


```
fun main() = runBlocking {  
    val orders = listOf(...)  
  
    val time = measureTimeMillis {  
        coroutineScope {  
            launch(CoroutineName("barista-1")) { processOrders(ordersChannel) }  
            launch(CoroutineName("barista-2")) { processOrders(ordersChannel) }  
        }  
    }  
    log("time: $time ms")  
}
```

```
val ordersChannel = produce(CoroutineName("cashier")) {  
    orders.forEach { send(it) }  
}
```


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

<https://jagstakk.page.link/examples>





<https://jagstalk.page.link/examples>



```
fun main() = runBlocking {
    val orders = listOf(...)
    val ordersChannel = produce(CoroutineName("cashier")) {
        orders.forEach { send(it) }
    }

    val time = measureTimeMillis {
        coroutineScope {
            launch(CoroutineName("barista-1")) { processOrders(ordersChannel) }
            launch(CoroutineName("barista-2")) { processOrders(ordersChannel) }
        }
    }
    log("time: $time ms")
}

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



```
fun main() = runBlocking {
    val orders = listOf(...)
    val ordersChannel = produce(CoroutineName("cashier")) {
        orders.forEach { send(it) }
    }

    val time = measureTimeMillis {
        coroutineScope {
            launch(CoroutineName("barista-1")) { processOrders(ordersChannel) }
            launch(CoroutineName("barista-2")) { processOrders(ordersChannel) }
        }
    }
    log("time: $time ms")
}

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

