Executive Summary.

- *It is important for a Java Programmer to understand the power and limitations of concurrent programming in Java using threads.*
- *Poor co-ordination that exists in threads on JVM is bottleneck and various methods to avoid such fallacies have been developed.*
- *Scala is a language that is presented in my presentation shows some of the major advantages in using Actor based Scala for concurrent programming in Java.*
- *The rich features of Scala and its interoperability with Java using scala library make this highly strong candidate in designing concurrent programs using Actor based Scala.*
  - *The actor model allows to write complex parallel and distributed programs in a very simple way*
  - *The actor model is said to be "The Solution" to exploit the multi-core architectures.*
  - *Communication model in Scala using synchronous, asynchronous communication and collective operations remains the best choice for computational intense tasks. Actors could be used for fast prototyping, or in environments (like Web Services) I/O intense.*
- *In this presentation I have depicted how Scala can be used on a popular social networking site like "Twitter" and show its use and efficiency.*
- *Modern hardware is equipped with multi-core and to exploit full efficiency of hardware in software we need to have a safe, high performing concurrent language like Scala.*
- *I present some of the features of Scala, Evaluation and advantages that gives the reader a clear understanding of why there is a need for this safe concurrent programming language.*

Concurrency In Java And Actor-Based Concurrency Using SCALA.

by Supriya Meka.

CSCI-5448 Spring 2011.

Prof. Kenneth Anderson.

The University Of Colorado, Boulder.

Presentation Overview.

1. Introduction to Concurrency.
2. Java Concurrency using Threads and Scala.
3. Problems using Threads.
4. Newer Generation Of Concurrency.
5. Introduction to Scala.
6. Scala Actor Model.
7. Overview of Actor Model.
8. Sample Code in Java with Scala.
9. Asynchronous Message Passing.
10. Communication Model in Scala.
11. Scala Actor Properties.
12. Programming Abstractions.
13. Scala and Twitter.
14. Performance Comparisons.
15. Conclusion.

Introduction:

- Applications are becoming increasingly concurrent, yet the traditional way of doing this by threads and locks, is very troublesome as we will see in the "Problems with Threads".

- Designing Objects for Concurrency in Java.
  - Parallel programming deals with Tightly coupled, fine-grained multiprocessors to achieve true concurrency.
  - Main application in Large scientific and engineering problems.

- Divide and Conquer Method:
  - For a large problem, we want to have at least as many threads as the number of available cores.

Java Concurrency and SCALA

- The traditional way of offering concurrency in a programming language is by using threads. True component systems have been an elusive goal of the software industry.

- Scala fuses object-oriented and functional programming in a statically typed programming language.
  - Scala is aimed at the construction of components and component systems.

- Scala, which stands for **Scalable Language** is a programming language that aims to provide both object-oriented and functions programming styles, while staying compatible with the Java Virtual Machine (JVM)

Problems using Threads

- Threading lead to a series of hard to debug problems as mentioned below:

1) Lost-Update problem:

Two processes try to increment the value of a shared object. It is possible that while execution gets interleaved, it leads to an incorrectly updated value of the shared resource due to not atomic nature of shared object.

2) Deadlock problem:

Using Locks if we try to avoid the above shared object problem then it leads to deadlock problem in which two processes try to acquire the same two locks, both wait on the other to release the lock, which will never happen.

Newer Generation Of Concurrency.

- "java.util.concurrent package"
- This package has nicely replaced the old threading API.
  - Wherever you use Thread class and its methods, you can now rely upon the ExecutorService and related classes.
  - Instead of using the synchronized construct, you can rely upon the Lock interface and its methods that give you better control over acquiring locks.
- Wherever you use wait/notify, you can now use synchronizers like CyclicBarrier and CountdownLatch .
  - Modern Java/JDK Concurrency helps in Scalability and Thread Safety

Scala Introduction : Actor Based

- The Actor Model, was first proposed by Carl Hewitt in 1973 and was improved, among others, by Gul Agha.
- This model takes a different approach to concurrency, which should avoid the problems caused by threading and locking.
- In the SCALA actor model, each object is an actor. This is an entity that has a mailbox and a behavior.

