Java Actoren

Contents

[Actoren 2](#_Toc68456827)

[Actor Model 2](#_Toc68456828)

[Actor 2](#_Toc68456829)

[Messages 2](#_Toc68456830)

[Akka 4](#_Toc68456831)

[Wichtige Momente 4](#_Toc68456832)

[Aktorsystem estellen 4](#_Toc68456833)

[Aktor Class erstllen 4](#_Toc68456834)

[Aktor ohne Parameter erstellen 4](#_Toc68456835)

[Aktor mit Parameter erstellen 4](#_Toc68456836)

[Child Actor mit context() erstellen 4](#_Toc68456837)

[Aktor createReceive() example 5](#_Toc68456838)

[match() 5](#_Toc68456839)

[matchAny() 5](#_Toc68456840)

[unhandled() 5](#_Toc68456841)

[tell() 6](#_Toc68456842)

[sender() 6](#_Toc68456843)

[self() 6](#_Toc68456844)

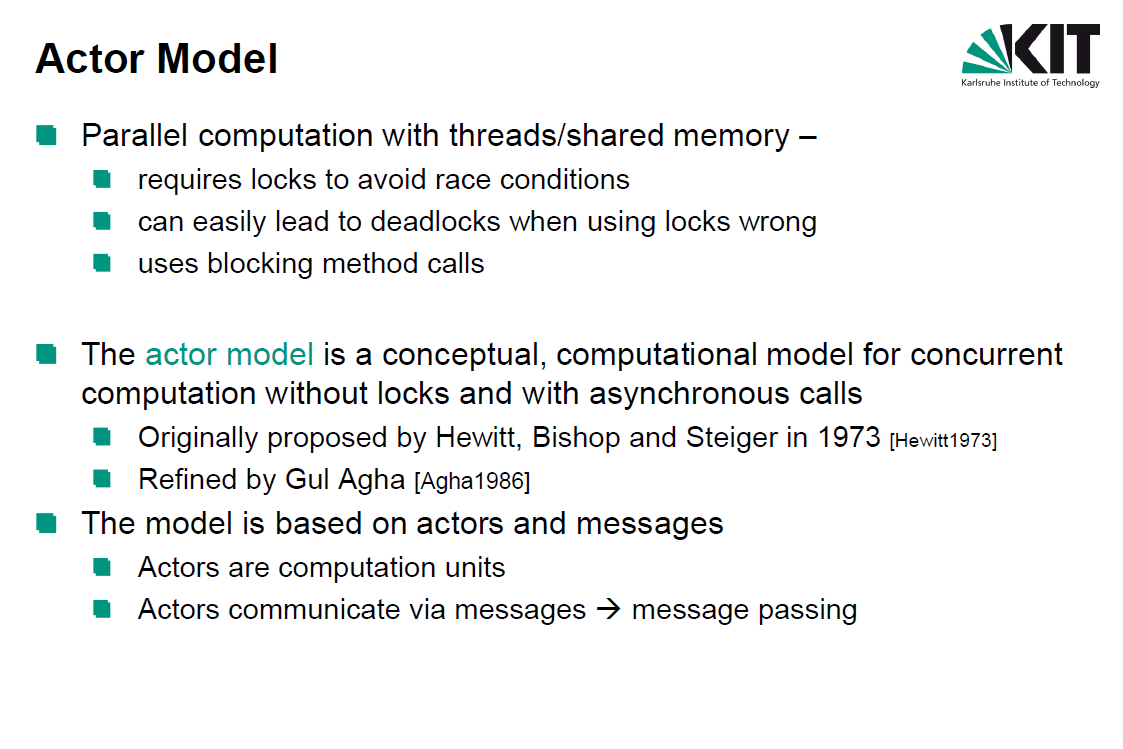
[Alle Objekte sind Aktoren 6](#_Toc68456845)

[Folien 7](#_Toc68456846)

