BB-601

Next generation application design

IoT Message Gateway

IoT Message Gateway: Requirements

- Communication gateway between IoT Device and the Backend applications
- Should abstract Device IoT messaging protocol from the Backend applications
- Expose REST APIs with JSON data format for consumption by Backend services
- Should be extensible to other messaging protocols with IoTDevice
- Support various protocols like BoxTalk, Http (REST) and **MQTT** (next)
- Should support (near) realtime APIs, websocket, webhooks (in future)
- Should be highly scalable and sustain high throughput (100s of thousands of concurrent users/toon display at peak)
- Horizontal scalability - to scale beyond one machine with hassle free clustering
- Simple and easy to implement (use of existing technologies, frameworks with minimal additions. should be based on Java11)
- Observability (monitoring - to know what is going on underneath the application)
- Metrics (performance metrics for the DEV team)
- Tracing (end-to-end tracing of messages or actions)

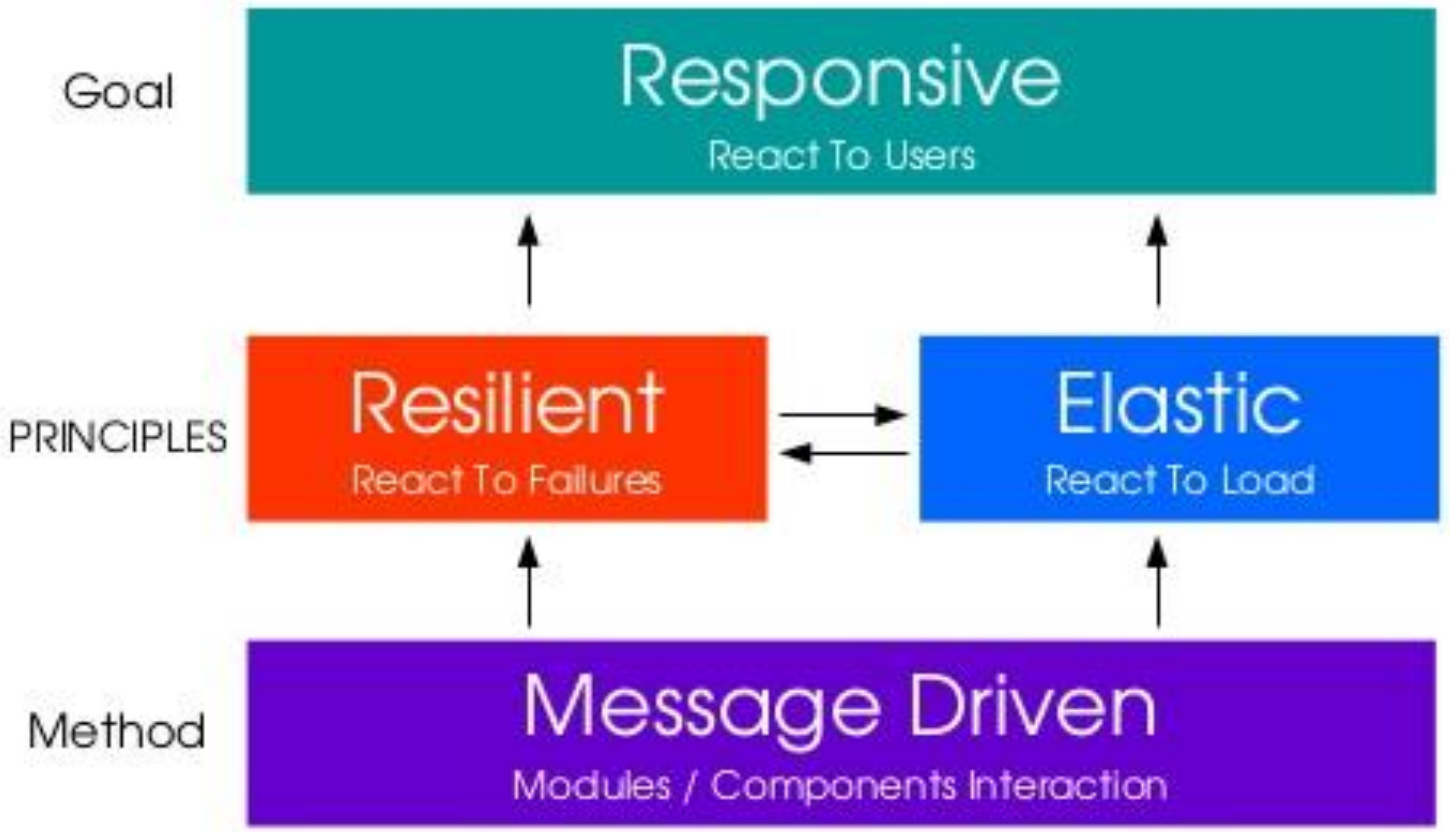
Existing Architecture: (Nothing wrong with it…)

