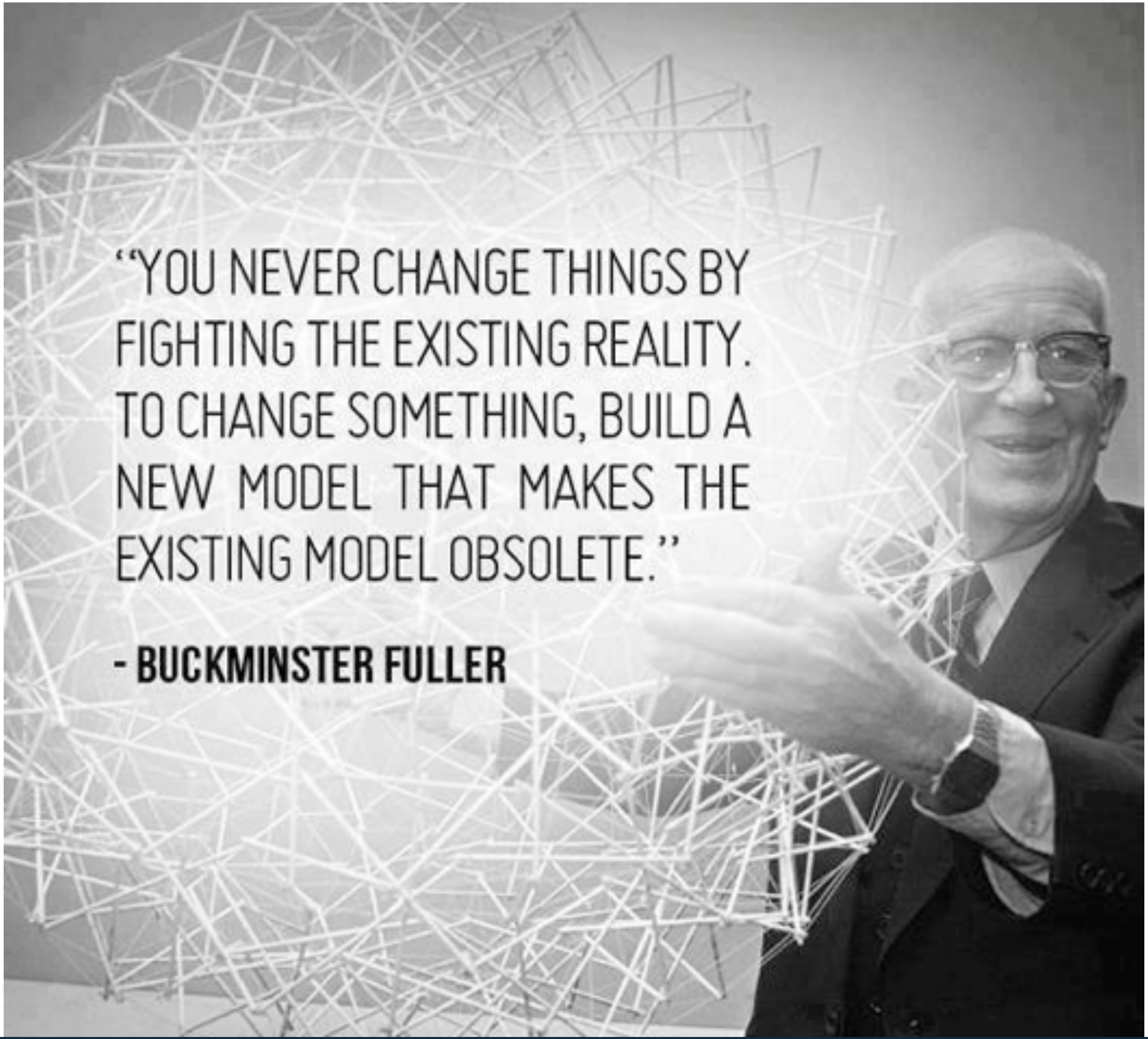
Typesafe

Tweetmap Java 8 Workshop

Ryan Knight
Typesafe
Consultant / Trainer
Twitter: @knight_cloud

Thank you!

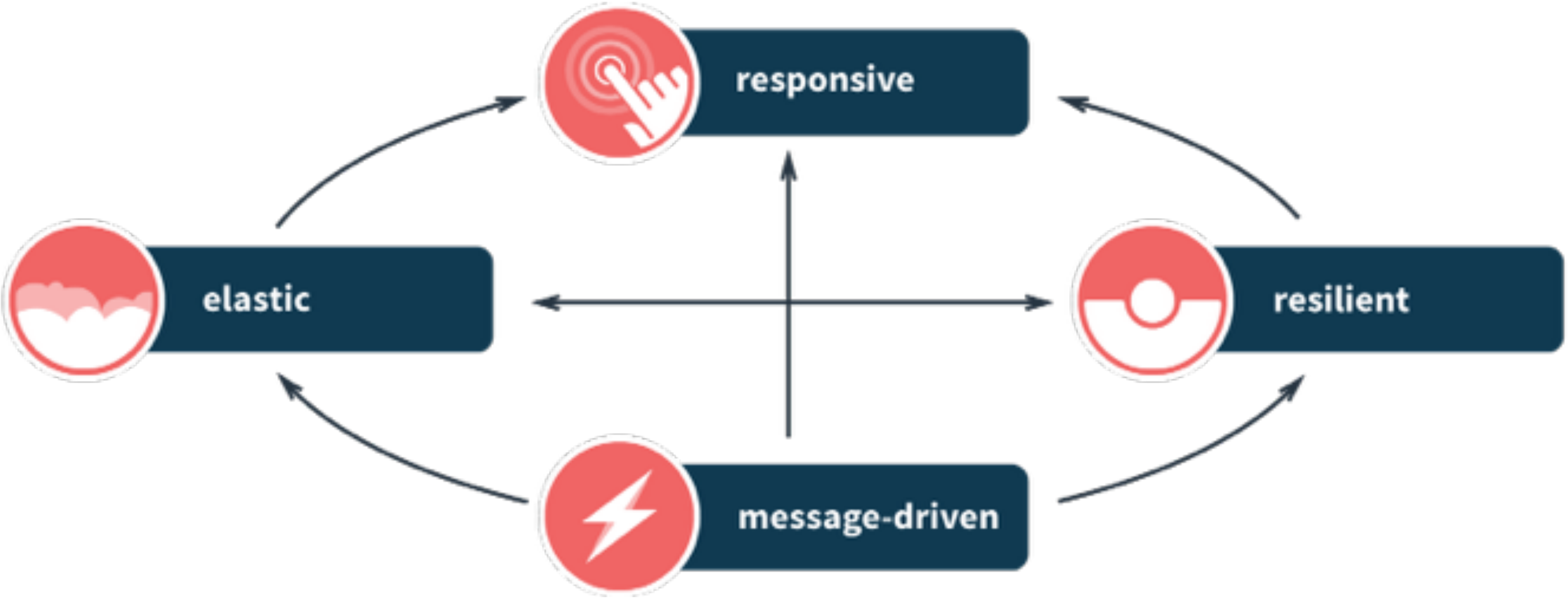
http://s4n.co



"YOU NEVER CHANGE THINGS BY FIGHTING THE EXISTING REALITY. TO CHANGE SOMETHING, BUILD A NEW MODEL THAT MAKES THE EXISTING MODEL OBSOLETE."

- BUCKMINSTER FULLER

Four Traits of Reactive Architectures



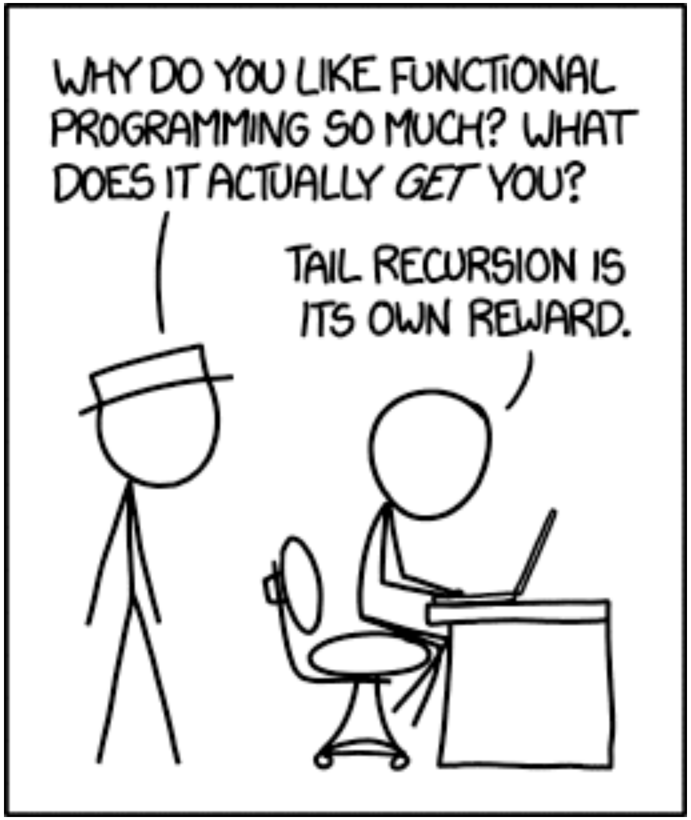
Building Blocks

Reactive Architectures

Java 8

Functions

Functional


WHY DO YOU LIKE FUNCTIONAL PROGRAMMING SO MUCH? WHAT DOES IT ACTUALLY *GET* YOU?
TAIL RECURSION IS ITS OWN REWARD.

