Queues, Topics, and Their Trade-offs

Five questions to pick the "right" one.

RAUL JUNCO
OCT 16, 2024

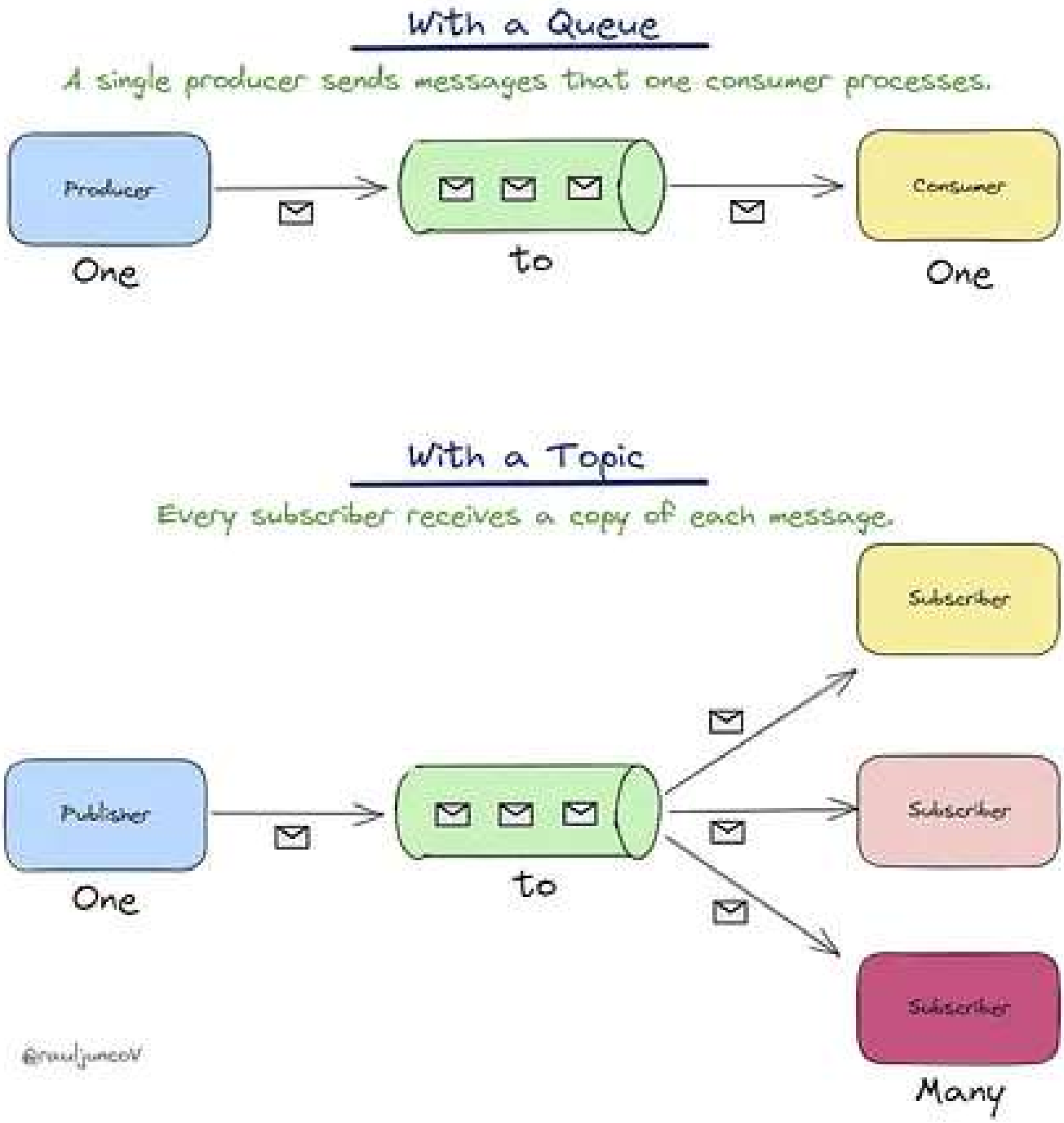
You may have heard about the benefits of adding asynchronous processing to your system design. With async processing, you can:
Build more resilient systems.
Build event-driven architectures.
Have scalable systems.

You can do this using **Queues** and **Topics**. But which one should you use? They serve different purposes, and making the right choice depends on your requirements.
Here are the five questions you can use to decide:

1. Application Requirements: One-to-One vs. One-to-Many

The most critical question is: Do you need one-to-one or one-to-many messaging?
Use a Queue to ensure that exactly one consumer processes each message. Queues follow a point-to-point model: a single producer sends messages that one consumer processes.

