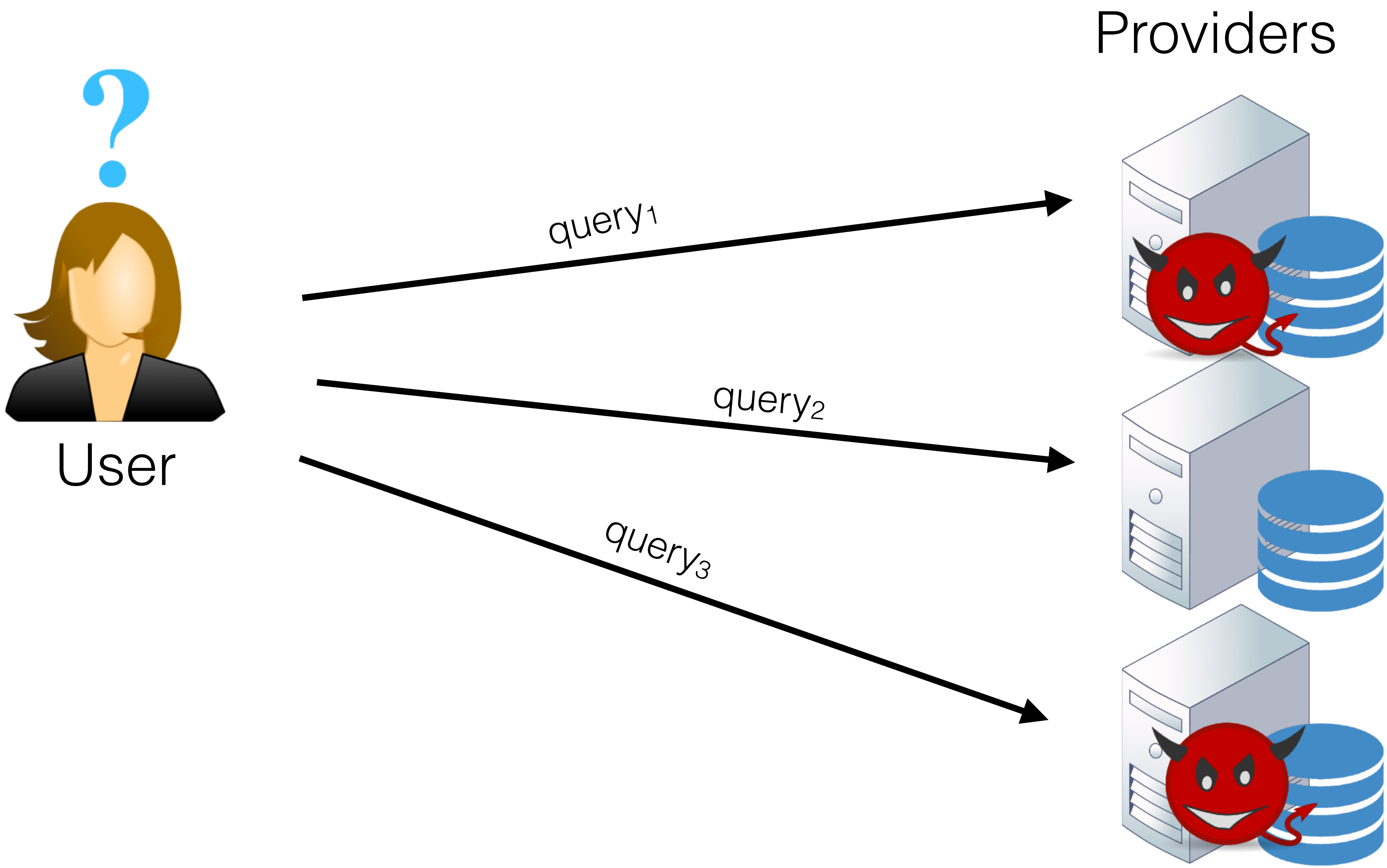


Splinter Design

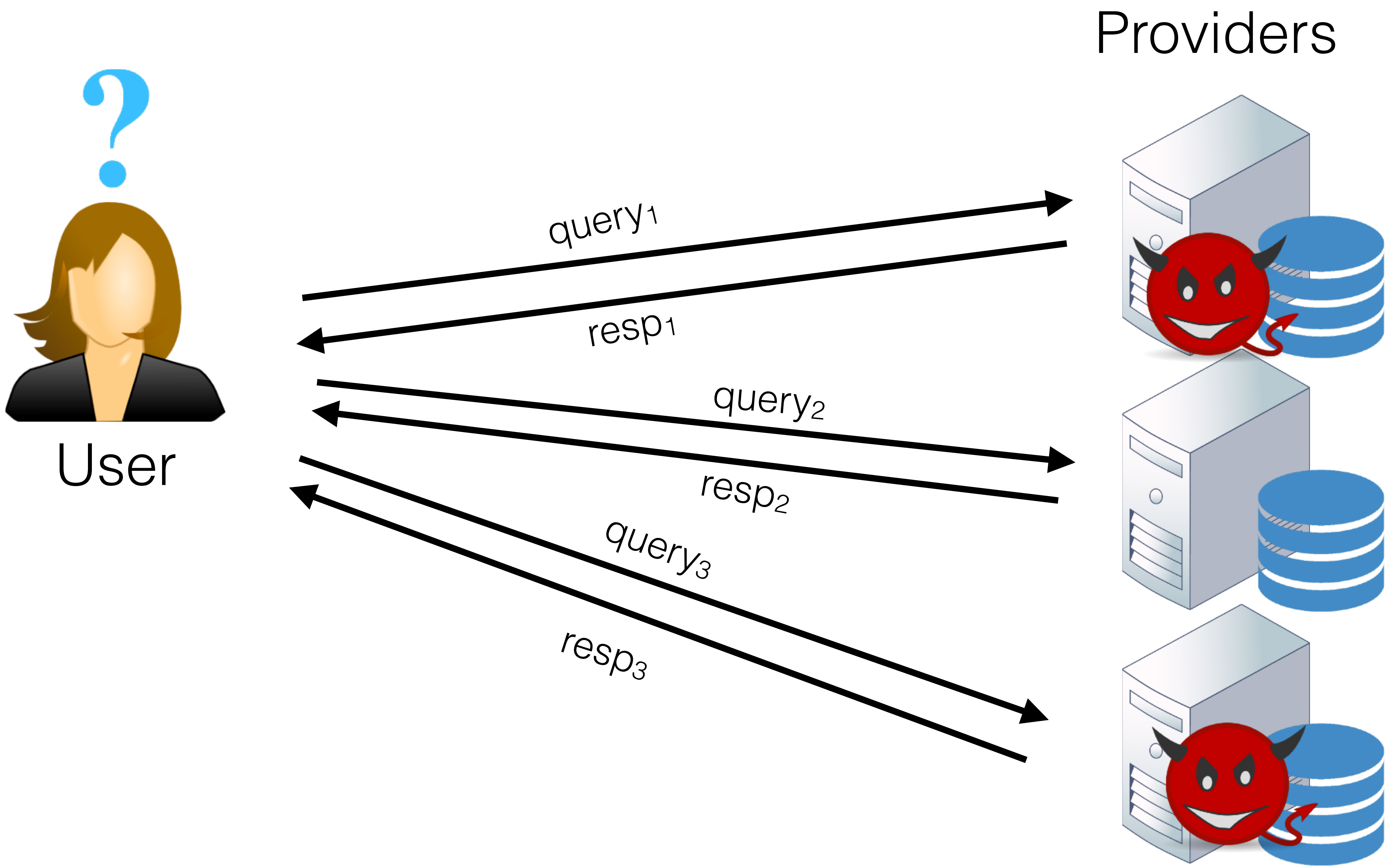
Providers

User

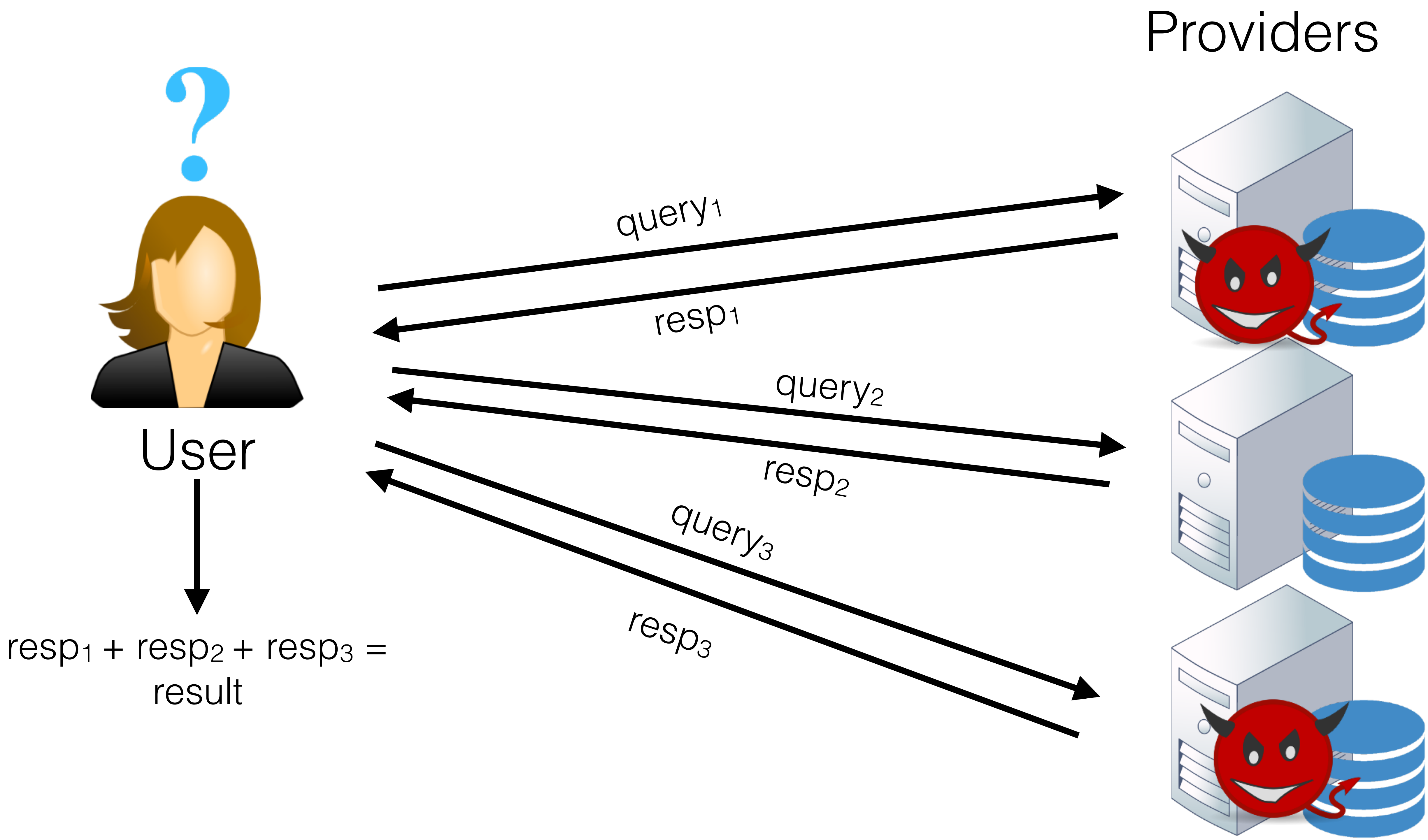
Splinter Design



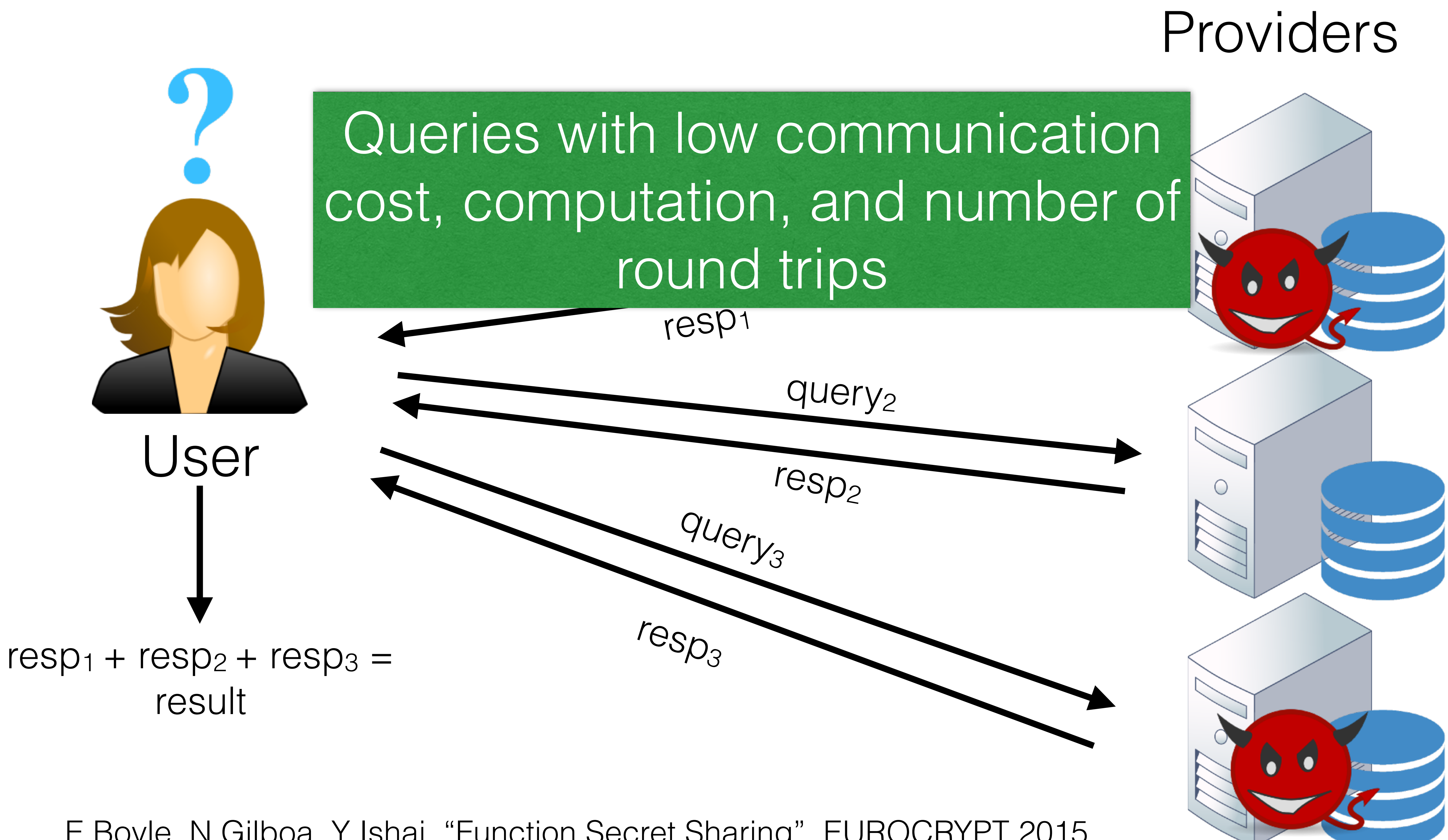
Splinter Design



Splinter Design



Splinter Design



E Boyle, N Gilboa, Y Ishai. “Function Secret Sharing”. EUROCRYPT 2015

Threat Model

- Data on the providers not sensitive and in cleartext
- Providers are passive adversaries
    - Try to learn user's query
    - Cannot tamper with query or database
- At least one provider does not collude with others

Performance

- Response times of **< 1.6 seconds** for databases with millions of records (NYC map, US flights, etc.)
- Up to **10x fewer** round trips than prior systems that use PIR and garbled circuits

Key Contributions

Splinter builds on Function Secret Sharing (FSS) to divide queries into opaque shares

- **New protocols** to run complex queries, such as MAX, TOPK, and disjunctions, over FSS
- **Optimized implementation** of FSS protocol using AES-NI instruction

Outline

- Splinter Queries
- Implementation
- Evaluation

Query Format

- Splinter supports a subset of SQL: projections, limiting filters, aggregates, no joins

```
SELECT aggregate1, aggregate2, … | projections
FROM table
WHERE condition
[GROUP BY expr1, expr2, …]
[ORDER BY expr1, expr2, …]
[LIMIT k]
```

Query Format

- Splinter supports a subset of SQL: projections, limiting filters, aggregates, no joins

```
SELECT aggregate1, aggregate2, … | projections
FROM table
WHERE condition
[GROUP BY expr1, expr2, …]
[ORDER BY expr1, expr2, …]
[LIMIT k]
```

