Parallel and Distributed Programming

Hello!

I am Diego Bonura

Mi occupo di:

- Frontend
- Backend
- Mobile
- IoT
- R&D

LOCCIONI

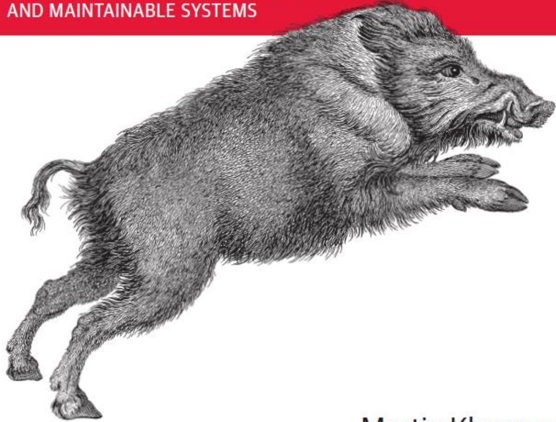
diego@bonura.dev

https://medium.com/@diegobonura

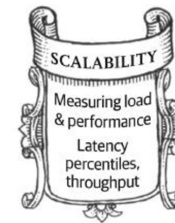
O'REILLY®

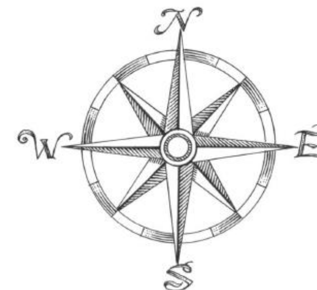
Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS

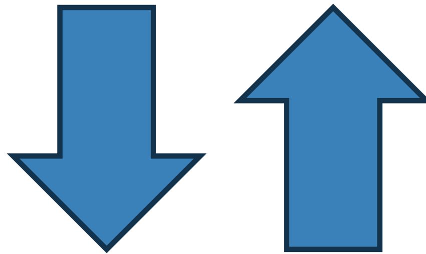


Martin Kleppmann



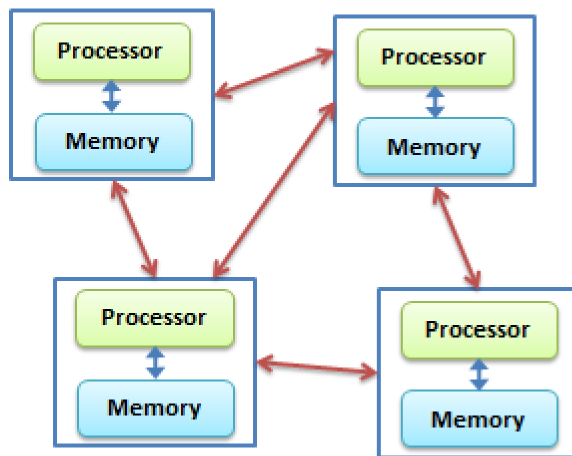


Distribuited programming is complex

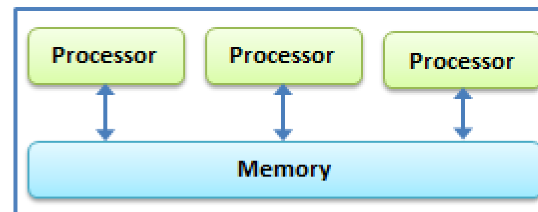


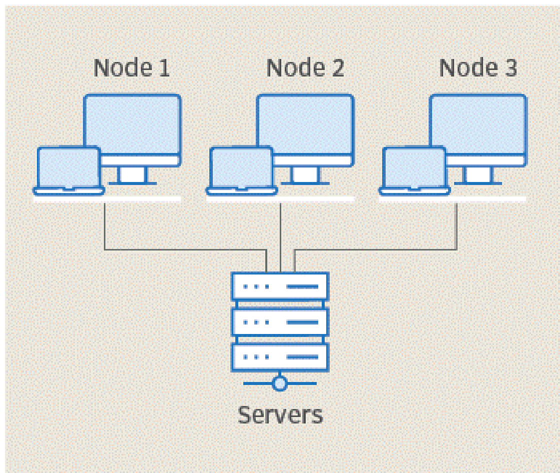
Use only on complex applications

Distributed Computing

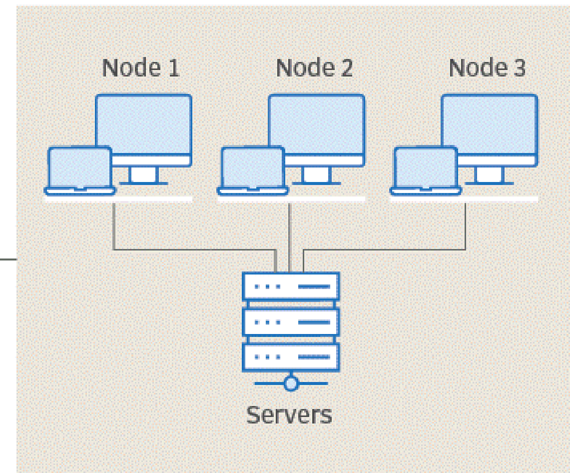


Parallel Computing




Node 1
Node 2
Node 3
Servers

Network


Node 1
Node 2
Node 3
Servers

Why?

- **Performance**
  - Maintains System Performance During High Demand Periods
  - Adapts to the Increase/Decrease Workloads and User Demands
- **Scalability**
  - Boosts Performance and Utilization through Collaboration
- **Resilience**
  - Ensures System Continuity in the Face of Failures
- **Redundancy**
  - Enhances User Experience with Geographically Distributed Systems

https://youtu.be/CZ3wIuvmHeM?si=eHIQEqZkHpZWhHDm&t=604

How?

Main types:

- Cluster Computing
  - https://www.mongodb.com/basics/clusters
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/high-availability.html
- Grid computing
  - https://en.wikipedia.org/wiki/Great_Internet_Mersenne_Prime_Search
  - https://en.wikipedia.org/wiki/SETI@home
- Cloud computing
  - https://www.linkedin.com/pulse/how-cloud-computing-made-netflix-possible-keimo-edwards/
  - https://cloudacademy.com/blog/aws-reinvent-netflix/
- Peer-2-Peer
  - Torrent
  - Bitcoin

Example of complex system?

Two of Twitter's main operations are:

Post tweet

- A user can publish a new message to their followers (4.6k requests/sec on average, over 12k requests/sec at peak).

Home timeline

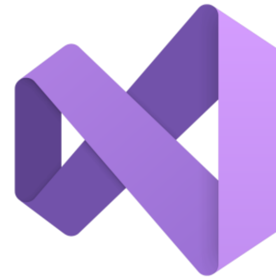
- A user can view tweets posted by the people they follow (300k requests/sec)….
- ….

Continue to book «Designing Data-Intensive Applications» page 11

Main agenda

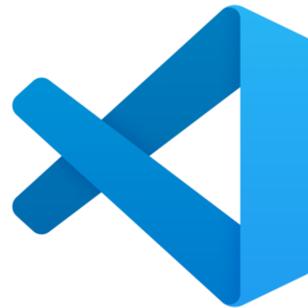
- Object oriented programming (message passing)
- Async programming
- In-process / out-of-process programming
- Distributed programming
  - Message brokers
  - Actor Model
  - Serialization
  - Transaction
  - Saga
  - Idempotent operations
  - Stream processing
  - Event sourcing
- Deploy a distributed application
- Infrastructure as code
- Update and maintain
- Observability

How to start?



https://visualstudio.microsoft.com/it/vs/community/

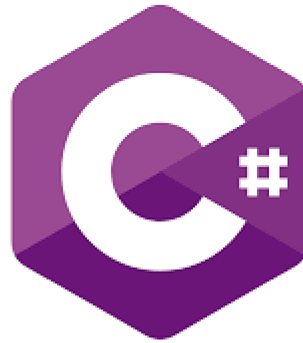
or



https://code.visualstudio.com/

https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csdevkit

How to start?



https://github.com/meriturva/Parallel-and-Distributed-Programming

Message Passing

Message passing is a technique for invoking behavior

```
public class Producer
{
    public void Start()
    {
        var consumer = new Consumer();
        int i = 0;
        while (true)
        {
            var result = consumer.Elaborate(i, i);
            Console.WriteLine($"Counter: {i} with result: {result}");
            i++;
        }
    }
}
```

Example project: 01 MessagePassing

https://en.wikipedia.org/wiki/Message_passing

Async programming

Code run in the background while other code is executing.

```
public class Producer
{
    public async Task StartAsync()
    {
        var consumer = new Consumer();
        int i = 0;
        while (true)
        {
            var result = await consumer.ElaborateAsync(i, i);
            Console.WriteLine($"Counter: {i} with result: {result}");
            i++;
        }
    }
}
```

Example project: 02 AsyncAwait

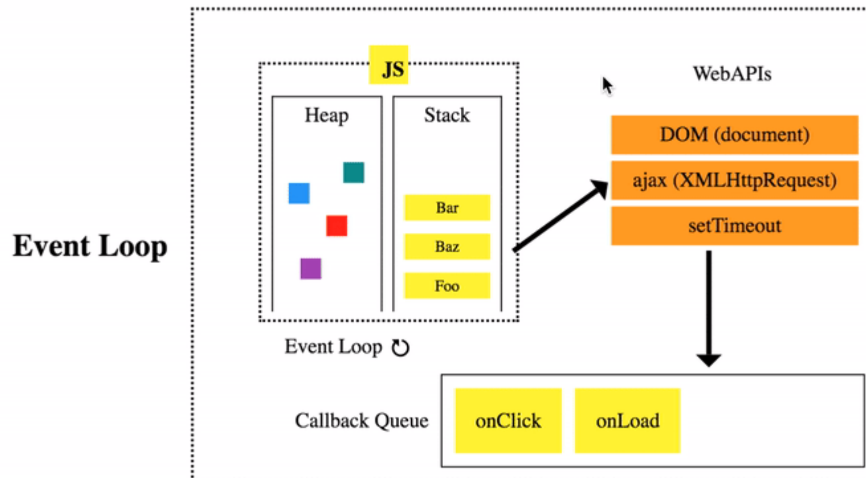
On the C# side of things, the compiler transforms your code into a state machine that keeps track of things like yielding execution when an await is reached and resuming execution when a background job has finished.

https://learn.microsoft.com/en-us/dotnet/csharp/asynchronous-programming/async-scenarios

Async programming (on single thread)

JavaScript is a single-thread!

```
async function doWork()
{
    console.log("frist");
    await wait(1000);
    console.log("second");
}

doWork();
```



https://www.youtube.com/watch?v=8aGhZQkoFbQ

Javascript – Callback and Promise

loupe help

Save + Run

```
1
2
3 function printHello() {
4     console.log('Hello from baz');
5 }
6
7 function baz() {
8     setTimeout(printHello, 3000);
9 }
10
11 function bar() {
12     baz();
13 }
14
15 function foo() {
16     bar();
17 }
18
19 foo();
```

Click me!