XKCD

Why Functional Rocks!

- Immutability
- Higher-Level of Abstraction
- Define the What not the How
- Eliminating side effects
- Inherent Parallelism

Function is the Foundation for Reactive Programming

- Easy to create callbacks
- Easy to handle Events and Async Results
- Avoid Inner Classes

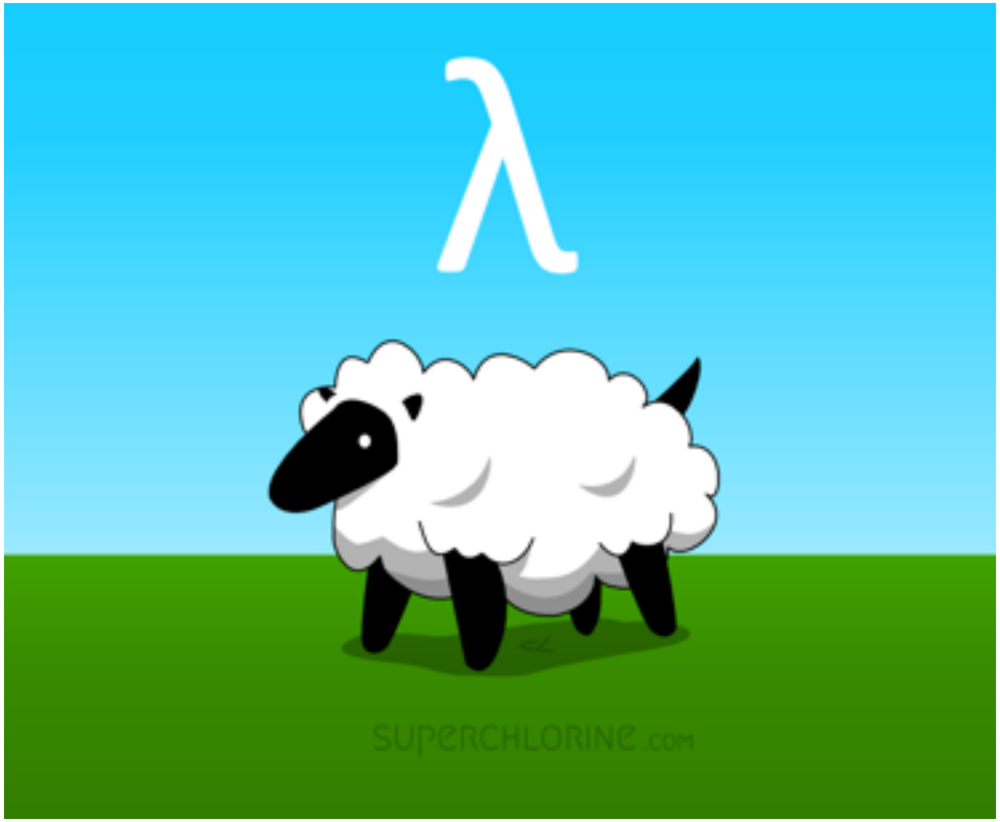
Functions as Values

- Similar to a method
    - Expression with 0 or more input arguments
- Store Functions in Values
- Pass Functions in Parameters
- Return Functions from other Functions
- Functions as Anonymous expression

Side-Affect Free

- Never Access Global State
- Never Modify Input
- Never Change the World

Java 8 Lambdas



Imperative Code

```
final List<Integer> numbers =
    Arrays.asList(1, 2, 3);

final List<Integer> numbersPlusOne =
    Collections.emptyList();

for (Integer number : numbers) {
 final Integer numberPlusOne = number + 1;
 numbersPlusOne.add(numberPlusOne);
}
```

We Want Declarative Code

- Remove temporary lists - List<Integer> numbersPlusOne = Collections.emptyList();
- Remove looping - for (Integer number : numbers)
- Focus on what - x+1

Lambda Expression Syntax

```
(int x, int y) -> x+y

() -> 59

//infers the type of x based on the parameters of the
functional method.
x -> x*2;


(int x, int y) -> x+y

n -> {
        int x = n+3;
        return (n == 5);
     }
```

Java 8

```java
import java.util.List;
import java.util.Arrays;
import java.util.stream.Collectors;

public class LambdaDemo {
 public static void main(String... args) {
    final List<Integer> numbers =
           Arrays.asList(1, 2, 3);

    final List<Integer> numbersPlusOne =
        numbers.stream().map(number -> number + 1).
                collect(Collectors.toList());
 }
}
```

λ

Functional Interface

- An interface with a single abstract method, called the functional method.
- Interface is implemented using a lambda or a method reference.
- Provide the target types for lambda expressions and method references.

Function - Functional Interface

```
@FunctionalInterface
public interface Function<T, R> {

    /**
     * Applies this function to the given argument.
     *
     * @param t the function argument
     * @return the function result
     */
    R apply(T t);
```

Target Type of Lambda

Function - Functional Interface

Function<T,R> - Function that accepts one argument and returns a result.

Signature:

Interface Function<T,R> - where T is the type of input, R is the type of result

R apply(T t) - applies this function to the given argument returning a type R

Predicate Functional Interface

```
@FunctionalInterface
public interface Predicate<T> {

   boolean test(T t);
```

Target Type of Lambda

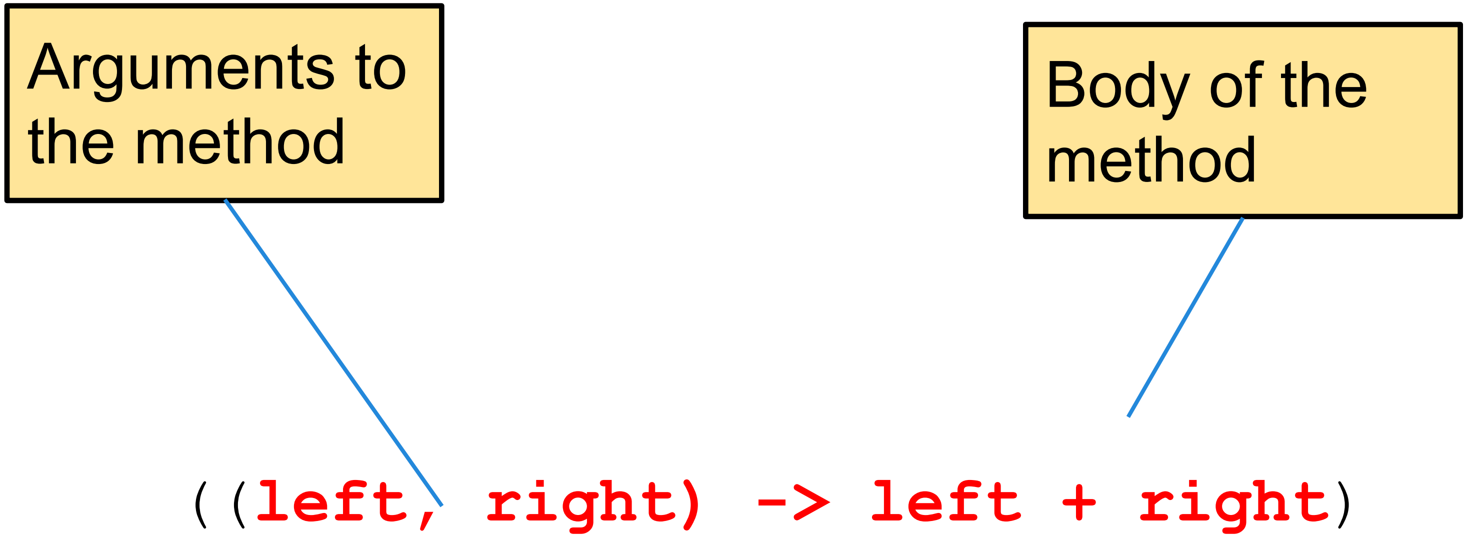
Lambdas in Action

```
public String test(
    BiFunction<String,
               String> combiner) {
   return combiner.accept(“Hello, “,
                          “World”);
}
…
String result =
  test((left, right) -> left + right);
//  result: Hello, World
```

