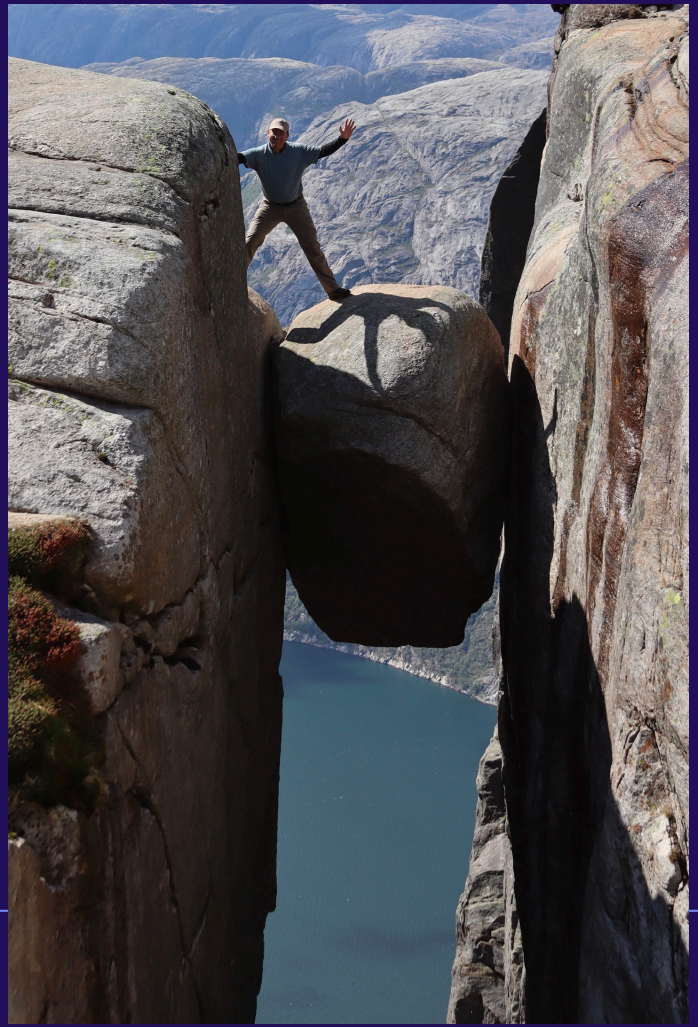
Welcome to ElixirConf® US 2024

Supercharging Kafka Processing with Broadway:

Achieving Unmatched Parallelism and Efficiency

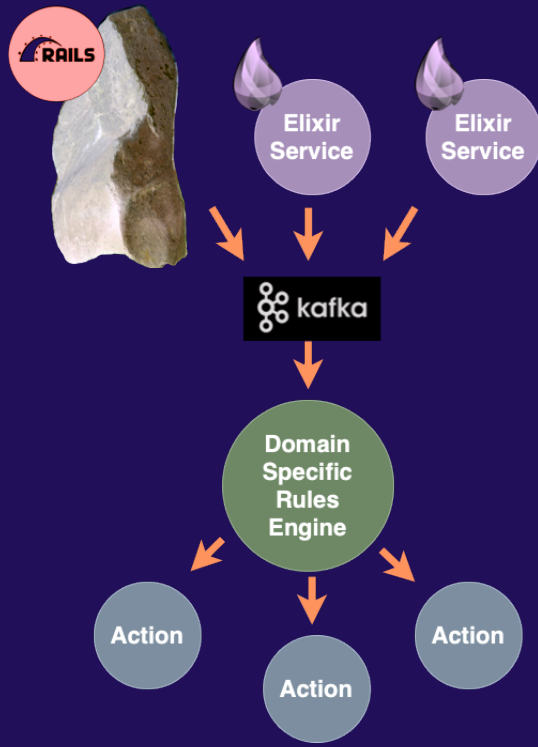
About

- Software Architect
- Johannesburg, South Africa
    - Learned to "do more with less"
- Co-founded a company that built workflow products using an event-driven architecture
    - Tampa, FL
- 20 Ruby open source projects
    - 40 million downloads
    - Scaling Ruby into the Enterprise
- Ruby to Elixir
    - Extreme concurrency
    - Lightning performance
    - Reducing cloud costs
    - "Do more with less"



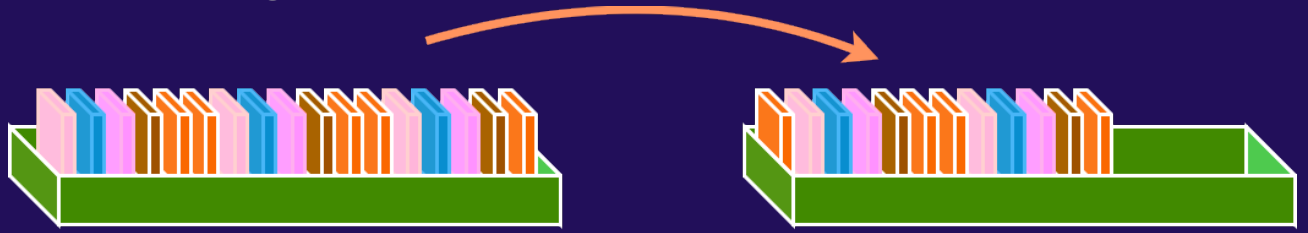
Problem?

