


Why Do We Need a Message Queue?

 BYTEBYTEGO
AUG 10, 2023 · PAID

 356

 5

 11

Share

In this issue, we’re diving deep into a widely-used middleware: the message queue.

Message queues have a long history. They are often used for communication between different systems. Figure 1 illustrates the concept of a message queue by comparing it to how things work at Starbucks.

At Starbucks, the cashier takes the order and collects money, then they write the customer’s name on a coffee cup to hand over to the next step. The coffee maker picks up the order and the cup and makes coffee. The customer then picks up the coffee at the counter. The three steps work asynchronously. The cashier just drops the order in the form of a coffee cup and does not wait for its completion. The coffee maker just drops the completed coffee on the counter and does not wait for the customer to pick it up.

When you place an order at Starbucks, the cashier takes the order and scribbles your name on a cup and moves to the next customer. A barista then picks up the cup, prepares your drink, and leaves it for you to collect. The beauty of this process is that each step operates independently. It is much like an asynchronous system.

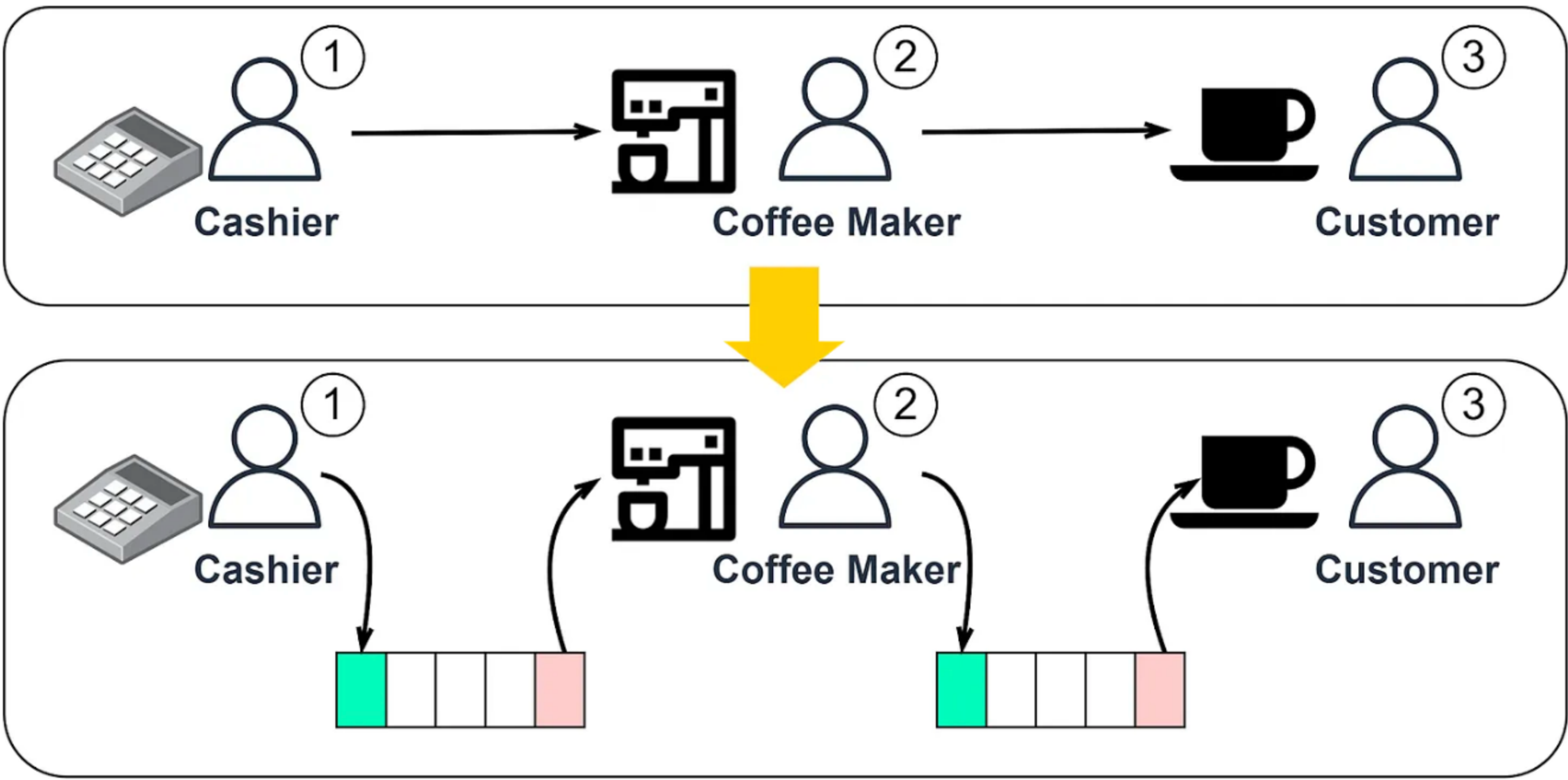


Figure 1 Starbucks as an analogy for message queues

This asynchronous processing, where each step doesn’t have to wait for the previous one to complete, significantly increases the throughput of the system. For instance, the cashier doesn’t wait for your drink to be made before taking another order.

A Message Queue Example

Now, let’s shift our focus to a real-world example: flash sales in e-commerce. Flash sales can strain systems due to surge in user activity. Many strategies are employed to manage this demand, and message queues often play a pivotal role in backend optimizations.

A simplified eCommerce flash sale architecture is listed in Figure 2.

Steps 1 and 2: A customer places an order to the order service.

Step 3: Before processing the payment, the order service reserves the selected inventory.

Step 4: The order service then sends a payment instruction to the payment service. The payment service fans out to 3 services: payment channels, notifications, and analytics.

Steps 5.1 and 6.1: The payment service sends the payment instruction to the payment channel service. The payment channel service talks to external PSPs (Payment Service Providers) to finalize the transaction.

Steps 5.2 and 6.2: The payment service sends a notification to the notification service, which then sends a notification to the customer via email or SMS.

Step 5.3: The payment service sends transaction details to the analytics service.

