Simplifying Distributed Systems with Microsoft Orleans

Lindsey Broos

.NET Consultant @ Team4Talent

Microsoft MVP

Pluralsight author

Board member @ Visug

Content crew member @ Techorama BE + NL

www.lindseybroos.be

lindsey@snowball.be

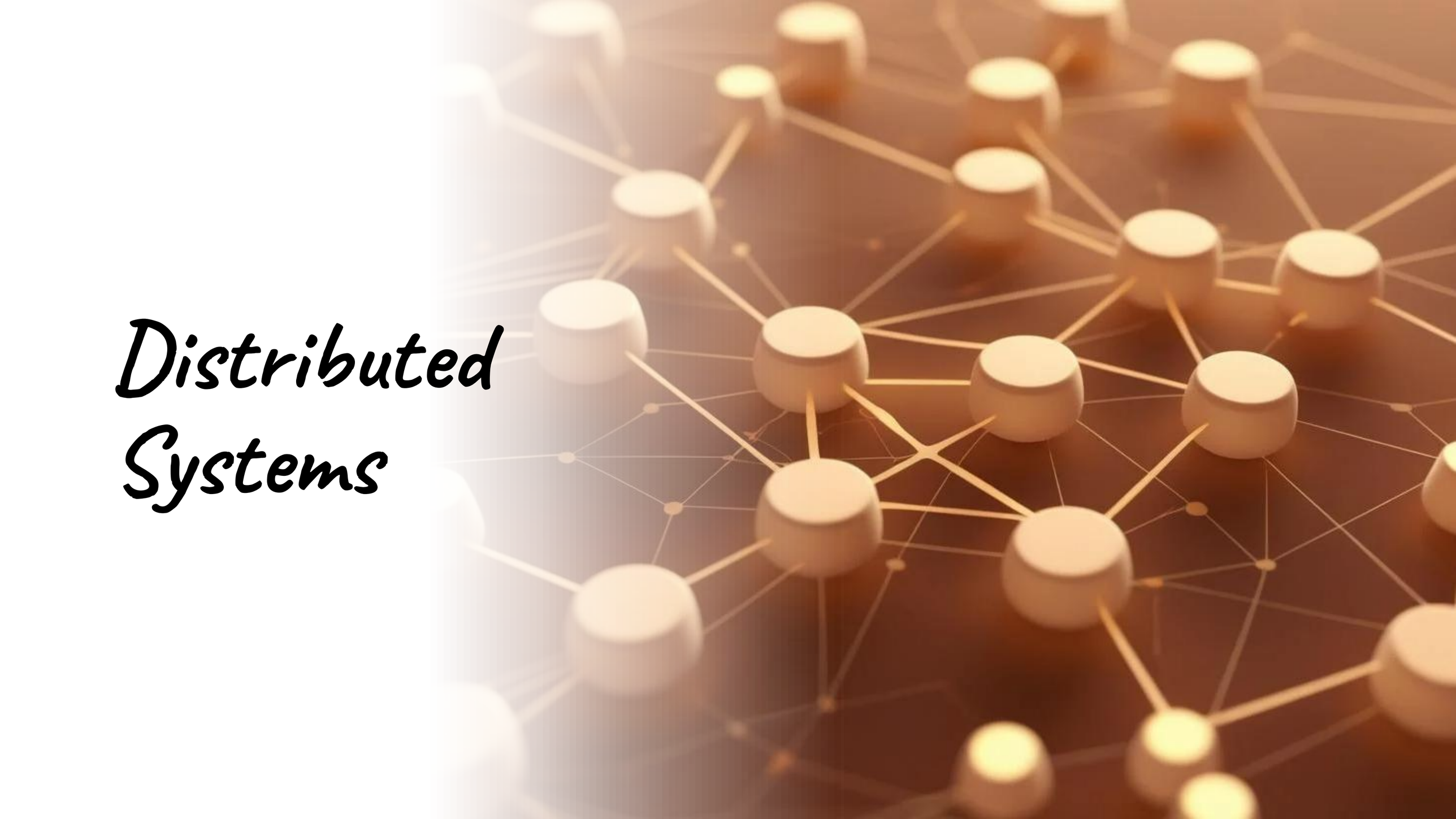
linkedin.com/in/lindsey-broos

Agenda

Distributed systems and their challenges

How does Microsoft Orleans make your life easier?