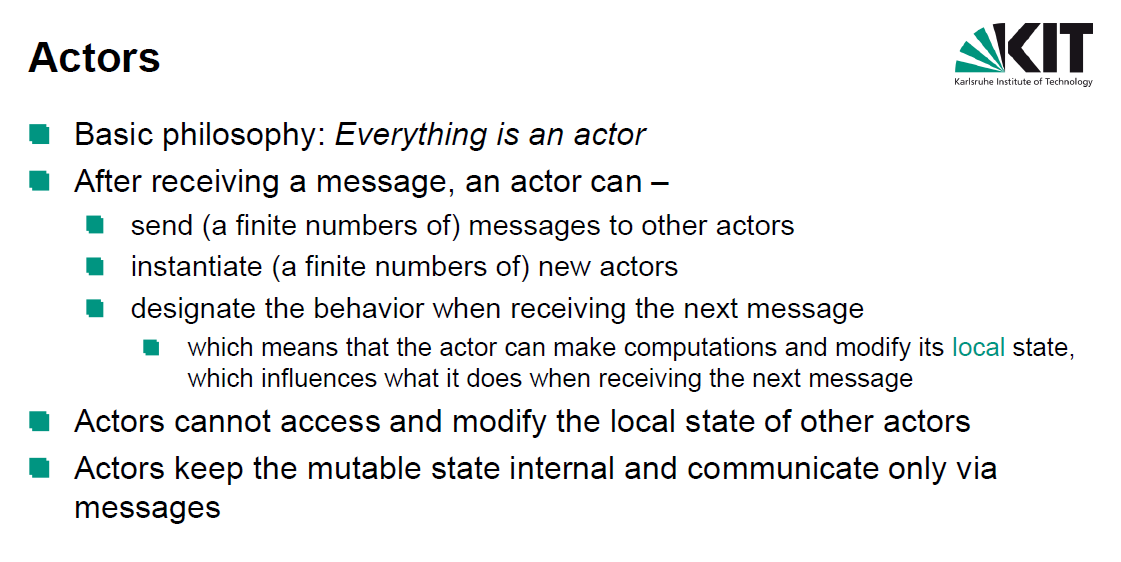
[Aufgaben 11](#_Toc68456847)