Edit

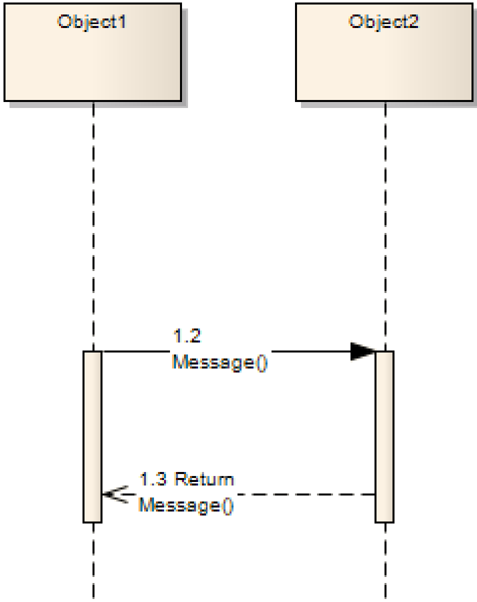
Call Stack

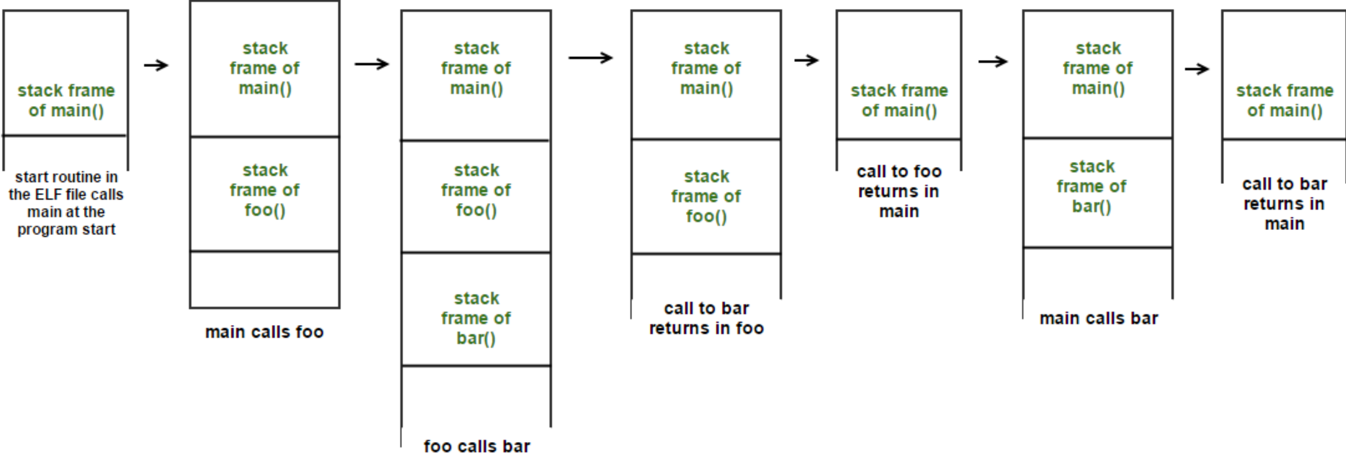
Web Apis

Callback Queue

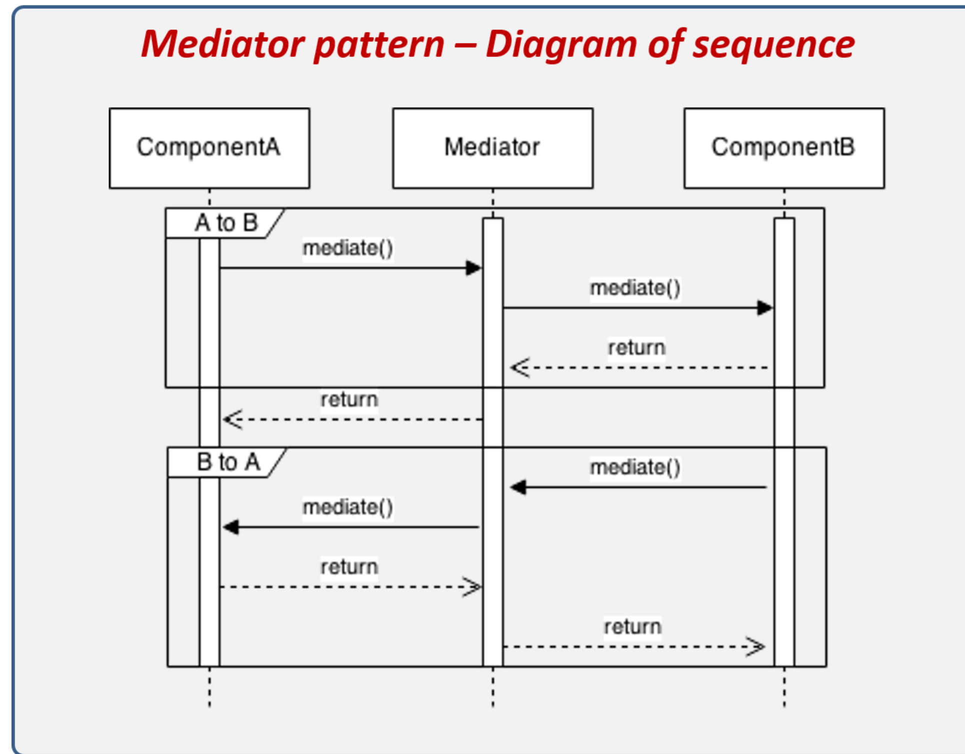
https://latentflip.com/loupe/

In-process / sync





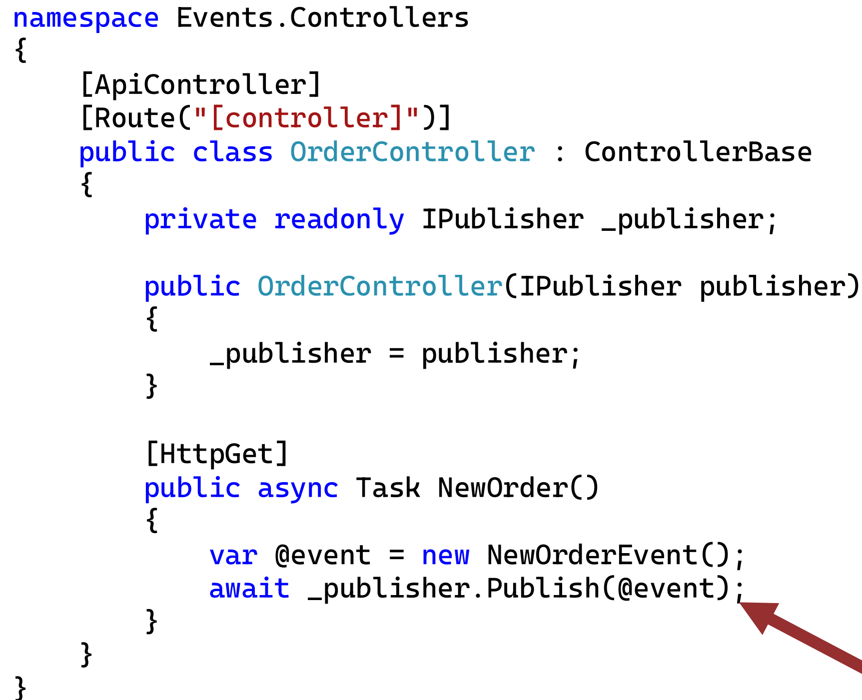
In-process / sync with mediator pattern



Objects no longer communicate directly with each other, but instead communicate through the mediator. This reduces the dependencies between communicating objects, thereby reducing coupling.

https://en.wikipedia.org/wiki/Mediator_pattern

In-process / sync with mediator pattern



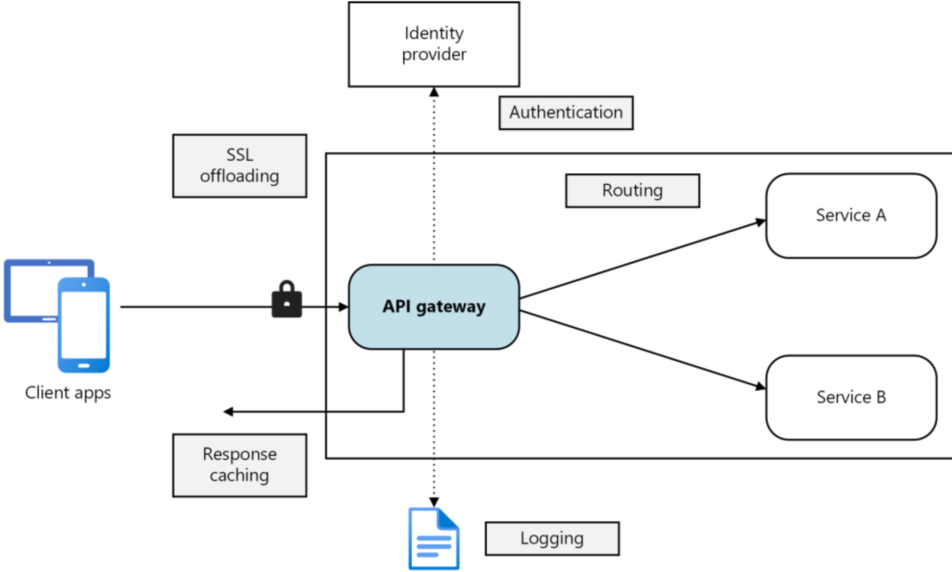
```
namespace Events.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class OrderController : ControllerBase
    {
        private readonly IPublisher _publisher;

        public OrderController(IPublisher publisher)
        {
            _publisher = publisher;
        }

        [HttpGet]
        public async Task NewOrder()
        {
            var @event = new NewOrderEvent();
            await _publisher.Publish(@event);
        }
    }
}
```

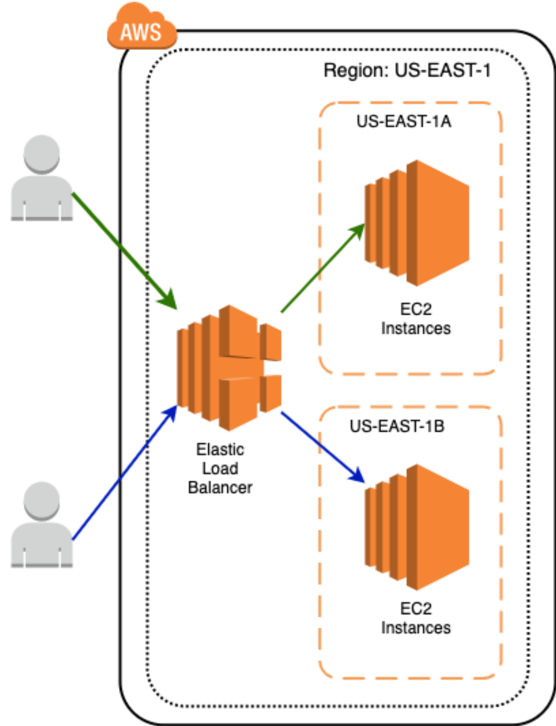
Example project: 03 EventsInProcessByMediator

In-process / sync with mediator pattern

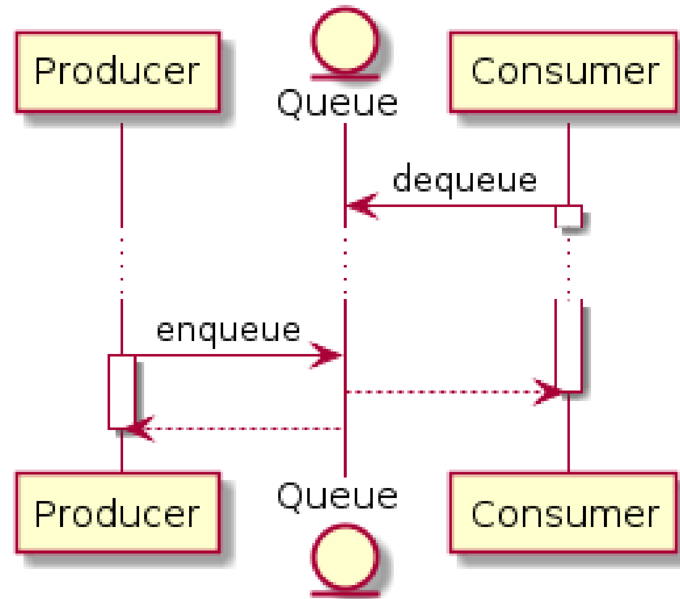
Performance
Scalability
Resilience
Redundancy

?

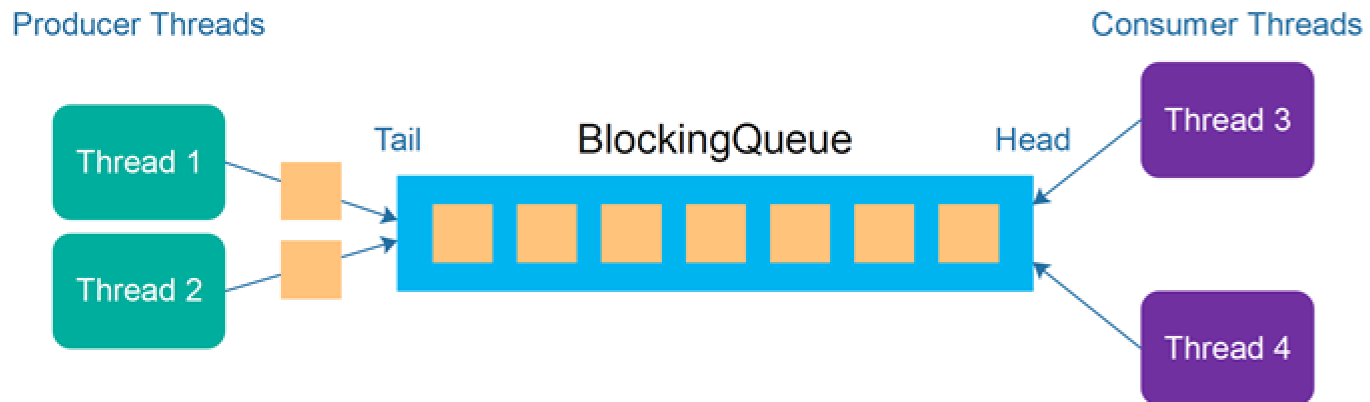




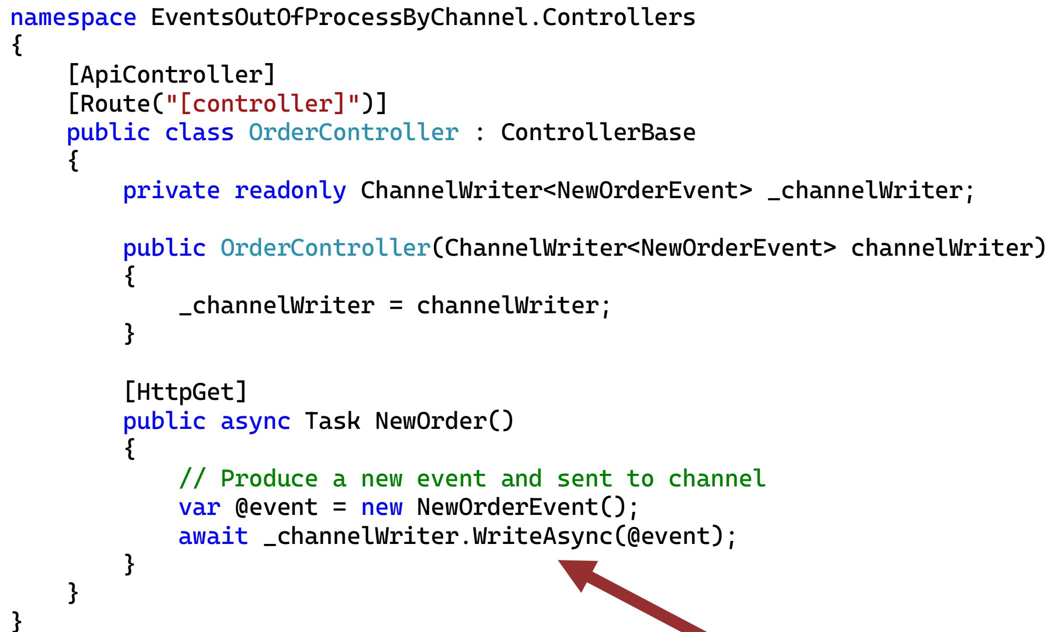
Out of process / async



Out of process / async with producer/consumer



Queue Producer



```
namespace EventsOutOfProcessByChannel.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class OrderController : ControllerBase
    {
        private readonly ChannelWriter<NewOrderEvent> _channelWriter;

        public OrderController(ChannelWriter<NewOrderEvent> channelWriter)
        {
            _channelWriter = channelWriter;
        }

        [HttpGet]
        public async Task NewOrder()
        {
            // Produce a new event and sent to channel
            var @event = new NewOrderEvent();
            await _channelWriter.WriteAsync(@event);
        }
    }
}
```

C# Channels are an implementation of the producer/consumer programming model.

https://learn.microsoft.com/en-us/dotnet/core/extensions/channels

Example project: 04 EventsOutOfProcessByChannel

Queue Consumer

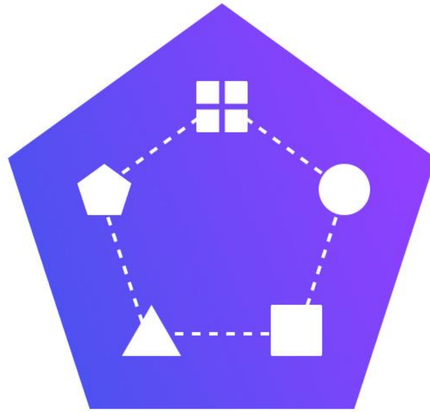
```
namespace EventsOutOfProcessByChannel
{
    public class Consumer
    {
        public static async ValueTask ConsumeWithWhileAsync(ChannelReader<NewOrderEvent> reader)
        {
            while (true)
            {
                var @event = await reader.ReadAsync();
                // Simulate some work
                Console.WriteLine($"Event elaborating {@event.Created}");
                Thread.Sleep(5000);
                Console.WriteLine($"Event comsumed {@event.Created}");
            }
        }
    }
}
```

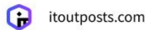
C# Channels are an implementation of the producer/consumer conceptual programming model.

https://learn.microsoft.com/en-us/dotnet/core/extensions/channels

Example project: 04 EventsOutOfProcessByChannel

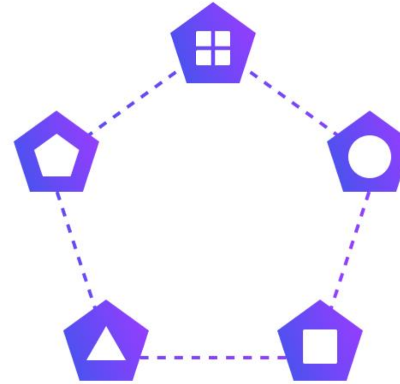
Monolith





In a monolithic application running on a single process, components invoke one another using language-level method or function calls.

