Distributed Tracing @ Jet


jet

Gina Maini
Senior Platform Engineer

@wiredsis

Leo Gorodinski

@eulerfx

Hussam Abu-Libdeh

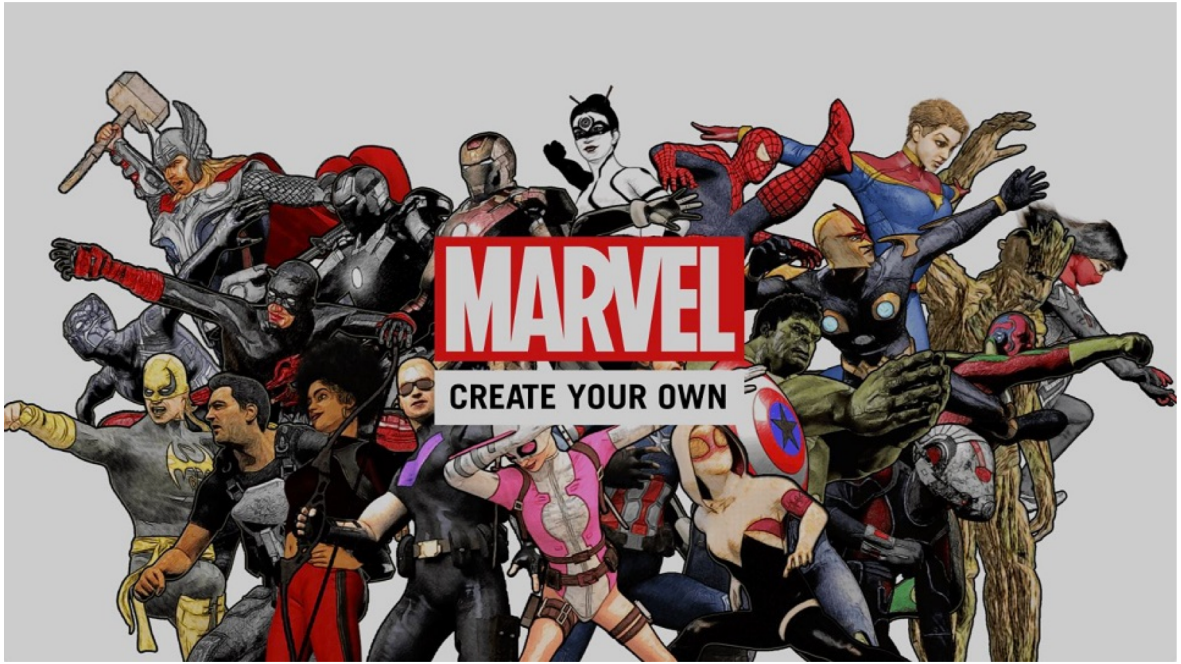
@hussam