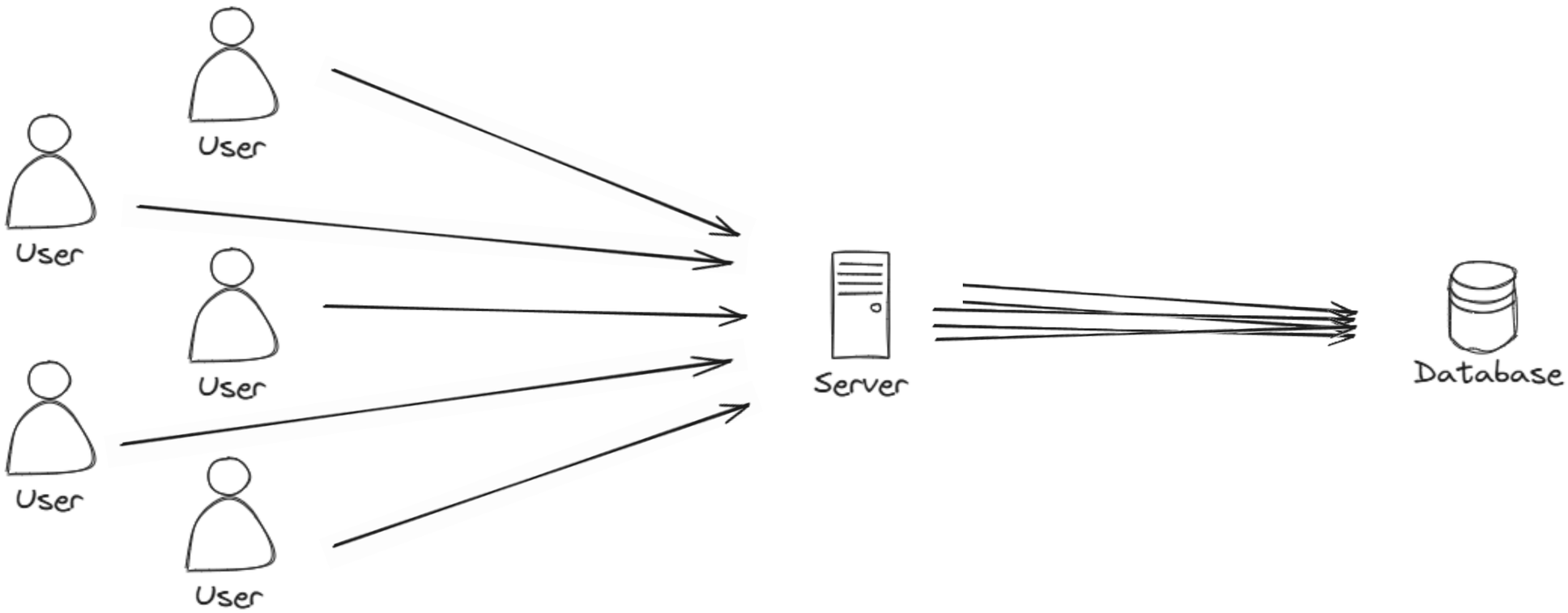
Distributed Systems



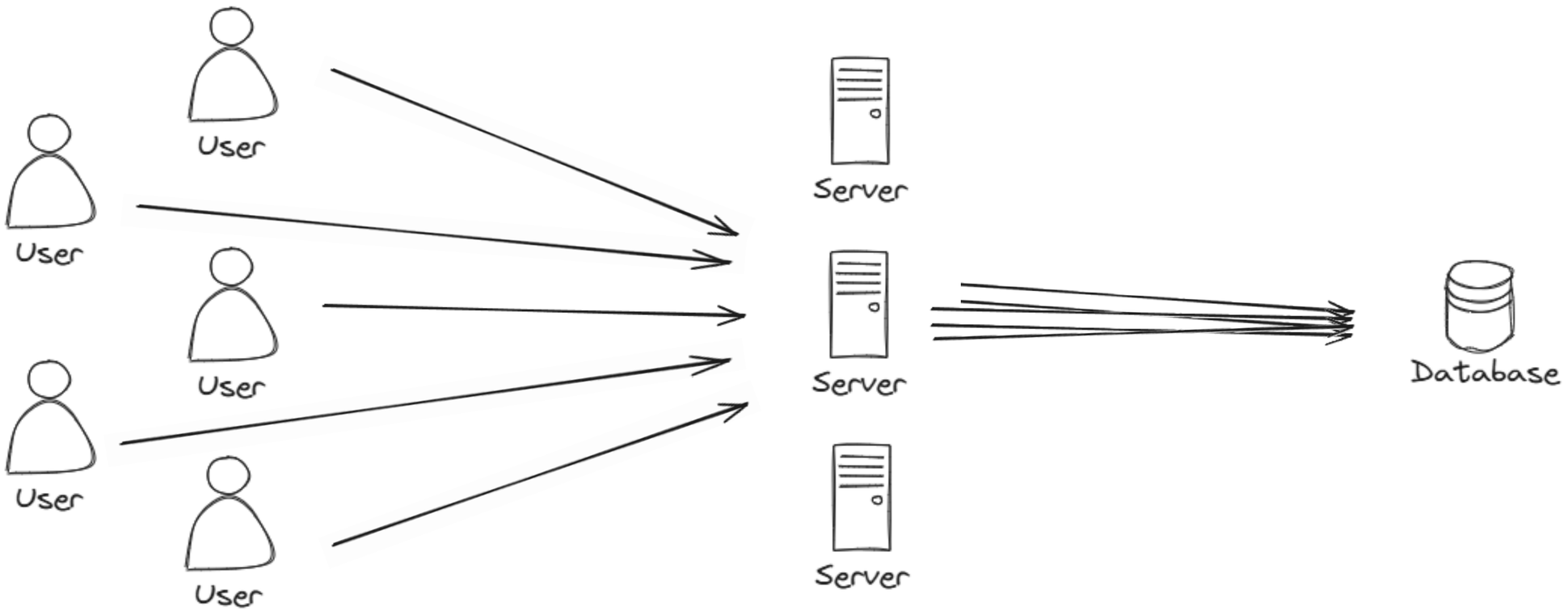
What is a distributed system?

A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another.

Distributed Systems



Distributed Systems



Challenges when building distributed systems

Scalabilty

Data Management

Concurrency

Fault Tolerance

Latency

Microsoft Orleans

What is Microsoft Orleans?

A cross-platform framework
that simplifies building scalable,
fault-tolerant, and stateful applications
using a virtual actor model.