# Actoren

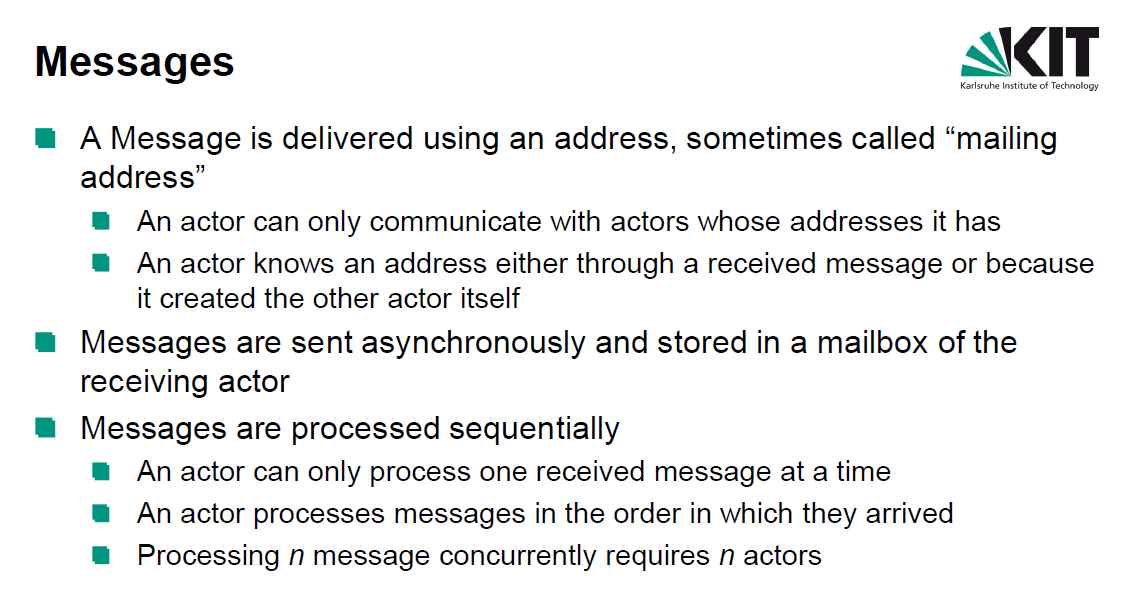
## Actor Model

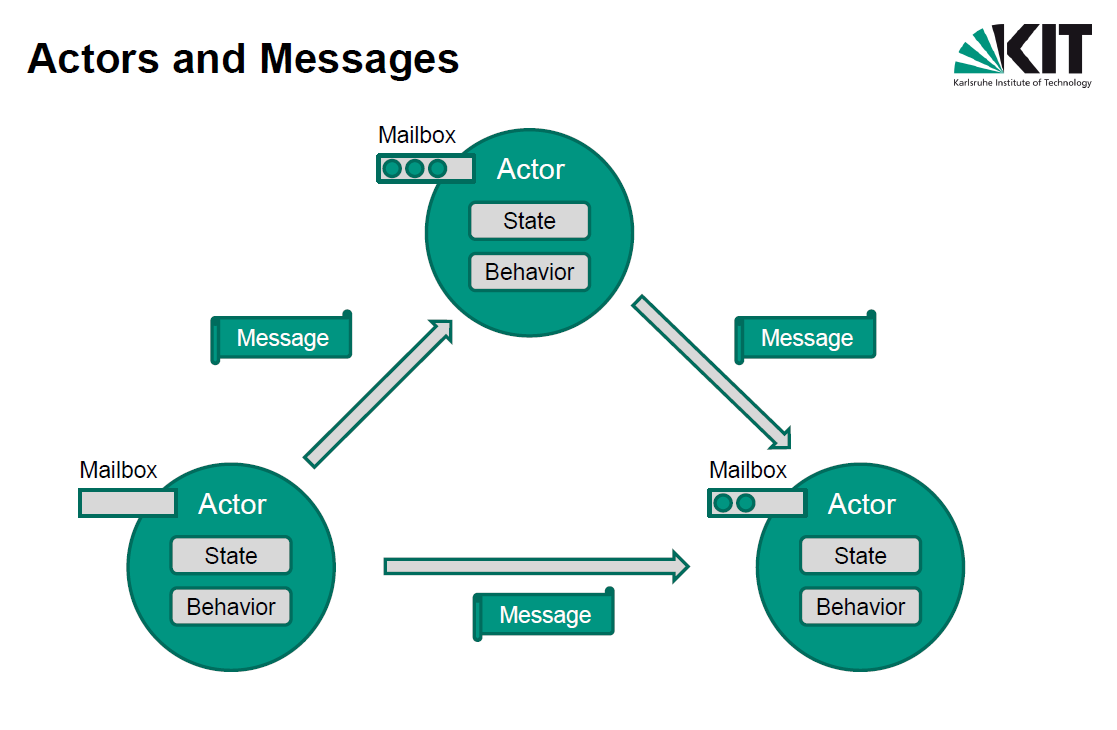


## Actor



# Messages





# Akka

## Wichtige Momente

### Aktorsystem estellen

ActorSystem actorSystem = ActorSystem.create("TeaseLisa");

### Aktor Class erstllen

extends **AbstractActor**, implementiere methode **public Receive createReceive()**

public class PingPong extends AbstractActor {  
 ActorRef partner;  
 String name;  
  
 public PingPong(ActorRef sendTo, String name) {  
 this.partner = sendTo;  
 this.name = name;  
 }

@Override  
public Receive createReceive() {  
 return receiveBuilder()  
 .match(Integer.class, message -> beat(message))  
 .build();  
}

### Aktor ohne Parameter erstellen

ActorRef lisa = actorSystem.actorOf(Props.create(Kid.class));

### Aktor mit Parameter erstellen

ActorRef philosoher = actorSystem.actorOf(  
 Props.create(Philosopher.class, table, name));

Struktur: Props.create(class, args);

### Child Actor mit context() erstellen

Man verwendet getContext().getSystem(), um actorSystem zu finden

public class Kid extends AbstractActor {  
 private int cursed = 0;  
 private ActorRef mommy;  
  
 public Kid() {  
 mommy = getContext().getSystem()

.actorOf(Props.create(Mommy.class));  
 }

### Aktor createReceive() example

Falls message ein String ist und message==“printHello“ ist, wird „Hello World!“ gedruckt

Otherwise unhandled(message)



### match()

Nur dann, wenn class passt und bedingung erfüllt (optional)

match(class, bedingung, aktion);

match(class, aktion);

### matchAny()

Beliebige message.

mathcAny(aktion);

### unhandled()

Mache nichts, default case

### tell()

sender sends a message to actor:

actor.tell(message, sender);

ping.tell(0, ActorRef.noSender());

message ist 0, sender() in ping gibt Null-Sender zurück

ping.tell(“abc”, pong);

message ist „abc“, sender() in pong gibt ActorRef von pong zurück

### sender()

Mit sender kann man ActorRef auf Sender vom Message bekommen, falls es beim tell() gesetzt wurde, ActorRef.noSender, falls es nicht gesetzt wurde.

