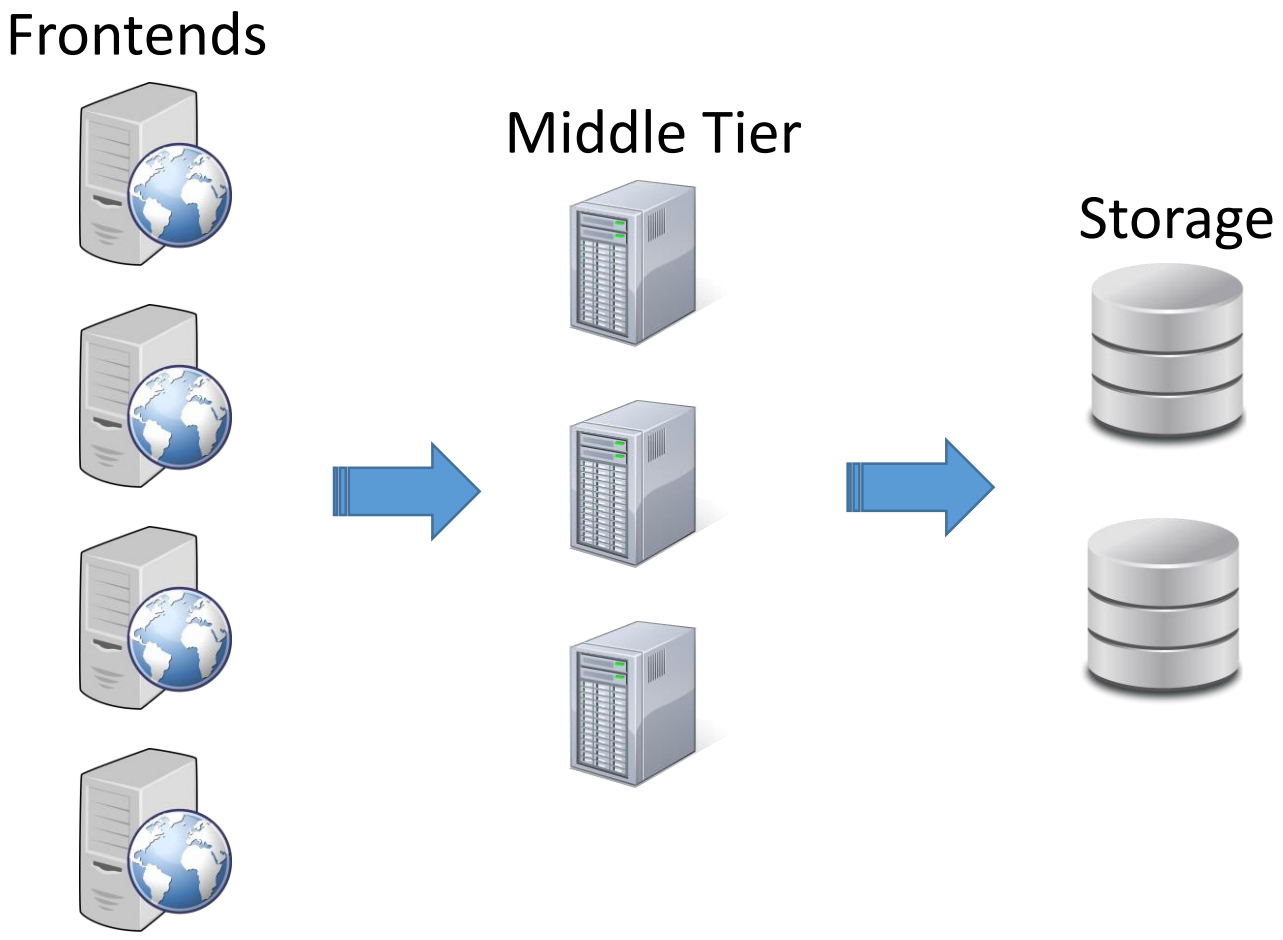
Orleans

Actors for High-Scale Services

Sergey Bykov

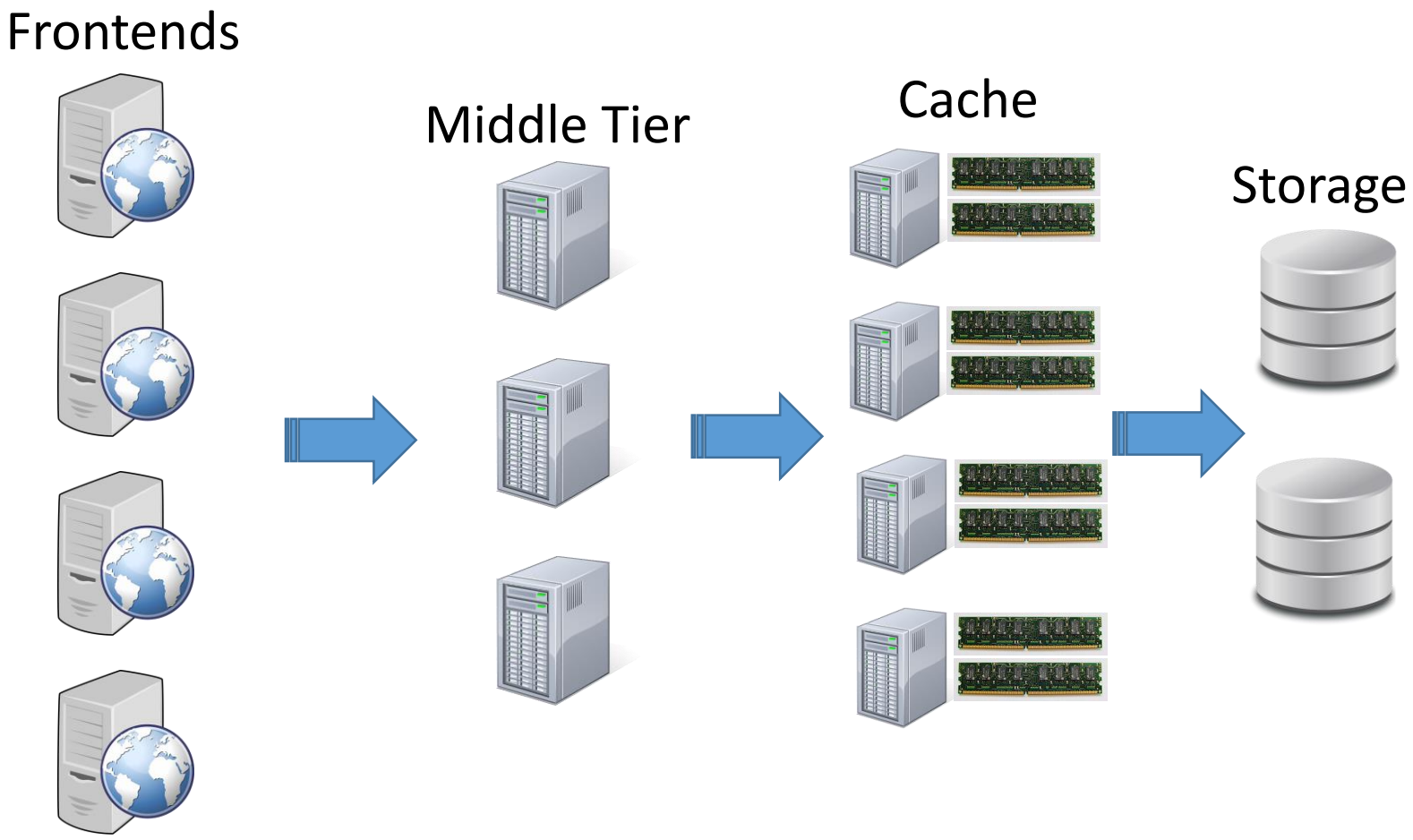
eXtreme Computing Group, Microsoft Research

3-Tier Architecture



