Oblivious Bucketization

Microsoft

ARA-aggr

"Attribution Reporting API with Aggregate **Reports**" **as a starting point**

Reports consist of

- A key k: string of bits

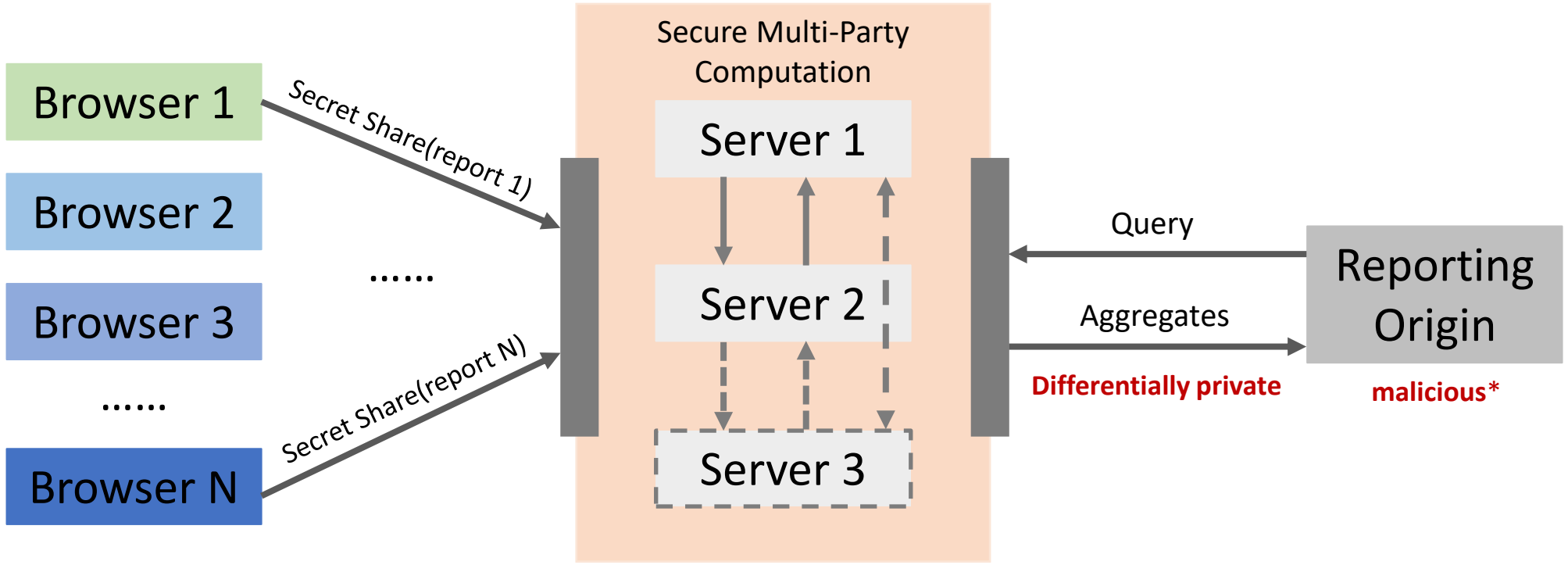
16-bit key
(AdCategory||Region)