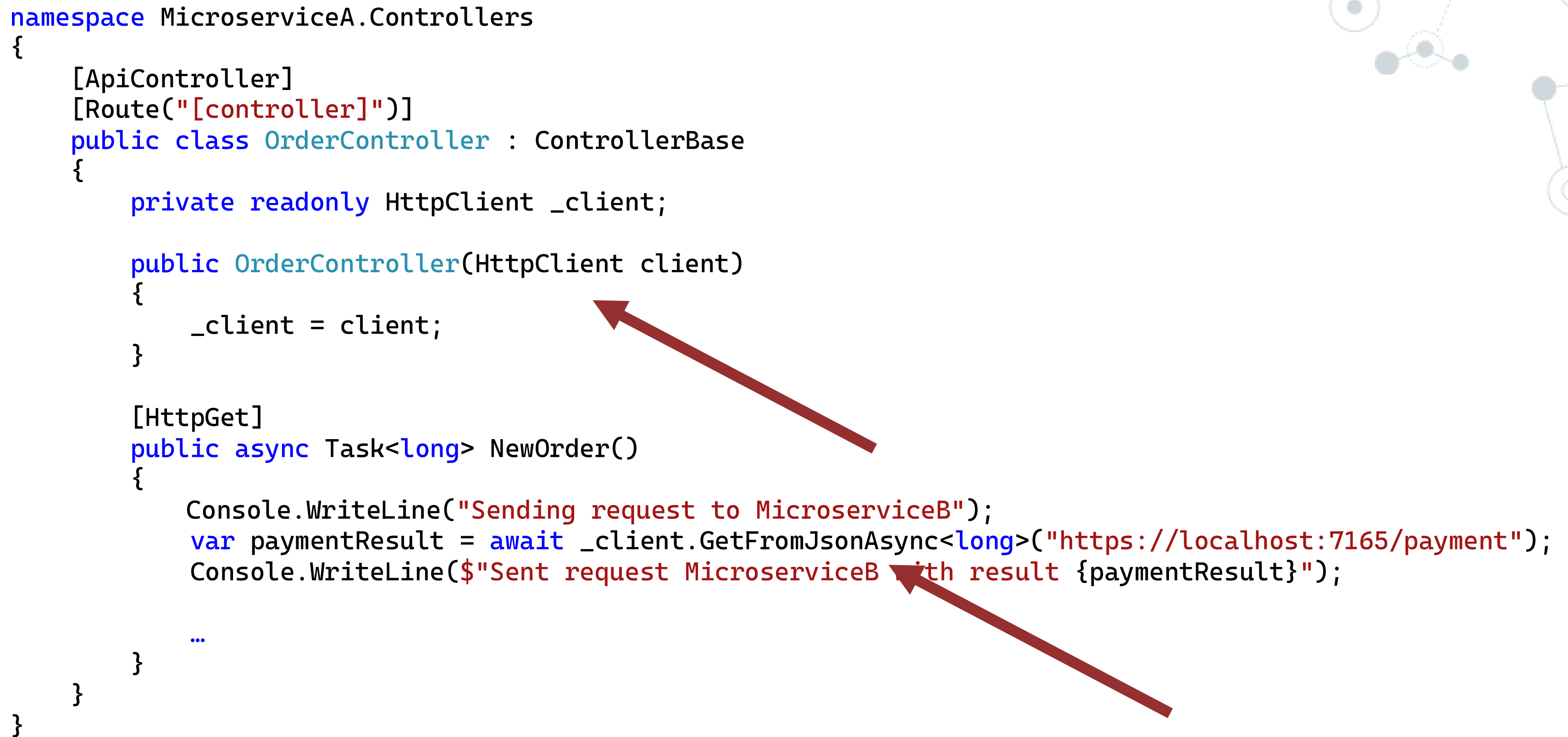
Microservices



A microservices-based application is a distributed system running on multiple processes or services, usually even across multiple servers or hosts

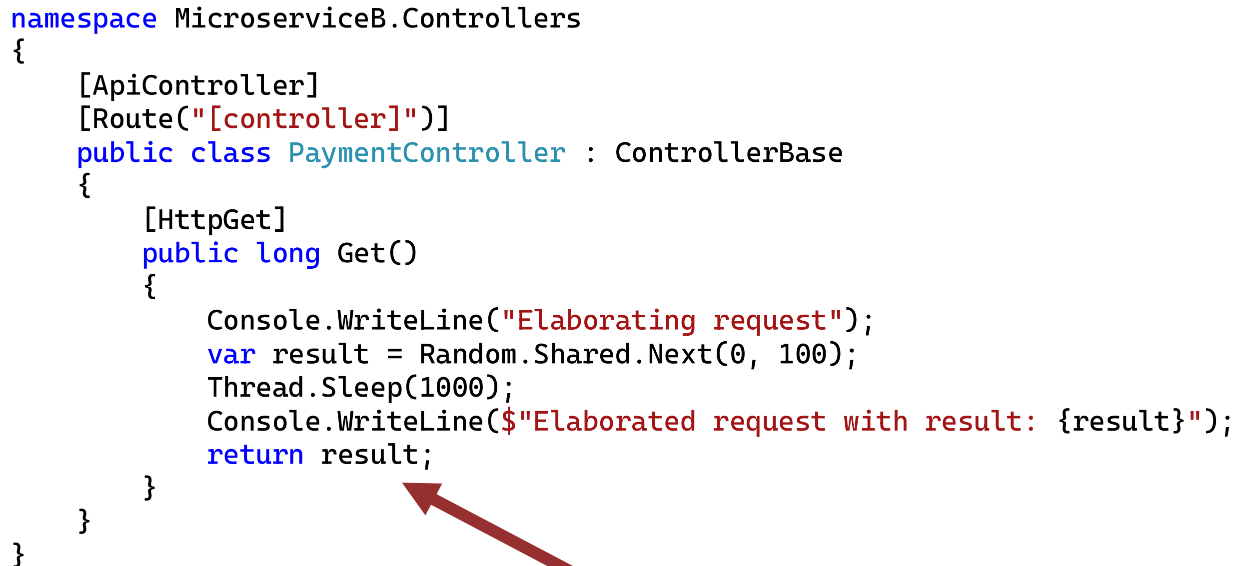
https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture

Out of-process / sync with microservice



Example project: 05 MicroserviceA/B

Out of-process / sync with microservice



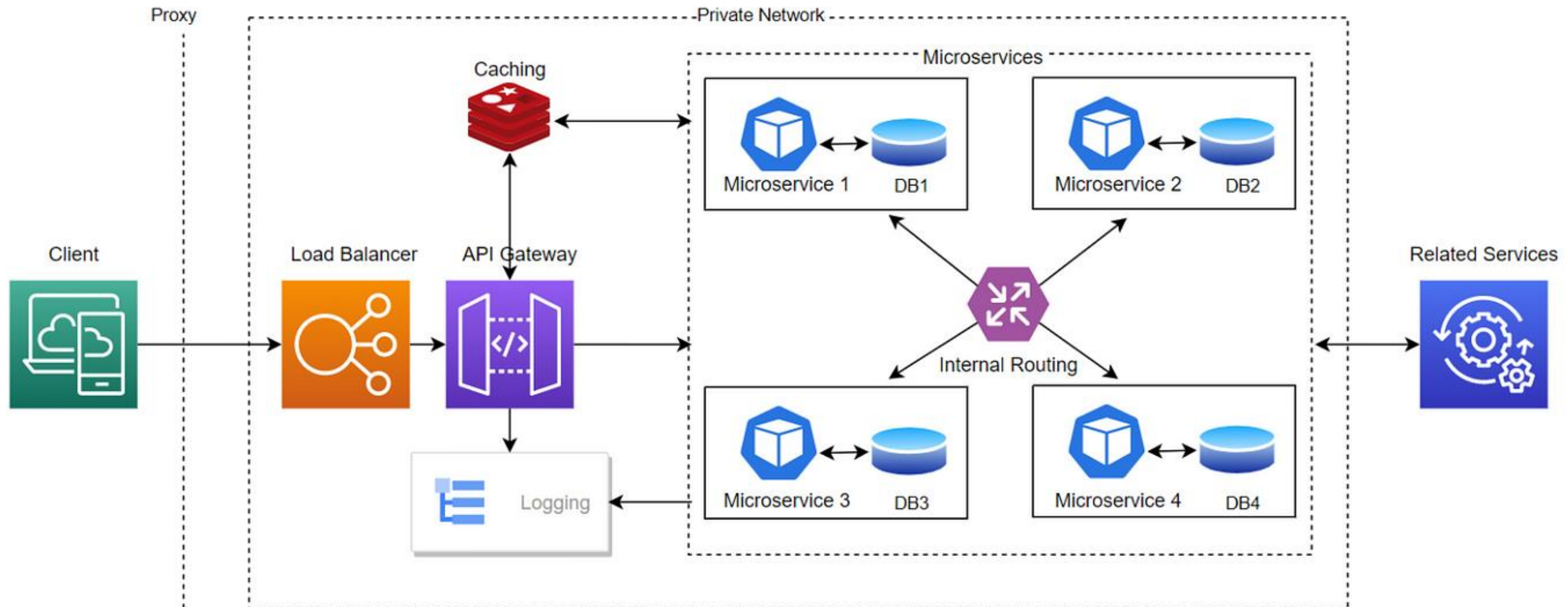
```
namespace MicroserviceB.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PaymentController : ControllerBase
    {
        [HttpGet]
        public long Get()
        {
            Console.WriteLine("Elaborating request");
            var result = Random.Shared.Next(0, 100);
            Thread.Sleep(1000);
            Console.WriteLine($"Elaborated request with result: {result}");
            return result;
        }
    }
}
```

Example project: 05 MicroserviceA/B

Out of-process / sync with microservice

Performance
Scalability
Resilience
Redundancy

?



https://medium.com/@beuttam/building-scalable-microservices-with-proxy-load-balancer-api-gateway-private-network-services-f25c73cc8e02

Out of-process / async with microservice - producer

```
namespace EventsOutOfProcessByDB.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class OrderController : ControllerBase
    {
        private readonly EventBusContext _eventBusContext;

        public OrderController(EventBusContext eventBusContext)
        {
            _eventBusContext = eventBusContext;
        }

        [HttpGet]
        public async Task NewOrder()
        {
            // Produce a new event and sent to channel
            var @event = new NewOrderEvent();
            @event.UserEmail = "diego@bonura.dev";

            var content = JsonSerializer.Serialize(@event, @event.GetType());
            var typeName = @event.GetType().FullName!;

            var message = new Message()
            {
                Type = typeName,
                Content = content
            };

            _eventBusContext.Add(message);
            await _eventBusContext.SaveChangesAsync();
        }
    }
}
```

Example project: 06 EventsOutOfProcessByDatabaseConsumer

Out of-process / async with microservice - consumer

```
protected override async Task ExecuteAsync(CancellationToken stoppingToken)
{
    while (true)
    {
        var messageToElaborate = _eventBusContext.Set<Message>().Where(m => m.ProcessedOn == null).OrderBy(m
=> m.OccurredOn).FirstOrDefault();
        if (messageToElaborate != null)
        {
            var type = AppDomain.CurrentDomain.GetAssemblies().Where(a => !a.IsDynamic).SelectMany(a =>
a.GetTypes()).FirstOrDefault(t => t.FullName == messageToElaborate.Type);
            var domainEvent = (INotification)JsonSerializer.Deserialize(messageToElaborate.Content, type);

            await _publisher.Publish(domainEvent);

            messageToElaborate.ProcessedOn = DateTime.Now;
            await _eventBusContext.SaveChangesAsync();
        }

        await Task.Delay(1000);
    }
}
```

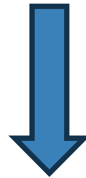
Example project: 06 EventsOutOfProcessByDatabaseConsumer

Out of-process / async with microservice consumer

Performance
Scalability
Resilience
Redundancy

?

Is it easy to add new consumers to increase performance?

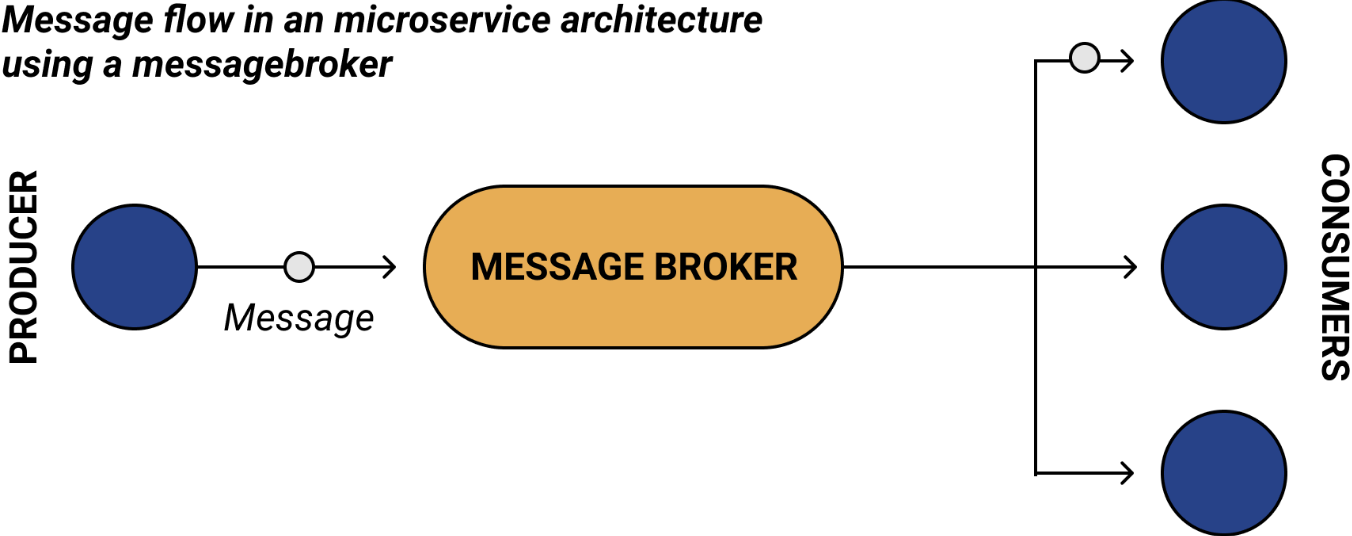


we need to introduce a row lock (on db side) or optimistic concurrency control (occ)

https://medium.com/@beuttam/building-scalable-microservices-with-proxy-load-balancer-api-gateway-private-network-services-f25c73cc8e02

an intermediary for messaging

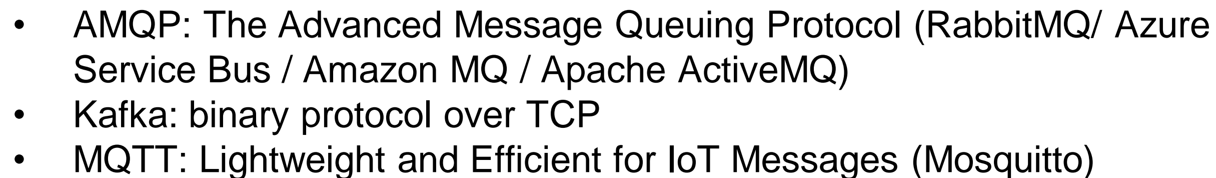
Message broker



Message broker

Message brokers

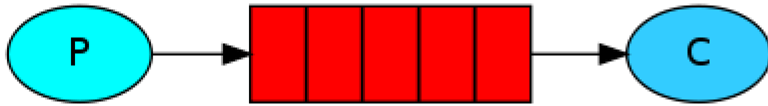
- can validate, store, route, and deliver messages to the appropriate destinations.
- act as intermediaries between other applications, allowing senders to issue messages without knowing where the recipients are located, whether or not they are active, or how many there are.
- simplifies the separation of processes and services within systems.

