AEROSPIKE

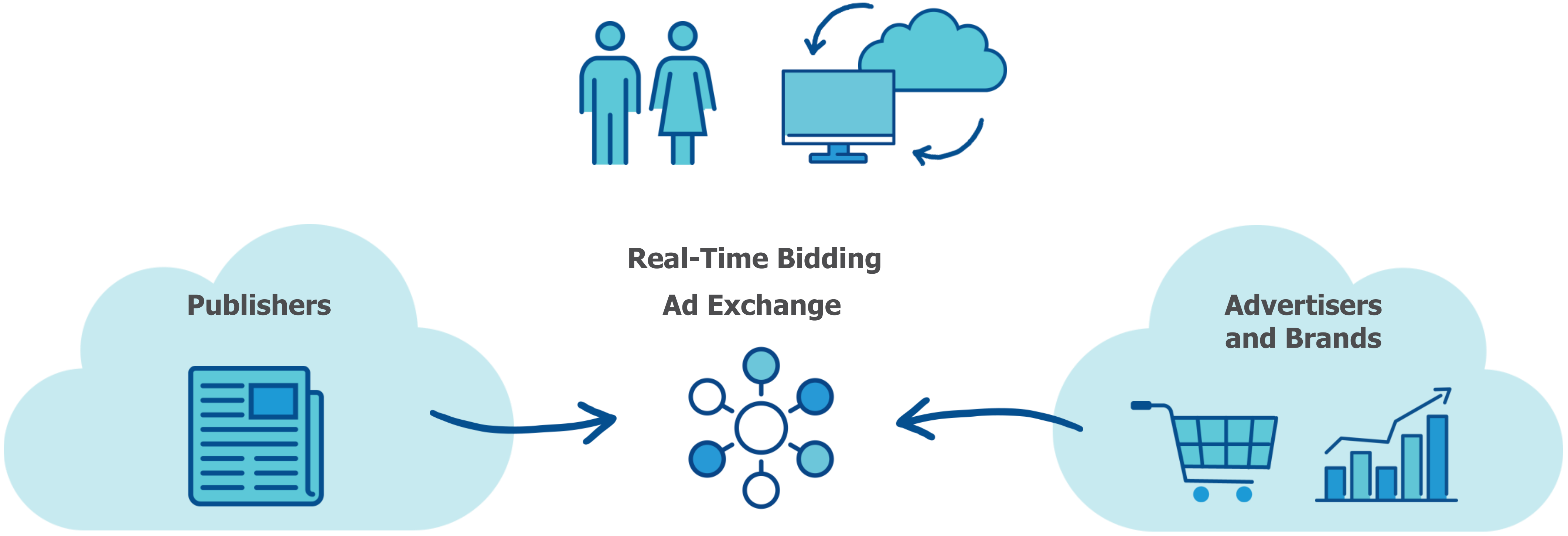
SUMMIT ‘19

theTradeDesk

Integrated Hyper-Scale Hot and Cold Store

Matt Cochran
Director of Engineering
The Trade Desk

The Crux of (or Intro to) AdTech



Real-Time Bidding

Ad Exchange

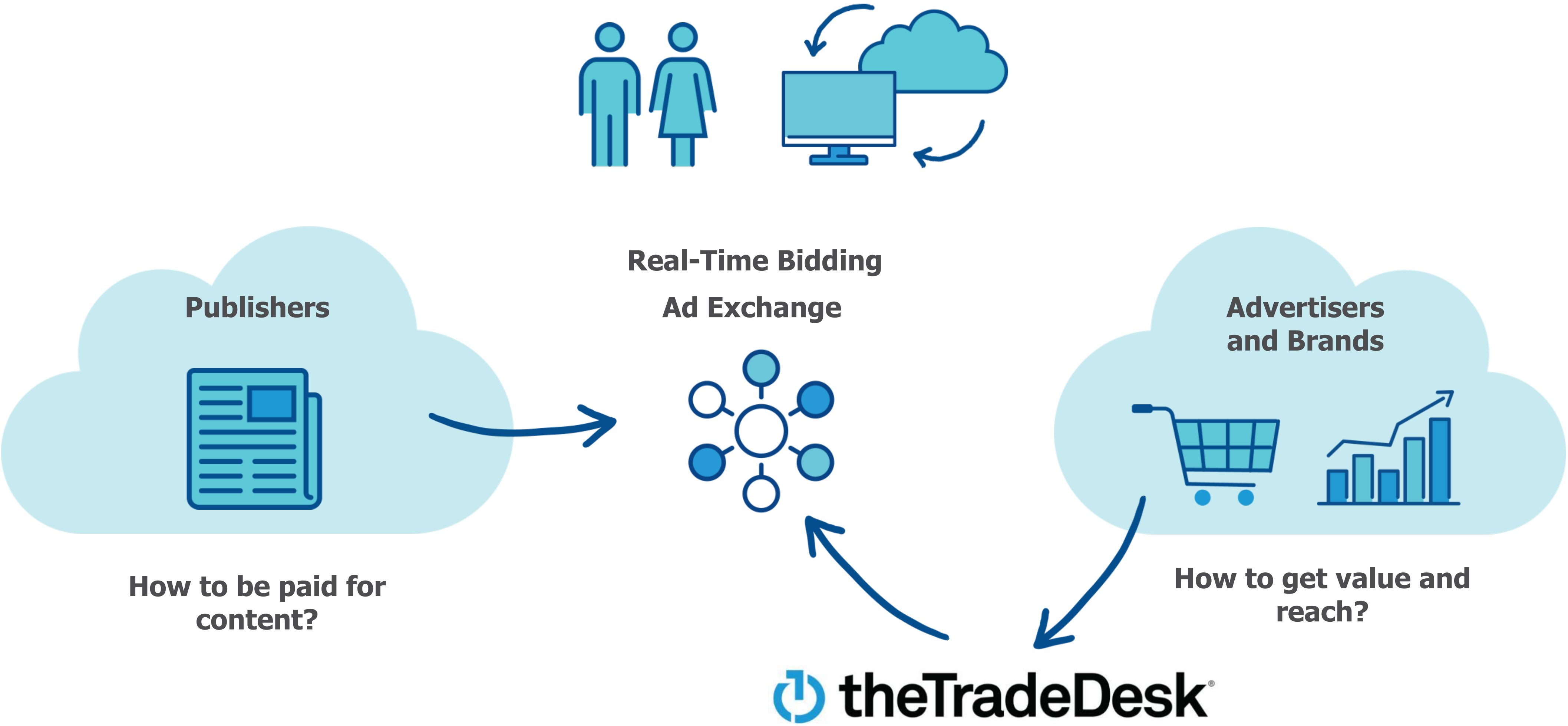
Publishers

Advertisers and Brands

How to be paid for content?

How to get value and reach?