- Domain Specific Rules Engine
- Consume events from Kafka
    - Rails monolith
    - Elixir services
    - Existing Topics
        - Partitioned by Tenant id
- Actions
    - HTTP Calls
- High Volume
    - Software as a Service (SaaS)



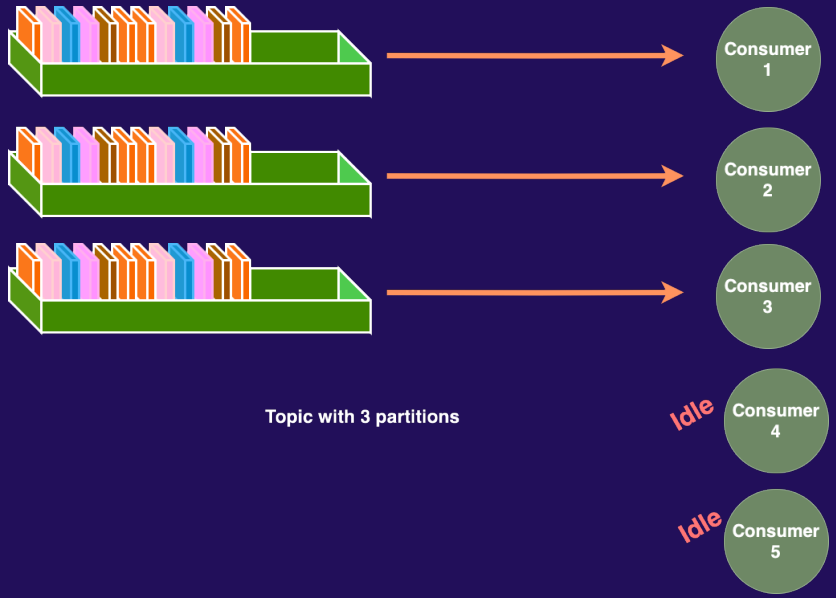
Apache Kafka

- High volume, high throughput, distributed **event streaming platform**
- Database transaction log
    - Immutable
    - Partition-able
    - Stored in order
    - Drop older log files



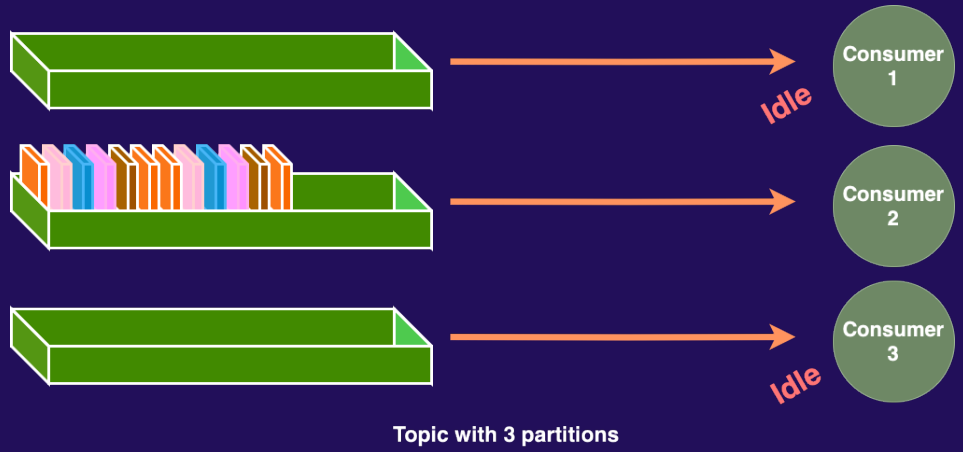
Single Partition Transaction Log

Apache Kafka: Consumer Scalability



- One active consumer per partition
  - Per consumer group
- Scalability limited to the number of partitions

Apache Kafka: Unbalanced Partitioning



- Key based partitioning
  - For example: Tenant ID
  - Retains order by tenant
- Multiple tenants on the same partition
- Noisy neighbor problem
  - One tenant impacts others
  - Enterprise customers
- Random partitioning
  - Better balancing
  - Cannot ensure ordering by tenant
  - Could process `update` before `create`

Apache Kafka: Partition Count

- Assign "just right" number of partitions at topic creation
    - Too few: Limit scalability
    - Too many: Performance issues
- Increasing partitions
    - Complex
    - Affects data distribution and consumer behavior
- No magic formula
    - Based on experience
    - Forged in production

Apache Kafka: Scalability

- How to increase Consumer concurrency?
    - Increase partition count
    - Dealing with unbalanced partitions

- Does Elixir offer another solution?