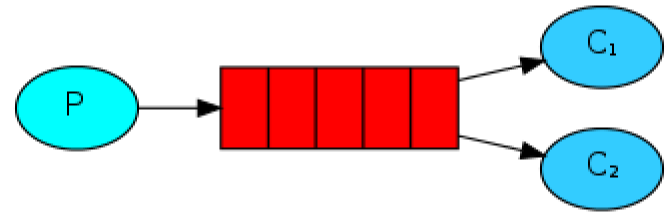


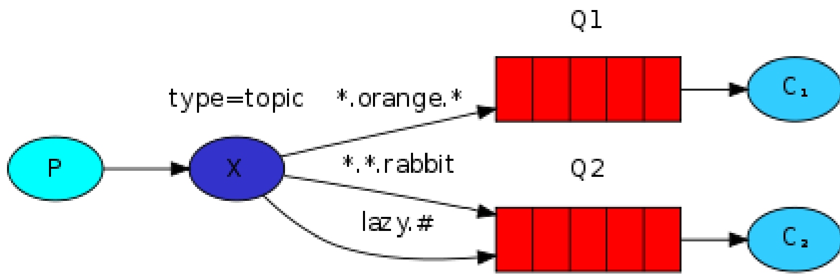
Protocols

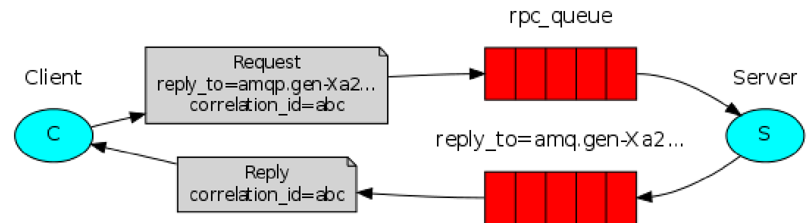
- AMQP: The Advanced Message Queuing Protocol (RabbitMQ/ Azure Service Bus / Amazon MQ / Apache ActiveMQ)
- Kafka: binary protocol over TCP
- MQTT: Lightweight and Efficient for IoT Messages (Mosquitto)

RabbitMQ

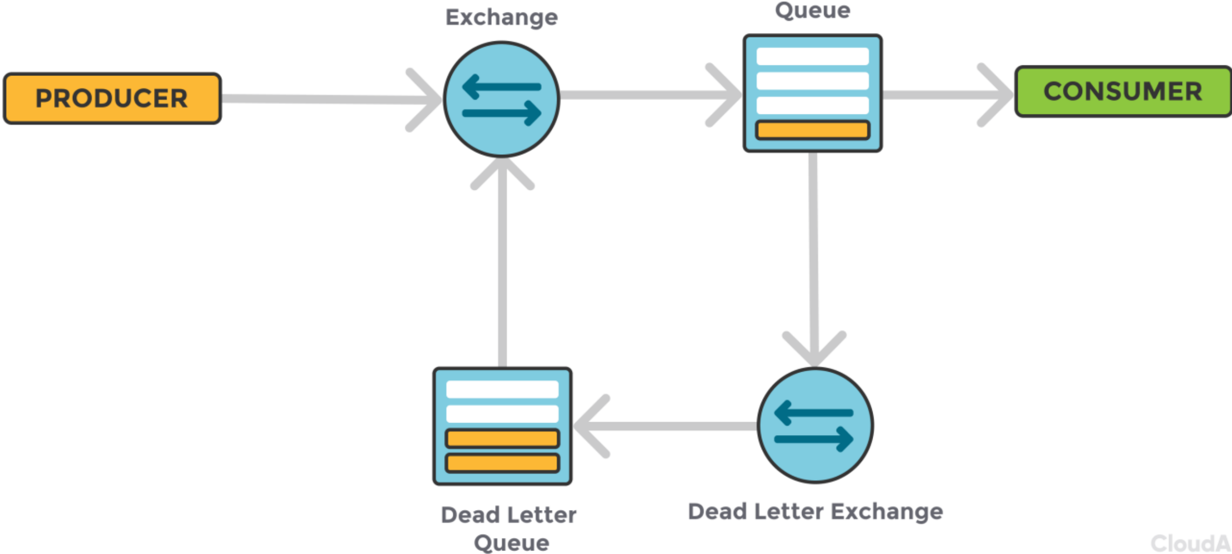




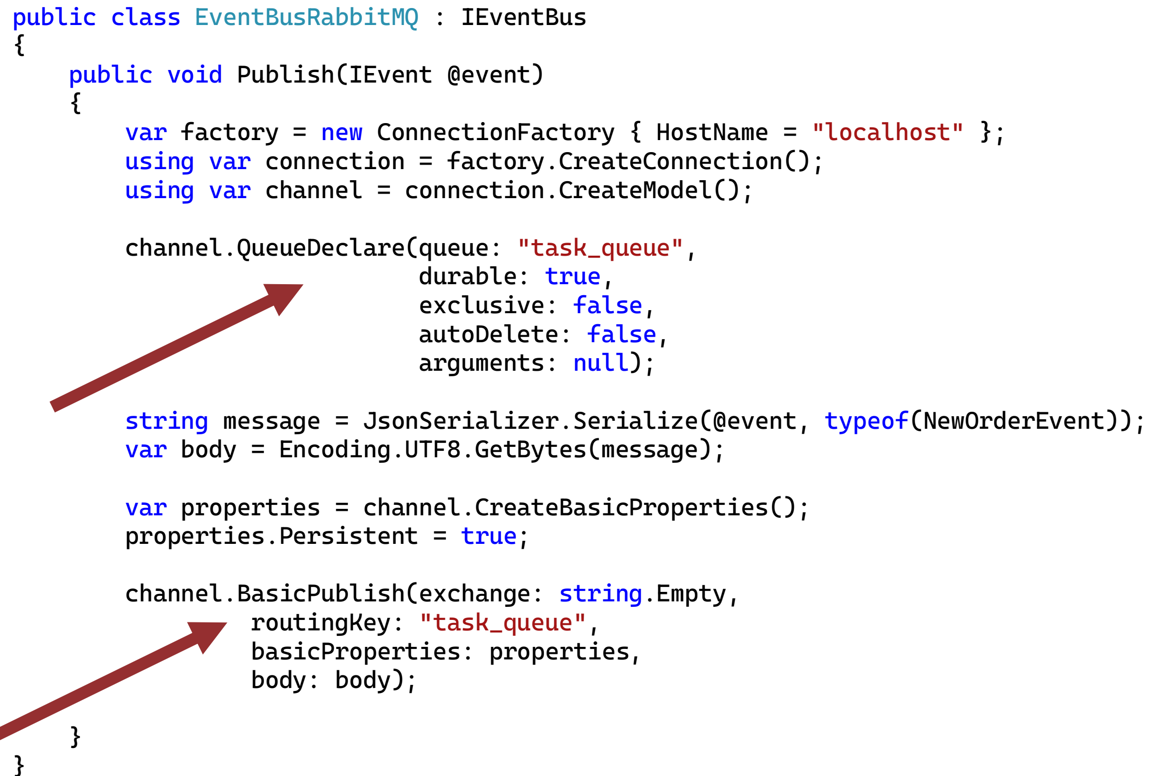




RabbitMQ



RabbitMQ - Producer



```
public class EventBusRabbitMQ : IEventBus
{
    public void Publish(IEvent @event)
    {
        var factory = new ConnectionFactory { HostName = "localhost" };
        using var connection = factory.CreateConnection();
        using var channel = connection.CreateModel();

        channel.QueueDeclare(queue: "task_queue",
                             durable: true,
                             exclusive: false,
                             autoDelete: false,
                             arguments: null);

        string message = JsonSerializer.Serialize(@event, typeof(NewOrderEvent));
        var body = Encoding.UTF8.GetBytes(message);

        var properties = channel.CreateBasicProperties();
        properties.Persistent = true;

        channel.BasicPublish(exchange: string.Empty,
                             routingKey: "task_queue",
                             basicProperties: properties,
                             body: body);

    }
}
```

RabbitMQ - Consumer

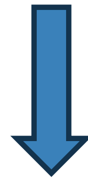
```
var factory = new ConnectionFactory { HostName = "localhost" };
using var connection = factory.CreateConnection();
using var channel = connection.CreateModel();

channel.QueueDeclare(queue: "task_queue",
                        durable: true,
                        exclusive: false,
                        autoDelete: false,
                        arguments: null);

channel.BasicQos(prefetchSize: 0, prefetchCount: 1, global: false);
var messageConsumer = new EventingBasicConsumer(channel);

messageConsumer.Received += async (model, ea) =>
{
    byte[] body = ea.Body.ToArray();
    var @event = (NewOrderEvent)JsonSerializer.Deserialize(body, typeof(NewOrderEvent));
    Console.WriteLine($"Received from {@event.UserEmail}");

    await Task.Delay(100);

    channel.BasicAck(deliveryTag: ea.DeliveryTag, multiple: false);
};

channel.BasicConsume(queue: "task_queue",
                     autoAck: false,
                     consumer: messageConsumer);

Console.ReadLine();
```

Distribute application with message broker

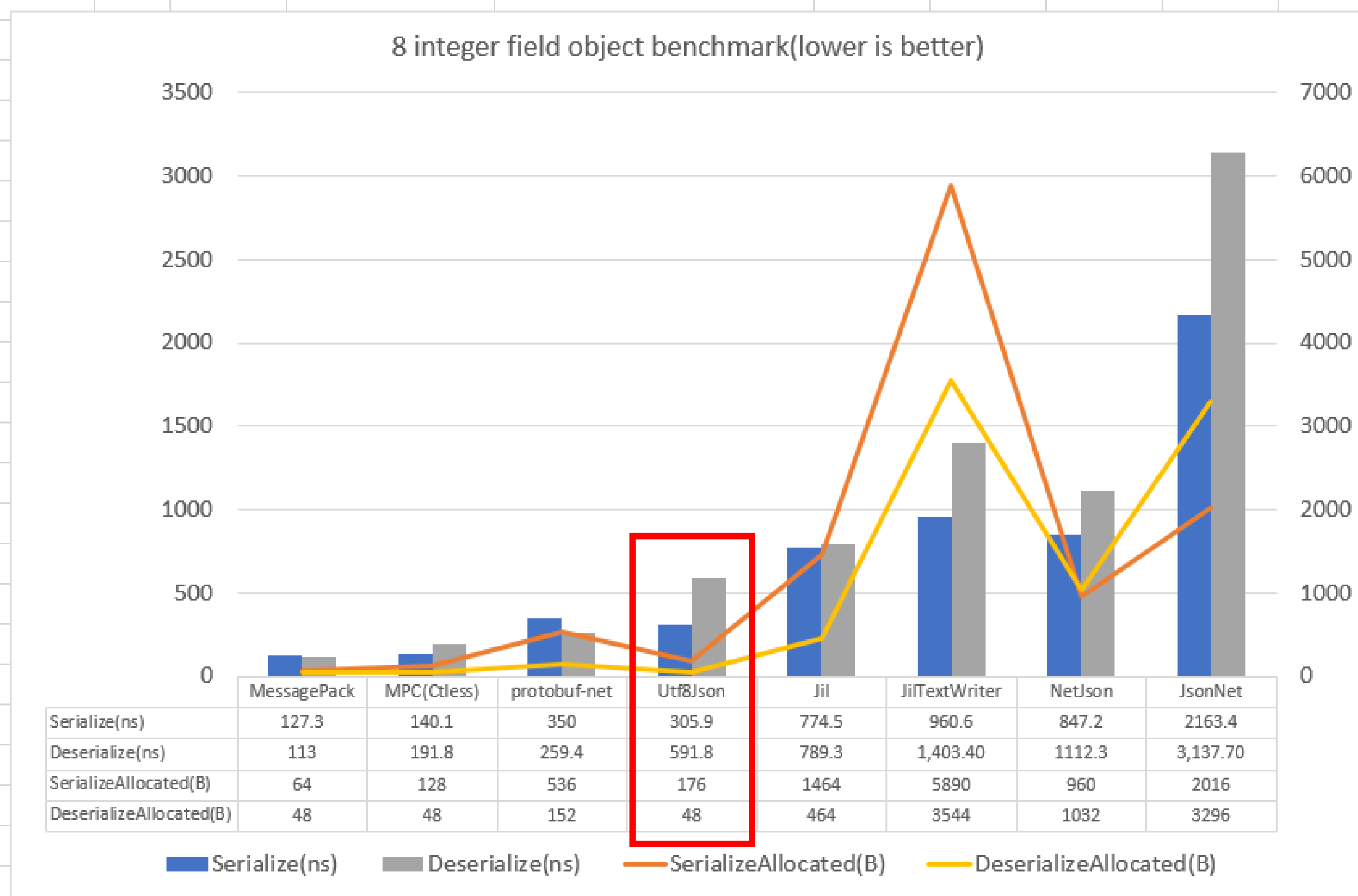
Performance
Scalability
Resilience
Redundancy

?