Erich Ess

@egarhardess

Not Your *Average* E-Comm

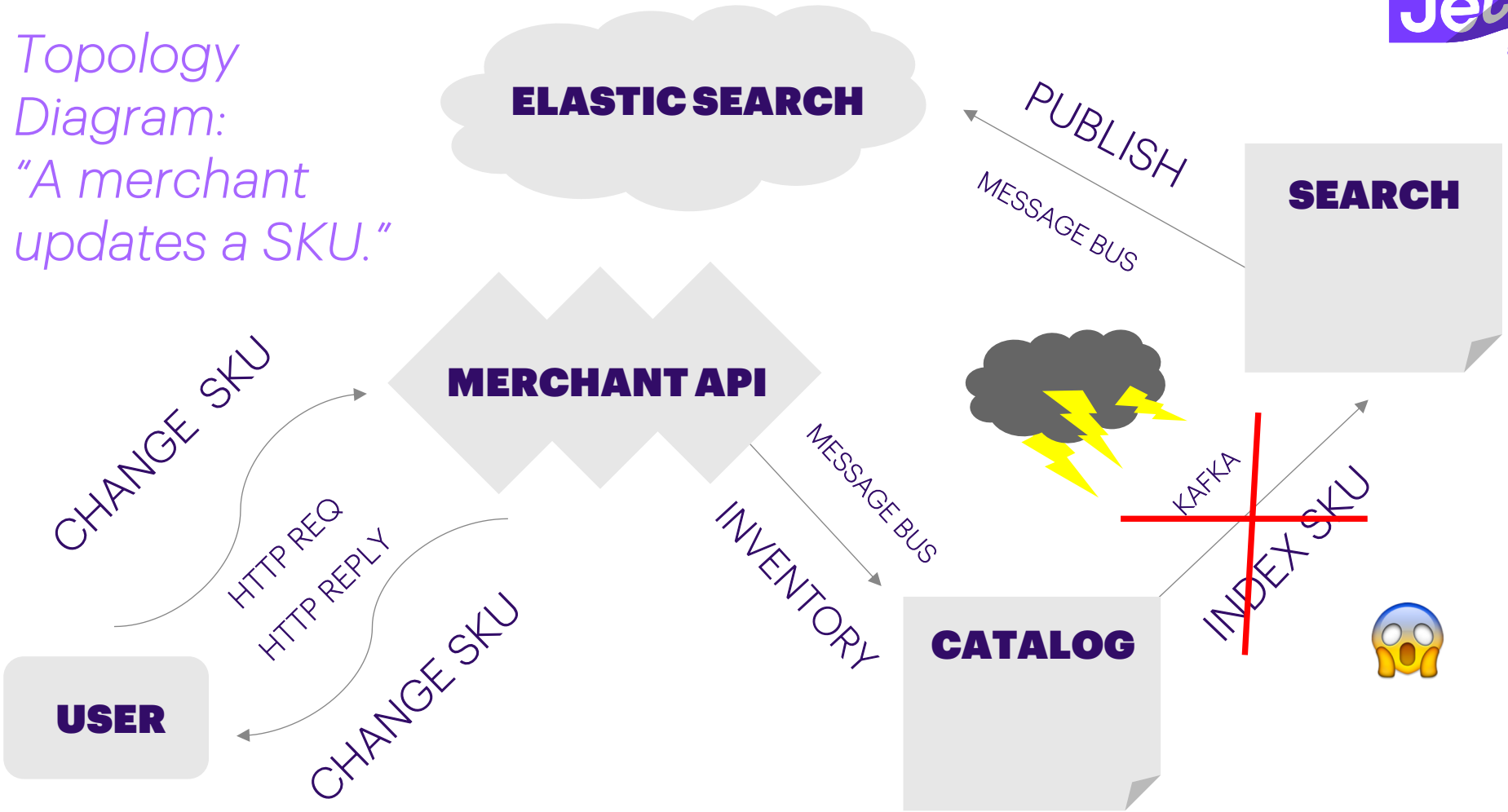
- Event Sourcing
- F# Language
- Multi-region
- Containers
- Asynchronous
- Hosted in Azure

