Akka:

Simpler Concurrency, Scalability & Fault-tolerance through Actors

Jonas Bonér
@jboner

We believe that...

- Writing correct concurrent applications is **too hard**

- Scaling out applications is **too hard**

- Writing highly fault-tolerant applications is **too hard**

It doesn't have to be like this

We need to raise the abstraction level

Locks & Threads

are...

...sometimes

plain evil

...but always the

wrong default

"Threads are to Concurrency as Witchcraft is to Physics"

"Hanging by a thread is the punishment for Shared State Concurrency"

- Gilad Bracha

Actors

one tool in the toolbox

Actor Model of Concurrency

- Implements Message-Passing Concurrency
- Share NOTHING
- Isolated lightweight processes
- Communicates through messages
- Asynchronous and non-blocking
- Each actor has a mailbox (message queue)

Actor Model of Concurrency

- Easier to reason about
- Raised abstraction level
- Easier to avoid
  - –Race conditions
  - –Deadlocks
  - –Starvation
  - –Live locks

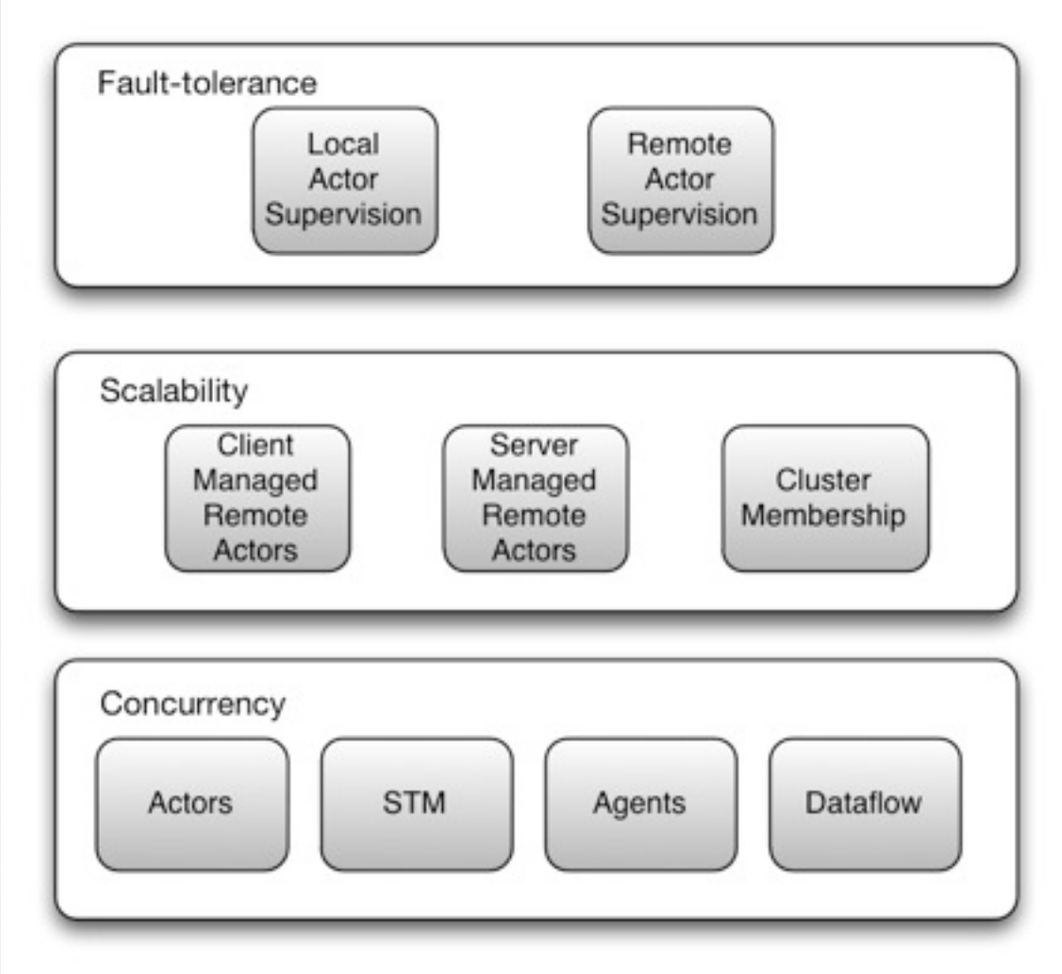
Introducing

STM Actors Agents

Dataflow Distributed Open Source

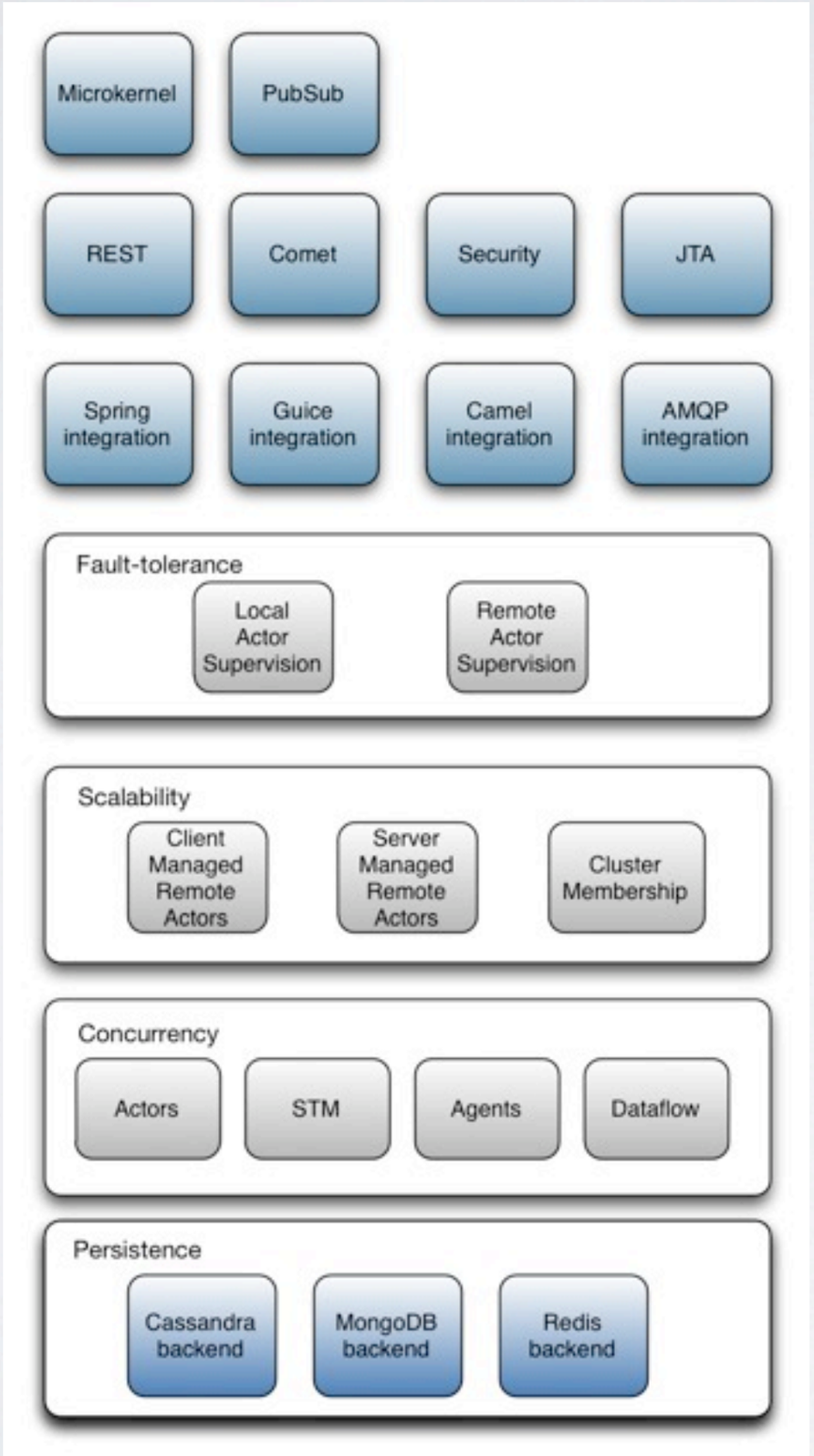
RESTful Secure Persistent

ARCHITECTURE



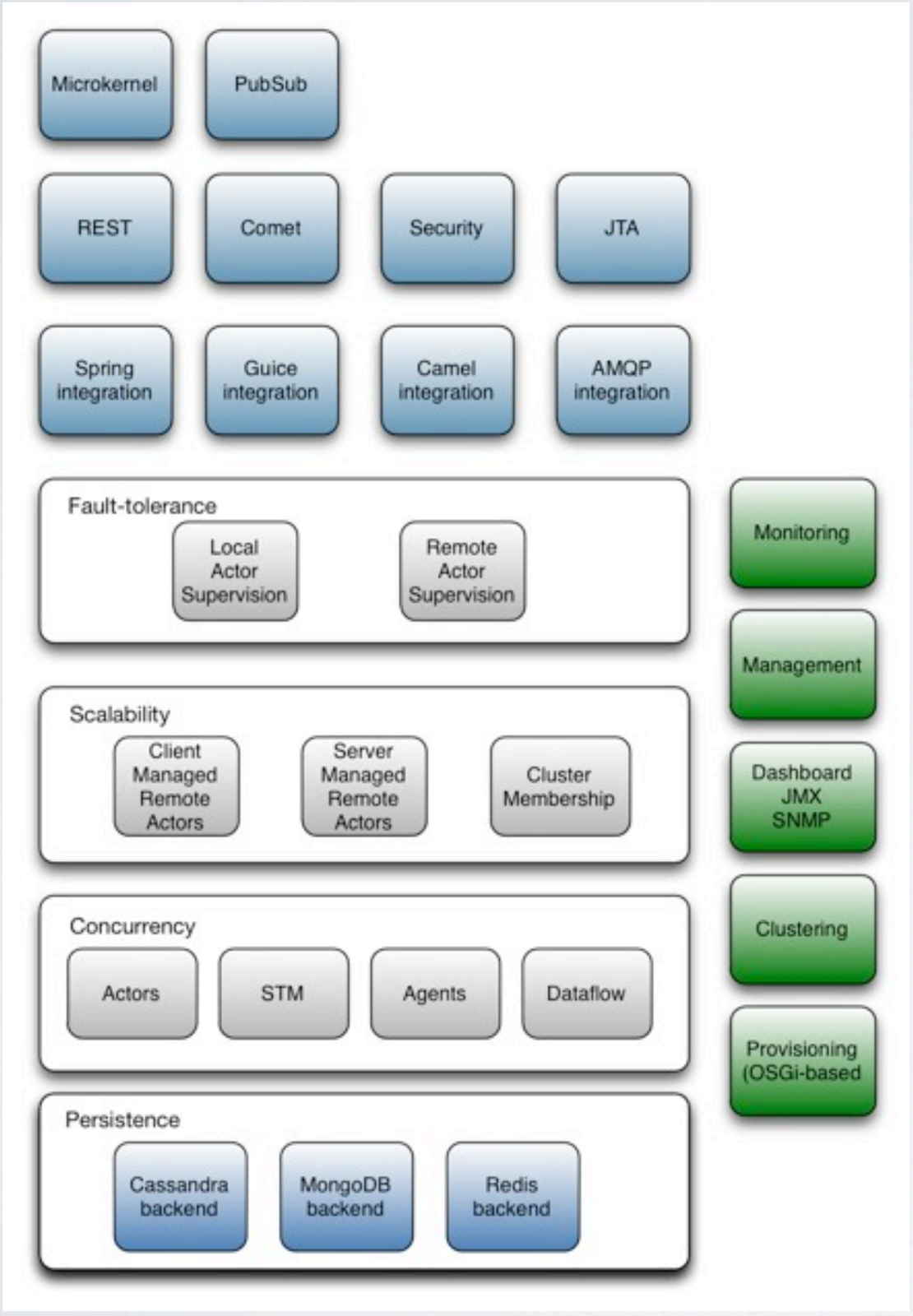
CORE SERVICES

ARCHITECTURE



ADD-ON MODULES

ARCHITECTURE



ENTERPRISE MODULES

Akka Actors

Two different models

- Thread-based
- Event-based
  - Very lightweight (600 bytes per actor)
  - Can easily create millions on a single workstation (6.5 million on 4 G RAM)
  - Does not consume a thread

Actors

```
case object Tick

class Counter extends Actor {
  private var counter = 0

  def receive = {
    case Tick =>
      counter += 1
      println(counter)
  }
}
```

Create Actors

```
import Actor._

val counter = actorOf[Counter]
```

counter is an ActorRef

Create Actors

```
val actor = actorOf(new MyActor(..))
```

create actor with constructor arguments

Start actors

```
val counter = actorOf[Counter]
counter.start
```

Start actors

```
val counter = actorOf[Counter].start
```

Stop actors

```
val counter = actorOf[Counter].start
counter.stop
```

init & shutdown callbacks

```
class MyActor extends Actor {

  override def init = {
    ... // called after ‘start’
  }

  override def shutdown = {
    ... // called before ‘stop’
  }
}
```

the self reference

```
class RecursiveActor extends Actor {
  private var counter = 0
  self.id = “service:recursive”

  def receive = {
    case Tick =>
      counter += 1
      self ! Tick
  }
}
```

Actors

anonymous

```
val worker = actor {
  case Work(fn) => fn()
}
```

Send: !

```
counter ! Tick
```

fire-forget

Send: !!

```
val result = (actor !! Message).as[String]
```

uses Future under the hood (with time-out)

Send: !!

```
val resultOption = actor !! Message

val result = resultOption
               .getOrElse(defaultResult)

val result = resultOption.getOrElse(
    throw new Exception("Timed out")))
```

returns an Option[ResultType] directly

Send: !!!

```
// returns a future
val future = actor !!! Message
future.await
val result = future.get


...
Futures.awaitOne(List(fut1, fut2, ...))
Futures.awaitAll(List(fut1, fut2, ...))
```

returns the Future directly

Reply

```
class SomeActor extends Actor {
  def receive = {
    case User(name) =>
      // use reply
      self.reply(“Hi ” + name)
  }
}
```

Reply

```
class SomeActor extends Actor {
  def receive = {
    case User(name) =>
      // store away the sender
      // to use later or
      // somewhere else
      ... = self.sender
  }
}
```

Reply

```
class SomeActor extends Actor {
  def receive = {
    case User(name) =>
      // store away the sender future
      // to resolve later or
      // somewhere else
      ... = self.senderFuture
  }
}
```

Good immutable messages

```
// define the case class
case class Register(user: User)

// create and send a new case class message
actor ! Register(user)

// tuples
actor ! (username, password)

// lists
actor ! List(“bill”, “bob”, “alice”)
```

Akka Dispatchers

Dispatchers

```
object Dispatchers {
  def newExecutorBasedEventDrivenDispatcher(name: String)

  def newExecutorBasedEventDrivenWorkStealingDispatcher(name: String)

  def newThreadBasedDispatcher(actor: ActorRef)

  def newReactorBasedThreadPoolEventDrivenDispatcher(name: String)

  def newReactorBasedSingleThreadEventDrivenDispatcher(name: String)
}
```

Set dispatcher

```
class MyActor extends Actor {
  self.dispatcher = Dispatchers
    .newThreadBasedDispatcher(self)

  ...
}

actor.dispatcher = dispatcher // before started
```

EventBasedDispatcher

Fluent DSL

```
val dispatcher =
  Dispatchers.newExecutorBasedEventDrivenDispatcher
dispatcher
    .withNewThreadPoolWithBoundedBlockingQueue(100)
    .setCorePoolSize(16)
    .setMaxPoolSize(128)
    .setKeepAliveTimeInMillis(60000)
    .setRejectionPolicy(new CallerRunsPolicy)
    .buildThreadPool
```

MessageQueues

Unbounded LinkedBlockingQueue

Bounded LinkedBlockingQueue

Bounded ArrayBlockingQueue

Bounded SynchronousQueue

Plus different options per queue

ActorRegistry

```
val actors = ActorRegistry.actors
val actors = ActorRegistry.actorsFor[TYPE]
val actors = ActorRegistry.actorsFor(id)
val actor = ActorRegistry.actorFor(uuid)

ActorRegistry.foreach(fn)
ActorRegistry.shutdownAll
```