The actor model

First described in 1973 by Hewitt, Bishop and Steiger

Basic building blocks

Actors can
- Receive messages
- Send messages
- Create other actors

Virtual actors

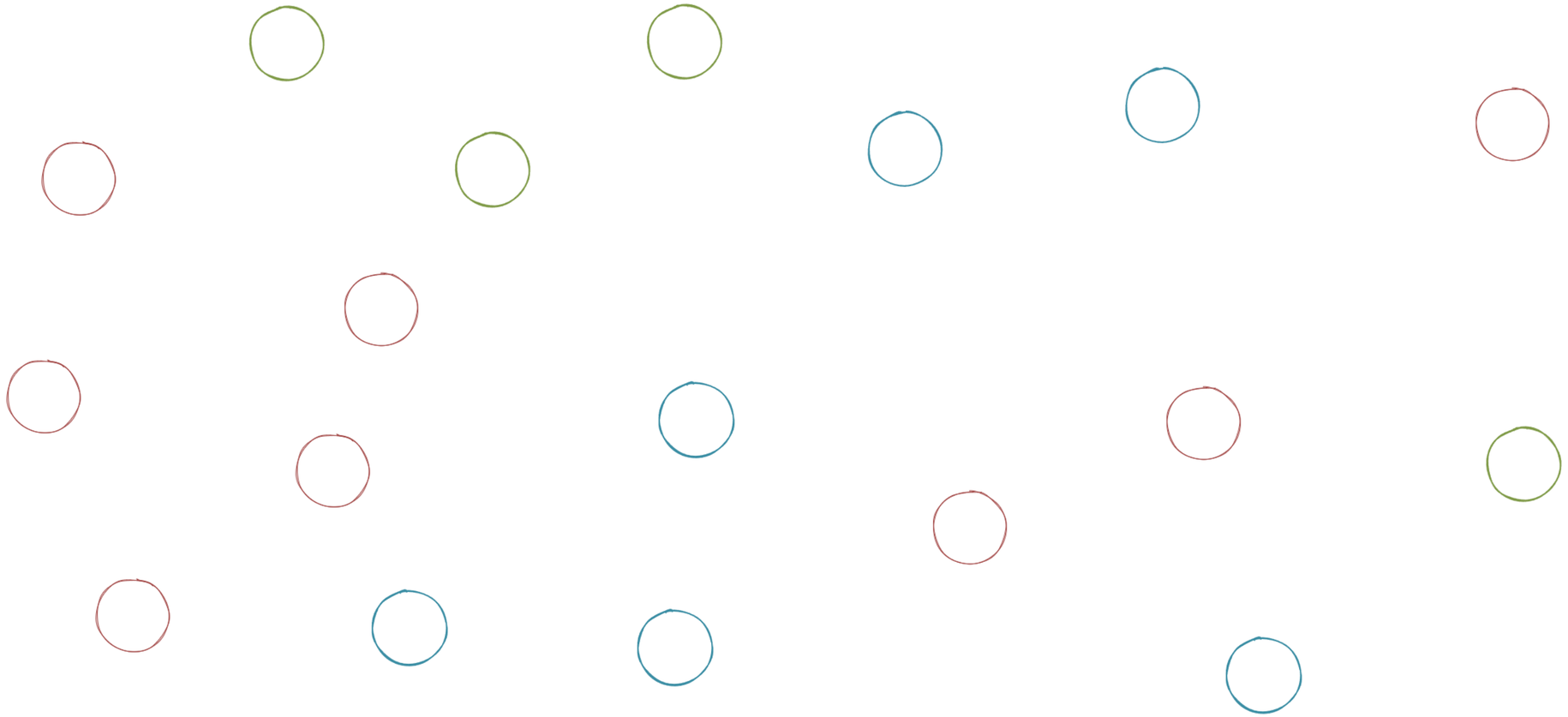
Virtually always exist

