Microsoft

Azure

What is a message queue and why would I use one?

Clemens Vasters, Twitter: @clemensv

Principal Architect, Microsoft Azure Messaging
Chair OASIS AMQP Technical Committee
Architect CNCF CloudEvents

Messengers


U.S.
MAIL


Rohrpost
HAUPT-
TELEGRAPHENAMT

Queues


https://flic.kr/p/AyAMQA
Giulio Gigante


https://flic.kr/p/bXhNqK

Agenda

What is a queue?

What is a message queue?

What are the common features?

Examples

Function Calls

Local function calls translate to direct commands executed on (a core on) the CPU

```
add(5,4);

int add(int a, int b)
{
    return a + b;
}
```

```
mov rcx, 5
mov rdx, 4
call add

add:
push rbp
mov rbp, rsp
mov rax, rcx
add rax, rdx
mov rsp, rbp
pop rbp
ret
```

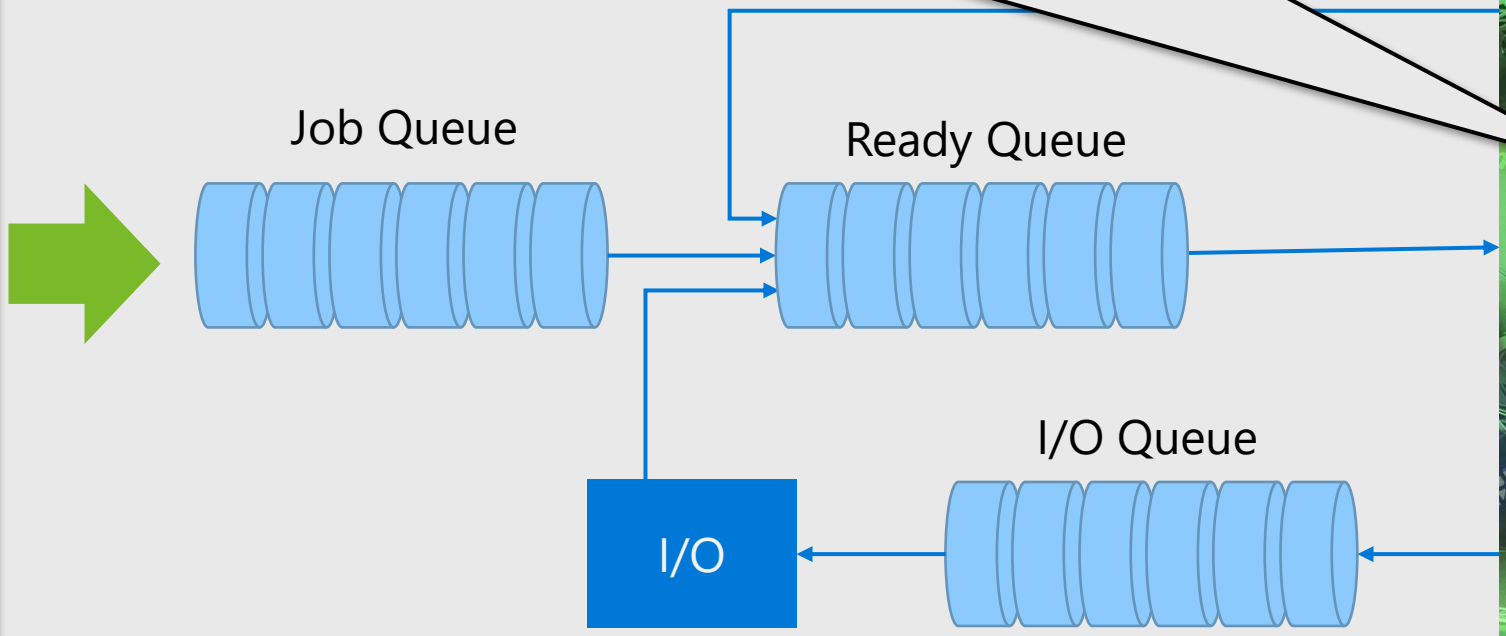
Processes and Threads

Many processes/threads need to share few cores. Programs need to get in line.

```
mov rcx, 5
mov rdx, 4
call add

add:
push rbp
mov rbp, rsp
mov rax, rcx
add rax, rdx
mov rsp, rbp
pop rbp
ret
```

OS Scheduler takes a waiting program (-thread) from the queue, maps it state onto the core, and has it execute for some predetermined time or until any I/O (including main memory access) is required.



Thread Pools

Threads are expensive to create so it's more efficient to share them

```
add(5,4);
add(1,9);
add(6,5);
```

Work Queue

Pool Thread

OS Scheduler takes a waiting program (-thread) from the queue, maps it state onto the core, and has it execute for some predetermined time or until any I/O (including main memory access) is required.

