Universitetet i Oslo
Institutt for Informatikk
Research group for Reliable Systems (PSY)
Andrea Pferscher

# INF 5170: Models of Concurrency

Fall 2025 — **Group Session 5** — 17.10.2025

### Topic: Actors and Message Passing

**Exercise 1** Implement an actor for a calculator that has two states (`DUAL` and `SNGL`). In state `DUAL`, it accepts messages of the following form.

- Message (`from`, `ADD`, `n`, `m`) should send back `n+m` to the sending actor `from`
- Message (`from`, `SNGL`) should switch to state `SNGL`

In state `SNGL`, it accepts messages of the following form.

- Message (`from`, `STORE`, `n`) should store value `n` in the memory of the actor
- Message (`from`, `INC`, `n`) should send back the sum of `n` and the stored value to the sending actor `from`
- Message (`from`, `DUAL`) should switch to state `DUAL`

**Exercise 2** Implement an actor process for calculators as above in Go. You find the code skeleton in Listing 1.

**Exercise 3** We discussed how to encode single-read futures with channels in the lecture.

- Describe in words how to encode multi-read futures with channels, in particular how to type the channel and what effect your encoding has on termiation.
- Use your encoding in Go and write a small program that uses goroutine that takes two numbers, computes their sum and sends back the results using a multi-read future (according to your encoding).

You can use the skeleton in Listing 2.

Listing 1: A skeleton for an actor process for calculators in Go

```
package main

import "fmt"

type OP int

const (
        ADD   OP = 0
        INC   OP = 1
        STORE OP = 2
        DUAL  OP = 3
        SNGL  OP = 4
)

type Msg struct {
        op  OP
        p1  int
        p2  int
        ret  chan int
}

type State struct {
        /* define  state  here*/
}

func loop1(ch chan Msg, state State) {
        /* add code here*/
}

/* add functions here */

func main() {
        // simple test case ,write more
        input := make(chan Msg)
        go loop1(input, /* your starting  state */)
        res := make(chan int)
        input <- Msg{STORE, 2, 0, res}
        input <- Msg{INC, 5, 0, res}
        fmt.Println(<-res) //should print 7
}
```

Listing 2: A skeleton for Exercise 3

```
package main

import "fmt"

func f(fut /* add type here*/, p1 int, p2 int) {
    /* add code here */
}

func main() {
    ch := make( /* add type here */ )
    go f(ch, 1, 2)

//should work for any number of reads, test  with 2
    fmt.Println(<-ch)
    fmt.Println(<-ch)
}
```