



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

```
fun main() = runBlocking {  
    val orders = listOf(...)  
    val ordersFlow: Flow<Menu.Cappuccino> = orders.asFlow()  
        .produceIn(this)  
        .asFlow()  
  
    val espressoMachine = EspressoMachine(this)  
    val time = measureTimeMillis {  
        flowOf(  
            processOrders(ordersFlow, espressoMachine),  
            processOrders(ordersFlow, espressoMachine)  
        )  
        .flattenMerge()  
        .collect { cappuccino ->  
            log("serve: $cappuccino")  
        }  
    }  
    log("time: $time ms")  
    espressoMachine.destroy()  
}
```

```
private fun <T> ReceiveChannel<T>.asFlow(): Flow<T> = flow {  
    consumeEach { value ->  
        emit(value)  
    }  
}
```





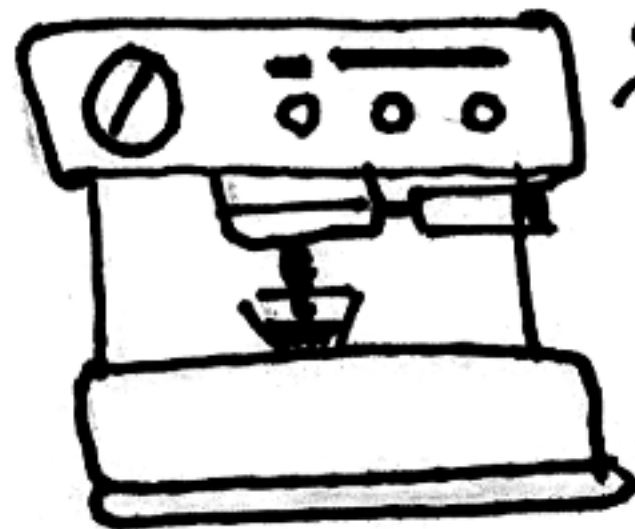
Menu

Order Accepted



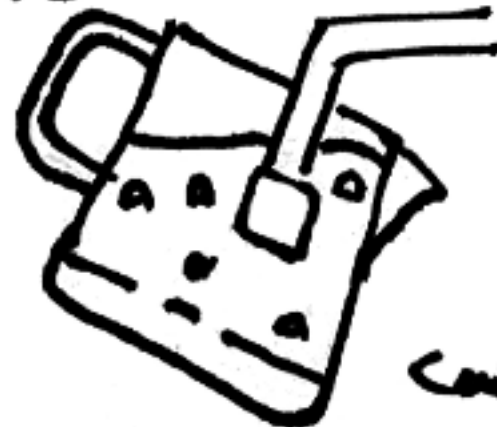
Grind Coffee Beans  
⌚ 20 sec

Espresso



Espresso  
⌚ 20 sec

Milk for Cappuccino

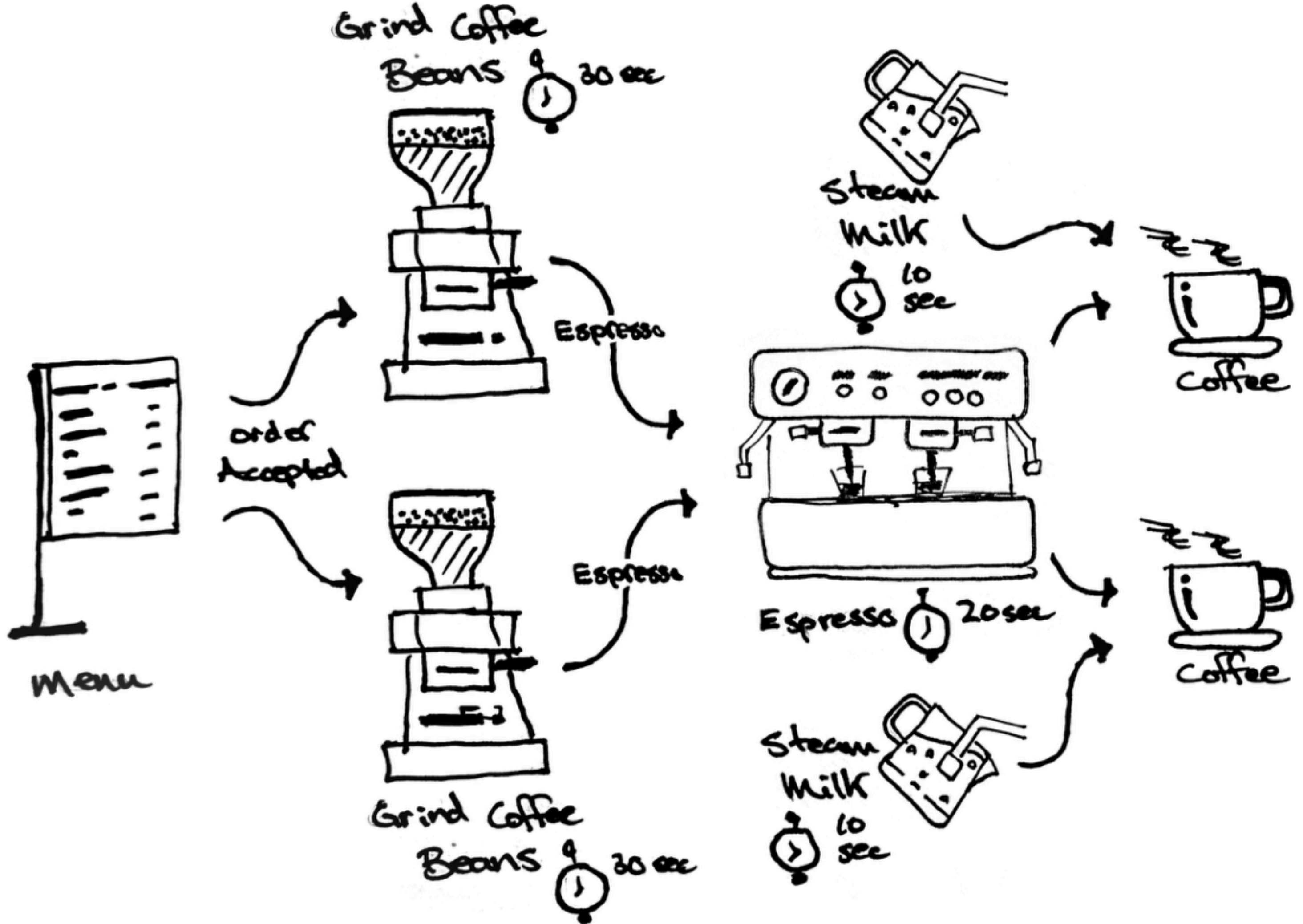


Steam Milk  
⌚ 10 sec

serve Cappuccino



Coffee















```
val sproMachines = ExprMachines(this)
```

fun main() = runBlocking {

provided in (this)

pressOrders (pressOrdersFlw, pressOrdersMachine);

pressOrders (pressesFlows pressMachine)



valtine = neasurineMiiis {

```
private fun <T> ReceiveChannel<T>.asFlow(): Flow<T> = flow {
```

expressMachine.destroy()

$$\text{val orders} \equiv \text{listOf}(\dots)$$

log('time:\$time ms')

log('serve:\$c\$' in c)

enit(vaire)

consists of each {value -->





• flatenMerge()

.cd1lect{capviccin ->

as follows:

```
val ordersFlow: Flow<Menu.Cappuccino> = orders.asFlow()
```