Supported conditions

- Splinter query algorithm for aggregates depends on condition type

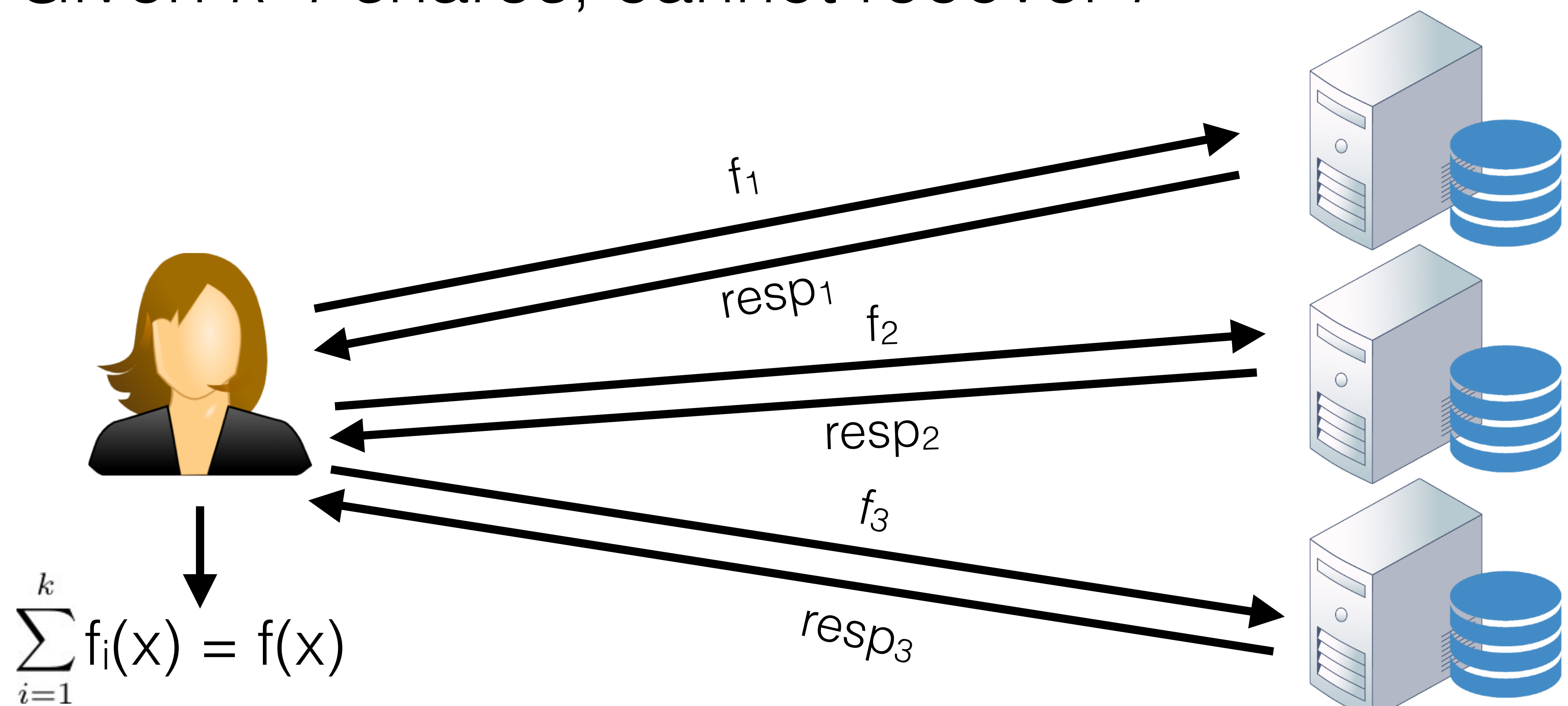
Supported conditions

- Splinter query algorithm for aggregates depends on condition type

Condition	Form
Equality-only	e_1 = $secret_1$ AND … AND e_n = $secret_n$
Intervals	$secret_1 \leq e_1 \leq secret_2$
Disjoint ORs	c_1 OR … OR c_n (c_i can be equality or interval condition)

FSS Properties

- Divides a function f into k shares, f_i, such that:
  - f_i can be evaluated quickly
  - $\sum_{i=1}^{k} f_i(x) = f(x)$
  - Given k-1 shares, cannot recover f



FSS Properties

- Efficient constructions exist for two cases:
  - Point functions: **f(x) = 1** if x = a, **0** otherwise
  - Interval functions: **f(x) = 1** if a ≤ x ≤ b, **0** otherwise

COUNT Query

COUNT Query

route	price
5	8
2	8
5	9
3	4
2	7

COUNT Query

```
SELECT COUNT(*) where route = 5
```

route	price
5	8
2	8
5	9
3	4
2	7

COUNT Query

```
SELECT COUNT(*) where route = ?
```

route	price
5	8
2	8
5	9
3	4
2	7

COUNT Query

```
SELECT COUNT(*) where route = ?
```

f(x) = 1 if x = 5 and **0** otherwise

route	price
5	8
2	8
5	9
3	4
2	7

COUNT Query

```
SELECT COUNT(*) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

route	price
5	8
2	8
5	9
3	4
2	7

COUNT Query

```
SELECT COUNT(*) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

$f_1(x) + f_2(x) = f(x)$

route	price
5	8
2	8
5	9
3	4
2	7

Having either f_1 or f_2 does **not** reveal any information about f

COUNT Query

`SELECT COUNT(*) where route =` $\boxed{?}$

FSS

f(x) = 1 if x = 5 and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

$f_1(x) + f_2(x) = f(x)$

route	**price**	f_1(route)	f_2(route)
5	8	10	-9
2	8	-3	3