The Crux of (or Intro to) AdTech



Real-Time Bidding

Ad Exchange

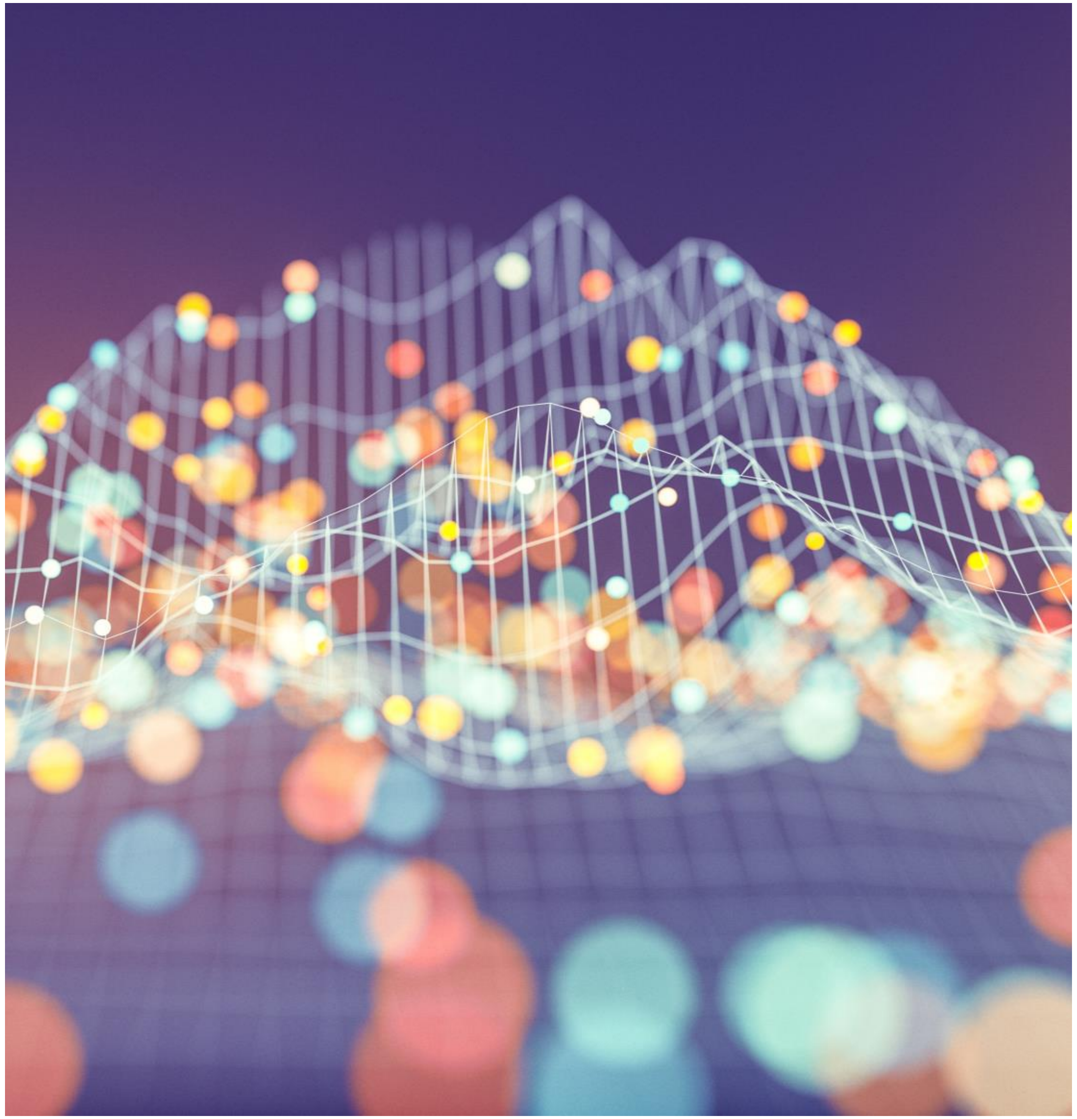
Publishers

How to be paid for content?

