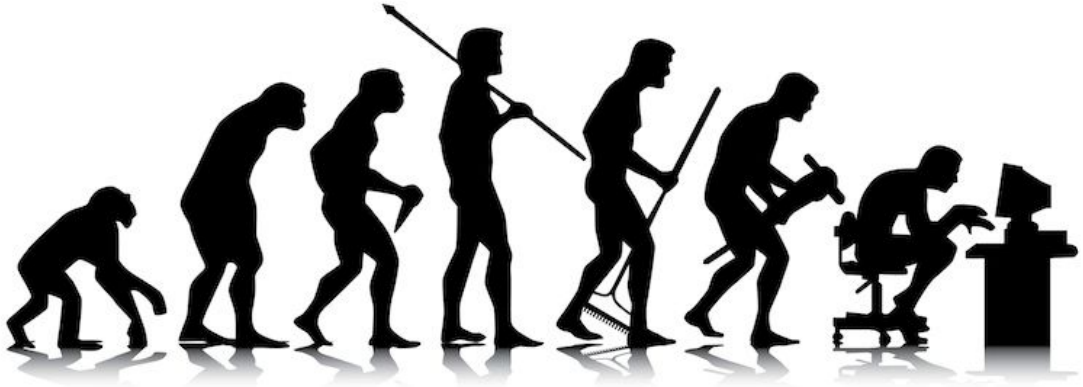
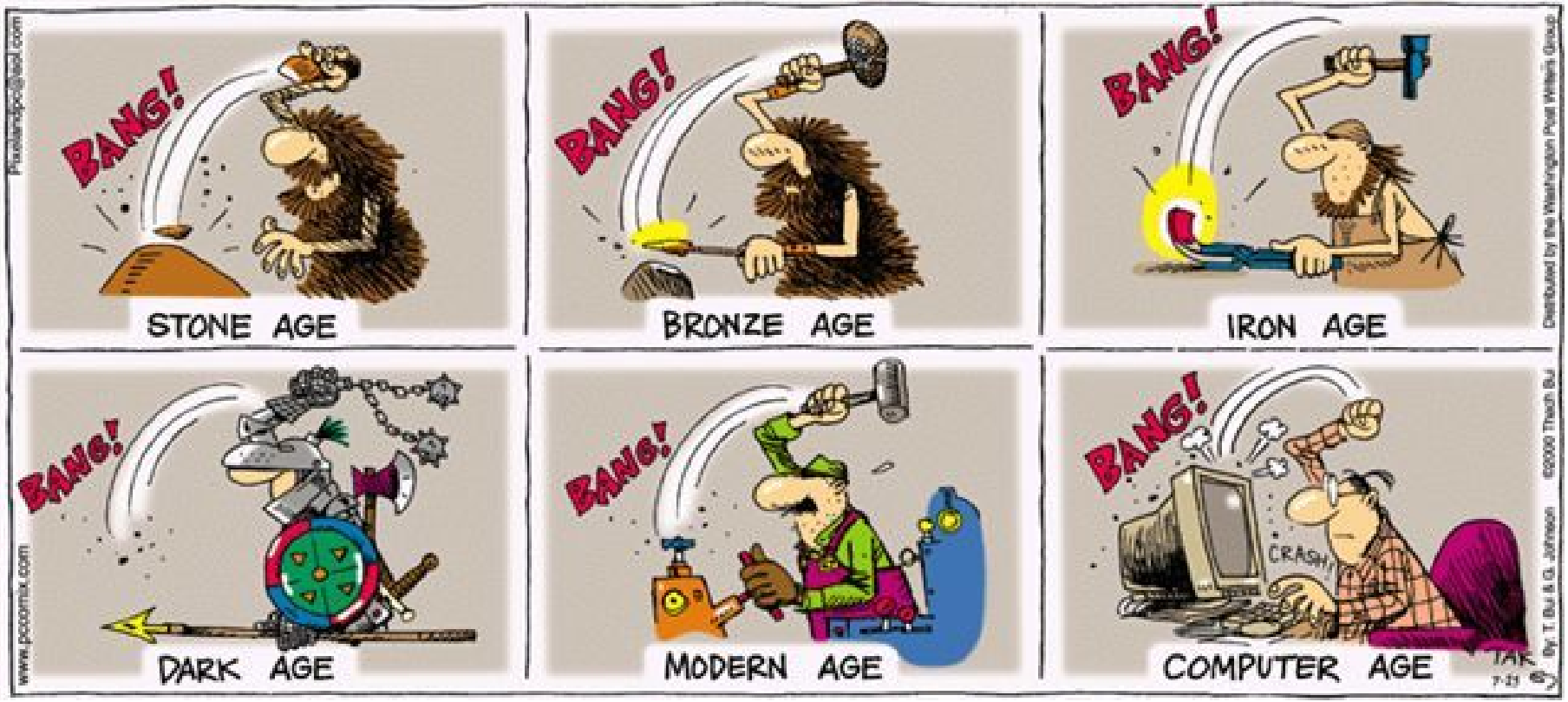
Survival of the Fittest - Streaming Architectures

by Michael Hansen




BANG!
STONE AGE
BANG!
BRONZE AGE
BANG!
IRON AGE
BANG!
DARK AGE
BANG!
MODERN AGE
BANG!
CRASH!
COMPUTER AGE
TAK
7-23
Distributed by the Washington Post Writers Group
©2000 Thach Bui
By T. Bui & G. Johnson
www.pcomix.com
Pantandpc@aol.com

Today's Talk

Is:

- Case study on adapting and evolving a streaming ecosystem - *with a focus on the subtleties that most frameworks won't solve for you*
- Evolving your Streaming stack requires diplomacy and salesmanship
- Keeping the focus on your use cases and why ***you*** do streaming in the first place.
- Importance of automation and self-service

Is ***not***:

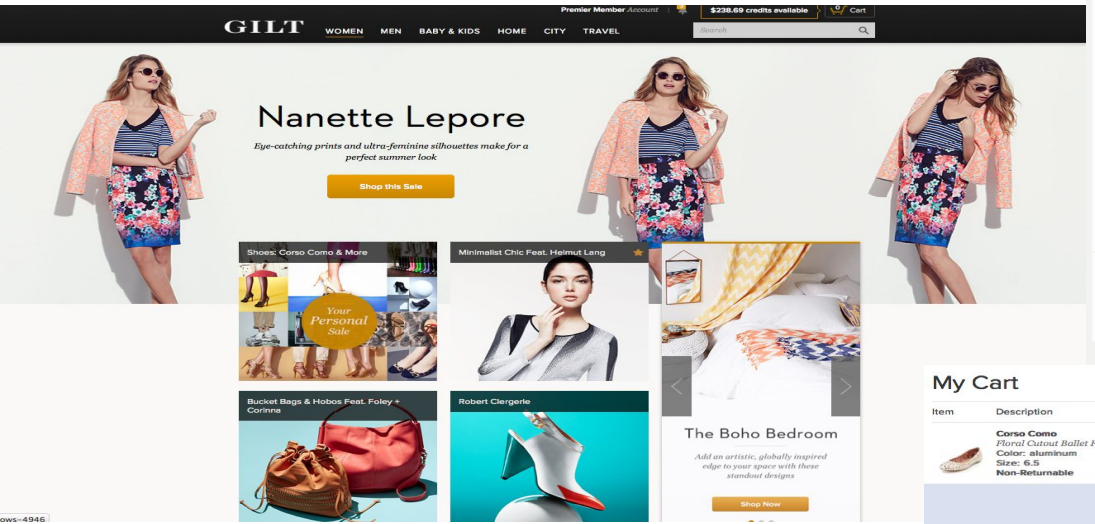
- An extensive comparison between current and past streaming frameworks
- About our evolution towards the "perfect" streaming architecture and solution (evolution does not produce perfection!)
- Archeology