- Heavily based on Servlet technology
- Single thread per request model
- Container managed threads (for eg tomcat.)
- Working with thread is hard (managing, error handling etc)
- Threads are very expensive (implements OS thread, CPU usages, context switching, memory etc)
- CPU, RAM are expensive in cloud.
- Easy to trace and debug
- Good performance at moderate load
- Not suitable for highly concurrent load for modern apps

Next Gen Architecture:

- Reactive architecture
- Reactive Manifesto (https://www.reactivemanifesto.org/)
  - **Responsive** - Responds in a timely manner
  - **Resilient** - The system stays responsive in the face of failure
  - **Elastic** - The system stays responsive under various workload
  - **Message driven** - Reactive systems rely on async message passing
- Reactive systems are more flexible, loosely coupled and scalable.

Reactive Manifesto



Responsive

- Means respond to users in a timely manner

As far as they (users) know, when the response time exceeds their expectation, the system is down. A slow response is a lot worse than no response.

- Excerpt from the book **Release it. (on my desk)**

- A responsive system depends on Resilient and Elastic

Resilient

- Means that the system stays responsive in the face of failure

A resilient system keeps processing transactions, even when there are transient impulses, persistent stresses, or component failures disrupting normal processing. This is what most people mean when they just say stability.

- Excerpt from the book **Release it. (on my desk)**

- So design applications for resiliency (circuit breaker, bulk heading, supervisor etc)

Elastic (Scalability)

- Elasticity is about resources. It is a constraint
  - CPU cores, Memory etc
- Scale up for responding to users
- Scale down to reduce costs

- An elastic system should be able to allocate / deallocate resources dynamically to match the demands

Message Driven

- Means that the components in the system should rely on message passing to establish Isolation and Abstraction over the resources and state of the system.