Use a Topic if you need to send messages to multiple consumers. Topics follow a publish-subscribe model—every subscriber receives a copy of each message. Topics work well for broadcasting events like notifications.



Sketched using Multiplayer

2. Message Durability and Delivery Guarantees

How important is message durability to your application?

Queues provide strong **message durability** by design. If you can't afford to lose messages, queues help by keeping messages until they are processed and acknowledged by a consumer.

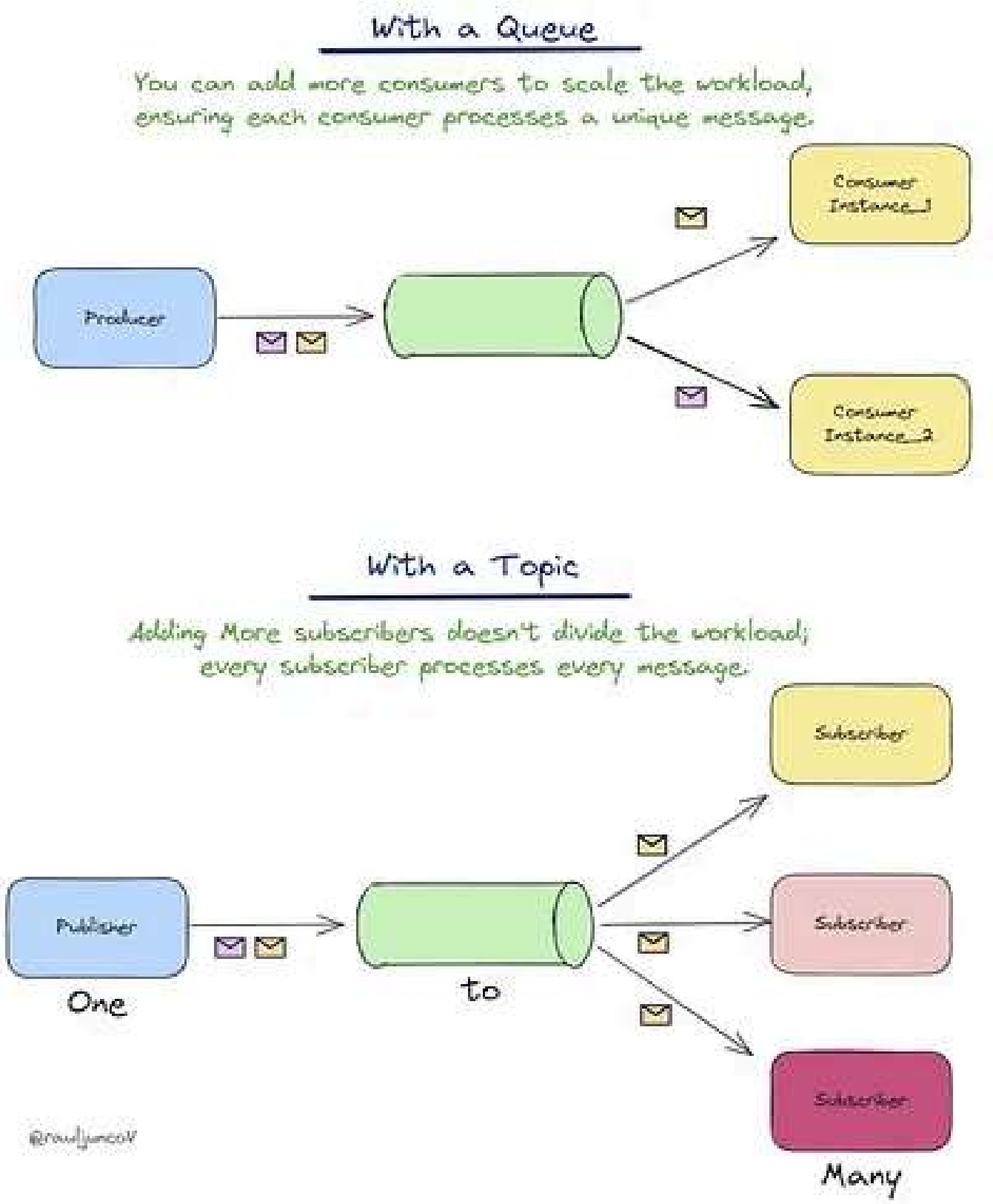
Topics can also offer durability, but this typically requires additional configuration and often depends on the technology. If you need a simple, reliable solution for keeping messages safe, queues are usually a better fit.