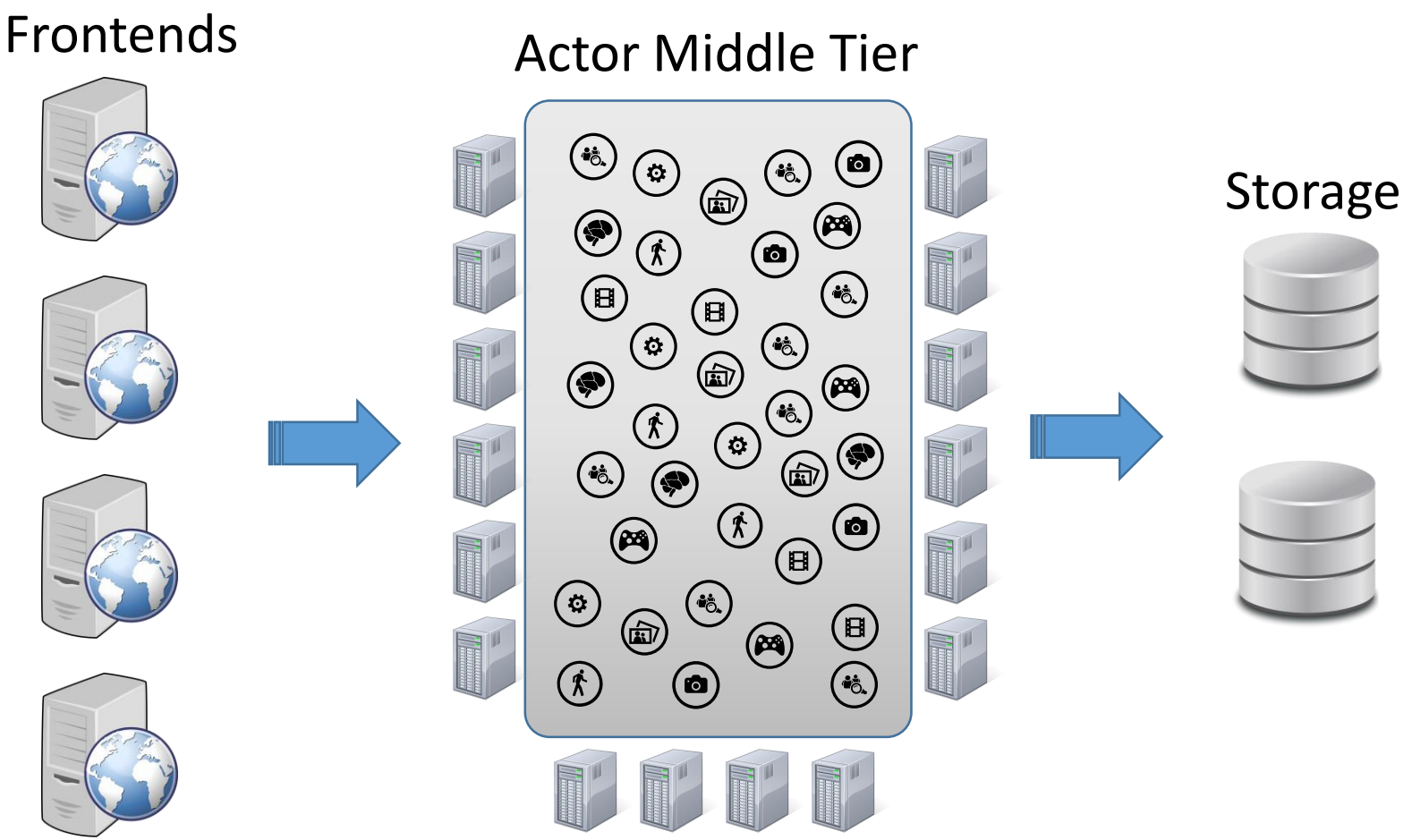
- **Stateless frontends**
- **Stateless middle tier**
- **Storage is the bottleneck**
  - **Latency**
  - **Throughput**
  - **Scalability**
- **Horizontal calls are problematic**
- **Data shipping**

