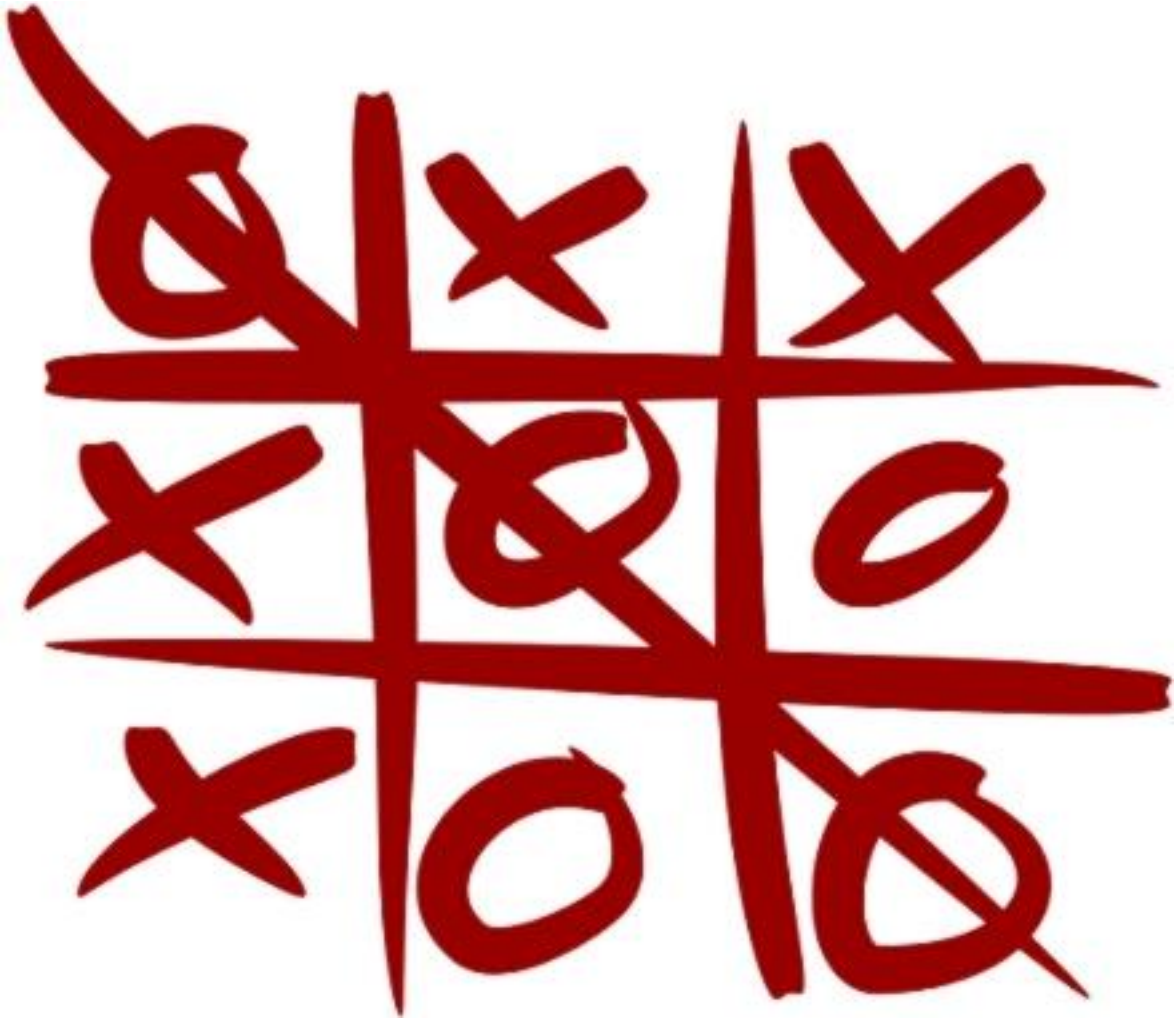
Microsoft

Astrid Hackenberg

Sr. Program Manager
Technical Evangelism &
Development - EMEA



Microsoft

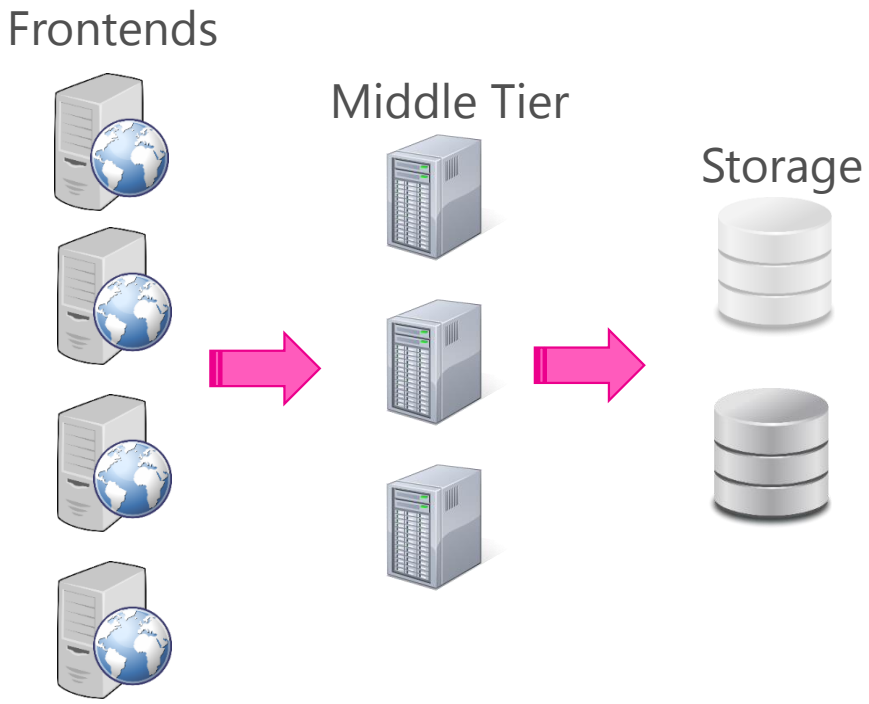
introducing Orleans and the Actor model

Problem: concurrency is hard*

Distribution, high throughput and low-latency make it even harder

*Correctness, performance and robustness are difficult to achieve in the face of concurrency

‘Traditional’ approach – 3 tier architecture



3 tier architecture moves concurrency problems to the database

- Stateless frontends
- Stateless middle tier
- Storage is the challenge and can become a bottleneck
  - latency, throughput, scalability, partitioning