Advertisers and Brands

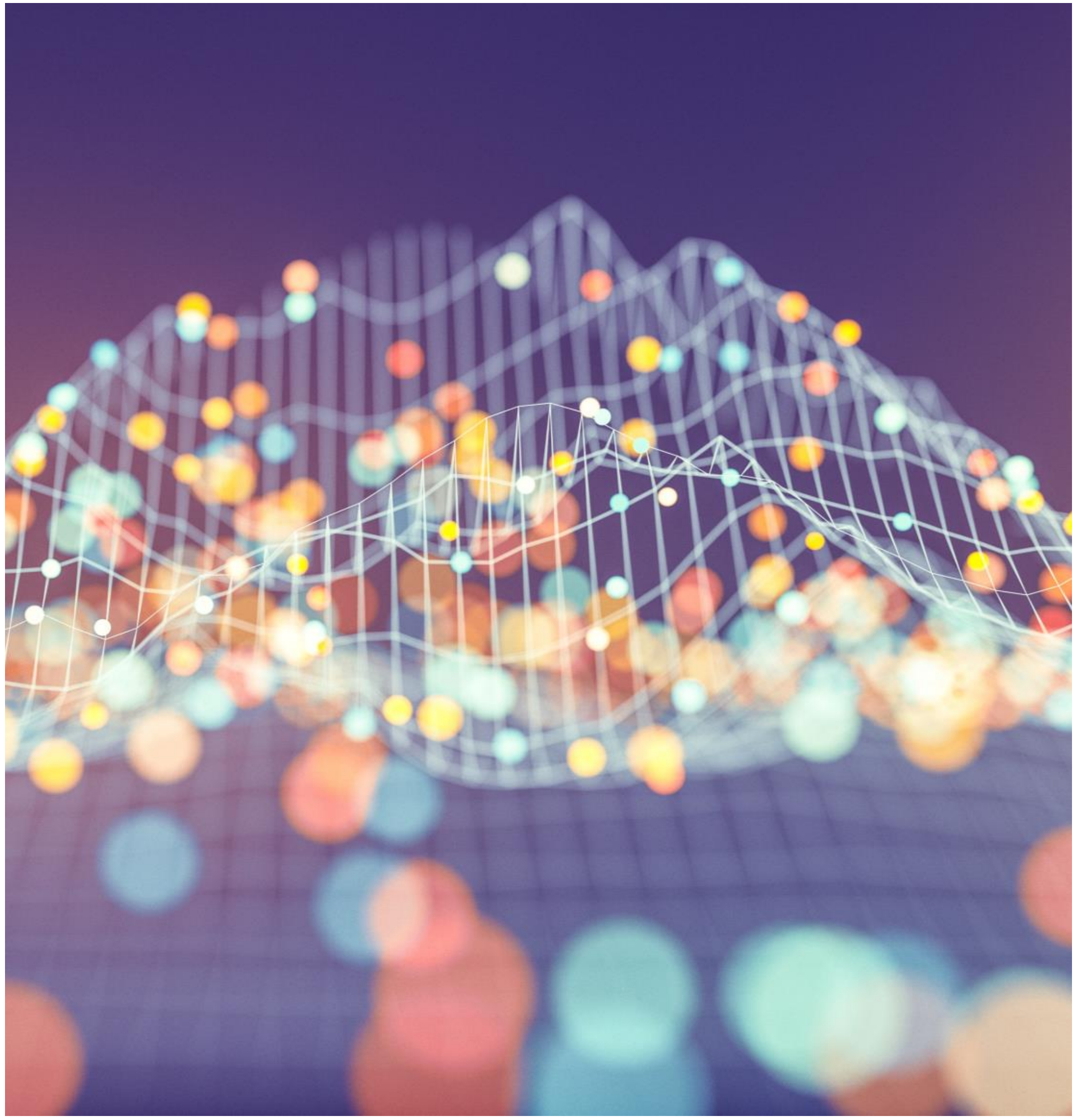
How to get value and reach?

theTradeDesk



Hot and cold storage

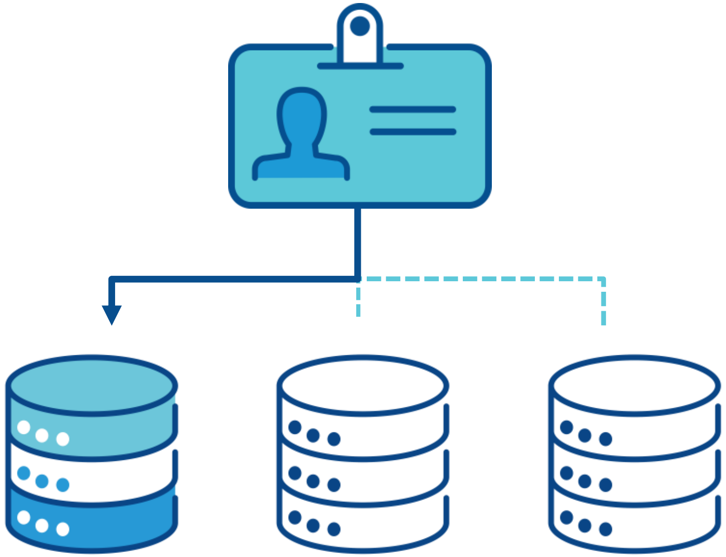
- **Firehose of data - 3pd ecosystem and platform for data**
  - Reading anonymous TDIDs at a rate of 10MM/sec means we have to optimize heavily to the read side in the bidding data centers
  - Data centers service real time traffic from exchange partners, these partners send us request for bids from internet activity that may not be in the same region
  - This results in data segments needing to be replicated globally
  - Serialization/Deserialization CPU costs on the read side mean we have to be careful about TDID record size
  - 50BN records - many with tens of thousands of data points or segments



