Microsoft

Building cloud-native services with Project Orleans

Microsoft Research
Faculty Summit
2015
July 8-9, 2015

Cloud-Native Services

Term coined by Hoop Somuah

Services that are built for the cloud

- Reliable
- Scalable
- Elastic
- High throughput, low latency
- Fast to build and iterate
- DevOps friendly

No lift-and-shift

Microsoft

What is Project "Orleans"?

Oversimplifying: "Distributed C#"

- Orleans runs your .NET objects on a cluster as if in a single process
- Define .NET interfaces and classes, deploy, send requests to them

Practically: "Toolset for building cloud-native services"

- Encapsulates best practices for building cloud-native services
- Framework for stateful near-real-time backends
- 3-5x less and simpler code to write, scalability by default

Academically: "Distributed virtual actor model"

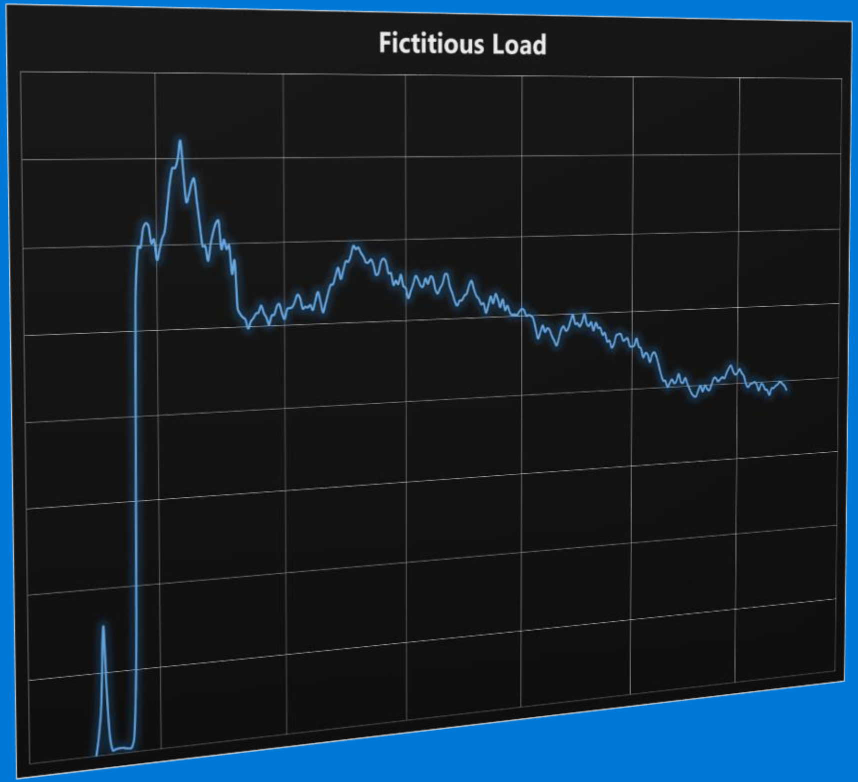
- Adaptation of the Actor Model for challenges of the Cloud
- Actors that exist eternally and never fail

