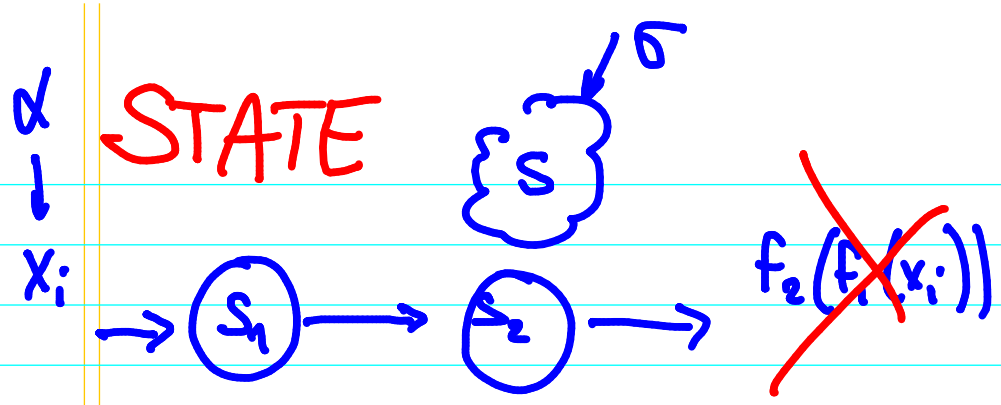
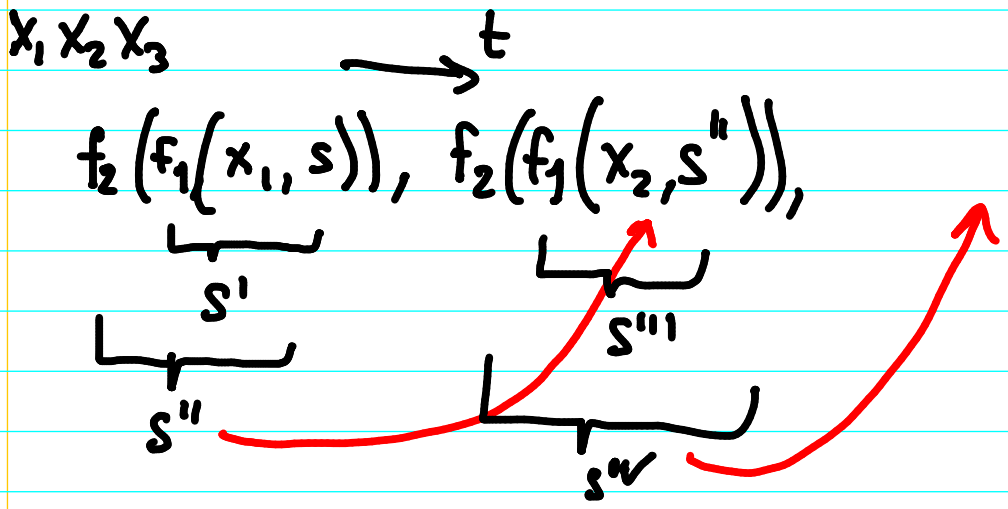
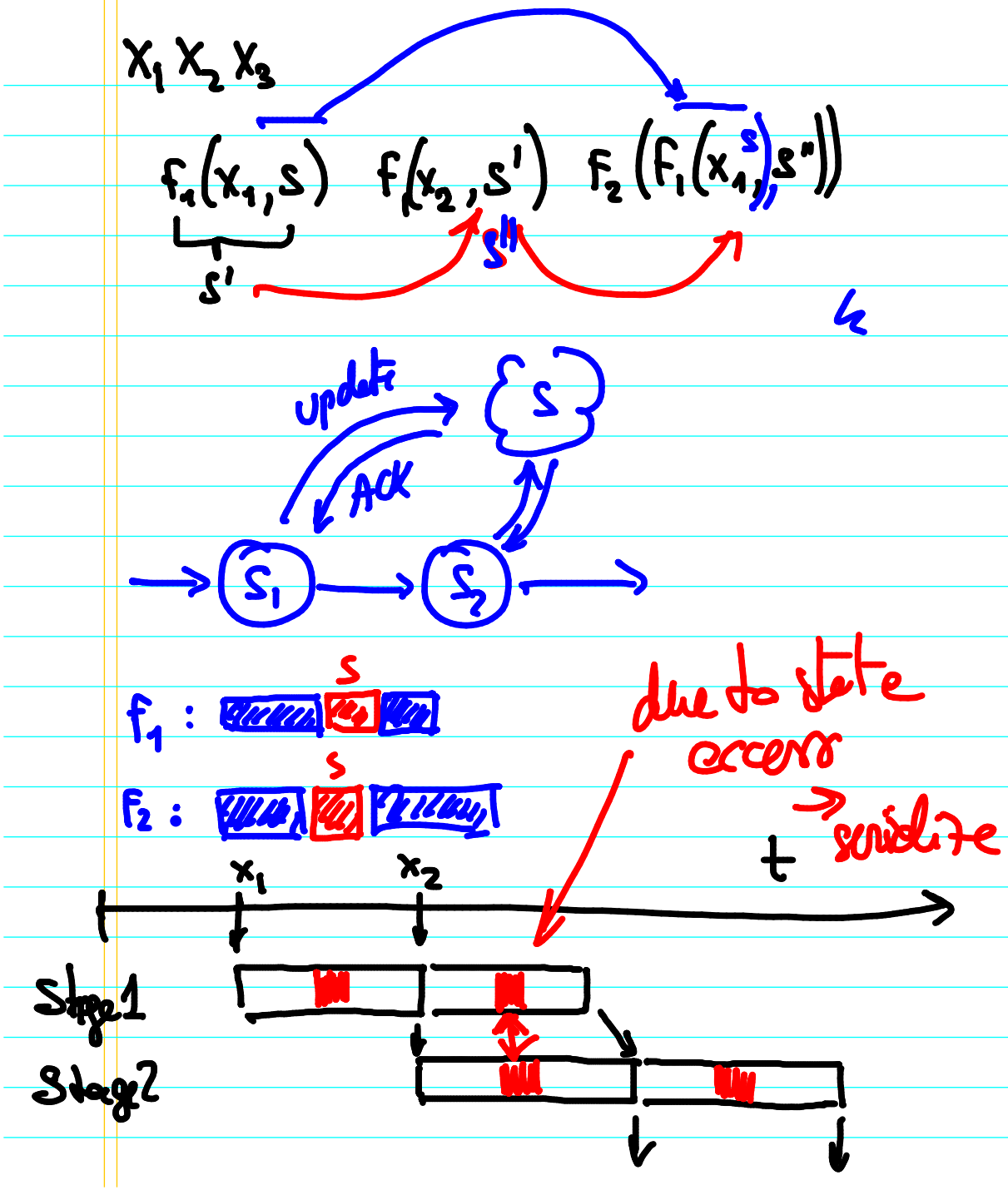