Is it easy to add new consumers to increase performance?



Serialization performance



	MessagePack	MPC(Ctless)	protobuf-net	Utf8Json	Jil	JilTextWriter	NetJson	JsonNet
Serialize(ns)	127.3	140.1	350	305.9	774.5	960.6	847.2	2163.4
Deserialize(ns)	113	191.8	259.4	591.8	789.3	1,403.40	1112.3	3,137.70
SerializeAllocated(B)	64	128	536	176	1464	5890	960	2016
DeserializeAllocated(B)	48	48	152	48	464	3544	1032	3296

https://github.com/neuecc/Utf8Json

Serialization performance

Json

Overview				Messages			Message rates		
Name	**Type**	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
task_queue	classic	D	running	1,835	0	1,835	36/s		

+/-

▸ **Add a new queue**

Protobuf

Overview				Messages			Message rates		
Name	**Type**	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
task_queue	classic	D	running	237	0	237	52/s		

+/-

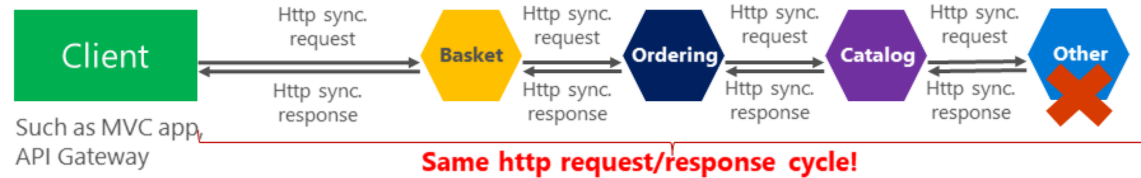
▸ **Add a new queue**

Communication types

Synchronous vs. async communication across microservices

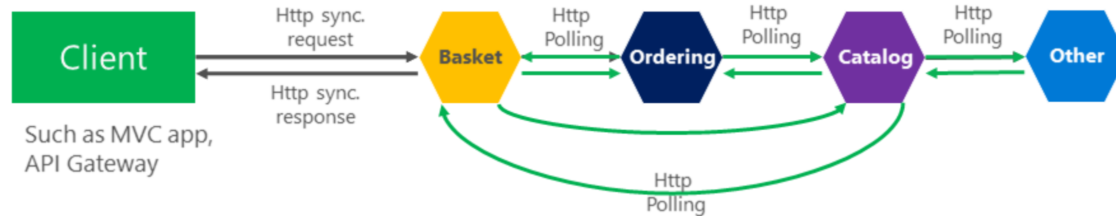
Anti-pattern

Synchronous
all request/response cycle



Asynchronous
Comm. across internal microservices
(EventBus: like **AMQP**)

"Asynchronous"
Comm. across internal microservices
(Polling: **Http**)



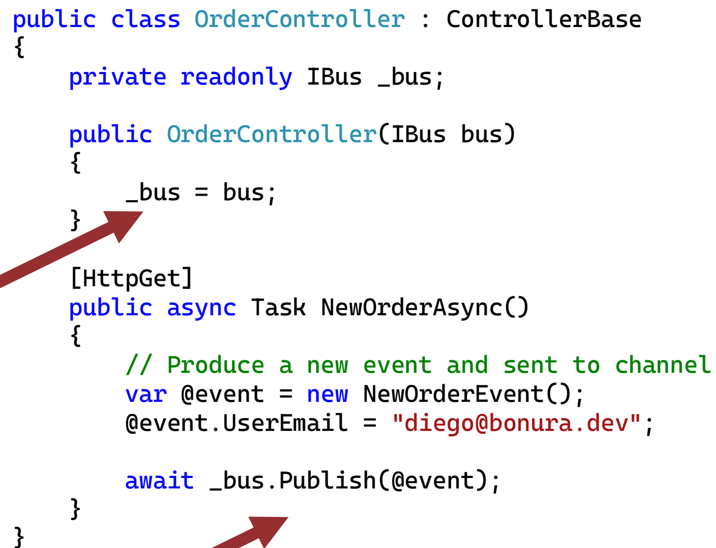
Distributed
application with a
framework

Easily build reliable distributed applications

MassTransit provides a developer-focused, modern platform for creating distributed applications without complexity.

- First class testing support
- Write once, then deploy using RabbitMQ, Azure Service Bus, and Amazon SQS
- Observability via Open Telemetry (OTEL)
- Fully-supported, widely-adopted, a complete end-to-end solution

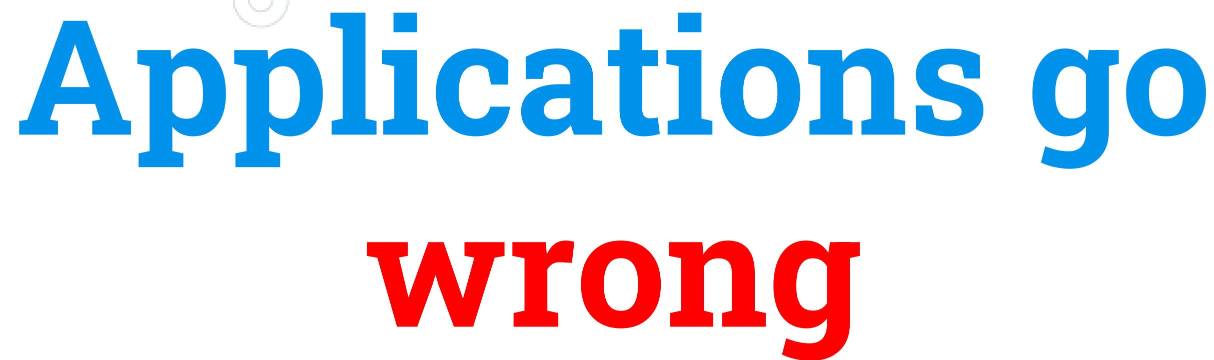
Masstransit - Producer



```
public class OrderController : ControllerBase
{
    private readonly IBus _bus;

    public OrderController(IBus bus)
    {
        _bus = bus;
    }

    [HttpGet]
    public async Task NewOrderAsync()
    {
        // Produce a new event and sent to channel
        var @event = new NewOrderEvent();
        @event.UserEmail = "diego@bonura.dev";

        await _bus.Publish(@event);
    }
}
```

Masstransit - Consumer

```
namespace DistributedAppWithMassTransitConsumer
{
    public class MessageConsumer : IConsumer<NewOrderEvent>
    {
        readonly ILogger<MessageConsumer> _logger;

        public MessageConsumer(ILogger<MessageConsumer> logger)
        {
            _logger = logger;
        }

        public Task Consume(ConsumeContext<NewOrderEvent> context)
        {
            _logger.LogInformation("Received ordine from: {email}", context.Message.UserEmail);

            return Task.CompletedTask;
        }
    }
}
```



The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair.

—Douglas Adams, *Mostly Harmless* (1992)

Applications go wrong

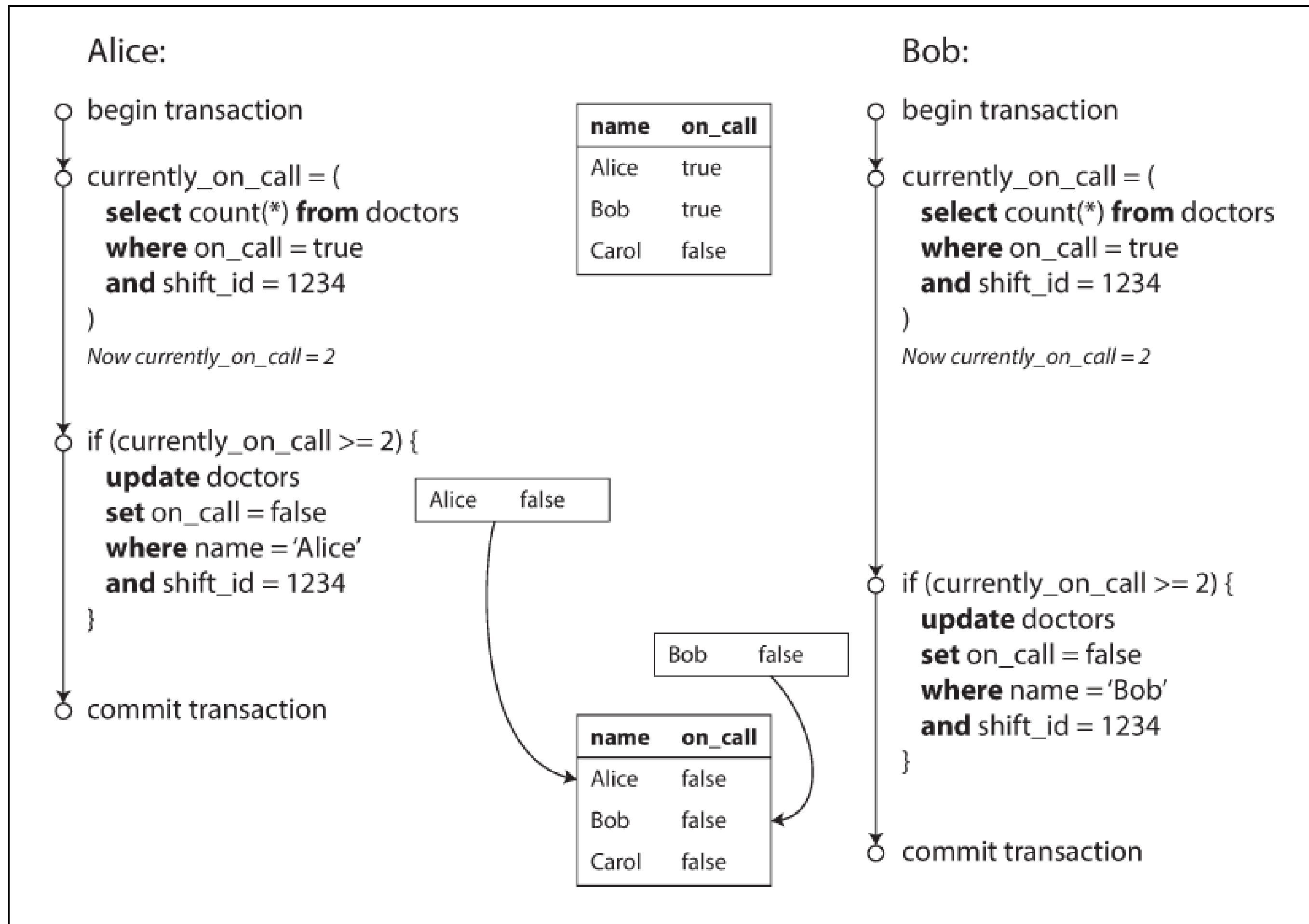
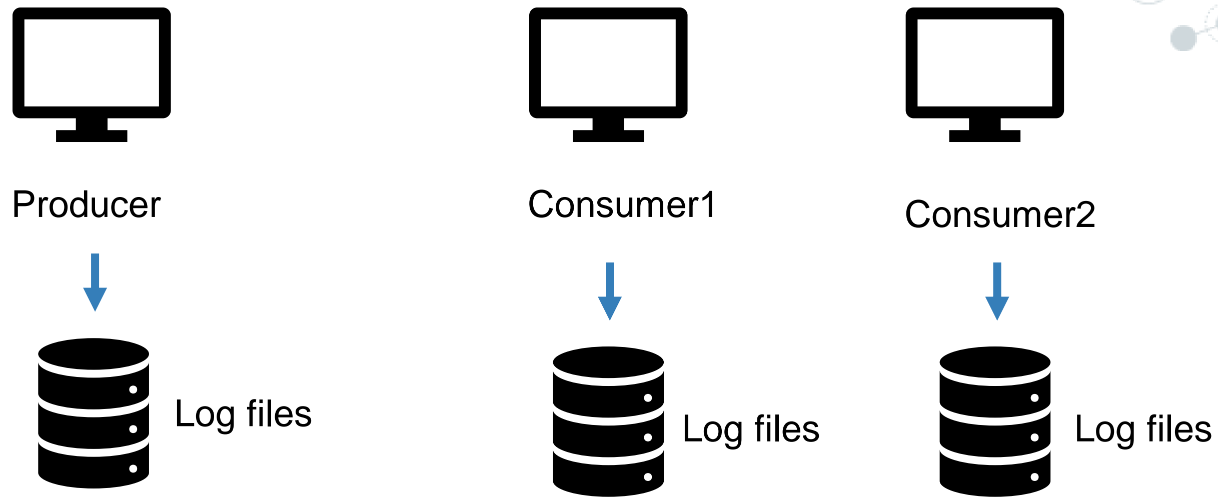


Figure 7-8. Example of write skew causing an application bug.

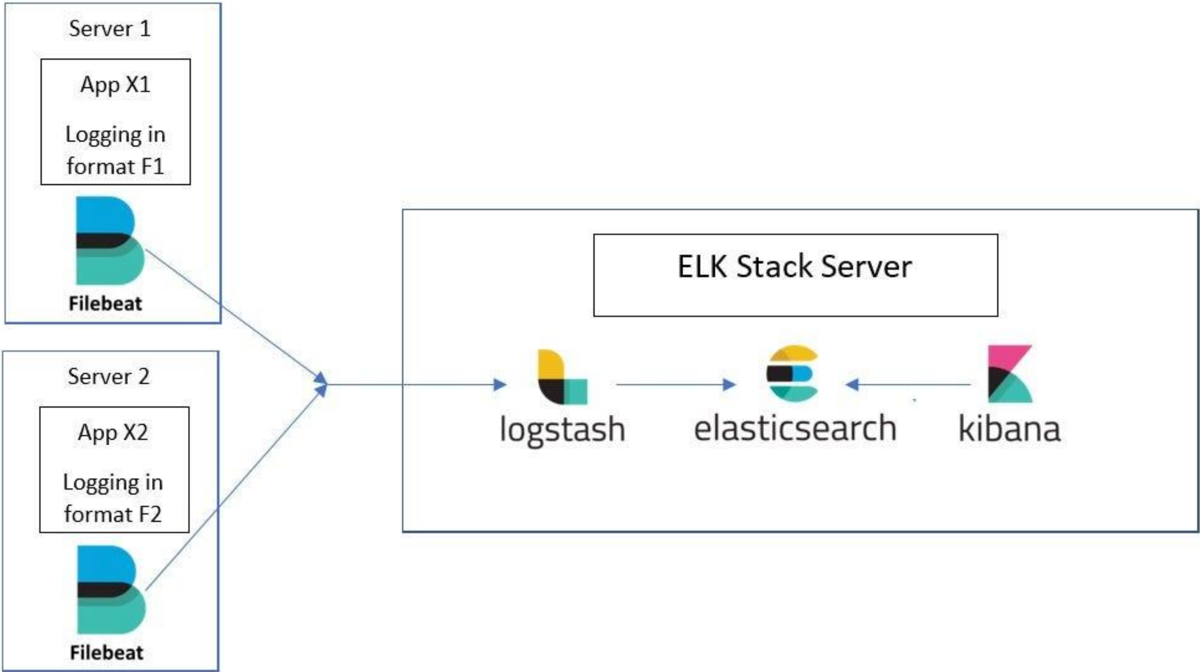
Page 246 of Design Data-Intensive Applications

Logging on distributed application



How to get information when things go wrong?