Cache Tier for Performance & Scalability



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

Actor Model 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…**

Problems with Actor Model Frameworks

- Too low level
  - App manages lifecycle of actors, exposed to distributed races
  - App has to deal with actor failures, supervision trees
  - App manages placement of actors – resource management
- *Developer has to be a distributed systems expert*

Orleans – Programming Model & Runtime

Two goals:

- Qualitatively simplify distributed programming
- Scalable by default

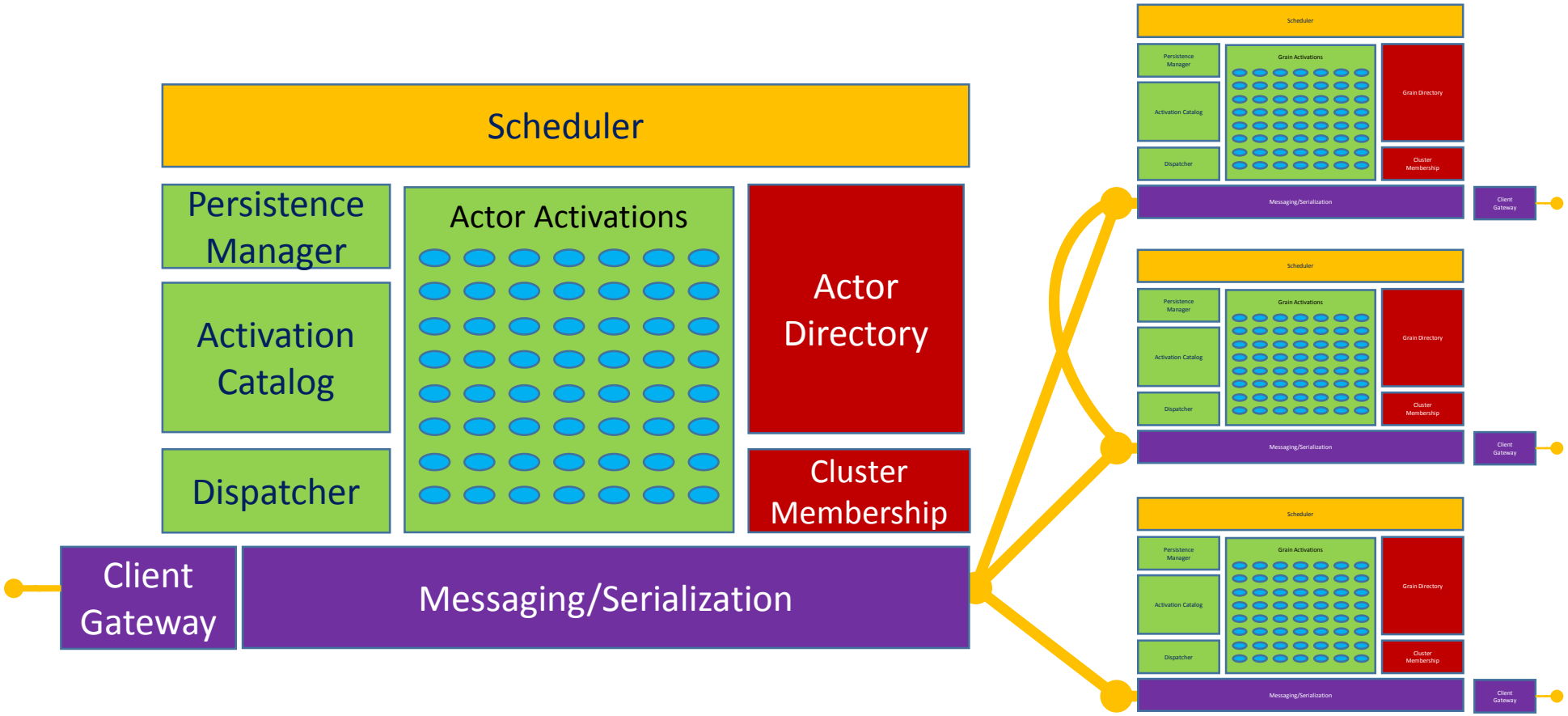
Key decisions:

- Built for .NET, written in C#
- Virtual actors
- Single-threaded event-based execution, using .NET async/await
- Automatic propagation of errors
- Automatic resource management
- Built-in support for persistence

Virtual Actors – Four Defining Features

1. *Virtual actors always exist, virtually*
   - Cannot be created, looked up or deleted
   - One can always make a call to an actor, using its type and identity
2. *Virtual actors are automatically instantiated*
   - If there is no in-memory instance, a message sent to it triggers instantiation
   - Transparent recovery from server failures
3. *Location transparency*
4. *Runtime can create multiple instances of an actor*
   - Implemented for stateless actors, prototyped for primary-copy replication

Distributed Runtime



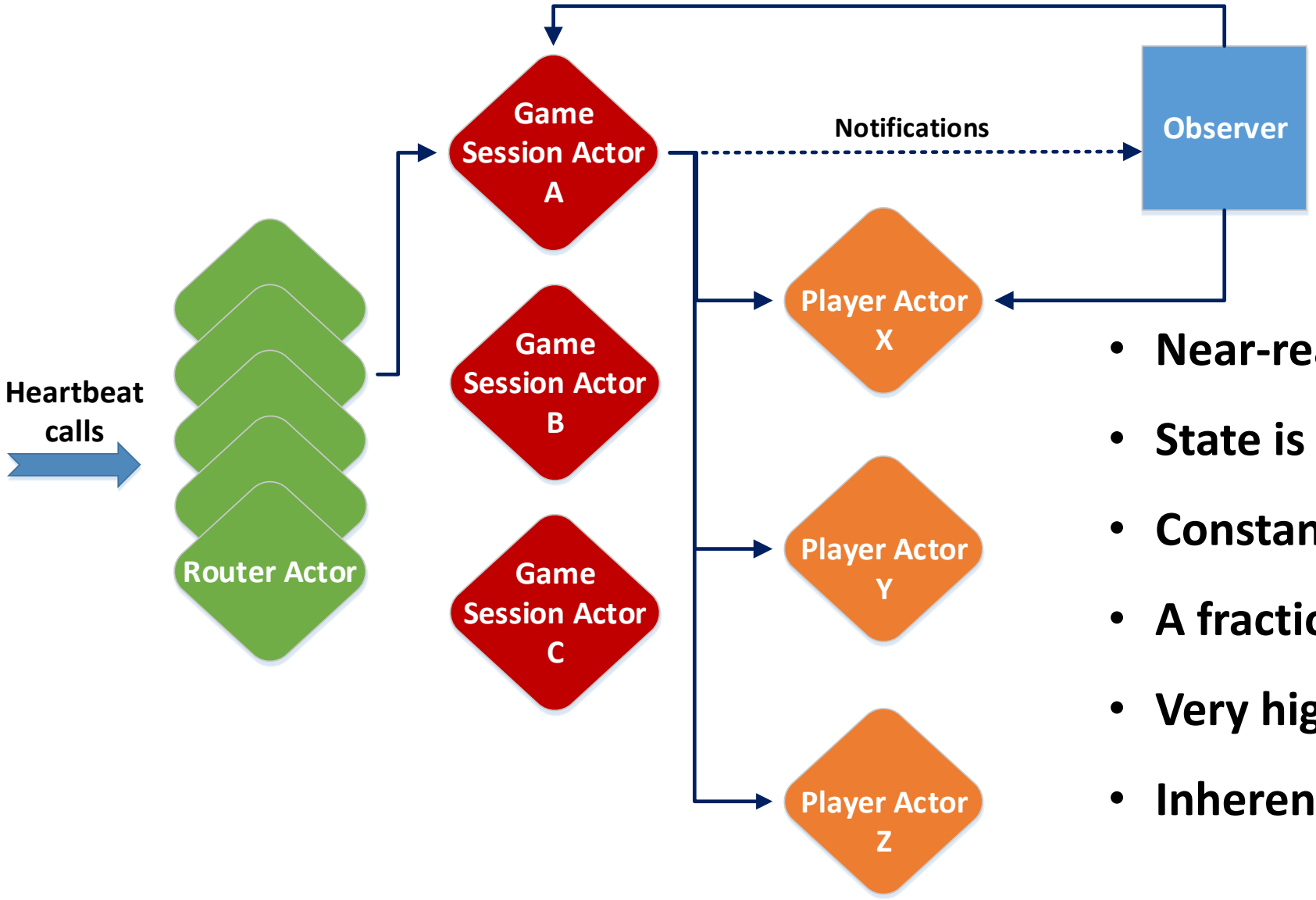
Distributed Runtime

- 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

Multiplayer Gaming – Unexpected Customer

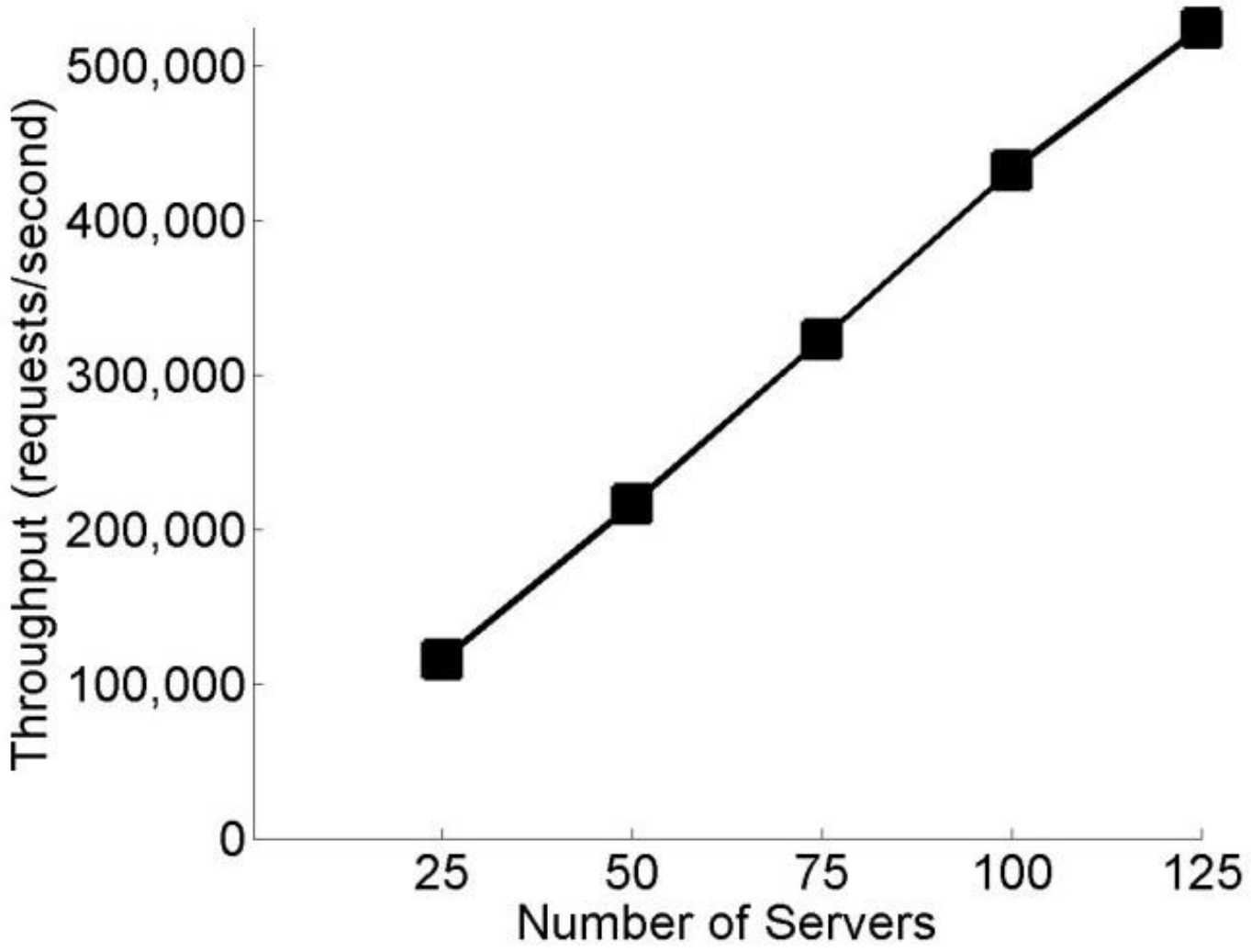
- Multiplayer gaming is a challenging problem
- Large scale fast-evolving social graph
- ‘Inverse’ scale demand
- Very demanding users – availability, performance
- Fast-pace development with fixed deadline