if (partner != null) {  
 partner.tell(message, self());  
} else {  
 sender().tell(message, self());  
}

### self()

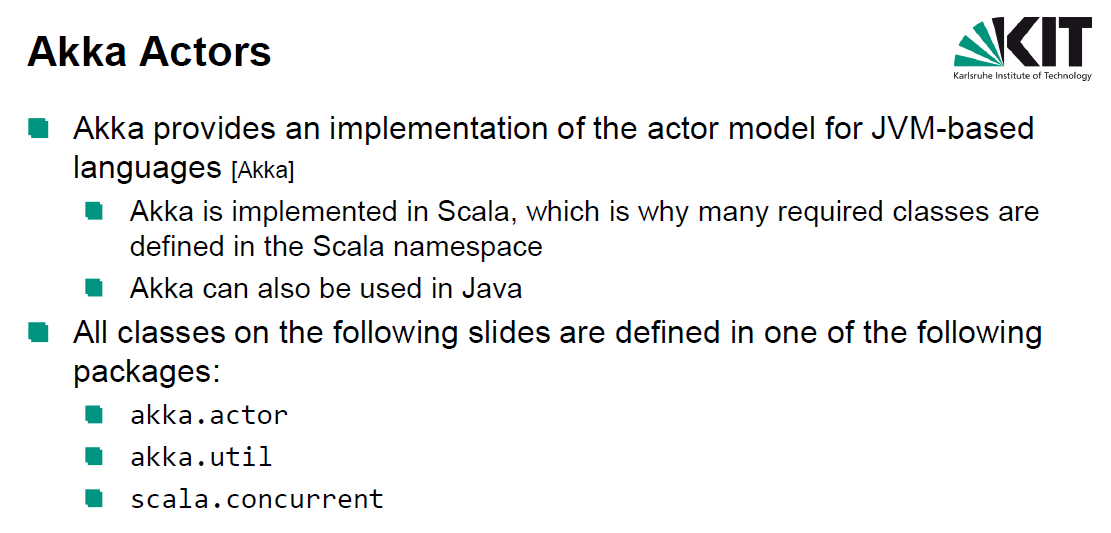
“”returns ActorRef of current Actor””