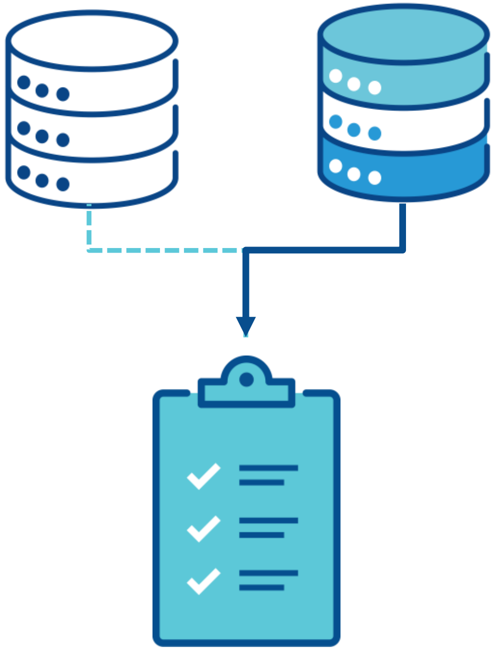
Hot and cold storage

- **Firehose of data - 3pd ecosystem and platform for data**
  - We’re a platform for data, so data partners need to send us data in advance and sell it in our marketplace
  - This means that not all data points are used, usage comes and goes as ad agency customers start and end campaigns targeting one or more of these segments
  - 3rd party data partners send us over 100K QPS and each query can add one or more segments to one or many TDIDs
  - This turns into about 20MM TDID/segment updates a second
  - Being careful means CPU costs for merging data onto TDID records as we have to pick the ‘best’ elements if a given record gets too large
  - However DCs need only the data on the record that is in use by an active campaign
  - Duplicative processing and storage – replicating to each DC means we pay the CPU and storage cost each time

Data storage use cases



Base record:
Anonymous TDID Record
-> list of segments



- **Frozen TDID case – TDIDs can come and go from data centers**
  - How to get only the TDIDs in the data center they are needed in
  - How to store only the TDIDs that are active
  - Classic hot/cold cache concept

- **Frozen Data/Reverse index: List of TDIDs for a given segment**
  - Data usage state change between active/inactive
  - How to keep only the actively used segment data onto the record
  - How to thaw segments that change to active

Solution: Cassandra!

