BlueVoyant: Detecting Security Dumpster Fires on the Internet

Alfredo Gimenez, Adam Najman, Tucker Leavitt, Tyler Flach

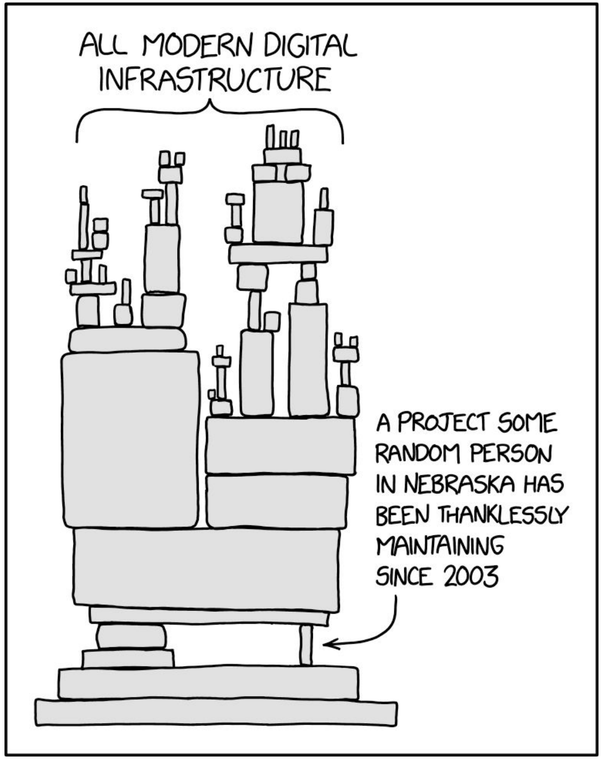
BEAM SUMMIT Austin, 2022

Who is BlueVoyant?
Who Are We?

Third Party Risk Detection What is it and Why is it Hard?

BEAM SUMMIT Austin, 2022

What is Third Party Risk?

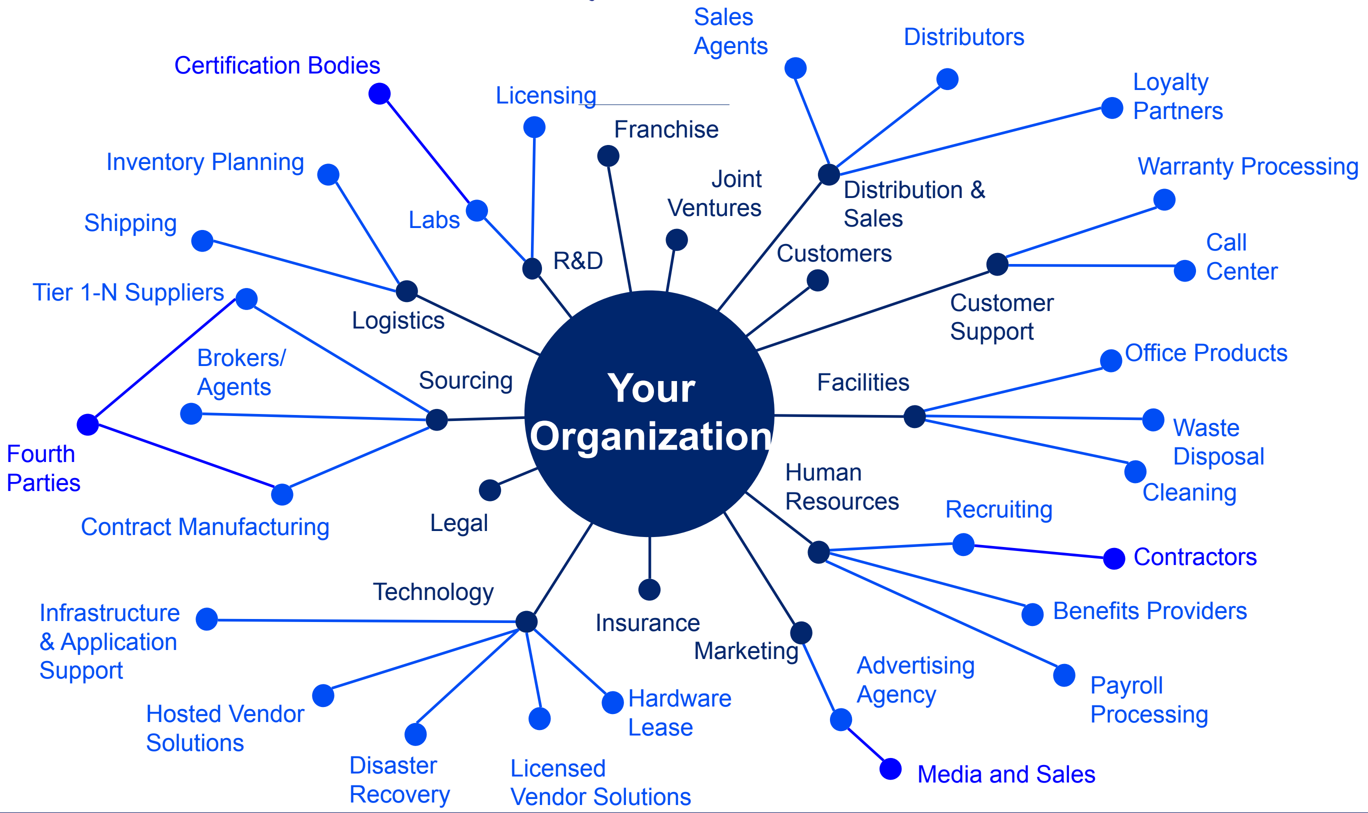

ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

https://xkcd.com/2347/

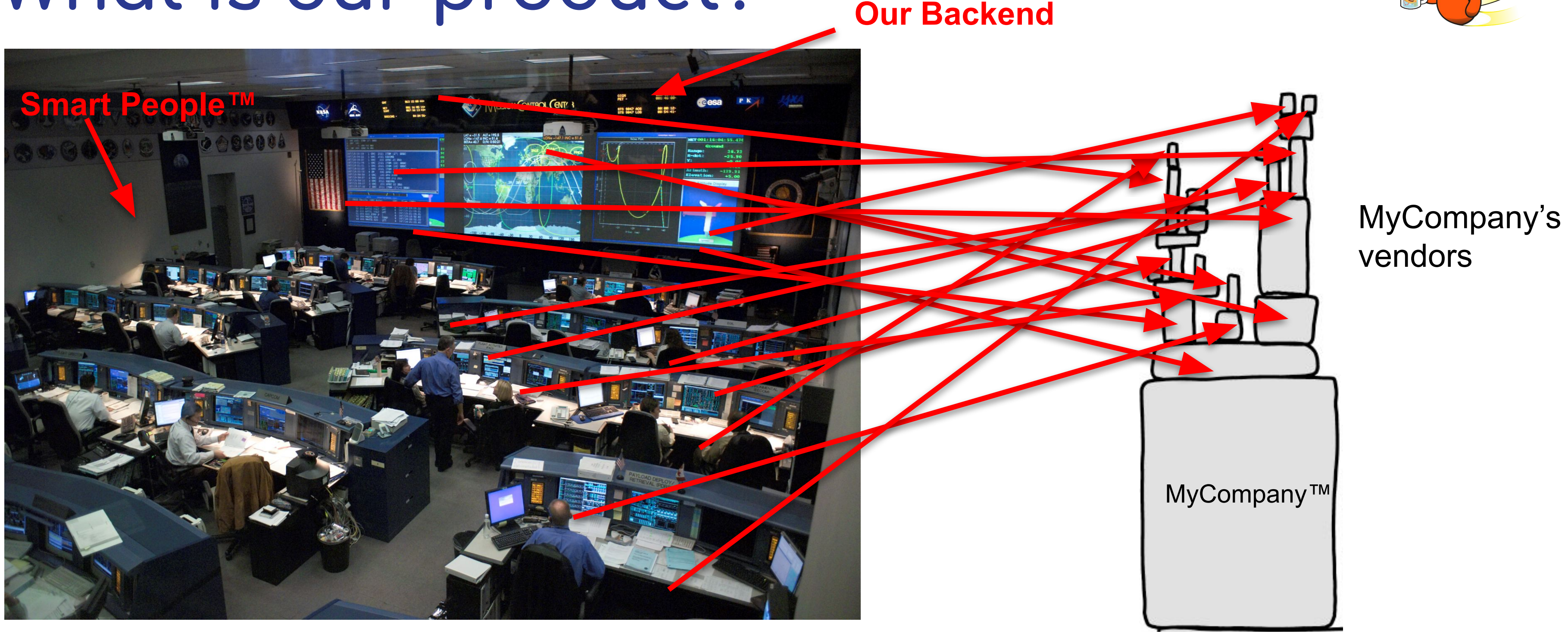
First Party Risk – hack the bank's central servers

Third Party Risk – Don't attack the bank directly; go after one of their suppliers or a smaller company they are buying (target the weakest link)

What is Third Party Risk?



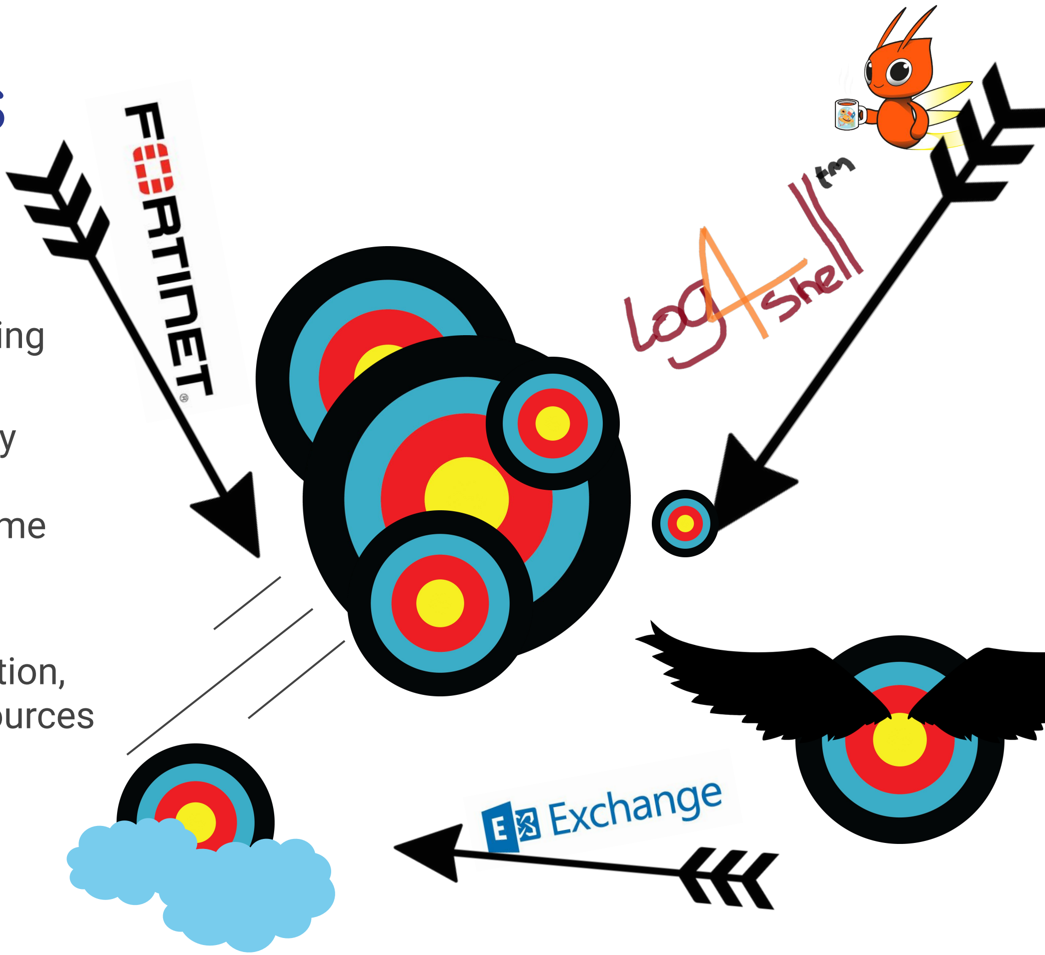
What is our product?