1011101111||000101

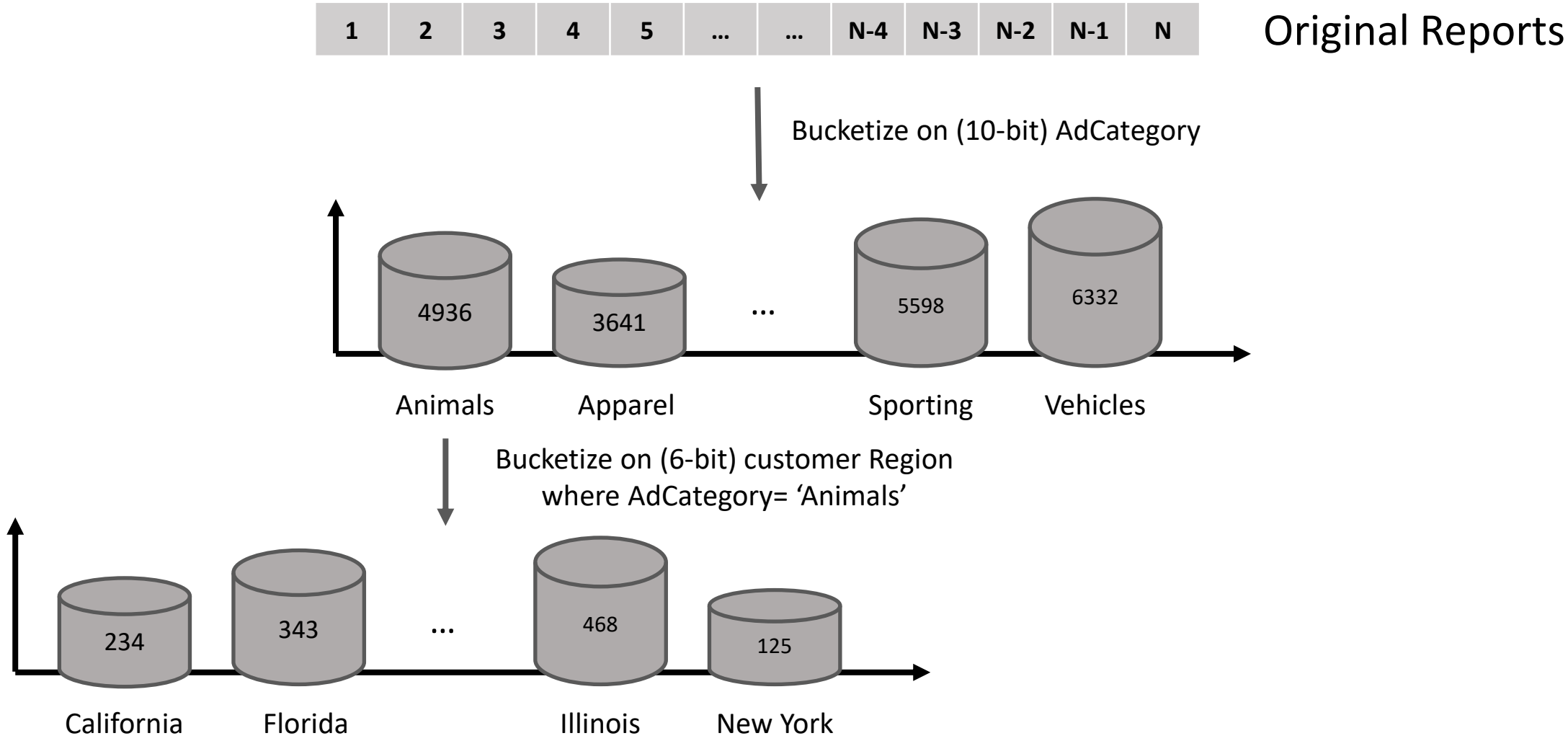
- A list of values $v_1, \ldots, v_m$: these are values one might want to aggregate

Each report (k, $v_1, \ldots, v_m$) will be secret shared with one share given to each of **2 aggregation servers**