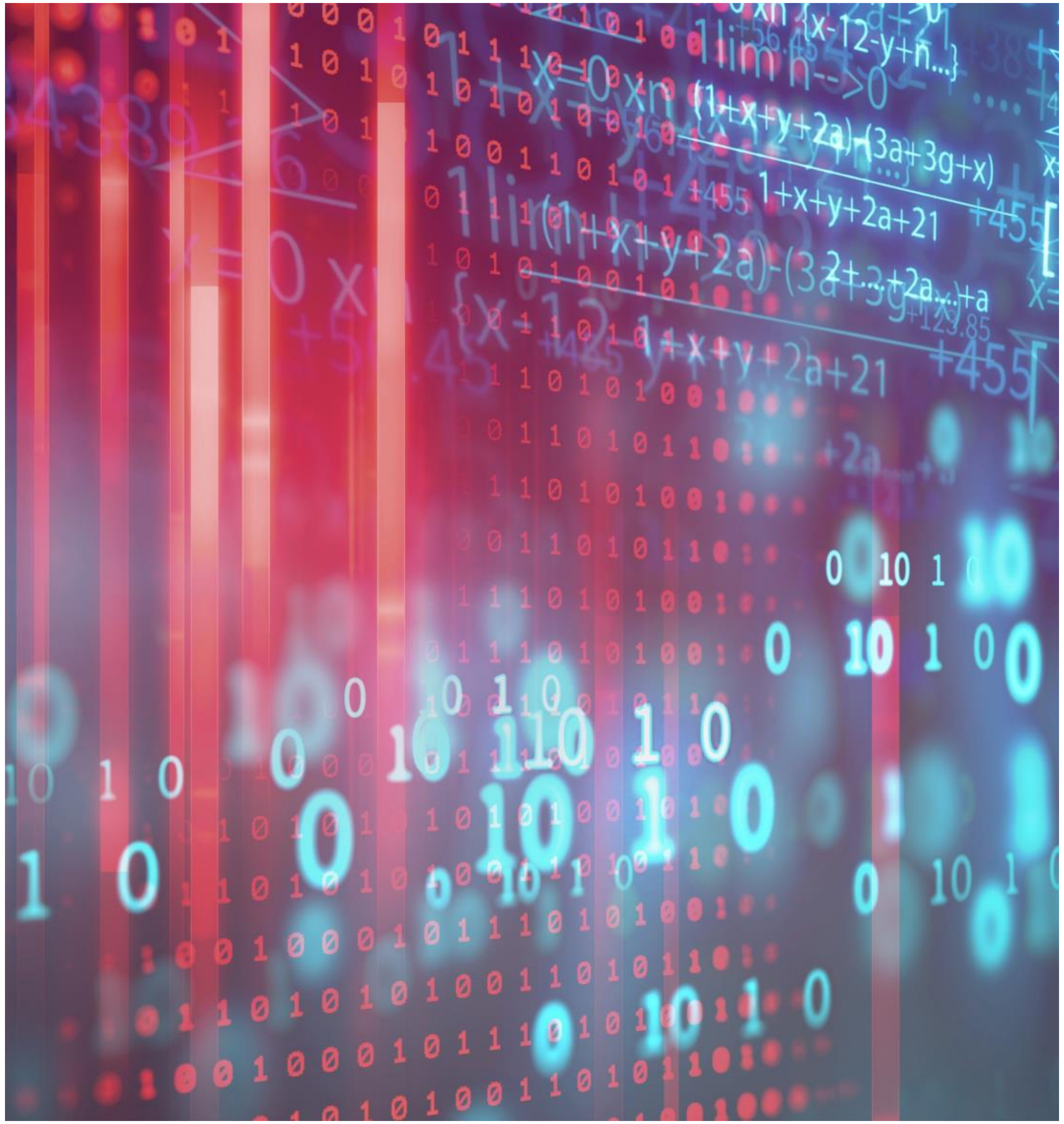
Phenomenal intake rates

- Spoke to the team at a popular video provider about their success with it

Data organization:

- TDID/SegmentID + metadata
- Write rates of 20MM rows/sec, many simultaneous updates on a given TDID

Avoiding read-update-write for speed dictates model TDID/Segment ID to support streaming



But wait, that's just one use case

How do we handle the secondary index?

- Far lower read rate for this, but cardinality can be very high: think of a transition of ”likely male" from active to inactive: 5BN TDIDs?

Ugh, can we even store this?

Problems everywhere: Use case 1

CPU Constraints

- Repetitive updates of the same elements on the same ID means lots of compaction