Call logs in one place



Call logs in one place

kibana

Discover
Visualize
Dashboard
Timelion
Prometheus
Alerting
Dev Tools
Management
Collapse

95,471,548 hits — New — Save — Open — Share — Inspect — 5 seconds — Last 60 days

Search... (e.g. status:200 AND extension:PHP) — Options — Refresh

Add a filter +

flb-*

Selected fields
? _source

Available fields
Popular
t logs.message
@fb_timestamp
t _id
t _index
_score
t _type
t kubernetes.anno...
? kubernetes.anno...
? kubernetes.anno...
t kubernetes.anno...

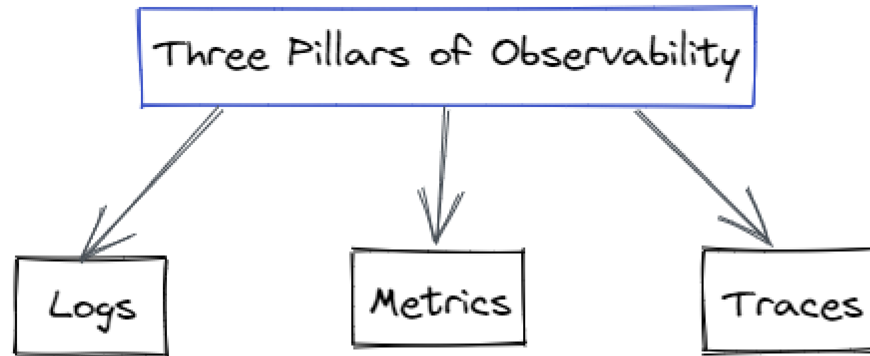
January 17th 2021, 11:00:56.983 - March 18th 2021, 11:00:56.983 — Auto

Count: 2,000,000 / 1,500,000 / 1,000,000 / 500,000 / 0
2021-01-24 2021-01-31 2021-02-07 2021-02-14 2021-02-21 2021-02-28 2021-03-07 2021-03-14
time per day

Time	_source
March 18th 2021, 11:00:39.239	@fb_timestamp: March 18th 2021, 11:00:39.239 log: stream: stdout time: March 18th 2021, 11:00:39.239 kubernetes.pod_name: api-78b695c46b-j8v69 kubernetes.namespace_name: default kubernetes.pod_id: f162dd48-e68d-461d-bcc6-b3581fc6a97a kubernetes.labels.app_kubernetes_io/component: backend kubernetes.labels.app_kubernetes_io/managed-by: hybris-operator
March 18th 2021, 11:00:39.238	@fb_timestamp: March 18th 2021, 11:00:39.238 log: stream: stdout time: March 18th 2021, 11:00:39.238 kubernetes.pod_name: api-78b695c46b-j8v69 kubernetes.namespace_name: default kubernetes.pod_id: f162dd48-e68d-461d-bcc6-b3581fc6a97a

On distributed application logs monitoring could be difficult

Main concepts of observability



Logs in the technology and development field give a written record of happenings within a system, similar to the captain's log on a ship.

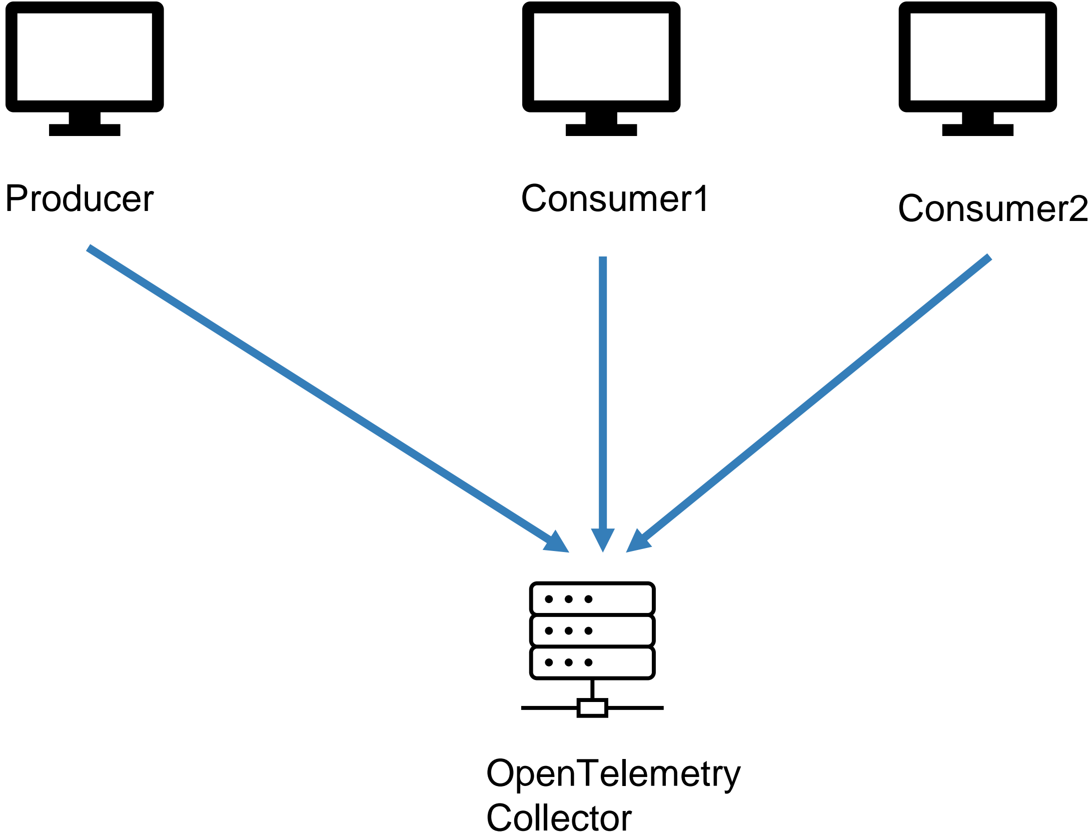
Metrics are a set of values that are tracked over time.

A **trace** is a means to track a user request from the user interface all the way through the system and back to the user when they receive confirmation that their request has been completed. As part of the trace, every operation executed in response to the request is recorded.

Observability standard

OpenTelemetry is an open-source CNCF (Cloud Native Computing Foundation) project formed from the merger of the OpenCensus and OpenTracing projects. It provides a collection of tools, APIs, and SDKs for capturing metrics, distributed traces and logs from applications.

OpenTelemetry on distributed application



Example

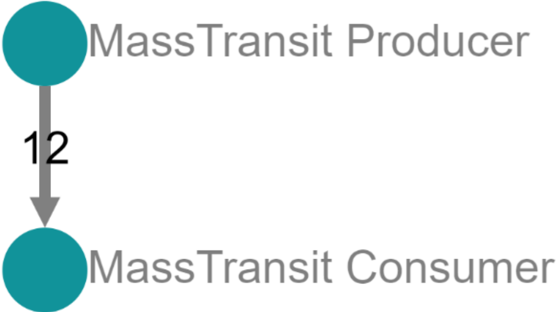
Trace:

MassTransit Producer: Order 182a1dc 10.58ms

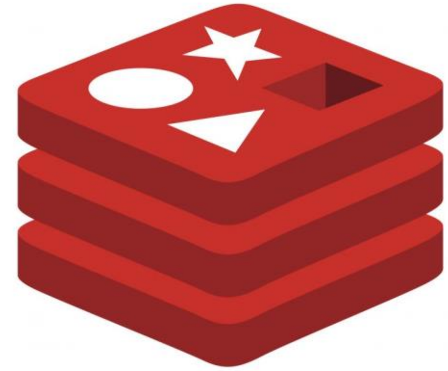
4 Spans | MassTransit Consumer (2) | MassTransit Producer (2)

Today 4:59:17 pm
a minute ago

Metric:



Distributed locks are a very useful primitive in many environments where different processes must operate with shared resources in a mutually exclusive way.

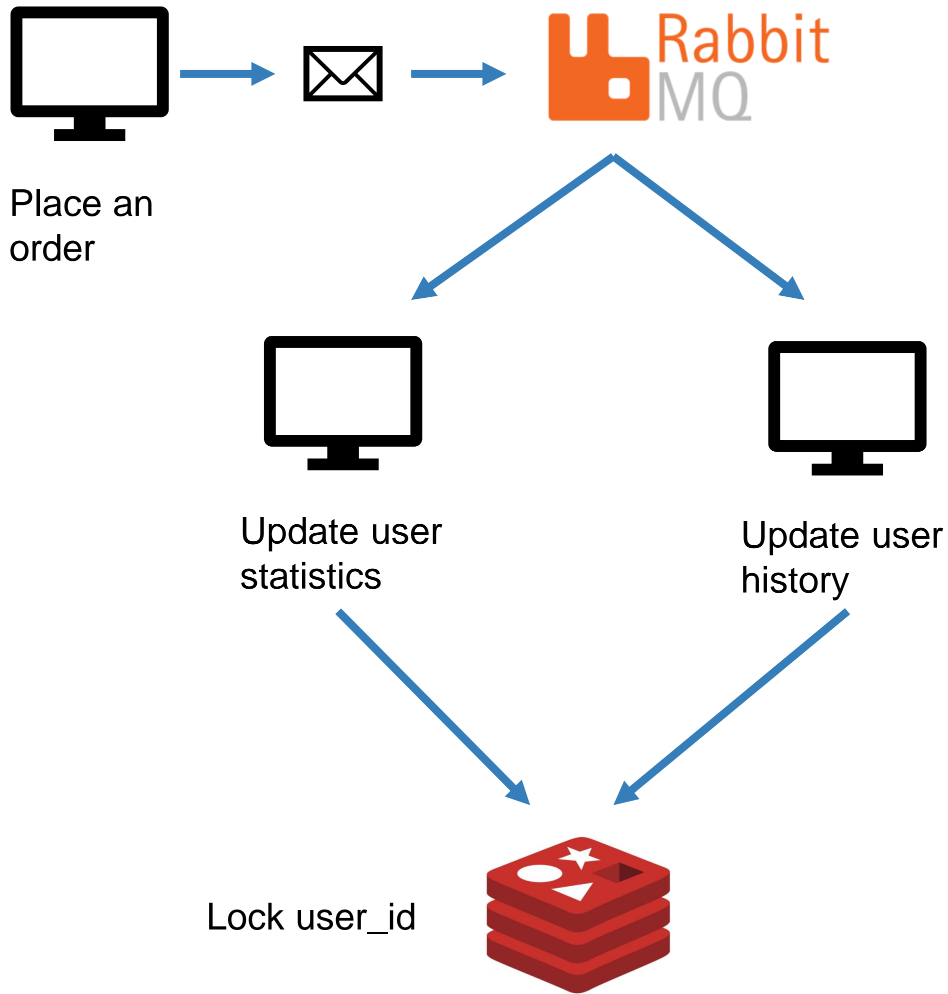


Redis

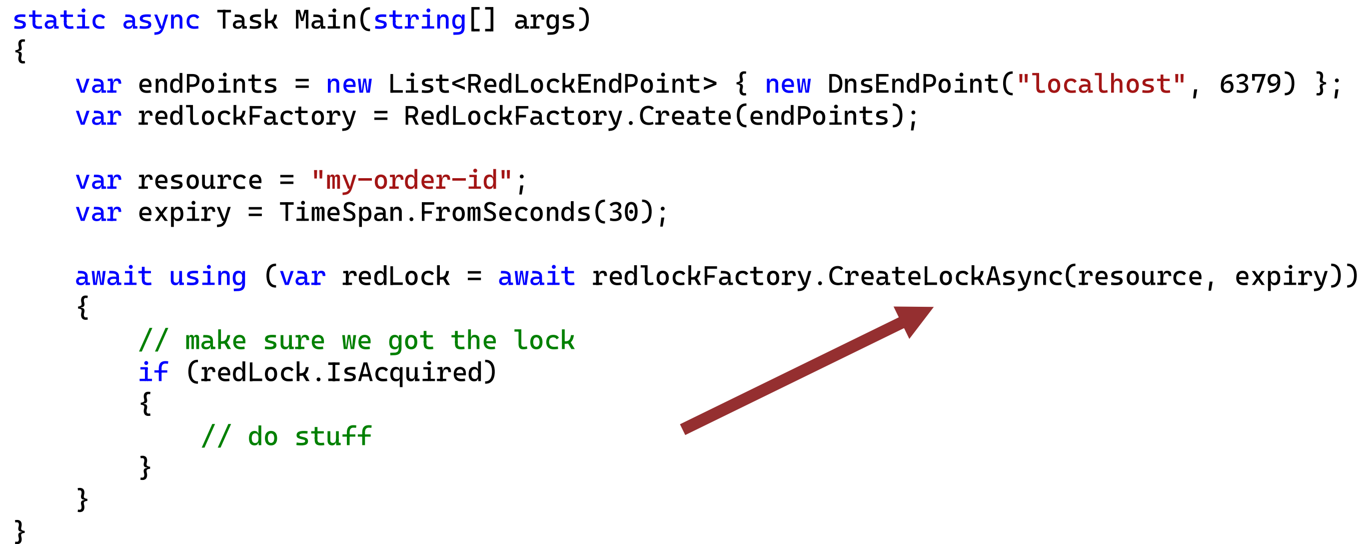
The open source, in-memory data store used by millions of developers as a database, cache, streaming engine, and message broker.

