Having fun with Play! 2, Akka and WebSockets

Victor Siu-Leung Chan

NE Scala Symposium 2014

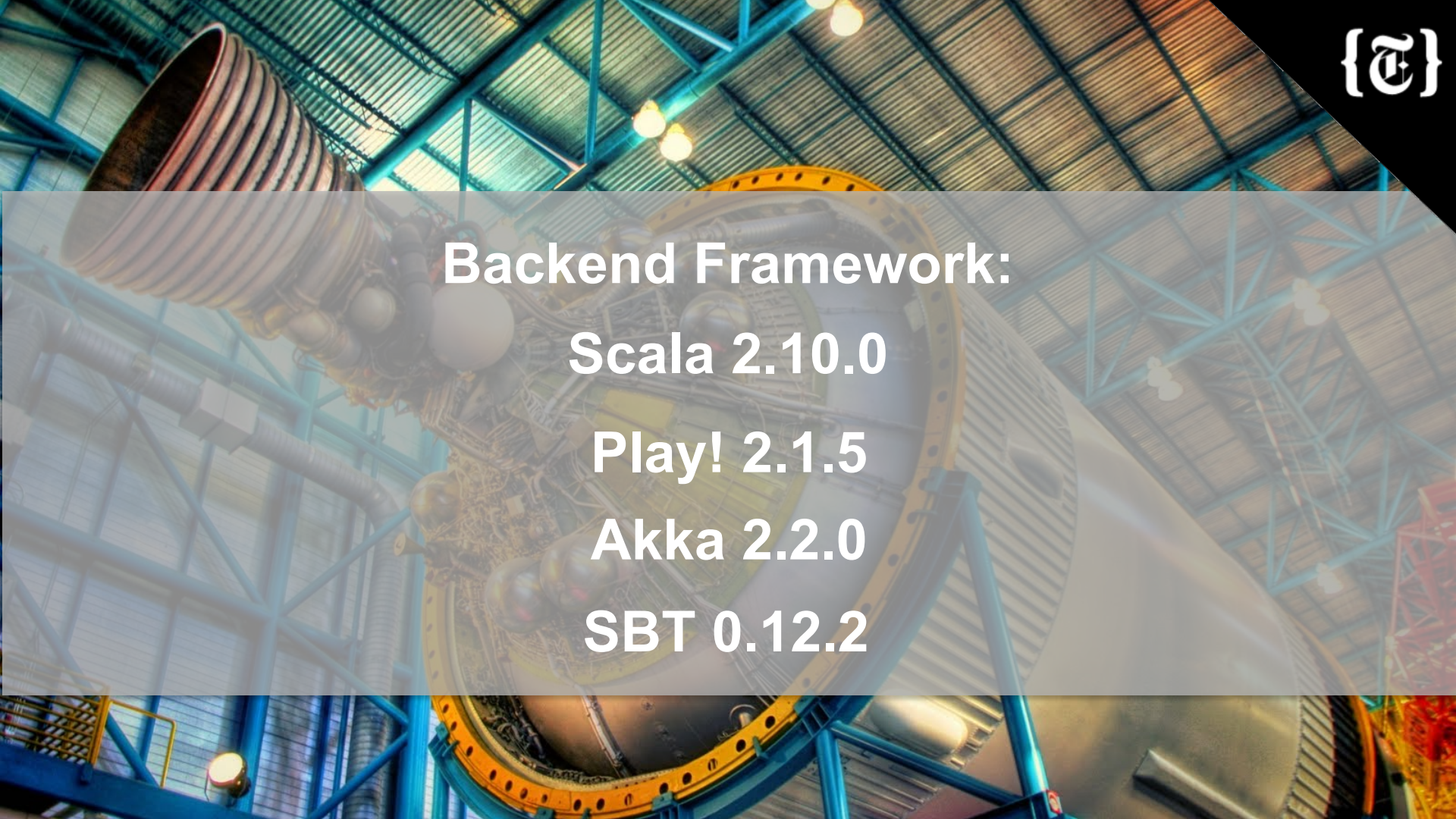
Blackbeard

NYT CMS to deliver news & articles

Single-page web application

High interaction among users

5 ~ 10 editors per article