Lambdas in Action

```
public String test(
    BiFunction<String,
               String> combiner) {
   return combiner.accept(“Hello, “,
                          “World”);
}
…
String result =
  test((left, right) -> left + right);
//  result: Hello, World
```

Lambdas in Action



Arguments to the method

Body of the method

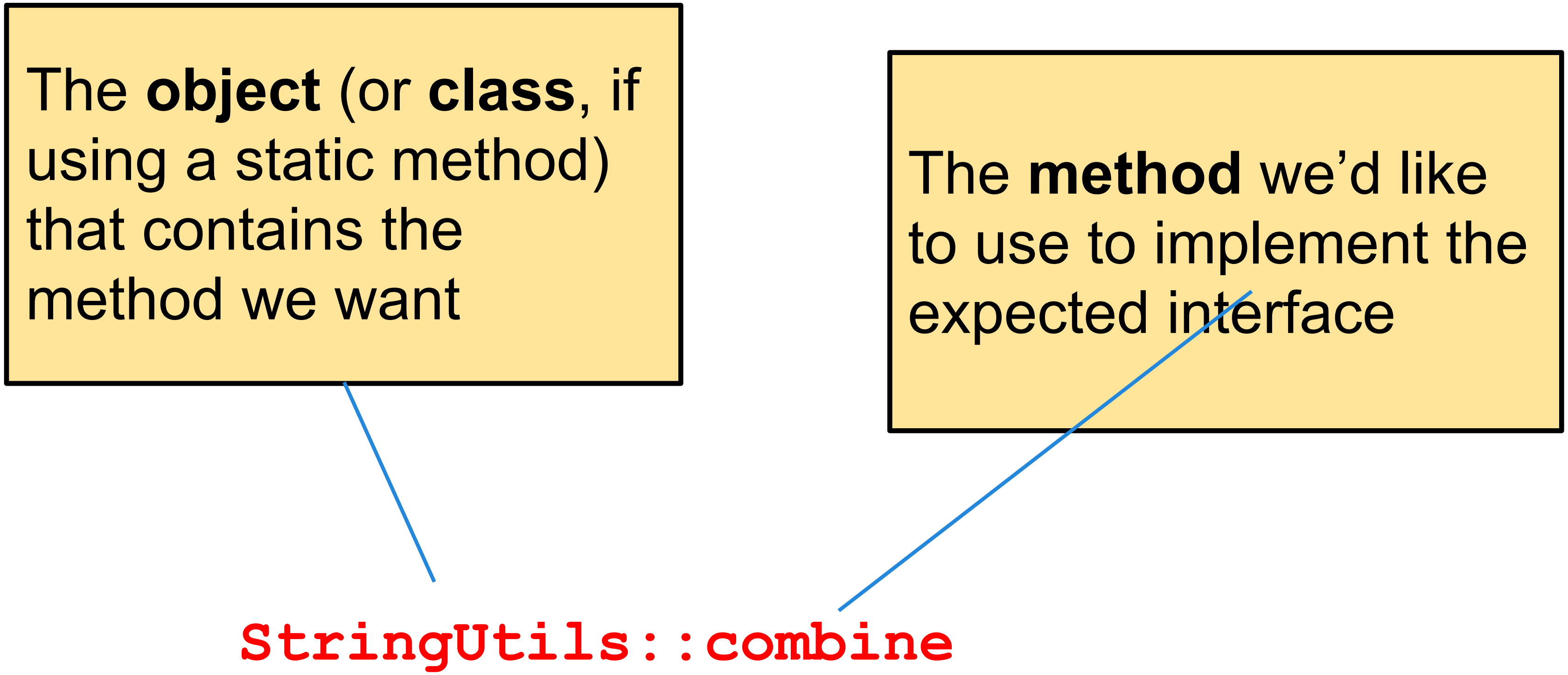
…

```
((left, right) -> left + right)
```

Method Reference

```
class StringUtils {
  public static String combine(String left,
                             String right)
{
    return left + right;
  }
}
...
String result =
  test(StringUtils::combine);
//  result: Hello, World
```

Method Reference



The **object** (or **class**, if using a static method) that contains the method we want

The **method** we'd like to use to implement the expected interface

```
StringUtils::combine
```

Other Functional Interfaces

- Consumer<T> - function from T to void
- Supplier<T> - function that doesn’t take any input and returns a type T
- Many other - look in:
  - package java.util.function;

Target Typing - The data type of a lambda expression is the target type.   The java compiler determines the target type in the following contexts:

- Variable declarations
- Assignments
- Return statements
- Array initializers
- Method or constructor arguments
- Lambda expression bodies
- Conditional expressions, ?:
- Cast expressions

Method references

- :: - operator
- Creates a lambda expression based on existing method, calling that method by name

Method reference examples

- A static method (ClassName::methName)
- An instance method of a particular object (instanceRef::methName)

```
Set<String> knownNames = new HashSet<>();
Predicate<String> isKnown = knownNames::contains;
```

- A super method of a particular object (super::methName)

Method reference examples

// An instance method of an arbitrary object of a particular type - (ClassName::methName)

```
            //Add people to the list

            BiConsumer<List<Person>, Person> longPeopleList = List::add;

            BiPredicate<Set<String>, String> genericKnown = Set::contains;
```

A class constructor reference (ClassName::new)

```
   //create a new Person ArrayList

   Supplier<List<Person>> arrayListSupplier = (Supplier<List<Person>>) ArrayList::new;
```

An array constructor reference (TypeName[]::new)

Function Contexts

```
// Assignment context
   Predicate<String> p = String::isEmpty;


   // Method invocation context
   stream.filter(e -> e.getSize() > 10)...


   // Cast context
   stream.map((ToIntFunction) e -> e.getSize())...
```

Functions In Collections

```
import static java.util.Comparator.comparing;

// comparing takes a Function that returns a key and it returns a  Comparator
// based on the key

Collections.sort(personList, comparing(Person::getGivenName));

personList.sort(comparing(Person::getGivenName));
```

Java 8 Optional

Use instead of null checks:

```
Optional<String> optStr= Optional.empty();

Optional.of

Optional.ifPresent
```

Futures and Promises

Futures

- A handle to a future value
- Result of a calculation or service call
- Small independent task executed asynchronously

Java 8 Futures

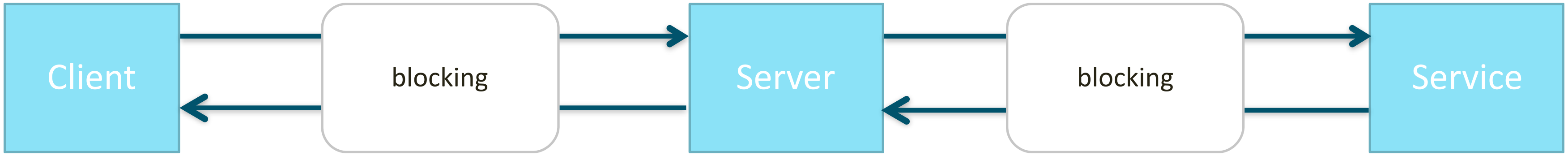
- java.util.concurrent.CompletableFuture
- Future operation specified as a Function
- Callback specified as Function when the Future completes

```
final CompletableFuture<Object> cacheFuture = CompletableFuture
                .supplyAsync(() -> {
                    return cacheRetriever.getCustomer(id);
                });

cacheFuture.thenApply(customer -> dbService.getOrders(customer))
```

