

OS-III

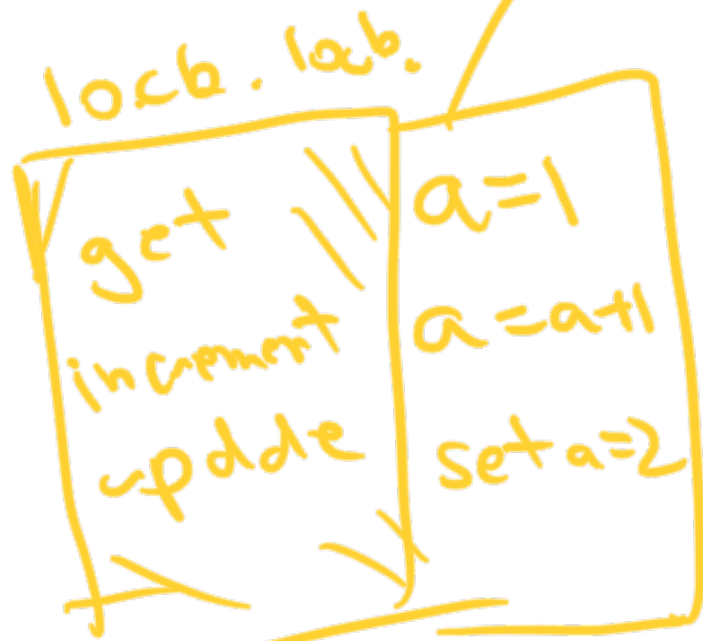
Thread Synchronisation

- ① Producer - Consumer problem
 - Semaphores
 - Base + mutex

- ② Concurrent DS
 - Atomic Integer
 - AS atomic integer

- Concurrent Hash maps

a = 1
a = a + 1
set a = 2



lock. unlock()

