https://jagstalk.page.link/examples

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

```
fun main() = runBlocking {
    val orders = listOf(...)
    val ordersFlow: Flow<Menu.Cappuccino> = orders.asFlow()

    val espressoMachine = EspressoMachine(this)
    val time = measureTimeMillis {
        ...
    }
    log("time: $time ms")
    espressoMachine.destroy()
}
```

https://jagstalk.page.link/examples

```
fun main() = runBlocking {
    val orders = listOf(...)
    val ordersFlow: Flow<Menu.Cappuccino> = orders.asFlow()

    val espressoMachine = EspressoMachine(this)
    val time = measureTimeMillis {
        ...
    }
    log("time: $time ms")
    espressoMachine.destroy()
}
```

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

https://jagstalk.page.link/examples

```
fun main() = runBlocking {
    val orders = listOf(...)
    val ordersFlow: Flow<Menu.Cappuccino> = orders.asFlow()

    val espressoMachine = EspressoMachine(this)
    val time = measureTimeMillis {
        ...
    }
    log("time: $time ms")
    espressoMachine.destroy()
}
```

```
private suspend fun processOrders(orders: Flow<Menu.Cappuccino>,
        espressoMachine: EspressoMachine) {
    orders.consumeEach {
        val groundBeans = grindCoffeeBeans(it.beans())
        coroutineScope {
            val espressoDeferred = async {
                espressoMachine.pullEspressoShot(groundBeans)
            }
            val steamedMilkDeferred = async {
                espressoMachine.steamMilk(it.milk())
            }
            val cappuccino = makeCappuccino(it, espressoDeferred.await(),
                    steamedMilkDeferred.await())
            log("serve: $cappuccino")
        }
    }
}
```