Job Queue

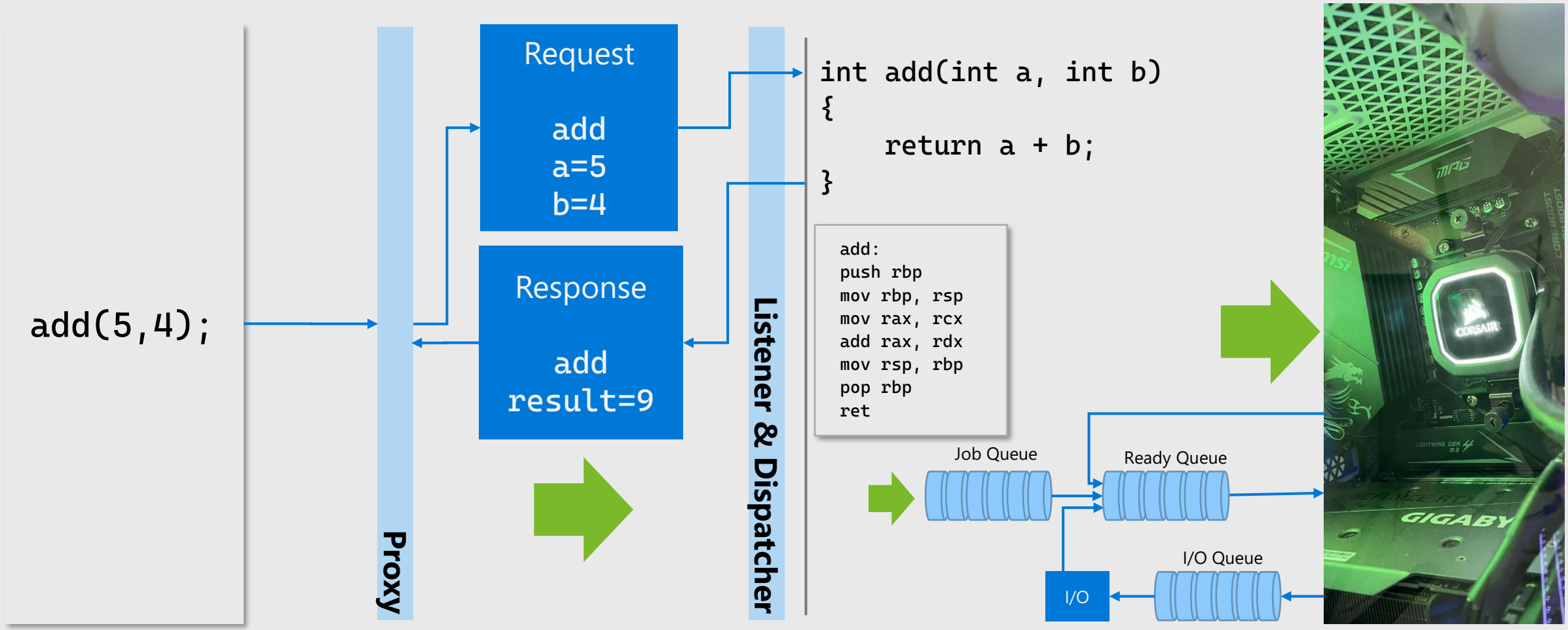
Ready Queue

I/O

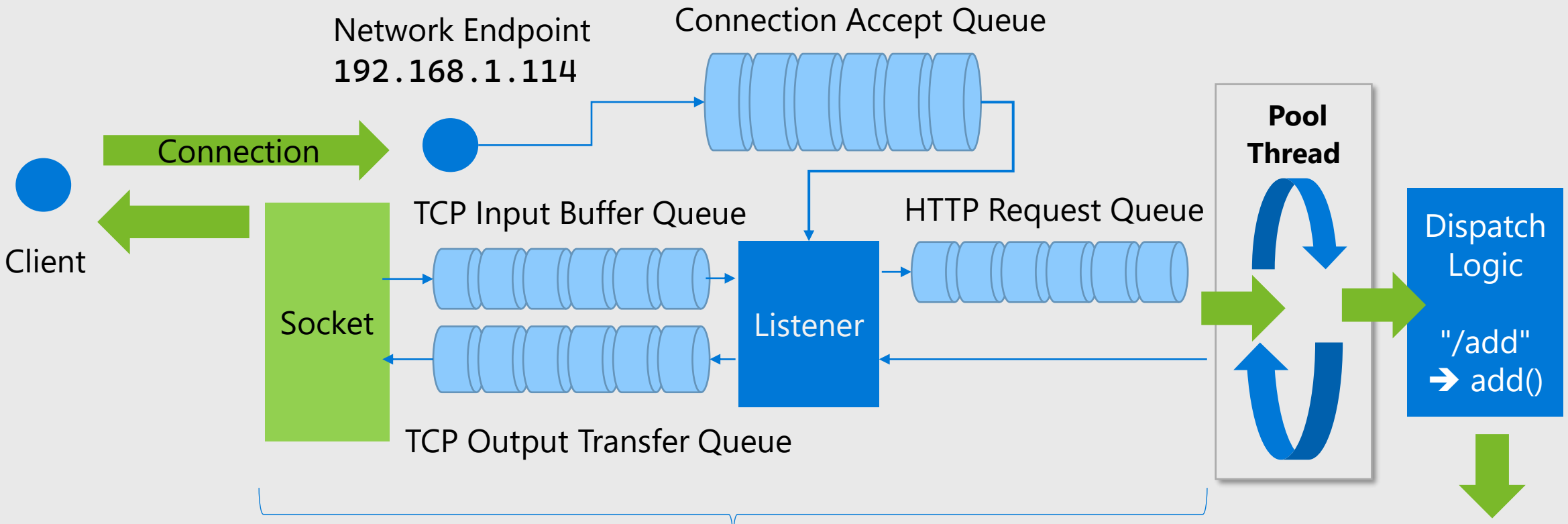
I/O Queue

Remote Function Calls

Remote Function calls translate to messages (requests and responses) transmitted over a network transport



Listener & Dispatcher?




```
int add(int a, int b)
{
    return a + b;
}
```

The client network proxy mirrors of all of this.

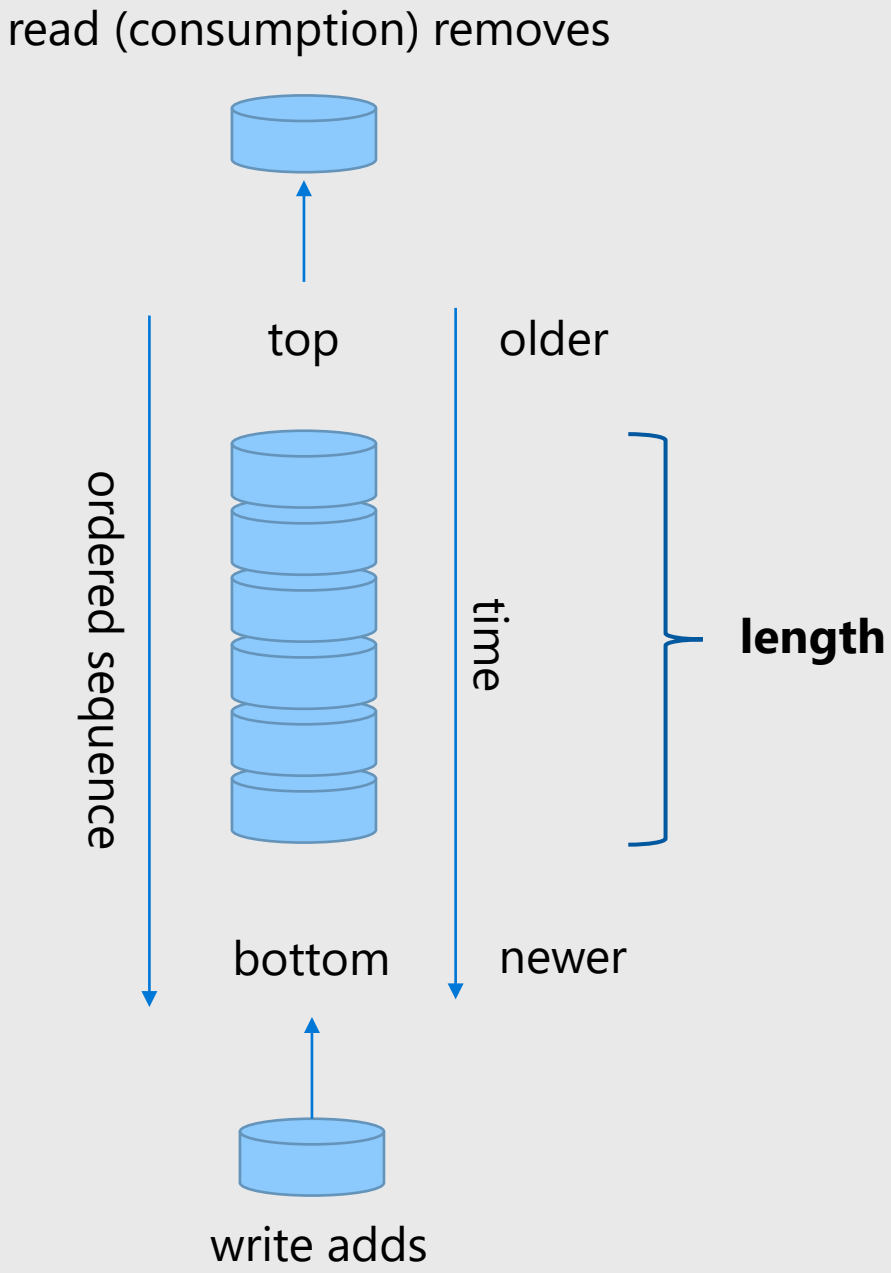
A single "synchronous" API call runs through dozens of queues because operating systems and runtimes use them to enable resource sharing and to scale better ...

Funny, huh?

What is a queue?

The queue is one of most fundamental and most important data structures in computing.

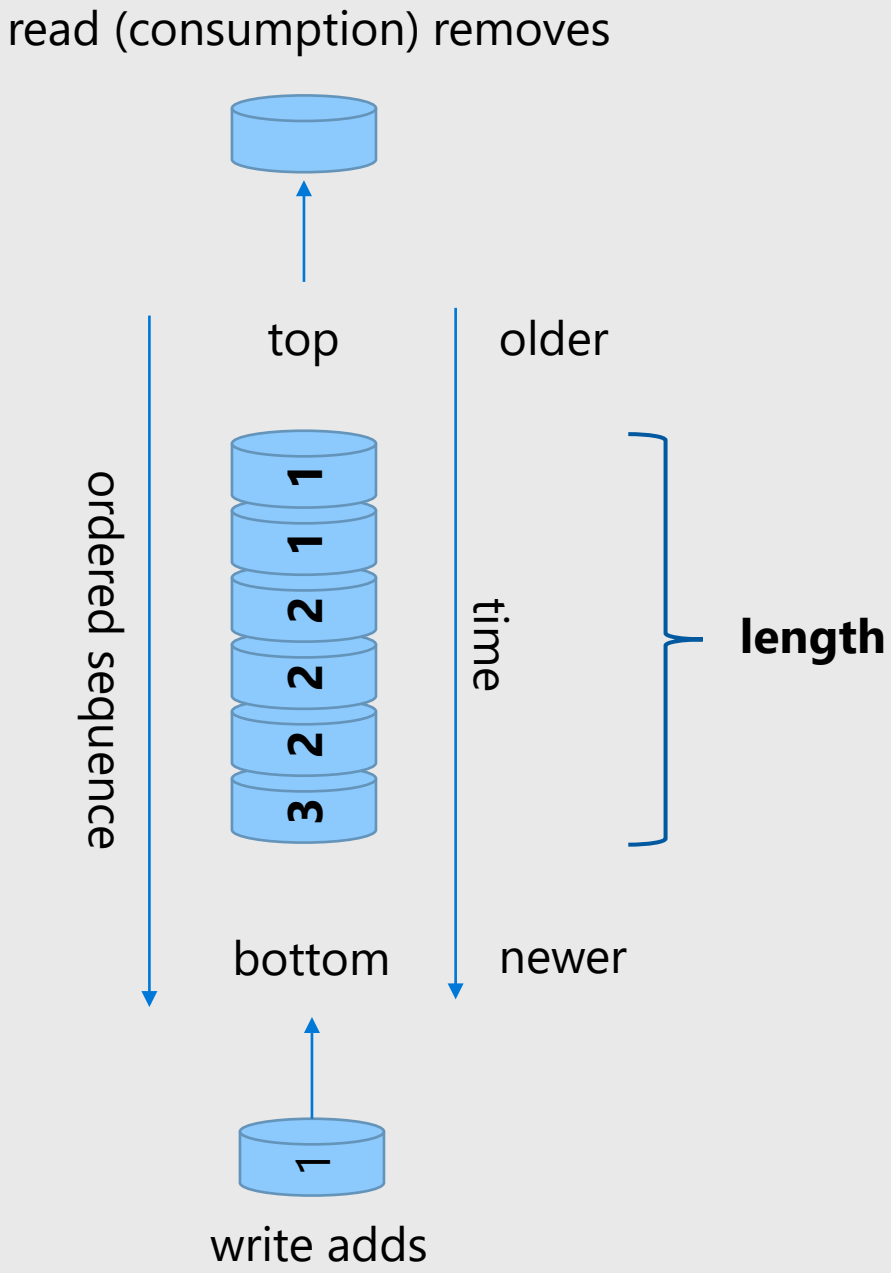
- Sequence of records, commonly ordered by moment of arrival.
- Write/add at the bottom, read/consume from the top.
- Consumed records are removed.
- Length of the queue can always be read.