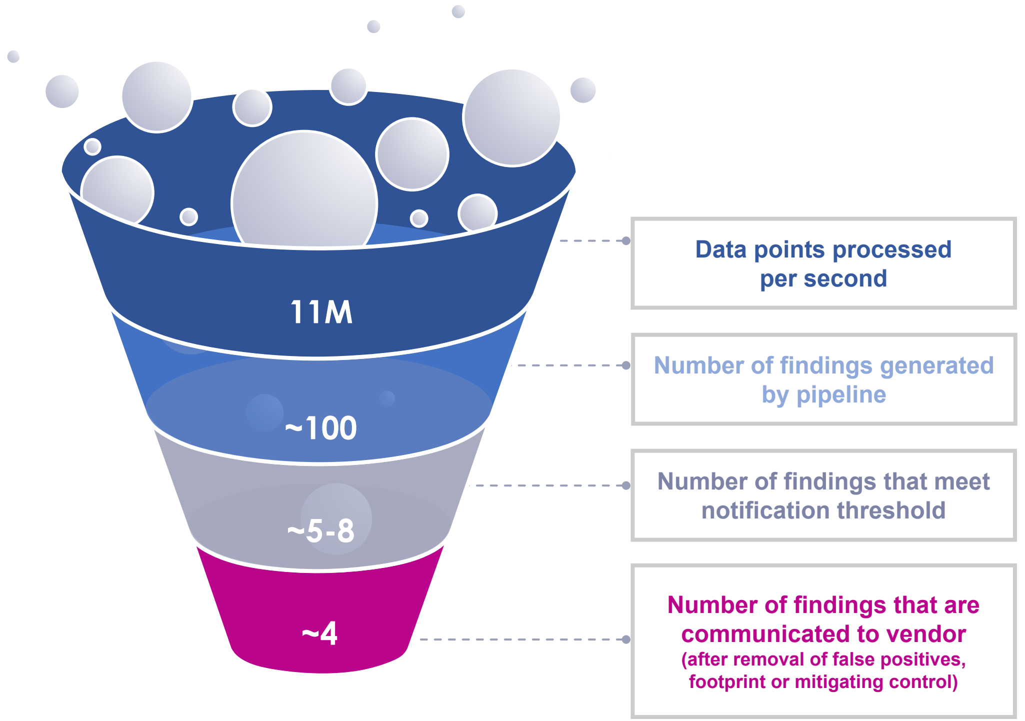
"One of your vendors has a remote office that didn't patch the latest VPN server vulnerability and has remote access exposed. We opened a case, and worked with them to patch the issue and close the port."

Big Moving Targets

- **Lots of data**
  - Some streaming, some batch
  - Up to 11M events per second incoming
- **Disparate data**
  - Different formats, semantics, delivery
  - Data arrives late, changes schema/semantics, changes in volume
- **Dynamic footprints**
  - Shared assets, cloud-hosted assets
  - Mergers/acquisitions, growth, reduction, adoption of third-party services/resources
- **Dynamic threat landscape**
  - Emerging vulnerabilities
    - (or knowledge of them)
  - Emerging attack vectors



Important Needles, Big Haystack



Data points processed per second

Number of findings generated by pipeline

Number of findings that meet notification threshold

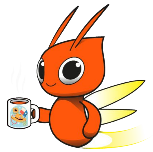
Number of findings that are communicated to vendor
(after removal of false positives, footprint or mitigating control)

Solution: Prophet

Parse and Index First – Ask Questions Later!

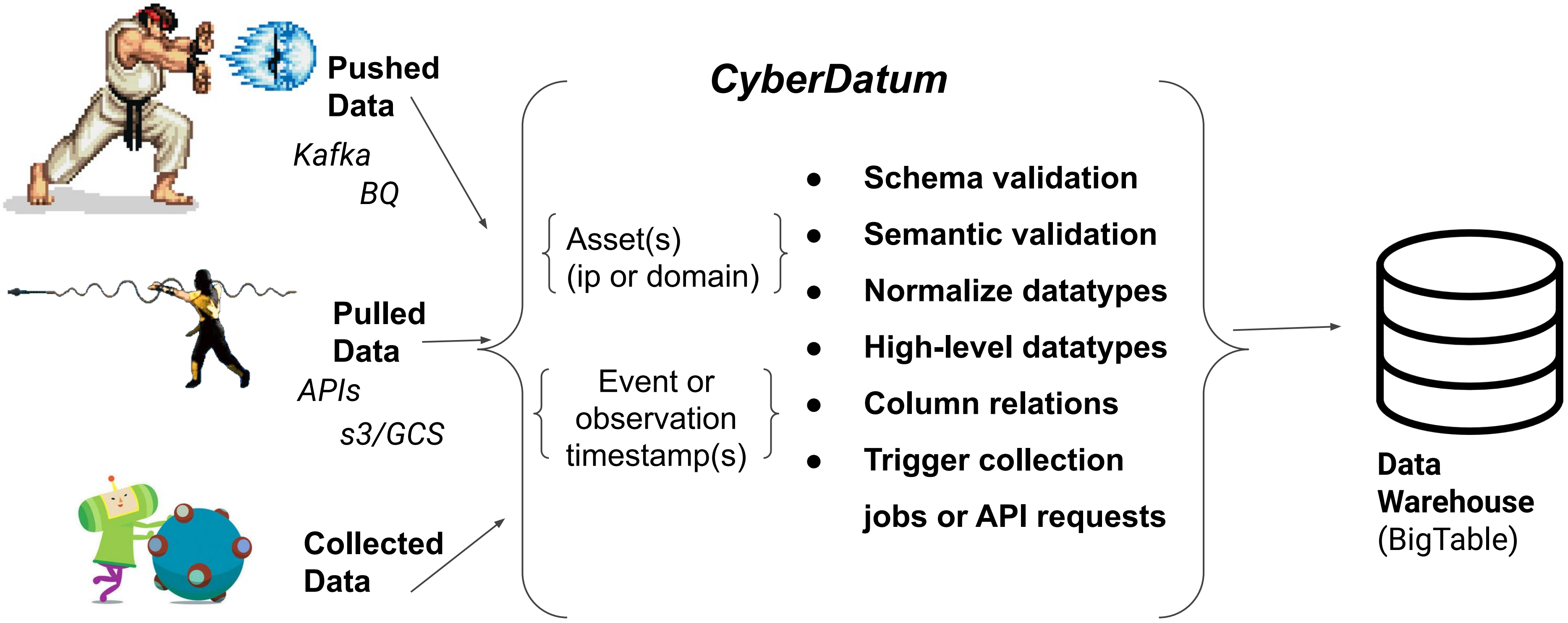
Any analytic we derive today may be wrong tomorrow, all we know is what was observed

Why Beam?
Why not <insert your favorite buzzword>?



- **B**atch and str**EAM** data sources ;-)
- Why not SQL on some backend?
  - We tried it!
    - SQL is limiting, and quickly gets hard to maintain
    - Latency XOR throughput
  - Real code (not just SQL), high-level types, caches, hit APIs
    - Workflows (not just reading/writing)
- Adds high throughput to a low-latency backend*
  - *With some effort (see: *rest of this talk*)
- Manage business logic, not virtual machines + tuning params (sorry spark)
  - Beam + Google Dataflow :chefs-kiss:

Beam as a Data Model Layer



Pushed Data

Kafka

BQ

Pulled Data

APIs

s3/GCS

Collected Data

CyberDatum

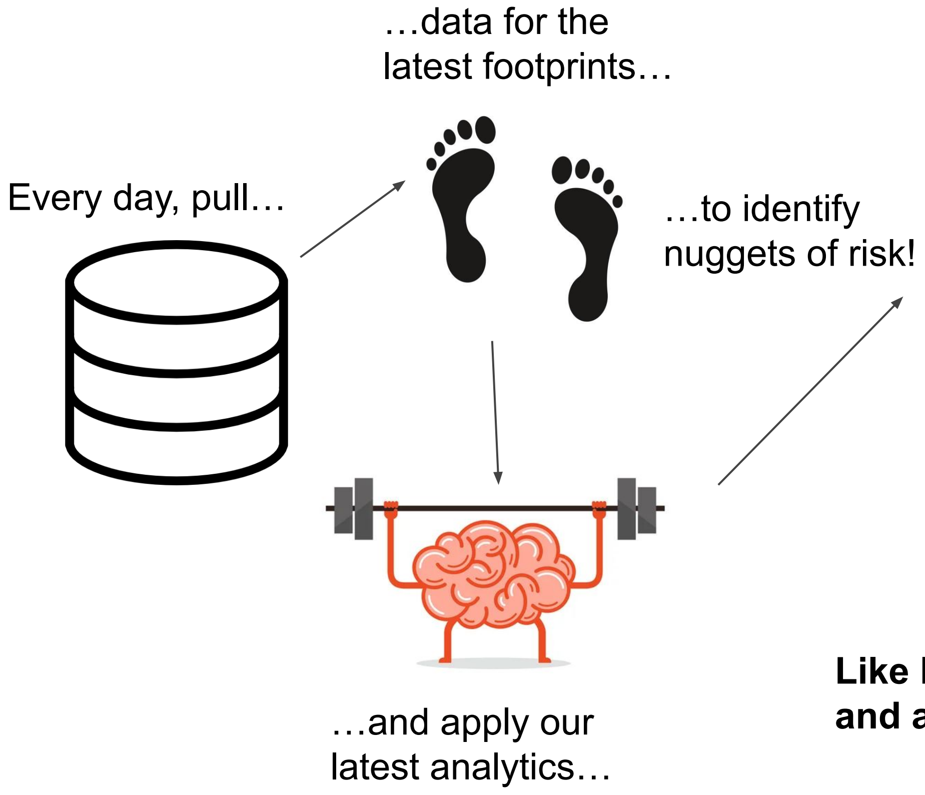
Asset(s)
(ip or domain)