300 ~ 400 articles per day

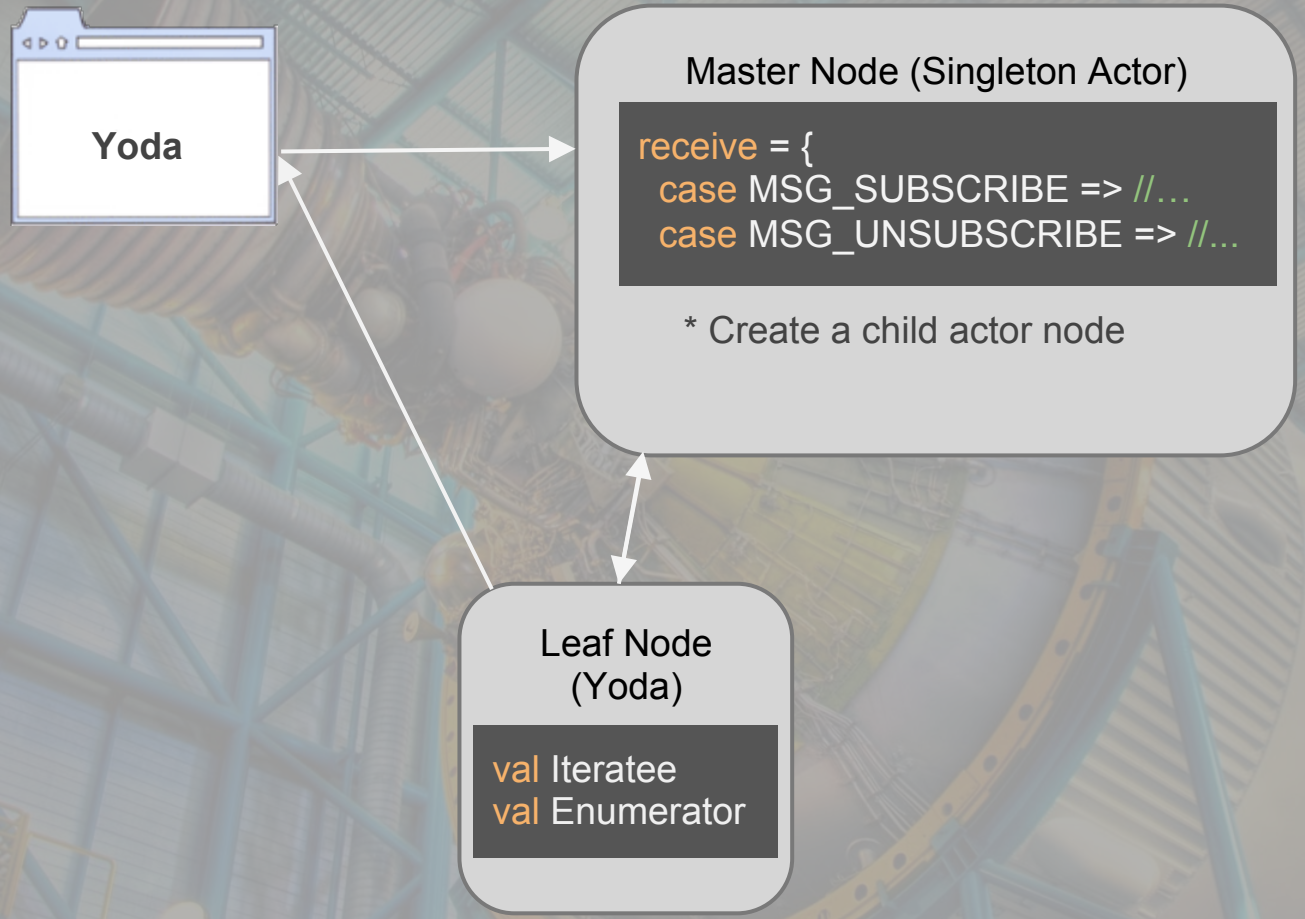

Backend Framework:
Scala 2.10.0
Play! 2.1.5
Akka 2.2.0
SBT 0.12.2

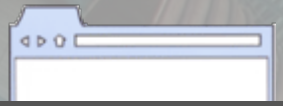
Master Node (Singleton Actor)

```
receive = {
  case MSG_SUBSCRIBE => //…
  case MSG_UNSUBSCRIBE => //...
```