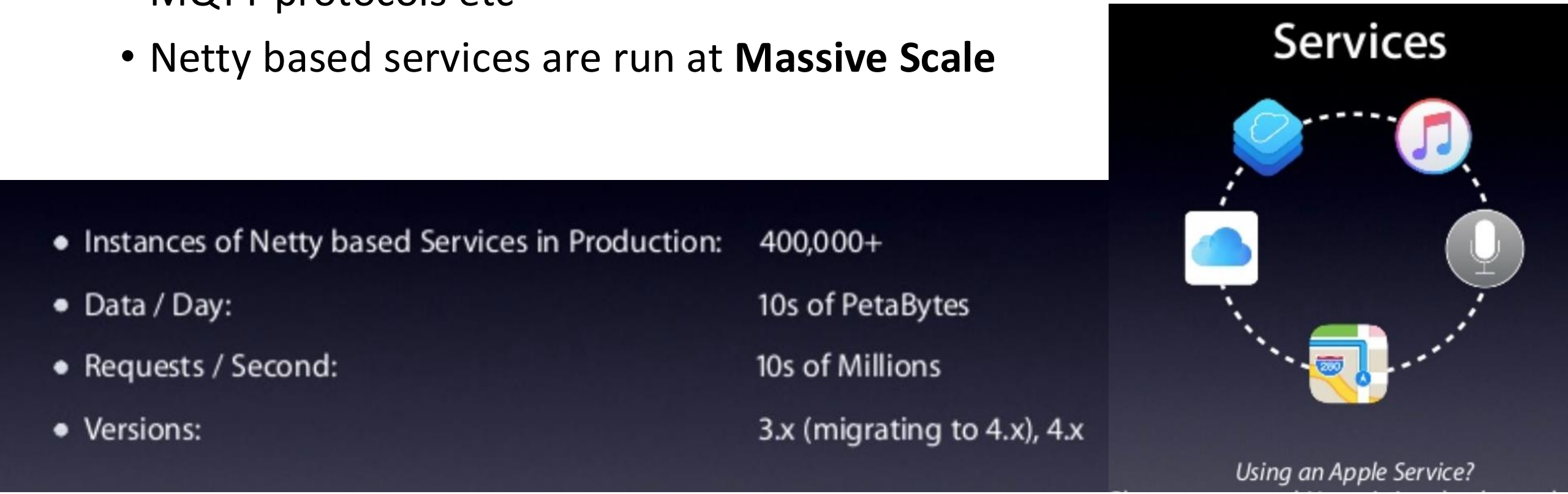
- Enables load management, elasticity, flow control, monitoring of message queues and back pressure.

Tools to build reactive systems

- SpringBoot 2 (NextGen IoC, HTTP framework, based on Netty and ProjectReactor)
- Akka toolkit (Actor based concurrency toolkit)
- Akka persistence (ES and CQRS)
- Immutables library (immutable messages)
- Lettuce.io for redis client library
- Kamon monitoring toolkit (metrics, tracing)

Why Netty

- Netty is a non-blocking, asynchronous, event driven network library.
- You can create high performance servers\clients based on HTTP(s), MQTT protocols etc
- Netty based services are run at **Massive Scale**