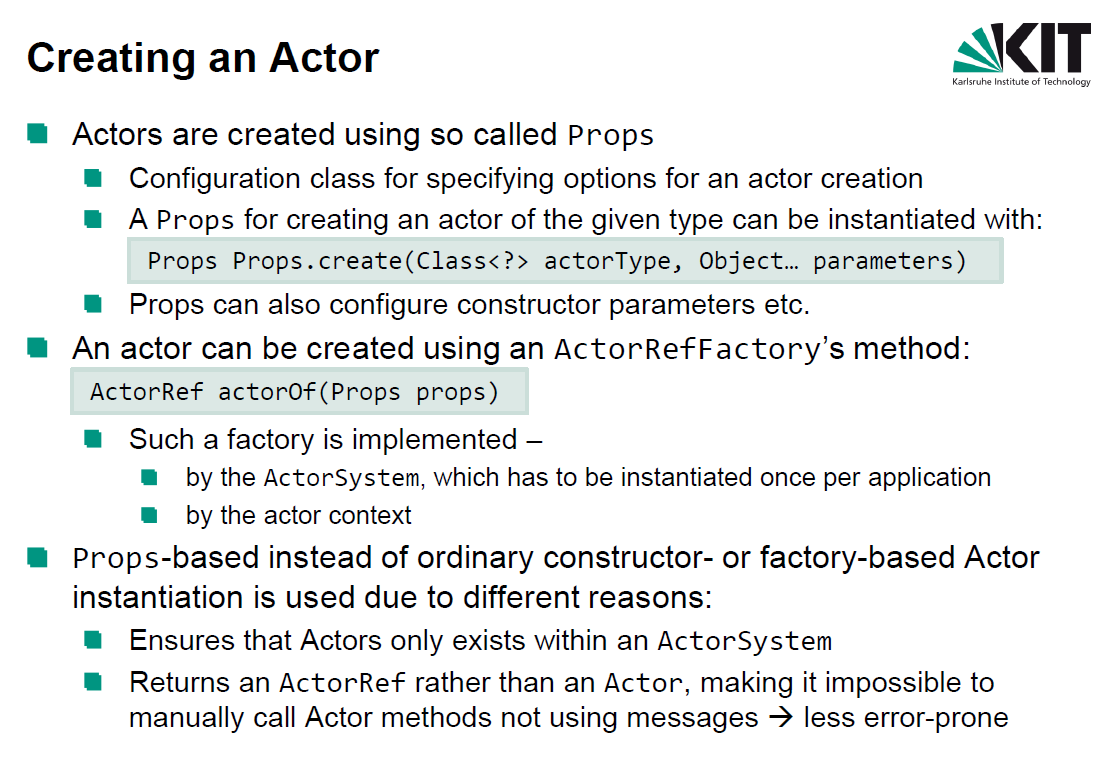
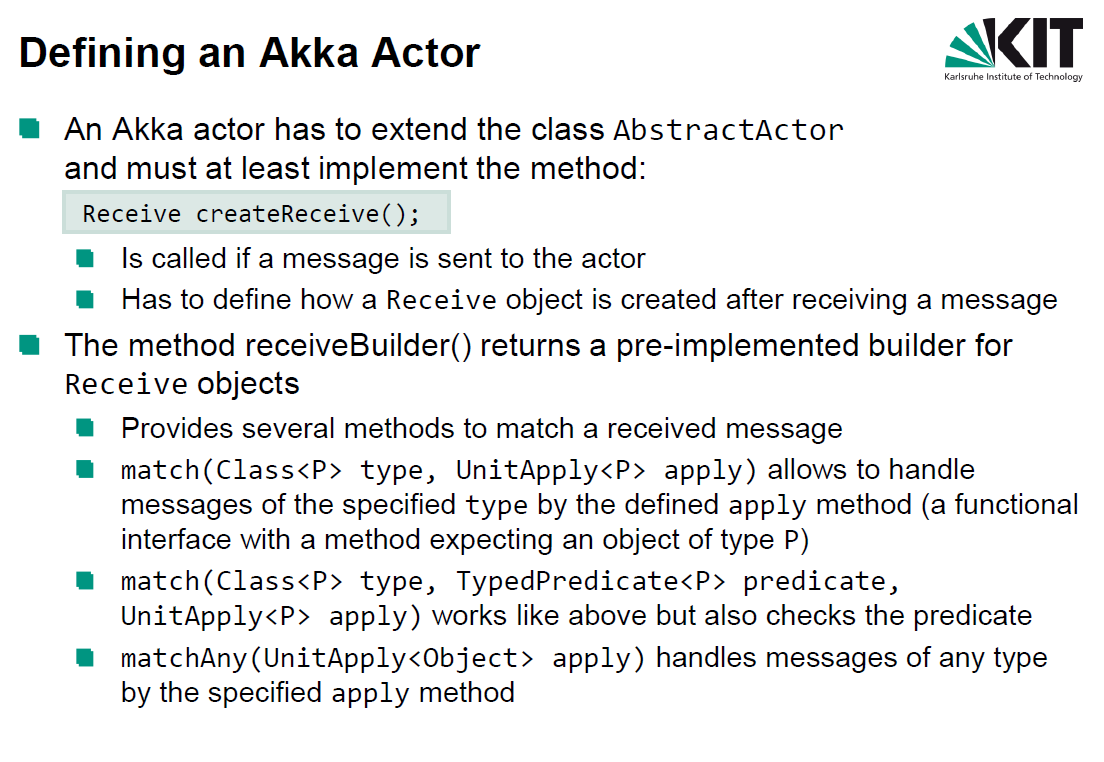
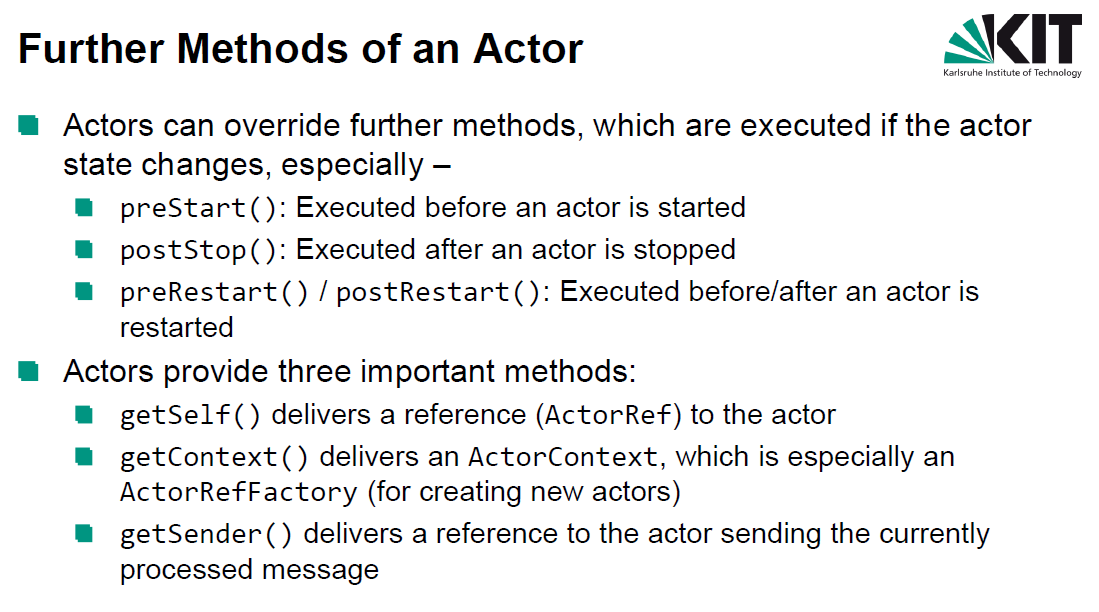
Kann man für tell verwenden