Let it crash

fault-tolerance

Stolen from

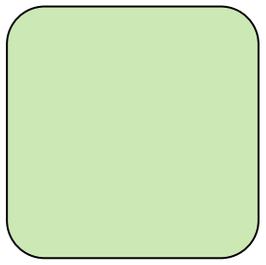
Erlang

9

nines

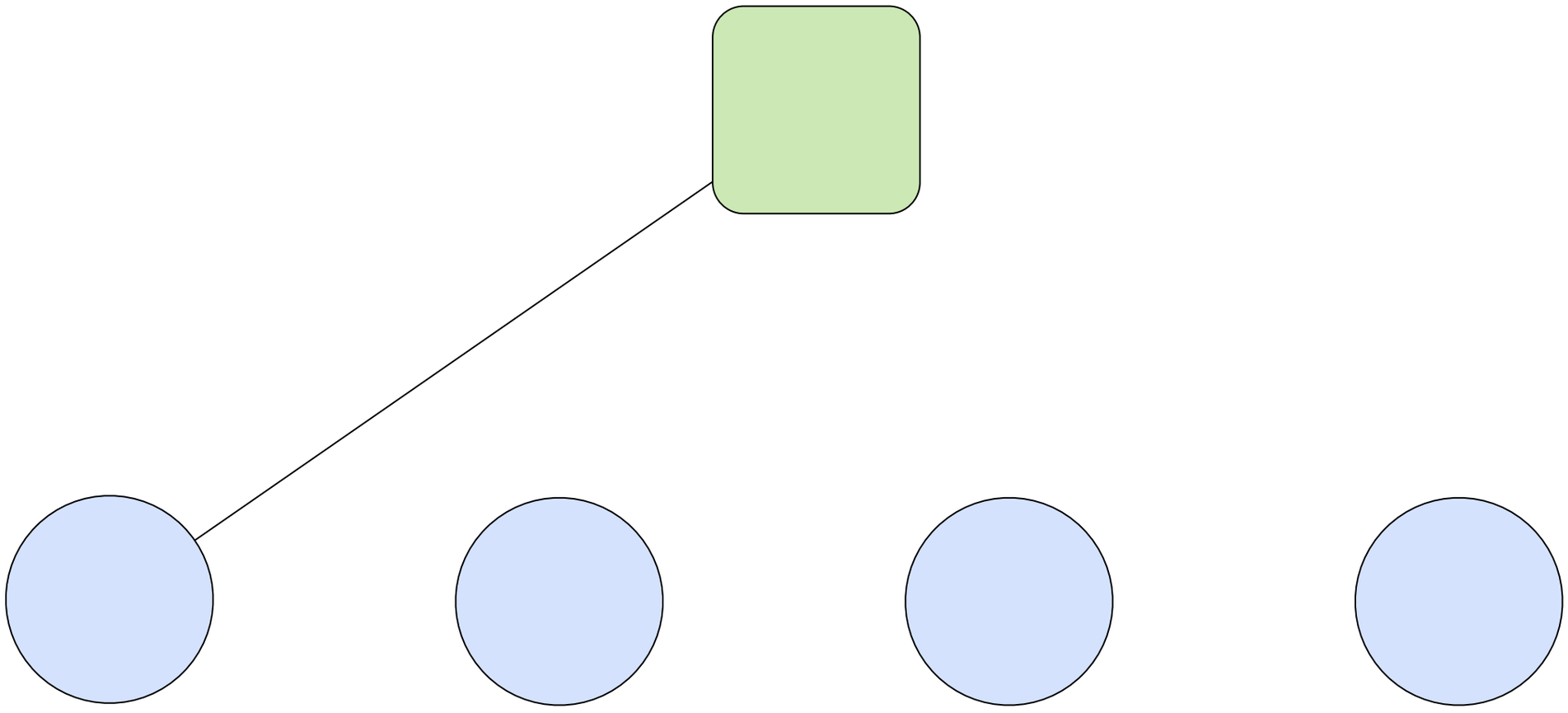
OneForOne

fault handling strategy



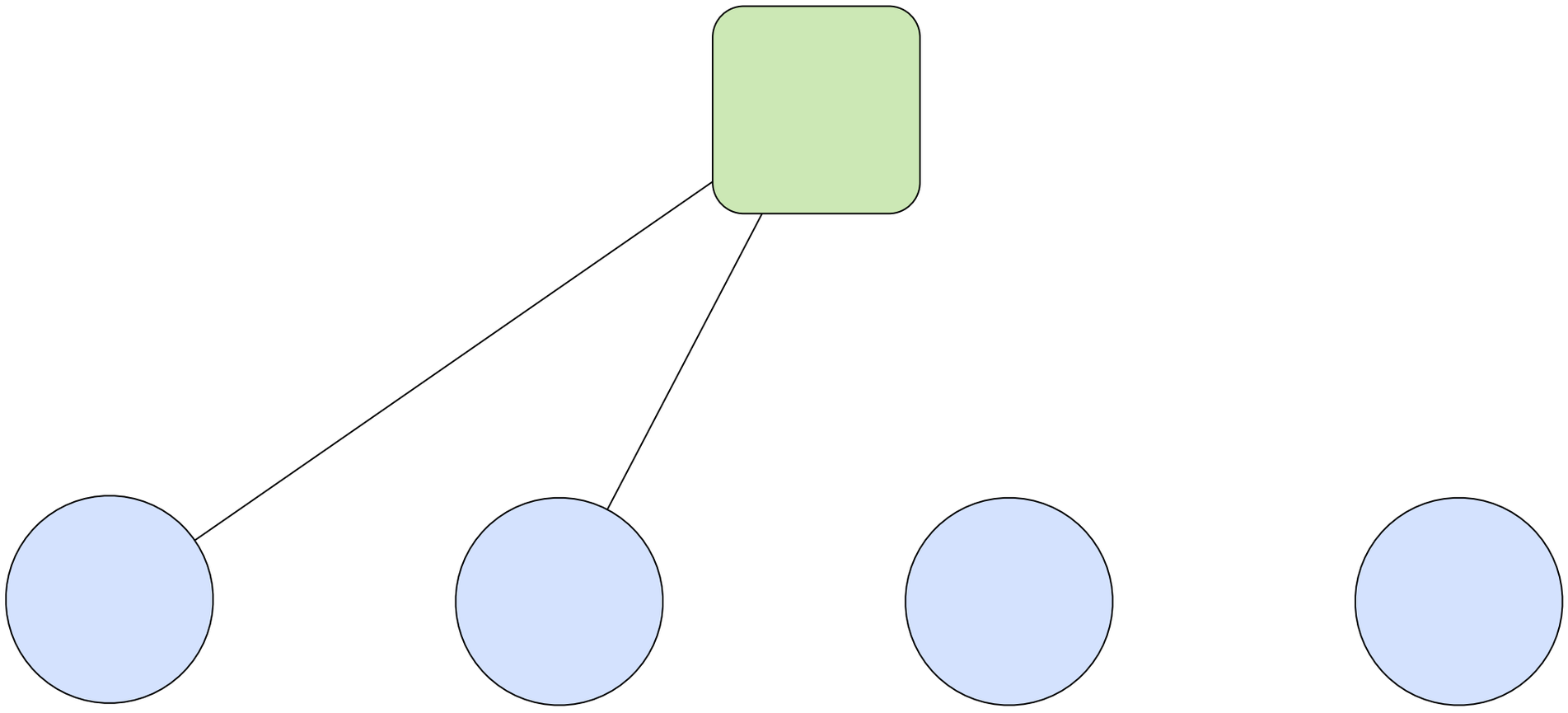
OneForOne

fault handling strategy



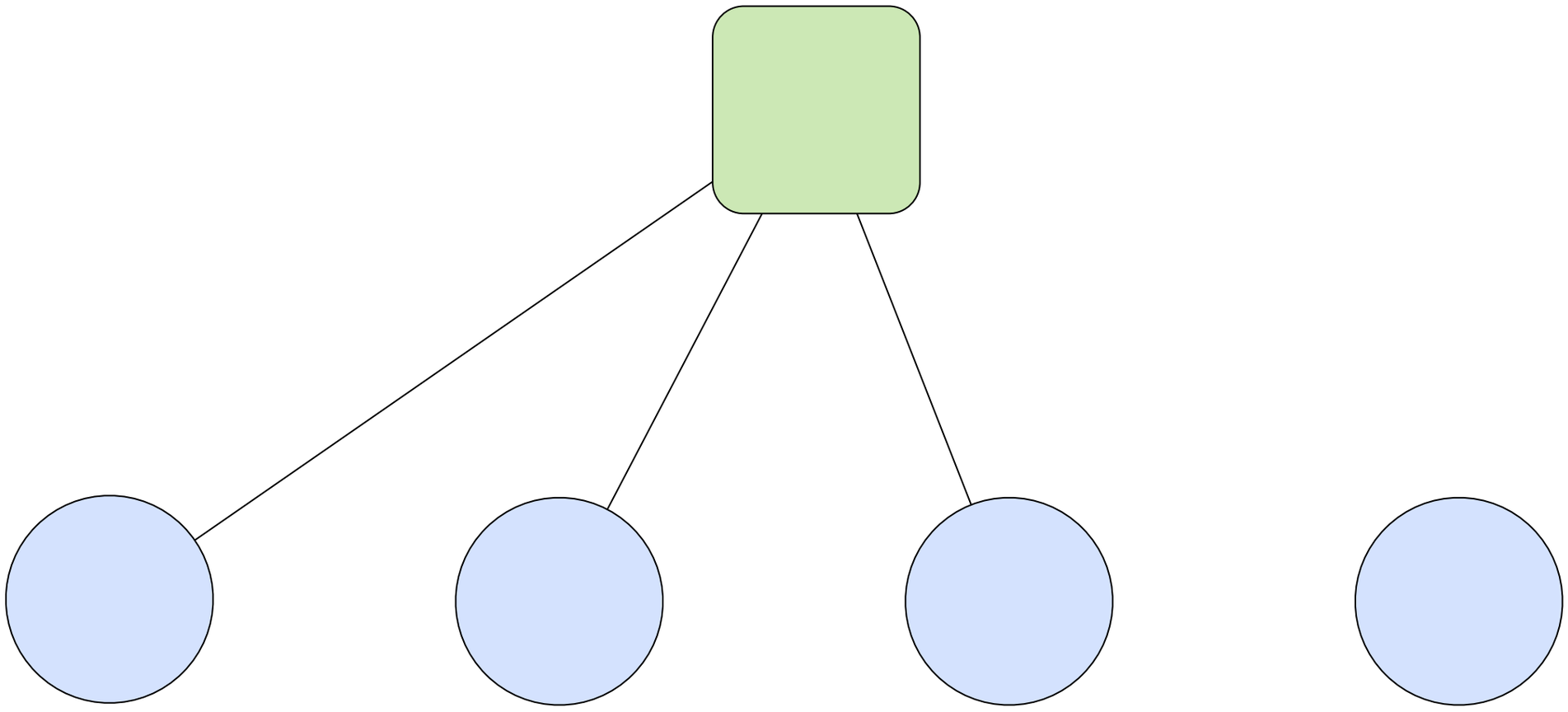
OneForOne

fault handling strategy



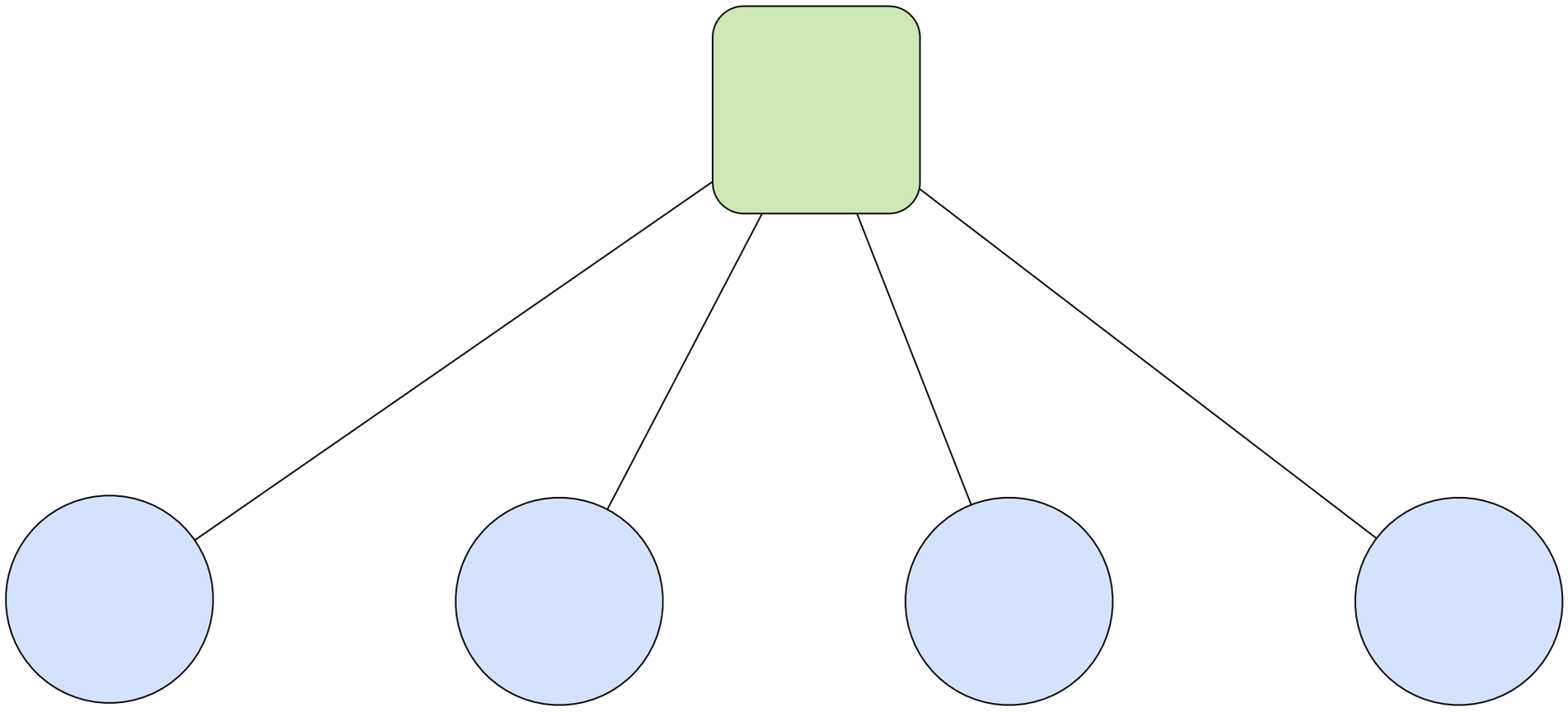
OneForOne

fault handling strategy



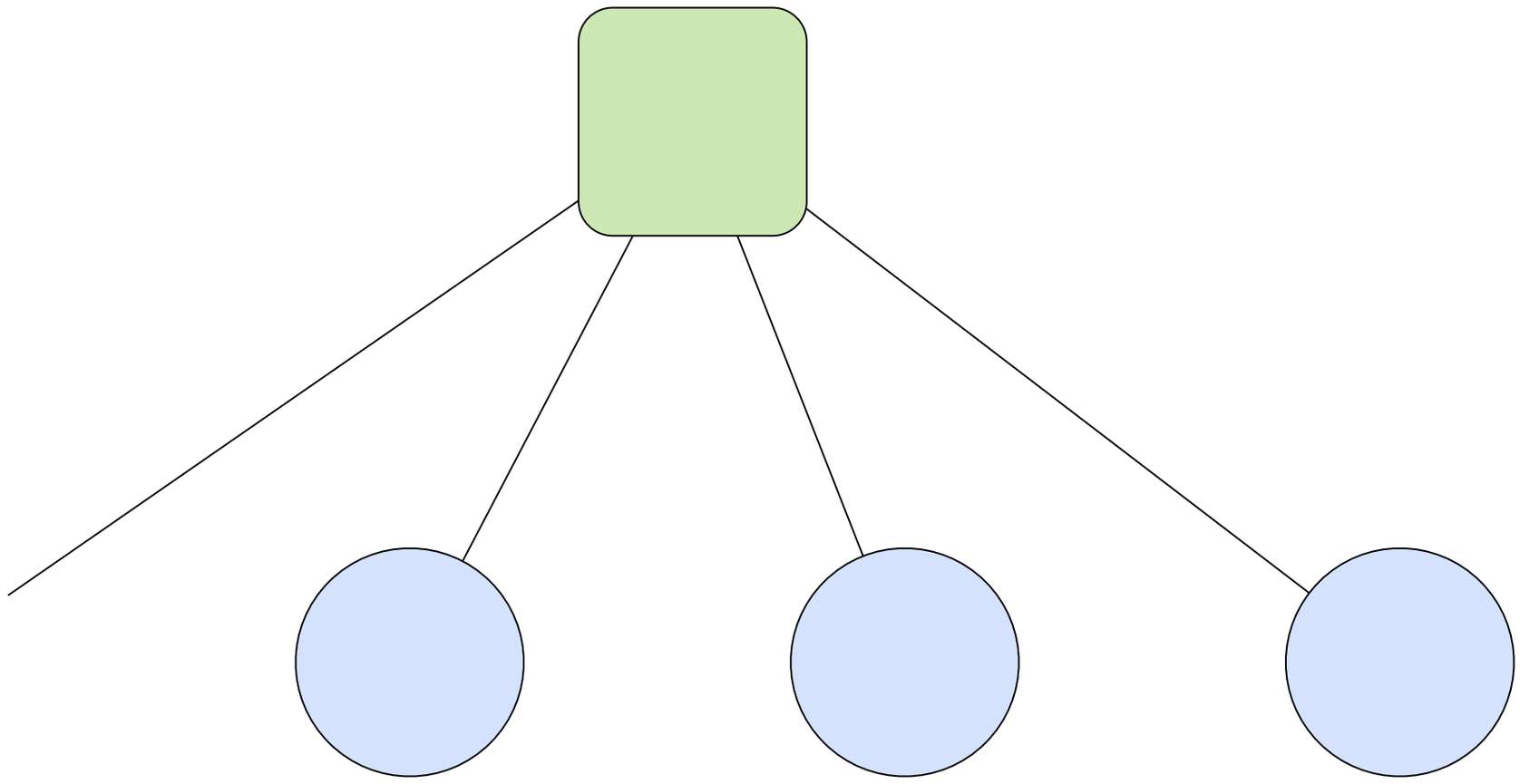
OneForOne

fault handling strategy



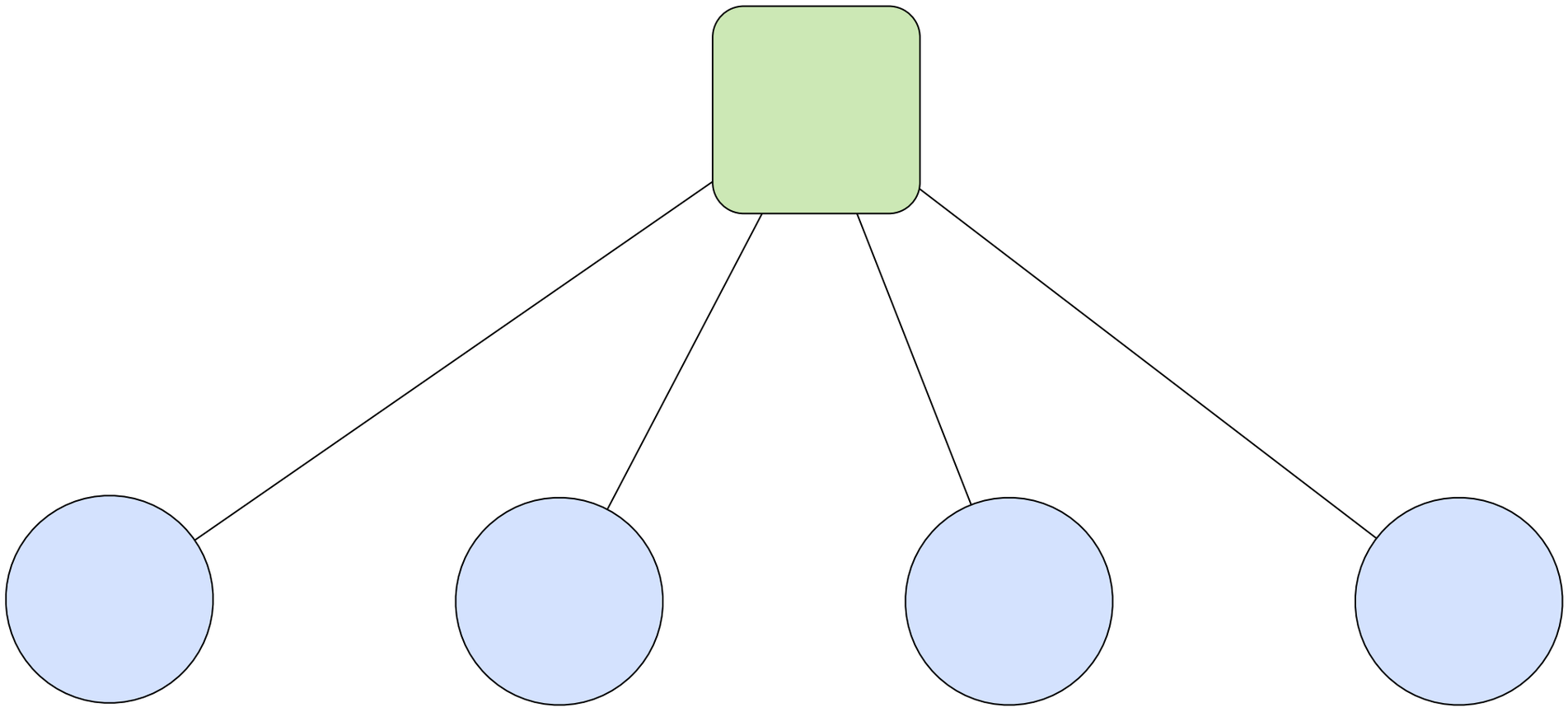
OneForOne

fault handling strategy



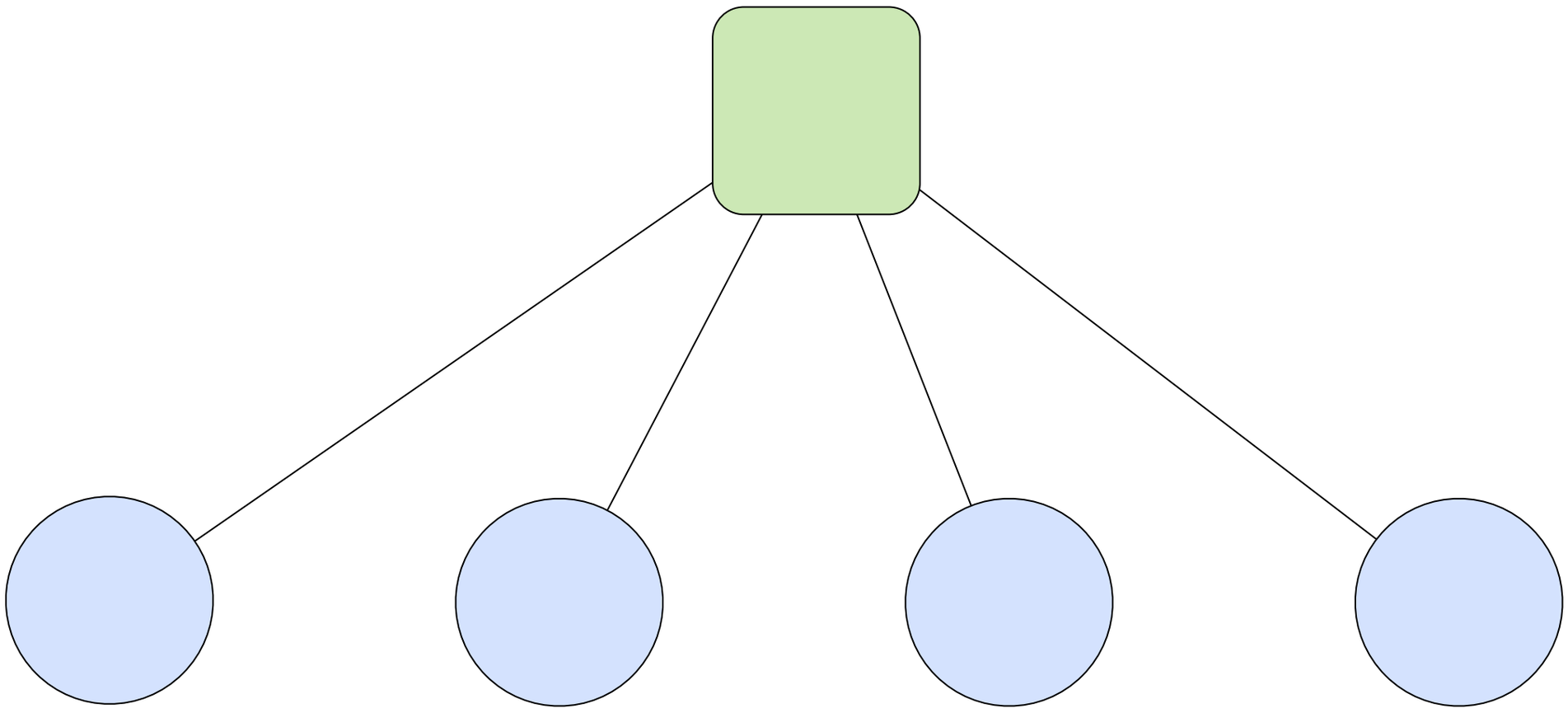
OneForOne

fault handling strategy



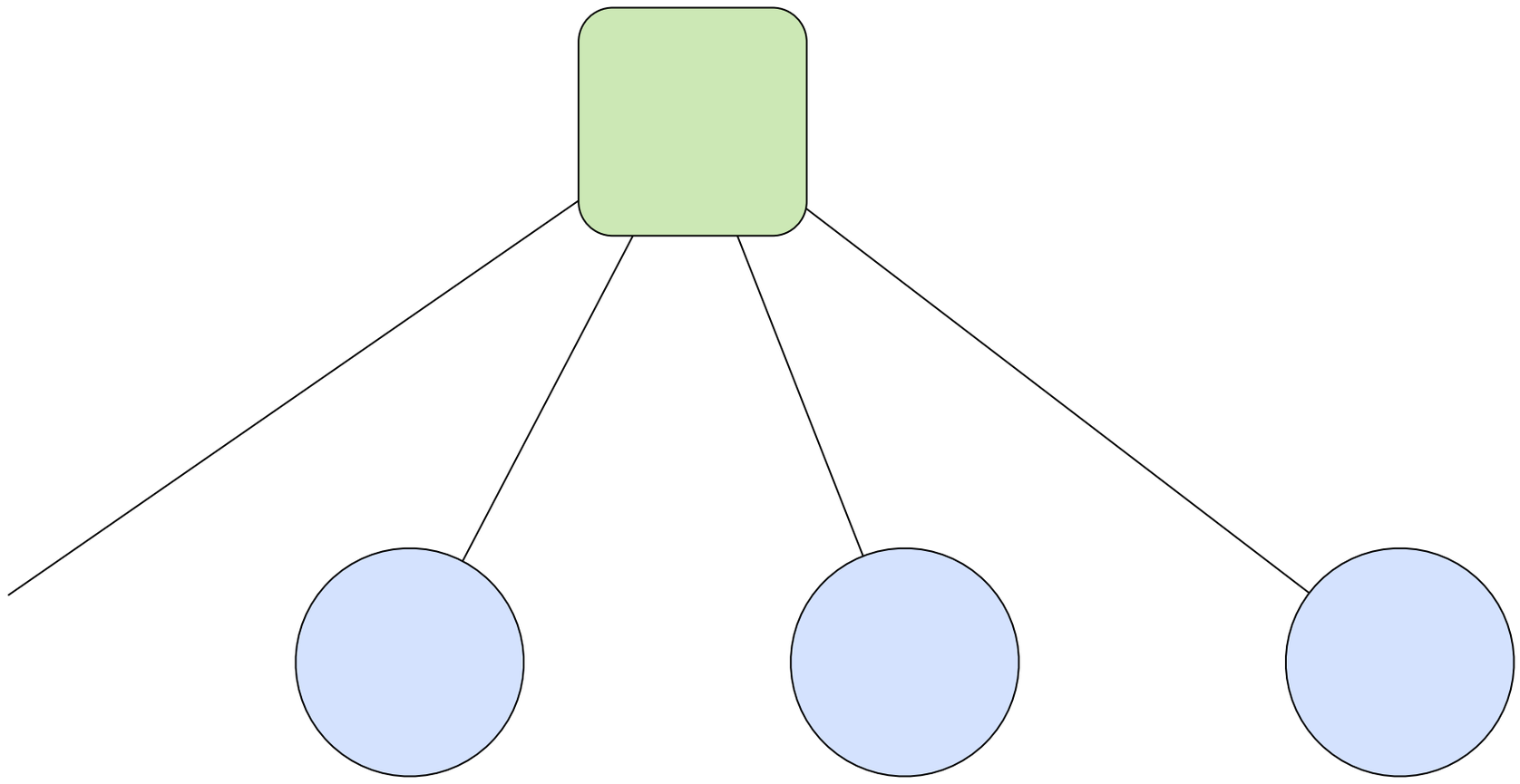
AllForOne

fault handling strategy



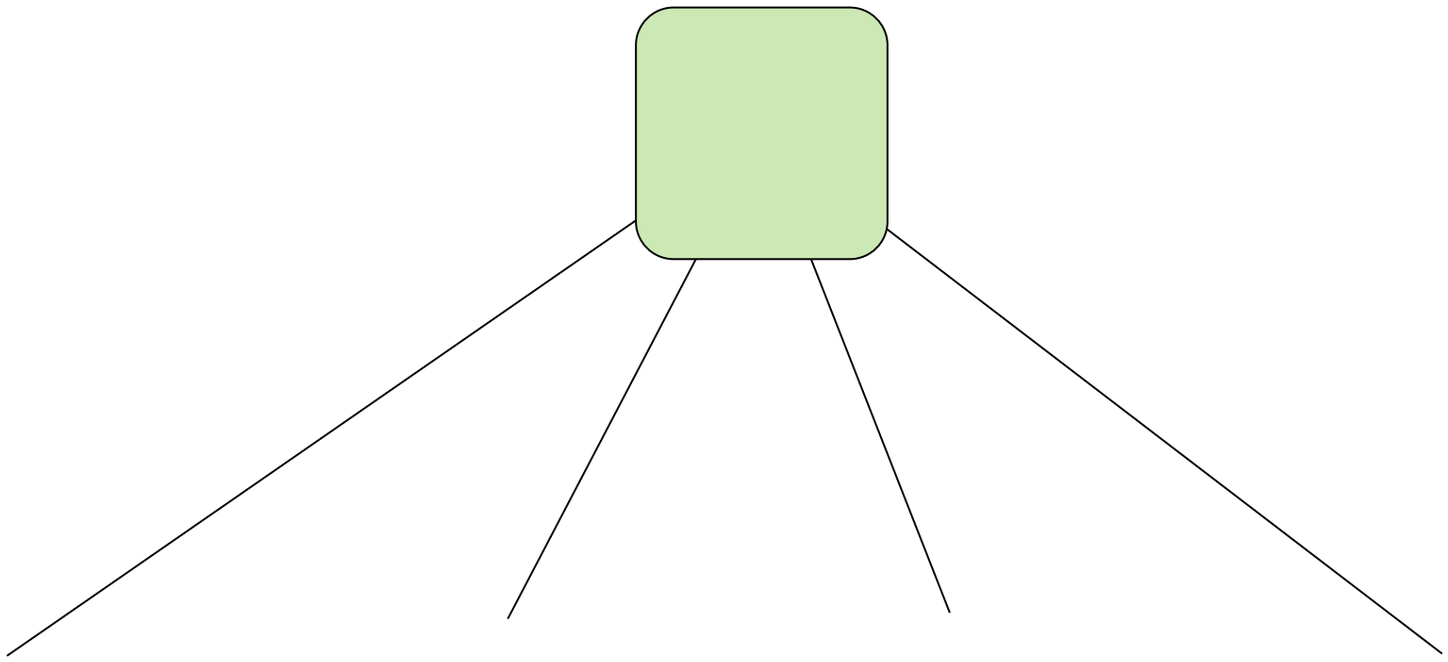
AllForOne

fault handling strategy



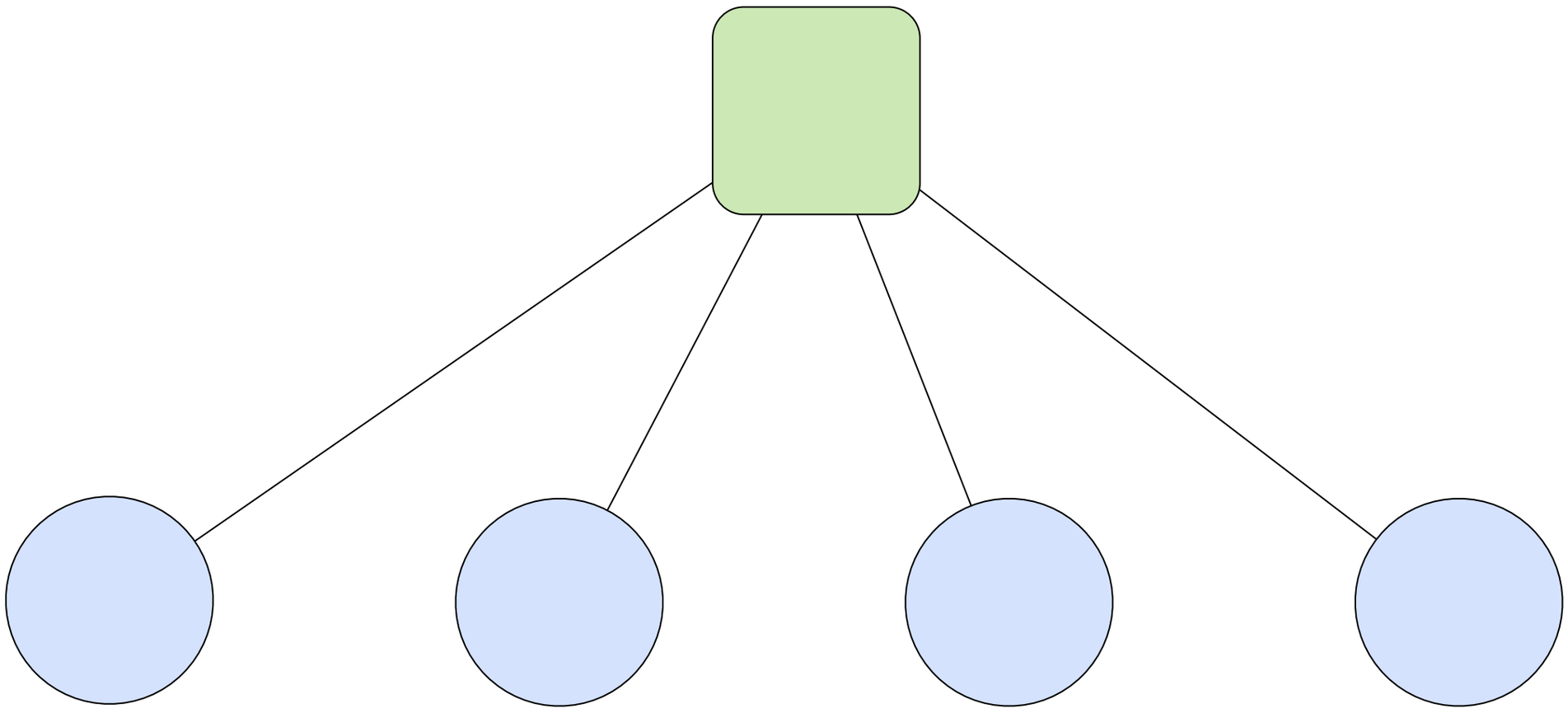
AllForOne

fault handling strategy

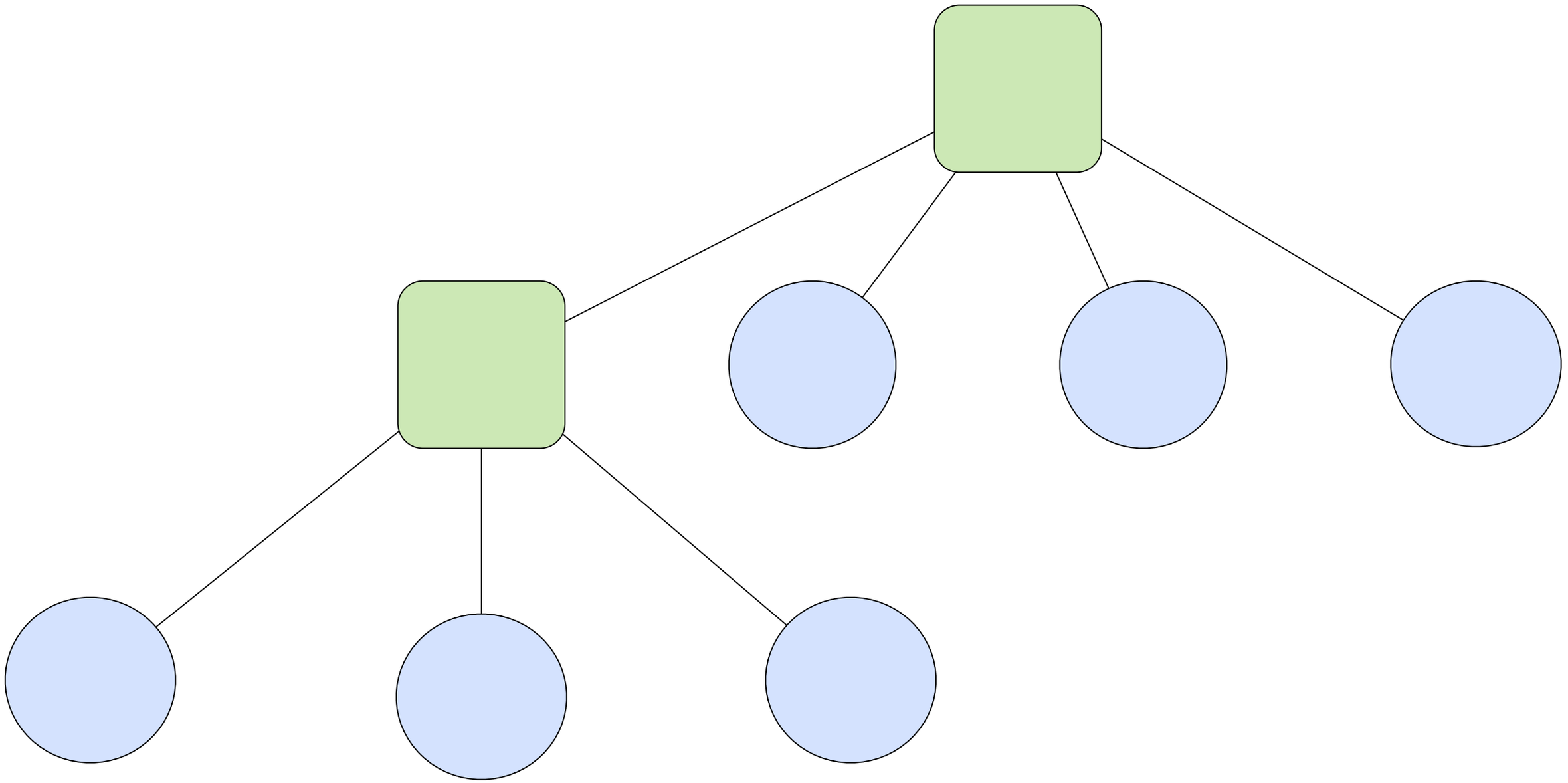


AllForOne

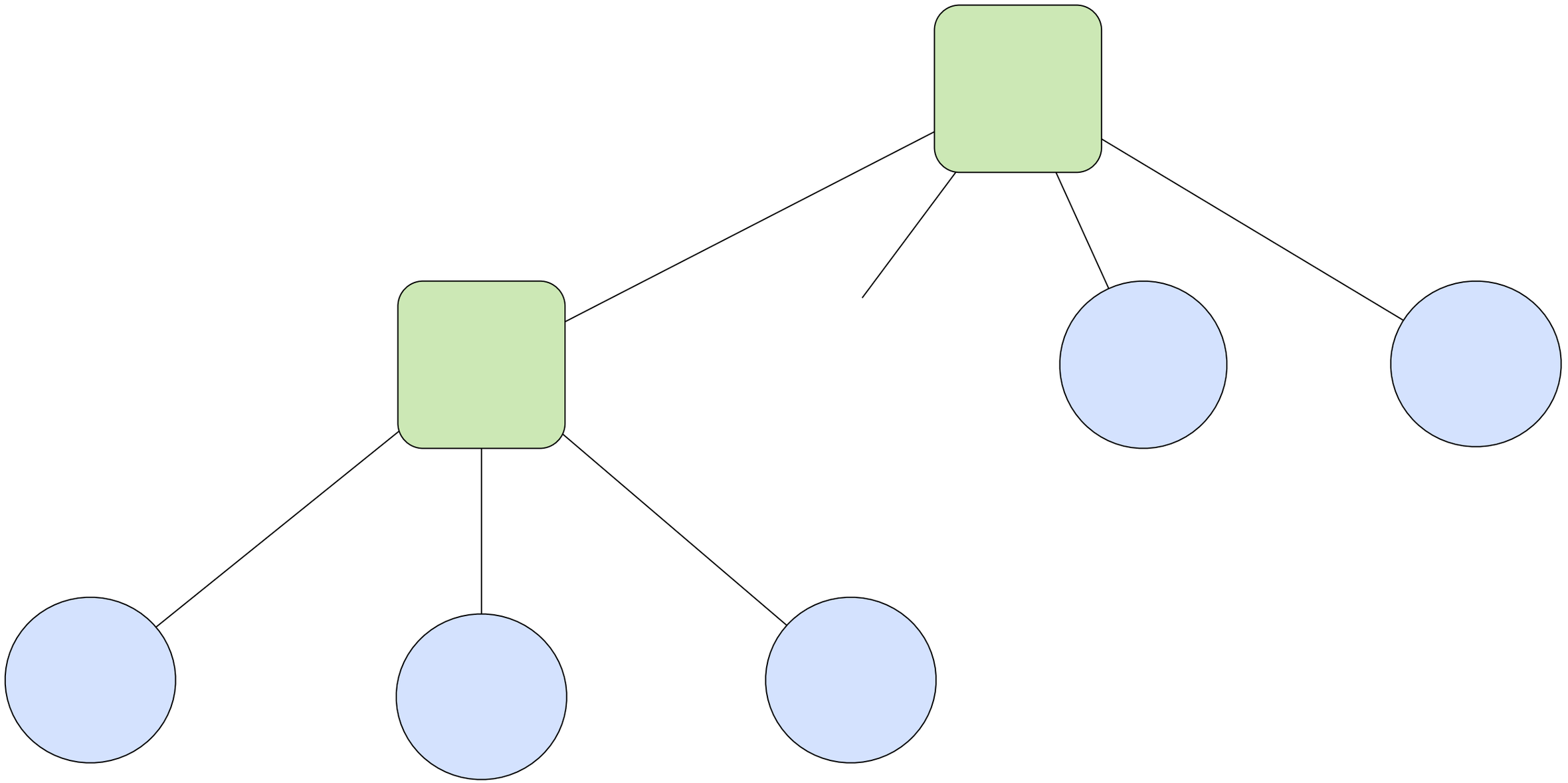
fault handling strategy



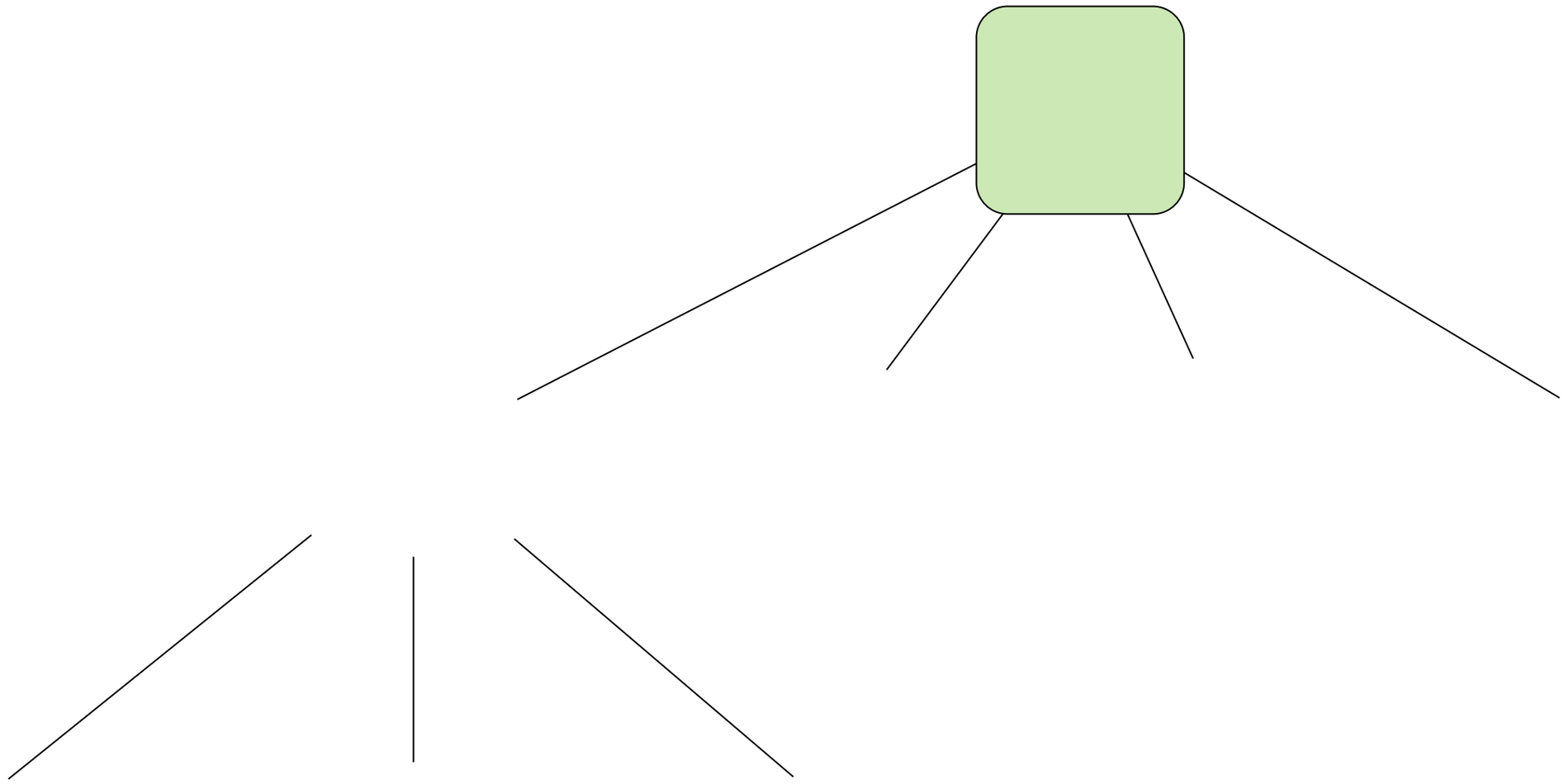
Supervisor hierarchies



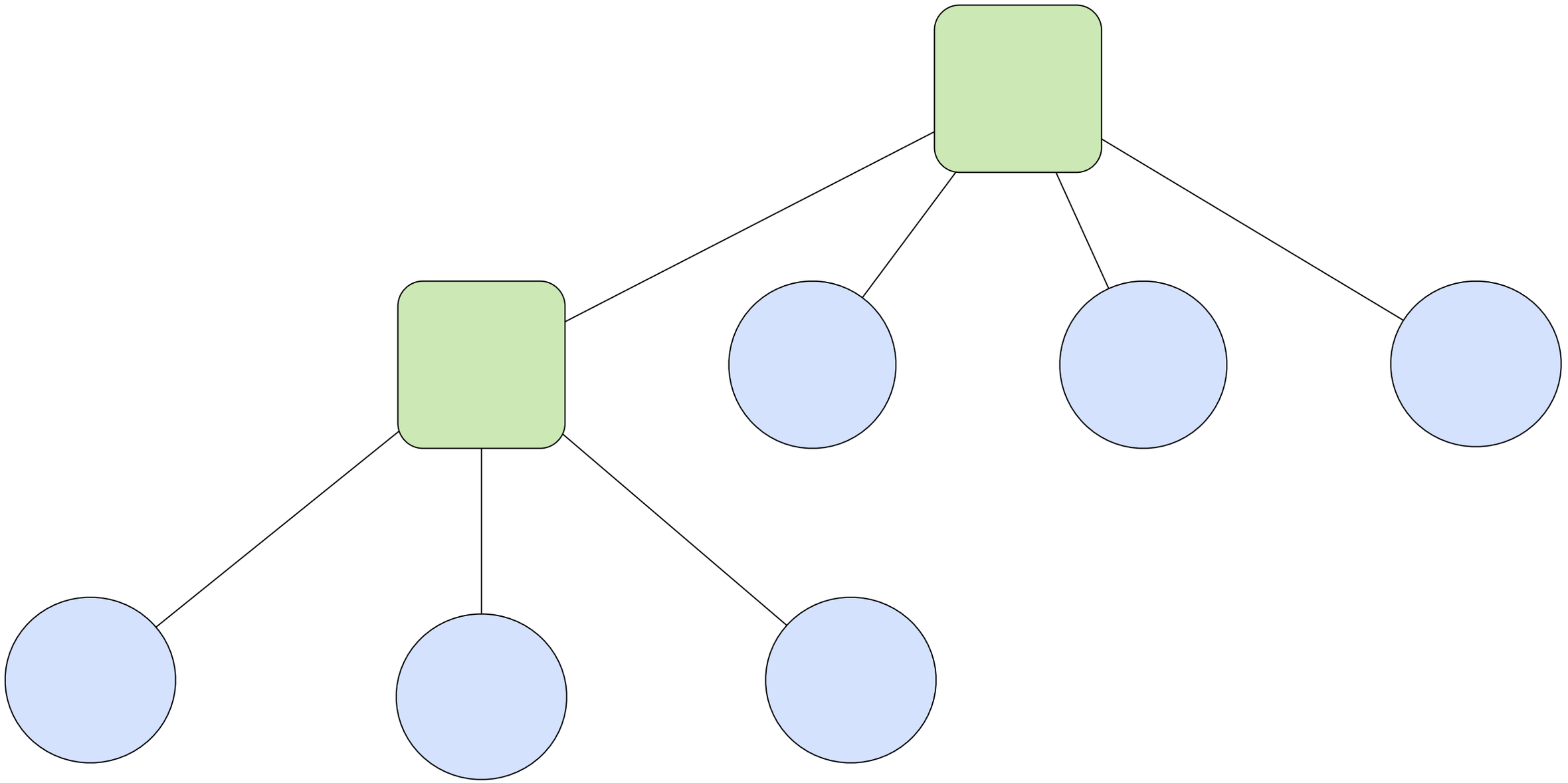
Supervisor hierarchies



Supervisor hierarchies



Supervisor hierarchies



Fault handlers

```
AllForOneStrategy(
  maxNrOfRetries,
  withinTimeRange)

OneForOneStrategy(
  maxNrOfRetries,
  withinTimeRange)
```

Linking

```