Singleton object of WebSocket endpoint

```
object MasterNode {

  /* Get the singleton master actor */
  lazy val master: ActorRef = {
    val masterActor = Akka.system.actorOf(Props[MasterNode],
      name = "masterNode")
    masterActor
  }
}
```


Yoda
Master Node (Singleton Actor)
receive = {
case MSG_SUBSCRIBE => //…
case MSG_UNSUBSCRIBE => //...
* Create a child actor node
Leaf Node
(Yoda)
val Iteratee
val Enumerator

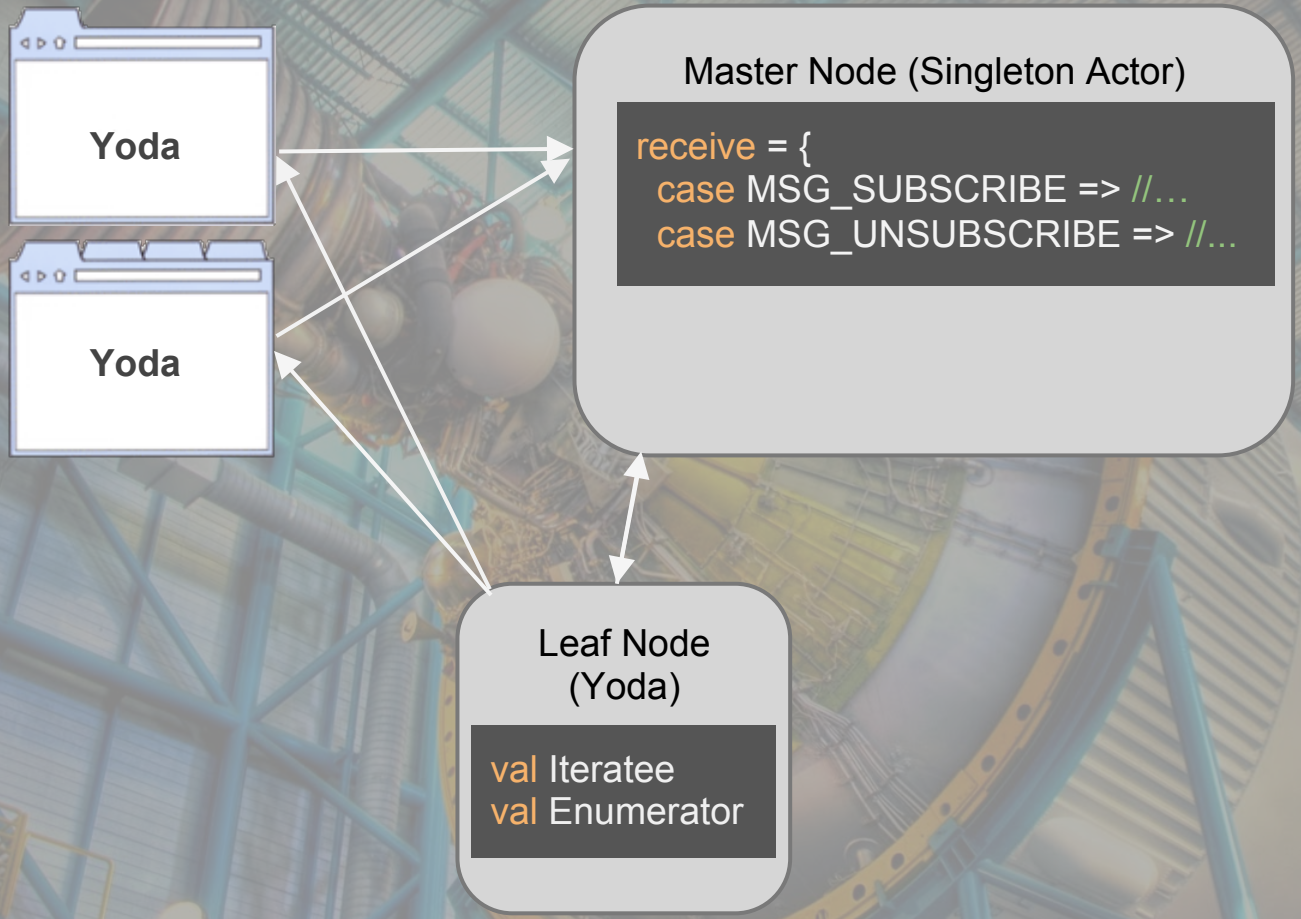


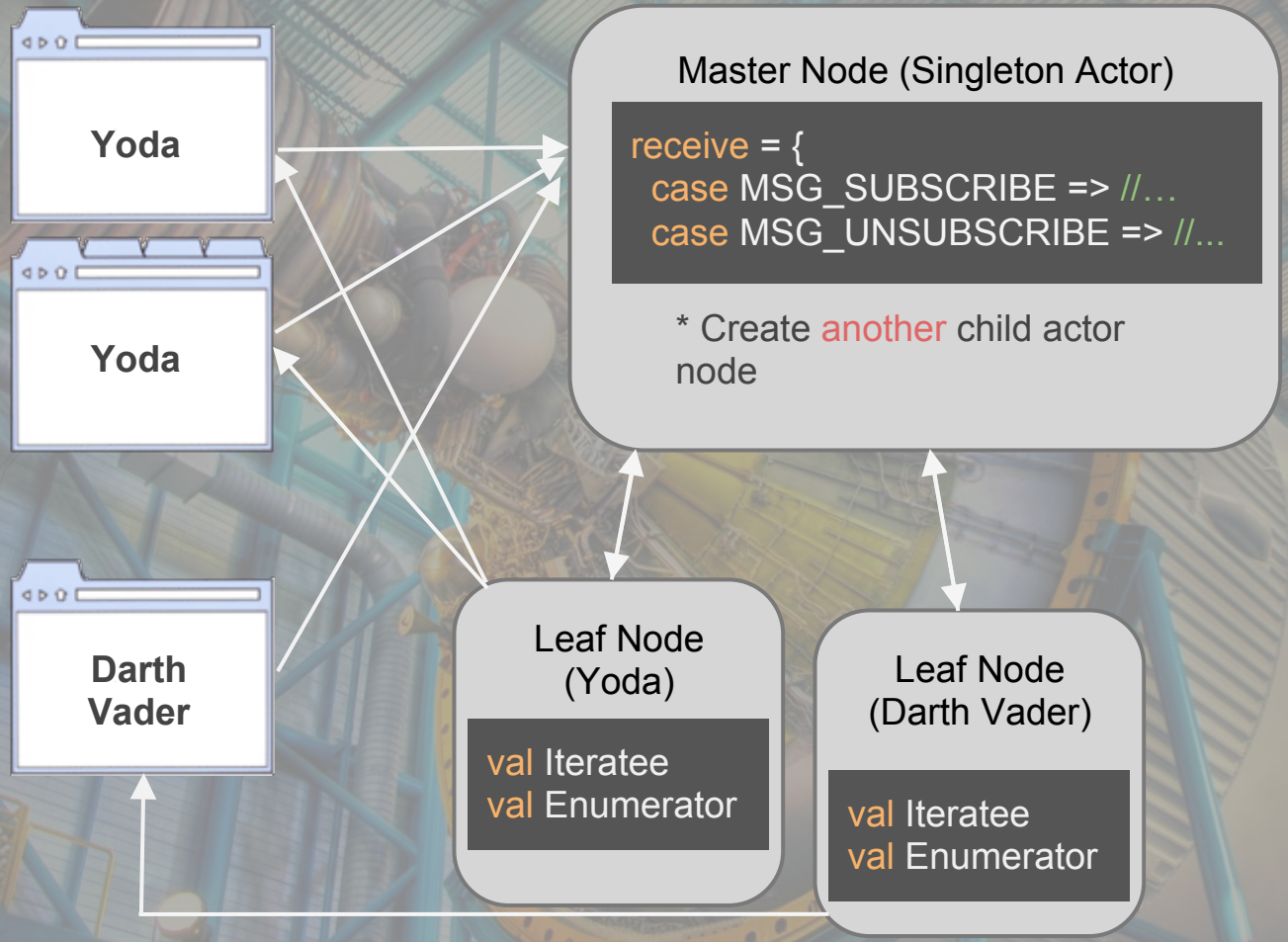
Master Node (Singleton Actor)