Variation: Priority queue

A priority queue first orders by a priority indicator attached to the records and then by time.

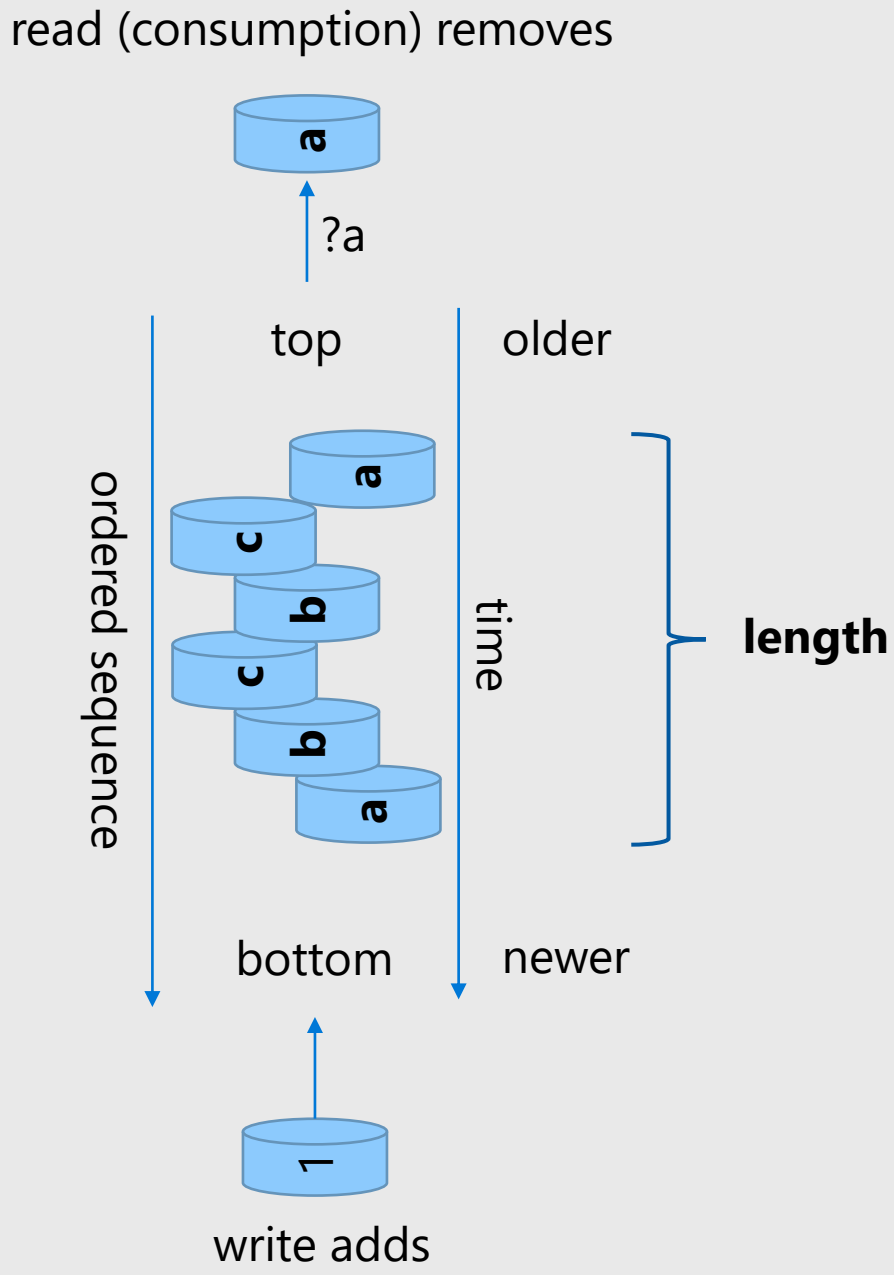
- Higher priorities rush to the top



Variation: Multiplex queue

A multiplex queue maintains arrival order but allows for labeled groups of records that can be read independently.

- Might avoid "head of line blocking"
- Reader will indicate which records it wants to read
- Readers get the oldest available message that matches their filter criterion
- Priority queues are a special multiplex queue



What is a message queue?

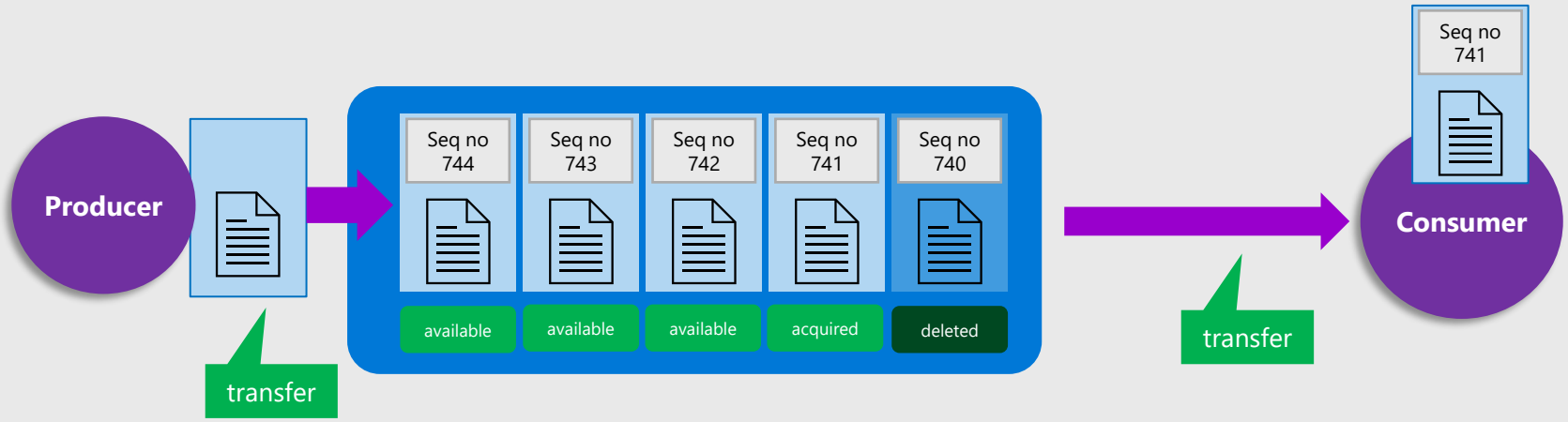
Accepts, stores, and makes messages available for consumption.

Queue individually manages the lifecycle of each message

Accepted, available, acquired, archived/deleted, rejected

Messages are exclusively acquired by one consumer.

The queue length can be queried.



For anyone thinking "Apache Kafka" here ...

Kafka is not a message queue.