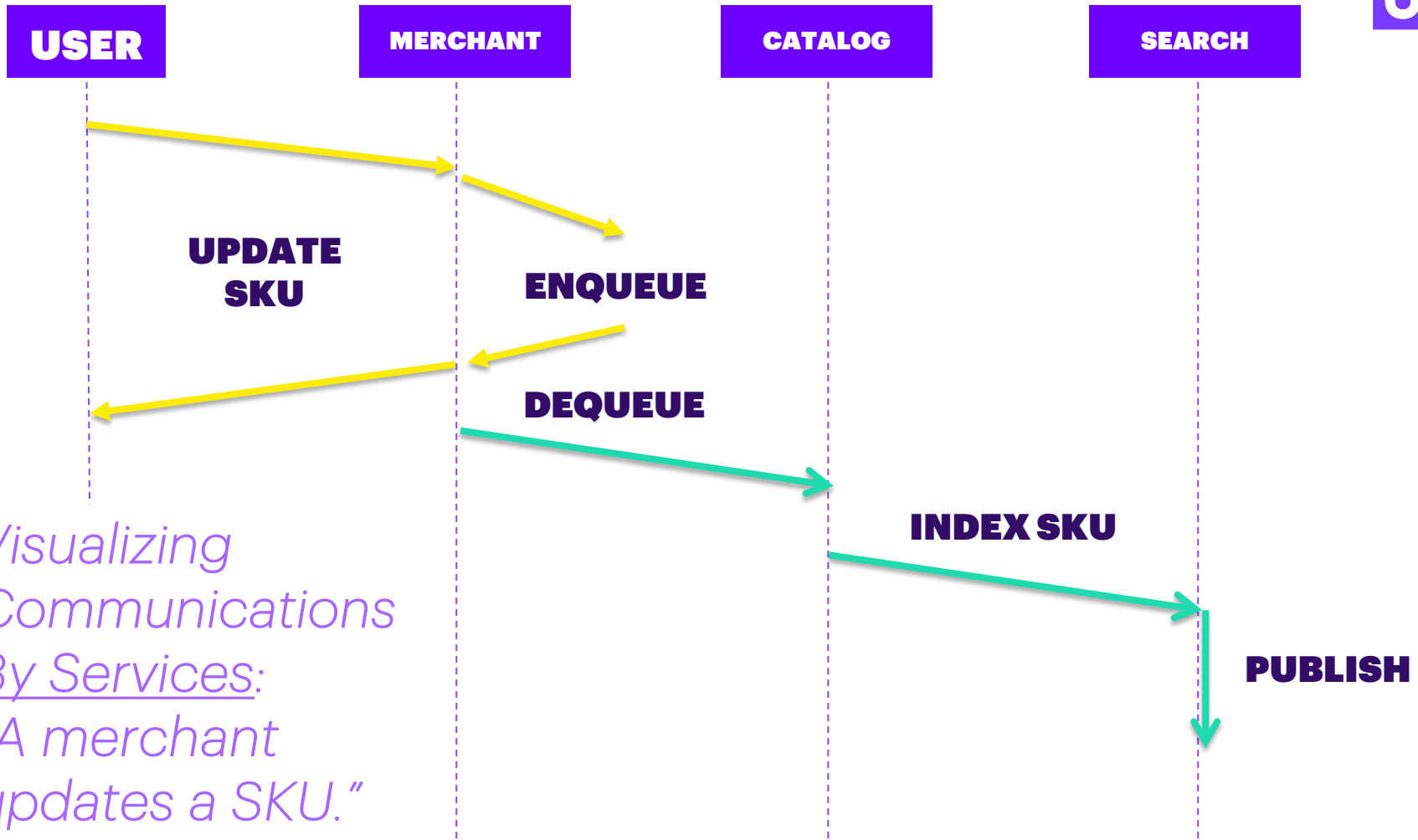


Why does Jet care about Distributed Tracing?