Created by: Salvatore Sanfilippo

https://redis.io/


Place an order
Rabbit MQ
Update user statistics
Update user history
Lock user_id

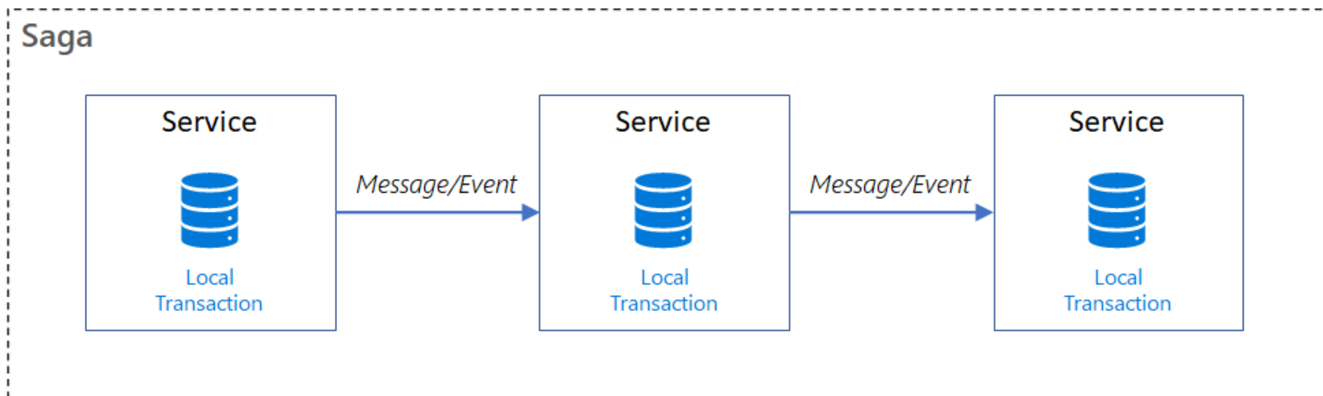
Redis lock



```
static async Task Main(string[] args)
{
    var endPoints = new List<RedLockEndPoint> { new DnsEndPoint("localhost", 6379) };
    var redlockFactory = RedLockFactory.Create(endPoints);

    var resource = "my-order-id";
    var expiry = TimeSpan.FromSeconds(30);

    await using (var redLock = await redlockFactory.CreateLockAsync(resource, expiry))
    {
        // make sure we got the lock
        if (redLock.IsAcquired)
        {
            // do stuff
        }
    }
}
```

Saga

When you have to orchestrate events!



Saga: consistency models

Immediate consistency: once a write operation (e.g., updating a piece of data) is completed, any subsequent read operation (e.g., retrieving that data) will reflect the updated value.

- expensive in terms of performance
- not ideal in all distributed systems

ACID (atomicity, consistency, isolation, durability).

Eventual consistency: may be a period of time during which different nodes or replicas in the system have different versions of the data.

- commonly used in systems like NoSQL databases

BASE (basically-available, soft-state, eventual consistency)

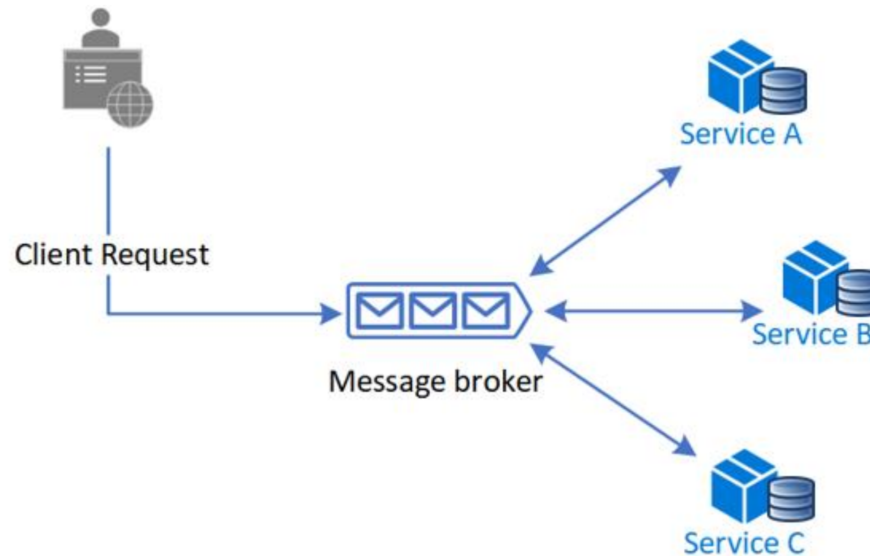
Saga: trade off

https://priyalwalpita.medium.com/steering-clear-of-distributed-monolith-traps-in-your-journey-to-effective-microservices-86671be0b604

https://www.youtube.com/watch?v=p2GIRToY5HI

Saga approaches: choreography and orchestration

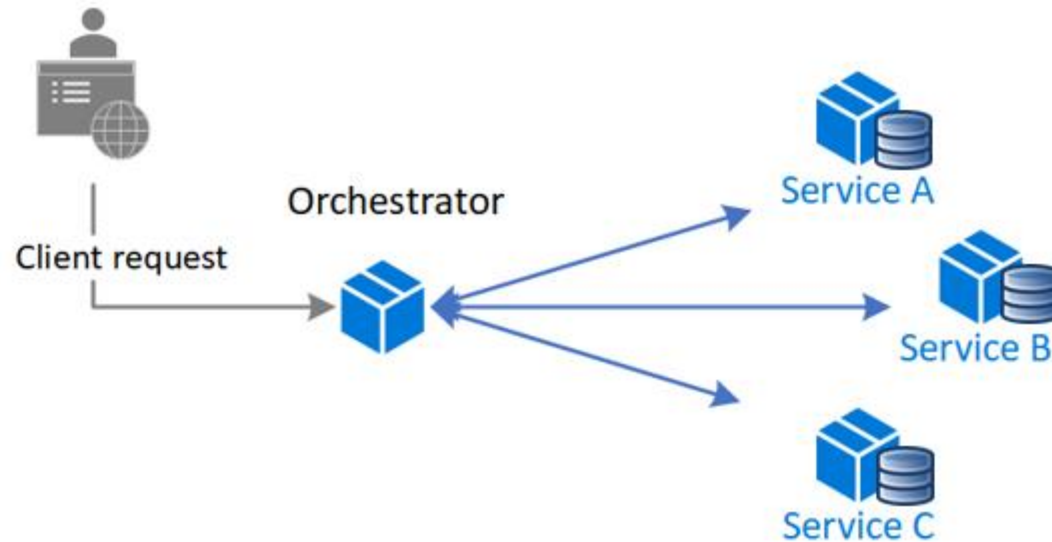
Choreography: without a centralized point of control



https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga

Saga approaches: choreography and orchestration

Orchestration: centralized controller tells participants what to execute



https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga

Saga choreography

```
public OrderStateMachine()
{
    InstanceState(x => x.CurrentState);

    Event(() => NewOrderEvent, x => x.CorrelateById(context => context.Message.OrderId));
    Event(() => OrderProcessed, x => x.CorrelateById(context => context.Message.OrderId));
    Event(() => OrderCancelled, x => x.CorrelateById(context => context.Message.OrderId));

    Initially(
        When(NewOrderEvent)
            .Then(context =>
            {
                context.Saga.ProcessingId = Guid.NewGuid();
            })
            .Publish(context => new ProcessOrder(context.Saga.CorrelationId))
            .TransitionTo(Pending)
            .Then(context => Console.Out.WriteLineAsync($"From New to Pending: {context.Saga.CorrelationId}"))
    );

    During(Pending,
        When(OrderProcessed)
            .TransitionTo(Accepted)
            .Then(context => Console.Out.WriteLineAsync($"From Pending to Accepted: {context.Saga.CorrelationId}"))
            .Finalize(),
        When(OrderCancelled)
            .TransitionTo(Cancelled)
            .Then(context => Console.Out.WriteLineAsync($"From Pending to Faulted: {context.Saga.CorrelationId} for reason:
{context.Message.Reason}"))
            .Finalize()
        );

    SetCompletedWhenFinalized();
}
```

Saga choreography

MassTransit elaborates saga and creates few queue and exchanges on RabbitMq

Exchanges

All exchanges (13)

Pagination

Page 1 of 1 - Filter: Regex ?

Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
/	(AMQP default)	direct	D			
/	Message	fanout	D			
/	OrderState	fanout	D			
/	SagaWithMasstransitShared:NewOrderEvent	fanout	D	0.00/s	0.00/s	
/	SagaWithMasstransitShared:OrderCancelled	fanout	D	0.00/s	0.00/s	
/	SagaWithMasstransitShared:OrderProcessed	fanout	D	0.00/s	0.00/s	
/	SagaWithMasstransitShared:ProcessOrder	fanout	D	0.00/s	0.00/s	
/	amq.direct	direct	D			
/	amq.fanout	fanout	D			
/	amq.headers	headers	D			
/	amq.match	headers	D			
/	amq.rabbitmq.trace	topic	D I			
/	amq.topic	topic	D			

Instead of calling methods, actors send messages to each other!

https://doc.akka.io/docs/akka/current/typed/guide/actors-intro.html

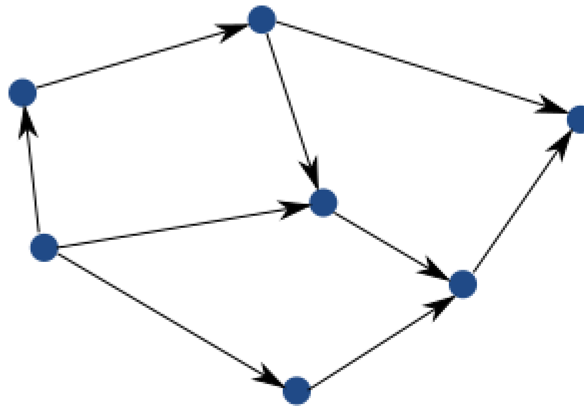
https://learn.microsoft.com/en-us/dotnet/orleans/overview

The Actor Model: A Paradigm for Concurrent and Distributed Computing

The actor model is a programming model in which each actor is a lightweight, concurrent, immutable object that encapsulates a piece of state and corresponding behavior. Actors communicate exclusively with each other using asynchronous messages.

Actor model

When we have a Producer and Consumer we usually send message to a queue

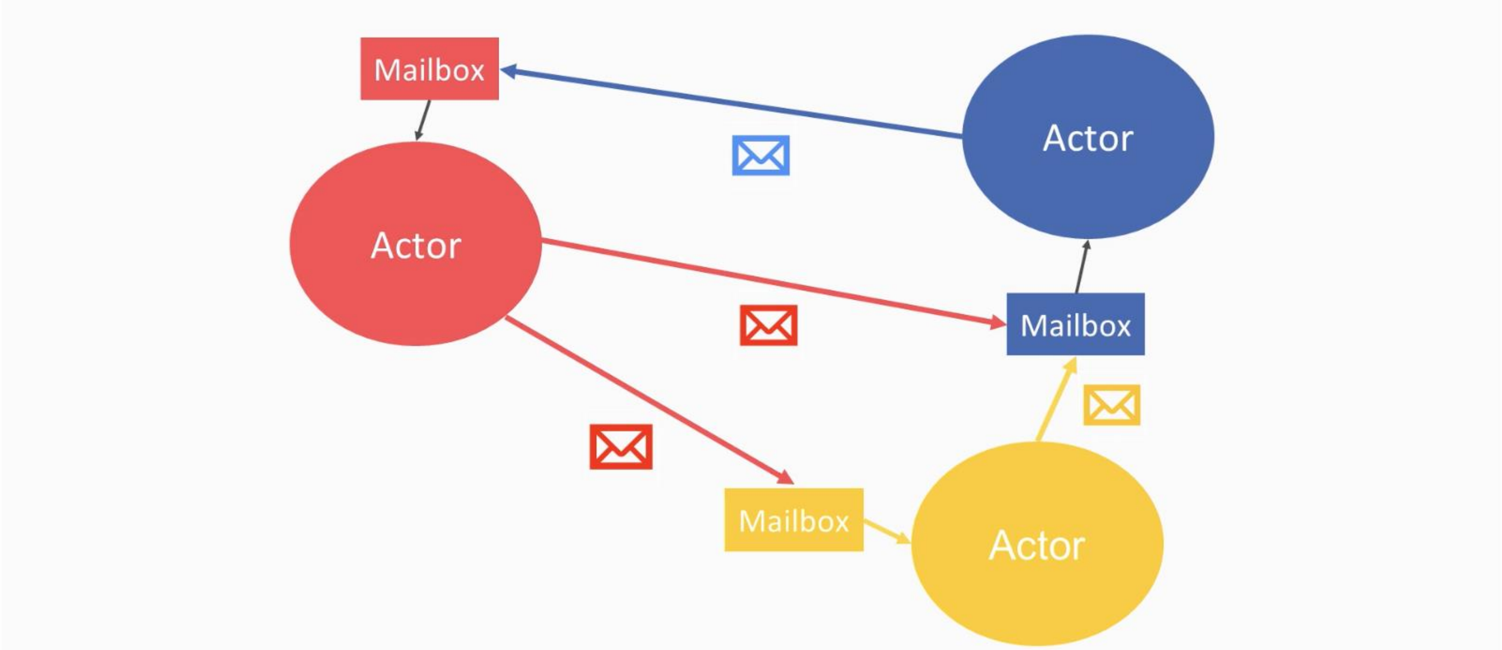


Actors interacting with each other
by sending messages to each other

On actor model, we can implement Producer and Consumer as actor.

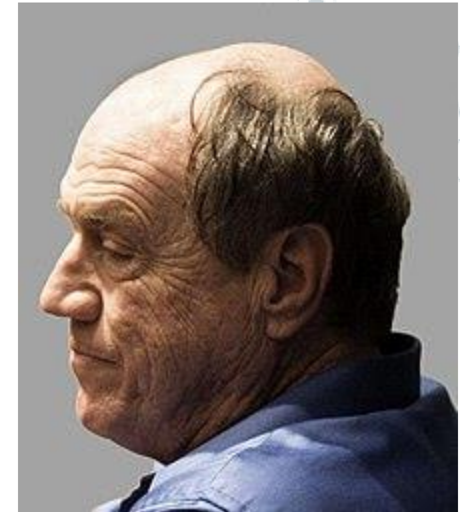
In Producer, we just get the actor reference of Consumer actor to send messages to Consumer’s mailbox.

Actor model



Actor model: History 1973

The Actor Model is a mathematical theory of computation that treats "Actors" as the universal conceptual primitives of concurrent digital computation.


Carl Hewitt

The actor model was inspired by physics

Actors is based on "behavior" as opposed to the "class" concept of object-oriented programming.