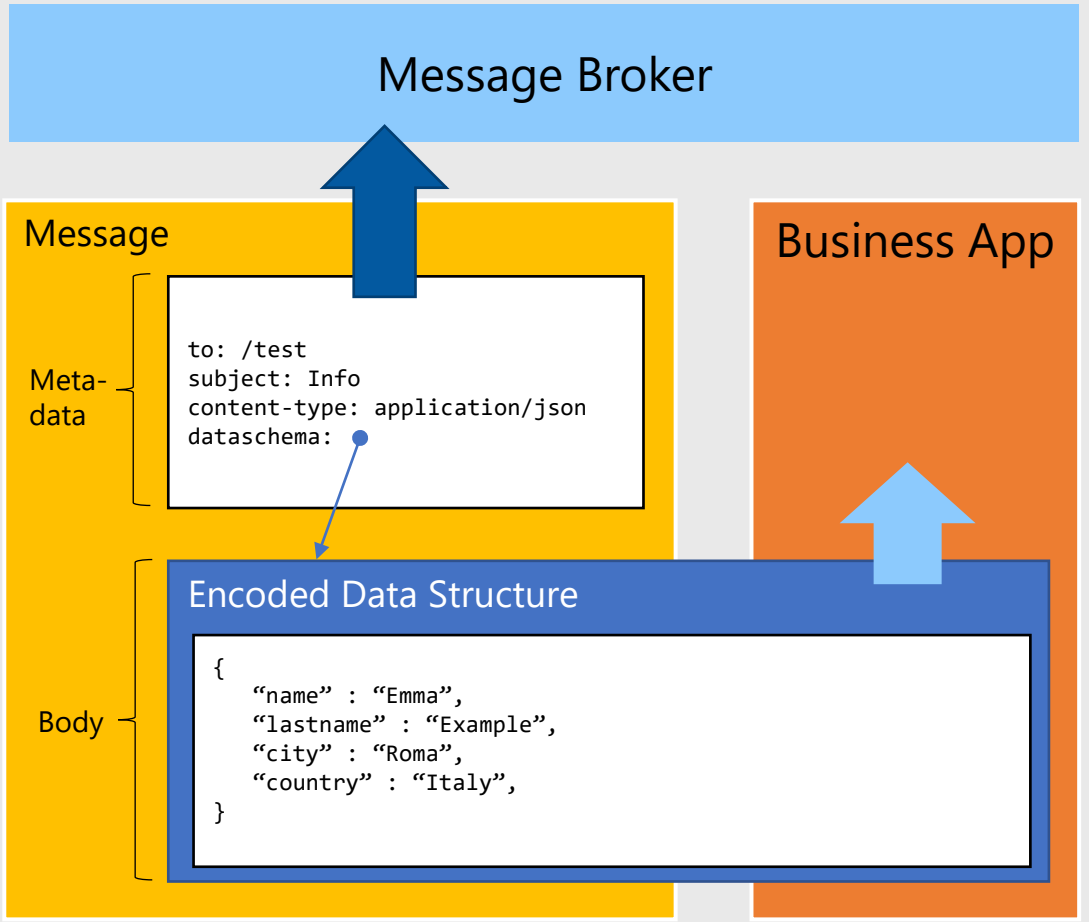
What is a message?

A message is an envelope annotated with metadata around a data structure to be moved between apps or services.

Data is for the apps

Metadata is for the messaging infrastructure and dispatch code.

To, Subject, Content-Type, CorrelationId, …




KOOLIBRI
KOOLIBRI PUBLISHERS Ltd.
Hiiu 38 • 11620 T...nn • ESTONIA
Tähitud
16. 4. 15
EESTI
1386
POST
530032
par avion
Germany
R
TALLINN
RR 30 159 181 5 EE
TÄHITUD - RECOMMANDÉ
VÄLJASTADA ISIKLIKULT - A REMETTRE EN MAIN PROPRE
VÄLJASTUSTEATEGA - AVIS DE RECEPTION
LUNAMAKS
KÄTTETOIMETAMINE TASUTUD
Deutsche Post
EINSCHREIBEN
RN 55 506 140 5DE
R

Send & Receive

Messages are up to the application to design. You can use any encoding (like JSON) for payloads.

```
string connectionString = "<connection_string>";
string queueName = "<queue_name>";
// since ServiceBusClient implements IAsyncDisposable we create it with "await using"
await using var client = new ServiceBusClient(connectionString);

// create the sender
ServiceBusSender sender = client.CreateSender(queueName);

// create a message that we can send. UTF-8 encoding is used when providing a string.
ServiceBusMessage message = new ServiceBusMessage("Hello world!");

// send the message
await sender.SendMessageAsync(message);

// create a receiver that we can use to receive the message
ServiceBusReceiver receiver = client.CreateReceiver(queueName);

// the received message is a different type as it contains some service set properties
ServiceBusReceivedMessage receivedMessage = await receiver.ReceiveMessageAsync();

// get the message body as a string
string body = receivedMessage.Body.ToString();
Console.WriteLine(body);
```

Reactive processing loop

Modern programming models have a standard receive loop built for you that you extend with handlers.

```
// create a processor that we can use to process the messages
await using ServiceBusProcessor processor = client.CreateProcessor(queueName, options);

// configure the message and error handler to use
processor.ProcessMessageAsync += MessageHandler;
processor.ProcessErrorAsync += ErrorHandler;

async Task MessageHandler(ProcessMessageEventArgs args)
{
    string body = args.Message.Body.ToString();
    Console.WriteLine(body);

    // we can evaluate application logic and use that to determine how to settle the message.
    await args.CompleteMessageAsync(args.Message);
}

Task ErrorHandler(ProcessErrorEventArgs args)
{
    // the error source tells me at what point in the processing an error occurred
    Console.WriteLine(args.ErrorSource);
    // the fully qualified namespace is available
    Console.WriteLine(args.FullyQualifiedNamespace);
    // as well as the entity path
    Console.WriteLine(args.EntityPath);
    Console.WriteLine(args.Exception.ToString());
    return Task.CompletedTask;
}

// start processing
await processor.StartProcessingAsync();
```


Financial Services

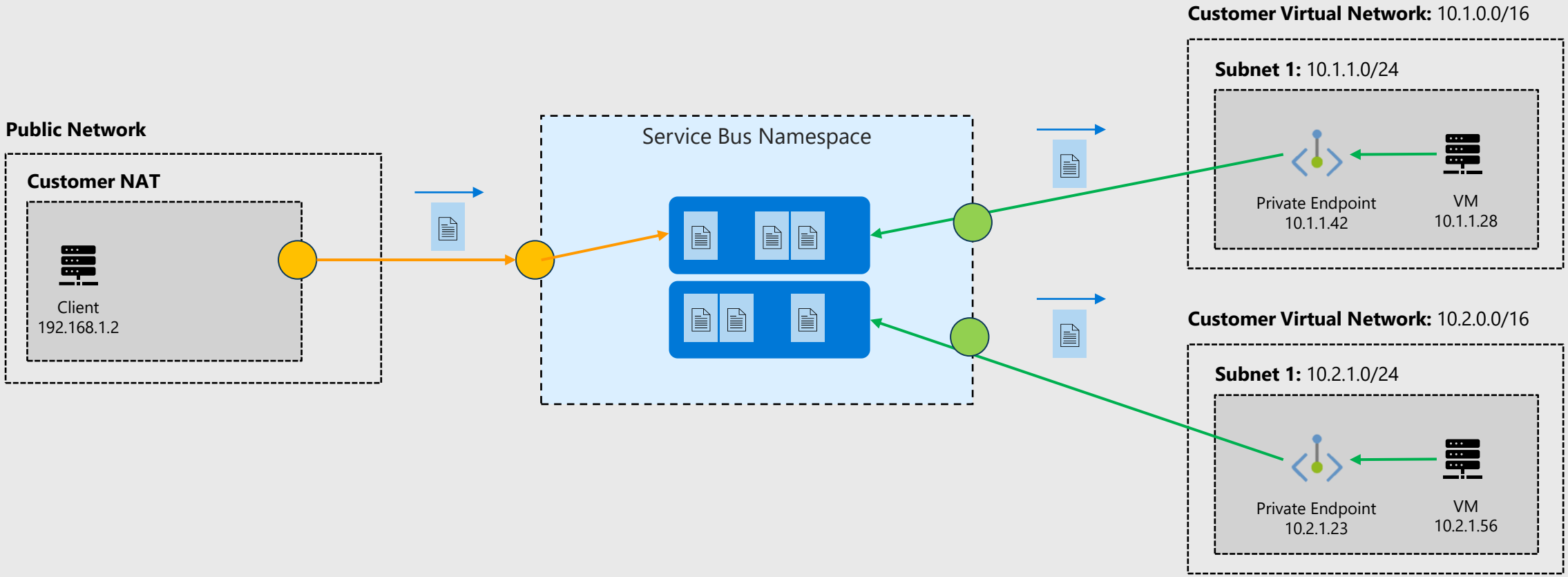

Contoso Bottling Co. – Paris
Production Line 2 – PET Softdrink
Industrial Automation
KRONES
Your Recommendations
Life Cycle Service – Gold
CONTOSO
Work Order
Health
Energy
OEE
Current Order
Next Order