Introducing the **Typesafe Platform**

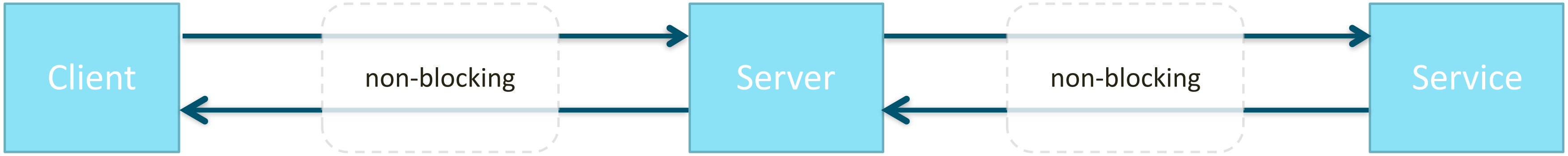
play

Traditional Request/Response



```
def getTweets = Action {
  Ok(WS.get("http://twitter.com/"))
}
```

Reactive Request/Response



```
def getTweets = Action.async {
  Ok(WS.get("http://twitter.com/"))
}}
```

High-Velocity
Web Framework for the Cloud

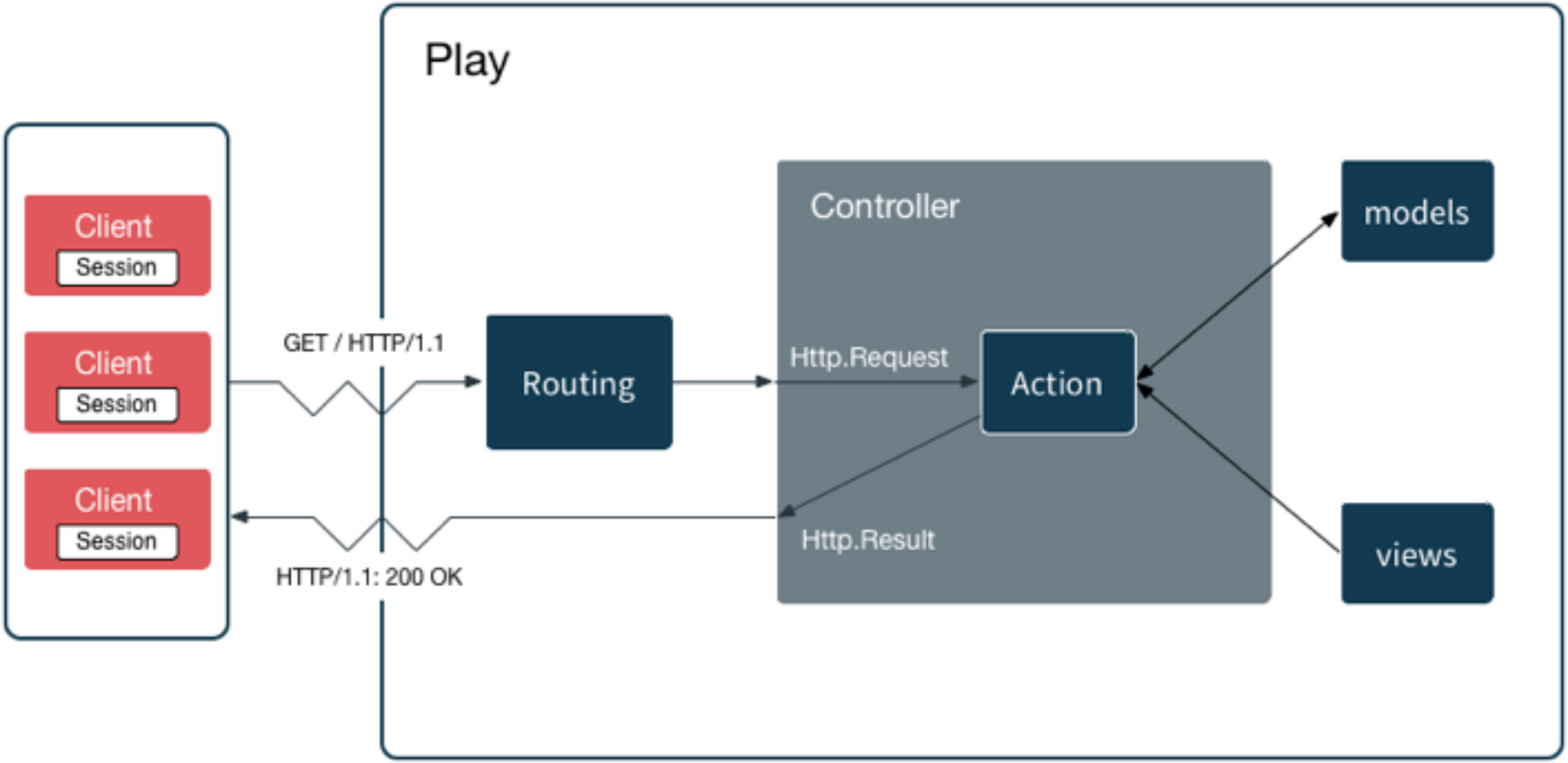
- Reactive and Asynchronous
- RESTful by default
- Type Safety - Fully Compiled
- Just hit refresh workflow
- No Magic - i.e. no dynamic mix-ins

Play Features

- Stateless - Predictable Horizontal Scalability
- Routes File
    - Declarative, Type-safe URL Mapping
    - Friendly URLs for SEO and Humans
- Websockets and SSE
- Tight integration with Akka

Anatomy of Play

Stateless: Session is stored in the browser cookie



Routing

```
# this is a comment

GET   /                 controllers.Application.index()

GET   /page             controllers.SomeOtherController.page()

POST  /persons/:name    controllers.PersonCtrl.update(name)


#  HTTP methods: GET, POST, PUT, DELETE, HEAD, OPTIONS
```

Controller

```
package controllers
import play.mvc.*;
public class Application extends Controller {
  public statuc Result index() {
    return ok("Hello everybody!"); // returns a Result
  }
}
```

Results

```
ok("Hello world!");                          // 200
badRequest("You had an error in your form");       // 400
notFound();                                  // 404
notFound("<h1>Page not found</h1>").as("text/html"); // 404
internalServerError("Oops");                 // 500
status(488, "Strange response type");
```

Application Configuration

Default configuration file: conf/application.conf

```
# Custom configration
my.custom.key="My Value"
file.to.be.included="filename.conf"
key.as.array=["one", "two", "three"]
```

Based on Typesafe config library

Which uses HOCON format

Access Configuration

First import the Application Context:

```
import static play.Play.application;

String key =
application().configuration().getString("my.custom.key");
```

Logging

```
import play.Logger


Logger.debug("Want to trace something down..");

Logger.warn("Watch out!");


String error = "Detailed error description";

Logger.error("Something went wrong: " + error);
```

Configuring Logging

```
# Root logger:
logger.root=ERROR
# Logger used by the framework:
logger.play=INFO
# Logger provided to your application:
logger.application=DEBUG
# Logger for a third party library
logger.org.springframework=INFO
```

Views

Directory: app/views/

Suffix for view files: .scala.html

Reference views: views.html...render()

// views/index.scala.html

views.html.index.render()

// views/foo/bar.scala.html

views.html.foo.bar.render()

Managed Assets

Compiles and minifies files automatically into CSS and JavaScript

Located in folder: app/assets

- app/assets/stylesheets → LESS
- app/assets/javascripts → CoffeeScript

Driven by SBT plugins

Static Assets

Sources in folder: public

- public/images → Images
- public/stylesheets → CSS
- public/javascripts → JavaScript

LESS

```
// plugins.sbt

addSbtPlugin("com.typesafe.sbt" % "sbt-less" % “1.0.0")
```

RequireJS

RequireJS is a JavaScript file and module loader

Improves speed and quality of your JavaScript files

```
// plugins.sbt

addSbtPlugin("com.typesafe.sbt" % "sbt-rjs" % "1.0.0")
```

GZIP

```
// plugins.sbt
addSbtPlugin("com.typesafe.sbt" % "sbt-gzip" % “1.0.0")
```

```
// build.sbt
pipelineStages := Seq(gzip)
```

Compress web assets

Smaller HTTP responses for static content

Asset Fingerprinting

Adding checksum for web assets

Far future caching based on fingerprint

Necessary to invalidate caches

```
// plugins.sbt

addSbtPlugin("com.typesafe.sbt" % "sbt-digest" % "1.0.0")

// build.sbt

pipelineStages := Seq(digest)
```

Assets Controller

Configured in conf/routes:

GET /assets/*file Assets.at("public", file)

Enables ETag

Build

- Activator for UI Build - activator ui
- Activator is a UI around sbt
- activator at the command line can be used to build
  - Commands: run, compile, test
  - Continuous mode: ~ (e.g. ~run)