link(actor)
unlink(actor)

startLink(actor)
spawnLink[MyActor]
```

trapExit

```
trapExit = List(
  classOf[ServiceException],
  classOf[PersistenceException])
```

Supervision

```
class Supervisor extends Actor {
  import self._
  trapExit = List(classOf[Throwable])
  faultHandler =
    Some(OneForOneStrategy(5, 5000))

  def receive = {
    case Register(actor) =>
      link(actor)
  }
}
```

Manage failure

```
class FaultTolerantService extends Actor {
  ...
  override def preRestart(reason: Throwable) = {
    ... // clean up before restart
  }
  override def postRestart(reason: Throwable) = {
    ... // init after restart
  }
}
```

Remote Actors

Remote Server

```
// use host & port in config
RemoteNode.start

RemoteNode.start("localhost", 9999)
```

Scalable implementation based on
NIO (Netty) & Protobuf

Two types of

remote actors

- Client-initated and managed
- Server-initiated and managed

Client-managed

supervision works across nodes

```
// methods in Actor class

spawnRemote[MyActor](host, port)

spawnLinkRemote[MyActor](host, port)

startLinkRemote(actor, host, port)
```

Client-managed

moves actor to server
client manages through proxy

```
val actorProxy = spawnLinkRemote[MyActor](
  “darkstar”,
  9999)

actorProxy ! message
```

Server-managed

register and manage actor on server
client gets “dumb” proxy handle