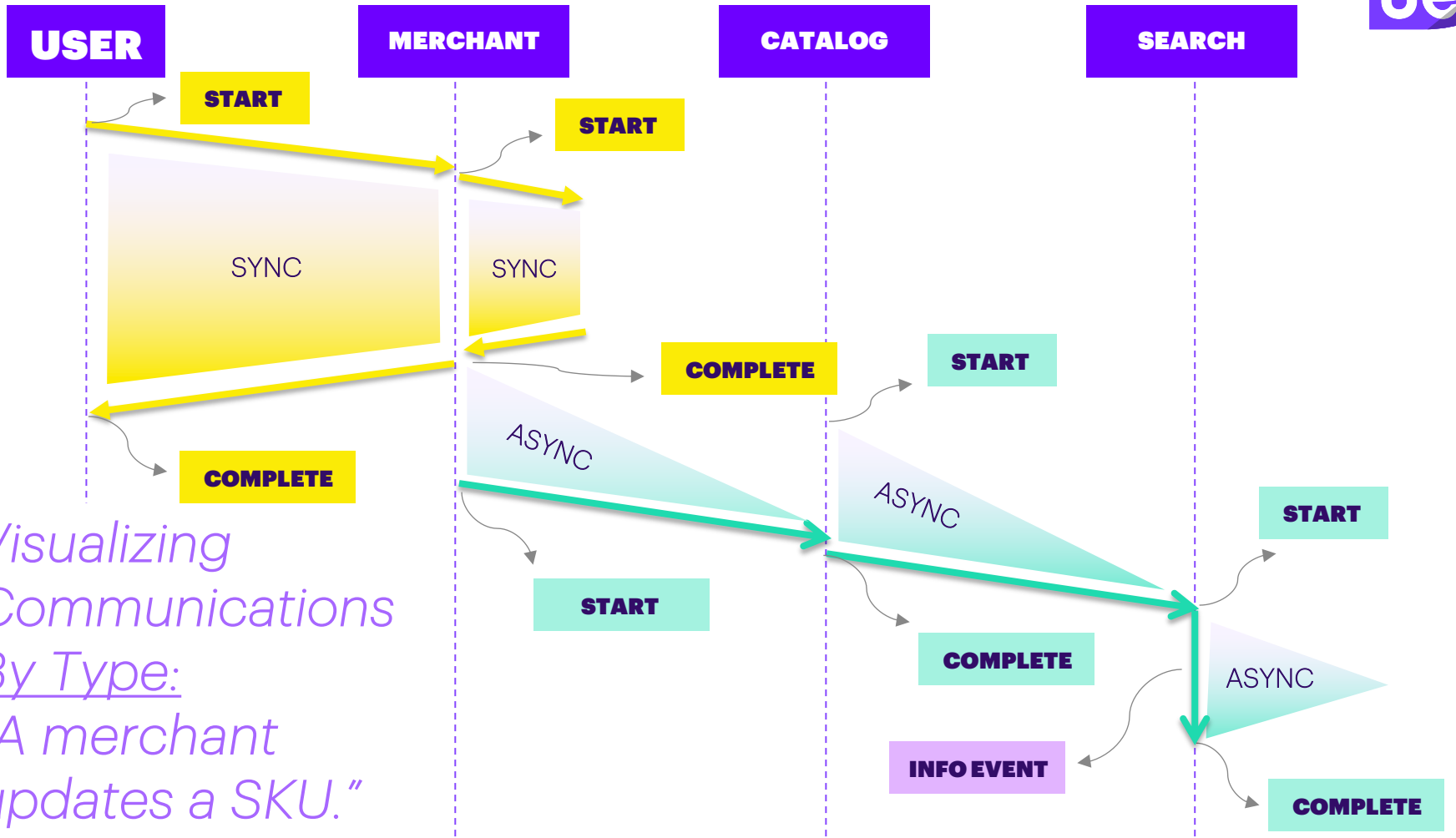
“Show me all inventory updates which failed for a given merchant.”

Topology Diagram: "A merchant updates a SKU."