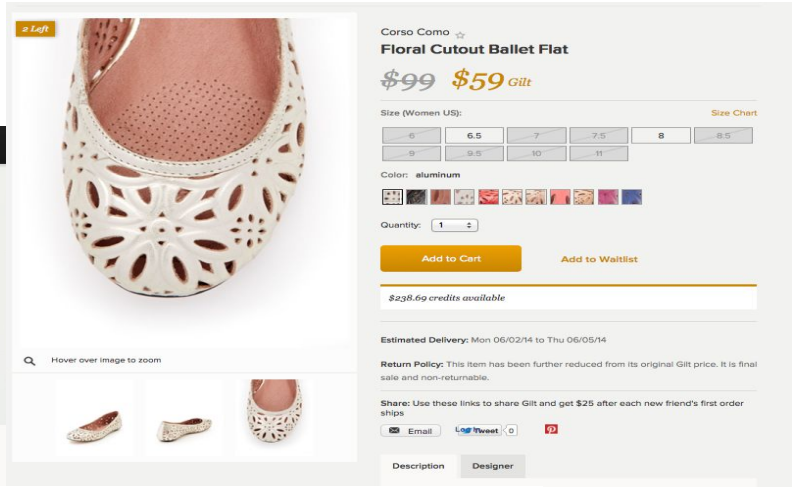
"Perfect is the enemy of good"
- Voltaire

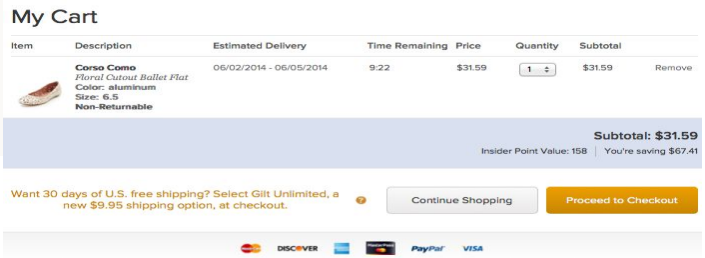
Gilt.com

A **Hudson's Bay Company** Division

What Is Gilt?





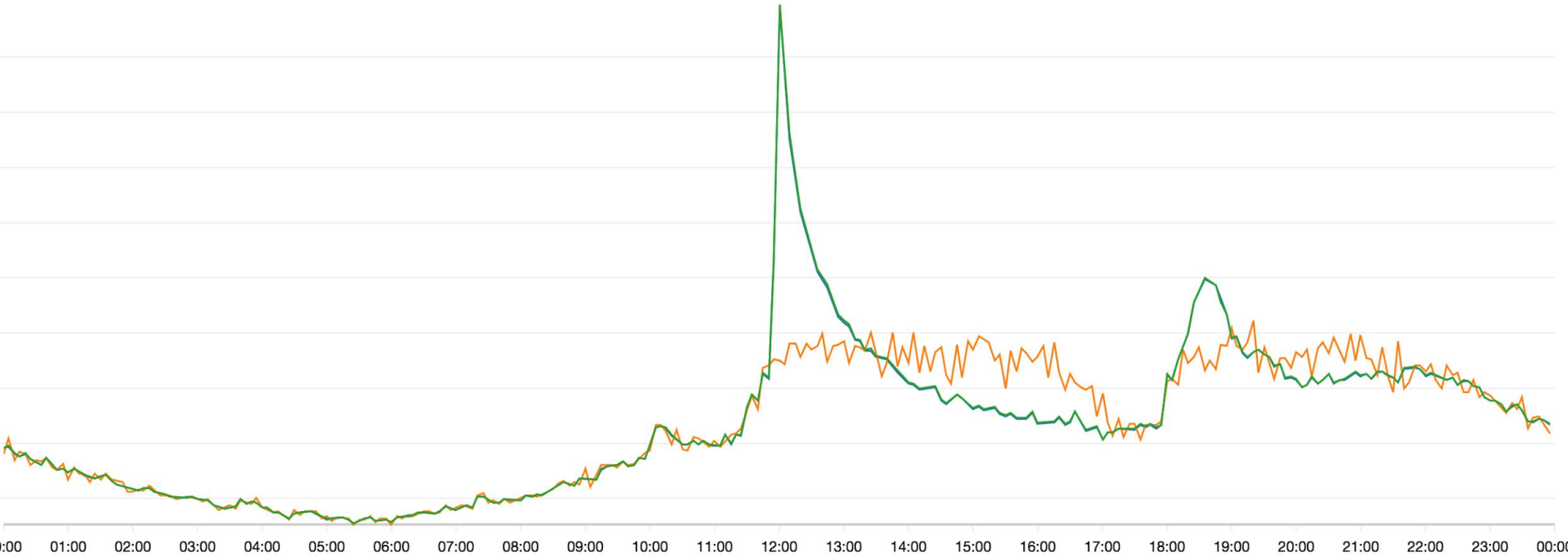


- An innovative online shopping destination with products by today's top designer labels, with 30-70% off retail
- New deals daily, released at noon – sense of urgency

Tech Philosophy

- Autonomous Tech Teams
- Voluntary adoption
- LOSA (lot's of small apps)
- Micro-service cosmos