Event or observation timestamp(s)

- **Schema validation**
- **Semantic validation**
- **Normalize datatypes**
- **High-level datatypes**
- **Column relations**
- **Trigger collection jobs or API requests**

Data Warehouse
(BigTable)

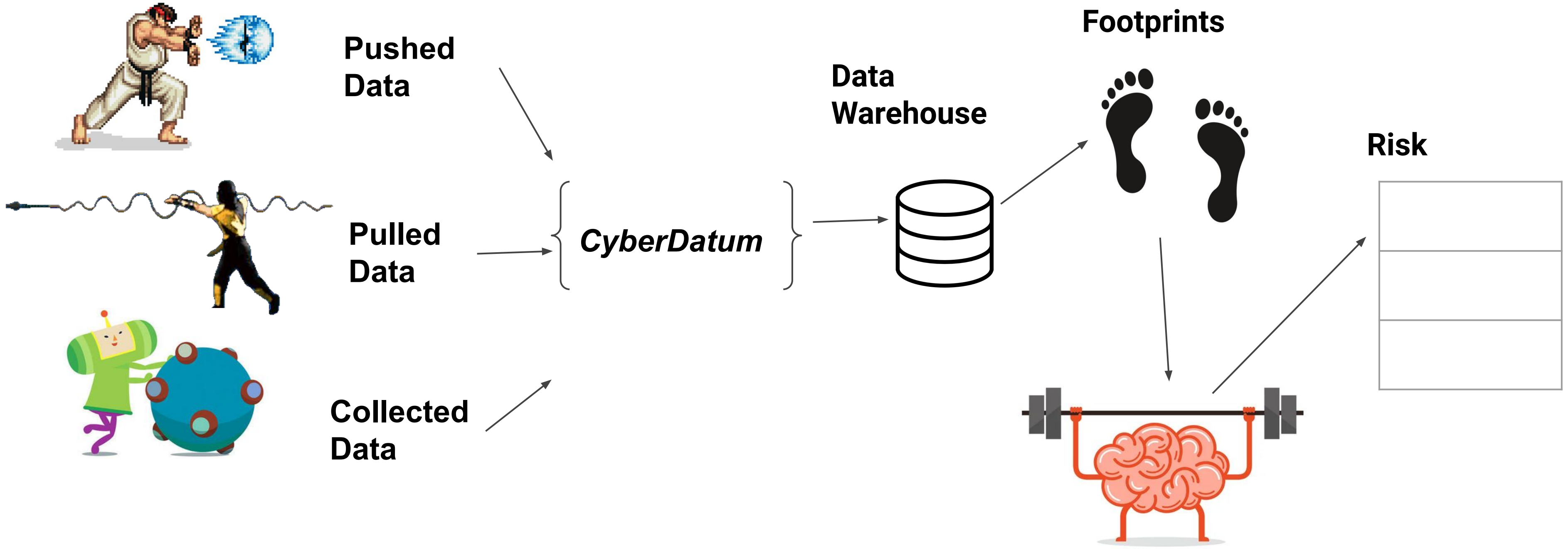
Beam as an Analytic Engine


Every day, pull…
…data for the latest footprints…
…and apply our latest analytics…
…to identify nuggets of risk!

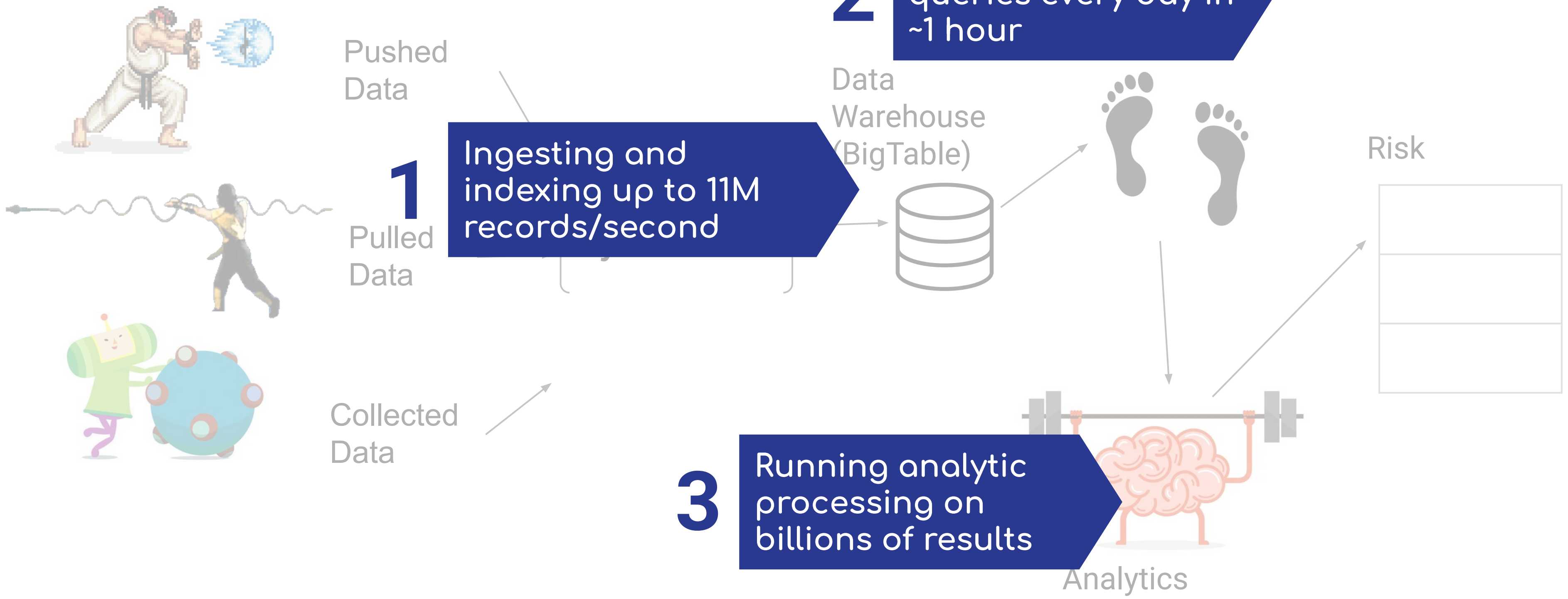
HealthCareCompany uses *DataManagementCompany*, which has a bunch of their personal health data (HIPAA) in a publicly readable bucket!
SaasCompany uses *HRCompany* that runs a mail server with known remote code vulnerabilites!
BigCompany recently acquired *SmallCompany*, who was actively targeted by known malicious botnet servers yesterday!

Like having a billion human analysts running daily queries and analysis on the results

Beam Challenges



Beam Challenges



1 Ingesting and indexing up to 11M records/second

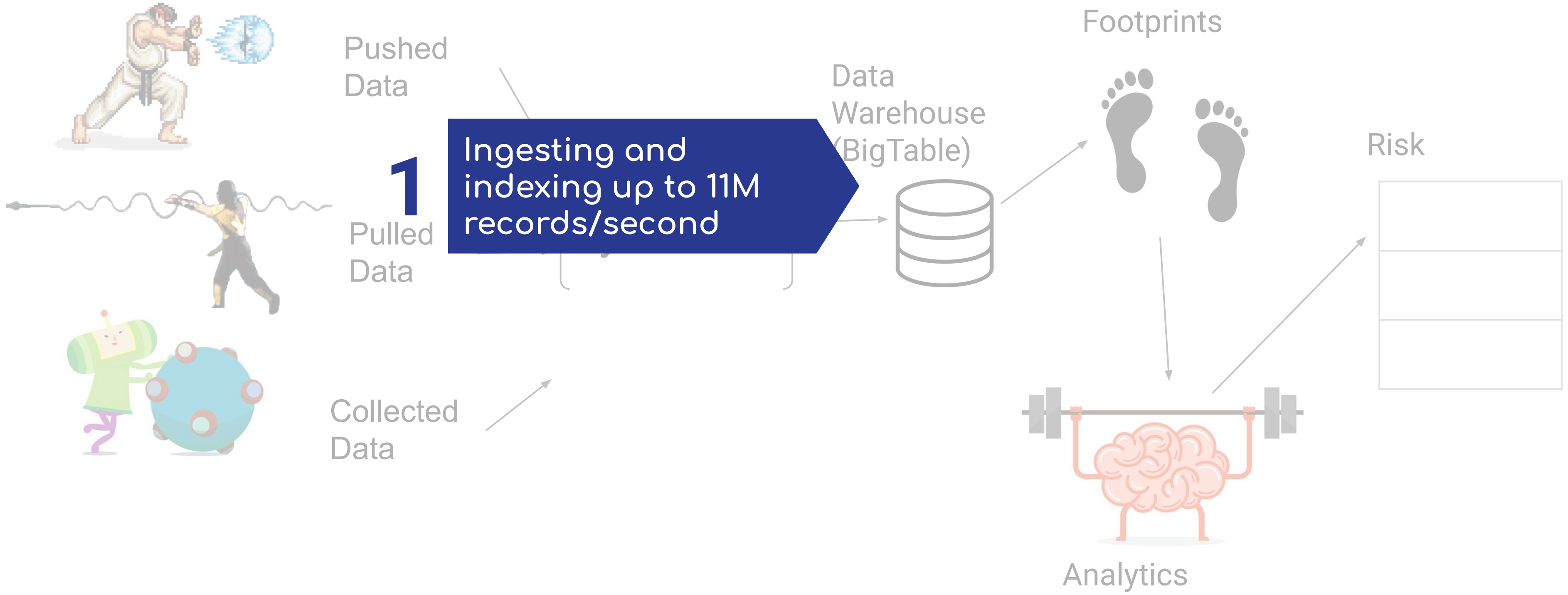
2 Running billions of queries every day in ~1 hour