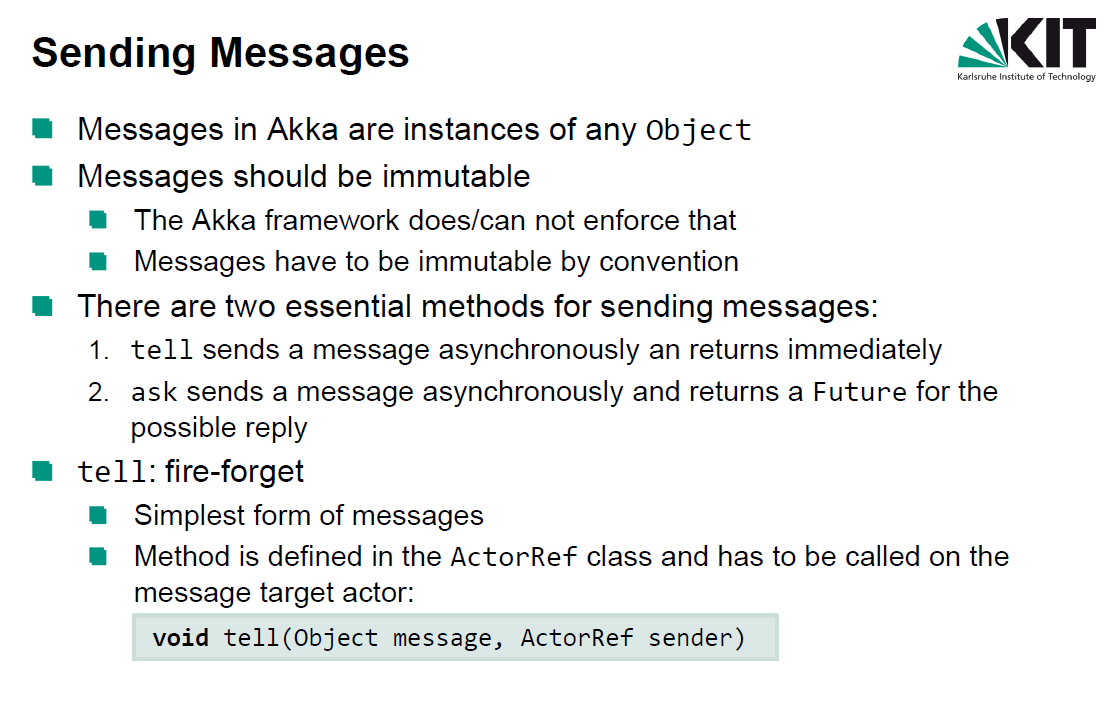
### Alle Objekte sind Aktoren

Speischende Philosopher: Alle Philosopher sind Aktoren und Tisch ist auch ein Aktor

## Folien







# Aufgaben