Automatic instantiated

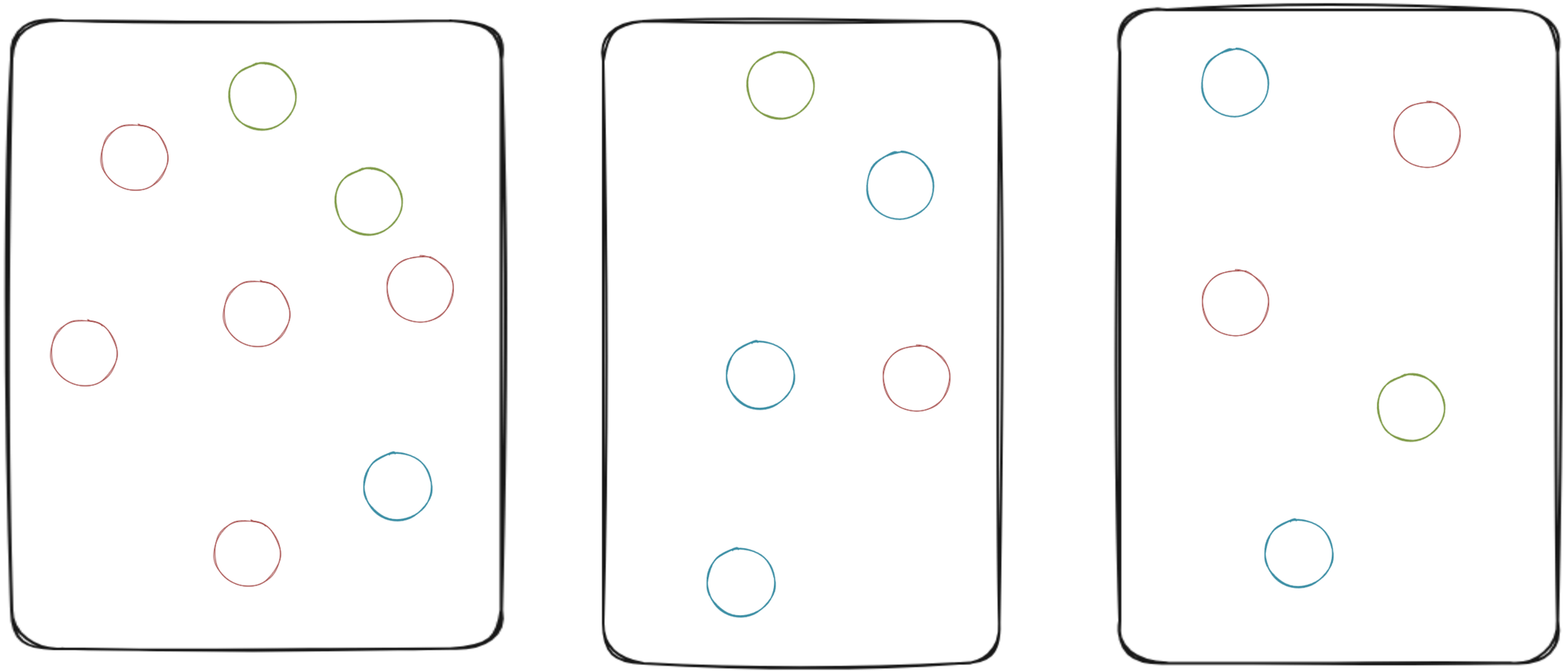
No need to know the location

Auto scale-out

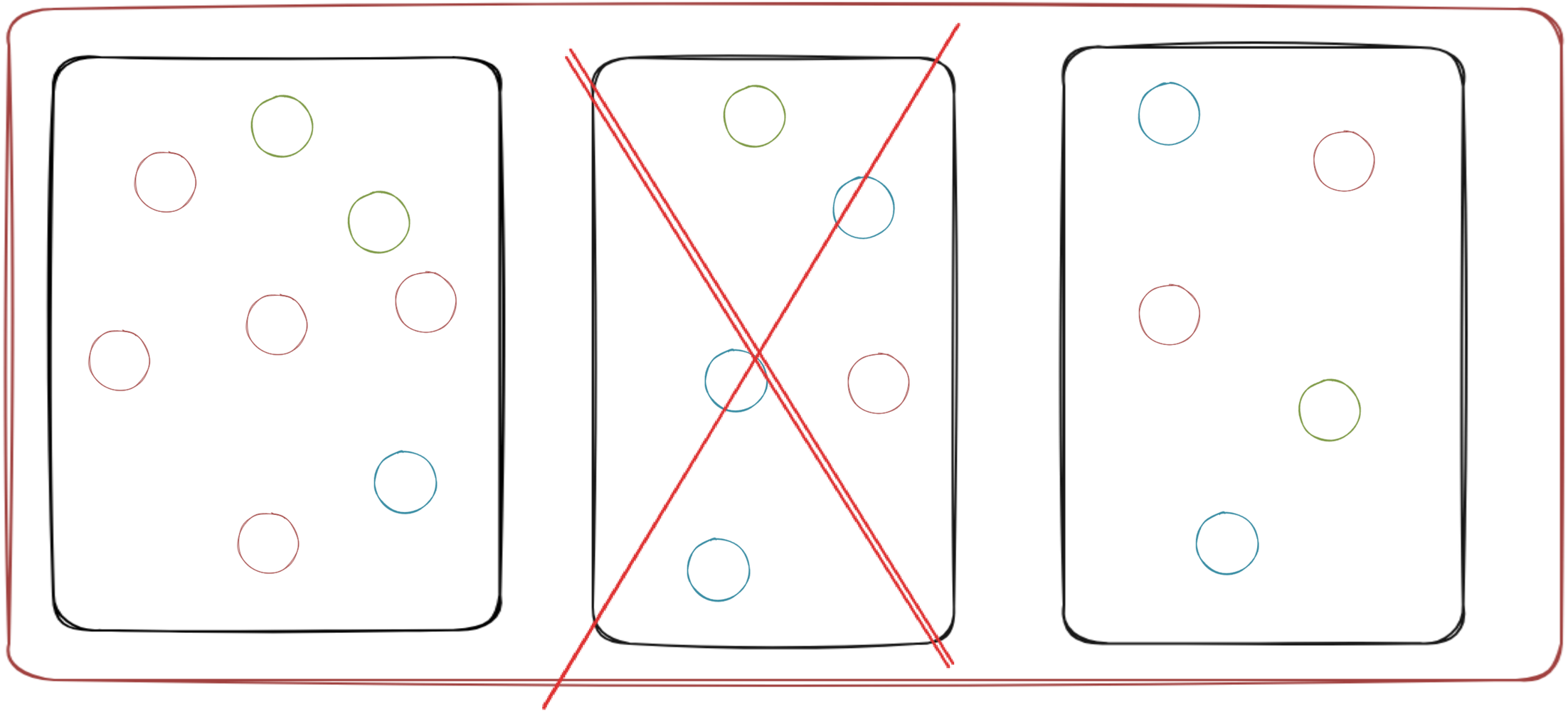
Grains



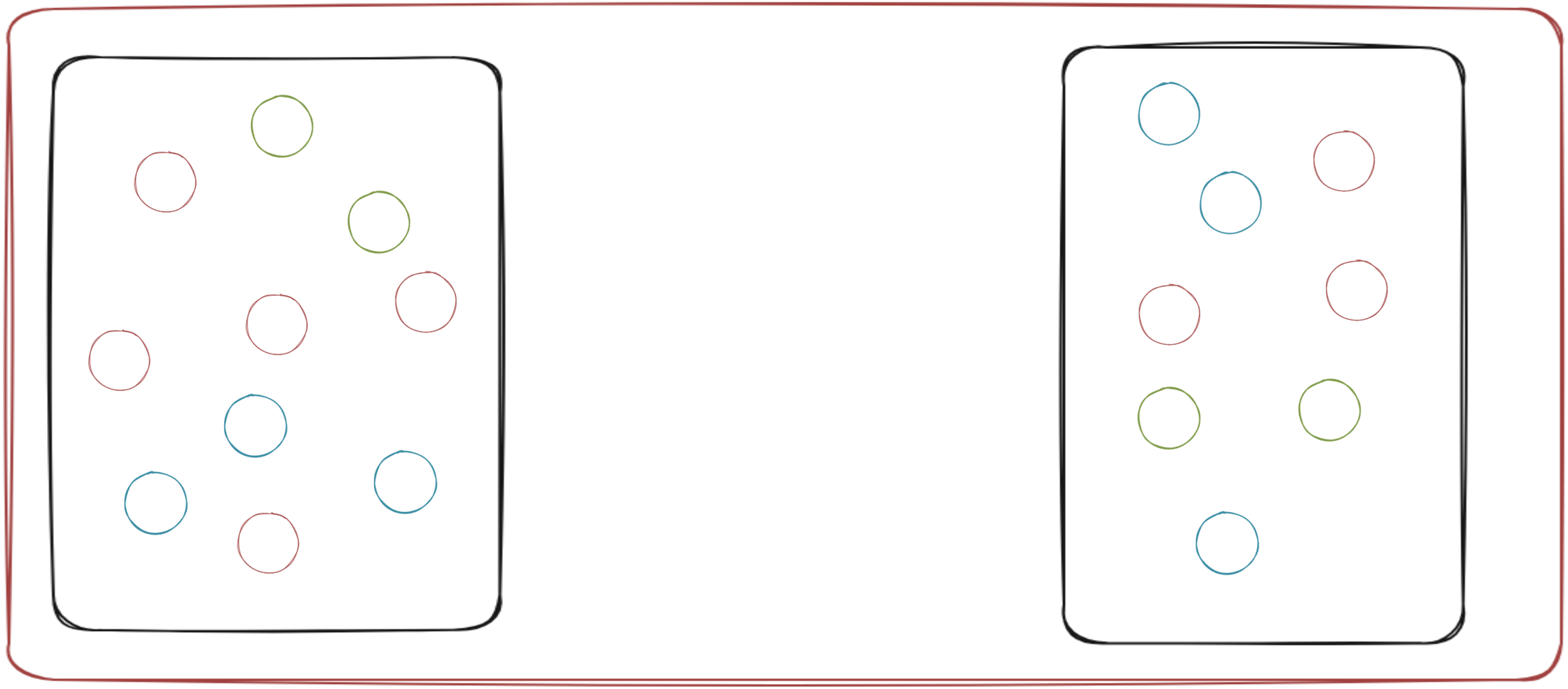
Grains – silo



Grains – silo - cluster



Grains – silo - cluster



A sidestep: .NET Aspire

- A set of powerful tools, templates and packages for building distributed applications
- Improves the developer’s experience when building distributed applications