3 Running analytic processing on billions of results

Ingesting and indexing up to 11M records/second

BEAM SUMMIT Austin, 2022

Beam Challenges

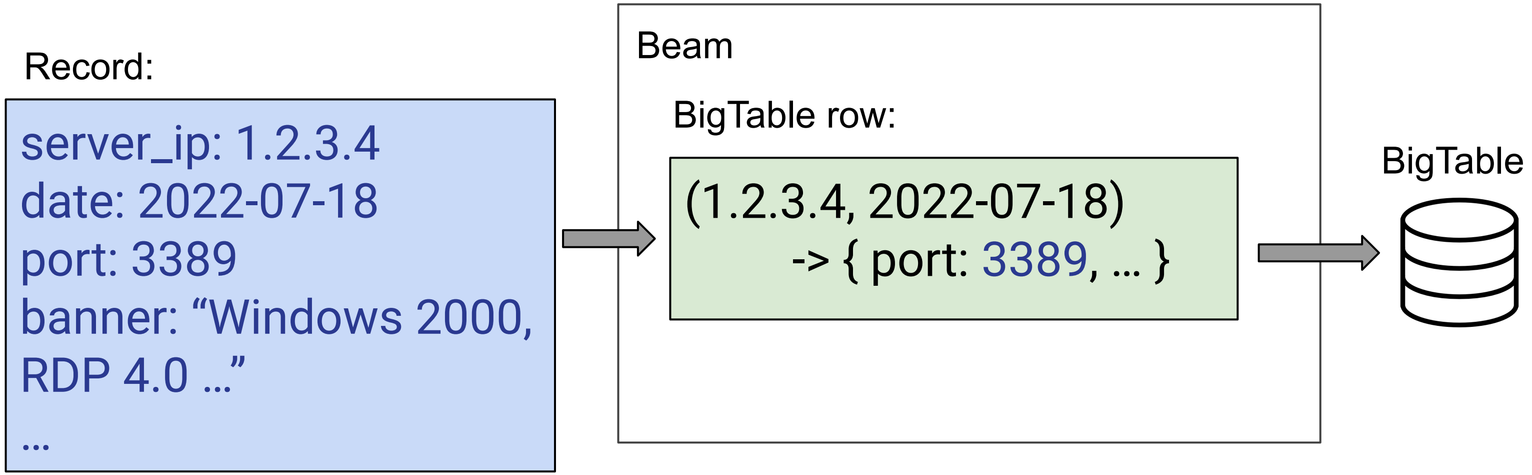


1

Ingesting and indexing up to 11M records/second

A Typical Datasource

- Manageable data volume for Dataflow and BigTable
- Batch jobs (mostly)
- Single index (one BigTable row per record)



server_ip: 1.2.3.4
date: 2022-07-18
port: 3389
banner: "Windows 2000, RDP 4.0 …"
…

(1.2.3.4, 2022-07-18)
-> { port: 3389, … }

- Dataflow + BigTable is an efficient and cost effective solution 🎉

The Problem Datasource

- High-volume
  - p50: 2M records/s, p95: 11 M records/s
- Continuous stream
- Multiple indexable fields (many BigTable rows per record)

Record:

```
client_ip: 1.2.3.4
qname: badguyz.net
answer_ip: 5.6.7.8
timestamp: 2020-12-09
...
```

Queryable as:

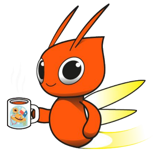
- → (1.2.3.4, 2020-12-09)
- → (badguyz.net, 2020-12-09)
- → (5.6.7.8, 2020-12-09)

PROBLEM CHILD
Attila the Hun. Ivan the Terrible. Al Capone.
They were all seven once.

We’d need to write at least ~25 M rows/s to BigTable

- Would need a very large BT cluster, too expensive

How do we reduce costs?



Bottleneck: BigTable key creation.

- Goal: reduce the number of keys we write to BigTable.
- group records into timestamp bins, and write each bin to an individual row:

(1.2.3.4, 2020-12-09T01:00:01) -> [record-A]

+

(1.2.3.4, 2020-12-09T01:00:59) -> [record-B]

=

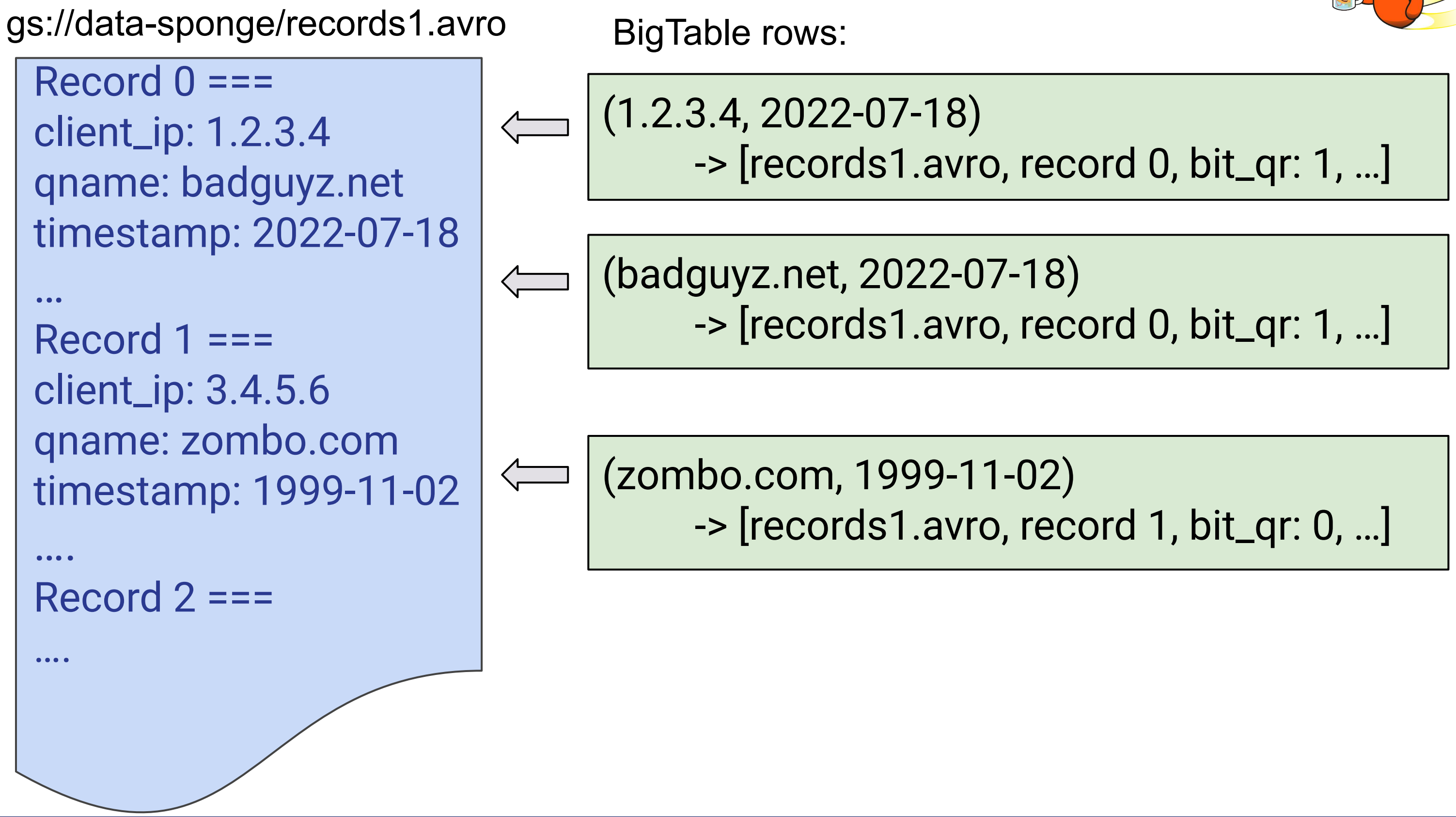
(1.2.3.4, 2020-12-09T01:00) -> [record-A, record-B]

- Improves BigTable write throughput
- But shuffle is expensive, need many dataflow workers to finish in time

Any problem in computer science can be solved with another level of indirection.

(attributed to David Wheeler).

Records on Google Cloud Storage, indices in BigTable



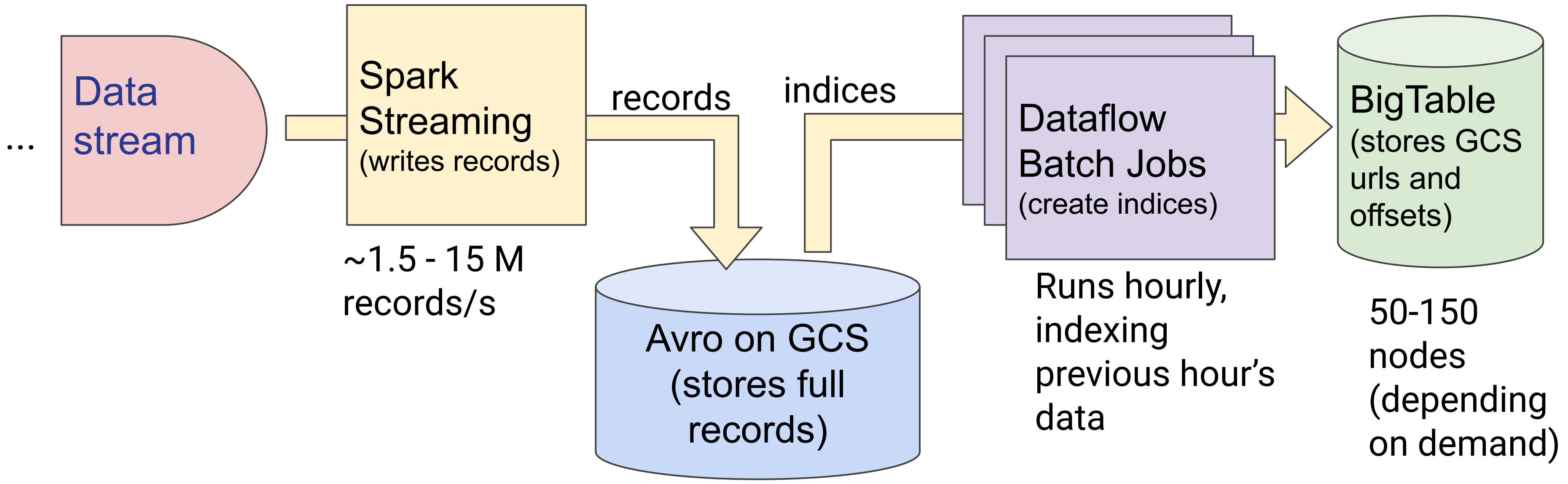
gs://data-sponge/records1.avro

```
Record 0 ===
client_ip: 1.2.3.4
qname: badguyz.net
timestamp: 2022-07-18
...
Record 1 ===
client_ip: 3.4.5.6
qname: zombo.com
timestamp: 1999-11-02
....
Record 2 ===
....
```