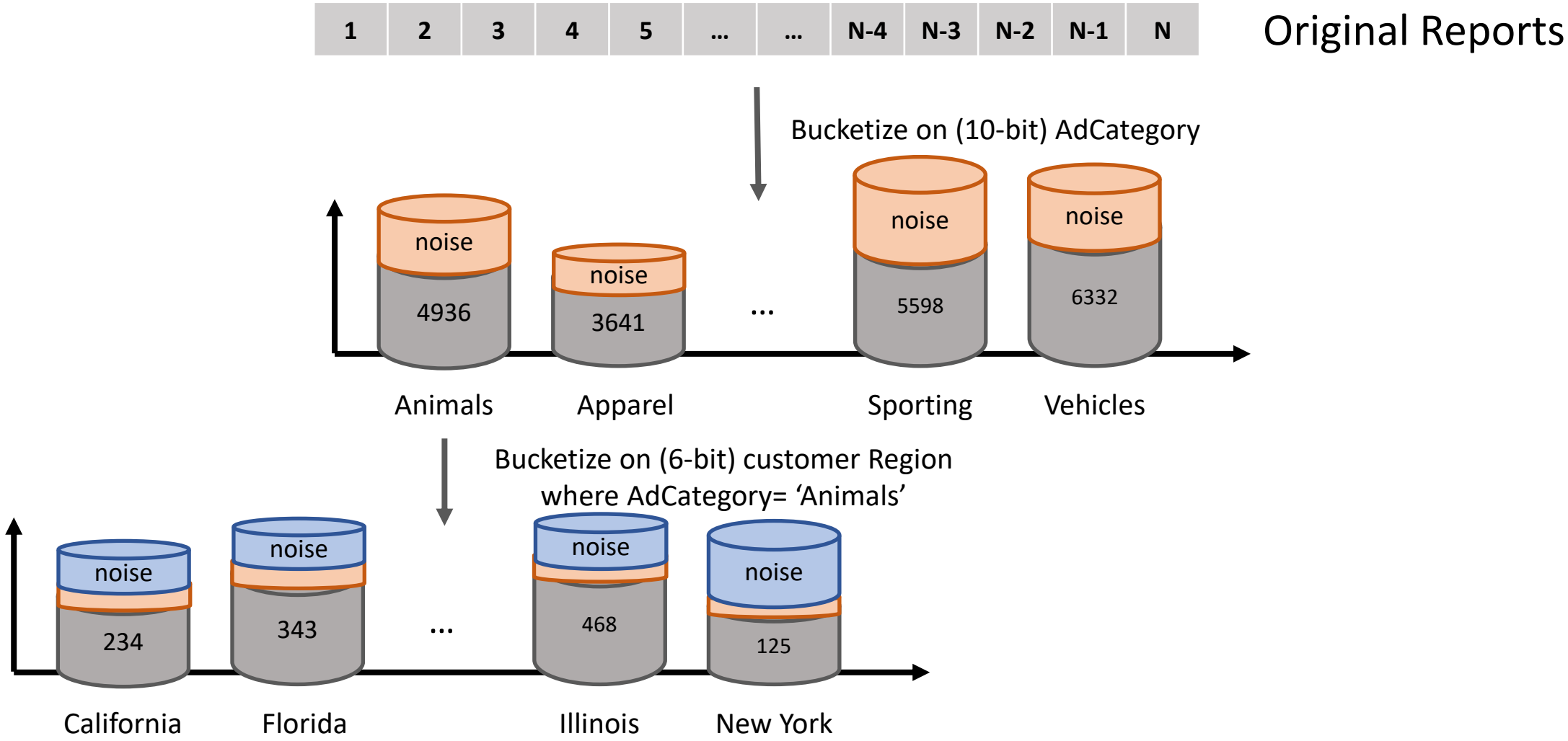
System Design



Flexible Query Structure



Differentially Private Aggregates



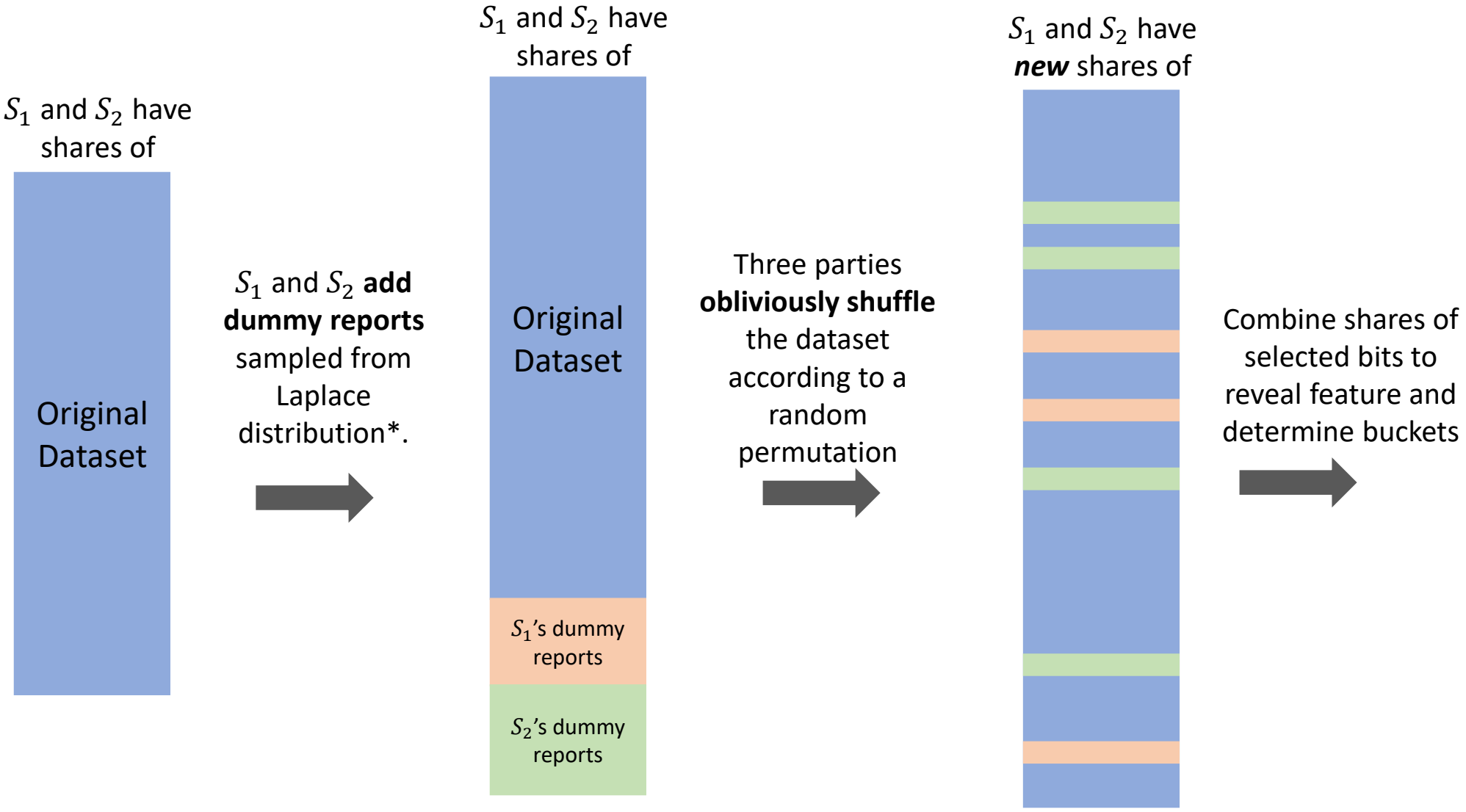
Oblivious Bucketization

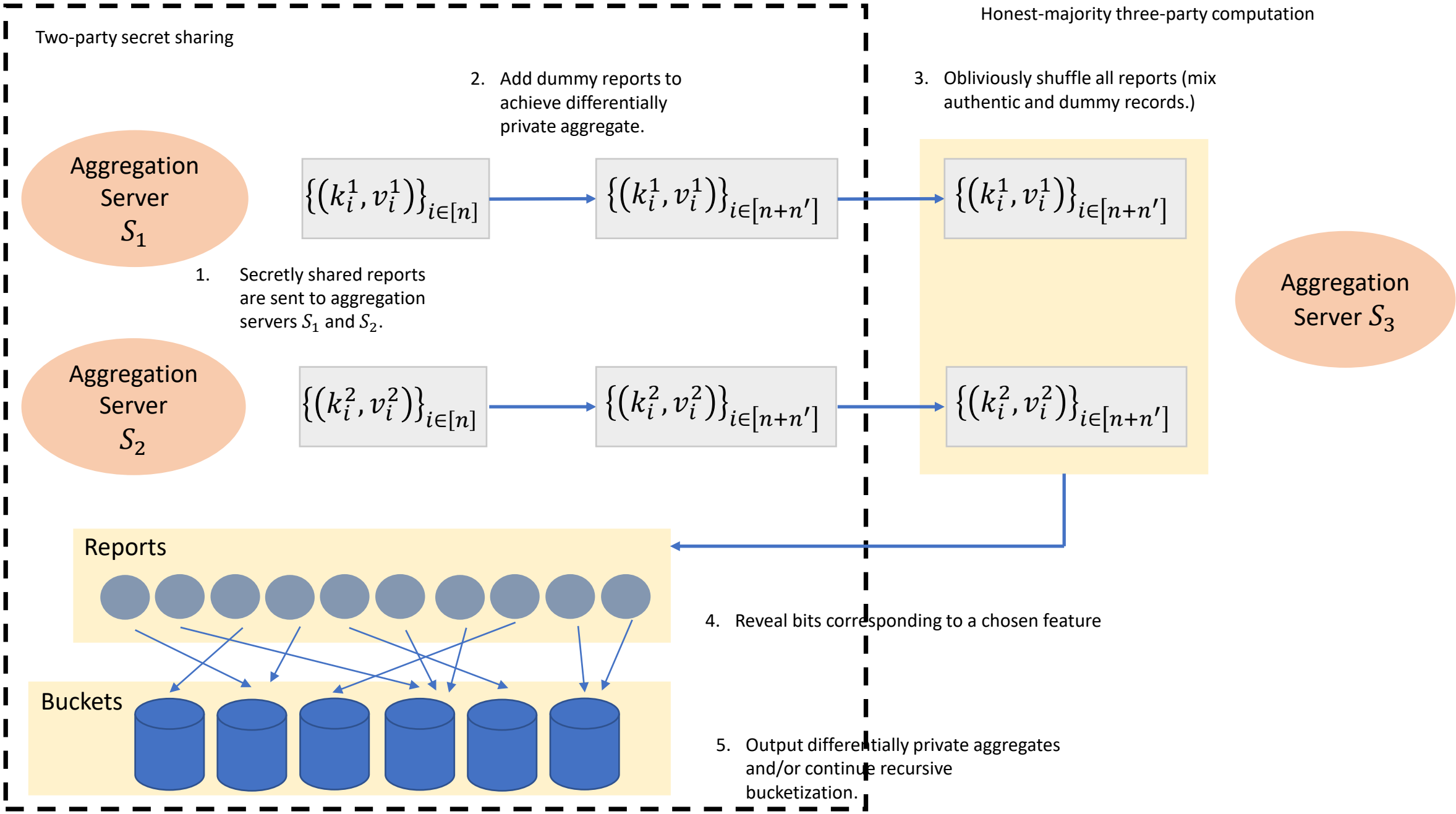
Goals:

- Provide the **same privacy guarantees** as (incremental) DPF
  - Only information revealed to reporting origin or aggregation servers is differentially private
- Allow for more **flexibility** in report length and aggregates computed
  - Build hierarchy of histograms using any subsets of key bits at each level
  - Choose which aggregates to compute on the fly – e.g. decide whether to compute SUM, or first subdivide further
- Provide **better efficiency**
  - Least expensive operations
  - BONUS: hierarchy allows better performance

Detailed analysis: 2021/1490.pdf (iacr.org)