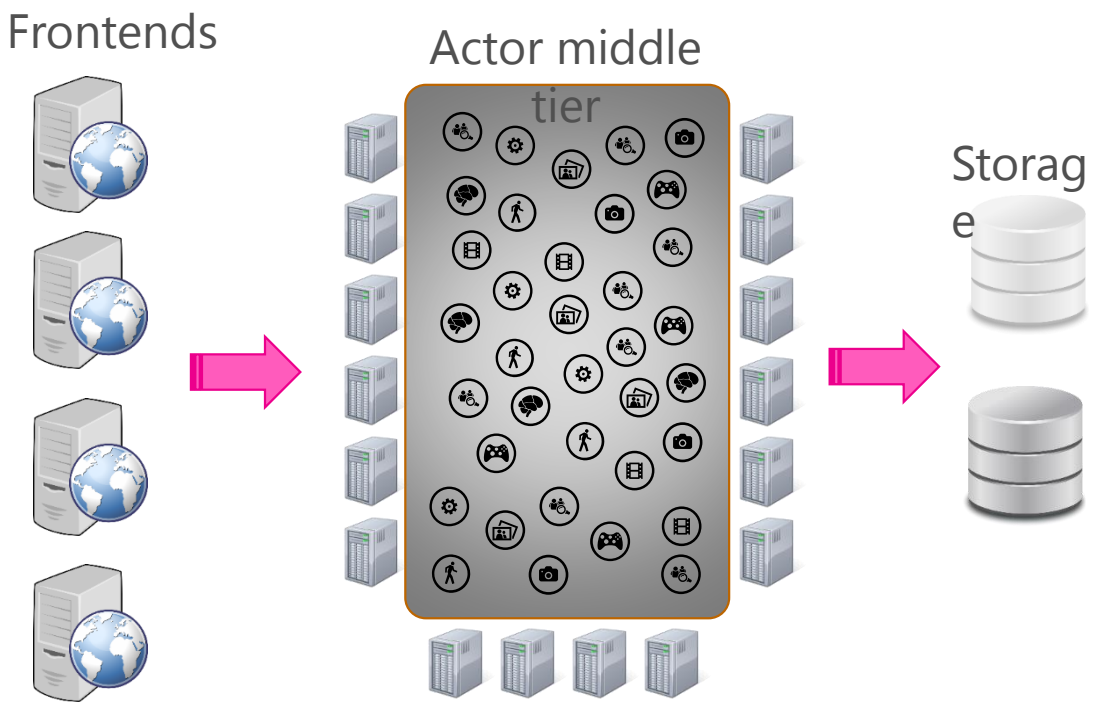
The Actor Model

Mathematical theory of computation
Introduced in 1973 by Hewitt, Bishop, and Steiger, influenced by Lisp, Simula and Smalltalk
Popularised by Joe Armstrong with Erlang in 1986, Erlang is behind CouchDB, Riak, RabbitMQ

A framework and basis for reasoning about concurrency, with Actors as primitives of concurrent computation

An Actor needs 3 things:
- Processing : logic / behaviour
- Storage : maintain internal state (mutable)
- Communication : send messages to other actors

Actor model as stateful middle tier



- Performance of cache
- Rich semantics
- Concurrency control
- Horizontal calls are natural
- OOP paradigm regained

Virtual Actors in “Orleans”: Grains

Game Grain Type

Game Grain (Instance) #2,548,308

Game Grain (Instance) #2,031,769

Game Grain #2,548,308
Activation #1 @ 192.168.1.1

Game Grain #2,031,769
Activation #1 @ 192.168.1.5

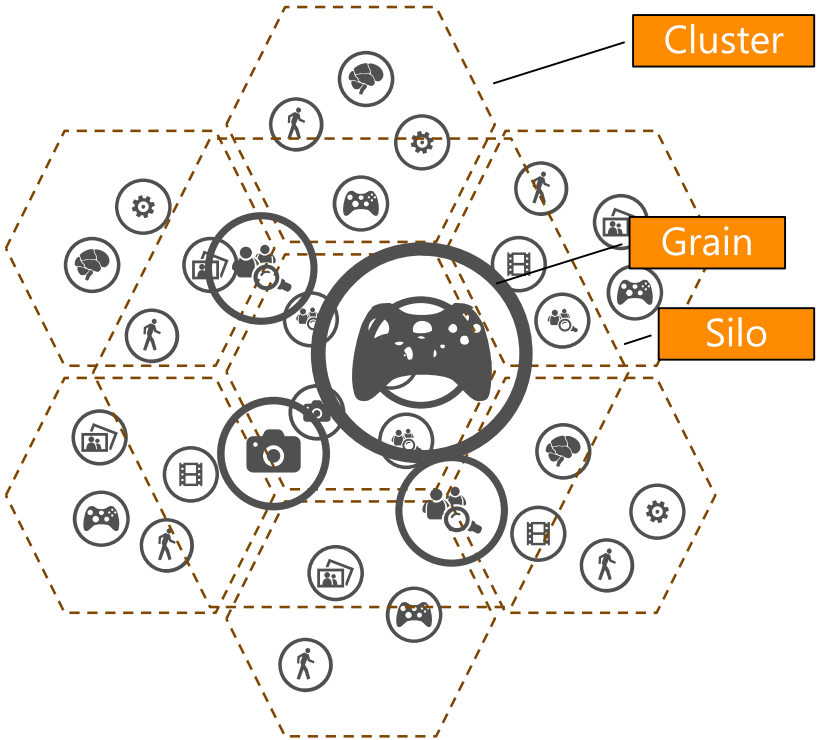
Grain Type

Grain (Instance)