Build System: build.sbt

- look at build.sbt in root directory
- Dependency resolution uses Ivy (which ressembles Maven)

Application Global Object

Handle global settings for application

Can be defined by creating Global.java in /app directory:

```
import play.*;

public class Global extends GlobalSettings {

 }
```

Global Events

```
public class Global extends GlobalSettings {
 @Override
 public void onStart(Application app) {
  Logger.debug("Log something..");
 }
 // Override Page Not Found Behaviour
 @Override
 public Promise<Result> onHandlerNotFound(RequestHeader request) {
  return Promise.pure(badRequest(views.html.notFound.render()));
 }
```

akka

Actors

- Isolated lightweight processes
- Message Based / Event Driven
- Non-Request Based Lifecycle
- Share nothing
- Isolated Failure Handling

Actors

- Run Asynchronously
- Processes one message at a time
- Everything inside the actor is sequential
- Sane Concurrency
- Isolated Failure Handling

Akka

- Actor Based Toolkit
- Simple Concurrency & Distribution
- Error Handling and Self-Healing
- Elastic and Decentralized
- Adaptive Load Balancing

Akka Clustering

- Peer-to-peer based cluster membership service
- Cluster Events
- Cluster-Aware Routers
- Uses gossip protocols
- No single point of failure or single point of bottleneck.
- Automatic node failure detector

Akka Clustering

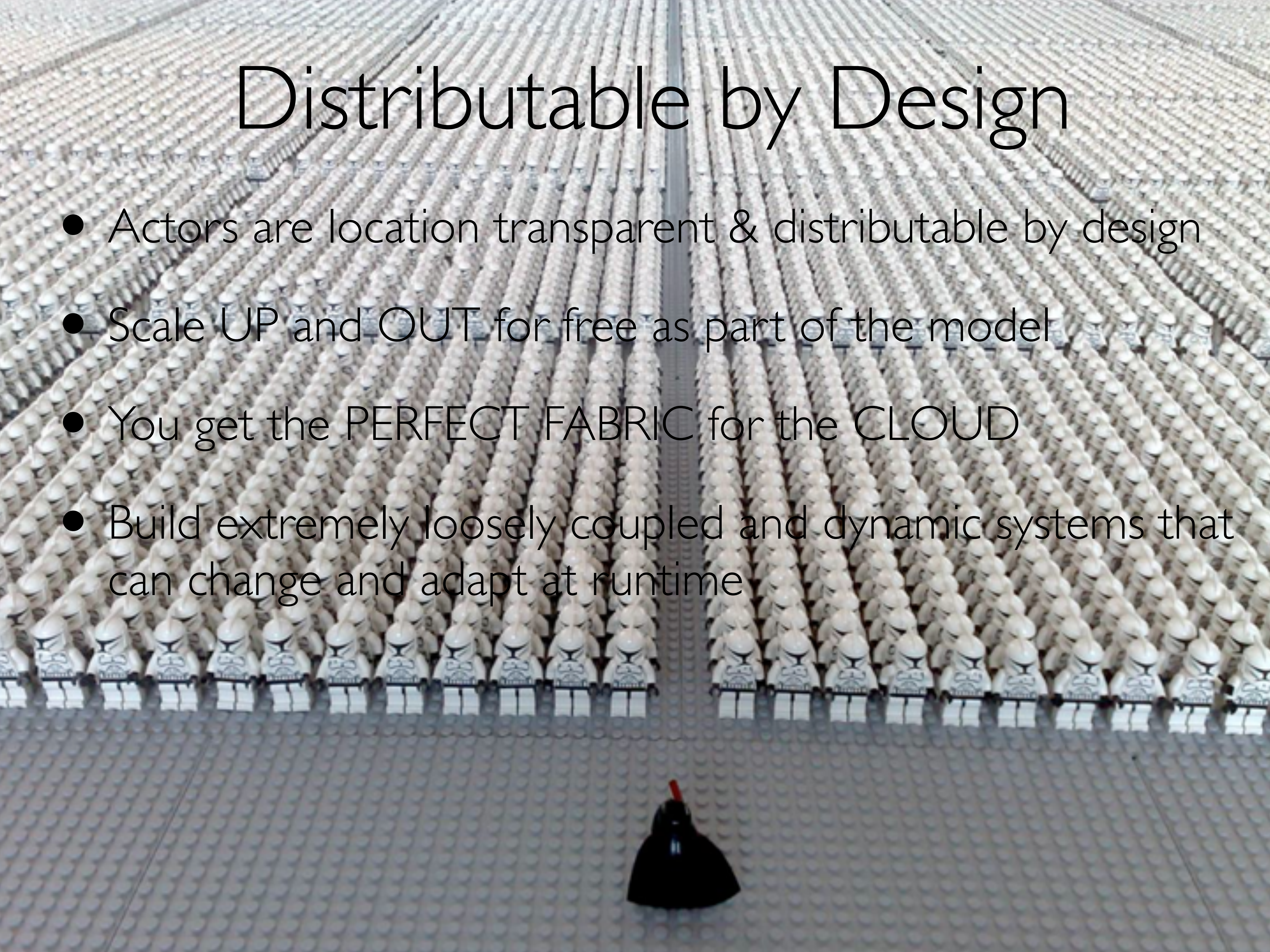
- Peer-to-peer based cluster membership service
- Uses gossip protocols
- No single point of failure or single point of bottleneck.
- Automatic node failure detector

Akka Persistence

- Based on Event Sourcing
- Messages persisted to Journal and replayed on restart
- Great for implementing
  - durable actors
  - replication
  - CQRS etc.

Distributable by Design

- Actors are location transparent & distributable by design
- Scale UP and OUT for free as part of the model
- You get the PERFECT FABRIC for the CLOUD
- Build extremely loosely coupled and dynamic systems that can change and adapt at runtime



Introducing

The Actor Model

Actor Features

- Each actor has a mailbox (message queue)
- Each actor has a parent handling its failures
- Interaction done via an Actor Reference
  - Location transparent - Distributable
  - Lifecycle Independent - Transparent Restart

The Actor Model

A computational model that embodies:

- ✓ Processing
- ✓ Storage
- ✓ Communication

Supports 3 axioms—when an Actor receives a message it can:

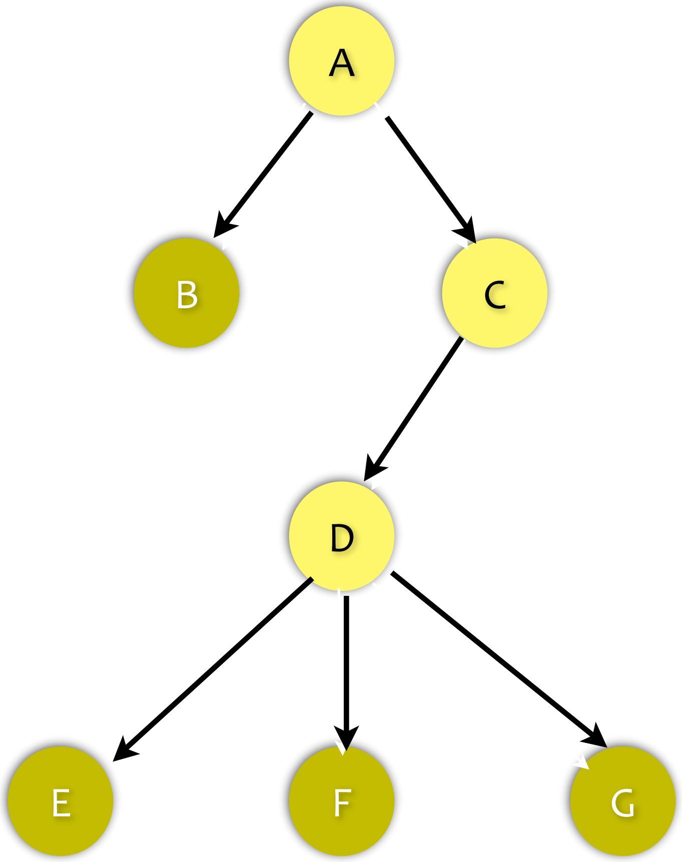
1. Create new Actors
2. Send messages to Actors it knows
3. Designate how it should handle the next message it receives