5	9	10	-9
3	4	7	-7
2	7	-3	3

Having either f_1 or f_2 does **not** reveal any information about f

COUNT Query

```
SELECT COUNT(*) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

$f_1(x) + f_2(x) = f(x)$

route	**price**	f_1(route)	f_2(route)
5	8	10	-9
2	8	-3	3
5	9	10	-9
3	4	7	-7
2	7	-3	3
		21	-19

Having either f_1 or f_2 does **not** reveal any information about f

COUNT Query

`SELECT COUNT(*) where route =` $\boxed{?}$

FSS

f(x) = 1 if x = 5 and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

$f_1(x) + f_2(x) = f(x)$

route	**price**	f_1(route)	f_2(route)
5	8	10	-9
2	8	-3	3
5	9	10	-9
3	4	7	-7
2	7	-3	3

21 + -19 = 2

Having either f_1 or f_2 does **not** reveal any information about f

COUNT Query

```
SELECT COUNT(*) where route = ?
```

FSS

$\boxed{\mathbf{f(x) = 1} \text{ if } x = 5}$ and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

$\mathbf{f_1(x) + f_2(x) = f(x)}$

route	**price**	f_1(route)	f_2(route)
5	8	10	-9
2	8	-3	3
5	9	10	-9
3	4	7	-7
2	7	-3	3
		21	+ -19 = 2

Having either f_1 or f_2 does **not** reveal any information about f

COUNT Query

```
SELECT COUNT(*) where route = ?
```

FSS

$\boxed{\mathbf{f(x) = 1} \text{ if } x = 5}$ and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

$\mathbf{f_1(x) + f_2(x) = f(x)}$

route	**price**	f_1(route)	f_2(route)	
5	8	10	-9	= 1
2	8	-3	3	
5	9	10	-9	= 1
3	4	7	-7	
2	7	-3	3	
		21	+ -19	= 2

Having either f_1 or f_2 does **not** reveal any information about f

COUNT Query

```
SELECT COUNT(*) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

$\mathbf{f_1(x) + f_2(x) = f(x)}$

route	**price**	f_1(route)	f_2(route)	
5	8	10	-9	
2	8	-3	3	= 0
5	9	10	-9	
3	4	7	-7	= 0
2	7	-3	3	= 0
		21	+ -19	= 2