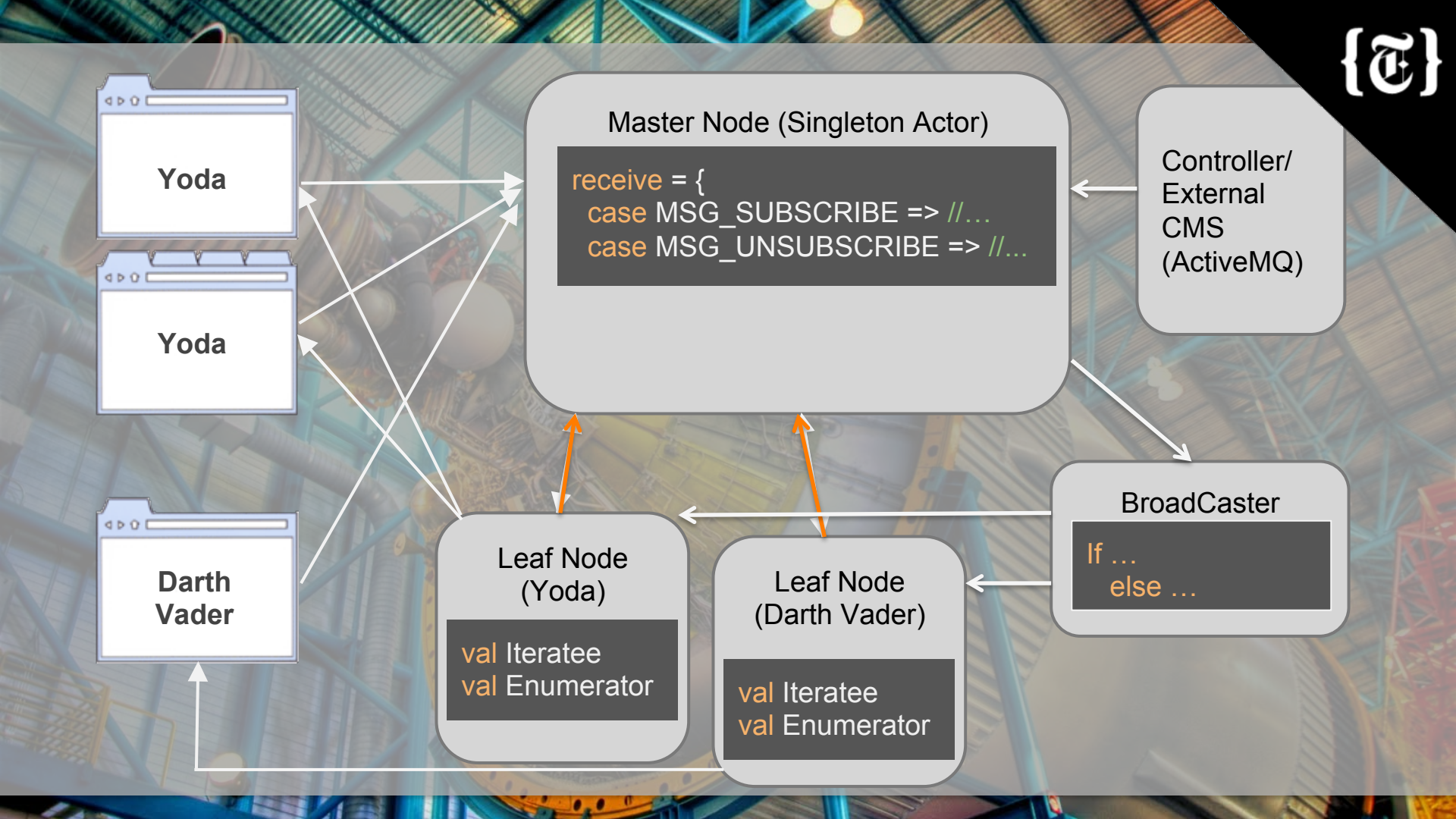
Iteratees and Enumerator

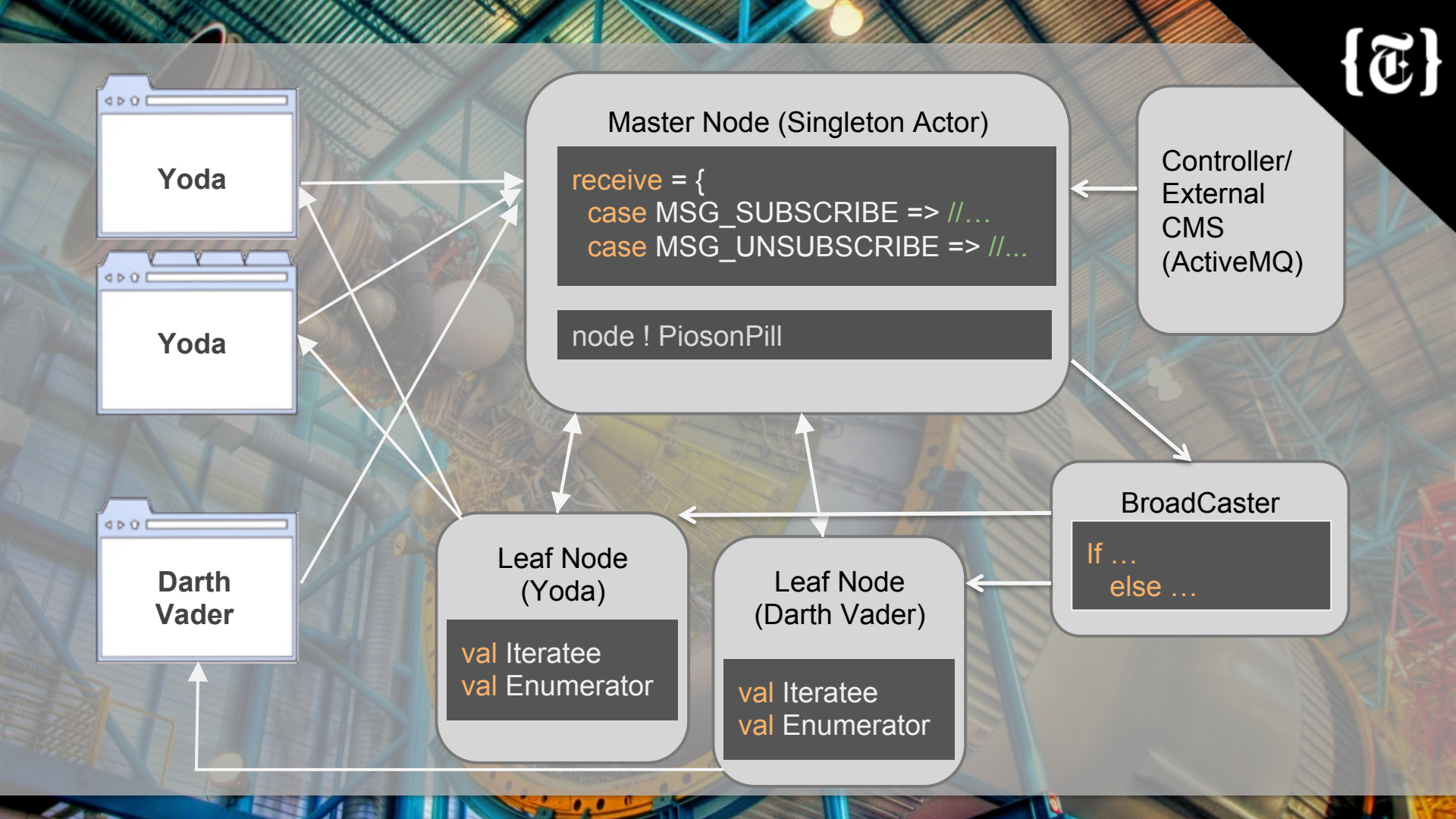
```
val in: Iteratee[JsValue, _] = Iteratee.foreach[JsValue] { msg =>
    context.parent ! Msg
}

val (outEnum, outChannel): (Enumerator[JsValue], Concurrent.Channel[JsValue]) =
    Concurrent.broadcast[JsValue]
```