Elixir Concurrency Model?

- Task
- GenServer
- GenStage
- Flow
- Broadway
- Oban

Task

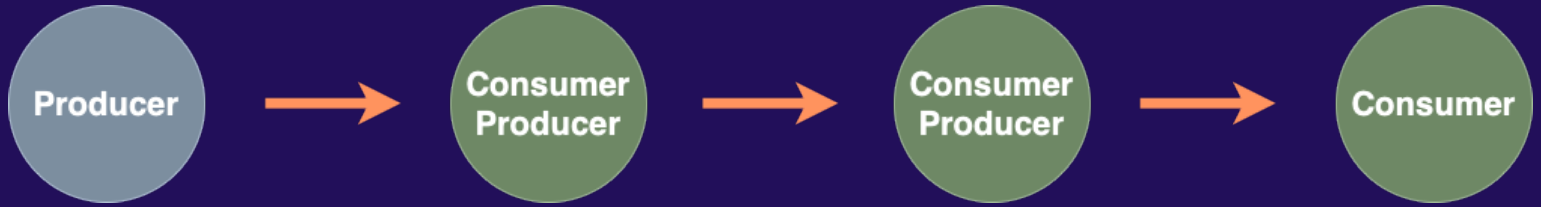
- Run code in another process
  - `Task.start/1`
  - `Task.async/1`
  - `Task.Supervisor`
    - Run and manage multiple tasks

GenServer

- Complex Tasks
- Maintain state
- Long running
- Pull model.
  - Kafka Consumer
    - Fetch one message at a time,
    - Process it,
    - Fetch the next message
- `PartitionSupervisor`
  - Run multiple GenServers.