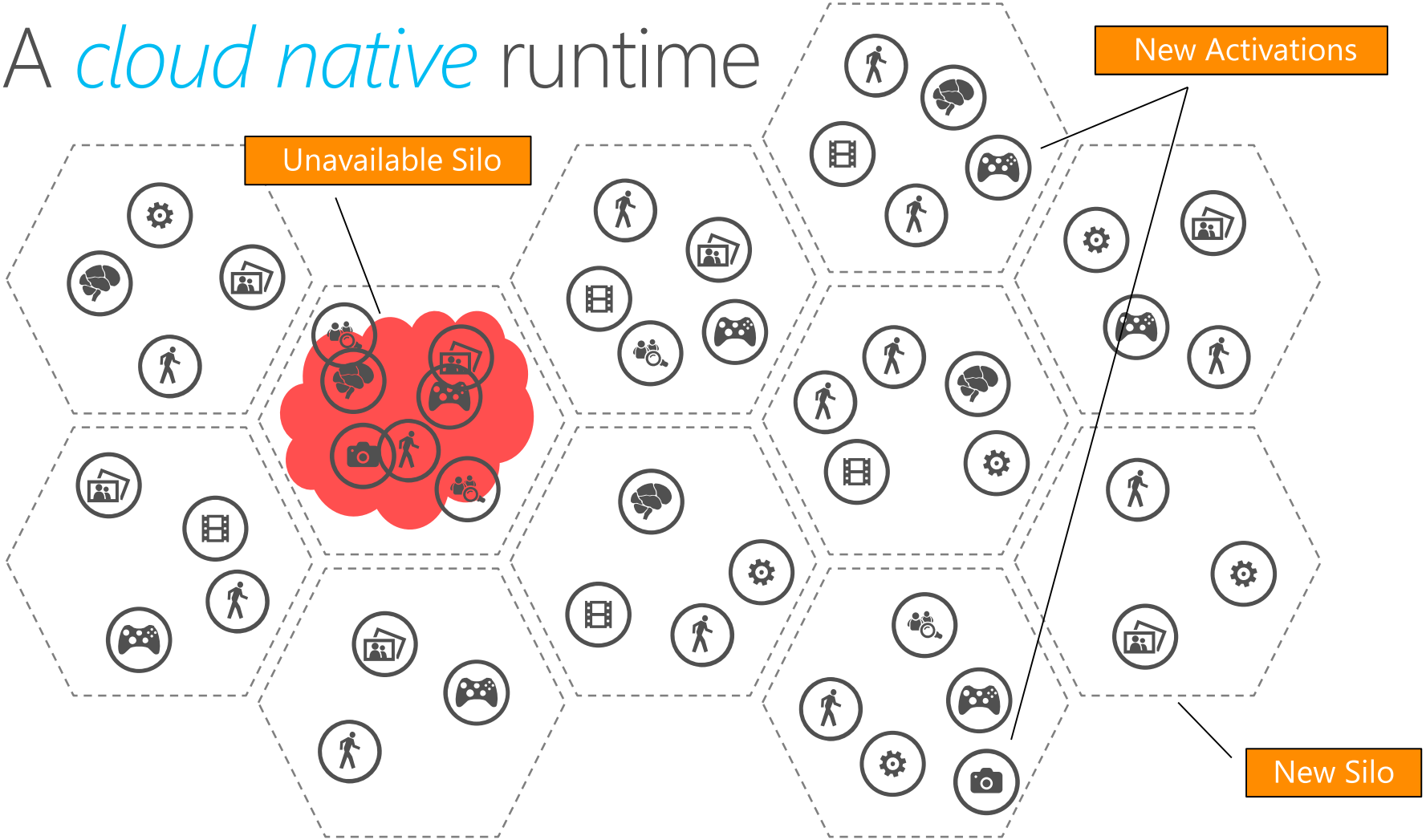
Grain Activation

Project “Orleans”

- Distributed Actor runtime
  - Virtual Actor model
  - Location transparency
- Built for .NET
  - Actors (*Grains*) are .NET objects
  - Messaging through .NET interfaces
  - Asynchronous through async/await in C#
  - Automatic error propagation
- *Silo*: runtime execution container
  - Implicit activation & lifecycle management
  - Coordinated placement
  - Multiplexed communication
  - Failure recovery



A *cloud native* runtime



Microsoft

Developing with Orleans

Grain interfaces

Marker interface to indicate grain interfaces

```
public interface IDevice : IGrain
{
    Task<string> MyStatus(int temperature);
}
```

All grain interface methods must be asynchronous

Implementing the grain type

Base class for all basic grain plumbing functionality

```
public class DeviceGrain : GrainBase, IDevice
{
    Task<string> MyStatus(int temperature)
    {
      var resp = "I am: " + (temperature>25 ? "sick" : "good") ;
      return Task.FromResult(resp);
    }
}
```