- Scala Stands for **Scala**ble Language

Scala Actor Model:

- Scala implements the actor model into a main-stream, Object-Oriented Language.

- Actors in Scala are available through the "**scala.actors library**".

- An actor is a computational entity and each actor runs concurrently with other actors: it can be seen as a small independently running process.
  - In this model all communications are performed asynchronously.
  - All communications happen by means of messages.

- Scala is Statically typed: includes a local type inference Systems.

Overview of the Actor Model

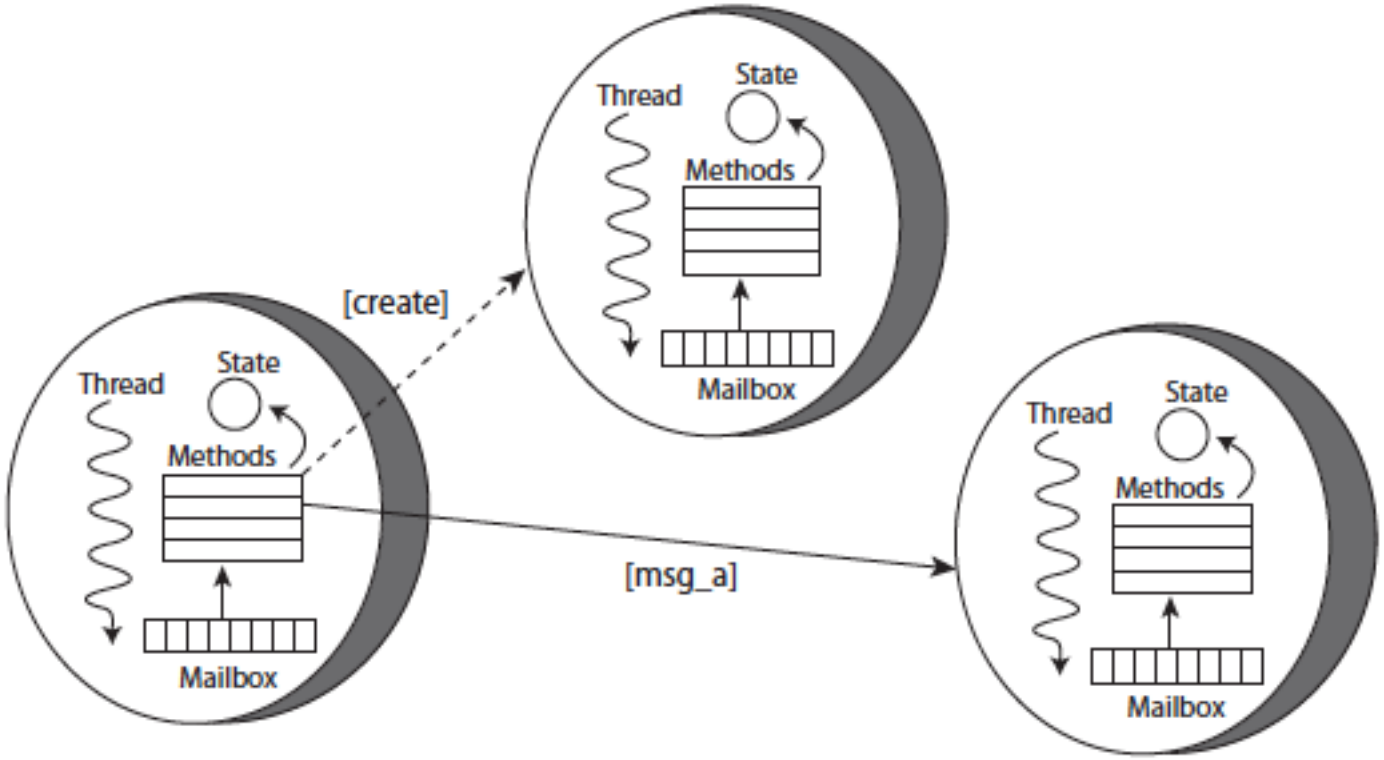

Figure 2: Actors are concurrent entities that exchange messages asynchronously.