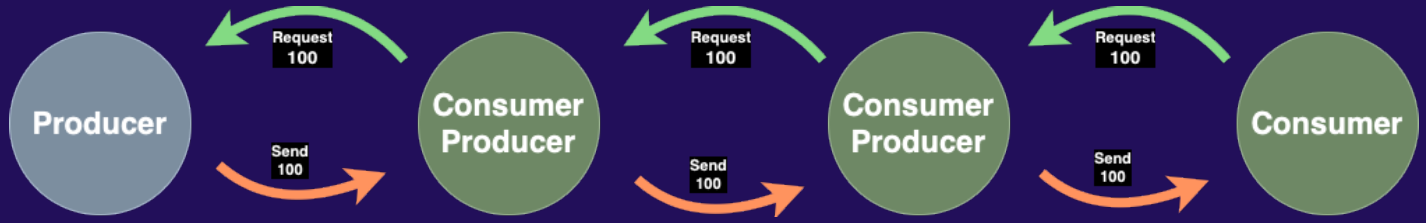
GenStage

- Multi-stage processing pipeline
- Producer pushes messages to consumers



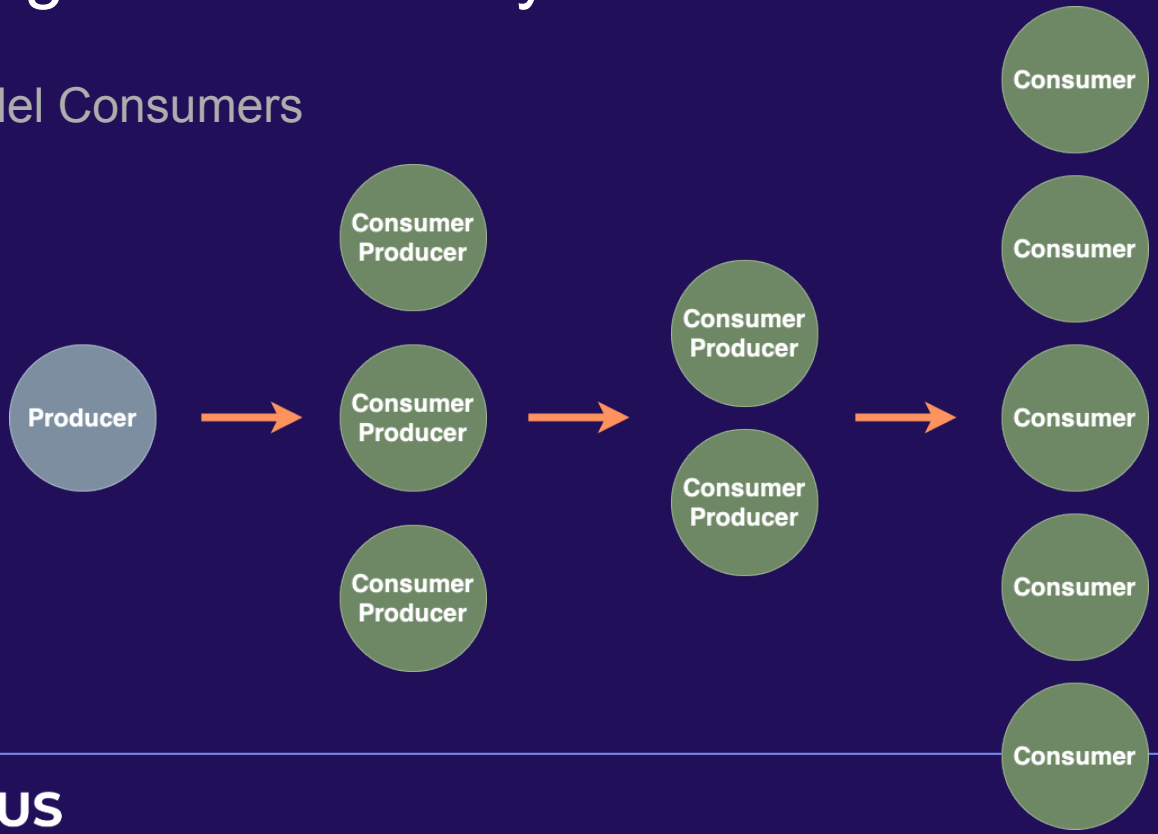
GenStage: Back Pressure

- Consumer requests messages
- Producer pushes messages
  - Up to requested count
- Consumer asks for more during processing



GenStage: Concurrency

- Parallel Consumers



Flow

- Concurrently process large datasets
  - `map, reduce`
  - `filter`
  - `group_by`
  - `take_sort`
- Partitioning
- Windows
- Triggers
- Back pressure

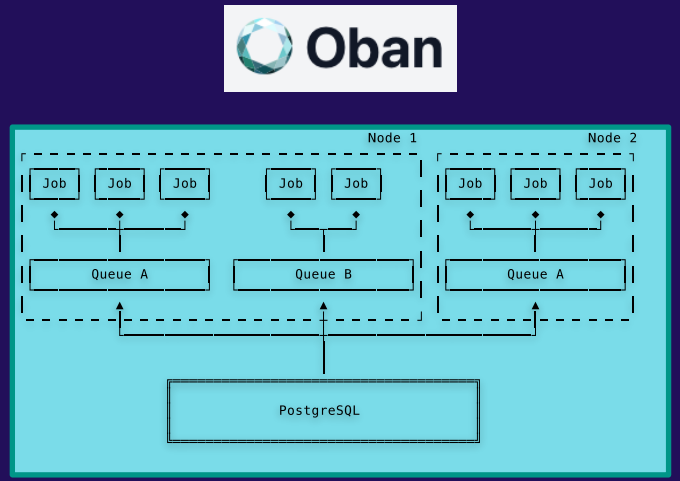
```
File.stream!("path/to/some/file")
|> Flow.from_enumerable()
|> Flow.flat_map(&String.split(&1, " "))
|> Flow.partition()
|> Flow.reduce(fn -> %{} end, fn word, acc ->
  Map.update(acc, word, 1, & &1 + 1)
end)
|> Enum.to_list()
```

Broadway

- Event Stream Processing
  - High volume, high throughput, event stream processing
- Multi-stage processing pipeline
- Back-pressure
- Rate Limiting
  - Prevent downstream DDOS
- Batching

Oban

- Job Processing
- PostgreSQL as a queue
- PostgreSQL is the bottleneck
  - Better suited to large jobs
  - Not ideal for Kafka Event processing

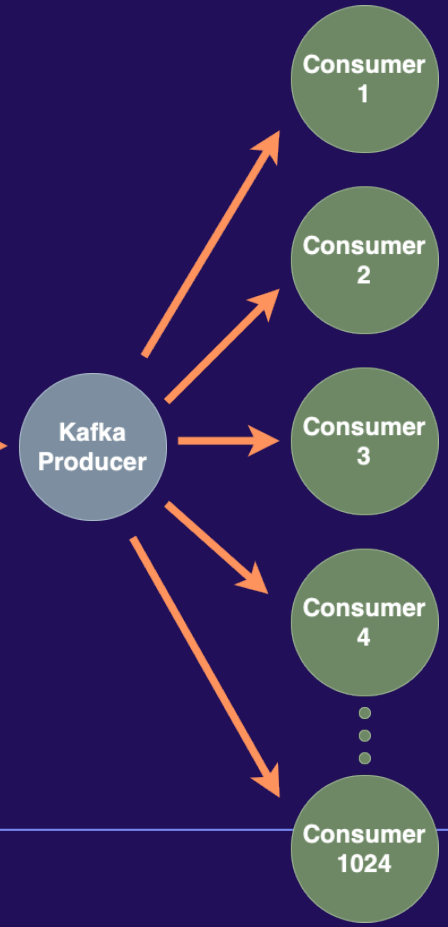


Elixir Concurrency Model?