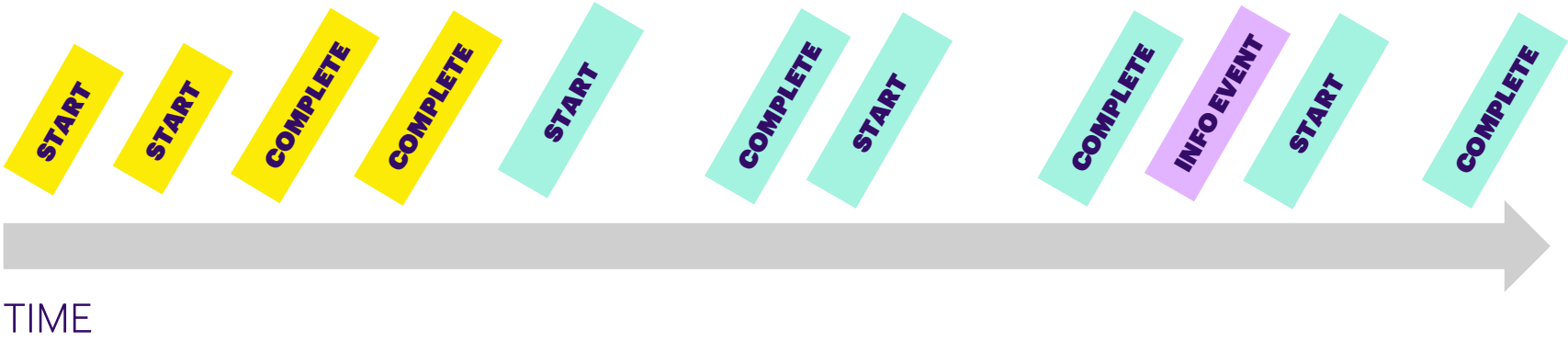




Visualizing Communications By Services:
"A merchant updates a SKU."



Visualizing Communications By Type:
"A merchant updates a SKU."



Visualizing An Event Log : "A merchant updates a SKU."

What exists in the ecosystem today?

Ecosystem Overview:

- **OpenTracing** *a vendor-neutral open standard for distributed tracing. Influenced by Dapper & Zipkin.*
- **Dapper** *Google's tracing platform, used mostly for RPC interactions.*
- **Zipkin** *Twitter's tracing system based on Dapper.*
- **OpenCensus** *A single distribution of libraries for metrics and distributed tracing with minimal overhead that allows you to export data to multiple backends.*

We decided to make a custom Distributed Tracing platform.
WCGW?


EVER MAKE A MISTAKE IN LIFE?
LET'S MAKE THEM BIRDS. YEAH, THEY'RE BIRDS NOW.
quickmeme.com

Problem: Can’t Trace All The Things All At Once

Solution:

- Treat it like a traditional MVP for a product, not R&D.
- Differentiate solutions on persona needs.
- Leverage cross-team coordination to identify which flows people “want to trace” and “need to trace.” Find Tracing “Sponsors” in your company.

Problem: Unwieldy Over-The-Wire Specification

Solution:

- Emit “baggage” both scoped to the Span & the entire Trace as ‘events’
- Since events are immutable and written to a log with a sequence number (using Kafka) we avoid mutable global state.
- Minimalist wire spec containing just Guids and “Operation Names”

Old Metadata Format:


```
type Header =

{
  HeaderVersion : int
  CorrelationIds : string list
  MessageId : string
  ParentIds : string list
  ProducerId : string
  PayloadSchemaUri : string
  TicksFromEpoch : int64
  MTags : Map<string,string>
  Ptags : Map<string,string>
}
```