```
RemoteNode.register(
  “service:id”, actorOf[MyService])
```

server part

Server-managed

```
val handle = RemoteClient.actorFor(
  “service:id”,
  “darkstar”,
  9999)

handle ! message
```

client part

Cluster Membership

```
Cluster.relayMessage(
  classOf[TypeOfActor], message)

for (endpoint <- Cluster)
  spawnRemote[TypeOfActor](
                endpoint.host,
                endpoint.port)
```

STM

another tool in the toolbox

What is STM?

STM: overview

- See the memory (heap and stack) as a transactional dataset
- Similar to a database
  - begin
  - commit
  - abort/rollback
- Transactions are retried automatically upon collision
- Rolls back the memory on abort

Managed References

- Separates Identity from Value
  - Values are immutable
  - Identity (Ref) holds Values
- Change is a function
- Compare-and-swap (CAS)
- Abstraction of time
- Must be used within a transaction

Managed References

```
val ref = Ref(Map[String, User]())

val users = ref.get

val newUsers = users + (“bill” -> User(“bill”))

ref.swap(newUsers)
```

Transactional datastructures

```
val users = TransactionalMap[String, User]()

val users = TransactionalVector[User]()
```

STM: API

```
import Transaction.Local._

atomic {
  ...
  atomic { // nested transactions compose
    ... // do something within a transaction
  }
}
```