DEMO: TicketRush

Grains

.NET objects

Single-threaded

Asynchronous

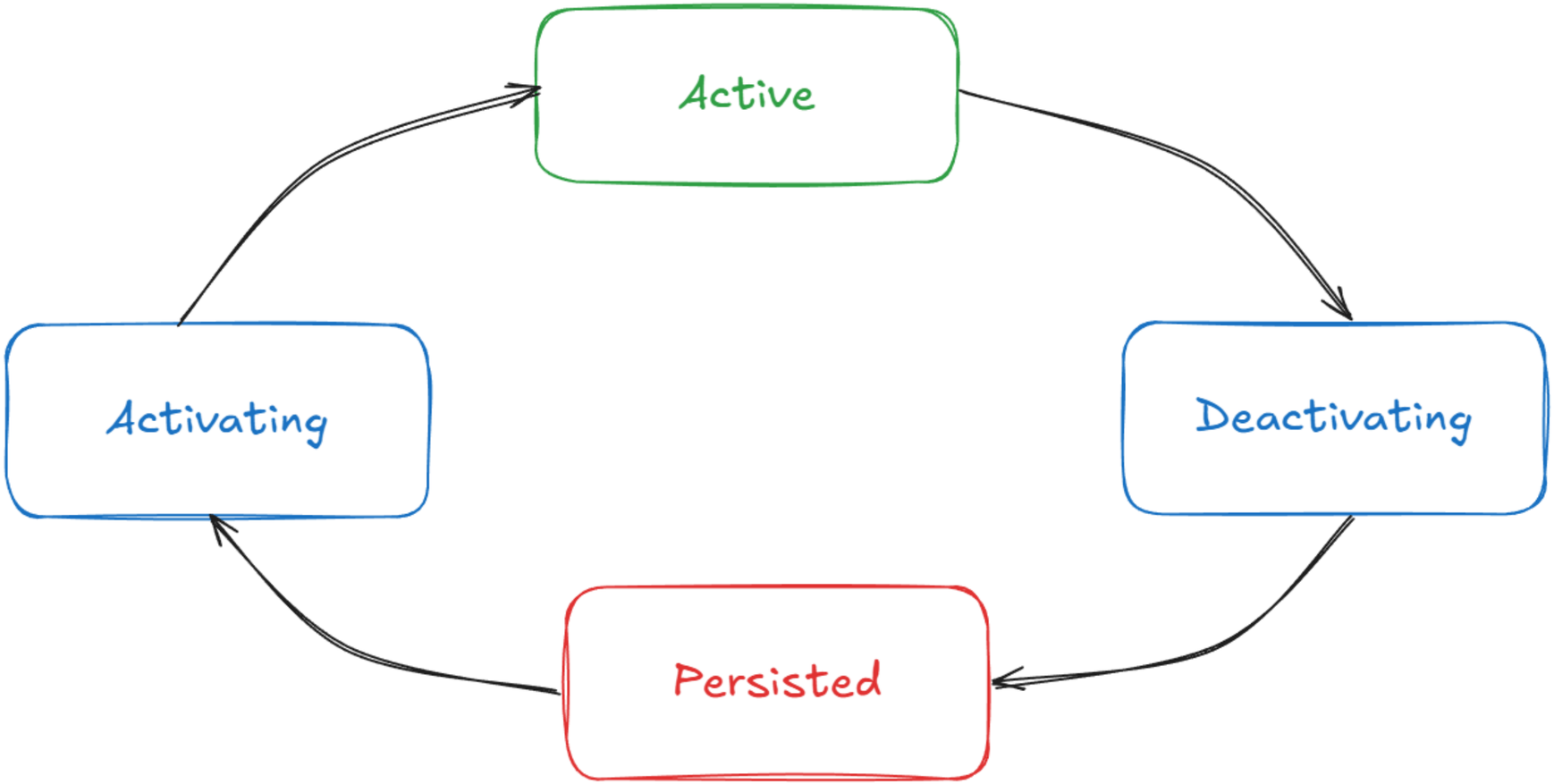
Messages

Grains = identity + behavior [+ state]

```
public interface IConcertGrain : IGrainWithIntegerKey
{

}
```

Grain lifecycle



Grains = identity + behavior [+ state]

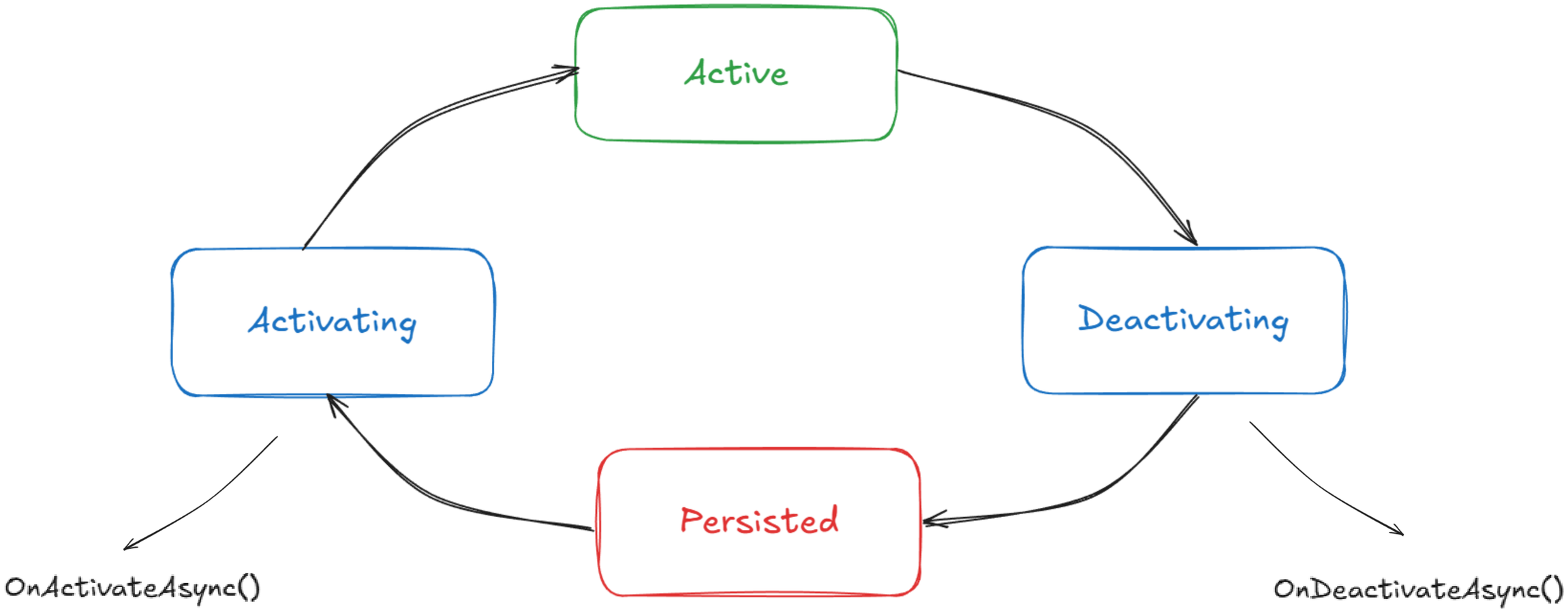
```
public interface IConcertGrain : IGrainWithIntegerKey
{
    Task<int> GetAvailableTickets();
    Task<decimal> GetPrice();
    Task<string?> BuyTicket(Guid userId);
}
```

Grains = identity + behavior [+ state]

```
public class ConcertGrain: Grain, IConcertGrain
{
        private List<string> ticketIds;
        ...
}
```

