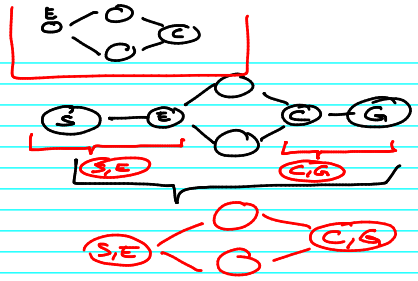
map (f) →

pipeline (seq(split), farm(F), seq(gather))

pipe (map(f), map(g)) →

map (pipeline (f, g))



DOUBLE / TRIPLE BUFFERING

→ COMM. OVERHEADS ↓ ∅

Stage i

```
while (not termination) {
    task = receive();
    res = compute(task);
    send(result);
}
```

Latency = $T_{comm} + T_{task} + T_{comm}$

overhead

useful computations

$\varepsilon = \frac{T_{task}}{2T_{comm} + T_{task}}$

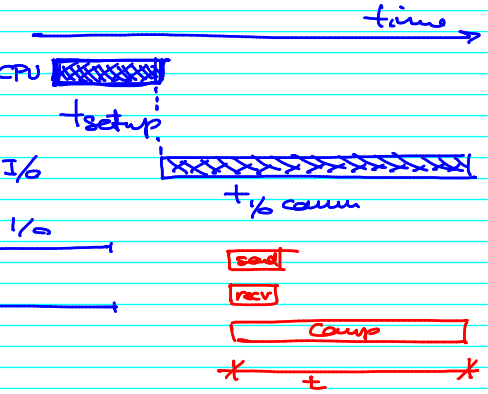
TRIPLE BUFF

act$_1$	receive	comp	send	recv
act$_2$	comp	send	recv	comp
act$_3$	send	receive	comp	send

t, t, t, time

$L \cong T_{task}$

$\varepsilon \cong 1$



IMPLEMENTING BUFFERING

(1) ASYNCHRONOUS I/O

SYNCHR write
↓ after write end

ASYNCHR I/O handle = pop(..);
⋮ other things
test(handle)
↳ check completion (type of)

TRIPLE BUFF

```
while(...){
SETUP | start send(res_{i-2})   } asynch I/O
      | start receive(task_{i+2})
      compute(task_i)
      wait(completed(send)
           && completed(recv))
}
```

(2) MULTITHREADING

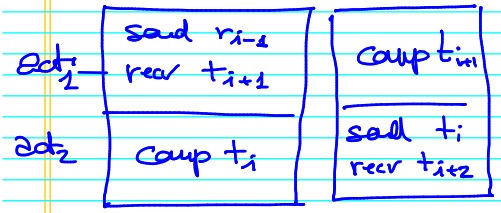
3 pthread_create(...)
start threads

body of threads :

condition(thread_id, please_id)
↳ case "x" : send
case "y" : receive
case "z" : compute