Sample Code in Java with Scala

```
import scala.actors.Actor
import scala.actors.Actor._

case class Inc (amount: Int)
    case class Value

 class Counter extends Actor {
     var counter: Int = 0;
    def act() = {
       while (true) {
            receive {
               case Inc(amount) =>
                        counter += amount
              case Value =>
                        println("Value is "+counter)
                        exit()
                        }
            }
    }
}
//  contd..
```

```
// continuation…
object ActorTest extends Application {
        val counter = new Counter
    counter.start()
          for (i <- 0 until 100000) {
                counter ! Inc(1)
        }
    counter ! Value
     // Output: Value is 100000
}
```

Scala has a uniform object model, in the sense that every value is an object and every operation is a method call

Async Msg Passing n Share nothing.

- Encourages shared-nothing process abstractions.
  - mutable state – private
  - shared state – immutable
- Asynchronous message passing.
- Pattern matching.
- Thread-based vs. Event-based Actors.
  - Thread-based actors - each run in its own JVM thread
  - Overhead in case of large amount of actors
  - Full stack frame suspension (receive)
- Event-based actors run on the same thread.
  - Use a react block instead of a receive block

Communication Model in Scala

- Communications among actors occur asynchronously using message passing.
  - Recipients of messages (a.k.a. "mailboxes") are identified by addresses, sometimes called "mailing address".

- An actor can only communicate with actors whose addresses it has:
  - It can obtain those from a message it receives, or if the address is for an actor it just created. The actor model gives no guarantee about message ordering, and also buffering is not necessary.

- The strength of this model comes with the use of **mailboxes** which are free of race conditions by definition (messages cannot be read in a inconsistent state).

Scala Actor Properties

- Fairness in Scheduling.
  - A message is eventually delivered to its destination actor, unless the destination actor is permanently "disabled" (in an infinite loop or trying to do an illegal operation).
  - Another notion of fairness states that no actor can be permanently starved.
- State Encapsulation.
  - An actor cannot directly (i.e., in its own stack) access the internal state of another actor. An actor may affect the state of another actor only by sending the second actor a message.
- Safe Messaging.
  - There is no shared state between actors. Message passing should have call-by-value semantics. This may require making a copy of the message contents, even on shared memory platforms.

Scala Actor Properties – Contd.

- Location Transparency:
  - Location transparency provides an infrastructure for programmers so that they can program without worrying about the actual physical locations.
  - Because one actor does not know the address space of another actor, a desirable consequence of location transparency is state encapsulation.

- Scala Actors, an actor's name is a memory reference, respectively, to the object representation of the actors (Scala).

- Mobility:
  - Location transparent naming also facilitates runtime migration of actors to different nodes, or mobility. Migration can enable runtime optimizations for load-balancing and fault-tolerance.
  - Actors provide modularity of control and encapsulation, mobility is quite natural to the Actor model.

Actor-based Mutability

- Isolating Mutability using Actors:
  - Shared mutability: where multiple threads can modify a variable is the root of concurrency problems.
  - Isolated mutability: where only one thread (or actor) can access a mutable variable, ever - is a nice compromise that removes most concurrency concerns.
- While your application is multithreaded, the actors themselves are single light weight threaded.
- There is no visibility and race condition concerns.
- Actors request operations to be performed, but they don't reach over the mutable state managed by other actors.

Programming Abstractions

- Two useful programming abstractions for communication and synchronization in actor programs:
  - Request-Reply Messaging Pattern
  - Local Synchronization Constraints

Request-Reply Messaging Pattern

- In this pattern, the sender of a message blocks waiting for the reply to arrive before it can proceed. This RPC-like pattern is sometimes also called synchronous messaging. For example, an actor that requests a stock quote from a broker needs to wait for the quote to arrive before it can make a decision whether or not to buy the stock.