Microsoft

HALO WAYPOINT


HALO
WAYPOINT
VIEWING TTL L ASKAN
RETURN TO YOUR RECORD
SERVICE RECORD
LAST PLAYED YESTERDAY
TTL L ASKAN
LSKN
CSR 49
SR 130
BIG TEAM BATTLE
PATHFINDER - 10
COMPLETED
MAX RANK
WAR GAMES MATCHMAKING
2294 GAMES COMPLETED
1443 WINS
13D 03H TOTAL PLAYTIME
WAR GAMES CUSTOM
35 GAMES COMPLETED
18 WINS
03H 23M TOTAL PLAYTIME
SPARTAN OPS
0 / 50 COMPLETED SOLO
0 / 50 COMPLETED COOP
00H 00M TOTAL PLAYTIME
CAMPAIGN
0 / 8 MISSIONS COMPLETED
SOLO
CO-OP
RECENT GAMES
STATS SUMMARY
MOST PLAYED VARIANT STATS
MOST USED WEAPON
SPECIALIZATIONS
COMMENDATIONS
SHARE
RECENT GAMES
OLDER
RECENT
HAVEN SLAYER WAR GAMES - YESTERDAY 160 SCORE 12 KILLS 4TH PLACE
SHUTOUT SLAYER WAR GAMES - YESTERDAY 215 SCORE 14 KILLS 2ND PLACE
SHUTOUT SLAYER WAR GAMES - YESTERDAY 250 SCORE 17 KILLS 6TH PLACE
HAVEN SLAYER WAR GAMES - YESTERDAY 180 SCORE 14 KILLS 5TH PLACE
SHUTOUT SLAYER WAR GAMES - YESTERDAY 125 SCORE 10 KILLS 3RD PLACE
STATS SUMMARY
OVERVIEW
13D 06H 17M TOTAL PLAYTIME
11.22.12 PLAYER SINCE
52 SPARTAN POINTS
229 CHALLENGES COMPLETED
0 / 7 TERMINALS FOUND
100% OVERALL COMMENDATIONS
52 LOADOUT ITEMS PURCHASED
WAR GAMES: MATCHMAKING
99%
13D 03H 04M TOTAL PLAYTIME
2294 TOTAL GAMES PLAYED
1443 TOTAL WINS
79349 TOTAL MEDALS
WAR GAMES: CUSTOM
1%
0D 03H 23M TOTAL PLAYTIME
35 TOTAL GAMES PLAYED
18 TOTAL WINS
753 TOTAL MEDALS
SPARTAN OPS
0%
0D 00H 00M TOTAL PLAYTIME
0 TOTAL GAMES PLAYED
0 / 50 COMPLETED SOLO
0 / 50 COMPLETED COOP
CAMPAIGN
0%
0D 00H 00M TOTAL PLAYTIME
0 TOTAL GAMES PLAYED
0 / 8 CAMPAIGN PROGRESS
MOST PLAYED VARIANT STATS
MATCHMAKING: SLAYER
8D 00H 26M TOTAL PLAYTIME
21287 TOTAL KILLS
43944 TOTAL MEDALS EARNED
1368 GAMES PLAYED
869 GAMES WON
1.48 K/D
CUSTOM: SLAYER
SHUTOUT
TTL L ASKAN
LEGENDARY
SLAYER BR
TEAM SLAYER
50
43
SHARE
VICTORY
BLUE TEAM
2 PLACE
215 SCORE
14 KILLS
MOST KILLED BY
6 KILLS
SAVAGE SILVA
ROID
KILLED MOST
5 KILLS
SAVAGE SILVA
ROID
TEAM RESULTS
VIEWING TTL L ASKAN'S GAME RESULTS
OUTCOME
COMBAT
K/D
1 TOSSEM JAHH CSR 26 270 SCORE 17 KILLS 34 MEDALS CSR 26 THIS GAME
2 TTL L ASKAN LSKN CSR 27 215 SCORE 14 KILLS 29 MEDALS CSR 26 THIS GAME
3 TTL YETI YETI CSR 17 200 SCORE 11 KILLS 29 MEDALS CSR 15 THIS GAME
4 TTL KYLI3 KYLI CSR 26 145 SCORE 8 KILLS 21 MEDALS CSR 24 THIS GAME
1 SAVAGE SILVA ROID CSR 34 240 SCORE 18 KILLS 29 MEDALS CSR 34 THIS GAME
2 TRULYLEGENDARYX KCID CSR 21 155 SCORE 11 KILLS 20 MEDALS CSR 21 THIS GAME
3 PDT WOLTEY WOLF CSR 23 130 SCORE 7 KILLS 19 MEDALS CSR 23 THIS GAME
4 ICOLIC BC BC CSR 24 105 SCORE 7 KILLS 14 MEDALS CSR 22 THIS GAME
TEAM COMPARISON
VIEWING TTL L ASKAN'S GAME RESULTS
50
43
29
8
6
-8
27
22
KILLS
ASSISTS
SPREAD
HEADSHOTS
MEDAL DISTRIBUTION
VIEWING TTL L ASKAN'S GAME RESULTS

Patterns of a Big Game Launch



- Huge traffic spike on launch
- Downtime at launch is **really bad**
- Also spikes on weekends and holidays
- Load steadies out over time