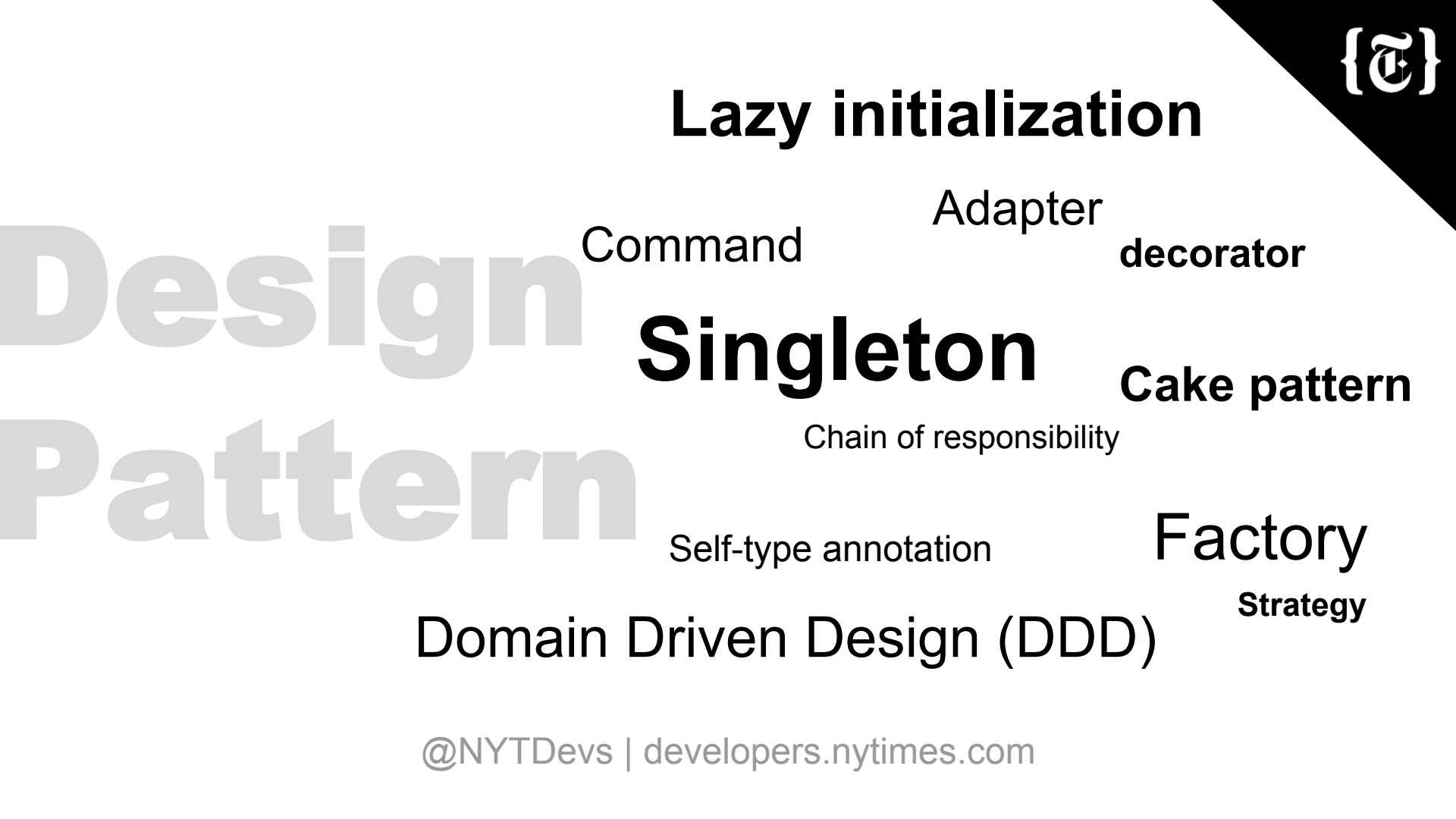
val Iteratee
val Enumerator


Yoda
Yoda
Master Node (Singleton Actor)
receive = {
case MSG_SUBSCRIBE => //…
case MSG_UNSUBSCRIBE => //...
Leaf Node
(Yoda)
val Iteratee
val Enumerator


Yoda
Yoda
Darth Vader
Master Node (Singleton Actor)
receive = {
case MSG_SUBSCRIBE => //…
case MSG_UNSUBSCRIBE => //...
* Create another child actor node
Leaf Node (Yoda)
val Iteratee
val Enumerator
Leaf Node (Darth Vader)
val Iteratee
val Enumerator


Yoda
Yoda
Darth Vader
Master Node (Singleton Actor)
receive = {
case MSG_SUBSCRIBE => //…
case MSG_UNSUBSCRIBE => //...
Controller/ External CMS (ActiveMQ)
Leaf Node (Yoda)
val Iteratee
val Enumerator
Leaf Node (Darth Vader)
val Iteratee
val Enumerator
BroadCaster
If …
else …


Yoda
Yoda
Darth Vader
Master Node (Singleton Actor)
receive = {
case MSG_SUBSCRIBE => //…
case MSG_UNSUBSCRIBE => //...
node ! PiosonPill
Controller/ External CMS (ActiveMQ)
Leaf Node (Yoda)
val Iteratee
val Enumerator
Leaf Node (Darth Vader)
val Iteratee
val Enumerator
BroadCaster
If …
else …


Design
Pattern
Lazy initialization
Adapter
Command
decorator
Singleton
Cake pattern
Chain of responsibility
Self-type annotation
Factory
Strategy
Domain Driven Design (DDD)

Result

Sharp learning curve

Quick development cycle. Agile

User satisfaction ☺

What we've learned

Filters for every HTTP request

```
object AuthFilter extends Filter with Secured {

  override def apply(next: RequestHeader => Result)(rh: RequestHeader): Result = {