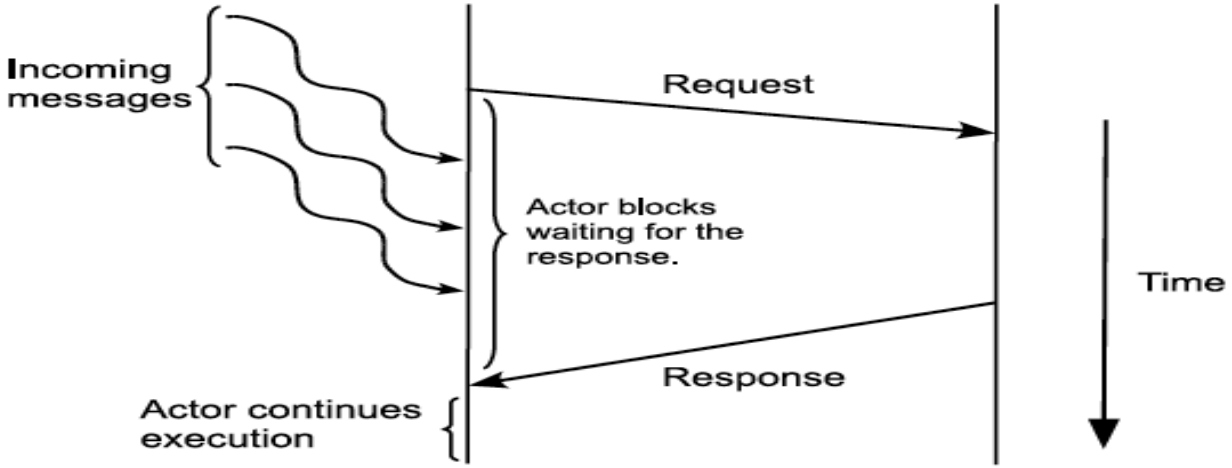

Figure 4 : Request-reply messaging pattern blocks the sender of a request until it receives the reply. All other incoming messages during this period are deferred for later processing.

- Request-reply messaging is almost universally supported in Actor languages and frameworks. It is available as a primitive in Scala Actors.

Scala Actors Local Synchronization

- Scala Actors library provides local synchronization constraints.
- Its support for constraints is based on pattern matching on incoming messages.
- Messages that do not match the receive pattern are postponed and may be matched by a different pattern later in execution.

- Local Synchronization Constraints
  - Each actor operates asynchronously and message passing is also subject to arbitrary communication delays; therefore, the order of messages processed by an actor is nondeterministic.
  - Sometimes an actor needs to process messages in a specific order. This requires that the order in which messages are processed is restricted.
  - Synchronization constraints simplify the task of programming such restrictions on the order in which messages are processed.

Scala Users:

Who is using Scala ?

Why Scala at Twitter

- A rich static type system that gets out of your way when you don't need it, is awesome when you do.
- Flexible syntax makes it easy to write readable, maintainable code.
- Traits for cross-cutting modularity.
- Lots of OOP goodness.
- Choice between immutable and mutable variables
- Powerful functional programming:
  - mapping, filtering, folding, currying, so much more.
- Lazy values.
- Pattern matching.
- XML literals, and clean syntax for working with XML documents built in.

JSON Twitter App in Scala.

- Scala Services at Twitter
  - **Kestrel: Queuing.**
  - **Flock(and Gizzard): social graph store.**
  - **Hawkwind: people search.**
  - **Hosebird: streaming API.**

```
val myJsonString = """
  {   "key": "value", "1": 2, "3": 4,     "myList":
  [5, 6, 7] """
Json.parse(myJsonString)
Json.build(List("user_id", 666))
```

Scala Concurrent Programming

- Scala Actors unify both programming models:
  - **Thread-based** or **Event-based**. Developers can trade efficiency for flexibility in a fine-grained way.

- **Thread-based implementation**: Each Actor is executed by a thread. The execution state is maintained by an associated thread stack.

- **Event-based implementation**: An Actor behavior is defined by a set of event handlers which are called from inside an event loop. The execution state of a concurrent process is maintained by an associated record of object.

Performance comparisons of Java Vs Scala

Using Scala shows a benefit in the performance even for simple applications.