Typical Traffic Pattern on Gilt.com

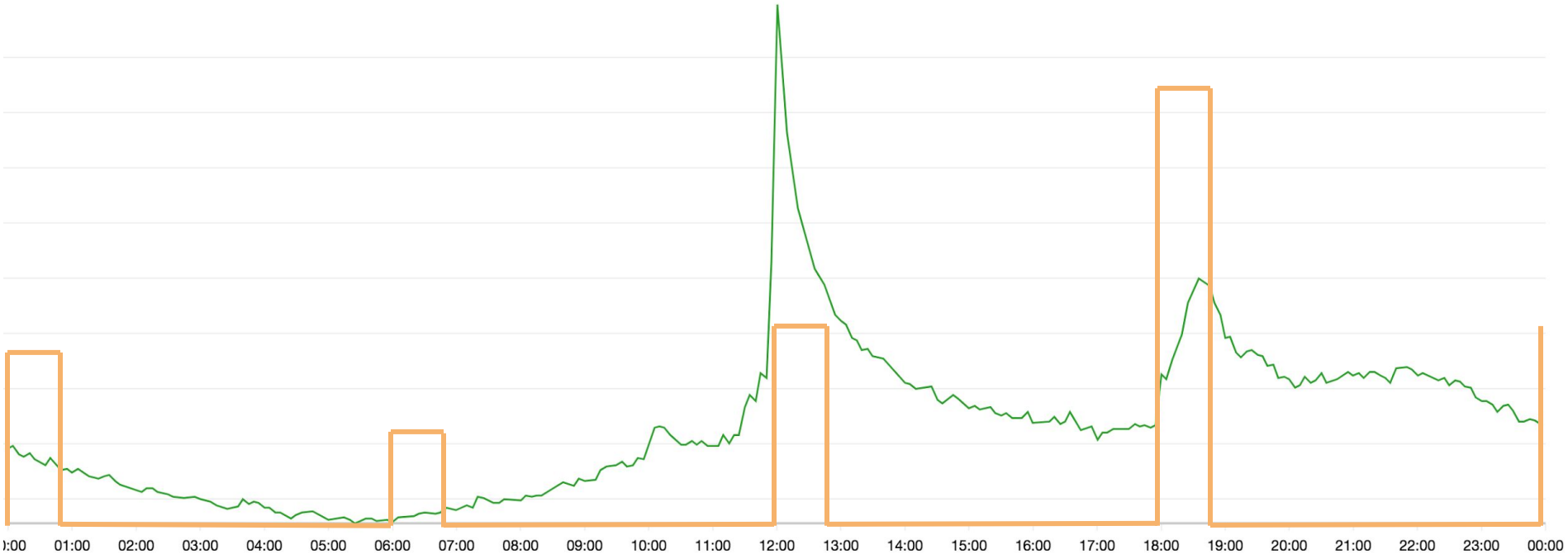


Batch vs. Streaming

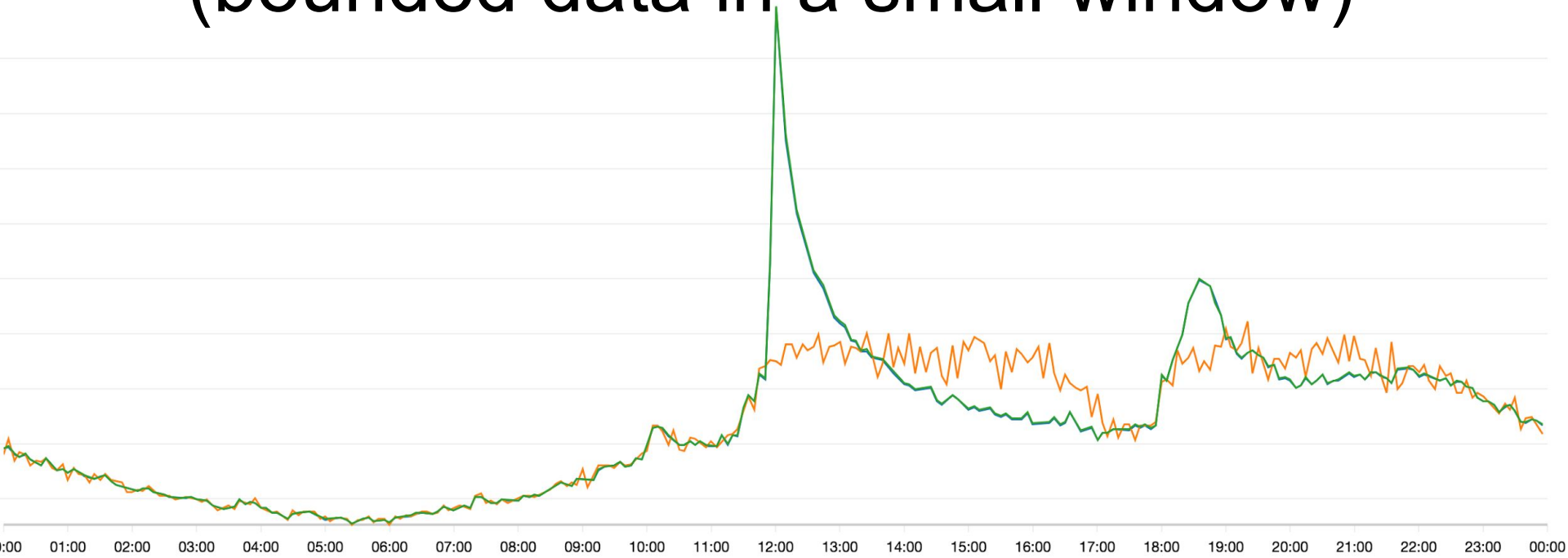
Is batch just a special case of streaming?

Recommended reads:
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
https://data-artisans.com/blog/batch-is-a-special-case-of-streaming

4 Batch Cycles per Day
(bounded data in a large window)



Micro-batch
(bounded data in a small window)



Real-time
(bounded data in a tiny window)

Gilt.com Use Cases

Must Have

- At-least-once semantics
- Metrics & Analytics
  - Near-real-time (<15 minutes)
- React-to-event
  - Real-time (< few second)
- Automation of data delivery (including all DevOps activity)
- Self-service for producers and consumers, alike
- “Bad” data should be rejected before entering the system
- High elasticity (you saw the traffic pattern!)

Nice-to-have, but ***not*** required

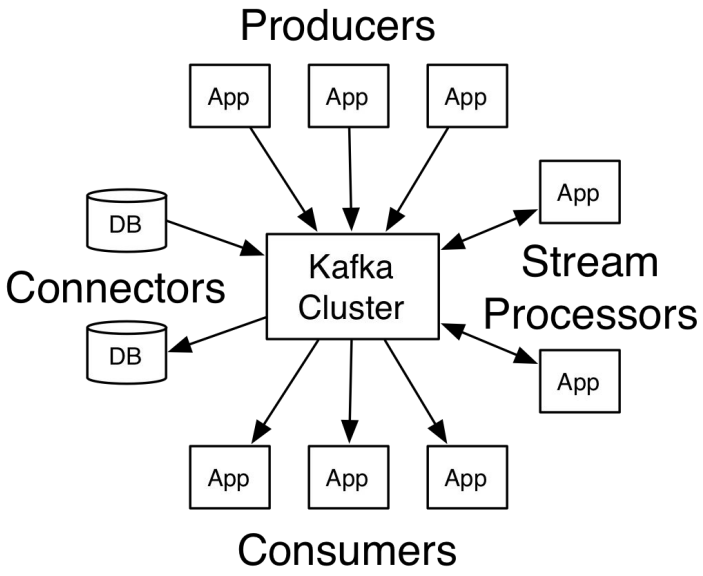
- Exactly-once semantics
- Real-time Metrics & Analysis
- Complex calculations or processing directly on streams in real-time

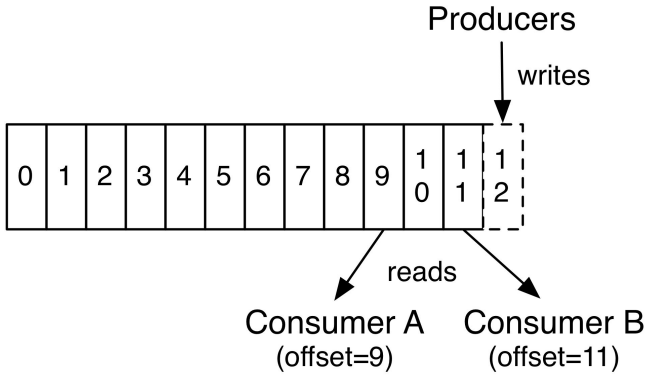
Stone Age

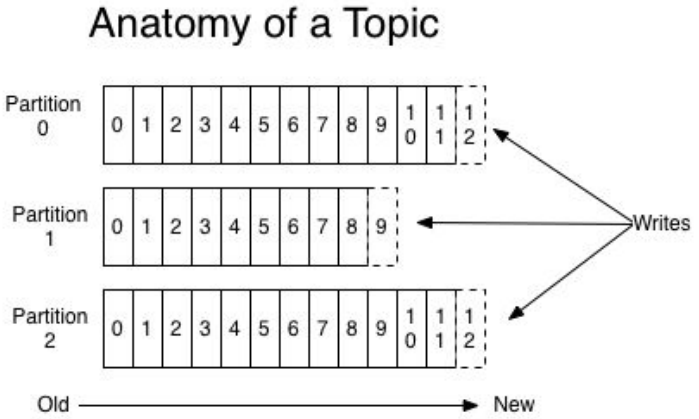
A progression of behavioral and cultural characteristics and changes, including the use of wild and domestic crops and of domesticated animals.

Brief Intro to Kafka

- Organized by Topics
- N partitions per Topic
- *Producers* - writers
- *Consumers* - readers
- Data *offset* controlled by *Consumer*







The Stone Age Data Streaming

- Apache logs - mobile and web
- `tail -f` on logs into a Kafka from a Docker container

```
#!/bin/bash

KAFKA_BROKERS="kafka.at.gilt.com:9092"

tail --lines=0 --follow=name --retry --quiet /var/log/httpd/access_log
/var/log/httpd/ssl_access_log | /opt/gilt/lib-kafka-console-producer/bin/produce --topic
log4gilt.clickstream.raw --batch-size 200 --kafka-brokers ${KAFKA_BROKERS}
```

Where `bin/produce` is:

```
exec $(dirname $0)/gjava com.giltgroupe.kafka.console.producer.Main "$@"
```