Having either f_1 or f_2 does **not** reveal any information about f

SUM Query

```
SELECT SUM(price) where route = ?
```

FSS

$\mathbf{f(x) = 1}$ if $x = 5$ and $\mathbf{0}$ otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

route	price
5	8
2	8
5	9
3	4
2	7

SUM Query

```
SELECT SUM(price) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

route	**price**	f_1(route)***price**	f_2(route)***price**
5	8	80	-72
2	8	-24	24
5	9	90	-81
3	4	28	-28
2	7	-21	21

SUM Query

```
SELECT SUM(price) where route = ?
```

FSS

$f(x) = 1$ if $x = 5$ and **0** otherwise

function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

Scale matching records by price

route	**price**	f_1(route)***price**	f_2(route)***price**
5	8	80	-72
2	8	-24	24
5	9	90	-81
3	4	28	-28
2	7	-21	21

SUM Query

```
SELECT SUM(price) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

Scale matching records by price

route	**price**	f_1(route)***price**	f_2(route)***price**
5	8	80	-72
2	8	-24	24
5	9	90	-81
3	4	28	-28
2	7	-21	21
		153	+ -136 = 17

MIN Query for Equality-Only

MIN Query for Equality-Only

route	price
5	8
2	8
5	9
3	4
2	7

MIN Query for Equality-Only

```
SELECT MIN(price) where route = 5
```

route	price
5	8
2	8
5	9
3	4
2	7

MIN Query for Equality-Only

```
SELECT MIN(price) where route = ?
```

route	price
5	8
2	8
5	9
3	4
2	7

MIN Query for Equality-Only

```
SELECT MIN(price) where route = ?
```

FSS

$\mathbf{f(x) = 1}$ if $x = 5$ and $\mathbf{0}$ otherwise

function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

route	price
5	8
2	8
5	9
3	4
2	7

MIN Query for Equality-Only

```
SELECT MIN(price) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

```
SELECT MIN(price)
GROUP BY route
```

Intermediate table

route	price
5	8
2	8
5	9
3	4
2	7

MIN Query for Equality-Only

```
SELECT MIN(price) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

```
SELECT MIN(price)
GROUP BY route
```

route	price
5	8
2	8
5	9
3	4
2	7

→

Intermediate table

route	MIN(price)
5	8
2	7
3	4

MIN Query for Equality-Only

```
SELECT MIN(price) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

```
SELECT MIN(price)
 GROUP BY route
```

route	price
5	8
2	8
5	9
3	4
2	7

Intermediate table

route	MIN(price)	
5	8	$f_i(route_1)$ * $MIN(price)_1$
2	7	$f_i(route_2)$ * $MIN(price)_2$
3	4	$f_i(route_3)$ * $MIN(price)_3$

MIN Query for Equality-Only

```
SELECT MIN(price) where route = ?
```

FSS

f(x) = 1 if x = 5 and **0** otherwise

→ function shares: $\mathbf{f_1}$, $\mathbf{f_2}$

```
SELECT MIN(price)
GROUP BY route
```

route	price
5	8
2	8
5	9
3	4
2	7

Intermediate table

route	MIN(price)
5	8
2	7
3	4

$f_i(\text{route}_1) * \text{MIN(price)}_1$

$+$

$f_i(\text{route}_2) * \text{MIN(price)}_2$

$+$

$f_i(\text{route}_3) * \text{MIN(price)}_3$

MIN Query for Intervals

```
SELECT MIN(price) where 2 ≤ route ≤ 6
```

MIN Query for Intervals

```
SELECT MIN(price) where ? ≤ route ≤ ?
```

MIN Query for Intervals

```
SELECT MIN(price) where [?] ≤ route ≤ [?]
```

1. Each provider computes a sorted table T:

```
SELECT route, price ORDER BY route
```

MIN Query for Intervals

`SELECT MIN(price) where ? ≤ route ≤ ?`

1. Each provider computes a sorted table T:

```
SELECT route, price ORDER BY route
```

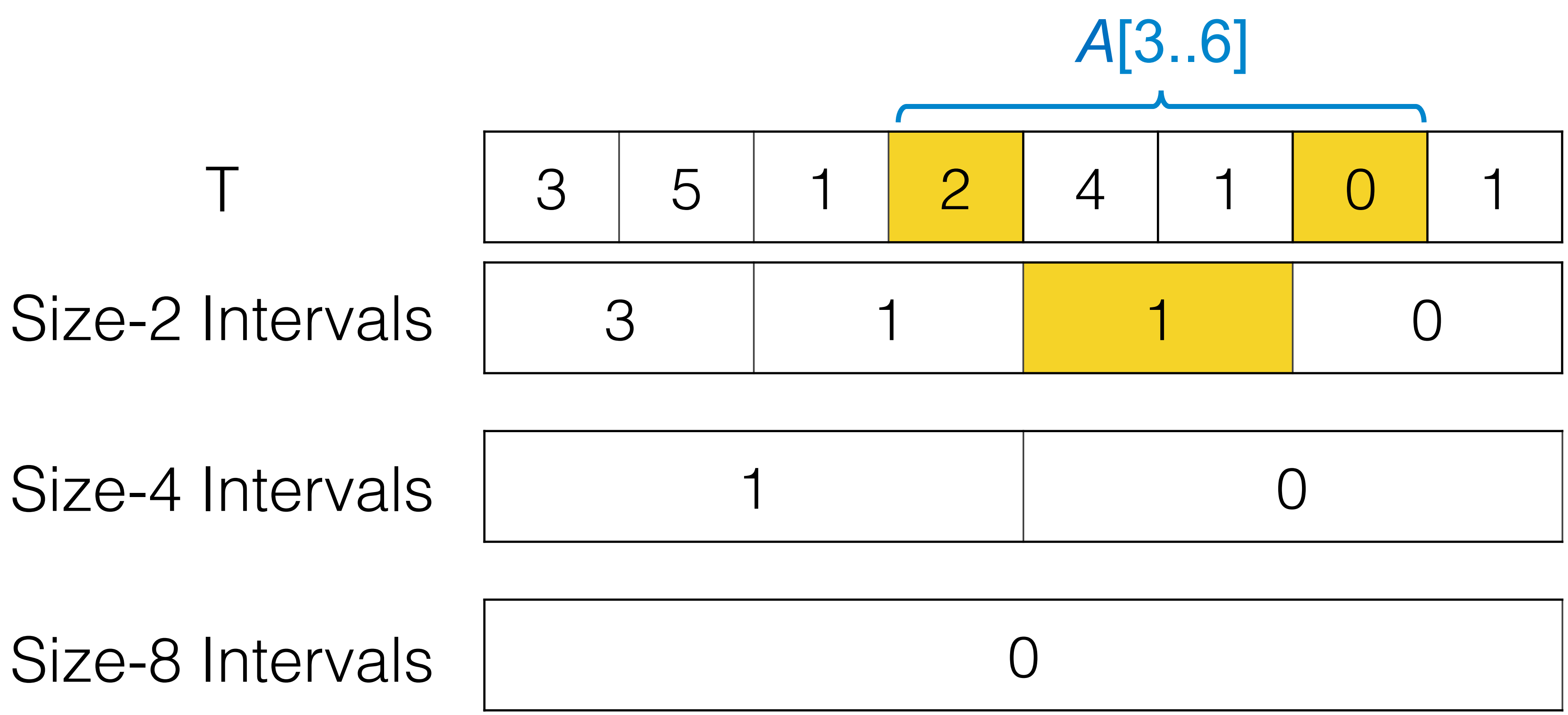
2. Providers find MIN on power-of-2 intervals:

<table>
<tr><td>T</td><td>3</td><td>5</td><td>1</td><td>2</td><td>4</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>Size-2 Intervals</td><td colspan="2">3</td><td colspan="2">1</td><td colspan="2">1</td><td colspan="2">0</td></tr>
<tr><td>Size-4 Intervals</td><td colspan="4">1</td><td colspan="4">0</td></tr>
<tr><td>Size-8 Intervals</td><td colspan="8">0</td></tr>
</table>

MIN Query for Intervals

3. Round 1: Find minimum and maximum indices where $3 \leq$ route ≤ 6. (2 point funcs)

4. Round 2: Select at most 2 intervals of each size to cover target interval (e.g. [3,6]). (log n point funcs)



Other algorithms

Algorithms	Supported queries
FSS	additive aggregates for all conditions (COUNT, SUM, AVG, STDEV, HISTOGRAM)
FSS + intermediate table	MAX, MIN, TOPK for equality-only
FSS + Fenwick tree-like data structure	MAX, MIN, TOPK for intervals
FSS + private binary search	MAX, MIN for disjoint ORs
FSS + private binary search + sampling	TOPK for disjoint ORs

Other algorithms

Algorithms	Supported queries
FSS	additive aggregates for all conditions (COUNT, SUM, AVG, STDEV, HISTOGRAM)
FSS + intermediate table	MAX, MIN, TOPK for equality-only
FSS + Fenwick tree-like data structure	MAX, MIN, TOPK for intervals
FSS + private binary search	MAX, MIN for disjoint ORs
FSS + private binary search + sampling	TOPK for disjoint ORs

Other algorithms

Algorithms	**Supported queries**
FSS	additive aggregates for all conditions (COUNT, SUM, AVG, STDEV, HISTOGRAM)
FSS + intermediate table	MAX, MIN, TOPK for equality-only
FSS + Fenwick tree-like data structure	MAX, MIN, TOPK for intervals
FSS + private binary search	MAX, MIN for disjoint ORs
FSS + private binary search + sampling	TOPK for disjoint ORs

Complexity of Splinter algorithms

Aggregate	Condition	Computation	Round Trips	Bandwidth
Sum-based	any	O(n)	1	O(1)
MAX/MIN	equality-only	O(n)	1	O(1)
MAX/MIN	intervals	O(n log n)	2	O(log n)
MAX/MIN	disjoint ORs	O(n log n)	O(log n)	O(log n)
TOPK	equality-only	O(n)	1	O(1)
TOPK	intervals	O(n log n)	2	O(log n)
TOPK	disjoint ORs	O(n log n)	O(log n)	O(log n)

Complexity of Splinter algorithms

Aggregate	Condition	Computation	Round Trips	Bandwidth
Sum-based	any	O(n)	1	O(1)
MAX/MIN	equality-only	O(n)	1	O(1)
MAX/MIN	intervals	O(n log n)	2	O(log n)
MAX/MIN	disjoint ORs	O(n log n)	O(log n)	O(log n)
TOPK	equality-only	O(n)	1	O(1)
TOPK	intervals	O(n log n)	2	O(log n)
TOPK	disjoint ORs	O(n log n)	O(log n)	O(log n)

Computation time is O(n log n) for all queries and communication costs much smaller than the database

Implementation

- Optimized FSS C++ library: 2000 LoC
- General Query Library: 1500 LoC
- Applications
  - Yelp clone, Flight search, Map routing

https://github.com/frankw2/libfss

Case Studies