The Essence of an Actor

0. DEFINE
1. CREATE
2. SEND
3. BECOME
4. SUPERVISE

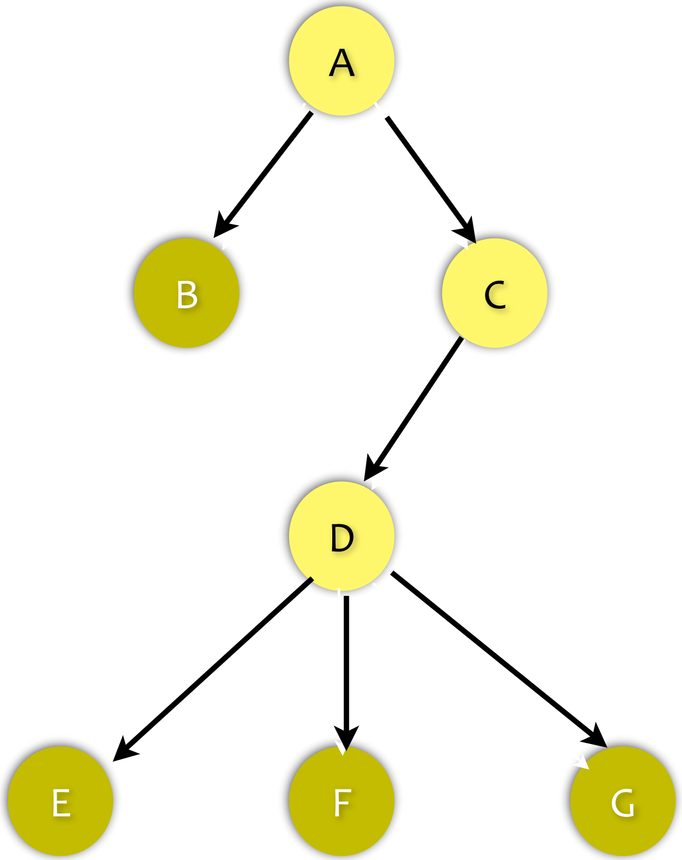
Akka Supervisor Hierarchies

- Hierarchy is the core of Akka's **supervision** strategy
  - Parents supervise children actors
  - Children delegate failure to parent



Akka Supervisor Hierarchies

- Easily scale a task by creating multiple instances of an actor and sending work using various routing strategies



Failure Recovery

- Supervisor hierarchies with “let-it-crash” semantics
- Lifecycle Monitoring
- Parent can resume, restart or terminate Child
- Error-prone tasks are delegated to child Actors - “Error Kernel Pattern”

0. D

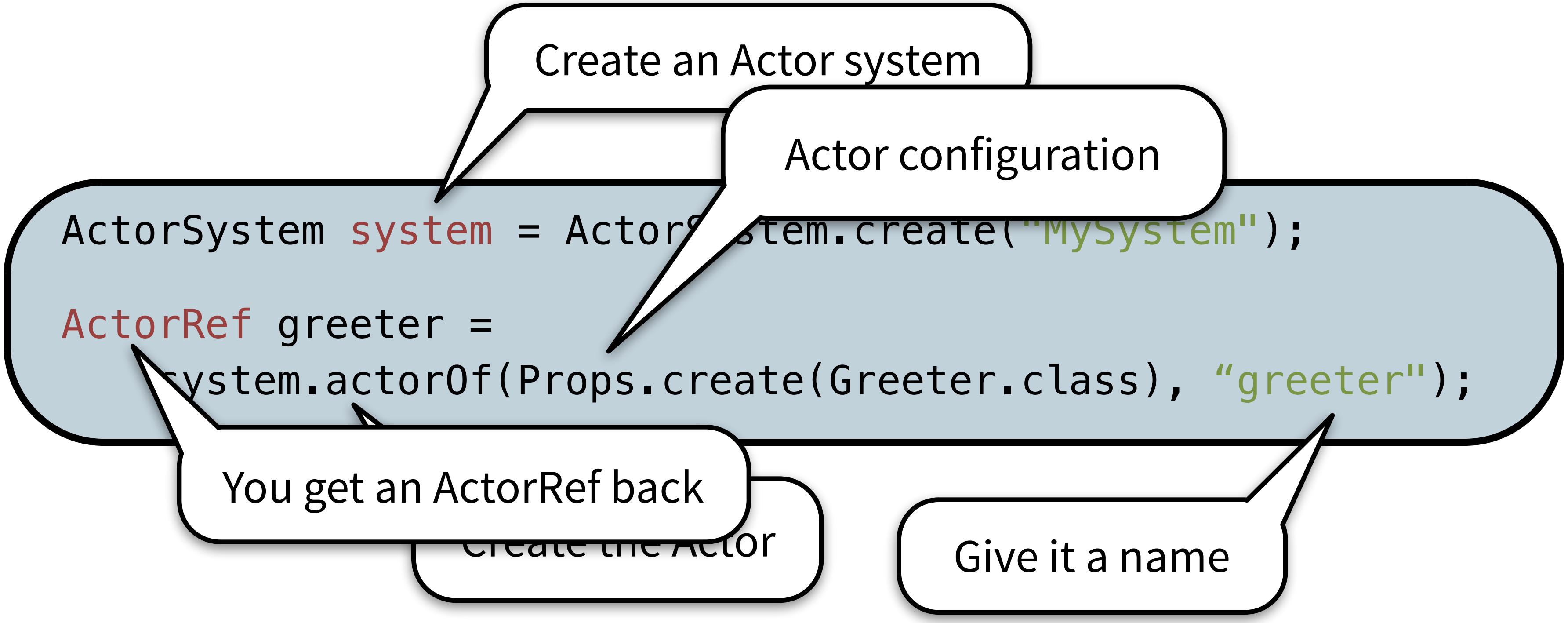
Define the message(s) the Actor should be able to respond

```
public class Greeting implements Serializable {
  public final String
  public Gre                                  ho; }
}


public class Greeter extends AbstractActor {{
  receive(ReceiveBuilder.
    match(Greeting.class, m -> {
      println("Hello " + m.who);
    }).
    matchAny(unknown -> {
      println("Unknown message " + unknown);
    }).build());
}}
```