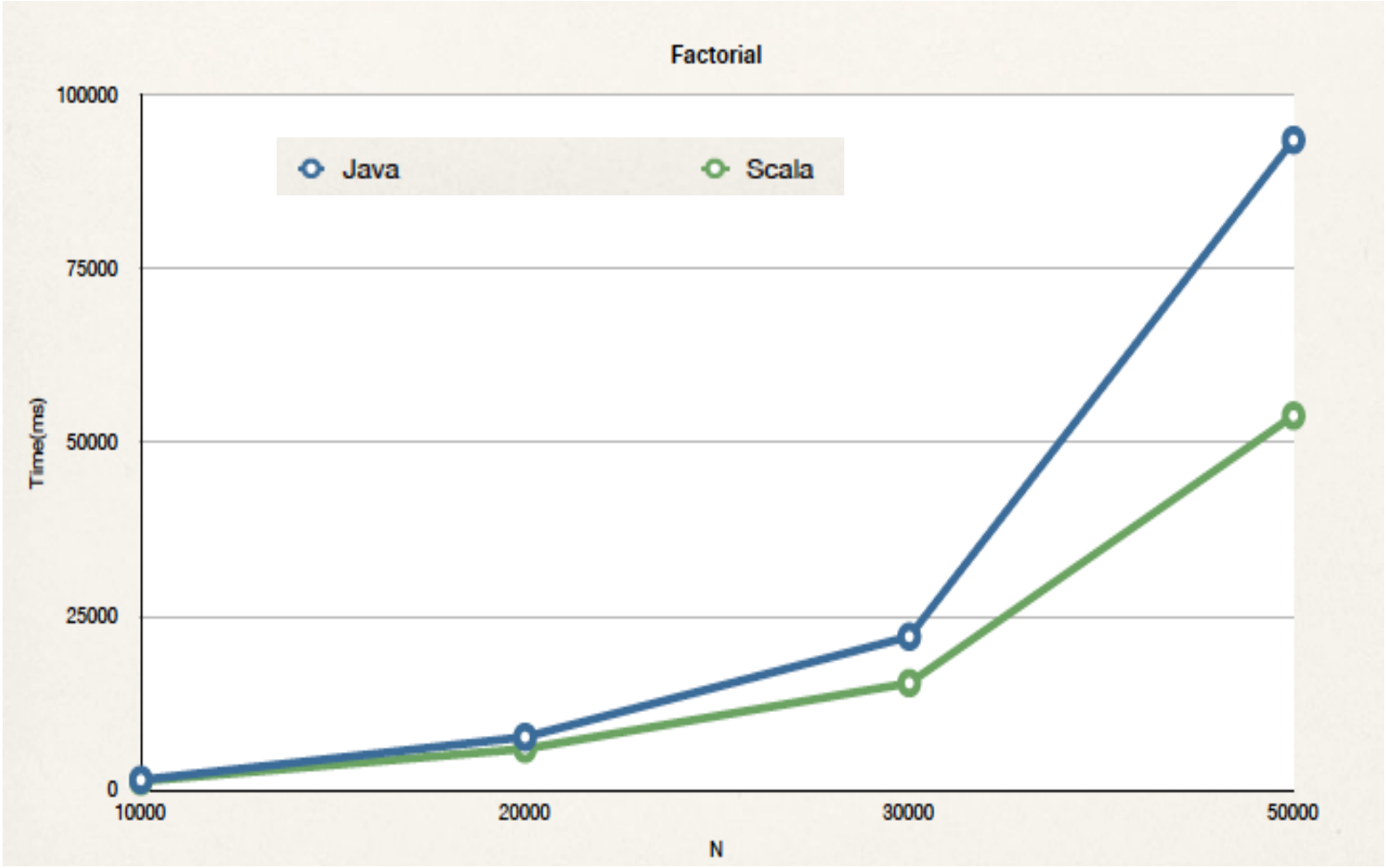

Figure 7: Speed up for Factorial in Java Vs Scala

Threading Performance Evaluation

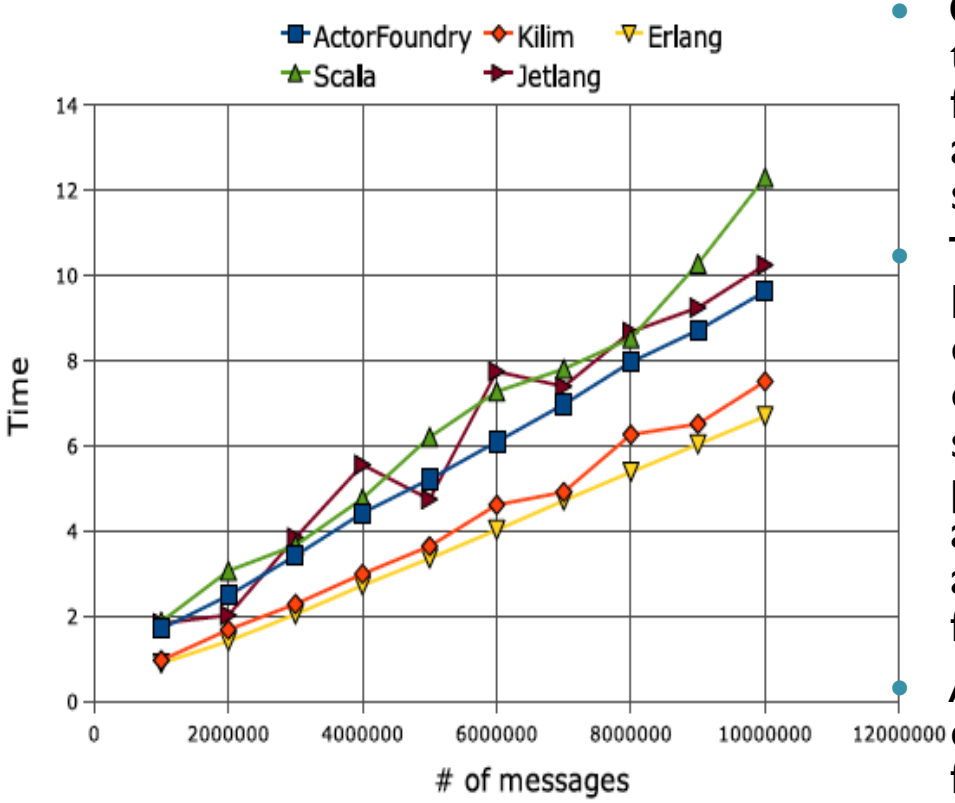

Figure 6: Threading Performance with other Frameworks and Scala.

- Observe that Kilim outperforms the rest (including Scala), since the framework provides light-weight actors and basic message passing support only.
- The programming model is low-level as the programmer has to directly deal with mailboxes, and it does not provide standard Actor semantics and common programming abstractions. This allows Kilim to avoid the costs associated with providing these features.
- ActorFoundry's performance is quite comparable to the other frameworks. This is despite the fact that ActorFoundry v1.0 preserves encapsulation, fairness, location transparency and mobility as in Scala.

Conclusion

- Scala is a Object oriented and functional
  - Statically typed
  - High performance
  - High interoperability with Java
  - Advanced type system
  - Advanced concurrency model
  - Support for modularity and extensibility
  - Built-in XML support

- Actors implemented as a Scala library implements Message-Passing Concurrency rather than Shared-State Concurrency.

- In Scala, patterns can be defined independently of case classes.

- Results suggest that safe messaging is the dominant source of inefficiency in actor systems.

Actor based Scala Pro's and Con's

	
Scala fuses object-oriented and functional programming in a statically typed programming language	This extensibility transfers responsibility from language designers to users.
Scala programs resemble Java programs in many ways and they can seamlessly interact with code written in Java	It is still as easy to design a bad libraries as it is to design a bad language.
It has flexible modular mixin-composition constructs for composing classes and traits	Type-system may be intimidating.
It allows decomposition of objects by pattern matching	
Scala compiles to the JVM	

Questions ?

Thank You!