Energy Grid
and Renewable Energy

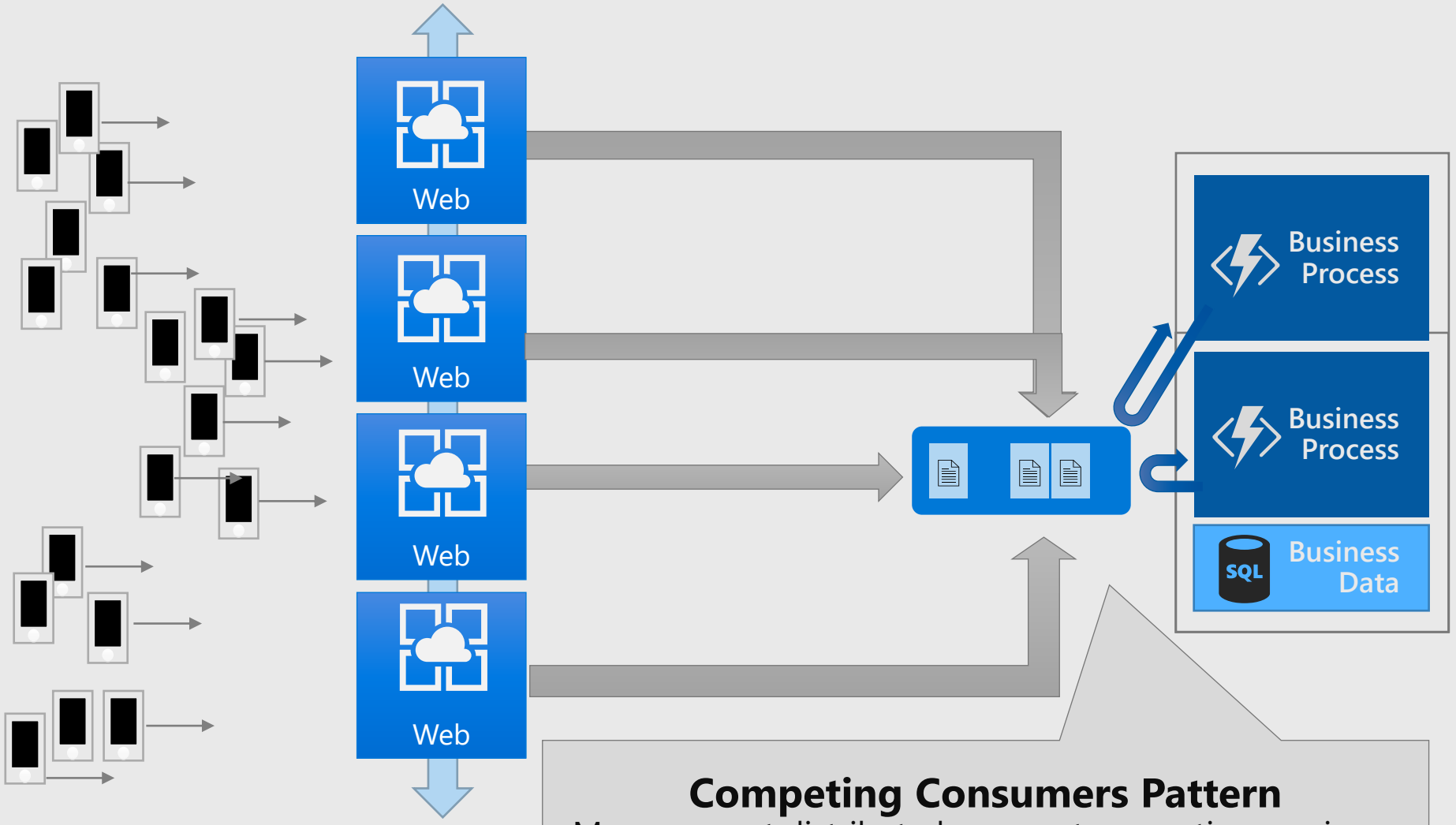

Intelligent Mobility and
Connected Car
1 City Centre
SCANIA

Queues are safe network bridges

Some queue brokers can be attached to one or more virtual networks and the public IP address space concurrently and act as safe "Layer 7" (app-level) routers.



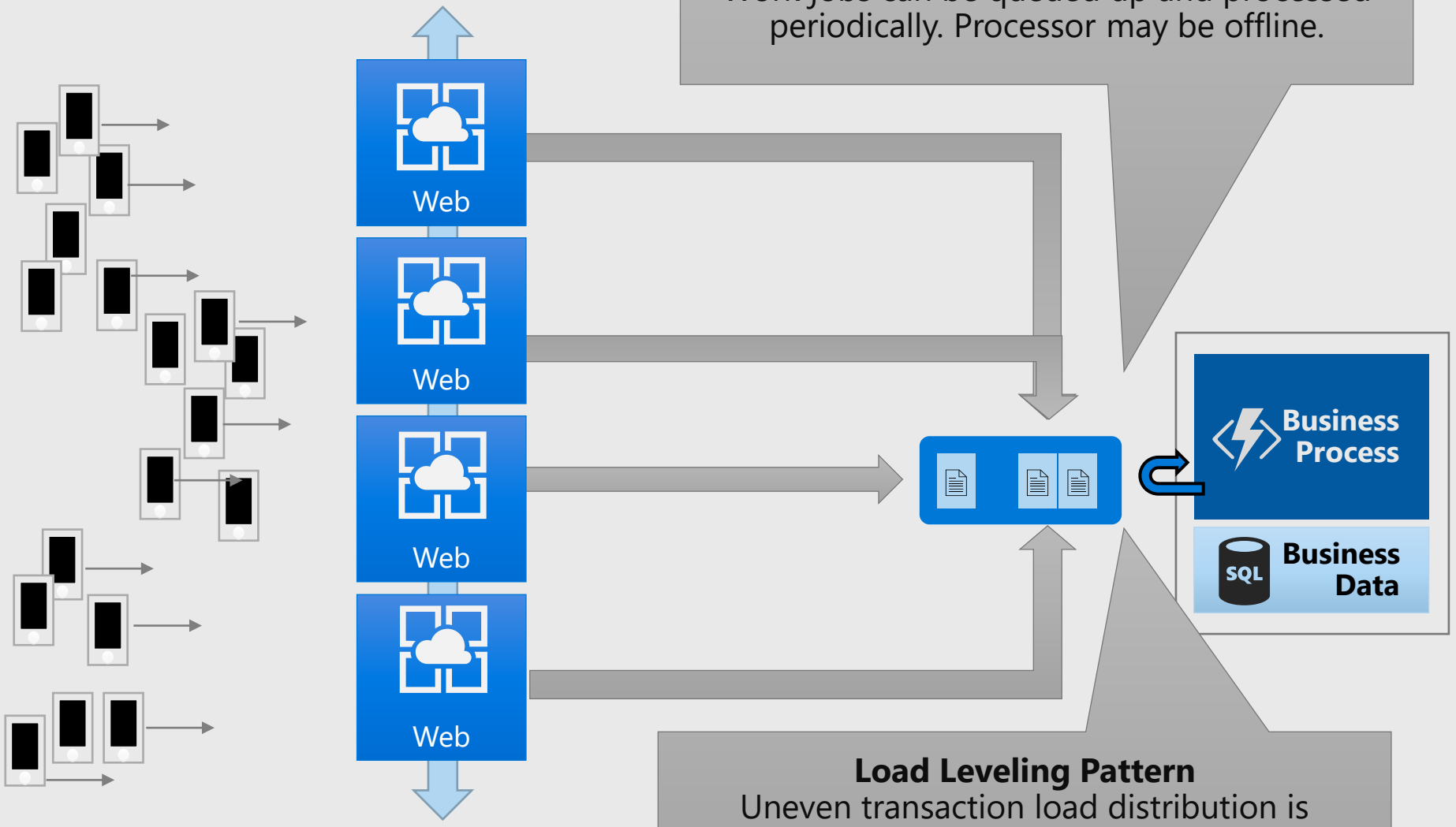
Competing Consumers



Competing Consumers Pattern
Messages get distributed amongst competing receivers.
Exclusive ownership of message.

Observing the queue length trends allows spinning up further resources as needed to handle exceptional load.

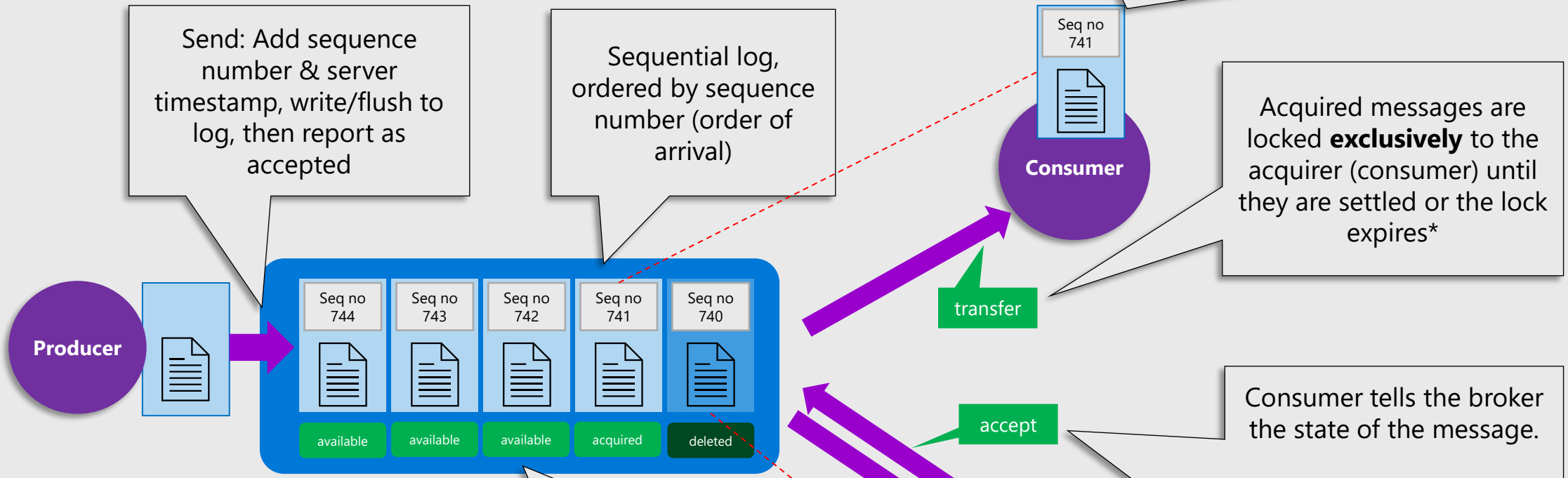
Load Leveling



Adding a message queue allows the business process to handle transactions at optimal capacity use and without getting overwhelmed