Define the Actor class

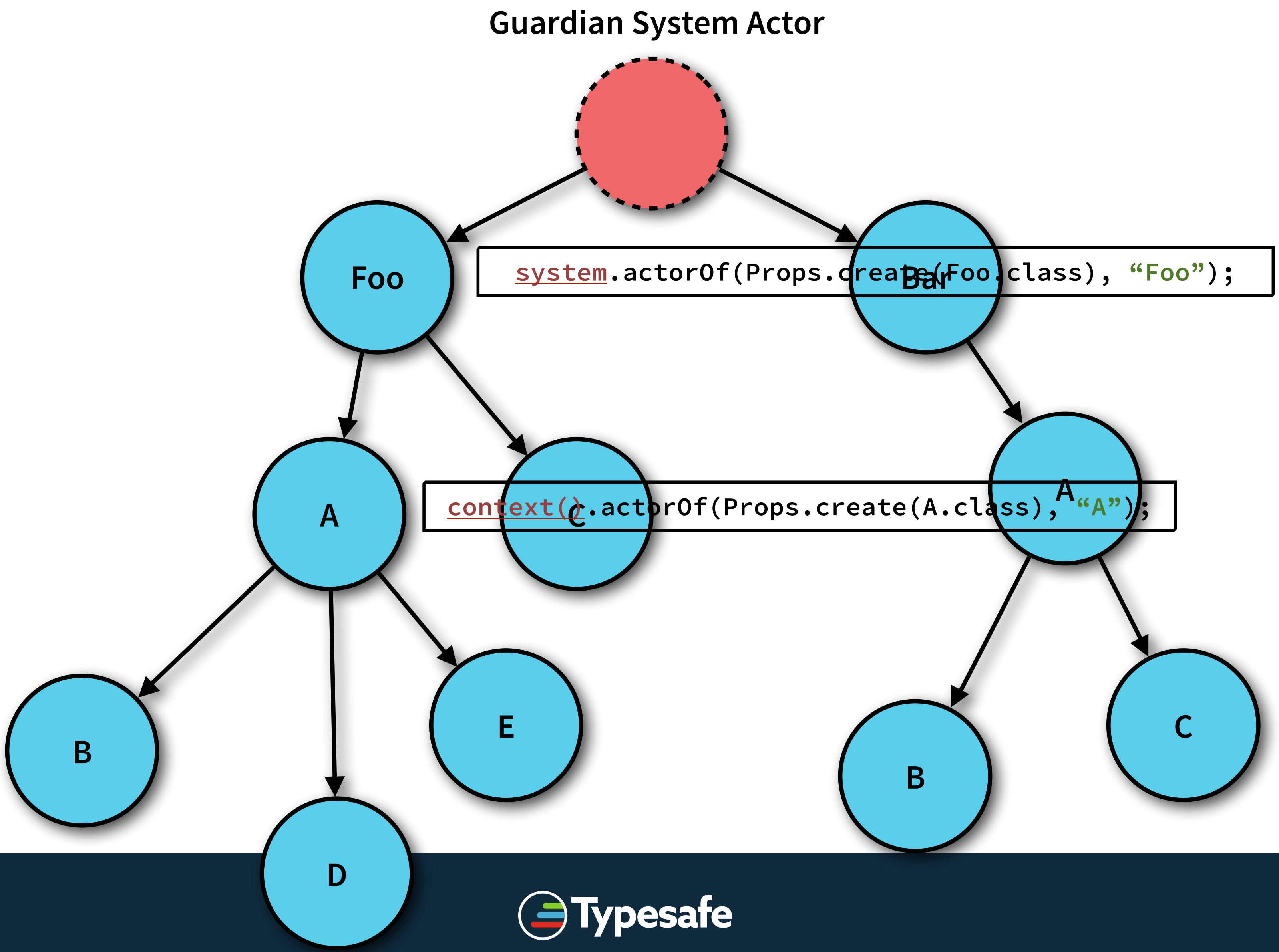
Define the Actor's behavior

1. CREATE



```
ActorSystem system = ActorSystem.create("MySystem");

ActorRef greeter =
   system.actorOf(Props.create(Greeter.class), “greeter");
```

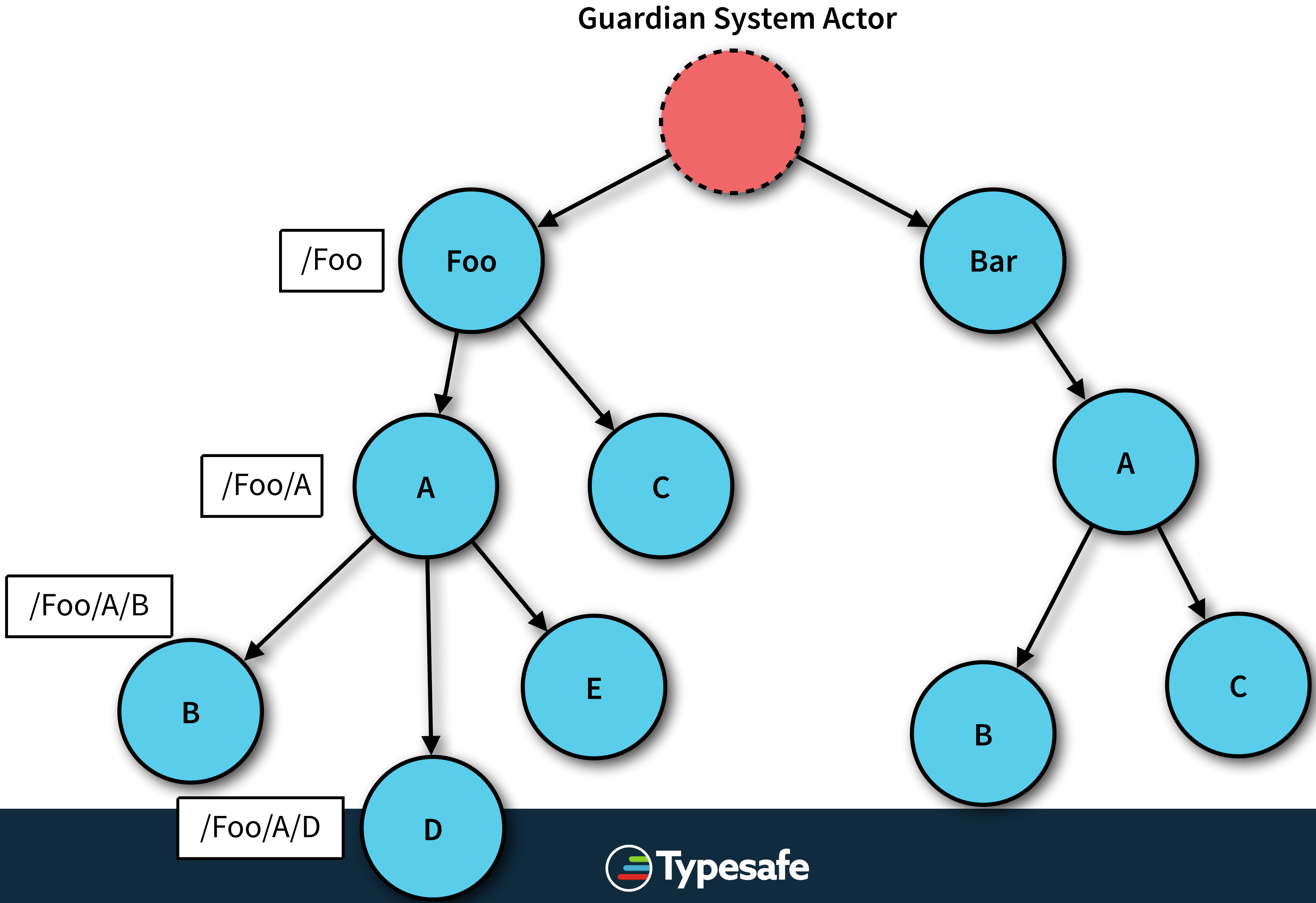
Actors can form hierarchies



```
system.actorOf(Props.create(Foo.class), “Foo”);
```

```
context().actorOf(Props.create(A.class), “A”);
```

Name resolution—like a file-system



2. SEND

Pass in the sender ActorRef

```
greeter.tell(new Greeting("Charlie Parker”), sender);
```

Send the message asynchronously

Bring it together

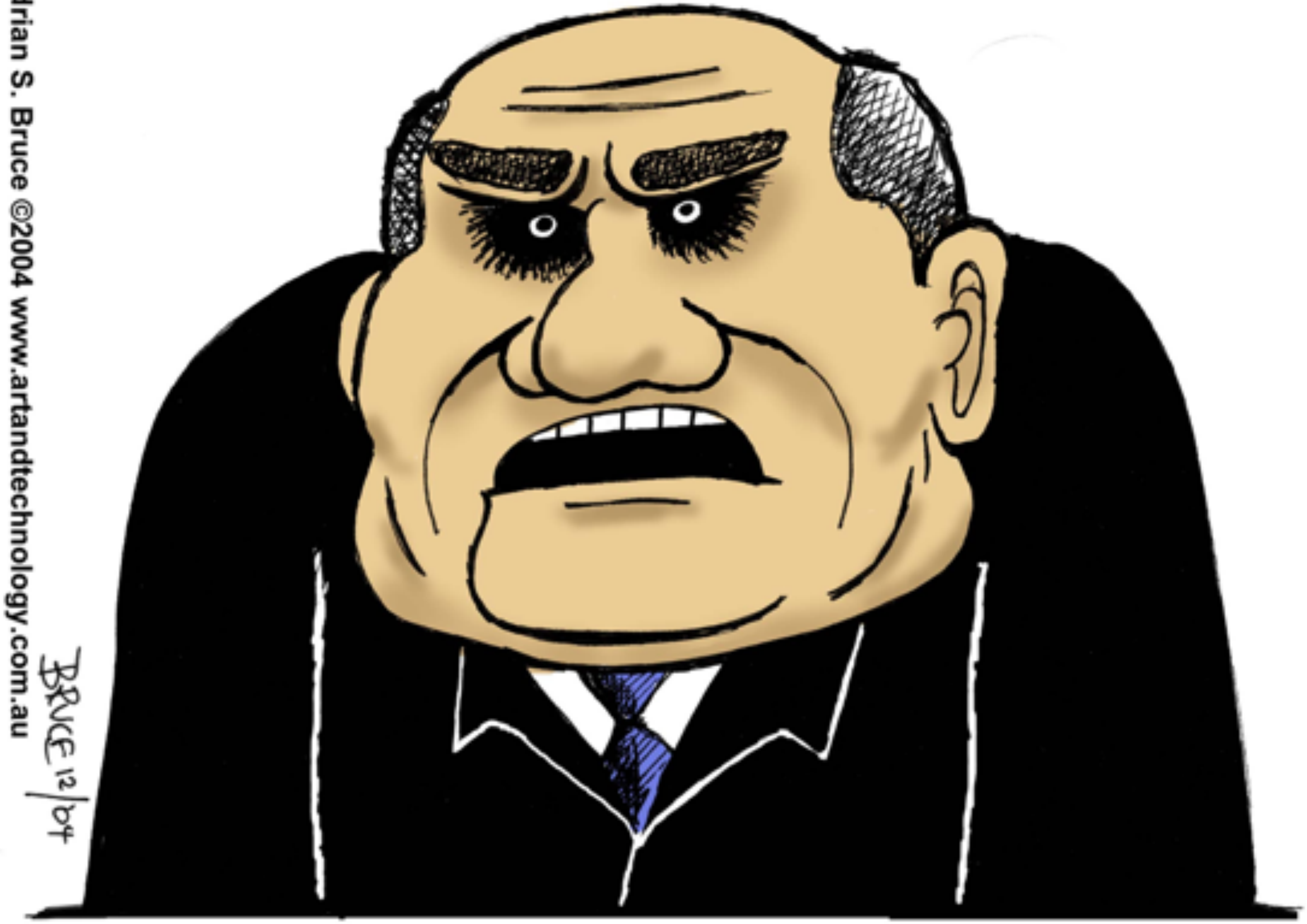
```
public class Greeting implements Serializable {
   public final String who;
   public Greeting(String who) { this.who = who; }
}
public class Greeter extends AbstractActor {{
  receive(ReceiveBuilder.
    match(Greeting.class, m -> {
      println(“Hello " + m.who);
    }).
    matchAny(unknown -> {
      println(“Unknown message " + unknown);
    }).build());
  }
}}

ActorSystem system = ActorSystem.create("MySystem");
ActorRef greeter = system.actorOf(Props.create(Greeter.class), “greeter");
greeter.tell(new Greeting(“Charlie Parker”));
```