- Compaction means CPU
- Compression means CPU
- Java means GC, even using optimized versions of GC
- …Which means more CPU

Need to have a TON of CPU for a relatively small amount of data

- High ratio of CPU to data size means more machines…

Data skew of TDID/Segment data combination was really high, so some machines saw significantly more (2-3x more)

- Cascading performance issues due to CPU consumption of all the above

…Profit? No

Problems everywhere: Use case 2

Facepalm example: never noticed segment population from a data partner that was effectively *ALL* TDIDs

- Skew of Segment/TDID made secondary indexes infeasible
- 2 cluster solution also had a lot of skew

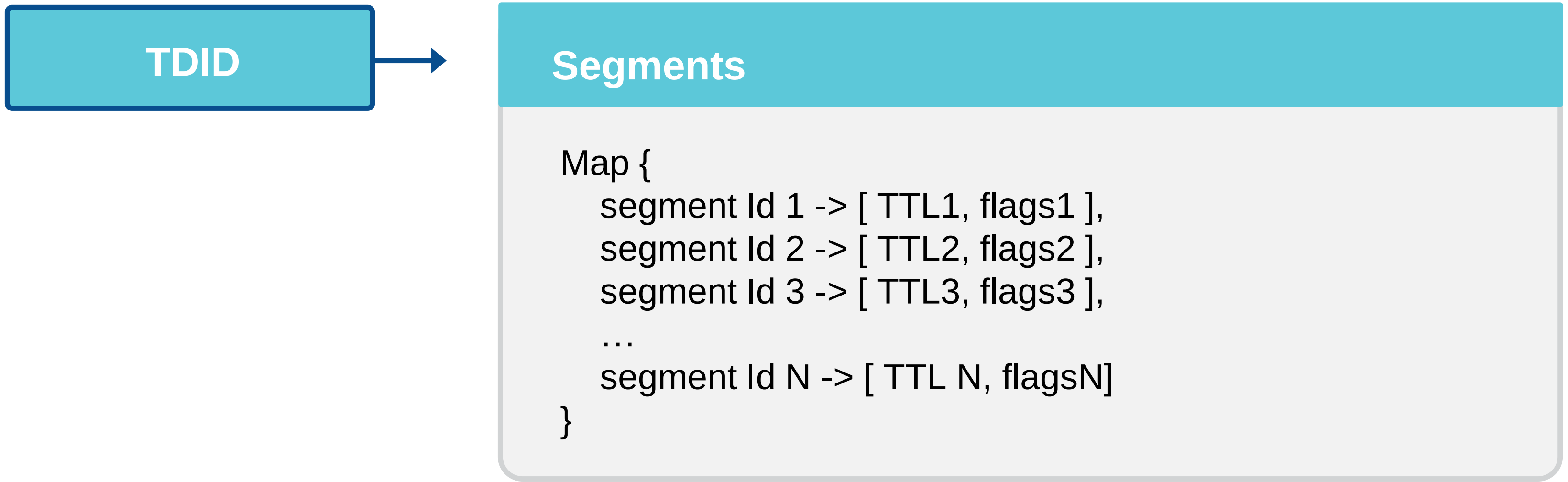
Finally made use case 1 "work":