Developer Experience

Key Concepts

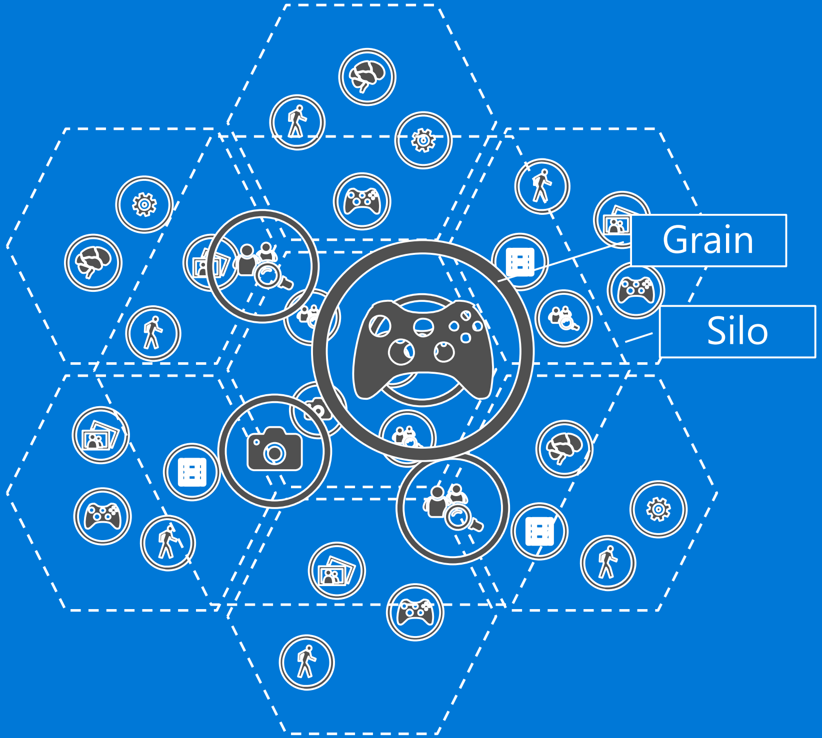
Distributed runtime

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



'Hello World' in Orleans – Interface

```
public interface IHello : IGrainWithIntegerKey
{
    Task<string> SayHello (string name);
}
```

‘Hello World’ in Orleans – Implementation

```
public class HelloGrain : Grain, IHello
{
    private int _counter;

    public async Task<string> SayHello (string name)
    {
        return string.Format(
           “Hello {0}. You are caller #{1}”, name, counter++));
    }
}
```

'Hello World' in Orleans – Invocation

```
GrainClient.Initialize(); // client-only



IHello grainRef = GrainFactory.GetGrain<IHello>(0);

string reply = await grainRef.SayHello (name);

Console.WriteLine("HelloGrain said:" + reply);
```

Beyond 'Hello World' – Grain Interface

```
public interface IUser : IGrain
{
    Task<string> GetName();
    Task SetName(string name);

    Task<string> GetStatus();
    Task UpdateStatus(string status);

    Task<List<IUser>> GetFriends();
    Task AddFriend(IUser friend);
    Task<string> GetFriendsStatus();
    Task<List<string>> GetFriendsUpdates();
}
```

- Grain interface is a.NET interface that extends *IGrain*
- All methods return *Task* or *Task<T>*
- Arguments and return values must be serializable, can be grain references
- Compiler auto-generates proxy classes

Microsoft

Beyond 'Hello World' – Invoking Grains

```
IUser me = GrainFactory.GetGrain<IUser>(myId);
IUser friend = GrainFactory.GetGrain<IUser>(friendId);

try
{
   await me.AddFriend(friend);
   Console.WriteLine("Added friend {0}.", friendId);
}

catch(Exception exc)
{
   Console.WriteLine("Failed to add {0} as friend: {1}", friendId, exc);
   throw;
}
```

- Reference grain interfaces project
- Call GetGrain() to obtain a reference to a grain for a given key
- Invoke interface methods on the reference (proxy)
- Handle returned TPL Task's properly
- Just like in a desktop app