Talking to grains

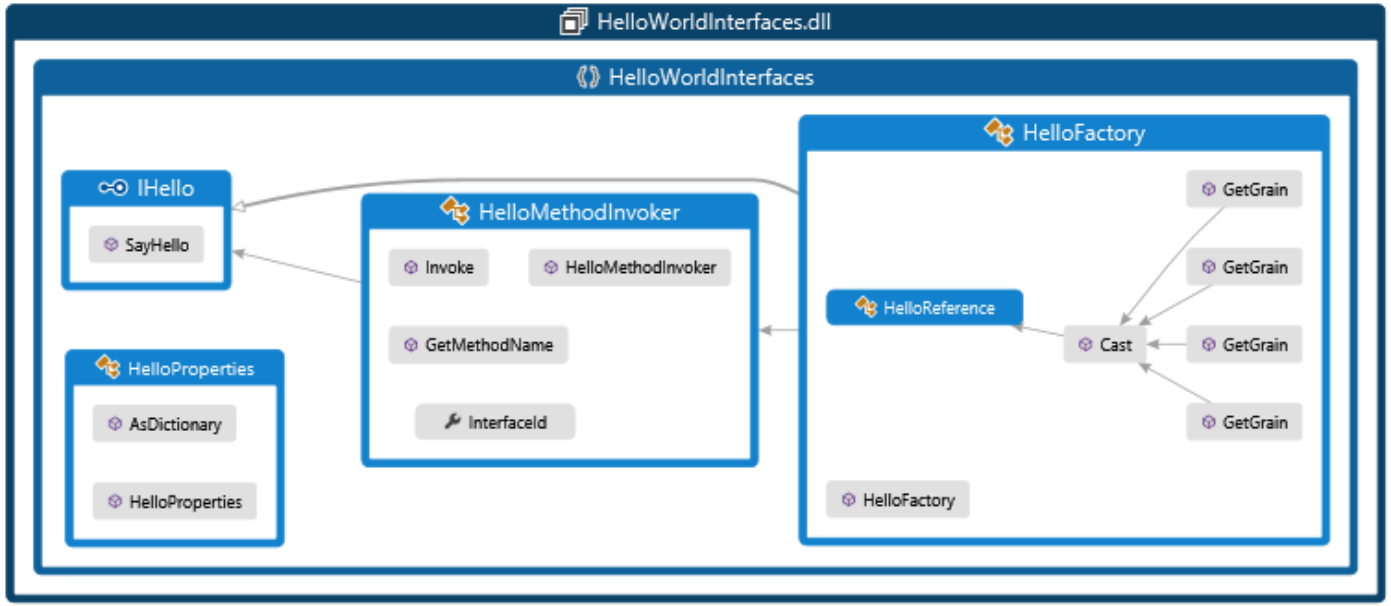
Factory Class is auto-generated at compile time

Grain Id (long , GUID or String)

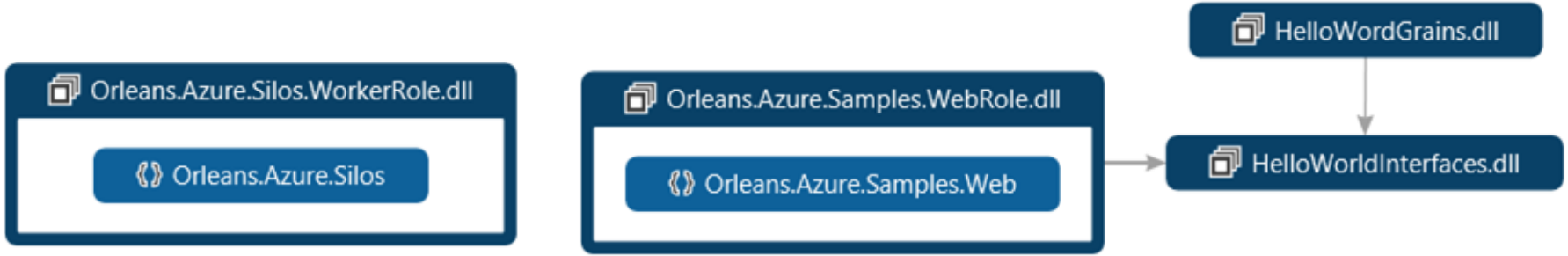
```
private async static Task SendMessage(long grainId)
{
    IDevice thing = DeviceFactory.GetGrain(grainId);
    string response = await thing.MyStatus(30);
    Console.WriteLine("Response: {0}", response);
}
```