    if (authRequired(rh)) {
      checkAuthentication(next)(rh)
    } else next(rh)
  }
```

What we've learned

Qualify private method, easier to test

```
package blackbeard.controllers

private[controllers] def workflowHelper = { … }  //private to the package

==========================================
object ArticleControllerSpec extends Specification {  //in testing package
    "private helper function" should  {
        "be accessed" in {
            workflowHelper
```

What we've learned

Define custom write for serializable object

```
implicit val articleWrites = Json.writes[Article]

def articleMinWrites = new Writes[Article] {
    def writes(a: Article): JsValue = {
    //get only the core fields ….

Json.toJson(article)(articleMinWrites)
```

What we've learned

Restful Call over WebSocket message to command

Good for frontend framework

Use Play JSON validation API

Keep actor logic as simple as possible

What we've learned

Beware of using Future within Actor receive - sender is not there!

```
protected def handleQueryMsg: Receive = {
  case MSG_QUERY(js) =>
      val f = Future {...}
      f map { result => sender ! MSG_RESULT(result) } // Dead letters!!!
```

What we've learned

Using Future, instead of actors (execution context)

```
implicit val ec: ExecutionContext =
ExecutionContext.fromExecutor(Executors.newCachedThreadPool())

Future { …. /* some expensive work */ }  //Running in ec!
```

Thank you!

{ Question? }

http://open.blogs.nytimes.com/

victor.chan@nytimes.com