Stone Age Stream Consumption

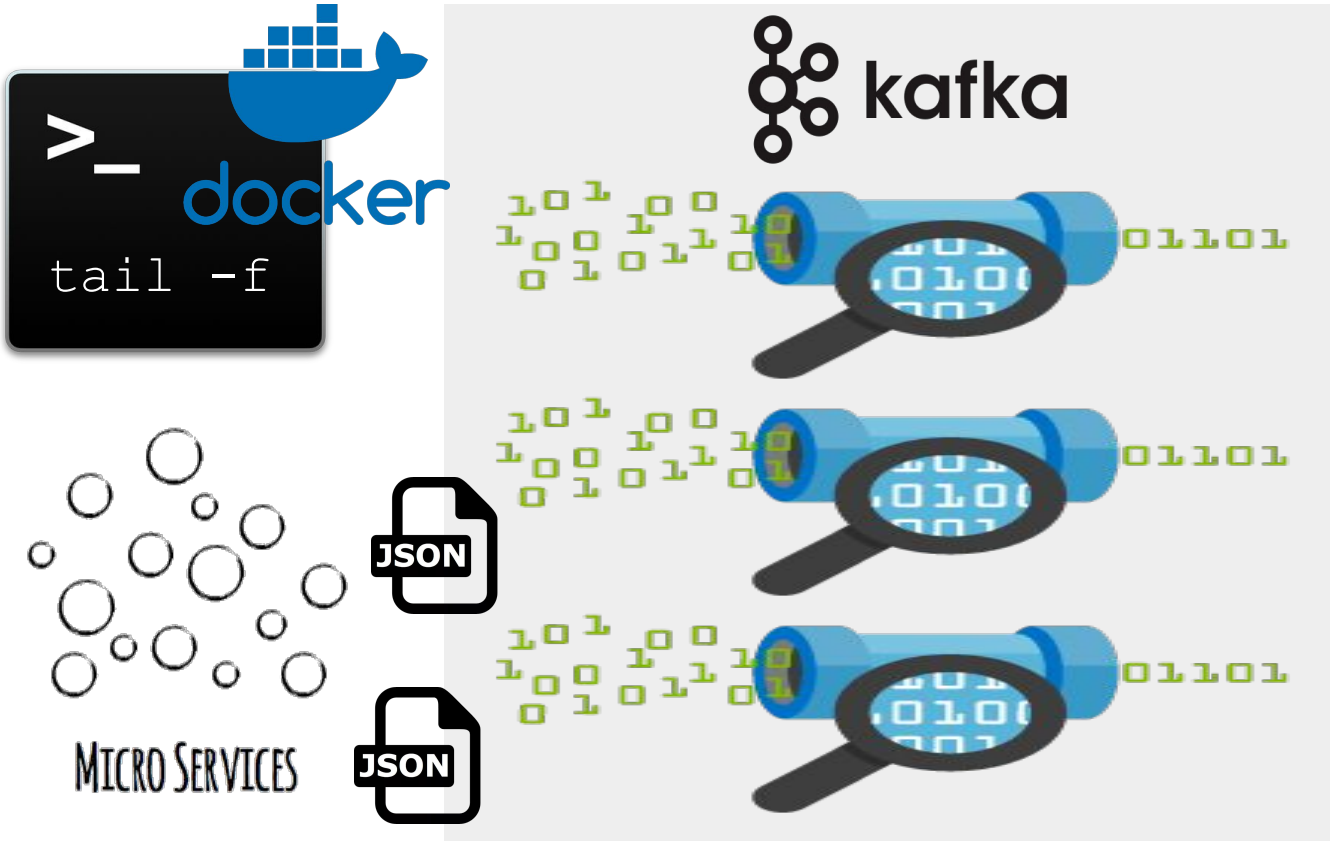
- Using convoluted SQL/MR (TeraData Aster) and
- Kafka offset logging in the Data Warehouse
- Parsing of event data from URL parameters and oddball name-value pairs - different in EVERY single Stream!

```
when position('&' in substr(substring(substr(substr(utmr, position('http://' in utmr)+6), length(split_part(substr(utmr, position('http://' in utmr)+6),'/',2))+2) from
E'[?|&]{1}'||m.keyword_2||'=.*$'),2)) > 1
    then replace(replace(split_part(substr(substr(substring(substr(substr(utmr, position('http://' in utmr)+6), length(split_part(substr(utmr, position('http://' in utmr)+6),'/',2))+2) from
E'[?|&]{1}'||m.keyword_2||'=.*$'),2), 1, position('&' in substr(substring(substr(substr(utmr, position('http://' in utmr)+6), length(split_part(substr(utmr, position('http://' in
utmr)+6),'/',2))+2) from E'[?|&]{1}'||m.keyword_2||'=.*$'),2))-1), '=', 2), '%20', ' '), '%2520', ' ')
    when position('&' in substr(substring(substr(substr(utmr, position('http://' in utmr)+6), length(split_part(substr(utmr, position('http://' in utmr)+6),'/',2))+2) from
E'[?|&]{1}'||m.keyword_2||'=.*$'),2)) <= 1
    then replace(replace(split_part(substr(substring(substr(substr(utmr, position('http://' in utmr)+6), length(split_part(substr(utmr, position('http://' in utmr)+6),'/',2))+2) from
E'[?|&]{1}'||m.keyword_2||'=.*$'),2), '=', 2), '%20', ' '), '%2520', ' ')
end as search_keyword
, m2.keyword_1 as social_referral_site(lower(cv3),'-1') = 'logged in' then 1 else 0 end as is_past_reg_wall
, case
    when position('&' in substr(substring(url from E'[?|&]{1}utm_medium=.*$'),2)) > 1
    then lower(replace(replace(split_part(substr(substr(substring(url from E'[?|&]{1}utm_medium=.*$'),2), 1, position('&' in substr(substring(url from E'[?|&]{1}utm_medium=.*$'),2))-1), '=',
2), '%20', ' '), '%2520', ' '))
    when position('&' in substr(substring(page_referrer_page_path from E'[?|&]{1}utm_medium=.*$'),2)) > 1
    then lower(replace(replace(split_part(substr(substring(page_referrer_page_path from E'[?|&]{1}utm_medium=.*$'),2), 1, position('&' in substr(substring(page_referrer_page_path from
E'[?|&]{1}utm_medium=.*$'),2))-1), '=', 2), '%20', ' '), '%2520', ' '))
    when length(substring(url from E'[?|&]{1}utm_campaign=.*$')) > 0 and search_engine is not null then 'cpc'::varchar
    when search_engine is not null then 'organic'::varchar
    when position('&' in substr(substring(page_referrer_page_path from E'[?|&]{1}utm_medium=.*$'),2)) <= 1
    then lower(replace(replace(split_part(substr(substring(page_referrer_page_path from E'[?|&]{1}utm_medium=.*$'),2), '=', 2), '%20', ' '), '%2520', ' '))
    when length(page_referrer_host_name) > 0
, case
    when position('&' in substr(substring(url from E'[?|&]{1}utm_source=.*$'),2)) > 1
    then lower(replace(replace(split_part(substr(substr(substring(url from E'[?|&]{1}utm_source=.*$'),2), 1, position('&' in substr(substring(url from E'[?|&]{1}utm_source=.*$'),2))-1), '=',
2), '%20', ' '), '%2520', ' '))
    when position('&' in substr(substring(page_referrer_page_path from E'[?|&]{1}utm_source=.*$'),2)) > 1
    then lower(replace(replace(split_part(substr(substring(page_referrer_page_path from E'[?|&]{1}utm_source=.*$'),2), 1, position('&' in substr(substring(page_referrer_page_path from
E'[?|&]{1}utm_source=.*$'),2))-1), '=', 2), '%20', ' '), '%2520', ' '))
    when position('&' in substr(substring(url from E'[?|&]{1}utm_source=.*$'),2)) <= 1
    then lower(replace(replace(split_part(substr(substring(url from E'[?|&]{1}utm_source=.*$'),2), '=', 2), '%20', ' '), '%2520', ' '))
    when position('&' in substr(substring(page_referrer_page_path from E'[?|&]{1}utm_source=.*$'),2)) <= 1
    then lower(replace(replace(split_part(substr(substring(page_referrer_page_path from E'[?|&]{1}utm_source=.*$'),2), '=', 2), '%20', ' '), '%2520', ' '))
    else search_engine
end as source
, case
    when position('&' in substr(substring(url from E'[?|&]{1}utm_term=.*$'),2)) > 1
    then lower(replace(replace(split_part(substr(substr(substring(url from E'[?|&]{1}utm_term=.*$'),2), 1, position('&' in substr(substring(url from '[?|&]{1}utm_term=.*$'),2))-1), '=', 2),
'%20', ' '), '%2520', ' '))
    when position('&' in substr(substring(url from E'[?|&]{1}utm_term=.*$'),2)) <= 1
    then lower(replace(replace(split_part(substr(substring(url from E'[?|&]{1}utm_term=.*$'),2), '=', 2), '%20', ' '), '%2520', ' '))
    when search_engine is not null then search_keyword
    when position('&' in substr(substring(page_referrer_page_path from E'[?|&]{1}utm_term=.*$'),2)) > 1
    then lower(replace(replace(split_part(substr(substring(page_referrer_page_path from E'[?|&]{1}utm_term=.*$'),2), 1, position('&' in substr(substring(page_referrer_page_path from
E'[?|&]{1}utm_term=.*$'),2))-1), '=', 2), '%20', ' '), '%2520', ' '))
    when position('&' in substr(substring(page_referrer_page_path from E'[?|&]{1}utm_term=.*$'),2)) <= 1
    then lower(replace(replace(split_part(substr(substring(page_referrer_page_path from E'[?|&]{1}utm_term=.*$'),2), '=', 2), '%20', ' '), '%2520', ' '))
end as keyword
```

Bronze Age

Characterized by the (over) use of bronze, proto-writing, and other early features of urban civilization.

Early Streaming Architecture



Everybody loves JSON!