- Task
- GenServer
- GenStage
- Flow
- Broadway
- Oban

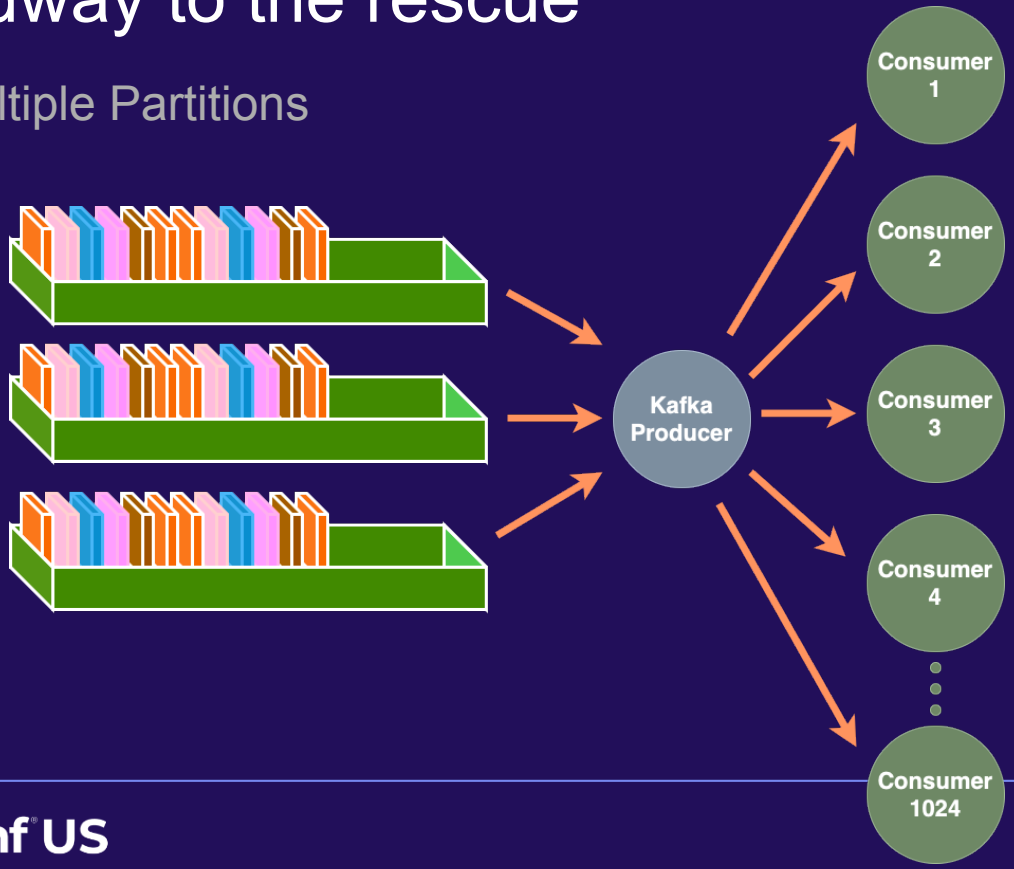
Broadway to the rescue

- Limited Partitions



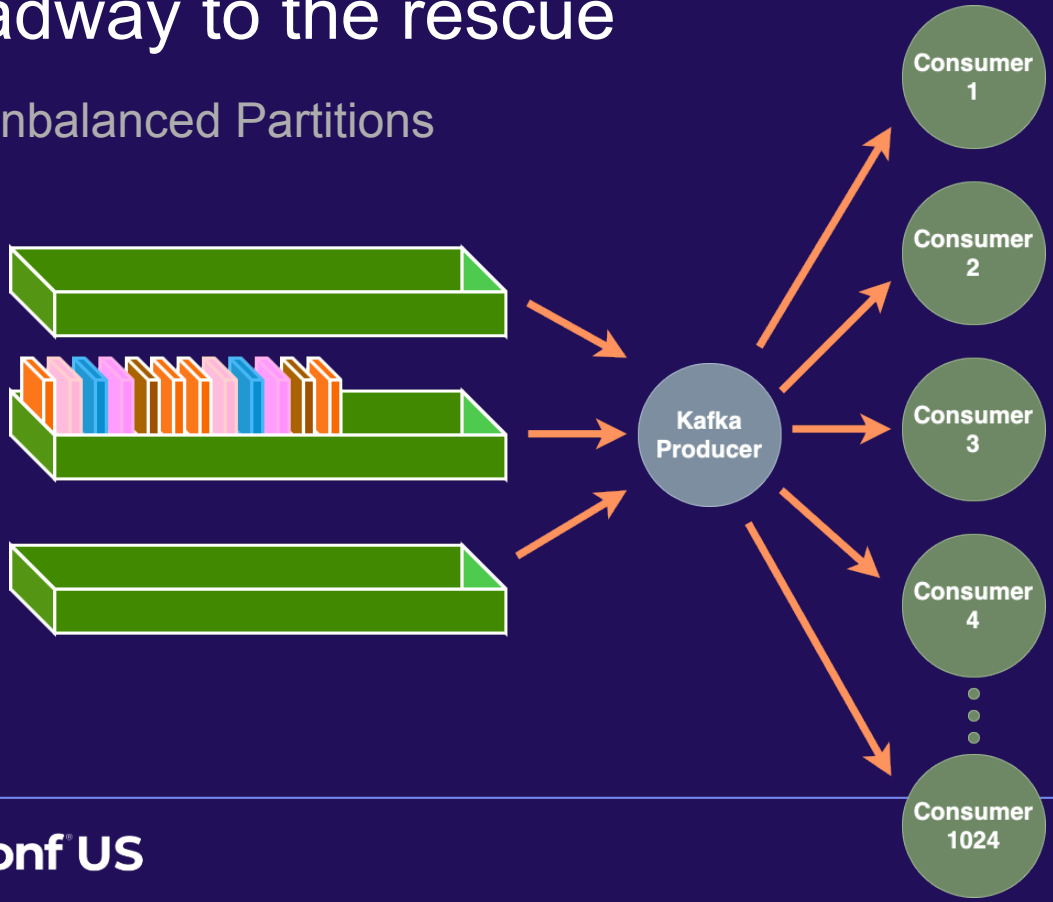
Broadway to the rescue

- Multiple Partitions



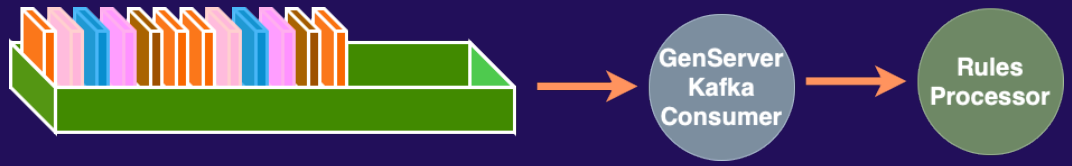
Broadway to the rescue