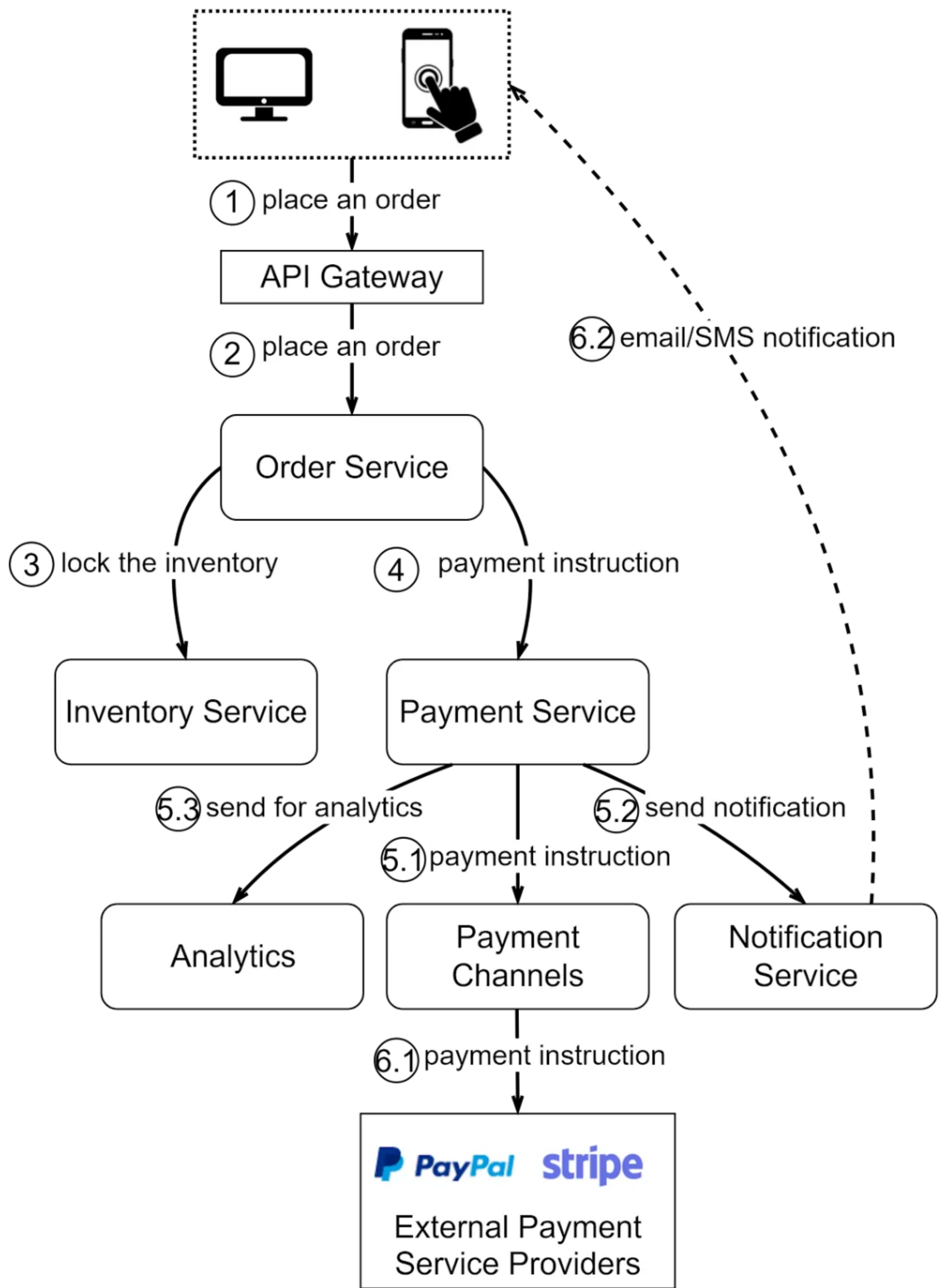


Figure 2 A simplified eCommerce flash sale architecture

A key takeaway here is that a seamless user experience is crucial during flash sales. To maintain service responsiveness despite high traffic, message queues can be integrated at multiple stages to ensure optimal performance.

Benefits of Message Queues

Fan-out

The payment service sends data to three downstream services for different purposes: payment channels, notifications, and analytics. This fan-out method is like someone shouting a message across a room; whoever needs to hear it, does. The producer simply drops the message on the queue, and the consumers process the message at their own pace.

Asynchronous Processing

Drawing from the Starbucks analogy, just as the cashier doesn't wait for the coffee to be made, the order service does not wait for the payments to finalize. The payment instruction is placed on the queue, and the customer is notified once it's finalized.

Rate Limiting

In a flash sale, there can be tens of thousands of concurrent users placing orders simultaneously. It is crucial to strike a balance between accommodating eager customers and maintaining system stability. A common approach is to cap the number of incoming requests within a specific time frame to match the capacity of the system. Excess requests might be rejected or asked to retry after a short delay. This approach ensures the system remains stable and doesn't get overwhelmed. For requests that make it through, message queues ensure they're processed efficiently and in order. If one part of the system is momentarily lagging, the order isn't lost. It's held in the queue until it can be processed. This ensures a smooth flow even under pressure.

Decoupling