Halo Presence Service

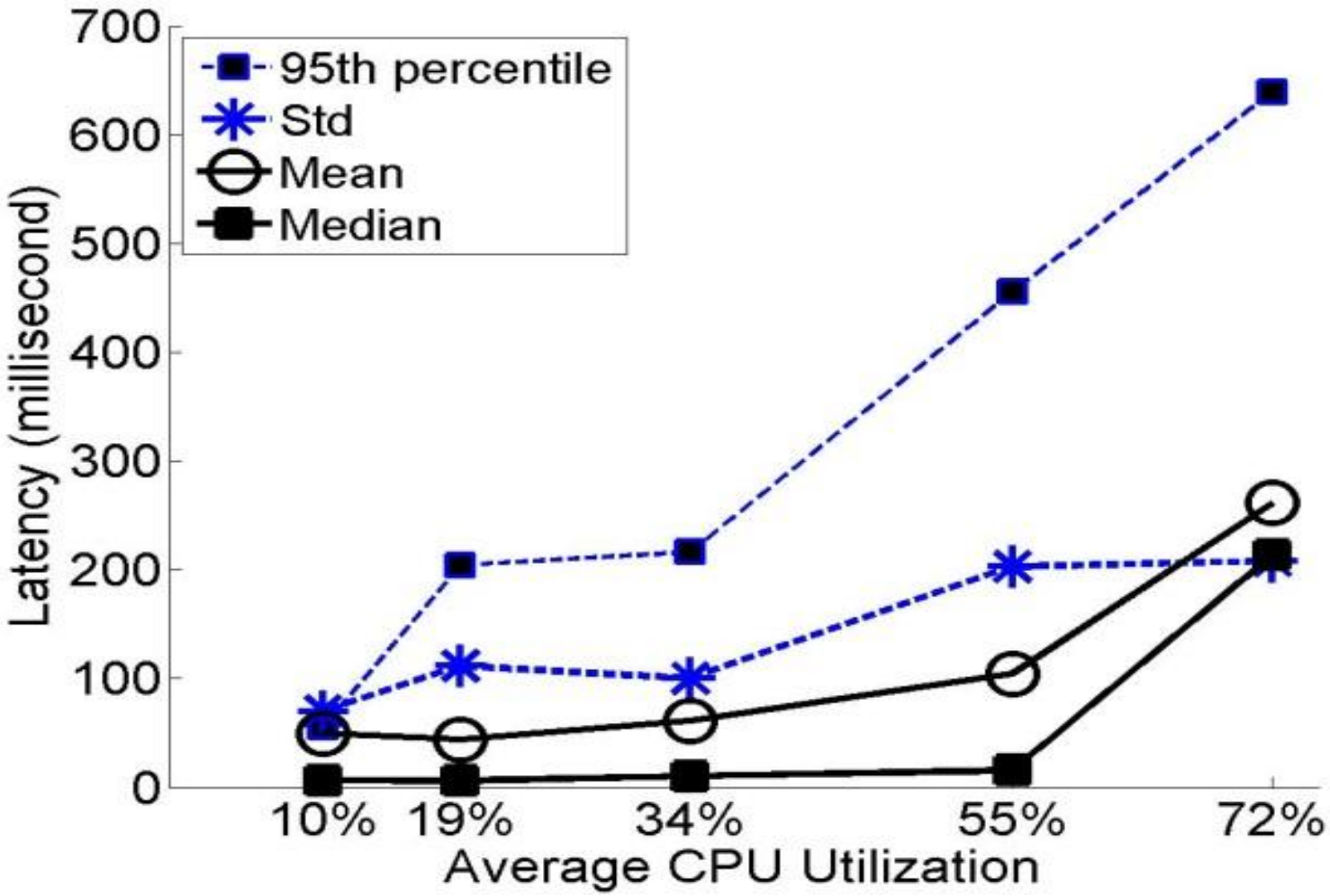


- Near-real-time processing
- State is mostly in memory
- Constantly evolving social graph
- A fraction of total user base online
- Very high throughput, low latency
- Inherent races