the magic of IGrain

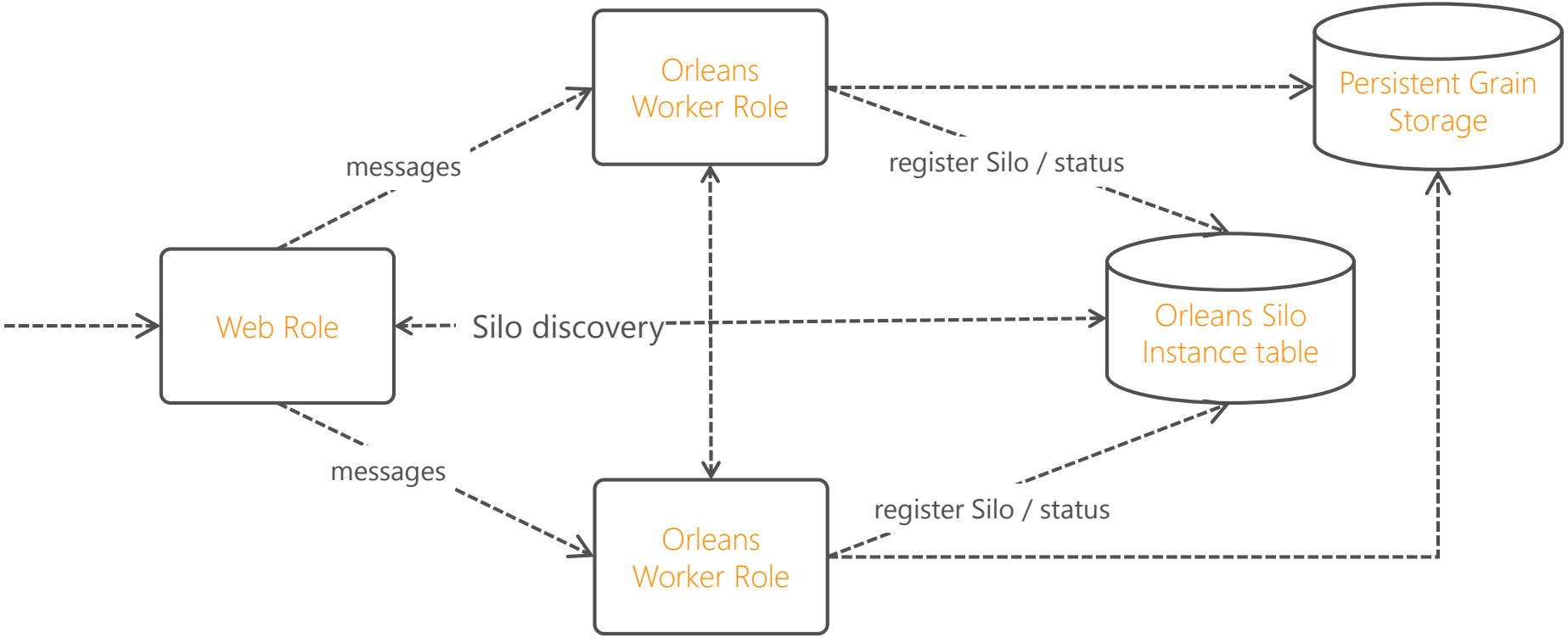
orleans.codegen.cs



Sample – Hello World on Azure



Orleans on Azure



Grains

- single activation (default)
  - only one instance of every grain,
- stateless worker
  - more then 1 instance can exist (per Silo)
  - no state reconciliation between instances
  - typically grains without local state or with static state
- grain reference
  - proxy object that implements the corresponding grain interface
  - can be passed as argument to a method call
  - used for communication between client code and hosted grain (Orleans Client library)

