```
class EspressoMachine(scope: CoroutineScope): CoroutineScope by scope {
    private data class EspressoShotRequest(
        val deferredEspressoShot: CompletableDeferred<Espresso>,
        val groundBeans: CoffeeBean.GroundBeans
    )

    private val portafilterOne: SendChannel<EspressoShotRequest> = actor {
        consumeEach { request ->
            val espresso = processEspressoShot(request.groundBeans)
            request.deferredEspressoShot.complete(espresso)
        }
    }

    private val portafilterTwo: SendChannel<EspressoShotRequest> = actor {...}

    suspend fun pullEspressoShot(groundBeans: CoffeeBean.GroundBeans): Espresso {
        val request = EspressoShotRequest(CompletableDeferred(), groundBeans)
        return select {
            portafilterOne.onSend(request) {
                request.deferredEspressoShot.await()
            }
            portafilterTwo.onSend(request) {
                TODO("sent to portafilter two - wait for result")
            }
        }
    }
    ...
}
```

https://jagstalk.page.link/examples

https://jagstalk.page.link/examples

```
class EspressoMachine(scope: CoroutineScope): CoroutineScope by scope {
    private data class EspressoShotRequest(
        val deferredEspressoShot: CompletableDeferred<Espresso>,
        val groundBeans: CoffeeBean.GroundBeans
    )

    private val portafilterOne: SendChannel<EspressoShotRequest> = actor {
        consumeEach { request ->
            val espresso = processEspressoShot(request.groundBeans)
            request.deferredEspressoShot.complete(espresso)
        }
    }

    private val portafilterTwo: SendChannel<EspressoShotRequest> = actor {...}

    suspend fun pullEspressoShot(groundBeans: CoffeeBean.GroundBeans): Espresso {
        val request = EspressoShotRequest(CompletableDeferred(), groundBeans)
        return select {
            portafilterOne.onSend(request) {
                request.deferredEspressoShot.await()
            }
            portafilterTwo.onSend(request) {
                TODO("sent to portafilter two - wait for result")
            }
        }
    }
    ...
}
```

https://jagstalk.page.link/examples

```
class EspressoMachine(scope: CoroutineScope): CoroutineScope by scope {
    ...
}

fun main() = runBlocking {
    ...
    val espressoMachine = EspressoMachine(this)
    val time = measureTimeMillis {
        coroutineScope {
            launch(CoroutineName("barista-1")) {
                processOrders(ordersChannel, espressoMachine)
            }
            launch(CoroutineName("barista-2")) {
                processOrders(ordersChannel, espressoMachine)
            }
        }
    }
    log("time: $time ms")
}

private suspend fun processOrders(orders: ReceiveChannel<Menu.Cappuccino>,
        espressoMachine: EspressoMachine) {
    orders.consumeEach {
        val groundBeans = grindCoffeeBeans(it.beans())
        val espresso = espressoMachine.pullEspressoShot(groundBeans)
        val steamedMilk = espressoMachine.steamMilk(it.milk())
        val cappuccino = makeCappuccino(it, espresso, steamedMilk)
        log("serve: $cappuccino")
    }
}
```