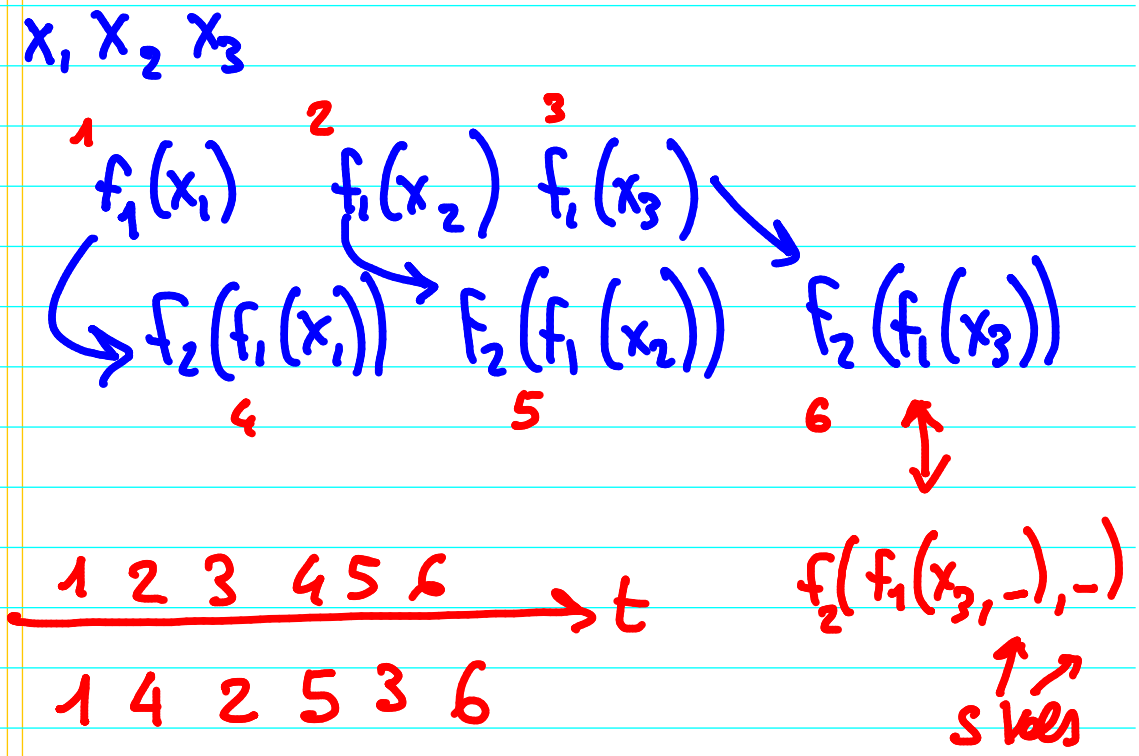
f_1: f_2:

$f_1 : \alpha \times \sigma \to \beta \times \sigma$

$f_1\ x, s \to x', s'$




$x_1\ x_2\ x_3$
$f_1(x_1, s)$
$f(x_2, s')$
$f_2(f_1(x_1, s), s'')$
s'
s''
update
ACK
S
S_1
S_2
f_1 :
f_2 :
s
s
due to state access
serialize
t
x_1
x_2
Stage1
Stage2

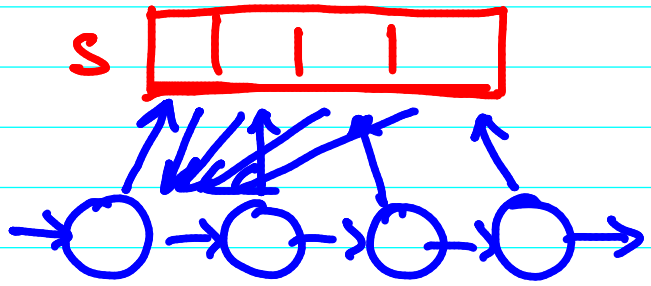

$x_1\ x_2\ x_3$
1
2
3
$f_1(x_1)$
$f_1(x_2)$
$f_1(x_3)$
$f_2(f_1(x_1))$
$f_2(f_1(x_2))$
$f_2(f_1(x_3))$
4
5
6
1 2 3 4 5 6
t
1 4 2 5 3 6
$f_2(f_1(x_3,-),-)$
skles

STRUCTURED STATE ACCESS

Recognize patterns in state access

1) readonly (e.g. scene & lights in a ray tracer)

2) "owner writes"



3) accumulate

associative commutative

$\forall$ process/thread : $s = s \oplus k$

4) "resource"

generic state variable

access through lock/mutex

ABSTRACT level

por comp patterns (pipe maps ...)

× ← CART PRODUCT

state access patterns

IMPLEMENTATION level

use diff implementations

∀ different access pattern

1) READONLY

COPYING !?

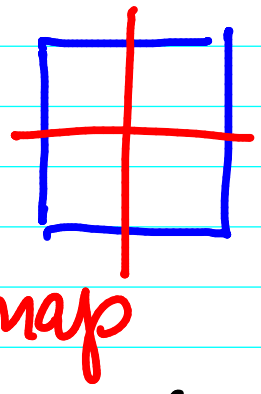
(No synchronization)

(LOCK / MUTEX)