3. Scalability

How does your application need to scale?

Queues can distribute work across multiple consumers. Adding more consumers will help scale the workload, ensuring each consumer processes a unique message. This distribution model is ideal for handling high throughput.

Topics broadcast messages to all subscribers. Adding more subscribers does not split the workload; every subscriber processes every message. Topics are suitable for sharing information but can't efficiently divide tasks. Queues are better for efficient load distribution.



Sketched using Multiplayer

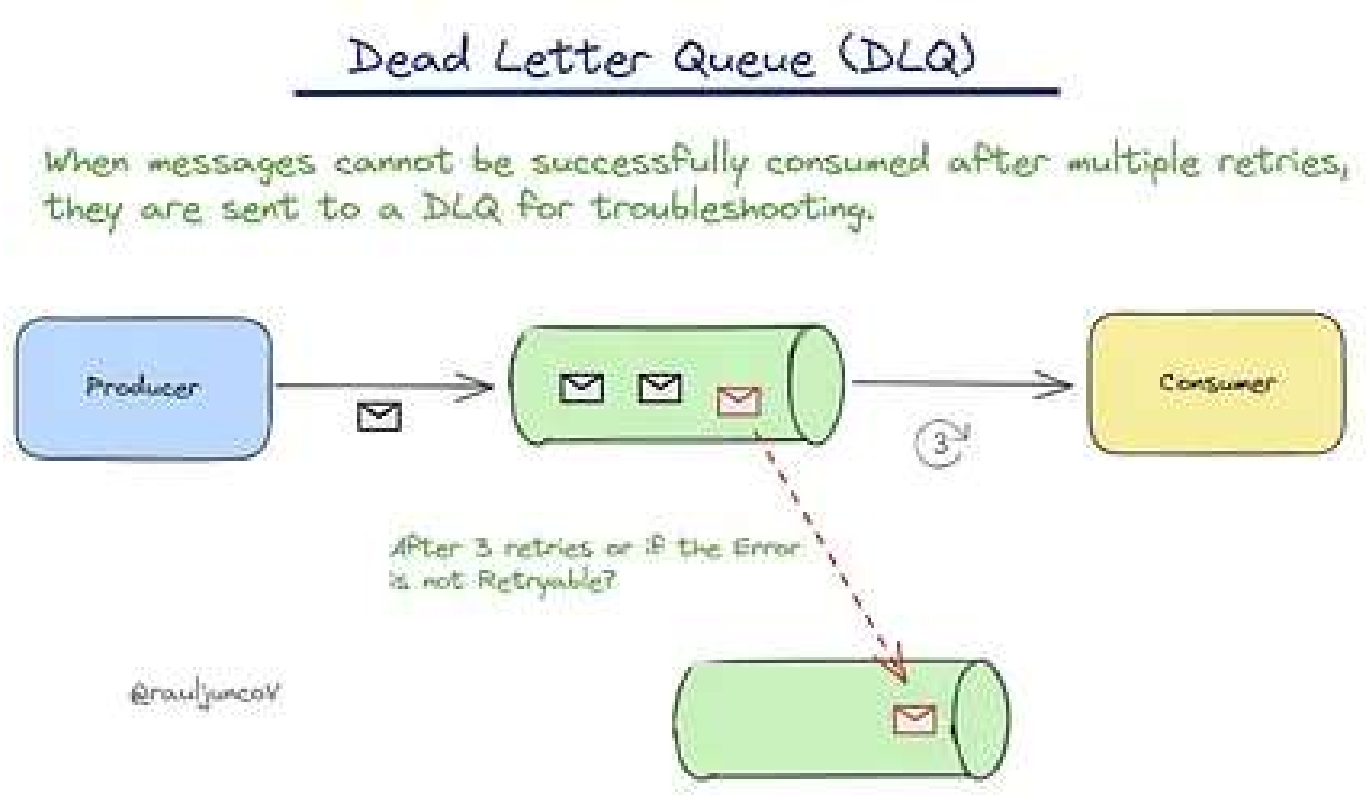
4. Handling Consumer Failures

What happens if a consumer is unavailable?

Queues handle consumer failures well. If a consumer goes offline, the queue retains the message until the consumer can process it, making queues ideal for reliability.