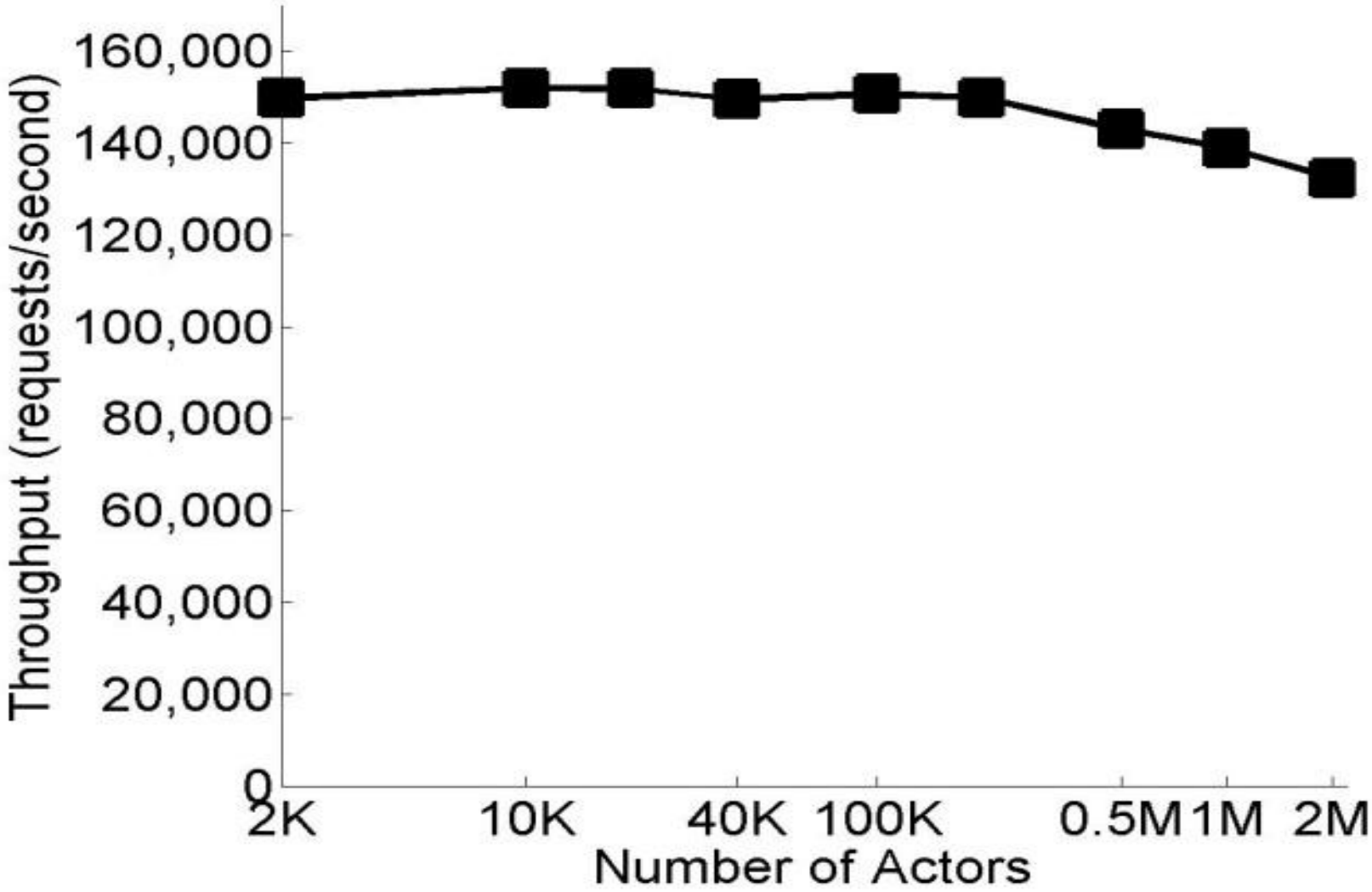
Scalability – Halo 4 Presence



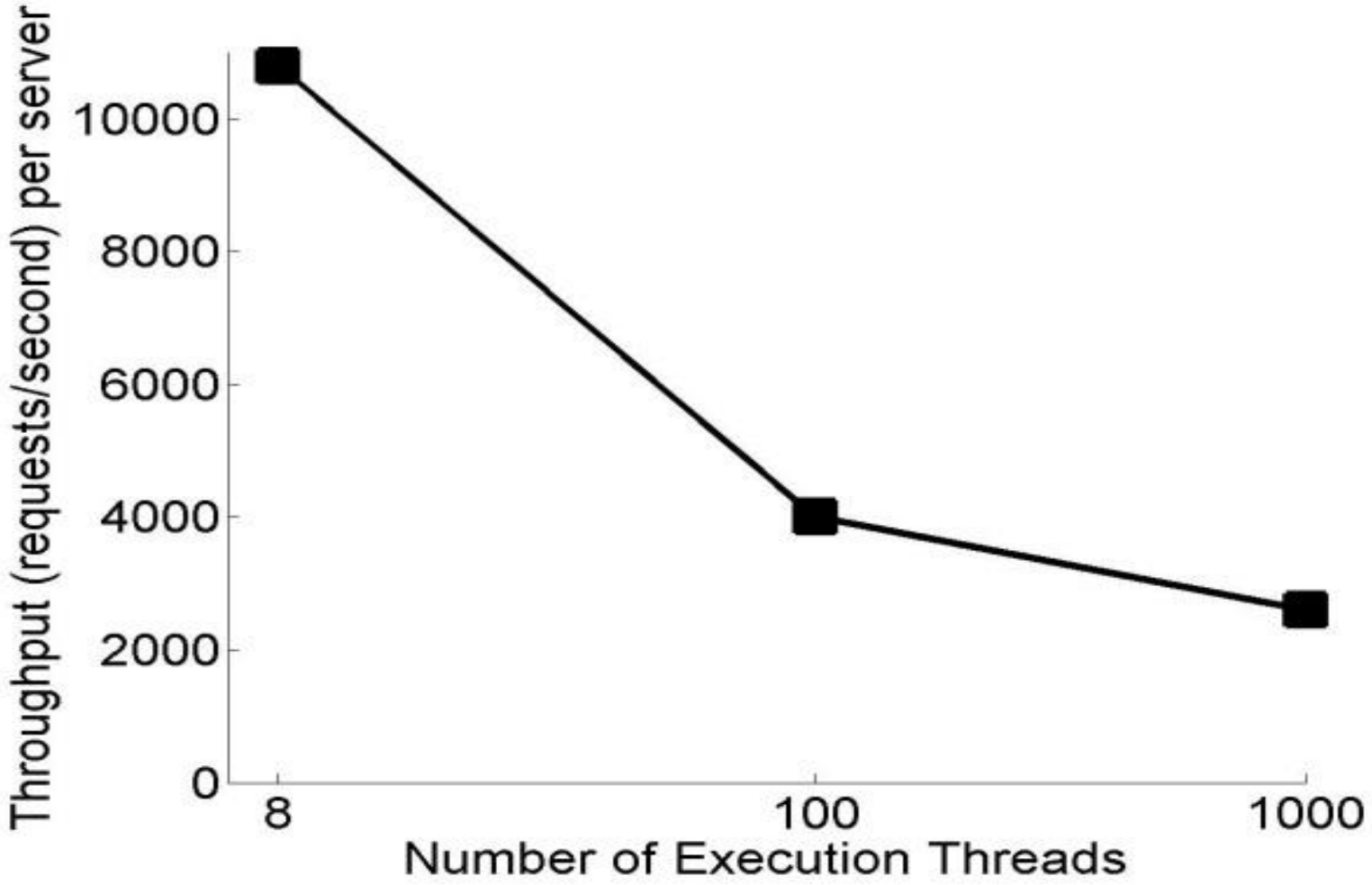
Latency as Function of Load – Halo 4 Presence