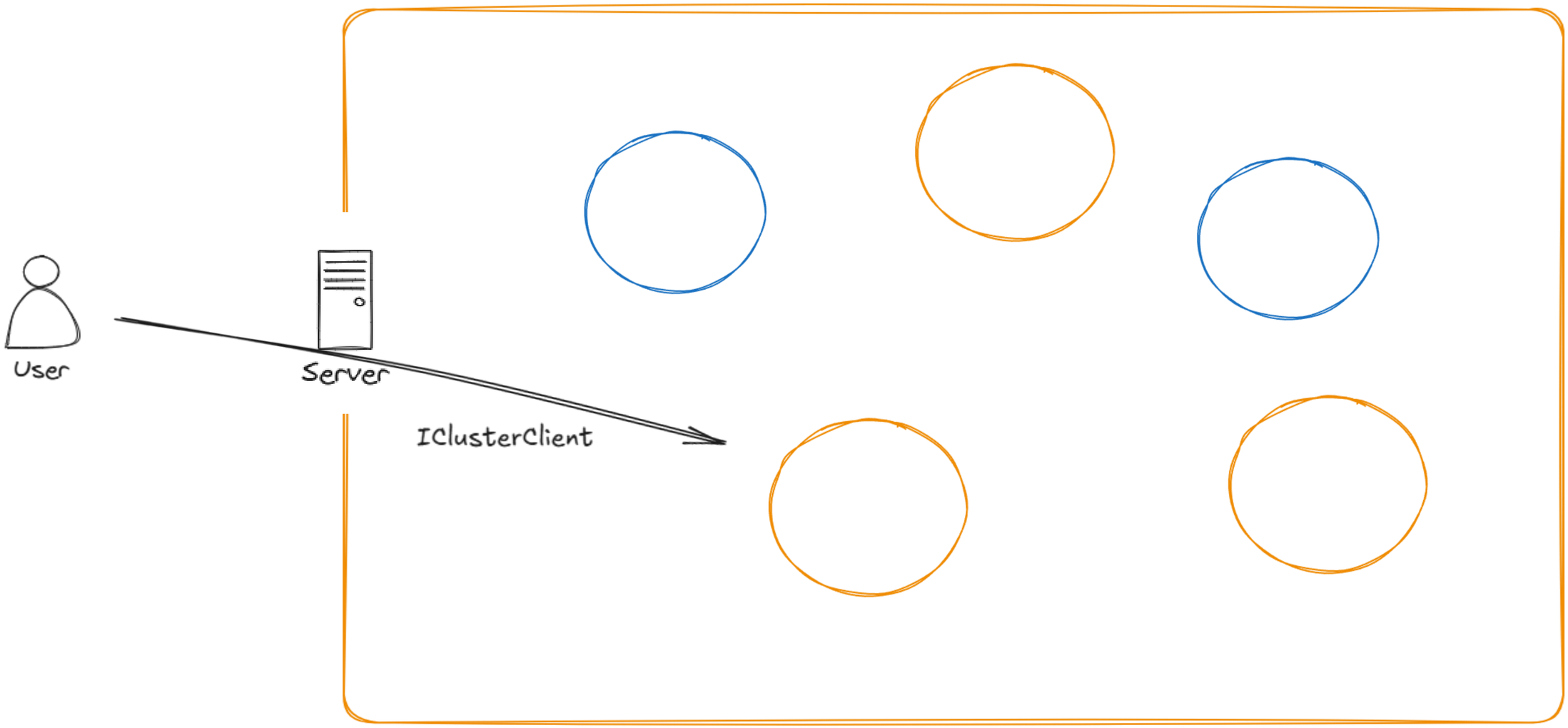
```
public class ConcertGrain: Grain, IConcertGrain
{
        private readonly IPersistentState<List<string>> _ticketIds;
        ...
}
```

Grain lifecycle



DEMO: Grains

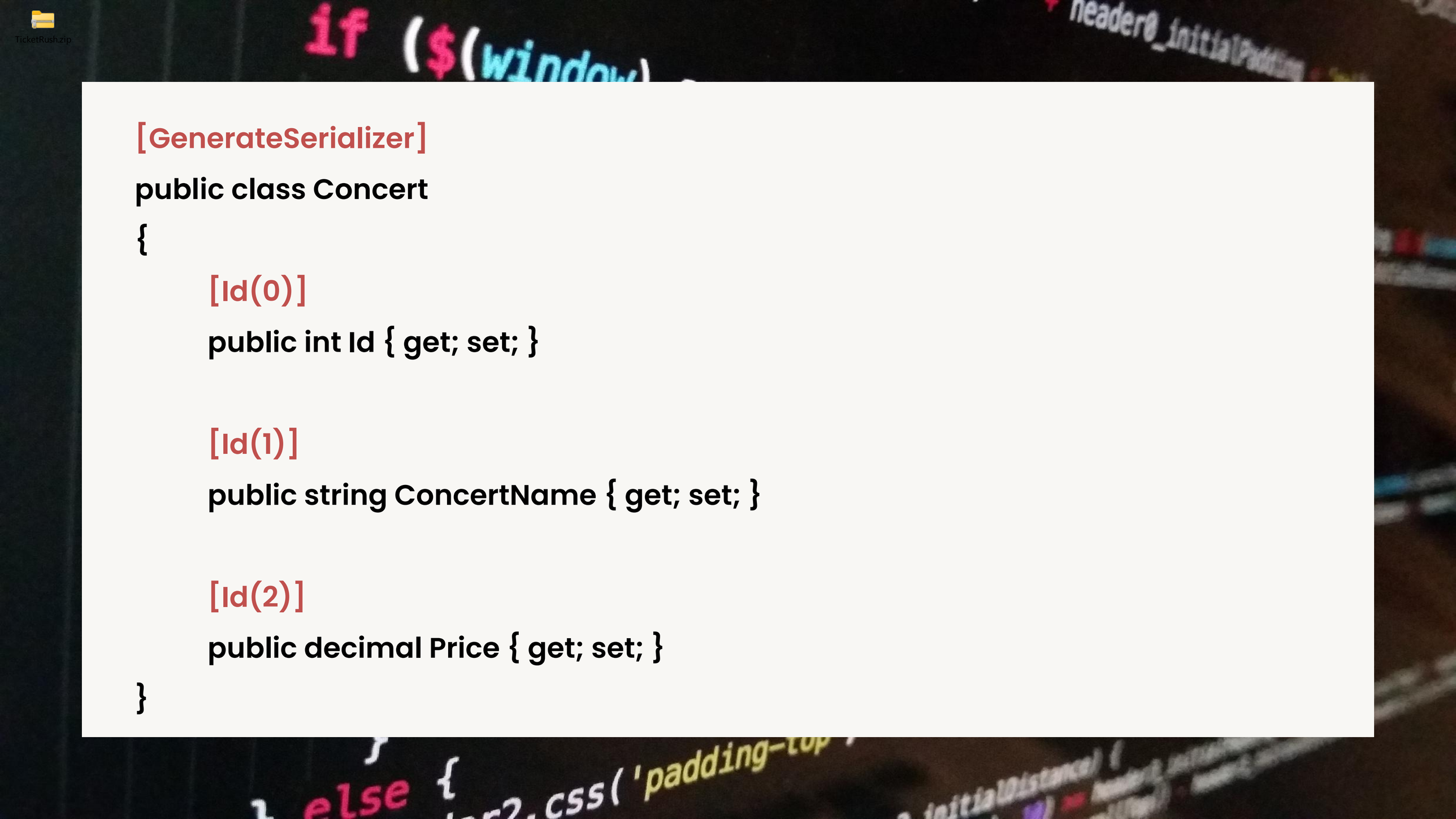
Grain communication




```
public class NewConcertSimulator
{
        private readonly IClusterClient _clusterClient;

        public NewConcertSimulator(IClusterClient clusterClient)
        {
                _clusterClient = clusterClient;
        }

        public Task CreateConcert(Concert concert)
        {
                var concertGrain = _clusterClient.GetGrain<IConcertGrain>(concert.Id);
                await concertGrain.SaveConcert(concert);
        }
}
```