- Services stream JSON directly to Kafka topics
- Consuming straight out of Kafka with Aster SQL/MR into a “hard-coded” JSON parser
- Changing JSON structure/data blows up ELT pipelines
- Not scalable in terms of engineering man-hours

```
begin;
create temp table messages distribute by hash(kafka_offset) as
select * from kafka_consumer (
  on (
    select
      kafka_topic,
      kafka_partition,
      max(end_offset) + 1 as kafka_offset
    from audit.kafka_transformation_log
    where transname = 'discounts' and status = 'end'
    group by kafka_topic, kafka_partition
  )
  partition by 1
  messages(10000000) -- Setting to arbitrarily large number
);

insert into raw_file.discounts
select
  kafka_partition,
  kafka_offset,
  json as kafka_payload,
  guid::uuid as guid,
  to_timestamp(updated_at, 'Dy, DD Mon YYYY HH24:MI:SS') as updated_at
from json_parser (
  on (select kafka_partition, kafka_offset, kafka_payload as json
      from messages)
  fields('updated_at', 'discount.guid as guid')
);
end:
```

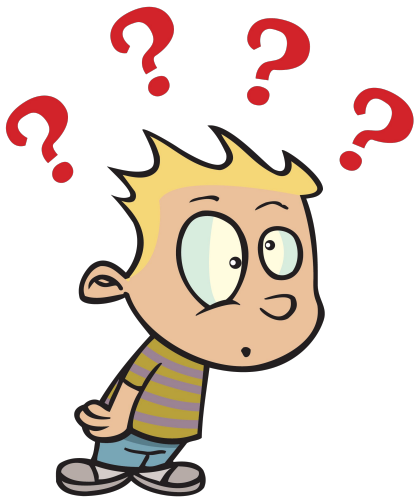
Early State of Affairs

Life's peachy for the data producers

The data consumer is screaming in agony

Why do we stream data in the first place?

- To learn from our data and use that knowledge to improve our business in a timely manner
- We want to react to things happening in our business in a “timely” manner (“timely” may not mean “real-time”!)
- But how?
  - We make it easy to track
  - We make it easy to access
  - We make it easy to measure
  - We make it easy to analyze
- Only one in four, so *not* there yet...

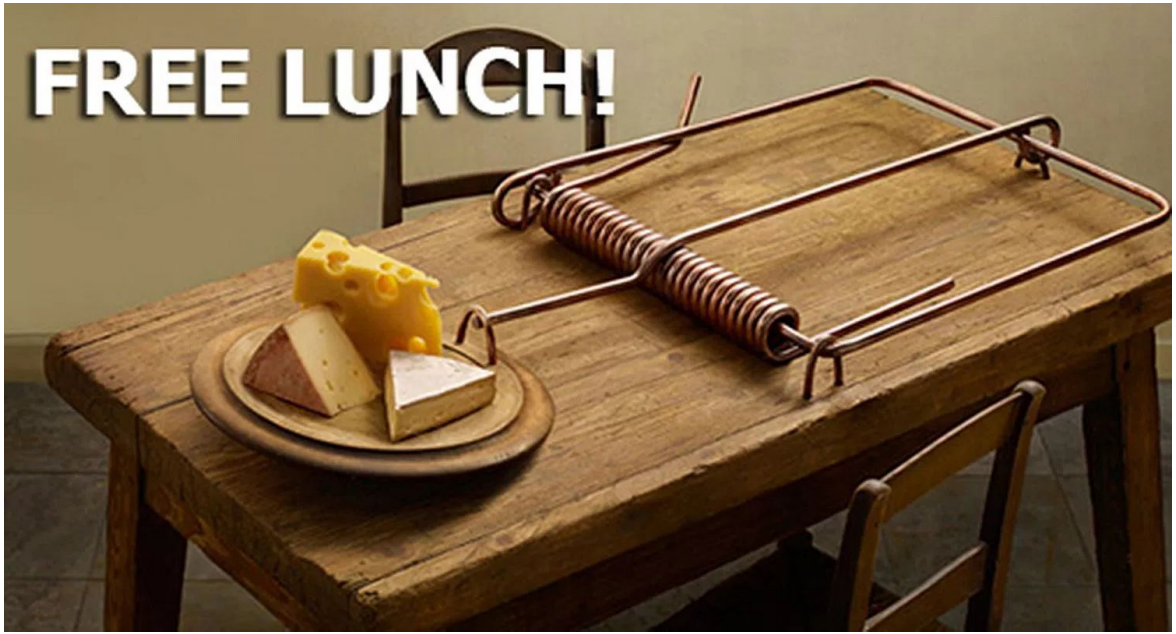


How to make it better for our data customer?

- Data-about-data
- Programmatically-well-defined structure
- A contract with guarantees about the content structure
- Standardized data (de)serialization.

There’s no free lunch!

This translates into more work and discipline for the data producers



Diplomacy

“Diplomacy is the art of letting somebody else have your way”
- David Frost

- ***Data producers*** want to be agile and productive, building whatever it is that they build, minimize complexity, and add great value to our business in a timely manner - *fair enough!*
- ***Data consumers*** wants to be able to tap into these data sources with little or no effort, so they can focus on the analysis or machine learning, which is how they add value to the business - *fair enough!*
- How do we all ***get aligned*** all make lots of money together!?

The Dawn of Civilization

- It's time to exit tribalism and enter civilization
- Use diplomacy to introduce ***rule of law***:
  - Data structures and schematization
  - (De)serialization of stream data
- Surprisingly, or maybe not, it was much harder to convince all Tech Teams to schematize and serialize data, than it was to pick a particular protocol to use
- Lobby and work your diplomacy/salesmanship across your organization - even if you're in a top-down hierarchical organization!

Roman Age

Rise of modern government, law, politics, engineering, art, literature, architecture, technology, warfare, religion, language and society.

Brief Intro to Avro - Our New *Rule-of-Law*

- A serialization protocol with rich data structures
- Schematized with clear evolution rules - backward and forward compatibility
- Persistent files contains schema in header
- Supports RPC
- Generic serialization - dynamic
- Specific serialization (w/ code generation) - type-safe
- Supported by several languages
- See: http://avro.apache.org/docs/current/spec.html

Naming and Standards

- Namespace indicate producer or group of producers (micro services)
- Ownership of these services may change, so we tie the naming to the services, as opposed to, the teams
- Avro did not cover all our needs out-of-box (1.7.x back then)

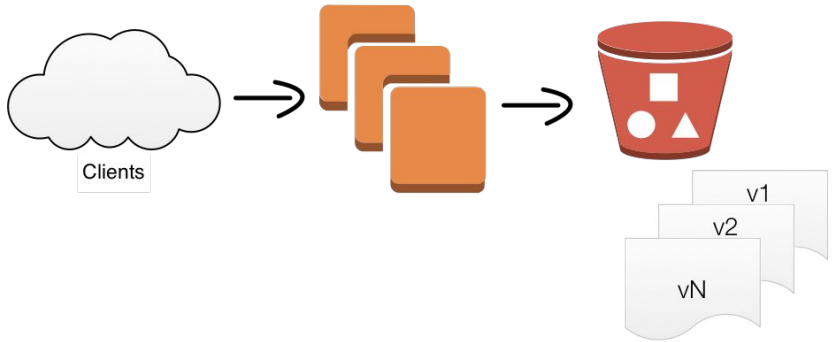
- Extend (de)serialization and standard schemas:
  - Money
  - UUID
  - Datetime
  - Decimal
- With 1.8.x Avro introduced “logical types” which takes care of datetime stuff, as well as, Decimal (with precision and scale)

Schema Management

- Things to tackle are schema evolution and versioning
  - Avro *“fingerprints”* (SHA-256)
- 9 out of 10 of our use cases only need backward compatibility
- What about breaking compatibility?
  - We decided to inject ***vX*** between namespace and name, e.g., com.gilt.tapstream.***v1***.AddToCartEvent
- Another microservice to help with this, that would be great...