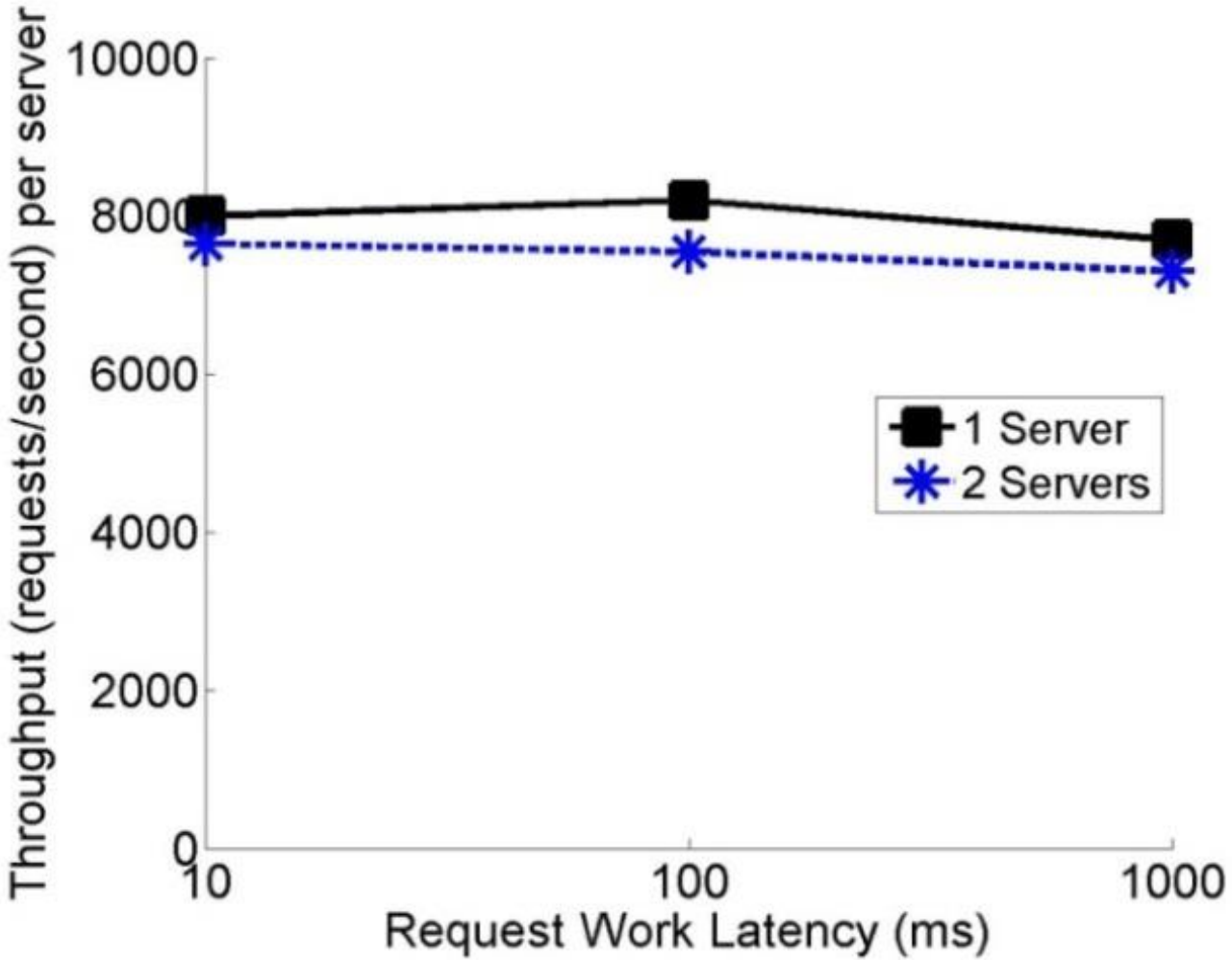
Throughput as Function of Number of Actors



Cooperative Multitasking



Throughput as Function of Latency



Summary

- Interactive services necessitate stateful middle-tier
- Actor model is a good fit for a wide variety of scenarios
- Virtual actor is a powerful concept
- Orleans:
  - Makes cloud-scale programming attainable to desktop developers
  - Uncompromised performance
  - Scalability by default
  - Proven in production by 1st-party services, notably all of Halo 4

Questions?