Grain persistence

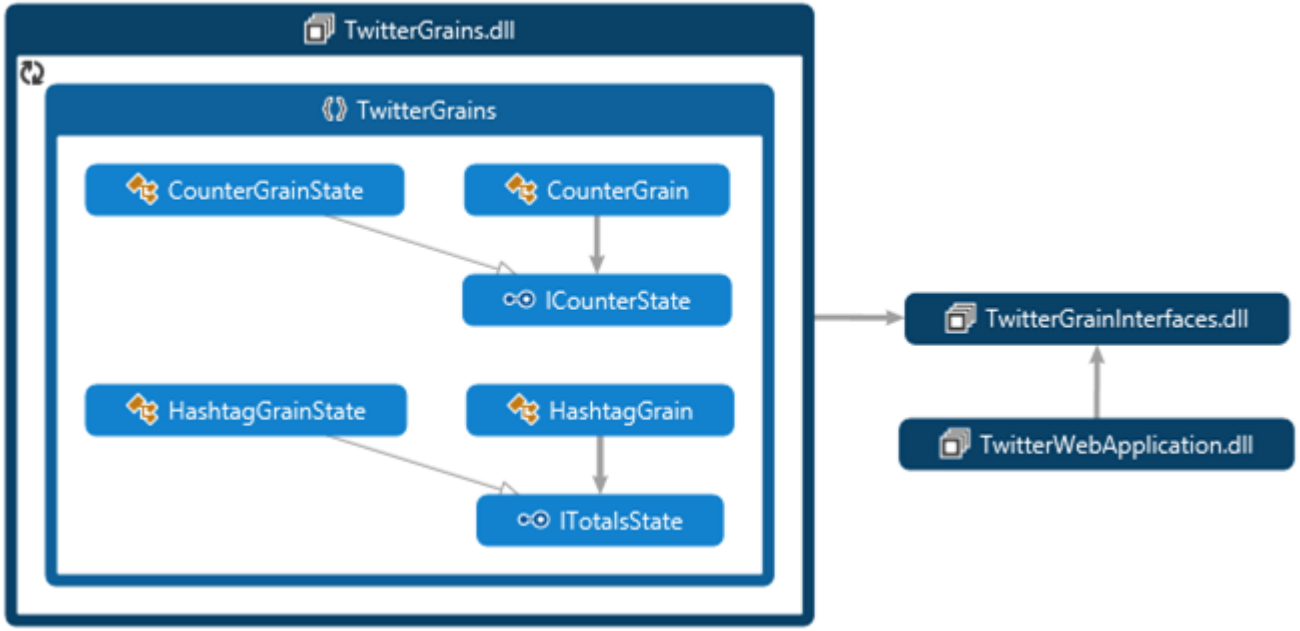
Orleans system managed persistence framework

Inherit Orleans.GrainBase<T>
<T> must be a type that implements IGrainState

[Storage] attribute specifies storage provider
provider is defined in the silo config file
Different grain types can use the same storage provider
Different grain types can use different configured storage providers, even if both are the same type

Grain state is automatically READ on grain activation
WRITE must be triggered explicitly for any changed grain state

Sample – Twitter Monitor



Microsoft

use cases for Orleans

Sample solutions

Adventure

- *Text Adventure game like Zork*
- Local client app using local Silo
- Grain relationship and dependency

AzureWebSample

- *Hello World web site*
- Orleans in Azure
- Running Silos in worker roles

Chirper

- *Chat application*
- Publish / Subscriber(Orleans Observer) model
- Multiple Interface implementation
- Grain pre-initialization

GPSTracker

- *Position and speed of device tracking*
- Internet of Things
- Web role with
  - SignalR. OWIN for messages
  - Orleans Silo
- PushNotifier is StatelessWorker

HelloWorld

- *Hello World console app*
- Usage of IGrain and GrainBase
- Start/stop Silo from code and run Silo in separate AppDomain

Presence

- *Multi-player game monitor*
- Typical game grains game & player
- Multi-cast messages (player notification)
- Stateless Presence grain (true Stateless worker)

StorageProviders

- *Technology test application*
- *Grain persistency*
- Provider implementation for JSON file storage and MongoDB storage

TicTacToe

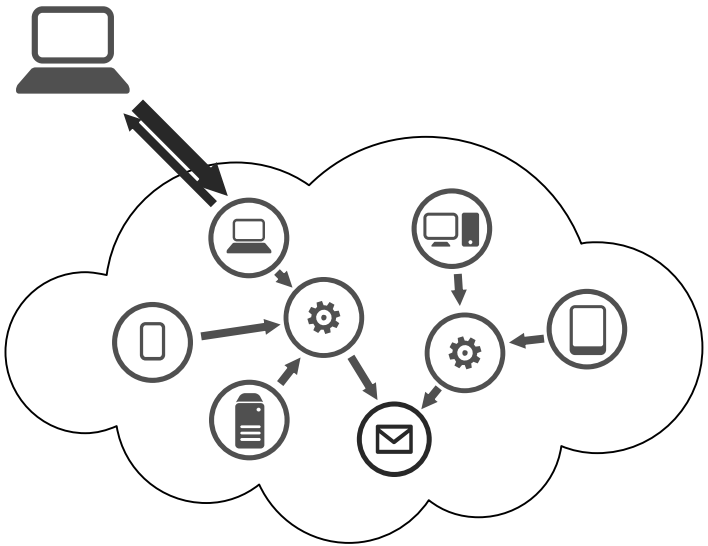
- *TicTacToe web game*
- Typical game grains game & player
- Singleton Pairing grain using cache
- Update screen with WindowsTimer