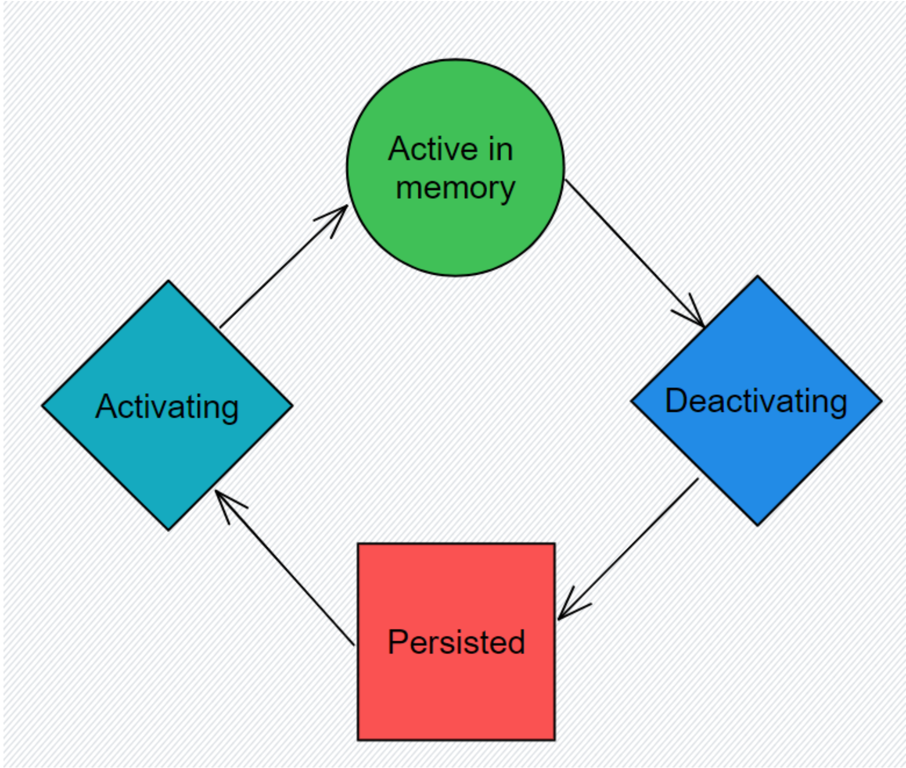
https://en.wikipedia.org/wiki/Actor_model

Actor model

Main principles:

1. **Isolation**: Actors are independent, with their own state and behavior.
2. **Single thread**: Actors process requests one at time
3. **Messaging**: Actors interact by exchanging asynchronous messages.
4. **Location Transparency**: Actors' locations are abstracted, enabling distribution.

Actor model: life cycle



Actor model: implementations

Java / c#

https://akka.io/

https://getakka.net/

Orleans

c#

https://learn.microsoft.com/en-us/dotnet/orleans/overview

Actor model implementations on Orleans Microsoft research (2010)

https://www.microsoft.com/en-us/research/project/orleans-virtual-actors/

Orleans invented the Virtual Actor abstraction

Actors are purely logical entities that always exist, virtually. An actor cannot be explicitly created nor destroyed, and its virtual existence is unaffected by the failure of a server that executes it. Since actors always exist, they are always addressable.

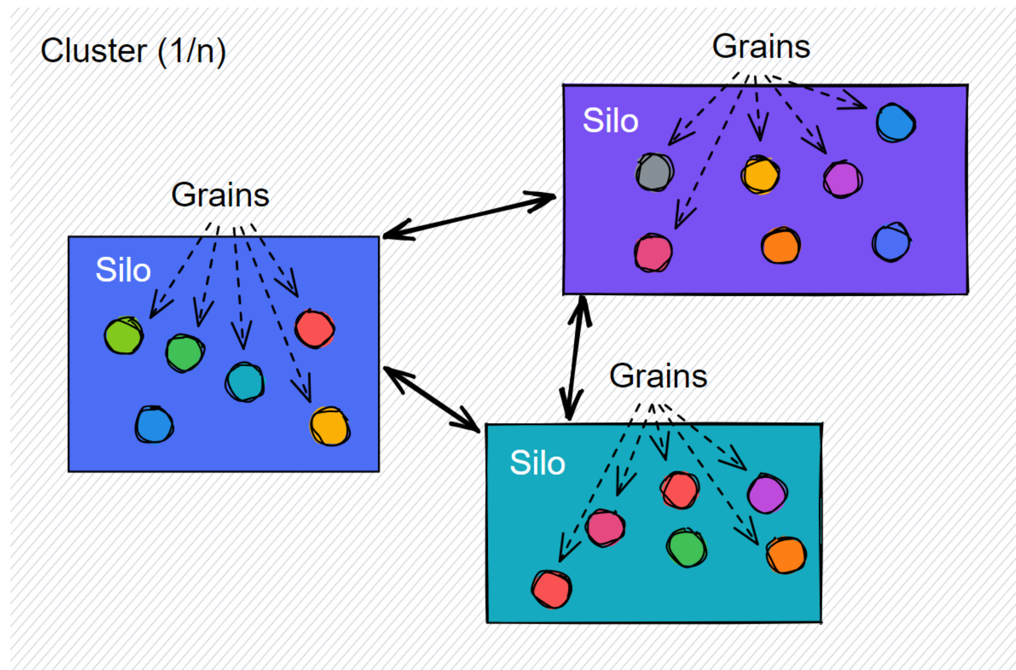
Actor model implementations on Orleans - Grain

1. **Grain**: grains are implementation of a virtual actor.
2. **Interfaces**: grains define interfaces.
3. **Grain:** has always an identity (string, number, guid)
4. **Persistence**: grains could volatile or persisted
5. **Lifecycle**: grains could be terminated to free computer resources

https://learn.microsoft.com/en-us/dotnet/orleans/overview#what-are-grains

Actor model implementations on Orleans - Silo

A silo hosts one or more grains



You can have any number of clusters, each cluster has one or more silos, and each silo has one or more grains

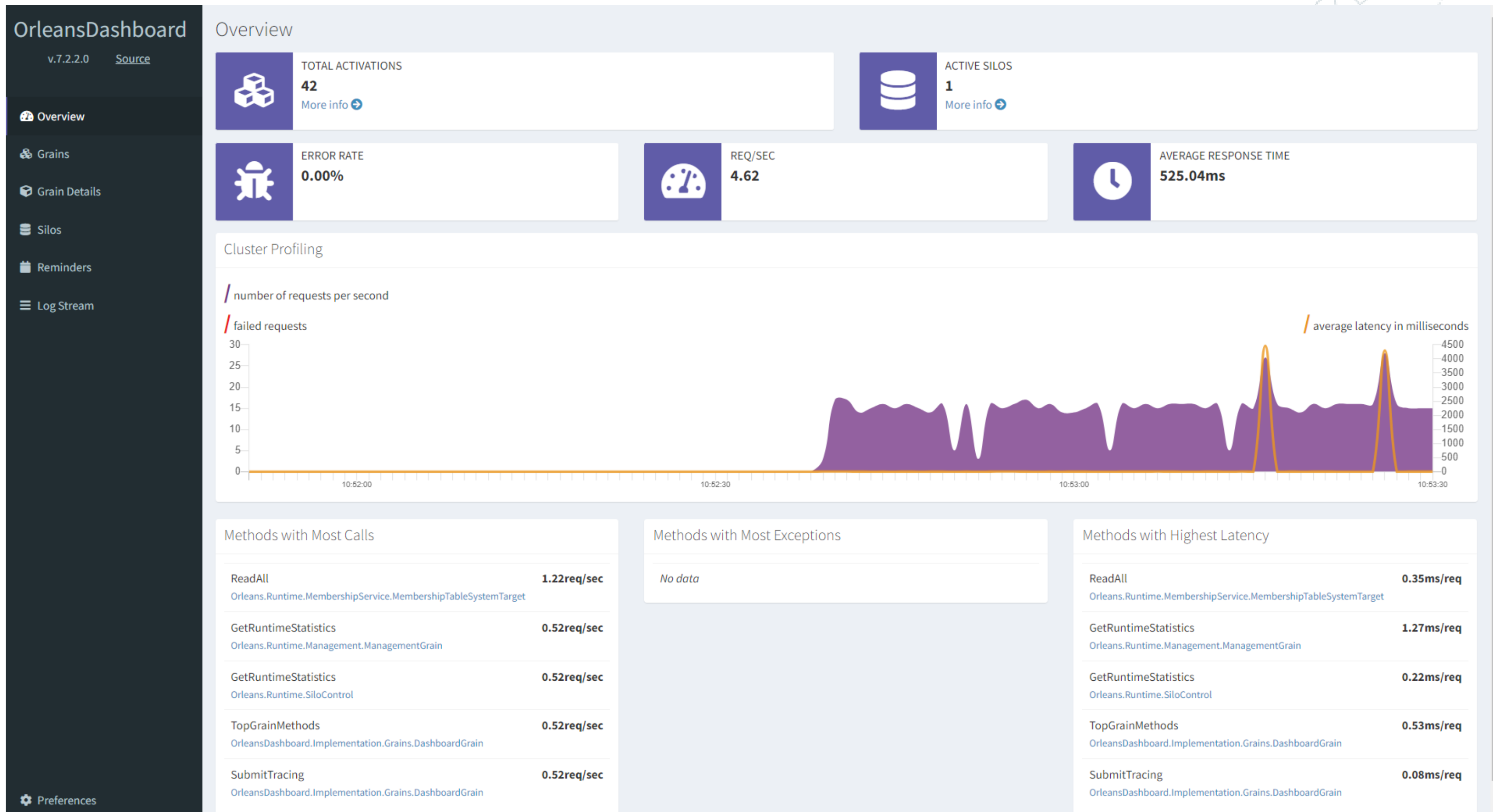
https://learn.microsoft.com/en-us/dotnet/orleans/overview#what-are-silos

Actor model implementations on Orleans - Silo

1. Host grains
2. Responsible to activate and deactivate grains
3. Typically: 1 silo per container/node
4. Could be embedded into main application or in separate container/node
5. Clustering silos is easy

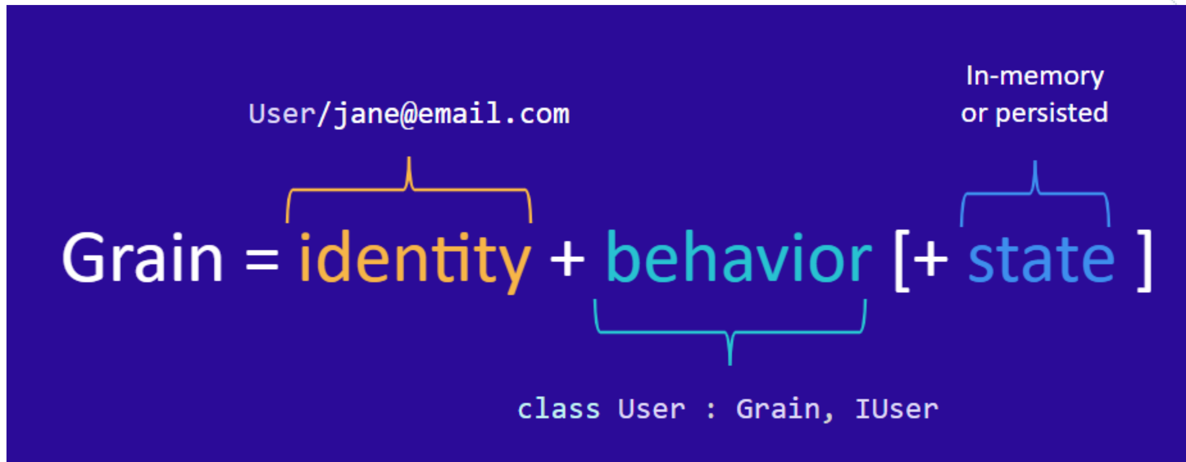
Actor model implementations on Orleans - Dashboard

https://github.com/OrleansContrib/OrleansDashboard



http://localhost:8080

Actor model implementations on Orleans – Calling actors



You can start an actor using grainFactory:

```
_grainFactory.GetGrain<IGrainA>("my-id");
```

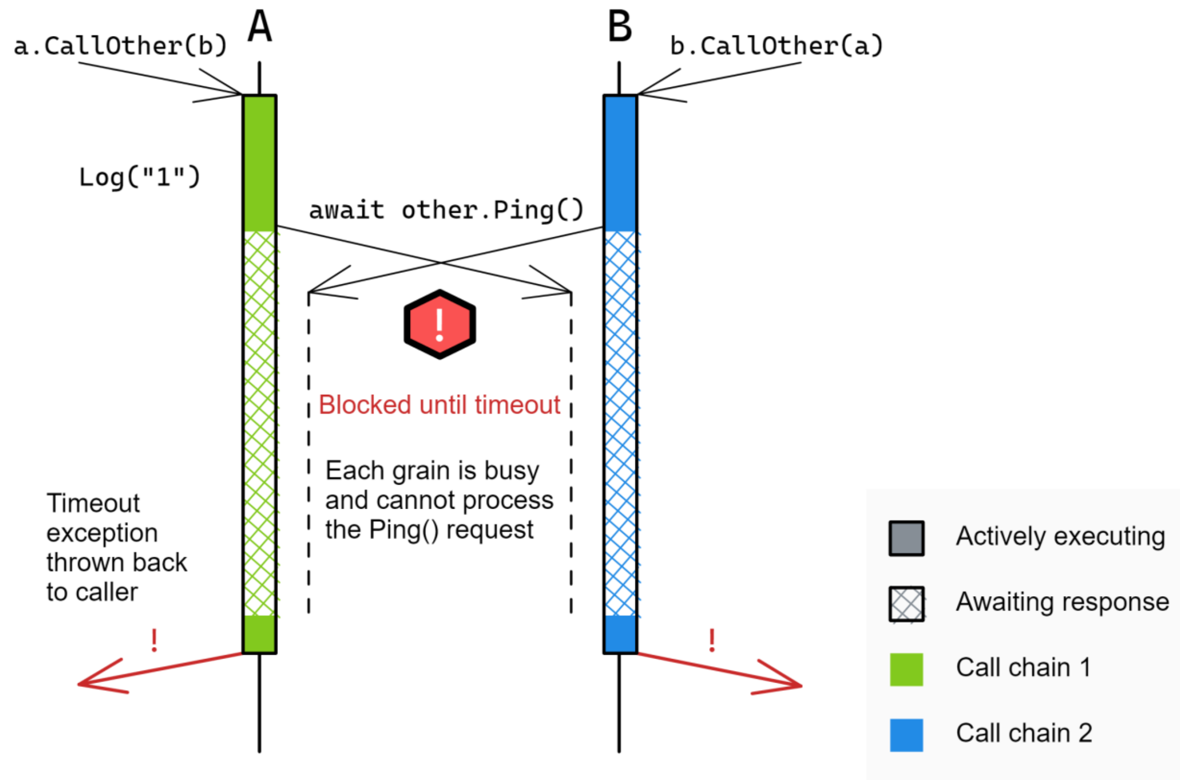
Inside an actor:

```
var grainB = this.GrainFactory.GetGrain<IGrainB>(id);
```

Orleans: Actor mailbox addresses are full typed

Actor model implementations on Orleans – Deadlock

Single thread: Actors process requests one at time



https://learn.microsoft.com/it-it/dotnet/orleans/grains/request-scheduling