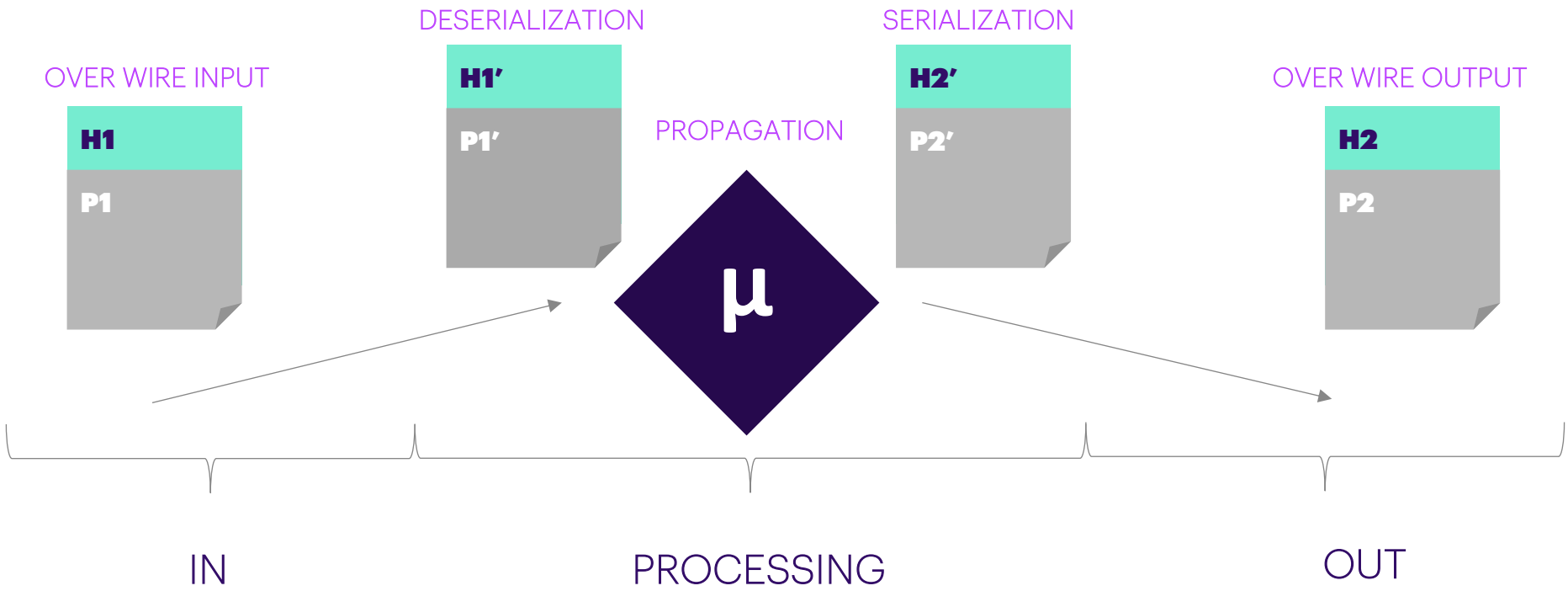
New Metadata Format:


```
type TraceContext =

{
  // a GUID which represents the
  // instance of the traced action
  trace_id : string

  // the operation name which generated
  // this trace data
  op : string

  // optional tags which don't get emitted in carrier
  // just key/value pairs
  tags : TraceTags
}
```

Problem: Not Great Library, Not Easy To Add-In Tracing Solution:

- Auto-calculate Span Latency information
- Integrate with Nomad Metadata
- Computation Expressions in F#
- Compatibility with multiple mediums: Kafka, Event Streaming Libs, Azure Databases, HTTP, etc
- Bussing of Telemetry Events

Propagation Overview:



Thinking in Envelopes


```
module Envelope =

  /// A message in its serialized state
  type Serialized = Serialized

  /// The message read from a data source
  type In = In

  // Currently in process
  type InProcess = InProcess

  /// The message which will be written to a data source
  type Out = Out
```

Thinking in Envelopes, Phantom Types


```
let prepareOutEnvelope
  serviceName schemaUri (outEnvelope:Envelope<'kind, _>)
  : Envelope<Out,_> =

  let header =
    Header.Propagate serviceName schemaUri [outEnvelope.Header]
  { Header = header
  Payload = outEnvelope.Payload }

let prepareOriginEnvelope serviceName schemaUri outPayload
: Envelope<Out,_> =

  let header = Header.Origin serviceName schemaUri
  { Header = header
  Payload = outPayload }
```

Envelopes all the way down...


```
// The madness continues…

let map f (envelope:Envelope<'kind,_>) : Envelope<InProcess,_> =
{ Header = envelope.Header; Payload = f envelope.Payload }

let resultMap
(f: 'a -> Result<'b, _>) (envelope:(Envelope<In, Result<'a, _>>))
: Envelope<InProcess, _> =

  envelope |> map (function
    | Ok j -> f j
    | Error err -> Error err)
```

Rework: Channel Types, Working over “mediums”...


```
/// A request-reply server channel wherein the channel
/// handles inputs of type 'i and produces outputs of type 'o.
/// The server is expressed in terms of inputs of type 'a
/// and outputs of type 'b.
type ReqRepServer<'i, 'o, 'a, 'b> =
{ ch : Channel
decoder : Dec<'i, 'a * TraceContext>
encoder : Enc<'o, 'b * TraceContext> }

/// A publish-subscribe channel with inputs of type
/// 'i, outputs of type ‘o and domain-specific message type 'a.
type PubClient<'m, 'a> =
{ ch : Channel
encode : Enc<'m, 'a * TraceContext> }

/// A publish-subscribe channel with inputs of type
/// 'i, outputs of type ‘o and domain-specific message type 'a.
type SubClient<'m, 'a> =
{ ch : Channel
decoder : Dec<'m, 'a * TraceContext> }
```