TwitterSentiment

- *Web monitor #hashtag sentiment*
- Complete solution
- Dispatcher handles multiple grain updates from client (hashtags)
- Advanced implementation; count total, string grain id (hashtag)

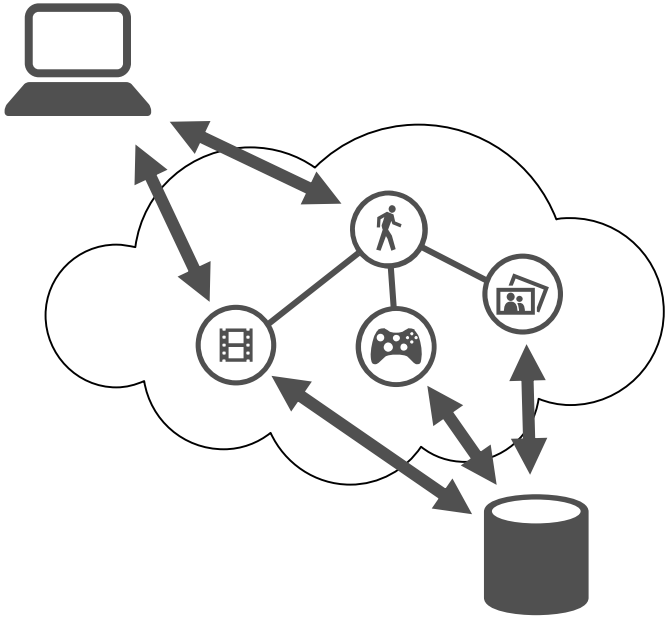
[Near] real-time analytics

- Devices send telemetry to the Cloud
- Per-device actors process and pre-aggregate incoming data
- Multi-level aggregation by actors
- Statistics, predictive analytics, fraud detection, etc.
- Control channel back to devices
- Grouping by location, category, etc.
- Elastically scales with # of devices



Intelligent cache

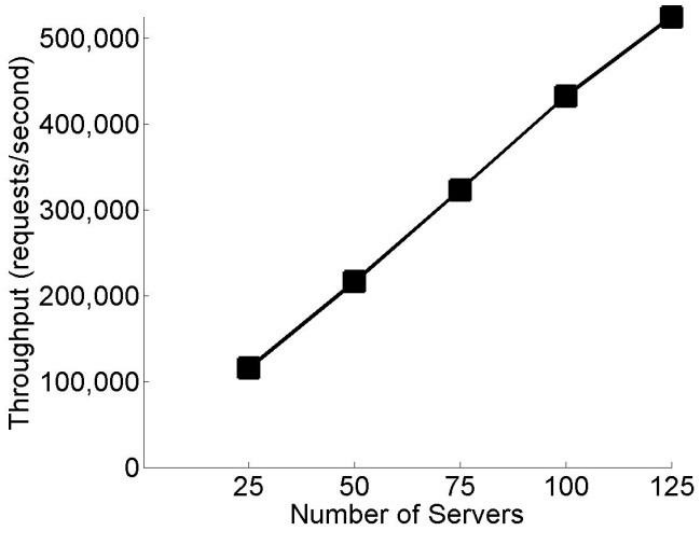
- Actors hold cache values
- Semantic operation on values
- Function shipping (method calls)
- Coordination across multiple values
- Automatic LRU eviction
- Transparent on-demand reactivation
- Write-through cache with optional batching



Scalability by default

- Near linear scaling to hundreds of thousands of requests per second
- Efficient resource usage
- Location transparency simplifies scaling up or down
- Complements Azure PaaS
- Easily adjust scale over time

Test Lab Numbers



Project “Orleans”

Distributed Actor runtime hosting virtual Actors with cloud-based scale-out & elasticity

Programming model and runtime that brings OOP (back to) cloud-based development

References

- Background Microsoft Research
- Introduction post
- The framework
- The samples