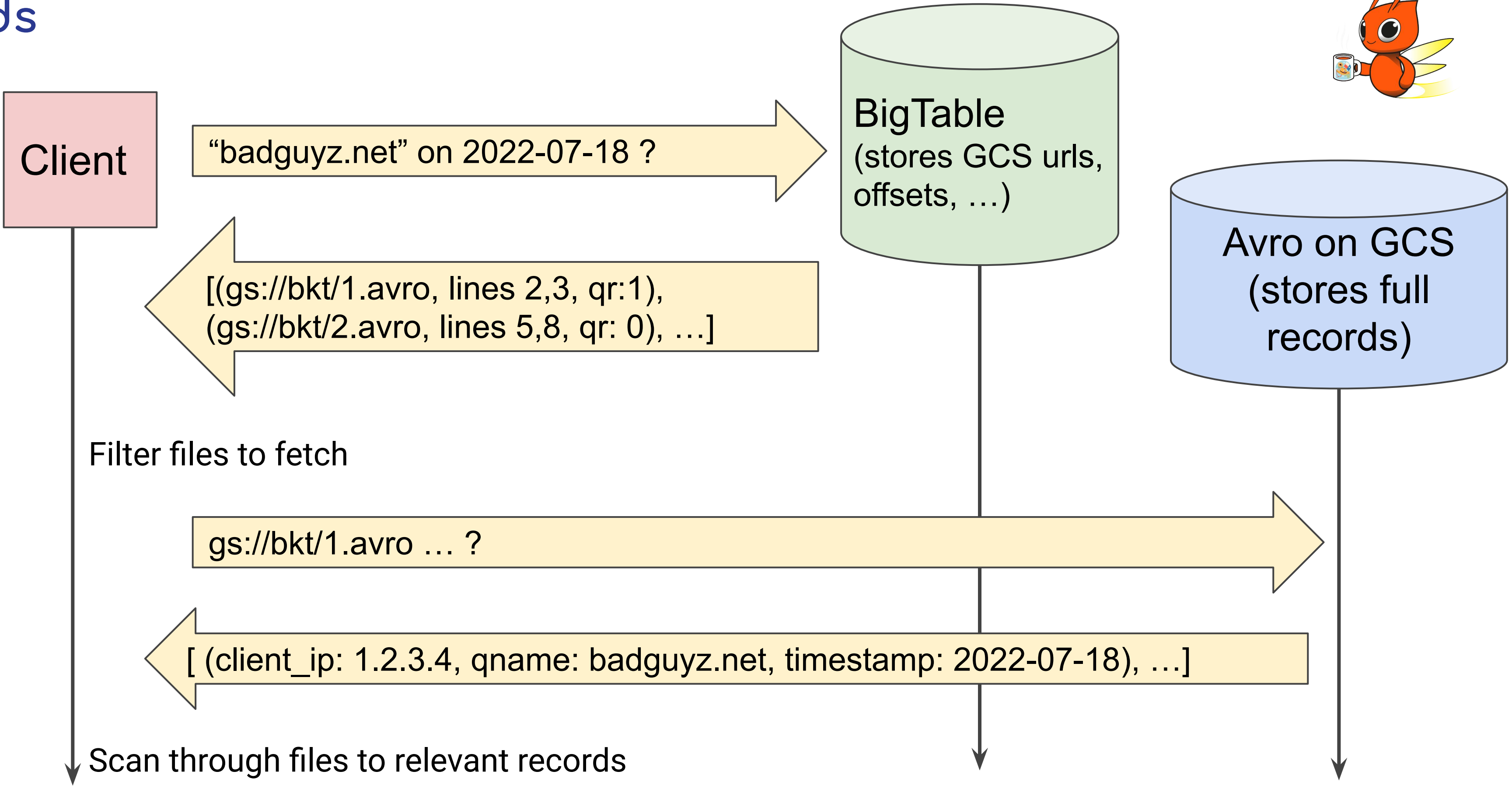
BigTable rows:

```
(1.2.3.4, 2022-07-18)
      -> [records1.avro, record 0, bit_qr: 1, ...]

(badguyz.net, 2022-07-18)
      -> [records1.avro, record 0, bit_qr: 1, ...]

(zombo.com, 1999-11-02)
      -> [records1.avro, record 1, bit_qr: 0, ...]
```

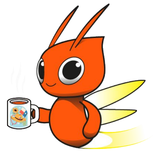
Writes


...
Data stream
Spark Streaming (writes records)
records
indices
Dataflow Batch Jobs (create indices)
BigTable (stores GCS urls and offsets)
~1.5 - 15 M records/s
Avro on GCS (stores full records)
Runs hourly, indexing previous hour's data
50-150 nodes (depending on demand)

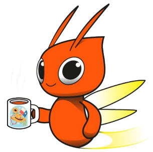
Reads


Client
"badguyz.net" on 2022-07-18 ?
BigTable
(stores GCS urls, offsets, …)
[(gs://bkt/1.avro, lines 2,3, qr:1),
(gs://bkt/2.avro, lines 5,8, qr: 0), …]
Avro on GCS (stores full records)
Filter files to fetch
gs://bkt/1.avro … ?
[(client_ip: 1.2.3.4, qname: badguyz.net, timestamp: 2022-07-18), …]
Scan through files to relevant records

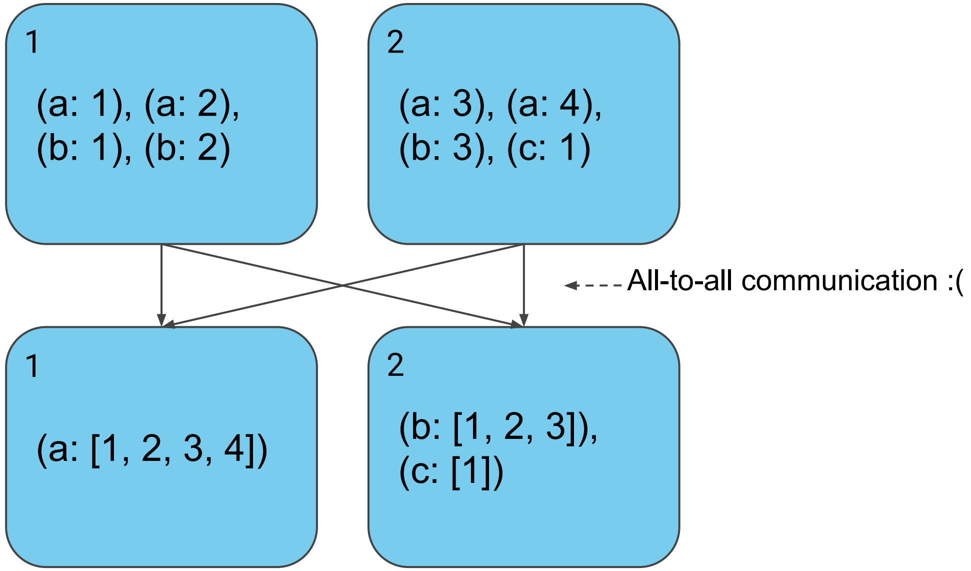
“Local Group-By”



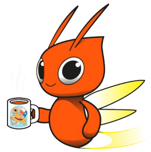
“Local Group-By”

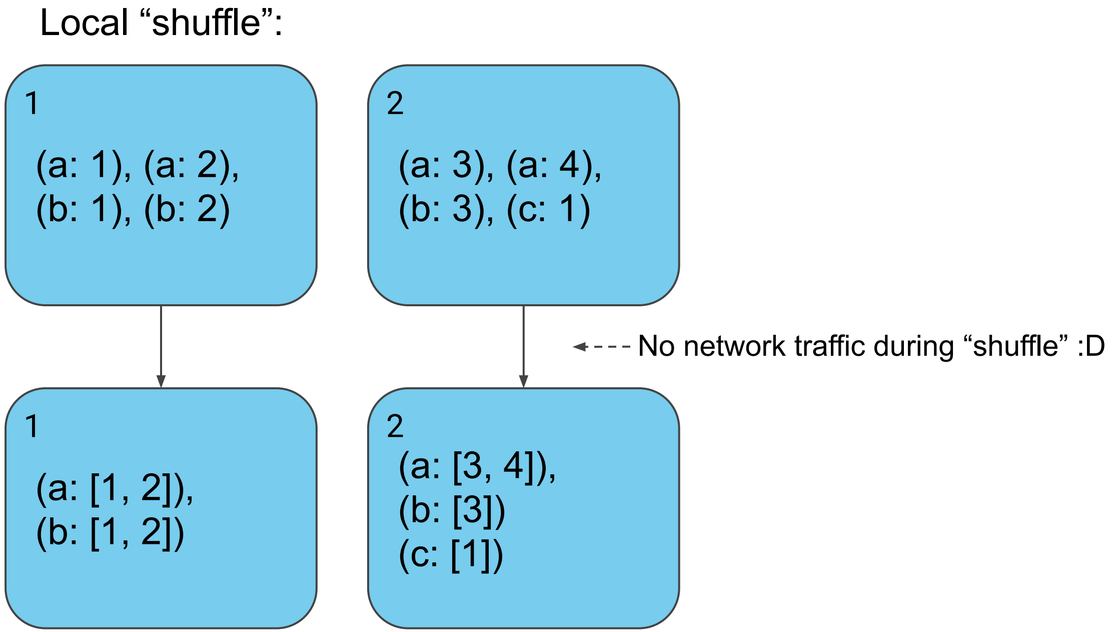


Classic map-reduce shuffle:



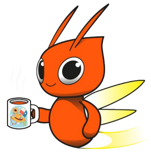
“Local Group-By”





We don’t care about getting all the “a”s on one worker, we just want fewer key-value pairs. Avoids expensive ($ and time) shuffles.

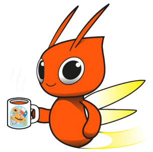
Local Group-By as a Beam DoFn



```
// Each DoFn gets its own local set of groups.
HashMap<String, List<RecordT>> groups;
public void startBundle() {
   groups = new HashMap<>();
}
public void processElement(@Element KV<String, RecordT> element, …) {
   groups.compute(element.getKey(), (key, group) -> {
       if (group == null) group = new ArrayList<RecordT>();
       group.append(element.getValue());
       return group;
   });
}
public void finishBundle(FinishBundleContext c) {
   for (KV<String, List<RecordT>> group : groups) {
       c.output(group);
   }
}
```

Local Group-By as Beam DoFns

```
// Simple way to get better grouping: Lather, rinse, repeat
PCollection<KV<String, List<RecordT>>> locallyGroupedRecords =
   records
       .apply("Locally group 1", ParDo.of(new LocalGroupBy()))
       .apply("Locally group 2", ParDo.of(new LocalGroupBy()))
       .apply("Locally group 3", ParDo.of(new LocalGroupBy()));
```

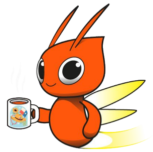


Gives more complete grouping per worker at the cost of more CPU time

- still better than a broad shuffle.