Our design uses message queues in various places. The overall architecture is different from the simplified version presented in Figure 2. Services interact with each other using well-defined message interfaces rather than depending tightly on each other. Each service can be modified and deployed independently. Each component can be developed in a different programming language. This brings flexibility to the architectural design.

Horizontal Scalability

Since the services are decoupled, we can scale them independently based on demand. Each service can serve in a different capacity, so we can scale based on their planned QPS (query per second) or TPS (transaction per second).

Message Persistence

Message queues can also be used as middleware that stores messages. If the upstream service crashes, the downstream service can always pick up the messages from the

message queue to process. In this way, the recovery function is moved out of each service and becomes the responsibility of the message queue.

Batch Processing

Sometimes in the processing flow, we need to batch the data to get the summary. For example, when the payment service sends updates to the analytics service, the analytics service does not need to perform real-time updates but rather set up a tumbling window to process in batches. The batch processing is the requirement of the downstream services, so there is no need for the payment service to know about it, just drop the messages into the queue.

Message Ordering

In a flash sale, there is a limited number of inventory items. For example, a flash sale offers only 10 iPhones, but there are over 10,000 users who place the order. How do we decide on the order? Having a message queue to keep all the orders will have a natural order: The first 10 in the queue will get the iPhone.

In Figure 3 we put everything together, where the services are connected via message queues and decoupled. In this way, the architecture can achieve higher throughput.