Rework: Channel Types, Working over “mediums”...


```
/// A request-reply server channel wherein the channel
/// handles inputs of type 'i and produces outputs of type 'o.
/// The server is expressed in terms of inputs of type 'a
/// and outputs of type 'b.
type ReqRepServer<'i, 'o, 'a, 'b> =
{ ch : Channel
decoder : Dec<'i, 'a * TraceContext>
encoder : Enc<'o, 'b * TraceContext> }

/// A publish-subscribe channel with inputs of type
/// 'i, outputs of type ‘o and domain-specific message type 'a.
type PubClient<'m, 'a> =
{ ch : Channel
encode : Enc<'m, 'a * TraceContext> }

/// A publish-subscribe channel with inputs of type
/// 'i, outputs of type ‘o and domain-specific message type 'a.
type SubClient<'m, 'a> =
{ ch : Channel
decoder : Dec<'m, 'a * TraceContext> }
```

Feeling experimental??

Let’s go to an editor.

New Metadata Format:


```
type TelemetryEvent =

{
  trace_id : string

  /// Either Start, Complete, or a Custom event type
  event_type : string

  /// The timestamp of the event being written
  timestamp : DateTimeOffset

  tags : TraceTags
}
```

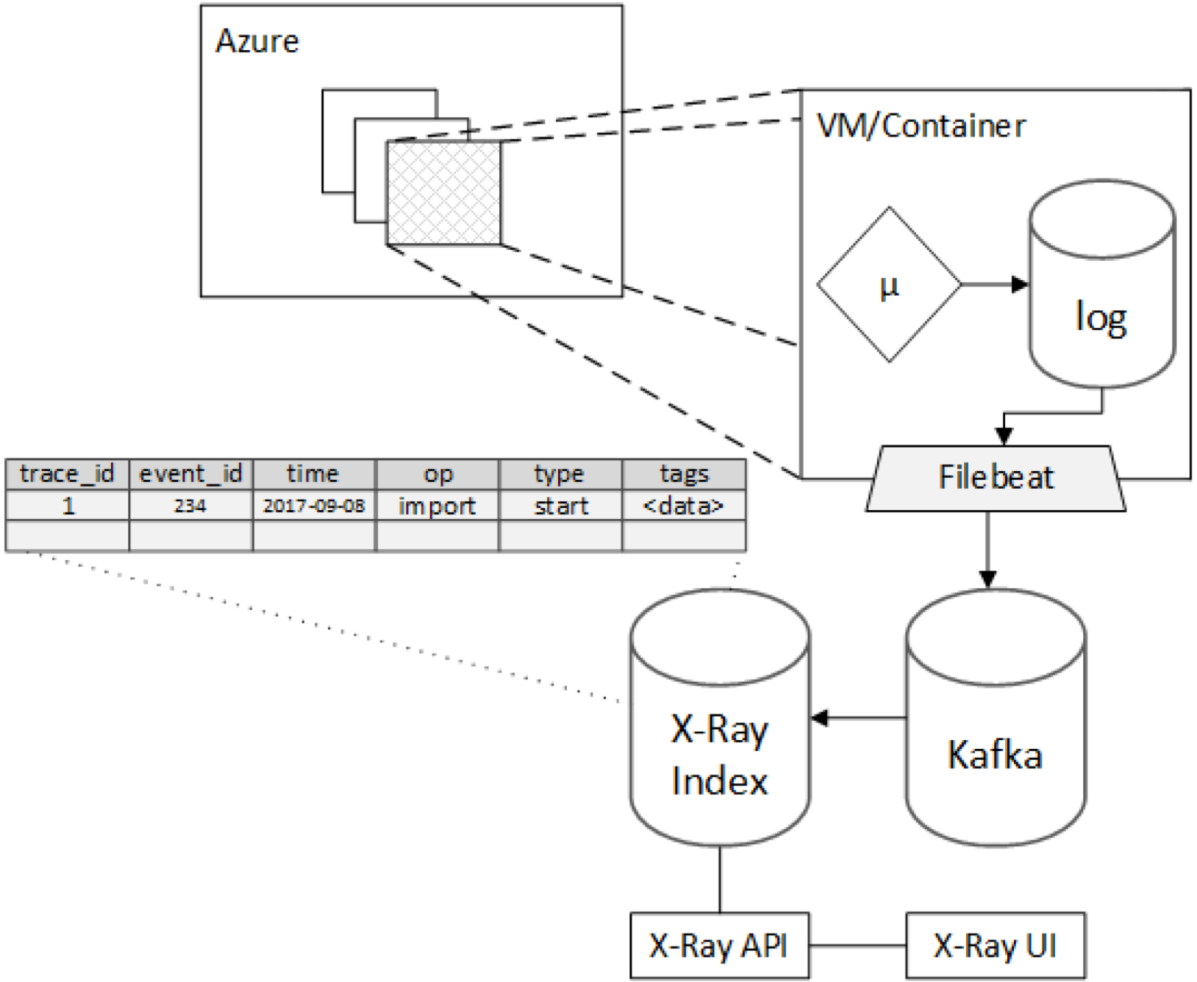
Visualizing An Event Log : "A merchant updates a SKU."

```
{
  "event_type": "START",

  "trace_id": "1a2b3",

   "timestamp": "123456789"

   "tags" : {
      "op_name": "IndexSkus"
      "merchant_id": "4c5d6e7"
    }
}
```