- 500 i3.2xl nodes
- RF=1 w/backups + replay as recovery strategy as the skew caused some machines to fail, which spread load to others, which then failed…
- Writes - 22MM rows/sec
- Reads - 20MM rows/sec

January 2018:
Discussion with Aerospike

- XDR available but global distribution and optimized storage format made it infeasible as we needed application level support
- Secondary indexes still problematic
- Deferred discussion to Fall 2019

October 2018: Aerospike proposal — data structure



TDID

Segments

```
Map {
    segment Id 1 -> [ TTL1, flags1 ],
    segment Id 2 -> [ TTL2, flags2 ],
    segment Id 3 -> [ TTL3, flags3 ],
    …
    segment Id N -> [ TTL N, flagsN]
}
```

- Users can have attributes associated with them in separate bins
- All segment IDs are stored in one record

October 2018: Aerospike proposal — use cases

Use Case 1 – Thaw a TDID

- Primary key read so will run quickly
- Returns all the segments for a TDID

Use Case 1 - Write a TDID's segments into the record

- Map operations to insert the elements server side, so does not need read-update-write
- Also do trim by TTL: removeByValueRange(null, [expiry, “”])

Use Case 2 - Find all TDIDs matching a segment

- Scan across all TDIDs
- Predicate filter: LUT < 90 days AND SegmentMap contains
  Key SEGMENT, maybe shard using mod TDID

October 2018: Aerospike proposal — sizing

From Tim F and Ronen:

- Significant data volume: 50B records x 750 x 20 bytes ~= 750TB raw data
- With RF=2, 50% fragmentation ~= 3PB data storage needed.
- With optimizations, the average record can come down to ~ 11,666 bytes, so 50B is ~583TB with 2.3PB needed

AWS sizing:

- 2.3PB ~= 1230 drives of 1.9TB each.
- I3.16xlarge has 8 drives each
- Would need 154 nodes

October 2018: Aerospike proposal — sizing

…Still too big: for the given footprint, we get compression in Cassandra and MsgPack is still not enough — comes out to 150 i3.16xl

Other concerns: scans across users even sharded is problematic (but a neat trick)

Srini: give us a month

Development

Aerospike delivered record-level compression in a month

- Reduced machines needed to 60
- Record-level compression good, Cassandra RLE better but advantages in Aerospike model well worth it

We came up with an alternative algo to solve for secondary index need

- Works better if all the segment IDs on record are in one location (as opposed to the TDID + SegmentId rows in Cassandra) - Removes need for scans

TDID -> [data] approach enabled 4 other use cases we didn't think of before due to the flexibility of the data model

Preliminary Result

Still turning this up – over 50% done, but biggest savings to come

- Improved overall TDID targeting for customers, reaching more of the browsers they are trying to reach
- Reducing number of DC processing machines now, more to come as we turn up higher.
- Reduction in AWS egress traffic

Future Benefits

- Aggregating 3rd party data provider requests also enables better control of firehose as well as DC CPU reduction
- Reduces data center Aerospike storage requirements, and eventually cluster sizes