Beyond 'Hello World' – Grains Class

```
public class UserGrain : Grain, IUser
{
    private List<IUser> _friends;
    …
    public async Task<string> GetFriendsStatus()
    {
        var tasks = new List<Task<string>>();
        foreach (var friend in _friends)
            tasks.Add(friend.GetStatus());

        await Task.WhenAll(tasks);

        var sb = new StringBuilder();
        foreach (var t in tasks)
            sb.AppendLine(t.Result);

        return sb.ToString();
    }
}
```

- Extend Grain
- Implement grain interface(s)
- Exclusive access to private fields
- No multi-threading
- Easy parallelism
- Handle returned TPL Task's properly
- Just like in a desktop app

Microsoft

Lots More Features…

Automatic cluster membership, recovery from failures

Automatic resource management, elasticity

Flexible placement policies

Grain timers and reminders

Support for persistence with a provider model

Support for streaming event processing

…

Microsoft

Orleans Benefits

Very easy to program reliable distributed/cloud systems

Scalability by default

Uncompromised performance

Proven in many production services

Runs anywhere

Open source!

Microsoft

How You Can Benefit

A vibrant open source project to leverage

- Easy enough for undergrads
- Deep enough for PhD students
- Architected for the Cloud, great fit for IoT, social, gaming, even workflow

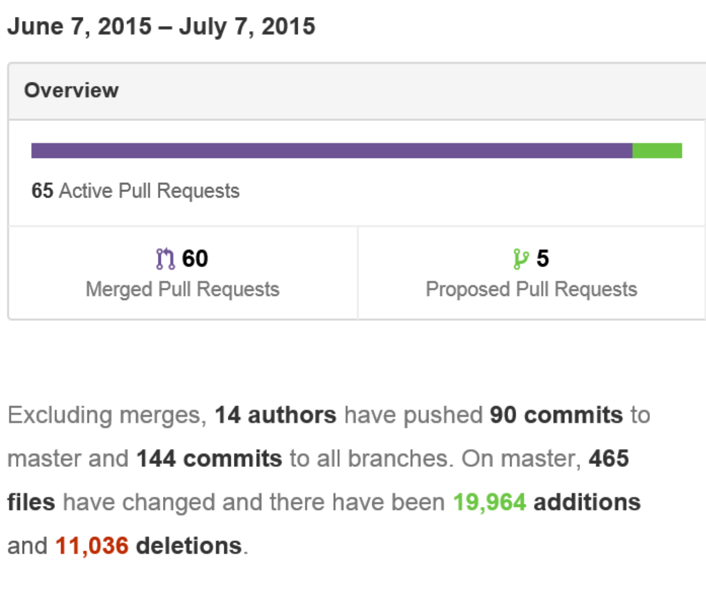
Build distributed scalable apps/services/systems in ‘user’ mode

Build system components/algorithms in ‘kernel’ mode (runtime)

Contribute to code used in production systems

Microsoft

Orleans Is Open Source



- On GitHub under an MIT license
- GitHub is the ‘master branch’
- Active and growing community that never sleeps
- Easy to contribute
- Pride of ownership – priceless
- *Join and enjoy the fun!*