Spiky loads are buffered by the queue until the processor can handle them

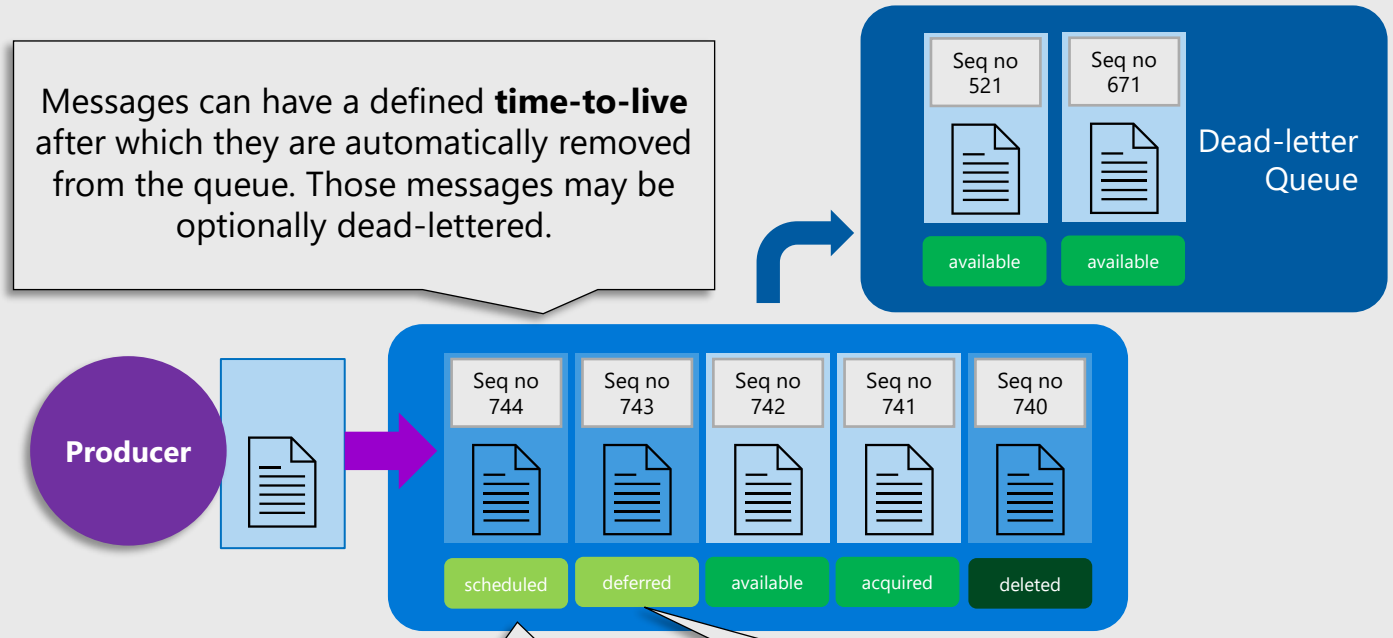
Azure Service Bus Queues (I)



* locks always have a timeout and instantly expire when the connection breaks. (As of 3/22, might change ;)

Azure Service Bus Queues (II)

Messages can have a defined **time-to-live** after which they are automatically removed from the queue. Those messages may be optionally dead-lettered.



Every queue and topic subscription has its own **dead-letter subqueue**: myqueue/$deadletterqueue

All messages that cannot be delivered (released more often than permitted) or have been rejected and optionally those that have expired and put into the dead-letter queue and remain there.

The reason for why and from where the message has been moved is remarked in the **DeadLetterReason**, **DeadLetterErrorDescription**, and **DeadLetterSource** properties.

The deadletter queue is otherwise a normal queue and can be used with the normal receiver clients.

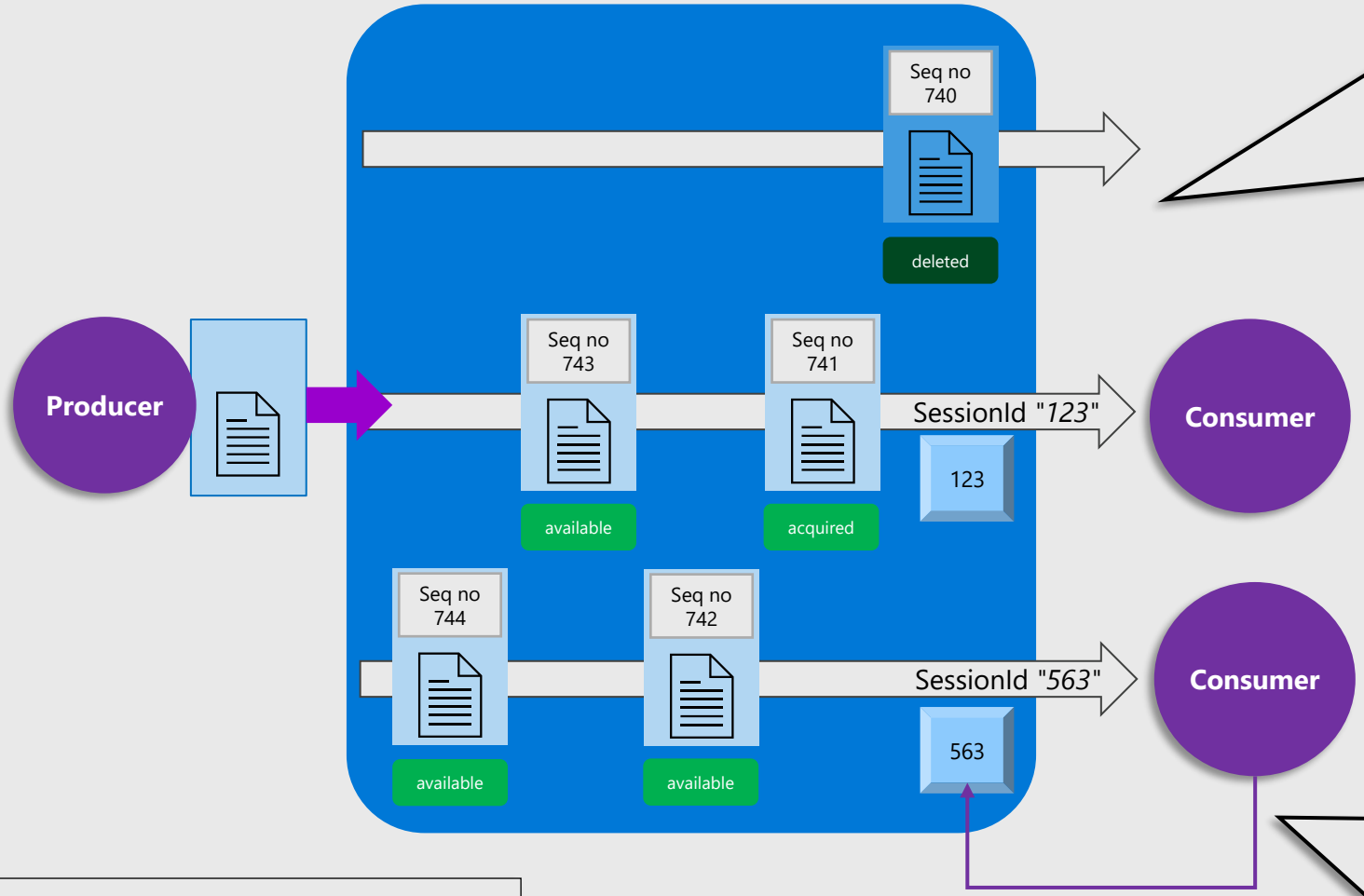
Scheduled messages have been accepted and stored in the queue log, but they are made available for delivery only after **ScheduledTimeUtc**.

Scheduled messages *can be canceled* using the sequence number returned by the scheduling API.

Deferred messages have been delivered to a consumer at least once and the consumer has decided to set them aside instead of settling them.

Deferred messages remain in the log but are not eligible for delivery until they are restored into the available state.