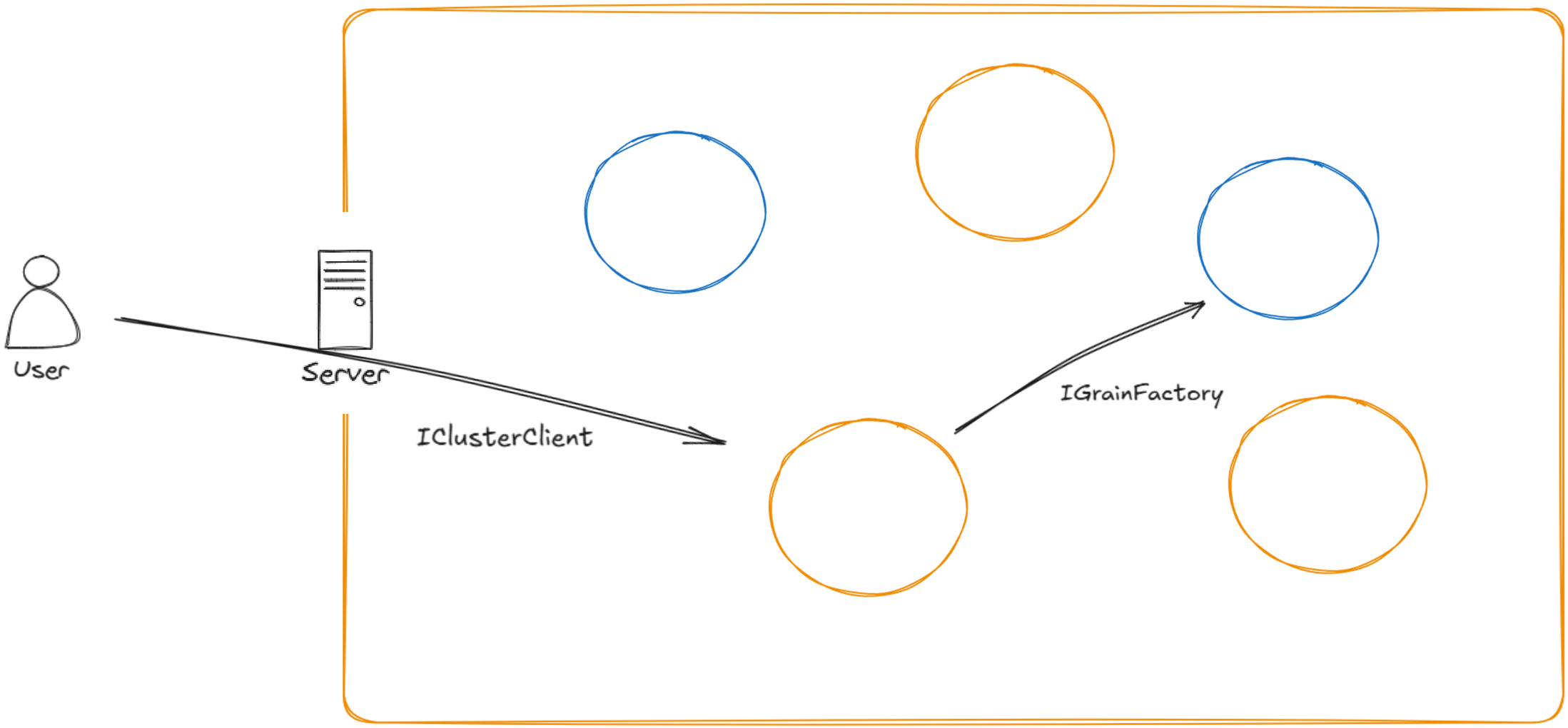



```
[GenerateSerializer]
public class Concert
{
    [Id(0)]
    public int Id { get; set; }

    [Id(1)]
    public string ConcertName { get; set; }

    [Id(2)]
    public decimal Price { get; set; }
}
```

Grain communication




```
public class UserGrain: Grain, IUserGrain
{
        private readonly IGrainFactory _grainFactory;

        public UserGrain(IGrainFactory grainFactory)
        {
                _grainFactory = grainFactory;
        }


        public async Task<bool> BuyTicket(int concertId)
        {
                var concertGrain = _grainFactory.GetGrain<IConcertGrain>(concertId);
                var ticketId = await concertGrain.BuyTicket(this.GetPrimaryKey());
        }
}
```

DEMO: Communication

Other useful features

Persistent state

Timers and Reminders

Observers

Streams

Transactions

...

DEMO: Other features

Is Microsoft Orleans suitable for every project?


Questions?

Thank you!