3. BECOME

```
public class Greeter extends AbstractActor {
  public Greeter {
    receive(ReceiveBuilder.
      match(Greeting.class, m -> {
        println(“Hello " + m.who);
      }).
      matchEquals(“stop"      {
        context().become(ReceiveBuilder.
          match(Greeting.class, m -> {
            println(“Go Away!”);
           }).build());
     }).build();
  }
}
```

Change the behavior

Enter Supervision

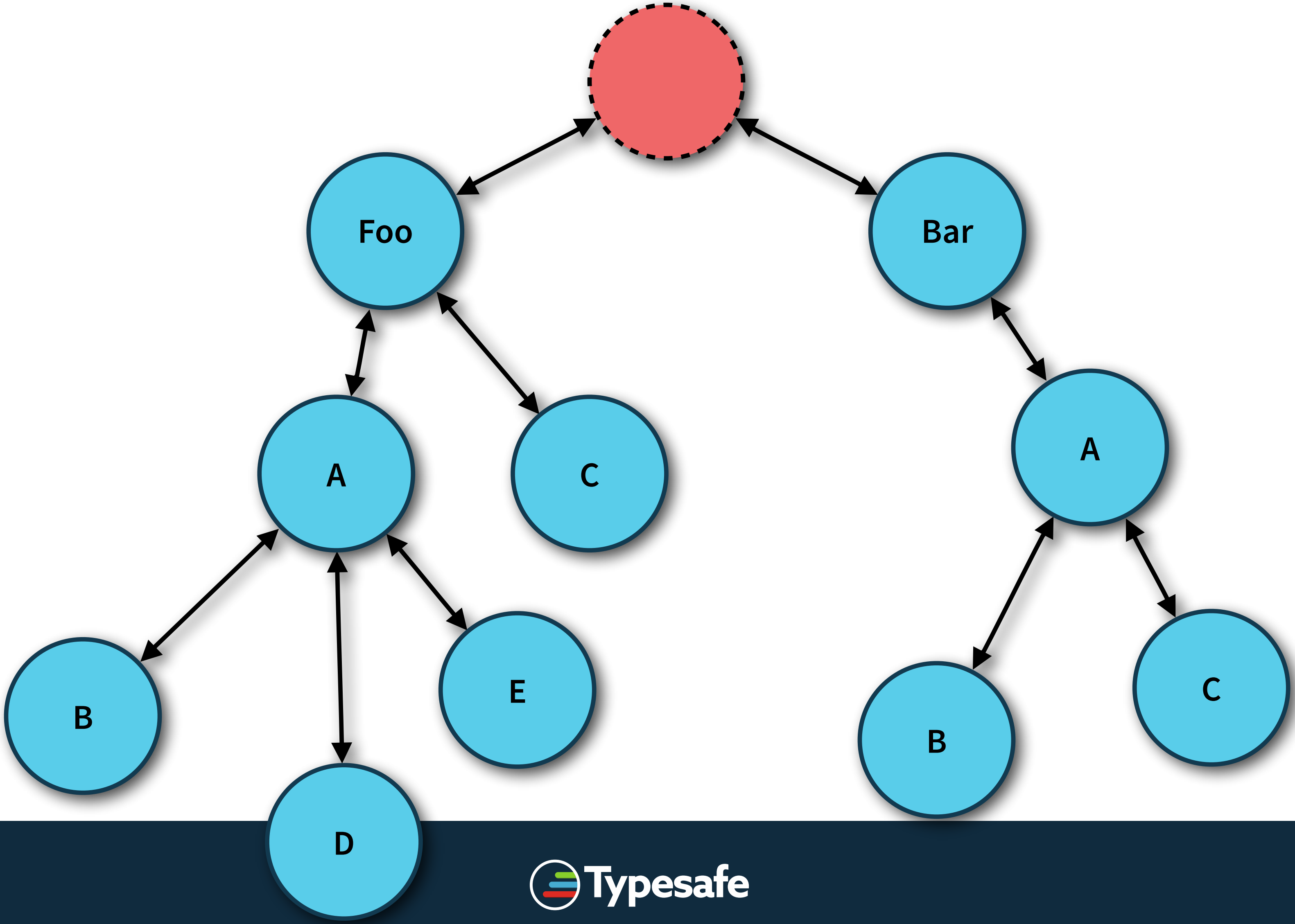


Adrian S. Bruce ©2004 www.artandtechnology.com.au

THE BEATINGS WILL CONTINUE UNTIL MORALE IMPROVES

Supervisor hierarchies

Automatic and mandatory supervision



4. SUPERVISE

Every single actor has a default supervisor strategy.
Which is usually sufficient.
But it can be overridden.

```
class Supervisor extends UntypedActor {
  private SupervisorStrategy strategy = new OneForOneStrategy(
    10, Duration.create(1, TimeUnit.MINUTES),
    DeciderBuilder.
      match(ArithmeticException.class,  e -> resume()).
      match(NullPointerException.class, e -> restart()).
      matchAny(                         e -> escalate()).
      build());

  @Override public SupervisorStrategy supervisorStrategy() {
    return strategy;
  }
```

Monitor through Death Watch

```
public class WatchA                  bstractActor {
  final ActorRef ch                  ).actorOf(Props.empty(), "child");

  public WatchActor() {
    context().watch(child);

    receive(ReceiveBuilder.
      match(Terminated.class,
            t -> t.actor().equals(child),
            t -> {
              … // handle termination
      }).build()
    );
  }
}
```

Create a child actor

Watch it

Handle termination message

Define a router

```
ActorRef router = context().actorOf(
  new RoundRobinPool(5).props(Props.create(Worker.class)),
  “router”)
```

…or from config

```
akka.actor.deployment {
  /service/router {
    router = round-robin-pool
    resizer {
      lower-bound = 12
      upper-bound = 15
    }
  }
}
```

Turn on clustering

```
akka {
  actor {
    provider = "akka.cluster.ClusterActorRefProvider"
    ...
  }

  cluster {
    seed-nodes = [
      “akka.tcp://ClusterSystem@127.0.0.1:2551",
      “akka.tcp://ClusterSystem@127.0.0.1:2552"
    ]

    auto-down = off
  }
}
```

Use clustered routers

Or perhaps use an AdaptiveLoadBalancingPool

```
akka.actor.deployment {
  /service/master {
    router = consistent-hashing-pool
    nr-of-instances = 100

    cluster {
      enabled = on
      max-nr-of-instances-per-node = 3
      allow-local-routees = on
      use-role = compute
    }
  }
}
```

Use clustered pub-sub

```
class Publisher extends Actor {
  val mediator =
    DistributedPubSubExtension(context.system).mediator

  def receive = {
    case in: String =>
      mediator ! Publish("content", in.toUpperCase)
  }
}
```

Questions?