Azure Service Bus Queues – Sessions



Sessions are unlimited, unbounded, ordered sequences of messages that are all labeled with the same **SessionId**. Using sessions requires a session-enabled queue or topic subscription.

Consumers acquire exclusive locks on the entire session and all messages belonging to it.

An active session behaves like an independent queue, without messages from other sessions blocking progress (no "head of line blocking")

A single queue can maintain hundreds of thousands of sessions and thus logical sub-queues.

The owning consumer of a session has read/write access to the **session state** that the broker optionally maintains. Session state can hold data up to the max size of a message permitted for the queue.

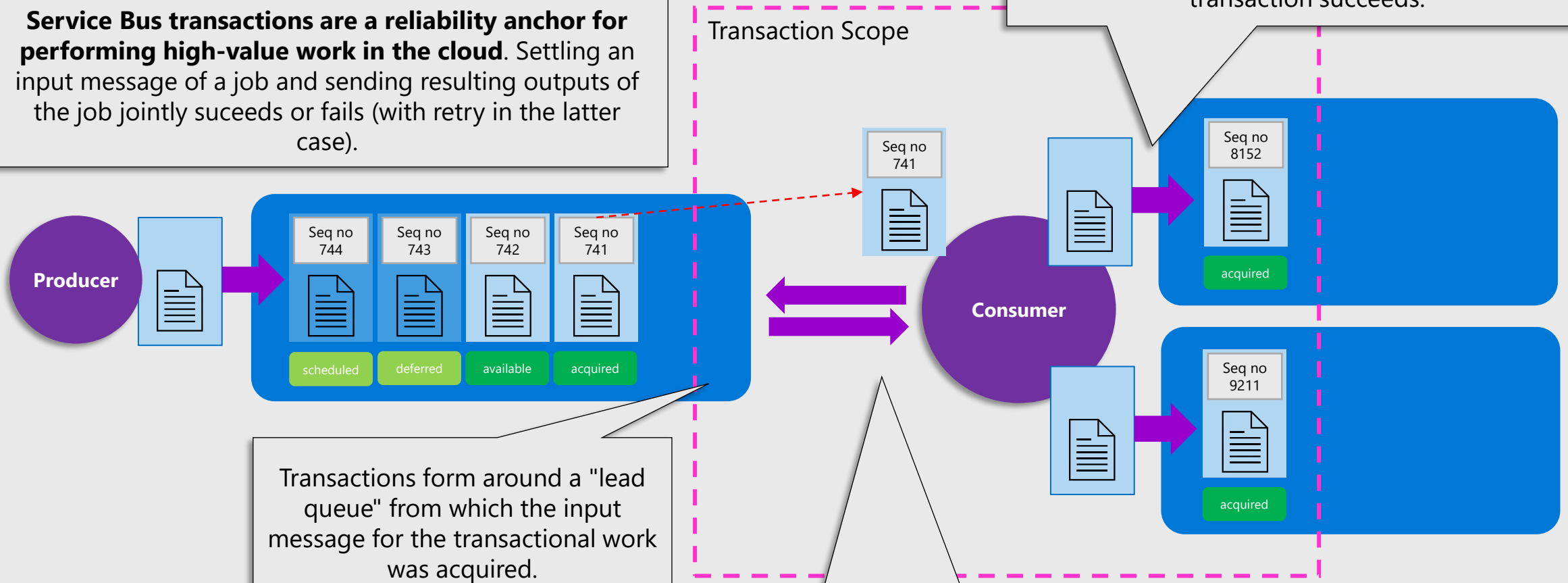
When a new consumer acquires the session, it can therefore access all info left by a prior consumer related to processing the session.

Built for long-running workflow processes.

Azure Service Bus Queues – Transactions

Service Bus transactions are a reliability anchor for performing high-value work in the cloud. Settling an input message of a job and sending resulting outputs of the job jointly suceeds or fails (with retry in the latter case).

A transactional operation can send messages to any number of output queues or topics as a result of the transactional work. Sends are held until the transaction succeeds.



Transactions form around a "lead queue" from which the input message for the transactional work was acquired.

Only one receiving queue can be enlisted in a transaction. Sessions are supported.

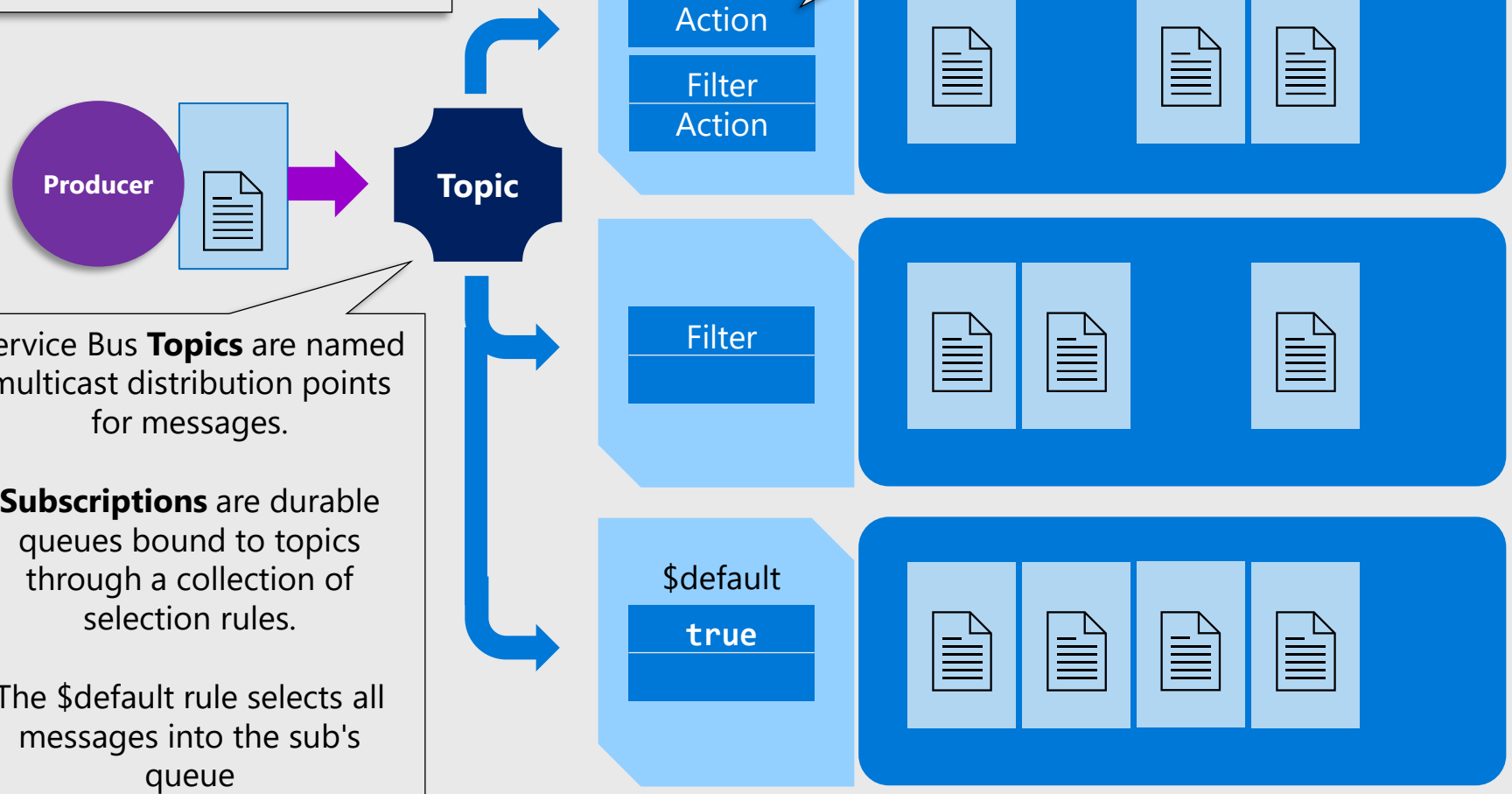
All settlement operations on the lead source queue are transactional: accepting (complete), releasing, rejecting (deadletter), deferral, session state ops.

Azure Service Bus Topics

For consumers, subscription queues have all features of queues, including a dead-letter queue. They can also support sessions.

Actions can add, remove, and edit all application message properties and system properties like TimeToLive or SessionId.

One subscription can have up to 2000 (!) rules. Each filter match yields a copy of the tested message.



Correlation filters match properties against values ("equals").

SQL filters allow WHERE-condition-style SQL expressions against the app and platform message properties.

```
sys.To LIKE
'region/DE/shops/%' AND
subject='catalogUpdate'
```

Service Bus **Topics** are named multicast distribution points for messages.