Svc-avro-schema-registry - the Keymaster

- A Play/Scala/Akka service
- Manage all registration and lookup of Avro schemas
- Handles versioning with Avro fingerprints (SHA-256)
- Checks for compatibility (and rejects if not)
- Does not support deletes (why?)
- Backed by an S3 versioned bucket



Schema Registry Api

Method and Path	Description
GET /subjects	Gets all subjects which have a schema registered.
PUT /subjects/:subject/schemas	Registers a new schema version for the specified subject. The body should contain a text representation of the schema (not a SchemaDetails). A new schema which violates evolution rules will result in a 409-Conflict response.
GET /subjects/:subject/schemas/:fingerprint	Gets the schema with the specified fingerprint
GET /subjects/:subject/latestSchema	Gets the latest schema for the specified subject

Injection of `TimeUuid` (Avro alone is not enough)

- Deduplication of data - micro-batch - data warehouse (at-least-once semantic needs help!)
- Build-in timestamp
- Guarantees 10,000 unique UUIDs per MAC address per millisecond
- But, *not* persistent in case of producer failure

```
{
  "type" : "record",
  "name" : "ProductPageViewedEvent",
  "namespace" : "com.gilt.clicksteam.v1",
  "fields" : [{
    "name" : "uuid",
    "type" : {
      "type" : "fixed",
      "name" : "UUID",
      "namespace" : "gfc.avro",
      "size" : 16
    }
  }, {
    "name" : "productId",
    "type" : "long"
  }, {
    "name" : "saleId",
    "type" : [ "null", "long" ],
    "default" : null
  }]
}
```

`TimeUuid` - Sorting and Sequence

- Used for sorting data for time series, analytics, and in ELT processes
- Extracting exact sequence of events (e.g., clickstream events)

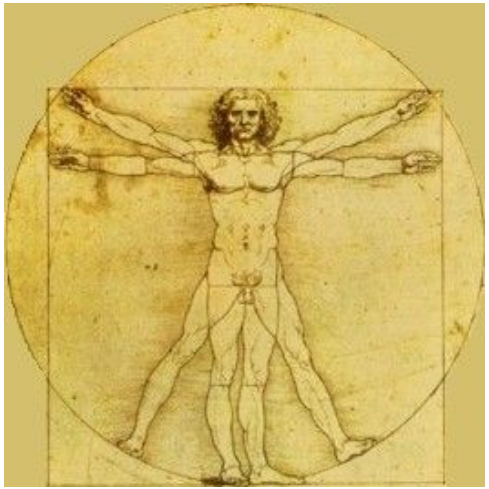
```scala
import java.util.{UUID, Random}

/** Based on http://www.ietf.org/rfc/rfc4122.txt */
object TimeUuid {
 private[this] val clockSeqAndNode = buildClockSeqAndNode()
 def apply(): UUID = new UUID(buildTime(Clock.time()), clockSeqAndNode)

 def apply(timeInMillis: Long): UUID =  new UUID(buildTime(convertToNanos(timeInMillis)), clockSeqAndNode)

 private def convertToNanos(timeInMillis: Long): Long = (timeInMillis - Clock.StartEpoch) * 10000

 private def buildTime(time: Long): Long = {
   var msb: Long = 0L
   msb |= (0x00000000ffffffffL & time) << 32
   msb |= (0x0000ffff00000000L & time) >>> 16
   msb |= (0x0fff000000000000L & time) >>> 48
   msb |= 0x0000000000001000L //Version 1 Uuid
   msb
 }

 private def buildClockSeqAndNode(): Long = {
   val clock: Long = new Random(System.currentTimeMillis).nextLong
   var lsb: Long = 0
   lsb |= (clock & 0x0000000000003FFFL) << 48 // clock sequence (14 bits)
   lsb |= 0x8000000000000000L // variant (2 bits)
   lsb |= Node.id // 6 bytes
   lsb
 }
}
```

See details or clone @
https://github.com/gilt/gfc-timeuuid

Anatomy of Avro Events

[H F I N G E P R I N T _..._ A V R O B Y T E S _..._]

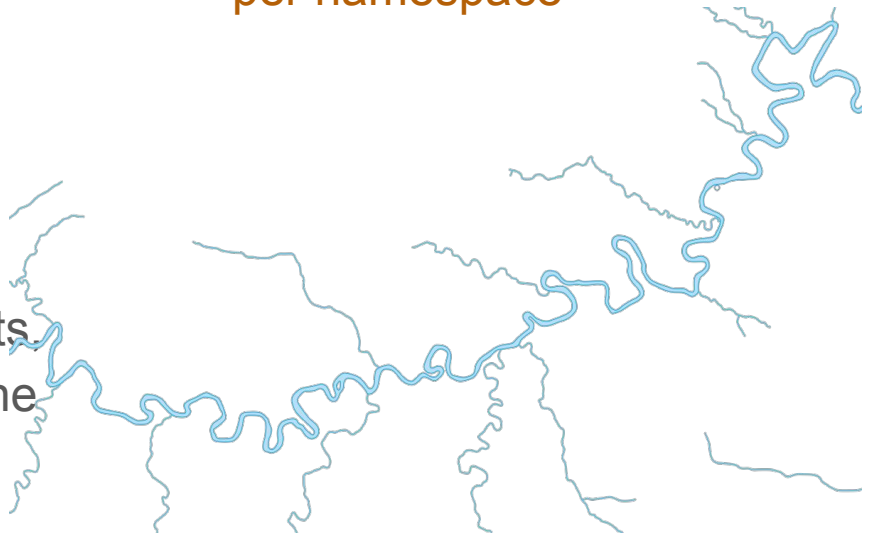
Position 0: header (0x81)
Position 1-32: fingerprint for schema (SHA-256)
Position 33-N: Avro record serialized to byte array



Many Small Streams or One Large River?

- At one point or another you need to decide how to organize your streams, along the ***spectrum*** of:
  - ***One stream per data type*** - simple write and read, but complex DevOps
  - A ***single stream for all data types*** - easy DevOps, but complex read
- Currently we are at the extreme of *one stream per data type*
- Including schema “fingerprints” on events give you the option of moving towards the other extreme of this spectrum

- In retrospect we should have chosen a middle ground of, say, one stream per namespace



Dark Age

Stagnant in terms of real technological and scientific progress.

Tested out Hadoop and Hive as our Data Lake for all Streaming

Too embarrassing to talk about...

Renaissance

The cultural bridge between the Dark Ages and modern history.

The Cloud

- The time we decided to scrap Hadoop and Hive, coincided with an overall company decision to move to the Cloud, in our case AWS
- S3 to the rescue - no need for an HDFS cluster!
- Replacing Kafka with Kinesis

Intro to AWS Kinesis

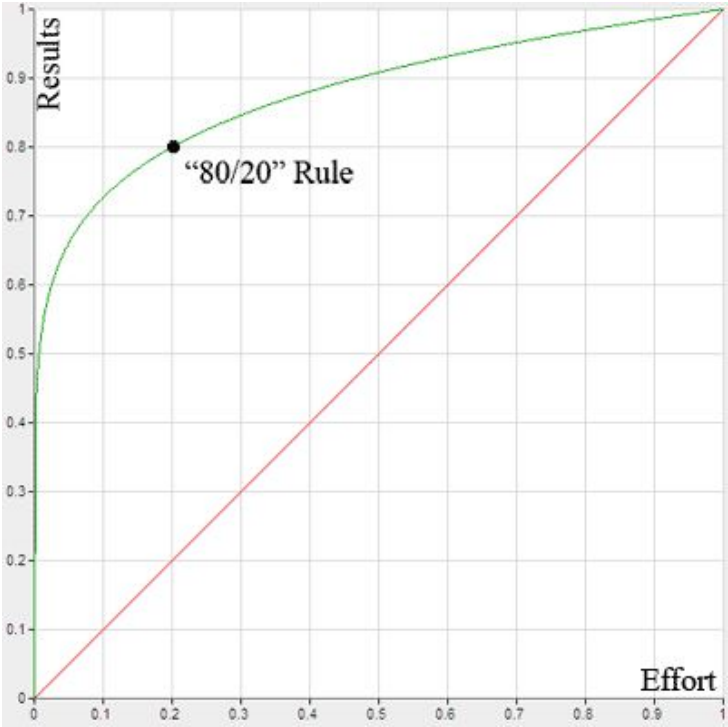
- Organized by *Streams*
- Each stream can have N *shards*
- To group data by shards, a *partition key* is required (we use our *TimeUuid*)
- Uses *sequence numbers*, analogous to Kafka's *offsets*.
- Some of the limits:
  - The maximum payload size 1 MB
  - *Describe Stream* - max 10 per second
  - *Get Records* can retrieve up to 10 MB of data
  - 5 *reads* per second, up to a max total rate of 2 MB per second
  - 1,000 records writes per second for writes, up to a max of 1 MB per second