Microsoft

Orleans on GitHub:
https://github.com/dotnet/orleans

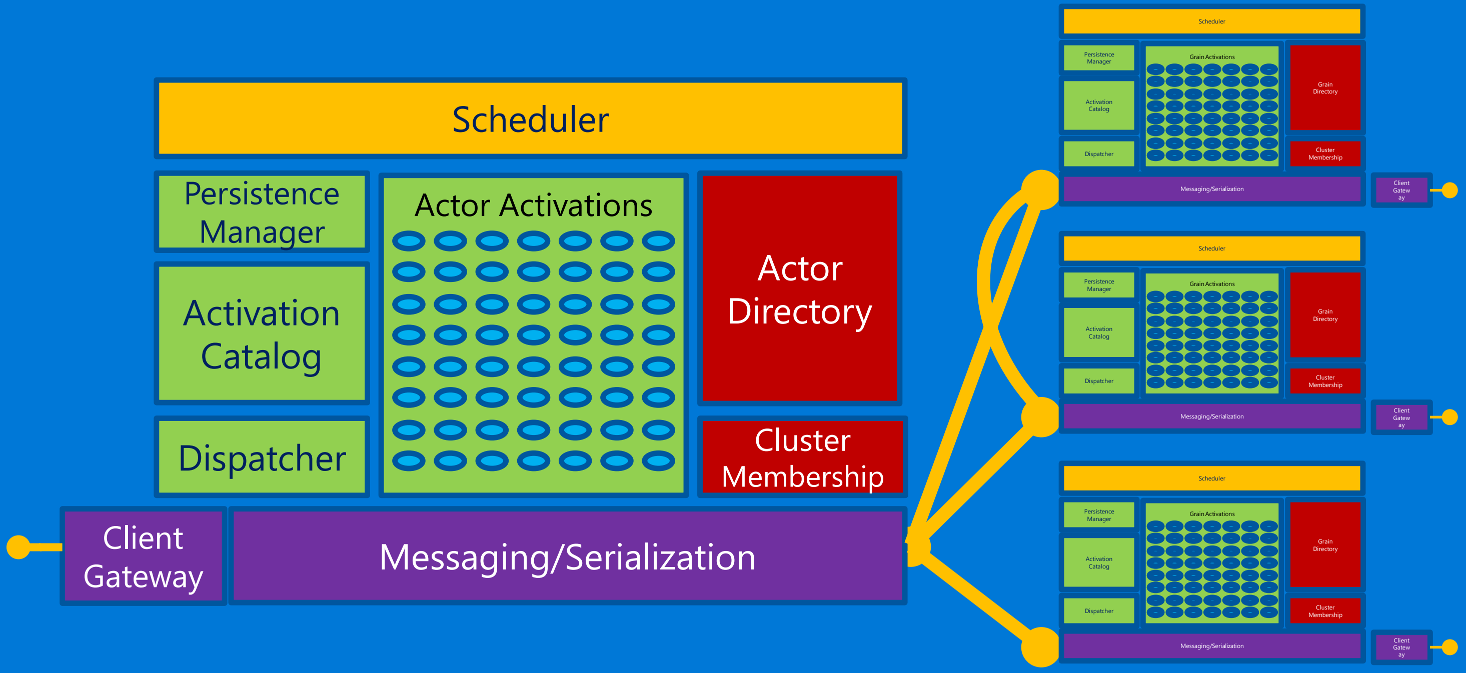
Documentation:
http://dotnet.github.io/orleans/

Ideas for Research and Course Projects:
http://dotnet.github.io/orleans/Student-Projects

Microsoft

Backup

Distributed Runtime



Distributed Runtime

Messaging is multiplexed over a small number of TCP connections

Actor directory is a custom DHT

Single-threaded execution on a small number of threads, one per core

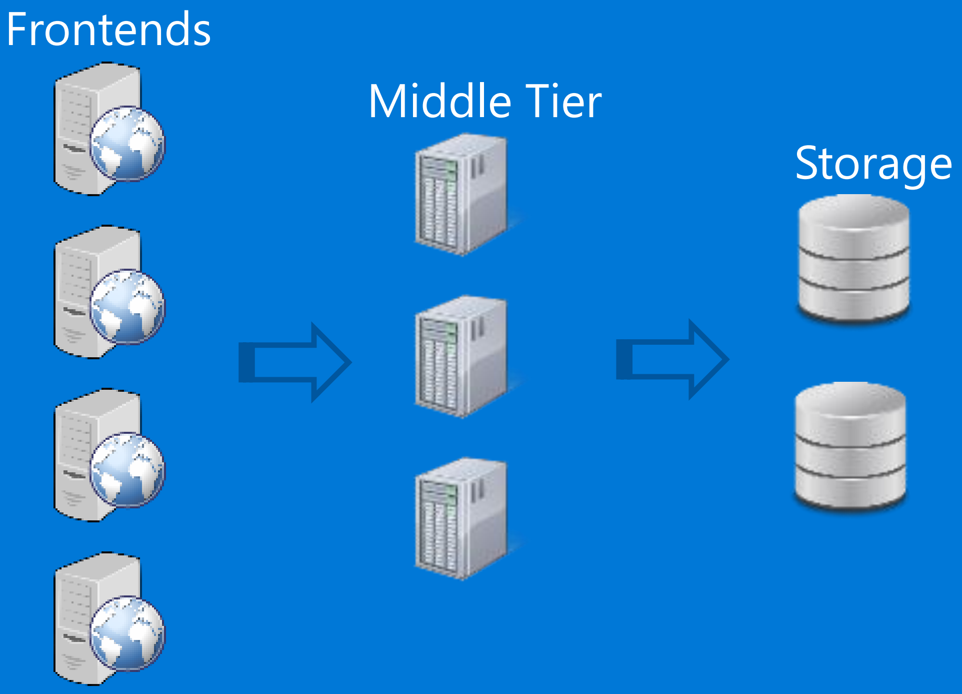
Performance benefits from cooperative multitasking