Achieving Privacy Goals




Two-party secret sharing
Honest-majority three-party computation
2. Add dummy reports to achieve differentially private aggregate.
3. Obliviously shuffle all reports (mix authentic and dummy records.)
Aggregation Server S_1
$\{(k_i^1, v_i^1)\}_{i\in[n]}$
$\{(k_i^1, v_i^1)\}_{i\in[n+n']}$
$\{(k_i^1, v_i^1)\}_{i\in[n+n']}$
1. Secretly shared reports are sent to aggregation servers S_1 and S_2.
Aggregation Server S_3
Aggregation Server S_2
$\{(k_i^2, v_i^2)\}_{i\in[n]}$
$\{(k_i^2, v_i^2)\}_{i\in[n+n']}$
$\{(k_i^2, v_i^2)\}_{i\in[n+n']}$
Reports
4. Reveal bits corresponding to a chosen feature
Buckets
5. Output differentially private aggregates and/or continue recursive bucketization.

Performance Evaluation

- Environment: Azure Standard D8s v4, 8vCPU, 32GB RAM. 60 ms network latency
- Dataset size: 10 million reports
- Task: Count => bucketize into 2^{16} buckets (includes only keys not values)

Key size (L bits)	**32**	**64**	**256**	**512**
Add dummy records	26 ms	47 ms	176 ms	355 ms
Shuffling	4 s	6 s	20 s	39 s
Reveal	2.5 s	3 s	7 s	11 s

Performance Evaluation

- Sparse domain: 32-bit keys forming 2^{20} buckets
  - Incremental DPF with “single user contribution”: 0.72 seconds (no sketching, no DP)
  - Bucketization with 1 million records: 10 seconds (end-to-end privacy)

- Heavy-Hitter with 256-bit key
- Dataset size : 400 000 reports (following zipf distribution)

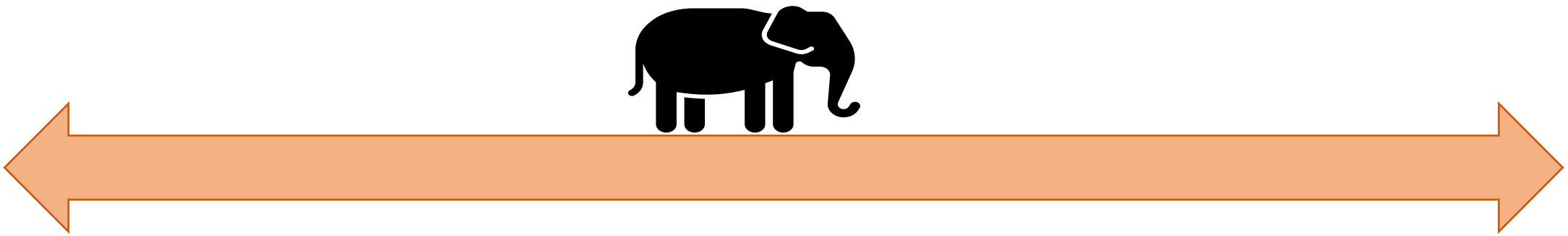
incremental DPF	53 mins
hierarchical Bucketization	28 mins

Wrap-Up Comparisons

COUNTs	Honest-but-curious model	Robustness (malicious clients)	Privacy (malicious server)	Correctness (malicious server)	DP	# of aggregators	Query flexibility
Prio	Yes	SNIPs	Yes	No	No	2	no
DPF	Yes	Sketching	Yes	No	Yes	2	no
Bucketization	Yes	No cost	Yes	No	Yes	3	yes

SUMs	Honest-but-curious model	Robustness (malicious clients)	Privacy (malicious server)	Correctness (malicious server)	DP	# of aggregators	Query flexibility
Prio	Yes	SNIPs	Yes	No	No	2	no
DPF	Yes	Sketching	Yes	No	Yes	2	no
Bucketization	Yes	modulo conversion	Yes	No	Yes	3	yes

Computation vs. Communication



Computation

Communication

- DPF is a successor of Prio
- DPF vs Bucketization: Two orthogonal solutions
- More communication does not necessarily mean more expensive protocol