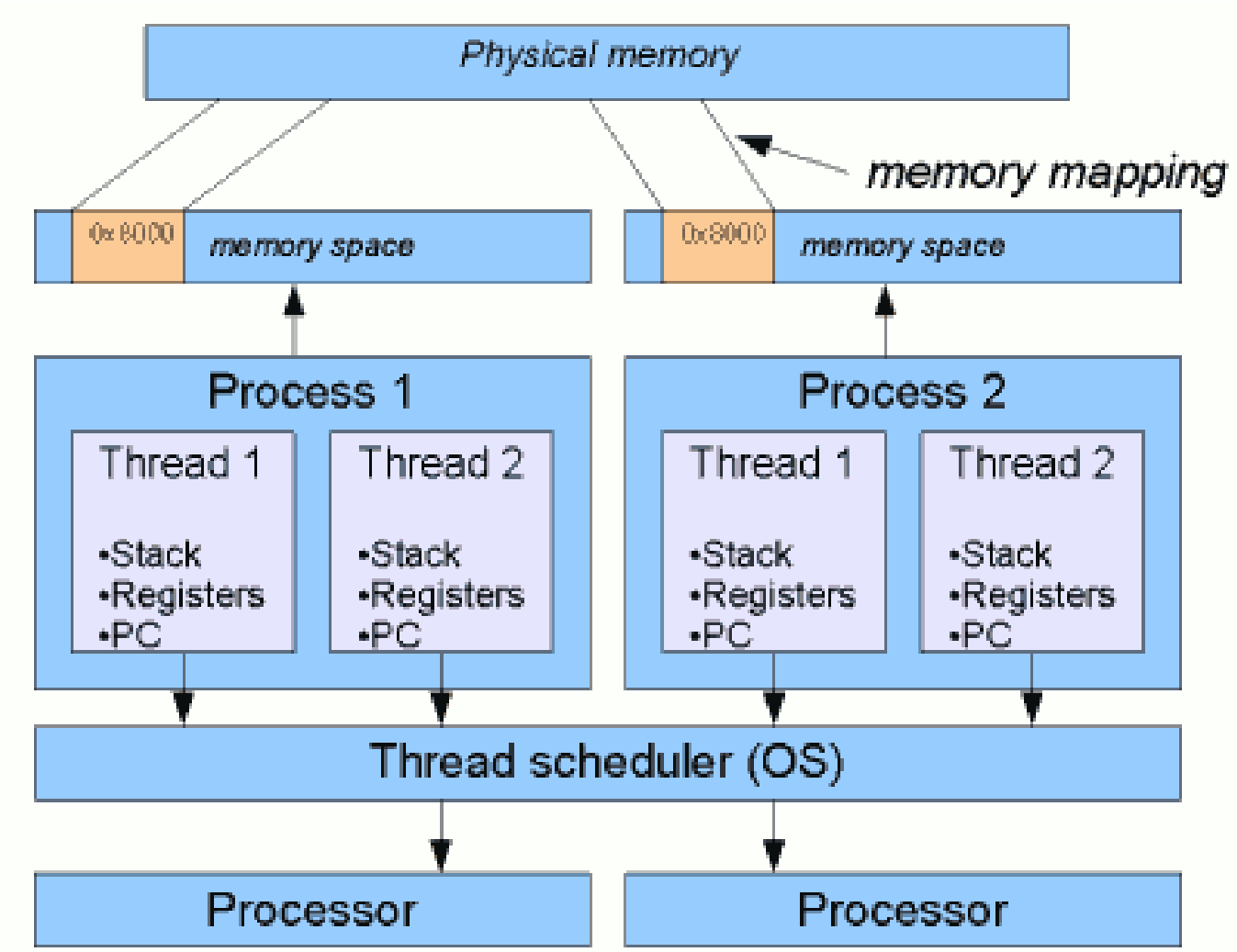
Akka

- Akka is a toolkit for building highly concurrent, distributed, and resilient message-driven applications for Java and Scala
- Akka is the implementation of the Actor Model on the JVM.
- Follows the principles of Reactive Manifesto for building reactive systems
- Resilient by design (self heal, stay responsive)
- High performance (50 million messages/per second, 2.5 million actors per GB of heap)