Application	# of rows	Size (MB)
Yelp clone	225,000	23
Flight search	6,100,000	225
NYC Map Routing	260,000 nodes 733,000 edges	300

Providers: 64-core x1 Amazon EC2 instance
Client: 2 GHz Intel Core i7 machine
Network latency: 14 ms

Case Studies

Application	# of rows	Size (MB)
Yelp clone	225,000	23
Flight search	6,100,000	225
NYC Map Routing	260,000 nodes 733,000 edges	300

All case studies based on real datasets

Providers: 64-core x1 Amazon EC2 instance
Client: 2 GHz Intel Core i7 machine
Network latency: 14 ms

Case Study Queries

Application	Query

Case Study Queries

Application	Query
Yelp clone	• `SELECT COUNT(*) WHERE category="Thai"` • `SELECT TOP 10 restaurant WHERE category="Mexican" AND hex2mi in (1, 2, 3) ORDER BY stars` • `SELECT restaurant, MAX(stars) WHERE category in ("Mexican", "Chinese", "Indian", "Greek", "Thai", "Japanese") GROUP BY category`

Case Study Queries

Application	Query
Yelp clone	• `SELECT COUNT(*) WHERE category="Thai"` • `SELECT TOP 10 restaurant WHERE category="Mexican" AND hex2mi in (1, 2, 3) ORDER BY stars` • `SELECT restaurant, MAX(stars) WHERE category in ("Mexican", "Chinese", "Indian", "Greek", "Thai", "Japanese") GROUP BY category`
Flight search	• `SELECT AVG(price) WHERE month=3 AND route = 5` • `SELECT TOP 10 flight_no WHERE route = 5 ORDER BY price`

Case Study Queries

Application	Query
Yelp clone	• `SELECT COUNT(*) WHERE category="Thai"` • `SELECT TOP 10 restaurant WHERE category="Mexican" AND hex2mi in (1, 2, 3) ORDER BY stars` • `SELECT restaurant, MAX(stars) WHERE category in ("Mexican", "Chinese", "Indian", "Greek", "Thai", "Japanese") GROUP BY category`
Flight search	• `SELECT AVG(price) WHERE month=3 AND route = 5` • `SELECT TOP 10 flight_no WHERE route = 5 ORDER BY price`
Map routing	• `SELECT grid_nodes WHERE grid_no = 5` • `SELECT path WHERE src = 4 and dst = 10`

Performance

Query	Dataset	Providers	Round Trips	Communication	Response Time
Count of Thai Restaurants	Yelp	2 3	1	3 KB 30 KB	57 ms 52 ms
Top 10 Mexican restaurants	Yelp	2 3	1	24 KB 2 MB	150 ms 542 ms
Best rated restaurant in category subset	Yelp	2 3	11	245 KB 1.2 MB	1.3 s 1.6 s
AVG monthly price	Flights	2 3	1	9 KB 450 KB	1.0 s 1.2 s
Top 10 cheapest flights	Flights	2 3	1	4 KB 20 KB	30 ms 39 ms
NYC Routing	Maps	2 3	2	45 KB 725 KB	1.2 s 1.0 s

Splinter has lower response times and fewer rounds trips compared to Olumofin et al.

System	Round Trips	Response Times
Olumofin et al.	log n (all queries)	2-18 seconds
Splinter	constant (most queries) log n (select queries)	50 ms - 1.6 seconds

Other related work:

- PIR systems (Readon et al., Popcorn)
- Garbled circuits (Wu et al., Embark)

Conclusion

- Splinter is the first practical system that protects users' queries on real datasets
- We develop new protocols to execute complex queries over FSS and have fewer round trips and lower response times than prior systems