2) OWNER WRITES

COPYING WITH NOTIFICATION

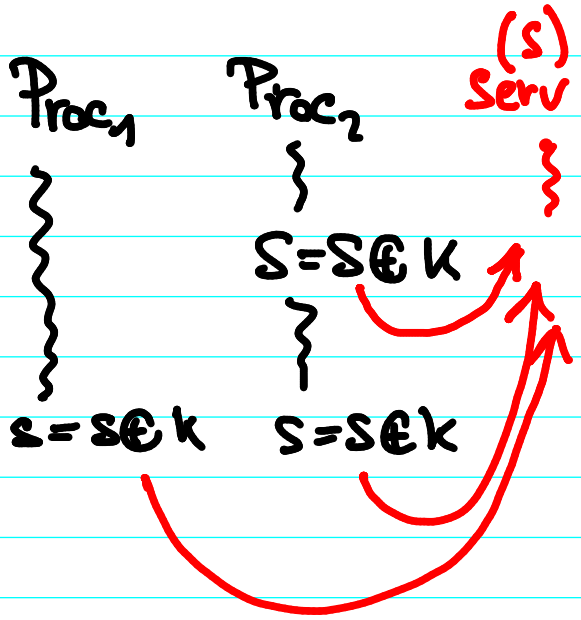
RAY TRACER → PIXEL SCREEN



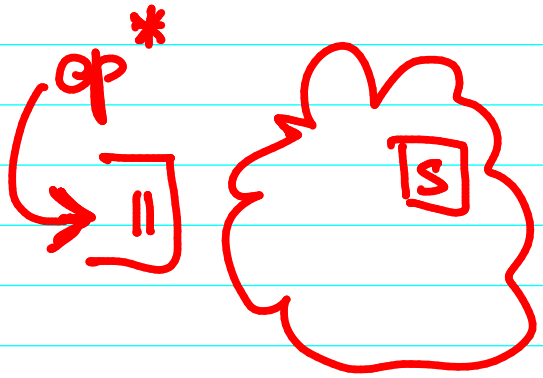
COPYING (if ∀ PROC/THREAD reads only its part.

3) Accumulate

Server + msg from proc/THREAD



4) "resource" "Server approach"



PARELLEL PROGR ⟶ IS A "MUST"

A "LIMITED" NUMBER
OF USEFUL PARALLEL
PATTERNS

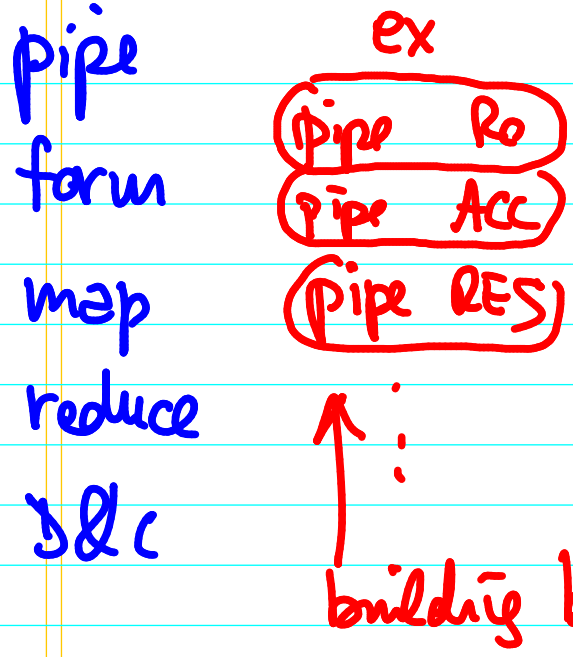
↙ ↘

COMPUTATIONS STATE ACCESS

└──────────┘

?

SET OF COMPOSABLE
BUILDING BLOCKS
SUPPORTING
PAR PROGRAMMING !



2nd choice → dynamic pairing

PSL farm f1 in(...) out(...)
worker in( ) out( )
end farm

farm f1 in(...) out(...) with(...)
↑ LIST of STATE

```
farm f1 in(int x) out(float y)
        with (int n)
   w in(x,n) out(y,n)
end farm
```

← would like to declare the kind of access

```
seq w in(int x, int n) out(float y, int n)
$c{
    y = f(x);
    n++;
}c$
end seq
```

→ within user code

```
type 'a state =
    RO of 'a | OW of 'a |
    Acc of 'a | RES of 'a ;;

pipe f1 f2 x = f2 (f1 (x));;
```

⇓

```
pipe f1 f2 x s:state = .....
```