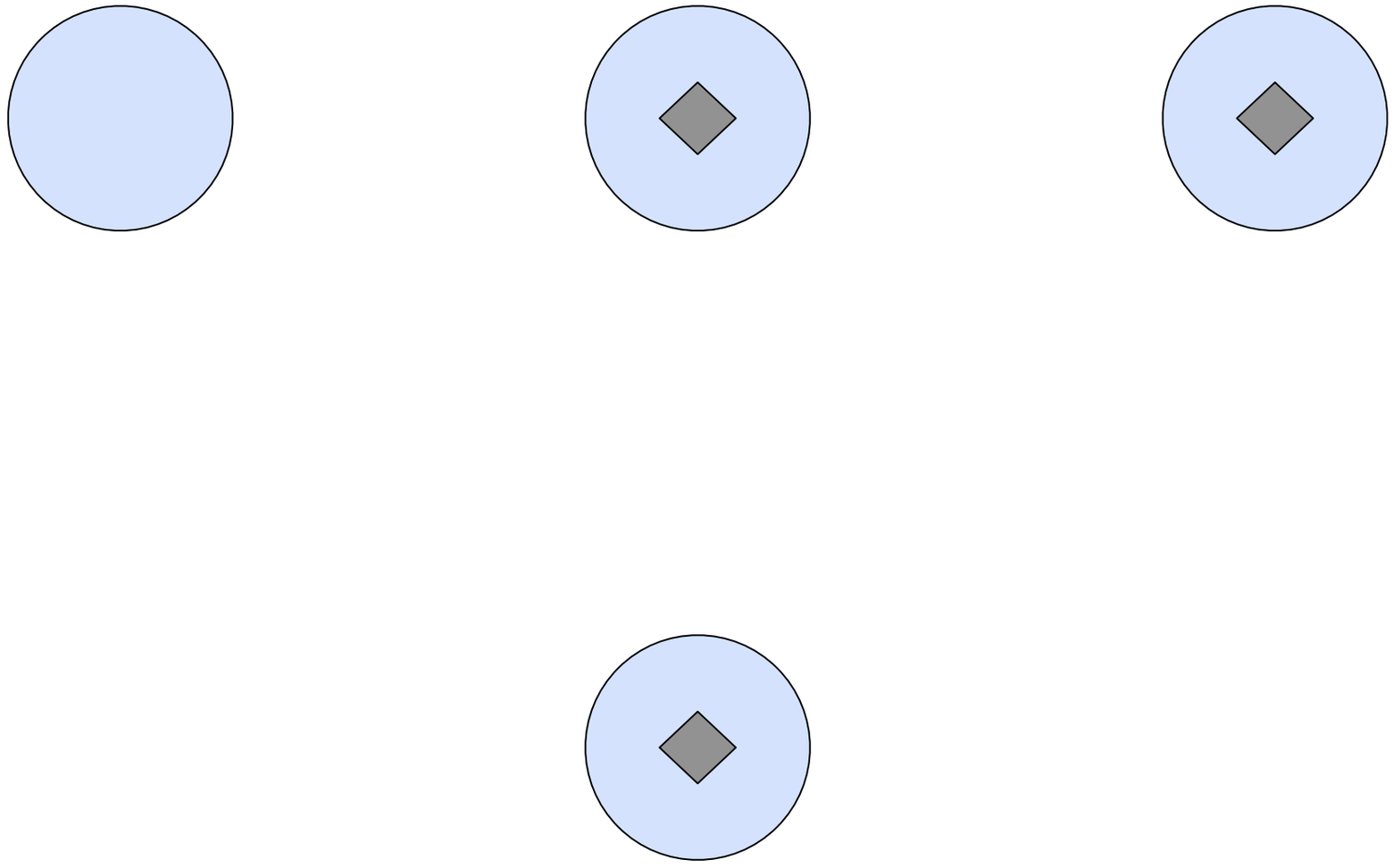
Actors + STM =

Transactors

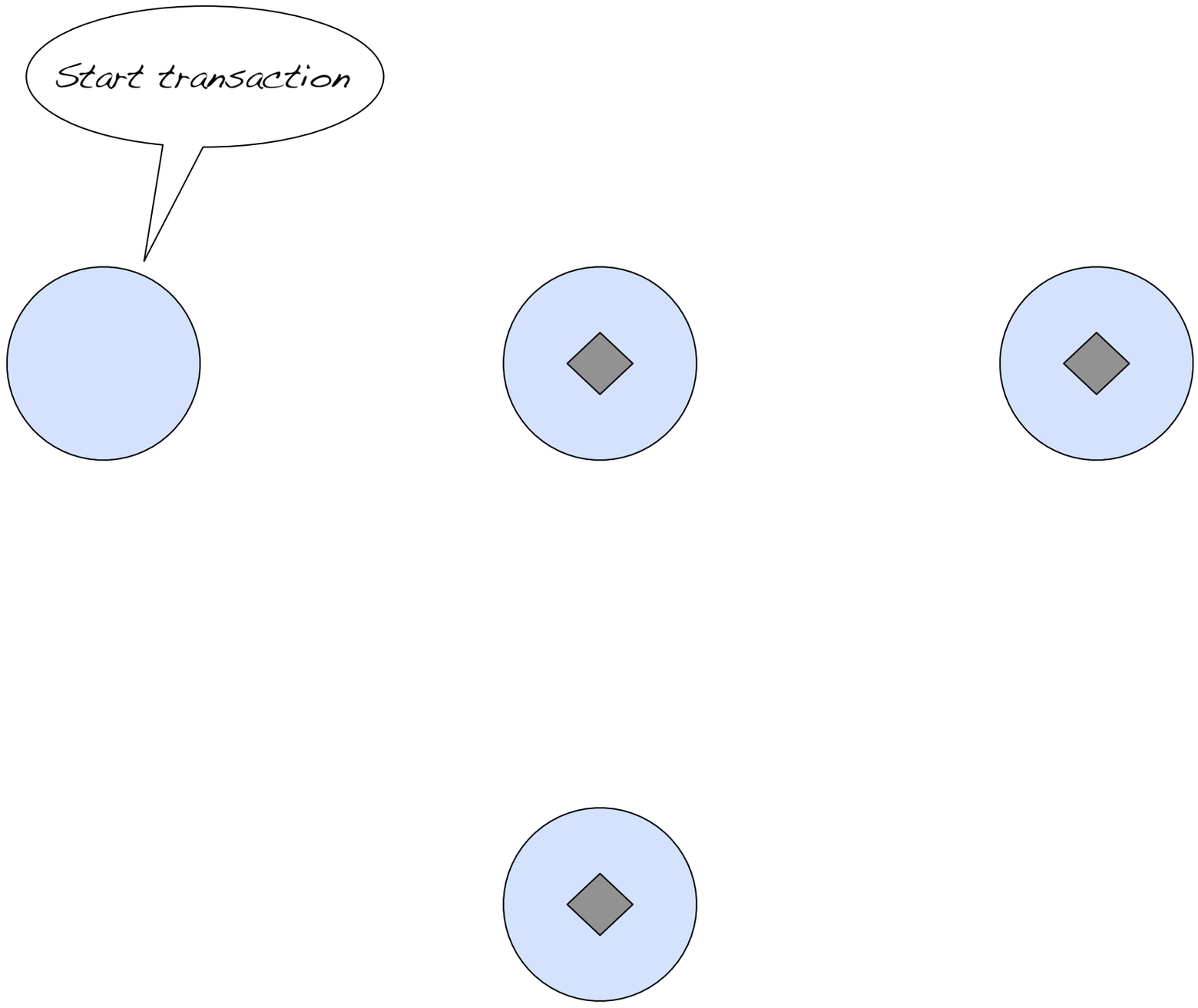
Transactors

```
class UserRegistry extends Transactor {

  private val storage =
    TransactionalMap[String, User]()

  def receive = {
    case NewUser(user) =>
      storage + (user.name -> user)
    ...
  }
}
```