Queues also support **Dead Letter Queues (DLQs)**, which act as a safety net for unprocessable messages. When messages cannot be consumed after several retries, they are sent to a DLQ for troubleshooting, preventing issues from clogging up the main queue.

With **Topics**, keeping track of which subscribers received which messages adds complexity, especially if order matters. Managing this state can introduce overhead and make handling consumer failures more challenging.



Sketched using Multiplayer

5. Flexibility vs. Stability

Is your system stable or evolving?

Topics provide flexibility that works well for fast-evolving systems. You can add more subscribers or change subscription models without re-architecting the entire system.

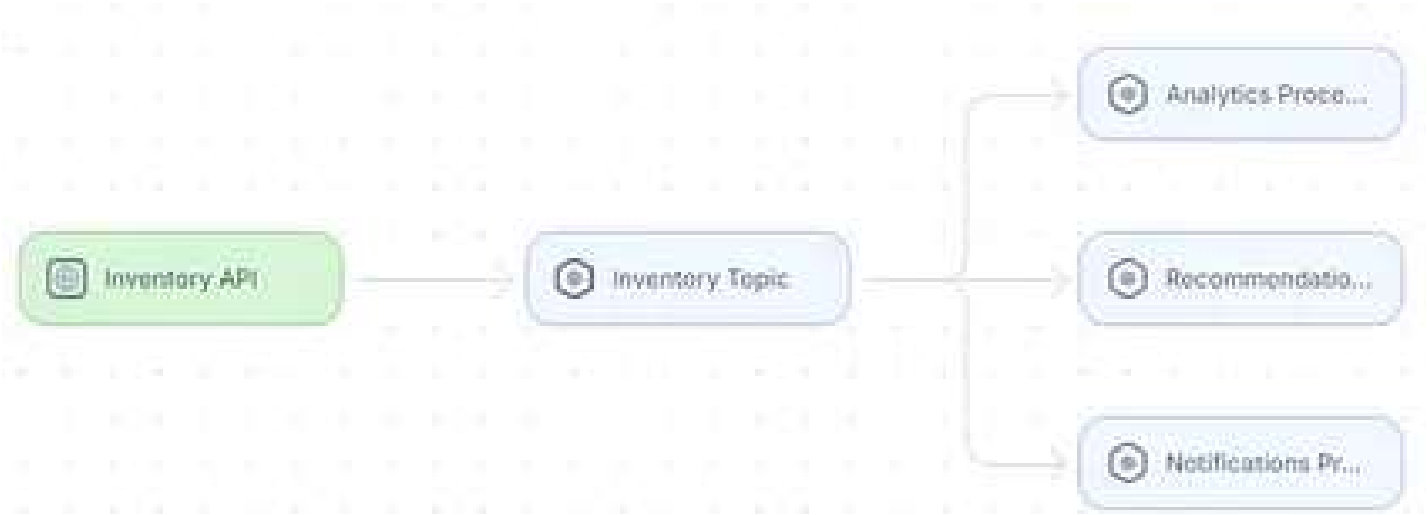
Queues, in contrast, offer simplicity. For stable, well-defined workflows, queues help keep things direct and reduce unnecessary complexity.

Let's put all these in context with a real-world example in an E-commerce System:

Use **queues** for order processing to ensure each order is handled exactly once.

Sketched using Multiplayer

Use **topics** for inventory updates to broadcast changes to multiple interested services (e.g., analytics, recommendations, and customer notifications).



Sketched using Multiplayer

Choosing the Right One

The decision between **topics** and **queues** ultimately depends on your application's requirements:

If you need **point-to-point communication**, use **queues**.

For **broadcasting information**, use **topics**.

If **durability** and **reliability** are essential, start with **queues**.

If you want **scalability** by distributing tasks across many consumers, **queues** again offer a better solution.

Likewise, if you need **flexibility** to adapt to evolving requirements, **topics** may give you the necessary agility.

Summary

Start with **queues** for a simple foundation, especially in the early stages of system development. As your system grows and requires more flexibility, adopting **topics** might make sense.

Keep these five guiding questions in mind:

. One-to-one or one-to-many messaging?
. How critical is message durability?
. Do you need workload distribution or information sharing?
. How do you handle consumer failures?
. Is the system stable or evolving?

Answering these questions will help you pick the right messaging tool for your architecture.

Remember: Queues keep it simple; Topics keep it flexible.

How do you pick the "right" one?