- Elastic and Distributed (no SPOF)

Why Akka: Actor Model

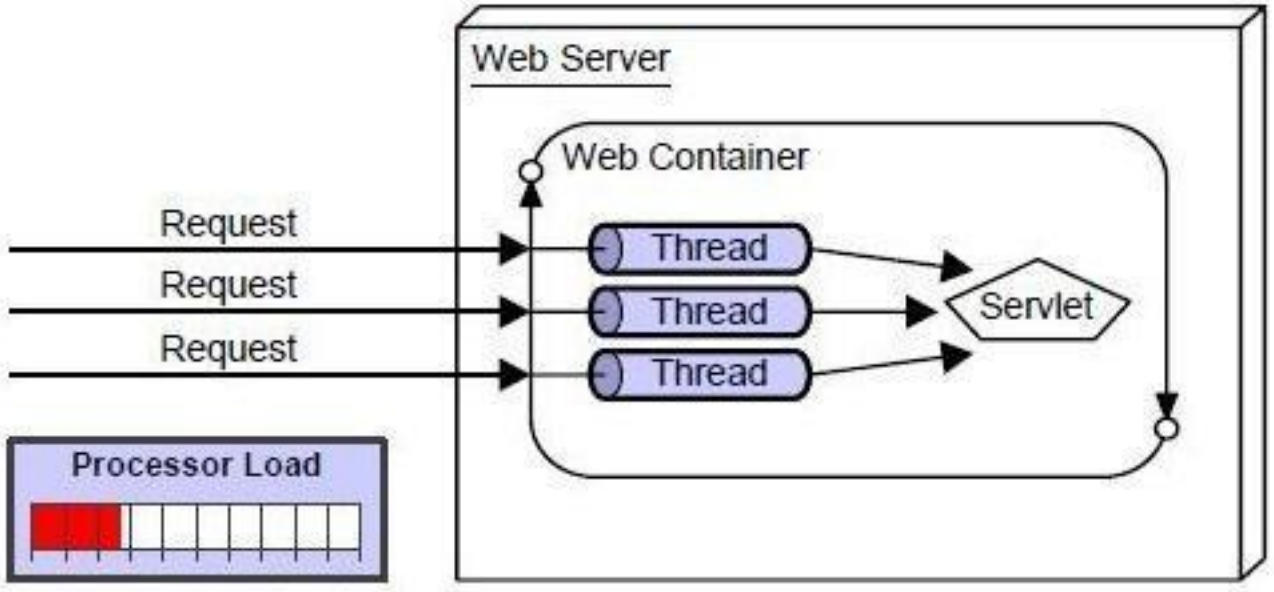
- Concurrent computation model that treats “actors” as the primitives (in current servlet tech, a thread is considered as primitives)
- Actors communicate via messages
- In response to message received, the actor can make
    - Local decisions
    - Create more actors
    - Send more messages
    - Determine its response
    - Update\change its private state