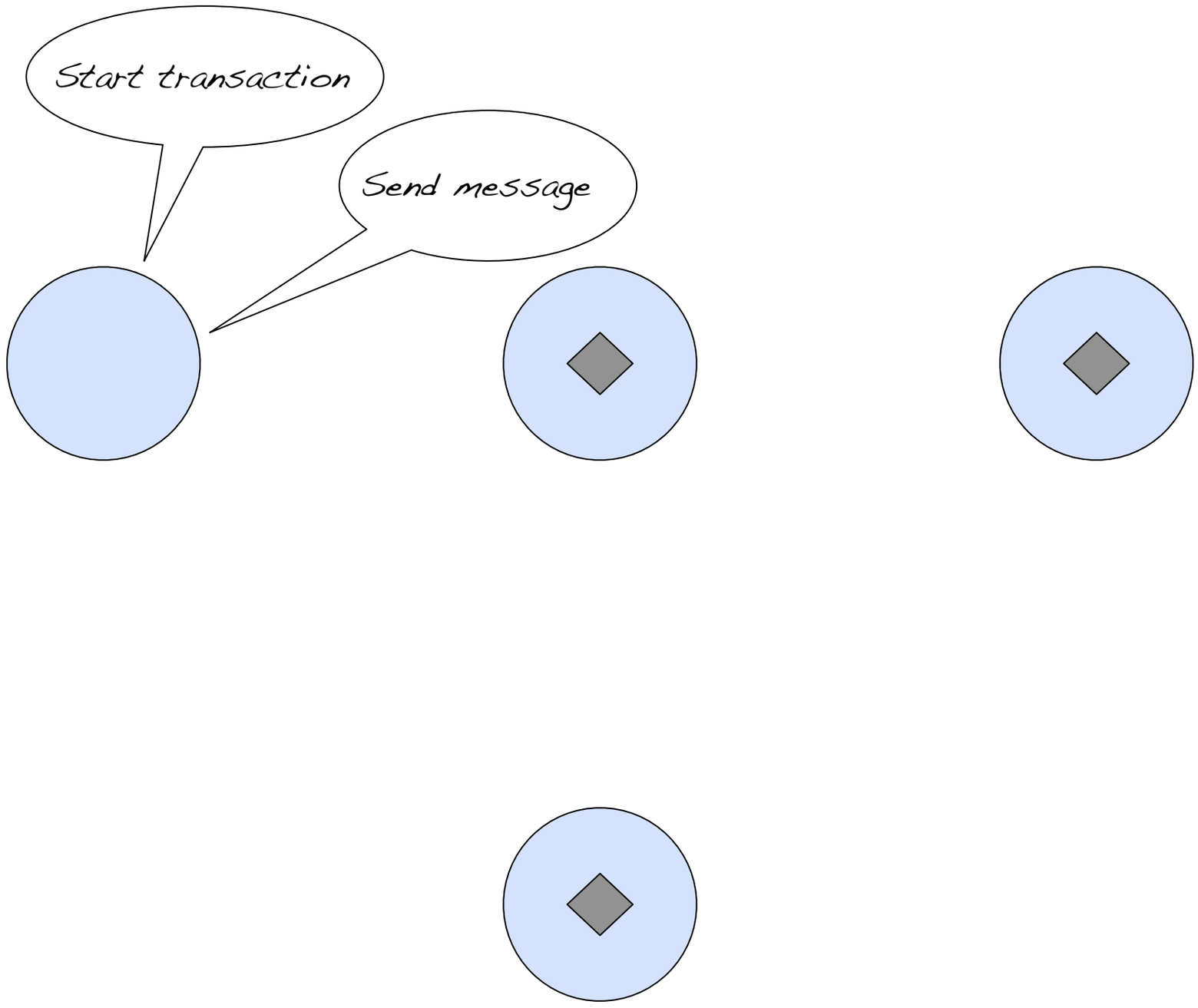
Transactors



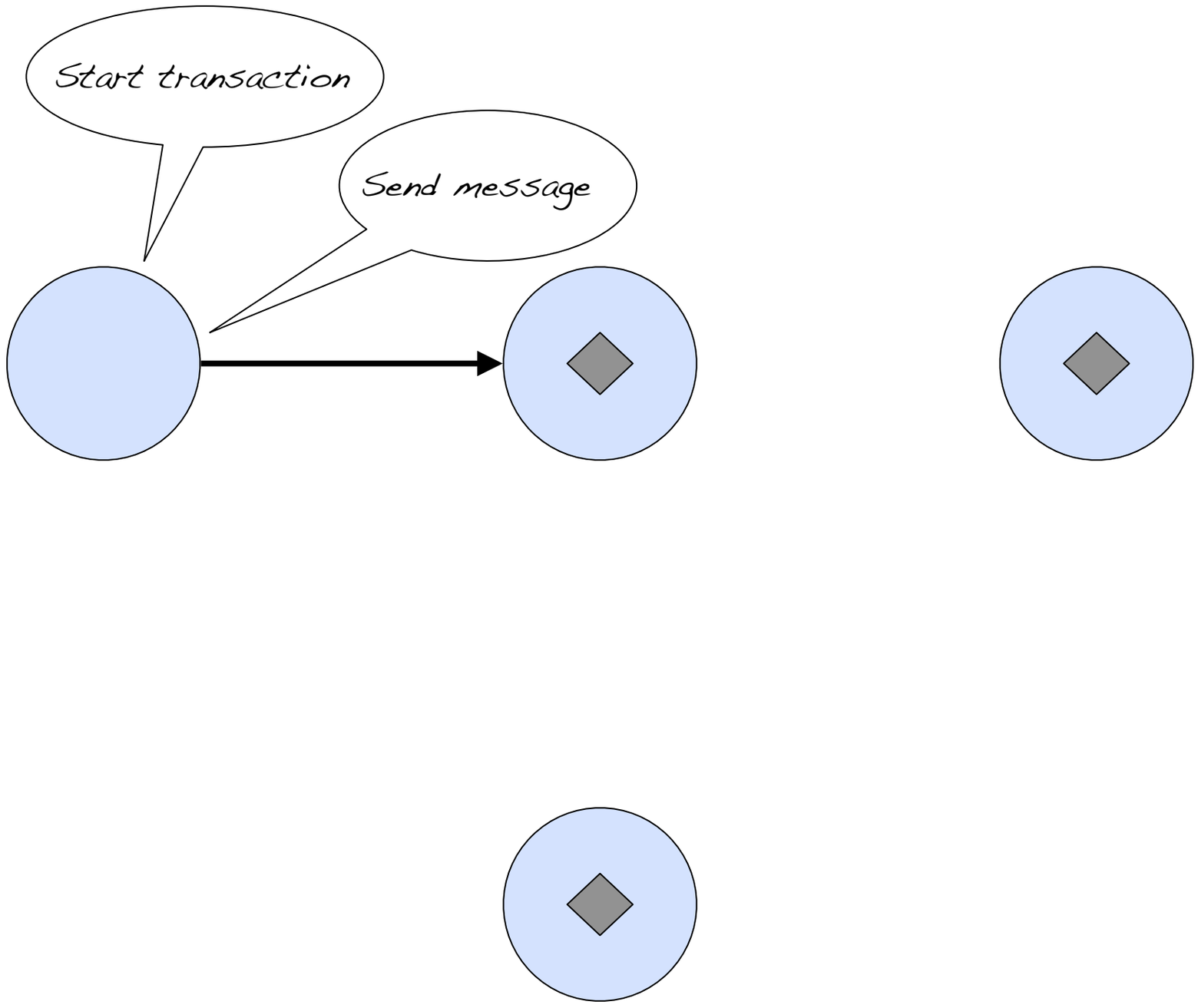
Transactors



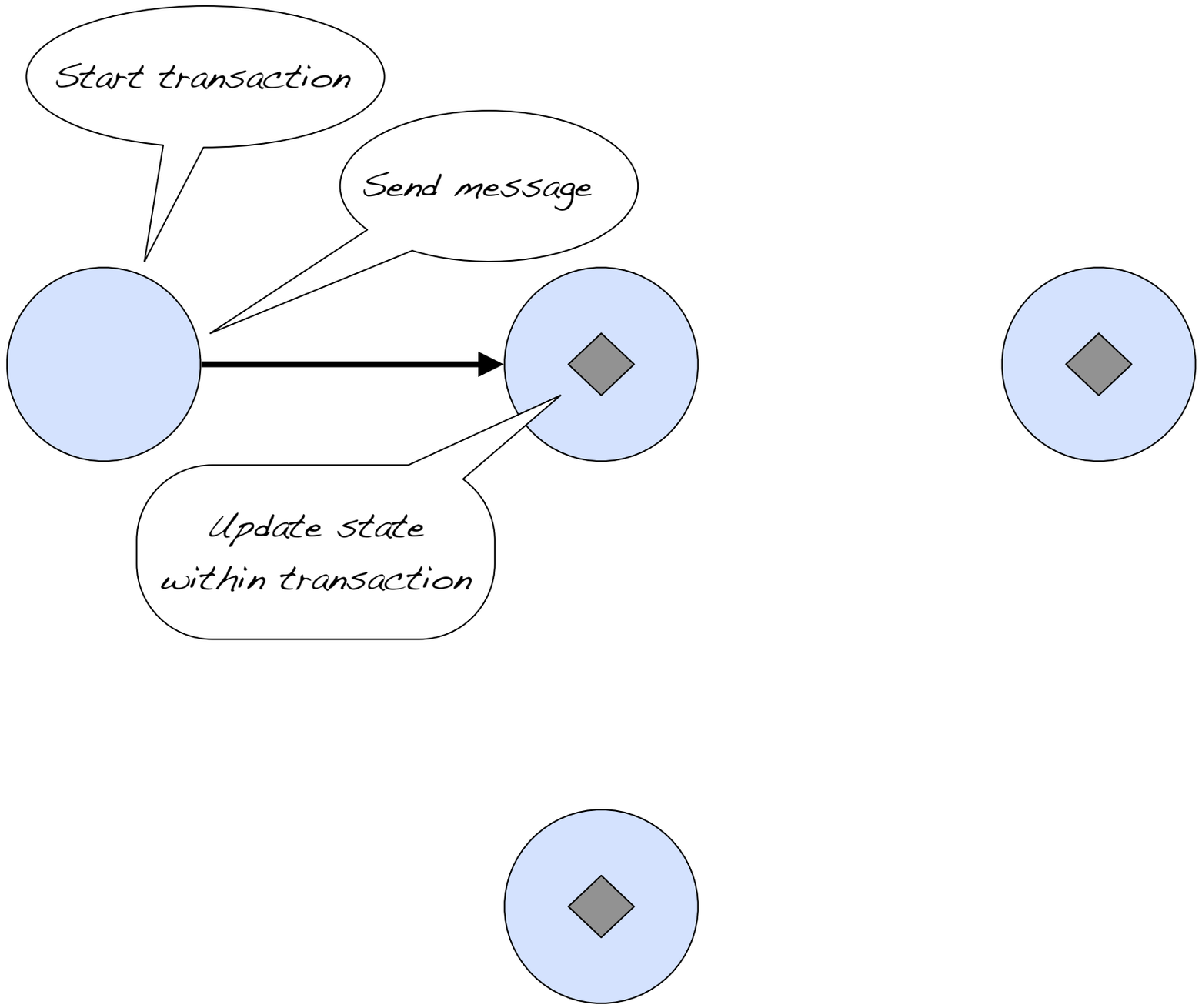
Transactors



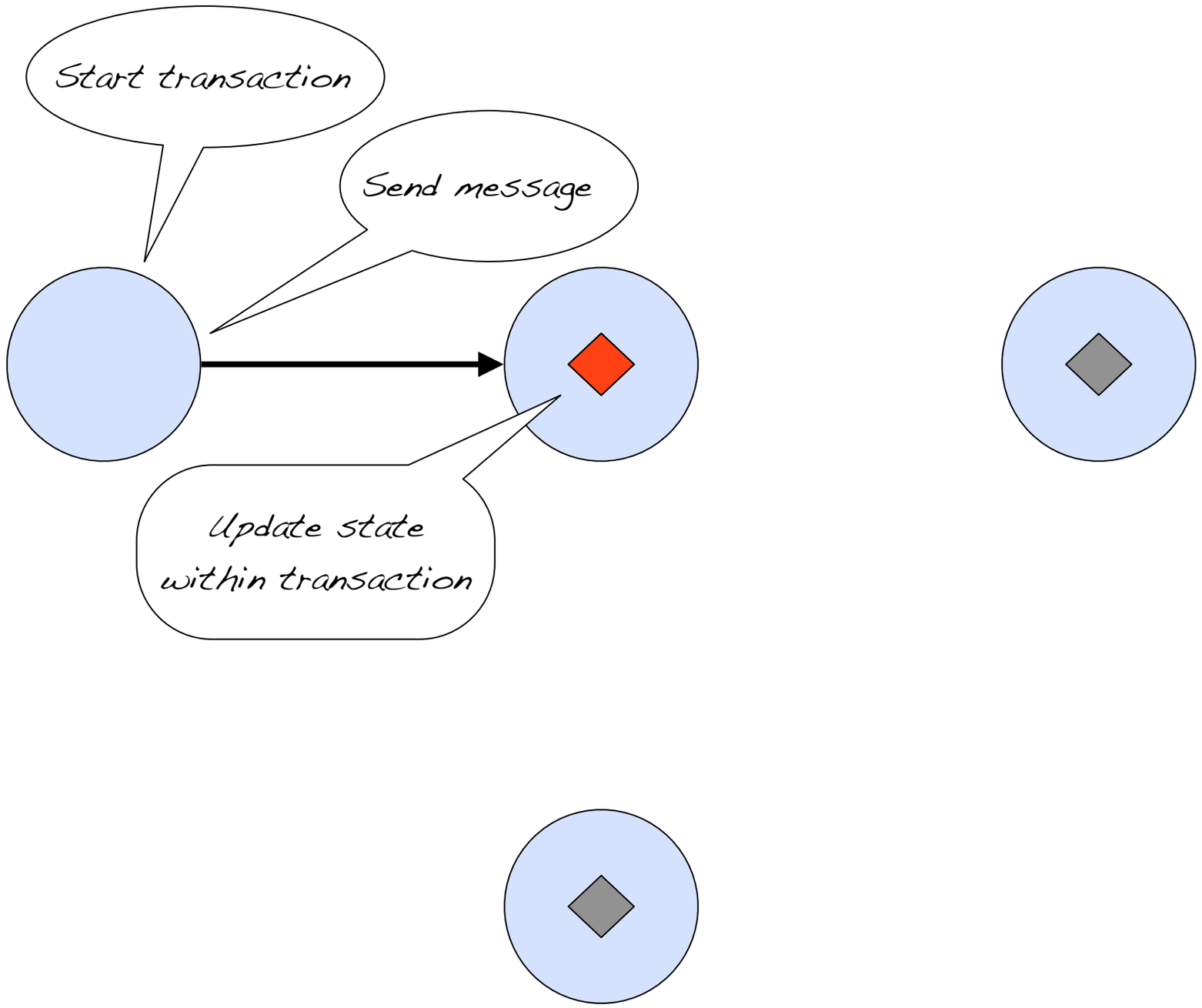
Transactors



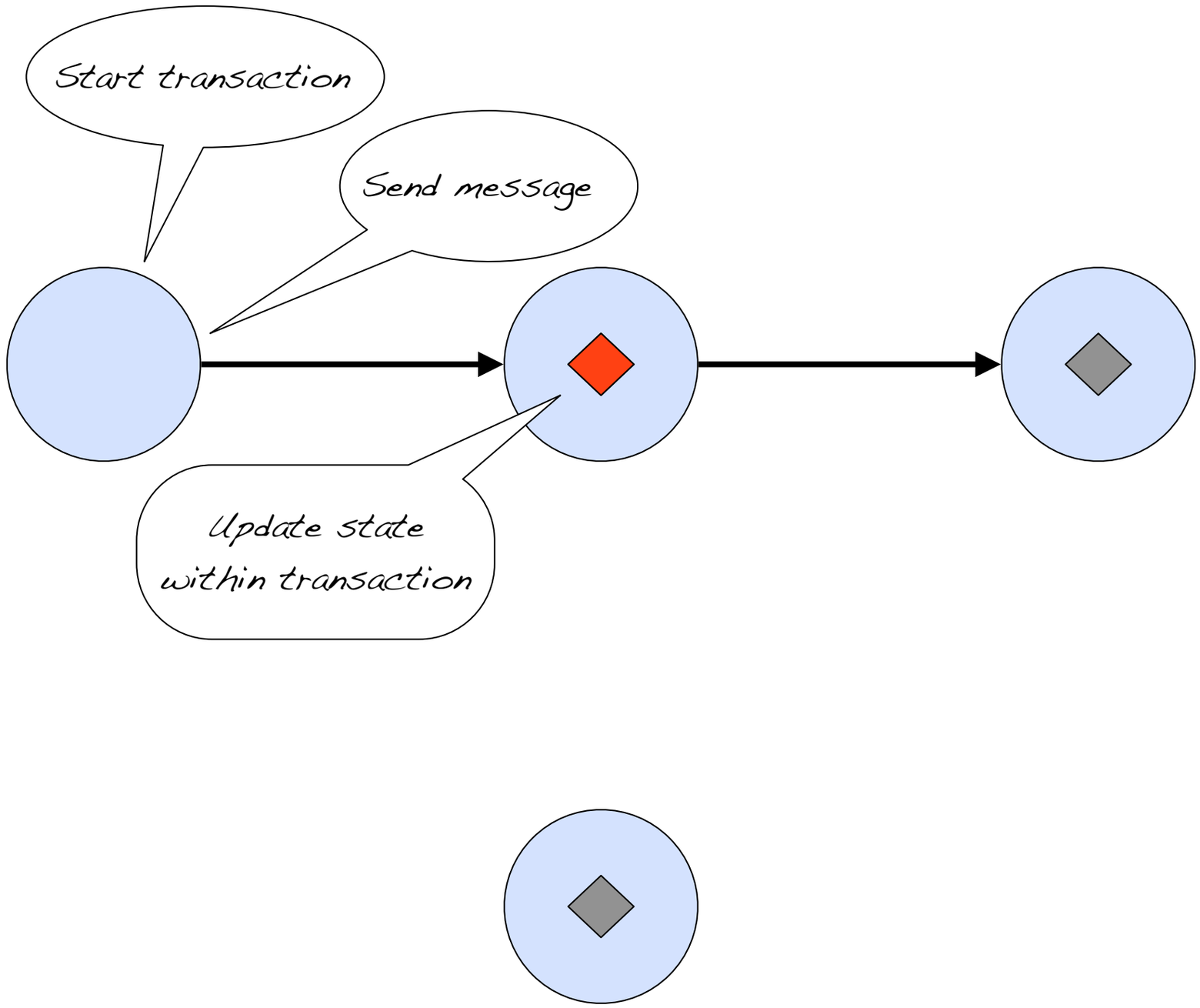
Transactors



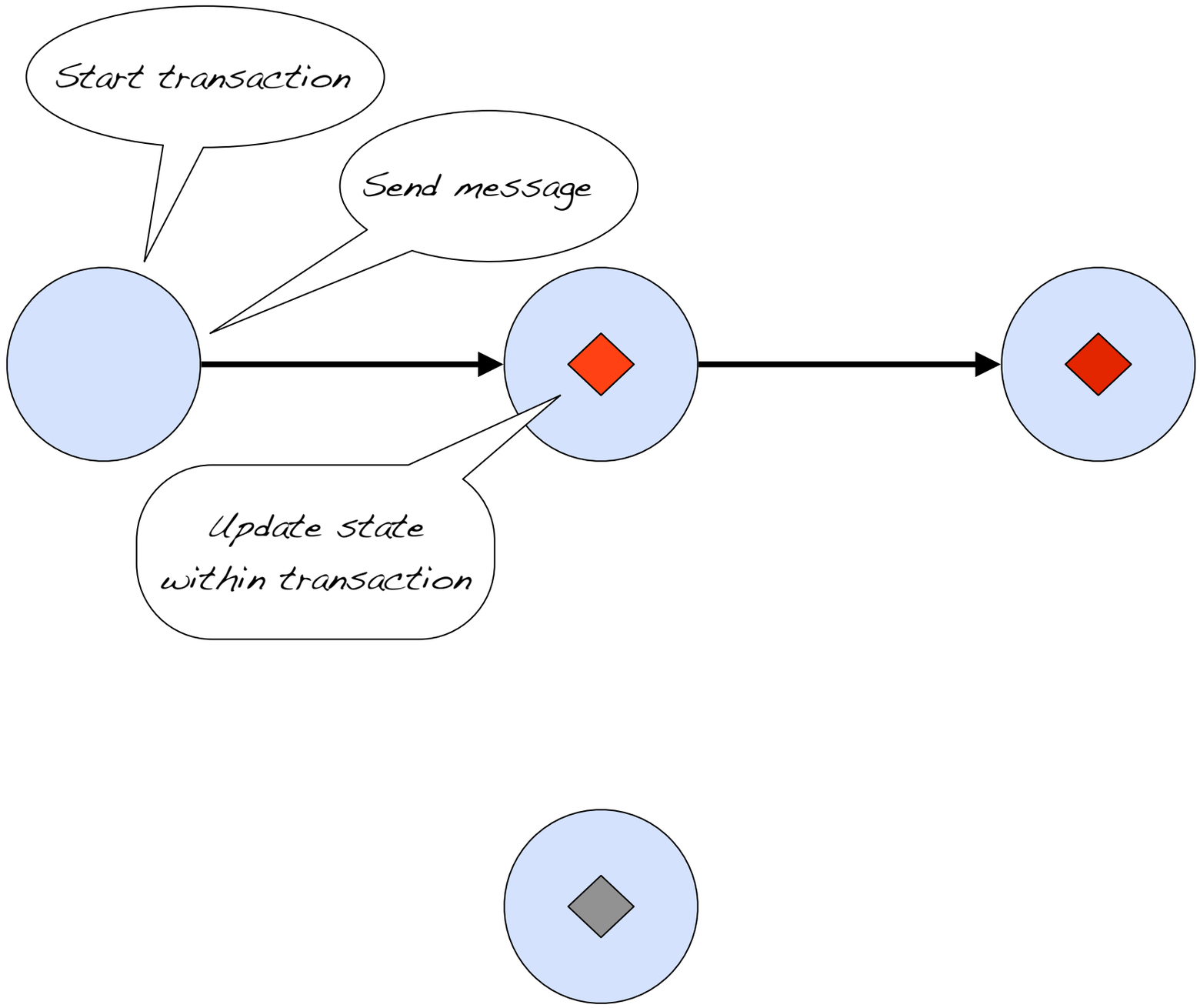
Transactors



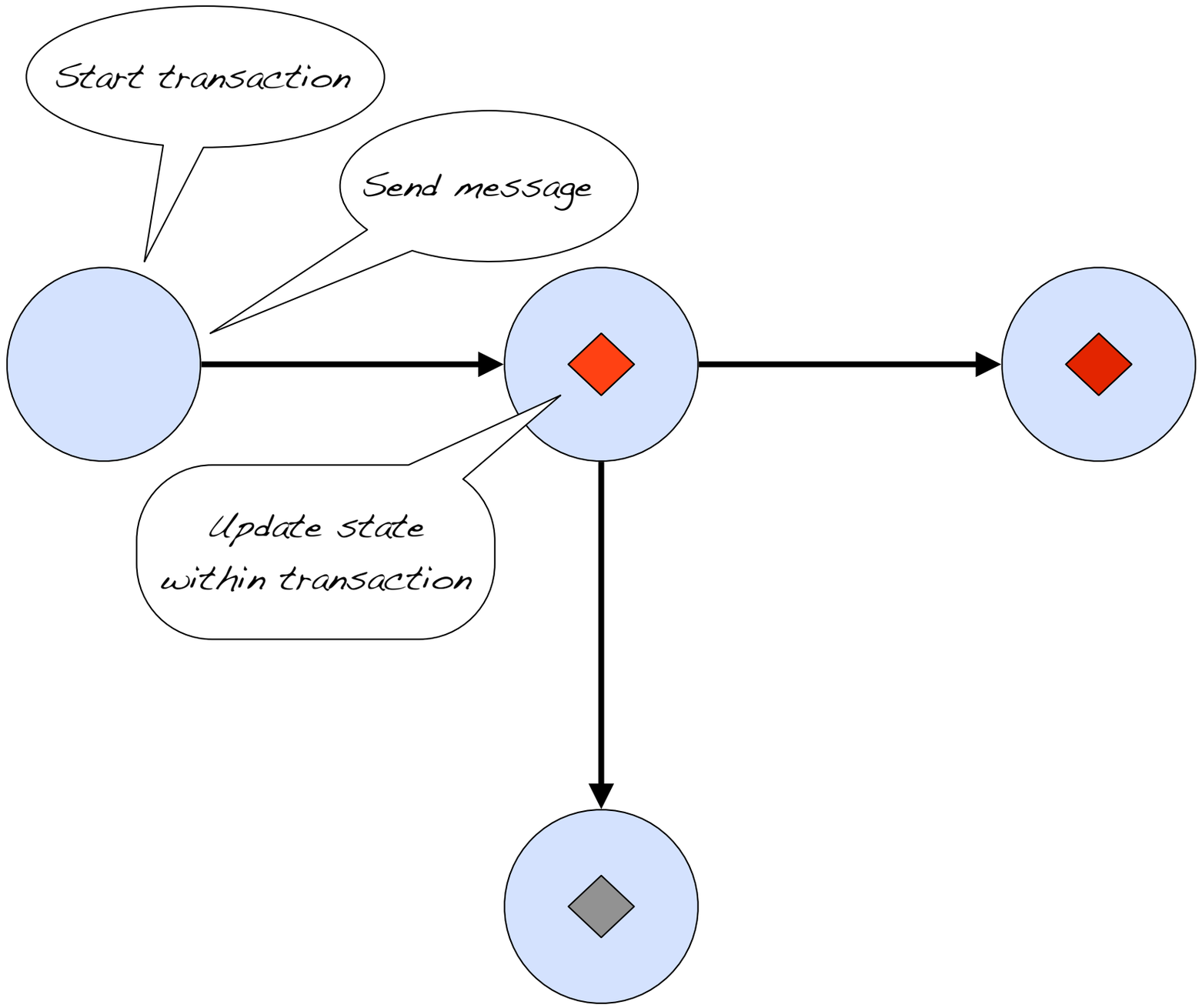
Transactors



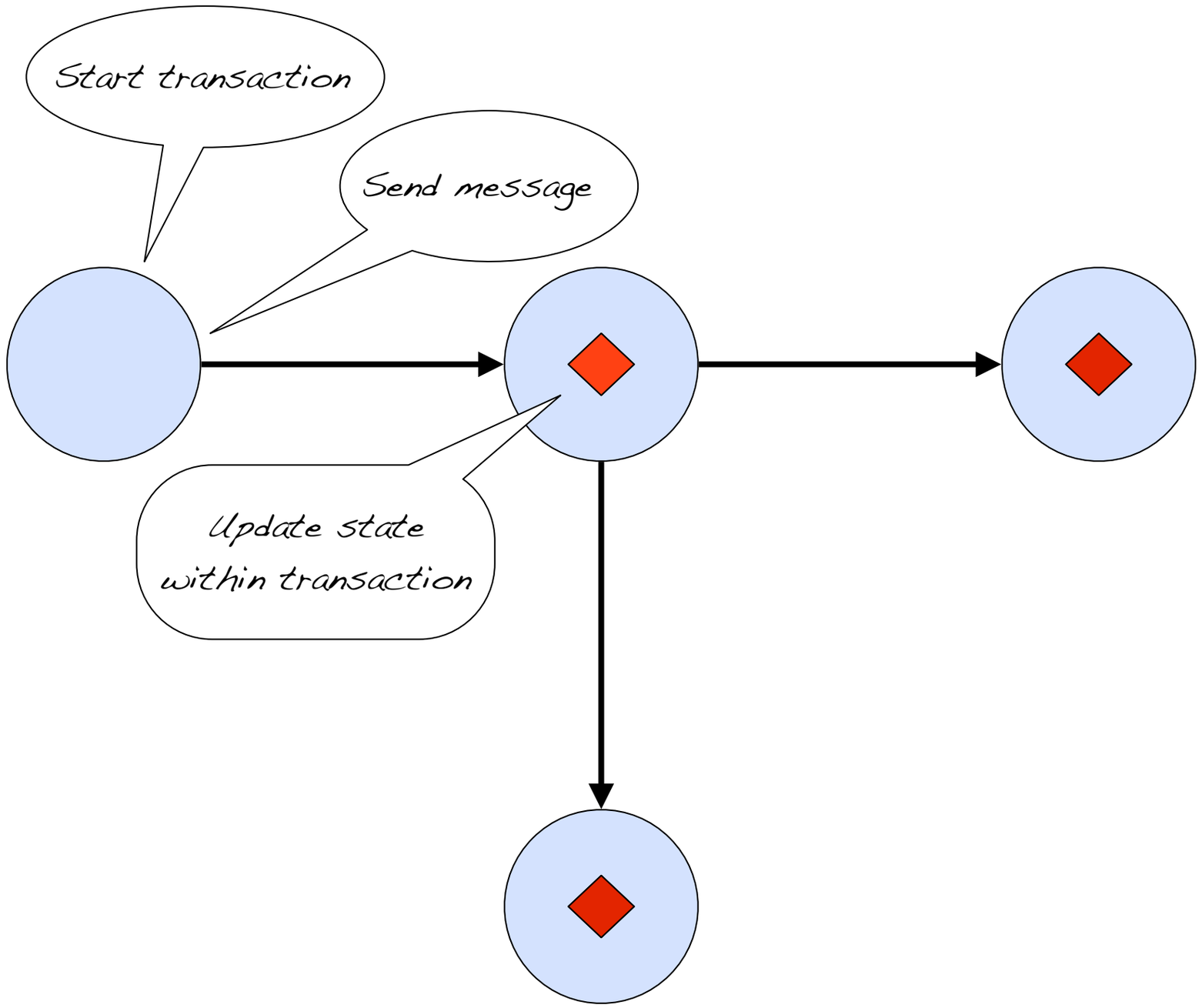
Transactors



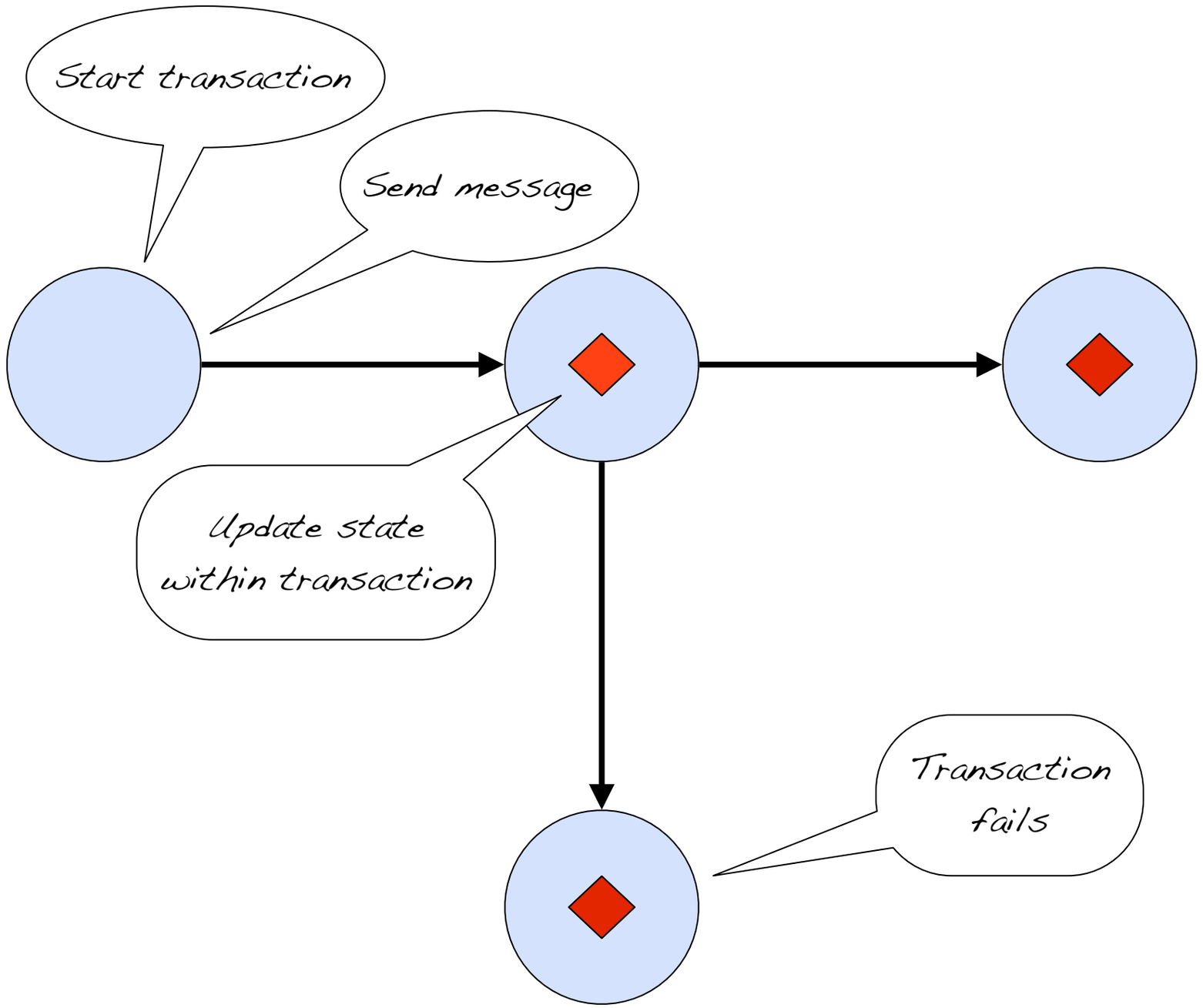
Transactors



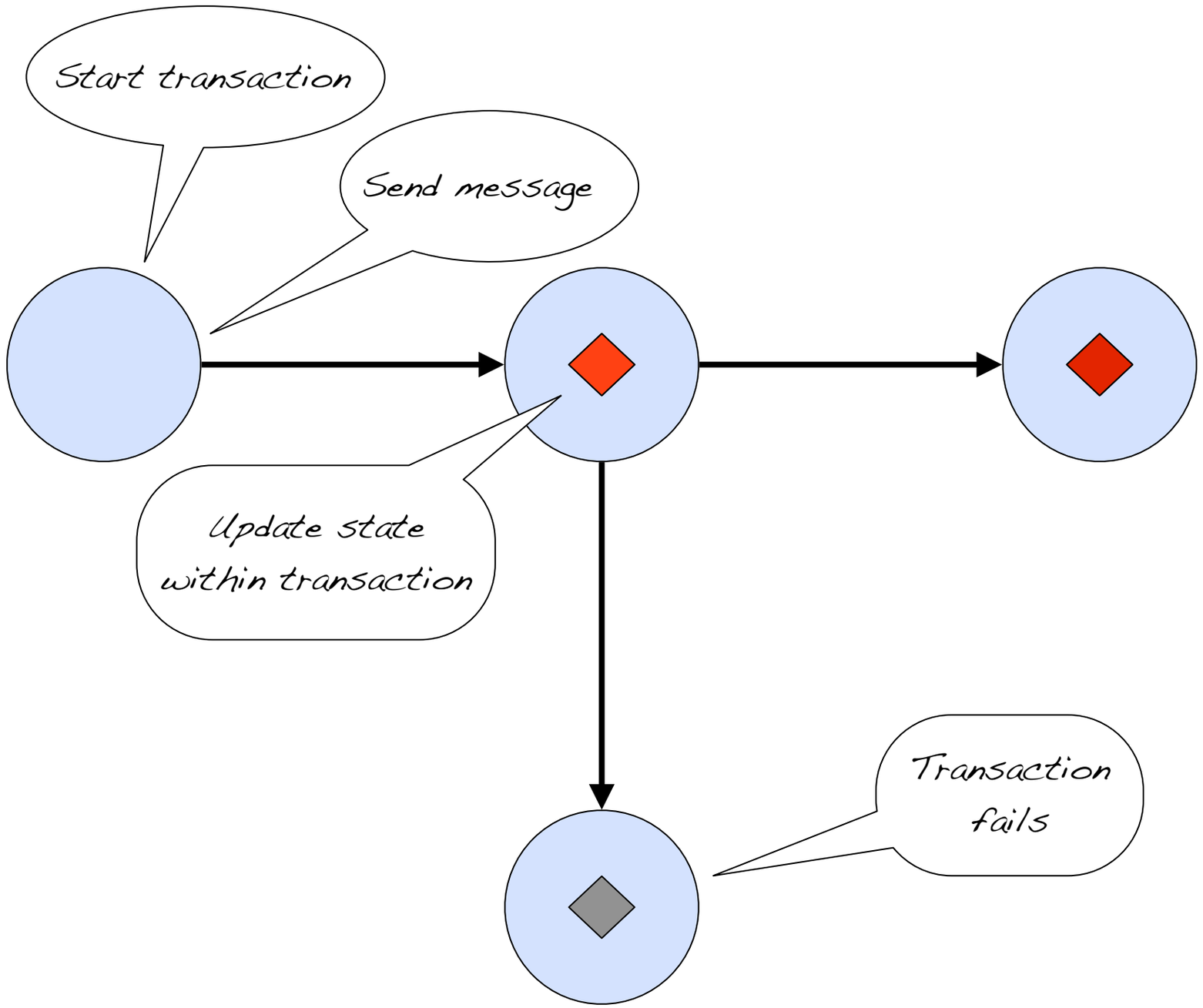
Transactors



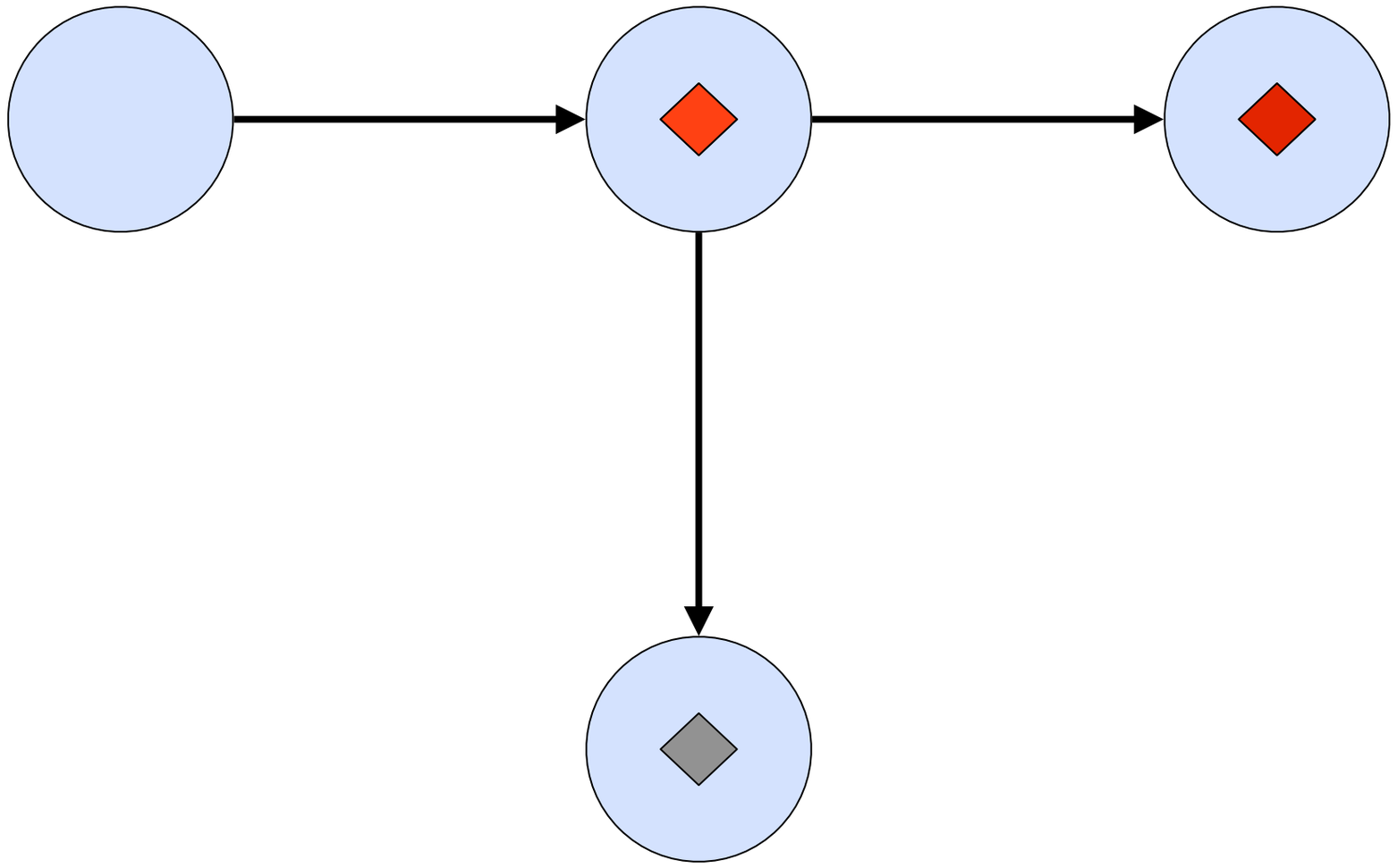
Transactors



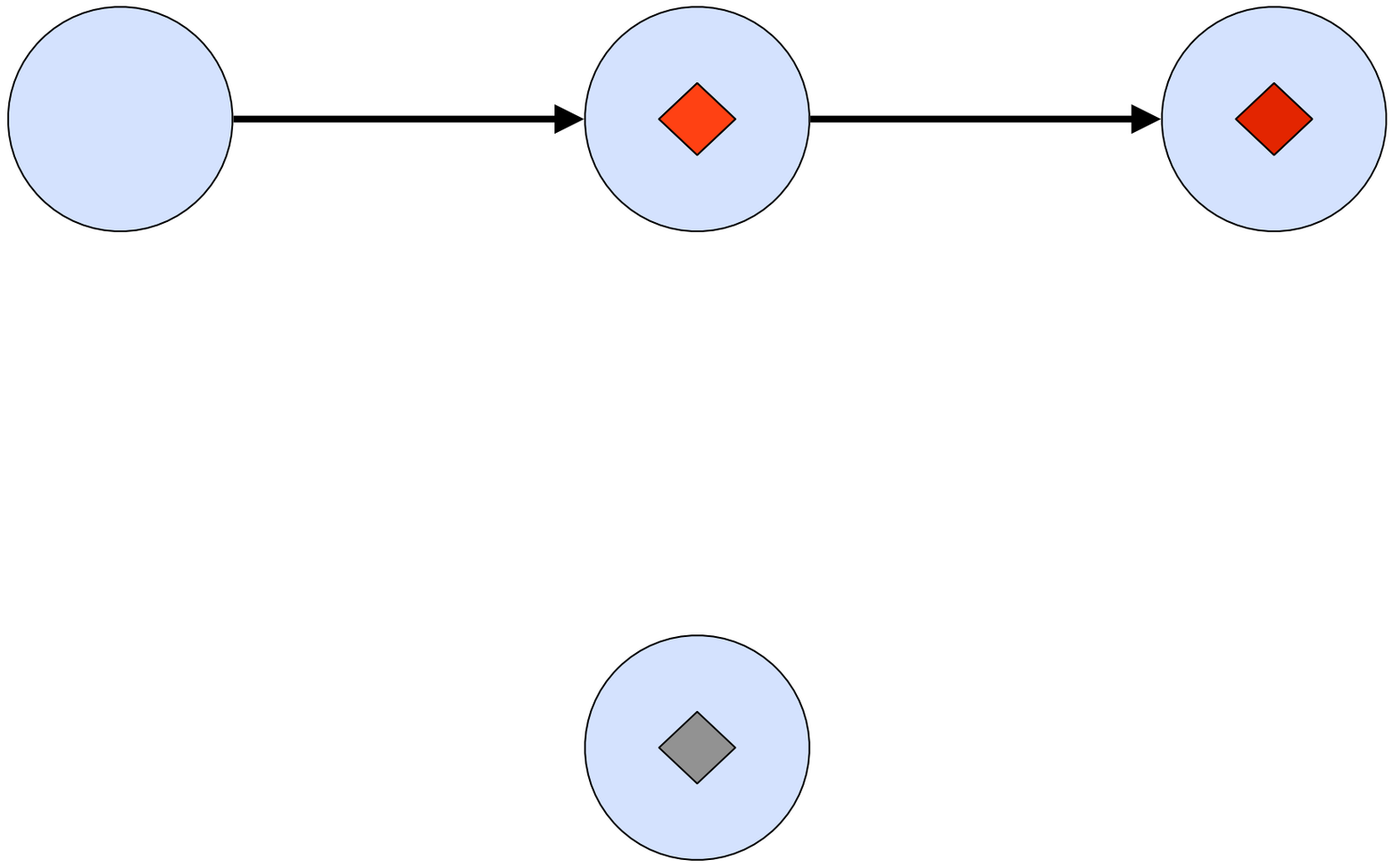
Transactors



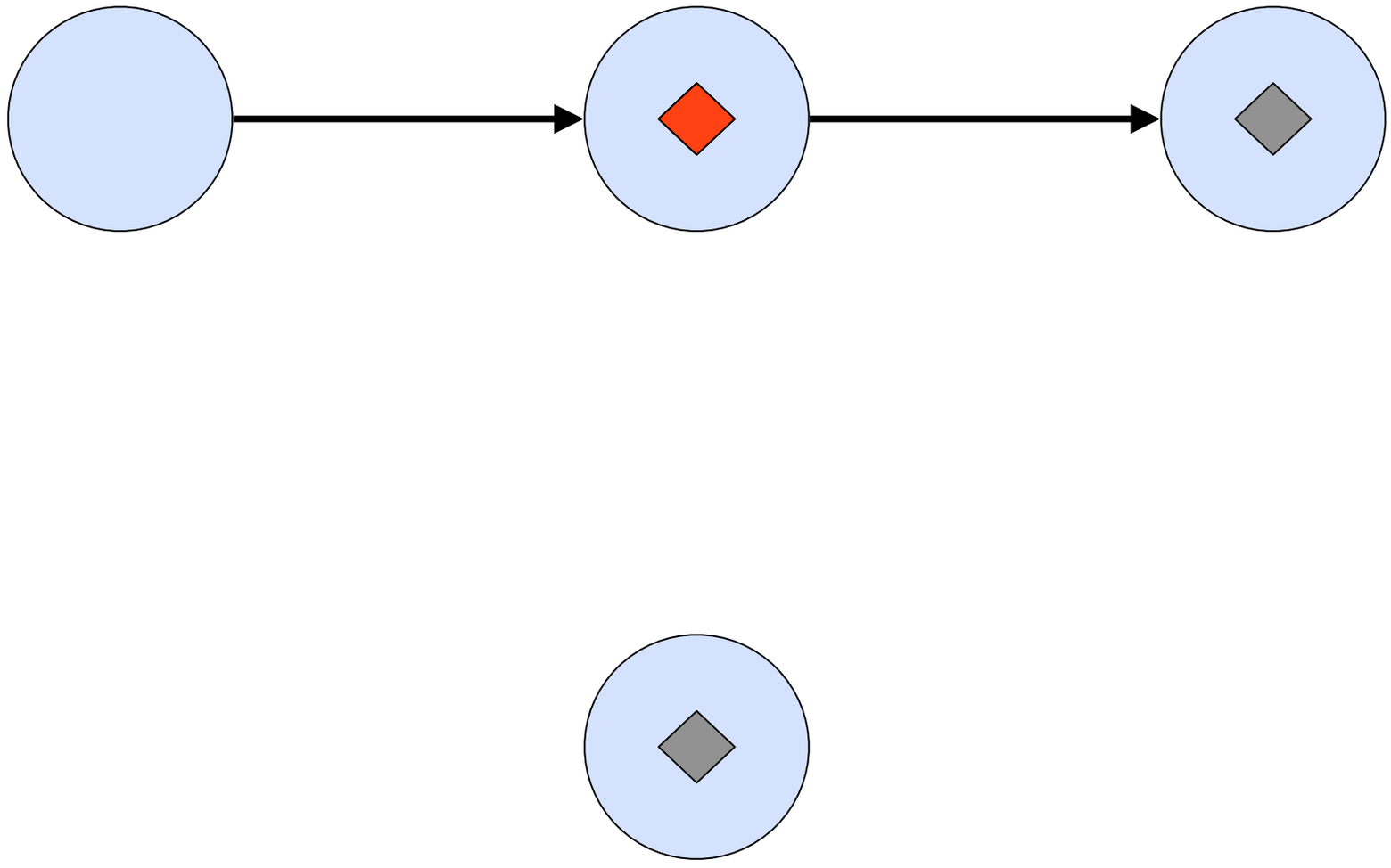
Transactors



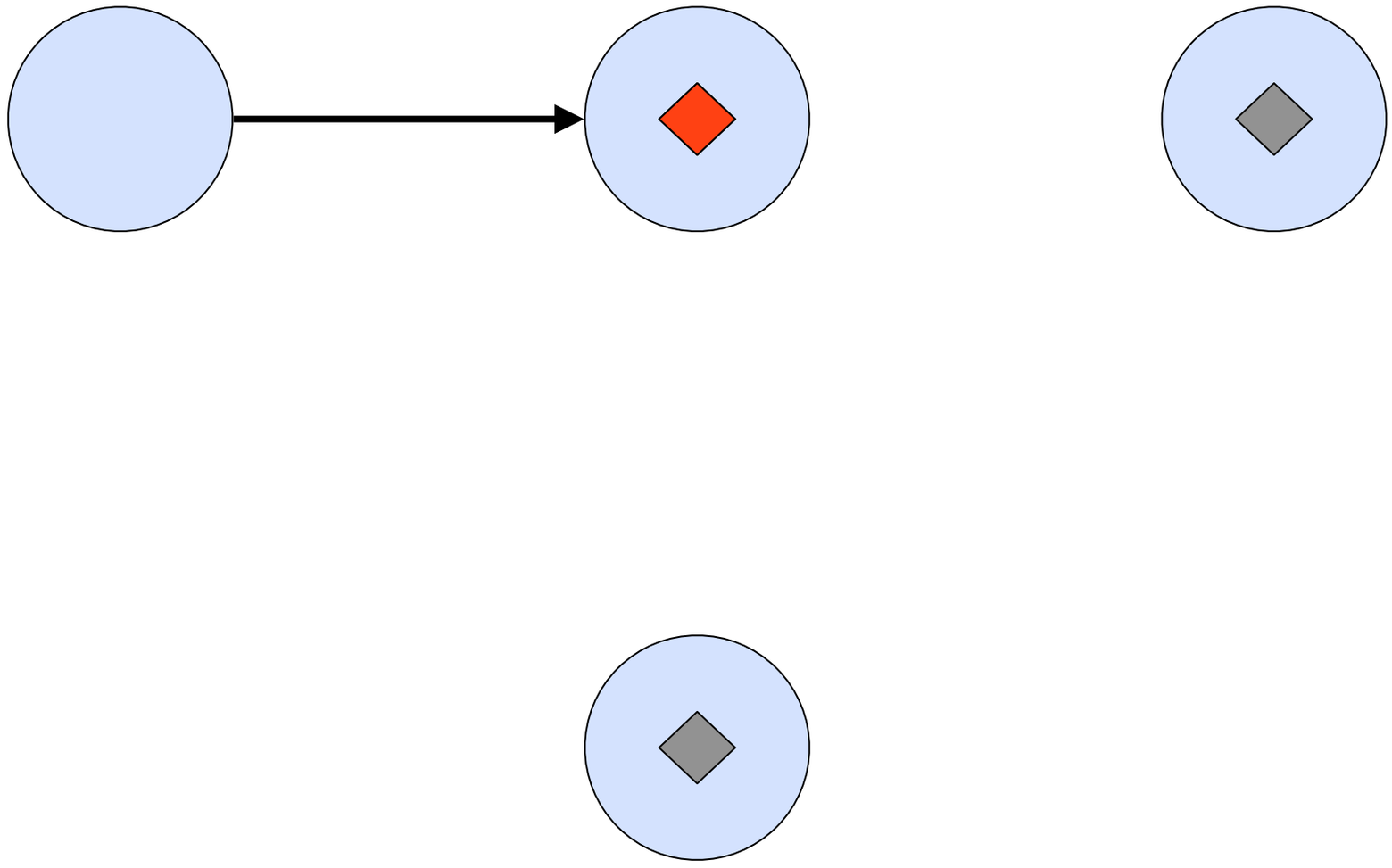
Transactors



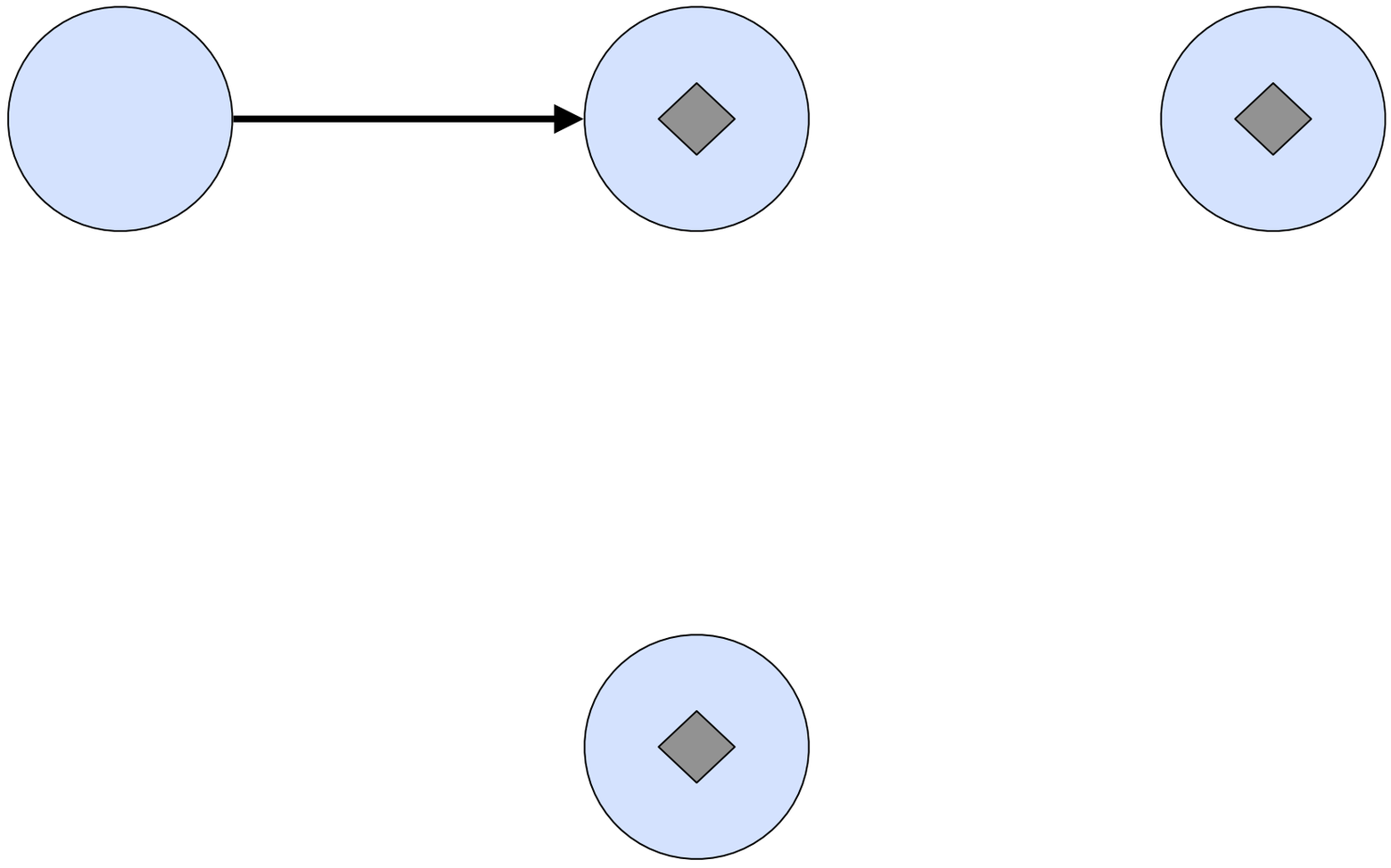
Transactors



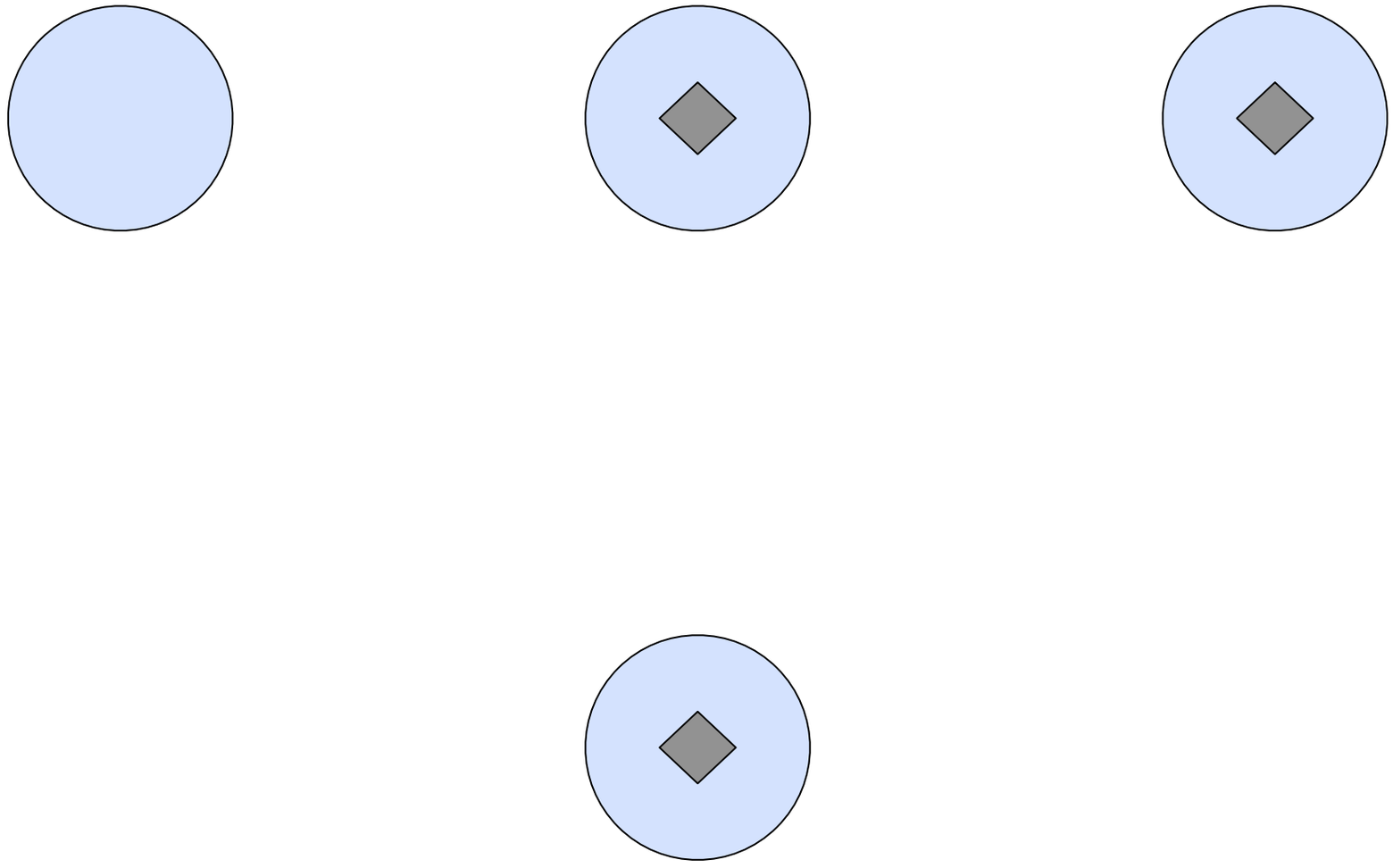
Transactors



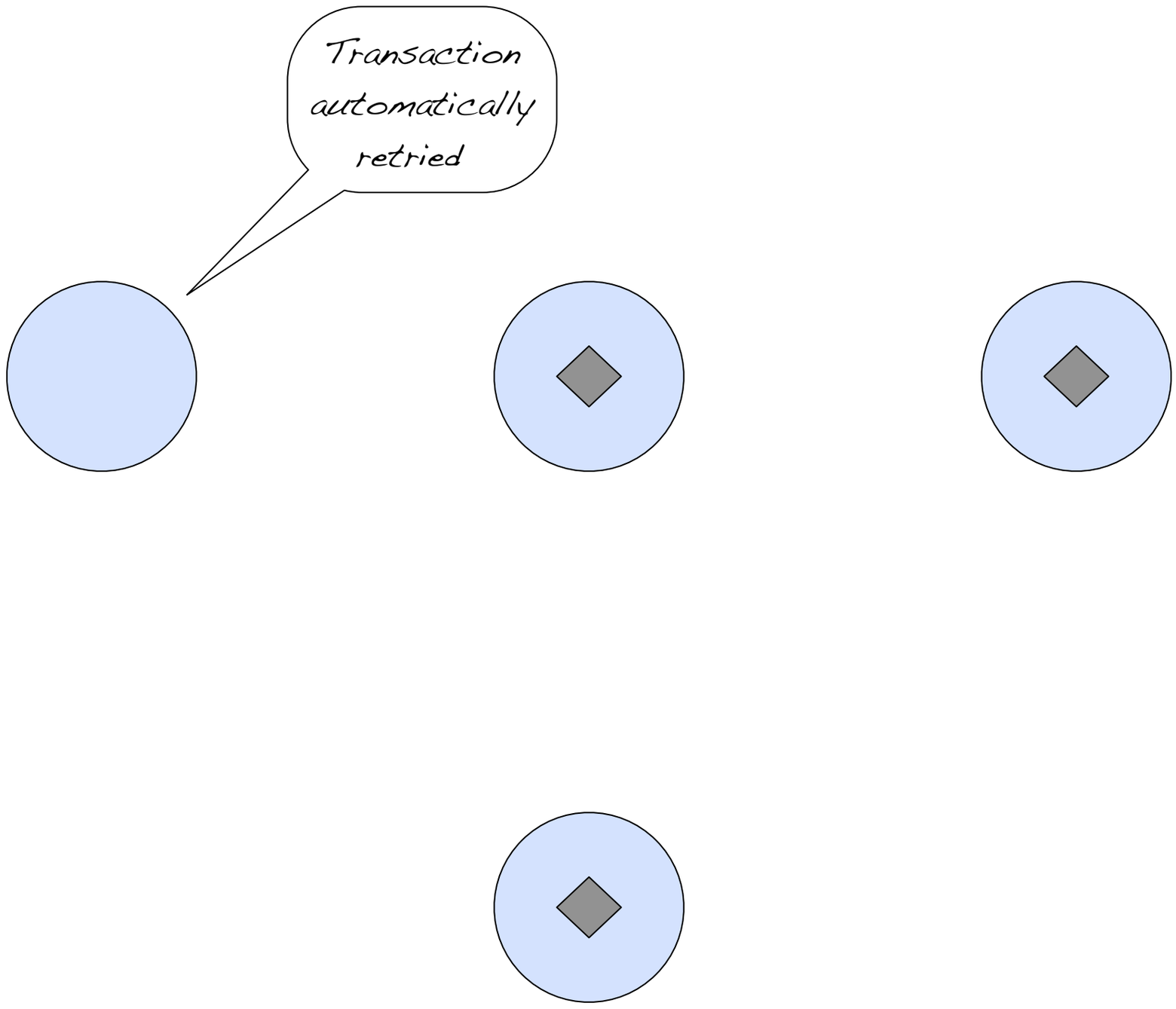
Transactors



Transactors



Transactors



Agents

yet another tool in the toolbox

Agents

```
val agent = Agent(5)

// send function asynchronously
agent send (_ + 1)

val result = agent() // deref
... // use result

agent.close
```

Cooperates with STM

(some of the)

Modules

Akka Spring

Spring integration

```
<beans>
  <akka:active-object id="myActiveObject"