Problem: Uncertain Querying & Indexing

Solution:

- Graph Definitions via Reflection
- Semantics around “starting” & “stopping” an “Operation”
- Projection from CosmosDB which sends snapshots to Splunk


Azure
VM/Container
μ
log
Filebeat
trace_id
event_id
time
op
type
tags
1
234
2017-09-08
import
start
<data>
X-Ray Index
Kafka
X-Ray API
X-Ray UI

Problem: Bad UI/UX
Solution:

- Current frameworks aren't a great fit.
- Leverage Splunk for custom dashboards per team.
- Iterating on multiple custom UIs that feature topology graphs (force-directed graphs) and various hierarchical tree views.

In the future…

- Unified Logging
- Schematization
- Higher level abstractions
- More integrations, more backends, more projections
- Open Source!?!?!?!

Lessons You Can Go To The Bank On:

1. Leverage product to target owners and “tracing sponsors” around your org.
2. You might need to make some flashy views just to get buy in from stakeholders even if it doesn’t help developers solve real problems.
3. This project takes time and careful requirement gathering. Carve out a useful tracing flow, one distributed action at a time.

THANK YOU FOR YOUR TIME!

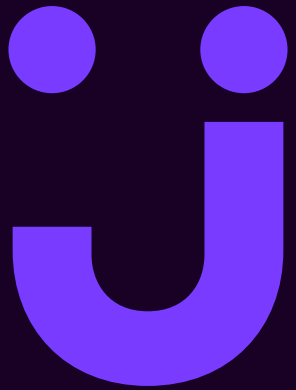
Come find me in the hallway if you:

- Want to talk about tracing
- Want to *not* talk about tracing
- Want to write F# for a living :]

@wiredsis

More reading…

- https://opencensus.io/
- http://opentracing.io/
- https://www.usenix.org/conference/nsdi-07/x-trace-pervasive-network-tracing-framework
- https://research.google.com/pubs/pub36356.html
- https://www.usenix.org/conference/nsdi-07/x-trace-pervasive-network-tracing-framework
- https://zipkin.io/
- https://research.fb.com/publications/canopy-end-to-end-performance-tracing-at-scale/
- https://medium.com/@eulerfx/scaling-event-sourcing-at-jet-9c873cac33b8 (He's Hiring)




jet