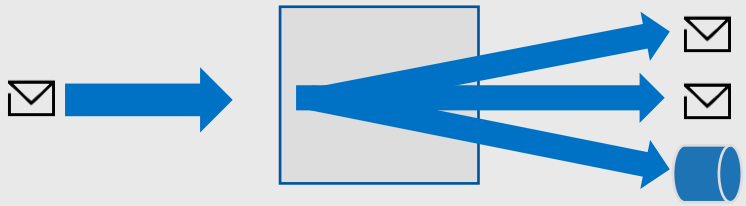
Subscriptions are durable queues bound to topics through a collection of selection rules.

The $default rule selects all messages into the sub's queue

JMS clients can use shared, unshared, durable and volatile subscriptions, with JMS-compliant SQL message selectors.

Discrete Event Router

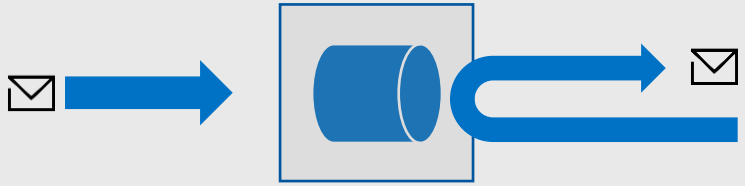
Azure Event Grid, AWS Event Bridge, Knative Eventing



Push-style distribution of discrete events to serverless workloads or other messaging infrastructures

Queue Pub/Sub Broker

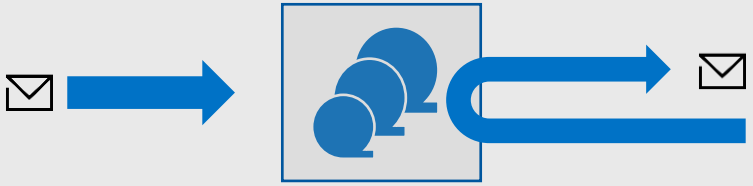
Azure Service Bus, AWS SQS/SNS, Google PubSub, Apache ActiveMQ, RabbitMQ, IBM MQ



Pull-style, queue-based transfer of jobs and control via message queues and topics

Event Stream Engine

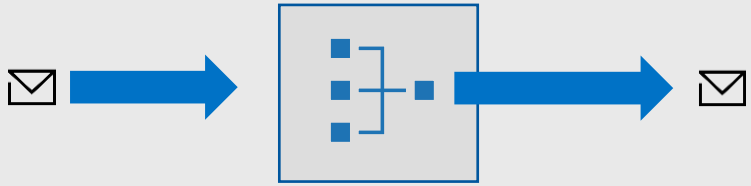
Azure Event Hubs, AWS Kinesis, Apache Kafka, Apache Pulsar, CNCF Pravega



Partitioned, high-volume, tape-drive-style sequential recording and unlimited, pull-style re-reads of event streams.

Event Stream Aggregator

Azure Stream Analytics, AWS Kinesis Analytics, Apache Samza, Apache Flink, etc.



Stateful processing of event streams yielding event streams and discrete events as continuous output

Event Streaming is not "modern" and Queues are not "traditional"

Both are patterns of state-of-the art messaging infrastructures.

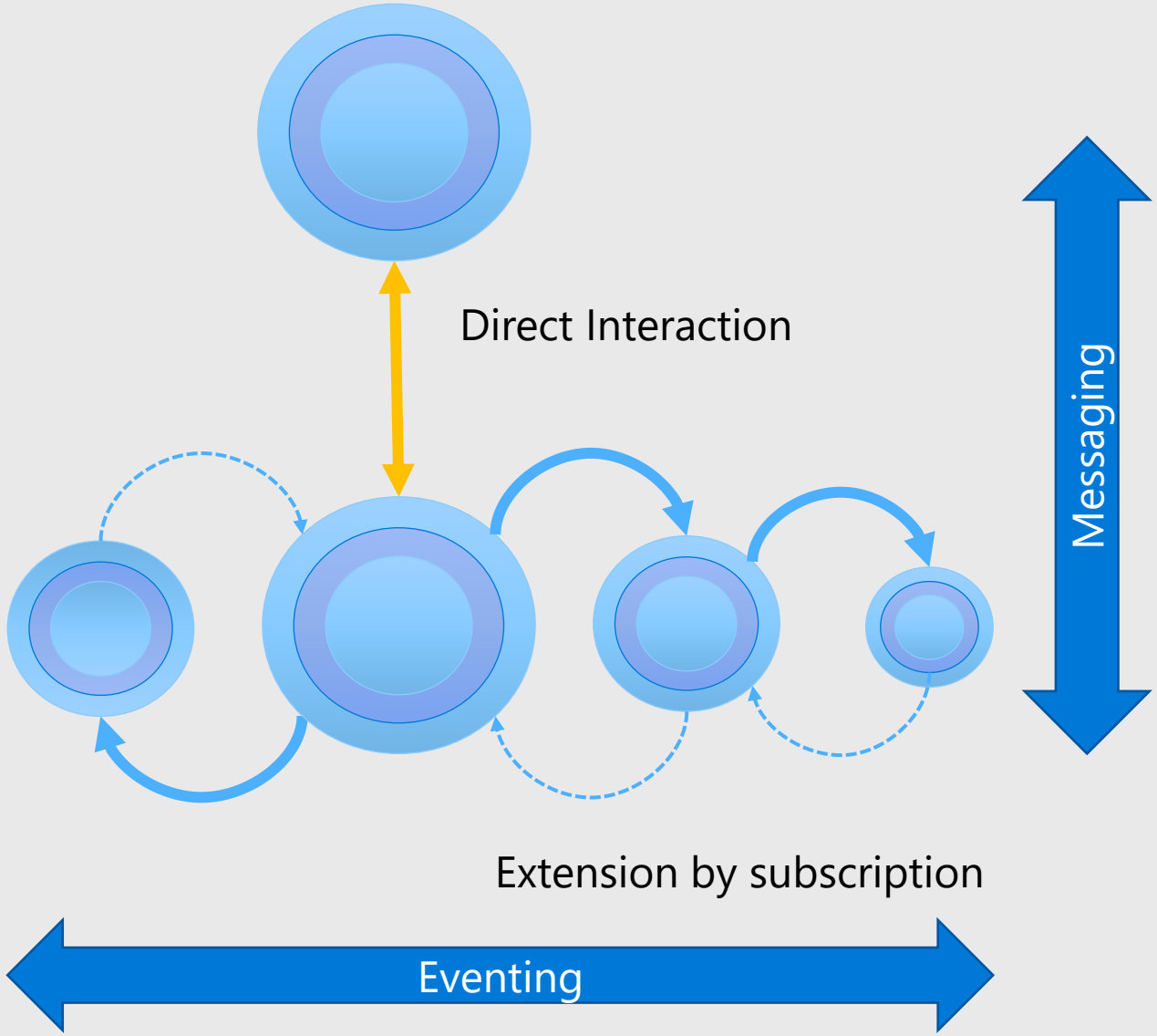
Modern apps use eventing and queue-based messaging

“Core” functions of services require direct, point-to-point, RPC or queue-based interaction:

Imperative: Commands, Requests

Extensions react to events or insights derived from event streams emitted by services.

Might turn to the emitting service to ask for details or perform actions.



© Copyright Microsoft Corporation. All rights reserved.

Creative Commons Attributions

https://commons.wikimedia.org/wiki/File:Junge_Frau_mit_Taubenpost.jpg

https://commons.wikimedia.org/wiki/File:The_Royal_Engineers_Signals_Service_on_the_Western_Front,_1914-1918_Q8877.jpg

https://commons.wikimedia.org/wiki/File:Pony_ExpressAdvert.jpg

https://commons.wikimedia.org/wiki/File:Phidippides.jpg

https://commons.wikimedia.org/wiki/File:Bundesarchiv_Bild_183-S99696,_Berlin,_Haupttelegrafenamt,_Motorradbote.jpg?uselang=de

https://commons.wikimedia.org/wiki/File:Photographic_copy_of_retouched_photograph_%28circa_1918,_original_print_in_Archives,_Public_Affairs_Department,_Sears_Merchandise_Group,_Hoffman_Estates,_Illinois%29._Photographer_HABS_ILL,16-CHIG,110A-85.tif

https://commons.wikimedia.org/wiki/File:Airmail_1930s_Detroit_Smykowski.jpg

https://commons.wikimedia.org/wiki/File:Soldat_in_der_Empfangsstation_f%C3%BCr_drahtlose_Telegraphie_-_CH-BAR_-_3239488.tif

https://commons.wikimedia.org/wiki/File:Registered_mail_from_Estonia_to_Germany.jpg?uselang=de

https://commons.wikimedia.org/wiki/File:HollerithMachine.CHM.jpg

http://bitsavers.trailing-edge.com/pdf/ibm/360/os/qtam/C28-6553-2_Telecommunications_Preliminary_Specifications_Dec65.pdf

https://commons.wikimedia.org/wiki/File:Gie%C5%82da_na_Wall_Street.JPG