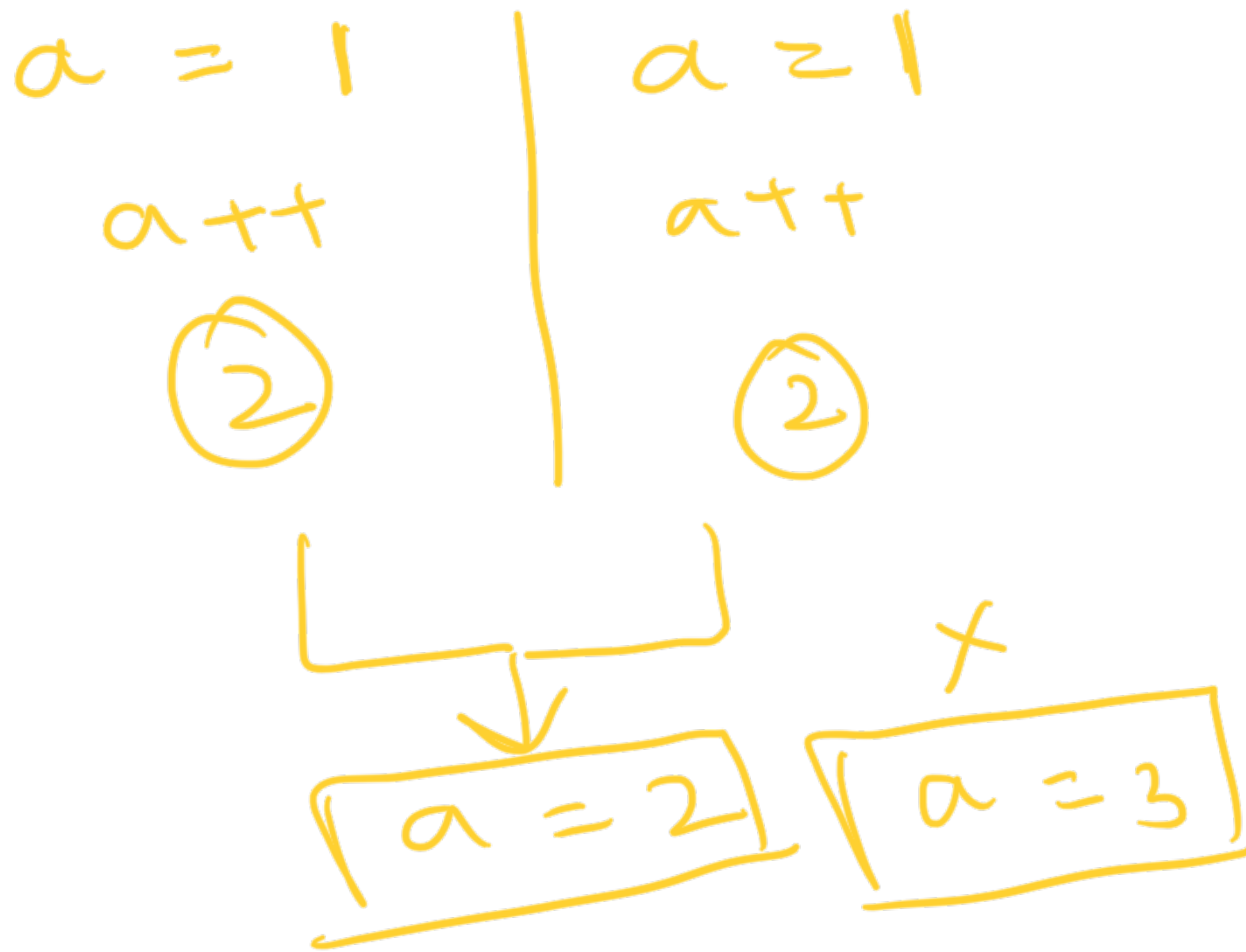


blocked

T2

- get
- increment
- update

2 + 1
T3



Thread synchronisation

- lock - mutex
- critical section

- Only one thread can access the critical section

T1

T2

T3

} waiting



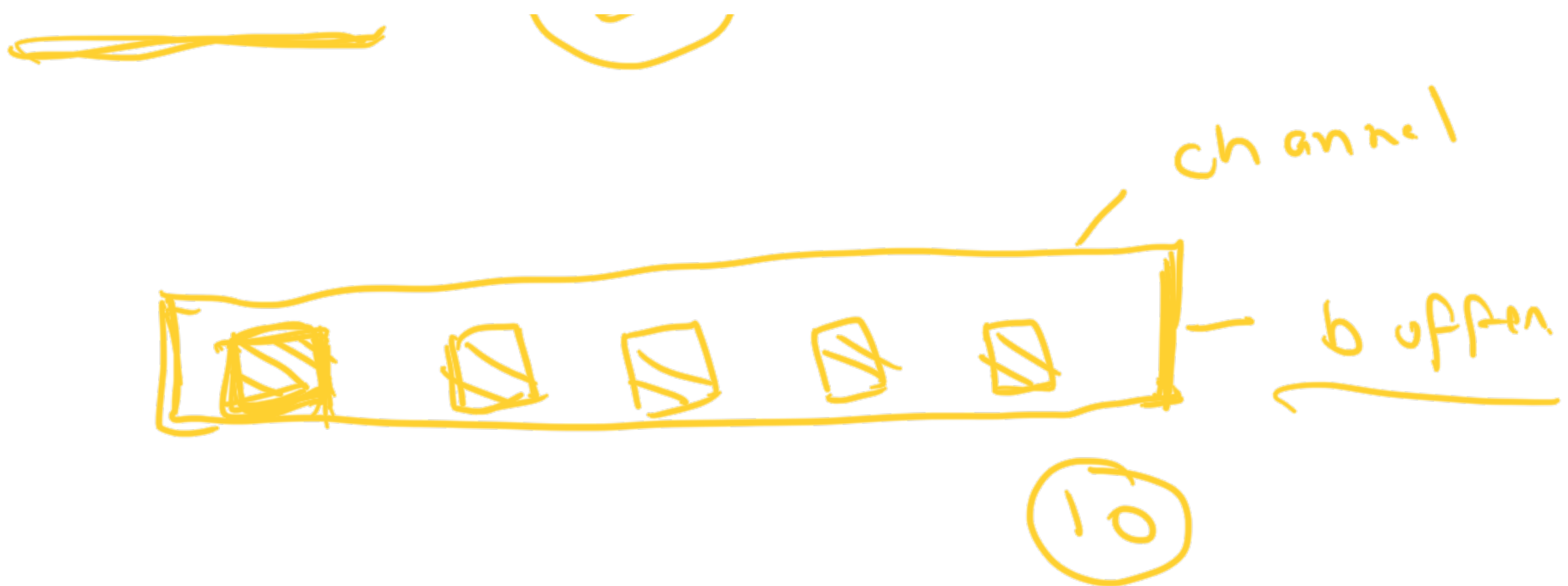
Producer - Consumer

Dijkstra

Producers =



unit of work
T starts, ip-d.