Can we group across DoFn threads as well?

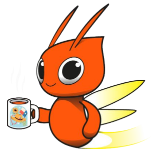
Local Group-By as a Beam DoFn



```
// All DoFns on a worker share a single groups map; here lie concurrency headaches...
static ConcurrentHashMap<String, List<RecordT>> groups = new ConcurrentHashMap();
static final Object mutex = new Object();
static int numActiveBundles = 0;

public void startBundle() {
   synchronized (mutex) {
       numActiveBundles++;
   }
}

public void processElement(@Element KV<String, RecordT> element, …) {
   // Same as before, add our element to the map.
}

public void finishBundle(FinishBundleContext c) {
   synchronized (mutex) {
       if (numActiveBundles-- > 0) {
           mutex.wait(); // DANGER ZONE: what if we never make progress here?
       } else {
           for (KV<String, List<RecordT>> group : groups) {
               c.output(group);
           }
           groups = new ConcurrentHashMap();
           mutex.notifyAll();
       }
   }
}
```

Better grouping, but much trickier code and dubious benefits

Local Group-By as a Spark Transform

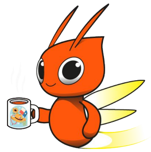


Spark provides an API for processing all of the elements of a single partition as a single iterable:

```
val groupedRecords = myRecords
 .mapPartitions(partitionIt: Iterator[(String, RecordT)]=> {
   val groups: HashMap[String, ArrayList[RecordT]]= new HashMap()
   partitionIt.forEach {
     case (key, record) => {
       val curGroup = groups.getOrElseUpdate(key, new ArrayList())
       curGroup.add(record)
     }
   }
   groups.toIterable
})
```

Can/should we add this API to Beam? Pros/Cons?

Local Group-By as a Spark Transform



Spark provides an API for processing all of the elements of a single partition as a single iterable:

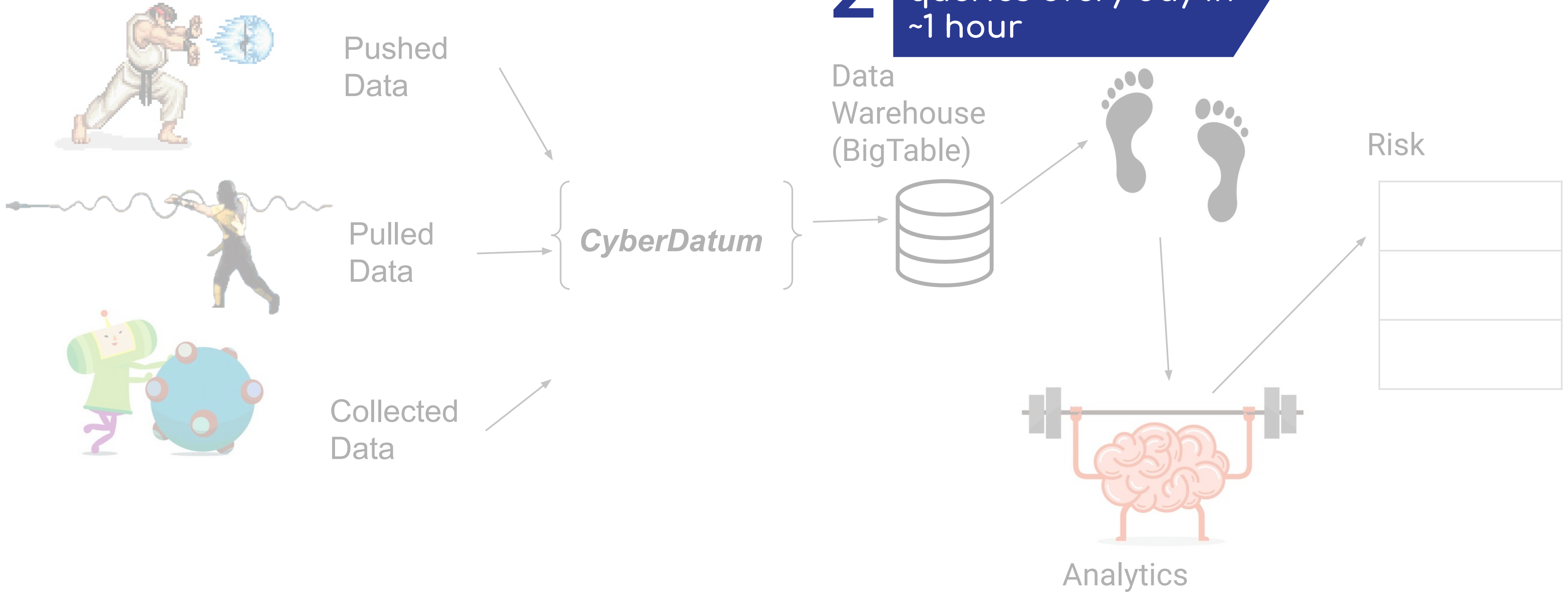
```
val groupedRecords = myRecords
 .mapPartitions(partitionIt: Iterator[(String, RecordT)]=> {
   val groups: HashMap[String, ArrayList[RecordT]]= new HashMap()      startBundle
   partitionIt.forEach {
     case (key, record) => {
       val curGroup = groups.getOrElseUpdate(key, new ArrayList())   processElement
       curGroup.add(record)
     }
   }
   groups.toIterable      finishBundle
})
```

Can/should we add this API to Beam? Pros/Cons?

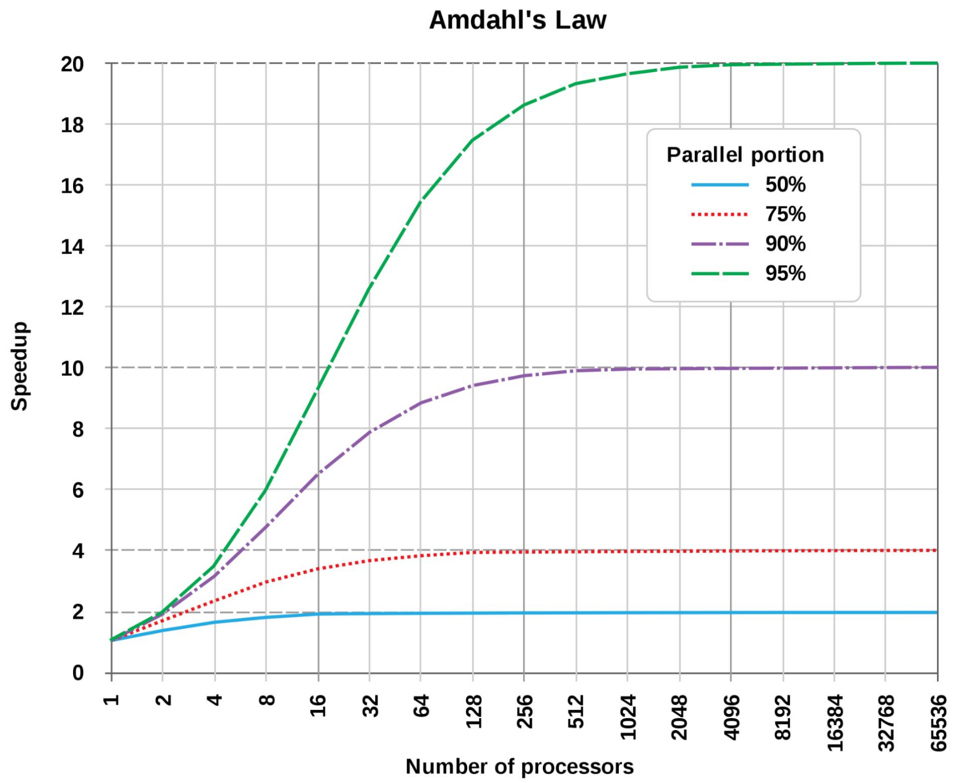
Running billions of queries every day in ~1 hour

Beam Challenges

2 Running billions of queries every day in ~1 hour



Amdahl's Law (0)



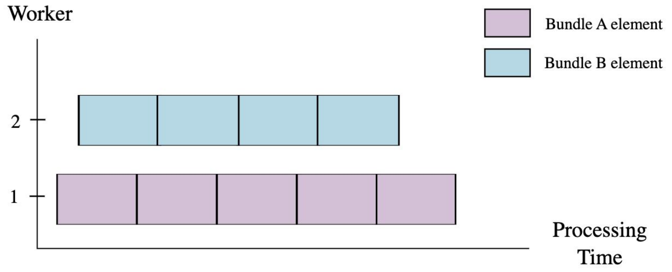


Figure 2: Two workers process the two bundles in parallel.

Inputs (1)

What's a CIDR?

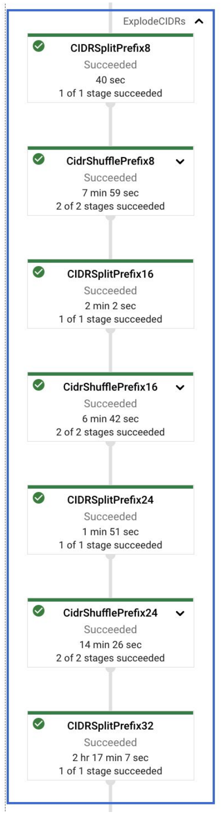
8.8.8.8/32 -> [8.8.8.8]

1.2.3.4/16 -> [1.2.0.0, 1.2.0.1, 1.2.0.2, ..., 1.2.255.255]

Inputs are compressed - don't want long tails on large ranges.

FanOut / Reshuffle at each of 8, 16, 24, 32 to distribute this

"decompression". We furthermore need to convolute wrt time.


ExplodeCIDRs

CIDRSplitPrefix8
Succeeded
40 sec
1 of 1 stage succeeded

CidrShufflePrefix8
Succeeded
7 min 59 sec
2 of 2 stages succeeded

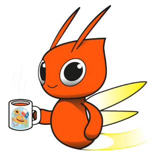
CIDRSplitPrefix16
Succeeded
2 min 2 sec
1 of 1 stage succeeded

CidrShufflePrefix16
Succeeded
6 min 42 sec
2 of 2 stages succeeded

CIDRSplitPrefix24
Succeeded
1 min 51 sec
1 of 1 stage succeeded

CidrShufflePrefix24
Succeeded
14 min 26 sec
2 of 2 stages succeeded

CIDRSplitPrefix32
Succeeded
2 hr 17 min 7 sec
1 of 1 stage succeeded



Naive Approach (2)

After exploding our CIDRs, we're left with:

~6.3 Billion Queries and hope for the best?