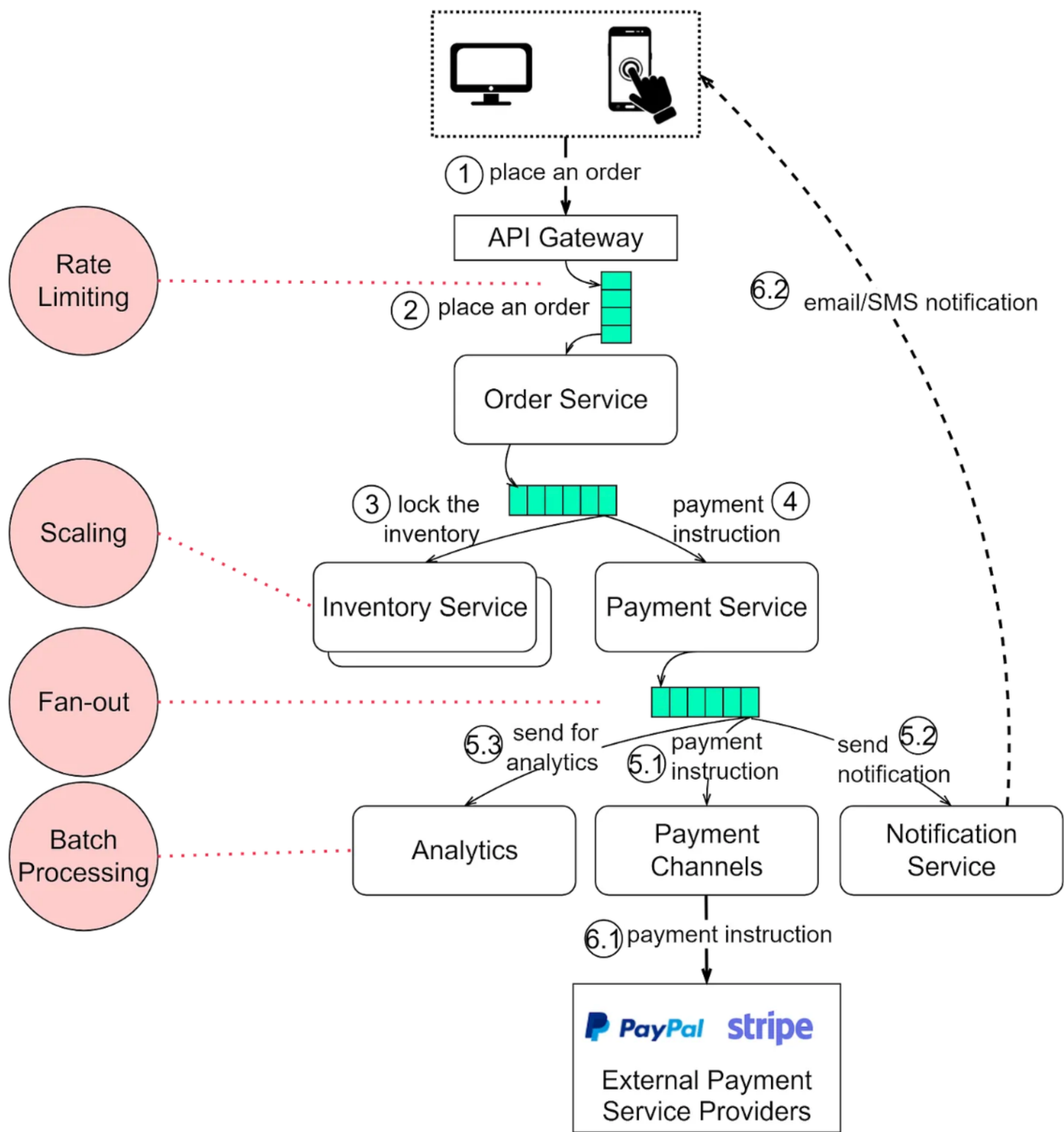


Figure 3 Use message queues for flash sale architecture

Keep reading with a 7-day free trial

Subscribe to **ByteByteGo Newsletter** to keep reading this post and get 7 days of free access to the full post archives.

Start trial

Already a paid subscriber? [Sign in](#)