                      target="com.biz.MyPOJO"
                      transactional="true"
                      timeout="1000" />

  ...
</beans>
```

Spring integration

```
<akka:supervision id="my-supervisor">

  <akka:restart-strategy failover="AllForOne"
                         retries="3"
                         timerange="1000">
    <akka:trap-exits>
      <akka:trap-exit>java.io.IOException</akka:trap-exit>
    </akka:trap-exits>
  </akka:restart-strategy>

  <akka:active-objects>
    <akka:active-object target="com.biz.MyPOJO"
                        lifecycle="permanent"
                        timeout="1000">
      <akka:restart-callbacks pre="preRestart"
                              post="postRestart"/>
    </akka:active-object>
  </akka:active-objects>
</akka:supervision>
```

Akka Camel

Camel: consumer

```
class MyConsumer extends Actor with Consumer {
  def endpointUri = "file:data/input"

  def receive = {
    case msg: Message =>
      log.info("received %s" format
      msg.bodyAs(classOf[String]))
  }
}
```

Camel: consumer

```
class MyConsumer extends Actor with Consumer {
  def endpointUri =
    "jetty:http://0.0.0.0:8877/camel/test"

  def receive = {
    case msg: Message =>
      reply("Hello %s" format
            msg.bodyAs(classOf[String]))