Switching to a Cloud-based Streaming Platform

- From Kafka to Kinesis - works similarly in principle, but *very* different temperament
- Back-pressure - when limits are hit, Kinesis jumps into “throttling mode” which further exacerbate backpressure
- With healthy overprovisioning on Kafka, backpressure was trivial, just resend if bounced
- Kinesis throttle is *melodramatic*, so we added:
  - Hook in autoscaling for your Kinesis streams
  - Coded a robust exponential backoff strategy on clients

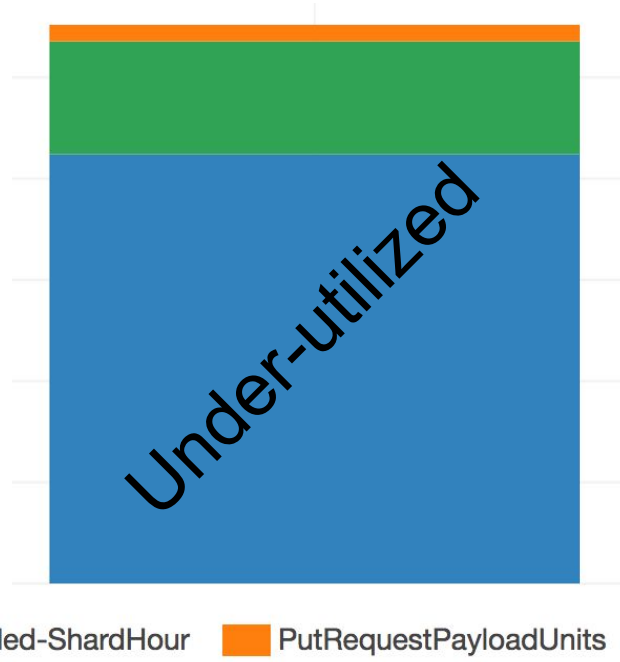
Creeping Cloud Costs

- Over-provision Kafka was relatively cheap, Kinesis is not!
- Pareto Principle: 20% of you data *types* accounts for 80% of your *volume*.
  - Not a problem in Kafka as we provision the overall cluster
  - Problem on Kinesis low-volume streams are under-utilized
- We had a couple of options:
  - Switch from many streams to one river
  - Implement Kinesis auto-scaling
- We went with the Kinesis autoscale option, for now, but need to consolidate streams as well



Kinesis Stream Cost Pattern

- The majority of a Kinesis Stream cost is in “shardhour” regardless of utilization rate!
- Example 250 records/sec @ 20 Kb (658.8 million PUT Payload Units per month)
  - 5 shards ~$54/month
  - PUT Payload ~$10/month
- This is ***not*** to say Kafka is always more economical, as it has significant DevOps and admin costs that Kinesis does not have
- Know your streaming patterns & volumes!



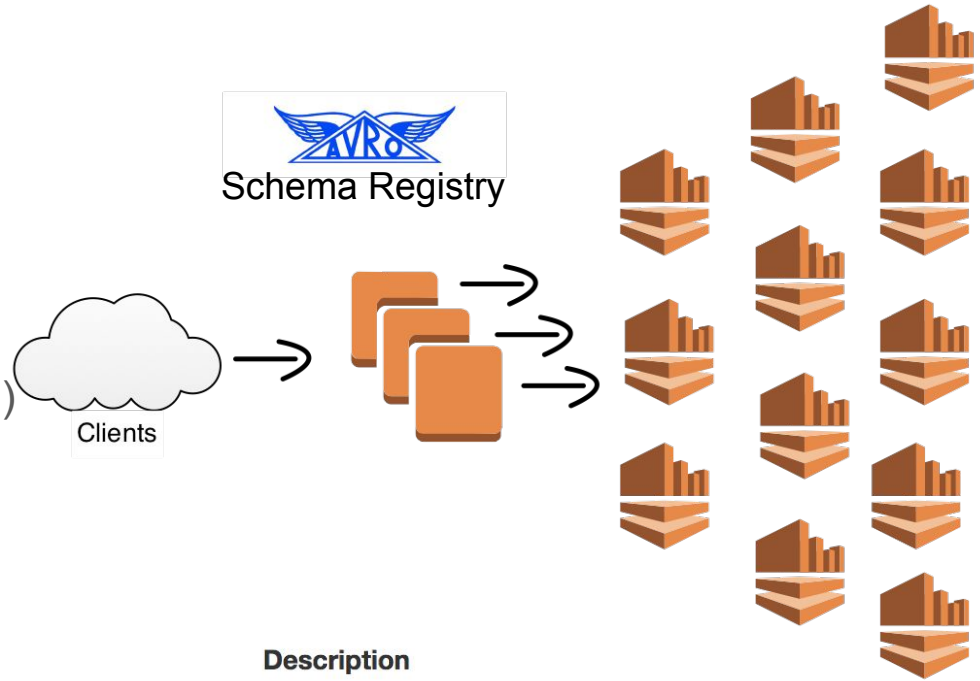
The Revolt

- Data producers do not want to use Avro, they want their JSON
- To stake off a revolt, we created a service, *svc-event*, to accept JSON and Avro events alike:
  - Compromise - JSON events still need an Avro schema in the registry
  - JSON events are converted to Avro
  - Avro is prefixed with its schema fingerprint
  - Batch of JSON events, an array of JSONs
- *svc-event* is in essence a “dumb waiter” passing on events to a streaming platform, such as Kafka or Kinesis

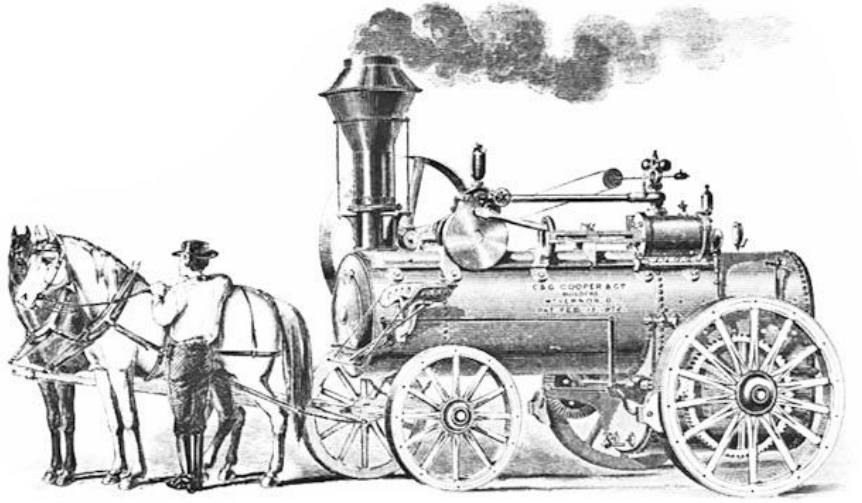
Svc-event - the Gatekeeper

- A Play/Scala/Akka service
- Accepts
  - JSON Payload
  - Avro Payloads
- Rejects payload if structure does not match the schema in store for `:streamId` (streamId = schema name)
- Back-pressure handling
- Fail “bad” data at the entrance
- Check for timeuuid presence

```
[HFINGEPRINT_..._
AVROBYTES_..._]
```



Method and Path		Description
PUT	/streams/:streamId/events/:guid	Takes a single event of type T
POST	/streams/:streamId/events	Takes a series of event of type T



Industrial Age

Characterized chiefly by the replacement of hand tools with power-driven machines and by the concentration of industry in large establishments.