Why Akka: How Threads Work

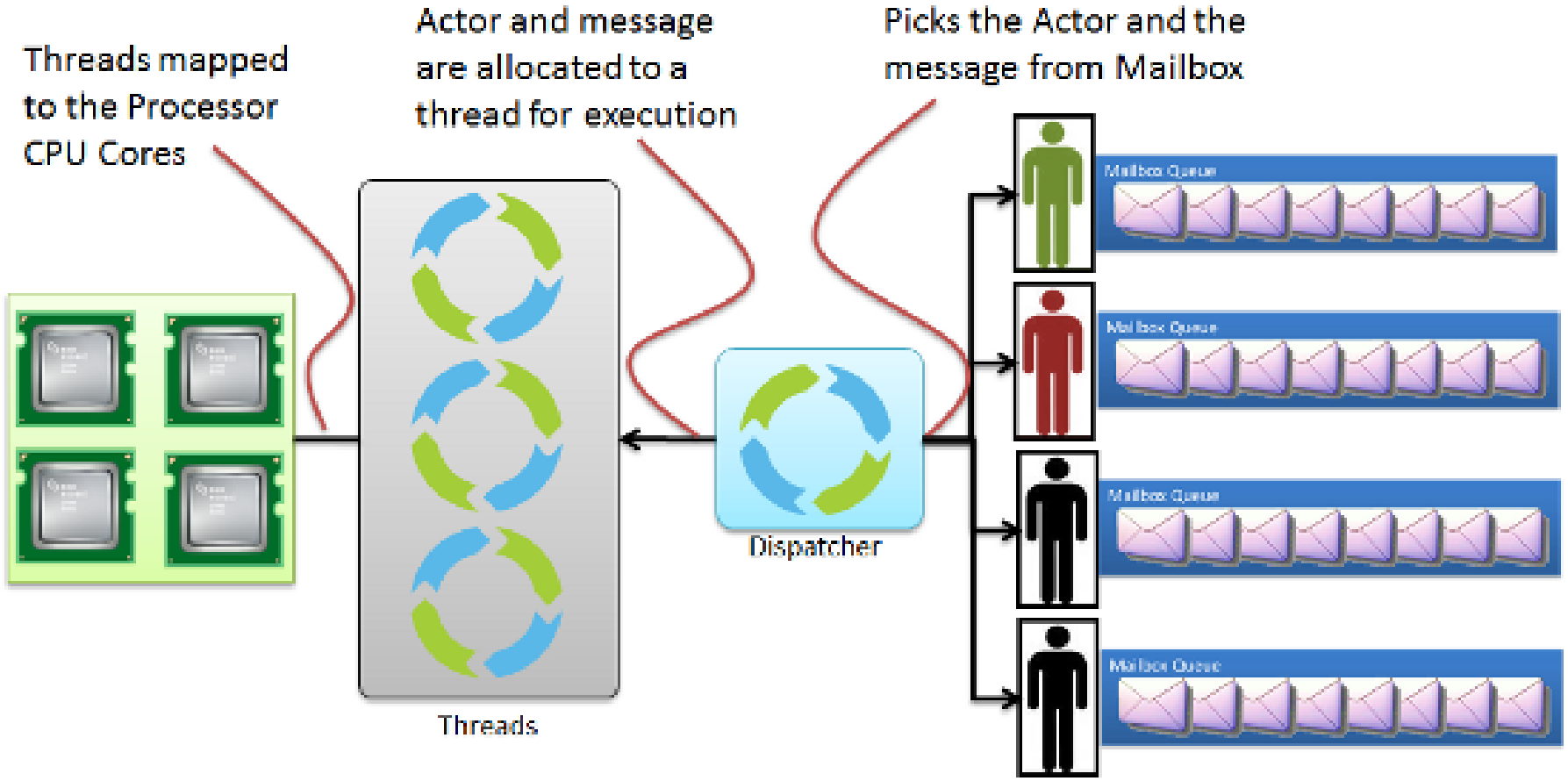


Why Akka: Current system

- Servlet model (tomcat server)