  }
}
```

Camel: producer

```
class CometProducer
  extends Actor with Producer {

  def endpointUri =
    "cometd://localhost:8111/test"
}
```

Camel: producer

```
val producer = actorOf[CometProducer].start

val time = "Current time: " + new Date
producer ! time
```

Akka Persistence

Akka Persistence API

```
// transactional Cassandra-backed Map
val map = CassandraStorage.newMap

// transactional Redis-backed Vector
val vector = RedisStorage.newVector

// transactional Mongo-backed Ref
val ref = MongoStorage.newRef
```

Turns the STM into ACID

Get data by id

```
// transactional Cassandra-backed Map
val map = CassandraStorage.getMap(uuid)

// transactional Redis-backed Vector
val vector = RedisStorage.getVector(uuid)

// transactional Mongo-backed Ref
val ref = MongoStorage.getRef(uuid)
```

For Redis only (so far)

```
val queue: PersistentQueue[ElementType] =
  RedisStorage.newQueue

val set: PersistentSortedSet[ElementType] =
  RedisStorage.newSortedSet
```

Akka Deployment

Deployment

- ☑ Deploy as dependency JAR in WEB-INF/lib etc.
- ☑ Run as microkernel
- ☑ Soon OSGi-enabled, then drop in any OSGi container (Spring DM server, Karaf etc.)

...and much more

REST

Security

Web

Comet

AMQP

JTA

Guice

Learn more

http://akkasource.org

EOF

?