Client Code Generation - Apidoc.me

- Code generation for clients - self-service for producers
  - svc-event
  - Avro schema registry
- Support for
  - Scala
  - Go client
  - Android client
  - Node
  - Play
  - Http4
  - Ruby client
- Learn more @ http://www.apidoc.me

api.json format

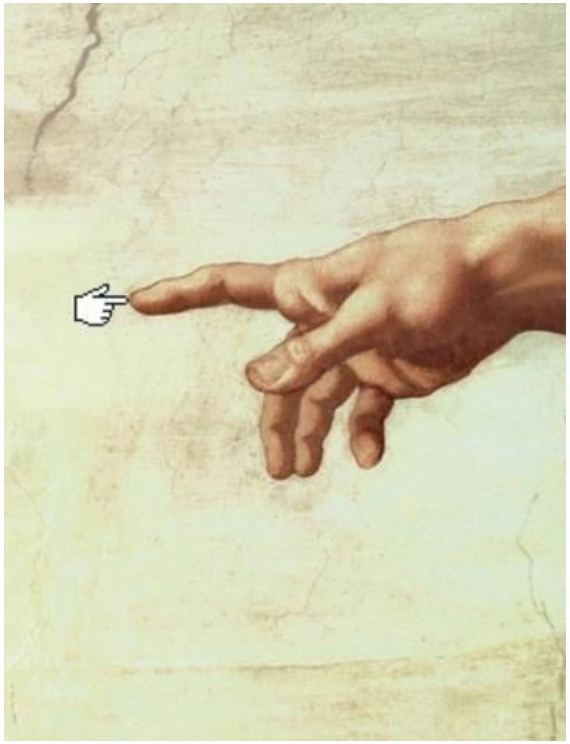
A schema is represented in JSON as a JSON object of the form:

```
{
  "name": string,
  "apidoc": JSON Object of Apidoc (optional),
  "info": JSON Object of Info (optional),
  "namespace": string (optional),
  "base_url": string (optional),
  "description": string (optional),
  "imports": JSON Array of Import (optional),
  "headers": JSON Array of Header (optional),
  "enums": JSON Object of Enum (optional),
  "models": JSON Object of Model (optional),
  "unions": JSON Object of Union (optional),
  "resources": JSON Object of Resource (optional),
  "attributes": JSON Array of Attribute (optional)
}
```

Better Schema Tooling

- The standard org.apache.avro.SchemaCompatibility library is pretty useless for medium and large schemas, as it simply throws back the entire schema JSON for each version, no hints as to which field is causing trouble.
- Created recursive “diff” detection to report on specific fields, and the reason, causing the overall schema to fail compatibility.
- Tools to upload entire Avro schema IDL Protocols, often used in source controls as are much more concise than their JSON-counterparts

Digital Age



What are we currently working on?

In the Works

- Tested Kinesis Analytics, but not exactly what we need:
  - Nice SQL interface
  - Feature rich with respect to time-series analytics
  - Only supports JSON and CSV, no extension hooks for binary formats
  - Does not scale well w.r.t. automatic DevOps and source code control
  - Does not feel mature yet
- Back to Spark Streaming using v2.x with Data Frames everywhere
  - Streaming and batch processes can share transformation code
  - Works great with the Data Store we are building on S3/Spark/Parquet
  - Streamline ELT across streams, files, and database replicas, alike
- Port *svc-event* from Akka Actors to Functional Streams for Scala (FS2), should give a nice performance boost -> less ec2 instances -> less $$$

Better Monitoring

- Cannot always rely on the timestamp assigned in `TimeUuid` as the source times may be incorrect or out of sync
- Need to inject a timestamp to incoming payloads in *svc-event* to precisely trace counts within any time series
- Ability to automate hook-ins for “unusual” pattern detection on our streams (easier said than done!)

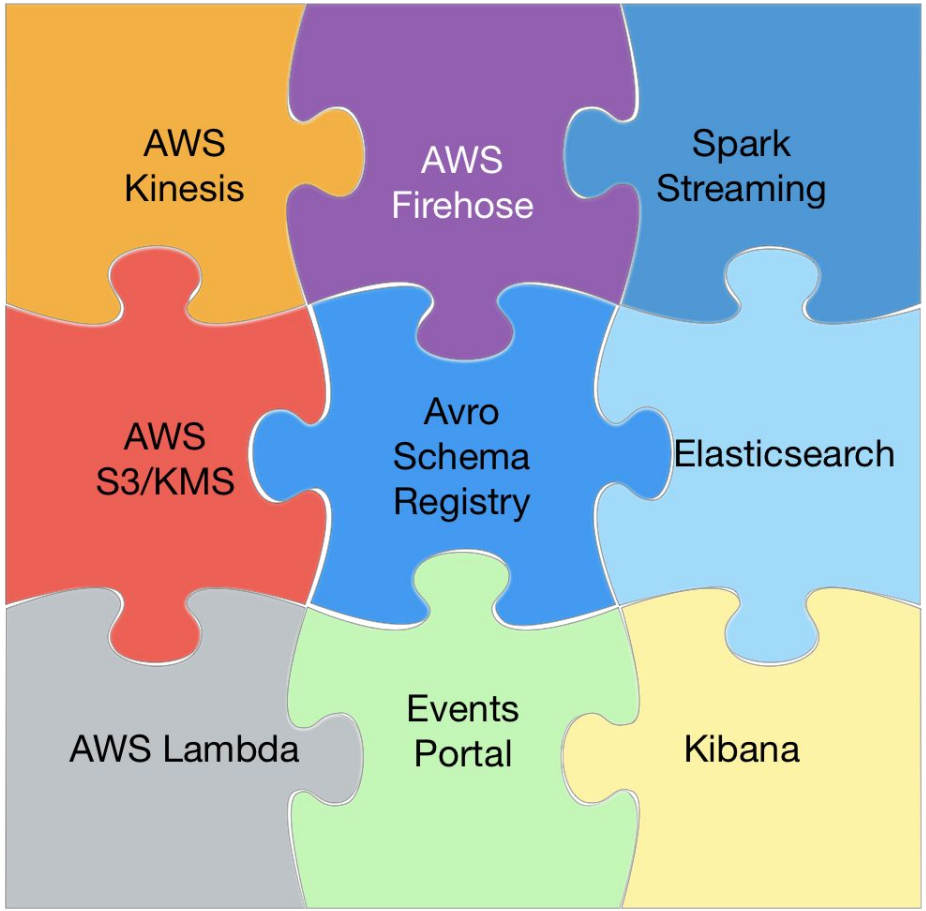
The Future

"It's hard to make predictions - especially about the future."

- Robert Storm Petersen

Next Evolution Step?

- Apache Flink, possibly?
- Apache Apex, no thanks!
- As a philosophical stance, we find it anti-agile to go all-in on a single framework or platform - platforms comes and goes, and switching is painful
- Find it more organic/agile to build a set of replaceable components

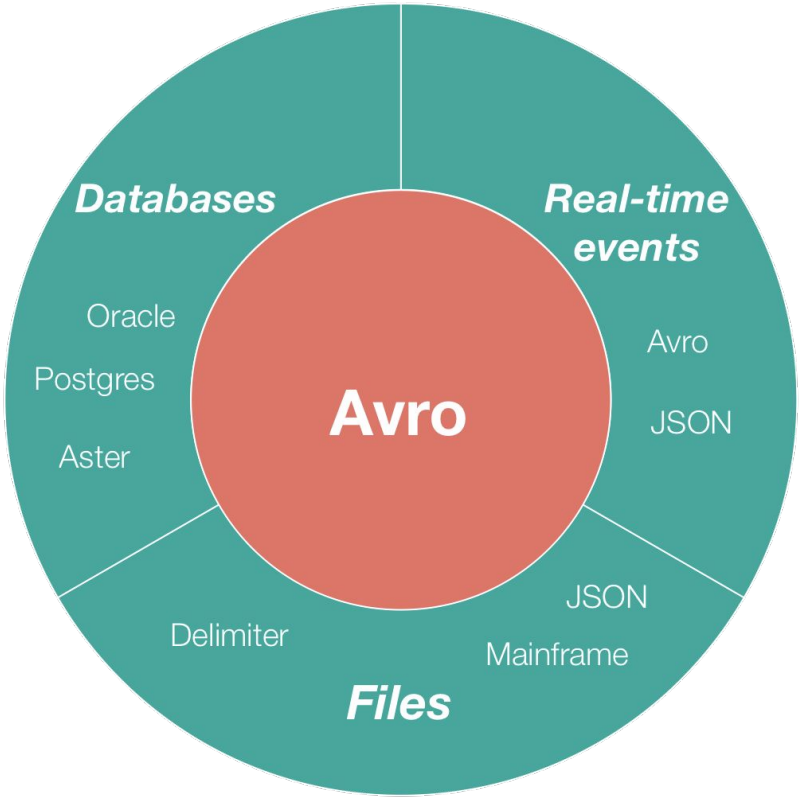


Bonus

Sometimes Evolution Creates Interesting Mutations

“Good” Genetic Mutations

- Avro led to other great usages around automation in our file batch and database replication process.
- Everything is now streamlined around Avro in a way that probably would not have happened if we had implemented a end-to-end (bloated) framework.
- Avro has become the Lingua Franca for our data and its processing automation.



Unintended Consequences

- A micro-service blasting “batch-like” payloads to streams

tech.gilt.com mhansen@gilt.com @hbcdigital