Consumer →

can only pick up items
if they are produced

- Bounded-buffer problem



Consumer
if cur_size > 0:
get item
current_size - 1

Producer
if (cur_size
 < buffer)
 Create item
cur_size + 1





if $cur_size > 3$

True

new writer
add to the queue

$2 < 3$
 P2
 P1

P1
 $cur = 2$
 if $2 < 3$

P2
 $cur = 2$
 push
 $cur = 3 + 1$

$$\text{curr} = 2 + 1$$

2nd

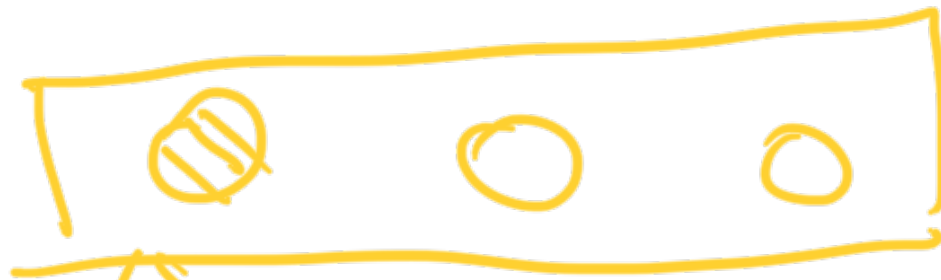


p1
curr size = 2
if 2 < 3
push

p2
curr size = 2
if 2 < 3
push

$$2 + 2 = 4$$

Con sum er



C1 (1)

if cur_size > 0 ⇒ T

C1.remove()

C2 (1)

if cur_size > 0 ⇒ T

C2.remove()

True

mutex



Kof ba
 Rabbit ma

Task 0



P



```

• lock(lockC)
  while (size < MAX) {

```

```

lock(lockC)
if (size > 0)

```

if (...)

Add to the ...

}

lock, unlock()

...

Range

lock, unlock()

→ Synchronization problem ✓

→ Sequential

Mutex in Java

— Synchronized

== Lock - Reentrant lock

① PC problem

② M steps

5:56 - 6:01

10:26 - 10:31

← 11:50

PC problem \rightarrow 20 | exceptions

\rightarrow Sync. issues

\rightarrow Thread sync

\rightarrow Mutex - synchronized

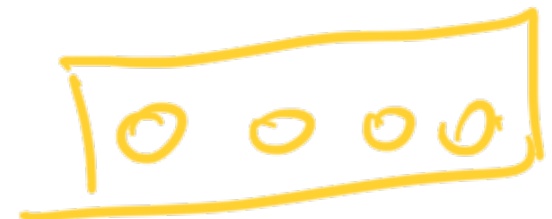
\rightarrow Sequential process

Ideal



1. Process \rightarrow Max size

\rightarrow 0



Dynamic - on the basis

20. max size

of size n no. of free slots
producers should change

② Consumers $\rightarrow 0 - 20$
 \rightarrow # of filled slots

$P \Rightarrow$ # of empty slots
 $C \Rightarrow$ # of filled slots

①

What do we need?

Multithreaded extend the CS

Semaphores

→ ways to synchronize multiple threads

Semaphore (10)

No ummy threads
Semaphore controls

Semaphore (1)

= mutex

→ Semaphore.acquire()

→ Semaphore.release()

Semaphore (0)

→ 3 producers can be active at the same time





if produce λ
 \rightarrow signal to the μ



→ signal to consumer

→ send a signal to producer.

P [Semaphores]

while (True) {

- ① acquire semaphore
- ② create unit & add to v
- ③ signal consumer.

C [Semaphores]

while (True)

- ① acquire
- ② consume
- ③ send a signal to producer.

Semaphore 20

Semaphore (0)

① Semaphore (19) $0 \rightarrow 1$

\hookrightarrow Semaphore Consum. (1)

	<u>S_p</u>	<u>S_e</u>	
0	20	0	
1	19	1	

\rightarrow decrease own count

\rightarrow increase consume count

How many instances can run at a moment?

$S_p(20) - 20$ more threads

Can be created