Actor activation management

- Automatic instantiation and placement (default is random)
- Garbage collection of idle activations

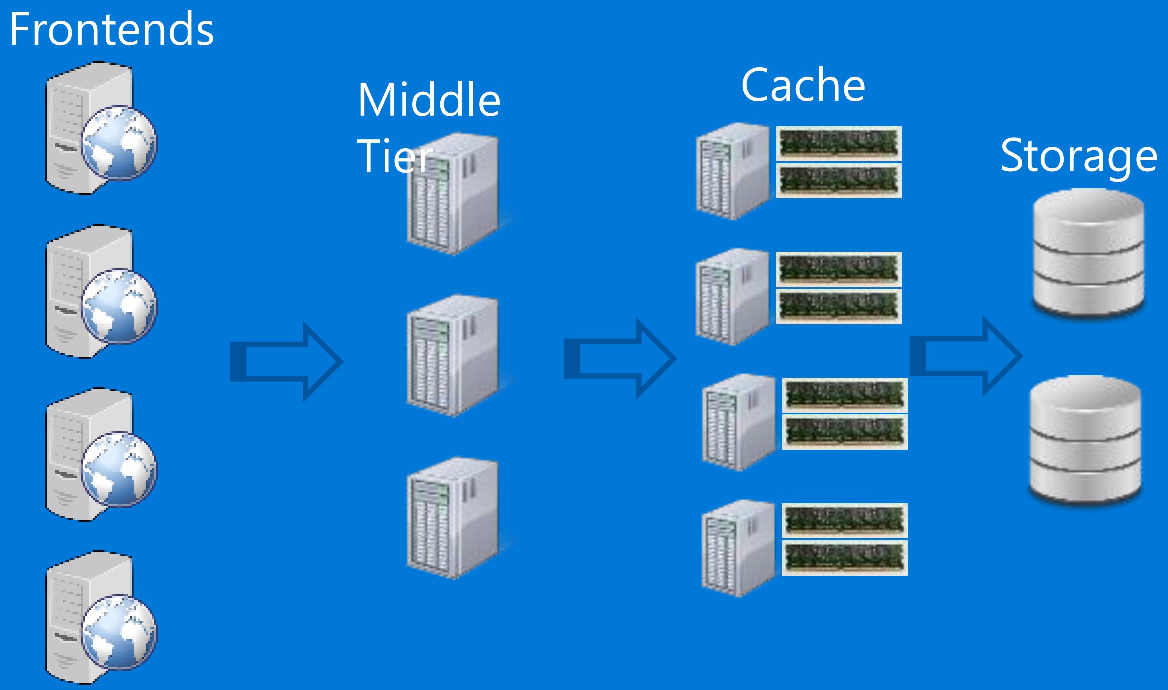
Custom cluster membership protocol, no Paxos