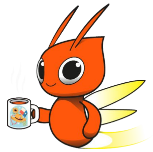
Translates to:

2048 Cores

~4-6 Hours of Runtime

Lots of OOMs / retries / *vacuous* queries (no results)

Wait a Minute… (3)



~6.3 Billion IPv4 addresses!?

- There's only ~3.7 Billion publicly addressable IP addresses
- Pigeonhole Principle?

GBK Confirms this… But that's still a lot of queries!

OK - queries are cut in half, but what about our OOMs/Retries/Vacuous Queries?

Problem Queries (4)

Vacuous Queries:

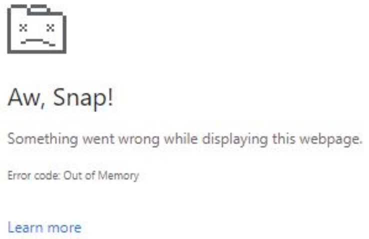
∃?　　->　　Nope

Still costs time to answer that question

OOMs:

∃?　　->　　Very Much **Yes**

Opposite problem - too much data per worker



Aw, Snap!

Something went wrong while displaying this webpage.

Error code: Out of Memory

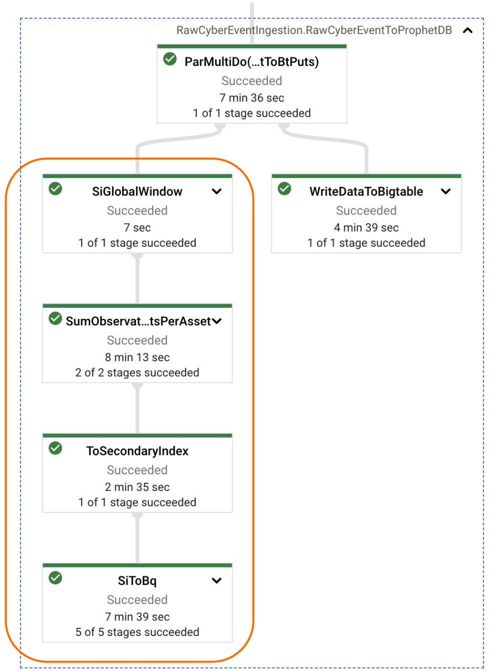
Learn more

Secondary Index (5)

```
SELECT ip, COUNT(*) FROM secondary_index
WHERE start_time <= timestamp
AND timestamp <= end_time GROUP BY ip
```

Now we know what IPs we **have data for**

and **how much data we have for each**.


RawCyberEventIngestion.RawCyberEventToProphetDB

ParMultiDo(...tToBtPuts)
Succeeded
7 min 36 sec
1 of 1 stage succeeded

SiGlobalWindow
Succeeded
7 sec
1 of 1 stage succeeded

WriteDataToBigtable
Succeeded
4 min 39 sec
1 of 1 stage succeeded

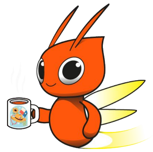
SumObservat...tsPerAsset
Succeeded
8 min 13 sec
2 of 2 stages succeeded

ToSecondaryIndex
Succeeded
2 min 35 sec
1 of 1 stage succeeded

SiToBq
Succeeded
7 min 39 sec
5 of 5 stages succeeded

Secondary Index is built by utilizing DAG structure of Beam - just tack on an additional operation to write IP/Timestamp to your favorite RDBMS as a side-effect of otherwise running your pipeline.

Do You Exist? (6)

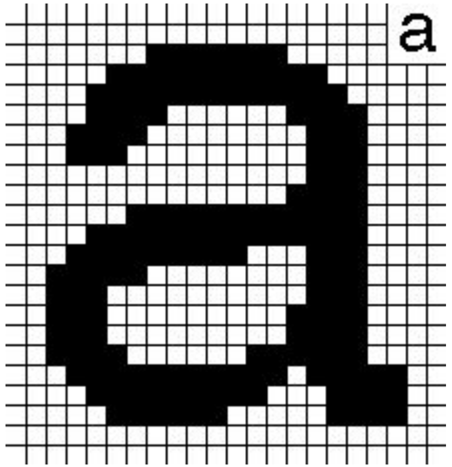


Bitmaps

- Naive: 2^32 - 1 = ~536MB
- RoaringBitmap: ~222 MB
- Computed via custom CombineFn

Broadcast via Side Input and do an In-memory Filter

- ~6.3 Billion -> ~1.5 Billion
- This is pre-GBK/Dedupe



Solving OOMs (7)

Not only do we have proof of existence, but we have **<IP, Count>**

Assume Uniform Distribution of IP over time range

We can now partition/split keyspace of a range accordingly

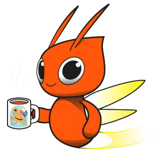
(1.2.3.4, [2022-04-01, 2022-05-01], 30)

-> ~30 of (1.2.3.4, [2022-04-**01**, 2022-**04-02**], 1),

(1.2.3.4, [2022-04-**02**, 2022-04-**03**], 1), …

(1.2.3.4, [2022-04-**30**, 2022-**05-01**], 1)

Slice N Dice (8)



Further split the queries according to Bigtable's sampleRowKeys()

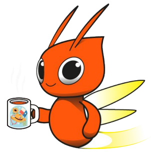
- Ensure queries are sympathetic to the underlying storage layer.
- Same process as before, but takes tablet-boundaries into account

BT RowKeys are 4096 bytes, lexicographically ordered.

We store IPv4 as Hex, to enable scanning not just single IPs, but CIDR ranges too.

After Bitmap/Split/Dedupe/GBK: **6.3B** -> **1.5B** -> **~450MM** Scans

Running the Queries (9)



Batch ~450MM Scans into collections of 256 each

~1.8MM "Scan Groups"

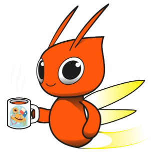
What happens when steps fused to scans fail part way through?

Entire step needs to be retried. Ouch!

Shuffle is your friend: "Fusion Break"

Reshuffle.viaRandomKey() will checkpoint you data preventing rescanning of Bigtable/External Service and recalculating scans.

MultiThreading! (10)



Beam model says you "don't *have* to think about this multithreading" but you may want to anyway.

Scio has some great examples if you speak Scala

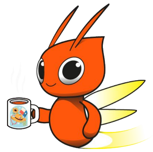
Think of DoFn's as individual Microservices

Beam handles networking/statefulset/message passing

What it doesn't do is benchmarking - you'll be spending a lot of time here.

Tip: Start with a fixed number of threads

End Results (11)



Cores:

2048 -> **348** cores (could go lower but we like the headroom)

Runtime:

~4-6 hours -> ~**1.5** hours (end-to-end)

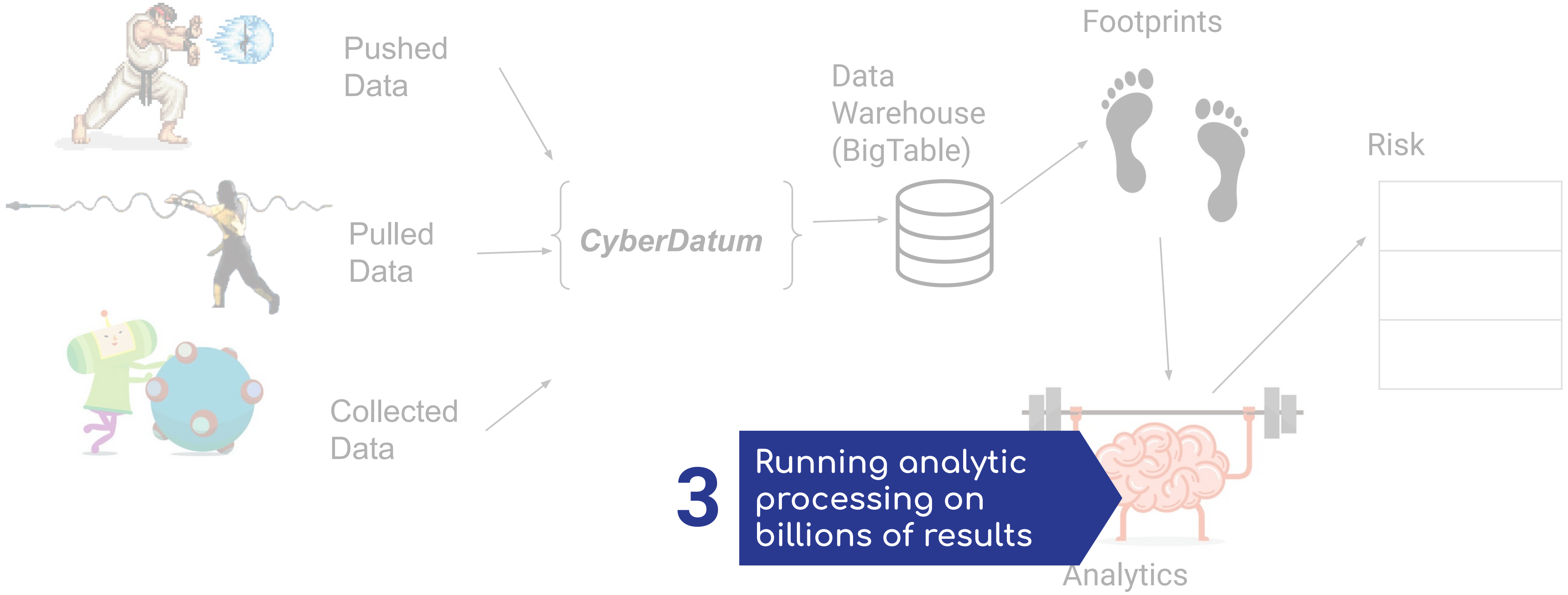
MGMT:

□ -> 🤑

Running analytic processing on billions of results

BEAM SUMMIT Austin, 2022

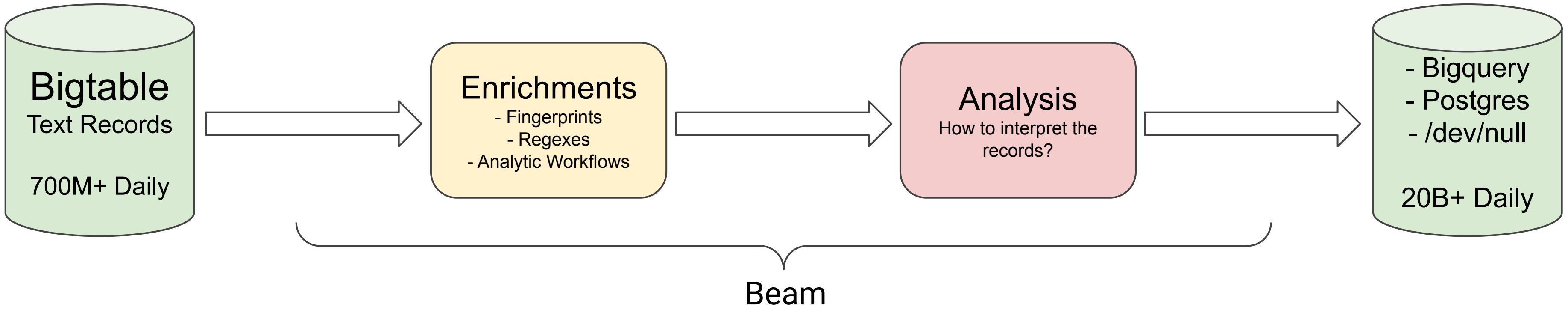
Beam Challenges



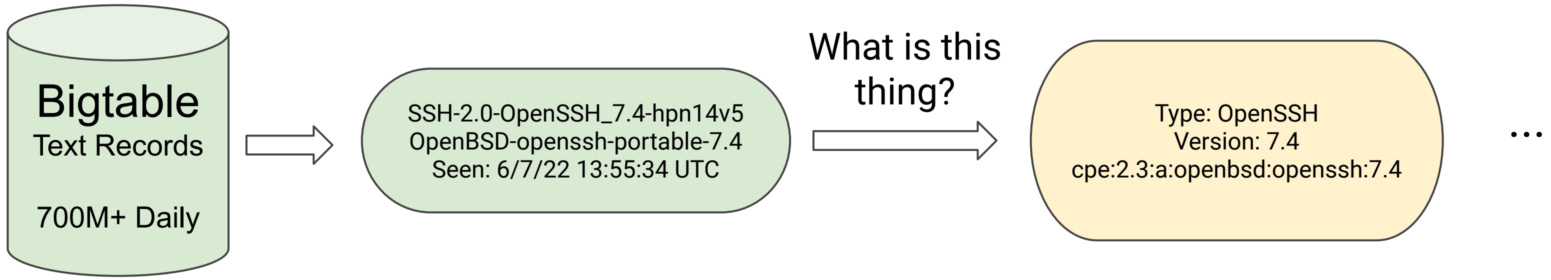
3 Running analytic processing on billions of results

Processing the Records

- Pull out observations that are attributed to our footprints
- Enrich observations with whatever the current threat landscape looks like
- Determine what these observations and enrichments are telling us
  - Do we even care about every single observation?
  - Can we determine who/what may be vulnerable? Who is running out of date code?

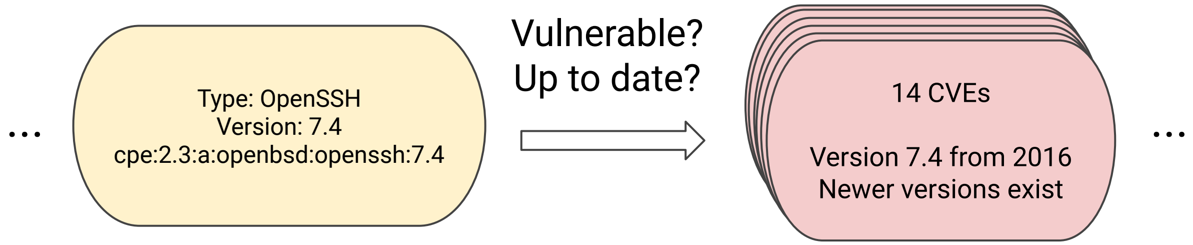


Enrichment


Bigtable
Text Records
700M+ Daily
SSH-2.0-OpenSSH_7.4-hpn14v5
OpenBSD-openssh-portable-7.4
Seen: 6/7/22 13:55:34 UTC
What is this thing?
Type: OpenSSH
Version: 7.4
cpe:2.3:a:openbsd:openssh:7.4
...

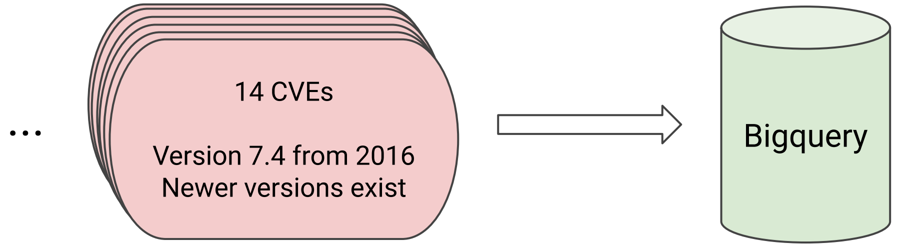
- Need to detect Software types and versions
  - Typical solution is plain regex - but that's too slow for this amount of data
  - Running a database of regexes against a database of outputs is an N^2 shuffle
  - Instead, we can run a compiled regex database scanner against each element (no shuffle!)
    - Maintained and curated to keep up with the changing landscape
- Problem
  - High amount of data overlap = lots of unnecessary cpu-heavy processing
  - GroupByKey to the rescue! We only need to process “unique records” once
    - 270M -> 6M records actually require processing

Analysis



- How to determine what to actually output? What is important?
  - Could use Side Input queries from BQ
    - Inefficient/Expensive to join records against large regex sets
    - Difficult to handle multi-CPE CVEs (Firefox.101 OR Firefox.102) AND (Ubuntu-20.04 OR Ubuntu-20.10)
  - Static prebuilt CVE mapping dictionaries
    - Combination of analyst fingerprinting and processed NIST CVE rulesets
    - Determine which CVEs apply to found CPEs
    - Determine how old the detected versions are
    - Very fast lookups in memory

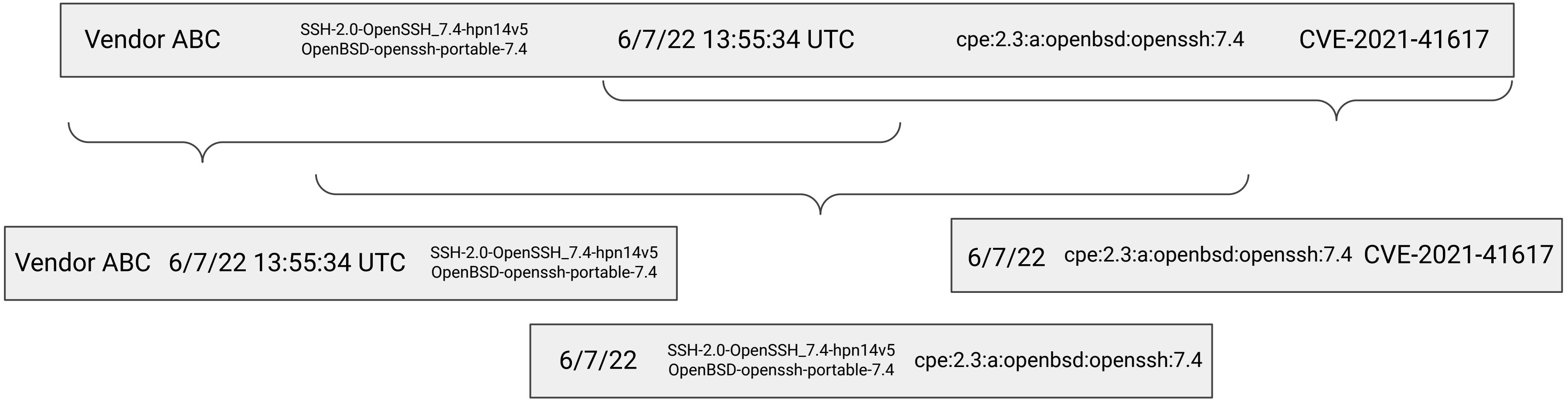
Output



- Finally we have output… **too much output**
  - 1 record => multiple software ids => potentially 100+ CVEs each
    - Extrapolated across millions of assets and footprints
  - Lots of redundant and costly output data!
    - Daily 20B records/6TB of highly redundant data
    - $$$ and downstream sadness
- GroupByKey to the rescue again

Normalization via GBK

Let's group by the record text, emit "normalized" records, and dedupe what we can.
This record… but billions more…

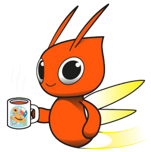


Vendor ABC SSH-2.0-OpenSSH_7.4-hpn14v5 OpenBSD-openssh-portable-7.4 6/7/22 13:55:34 UTC cpe:2.3:a:openbsd:openssh:7.4 CVE-2021-41617
Vendor ABC 6/7/22 13:55:34 UTC SSH-2.0-OpenSSH_7.4-hpn14v5 OpenBSD-openssh-portable-7.4
6/7/22 cpe:2.3:a:openbsd:openssh:7.4 CVE-2021-41617
6/7/22 SSH-2.0-OpenSSH_7.4-hpn14v5 OpenBSD-openssh-portable-7.4 cpe:2.3:a:openbsd:openssh:7.4

Using GBK and Beam Distinct transforms: 20B records (6TB) realized data => 3 datasets at 30GB total

- Also! The normalized data can be used as "ground truth" datasets now, all as a side effect of trying to save time and money

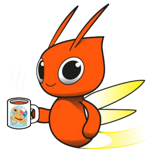
Wrap up



- **Before-Beam**: many disparate processes with different stacks
  - Anywhere from 1-10 hours to process individually
  - Complexity of maintenance
  - Limited to SQL or cross-platform chaos

- **The Beam Way** - all data processed within a single integrated codebase
  - 1hr process to generate the same outputs (and in cheaper/efficient ways)
  - Only one code base and architecture now :D
  - Ad-hoc analysis, end-to-end A/B testing

Beam Questions and Feature Requests



- Controlling bundle sizes? (GroupIntoBatches incurs shuffle)
- Operations on a bundle or all bundles local to a single worker?
- Splittable DoFn for querying BigTable where results may have long tails?
- Fusion break – can we do it without a shuffle?

Alfredo Gimenez
alfredo.gimenez@bluevoyant.com

Adam Najman
adam.najman@bluevoyant.com

Tucker Leavitt
tucker.leavitt@bluevoyant.com

Tyler Flach
tyler.flach@bluevoyant.com

BlueVoyant https://bluevoyant.com

Always looking for good data engineers

BEAM SUMMIT Austin, 2022