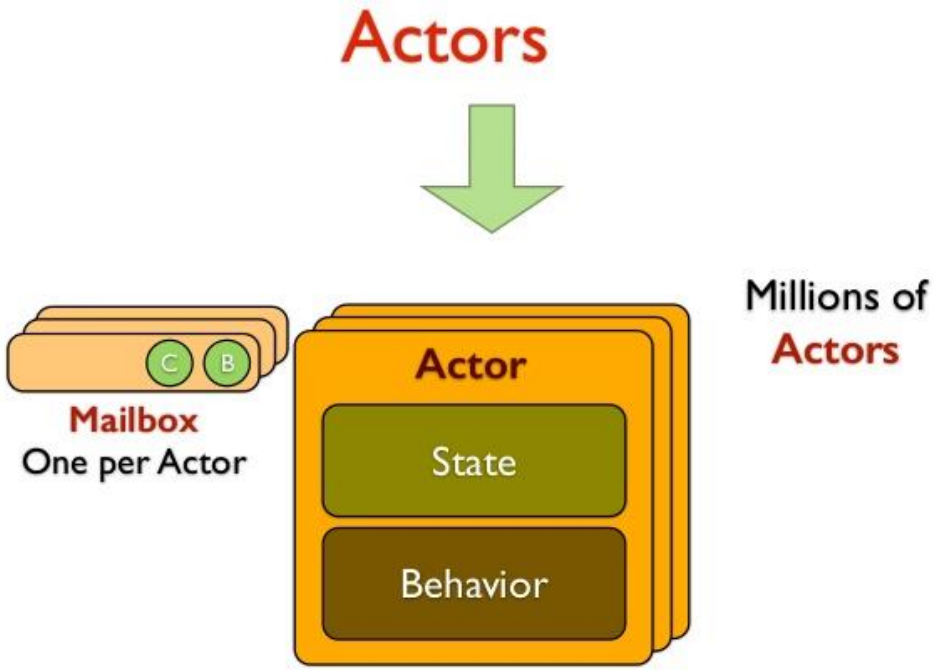


Why Akka: Akka system

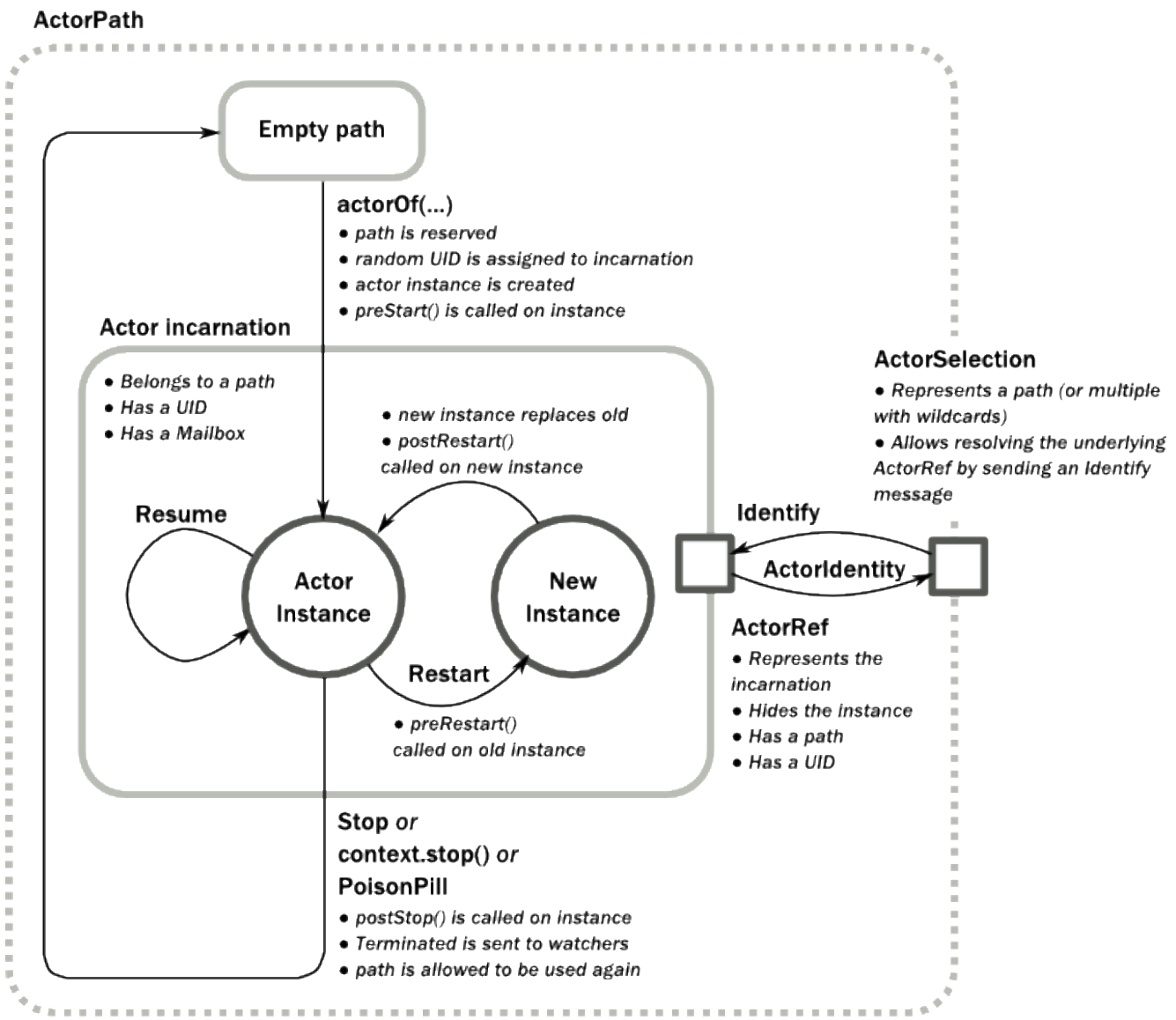


Why Akka: Actors vs Threads

- Akka actor
  - State (Storage)
  - Behavior (Processing)
  - Mailbox (Message Queue)
  - Light weight (300 bytes –MBs for Thread)
  - Keeps internal state
  - Async and non-blocking
  - Small call stack
  - Scalable and fast



Akka : Lifecycle



Session

- Actor system
- Actor lifecycle
- Actor creation/lookup/sending messages
- Clustering/Sharding
- Persistence
- Fault tolerance

Sequence Diagram