3-Tier Architecture



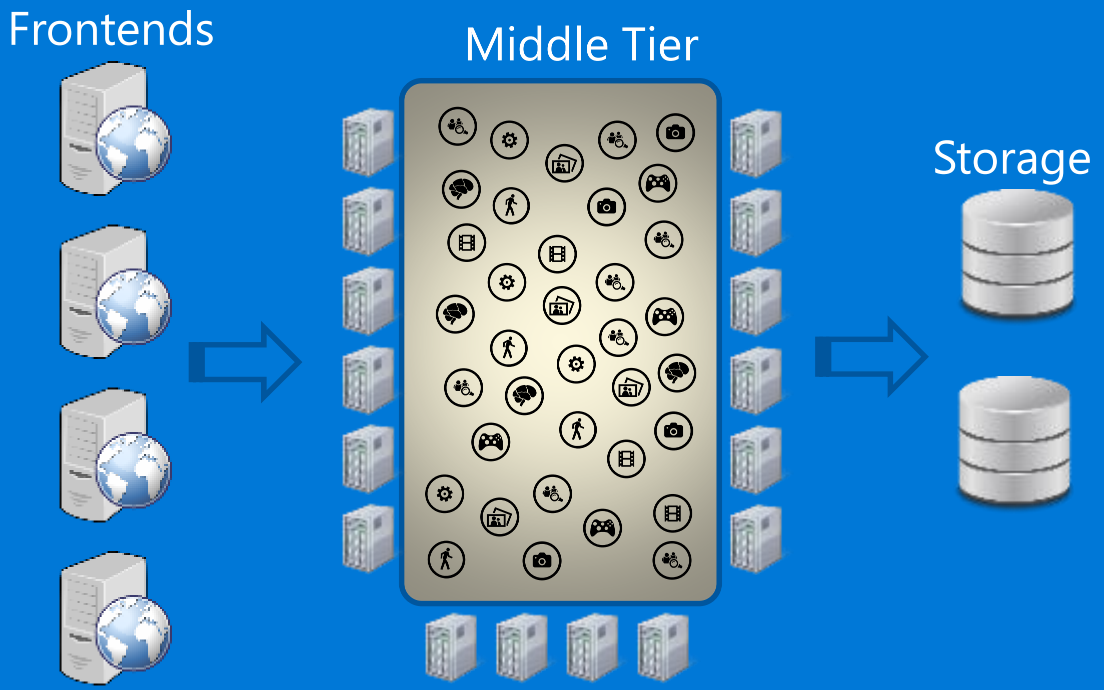
- Stateless frontends
- Stateless middle tier
- Storage is the bottleneck
  - Latency
  - Throughput
  - Scalability
- Horizontal calls are problematic
- Data shipping

Cache Tier for Performance & Scalability



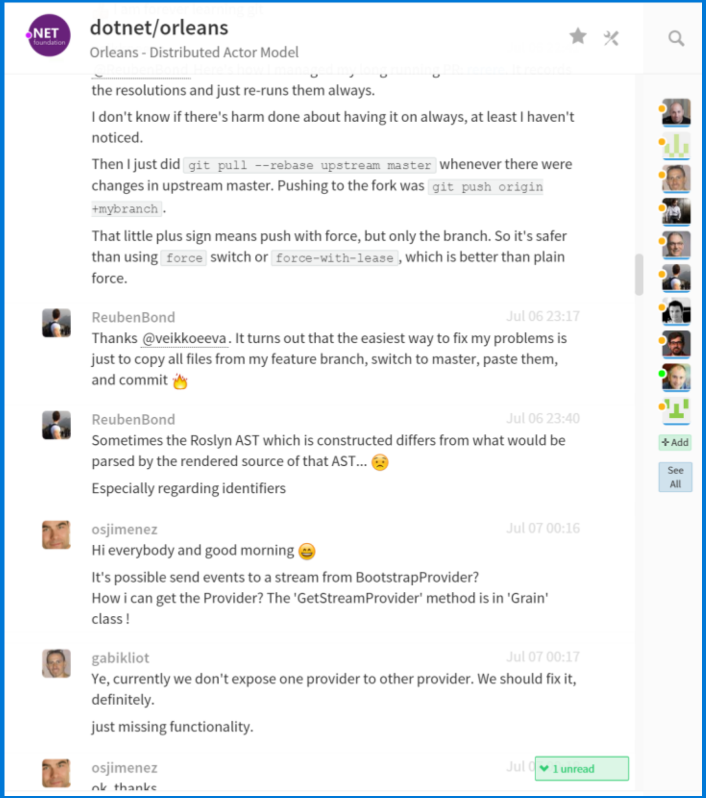
- Much better performance
- Lost semantics of storage
- Lost concurrency control
- Horizontal calls are still problematic
- Still data shipping

Actors as Stateful Middle Tier



- Performance of cache
- Rich semantics
- Concurrency control
- Horizontal calls are natural
- OOP paradigm regained
- Function shipping
- But there are still problems...

Community That Never Sleeps



US
UK
Australia
Finland
Ukraine
Hungary,
Netherlands
…

Microsoft