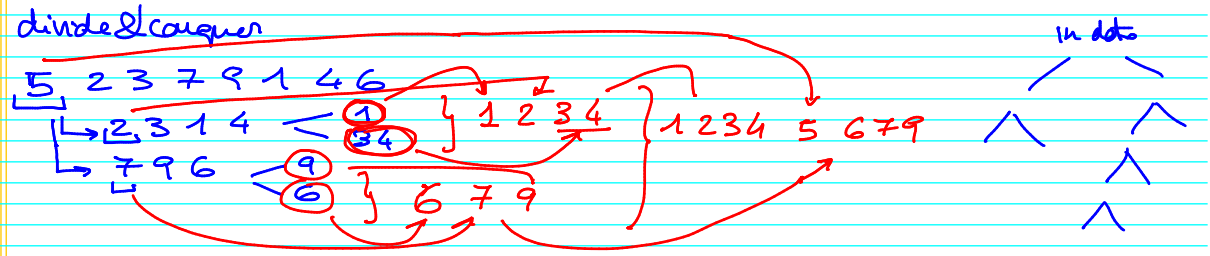
DOUBLE BUFFERING

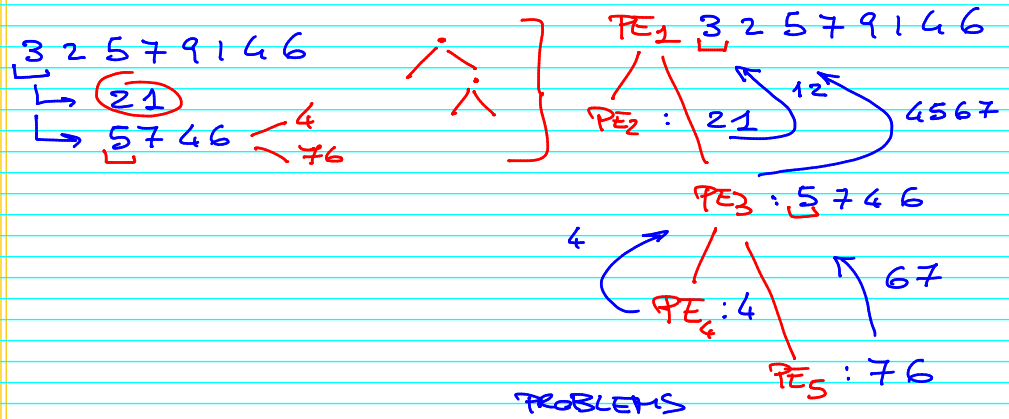
↳ reduced version of triple buffering



FARM WITH FEEDBACK

divide&conquer

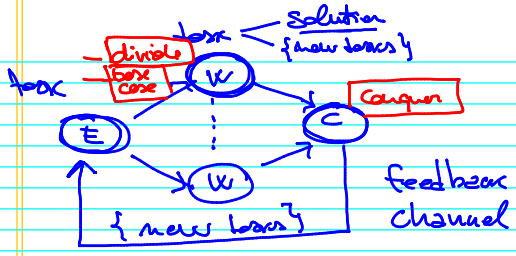


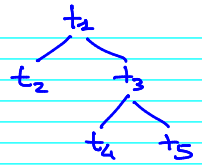


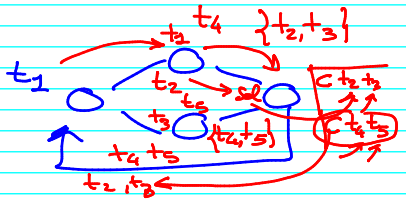
PROBLEMS

1) RESOURCE RECRUITMENT
2) MANAGE FREE RESOURCE

DYNAMIC RESOURCE MGT

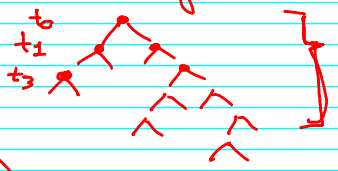




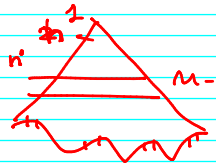



```
divConq bc bcs d split c x =
    if (bc x)          (bcs x)
    else (conquer
            map (dlc ... (split x)) ;
```

Performance

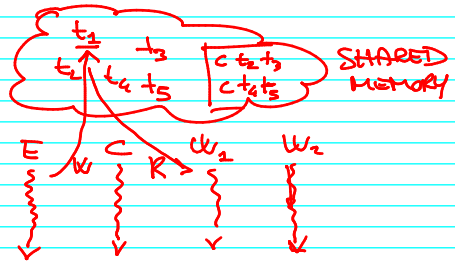


there is a max parallelism degree that depends on the computation AND on the input data



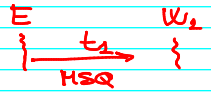
FASTFLOW

DdC (FARM WITH FEEDBACK) ON A MULTICORE?



Message queues between "concurrent activities" (threads)

① only contain pointers
② guarantee synchronization



$E - W_1$ [1]
$E - W_2$ [1]
$W_1 + W_2 - C$ [1]
$C - E$ [1]

pipe writes & reads are guaranteed to be atomic iff dim(data) < CONSTANT (512 byte at most)

E W_1

```
retcode = pipe (fd);
pthread_create < th1
               < th2
th1                th2
write (fd[1], ....)   read (fd[0], ...)
```

POSIX

retcode = pipe(int fd[])

syscall