- Unbalanced Partitions



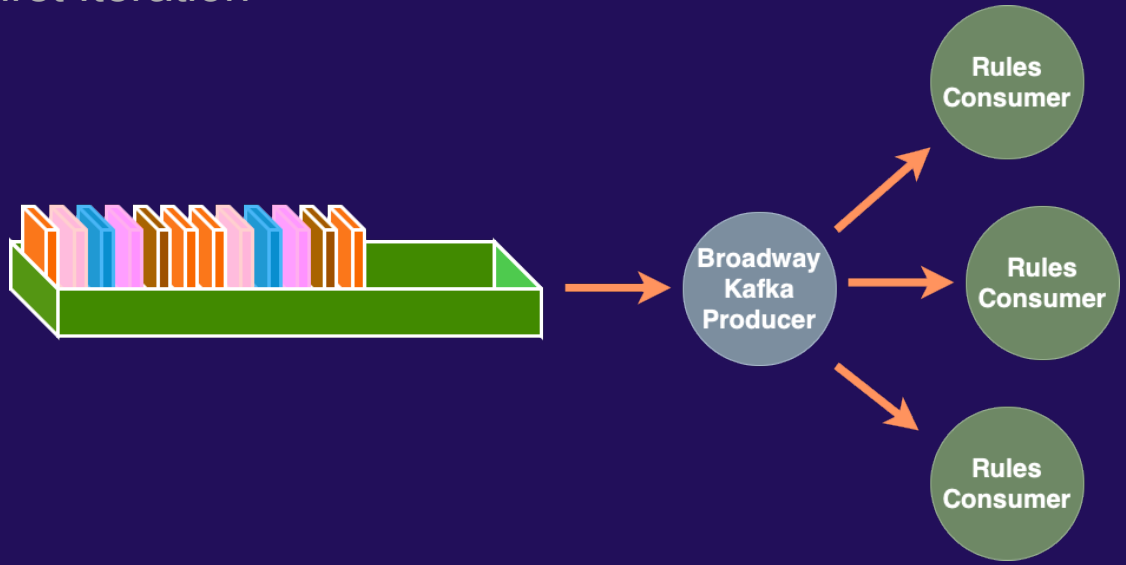
Rules Engine Solution

- Before Broadway
  - GenServer
  - Planned: PartitionSupervisor



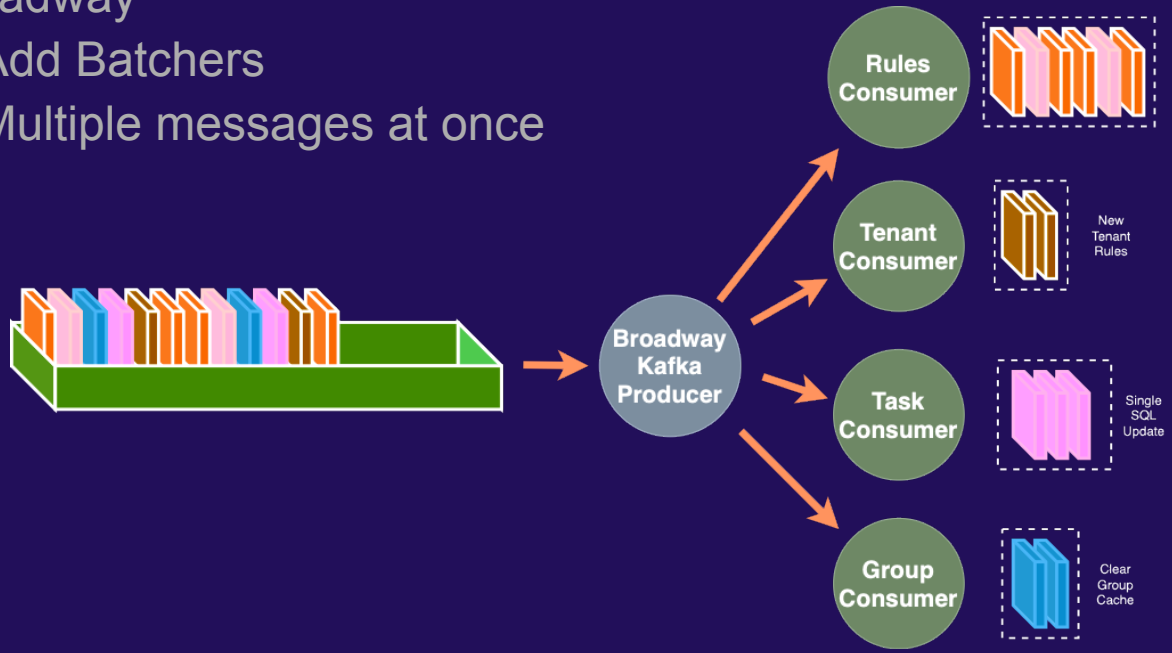
Rules Engine Solution

- Broadway
  - First Iteration



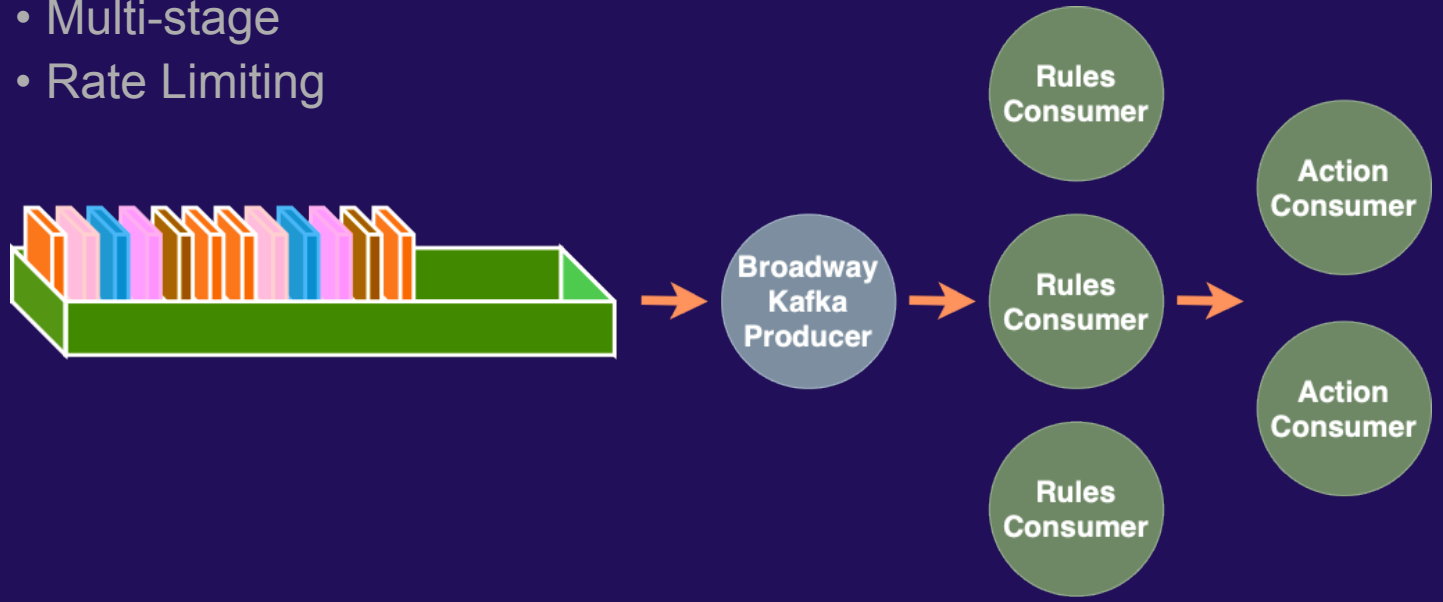
Rules Engine Solution

- Broadway
  - Add Batchers
  - Multiple messages at once



Rules Engine Solution

- Broadway
  - Multi-stage
  - Rate Limiting



Do Differently

- Go straight to Broadway
- Data Source Independent
  - Amazon SQS
  - Google Cloud PubSub
  - Apache Kafka
  - RabbitMQ
  - …

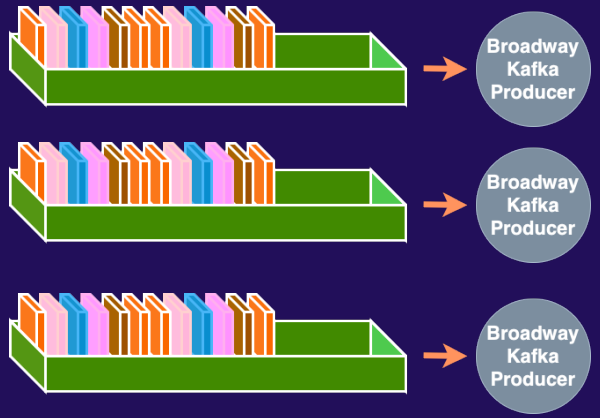
```elixir
defmodule MyBroadway do
  use Broadway

  def start_link(_opts) do
    Broadway.start_link(__MODULE__,
      name: __MODULE__,
      producer: [
        module: {BroadwayKafka.Producer, [
          hosts: [localhost: 9092],
          group_id: "group_1",
          topics: ["test"],
        ]},
        concurrency: 1
      ],
      processors: [
        default: [
          concurrency: 10
        ]
      ]
    )
  end

  def handle_message(_, message, _) do
    IO.inspect(message.data, label: "Got message")
    message
  end
end
```

Challenges

- Match Producers to partitions
  - Kafka Ack did not always flow back



Kafka Best Practices: Format

- Standardized message format
  - JSON
  - Event Name
  - Event Type
  - Event Data
  - Idempotency key
    - Prevent duplicate processing

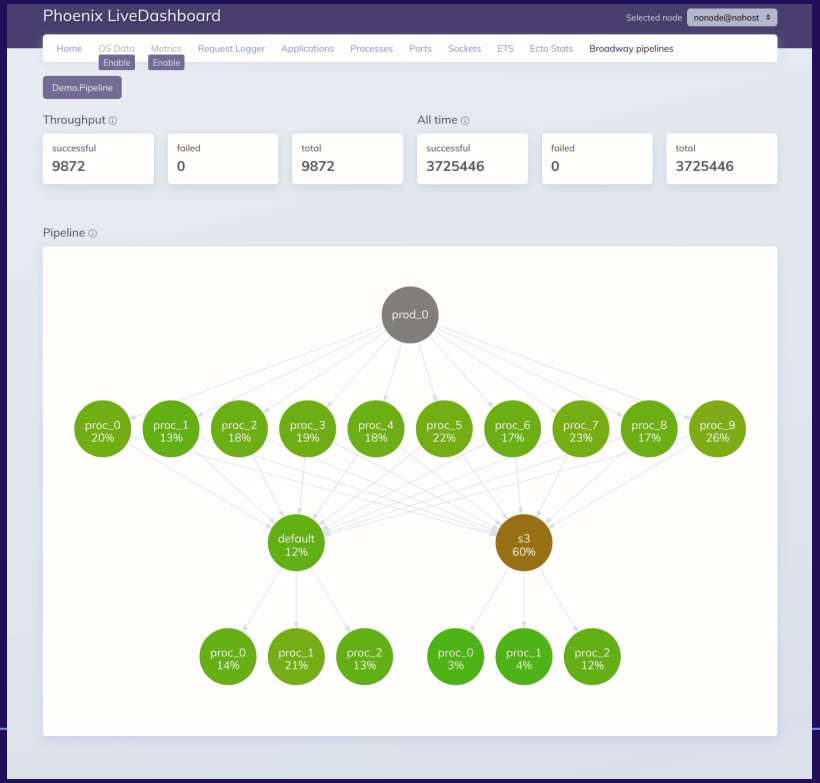
Kafka Best Practices: Creating new topics

- The "right" partition count
    - Too few: Limit scalability
    - Too many: Performance issues
- Increasing partitions
    - Complex
    - Affects data distribution and consumer behavior
- Using a Partition Key
    - Message ordering by key
    - Unbalanced partitions

Broadway Features

- Back-pressure
- Automatic acknowledgements at the end of the pipeline
- Batching
- Fault tolerance
- Graceful shutdown
- Metrics
- Custom failure handling
- Ordering and partitioning
- Rate-limiting
- Live View Dashboard

Broadway Dashboard



Questions?

- Github
  - reidmorrison
- Linked In
  - linkedin.com/in/reidmorrison